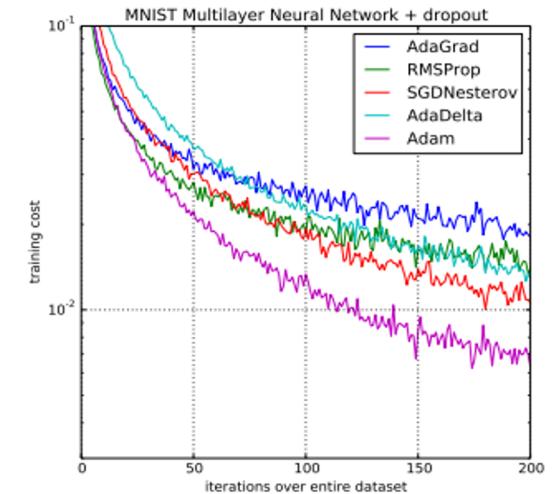
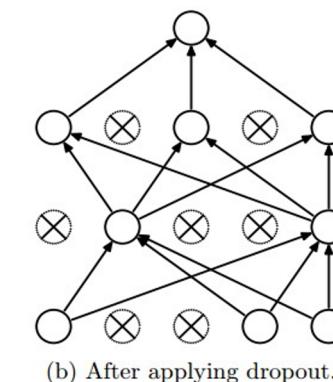
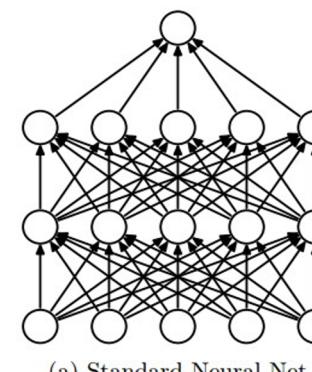


## Clase 4

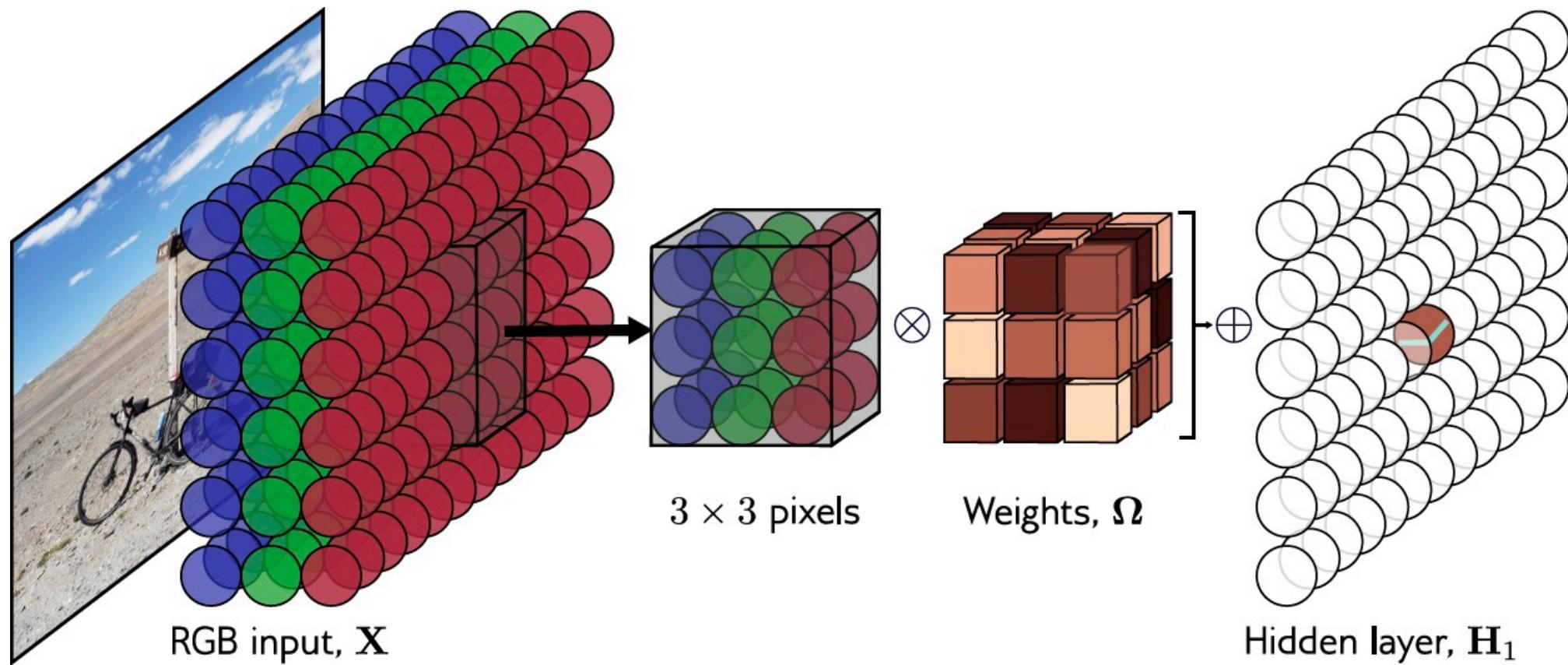
# Entrenando una red neuronal

Enzo Ferrante

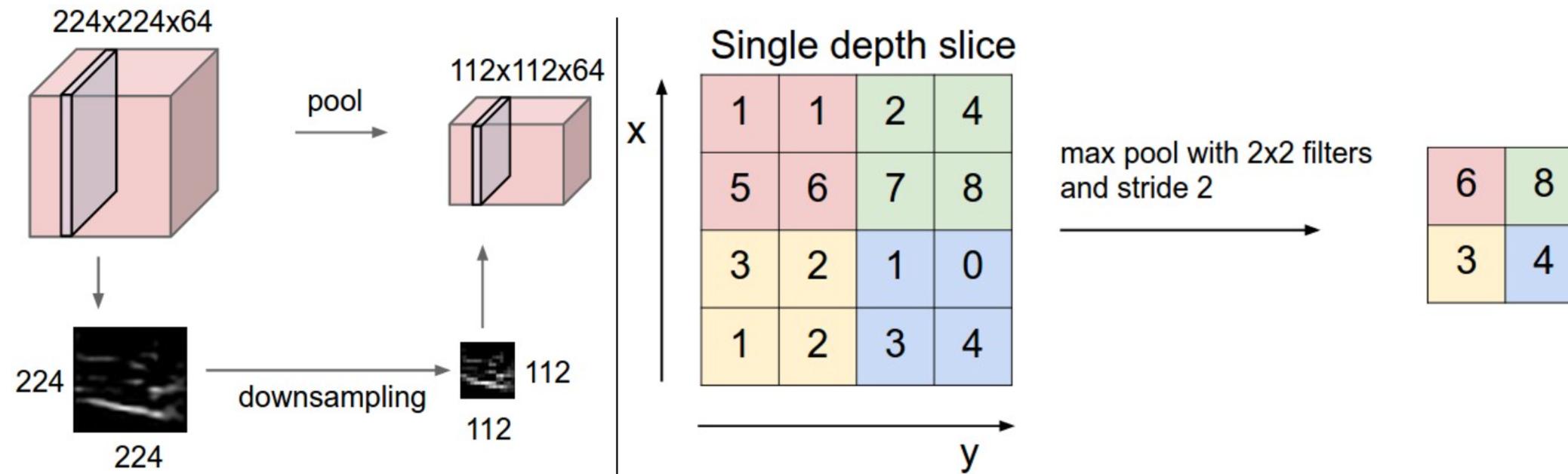
 eferrante@sinc.unl.edu.ar



# Redes Neuronales Convolucionales

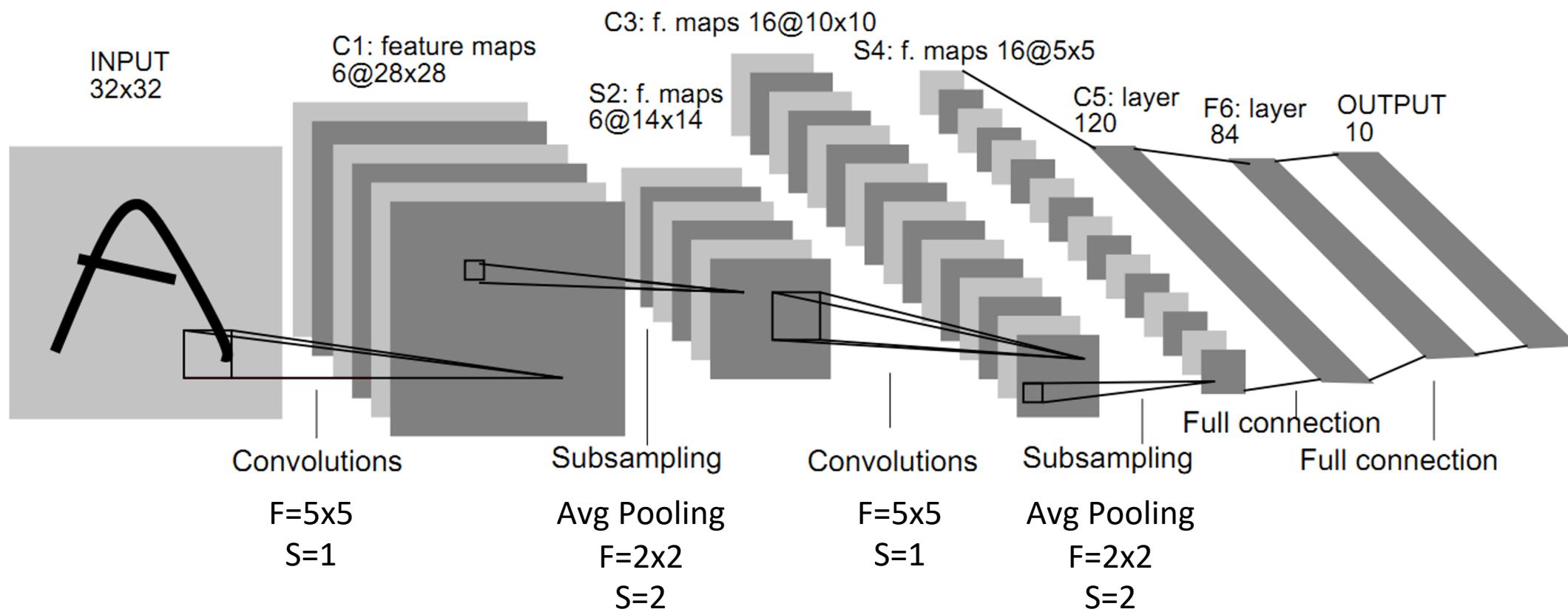


# Operaciones de Pooling



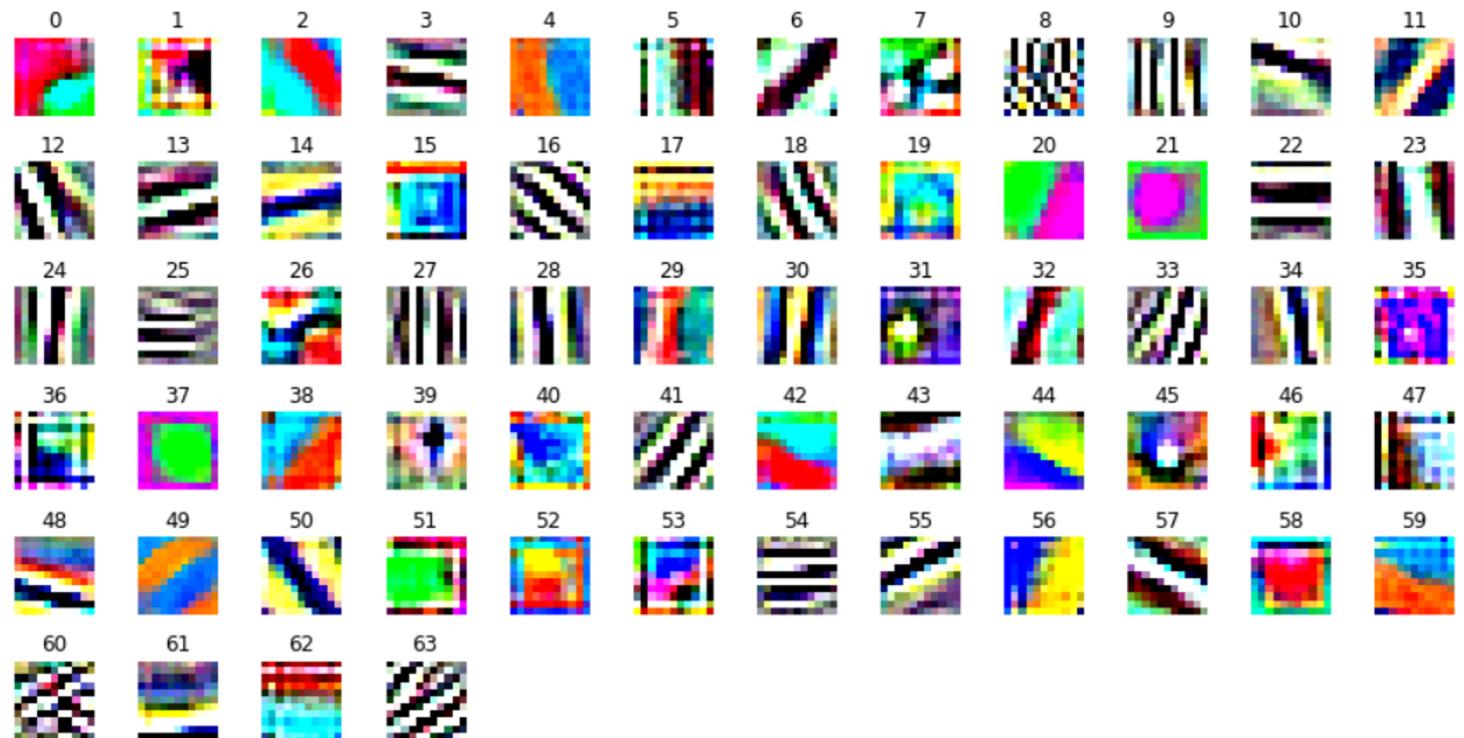
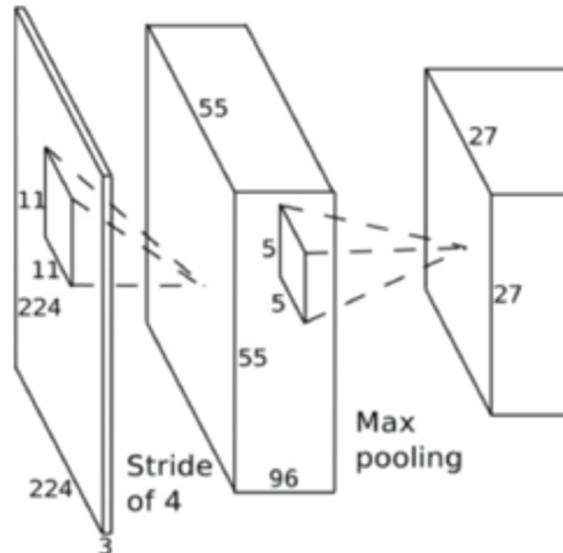
# Redes Neuronales Convolucionales

$$O = \frac{(W - F + 2P)}{S} + 1$$



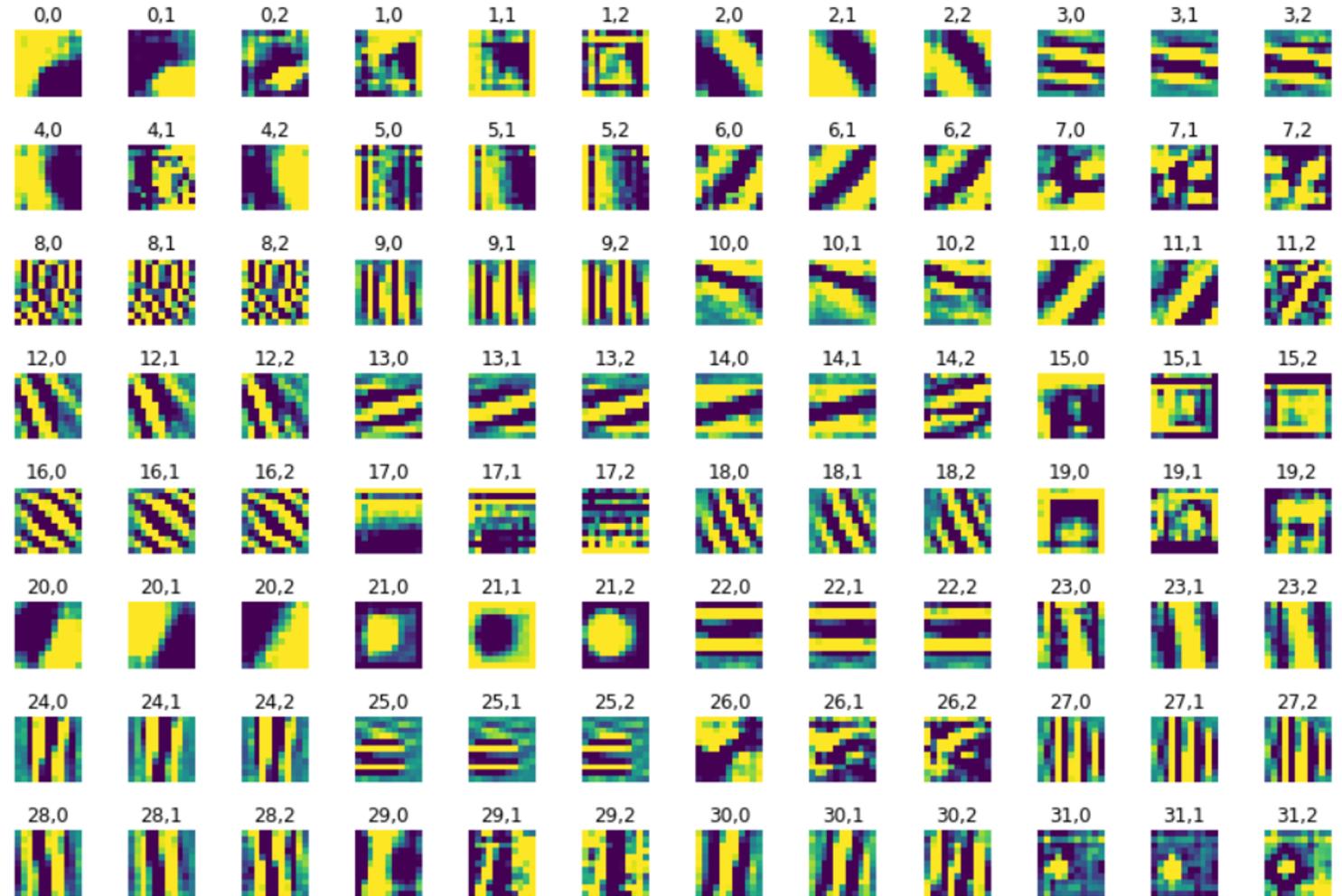
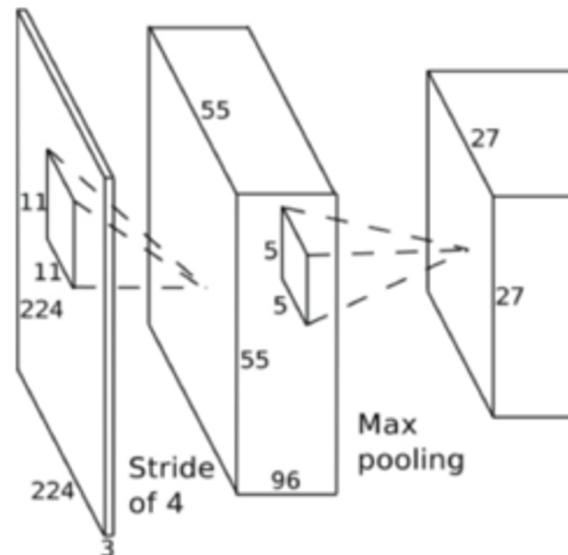
# Visualizando los filtros de una CNN entrenada

Filtros de la capa 1 de AlexNet visualizados a color (3 canales)



