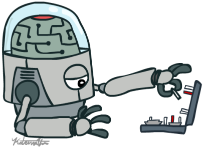


Aprendizaje Supervisado

A. Atutxa

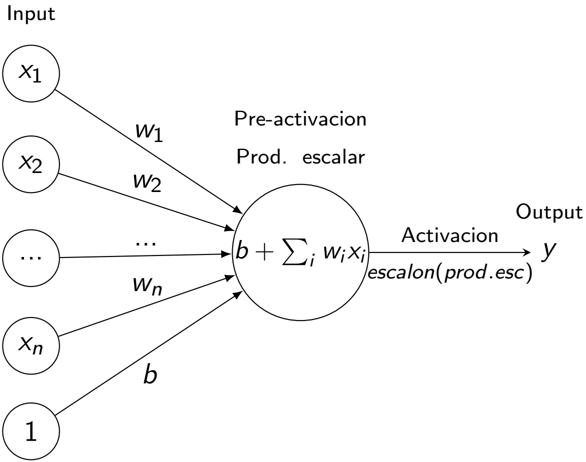
LSI BIlbao

Noviembre 2023

[1] Basado en el lecciones de Andrew Ng (Standford), Razvan Pascanu(DeepMind), Jay Alammal y Pieter Abbel (Berkeley)

Overview

El Perceptron



Perceptrón: Algoritmo

$\mathbf{w} \in \mathbb{R}^n$, $b \in \mathbb{R}$, y^{ok} clase real $\in \{0, 1\}$, y^* clase predicha $\in \{0, 1\}$

Algorithm 1 pseudocodigo Perceptrón clasificación binaria

```
w = random()
while ¬convergencia do
    prodEsc = f(x) * w + b
    clasificar {1, 0} dependiendo de prodEsc
    actualizar w para mejorar predicción //Acerco—Alejo
end while
```

Perceptrón: Algoritmo

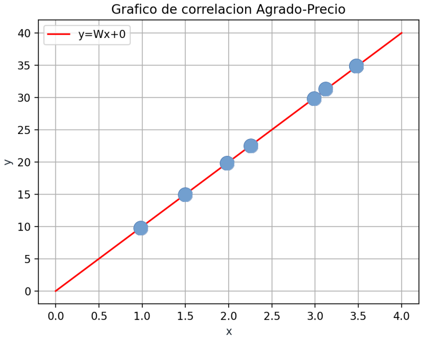
$\mathbf{w} \in \mathbb{R}^n$, $b \in \mathbb{R}$, y^{ok} clase real $\in \{0,1\}$, y^* clase predicha $\in \{0,1\}$

Algorithm 2 pseudocodigo en pasos genéricos Perceptrón clasificación binaria

$w = random()$
while $\neg$convergencia **do**
  $prodEsc = f(x) * w + b$
  if $prodEsc \geqslant 0$ **then**
    $y^* = 1$
  else
    $y^* = 0$
  end if
  $w = w + (y^{ok} - y^*) * f(x)$ //Acerco—Alejo según error
end while

Aprendizaje Supervisado: Estudio del precio del Chocolate

Nivel de Agrado	Dispuesto a Pagar
1	10
1.5	15
2	20
2.25	22.5
3	30
3.1	31
3.5	35
4	???

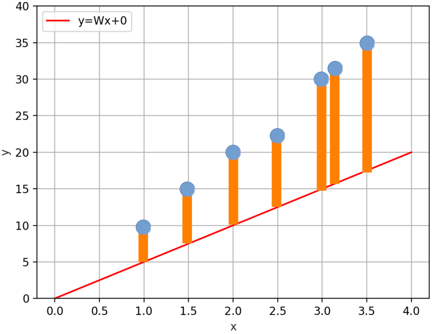


¿que valor está dispuesto a pagar el último cliente?
¿qué valor le corresponde a W?

[1] Ejemplo adaptado, fuente: Jay Alammar

Aprendizaje Supervisado: Estudio del precio del Chocolate

Nivel de Agrado	Dispuesto a Pagar
1	10
1.5	15
2	20
2.25	22.5
3	30
3.1	31
3.5	35
4	???



