Clase 7: Redes Neuronales Artificiales Conceptos Preliminares

Marcelo Luis Errecalde[1,2]

[1]Universidad Nacional de San Luis, Argentina
[2]Universidad Nacional de la Patagonia Austral, Argentina
e-mails: merreca@unsl.edu.ar, merrecalde@gmail.com

Curso: Minería de Datos
Universidad Nacional de San Luis - Año 2018

Resumen

Repaso: Modelos Lineales

Modelos lineales: hacen sus predicciones usando una función lineal de las características de entrada del tipo:

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

Se suele incorporar el bias (b) como un parámetro más (w_0) cuya entrada es seteada siempre a la unidad (1):

$$z = \left(\sum_{i=0}^{n} w_i x_i \right) = w_0 + w_1 x_1 \ldots w_n x_n$$

Repaso: Modelos Lineales

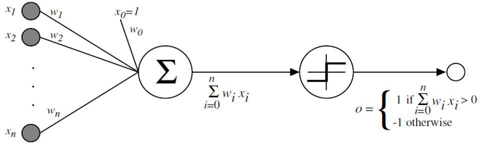
Esta función lineal cumplirá distintos roles dependiendo del tipo de predicción esperada:

- **Numérica** (regresión lineal): la salida (z) es el valor a predecir (un número) y es una función lineal de las características: una línea, un plano o un hiperplano
- **Categórica** (clasificación lineal): la salida (z) también es una función lineal de las características (una línea, un plano o un hiperplano) pero es utilizado como un límite/superficie de decisión. En otras palabras, un clasificador lineal (binario) es un clasificador que separa dos clases usando una línea, un plano o un hiperplano.

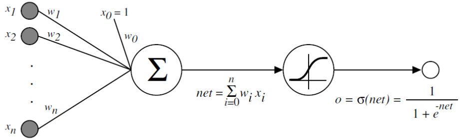
Modelos lineales y tipos de unidades

La forma en que el valor z es usado para establecer un límite de decisión da origen a distintos tipos de unidades:

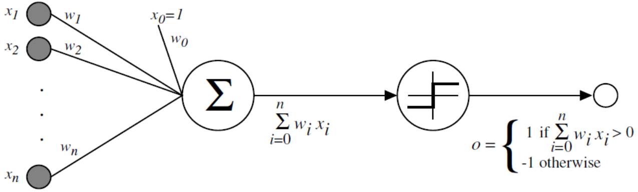
- Unidad con **umbral "duro"** (perceptrón)



- Unidad **lineal** (adaline)
- Unidad **sigmoide** (regresión logística)



El Perceptron

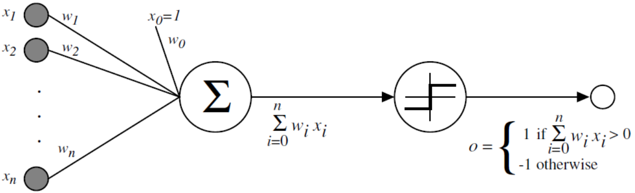


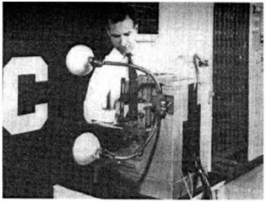
$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{si } w_0 + w_1 x_1 + \ldots + w_n x_n > 0 \\ -1 & \text{en otro caso} \end{cases}$$

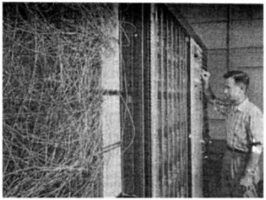
O en notación de vector más simple:

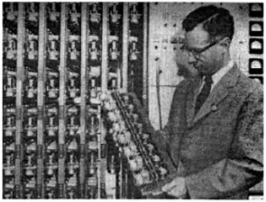
$$o(\vec{x}) = \begin{cases} 1 & \text{si } \vec{x} \cdot \vec{w} > 0 \\ -1 & \text{en otro caso} \end{cases}$$

El Perceptron .... real

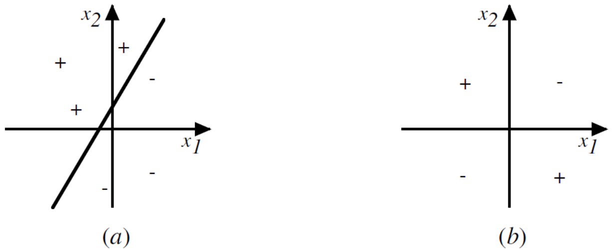








Límite de decisión de un perceptron



Permite representar algunas funciones útiles

- ¿ Qué pesos representan $g(x_1, x_2) = AND(x_1, x_2)$?

