

*Departamento de Física Médica - Centro atómico Bariloche - IB*

# Entrenamiento de redes neuronales

Ariel Hernán Curiale  
[ariel.curiale@cab.cnea.gov.ar](mailto:ariel.curiale@cab.cnea.gov.ar)

La mayoría de los slides fueron adaptados de Fei Fei Li, J. Johnson y S. Yeung, cs231n, Stanford 2017.



**UNCUYO**  
UNIVERSIDAD  
NACIONAL DE CUYO

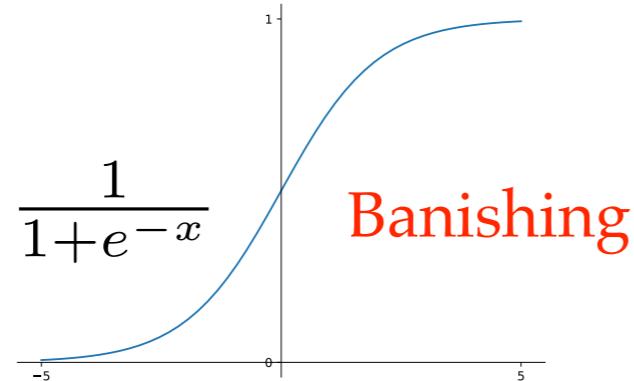


# Funciones de activación

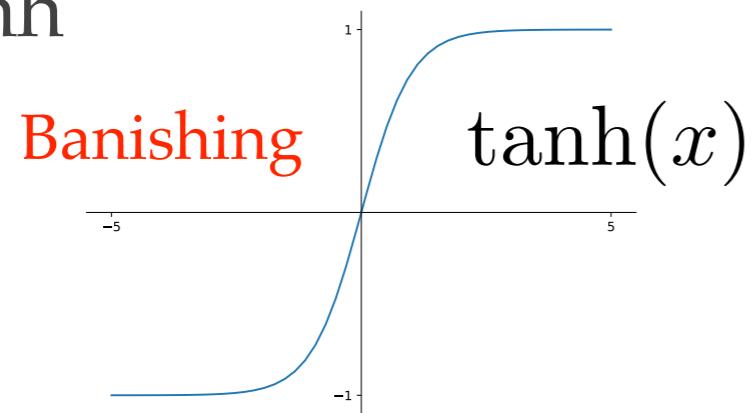
Recordemos las funciones de activación que vimos

- ❖ Sigmoid

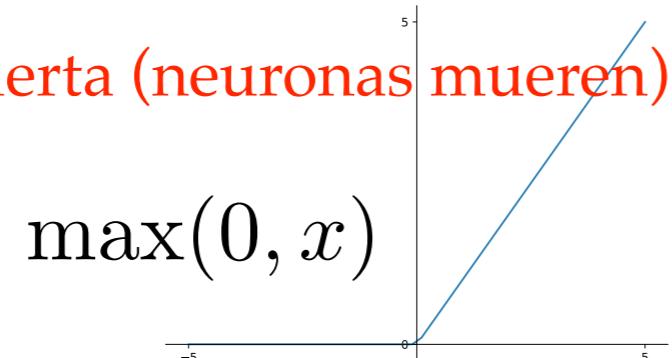
$$f(x) = \frac{1}{1+e^{-x}}$$



- ❖ tanh

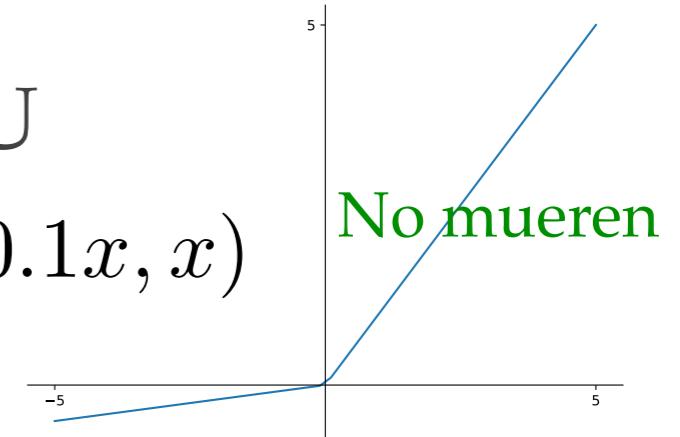


- ❖ ReLU



- ❖ Leaky ReLU

$$\max(0.1x, x)$$



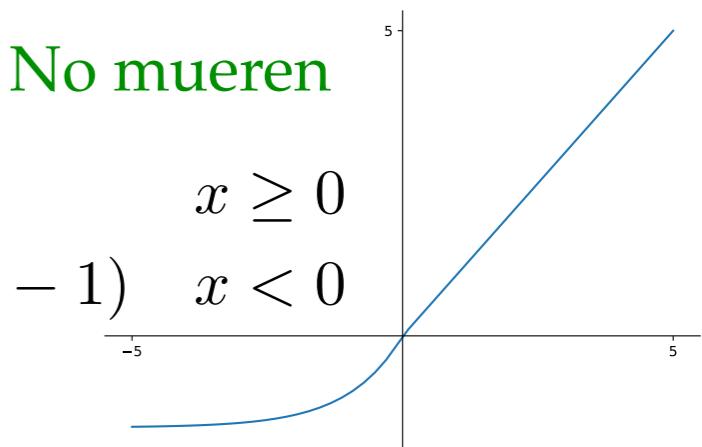
- ❖ Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

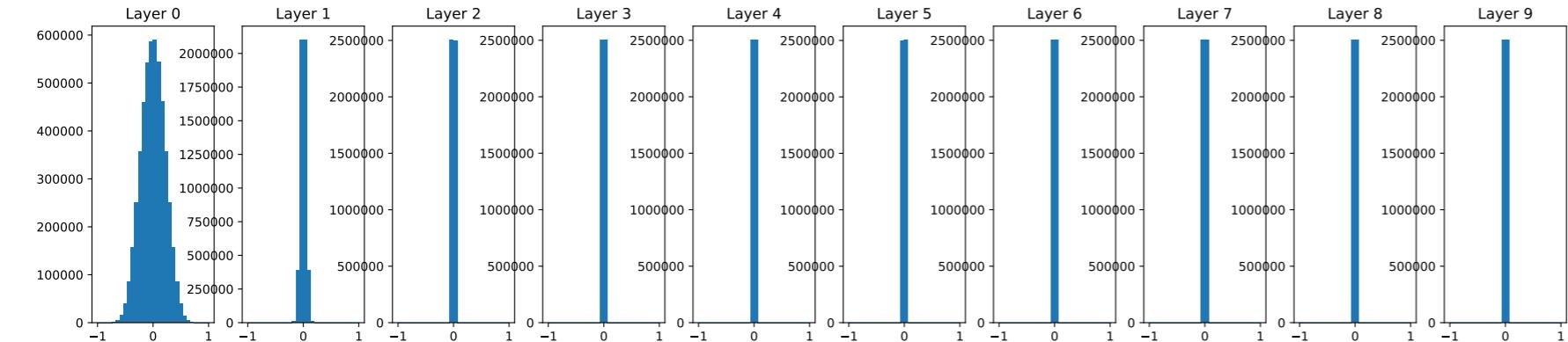
No mueren

- ❖ ELU

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

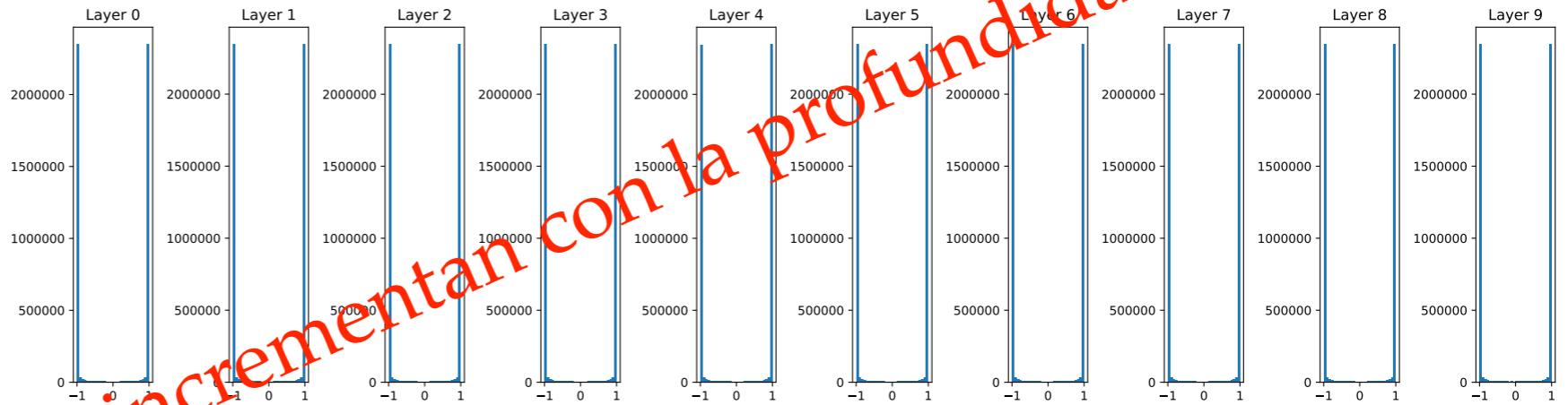


# Inicialización

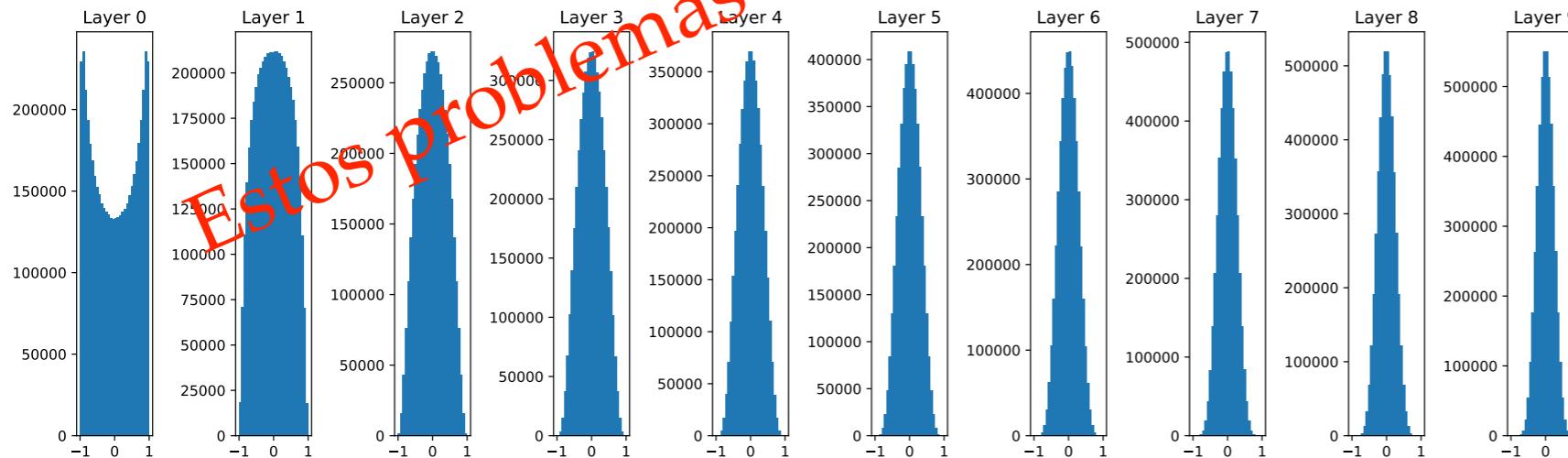


Muy chica: Las activaciones se van a cero y por tanto no se aprende

Muy grande: Las activaciones se saturan y por tanto no se aprende



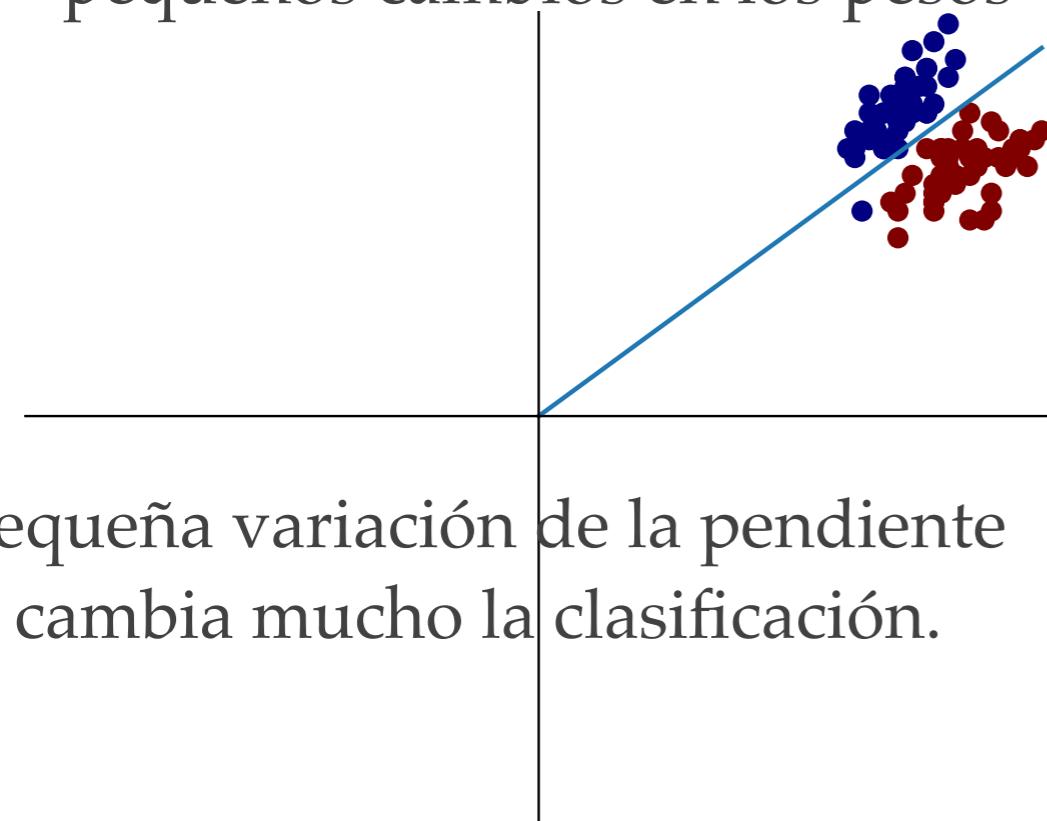
Justa: Las activaciones presentan una buena dist. en todas las capas



Estos problemas se incrementan con la profundidad de la red

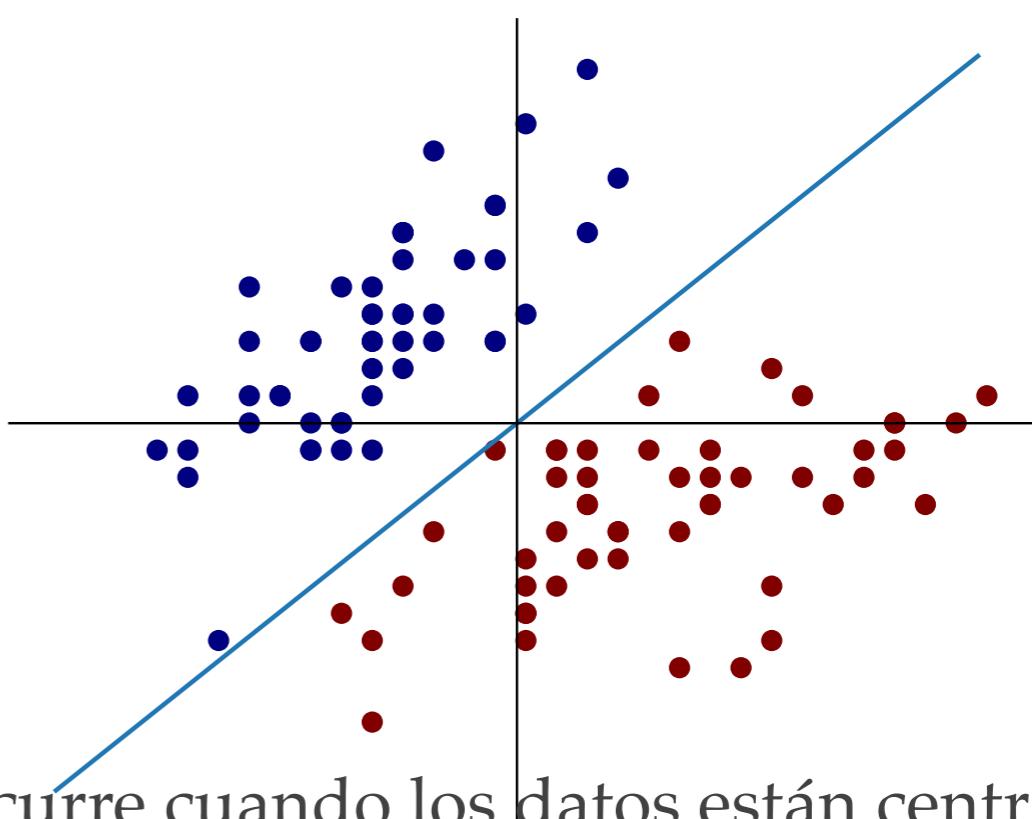
# Preprocesado

Difícil de optimizar, muy sensible a pequeños cambios en los pesos



Pequeña variación de la pendiente cambia mucho la clasificación.

