Aprendizaje profundo

Entrenamiento de redes profundas

Gibran Fuentes-Pineda

Agosto 2025

Motivación de redes neuronales profundas (1)

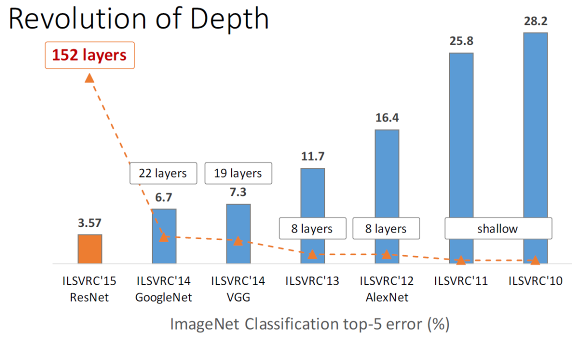


Imagen tomada de diapositivas de Kaiming He (ICML 2016)

Motivación de redes neuronales profundas (2)

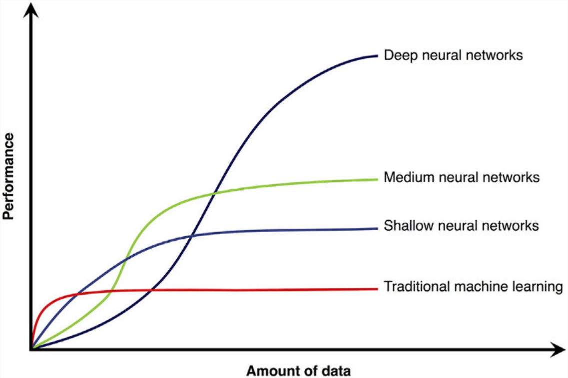
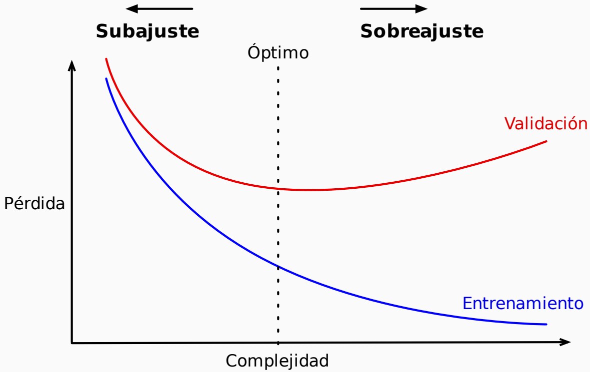
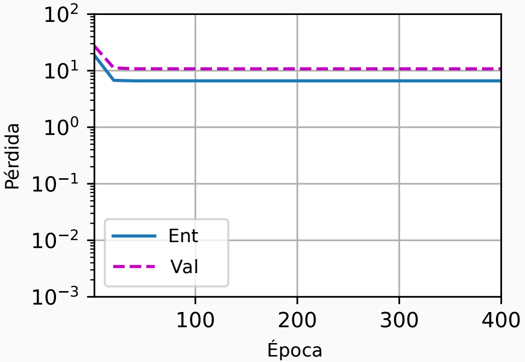


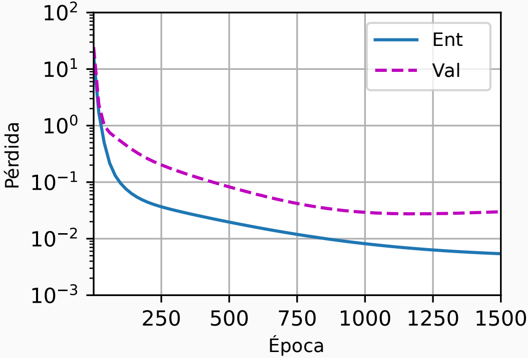
Imagen tomada de Tang et al. 2018

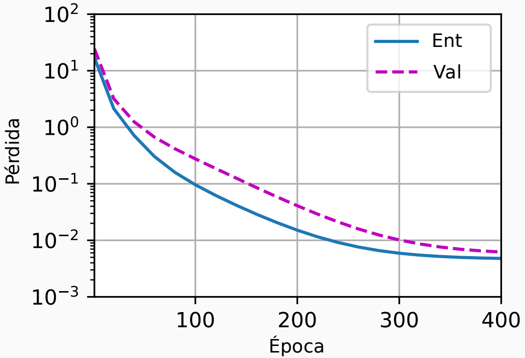
Sobreajuste y complejidad




10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
Pérdida
100
200
300
400
Época
Ent
Val

