Tesis doctoral

MODELOS PREDICTIVOS BASADOS EN REDES NEURONALES RECURRENTES DE TIEMPO DISCRETO

Juan Antonio Pérez Ortiz

Julio de 2002

Memoria presentada para optar al grado de doctor

Dirigida por Mikel L. Forcada y Jorge Calera Rubio

Universidad de Alicante
Departamento de Lenguajes y Sistemas Informáticos

Tesis doctoral

MODELOS PREDICTIVOS BASADOS EN REDES NEURONALES RECURRENTES DE TIEMPO DISCRETO

Juan Antonio Pérez Ortiz

Julio de 2002

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante

MODELOS PREDICTIVOS BASADOS EN REDES NEURONALES RECURRENTES DE TIEMPO DISCRETO

Juan Antonio Pérez Ortiz

Resumen

Este trabajo estudia la aplicación de distintos modelos de redes neuronales recurrentes de tiempo discreto a diversas tareas de carácter predictivo.

Las redes neuronales recurrentes son redes neuronales que presentan uno o más ciclos en el grafo definido por las interconexiones de sus unidades de procesamiento. La existencia de estos ciclos les permite trabajar de forma innata con secuencias temporales. Las redes recurrentes son sistemas dinámicos no lineales capaces de descubrir regularidades temporales en las secuencias procesadas y pueden aplicarse, por lo tanto, a multitud de tareas de procesamiento de este tipo de secuencias. Esta tesis se centra en la aplicación de las redes neuronales recurrentes a la predicción del siguiente elemento de secuencias de naturaleza simbólica o numérica.

No obstante, la predicción en sí no es el objetivo último: en esta tesis la capacidad predictiva de las redes recurrentes se aplica a la compresión de señales de voz o de secuencias de texto, a la inferencia de lenguajes regulares o sensibles al contexto, y a la desambiguación de las palabras homógrafas de una oración.

Los modelos concretos de redes utilizados son, principalmente, la red recurrente simple, la red parcialmente recurrente y el modelo neuronal de memoria a corto y largo plazo; este último permite superar el llamado problema del gradiente evanescente que aparece cuando los intervalos de tiempo mínimos entre eventos interdependientes son relativamente largos. Para determinar valores correctos de los parámetros libres de las redes se usan dos algoritmos, el clásico algoritmo del descenso por el gradiente y una forma del filtro de Kalman extendido.

Palabras clave: redes neuronales recurrentes de tiempo discreto, modelos predictivos en línea y fuera de línea, aprendizaje de secuencias simbólicas, predicción de señales de voz, etiquetado de categorías léxicas.

PREFACIO

Las redes neuronales recurrentes son uno de los modelos posibles para tratar secuencias temporales. Su principal ventaja estriba en la posibilidad de almacenar una representación de la historia reciente de la secuencia, lo que permite, a diferencia de lo que ocurre con las redes neuronales no recurrentes, que la salida ante un determinado vector de entrada pueda variar en función de la configuración interna actual de la red.

Como se verá a lo largo de esta tesis, las redes recurrentes han sido muy utilizadas en multitud de tareas relacionadas con el procesamiento de secuencias temporales. Dentro de estas tareas, la predicción de secuencias, en la que se estima el valor futuro de uno o más elementos de la secuencia a partir de la historia observada, tiene importantes aplicaciones en campos tales como la inferencia de lenguajes o la compresión de señales. Esta tesis se centra en este aspecto del procesamiento de secuencias discretas con redes recurrentes de tiempo discreto y estudia tareas que van desde la predicción en línea de secuencias simbólicas o de voz hasta la desambiguación de las palabras homógrafas de una oración a partir de la predicción de sus categorías léxicas, pasando por la inferencia de lenguajes no regulares.

Este trabajo demuestra algunas limitaciones de los modelos neuronales recurrentes *tradicionales* y cómo algunas de ellas pueden ser superadas mediante modelos o algoritmos de entrenamiento más elaborados.

La tesis ha sido posible gracias a las ideas y aportaciones continuas durante más de tres años de los Drs. Mikel L. Forcada y Jorge Calera Rubio del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante. No obstante, la parte de la tesis relacionada con el modelo denominado *memoria a corto y largo plazo* es fruto de mi estancia de dos meses en 2000 en el Istituto Dalle Molle di Studi sull'Inteligenza Artificiale (IDSIA) de Lugano (Suiza) bajo la supervisión del Dr. Jürgen Schmidhuber.

Estructura de la tesis

Se ofrece a continuación un pequeño resumen de lo que se discutirá en cada uno de los capítulos. Los primeros tienen carácter introductorio:

Capítulo 1. Este capítulo presenta los problemas que se estudian en esta tesis.

Capítulo 2. Se introduce el modelo de red neuronal recurrente de tiempo discreto y las principales tareas a las que se ha aplicado dentro del campo del procesamiento de secuencias.

Capítulo 3. Un repaso a los modelos neuronales que se utilizan a lo largo de la tesis, tanto los que se pueden considerar "clásicos", como algunos más recientes (en concreto, dentro de este último grupo, la red neuronal recurrente en cascada y la red basada en las memorias a corto y largo plazo).

Capítulo 4. En este capítulo se introducen los principales algoritmos de entrenamiento supervisado para redes recurrentes, especialmente el descenso por el gradiente y el filtro de Kalman extendido desacoplado. Con todo, las redes neuronales recurrentes presentan algunos problemas fundamentales que son analizados también.

Capítulo 5. Una vez introducidos los problemas estudiados en la tesis, así como los aspectos fundamentales de las redes neuronales recurrentes, se presentan los enfoques con los que previamente han sido abordados estos problemas, haciendo especial énfasis en las soluciones de carácter neuronal.

Los capítulos anteriores son básicamente un repaso al estado de la cuestión, además de presentar una formalización de los conceptos introducidos. Los capítulos que se describen a continuación constituyen la aportación original de esta tesis:

Capítulo 6. El objetivo es estudiar el uso de las redes recurrentes para predecir el siguiente símbolo de una secuencia de texto. Se hace énfasis en la predicción en línea, un problema mucho más difícil que la inferencia gramatical con redes neuronales clásica. El modelo de probabilidad desarrollado por la red se utiliza en un compresor aritmético, de forma que la razón de compresión se usa como medida de la calidad del predictor. A modo de estudio preliminar, se analiza también la predicción sobre secuencias derivadas de automatas finitos deterministas o sobre secuencias caóticas.

Capítulo 7. El modelo de memorias a corto y largo plazo (LSTM) entrenado mediante el descenso por el gradiente es capaz de resolver problemas muy difíciles de resolver con las redes tradicionales. Aquí se aplica por primera vez a este modelo un algoritmo de entrenamiento basado en el filtro de Kalman extendido y se observa cómo los resultados mejoran ostensiblemente, en cuanto a velocidad y robustez, los del algoritmo original en un par de tareas de inferencia de lenguajes.

Capítulo 8. En este capítulo se explora el uso de las redes recurrentes para la desambiguación léxica de corpus textuales, basándose en la predicción de la categoría léxica de la palabra siguiente a las ya vistas de una oración. El enfoque presentado aquí no necesita ningún texto desambiguado manualmente, lo que lo convierte probablemente en el primer método neuronal que posee esta cualidad. Los experimentos demuestran que el rendimiento de este enfoque es, como mínimo, similar al de un modelo oculto de Markov estándar entrenado mediante el algoritmo de Baum y Welch.

Capítulo 9. Aquí se presenta un estudio comparativo del rendimiento de las redes cuando son entrenadas en línea para predecir la próxima muestra de una señal de voz digitalizada. La comparación se realiza principalmente con modelos lineales y con una red recurrente en cascada que fue propuesta en su día para realizar esta tarea. Los resultados confirman los de otros trabajos que encontraron serias limitaciones a la hora de trabajar con series numéricas, especialmente al usar un algoritmo de aprendizaje basado en el descenso por el gradiente. El filtro de Kalman aplicado al entrenamiento de estas redes, por otro lado, supera parcialmente algunas de estas limitaciones.

Capítulo 10. Este capítulo recoge las principales conclusiones que se deducen de todo lo anterior, además de presentar una lista detallada de posibles trabajos de investigación para el futuro.

Apéndice A Este apéndice muestra cómo entrenar un modelo oculto de Markov para realizar la desambiguación de las categorías morfológicas de una oración, técnica que se utiliza en los experimentos del capítulo 8.

Publicaciones

Esta tesis doctoral recoge algunos trabajos publicados en congresos o revistas internacionales:

- Juan Antonio Pérez-Ortiz, Jorge Calera-Rubio y Mikel L. Forcada, 2001. “Online text prediction with recurrent neural networks”, *Neural Processing Letters* **14**(2), 127–140.

- Juan Antonio Pérez-Ortiz y Mikel L. Forcada, 2001. “Part-of-speech tagging with recurrent neural networks”, en *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2001*, 1588–1592.

- Juan Antonio Pérez-Ortiz, Jorge Calera-Rubio y Mikel L. Forcada, 2001. “A comparison between recurrent neural architectures for real-time nonlinear prediction of speech signals”, en D. J. Miller, T. Adali, J. Larsen, M. Van Hulle y S. Douglas, coordinadores, *Neural Networks for Signal Processing XI, Proceedings of the 2001 IEEE Neural Networks for Signal Processing Workshop, NNSP 2001*, 73–81, IEEE Signal Processing Society.

- Juan Antonio Pérez-Ortiz, Jorge Calera-Rubio y Mikel L. Forcada, 2001. “Online symbolic-sequence prediction with discrete-time recurrent neural networks”, en G. Dorffner, H. Bischof y K. Hornik, coordinadores, *Proceedings of the International Conference on Artificial Neural Networks, ICANN 2001*, vol. 2130 de *Lecture Notes in Computer Science*, 719–724, Springer-Verlag, Berlín.

- Felix A. Gers, Juan Antonio Pérez-Ortiz, Douglas Eck y Jürgen Schmidhuber, 2002. “DEKF–LSTM”, en M. Verleysen, coordinador, *Proceedings of the 10th European Symposium on Artificial Neural Networks, ESANN 2002*, 369–376, D-side Publications.

También son consecuencia de las investigaciones desarrolladas para la tesis los siguientes artículos todavía inéditos, aunque ya aceptados:

- Juan Antonio Pérez-Ortiz, Felix A. Gers, Douglas Eck y Jürgen Schmidhuber, 2002. “Kalman filters improve LSTM network performance in hard problems”, *Neural Networks*, aceptado con modificaciones.

- Felix A. Gers, Juan Antonio Pérez-Ortiz, Douglas Eck y Jürgen Schmidhuber, agosto de 2002. “Learning context sensitive languages with LSTM trained with Kalman filters”, en *Proceedings of the International Conference on Artificial Neural Networks, ICANN 2002, Lecture Notes in Computer Science*, Springer-Verlag, Berlín; aceptado.

- Juan Antonio Pérez-Ortiz, Jürgen Schmidhuber, Felix A. Gers y Douglas Eck, agosto de 2002. “Improving long-term online prediction with decoupled extended Kalman filters”, en *Proceedings of the International Conference on Artificial Neural Networks, ICANN 2002, Lecture Notes in Computer Science*, Springer-Verlag, Berlín; aceptado.

Agradecimientos Vaya ahora mi agradecimiento a todos los que han contribuido de forma directa al desarrollo de este trabajo: a mis directores de tesis, a los miembros del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante y a todos los que me acogieron en el IDSIA de Lugano. También han sido fundamentales para la elaboración de este trabajo la beca de formación de personal investigador FPI-99-14-268 de la Generalitat Valenciana y los proyectos de la Comisión Interministerial de Ciencia y Tecnología TIC97-0941 y TIC2000-1599-C02-02.

Finalmente, un sincero *¡gracias!* a todos los que habéis estado ahí mientras escribía esta tesis.

Juan Antonio Pérez Ortiz
Alicante, 5 de julio de 2002

ÍNDICE GENERAL

1. LOS PROBLEMAS

> Esta tesis aborda diversos aspectos relativos al uso de redes neuronales recurrentes de tiempo discreto en tareas de predicción de secuencias. En este capítulo se presenta los problemas estudiados y los motivos por lo que es importante abordarlos.

Las redes neuronales recurrentes (Kolen y Kremer 2001) (Haykin 1999, cap. 15) (Hertz et al. 1991, cap. 7) son, en principio, un modelo computacional más potente que las clásicas redes neuronales hacia adelante. Esta mayor potencia proviene del hecho de que las redes recurrentes son capaces de procesar secuencias temporales gracias a la posibilidad de recordar parcialmente la historia relevante de la secuencia por medio de una representación en forma de estado. Esta *memoria* no existe de manera natural en las redes no recurrentes, utilizadas principalmente para el procesamiento de datos estáticos.

A simple vista el rasgo que diferencia las redes neuronales recurrentes de las que no lo son es simple: la existencia de, como mínimo, una conexión cíclica (*recurrente*) entre las neuronas que las configuran. Esta diferencia tiene, sin embargo, profundas implicaciones en la capacidad de computación del modelo y en los algoritmos de entrenamiento necesarios para conseguirla.

A la hora de clasificar las redes neuronales recurrentes se suele considerar la forma en la que el tiempo transcurre durante la actividad de la red. En este sentido podemos distinguir entre *redes neuronales de tiempo discreto* y *redes neuronales de tiempo continuo*. En las redes neuronales de tiempo discreto los cálculos ocurren a saltos, como si un reloj marcara el ritmo de procesamiento y se asume que en cada paso de tiempo la salida de cada neurona se obtiene instantáneamente a partir de sus entradas. En el caso de las redes neuronales de tiempo continuo (Pearlmutter 1995), las entradas y salidas de la red son función de una variable temporal continua y la dinámica de sus neuronas viene descrita por una ecuación diferencial.

En este trabajo se estudia la capacidad de las redes recurrentes para predecir el siguiente componente de distintos tipos de secuencias. Ya que las secuencias son de tiempo discreto, las redes también tienen que serlo forzosamente. La predicción del siguiente elemento con redes recurentes es un tema clásico; véase, por ejemplo, los trabajos de Cleeremans et al. (1989) y Elman (1990). En este contexto es de esperar que la red desarrolle un modelo interno de la información relevante sobre una cierta cantidad de la historia pasada de la secuencia que le permita realizar predicciones acertadas sobre el futuro.

Dado que la tesis se centra en el procesamiento de secuencias temporales (Weigend y Gershenfeld 1994), es momento de introducir este concepto.

1.1. Secuencias

Informalmente, diremos que una *secuencia* es una serie de datos tomados de un conjunto S y que representaremos como

$$s[n] = s[1], s[2], \ldots, s[L_s]$$

donde L_s es la *longitud* de la secuencia. En una secuencia es posible que $s[t] = s[t']$ para $t \neq t'$. Los valores entre corchetes son *índices* (normalmente valores enteros) que reflejan el orden de los distintos datos dentro de la secuencia; cuando este índice se refiere al tiempo se dice que la secuencia es de tipo *temporal*.

Supongamos un sistema que trabaje con secuencias temporales. Podemos intentar caracterizar los tipos de operaciones que el sistema puede realizar sobre una secuencia (Forcada y Gori 2001). Para ello, consideremos que los elementos de la secuencia de entrada $u[t]$ pertenecen a un conjunto U y que la secuencia tiene longitud L_u, es decir, las secuencias de entrada son de la forma

$$u[t] \in U, \qquad t = 1, 2, \ldots, L_u \tag{1.1}$$

El sistema transforma la secuencia de entrada en una secuencia de salida, digamos $y[t]$. Consideremos que los elementos de la secuencia de salida pertenecen a un conjunto Y y que la longitud de la secuencia es L_y, esto es,

$$y[t] \in Y, \qquad t = 1, 2, \ldots, L_y \tag{1.2}$$

Según la forma de las variables anteriores, las operaciones que un sistema que trabaje con secuencias puede desarrollar pueden dividirse en cuatro grupos:

- *Traducción o filtrado de secuencias.* Este es el caso más general de procesamiento de secuencias. Una secuencia $u[1], u[2], \ldots, u[L_u]$ se transforma en otra secuencia $y[1], y[2], \ldots, y[L_y]$. Un caso interesante dentro de este tipo de procesamiento es la traducción *síncrona* en la que el sistema lee y genera una componente en cada paso, con lo que ambas secuencias crecen al mismo ritmo y $L_u = L_y$.

- *Clasificación de secuencias.* Aquí $L_y = 1$. El sistema asigna a toda la secuencia de entrada un único valor o etiqueta del conjunto Y.

- *Generación de secuencias.* Ahora $L_u = 1$. En este modo, el sistema genera una secuencia de salida para una única entrada.

- *Predicción de secuencias.* En este caso $U = Y$. El sistema lee una secuencia $u[1], u[2], \ldots, u[t]$ y produce como salida una posible continuación de la secuencia de entrada $\hat{u}[t+1], \hat{u}[t+2], \ldots$ Normalmente se predice un único valor futuro, generalmente $\hat{u}[t+1]$, aunque en algunos casos el valor predicho es $\hat{u}[t+k]$ para un valor pequeño de k.

Los problemas abordados en esta tesis se pueden incluir dentro del último grupo y se introducen a continuación.

1.2. Los problemas

Las tareas de predicción que estudia esta tesis se pueden clasificar desde varios ejes. Uno de ellos atiende al origen de los componentes de la secuencia; así, en el caso de predicción sobre señales numéricas, tendremos $U = \mathbb{R}$ y en el caso de predicción sobre series simbólicas $U = \Sigma$, donde Σ es un alfabeto cualquiera. Por otro lado, puede considerarse la aplicación concreta de las tareas de predicción estudiadas. Esta última clasificación es la que determina en gran medida la estructura de los contenidos de la tesis y considera fundamentalmente cuatro aplicaciones: la compresión de secuencias simbólicas, la inferencia de lenguajes regulares y sensibles al contexto, la desambiguación de las partes de una oración y la compresión de señales de voz digitalizadas.

1.2.1. Compresión de secuencias simbólicas

Las limitaciones impuestas por la capacidad de los dispositivos de almacenamiento y por el ancho de banda de los medios de transmisión obliga a comprimir la información. Las técnicas de compresión suelen ser distintas según la naturaleza simbólica o numérica de las secuencias a comprimir.

Dentro del primer grupo, los sistemas compresores (Bell et al. 1990; Nelson y Gailly 1995) suelen basarse en un modelo de probabilidad que determine las probabilidades de aparición de cada símbolo en un contexto secuencial determinado. De esta manera, la codificación elegida para un símbolo concreto está en función de las previsiones realizadas antes de su aparición por el modelo y se puede usar codificaciones más pequeñas para los símbolos más frecuentes, reduciendo así el número de bits necesarios para codificar la información.

Las redes recurrentes son candidatas para dicho modelo de probabilidad y en esta tesis se estudia la eficiencia de este enfoque, principalmente con secuencias de texto en lenguaje natural, aunque también se consideran otros tipos de secuencias simbólicas para poder matizar los resultados obtenidos. En esta tesis nos hemos planteado una restricción importante: que el modelo de probabilidad debe obtenerse *al vuelo* (lo que se llama habitualmente *en línea*), de forma que la codificación se realice *sobre la marcha* conforme se procesa la secuencia de entrada.

1.2.2. Inferencia de lenguajes con dependencias a largo plazo

La *inferencia de lenguajes* (o *inferencia gramatical*) consiste (Sakakibara 1997) en deducir un modelo (por ejemplo, un autómata finito o una gramática) de un lenguaje a partir de un conjunto de cadenas de ejemplo. Las aplicaciones de la inferencia de lenguajes se extienden por multitud de campos como el reconocimiento sintáctico de formas, la lingüística computacional o la bioinformática.

Las redes neuronales recurrentes de tiempo discreto han sido ampliamente utilizadas en tareas de inferencia de lenguajes (Forcada 2002). Se pueden distinguir dos métodos principales (Alquézar y Sanfeliu 1994) para abordar la tarea en función de si todas las cadenas de ejemplo pertenecen al lenguaje a inferir (llamadas muestras *positivas*) o si se incluyen también cadenas que no pertenecen a él (muestras *negativas*) convenientemente identificadas. En el primer caso se entrena la red recurrente para que prediga en cada paso el siguiente símbolo de cada una de las muestras positivas y se establece un umbral de predicción para intentar que la red acepte las cadenas correctas y rechace las incorrectas. El segundo caso se resuelve como una tarea de clasificación en la que el entrenamiento se lleva a cabo para discernir si una cadena pertenece o no al lenguaje tras haberla visto completamente.

En este trabajo se considera la modalidad predictiva para la inferencia de lenguajes regulares y sensibles al contexto que presentan *dependencias a largo plazo*, que, como se verá, suelen ser difíciles de manejar con las redes recurrentes tradicionales (Hochreiter et al. 2001). Por ello, se considerará también el modelo recurrente de memorias a corto y largo plazo (Hochreiter y Schmidhuber 1997), ideado para superar este problema.

1.2.3. Desambiguación categorial

La existencia de ambigüedades en las lenguas es uno de los principales escollos para conseguir sistemas de calidad para su procesamiento automático. La correspondiente *desambiguación* es un paso intermedio muy importante (Manning y Schütze 1999) en muchas aplicaciones como el reconocimiento del habla o la traducción automática.

En esta tesis se estudia el problema de la desambiguación de aquellas palabras a las que se puede asociar más de una categoría léxica (problema conocido también como *etiquetado de las partes de la oración*; en inglés *PoS tagging* por *part-of-speech tagging*).[1] Un *etiquetador morfosintáctico* es, en este contexto, un programa que asigna a cada palabra de un texto una categoría léxica de entre un conjunto previamente definido de categorías.

Las categorías léxicas (también llamadas *partes de la oración*) pueden ser muy amplias (como "verbo") o muy específicas (como "verbo transitivo, 3.ª persona del singular del presente de indicativo"), según la aplicación concreta que queramos darles. A la mayor parte de las palabras de cualquier oración se les puede asignar fácilmente una única etiqueta léxica mediante la consulta de un diccionario o *léxico*.[2] Sin embargo, como ya se ha dicho, hay también muchas palabras que son *ambiguas* en el sentido de que se les puede asignar más de una categoría léxica; por ejemplo, en español la palabra *bajo* puede ser un nombre —un instrumento que produce sonidos graves—, un adjetivo —de poca altura—, un adverbio —equivalente a *abajo*— o una preposición —*debajo de*.

La elección de la categoría léxica correcta puede ser crucial, por ejemplo, al traducir a otro idioma. La mayoría de los desambiguadores categoriales se basan en la suposición de que la categoría correcta puede determinarse a partir del contexto en el que aparece la palabra, o que, al menos, este contexto hace que una de ellas sea más probable. Normalmente, la decisión se toma en función de las categorías léxicas de las palabras vecinas, asumiendo que la sintaxis es la mejor aliada a la hora de desambiguar una palabra. Esta idea puede fallar, no obstante, en frases como en la clásica frase en inglés "Time flies like an arrow", donde, además de la interpretación principal

[1] Aunque la ambigüedad se puede presentar en distintos niveles, nos centraremos en el nivel léxico. Dentro de este nivel hay dos causas principales de ambigüedad: la *polisemia* y la *homografía*. Una palabra es *polisémica* cuando puede tener más de una interpretación, todas con la misma categoría léxica. Por otro lado, en las palabras *homógrafas*, que son las que aquí nos interesan, las distintas interpretaciones pertenecen a categorías léxicas diferentes.

[2] También puede usarse un *adivinador*, que deduce la categoría léxica de la palabra a partir de su forma superficial (por ejemplo, en español prácticamente todas las palabras acabadas en -*abais* son verbos).

(nombre–verbo–conjunción...), existen otras dos sintácticamente impecables, aunque semánticamente improbables (verbo–nombre–conjunción... y nombre–nombre–verbo...).[3]

Diremos que un texto está *completamente etiquetado* o *etiquetado sin ambigüedades* cuando cada aparición de una palabra lleva asignada la etiqueta léxica correcta. Por otro lado, un texto *parcialmente etiquetado* o *etiquetado ambiguamente* es aquel en el que se asocia (mediante un léxico o un adivinador) a cada palabra (independientemente del contexto en el que aparece) un conjunto de posibles categorías léxicas; en este caso, a las palabras ambiguas y desconocidas les corresponde más de una etiqueta léxica.[4]

La obtención de corpus significativos de textos completamente etiquetados es un problema para todas las lenguas, especialmente para las minoritarias. Por ello, es muy recomendable diseñar desambiguadores categoriales que no los necesiten.

Las palabras que comparten el mismo conjunto de categorías léxicas se dice que pertenecen a la misma *clase de ambigüedad* (Cutting et al. 1992); por ejemplo, las palabras *canto* y *libro* pertenecen a la misma clase de ambigüedad {nombre, verbo}.

En esta tesis se plantea de forma original la desambiguación categorial como un problema de predicción de clases de ambigüedad con redes recurrentes, como se verá más adelante. Además, el etiquetador morfosintáctico propuesto necesita solo corpus de textos parcialmente etiquetados.

1.2.4. Predicción de señales de voz

La predicción de señales de voz en tiempo real (Barnwell et al. 1996) es una parte muy importante de sistemas de comunicación actuales como los sistemas de telefonía móvil. Si suponemos que el valor de la señal en el instante t puede predecirse a partir de su valor en instantes anteriores, se puede conseguir reducir la tasa de bits (siempre en función de la calidad del predictor) codificando eficientemente la diferencia entre la señal real en el instante t y la señal predicha, como se observa en la figura 1.1.

Aunque se ha comprobado que los predictores de voz no lineales mejoran la capacidad predictiva de los lineales (Townshend 1991), la mayoría

[3] Las tres posibles interpretaciones serían "El tiempo vuela como una flecha", "Cronometra a las moscas como una flecha" y "A las moscas del tiempo les gusta una flecha".

[4] A las palabras desconocidas se les asigna normalmente el conjunto de las categorías *abiertas*, formada por aquellas categorías a las que es posible añadir nuevas palabras del lenguaje: nombres comunes y propios, verbos, adjectivos y adverbios. Esta asignación se refina normalmente con la ayuda de un adivinador, como ya se ha explicado anteriormente.

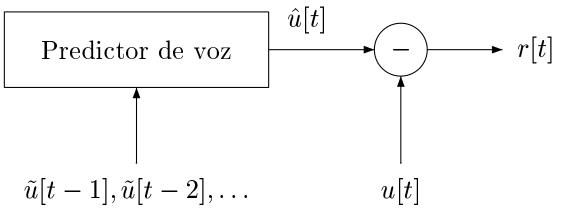


Figura 1.1: Esquema de la codificación predictiva sobre una señal $u[t]$. La señal enviada por el canal (después de ser codificada) es $r[t]$.

de los estándares actuales (Barnwell et al. 1996) consideran modelos lineales adaptativos para implementarlos debido al equilibrio que proporcionan entre complejidad y rendimiento. Los modelos no lineales como las redes neuronales recurrentes deberían, en principio, superar a las técnicas tradicionales.

Estudiaremos, por tanto, el rendimiento de ciertas redes a la hora de predecir la siguiente muestra de una secuencia de voz digitalizada.

2. REDES NEURONALES RECURRENTES

Este capítulo introduce los elementos principales que definen una neurona y la forma en que estas se combinan para constituir redes neuronales recurrentes. Además de plantear un modelo formal de red neuronal recurrente, que será utilizado en posteriores capítulos, se describen brevemente las principales aplicaciones de este tipo de redes.

2.1. Redes neuronales

Una *red neuronal* puede verse como una *máquina* diseñada originalmente para modelizar la forma en que el sistema nervioso de un ser vivo realiza una determinada tarea. Para lograr este objetivo, una red neuronal está formada por un conjunto de unidades de procesamiento interconectadas llamadas *neuronas*.

Cada neurona recibe como entrada un conjunto de señales discretas o continuas, las pondera e integra, y transmite el resultado a las neuronas conectadas a ella. Cada conexión entre dos neuronas tiene una determinada importancia asociada denominada *peso sináptico* o, simplemente, *peso*. En los pesos se suele guardar la mayor parte del conocimiento que la red neuronal tiene sobre la tarea en cuestión. El proceso mediante el cual se ajustan estos pesos para lograr un determinado objetivo se denomina *aprendizaje* o *entrenamiento* y el procedimiento concreto utilizado para ello se conoce como *algoritmo de aprendizaje* o *algoritmo de entrenamiento*. El ajuste de pesos es la principal forma de aprendizaje de las redes neuronales, aunque hay otras formas posibles.[1]

El artículo de McCulloch y Pitts (1943) se considera como el punto de arranque de la investigación en redes neuronales; este trabajo introdujo también la teoría de autómatas finitos como modelo computacional. McCulloch y Pitts propusieron un modelo simplificado de la actividad nerviosa real en el que cada neurona de una red neuronal podía activarse o desactivarse en función de lo que hicieran las neuronas conectadas a ella. Debido a que una neurona solo podía estar activada o desactivada, la capacidad computacional

[1] Por ejemplo, la modificación del número de neuronas o de la forma de conectarlas.

de la red completa se define en términos del conjunto de predicados lógicos que es capaz de computar. En este artículo ya aparecen redes neuronales recurrentes, a las que se denomina redes *con ciclos*.

Unos años después, Kleene (1956) reformuló algunos de estos resultados e introdujo una notación más compacta y general. En su artículo se define el concepto de *expresión regular* (allí llamado *evento regular*), tan importante para la teoría de lenguajes actual y sus aplicaciones. A partir de ahí, el campo de las redes neuronales y el de la teoría de lenguajes comienzan a tomar caminos separados. De hecho, el segundo acaparó una creciente atención desde entonces hasta nuestros días, mientras que el primero quedó prácticamente olvidado a raíz del trabajo de Minsky y Papert (1969). Salvo algunos trabajos aislados (Werbos 1974), las redes neuronales quedan relegadas a un segundo plano hasta la llegada de los ochenta.

Las redes neuronales destacan por su estructura fácilmente paralelizable y por su elevada capacidad de *generalización* (capacidad de producir salidas correctas para entradas no vistas durante el entrenamiento). Otras propiedades interesantes son:

- *No linealidad.* Una red neuronal puede ser lineal o no lineal. Esta última característica es muy importante, especialmente si se intenta modelizar sistemas generados mediante pautas no lineales.
- *Adaptabilidad.* Las redes neuronales son capaces de reajustar sus pesos para adaptarse a cambios en el entorno. Esto es especialmente útil cuando el entorno que suministra los datos de entrada es *no estacionario*, es decir, algunas de sus propiedades varían con el tiempo.
- *Tolerancia ante fallos.* Una red neuronal es tolerante ante fallos en el sentido de que los posibles fallos operacionales en partes de la red solo afectan débilmente al rendimiento de esta. Esta propiedad es debida a la naturaleza distribuida de la información almacenada o procesada en la red neuronal.

2.1.1. Modelo de neurona

En el modelo más habitual de neurona se identifican cinco elementos básicos para la j-ésima neurona de una red de tiempo discreto:

- Un conjunto de n señales de entrada, $z_i[t]$, $i = 1, \ldots, n$, que suministran a la neurona los datos del entorno; estos datos pueden ser externos a la red neuronal, pertenecientes a la salida de otras neuronas de la red, o bien correspondientes a la salida anterior de la propia neurona.

- Un conjunto de *sinapsis*, caracterizada cada una por un *peso* propio W_{ji}, $i = 1, \ldots, n$. El peso W_{ji} está asociado a la sinapsis que conecta la unidad i-ésima con la neurona j-ésima.
- Un *sesgo* W_j cuya presencia aumenta la capacidad de procesamiento de la neurona y que eleva o reduce la entrada a la neurona, según sea su valor positivo o negativo.
- Un *sumador* o *integrador* que suma las señales de entrada, ponderadas con sus respectivos pesos, y el sesgo.
- Una *función de activación* g que suele limitar la amplitud de la salida de la neurona.

Utilizando la notación definida anteriormente podemos describir la operación de una neurona mediante la ecuación que determina su *activación* en el instante $t + 1$:

$$z_j[t+1] = g\left(\sum_{i=1}^{n} W_{ji} z_i[t] + W_j\right) \tag{2.1}$$

Es habitual, y así se hará normalmente en este trabajo, considerar el sesgo como un peso más de la red y no distinguirlo del resto de pesos sinápticos. Por tanto, mientras no se diga lo contrario, el término *pesos* se refiere indistintamente tanto a W_{ji} como a W_j.

La función de activación es la que define finalmente la salida de la neurona. Las funciones de activación más utilizadas habitualmente son las siguientes:

1. *Función identidad.* Tiene la forma $g_I(x) = x$ y se utiliza cuando no se desea acotar la salida de la neurona.
2. *Función escalón.* Adopta la forma
$$g_E(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2.2}$$
y proporciona una salida con dos posibles valores. Es habitual encontrársela con el nombre de *función de Heaviside*.
3. *Función logística.* Las funciones sigmoideas son un conjunto de funciones crecientes, monótonas y acotadas que provocan una transformación no lineal de su argumento. Una de las más utilizadas es la función logística definida por
$$g_L(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

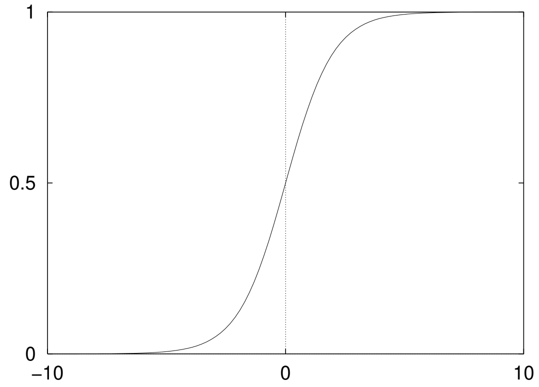


Figura 2.1: Representación gráfica de la función logística, $1/(1+e^{-x})$, para x entre -10 y $+10$.

La función logística está acotada entre 0 y 1. En la figura 2.1 se muestra su representación.

4. *Función tangente hiperbólica.* Otra función sigmoidea es la tangente hiperbólica:

$$g_T(x) = \tanh(x) \tag{2.4}$$

En este caso, la función está acotada entre -1 y 1. La función logística y la tangente hiperbólica se relacionan mediante la ecuación:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.5}$$

5. *Función de base radial.* Las más habituales son funciones gaussianas no monótonas del tipo

$$g_B(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right) \tag{2.6}$$

donde σ define la anchura. La función alcanza su valor máximo cuando la entrada es cero.

2.1.2. Poder computacional

El poder computacional de una única neurona es bastante limitado. Si consideramos una neurona con función de activación escalón, este poder de computación puede estudiarse desde dos puntos de vista equivalentes: el de la clasificación y el de la representación lógica.

- *Clasificación.* Dado un conjunto de n_P vectores de entrada, cada uno de n componentes, clasificados como pertenecientes a una de entre dos clases, una neurona puede (mediante la asignación correcta de valores a sus pesos y a su sesgo) clasificarlos correctamente[2] únicamente si se cumple la *condición de separabilidad lineal*: los conjuntos de vectores pertenecientes a cada clase deben ser separables por un hiperplano en el espacio n_P-dimensional.

- *Representación lógica.* La fracción de funciones lógicas de n variables que pueden representarse mediante una neurona con función de activación escalón decrece con n. Por ejemplo, para $n = 1$ las 4 posibles funciones (identidad, negación, siempre cierto y siempre falso) son computables a través de una neurona. Sin embargo, para $n = 2$ ya existen dos funciones no computables (la *o* exclusiva y su negación).

  Para n variables, hay un total de 2^{2^n} funciones lógicas. El número de funciones dentro de esas 2^{2^n} que son linealmente separables, es decir, que se pueden implementar con una neurona, $\mathrm{NFL}(n)$, no tiene una expresión exacta para $n > 8$, aunque la siguiente (Horne y Hush 1996) permite acotar su valor:

  $$4 \cdot 2^{n(n-1)/2} \leq \mathrm{NFL}(n) \leq \frac{2^{n^2+1}}{n!} \tag{2.7}$$

  que representa una fracción cada vez menor del total de funciones cuando se aumenta n.

Las limitaciones anteriores se cumplen independientemente de la función de activación utilizada. Si se interconecta un conjunto de neuronas formando una red de manera que la salida de algunas de ellas sea entrada de otras y se utilizan funciones de activación no lineales, pueden superarse estas limitaciones y las redes neuronales resultantes son capaces de emular la máquina universal de Turing (Hopcroft y Ullman 1979) y de computar, por tanto, cualquier función computable (Siegelmann y Sontag 1991). La salida de algunas de esas neuronas se convierte en la salida de la red neuronal y al resto de neuronas se las denomina unidades *ocultas* o de *estado*.

El problema que surge entonces es cómo determinar el valor de los pesos y sesgos de la red para poder computar una función determinada. Dado que el espacio de exploración puede ser enorme, se hace necesario el uso de heurísticas a través de algoritmos de entrenamiento, que no siempre logran dar con una solución adecuada.

[2] Si, por ejemplo, se utiliza una función de activación escalón, esta clasificación puede hacerse asignando el valor de salida 1 a los vectores de una clase y el valor 0 a los de la otra.

2.1.3. Topologías neuronales

La forma en que se interconectan las neuronas de una red neuronal define un grafo dirigido. Si este grafo es acíclico, la red se denomina *red neuronal hacia adelante* (en inglés, *feedforward network*) y en el caso de que posea ciclos, se denomina *red neuronal recurrente.* En el primer grupo están, entre otros, los perceptrones, las máquinas de vectores soporte y las redes de funciones de base radial[3] (Haykin 1999).

En el caso de las redes recurrentes, los ciclos existentes tienen un profundo impacto en la capacidad de aprendizaje de la red y las hacen especialmente indicadas para el procesamiento de secuencias temporales; son estas últimas, las redes recurrentes, las que se usan en esta tesis.

2.2. Redes recurrentes

Además de la ya discutida al comienzo del capítulo 1 (redes de tiempo continuo y redes de tiempo discreto), otras posibles clasificaciones dividen las redes neuronales recurrentes en redes de *estado continuo* o de *estado discreto*, según la forma de las funciones de activación empleadas. Según la forma en que se utilicen, cabe también hablar de redes recurrentes *de relajación* y de redes recurrentes *para el procesamiento temporal.* Las primeras evolucionan durante una serie de iteraciones desde un estado inicial a un estado normalmente estacionario, momento en el que se consulta la salida de la red; la salida de las redes usadas para el procesamiento temporal, por otro lado, se consulta continuamente tras cada iteración.

Entre los grupos anteriores, esta tesis se centra en las redes neuronales recurrentes de tiempo discreto con funciones de activación continuas aplicadas al procesamiento temporal de secuencias. Trabajaremos con sistemas dinámicos que poseen un estado que cambia conforme se procesa la secuencia de entrada y que proporciona una secuencia de salida a partir de las activaciones en cada instante de tiempo de las neuronas de salida. Más formalmente, una red neuronal recurrente de tiempo discreto (RNR a partir de ahora) de las que se van a considerar en esta tesis puede verse (Carrasco et al. 2000; Forcada y Gori 2001) como una séxtupla $N = (X, U, Y, \boldsymbol{g}_X, \boldsymbol{g}_Y, \boldsymbol{x}_0)$ donde:

- $X = [S_0, S_1]^{n_X}$ es el espacio de estados de la RNR.[4] S_0 y S_1 son los valores que definen el rango de salida de las funciones de activación

[3] Aunque existen también algunos trabajos sobre redes recurrentes de funciones de base radial (Cid-Sueiro et al. 1994).

[4] Esta condición puede relajarse para permitir que el intervalo de X sea abierto. La red LSTM, que veremos en el siguiente capítulo, es un ejemplo de RNR en la que ocurre esto.

utilizadas en la función siguiente estado y n_X es el número de unidades de estado.

- $U = \mathbb{R}^{n_U}$ es el espacio de los vectores de entrada y n_U el número de componentes de la entrada.
- $Y = [T_0, T_1]^{n_Y}$ es el espacio de salida[5] de la RNR. T_0 y T_1 definen el rango de salida de las funciones de activación utilizadas en la función de salida y n_Y es el número de componentes de los vectores de salida.
- $\boldsymbol{g}_X : X \times U \to X$ es la función de siguiente estado, que computa un nuevo estado $\boldsymbol{x}[t]$ a partir del estado anterior $\boldsymbol{x}[t-1]$ y la entrada actual $\boldsymbol{u}[t]$.
- $\boldsymbol{g}_Y$ es la función de salida, que habitualmente toma una de las dos formas siguientes:
    1. $\boldsymbol{g}_Y : X \times U \to Y$. La salida $\boldsymbol{y}[t]$ se obtiene a partir del estado anterior $\boldsymbol{x}[t-1]$ y la entrada actual $\boldsymbol{u}[t]$; este tipo de redes se conocen como *máquinas neuronales de estados de Mealy*.
    2. $\boldsymbol{g}_Y : X \to Y$. La nueva salida $\boldsymbol{y}[t]$ se obtiene a partir del estado recien alcanzado $\boldsymbol{x}[t]$; estas redes se conocen como *máquinas neuronales de estados de Moore*.
- $\boldsymbol{x}_0$ es el estado inicial de la RNR, es decir, $\boldsymbol{x}[0]$.

En el capítulo 3 se muestran algunas topologías neuronales de carácter recurrente; en el capítulo 4 se describen los algoritmos de entrenamiento más utilizados sobre RNR.

2.3. Aplicación de las redes recurrentes al procesamiento de secuencias

A continuación se enumeran, siguiendo a Forcada y Gori (2001), algunas de las tareas relacionadas con el procesamiento de secuencias a las que se han aplicado las RNR. Los elementos de la siguiente lista no son excluyentes, es decir, alguna tarea podría englobarse en más de una categoría.

Predicción de series temporales. Esta es una de las aplicaciones más habituales de las RNR. A partir de la historia pasada de una o más variables, la red neuronal debe proporcionar una predicción lo más correcta posible de su valor futuro. La mayor parte de los estudios de

[5] Si la función de salida no está acotada, el intervalo de Y puede ser abierto.

este tipo se centran en series económicas (McCluskey 1993) o tomadas de fenómenos naturales (Aussem et al. 1995), pero hay otras aplicaciones como la continuación de melodías inacabadas (Mozer 1994). Por otro lado, muchas de las tareas que se indican en los siguientes grupos y cualquiera de las abordadas en esta tesis pueden enfocarse como una tarea de predicción.

Procesamiento del lenguaje humano. El análisis sintáctico de frases o el estudio de regularidades en el lenguaje son algunas de las tareas relacionadas con el lenguaje humano (escrito) a las que se han aplicado las RNR (Elman 1990; 1991).

Ecualización de canales digitales. Los efectos del canal sobre la señal transmitida en comunicaciones digitales pueden hacer que esta sea irreconocible al llegar al receptor. Se hace necesario, por tanto, el uso de algún tipo de filtro inverso que deshaga estos efectos y proporcione una señal similar a la original. Esta tarea de traducción de señales se conoce normalmente como *ecualización* y varios trabajos se han acercado a ella con RNR (Kechriotis et al. 1994; Ortiz Fuentes y Forcada 1997; Cid-Sueiro et al. 1994).

Codificación del habla. Existe gran cantidad de técnicas para comprimir una señal de voz de manera que pueda ser transmitida por un canal con el menor número de bits por segundo posible (para una calidad de recepción determinada). Algunas de estas técnicas se basan en la llamada *codificación predictiva*; en ella lo que se envía no es la señal, sino la *diferencia* entre su valor real y el valor predicho por un determinado predictor. Si el predictor es bueno, esta diferencia será pequeña y se necesitarán pocos bits para codificarla. Las RNR también han sido usadas como predictores para la codificación del habla (Haykin y Li 1995).

Reconocimiento del habla. El reconocimiento del habla puede considerarse como una tarea de traducción de secuencias (por ejemplo, cuando se asigna una secuencia de fonemas a una secuencia de vectores acústicos obtenidos mediante el procesamiento de una señal de voz) o como una tarea de clasificación de secuencias (por ejemplo, al asignar una palabra a una serie de vectores acústicos). Existen varias referencias que utilizan RNR para el reconocimiento del habla (Robinson y Fallside 1991).

Inferencia gramatical. Dado un conjunto de cadenas pertenecientes a un cierto lenguaje, uno de los problemas más estudiados es la inferencia de un modelo (un autómata finito o una gramática independiente del contexto, por ejemplo) que describa de manera correcta ese lenguaje.

Este es posiblemente el campo en el que las RNR han proporcionado mejores resultados (Cleeremans et al. 1989; Castaño et al. 1995; Carrasco et al. 2000).

Control de sistemas. Las RNR pueden ser también entrenadas (Puskorius y Feldkamp 1994) para controlar un sistema real de manera que su salida siga un determinado patrón temporal.

3. MODELOS

En este capítulo se describen algunos modelos de RNR, haciendo especial énfasis en aquellos que se van a utilizar en esta tesis. No pretende, por tanto, ser una recopilación exhaustiva de los distintos tipos de RNR. Por su relativa novedad y escasa difusión hasta la fecha, se muestra con más detalle el modelo de memorias a corto y largo plazo.

