Aprendizaje profundo

Entrenamiento de redes neuronales profundas

Gibran Fuentes-Pineda

Septiembre-Octubre 2021

Preprocesamiento de datos

Preprocesamiento

- Re-escalado

$$\mathbf{x}' = \frac{\mathbf{x} - min(\mathbf{x})}{max(\mathbf{x}) - min(\mathbf{x})}$$

- Estandarización

$$\mathbf{x}' = \frac{\mathbf{x} - \bar{x}}{\sigma}$$

- Magnitud unitaria

$$\mathbf{x}' = \frac{\mathbf{x}}{\|\mathbf{x}\|}$$

Preprocesamiento: imágenes

- Radio uniforme

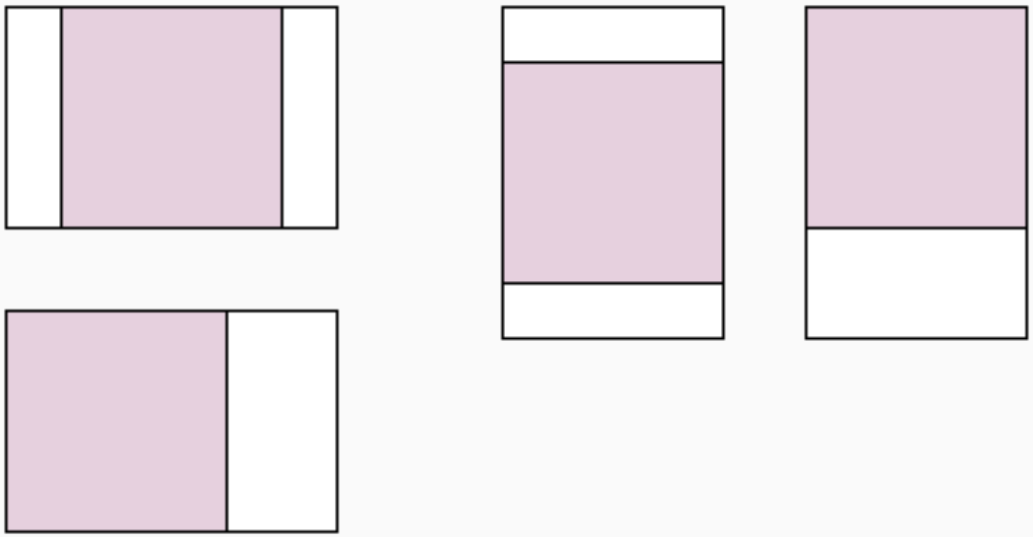


Imagen tomada de Nikhil B. Image Data Pre-Processing for Neural Networks, 2017.

Preprocesamiento: imágenes

- Radio uniforme
- Escalado de valores de pixeles

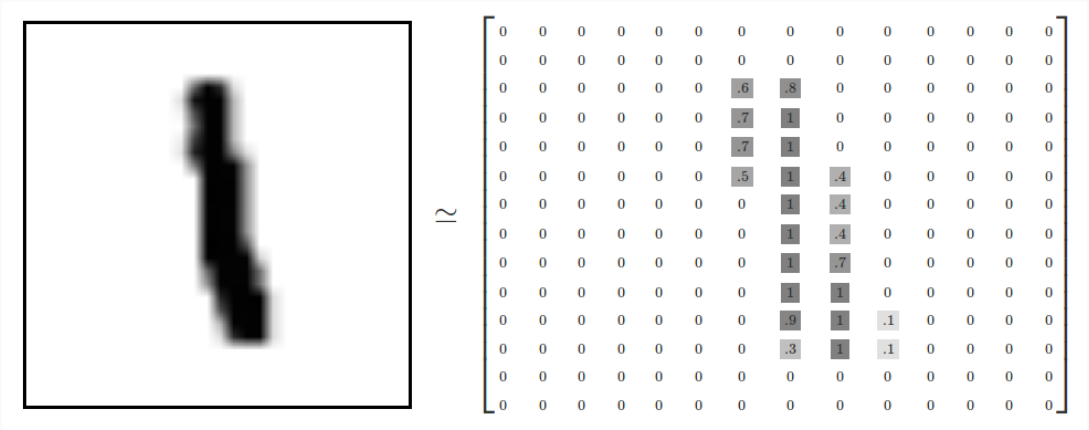


Imagen tomada de presentación de Yubei Chen. Part I: Manifold Learning, 2018.

Preprocesamiento: audio

- Filtrado

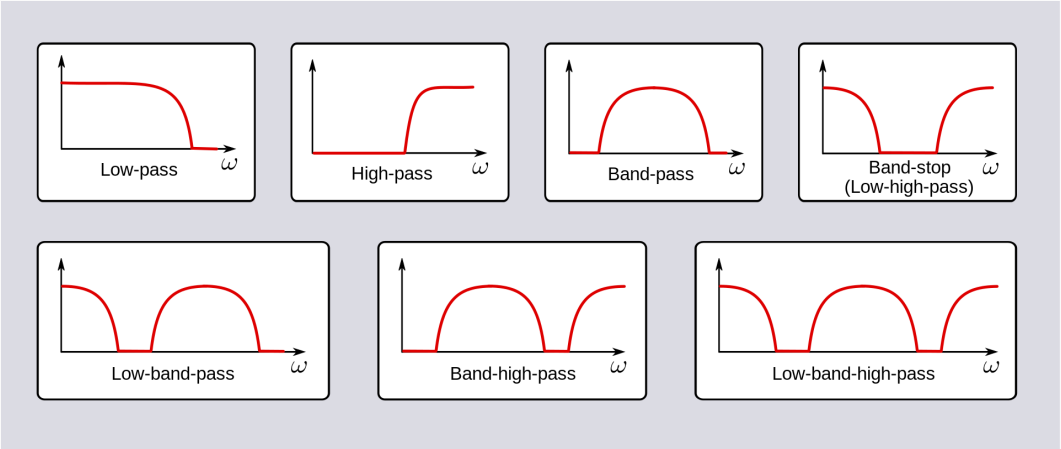


Imagen del usuario SpinningSpark de Wikipedia (entrada Filter (signal processing)). CC BY-SA 3.0

Preprocesamiento: audio

- Filtrado
- Espectograma

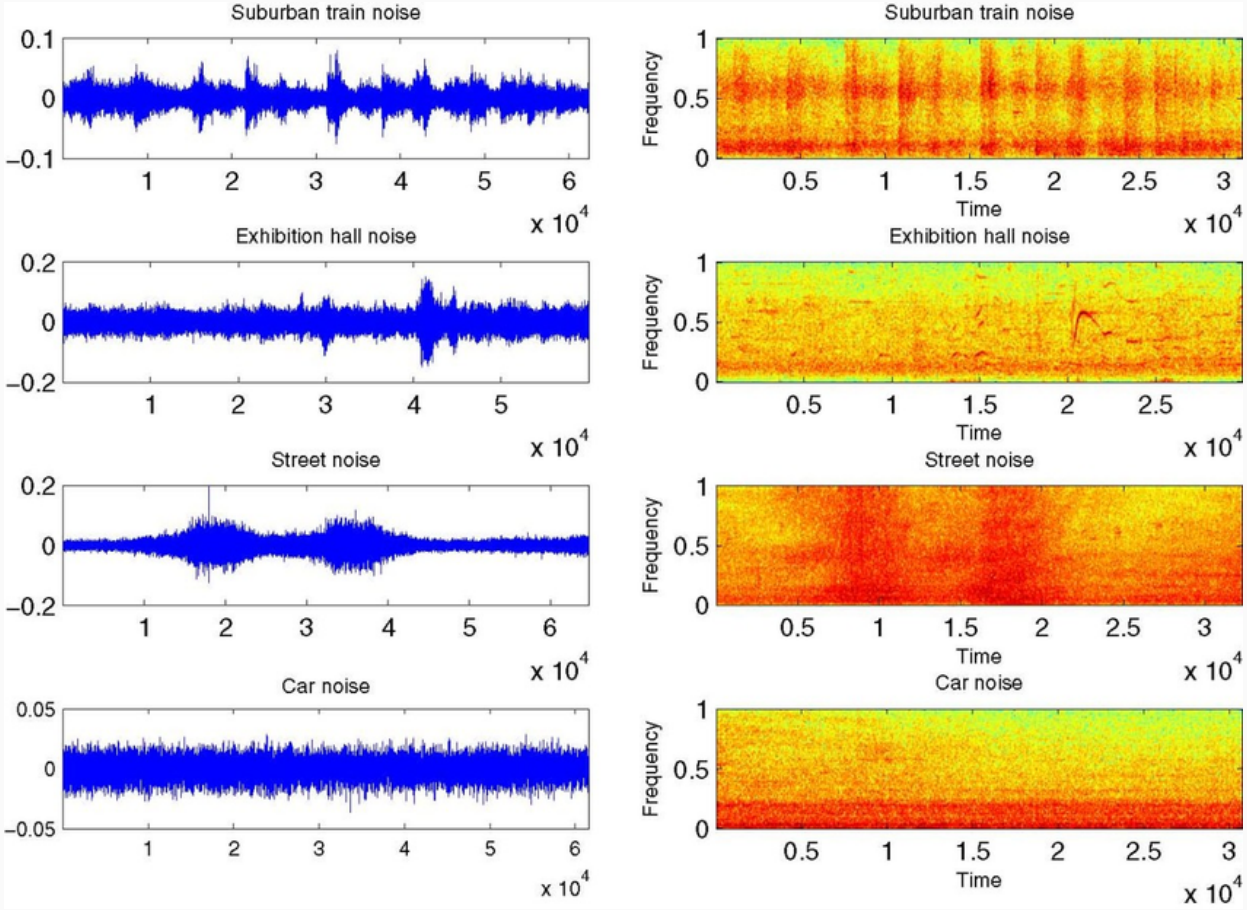


Imagen tomada de Zouhir and Ouni. A bio-inspired feature extraction for robust speech recognition, 2014.

Preprocesamiento: texto como bolsas

- Bolsas de palabras
  - Stopwords
  - Stemming
  - Quitar caracteres

Preprocesamiento: representaciones distribuidas

- Encajes de palabra

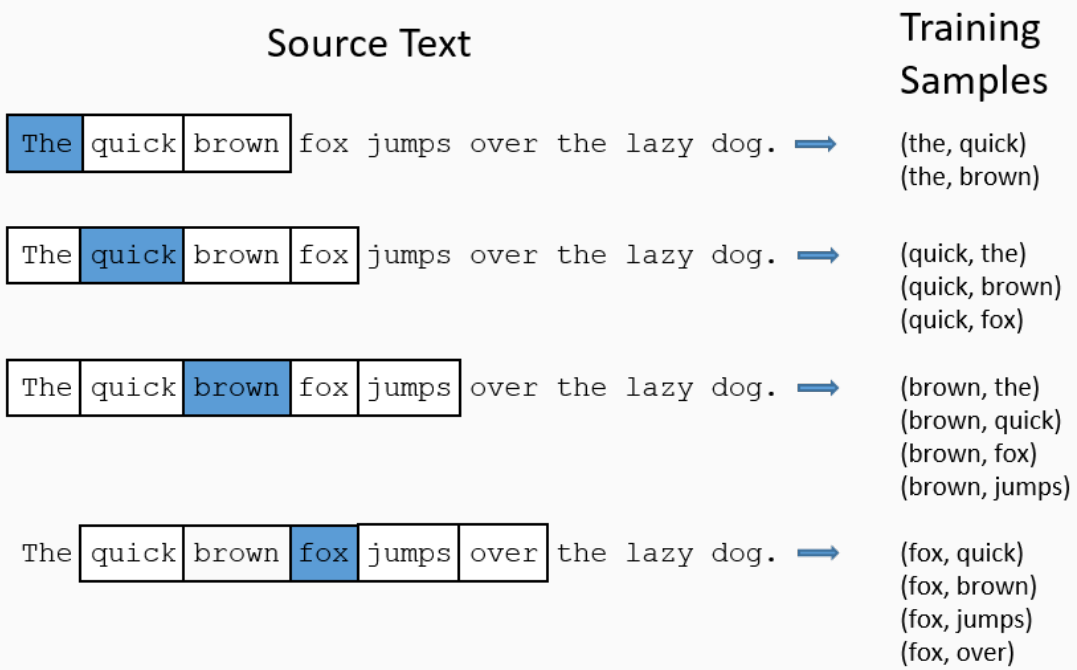


Imagen tomada de McCormick. Word2Vec Tutorial – The Skip-Gram Model, 2016.

Preprocesamiento: palabras como vectores densos

- Encajes de palabra
- Word2Vec

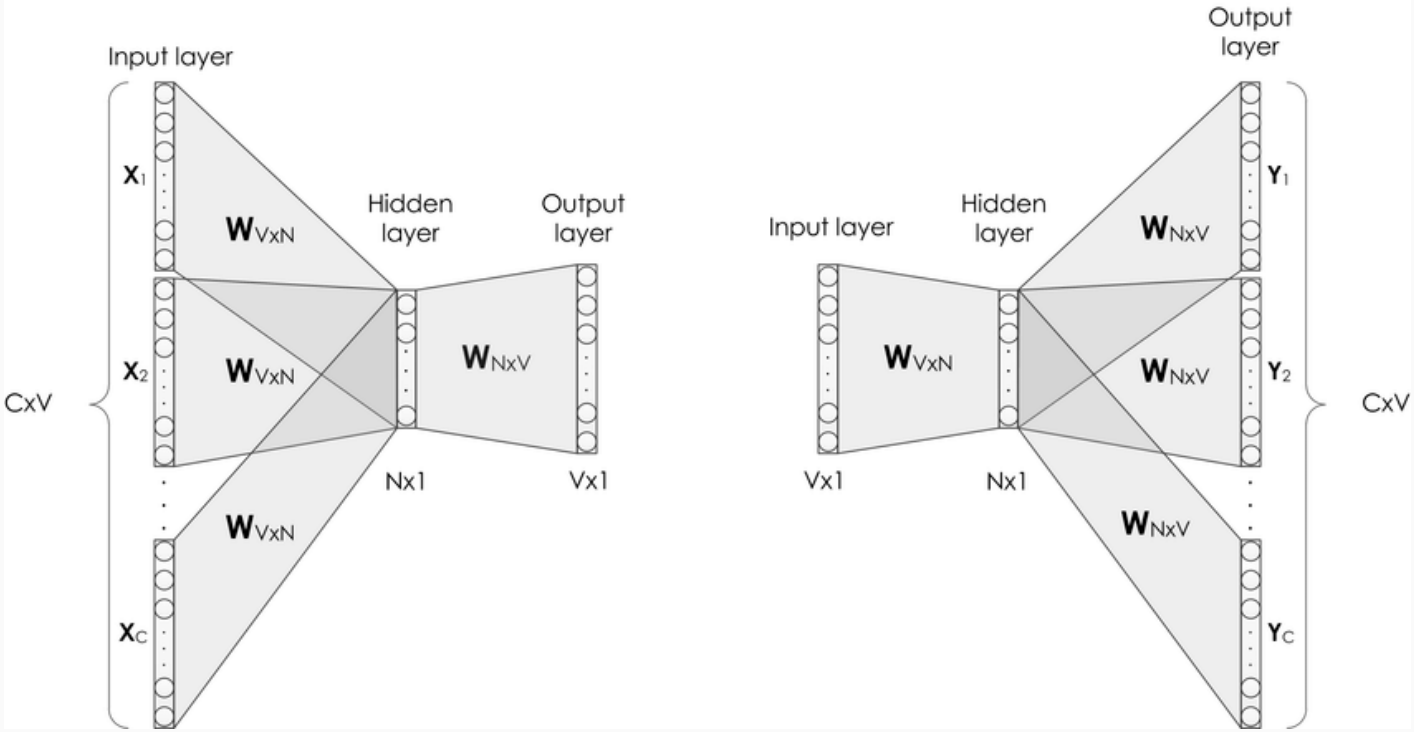


Imagen tomada de Tutubalina y Nikolenko. Demographic Prediction Based on User Reviews about Medications, 2017.

