

Aprendizaje en redes neuronales

Víctor Mijangos

Facultad de Ingeniería



UNAM

Problema de aprendizaje como problema de optimización



Problema de aprendizaje

Las redes neuronales, como otros problemas de aprendizaje, pueden verse como una función $\phi : X \rightarrow Y$, tal que:

$$\phi(x) = \hat{y}$$

Tal que \hat{y} representa una predicción que busca ser cercana a una categoría de supervización y .

El **problema de aprendizaje** se puede plantear como:

El proceso de aprendizaje consiste de escoger una función apropiada ϕ^ de un conjunto de funciones $\{\phi : X \rightarrow Y\}$.*



Discriminativo vs. Generativo

Las funciones de las redes neuronales pueden adquirir diferentes formas según si se asume una distribución de los datos o si no.

Asimismo, existe una distinción importante entre la forma de estimar la probabilidad por parte de la red:

- ▶ Modelo discriminativo (FeedForwards):

$$\phi(x) = p(Y = y|x)$$

- ▶ Modelo generativo (Variational AutoEncoders):

$$\phi(x) = p(Y = y, x) = p(x|Y = y)p(Y = y)$$

Cada forma de estimar resulta en un tipo diferente de algoritmo. Asimismo, pueden estimarse métodos híbridos [1].



Parámetros

Las funciones que nos interesan son aquellas que dependen de un conjunto de parámetros (pesos). Es decir, aquellas funciones:

$$\phi : X \times \Theta \rightarrow Y$$

donde $\Theta = \{\theta\}$ es un conjunto de parámetros.

En las redes neuronales, estos parámetros representan los pesos de la red:

$$\theta = \{W^{(1)}, W^{(2)}, \dots, W^{(K)}, b^{(1)}, b^{(2)}, \dots, b^{(K)}\}$$

El **problema de aprendizaje** se reduce a encontrar el conjunto de parámetros $\theta \in \Theta$ que nos den la función adecuada $\phi(x; \theta) = \hat{y} \approx y$.



Búsqueda de parámetros

En el caso del perceptrón, los parámetros se obtienen por una regla simple (algoritmo del perceptrón).

Requerimos de una forma general de obtener los parámetros sin importar el tipo y la arquitectura de una red.

Determinado un tipo y arquitectura neuronal, se define un **espacio de parámetros** con la propiedad de ser **continuo**. Denotaremos un conjunto de parámetros como:

$$\theta \in \Theta$$



Emmpirical risk minimization

Para encontrar el conjunto de parámetros óptimo $\hat{\theta}$ pasamos del problema de aprendizaje a un problema de **optimización** [3].

Definición (Función de riesgo)

A una función $R : \Theta \rightarrow \mathbb{R}$, donde Θ es un espacio de soluciones, se le denomina *función de riesgo* (función objetivo/costo).

Hablamos del problema de **minimización del riesgo empírico** cuando, a partir de datos de entrenamiento, buscamos:

$$\hat{\theta} = \arg \min_{\theta \in \Theta} R(\theta)$$



Funciones de riesgo

La **función de riesgo** puede definirse de cualquier forma que nos ayude a solucionar el problema (asumimos $Y \sim q$).

Para las redes neuronales es común utilizar alguna función del siguiente tipo (ϕ distribución estimada por la red):

- Divergencia KL (GANs, Variational AutoEncoders):

$$R(\theta) = \mathbb{E}_q \left[\ln \frac{q(x)}{\phi(x; \theta)} \right]$$

- Entropía cruzada (FeedForwards, Recurrentes):

$$R(\theta) = \mathbb{E}_q [\ln \phi(x; \theta)]$$

- Error cuadrático medio (AutoEncoders, Denoising AutoEncoders):

$$R(\theta) = \frac{1}{2} \mathbb{E} [(y - \phi(x; \theta))^2]$$



Función de riesgo en redes FeedForward

En las redes FeedForward, la función $\phi(x) = \phi(x; \theta)$ constituye la red (forward).

Este tipo de redes se aplican a problemas **supervisados**, donde se cuenta con un conjunto de entrenamiento:

$$\mathcal{S} = \{(x, y) : x \in \mathbb{R}^d, y \sim q\}$$

De tal forma que problema de aprendizaje se plantea como:

$$\arg \min_{\theta \in \Theta} R(\theta) = \arg \min_{\theta \in \Theta} \sum_x \sum_y y \ln \phi(x; \theta) \quad (1)$$



Gradiente descendiente



¿Cómo optimizar los parámetros?

