e-Capítulo 7

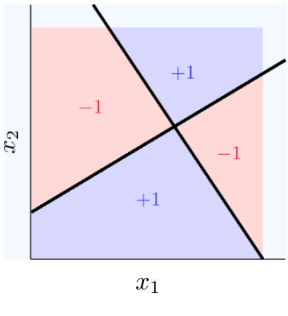
Redes neuronales

Las redes neuronales son un modelo de "inspiración biológica" que ha tenido un éxito considerable en aplicaciones que van desde la predicción de series temporales hasta la visión. Las redes neuronales son una generalización del perceptrón que utiliza una transformación de características que se *aprende a* partir de los datos. Como tales, son un modelo muy potente y flexible.

Las redes neuronales son un buen modelo candidato para aprender de los datos porque pueden aproximar eficazmente funciones objetivo complejas y vienen con buenos algoritmos para ajustar los datos. Comenzaremos con las propiedades básicas de las redes neuronales y cómo entrenarlas a partir de datos. Presentaremos una serie de técnicas útiles para ajustar los datos minimizando el error en la muestra. Dado que las redes neuronales son un modelo muy flexible, con un gran poder de aproximación, es fácil sobreajustar los datos; estudiaremos una serie de técnicas para controlar el sobreajuste específicas de las redes neuronales.

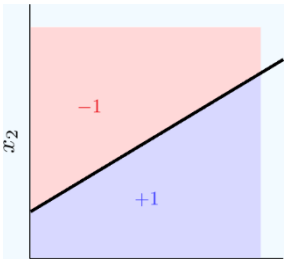
7.1 El perceptrón multicapa (MLP)

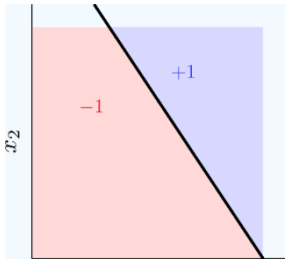
El perceptrón no puede implementar funciones de clasificación sencillas. Para ilustrarlo, utilizamos el tar- get de la derecha, que está relacionado con la función booleana XoR. En este ejemplo, / no puede escribirse como sign(w "x). Sin embargo, / se compone de dos partes lineales. De hecho, como pronto veremos, podemos descomponer J en dos perceptrones simples, correspondientes a las líneas de la figura, y luego combinar las salidas de estos dos perceptrones de forma sencilla para recuperar J.



Los dos perceptrones se muestran a continuación.

La analogía con las neuronas biológicas, aunque inspiradora, no debe llevarse demasiado lejos; al fin y al cabo, construimos aviones con alas que no aletean. Del mismo modo, las redes neuronales, cuando se aplican al aprendizaje a partir de datos, no se parecen demasiado a sus homólogas biológicas.





*1 *1

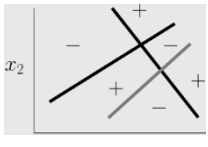
$h_1(\mathbf{x}) = \text{sign}(\mathbf{w}_1^{\text{T}}\mathbf{x})$ hz (x) = signo(z>J

El objetivo J es igual a -1-1 cuando exactamente uno de fit , bz es igual a -1-1. Esta es la función booleana xOR: / - XOR(mosca >z) donde -1-1 representa "VERDADERO" y -1 representa "FALSO". Podemos reescribir J utilizando las operaciones más sencillas OR y AND: OR(fi 2 = +1 si al menos una de fi , fi2 es igual a -I-1 y AND(fit , fi2) = -I-1 si ambas fi , fi2 son iguales a -I-1. Utilizando la notación booleana estándar (multiplicación por
AND, suma para OR y barra para negación),

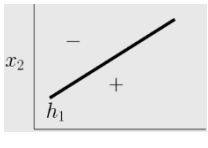
$$f = h_1\overline{h_2} + \overline{h_1}h_2.$$

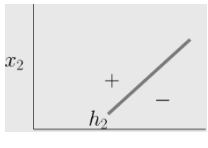
Ejercicio 7.1

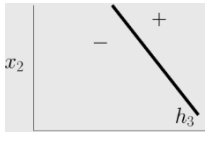
Consideremos una función objetivo / cuyas regiones "+" y "-" se ilustran a continuación.



El objetivo / tiene tres componentes de perceptrón ñ1, ñz, 3 .







Demuestra que

$$f = \overline{h_1}h_2h_3 + h_1\overline{h_2}h_3 + h_1h_2\overline{h_3}.$$

¿Existe una forma sistemática de pasar de un objetivo que es una descomposición de perceptrones a una fórmula booleana como ésta? [*Pista: considere sólo las regiones* de / *que son '-i-' y utilice la* forma normal disyuntiva {o de ANDS).j

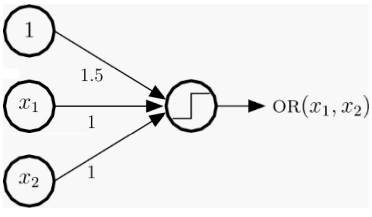
El ejercicio 7.1 muestra que un objetivo complicado, que se compone de percep- trones,

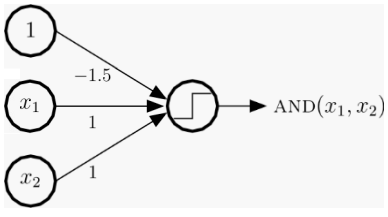
es una disyunción de conjunciones (OR Of ANDs) aplicadas al componente

perceptrones. Se trata de una idea útil, porque el perceptrón puede aplicar oR y AND:

$$\begin{aligned} \text{OR(zt +2)} &= \text{signo(z +2 3)} \\ \text{AND}(x_1, x_2) &= \text{sign}(x_1 + x_2 - 1.5). \end{aligned}$$

Esto implica que estos objetivos más complicados son, en última instancia, meras combinaciones de perceptrones. Para ver cómo combinar los perceptrones para obtener /, introducimos una representación gráfica de perceptrones, empezando por OR y AND:





Un nodo envía un valor a una flecha. El peso de una flecha multiplica esta salida y pasa el resultado al siguiente nodo. Todo lo que llega a este siguiente nodo se suma y luego se transforma mediante sign( ) para obtener la salida final.

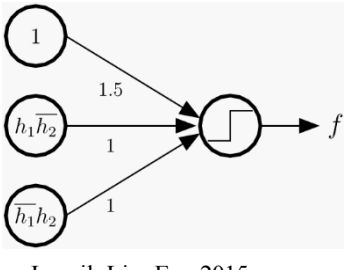
Ejercicio 7.2

(a) Los booleanos or y AND de dos entradas pueden ampliarse a más de dos entradas: OR(zi, zv) -{-1 si alguna de las *M* entradas es ml; AND(+1' ... z ,) ml si todas las entradas son iguales a Al. Dar gráfico representaciones de or(z , ... , zv) y AND(zt , ... , m) .

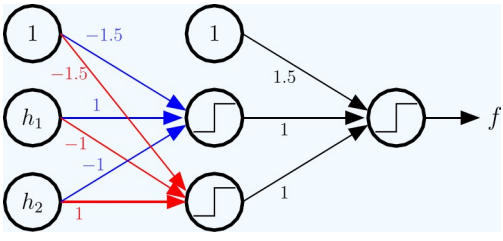
(b) Dar la representación gráfica del perceptrón: //(x) sign(w "x).

(c) Dar la representación gráfica de oR(z , ,).

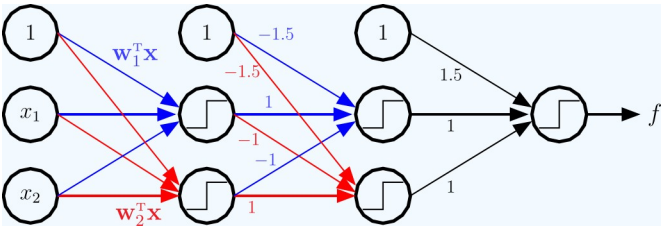
En MLP para un objetivo complejo. Dado que J = bi bz>i >z, que es un OR de las dos entradas fit b2 y fit fi2, utilizamos primero el perceptrón oR, para obtener:



Las dos entradas fit fi2 y fit b2 son AND. Como tales, pueden simularse mediante la salida de dos perceptrones AND. Para tratar la negación de las entradas del AND, negamos los pesos multiplicando las entradas negadas (como se ha hecho en el Ejercicio 7.1(c)). La representación gráfica resultante de J es:



Los pesos azul y rojo simulan los dos AND requeridos. Por último, como fi = sign(w*x) y fi2 = sign(w_2 x) son perceptrones, ampliamos los nodos fi y fi2 para obtener la representación gráfica de /.



El siguiente ejercicio te pide que calcules una fórmula algebraica explícita para J. La representación gráfica visual es mucho más ordenada y fácil de generalizar.

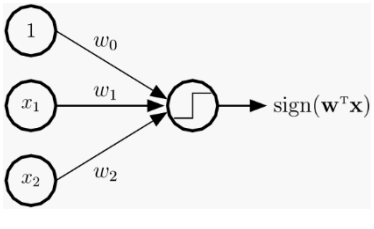
Ejercicio 7.3

Utiliza la representación gráfica para obtener una fórmula explícita para / y demuéstralo:

`/(x) = sign sign(#1(x) # z (x) ₂) sign(#1(x) #z(x)+₂ )+ j),`

donde ñi (x)sign (w1x) y hz(x) sign(z -)

Comparemos la forma gráfica de J con la forma gráfica del perceptrón simple, que se muestra a la derecha. Se utilizan más capas de nodos entre la entrada y la salida para implementar /, en comparación con el perceptrón simple, por lo que lo llamamos perceptrón multicapa (MLP). Las capas adicionales



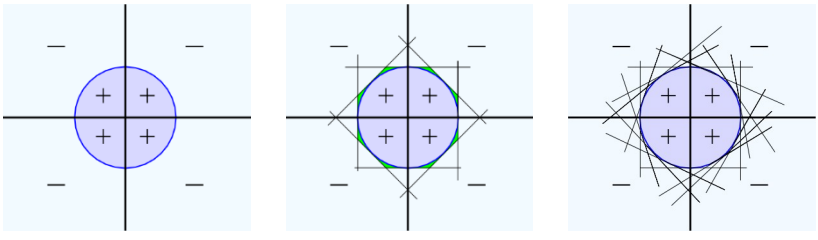
se denominan *capas ocultas*.

Observe que las capas sólo alimentan hacia adelante a la capa siguiente (no hay flechas que apunten hacia atrás ni saltos a otras capas). La capa de entrada (más a la izquierda) no se cuenta como una capa, por lo que en este ejemplo hay 3 capas (2 capas ocultas con 3 nodos cada una y una capa de salida con 1 nodo). El perceptrón simple no tiene capas ocultas, sólo entrada y salida. La adición de capas ocultas es lo que nos permitió implementar el objetivo más complicado.

Ejercicio 7.4

Para la función objetivo del Ejercicio 7.1, dé el MLP en forma gráfica, así como la forma algebraica explícita.

Si J puede descomponerse en perceptrones utilizando un oR de ANDS, entonces puede implementarse mediante un perceptrón de 3 capas. Si J no es estrictamente descomponible en perceptrones, pero la frontera de decisión es suave, entonces un perceptrón de 3 capas puede acercarse arbitrariamente a la implementación de J. A continuación se presenta una ilustración de "prueba por imágenes" para una función objetivo de disco:



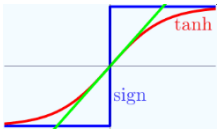
Objetivo 8 perceptrones 16 perceptrones

La demostración formal es análoga al teorema del cálculo según el cual cualquier función continua en un conjunto compacto puede aproximarse de forma arbitraria mediante funciones escalonadas. El perceptrón es el análogo de la función escalón. Así pues, hemos encontrado una generalización del perceptrón simple que se parece mucho al propio perceptrón simple, salvo por la adición de más capas. Hemos adquirido la capacidad de modelar funciones objetivo más complejas añadiendo más nodos *(*unidades *ocultas)* en las capas ocultas esto corresponde a permitir más perceptrones en la descomposición de J. De hecho, un MLP de 3 capas adecuadamente grande puede aproximar casi cualquier función objetivo y ajustarse a cualquier conjunto de datos, por lo que es un modelo de aprendizaje muy potente. Utilícelo con cuidado. Si su MLP es demasiado grande puede perder capacidad de generalización.

Una vez fijado el tamaño del MLP (número de capas ocultas y número de unidades ocultas en cada capa), se aprenden los pesos de cada enlace (flecha) ajustando los datos. Consideremos el perceptrón simple,

// (Xt - @ (W "X) .

Cuando 6(s) = sign(s), aprender los pesos ya era un problema combinatorio difícil y contaba con una variedad de algoritmos, incluido el algoritmo de bolsillo, para ajustar los datos (Capítulo 3). El problema de optimización combinatoria es aún más difícil con el MLP, por la misma razón, a saber, que la función sign( ) no es suave; una aproximación suave y diferenciable a sign( ) nos permitirá utilizar métodos analíticos, en lugar de métodos puramente combinatorios, para encontrar los pesos óptimos. Por lo tanto, aproximamos o "suavizamos" la función sign( ) utilizando la función tanh( ). La MLP se denomina a veces red neuronal de umbral (duro) porque la función de transformación es un umbral duro en cero. Aquí, elegimos 8(z) = tanh(z) que es in-

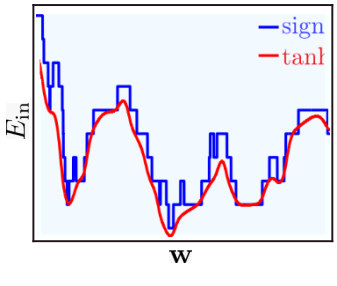


entre lineal y el umbral duro: casi lineal para z 0 y casi -1-1 para |z | grande. La función tanh( ) es otro ejemplo de *sigmoide* (porque su forma se parece a una "s" aplanada), relacionada con la sigmoide que utilizamos para la regresión logística.[2] Estas redes se denominan redes neu- rales sigmoidales. Del mismo modo que podríamos utilizar los pesos aprendidas de la regresión lineal para la clasificación, podríamos utilizar pesos aprendidos utilizando la red neuronal sigmoidal con función de activación tanh( ) para la clasificación sustituyendo la función de activación de salida por sign( ).

Ejercicio 7.5

Dados w1 y > 0, hallar wz tal que |sign(w1xn)- tanh(2 -) þara xn E @.
[*Pista: Para* /arge *enoL/gh* a, sign (z) tanh(oz) ./

El ejemplo anterior ejemplo anterior muestra que la función sign( ) puede aproximarse mucho a la función tanh( ). Una ilustración concreta de esto se muestra en la figura de la derecha. La figura muestra cómo varía el error en la muestra A,q con uno de los pesos en w en un problema de ejemplo para el perceptrón (curva azul) en comparación con la versión sigmoidal (curva roja). La aproximación sigmoidal captura el gen-

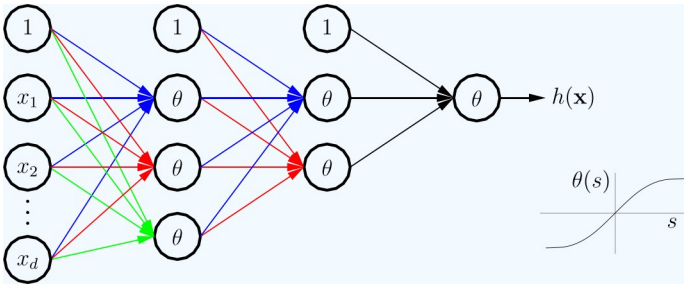


eral forma del error, de modo que si minimizamos el error sigmoidal dentro de la muestra, obtenemos una buena aproximación a la minimización del error de clasificación dentro de la muestra.

2En la regresión logística, utilizamos la sigmoidea porque queríamos una probabilidad como salida. Aquí, utilizamos el tanh(-) 'suave' porque queremos una función objetivo amigable para optimizar.

7.2 Redes neuronales

La red neuronal es nuestro MLP "suavizado". Comencemos con una representación gráfica de una *red neuronal de avance* (el único tipo que consideraremos).



capa de entrada = 0 hiddei|ayei 0 < f<L salida |a) erf=L

La representación gráfica muestra una función en nuestro conjunto de hipótesis. Aunque esta visión gráfica es estética e intuitiva, con la información "fluyendo" desde las entradas en el extremo izquierdo, a lo largo de los enlaces y a través de los nodos ocultos, en última instancia a la salida fi(x) en el extremo derecho, será necesario describir algorítmicamente la función que se está calculando. Las cosas se van a complicar y esto requiere una notación muy sistemática.

7.2.1 Notación

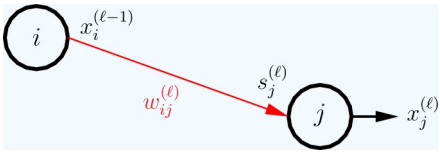
Hay capas etiquetadas por fi = 0, 1, 2, ... En nuestro ejemplo anterior, £ = 3, es decir, tenemos tres capas (la capa de entrada é - 0 no suele considerarse una capa y sirve para alimentar las entradas). La capa é = £ es la capa de salida, que determina el valor de la función. Las capas intermedias, 0 é £ , son las capas ocultas. Utilizaremos el superíndice " para referirnos a una capa concreta. Cada capa é tiene una "dimensión" *d!"* lo que significa que tiene d *!')* -I- 1 nodos, etiquetados 0, 1, ... , d!" . Cada capa tiene un nodo especial, que se llama nodo él (etiquetado 0). Este nodo de sesgo está configurado para tener una salida 1, que es análoga a la convención ficticia z9 = 1 que teníamos para los modelos lineales.