Si la inicialización random hubiese sido W=5 estaríamos cometiendo un error $(y - y^*)$. Y este error será el que guíe el aprendizaje como veremos

Perceptrón: Algoritmo

$\mathbf{w} \in \mathbb{R}^n$, $b \in \mathbb{R}$, y valor real $\in \mathbb{R}$, y^* valor predicho $\in \mathbb{R}$

Algorithm 3 pseudocodigo Regresión Lineal

$w = random()$
while $\neg$convergencia **do**
    $prodEsc = f(x) * w + b$
    //en Regresion sobra la step function
    if $prodEsc \geqslant 0$ **then**
        $y^* = 1$
    else
        $y^* = 0$
    end if
    $w = w + (y - y^*)^2 f(x)$ //Siempre acerco.
end while

Aprendizaje Supervisado: Escenario básico

- Siendo X el espacio de los valores de entrada.
- Siendo Y el espacio de los valores de salida.
- Para un determinado conjunto de datos $D \subset X \times Y$ encuentra la función h tal que

$$h : X \rightarrow Y$$

- h suele denominarse *hipótesis*
- La categorización del problema dependen del dominio de Y
  - si $Y \in \mathbb{R}$: es un problema de regresión
  - si $Y \in val1, val2...val_n$: es un problema de clasificación

[1] Transparencia de Razvan Pascanu

Mejorando las predicciones y su interpretación

- **Mejorando la función de activación** (en el caso de la clasificación)
- **Mejorando la actualización de pesos:** Evitar que sean bruscos, añadir generalización y obtener no solo un buen w sino el mejor.

Hacia la Regresión Logística: clasificación binaria

$\mathbf{w} \in \mathbb{R}^n$, $b \in \mathbb{R}$, y^{ok} clase real $\in \{0, 1\}$, y^* predicha $\in \{0, 1\}$

Algorithm 4 pseudocodigo a caballo entre Perceptrón y RLog

```
w = random()
while ¬convergencia do
    prodEsc = f(x) * w + b
    conf = 1 / (1 + e^(−prodEsc))   //sigmoide
    if conf ⩾ 0.5 then
        y* = 1
    else
        y* = 0
    end if
    w = w + (y − y*) * f(x)//Aunque mejorable, luego hablamos de cómo...
end while
```

Hacia la Regresión Logística: clasificación multiclase

$\mathbf{w} \in \mathbb{R}^n$, $b \in \mathbb{R}$, y^{ok} clase real $\in \mathbb{N}^k$, y^* clase predicha $\in \mathbb{N}^k$

Algorithm 5 pseudocod. a caballo entre Perceptrón y RLog multicl.

$w = random()$
while $\neg$convergencia **do**
    for $i = 1$ to k **do**
        $exp_i = e^{f(x)*w_i+b}$
        $softmax_i = \frac{exp_i}{\sum_1^k exp_i}$
    end for
    $y^* = \arg\max_i(softmax_i = \frac{exp_i}{\sum_1^k exp_i})$//mantengo el resto de momento
    if $y^* \neq y^{ok}$ **then**
        $w^{ok} = w^{ok} + f(x)$
        $w^* = w^* - f(x)$
    end if
end while

Hacia la Regresión Logística: clasificación multiclase

- ¿Puede ayudar actualizar los pesos aunque hayamos acertado?
- Es decir,¿puede ayudar plantearlo como regresión intentando aprender la distribución de probabilidades de las clases?

Aprendizaje Supervisado: Regresión Logística

Pred	Real	¿OK?
0.8	1	SI
0.9	1	SI
0.55	0	NO

Table: 2^{o} *Predictor*

Pred	Real	¿OK?
0.6	1	SI
0.55	1	SI
0.6	0	NO

Table: 1^{er} *Predictor*

No solo queremos medir si hemos acertado o no

Queremos medir cuánto nos hemos desviado y que la desviación guíe nuestro aprendizaje.

Mejorando las predicciones y su interpretación

Funciones medir la desviación (o error o coste): cross-entropy y error cuadrático (Mean Squared Error). Queremos minimizar nuestro error.