Como muchos problemas de optimización, podemos encontrar el mínimo a partir del gradiente de la función. Buscamos:

$$\nabla_{\theta} R(\theta) = 0$$

Encontrar estos valores dependerá de las propiedades que tenga la función R .

Por ejemplo, el teorema de valores extremos nos dice que una función continua alcanza su mínimo (y máximo) en un intervalo cerrado.

Sin embargo, este no es el caso para muchas de las funciones de riesgo.

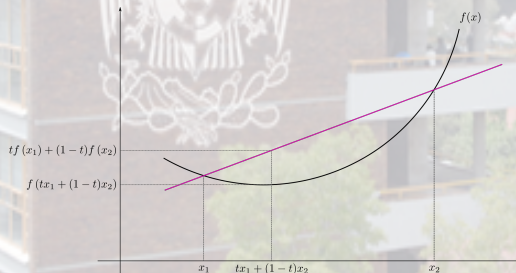


Convexidad

Una **función convexa** es aquella que cumple que para toda $t \in [0, 1]$:

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

Una función de este tipo siempre tendrá un mínimo.



Pero, de nuevo, las funciones de riesgo no siempre serán convexas.



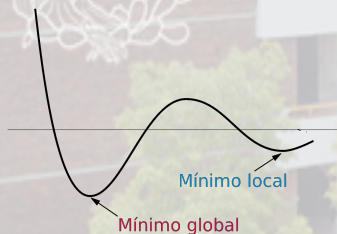
Mínimos globales y locales

El **mínimo global** es la $\hat{\theta}$ tal que $\forall \theta \in \Theta$:

$$R(\hat{\theta}) < R(\theta)$$

A esta solución se le llamará **mínimo global**.

Si la desigualdad se cumple sólo para una vecindad ($\{\theta : \|\theta - \hat{\theta}\| \leq r\}$), se le denomina **mínimo local**.



Gradiente descendiente

La derivada de la función $R(\theta)$ nos da información sobre el comportamiento de la función:

- ▶ Si $\nabla_{\theta}R(\theta)$ es positiva, la función asciende. Se debe disminuir el valor de θ :

$$\Delta\theta = -\nabla_{\theta}R(\theta) < 0$$

- ▶ Si $\nabla_{\theta}R(\theta)$ es negativa, la función descende. Se debe aumentar el valor de θ :

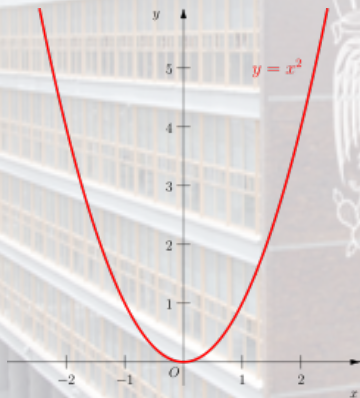
$$\Delta\theta = -\nabla_{\theta}R(\theta) > 0$$

La regla de actualización puede verse como:

$$\theta \leftarrow \theta - \nabla_{\theta}R(\theta)$$



Probelma de gradiente descendiente



- Podemos definir, una función $f(x) = x^2$
- Sabemos que $f'(x) = 2x$, por tanto, para minimizar la función:

$$x \leftarrow x - 2x$$

- Si inicializamos con $x = 1$, entonces:
 $x \leftarrow 1 - 2 \cdot 1 = -1$
- Aplicando de nuevo el método de GD, tendremos $x = 1$. El algoritmo nunca convergerá al mínimo.



Rango de aprendizaje

El problema anterior puede solucionarse modificando la regla de actualización $x \leftarrow x - \frac{1}{2}f'(x)$ (convergerá en el primer paso).

En general se toma un **rango de aprendizaje** $\eta \in \mathbb{R}$ para controlar la forma en que el GD desciende sobre la función.

El método de **gradiente descendiente** (GD) se define por la regla:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} R(\theta)$$

- ▶ Si η es muy grande, el algoritmo podría nunca converger.
- ▶ Si η es muy pequeño puede demorarse mucho en converger.



