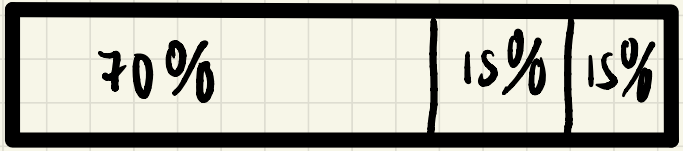
2. Entrenamiento Regularización Optimización

Marzo 2024

① ENTRENAMIENTO : DATOS

- LAS MUESTRAS DE DESARROLLO (VALIDACIÓN) y PRUEBA SON CADA VEZ MENORES:

TRADICIONALMENTE

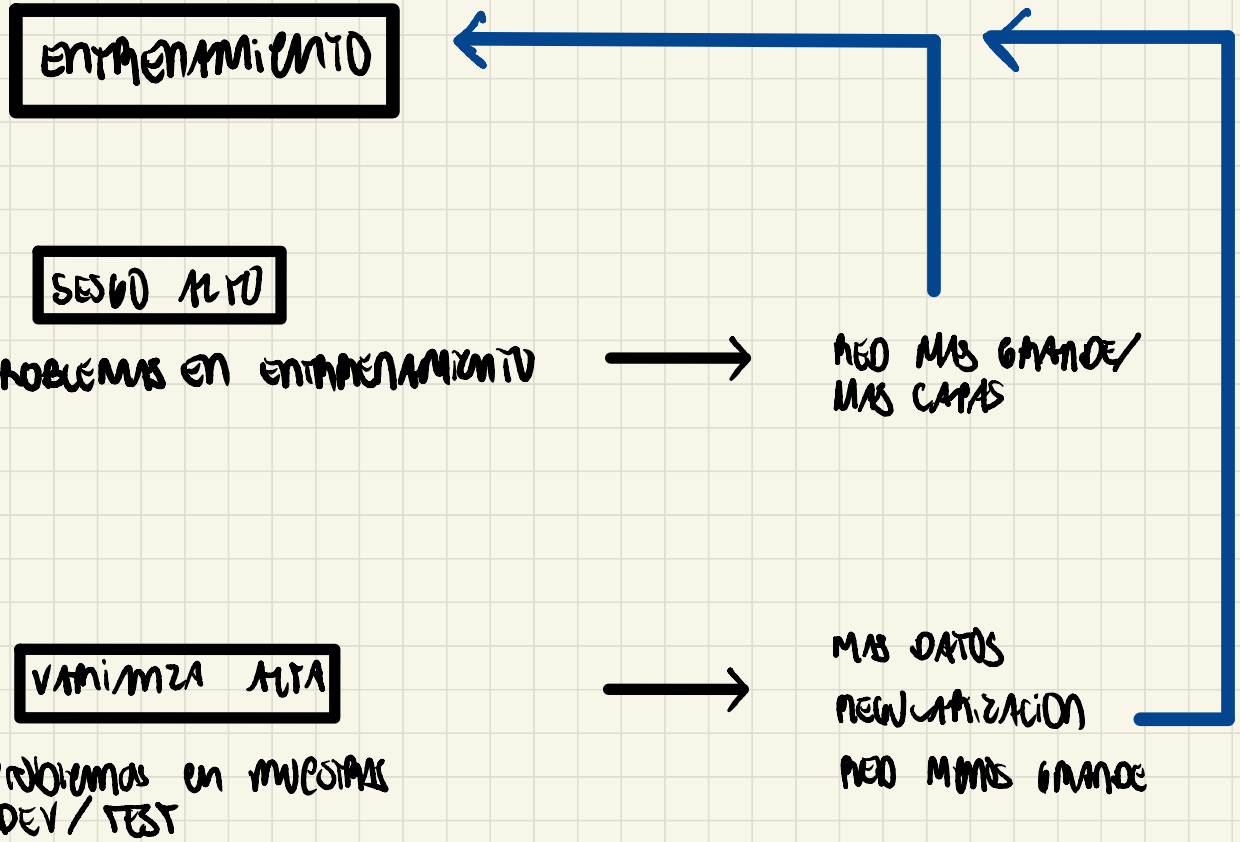


- COMO VERÍAMOS MÁS ADELANTE LO MÁS IMPORTANTE ES QUE LA MUESTRA DE DESARROLLO Y PRUEBA VENGAN DE LA MISMA DISTRIBUCIÓN.

DL

SE NECESITAN MUCHOS DATOS PARA ENTRENAR Y ERROR FUERA DE MUESTRA SE ESTIMA CON 2% DE MUCHOS DATOS

② ENTRENAMIENTO: ANALISIS SESGO (APROXIMACION) Vs VARIANZA (ESTIMACION)



EJEMPLO : SESGO VS. VARIANZA

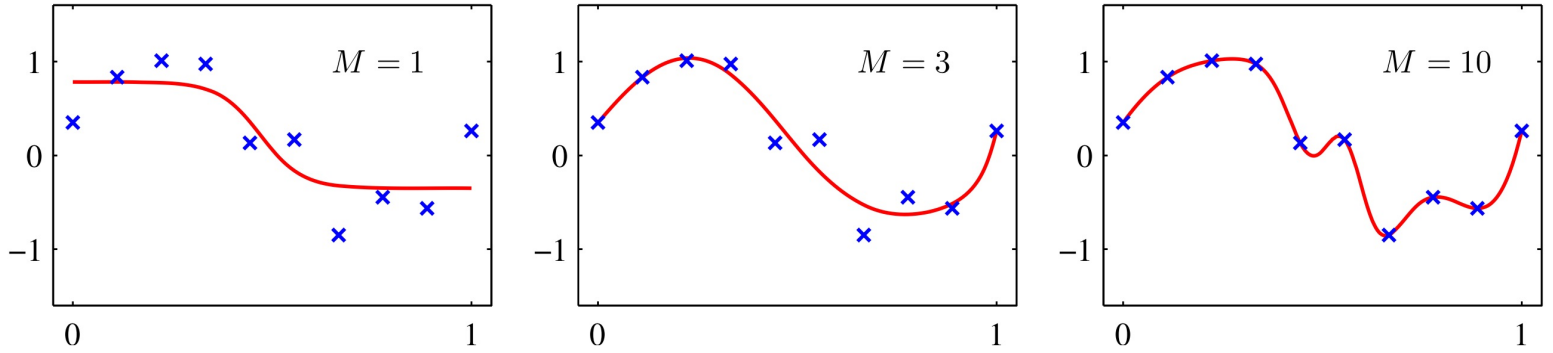
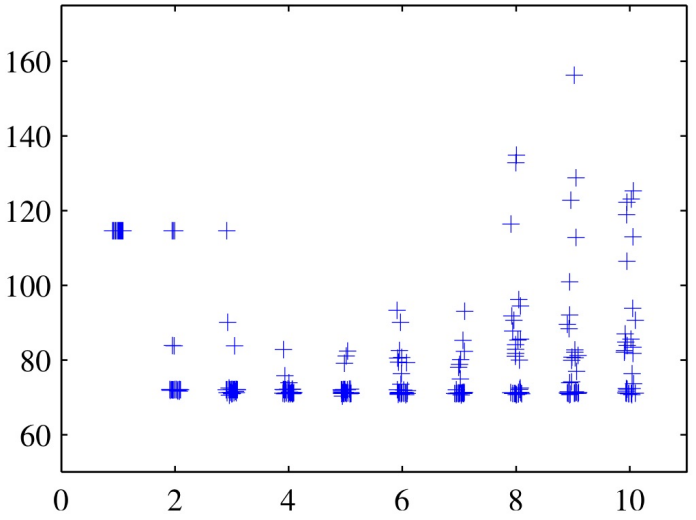


Figure 5.9 Examples of two-layer networks trained on 10 data points drawn from the sinusoidal data set. The graphs show the result of fitting networks having $M = 1$, 3 and 10 hidden units, respectively, by minimizing a sum-of-squares error function using a scaled conjugate-gradient algorithm.