Sobreajuste





Ent
Val
Pérdida
Época
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
100
200
300
400

Sobreajuste y regularización: estrategias de regularización

- Estrategias para disminuir sobreajuste en redes neuronales
  - Penalización de función de pérdida
  - Adición de ruido a entradas, salidas y/o parámetros
  - Ensambles
  - Paro temprano (*early stopping*)
  - Aprendizaje de múltiples tareas
  - Dropout
  - Acrecentamiento de datos (*data augmentation*)
  - Normalización por lotes (*batch normalization*)
  - Descenso por gradiente estocástico (SGD) y variantes
  - Suavizado de etiquetas (*label smoothing*)
  - Promediado estocástico de parámetros (*stochastic weight averaging*)

Sobreajuste y regularización: penalización por norma ℓ_1 y ℓ_2

- Norma ℓ_1

$$\tilde{E}(\boldsymbol{\theta}) = -\sum_{i=1}^{n}\{y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)\} + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_1$$

- Norma ℓ_2

$$\tilde{E}(\boldsymbol{\theta}) = -\sum_{i=1}^{n}\{y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)\} + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$$

Sobreajuste y regularización: paro temprano

- Se detiene entrenamiento si pérdida o métrica de validación no aumenta después de varios pasos
- Usualmente se elige el modelo con mejor desempeño en el conjunto de validación

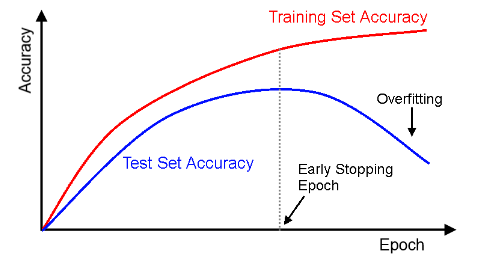


Imagen tomada de `https://deeplearning4j.org/earlystopping`

- Tener una representación genérica compartida entre 2 tareas relacionadas

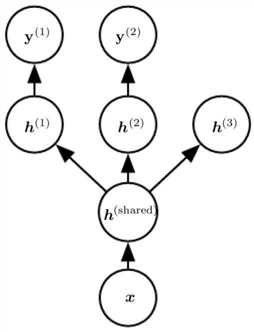
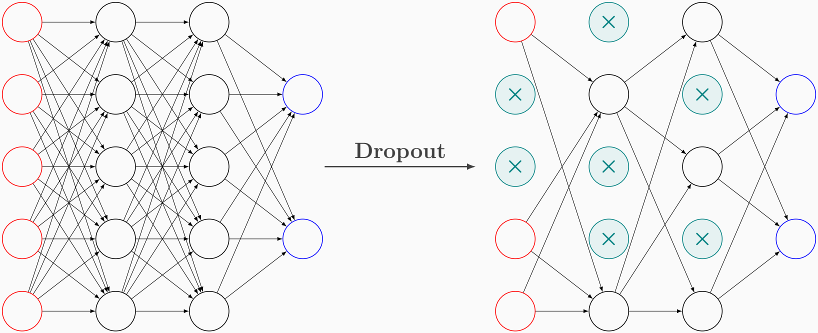


Imagen tomada de Goodfellow et al. Deep Learning, 2016

- Se desactiva neuronas de forma aleatoria [1] para evitar co-adaptación



[1] Probabilidad es típicamente cercana a 0.5

- La salida de la i-ésima neurona está dada por

$$\begin{aligned} z_i^{\{\ell+1\}} &= \mathbf{w}_i^{\{\ell+1\}} \widetilde{\mathbf{y}}^{\{\ell\}} + b_i^{\{\ell+1\}} \\ y_i^{\{\ell+1\}} &= \phi(z_i^{\{\ell+1\}}) \end{aligned}$$

donde $\widetilde{\mathbf{y}}$ es una máscara binaria sobre las salidas de las neuronas con 1s para las activas y 0s para las inactivas

$$\begin{aligned} r_j &\sim Bernoulli(p_{\text{dropout}}) \\ \widetilde{\mathbf{y}}^{\{\ell\}} &= \mathbf{r}^{\{\ell\}} * \mathbf{y}^{\{\ell\}} \end{aligned}$$

