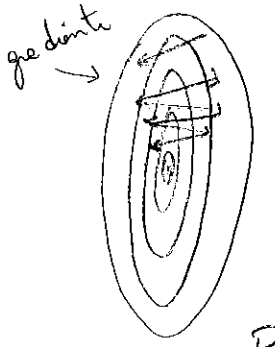
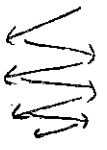
Optimización (Actualización) [empezar con "learning rate decay"]

+ Descenso de gradiente con momentum



pero me gustaría descender en dirección ↓

Solución posible: no llegar por el gradiente si no combinar gradientes anteriores y "promediando" la dirección horizontal se cancela



promedio ⟹

Para no querer promediar todo la historia ⟹ le doy mayor peso relativo a los últimos valores ⟹ promedio exponencial móvil!

En cada iteración mantengo el promedio móvil de $\frac{\partial \mathcal{L}}{\partial W}$ para cada parámetro W

$\beta = 0$ recupero descenso estándar

$$\begin{cases} \overline{\partial W} := \boxed{\beta}\,\overline{\partial W} + (1-\beta)\frac{\partial \mathcal{L}}{\partial W} & \leftarrow \text{promedio móvil} \\ W := W - \lambda\,\overline{\partial W} & \leftarrow \text{descender por el promedio del gradiente} \end{cases}$$

← nuevo (hiper) parámetro

Interpretación: velocidad + aceleración + fricción

- la regla de actualización la puedo escribir como

$$W := W + \left(\underbrace{-\lambda\beta}_{\alpha}\,\overline{\partial W} \underbrace{- \lambda(1-\beta)}_{\lambda'}\frac{\partial \mathcal{L}}{\partial W} \right)$$

esto induce otro cálculo para el promedio como

$$\begin{cases} V_{\partial W} := \alpha V_{\partial W} - \lambda' \frac{\partial \mathcal{L}}{\partial W} & \leftarrow \text{velocidad } V_{\partial W} \text{ (para el cambio en } W\text{) que se actualiza según } -\lambda'\frac{\partial \mathcal{L}}{\partial W} \text{ (aceleración) y según } \alpha \text{ mide la fricción} \\ W := W + V_{\partial W} & \leftarrow \text{parámetro cambia según la nueva velocidad} \end{cases}$$

implementación más típica de momentum ⟹ hiperparámetros: α, λ' ($\alpha = 0$ recupero descenso de gradiente estándar)

Variante Nesterov: Para cada parámetro W

$$\tilde{W} := W + \alpha V_{\partial W} \quad \leftarrow \text{lo hago para todos los parámetros}$$

$$V_{\partial W} := \alpha V_{\partial W} - \lambda' \frac{\partial \mathcal{L}}{\partial W}\Big|_{\tilde{W}} \quad \leftarrow \text{necesito hacer backpropagation para los parámetros } \tilde{W}$$

el gradiente es evaluado en $W + \alpha V_{\partial W} \Rightarrow$ trata de "corregir" el momentum estándar.

$$W := W + V_{\partial W} \quad \leftarrow \text{misma actualización que momentum}$$

Idea: Me anticipo al cambio de posición y calculo el gradiente allí

Valor de β usados: 0.9

* Algoritmos con tasa de aprendizaje adaptable

Idea: cambiar la tasa de aprendizaje durante el entrenamiento

— Más simple: no cambiar nada (Opción 0)

— No tan simple: hacer que la tasa vaya "decayendo" de a poco

¿Por qué? a medida que nos acercamos al mínimo queremos dar "pasos" más pequeños

Idea 1:

$$\lambda := \frac{1}{1 + d * t} \lambda_0$$

hiperparámetro $d \geq 0$

Idea 2:

$$\lambda := d^t * \lambda_0$$

hiperparámetro $d \leq 1$

Idea 3:
actualizar λ después de cada iteración hasta llegar a un mínimo. Pasar λ_0 a λ_T

$$\lambda := (1 - \frac{t}{T})\lambda_0 + \frac{t}{T}\lambda_T$$

λ_0 tasa inicial
λ_T tasa final (mínima)
actualización por T

En todos los casos puedo llegar a un mínimo luego de lo cual se mantiene constante. Todos se pueden hacer por epoch o minibatch (esto define el t)

— Adagrad: adaptar la tasa de aprendizaje de cada parámetro por separado escalándolos de manera inversamente proporcional a la raíz cuadrada de la suma de todos los valores históricos al cuadrado

Para cada parámetro W en cada iteración:

Valor sugerido de λ: 0.01

$$r_{dW} := r_{dW} + \left(\frac{\partial \mathcal{L}}{\partial W} * \frac{\partial \mathcal{L}}{\partial W}\right) \quad \text{multiplicación punto a punto}$$

$$W := W - \left(\lambda \frac{1}{\sqrt{r_{dW}}} * \frac{\partial \mathcal{L}}{\partial W}\right)$$

sqrt punto a punto

(para implementarlo se usa $\frac{1}{\sqrt{r_W} + \varepsilon}$ para estabilidad numérica con $\varepsilon = 10^{-7}$)

→ La dirección con el mayor gradiente disminuye su tasa de aprendizaje más rápido

