

# Redes Neuronales

**Matias Vera - Juan Zuloaga**

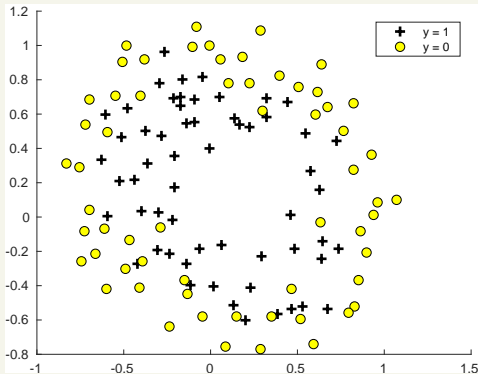
Centro de Simulación Computacional para Aplicaciones Tecnológicas

# Agenda

- 1 Redes Neuronales como aproximador universal
- 2 Deep World
- 3 ¿Cómo entrenar una deep net?

# Las soluciones lineales pueden ser insuficientes

¿Como reducir el riesgo empírico?



# Aproximación Universal

## Teorema

Sea  $\mathcal{A} \subset \mathbb{R}^m$  un conjunto cerrado y acotado y sea  $g : \mathbb{R} \rightarrow \mathbb{R}$ . Para cada función continua  $f : \mathcal{A} \rightarrow \mathbb{R}$  y  $\epsilon > 0$  existe un  $n \in \mathbb{N}$  y  $\{\mathbf{w}_i, b_i, \alpha_i\}_{i=1}^n$  con  $\mathbf{w}_i \in \mathbb{R}^m$  y  $b_i, \alpha_i \in \mathbb{R}$  para todo  $i = 1, \dots, n$  tales que

$$\left| f(\mathbf{x}) - \sum_{i=1}^n \alpha_i g(\mathbf{w}_i^T \mathbf{x} + b_i) \right| < \epsilon \quad \forall \mathbf{x} \in \mathcal{A}$$

si y solo si  $g$  no es un polinomio.

Necesito funciones no lineales!

---

*Leshno and Schocken 1993*: "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function".

# Aproximación Universal

## Teorema

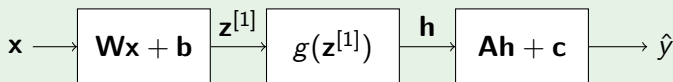
Sea  $\mathcal{A} \subset \mathbb{R}^m$  un conjunto cerrado y acotado y sea  $g : \mathbb{R} \rightarrow \mathbb{R}$ . Para cada función continua  $f : \mathcal{A} \rightarrow \mathbb{R}$  y  $\epsilon > 0$  existe un  $n \in \mathbb{N}$  y  $\{\mathbf{w}_i, b_i, \alpha_i\}_{i=1}^n$  con  $\mathbf{w}_i \in \mathbb{R}^m$  y  $b_i, \alpha_i \in \mathbb{R}$  para todo  $i = 1, \dots, n$  tales que

$$\left| f(\mathbf{x}) - \sum_{i=1}^n \alpha_i g(\mathbf{w}_i^T \mathbf{x} + b_i) \right| < \epsilon \quad \forall \mathbf{x} \in \mathcal{A}$$

si y solo si  $g$  no es un polinomio.

Necesito funciones no lineales!

## Regresión



*Leshno and Schocken 1993: "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function".*

# Aproximación Universal

## Teorema

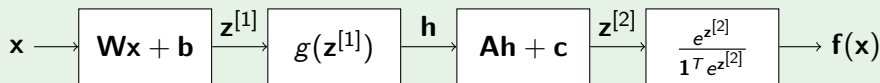
Sea  $\mathcal{A} \subset \mathbb{R}^m$  un conjunto cerrado y acotado y sea  $g : \mathbb{R} \rightarrow \mathbb{R}$ . Para cada función continua  $f : \mathcal{A} \rightarrow \mathbb{R}$  y  $\epsilon > 0$  existe un  $n \in \mathbb{N}$  y  $\{\mathbf{w}_i, b_i, \alpha_i\}_{i=1}^n$  con  $\mathbf{w}_i \in \mathbb{R}^m$  y  $b_i, \alpha_i \in \mathbb{R}$  para todo  $i = 1, \dots, n$  tales que

$$\left| f(\mathbf{x}) - \sum_{i=1}^n \alpha_i g(\mathbf{w}_i^T \mathbf{x} + b_i) \right| < \epsilon \quad \forall \mathbf{x} \in \mathcal{A}$$

si y solo si  $g$  no es un polinomio.

