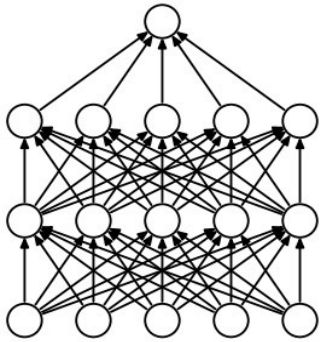
Clase 5

Entrenando una red neuronal

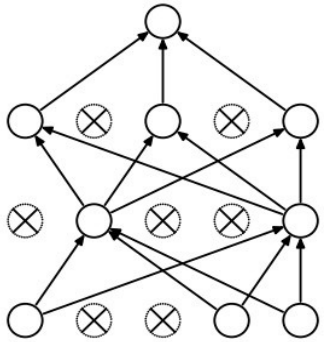
Enzo Ferrante

eferrante@sinc.unl.edu.ar

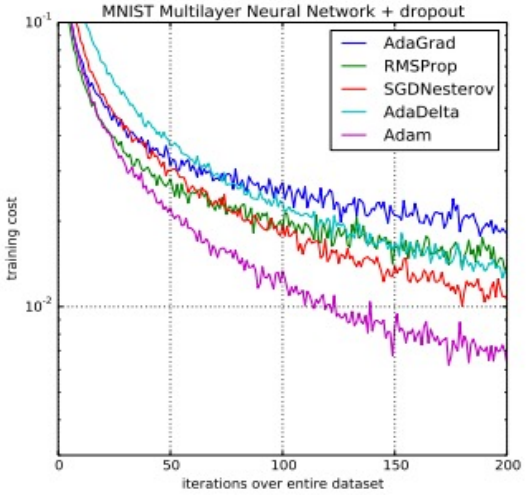
@enzoferrante



(a) Standard Neural Net



(b) After applying dropout.

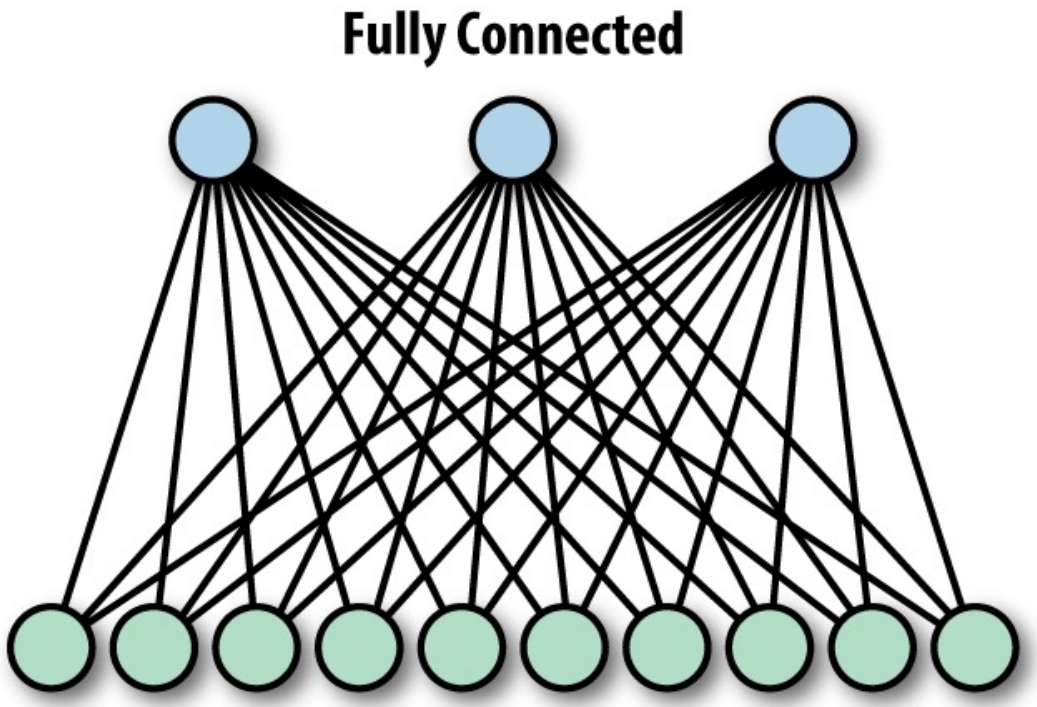


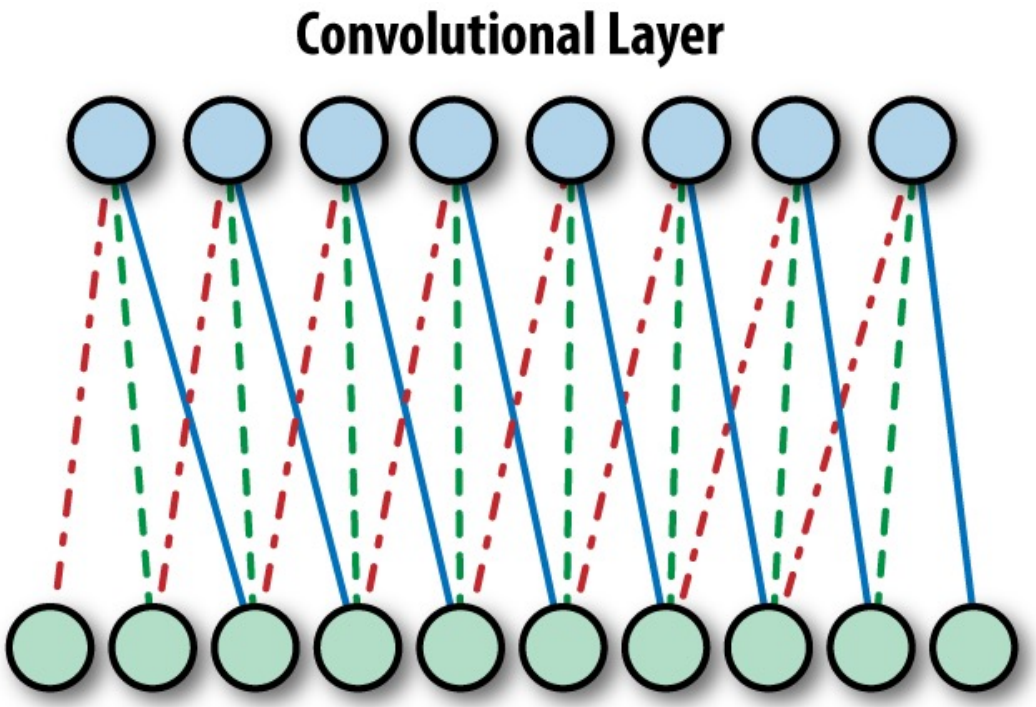
Universidad de SanAndrés

UNL CONICET

s i n c (i)

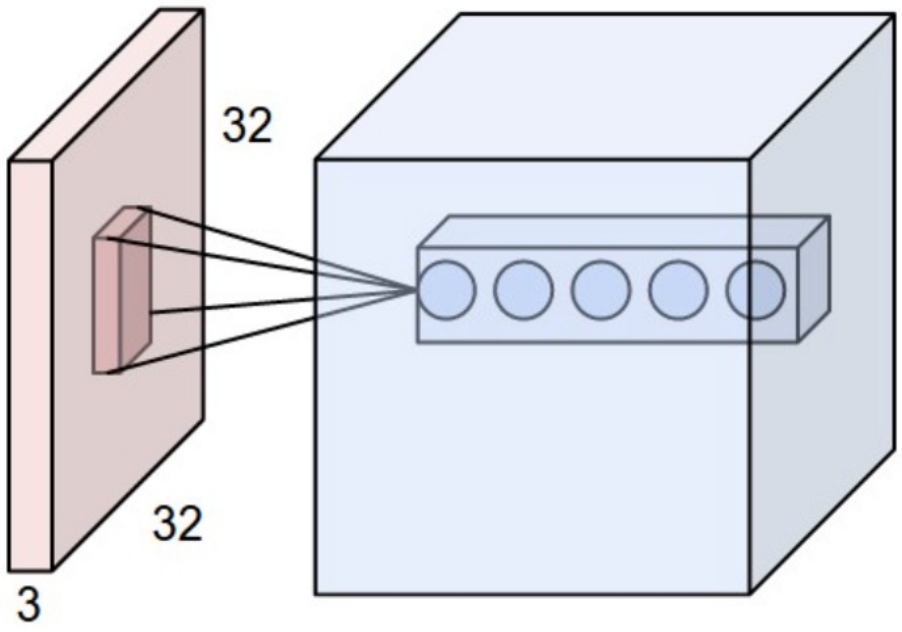
Redes Neuronales Convolucionales

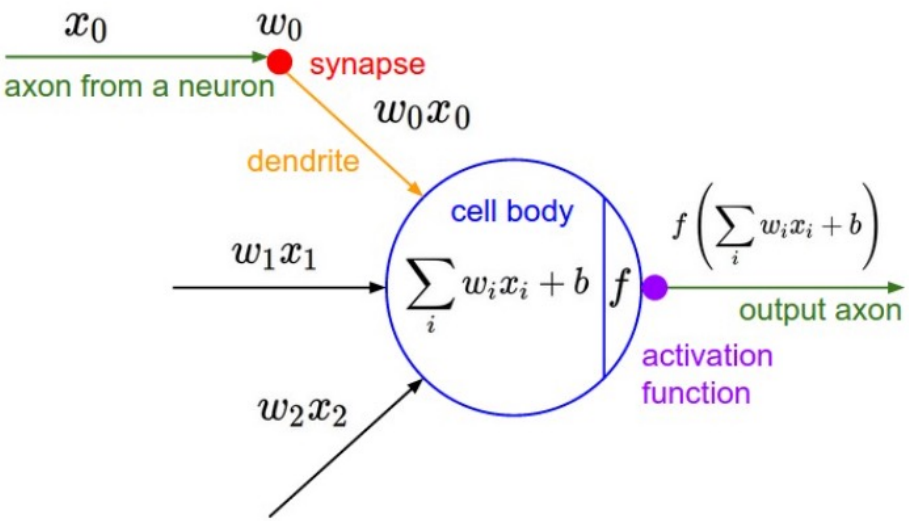




Fuente: https://www.oreilly.com/library/view/learning-tensorflow/9781491978504/ch04.html