3.1. Modelos recurrentes tradicionales

En este apartado se presentan algunos modelos recurrentes que podemos considerar "clásicos" (Carrasco et al. 2000), ya que han sido utilizados en multitud de referencias desde prácticamente los inicios de las RNR.

3.1.1. Redes de primer orden

Una de las topologías neuronales más utilizadas en este trabajo es la red recurrente de propagación de errores o red parcialmente recurrente (Robinson y Fallside 1991) de primer orden (RPR), cuya dinámica viene dada por:

$$y_i[t] = g_Y(Y_i[t]) \qquad i = 1, \ldots, n_Y \tag{3.1}$$

$$Y_i[t] = \sum_{j=1}^{n_U} W_{i,j}^{y,u} u_j[t] + \sum_{j=1}^{n_X} W_{i,j}^{y,x} x_j[t-1] + W_i^y \tag{3.2}$$

$$x_i[t] = g_X(X_i[t]) \qquad i = 1, \ldots, n_X \tag{3.3}$$

$$X_i[t] = \sum_{j=1}^{n_U} W_{i,j}^{x,u} u_j[t] + \sum_{j=1}^{n_X} W_{i,j}^{x,x} x_j[t-1] + W_i^x \tag{3.4}$$

donde g_Y y g_X son funciones de activación equivalentes a las de la sección 2.2, n_X es el número de neuronas de estado, n_U es el número de entradas a la red y n_Y es el número de neuronas de salida. Un diagrama de la RPR puede observarse en la figura 3.1.

A continuación sigue una breve explicación sobre la notación utilizada para los pesos y sesgos de la red: los superíndices indican el cálculo en el que

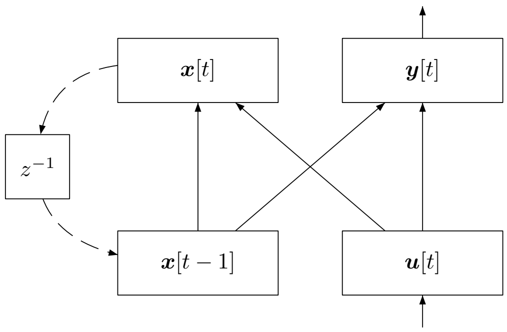


Figura 3.1: Esquema de la red parcialmente recurrente de primer orden (Robinson y Fallside 1991). El bloque etiquetado con z^{-1} representa un vector de células de retardo temporal.

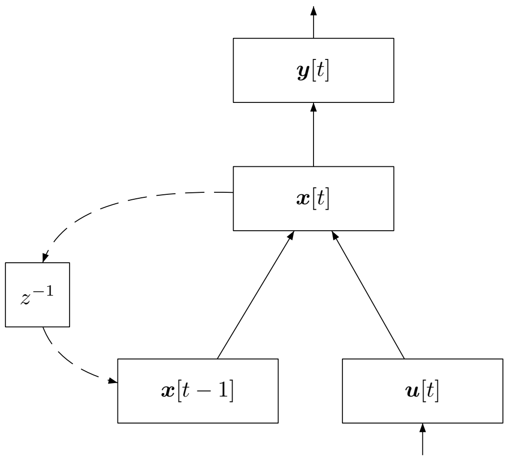


Figura 3.2: Esquema de la red recurrente simple de primer orden (Elman 1990).

está implicado el peso: por ejemplo, $W_{i,j}^{y,u}$ indica que ese peso contribuye a determinar la salida y a partir de la entrada u. Por otra parte, W_i^x indica que este peso es un sesgo implicado en el cálculo del estado x. Los subíndices muestran las unidades concretas que se ven afectadas (conectadas) y van paralelos a los superíndices.

Otro tipo de red es la red recurrente simple de primer orden (RRS) propuesta por Elman (1990), que puede observarse en la figura 3.2 y cuya

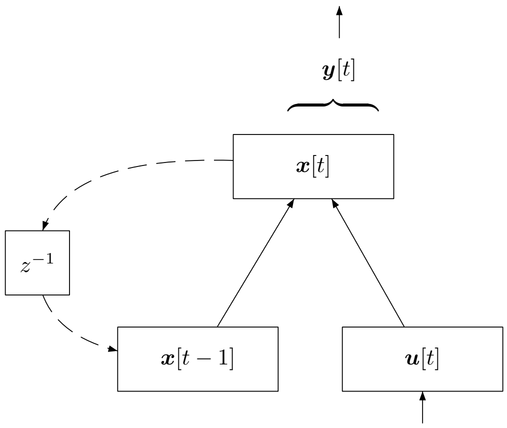


Figura 3.3: Esquema de la red totalmente recurrente de primer orden (Williams y Zipser 1989).

dinámica viene determinada por las ecuaciones:

$$y_i[t] = g_Y(Y_i[t]) \qquad i = 1, \ldots, n_Y \tag{3.5}$$

$$Y_i[t] = \sum_{j=1}^{n_X} W_{i,j}^{y,x} x_j[t] + W_i^y \tag{3.6}$$

$$x_i[t] = g_X(X_i[t]) \qquad i = 1, \ldots, n_X \tag{3.7}$$

$$X_i[t] = \sum_{j=1}^{n_U} W_{i,j}^{x,u} u_j[t] + \sum_{j=1}^{n_X} W_{i,j}^{x,x} x_j[t-1] + W_i^x \tag{3.8}$$

La red totalmente recurrente (Williams y Zipser 1989) de primer orden (RTR) viene descrita por:

$$y_i[t] = x_i[t] \qquad i = 1, \ldots, n_Y \tag{3.9}$$

$$x_i[t] = g_X\left(\sum_{j=1}^{n_U} W_{i,j}^{x,u} u_j[t] + \sum_{j=1}^{n_X} W_{i,j}^{x,x} x_j[t-1] + W_i^x \right) \tag{3.10}$$

normalmente con $n_X \geq n_Y$. Un esquema de la RTR se muestra en la figura 3.3.

Según lo discutido en la sección 2.2, la RRS y la RTR son máquinas de Moore neuronales y la RPR es una máquina de Mealy neuronal. El estado de la red se define como $\boldsymbol{x}[t]$.

3.1.2. Redes de segundo orden

La red recurrente simple de segundo orden (RRS2), utilizada por Carrasco et al. (1996) y por Blair y Pollack (1997), viene dada por:

$$y_i[t] = g_Y \left(\sum_{j=1}^{n_X} W_{i,j}^{y,x} x_j[t] + W_i^y \right) \tag{3.11}$$

$$x_i[t] = g_X \left(\sum_{j=1}^{n_X} \sum_{k=1}^{n_U} W_{i,j,k}^{x,x,u} x_j[t-1] u_k[t] + W_i^x \right) \tag{3.12}$$

También puede obtenerse una versión de segundo orden (RPR2) de la RPR. Esta topología ha sido utilizada, entre otros, por Omlin y Giles (1996) y responde a las ecuaciones:

$$y_i[t] = g_Y \left(\sum_{j=1}^{n_X} \sum_{k=1}^{n_U} W_{i,j,k}^{y,x,u} x_j[t-1] u_k[t] + W_i^y \right) \tag{3.13}$$

$$x_i[t] = g_X \left(\sum_{j=1}^{n_X} \sum_{k=1}^{n_U} W_{i,j,k}^{x,x,u} x_j[t-1] u_k[t] + W_i^x \right) \tag{3.14}$$

La red totalmente recurrente de segundo orden (RTR2) propuesta por Giles et al. (1992) se define a partir de:

$$y_i[t] = x_i[t] \tag{3.15}$$

$$x_i[t] = g_X \left(\sum_{j=1}^{n_X} \sum_{k=1}^{n_U} W_{i,j,k}^{x,x,u} x_j[t-1] u_k[t] + W_i^x \right) \tag{3.16}$$

donde, como en la RTR, se suele hacer $n_X \geq n_Y$.

Según lo discutido en la sección 2.2, la RRS2 y la RTR2 son máquinas de Moore neuronales de segundo orden y la RPR2 es una máquina de Mealy neuronal de segundo orden. El estado de la red se considera, de nuevo, almacenado en $\boldsymbol{x}[t]$.

3.1.3. Adición de una ventana temporal

Todos los modelos anteriores de RNR pueden ser ampliados con la incorporación a sus entradas de una memoria explícita a corto plazo. De esta forma, la entrada a la red consistirá en el valor actual $u[t]$ concatenado con los $p-1$ valores anteriores $u[t-1], \ldots, u[t-p+1]$. A p se le conoce como orden de la memoria de entrada u *orden de entrada*. Una de las primeras utilizaciones de esta idea (con redes no recurrentes) fue el proyecto NetTalk (Sejnowski y Rosenberg 1987).

Con esta ampliación, la red recurrente tiene a su disposición de forma explícita la historia más reciente y puede, en teoría, utilizar el estado para almacenar información relativa a un pasado más remoto.

Al incorporar una ventana de entradas de orden p la ecuación (3.4), por ejemplo, se convierte ahora en:

$$X_i[t] = \sum_{j=1}^{n_U} \sum_{k=1}^{p} W_{i,j(k)}^{x,u} u_j[t-k+1] + \sum_{j=1}^{n_X} W_{i,j}^{x,x} x_j[t-1] + W_i^x \tag{3.17}$$

donde se ha utilizado $W_{i,j(k)}^{x,u}$ para referirse al peso que une $u_j[t-k+1]$ con la neurona i del estado. Las ecuaciones de las otras redes recurrentes se modificarían de manera similar.

A efectos prácticos, es aconsejable no alterar las ecuaciones y considerar que la entrada de la red tiene tamaño $n'_U = p \cdot n_U$, además de controlar esa entrada realizando el desplazamiento de todos los componentes en cada instante de tiempo para descartar la información más antigua e incorporar la nueva.

Ya que en esta tesis se pretende estudiar la capacidad de las RNR para desarrollar *per se* una representación interna de los contextos de las secuencias procesadas, no se considerarán apenas modificaciones de este tipo, salvo en el capítulo 9.

3.1.4. Otras redes

Las RNR de primer y segundo orden mostradas hasta ahora pueden considerarse *de estado oculto*; el estado de estas redes se almacena en $\boldsymbol{x}[t]$. Existen otras redes, como la red NARX, que se pueden considerar de *estado visible*, ya que su estado es simplemente una combinación de las entradas y salidas anteriores.

Además de incorporar a la red las $p-1$ entradas anteriores, las redes NARX[1] (Narendra y Parthasarathy 1990) añaden las q salidas anteriores de la red. Una red NARX con orden de entrada p y *orden de salida* q viene definida por las ecuaciones:

$$y_i[t] = g_Y\left(\sum_{j=1}^{n_Z} W_{i,j}^{y,z} z_j[t] + W_i^y\right) \tag{3.18}$$

$$z_i[t] = g_Z\left(\sum_{j=1}^{n_U}\sum_{k=1}^{p} W_{i,j(k)}^{z,u} u_j[t-k+1] + \sum_{j=1}^{n_Y}\sum_{k=1}^{q} W_{i,j(k)}^{z,y} y_j[t-k] + W_i^z\right) \tag{3.19}$$

Se ha usado z_i para las neuronas intermedias y no x_i para evitar confusiones, ya que, como se ha comentado, el estado de las NARX no está en ellas, sino en la ventana de entradas y en la de salidas.

Cuando el estado de la red NARX está formado únicamente por las entradas anteriores de la red, es decir, cuando $q=0$, se obtiene una red denominada tradicionalmente *red neuronal de retardos temporales* (Sejnowski y Rosenberg 1987) (TDNN, por el inglés *time-delayed neural network*).

Por otro lado, si eliminamos el conjunto de neuronas intermedias (con activaciones $z_i[t]$) de la red NARX para permitir así la conexión directa entre la entrada y la salida de la red, y usamos la identidad como función de activación para g_Y, obtenemos el modelo de filtro de *respuesta de tiempo infinito al impulso* (IIR, por el inglés *infinite-time impulse response*), ampliamente utilizado en teoría de la señal (Oppenheim y Schafer 1989; Proakis y Manolakis 1998) y cuya ecuación es:

$$y_i[t] = \sum_{j=1}^{n_U}\sum_{k=1}^{p} W_{i,j(k)}^{y,u} u_j[t-k+1] + \sum_{j=1}^{n_Y}\sum_{k=1}^{q} W_{i,j(k)}^{y,y} y_j[t-k] + W_i^y \tag{3.20}$$

Finalmente, si las consideraciones anteriores se aplican a una red TDNN (esto es, si se hace $q=0$ en la ecuación anterior), se obtiene las ecuaciones de un filtro de *respuesta de tiempo finito al impulso* (FIR, por el inglés *finite-time impulse response*), también muy usado en teoría la señal (Oppenheim y Schafer 1989; Proakis y Manolakis 1998):

$$y_i[t] = \sum_{j=1}^{n_U}\sum_{k=1}^{p} W_{i,j(k)}^{y,u} u_j[t-k+1] + W_i^y \tag{3.21}$$

[1] Aunque aparezca en una sección aparte, la red NARX también puede considerarse como una red de primer orden.

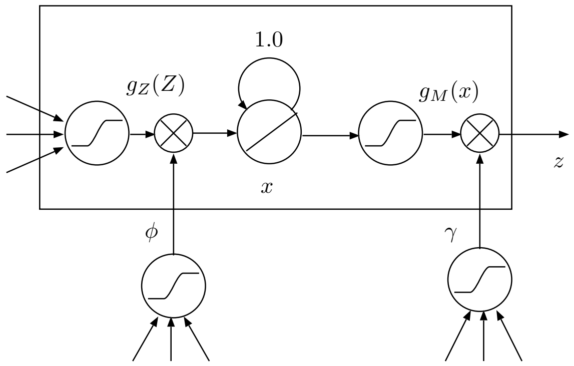


Figura 3.4: Un bloque de memoria con una única celda. La entrada de la celda se representa con Z, la activación de la compuerta de entrada con ϕ, la activación de la compuerta de salida con γ, la activación del CEC con x y la activación global de la celda con z.

3.2. Modelo de memoria a corto y largo plazo

Para comprender totalmente el modelo de memoria a corto y largo plazo (LSTM, por el inglés *long short-term memory*) es fundamental conocer el problema del gradiente evanescente que las motiva y la manera con la que el modelo intenta superar este problema. Pero para ello es necesario discutir previamente ciertos aspectos relativos a los algoritmos de entrenamiento de RNR y al cálculo del gradiente de la función de error. En el capítulo siguiente se estudian estos aspectos y, entonces, se introducen los principios en los que se basa la red LSTM. En este apartado, por tanto, nos limitaremos a presentar la configuración y las ecuaciones que definen el modelo.

El componente básico del modelo LSTM (Hochreiter y Schmidhuber 1997) es el *bloque de memoria*, que contiene una o más *celdas* de memoria, una *compuerta de entrada* y una *compuerta de salida*. Las compuertas son unidades multiplicativas con activación continua (normalmente dentro del intervalo unidad) y son compartidas por todas las celdas que pertenecen a un mismo bloque de memoria. Cada celda contiene una unidad lineal con una conexión recurrente local llamada *carrusel de error constante* (CEC); la activación del CEC se conoce como el *estado* de la celda.

La figura 3.4 muestra uno de estos bloques de memoria con una única celda; esta figura es útil también para introducir la notación relativa al modelo que se utilizará a lo largo de la tesis. La figura 3.5 muestra un bloque de memoria con dos celdas, que comparten las compuertas del bloque.

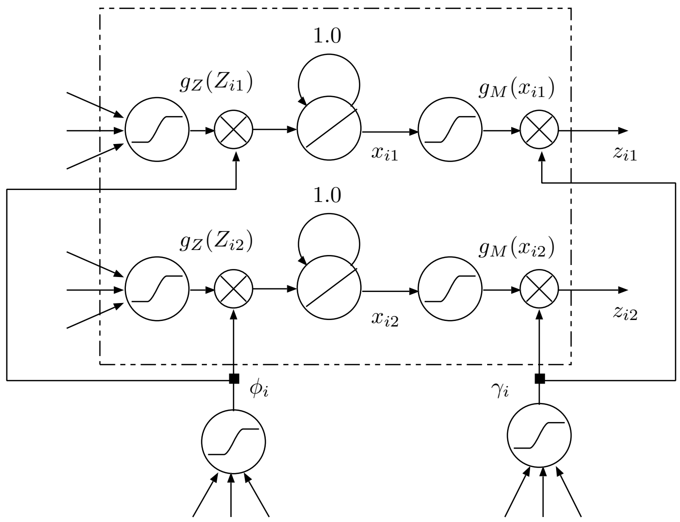


Figura 3.5: El bloque de memoria i-ésimo con dos celdas.

Cada celda recibe como entrada una colección de valores (ponderados mediante los pesos correspondientes) provenientes de la entrada de la red y de las salidas de todas las celdas del modelo en el instante anterior. La compuerta de entrada se encarga de permitir o impedir el acceso de estos valores al CEC del interior de la celda. La compuerta de salida realiza una acción similar sobre la salida de la celda, tolerando o reprimiendo la difusión del estado del CEC al resto de la red.

Los bloques de memoria configuran una red LSTM como puede verse en la figura 3.6, donde no se indican los sesgos de las distintas neuronas del modelo. La existencia de las conexiones con pesos $W^{y,u}$ determina la naturaleza de la red según lo discutido en 2.2. Así, si se permite la existencia de esta conexión, la red LSTM se puede considerar como una máquina neuronal de estados de Mealy; si no se permite, la red LSTM puede considerarse como una máquina neuronal de estados de Moore. El estado de la red LSTM está formado por las activaciones de las compuertas, el CEC y las celdas de los bloques de memoria.[2]

Basándonos en la notación ya introducida, a continuación se describe brevemente la correspondiente a la red LSTM. Sean n_U, n_Y, n_M y n_C el número de neuronas de entrada, salida, bloques de memoria y celdas por bloque, respectivamente. La entrada en el instante t se denota con $\boldsymbol{u}[t]$ y

[2]Nótese cómo los valores de algunos componentes del estado, en especial la activación de los CEC, no están acotados.

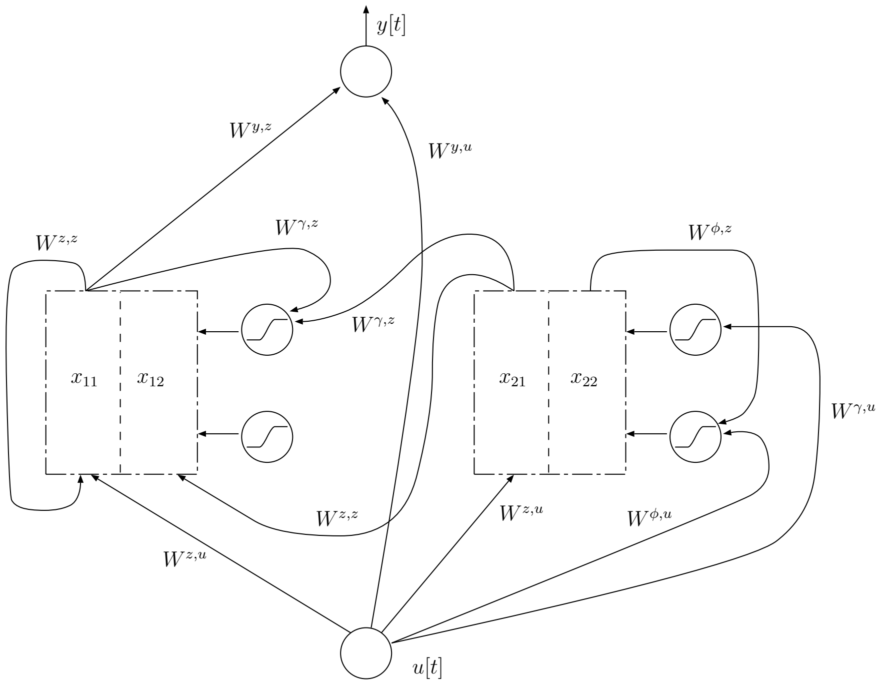


Figura 3.6: Una red LSTM con dos bloques de memoria de dos celdas cada uno. Solo se muestran algunas conexiones y no se muestran los sesgos.

la salida correspondiente con $\boldsymbol{y}[t]$. La salida de la j-ésima celda del bloque i-ésimo se representa con $z_{ij}[t]$.

Como ya se vio antes, al representar los pesos, los superíndices indican el cálculo en el que está involucrado el peso en cuestión: el "ϕ, z" en $W^{\phi,z}$ indica que el peso se usa para calcular la activación de una compuerta de entrada (ϕ) a partir de la de una celda (z); el "γ" en W^{γ} indica que el sesgo se usa para calcular la activación de una compuerta de salida. Los subíndices indican las unidades particulares afectadas por el peso y van paralelos a los superíndices.

3.2.1. Cálculo de la función de salida

La activación de la compuerta de entrada del i-ésimo bloque de memoria ϕ_i se calcula como:

$$\Phi_i[t] = \sum_{j=1}^{n_M} \sum_{k=1}^{n_C} W_{i,jk}^{\phi,z}\, z_{jk}[t-1] + \sum_{j=1}^{n_U} W_{i,j}^{\phi,u}\, u_j[t] + W_i^{\phi} \tag{3.22}$$

$$\phi_i[t] = g_C(\Phi_i[t]) \tag{3.23}$$

donde g_C es la función de activación de todas las compuertas de la red (la función logística, si no se dice lo contrario).

La activación de la compuerta de salida se calcula como sigue:

$$\Gamma_i[t] = \sum_{j=1}^{n_M} \sum_{k=1}^{n_C} W_{i,jk}^{\gamma,z} \, z_{jk}[t-1] + \sum_{j=1}^{n_U} W_{i,j}^{\gamma,u} \, u_j[t] + W_i^{\gamma} \tag{3.24}$$

$$\gamma_i[t] = g_C(\Gamma_i[t]) \tag{3.25}$$

El estado interno de la celda de memoria se calcula sumando la entrada modificada por la compuerta correspondiente con el estado en el instante anterior $t-1$:

$$x_{ij}[t] = x_{ij}[t-1] + \phi_i[t] \, g_Z(Z_{ij}[t]) \tag{3.26}$$

donde g_Z es una función de activación (normalmente sigmoidea y acotada) y:

$$Z_{ij}[t] = \sum_{k=1}^{n_M} \sum_{l=1}^{n_C} W_{ij,kl}^{z,z} \, z_{kl}[t-1] + \sum_{k=1}^{n_U} W_{ij,k}^{z,u} \, u_k[t] + W_{ij}^{z} \tag{3.27}$$

con $x_{ij}[0] = 0$ para todo ij. La salida de la celda se calcula ajustando el estado del CEC mediante una nueva función de activación g_M y multiplicando el valor resultante por la activación de la compuerta de salida:

$$z_{ij}[t] = \gamma_i[t] \, g_M(x_{ij}[t]) \tag{3.28}$$

Finalmente, si permitimos la conexión directa entre la entrada y las neuronas de salida, la salida global de la red se calcula mediante:

$$Y_i[t] = \sum_{j=1}^{n_M} \sum_{k=1}^{n_C} W_{i,jk}^{y,z} \, z_{jk}[t] + \sum_{j=1}^{n_U} W_{i,j}^{y,u} \, u_j[t] + W_i^{y} \tag{3.29}$$

$$y_i[t] = g_Y(Y_i[t]) \tag{3.30}$$

donde g_Y es, otra vez, una función de activación adecuada.

Los pesos que inciden en las compuertas de entrada y salida se suelen iniciar de forma que $\phi_i[0]$ y $\gamma_i[0]$ estén cerca de 0; de esta manera los bloques de memoria están desactivados inicialmente y el entrenamiento se centra en las conexiones directas entre la entrada y las neuronas de salida. Así, el protagonismo de los bloques de memoria va aumentando paulatinamente conforme el algoritmo de aprendizaje determina su rol.

Finalmente, el número de pesos a ajustar en una red LSTM es $(n_M n_C + n_U + 1)(n_Y + n_M n_C + 2n_M)$.

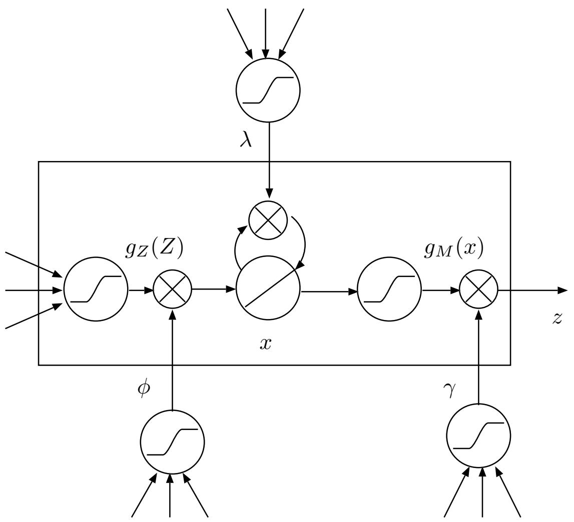


Figura 3.7: Un bloque de memoria con una compuerta de olvido con activación λ.

3.2.2. Limitaciones de la red LSTM original

El modelo inicial de la red LSTM (Hochreiter y Schmidhuber 1997) ha sido aumentado desde su concepción original para superar algunos problemas detectados. A continuación, se muestran dos de las principales modificaciones.

Reticencia a olvidar. Cuando la red LSTM presentada hasta ahora se aplica a tareas de procesamiento de secuencias de longitud arbitrariamente larga de forma continua, el modelo se vuelve inestable debido a que bajo determinadas circunstancias el estado de los CEC crece indefinidamente (Gers et al. 2000). Para paliar este problema, se incorpora una tercera compuerta a los bloques de memoria: la *compuerta de olvido*.

La compuerta de olvido (Gers et al. 2000) puede rebajar e incluso anular el estado interno de la celda, esto es, la activación del CEC, cuando sus contenidos caducan. Estas compuertas permiten que la red LSTM pueda procesar establemente secuencias de longitud arbitrariamente larga.

La figura 3.7 muestra la nueva imagen de los bloques de memoria con la adición de la compuerta de olvido. Como ocurría con las compuertas de entrada y de salida, la compuerta de olvido es compartida por todas las celdas del bloque.

La activación de las compuertas de olvido λ_i se obtiene calculando:

$$\Lambda_i[t] = \sum_{j=1}^{n_M} \sum_{k=1}^{n_C} W_{i,jk}^{\lambda,z} \, z_{jk}[t-1] + \sum_{j=1}^{n_U} W_{i,j}^{\lambda,u} \, u_j[t] + W_i^{\lambda} \quad (3.31)$$

$$\lambda_i[t] = g_C(\Lambda_i[t]) \quad (3.32)$$

Al considerar las compuertas de olvido, la ecuación (3.26) cambia su forma. El estado interno de la celda de memoria se calcula ahora sumando la entrada modificada por la compuerta correspondiente y el estado en el instante anterior $t-1$ multiplicado por la correspondiente compuerta de olvido:

$$x_{ij}[t] = \lambda_i[t] \, x_{ij}[t-1] + \phi_i[t] \, g_Z(Z_{ij}[t]) \quad (3.33)$$

Los pesos de las compuertas de olvido se inicializan normalmente de manera que $\lambda_i[0]$ esté cerca de 1; con esta inicialización, las celdas no olvidan nada hasta que aprendan cómo olvidar.

Aislamiento del estado. Al modelo base anterior se le ha añadido recientemente (Gers y Schmidhuber 2001) una serie de *conexiones de mirilla*, que permiten la conexión directa entre el CEC y las compuertas que lo controlan. Aunque la misión de las compuertas es controlar de un modo u otro el estado de los CEC, el modelo original no permitía que las compuertas pudieran acceder directamente a dicho estado interno; con esta ampliación del modelo, cada compuerta dispone de una *mirilla* desde la que poder observar el interior del bloque de memoria. Estas nuevas conexiones son necesarias, como veremos, para poder aprender algunas tareas.

La figura 3.8 muestra el bloque de memoria de la figura 3.7 con el añadido de las conexiones de mirilla. Cada conexión de mirilla tiene un peso asociado que deberá ser ajustado por el algoritmo de entrenamiento correspondiente.

3.3. Red recurrente en cascada

El último de los modelos que veremos en esta sección es la red neuronal recurrente en cascada (RNRC), propuesta por Haykin y Li (1995) para realizar la predicción de la siguiente muestra de una señal de voz. Dado que la RNRC se ha utilizado únicamente en este tipo de tareas de procesamiento de señales, describiremos el modelo en este contexto, de manera que la salida de la red será un único valor y la entrada será la concatenación de las muestras recientes de la señal, esto es, una ventana temporal de entradas.

Grosso modo, la RNRC puede considerarse como una serie de n_M redes en cascada con pesos compartidos. Todas las redes son RPR, excepto la

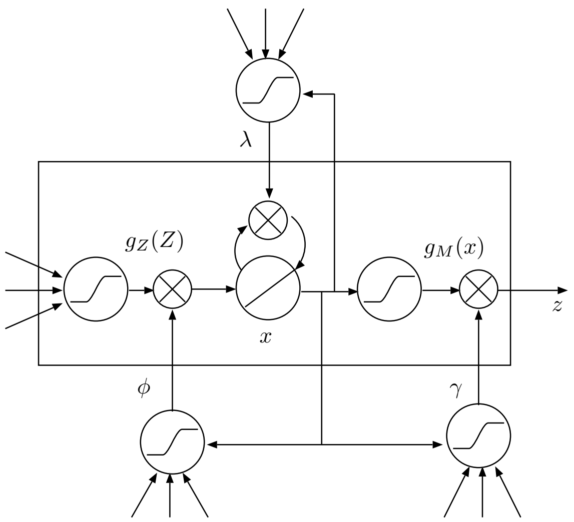


Figura 3.8: Un bloque de memoria con conexiones de mirilla, que conectan el estado del CEC con cada una de las compuertas.

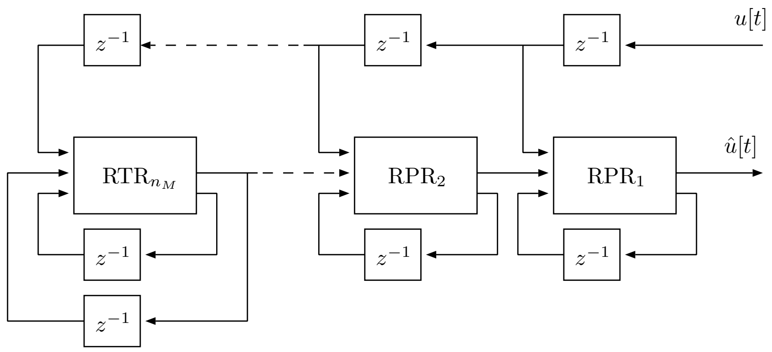


Figura 3.9: Esquema de la red recurrente en cascada (Haykin y Li 1995).

n_M-ésima, que es una RTR con el mismo número de pesos (véase el apartado 3.1.1). Un esquema de la RNRC se muestra en la figura 3.9.

Teniendo en cuenta que cada módulo de la red puede verse como una RPR, que los pesos son idénticos para los distintos módulos y las ecuaciones (3.1) a (3.4), la salida del módulo m-ésimo en el instante t viene definida por:

$$y^{(m)}[t] = g_Y \left(\sum_{i=1}^{n_X} x_i^{(m)}[t-1]\, W_i^{y,x} + v^{(m)} W^{y,v} + \sum_{i=1}^{n_U} u_i^{(m)}[t]\, W_i^{y,u} + W^y \right) \tag{3.34}$$

donde el estado del i-ésimo módulo es:

$$x_i^{(m)}[t] = g_X \left(\sum_{j=1}^{n_X} x_j^{(m)}[t-1]\, W_{i,j}^{x,x} + v^{(m)} W_i^{x,v} + \sum_{j=1}^{n_U} u_j^{(m)}[t]\, W_{i,j}^{x,u} + W_i^x \right) \tag{3.35}$$

En lo anterior, g_X y g_Y son funciones de activación y $v^{(m)}$ es la salida del módulo anterior si $m < n_M$ o bien la salida pasada del propio módulo si $m = n_M$, es decir:

$$v^{(m)}[t] = \begin{cases} y^{(m+1)}[t] & m < n_M \\ y^{(n_M)}[t-1] & m = n_M \end{cases} \tag{3.36}$$

El error de la red[3] es una combinación lineal de los errores $e^{(m)}$ de cada uno de los módulos:[4]

$$E[t] = \frac{1}{2} \sum_{m=1}^{n_M} \lambda^{m-1} \left(e^{(m)}[t] \right)^2 \tag{3.37}$$

donde λ es un *factor de olvido exponencial* en el rango $0 < \lambda \leq 1$; el inverso de $1 - \lambda$ es una especie de indicador de la *memoria* de la RNRC. La señal de error de cada módulo, utilizada en la ecuación anterior es:

$$e^{(m)}[t] = \left(d^{(m)}[t] - y^{(m)}[t] \right) \tag{3.38}$$

Al aplicar la red a la predicción de señales numéricas de la forma $s[t]$, se hace $n_U = p$, donde p es el orden del predictor. Las p muestras de entrada a cada módulo son:

$$u_i^{(m)}[t] = s[t - (m + i - 1)] \tag{3.39}$$

y la salida deseada del módulo m es:

$$d^{(m)}[t] = s[t - m + 1] \tag{3.40}$$

[3]Como veremos en el siguiente capítulo, el error es utilizado por los algoritmos de entrenamiento para ajustar los pesos de la red neuronal.

[4]Nótese que el superíndice de λ es en este caso un exponente.

De esta manera la salida $y^{(m)}[t]$ se puede interpretar como una estimación del valor de la señal en el instante $t - m + 1$ y, por tanto:

$$y^{(1)} = \hat{s}[t] \tag{3.41}$$

La salida del primer módulo es la predicción global del sistema.

4. ENTRENAMIENTO

Una vez seleccionado el modelo neuronal con el que resolver un determinado problema, surge la cuestión de cómo deteminar el valor de los pesos de la red que permiten resolverlo con éxito. Los algoritmos de entrenamiento se encargan de intentar encontrar esta configuración correcta. En este capítulo se muestran algunos de los más utilizados con RNR, principalmente el descenso por el gradiente y el filtro de Kalman extendido desacoplado. Ambos se basan en el cálculo de la derivada de la función de error; dos son las formas más habituales de obtener estas derivadas: el aprendizaje recurente en tiempo real y la retropropagación a través del tiempo.

4.1. Algoritmos de entrenamiento supervisado

Cuando se desea resolver una determinada tarea con la ayuda de una RNR, lo primero que se debe considerar es el tipo de red que se va a utilizar. La elección no es trivial: hay problemas que se resuelven aceptablemente bien con un tipo de red, pero que son muy difíciles (o imposibles) de resolver con otro.

El número de entradas y de neuronas de salida vienen determinados por la naturaleza de la tarea a resolver, por el tipo de secuencias a procesar o por la codificación utilizada para sus elementos. Otros valores como el número de neuronas de estado o el orden de entrada o salida deberán ser determinados tras experimentar con distintas combinaciones o usar los límites dados en la teoría para algunos tipos de tarea.[1]

A continuación debe entrenarse la red para ajustar sus parámetros libres (los pesos habitualmente). Atendiendo a la forma en que se presentan los datos, los algoritmos de aprendizaje pueden dividirse en dos categorías:

[1]Por ejemplo, un autómata finito determinista puede codificarse sobre una RRS con $n_X = |Q||\Sigma|$ neuronas de estado (Carrasco et al. 2000). Si estamos intentando que una RRS aprenda un lenguaje regular a partir de ejemplos, podemos hacer una estimación de $|Q|$ y utilizarlo para calcular el valor de n_X.

- *Entrenamiento supervisado.* En este tipo de algoritmos la red neuronal cuenta con el apoyo externo de un “maestro” que informa de la corrección de la salida producida por la red de acuerdo con la salida considerada correcta.

- *Entrenamiento no supervisado.* En este caso no existe tal maestro y la red neuronal debe extraer sin ayuda características de los datos que se le suministra.

Este trabajo se centra en el uso de RNR para la predicción del siguiente elemento de distintas secuencias; por ello, todos los algoritmos de aprendizaje estudiados serán supervisados. Durante el entrenamiento, la entrada al algoritmo será una representación del elemento actual (o del actual y de algunos de los anteriores, si se utiliza una ventana temporal de entradas) y la salida deseada será la representación del siguiente elemento.

Para entrenar la RNR de forma supervisada se necesita normalmente algún tipo de *medida del error* $E[t]$ que describa la adecuación de la salida proporcionada por la red al valor deseado. Los parámetros se ajustan intentando minimizar este error.

La función de error más habitual es la función de error cuadrático, definida para el instante t como:

$$E[t] = \frac{1}{2} \sum_{i=1}^{n_Y} (d_i[t] - y_i[t])^2 \tag{4.1}$$

donde $d_i[t]$ es la *salida deseada* u *objetivo* para la i-ésima neurona de salida en el instante t e $y_i[t]$ es la salida correspondiente de la red.

Una posible forma de encontrar la solución que minimice el valor del error es la búsqueda exhaustiva sobre todas las posibles combinaciones de valores de los pesos (o sobre un conjunto finito lo suficientemente significativo de posibles valores). Evidentemente, esta forma de resolución es intratable en la mayoría de los casos. Si el problema a aprender es sencillo, puede que una estrategia basada en generar aleatoriamente conjuntos de valores para los pesos funcione (Schmidhuber y Hochreiter 1996). En general, sin embargo, se hace necesaria la utilización de algún tipo de heurística que recorte el espacio de soluciones a explorar; esta es la labor de los algoritmos de entrenamiento.

4.1.1. Aprendizaje en línea y fuera de línea

Supongamos una red neuronal que se está utilizando para el procesamiento de secuencias. Si la red se aplica a la clasificación de secuencias, por ejemplo, el conjunto de entrenamiento contendrá una serie de secuencias cuya clasificación es conocida de antemano. El algoritmo de entrenamiento debe conseguir que la red "aprenda" estas clasificaciones. Otras veces puede quererse que la red procese una única secuencia en tiempo real.

En algunos casos, la red neuronal se somete a una fase de entrenamiento, tras la cual sus pesos se *congelan*. Durante esta fase a la red se le presentan (normalmente más de una vez) los datos del llamado *conjunto de entrenamiento*. A continuación, esta red neuronal se *evalúa* sobre un nuevo conjunto de datos para determinar la corrección del aprendizaje.

En otros casos, las fases de entrenamiento y evaluación no están tan claramente separadas y la salida de la red se usa simultáneamente como punto de referencia para cambiar los pesos y como producto utilizado en la resolución de la tarea en cuestión.

Cada tipo de procesamiento requiere una estrategia de aprendizaje distinta. Según la forma de actualizar los parámetros libres (pesos y sesgos, normalmente) de la red neuronal, los algoritmos de entrenamiento supervisado pueden dividirse en las siguientes cuatro clases:

- *Entrenamiento en línea.* Puede subdividirse en:
  - *En línea por elementos o en línea puro.* Este tipo de algoritmo se utiliza cuando se pretende que la red trabaje *en tiempo real*, dando una salida lo más correcta posible a los elementos de la secuencia suministrados en cada instante. Es habitual en este caso no distinguir las fases de entrenamiento y evaluación. Se considera una función de error instantáneo y los pesos se actualizan inmediatamente después de considerar cada elemento.

    Este tipo de entrenamiento se hace especialmente necesario al tratar con entornos no estacionarios en los que las estadísticas de las fuentes de datos cambian con el tiempo.
  - *En línea por secuencias.* En este caso, los pesos se siguen ajustando tras el procesamiento de cada elemento, pero, además, se permite reiniciar el estado de la red en momentos determinados del entrenamiento, normalmente al final de cada secuencia.
- *Entrenamiento fuera de línea.* Los algoritmos pertenecientes a este tipo de entrenamiento pueden a su vez subdividirse en:
  - *Fuera de línea por secuencias.* La actualización de los pesos realizada por el algoritmo se lleva a cabo tras la presentación de cada secuencia. La función de error considera todos los errores

instantáneos cometidos sobre cada uno de los elementos de la secuencia.

– *Fuera de línea por épocas o por lotes.* Los pesos se actualizan una vez presentadas *todas las secuencias*, es decir, únicamente después de ver todo el conjunto de entrenamiento, periodo que se denomina *época*. La función de error considera, por tanto, los errores cometidos sobre todo ese conjunto.

4.2. Predicción numérica con redes recurrentes

La forma de utilizar una RNR para la predicción del siguiente elemento de una secuencia numérica es aparentemente sencilla: la muestra $\boldsymbol{s}[t]$ se introduce en las entradas de la RNR (directamente con su valor o bien con un valor normalizado sobre un determinado rango) y se computa con ella la salida correspondiente $\boldsymbol{y}[t]$. Esta salida se considera como una estimación del valor de la siguiente muestra de la señal, esto es, $\boldsymbol{y}[t] = \hat{\boldsymbol{s}}[t+1]$, con lo que el valor de la salida deseada usado en la función de error es $\boldsymbol{d}[t] = \boldsymbol{s}[t+1]$.

La función de activación de las neuronas de salida debe ajustarse adecuadamente al rango de posibles valores de la señal.

4.3. Predicción simbólica con redes recurrentes

El caso de predicción sobre secuencias simbólicas es un poco más elaborado. Consideremos que tenemos un alfabeto $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ a partir del cual se generan secuencias temporales de la forma $s[1], \ldots, s[t], \ldots, s[L]$. Para predecir el siguiente símbolo de la secuencia con una RNR debemos determinar varias cosas: cómo se representa cada uno de los símbolos de Σ y cómo se realiza el entrenamiento de la red para esta tarea.

La forma más habitual de codificar los distintos símbolos $\sigma_i \in \Sigma$ para su procesamiento por una RNR es la denominada codificación *exclusiva*. En ella, todos los símbolos se codifican mediante vectores unitarios de tamaño $|\Sigma|$, de forma que la representación del símbolo σ_i en un espacio $[0,1]^{|\Sigma|}$ se obtiene a través de la función de codificación:

$$C_\Sigma : \Sigma \longrightarrow [0,1]^{|\Sigma|} \tag{4.2}$$

en la que el j-ésimo componente de la imagen es:

$$(C_\Sigma(\sigma_i))_j = \delta_{i,j} \qquad \sigma_i \in \Sigma,\ j = 1, \ldots, |\Sigma|$$

y donde δ es la función *delta de Kronecker*, definida como:

$$\delta_{i,j} = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{en otro caso} \end{cases} \tag{4.3}$$

Es decir, el símbolo σ_i se representa mediante un vector unitario en el que todos los componentes excepto el i-ésimo son cero. Cada símbolo se representa con el mismo vector durante todo el entrenamiento.

Cuando se entrena una RNR para predecir las probabilidades del siguiente símbolo de una secuencia, en el instante t se alimenta la red con la entrada:

$$\boldsymbol{u}[t] = C_\Sigma(s[t]) \tag{4.4}$$

y la salida obtenida $y_i[t]$ se puede interpretar (como veremos a continuación), después de normalizarla para que todos sus componentes sumen uno, como la probabilidad de que el siguiente símbolo de la secuencia sea σ_i.[2] Para reajustar los pesos de la red, se considera como salida deseada para el algoritmo de entrenamiento supervisado:

$$\boldsymbol{d}[t] = C_\Sigma(s[t+1]) \tag{4.5}$$

Cuando la codificación exclusiva se aplica a las entradas, el número de entradas es $n_U = |\Sigma|$, y puede considerarse que cada símbolo selecciona una determinada dinámica de la red. Al aplicar este tipo de codificación también a las salidas deseadas, el número de neuronas de salida es $n_Y = |\Sigma|$.

4.3.1. Convergencia

A continuación se demuestra que en el caso de que el entorno sea estacionario, el entrenamiento se haga fuera de línea por épocas (es decir, se consideren todos los datos disponibles antes de reestimar los parámetros) y se utilice una función de error cuadrático, el mínimo de esta se produce cuando la salida $y_i[t]$ de la red es la probabilidad condicionada de obtener σ_i después de haber visto todos los símbolos de la secuencia hasta el instante t (Kremer 1997).

En efecto, la contribución al error total debida al símbolo $s[t]$ de una de las secuencias viene dada, si consideramos la función de error cuadrático (4.1), por:

$$\frac{1}{2}\sum_{i=1}^{n_Y}(d_i[t] - y_i[t])^2 \tag{4.6}$$

donde la salida deseada $\boldsymbol{d}[t]$ es la codificación exclusiva del símbolo $s[t+1]$. El error asociado a la neurona de salida i-ésima es, por tanto, $(1-y_i)^2$ si $s[t+1] = \sigma_i$ e y_i^2 en caso contrario.

[2] En las tareas de predicción el alfabeto de entrada y el de salida suele ser el mismo.