Cada flecha representa un peso o fuerza de conexión de un nodo en una capa a un nodo en la capa nez/ fiipfier. Nótese que los nodos de sesgo no tienen pesos entrantes. No hay otros pesos de conexión". Un nodo con un peso entrante indica que alguna señal se introduce en este nodo. Cada nodo de este tipo con una entrada tiene un trons/ormntion /unc/ion 8. Si 6(s) = señal(s), entonces tenemos el MLP para la clasificación. Como hemos mencionado antes, utilizaremos una versión suave del MLP con B(z) - tanh(z) para aproximar la función sign( ). El tanh( ) es un umbral suave o sigmoide, y ya vimos un sigmoide relacionado

*En una configuración más general, los pesos pueden conectar dos nodos cualesquiera, además de ir hacia atrás (es decir, puede haber ciclos). Este tipo de redes se denominan redes neuronales recurrentes, y no las consideramos aquí.

cuando hablamos de la regresión logística en el capítulo 3. En última instancia, cuando hacemos clasificación, sustituimos la sigmoide de salida por el umbral duro sign( ). Como comentario, si estuviéramos haciendo regresión en su lugar, toda nuestra discusión pasa con la transformación de salida que se sustituye por la función de identidad (sin transformación) para que la salida sea un número real. Si estuviéramos haciendo regresión logística, reemplazaríamos el sigmoide de salida tanh( ) por el sigmoide de regresión logística.

El modelo de red neuronal Hpq se especifica una vez que se determina el ar- *chitectura* de la red neuronal, es decir, la dimensión de cada capa d *d!'), d('), ... ¡, d!)* (ñes el número de capas). Una hipótesis finn se especifica seleccionando pesos para los enlaces. Acerquémonos a un nodo de la capa oculta capa I, para ver qué pesos hay que especificar.



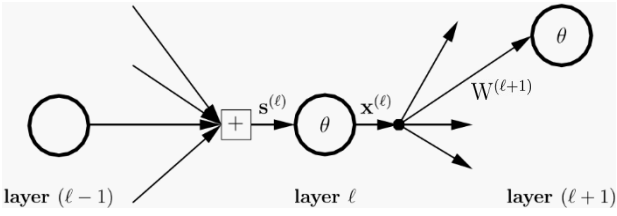
capa (I 1) capa I

Un nodo tiene una señal de entrada *s* y una salida z. Los pesos de los enlaces al nodo desde la capa anterior están en '), por lo que los pesos están indexados por la capa a la que van. Así, la salida de los nodos de la capa - 1 se multiplica por los pesos '). Utilizamos subíndices para indexar los nodos de una capa.

Así, es el peso que entra en el nodo j de la capa é *desde* el nodo i de la capa anterior, la señal que entra en el nodo ¡ de la capa é es s ", y la salida de este nodo

es z ". Hay algunos nodos especiales en la red. Los nodos cero de cada capa son nodos constantes, con salida 1. No tienen peso entrante, pero sí saliente. Los nodos de la capa de entrada é = 0 son para los valores de entrada, y no tienen peso entrante ni función de transformación.

En la mayoría de los casos, sólo necesitamos tratar la red capa por capa, por lo que introducimos la notación vectorial y matricial para ello. Recogemos todas las señales de entrada a los nodos 1, ... , *d"'* en la capa é en el vector s '). Del mismo modo, recogemos la salida de los nodos 0, … , d" en el vector x" ; nótese que x" C (1} xi^d "' debido al nodo de polarización 0. Hay enlaces que conectan las salidas de todos los nodos de la capa anterior con las entradas de la capa é . Así, en la capa é, tenemos a (d " -I- 1) x d" matriz de pesos W '). La entrada (i, j) de W" va del nodo i de la capa anterior al nodo j de la capa é .



parámetros layer é

señales in	$\mathbf{s}^{(\ell)}$	$d^{(\ell)}$ vector de entrada dimensional
salidas	$\mathbf{x}^{(\ell)}$	$d^{(\ell)}+1$ vector de salida unidimensional
pesos de interconexiones in	$\mathrm{W}^{(\ell)}$ (W ')	$(d^{(\ell-1)}+1) \times d^{(\ell)}$ matriz dimensional
pesos de interconexiones out	$\mathrm{W}^{(\ell+1)}$ (W '+')	$(d^{(\ell)}+1) \times d^{(\ell+1)}$ matriz dimensional

Después de fijar los pesos W ') para é = 1, ... , ñ, habrá especificado una hipótesis de red neuronal concreta fi C Hqp. Recopilamos todas estas matrices de pesos en un único parámetro de peso w (W '), W^2), ... , W *)}, y a veces escribiremos fi(x; w) para indicar explícitamente la dependencia de la hipótesis respecto a w.

7.2.2 Propagación hacia delante

La hipótesis fi(x) de la red neuronal se calcula mediante el algoritmo *de propagación hacia delante*. En primer lugar, observe que las entradas y salidas de una capa están relacionadas por la función de transformación,

$$\mathbf{x}^{(\ell)} \qquad \begin{matrix} 1 \\ \theta(\mathbf{s}^{(\ell)}) \end{matrix} . \tag{7.1}$$

donde 8(s''') es un vector cuyos componentes son 6(s"). Para obtener el vector de entrada en la capa é, calculamos la suma ponderada de las salidas de la capa anterior

capa, con pesos especificados en W ') : s " $_p$ "*q*" en 'z " . Este proceso se representa de forma compacta mediante la ecuación matricial

$$\mathbf{s}^{(\ell)} = (\mathrm{W}^{(\ell)})^{\mathrm{T}} \mathbf{x}^{(\ell-1)}. \tag{7.2}$$

Todo lo que queda es inicializar la capa de entrada a x" = x (por lo que *d!"* = *d,* la dimensión de entrada)[4] y utilizar las ecuaciones (7.2) y (7.1) en la siguiente cadena,

$$\mathbf{x} = \mathbf{x}^{(0)} \xrightarrow{\mathrm{W}^{(1)}} \mathbf{s}^{(1)} \xrightarrow{\theta} \mathbf{x}^{(1)} \xrightarrow{\mathrm{W}^{(2)}} \mathbf{s}^{(2)} \xrightarrow{\theta} \mathbf{x}^{(2)} \cdots \longrightarrow \mathbf{s}^{(L)} \xrightarrow{\theta} \mathbf{x}^{(L)} = h(\mathbf{x}).$$

'Recordemos que los vectores de entrada también se aumentan con z = 1.

Propagación hacia delante para calcular fi(x):

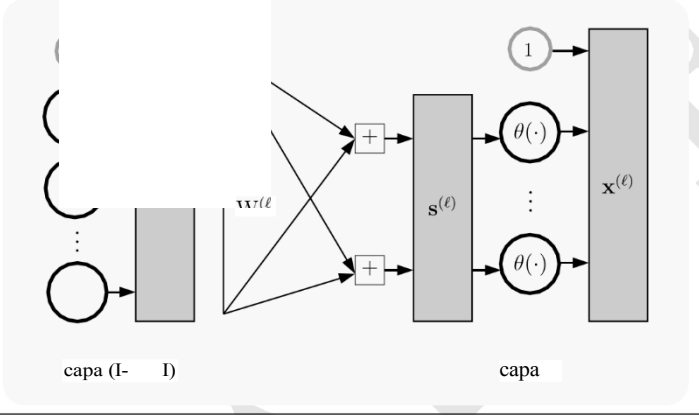
1: x " +- x [Inicialización]

2. for é - 1 to £ do [Propagación hacia delante]

3: $\mathbf{s}^{(\ell)} \leftarrow (\mathrm{W}^{(\ell)})^{\mathrm{T}}\mathbf{x}^{(\ell-1)}$

4:

5: $h($ [Salida]



Tras la propagación hacia delante, el vector de salida x ') en cada capa / = 0, ... , *L* se ha calculado.

Ejercicio 7.6

Sean V y @ el número de nodos y pesos de la red neuronal,

$$d''', \qquad @ - \qquad d''' \quad (d'' \quad ^{1} + 1).$$
$$\ell=0 \qquad\qquad\qquad \ell=1$$

En términosde y @,cuántos cálculos se realizan en la propa- gación hacia delante (sumas, multiplicaciones y evaluaciones de 9).

{Respuesta. 0(@) multiplicaciones y sumas, y 0(V) 8-evaluaciones).

Si queremos calcular Aip , sólo necesitamos fi(xq) y pq. Para la suma de cuadrados,

$$E_{\mathrm{in}}(\mathbf{w}) \qquad \frac{1}{N}\sum_{}^{N}(h(\mathbf{x}_n;\mathbf{w}) - y_n)^2$$

$$\frac{1}{N}\sum_{}^{N}(\mathbf{x}^{(L)} - y_n)^2.$$

Ahora discutiremos cómo minimizar €ip para obtener los pesos aprendidos. Será una aplicación directa del descenso de gradiente, con un algoritmo especial que calcula el gradiente eficientemente.

7.2.3 Algoritmo de retropropagación

En el capítulo 3 estudiamos un algoritmo para llegar a un mínimo local de una superficie de error suave dentro de la muestra, a saber, el descenso de gradiente: inicializar los pesos a w(0) y para t = 1, 2, ... actualizar los pesos dando un paso en la dirección del gradiente negativo,

lo llamamos (por lotes) descenso *por gradiente.* Para implementar el descenso por gradiente, necesitamos el gradiente.

Ejercicio 7.7

Para el perceptrón sigmoidal, /r(x) tanh(w "x) , dejemos que el error en la muestra be $E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum$ (tanh(w "x) //)² Demuestre que

$$\nabla(w) = - \sum_{n=1} (\tanh(w'x\ )\ q\ )(1 - \tanh^2(w'x\ ))x\ .$$

Si w ,qué ocurre con el gradiente; cómo se relaciona esto con por qué es difícil optimizar el perceptrón.

Consideremos ahora la red neuronal multicapa sigmoidal con 6(z) = tanh(z). Como fi(x) es suave, podemos aplicar el descenso de gradiente a la función de error resultante. Para ello, necesitamos el gradiente WAiq (w). Recordemos que el vector de pesos w contiene todas las matrices de pesos W'*), ... W'", y necesitamos las derivadas con respecto a todos estos pesos. A diferencia del perceptrón sigmoidal del Ejercicio 7.7, para la red sigmoidal multicapa no existe una expresión simple de forma cerrada para el gradiente. Consideremos un error en la muestra que es la suma de los errores puntuales sobre los puntos de datos (como lo es el error en la muestra al cuadrado),

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum^{N} \text{en.}$$

donde eg = e(fi(xp), pp). Para el error al cuadrado, e (fi, 9) (fi - p)°. Para calcular el gradiente de A¡p , necesitamos su derivada con respecto a cada matriz de pesos:

$$\frac{\partial E_{\text{in}}}{\partial \mathrm{W}^{(\ell)}} = \frac{1}{N}\sum_{n}^{N} \frac{\partial \mathrm{e}_n}{\partial \mathrm{W}^{(\ell)}} \qquad (7.3)$$

El elemento básico de (7.3) es la derivada parcial del error en a punto de datos e, con respecto al W '). Una forma rápida y sucia de obtener $\frac{\partial \mathrm{e}}{\partial \mathrm{W}^{(\ell)}}$ es

utilizar el enfoque numérico de diferencias finitas. La complejidad de obtener las derivadas parciales con respecto a cada peso es O(Q²), donde Q es el número de pesos (véase el problema 7.6). A partir de (7.3), tenemos que calcular estas derivadas para cada punto de datos, por lo que el enfoque numérico es prohibitivo desde el punto de vista computacional. Ahora derivamos un elegante algoritmo de programación dinámica conocido como *retropropagación. La retropropagación* nos permite calcular las derivadas parciales con respecto a cada peso de manera eficiente, utilizando O(Q) computación. Describimos la retropropagación para obtener la derivada parcial del error e, pero el algoritmo es general y se puede utilizar para obtener la derivada parcial de cualquier función de la salida fi(x) con respecto a los pesos.

La retropropagación se basa en varias aplicaciones de la regla de la cadena para escribir derivadas parciales en la capa é utilizando derivadas parciales en la capa (é -I- 1). Para describir el algoritmo, definimos el *vector de* sensibilidad para la capa é, que es la sensibilidad (gradiente) del error e con respecto a la señal de entrada s ') que entra en la capa é. Denotamos la sensibilidad por d" ,

$$\boldsymbol{\delta}^{(\ell)} \quad \frac{\text{'9e}}{\partial \mathbf{s}^{(\ell)}}.$$

La sensibilidad cuantifica cómo cambia e con s('). Utilizando la sensibilidad, podemos escribir las derivadas parciales con respecto a los pesos W(') como

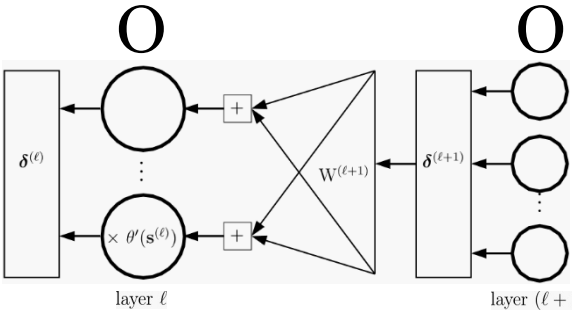
$$\frac{\partial \text{e}}{\partial \text{W}^{(\ell)}} = \mathbf{x}^{(\ell-1)} (\boldsymbol{\delta}^{(\ell)})^{\text{T}}. \qquad (74)$$

Derivaremos esta fórmula más adelante, pero por ahora examinémosla de cerca. Las derivadas parciales de la izquierda forman una matriz de dimensiones (d '**) -I- 1) d ') y el "producto exterior" de los dos vectores de la derecha da exactamente una ma- triz. Las derivadas parciales tienen contribuciones de dos componentes. (i) El vector de salida de la capa de la que proceden los pesos; cuanto mayor sea la salida, más sensible será e a los pesos de la capa. (ii) El vector de sensibilidad de la capa a la que van los pesos; cuanto mayor sea el vector de sensibilidad, más sensible será e a los pesos de esa capa.

Las salidas x ') para cada capa é 1 0 pueden calcularse mediante una propagación hacia delante. Por tanto, para obtener las derivadas parciales, basta con obtener los vectores de sen- sibilidad d ') para cada capa é 1 (recuerde que no hay señal de entrada a la capa I - 0). Resulta que los vectores de sensibilidad pueden obtenerse ejecutando una versión ligeramente modificada de la red neuronal *hacia atrás,* y de ahí el nombre de retropropagación. En la propagación hacia delante, cada capa produce el vector x(') y en la retropropagación, cada capa produce (hacia atrás) el vector d('). En la propagación hacia delante, calculamos x(') a partir de x('**) y en la propagación hacia atrás, calculamos d ') a partir de d '+'). La idea básica se ilustra en la siguiente figura.

La programación dinámica es una elegante técnica algorítmica en la que se construye una

solución a un problema complejo utilizando las soluciones a problemas relacionados pero más sencillos.



Como se puede ver en la figura, la red neuronal está ligeramente modificada sólo en que hemos cambiado la función de transformación de los nodos. En la propagación hacia delante, la transformación era la sigmoidea 8( ). En la retropropagación, la transformación es la *multiplicación por 8*' (s") , donde s ') es la entrada al nodo. Así que la función de transformación es ahora diferente para cada nodo, y depende de la entrada al nodo, que depende de x. Esta entrada se calculó en la propagación hacia adelante. Para la función de transformación tanh( ), tanh' (s"') = 1 - tanh^2 (s ')) = 1 - x ') x"', donde significa multiplicación por componentes.

En la figura, la capa (é+ 1) produce (hacia atrás) el vector de sensibilidad d '+', que se multiplica por los pesos W '+', se suma y se pasa a los nodos de la capa é. Los nodos de la capa fi se multiplican por B' (s ')) para obtener d ". Usando &, una notación abreviada para este paso de retropropagación es:

$$\boldsymbol{\delta}^{(\ell)} = \theta'(\mathbf{s}^{(\ell)}) \otimes \left[\mathrm{W}^{(\ell+1)}\boldsymbol{\delta}^{(\ell+1)}\right]_1^{d^{(\ell)}}, \tag{7.5}$$

donde el vector W '+')d '+') contiene los componentes 1, ... d!') del vector W('+*'d "+*) (excluyendo el componente de sesgo que tiene índice 0). Esta fórmula no es sorprendente. La sensibilidad de e a las entradas de la capa é es proporcional
a la pendiente de la función de activación en la capa é (una mayor pendiente significa una menor
un cambio en s" tendrá un mayor efecto sobre x"), el tamaño de los pesos que salen de la capa (pesos más grandes significan que un pequeño cambio en s" tendrá más impacto en s '+')) y la sensibilidad en la capa siguiente (un cambio en la capa é afecta a las entradas de la capa é + 1, por lo que si e es más sensible a la capa é + 1, entonces también será más sensible a la capa é).