Pero algunas funciones no son representables: ...

- Ejemplo: las que no son linealmente separables

Regla de entrenamiento del perceptron

$$w_i \leftarrow w_i + \Delta w_i$$

donde

$$\Delta w_i = \eta(t - o)x_i$$

Donde: ...

- $t = c(\vec{x})$ es el valor objetivo ("target")
- o es la salida del perceptron ("output")
- η es una constante pequeña (p.ej 0.1) llamada taza de aprendizaje ("learning rate")

Regla de entrenamiento del perceptron

Se puede probar que convergerá si: ...

- Los datos de entrenamiento son linealmente separable
- y η es suficientemente pequeña

Unidad lineal y descenso del gradiente

Consideremos ahora una unidad lineal más simple donde

$$o = w_0 + w_1 x_1 \ldots w_n x_n$$

- Sería equivalente a la primera parte de salida del perceptron (sin “tresholding”)
- Es el tipo de ecuación que utilizamos con regresión lineal

y aprendamos los w_i's que minimizan el error cuadrado

$$E[\vec{w}] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

donde D es un conjunto de ejemplos de entrenamiento

Optimización

Encontrar los pesos $\vec{w}$ que minimizan un error (o pérdida) es un problema de optimización, que podría ser abordado de distintas maneras:

- Búsqueda aleatoria
- Búsqueda local aleatoria
- Descenso del gradiente

Búsqueda aleatoria

Muy simple (pero mala) idea:

- Probar muchos pesos $\vec{w}$ aleatorios
- Mantener el que anduvo mejor
- Con este enfoque se obtiene una accuracy $\approx$ 15.5 (en un problema con 10 clases)
- Mejor idea: Partir con un $\vec{w}$ aleatorio inicial y luego ir refinándolo en forma iterativa, mejorándolo a lo largo del tiempo para obtener una pérdida menor.

Analogía: el senderista vendado

Búsqueda local aleatoria

Mejor idea que la anterior:

- Desde mi posición actual (pesos $\vec{w}$) extiendo un pie en una dirección aleatoria
- Hago un paso en esa dirección, sólo si conduce colina abajo
- Implementación: Dado un $\vec{w}$ inicial, genero perturbaciones aleatorias $\delta\vec{w}$
- Si $\vec{w} + \delta\vec{w}$ genera menor error, uso estos pesos perturbados

Descenso del gradiente

No movernos localmente en una dirección aleatoria sino en la dirección que produce el descenso más pronunciado

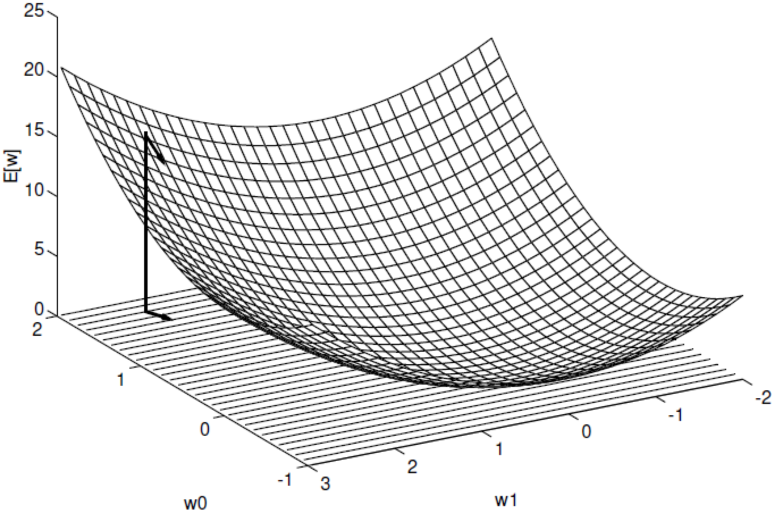
- El enfoque usual para determinar la pendiente más pronunciada (en una dimensión) es la derivada de una función:

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

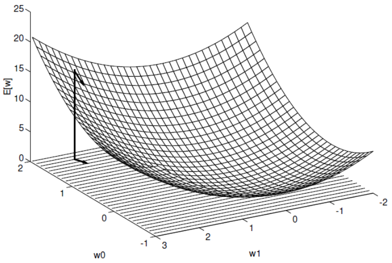
- Cuando la función toma un vector $\vec{w}$ como entrada, se generaliza a la idea de gradiente: vector de derivadas parciales en cada dimensión

$$\nabla f[\vec{w}] \equiv \left[\frac{\partial f}{\partial w_0}, \frac{\partial f}{\partial w_1}, \ldots, \frac{\partial f}{\partial w_n} \right]$$

Función de error y descenso del gradiente



¿Como calcular el gradiente?



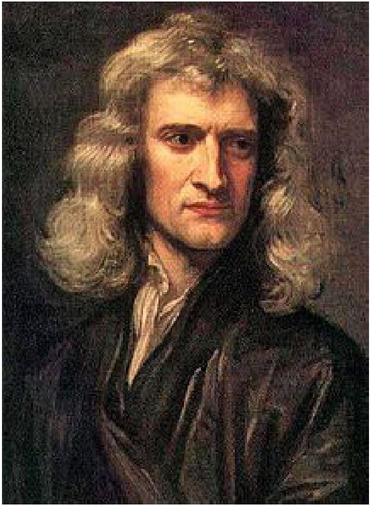
- Numéricamente (con diferencias finitas)
- Analíticamente (mediante las herramientas clásicas del cálculo)

Cómputo del gradiente numérico

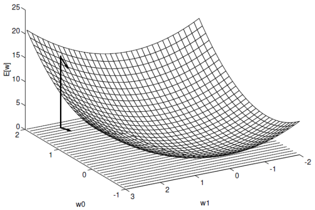
- Hace una aproximación a la definición de derivada basada en el límite.
- Se toma un h muy pequeño (ej. $h = 0.00001$) y por cada w_i se calcula el error para $f(w)$ y para el f con ese peso incrementado en h y se computa el $\frac{f(x+h)-f(x)}{h}$ para ese peso.
- Método sencillo ...
- ... pero lento

.... y ahora, quien podrá ayudarnos ?????

los reyes del cálculo!!! ⇒ gradiente analítico



Descenso del gradiente



Gradiente

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, , \frac{\partial E}{\partial w_n} \right]$$

Regla de entrenamiento:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

o sea

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Descenso del gradiente

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

operando ...

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{i,d})$$

o sea ...

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{i,d}$$

Descenso del gradiente (algoritmo)

DESCENSO-GRADIENTE(D, η)

Cada ejemplo de entrenamiento es un par de la forma $\langle \vec{x}, t \rangle \in D$, donde $\vec{x}$ es el vector de valores de entrada y t es el valor de salida objetivo. η es la taza de aprendizaje (ej, 0.05)

- Inicializar cada w_i con algún valor aleatorio pequeño
- Hasta que se cumpla la condición de terminación
  - Inicializar cada Δw_i en 0.
  - Por cada $\langle \vec{x}, t \rangle \in D$
    - Presentar la instancia $\vec{x}$ a la unidad y computar la salida o
    - Por cada peso w_i de la unidad lineal,

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

  - Por cada peso de la unidad lineal w_i,

$$w_i \leftarrow w_i + \Delta w_i$$

Resumen

La regla de entrenamiento del perceptron garantiza el éxito si

- Los ejemplos de entrenamiento son linealmente separables
- Taza de aprendizaje η suficientemente pequeña

La regla de entrenamiento de la unidad lineal usa descenso del gradiente

- Garantiza la convergencia a las hipótesis con mínimo error cuadrado
- Dada una taza de aprendizaje η suficientemente pequeña
- Aún cuando los datos de entrenamiento contienen ruido
- Aún cuando los datos de entrenamiento no son separables por H

Descenso del gradiente (estocástico) incremental

DESCENSO-GRADIENTE **modo Batch**

Hacer hasta satisfacer las condiciones

1. Computar el gradiente $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

DESCENSO-GRADIENTE **modo incremental**

Hacer hasta satisfacer las condiciones

- Por cada ejemplo de entrenamiento $d \in D$
  1. Computar el gradiente $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \quad \text{versus} \quad E_d[\vec{w}] \equiv \frac{1}{2} (t_d - o_d)^2$$