- p_{dropout} es un hiperparámetro que indica la probabilidad de que una neurona se mantenga activa

- Puede verse como entrenar simultáneamente múltiples redes eliminando neuronas de una red base

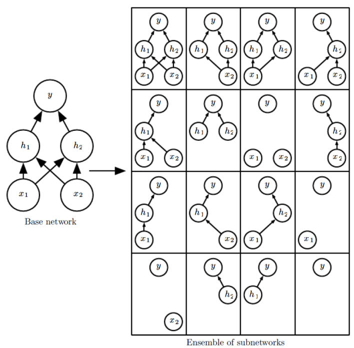


Imagen tomada de Goodfellow et al. Deep Learning, 2016

- En vez de promediar las salidas de todas las redes entrenadas, se obtiene la salida de una sola red con los pesos y sesgos ($\boldsymbol{\theta} = \{\mathsf{W}, \mathsf{b}\}$) escalados

$$\boldsymbol{\theta}_{\text{inferencia}} = p_{\text{dropout}} \cdot \boldsymbol{\theta}$$

- De esta forma se combinan 2^n redes en una sola

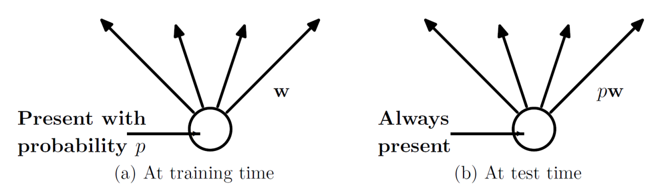


Imagen tomada de Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014

- De forma alternativa se divide entre p_{dropout} a $\widetilde{\mathbf{y}}^{\{\ell\}}$ durante el entrenamiento.

Sobreajuste y generalización: suavizado de etiquetas

- Se agrega ruido a la representación 1 de k de las etiquetas, reemplazando los ceros con $\frac{\epsilon}{k-1}$ y los unos con $1-\epsilon$.

- $\epsilon \in (0,1)$ se puede interpretar como la probabilidad de que una etiqueta sea errónea.

- Por ej., para $k=5$, la representación de la etiqueta para la clase 3
$$[0,0,1,0,0]\,.$$
La etiqueta suavizada con $\epsilon = 0.01$ sería
$$[0.0025, 0.0025, 0.99, 0.0025, 0.0025]\,.$$

- *Stochastic Weight Averaging* (SWA) promedia los pesos y sesgos de las actualizaciones realizadas al entrenar una red usando una programación de la tasa de aprendizaje modificada

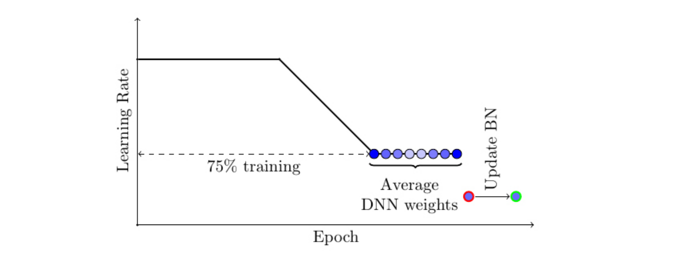


Imagen tomada de `https://pytorch.org/blog/pytorch-1.6-now-includes-stochastic-weight-averaging/`.

- SWA es similar a *Polyak-Ruppert averaging*[2] pero usa una tasa de aprendizaje cíclica o un tasa grande constante y un promedio regular, en lugar de una tasa que decae y el promedio móvil exponencialmente ponderado.
- Ha mostrado mejorar el rendimiento de modelos entrenados en diferentes aplicaciones, agregando muy poco costo computacional

[2] Propuesto de forma independiente por B. Polyak y D. Ruppert.