Necesito funciones no lineales!

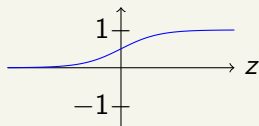
## Clasificación



*Leshno and Schocken 1993: "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function".*

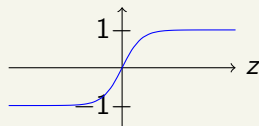
# Funciones de Activación $g$

$$g(z) = \frac{1}{1+e^{-z}}$$



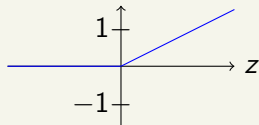
Sigmoide

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



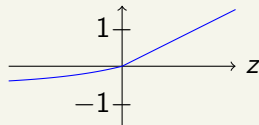
Tangente hiperbólica

$$g(z) = \max\{z, 0\}$$



ReLU

$$g(z) = \begin{cases} z & z \geq 0 \\ \alpha(e^z - 1) & z < 0 \end{cases}$$



$\alpha$ -ELU

# Agenda

- 1 Redes Neuronales como aproximador universal
- 2 Deep World
- 3 ¿Cómo entrenar una deep net?

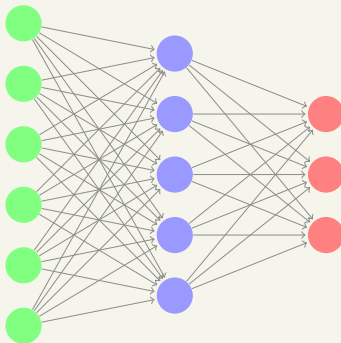


# Deep Learning

Unidades  
de entrada

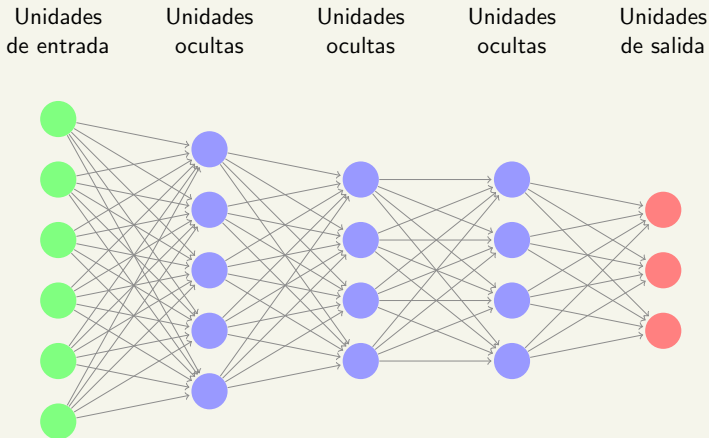
Unidades  
ocultas

Unidades  
de salida



**Red neuronal básica**

# Deep Learning



**Deep Learning Architecture**

## ¿Por que muchos layers?

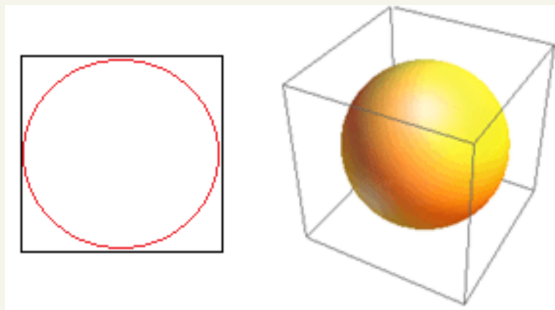
Los teoremas de aproximación universal nos hablan de la existencia de parámetros, no como elegirlos ni cuantas unidades ocultas usar. Algunas ventajas del deep-world:

- Las redes profundas suelen ser más eficientes en la etapa del entrenamiento y tienen mejores propiedades de generalización.
- La cantidad de unidades ocultas necesarias para aproximar suele decaer exponencialmente con la profundidad de la red.
- En algunos casos se puede demostrar que la cantidad de unidades ocultas necesarias crece exponencialmente con la dimensión de la entrada (otra vez la maldición).

