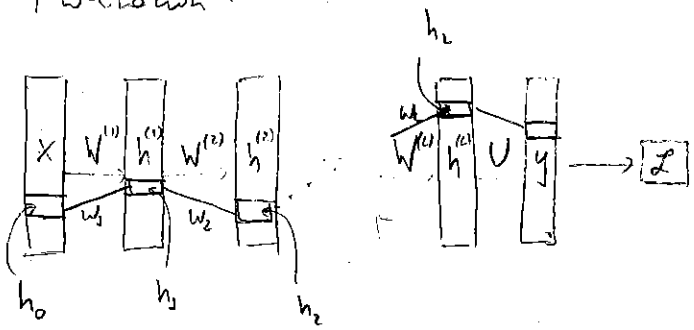
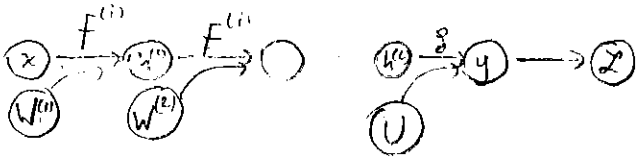
Optimización (Inicialización)

→ Idea: mejorar los procesos de entrenamiento, reducir
+ Inicialización, normalización
+ Algoritmos para descenso de gradiente más efectivos

Motivación:





sea h_i una unidad cualquiera de la capa oculta de $h^{(i)}$

$$h_i = f_i(h_{i-1} w_i + \underbrace{b_i'}_{\text{Luego que no depende de } w_i \text{ ni de } h_{i-1}}) \quad \text{para } i \in \{1, \dots, L\}$$

ni hago Back Propagation desde $\mathcal{L}$ hasta w_1 por la regla identificada antes

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial h_L} \cdot \frac{\partial h_L}{\partial h_{L-1}} \cdot \frac{\partial h_{L-1}}{\partial h_{L-2}} \cdots \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial h_L} \left(\prod_{j=2}^{L} \frac{\partial h_j}{\partial h_{j-1}} \right) \cdot \frac{\partial h_1}{\partial w_1}$$

¿ $\frac{\partial h_j}{\partial h_{j-1}}$? $\Rightarrow$ $\frac{\partial f_j(h_{j-1} w_j + b_j')}{\partial h_{j-1}} = f_j'(\underbrace{h_{j-1} w_j + b_j'}_{\text{Llamemos a esto } u_j}) \cdot w_j$

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial h_L} \left(\prod_{j=2}^{L} f_j'(u_j) \right) \left(\prod_{j=2}^{L} w_j \right) \frac{\partial h_1}{\partial w_1}$$

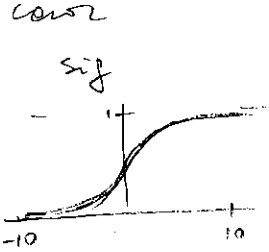
$$h_1 = f_1(x w_1 + b_1')$$

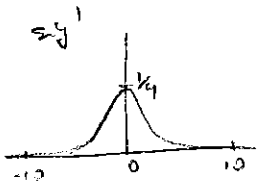
$$\frac{\partial h_1}{\partial w_1} = f_1'(u_1)\, x$$

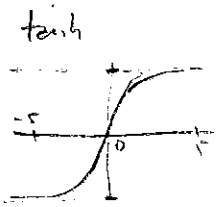
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial h_L} \left(\prod_{j=1}^{L} f_j'(u_j) \right) \left(\prod_{j=2}^{L} w_j \right) x$$

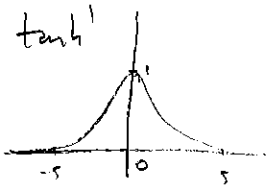
- basta que uno de estos valores sea 0 para que el gradiente sea 0
- sea 0 $\Rightarrow \frac{\partial \mathcal{L}}{\partial w_1} = 0 \Rightarrow$ no hay aprendizaje para w_1
- si varios de ellos son valores pequeños, $\Rightarrow \frac{\partial \mathcal{L}}{\partial w_1}$ será muy (muy) pequeño si L es muy grande

cosas









$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial \mathcal{L}}{\partial h_L} \cdot \prod_{j=1}^{L} f'(u_j) \prod_{j=2}^{L} w_j \; x \qquad sig'(u_j) \leq \frac{1}{4}$$

