Clase 18: Redes neuronales (parte 2)

MDS7104 Aprendizaje de Máquinas

Felipe Tobar

Iniciativa de Datos e Inteligencia Artificial
Universidad de Chile

31 mayo 2024

UNIVERSIDAD
DE CHILE

Forward propagation

En una red neuronal *feedforward*, la información fluye a través de la red desde la *entrada* x hasta la *salida* $\hat{y}$. Esto se conoce como **forward propagation** y es la forma en que, durante el entrenamiento, se calcula el costo $J(X, \theta)$

El algoritmo para **forward propagation** es el siguiente.
Requisitos: $W^{(k)}, b^{(k)}, f^{(k)}$ $k \in \{1, ..., l\}$, U , c , g , x input e y output

1. $h^{(0)} \leftarrow x$
2. para $k = 1, \ldots, l$
   - $u^{(k)} \leftarrow h^{(k-1)} W^{(k)} + b^{(k)}$
   - $h^{(k)} \leftarrow f^{(k)}(u^{(k)})$
3. $\hat{y} \leftarrow g(h^{(l)} U + c)$
4. $J \leftarrow L(\hat{y}, y)$

Observación

En la primera iteración del entrenamiento, los pesos son generados aleatoriamente

Backpropagation - Introducción

- El algoritmo de **backpropagation** consiste en transmitir la *información* en sentido inverso, es decir, desde la salida hasta la entrada.
- La utilidad de esto es el cálculo eficiente de cómo cada peso (o parámetro) afecta a la función de costo, donde *eficiente* es en este caso *recursivo*.
- Específicamente, se calculará el gradiente del función de costo con respecto de cada parámetro, donde la recursión es natural debido a la forma composicional de la red.
- Evaluar el gradiente sin la recursión es computacionalmente demandante e implica un cálculo innecesario de términos redundantes.
- La implementación de backpropagation es fundamental para gradiente estocástico
- Como veremos, backpropagation es principalmente operaciones lineales con lo que el uso de GPUs acelera dramáticamente el entrenamiento

Backpropagation - Contexto

- Notación: $h^{(k-1)} \mapsto u^{(k)} = W^{(k)} h^{(k-1)} + b^{(k)} \mapsto h^{(k)} = f(u^{(k)})$
- Entrenamiento mediante **mini-batches**
- Entrada x_d produce estímulo $u_{dj}^{(k)}$ y activación $h_{dj}^{(k)}$ en nodo j de capa k
- Función de costo: MSE en un problema de regresión:

$$J(X, \theta) = \frac{1}{N} \sum_{d=1}^{N} (\hat{y}_d - y_d)^2.$$

Objetivo: actualizar $w_{ij}^{(k)}$: peso que conecta neurona i en capa $k-1$ con neurona j en la capa k.

Para utilizar gradiente estocástico calculamos $\frac{\partial J(X,\theta)}{\partial w_{ij}^{(k)}}$, donde

$$\frac{\partial J(X, \theta)}{\partial w_{ij}^{(k)}} = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial}{\partial w_{ij}^{(k)}} \left(\frac{1}{2} (\hat{y}_d - y_d)^2 \right) = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial J_d}{\partial w_{ij}^{(k)}}.$$

Backpropagation - Resolución I

Utilizando la regla de la cadena

$$\frac{\partial J_d}{\partial w_{ij}^{(k)}} = \frac{\partial J_d}{\partial u_{dj}^{(k)}} \frac{\partial u_{dj}^{(k)}}{\partial w_{ij}^{(k)}}.$$

La expresión $\frac{\partial J_d}{\partial u_{dj}^{(k)}}$ corresponde a un término de **error** y lo denotaremos

$$\delta_{dj}^{(k)} \equiv \frac{\partial J_d}{\partial u_{dj}^{(k)}}.$$

Mientras que para el otro término tenemos que

$$\frac{\partial u_{dj}^{(k)}}{\partial w_{ij}^{(k)}} = \frac{\partial}{\partial w_{ij}^{(k)}} \left(\sum_{a=1}^{k_k} w_{aj}^{(k)} h_{da}^{(k-1)} + b_j^{(k)} \right) = h_{di}^{(k-1)},$$

y así

$$\frac{\partial J_d}{\partial w_{ij}^{(k)}} = \delta_{dj}^{(k)} h_{di}^{(k-1)}.$$

El gradiente total es la suma de los N gradientes:

$$\frac{\partial J}{\partial w_{ij}^{(k)}} = \sum_{d=1}^{N} \delta_{dj}^{(k)} h_{di}^{(k-1)} \Rightarrow \frac{\partial J}{\partial W^{(k)}} = (h^{(k-1)})^\top \otimes \delta^{(k)},$$

donde $\otimes$ es el producto elemento-por-elemento y omitiremos la constante $1/N$.

Backpropagation - Resolución II

Ahora, veremos que $\delta_{dj}^{(k)}$ sigue una recursión.
Nuevamente usando la regla de la cadena:

$$\delta_{dj}^{(k)} = \frac{\partial J_d}{\partial u_{dj}^{(k)}} = \sum_{a=1}^{k_{k+1}} \frac{\partial J_d}{\partial u_{da}^{(k+1)}} \frac{\partial u_{da}^{(k+1)}}{\partial u_{dj}^{(k)}} = \sum_{a=1}^{k_{k+1}} \delta_{da}^{(k+1)} \frac{\partial u_{da}^{(k+1)}}{\partial u_{dj}^{(k)}},$$

donde es directo que

$$\frac{\partial u_{da}^{(k+1)}}{\partial u_{dj}^{(k)}} = w_{ja}^{(k+1)} f'(u_{dj}^{(k)}) \Rightarrow \delta_{dj}^{(k)} = f'(u_{dj}^{(k)}) \sum_{a=1}^{k_{k+1}} w_{ja}^{(k+1)} \delta_{da}^{(k+1)}.$$

Obtuvimos: una expresión para el gradiente en función de $\delta_{dj}^{(k)}$ y una recursión para $\delta_{dj}^{(k)}$, con lo que podemos calcular el gradiente en la capa k en base al gradiente de la capa $k+1$ (de aquí el nombre *backward propagation*).