Preprocesamiento: video

- Preprocesamiento por marco
- Muestreo de marcos (por ej., uniforme o por movimiento)
- Flujo de movimiento

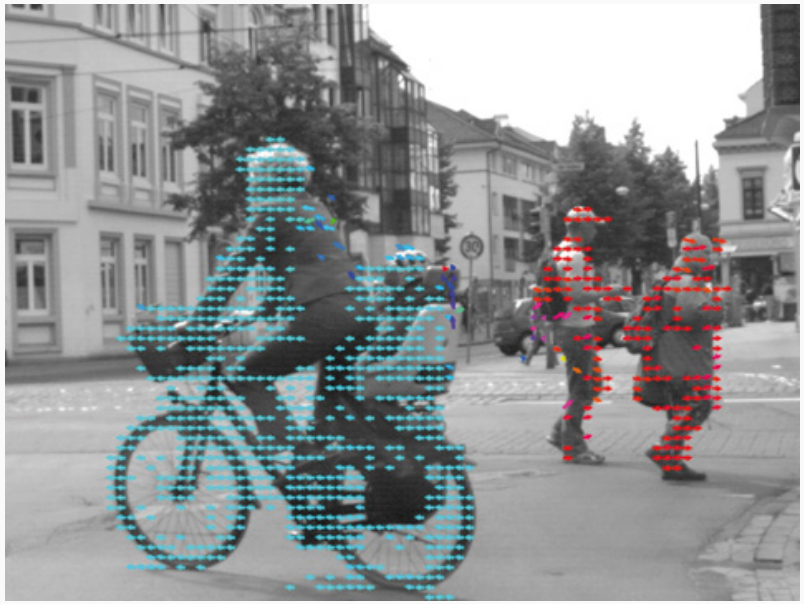
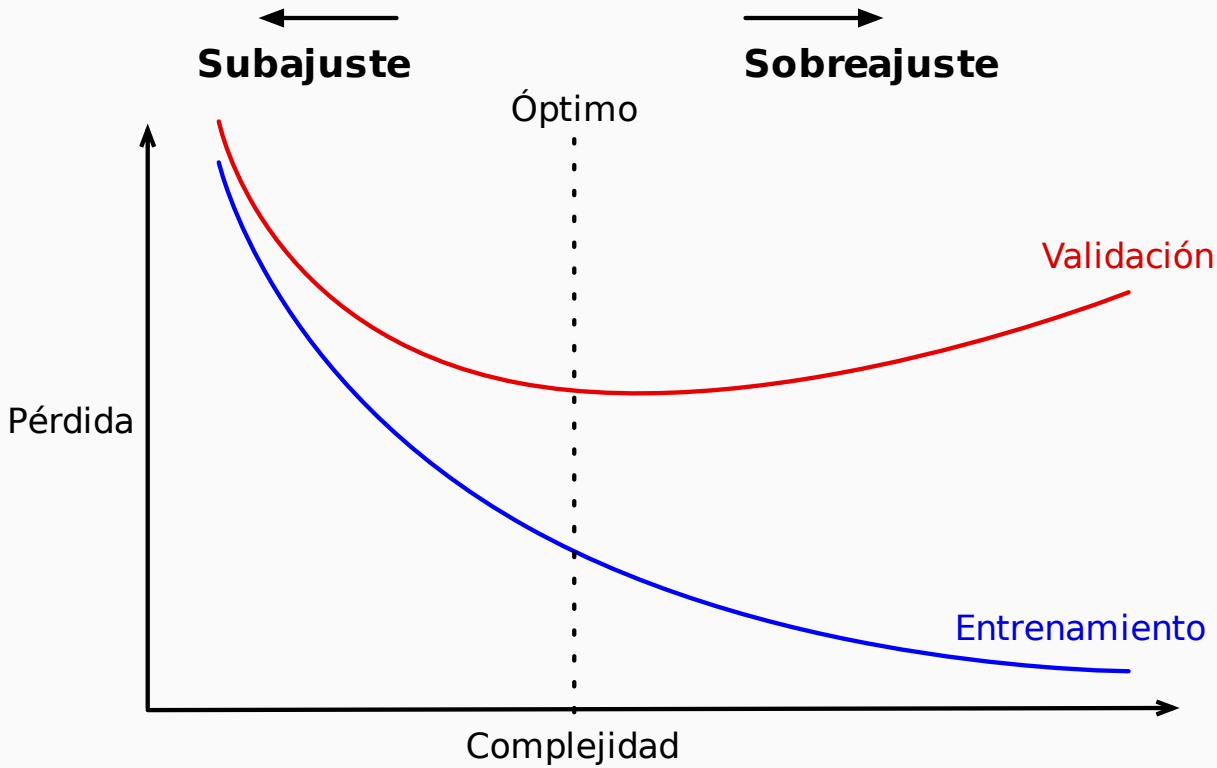


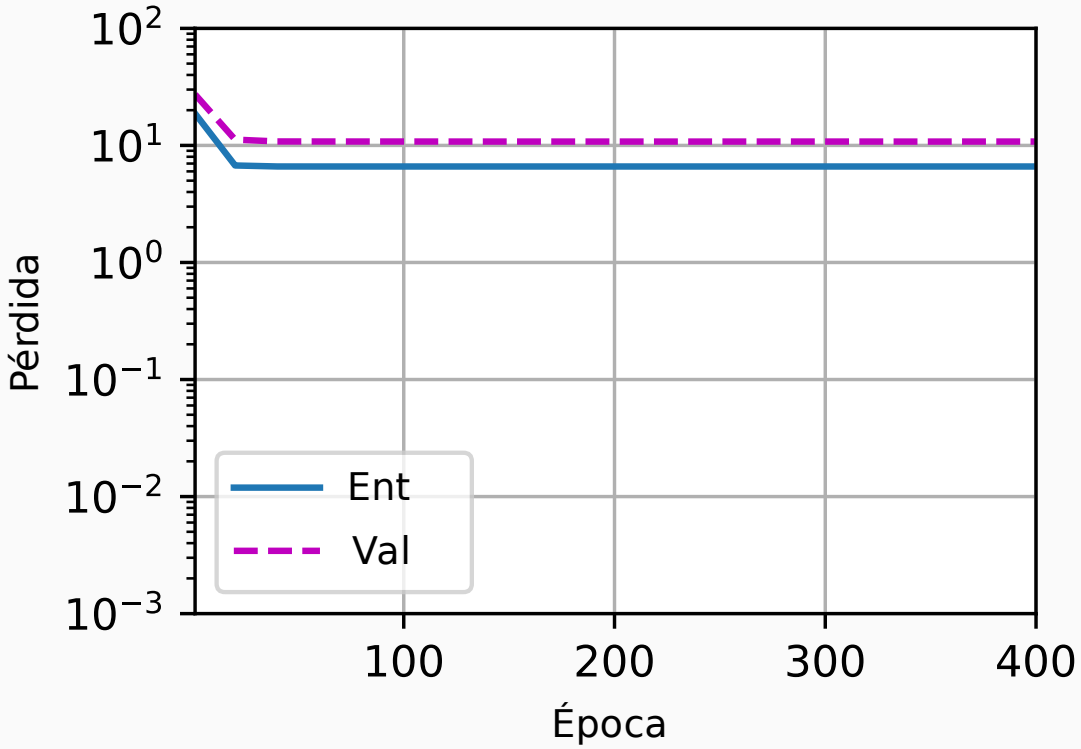
Imagen tomada de `https://www.commonlounge.com/discussion/1c2eaa85265f47a3a0a8ff1ac5fbce51`

Sobreajuste y cómo atacarlo

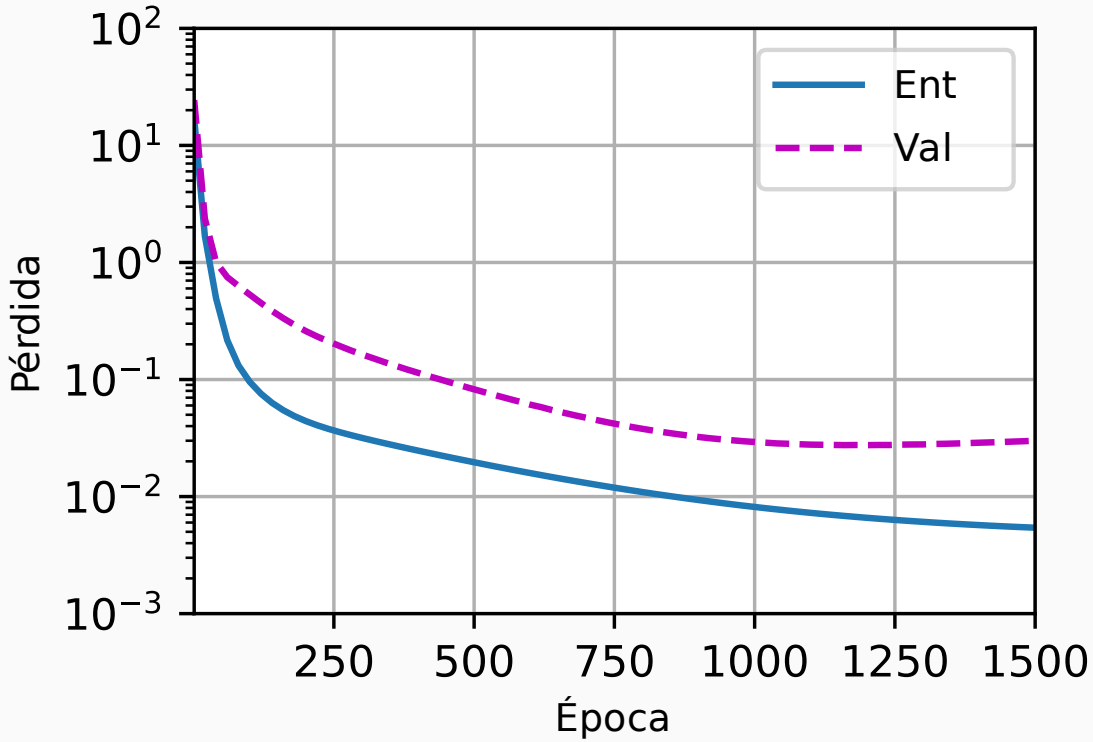
Sobreajuste y complejidad



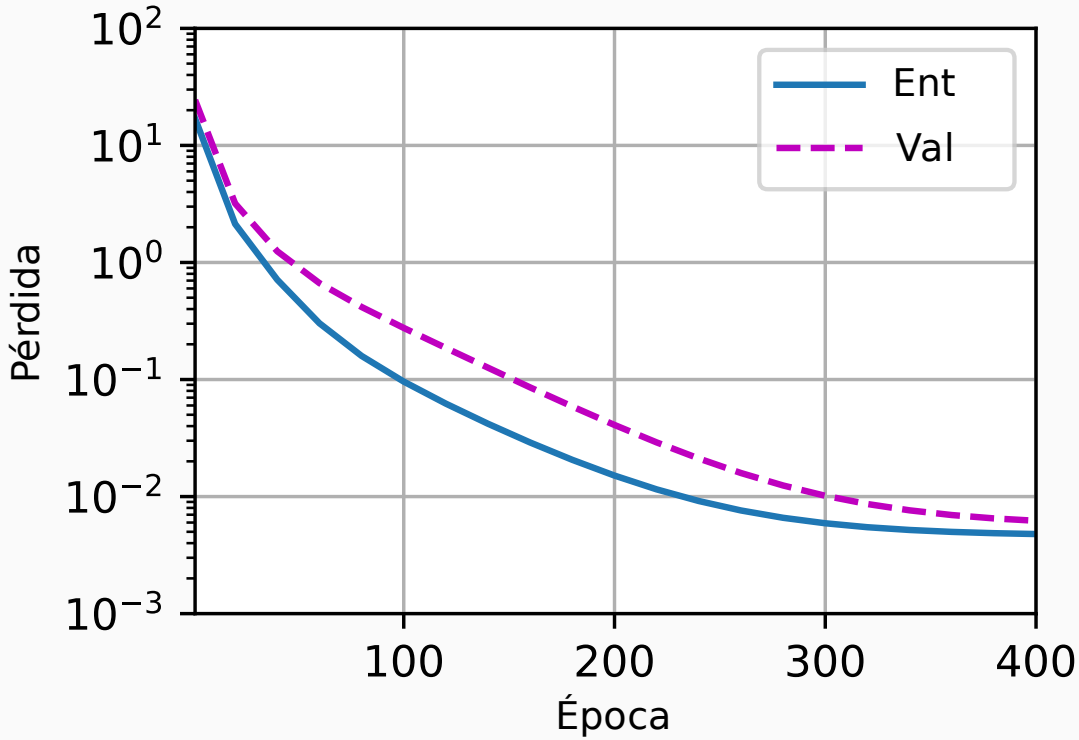
Subajuste



Sobreajuste



Ajuste normal



Regularización en redes neuronales

- Estrategias para disminuir sobreajuste en redes neuronales
  - Penalización de función de error (o función de pérdida)
  - Adición de ruido a entradas, salidas y/o parámetros
  - Ensambles
  - Paro temprano
  - Aprendizaje de múltiples tareas
  - Dropout
  - Acrecentamiento de datos (*data augmentation*)
  - Normalización por lotes
  - Versiones estocásticas del descenso por gradiente y variantes

