Clase 18 - Redes neuronales (parte 2)

Aprendizaje de Máquinas - MA5204

Felipe Tobar

Department of Mathematical Engineering &
Center for Mathematical Modelling
Universidad de Chile

14 de marzo de 2021

Forward propagation

Al usar una red neuronal *feedforward*, la información fluye a través de la red desde el ingreso de un input x hasta producir un output $\hat{y}$. Esto se conoce como **forward propagation** y es la que durante el entrenamiento calcula el costo $J(X, \theta)$

El algoritmo que la describe es el siguiente

Requisitos: $W^{(k)}, b^{(k)}, f^{(k)}$ $k \in \{1, ..., l\}$, U , c , g y x input

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

ObservaciÃşn

Los pesos y bias en la primera iteración del algoritmo son generados aleatoriamente

Forward propagation

Al usar una red neuronal *feedforward*, la información fluye a través de la red desde el ingreso de un input x hasta producir un output $\hat{y}$. Esto se conoce como **forward propagation** y es la que durante el entrenamiento calcula el costo $J(X, \theta)$
El algoritmo que la describe es el siguiente
Requisitos: $W^{(k)}, b^{(k)}, f^{(k)}$ $k \in \{1, ..., l\}$, U , c , g y x input

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

ObservaciÃşn

Los pesos y bias en la primera iteración del algoritmo son generados aleatoriamente

Forward propagation

Al usar una red neuronal *feedforward*, la información fluye a través de la red desde el ingreso de un input x hasta producir un output $\hat{y}$. Esto se conoce como **forward propagation** y es la que durante el entrenamiento calcula el costo $J(X, \theta)$
El algoritmo que la describe es el siguiente
Requisitos: $W^{(k)}, b^{(k)}, f^{(k)}$ $k \in \{1, ..., l\}$, U , c , g y x input

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

ObservaciÃşn

Los pesos y bias en la primera iteración del algoritmo son generados aleatoriamente

Forward propagation

Al usar una red neuronal *feedforward*, la información fluye a través de la red desde el ingreso de un input x hasta producir un output $\hat{y}$. Esto se conoce como **forward propagation** y es la que durante el entrenamiento calcula el costo $J(X, \theta)$
El algoritmo que la describe es el siguiente
Requisitos: $W^{(k)}, b^{(k)}, f^{(k)}$ $k \in \{1, ..., l\}$, U , c , g y x input

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

ObservaciÃşn

Los pesos y bias en la primera iteración del algoritmo son generados aleatoriamente

Forward propagation

Al usar una red neuronal *feedforward*, la información fluye a través de la red desde el ingreso de un input x hasta producir un output $\hat{y}$. Esto se conoce como **forward propagation** y es la que durante el entrenamiento calcula el costo $J(X, \theta)$
El algoritmo que la describe es el siguiente
Requisitos: $W^{(k)}, b^{(k)}, f^{(k)}$ $k \in \{1, ..., l\}$, U , c , g y x input

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

ObservaciÃşn

Los pesos y bias en la primera iteración del algoritmo son generados aleatoriamente

Forward propagation

Al usar una red neuronal *feedforward*, la información fluye a través de la red desde el ingreso de un input x hasta producir un output $\hat{y}$. Esto se conoce como **forward propagation** y es la que durante el entrenamiento calcula el costo $J(X, \theta)$
El algoritmo que la describe es el siguiente
Requisitos: $W^{(k)}, b^{(k)}, f^{(k)}$ $k \in \{1, ..., l\}$, U , c , g y x input

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

ObservaciÃşn

Los pesos y bias en la primera iteración del algoritmo son generados aleatoriamente

Forward propagation

Al usar una red neuronal *feedforward*, la información fluye a través de la red desde el ingreso de un input x hasta producir un output $\hat{y}$. Esto se conoce como **forward propagation** y es la que durante el entrenamiento calcula el costo $J(X, \theta)$
El algoritmo que la describe es el siguiente
Requisitos: $W^{(k)}, b^{(k)}, f^{(k)}$ $k \in \{1, ..., l\}$, U , c , g y x input

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

ObservaciÃşn

Los pesos y bias en la primera iteración del algoritmo son generados aleatoriamente