Algoritmo para GD

Input: Función $R : \Theta \rightarrow \mathbb{R}$; $T > 0$ número de iteraciones; η rango de aprendizaje.

Inicialización: Seleccionar $\theta^{(0)} \in \Theta$ aleatoriamente.

Inducción: En la iteración $t \in \{1, 2, \dots, T\}$ hacer:

Calcular $R(\theta^{(t)})$ y $\nabla_{\theta^{(t)}} R(\theta^{(t)})$.

Actualizar los valores con respecto a la regla:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \nabla_{\theta^{(t)}} R(\theta^{(t)})$$

Terminación: El algoritmo termina después de T iteraciones (o cuando el gradiente es 0).

Output: El valor $\hat{\theta}$ que minimiza la función R .



GD en el perceptrón

La función del perceptrón no es derivable, pero podemos cambiar esta función por la sigmoide:

$$\phi(x) = \sigma(wx)$$

Y utilizar como función de riesgo la entropía cruzada:

$$\begin{aligned} R(w) &= - \sum_x y \ln \phi(x) + (1 - y) \ln(1 - \phi(x)) \\ &= - \sum_x y \ln \sigma(wx) + (1 - y) \ln \sigma(-wx) \end{aligned}$$



GD en el perceptrón

Buscaremos las derivadas parciales para cada ejemplo x :

$$\frac{\partial R}{\partial w_i} = \frac{\partial y \ln \phi(x) + (1 - y) \ln(1 - \phi(x))}{\partial w_i} \quad (2)$$

$$= \frac{\partial y \ln \sigma(wx) + (1 - y) \ln \sigma(-wx)}{\partial w_i} \quad (3)$$

$$= (y - \phi(x))x_i \quad (4)$$

Así, la optimización para un perceptrón de este tipo es (en cada ejemplo):

$$w_i \leftarrow w_i - \eta(y - \phi(x))x_i$$



Variaciones del algoritmo de gradiente descendiente



Batch Gradient Descent

El algoritmo de GD actualiza los pesos a partir de la regla:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} R(\theta)$$

Donde el gradiente se obtiene de derivar la función de riesgo y depende de observar **todos los ejemplos**.

Es decir, en cada actualización de los pesos, se deben observar todos los ejemplos. A este tipo de actualización se le conoce como **(Batch) Gradient Descent**.

Sin embargo, este tipo de método de actualización presenta algunos problemas:

- ▶ Puede ser intratable si el conjunto de entrenamiento es muy grande (limitaciones de memoria).
- ▶ Si se desea actualizar *online*, se tendría que calcular el gradiente sobre todos los ejemplos y no sólo sobre los nuevos.



Stochastic Gradient Descent

Para solucionar estos problemas, el método de **Stochastic Gradient Descent** (SGD) realiza una sola actualización en cada paso.

Por cada iteración, SGD actualiza cada uno de los ejemplos (de forma aleatoria):

$$\theta \leftarrow \theta - \eta \nabla_{\theta} R(\theta; x, y)$$

Donde, $\nabla_{\theta} R(\theta; x, y)$ indica que la derivada se ha obtenido únicamente al observar el par de entrenamiento (x, y) .



Mini-batch Gradient Descent

El método de **Mini-batch Gradient Descent** considera mini-batches de tamaño n . Un mini-batch es un subconjunto aleatorio de pares de entrenamiento:

$$Batch_n = \{(x^{(i)}, y^{(i)}) : i = 1, \dots, n\} \subseteq X$$

El gradiente descendiente por mini-batches actualiza los pesos de la red cada vez que recorre uno de estos mini-batches:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} R(\theta; Batch_n)$$

Y realiza esta actualización hasta agotar el total de ejemplos.



Otros métodos de GD

Sin embargo, los métodos de (Batch) GD, SGD y el método por mini-batches presentan algunos problemas [2]:

1. La selección del rango de aprendizaje η es dificultosa, y requiere de exploración.
2. El rango de aprendizaje η es el mismo para todos los ejemplos, no considera el caso en que los datos sean dispersos.
3. Cuando la función de riesgo es no convexa, se tiene mayor posibilidad de caer en un mínimo local.