- Problemas con el desvanecimiento y explosión de respuestas (hacia adelante) y gradientes (hacia atrás)

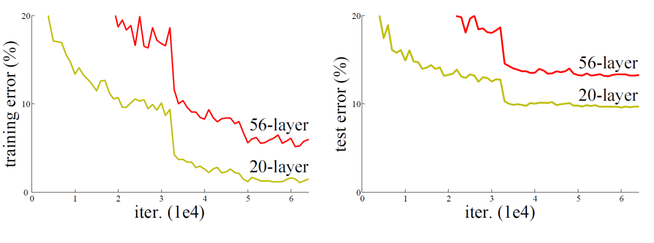
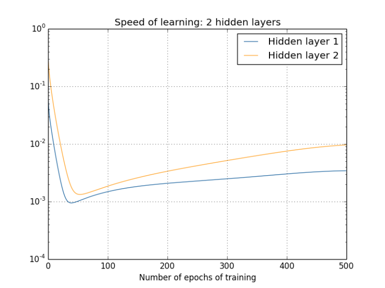


Imagen tomada de He et al. *Deep Residual Learning for Image Recognition*, 2015

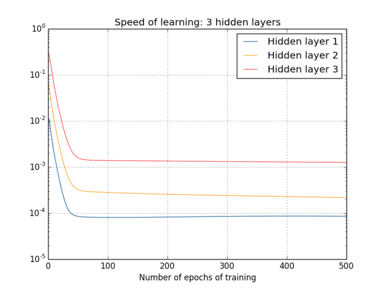
- Gradientes de primeras capas se vuelven muy pequeños si la red es muy profunda
- Muy lento actualizar pesos de estas capas



Tomado de `http://neuralnetworksanddeeplearning.com/chap5.html`

Explosión y desvanecimiento del gradiente: 3 capas ocultas

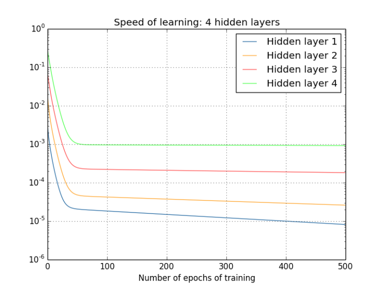
- Gradientes de primeras capas se vuelven muy pequeños si la red es muy profunda
- Muy lento actualizar pesos de estas capas



Tomado de `http://neuralnetworksanddeeplearning.com/chap5.html`

Explosión y desvanecimiento del gradiente: 4 capas ocultas

- Gradientes de primeras capas se vuelven muy pequeños si la red es muy profunda
- Muy lento actualizar pesos de estas capas



Tomado de `http://neuralnetworksanddeeplearning.com/chap5.html`

Explosión y desvanecimiento del gradiente: mitigación

- Recorte de gradientes (*gradient clipping*)
- Emplear funciones de activación no saturadas en capas ocultas
- Incorporar conexiones residuales a la red
- Inicializar pesos y sesgos con heurísticas apropiadas
- Emplear normalización por lotes

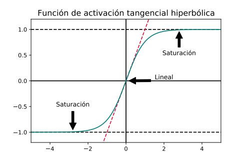
Explosión y desvanecimiento del gradiente: clipping

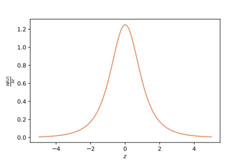
- Estrategia para evitar la explosión del gradiente
- La idea general es limitar la magnitud de los valores de los gradientes.
- Por ej.

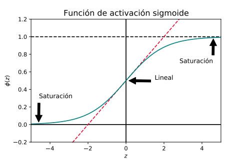
$$\text{Si } \|\nabla\mathcal{L}(\boldsymbol{\theta})\| > \eta, \text{entonces } \frac{\eta \cdot \nabla\mathcal{L}(\boldsymbol{\theta})}{\|\nabla\mathcal{L}(\boldsymbol{\theta})\|}$$

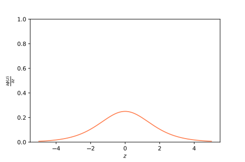
donde η es un umbral de la norma.