Más fácil de optimizar, menos sensible a pequeños cambios en los pesos



Este mismo problema motiva a usar BN:  
pequeños cambios en los pesos pueden arruinar  
la clasificación incrementando la loss y haciendo  
más difícil la optimización

Esto no ocurre cuando los datos están centrados.  
Pequeñas perturbaciones de la función de costo no  
cambia la clasificación

# Batch Normalization

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_{1\dots m}\}$ ;  
Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\begin{aligned}\boxed{\mu_{\mathcal{B}}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && // \text{mini-batch mean} \\ \boxed{\sigma_{\mathcal{B}}^2} &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 && // \text{mini-batch variance} \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} && // \text{normalize} \\ y_i &\leftarrow \boxed{\gamma} \hat{x}_i + \boxed{\beta} \equiv \text{BN}_{\gamma, \beta}(x_i) && // \text{scale and shift}\end{aligned}$$

se calculan durante el  
entrenamiento para  
cada característica

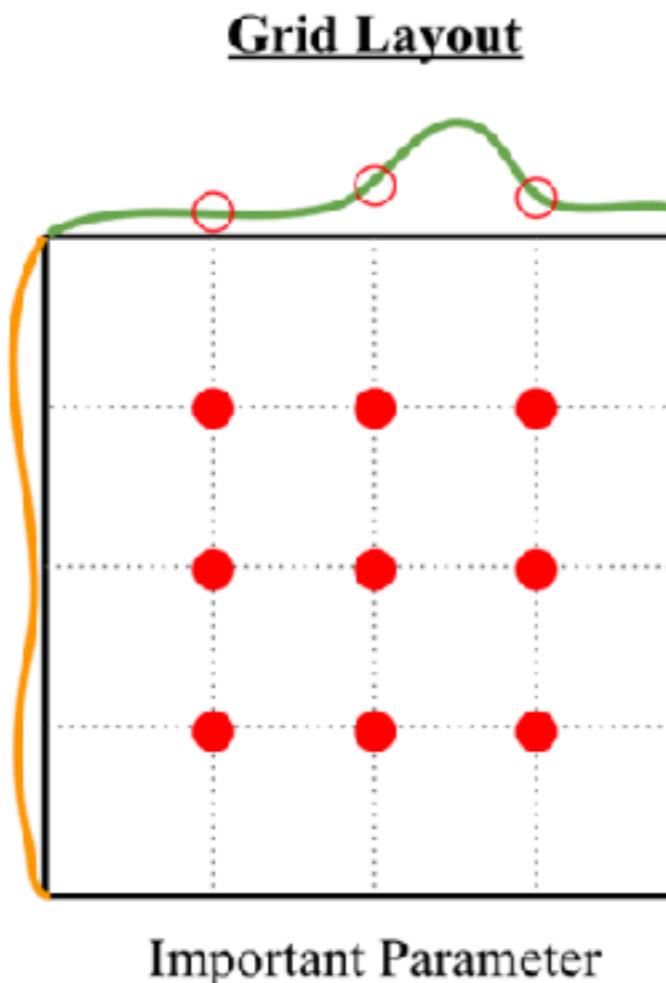
Se aprenden para  
cada característica

**Algorithm 1:** Batch Normalizing Transform, applied to  
activation  $x$  over a mini-batch.

[Ioffe and Szegedy, 2015]

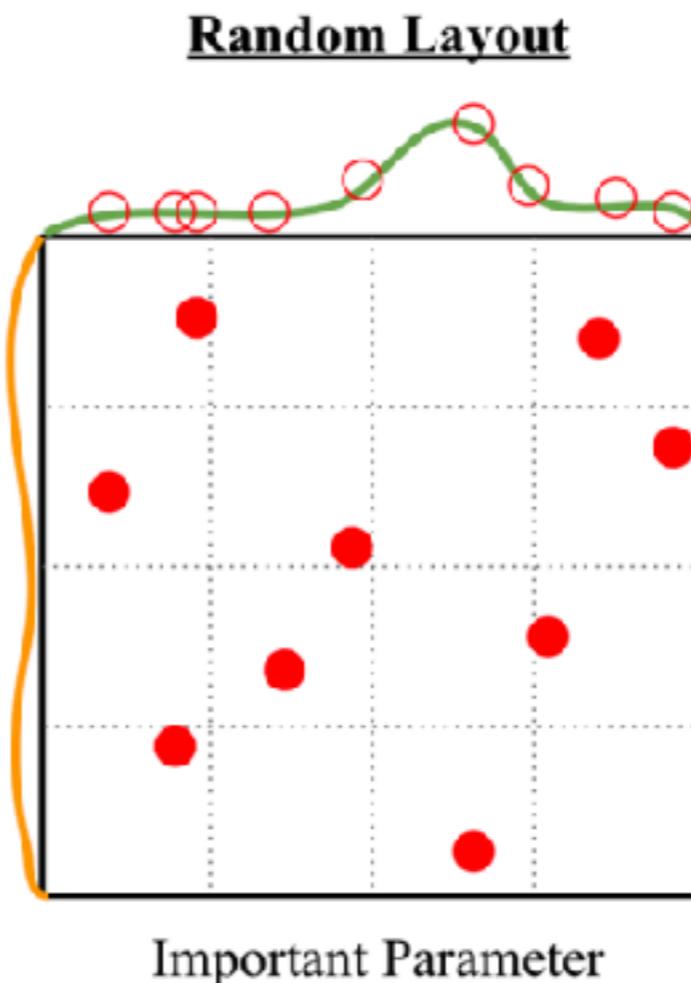
# Hiperparámetros y Monitoreo

- ❖ Coarse to fine: Iteramos hasta encontrar los hiperparámetros
  - ❖ Grid vs. Random



Unimportant Parameter

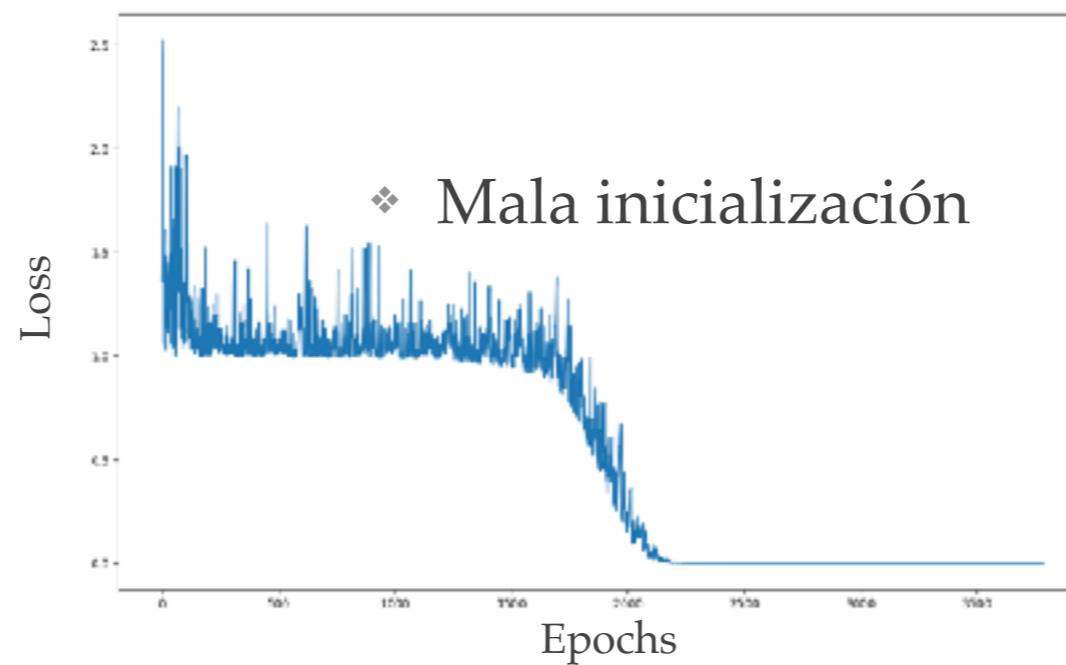
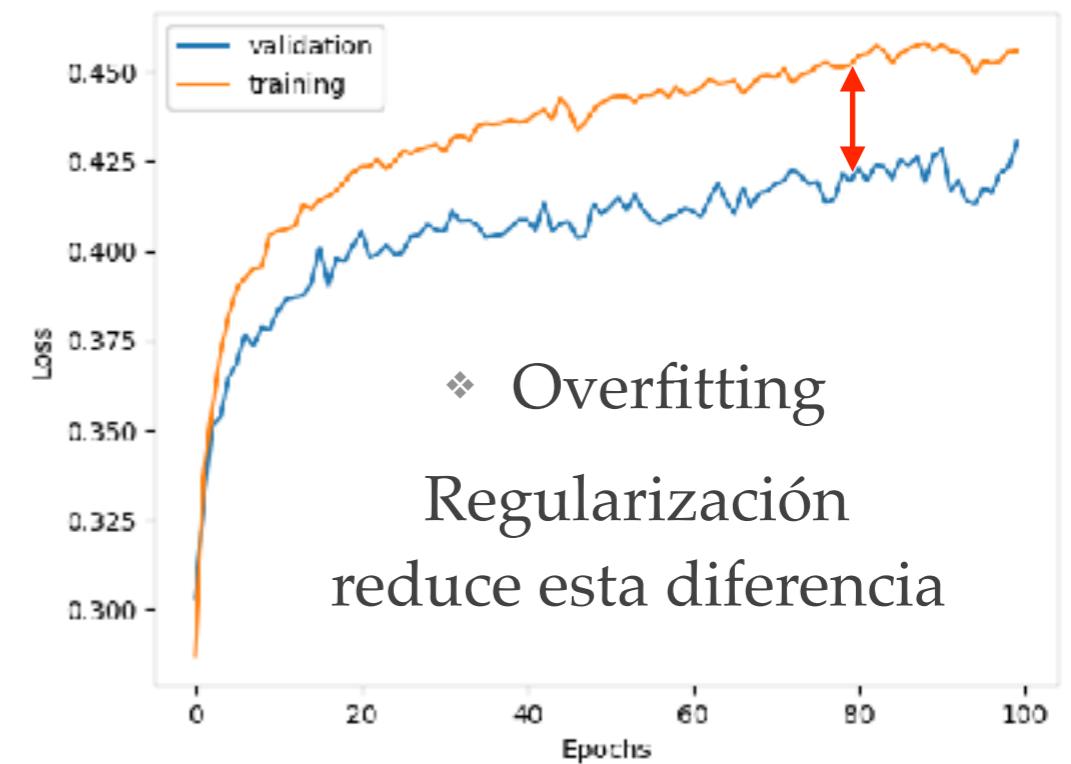
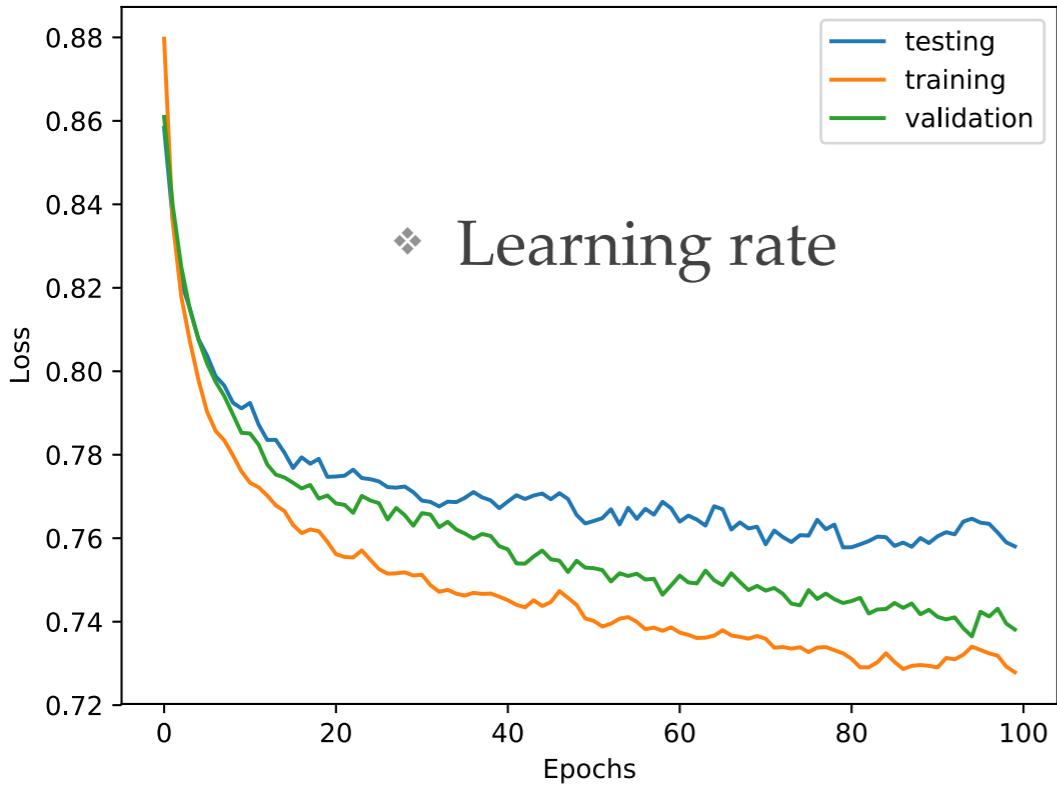
Important Parameter



Unimportant Parameter

Important Parameter

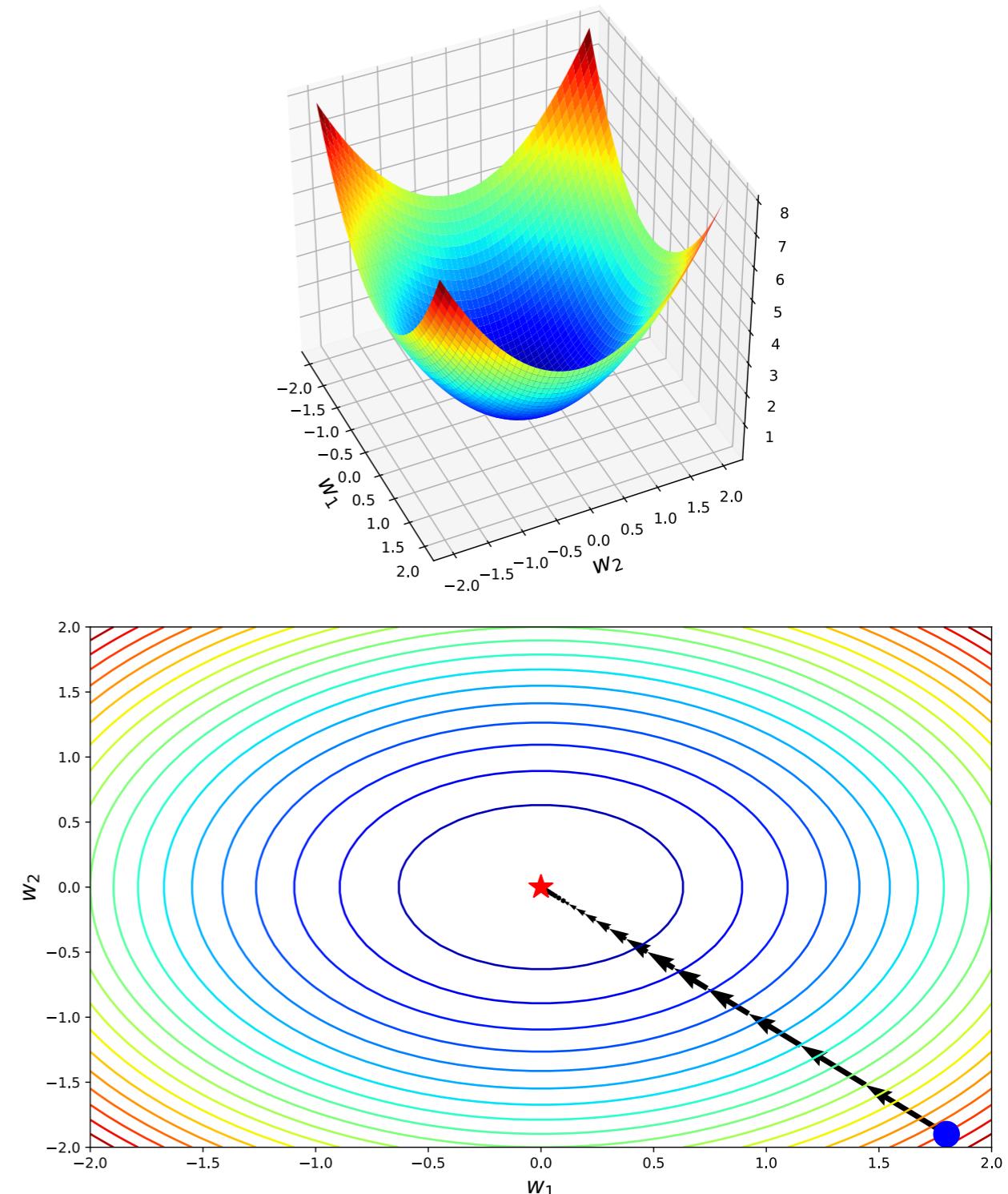
# Hiperparámetros y Monitoreo



# Optimización

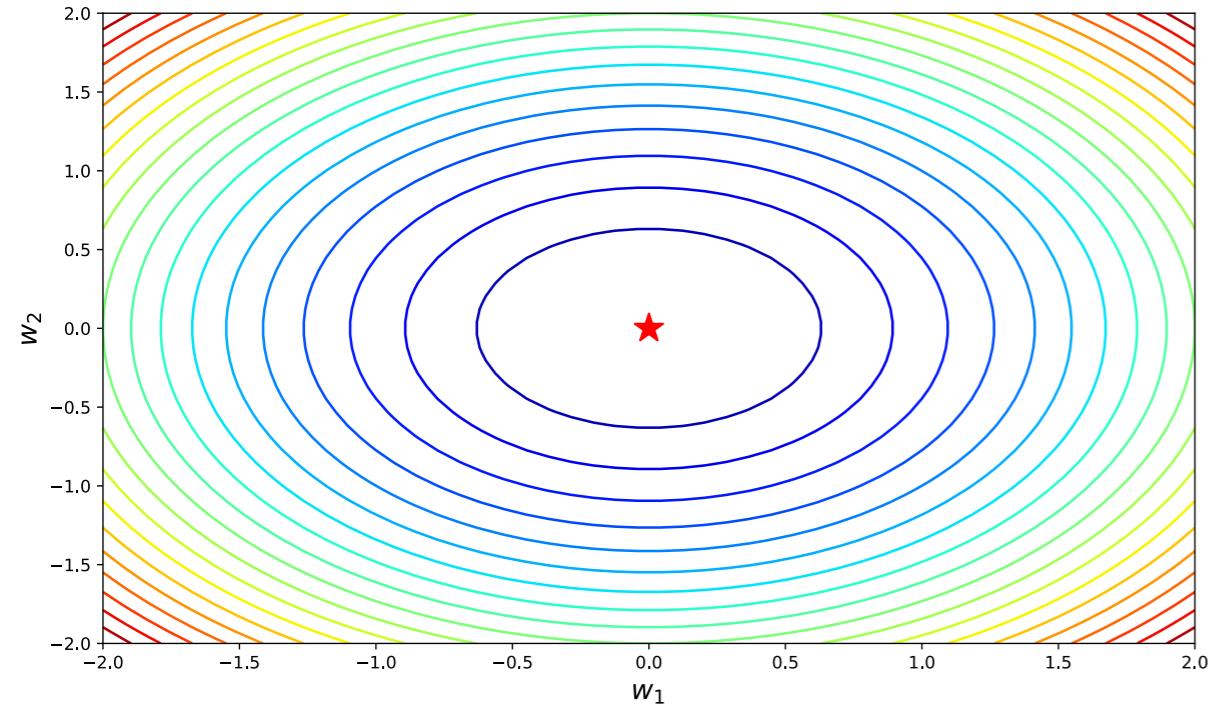
# Stochastic Gradient Descent (SGD)

```
1 def SGD(x, y, weights, epochs, lr=1e-3,
2         bs=100, tol=1e-5):
3     delta = np.inf
4     nit=0
5     path = [weights.copy()]
6     while nit < epochs and delta > tol:
7         idd = np.r_[x.shape[0]]
8         np.random.shuffle(idd)
9         l0 = loss_func(x,y,weights)
10        for i in range(0,x.shape[0], bs):
11            idi = idd[i:it+bs]
12            x_batch = x[idi]
13            y_batch = y[idi]
14            loss, dw = loss_gradient(x_batch,
15                                      y_batch, weights)
16            weights -= lr * dw
17            path.append(weights.copy())
18            ln = loss_func(x,y,weights)
19            delta = (l0-ln)**2
20            nit +=1
21    return path
```