# Características del Deep-world

- Big Data (A partir de 30K o 40K muestras)
- Alta dimensionalidad de la entrada (mayor a 10)
- Cantidad de parámetros a entrenar mayor (o al menos comparable) que la cantidad de muestras.
- Suficientes hiperparámetros para considerar hacer un barrido en grilla una mala opción.

# La maldición de la dimensionalidad



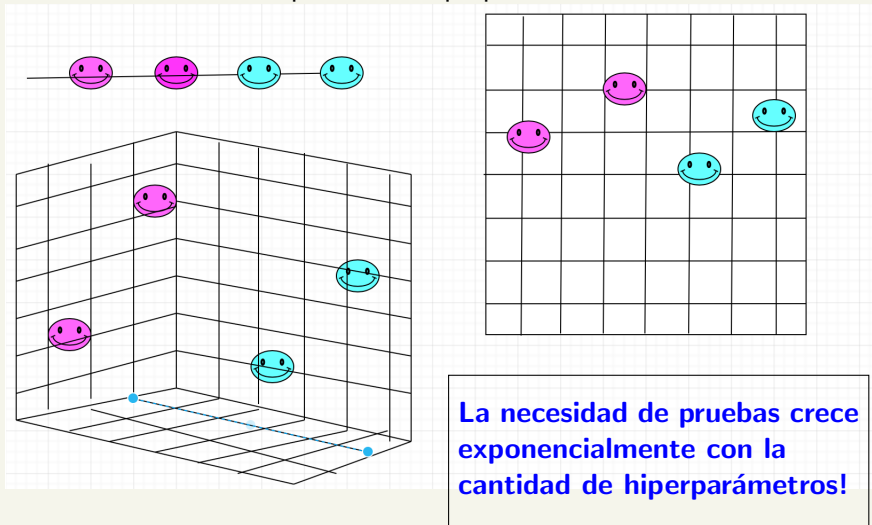
- 2d:  $\frac{\pi r^2}{(2r)^2} \approx 78.5\%$
- 3d:  $\frac{\frac{4}{3}\pi r^3}{(2r)^3} \approx 52.3\%$
- 10d:  $\frac{r^{10}}{(2r)^{10}} \pi^5 \approx 0.25\%$

En grandes dimensiones:

- Los puntos están muy lejos.
- Las estructuras son muy sparse.
- La distancia euclídea no es buena métrica.
- La “necesidad” de muestras crece exponencialmente con la dimensión.
- La cantidad de mínimos locales crece exponencialmente con la dimensión.

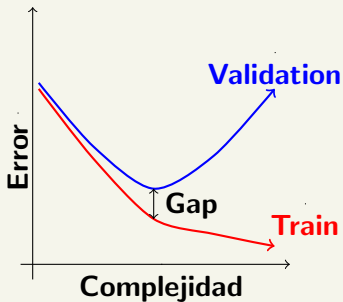
# La maldición de la dimensionalidad

La maldición también aplica a los hiperparámetros



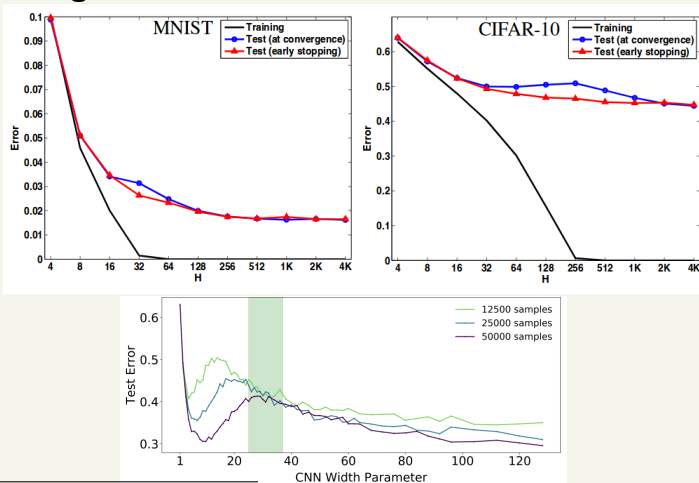
# Compromiso Sesgo/Varianza

## Aprendizaje Clásico



# Compromiso Sesgo/Varianza

## Deep Learning



Neyshabur et al. 2017: “Geometry of optimization and implicit regularization in deep learning”.