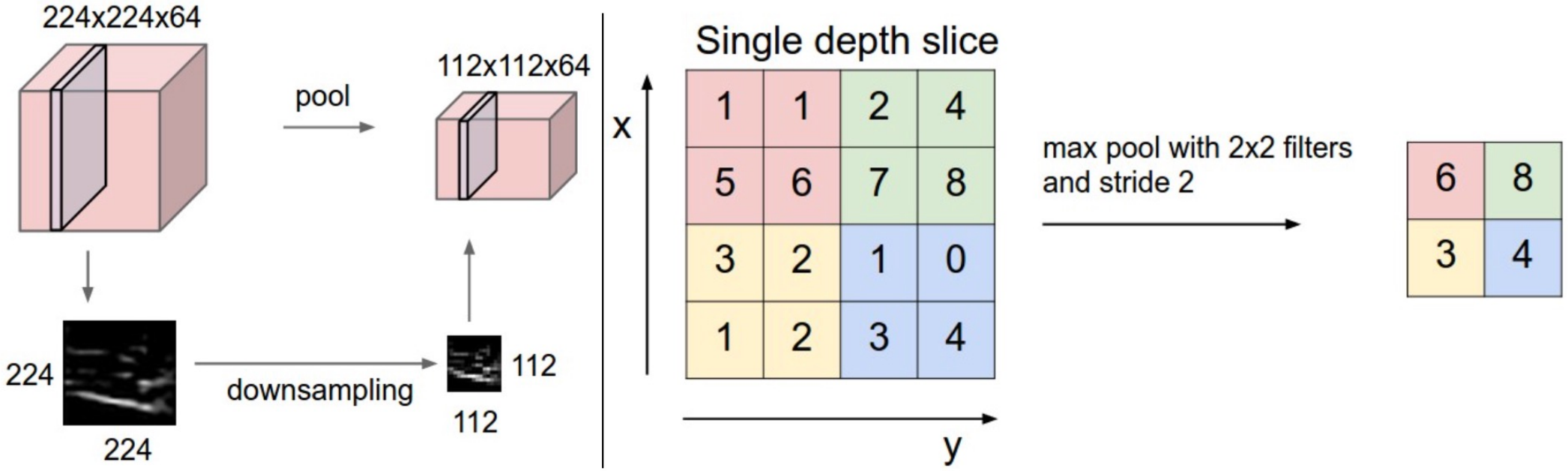
Redes Neuronales Convolucionales





https://cs231n.github.io/convolutional-networks/

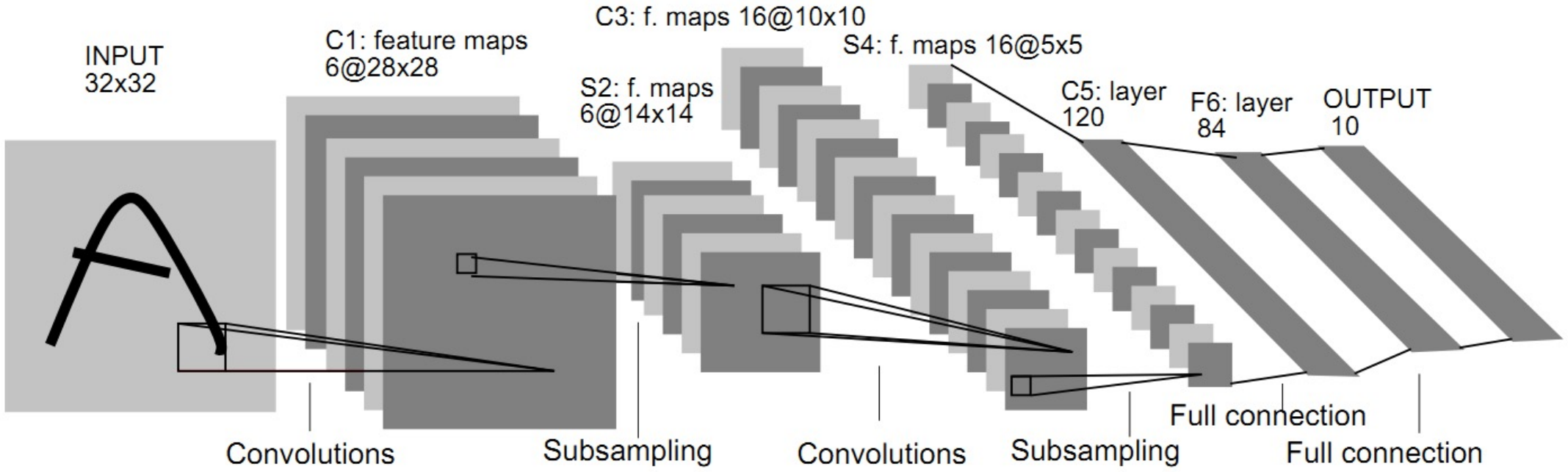
Operaciones de Pooling



https://cs231n.github.io/convolutional-networks/

Redes Neuronales Convolucionales

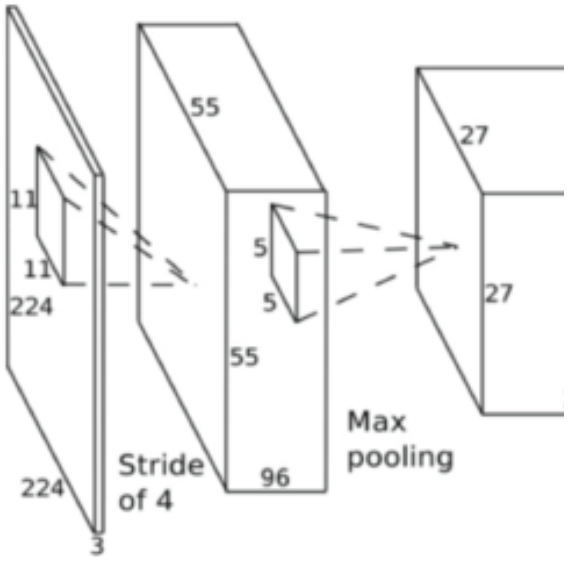
$$O = \frac{(W - F + 2P)}{S} + 1$$

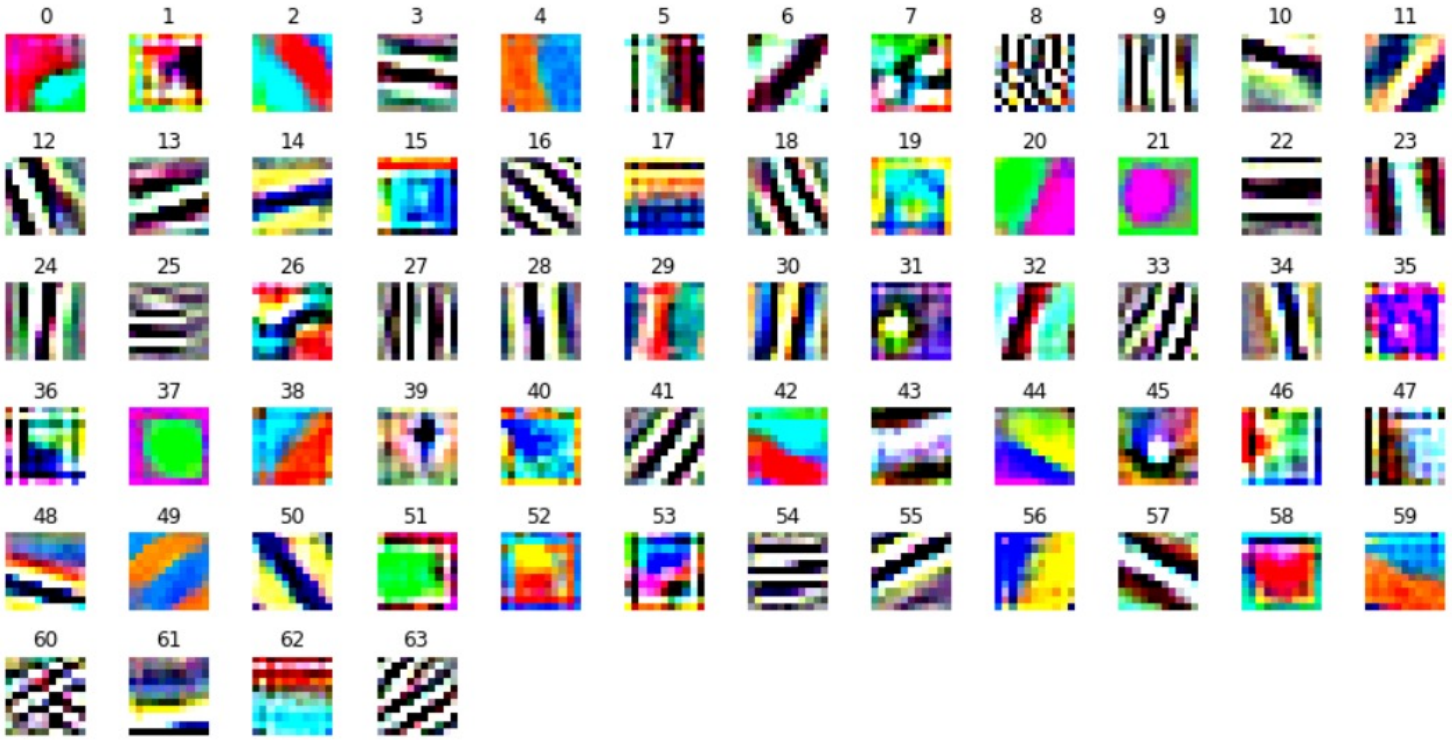


Convolutions	Subsampling	Convolutions	Subsampling
F=5x5	Avg Pooling	F=5x5	Avg Pooling
S=1	F=2x2	S=1	F=2x2
	S=2		S=2