Supongamos que N secuencias de la muestra de entrenamiento tienen el prefijo $v = s[1], s[2], \ldots, s[t]$ en común y que de ellas n continúan con el símbolo σ_i y $N-n$ con un símbolo distinto de $\Sigma - \{\sigma_i\}$. Si estas N secuencias comparten el mismo prefijo v, significa que el mismo estado $\boldsymbol{x}[t]$ y, por tanto, la misma salida $\boldsymbol{y}[t]$ serán obtenidos exactamente N veces durante una época en el contexto de v. Entonces, puede considerarse el error acumulado debido al prefijo v como:

$$n(1 - y_i[t])^2 + (N - n)(y_i[t])^2 \tag{4.7}$$

Derivando la ecuación anterior con respecto a $y_i[t]$, obtenemos:

$$-2n(1 - y_i[t]) + 2(N - n)y_i[t] \tag{4.8}$$

La segunda derivada es $2N > 0$. Luego el mínimo se obtiene cuando $y_i[t] = n/N$, es decir, cuando el valor predicho por la red neuronal para el símbolo σ_i tras leer el prefijo v coincide con la frecuencia relativa con que σ_i sigue a v. Un buen algoritmo de entrenamiento debería descubrir este mínimo.

En el caso del aprendizaje en línea se puede constatar empíricamente que las salidas tienden a valores cercanos a las probabilidades reales, aunque la propia naturaleza temporal del entrenamiento no permite demostrar una convergencia eventual como en el caso anterior. Las dos principales suposiciones de la discusión anterior que no se mantienen en el entrenamiento en línea son:

1. Como los pesos se ajustan de forma continua, las N apariciones del prefijo v no resultarán en el mismo estado y en la misma salida de la red.

2. La función de error ya no es global y la existencia de unos mínimos adecuados ya no es demostrable, al menos no de forma tan sencilla como en el caso del aprendizaje fuera de línea.

El entrenamiento en línea puede ser preferible en ocasiones, ya que suele provocar una reducción en el tiempo de aprendizaje con respecto a las estrategias fuera de línea y, además, es idóneo para entornos no estacionarios. Elman (1990) conjeturó que la RRS debería desarrollar representaciones internas de las propiedades temporales de las secuencias de entrada. Aunque así ocurre de hecho, el entrenamiento usado en su artículo era en línea y, como ya hemos comentado, la falta de un modelo teórico completo para este tipo de aprendizaje hace difícil demostrar algún tipo de equivalencia "en el límite" con el aprendizaje fuera de línea.

4.4. Métodos basados en derivadas

Los algoritmos de entrenamiento modifican los parámetros configurables de la red[3] intentando minimizar la medida de error E; es un problema complejo de optimización sin restricciones para el que es necesario aplicar criterios heurísticos.

Los principales algoritmos de entrenamiento se basan en el cálculo del *gradiente* de la función de error, esto es, de la derivada de la función de error con respecto a los distintos parámetros ajustables de la red. Se trata de intentar encontrar el mínimo de la función de error mediante la búsqueda de un punto donde el gradiente se anule.[4]

Una de las variantes basadas en el gradiente más utilizadas es el *descenso por el gradiente*. En él los sucesivos ajustes realizados a los parámetros se hacen de forma individual para cada uno de ellos, digamos W_i, en sentido opuesto al vector de gradiente $\partial E[n]/\partial W_i[n]$:

$$W_i[n+1] = W_i[n] - \alpha \frac{\partial E[n]}{\partial W_i[n]} \tag{4.9}$$

donde α es un parámetro conocido como *tasa de aprendizaje*, que ha de tomar un valor convenientemente pequeño. Al pasar de la iteración[5] n a la $n+1$, el algoritmo aplica la corrección:

$$\Delta W_i[n] = W_i[n+1] - W_i[n] = -\alpha \frac{\partial E[n]}{\partial W_i[n]} \tag{4.10}$$

Puede demostrarse (Haykin 1999) que para valores positivos muy pequeños de la tasa de aprendizaje y funciones de error globales, la formulación del algoritmo de descenso por el gradiente permite que la función de error decrezca en cada iteración. La tasa de aprendizaje α tiene, por tanto, una enorme influencia en la convergencia del método de descenso por el gradiente. Si α es pequeña, el proceso de aprendizaje se desarrolla suavemente, pero la convergencia del sistema a una solución estable puede llevar un tiempo excesivo. Si α es grande, la velocidad de aprendizaje aumenta, pero existe el riesgo de que el proceso de aprendizaje diverja y el sistema se vuelva inestable.

Es habitual añadir un término de *momento* (Plaut et al. 1986; Rumelhart et al. 1986) a (4.10) que en ocasiones puede acelerar el aprendizaje y reducir

[3] Normalmente los pesos de la red, aunque también pueden considerarse otros elementos como el estado inicial $\boldsymbol{x}[0]$ (Bulsari y Saxén 1995; Forcada y Carrasco 1995).

[4] Esta condición es necesaria, pero no suficiente debido a la existencia de mínimos locales, máximos o puntos de silla; de ahí el carácter heurístico del método.

[5] El momento preciso de la actualización de los parámetros depende del carácter en línea o fuera de línea del entrenamiento.

el riesgo de que el algoritmo se vuelva inestable. La nueva ecuación de actualización del parámetro ajustable W_i tiene la forma:

$$\Delta W_i[n] = W_i[n+1] - W_i[n] = -\alpha \frac{\partial E[n]}{\partial W_i[n]} + \gamma \Delta W_i[n-1] \tag{4.11}$$

donde α es la tasa de aprendizaje y γ es la *constante de momento*.

El efecto del momento es el siguiente: si la derivada parcial del error tiene el mismo signo algebraico durante varias iteraciones seguidas (lo que indicaría que se está descendiendo por una "ladera"), el término $\Delta W_i[n]$ irá creciendo y el incremento del parámetro será mayor; si la derivada parcial cambia de signo constantemente (indicación de que el algoritmo se encuentra en una zona complicada), el valor de $\Delta W_i[n]$ se va reduciendo y el parámetro se ajusta de forma más precisa.

Existen otros métodos de optimización más sofisticados (por ejemplo, métodos que consideran la información suministrada por las derivadas de segundo orden[6]), que, en general, proporcionan mejores resultados que el descenso por el gradiente, a veces simplemente con una leve modificación. Algunos de ellos son el método de Newton, el algoritmo de Levenberg-Marquardt o el método de los gradientes conjugados (Shepherd 1997). Todos ellos han sido utilizados abundantemente sobre redes no recurrentes y escasamente con redes recurrentes (Chang y Mak 1999; Chan y Szeto 1999).

A continuación veremos tres algoritmos de entrenamiento que se basan en el cálculo del gradiente. Los dos primeros, el aprendizaje recurrente en tiempo real y la retropropagación a través del tiempo, de hecho, usan el descenso por el gradiente y se pueden considerar más bien como formas distintas de calcular el valor de la derivada correspondiente. El tercero de ellos, el filtro de Kalman extendido desacoplado, usa de un modo más elaborado que el descenso por el gradiente las derivadas de la función de error, calculadas de cualquiera de las dos formas anteriores.

4.5. Aprendizaje recurrente en tiempo real

Como ya se ha dicho al final del apartado anterior, el *aprendizaje recurrente en tiempo real* (RTRL, por el inglés *real-time recurrent learning*) (Williams y Zipser 1989) se considerará aquí como una forma de calcular las derivadas parciales de la función de error, aunque algunos autores se refieren a él como un algoritmo de entrenamiento *per se* al combinarlo con el ajuste de pesos realizado con el descenso por el gradiente.

[6]En cualquier caso, los métodos de segundo orden no evitan el problema de los mínimos locales.

Con un ejemplo es más sencillo entender la forma de calcular las derivadas en RTRL.

4.5.1. Ejemplo de cálculo de las derivadas del error

En este apartado se derivarán las ecuaciones de RTRL para una RRS, cuya dinámica viene definida por las ecuaciones (3.5) a (3.8). La derivación de las ecuaciones para otros tipos de redes recurrentes suele ser muy similar a la de la RRS.

Consideremos una función de error cuadrático como la de (4.1). Aplicando la regla de la cadena y considerando un parámetro ajustable cualquiera, se tiene que:

$$\frac{\partial E[t]}{\partial \square} = -\sum_{l=1}^{n_Y} (d_l[t] - y_l[t]) \frac{\partial y_l[t]}{\partial \square} \tag{4.12}$$

En lo anterior, la derivada $\partial y_l[t]/\partial \square$ depende del parámetro concreto considerado. A continuación se dan las expresiones de estas derivadas para todos los pesos y sesgos de la red:[7]

$$\frac{\partial y_l[t]}{\partial W_i^y} = g_Y'(Y_l[t])\, \delta_{l,i} \tag{4.13}$$

$$\frac{\partial y_l[t]}{\partial W_{i,j}^{y,x}} = g_Y'(Y_l[t])\, x_j[t] \delta_{l,i} \tag{4.14}$$

$$\frac{\partial y_l[t]}{\partial W_j^x} = g_Y'(Y_l[t]) \sum_{i=1}^{n_X} W_{l,i}^{y,x} \frac{\partial x_i[t]}{\partial W_j^x} \tag{4.15}$$

$$\frac{\partial y_l[t]}{\partial W_{j,k}^{x,u}} = g_Y'(Y_l[t]) \sum_{i=1}^{n_X} W_{l,i}^{y,x} \frac{\partial x_i[t]}{\partial W_{j,k}^{x,u}} \tag{4.16}$$

$$\frac{\partial y_l[t]}{\partial W_{j,k}^{x,x}} = g_Y'(Y_l[t]) \sum_{i=1}^{n_X} W_{l,i}^{y,x} \frac{\partial x_i[t]}{\partial W_{j,k}^{x,x}} \tag{4.17}$$

[7] La derivada de la función logística es $g_L(x)(1 - g_L(x))$. La derivada de la función tangente hiperbólica es $1 - g_T^2(x)$.

Para la derivación de las ecuaciones anteriores debe tenerse en cuenta las siguientes expresiones:

$$\frac{\partial W_i^y}{\partial W_j^y} = \delta_{i,j} \tag{4.18}$$

$$\frac{\partial W_{i,j}^{y,x}}{\partial W_{k,l}^{y,x}} = \delta_{i,k}\,\delta_{j,l} \tag{4.19}$$

$$\frac{\partial W_i^x}{\partial W_j^x} = \delta_{i,j} \tag{4.20}$$

$$\frac{\partial W_{i,j}^{x,u}}{\partial W_{k,l}^{x,u}} = \delta_{i,k}\,\delta_{j,l} \tag{4.21}$$

$$\frac{\partial W_{i,j}^{x,x}}{\partial W_{k,l}^{x,x}} = \delta_{i,k}\,\delta_{j,l} \tag{4.22}$$

donde la función $\delta_{i,j}$ es la *delta de Kronecker*, ya definida en (4.3).

Las derivadas del estado $x_i[t]$ de las ecuaciones (4.15) a (4.17) son recurrentes en RTRL como resultado de la propia recurrencia de la red:

$$\frac{\partial x_i[t]}{\partial W_j^x} = g_X'(X_i[t]) \left(\delta_{i,j} + \sum_{k=1}^{n_X} W_{i,k}^{x,x} \frac{\partial x_k[t-1]}{\partial W_j^x} \right) \tag{4.23}$$

$$\frac{\partial x_i[t]}{\partial W_{j,k}^{x,u}} = g_X'(X_i[t]) \left(u_k[t]\delta_{i,j} + \sum_{m=1}^{n_X} W_{i,m}^{x,x} \frac{\partial x_m[t-1]}{\partial W_{j,k}^{x,u}} \right) \tag{4.24}$$

$$\frac{\partial x_i[t]}{\partial W_{j,k}^{x,x}} = g_X'(X_i[t]) \left(x_k[t-1]\delta_{i,j} + \sum_{m=1}^{n_X} W_{i,m}^{x,x} \frac{\partial x_m[t-1]}{\partial W_{j,k}^{x,x}} \right) \tag{4.25}$$

La implementación de un algoritmo de descenso por el gradiente a partir de estas ecuaciones es sencilla.

4.6. Retropropagación en el tiempo

Al igual que hicimos con RTRL, consideraremos la retropropagación a través del tiempo (BPTT, por el inglés *backpropagation through time*) (Rumelhart et al. 1986; Williams y Peng 1990) como una forma de calcular las derivadas parciales de la función de error con respecto a los parámetros ajustables de la red, aunque hay autores que denominan BPTT a la combinación de lo anterior con el descenso por el gradiente.

Al calcular las derivadas parciales en BPTT se asume que el comportamiento temporal de la RNR puede ser *desplegado* en el espacio en forma de

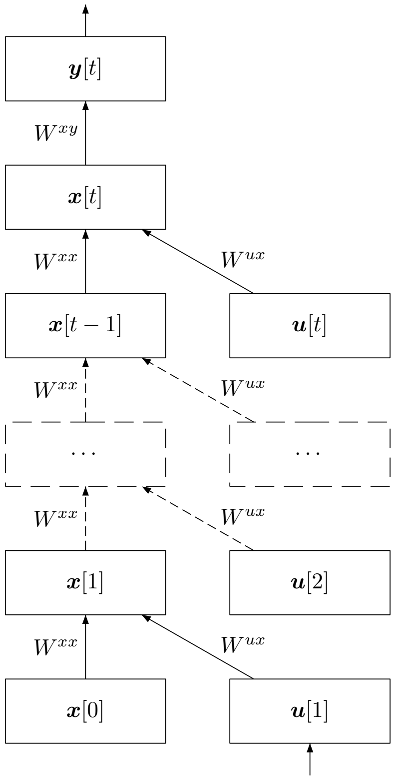


Figura 4.1: Una RRS desplegada en el instante t según BPTT.

red hacia adelante. Es posible aplicar entonces el conocido algoritmo de retropropagación (Rumelhart et al. 1986) para calcular las derivadas parciales de este tipo de redes.

El despliegue de la RNR hace que la red hacia adelante (*red extendida*) vaya creciendo una y otra vez tras cada instante de tiempo. Así, suponiendo una RRS, las unidades de entrada y las unidades de estado del instante t se convierten en dos nuevas capas en la red extendida; las unidades de entrada y las unidades ocultas del instante $t-1$ se convierten también en dos nuevas capas de la red extendida; y así sucesivamente hasta llegar al primer instante de tiempo. Como realmente solo existe un conjunto de unidades de entrada y de unidades ocultas, los pesos equivalentes en las distintas capas virtuales han de tener idéntico valor. El algoritmo de retropropagación permite obtener la contribución al error total de cada una de las versiones de los pesos, pero a la hora de actualizarlos debe considerarse la suma de las contribuciones de los pesos equivalentes. En la figura 4.1 se muestra la red desplegada utilizada en el instante t.

Cuando el entrenamiento es por secuencias, el tamaño de cada secuencia determina el de la red extendida; en el caso de una secuencia de longitud relativamente extensa, las necesidades temporales y espaciales del algoritmo crecerían linealmente conforme la red fuera procesando las entradas. Por ello, en estos casos, la historia de la red se *trunca* y se considera irrelevante cualquier información anterior a t_0 instantes de tiempo. El valor t_0 se conoce como *umbral de truncamiento* y la técnica resultante como BPTT truncada.

4.6.1. Ejemplo de cálculo de las derivadas del error

En este apartado se deriva las ecuaciones de BPTT para una RRS con la dinámica definida por las ecuaciones (3.5) a (3.8), y que desplegada en el tiempo tiene el aspecto de la figura 4.1. La red neuronal de esa figura es una red no recurrente con lo que las derivadas de la función de error serán las mismas que las calculadas con la técnica de retropropagación (Rumelhart et al. 1986), de la que no mostraremos aquí los detalles. Si se utiliza el descenso por el gradiente, el algoritmo se limita a actualizar cada peso (no se muestran las ecuaciones de los sesgos) mediante la llamada *regla delta generalizada* como sigue:

$$\Delta W_{i,j}^{y,x}[t] = \alpha\, \delta_i^Y[t]\, x_j[t] \tag{4.26}$$

$$\Delta W_{i,j}^{x,x}[t] = \alpha \sum_{\tau=1}^{t} \delta_i^X[\tau]\, x_j[\tau-1] \tag{4.27}$$

$$\Delta W_{i,j}^{x,u}[t] = \alpha \sum_{\tau=1}^{t} \delta_i^X[\tau]\, u_j[\tau] \tag{4.28}$$

donde la *señal de error* δ^Y y *la señal de error retropropagada* δ^X se definen a partir de:

$$\delta_i^Y[t] = \frac{\partial E[t]}{\partial Y_i[t]} \tag{4.29}$$

$$\delta_i^X[t] = g_X'(X_i[t]) \sum_{j=1}^{n_Y} \delta_j^Y[t]\, W_{j,i}^{y,x} \tag{4.30}$$

y para $1 \leq \tau < t$,

$$\delta_i^X[\tau] = g_X'(X_i[\tau]) \sum_{j=1}^{n_X} \delta_j^X[\tau+1]\, W_{j,i}^{x,x} \tag{4.31}$$

La señal de error retropropagada puede verse como un emisario del presente t que viaja hacia atrás en el tiempo para influir en el pasado de forma que este contribuya en mayor medida a los acontecimientos que se desearía haber observado en el presente.

4.7. Filtros de Kalman

El filtro de Kalman (Kalman 1960) sirve para estimar el estado de un determinado sistema dinámico lineal cuyo modelo no es completamente conocido y al que se accede a través de un proceso de medición que también posee un cierto nivel de ruido. El filtro permite utilizar la información incompleta del modelo para mejorar de forma recursiva la estimación del estado del sistema proporcionada por la medición. Por otro lado, el filtro de Kalman extendido es una adaptación del anterior que permite trabajar sobre sistemas no lineales.

El filtro de Kalman ha sido objeto de un gran número de investigaciones y aplicaciones en multitud de áreas como la navegación marítima, la instrumentación en centrales nucleares o la visión artificial. Aquí nos centraremos en su aplicación al entrenamiento supervisado de redes neuronales.

Normalmente, los algoritmos de descenso por el gradiente son menos rápidos de lo deseado debido a que solo utilizan la última estimación del gradiente: las derivadas de la función de error solo toman en cuenta la distancia entre la salida actual y la correspondiente salida deseada sin usar a la hora de actualizar los parámetros ninguna información sobre la historia anterior del entrenamiento.

El filtro de Kalman extendido desacoplado (Puskorius y Feldkamp 1994; Haykin 1999) se basa en el filtro de Kalman extendido para superar la limitación anterior y considerar el entrenamiento como un problema de filtrado óptimo en el que se encuentra recursivamente una solución al problema de los mínimos cuadrados.[8] En todo momento se utiliza toda la información suministrada a la red hasta el instante actual, incluidas todas las derivadas calculadas desde la primera iteración del proceso de aprendizaje. Sin embargo, el algoritmo funciona de tal modo que solo es necesario almacenar explícitamente los resultados de la última iteración.

[8]El problema de los mínimos cuadrados consiste en encontrar la curva que mejor aproxima un conjunto de datos determinado de manera que se minimice la distancia media entre los datos y la curva.

4.7.1. El filtro de Kalman

El filtro de Kalman (1960) (FK) intenta estimar el estado[9] $\boldsymbol{w}[t] \in \mathbb{R}^n$ de un sistema dinámico lineal de tiempo discreto gobernado por la ecuación:

$$\boldsymbol{w}[t+1] = A\boldsymbol{w}[t] + B\boldsymbol{u}[t] + \boldsymbol{\omega}[t] \tag{4.32}$$

donde $\boldsymbol{u}[t]$ es la entrada del sistema, con una medición $\boldsymbol{d}[t] \in \mathbb{R}^m$ que es:

$$\boldsymbol{d}[t] = H\boldsymbol{w}[t] + \boldsymbol{\nu}[t] \tag{4.33}$$

donde A, B y H son conocidas. Las variables aleatorias $\boldsymbol{\omega}[t]$ y $\boldsymbol{\nu}[t]$ representan el ruido del proceso y de la medición, respectivamente. Se asume que se trata de ruido blanco[10] de media cero y con matrices de covarianza diagonales $Q[t]$ y $R[t]$:

$$\langle \boldsymbol{\omega}[t]\, \boldsymbol{\omega}^T[t] \rangle = Q[t] \tag{4.34}$$

$$\langle \boldsymbol{\nu}[t]\, \boldsymbol{\nu}^T[t] \rangle = R[t] \tag{4.35}$$

En cada paso el filtro proyecta la estimación del estado actual y de la covarianza actual hacia adelante en el tiempo para obtener una estimación *a priori* para el siguiente paso. Después utiliza los resultados de la medición real para mejorar esta estimación y obtener una estimación *a posteriori*. Este proceso también puede verse como un ciclo de predicción y corrección.

Sea $\hat{\boldsymbol{w}}^-[t]$ la estimación *a priori* del estado en el instante t a partir del conocimiento anterior al paso t:

$$\hat{\boldsymbol{w}}^-[t] = A\hat{\boldsymbol{w}}[t-1] + B\boldsymbol{u}[t-1] \tag{4.36}$$

La estimación *a posteriori* del estado, $\hat{\boldsymbol{w}}[t]$, se obtiene como una combinación lineal de la estimación *a priori* $\hat{\boldsymbol{w}}^-[t]$ y la diferencia ponderada entre la medición real $\boldsymbol{d}[t]$ y una predicción de la medida $H\hat{\boldsymbol{w}}^-[t]$:

$$\hat{\boldsymbol{w}}[t] = \hat{\boldsymbol{w}}^-[t] + K[t](\boldsymbol{d}[t] - H\hat{\boldsymbol{w}}^-[t]) \tag{4.37}$$

La expresión $(\boldsymbol{d}[t] - H\hat{\boldsymbol{w}}^-[t])$ se denomina *residuo* o *innovación de la medida* y refleja la discrepancia entre la medición predicha y la real.

[9] Es importante destacar que la noción de estado utilizada en este apéndice es diferente a la del resto del trabajo (normalmente, como los valores de activación de las unidades recurrentes de una RNR); por ello, para evitar confusiones, la notación utilizada en este apéndice para el estado es $\boldsymbol{w}[t]$ y no $\boldsymbol{x}[t]$ como es habitual en la bibliografía sobre el tema.

[10] El espectro del ruido blanco es continuo y uniforme sobre una determinada banda de frecuencia.

Consideremos ahora los errores de la estimación *a priori* y de la estimación *a posteriori*:

$$\boldsymbol{e}^-[t] = \boldsymbol{w}[t] - \hat{\boldsymbol{w}}^-[t] \tag{4.38}$$

$$\boldsymbol{e}[t] = \boldsymbol{w}[t] - \hat{\boldsymbol{w}}[t] \tag{4.39}$$

La covarianza *a priori* del error de estimación es:

$$\left\langle \boldsymbol{e}^-[t]\,(\boldsymbol{e}^-[t])^T \right\rangle = P^-[t] \tag{4.40}$$

y *a posteriori*:

$$\left\langle \boldsymbol{e}[t]\,(\boldsymbol{e}[t])^T \right\rangle = P[t] \tag{4.41}$$

La matriz de ganancia K se elige de manera que se minimice la covarianza del error *a posteriori* (4.41). Una posibilidad es:

$$K[t] = P^-[t]H^T\big(HP^-[t]H^T + R[t]\big)^{-1} \tag{4.42}$$

Debido al ruido, la ecuación (4.36) tiene asociada una covarianza del error *a priori* que se calcula mediante:

$$P^-[t] = AP[t-1]A^T + Q[t] \tag{4.43}$$

La covarianza del error *a posteriori* se obtiene de:

$$P[t] = (I - K[t]H)P^-[t] \tag{4.44}$$

Un ciclo del algoritmo consiste en evaluar, por este orden, las ecuaciones (4.36), (4.43), (4.42), (4.37) y (4.44). La naturaleza recursiva del filtro hace que la estimación del estado del sistema esté en función de todas las mediciones del pasado pero sin tenerlas que considerar explícitamente.

El rendimiento del filtro puede mejorarse mediante el control de las matrices $Q[t]$ y $R[t]$. Estas matrices pueden fijarse antes del funcionamiento del filtro o pueden ir cambiándose dinámicamente. Así, $R[t]$ tendrá que ser ajustada en función de nuestra confianza en el mecanismo responsable de la medición. Por otra parte, con $Q[t]$ se puede modelizar nuestra incertidumbre en el modelo (4.32); a veces, un modelo aproximado o incluso alejado del real puede ser útil si se introduce suficiente ruido en la matriz $Q[t]$.

4.7.2. El filtro de Kalman extendido

Normalmente, el proceso a estimar o la ecuación de medición son no lineales. Debe hacerse, por tanto, una aproximación a este caso. Un filtro de Kalman que linealiza en torno a la media y a la covarianza actual se denomina un filtro de Kalman extendido (FKE).[11]

Ahora el proceso vuelve a tener un vector de estado $\boldsymbol{w}[t] \in \mathbb{R}^n$, pero la ecuación que lo gobierna es:

$$\boldsymbol{w}[t+1] = \boldsymbol{f}(\boldsymbol{w}[t], \boldsymbol{u}[t]) + \boldsymbol{\omega}[t] \tag{4.45}$$

con una medición $\boldsymbol{d} \in \mathbb{R}^m$ que es:

$$\boldsymbol{d}[t] = \boldsymbol{h}(\boldsymbol{w}[t]) + \boldsymbol{\nu}[t] \tag{4.46}$$

donde las variables aleatorias $\boldsymbol{\omega}[t]$ y $\boldsymbol{\nu}[t]$ representan, como antes, el ruido de media cero del proceso y de la medida, respectivamente (con matrices de covarianza $Q[t]$ y $R[t]$). Las funciones $\boldsymbol{f}$ y $\boldsymbol{h}$ son funciones no lineales que relacionan el estado en el instante t con el estado en el instante $t+1$ y con la medición $\boldsymbol{d}[t]$, respectivamente.

Mediante la linealización de las ecuaciones de estado y de medición se llega a una serie de ecuaciones equivalentes a las del caso lineal (véase, por ejemplo, el texto de Welch y Bishop (2002) para más detalles). Así, la estimación *a priori* del estado (4.36) se aproxima ahora haciendo:

$$\hat{\boldsymbol{w}}^-[t] = \boldsymbol{f}(\hat{\boldsymbol{w}}[t-1], \boldsymbol{u}[t-1]) \tag{4.47}$$

y la covarianza del error *a priori* (4.43) se calcula con:

$$P^-[t] = A[t-1]P[t-1](A[t-1])^T + W[t-1]Q[t-1](W[t-1])^T \tag{4.48}$$

donde A y W son matrices nuevas. La matriz A se define ahora como la matriz de derivadas parciales (jacobiano) de $\boldsymbol{f}$ respecto al estado:

$$A[t] = \begin{pmatrix} \frac{\partial f_1}{\partial w_1}[t] & \frac{\partial f_1}{\partial w_2}[t] & \cdots & \frac{\partial f_1}{\partial w_n}[t] \\ \frac{\partial f_2}{\partial w_1}[t] & \frac{\partial f_2}{\partial w_2}[t] & \cdots & \frac{\partial f_2}{\partial w_n}[t] \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial w_1}[t] & \frac{\partial f_n}{\partial w_2}[t] & \cdots & \frac{\partial f_n}{\partial w_n}[t] \end{pmatrix} \tag{4.49}$$

[11] Existen otras propuestas para aplicar el FK a sistemas no lineales distintas a la que mostraremos en este apartado (Julier y Uhlmann 1997).

donde se ha abreviado $f_n[t]$ por $f_n(\hat{\boldsymbol{w}}[t], \boldsymbol{u}[t])$. Por otra parte, W es la matriz de derivadas parciales de $\boldsymbol{f}$ respecto al ruido $\boldsymbol{\omega}$:

$$W[t] = \begin{pmatrix} \frac{\partial f_1}{\partial \omega_1}[t] & \frac{\partial f_1}{\partial \omega_2}[t] & \cdots & \frac{\partial f_1}{\partial \omega_n}[t] \\ \frac{\partial f_2}{\partial \omega_1}[t] & \frac{\partial f_2}{\partial \omega_2}[t] & \cdots & \frac{\partial f_2}{\partial \omega_n}[t] \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial \omega_1}[t] & \frac{\partial f_n}{\partial \omega_2}[t] & \cdots & \frac{\partial f_n}{\partial \omega_n}[t] \end{pmatrix} \tag{4.50}$$

donde de nuevo se ha escrito $f_n[t]$ por $f_n(\hat{\boldsymbol{w}}[t], \boldsymbol{u}[t])$.

La matriz de ganancia (4.42) se obtiene en el filtro extendido a partir de la ecuación:

$$K[t] = P^-[t](H[t])^T \left(H[t]P^-[t](H[t])^T + V[t]R[t](V[t])^T\right)^{-1} \tag{4.51}$$

donde H es aquí el jacobiano de las derivadas parciales de $\boldsymbol{h}$ respecto al estado:

$$H[t] = \begin{pmatrix} \frac{\partial h_1}{\partial w_1}[t] & \frac{\partial h_1}{\partial w_2}[t] & \cdots & \frac{\partial h_1}{\partial w_n}[t] \\ \frac{\partial h_2}{\partial w_1}[t] & \frac{\partial h_2}{\partial w_2}[t] & \cdots & \frac{\partial h_2}{\partial w_n}[t] \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial h_m}{\partial w_1}[t] & \frac{\partial h_m}{\partial w_2}[t] & \cdots & \frac{\partial h_m}{\partial w_n}[t] \end{pmatrix} \tag{4.52}$$

donde se ha abreviado $h_m[t]$ por $h_m(\hat{\boldsymbol{w}}^-[t])$, y V es la matriz de derivadas parciales de $\boldsymbol{f}$ respecto a $\boldsymbol{\nu}$:

$$V[t] = \begin{pmatrix} \frac{\partial h_1}{\partial \nu_1}[t] & \frac{\partial h_1}{\partial \nu_2}[t] & \cdots & \frac{\partial h_1}{\partial \nu_m}[t] \\ \frac{\partial h_2}{\partial \nu_1}[t] & \frac{\partial h_2}{\partial \nu_2}[t] & \cdots & \frac{\partial h_2}{\partial \nu_m}[t] \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial h_m}{\partial \nu_1}[t] & \frac{\partial h_m}{\partial \nu_2}[t] & \cdots & \frac{\partial h_m}{\partial \nu_m}[t] \end{pmatrix} \tag{4.53}$$

donde otra vez se ha escrito $h_m[t]$ por $h_m(\hat{\boldsymbol{w}}^-[t])$.

La estimación *a posteriori* del estado (4.37) utiliza también aquí K para ponderar la diferencia entre la medición real y una predicción de la medida:

$$\hat{\boldsymbol{w}}[t] = \hat{\boldsymbol{w}}^-[t] + K[t](\boldsymbol{d}[t] - \boldsymbol{h}(\hat{\boldsymbol{w}}^-[t])) \tag{4.54}$$

Finalmente, la covarianza del error tiene una forma similar a la del caso lineal (4.44), aunque debe tenerse en cuenta que ahora $H[t]$ se calcula de manera diferente al caso lineal:

$$P[t] = (I - K[t]H[t])P^-[t] \tag{4.55}$$

La operación básica del FKE se puede resumir en los siguientes pasos. En primer lugar, se proyectan las estimaciones del estado y de la covarianza del error del instante t al $t+1$ haciendo uso de las ecuaciones (4.47) y (4.48). Después ha de hacerse uso de estas nuevas estimaciones *a priori* para obtener unas corregidas al considerar la medición $\boldsymbol{d}[t]$. Las ecuaciones (4.51), (4.54) y (4.55), por este orden, nos dan las estimaciones *a posteriori* del estado y de la covarianza del error. De nuevo, $R[t]$ y $Q[t]$ son parámetros ajustables del algoritmo.

4.8. Entrenamiento de una red neuronal con el filtro de Kalman

Para poder aplicar el FK al entrenamiento de redes neuronales (recurrentes o no), el aprendizaje se considera como un problema de filtrado en el que los parámetros óptimos de la red se estiman de forma recursiva a partir de las ecuaciones del filtro (Puskorius y Feldkamp 1991; Haykin 2001). El algoritmo es especialmente idóneo para situaciones de aprendizaje en línea, en las que los pesos se ajustan continuamente, aunque también puede aplicarse al procesamiento fuera de línea (Feldkamp y Puskorius 1994).

Comencemos considerando que el estado de la red, que denotaremos por $\boldsymbol{w}[t]$, viene dado por *los valores de sus pesos*.[12]

La ecuación que describe el sistema es lineal y sigue la ecuación (4.32) con $A = I$, $B = 0$ y $\boldsymbol{\omega} = 0$ (esta última igualdad la reconsideraremos más adelante):

$$\boldsymbol{w}[t+1] = \boldsymbol{w}[t] \tag{4.56}$$

Esta ecuación asume que el sistema se encuentra en un estado *óptimo* y estable, por tanto. Este estado puede corresponder a un mínimo local o global de la superficie de error.

[12] El estado de las redes neuronales recurrentes de tiempo discreto $\boldsymbol{x}[t]$ no se representa explícitamente en las ecuaciones del filtro de Kalman.

La medida es la salida deseada de la red neuronal. Se trata, por tanto, de una ecuación no lineal como la ecuación (4.46) con la forma:

$$\boldsymbol{d}[t] = \boldsymbol{y}[t] + \boldsymbol{\nu}[t] \tag{4.57}$$

donde $\boldsymbol{y}[t]$ es la salida de la red cuando se aplica a sus entradas $\boldsymbol{u}[t]$, esto es, la no linealidad global de la red $\boldsymbol{y}[t] = \boldsymbol{h}(\boldsymbol{w}[t])$.

Debido a que la ecuación de estado es lineal, el filtro de Kalman utilizará las ecuaciones (4.36) y (4.43) con $A = I$, $B = 0$ y $Q[t] = 0$. La no linealidad de la ecuación (4.57) añade las restantes ecuaciones del filtro: las ecuaciones (4.51), (4.54) y (4.55) con $\boldsymbol{h}$ igual a la salida de la red $\boldsymbol{y}[t]$ cuando utiliza los pesos $\boldsymbol{w}[t]$.

El jacobiano $V[t]$ de la ecuación (4.51) se hace normalmente igual a la matriz unidad, $V[t] = I$, ante la dificultad de una estimación correcta de su valor. Se asume, entonces, que su influencia está de alguna manera "oculta" dentro de $R[t]$.

El valor de la medida real de la ecuación (4.54) es el valor de la salida deseada de la red. Las derivadas parciales de la matriz $H[t]$ se calculan normalmente mediante BPTT o RTRL.

Con todo lo anterior, ya tendríamos una versión del denominado *filtro de Kalman extendido global* (FKEG). No obstante, para su utilización real como algoritmo de entrenamiento de redes neuronales es aconsejable la introducción de algunas modificaciones.

4.8.1. El filtro de Kalman extendido desacoplado

Cuando se trabaja con redes de cierto tamaño, el vector de estado $\boldsymbol{w}[t]$ puede tener un número considerable de componentes (tantos como pesos tenga la red). Ello ocasiona que los cálculos sobre matrices como $H[t]$ requieran una cantidad elevada de recursos computacionales, incluso para redes de tamaño moderado. El *filtro de Kalman extendido desacoplado* (FKED) reduce esta complejidad (Puskorius y Feldkamp 1991; Haykin 2001).

El FKED divide los pesos de la red en g grupos, $\boldsymbol{w}_i$, $i = 1, \ldots, g$, para lograr que el problema sea computacionalmente tratable. Habrá tantos grupos como neuronas en la red[13] y dos pesos pertenecerán al mismo grupo si forman parte de la entrada de una misma neurona. La versión desacoplada, por lo tanto, aplica el filtro de Kalman extendido a cada neurona independientemente para estimar el valor óptimo de los pesos que llegan a

[13] En el caso de la red LSTM, se considera un grupo para cada neurona, celda de memoria y compuerta diferente, con lo que resulta $g = n_M(n_C + 3) + n_Y$, si se usan compuertas de olvido.

ella. De esta forma, solo se consideran las interdependencias locales durante el entrenamiento.

La principal diferencia entre la versión desacoplada y la global es la sustitución de la matriz $H[t]$ de la ecuación (4.52) por g matrices de la forma:

$$H_i[t] = \begin{pmatrix} \frac{\partial y_1}{\partial w_1^{(i)}}[t] & \frac{\partial y_1}{\partial w_2^{(i)}}[t] & \cdots & \frac{\partial y_1}{\partial w_n^{(i)}}[t] \\ \frac{\partial y_2}{\partial w_1^{(i)}}[t] & \frac{\partial y_2}{\partial w_2^{(i)}}[t] & \cdots & \frac{\partial y_2}{\partial w_n^{(i)}}[t] \\ \multicolumn{4}{c}{\dotfill} \\ \frac{\partial y_{n_Y}}{\partial w_1^{(i)}}[t] & \frac{\partial y_{n_Y}}{\partial w_2^{(i)}}[t] & \cdots & \frac{\partial y_{n_Y}}{\partial w_n^{(i)}}[t] \end{pmatrix} \tag{4.58}$$

con $i = 1, \ldots, g$, y donde $w_j^{(i)}$ es el j-ésimo peso del grupo i. Se ha supuesto que n es el número de pesos del grupo i. Con esto, la matriz $H[t]$ es simplemente la *concatenación* de las matrices $H_i[t]$:

$$H[t] = (H_1[t], H_2[t], \ldots, H_g[t]) \tag{4.59}$$

Es fácil observar que el filtro de Kalman extendido desacoplado se reduce al global cuando $g = 1$.

A continuación se muestra el algoritmo de la versión desacoplada del filtro de Kalman extendido:[14]

1. Hacer g igual al número de neuronas de la red.

2. Iniciar los pesos de la red, $\hat{\boldsymbol{w}}_i[0]$, $i = 1, \ldots, g$.

3. Iniciar los elementos de la diagonal de $R[0]$ y $P_i[0]$.

[14] Para el cálculo de la matriz inversa en los experimentos de esta tesis se utilizó la versión de 23 agosto de 1999 de `newmat10`, desarrollada por Davies (1994). La rutina para el cálculo de la inversa se basa en la que aparece en el libro de Press et al. (1988).

4. Para $t = 1, 2, \ldots$ e $i = 1, \ldots, g$, calcular las siguientes ecuaciones:

$$\hat{\boldsymbol{w}}_i^-[t] = \hat{\boldsymbol{w}}_i[t-1] \tag{4.60}$$

$$P_i^-[t] = P_i[t-1] \tag{4.61}$$

$$K_i[t] = P_i^-[t](H_i[t])^T \left(\sum_{j=1}^{g} H_j[t]P_j^-[t](H_j[t])^T + R[t] \right)^{-1} \tag{4.62}$$

$$\hat{\boldsymbol{w}}_i[t] = \hat{\boldsymbol{w}}_i^- + K_i[t](\boldsymbol{d}[t] - \boldsymbol{y}[t]) \tag{4.63}$$

$$P_i[t] = (I - K_i[t]H_i[t])P_i^-[t] \tag{4.64}$$

donde $\boldsymbol{d}[t]$ es la salida deseada de la red en el instante t e $\boldsymbol{y}[t]$ es la salida real de la red para la entrada $\boldsymbol{u}[t]$. Nótese que las dos primeras ecuaciones son innecesarias en la implementación del algoritmo y puede trabajarse con las estimaciones *a posteriori* del instante anterior directamente.

5. Actualizar $R[t]$.

4.8.2. Control de la divergencia del filtro

La forma no lineal del FKED provoca numerosas dificultades numéricas a la hora de su implementación, que hacen que el filtro diverja de la solución correcta.

Una forma heurística de evitar esta divergencia (Haykin 1999, p. 769) es añadir ruido a la ecuación del proceso, haciendo que $\boldsymbol{\omega}[t] \neq 0$. El único cambio sobre la forma descrita anteriormente para el FKED es la ecuación (4.64) que se convierte en:

$$P_i[t] = (I - K_i[t]H_i[t])\, P_i^-[t] + Q_i[t] \tag{4.65}$$

Además de evitar la divergencia, la introducción de $Q_i[t]$ tiene el efecto secundario de hacer que el algoritmo tenga menor propensión a quedarse atrapado en mínimos locales. Normalmente se usa la misma matriz para todos los grupos, por lo que hablaremos simplemente de $Q[t]$. A continuación se indican algunas pautas para dar valores a los parámetros iniciales del FKED.

4.8.3. Parámetros iniciales del algoritmo

Los parámetros a ajustar en la inicialización del FKED son:

- El valor inicial de la covarianza del error *a posteriori* $P_i[0]$; este valor se suele inicializar como $P_i[0] = \delta I$, donde δ es una constante positiva.
- Los elementos diagonales de las matriz inicial de covarianza del ruido de la medida $R[0]$; estos elementos se templan normalmente, como se verá más abajo, desde un valor inicial a valores más bajos conforme avanza el entrenamiento.[15]
- Los elementos de la diagonal de la matriz de covarianza inicial del error del proceso $Q[0]$; estos valores también se templan.

El templado de los elementos de las matrices de covarianza pasa por darles un valor o temperatura inicial e ir decrementándolo paulatinamente según una determinada *tasa de reducción de la temperatura* T. Por ejemplo, consideremos que los valores de la diagonal de la matriz $R[0]$ se inicializan con un valor $R_{\text{máx}}$ y se templan con una tasa T hasta alcanzar el valor $R_{\text{mín}}$, lo que expresaremos con la notación:

$$R[t] : R_{\text{máx}} \xrightarrow{T} R_{\text{mín}}$$

La ecuación que se aplicará en esta tesis para obtener la evolución de los parámetros es:

$$R[t] = \frac{R_{\text{máx}} - R_{\text{mín}}}{\exp(t/T)} + R_{\text{mín}} \tag{4.66}$$

La figura 4.2 muestra cómo influye la tasa de reducción de la temperatura T en la evolución de $R[t]$.

4.9. Coste computacional

La complejidad del algoritmo de descenso por gradiente viene determinada principalmente por la complejidad de los esquemas de cálculo de las derivadas de la función de error empleados.

La complejidad temporal de BPTT cuando se aplica a redes de primer orden como las del epígrafe 3.1.1 con $n_X \geq n_U$ y $n_X \geq n_Y$ es $O(n_X^2)$,[16] esto es, una complejidad temporal asintótica de la misma familia que la de los cálculos que determinan el valor de la salida de la RNR a partir de la

[15] Podría pensarse que el templado no es compatible con el aprendizaje en línea. Aun así, si el entorno es estacionario o semiestacionario, el templado puede usarse para reducir nuestra incertidumbre sobre él conforme se avanza en el procesamiento de la entrada. El templado debe descartarse o se debe incrementar los valores inferiores cuando el entorno es no estacionario.

[16] La ecuación (4.31) se implementa en un bucle con $i = 1, \ldots, n_X$.

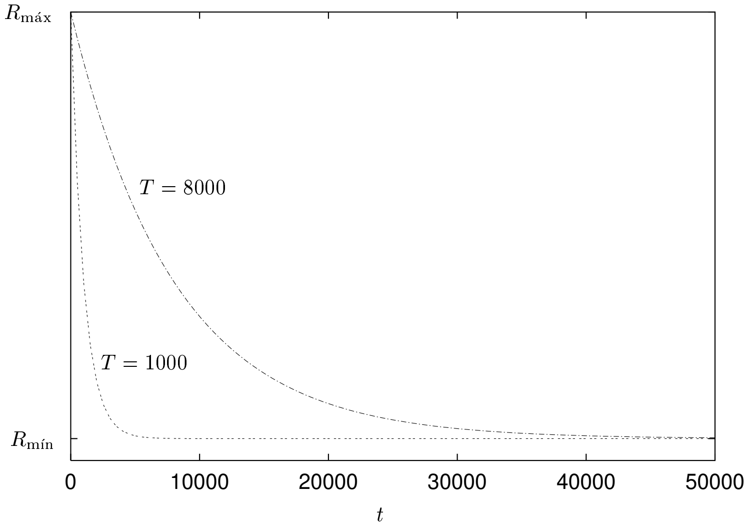


Figura 4.2: Evolución de la matriz de covarianza del error del FKED según la tasa T.

entrada actual y del estado de la red. Esta es una de las grandes ventajas de BPTT. El coste espacial, sin embargo, es uno de sus grandes inconvenientes, ya que es necesario almacenar varias réplicas de la red; además, en el caso de BPTT no truncada, debe guardarse todas las entradas procesadas hasta el instante actual.

Aunque la complejidad espacial al calcular las derivadas con RTRL es mucho menor que la de BPTT, su complejidad temporal es muy superior. Cuando se aplica RTRL a RNR de primer orden con $n_X \geq n_U$ y $n_X \geq n_Y$, la complejidad temporal es $O(n_X^4)$.[17]

Muchos autores, por ejemplo Schmidhuber (1992), han propuesto algoritmos híbridos que combinan eficientemente lo mejor de ambos algoritmos.