- La relación entre el número de unidades ocultas M y el error fuera de muestra es difícil de descubrir por los óptimos locales

Figure 5.10 Plot of the sum-of-squares test-set error for the polynomial data set versus the number of hidden units in the network, with 30 random starts for each network size, showing the effect of local minima. For each new start, the weight vector was initialized by sampling from an isotropic Gaussian distribution having a mean of zero and a variance of 10.



③ ENTRENAMIENTO: DATA AUGMENTATION

- rotations / translations / reflections
- zooming
- adding noise

① **REGULARIZACIÓN**: WEIGHT DECAY

$$J(W^{[1]}, b^{[1]}, \dots W^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(y^{(i)}, \hat{y}^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} \|W^{[l]}\|_F^2$$

donde:

$$\|W^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} (w_{ij}^{[l]})^2$$

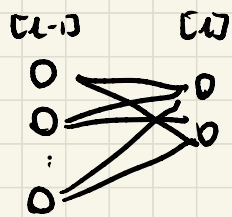
↓

La misma que la norma L_2

↓

$n_{[l]} \times n_{[l-1]}$

$$\begin{bmatrix} w_{11} & \dots & w_{1n^{[l-1]}} \\ \vdots & & \\ w_{n^{[l]}1} & \dots & \end{bmatrix} \begin{Bmatrix} a_1^{[l-1]} \\ \vdots \\ a_{n^{[l-1]}}^{[l-1]} \end{Bmatrix}$$



- cuando hacemos backpropagation:

$$dW^{[l]} \leftarrow \underbrace{\text{backprop anterior}}_{} + \frac{\lambda}{m} W^{[l]}$$

- luego el algoritmo de gradiente descendiente:

$$W^{[l]} \leftarrow W^{[l]} - \alpha\, dW^{[l]}_{orig.} - \alpha \frac{\lambda}{m} W^{[l]}$$

$dW^{[l]}_{orig.}$: gradiente original sin regularización

$$W^{[l]} \leftarrow \underbrace{\left(1 - \frac{\alpha\lambda}{m}\right) W^{[l]}}_{\text{weight decay}} - \alpha\, dW^{[l]}_{orig.}$$

- Regularización funciona por la razones que ya vimos en el curso (i.e. selección de variable suave)

- Adicionalmente si: $\lambda \uparrow$, $\|W\|_F \downarrow 0$
  $\Rightarrow \|z\| \downarrow 0$ y $g(z)$ es casi una función lineal (e.g. $g \approx \tanh$ o hew) que es una función muy simple.

REGULARIZACIÓN: DESDE UN PUNTO DE VISTA BAYESIANO

- Consideramos un modelo lineal de funciones base:

$$y(x, w) = w^T \phi(x)$$

donde:

$$\phi(x) = (1, \phi_1(x), \ldots, \phi_{n-1}(x))^T$$

- Las funciones ϕ_i se llaman funciones base.
- Supongamos ahora que: $\hat{y} = y(x, w) + \varepsilon$ donde $\varepsilon \sim N(0, \beta^{-1})$
- Luego la distribución de probabilidad:

$$p(\hat{y} \mid x, w, \beta) = N(\hat{y} \mid y(x, w), \beta^{-1})$$

- Suponiendo independencia condicional de las observaciones la log verosimilitud es:

$$\frac{M}{2}\ln(\beta) - \frac{M}{2}\ln(2\pi) - \beta \sum_{i=1}^{m} (y_i - \underbrace{w^T \phi(x_i)}_{\hat{y}_i})^2$$

y replicamos el resultado que <u>maximizar la verosimilitud</u> es lo mismo que <u>minimizar el error cuadrático</u>

- Introducción de prior: ↳ β conocido

- verosimilitud: $p(\hat{y} \mid X, w, \beta) = \mathcal{N}(\hat{y} \mid y(X, w), \beta^{-1})$

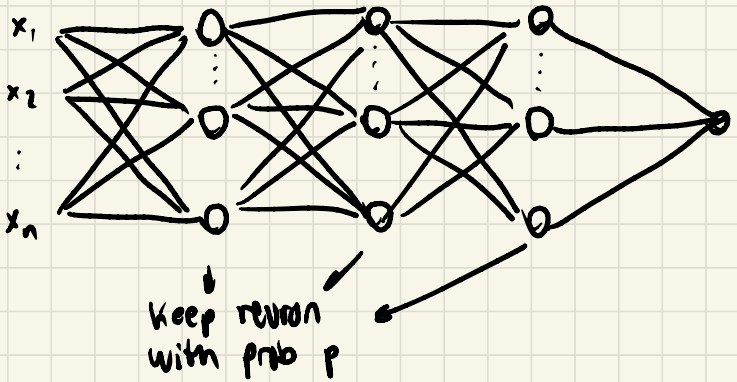
- prior conjugada: $p(w \mid \alpha) = \mathcal{N}(w \mid 0, \alpha^{-1} I)$

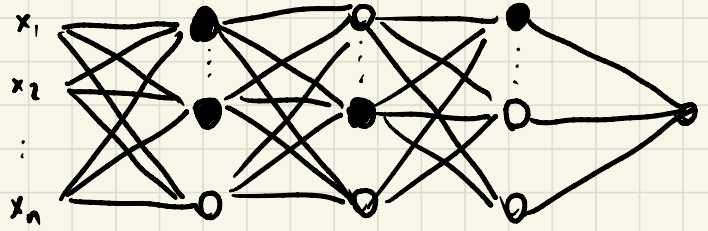
- La posterior también es normal: ↳ por simplicidad centrado en cero

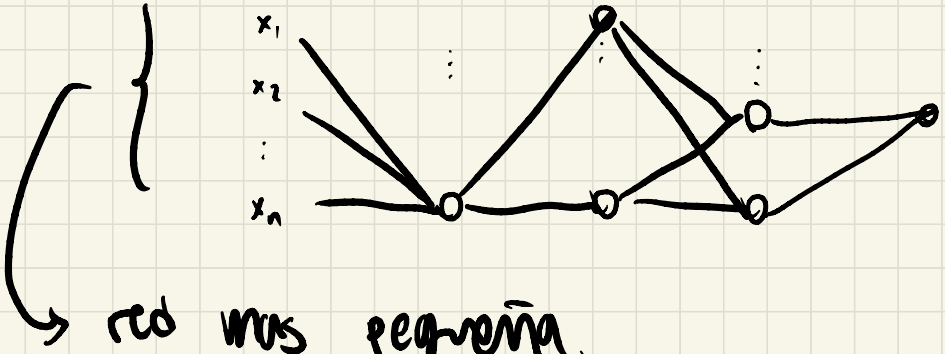
(i.e. posterior de todos los datos observados)

$$\log p(w \mid y) = -\frac{\beta}{2} \sum_{i=1}^{n} (y_i - w^T \phi(x_i))^2 - \frac{\alpha}{2} w^T w + \text{const.}$$