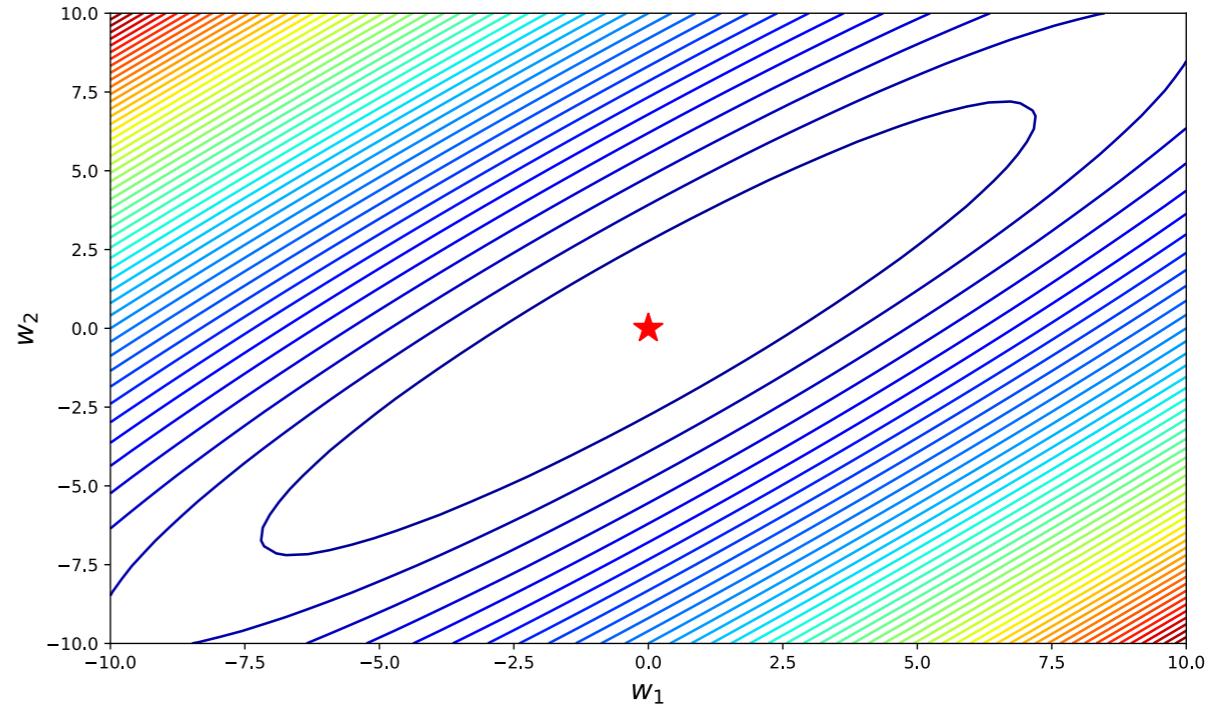


# Número de condición

- ❖ Número de condición de una función,  $K$ , mide cuánto varía la función frente a variaciones de la entrada. Para matrices (matriz Hessiana) representa el ratio entre el máximo y mínimo autovalor. **En redes profundas, el número de condición de la matriz Hessiana es grande en la práctica**



$K \sim 1$

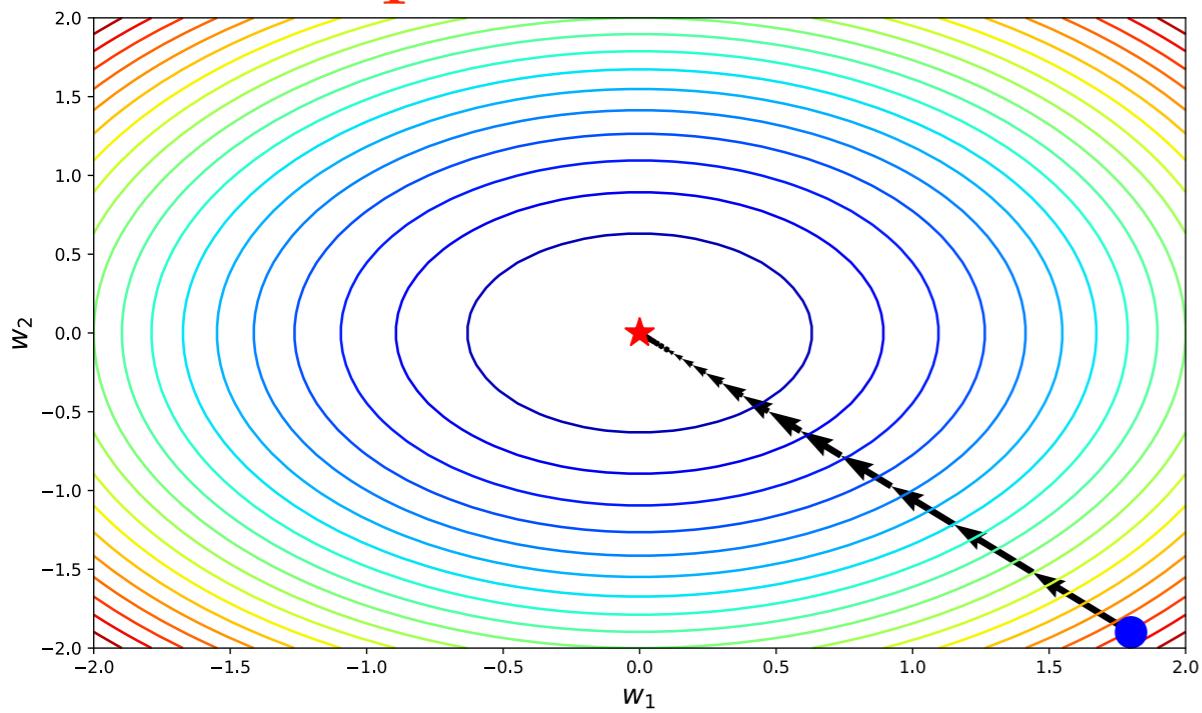


$K \gg 1$

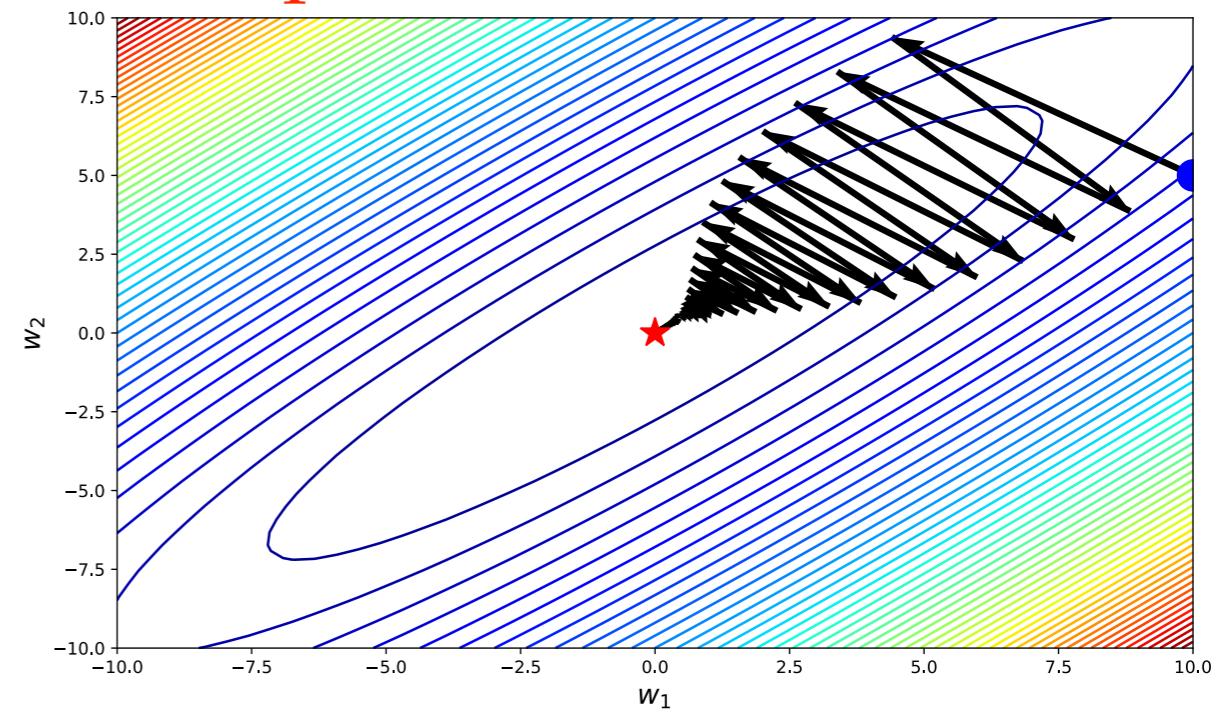
# Problemas con SGD

- ❖ Efecto zig-zag en funciones con  $K \gg 1$ : Optimización lenta en la dirección de menor variación y alternante en la de mayor variación.

En redes profundas hay muchas direcciones donde el proceso de optimización es más lento (en la práctica más del 80%).



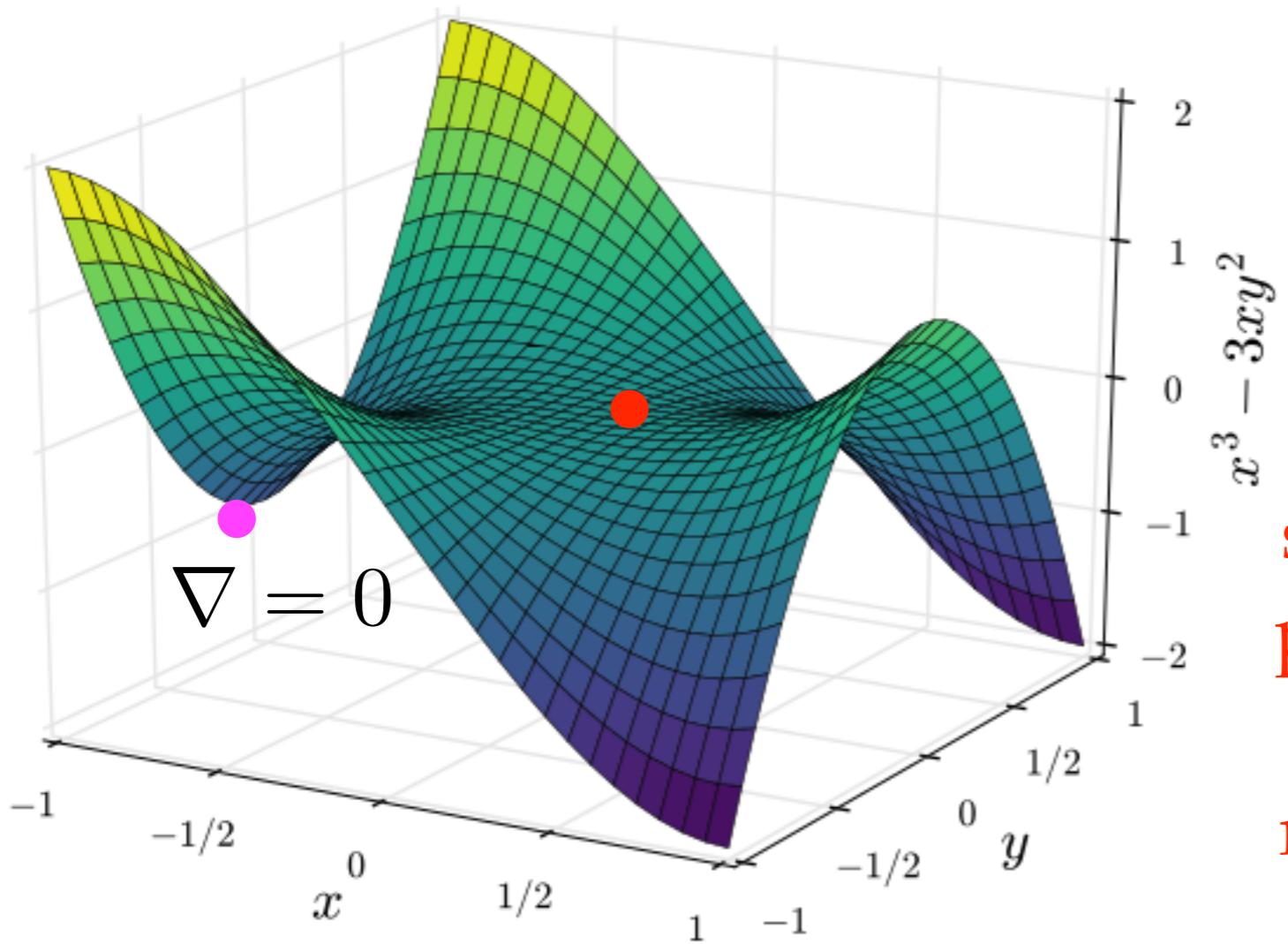
$K \sim 1$



$K \gg 1$

# Problemas con SGD

- ❖ Mínimos locales y Puntos de ensilladura: El método SGD



En espacios de alta dimensionalidad los puntos de ensilladura son mucho más comunes

El proceso de optimización se vuelve muy lento en estos pts (ensilladura), algo que no pasa comúnmente en mínimos locales (ver gráfica)

# Problemas con SGD

$$f(x) = w_0 + w_1 x_1 + w_2 * x_2$$

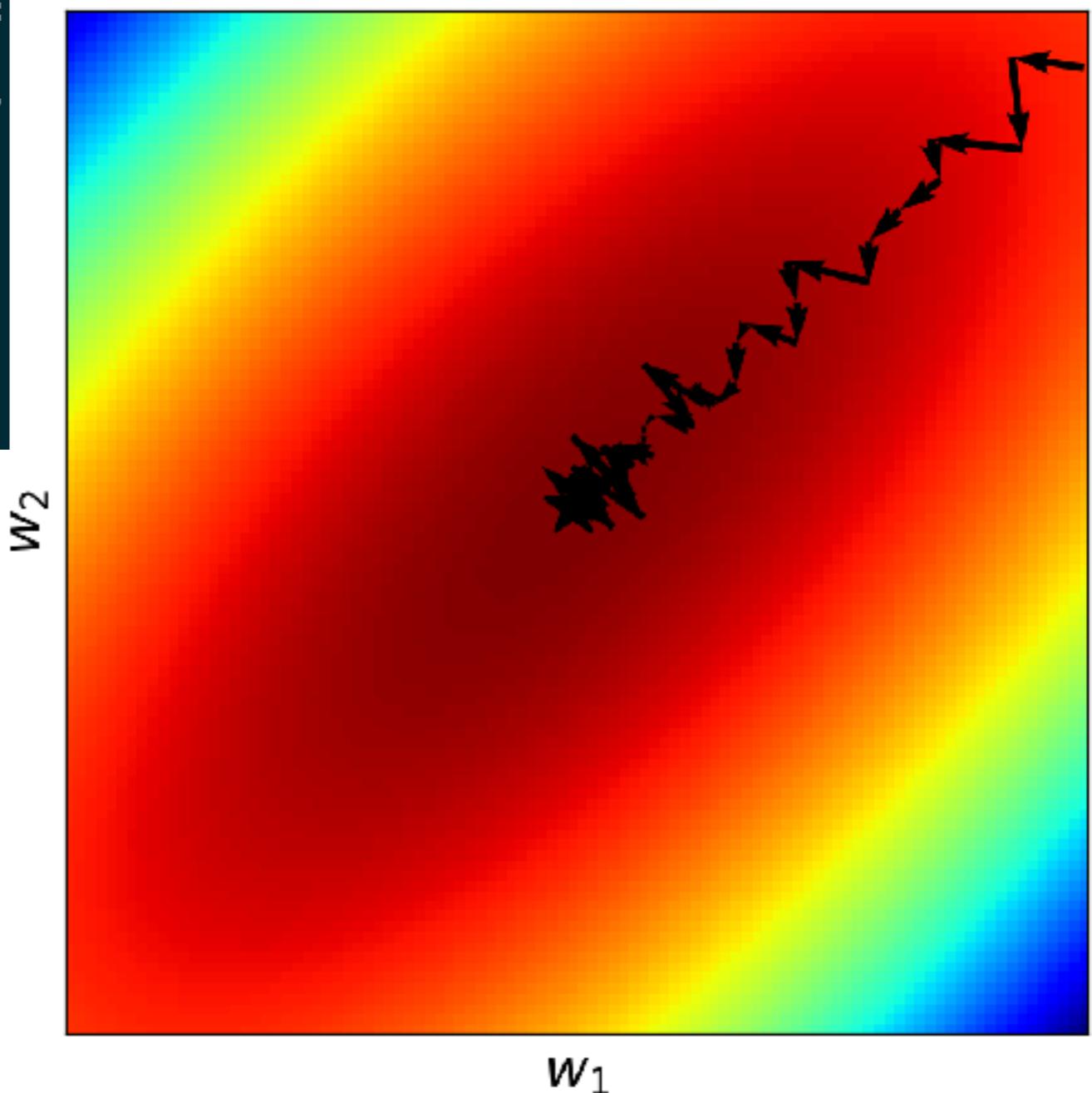
$$\text{MSE} = \frac{1}{N} \sum_i ||y_i - f(x_i)||_2^2$$

```
1 def loss_gradient(data_batch, y_batch, weights):
2     data_input = np.hstack([
3         np.ones((data_batch.shape[0], 1)),
4         data_batch])
5     predict = data_input.dot(weights)
6     diff = y_batch - predict
7     loss = np.square(diff).mean()
8     dy = -2*diff
9     dw = data_input.T.dot(dy)
10    dw /= data_batch.shape[0]
11    return loss, dw
```



El gradiente es estocástico,  
por eso suele ser ruidoso

La optimización tarda  
más tiempo



# SGD + Momento

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

VS

SGD + Momento

$$v_0 = 0$$

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

Tiene en cuenta la velocidad con la que viene

- ❖ Calculamos la velocidad, que sería una especie de promedio del gradiente, así por más que el gradiente cambie un poco se continua en la misma dirección que venía
- ❖ Rho representa la fricción, normalmente es 0.9 o 0.99