Forward propagation

Al usar una red neuronal *feedforward*, la información fluye a través de la red desde el ingreso de un input x hasta producir un output $\hat{y}$. Esto se conoce como **forward propagation** y es la que durante el entrenamiento calcula el costo $J(X, \theta)$
El algoritmo que la describe es el siguiente
Requisitos: $W^{(k)}, b^{(k)}, f^{(k)}$ $k \in \{1, ..., l\}$, U , c , g y x input

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

ObservaciÃşn

Los pesos y bias en la primera iteración del algoritmo son generados aleatoriamente

Backpropagation - Introducción

El algoritmo de **backpropagation** permite que la información del costo fluya en sentido inverso a través de la red para calcular el gradiente de manera computacionalmente eficiente. El gradiente se calculará de esta forma ya que, aunque es posible obtener una expresión analítica para este, evaluar la expresión puede ser muy caro computacionalmente. Luego de obtener el gradiente, otro algoritmo como descenso de gradiente estocástico realizará el aprendizaje usando la expresión que fue calculada.

El algoritmo de backpropagation ha experimentatdo un resurgimiento en estas últimas décadas por su implementación en redes neuronales para tareas tales como reconocimiento de imágenes o procesamiento de lenguaje natural.
Es considerado un algoritmo eficiente e implementaciones modernas de estas aprovechan el paralelismo de las GPU para mejorar su rendimiento.

El gradiente se calculará de esta forma ya que, aunque es posible obtener una expresión analítica para este, evaluar la expresión puede ser muy caro computacionalmente.
A continuación se describe brevemente el algoritmo:

Backpropagation - Introducción

El algoritmo de **backpropagation** permite que la información del costo fluya en sentido inverso a través de la red para calcular el gradiente de manera computacionalmente eficiente. El gradiente se calculará de esta forma ya que, aunque es posible obtener una expresión analítica para este, evaluar la expresión puede ser muy caro computacionalmente. Luego de obtener el gradiente, otro algoritmo como descenso de gradiente estocástico realizará el aprendizaje usando la expresión que fue calculada.

El algoritmo de backpropagation ha experimentatdo un resurgimiento en estas últimas décadas por su implementación en redes neuronales para tareas tales como reconocimiento de imágenes o procesamiento de lenguaje natural.
Es considerado un algoritmo eficiente e implementaciones modernas de estas aprovechan el paralelismo de las GPU para mejorar su rendimiento.

El gradiente se calculará de esta forma ya que, aunque es posible obtener una expresión analítica para este, evaluar la expresión puede ser muy caro computacionalmente.
A continuación se describe brevemente el algoritmo:

Backpropagation - Introducción

El algoritmo de **backpropagation** permite que la información del costo fluya en sentido inverso a través de la red para calcular el gradiente de manera computacionalmente eficiente. El gradiente se calculará de esta forma ya que, aunque es posible obtener una expresión analítica para este, evaluar la expresión puede ser muy caro computacionalmente. Luego de obtener el gradiente, otro algoritmo como descenso de gradiente estocástico realizará el aprendizaje usando la expresión que fue calculada.

El algoritmo de backpropagation ha experimentatdo un resurgimiento en estas últimas décadas por su implementación en redes neuronales para tareas tales como reconocimiento de imágenes o procesamiento de lenguaje natural.
Es considerado un algoritmo eficiente e implementaciones modernas de estas aprovechan el paralelismo de las GPU para mejorar su rendimiento.

El gradiente se calculará de esta forma ya que, aunque es posible obtener una expresión analítica para este, evaluar la expresión puede ser muy caro computacionalmente.
A continuación se describe brevemente el algoritmo:

Backpropagation - Introducción

El algoritmo de **backpropagation** permite que la información del costo fluya en sentido inverso a través de la red para calcular el gradiente de manera computacionalmente eficiente. El gradiente se calculará de esta forma ya que, aunque es posible obtener una expresión analítica para este, evaluar la expresión puede ser muy caro computacionalmente. Luego de obtener el gradiente, otro algoritmo como descenso de gradiente estocástico realizará el aprendizaje usando la expresión que fue calculada.