Finalmente, el FKED se apoya en el cálculo de la derivada del error, que puede realizarse tanto con BPTT como con RTRL. El coste es, por tanto, como mínimo igual al del algoritmo de cálculo de derivadas utilizado. Además, el FKED realiza en cada paso un buen número de operaciones con matrices, incluido el cálculo de la inversa de una matriz de tamaño $n_Y \times n_Y$.

[17]La ecuación (4.25) se implementa en una serie de bucles anidados con $i, j, k = 1, \ldots, n_X$.

4.10. Métodos no basados en derivadas

Los algoritmos basados en gradientes o en derivadas del error son los más utilizados, con diferencia, para el entrenamiento de RNR. Pero existen otros algoritmos que evitan el costoso cálculo de estas derivadas y que se basan en la perturbación de los pesos de la red. Veamos dos algoritmos, indicados para redes tanto recurrentes como no recurrentes y únicamente para el entrenamiento fuera de línea.[18]

Alopex. Unnikrishnan y Venugopal (1994) actualizan los parámetros de la red mediante pequeñas perturbaciones de sus valores, en uno u otro sentido según la correlación entre los sentidos de las perturbaciones recientes y el cambio en el error cometido sobre todo el conjunto de entrenamiento. Es un algoritmo especialmente interesante ya que no necesita conocer la forma de la red ni la función de error utilizada, lo que lo hace útil también para cualquier tarea de optimización distinta a la de las redes neuronales.

Algoritmo de Cauwenberghs. Cauwenberghs (1993) sigue una regla de aprendizaje parecida a la de Alopex. Se suma una perturbación aleatoria $\boldsymbol{\pi}$ al vector de pesos actual $\boldsymbol{W}$ y se calcula el error resultante $E(\boldsymbol{W}+\boldsymbol{\pi})$. Este nuevo error se utiliza para actualizar el vector de pesos en la dirección de $\boldsymbol{\pi}$, si es menor que el error anterior $E(\boldsymbol{W})$, o en dirección contraria, si es mayor que $E(\boldsymbol{W})$. El nuevo vector de pesos es, por tanto, $\boldsymbol{W} - \alpha(E(\boldsymbol{W} + \boldsymbol{\pi}) - E(\boldsymbol{W}))\boldsymbol{\pi}$, donde α es la tasa de aprendizaje.

4.11. Problemas en el aprendizaje

Existen varias circunstancias que pueden hacer que una determinada tarea de procesamiento de secuencias no pueda ser resuelta mediante una RNR. En primer lugar, como ya se comentó al comienzo del capítulo, es posible que el modelo neuronal elegido no sea idóneo para esa tarea en particular, circunstancia esta difícil de evaluar en muchas ocasiones. En segundo lugar, aun suponiendo que el modelo elegido sea adecuado (incluyendo tanto el número de neuronas como la representación de la información de entrada y salida), es posible que el algoritmo de entrenamiento empleado no sea capaz de encontrar un valor de los pesos correcto. Los motivos que pueden llevar a ello son principalmente dos: la existencia de mínimos locales o de dependencias a largo plazo.

[18] Hay una carencia casi total de algoritmos de entrenamiento en línea para RNR que no estén basados en el cálculo de derivadas.

4.11.1. Mínimos locales

La función de error E define una superficie multidimensional (hipersuperficie) conocida como *hipersuperficie de error*. Normalmente, la hipersuperficie de error tiene un mínimo global (posiblemente múltiples mínimos globales debido a simetrías de la red) y muchos mínimos locales, que pueden no corresponder a una solución correcta del problema (Bianchini et al. 1994). Estos mínimos locales son consecuencia de la elevada dimensionalidad del espacio de búsqueda y son el mayor problema, al quedar atrapados en ellos, de casi todos los algoritmos de aprendizaje de redes neuronales, especialmente de los que realizan una búsqueda local como los basados en el gradiente.

En cualquier caso, el problema de los minimos locales no es específico de las RNR y afecta a la practica totalidad de los modelos neuronales.

4.11.2. El gradiente evanescente

Aunque en un principio el estado de una RNR puede almacenar la información relevante sobre la historia de una secuencia, la práctica totalidad de los algoritmos de entrenamiento encuentran grandes problemas (en ocasiones insalvables) para *mantener* esta información, especialmente cuando el intervalo de tiempo entre la presencia de una determinada entrada y la salida deseada correspondiente es relativamente largo (normalmente a partir de unos 10 instantes de tiempo). Esto hace que a la hora de la verdad muchas RNR tengan poca ventaja sobre las redes no recurrentes con ventana temporal.

Dependencias a largo plazo. Dada una fuente que genera una secuencia simbólica de la forma $s[1], \ldots, s[t_u], \ldots, s[t_v], \ldots$, diremos que existe una dependencia a largo plazo entre el símbolo[19] del instante t_v y el del instante t_u, y lo expresaremos mediante $s[t_v] \hookleftarrow s[t_u]$, si se cumplen las siguientes condiciones:

1. El valor de $s[t_v]$ depende del valor de $s[t_u]$;
2. $t_v \gg t_u$;
3. No existe t_w con $t_u < t_w < t_v$ tal que $s[t_v] \hookleftarrow s[t_w] \hookleftarrow s[t_u]$.

Los algoritmos de entrenamiento de RNR suelen ser inacapaces de constatar las dependencias a largo plazo debido a que la salida actual de la red

[19] Esta definición puede aplicarse también a cualquier tipo de eventos temporales, no solo a secuencias simbólicas.

es muy poco sensible a una entrada antigua. En el caso de los algoritmos basados en el gradiente, el hecho anterior puede formularse con cierto detalle (Bengio et al. 1994; Hochreiter et al. 2001) a través del llamado *problema del gradiente evanescente*, que veremos a continuación.

Flujo de error. Consideremos las señales de error definidas en las ecuaciones (4.29) a (4.31). Aunque los resultados que se van a mostrar en esta sección son ciertos independientemente del modelo recurrente considerado, de la función de error utilizada (siempre que sea derivable) y de la manera en que se calcule el gradiente, me basaré en el método visto en el ejemplo de BPTT para una RRS del apartado 4.6.1, ya que el análisis es relativamente sencillo a partir de él.

En primer lugar, consideremos el flujo de error *local* entre la i-ésima neurona de salida y la j-ésima neurona de estado. Consideremos el caso en el que el error cometido en el instante t en la neurona de salida, que viene representado por $\delta_i^Y[t]$, "viaja" *hacia atrás* en el tiempo hasta llegar a la neurona de estado del instante $s \leq t$. Esta señal de error intenta "modificar el pasado" de manera que se obtenga un presente más conforme con la salida deseada $\boldsymbol{d}[t]$. El flujo del error queda escalado, por lo tanto, según:

$$\frac{\partial \delta_j^X[s]}{\partial \delta_i^Y[t]} = \begin{cases} g'_X(X_j[t])\, W_{j,i}^{y,x} & s = t \\ g'_X(X_j[s]) \sum_{k=1}^{n_X} \left(\dfrac{\partial \delta_k^X[s+1]}{\partial \delta_i^Y[t]}\, W_{k,j}^{x,x} \right) & s < t \end{cases} \tag{4.67}$$

Si desarrollamos la ecuación anterior, obtenemos:

$$\frac{\partial \delta_j^X[s]}{\partial \delta_i^Y[t]} = \sum_{l_{s+1}=1}^{n_X} \sum_{l_{s+2}=1}^{n_X} \cdots \sum_{l_{t-1}=1}^{n_X} \left(g'_X(X_j[s])\, W_{i,l_{t-1}}^{y,x} \prod_{\tau=s+1}^{t-1} \left(g'_X(X_{l_\tau}[\tau])\, W_{l_\tau,l_{\tau+1}}^{x,x} \right) \right) \tag{4.68}$$

En general, el flujo de error no será local, ya que los errores de las n_Y neuronas de salida viajarán en el tiempo intentando modificar la j-ésima neurona de estado en el instante s. Sin embargo, esto no supone inconveniente alguno para el análisis de esta sección. Efectivamente, el flujo de error *global* se obtiene a partir de:

$$\sum_{i=1}^{n_Y} \frac{\partial \delta_j^X[s]}{\partial \delta_i^Y[t]} \tag{4.69}$$

y, por lo tanto, si cada error local se desvaneciera o explotara, como veremos ahora, también lo haría el error global.

Si el término $|g'_X(X_{l_\tau}[\tau])\, W^{x,x}_{l_\tau,l_{\tau+1}}|$ de (4.68) es mayor que 1 para todo τ, entonces el productorio crece exponencialmente según la distancia temporal entre t y s, es decir, el error *explota* y a la neurona de estado le llega una señal de error que puede hacer oscilar los pesos y volver inestable el aprendizaje (Hochreiter et al. 2001). Por otro lado, si $|g'_X(X_{l_\tau}[\tau])\, W^{x,x}_{l_\tau,l_{\tau+1}}|$ es menor que 1 para todo τ, el productorio decrece exponencialmente con la distancia entre t y s, es decir, el flujo de error se desvanece y no es posible salvar esta distancia, lo que es un problema cuando $\sigma[t] \hookleftarrow \sigma[s]$.

Nótese cómo incrementar el número de unidades de estado n_X no aumenta necesariamente el flujo del error, ya que los sumatorios de (4.68) pueden tener signos diferentes. Este análisis es aplicable a muchas otras RNR, no solo a la RRS; por ejemplo, Hochreiter et al. (2001) lo aplicaron a una RTR.

A la vista de lo anterior, parece plausible que una solución al problema pase por intentar que $g'_X(X_{l_\tau}[\tau])\, W^{x,x}_{l_\tau,l_{\tau+1}} = 1$ para todo τ, esto es, un *flujo de error constante*. La red LSTM se basa en esta idea.

4.12. Justificación de la red LSTM

Para simplificar el análisis de este apartado, consideremos una RNR con una única neurona conectada a sí misma (Hochreiter y Schmidhuber 1997). En ese caso, el término $g'_X(X_{l_\tau}[\tau])\, W^{x,x}_{l_\tau,l_{\tau+1}}$ obtenido en el apartado anterior se puede simplificar a $g'(X_i[\tau])\, W^{x,x}_{i,i}$.

Para garantizar un flujo de error constante, hemos visto que se puede intentar conseguir que $g'(X_i[\tau])\, W^{x,x}_{i,i} = 1$. Integrando esta ecuación respecto a $X_i[t]$ se tiene que:

$$g(X_i[\tau]) = \frac{X_i[\tau]}{W^{x,x}_{i,i}} \tag{4.70}$$

Luego una de las formas de asegurar el flujo de error constante a través de esta neurona es obligar a que $W^{x,x}_{i,i} = 1$ y usar una función de activación identidad para g. A una unidad de este tipo la denominaremos *carrusel de error constante* (CEC) y es la piedra angular de la red LSTM.

Evidentemente, el estudio precedente es extremadamente simple. En general, una neurona recibirá entradas adicionales y actuará sobre más neuronas aparte de ella misma. ¿Cómo podemos garantizar el flujo de error constante sobre una topología más compleja?

Para responder a esta pregunta, podemos considerar dos posibles problemas, que tienen lugar especialmente cuando se consideran dependencias a largo plazo: los que se producen a la entrada de la neurona y los que se producen a su salida.

1. *Conflictos en los pesos de entrada.* Supongamos que la neurona que estamos estudiando ha de gestionar adecuadamente una determinada dependencia a largo plazo $\sigma[t] \hookleftarrow \sigma[s]$ activándose en el instante s y desactivándose en el instante t. En este caso, el algoritmo de entrenamiento intentará que los pesos que llegan a la unidad lineal provenientes de otras neuronas consigan un doble objetivo: por un lado, mantener la neurona activa durante el intervalo correspondiente; por otro lado, proteger la entrada de la neurona de activaciones irrelevantes que pueden desactivarla. Esta circunstancia hace que el aprendizaje sea difícil e invita a controlar de algún modo las *operaciones de escritura* a través de los pesos entrantes.

2. *Conflictos en los pesos de salida.* Los pesos que enlazan la salida de la unidad lineal con otras neuronas también se ajustarán intentando lograr un doble objetivo: por un lado, acceder a la información almacenada en la neurona; por otro, evitar que esta información contamine a las neuronas enlazadas cuando no corresponde. De manera similar al punto anterior, parece útil imponer cierto control en las *operaciones de lectura*.

Las celdas de los bloques de memoria de la red LSTM (véase el apartado 3.2) instrumentan las ideas anteriores. El CEC con activación lineal y conexión recurrente es la característica principal de las celdas de memoria. Las compuertas de entrada vetan la entrada indeseada al CEC y las compuertas de salida hacen lo mismo con su salida.

Los CEC obligan a que el flujo del error sea constante (Hochreiter y Schmidhuber 1997) y permiten superar el problema fundamental del gradiente evanescente; los CEC impiden que la influencia de la señal se debilite rápidamente al ir hacia atrás en el tiempo. Como es de esperar, los errores correspondientes al CEC o a cualquiera de las compuertas se utilizan en la actualización de los pesos. Sin embargo, los CEC son los únicos que siguen la pista del error hacia atrás en el tiempo; el resto de errores se truncan en el algoritmo de entrenamiento. Los gradientes ajenos al CEC, por tanto, pueden desvanecerse exponencialmente de la misma forma que en las RNR tradicionales.

El seguimiento de las dependencias a muy largo plazo en los CEC permite que la red LSTM pueda detectar adecuadamente eventos interdependientes

$$(1)\ \frac{\partial x_{ij}[t]}{\partial W_k^{\gamma}} \approx 0 \qquad (2)\ \frac{\partial x_{ij}[t]}{\partial W_{k,l}^{\gamma,u}} \approx 0 \qquad (3)\ \frac{\partial x_{ij}[t]}{\partial W_{k,lm}^{\gamma,z}} \approx 0$$

$$(4)\ \frac{\partial z_{ij}[t-1]}{\partial W_k^{\gamma}} \approx 0 \qquad (5)\ \frac{\partial z_{ij}[t-1]}{\partial W_{k,l}^{\gamma,u}} \approx 0 \qquad (6)\ \frac{\partial z_{ij}[t-1]}{\partial W_{k,lm}^{\gamma,z}} \approx 0$$

$$(7)\ \frac{\partial \gamma_i[t]}{\partial W_j^{\phi}} \approx 0 \qquad (8)\ \frac{\partial \gamma_i[t]}{\partial W_{j,k}^{\phi,u}} \approx 0 \qquad (9)\ \frac{\partial \gamma_i[t]}{\partial W_{j,kl}^{\phi,z}} \approx 0$$

$$(10)\ \frac{\partial Z_{ij}[t]}{\partial W_k^{\phi}} \approx 0 \qquad (11)\ \frac{\partial Z_{ij}[t]}{\partial W_{k,l}^{\phi,u}} \approx 0 \qquad (12)\ \frac{\partial Z_{ij}[t]}{\partial W_{k,lm}^{\phi,z}} \approx 0$$

$$(13)\ \frac{\partial \phi_i[t]}{\partial W_{jk}^{z}} \approx 0 \qquad (14)\ \frac{\partial \phi_i[t]}{\partial W_{jk,l}^{z,u}} \approx 0 \qquad (15)\ \frac{\partial \phi_i[t]}{\partial W_{jk,lm}^{z,z}} \approx 0$$

$$(16)\ \frac{\partial \gamma_i[t]}{\partial W_{jk}^{z}} \approx 0 \qquad (17)\ \frac{\partial \gamma_i[t]}{\partial W_{jk,l}^{z,u}} \approx 0 \qquad (18)\ \frac{\partial \gamma_i[t]}{\partial W_{jk,lm}^{z,z}} \approx 0$$

Figura 4.3: Truncamientos totales realizados en el cálculo de las derivadas de la red LSTM en el instante t. Las aproximaciones de esta figura se cumplen para cualquier valor de los subíndices.

separados por miles de instantes de tiempo discreto, mientras que las RNR tradicionales no suelen ser capaces de manejar correctamente intervalos superiores a unos 10 instantes de tiempo.

4.12.1. Cálculo del gradiente

Como se ha visto en este capítulo, los algoritmos de entrenamiento basados en el gradiente necesitan calcular las derivadas parciales de la no linealidad global de la red, representada por $\boldsymbol{y}[t]$, con respecto a cada uno de los pesos. Estas derivadas son $\partial y_i[t]/\partial W_j[t]$ donde $i = 1, \ldots, n_Y$ y W_j es un peso sináptico cualquiera.

La forma de calcular las derivadas de la función de salida de la red LSTM unida a la propia topología del modelo permite superar el problema del gradiente evanescente, como se ha dicho anteriormente. El truncamiento realizado sobre ciertas derivadas garantiza que los errores no pueden volver a entrar en las celdas y esto permite el flujo constante del error en el interior de la celda de memoria.

$$(19)\ \frac{\partial x_{ij}[t]}{\partial W_k^{\phi}} \approx 0, \quad \text{si } i \neq k \qquad (20)\ \frac{\partial x_{ij}[t]}{\partial W_{k,l}^{\phi,u}} \approx 0, \quad \text{si } i \neq k$$

$$(21)\ \frac{\partial x_{ij}[t]}{\partial W_{k,lm}^{\phi,z}} \approx 0, \quad \text{si } i \neq k \qquad (22)\ \frac{\partial x_{ij}[t]}{\partial W_{kl}^{z}} \approx 0, \quad \text{si } ij \neq kl$$

$$(23)\ \frac{\partial x_{ij}[t]}{\partial W_{kl,m}^{z,u}} \approx 0, \quad \text{si } ij \neq kl \qquad (24)\ \frac{\partial x_{ij}[t]}{\partial W_{kl,mn}^{z,z}} \approx 0, \quad \text{si } ij \neq kl$$

Figura 4.4: Truncamientos parciales realizados en el cálculo de las derivadas de la red LSTM en el instante t.

A continuación se describen las ecuaciones del gradiente de la red LSTM; en los casos en los que se realiza algún truncamiento, se indica la expresión truncada según los números que aparecen en las figuras 4.3 y 4.4. El gradiente con respecto a los pesos que inciden en la capa de salida es:

$$\frac{\partial y_i[t]}{\partial W_j^{y}} = g'_Y(Y_i[t])\delta_{i,j} \tag{4.71}$$

$$\frac{\partial y_i[t]}{\partial W_{j,k}^{y,u}} = g'_Y(Y_i[t])u_k[t]\delta_{i,j} \tag{4.72}$$

$$\frac{\partial y_i[t]}{\partial W_{j,kl}^{y,z}} = g'_Y(Y_i[t])z_{kl}[t]\delta_{i,j} \tag{4.73}$$

Las derivadas con respecto a los pesos de las compuertas de salida son:

$$\frac{\partial y_i[t]}{\partial W_j^{\gamma}} \stackrel{1,4}{\approx} g'_Y(Y_i[t])g'_C(\Gamma_j[t]) \sum_{k=1}^{n_C} W_{i,jk}^{y,z} g_M(x_{jk}[t]) \tag{4.74}$$

$$\frac{\partial y_i[t]}{\partial W_{j,k}^{\gamma,u}} \stackrel{2,5}{\approx} g'_Y(Y_i[t])g'_C(\Gamma_j[t])u_k[t] \sum_{l=1}^{n_C} W_{i,jl}^{y,z} g_M(x_{jl}[t]) \tag{4.75}$$

$$\frac{\partial y_i[t]}{\partial W_{j,kl}^{\gamma,z}} \stackrel{3,6}{\approx} g'_Y(Y_i[t])g'_C(\Gamma_j[t])z_{kl}[t-1] \sum_{m=1}^{n_C} W_{i,jm}^{y,z} g_M(x_{jm}[t]) \tag{4.76}$$

Las derivadas de la función de salida con respecto a los pesos de la compuerta de entrada, con indicación de los truncamientos efectuados, son:

$$\frac{\partial y_i[t]}{\partial W_j^{\phi}} \overset{7,19}{\approx} g'_Y(Y_i[t])\gamma_j[t]\sum_{k=1}^{n_C} W_{i,jk}^{y,z} g'_M(x_{jk}[t])\frac{\partial x_{jk}[t]}{\partial W_j^{\phi}} \tag{4.77}$$

$$\frac{\partial y_i[t]}{\partial W_{j,k}^{\phi,u}} \overset{8,20}{\approx} g'_Y(Y_i[t])\gamma_j[t]\sum_{l=1}^{n_C} W_{i,jl}^{y,z} g'_M(x_{jl}[t])\frac{\partial x_{jl}[t]}{\partial W_{j,k}^{\phi,u}} \tag{4.78}$$

$$\frac{\partial y_i[t]}{\partial W_{j,kl}^{\phi,z}} \overset{9,21}{\approx} g'_Y(Y_i[t])\gamma_j[t]\sum_{m=1}^{n_C} W_{i,jm}^{y,z} g'_M(x_{jm}[t])\frac{\partial x_{jm}[t]}{\partial W_{j,kl}^{\phi,z}} \tag{4.79}$$

donde se han utilizado los siguientes términos recurrentes:

$$\frac{\partial x_{ij}[t]}{\partial W_i^{\phi}} \overset{10}{\approx} g_Z(Z_{ij}[t])g'_C(\Phi_i[t]) + \frac{\partial x_{ij}[t-1]}{\partial W_i^{\phi}} \tag{4.80}$$

$$\frac{\partial x_{ij}[t]}{\partial W_{i,k}^{\phi,u}} \overset{11}{\approx} g_Z(Z_{ij}[t])g'_C(\Phi_i[t])u_k[t] + \frac{\partial x_{ij}[t-1]}{\partial W_{i,k}^{\phi,u}} \tag{4.81}$$

$$\frac{\partial x_{ij}[t]}{\partial W_{i,kl}^{\phi,z}} \overset{12}{\approx} g_Z(Z_{ij}[t])g'_C(\Phi_i[t])z_{kl}[t-1] + \frac{\partial x_{ij}[t-1]}{\partial W_{i,kl}^{\phi,z}} \tag{4.82}$$

Finalmente, el gradiente con respecto a los pesos que entran directamente en los bloques de memoria son:

$$\frac{\partial y_i[t]}{\partial W_{jk}^{z}} \overset{16,22}{\approx} g'_Y(Y_i[t])W_{i,jk}^{y,z}\gamma_j[t]g'_M(x_{ij}[t])\frac{\partial x_{ij}[t]}{\partial W_{jk}^{z}} \tag{4.83}$$

$$\frac{\partial y_i[t]}{\partial W_{jk,l}^{z,u}} \overset{17,23}{\approx} g'_Y(Y_i[t])W_{i,jk}^{y,z}\gamma_j[t]g'_M(x_{ij}[t])\frac{\partial x_{ij}[t]}{\partial W_{jk,l}^{z,u}} \tag{4.84}$$

$$\frac{\partial y_i[t]}{\partial W_{jk,lm}^{z,z}} \overset{18,24}{\approx} g'_Y(Y_i[t])W_{i,jk}^{y,z}\gamma_j[t]g'_M(x_{ij}[t])\frac{\partial x_{ij}[t]}{\partial W_{jk,lm}^{z,z}} \tag{4.85}$$

donde se han utilizado los siguientes términos recurrentes:

$$\frac{\partial x_{ij}[t]}{\partial W_{ij}^{z}} \overset{13}{\approx} \phi_i[t]g'_Z(Z_{ij}[t]) + \frac{\partial x_{ij}[t-1]}{\partial W_{ij}^{z}} \tag{4.86}$$

$$\frac{\partial x_{ij}[t]}{\partial W_{ij,k}^{z,u}} \overset{14}{\approx} \phi_i[t]g'_Z(Z_{ij}[t])u_k[t] + \frac{\partial x_{ij}[t-1]}{\partial W_{ij,k}^{z,u}} \tag{4.87}$$

$$\frac{\partial x_{ij}[t]}{\partial W_{ij,kl}^{z,z}} \overset{15}{\approx} \phi_i[t]g'_Z(Z_{ij}[t])z_{kl}[t-1] + \frac{\partial x_{ij}[t-1]}{\partial W_{ij,kl}^{z,z}} \tag{4.88}$$

5. LAS SOLUCIONES PREVIAS

En el capítulo 1 se presentaron los problemas que se estudian en esta tesis. Una vez introducidos en los capítulos anteriores los conceptos fundamentales de las redes recurrentes, mostramos ahora las soluciones previas tanto neuronales como no neuronales planteadas para resolver dichos problemas.

Como este capítulo complementa lo discutido en 1.2, lo dividiremos en los mismos apartados.

5.1. Compresión de secuencias simbólicas

5.1.1. Entropía

Supongamos una secuencia simbólica $s = s[1], s[2], \ldots, s[L]$ generada bajo un determinado modelo estocástico M basado en estados (por ejemplo, un modelo oculto de Markov de los discutidos en el apéndice A) a partir de un alfabeto $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$. Tras haber generado un determinado prefijo de s, digamos $s[1], s[2], \ldots, s[t]$ con $1 \leq t < L$, el modelo M se encontrará en un estado $q[t] \in Q$, donde Q es el espacio de estados del modelo. Con ayuda de $p(\sigma_i \,|\, q[t])$ podemos conocer la probabilidad de que el siguiente símbolo de la secuencia sea σ_i:

$$p(s[t+1] = \sigma_i \,|\, s[1], s[2], \ldots, s[t]) = p(\sigma_i \,|\, q[t]) \tag{5.1}$$

La probabilidad de una secuencia s de longitud $|s| = L$ se obtiene fácilmente a partir de:

$$p(s) = \prod_{t=1}^{L} p(s[t] \,|\, s[1], s[2], \ldots, s[t-1]) = \prod_{t=1}^{L} p(s[t]|q[t-1]) \tag{5.2}$$

con $q[0] = q_I$, donde q_I es el estado inicial del modelo.

Sea Σ^L el conjunto de secuencias de longitud L que pueden ser generadas a partir del alfabeto Σ. Podemos definir la *entropía* (Cover y Thomas 1991;

Charniak 1993) de este conjunto como:

$$H_L(M) = -\sum_{s \in \Sigma^L} p(s) \log p(s) \tag{5.3}$$

que se mide en *bits* si el logaritmo es en base 2. La entropía es una medida del grado de incertidumbre asociado a las secuencias de Σ^L y define una cota inferior del número medio de bits necesarios para transmitir las secuencias del conjunto.

Nótese cómo la entropía crece con L. Ya que normalmente no se necesita trabajar con una longitud determinada, una medida más aconsejable es la *entropía por símbolo*, que es:

$$\frac{1}{L} H_L(M) = -\frac{1}{L} \sum_{s \in \Sigma^L} p(s) \log p(s) \tag{5.4}$$

De hecho, lo realmente interesante es poder considerar secuencias de longitud arbitraria. Por ello, lo más habitual es hablar de $H(M)$, la *entropía del modelo* M, que es el límite de la entropía por símbolo cuando crece la longitud de la secuencia:

$$H(M) = -\lim_{L \to \infty} \frac{1}{L} \sum_{s \in \Sigma^L} p(s) \log p(s) \tag{5.5}$$

Supongamos ahora que establecemos un modelo de probabilidad alternativo $\hat{M}$ para la probabilidad q de cada secuencia s. Este modelo alternativo permite obtener las probabilidades $q(s[t+1] = \sigma_i | s[1], s[2], \ldots, s[t])$ del siguiente símbolo de una secuencia s y, por tanto, la probabilidad total de la secuencia, $q(s)$, mediante (5.2).

La *entropía cruzada* mide el parecido de ambos modelos de probabilidad (el alternativo y el real) y alcanza su valor mínimo (igual a la entropía del modelo real) cuando coinciden. La entropía cruzada es, por tanto, una medida de la corrección del predictor alternativo y se define como:

$$H_L(M, \hat{M}) = -\sum_{s \in \Sigma^L} p(s) \log q(s) \tag{5.6}$$

Nótese que $H_L(M, M) = H_L(M)$. Se cumple, además, para cualquier modelo $\hat{M}$ que:

$$H_L(M) \leq H_L(M, \hat{M}) \tag{5.7}$$

donde la igualdad se da solo cuando $M = \hat{M}$.

De igual manera, podemos definir la *entropía cruzada por símbolo* como:

$$\frac{1}{L}H_L(M,\hat{M}) = -\frac{1}{L}\sum_{s\in\Sigma^L} p(s)\log q(s) \tag{5.8}$$

y la *entropía cruzada* entre M y $\hat{M}$ como:

$$H(M,\hat{M}) = -\lim_{L\to\infty}\frac{1}{L}\sum_{s\in\Sigma^L} p(s)\log q(s) \tag{5.9}$$

Una medida similar a la entropía cruzada es la *entropía relativa*, también llamada *divergencia de Kullback-Leibler*, que viene definida por:

$$\sum_s p(s)\log\frac{p(s)}{q(s)} = H(M,\hat{M}) - H(M) \tag{5.10}$$

La divergencia de Kullback-Leibler es siempre positiva, aunque no simétrica, y vale cero cuando ambas distribuciones de probabilidad coinciden.

Normalmente no es posible conocer las probabilidades reales de la secuencia, porque se desconoce el modelo de estados M o, incluso, porque la secuencia no ha sido generada a través de uno. Deberemos, por tanto, modelizar buenos predictores que estimen esas probabilidades y que minimicen medidas como la entropía cruzada o la entropía relativa.

Como veremos a continuación, un buen modelo de probabilidad es muy útil a la hora de comprimir una secuencia. En este contexto, la *razón de compresión* se define como la razón entre la longitud en bits de la secuencia original usando un código de longitud constante y la longitud en bits de la comprimida.

5.1.2. Compresión de Huffman

Como ya se ha indicado, la existencia de un modelo de las probabilidades del siguiente símbolo de una secuencia puede utilizarse para comprimirla.

Si los símbolos del alfabeto Σ son equiprobables, entonces cada símbolo se codificará con $\lceil \log_2 |\Sigma| \rceil$ bits,[1] donde $\lceil x \rceil$ indica el menor entero mayor o igual que x.

Sin embargo, si los símbolos del alfabeto no son equiprobables, puede intentarse asignar diferentes longitudes a sus codificaciones de manera que las

[1] En toda esta discusión consideraremos que el sistema de codificación es binario. Existen también resultados para el caso de alfabetos de codificación de más de dos símbolos (Rifà y Huguet 1991).

longitudes más cortas se asignen a los símbolos más probables. No obstante, debe tenerse cuidado en codificar los símbolos de manera que la concantenación de sus codificaciones sea recuperable de forma unívoca y proporcione la secuencia original. Una codificación que cumpla esta propiedad se denomina *codificación instantánea*.

La asignación de códigos de diferente longitud es el principio del célebre algoritmo de compresion de Huffman (Bell et al. 1990; Nelson y Gailly 1995). El algoritmo de Huffman original recibe como entrada un alfabeto $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ y un conjunto de probabilidades de aparición $p(\sigma_i)$ de cada uno de los símbolos del alfabeto;[2] como salida proporciona un código para cada símbolo. La codificación resultante es, además, instantánea.

Los pasos para construir una codificación de Huffman se pueden describir (Rifà y Huguet 1991) como sigue:

1. Ordenar todos los símbolos de Σ por sus probabilidades asociadas.
2. Combinar reiteradamente los dos símbolos con menor probabilidad para formar un símbolo compuesto cuya probabilidad será la suma de las probabilidades de los dos símbolos; estas agrupaciones determinan un arbol binario en el que cada nodo es la probabilidad de todos sus descendientes.
3. Al descender por el árbol hacia cada hoja se forma el código correspondiente al símbolo de esa hoja, si se toma un 0 cada vez que se desciende a un nodo derecho y un 1 en caso contrario (o viceversa).

El algoritmo de Huffman puede considerarse óptimo en el sentido de que las secuencias codificadas según el algoritmo tienen la menor longitud posible (siempre que las frecuencias de sus símbolos se ajusten a las indicadas). Sin embargo, veremos que esto es cierto solo si partimos de la base de que la codificación se ha de realizar asignando códigos símbolo a símbolo, lo cual comporta limitaciones obvias aun cuando se permita que las probabilidades de aparición cambien dinámicamente. Otras técnicas, como la compresión aritmética, desarrollan una codificación *global* de la secuencia y suelen conseguir mayores razones de compresión.

5.1.3. Compresión aritmética

Aunque la codificación de Huffman se presenta a veces como la forma perfecta de comprimir datos, la realidad es que los códigos generados por ella solo proporcionan codificaciones óptimas (en el sentido de un número

[2] Existen también adaptaciones para manejar probabilidades dinámicas contextuales.

medio de bits por símbolo igual a la entropía) cuando las probabilidades de los símbolos son potencias enteras de $1/2$, lo cual no es el caso habitual, y cuando el contexto previo de los símbolos no facilita su codificación.

La compresión aritmética (Bell et al. 1990; Nelson y Gailly 1995) no presenta ninguna de estas dos restricciones y obtiene el límite teórico de la entropía para cualquier mensaje de entrada cuando se utiliza un modelo correcto de probabilidad. Además, su uso en entornos en los que las probabilidades de los símbolos cambian dinámicamente es directo, lo que no ocurre con los códigos de Huffman.

La compresión aritmética funciona representando cada secuencia completa con un subintervalo $\mathcal{I}$ del intervalo real $[0, 1)$. Cuanto más largo es el mensaje, el intervalo necesario para representarlo se hace más pequeño y el número de bits necesarios para especificar el intervalo aumenta. Los sucesivos símbolos del mensaje van reduciendo este intervalo según sus probabilidades. Los símbolos más probables lo reducen en menor medida y, por lo tanto, añaden menos bits al mensaje codificado.

Para mostrar las ideas anteriores con un ejemplo, concentrémonos en un sistema sencillo en el que las probabilides son *estacionarias* y no cambian con el paso del tiempo. Supongamos que el alfabeto es $\Sigma = \{A, E, I, O, U\}$ con las probabilidades estáticas $p(A) = 0.12$, $p(E) = 0.42$, $p(I) = 0.09$, $p(O) = 0.3$ y $p(U) = 0.07$ (Press et al. 1992, pp. 910ss.).

La figura 5.1 muestra cómo se codifica la secuencia IOU. El intervalo $[0, 1)$ se divide en 5 segmentos, uno por cada símbolo del alfabeto; la longitud de cada segmento es igual a la probabilidad del símbolo correspondiente. El primer símbolo de la secuencia, I, estrecha el rango de $\mathcal{I}$ a $0.37 \leq \mathcal{I} < 0.46$. Este subintervalo se divide en 5 subintervalos a su vez, cada uno con longitudes proporcionales a la de las probabilidades correspondientes. El siguiente símbolo, O, reduce el intervalo $\mathcal{I}$ aún más, de manera que ahora $0.3763 \leq \mathcal{I} < 0.4033$. Finalmente, el símbolo U proporciona el intervalo final[3] $0.37630 \leq \mathcal{I} < 0.37819$. Cualquier número de este intervalo puede considerarse como la codificación de la secuencia en cuestión; en particular, podemos considerar la fracción binaria 0.011000001 (el número binario que menos bits necesita dentro del intervalo) que representa a la secuencia IOU y puede enviarse con 9 bits (no es necesario enviar la parte entera, ya que siempre es cero).

[3] Este intervalo no representa únicamente a la secuencia IOU, sino a cualquier secuencia que comience por $IOU\ldots$; para poder distinguirlas, se suele añadir al alfabeto un símbolo especial de terminación de secuencia al que se asigna una probabilidad muy pequeña y que solo aparece como último símbolo de cada secuencia procesada.

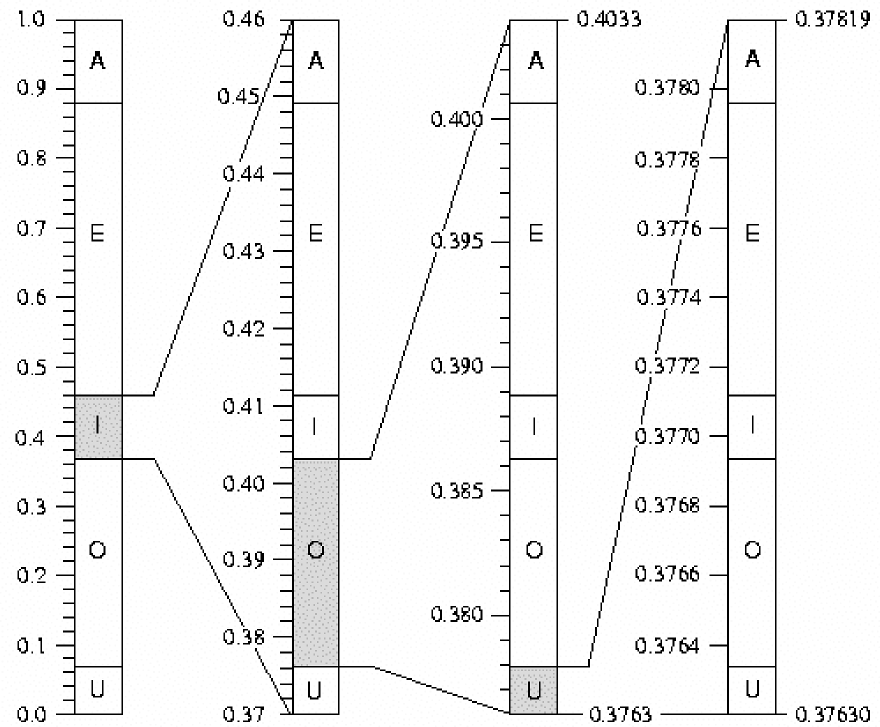


Figura 5.1: Codificación aritmética de la secuencia *IOU* según el alfabeto y las probabilidades indicadas en el texto. *Figura tomada del libro de Press et al. (1992).*

Evidentemente, la descodificación consiste en invertir el proceso de manera que el intervalo considerado se va ensanchando hasta llegar a $[0, 1)$. El descodificador, por lo tanto, ha de conocer el modelo de probabilidad aplicado en cada instante de tiempo.

La implementación práctica del algoritmo requiere algunas consideraciones adicionales debido a la limitada precisión numérica de los ordenadores, que impide de entrada la representación de cantidades con un número arbitrario de cifras decimales. Estos detalles de implementación pueden encontrarse en el trabajo de Nelson (1991).

El ejemplo anterior suponía un modelo no adaptativo de las probabilidades del siguiente símbolo. En el caso de utilizar un modelo adaptativo las probabilidades pueden reestimarse en cada paso, ajustándolas después de procesar cada símbolo y considerando la historia previa de la secuencia. Para ello hace falta un *predictor* que determine las probabilidades del siguiente símbolo de la secuencia que se utilizan para estrechar el rango $\mathcal{I}$. La razón de compresión obtenida con la compresión aritmética puede utilizarse como medida de la calidad del predictor, en tanto que mejores predictores proporcionarán mayores razones de compresión.

5.1.4. Modelos de probabilidad de n-gramas

Para poder utilizar adecuadamente la compresión aritmética se necesita construir un modelo estadístico de los datos. El más sencillo es un modelo fijo; por ejemplo, una lista con las frecuencias de aparición de los distintos símbolos. Sin embargo, un modelo que considere únicamente las probabilidades estáticas de los diferentes símbolos no suele proporcionar una buena estimación de la entropía real de los datos, como ya hemos comentado. Esto ocurre especialmente cuando el entorno en que se genera la secuencia es *no estacionario*, es decir las propiedades del modelo estocástico que la genera varían con el tiempo.

Una posibilidad para mejorar el modelo anterior es utilizar un modelo de probabilidad adaptativo que evolucione con la secuencia. Los modelos de n-gramas son uno de los más conocidos dentro de esta última categoría.

En un modelo de n-gramas se asume que la identidad del siguiente símbolo de la secuencia está condicionada por las de los $n-1$ símbolos anteriores, esto es, la ecuación (5.2) se transforma en:

$$p(s[t+1] \mid s[1], s[2], \ldots, s[t]) = p(s[t+1] \mid s[t-(n-2)], \ldots, s[t]) \quad (5.11)$$

En el caso de los *bigramas* (donde $n = 2$), la ecuación anterior se convierte en:

$$p(s[t+1]|s[1], s[2], \ldots, s[t]) = p(s[t+1]|s[t]) \quad (5.12)$$

La estimación de las probabilidades del símbolo siguiente es sencilla en este caso y puede hacerse mediante:

$$p(s[t+1] = \sigma_i \mid s[t]) = \frac{C(s[t]\sigma_i)}{\sum_{j=1}^{|\Sigma|} C(s[t]\sigma_j)} \quad (5.13)$$

donde $C(\cdot)$ representa el número de veces que el argumento ha aparecido en la secuencia observada hasta el instante actual. La generalización al caso de n-gramas es directa.

Para evitar que las probabilidades calculadas con la expresión anterior sean cero, se suele reservar una pequeña parte de la probabilidad para subsecuencias de n-gramas que no han aparecido todavía en la secuencia.

Las razones de compresión obtenidas son muy buenas (Nelson 1991) cuando se utilizan modelos dinámicos y adaptativos de n-gramas y se mantienen simultáneamente modelos de distintos órdenes. Por ejemplo, en la técnica conocida como *predicción por concordancia parcial* se busca una concordancia de la subsecuencia más reciente en un modelo de orden n. Si no se encuentra ninguna concordancia, se baja a un modelo de orden $n-1$, y así sucesivamente hasta llegar a un modelo de orden 0.[4]

Las RNR permiten obtener un modelo de las probabilidades del siguiente símbolo mediante lo que en principio podría considerarse como un modelo de ∞-gramas.[5]

Por otra parte, es de vital importancia que tanto el compresor como el descompresor desarrollen el mismo modelo adaptativo de probabilidad.[6]

5.1.5. Programas de compresión

Ya que en el capítulo 6 mostraremos los resultados obtenidos con ellos, se presentan aquí algunos programas conocidos de compresión y las técnicas en las que se basan.[7]

El programa `gzip` utiliza el algoritmo de compresión de Lempel y Ziv (Ziv y Lempel 1977; Nelson y Gailly 1995), llamado así en honor a sus creadores. Ambos propusieron dos versiones de su algoritmo, una en 1977 (LZ77) y otra en 1978 (LZ78); `gzip` utiliza la primera. Otro conocido programa de entornos Unix, `compress`, se basa en LZW (por Lempel, Ziv y Welch), una variación de 1984 de LZ78. Tanto LZ77 como LZ78 se consideran *compresores sustitucionales*: la idea básica es sustituir la presencia de una determinada subsecuencia de la secuencia a comprimir por una referencia a una aparición previa de esa misma subsecuencia. La compresión obtenida con estos compresores sustitucionales suele ser mayor que la obtenida con técnicas basadas en la codificación de Huffman (adaptativas o no).

[4] Un modelo de monogramas solo tiene en cuenta el número de veces que ha aparecido el símbolo en cuestión; un modelo de 0-gramas considera que todos los símbolos son equiprobables.

[5] En realidad, la memoria de las RNR estándar es muy limitada y se reduce a unos cuantos símbolos debido a los problemas que aparecen al tener que tratar con dependencias a largo plazo (véase el apartado 4.11.2).

[6] En el caso de las redes neuronales, esto implica partir de los mismos valores para todos los parámetros del sistema (los pesos pueden iniciarse aleatoriamente siempre que tanto compresor como descompresor utilicen el mismo generador de números aleatorios iniciado con una semilla idéntica).

[7] Debe tenerse en cuenta que nuestra intención es evaluar la capacidad de predicción de las RNR sobre secuencias simbólicas; por lo tanto, no tiene mucho sentido comparar los resultados de algunos programas como `gzip` o `bzip2` con los de un compresor aritmético basado en un predictor neuronal. Las razones de compresión con este tipo de compresores se mostrarán para hacernos una idea de cuál puede ser un buen resultado.

El programa `bzip2` utiliza la transformada de Burrows y Wheeler seguida de compresión de Huffman. Esta transformada (Burrows y Wheeler 1994; Nelson y Gailly 1995) es un algoritmo de reordenación reversible; su salida ordenada se comprime fácilmente con las técnicas habituales de compresión. La razón de compresión suele ser mayor que la obtenida con compresores basados en LZ77 como `gzip`. Sin embargo, `bzip2` comprime las secuencias en bloques por lo que no es muy adecuado para la compresión en línea.

Ninguno de los compresores anteriores utilizan (al menos, no como parte fundamental de su operación) un modelo probabilístico.

Por otro lado, uno de los pocos compresores aritméticos existentes es el desarrollado por Nelson (1991). Este compresor utiliza una combinación de modelos de n-gramas de distintos órdenes para estimar las probabilidades, como se indicó antes.

5.1.6. Diferencias con la inferencia gramatical clásica

Como ya se comentó en 1.2.1, en esta tesis el modelo de probabilidad se obtiene *en línea*: las RNR trabajan en tiempo real dando una salida tan correcta como sea posible para cada elemento de la secuencia suministrado en cada iteración; esta salida se considera como una predicción de las probabilidades del siguiente símbolo de la secuencia. Esto supone una diferencia fundamental con la forma clásica de abordar la inferencia gramatical, una tarea que como ya se vió en 1.2.2 ha sido ampliamente estudiada con RNR.