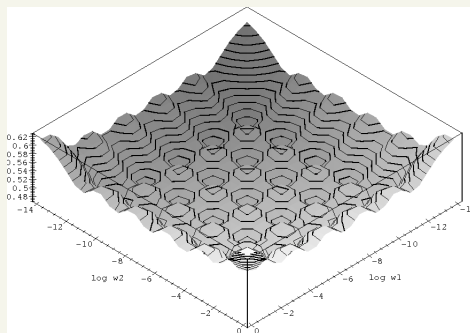
Nakkiran et al. 2019: “Deep Double Descent: Where bigger models and more data hurt”.



# La sobreparametrización

Otros impactos de la sobreparametrización:

- Muchos *saddle points*.
- La cantidad de mínimos locales es exponencial con la dimensión.
- Incluso los mínimos globales no son todos iguales para la generalización.

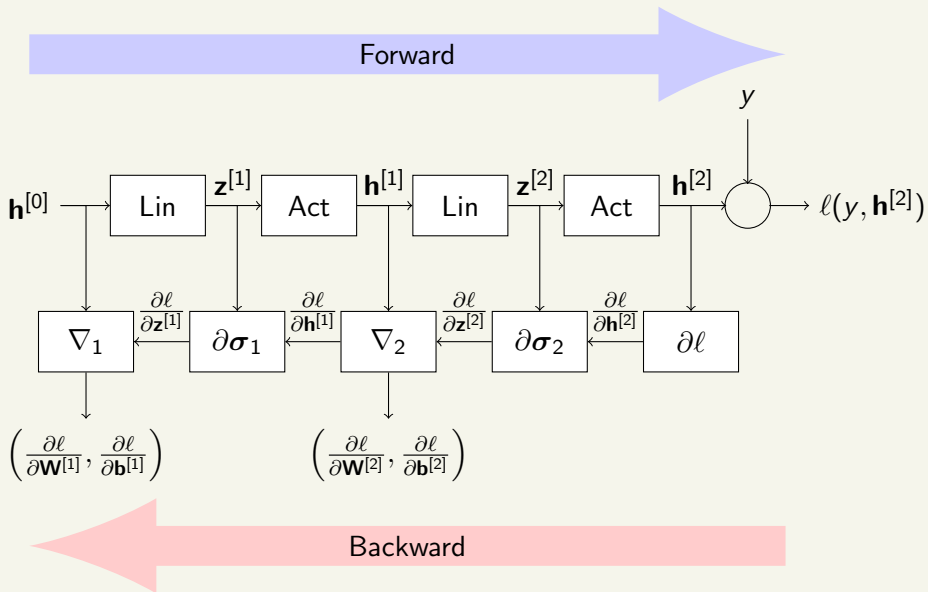


Auer et al. 1995: "Exponentially many local minima for single neurons".

# Agenda

- 1 Redes Neuronales como aproximador universal
- 2 Deep World
- 3 ¿Cómo entrenar una deep net?

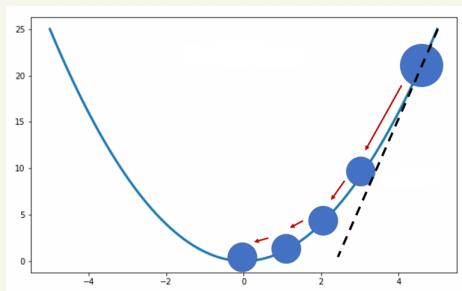
# Forward-Backward Propagation



# Gradiente Descendente

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} \frac{\partial \ell(y_i, \mathbf{h}_i^{[L]})}{\partial \theta} = \mathbf{0}$$

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial J(\theta_t)}{\partial \theta}$$



**Si tengo muchos datos y muchos parámetros es muy pesado**

# Gradiente Descendente Estocástico

Pero computar  $n_{tr}$  gradientes en cada iteración es pesado, entonces surge la versión estocástica:

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial L(\theta_t)}{\partial \theta}$$

con  $L(\theta) = \ell(y, \mathbf{h}^{[L]})$ , donde en cada paso se sortea una nueva variable aleatoria (es decir muestras distintas) introduciendo una aleatoriedad al problema. Algunas características son:

- El cómputo es mucho mas liviano que la versión clásica.
- La aleatoriedad me hace converger en una “bola de ruido” (problema de sesgo).
- Mejora la capacidad de generalización agregando ruido al proceso de aprendizaje.