- **Cross-Entropy:**
  - **Clasificación binaria:** $-\frac{1}{N} y_i \log(\sigma_i) + (1 - y_i) \log(1 - \sigma_i)$
- **MSE:**
  - **Clasificación binaria:** $\frac{1}{2N}(\sigma_i - y_i)^2$

Calculémoslo para el 1er ejemplo de los predictores anteriores 1 y 2

Mejorando las predicciones y su interpretación

- Primera red 1er item de entrenamiento:
  - cross-entropy $:-\frac{1}{N}(y_i \log(\sigma_i) + (1-y_i)\log(1-\sigma_i))$
    $-(1 * log(0.6)) + ((1-1) * log(1-0.6)) =$
    $-log(0.6) + 0 = 0.22$
  - MSE $\frac{1}{2N}(\sigma i - y_i)^2$: $\frac{1}{2}(0.6-1)^2 = \frac{1}{2}(0.4^2) = 0.08$
- Segunda red 1er item de entrenamiento:
  - cross-entropy $:-y_i \log(\sigma_i) + (1-y_i)\log(1-\sigma_i)$
    $-(1 * log(0.8)) + ((1-1) * log(1-0.8)) =$
    $-log(0.8) + 0 = 0.09$
  - MSE $\frac{1}{2}(\sigma i - y_i)^2$: $\frac{1}{2}(0.8-1)^2 = \frac{1}{2}(0.2^2) = 0.02$

Aprendizaje Supervisado: Regresión Logística

Pred			Real			¿OK?
0.1	0.2	0.7	0	0	1	SI
0.1	0.7	0.2	0	1	0	SI
0.3	0.4	0.3	1	0	0	NO

Table: 2º *Predictor*

Pred			Real			¿OK?
0.3	0.3	0.4	0	0	1	SI
0.3	0.4	0.3	0	1	0	SI
0.1	0.2	0.7	1	0	0	NO

Table: 1er *Predictor*

Mejorando las predicciones y su interpretación

Funciones medir la desviación (o error o coste): cross-entropy y error cuadrático (Mean Squared Error). Queremos minimizar nuestro error.

- **Cross-Entropy:**
  - **Clasificación multiclase:** $\frac{-1}{N}\sum_{i=1}^{k=numClases} y_i \ln(softmax_i)$
- **MSE:**
  - **Clasificación multiclase:** $\frac{1}{2N}\sum_{i=1}^{k=numClases}(softmax_i - y_i)^2$

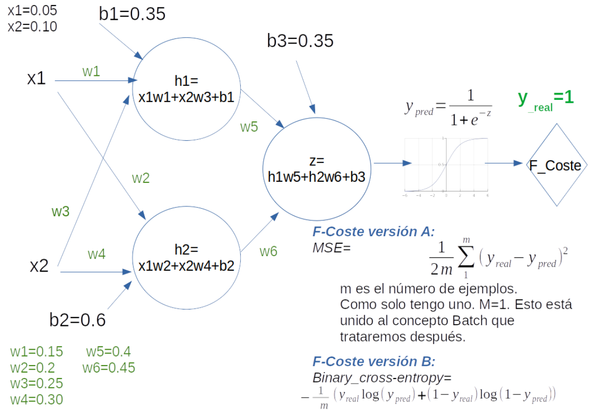
Calculémoslo para el 1er ejemplo de los predictores anteriores 1 y 2

Mejorando las predicciones y su interpretación

- Primera red 1er item de entrenamiento:
  - cross-entropy $\frac{1}{N}\sum_{i=1}^{k} -y_i \ln(exp_i)$:
    $-((ln(0.3)*0)+(ln(0.3)*0)+(ln(0.4)*1)) = -ln(0.4) = 0.9163$
  - MSE $\frac{1}{2M}\sum_{i=1}^{k}(exp_i - y_i)^2$:
    $(0.3-0)^2+(0.3-0)^2+(0.4-1)^2 = 0.09+0.09+0.36 = \frac{1}{2}*0.54$
- Segunda red 1er item de entrenamiento:
  - cross-entropy:
    $-\frac{1}{N}*((ln(0.1)*0)+(ln(0.2)*0)+(ln(0.7)*1)) = -ln(0.7) = 0.3566$
  - MSE: $\frac{1}{2M}*((0.1-0)^2+(0.2-0)^2+(0.7-1)^2) = \frac{1}{2}*(0.01+0.04+0.09) = \frac{1}{2}*0.14$