Explosión y desvanecimiento del gradiente: función de activación (1)

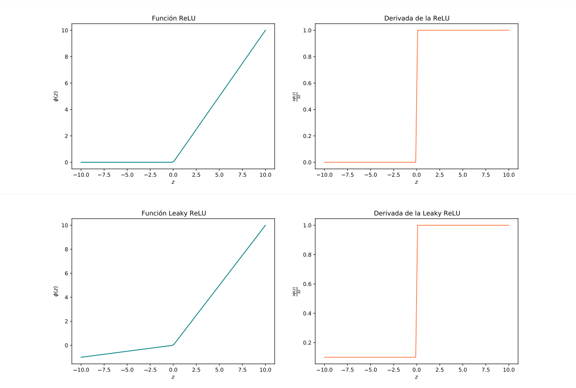




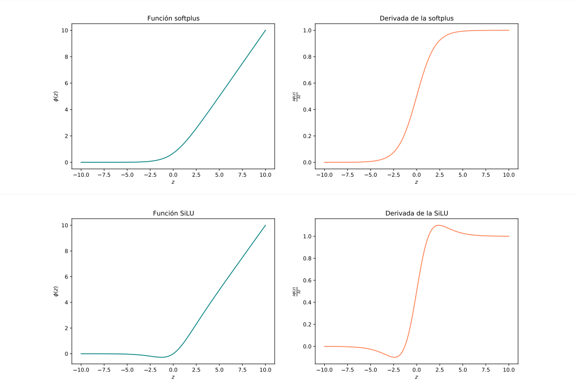




Explosión y desvanecimiento del gradiente: función de activación (2)



Explosión y desvanecimiento del gradiente: función de activación (3)



Explosión y desvanecimiento del gradiente: conexiones residuales

- Agregando conexiones residuales en la arquitectura

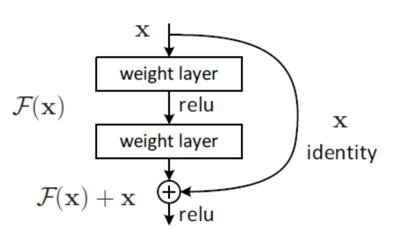


Imagen tomada de He et al. Deep Residual Learning for Image Recognition, 2015

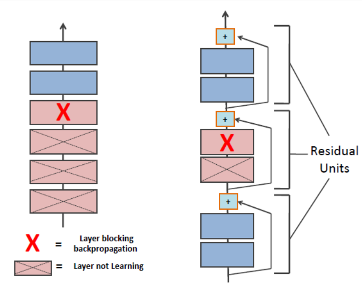


Imagen de Kevin Murphy, tomada de Probabilistic Machine Learning: An Introduction, 2021

Explosión y desvanecimiento del gradiente: inicialización (1)

- Muestreados de normal con media 0 y varianza 0.01.
    - Funciona para redes pequeñas
    - Para redes profundas activaciones tienden a volverse 0
- Muestreados de normal con media 0 y varianza 1
    - Genera saturación de neuronas y gradientes se vuelven 0

- Muestrear de normal con media 0 y varianza fija σ^2 puede contribuir al desvanecimiento/explosión de gradientes
  - El valor esperado y la varianza de la salida de una neurona lineal $a = \sum_{i=1}^{d} (w_i \cdot x_i)$ con d entradas es

$$\mathbb{E}[a] = \mathbb{E}\left[\sum_{j=1}^{d}\left(w_j \cdot x_j\right)\right] = \sum_{j=1}^{d} \overbrace{\mathbb{E}[w_j \cdot x_j]}^{w_j \perp\!\!\!\perp x_j} = \sum_{j=1}^{d}\left(\overbrace{\mathbb{E}[w_j]}^{0} \cdot \mathbb{E}[x_j]\right) = 0$$

$$\mathbb{V}[a] = \mathbb{V}\left[\sum_{j=1}^{d}\left(w_j \cdot x_j\right)\right] = \sum_{j=1}^{d} \mathbb{V}\left[w_j \cdot x_j\right] = \sum_{j=1}^{d} \mathbb{E}[(w_j \cdot x_j)^2] - \overbrace{\mathbb{E}[w_j \cdot x_j]}^{0}{}^2$$

$$= \sum_{j=1}^{d} \overbrace{\mathbb{E}[w_j^2 \cdot x_j^2]}^{w_j \perp\!\!\!\perp x_j} = \sum_{j=1}^{d} \overbrace{\mathbb{E}[w_j^2]}^{\sigma^2} \cdot \overbrace{\mathbb{E}[x_j^2]}^{\tau^2} = d \cdot \sigma^2 \cdot \tau^2$$