## Bola de ruido

Se analizará  $\theta_{t+1} = \theta_t - \alpha \frac{\partial L(\theta_t)}{\partial \theta}$  para  $\alpha \leq \frac{2}{L+\mu}$ :

$$\begin{aligned}\mathbb{E} [\|\theta_{t+1} - \theta^*\|^2] &= \mathbb{E} [\mathbb{E} [\|\theta_{t+1} - \theta^*\|^2 \mid \theta_t]] \\&= \mathbb{E} \left[ \text{var}(\theta_{t+1} - \theta^* \mid \theta_t) + \|\mathbb{E}[\theta_{t+1} - \theta^* \mid \theta_t]\|^2 \right] \\&\leq \mathbb{E} \left[ \alpha^2 \text{var} \left( \frac{\partial L(\theta_t)}{\partial \theta} \mid \theta_t \right) + (1 - \alpha\mu)^2 \|\theta_t - \theta^*\|^2 \right] \\&\leq \alpha^2 M + (1 - \alpha\mu)^2 \mathbb{E} [\|\theta_t - \theta^*\|^2]\end{aligned}$$

con  $M \geq \mathbb{E} \left[ \text{var} \left( \frac{\partial L(\theta_t)}{\partial \theta} \mid \theta_t \right) \right]$  para todo  $t \in \mathbb{N}$ .

## Bola de ruido

Se analizará  $\theta_{t+1} = \theta_t - \alpha \frac{\partial L(\theta_t)}{\partial \theta}$  para  $\alpha \leq \frac{2}{L+\mu}$ :

$$\begin{aligned}\mathbb{E} [\|\theta_{t+1} - \theta^*\|^2] &= \mathbb{E} [\mathbb{E} [\|\theta_{t+1} - \theta^*\|^2 \mid \theta_t]] \\ &= \mathbb{E} \left[ \text{var}(\theta_{t+1} - \theta^* \mid \theta_t) + \|\mathbb{E}[\theta_{t+1} - \theta^* \mid \theta_t]\|^2 \right] \\ &\leq \mathbb{E} \left[ \alpha^2 \text{var} \left( \frac{\partial L(\theta_t)}{\partial \theta} \mid \theta_t \right) + (1 - \alpha\mu)^2 \|\theta_t - \theta^*\|^2 \right] \\ &\leq \alpha^2 M + (1 - \alpha\mu)^2 \mathbb{E} [\|\theta_t - \theta^*\|^2]\end{aligned}$$

con  $M \geq \mathbb{E} \left[ \text{var} \left( \frac{\partial L(\theta_t)}{\partial \theta} \mid \theta_t \right) \right]$  para todo  $t \in \mathbb{N}$ . La convergencia se da dentro de

$$\lim_{t \rightarrow \infty} \mathbb{E} [\|\theta_t - \theta^*\|^2] \leq \frac{\alpha^2 M}{1 - (1 - \alpha\mu)^2} = \frac{\alpha M}{\mu(2 - \alpha\mu)}$$

Tradeoff entre velocidad de convergencia y bola de ruido!

# Gradiente Descendente Estocástico por minibatch

¿No habrá nada intermedio?

$$\theta_{t+1} = \theta_t - \alpha \frac{1}{B} \sum_{i=1}^B \frac{\partial L_i(\theta_t)}{\partial \theta}$$

donde en cada paso se sortean  $B$  una nuevas variables aleatorias i.i.d. El “batchsize”  $B$  nos permite explotar el tradeoff según nuestra conveniencia. Del análisis de la bola de ruido, lo único que cambia es  $M$ :

$$\mathbb{E} \left[ \text{var} \left( \frac{1}{B} \sum_{i=1}^B \frac{\partial L_i(\theta_t)}{\partial \theta} \middle| \theta_t \right) \right] = \frac{1}{B} \mathbb{E} \left[ \text{var} \left( \frac{\partial L_1(\theta_t)}{\partial \theta} \middle| \theta_t \right) \right]$$

Obteniendo una bola de ruido de la forma:

$$\lim_{t \rightarrow \infty} \mathbb{E} [\|\theta_t - \theta^*\|^2] \leq \frac{\alpha M}{B\mu(2 - \alpha\mu)}$$