Un método típico de que se puede incorporar a la optimización por gradiente es la regularización, donde actualiza sobre una nueva función de riesgo:

$$R'(\theta) = R(\theta) + \gamma\Omega(\theta)$$



Adagrad

Existe una buena cantidad de métodos basados en GD [2]. Muchos de ellos se enfocan en la adaptación del rango de aprendizaje.

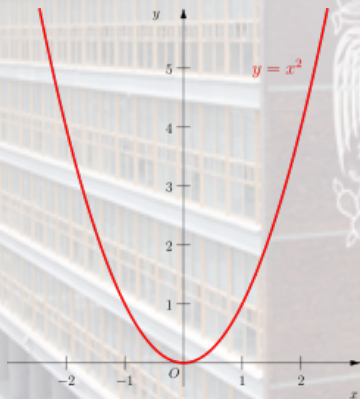
Un método común es **Adagrad** que usa un rango distinto para cada θ_i en cada época:

$$\theta_i \leftarrow \theta_i - \frac{\eta}{\sqrt{\mu_i} + \epsilon} \nabla_{\theta_i} R(\theta)$$

Donde $\mu_i \leftarrow \mu_i + (\nabla_{\theta_i} R(\theta))^2$ y, generalmente, $\epsilon = 1e - 8$.



Ejemplo de Adagrad



Retomemos la función $f(x) = x^2$ cuyo mínimo es 0. Y tomemos $\eta = 1$.

Sabemos que el GD **no converge** cuando $\eta = 1$. Pero la convergencia si se da con Adagrad.

- ▶ Inicialicemos con $x = 1$ como en el caso anterior. Y con $\mu = 0, \epsilon = 1e - 18$.
- ▶ En la primera iteración:

$$\mu = 0 + 2(1)^2 = 4$$

Por lo que

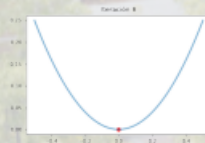
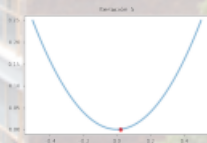
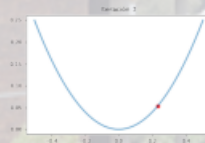
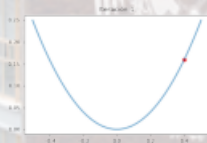
$$\frac{\eta}{\sqrt{4 + 1e - 8}} \approx \frac{1}{2}$$



Ejemplo de Adagrad

En el caso anterior, podemos ver que la convergencia se da prácticamente en la primera iteración.

Si la elección del valor inicial varia, la convergencia se seguirá asegurando. Por ejemplo, con $x = \frac{2}{5} = 0,4$:



Adam

Otro método común es **Adam**, determinado por:

$$\theta_i \leftarrow \theta_i - \frac{\eta}{\sqrt{\hat{\nu}} + \epsilon} \hat{m}$$

Tal que $(\beta_1, \beta_2 \in [0, 1])$:

$$\hat{m} = \frac{m}{1 - \beta_1}$$

$$\hat{\nu} = \frac{\nu}{1 - \beta_2}$$

Y los valores m y ν se actualizan en cada época por:

$$m \leftarrow \beta_1 m + (1 - \beta_1) \nabla_{\theta} R(\theta)$$

$$\nu \leftarrow \beta_2 \nu + (1 - \beta_2) [\nabla_{\theta} R(\theta)]^2$$



Adam y Adagrad

El método de Adam puede verse como una generalización del método de Adagrad. Tenemos que si: $\beta_1 = 0$ y $\beta_2 = \frac{1}{2}$
Entonces:

$$\begin{aligned} m &\leftarrow \nabla_{\theta} R(\theta) \\ \nu &\leftarrow \frac{1}{2}(\nu + (\nabla_{\theta} R(\theta))^2) \end{aligned}$$

Por lo que $\hat{m} = \nabla_{\theta} R(\theta)$ y $\hat{\nu} \leftarrow \hat{\nu} + (\nabla_{\theta} R(\theta))^2$, de donde la regla de actualización será:

$$\theta_i \leftarrow \theta_i - \frac{\eta}{\sqrt{\hat{\nu}} + \epsilon} \nabla_{\theta_i} R(\theta)$$



Backpropagation



Optimización en redes multicapa

Si bien la optimización con GD optimiza los pesos adecuadamente, surge el problema de tener redes con muchas capas.