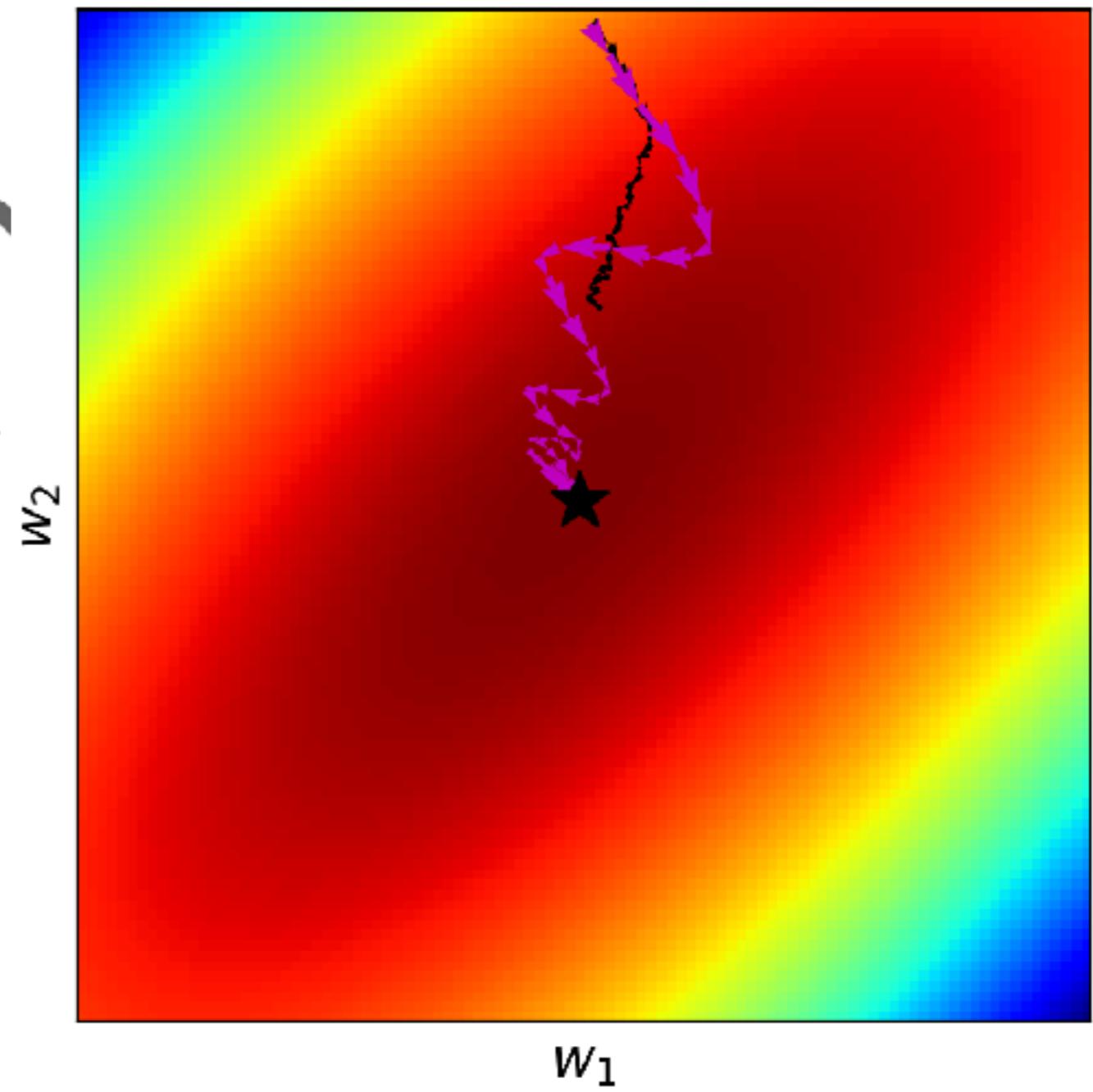
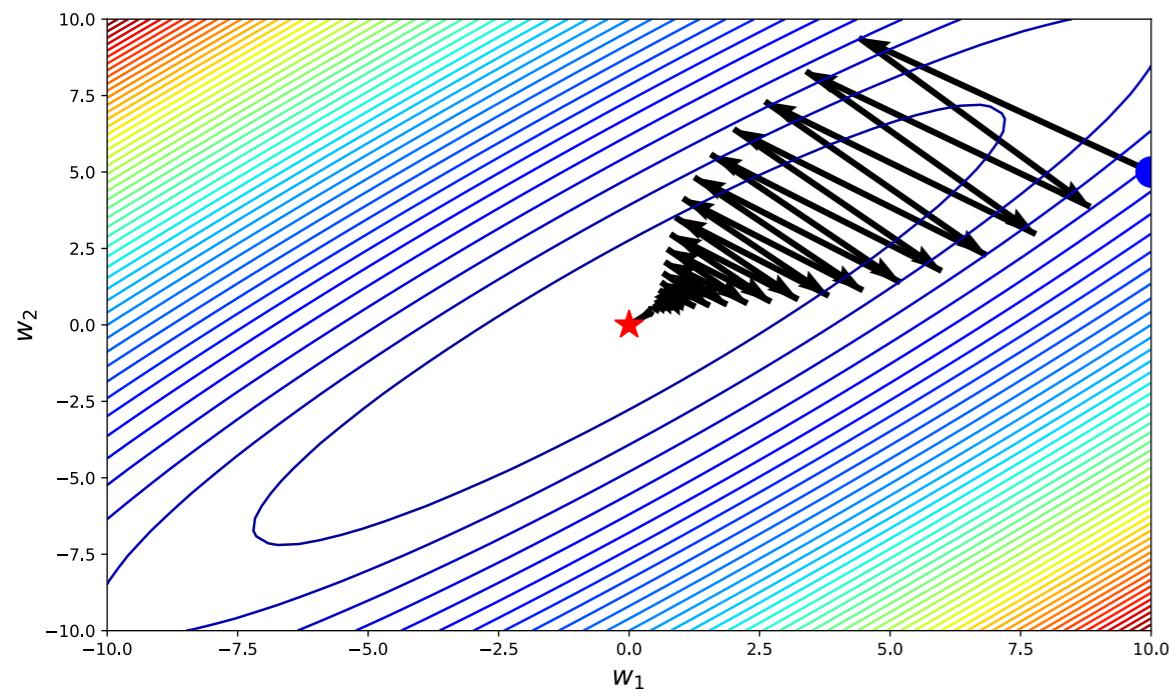
# SGD + Momento

Min. locales



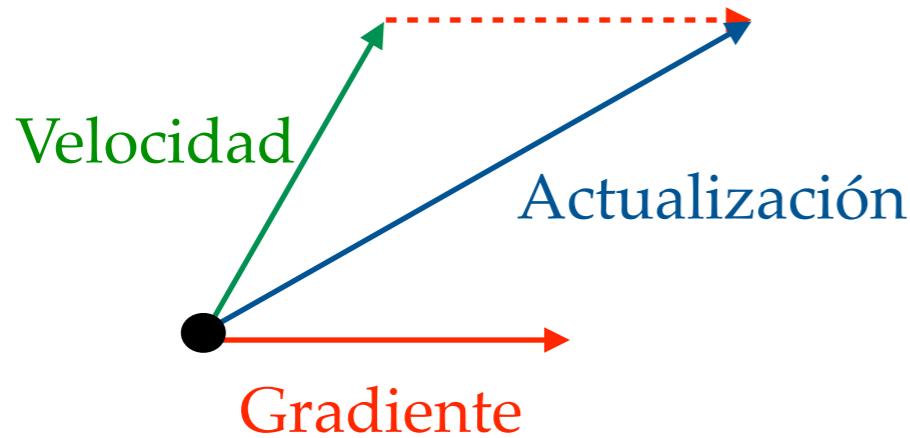
Pts. ensilladura

Funciones mal condicionadas



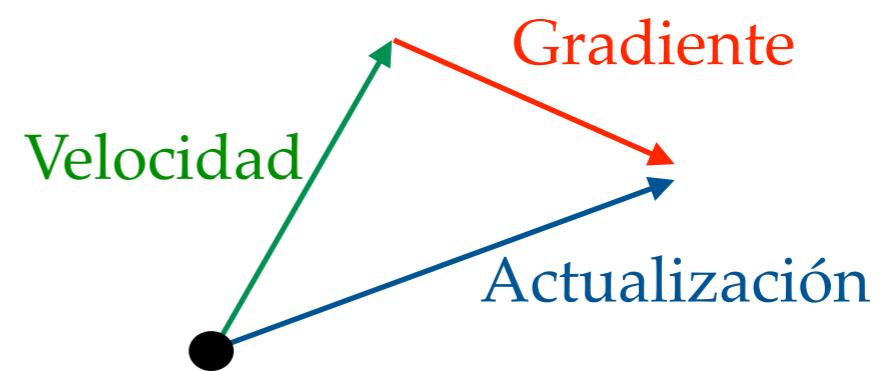
# Método de Nesterov

Momento clásico



Damos un salto en la dir.  
del gradiente y corregimos  
con la velocidad

Método de Nesterov

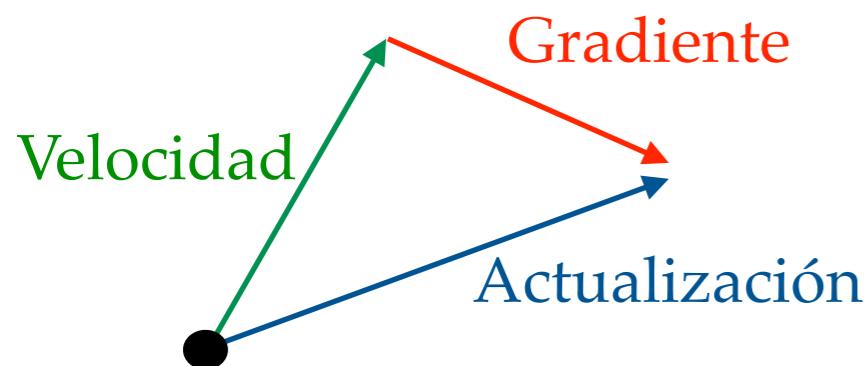


¡Mejor corregir antes de dar el salto !

Corregimos la posición con la  
velocidad y luego calculamos el  
gradiente para ver a donde ir

# Método de Nesterov

## Método de Nesterov



$$\begin{aligned} v_{t+1} &= \rho v_t - \alpha \nabla f(x_t + \rho v_t) \\ x_{t+1} &= x_t + v_{t+1} \end{aligned}$$

$\hat{x}_t = x_t + \rho v_t$

Interesantes propiedades matemáticas en funciones convexas  
(No pasa en NN)

# Método de Nesterov

$$\hat{x}_t = x_t + \rho v_t$$

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t) \longrightarrow v_{t+1} = \rho v_t - \alpha \nabla f(\hat{x}_t)$$

$$x_{t+1} = x_t + v_{t+1} = x_t + \rho v_t - \alpha \nabla f(\hat{x}_t) = \hat{x}_t - \alpha \nabla f(\hat{x}_t)$$

$$x_{t+1} \boxed{+ \rho v_{t+1} - \rho v_{t+1}} = \hat{x}_t - \alpha \nabla f(\hat{x}_t)$$

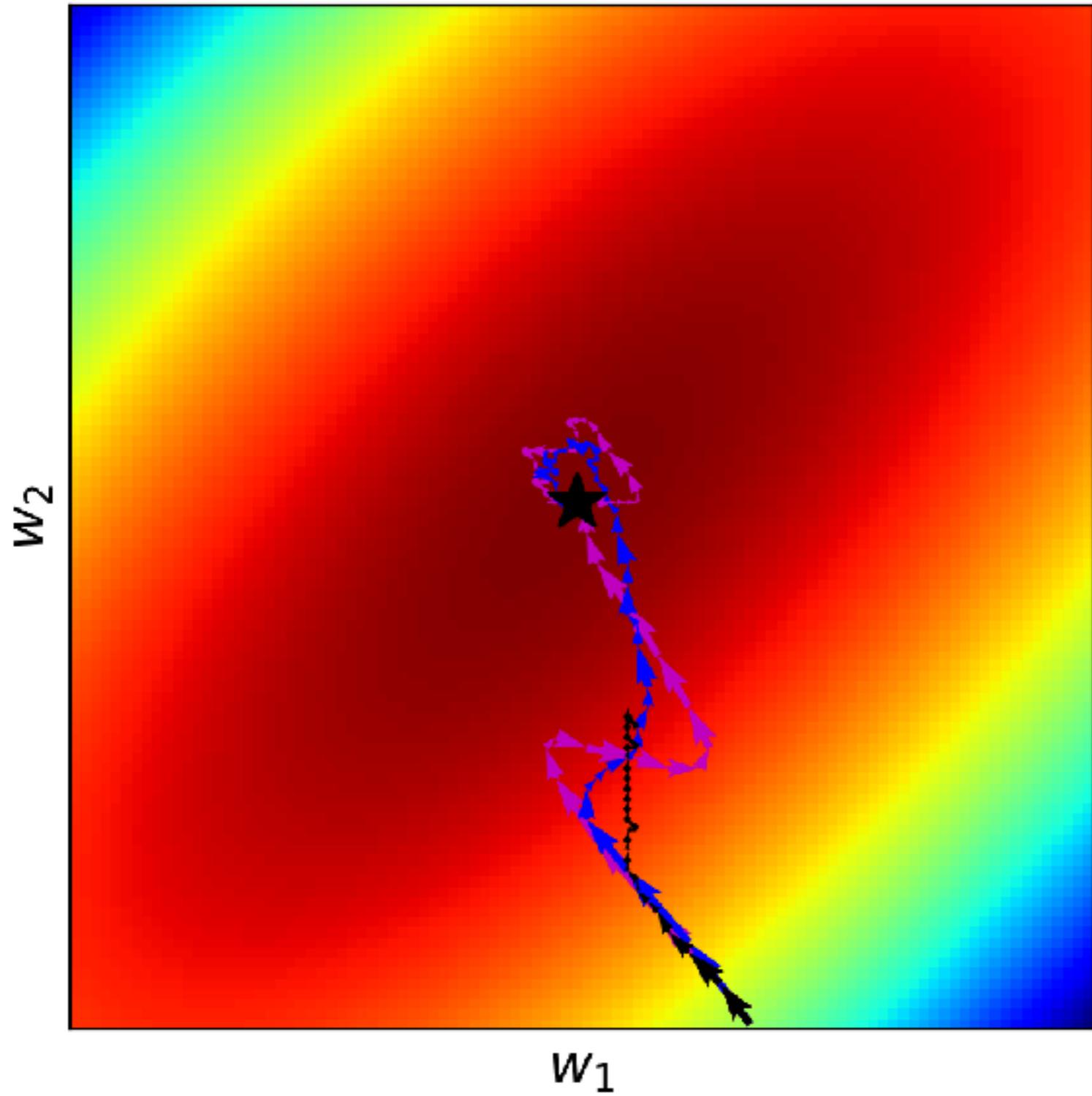
$$\hat{x}_{t+1} - \rho v_{t+1} = \hat{x}_t - \alpha \nabla f(\hat{x}_t)$$

$$\hat{x}_{t+1} = \hat{x}_t + \rho v_{t+1} - \alpha \nabla f(\hat{x}_t) = \hat{x}_t + \rho v_{t+1} - \alpha \nabla f(\hat{x}_t) \boxed{+ \rho v_t - \rho v_t}$$

$$\hat{x}_{t+1} = \hat{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)$$

$$\boxed{v_{t+1} = \rho v_t - \alpha \nabla f(\hat{x}_t)}$$
$$\boxed{\hat{x}_{t+1} = \hat{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)}$$

# SGD vs Momento vs Nesterov



- SGD: queda muy lejos del óptimo
- Momento: se desvía mucho
- Nesterov: la corrección del impulso es mejor

# Gradiente Adaptativo (AdaGrad)

- ❖ AdaGrad [Duchi et al., 2011]: Se lleva cuenta del gradiente al cuadrado en lugar de mantener la velocidad

$$\begin{aligned} g_0 &= 0 \\ g_{t+1} &= g_t + \nabla^2 f(x_t) \\ x_{t+1} &= x_t - \alpha \frac{\nabla f(x_t)}{\sqrt{g_{t+1}} + 1e^{-7}} \end{aligned}$$

$\frac{1}{\sqrt{g_{t+1}} + 1e^{-7}}$   Compensa las dimensiones  
(element wise) de mayor y  
menor variabilidad

# Gradiente Adaptativo (AdaGrad)

- ❖ AdaGrad [Duchi et al., 2011]: Se lleva cuenta del gradiente al cuadrado en lugar de mantener la velocidad

$$\begin{aligned} g_0 &= 0 \\ g_{t+1} &= g_t + \nabla^2 f(x_t) \\ x_{t+1} &= x_t - \alpha \frac{\nabla f(x_t)}{\sqrt{g_{t+1}} + 1e^{-7}} \end{aligned}$$

El gradiente crece sin parar, anulando el update

Bueno para funciones convexas

Malo para el resto

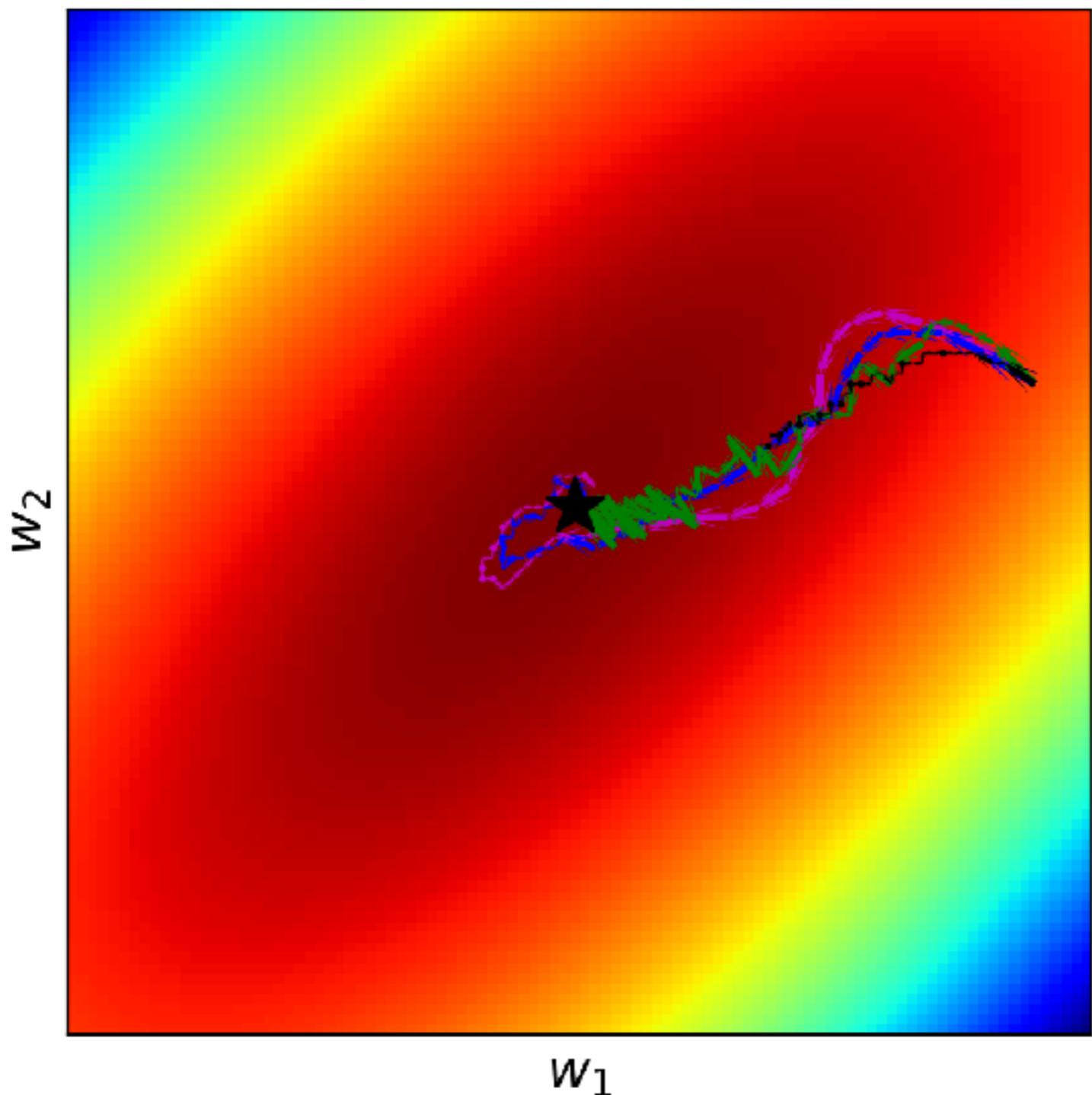
# RMSProp

- ❖ RMSProp [Tieleman and Hinton, 2012]: Agrega un termino de peso sobre el historial del gradiente

$$g_{t+1} = \gamma g_t + (1 - \gamma) \nabla^2 f(x_t)$$

$$\gamma \sim 0.9, 0.99$$

# RMSProp



- SGD: queda muy lejos del óptimo
- Momentum: se desvía mucho
- Nesterov: la corrección del impulso es mejor
- RMSProp

---

# Resumen

---

- ❖ **SGD + Momento:** tiene en cuenta la velocidad con la que viene. Esto permite escapar de puntos de ensilladura o de bajo gradiente.
- ❖ **Nesterov:** La corrección (por medio de la velocidad) la hace calculando el gradiente en la posición que indica la velocidad. Evita que se desvíe mucho.
- ❖ **Adagrad/RMSProp:** Agrega una escala basada en el gradiente al cuadrado para mitigar el problema de funciones mal condicionadas.