# Sobre el batchsize

- Batches grandes proporcionan una estimación más precisa del gradiente, pero de forma mas pesada a nivel computacional.
- Las arquitecturas multicore poseen un mínimo valor del batchsize debajo del cual no hay reducción en el tiempo de procesamiento.
- Si todos las muestras del batch se procesan en paralelo, entonces la cantidad de memoria escala con el tamaño del batch, limitando superiormente el batchsize.
- Algunos tipos de hardware logran un mejor tiempo de ejecución con tamaños específicos de arrays. Especialmente las GPU suelen ser más rápidas con batchsize que sean potencias de 2.
- Los batches pequeños ofrecen un efecto de regularización debido al ruido que se agrega al proceso de aprendizaje. La generalización suele ser mejor para un batchsize de 1. Pero batches tan pequeños requieren bajos learning rates lo que puede demorar el proceso de aprendizaje.

# Sobre el batchsize

## En la práctica...

Se utilizan datos distintos hasta que se acaban, realizando en principio  $n_{\text{tr}}/B$  pasos del algoritmo. Como esto suele ser insuficiente, los datos se mezclan y se vuelven a usar (construyendo batches distintos). A cada pasada de todos los datos se lo llama epoch.

Hiperparámetros de esta etapa:

- Batch size ( $B$ )
- Número de epochs (o condición de stop)

# Diferentes tipos de optimizadores

Variantes del gradiente descendente estocástico (SGD) implementadas en Keras:

- SGD with momentum
- Nesterov
- RMSProp
- Adagrad
- Adadelata
- Adam
- Adamax
- AMSGrad
- Nadam

# Momentum: Exponentially Weighted Averages

## Objetivo

Bajar la bola de ruido con alta velocidad de aprendizaje ( $\alpha$  grande) sin aumentar la carga computacional (batchsize moderado). Es decir, buscar reemplazos para  $\mathbf{g}_t = \frac{1}{B} \sum_{i=1}^B \frac{\partial L_i(\theta_{t-1})}{\partial \theta}$ .

**Momentum** (no le creas tanto):  $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$  con  $\beta_1 \in [0, 1)$  (típico  $\beta_1 = 0.9$ ).

# Momentum: Exponentially Weighted Averages

## Objetivo

Bajar la bola de ruido con alta velocidad de aprendizaje ( $\alpha$  grande) sin aumentar la carga computacional (batchsize moderado). Es decir, buscar reemplazos para  $\mathbf{g}_t = \frac{1}{B} \sum_{i=1}^B \frac{\partial L_i(\theta_{t-1})}{\partial \theta}$ .

**Momentum** (no le creas tanto):  $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$  con  $\beta_1 \in [0, 1)$  (típico  $\beta_1 = 0.9$ ).

$$\mathbf{m}_5 = (1 - \beta_1) \mathbf{g}_5 + (1 - \beta_1) \beta_1 \mathbf{g}_4 + (1 - \beta_1) \beta_1^2 \mathbf{g}_3 + (1 - \beta_1) \beta_1^3 \mathbf{g}_2 + (1 - \beta_1) \beta_1^4 \mathbf{g}_1$$

# Momentum: Exponentially Weighted Averages

## Objetivo

Bajar la bola de ruido con alta velocidad de aprendizaje ( $\alpha$  grande) sin aumentar la carga computacional (batchsize moderado). Es decir, buscar reemplazos para  $\mathbf{g}_t = \frac{1}{B} \sum_{i=1}^B \frac{\partial L_i(\theta_{t-1})}{\partial \theta}$ .

**Momentum** (no le creas tanto):  $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$  con  $\beta_1 \in [0, 1)$  (típico  $\beta_1 = 0.9$ ).

$$\mathbf{m}_5 = (1 - \beta_1) \mathbf{g}_5 + (1 - \beta_1) \beta_1 \mathbf{g}_4 + (1 - \beta_1) \beta_1^2 \mathbf{g}_3 + (1 - \beta_1) \beta_1^3 \mathbf{g}_2 + (1 - \beta_1) \beta_1^4 \mathbf{g}_1$$

$$\mathbf{m}_t = (1 - \beta_1) \sum_{k=0}^{t-1} \beta_1^k \mathbf{g}_{t-k}$$

**Respuesta:**  $\theta_t = \theta_{t-1} - \alpha \mathbf{m}_t$ .

**Learning rate efectivo:** Fue atenuado  $\alpha_{\text{ef}} = \alpha(1 - \beta_1)$ .

$$\theta_t = \theta_{t-1} - \alpha \beta_1 \mathbf{m}_{t-1} - \alpha(1 - \beta_1) \mathbf{g}_t$$