② REGULARIZATION: DROPOUT







- en el momento de implementar es importante reescalar las funciones de activacion porque:

$$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$$

↓

si estas se reducirán en p entonces $z^{[l]}$ tambien se reduce por un factor de p. LA IDEA DE <u>inverted dropout</u> es reescalar asi:

$$z^{[l]} = \underbrace{w^{[l]} a^{[l-1]}}_{p} + b^{[l]}$$

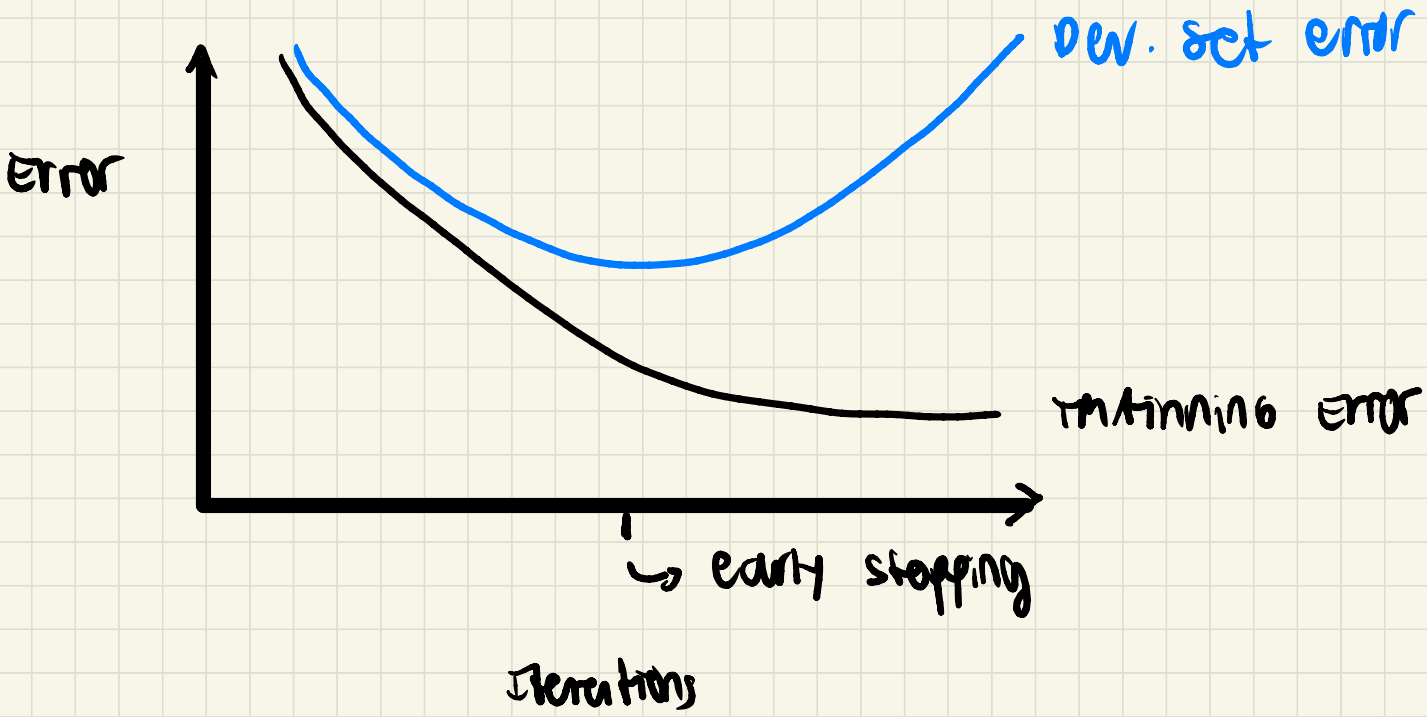
esto resulta de mantener el valor esperado de $z^{[l]}$ constante

Observaciones

- Dropout distribuye la importancia DE LAS NEURONAS impidiendo que un peso tome demasiada importancia
- en principio se puede usar una probabilidad dropout en cada capa (e.g. si hay muchos pesos en una capa)
- se puede hacer incluso en la capa de entrada (no muy común)
- cuando se hacen predicciones no se aleatoriza.

- Un problema es que la función de costo no es decreciente durante el entrenamiento. La función no está bien definida. Luego se puede verificar el código primero sin dropout y después se regulariza.

③ EARLY STOPPING



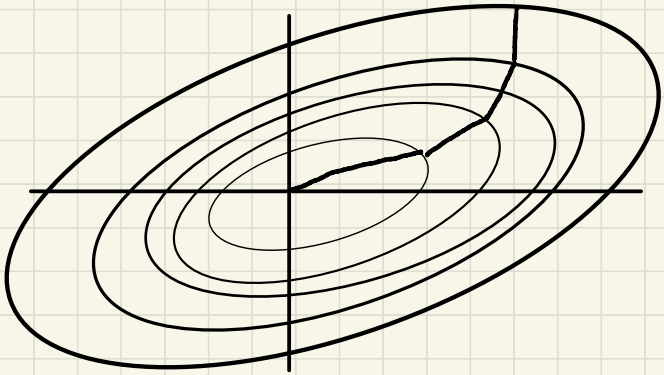
- un problema con esta estrategia es que mezcla dos problemas: | optimizacion de $J(w,b)$
| Avoid overfitting.

cuando se hace early stopping se deja de optimizar buscando reducir overfitting. Separar los dos problemas se conoce como orthogonalizacion.

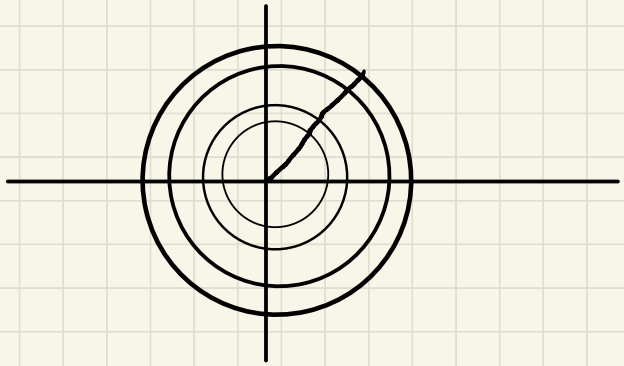
- Una mejor alternativa pero más costosa computacionalmente es hacer validacion cruzada o usar el Dev set para elegir hiperparametros. Por ejemplo de regularizacion L^2.