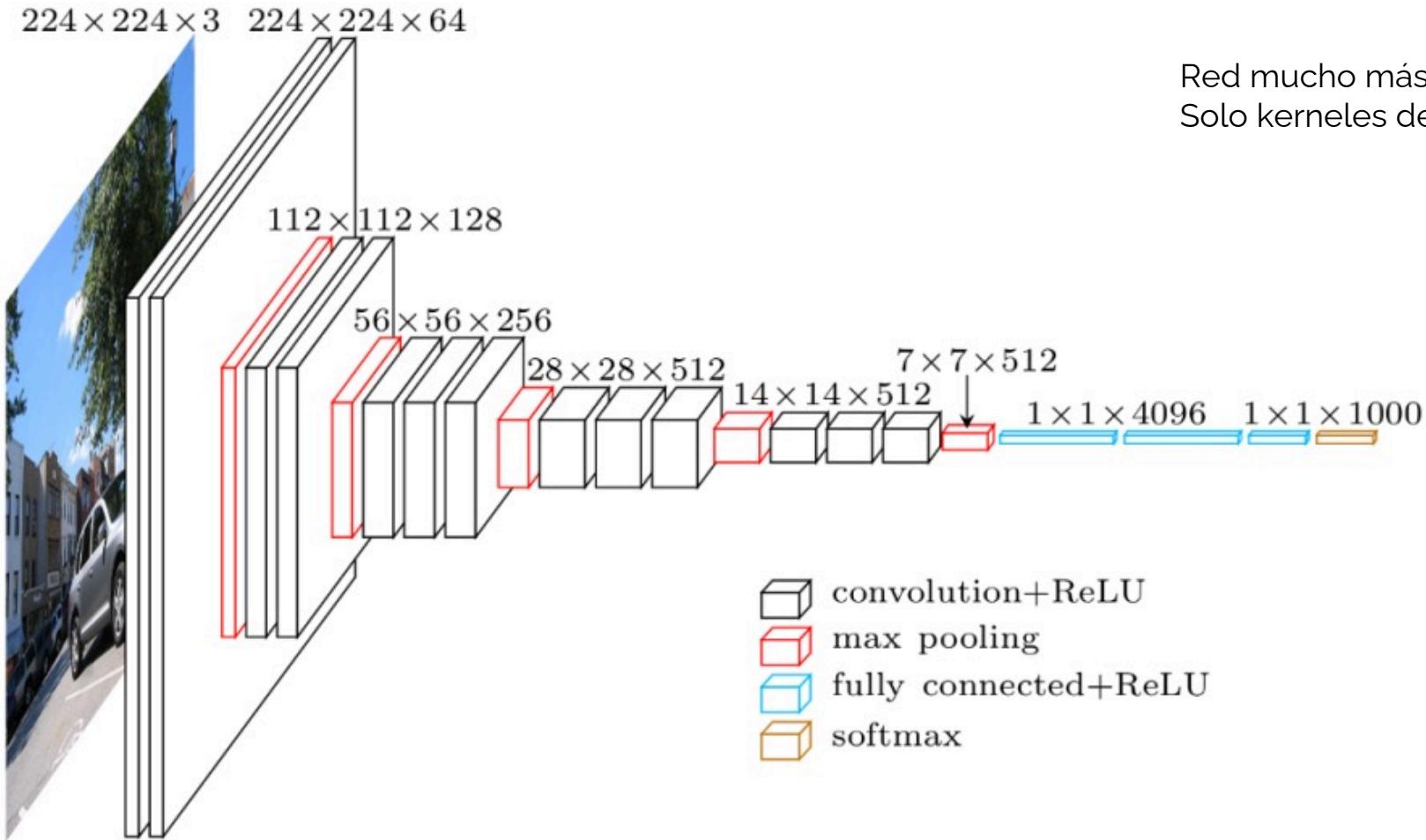
# Visualizando los filtros de una CNN entrenada

Filtros de la capa 1 de AlexNet visualizados de a 1 canal



# Algunas arquitecturas clásicas para clasificación

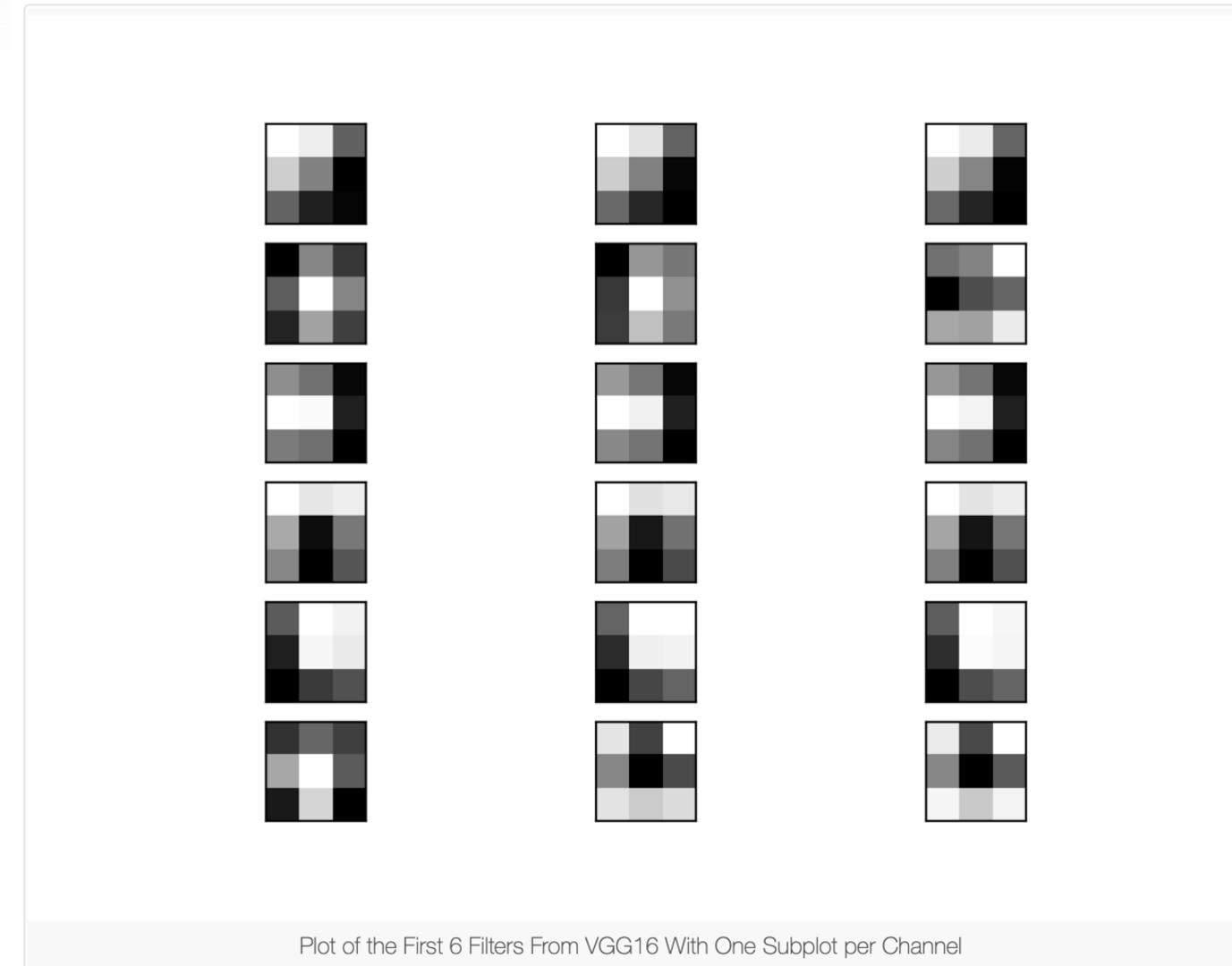
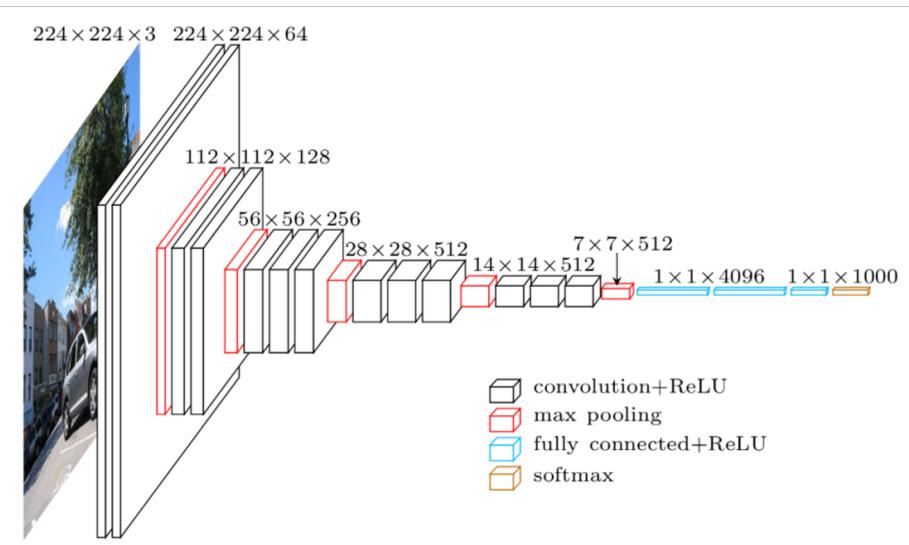
VGGNet (2014)



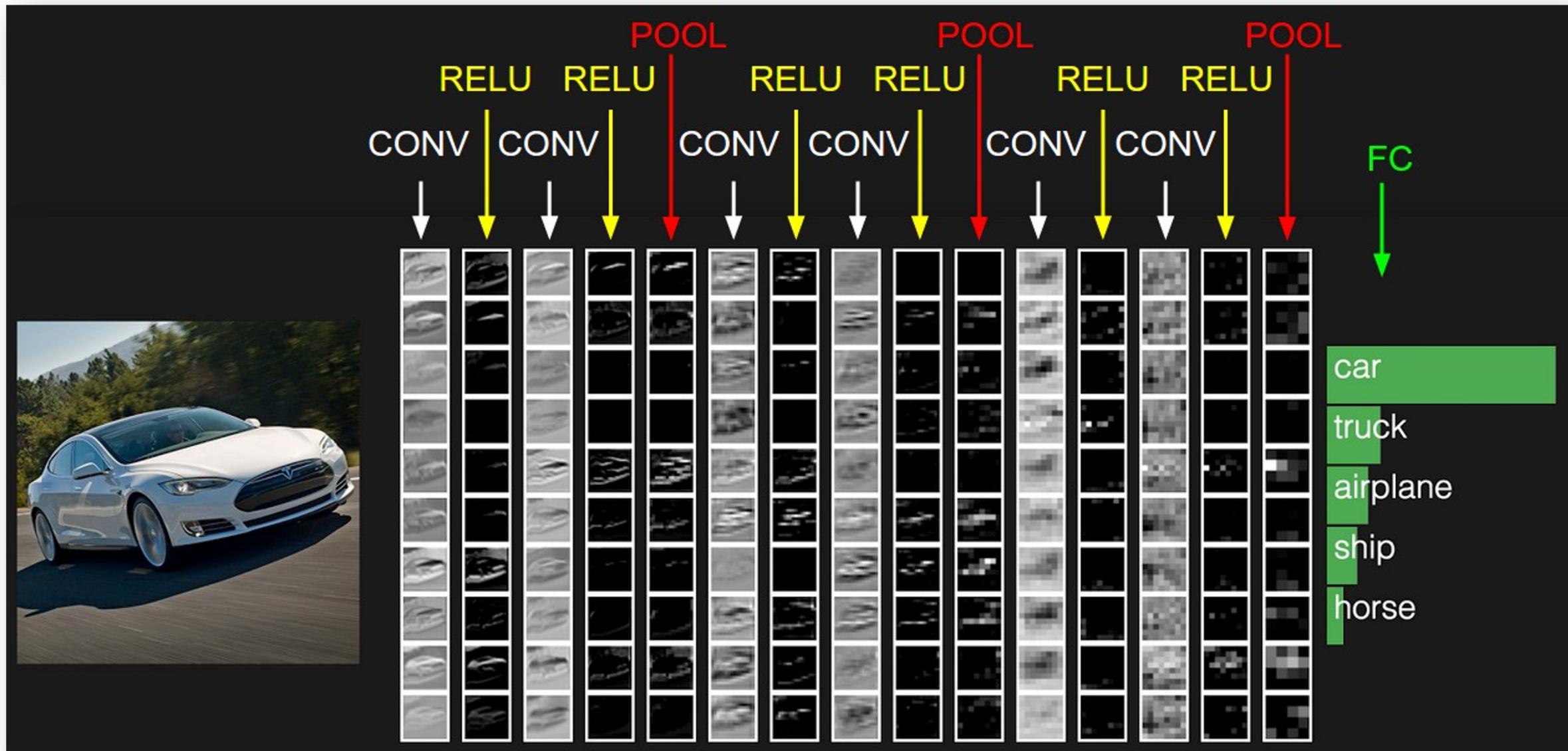
138 millones de parámetros

# Visualizando los filtros de una CNN entrenada

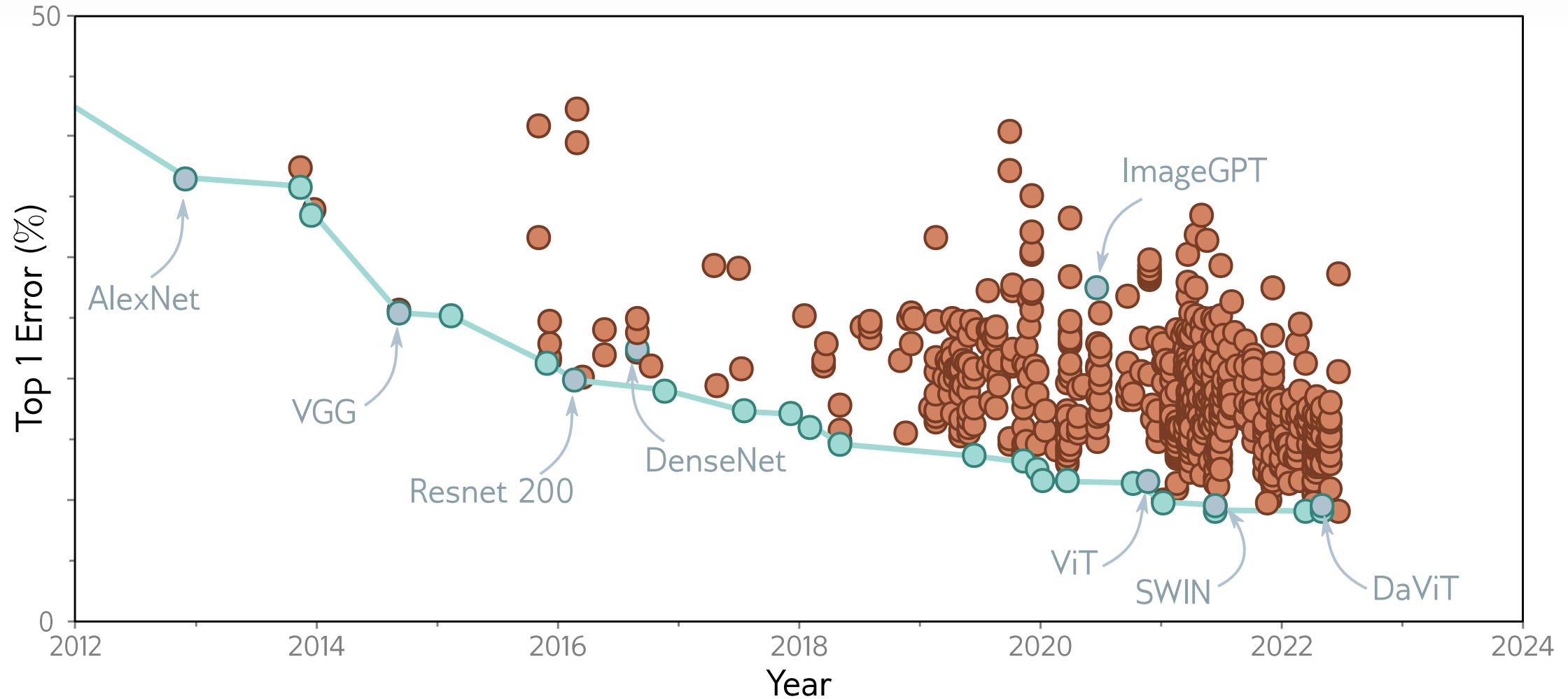
Filtros de la capa 1 de VGG Net (3x3)



# Visualizando los feature maps de una CNN entrenada



# Evolución de las arquitecturas



---

# **Entrenando una red neuronal para clasificación**

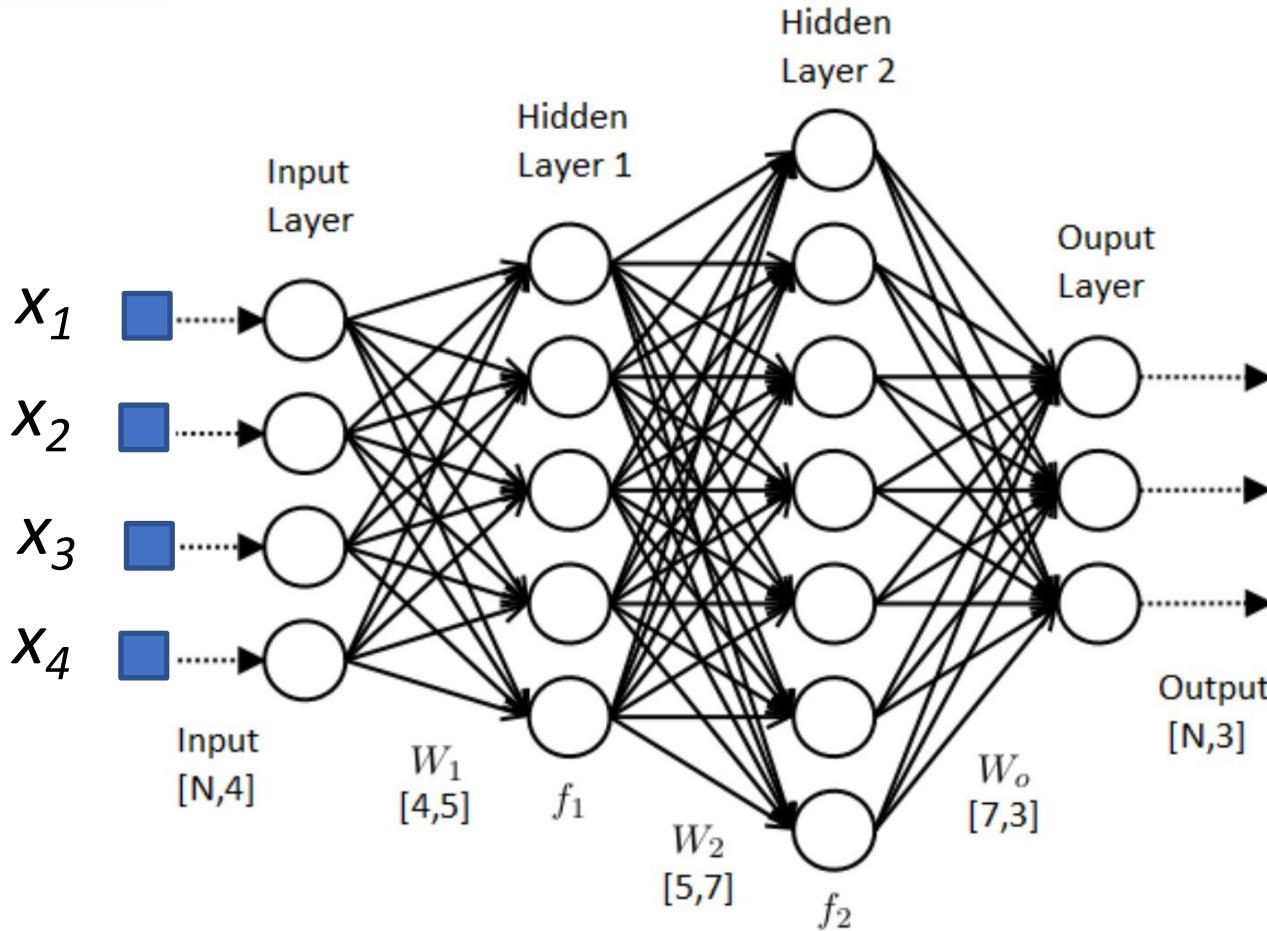
---

# Perceptrón Multicapa

Ejemplo: Problema de clasificación con 4 features de entrada y 3 clases de salida