Aunque una de las formas de realizar la inferencia gramatical es entrenar la red para que aprenda a predecir el siguiente símbolo de la secuencia, el *modus operandi* seguido tradicionalmente es bastante distinto al de nuestro problema. Bajo el enfoque tradicional, puede demostrarse fácilmente (escribiendo el error de predicción total como una suma sobre todos los prefijos de todas las secuencias de la muestra) que el modelo neuronal ideal obtenido para un conjunto finito de secuencias finitas por medio de (véase el apartado 4.3.1 para la demostración de estas ideas):

1. entrenamiento intensivo fuera de línea,

2. uso de una familia de funciones de error global (como la función de error cuadrático o la distancia de Kullback-Leibler), y

3. codificación exclusiva de las salidas deseadas,[8]

[8]Esto implica que la propiedad de convergencia aquí señalada no puede aplicarse a señales numéricas, únicamente a símbolos discretos.

proporciona una salida que aproxima tanto como sea posible las frecuencias relativas del siguiente símbolo observadas en la muestra finita; estas salidas pueden utilizarse como probabilidades aproximadas al tratar secuencias nuevas.

El problema considerado en este trabajo es diferente en tres aspectos principales: se procesa una única secuencia ilimitada, el procesamiento es en línea y la función de error utilizada es local. Nuestro trabajo se basa en la conjetura, similar a la planteada por Elman (1990, p. 197) de que, incluso bajo estas condiciones distintas, la salida de la red puede seguir considerándose como una aproximación a las probabilidades del siguiente símbolo.

5.1.7. Trabajos neuronales

Se analiza, por lo tanto, el uso de RNR para predecir en línea el siguiente elemento de una secuencia. La compresión aritmética se usa para evaluar la calidad del predictor. En los experimentos se consideran diferentes generadores de secuencias simbólicas que van desde máquinas de estados finitos a textos en lenguaje humano. Al contrario que anteriores trabajos (Cleeremans et al. 1989; Schmidhuber y Stefan 1996; Tiño y Köteles 1999), que utilizaban predicción fuera de línea con estas secuencias, en este trabajo nos concentraremos en la predicción *en línea*.

Hay varios trabajos que discuten el uso de redes neuronales como modelo probabilístico para la codificación aritmética de textos en lenguaje humano. Schmidhuber y Stefan (1996) usan un perceptrón multicapa con una ventana temporal de entradas para obtener resultados prometedores mediante retropropagación estándar (Rumelhart et al. 1986). Un enfoque muy similar es el seguido por Long et al. (1999), pero los símbolos de la secuencia a comprimir se recodifican previamente para acercar entre sí las representaciones de los símbolos con un contexto posterior similar. Ambas técnicas, sin embargo, trabajan bajo la suposición de un entrenamiento fuera de línea, lo que las hace inadecuadas para entornos no estacionarios o aplicaciones en tiempo real.

Mahoney (2000) utiliza codificación aritmética y un perceptrón sin capa oculta para comprimir textos en línea, aunque la técnica no es completamente neuronal. Su perceptrón tiene miles o incluso millones de entradas, ya que se usa una entrada distinta para cada contexto posible, desde el símbolo anterior a los 5 anteriores. Esta elevada complejidad espacial se ve compensada por una baja complejidad temporal, ya que únicamente hace falta actualizar un número reducido de pesos tras cada símbolo. Además, la predicción se realiza bit a bit con lo que solo se precisa una neurona de salida.

La técnica, sin embargo, no es completamente neuronal, ya que junto a cada peso se almacenan dos contadores que se encargan de guardar el número de apariciones del contexto asociado. Los resultados de sus experimentos superan los de otras técnicas de compresión.

En cuanto a las secuencias de estados finitos, es de esperar buenos resultados con RNR, ya que se ha demostrado (Goudreau et al. 1994; Alquézar y Sanfeliu 1995; Horne y Hush 1996; Carrasco et al. 2000) la capacidad de distintas RNR de primer y segundo orden para codificar máquinas de estados finitos.

5.2. Inferencia de lenguajes con dependencias a largo plazo

Como ya se comentó en el apartado 1.2.2, una de las formas de realizar la inferencia de lenguajes es mediante la predicción del siguiente símbolo de una secuencia. También allí se comentó cómo la red LSTM permite superar parcialmente el problema que surge al tratar las dependencias a largo plazo.

Todos los trabajos anteriores de inferencia de lenguajes con LSTM usaban un algoritmo de entrenamiento de descenso por el gradiente para actualizar los pesos. En los trabajos desarrollados para esta tesis, se aplica por primera vez el FKED a la red LSTM y se comparan los resultados obtenidos con los del algoritmo original y con los conseguidos con RNR tradicionales.

En primer lugar, se considera la predicción en línea sobre una secuencia generada por el autómata simétrico de Reber (Smith y Zipser 1989) de la figura 6.4. Se trata de la primera vez en que la red LSTM se aplica a una situación de aprendizaje completamente en línea: la red tiene que dar en tiempo real una salida tan correcta como sea posible para la entrada aplicada en cada instante de tiempo. Con los experimentos se estudiará qué clase de modelo interno desarrolla la red y si este es comparable al obtenido en trabajos anteriores (Hochreiter y Schmidhuber 1997; Gers et al. 2000), en los que la inferencia se realizaba fuera de línea o solo parcialmente en línea.

En segundo lugar, se ahonda en algunos resultados previos en torno a la capacidad de las RNR para aprender un sencillo lenguaje sensible al contexto, $a^n b^n c^n$, con fuertes dependencias a largo plazo conforme se aumenta el valor de n.

Se ha demostrado que las RNR son capaces de reconocer lenguajes derivados de máquinas de estados finitos[9]: la mayor parte de los trabajos relacionados con la inferencia de lenguajes con RNR se ha centrado en ellos.

[9]Véase el libro inédito de Forcada (2002) para un repaso del estado de la cuestión de este aspecto.

Solo unos cuantos autores han intentado enseñar a las RNR a extraer las reglas de algunos lenguajes independientes del contexto o sensibles al contexto (Sun et al. 1993; Wiles y Elman 1995; Tonkes y Wiles 1997; Rodriguez y Wiles 1998; Rodriguez et al. 1999; Bodén y Wiles 2000), que requieren el equivalente funcional a una o dos pilas, respectivamente, de tamaño ilimitado.

Algunos de los trabajos anteriores fracasaron al intentar aprender pequeños conjuntos de entrenamiento de lenguajes independientes del contexto. Aunque otros trabajos lo consiguieron e incluso algunos llegaron a aprender pequeños conjuntos de entrenamiento de lenguajes sensibles al contexto (Chalup y Blair 1999; Bodén y Wiles 2000), las redes obtenidas eran incapaces de generalizar adecuadamente con conjuntos de evaluación ligeramente mayores.

En esta tesis nos concentraremos en uno de los pocos lenguajes sensibles al contexto a cuyo aprendizaje se han destinado las RNR, a saber, $a^n b^n c^n$. Las RNR tradicionales no son capaces de generalizar adecuadamente con este sencillo lenguaje: Chalup y Blair (1999) constataron que una RRS entrenada mediante un algoritmo voraz podía aprender el conjunto de entrenamiento para $n \leq 12$, pero no aportaron datos sobre la capacidad de generalización. Bodén y Wiles (2000), por otra parte, entrenaron una red secuencial en cascada de segundo orden con BPTT; para un conjunto de entrenamiento con $n \in [1, 10]$, las mejores redes generalizaban a $n \in [1, 18]$, pero el entrenamiento solo finalizaba con éxito un 2% de las veces (Bodén y Wiles 2002).

En cualquier caso, debe matizarse que en esta tarea la RNR no se entrena con un lenguaje no regular, ya que una parte finita de un lenguaje como $a^n b^n c^n$ es directamente regular. De lo que se trata (Bodén y Wiles 2002) al estudiar la capacidad de generalización de la red es de comprobar si esta descubre una solución distinta a la de un autómata finito.

En esta tesis extenderemos los resultados obtenidos sobre $a^n b^n c^n$ por Gers y Schmidhuber (2001) con LSTM y descenso por el gradiente para comprobar si se mantienen, empeoran o mejoran al usar el FKED.

5.3. Desambiguación categorial

Son varios los enfoques propuestos para la desambiguación categorial automática. La mayoría pueden asociarse a uno de los siguiente grupos:

1. Enfoques basados en *reglas* (Brill 1992), que se basan en el conocimiento lingüístico para determinar la categoría léxica de una palabra ambigua;

2. Enfoques *estadísticos*, que usan las estadísticas recogidas de textos completa o parcialmente etiquetados para estimar la verosimilitud de cada posible interpretación de una frase o fragmento de esta y escoger la desambiguación más verosímil; los más representativos de estos métodos son los modelos ocultos de Markov (MOM) (Cutting et al. 1992; Rabiner 1989) entrenados mediante el clásico algoritmo de Baum y Welch; este método se discute con detalle en el apéndice A.

3. Evidentemente, son también posibles los enfoques *híbridos* que combinan características de los dos anteriores.

Todas las técnicas neuronales se engloban principalmente dentro del segundo grupo. A continuación las repasamos brevemente.

5.3.1. Trabajos neuronales

Las RNR se han usado ampliamente desde su concepción en tareas de procesamiento del lenguaje humano. Elman (1990) fue de los primeros en hacerlo: entrenó su red recurrente simple (RRS) para predecir la siguiente palabra de un corpus sintético de oraciones gramaticales sencillas (de dos o tres palabras) generadas aleatoriamente a partir de un pequeño vocabulario de palabras no ambiguas sin etiquetar. Elman encontró que, tras el aprendizaje, el estado al que llegaba la red tras leer una palabra permitía agrupar las palabras en categorías que podían ser identificadas con las tradicionales (nombre, verbo transitivo, verbo intransitivo, etc.). Para poder realizar adecuadamente esta agrupación la red tenía que desarrollar una forma de obtener al vuelo una representación de la sintaxis de la parte de oración vista hasta ese momento. Los resultados de los trabajos de Elman tienen importantes connotaciones desde el punto de vista cognitivo: los humanos aprendemos sintaxis y morfología simplemente por nuestra exposición continua a las co-apariciones internas en oraciones no etiquetadas. Este aspecto del trabajo de Elman queda atenuado por la poca verosimilitud neurobiológica (Forcada y Carrasco 2001) de los modelos de tiempo discreto que son las RNR.

En esta tesis se estudia la aplicación de una RNR al problema de la desambiguación categorial de las palabras ambiguas de una oración a partir de la información almacenada en el estado de la red. Aunque el uso de una RNR en esta tarea no es del todo nueva (Marques y Lopes 1996), sí que es la primera vez que el problema se formula como un problema de predicción y se utiliza un corpus solo parcialmente etiquetado.

Schmid (1994) utiliza un perceptrón sin capa oculta para etiquetar textos en inglés. Las entradas a la red son una codificación de la palabra actual, las p anteriores y las f siguientes (en total $p + f + 1$ palabras). La red se entrena para dar como salida la categoría de la palabra actual. Cada palabra se representa mediante un vector en el que el componente j-ésimo es la probabilidad de la j-ésima categoría léxica dada la palabra en cuestión (estas probabilidades se obtienen mediante maxima verosimilitud, *contando* sobre un texto completamente etiquetado). En el caso especial de las p palabras precedentes, la codificación es una combinación lineal de ese vector y la salida dada por la red para la palabra correspondiente. El etiquetador usa un diccionario y un árbol de sufijos (un adivinador que determina la categoría de las palabras que no estén en el diccionario). El entrenamiento consta, en total, de 4 000 000 iteraciones sobre un corpus de 1 000 000 palabras. Los resultados se muestran para el caso $p = 3$ y $f = 2$, aunque el autor indica que los resultados son muy similares cuando $p = 2$ y $f = 1$. El modelo neuronal obtiene un 2% de mejora sobre un MOM y resultados similares a los de un modelo de trigramas. Si se utiliza una capa oculta, los resultados son a veces ligeramente mejores y otras veces ligeramente peores, por lo que esta circunstancia no parece influir mucho.

Marques y Lopes (1996) siguen un enfoque muy parecido al anterior para el portugués, pero usan un conjunto de entrenamiento mucho más pequeño: un corpus completamente etiquetado de unas 3 400 palabras y entre 5 000 y 10 000 iteraciones de entrenamiento. Además de un perceptrón con y sin capa oculta, utilizan una RRS. Como contexto usan $p = 0$ y $f = 1$. El número de categorías léxicas consideradas es 35 y los mejores resultados se obtienen con el perceptrón sin capa oculta. El artículo, sin embargo, no muestra una comparación con otros métodos.

Ma y Isahara (1997) utilizan también un enfoque similar al de Schmid (1994), en este caso para el tailandés, pero amplían el modelo neuronal y utilizan una combinación de perceptrones con una capa oculta. Cada perceptrón utiliza valores diferentes para el tamaño del contexto (p y f). Los resultados se combinan de dos maneras distintas: bien tomando la salida de la red con mayor tamaño de contexto ($p+f$) que proporciona una estimación clara para una determinada categoría (neurona con activación superior a un umbral), o bien decidiendo la categoría de la palabra solo cuando todas las redes coinciden en una concreta. El conjunto de entrenamiento consta de 3 162 frases con 8 242 palabras ambiguas y el conjunto de categorías léxicas tiene tamaño 53.

En otro de sus artículos, Ma et al. (1999) consideran un modelo de un único perceptrón con una capa oculta, pero la salida de la red es corregida mediante un conjunto de reglas de transformación como las utilizadas por Brill (1992). De esta manera la tasa de acierto del etiquetador mejora alrededor de un 1% sobre el modelo neuronal original. Se utiliza un perceptrón con capa oculta entrenado mediante retropropagación y la longitud del contexto previo y posterior (p y f) se selecciona dinámicamente (prioridad del contexto más largo). El tamaño del corpus completamente etiquetado usado en el entrenamiento es mucho menor que el de Schmid (1994): 8 322 frases con 22 311 palabras ambiguas (número de etiquetas, 47). Los resultados son ligeramente mejores que los del clasificador multineuronal de Ma y Isahara (1997).

Como ya se ha indicado, aquí se estudia un nuevo enfoque para el problema de la desambiguación de categorias léxicas. Es el primer trabajo de este tipo que utiliza una RNR y en el que la resolución del problema se plantea en terminos de predicción.[10] Nuestro enfoque guarda cierto parecido con el enfoque estadístico (en concreto con el de tipo Baum y Welch), ya que solo se requiere un texto parcialmente etiquetado[11] y, bajo ciertas condiciones, las salidas de la red se pueden interpretar como probabilidades.

5.4. Predicción de señales de voz

Existe un abanico enorme de técnicas de predicción de señales de voz, aunque, como se indicó en el apartado 1.2.4, las lineales son las más usadas con diferencia por su equilibrio entre sencillez y eficiencia.

Así, suele utilizarse un filtro FIR o un filtro IIR (véase el apartado 3.1.4) entrenados mediante alguno de los múltiples algoritmos existentes (Oppenheim y Schafer 1989; Proakis y Manolakis 1996).

5.4.1. Trabajos neuronales

Las RNR aparentan ser una buena alternativa no lineal para la predicción de voz: la memoria resultante de la existencia de conexiones recurrentes y la capacidad de adaptarse a cambios en el entorno las hace, en principio, apropiadas para señales no estacionarias como la voz.

Los únicos trabajos sobre predicción de señales de voz en tiempo real con RNR siguen la línea del trabajo de Haykin y Li (1995), que diseñaron una RNRC (véase el capítulo 3) para realizar la predicción de señales de voz. La RNRC se usa en una sucesión en cascada de predictores no lineales y lineales.

[10] Como usamos corpus no etiquetados, la red se entrena para predecir la clase de ambigüedad de la siguiente palabra y no para emitir la clase de ambigüedad o la etiqueta correcta de la palabra actual (predicción *versus* desambiguación).

[11] Aunque los trabajos neuronales citados anteriormente usaban corpus etiquetados *a mano* relativamente pequeños, el nuestro los evita por completo.

Baltersee y Chambers (1998) comprobaron que el rendimiento de la RNRC es insuficiente (comparable al de un filtro lineal) cuando se utiliza para su entrenamiento el descenso por el gradiente con RTRL y propusieron en su lugar el uso del FKED. Con este último, obtuvieron mejoras de unos 2 dB sobre un filtro lineal. Debe señalarse, sin embargo, que los resultados presentados en ambos artículos pertenecen al *caso mejor*: solo se realiza un único experimento con parámetros *ad hoc*.

Como ya se vio, la RNRC se compone de varias RPR de primer orden que comparten los mismos valores de sus pesos y que están conectadas de tal manera que la salida de una de ellas es la entrada de la siguiente. La RNRC puede considerarse como un modelo diseñado *ad hoc* (de hecho, no existen otros trabajos que las usen). No hay, sin embargo, estudios sobre el rendimiento de RNR clásicas de propósito general al aplicarlas a la predicción en línea de señales de voz. En esta tesis se compara los resultados de Baltersee y Chambers (1998) con los nuevos resultados obtenidos para algunas RNR clásicas. La predicción se realiza directamente sobre la señal de voz sin preprocesar.

6. COMPRESIÓN DE SECUENCIAS SIMBÓLICAS

Este capítulo estudia el uso de las RNR como modelo de probabilidad para un compresor aritmético de texto en lenguaje natural. Como aperitivo se estudia la compresión de secuencias más sencillas, principalmente derivadas de máquinas de estados finitos. Las ideas básicas de este capítulo se han publicado en las actas de un congreso internacional (Pérez-Ortiz et al. 2001b) y en una revista internacional (Pérez-Ortiz et al. 2001c).

La introducción a los conceptos de este capítulo puede encontrarse en los apartados 1.2.1 y 5.1.

6.1. Método

Se trata de estudiar la idoneidad de las RNR como modelo de probabilidad del siguiente símbolo de una secuencia procesada en línea, es decir, se ha de proporcionar una salida inmediata ante cada nuevo símbolo procesado sin posibilidad de reconsiderar la subsecuencia ya contemplada. La existencia de un modelo correcto de este tipo se puede utilizar para comprimir las secuencias por medio de, por ejemplo, un compresor aritmético. De hecho, puede considerarse de forma equivalente que los experimentos intentan evaluar la capacidad predictiva en línea de las RNR y que la compresión aritmética se usa únicamente para evaluar la calidad del predictor, ya que cuanto mejor es la predicción, mejor es la razón de compresión obtenida.

Podría alegarse, por tanto, que el habitual error cuadrático medio es una medida más sencilla e igualmente útil para evaluar la calidad del predictor. Sin embargo, esta medida solo es aplicable en el caso de que conozcamos las probabilidades reales de los posibles siguientes símbolos de la secuencia. En el caso de que estas probabilidades no sean conocidas (como ocurre, por ejemplo, con las secuencias textuales), el error solo puede basarse en la diferencia entre el símbolo predicho (normalmente aquel con mayor probabilidad) y el observado en el siguiente instante de tiempo. Es necesaria, por tanto, una forma de error alternativa que considere el vector completo de probabilidades estimadas: la compresión aritmética es una solución adecuada. En definitiva, la codificación aritmética es una alternativa empírica para medir la calidad de un predictor (neuronal o no) cuando no se pueden calcular los valores teóricos.

Como modelo de probabilidad de referencia para un compresor aritmético usaremos el programa de Nelson (1991) con un modelo combinado de $[0,4]$-gramas.

6.1.1. Secuencias de estados finitos

A modo de aperitivo, consideraremos unas secuencias de estructura más sencilla que las de texto. Estas secuencias se basan principalmente en máquinas de estados finitos y es previsible obtener buenos resultados con ellas, ya que se ha demostrado que las RNR son capaces de aprender fuera de línea lenguajes regulares (Cleeremans et al. 1989) e incluso emular máquinas de estados finitos (Carrasco et al. 2000).

Las secuencias de estados finitos se derivan de lo que llamaremos *fuentes secuenciales de estados finitos* (FSEF), que pueden definirse como una quíntupla $M = (\Sigma, Q, q_I, \delta, P)$, donde

- $\Sigma = \sigma_1, \ldots, \sigma_{|\Sigma|}$ es el alfabeto de entrada.
- $Q = q_1, \ldots, q_{|Q|}$ es un conjunto finito no vacío de estados.
- q_I es el estado inicial, $q_I \in Q$.
- δ es la función de transición de estado, $\delta : Q \times \Sigma \to Q$.
- P es un conjunto de probabilidades de transición de estados de la forma $p(q_j, \sigma_i)$, que indica la probabilidad de que el estado $q_j \in Q$ emita el símbolo $\sigma_i \in \Sigma$ y se realice una transición al estado $\delta(q_j, \sigma_i)$. Nótese que para todo q_j se tiene que $\sum_i p(q_j, \sigma_i) = 1$.

Una fuente secuencial de estados finitos (FSEF) puede considerarse como un autómata finito determinista (Hopcroft y Ullman 1979) en el que cada transición tiene una probabilidad, no existen estados de aceptación y la longitud de las secuencias generadas puede ser arbitrariamente larga.[1]

La figura 6.1 muestra la FSEF 1, una FSEF sencilla que se usará en los experimentos. Un buen predictor para las secuencias generadas a partir de

[1]Para ello debemos exigir que no existan en la fuente estados *de absorción*, pero esto viene forzado por la propia definición, que obliga a que la suma de las probabilidades del siguiente símbolo para un estado determinado sea uno.

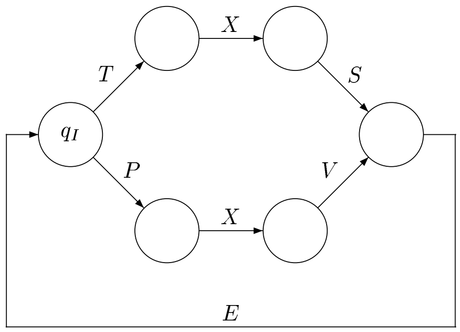


Figura 6.1: Diagrama de transiciones de la FSEF 1. El símbolo inicial aparece etiquetado como q_I. Todas las transiciones que parten de un mismo estado son equiprobables.

esta fuente debe tener una pequeña memoria a corto plazo para predecir correctamente el símbolo que sigue a X.

La figura 6.2 muestra la FSEF 2 basada en el autómata de Reber (Smith y Zipser 1989), al que se ha añadido una conexión recurrente que permita generar secuencias de longitud indeterminada.[2]

La FSEF 3, una versión continua del autómata de Reber simétrico (Smith y Zipser 1989), se muestra en la figura 6.4. Esta FSEF presenta dependencias a largo plazo (provocadas por los símbolos P o T), aunque estas apenas se manifiestan durante una secuencia.

6.1.2. Secuencias caóticas

Las fuentes secuenciales anteriores se basaban en la existencia de un estado representado por el conjunto Q. La fuente se encuentra en cada instante de tiempo en un determinado estado y a partir de él puede determinarse en cierta medida su evolución inminente. En esta sección se discute una fuente secuencial no basada en estados *finitos*: una fuente caótica.

Un *sistema dinámico* (siempre no lineal) se considera *caótico*[3] si presenta un comportamiento aperiódico (esto es, resultado de oscilaciones que no se repiten nunca, de periodo infinito) resultado de un modelo totalmente

[2] En adelante se usará en ocasiones el nombre del autómata finito subyacente para referirse a la FSEF obtenida a partir de él.

[3] Para una buena introducción a la teoría del caos y para la definición de los términos utilizados en los siguientes párrafos puede consultarse el libro de Martín et al. (1995) o el de Hilborn (2000).

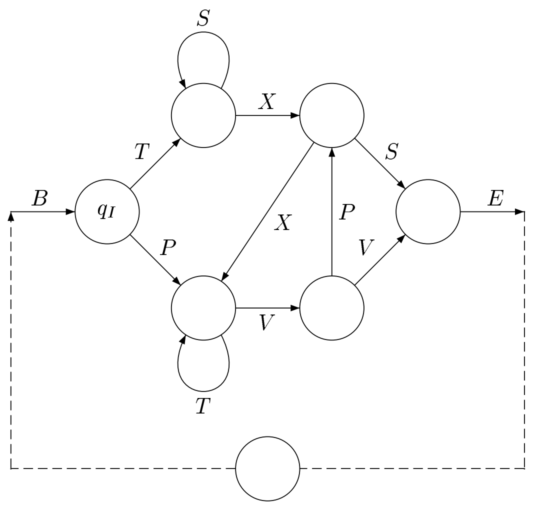


Figura 6.2: Diagrama para la FSEF 2 basada en el autómata de Reber. Todas las transiciones que parten de un mismo estado son equiprobables. Las transiciones no mostradas en el diagrama tienen probabilidad nula.

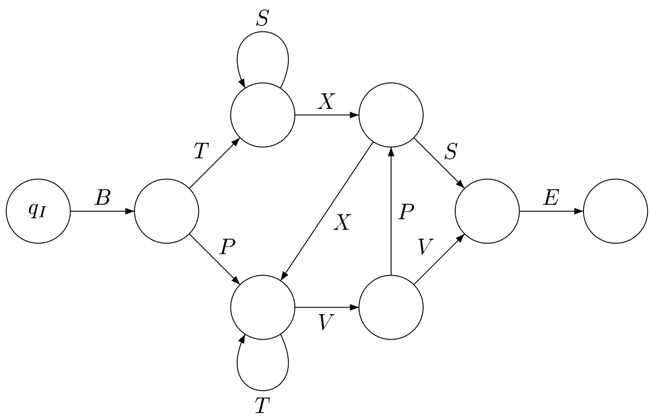


Figura 6.3: Diagrama de transiciones para el autómata de Reber estándar.

determinista y que presenta gran sensibilidad a las condiciones iniciales. La sensibilidad a las condiciones iniciales implica que existe una *divergencia exponencial* de trayectorias inicialmente muy próximas en el espacio de fases, fenómeno que se conoce como *estiramiento*. Otra propiedad existente sobre el espacio de fases y opuesta al estiramiento es el *plegamiento*, que conlleva que dos trayectorias muy lejanas pueden eventualmente acercarse.

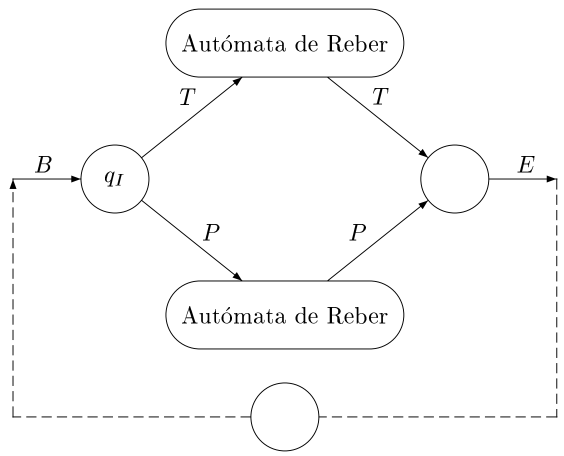


Figura 6.4: Diagramas de transiciones para la FSEF 3, obtenida a partir del autómata de Reber de la figura 6.3. Todas las transiciones que parten de un mismo estado son equiprobables. Las transiciones no mostradas en el diagrama tienen probabilidad nula. La línea inferior indica la variante continua; la discontinua se conoce normalmente como autómata de Reber simétrico.

Si representamos la trayectoria de un punto inicial bajo un sistema dinámico caótico, veremos que las dos fuerzas anteriores entran en acción de forma que se genera una estructura confinada en una región del espacio de fases que se conoce como *atractor extraño*. Las trayectorias del *espacio de fases* nunca se cortan entre sí, pues esto supondría un comportamiento periódico. Antes del descubrimiento del caos, los *ciclos límite* eran los atractores más complejos que se conocían. Hoy día se puede decir que cada tipo de sistema caótico lleva asociado atractores de características peculiares.

Las particularidades del atractor extraño lo convierten en un posible generador de secuencias caóticas. Para que estas secuencias sean simbólicas debe realizarse una división en regiones del espacio de fases, de forma que se asocie un símbolo a cada una de ellas.

En los experimentos consideraremos una secuencia obtenida de la evolución de la actividad de un láser real en regimen caótico (Weigend y Gershenfeld 1994), convertida en simbólica siguiendo el mismo procedimiento que Tiňo et al. (2000), mostrado a continuación.

El laser caótico produce subsecuencias relativamente predecibles seguidas por subsecuencias de alta impredicibilidad que requieren una memoria mayor (Tiňo et al. 2000). La secuencia numérica de activaciones del láser $y[t]$ se convierte en una secuencia simbólica $s[t]$ sobre el alfabeto $\Sigma = \{a, b, c, d\}$

mediante la transformación:

$$s[t] = \begin{cases} a & 0 \leq \Delta y[t] < \rho_2 \\ b & \rho_2 \leq \Delta y[t] \\ c & \rho_1 \leq \Delta y[t] < 0 \\ d & \Delta y[t] < \rho_1 \end{cases} \tag{6.1}$$

donde $\Delta y[t] = y[t] - y[t-1]$ y los parámetros ρ_1 y ρ_2 se hacen corresponder con los percentiles del 10% y del 90%, respectivamente, obtenidos por adelantado sobre la secuencia de activaciones del láser.

6.1.3. Textos

La dinámica de los textos en lenguaje humano es complicada. Puede considerarse un texto en lenguaje humano como una secuencia sobre un alfabeto finito Σ formado, en el caso de alfabetos occidentales, por las letras mayúsculas, las letras minúsculas, los dígitos y los signos de puntuación, entre otros. Esto hace que el tamaño del alfabeto sea demasiado grande para algunos experimentos concretos, por lo que a veces los textos reales se transforman en versiones simplificadas de los mismos, prescindiendo de los signos de puntuación o convirtiendo todas las minúsculas a mayúsculas, por ejemplo.

Nosotros consideraremos algunos textos en inglés pertenecientes a un conjunto de ensayos sobre la obra del director de cine polaco Krzysztof Kieslowski.[4] Los textos reales se convirtieron en textos sintéticos sobre un alfabeto más reducido de $|\Sigma| = 27$ símbolos ignorando los dígitos y signos de puntuación y convirtiendo todos los caracteres alfabéticos a mayúsculas sin acentuar. De esta manera los textos están formados únicamente por letras mayúsculas y espacios en blanco.

6.2. Parámetros

Los resultados que se muestran más adelante ilustran la razón de compresión (RC) en función del número de neuronas de estado, n_X. Se usó codificación exclusiva para los símbolos; los valores de n_U y n_Y dependen, por tanto, del tamaño del alfabeto correspondiente. Los modelos de RNR son la RRS y la RPR, entrenados tanto con descenso por el gradiente como con el FKED con las derivadas calculadas según RTRL. Nótese que en los experimentos $n_X = 0$ significa que no hay neuronas de estado; en el caso de la RRS esto implica, además, que solo se ajustan los sesgos de las neuronas de salida.

[4] Estos textos se encuentran en `http://www.petey.com/kk/`.

La tasa de aprendizaje del descenso por el gradiente es $\alpha = 0.9$ y el momento es $\gamma = 0.4$, valores que fueron escogidos tras experimentos preliminares. En el caso del FKED se tomaron los siguientes valores de los parámetros del algoritmo:

- $Q[t] : 10^{-2} \stackrel{T=1000}{\longrightarrow} 10^{-6}$
- $R[t] : 100 \stackrel{T=1000}{\longrightarrow} 3$
- $P[0] = 100I$

Los resultados neuronales mostrados son la media aritmética de 7 experimentos distintos sobre cada secuencia; la varianza es muy pequeña en todos los casos y no se mostrará. Los valores iniciales de los pesos se tomaron de una distribución uniforme en $[-0.2, 0.2]$. Consideraremos los tres tipos de secuencias comentados anteriormente: secuencias generadas por máquinas de estados finitos, secuencias caóticas y textos en lenguaje humano.

También se realizaron experimentos con la RTR y la red NARX; como las RC obtenidas son similares, bien a las de la RRS, bien a las de la RPR (dependiendo de la secuencia y de los parámetros concretos considerados), omitiré los resultados de aquellas.

6.3. Resultados

6.3.1. Secuencias de estados finitos

El cuadro 6.1 muestra los resultados obtenidos con `gzip`, `bzip2` y el compresor aritmético de Nelson (1991) con $[0, 4]$-gramas para secuencias de longitud 20 000 derivadas de las FSEF comentadas anteriormente. Ya que tenemos las máquinas generadoras de estas secuencias, podemos también utilizar el compresor aritmético con el modelo exacto de probabilidad. Debido a las propiedades de la codificación aritmética, esto dará la mejor RC posible. Los resultados de este modelo se muestran en el cuadro 6.1 en la fila etiquetada como "aritmético exacto".

Las figuras 6.5 a 6.7 muestran los resultados obtenidos por la RRS y la RPR. Como se puede ver, el número de neuronas de estado afecta a la RC obtenida, aunque para valores de $n_X \geq 10$ esta influencia no es muy significativa. Ambos modelos, RRS y RPR, dan resultados comparables. El FKED proporciona RC cercanas a las del modelo de $[0, 4]$-gramas (que, a su vez, son cercanas a las del compresor exacto), mientras que las del descenso por el gradiente son inferiores.

	FSEF 1	FSEF 2	FSEF 3
Aritmético exacto	11.93	3.99	5.10
$[0, 4]$-gramas	11.29	3.83	4.23
`gzip`	6.10	2.27	2.46
`bzip2`	8.89	3.34	3.63

Cuadro 6.1: Razones de compresión obtenidas para las FSEF con compresores no neuronales.

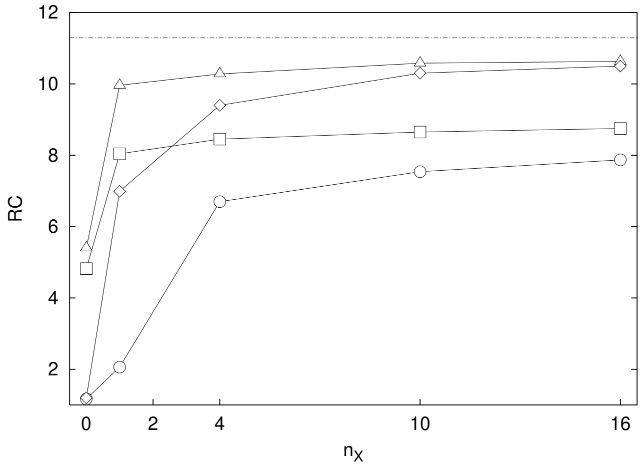


Figura 6.5: Razones de compresión para la secuencia derivada de la FSEF 1. Se muestran los resultados con RPR y descenso por el gradiente (□), RRS y descenso por el gradiente (○), RPR y FKED (△), y RRS y FKED (◊). La línea constante es la razón de compresión del modelo de $[0, 4]$-gramas, indicada también en el cuadro 6.1.

Aunque no se mostrará aquí, es posible encontrar para cada secuencia y modelo particular un conjunto de valores de los parámetros ajustables que acercan aún más las RC neuronales a las del modelo exacto.

6.3.2. Secuencias caóticas

Los resultados para la secuencia del láser caótico de longitud 10 000 con las RNR se muestran en la figura 6.8. El modelo de $[0, 4]$-gramas da una RC de 2.73 para esta secuencia. Evidentemente, en este caso no cabe hablar de un modelo de probabilidad exacto.

Al usar descenso por el gradiente, las diferencias entre la RRS y la RPR son mayores que antes. De nuevo, el FKED supera con creces los resultados del descenso por el gradiente. En cualquier caso, la RPR entrenada con este

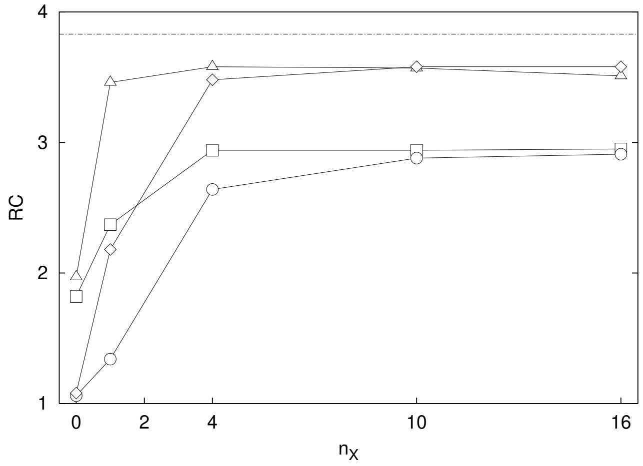


Figura 6.6: Razones de compresión para la secuencia derivada de la FSEF 2. Se muestran los resultados con RPR y descenso por el gradiente (□), RRS y descenso por el gradiente (○), RPR y FKED (△), y RRS y FKED (◊). La línea constante es la razón de compresión del modelo de $[0, 4]$-gramas, indicada también en el cuadro 6.1.

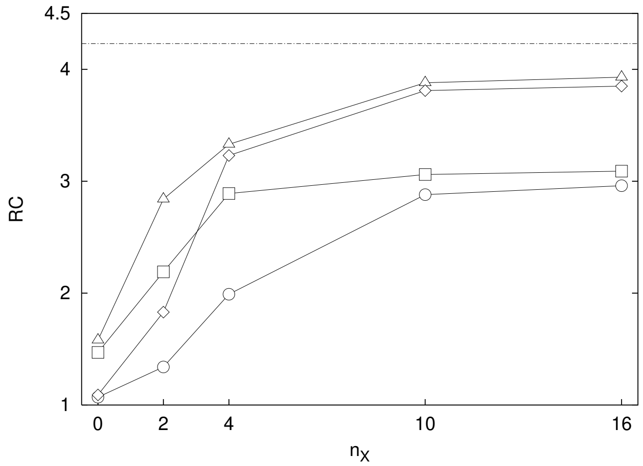


Figura 6.7: Razones de compresión para la secuencia derivada de la FSEF 3. Se muestran los resultados con RPR y descenso por el gradiente (□), RRS y descenso por el gradiente (○), RPR y FKED (△), y RRS y FKED (◊). La línea constante es la razón de compresión del modelo de $[0, 4]$-gramas, indicada también en el cuadro 6.1.

algoritmo da RC (con $n_X > 4$) similares a las del modelo de $[0, 4]$-gramas, y el FKED (independientemente del modelo neuronal) consigue razones mucho mayores (cercanas a 4).

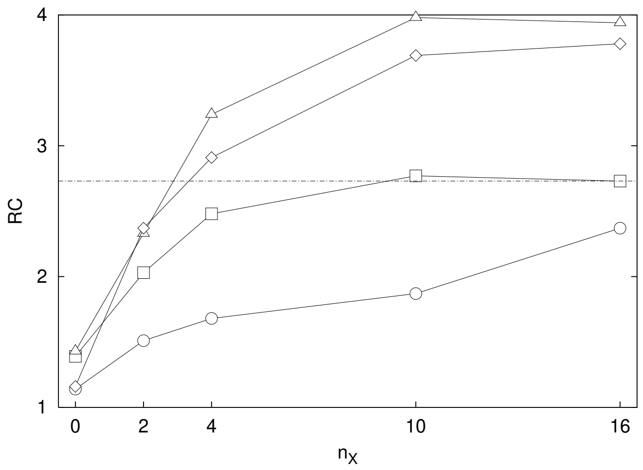


Figura 6.8: Razones de compresión para la secuencia del láser caótico. Se muestran los resultados con RPR y descenso por el gradiente (□), RRS y descenso por el gradiente (○), RPR y FKED (△), y RRS y FKED (◊). La línea constante es la razón de compresión del modelo de $[0, 4]$-gramas.

	`decalog1`	`kiesdis3`	`vidwchdg`
$[0, 4]$-gramas	1.74	1.75	1.85
`gzip`	1.53	1.58	1.60
`bzip2`	1.71	1.72	1.83

Cuadro 6.2: Razones de compresión para las secuencias de texto con compresores no neuronales.

6.3.3. Textos en lenguaje humano

Ahora aplicaremos la misma estrategia que en los apartados anteriores para estudiar la influencia del algoritmo de aprendizaje y del número de estados en las razones de compresión obtenidas para tres textos en inglés. El cuadro 6.2 muestra las RC con `gzip`, `bzip2` y el modelo de $[0, 4]$-gramas. Obviamente, tampoco existe ningún modelo de probabilidad exacto en este caso. El número de símbolos de cada secuencia es: `decalog1`, 19 385; `kiesdis3`, 18 666; y `vidwchdg`, 62 648.

Las figuras 6.9 a 6.11 (prácticamente idénticas) muestran las RC obtenidas por las RNR para las tres secuencias de texto consideradas. Puede observarse fácilmente que ambos modelos tienen problemas para aprender las estadísticas de las secuencias: el valor de n_X no tiene una influencia clara en la RC obtenida con la RPR. En el caso de una RPR, esto implica que la red está desarrollando una especie de modelo de bigramas: solo el

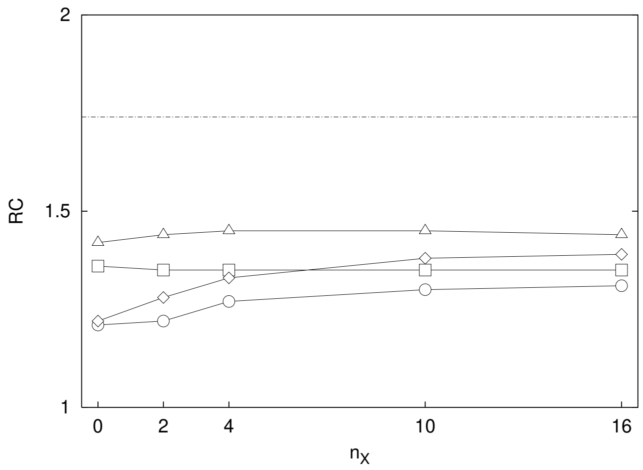


Figura 6.9: Razones de compresión para la secuencia en inglés `decalog1`. Se muestran los resultados con RPR y descenso por el gradiente (□), RRS y descenso por el gradiente (○), RPR y FKED (△), y RRS y FKED (◊). La línea constante es la razón de compresión del modelo de $[0, 4]$-gramas, indicada también en el cuadro 6.2.

símbolo actual se considera al predecir el siguiente. La RRS, por otro lado, usa sus neuronas de estado, ya que a valores mayores de n_X las RC son también mayores, pero los resultados son peores que los de la RPR. Debe destacarse, no obstante, que los resultados obtenidos con las otras técnicas son solo ligeramente mejores.

El FKED, por otra parte, explota positivamente (al menos parcialmente) el número de neuronas de estado. A pesar de estos resultados ligeramente mejores (en consonancia con lo comprobado hasta ahora para este algoritmo), las RC son todavía mucho menores que las obtenidas con el modelo de n-gramas.

6.4. Discusión

El algoritmo de entrenamiento del FKED supera al de descenso por el gradiente cuando se trata de predecir en línea los símbolos de una secuencia simbólica. En el caso de las secuencias de estados finitos, los resultados del FKED son comparables a los obtenidos con n-gramas y están cerca del óptimo. Al procesar la secuencia caótica, incluso los resultados del descenso por el gradiente son similares a los del modelo de $[0, 4]$-gramas y el FKED los supera notablemente. Sin embargo, al trabajar con secuencias en lenguaje humano, los resultados del FKED son mucho peores que los de un modelo

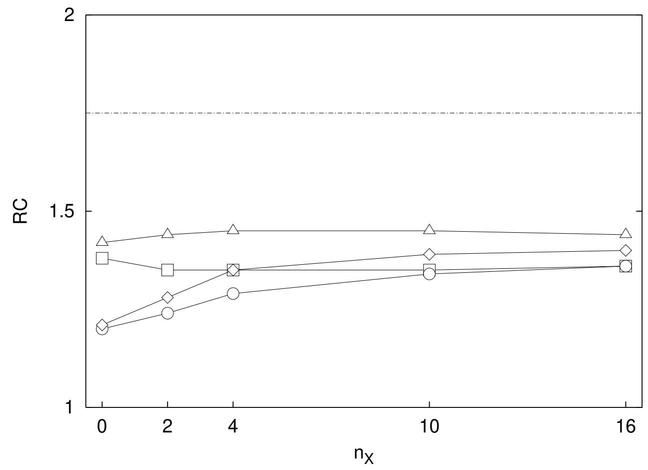


Figura 6.10: Razones de compresión para la secuencia en inglés `kiesdis3`. Se muestran los resultados con RPR y descenso por el gradiente (□), RRS y descenso por el gradiente (○), RPR y FKED (△), y RRS y FKED (◊). La línea constante es la razón de compresión del modelo de $[0,4]$-gramas, indicada también en el cuadro 6.2.