---

# Adaptive Moment Estimation

---

- ❖ Adam [Kingma and Ba, 2014] : Intenta integrar las bondades de los métodos vistos. Se lo puede pensar como agregar el termino de momento en el método RMSProp

# Adaptive Moment Estimation

- ❖ Adam [Kingma and Ba, 2014] : Se lo puede pensar como agregar el termino de momento en el método RMSProp

Momento

RMSProp

$$\begin{aligned} v_0 &= 0 & \rho, \gamma &\sim 0.9, 0.99 \\ g_0 &= 0 \\ v_{t+1} &= \rho v_t - (1 - \rho) \nabla f(x_t) \\ g_{t+1} &= \gamma g_t + (1 - \gamma) \nabla^2 f(x_t) \\ x_{t+1} &= x_t - \alpha \frac{v_{t+1}}{\sqrt{g_{t+1}} + 1e^{-7}} \end{aligned}$$

¿Qué pasa con el paso de actualización en la primer iteración ?

$g_{t+1} \sim 0$   $\longrightarrow$  Actualización muy grande al inicio

# Adaptive Moment Estimation

- ❖ Adam [Kingma and Ba, 2014] : Se lo puede pensar como agregar el termino de momento en el método RMSProp

Momento  $\longrightarrow$

$$\begin{aligned} v_0 &= 0 & \rho, \gamma &\sim 0.9, 0.99 \\ g_0 &= 0 \end{aligned}$$

RMSProp  $\longrightarrow$

$$\begin{aligned} v_{t+1} &= \rho v_t - (1 - \rho) \nabla f(x_t) \\ g_{t+1} &= \gamma g_t + (1 - \gamma) \nabla^2 f(x_t) \end{aligned}$$

Bias  $\longrightarrow$

$$\hat{v} = \frac{v_{t+1}}{1 - \rho^{t+1}} \quad \hat{g} = \frac{g_{t+1}}{1 - \gamma^{t+1}}$$

$$x_{t+1} = x_t - \alpha \frac{\hat{v}}{\sqrt{\hat{g}} + 1e^{-7}}$$

# Adaptative Learning Rate Method

- ❖ Adadelta [Zeiler 2012]: Es una extensión de método Adagrad/RMSProp donde se reduce de forma monótona la tasa de aprendizaje

No tiene  
learning rate

$$g_0 = 0 \quad \Delta_o = 0 \quad \rho \sim 0.9, 0.99$$

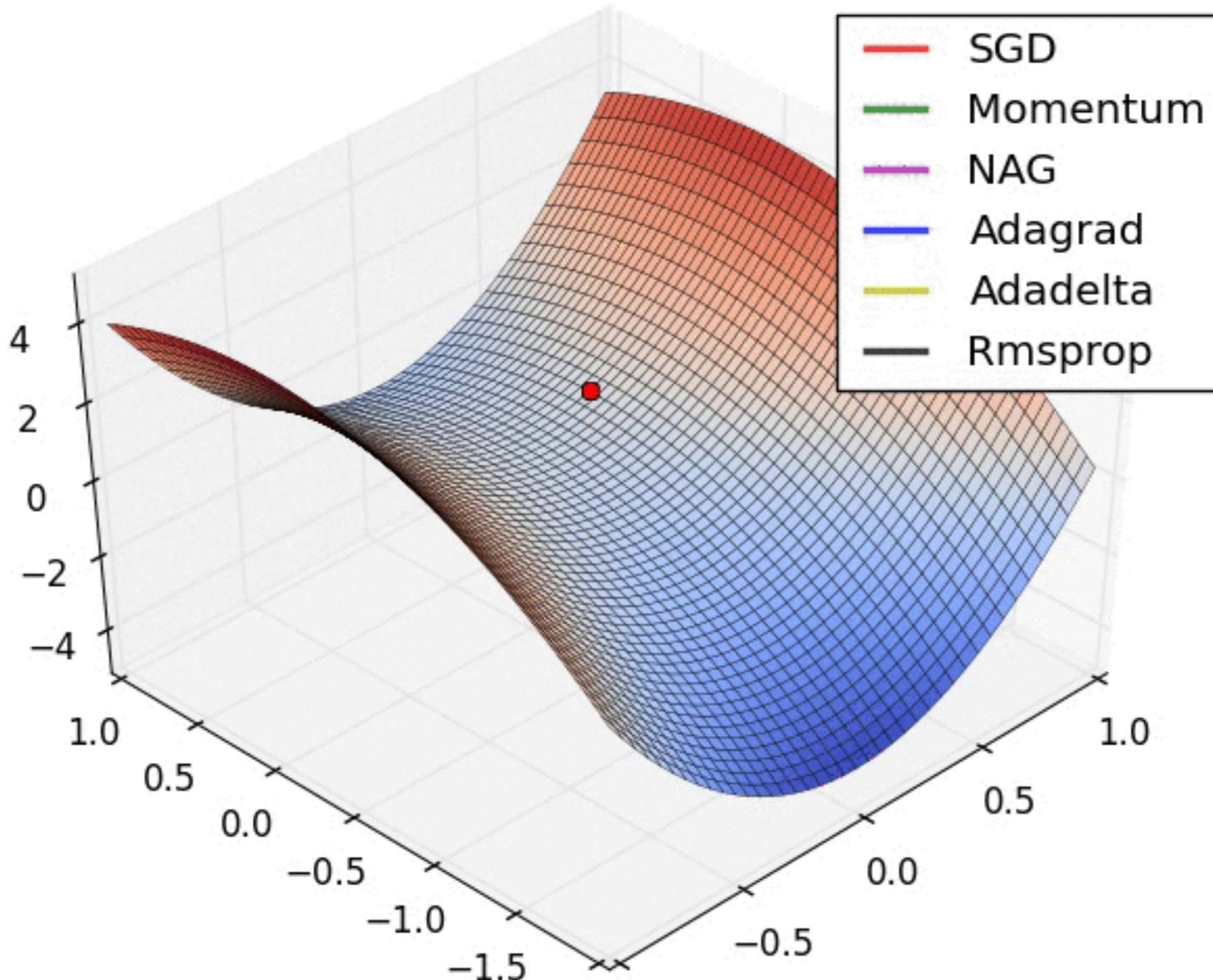
$$g_{t+1} = \rho g_t + (1 - \rho) \nabla^2 f(x_t)$$

$$u = -\frac{\sqrt{\Delta_t + \epsilon}}{\sqrt{g_t + \epsilon}}$$

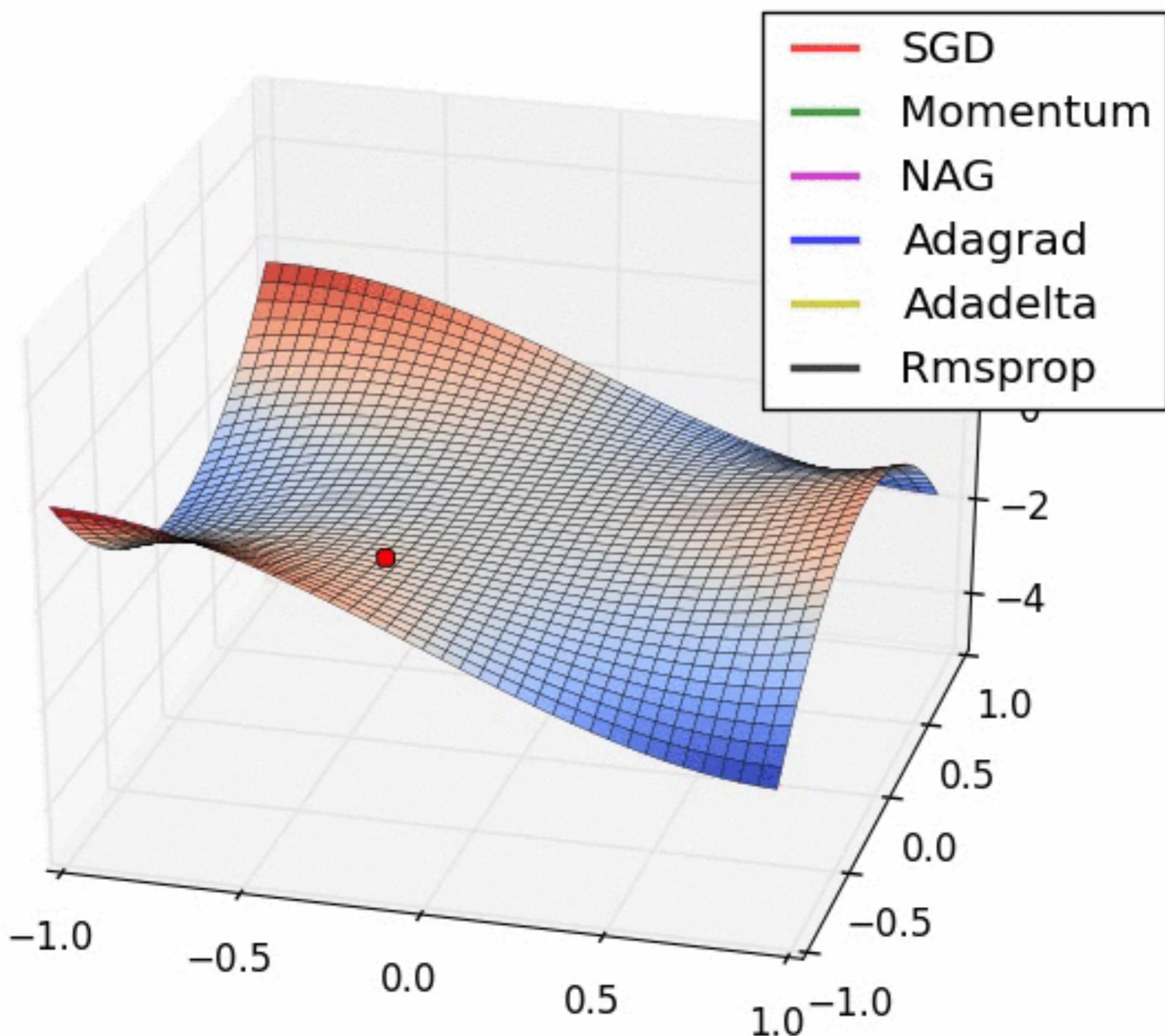
$$\Delta_{t+1} = \rho \Delta_t + (1 - \rho) u^2$$

$$x_{t+1} = x_t + u$$

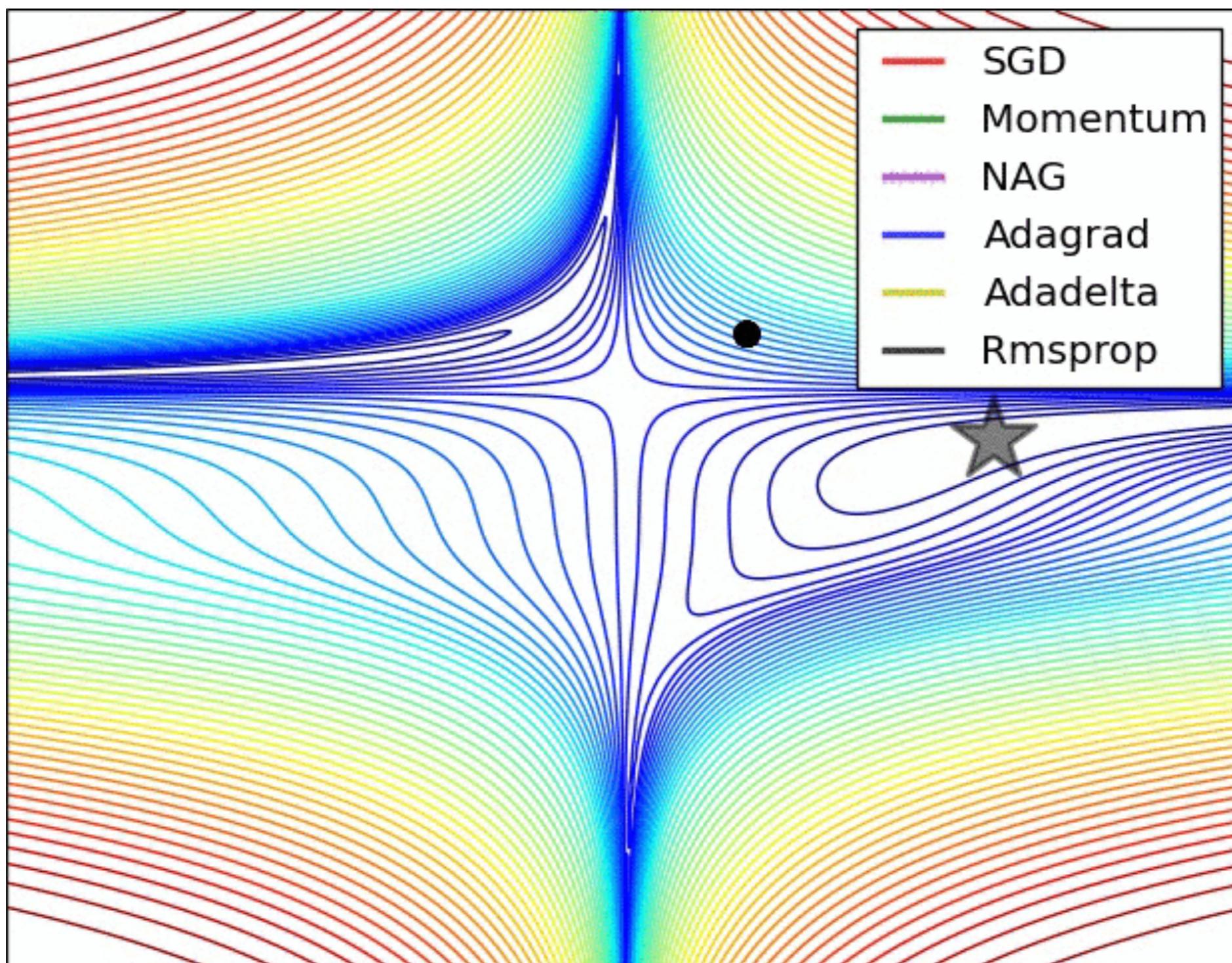
# Optimizadores



# Optimizadores



# Optimizadores

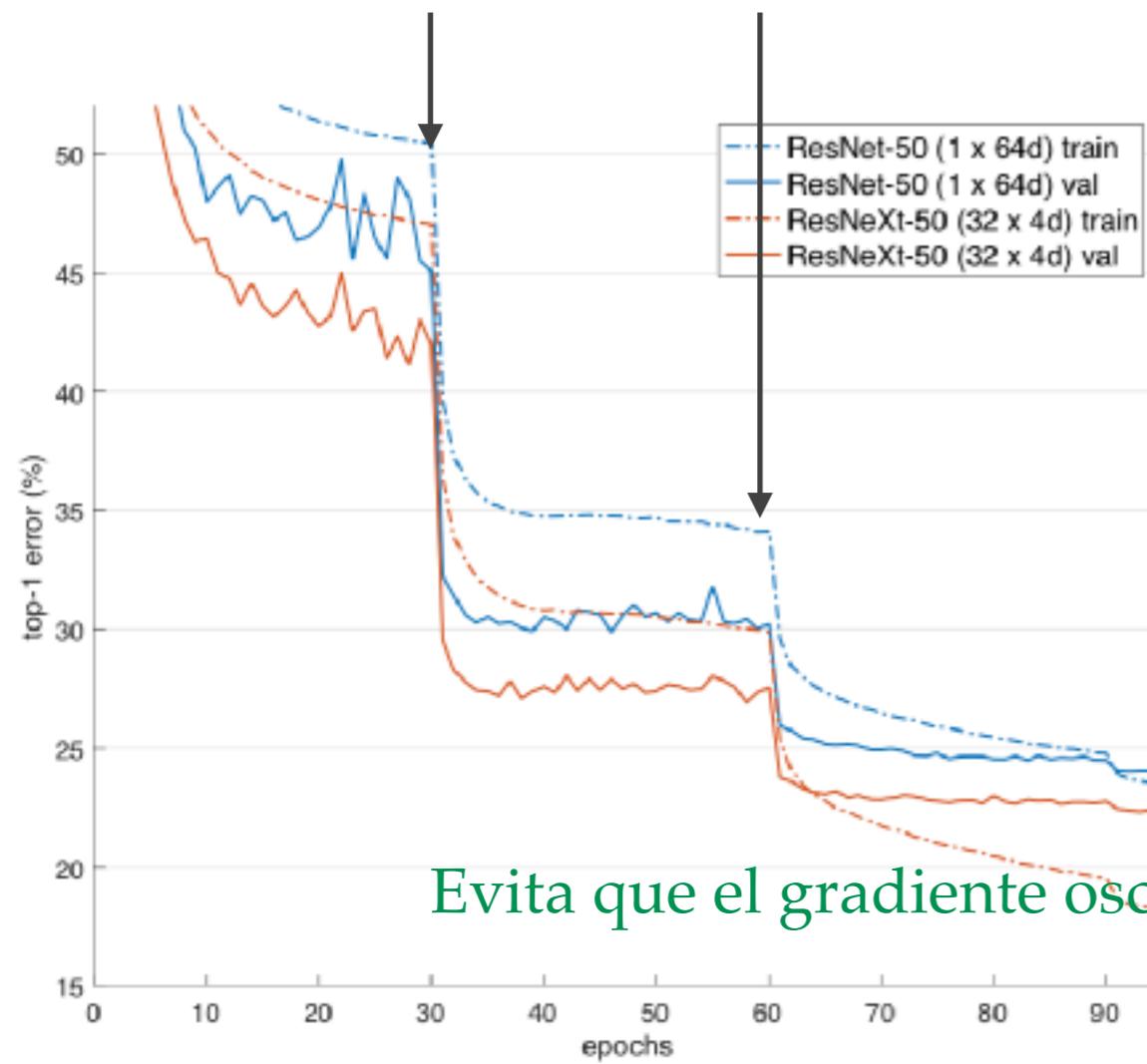


# Optimizadores

- ❖ Adadelta: Ajusta la tasa de aprendizaje de forma automática.
- ❖ SGD, SGD + Momento, Adagrad, RMSProp y Adam: Todos tienen como hiperparámetro la tasa d aprendizaje.
  - ❖ Variantes sobre la tasa de aprendizaje:
    - ❖ Decaimiento según la cantidad de pasos, por ejemplo, a la mitad.
    - ❖ Exponencial  $\alpha = \alpha_0 e^{-k t}$   
 $k$ : tasa de decaimiento
    - ❖ Decaimiento  $1/t$   $\alpha = \frac{\alpha_0}{1 + k t}$   
nuevo hiperparámetro