El algoritmo de backpropagation ha experimentatdo un resurgimiento en estas últimas décadas por su implementación en redes neuronales para tareas tales como reconocimiento de imágenes o procesamiento de lenguaje natural.
Es considerado un algoritmo eficiente e implementaciones modernas de estas aprovechan el paralelismo de las GPU para mejorar su rendimiento.

El gradiente se calculará de esta forma ya que, aunque es posible obtener una expresión analítica para este, evaluar la expresión puede ser muy caro computacionalmente.
A continuación se describe brevemente el algoritmo:

Backpropagation - Introducción

El algoritmo de **backpropagation** permite que la información del costo fluya en sentido inverso a través de la red para calcular el gradiente de manera computacionalmente eficiente. El gradiente se calculará de esta forma ya que, aunque es posible obtener una expresión analítica para este, evaluar la expresión puede ser muy caro computacionalmente. Luego de obtener el gradiente, otro algoritmo como descenso de gradiente estocástico realizará el aprendizaje usando la expresión que fue calculada.

El algoritmo de backpropagation ha experimentatdo un resurgimiento en estas últimas décadas por su implementación en redes neuronales para tareas tales como reconocimiento de imágenes o procesamiento de lenguaje natural.
Es considerado un algoritmo eficiente e implementaciones modernas de estas aprovechan el paralelismo de las GPU para mejorar su rendimiento.

El gradiente se calculará de esta forma ya que, aunque es posible obtener una expresión analítica para este, evaluar la expresión puede ser muy caro computacionalmente.
A continuación se describe brevemente el algoritmo:

Backpropagation - Preliminares

- Utilicemos la notación para el paso intermedio antes de aplicar la función de activación como $u = hW + b$
- Supongamos además que el entrenamiento de la red se realiza con un paquete de datos (batch), es decir $(x^d)_{d=1}^N$ conjunto de N inputs, luego $u_{dj}^{(k)}$ corresponde al valor de u para el input d en el nodo j para la capa k
- Se verá el algoritmo de backpropagation para función de error MSE en un problema de regresión.

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

Nuestro objetivo será actualizar todos los valores $w_{ij}^{(k)}$ (peso del nodo i al nodo j en la capa k). Es decir, para utilizar el método del descenso de gradiente es necesario calcular $\frac{\partial J(X,\theta)}{\partial w_{ij}^{(k)}}$ notemos que

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

Veamos esto último

Backpropagation - Preliminares

- Utilicemos la notación para el paso intermedio antes de aplicar la función de activación como $u = hW + b$
- Supongamos además que el entrenamiento de la red se realiza con un paquete de datos (batch), es decir $(x^d)_{d=1}^N$ conjunto de N inputs, luego $u_{dj}^{(k)}$ corresponde al valor de u para el input d en el nodo j para la capa k
- Se verá el algoritmo de backpropagation para función de error MSE en un problema de regresión.

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

Nuestro objetivo será actualizar todos los valores $w_{ij}^{(k)}$ (peso del nodo i al nodo j en la capa k). Es decir, para utilizar el método del descenso de gradiente es necesario calcular $\frac{\partial J(X,\theta)}{\partial w_{ij}^{(k)}}$ notemos que

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

Veamos esto último

Backpropagation - Preliminares

- Utilicemos la notación para el paso intermedio antes de aplicar la función de activación como $u = hW + b$
- Supongamos además que el entrenamiento de la red se realiza con un paquete de datos (batch), es decir $(x^d)_{d=1}^N$ conjunto de N inputs, luego $u_{dj}^{(k)}$ corresponde al valor de u para el input d en el nodo j para la capa k
- Se verá el algoritmo de backpropagation para función de error MSE en un problema de regresión.

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

Nuestro objetivo será actualizar todos los valores $w_{ij}^{(k)}$ (peso del nodo i al nodo j en la capa k). Es decir, para utilizar el método del descenso de gradiente es necesario calcular $\frac{\partial J(X,\theta)}{\partial w_{ij}^{(k)}}$ notemos que

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

Veamos esto último

Backpropagation - Preliminares

- Utilicemos la notación para el paso intermedio antes de aplicar la función de activación como $u = hW + b$
- Supongamos además que el entrenamiento de la red se realiza con un paquete de datos (batch), es decir $(x^d)_{d=1}^N$ conjunto de N inputs, luego $u_{dj}^{(k)}$ corresponde al valor de u para el input d en el nodo j para la capa k
- Se verá el algoritmo de backpropagation para función de error MSE en un problema de regresión.