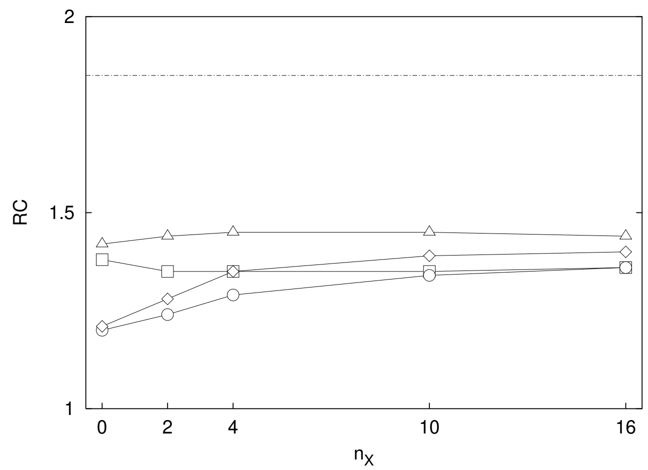


Figura 6.11: Razones de compresión para la secuencia en inglés `vidwchdg`. Se muestran los resultados con RPR y descenso por el gradiente (□), RRS y descenso por el gradiente (○), RPR y FKED (△), y RRS y FKED (◊). La línea constante es la razón de compresión del modelo de $[0,4]$-gramas, indicada también en el cuadro 6.2.

de $[0,4]$-gramas. La predicción sobre textos en lenguaje humano parece ser una tarea difícil para ambos algoritmos.

Debe tenerse en cuenta, sin embargo, que el número de parámetros de un modelo combinado de n-gramas como el utilizado en el compresor de Nelson (donde $n = 4, 3, 2, 1, 0$) es muy superior al de las RNR utilizadas en este trabajo. El número teórico máximo de parámetros de un modelo de n-gramas es $|\Sigma|^n$ donde Σ es el alfabeto de las secuencias sobre las que trabaja el predictor.[5] La RRS utilizada en los experimentos, por ejemplo, tiene un número de pesos igual a $(|\Sigma| \cdot n_X)^2 + n_X^2 + n_X + |\Sigma|$; a este número de parametros habría que añadirle los utilizados por el algoritmo de entrenamiento correspondiente.

Puede afirmarse, en definitiva, que el FKED supera al descenso por el gradiente cuando se trata de predecir los símbolos de una secuencia simbólica. El FKED, sin embargo, tiene una complejidad temporal superior a la del descenso por el gradiente. La dinámica de los textos reales, en cualquier caso, parece ser demasiado compleja para ser aprendida en línea correctamente por una RNR.

Finalmente, debe tenerse en cuenta que los experimentos realizados con las máquinas de estados finitos no permiten detectar si la RNR ha aprendido completamente las máquinas correspondientes. Por ejemplo, la FSEF 3 presenta dependencias a largo plazo, como veremos en el capítulo siguiente, que se manifiestan relativamente de tarde en tarde. Estas dependencias no suelen ser aprendidas adecuadamente por las RNR consideradas en este capítulo, pero debido a su aparición ocasional, este hecho apenas aparecerá reflejado en las RC obtenidas.

Alternativas de predicción de textos. En todos los experimentos con textos en lenguaje natural realizados con RNR en este capítulo, el valor 1.5 parece ser una cota superior de las RC obtenibles (véase las figuras 6.9 a 6.11). Aunque es necesario un estudio más profundo de los motivos de este comportamiento, los resultados apuntan a que la RNR no es capaz de desarrollar un modelo adecuado de los contextos de la secuencia similar al que desarrolla un modelo de $[0, 4]$-gramas. Sin embargo, el rendimiento de las RNR no es tan malo si se compara con otros modelos como el que veremos a continuación.

Forcada (2001) ha estudiado la construcción de autómatas estocásticos de estados finitos para implementar un sistema de predicción textual similar al que incorporan los teléfonos móviles actuales a la hora de escribir un mensaje. Aunque sus resultados son para el catalán, lo cierto es que la razón

[5] A la hora de una implementación real, el espacio requerido es inferior a $|\Sigma|^n$, ya que muchos de los parámetros nunca se alteran y conservan su valor inicial. Es posible, por tanto, encontrar formas eficientes de almacenar estos parámetros, por ejemplo, utilizando árboles (Nelson 1991). Aún así el número de parámetros utilizados es con diferencia superior al de las RNR utilizadas en los experimentos.

de compresión obtenida por estos sistemas es muy similar a la obtenida en este capítulo para RNR.

En efecto, Forcada (2001) considera un alfabeto de aproximadamente 40 símbolos y un teclado de 10 teclas. Si no se utiliza predicción de ningún tipo, el número de bits por símbolo al codificar un mensaje será $\log_2 40 = 5.32$. Si se considera un predictor, en lugar de enviar los distintos símbolos codificados, se puede considerar enviar la secuencia de teclas pulsadas por el usuario para escribir el mensaje.

Pues bien, Forcada (2001, cuadro 3) asegura que el número medio de pulsaciones por símbolo realizadas al utilizar una estrategia predictiva es 1.11. Como la codificación de cada una de las 10 teclas requiere $\log_2 10 = 3.32$, el número medio de bits por símbolo al codificar un mensaje será $1.11 \times 3.32 = 3.67$. Luego la RC será de:

$$\frac{5.32}{3.67} = 1.45$$

Este resultado se aproxima bastante a los obtenidos por las RNR en los experimentos de este capítulo.

7. INFERENCIA DE LENGUAJES CON DEPENDENCIAS A LARGO PLAZO

En este capítulo se estudia la inferencia en línea de un lenguaje regular y la inferencia fuera de línea de un lenguaje sensible al contexto, ambas planteadas como un problema de predicción. Los dos lenguajes presentan marcadas dependencias a largo plazo por lo que no suelen ser bien aprendidos por las RNR tradicionales. Los resultados de estas se comparan con los de la red LSTM, teóricamente adecuada para manejar este tipo de dependencias. Los contenidos de este capítulo, publicados en varios congresos internacionales (Gers et al. 2002b; Pérez-Ortiz et al. 2002b; Gers et al. 2002a) y aceptados con modificaciones en una revista internacional (Pérez-Ortiz et al. 2002a), muestran el primer uso de LSTM en tareas de predicción en línea, así como la primera ocasión en que el FKED se aplica a este modelo.

Los experimentos de inferencia desarrollados en este capítulo pueden dividirse en dos grupos: relativos a la inferencia en línea de un lenguaje regular derivado del autómata de Reber simétrico (Smith y Zipser 1989) y relativos a la inferencia fuera de línea del lenguaje sensible al contexto $a^n b^n c^n$. Por ello, cada uno de los siguientes apartados aparece dividido en estos dos bloques.

Como introducción a los contenidos de este capítulo, puede consultarse las secciones 1.2.2 y 5.2.

7.1. Método

Aprendizaje del lenguaje de Reber

En este experimento, se usa principalmente LSTM con compuertas de olvido, pero sin conexiones de mirilla (véase el capítulo 3), para predecir el siguiente símbolo sobre una secuencia simbólica continua con dependencias a largo plazo. También se muestran los resultados obtenidos con RNR tradicionales. El énfasis se pone en el procesamiento en línea *puro*, es decir, los pesos se actualizan símbolo a símbolo y la secuencia de entrada es única y no está dividida en subsecuencias claramente demarcadas.

Gers et al. (2000) estudian un problema similar, pero abordado desde un punto de vista próximo a lo que hemos llamado en el apartado 4.1.1 *aprendizaje en línea por secuencias*. En sus experimentos, los pesos se actualizan tras la presentación de cada símbolo: una estrategia propia del aprendizaje en línea puro. Sin embargo, la secuencia actual de entrada se rechaza tan pronto como la red comete un error sobre ella; a continuación, la red se inicializa y el procesamiento continúa con una nueva secuencia: esta forma de presentar la información a la red es más propia del aprendizaje fuera de línea. Además, la red se evalúa con los pesos congelados. El enfoque adoptado en su trabajo puede considerarse, por tanto, a medio camino entre ambos tipos de aprendizaje.

Aquí aplicaremos la red LSTM y la RRS al mismo tipo de secuencias, pero con un enfoque en línea *puro*: existe un único flujo de entrada, el aprendizaje continúa aunque la red cometa errores, y el entrenamiento y la evaluación no están divididos en fases separadas. En el aprendizaje en línea puro, es importante que la red sea capaz de desarrollar un modelo sin poder apoyarse en las inicializaciones que permiten “arrancar” las redes desde un determinado punto fijo. Podemos decir que este tipo de entrenamiento es en “caída libre”, ya que carece siquiera de este mínimo punto de apoyo.

Por otro lado, en el trabajo de Gers et al. (2000) solo se utilizaba el algoritmo de descenso por el gradiente. Aquí se aplicará el FKED por primera vez al modelo LSTM.

La secuencia de entrada se genera a partir de la variante continua del autómata de Reber simétrico (FSEF 3), que se muestra en la figura 6.4. Como se aprecia en la figura, las dependencias a largo plazo aparecen porque, en ocasiones, determinar si el siguiente símbolo de la secuencia es P o T exige considerar la historia antigua de la secuencia.

Contaremos el número de símbolos que necesita la red LSTM para generar predicciones sin error durante al menos 1000 símbolos, número que representaremos mediante β_{1000}; aquí “sin error” quiere decir que el símbolo correspondiente a la neurona ganadora sea uno de los símbolos para los que existe una transición desde el estado actual del autómata de Reber simétrico.

Por otro lado, al aprender en línea, la presencia recurrente de determinadas subsecuencias hace que la red olvide la historia pasada y confíe en las observaciones más recientes.[1] Por ello, tras un periodo de entrenamiento

[1] De hecho, esto es lo que uno esperaría de un modelo que es capaz de actuar correctamente bajo entornos no estacionarios.

inicial es habitual que las RNR (LSTM en especial) cometan algunos errores aislados, tras lo que siguen realizando predicciones correctas. Para tener una medida más tolerante de la calidad predictiva del sistema, medimos también el momento en el que se produce el error n-ésimo tras las 1000 primeras predicciones sucesivas sin error: consideraremos dos valores para n, a saber, 1 y 10, que representaremos mediante β^{1}_{1000} y β^{10}_{1000}, respectivamente.

Aprendizaje de $a^n b^n c^n$

En este experimento utilizaremos la red LSTM con compuertas de olvido y conexiones de mirilla[2] para aprender y generalizar el lenguaje sensible al contexto $a^n b^n c^n$. Como se comentará más adelante, el modo de entrenamiento será en línea por secuencias o fuera de línea por secuencias, según el algoritmo de entrenamiento concreto utilizado.

La red observará secuencialmente, símbolo a símbolo, muestras positivas del lenguaje. El objetivo en cada iteración es predecir los posibles símbolos siguientes, incluido el símbolo de fin de cadena \$, que es también un símbolo especial de arranque con el que comienzan todas las secuencias.[3] Cuando es posible más de un símbolo en el siguiente paso, todos ellos deben ser predichos adecuadamente y ninguno de los otros. Por ejemplo, en el caso de $n = 3$, las entradas y salidas deseadas serían:

Entrada:	S	a	a	a	b	b	b	c	c	c
Salida deseada:	a/\$	a/b	a/b	a/b	b	b	c	c	c	\$

Una secuencia es aceptada si todas las predicciones realizadas sobre ella son correctas y rechazada en caso contrario. Diremos que el sistema ha aprendido el lenguaje hasta las secuencias de longitud L, si es capaz de predecir todas las secuencias con tamaño menor o igual que L.

En cuanto al entrenamiento y la evaluación, ambos se turnan: cada 1000 secuencias de entrenamiento, los pesos de la red se congelan y se procede a la evaluación. Los conjuntos de aprendizaje y evaluación incorporan todas las secuencias válidas (solo muestras positivas) con longitud menor o igual a $3n$. El entrenamiento termina cuando en un turno o época se aceptan todas las cadenas del conjunto de entrenamiento. Los resultados mostrados son la media aritmética de los obtenidos con 10 redes entrenadas independientemente con diferentes inicializaciones de los pesos (las mismas para cada experimento). Llamaremos *conjunto de generalización* al conjunto de evaluación más grande que es aceptado por la red.

[2]Esta tarea no puede aprenderse sin las conexiones de mirilla.

[3]Nótese que \$ no es un *separador*, sino un *finalizador* de secuencias, ya que los modos de entrenamiento empleados reinician la red tras cada secuencia.

Estudiaremos el comportamiento de la red LSTM con dos tipos de conjuntos de entrenamiento para $a^n b^n c^n$:

1. con $n \in [1, N]$,
2. con $n \in [N-1, N]$.

Para valores grandes de N, el segundo caso es más difícil, ya que no existe el apoyo de las secuencias más cortas, que son más fáciles de aprender al atenuar las dependencias a largo plazo. Aquí nos centraremos en $n \in [1, 10]$ y $n \in [20, 21]$.

7.2. Parámetros

Aprendizaje del lenguaje de Reber

Siguiendo lo propuesto por Gers et al. (2000), la red LSTM tiene $n_M = 4$ bloques de memoria con $n_C = 2$ celdas cada uno. Como usamos codificación exclusiva para los símbolos y como el tamaño del alfabeto del autómata de Reber es $|\Sigma| = 7$, consideramos una red LSTM con $n_U = n_Y = 7$. Los sesgos de las compuertas de entrada W^ϕ y de salida W^γ se inicializan por bloques: -0.5 para el primer bloque, -1 para el segundo, -1.5 para el tercero, etc.[4] Los sesgos de las compuertas de olvido W^λ se inicializan con valores simétricos: 0.5 para el primer bloque, 1 para el segundo, etc. El resto de los pesos de la red se inicializan de forma aleatoria según una distribución uniforme en $[-0.2, 0.2]$.

Como existen conexiones directas entre las entradas y las unidades de salida, la red LSTM se puede considerar una máquina neuronal de estados de Mealy.

La función de activación g_Z se toma como $g_Z(x) = g_T(x) = \tanh(x)$; $g_M(x)$ es la función identidad; y la función de activación de las compuertas es la función logística $g_C(x) = g_L(x)$.

Los algoritmos de entrenamiento considerados son el descenso por el gradiente y el FKED. Las derivadas se calculan según RTRL en el caso de la RRS y según lo indicado en 4.12.1 para la red LSTM. En el caso del descenso por el gradiente, la tasa de aprendizaje es $\alpha = 0.5$ y el momento es $\gamma = 0$. En el caso del FKED, los valores para los parámetros libres del algoritmo son:

- $Q[t] : 10^{-2} \xrightarrow{T=8000} 10^{-6}$

[4] Véase el apartado 3.2 para una discusión sobre esta inicialización.

- $R[t] : 100 \stackrel{T=8000}{\longrightarrow} 3$
- $P[0] = 100I$

Los valores anteriores proporcionaron buenos resultados en experimentos preliminares.

Para comparar los resultados de LSTM con los de una RNR tradicional se considera una RRS con un número de parámetros muy similar a los del modelo LSTM: 364 parámetros ajustables tomando $n_X = 13$. Las funciones de activación g_Y y g_X son la función logística $g_L(x)$ y los parámetros de entrenamiento los mismos que los indicados arriba para la red LSTM.

Aprendizaje de $a^n b^n c^n$

Siguiendo a Gers y Schmidhuber (2001), consideramos una red LSTM configurada como una máquina neuronal de estados de Mealy con $n_M = 2$ bloques de memoria con $n_C = 1$ celda cada uno.[5] Los sesgos de las compuertas de entrada, olvido y salida (W^ϕ, W^λ y W^γ, respectivamente) se inicializan a -1.0, $+2.0$ y -2.0, respectivamente (aunque los valores exactos de esta inicialización no son muy críticos). El resto de pesos se inicializan aleatoriamente según una distribución uniforme en $[-0.1, 0.1]$. Las funciones de activación g_Z y g_M son la función identidad. La función de activación de las neuronas de salida es una función logística con rango $(-2, 2)$, esto es, $g_Y(x) = 4g_L(x) - 2$.

Los símbolos se codifican mediante codificación exclusiva, con la salvedad de que en lugar de vectores unitarios se consideran vectores cuyos componentes son $+1$ o -1 para ampliar el rango. Como tenemos los tres símbolos a, b, c y el marcador especial, queda $n_U = n_Y = 4$. Con $+1$ se indica que un determinado símbolo está activado y con -1 que no lo está; la frontera de decisión para la salida de la red es 0.

En el caso del algoritmo de descenso por el gradiente, los pesos se actualizan fuera de línea tras cada secuencia. Se consideran dos formas de descenso por el gradiente: con un momento (Plaut et al. 1986) de valor $\gamma = 0.99$ o sin él. Los resultados se muestran para diversos valores de la tasa de aprendizaje α. Como máximo se presentan 10^7 secuencias de entrenamiento y se evalúa con secuencias de longitud máxima menor que 1500 ($n \leq 500$).

En el caso del FKED se utiliza un entrenamiento también por secuencias, pero esta vez en línea. Los parámetros utilizados en el FKED son para el caso $n \in [1, 10]$:

[5] Esto hace que el número total de pesos a ajustar sea 84 (72 conexiones entre unidades y 12 sesgos).

- $Q[t] : 5 \cdot 10^{-3}$
- $R[t] : 100 \stackrel{T=1000}{\longrightarrow} 1$

y para el caso $n \in [20, 21]$:

- $Q[t] : 5 \cdot 10^{-3} \stackrel{T=1000}{\longrightarrow} 10^{-6}$
- $R[t] : 100 \stackrel{T=1000}{\longrightarrow} 1$

Los valores anteriores proporcionaron buenos resultados en experimentos preliminares, pero no son críticos en el sentido de que hay un amplio rango de valores que resultan en un redimiento similar. La influencia del otro parámetro, la matriz de covarianza del error inicial $P[0]$, se estudiará más adelante. El máximo número de secuencias presentadas en el caso del FKED es 10^5 y la evaluación se realiza con secuencias de longitud máxima menor o igual a 30 000 ($n \leq 10\,000$).

7.3. Resultados

Aprendizaje del lenguaje de Reber

Resultados con la red LSTM con descenso por el gradiente. El cuadro 7.1 muestra los resultados para 9 secuencias diferentes sobre 9 redes LSTM inicializadas independientemente y entrenadas con el algoritmo original de descenso por el gradiente. En uno de los casos (fila 5), no se hallaron subsecuencias de predicción correctas (para 1000 símbolos seguidos) tras los primeros 1 000 000 símbolos; esto se indica en el cuadro mediante $1\,000\,000^+$.

Es de destacar que el número medio de símbolos necesarios para aprender a predecir sostenidamente con el método en línea puro (miles de símbolos) es mucho más pequeño que los necesitados por Gers et al. (1999; 2000) (millones de símbolos). Esto reafirma la rapidez de aprendizaje del entrenamiento en línea.

Resultados de LSTM con el FKED. El rendimiento es mucho mejor con el FKED. El tiempo necesario para obtener 1000 predicciones seguidas sin error, β_{1000}, es generalmente inferior que el usado por el algoritmo original de entrenamiento basado en el descenso por el gradiente, lo que indica una convergencia más rápida (compárese con el cuadro 7.2). Sin embargo, el número de símbolos procesados antes del décimo error, β_{1000}^{10}, es también inferior, indicando una degradación del rendimiento más rápida. El FKED parece incrementar la capacidad de aprendizaje en línea a la vez que reduce

Red	β_{1000}	β^{1}_{1000}	β^{10}_{1000}
1	39229	143563	178229
2	102812	111442	144846
3	53730	104163	141801
4	54565	58936	75666
5	1000000^{+}	–	–
6	111483	113715	136038
7	197748	199445	235387
8	54629	123565	123595
9	85707	86742	92312

Cuadro 7.1: Iteraciones necesarias para que la red LSTM (entrenada en línea con el algoritmo original de descenso por el gradiente) consiga 1000 predicciones correctas seguidas (predicción sostenible). También se muestra el número de iteraciones antes de que aparezca un error y antes de que aparezcan 10 errores. La red número 5 no mostró ninguna predicción sostenible antes del símbolo 1 000 000.

Red	β_{1000}	β^{1}_{1000}	β^{10}_{1000}
1	29304	30347	30953
2	19758	25488	322980
3	20487	22235	24106
4	26175	27542	33253
5	18015	19365	22241
6	16667	29826	1000000^{+}
7	23277	24796	26664
8	1000000^{+}	–	–
9	29742	31535	594117

Cuadro 7.2: Iteraciones necesarias para que LSTM (entrenada en línea con el FKED) consiga 1000 predicciones seguidas correctas. También se muestra el número de iteraciones antes del siguiente error y de los siguientes 10 errores. La red 8 no obtuvo una predicción sostenible antes de 1 000 000 de símbolos. La fila 6 muestra un resultado especialmente bueno: solo se producen 3 errores antes del símbolo 1 000 000.

las capacidad de memoria a largo plazo de LSTM. Existen, no obstante, 3 casos destacables (filas 2, 6 y 9 del cuadro 7.2), en los que hace falta una subsecuencia muy larga (cientos de miles de símbolos) para que se produzca el décimo error.

Resultados con la RRS. La RRS entrenada con descenso por el gradiente con RTRL no puede obtener predicciones sostenibles (β_{1000}), al menos después de probar con distintas inicializaciones de los pesos y tras secuencias

Red	β_{1000}	β_{1000}^{1}	β_{1000}^{10}
1	132827	134035	136155
2	115462	117118	120274
3	57363	58989	61403
4	265088	266227	289391
5	148496	149894	153442
6	357745	359807	363163
7	171768	173997	193946
8	58077	59222	65463
9	154269	155443	158609

Cuadro 7.3: Iteraciones necesarias para que la RRS con $n_X = 13$ entrenada en línea con el FKED consiga 1000 predicciones seguidas correctas. También se muestra el número de iteraciones antes del siguiente error y de los siguientes 10 errores.

extremadamente largas de 10^6 símbolos.[6] Las dependencias a largo plazo son las culpables de este comportamiento, ya que desde muy pronto (a partir del símbolo 3000 o 4000, aproximadamente) la RRS predice correctamente en todos los casos excepto cuando aparecen los símbolos P y T del extremo derecho del autómata.[7]

Incluso 100 predicciones correctas seguidas ocurrían muy de tarde en tarde, aunque, por otro lado, no es muy ilustrativo considerar valores como el de β_{100}, ya que hay una probabilidad no despreciable de que la predicción a largo plazo corresponda al mismo símbolo durante este intervalo y la RNR actúe correctamente aun ignorando esa dependencia.

El FKED aplicado a la RRS, sin embargo, sí que permitía predicciones sostenibles sin error del tipo β_{1000} como puede observarse en el cuadro 7.3.

En cualquier caso, hay un detalle importante que diferencia el aprendizaje realizado por la RRS del realizado por LSTM. Cuando la RRS realiza correctamente la predicción de uno de los dos símbolos, P o T, afectado por dependencias a largo plazo, las probabilidades emitidas por la red para ambos están en torno a $1/2$ y el símbolo escogido es el correspondiente al de probabilidad mayor.[8] El comportamiento de LSTM es distinto, como

[6] El mismo comportamiento se observó con la RPR.

[7] Este aprendizaje rápido de las dependencias a corto plazo se puede observar también en el caso de la red LSTM.

[8] Esto significa que de haber impuesto una política más conservadora para el símbolo predicho por la RNR, la RRS con el FKED no habría podido realizar predicciones sostenibles.

	$\gamma = 0$		$\gamma = 0.99$	
α	**Secuencias** $[\times 10^3]$	**% aprendido**	**Secuencias** $[\times 10^3]$	**% aprendido**
10^{-1}	–	0	–	0
10^{-2}	–	0	–	0
10^{-3}	68	100	–	0
10^{-4}	351	100	20	90
10^{-5}	3562	100	45	90
10^{-6}	–	0	329	100
10^{-7}	–	0	3036	100

Cuadro 7.4: Resultados con LSTM y descenso por el gradiente para $a^n b^n c^n$ con conjuntos de entrenamiento con $n \in [1, 10]$ y con distintas tasas de aprendizaje α y valores del momento γ.

veremos, ya que la red “sabe” lo que viene después y la probabilidad del símbolo predicho es muy superior a la del resto.

Finalmente, analizando la subsecuencia correspondiente a una predicción sostenible con la RRS, puede observarse que la red falla en cuanto la separación entre el evento que causa la dependencia a largo plazo y su manifestación en la secuencia es ligeramente mayor que 10 (por ejemplo, 20); en este caso las probabilidades ni siquiera están en torno a 1/2.

Aprendizaje de $a^n b^n c^n$

Resultados de LSTM con descenso por el gradiente. Cuando se usa el descenso por el gradiente, la red LSTM aprende las dos clases de conjuntos de entrenamiento y generaliza bien. Los datos relativos al aprendizaje del conjunto de entrenamiento se muestran en los cuadros 7.4 y 7.5 para distintos valores de la tasa de aprendizaje α. En ellos se muestra el número medio de secuencias necesarias hasta conseguir una fase de entrenamiento (1000 secuencias) sin errores y el porcentaje de experimentos en los que se logró. En cuanto a la generalización, para el caso $n \in [1, 10]$, la mejor generalización es $n \in [1, 52]$ y la generalización media es $n \in [1, 28]$; por otro lado, para el caso $n \in [20, 21]$, la mejor generalización es $n \in [10, 27]$ y la generalización media es $n \in [17, 23]$.

La red LSTM funciona adecuadamente para un amplio rango de tasas de aprendizaje (unos tres órdenes de magnitud) como puede verse en los cuadros 7.4 y 7.5. La utilización del momento ayuda claramente a mejorar la velocidad de aprendizaje, además de tolerar el mismo rango para α.

	$\gamma = 0$		$\gamma = 0.99$	
α	**Secuencias** $[\times 10^3]$	**% aprendido**	**Secuencias** $[\times 10^3]$	**% aprendido**
10^{-1}	–	0	–	0
10^{-2}	–	0	–	0
10^{-3}	1170	30	–	0
10^{-4}	7450	30	–	0
10^{-5}	1205	20	127	20
10^{-6}	–	0	1506	20
10^{-7}	–	0	1366	10

Cuadro 7.5: Resultados con LSTM y descenso por el gradiente para $a^n b^n c^n$ con conjuntos de entrenamiento con $n \in [20, 21]$ y con distintas tasas de aprendizaje α y valores del momento γ.

δ	**Secuencias** $[\times 10^3]$	**% aprendido**	**Generalización media**	**Mejor generalización**
10^{-3}	2	20	$[1, 43]$	$[1, 68]$
10^{-2}	2	80	$[1, 48]$	$[1, 77]$
10^{-1}	2	100	$[1, 280]$	$[1, 1162]$
1	2	60	$[1, 291]$	$[1, 1385]$
10	2	100	$[1, 434]$	$[1, 2743]$
10^2	2	70	$[1, 1082]$	$[1, 10000^+]$
10^3	2	80	$[1, 865]$	$[1, 3796]$

Cuadro 7.6: Resultados de LSTM con el FKED con $n \in [1, 10]$ para distintos valores iniciales de la matriz de covarianza del error $P[0] = \delta I$.

Resultados de LSTM con el FKED. Como puede comprobarse en los cuadros 7.6 y 7.7, los resultados del FKED mejoran notablemente los del algoritmo de descenso por el gradiente. Conjuntos de entrenamiento muy pequeños con valores de $n \in [1, 10]$ llegan a ser suficientes para una generalización perfecta hasta valores de $n \in [1, 2000]$ y superiores: en concreto, en uno de los experimentos con $P[0] = 10^2 I$ se obtuvo un conjunto de generalización con $n \in [1, 10000]$, el máximo evaluado.

Más aún, en el caso de $n \in [1, 10]$, el entrenamiento finaliza normalmente tras solo 2000 secuencias de entrenamiento (un par de épocas), mientras que el algoritmo original necesita un número mucho mayor de secuencias; algo similar ocurre en el caso de $n \in [20, 21]$.

Un problema menor del FKED es su inestabilidad ocasional: el aprendizaje ocurre al comienzo de la fase de entrenamiento o nunca. Todos los fallos en los cuadros 7.6 y 7.7 son debidos a esto. En general, sin embargo, cuando $n \in [1, 10]$, el aprendizaje se produce extremadamente rápido y dejan de

δ	**Secuencias** $[\times 10^3]$	**% aprendido**	**Generalización media**	**Mejor generalización**
10^{-3}	–	0	–	–
10^{-2}	4	80	$[17, 23]$	$[11, 24]$
10^{-1}	5	70	$[14, 25]$	$[9, 27]$
1	3	40	$[14, 26]$	$[12, 28]$
10	5	50	$[12, 29]$	$[8, 34]$
10^2	4	40	$[12, 28]$	$[9, 32]$
10^3	5	50	$[13, 27]$	$[8, 32]$

Cuadro 7.7: Resultados de LSTM con el FKED con $n \in [20, 21]$ para distintos valores iniciales de la matriz de covarianza del error $P[0] = \delta I$.

producirse fallos sobre el entrenamiento tras un número de secuencias entre 200 y 400.

7.4. Discusión

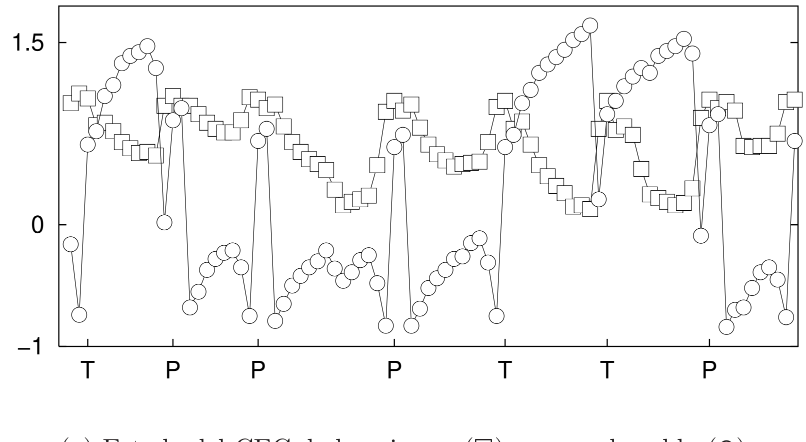
Aprendizaje del lenguaje de Reber

Como ya hemos comentado, en el caso del aprendizaje simbólico en línea, el FKED reduce significativamente el número de iteraciones necesarias para obtener predicciones sin error. Sin embargo, tiende a olvidar más fácilmente.

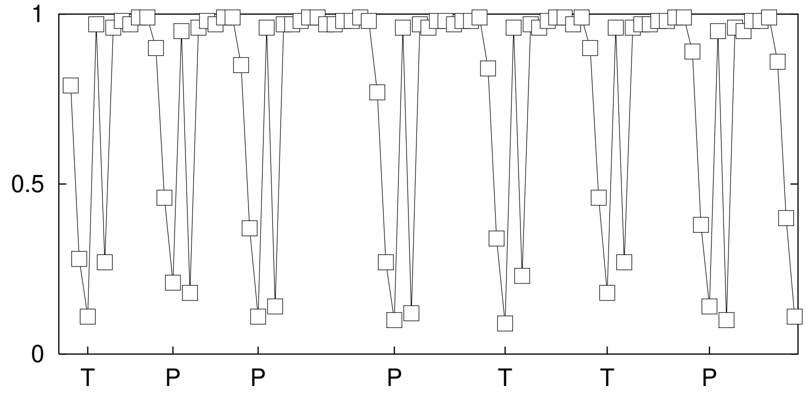
Análisis de la solución. El estudio de la evolución de las activaciones de las celdas y las compuertas de la red LSTM durante el aprendizaje en línea de la secuencia de Reber, revela que el comportamiento de la red es muy similar al observado en anteriores experimentos que no eran completamente en línea (Gers et al. 2000, apartado 4.4), esto es, un bloque de memoria se especializa en la información a largo plazo, mientras que los otros se centran en las variaciones a corto plazo. Independientemente de lo anterior, todos los bloques aprenden a reinicializarse mediante la anulación de la activación de la compuerta de olvido correspondiente. Este comportamiento es común a ambos algoritmos de entrenamiento.

Obsérvense las figuras 7.1 y 7.2. En ellas se muestran las activaciones típicas simultáneas de dos de los bloques de memoria de la red durante 170 símbolos sucesivos tomados de una subsecuencia sin errores de predicción. La información de largo plazo aparece etiquetada con los símbolos (P o T) que deben almacenarse hasta que la secuencia alcance el *lado derecho* de la máquina de estados finitos.

La figura 7.1 muestra las activaciones de los CEC y de la compuerta de olvido del tercer bloque de memoria, encargado de conservar la información



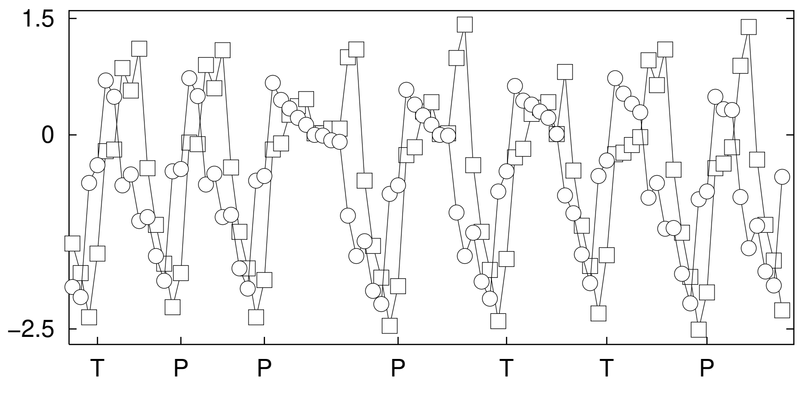
(a) Estado del CEC de la primera (□) y segunda celda (○).



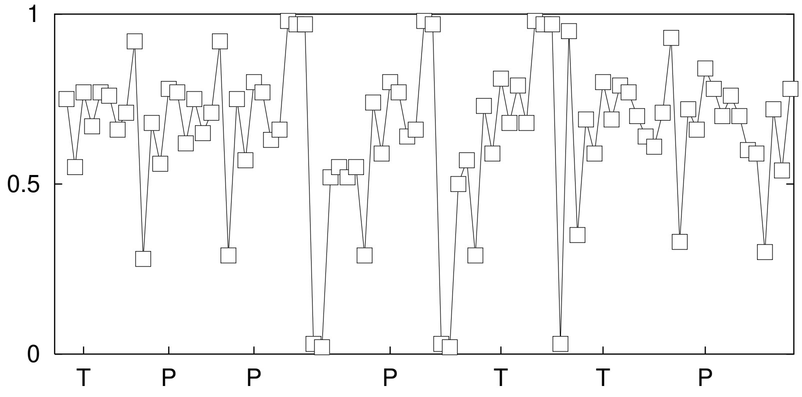
(b) Compuerta de olvido (□).

Figura 7.1: Activaciones del tercer bloque de memoria de una red LSTM en el tramo en que realiza una predicción sostenible del lenguaje de Reber. Este bloque memoriza la información de largo plazo de la secuencia.

de largo plazo de la secuencia. El estado de los CEC se atenúa por acción de la compuerta de olvido sincronizada con la aparición de los símbolos P o T a memorizar. En concreto, el CEC de la segunda celda permanece positivo mientras se ha de recordar una T y negativo cuando se ha de recordar una P. La compuerta de olvido del tercer bloque de memoria está abierta la mayor parte del tiempo, permitiendo que los CEC retengan la información correspondiente; cuando esta información ya ha sido utilizada, la compuerta de olvido se cierra casi completamente y anula el estado de las celdas.



(a) Estado del CEC de la primera (□) y segunda celda (○).



(b) Compuerta de olvido (□).

Figura 7.2: Activaciones del primer bloque de memoria de una red LSTM en el tramo en que realiza una predicción sostenible del lenguaje de Reber. Este bloque procesa la información de corto plazo de la secuencia.

La figura 7.2, por otra parte, muestra las activaciones del primer bloque de memoria, encargado (al igual que los otros dos bloques, no mostrados) de capturar el comportamiento de corto plazo de la secuencia, necesario para predecir el resto de símbolos. En este caso, la compuerta de olvido se abre o se cierra en función de otros eventos, no asociados a la aparición de los símbolos P o T correspondientes a las dependencias a largo plazo.

Aprendizaje de $a^n b^n c^n$

Cuando la red LSTM se combina con el FKED se mejoran notablemente los resultados del algoritmo de descenso por el gradiente utilizado originalmente y se obtiene una convergencia más rápida y un rendimiento mejorado.

Análisis de la solución. En general, la red LSTM resuelve el problema de la inferencia de $a^n b^n c^n$ usando una combinación de dos contadores, instanciados por separado en los dos bloques de memoria.

La figura 7.3 muestra las activaciones que se producen en una red LSTM entrenada con el FKED. Las activaciones son las obtenidas al ir aplicando los sucesivos símbolos de la secuencia $a^6 b^6 c^6$ a las entradas de una red que ya ha aprendido las secuencias de $a^n b^n c^n$ con $n \leq 10$.

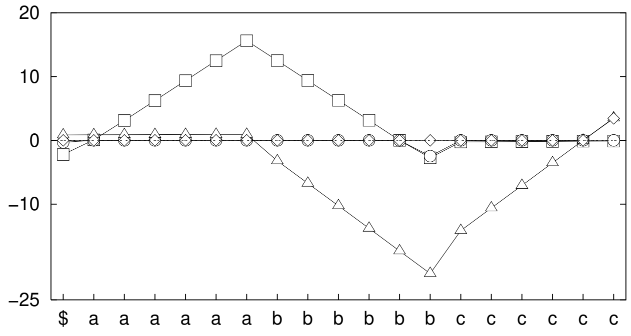
Como puede verse en la gráfica superior, el CEC del primer bloque de memoria se va incrementando con cada aparición del símbolo a y comienza a decrementarse a partir de la primera b. La aparición de una c en la entrada desencadena la clausura de la compuerta de entrada y de la de olvido (véase la gráfica intermedia), lo que provoca la desactivación del contador implementado por este primer bloque.

De forma similar, el CEC del segundo bloque está desactivado hasta que aparece la primera b, instante en que la compuerta de entrada se abre y la compuerta de olvido se cierra momentáneamente para reiniciar convenientemente el estado de la celda (véase la grafica inferior de la figura 7.3). A continuación, el contador implementado por el CEC del segundo bloque comienza a decrementarse hasta que la aparición de la primera c invierte la tendencia del contador y desencadena un incremento progresivo.

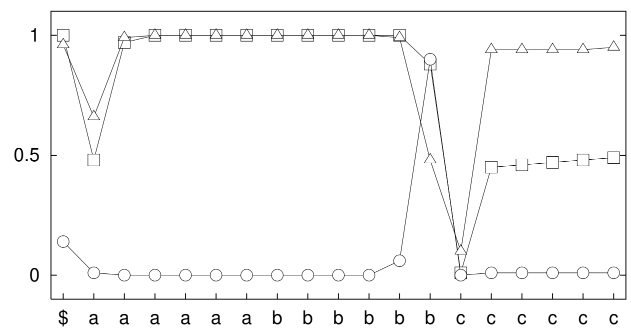
En resumen, uno de los bloques de memoria resuelve $a^n b^n$, mientras que el otro resuelve $b^n c^n$. Todo el sistema funciona de manera extremadamente precisa y robusta en ambos algoritmos de entrenamiento.

Complejidad. Aunque el número de iteraciones que necesita el FKED es mucho menor que las usadas por el descenso por el gradiente, lo cierto es que aquel tiene un coste computacional, especialmente el temporal, mucho mayor. Si calculamos el tiempo necesario para realizar el entrenamiento sobre 1000 secuencias con el FKED, este resulta ser aproximadamente unas 15 veces superior al invertido por el descenso por el gradiente.

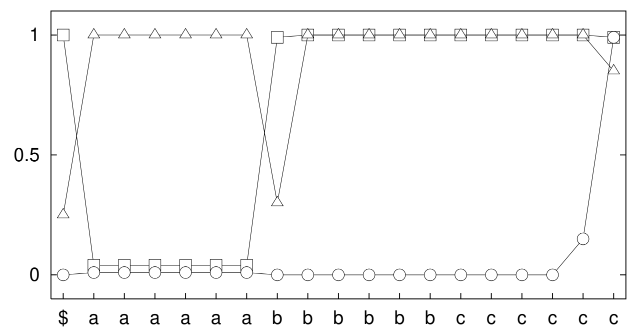
La comparación del descenso por el gradiente y el FKED al usar esta medida relativa permite constatar que la complejidad adicional de la red LSTM entrenada con el FKED se ve ampliamente compensada con el menor número de secuencias necesarias para aprender el conjunto de entrenamiento.



(a) Activación del CEC y de la salida del primer bloque de memoria (□ y ○, respectivamente) y del segundo (△y ◊, respectivamente).



(b) Compuerta de entrada (□), salida (○) y olvido (△) del primer bloque de memoria, encargado de $a^n b^n$.



(c) Compuerta de entrada (□), salida (○) y olvido (△) del segundo bloque de memoria, encargado de $b^n c^n$.

Figura 7.3: Activaciones de los bloques de memoria de una red LSTM que ha aprendido el lenguaje $a^n b^n c^n$.

Por ejemplo, en el caso de $n \in [20, 21]$, el FKED con $P[0] = 10^{-2}I$ obtiene un 80% de soluciones correctas con 4 000 secuencias de entrenamiento, lo que equivale en tiempo a $4\,000 \times 15 = 60\,000$ secuencias con descenso por el gradiente; pero, como puede verse en el cuadro 7.5, el descenso por el gradiente requiere una media de 127 000 secuencias y solo obtiene un 20% de soluciones correctas en el caso mejor.

LSTM es la primera RNR que generaliza el lenguaje $a^n b^n c^n$ con valores de n muy superiores a los utilizados durante el entrenamiento. Al usar el FKED se obtiene un sistema que necesita un número de secuencias varios órdenes de magnitud por debajo del algoritmo de entrenamiento estándar y que generaliza aún mejor. El modelo LSTM combinado con el FKED necesita ver únicamente cadenas de longitud menor que $a^{11}b^{11}c^{11}$ para extraer las reglas generales del lenguaje sensible al contexto $a^n b^n c^n$ y generalizar correctamente con secuencias como $a^{1000}b^{1000}c^{1000}$ y aún mayores.

8. DESAMBIGUACIÓN CATEGORIAL

> Este capítulo muestra cómo utilizar una RRS para desambiguar las categorías léxicas de las palabras homógrafas de un texto. Se trata de uno de los primeros ejemplos de uso de RNR en esta tarea. Además, el enfoque seguido no necesita textos completamente etiquetados y es original en el hecho de que la desambiguación se enfoca como un problema de predicción. Los contenidos de este capítulo han sido publicados en las actas de un congreso internacional (Pérez-Ortiz y Forcada 2001).

Para la introducción referente a los conceptos de este capítulo puede consultarse los apartados 1.2.3 y 5.3. Allí se vió que un etiquetador morfológico es un programa que asigna a cada palabra de un texto una categoría léxica; esta tarea es especialmente importante cuando uno se encuentra con palabras ambiguas, a las que puede corresponder, en principio, más de una etiqueta dependiendo del contexto. La mayoría de los etiquetadores se basan en la suposición de que las palabras vecinas son útiles a la hora de desambiguar estas palabras. Aquí haremos uso de esa información contextual con una RNR.

8.1. Método

En estos experimentos se explora el uso de una RRS para el etiquetado de las categorías léxicas de las palabras de una oración. En primer lugar, el texto de entrenamiento se etiqueta parcialmente mediante un diccionario o léxico (un analizador morfológico) que asigna a cada palabra una *clase de ambigüedad*, esto es, un conjunto de posibles categorías léxicas (este conjunto tiene un único elemento en el caso de las palabras no ambiguas). En este trabajo se usan clases de ambigüedad y no palabras: los primeros estudios de Elman (1990) se realizaron sobre pequeños vocabularios, pero los vocabularios reales tienen miles de entradas, mientras que el número de clases de ambigüedad es normalmente de unos cientos; esto reduce drásticamente el tamaño de la tarea de predicción.

Todos los trabajos neuronales anteriores (véase el apartado 5.3) se basan en entrenar la red a partir de corpus completamente etiquetados, además de usar para las palabras representaciones basadas también en textos completamente etiquetados. Este trabajo es también un ejemplo de acercamiento al tema desde un punto de vista neuronal, pero la gran diferencia es que la RNR se entrena a partir de un corpus parcialmente etiquetado.

Como se verá con más detalle a continuación, se entrena una RRS para que aprenda a predecir la clase de ambigüedad de la siguiente palabra del texto; a continuación, se usa un perceptrón sin capa oculta para extraer la categoría léxica a partir de la información de estado desarrollada por la RNR para el texto considerado durante la primera fase del entrenamiento.

Los experimentos mostrarán las tasas de error al etiquetar algunos textos del corpus del Penn Treebank (Marcus et al. 1993). También se indican los resultados con un modelo oculto de Markov (MOM, véase el apéndice A) con los mismos datos.