Penalizando pesos y sesgos con norma ℓ_1 y ℓ_2

- Norma ℓ_1

$$\hat{\mathcal{L}}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_1$$

- Norma ℓ_2

$$\hat{\mathcal{L}}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$$

Paro temprano

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

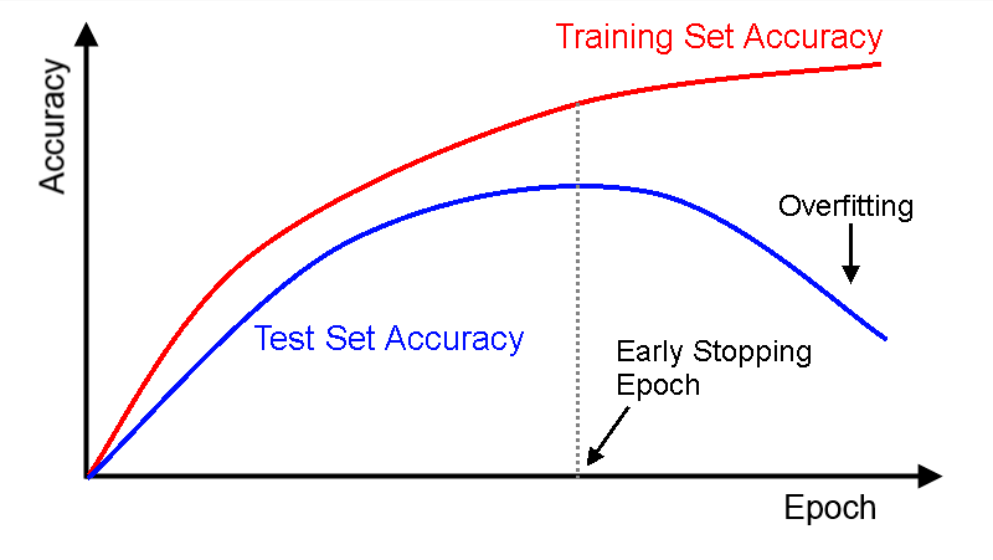


Imagen tomada de `https://deeplearning4j.org/earlystopping`

Aprendizaje de múltiples tareas

- Tener una representación genérica compartida entre múltiples tareas relacionadas

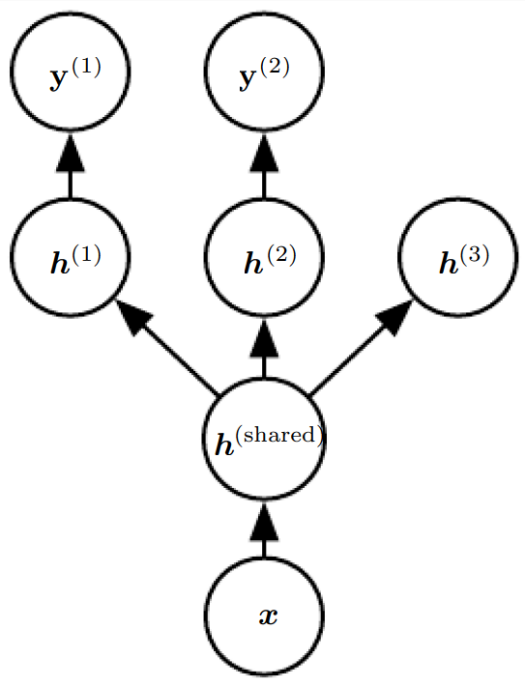


Imagen tomada de Goodfellow et al. Deep Learning, 2016

Dropout (desactivación)

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

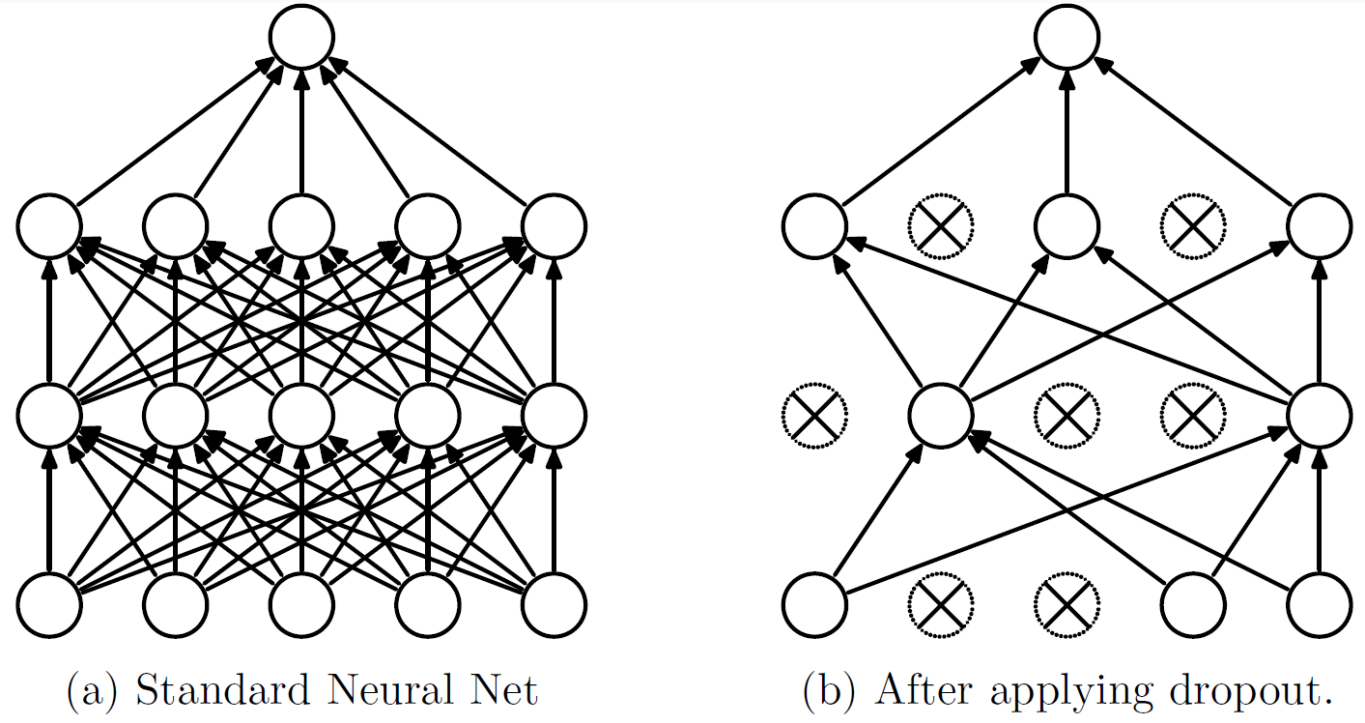


Imagen tomada de Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014

[1] Probabilidad es típicamente 0.5

Dropout en entrenamiento

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

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

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

$$r_j \sim Bernoulli(P)$$
$$\widetilde{\mathbf{y}}^{\{\ell\}} = \mathbf{r}^{\{\ell\}} * \mathbf{y}^{\{\ell\}}$$

- P es un hiperparámetro que indica la probabilidad de que una neurona se mantenga activa

Dropout como ensamble

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

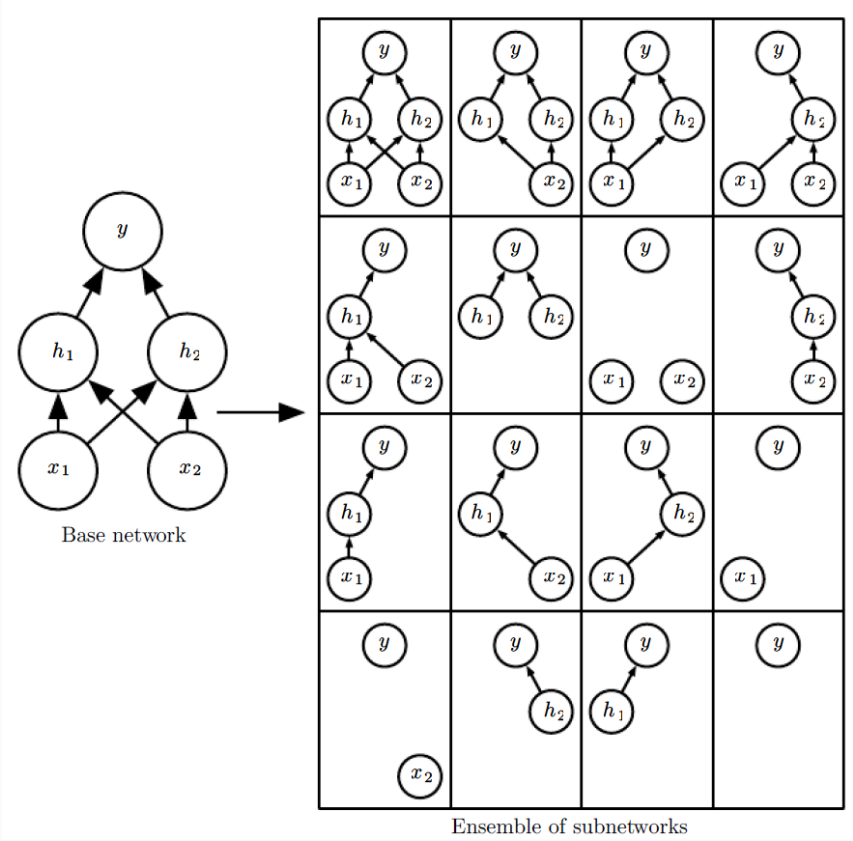


Imagen tomada de Goodfellow et al. Deep Learning, 2016

Dropout en inferencia

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

$$\boldsymbol{\theta}_{\text{inferencia}} = P \cdot \boldsymbol{\theta}$$

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

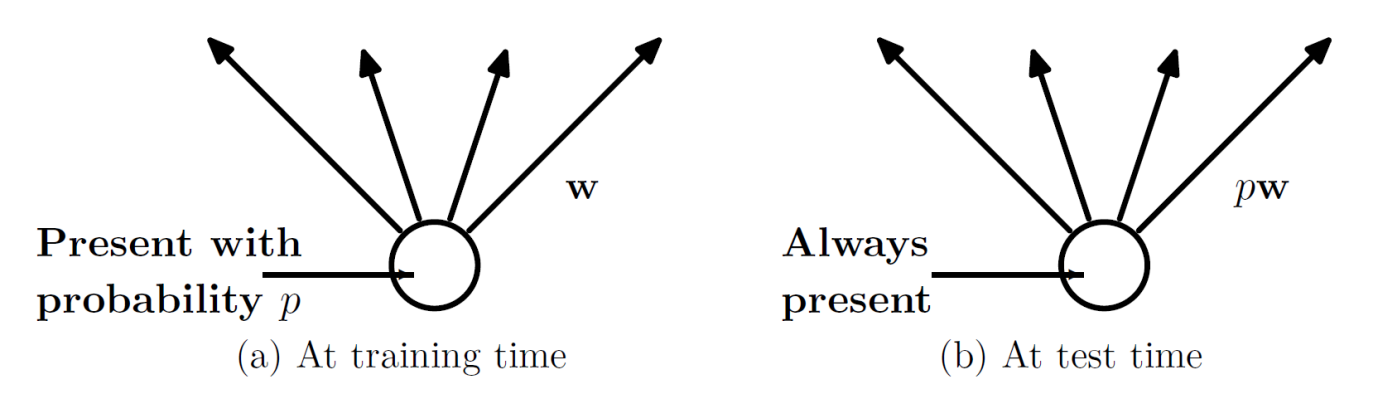
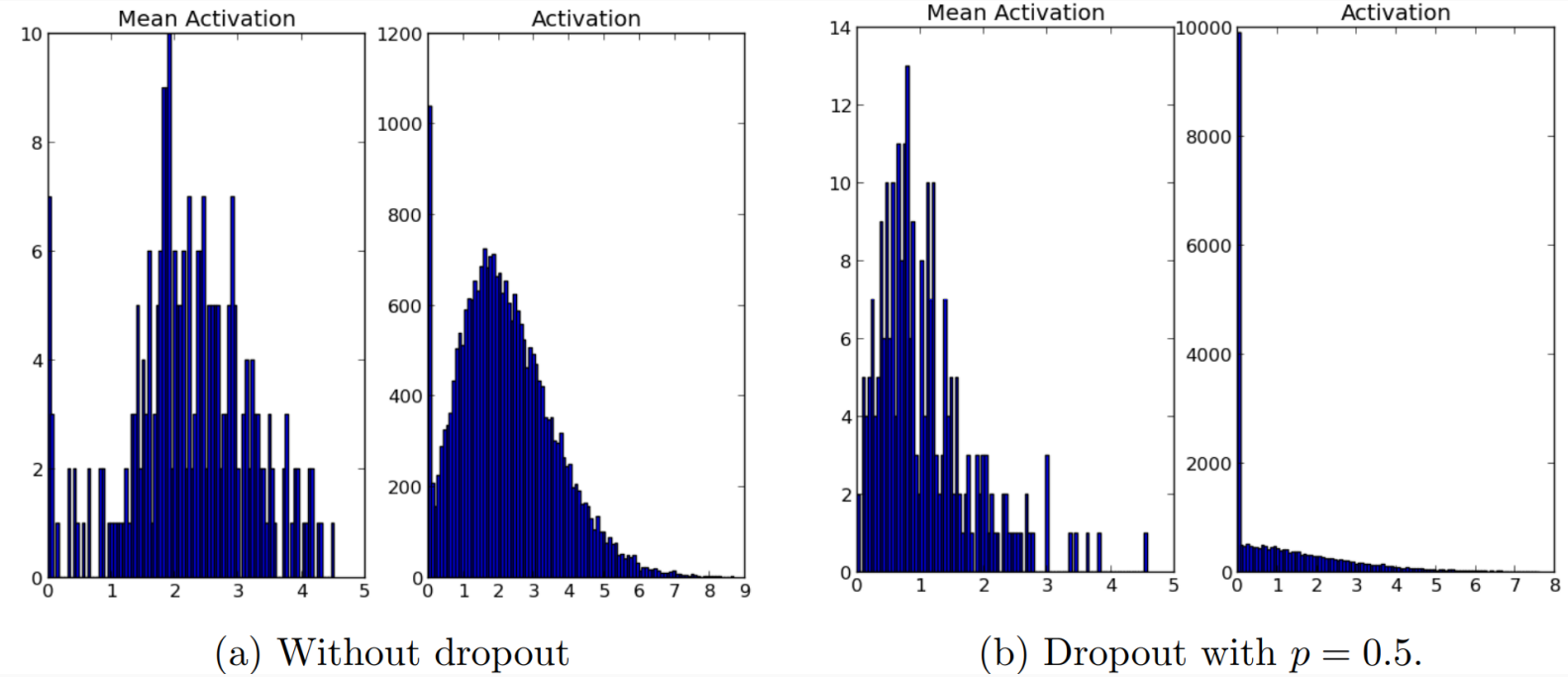


Imagen tomada de Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014

Activaciones con Dropout



(a) Without dropout

(b) Dropout with $p = 0.5$.