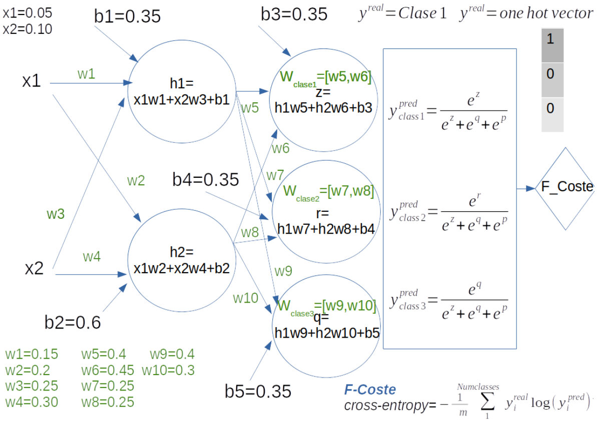
Mejorando las predicciones y su interpretación

Ejercicio clasificación binaria: predicción en alimentación hacia adelate (feed-forward)



Intro AS
A. Atutxa
Recapitulando sobre perceptrón
Mejorando la interpretación
Mejorando las actualizaciones

Ejercicio clasificación multiclase: predicción en alimentación hacia adelate (feed-forward)



Demo

`http://playground.tensorflow.org/`

Mejorando las predicciones y su interpretación

Actualización de pesos: Evitar que sean bruscos, añadir generalización y obtener no solo un buen *w* sino el mejor.

- **MIRA:** necesidad de añadir un elemento que minimice la actualización manteniendo el poder predictor. Concepto τ (similar al learning rate η)
- **Average Perceptrón:** Mejorar la generalización, suavizando el impacto de cada ejemplo sobre la actualización.
- ¿Se puede hacer aún mejor?

Mejorando las predicciones y su interpretación

Actualización de pesos: Evitar que sean bruscos, añadir generalización y obtener no solo un buen *w* sino el mejor.

- **MIRA:** necesidad de añadir un elemento que minimice la actualización manteniendo el poder predictor. Concepto τ (similar al learning rate η)
- **Average Perceptrón:** Mejorar la generalización, suavizando el impacto de cada ejemplo sobre la actualización.
- ¿Se puede hacer aún mejor?

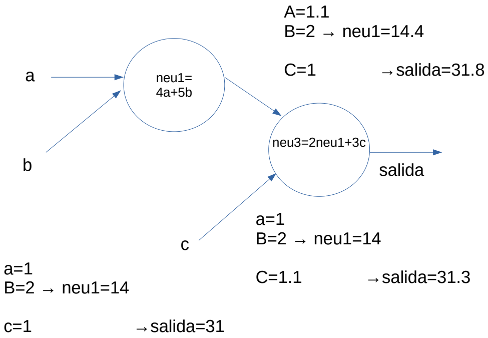
  Descenso del Gradiente (Gradient Descent)

Descenso del Gradiente

- Tengo una función que me mide la cantidad de error. Cross-Entropy.
- Es una función derivable.
- La derivada mide la rapidez y la dirección del cambio de la función con respecto al cambio de una variable de esa función.

Un vídeo que explica muy bien la intuición es: `https://www.youtube.com/watch?v=tIeHLnjs5U8`

Descenso del Gradiente



Derivada de la función coste

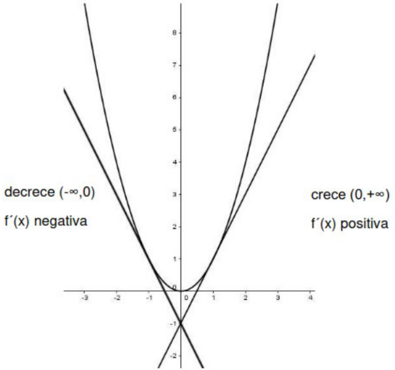
$$f'(w) = \lim_{h \to 0} \frac{f(w+h) - f(w)}{h}$$

$$\mathrm{w} \leftarrow w - \eta * \nabla_w funcCoste(w)$$

$$\begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix} \text{ - } \eta * \begin{bmatrix} \frac{\partial funcCoste}{\partial w_1} \\ \frac{\partial funcCoste}{\partial w_2} \\ \dots \\ \frac{\partial funcCoste}{\partial w_n} \end{bmatrix}$$