Descenso del gradiente (estocástico) incremental

DESCENSO-GRADIENTE modo Batch
Hacer hasta satisfacer las condiciones

1. Computar el gradiente $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

DESCENSO-GRADIENTE modo incremental
Hacer hasta satisfacer las condiciones

- Por cada ejemplo de entrenamiento $d \in D$
  1. Computar el gradiente $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

El descenso del gradiente incremental puede aproximar descenso del gradiente Batch, arbitariariamnte cercano en la medida que η es hecha suficientemente pequeña

Unidad de perceptron y unidad lineal: consideraciones finales

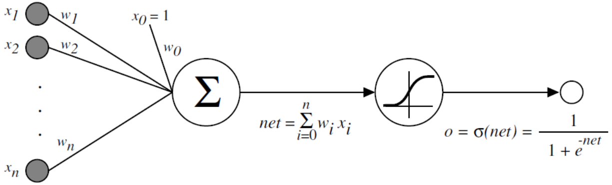
Los dos tipos de unidades se podrían usar en estructuras de múltiples niveles (capas) para obtener superficies de decisión más complejas (no lineales). Problemas:

- Múltiples capas de unidades lineales en cascada sólo producen funciones lineales
- La unidad del perceptron introduce una no linealidad pero su umbral discontinuo la hace no diferenciable (no apta para descenso del gradiente)

Se necesita una unidad cuya salida es una función no lineal de sus entradas, y simultáneamente es una función diferenciable de sus entradas ..... $\Rightarrow$

Unidad Sigmoide !!!!

Unidad Sigmoide



$\sigma(x)$ es la función sigmoide: $\frac{1}{1+e^{-x}}$

Propiedad útil: $\frac{\partial\sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$

Podemos derivar reglas del descenso del gradiente para entrenar

- Una unidad sigmoide (regresión logística)
- Redes multicapas de unidades sigmoides (backprop.)

Regresión Logística

- Clasificador probabilístico (aunque se llame “regresión) que genera un modelo discriminativo
- Usa una unidad sigmoide que toma como entrada el producto punto de su vector de entrada y los pesos de las features (un número real arbitrario)

$$z = \left(\sum_{i=0}^{n} w_i x_i \right) = w_0 + w_1 x_1 \ldots w_n x_n$$

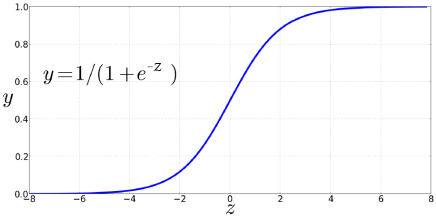
- ... y genera como salida un número entre 0 y 1, utilizando la función sigmoide (o función logística): $y = \sigma(z) = \frac{1}{1+e^{-z}}$

Regresión Logística

- Clasificador probabilístico (aunque se llame “regresión) que genera un modelo discriminativo
- Usa una unidad sigmoide que toma como entrada el producto punto de su vector de entrada y los pesos de las features (un número real arbitrario)

$$z = \left(\sum_{i=0}^{n} w_i x_i \right) = w_0 + w_1 x_1 \ldots w_n x_n$$

- Salida: $y = \sigma(z) = \frac{1}{1+e^{-z}}$



Regresión Logística

- En el escenario standard (caso binario), dado un ejemplo de test x, se calcula la probabilidad $p(y = 1|x)$ (probabilidad de que x pertenezca a la clase positiva):

$$P(y = 1|x) = \sigma\left(\sum_{i=0}^{n} w_i x_i\right) = \frac{1}{1 + e^{-(\sum_{i=0}^{n} w_i x_i)}}$$