- Para una capa con n_e entradas y n_s salidas
  - Uniforme de Glorot y Bengio (2010)

$$\boldsymbol{\theta} \sim \mathcal{U}\left[-\sqrt{\frac{6}{n_e + n_s}}, \sqrt{\frac{6}{n_e + n_s}}\right]$$

  - Normal de Glorot y Bengio (2010)

$$\boldsymbol{\theta} \sim \mathcal{N}\left(0, \frac{2}{n_e + n_s}\right)$$

  - Uniforme de He et al. (2015)

$$\boldsymbol{\theta} \sim \mathcal{U}\left[-\sqrt{\frac{2}{n_e}}, \sqrt{\frac{2}{n_e}}\right]$$

  - Normal de He et al. (2015)

$$\boldsymbol{\theta} \sim \mathcal{N}\left(0, \frac{2}{n_e}\right)$$

- Cambio en distribución de activaciones por cambio en parámetros durante entrenamiento

- Hace más lento el aprendizaje

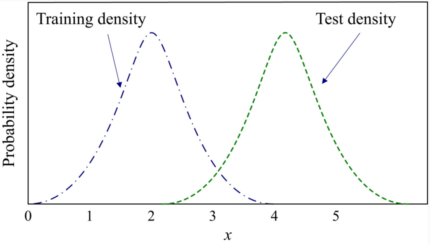


Imagen tomada de Raza et al. EWMA model based shift-detection methods for detecting covariate shifts in non-stationary environments, 2015

1. Media y varianza del lote

$$\boldsymbol{\mu}_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} \mathsf{x}^{\{i\}}$$

$$\boldsymbol{\sigma}^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} (\mathsf{x}^{\{i\}} - \boldsymbol{\mu}_{\mathcal{B}}))^2$$

2. Normalización

$$\hat{\mathsf{x}}^{\{i\}} \leftarrow \frac{\mathsf{x}^{\{i\}} - \boldsymbol{\mu}_{\mathcal{B}}}{\sqrt{\boldsymbol{\sigma}^2_{\mathcal{B}} + \epsilon}}$$

3. Escalado y desplazamiento

$$\mathsf{y}^{\{i\}} \leftarrow \boldsymbol{\gamma} \odot \hat{\mathsf{x}}^{\{i\}} + \boldsymbol{\beta}$$

donde $\odot$ es el producto de Hadamard (elemento a elemento), ϵ es un valor pequeño y m es el tamaño del lote.

Capas de normalización: normalización por lotes (2)

1. Normalizar la red con mini-lote
2. Entrenar la red con retro-propagación
3. Transformar estadísticos del lote a estadísticos de población

Capas de normalización: beneficios de la normalización por lotes

- Acelera el entrenamiento
- Permite tasas de aprendizaje más grandes
- Facilita la heurísticas de inicialización de pesos y sesgos
- Hace posible usar funciones de activación saturadas (por ej. sigmoide)
- Actúa como un tipo de regularizador
- Facilita la creación de redes profundas

- Estima los estadísticos sobre las neuronas de la misma capa ℓ

$$\mu^{\{\ell\}} = \frac{1}{h} \cdot \sum_{i=1}^{h} a_i^{\{\ell\}}$$

$$\sigma^{\{\ell\}} = \sqrt{\frac{1}{h} \cdot \sum_{i=1}^{h} \left(a_i^{\{\ell\}} - \mu^{\{\ell\}}\right)^2}$$

donde h es el número de neuronas en la capa.

- Todas las neuronas de la capa ℓ se normalizan usando la misma $\mu^{\{\ell\}}$ y $\sigma^{\{\ell\}}$, pero diferentes ejemplos tienen diferentes estadísticos.

- Al igual que la normalización por lotes, se agrega un desplazamiento y escalado adaptable

$$\hat{a}^{(i)} = \frac{\boldsymbol{\gamma}}{\sigma^{(i)}} \odot (\mathsf{a}^{(i)} - \mu^{(i)}) + \boldsymbol{\beta}$$

donde γ y β son parámetros que se entrenan.