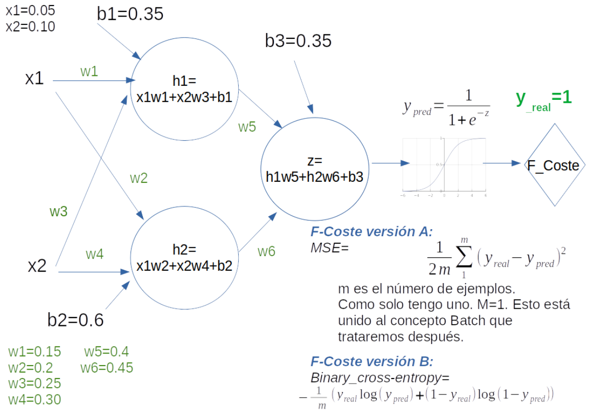
Derivada de la función de coste

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$



Mejorando las predicciones y su interpretación

Ejercicio clasificación binaria: predicción en alimentación hacia adelate (feed-forward)



Mejorando las predicciones y su interpretación

Queremos actualizar w_5 en base a lo que ha influido en la desviación que se ha producido. ¿Cómo medimos la "aportación de w_5 en la variación de la función de Coste"? La derivada mide eso, el ratio de cambio. $\frac{\delta F-Coste}{\delta w_5}$

Se aplica la regla de la cadena:
$\frac{\delta F-Coste}{\delta w_5}=\frac{\delta F-Coste}{\delta \sigma(z)}\frac{\delta \sigma(z)}{\delta z}\frac{\delta z}{\delta w_5}$

Derivada de la función de Coste

- Si la función de coste hubiese sido MSE, la derivada sería $\frac{1}{2}2(y - \sigma(z))$
- Si la función de coste hubiese sido Binary Cross-Entropy, la derivada sería $-\frac{y}{\sigma(z)} + \frac{1-y}{1-\sigma(z)}$

El resto de las derivadas son las mismas independientemente de la función de coste empleada. Es decir, $\frac{\delta\sigma(y)}{\delta y}\frac{\delta y}{\delta w_5}$

Derivada de la función sigmoide

Intro AS

A. Atutxa

Recapitulando sobre perceptrón

Mejorando la interpretación

Mejorando las actualizaciones

$$\frac{\delta\sigma(z)}{\delta z} = \frac{\frac{\delta numerador}{\delta(z)} * denominador - \frac{\delta denominador}{\delta(z)} * numerador}{(1+e^{-z})^2}$$

$$\frac{\delta\sigma(z)}{\delta z} = \frac{0*(1+e^{-z}) - (-e^{-z})*1}{(1+e^{-z})^2} = \frac{e^{-z}}{(1+e^{-z})^2}$$

Truco para intentar simplificarlo en base a datos que ya hemos calculado en el forward

$$\frac{\delta\sigma(z)}{\delta z} = \frac{e^{-z}+1-1}{(1+e^{-z})^2} = \frac{e^{-z}+1}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2} = \frac{1}{(1+e^{-z})} - \frac{1}{(1+e^{-z})^2} = \frac{1}{(1+e^{-z})} * \left(1 - \frac{1}{(1+e^{-z})}\right) = \sigma(z) * (1 - \sigma(z))$$

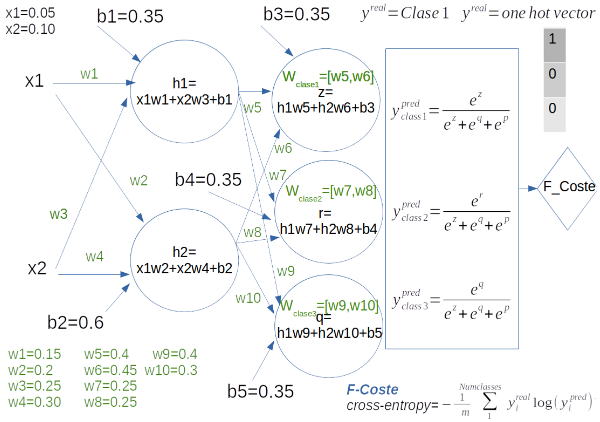
Derivada de la función sigmoide

$$\frac{\delta z}{\delta w_5} = h1$$

$$(-\frac{y}{\sigma(z)} + \frac{1-y}{1-\sigma(z)}) * (\sigma(z) * (1 - \sigma(z)) * h1$$

Ejercicio: Feedforward

Ejercicio clasificación multiclase: predicción en alimentación hacia adelate (feed-forward)



Mejorando las predicciones y su interpretación

Como antes con la clasificación binaria, en la clasificación multiclase también querremos actualizar w_5 en base a lo que ha influido en la desviación que se ha producido. Igual que antes, la derivada mide eso, el ratio de cambio. $\frac{\delta F-Coste}{\delta w_5}$

Ejercicio: Backpropagation

Vamos a ver cómo calcular la derivada de la función de coste que en este caso será la cross-entropy.