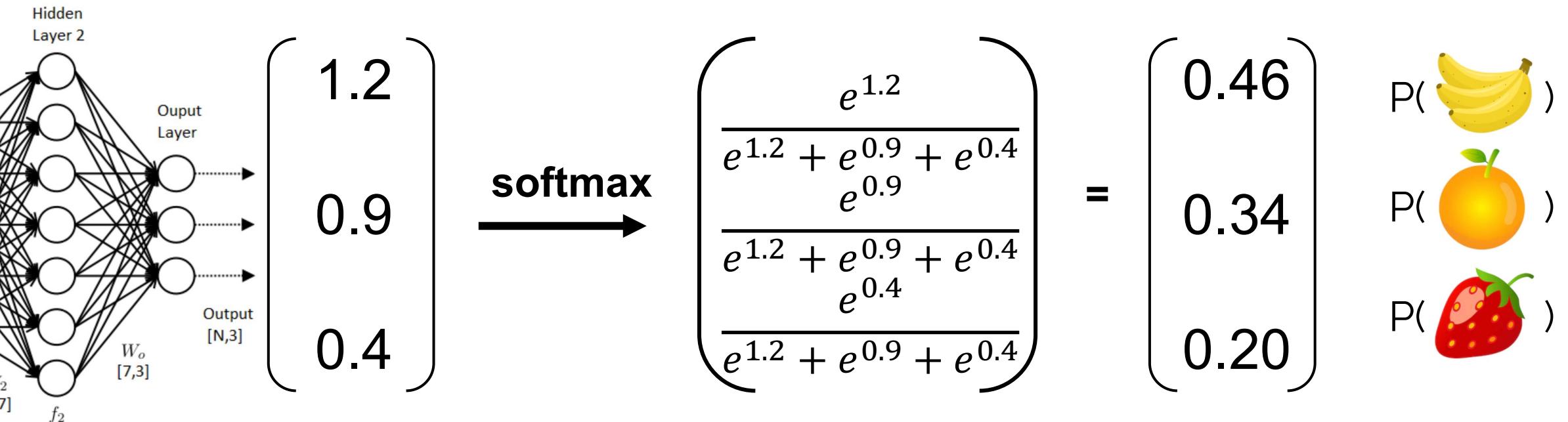
Color promedio  
Peso  
Textura  
Índice de esfericidad



# Softmax como función activación de salida

Permite obtener una "interpretación probabilística" de la salida

$$\text{softmax}(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{c=1}^C e^{z_c}}$$



---

# Función de pérdida

---

# Función de pérdida

Entropía Cruzada entre dos distribuciones de probabilidad p y q

$$CE(p, q) = - \sum_x p(x) \log q(x)$$

Distribución Ground Truth  
↓  
Distribución Estimada

Categoría	Etiqueta
Gato	1
Perro	2
Vaca	3

Notación One hot		
1	0	0
0	1	0
0	0	1

---

# **Funciones de activación**

---

# Funciones de activación

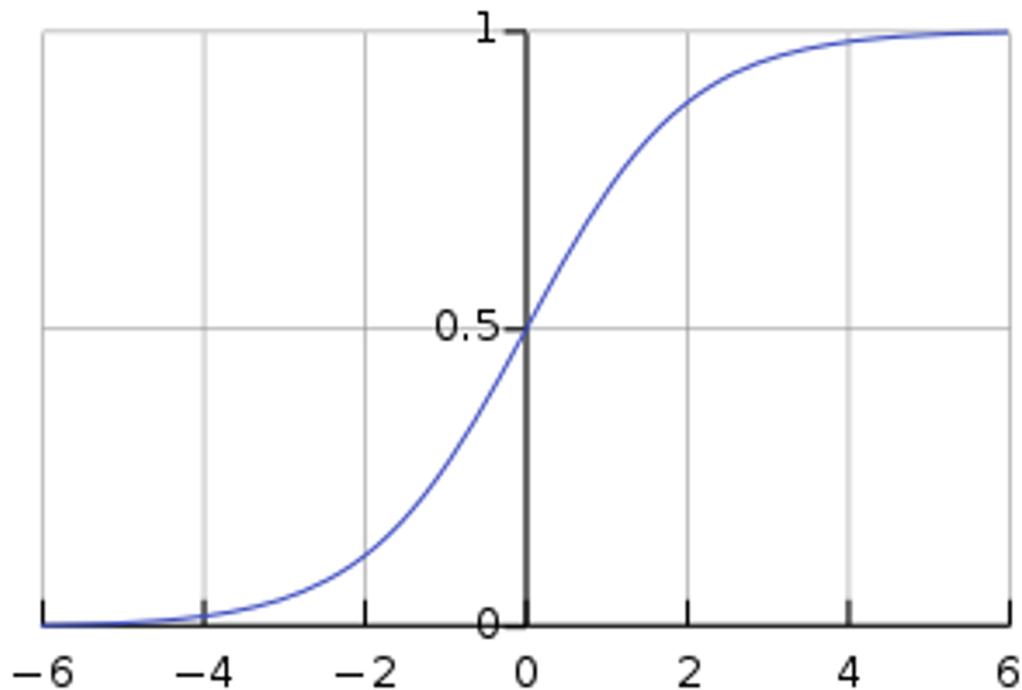
Fully Connected  
Layer

$$z = \sigma(b + \sum_i w_i x_i)$$

Convolutional  
Layer

$$z_{i,j} = \sigma \left( b + \sum_{f_1=0}^F \sum_{f_2=0}^F \sum_{d=0}^D w_{f_1,f_2,d} x_{i+f_1,j+f_2,d} \right)$$

# Función de activación Sigmoidea

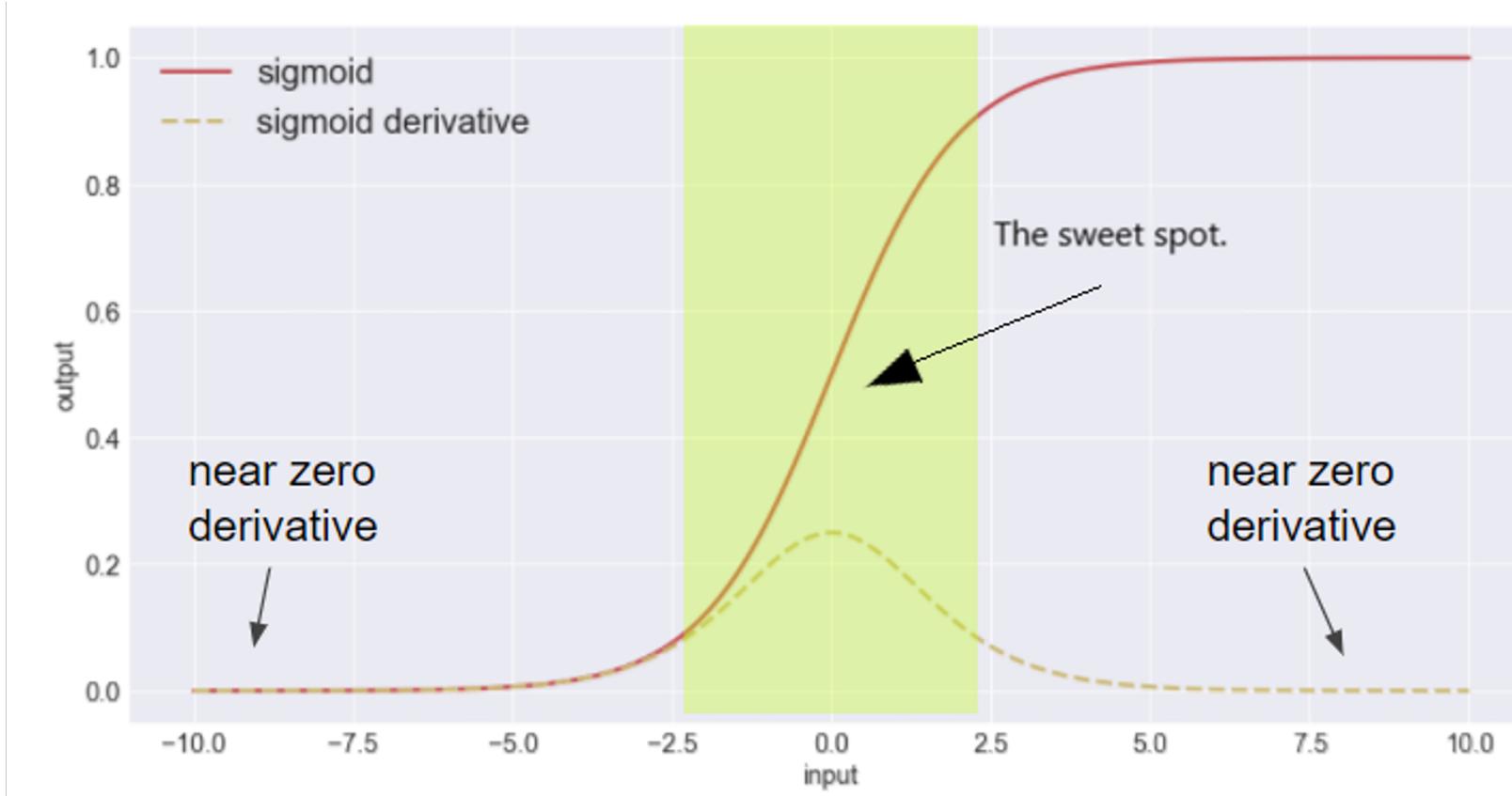


$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

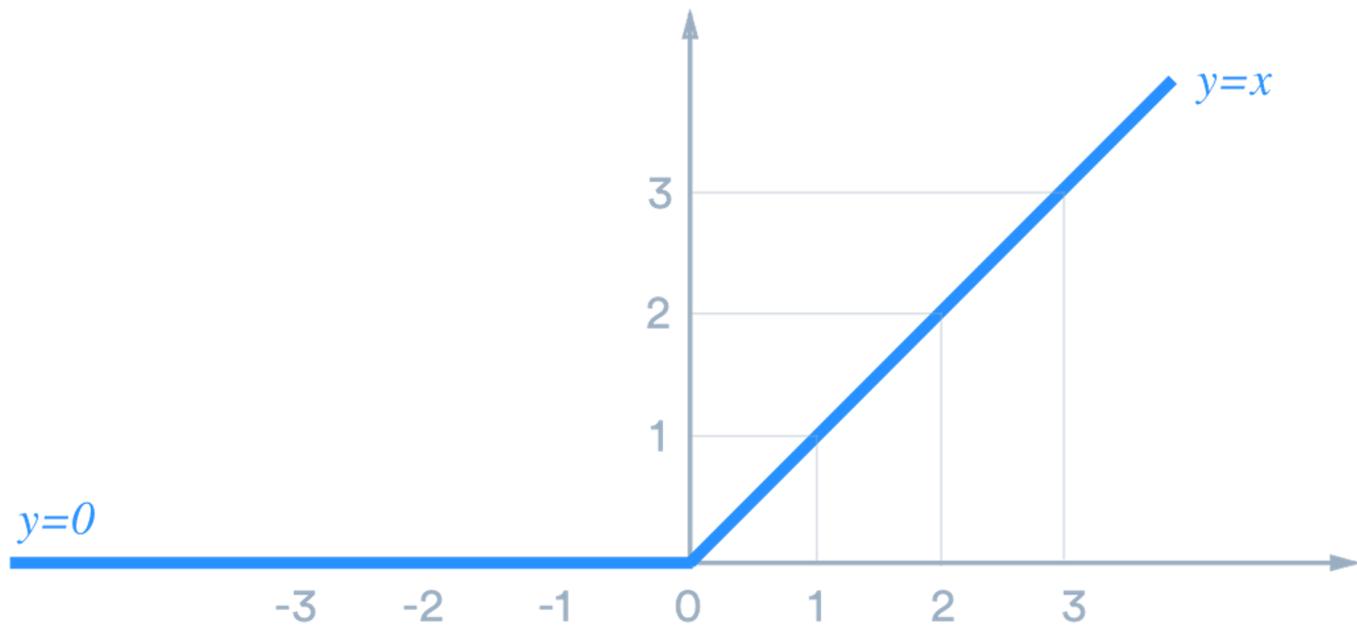
## Problema:

- El gradiente tiende a 0 cuando X tiende a  $\infty$  y  $-\infty$
- La multiplicación de pequeños gradientes por la regla de la cadena hace que el **gradiente se desvanezca**

# Función de activación Sigmoidea



# Función de activación ReLu

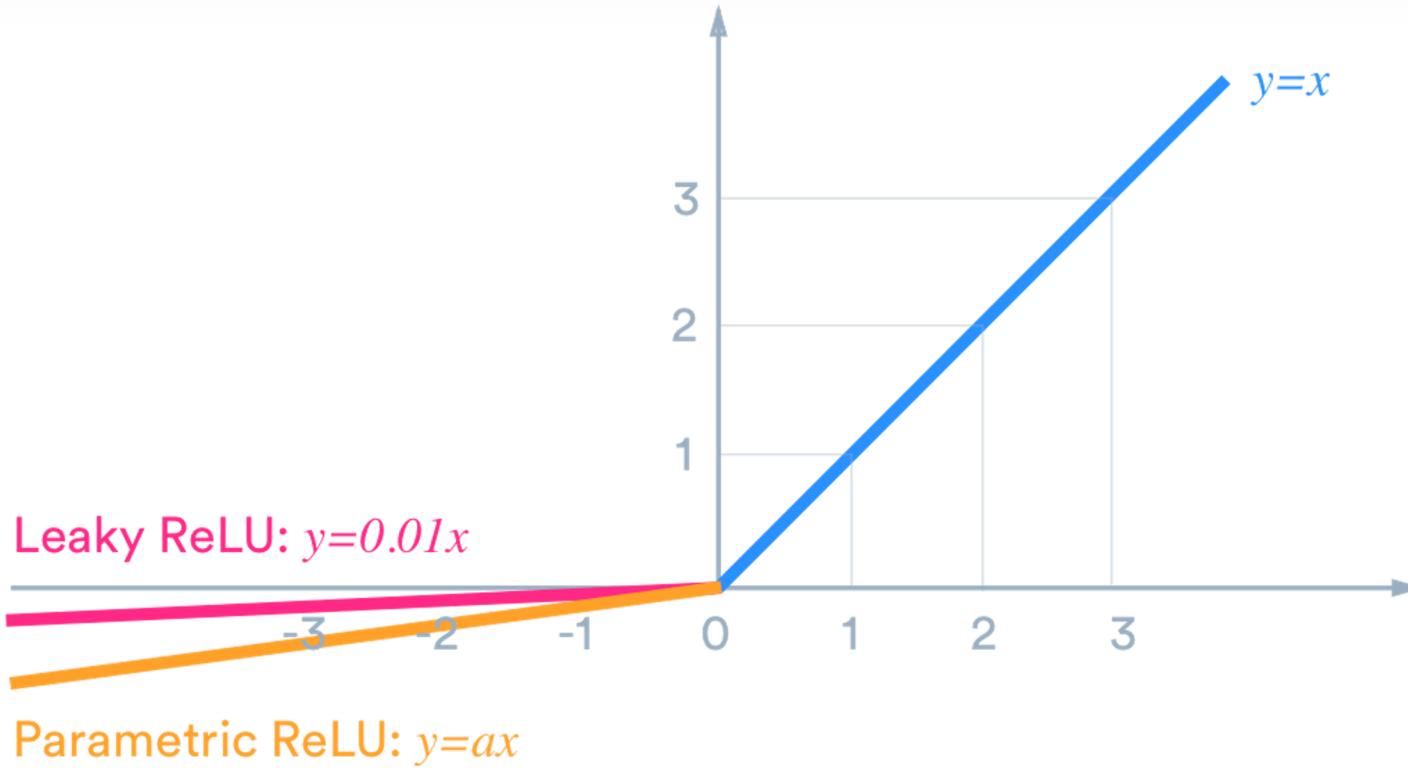


$$\text{ReLU}(x) = \max(0, x)$$

Menor probabilidad de que el gradiente se desvanezca que en el caso de la función de activación sigmoidea

Problema: el gradiente es 0 para  $x$  negativos, y puede desencadenar un proceso de **Dying Relu**.

# Función de activación Leaky ReLu



Evita el problema de la Dying Relu  
ya que tambien genera gradientes no  
nulos para valores de x negativos

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{si } x > 0 \\ 0.01x & \text{si } x \leq 0 \end{cases}$$

# Inicialización de pesos

Iniciar los pesos con  $W = 0$ ?

**Falta de asimetría** en la inicialización de los pesos

**Todas las neuronas terminan aprendiendo lo mismo**

# Inicialización de pesos

Inicialización aleatoria sampleando de  $\mathcal{N} = (\mu = 0, \sigma^2)$

- Cómo lograr que las neuronas operen en un rango razonable al comenzar el entrenamiento?