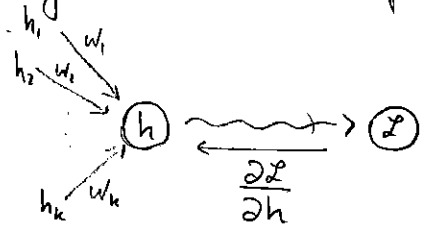
① si los w_j son todos muy pequeños $\Rightarrow \prod_{j=2}^{L} w_j$ será muy (muy) pequeño

② sup. que usamos sigmoid
$\Rightarrow$ si los u_j's son muy grandes el aprendizaje si siquiera comenzará
∴ mucho cuidado al inicializar !! $\Rightarrow$ hay que tratar de mantenerse en el único activo del sigmoid (cerca del 0)

③ sup. sigmoid todavía sin importar el valor de $u_j \Rightarrow sig'(u_j) \leq \frac{1}{4}$

$\Rightarrow \prod_{j=1}^{L} sig'(u_j) \leq \left(\frac{1}{4}\right)^L \Rightarrow$ si L es muy grande el gradiente se irá a 0 (vanishing gradient problem). Pero lo mismo para todos los caminos que llevan a W_1

④ sigmoid tiene otro problema más:



$$h = sig(h_1 w_1 + h_2 w_2 + \cdots + h_k w_k + b)$$

$$\frac{\partial \mathcal{L}}{\partial w_i} = h_i \cdot sig'(u) \cdot \frac{\partial \mathcal{L}}{\partial h}$$

si todos los h_i's son positivos $\Rightarrow$ el signo de $\frac{\partial \mathcal{L}}{\partial w_i}$ depende solo de $sig'(u) \cdot \frac{\partial \mathcal{L}}{\partial h}$

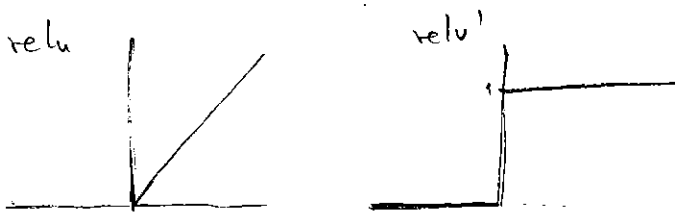
$\Rightarrow$ todos los $\frac{\partial \mathcal{L}}{\partial w_i}$ tendrán siempre el mismo signo.

Note que $sig(u)$ es siempre positivo $\Rightarrow$ en cada iteración de una red con activación sig todos los pesos asociados a una unidad tendrán gradiente con el mismo signo. (En general nos gustaría que para cada unidad, los inputs estén centrados en el 0 !!!)

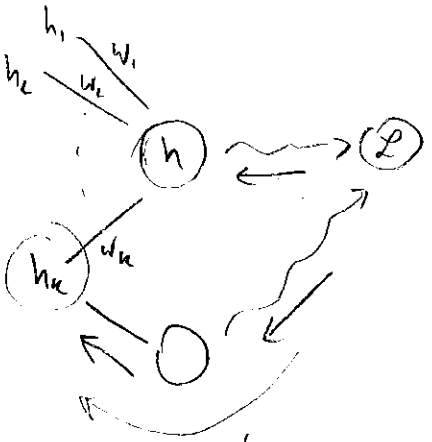
★ Morejo 1 : NUNCA USAR sigmoid EN DEEP LEARNING

Para tanh $\Rightarrow$ Tenemos los problemas ② y ③ pero no tenemos el ④

★ Morejo 2 : entre tanh y sigmoid, siempre preferir tanh



con relu no hay desvanecimiento de gradiente $\frac{\partial \mathcal{L}}{\partial w_i}$ no tiende a 0 en mucho mas "discreto"



gradiente que puede cambiar el valor de activación de la relu

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial h} \cdot \underbrace{relu'(u)}_{0 \text{ o } 1} \cdot h_i$$

- si es 0 $\Rightarrow w_i$ no se actualizará pero no significa que nunca más se actualizará porque los h_i's pueden seguir recibiendo gradiente por otros caminos

Pero: relu tiene el problema (4) de sigmoid (los inputs de cada unidad no están centrados en 0).