① **OPTIMIZACIÓN** : NORMALIZACIÓN, IMP/EXPLOSIÓN, GRAD CHECKING

- si se normalizan los datos gradiente descendiente funciona mejor
- $F: \mathbb{R}^3 \to \mathbb{R}$



SIN NORMALIZAR



NORMALIZADO

- IMPLOSIÓN / EXPLOSIÓN DE GRADIENTES

CONSIDERE ESTA RED:

$g(z) = z$, $b = 0$ en todas las capas

$\Rightarrow \hat{y} = W^{(L)} \cdots W^{(1)} x$

dependiendo de $W_{ij}^{(l)} \gtrless 1$ por muy poco, puede crecer o decrecer exponencialmente.

- una forma de corregir esto es inicializando mejor los pesos.

como $z = w_1 x_1 + w_2 x_2 \ldots + w_n x_n + b$
entre más elementos en la suma potencialmente más grande z.
entonces se puede normalizar w de tal forma que la suma se "controle": $Var(w) = \frac{1}{n}$.

en la práctica lo mejor es $var(w) = \sqrt{\frac{2}{n}}$
LUEGO SE IMPLEMENTA COMO:

$$w^{[l]} = \text{random} * \sqrt{\frac{2}{n^{[l-1]}}}$$

Esta es la forma más común de hacerlo cuando las funciones de activación son ReLU. Para otras funciones se usan otras normalizaciones.

Gradient checking

Tomar $W^{[1]}, \dots W^{[L]}, b^{[1]}, \dots b^{[L]}$ y poner como un solo parámetro θ.

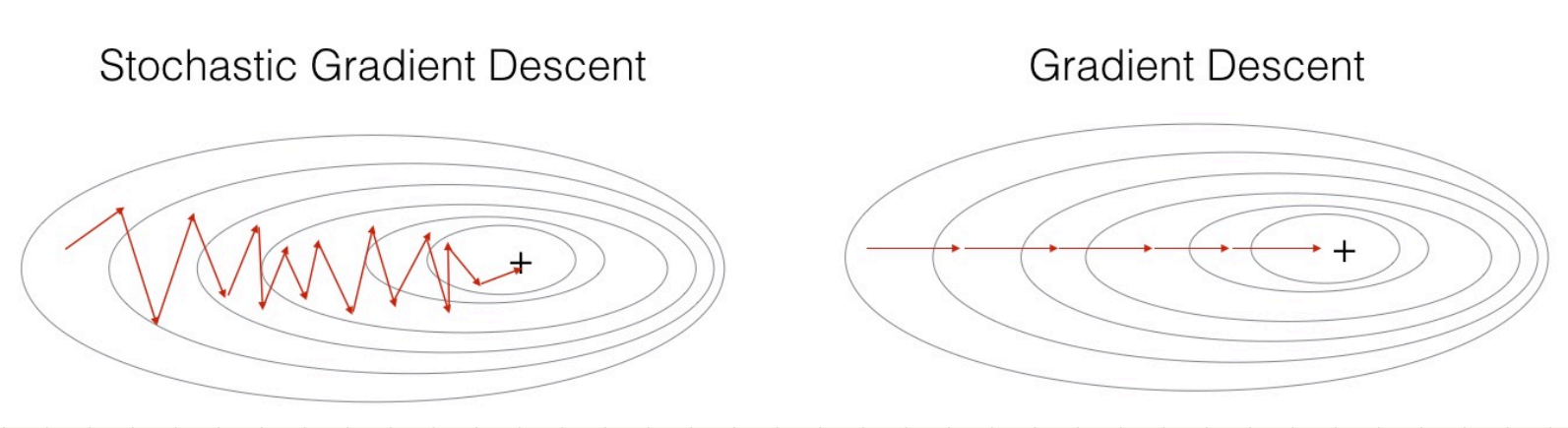
Escribir el costo como función de θ: $J(\theta_1, \dots \theta_M)$

Verificar:

$$d\theta[i] \overset{?}{=} d\theta_{approx} = \text{derivada numérica usando la secante.}$$

Por ejemplo: $$\frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2} < \varepsilon$$

② GRADIENT DESCENT : STOCHASTIC vs. BATCH

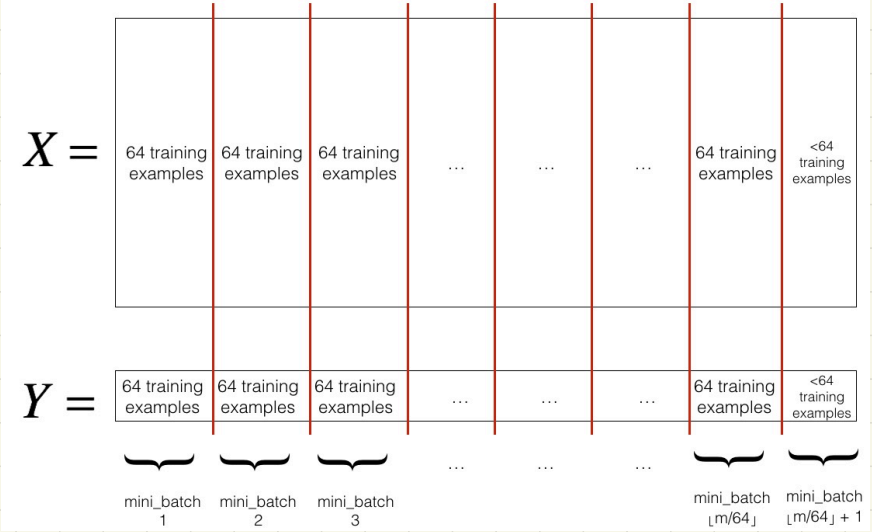


ACTUALIZACION RÁPIDA DE PARÁMETROS

ACTUALIZACION LENTA DE PARÁMETROS

NO APROVECHA VECTORIZACIÓN

③ BATCH vs MINI BATCH GRADIENT DESCENT



$$X = [x^{(1)} \; \ldots \; x^{(m)}]$$

$$x^{\{1\}} \quad \ldots \quad x^{\{t\}}$$

$$Y = [y^{(1)} \; \ldots \; y^{(m)}]$$

$$y^{\{1\}} \quad \ldots \quad y^{\{m\}}$$

MÉTODO DE PARTICIONES

$$X = \begin{pmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m-1)} & x_0^{(m)} \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m-1)} & x_1^{(m)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{12286}^{(1)} & x_{12286}^{(2)} & \dots & x_{12286}^{(m-1)} & x_{12286}^{(m)} \\ x_{12287}^{(1)} & x_{12287}^{(2)} & \dots & x_{12287}^{(m-1)} & x_{12287}^{(m)} \end{pmatrix}$$

$$Y = \begin{pmatrix} y^{(1)} & y^{(2)} & \dots & y^{(m-1)} & y^{(m)} \end{pmatrix}$$

$$X = \begin{pmatrix} x_0^{(5)} & x_0^{(16)} & \dots & x_0^{(2)} & x_0^{(m-1)} \\ x_1^{(5)} & x_1^{(16)} & \dots & x_1^{(2)} & x_1^{(m-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{12286}^{(5)} & x_{12286}^{(16)} & \dots & x_{12286}^{(2)} & x_{12286}^{(m-1)} \\ x_{12287}^{(5)} & x_{12287}^{(16)} & \dots & x_{12287}^{(2)} & x_{12287}^{(m-1)} \end{pmatrix}$$

$$Y = \begin{pmatrix} y^{(5)} & y^{(16)} & \dots & y^{(2)} & y^{(m-1)} \end{pmatrix}$$