Cross-entropy[1]: $\sum_{i=1}^{k=numClases} -y_i \ln(softmax_i)$

Para empezar la derivada de un sumatorio es el sumatorio de las derivadas. Hemos puesto un ejemplo concreto donde queremos derivar sobre w_5 para calcular la actualización de ese parámetro.

[1] Recordad que w_5 participa en el cálculo de e^{p_1}, e^{p_2} y e^{p_3} y estos en el cálculo de todos los $softmax_i$, en todos los denominadores $(e^{p_1} + e^{p_2} + e^{p_3})$ y en un numerador.

Ejercicio: Backpropagation

Vamos a ver cómo calcular la derivada de la función de coste que en este caso será la cross-entropy.

Cross-entropy[2]: $\sum_{n=1}^{k=numClases} -y_i \ln(softmax_i)$

Para empezar la derivada de un sumatorio es el sumatorio de las derivadas con su correspondiente regla de la cadena

$$\sum_{i=1}^{k=numClases} -y_i \frac{\delta \ln(softmax(p_i))}{\delta(softmax(p_i))} \frac{\delta softmax(p_i)}{\delta p_1} \frac{\delta(p_1)}{\delta w_5}$$

[2] Recordad que w_5 participa en el cálculo de e^{p_1}, e^{p_2} y e^{p_3} y estos en el cálculo de todos los $softmax_i$, en todos los denominadores $(e^{p_1} + e^{p_2} + e^{p_3})$ y en un numerador.

Ejercicio: Backpropagation

Vamos a ver cómo calcular la derivada de la función de coste que en este caso será la cross-entropy.

Cross-entropy: $\sum_{n=1}^{k=numClases} -y_i \ln(softmax_i)$

Para empezar la derivada de un sumatorio es el sumatorio de las derivadas con su correspondiente regla de la cadena $\sum_{i=1}^{k=numClases} -y_i \frac{\delta \ln(softmax(p_i))}{\delta(softmax(p_i))} \frac{\delta softmax(p_i)}{\delta p_1} \frac{\delta(p_1)}{\delta w_5}$

Ejercicio: Backpropagation

Intro AS

A. Atutxa

Recapitulando sobre perceptrón

Mejorando la interpretación

Mejorando las actualizaciones

Sabiendo que la derivada de $ln(x) = \frac{1}{x}$
$\sum_{i=1}^{k=numClases} \frac{-y_i}{softmax(p_i)} \frac{\delta softmax(p_i)}{\delta p_1} \frac{\delta(p_1)}{\delta w_5}$

queda saber 1.$\frac{\delta softmax(p_i)}{\delta p_1}$ y 2.$\frac{\delta(p_1)}{\delta w_5}$

- Respecto al 1. $\rightarrow$ la derivada de $\frac{\delta softmax(p_i)}{\delta p_1}$

  Cuando $i = 1$ será $\frac{\delta(\frac{e^{p_1}}{(e^{p_1}+e^{p_2}+e^{p_3})})}{\delta p_1}$

  Cuando $i \neq 1$ será $\frac{\delta(\frac{e^{p_{j=2,3}}}{(e^{p_1}+e^{p_2}+e^{p_3})})}{\delta p_1}$

Ejercicio: Backpropagation

Cuando $i = 1$ la derivada de $\frac{\delta softmax(p_i)}{\delta p_1}$ será $\frac{\delta(\frac{e^{p_1}}{(e^{p_1}+e^{p_2}+e^{p_3})})}{\delta p_1}$