- La normalización por capas realiza la misma operación tanto en entrenamiento como en prueba

Promedio móvil ponderado exponencialmente (PMPE)

- Definido por

$$e^{[t+1]} = \beta \cdot e^{[t]} + (1 - \beta) \cdot s^{[t]}$$

donde $s^{[t]}$ y $e^{[t]}$ son el valor y el PMPE en el tiempo t

- Expandiendo

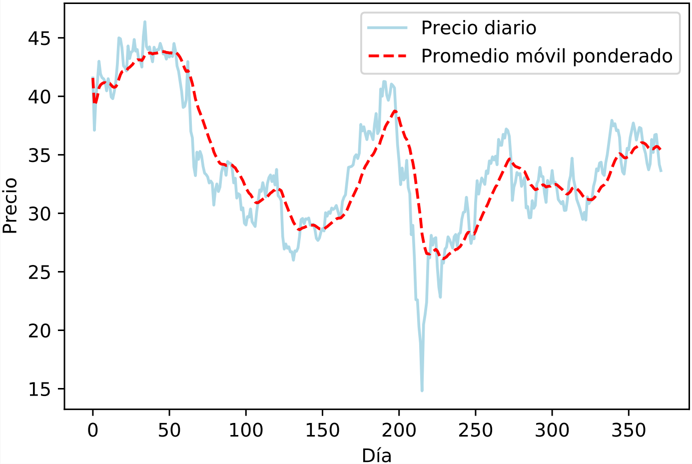
$$\begin{aligned} e^{[1]} &= s^{[1]} \\ e^{[2]} &= \beta \cdot e^{[1]} + (1 - \beta) \cdot s^{[1]} \\ e^{[3]} &= \beta \cdot e^{[2]} + (1 - \beta) \cdot s^{[2]} \\ \vdots &= \vdots \\ e^{[n]} &= \beta \cdot e^{[n-1]} + (1 - \beta) \cdot s^{[n-1]} \end{aligned}$$

- Visto de otra forma

$$
\begin{aligned}
e^{[4]} &= \beta \cdot \left\{(1-\beta) \cdot s^{[3]} + \beta \cdot \left[(1-\beta) \cdot s^{[2]} + \beta s^{[1]}\right]\right\} + (1-\beta) \cdot s^{[4]} \\
&= (1-\beta) \cdot \left[s^{[4]} + \beta \cdot s^{[3]} + \beta \cdot \beta \cdot s^{[2]} + \frac{\beta \cdot \beta \cdot \beta}{(1-\beta)} \cdot s^{[1]}\right]
\end{aligned}
$$

Promedio móvil ponderado exponencialmente (PMPE)

•



Recordando el descenso por gradiente estocástico

- Actualiza iterativamente los parámetros con base en los gradientes de la función de pérdida

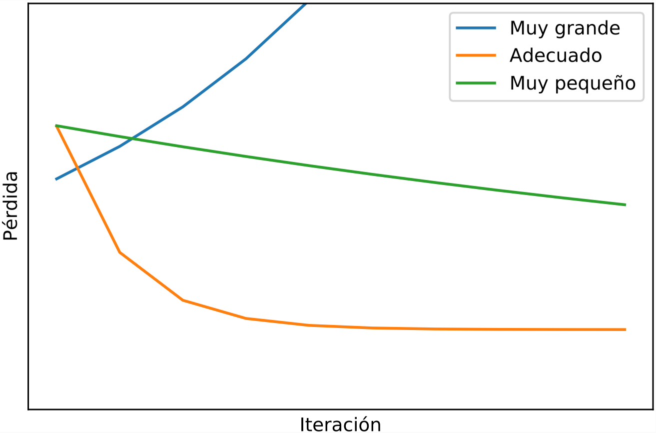
$$\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]} - \alpha \nabla \mathcal{L}(\boldsymbol{\theta}^{[t]})$$

  donde

$$\nabla \mathcal{L}(\boldsymbol{\theta}^{[t]}) = \left[\frac{\partial \mathcal{L}}{\partial \theta_0^{[t]}}, \cdots, \frac{\partial \mathcal{L}}{\partial \theta_d^{[t]}} \right]$$

- El descenso por gradiente estocástico (SGD) aproxima $\nabla \mathcal{L}(\boldsymbol{\theta}^{[t]})$ con un minilote de ejemplos de entrenamiento

Sensibilidad a tasa de aprendizaje α



Programación de la tasa de aprendizaje

- Constante por periodos
- Declive polinomial
- Declive exponencial

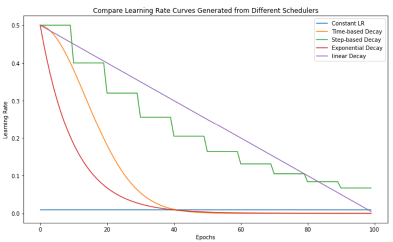


Imagen tomada de Katherine (Yi) Li. How to Choose a Learning Rate Scheduler for Neural Networks. MLOps Blog, 2024.