METODO DE ALEATORIZACION

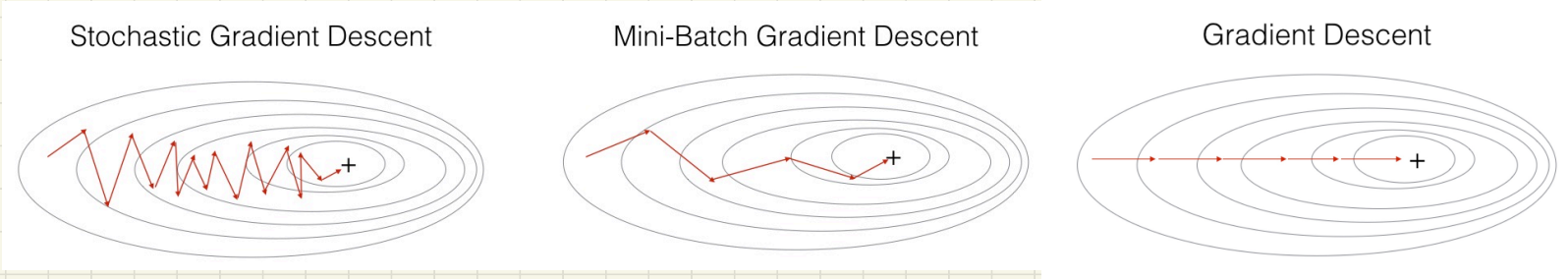
- en cada miniBATCH se estima todo: Forward, Backward, costo, etc y se calcula nuevos W, b.

repitiendo este proceso con todos los minibatches se actualiza respectivamente los W, b.

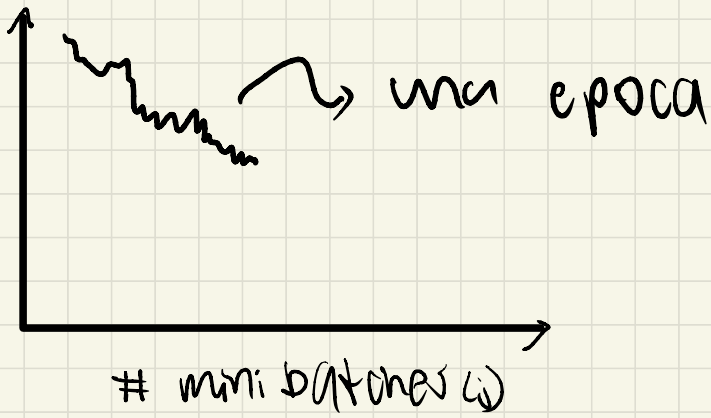
- Luego en contraste a batch gradient descent una sola pasada por todos los datos permite actualizar una sola vez los parámetros
- Con Mini batch gradient descent cuando se hace una pasada completa de los datos (llamado época) se actualizan varias veces los parámetros.
- Al hacer esto con varias épocas se llega mucho más rápido a los parámetros con mínimo en costo.

MINIBATCHES TAMAÑO 1

BATCH



- CUANDO SE USA MINIBATCHES EL COSTO NO ES MONOTONO:



Toma mucho tiempo por iteración

- Si los minibatches tienen tamaño m : BATCH GRADIENT DESCENT
- " " " " " 1 : stochastic gradient descent

↓

MUY inestable convergencia no aprovecha eficientemente LA VECTORIZACION

- Un punto intermedio es lo óptimo

Mejores prácticas: 64, 128, 256, 512

④ exponential weighted gradient descent

- observation $\{\theta_t\}$
- exponentially weighted average $v_t = \beta v_{t-1} + (1-\beta)\theta_t$
- corrección de sesgo: al comienzo v_t puede estar sistemáticamente por debajo de θ_t ($\theta_t \geq 0$) y para corregir esto en primeras iteraciones:

$$v_t \rightarrow \frac{v_t}{1-\beta^t}$$

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

$$v_1 = \beta v_0 + (1-\beta)\theta_1$$

$$v_2 = \beta^2 v_0 + \beta(1-\beta)\theta_1 + (1-\beta)\theta_2$$

$$v_3 = \cancel{\beta^3 v_0}^{\,0} + \beta^2(1-\beta)\theta_1 + \beta(1-\beta)\theta_2 + (1-\beta)\theta_3 \quad , \; v_0 = 0$$

$$\vdots$$

$$v_t = \left[\sum_{i=1}^{t} \beta^{t-i}\,\theta_i\right](1-\beta)$$

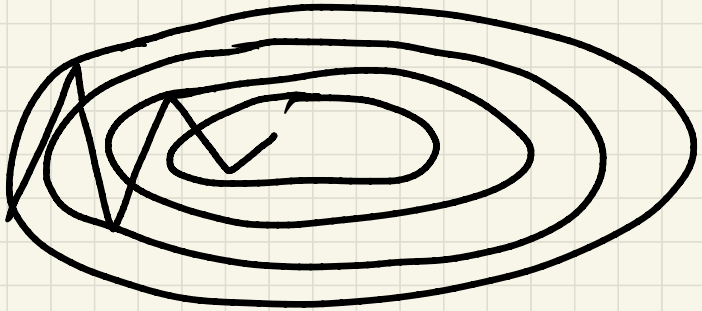
$$v_t = \theta\left[\sum_{i=1}^{t} \beta^{t-i}\right](1-\beta) \quad , \; \delta_i \;\; \theta_i = \theta$$

$$= \theta\left[\frac{1-\beta^t}{1-\beta}\right](1-\beta)$$

$$= \theta(1-\beta^t)$$

NOTA : TODO LOS SIGUIENTES ALGORITMOS SE IMPLEMENTAN EN CADA MINI BATCH.

⑤ Gradient descent with momentum.



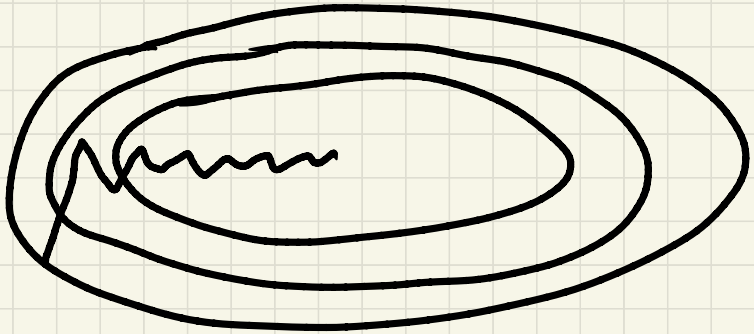
- Los movimientos en en eje y no son eficientes
- En el eje x si son mas eficientes.

- El método de momentum consiste en suavizar los gradientes en cada iteración utilizando el método anterior de "pesos exponenciales"

La idea es disminuir esa variaciones en y y aumentarlas en x.