Visualizando los filtros de una CNN entrenada

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

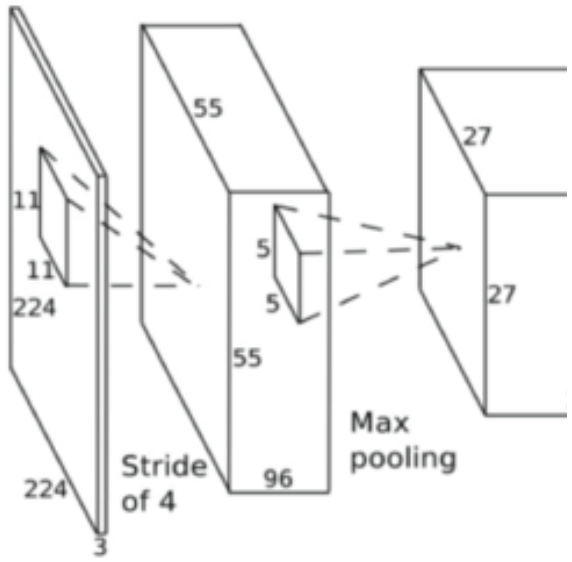


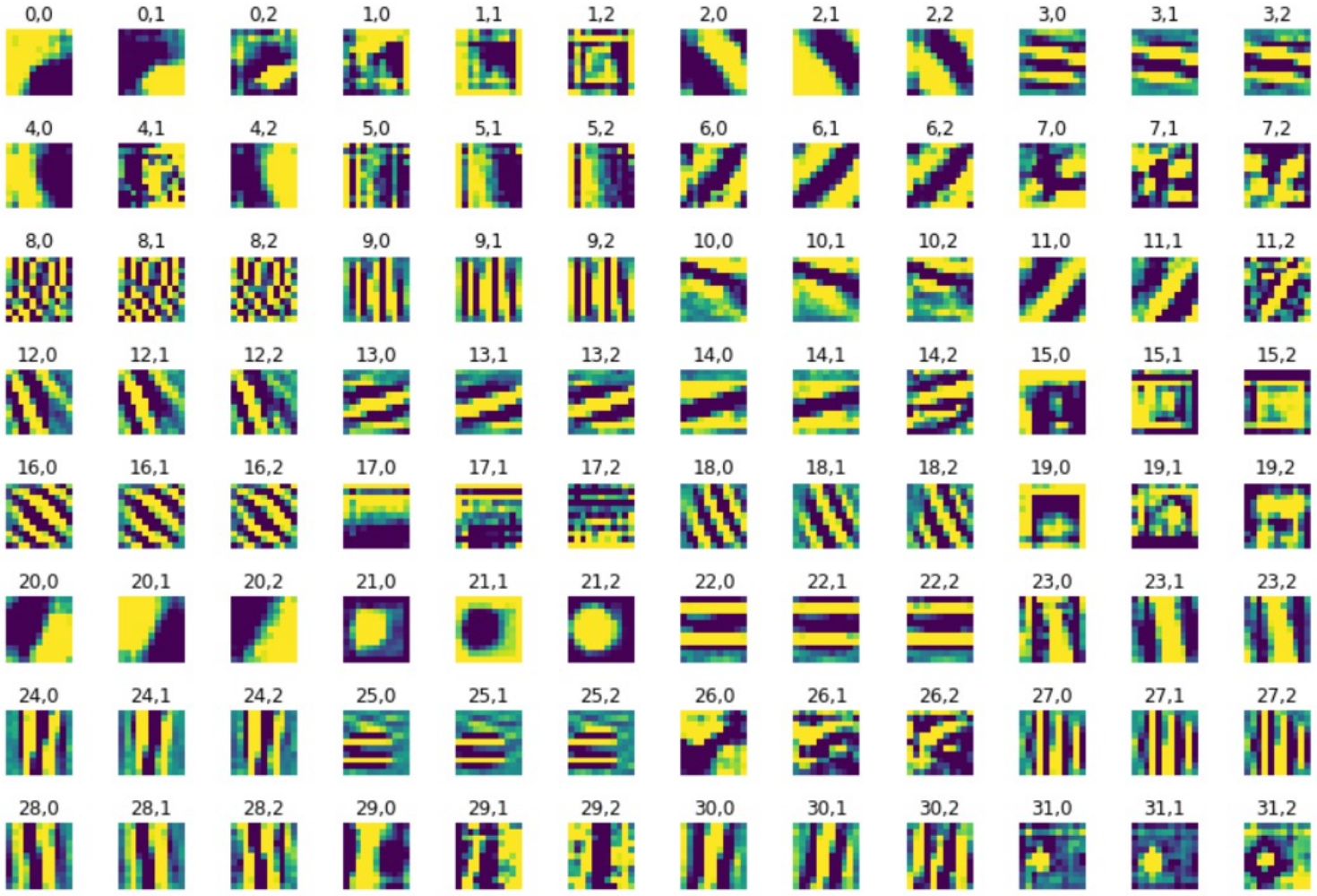


https://towardsdatascience.com/visualizing-convolution-neural-networks-using-pytorch-3dfa8443e74e

Visualizando los filtros de una CNN entrenada

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



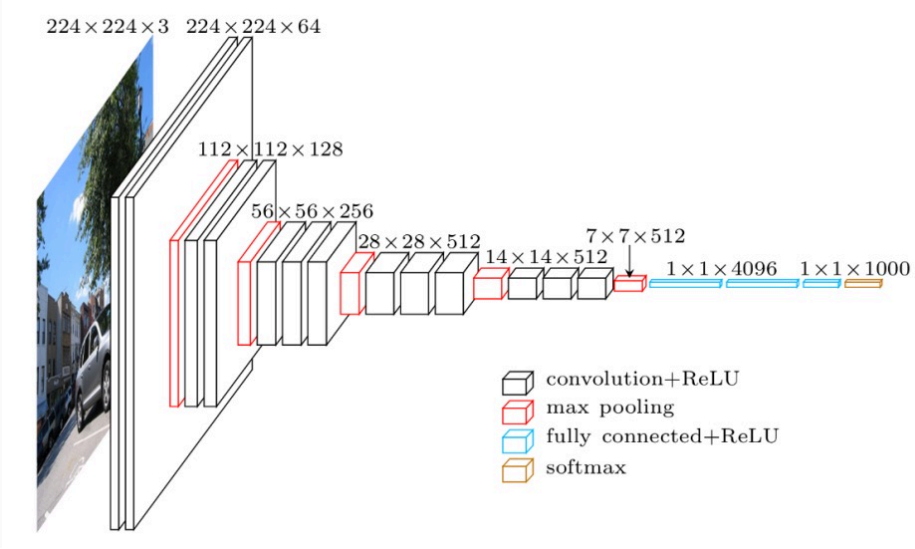


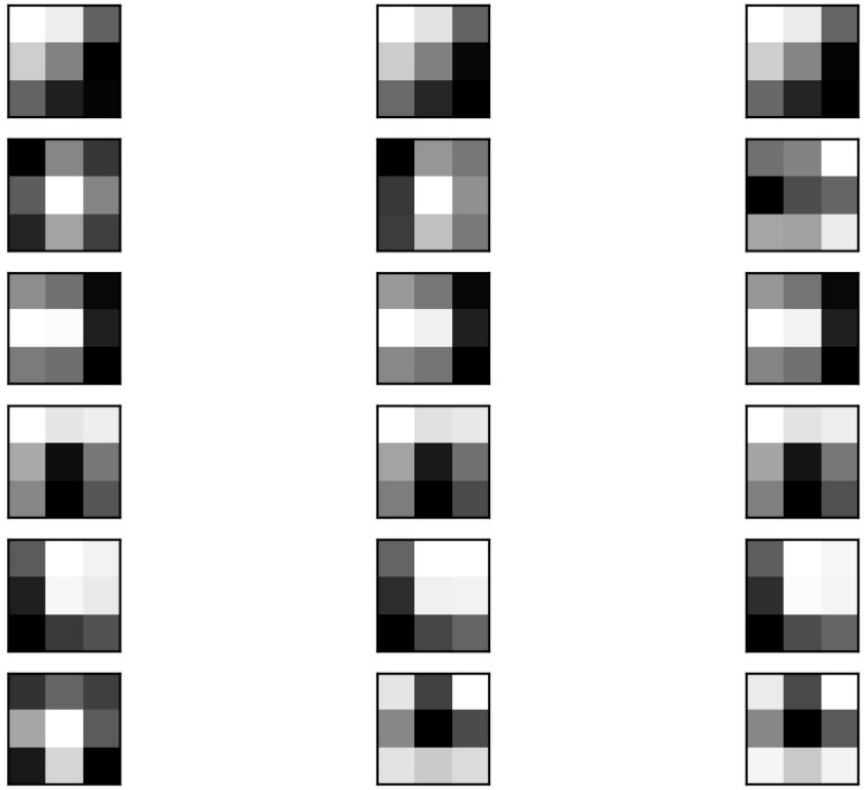
https://towardsdatascience.com/visualizing-convolution-neural-networks-using-pytorch-3dfa8443e74e

Visualizando los filtros de una CNN entrenada

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

Kernels de una red convolucional, que me generaran los mapas de caracteristicas o feature maps (las features de las CNN)
FILTROS / KERNELS:

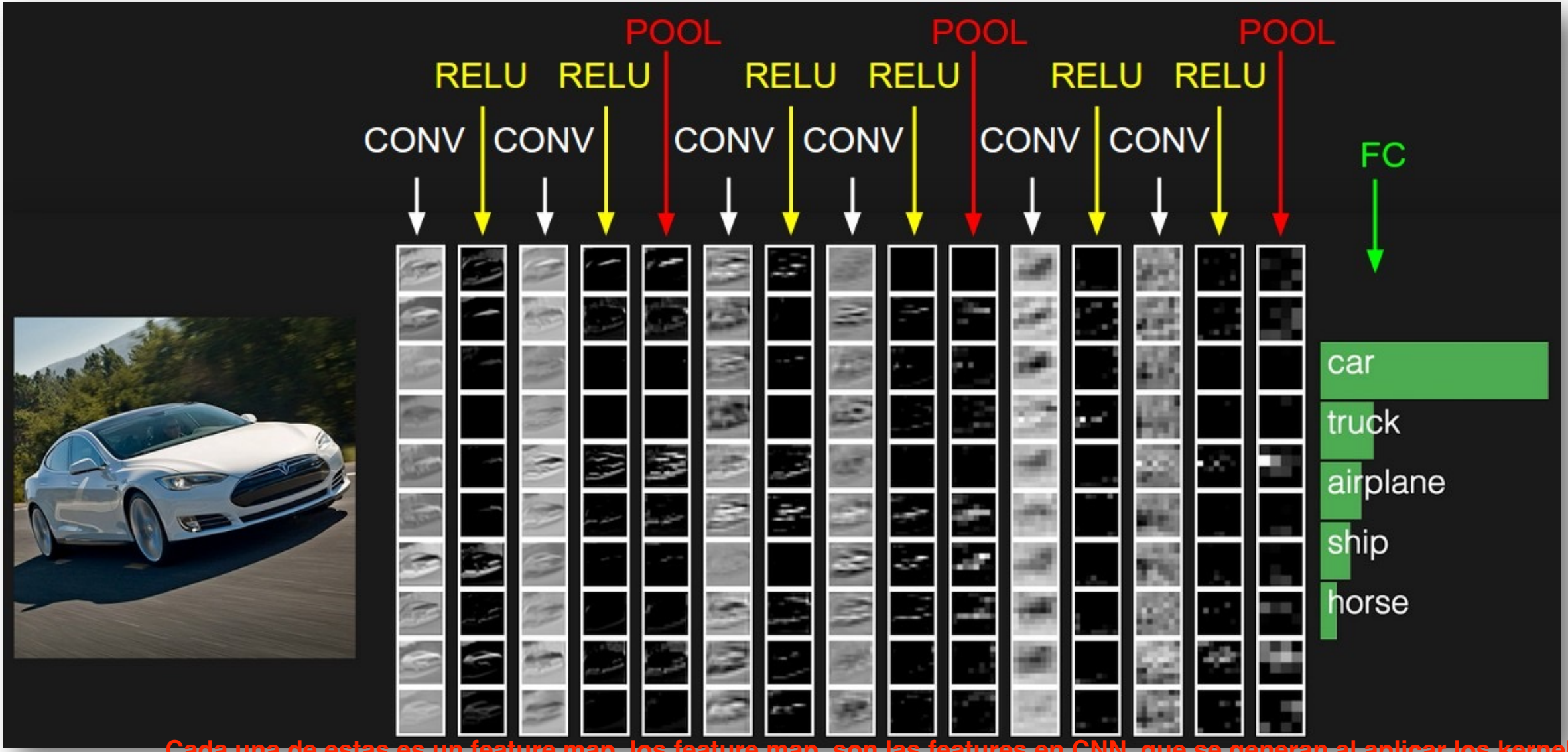




Plot of the First 6 Filters From VGG16 With One Subplot per Channel

https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/

Visualizando los feature maps de una CNN entrenada



Cada una de estas es un feature map, los feature map, son las features en CNN, que se generan al aplicar los kernels
Tengo 10 kernels en la primer capa, ya que N feature maps = N kernels

https://cs231n.github.io/convolutional-networks/

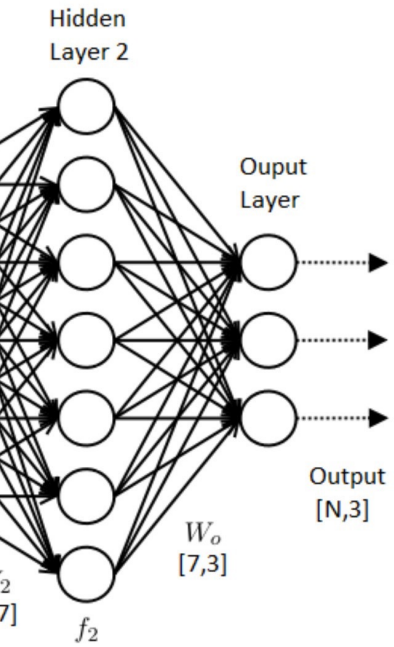
Entrenando una red neuronal para clasificación

Salida softmax

Softmax como función activación de salida

Permite obtener una "interpretación probabilística" de la salida

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



$$\begin{pmatrix} 1.2 \\ 0.9 \\ 0.4 \end{pmatrix} \xrightarrow{\textbf{softmax}} \begin{pmatrix} \dfrac{e^{1.2}}{e^{1.2}+e^{0.9}+e^{0.4}} \\ \dfrac{e^{0.9}}{e^{1.2}+e^{0.9}+e^{0.4}} \\ \dfrac{e^{0.4}}{e^{1.2}+e^{0.9}+e^{0.4}} \end{pmatrix} = \begin{pmatrix} 0.46 \\ 0.34 \\ 0.20 \end{pmatrix}$$

Función de pérdida

Función de pérdida

Entropía Cruzada entre dos distribuciones de probabilidad p y q

Distribución Ground Truth

$$CE(p,q) = -\sum_{x} p(x) \log q(x)$$

Distribución Estimada

Categoría	Etiqueta
Gato	1
Perro	2
Vaca	3

Notación One hot		
1	0	0
0	1	0
0	0	1

Función de pérdida

Entropía Cruzada vs Accuracy

$$\text{Pred} \quad \begin{pmatrix} 0{,}1 \\ 0{,}2 \\ \boxed{0{,}7} \end{pmatrix} \begin{pmatrix} 0{,}3 \\ \boxed{0{,}4} \\ 0{,}3 \end{pmatrix} \begin{pmatrix} 0{,}3 \\ 0{,}3 \\ \boxed{0{,}4} \end{pmatrix}$$

$$\text{GT} \quad \begin{pmatrix} 1 \\ 0 \\ \boxed{0} \end{pmatrix} \begin{pmatrix} 0 \\ \boxed{1} \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ \boxed{1} \end{pmatrix}$$

Tasa de acierto = 2/3

Entropía cruzada = 4.14

$$\text{Pred} \quad \begin{pmatrix} 0{,}3 \\ \boxed{0{,}4} \\ 0{,}3 \end{pmatrix} \begin{pmatrix} 0{,}1 \\ \boxed{0{,}7} \\ 0{,}2 \end{pmatrix} \begin{pmatrix} 0{,}1 \\ 0{,}2 \\ \boxed{0{,}7} \end{pmatrix}$$

$$\text{GT} \quad \begin{pmatrix} 1 \\ \boxed{0} \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ \boxed{1} \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ \boxed{1} \end{pmatrix}$$

Tasa de acierto = 2/3

Entropía cruzada = 1.92

Funciones de activación

Funciones de activación

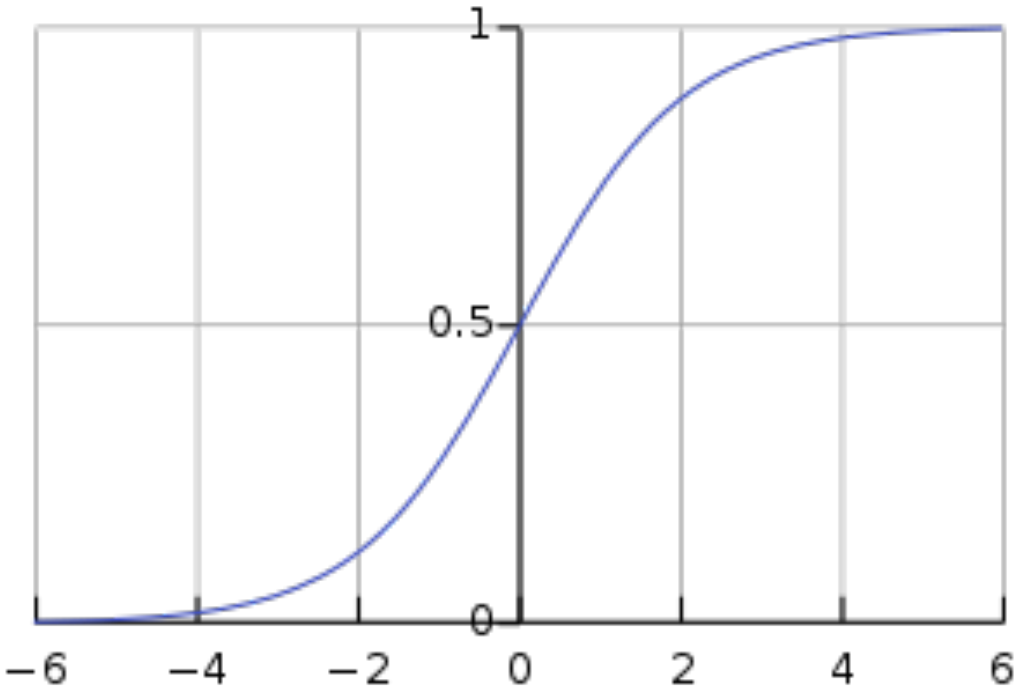
Fully Connected Layer

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

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 ∞ y -∞
- La múltiplicación de pequeños gradientes por la regla de la cadena hace que el **gradiente se desvanezca**