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

Acrecentamiento de imágenes: transformaciones

- Traslación

$$\begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

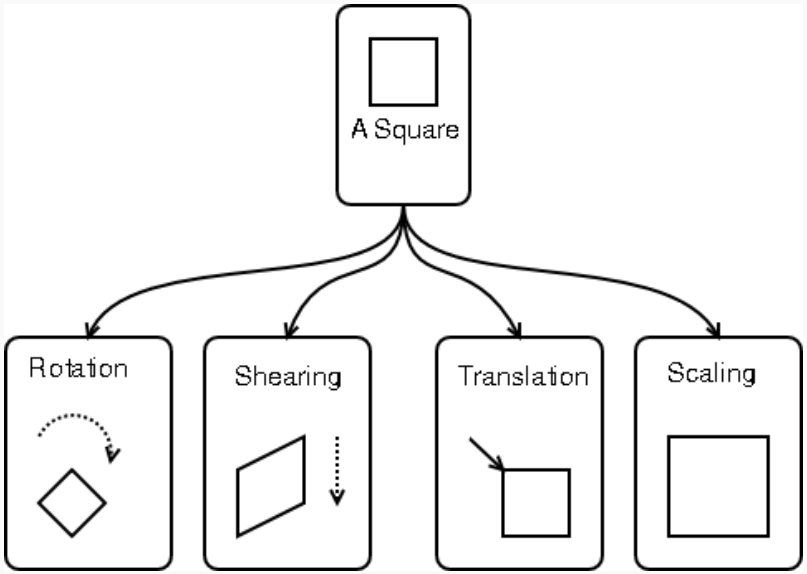


Imagen tomada de `https://people.gnome.org/~mathieu/libart/libart-affine-transformation-matrices.html`

Acrecentamiento de imágenes: transformaciones

- Traslación
- Rescalado

$$\begin{bmatrix} v_x & 0 & 0 \\ 0 & v_y & 0 \\ 0 & 0 & v_z \end{bmatrix}$$

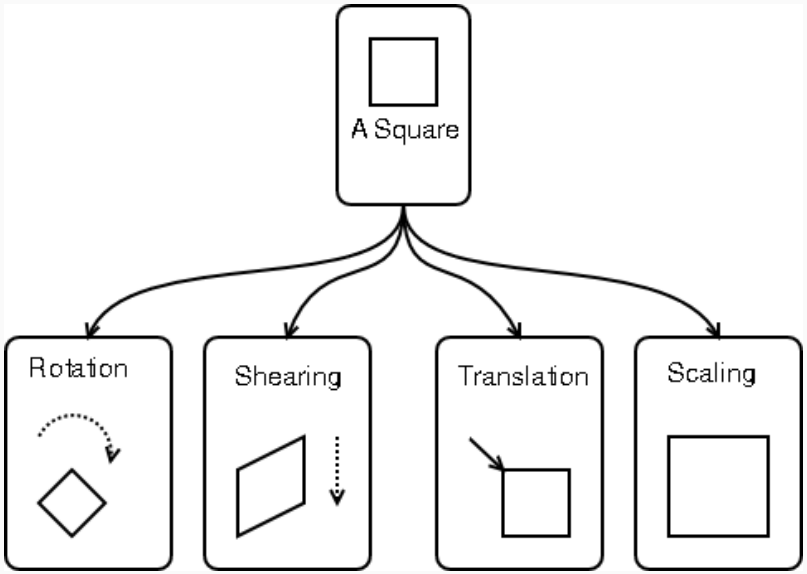


Imagen tomada de `https://people.gnome.org/~mathieu/libart/libart-affine-transformation-matrices.html`

Acrecentamiento de imágenes: transformaciones

- Traslación
- Rescalado
- Giro

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

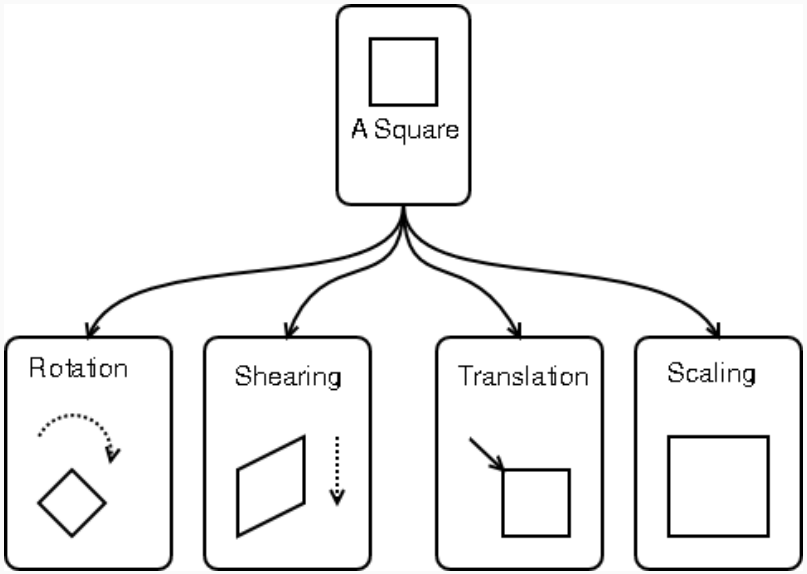


Imagen tomada de `https://people.gnome.org/~mathieu/libart/libart-affine-transformation-matrices.html`

Acrecentamiento de imágenes: transformaciones

- Traslación
- Rescalado
- Giro
- Rotación

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

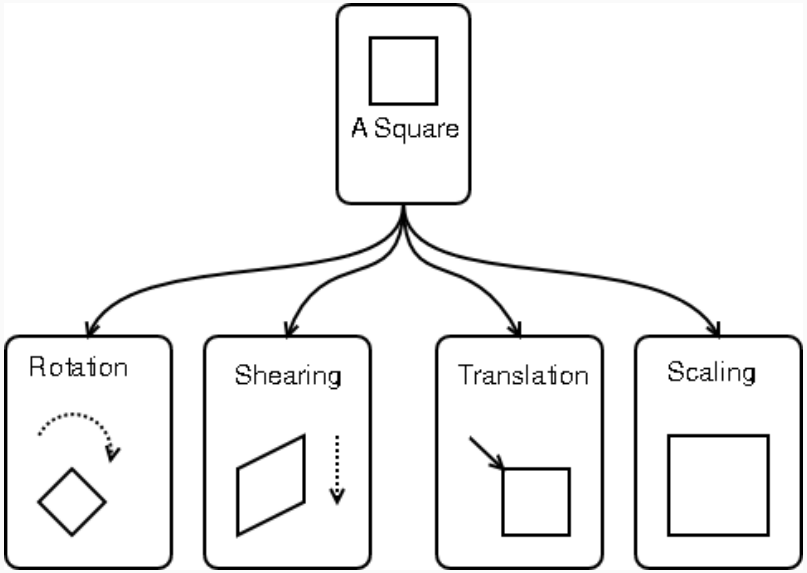


Imagen tomada de `https://people.gnome.org/~mathieu/libart/libart-affine-transformation-matrices.html`

Acrecentamiento de imágenes: transformaciones

- Traslación
- Rescalado
- Giro
- Rotación
- Deformación de corte

$$\begin{bmatrix} 1 & c_x = 0.5 & 0 \\ c_y = 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

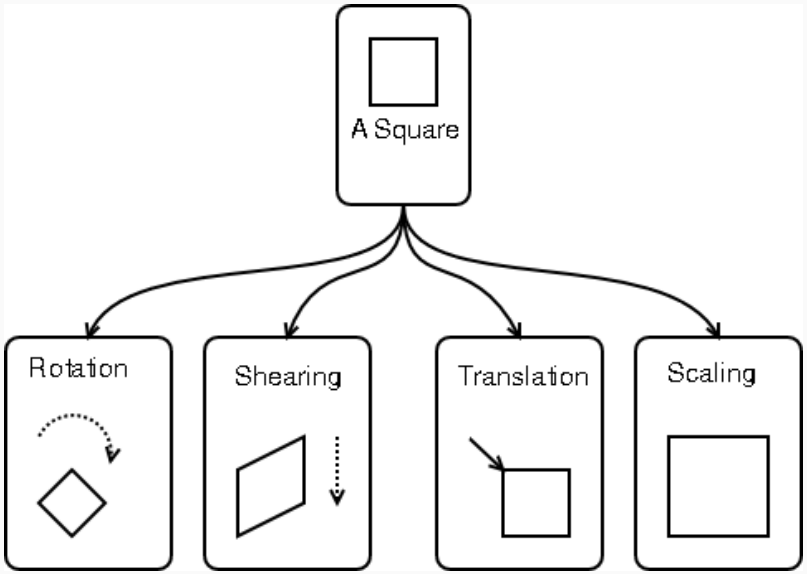


Imagen tomada de `https://people.gnome.org/~mathieu/libart/libart-affine-transformation-matrices.html`

Acrecentamiento de imágenes: transformaciones

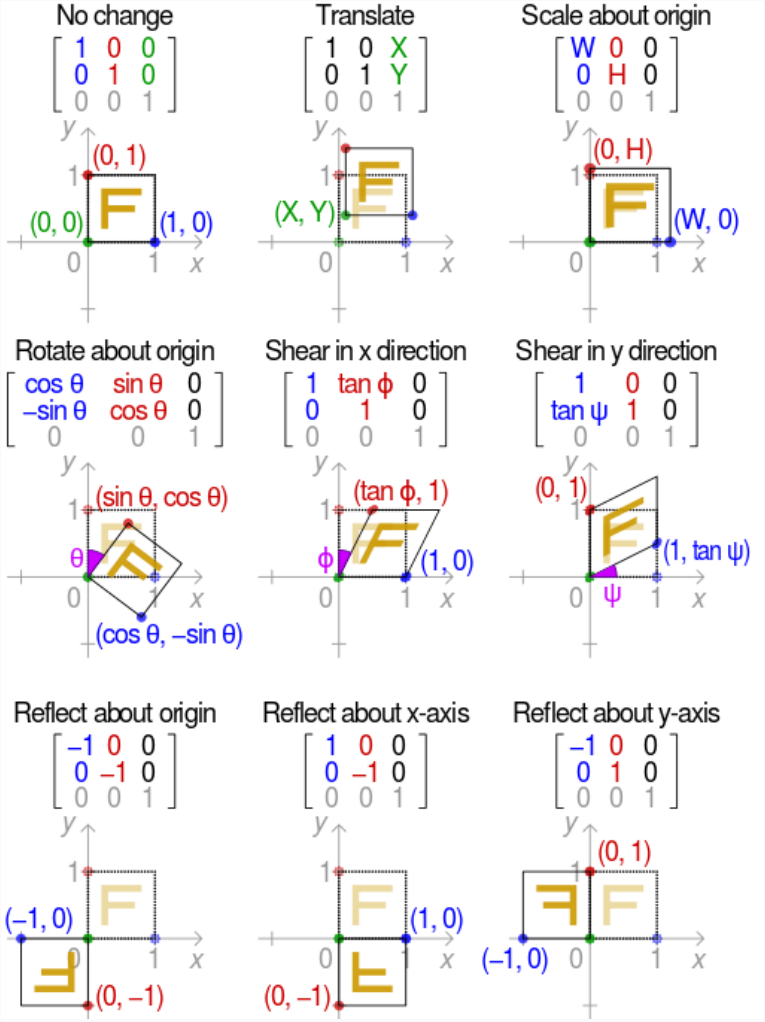


Imagen del usuario de Wikipedia Cmglee (entrada Affine transformation). CC BY-SA 3.0

Acrecentamiento de imágenes: ejemplos

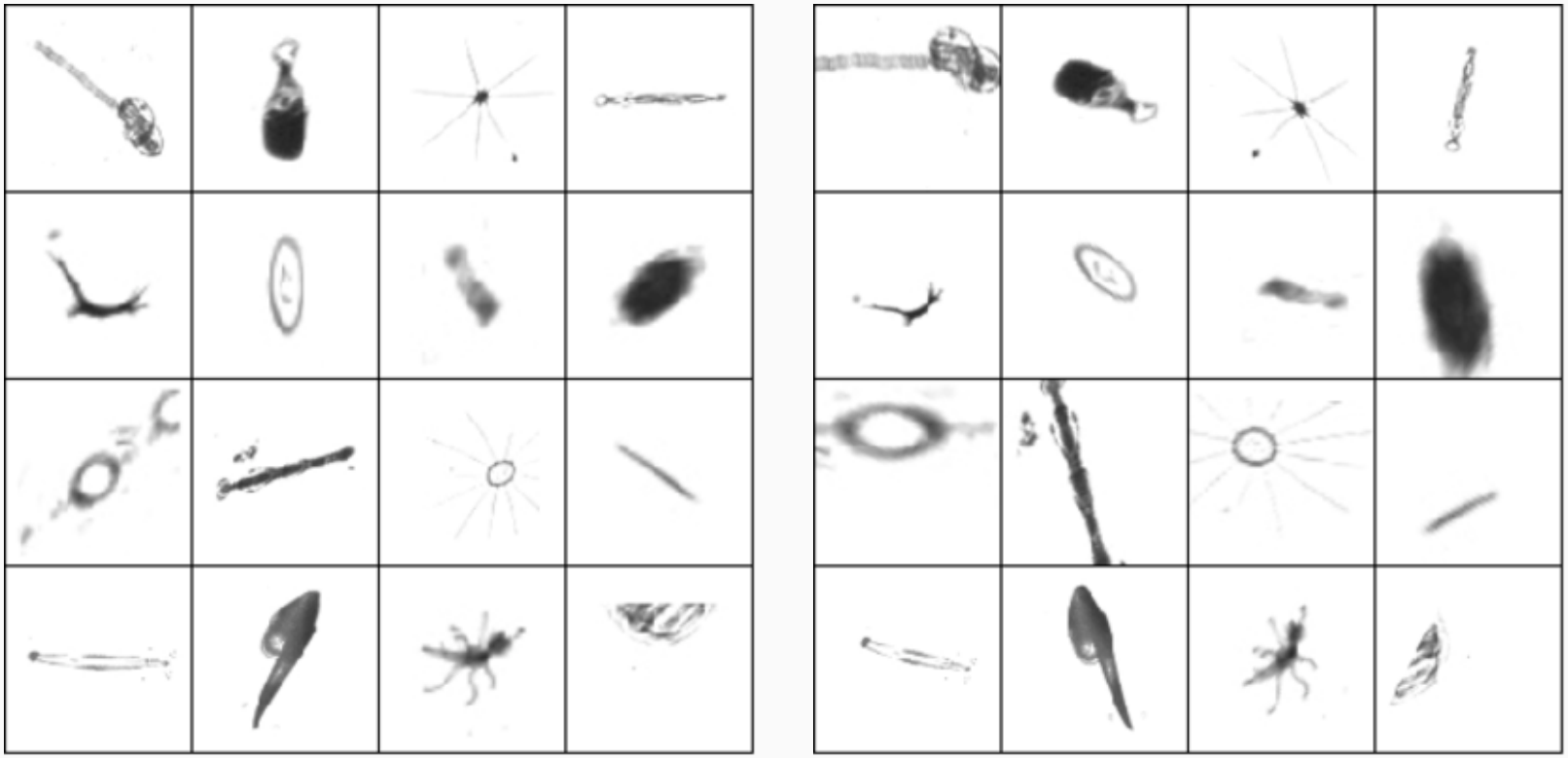


Imagen tomada de Dieleman. Classifying plankton with deep neural networks, 2015.