- .... y que pertenezca a la clase negativa ($p(y = 0|x)$) =

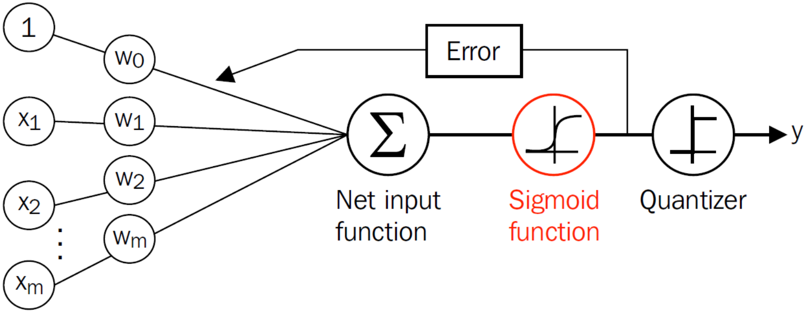
$$P(y = 0|x) = 1 - \sigma\left(\sum_{i=0}^{n} w_i x_i\right) = \frac{e^{-(\sum_{i=0}^{n} w_i x_i)}}{1 + e^{-(\sum_{i=0}^{n} w_i x_i)}}$$

- La salida (o) es la clase más probable:

$$o(x) = \begin{cases} 1 & \text{si } P(y = 1|x) > 0.5 \\ 0 & \text{en otro caso} \end{cases}$$

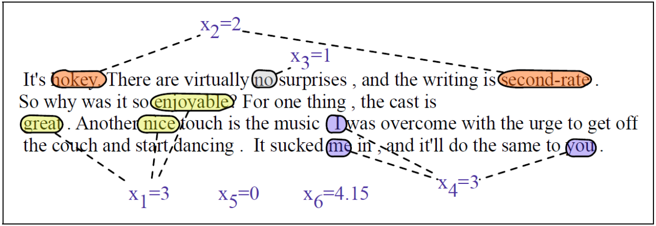
Regresión Logística

Gráficamente:



Ejemplo: análisis de sentimiento

“Mini” documento de test: extracto de crítica



Features del documento:

Var	Definition	Value
x_1	count(positive lexicon $\in$ doc)	3
x_2	count(negative lexicon $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

Ejemplo: análisis de sentimiento

Features del documento:

Var	Definition	Value
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

Supongamos ya se aprendieron los siguientes pesos: $[w_1, w_2, w_3, w_4, w_5, w_6] = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$ y $w_0 = 0.1$

Ejemplo: w_1 indica cuan importante es el número de palabras positivas del lexicon (*great*, *nice*, *enjoyable*, etc) para una decisión de sentimiento positivo

Ejemplo: análisis de sentimiento

$$
\begin{aligned}
p(+|x) = P(y=1|x) &= \sigma\left(\sum_{i=0}^{n} w_i x_i\right) \\
&= \sigma([0.1, 2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot \\
&\quad [1, 3, 2, 1, 3, 0, 4.15]) \\
&= \sigma(1.805) \\
&= \frac{1}{1+e^{-1.805}} \\
&= 0.86
\end{aligned}
$$

$$
p(-|x) = P(y=0|x) = 1 - \sigma\left(\sum_{i=0}^{n} w_i x_i\right) = 0.14
$$

Aprendizaje en Regresión Logística

Al igual que antes, tendremos que:

- Especificar la función de pérdida o función de costo
- Determinar el gradiente de esta función para actualizar (aprender) los pesos

Como función de costo usaremos la **pérdida de entropía cruzada** (en inglés **cross entropy loss**)

Pérdida de entropía cruzada (PEC)

Esto normalmente se expresa como un error L y la idea es ajustar esos pesos para que ese error vaya disminuyendo.

- Como toda función de pérdida $L(\hat{y}, y)$, ésta debe reflejar cuanto difiere la salida del modelo $\hat{y}$ (antes denotada o) de la verdadera salida y (antes denotada t)
- Una forma (usada en regresión lineal y la unidad lineal) sería usar el error cuadrado medio L_{MSE} entre $\hat{y}$ y y:

$$L_{MSE}(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2$$

y la pérdida (error) sobre todos los ejemplos de entrenamiento seria:

$$L[\vec{w}] = \frac{1}{2}\sum_{d \in D}(y_d - \hat{y}_d)^2$$

Pérdida de entropía cruzada (PEC)