Lo anterior en su forma matricial

$$\delta^{(k)} = f'(u^{(k)}) * \left(\delta^{(k+1)} \otimes \ (W^{(k+1)})^T\right).$$

Lo único que queda para presentar el algoritmo final es calcular los gradientes en la última capa (capa de output)

Backpropagation - Capa de salida

Asumiendo el problema de regresión con salida escalar función de error MSE, tenemos

$$\delta_{d1}^{(l)} = \frac{\partial J_d}{\partial u_{d1}^{(l)}} = (\hat{y}_d - y_d)(\hat{y}_d)'.$$

Además, la función de activación en la salida es lineal y por tanto $(\hat{y}_d)' = 1$, finalmente el término de normalización N se agrega en este paso:

$$\delta_1^{(l)} = \frac{1}{N}(\hat{y} - y)$$

Ejercicio propuesto

- *Encontrar una expresión para el gradiente de los bias* $(b^{(k)})_k$
- *Calcular el gradiente para un problema con unidad de output sigmoidal (problema de clasificación binario) o para un problema con unidad de salida softmax (problema de clasificación multiclase)*

Backpropagation - Algoritmo

1. Calcular la fase *forward*, guardar los valores $(\hat{y})$, $(u^{(k)})$ y $(h^{(k)})$.
2. Evaluar el error de la última capa $\delta_1^{(l)}$ según la unidad de output
3. Actualizar $U \leftarrow U - \lambda \frac{\partial J}{\partial U}$
4. Actualizar $c \leftarrow c - \lambda \frac{\partial J}{\partial c}$
5. Para $k = l, \ldots, 1$ capas de la red
   - Propagar hacia atrás y calcular el error $\delta^{(k)}$ de las ecuaciones en la capa oculta
   - Evaluar las derivadas parciales $\frac{\partial J}{\partial W^{(k)}}$ y $\frac{\partial J}{\partial b^{(k)}}$ para todos los nodos de la capa k mediante las ecuaciones preliminares, guardar los valores
   - Actualizar los pesos y bias mediante descenso de gradiente

$$W^{(k)} \leftarrow W^{(k)} - \lambda \frac{\partial J}{\partial W^{(k)}}, \quad b^{(k)} \leftarrow b^{(k)} - \lambda \frac{\partial J}{\partial b^{(k)}}$$

Observación

La cantidad de veces que se pase por todos los datos de entrenamiento se conoce como ***épocas***

Regularización

Las redes neuronales, recientemente referidas como *deep learning* por su cantidad de capas, son aplicadas a tareas desafiantes como el procesamiento de imágenes, audio, y texto.

Controlar la complejidad de un modelo no solo se reduce a encontrar la topología de la red y sus parámetros, sino que en la práctica modelo con el mejor ajuste es, en general, un modelo profundo apropiadamente regularizado.

El objetivo de las técnicas de regularización es el de reducir el *error de generalización*, es decir, el error esperado al clasificar datos nuevos pero manteniendo la capacidad del modelo (profundidad, cantidad de nodos, funciones de activación, etc...)

A continuación veremos algunas de ellas:

Técnicas de regularización

- **Regularización L^2**
  Se basa en limitar la norma de los parámetros del modelo es la ya conocida **regularización L^2** (o **ridge regression**), mediante la cual se obtiene la función objetivo regularizada $\tilde{J}$:

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \frac{\alpha}{2}||\boldsymbol{\theta}||_2^2.$$

  No es difícil notar que esta regularización es equivalente a la actualización de parámetros según gradiente estocástico de la forma

$$W^{(k)} \leftarrow (1-\beta)W^{(k)} - \lambda \frac{\partial J}{\partial W^{(k)}}, \quad \beta = \lambda\alpha$$

  y por eso también es llamada *weight decay*.

- **Dropout**

  Esta técnica de regularización consiste en entrenar en cada iteración una fracción de los pesos mediante una elección aleatoria. Su implementación es bastante simple y basta con definir para cada capa k la probabilidad $1 - p_k$ de *apagar* una neurona en particular.

$$h^{(k)} = f^{(k)}(h^{(k-1)}W^{(k)} + b^{(k)}) * \frac{M^{(k)}}{p_k}, \quad M_i^{(k)} \sim \text{Bernoulli}(p_k) \ \ \forall i, k.$$

  Notar que al aplicar backward, también es necesario 'apagar' las neuronas que no participaron del forward para evitar que sean entrenadas.

- **Early Stopping**

  La cantidad de épocas disminuye el error de entrenamiento pero no siempre el de generalización, es más, con una cantidad grande de épocas, el error de generalización aumenta y esto es porque la red aprende con exactitud cada input en el conjunto de train y no sus características más importantes, a esto se le denomina **overfitting**.

  La técnica de *Early stopping* consiste en evitar esto, deteniendo el algoritmo antes de llegar al overfitting según un parámetro de decisión. Es importante destacar que esto se realiza evaluando sobre el conjunto de validación.

Clase 18: Redes neuronales (parte 2)

MDS7104 Aprendizaje de Máquinas

Felipe Tobar

Iniciativa de Datos e Inteligencia Artificial
Universidad de Chile

31 mayo 2024

UNIVERSIDAD
DE CHILE