- Para una neurona  $z = \sigma(w_1x_1 + w_2x_2 + \dots + w_nx_n)$

- Mientras más grande  $n$  → más pequeños los  $w_i$

(Método Xavier o Glorot, útil para **tanh**)

- Samplear los pesos **w** de una distribución normal con varianza  $\sigma^2 = \frac{1}{\text{Cantidad de entradas de la neurona}}$

# Inicialización de pesos

- Método He-et-al (2015): <https://arxiv.org/abs/1502.01852>

Samplear los pesos  $\mathbf{w}$  de una distribución normal con varianza:

$$\mathcal{N} = (\mu = 0, \sigma^2 = \frac{2}{n})$$

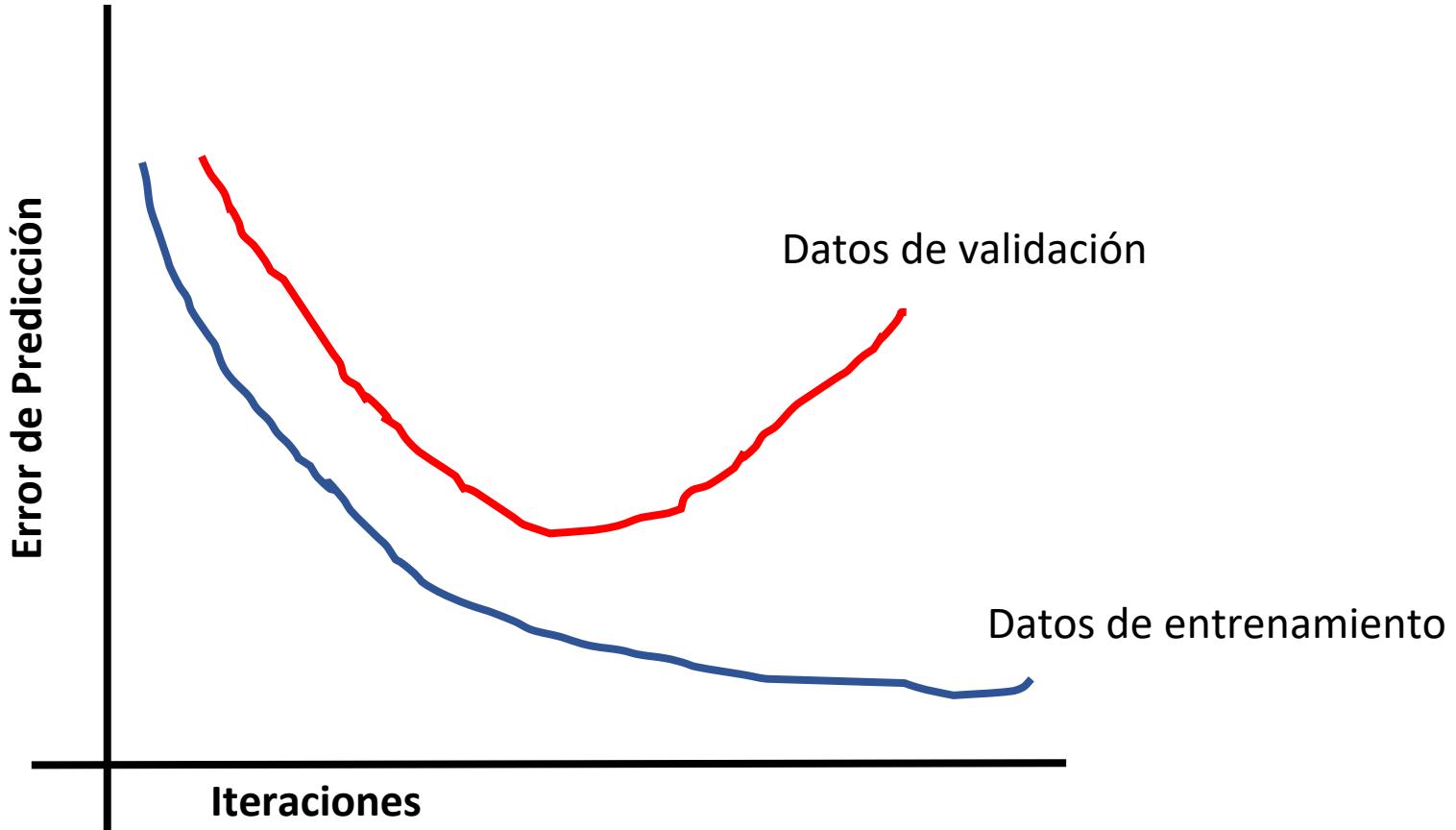
Útil para funciones **ReLU**

---

## **Sobreajuste y regularización**

---

# Sobre ajuste (overfitting)



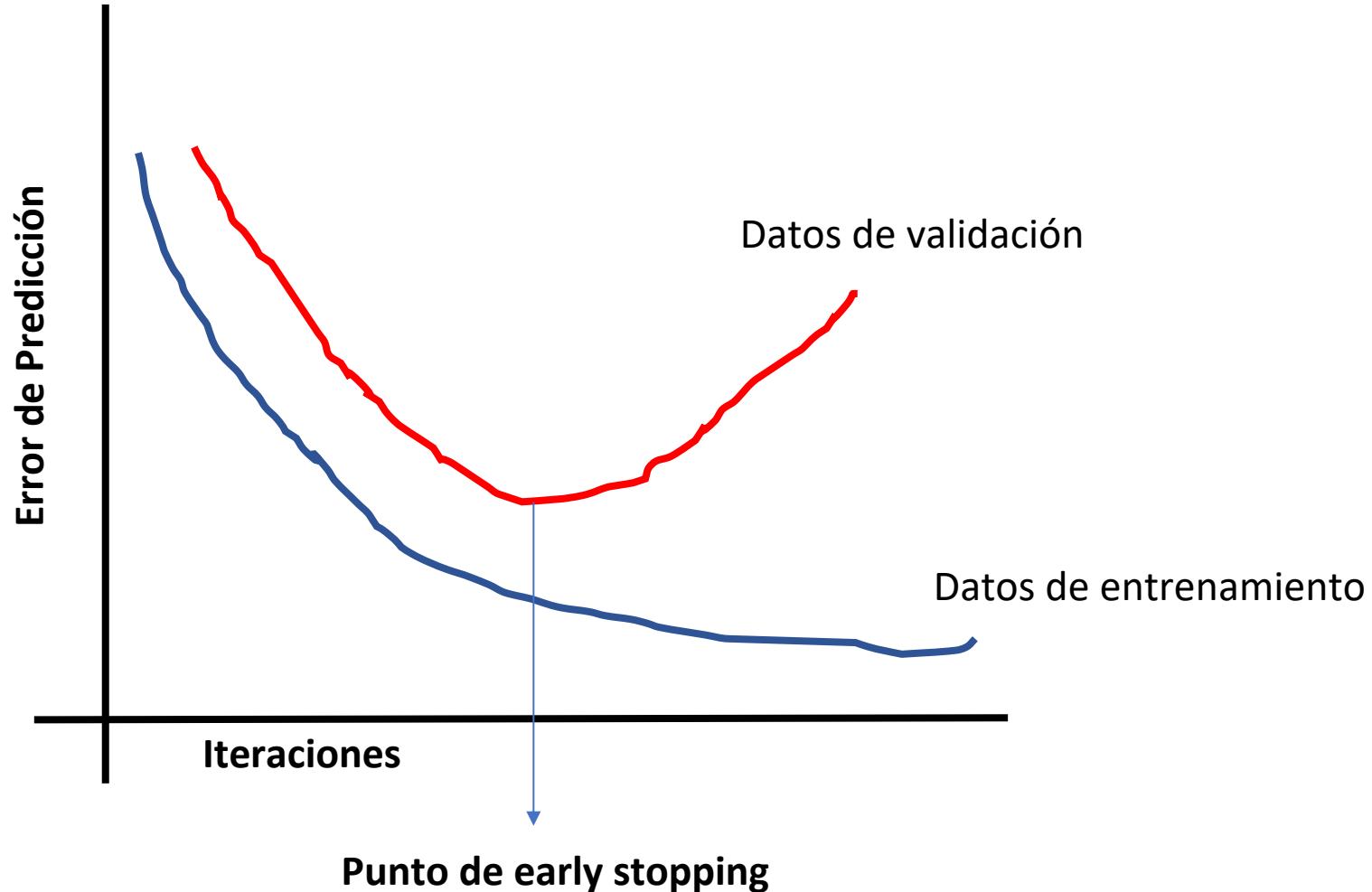
# **Sobre ajuste (overfitting)**

## Cómo evitarlo?

Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

Deep Learning. Goodfellow et al, 2016

# Sobre ajuste (overfitting)



# L2 Regularization (Weight Decay)

- Existen muchos  $\mathbf{w}$  que pueden minimizar una función de pérdida para un dataset de entrenamiento dado.
- Puedo reducir la complejidad de mi modelo acotando la cantidad de posibles modelos a construir.
- Una forma de hacerlo es evitando que existan  $w_i$  muy grandes, es decir, agregando una restricción sobre el valor que pueden tomar los pesos  $w_i$ .

# L2 Regularization (Weight Decay)

Agregar un término de regularización a la función de pérdida que sea la norma euclídea cuadrada de los pesos

$$\|\mathbf{w}\|_2^2 = \left( \left( \sum_{m=1}^M w_m^2 \right)^{1/2} \right)^2 = \sum_{m=1}^M w_m^2$$

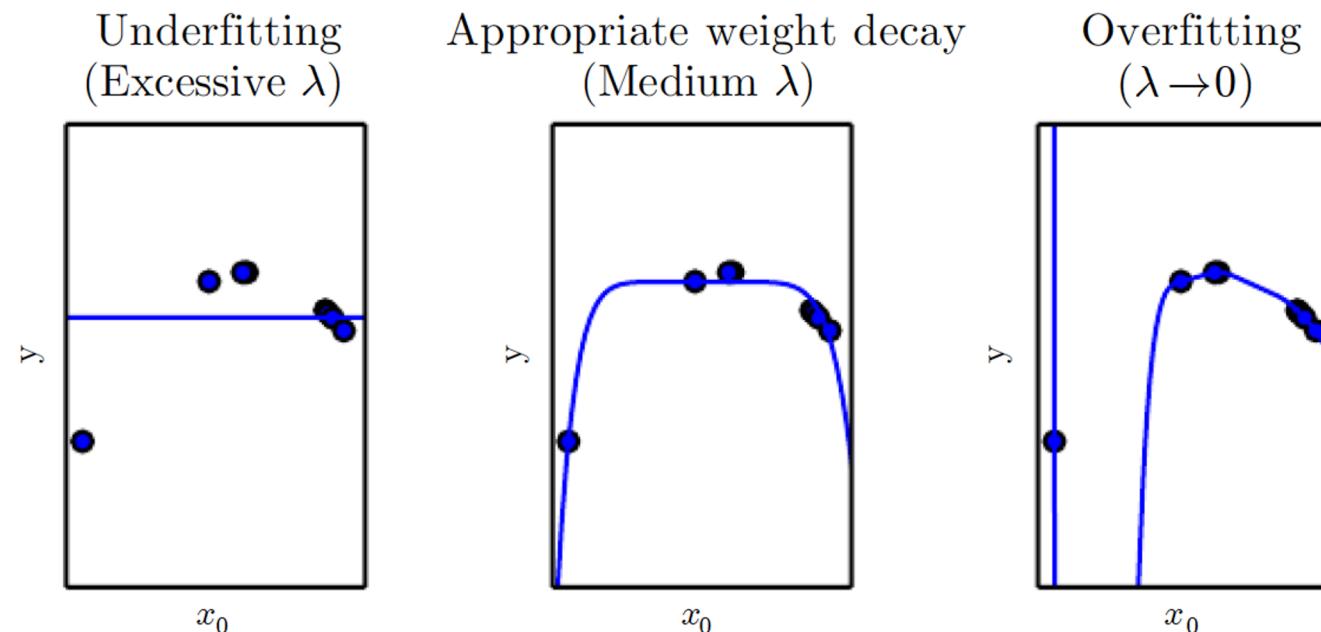
$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^N \mathcal{L}((\mathbf{x}, d)_n; \mathbf{w}) + \frac{1}{2} \lambda \overbrace{\|\mathbf{w}\|_2^2}$$

N = Número de data samples en el training set

M = Número de parámetros en w

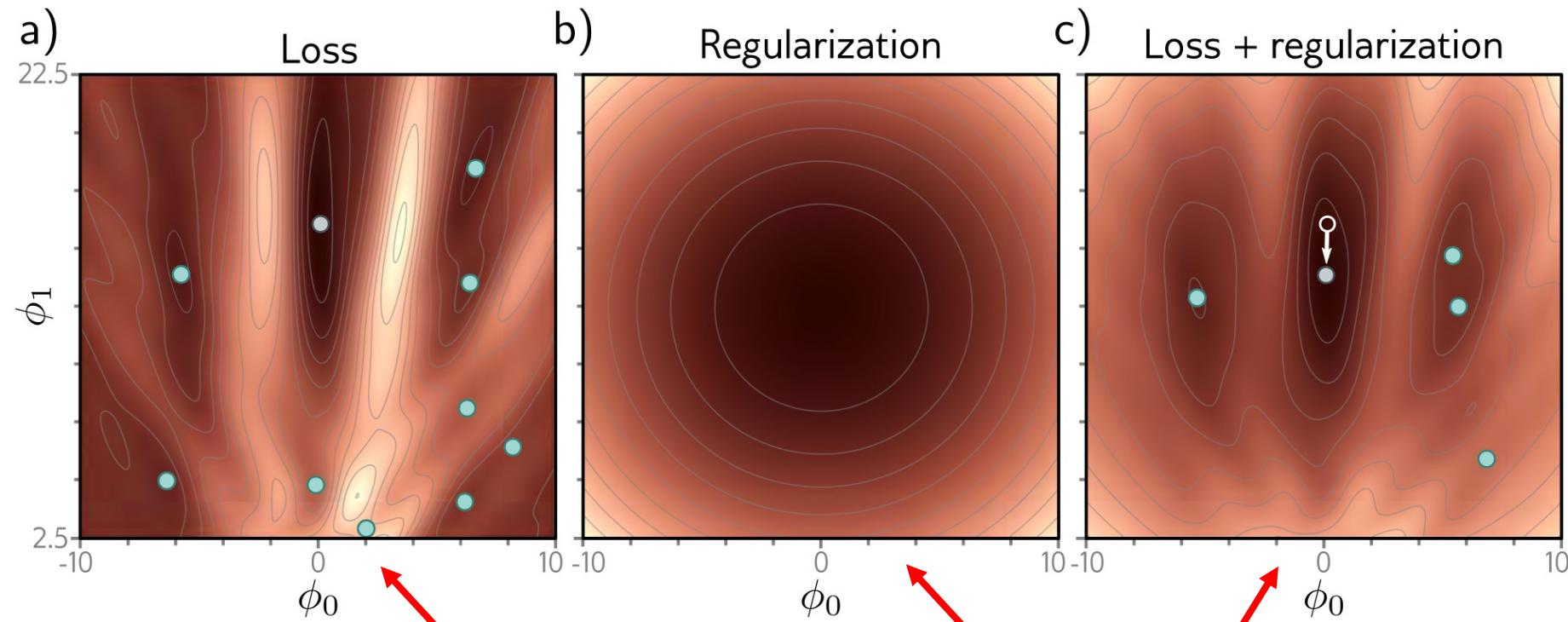
# Regularización: L2 Weight-decay

Influencia del parámetro  $\lambda$ : Ejemplo al fittear un polinomio de orden 9 sobre datos sampleados de una función cuadrática



$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^N \mathcal{L}((\mathbf{x}, d)_n; \mathbf{w}) + \frac{1}{2} \lambda \|\mathbf{w}\|_2^2$$

# Efecto de la regularización en la función de pérdida



$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^N \mathcal{L}((\mathbf{x}, d)_n; \mathbf{w}) + \frac{1}{2} \lambda \|\mathbf{w}\|_2^2$$

Figura extraída de (Prince, 2023)

---

## Aumentación de datos

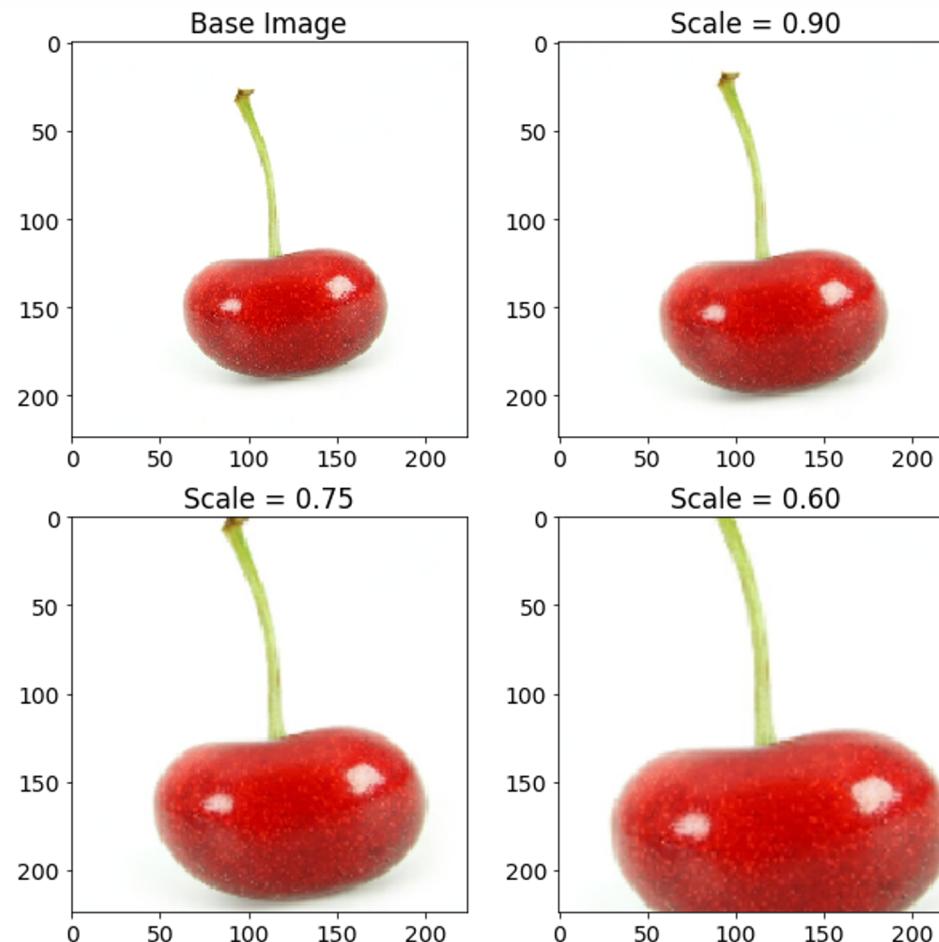
---

# Aumentación de datos

- **En tiempo de entrenamiento**, aumentar artificialmente el dataset utilizando transformaciones en los datos y conservando las etiquetas
- **En tiempo de prueba**, es posible generar N versiones de la imagen de test, estimar las predicciones y promediarlas