- Sin embargo, L_{MSE} se torna difícil de optimizar cuando se aplica a clasificación probabilística.
- Más apropiada, es visualizar la clasificación como un experimento con distribución Bernoulli donde la función de probabilidad es $p(x) = p^x(1-p)^{1-x}$ y $x \in \{0, 1\}$. En este caso, la probabilidad de éxito (resultado es 1) es p y de falla es $q = 1 - p$.
- Ahora, se buscan los pesos que maximicen la probabilidad del rótulo correcto $p(y|x)$ dada la observación x:

$$p(y|x) = \hat{y}^y(1-\hat{y})^{1-y}$$

- Esta fórmula de la probabilidad que nuestro clasificador asigna a un rótulo ($y = 0$ o $y = 1$), reduce a $\hat{y}$ cuando $y = 1$ y $1 - \hat{y}$ cuando $y = 0$

Pérdida de entropía cruzada (PEC)

- Aplicando log a ambos lados:

$$\begin{aligned}\log p(y|x) &= \log[\hat{y}^y(1-\hat{y})^{1-y}] \\ &= y\log\hat{y} + (1-y)\log(1-\hat{y})\end{aligned}$$

- Esta ecuación debería ser maximizada, o bien minimizada si invertimos el signo y la convertimos en la pérdida de entropía cruzada (L_{CE})

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y\log\hat{y} + (1-y)\log(1-\hat{y})]$$

- Si incorporamos el hecho de que $\hat{y} = \sigma(\sum_i w_i x_i)$ tenemos

$$L_{CE}(w) = -[y\log\sigma\left(\sum_i w_i x_i\right) + (1-y)\log(1-\sigma\left(\sum_i w_i x_i\right))]$$

Pérdida de entropía cruzada (PEC)

Ahora podemos definir el costo (J) para todo el data set con los pesos w como la pérdida promedio de todos los ejemplos:

$$
\begin{aligned}
J(w) &= \frac{1}{m}\sum_{j=1}^{m} L_{CE}(\hat{y}^{(j)}, y^{(j)}) \\
&= -\frac{1}{m}\sum_{j=1}^{m}[y^{(j)}\log\sigma(w\cdot x^{(j)}) + (1-y^{(j)})\log(1-\sigma(wx^{(j)}))]
\end{aligned}
$$

Veremos ahora como encontrar los gradientes de los pesos, que me permitan actualizarlos de manera tal de minimizar este costo

Descenso del gradiente para PEC

- Nuevamente el problema es minimizar $J(w)$, es decir encontrar el $\hat{\theta}$ (los pesos w en este caso) que cumplan:

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{m} \sum_{j=1}^{m} L_{CE}(y^{(j)}, x^{(j)}; \theta)$$

- Como ya vimos, la actualización de θ basada en el gradiente de una función de pérdida entre un estimador $f(x; \theta)$ y la función real y es:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x; \theta), y)$$

- Por otra parte, para regresión logística vimos que esta función es $L_{CE}(w)$ y su derivada para un vector de observación x es:

$$\frac{\partial L_{CE}(w)}{\partial w_j} = [\sigma(w \cdot x) - y] x_j$$

Descenso del gradiente para PEC

- Mientras que para la función de costo $J(w)$ para un 'batch' (lote) de datos o el data set entero es:

$$\frac{\partial J(w)}{\partial w_j} = \sum_{i=1}^{m} [\sigma(w \cdot x^{(i)}) - y^{(i)}] x_j^{(i)}$$

- Ya vimos que el ajuste de los pesos w suele hacerse:
  - Por cada ejemplo de entrenamiento (SGD "extremo")
  - En pequeños lotes (minibatchs)
- La taza de aprendizaje η (usada en la actualización $w_j \leftarrow w_j - \eta \frac{\partial J(w)}{\partial w_j}$), suele comenzar con un valor más grande, que es decrementado en función del número de iteraciones.

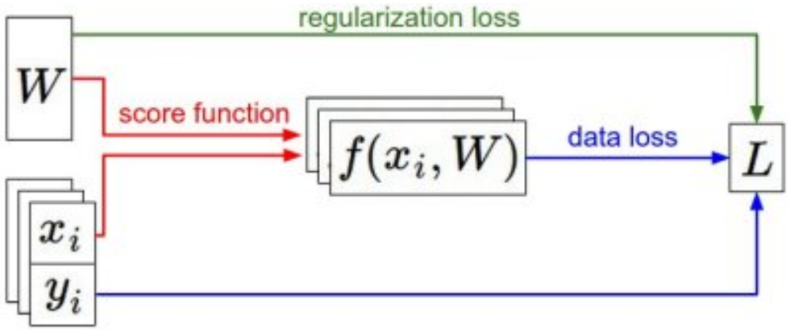
Regularización

- Como vimos en regresión lineal, una forma de evitar el sobreajuste es agregar en la función a optimizar un término de **regularización** $R(W)$.
- Este término, penaliza los pesos muy grandes
- De esta forma, la función de pérdida L tiene una componente que tiene ver con los datos (pérdida de los datos) y otra que tiene que ver con los pesos (pérdida de regularización):