Derivaremos esta recursión hacia atrás más adelante. Por ahora, observe que si sabemos d '+'), entonces se puede obtener d '). Utilizamos d *) para sembrar el

proceso hacia atrás, y podemos obtener que explícitamente porque e $(x'' - p)^2$ $(B(s^*)) - p)^2$.

Por lo tanto,

$$\begin{aligned}\boldsymbol{\delta}^{(L)} &= \frac{\partial \mathrm{e}}{\partial \mathbf{s}^{(L)}} \\ &= \frac{\partial}{\partial \mathbf{s}^{(L)}} (\mathbf{x}^{(L)} - y)^2 \\ &= 2(\mathrm{x}\ ''\quad \mathrm{p}) \frac{\mathbf{x}^{(L)}}{\mathbf{s}^{(L)}} \\ &= 2(\mathbf{x}^{(L)} - y)\theta'(\mathbf{s}^{(L)}).\end{aligned}$$

Cuando la transformación de salida es tanh( ), B'(s *)) - 1 (z *))^2 (clasifica- ción) ; cuando la transformación de salida es la identidad (regresión), 8' (s *)) - 1. Ahora, utilizando (7.5), podemos calcular todas las sensibilidades:

$$\boldsymbol{\delta}^{(1)} \longleftarrow \boldsymbol{\delta}^{(2)} \cdots \longleftarrow \boldsymbol{\delta}^{(L-1)} \longleftarrow \boldsymbol{\delta}^{(L)}.$$

Tenga en cuenta que como sólo hay un nodo de salida, s* es un escalar, y también lo es d '). El siguiente cuadro de algoritmos resume la retropropagación.

Retropropagación para calcular sensibilidades d ').
Entrada: un punto de datos (x, p).

0: Ejecutar propagación hacia adelante en x para calcular y guardar:

s(') para I = 1, ... , ñ;
x ') para I = 0, ... , ñ.

' '' 2 (' ''– ') a ('' ') [Inicialización]

$$\theta'(s^{(L)}) = \begin{cases} 1 - (x^{(L)})^2 & \theta(s) = \tanh(s); \\ 1 & \theta(s) = s. \end{cases}$$

2: para é = ñ - 1 a 1 hacer [Back-Propagation]

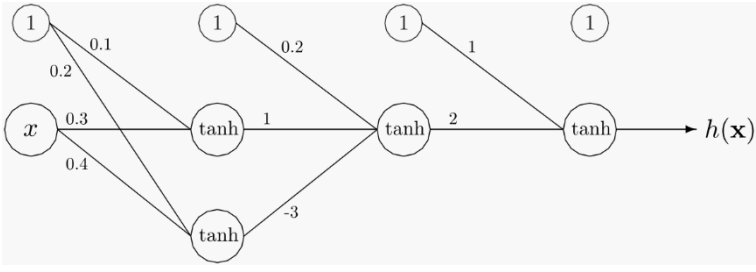
3: Let $\theta'(\mathbf{s}^{(\ell)}) = \left[1 - \mathbf{x}^{(\ell)} \otimes \mathbf{x}^{(\ell)}\right]_1^{d^{(\ell)}}$.

4: Calcular la sensibilidad d ') a partir de d '+'):

$$\boldsymbol{\delta}^{(\ell)} \leftarrow \theta'(\mathbf{s}^{(\ell)}) \otimes \left[\mathrm{W}^{(\ell+1)} \boldsymbol{\delta}^{(\ell+1)}\right]_1^{d^{(\ell)}}$$

En el paso 3 , asumimos transformaciones de nodo oculto tanh. Si las funciones de transformación de la unidad oculta no son tanh( ), la derivada del paso 3 debe actualizarse en consecuencia. Usando la propagación hacia adelante, calculamos x(') para I = 0, ... , ñ y utilizando la retropropagación, calculamos d ') para I = 1, ... , ñ. Por último, obtenemos la derivada parcial del error en un punto de datos simple mediante la ecuación (7.4). Nada ilumina mejor las partes móviles que trabajar un ejemplo de principio a fin.

Ejemplo 7.1. Considere la siguiente red neuronal.



Hay una sola entrada, y las matrices de pesos son:

$$W^{(1)} \quad \begin{matrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{matrix} \quad p(z)'' \quad \begin{matrix} 0.2 \\ 1 \\ -3 \end{matrix} \quad p() \quad \begin{matrix} 1 \\ 2 \end{matrix}$$

Para el punto de datos z = 2, p = 1, la propagación hacia delante da:

$x^{(0)}$	$s^{(1)}$	$x^{(1)}$	$s^{(2)}$	$x^{(2)}$	$s^{(3)}$	$x^{(3)}$
$\begin{bmatrix}1\\2\end{bmatrix}$	$\begin{bmatrix}0.1 & 0.3\\0.2 & 0.4\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}0.7\\1\end{bmatrix}$	$\begin{bmatrix}1\\0.60\\0.76\end{bmatrix}$	[-1.18]	$\begin{bmatrix}1\\-0.90\end{bmatrix}$	[-0.8]	-0.66

Más arriba mostramos cómo se calcula s ') (W ')) "x '). La retropropagación da

$\delta^{(3)}$	$\delta^{(2)}$	$\delta^{(1)}$
$[-1.855]$	$[(1-0.9^2)\cdot 2\cdot -1.855] = [-0.69]$	$\begin{bmatrix}-0.41\\0.88\end{bmatrix}$

Hemos mostrado explícitamente cómo se obtiene d²) a partir de d °). Ahora es sencillo combinar los vectores de salida x ') con los vectores de sensibilidad d ') utilizando (7.4) para obtener las derivadas parciales que se necesitan para el gradiente:

$$\frac{de}{d\backslash V(')} \quad {}^{(0)}(\delta^{(1)})^{T} = \begin{matrix} -044 & 088 \\ -088 & 175 \end{matrix} \quad ' \quad \frac{de}{dM2)} \quad \begin{matrix} -0.69 \\ -0.42 \\ -0.53 \end{matrix} \quad \frac{ser}{\hat{a}\backslash'/()} \quad \begin{matrix} -1.85 \\ 1.67 \end{matrix}$$

Ejercicio 7.8

Repita los cálculos del ejemplo 7.1 para el caso en que la trans- formación de salida sea la identidad. Debe calcular s('), x('), d(' y de/dW('

Derivemos (7.4) y (7.5), que son las ecuaciones centrales de la retropropagación. No hay nada más que la aplicación repetida de la regla de la cadena. Si quieres confiar en nuestras matemáticas, no te perderás gran cosa por seguir adelante.

> Comienza el salto seguro: Si confías en nuestras matemáticas, puedes saltarte esta parte sin comprometer la secuencia lógica. Un recuadro verde similar te indicará cuándo volver a unirte.

Para empezar, veamos más de cerca la derivada parcial, fie/fiW '). La situación se ilustra en la figura 7.1.

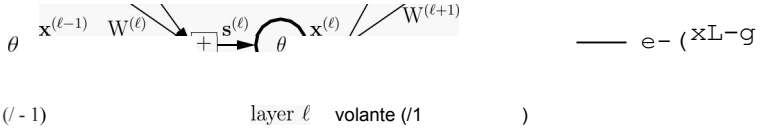


Figura 7.1: Cadena de dependencias de W(' a x) .

Podemos identificar la siguiente cadena de dependencias por las que W ') influye en la salida x '), y por tanto en el error e.

$$\mathrm{W}^{(\ell)} \longrightarrow \mathbf{s}^{(\ell)} \longrightarrow \mathbf{x}^{(\ell)} \longrightarrow \mathbf{s}^{(\ell+1)} \cdots \longrightarrow \mathbf{x}^{(L)} = h.$$

Para derivar (7.4), nos limitamos a un único peso y utilizamos la regla de la cadena. Para un solo peso en un cambio en in ' sólo afecta a s " y así por la regla de la cadena,

$$\frac{\partial \mathrm{e}}{\partial w_{ij}^{(\ell)}} \quad \frac{\partial \mathbf{s}_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} \quad \frac{\partial \mathrm{e}}{\partial \mathbf{s}_j^{(\ell)}} = \mathbf{x}_i^{(\ell-1)} \cdot \boldsymbol{\delta}_j^{(\ell)},$$

donde la última igualdad se deduce porque s" $'_{-}{}'q''\ w_{\alpha j}^{(\ell)} \mathbf{x}_\alpha^{(\ell-1,}$ y por definición de dq". Hemos derivado la forma componente de (7.4).

Derivamos ahora la forma componente de (7.5). Como e depende de s(') sólo a través de x(') (véase la figura 7.1), por la regla de la cadena, tenemos:

$$\boldsymbol{\delta}_j^{(\ell)} = \frac{\mathrm{de}}{\partial \mathbf{s}_j^{(\ell)}} \qquad \frac{\mathrm{d\ e}}{\partial \mathbf{x}_j^{(\ell)}} \frac{\partial \mathbf{x}_j^{(\ell)}}{\partial \mathbf{s}_j^{(\ell)}} = 8\,' \left(\mathbf{s}_j^{(\ell)}\right) \cdot \frac{\partial \mathrm{e}}{\partial \mathbf{x}_j^{(\ell)}}$$

Para obtener la derivada parcial fie/fix '), necesitamos entender cómo cambia e debido a cambios en x '). De nuevo, a partir de la figura 7.1, un cambio en x ') sólo afecta a

s('+*) y, por tanto, e. Dado que un componente concreto de x(') puede afectar a todos los componentes de s '+'), tenemos que sumar estas dependencias utilizando la regla de la cadena:

$$\frac{\text{de}}{\text{g ')}} \qquad \sum_{jk-!}^{d^{(\ell+1)}} \frac{\partial \mathbf{s}_k^{(\ell+1)}}{\partial \mathbf{x}_j^{(\ell)}} \cdot \frac{\partial \text{e}}{\partial \mathbf{s}_k^{(\ell+1)}} \text{-} \quad \sum_{jk-!}^{d^{(\ell+1)}} w_{jk}^{(\ell+1)} \boldsymbol{\delta}_k^{(\ell+1)}.$$

Juntando todo esto, hemos llegado a la versión componente de (7.5)

$$\boldsymbol{\delta}_j^{(\ell)} = \theta'(\mathbf{s}_j^{(\ell)}) \sum^{d^{(\ell+1)}} w_{jk}^{(\ell+1)} \boldsymbol{\delta}_k^{(\ell+1)}, \tag{7.6}$$

Intuitivamente, el primer término proviene del impacto de s ') sobre x ') ; el sumatorio es el impacto de x ') sobre s '+'), y el impacto de s '+') sobre fi es lo que nos devuelve las sensibilidades en la capa (I -I- 1), dando lugar a la recursión hacia atrás.

Fin del salto seguro: Los que saltaron se reúnen ahora con nosotros para discutir cómo la retropropagación nos da Wfiiq.

La retropropagación funciona con un punto de datos (x, p) y pesos w (W '), ... , W *)}. Puesto que ejecutamos una propagación hacia delante y hacia atrás para calcular las salidas x"' y las sensibilidades d"' , el tiempo de ejecución es del orden del número de pesos de la red. Calculamos una vez para cada punto de datos (xp, pq) para obtener WA,q(xp) y, utilizando la suma en (7.3), agregamos estos gradientes de punto único para obtener el gradiente ho/cfi completo WAip. A continuación resumimos el algoritmo.

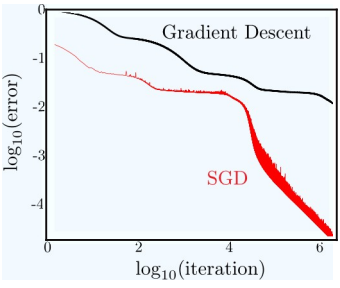
Algoritmo para calcular A¡q (w) y g WA¡q (w).
Entrada: w (W'*), ... , W *)}; P (x , 9) ... (xy , pq).
Salida: errorin ( ) y gradiente g (G ", ... , G " }.

1: Inicializar: Aip = 0 y G ') = 0 W ') para é = 1, ... , L 2: Para cada punto de datos (xp , pp), n - 1, ... , N, do
3: Calcular x" para fi - 0, ... , *L.* propagación hacia
4: Calcula d ') para fi - *L,* ... , 1. delante
5: #;n #;n -/- (x"))2 retropropagaciónJ
s. para I =¯1, ... , ñ do
7: $G^{(\ell)}(\mathbf{x}_n) = [\mathbf{x}^{(\ell-1)}(\boldsymbol{\delta}^{(\ell)})^{\text{T}}]$
8: yC) GC) y GJ(x,)

(G ') (xp) es el gradiente en el punto de datos xp). La actualización del peso para una sola iteración del descenso de gradiente de tasa de aprendizaje fija es W" +- W"

- VG ", para é = 1, ... , L. Hacemos todo esto para una iteración del descenso de gradiente, un cálculo costoso para sólo un pequeño paso. , *L*. Hacemos todo esto para una iteración del descenso gradiente, un cálculo costoso para sólo un pequeño paso.

En el Capítulo 3, discutimos el *descenso de gradiente* s/ocfios- lie *(S 'D)* como una alternativa más eficiente al modo por lotes. En lugar de esperar al gradiente agregado G ') al final de la iteración, se actualizan inmediatamente los pesos a medida que se procesa secuencialmente cada punto de datos utilizando el gradiente de punto único en el paso 7 del algoritmo: W" W" VG" (xp). En esta versión secuencial, todavía se ejecuta una propagación hacia delante y hacia atrás para cada punto de datos.



punto de datos, pero realiza *N* actualizaciones de los pesos. A la derecha se muestra una comparación del descenso gradiente por lotes con SGD. Utilizamos 500 ejemplos de entrenamiento de los datos de dígitos y una red neuronal de 2 capas con 5 unidades ocultas y una tasa de aprendizaje p = 0,01. La curva SGD es errática porque no se está minimizando el error total en cada punto de datos, sino que se realizan N actualizaciones de los pesos. La curva SGD es errática porque no se minimiza el error total en cada iteración, sino el error en un punto de datos específico. Un método para amortiguar este comportamiento errático consiste en disminuir la tasa de aprendizaje a medida que avanza la minimización.

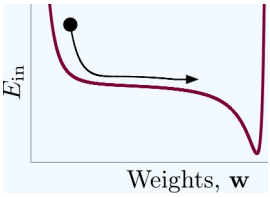
La velocidad a la que se minimiza +i. an depende en gran medida del algoritmo de optimización que se utilice. SGD parece significativamente mejor que el simple descenso de gradiente, pero podemos hacerlo mucho mejor, incluso SGD no es muy eficiente. En la Sección 7.5, discutimos algunos otros métodos potentes (por ejemplo, conju-
gate gradients) que pueden mejorar significativamente el descenso de gradiente y el SGD, al hacer un uso más eficaz del gradiente.

Inicialización y terminación. Elegir los pesos iniciales y decidir cuándo detener el descenso de gradiente puede ser complicado, en comparación con la regresión logística, porque el error en la muestra ya no es convexo. Según el Ejercicio 7.7, si los pesos se inicializan demasiado grandes para que tanh(w*xp) -I-1, entonces el gradiente será cercano a cero y el algoritmo no llegará a ninguna parte. Esto es especialmente problemático si se inicializan los pesos con el signo equivocado. Normalmente es mejor inicializar los pesos a valores srm// ran- dom donde tanh(w*xp) 0 para que el algoritmo tenga la flexibilidad de mover los pesos fácilmente para ajustarse a los datos. Una buena opción es inicializar usando pesos aleatorios gaussianos, en; $N(0, o2)$ donde w2 es pequeño. Pero, ¿cómo de pequeño debe ser cr2? Una heurística sencilla es que queremos que |w*xp |² sea pequeño. Dado que Ep |w*xq |°J = w2 | xq ||² , deberíamos elegir w de modo que w maxp ||xp | 1.

Ejercicio 7.9

¿Qué puede salir mal si se inicializan todos los pesos exactamente a cero?

¿Cuándo nos detenemos? Es arriesgado basarse únicamente en el tamaño del gradiente para detenerse. Como se ilustra a la derecha, podría detenerse prematuramente cuando la iteración alcanza una región relativamente plana (lo que es más común de lo que podría sospecharse). En la práctica, lo mejor es una combinación de criterios de parada, por ejemplo, detenerse sólo cuando se produce una mejora marginal del error junto con un gradiente pequeño.



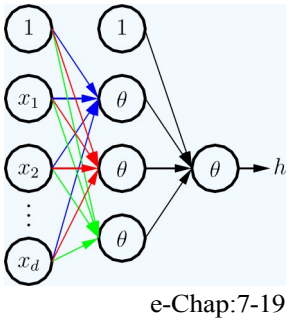
más un límite superior en el número de iteraciones.