En estos casos, la red puede verse como una composición de funciones:

$$p(Y = y|x) = \phi(h^{(K)}(\dots h^{(1)}(x)))$$

Así, para encontrar la derivada de la función de riesgo $R(\theta)$ se puede utilizar la regla de la cadena varias veces:

$$\nabla_{\theta} R(\theta) = \nabla_{\phi} R \nabla_{h^{(K)}} \phi \dots \nabla_{\theta_{ij1}} h^{(1)}$$

Ya que derivamos sobre los parámetros θ , también debe observarse que estos se encuentran a través de todas las capas.



Algoritmo de backpropagation: intuición

Aplicar la regla de la cadena para obtener las derivadas sobre todos los pesos de la red resulta muy complejo.

El algoritmo de **backpropagation** es capaz de obtener la derivada propagando la derivación desde la capa de salida hasta la de entrada.

Ejemplo: Considérese una red con una sola capa cuyo forward esté dado por:

- ▶ $a^{(1)} = W^{(1)}x$
- ▶ $h = \tanh(a)$
- ▶ $a^{(2)} = W^{(2)}h$
- ▶ $\phi(x) = \sigma(a^{(2)})$

Supóngase que la función de riesgo está dada por $R(\theta) = \frac{1}{2} \sum_x \sum_y (y - \phi(x))^2$



Algoritmo de backpropagation: intuición

En primer lugar, obtendremos las derivadas parciales de la función de riesgo sobre los parámetros. En este caso, el conjunto de parámetros es:

$$\theta = \{W^{(1)}, W^{(2)}\}$$

Podemos denotar $\theta_{i,j,k} = W_{ij}^{(k)}$; si aplicamos **SGD** derivaremos sólo sobre un ejemplo:

$$\begin{aligned}\frac{\partial R}{\partial W_{ij}^{(2)}} &= \frac{\partial}{\partial W_{ij}^{(2)}} \frac{1}{2} \sum_y (y - \phi(x))^2 \\ &= \sum_y \frac{\partial}{\partial W_{ij}^{(2)}} \frac{1}{2} (y - \phi(x))^2\end{aligned}$$

Lo cual nos dice que tenemos que derivar sobre cada una de las neuronas de salida de la red.



Algoritmo de backpropagation: intuición

Necesitamos observar las derivadas para cada una de las matrices de parámetros. Para la **capa de salida**, tenemos:

$$\begin{aligned}\frac{\partial R}{\partial W_{ij}^{(2)}} &= \frac{\partial R}{\partial \phi_i} \frac{\partial \phi_i}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial W_{ij}^{(2)}} \\ &= (\phi_i(x) - y_i) \sigma(a^{(2)}) (1 - \sigma(a^{(2)})) h_j\end{aligned}$$

Pues tenemos que: $\frac{\partial R}{\partial \phi} = \frac{\partial}{\partial \phi} \frac{1}{2} (y_i - \phi_i(x))^2 = -(y_i - \phi_i(x))$

Mientras que $\frac{\partial \phi_i}{\partial a_i^{(2)}}$ es la derivada del sigmoide; finalmente:

$$\frac{\partial a_i^{(2)}}{\partial W_{ij}^{(2)}} = \frac{\partial}{\partial W_{ij}^{(2)}} W_{i \cdot}^{(2)} h = \sum_k \frac{\partial}{\partial W_{ij}^{(2)}} W_{ik}^{(2)} h_k = h_j$$



Algoritmo de backpropagation: intuición

Para la **capa oculta** tenemos que obtener la derivada parcial:

$$\frac{\partial R}{\partial W_{ij}^{(1)}} = \left[\frac{\partial R}{\partial \phi} \frac{\partial \phi}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial h_i} \right] \left[\frac{\partial h_i}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial W_{ij}^{(1)}} \right]$$

La primera parte corresponde a la derivación:

$$\begin{aligned} \frac{\partial R}{\partial \phi} \frac{\partial \phi}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial h_i} &= \sum_q \frac{\partial R}{\partial \phi_q} \frac{\partial \phi_q}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial h_i} \\ &= \sum_q \frac{\partial R}{\partial \phi_q} \frac{\partial \phi_q}{\partial a^{(2)}} W_{iq}^{(2)} \\ &= \sum_q W_{iq}^{(2)} (\phi_q(x) - y_q) \sigma_q(a^{(2)}) (1 - \sigma_q(a^{(2)})) \end{aligned}$$