Acrecentamiento de audio

- Ruido aditivo

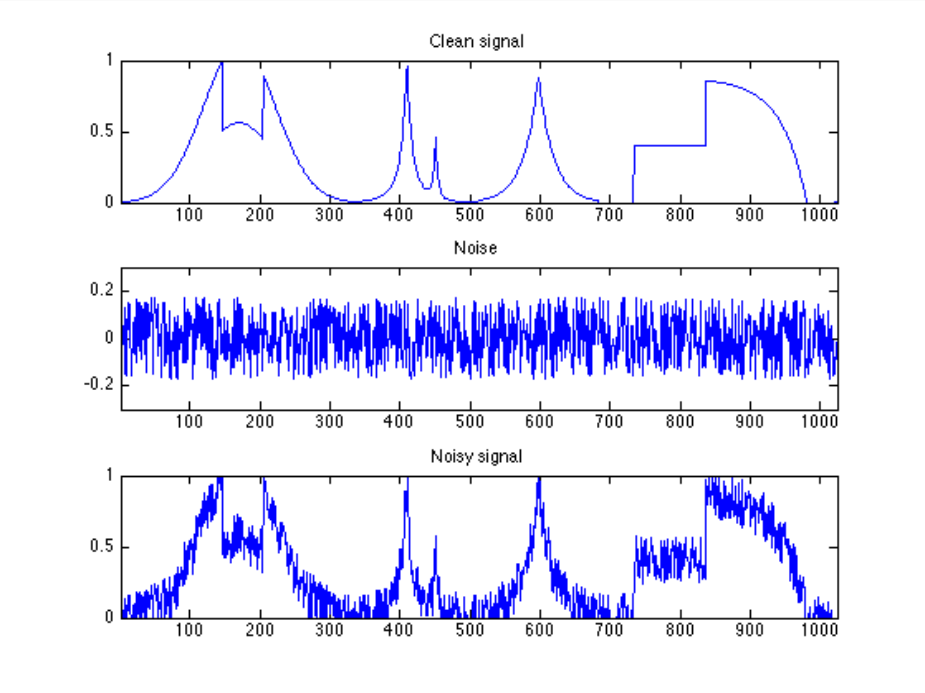


Imagen tomada de Peyre. Signal and Image Noise Models, 2008

Acrecentamiento de audio

- Ruido aditivo
- Enmascaramiento del espectograma

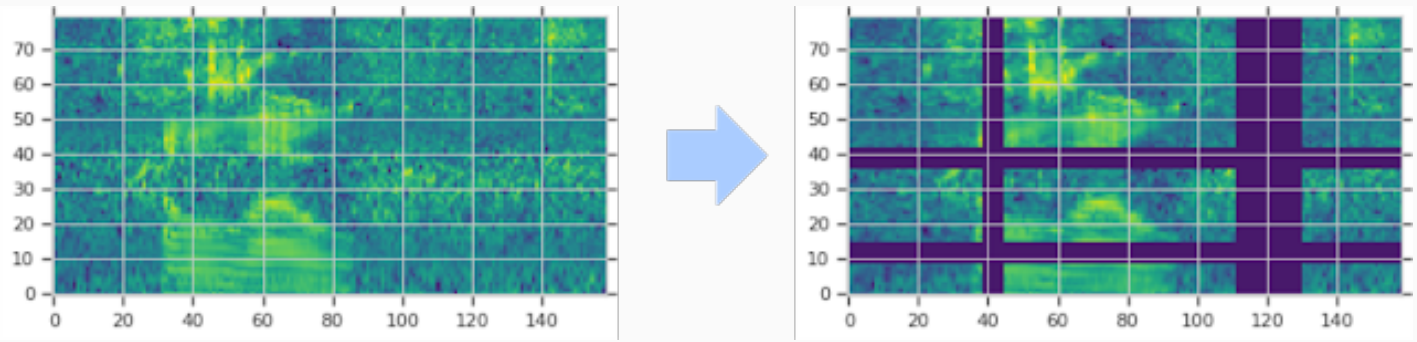


Imagen tomada de `https://ai.googleblog.com/2019/04/specaugment-new-data-augmentation.html`

Acrecentamiento en texto

- Símbolos
  - Insertar/cambiar/quitar símbolos aleatoriamente
  - Simular errores de teclado
  - Simular errores de OCR
- Palabra
  - Cambiar/quitar palabras aleatoriamente o con algún modelo de lenguaje o bolsa de palabras
  - Cambiar palabras por sinónimos
  - Cambiar palabras de acuerdo a errores de escritura

Desvanecimiento y explosión del gradiente

Explosión y desvanecimiento del gradiente

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

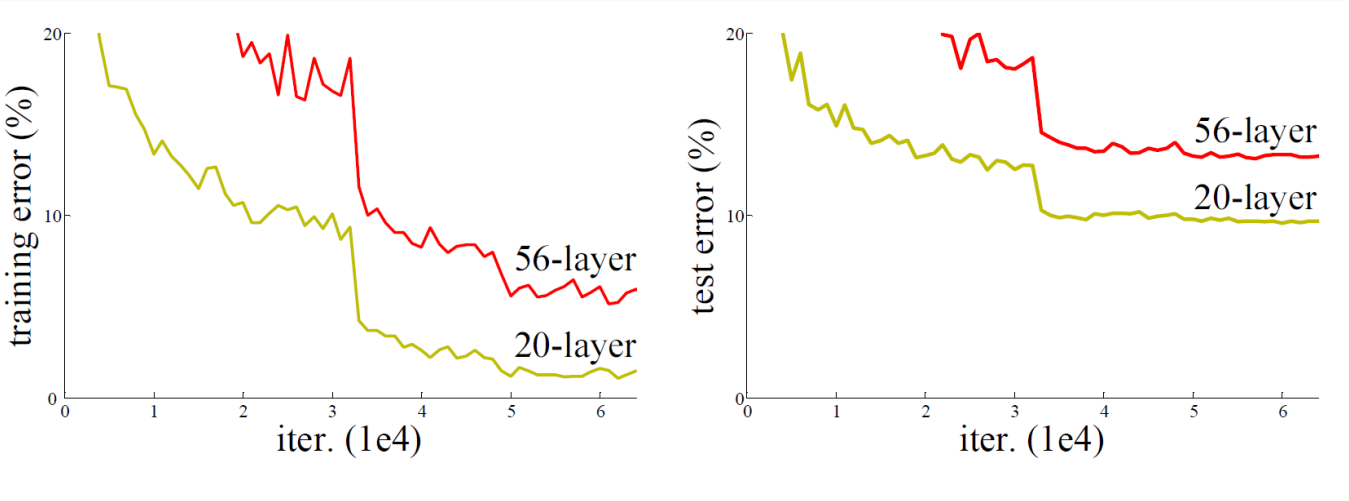
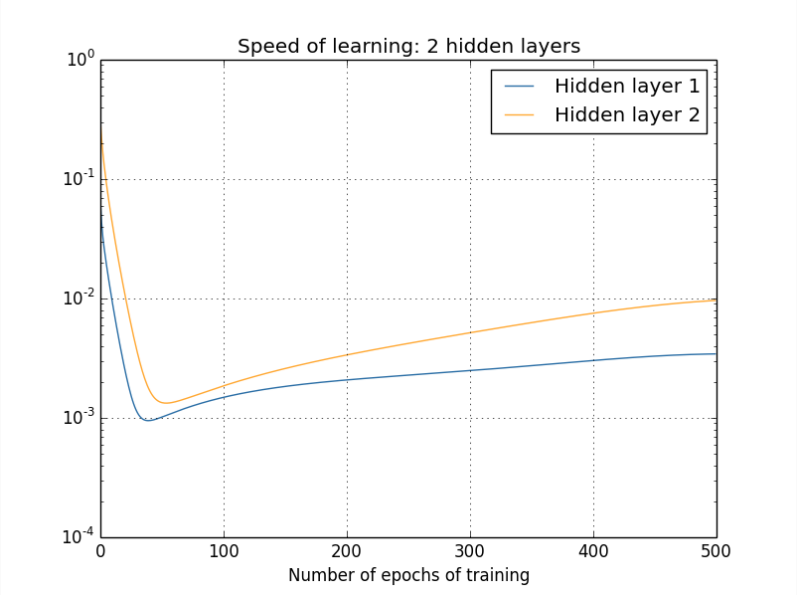


Imagen tomada de He et al. *Deep Residual Learning for Image Recognition*, 2015

Explosión y desvanecimiento del gradiente: 2 capas ocultas

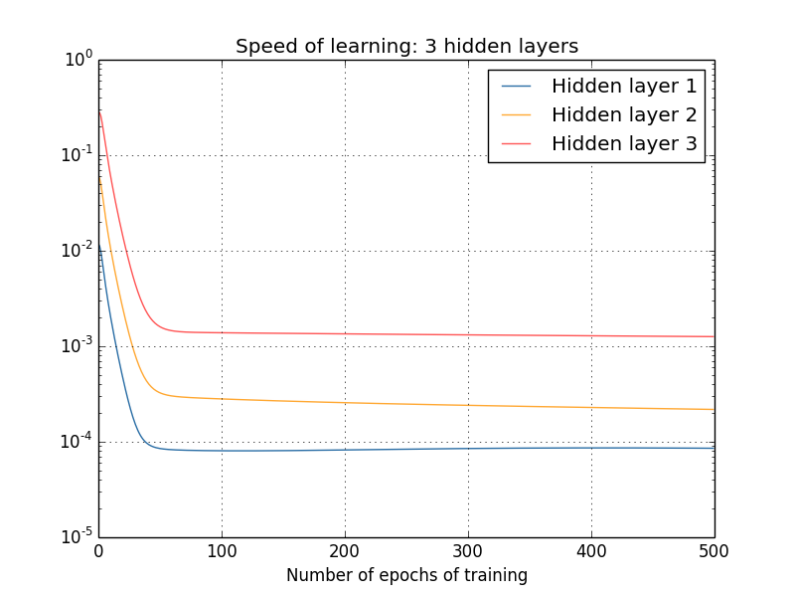
- Gradientes de primeras capas se vuelven muy pequeños si la red es muy profunda
- Muy lento actualizar pesos de estas capas



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

Explosión y desvanecimiento del gradiente: 3 capas ocultas

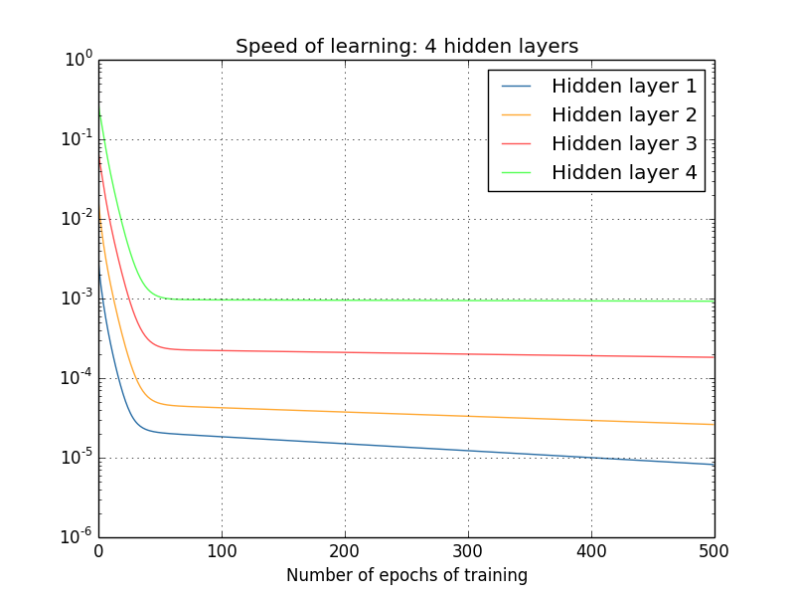
- Gradientes de primeras capas se vuelven muy pequeños si la red es muy profunda
- Muy lento actualizar pesos de estas capas



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

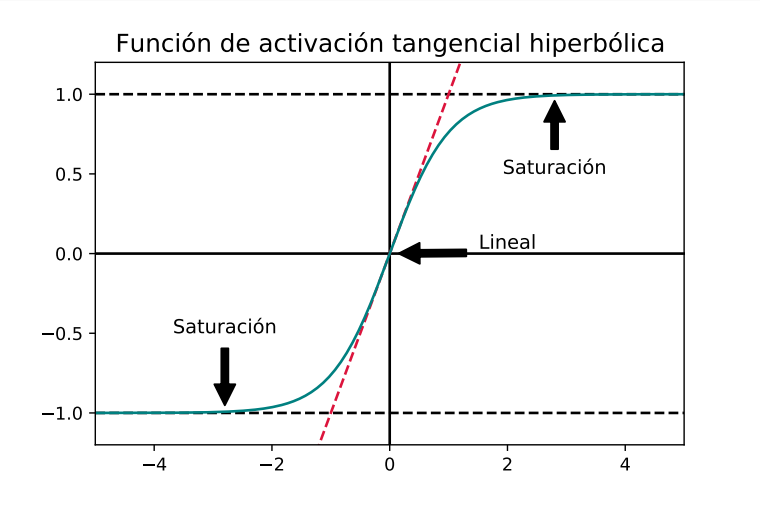
Gradient clipping

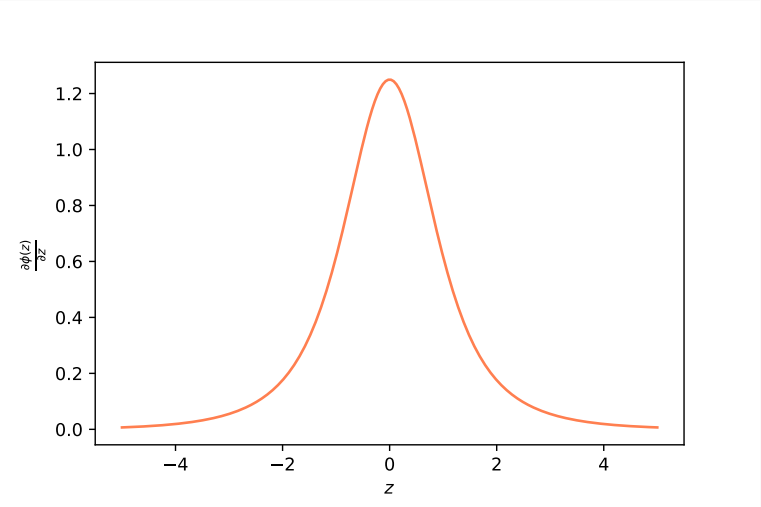
- Estrategia para evitar la explosión del gradiente
- La idea general es limitar la magnitud de los valores de los gradientes.
- Por ej.