# Decaimiento de la tasa de aprendizaje

Decaimiento de la tasa de aprendizaje



Decaimiento de la tasa de aprendizaje

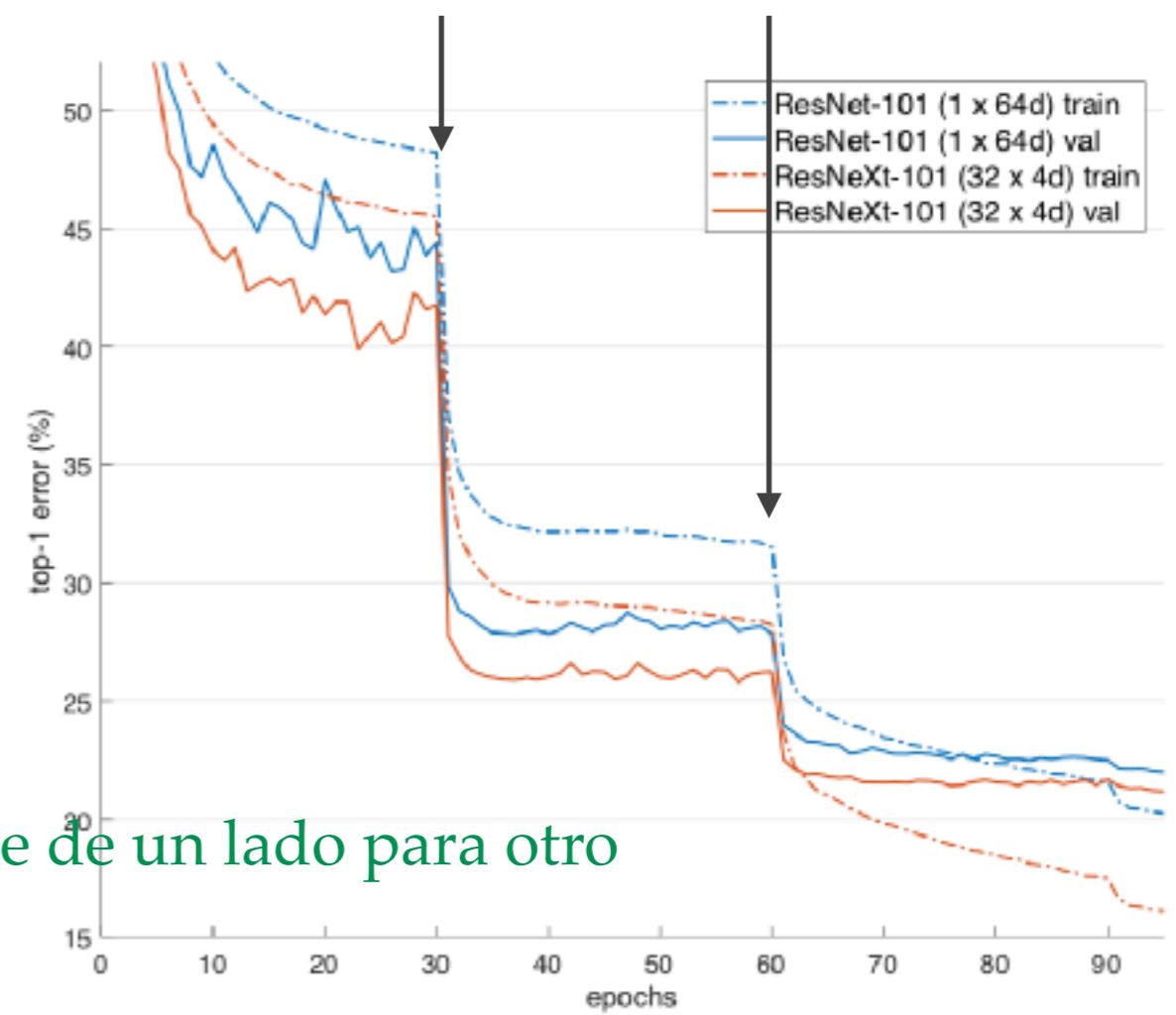
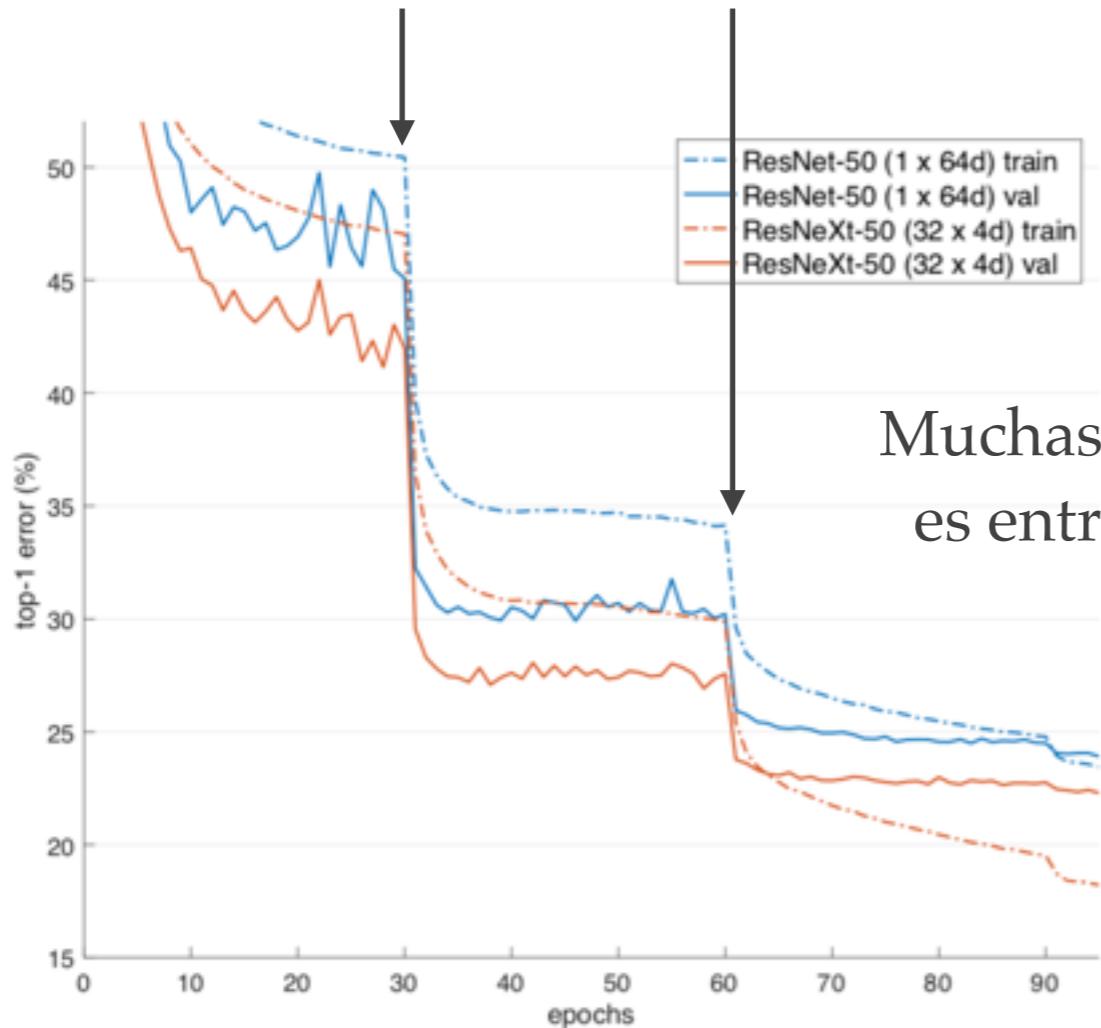


Figure 5. Training curves on ImageNet-1K. (**Left**): ResNet/ResNeXt-50 with preserved complexity (~4.1 billion FLOPs, ~25 million parameters); (**Right**): ResNet/ResNeXt-101 with preserved complexity (~7.8 billion FLOPs, ~44 million parameters).

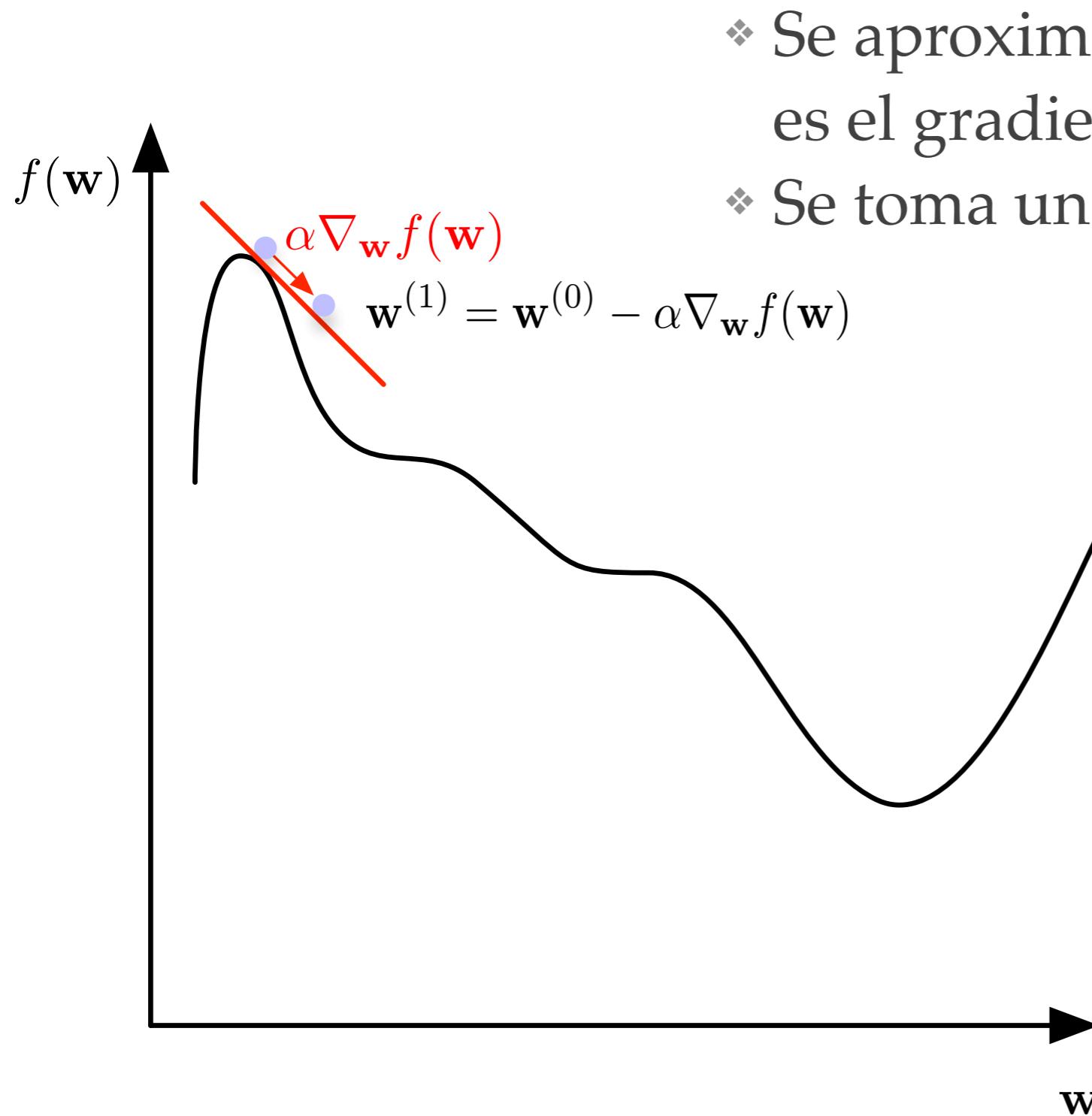
# Decaimiento de la tasa de aprendizaje

## Decaimiento de la tasa de aprendizaje



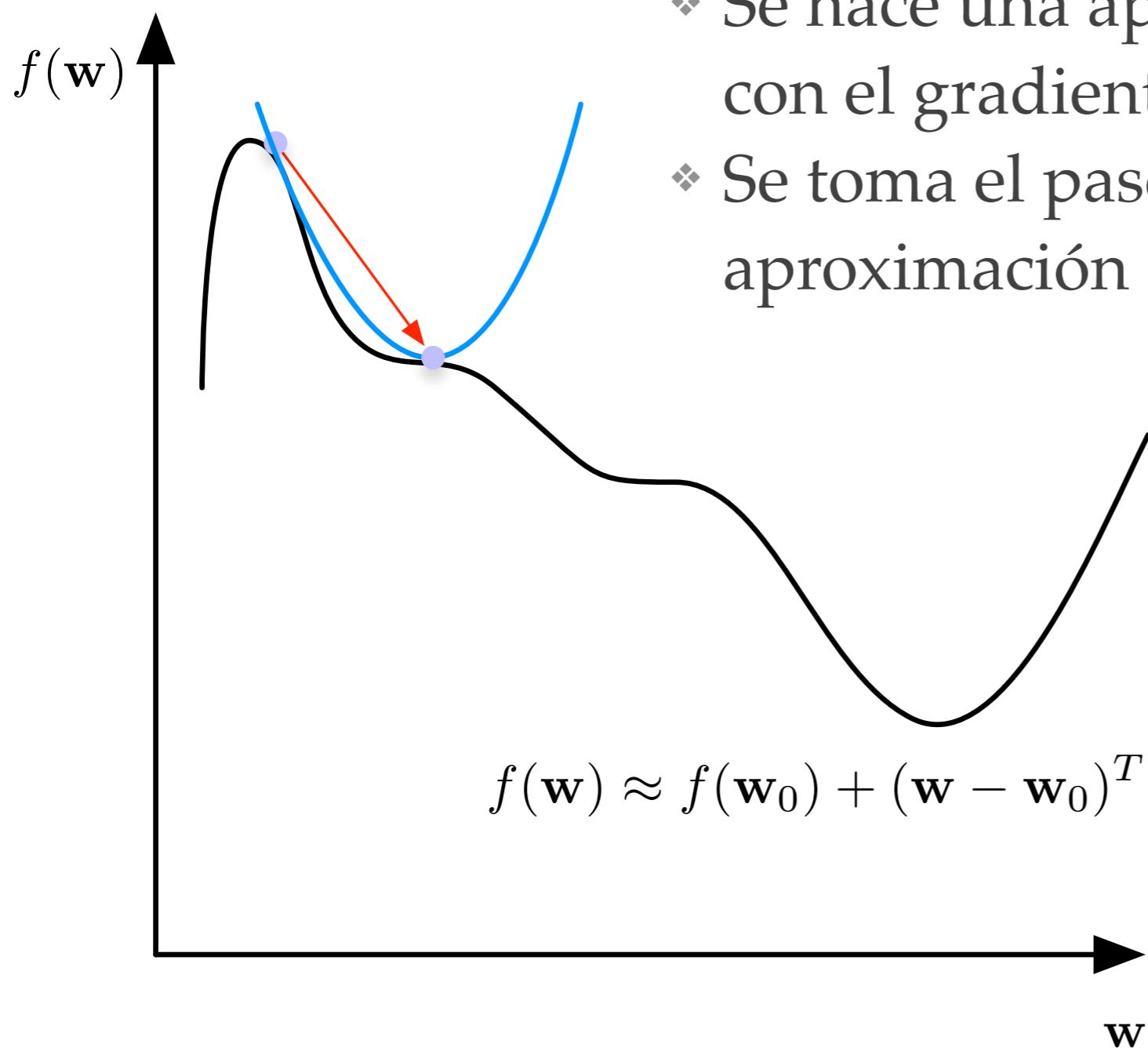
Muchas veces la forma de entrenar esta tasa de decaimiento es entrenar sin decaimiento y ver donde conviene ponerlo (cuando oscila mucho) o con que intensidad

# Optimizadores de 1 orden



- ❖ Se aproxima una recta cuya pendiente es el gradiente
- ❖ Se toma un paso en esa recta

# Optimizadores de 2 orden



- ❖ Se hace una aproximación cuadrática con el gradiente y la matriz Hessiana
- ❖ Se toma el paso que minimiza la aproximación

No existe la tasa de aprendizaje

$$H \sim O(N^2) \quad H^{-1} \sim O(N^3)$$

Deep learning millones de parámetros

Expansión de Taylor de 2 orden

$$\tilde{\mathbf{w}} = \mathbf{w}_0 - H^{-1} \nabla_{\mathbf{w}} f(\mathbf{w}_0)$$

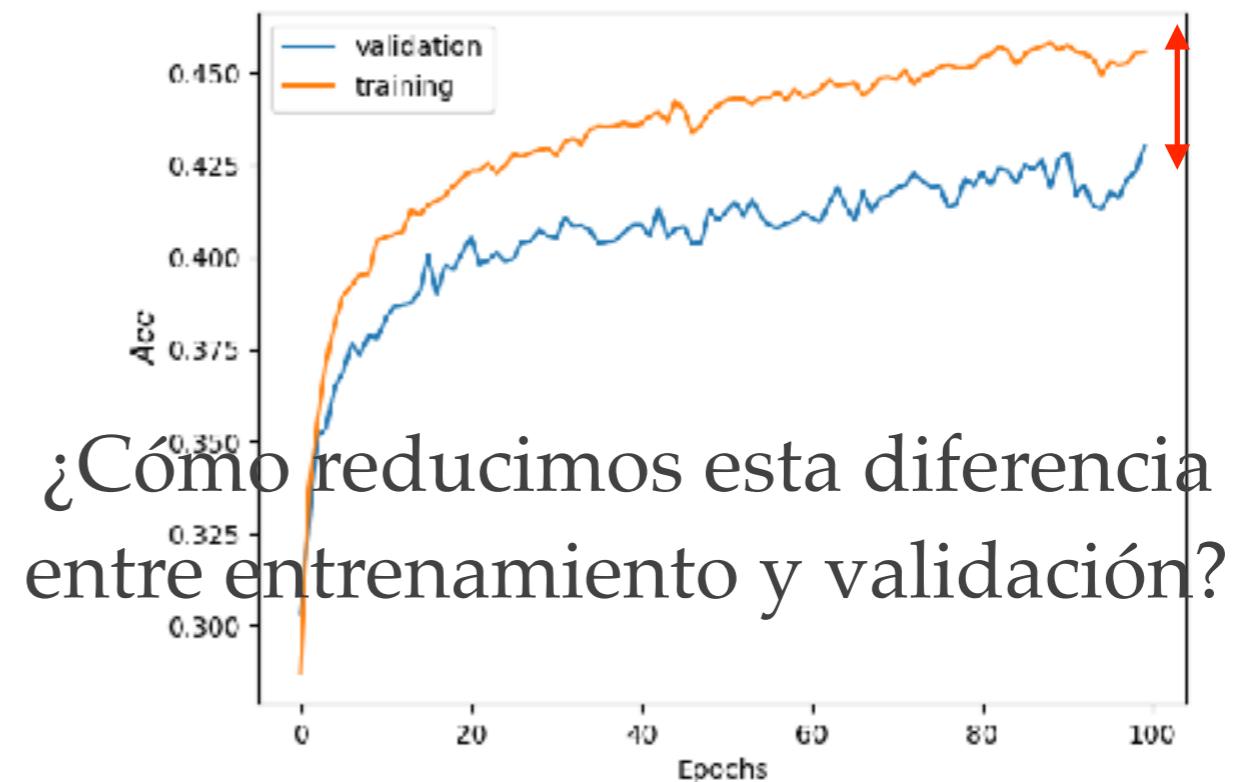
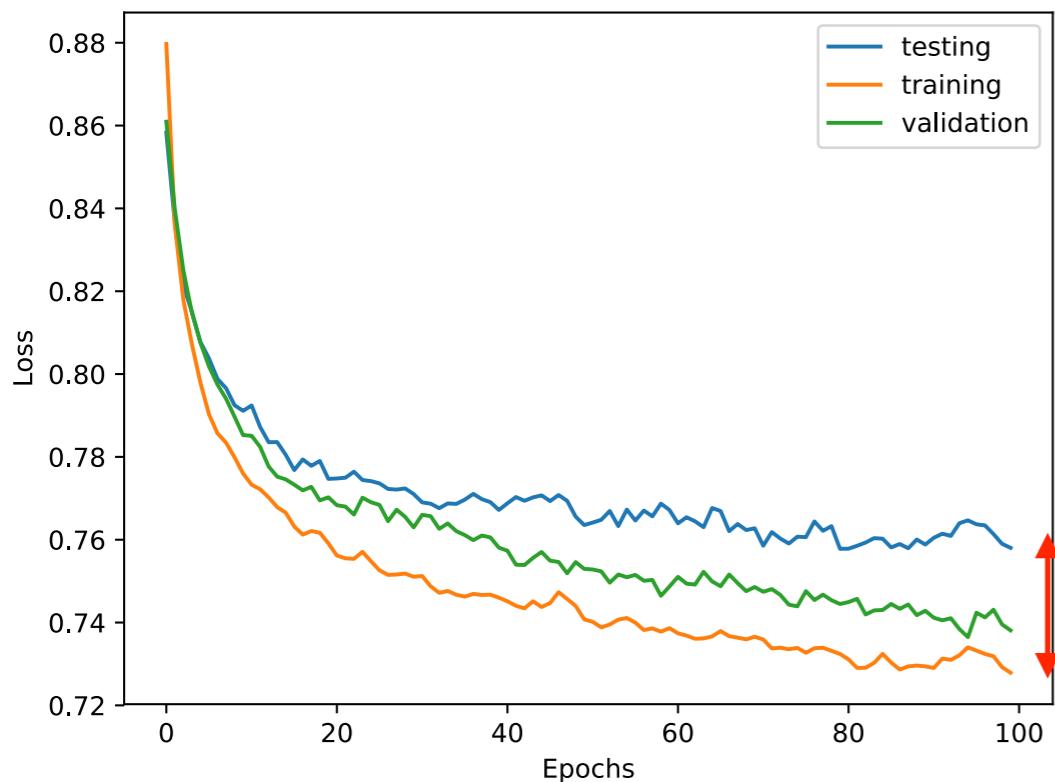
# Optimizadores de 2 orden