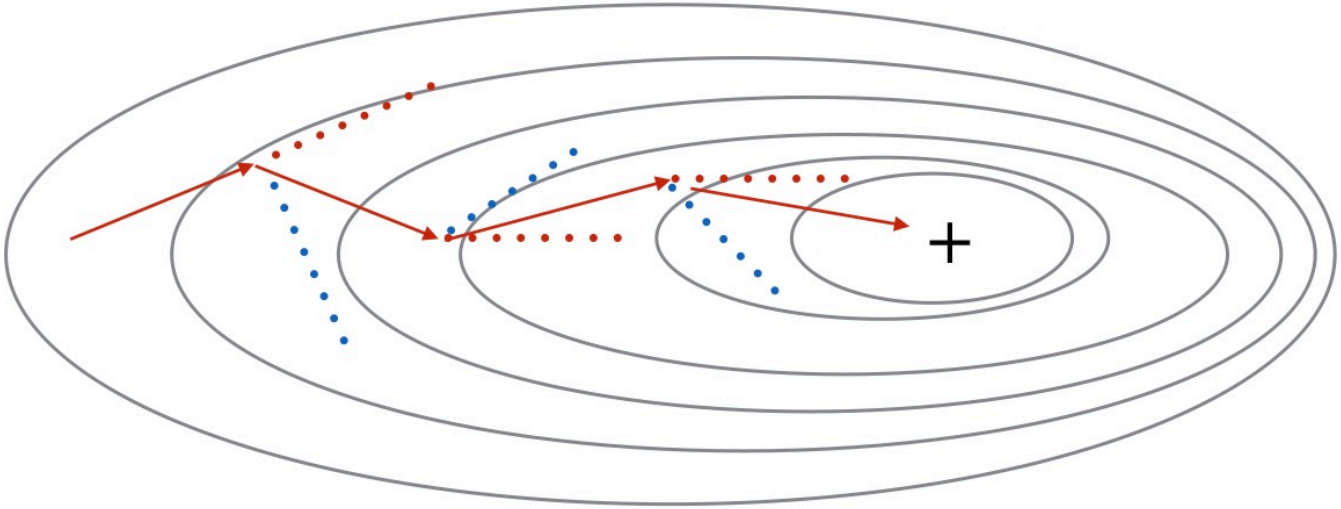
$$\left.\begin{aligned} V_{dw} &= \beta V_{dw} + (1-\beta) dW \\ V_{db} &= \beta V_{db} + (1-\beta) db \end{aligned}\right\}$$ se puede incluir el ajuste por sesgo.

¿por que funciona? Disminuye el recorrido en el eje Y.



$$W = W - \alpha V_{dw}$$
$$b = b - \alpha V_{db}$$

Momentum



⑤ RMS PROP

- PARA DISMINUIR LAS OSCILACIONES en Y y AUMENTARLAS en X:

- $S_{dw} = \beta S_{dw} + (1-\beta) dW * dW$

  $S_{db} = \beta S_{db} + (1-\beta) db * db$

- $w = w - \alpha \frac{dw}{\sqrt{S_{dw}}}$

- $b = b - \alpha \frac{db}{\sqrt{S_{db}}}$

} PARA EVITAR DIVIDIR POR CERO USUALMENTE SE MODIFICA $S_{dw} \leftarrow S_{dw} + \epsilon$, $S_{db} \leftarrow S_{db} + \epsilon$

⑥ ADAM OPTIMIZATION ALGORITHM (ADAPTIVE MOMENT ESTIMATION)

ESTE ALGORITMO COMBINA MOMENTUM y RMSprop

$V_{dW} = 0$, $S_{dW} = 0$, $V_{db} = 0$, $S_{db} = 0$

- V_{dW}, V_{db} actualizar con momentum y parámetro β_1
- S_{dW} , S_{db} " " " RMSprop y " β_2

→ Hacer corrección de sesgos de ambos

Actualización final:

$$W \leftarrow W - \alpha V_{dW} / \sqrt{S_{dW}} \quad , \quad b = b - \frac{\alpha V_{db}}{\sqrt{S_{db}}}$$