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

Nuestro objetivo será actualizar todos los valores $w_{ij}^{(k)}$ (peso del nodo i al nodo j en la capa k). Es decir, para utilizar el método del descenso de gradiente es necesario calcular $\frac{\partial J(X,\theta)}{\partial w_{ij}^{(k)}}$ notemos que

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

Veamos esto último

Backpropagation - Preliminares

- Utilicemos la notación para el paso intermedio antes de aplicar la función de activación como $u = hW + b$
- Supongamos además que el entrenamiento de la red se realiza con un paquete de datos (batch), es decir $(x^d)_{d=1}^N$ conjunto de N inputs, luego $u_{dj}^{(k)}$ corresponde al valor de u para el input d en el nodo j para la capa k
- Se verá el algoritmo de backpropagation para función de error MSE en un problema de regresión.

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

Nuestro objetivo será actualizar todos los valores $w_{ij}^{(k)}$ (peso del nodo i al nodo j en la capa k). Es decir, para utilizar el método del descenso de gradiente es necesario calcular $\frac{\partial J(X,\theta)}{\partial w_{ij}^{(k)}}$ notemos que

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

Veamos esto último

Backpropagation - Preliminares

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, será la suma de los N gradientes y que expresaremos en su forma matricial

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

Omitiremos la constante $1/N$ hasta el final del algoritmo.

Backpropagation - Preliminares

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, será la suma de los N gradientes y que expresaremos en su forma matricial

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

Omitiremos la constante $1/N$ hasta el final del algoritmo.

Backpropagation - Preliminares

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, será la suma de los N gradientes y que expresaremos en su forma matricial

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

Omitiremos la constante $1/N$ hasta el final del algoritmo.

Backpropagation - Preliminares

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, será la suma de los N gradientes y que expresaremos en su forma matricial

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

Omitiremos la constante $1/N$ hasta el final del algoritmo.

Backpropagation - Preliminares

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, será la suma de los N gradientes y que expresaremos en su forma matricial

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

Omitiremos la constante $1/N$ hasta el final del algoritmo.

Backpropagation - Capas ocultas

Nuevamente, de la regla de la cadena

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

no es difícil ver que

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

Hemos encontrado una expresión para calcular el gradiente en una capa k en base al gradiente de la siguiente capa $k+1$ (De aquí el nombre backward propagation).

Lo anterior en su forma matricial

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

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

Backpropagation - Capas ocultas

Nuevamente, de la regla de la cadena

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

no es difícil ver que

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

Hemos encontrado una expresión para calcular el gradiente en una capa k en base al gradiente de la siguiente capa $k+1$ (De aquí el nombre backward propagation).

Lo anterior en su forma matricial

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

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

Backpropagation - Capas ocultas

Nuevamente, de la regla de la cadena

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

no es difícil ver que

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

Hemos encontrado una expresión para calcular el gradiente en una capa k en base al gradiente de la siguiente capa $k+1$ (De aquí el nombre backward propagation).

Lo anterior en su forma matricial

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

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

Backpropagation - Capas ocultas

Nuevamente, de la regla de la cadena

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

no es difícil ver que

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

Hemos encontrado una expresión para calcular el gradiente en una capa k en base al gradiente de la siguiente capa $k+1$ (De aquí el nombre backward propagation).

Lo anterior en su forma matricial

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

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

Backpropagation - Capas ocultas

Nuevamente, de la regla de la cadena

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

no es difícil ver que

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

Hemos encontrado una expresión para calcular el gradiente en una capa k en base al gradiente de la siguiente capa $k+1$ (De aquí el nombre backward propagation).

Lo anterior en su forma matricial

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

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

Backpropagation - Capas ocultas

Nuevamente, de la regla de la cadena

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

no es difícil ver que

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

Hemos encontrado una expresión para calcular el gradiente en una capa k en base al gradiente de la siguiente capa $k+1$ (De aquí el nombre backward propagation).