- ❖ Métodos Quasi-Newton (**BFGS** muy popular): No invierten la matriz Hessiana, la aproximan  $O(N^2)$
- ❖ **L-BFGS** (Limited memory BFGS): No almacenan completamente la matriz inversa de la Hessiana
  - ❖ Anda muy bien en modo determinista (todo el batch de datos)
  - ❖ No anda muy bien con mini-batch (área activa de investigación)
- ❖ En la práctica **Adam** es una muy buena opción para casi todo los problemas de NN, pero si se puede, **L-BFGS** es una muy buena opción (no usar mini-batch)

# Generalización

# Generalización

- ❖ Hasta ahora nos preocupamos por reducir el error con los datos de entrenamiento
- ❖ Lo que realmente buscamos es aumentar el poder de generalización, es decir, reducir el error en la validación



---

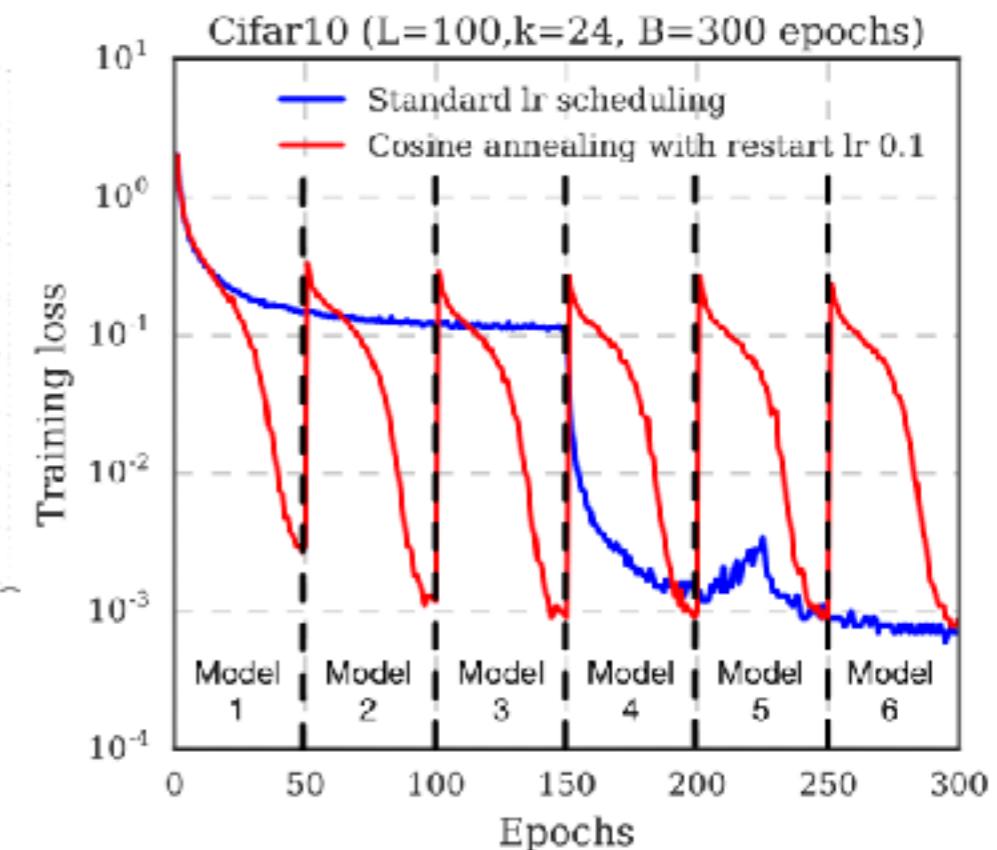
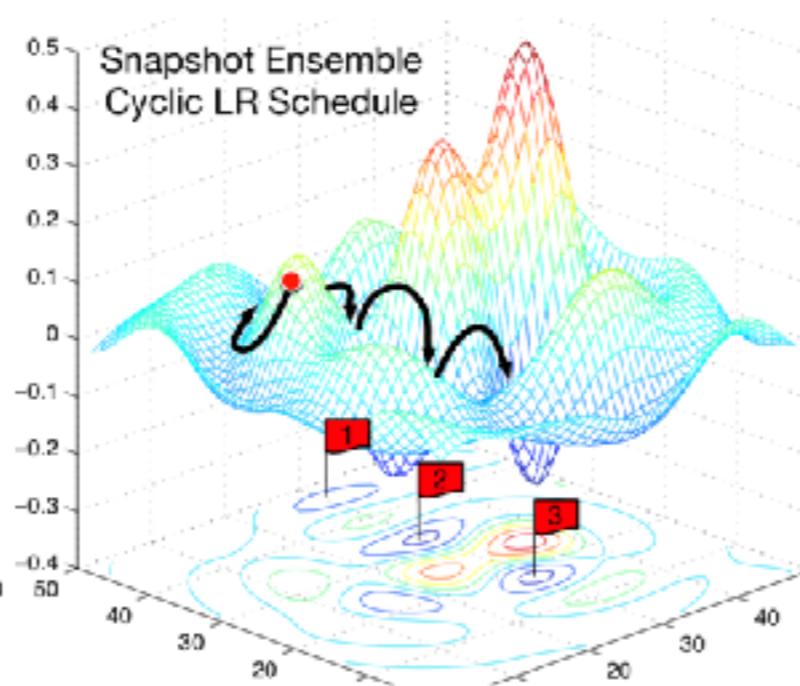
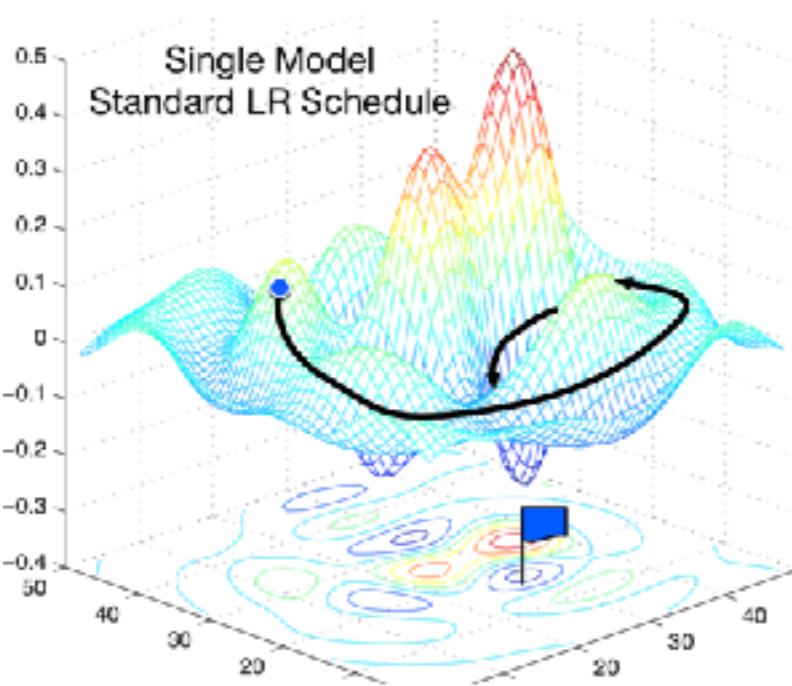
# Model Ensembles

---

- ❖ Idea muy simple:
  - ❖ Entrenar varios modelos de forma independiente, por ejemplo, 10 modelos con diferente inicialización de pesos.
  - ❖ Evaluar de forma independiente, y luego promediar.
- ❖ Con este proceso se gana un 2% de mejora

# Model Ensembles

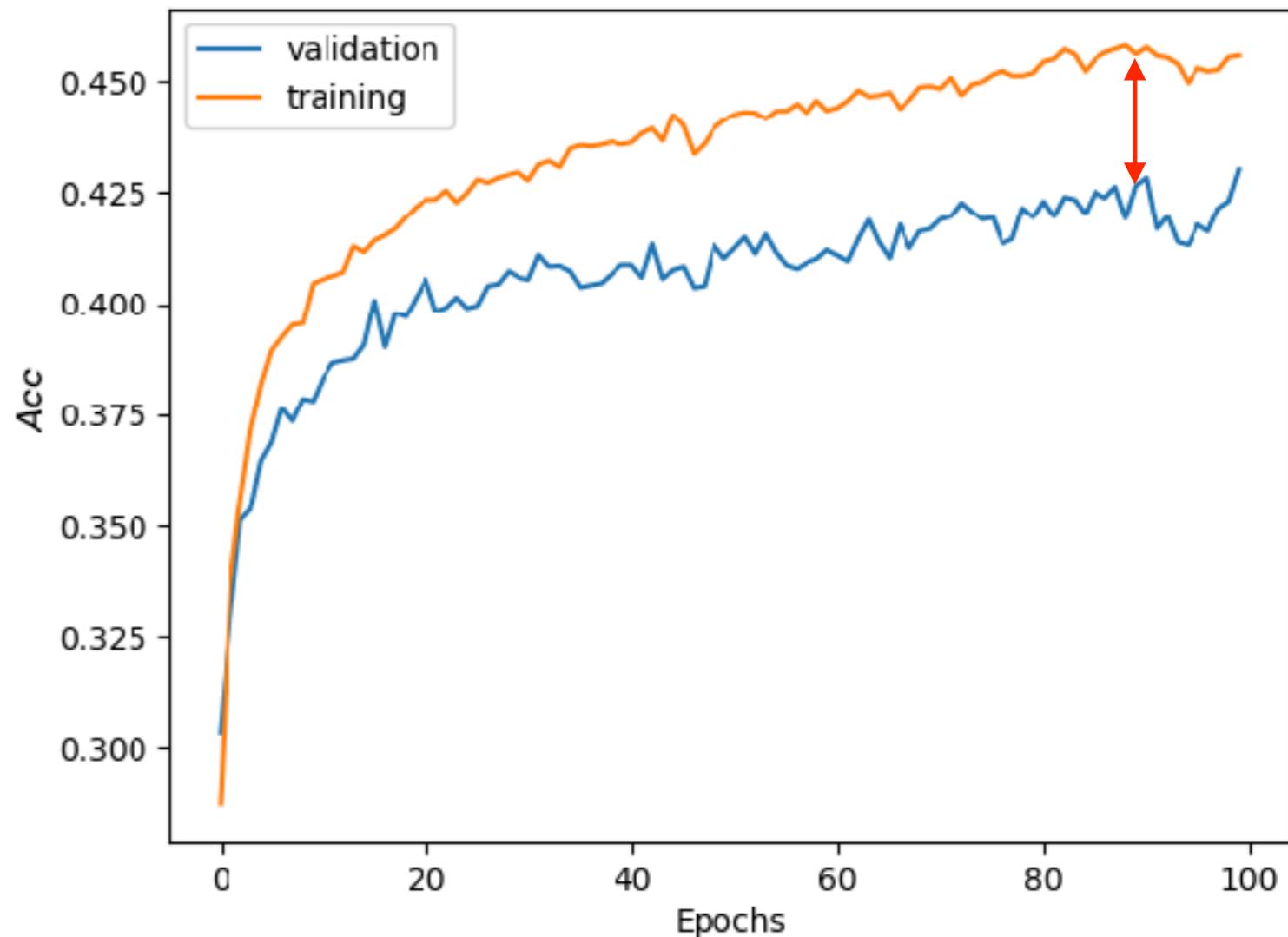
- ❖ En lugar de entrenar completamente muchos modelos, se puede tomar snapshots de un sólo modelo durante el entrenamiento [Huang et al., 2017]



Además, combinan este enfoque con un esquema de modificación del lr para que los snapshot vayan convergiendo a diferentes mínimos

# Regularización

- ❖ La regularización nos permite mejorar la generalización
  - ❖ Agregando un término de regularización
  - ❖ Dropout
  - ❖ Batch Normalization
  - ❖ Data Augmentation
  - ❖ .....



# Termino de Regularización

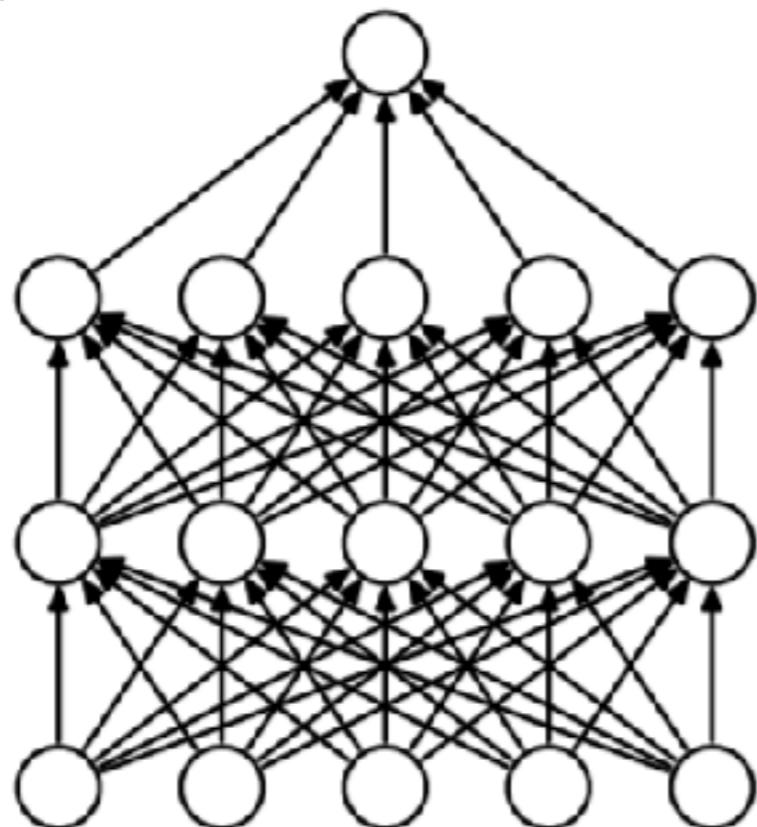
Agregando un termino de regularización

$$\text{MSE} + \lambda R(W) = L(y') = \frac{1}{N} \sum_i ||y'^{(i)}||_2^2 + \boxed{\lambda R(W)}$$

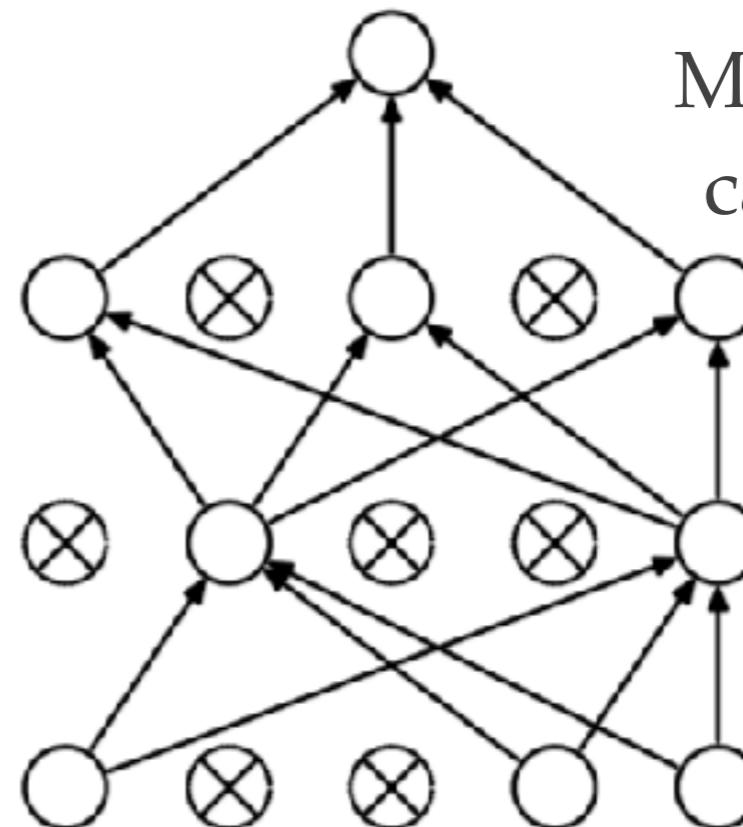
- ❖ L2     $R(W) = \sum_{k,l} W_{k,l}^2$               0.5 \* L2
- ❖ L1     $R(W) = \sum_{k,l} |W_{k,l}|$
- ❖ Elastic net (L1 + L2)     $R(W) = \sum_{k,l} \beta W_{k,l}^2 + |W_{k,l}|$

# Dropout

- ❖ Se anula la activación de algunas neuronas en la etapa feed-forward de forma aleatoria. La prob. de drops es un hiperparámetros (un comienzo puede ser 0.5)



(a) Standard Neural Net



(b) After applying dropout.