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{pérdida de los datos}} + \underbrace{\lambda R(W)}_{\text{pérdida de regularización}}$$

Regularización

Gráficamente:



Formas comunes de regularización

- **Regularización** $L2$: Usa el cuadrado de la norma $L2$ ($||W||_2$), es decir

$$R(W) = ||W||_2^2 = \sum_{j=1}^{N} w_j^2$$

- **Regularización** $L1$: Usa la norma $L1$ ($||W||_1$), es decir

$$R(W) = ||W||_1 = \sum_{j=1}^{N} |w_j|$$

- El factor de regularización λ controla la intensidad de la regularización (más alto, pesos más bajos). En scikit-learn se controla indirectamente con el parámetro $C = 1/\lambda$ (más alto, menos incidencia regularización, pesos más altos)

Regresión Logística Multinomial

- Aplicable en problemas con más de dos clases
- También llamada regresión softmax o clasificador maxent
- La variable de salida y toma más de dos valores (clases) $\Rightarrow$ buscamos saber la probabilidad de que y sea de cada clase potencial $c \in C$, $p(y = c|x)$
- RLM usa una generalización de la sigmoide llamada función softmax

Función *Softmax*

- Toma un vector $z = [z_1, z_2, \ldots, z_k]$ de k valores arbitrarios y lo mapea en una distribución de probabilidad (valores entre 0 y 1 que suman a 1)
- Como la sigmoide, es una función exponencial. Con $|z| = k$, se define como:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

  donde $(1 \leq i \leq k)$
- El softmax de un vector de entrada $z = [z_1, z_2, \ldots, z_k]$, también es un vector:

$$softmax(z) = \left[\frac{e^{z_1}}{\sum_{i=1}^{k} e^{z_i}}, \frac{e^{z_2}}{\sum_{i=1}^{k} e^{z_i}}, \ldots, \frac{e^{z_k}}{\sum_{i=1}^{k} e^{z_i}} \right]$$

Función *Softmax*

- Ejemplo. Dado un vector

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

- El softmax resultado es

$$softmax(z) = [0.055, 0.090, 0.0067, 0.10, 0.74, 0.010]$$

- También aqui tendremos como entrada el producto punto entre un vector de pesos w y un vector de entrada x. Pero ahora, tendremos un vector de pesos w_c por cada una de las k clases

$$p(y = c|x) = \frac{e^{w_c \cdot x}}{\sum_{j=1}^{k} e^{w_j \cdot x}}$$

Regresión Logística en Scikit-learn

- Clase: `sklearn.linear_model.LogisticRegression`
- Principales Parámetros:
  1. **penalty**: `'l1'` o `'l2'`
  2. **C**: (1.0 por default)
  3. **solver**: `'newton-cg'`, `'lbfgs'`, `'liblinear'`, `'sag'` o `'saga'`
  4. **multi_class**: `'ovr'`, `'multinomial'` o `'auto'`

Otras funciones de activación

- Se pueden generalizar las ideas vistas previamente, y considerar que las unidades computan un valor

$$\hat{y} = \Phi(w \cdot x)$$

- Es decir, se computa primero el producto punto $z = w \cdot x$ (a este producto se le suele denotar *net*) y luego se aplica una función de activación Φ
- Hasta ahora, sólo hemos utilizado 3, pero existen varias otras que se detallan a continuación

Ejemplos de funciones de activación

- $\Phi(v) = v$ (función identidad)
- $\Phi(v) = sign(v)$ (función signo)
- $\Phi(v) = \frac{1}{1+e^{-v}}$ (función sigmoide)
- $\Phi(v) = \max\{v, 0\}$ (ReLU)
- $\Phi(v) = \frac{e^{2v}-1}{e^{2v}+1}$ (función tanh)
- $\Phi(v) = \max\{\min[v, 1], -1\}$ (tanh dura)