Lo anterior en su forma matricial

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

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

Backpropagation - Capa de output

Estamos suponiendo un problema de regresión por lo que el output será de una sola salida y la función de error es MSE, entonces

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

Además, la función de activación en el output será lineal y por tanto $(\hat{y}_d)' = 1$, finalmente el término de normalización N se agrega en este paso. La forma matricial queda en

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

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 output softmax (problema de clasificación multiclase)*

Backpropagation - Capa de output

Estamos suponiendo un problema de regresión por lo que el output será de una sola salida y la función de error es MSE, entonces

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

Además, la función de activación en el output será lineal y por tanto $(\hat{y}_d)' = 1$, finalmente el término de normalización N se agrega en este paso. La forma matricial queda en

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

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 output softmax (problema de clasificación multiclase)*

Backpropagation - Capa de output

Estamos suponiendo un problema de regresión por lo que el output será de una sola salida y la función de error es MSE, entonces

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

Además, la función de activación en el output será lineal y por tanto $(\hat{y}_d)' = 1$, finalmente el término de normalización N se agrega en este paso. La forma matricial queda en

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

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 output softmax (problema de clasificación multiclase)*

Backpropagation - Capa de output

Estamos suponiendo un problema de regresión por lo que el output será de una sola salida y la función de error es MSE, entonces

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

Además, la función de activación en el output será lineal y por tanto $(\hat{y}_d)' = 1$, finalmente el término de normalización N se agrega en este paso. La forma matricial queda en

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

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 output softmax (problema de clasificación multiclase)*

Backpropagation - Capa de output

Estamos suponiendo un problema de regresión por lo que el output será de una sola salida y la función de error es MSE, entonces

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

Además, la función de activación en el output será lineal y por tanto $(\hat{y}_d)' = 1$, finalmente el término de normalización N se agrega en este paso. La forma matricial queda en

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

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 output softmax (problema de clasificación multiclase)*

Backpropagation - Algoritmo

1. Calcular la fase de forward, guardar los valores $(\hat{y})_d$, $(u_{ij}^{(k)})_{ijd}$ y $(h_{ij}^{(k)})_{ijd}$.
2. Evaluar el error de la última capa $\delta_1^{(l)}$ utilizando las ecuaciones en la capa de output
3. Para $k = 1 \dots l-1$ capas de la red
   - Propagar hacia atrás y calcular el error $\delta^{(k)}$ de las ecuaciones en la capa oculta
   - Evaluar las derivadas parciales $\frac{\partial J}{\partial w_{ij}^{(k)}}$ y $\frac{\partial J}{\partial b_j^{(k)}}$ para todos los nodos de la capa k mediante las ecuaciones preliminares, guardar los valores
4. Actualizar los pesos y bias mediante descenso de gradiente

$$(w_{ij}^{(k)})' = w_{ij}^{(k)} - \lambda \frac{\partial J}{\partial w_{ij}^{(k)}}, \quad (b_j^{(k)})' = b_j^{(k)} - \lambda \frac{\partial J}{\partial b_j^{(k)}}, \quad \forall i, j, k$$

ObservaciÃşn

- *Usualmente los datos son entregados en paquetes de datos (batch) y los pesos/bias actualizados mediante descenso de gradiente estocástico*
- *La cantidad de veces que se pase por todos los datos de entrenamiento se conoce como* ***épocas***

Backpropagation - Algoritmo

1. Calcular la fase de forward, guardar los valores $(\hat{y})_d$, $(u_{ij}^{(k)})_{ijd}$ y $(h_{ij}^{(k)})_{ijd}$.
2. Evaluar el error de la última capa $\delta_1^{(l)}$ utilizando las ecuaciones en la capa de output
3. Para $k = 1 \dots l-1$ capas de la red
   - Propagar hacia atrás y calcular el error $\delta^{(k)}$ de las ecuaciones en la capa oculta
   - Evaluar las derivadas parciales $\frac{\partial J}{\partial w_{ij}^{(k)}}$ y $\frac{\partial J}{\partial b_j^{(k)}}$ para todos los nodos de la capa k mediante las ecuaciones preliminares, guardar los valores
4. Actualizar los pesos y bias mediante descenso de gradiente

$$(w_{ij}^{(k)})' = w_{ij}^{(k)} - \lambda \frac{\partial J}{\partial w_{ij}^{(k)}}, \quad (b_j^{(k)})' = b_j^{(k)} - \lambda \frac{\partial J}{\partial b_j^{(k)}}, \quad \forall i, j, k$$