Más común en  
capas densas

---

# Dropout

---

- ❖ Fuerza a la red a tener una representación redundante
- ❖ Previene la co-adaptación de las características:
  - ❖ Se cancelan algunas entre ellas
  - ❖ Muy similares
- ❖ Otra visión: Se convierte en “ensemble” de modelos a gran escala ya que para cada etapa feed-forward tenemos un modelo diferente que converge a mínimos locales. Luego al utilizar todas las neuronas es como promediar todos los modelos

# Dropout

- ❖ Analicemos que pasa durante el entrenamiento y veamos qué corrección es necesaria hacer en la etapa de testing
  - ❖ Durante el entrenamiento nuestra salida es aleatoria ya que se descartan neuronas de forma aleatoria

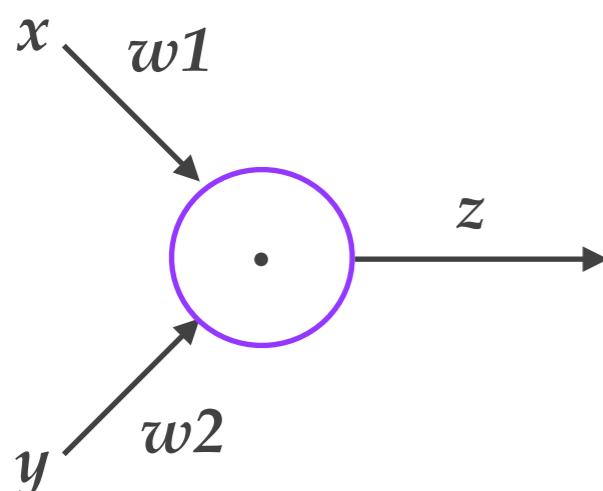
Para eliminar la aleatoriedad podemos tomar el promedio

# Imposible de computar

$$y = f_{\mathbf{w}}(x) = \text{E}(f_{\mathbf{w}}(x, z)) = \boxed{\int P(z)f_{\mathbf{w}}(x, z)dz}$$

# Dropout

- ❖ Supongamos tenemos una sola neurona de la siguiente forma y aplicamos una prob. de drops de 0.5



Multiplica por la  
probabilidad de drops

- ❖ En la etapa de testing

$$E(z) = w_1 x + w_2 y$$

- ❖ En la etapa de entrenamiento

$$E(z) = \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + w_2 0)$$

$$+ \frac{1}{4}(w_1 0 + w_2 y) + \frac{1}{4}(w_1 0 + w_2 0)$$

$$E(z) = \boxed{\frac{1}{2}}(w_1 x + w_2 y)$$

En testing hay que  
multiplicar por la  
probabilidad de  
dropout

# Inverted dropout

- ❖ La idea es escalar en training para no tener que hacer modificación en testing

Drops !

```
1   H = np.maximum(0, X.dot(W) + bias)
2   U = (np.random.rand(*H.shape) < p) / p
3   H *= U
```

Hacemos la corrección en el entrenamiento

Neuronas a desactivar

- ❖ En la práctica se utiliza Inverted Dropout
- ❖ Se tarda más en entrenar, ya que optimiza una parte del modelo, pero generaliza mejor

---

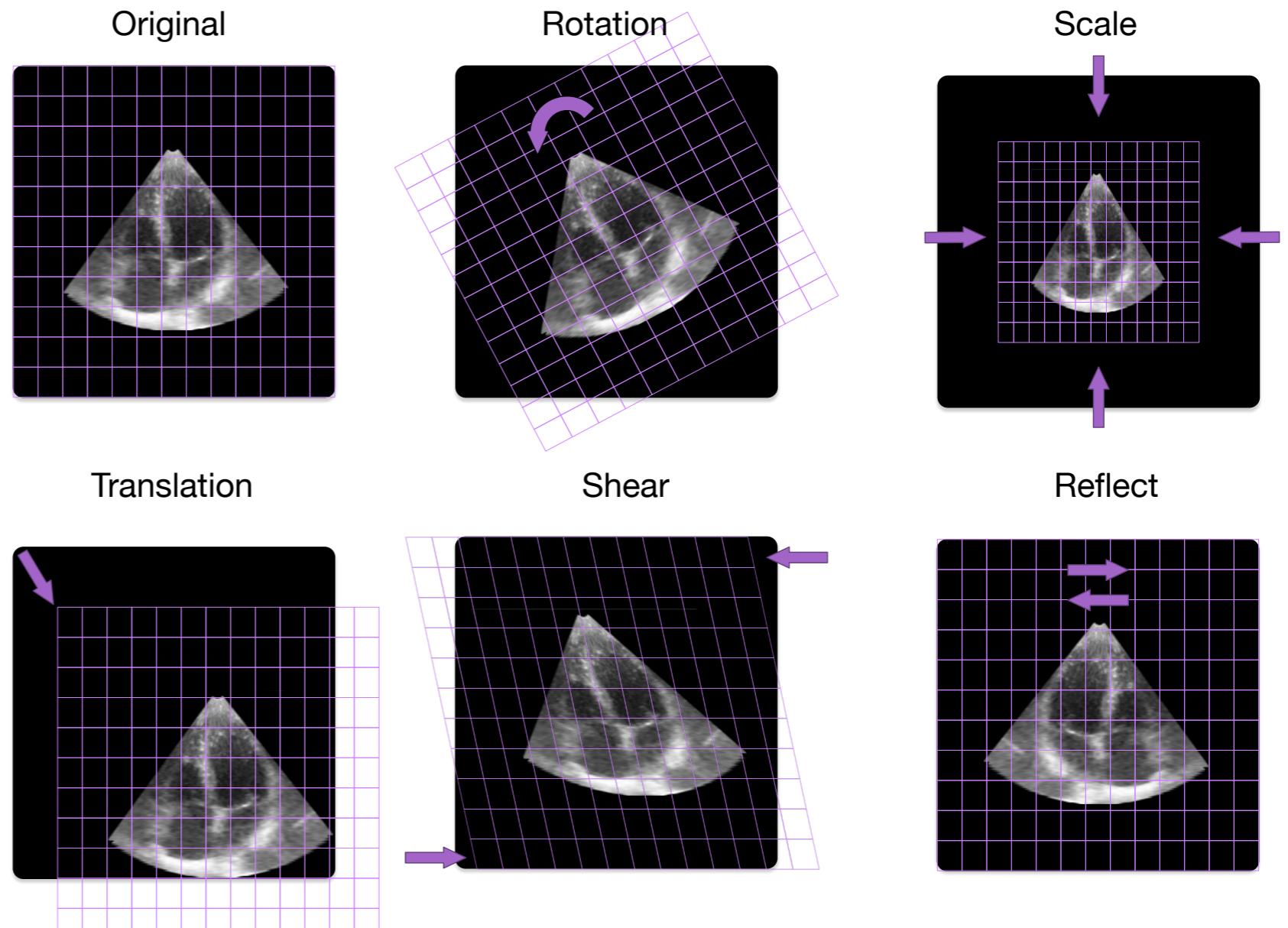
# Batch Normalization

---

- ❖ La estrategia de batch normalization puede verse como una regularización ya que al momento de normalizar, como se hace en un mini-batch, introduce un grado de aleatoriedad

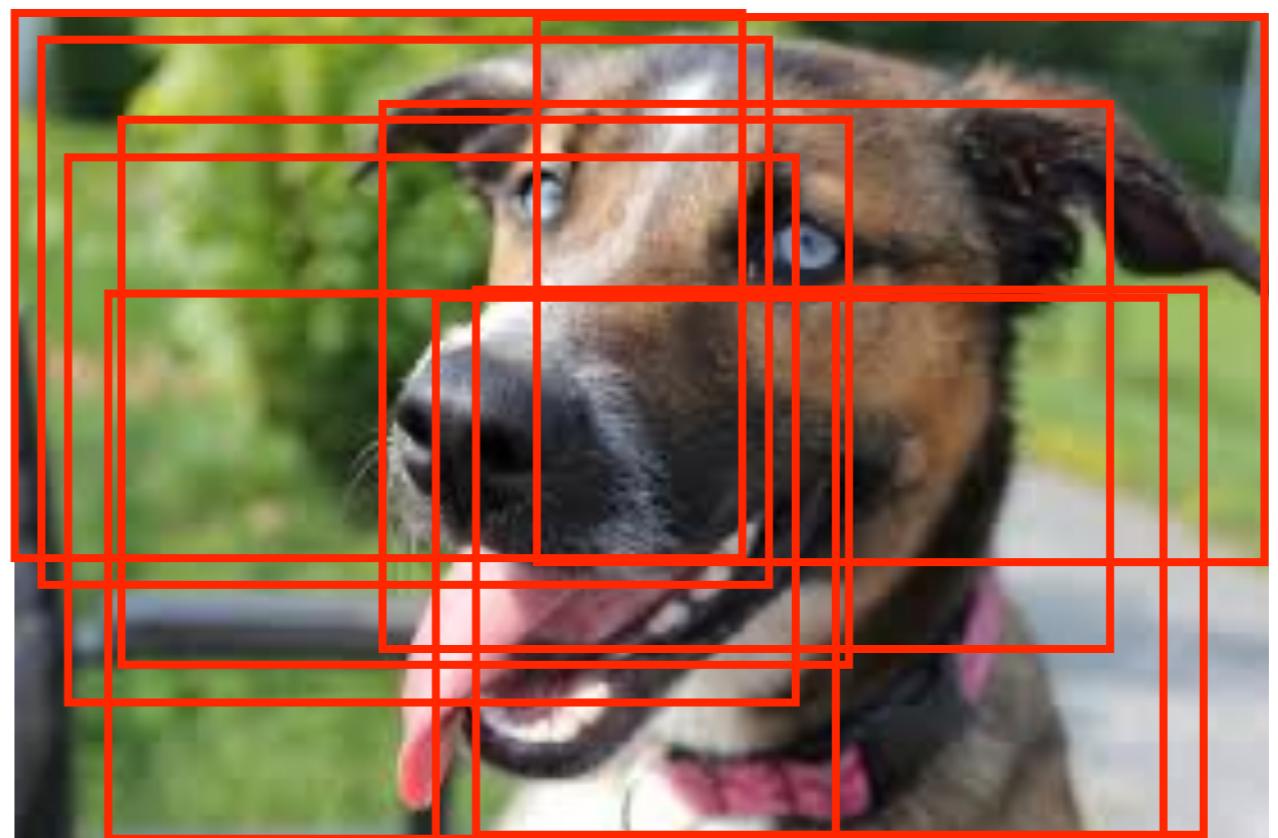
# Aumentación de datos

- ❖ La aumentación de datos o “data augmentation” agrega un tipo de regularización



# Aumentación de datos

- ❖ Cortar + Aumentar (crop & scale): se pueden cortar partes de la imagen y aumentarlas al tamaño de entrada de la red.
- ❖ La idea es que partes de un objeto siguen siendo el mismo objeto



# Aumentación de datos

- ❖ Color Jitter

Simple: se transforma el contraste y brillo de forma aleatoria

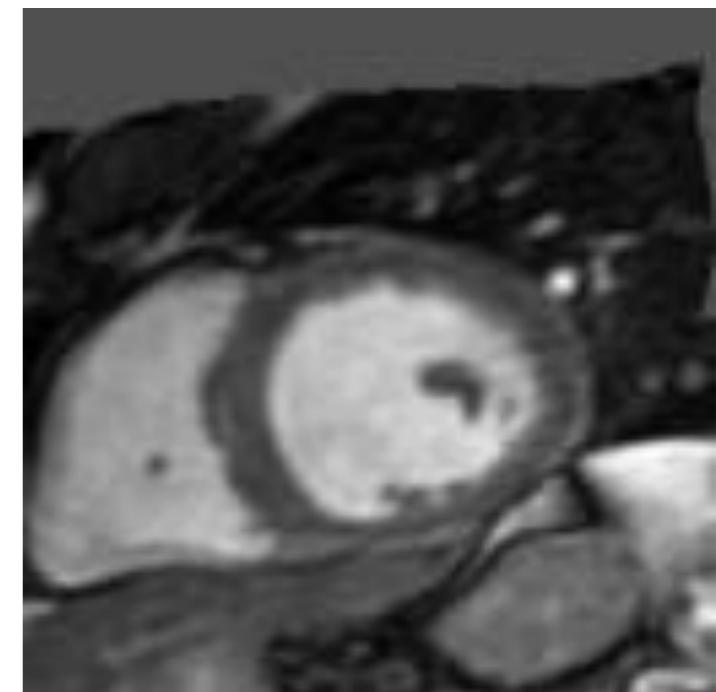
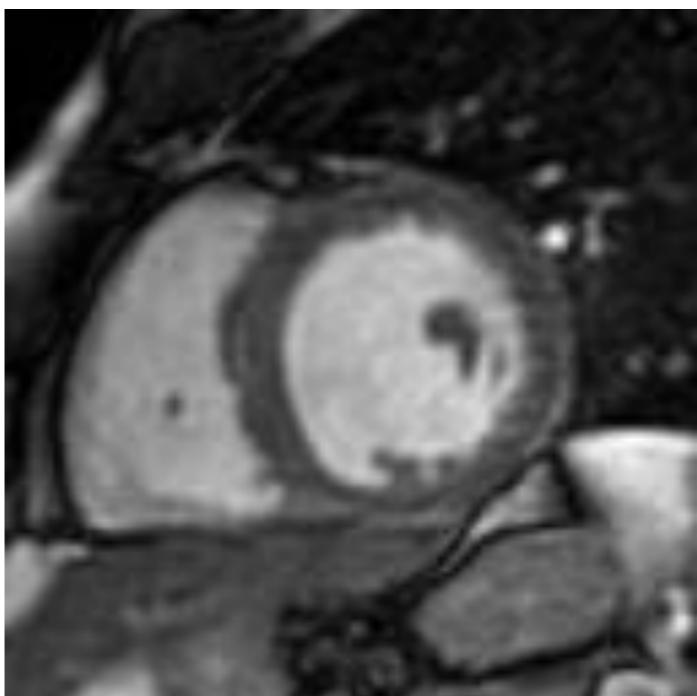


- Complejo:

- ❖ PCA por canal de colores a todo el conj. de entrenamiento
- ❖ Se toma un offset en la dirección de la componente principal
- ❖ Se agrega este offset a todos los pixeles de una imagen de entrenamiento

# Aumentación de datos

- ❖ Mezcla aleatoria de :
  - ❖ Traslación
  - ❖ Rotación
  - ❖ Shearing
  - ❖ Scale
  - ❖ Deformaciones no rígidas



---

# Transferencia de aprendizaje

---

“The ability of a system to recognize and apply knowledge and skills learned in previous tasks to novel tasks.”

Defense Advanced Research Projects Agency (DARPA)’s Information Processing Technology Office

# Transferencia de conocimiento

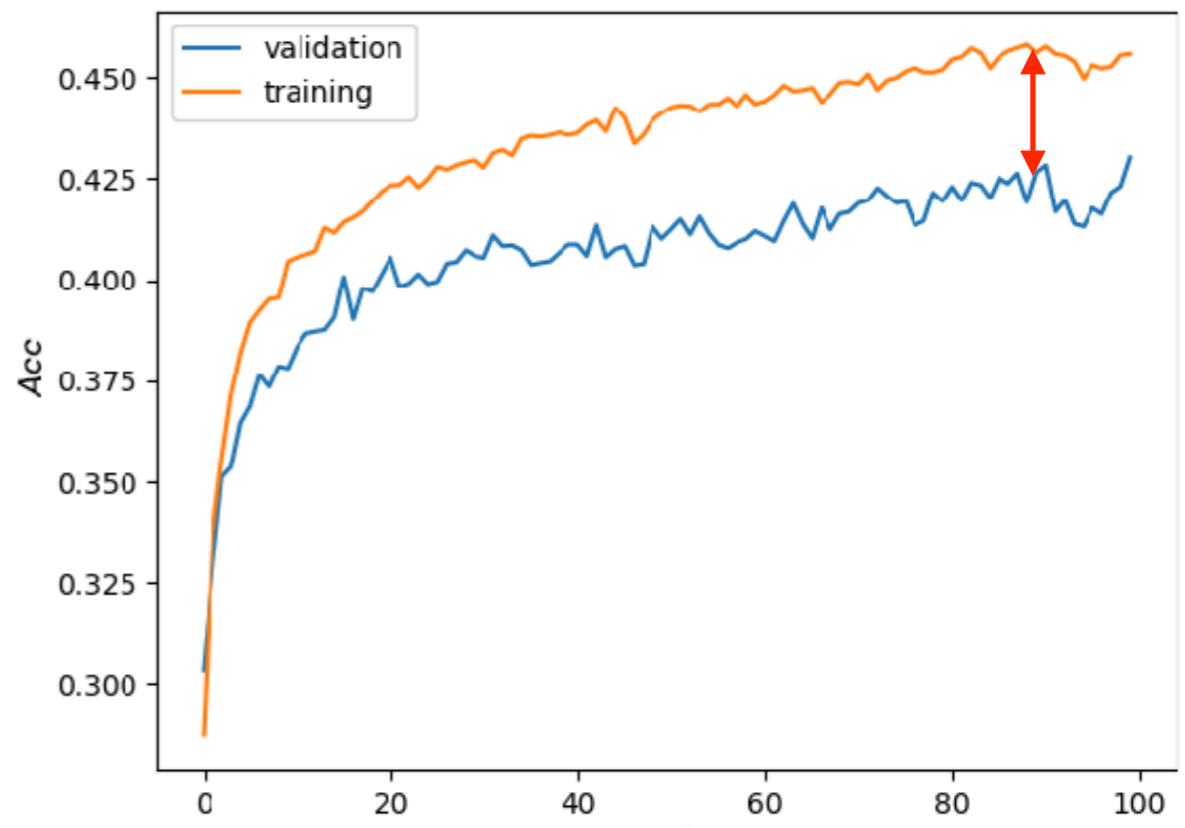
- ❖ La transferencia de aprendizaje (transfer learning) es otra forma de mejorar nuestra generalización y reducir la diferencia entre validación y entrenamiento

Muchas veces es útil ya que no se tiene una gran cantidad de datos

¿En qué casos puede ser realmente útil?

Es un área de investigación activa, ya que no es trivial extraer sólo las cosas que se necesitan para nuestro problema en particular

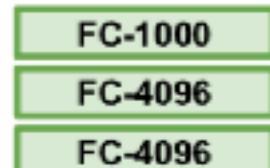
M. Raghu, C. Zhang, J. M. Kleinberg, and S. Bengio,  
“Transfusion: Understanding transfer learning with applications  
to medical imaging,” arXiv preprint arXiv:1902.07208, 2019.



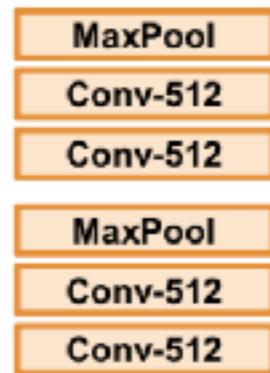
Yosinski, Jason, et al. "How transferable are features in deep neural networks?." *Advances in neural information processing systems*. 2014.

# Transferencia de aprendizaje en CNNs

Muchos datos para un problema dado  
(Imagenet 10k clases)

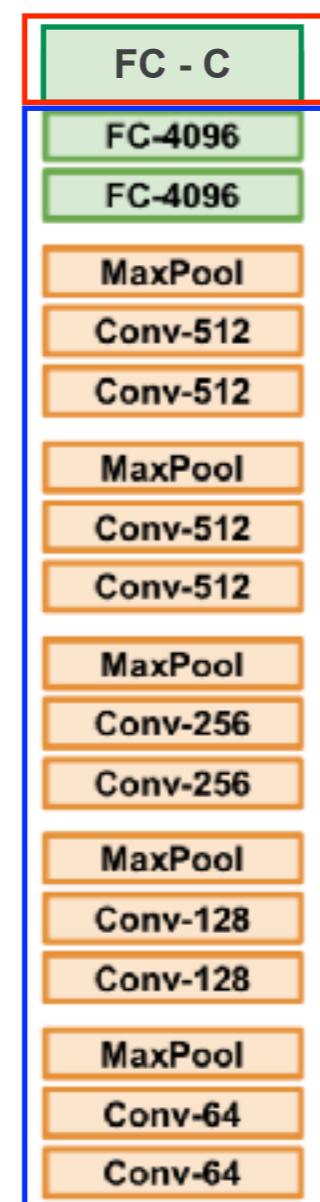


Entrenamos



Fijos, no entrenamos

Pocos datos (C clases)



Entrenamos  
Lr bajo,  
1/10 del  
original  
buen punto  
de partida

Fijos, no  
entrenamos

Más datos (C clases)



Fijos, no  
entrenamos

Nueva entrada



Entrenamos

Entrenamos

Entrenamos

# Transferencia de aprendizaje en CNNs