8.1.1. Fases de entrenamiento

En nuestro enfoque, la RRS se entrena en dos fases: en primer lugar, el texto de entrenamiento se etiqueta parcialmente mediante un léxico o analizador morfológico, de manera que se asigna a cada palabra un conjunto de categorías léxicas. Así, el texto de entrenamiento pasa a ser un conjunto de secuencias (una por cada oración) de clases de ambigüedad de la forma $\boldsymbol{a}[1], \ldots, \boldsymbol{a}[t], \ldots$

Tras esto, comienza un proceso de entrenamiento fuera de línea por secuencias en dos fases:

Primera fase. La RRS es entrenada para predecir la clase de ambigüedad de la siguiente palabra $\boldsymbol{y}[t] \simeq \boldsymbol{a}[t+1]$ a partir de la clase de ambigüedad de la palabra actual $\boldsymbol{a}[t]$, e, indirectamente, a partir de las clases de las palabras anteriores de la frase, $\boldsymbol{a}[1], \cdots, \boldsymbol{a}[t-1]$, que idealmente estarán representadas en el estado de la red, $\boldsymbol{x}[t]$. Es de esperar que de esta manera la RRS aprenderá a desarrollar en su estado $\boldsymbol{x}[t]$ una representación sintáctica del prefijo visto de la oración, lo que permitirá realizar una predicción acertada de la clase de ambigüedad de la siguiente palabra.

Como ya se ha hecho en otros lugares de esta tesis, se utiliza la codificación exclusiva tanto para las entradas como para las salidas deseadas; así, al utilizar una función de error cuadrática, las salidas pueden interpretarse como probabilidades (véase el apartado 4.3.1). Como algoritmo de entrenamiento se utiliza descenso por el gradiente con

RTRL.[1] Además, también se aprende durante el entrenamiento el estado inicial (Bulsari y Saxén 1995; Forcada y Carrasco 1995) de forma que la RRS se reinicia al estado aprendido al principio de cada frase para evitar las interferencias entre frases, ya que, a fin de cuentas, solo el contexto oracional influye en la categoría léxica.

Segunda fase. Después del entrenamiento, cada palabra del texto se etiqueta con el vector de estado $\boldsymbol{x}[t]$ calculado para ella por la red; entonces, para cada palabra del texto se entrena un perceptrón para que aprenda su categoría léxica a partir del vector de estado asignado a la palabra que está f posiciones a su derecha, $\boldsymbol{x}[t+f]$. El valor de f representa la cantidad de contexto posterior (a la derecha) que se necesita para desambiguar una palabra; la cantidad de contexto anterior (a la izquierda) necesaria es determinada por el propio algoritmo de entrenamiento. Nótese cómo para que esto funcione, hay que añadir f marcadores artificiales de final de frase (una palabra no ambigua) y aprender a predecirlos durante la primera fase como el resto de palabras.[2]

En esta segunda fase se asigna una neurona de salida a cada categoría léxica. A la hora de representar la salida deseada $\boldsymbol{d}[t]$ para una palabra ambigua, se han utilizado tres esquemas distintos:

1. Los componentes correspondientes a categorías de la clase de ambigüedad se ponen a 1 y el resto de componentes a 0.
2. Como el punto anterior, pero usando $1/\theta$ en lugar de 1, donde $\theta \geq 1$ es el tamaño de la clase de ambigüedad.
3. Solo se impone una salida deseada con valor 0 a las categorías no incluidas en la clase de ambigüedad; el resto no contribuyen a la función de error.

Aunque el tercer enfoque es el único que garantiza la convergencia a las probabilidades de las etiquetas (con una función de error cuadrática, por ejemplo) en el caso de las palabras ambiguas, los experimentos demuestran que el primer enfoque es el que proporciona los mejores resultados.

La combinación de la parte del estado de la RRS y el perceptrón se utiliza para determinar la categoría léxica de las palabras de oraciones nuevas. Las figuras 8.1 y 8.2 ilustran el esquema de entrenamiento de la primera y segunda fases, respectivamente.

[1] El algoritmo de entrenamiento no tiene un papel vital en un trabajo como este, que pretende ser una demostración de una idea; por eso usamos RTRL y no formas más idóneas, pero más costosas, como BPTT.

[2] También habría sido posible utilizar el mismo marcador artificial f veces seguidas al final de cada frase.

$$
\begin{array}{ccccccccccccccc}
\tilde{\boldsymbol{a}}[1] & & \tilde{\boldsymbol{a}}[2] & & \tilde{\boldsymbol{a}}[3] & & \dots & \tilde{\boldsymbol{a}}[L] & & \tilde{\#}_1 & & \tilde{\#}_2 & & \dots & \tilde{\#}_{f+1} \\
\uparrow & & \uparrow & & \uparrow & & & \uparrow & & \uparrow & & \uparrow & & & \uparrow \\
\boldsymbol{x}[0] & \to & \boldsymbol{x}[1] & \to & \boldsymbol{x}[2] & \to & \dots & \boldsymbol{x}[L-1] & \to & \boldsymbol{x}[L] & \to & \boldsymbol{x}[L+1] & \to & \dots & \boldsymbol{x}[L+f] \\
 & & \uparrow & & \uparrow & & & \uparrow & & \uparrow & & \uparrow & & & \uparrow \\
 & & \boldsymbol{a}[1] & & \boldsymbol{a}[2] & & \dots & \boldsymbol{a}[L-1] & & \boldsymbol{a}[L] & & \#_1 & & \dots & \#_f
\end{array}
$$

Figura 8.1: Primera fase del entrenamiento: predicción de la siguiente clase de ambigüedad. el símbolo "˜" significa *valor predicho*; los "#" son los marcadores de final de frase. Nótese la secuencia de f marcadores de final de frase, necesaria para poder etiquetar las últimas f palabras de cada frase en la segunda fase del entrenamiento.

$$
\begin{array}{cccccccccccccc}
 & & & & & \boldsymbol{d}[1] & & \ldots & \boldsymbol{d}[L-f-1] & & \boldsymbol{d}[L-f] & & \ldots & \boldsymbol{d}[L] \\
 & & & & & \uparrow & & & \uparrow & & \uparrow & & & \uparrow \\
\boldsymbol{x}[0] & \rightarrow & \boldsymbol{x}[1] & \rightarrow & \ldots & \boldsymbol{x}[f+1] & \rightarrow & \ldots & \boldsymbol{x}[L-1] & \rightarrow & \boldsymbol{x}[L] & \rightarrow & \ldots & \boldsymbol{x}[L+f-1] \\
 & & \uparrow & & & \uparrow & & & \uparrow & & \uparrow & & & \uparrow \\
 & & \boldsymbol{a}[1] & & \ldots & \boldsymbol{a}[f+1] & & \ldots & \boldsymbol{a}[L-1] & & \boldsymbol{a}[L] & & \ldots & \#_f
\end{array}
$$

Figura 8.2: Segunda fase del entrenamiento: predicción de la etiqueta correcta a partir de la información de estado. La salida deseada $\boldsymbol{d}[t]$ sigue uno de los tres esquemas de la página 115.

8.1.2. Modelos alternativos

Para evaluar los resultados de nuestro enfoque, los compararemos con los obtenidos con otros tres modelos:

- *Modelo de Markov.* Un modelo oculto de Markov (MOM) estándar entrenado mediante el algoritmo de Baum y Welch, como se muestra en el apéndice A.

- *Aleatorio.* Un modelo que escoge aleatoriamente una etiqueta léxica en cada clase de ambigüedad con probabilidad $1/\theta$, donde θ es el número de etiquetas de la clase.

- *Etiqueta predominante.* Un modelo de etiqueta predominante globalmente en el que se selecciona siempre la categoría léxica más probable de cada clase de ambigüedad según el muestreo realizado en un corpus completamente etiquetado; el Penn Treebank, en este caso.

Los resultados del etiquetado sobre los textos de evaluación se comparan con una versión etiquetada "a mano" de los mismos textos. El primero de los modelos anteriores es la solución estándar al problema, mientras que los otros dos se usan como punto base de referencia. El último modelo es, de hecho, injusto porque utiliza información que no está disponible en el conjunto de entrenamiento.

8.2. Parámetros

Los experimentos determinaron las tasas de error obtenidas al etiquetar textos del corpus Penn Treebank versión 3 (Marcus et al. 1993). Se construyó un léxico de 14 276 entradas a partir de las primeras 20 secciones de los 24 conjuntos de datos correspondientes al *Wall Street Journal*; se eliminaron todas las palabras que aparecían menos de 4 veces (con lo que se cubre el 95%) y las etiquetas que se asociaban a una palabra menos de un 5% de las veces. No se utilizó ningún adivinador.

El corpus de entrenamiento tiene 46 461 palabras; el corpus independiente de evaluación tiene 47 397 palabras, de las cuales 6 574 son ambiguas según el léxico y 2 290 son desconocidas. Las 45 etiquetas originales del Penn Treebank se redujeron a 19 etiquetas más *gruesas* (a las que debe sumarse los marcadores artificiales de fin de frase) eliminando algunas distinciones léxicas; con ello el número de clases de ambigüedad observadas era de 82. El nuevo conjunto de etiquetas se muestra en el cuadro 8.1 y una breve descripción de su significado en el cuadro 8.2.

Nueva etiqueta	**Etiquetas que agrupa**
XSYM	\# \$ '' ( ) , LS SYM ``
.	.
:	:
XCC	CC
XCD	CD
XDT	DT PDT PRP$
XEX	EX
XFW	FW
XIN	IN
XJJ	JJ JJR JJS
XVB	MD VB VBD VBG VBN VBP VBZ
XNN	NN NNS NNP NNPS
XPOS	POS
XPRP	PRP
XRB	RB RBR RBS
XRP	RP
XTO	TO
XUH	UH
XWH	WDT WP WP$ WRB

Cuadro 8.1: Reducción del etiquetario del Penn Treebank (Marcus et al. 1993). Por ejemplo, la categoría XNN agrupa lo que originalmente estaba separado como nombre propio o común y singular o plural.

Para entrenar la RRS, se usa entrenamiento fuera de línea por secuencias y descenso por el gradiente con RTRL. La tasa de aprendizaje es $\alpha = 0.05$ (sin momento). La influencia del número de neuronas de estado n_X se muestra en los experimentos. Los pesos iniciales se tomaron aleatoriamente en el rango $[-0.2, 0.2]$. Todos los resultados neuronales son la media de tres experimentos distintos.

Los resultados se expresan como el porcentaje de etiquetas incorrectas asignadas a palabras ambiguas (incluyendo las desconocidas), no como el porcentaje general de etiquetas correctas, una medida habitual, pero confusa, ya que muchas palabras no son ambiguas y, además, el hecho de que en algunos idiomas la ambigüedad sea mayor que en otros impide la comparación de etiquetadores cuyos experimentos se muestren sobre distintos idiomas.

Etiqueta	**Descripción**
XSYM	Símbolo
.	Punto
:	Dos puntos
XCC	Conjunción coordinativa
XCD	Número cardinal
XDT	Determinante o pronombre posesivo
XEX	*There* existencial
XFW	Palabra que no está en inglés
XIN	Preposición o conjunción subordinativa
XJJ	Adjetivo
XVB	Verbo
XNN	Nombre
XPOS	Terminación del posesivo
XPRP	Pronombre personal
XRB	Adverbio
XRP	Partícula
XTO	*To*
XUH	Interjección
XWH	Determinante, pronombre o adverbio con *wh-*

Cuadro 8.2: Descripción de las etiquetas categoriales empleadas en los experimentos.

8.3. Resultados

Una serie de experimentos permitió constatar que los mejores resultados se obtienen con $f = 0$, es decir, usando directamente en la segunda fase del entrenamiento el estado de la RRS obtenido para una palabra como referencia para determinar su categoría léxica. Como ejemplo, en el cuadro 8.3 se muestran los resultados con $n_X = 12$ y $f = 0, 1$. El valor $n_X = 12$ es un buen compromiso entre el tamaño de la red y las tasas de error obtenidas como puede observarse en el cuadro 8.4, donde pueden verse los resultados según el número de iteraciones en cada fase con distintos números de neuronas de estado y con $f = 0$. Además, este valor hace que el número de parámetros de la RRS a ajustar sea muy similar a los del MOM empleado.

Modelo oculto de Markov. Los resultados del MOM se muestran en el cuadro 8.5 en función del número de iteraciones del algoritmo de Baum y Welch.

Aleatorio. El etiquetado aleatorio proporciona una tasa de etiquetas incorrectas entre el 61.8% y el 62.9% (5 479 y 5 572 errores, respectivamente),

		Iteraciones fase 1 →					
		0	100	200	300	400	500
	0	98.7, 95.4	61.8, 67.8	62.1, 66.7	62.0, 66.6	62.1, 66.4	62.3, 66.4
Iteraciones	100	96.7, 92.0	59.9, 46.4	60.1, 47.3	60.2, 47.4	60.2, 47.4	60.2, 47.5
fase 2	200	96.8, 90.4	55.2, 45.6	55.3, 45.4	55.2, 45.4	55.1, 45.2	54.9, 45.0
↓	300	96.1, 90.6	55.8, 44.9	56.0, 44.5	55.7, 44.4	55.7, 44.2	55.4, 44.2
	400	96.5, 90.6	55.8, 44.4	55.8, 44.9	55.8, 45.1	55.7, 45.0	55.8, 45.0
	500	96.9, 89.6	55.5, 45.1	55.5, 45.5	55.5, 45.5	55.5, 45.6	55.7, 45.5

Cuadro 8.3: Tasas de error con 12 neuronas de estado y contexto derecho $f = 1,0$ según el número de iteraciones de cada fase. Horizontal: iteraciones de la fase 1; vertical: iteraciones de la fase 2.

		Iteraciones fase 1 →					
		0	100	200	300	400	500
	0	88.9, 95.4, 100.0	68.7, 67.8, 59.7	73.4, 66.7, 58.1	71.8, 66.6, 57.9	70.6, 66.4, 58.2	69.6, 66.4, 58.8
Iteraciones	100	87.2, 92.0, 100.0	60.2, 46.4, 49.4	60.0, 47.3, 49.6	59.6, 47.4, 49.6	59.4, 47.4, 49.6	59.5, 47.5, 49.6
fase 2	200	86.0, 90.4, 99.7	61.6, 45.6, 47.5	64.4, 45.4, 47.5	65.0, 45.4, 47.6	65.1, 45.2, 47.5	65.6, 45.0, 47.5
↓	300	86.2, 90.6, 99.2	60.6, 44.9, 46.1	62.6, 44.5, 46.1	64.6, 44.4, 45.7	65.2, 44.2, 45.2	65.1, 44.2, 45.1
	400	86.3, 90.6, 99.2	60.4, 44.4, 44.5	61.5, 44.9, 44.0	63.1, 45.1, 43.8	64.0, 45.0, 43.6	64.8, 45.0, 43.5
	500	86.4, 89.6, 98.8	60.2, 45.1, 45.6	61.7, 45.5, 45.1	63.0, 45.5, 44.7	63.7, 45.6, 44.4	64.1, 45.5, 44.2

Cuadro 8.4: Tasas de error con $n_X = 4, 12, 24$ y contexto derecho $f = 0$ según el número de iteraciones de cada fase. Horizontal: iteraciones de la fase 1; vertical: iteraciones de la fase 2.

Iteraciones	0	1	2	3	4	5	6	7	8
Error	62.8	51.7	48.8	47.2	46.0	45.5	45.3	45.3	45.5

Cuadro 8.5: Tasas de error con un MOM en función del número de iteraciones del algoritmo Baum y Welch.

según se observó en 14 experimentos con distintas semillas para el generador de números aleatorios.

Etiqueta predominante. Esta estrategia da tasas de error de 20.5%, un resultado razonablemente bueno, aunque debe tenerse en cuenta que se basa en la disponibilidad de un corpus etiquetado "a mano" y que este método no puede hacer nada con las palabras desconocidas (no existentes en este caso), las cuales suelen representar un porcentaje considerable de las palabras de los textos de evaluación.

Etiquetador neuronal. Consideraremos los resultados de un modelo neuronal con aproximadamente el mismo numero de parámetros que los MOM correspondientes; por ello nos centraremos en los resultados con 12 neuronas de estado, aunque los resultados con 24 eran ligeramente mejores. Los resultados para $f = 0$ y $f = 1$ se muestran el cuadro 8.3 en función del número de iteraciones en cada fase. Como puede verse, añadir una palabra de contexto posterior confunde a la RRS, que hace mejores predicciones basándose únicamente en el contexto pasado. También se observa cómo un entrenamiento intenso en la fase 2 es más importante que en la fase 1.

La tasa de etiquetado correcto *global* sobre palabras ambiguas y no ambiguas de la RRS y del MOM está en torno al 92%: los mejores resultados son un 91.5% (4 011 errores) para el MOM y un 91.9% (3 852 errores) para la RNR.

8.4. Discusión

Un etiquetador neuronal sencillo con aproximadamente el mismo número de parámetros ajustables obtiene basicamente los mismos resultados (en torno a un 45% de etiquetas incorrectas) que un MOM estándar entrenado con el algoritmo de Baum y Welch. El etiquetador neuronal, sin embargo, toma las decisiones relativas a las palabras ambiguas sin tener en cuenta las palabras posteriores, mientras que el MOM tiene que posponer esta decisión hasta encontrar una palabra no ambigua (véase el apéndice A). Esta capacidad no parece dar ventaja alguna al MOM sobre la RRS.

Como se ha podido constatar, la información almacenada en el estado de una RNR entrenada para predecir la categoría léxica de la siguiente palabra puede ser útil en el problema del etiquetado categorial. Los resultados indican que el rendimiento de ambos enfoques, el basado en MOM y el neuronal, son comparables, aunque este último ignora el contexto posterior.

Una diferencia significativa entre nuestro método y los MOM es que estos desambiguan *globalmente* (el algoritmo de Viterbi realiza una optimización global sobre toda la frase o, como mínimo, sobre segmentos delimitados por las palabras no ambiguas), mientras que el nuestro lo hace *localmente*. En cualquier caso, el coste temporal de entrenamiento es significativamente mayor para el caso neuronal (horas) que para el MOM (minutos).

9. PREDICCIÓN DE SEÑALES DE VOZ

Los únicos trabajos previos que usan RNR para la prediccion de señales de voz se basan en la RNRC, un modelo formado por una sucesión de redes sencillas en cascada. En este capítulo estudiaremos cómo se comportan en esta tarea las RNR clásicas. Los siguientes resultados han sido presentados en un congreso internacional (Pérez-Ortiz et al. 2001a).

La introducción a los contenidos de este capítulo puede encontrarse en los apartados 1.2.4 y 5.4.

9.1. Método

Siguiendo la línea de los trabajos de Haykin y Li (1995) y Baltersee y Chambers (1998), el modelo de predictor que usaremos está compuesto por un predictor no lineal (una RNR) que debería proporcionar una salida localmente más lineal que la señal de entrada, seguido de un predictor lineal (un filtro) que sacaría partido de esta linealización. Cada módulo se entrena de forma separada. Es de esperar que "esta combinación de un filtro no lineal con un filtro lineal debería poder extraer tanto la información no lineal como la lineal contenida en la señal de entrada con vistas a producir la predicción" (Haykin y Li 1995).

La figura 9.1 muestra un diagrama para el modelo en cascada completo. El primer módulo se entrena para predecir la muestra $u[t]$ a partir de las p muestras anteriores[1] y de la información almacenada en el estado de la red. La señal predicha $\hat{u}[t]$ se introduce, entonces, en el módulo lineal, que es entrenado para predecir la muestra del instante $t+1$. Esta última se considera la salida global del sistema. Como se muestra en el diagrama (siguiendo los artículos citados antes), el módulo no lineal tiene orden de entrada p, y p_L es el orden correspondiente del predictor lineal. Además

[1] La introducción explícita en la red de las muestras recientes le da una ventaja adicional sobre las redes de una única entrada que solo pueden acceder a esta historia a través de su estado. Los experimentos determinarán la importancia de este añadido.

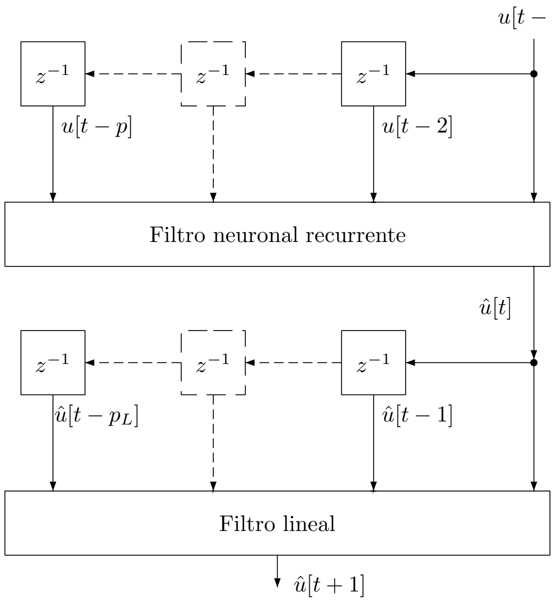


Figura 9.1: Modelo de predictor que combina filtros lineales y no lineales (Haykin y Li 1995).

de los resultados de este modelo híbrido, se mostrarán también en algunos casos los resultados de ambos tipos de predictores por separado.

A continuación se describen los modelos usados en las partes lineal y no lineal del predictor.

9.1.1. Predictores no lineales

Compararemos el rendimiento de las siguientes RNR (véase el capítulo 3) cuando actúan como predictores no lineales: la RNRC, la RRS y la RPR. También se experimentó con otros modelos, como la RTR o la red NARX, para obtener resultados similares a los de la RRS o la RPR, que no se mostrarán aquí. Como algoritmos de entrenamiento en línea se consideran el descenso por el gradiente (que representaremos por DG en las figuras y cuadros de este capítulo) y el FKED, ambos con derivadas calculadas según RTRL.

9.1.2. Predictores lineales

El predictor lineal es un filtro FIR (véase el capítulo 3) con los pesos adaptados por medio del algoritmo de *mínimos cuadrados* (LMS, por el inglés *least-mean-square*) (Oppenheim y Schafer 1989; Proakis y Manolakis 1996) o por medio del algoritmo de *mínimos cuadrados recursivo* (RLS, por el inglés *recursive least-squares*) (Oppenheim y Schafer 1989; Proakis y Manolakis 1996); el primero guarda relación con el descenso por el gradiente y el segundo con el filtro de Kalman.

Para determinar un rendimiento base, también evaluaremos la calidad de predicción con el predictor lineal sin parámetros más sencillo posible: el que calcula $\hat{u}[t+1] = u[t]$.

9.2. Parámetros

Estudiaremos la calidad de los predictores con las tres señales (todas de 10 000 muestras) utilizadas por Baltersee y Chambers (1998).[2] El rendimiento se mide con la *ganancia de predicción* (GP), expresada en decibelios, que se define como:

$$G = 10 \log_{10} \left(\frac{S_u^2}{S_e^2} \right) \tag{9.1}$$

donde S_u^2 es la varianza estimada de la señal de voz $u[t]$ y S_e^2 es la varianza estimada de la señal de error $e[t] = u[t] - \hat{u}[t]$. La GP puede verse como una relación señal-ruido para el ruido introducido por la predicción errónea.

Las amplitudes de las tres señales pertencen al rango $[0, 1]$, por lo que utilizaremos la función sigmoidea logística g_L para las activaciones de las neuronas de salida de las RNR y, en general, para todas las funciones de activación.

Basándonos de nuevo en los trabajos previamente citados, llevaremos a cabo con los modelos neuronales un aprendizaje inicial por épocas sobre 200 muestras de la señal de entrada.[3] El número de épocas utilizado es de 200 para el descenso por el gradiente y de 5 para el FKED. Estos valores son los que dieron mejores resultados en distintos experimentos preliminares: valores mayores para el caso del FKED o menores para el descenso por el gradiente reducían la GP en unos cuantos decibelios.

[2] Las señales `s1`, `s2` y `s3` están disponibles en la página de J. Baltersee en la dirección `http://www.ert.rwth-aachen.de/Personen/baltersee.html`.

[3] Esto atenúa parcialmente la naturaleza en línea de la predicción, pero podría ser aceptable si su complejidad es baja. Los experimentos demostraron que esta inicialización tenía gran influencia en las ganancias de predicción conseguidas.

Entrenamiento	Señal 1	Señal 2	Señal 3
LMS	8.99	7.98	5.82
RLS	13.32	11.60	9.66

Cuadro 9.1: Ganancias de predicción en decibelios para un filtro FIR de orden $p_L = 12$.

Entrenamiento	Señal 1	Señal 2	Señal 3
RNRC (DG) + LMS	10.25	9.49	7.30
RNRC (DG) + RLS	13.01	11.80	9.24
RNRC (FKED) + RLS	14.73	13.59	10.90

Cuadro 9.2: Ganancias de predicción en decibelios con una RNRC. Valores tomados del trabajo de Baltersee y Chambers (1998).

A menos que se diga lo contrario, todas las GP mostradas son la media de 7 inicializaciones distintas de los pesos; la varianza de todos los resultados estaba por debajo de 0.3. Los pesos iniciales se tomaron aleatoriamente de una distribución en [-0.2,0.2].

Cuando se usa el descenso por el gradiente con la RRS y la RPR, los parámetros son $\alpha = 0.3$ y $\gamma = 0$. Por otro lado, los parámetros del FKED sobre estos modelos son:

- $Q[t] : 10^{-2} \stackrel{T=1000}{\longrightarrow} 10^{-6}$
- $R[t] : 100 \stackrel{T=1000}{\longrightarrow} 3$
- $P[0] = 1000I$

9.3. Resultados

Las GP obtenidas con un filtro lineal de orden $p_L = 12$ al usar los algoritmos LMS o RLS se muestran en el cuadro 9.1. En este caso, el factor de olvido para RLS es 0.998, los elementos de la diagonal de la matriz de correlación inversa de RLS se inicializan a 100, y LMS usa una constante de adaptación de 0.2. Estos valores son los que proporcionaron mejores resultados en experimentos preliminares.

Los resultados con la RNRC están tomados de los experimentos de Baltersee y Chambers (1998) y se muestran en el cuadro 9.2. En su artículo se pueden encontrar detalles sobre los parámetros de entrenamiento utilizados. Debe destacarse que en el citado artículo no aparece indicación alguna sobre

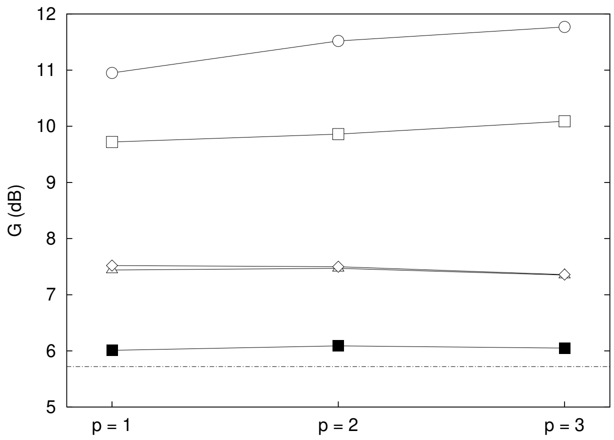


Figura 9.2: Ganancias de predicción para la señal 1 con la RRS y DG (□), FKED (○), DG+LMS (△), FKED+LMS (◇) y FKED+RLS (■). La línea constante representa el rendimiento base, $\hat{u}[t+1] = u[t]$.

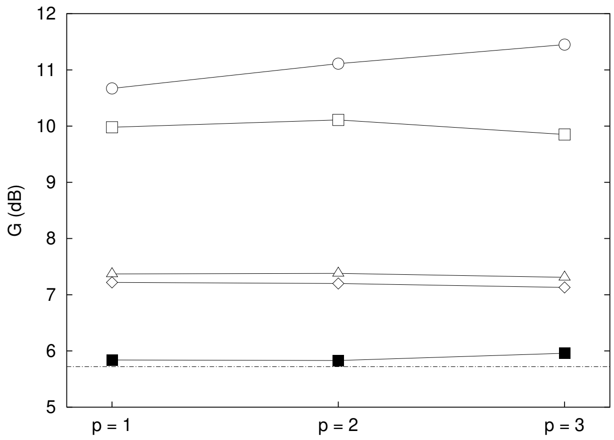


Figura 9.3: Ganancias de predicción para la señal 1 con RPR y DG (□), FKED (○), DG+LMS (△), FKED+LMS (◇) y FKED+RLS (■). La línea constante representa el rendimiento base, $\hat{u}[t+1] = u[t]$.

resultados medios, ya que solo se muestran los resultados de un experimento con parámetros *ad hoc* (elegidos de forma diferente para cada señal). En cualquier caso, incluso los mejores resultados obtenidos aquí con el resto de modelos recurrentes son, como se verá, peores que los obtenidos con la RNRC.

Los cuadros y las gráficas indican el algoritmo de entrenamiento en línea utilizado para las RNR y, de haberlos, el utilizado para los filtros lineales,

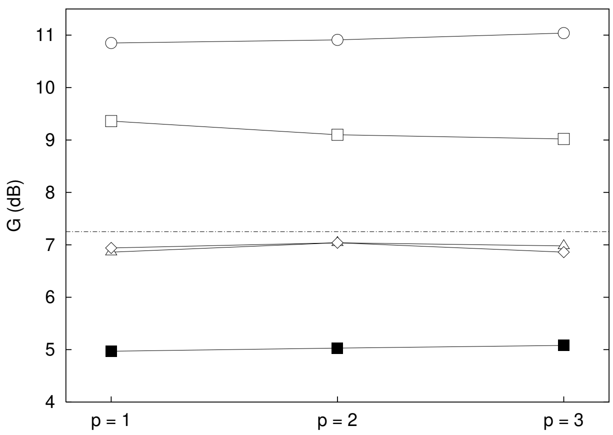


Figura 9.4: Ganancias de predicción para la señal 2 con RRS y DG (□), FKED (○), DG+LMS (△), FKED+LMS (◊) y FKED+RLS (■). La línea constante representa el rendimiento base, $\hat{u}[t+1] = u[t]$.

separados por un signo de suma. El orden del filtro lineal es en todos los casos $p_L = 12$.

Los resultados al usar los predictores basados en RRS y RPR con $n_X = 5$ se muestran en las figuras de la 9.2 a la 9.7 para diferentes valores del orden de entrada p. En estos casos los parámetros son 0.2 para la constante de adaptación de LMS, 1 para el factor de olvido de RLS (es decir, no se considera el factor de olvido: los valores por debajo de 1 hacían que el sistema se volviera inestable) y 1000 para los elementos iniciales de las matrices de correlación de RLS.

El valor $n_X = 5$ y los órdenes de entrada $p = 1, 2, 3$ se eligieron de modo que el número de parámetros a aprender fuera comparable con los usados por Baltersee y Chambers (1998), que consideraron RNRC con unos 35 pesos ajustables.[4] En cualquier caso, las RRS y RPR con un número diferente de neuronas de estado dieron resultados que no variaban significativamente con respecto a los presentados para $n_X = 5$; por ejemplo, con $n_X = 1$ los resultados con descenso por el gradiente son prácticamente los mismos, mientras que los del FKED están 1 dB por debajo; con $n_X = 10$, el descenso por el gradiente vuelve a dar GP similares, mientras que el FKED las mejora muy poco (entre 0 y 0.5 dB, según el modelo de red y la señal concreta).

[4] El número de pesos, incluidos los sesgos, de una RRS de orden p con una única salida es $(p + n_X + 2)n_X + 1$; en el caso de una RPR de orden p con una única salida es $(p + n_X + 2)n_X + p + 1$.

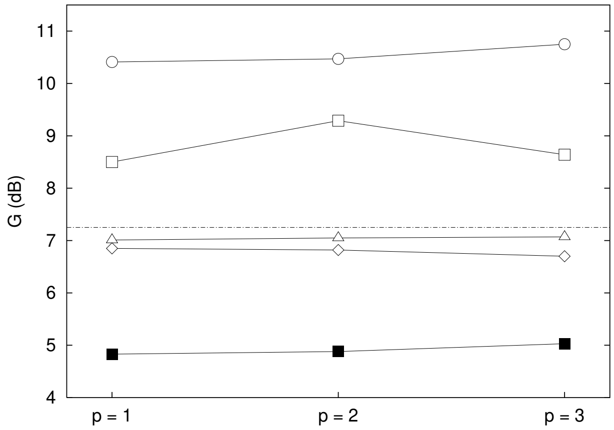


Figura 9.5: Ganancias de predicción para la señal 2 con RPR y DG (□), FKED (○), DG+LMS (△), FKED+LMS (◇) y FKED+RLS (■). La línea constante representa el rendimiento base, $\hat{u}[t+1] = u[t]$.

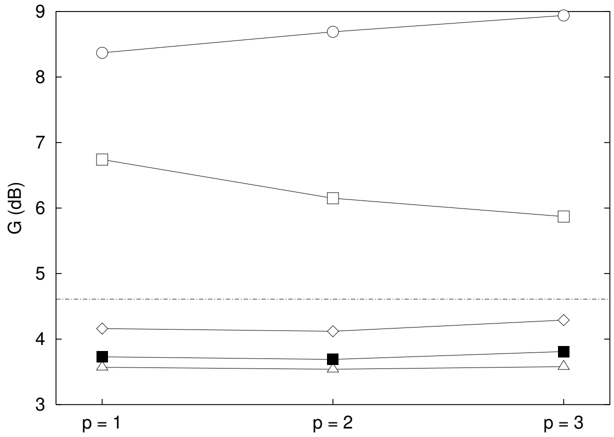


Figura 9.6: Ganancias de predicción para la señal 3 con RRS y DG (□), FKED (○), DG+LMS (△), FKED+LMS (◇) y FKED+RLS (■). La línea constante representa el rendimiento base, $\hat{u}[t+1] = u[t]$.

Finalmente, las GP de un filtro simple de la forma $\hat{u}[t+1] = u[t]$ se muestran como líneas constantes en las figuras 9.2 a 9.7. Este es el modo más sencillo de predecir la siguiente secuencia y se muestra aquí como referencia base.

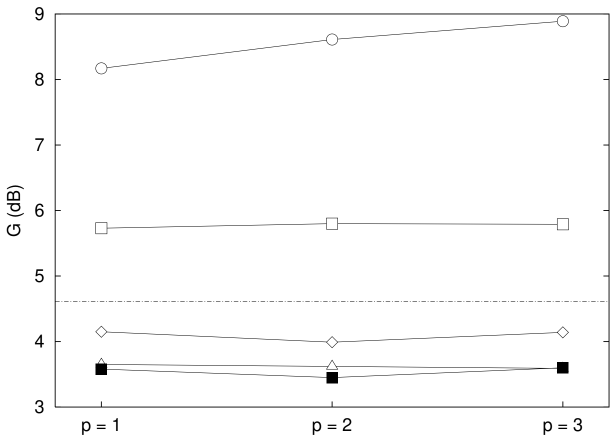


Figura 9.7: Ganancias de predicción para la señal 3 con RPR y DG (□), FKED (○), DG+LMS (△), FKED+LMS (◊) y FKED+RLS (■). La línea constante representa el rendimiento base, $\hat{u}[t+1] = u[t]$.

9.4. Discusión

De entre los tres modelos recurrentes estudiados, solo la RNRC entrenada con el FKED y seguida por un filtro lineal entrenado con RLS supera claramente (entre 1 dB y 2 dB por encima) la GP de un filtro lineal de orden 12 entrenado con RLS. El resto de configuraciones neuronales (en cascada o no) se comportan mucho peor que un sencillo filtro FIR con menos parámetros entrenado con RLS.

Al usar aisladamente la RRS o la RPR, el FKED produce resultados mucho mejores que los del algoritmo de descenso por el gradiente: el FKED permite obtener GP superiores entre 1 dB y 3 dB. Los resultados con ambos modelos y el FKED confirman de forma consistente anteriores estudios (Birgmeier 1996) que situaban en unos 3 dB la mejora de los predictores no lineales sobre los lineales entrenados con LMS. Sin embargo, ninguno de los algoritmos de entrenamiento permite alcanzar las GP de un filtro FIR entrenado con RLS.

Curiosamente, al situar en cascada predictores basados en RRS o RPR y predictores lineales, los resultados son peores que cuando se utilizan los predictores no lineales (RRS y RPR) aisladamente. Los resultados de estas configuraciones en cascada son muy negativos al compararlos con la referencia base (un predictor que se limita a reproducir la muestra actual); de hecho, para las señales 2 y 3 incluso son peores.

En resumen, se está produciendo la siguiente situación: podemos considerar que tenemos dos tipos de predictores no lineales, P (una RNRC) y S (una RRS o una RPR, ambas con comportamientos similares), y que, de manera opcional, alimentamos un predictor lineal L con sus salidas. Sea G_P, G_S las GP de los predictores no lineales por separado, G_L la GP del predictor lineal, y G_{PL}, G_{SL} las GP del modelo híbrido en cascada. A partir de los resultados anteriores podemos concluir:[5]

$$G_{PL} > G_S \tag{9.2}$$
$$G_{PL} > G_P \tag{9.3}$$
$$G_{SL} < G_S \tag{9.4}$$
$$G_{PL} > G_L \tag{9.5}$$
$$G_{SL} < G_L \tag{9.6}$$

De (9.5) y (9.6), podemos concluir que P filtra adecuadamente la señal para su posterior tratamiento lineal cancelando al menos localmente las no linealidades, mientras que, por el contrario, S parece amplificar estas no linealidades y degradar el rendimiento del filtro lineal. Este aspecto parece importante y merece un estudio más detallado.

De (9.3) y (9.4) deducimos que ninguna de las configuraciones en cascada es adecuada para la RRS o la RPR, mientras que es más que recomendable para la RNRC. Las ecuaciones (9.2) y (9.4) afirman la superioridad de la RNRC en cascada sobre los otros modelos recurrentes.

Al comparar la RPR y la RRS, esta da GP ligeramente superiores. Una explicación posible es que al usar la RRS se necesita usar correctamente la información del estado, mientras que con la RPR podemos ignorar esta información y concentrarnos exclusivamente en las conexiones directas entre las capas de entrada y salida. La dependencia positiva del orden de entrada p es evidente al usar el FKED, pero es menos obvia en el caso del descenso por el gradiente (de hecho, en algunos casos, al incrementar p se reduce la correspondiente GP).

Finalmente, en experimentos adicionales, los predictores basados en la RRS y la RPR fueron introducidos en un sistema de codificación de voz *real* que seguía el estándar G721 relativo a la modulación adaptativa diferencial de pulsos (ADPCM, *adaptive differential pulse code modulation*) (Benvenuto et al. 1987). Reemplazamos el predictor IIR del estándar G721 (dos polos y seis ceros) con RNR y RPR, sin cambiar el cuantizador adaptativo del estándar. Los nuevos resultados confirman los anteriores: en este caso, solo

[5] La ecuación (9.3) debe de ser cierta, ya que Baltersee y Chambers no muestran resultados individuales para la RNRC.

el FKED obtiene tasas de compresión similares a las del filtro IIR original y el descenso por el gradiente da ganancias mucho más bajas.

Este trabajo constata, de nuevo, la notable superioridad del algoritmo de entrenamiento basado en el FKED sobre el descenso por el gradiente. Además, la señal predicha por RRS y RPR presenta un carácter no lineal más marcado que el de la propia señal original, lo que hace inviable un predictor lineal posterior en una configuración en cascada. El rendimiento de la RNRC, sin embargo, es mejorado por el predictor lineal.

Algunos trabajos anteriores detectaron severas limitaciones (Gers et al. 2001; Hallas y Dorffner 1998) de las RNR al ser aplicadas a tareas de predicción numérica no lineales. Los hallazgos de este capítulo sugieren similares conclusiones.

10. CONCLUSIONES Y PERSPECTIVA

A modo de clausura, se presentan ahora las principales conclusiones, las contribuciones de la tesis, y algunas ideas para desarrollar futuras investigaciones.

10.1. Conclusiones

En esta tesis se han estudiado diversas aplicaciones de las redes neuronales recurrentes (RNR, capítulo 2) de tiempo discreto usadas en modo predictivo. Como se vio al plantear inicialmente los problemas a resolver (capítulo 1), una estimación adecuada del siguiente elemento de una secuencia temporal puede ser útil para comprimir secuencias simbólicas, inferir lenguajes, desambiguar palabras homógrafas o comprimir señales de voz digitalizada, tareas cuyo estudio se ha abordado en los capítulos 6, 7, 8 y 9, respectivamente.

Se ha podido comprobar que RNR como la red recurrente simple (RRS) o la red parcialmente recurrente (RPR) no tienen tanta memoria como podría presumirse en primera instancia (capítulo 3). Aunque trabajan adecuadamente con lenguajes sencillos y altamente estructurados (especialmente lenguajes regulares), surgen diversos problemas a la hora de considerar fuentes secuenciales más elaboradas. Así, las RNR estudiadas no logran capturar acertadamente la dinámica de los textos en lenguaje natural y son claramente superadas por modelos más sencillos como los basados en n-gramas (capítulo 6). Por otro lado, los resultados obtenidos con secuencias numéricas correspondientes a señales de voz digitalizada son poco alentadores (capítulo 9): las ganancias de predicción de las redes recurrentes clásicas no superan las obtenidas con sencillos filtros lineales entrenados mediante el algoritmo de mínimos cuadrados recursivo (RLS).

Estos experimentos se llevaron a cabo considerando dos algoritmos de entrenamiento: el sencillo descenso por el gradiente y el más complejo filtro de Kalman extendido desacoplado (FKED). Aunque los problemas constatados se siguen produciendo al utilizar el FKED, este saca mayor provecho de la capacidad memorística teórica de las RNR y supera ampliamente en todos los casos estudiados los resultados del descenso por el gradiente.

En cualquier caso, debe tenerse en cuenta que los experimentos desarrollados en los capítulos 6 y 9 consideran un modo de operación completamente en línea (capítulo 4), es decir, las salidas de la RNR son utilizadas inmediatamente y la red no tiene opción de reconsiderar ninguna de ellas. La predicción en línea es, por tanto, más exigente que la realizada fuera de línea, pero este es el escenario predictivo a estudio.

Por otro lado, un problema adicional surge cuando la estimación del siguiente elemento de la secuencia depende del valor de un elemento relativamente lejano en el tiempo. Estas dependecias a largo plazo (capítulo 4) suelen ser un obstáculo insalvable para la mayoría de las RNR. Este problema puede estudiarse desde la perspectiva del gradiente evanescente (capítulo 4), fenómeno que afecta en mayor o menor medida a todos los algoritmos de entrenamiento basados en el cálculo de la derivada de la función de error y que les impide manejar con corrección este tipo de dependencias. Para superar este problema, recientemente se ha propuesto el modelo neuronal de memorias a corto y largo plazo (capítulo 3), conocido como red LSTM.

En esta tesis se ha aplicado la red LSTM a la inferencia mediante predicción de dos tipos de lenguajes, ambos con dependencias a largo plazo: un lenguaje regular y un lenguaje sensible al contexto. Las RNR convencionales no pueden manejar las dependencias a largo plazo presentes en estas secuencias. Sin embargo, en algunos trabajos previos, varios autores comprobaron que la red LSTM sí podía hacerlo. En esta tesis he llevado aún más lejos (capítulo 7) esta capacidad al combinar la red LSTM con el FKED. Con este algoritmo de entrenamiento se pueden superar dependencias separadas por varios miles de símbolos, valores nunca alcanzados anteriormente con LSTM. Estos resultados corroboran las afirmaciones hechas más arriba: el FKED combate eficazmente el efecto de amnesia de las RNR; el mayor coste temporal del FKED con respecto al habitual algoritmo de descenso por el gradiente se compensa muchas veces con una velocidad de aprendizaje superior. Además, la solución desarrollada por la red LSTM con el FKED es igual a la que se obtiene con el algoritmo de descenso por el gradiente, pese a las diferencias notables existentes entre ambos algoritmos de entrenamiento.

Finalmente, se ha usado de una forma original la información secuencial almacenada en el estado de una RRS para desambiguar las palabras homógrafas de una oración (capítulo 8). Los resultados son similares a los proporcionados por un modelo oculto de Markov (MOM), el modelo de referencia habitual en estos casos (apéndice A), con igual número de parámetros. Los resultados son también similares a los de otros enfoques neuronales que, sin embargo, necesitan textos completamente etiquetados para el entrenamiento, necesidad superada con el planteamiento aquí propuesto.

10.2. Contribuciones de esta tesis

Las principales aportaciones de esta tesis son las siguientes:

1. Los resultados obtenidos con el FKED en las distintas tareas predictivas abordadas en la tesis confirman los de otros trabajos: el FKED supera en muchos aspectos al descenso por el gradiente como algoritmo de entrenamiento en línea de RNR. Todo trabajo con RNR, especialmente aquellos que estudian procesos que se desarrollan en línea, debería considerar el FKED como algoritmo de entrenamiento en los experimentos.