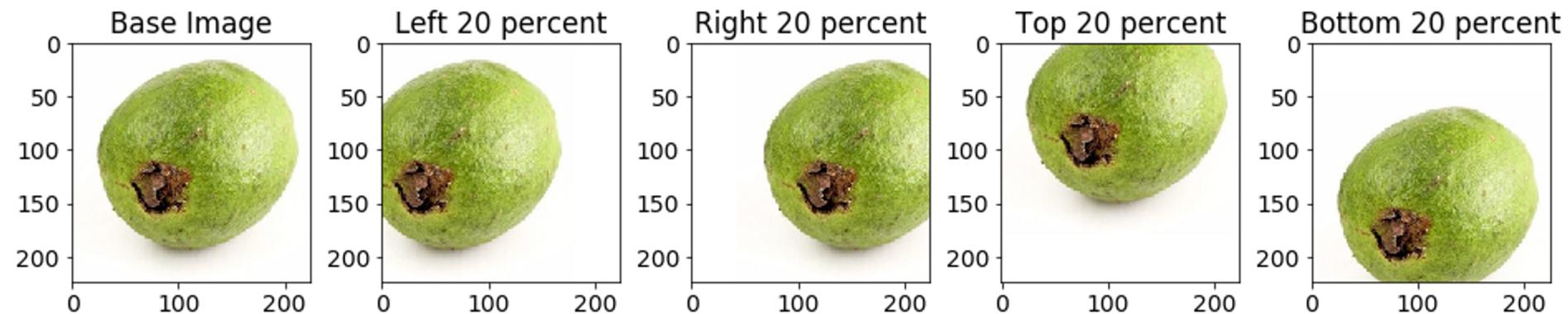
# Aumentación de datos

## Scaling



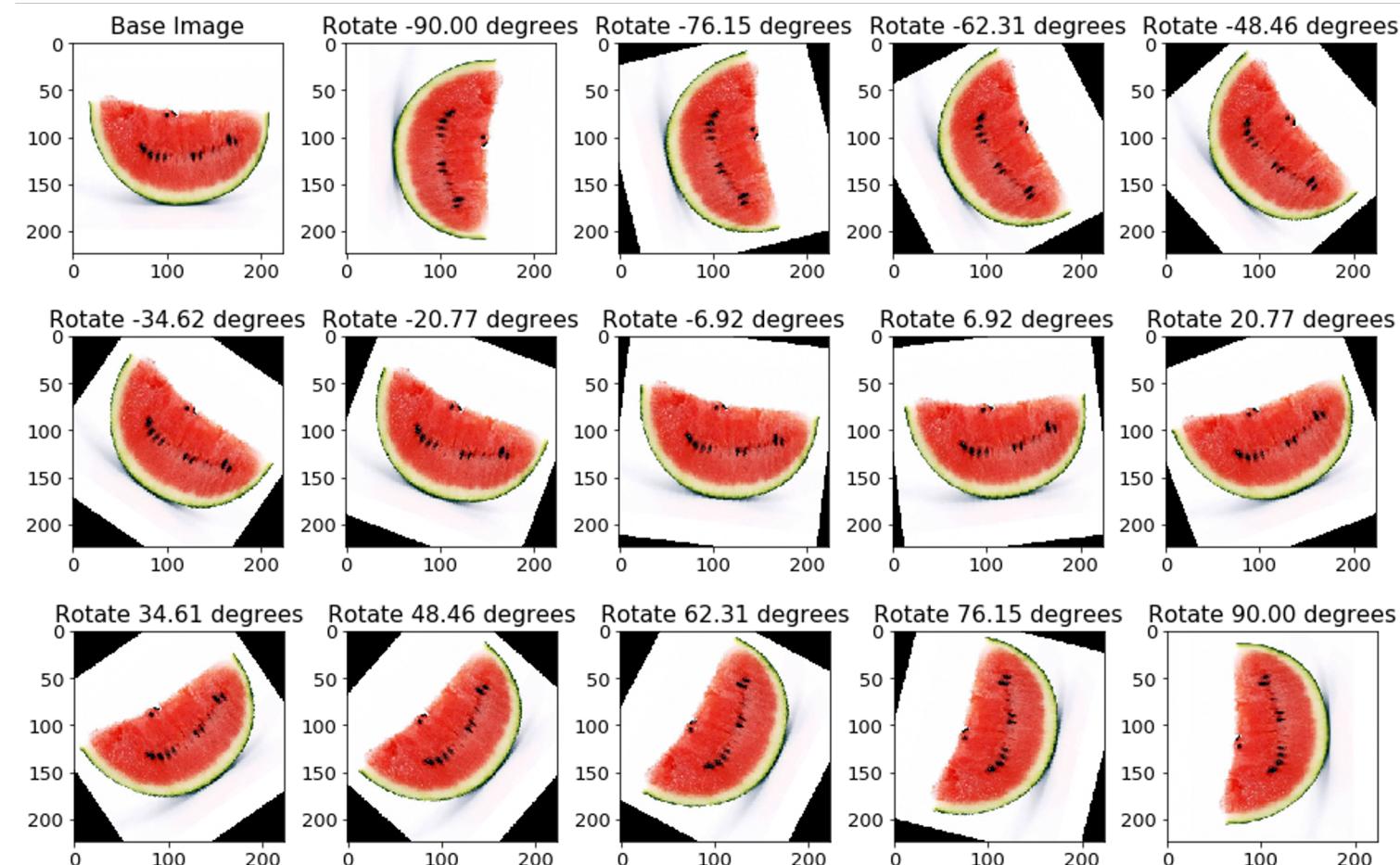
# Aumentación de datos

Translation



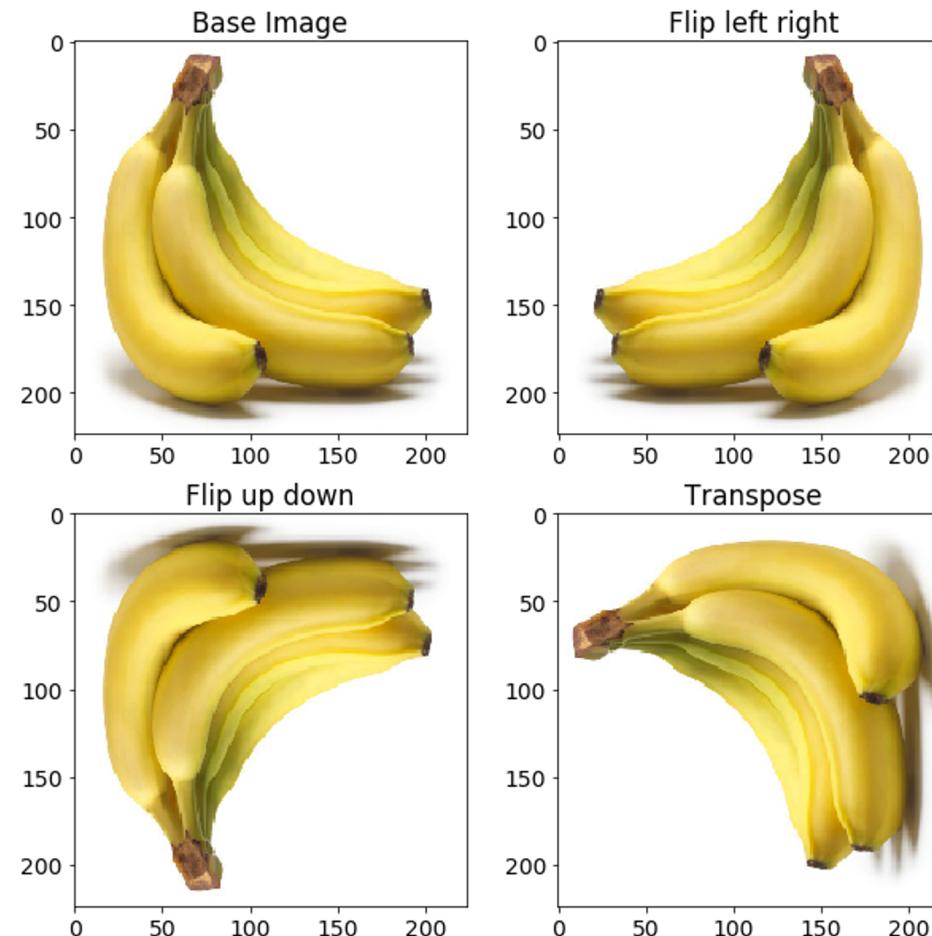
# Aumentación de datos

## Rotation



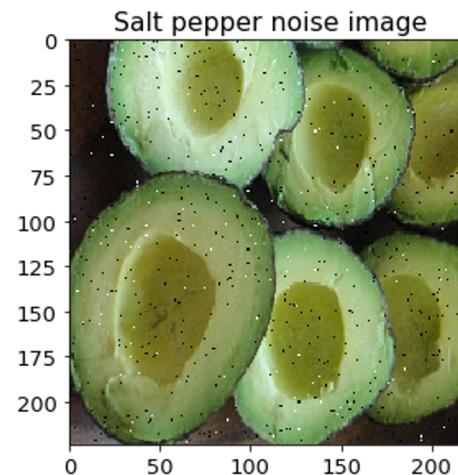
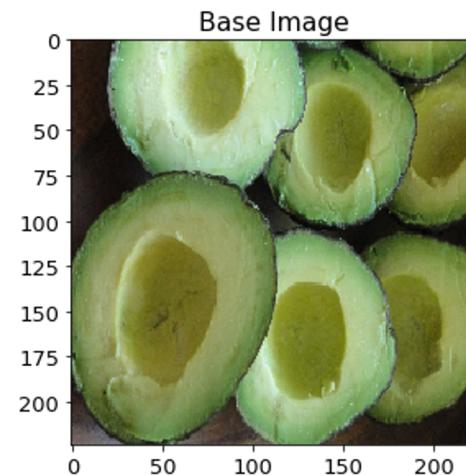
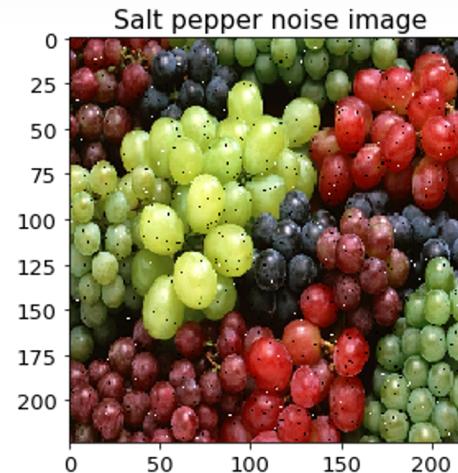
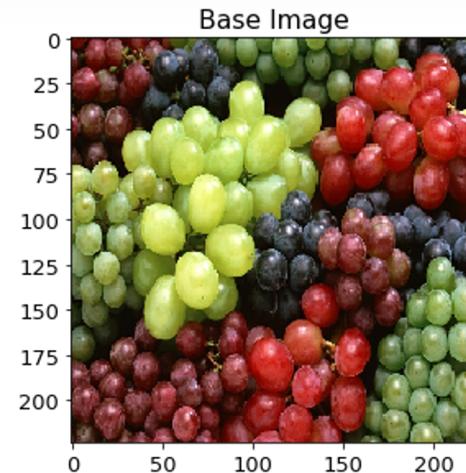
# Aumentación de datos

## Flipping



# Aumentación de datos

## Ruido



# Aumentación de datos

a) Original



b) Flip



c) Rotate and crop



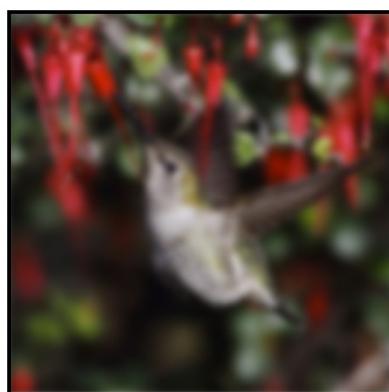
d) Vertical stretch



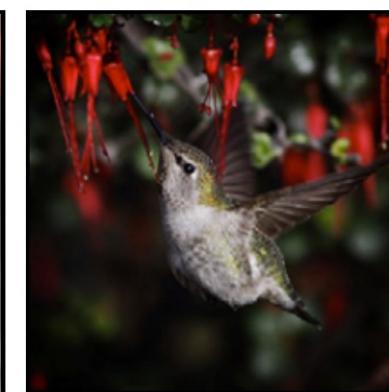
e) Color balance



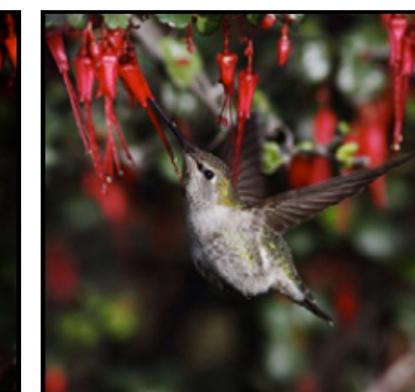
f) Blur



g) Vignette

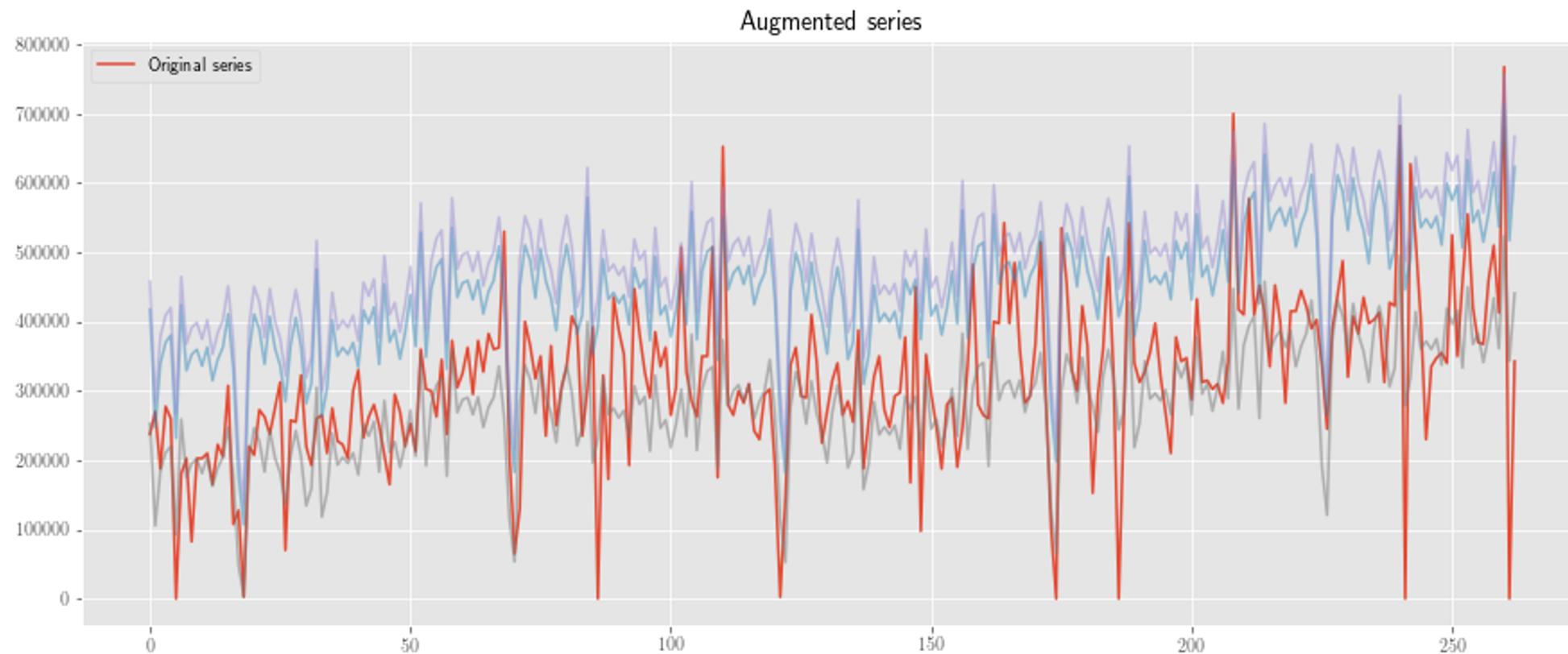


h) Pincushion



# Aumentación de datos

No sólo para imágenes



---

# **Dropout**

---

# Regularización: Dropout (2015)

Journal of Machine Learning Research 15 (2014) 1929-1958

Submitted 11/13; Published 6/14

## Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava

Geoffrey Hinton

Alex Krizhevsky

Ilya Sutskever

Ruslan Salakhutdinov

Department of Computer Science  
University of Toronto  
10 Kings College Road, Rm 3302  
Toronto, Ontario, M5S 3G4, Canada.

NITISH@CS.TORONTO.EDU  
HINTON@CS.TORONTO.EDU

KRIZ@CS.TORONTO.EDU

ILYA@CS.TORONTO.EDU

RSALAKHU@CS.TORONTO.EDU

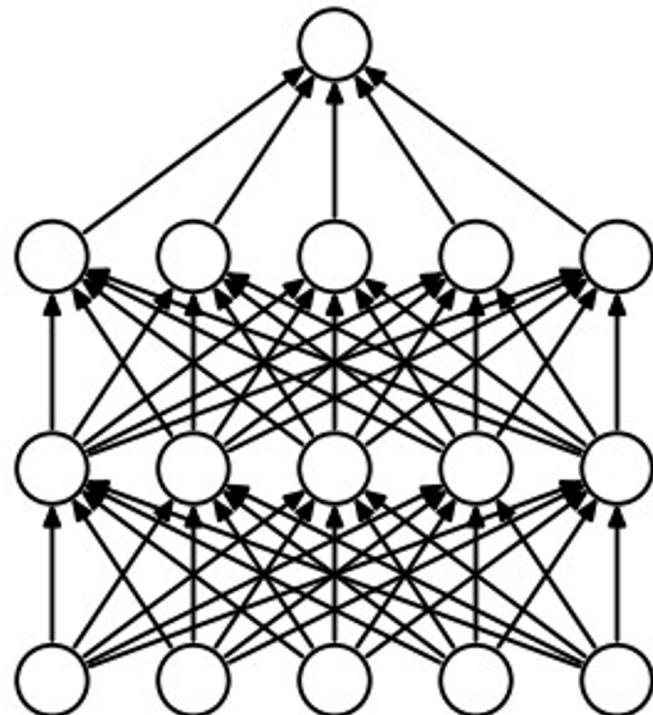
Editor: Yoshua Bengio

### Abstract

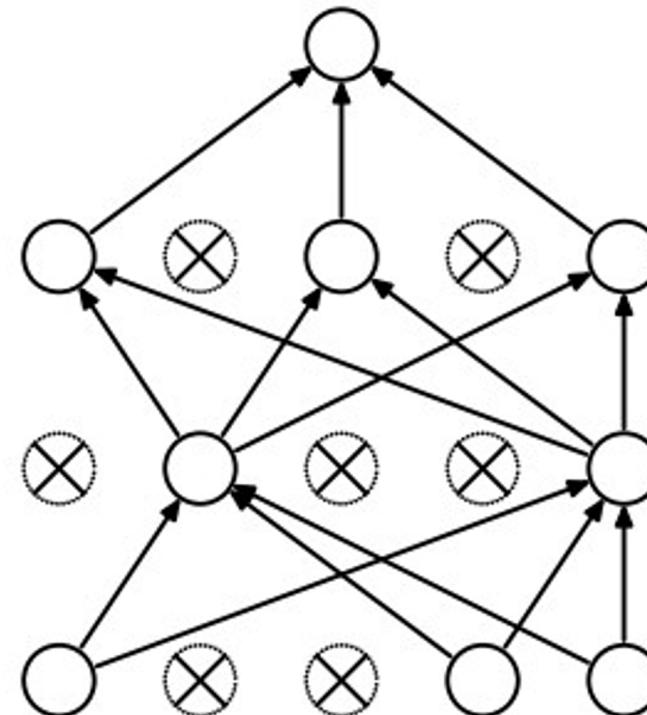
Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. They are slow to use, making it difficult to deal with different large problems. Dropout is a simple, effective technique for regularization of fully connected neural networks. It is computationally efficient, and it can be applied to any neural network, without requiring any modification to the architecture or the training algorithm. It is based on randomly dropping units during training time. It is shown to be an effective way to prevent overfitting, and it can be used alone or in combination with other regularization methods. The proposed method is very simple to implement, requires very little computation, and achieves state-of-the-art performance on several benchmarks.

# Regularización: Dropout

Aleatoriamente, con un probabilidad (**1-p**), las neuronas son ignoradas en la pasada forward durante entrenamiento



(a) Standard Neural Net



(b) After applying dropout.

En test time, todas las neuronas son utilizadas y los pesos de salida de la neurona son multiplicados por  $p$

# Regularización: Dropout

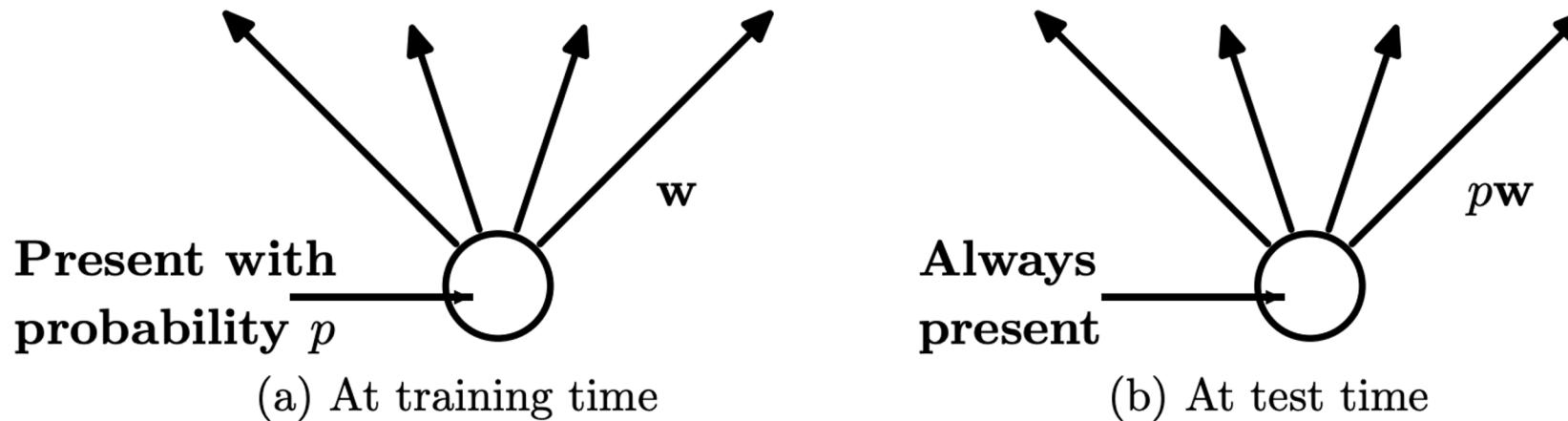


Figure 2: **Left:** A unit at training time that is present with probability  $p$  and is connected to units in the next layer with weights  $w$ . **Right:** At test time, the unit is always present and the weights are multiplied by  $p$ . The output at test time is same as the expected output at training time.