Algoritmo de backpropagation: intuición

Para la segunda aprte de la derivación tenemos que derivar la activación $\frac{\partial h_i}{\partial a^{(2)}}$ (tangente hiperbólica) y la pre-activación:

$$\frac{\partial a^{(2)}}{\partial W_{ij}^{(1)}} = \sum_k \frac{\partial}{\partial W_{ij}^{(1)}} W_{ik}^{(1)} x_k = x_j$$

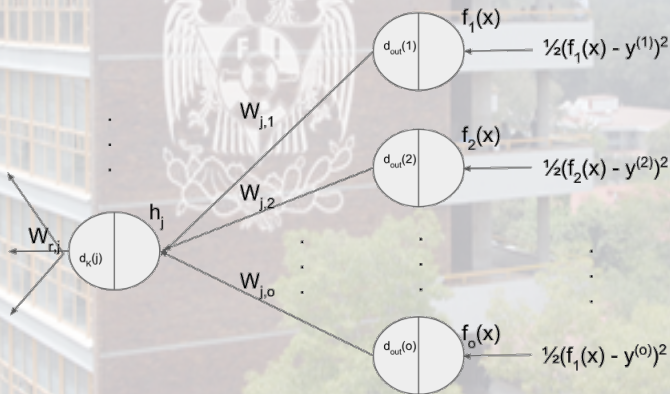
Por tanto, para la **capa oculta** tenemos que obtener la derivada parcial:

$$\begin{aligned} \frac{\partial R}{\partial W_{ij}^{(1)}} &= \left[\frac{\partial R}{\partial \phi} \frac{\partial \phi}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial h_i} \right] \left[\frac{\partial h_i}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial W_{ij}^{(1)}} \right] \\ &= \left[\sum_q W_{iq}^{(2)} (\phi_q(x) - y_q) \sigma(a_q^{(2)}) (1 - \sigma(a_q^{(2)})) \right] [(1 - \tanh^2) x_j] \end{aligned}$$



Algoritmo de backpropagation: intuición

La idea del método de retro-propagación es propagar el error hacia tras de la red, en sentido opuesto:



Algoritmo de backpropagation: intuición

En resumen, la **capa de salida** depende de tres derivadas parciales:

$$\frac{\partial R}{\partial W_{ij}^{(2)}} = \frac{\partial R}{\partial \phi_i} \frac{\partial \phi_i}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial W_{ij}^{(2)}}$$

Mientras que en la **capa oculta** podemos escribirla como:

$$\frac{\partial R}{\partial W_{ij}^{(1)}} = \left[\sum_q \frac{\partial R}{\partial \phi_q} \frac{\partial \phi_q}{\partial a^{(2)}} W_{iq}^{(2)} \right] \left[\frac{\partial h_i}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial W_{ij}^{(1)}} \right]$$

En ambas derivadas, podemos encontrar un factor común, por lo que podemos escribir este factor como una variable:

$$d_{out}(i) = \frac{\partial R}{\partial \phi_i} \frac{\partial \phi_i}{\partial a}$$



Algoritmo de backpropagation: capas ocultas

Cuando agregamos más capas ocultas, podremos observar un patrón. Por ejemplo, pensemos que derivamos una red con **tres capas ocultas** ($\theta = \{W^{(1)}, W^{(2)}, W^{(3)}\}$):

$$\frac{\partial R}{\partial W_{ij}^{(2)}} = \frac{\partial R}{\partial \phi} \frac{\partial \phi}{\partial \mathbf{a}^{(3)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}_i^{(2)}} \frac{\partial \mathbf{h}_i^{(2)}}{\partial \mathbf{a}^{(2)}} \frac{\partial a^{(2)}}{\partial W_{ij}^{(2)}}$$

Podemos ver que en la capa inferior pasa algo similar a la capa de salida:

$$\begin{aligned} \frac{\partial R}{\partial W_{ij}^{(1)}} &= \frac{\partial R}{\partial \phi} \frac{\partial \phi}{\partial \mathbf{a}^{(3)}} \frac{\partial \mathbf{a}^{(3)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial h_i^{(1)}} \frac{\partial h_i^{(1)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial W_{ij}^{(1)}} \\ &= \left[\sum_q \frac{\partial R}{\partial \phi} \frac{\partial \phi}{\partial \mathbf{a}^{(3)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{h}_q^{(2)}} \frac{\partial \mathbf{h}_q^{(2)}}{\partial \mathbf{a}^{(2)}} W_{iq}^{(2)} \right] \frac{\partial h_i^{(1)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial W_{ij}^{(1)}} \end{aligned}$$

El patrón que se repite se almacenará como una variable $d_3(i)$.