Función de activación Sigmoidea

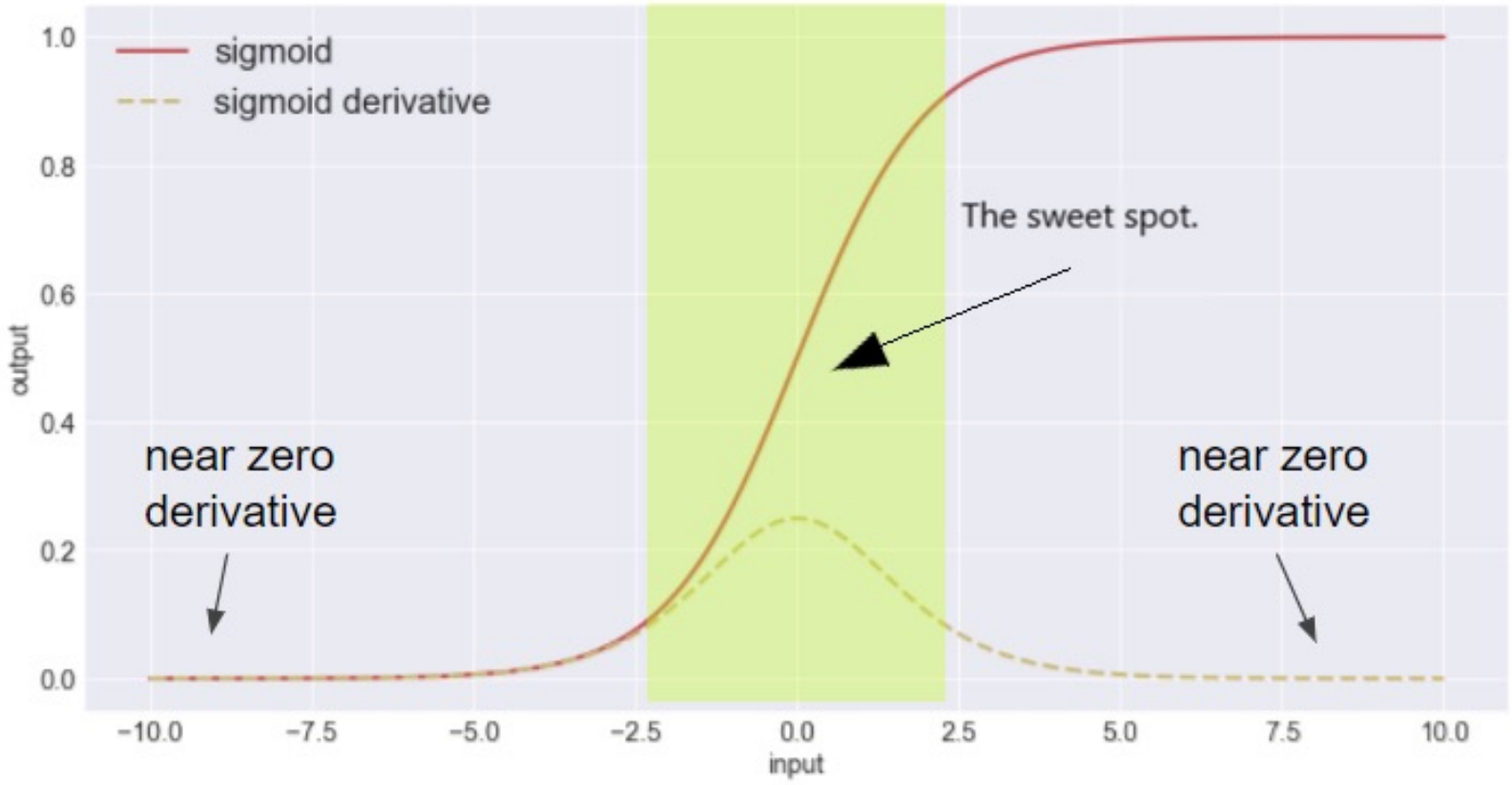
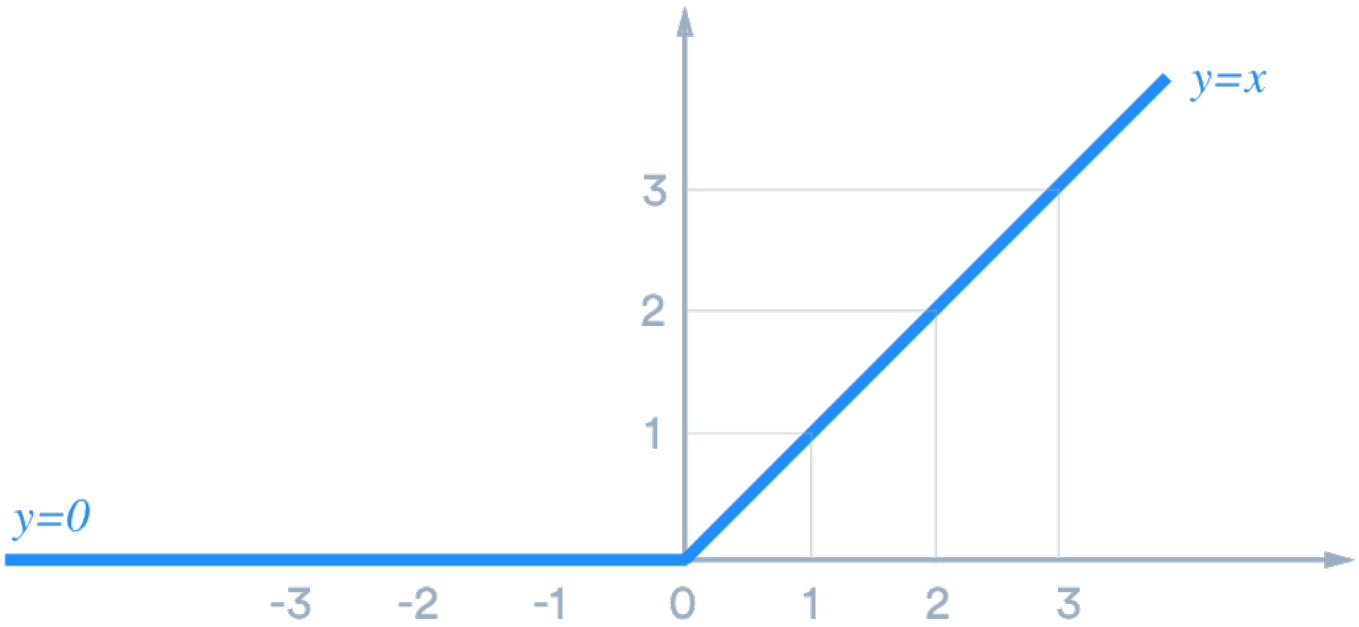


Imagen extraida de: https://towardsdatascience.com/intuit-and-implement-batch-normalization-c05480333c5b

Función de activación ReLu

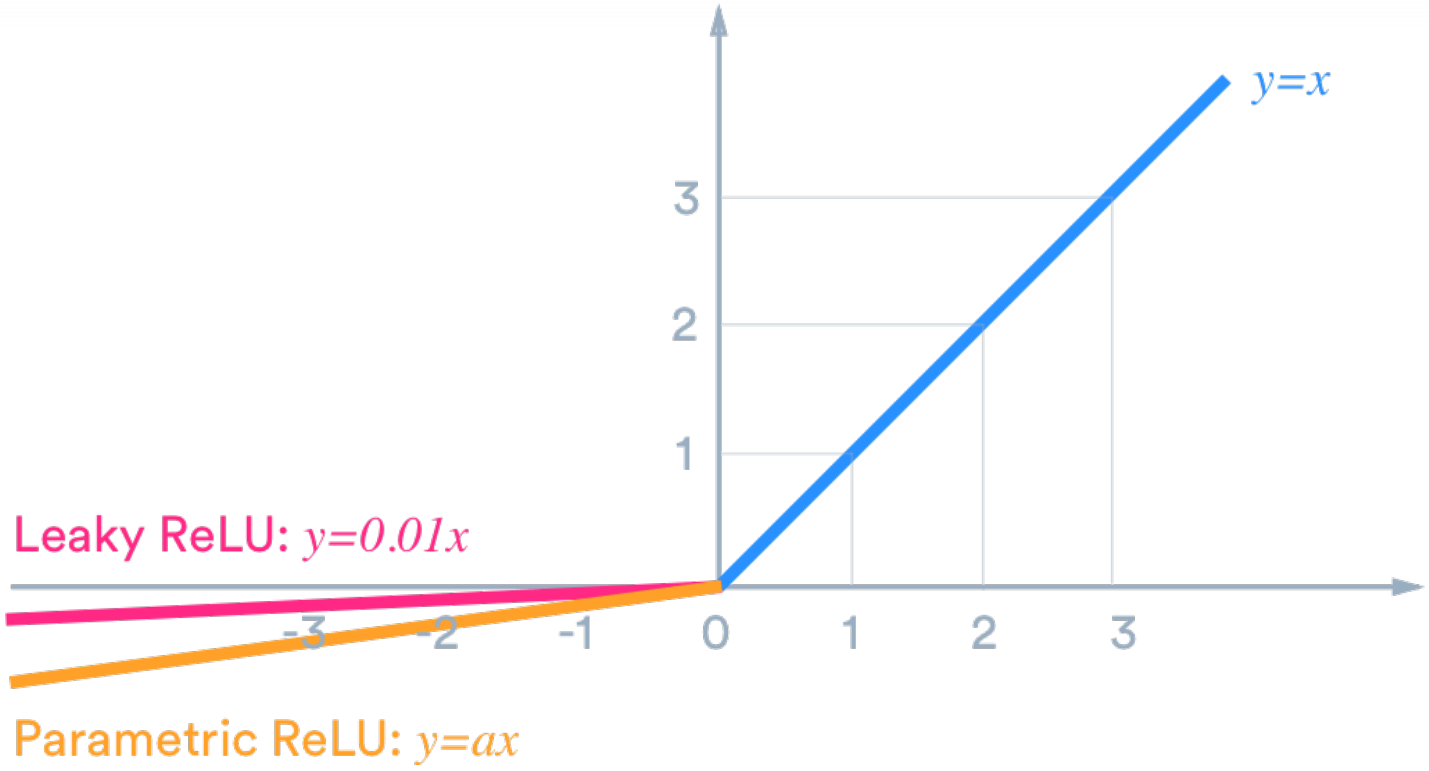


Menor probabilidad de que el gradiente se desvanezca que en el caso de la función de activación sigmoidea