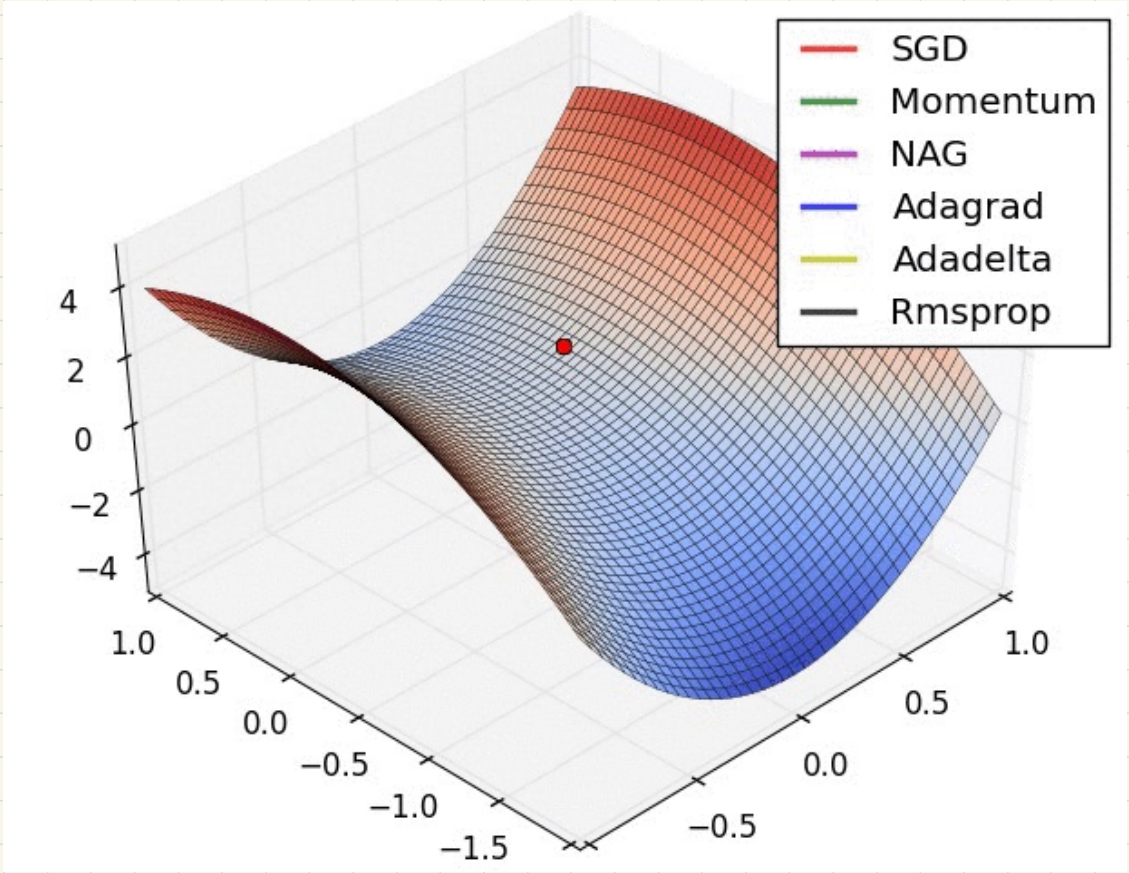
Hiperparâmetros

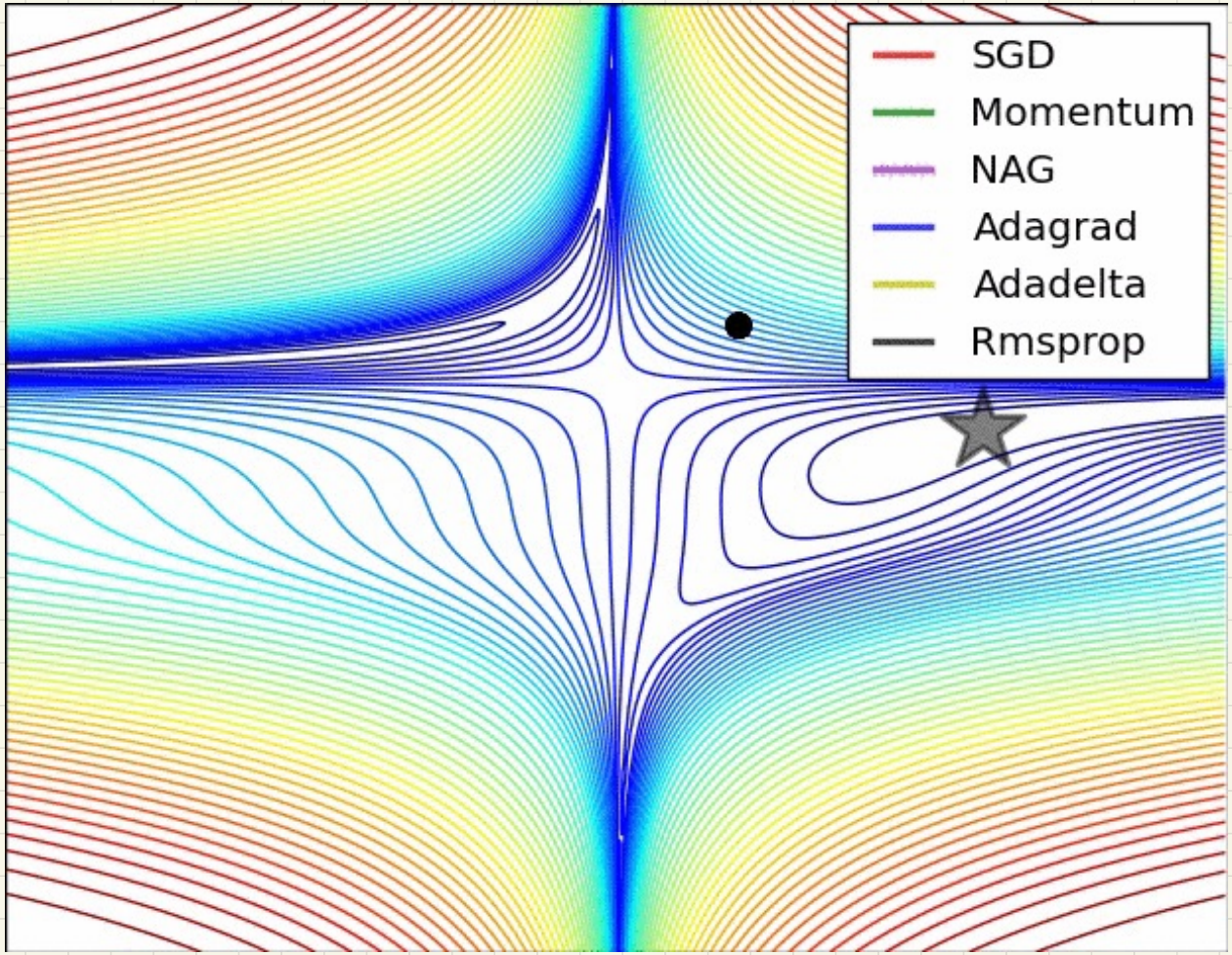
α learning rate calibrando

$\beta_1 \sim 0.9$

$\beta_2 \sim 0.999$

$\epsilon \sim 10^{-8}$


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

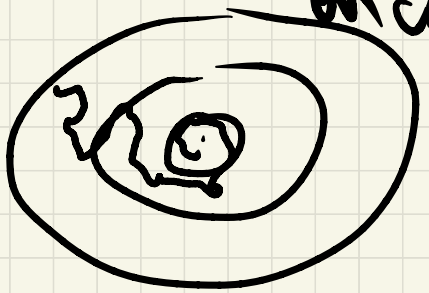
⑦ LEARNING RATE DECAY

- Si α es grande se pueden dar saltos ineficientes cerca del optimo
- Una forma de ir lo ajustando es:

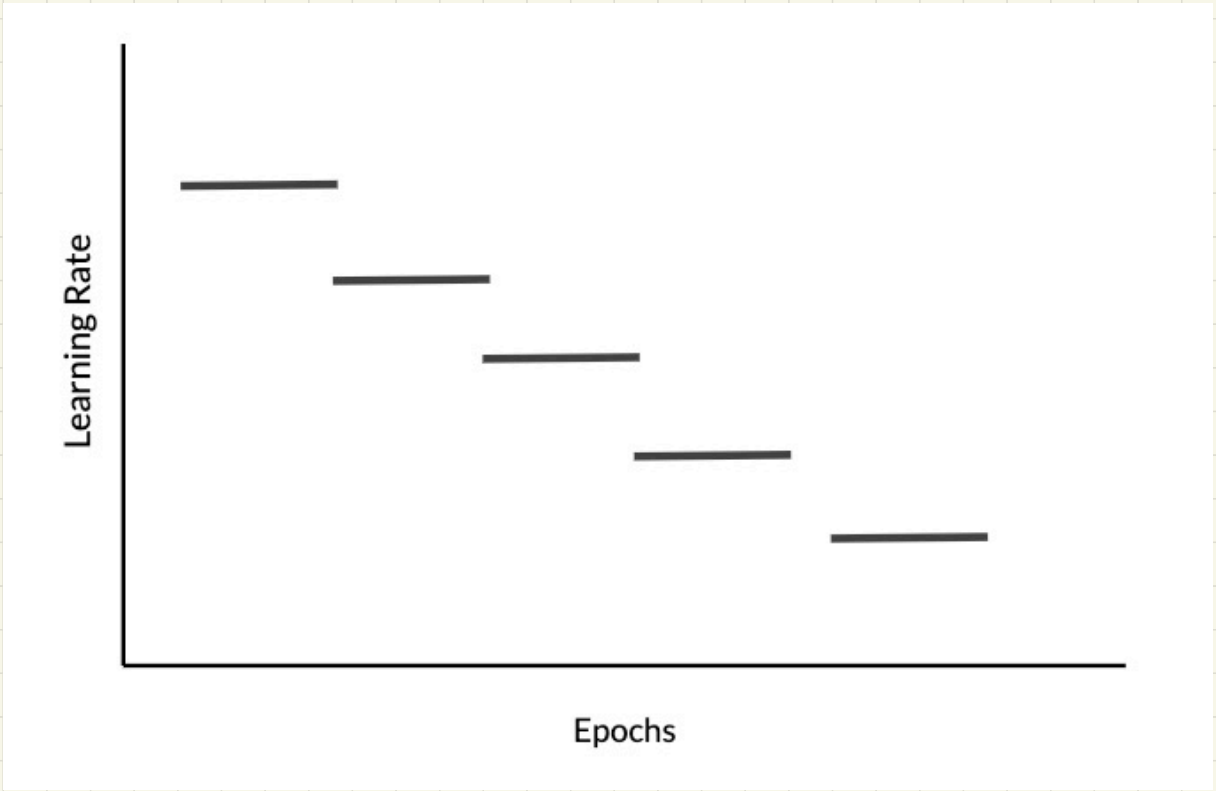
$$\alpha = \frac{1}{1 + \text{decayRate} \cdot \text{epochNumber}} \times \alpha_0$$