7.2.4 Regresión para la clasificación

En el capítulo 3, mencionamos que podía utilizar los pesos resultantes de la regresión lineal como pesos de perceptrón para la clasificación, y puede hacer lo mismo con las redes neuronales. Específicamente, ajuste los datos de clasificación (pq - -1-1) como si fuera un problema de regresión. Esto significa que utiliza la función de identidad como transformación del nodo de salida, en lugar de tanh( ). Esto puede ser de gran ayuda debido a las "regiones planas" a las que es susceptible la red cuando se utiliza el descenso de gradiente, lo que ocurre a menudo en el entrenamiento. La razón de estos periodos planos en la optimización es la naturaleza excepcionalmente plana de la función tanh cuando su argumento se hace grande. Si por alguna razón los pesos se hacen grandes hacia el principio del entrenamiento, entonces la superficie de error empieza a parecer plana, porque la tanh se ha saturado. Ahora, el descenso de gradiente no puede hacer ningún progreso y usted podría pensar que está en un mínimo, cuando en realidad está lejos de un mínimo. El problema de una superficie de error plana se mitiga considerablemente cuando la transformación de salida es la identidad, porque puedes recuperarte de un mal movimiento inicial si resulta que te lleva a pesos grandes (la salida lineal nunca se satura). Para un ejemplo concreto de un error prematuramente plano en la muestra, vea la figura del Ejemplo 7.2 en la página 25.

7.3 Aproximación frente a generalización

Un MLP suficientemente grande con 2 capas ocultas puede aproximar funciones de decisión suaves arbitrariamente bien. Resulta que basta con una sola capa oculta'. Una red neuronal con una sola capa oculta que tiene m unidades ocultas *(d!" m*) implementa una función de la forma



$$h(\mathbf{x}) = \theta\left(w_{01}^{(2)} + \sum_{j=1}^{m} w_{j1}^{(2)}\,\theta\left(\sum_{i=0}^{d} w_{ij}^{(1)} x_i\right)\right)$$

"Aunque una capa oculta es suficiente, no es necesariamente la forma más eficiente de ajustarse a los datos; por ejemplo, puede existir una red de 2 capas ocultas mucho más pequeña.

Se trata de una representación engorrosa para una red tan sencilla. Una notación simplificada para este caso especial es mucho más conveniente. Para los pesos de la segunda capa, utilizaremos simplemente vq, , ... y utilizaremos v para denotar la columna j de la matriz de pesos de la primera capa W(*), para j = 1 ... rn. Con esta notación más sencilla, la hipótesis tiene un aspecto mucho más agradable:

$$h(\mathbf{x}) = \theta\left(w_0 + \sum_{i=1} w_j \theta\left(\mathbf{v}_j^{\mathrm{T}}\mathbf{x}\right)\right)$$

Red neuronal frente a transformadas no lineales. Recordemos el modelo lineal del capítulo 3, con la transformada no lineal 4-(x) que transforma x en z:

$$\mathbf{x} \rightarrow \mathbf{z} = \mathbf{\Phi}(\mathbf{x}) = [1, \phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_m(\mathbf{x})]^{\mathrm{T}}.$$

El modelo lineal con transformada no lineal es una hipótesis de la forma

$$h(\mathbf{x}) = \theta\left(w_0 + \sum_{\quad}^{M} w_j \phi_j(\mathbf{x})\right).$$

Las a' ( ) se denominan funciones de base. A primera vista, la red neuronal y el modelo lineal parecen casi idénticos, estableciendo 8(v; x) = d' (x). Sin embargo, hay una sutil diferencia que tiene un gran impacto práctico. Con la transformada no lineal, las funciones de base a' ( ) se fijan de antemano antes de observar los datos. Con la red neuronal, la "función base" 8(vjx) tiene un parámetro v en su interior, y podemos ajustar v sin ver los datos. En primer lugar, esto tiene un impacto computacional porque el parámetro v aparece *dentro de* la no linealidad 8 ( ) ; el modelo ya no es lineal en sus parámetros. Vimos un efecto similar con los centros de la red de función de base radial en el Capítulo 6. Los modelos que no son lineales en sus parámetros plantean un reto computacional importante cuando se trata de ajustarlos a los datos. En segundo lugar, significa que podemos tfie verdaderos hosts /unc/iones a /fie *datos. Las funciones de* base sintonizable, aunque computacionalmente más difíciles de ajustar a los datos, nos dan mucha más flexibilidad para ajustar los datos que las funciones de base fija. Con m funciones de base sintonizables se tiene aproximadamente el mismo poder de aproximación para ajustar los datos que con
md funciones de base fijas. Para *d* grandes, las funciones de base sintonizables tienen considerablemente
más potencia.

Ejercicio 7\10

No es de extrañar que añadir nodos en la capa oculta proporcione a la red neuronal una mayor capacidad de aproximación, ya que se añaden más parámetros.

¿Cuántos parámetros de peso hay en una red neuronal con arquitectura especificada por d $(d''', d'^{1}{}', \ldots, d''')$, un vector que da el número de nodos en cada capa? Evalúe su fórmula para una red de 2 capas ocultas con 10 nodos ocultos en cada capa oculta.

Capacidad de aproximación de la red neuronal. Es posible cuantificar cómo crece la capacidad de aproximación de la red neuronal a medida que aumenta m, el número de unidades ocultas. Estos resultados entran dentro del campo conocido como teoría de aproximación funcional, un campo que, en el contexto de las redes neuronales, ha producido algunos resultados interesantes. Por lo general, se empieza por hacer alguna suposición sobre la suavidad (o complejidad) de la función objetivo J. Desde el punto de vista teórico, se ha perdido algo de generalidad en comparación con, por ejemplo, el análisis VC. Sin embargo, en la práctica, estas suposiciones están bien porque las funciones objetivo suelen ser suaves. Si se supone que los datos son generados por una función objetivo con una complejidad' de Cy como máximo, entonces existe una variedad de límites sobre lo pequeño que puede ser un error en la muestra con m unidades ocultas. Para la regresión con error al cuadrado, se puede lograr un error en la muestra

$$E_{\text{in}}(h) = \frac{1}{N}\sum^{N} h(\mathbf{x}_n) - y_n)^2 < \frac{(2RC_f)^2}{}$$

donde fi = maxp ||xp | es el "radio" de los datos. El error en la muestra disminuye inversamente con el número de unidades ocultas. Para la clasificación, existe un resultado similar con una dependencia ligeramente peor de rn. Con alta probabilidad,

$$E_{\text{in}} \leq E^*_{\text{out}} + O(C_f/\sqrt{m}),$$

donde $_{\text{Appt}}$ es el error fuera de muestra del clasificado óptimo que discutimos en el Capítulo 6. El mensaje es que A;p puede hacerse pequeño eligiendo una capa oculta suficientemente grande.

Generalización y dimensión VC. Para m suficientemente grande, podemos conseguir que Aip sea pequeño, así que lo que queda es asegurar th£tf $_{\text{en}}$ opt. Debemos fijarnos en la dimensión VC. Para la red neuronal de dos capas de umbral duro (MLP) donde 8(z) = sign(z), mostramos un simple límite en la dimensión VC:

$$dzz \quad \text{(const) } md \log(\text{md}). \tag{7.7}$$

Para una red neuronal sigmoide general, *dzz* puede ser infinito. Para la sigmoide tanh( ), con nodo de salida sign( ), se puede demostrar que *dzz O (* HQ) donde U es el número de nodos ocultos y Q es el número de pesos; para el caso de dos capas

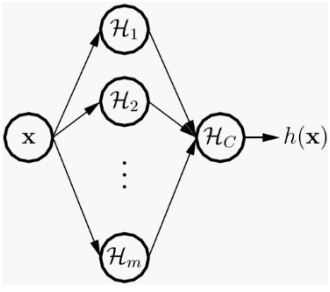
$$d \quad \text{- } O(md(m \quad d)).$$

La red tanh( ) tiene mayor dimensión VC que el MLP de 2 capas, lo que no es sorprendente porque tanh( ) puede aproximarse a sign( ) eligiendo pesos suficientemente grandes. Por lo tanto, todas las dicotomías que pueden ser implementadas por el MLP también pueden ser implementadas por la red neuronal tanh( ).

7No describimos los detalles de cómo se puede medir la complejidad de un objetivo. Una

medida es el tamaño de los componentes de "alta frecuencia" de J en su transformada de Fourier. Otra medida más restrictiva es el número de derivadas acotadas que tiene /.

Para derivar (7.7), en realidad mostraremos un resultado más general. Consideremos el conjunto de hipótesis ilustrado por la red de la derecha. El nodo oculto i de la capa oculta implementa una función fi; C 7-/i que mapea *id* a(-I-l, -1}; el nodo de salida implementa una función fi, C 7-fy que mapea a (-1-1, -1}. Este nodo de salida combina las salidas de los nodos de la capa oculta para implementar la hipótesis



$$h(\mathbf{x}) = h_C(h_1(\mathbf{x}), \ldots, h_m(\mathbf{x})).$$

(Para el MLP de 2 capas, todos los conjuntos de hipótesis son perceptrones).

Supongamos que la dimensión VC de H, es d y la dimensión VC de Hy es *d* . Fijemos xj , ... , xy, y las hipótesis fi , ... , fip implementadas por los nodos ocultos. Las hipótesis fit, ... , fly son ahora funciones de base fija que definen una transformada a R",

$$\mathbf{x}_1 \to \mathbf{z}_1 = \begin{matrix} h_1(\mathbf{x}_1) \\ \\ h_m(\mathbf{x}_1) \end{matrix} \quad \cdots \quad \mathbf{x}_N \to \mathbf{z}_N = \begin{matrix} h_1(\mathbf{x}_N) \\ \\ h_m(\mathbf{x}_N) \end{matrix}$$

Los puntos transformados son vectores binarios en " . Dados fit , ... , fly, los puntos xj , ... , xy se transforman en una disposición de puntos z , ... , zy enR'. Utilizando nuestra flexibilidad para elegir fi , ... , fip, ahora acotamos el número de posibles arreglos *di fferenl* z , ... , zy que podemos obtener.

Los primeros componentes de todos los zq vienen dados por fit (x), ... , fit (xy), que es una dicotomía de x , ... , xy implementada por fit . Como la dimensión VC de 7-I es *d ,* hay como máximo N^d dicotomías de este tipo.[8] Es decir, hay como máximo N^d formas diferentes de elegir asignaciones a *todos los* primeros componentes de la zq . Del mismo modo, una asignación a todos los componentes Cth puede elegirse como máximo de N^d maneras. Así pues, el número total de disposiciones posibles para Z J , ... , Zy iS £tt mOSt

$$\prod N^{d_i} = N^{\sum_{i=1}^{m} d_i}.$$

Cada una de estas disposiciones puede dicotomizarse como máximo de *N* maneras^{d} -′, ya que la dimensión VC de Wp es *d* . Cada una de estas dicotomías para un arreglo particular da una dicotomía de los datos x , ... , xp. xp. Por lo tanto, el número máximo de dicotomías diferentes que podemos implementar en xj , ... xy está limitado por el producto: el número de disposiciones posibles por el número de maneras

8Recordemos que para cualquier conjunto de hipótesis con dimensión VC *dvo* y cualquier *N* > *dcc ,* rn(N) (el número máximo de dicotomías implementables) está limitado por $ed/dvc)^d \circ < N^d \circ\circ$ (para simplificar suponemos que *dcc* > 2).

de dicotomizar un arreglo particular. Hemos demostrado que

$$m(N) \leq N^{d_c} \cdot N^{\sum_{i=1}^{m} d_i} = N^{d_c + \sum_{i=1}^{m} d_i}.$$

Sea *D -- d* -l- *d;*. Después de un poco de álgebra (dejada al lector), si *N* 2D $\log_2 D$, entonces(N) de lo que concluimos que dpz2D $\log_2 D$. Para el MLP de 2 capas, d = d - l- 1 y d, = rn -l- 1 , por lo que tenemos que *D* - d_ *d; - m(d* 2) -l- 1 - O *(md). Por* tanto, *dzz - O* (md log(md)). Nuestro análisis parece muy burdo, pero es casi ajustado: es posible romper n(md) puntos con m unidades ocultas (véase el problema 7.16), por lo que el límite superior puede ser flojo como mucho en un factor logarítmico. Utilizando la dimensión VC, la barra de error de generalización del capítulo 2 es O(*N)/N)* que para el org*(d* MLP de 2 capas es O((d o *(d)* log *)/N)* .

Obtendremos una buena generalización si m no es demasiado grande y podemos ajustar los datos si m es suficientemente grande. Se impone un equilibrio. Por ejemplo, elegir m ₚ como *N* -' *on* tout in £tnd Aip -' fi q,. Es decir, A..t q, (el rendimiento óptimo) a medida que crece *N,* y m crece de forma sublineal con *N.* En la práctica, el régimen "asintótico" es un lujo y no basta con fijar rn . Estos resultados teóricos son una buena orientación, pero el mejor rendimiento fuera de la muestra suele obtenerse cuando se controla el sobreajuste utilizando la validación (para seleccionar el número de unidades ocultas) y la regularización para evitar el sobreajuste.

Concluimos con una nota sobre dónde se sitúan las redes neuronales en el debate paramétrico-no paramétrico. Hay parámetros explícitos que aprender, por lo que para- métrico parece correcto. Pero también destacan los rasgos distintivos de los modelos no paramétricos: la red neuronal es genérica y flexible y puede alcanzar un rendimiento óptimo cuando *N* crece. Ni los modelos paramétricos ni los no paramétricos abarcan toda la historia. Optamos por etiquetar las redes neuronales como *scmi-param e Iric.*

7.4 Regularización y validación

La red neuronal multicapa es potente y, unida al descenso por gradiente (un buen algoritmo para minimizar Aip), tenemos una receta para el sobreajuste. Analizamos algunas técnicas prácticas de ayuda.

7.4.1 Penalizaciones de complejidad basadas en el peso

Al igual que con los modelos lineales, se puede regularizar el aprendizaje utilizando una penalización de complejidad minimizando un error aumentado (error penalizado en la muestra). El regularizador de caída de peso al cuadrado es popular, ya que tiene un error aumentado:

$$E_{\text{aug}}(\mathbf{w}) \qquad E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N} \sum (w_{ij}^{(\ell)})^2$$

El parámetro de regularización 3 se selecciona mediante validación, como se explica en el Capítulo 4. Para aplicar el descenso de gradiente, necesitamos Wfi,qg (w). El término de penalización añade

al gradiente un término proporcional a los pesos,

$$\frac{\partial E_{\text{aug}}(\mathbf{w})}{\partial \mathrm{W}^{(\ell)}} \quad \frac{\partial E_{\text{in}}(\mathbf{w})}{\partial \mathrm{W}^{(\ell)}} + \frac{2\lambda}{N}\mathrm{W}^{(\ell)}$$

Sabemos cómo obtener fiA;p /fiW" mediante retropropagación. El término de penalización añade un componente a la actualización del peso que está en la dirección negativa de w,
es decir, hacia pesos cero de ahí el término decaimiento del peso.

Otro regularizador similar es la eliminación del ceipfit, con error aumentado:

Laug(,A)' Min( ) $$\frac{\lambda}{N}\sum_{\ell,i,j}\frac{(w_{ij}^{(\ell)})^2}{1+(w_{ij}^{(\ell)})^2}.$$

Para un peso pequeño, el término de penalización es muy parecido al decaimiento del peso, y lo reducirá a cero. Para un peso grande, el término de penalización es aproximadamente una constante y contribuye poco al gradiente. Los pesos pequeños decaen más rápido que los pesos grandes, y el efecto es "eliminar" esos pesos más pequeños.

Ejercicio 7%ll

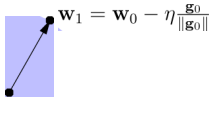
Para la eliminación del peso, demues... $\frac{\partial E_{\text{aug}}}{\partial w_{ii}^{(\ell)}} = \frac{\partial E_{\text{in}}}{\partial w_{ii}^{(\ell)}} + \frac{2\lambda}{N}\cdot\frac{w_{ij}^{(\ell)}}{(1+(w_{ij}^{(\ell)})^2)^2}$.

Argumente que la eliminación de pesos reduce más rápidamente los pesos pequeños que los grandes.

7.4.2 Parada anticipada

Otro método de regularización, que a primera vista no se parece a la regularización, es la parada temprana. Un método iterativo como el descenso de gradiente no explora todo el conjunto de hipótesis de una vez. Con más iteraciones, se explora más del conjunto de hipótesis. Esto significa que utilizando menos iteraciones, se explora un conjunto de hipótesis más pequeño y se obtiene una mejor generalización".

Consideremos el descenso de gradiente de paso fijo con tamaño de paso p. En el primer paso, empezamos en los pesos wk, y damos un paso de tamaño p hasta w = wk - p * .



Porque hemos dado un paso en la dirección de el gradiente negativo, hemos 'mirado' todos los hipótesis en la región sombreada de la derecha.

Esto se debe a que un paso en el gradiente negativo conduce a la disminución más brusca de

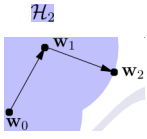
en ( ) , y por tanto w minimizaen ( ) entre todas las ponderaciones con | w - wp | | p.

Buscamos indirectamente en todo el conjunto de hipótesis

y elegimos la hipótesis wt C by con el mínimo error dentro de la muestra.

Si queremos ser rigurosos con la corrección, el conjunto de hipótesis explorado podría depender del conjunto de datos, por lo que no podemos aplicar directamente el análisis de la CV, que requiere que el conjunto de hipótesis esté fijado de antemano. Dado que sólo estamos ilustrando la idea principal, pasaremos por alto estos tecnicismos.