Algoritmo de backpropagation

Para describir un algoritmo general, primero obtendremos la derivada sobre la capa de salida. Sea $a = W^{(K+1)}h^{(K)}(x) + b^{(K+1)}$. Debemos obtener las siguientes variables:

$$d_{out}(j) = \frac{\partial R}{\partial \phi_j(a)} \frac{\partial \phi_j(a)}{\partial a}$$

Para obtener la derivada, nótese que:

$$\begin{aligned} \frac{\partial R}{\partial W_{i,j}^{(K+1)}} &= \frac{\partial R}{\partial \phi_j(a)} \frac{\partial \phi_j(a)}{\partial a} h_i^{(K)} \\ &= d_{out}(j) h_i^{(K)} \end{aligned}$$



Derivación del error

Posteriormente, obtendremos la derivada sobre las capas ocultas. Definimos las variables:

$$\begin{aligned}d_k(j) &= \frac{\partial R}{\partial h^{(k+1)}} \frac{\partial h^{(k+1)}}{\partial h_j^{(k)}} \frac{\partial h_j^{(k)}}{\partial a^{(k)}} \\&= \frac{\partial h_j^{(k)}}{\partial a^{(k)}} \sum_q W_{j,q}^{(k+1)} d_{k+1}(q)\end{aligned}$$

En este caso, la derivada viene dada por ($0 \leq k \leq K$, $h^{(0)} = x$):

$$\begin{aligned}\frac{\partial R}{\partial W_{i,j}^{(k)}} &= \left[\frac{\partial h_j^{(k)}}{\partial a^{(k)}} \sum_q W_{j,q}^{(k+1)} d_{k+1}(q) \right] h_i^{(k-1)} \\&= d_k(j) h_i^{(k-1)}\end{aligned}$$



Algoritmo de backpropagation

Input: Una función de riesgo $R(\theta) \equiv R \circ \phi \circ h^{(K)} \circ h^{(K-1)} \circ \dots \circ h^{(1)}$ donde ϕ es una red neuronal con pesos $\theta = \{W^{(K+1)}, W^{(K)}, \dots, W^{(1)}\}$.

Inicialización: Para la función en la capa de salida calcular las variables:

$$d_{out}(j) = \frac{\partial R}{\partial \phi_j(a)} \frac{\partial \phi_j(a)}{\partial a}$$

Y obtener la derivada parcial: $\frac{\partial R}{\partial W_{ij}^{(K+1)}} = d_{out}(j) h_i^{(K)}$

Inducción: Para toda capa oculta k con $k = K, K-1, \dots, 1$ calcular las variables:

$$d_k(j) = \frac{\partial h_j^{(k)}}{\partial a^{(k)}} \sum_q W_{j,q}^{(k+1)} d_{k+1}(q)$$

Y obtener la derivada parcial como: $\frac{\partial R}{\partial W_{i,j}^{(k)}} = d_k(j) h_i^{(k-1)}$, donde $h^{(0)} = x$.

Output: Derivada de la función de riesgo a partir de las derivadas parciales.



Algoritmo de aprendizaje

Inicialización: Aleatoriamente $\theta = \{W^1, \dots, W^{(K+1)}\}$

Entrenamiento: Para todo x en los datos:

Forward: Recorrer la red hasta obtener los valores de salida:

$$f(x) = \phi[W^{(K+1)}h^{(K)}(x) + b^{(K+1)}]$$

Backward: Actualizar los pesos de salida utilizando backpropagation:

$$W_{i,j}^{(K+1)} \leftarrow W_{i,j}^{(K+1)} - \eta \cdot d_{out}(j) \cdot h_i^{(K)}$$

Actualizar los pesos de las capas ocultas:

$$W_{i,j}^{(k)} \leftarrow W_{i,j}^{(k)} - \eta \cdot d_k(j) \cdot h_i^{(k-1)}$$

Terminación: Cuando se alcanza un error igual a 0 o después de T iteraciones.