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

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

Función de activación Leaky ReLu



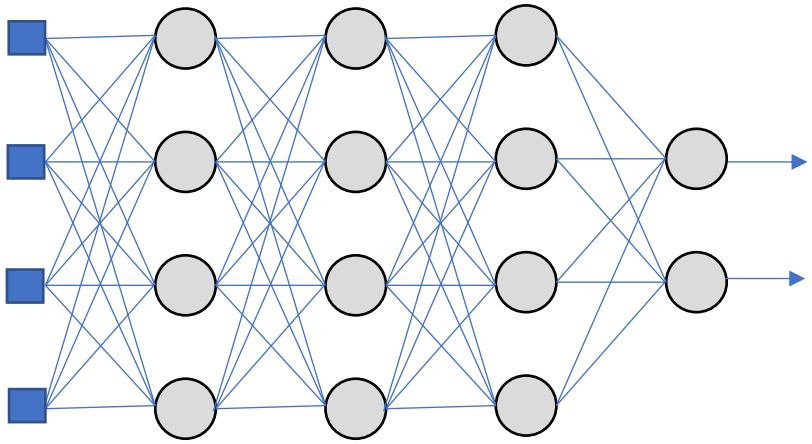
Evita el problema de la Dying Relu ya que tambien genera gradientes no nulos para valores de x negativos

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

Inicialización de pesos

Inicializar 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 $z = \sigma(w_1 x_1 + w_2 x_2 + ... + w_n x_n)$

- 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 $\sigma^2 = \frac{1}{\boxed{n}}$

Cantidad de entradas de la neurona

Inicialización de pesos

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

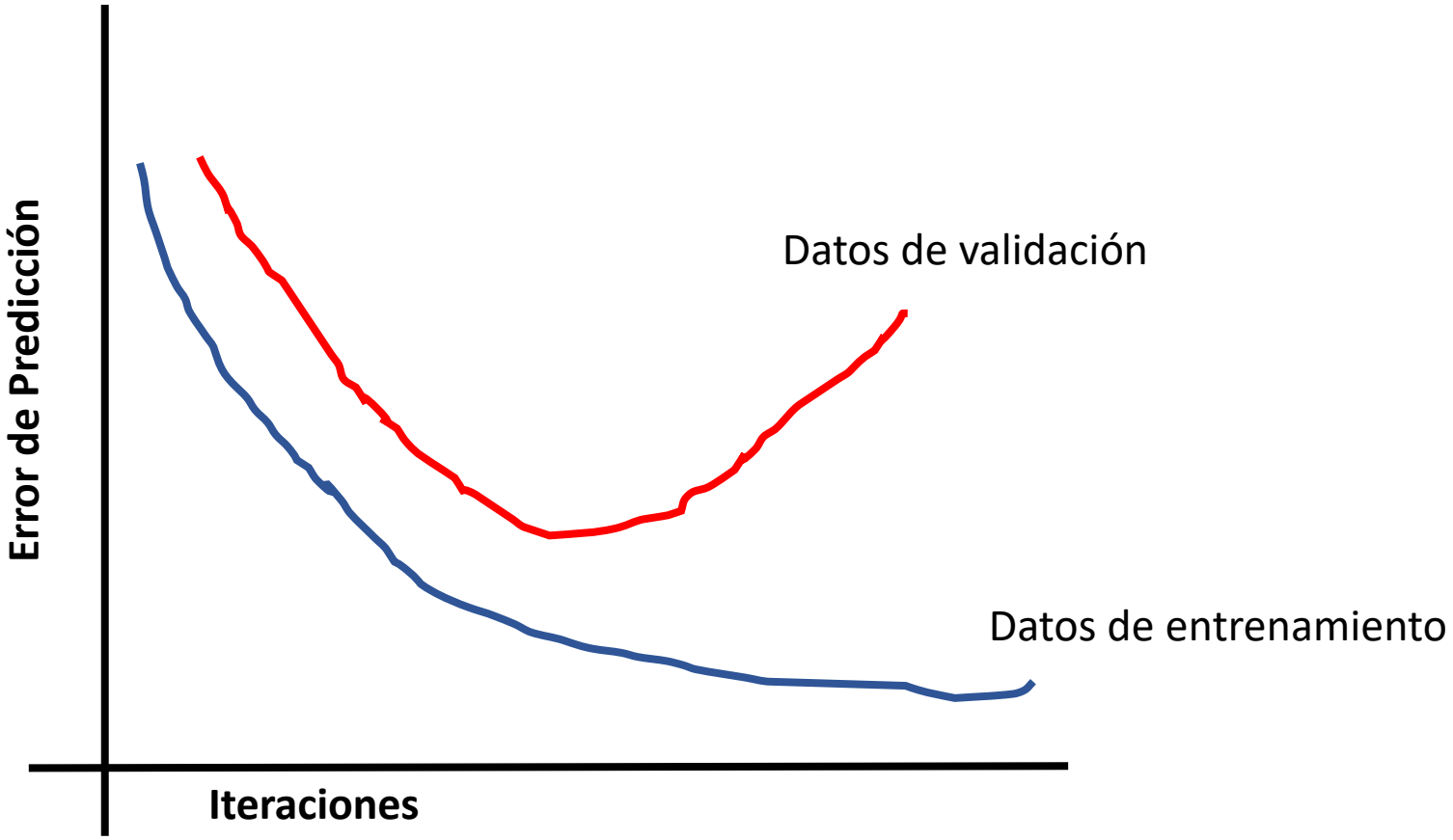
Samplear los pesos **w** de una distribución normal con varianza:

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

Útil para funciones **ReLu**

Sobreajuste y regularización

Sobre ajuste (overfitting)



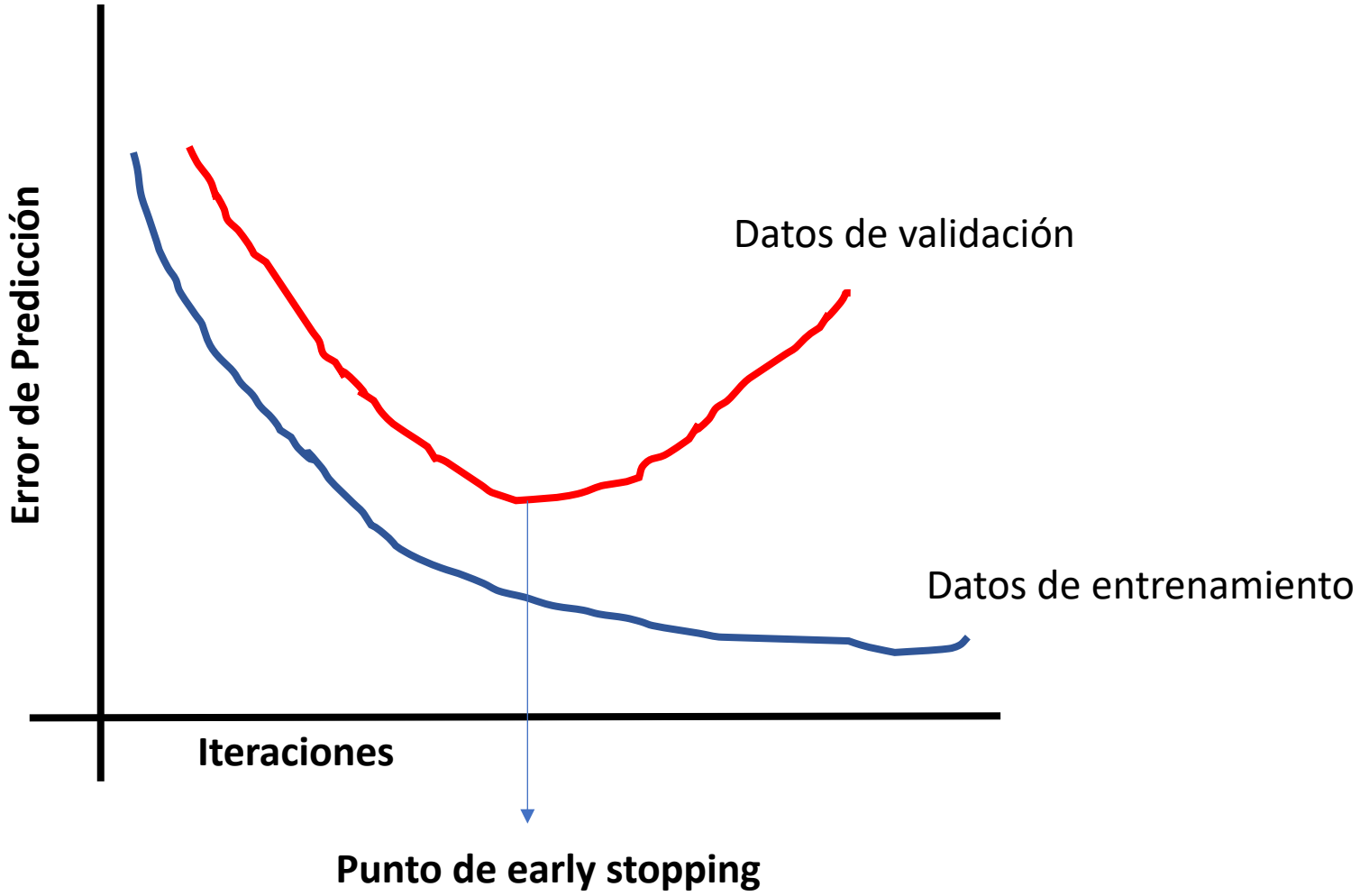
Sobre ajuste (overfitting)

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 $\mathbf{w_i}$ muy grandes, es decir, agregando una restricción sobre el valor que pueden tomar los pesos $\mathbf{w_i}$.