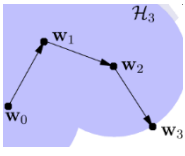
Consideremos ahora el segundo paso, ilustrado a la derecha, que pasa a Exploramos indirectamente el conjunto de hipótesis de pesos con | w - w | $\mathbf{p}$, eligiendo la mejor. Como wj ya era el minimizador de fi,q sobre W9, esto significa que 2 es el minimizador de €ip entre todas las hipótesis en W2, donde



$$\mathcal{H}_2 = \mathcal{H}_1 \cup \{\mathbf{w} : \ \|\mathbf{w} - \mathbf{w}_1\| \leq \eta\}.$$

Obsérvese que W W2. Del mismo modo, definimos el conjunto de hipótesis

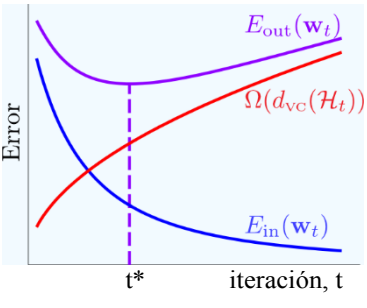
$$\mathcal{H}_3 = \mathcal{H}_2 \cup \{\mathbf{w} : \ \|\mathbf{w} - \mathbf{w}_2\| \leq \eta\},$$



y en la 3ª iteración, elegimos pesos w3 que min- imizan en OVer w 3 Podemos continuar este argumento a medida que avanza el descenso gradiente, y definir una secuencia anidada de conjuntos de hipótesis

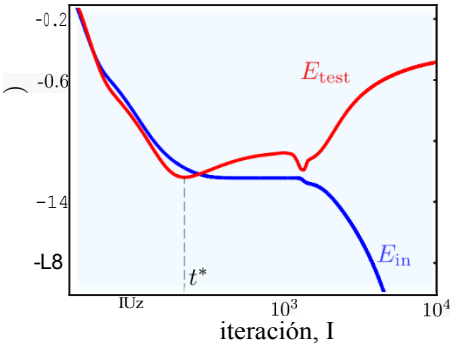
$$\mathcal{H}_1 \subset \mathcal{H}_2 \subset \mathcal{H}_3 \subset \mathcal{H}_4 \subset \cdots.$$

A medida que I aumenta, in (t) disminuye y *dpz (F)* aumenta. Por lo tanto, expectamos ver la compensación aproximación-generalización que se ilustró en la Figura 2.3 (reproducida aquí con la iteración I como proxy para *dzz)*:



La cifra sugiere que puede ser mejor parar pronto en algún I*, mucho antes de que alcanzando un mínimo de i-n De hecho, este panorama se observa en la práctica.

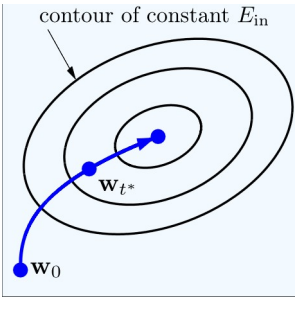
Ejemplo T.2. Volvemos a la tarea de dígitos de clasificar "1" frente a todos los demás dígitos, con 70 puntos de datos seleccionados aleatoriamente y una pequeña red neuronal sigmoidal con una sola unidad oculta y un nodo de salida tanh( ). La siguiente figura muestra el error en la muestra y el error de prueba frente al número de iteraciones.



Las curvas refuerzan nuestra discusión teórica: el error de prueba disminuye inicialmente a medida que la ganancia de aproximación supera la peor barra de error de generalización; después, el error de prueba aumenta a medida que la barra de error de generalización empieza a dominar la ganancia de aproximación, y el sobreajuste se convierte en un problema grave.

En el ejemplo anterior, a pesar de utilizar una red neuronal parsimoniosa con un único nodo oculto, el sobreajuste era un problema porque los datos son ruidosos y la función objetivo es compleja, por lo que tanto el ruido estocástico como el determinista son significativos. Tenemos que regularizar.

En el ejemplo, es mejor parar pronto en t* y cons/rninar el aprendizaje al conjunto de hipótesis más pequeño 7-I,. . En este sentido, la detención temprana es una forma de regularización. La parada temprana está relacionada con el decaimiento del peso, como se ilustra a la derecha. Se inicializa wp cerca de cero; si se detiene pronto en wt. se habrá detenido en pesos más cercanos a wk, es decir, pesos más pequeños. La parada temprana logra indirectamente pesos más pequeños, que es lo que el decaimiento del peso logra directamente. Para determinar cuándo detener el entrenamiento, utilice un conjunto de validación para controlar la error de validación en la iteración I a medida que minimizas el error del conjunto de entrenamiento. Indique los pesos w,. que tienen el mínimo error de validación cuando termine el entrenamiento.



Después de seleccionar I*, es tentador utilizar todos los datos para entrenar para iteraciones I*. Desafortunadamente, volver a añadir los datos de validación y entrenar para I* iteraciones puede conducir a un conjunto de pesos completamente diferente. La estimación de validación del rendimiento sólo es

válida para w,. (los pesos que debe emitir). Esto parece ir en contra de la sabiduría de la curva de aprendizaje decreciente del capítulo 4: si aprendes con más datos, obtienes una hipótesis final mejor.[10]

* Tampoco se recomienda utilizar todos los datos para entrenar hasta un error en la muestra de >i "i (ut-). Además, un error en la muestra de <i. i (<t-) puede incluso no ser alcanzable con todos los datos.

Ejercicio 7\12

¿Por qué la salida con/- en lugar de entrenar con todos los datos para /" iteraciones no va en contra de la sabiduría de que aprender con más datos es mejor.

{Pista: "Más ***datos es mejor" se aplica a un*** *modelo* **fijo** *(J-[, A).* La ***parada*** *temprana es la selección del modelo en un conjunto de hipótesis anidadas* H1 C H2 C - - *determinadas por O" ; . ¿Qué ocurre si se utilizan todos los datos D)?*

Cuando se utiliza la parada anticipada, existe un equilibrio habitual a la hora de elegir el tamaño del conjunto de validación: si es demasiado grande, habrá pocos datos para entrenar; si es demasiado pequeño, el error de validación no será fiable.

Ejercicio 7\13

Supongamos que ejecuta el descenso por gradiente durante 1000 iteraciones. Tiene 500 ejemplos en @, y utiliza 450 para c..in y 50 para @"\\ . Se obtiene el peso de la iteración 50, con A"\ (wso) 0 . 05 yStrain(50) ' 0-04.

(a) ¿Es C "i(who) 0,05 una estimación insesgada de *T: "i (wsD)?*

(b) Utilice el límite de HoefFding para obtener un límite para *T: "i* utilizando *T:"* más una barra de error. El límite debe mantenerse con una probabilidad de al menos 0,1.

(c) ¿Puedes enlazar *T: "i* utilizando *T: ";* o necesitas más información?

El ejemplo 7.2 también ilustra otro problema común con el nodo de salida sigmoidal: el descenso de gradiente a menudo encuentra una región plana donde A,q disminuye muy poco." Se puede detener el entrenamiento pensando que se ha encontrado un mínimo local. Esta "parada temprana" por error se denomina a veces la propiedad de "autorregularización" de las redes neuronales sigmoidales. La regularización accidental debida a una convergencia mal interpretada no es fiable. La validación es mucho mejor.

7.4.3 Experimentos con datos numéricos

Pongamos en práctica la teoría en la tarea de los dígitos (clasificar "1" frente a todos los demás dígitos). Aprendemos sobre 500 puntos de datos elegidos al azar utilizando una red neuronal sigmoidal con una capa oculta y 10 nodos ocultos. Hay 41 pesos (parámetros sintonizables), por lo que hay más de 10 ejemplos por grado de libertad, lo que es bastante razonable. Utilizamos la transformación de salida de identidad 8(s) - s para reducir la posibilidad de quedarse atascado en una región plana de la superficie de error. Al final del entrenamiento, utilizamos la transformación de salida 6(s) = sign(s) para clasificar realmente los datos. Después de más de 2 *millones de* iteraciones de descenso de gradiente, conseguimos acercarnos a un mínimo local. El resultado

se muestra en la figura 7.2.

No hace falta ser un genio para darse cuenta del sobreajuste. La figura 7.2 demuestra la capacidad de aproximación de una red neuronal de tamaño moderado. Probemos el peso

" La función de transformación lineal de la salida ayuda a evitar esas regiones excesivamente planas.

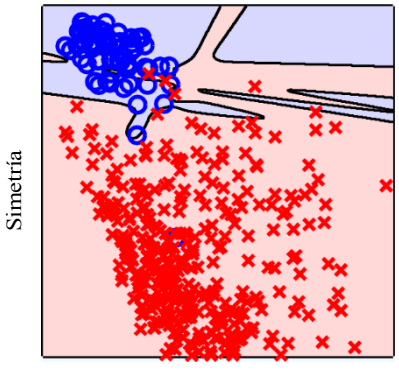
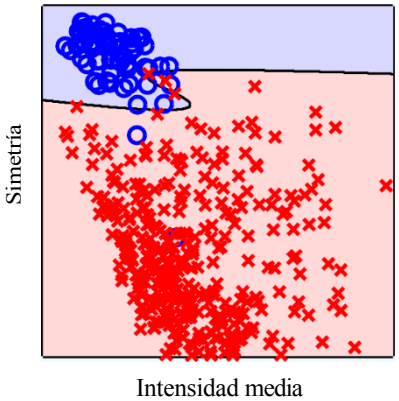


Figura 7.2: Red neuronal de 10 unidades ocultas entrenada con descenso gradiente en 500 ejemplos de los datos de dígitos (sin regularización). Los círculos azules son el dígito "1" y las x rojas son los demás dígitos. El sobreajuste es rampante.

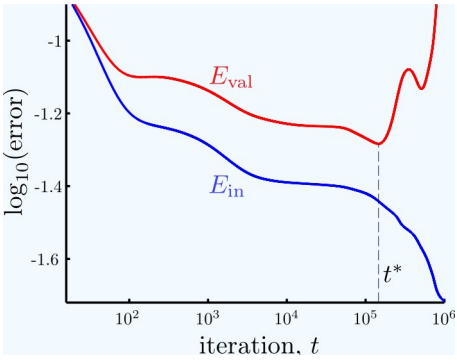
para combatir el sobreajuste. Minimizamos A,pg(w:) ' en ( ) * , con I = 0,01. Obtenemos un separador mucho más creíble, que se muestra a continuación.



Como ilustración final, probemos la parada anticipada con un conjunto de validación de tamaño 50 (una décima parte de los datos); así, el conjunto de entrenamiento tendrá ahora un tamaño de 450.

La dinámica de entrenamiento del descenso de gradiente se muestra en la Figura 7.3(a). La función de transformación de salida lineal ha ayudado a que no haya periodos extremadamente planos en el error de entrenamiento. El límite de clasificación con parada temprana en I* se muestra en la Figura 7.3(b). El

resultado es similar a la caída de pesos. En ambos casos, el límite de clasificación regularizado es más creíble. En última instancia, lo que importa son las estadísticas cuantitativas, que se resumen a continuación.



(a) Dinámica de formación

Simetría

Intensidad media

(b) Hipótesis final

Figura 7.3: Parada temprana con 500 ejemplos de los datos de dígitos. (a) Errores de entrenamiento y validación para el descenso por gradiente con un conjunto de entrenamiento de tamaño 450 y un conjunto de validación de tamaño 50. (b) La hipótesis final "regularizada" obtenida mediante la parada temprana en t*, el error de validación mínimo.

	E_{train}	E_{val}	E_{in}	E_{out}
No Regularization	–	–	0.2%	3.1%
Weight Decay	–	–	1.0%	2.1%
Early Stopping	1.1%	2.0%	1.2%	2.0%

7.5 Mejorar el descenso por gradiente

El descenso gradiente es un método simple para minimizar Aip que tiene problemas de con- vergencia, especialmente con superficies de error planas. Una solución es minimizar un error más amigable, que es por lo que ayuda el entrenamiento con un nodo de salida lineal. En lugar de cambiar la medida del error, hay mucho margen para mejorar el propio algoritmo. El descenso gradiente da un paso de tamaño p en la dirección del gradiente negativo. ¿Cómo debemos determinar p y es el gradiente negativo la mejor dirección en la que moverse?

Ejercicio 7\14

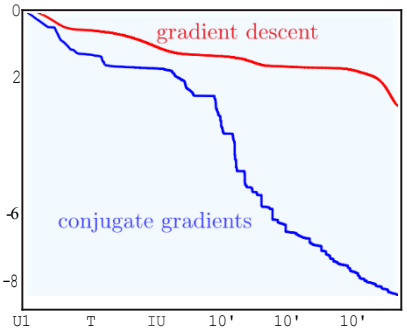
Consideremos la función de error(w) (w- w")"@(w - w"), donde @ es una matriz definida positiva arbitraria. Fijemos w0 .

Demuestre que el gradiente 9A(w)-@w" Qué pesos minimizan A(w). ¿El descenso de gradiente le mueve en la dirección de estos pesos óptimos?

Concilia tu respuesta con la afirmación del capítulo 3 de que el gradiente es la mejor dirección para dar un paso. *{Pista. ¿De qué tamaño era el paso?)*

El ejercicio anterior muestra que el gradiente negativo no es necesariamente la mejor dirección para un paso grande, y nos gustaría dar pasos más grandes para aumentar

la eficacia de la optimización. La siguiente figura muestra dos algoritmos: nuestro viejo amigo el descenso por gradiente y nuestro pronto amigo el descenso por gradiente conjugado. Ambos algoritmos minimizan A,q para una red neuronal de 5 unidades ocultas que ajusta datos de 200 dígitos. La diferencia de rendimiento es espectacular.

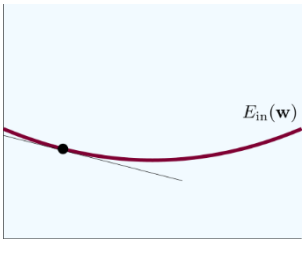


A continuación analizaremos métodos para "reforzar" el descenso de gradiente, pero sólo arañaremos la superficie de este importante tema conocido como optimización numérica. Los dos pasos principales en un procedimiento de optimización iterativo son determinar:

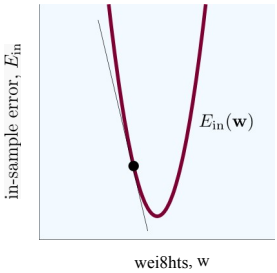
1. ¿En qué dirección hay que buscar un óptimo local?
2. ¿Hasta qué punto hay que dar un paso en esa dirección?

7.5.1 Elección de la tasa de aprendizaje p

En el descenso por gradiente, la tasa de aprendizaje p multiplica el gradiente negativo para dar el movimiento -pWfiip. El tamaño del paso dado es proporcional a p. El tamaño óptimo del paso (y, por tanto, la tasa de aprendizaje p) depende de lo ancha o estrecha que sea la superficie de error cerca del mínimo.



ancho: utilizar p grande.



estrecho: utilizar p minúscula.

Cuando la superficie es más ancha, podemos dar pasos más grandes sin excedernos; como || WAip || es pequeño, necesitamos una p grande. Como no sabemos de antemano lo ancha que es la superficie, es fácil elegir un valor ineficiente para p.

Descenso por gradiente con tasa de aprendizaje variable. Una simple heurística que adapta la tasa de aprendizaje a la superficie de error funciona bien en la práctica. Si el error disminuye, se aumenta p; si no, el paso ha sido demasiado grande, así que se rechaza la actualización y se disminuye p. A cambio de un pequeño esfuerzo adicional, obtenemos un aumento significativo del descenso por gradiente.

Descenso por gradiente **con tasa de** aprendizaje **variable**:

1: Inicializar w(0), y pp en / = 0. Establecer o1 y ß 1.
2: mientras no se cumpla el criterio de parada hacer
3: Sea g () - vL, (w (/) y sea v(/) = - g ( ) .
4: **si A, (w/) T v/)) <f" (w ( )) entonces**
5: acep: (+ 1) ( ) + v ( ) "- "
6: si no
7: rechazar: w(t -t- 1) = w(t); pt+ t = fipt.
8: Iterar hasta el paso siguiente, t +- / -t- 1.

Por lo general, lo mejor es utilizar un parámetro de incremento conservador, por ejemplo n 1 ,05 - 1,1, y un parámetro de decremento un poco más agresivo, por ejemplo fi 0 ,5 - 0,8. Esto se debe a que, si el error no disminuye, entonces se está en una situación inusual y se requiere una acción más drástica.

Después de pensarlo un poco, cabe preguntarse por qué necesitamos una tasa de aprendizaje. Una vez determinada la dirección en la que hay que moverse, v(I), ¿por qué no seguir simplemente en esa dirección hasta que el error deje de disminuir? Esto nos lleva al descenso por gradiente *más pronunciado* con *búsqueda de* línea.

Descenso más pronunciado. El descenso gradiente elige una dirección de descenso v(I) - g (I) y actualiza los pesos a w(I -I- 1) w(I) -I- lv(I). En lugar de elegir p arbitrariamente, elegiremos el óptimo p que minimice Aip (w(I -1- 1)). Una vez que tengas la dirección en la que moverte, hazlo lo mejor posible moviéndote a lo largo de la línea w(t) + lv(t) y deteniéndote cuando Aip sea mínimo (de ahí el término búsqueda lineal).
Es decir, elegir un tamaño de paso p*, donde

q'(t) - argiinL; (w(t)1 qv(f)).