↓

hiperparámetro



Puede dar muchos saltos en cada mini batch


Learning Rate
Epochs

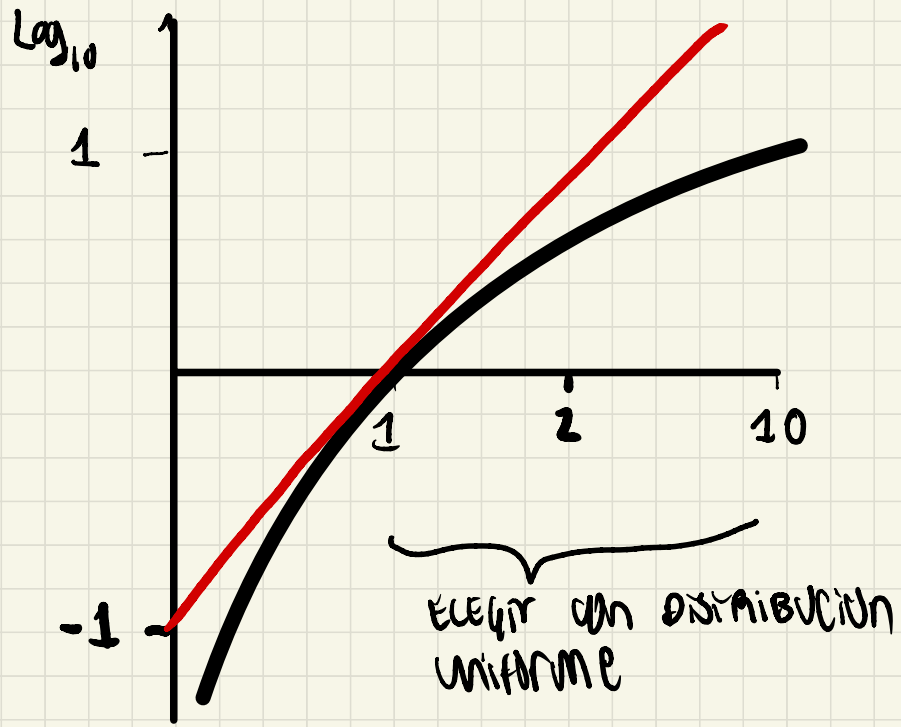
① CALIBRACIÓN DE HYPERPARÁMETROS

α, β, β_1, β_2, ε, L, n_l, learning rate decay, minibatch size
1 2 2 3 3 2

- RANKING DE MÁS IMPORTANTES: 1, 2, 3
- ESTRATEGIAS CALIBRACIÓN:
  1. ELEGIR ALEATORIAMENTE
  2. De grueso a fino
  3. ESCALA APROPIADA:

     Si $\alpha \in [0,1]$ por prob. más cercano a cero

     MUESTREAR USANDO ESCALA LOGARÍTMICA.


Log_{10}
1
-1
1
2
10
ELEGIR CON DISTRIBUCION UNIFORME

② BATCH NORMALIZATION

- ¿Cómo normalizar en pasos intermedios para facilitar el aprendizaje de los parámetros del paso siguiente?
- Suponemos que estamos en capa l

Definir $\mu^{[l]} = \frac{1}{m}\sum_{i=1}^{m} z^{[l](i)}$

$\sigma^{[l]} = \frac{1}{m}\sum (z^{[l](i)} - \mu^{[l]})^2$

$\Rightarrow z_{norm}^{[l](i)} = \frac{z^{[l](i)} - \mu^{[l]}}{\sqrt{\sigma^{[l]} + \varepsilon}}$

- BATCH normalization $\tilde{z}^{[l](i)} = \gamma z_{norm}^{[l](i)} + \beta$

γ, β → PARÁMETROS PARA APRENDER

- Obsérvese que si: $\left.\begin{array}{l} \gamma = \sqrt{\sigma^{[l]} + \varepsilon} \\ \beta = \mu \end{array}\right\} \Rightarrow \tilde{z}^{[l](i)} = z_{norm}^{[l](i)}$

Luego poder calibrar γ, β permite controlar la media y varianza de los z normalizados y que no sean cero y uno.

- PARA ENTRENAR en el entrenamiento de la red con cada mini-batch se puede hacer esto):

$$X^{\{1\}} \xrightarrow{W^{[1]}} z^{[1]} \xrightarrow{\gamma^{[1]}, \beta^{[1]}} \tilde{z}^{[1]} \rightarrow g^{[1]}(\tilde{z}^{[1]}) \xrightarrow{W^{[2]}} z^{[2]} \longrightarrow \dots$$

Obsérvese que no es necesario incluir el sesgo porque será normalizado se elimina

- Implementación de Gradiente Descendente

loop $t = 1 \dots$ num Mini batches

dado $X^{\{t\}}$ forward propagate

en cada capa reemplazar $Z^{[l]}$ por $\tilde{Z}^{[l]}$

hacer Backprop : $dW^{[l]}$, ~~$db^{[l]}$~~, $d\beta^{[l]}$, $d\gamma^{[l]}$

Actualizar parámetros de la forma estándar

end loop

- se puede combinar con momentum, RMSprop, Adam.

- ¿cuando se va a hacer la predicción con un ejemplo que μ, σ^2 se usa?

- Si el modelo se entrena con mini batches entonces todos tienen el mismo tamaño:

$X^{\{1\}}$ $X^{\{2\}}$ ...

$\downarrow$ $\downarrow$

$z^{[1]\{1\}}$ $z^{[1]\{2\}}$ $\rightarrow$ μ se calcula usando exponentially weighted average

$\sigma^{[1]\{1\}}$ $\sigma^{[1]\{2\}}$ $\rightarrow$ σ^2

- Este μ, σ^2 se usa en el momento de predecir un ejemplo: z_{norm} se calcula usando este μ, σ^2 y con esto $\tilde{z}$ y se calcula toda la red.

① MULTICLASES

- Sea C = # clases
- $n^{[L]} = 4$
- Función de pérdida: $\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{C} y_j \ln(\hat{y}_j)$
- Función de costo: $J(w, b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$
- Cuando uno comienza el backprop: $\frac{\partial \mathcal{L}}{\partial z^{[L]}} = dz^{[L]} = \hat{y} - y$ que es un vector $(4,1)$
- En vez de hacer a mano el resto la idea es usar un framework