En test time, todas las neuronas son utilizadas y los pesos de salida de la neurona son multiplicados por  $p$

# Regularización: Dropout

Sin Dropout:

$$\begin{aligned} z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)}, \\ y_i^{(l+1)} &= f(z_i^{(l+1)}), \end{aligned}$$

---

Con Dropout:

$$\begin{aligned} r_j^{(l)} &\sim \text{Bernoulli}(p), \\ \tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\ z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \tilde{\mathbf{y}}^l + b_i^{(l+1)}, \\ y_i^{(l+1)} &= f(z_i^{(l+1)}). \end{aligned}$$

# Regularización: Dropout

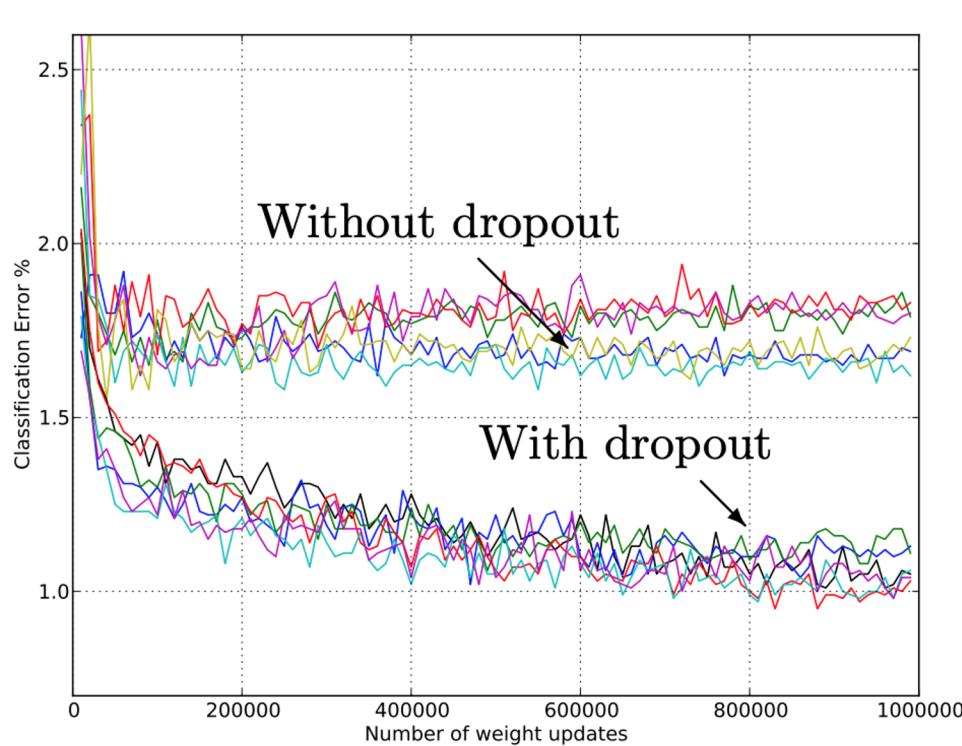


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

# Regularización: Dropout

**Pregunta:** Cómo usar Dropout para tener una idea sobre la incertidumbre de las predicciones de mi red?

Paper: <https://arxiv.org/pdf/1807.07356.pdf>

---

## **Normalización de los datos de entrada**

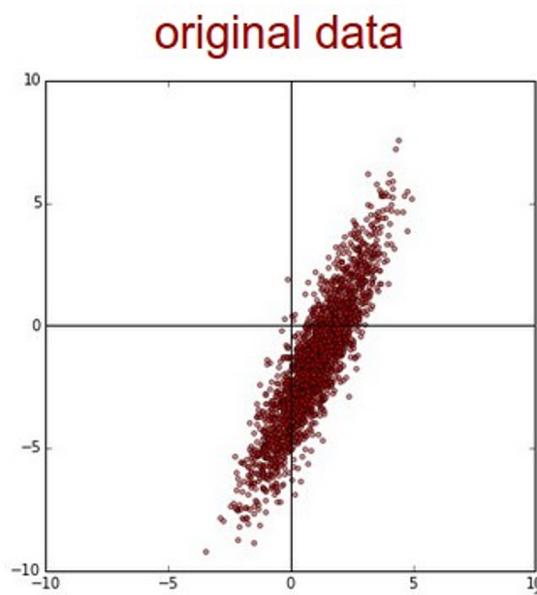
---

# Normalización de las entradas

Cuando cada dimensión se mueve en un rango de valores diferente

Matriz de datos de  
entrenamiento  
 $[ N \times D ]$

$x_{1,1}$	...	$x_{1,D}$
.		
.		
$x_{N,1}$	...	$x_{N,D}$



IMPORTANTE: La media y el desvío son siempre calculados usando sólo los datos de training, pero todos los datos (test y validación también) son normalizados.

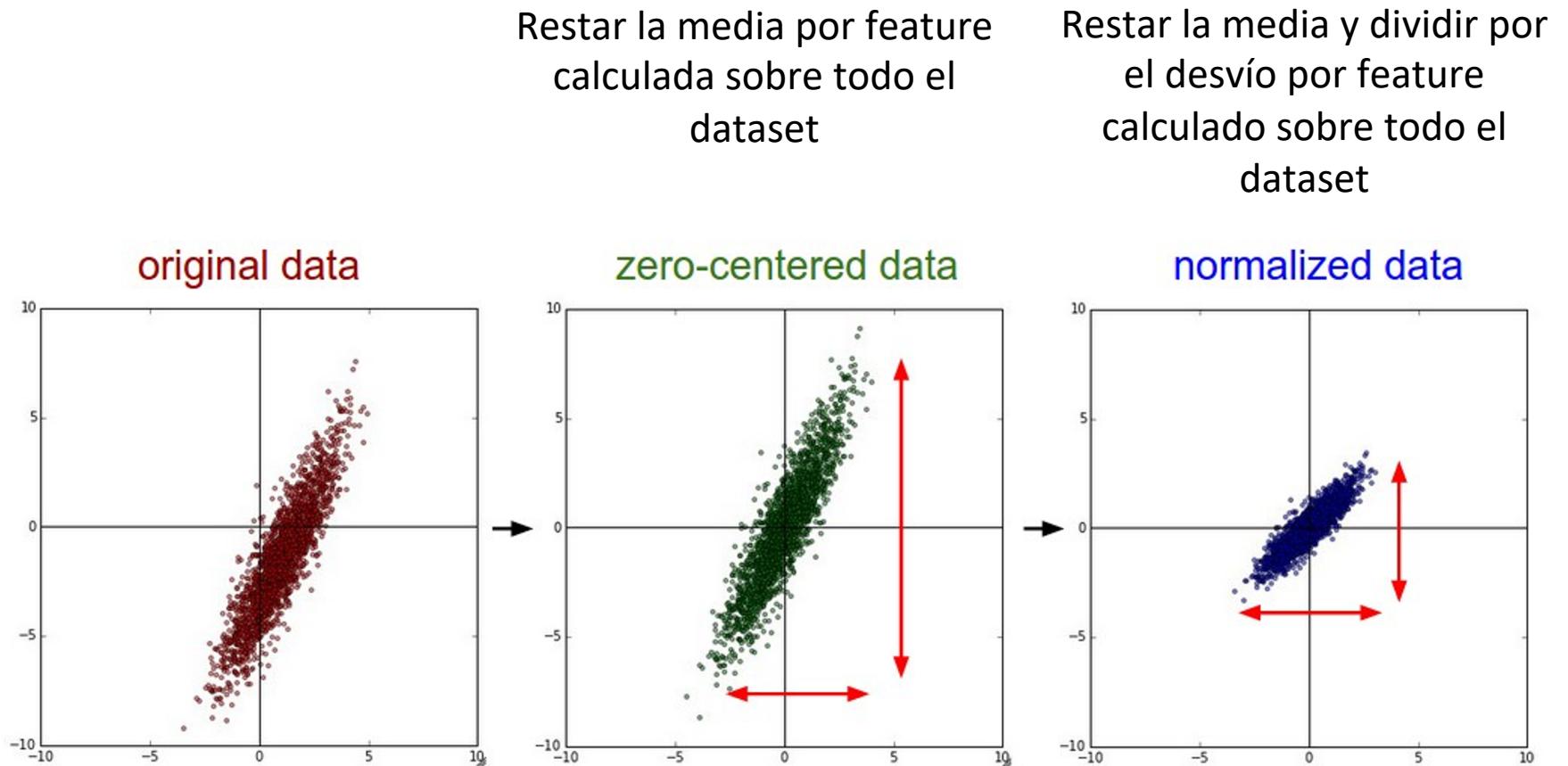
Imagen extraída de <http://cs231n.github.io/neural-networks-2/>

# Normalización de las entradas

Cuando cada dimensión se mueve en un rango de valores diferente

Matriz de datos de  
entrenamiento  
 $[ N \times D ]$

$x_{1,1}$	$\dots$	$x_{1,D}$
.		
.		
$x_{N,1}$	$\dots$	$x_{N,D}$



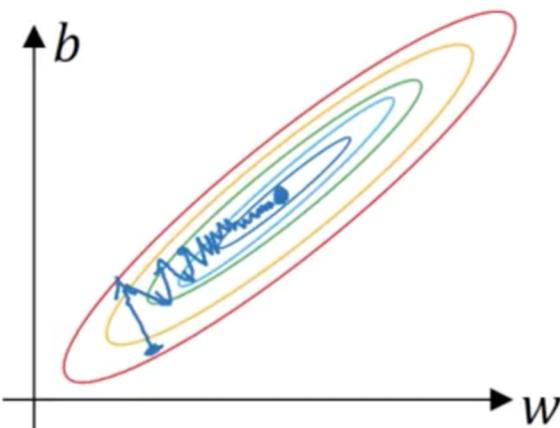
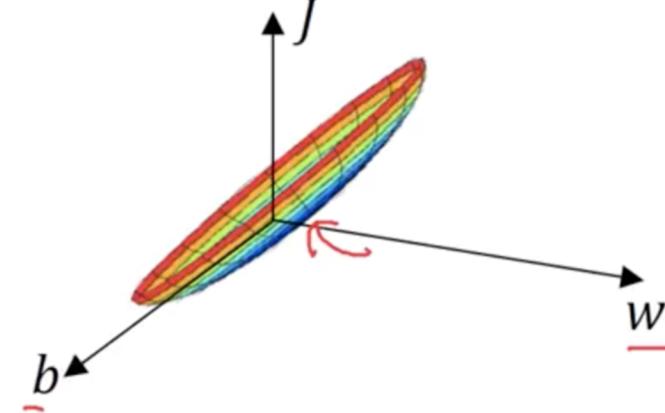
**IMPORTANTE:** La media y el desvío son siempre calculados usando sólo los datos de training, pero todos los datos (test y validación también) son normalizados.

Imagen extraída de <http://cs231n.github.io/neural-networks-2/>

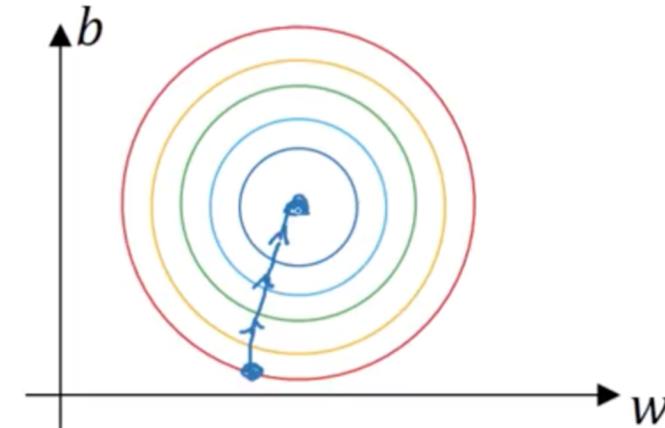
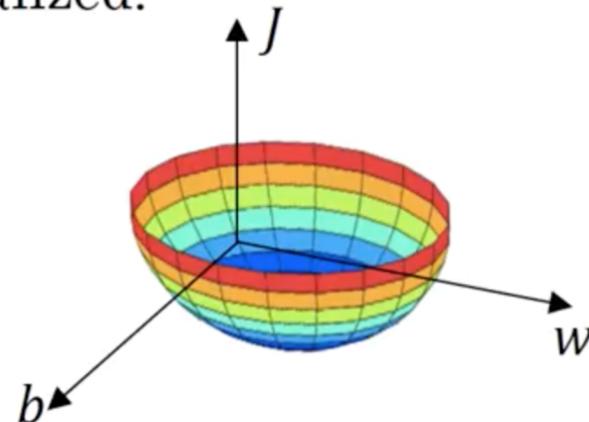
# Normalización de las entradas

Por qué normalizar los datos?

Unnormalized:



Normalized:



$$J(w, b) = \frac{1}{N} \sum_{n=1}^N \mathcal{L}((\mathbf{x}, d)_n; w, b)$$

# Normalización de las entradas

En análisis de imágenes

- El rango de las "features" (pixeles) es el mismo (0-255), por lo tanto, en general no es necesario dividir por el desvío.
- También es usual considerar el valor de **la media y desvío** calculado sobre **todos los pixeles de todas las imágenes** de entrenamiento, en lugar de considerarlo por feature (pixel).
- En imágenes multi-canal (p. ej. RGB), cada canal suele ser normalizado independientemente (una media y desvío por cada canal).

---

# **Batch Normalization**

---

# Normalización en capas ocultas: Batch Normalization (2015)

## Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe  
Google Inc., sioffe@google.com

Christian Szegedy  
Google Inc., szegedy@google.com

### Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout.

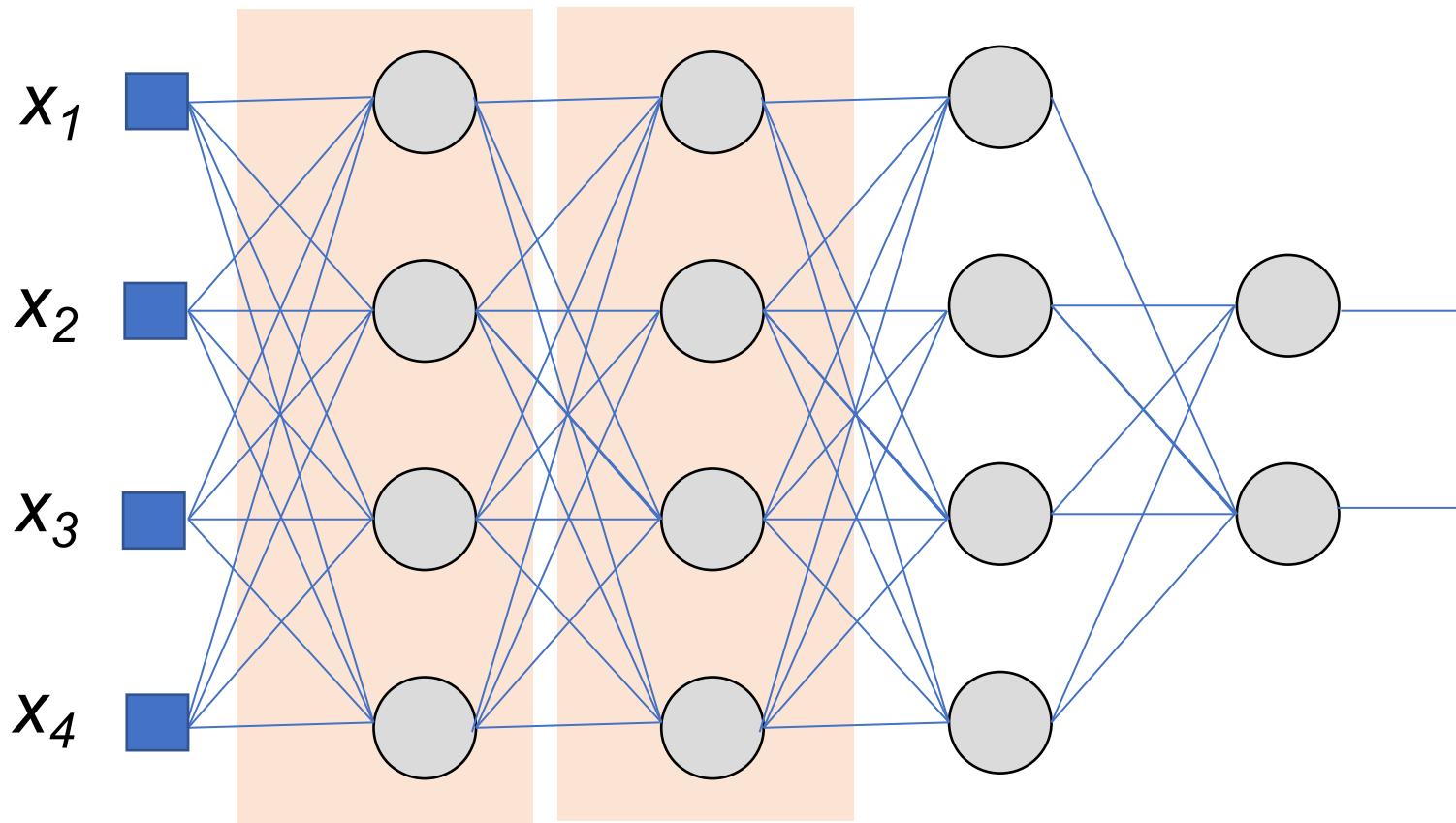
Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than  $m$  computations for individual examples, due to the parallelism afforded by the modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

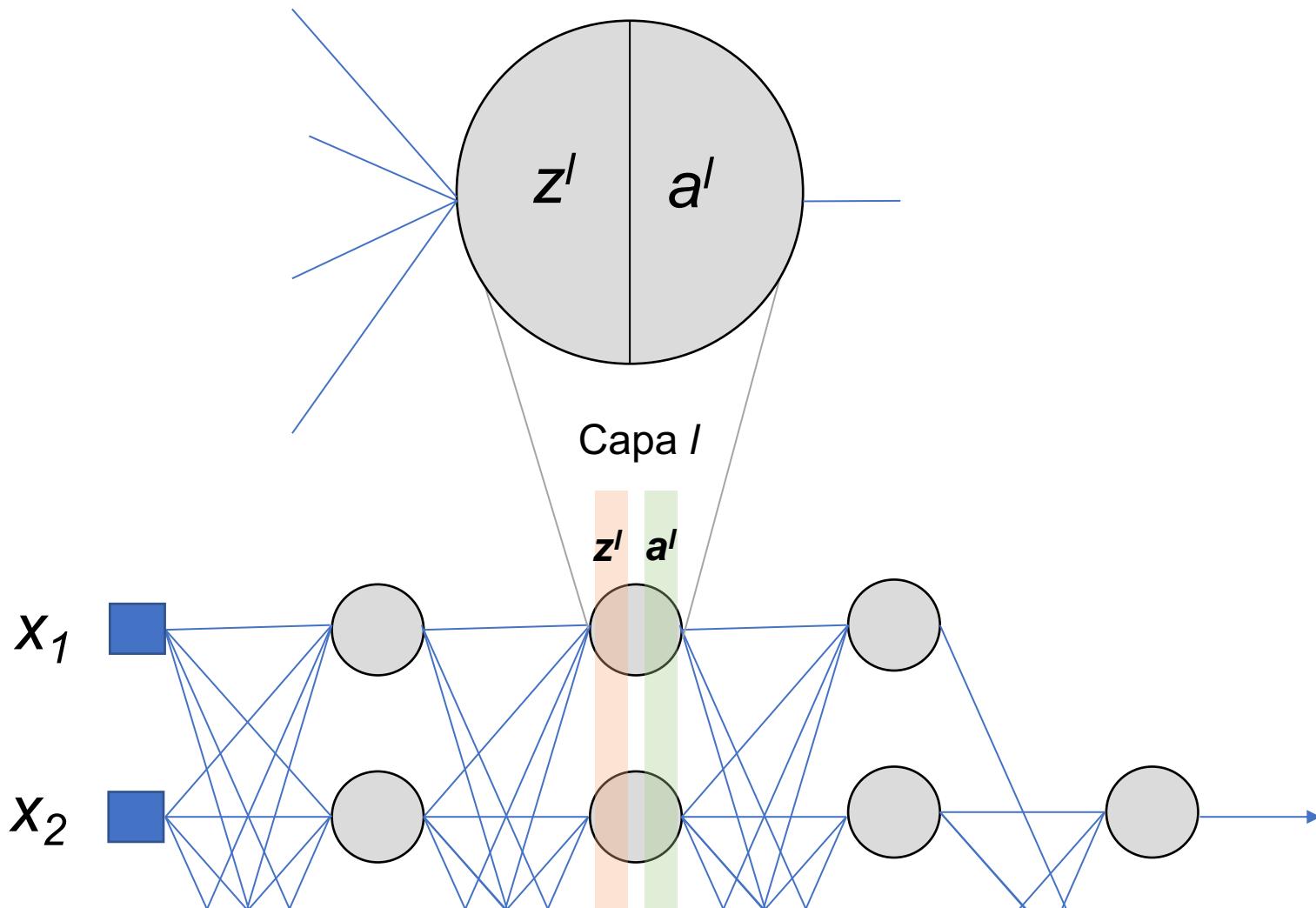
The change in the distributions of layers' inputs creates a problem because the layers need to continu-

# Normalización en capas ocultas: Batch Normalization

**Idea:** Ya que la normalización de los datos de entrada ayuda al entrenamiento, pq no normalizar las entradas de todas las capas ocultas?