Descenso más pronunciado (Gradient Descent -1- Line Search):

1: Inicializar w(0) y fijar I - 0;
2: mientras no se cumpla el criterio de parada hacer
3: Sea g(n) = ep ( ) , y fijar v(t) = - g(t).
4: Sea p* = ar gminq en (( ) QV(·
5: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta^* \mathbf{v}(t)$.
6: Iterar al paso siguiente, t '- t -t- 1.

La búsqueda lineal del paso 4 es un problema de optimización unidimensional.

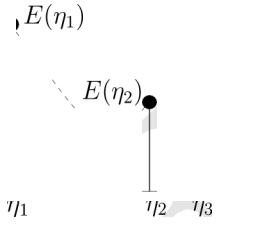
La búsqueda lineal es un paso importante en la mayoría de los algoritmos de optimización, por lo que se requiere un algoritmo eficiente. Escriba A(p) para A,q(w(I) -I- lv(I)). El objetivo es encontrar un mínimo de A (p). Damos un algoritmo sencillo basado en la búsqueda binaria.

Búsqueda en la línea. La idea es encontrar un intervalo en la línea que contenga un mínimo local. Luego, reducir rápidamente el tamaño de este intervalo manteniendo como invariante el hecho de que contiene un mínimo local.

El invariante básico es un arreglo en U:

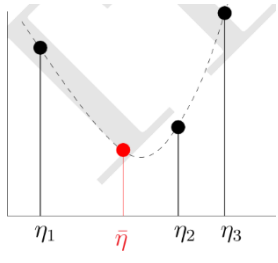
E gz)

$$\eta_1 < \eta_2 < \eta_3 \quad \text{with}$$
$$E(\eta_2) < \min\{E(\eta_1), E(\eta_3)\}.$$



Como fi es continua, debe haber un mínimo local en el intervalo p , p3). Consideremos ahora el punto medio del intervalo,

$$\bar{\eta} = \frac{\tilde{}}{2}(\eta_1 + \eta_3),$$



de ahí el nombre de *algoritmo de* bisección. Supongamos que h < c2 como se muestra. Si fi(p) < +(c2) entonces (ci c. c2) es un nuevo arreglo U más pequeño; y, si A( ) > +(z), entonces (F, 9z, } es el nuevo arreglo U más pequeño. En cualquier caso, el proceso de bisección se puede iterar con el nuevo arreglo en U . Si p resulta ser igual a c2 perturb p ligeramente para resolver la degeneración. Dejamos al lector que determine cómo obtener el nuevo arreglo U más pequeño para el caso p >

Un algoritmo eficaz para encontrar una disposición inicial en U es comenzar con pt - 0 y - e para algún paso e. Si A(9z) (pt), considere la secuencia 9 = 0, e, 2e, 4e, 8e,

...

(cada vez que se duplica el paso). En algún momento, el error debe aumentar. Cuando el error aumenta por primera vez, los tres últimos pasos dan una disposición en U. Si, en cambio, A(p) A (p2) , consideramos la secuencia

```
=^,0,-c,-2c,-4c,-8^,
```

(el paso sigue duplicándose pero en sentido inverso). De nuevo, cuando el error aumenta por primera vez, los tres últimos pasos dan un arreglo en U.'[2]

Ejercicio 7\15

Demuestre que |//a - //i] decrece exponencialmente en el algoritmo de bisección.
{Pista.' demuestran que dos iteraciones reducen al menos a la mitad el tamaño del intervalo).

El algoritmo de bisección continúa bisecando el intervalo y actualizando a una nueva disposición en U hasta que la longitud del intervalo |93 - p | es lo suficientemente pequeña, momento en el cual

'² No nos preocupamos por €(pJ) = (02)- tales vínculos pueden romperse con pequeñas perturbaciones.

puede devolver el punto medio del intervalo como mínimo local aproximado. Normalmente 20 iteraciones de bisección son suficientes para obtener una solución aceptable. En el problema 7.8 se da un algoritmo de interpolación cuadrática mejor, que en la práctica sólo necesita unas 4 iteraciones.

Ejemplo 7.3. Ilustramos estas tres heurísticas para mejorar el gradiente descenso en nuestro reconocimiento de dígitos (clasificación de "1" frente a otros dígitos). Utilizamos
200 puntos de datos y una red neuronal con 5 unidades ocultas. En la Figura 7.4 mostramos el rendimiento del descenso por gradiente, el descenso por gradiente con tasa de aprendizaje variable y el descenso más pronunciado (búsqueda lineal). La tabla siguiente resume el error en la muestra en varios puntos de la optimización.

Método	Tiempo de optimización		
	10 segundos	1.000 seg.	50.000 seg.
Descenso gradual	0.079	0.0206	0.00144
Descenso Gradiente Estocástico	0.0213	0.00278	0.000022
Tasa de aprendizaje variable	0.039	0.014	0.00010
Descenso más pronunciado	0.043	0.0189	0.000012

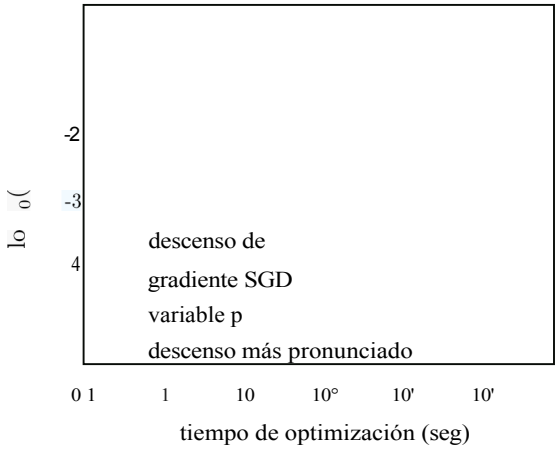


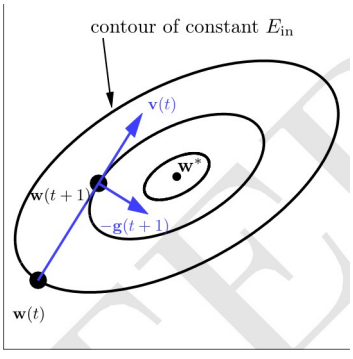
Figura 7.4: Descenso gradual, tasa de aprendizaje variable y descenso más pronunciado utilizando datos de dígitos y una red neuronal de 2 capas y 5 unidades ocultas con salida lineal. Para la tasa de aprendizaje variable, o = 1,1 y fi = 0,8.

Obsérvese que el SGD es bastante competitivo. La figura ilustra por qué es difícil saber cuándo dejar de minimizar. Una región plana "atrapó" a todos los métodos, a pesar de que utilizamos una transformación lineal del nodo de salida. Es muy difícil diferenciar entre una región plana (que suele estar causada por un valle muy pronunciado que conduce a un comportamiento en zig-zag

ineficiente) y un verdadero mínimo local.

7.5.2 Minimización del gradiente conjugado

El gradiente conjugado es la reina entre los métodos de op- timización porque aprovecha un principio sencillo. No deshaga lo que ya ha conseguido. Cuando finaliza una búsqueda lineal, dado que el error no puede reducirse aún más retrocediendo o avanzando a lo largo de la dirección de búsqueda, debe ser que el nuevo gradiente y la dirección de búsqueda lineal anterior sean ortogonales. Esto significa que se ha conseguido que una de las componentes del gradiente sea cero, es decir, la componente a lo largo de la dirección de búsqueda v(/) (véase la figura) .



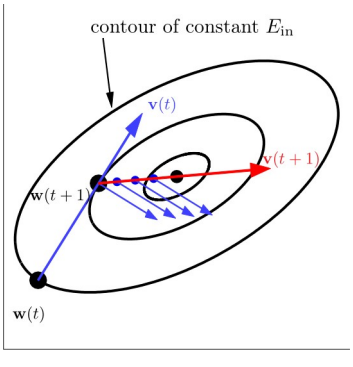
U"l

Si la siguiente dirección de búsqueda es la negativa del nuevo gradiente, será ortogonal a la dirección de búsqueda anterior.

Se está en un mínimo local cuando el gradiente es cero, y poner un componente a cero es sin duda un paso en la dirección correcta. A medida que se desplaza a lo largo de la siguiente dirección de búsqueda (por ejemplo, el nuevo gradiente negativo), el gradiente cambiará y puede que no permanezca ortogonal a la dirección de búsqueda anterior, una tarea que realizó laboriosamente en la búsqueda de línea anterior. El algoritmo de gradiente conjugado elige la siguiente dirección v(I -I- 1) de forma que el gradiente a lo largo de esta dirección, *permanecerá perpendicular Io a la dirección de* búsqueda *anterior* v(I). Esto se llama la dirección conjugada, de ahí el nombre.

Después de una búsqueda lineal a lo largo de esta nueva dirección v(/ + 1) para minimizar min' se habrán puesto a cero dos componentes del gradiente. En primer lugar, el gradiente permanece perpendicular a la dirección de búsqueda anterior v(I).

En segundo lugar, el gradiente será ortogonal a v (/ + 1) debido a la búsqueda de la línea (véase la figura). El gradiente a lo largo de la nueva dirección v(I -I- 1) se muestra con las flechas azules de la figura. Como v(I -I- 1) es conju- nal a v(I), observe cómo el gradiente a medida que nos movemos a lo largo de v(I -I- 1) permanece ortogonal a la dirección anterior v(/).



w_1

Ejercicio 7.16

¿Por qué la nueva dirección de búsqueda pasa por los pesos óptimos?

Hemos progresado. Ahora dos componentes del gradiente son cero. En dos

dimensiones, esto significa que el propio gradiente debe ser cero y hemos terminado.

En una dimensión más alta, si pudiéramos continuar poniendo un componente del gradiente a cero con cada búsqueda de línea, manteniendo todos los componentes anteriores a cero, eventualmente pondríamos cada componente del gradiente a cero y estaríamos en un mínimo local. Nuestra discusión es válida para una función de error cuadrática idealizada. En general, la minimización del gradiente conjugado implementa nuestras expectativas idealizadas aproximadamente. Sin embargo, funciona a las mil maravillas porque la configuración idealizada es una buena aproximación una vez que nos acercamos a un mínimo local, y aquí es donde algoritmos como el descenso de gradiente se vuelven ineficaces.

Pasemos ahora a los detalles. El algoritmo construye la dirección de búsqueda actual como una combinación lineal de la dirección de búsqueda anterior y el gradiente actual,

v(t) - -g(t) -t- " v(t - *)

donde

$$\mu_t = \frac{\mathbf{g}(t+1)^{\mathrm{T}}(\mathbf{g}(t+1) - \mathbf{g}(t))}{\mathbf{g}(t)^{\mathrm{T}}\mathbf{g}(t)}.$$

El término yt v(I - 1) se denomina término de impulso porque te pide que sigas moviéndote en la misma dirección en la que te estabas moviendo. El multiplicador y, se denomina parámetro de impulso. El algoritmo completo de descenso de gradiente conjugado se resume en el siguiente cuadro de algoritmo.

Descenso por gradiente conjugado:

1: Inicializar w(0) y fijar I = 0; fijar v(- 1) = 0
2: mientras no se cumpla el criterio de parada hacer
3: Sea v(I) = -g(I) + ytv(I - 1), donde

$$\underline{\mathbf{g}(t+1)^{\mathrm{T}}(\mathbf{g}(t+1) - \mathbf{g}(t))}$$

4: Let $\eta^* = \operatorname{argmin}_\eta E_{\text{in}}(\mathbf{w}(t) + \eta\mathbf{v}(t))$.
5: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta^*\mathbf{v}(t)$;
6: Iterar al paso siguiente, I '- I -I- 1;

La única diferencia entre el descenso por gradiente conjugado y el descenso más pronunciado está en el paso 3, en el que la dirección de búsqueda de la línea es distinta de la del gradiente negativo. Contrariamente a la intuición, la dirección del gradiente negativo no siempre es la mejor dirección para moverse, ya que puede deshacer parte del buen trabajo realizado anteriormente. En la práctica, para las superficies de error que no son exactamente cuadráticas, los v(I) son sólo aproximadamente conjugados y se recomienda "reiniciar" el algoritmo poniendo yt a cero cada cierto tiempo (por ejemplo cada *d* iteraciones). Es decir, cada *d* iteraciones se lanza una iteración de descenso más pronunciado.

Ejemplo 7.4. Siguiendo con el ejemplo de los dígitos, comparamos el gradiente conjugado y el descenso más pronunciado del campeón anterior en la siguiente tabla y en la Figura 7.5.

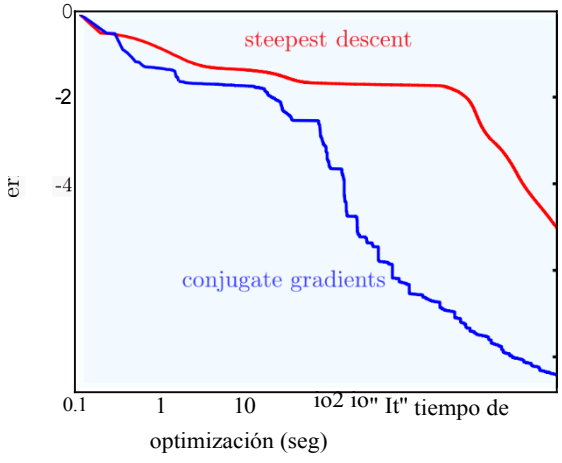


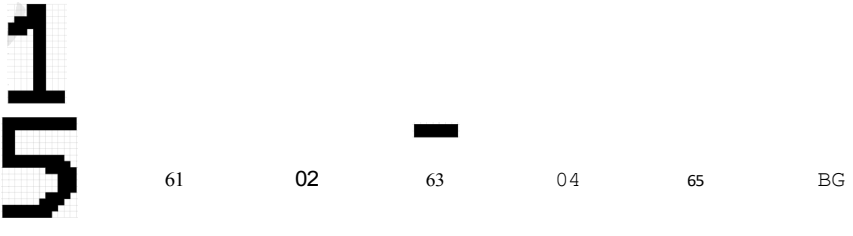
Figura 7.5: Descenso más pronunciado frente a descenso de gradiente conjugado utilizando 200 ejemplos de los datos de dígitos y una red neuronal sigmoidal de 2 capas con 5 unidades ocultas.

Method	Optimization Time		
Método	Optimización		
	10 seconds	1,000 sec	50,000 sec
Steepest Descent	0.043	0.0189	1.2×10^{-5}
Descenso más pronunciado	0.043	1.13×10^{-6}	2.73×10^{-9}
Conjugate Gradients	0.0200		
Gradientes conjugados	0.0200		2.73 x 10

La diferencia de rendimiento es espectacular. □

7.6 Aprendizaje profundo: Redes con muchas capas

La aproximación universal dice que una sola capa oculta con suficientes unidades ocultas puede aproximar cualquier función objetivo. Pero puede que no sea una forma natural de representar la función objetivo. A menudo, muchas capas imitan mejor el aprendizaje humano. Empecemos con el problema del reconocimiento de dígitos para clasificar "1" frente a "5". Un primer paso natural es descomponer los dos dígitos en componentes básicos, igual que se descompone una cara en dos ojos, una nariz, una boca, dos orejas, etc. He aquí un intento de clasificar un "1" y un "5" prototípicos.

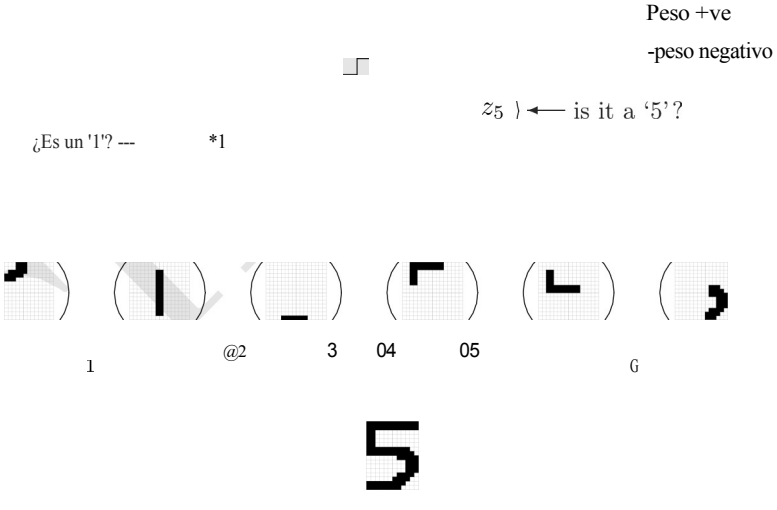


De hecho, podríamos argumentar plausiblemente que cada '1' debería contener un ct 82 y 3 y, cada '5' debería contener un d , 4 8s 8 y quizás un poco de a't. Hemos utilizado deliberadamente la notación a que utilizamos anteriormente para las coordenadas de la característica transformar 4'. Estas formas básicas son *rasgos* de la entrada y, por ejemplo, nos gustaría que e fuera grande (cercano a 1) si su rasgo correspondiente está en la imagen de entrada y pequeño (cercano a -1) en caso contrario.