- Cíclico

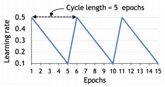


Figure 4: Cyclic learning rate policy (see Eq.1) with linear learning rate decay and warm restarts.

Figura tomada de Mehta et al. in ESPNetv2: A Light-weight, Power Efficient, and General Purpose Convolutional Neural Network, 2019.

- Introduce término de velocidad a la actualización (acumula declive)[3]

$$
\begin{aligned}
\mathsf{m}^{[t+1]} &= \mu \cdot \mathsf{m}^{[t]} - \alpha \cdot \nabla \mathcal{L}(\boldsymbol{\theta}^{[t]}) \\
\boldsymbol{\theta}^{[t+1]} &= \boldsymbol{\theta}^{[t]} + \mathsf{m}^{[t+1]}
\end{aligned}
$$

- Momento de Nesterov

$$
\begin{aligned}
\mathsf{m}^{[t+1]} &= \mu \cdot \mathsf{m}^{[t]} - \alpha \cdot \nabla \mathcal{L}(\boldsymbol{\theta}^{[t]} + \mu \cdot \mathsf{m}^{[t]}) \\
\boldsymbol{\theta}^{[t+1]} &= \boldsymbol{\theta}^{[t]} + \mathsf{m}^{[t+1]}
\end{aligned}
$$

[3] En PyTorch se calcula $\mathsf{m}^{[t+1]} = \mu \cdot \mathsf{m}^{[t]} + \nabla \mathcal{L}(\boldsymbol{\theta}^{[t]})$ y $\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]} - \alpha \cdot \mathsf{m}^{[t+1]}$

- Actualiza los parámetros a partir de los promedios móviles ponderados de los gradientes al cuadrado (segundo momento de los gradientes)

$$\mathsf{v}^{[t+1]} = \beta \cdot \mathsf{v}^{[t]} + (1-\beta) \cdot \left[\nabla \mathcal{L}(\boldsymbol{\theta}^{[t]})\right]^2$$
$$\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]} - \frac{\alpha}{\sqrt{\mathsf{v}^{[t+1]} + \epsilon}} \odot \nabla \mathcal{L}(\boldsymbol{\theta}^{[t]})$$

donde $\odot$ denota el producto de Hadamard

Optimizador Adam (1)

- Se estima el primer (la media) y segundo (la varianza no centrada) momentos de los gradientes

$$\mathrm{m}^{[t+1]} = \beta_1 \cdot \mathrm{m}^{[t]} + (1 - \beta_1) \cdot \nabla \mathcal{L}(\boldsymbol{\theta}^{[t]})$$

$$\mathrm{v}^{[t+1]} = \beta_2 \cdot \mathrm{v}^{[t]} + (1 - \beta_2) \cdot \left[\nabla \mathcal{L}(\boldsymbol{\theta}^{[t]})\right]^2$$

- Debido a que estas estimaciones están sesgadas hacia 0 (se inicializan con 0), se realiza una corrección

$$\hat{\mathrm{m}}^{[t+1]} = \frac{\mathrm{m}^{[t+1]}}{1 - \beta_1^{t+1}}$$

$$\hat{\mathrm{v}}^{[t+1]} = \frac{\mathrm{v}^{[t+1]}}{1 - \beta_2^{t+1}}$$

donde β_1^{t+1} y β_2^{t+1} son los factores de ponderación $\beta_1, \beta_2 \in [0, 1)$ elevados a la potencia $t + 1$

- Para actualizar los parámetros se usan las estimaciones de los momentos de los gradientes en el tiempo t

$$\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]} - \frac{\alpha}{\sqrt{\hat{\mathsf{v}}^{[t+1]}} + \epsilon} \cdot \hat{\mathsf{m}}^{[t+1]}$$