L2 Regularization (Weight Decay)

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

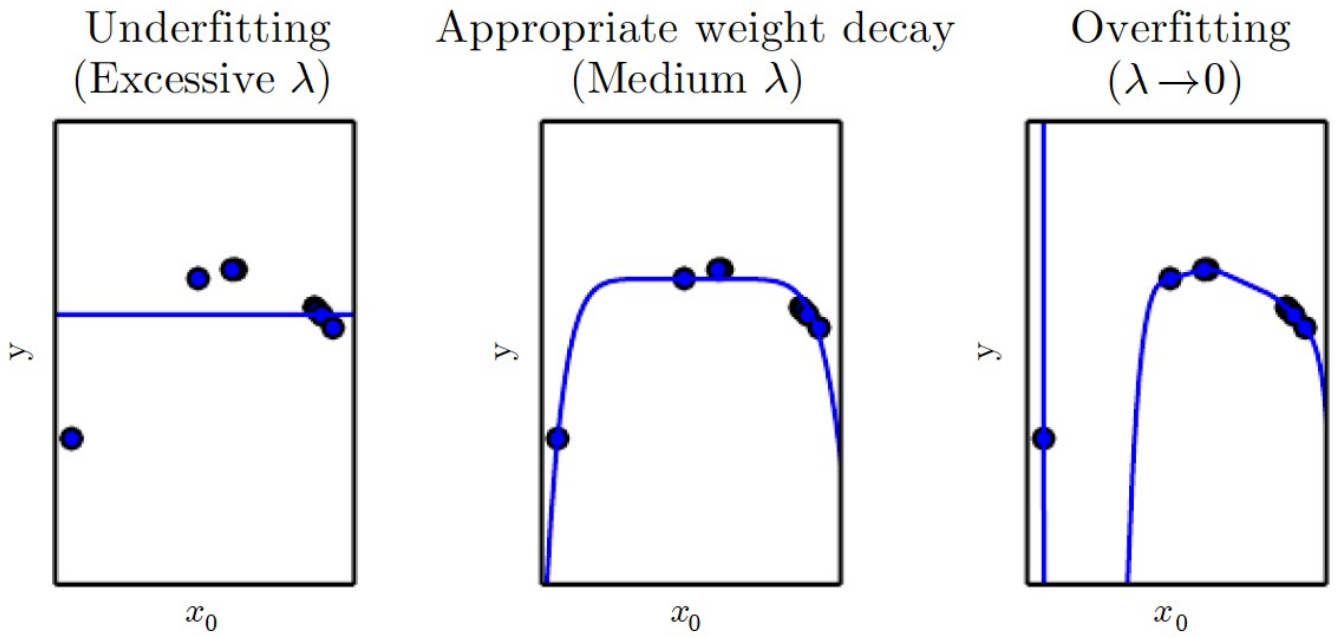
$$\|\mathbf{w}\|_2^2 = \left(\left(\sum_{m=1}^{M} w_m^2\right)^{1/2}\right)^2 = \sum_{m=1}^{M} w_m^2$$

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

N = Número de data samples en el training set **M = Número de parámetros en w**

Regularización: L2 Weight-decay

Influencia del parámetro λ: Ejemplo al fitear un polinomio de orden 9 sobre datos sampleados de una función cuadrática



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

Por qué se llama Weight-decay?

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

Al resolver el problema de optimización con gradiente descendente

$$\mathbf{w} = \mathbf{w} - \delta \nabla_{\mathbf{w}} \mathcal{L} \underbrace{-\lambda \mathbf{w}}_{\text{Weight decay}}$$

Qué pasa si iteramos con el gradiente = 0?