# Bias Correction

Momentum subestima mucho los gradientes en las primeras iteraciones:

$$\mathbf{m}_1 = (1 - \beta_1)\mathbf{g}_1 = 0.1\mathbf{g}_1$$

$$\mathbf{m}_2 = (1 - \beta_1)\mathbf{g}_2 + (1 - \beta_1)\beta_1\mathbf{g}_1 = 0.1\mathbf{g}_2 + 0.09\mathbf{g}_1$$

## Bias Correction

Momentum subestima mucho los gradientes en las primeras iteraciones:

$$\mathbf{m}_1 = (1 - \beta_1)\mathbf{g}_1 = 0.1\mathbf{g}_1$$

$$\mathbf{m}_2 = (1 - \beta_1)\mathbf{g}_2 + (1 - \beta_1)\beta_1\mathbf{g}_1 = 0.1\mathbf{g}_2 + 0.09\mathbf{g}_1$$

**Solución:** Verifiquemos que sea una combinación convexa de gradientes.

$$\hat{\mathbf{m}}_t = \frac{(1 - \beta_1) \sum_{k=0}^{t-1} \beta_1^k \mathbf{g}_{t-k}}{(1 - \beta_1) \sum_{k=0}^{t-1} \beta_1^k} = \frac{\mathbf{m}_t}{1 - \beta_1^t}$$

**Respuesta:**  $\theta_t = \theta_{t-1} - \alpha \hat{\mathbf{m}}_t$ .

**Learning rate efectivo:** Atenuación paulatina  $\alpha_{\text{ef}} = \alpha \frac{1 - \beta_1}{1 - \beta_1^t}$ .



# Nesterov's accelerated gradient

**Classical momentum:**  $\theta_t = \theta_{t-1} - \alpha\beta_1\mathbf{m}_{t-1} - \alpha(1 - \beta_1)\frac{\partial L(\theta_{t-1})}{\partial \theta}$

---

*Sutskever et al. 2013:* “On the importance of initialization and momentum in deep learning”.

# Nesterov's accelerated gradient

**Classical momentum:**  $\theta_t = \theta_{t-1} - \alpha\beta_1\mathbf{m}_{t-1} - \alpha(1 - \beta_1)\frac{\partial L(\theta_{t-1})}{\partial \theta}$

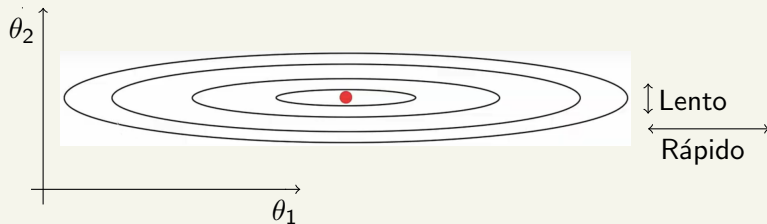
**Nesterov momentum:**  $\theta_t = \theta_{t-1} - \alpha\beta_1\mathbf{m}_{t-1} - \alpha(1 - \beta_1)\frac{\partial L(\theta_{t-1} - \alpha\beta_1\mathbf{m}_{t-1})}{\partial \theta}$

La idea del gradiente es que me diga cuanto y hacia donde me tengo que mover para acercarme al mínimo. Nesterov usa toda la información que tengo actualmente antes de elegir cuanto y hacia donde moverse.

---

*Sutskever et al. 2013:* "On the importance of initialization and momentum in deep learning".

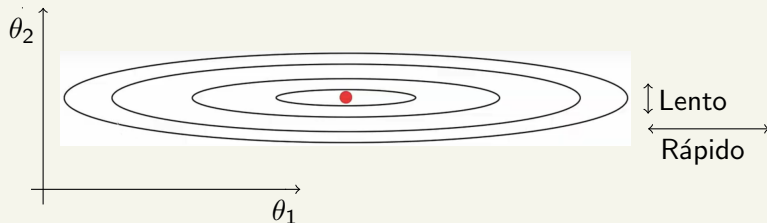
# Adaptive Gradient (Adagrad)



---

*Duchi et al. 2011: "Adaptive subgradient methods for online learning and stochastic optimization".*

# Adaptive Gradient (Adagrad)



**Solución:** Usar la historia del gradiente para escalarlo. Se define un parámetro que evite errores numéricos  $\epsilon \sim 10^{-8}$ .