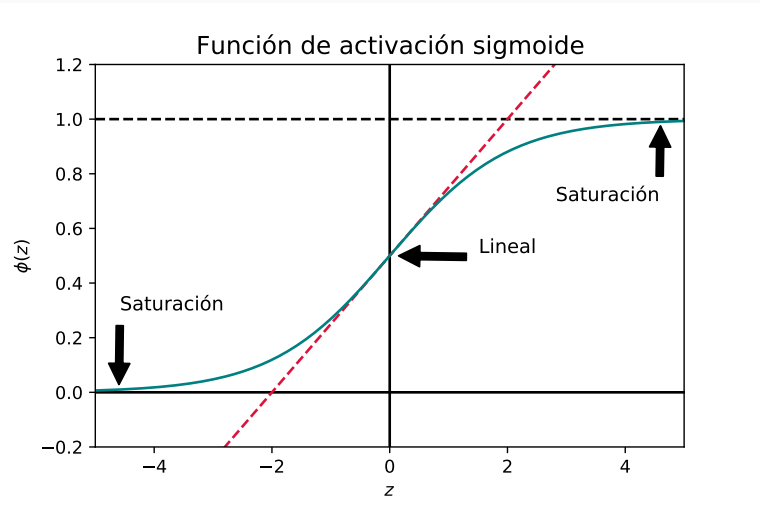
$$\text{Si } \|\nabla \mathcal{L}(\boldsymbol{\theta})\| > \eta, \text{entonces } \frac{\eta \cdot \nabla \mathcal{L}(\boldsymbol{\theta})}{\|\nabla \mathcal{L}(\boldsymbol{\theta})\|}$$

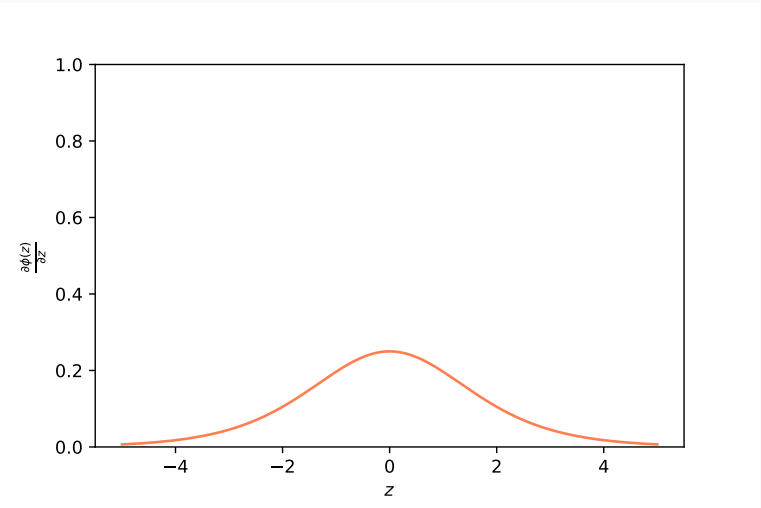
  donde η es un umbral de la norma.

Funciones de activación saturadas

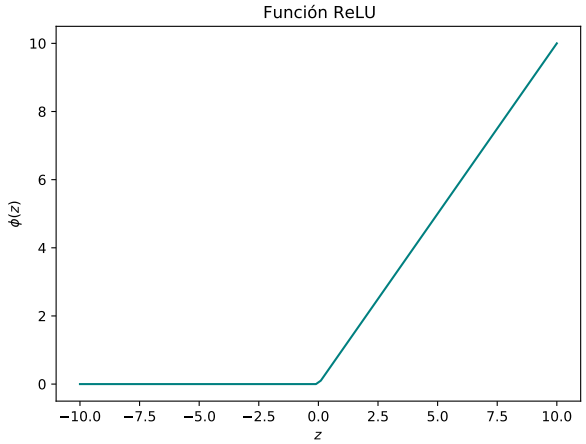


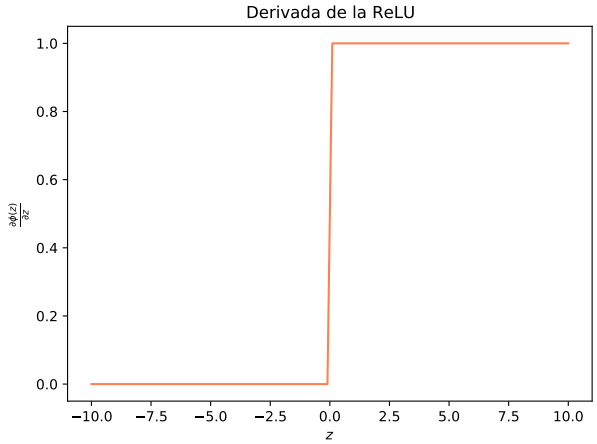


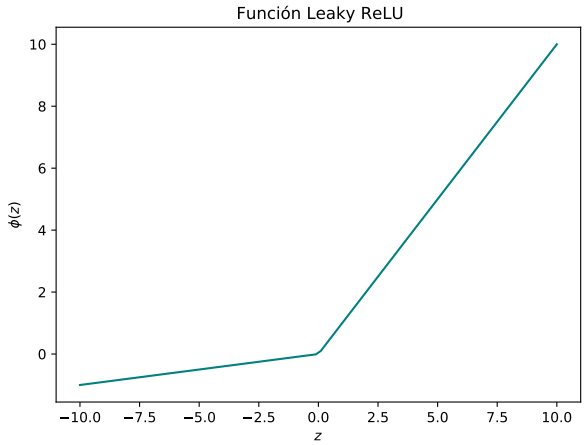


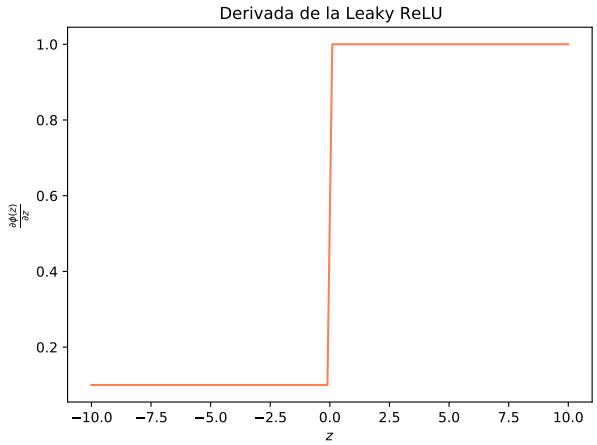


Funciones de activación no saturadas (1)

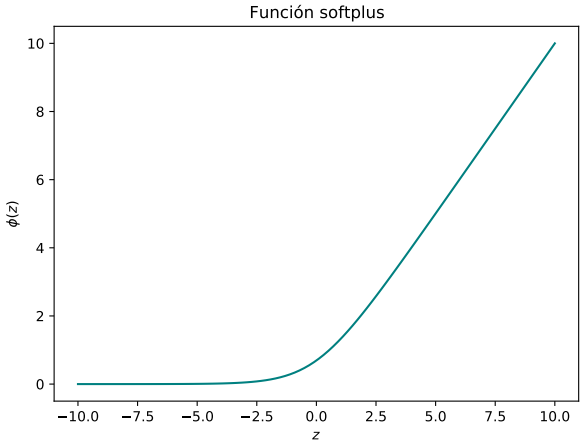


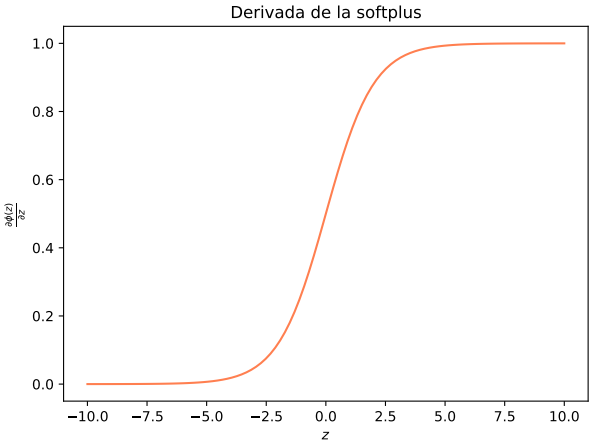


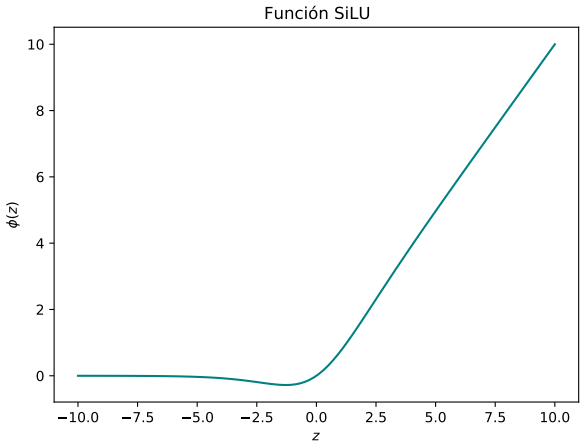


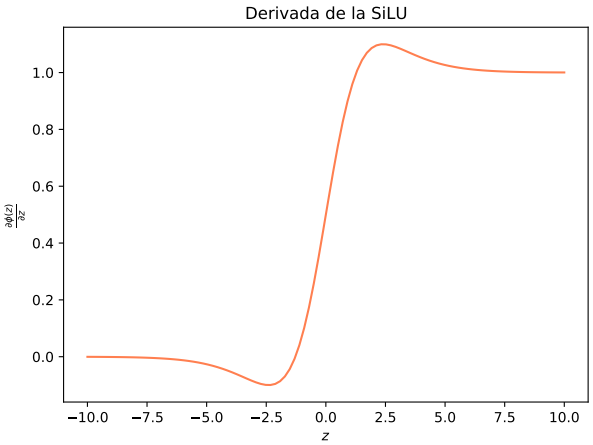


Funciones de activación no saturadas (2)









Explosión y desvanecimiento del gradiente: conexiones residua-les

- Agregando conexiones residuales en la arquitectura

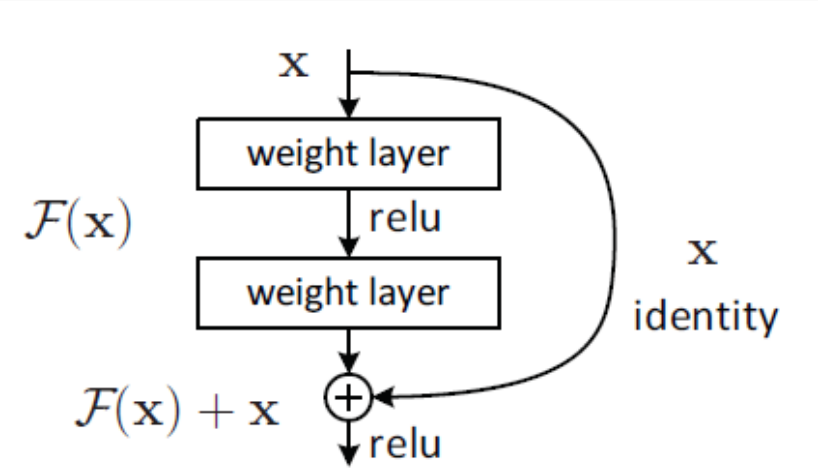


Imagen tomada de He et al. Deep Residual Learning for Image Recognition, 2015

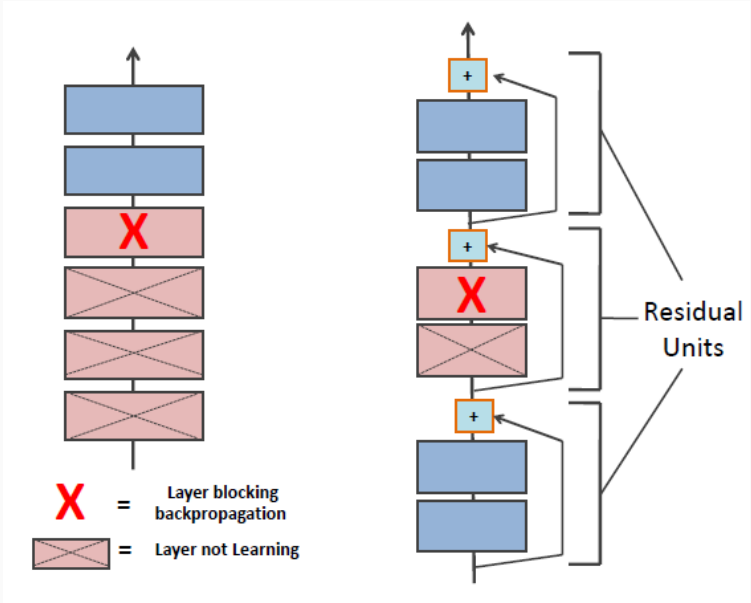


Imagen de Kevin Murphy, tomada de Probabilistic Machine Learning: An Introduction, 2021

Explosión y desvanecimiento del gradiente: inicialización (1)

- Números aleatorios de distribución gaussiana con media 0 y varianza 0.01.
  - Funciona en redes pequeñas
  - Para redes profundas activaciones tienden a volverse 0
- Números aleatorios de distribución gaussiana con media 0 y varianza 1
  - Genera saturación de las neuronas y gradientes se vuelven 0

Explosión y desvanecimiento del gradiente: inicialización (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)

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

  - Normal de He et al. (2015)