En la práctica relu funciona mejor que tanh !!! (de hecho es una de las razones del despegue de redes neuronales profundas)

★ Mensaje 3: SIEMPRE USAR relu EN DEEP LEARNING

---

hay alternativas a relu que centran su output en 0:

- Leaky RELU
- ELU, CELU etc.

Observación: los pesos iniciales pueden importar sustancialmente en el entrenamiento

+ Inicialización:

- las activaciones siempre se verán como $f(h_1 w_1 + h_2 w_2 + \cdots + h_k w_k)$

queremos que $z = h_1 w_1 + h_2 w_2 + \cdots + h_k w_k$ no sea demasiado variable a pesar de que los elijamos al azar.

intento: $w_i = \underbrace{\text{randnormal}}_{\text{distribución normal}} \cdot \sqrt{\frac{1}{k}}$ $\left\{ \begin{array}{l} \text{origina que la media sea} \\ 0 \text{ y la varianza } \frac{1}{k} \\ \Rightarrow \text{varianza de la suma es } 1 \end{array} \right.$

k = dimensión de entrada a la capa

Inicialización de Xavier

En notación matricial más

$W^{(i)} = \text{randn}(d_{i-1}, d_i) * \sqrt{\frac{1}{d_{i-1}}}$

- Xavier funciona bien en general, pero empíricamente se ha visto que para relu una mejor inicialización es

$$w_i = \text{randnormal} \cdot \sqrt{\frac{2}{k}}$$

[En notación $W^{(i)} = \text{randn}(d_{i-1}, d_i) * \sqrt{\frac{2}{d_{i-1}}}$

- Otra propuesta incluye dividir por la suma de las dimensiones

$$W^{(i)} = \text{randn}(d_{i-1}, d_i) * \sqrt{\frac{1}{d_{i-1}+d_i}}$$

- Todas estas son empíricas, también se podría "buscar" la varianza correcta como un hiperparámetro

---

En general queremos que las outputs de muchas capas sean 0-centered (también nos gustaría que tuvieran la misma varianza en todas las direcciones). La primera capa es el input y ahí podemos asegurarlo !!!

$D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(N)}, y^{(N)})\}$ y nos X tenemos con los $x^{(i)}$'s $X(N, C)$

$\mu := \frac{1}{N} \sum_{i=1}^{N} x^{(i)}$ $\left(= \frac{1}{N} \text{sum}_1(X) \right)$ ← media para cada feature

$\mu(C)$

$X := X - \mu$ ← resto la media componente a componente a todos los ejemplos

↑ broadcast ⟹ los datos quedan "centrados"

$V := \frac{1}{N} \sum_{i=1}^{N} (x^{(i)}) * (x^{(i)})$ $\left(= \frac{1}{N} \text{sum}_1(X * X) \right)$