Aplicando la regla del cociente

$$\frac{\delta(\frac{e^{p_1}}{(e^{p_1}+e^{p_2}+e^{p_3})})}{\delta p_1} = \frac{\frac{\delta numerador}{\delta(p_1)} * denominador - \frac{\delta denominador}{\delta(p_1)} * numerador}{(denominador)^2}$$

$$= \frac{e^{p_1} * (e^{p_1}+e^{p_2}+e^{p_3}) - e^{p_1} * e^{p_1}}{(e^{p_1}+e^{p_2}+e^{p_3})^2} =$$

$$\frac{e^{p_1}}{(e^{p_1}+e^{p_2}+e^{p_3})} * \frac{(e^{p_1}+e^{p_2}+e^{p_3}) - e^{p_1}}{(e^{p_1}+e^{p_2}+e^{p_3})}$$

$$= softmax(p_1) * (1 - softmax(p_1))$$

Cuando $i \neq 1$ la derivada de $\frac{\delta(\frac{e^{p_{j=2,3}}}{(e^{p_1}+e^{p_2}+e^{p_3})})}{\delta p_1}$

Ejercicio: Backpropagation

Intro AS

A. Atutxa

Recapitulando sobre perceptrón

Mejorando la interpretación

Mejorando las actualizaciones

Cuando $i \neq 1$ la derivada de $\frac{\delta softmax(j{=}2{,}3)}{\delta p_1}$ será

$$\frac{\delta\left(\frac{e^{p_{j=2,3}}}{(e^{p_1}+e^{p_2}+e^{p_3})}\right)}{\delta p_1}$$

Aplicando la regla del cociente i fijando j=2 (con j=3 será igual)

$$\frac{\delta\left(\frac{e^{p_{j=2,3}}}{(e^{p_1}+e^{p_2}+e^{p_3})}\right)}{\delta p_1} = \frac{\frac{\delta numerador}{\delta(p_1)} * denominador - \frac{\delta denominador}{\delta(p_1)} * numerador}{(denominador)^2}$$

$$= \frac{0*(e^{p_1}+e^{p_2}+e^{p_3}) - e^{p_1} * e^{p_2}}{(e^{p_1}+e^{p_2}+e^{p_3})^2} = -\frac{e^{p_1}}{(e^{p_1}+e^{p_2}+e^{p_3})} * \frac{e^{p_2}}{(e^{p_1}+e^{p_2}+e^{p_3})}$$

$$= -softmax(p_1) * softmax(p_2)$$

Ejercicio: Backpropagation

Juntándolo todo:

$$(\sum_{i=2}^{k=numClases} \frac{-y_i}{softmax(p_i)} * -softmax(p_i) * softmax(p_1)) + \frac{-y_1}{softmax(p_1)} * softmax(p_1)(1 - softmax(p_1)))\frac{\delta(p_1)}{\delta w_5}$$

- Los softmax redundantes se eliminan

$$(\sum_{i=2}^{k=numClases} -y_i * softmax(p_1)) - y_1(1 - softmax(p_1)))\frac{\delta(p_1)}{\delta w_5}$$

- el $softmax(p_1)$ se saca fuera del sumatorio

$$(softmax(p_1)\sum_{i=2}^{k=numClases} -y_i) - y_1(1 - softmax(p_1)))\frac{\delta(p_1)}{\delta w_5}$$

- y recordar que $\sum_{i=1}^{k=numClases} -y_i) = 1$
- y de esto se sigue que $\sum_{i=2}^{k=numClases} -y_i = 1 - y_1$
- Reescribiendolo quedaría:

$$(softmax(p_1) * (1 - y_1) - y_1(1 - softmax(p_1)))\frac{\delta(p_1)}{\delta w_5}$$

- Simplificando: $(softmax(p_1) - y_1)\frac{\delta(p_1)}{\delta w_5}$

Actualización por lotes (batch), Estocástica o por Minilotes

¿Cúando realizo la actualización?

- por cada item de entrenamiento (Stochastic Gradient Descent). Desventaja: No me permiten vectorizar
- al final de un epoch (Batch Gradient Descent) Desventaja: Si tengo muchos datos hasta el final de cada epoch no actualizo.
- cada J ejemplos (Mini-batch Gradient Descent) Ventaja: Es un compromiso entre los dos anteriores

Se recomienda visualizar el vídeo de Andrew Ng colgado en el servidor. `http://lsi.bp.ehu.es/asignaturas/TIA/`

Más cuestiones técnicas

- Dropout
- ReLu
- Learning Rate
- Diferencia entre parámetros e hiperparámetros: Conjunto de validación versus conjunto de test