# Normalización en capas ocultas: Batch Normalization



# Batch Normalization

Dados  $N$  training samples en un minibatch  $\square$  Batch Norm en la capa  $l$ :

**Vectores de tamaño** = Cantidad de neuronas en la capa  $l$  (ej: 4)

1. Media del minibatch

$$\mu = \frac{1}{N} \sum_{n=1}^N \mathbf{z}_n$$

2. Varianza del minibatch

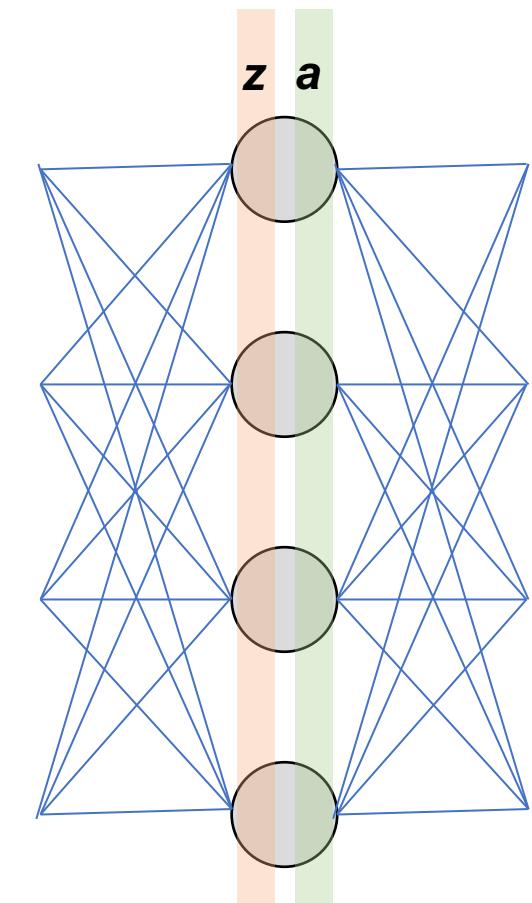
$$\sigma^2 = \frac{1}{N} \sum_{n=1}^N (\mathbf{z}_n - \mu)^2$$

3. Normalizo

$$\hat{\mathbf{z}}_n = \frac{\mathbf{z}_n - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad \text{Evita inestabilidad}$$

4. Re-escalo

$$\mathbf{z}_n^{BN} = \gamma \hat{\mathbf{z}}_n + \beta$$



Parámetros entrenables (por capa)

# Batch Normalization

Dados  $N$  training samples en un minibatch  $\square$  Batch Norm en la capa  $l$ :

1. Media del minibatch

$$\mu = \frac{1}{N} \sum_{n=1}^N \mathbf{z}_n$$

Se computan a partir de los datos de cada minibatch

2. Varianza del minibatch

$$\sigma^2 = \frac{1}{N} \sum_{n=1}^N (\mathbf{z}_n - \mu)^2$$

3. Normalizar

$$\hat{\mathbf{z}}_n = \frac{\mathbf{z}_n - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Se van aprendiendo junto al resto de los parámetros de la red usando gradiente descendente.

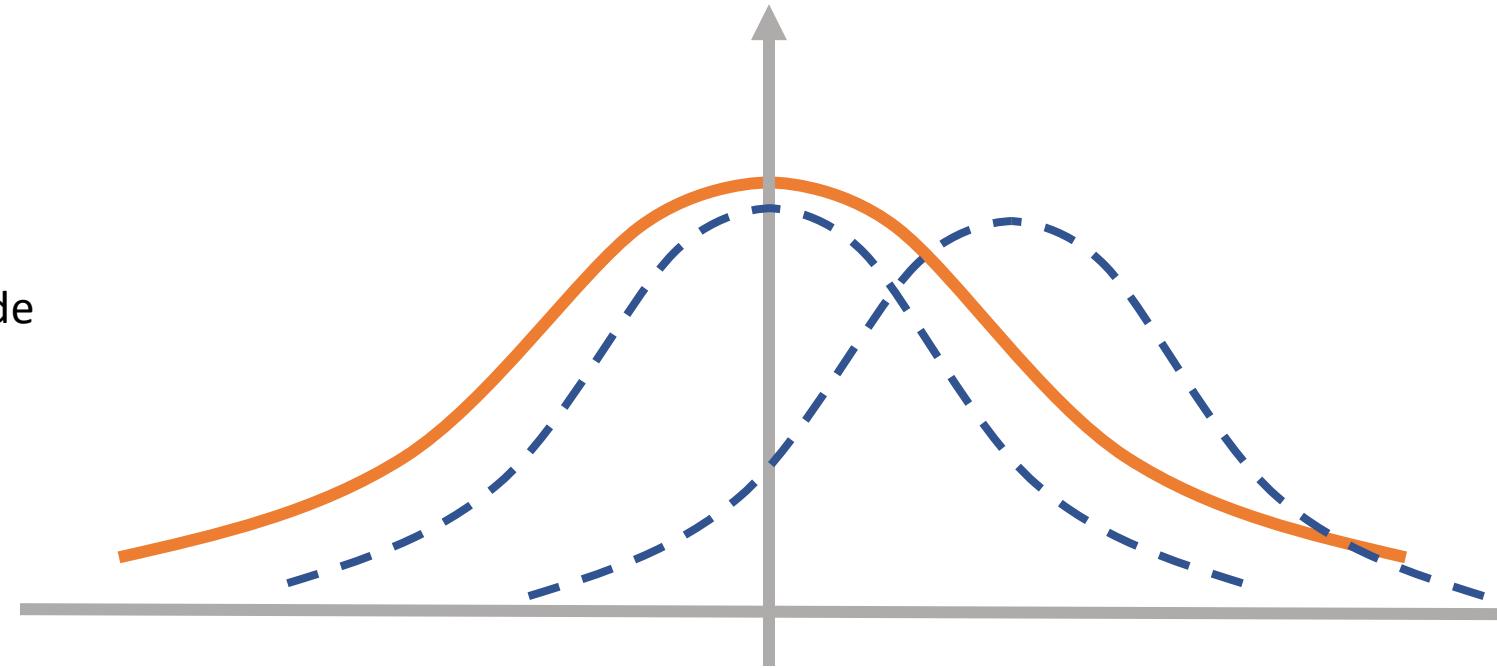
4. Re-escalar

$$\mathbf{z}_n^{BN} = \gamma \hat{\mathbf{z}}_n + \beta$$

# Batch Normalization

Efecto de la operación de re-escalado

Histograma de  
activaciones para una neurona de  
la capa  $l$ , correspondiente a  
un minibatch



$$\hat{z}_n = \frac{z_n - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$



$$z_n^{BN} = \gamma \hat{z}_n + \beta$$

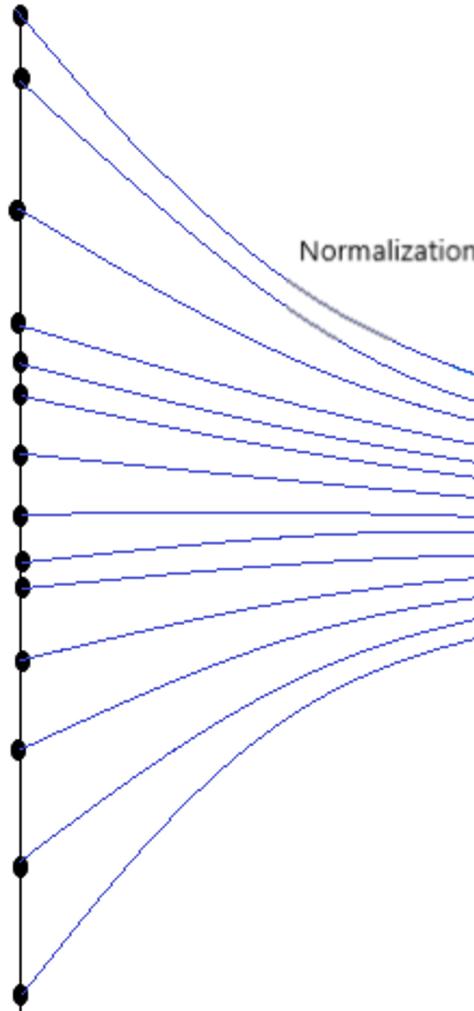
# Efectos obtenidos al aplicar Batch Norm

- **Rango de valores de salida más estable** en las capas ocultas  
En consecuencia, simplifica el proceso de aprendizaje de los pesos de dichas capas
- **Efecto regularizador:** el cálculo de la media y varianza es por minibatch  
Esto introduce ruido al proceso de entrenamiento, produciendo un pequeño efecto de regularización

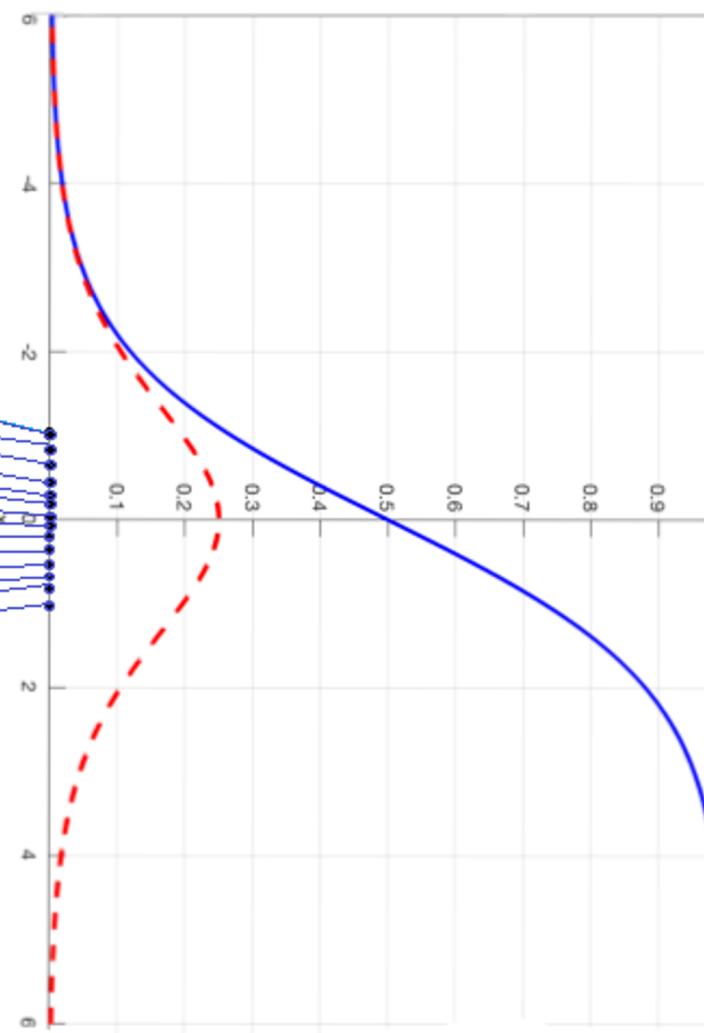
*"The regularization effect is a result of the fact that **an individual sample and its downstream activations are always seen by the model as shifted and scaled**, depending on the statistics across the randomly extracted minibatch. This is in itself a form of principled augmentation."*

# Efectos obtenidos al aplicar Batch Norm

Activation Inputs



Sigmoid Activation and Gradient



Permite usar funciones de activación con saturación sin caer en el problema del gradiente que se desvanece

# Batch Normalization

- En capas **fully connected**, se utiliza una media y varianza por cada neurona de *la capa*.

$\mu, \sigma^2, \gamma, \beta$  **son vectores** de tamaño [Número de neuronas]

- En **capas convolucionales**, dado que los filtros son compartidos por todo el *feature map*, se utiliza una única media y varianza por cada canal de entrada para normalizar.

$\mu, \sigma^2, \gamma, \beta$  **son escalares** (distintos por cada canal de entrada)

Ejemplos en StackOverflow:

<https://stackoverflow.com/questions/38553927/batch-normalization-in-convolutional-neural-network>

# Batch Normalization en Test Time

Normalizar       $\hat{\mathbf{z}}_n = \frac{\mathbf{z}_n - \mu}{\sqrt{\sigma^2 + \varepsilon}}$       Re-escalar       $\mathbf{z}_n^{BN} = \gamma \hat{\mathbf{z}}_n^+ \beta$

- Por cada capa  $l$  usamos los parámetros aprendidos:  $\gamma_l, \beta_l$
- Tenemos solo un data sample  $\bowtie$  Qué usamos para  $\mu, \sigma^2$ ?

**La media móvil exponencial de  $\mu, \sigma^2$   
acumulada durante el entrenamiento**

# Media Móvil Exponencial

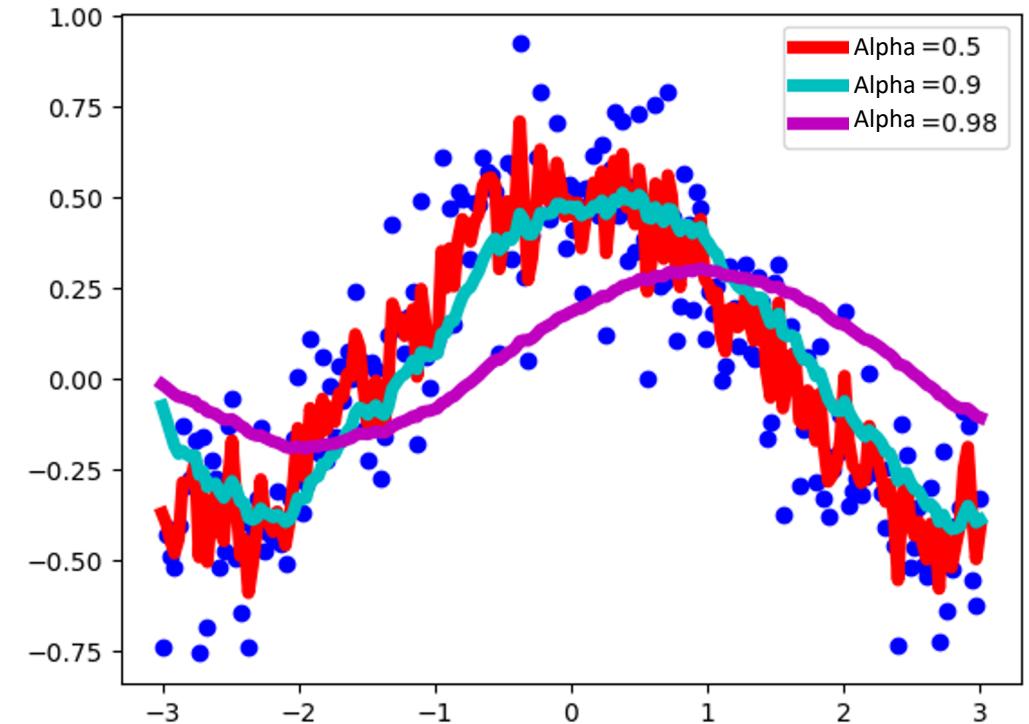
Ej: Dada una serie de valores

$$\hat{\mu}_t = \alpha \hat{\mu}_{t-1} + (1 - \alpha) \mu_t$$

Ej:

$$\hat{\mu}_t = 0.9 \hat{\mu}_{t-1} + 0.1 \mu_t$$

$$\mu_1, \mu_2, \dots, \mu_k$$



---

# **Variantes del gradiente descendiente**

---

# Variantes del gradiente descendiente

- **Gradiente descendiente clásico**
- **Gradiente descendiente estocástico**
- **Gradiente descendiente por mini-batches**

# Variantes del gradiente descendiente

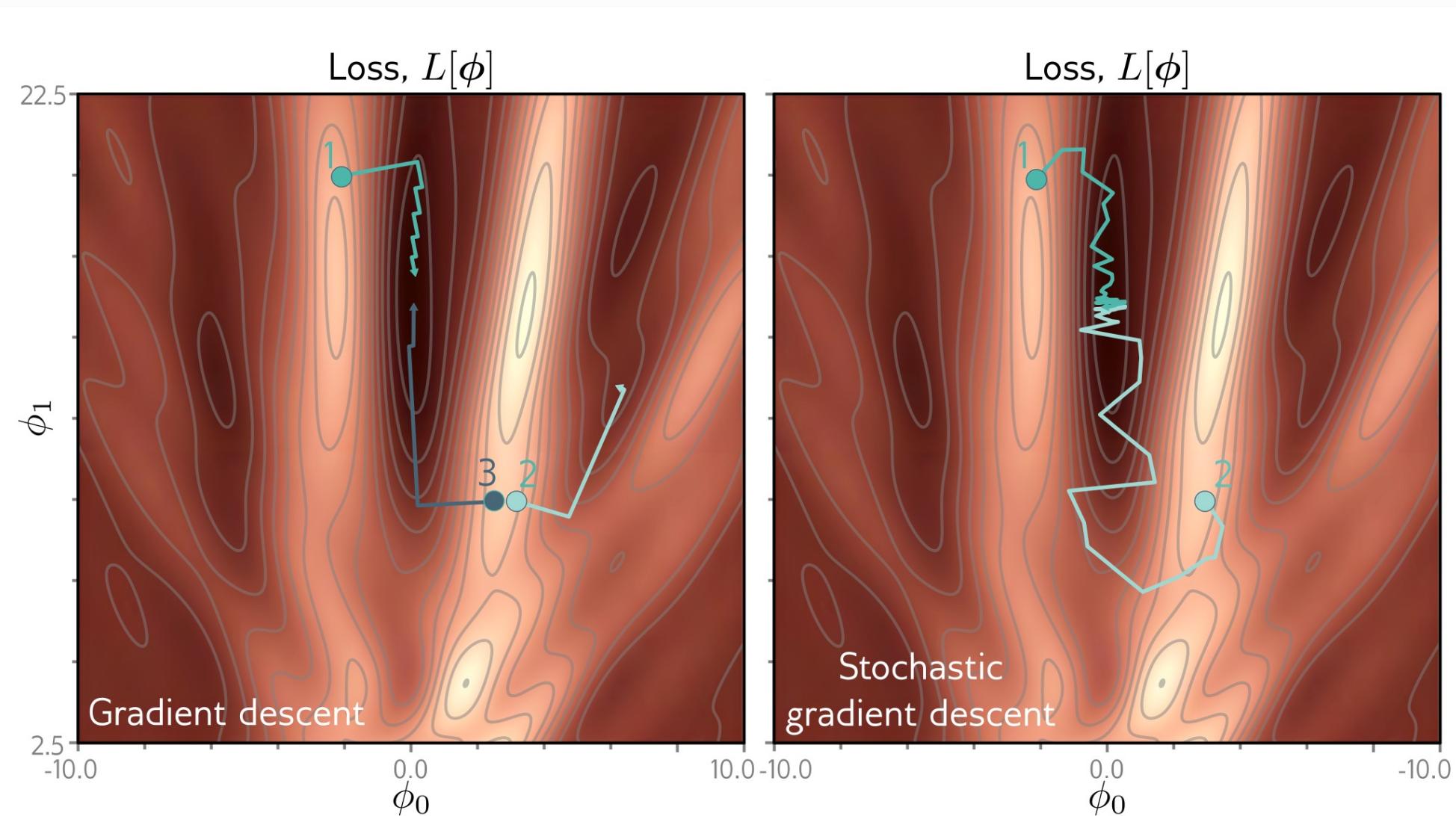
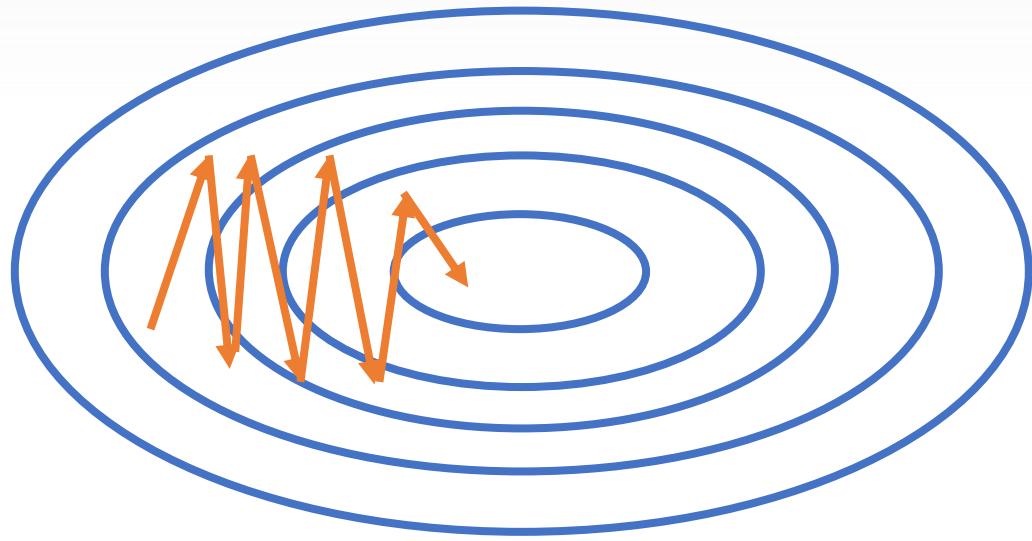