↑ broadcast

$X := \frac{X}{\sqrt{V}}$ ← broadcast $\left(= X * 1/\text{sqrt}(V) \right)$

$a_1, a_2, \dots, a_N$

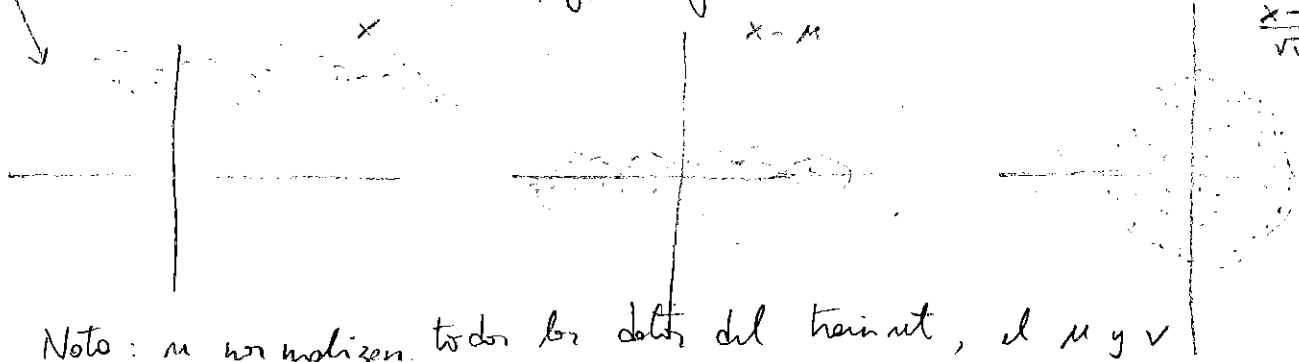
$\mu = \frac{1}{N} \sum a_i$

$V = \frac{1}{N} \sum (a_i - \mu)^2$

$a_i' = \frac{a_i - \mu}{\sqrt{V}}$ ← normaliza la media a 0, varianza 1

★ Recordar: siempre normalizar el input (no duele)

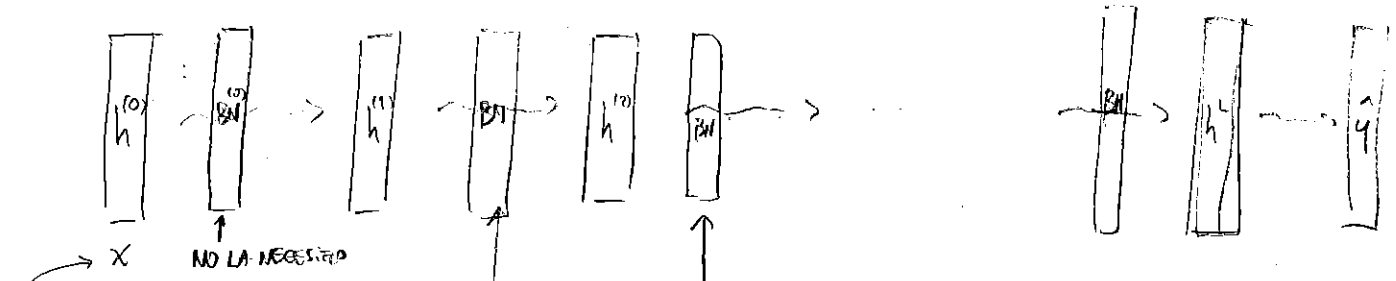
- en el caso de imágenes generalmente sólo se centra (valores entre 0-256)



Nota: se normalizan todos los datos del training set, el μ y V computados ahí se usan para normalizar en test time

Normalización por Paquetes (Batch Normalization)

Idea: normalizar el input hace buenas propiedades para el entrenamiento. Podemos hacer algo similar dentro de la red?



normalizamos tb el train set y luego se la pasamos a la red

Trataré de normalizar entre medio de las capas !!! ¿Cómo?

Nos aprovechamos que los ejemplos vienen en paquetes

$$h^{(i+1)} := f^{(i+1)}\left(\underbrace{h^{(i)}}_{\text{input de la capa } i+1} W^{(i+1)} + b^{(i+1)}\right)$$

$\Rightarrow$ normalizamos $h^{(i)}$ a lo largo del batch

$h^{(i)}(B, d_i)$ ← dimensión

$$\mu^{(i)} := \frac{1}{B}\sum_{j=1}^{B} h^{(i)}_{j,:} = \frac{1}{B} * \text{sum}_1\left(h^{(i)}\right)$$

$\mu^{(i)}(d_i)$ ← dimensión

$v^{(i)}(d_i)$ ←

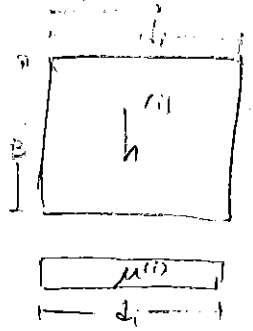
$$v^{(i)} := \frac{1}{B}\sum_{j=1}^{B}\left(\left(h^{(i)} - \mu^{(i)}\right)^2\right)_{j,:}$$

$$\tilde{h}^{(i)} := \frac{h^{(i)} - \mu^{(i)}}{\sqrt{v^{(i)} + \varepsilon}}$$

para evitar división por 0

← normalicé los elementos del batch

broadcast !! (eje normalización por cada feature)



Opción 1) $h^{(i+1)} = f^{(i+1)}\left(\tilde{h}^{(i)} W^{(i+1)} + b^{(i+1)}\right)$

Es derivable! puedo seguir usando Back Prop

$$\frac{\partial h^{(i+1)}}{\partial h^{(i)}} = \frac{\partial h^{(i+1)}}{\partial \tilde{h}^{(i)}} \cdot \frac{\partial \tilde{h}^{(i)}}{\partial h^{(i)}}$$

Posible problema: estamos bajando la capacidad de la red !!