→ Con: promediar desde el principio puede penalizar demasiado temprano la tasa

→ Adagrad fue diseñado para superficies convexas en donde tiene muy buenas propiedades (teóricas y prácticas), pero DL es altamente no convexo

- RMS Prop. [2012] Idea: similar a AdaGrad pero ponderando la suma de los cuadrados con un promedio exponencial móvil (root mean squared propagation)

β propuesto: 0.9
λ: 0.001

$$S_{\partial W} := \beta S_{\partial W} + (1-\beta) \frac{\partial \mathcal{L}}{\partial W} * \frac{\partial \mathcal{L}}{\partial W}$$

(punto a punto)

$$W := W - \lambda \frac{1}{\sqrt{S_{\partial W}}} * \frac{\partial \mathcal{L}}{\partial W}$$

(igual que antes para implementarlo se usa $\frac{1}{\sqrt{S_{\partial W} + \varepsilon}}$ con $\varepsilon \approx 10^{-7}$ o 10^{-8})

se puede combinar con momentum tb (aunque en ese caso queda con equivalente al siguiente)

- Adam: [2015] (Adaptive Moments) RMSProp + Momentum

$$\overline{\partial W} := \beta_1 \overline{\partial W} + (1-\beta_1) \frac{\partial \mathcal{L}}{\partial W} \quad \leftarrow \text{momentum}$$

$$S_{\partial W} := \beta_2 S_{\partial W} + (1-\beta_2) \frac{\partial \mathcal{L}}{\partial W} * \frac{\partial \mathcal{L}}{\partial W} \quad \leftarrow \text{RMSProp}$$

$$W := W - \lambda \frac{1}{\sqrt{S_{\partial W}}} * \overline{\partial W}$$

Típicamente se implementa con "bias correction" $\Rightarrow$ al inicio no tenemos con qué promedios (en cálculo de $\overline{\partial W}$ y $S_{\partial W}$) lo que implica que estamos considerando como si los primeros valores fueran todos 0 $\Rightarrow$ esto induciría un bias hacia 0 al inicio del entrenamiento. Corregimos este hecho dividiendo por un número menor que 1 (así amplificando el promedio inicial) pero que tenderá rápidamente a 1 para volver al promedio que calculamos. Usamos $(1-\beta_1^t)$ para $\overline{\partial W}$ y $(1-\beta_2^t)$ para $S_{\partial W}$ con t la iteración del descenso de gradiente.

Formulación final

$$\overline{\partial W} := \beta_1 \overline{\partial W} + (1-\beta_1) \frac{\partial \mathcal{L}}{\partial W} \qquad S_{\partial W} := \beta_2 S_{\partial W} + (1-\beta_2) \frac{\partial \mathcal{L}}{\partial W} * \frac{\partial \mathcal{L}}{\partial W}$$

$$\hat{\partial W} := \frac{\overline{\partial W}}{1-\beta_1^t} \qquad \hat{S}_{\partial W} := \frac{S_{\partial W}}{1-\beta_2^t}$$

$\varepsilon = 10^{-8}$ para estabilidad

$$W := W - \lambda \frac{1}{\sqrt{\hat{S}_{\partial W}}} * \hat{\partial W}$$

Tres hiperparámetros $\lambda, \beta_1, \beta_2$. Valores sugeridos: $\lambda = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ cambiar solo estos !!! si fuera necesario

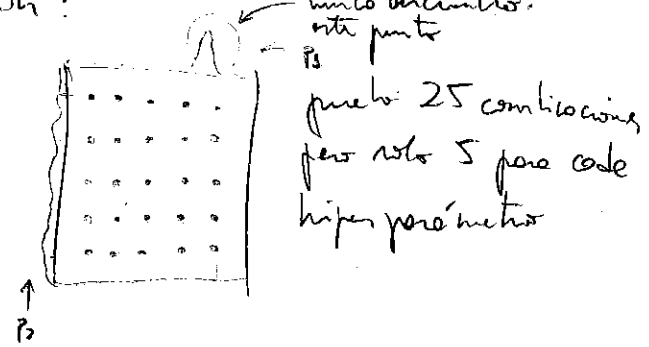
Eligiendo hiperparámetros: hay demasiados hiperparámetros :(
no hay que decir más que hubo la mejor combinación

Tip 1: para arquitecturas usar algunas que el estado del arte dice que tiene resultados (aunque no dicen: con/sin tamaño de capas y cantidad si hay underfitting)

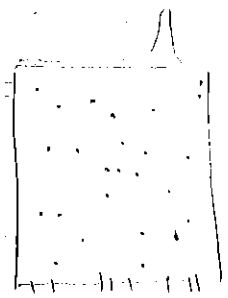
Tip 2: para learning rate o parámetro de decay, buscar en escala exponencial
0.1 ; 0.01 , 0.001 , etc
si quiero buscar entre 0.1 y 0.001 $\Longrightarrow$ elijo un número $t \in [1,3]$
y pruebo 10^{-t}
para capas o neuronas en escala lineal está bien

Tip 3: tasa de aprendizaje debe ser el hiperparámetro más importante

Tip 4: hacer búsqueda random de los hiperparámetros (NO GRID SEARCH!!!)
razón:



pruebo 25 combinaciones pero solo 5 para cada hiperparámetro



pruebo 25 valores para cada hiperparámetro

Tip 5: Zoom-in en región donde los hiperparámetros dieron buenos resultados y continuar ahí