Aspectos técnicos importantesi

- Dropout

  El sobreajuste es un problema grave a medida que vamos añadiendo capas a nuestra red y neuronas en cada capa. El dropout es una técnica que permite abordar este problema. La idea clave es **desactivar aleatoriamente** neuronas (junto con sus conexiones) durante el entrenamiento. Así, conseguimos que todas las neuronas no se entrenen con todos los ejemplos de entrenamiento dado que se habrán desactivado con alguno de los ejemplos. Es como simulásemos distintos entrenamientos para cada neurona dado que cuando se ha desactivado no se ha entrenado sobre ese ejemplo.

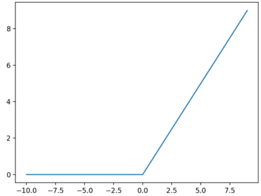
Aspectos técnicos importantes

- ReLU: Introduciendo no linearidad.

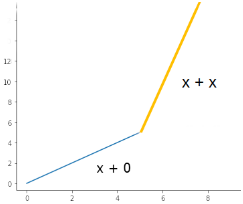
  Las unidades ReLu introducen no linearidad, permitiendo la adaptación a cualquier contorno no lineal. Se define como max (0, x). Mirar explicación del enunciado del labo

Más cuestiones técnicas

ReLu



Supongamos que conectamos la salida x a una unidad ReLu de forma x+ReLu(x-5), entonces la nueva salida sería



Aspectos técnicos importantes

- Learning Rate o Tasa de Aprendizaje α (otras veces se le llama η

  La tasa de aprendizaje es un hiperparámetro que permite **regular** cuánto ajustamos los pesos de nuestro modelo.

  $w_i = w_i + \alpha(\frac{\delta F - Coste}{\delta w_5})$

  Recordad que queremos evitar actualizaciones bruscas.
  Hay que hacer un barrido de valores normalmente entre 0.1 y 0.0001 para encontrar el valor que nos permita regular mejor la actualización

Aspectos técnicos importantes

- Diferencia entre parámetros e hiperparámetros:
  - Los parámetros son los pesos que vamos actualizando AUTOMATICAMENTE a través de las iteraciones.
  - Los hiperparámetros son los valores que nos permiten generar distintos modelos. Por ejemplo, el número de iteraciones, el porcentage de dropout, el learning-rate....

Aspectos técnicos importantes

- El número de iteraciones es un hiperparámetro, a veces en vez de decir explicitamente cuantas iteraciones (epochs) se harán sobre el train, se marca cuantas iteraciones sin cambio en la tasa de error para decidir cuando parar.
- El learning rate es otro hiperparámetro cuyos valores a probar suelen ir de 0.1 a 0.0001

Haremos un barrido de estos hiper parámetros para generar distintos modelos. Si vemos que el número de combinaciones es muy grande, se puede hacer un barrido random, es decir generar conbinaciones de forma random y una vez que tengamos la mejor combinación, afinar el barrido.

Aspectos técnicos importantes

- Cada modelo generado con los barridos debe ser evaluado. Para eso es necesario disponer de **set de validación** (para medir que modelo es el mejor). Es importante que también exista un **set de test** (el set que me permitirá hacer la evaluación definitiva y honesta) que permitirá a otros científicos o modelos compararse con el que se ha construido.

Aspectos técnicos importantes

- Conclusión si queremos ajustar hiperparámetros: Dividiremos siempre que tengamos hiperparámetros nuestros datos en **3 conjuntos** train, validation y test. Hay que asegurarse de que la división se hace de manera random y que se mantiene la distribución de los datos originales.

  Métodos como *StratifiedShuffleSplit* de sklearn nos permiten obtener los sets. *StratifiedShuffleSplit* es una combinación de *ShuffleSplit* y *StratifiedKFold* de forma que la proporción de distribución de etiquetas de clase es casi uniforme entre los conjuntos generados.

Vídeos interesantes

En la url `http://lsi.bp.ehu.es/asignaturas/TIA/` encontraréis vídeos sobre el gradiente y otros conceptos que se han trabajado en este tema. Los vídeos son de Andrew Ng que es uno de los precursores de Coursera y de los cursos de ML. Os recomiendo que los véais.
`https://www.youtube.com/watch?v=tIeHLnjs5U8`