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

Normalización

El problema del desplazamiento covariable interno

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

- Hace más lento el aprendizaje

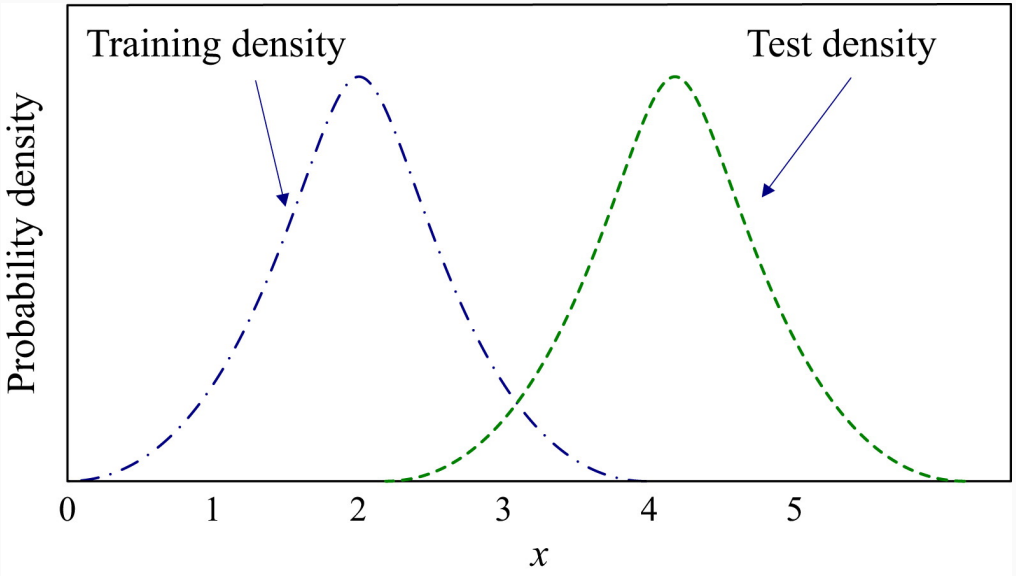


Imagen tomada de Raza et al. EWMA model based shift-detection methods for detecting covariate shifts in non-stationary environments, 2015

Normalización por lotes

- Converge más rápido si entradas tienen media 0, varianza 1 y no están correlacionadas
  1. Media y varianza del lote

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

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

  2. Normalización

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

  3. Escalado y desplazamiento

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

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

Entrenamiento e inferencia con normalización por lotes

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

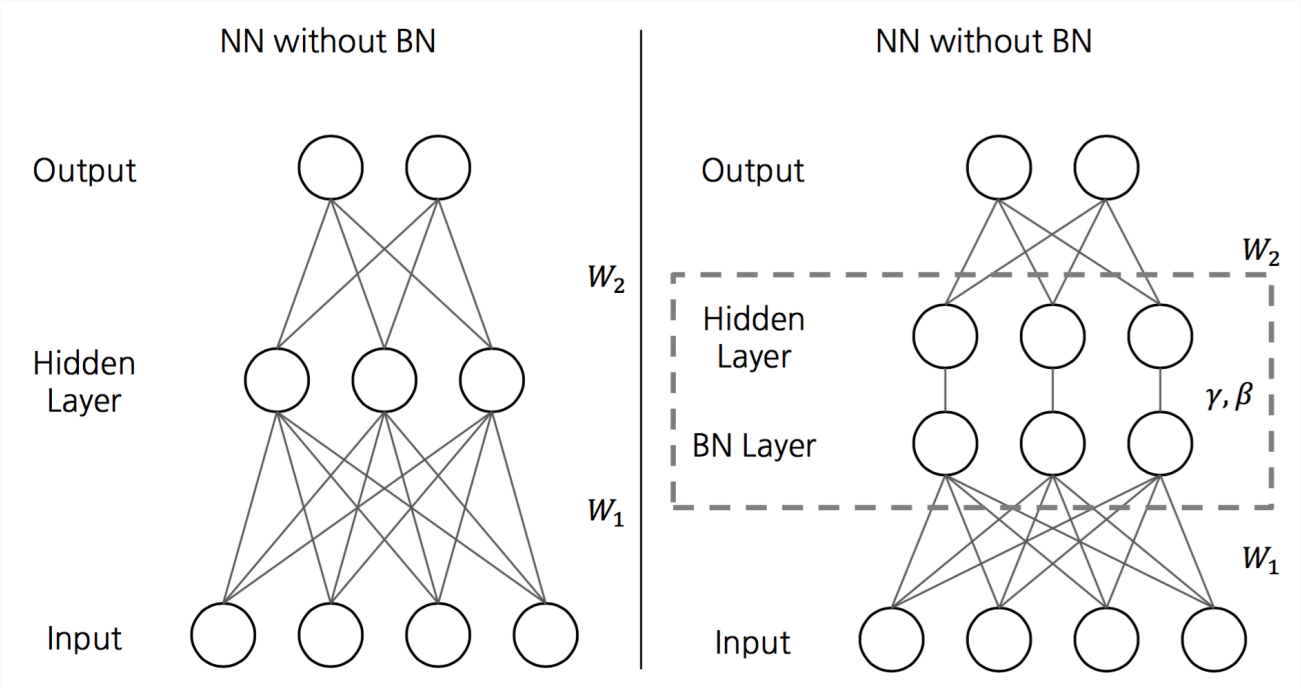


Imagen tomada de `https://wiki.tum.de/display/lfdv/Batch+Normalization`

Beneficios de normalización por lotes

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

Normalización por capas (1)

- 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.

Normalización por capas (2)

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

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

  donde γ y β son parámetros que se entrenan.

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

Optimizadores: descenso por gradiente y variantes

Problema de optimización

- Usualmente el aprendizaje en redes neuronales se formula como un problema de optimización

$$\hat{\boldsymbol{\theta}} \leftarrow \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$$

  donde $\mathcal{L}(\boldsymbol{\theta}) : \mathbb{R}^d \rightarrow \mathbb{R}$ es una función escalar, conocida como función de pérdida, costo o error, la cual depende de los d parámetros $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}\} \in \mathbb{R}^d$ de la red

- Al punto $\hat{\boldsymbol{\theta}}$ con el valor mínimo de la función de pérdida se le conoce como mínimo global

Conjuntos convexos

- Un conjunto $\mathcal{C}$ es convexo si, para cualquier par de puntos $x, y \in \mathcal{C}$, tenemos que

$$\lambda \cdot x + (1 - \lambda) \cdot y \in \mathcal{C}, \forall \lambda \in [0, 1]$$

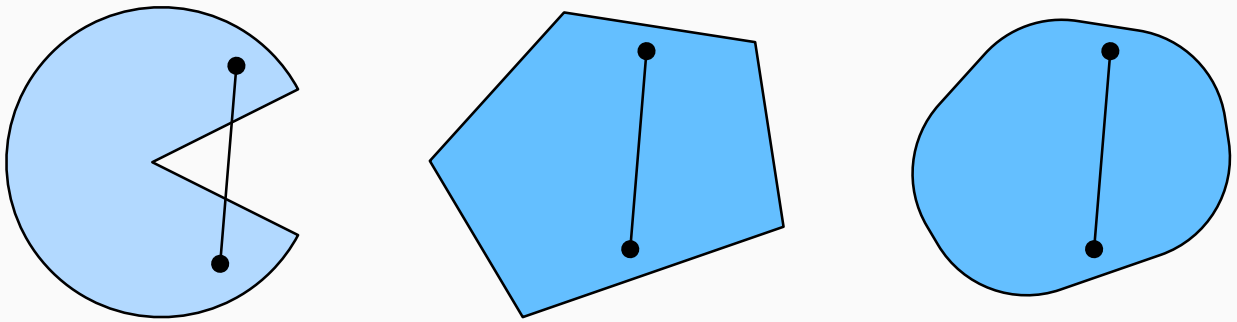


Figura de Zhang et al., tomada de Dive into Deep Learning, 2021.

Intersección de conjuntos convexos

- Si $\mathcal{C}_1$ y $\mathcal{C}_2$ son dos conjuntos convexos, entonces el conjunto $\mathcal{C}_1 \cap \mathcal{C}_2$ es también convexo.

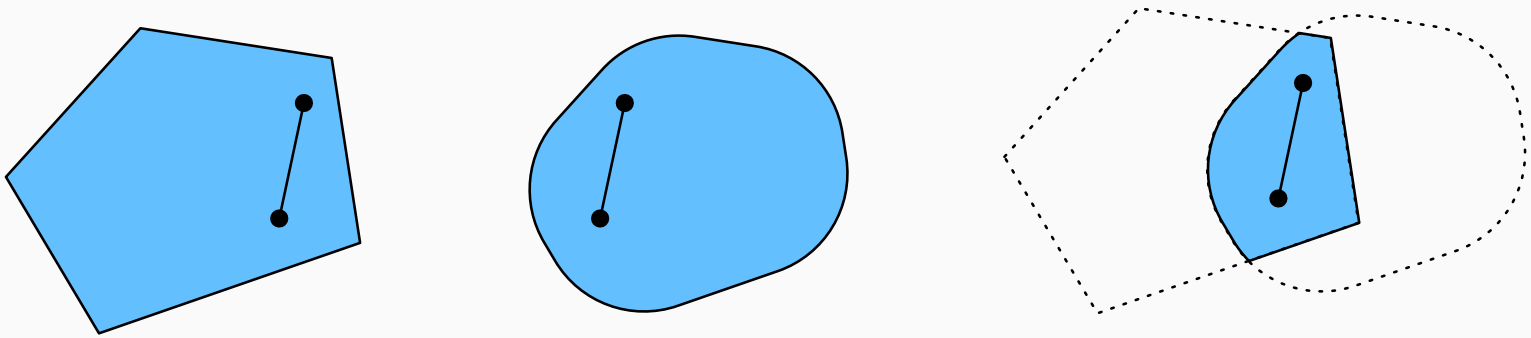


Figura de Zhang et al., tomada de Dive into Deep Learning, 2021.

Unión de conjuntos convexos

- Si $\mathcal{C}_1$ y $\mathcal{C}_2$ son dos conjuntos convexos, entonces el conjunto $\mathcal{C}_1 \cup \mathcal{C}_2$ es no convexo.

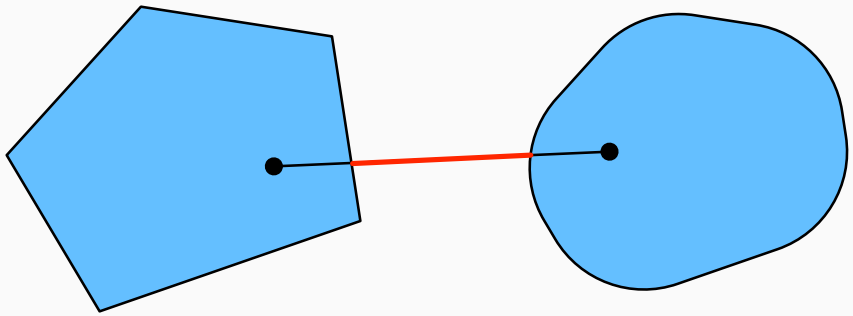


Figura de Zhang et al., tomada de Dive into Deep Learning, 2021.

Funciones convexas

- Una función $f : \mathcal{C} \to \mathbb{R}$ es convexa si su epigrafo es un conjunto convexo.

- El epigrafo de una función f es el conjutno de puntos que se encuentra encima de ella

$$epi(f) = \{(x, \mu) : x \in \mathbb{R}^d, \mu \in \mathbb{R}, f(x) \leq \mu\} \subseteq \mathbb{R}^{d+1}$$

- Dado un conjunto convexo $\mathcal{C}$, una función es convexa si para cualquier par de puntos $x, y \in \mathcal{C}$

$$\lambda \cdot f(x) + (1 - \lambda) \cdot f(y) \geq f(\lambda \cdot x + (1 - \lambda) \cdot f(y))$$

Propiedades de funciones convexas

- Mínimos locales son mínimos globales

- Una función $f : \mathbb{R} \to \mathbb{R}$ doblemente diferenciable
  - Es convexa si $\frac{d^2 f(x)}{dx^2} \geq 0$
  - Es estrictamente convexa si $\frac{d^2 f(x)}{dx^2} > 0$
  - Es fuertemente convexa con parámetro m si $\frac{d^2 f(x)}{dx^2} \geq m > 0$

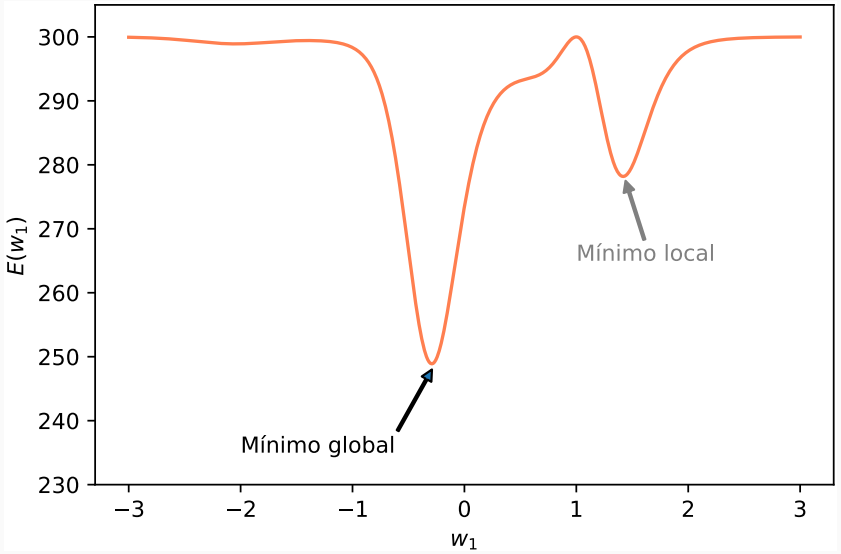
- Una función $f : \mathbb{R}^d \to \mathbb{R}$ doblemente diferenciable
  - Es convexa si el hessiano de f es semidefinido positivo
  - Es estrictamente convexa si el hessiano de f es definido positivo
  - Es fuertemente convexa con parámetro m si

$$(\nabla f(\mathbf{x}) - \nabla f(\mathbf{y}))^\top (\mathbf{x} - \mathbf{y}) \geq m \cdot \|\mathbf{x} - \mathbf{y}\|_2^2$$