$\tilde{h}^{(i)}$ como transformación es lineal

$\Rightarrow$ mi red lo podría haber aprendido

Solución: devolverle la capacidad a la red $\Rightarrow$ permitirle que deshaga la normalización si así lo quiere

$$\hat{h}^{(i)} := \gamma^{(i)} * \tilde{h}^{(i)} + \beta^{(i)}$$

$\tilde{h}^{(i)}$ (B, d_i)
$\gamma^{(i)}$ (d_i) ← dimensiones
$\beta^{(i)}$ (d_i)

↓ nuevos parámetros de la red !!!

Se actualizan igual que todos los otros calculando primero el gradiente $\frac{\partial \mathcal{L}}{\partial \gamma^{(i)}}$ $\frac{\partial \mathcal{L}}{\partial \beta^{(i)}}$ $\left(= \frac{\partial \mathcal{L}}{\partial \hat{h}^{(i)}} \cdot \frac{\partial \hat{h}^{(i)}}{\partial \beta^{(i)}}\right)$

Nueva fórmula

$$h^{(i)} = f^{(i)}\left(\hat{h}^{(i-1)} W^{(i)} + b^{(i)}\right)$$
$$\hat{h}^{(i)} = \gamma^{(i)} \, \text{norm}\left(h^{(i)}\right) + \beta^{(i)}$$

$$\text{norm}\left(h^{(i)}\right) := \frac{h^{(i)} - \mu^{(i)}}{\sqrt{v^{(i)} + \varepsilon}}$$

↑ estabilidad numérica

Observaciones:

- Paper original (2015!) propone normalización antes de la no linealidad

$$u^{(i)} = h^{(i-1)} W^{(i)} + b^{(i)}$$
$$\hat{u}^{(i)} = \gamma^{(i)} \, \text{norm}\left(u^{(i)}\right) + \beta^{(i)}$$
$$h^{(i)} = f^{(i)}\left(\hat{u}^{(i)}\right)$$

en este caso $b^{(i)}$ y $\beta^{(i)}$ funcionan como términos de bias redundantes puedo obviarme de los $b^{(i)}$'s (solo usar $\beta^{(i)}$'s)

- En test time puede no tener sentido usar $\mu^{(i)}$ y $v^{(i)}$ calculados desde un batch $\Rightarrow$ debo fijar esos valores al finalizar el entrenamiento. Una forma es ir manteniendo un promedio de $\mu^{(i)}$, $v^{(i)}$ durante el entrenamiento, digamos $\hat{\mu}^{(i)}$, $\hat{v}^{(i)}$, y usar esos valores en test time.

- La opción más usada: **"promedio movil exponencial"** ← idea

$$\bar{\mu}^{(i)t} = \delta \bar{\mu}^{(i)t-1} + (1-\delta)\mu^{(i)t}$$
$$\bar{v}^{(i)t} = \delta \bar{v}^{(i)t-1} + (1-\delta)v^{(i)t}$$

$0 \leq \delta \leq 1$ $\quad \delta \to 1 \Rightarrow$ recuerdo desde el principio
$0 \leq \delta < \frac{1}{2} \Rightarrow$ me quedo con el más reciente

valor típico: $\delta = 0.9$ $\approx$ promedio ponderado de $\left(\delta = 0.5 \Rightarrow \text{últ. } 2,\ \delta = 0.98 \Rightarrow \text{últ. } 50\right)$ aprox. los últimos 10

Implementación de hiperparámetro

- En cada iteración calculo $\mu^{(i)}$ actualizo $\bar{\mu}^{(i)}$ como

$$\bar{\mu}^{(i)} := \delta \bar{\mu}^{(i)} + (1-\delta)\mu^{(i)}$$

igual para $\bar{v}^{(i)}$