ObservaciÃşn

- *Usualmente los datos son entregados en paquetes de datos (batch) y los pesos/bias actualizados mediante descenso de gradiente estocástico*
- *La cantidad de veces que se pase por todos los datos de entrenamiento se conoce como* ***épocas***

Backpropagation - Algoritmo

1. Calcular la fase de forward, guardar los valores $(\hat{y})_d$, $(u_{ij}^{(k)})_{ijd}$ y $(h_{ij}^{(k)})_{ijd}$.
2. Evaluar el error de la última capa $\delta_1^{(l)}$ utilizando las ecuaciones en la capa de output
3. Para $k = 1 \dots l-1$ capas de la red
   - Propagar hacia atrás y calcular el error $\delta^{(k)}$ de las ecuaciones en la capa oculta
   - Evaluar las derivadas parciales $\frac{\partial J}{\partial w_{ij}^{(k)}}$ y $\frac{\partial J}{\partial b_j^{(k)}}$ para todos los nodos de la capa k mediante las ecuaciones preliminares, guardar los valores
4. Actualizar los pesos y bias mediante descenso de gradiente

$$(w_{ij}^{(k)})' = w_{ij}^{(k)} - \lambda \frac{\partial J}{\partial w_{ij}^{(k)}}, \quad (b_j^{(k)})' = b_j^{(k)} - \lambda \frac{\partial J}{\partial b_j^{(k)}}, \quad \forall i, j, k$$

ObservaciÃşn

- *Usualmente los datos son entregados en paquetes de datos (batch) y los pesos/bias actualizados mediante descenso de gradiente estocástico*
- *La cantidad de veces que se pase por todos los datos de entrenamiento se conoce como* ***épocas***

Backpropagation - Algoritmo

1. Calcular la fase de forward, guardar los valores $(\hat{y})_d$, $(u_{ij}^{(k)})_{ijd}$ y $(h_{ij}^{(k)})_{ijd}$.
2. Evaluar el error de la última capa $\delta_1^{(l)}$ utilizando las ecuaciones en la capa de output
3. Para $k = 1 \dots l-1$ capas de la red
   - Propagar hacia atrás y calcular el error $\delta^{(k)}$ de las ecuaciones en la capa oculta
   - Evaluar las derivadas parciales $\frac{\partial J}{\partial w_{ij}^{(k)}}$ y $\frac{\partial J}{\partial b_j^{(k)}}$ para todos los nodos de la capa k mediante las ecuaciones preliminares, guardar los valores
4. Actualizar los pesos y bias mediante descenso de gradiente

$$(w_{ij}^{(k)})' = w_{ij}^{(k)} - \lambda \frac{\partial J}{\partial w_{ij}^{(k)}}, \quad (b_j^{(k)})' = b_j^{(k)} - \lambda \frac{\partial J}{\partial b_j^{(k)}}, \quad \forall i, j, k$$

ObservaciÃşn

- *Usualmente los datos son entregados en paquetes de datos (batch) y los pesos/bias actualizados mediante descenso de gradiente estocástico*
- *La cantidad de veces que se pase por todos los datos de entrenamiento se conoce como* ***épocas***

Backpropagation - Algoritmo

1. Calcular la fase de forward, guardar los valores $(\hat{y})_d$, $(u_{ij}^{(k)})_{ijd}$ y $(h_{ij}^{(k)})_{ijd}$.
2. Evaluar el error de la última capa $\delta_1^{(l)}$ utilizando las ecuaciones en la capa de output
3. Para $k = 1 \dots l-1$ capas de la red
   - Propagar hacia atrás y calcular el error $\delta^{(k)}$ de las ecuaciones en la capa oculta
   - Evaluar las derivadas parciales $\frac{\partial J}{\partial w_{ij}^{(k)}}$ y $\frac{\partial J}{\partial b_j^{(k)}}$ para todos los nodos de la capa k mediante las ecuaciones preliminares, guardar los valores
4. Actualizar los pesos y bias mediante descenso de gradiente

$$(w_{ij}^{(k)})' = w_{ij}^{(k)} - \lambda \frac{\partial J}{\partial w_{ij}^{(k)}}, \quad (b_j^{(k)})' = b_j^{(k)} - \lambda \frac{\partial J}{\partial b_j^{(k)}}, \quad \forall i, j, k$$