Ejercicio 7%U

La forma básica 03 está tanto en el '1' como en el '5'. ¿Qué otros dígitos esperas que contengan cada forma básica 0i 06 Cómo seleccionarías formas básicas adicionales si quisieras distinguir entre todos los dígitos. (¿Qué propiedades deben satisfacer las formas básicas útiles?)

Podemos construir un clasificador para '1' frente a '5' a partir de estas formas básicas. ¿Recuerdas cómo, al principio del capítulo, construimos una función booleana compleja a partir de las funciones "básicas" Y y O? Vamos a imitar ese proceso aquí. La función compleja que estamos construyendo es el clasificador de dígitos y las funciones básicas son nuestras características. Supongamos, por ahora, que tenemos funciones de características ei que calculan la presencia (-I-1) o ausencia (-1) de la característica correspondiente. Observa detenidamente la siguiente red y ejecútala desde la entrada hasta la salida.



Ignorando detalles como los valores exactos de los pesos, el nodo zt responde a la pregunta "¿es la imagen un '1'?" y de forma similar el nodo -s responde "¿es la imagen un '5'?". Veamos por qué. Si han hecho su trabajo correctamente cuando introducimos un '1', ct 82 83 computa +1, y 4 " 6 computa -1. Combinando e , ... d'b con los signos de los pesos en las aristas salientes, todas las entradas a c serán positivas, por lo que zt da -1-1; todas menos una de las entradas a -s son negativas, por lo que zb da -1. Un análisis similar es válido si introducimos el "5". El resultado final

combina c y zb en la salida final. En este punto, es útil rellenar todos los espacios en blanco con un ejercicio.

Ejercicio 7\18

Dado que la entrada x es una imagen, es conveniente representarla como una matriz tz'jt de sus píxeles que son negros (z'j 1) o blancos (z'j 0). La forma básica *&k* identifica un conjunto de estos píxeles que son negros.

(a) Demuestre que (eature *&k* puede ser calculada por el nodo de la red neuronal

$$\phi_k(\mathbf{x}) = \tanh\left(w_0 + \sum_{ij} w_{ij} x_{ij}\right)$$

(b) ¿Cuáles son las entradas del nodo de la red neuronal?

(c) ¿Qué valores eliges para los pesos? [*Sugerencia: considere por separado los* pesos *de los píxeles z;) C &# y los z;) @ &#.)*

(d) ¿Cómo elegirías vo? (No todos los dígitos se escriben igual, por lo que es posible que una forma básica no siempre se represente exactamente en la imagen).

(e) Dibuja la red final, rellenando todos los detalles que puedas.

Se habrá dado cuenta de que la salida de zj es todo lo que necesitamos para resolver nuestro problema. Este no sería el caso si estuviéramos resolviendo el problema multiclase completo con los nodos z3, ... , zg correspondientes a los diez dígitos. , zg correspondientes a los diez dígitos. Además, hemos resuelto nuestro problema con relativa facilidad, ya que nuestra red "profunda" sólo tiene dos capas ocultas. En un problema más complejo, como el reconocimiento de caras, el proceso empezaría igual que aquí, con formas básicas. En el siguiente nivel, constituiríamos formas más complicadas a partir de las formas básicas, pero aún no habríamos llegado a casa. Estas formas más complicadas constituirían formas aún más complicadas hasta que por fin tuviéramos objetos realistas como ojos, boca, orejas, etc. Habría una jerarquía de características "básicas" hasta que resolviéramos nuestro problema al final.

Y ahora, el remate y el quid de la historia. Primero el remate. Enfoque la red que hemos construido y observe lo que hacen las distintas capas. La primera capa construye una representación de bajo nivel de formas básicas; la siguiente capa construye una representación de nivel superior a partir de estas formas básicas. A medida que subimos capas, obtenemos representaciones más complejas a partir de partes más sencillas de la capa anterior: una descomposición "inteligente" del problema, que empieza por lo sencillo y se va haciendo más complejo, hasta que finalmente se resuelve el problema. Esta es la promesa de la red profunda, que proporciona una visión similar a la humana de cómo se está resolviendo el problema basándose en una jerarquía de representaciones más complejas de la entrada. Si bien con una sola capa oculta podríamos alcanzar una solución de precisión similar, no obtendríamos esa percepción. El panorama es halagüeño para nuestro problema de reconocimiento

intuitivo de dígitos, pero aquí está el quid de la cuestión: para un problema de aprendizaje complejo, ¿cómo automatizamos todo esto en un algoritmo informático?

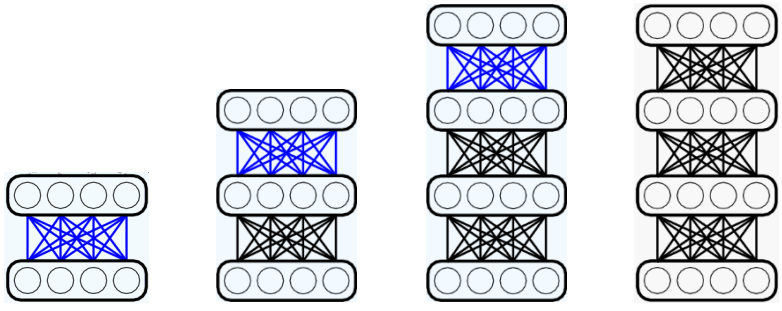


Figura 7.6: Algoritmo de aprendizaje profundo Greedy. (a) Se aprenden los pesos de la primera capa. (b) Se fija la primera capa y se aprenden los pesos de la segunda. (c) Se fijan las dos primeras capas y se aprenden los pesos de la tercera. (d) Los pesos aprendidos pueden utilizarse como punto de partida para ajustar toda la red.

7.6.1 Algoritmo codicioso de aprendizaje profundo

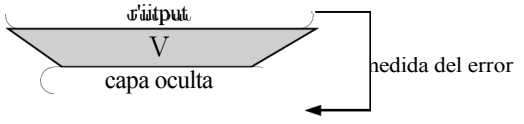
Históricamente, las redes neuronales poco profundas (de una sola capa oculta) han prevalecido sobre las redes profundas porque son difíciles de entrenar, presentan muchos mínimos locales y, en relación con el número de parámetros de caída, tienen una gran tendencia a sobreajustarse (la composición de no linealidades suele ser mucho más potente que una combinación lineal de no linealidades). Recientemente, algunas heurísticas sencillas han demostrado empíricamente un buen rendimiento y han devuelto el protagonismo a las redes profundas. De hecho, el mejor algoritmo actual para el reconocimiento de dígitos es una red neuronal profunda entrenada con dichas heurísticas.

La heurística codiciosa tiene una forma general. Se aprenden los pesos de la primera capa W ') y se fijan.'° La salida de la primera capa oculta es una transformación no lineal de las entradas xq -+ xq '. Estas salidas x$" se utilizan para entrenar los pesos de la segunda capa W^2 *), manteniendo fijos los pesos de la primera capa.* Este
es la esencia del algoritmo codicioso, elegir "codiciosamente" los pesos de la primera capa,
fijarlos, y luego pasar a los pesos de la segunda capa. Se ignora la posibilidad de que puedan existir mejores ponderaciones de la primera capa si se tiene en cuenta lo que hace la segunda capa. El proceso continúa con las salidas $x(^2)$ utilizadas para aprender las ponderaciones W °), y así sucesivamente.

'"Recordemos que utilizamos el superíndice (-)) para denotar la capa fi.

Algoritmo Greedy de aprendizaje profundo:

1. para I = 1, ... , ñ hacer , ñ do
2: W) W '*) son dadas de iteraciones anteriores. 3: Calcular las salidas x$ de la capa é - 1 para n = 1, ... , N.
4: Utilizar (xq " } para aprender los pesos W' entrenando una red neuronal simple de capa oculta. (W) W '*) son fijas).



$$\left(\mathbf{x}_n^{(\ell-1)}, \quad n = 1, \ldots, N \right)$$

Tenemos que aclarar el paso 4 del algoritmo. Se aprenden los pesos W(') y V, aunque V no es necesario en el algoritmo. Para aprender los pesos, minimizamos un error (que dependerá de la salida de la red), y ese error aún no está definido. Para definir el error, primero debemos definir la salida y luego cómo calcular el error a partir de la salida.

Autocodificador no supervisado. Un enfoque consiste en tomar en serio la noción de que la capa oculta ofrece una representación de alto nivel de las entradas. Es decir, deberíamos ser capaces de reconstruir todos los aspectos importantes de la entrada a partir de la salida de la capa oculta. Una prueba natural es reconstruir la propia entrada: la salida será xp , una predicción de la entrada xp; y, el error es la diferencia entre las dos. Por ejemplo, utilizando el error al cuadrado,

$$ep - | xp - xp ||^2 .$$

Cuando todo está dicho y hecho, obtenemos los pesos sin usar los objetivos pp y la capa oculta da una codificación de las entradas, de ahí el nombre de autocodificador no supervisado. Esto recuerda a la red de funciones de base radial del capítulo 6, en la que utilizamos una técnica no supervisada para aprender los centros de las funciones de base, lo que proporcionó un conjunto representativo de entradas como centros. Aquí, vamos un paso más allá y diseccionamos el propio espacio de entrada en trozos representativos del problema de aprendizaje. Al final, los objetivos tienen que volver a aparecer (normalmente en la capa de salida).

Red profunda supervisada. El enfoque anterior se adhiere al objetivo filosófico de que las capas ocultas proporcionen una representación jerárquica "inteligente" de las entradas. Un enfoque más directo consiste en entrenar la red de dos capas en los objetivos. En este caso, la salida es el objetivo predicho y la

medida de error ep (pp, p) se calcularía de la forma habitual (por ejemplo, error al cuadrado, error de entropía cruzada, etc.).

En la práctica, no hay un veredicto sobre qué método es mejor. El campo de los autocodificadores no supervisados está ligeramente más concurrido que el de los supervisados. Pruebe los dos y vea qué funciona para su problema, suele ser lo mejor. Una vez que tenga su medida de error, simplemente busque en su caja de herramientas de optimización y minimice el error utilizando su método favorito (descenso de gradiente, descenso de gradiente estocástico, descenso de gradiente conjugado, ...).). Una táctica común es utilizar primero el autocodificador no supervisado para establecer los pesos y luego afinar toda la red utilizando el aprendizaje supervisado. La idea es que el paso no supervisado te lleve al mínimo local correcto de la red completa. Pero, independientemente del bando al que pertenezcas, tienes que elegir la arquitectura de la red profunda (el número de capas ocultas y su tamaño), y para eso no hay ninguna poción mágica. Tendrás que recurrir a viejos trucos como la validación, o a una comprensión profunda del problema (nuestra red hecha a mano para la tarea "1" frente a "5" sugiere una red profunda con seis nodos ocultos en la primera capa oculta y dos en la segunda).

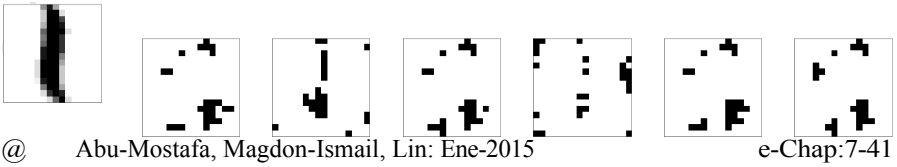
Ejercicio 7%19

Anteriormente, para nuestro problema de los dígitos, utilizamos la simetría y la intensidad. ¿Cómo se relacionan estas características con las redes profundas? ¿Siguen siendo necesarias?

Ejemplo T.S. Aprendizaje profundo para el reconocimiento de dígitos. Volvamos al problema de clasificación de dígitos '1' frente a '5' utilizando una arquitectura de red profunda

$$d('),\ d*'),\ d^{2)},\ \mathrm{d}(^{3})]\qquad \mathrm{t256,\ 6,\ 2,\ I}].$$

(La misma arquitectura que construimos antes a mano, con 16 x 16 píxeles de entrada y 1 de salida). Utilizaremos el descenso de gradiente para entrenar las redes de dos capas en el algoritmo codicioso. En el problema 7.7 se da una forma matricial conveniente para el gradiente de la red de dos capas. Para el codificador automático no supervisado, la salida objetivo es la matriz de entrada X. Para la red profunda supervisada, la salida objetivo es sólo el vector objetivo y. Utilizamos el enfoque supervisado con 1.000.000 iteraciones de descenso de gradiente para cada paso greedy supervisado utilizando una muestra de 1.500 ejemplos de los datos de dígitos. A continuación se muestra lo que aprendieron las 6 unidades ocultas de la primera capa oculta. Para cada nodo oculto de la primera capa oculta, mostramos los píxeles correspondientes a los 20 pesos entrantes más altos.



61 02 63 04 65 BG

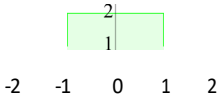
Los datos reales no son tan limpios como nuestros análisis idealizados. No se sorprenda. Sin embargo, podemos discernir que az ha seleccionado los píxeles (formas) en el típico "1" que es poco probable que estén en un típico "5". Las demás características parecen centrarse en el "5" y, en cierta medida, coinciden con las que hemos construido a mano. No vamos a insistir en si la representación refleja la intuición humana.
alcance. Lo importante es que este resultado es automático y depende exclusivamente de los datos (aparte de nuestra elección de la arquitectura de red); lo que importa es el rendimiento fuera de la muestra. Para las distintas arquitecturas, realizamos más de 1.000 experimentos de validación seleccionando 500 puntos de entrenamiento aleatorios cada vez y los datos restantes como conjunto de pruebas.

Arquitectura de redes profundas	A;p	prue ba
t256,3,2, I]	0	0.170%
t256,6,2, I]	0	0.187%
t256, 12,2, 1/	0	0,187Po
t256,24,2, 1/	0	0,183Po

en US £tlways cero porque hay muchos parámetros, incluso con sólo 3 unidades ocultas en la primera capa oculta. Esto huele a sobreajuste. Pero el resultado de la prueba es impresionante, con un 99,8% de precisión, que es lo único que nos importa. Nuestras características de simetría e intensidad construidas a mano eran buenas, pero no tanto. O

7.7 Problemas

Problema 7.1 Implementa la siguiente función de decisión utilizando un perceptrón de 3 capas.



Problema 7.2 Un conjunto o(*M* hiperplanos generalmente dividirá el espacio en algún número o(regiones. Cada punto de @' puede etiquetarse con un vector de *M* dimensiones que determina en qué lado de cada plano se encuentra. Así, por ejemplo, si *M* -- 3, entonces un punto con un vector (-1, -|-1, -|-1) está en el lado -1 del primer hiperplano, y en el lado -{-1 del segundo y tercer hiperplanos. Una región se define como el conjunto de puntos con la misma etiqueta.

(a) Demuestra que las regiones con la misma etiqueta son convexas.

(b) Demostrar que *M* hiperplanos pueden crear a lo sumo 2" regiones distintas.

(c) [difícil] ¿Cuál es el número máximo de regiones creadas por *M* hiperplanos en *d* dimensiones?

{Respuesta: $\sum_{z=d}^{d}$ *) .)*

{Pista: Utilice la inducción y deje que B(M, d) sea el número de regiones creadas por M d - \) -en el espacio d. Consideremos ahora la adición del (M \) hiperplano. Demostrar que este hiperplano interseca a lo sumo B(M, d \) de las regionesB M, d) . Por cada región que cruza, añade exactamente una región, por lo que B(M \, d) B(M, d) B(M, d- \). (¿Te suena esta recurrencia?) Evalúa las condiciones de contorno: B(M, \) y B(\, d), y proceder a partir de ahí. Para ver que el hiperplano M \ sólo interseca regiones B(M, d - \), argumentar de la siguiente manera. Tratar el hiperplano M como un espacio (d - \) -dimensional, y proyectar los M hiperplanos iniciales en este espacio para obtener M hiperplanos en un espacio d - \) -dimensional. -d). Estos M hiperplanos pueden crear como máximo B(M, d \) regiones en este espacio. Argumentemos que esto significa que el hiperplano M \ sólo interseca a lo sumoB M, d \) de las regiones creadas por los hiperplanos M en el espacio d).

Problema 7.3Supongamos que una función objetivo / (para clasificación) está representada por un número de hiperplanos, donde las diferentes regiones definidas por los hiperplanos (véase el Problema 7.2) podrían clasificarse 41 o -1, como en los ejemplos bidimensionales que hemos considerado en el texto. Sean los hiperplanos 1' 2 ' … , /rg/ , donde lip (x) sign(w - x) . Consideremos todas las regiones que se clasifican -|-1, y que una de tales regiones sea r". Sea c (c , cz, ... , c) la etiqueta de cualquier punto de la región (todos los puntos de una región dada tienen la misma etiqueta); la etiqueta eg II indica en qué lado o(ilp se encuentra el punto. Defina el término AND correspondiente a la región r mediante

/, /' /r2² ... /r * , donde /r - $\begin{cases} h_m & \text{if } c_m = +1, \\ h_m & \text{if } c_m = -1. \end{cases}$

Demuestre que / /" -|- fr, -|- -|- fr , donde r1, ... , rd son todas las regiones positivas. (Usamos multiplicación para los operadores AN D y suma para los operadores OR).

Problema 7.4 Remitiéndonos al problema 7 .3, cualquier función objetivo que pueda descomponerse en hiperplanos ñ , ... , ñ+/ puede representarse por , ñ+/ puede representarse por "4 fr, -|- 4 fr , donde hay *k* regiones positivas.