Ejemplo

Considérese una red neuronal con una sola capa oculta y con dos salidas. Supóngase que los bias son 0. Para la capa oculta, considérese que:

$$h(x) = Ux$$

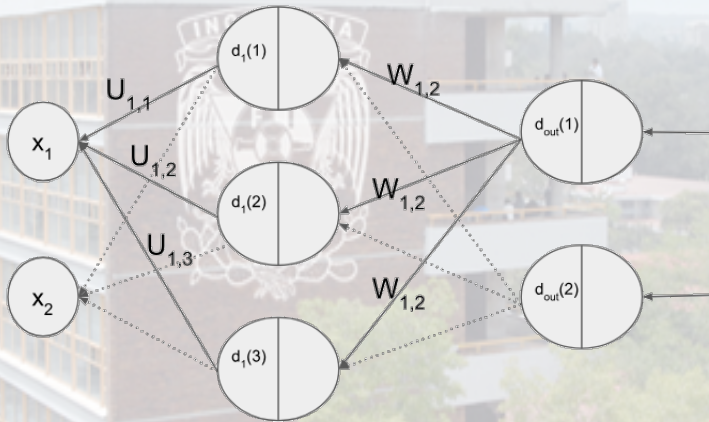
Para la capa de salida, supóngase que:

$$f(x) = \sigma(Wh(x))$$

Elijamos como función de riesgo al error cuadrático medio.



Ejemplo



Ejemplo

Necesitamos derivar el sigmoide. Tenemos que:

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

Como asumimos que la red neuronal tenía sólo una salida:

$$d_{out}(j) = (f_j(x) - y(j))f_j(x)(1 - f_j(x))$$

De donde:

$$W_{i,j} \leftarrow W_{i,j} - \eta d_{out}(j) h_i(x)$$



Ejemplo

Para las capas ocultas, ya que tenemos una función lineal sobre x , tenemos que:

$$d_1(j) = \sum_q w_{j,q} d_{out}(j)$$

Y la actualización de los pesos se da como:

$$U_{i,j} \leftarrow U_{i,j} - \eta d_1(j) x_i$$



Regularización de Tychonoff

Es común que en la práctica se utilice la regularización de Tychonoff (*weight decay*). Así, esta regularización se define dentro de la función de riesgo:

$$R'(\theta) = R(\theta) + \gamma \Omega(\theta) \quad (7)$$

Pueden tomarse las funciones (o su combinación):

- ▶ $L_1: \Omega(\theta) = \|\theta\|_1$
- ▶ $L_2: \Omega(\theta) = \|\theta\|_2^2$



Regularización de Tychonoff

La función de riesgo, puede tomar la forma (L_2 y entropía cruzada):

$$R'(\theta) = - \sum_S y \ln \phi(x) + \gamma ||\theta||^2$$

Nótese que la derivación del regularizador puede realizarse como:

$$\frac{\partial \gamma ||\theta||^2}{\partial \theta_{ij}} = 2\gamma \theta_{ij}$$






Regularización de Tychonoff

De tal forma, la actualización de los parámetros puede realizarse de la siguiente forma:

$$\begin{aligned}\theta_{ij} &\leftarrow \theta_{ij} - \eta \frac{\partial R'}{\partial \theta_{ij}} \\ &\leftarrow \theta_{ij} - \eta \frac{\partial R}{\partial \theta_{ij}} - \eta \frac{\partial \gamma ||\theta||^2}{\partial \theta_{ij}} \\ &\leftarrow \theta_{ij} - \eta \frac{\partial R}{\partial \theta_{ij}} - \eta 2\gamma \theta_{ij}\end{aligned}$$



References

-  JM Bernardo, MJ Bayarri, JO Berger, AP Dawid, D Heckerman, AFM Smith, and M West.
Generative or discriminative? getting the best of both worlds.
Bayesian statistics, 8(3):3–24, 2007.
-  Sebastian Ruder.
An overview of gradient descent optimization algorithms.
arXiv preprint arXiv:1609.04747, 2016.
-  Vladimir Vapnik.
Statistical learning theory. 1998, volume 3.
Wiley, New York, 1998.



The End