ObservaciÃşn

- *Usualmente los datos son entregados en paquetes de datos (batch) y los pesos/bias actualizados mediante descenso de gradiente estocástico*
- *La cantidad de veces que se pase por todos los datos de entrenamiento se conoce como* ***épocas***

Backpropagation - Algoritmo

1. Calcular la fase de forward, guardar los valores $(\hat{y})_d$, $(u_{ij}^{(k)})_{ijd}$ y $(h_{ij}^{(k)})_{ijd}$.
2. Evaluar el error de la última capa $\delta_1^{(l)}$ utilizando las ecuaciones en la capa de output
3. Para $k = 1 \dots l-1$ capas de la red
   - Propagar hacia atrás y calcular el error $\delta^{(k)}$ de las ecuaciones en la capa oculta
   - Evaluar las derivadas parciales $\frac{\partial J}{\partial w_{ij}^{(k)}}$ y $\frac{\partial J}{\partial b_j^{(k)}}$ para todos los nodos de la capa k mediante las ecuaciones preliminares, guardar los valores
4. Actualizar los pesos y bias mediante descenso de gradiente

$$(w_{ij}^{(k)})' = w_{ij}^{(k)} - \lambda \frac{\partial J}{\partial w_{ij}^{(k)}}, \quad (b_j^{(k)})' = b_j^{(k)} - \lambda \frac{\partial J}{\partial b_j^{(k)}}, \quad \forall i, j, k$$

ObservaciÃşn

- *Usualmente los datos son entregados en paquetes de datos (batch) y los pesos/bias actualizados mediante descenso de gradiente estocástico*
- *La cantidad de veces que se pase por todos los datos de entrenamiento se conoce como* ***épocas***

Backpropagation - Algoritmo

1. Calcular la fase de forward, guardar los valores $(\hat{y})_d$, $(u_{ij}^{(k)})_{ijd}$ y $(h_{ij}^{(k)})_{ijd}$.
2. Evaluar el error de la última capa $\delta_1^{(l)}$ utilizando las ecuaciones en la capa de output
3. Para $k = 1 \dots l-1$ capas de la red
   - Propagar hacia atrás y calcular el error $\delta^{(k)}$ de las ecuaciones en la capa oculta
   - Evaluar las derivadas parciales $\frac{\partial J}{\partial w_{ij}^{(k)}}$ y $\frac{\partial J}{\partial b_j^{(k)}}$ para todos los nodos de la capa k mediante las ecuaciones preliminares, guardar los valores
4. Actualizar los pesos y bias mediante descenso de gradiente

$$(w_{ij}^{(k)})' = w_{ij}^{(k)} - \lambda \frac{\partial J}{\partial w_{ij}^{(k)}}, \quad (b_j^{(k)})' = b_j^{(k)} - \lambda \frac{\partial J}{\partial b_j^{(k)}}, \quad \forall i, j, k$$

ObservaciÃşn

- *Usualmente los datos son entregados en paquetes de datos (batch) y los pesos/bias actualizados mediante descenso de gradiente estocástico*
- *La cantidad de veces que se pase por todos los datos de entrenamiento se conoce como* ***épocas***

Regularización

Las redes neuronales y algoritmos de deep learning son aplicados a tareas extremadamente complejas como lo son el procesamiento de imágenes, audio, y texto. Controlar la complejidad de un modelo no solo se reduce a encontrar el tamaño y cantidad de parámetros adecuados, como se ha visto para otros modelos de aprendizaje de máquinas, sino que en la práctica el modelo con el mejor ajuste por lo general será un modelo grande (profundo) que ha sido regularizado apropiadamente.