Cuál es la estructura del perceptrón de 3 capas (número de unidades ocultas en cada capa) que implementará esta (unción, demostrando el siguiente teorema:

Teorema. Cualquier (unción de decisión cuyas regiones II se definen en términos de las regiones creadas por un conjunto de hiperplanos puede ser implementada por un perceptrón de 3 capas.

Problema 7.5 [Difícil] Enuncie y demuestre una versión de un *teorema de Aproxi- mación Universal:*

Teorema. Cualquier función objetivo / ((o clasificación) definida en \0, \/ ^{d}, cuyas superficies límite de clasificación son suaves, puede ser aproximada arbitrariamente por un perceptrón de 3 capas.

{Pista: Descomponer el hipercubo unitario en c-hypercubes (de ellos), El volumen de estos €-hypercubes que interseca los límites de clasificación debe tender a cero (¿por qué? - utilizar suavidad). Así, la función que toma el valor de f en cualquier "-hipercubo que no interseca la frontera y un valor arbitrario en estos c-hypercubes frontera se aproximará a f arbitrariamente cerca, como " 0 .)

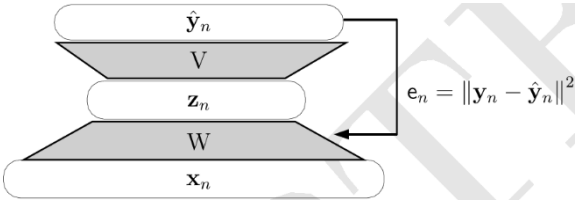
Problema 7.6 La aproximación de ditrerencia finita para obtener el gradiente se basa en la siguiente fórmula de cálculo:

$$\frac{\partial h}{\partial w_{ij}^{(\ell)}} - \frac{h(w_{ij}^{(\ell)} + \epsilon) - h(w_{ij}^{(\ell)} - \epsilon)}{2c} + O(\epsilon^2),$$

donde //(mty 4 E) to denota el valor de (unción cuando todos los pesos se mantienen en sus valores en w excepto (o el peso vty), que es perturbado por r. Para obtener el gradiente, necesitamos la derivada parcial con respecto a cada peso.

Demuestre que la complejidad computacional de obtener todas estas derivadas parciales es O(W^2). */Pista: hay que hacer dos propagaciones hacia delante para cada peso).*

Problema 7.7 Considere la siguiente red de 2 capas, con el *vector de* salida y. Esta es la red de dos capas utilizada para el algoritmo de red profunda codiciosa.



Recoge los vectores de entrada xn (junto con una columna de unos) como filas de la matriz de datos de entrada X, y de forma similar forma Z a partir de z . Las matrices objetivoY e Y se (ormed (rom yu e yu respectivamente. Supongamos que el nodo de salida es lineal y que la activación de la capa oculta es 8( ).

(a) Demuestre que el error dentro de la muestra es

$$\text{" } = /\text{tace } Y-Y)Y-Y) \ ,$$

donde

Z 1,8(XKQ y

X es N z d 1)
W es d 1) x d''
Z es N z d'' $I)$
V -es $(d'' + I)$ x dim(y) Y, Y
son N x dim(y)

(Es conveniente descomponer V en su primera fila Vo correspondiente a los sesgos y sus restantes filas V1: 1 es el vector *N* x 1 de unos).

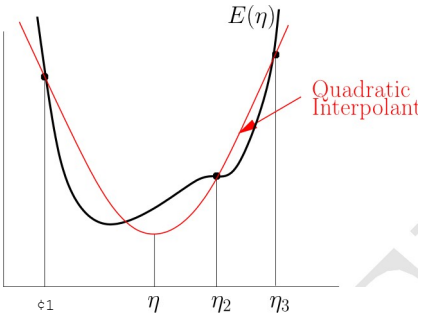
(b) derivar las matrices de gradiente:

$\frac{\partial E_{\text{in}}}{\partial \mathrm{V}}$ 2Z "ZV - 2Z "Y

$\frac{\partial E_{\text{in}}}{\partial \mathrm{W}}$, 2X'/(Xw)o(qXw)V,v;+1v,v,-Y,)

donde @ significa multiplicación por elementos. Algunas de las derivadas matriciales de funciones que implican la traza del apéndice pueden ser útiles.

Problema 7.8 Interpolación cuadrática para la búsqueda de líneas Supongamos que se ha encontrado un arreglo en U, como se ilustra a continuación.



En lugar de utilizar la bisección para construir el punto ¿, la interpolación cuadrática ajusta una curva cuadrática C(q) oq^2 bq c a los tres puntos y utiliza el mínimo de esta interpolación cuadrática como q.

(a) Demuestre que el mínimo o(la interpolante cuadrática para un arreglo en U está dentro del intervalo tqi , qst.

(b) Sea ei A (//i), ez C(Vz), e3 A (//3). Obtener la función cuadrática que interpola los tres puntos ((//i , ei), (vz, e2), (//s, e3)}. Demostrar que el mínimo de esta interpolante cuadrática viene dado por:

$$\_1 \lceil (e_1-e_2)(\eta_1^2-\eta_3^2)-(e_1-e_3)(\eta_1^2-\eta_2^2)\_$$
$$2 \lfloor (e_1-e_2)(\eta_1-\eta_3)-(e_1-e_3)(\eta_1-\eta_2)$$

{Pista: e – $a\eta_1^2+b\eta_1+c,\ e_2=a\eta_2^2+b\eta_2+c,\ e_3=a\eta_3^2+b\eta_3+c$. Solve para a, b, c y el mínimo de la cuadrática viene dado por ij -- -b/2o.)

(c) Dependiendo de si C(¿) es menor que A(v2), y de si está a la izquierda o a la derecha de v2, hay 4 casos.
En cada caso, ¿cuál es la disposición en U más pequeña?

(d) ¿Y si - Vz un caso degenerado?

Nota: en general, las interpolaciones cuadráticas convergen muy rápidamente a una q localmente óptima. En la práctica, 4 iteraciones son más que suficientes.

Problema T.9 [Convergencia de la minimización de Monte-Carlo] Supongamos que el mínimo global w" está en el cubo unitario y la superficie de error es cuadrática cerca de w* Entonces, cerca de w*,

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}\text{(w-w')'H(w-w')}$$

donde el hessianoH es definido positivo y simétrico.

(a) Si se muestrea uniformemente w en el cubo unitario, demuestre que

$$P[E \leq E(\mathbf{w}^*) + \epsilon] = \int_{\text{x*Hx<2e}} d^d\mathbf{x} = \frac{d\,2)}{\text{morir}}$$

donde *5q(r)* es el volumen de la esfera d-dimensional de radio r,

$$S_d(r) = \pi^{d/2} r^d / \Gamma(\tfrac{d}{2} + 1).$$

[Pistas: P cc(w") + c = *P* }(w - w")*H(w - w") '

Supongamos que la ortogono I motriz A *dragonoliza* H: A*HA - *ding* 52, ... , Id]. *Cambiar voriobles a* u = $\lambda_1^2\lambda_2^2 \cdots \lambda_d^2$.]let H =

(b) Suponga que muestrea *M* veces y elige las ponderaciones con error mínimo, wm;n. Demuestre que

$$P[E(\mathbf{w}_{\min}) > E(\mathbf{w}^*) + \epsilon] \approx \left(1 - \frac{1}{\pi d}\left(\mu\frac{\epsilon}{\sqrt{d}}\right)^d\right)^N,$$

donde // / y A es la media geométrica de los valores propios de H.

(Puede utilizar F(m4) "e-"2 .)

(c) Demuestre que i(*N* ( ) log , entonces con probabilidad al menos 1 - q, $\frac{1}{\eta}$

(Puede utilizar log(1 - a]a para smalla y *(rd)"d* 1.)

Problema 7.10 Para una red neuronal con al menos 1 capa oculta y transformaciones tank( ) en cada nodo que no sea de entrada, cuál es el gradiente (con respecto a los pesos) si todos los pesos se fijan en cero.

¿Es buena idea inicializar los pesos a cero?

Problema 7.11 [Tasa óptima de **aprendizaje**] Supongamos que estamos en las proximidades de un mínimo local, w", de la superficie de error, o que la superficie de error es cuadrática. La expresión de la función de error viene dada por

$$E(\mathbf{w}_t) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w}_t - \mathbf{w}^*)\mathbf{H}(\mathbf{w}_t - \mathbf{w}^*) \qquad (7.8)$$

de donde es fácil ver que el gradiente viene dado por g/ H (w/ - w"). Las actualizaciones de peso vienen dadas entonces por w'+1 w' - //H(w/ - w"), y restando w* de ambos lados, vemos que

$$\epsilon_{t+1} = (\mathrm{I} - \eta\mathrm{H})\epsilon_t \qquad (7.9)$$

ComoH es simétrico, se puede formar una base ortonormal con sus vectores propios. Proyectando c, y c, i sobre esta base, vemos que en esta base, cada

componente

y dejando que c(a) sea el componente a*" en esta base, vemos que

-.' '-m',".(" ('.'o'

por lo que vemos que cada componente exhibe convergencia lineal con su propia co-eficiencia de convergencia *kg* -- 1 - qJo. El peor componente dominará la convergencia por lo que nos interesa elegir q de forma que se minimice el *ku de* mayor magnitud. Dado queH es definida positiva, todos los Jo son positivos, por lo que es fácil ver que uno debe elegir q para que 1 - qJ ,n 1 - A y 1 - A, o uno debe elegir. Resolviendo para la q óptima, se encuentra que

$$\eta_{opt} = \frac{2}{\lambda_{min} + \lambda_{max}} \qquad k_{opt} = \frac{\lambda_{max}}{\lambda_{min}} = \frac{1+c}{1-c} \tag{7.11}$$

donde c ,n/J es el número de condición de H, y es una medida importante de la estabilidad de H. Cuando c 0, se suele decir que H está mal condicionado. Entre otras cosas, esto afecta a la capacidad de calcular numéricamente la inversa de H.

Problema 7 .12 [Con una tasa de aprendizaje variable, supongamos que //' -+ 0 que satisface 1///' co y 1///2 co, por ejemplo se podría

elija q, 1/(I -]- 1). Demostrar que el descenso de gradiente convergerá a un mínimo local.

Problema 7.13 [Aproximación de ditrerencia finita al hessiano]

(a) Consideremos la función C(v , z). Demostrar que la aproximación por diferencias finitas a las derivadas parciales de segundo orden vienen dadas por

$$\frac{\partial^2 E}{\partial w_1^2} \quad *(*,+2*,*e)**(*,-2\ +2)-2*(*„*a) \quad {}_{4\ 2}$$

$$\frac{\partial^2 E}{\partial w_2^2} \quad H(u\ "i2-F2ñ)+H(u\ "+2-2h)-2H(u\ "u\ ) \quad {}_{4\ 2}$$

$$\frac{\partial^2 E}{\partial w_2^2} \quad \frac{E(w_1+h,w_2+h)+E(w_1-h,w_2-h)-E(w_1+h,w_2-h)-E(w_1-h,w_2+h)}{4h^2}$$

(b) Proporcione un algoritmo para calcular la aproximación por diferencias finitas a la matriz hessiana para Min(w), el error en la muestra para una red neuronal multicapa con pesos w - tW*!), ... , W*')].

(c) Calcule el tiempo de ejecución asintótica de su algoritmo en función del número de pesos de su red y del número de puntos de datos.

Problema 7.14 Supongamos que damos un paso fijo en alguna dirección, nos preguntamos cuál es la dirección óptima (o este paso fijo suponiendo que el modelo cuadrático para la superficie de error es exacto:

$$E_{\text{in}}(\mathbf{w}_t + \delta\mathbf{w}) = \text{z}_{\text{in}}(\mathbf{w}_t) + \mathbf{g}_t^{\text{T}}\Delta\mathbf{w} + \frac{1}{2}\Delta\mathbf{w}^{\text{T}}\text{H}_t\Delta\mathbf{w}.$$

Así que queremos minimizar Am(Aw) con respecto a Aw sujeto a la restricción de que el tamaño del paso es q, e s decir, que Aw Awq²

(a) Demostrar que el Lagrangiano para este problema de minimización restringida es:

$$\mathcal{L} = E_{\text{in}}(\mathbf{w}_t) + \mathbf{g}_t^{\mathrm{T}}\Delta\mathbf{w} + \frac{1}{2}\Delta\mathbf{w}^{\mathrm{T}}(\mathrm{H}, \quad 2\mathrm{oI})\ \mathrm{fiw}, \tag{7.12}$$

dondea es el multiplicador de Lagrange.

(b) Resuelve (o Aw anda y demuestra que satisfacen las dos ecuaciones:

$$\mathrm{Aw} = -(\mathrm{H}/ + 2\mathrm{nI})\ 'g''$$
$$\Delta\mathbf{w}^{\mathrm{T}}\Delta\mathbf{w} \quad n^2.$$

(c) Demuestre quea satisface la ecuación implícita:

$$\frac{1}{2}\ {}^{2}(\mathrm{Aw}'\mathrm{g}, + \mathrm{Aw}'\mathrm{H}, \mathrm{Aw}).$$

Argumentar que el segundo término es 8(1) y el primero es O(| g, | /q). Por tanto, a es grande para un tamaño de paso p pequeño.

(d) Supongamos quea es grande. Demostrar que, a orden principal e n $\frac{1}{\eta}$

2/

Por lo tantoa es grande, coherente con la expansión de Aw a orden principal en . *{Pista: expandir &w al orden principal en .)*

(e) Utilizando (d), demuestre que Aw - tH, ^{1} g/. + ",t lt

Problema 7 .15 La aproximación hessiana del producto exterior es H $_{1}$ g "gj. SeaHr $_{1}$ g gj sea la suma parcial a *k,* y sea HU su inversa.

(a) Demuestra que HUDH¿ $$!\frac{\mathrm{H}_k^{-1}\mathbf{g}_{k+1}\mathbf{g}_{k+1}^{\mathrm{T}}\mathrm{H}_k^{-1}}{1 + \mathbf{g}_{k+1}^{\mathrm{T}}\mathrm{H}_k^{-1}\mathbf{g}_{k+1}}$$

{Pistas: Hr-F1 = H\ g\- Ok -F1* *Y,* (A + zz")-' = A-* - $\frac{\mathtt{A-'xx'A-'}}{\mathtt{1+z*A-'z}}$]s

(b) Utilice la parte (a) para dar un algoritmo O(NW²) para calcular HQ , el mismo tiempo que se tarda en calcular H. (T es el número de dimensiones en g).

Nota: normalmente, este algoritmo se inicializa con Ho cl para algún c pequeño. Así que el algoritmo realmente calcula (H + cl) -*; los resultados no son muy sensibles a la elección o(E, siempre que E sea pequeño.

Problema 7.16 En el texto, hemos calculado un límite superior en la dimensión VC o(el perceptrón de 2 capas es *dv -- O md \og md))* donde la es el número de unidades ocultas en la capa oculta. Demostrar que este límite es esencialmente apretado mostrando que *d c -- U(md).* Para ello, mostrar que es posible encontrar *md* puntos que pueden ser destrozados cuando la es incluso como (ollows.

Consideremos cualquier conjunto de *N* puntos *xi , ... , xN* en posición general con *N -- md. N* puntos en *d* dimensiones están en posición general i(ningún subconjunto de d4 1 puntos yace en un hiperplano de *d* - 1 dimensiones. Ahora, considere cualquier dicotomía en estos puntos con *r ol* los puntos clasificados 41. Sin pérdida de generalidad, vuelva a etiquetar los puntos de modo que x1, ... , xr sean 41.

(a) Demuestre que, sin pérdida de generalidad, puede suponer que r N *!2. Por* lo tanto, para el resto del problema puede suponer que r N/2.

(b) Dividir los puntos positivos en grupos de tamaño *d.* El último grupo puede tener menos de *d* puntos. Demostrar que el número de grupos es como máximo '. Etiquetar estos grupos 6' para I =: 1 ... g N *!2.*

(c) Demostrar que (o cualquier subconjunto de *k* puntos con *k d,* existe un hiperplano que contiene esos puntos y no otros.

(d) Por la parte anterior, sea w" 6, es el hiperplano que pasa por los puntos del grupo @', y no contiene ningún otro. Entonces

$$\mathbf{w}_i^{\mathrm{T}}\mathbf{x}_n + b_i = 0$$

si y enly si x" E @ . Demuestre que es posible encontrar ñ suficientemente pequeño para que para xu E @"

|wtx "4 6 | < //,

y para xu g @/

$$|\mathbf{w}_i^{\mathrm{T}}\mathbf{x}_n + b_i| > h.$$

(e) Demuestre que para xn E O/,

$$\mathrm{sign}(\mathbf{w}_i^{\mathrm{T}}\mathbf{x}_n + b_i + h) + \mathrm{sign}(-\mathbf{w}_i^{\mathrm{T}}\mathbf{x}_n - b_i + h) = 2,$$

y para xn g @/

sign(cera +d; S) sign(-cera -d,+h)=0

(f) Utiliza los resultados obtenidos hasta ahora para construir un MLP de 2 capas con 2r unidades ocultas que implemente la dicotomía (que era arbitraria). Completa el argumento para demostrar que *d c md.*