

Aprendizaje en redes neuronales

Dra. Ximena Gutierrez Vasques & Mtro. Víctor Mijangos

Facultad de Ingeniería

13-17 de enero, 2020



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.

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

- Divergencia KL (GANs, Variational AutoEncoders):

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

- Entropía cruzada (FeedForwards, Recurrentes):

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

- Error cuadrático medio (AutoEncoders, Denoising AutoEncoders):

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



Discriminativo vs. Generativo

Las funciones de riesgo 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].



¿Cómo optimizar los parámetros?

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

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

Aunque muchas veces no será posible la igualdad. En estos casos, buscamos $\hat{\theta}$ tal que $\forall \theta \in \Theta$:

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

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 :

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

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

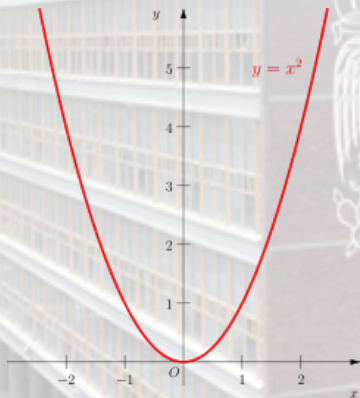
$$\Delta\theta = \nabla_{\theta}R(\theta)$$

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.



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:

$$R(w) = -\mathbb{E}[y \ln \phi(x) + (1 - y) \ln(1 - \phi(x))]$$



GD en el perceptrón

Buscaremos las derivadas parciales para cada ejemplo x :

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

$$= \frac{\partial y \ln \sigma(wx) + (1 - y) \ln(\sigma(wx))}{\partial w_i} \quad (2)$$

$$= (y - \phi(x))x_i \quad (3)$$

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

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



Algunos tipos de GD

Algunas formas de actualizar los parámetros con GD son [2]:

1. (Batch) Gradient Descent: Por cada iteración actualiza por cada uno de los ejemplos (de forma aleatoria):

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

2. Stochastic Gradient Descent:

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

3. Mini-batch gradient descent: Por cada iteración, se toman mini-batches de tamaño n hasta completar todos los ejemplos:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} R(\theta; \text{batch}_n)$$



Otros métodos de GD

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, $\epsilon = 1e - 8$.



Otros métodos de GD

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:

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

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



Optimización en redes multicapa

Si bien la optimización con GD funge 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:

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



Algoritmo de backpropagation: intuición

Sin embargo, aplicar la regla de la cadena puede ser tedioso.

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 $R(\theta) = \frac{1}{2} \sum_{\mathbb{S}} (y - \phi(x))^2$



Algoritmo de backpropagation: intuición

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

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

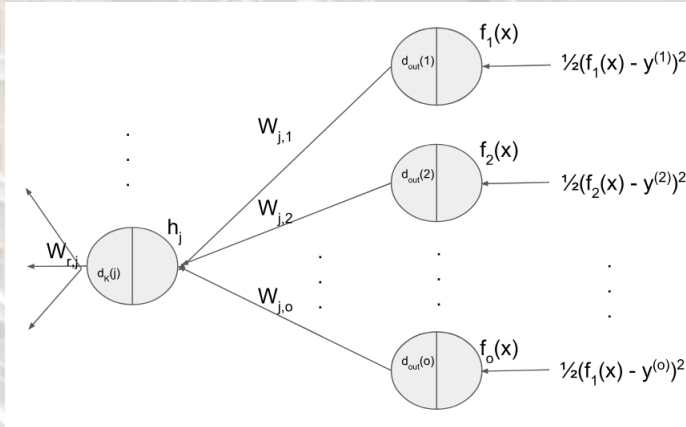
Para la **capa oculta** tenemos:

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



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

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:

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

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

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



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:

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

Actualizar los pesos de las capas ocultas:

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

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:

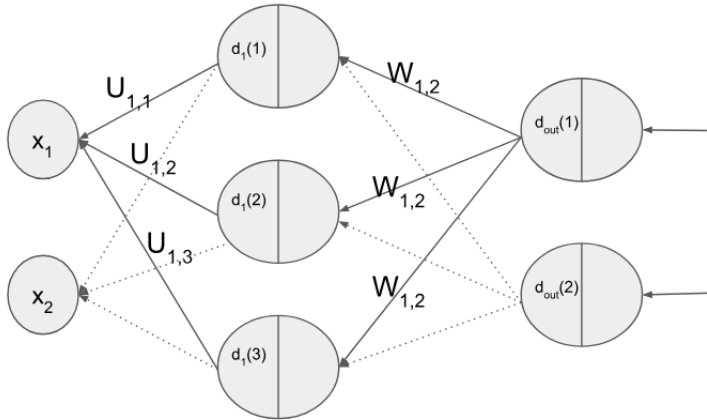
$$h(x) = Ux$$

Para la capa de salida, supóngase que:

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



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:

$$R'(\theta) = R(\theta) + \gamma \Omega(\theta) \quad (4)$$

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