Mínimos locales

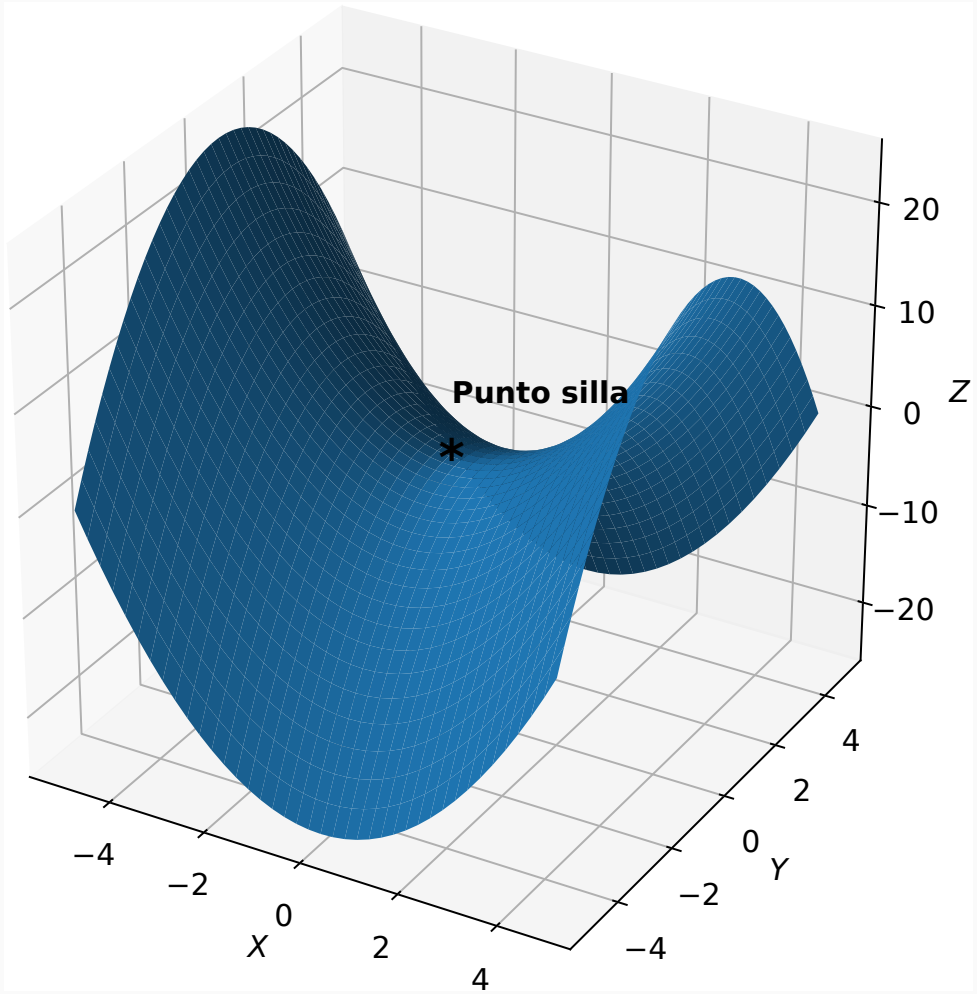
- Las funciones de pérdida en redes neuronales multicapa usualmente son no convexas y encontrar siempre el mínimo global es intratable

- En algunos casos se busca garantizar que al menos se encuentre un mínimo local, esto es, un punto $\boldsymbol{\theta}^*$ cuyo valor sea menor o igual que el de todos sus vecinos



Puntos silla

- Pueden existir varios puntos silla



Promedio móvil ponderado exponencialmente (PMPE)

- Definido por

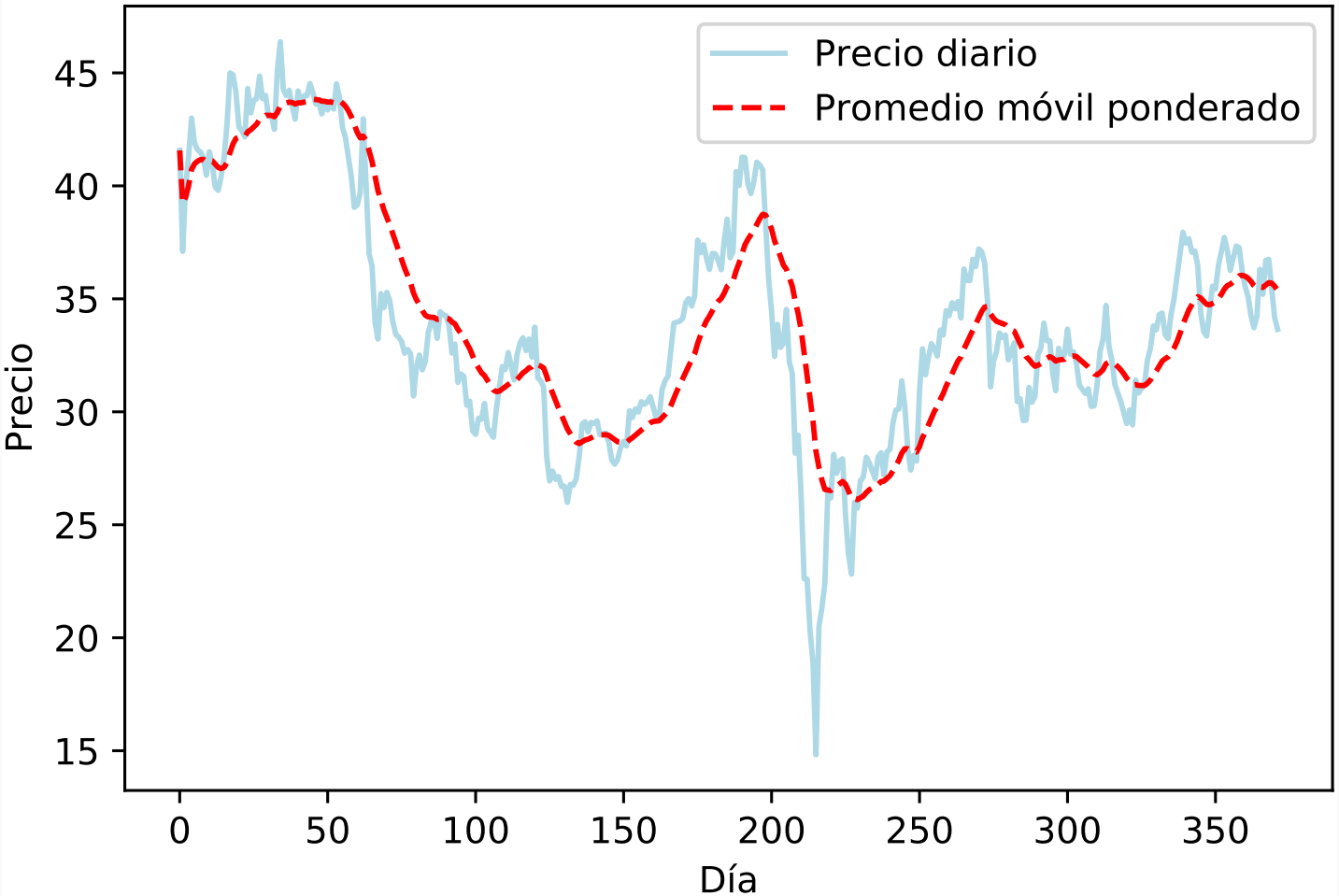
$$e^{[t+1]} = \beta \cdot e^{[t]} + (1 - \beta) \cdot s^{[t]}$$

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

- Expandiendo

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

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

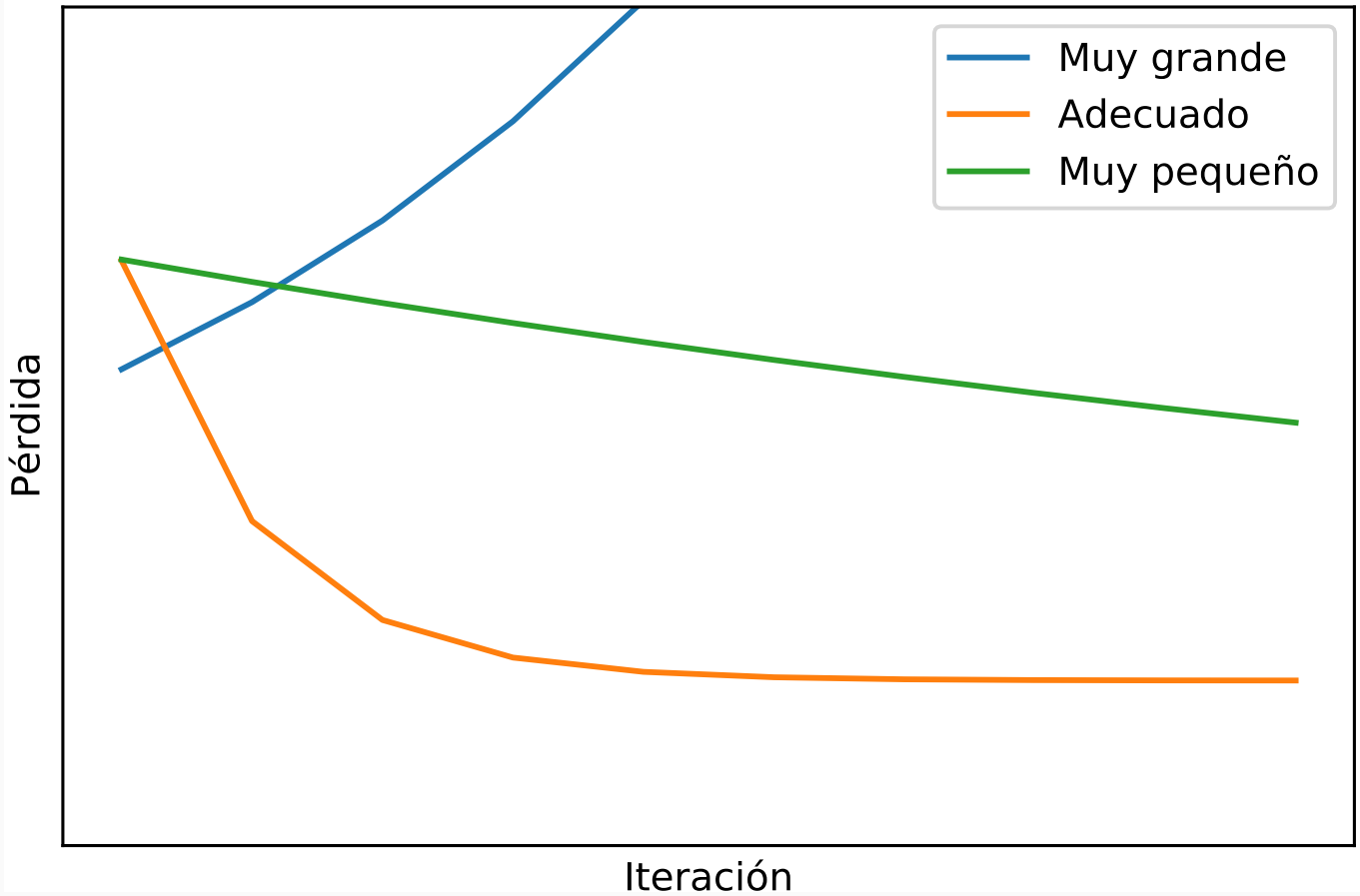
$$\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]} - \alpha \nabla \mathcal{L}(\boldsymbol{\theta}^{[t]})$$

  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

- Se va ajustando la tasa de aprendizaje con el tiempo
- Estrategias
  - Constante por periodos
  - Declive polinomial
  - Declive exponencial
  - Cíclico

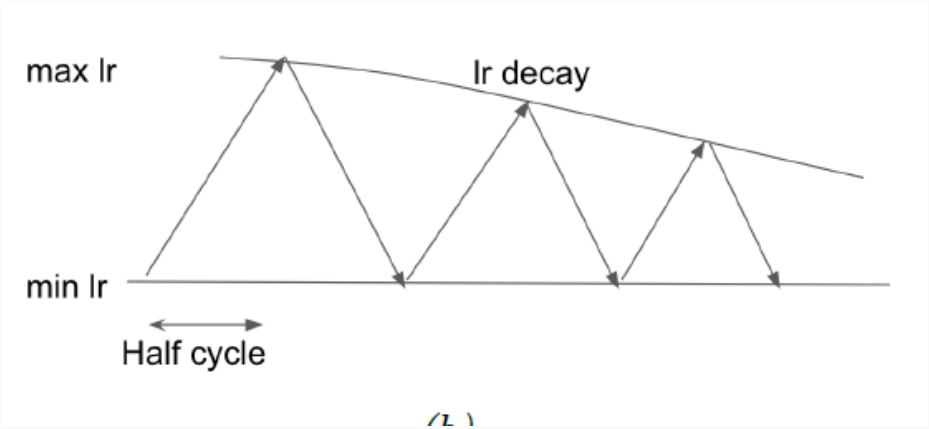


Imagen tomada de K. Murphy. Probabilistic Machine Learning, 2022.

SGD con momento

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

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

- Momento de Nesterov

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

RMSProp

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

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

$$\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]} - \frac{\alpha}{\sqrt{\mathbf{v}^{[t+1]} + \epsilon}} \odot \nabla \mathcal{L}(\boldsymbol{\theta}^{[t]})$$

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

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

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

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

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

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

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 $t + 1$

Optimizador Adam (2)

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

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

Promediando los parámetros (1)

- *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

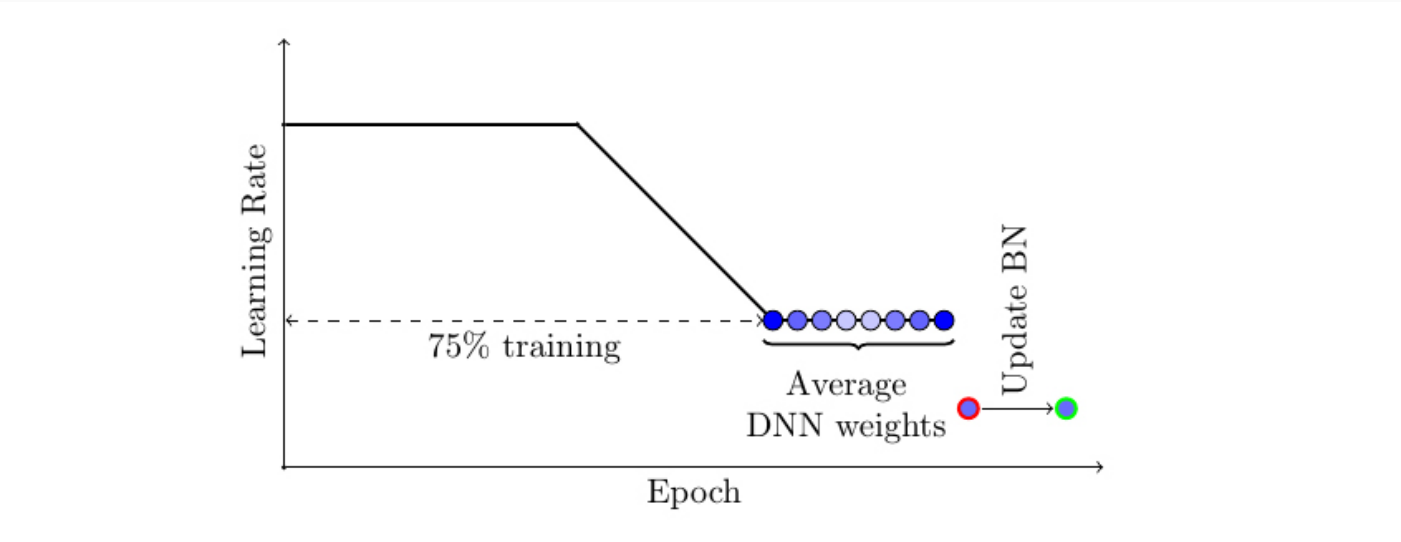


Imagen tomada de `https://pytorch.org/blog/pytorch-1.6-now-includes-stochastic-weight-averaging/`.

Promediando los parámetros (2)

- 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.