2. En particular, el FKED ha llevado un poco más allá la capacidad de la red LSTM para manejar adecuadamente las dependencias a largo plazo. Si bien los resultados previos sobre el lenguaje sensible al contexto $a^n b^n c^n$ obtenidos mediante el descenso por el gradiente superaban ampliamente los de otras RNR, el FKED supera a ambas tanto en velocidad de aprendizaje como en capacidad de generalización: solo con observar secuencias del lenguaje con $n \in [1, 10]$, se consiguen generalizaciones para valores de n por encima de 1000.

3. Se ha aplicado la red LSTM por primera vez a una tarea de predicción completamente en línea. Los trabajos anteriores habían utilizado, como mucho, un entrenamiento en línea por secuencias (capítulo 4). La red LSTM es adecuada para el entrenamiento en línea puro: la adición de la compuerta de olvido permite que el estado de las celdas de la red no aumente sin control, por muy larga que sea la secuencia procesada de forma continua.

4. Aunque no es la primera vez que una RNR se aplica a la desambiguación categorial, sí que es la primera vez que el problema se plantea desde un punto de vista predictivo. Es más, se trata del primer enfoque neuronal (recurrente o no) que no necesita un corpus completamente etiquetado para el entrenamiento, aspecto este que lo acerca más a modelos como los MOM.

5. Las alternativas recurrentes usadas con anterioridad para realizar la predicción sobre secuencias de voz se basaban en la red neuronal recurrente en cascada (RNRC), un modelo especializado para este tipo de tarea (capítulo 3). En esta tesis se ha estudiado si modelos de RNR más sencillos son capaces de mantener el rendimiento de la RNRC. Los resultados, sin embargo, son poco alentadores y remarcan las conclusiones de otros trabajos que han constatado la superioridad de modelos simples basados en una ventana temporal de entradas sobre las RNR a la hora de procesar algunas secuencias numéricas.

10.3. Sugerencias para próximos trabajos

Algunos aspectos que merecen un estudio más detallado en próximos trabajos son los siguientes:

1. Los valores de los parámetros del FKED usados en los experimentos se determinaron a partir de una serie de experimentos preliminares. Para paliar esta búsqueda empírica de los valores más adecuados, debe realizarse un estudio sistemático sobre la influencia de los distintos parámetros del FKED en el rendimiento del algoritmo.

2. En el capítulo 6 se ha estudiado la predicción en línea sobre secuencias generadas por máquinas de estados finitos. Además, en el apartado 5.1.6 se ha visto las principales diferencias del enfoque allí seguido con la inferencia gramatical clásica con RNR.

   Hay varios métodos propuestos (Giles et al. 1992) para extraer modelos de estados finitos a partir del conocimiento simbólico adquirido por una RNR cuando ha sido entrenada para inferir lenguajes regulares mediante ese enfoque clásico. Debería estudiarse si todavía es posible extraer algún modelo de estados finitos cuando el aprendizaje se realiza en línea y si es comparable al extraído fuera de línea. Aunque la convergencia de la RNR a las probabilidades reales parece difícil de demostrar en este caso, puede estudiarse empíricamente si la presencia reiterada de una subsecuencia hace que el estado de la red alcance un punto fijo tras procesar dicha subsecuencia o, al menos, una región del espacio de estados desde la que se obtengan salidas similares. La separación en regiones del espacio de estados puede realizarse, por ejemplo, mediante cuantización vectorial sobre la secuencia de activaciones del estado (Čerňanský y Beňušková 2001). Un estudio teórico debería determinar si la salida de la red puede todavía considerarse en el caso de la predicción en línea como una buena aproximación a las probabilidades reales.

3. El proceso de extración de un modelo de estados finitos discutido en el punto anterior también debería aplicarse al modelo aprendido en el caso de la desambiguación categorial (capítulo 8) con vistas a formular el conjunto de reglas de etiquetado aprendidas por el sistema.

4. También debe evaluarse la influencia del tamaño del corpus de entrenamiento en el rendimiento del desambiguador categorial, así como la adecuación de la segunda, tercera, etc. neurona de salida con mayor activación: ¿corresponde a una categoría válida para la palabra?

5. En cuanto a la predicción de secuencias textuales, se puede estudiar el rendimiento de la red LSTM aplicada a esta tarea para comprobar

si supera los resultados de las RNR tradicionales, así como analizar la solución alcanzada por el modelo.

6. En cuanto a la inferencia del lenguaje $a^n b^n c^n$ con la red LSTM, debería considerarse otros lenguajes sensibles al contexto, además de tener en cuenta también muestras negativas.

7. No hay trabajos que intenten extraer el modelo (un autómata finito o un autómata de pila, por ejemplo) aprendido por la red LSTM cuando se aplica a la inferencia de lenguajes. Esta labor debe llevarse a cabo, además de realizar un estudio profundo sobre las clases de lenguajes aprendibles con LSTM, una caracterización general mucho más importante que el hecho de que pueda aprender uno u otro lenguaje determinado.

8. El rendimiento de las redes LSTM a la hora de manejar dependencias a largo plazo puede compararse con otras alternativas como los MOM jerárquicos (Fine et al. 1998).

9. El modelo LSTM intenta conseguir un flujo de error constante a través de los carruseles de error constante (CEC) para superar el problema del gradiente evanescente y gestionar adecuadamente las dependencias a largo plazo (capítulo 4). A ello contribuyen tanto la configuración topológica del modelo como la forma de calcular las derivadas parciales de la función de error.

   Un estudio interesante es determinar la importancia de estas derivadas a la hora de considerar las dependencias a largo plazo. Para ello se puede aplicar un algoritmo como Alopex (véase el apartado 4.10), que no se basa en el gradiente, y evaluar los resultados obtenidos.

10. También puede ser interesante estudiar si el hecho de que un bloque de memoria de la red LSTM pueda tener más de una celda aporta alguna ventaja al modelo. Aparentemente, al compartir todas las celdas de un bloque de memoria las mismas compuertas, las celdas deberían terminar adoptando un rol similar y resultar incluso redundantes.

11. Recientemente se han propuesto otras alternativas distintas al filtro de Kalman extendido (FKE) para realizar el filtrado de sistemas no lineales (Julier y Uhlmann 1997; Merwe et al. 2000) que parecen mejorar los resultados del FKE. Puede considerarse su aplicación a algunas de las tareas consideradas en esta tesis.

A. MODELOS DE MARKOV PARA EL ETIQUETADO CATEGORIAL

En este apéndice se muestra la forma de ajustar los parametros de un modelo oculto de Markov de forma que se pueda realizar la desambiguación categorial de las palabras de una oración. Los modelos ocultos de Markov se usan en los experimentos del capítulo 8.

Un modelo oculto de Markov (MOM) (Rabiner 1989) es un sistema dinámico de tiempo discreto capaz de emitir una secuencia de salidas observables. Un MOM se define como un modelo de estados de la forma $\lambda = (S, V, A, B, \pi)$, donde S es el conjunto de estados, V es el conjunto de salidas observables, A son las probabilidades de transición entre estados, B son las probabilidades de que cada estado emita los posibles elementos observables y π define las probabilidades de que el sistema comience desde cada uno de los estados de S. El sistema emite una salida cada vez que llega a un estado tras realizar una transición.

Las aplicaciones de los MOM son múltiples. Si logramos descubrir el MOM más verosímil para una determinada secuencia de observaciones, tendremos un modelo del proceso subyacente; de esto se encargan algoritmos como el de Baum y Welch, que discutiremos a continuación. Por otro lado, es posible obtener la probabilidad bajo un MOM determinado de una secuencia de observaciones y, lo que muchas veces es aún más importante, encontrar la secuencia de estados que produce el camino de máxima verosimilitud para una secuencia de observaciones dada. La secuencia de estados puede interpretarse como una explicación de la observación si cada estado tiene un significado diferenciado; para encontrarla existen algoritmos como el de Viterbi.

En este apéndice se presenta en detalle,[1] para paliar la ausencia de descripciones adecuadas en la bibliografía, una adaptación del algoritmo de

[1] Los contenidos de este apéndice están basados en una comunicación personal de los Drs. Rafael C. Carrasco y Mikel L. Forcada.

Baum y Welch que facilita la desambiguación categorial y se da una breve justificación matemática de la misma siguiendo la notación de Rabiner (1989).

A.1. Aplicación al etiquetado categorial

A.1.1. Simplificaciones

Sea el modelo oculto de Markov $\lambda = (S, V, A, B, \pi)$ con estados $S = \{s_1, s_2, \dots, s_N\}$, salidas $V = \{v_1, v_2, \dots, v_M\}$, probabilidades de transición $A = \{a_{ij}\}$ $(i, j = 1 \dots, N)$, probabilidades de emisión $B = \{b_j(v_k)\}$ $(j = 1, \dots, N,\ k = 1, \dots M)$ y probabilidades iniciales $\pi = \{\pi_i\}$ $(i = 1, \dots, N)$. Cuando este tipo de modelos se usan para etiquetar un texto podemos suponer que cada palabra ha sido substituida (usando un diccionario o léxico) por la clase de ambigüedad o conjunto de etiquetas que admite esa palabra. En este caso, cada estado del MOM se corresponde con una etiqueta léxica y el conjunto de salidas está formado por todas las posibles clases de ambigüedad (los elementos de V son subconjuntos de S). Por ello, usaremos indistintamente las nociones de *palabra* y *clase de ambigüedad* para referirnos a las salidas del modelo oculto.

Además, podemos hacer las siguientes suposiciones:

1. Que la secuencia de texto $O_1 \dots O_T$ que queremos analizar siempre viene precedida de una palabra no ambigua $O_0 = \{I\}$. Parece razonable que I sea la etiqueta que representa el final de oración.[2]

2. Que el texto acaba en una palabra no ambigua $O_T = \{F\}$. Aquí es si cabe más razonable tomar F como el punto final de las oraciones, dado que normalmente tanto los textos de entrenamiento como los de prueba contienen frases completas.

3. Que toda clase de ambigüedad contiene, al menos, la etiqueta correcta. Por ejemplo, desde el estado asociado a XVB no se puede generar una clase $\{XJJ, XNN\}$. Consiguientemente, una clase no ambigua $\{X\}$ solo puede ser generada por el estado correspondiente X.

A.1.2. Probabilidades hacia adelante

El hecho de añadir la clase no ambigua $O_0 = \{I\}$ evita que π_i sea simplemente 1 para la categoría de la primera palabra del corpus de entrenamiento

[2] Además, así el etiquetado de las palabras de una oración determinada no depende de si la oración se encuentra al principio del texto o en cualquier otra parte del mismo.

y cero para las demás. Esto no parece adecuado: primero, porque otro texto también correcto podría empezar por otra categoría; y segundo, porque no tendría mucho sentido realizar un entrenamiento, ya que se puede fijar el valor de π_i directamente.

Teniendo en cuenta que $\pi_i = \delta_{I,s_i}$ y $b_i(\{I\}) = \delta_{I,s_i}$, donde δ es la delta de Kronecker definida en (4.3), podemos rescribir las ecuaciones (19) y (20) de Rabiner (1989) empezando en $t = 0$ como sigue:

$$\alpha_0(i) = \delta_{I,s_i} \tag{A.1}$$

y, para $t = 1, \ldots, T$,

$$\alpha_t(i) = \sum_{j=1}^{N} \alpha_{t-1}(j)\ a_{ji}\ b_i(O_t) \tag{A.2}$$

De esta manera podemos prescindir de las probabilidades iniciales. Esta forma es, además, más semejante a la de las probabilidades hacia atrás del apartado siguiente.

A.1.3. Probabilidades hacia atrás

De forma análoga, las variables hacia atrás, correspondientes a las ecuaciones (24) y (25) de Rabiner (1989), son:

$$\beta_T(i) = 1 \tag{A.3}$$

y, para $t = 1, \ldots, T$,

$$\beta_{t-1}(i) = \sum_{j=1}^{N} a_{ij}\ b_j(O_t)\ \beta_t(j) \tag{A.4}$$

A.1.4. Otras probabilidades

La probabilidad de una secuencia $\mathbf{O} = O_1 \ldots O_T$ se puede calcular a a partir de las probabilidades hacia adelante y hacia atrás de la siguiente forma:

$$P(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \alpha_t(i)\beta_t(i) \tag{A.5}$$

donde t puede elegirse libremente en $\{0, 1, \ldots, T\}$. En particular,

$$P(\mathbf{O}|\lambda) = \beta_0(I) = \alpha_T(F) \tag{A.6}$$

La segunda igualdad es consecuencia de la hipótesis 2 y de que podamos escribir directamente $\alpha_T(i) = 0$ si $i \neq F$ siempre que $O_T = \{F\}$.

El número esperado de veces que se pasa por el estado i al generar la secuencia $\mathbf{O}$ se define como:

$$\Gamma_i = \sum_{t=0}^{T-1} \gamma_t(i) \tag{A.7}$$

donde (Rabiner 1989, ec. 27):

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(\mathbf{O}|\lambda)} \tag{A.8}$$

Por tanto,

$$\Gamma_i = \frac{1}{P(\mathbf{O}|\lambda)} \sum_{t=0}^{T-1} \alpha_t(i)\beta_t(i) \tag{A.9}$$

Para un texto completo (que empieza y acaba por final de frase, esto es $I = F$) se cumple que $\alpha_0(i)\beta_0(i) = \alpha_T(i)\beta_T(i) = P(\mathbf{O}|\lambda)$ por lo que, trivialmente, podemos desplazar la suma temporal como sigue:

$$\sum_{t=0}^{T-1} \alpha_t(i)\beta_t(i) = \sum_{t=1}^{T} \alpha_t(i)\beta_t(i) \tag{A.10}$$

Este resultado (que nos resultará útil más adelante) puede entenderse de forma intuitiva: el estado final de frase se visita una vez al principio del texto y otra al final; como solo tenemos que contar una, da igual cuál contemos. Esto hace que podamos cambiar algunas de las sumas temporales de 1 a T que aparecen en Rabiner (1989) por sumas de 0 a $T-1$.

A continuación definimos, por un lado, el número esperado de veces que el modelo pasa del estado i al estado j,

$$\Xi_{ij} = \sum_{t=0}^{T-1} \xi_t(i,j) \tag{A.11}$$

donde (Rabiner 1989, ec. 37):

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(\mathbf{O}|\lambda)} \tag{A.12}$$

Así,

$$\Xi_{ij} = \frac{1}{P(\mathbf{O}|\lambda)} \sum_{t=0}^{T-1} \alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j) \tag{A.13}$$

Por otro lado, definimos el número esperado de veces que el modelo pasa por el estado j mientras se observa el símbolo v_k como:

$$\Phi_{jk} = \sum_{t=0}^{T-1} \varphi_t(j,k) \tag{A.14}$$

donde:

$$\varphi_t(j,k) = \frac{\alpha_t(j)\beta_t(j)\delta_{v_k,O_t}}{P(\mathbf{O}|\lambda)} \tag{A.15}$$

Es decir,

$$\Phi_{jk} = \frac{1}{P(\mathbf{O}|\lambda)} \sum_{t=0}^{T-1} \alpha_t(j)\beta_t(j)\delta_{v_k,O_t} \tag{A.16}$$

Para la depuración del programa puede ser útil tener en cuenta que:

$$\Gamma_i = \sum_{k=1}^{M} \Phi_{ik} = \sum_{j=1}^{N} \Xi_{ij} \tag{A.17}$$

El cálculo de Γ_i, Ξ_{ij} y Φ_{jk} requiere realizar dos pasadas sobre el texto: una hacia adelante y otra hacia atrás. En la primera se calculan todos los $\alpha_t(i)$ y la verosimilitud del texto $P(\mathbf{O}|\lambda)$ y en la segunda se calculan Γ_i, Ξ_{ij} y Φ_{jk} incrementalmente, por lo que basta con almacenar $\beta_{t+1}(j)$ en cada iteración.

A.1.5. Nuevos parámetros

Con las suposiciones anteriores, obtenemos las siguientes fórmulas de Baum y Welch para la actualización de los parámetros:

$$\overline{a_{ij}} = \frac{\Xi_{ij}}{\Gamma_i} \tag{A.18}$$

y

$$\overline{b_j}(k) = \frac{\Phi_{jk}}{\Gamma_j} \tag{A.19}$$

donde se ha hecho uso de la adición de la clase de ambigüedad inicial para que los denominadores sean iguales, en contraste con las ecuaciones (40b) y (40c) de Rabiner (1989), en las que estos denominadores son diferentes.

A.1.6. Segmentación

Cada vez que en el texto que se está etiquetando aparece una palabra no ambigua, el modelo de Markov oculto solo puede estar en el estado correspondiente a su etiqueta (hipótesis 3). Esto permite reorganizar el cálculo de manera muy eficiente como ya se sugiere en el trabajo de Cutting et al. (1992), ya que no es preciso mantener en memoria todo el texto sino solo la porción de texto entre dos categorías no ambiguas (ambas inclusive) y tratarla como si fuese un texto completo.

Supongamos que el texto está segmentado en G segmentos o grupos. Cada segmento g empieza en $t = i_g$ y termina en $t = f_g$, con $f_g = i_{g+1}$ y $T_g = f_g - i_g$, y tiene una palabra inicial que pertenece a la clase no ambigua $\{I_g\}$, una palabra final que pertenece a la clase no ambigua $\{F_g\}$, y cero o más palabras, todas ambiguas, situadas entre ellas. Estos segmentos son normalmente bastante cortos.

Cada uno de los valores esperados Ξ_{ij}, Φ_{jk} y Γ_i se pueden calcular como una suma para todos los segmentos:

$$\Xi_{ij} = \sum_{g=1}^{G} \Xi_{ij}^{(g)} \tag{A.20}$$

$$\Phi_{jk} = \sum_{g=1}^{G} \Phi_{jk}^{(g)} \tag{A.21}$$

$$\Gamma_i = \sum_{g=1}^{G} \Gamma_i^{(g)} \tag{A.22}$$

y el cálculo para cada segmento se puede realizar como si se tratase de un texto independiente, usando solo información local al mismo.

Describiremos con algún detalle el cálculo de $\Xi_{ij}^{(g)}$.[3] Es sencillo comprobar que:

$$\alpha_{i_g}(i) = P(O_1 \ldots O_{i_g})\delta_{i,I_g} \tag{A.23}$$

y que:

$$P(O_1 \ldots O_T) = P(O_1 \ldots O_{f_g})\beta_{f_g}(F_g) \tag{A.24}$$

[3] El cálculo de $\Phi_{jk}^{(g)}$ y $\Gamma_i^{(g)}$ es completamente análogo.

Si definimos, para $i, j = 1 \dots N$,

$$\alpha^g_{t-i_g}(i) = \frac{\alpha_t(i)}{\alpha_{i_g}(I_g)} \tag{A.25}$$

y

$$\beta^g_{t-i_g}(j) = \frac{\beta_t(j)}{\beta_{f_g}(F_g)} \tag{A.26}$$

se sigue que:

$$\Xi^{(g)}_{ij} = \frac{1}{P^g} \sum_{\tau=0}^{T_g-1} \alpha^g_\tau(i) a_{ij} b_j(O_{i_g+\tau+1}) \beta^g_{\tau+1}(j) \tag{A.27}$$

ecuación completamente análoga a la A.13 excepto por el detalle de que P^g no es $P(O_{i_g} \dots O_{f_g})$, porque en general $I_g \neq F_g$, sino:

$$P^{(g)} = \frac{P(\mathbf{O}|\lambda)}{\alpha_{i_g}(I_g)\beta_{f_g}(F_g)} = \alpha^g_{T_g}(F_g) \tag{A.28}$$

Las nuevas probabilidades hacia adelante α^g y las variables hacia atrás β^g de cada grupo se definen recursivamente de manera similar a las del texto completo:

$$\alpha^g_\tau(i) = \left[\sum_{j=1} \alpha^g_{\tau-1}(j) a_{ji} \right] b_i(O_{i_g+\tau}) \tag{A.29}$$

$$\alpha^g_0(i) = \delta_{i,I_g} \tag{A.30}$$

$$\beta^g_\tau(i) = \sum_{j=1} a_{ij} \beta^g_{\tau+1}(j) b_j(O_{i_g+\tau+1}) \tag{A.31}$$

$$\beta^g_{T_g}(i) = 1 \;\; \forall j \in [1, N] \tag{A.32}$$

Las ecuaciones (A.9) y (A.16) se convierten análogamente en:

$$\Gamma^{(g)}_i = \frac{1}{P^g} \sum_{\tau=0}^{T_g-1} \alpha^g_\tau(i) \beta^g_\tau(i) \tag{A.33}$$

y

$$\Phi^{(g)}_{jk} = \frac{1}{P^g} \sum_{\tau=0}^{T_g-1} \alpha^g_\tau(j) \beta^g_\tau(j) \delta_{v_k, O_{i_g+\tau}}. \tag{A.34}$$

Lo dicho al final del epígrafe A.1.4 sobre la implementación del cálculo de Γ_i, Ξ_{ij} y Φ_{jk} se puede adaptar fácilmente para el cálculo de $\Gamma^{(g)}_i$, $\Xi^{(g)}_{ij}$ y $\Phi^{(g)}_{jk}$.

A.1.7. Inicialización

El algoritmo de Baum y Welch se puede inicializar, en ausencia de todo conocimiento, con los siguientes valores:

$$a_{ij} = \frac{1}{N} \tag{A.35}$$

y

$$b_j(k) = \begin{cases} \frac{1}{N_j} & \text{si } s_j \in v_k \\ 0 & \text{si no} \end{cases} \tag{A.36}$$

donde N_j es el número de clases de ambigüedad en las que puede manifestarse la etiqueta s_j, es decir, $\text{card}\{v : s_j \in v\}$.

Esta inicialización es la utilizada en los experimentos del capítulo 8.

ÍNDICE DE SÍMBOLOS IMPORTANTES

ÍNDICE DE ABREVIATURAS

BIBLIOGRAFÍA

ALQUÉZAR, R. y A. SANFELIU (1994). "Inference and recognition of regular grammars by training recurrent neural networks to learn the next-symbol prediction task", en F. Casacuberta y A. Sanfeliu, coordinadores, *Advances in pattern recognition and aplications*, World Scientific.

ALQUÉZAR, R. y A. SANFELIU (1995). "An algebraic framework to represent finite state automata in single-layer recurrent neural networks", *Neural Computation*, **7**(5), 931–949.

AUSSEM, ALEX, FIONN MURTAGH y MARC SARAZIN (1995). "Dynamical recurrent neural networks – towards environmental time series prediction", *International Journal Neural Systems*, **6**, 145–170.

BALTERSEE, J. y J. A. CHAMBERS (1998). "Non-linear adaptive prediction of speech signals using a pipelined recurrent network", *IEEE Transactions on Signal Processing*, **46**(8).

BARNWELL, T. P., K.ÑAYEBI y C. H. RICHARDSON (1996). *Speech coding: a computer laboratory textbook*, Georgia Tech digital signal processing laboratory series, John Wiley & Sons.

BELL, T. C., J. G. CLEARY y I. H. WITTEN (1990). *Text compression*, Prentice-Hall.

BENGIO, Y., P. SIMARD y P. FRASCONI (1994). "Learning long-term dependencies with gradient descent is difficult", *IEEE Transactions on Neural Networks*, **5**(2), 157–166.

BENVENUTO, N., G. BERTOCCI y W. R. DAUMER (1987). "The 32-kb/s ADPCM coding standard", *AT&T Technical Journal*, **2**, 270–280.

BIANCHINI, M., M. GORI y M. MAGGINI (1994). "On the problem of local minima in recurrent neural networks", *IEEE Transactions on Neural Networks*, **5**(2), 167–177.

BIRGMEIER, M. (1996). "Nonlinear prediction of speech signals using radial basis function networks", en *Proceedings of the European Signal Processing Conference*.

Blair, A. y J. B. Pollack (1997). “Analysis of dynamical recognizers”, *Neural Computation*, **9**(5), 1127–1142.

Bodén, M. y J. Wiles (2000). “Context-free and context-sensitive dynamics in recurrent neural networks”, *Connection Science*, **12**(3).

Bodén, M. y J. Wiles (2002). “On learning context free and context sensitive languages”, *IEEE Transactions on Neural Networks*, **13**(2), 491–493.

Brill, E. (1992). “A simple rule-based part-of-speech tagger”, en *Proceedings Third Conference on Applied Natural Language Processing*.

Bulsari, A. B. y H. Saxén (1995). “A recurrent network for modeling noisy temporal sequences”, *Neurocomputing*, **7**(1), 29–40.

Burrows, M. y D. J. Wheeler (1994). “A block-sorting lossless data compression algorithm”, *informe técnico 124*, Digital Systems Research Center.

Carrasco, R. C., M. L. Forcada y L. Santamaría (1996). “Inferring stochastic regular grammars with recurrent neural networks”, en *Grammatical inference: learning syntax from sentences*, vol. 1147 de *Lecture Notes in Artificial Intelligence*, págs. 274–281, Springer-Verlag, Berlín.

Carrasco, R. C., M. L. Forcada, M. A. Valdés-Muñoz y R. P. Ñeco (2000). “Stable-encoding of finite-state machines in discrete-time recurrent neural nets with sigmoid units”, *Neural Computation*, **12**(9), 2129–2174.

Castaño, M. A., E. Vidal y F. Casacuberta (1995). “Finite state automata and connectionist machines: a survey”, en *New trends in neural computation*, vol. 930 de *Lecture Notes in Computer Science*, págs. 433–440, Springer-Verlag.

Cauwenberghs, G. (1993). “A fast-stochastic error-descent algorithm for supervised learning and optimization”, en S. J. Hanson, J. D. Cowan y C. L. Giles, coordinadores, *Advances in Neural Information Processing Systems*, vol. 5, págs. 244–251, Morgan Kaufmann.

Čerňanský, M. y L. Beňušková (2001). “Finite-state Reber automaton and the recurrent neural networks trained in supervised and unsupervised manner”, en G. Dorffner, H. Bischof y K. Hornik, coordinadores, *Proceedings of the International Conference on Artificial Neural Networks*, vol. 2130 de *Lecture Notes in Computer Science*, págs. 737–742, Springer-Verlag, Berlín.

CHALUP, S. y A. BLAIR (1999). “Hill climbing in recurrent neural networks for learning the $a^n b^n c^n$ language”, en *Proceedings of the 6th Conference on Neural Information Processing*, págs. 508–513.

CHAN, L. W. y C. C. SZETO (1999). “Training recurrent network with block-diagonal approximated Levenberg-Marquardt algorithm”, en *Proceedings of the International Joint Conference on Neural Networks*.

CHANG, W. F. y M. W. MAK (1999). “A conjugate gradient learning algorithm for recurrent neural networks”, *Neurocomputing*, **24**, 173–189.

CHARNIAK, E. (1993). *Statistical language learning*, MIT Press, Cambridge.

CID-SUEIRO, J., A. ARTES-RODRIGUEZ y A. R. FIGUEIRAS-VIDAL (1994). “Recurrent radial basis function networks for optimal symbol-by-symbol equalization”, *Signal Proc.*, **40**, 53–63.

CLEEREMANS, A., D. SERVAN-SCHREIBER y J. L. MCCLELLAND (1989). “Finite state automata and simple recurrent networks”, *Neural Computation*, **1**(3), 372–381.

COVER, T. M. y J. A. THOMAS (1991). *Elements of information theory*, John Wiley and Sons, Nueva York.

CUTTING, D., J. KUPIEC, J. PEDERSEN y P. SIBUN (1992). “A practical part-of-speech tagger”, en *Proceedings of Third Conference on Applied Natural Language Processing*, págs. 133–140.

DAVIES, R. B. (1994). “Writing a matrix package in C++”, en *Second Annual Object-Oriented Numerics Conference*, págs. 207–213.

ELMAN, J. L. (1990). “Finding structure in time”, *Cognitive Science*, **14**, 179–211.

ELMAN, J. L. (1991). “Distributed representations, simple recurrent networks, and grammatical structure”, *Machine Learning*, **7**, 195–225.

FELDKAMP, L. A. y G. V. PUSKORIUS (1994). “Training controllers for robustness: multi-stream DEKF”, en *IEEE International Conference on Neural Networks*, págs. 2377–2382.

FINE, S., Y. SINGER y N. TISHBY (1998). “The hierarchical hidden Markov model: analysis and applications”, *Machine Learning*, **32**(1), 41–62.

FORCADA, M. L. (2001). “Corpus-based stochastic finite-state predictive text entry for reduced keyboards: application to Catalan”, en *Procesamiento del Lenguaje Natural*, vol. 27, págs. 65–70.

Forcada, M. L. y R. C. Carrasco (1995). “Learning the initial state of a second order recurrent neural network during regular-language inference”, *Neural Computation*, **7**, 923–930.

Forcada, M. L. y R. C. Carrasco (2001). “Finite-state computation in analog neural networks: steps towards biologically plausible models?”, en S. Wermter, J. Austin y D. Willshaw, coordinadores, *Emergent Neural Computational Architectures based on Neuroscience*, vol. 2036 de *Lecture Notes in Computer Science*, págs. 482–486, Springer-Verlag.

Forcada, M. L. y M. Gori (2001). “Neural nets, recurrent”, en J. G. Webster, coordinador, *Wiley Encyclopedia of Electrical and Electronics Engineering*, John Wiley & Sons.

Forcada, Mikel L., coordinador (2002). *Neural networks: automata and formal models of computation*, inédito, disponible en `http://www.dlsi.ua.es/~mlf/nnafmc/`.

Gers, F. A., D. Eck y J Schmidhuber (2001). “Applying LSTM to time series predictable through time-window approaches”, en *Proceedings of the International Conference on Artificial Neural Networks*.

Gers, F. A., J. A. Pérez-Ortiz, D. Eck y J. Schmidhuber (2002a). “DEKF–LSTM”, en M. Verleysen, coordinador, *Proceedings of the European Symposium on Artificial Neural Networks*, págs. 369–376, D-side Publications.

Gers, F. A., J. A. Pérez-Ortiz, D. Eck y J. Schmidhuber (2002b). “Learning context sensitive languages with LSTM trained with Kalman filters”, en *Proceedings of the International Conference on Artificial Neural Networks*, Lecture Notes in Computer Science, Springer-Verlag, Berlín, aceptado.

Gers, F. A. y J. Schmidhuber (2001). “LSTM recurrent networks learn simple context free and context sensitive languages”, *IEEE Transactions on Neural Networks*, **12**(6), 1333–1340.

Gers, F. A., J. Schmidhuber y F. Cummins (1999). “Learning to forget: continual prediction with LSTM”, en *Proceedings of the International Conference on Artificial Neural Networks*, págs. 850–855.

Gers, F. A., J. Schmidhuber y F. Cummins (2000). “Learning to forget: continual prediction with LSTM”, *Neural Computation*, **12**(10), 2451–2471.

Giles, C. L., C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun y Y. C. Lee (1992). “Learning and extracting finite state automata with second-order recurrent neural networks”, *Neural Computation*, **4**(3), 393–405.

GOUDREAU, M. W., C. L. GILES, S. T. CHAKRADHAR y D. CHEN (1994). "First-order vs. second order single layer recurrent neural networks", *IEEE Transactions on Neural Networks*, **5**(3), 511–513.

HALLAS, M. y G. DORFFNER (1998). "A comparative study on feedforward and recurrent neural networks in time series prediction using gradient descent learning", en R. Trappl, coordinador, *Cybernetics and Systems 98, Proceedings of 14th European Meeting on Cybernetics and Systems Research*, págs. 644–647.

HAYKIN, S. (1999). *Neural networks: a comprehensive foundation*, Prentice-Hall, New Jersey, 2.ª ed.

HAYKIN, S., coordinador (2001). *Kalman filtering and neural networks*, Wiley.

HAYKIN, S. y L. LI (1995). "Non-linear adaptive prediction of non-stationary signals", *IEEE Transactions on Signal Processing*, **43**(2), 526–535.

HERTZ, J., A. KROGH y R. G. PALMER (1991). *Introduction to the theory of neural computation*, Addison-Wesley.

HILBORN, R. C. (2000). *Chaos and nonlinear dynamics: an introduction for scientists and engineers*, Oxford University Press.

HOCHREITER, S., Y. BENGIO, P. FRASCONI y J. SCHMIDHUBER (2001). "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies", en S. C. Kremer y J. F. Kolen, coordinadores, *A field guide to dynamical recurrent neural networks*, IEEE Press.

HOCHREITER, S. y J. SCHMIDHUBER (1997). "Long short-term memory", *Neural Computation*, **9**(8), 1735–1780.

HOPCROFT, J. E. y J. D. ULLMAN (1979). *Introduction to automata theory, languages and computation*, Addison-Wesley.

HORNE, B. G. y D. R. HUSH (1996). "Bounds on the complexity of recurrent neural network implementations of finite state machines", *Neural networks*, **9**(2), 243–252.

JULIER, S. J. y J. K. UHLMANN (1997). "A new extension of the Kalman filter to nonlinear systems", en *Proceedings of AeroSense, the 11th International Symposium on Aerospace/Defence Sensing, Simulation and Controls*.

KALMAN, R. E. (1960). "A new approach to linear filtering and prediction problems", *Transaction of the ASME – Journal of Basic Engineering*, págs. 35–45.

KECHRIOTIS, G., E. ZERVAS y E. S. MANOLAKOS (1994). "Using recurrent neural networks for adaptive communication channel equalization", *IEEE Trans. on Neural Networks*, **5**(2), 267–278.

KLEENE, S. C. (1956). "Representation of events in nerve nets and finite automata", en C. E. Shannon y J. McCarthy, coordinadores, *Automata studies*, Princeton University Press.

KOLEN, J. F. y S. C. KREMER, coordinadores (2001). *A field guide to dynamical recurrent networks*, IEEE Press.

KREMER, S. C. (1997). "Parallel stochastic grammar induction", en *Proceedings International Conference on Neural Networks*, págs. 612–616.

LONG, P. M., A. I. NATSEV y J. S. VITTER (1999). "Text compression via alphabet re-representation", *Neural Networks*, **12**, 755–765.

MA, Q. y H. ISAHARA (1997). "Part-of-speech tagging of Thai corpus with the logically combined neural networks", en *Proceedings of the Natural Language Processing Pacific Rim Symposium*, págs. 537–540.

MA, Q., M. MURATA, M. UTIYAMA, K. UCHIMOTO y H. ISAHARA (1999). "Part of speech tagging with mixed approaches of neural networks and transformation rules", en *Workshop on Natural Language Processing and Neural Networks*, Beijing, China.

MAHONEY, M. V. (2000). "Fast text compression with neural networks", en *13th International FLAIRS Conference*, Orlando, Florida.

MANNING, C. D. y H. SCHÜTZE (1999). *Foundations of statistical natural language processing*, MIT Press.

MARCUS, M. P., B. SANTORINI y M. A. MARCINKIEWICZ (1993). "Building a large annotated corpus of English: the Penn Treebank", *Computational Linguistics*, **19**, 313–330.

MARQUES, N. C. y G. P. LOPES (1996). "Using neural nets for Portuguese part-of-speech tagging", en *Proceedings of the Fifth International Conference on The Cognitive Science of Natural Language Processing*, Dublin City University, Ireland.

MARTÍN, M. A., M. MORÁN y M. REYES (1995). *Iniciación al caos: sistemas dinámicos*, Síntesis, Madrid.

MCCLUSKEY, P. G. (1993). "Feedforward and recurrent neural networks and genetic programs for stock market and time series forecasting", *informe técnico CS-93-36*, Brown University.

McCulloch, W. S. y W. H. Pitts (1943). "A logical calculus of the ideas immanent in nervous activity", *Bulletin of Mathematical Biophysics*, págs. 115–133.

Merwe, R., A. Doucet, N. de Freitas y E. Wan (2000). "The unscented particle filter", en *Advances in Neural Information Processing Systems*, vol. 13, MIT Press.

Minsky, M. L. y S. A. Papert (1969). *Perceptrons*, MIT Press, Cambridge.

Mozer, M. C. (1994). "Neural network music composition by prediction: exploring the benefits of psychoacoustic constraints and multiscale processing", *Connection Science*, **6**, 247–280.

Narendra, K. S. y K. Parthasarathy (1990). "Identification and control of dynamical systems using neural networks", *IEEE Transactions on Neural Networks*, **1**, 4–27.

Nelson, M. (1991). "Arithmetic coding + statistical modeling = data compression", *Dr. Dobb's Journal*.

Nelson, M. y J.-L. Gailly (1995). *The data compression book*, M&T Books, New York, 2.ª ed.

Omlin, C. W. y C. L. Giles (1996). "Constructing deterministic finite-state automata in recurrent neural networks", *Journal of the ACM*, **43**(6), 937–972.

Oppenheim, A. V. y R. W. Schafer (1989). *Discrete-time signal processing*, Prentice-Hall.

Ortiz Fuentes, J. D. y M. L. Forcada (1997). "A comparison between recurrent neural architectures for digital equalization", en *International Conference on Acoustics, Speech and Signal Processing*, vol. 4, págs. 3281–3284.

Pearlmutter, B. A. (1995). "Gradient calculations for dynamic recurrent neural networks: a survey", *IEEE Transactions on Neural Networks*, **6**(5), 1212–1228.

Pérez-Ortiz, J. A., J. Calera-Rubio y M. L. Forcada (2001a). "A comparison between recurrent neural architectures for real-time nonlinear prediction of speech signals", en D. J. Miller, T. Adali, J. Larsen, M. Van Hulle y S. Douglas, coordinadores, *Neural Networks for Signal Processing XI, Proceedings of the 2001 IEEE Neural Networks for Signal Processing Workshop*, págs. 73–81, IEEE Signal Processing Society.

PÉREZ-ORTIZ, J. A., J. CALERA-RUBIO y M. L. FORCADA (2001b). "Online symbolic-sequence prediction with discrete-time recurrent neural networks", en G. Dorffner, H. Bischof y K. Hornik, coordinadores, *Proceedings of the International Conference on Artificial Neural Networks*, vol. 2130 de *Lecture Notes in Computer Science*, págs. 719–724, Springer-Verlag, Berlín.

PÉREZ-ORTIZ, J. A., J. CALERA-RUBIO y M. L. FORCADA (2001c). "Online text prediction with recurrent neural networks", *Neural Processing Letters*, **14**(2), 127–140.

PÉREZ-ORTIZ, J. A. y M. L. FORCADA (2001). "Part-of-speech tagging with recurrent neural networks", en *Proceedings of the International Joint Conference on Neural Networks*, págs. 1588–1592.

PÉREZ-ORTIZ, J. A., F. A. GERS, D. ECK y J. SCHMIDHUBER (2002a). "Kalman filters improve LSTM network performance in hard problems", *Neural Networks*, aceptado con modificaciones.

PÉREZ-ORTIZ, J. A., J. SCHMIDHUBER, F. A. GERS y D. ECK (2002b). "Improving long-term online prediction with decoupled extended Kalman filters", en *Proceedings of the International Conference on Artificial Neural Networks*, Lecture Notes in Computer Science, Springer-Verlag, Berlín, aceptado.

PLAUT, D. C., S. J. NOWLAN y G. E. HINTON (1986). "Experiments on learning back propagation", *informe técnico CMU–CS–86–126*, Department of Computer Science, Carnegie–Mellon University.

PRESS, W. H., B. P. FLANNERY, S. A. TEUKOLSKY y W. T. VETTERLING (1988). *Numerical recipes in C*, Cambridge University Press, Cambridge.

PRESS, W. H., S. A. TEUKOLSKY, W. T. VETTERLING y B. P. FLANNERY (1992). *Numerical recipes in C: the art of scientific computing*, Cambridge University Press, 2.ª ed.

PROAKIS, J. y D. MANOLAKIS (1996). *Digital signal processing*, Prentice Hall, 3.ª ed.

PROAKIS, J. G. y D. G. MANOLAKIS (1998). *Tratamiento digital de señales: principios, algoritmos y aplicaciones*, Prentice-Hall.

PUSKORIUS, G. V. y L. A. FELDKAMP (1991). "Decoupled extended Kalman filter training of feedforward layered networks", en *International Joint Conference on Neural Networks*, vol. 1, págs. 771–777.

PUSKORIUS, G. V. y L. A. FELDKAMP (1994). "Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks", *IEEE Transactions on Neural Networks*, **5**(2), 279–297.

RABINER, L. R. (1989). "A tutorial on hidden Markov models and selected applications in speech recognition", *Proceedings of the IEEE*, **77**(2), 257–286.

RIFÀ, J. y LL. HUGUET (1991). *Comunicación digital*, Masson, Barcelona.

ROBINSON, A. J. y F. FALLSIDE (1991). "A recurrent error propagation speech recognition system", *Computer Speech and Language*, **5**, 259–274.

RODRIGUEZ, P. y J. WILES (1998). "Recurrent neural networks can learn to implement symbol-sensitive counting", en *Advances in Neural Information Processing Systems, 10*, págs. 87–93, The MIT Press.

RODRIGUEZ, P., J. WILES y J. ELMAN (1999). "A recurrent neural network that learns to count", *Connection Science*, **11**(1), 5–40.

RUMELHART, DAVID E., GEOFFREY E. HINTON y RONALD J. WILLIAMS (1986). "Learning representations by back-propagating errors", *Nature*, **323**, 533–536.

SAKAKIBARA, Y. (1997). "Recent advances of grammatical inference", *Theoretical Computer Science*, **185**, 15–45.

SCHMID, H. (1994). "Part-of-speech tagging with neural networks", en *Proceedings of the International Conference on Computational Linguistics*, págs. 172–176.

SCHMIDHUBER, J. (1992). "A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running networks", *Neural Computation*, **4**(2), 243–248.

SCHMIDHUBER, J. y S. HOCHREITER (1996). "Guessing can outperform many long time lag algorithms", *informe técnico IDSIA-19-96*, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale.

SCHMIDHUBER, J. y H. STEFAN (1996). "Sequential neural text compression", *IEEE Transactions on Neural Networks*, **7**(1), 142–146.

SEJNOWSKI, T. J. y C. R. ROSENBERG (1987). "Parallel networks that learn to pronounce english text", *Complex Systems*, **1**, 145–168.

SHEPHERD, A. J. (1997). *Second-order methods for neural networks*, Springer, Londres.

SIEGELMANN, H. T. y E. D. SONTAG (1991). "Turing computability with neural nets", *Applied Mathematics Letters*, **4**, 77–80.

SMITH, A. W. y D. ZIPSER (1989). "Learning sequential structures with the real-time recurrent learning algorithm", *International Journal of Neural Systems*, **1**(2), 125–131.

SUN, G. Z., C. LEE GILES, H. H. CHEN y Y. C. LEE (1993). “The neural network pushdown automaton: model, stack and learning simulations”, *informe técnico CS-TR-3118*, University of Maryland, College Park.

TIŇO, P. y M. KÖTELES (1999). “Extracting finite state representations from recurrent neural networks trained on chaotic symbolic sequences”, *IEEE Transactions on Neural Networks*, **10**(2), 284–302.

TIŇO, P., M. STANČÍK y Ľ. BEŇUŠKOVÁ (2000). “Building predictive models on complex symbolic sequences via a first-order recurrent BCM network with lateral inhibition”, en *International Symposium on Computational Intelligence*.

TONKES, B. y J. WILES (1997). “Learning a context-free task with a recurrent neural network: an analysis of stability”, en *Proceedings of the Fourth Biennial Conference of the Australasian Cognitive Science Society*.

TOWNSHEND, B. (1991). “Nonlinear prediction of speech”, en *Proceedings International Conference Acoustics, Speech and Signal Processing*, págs. 425–428.

UNNIKRISHNAN, K. P. y K. P. VENUGOPAL (1994). “Alopex: a correlation-based learning algorithm for feedforward and recurrent neural networks”, *Neural Computation*, **6**, 469–490.

WEIGEND, A. S. y N. A. GERSHENFELD, coordinadores (1994). *Time series prediction: forecasting the future and understanding the past*, Proceedings of the NATO Advanced Research Workshop on Comparative Times, Addison-Wesley.

WELCH, G. y G. BISHOP (2002). “An introduction to the Kalman filter”, *informe técnico TR 95-041*, University of North Carolina at Chapel Hill, Department of Computer Science.

WERBOS, P. J. (1974). *Beyond regression: new tools for prediction and analysis in the behavioral sciences*, tesis doctoral, Harvard University.

WILES, J. y J. ELMAN (1995). “Learning to count without a counter: a case study of dynamics and activation landscapes in recurrent networks”, en *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, págs. 482–487, MIT Press, Cambridge.

WILLIAMS, R. J. y J. PENG (1990). “An efficient gradient-based algorithm for on-line training of recurrent network trajectories”, *Neural Computation*, **2**(4), 490–501.

WILLIAMS, R. J. y D. ZIPSER (1989). “A learning algorithm for continually training recurrent neural networks”, *Neural Computation*, **1**, 270–280.

ZIV, J. y A. LEMPEL (1977). "A universal algorithm for sequential data compression", *IEEE Transactions on Information Theory*, **23**(3), 337–349.