**Respuesta:**  $\theta_t = \theta_{t-1} - \alpha \frac{\mathbf{g}_t}{\sqrt{\sum_{\tau=1}^t \mathbf{g}_\tau^2 + \epsilon}}$ .

**Learning rate efectivo:** Se adapta a cada componente

$$\alpha_{\text{ef}} = \frac{\alpha}{\sqrt{\sum_{\tau=1}^t \mathbf{g}_\tau^2 + \epsilon}}.$$

---

*Duchi et al. 2011:* “Adaptive subgradient methods for online learning and stochastic optimization”.

# Root Mean Square Propagation

## RMSProp

Exponentially weighted averages para los cuadrados:  $\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$ .

**Respuesta:**  $\theta_t = \theta_{t-1} - \alpha \frac{\mathbf{g}_t}{\sqrt{\mathbf{v}_t + \epsilon}}$ .

**Learning rate efectivo:**  $\alpha_{\text{ef}} = \frac{\alpha}{\sqrt{\mathbf{v}_t + \epsilon}}$ .

## RMSProp with bias correction

Misma idea:  $\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t}$ .

**Respuesta:**  $\theta_t = \theta_{t-1} - \alpha \frac{\mathbf{g}_t}{\sqrt{\hat{\mathbf{v}}_t + \epsilon}}$ .

**Learning rate efectivo:**  $\alpha_{\text{ef}} = \frac{\alpha}{\sqrt{\hat{\mathbf{v}}_t + \epsilon}}$ .

## ADAM

Combinar RMSProp con momentum (ambos con bias correction).

**Respuesta:**  $\theta_t = \theta_{t-1} - \alpha \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t + \epsilon}}$ .

**Learning rate efectivo:**  $\alpha_{\text{ef}} = \alpha \frac{1 - \beta_1}{(1 - \beta_1^t)(\sqrt{\hat{\mathbf{v}}_t + \epsilon})}$ .

---

Kingma-Lei Ba 2015: "ADAM: A method for stochastic optimization".

# Variantes de ADAM

## ADAMAX

Misma idea de RMSProp pero con norma infinito:  $\mathbf{u}_t = \max\{\beta_2 \mathbf{u}_{t-1}, |\mathbf{g}_t|\}$ .

**Respuesta:**  $\theta_t = \theta_{t-1} - \alpha \frac{\hat{\mathbf{m}}_t}{\mathbf{u}_t}$ .

**Learning rate efectivo:**  $\alpha_{\text{ef}} = \alpha \frac{1-\beta_1}{(1-\beta_1^t) \mathbf{u}_t}$ .

## AMSGrad

Variante sobre los  $\mathbf{v}_t$  que corrige algunos problemas de convergencia propios de RMSProp.

## NADAM

Si ADAM es RMSProp con momentum, NADAM es RMSProp con Nesterov momentum.

---

*Kingma-Lei Ba 2015: "ADAM: A method for stochastic optimization".*

*Reddi et al. 2018: "On the convergence of ADAM and beyond".*

*Dozat 2016: "Incorporating Nesterov Momentum into ADAM".*

# Adadelta

Deja que el learning rate se elija solo!

$$\begin{aligned}\mathbf{s}_t &= \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2 \\ \Delta_t &= \beta_2 \Delta_{t-1} + (1 - \beta_2) \theta_t^2 \\ \theta_t &= \theta_{t-1} - \alpha_0 \sqrt{\frac{\Delta_{t-1} + \epsilon}{\mathbf{s}_t + \epsilon}} \mathbf{g}_t\end{aligned}$$

**Learning rate efectivo:**  $\alpha_{\text{ef}} = \alpha_0 \sqrt{\frac{\Delta_{t-1} + \epsilon}{\mathbf{s}_t + \epsilon}}$ .

## Remark

$\alpha_0$  NO ES el learning-rate! Se suele setear en  $\alpha_0 = 1$  (mientras que los  $\alpha$  suelen ser bastante más chicos). De hecho tanto en el artículo original como en algunas implementaciones no está este hiperparámetro.

---

Zeiler 2012: "ADADELTA: An adaptive learning rate method".