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

A continuación veremos algunas de ellas:

Regularización

Las redes neuronales y algoritmos de deep learning son aplicados a tareas extremadamente complejas como lo son el procesamiento de imágenes, audio, y texto. Controlar la complejidad de un modelo no solo se reduce a encontrar el tamaño y cantidad de parámetros adecuados, como se ha visto para otros modelos de aprendizaje de máquinas, sino que en la práctica el modelo con el mejor ajuste por lo general será un modelo grande (profundo) que ha sido regularizado apropiadamente.

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

A continuación veremos algunas de ellas:

Regularización

Las redes neuronales y algoritmos de deep learning son aplicados a tareas extremadamente complejas como lo son el procesamiento de imágenes, audio, y texto. Controlar la complejidad de un modelo no solo se reduce a encontrar el tamaño y cantidad de parámetros adecuados, como se ha visto para otros modelos de aprendizaje de máquinas, sino que en la práctica el modelo con el mejor ajuste por lo general será un modelo grande (profundo) que ha sido regularizado apropiadamente.
El objetivo de las técnicas de regularización es el de reducir el *error de generalización*, es decir, el error esperado al clasificar datos nunca antes vistos 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**

  Una regularización que 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 dificil ver que esta regularización es equivalente a la actualización de parámetros según gradiente estocástico de la forma

  $$(w_{ij}^{(k)})' = (1-\beta)w_{ij}^{(k)} - \lambda\frac{\partial J}{\partial w_{ij}^{(k)}}, \quad \beta = \lambda\alpha$$

  y por eso también es llamada *'weight decay'* y es la forma en que se implementa

Técnicas de regularización

- **Regularización L^2**
  Una regularización que 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 dificil ver que esta regularización es equivalente a la actualización de parámetros según gradiente estocástico de la forma

$$(w_{ij}^{(k)})' = (1-\beta)w_{ij}^{(k)} - \lambda\frac{\partial J}{\partial w_{ij}^{(k)}}, \quad \beta = \lambda\alpha$$

  y por eso también es llamada *'weight decay'* y es la forma en que se implementa

Técnicas de regularización

- **Regularización L^2**
  Una regularización que 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 dificil ver que esta regularización es equivalente a la actualización de parámetros según gradiente estocástico de la forma

$$(w_{ij}^{(k)})' = (1-\beta)w_{ij}^{(k)} - \lambda\frac{\partial J}{\partial w_{ij}^{(k)}}, \quad \beta = \lambda\alpha$$

  y por eso también es llamada *'weight decay'* y es la forma en que se implementa

Técnicas de regularización

- **Regularización L^2**
  Una regularización que 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 dificil ver que esta regularización es equivalente a la actualización de parámetros según gradiente estocástico de la forma

$$(w_{ij}^{(k)})' = (1-\beta)w_{ij}^{(k)} - \lambda\frac{\partial J}{\partial w_{ij}^{(k)}}, \quad \beta = \lambda\alpha$$

  y por eso también es llamada *'weight decay'* y es la forma en que se implementa

Técnicas de regularización

- **Regularización L^2**
  Una regularización que 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 dificil ver que esta regularización es equivalente a la actualización de parámetros según gradiente estocástico de la forma

$$(w_{ij}^{(k)})' = (1-\beta)w_{ij}^{(k)} - \lambda\frac{\partial J}{\partial w_{ij}^{(k)}}, \quad \beta = \lambda\alpha$$

  y por eso también es llamada *'weight decay'* y es la forma en que se implementa

- **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.

- Otras técnicas: **Bootstrap , data augmentation , noise injection**

- **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.

- Otras técnicas: **Bootstrap , data augmentation , noise injection**

- **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.

- Otras técnicas: **Bootstrap , data augmentation , noise injection**

- **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.

- Otras técnicas: **Bootstrap , data augmentation , noise injection**

- **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.

- Otras técnicas: **Bootstrap , data augmentation , noise injection**

- **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.

- Otras técnicas: **Bootstrap , data augmentation , noise injection**

Clase 18 - Redes neuronales (parte 2)

Aprendizaje de Máquinas - MA5204

Felipe Tobar

Department of Mathematical Engineering &
Center for Mathematical Modelling
Universidad de Chile

14 de marzo de 2021