Figura extraída de (Prince, 2023)

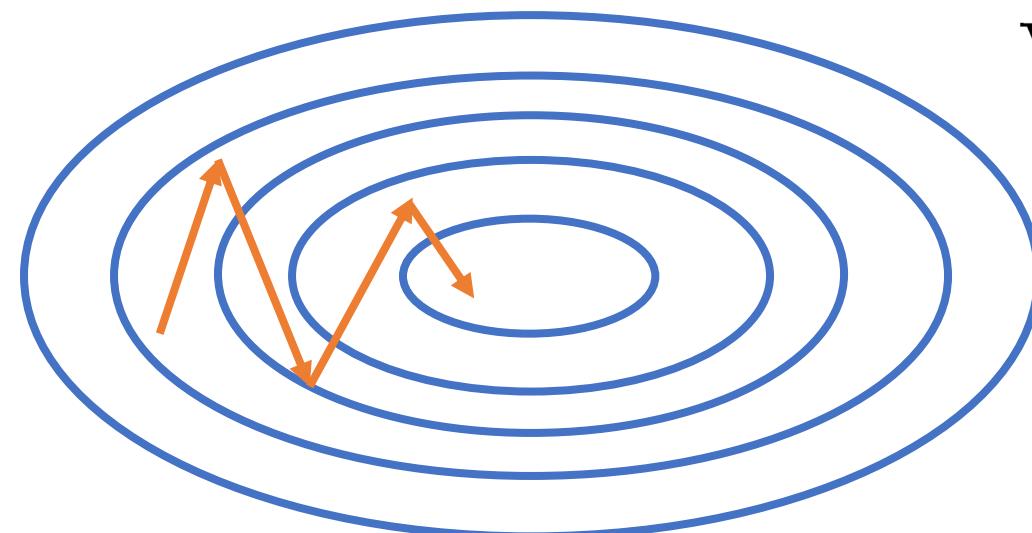
# Gradiente descendiente con Momentum

Sin momentum



$$\mathbf{W} = \mathbf{W} - \delta \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W})$$

Con momentum



Alfa suele tomarse = 0.9 (promedio 10 últimos)

$$\mathbf{V}_t = \boxed{\alpha} \mathbf{V}_{t-1} + (1 - \alpha) \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W})$$

$$\mathbf{W} = \mathbf{W} - \delta \mathbf{V}_t$$

$$\mathbf{V}_0 = \vec{0}$$

# Gradiente descendiente con Momentum

- Suaviza los pasos dados por el gradiente descendiente en las dimensiones más oscilantes
- Acelera la convergencia del método

# Gradiente descendiente con Momentum

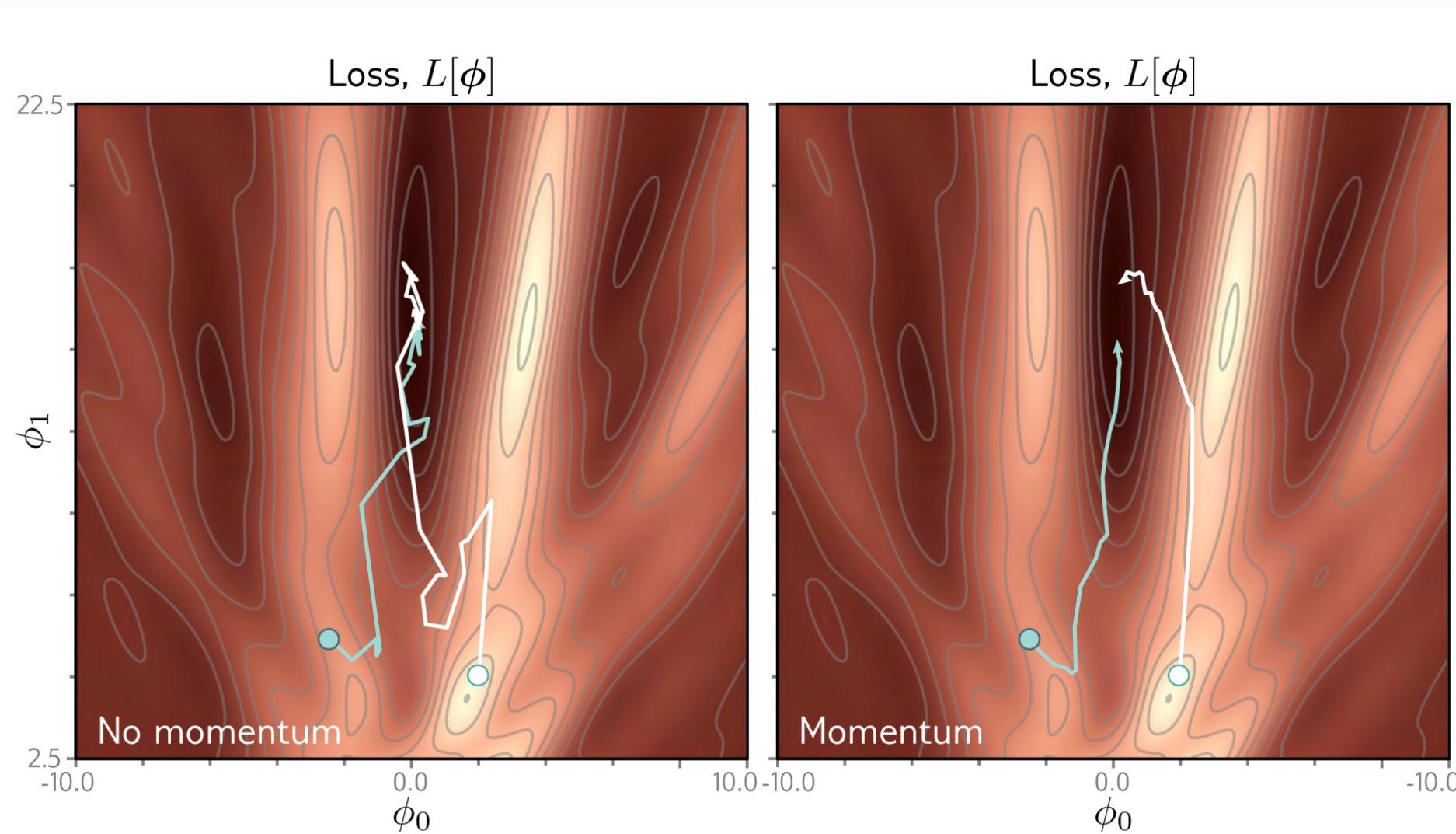


Figura extraída de (Prince, 2023)

# Variando el learning rate

- **LR Alto:** el algoritmo puede oscilar caóticamente
- **LR Bajo:** el algoritmo puede dejar de aprender (o tomar muuucho tiempo)
- **Step Decay:** Reducir el learning rate (ej: dividir por 2) cada una cantidad fija de épocas, o cuando cuando el error en validación no mejora.
- **Exponential Decay:** decaer el paso con una exp negativa     $\delta = \delta_0 e^{-kt}$

# Learning rate adaptativo por parámetro

- Los métodos anteriores aplican el mismo **LR global a todos los parámetros W**

$$\boxed{\delta} \nabla_{\mathbf{w}} \mathcal{L} = \boxed{\delta} \left( \frac{\partial \mathcal{L}}{\partial w_1}, \dots, \frac{\partial \mathcal{L}}{\partial w_D} \right)$$

- Los **métodos adaptativos escalan el LR por cada parámetro**. Ejemplos:
  - RMSProp [http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture\\_slides\\_lec6.pdf](http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf)
  - Adadelta <http://arxiv.org/abs/1212.5701>
  - Adagrad <http://jmlr.org/papers/v12/duchi11a.html>
  - Adam (Momentum + RMSProp): <https://arxiv.org/abs/1412.6980>

# Learning rate adaptativo por parámetro

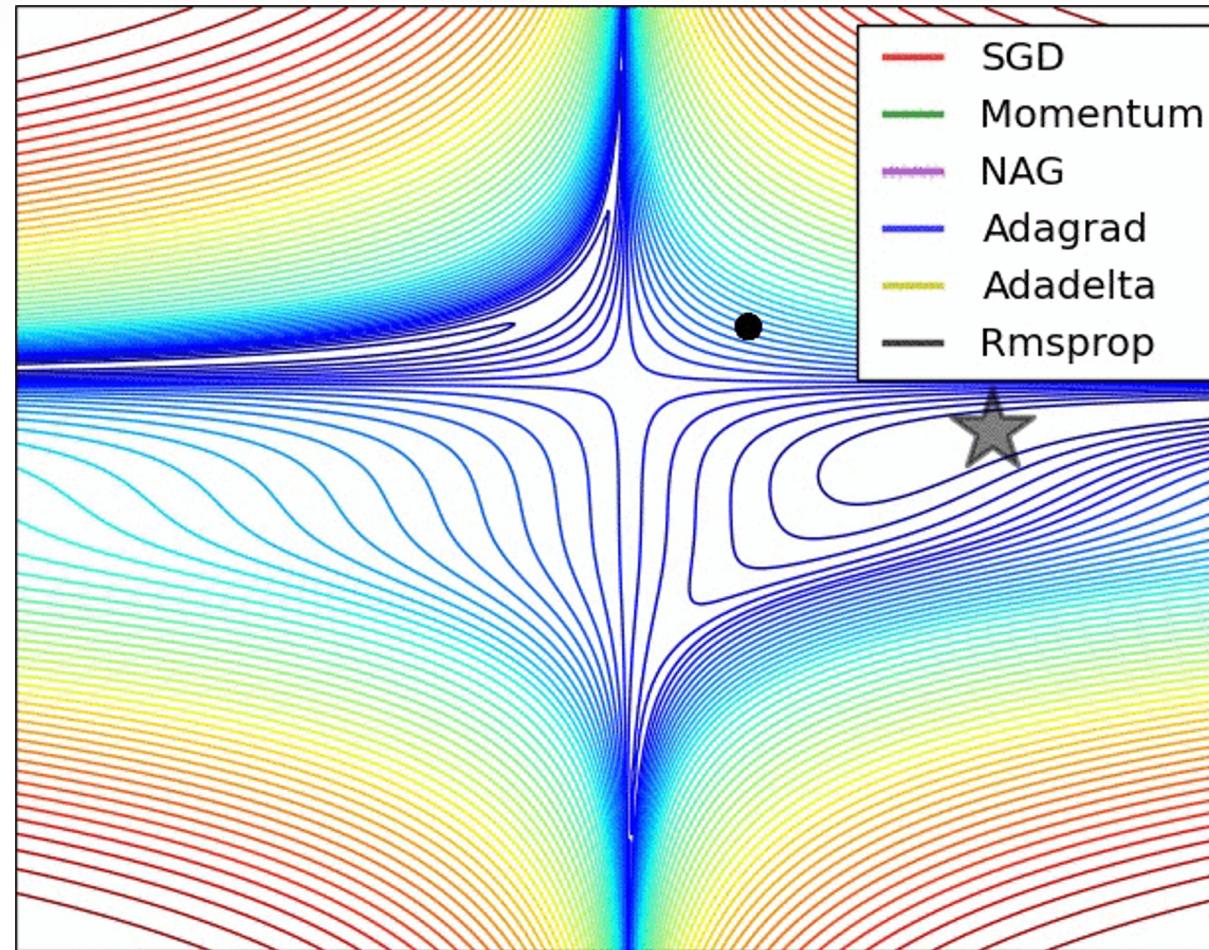


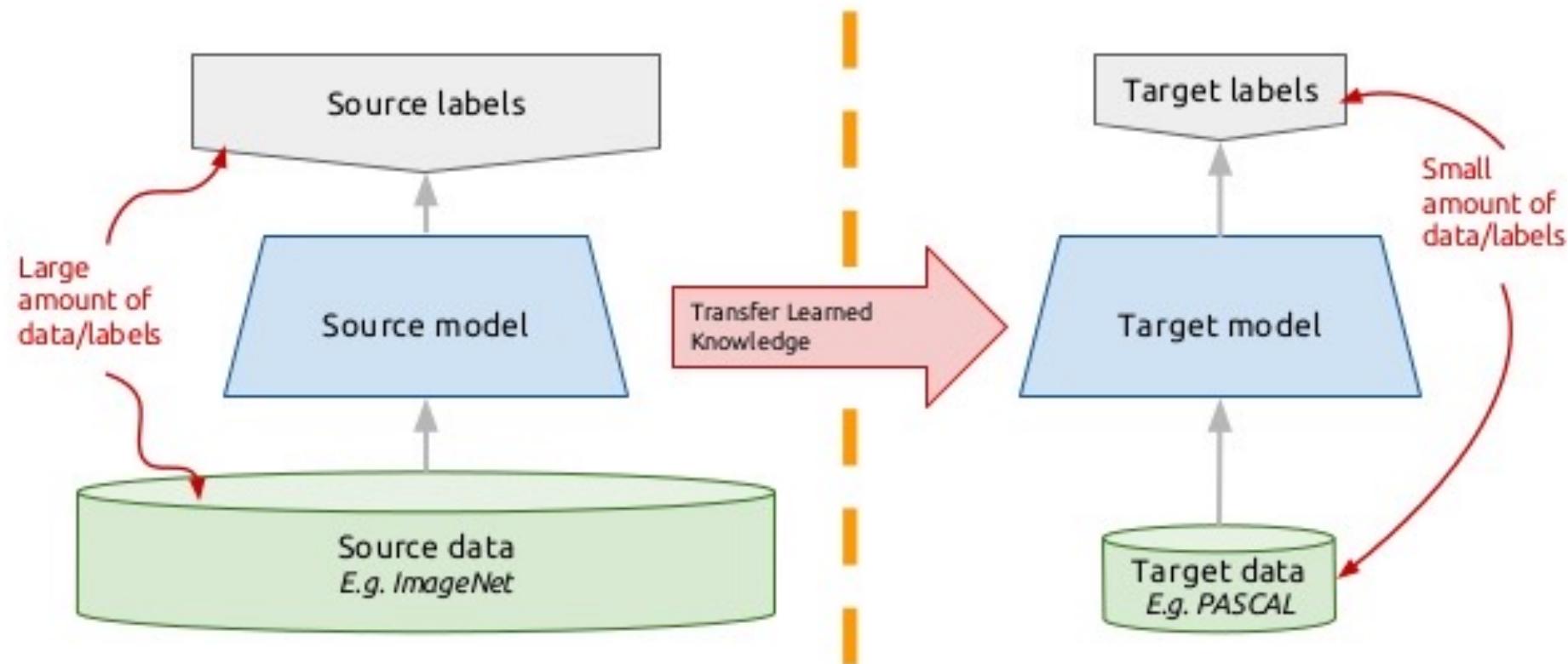
Imagen extraída de : <http://ruder.io/optimizing-gradient-descent/index.html#fn13> (Muy buen review!)

---

# **Variantes del gradiente descendiente**

---

# Transferencia de conocimiento



# Transferencia de conocimiento

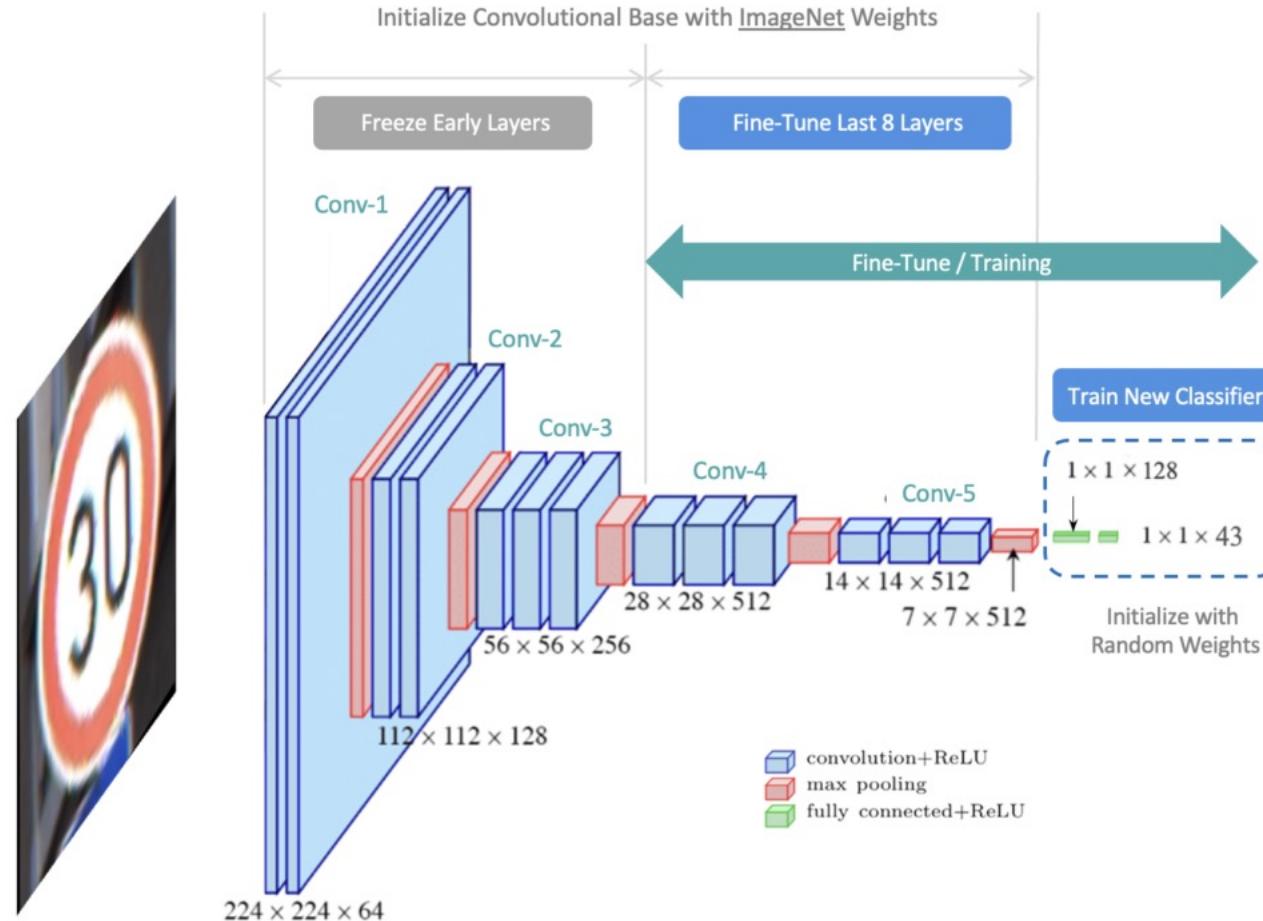


Imagen extraída de <https://learnopencv.com/fine-tuning-pre-trained-models-tensorflow-keras/>

## Clase 4

# Entrenando una red neuronal

Enzo Ferrante

 eferrante@sinc.unl.edu.ar