Aumentación de datos

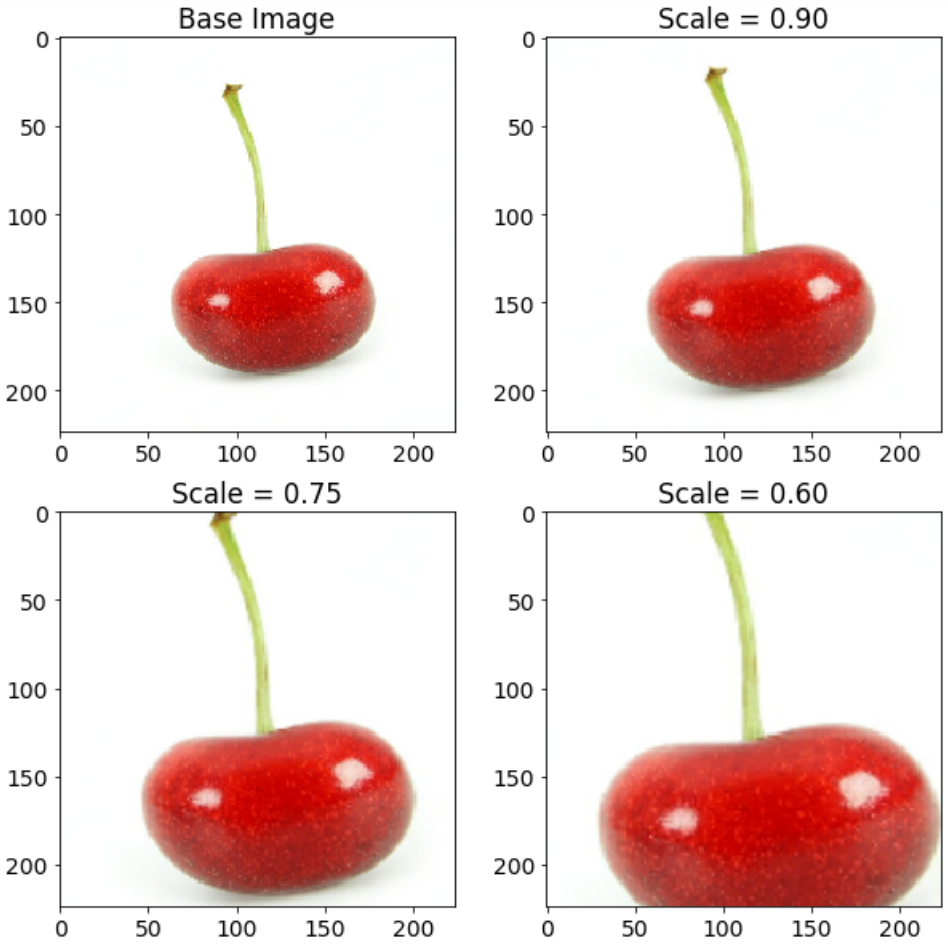
Aumentación de datos

- **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

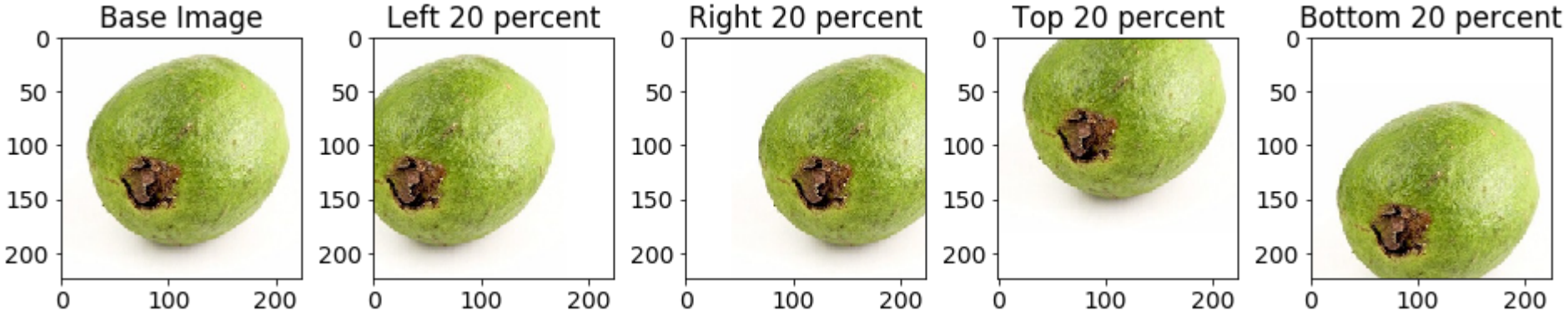
Scaling



https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9

Aumentación de datos

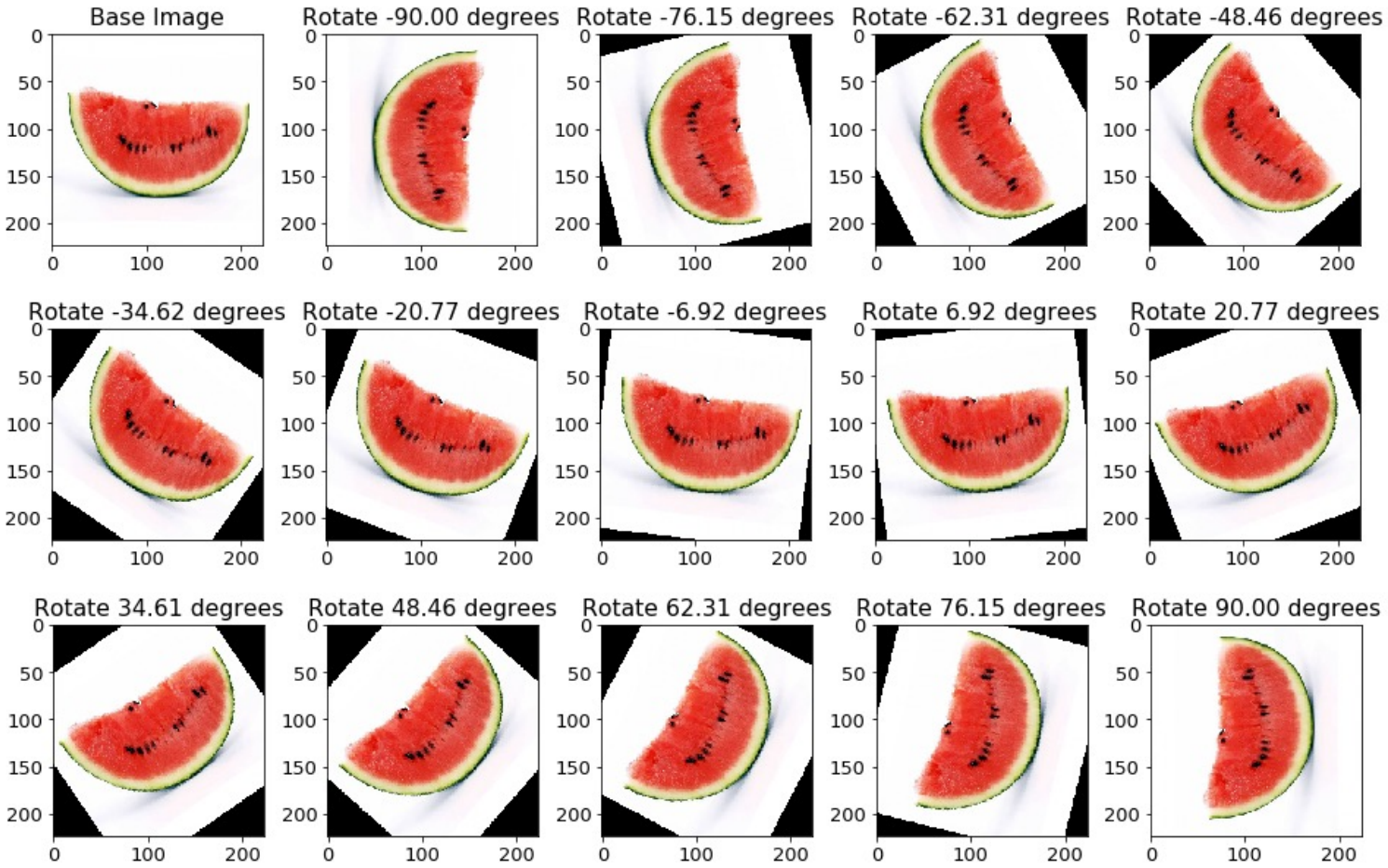
Translation



https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9

Aumentación de datos

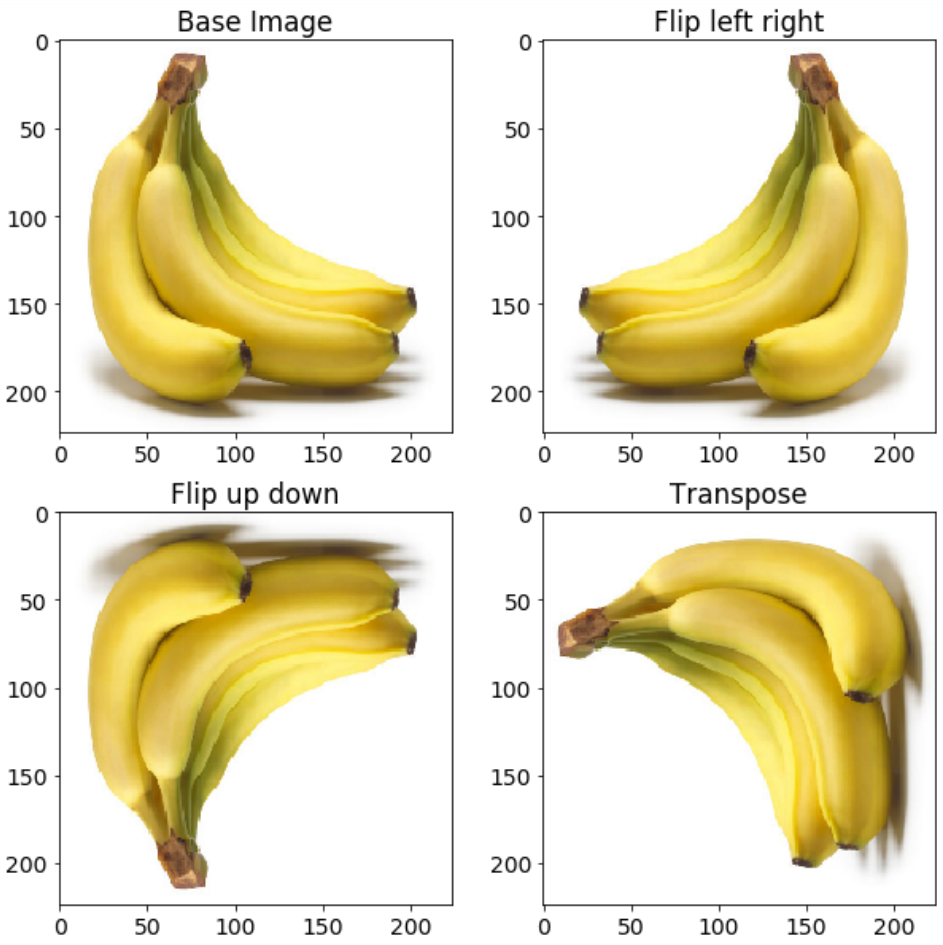
Rotation



https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9

Aumentación de datos

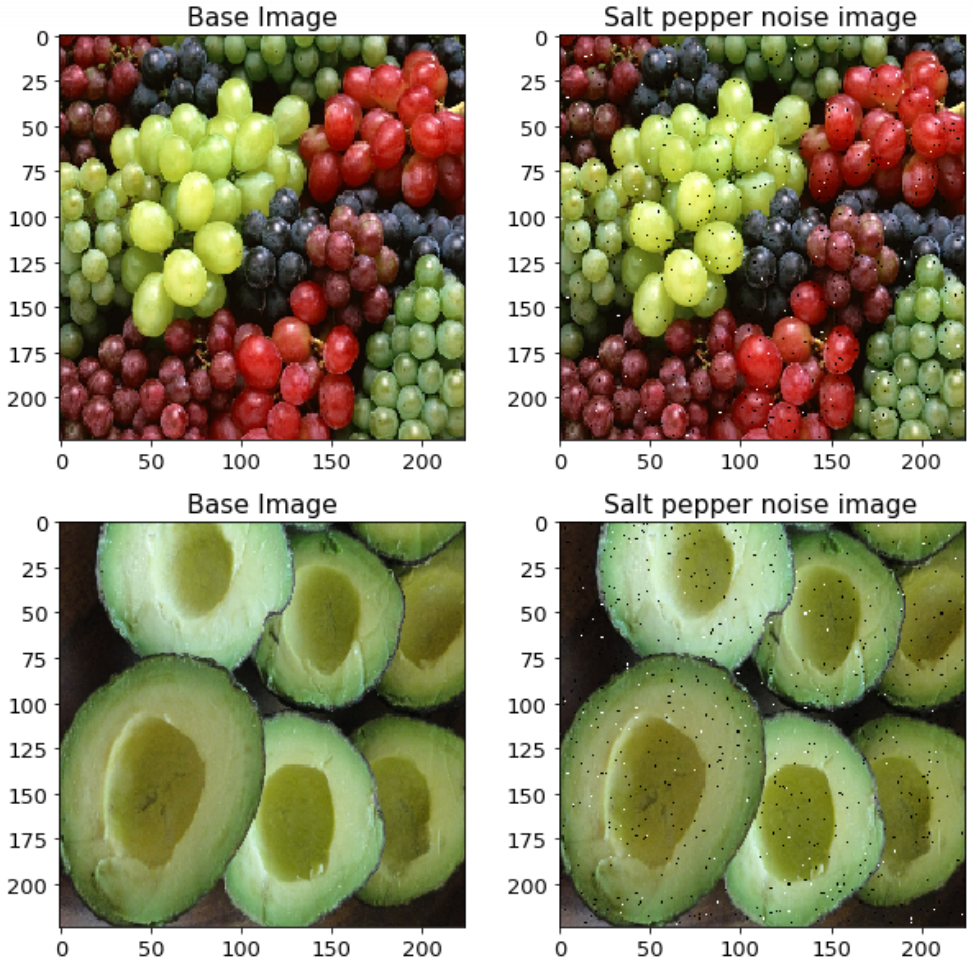
Flipping



https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9

Aumentación de datos

Ruido



https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9

Aumentación de datos

No sólo para imágenes

Fuente: https://stats.stackexchange.com/questions/320952/data-augmentation-strategies-for-time-series-forecasting

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 NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.

Editor: Yoshua Bengio

Abstract

Deep neural nets with a large number of parameters are very powerful machi
systems. However, overfitting is a serious problem in such n
slow to use, making it difficult to deal
different large

Regularización: Dropout

- **Intuición**: combinar las predicciones de múltiples modelos entrenados con el mismo fin (ensemble learning) es una forma de prevenir overfitting.

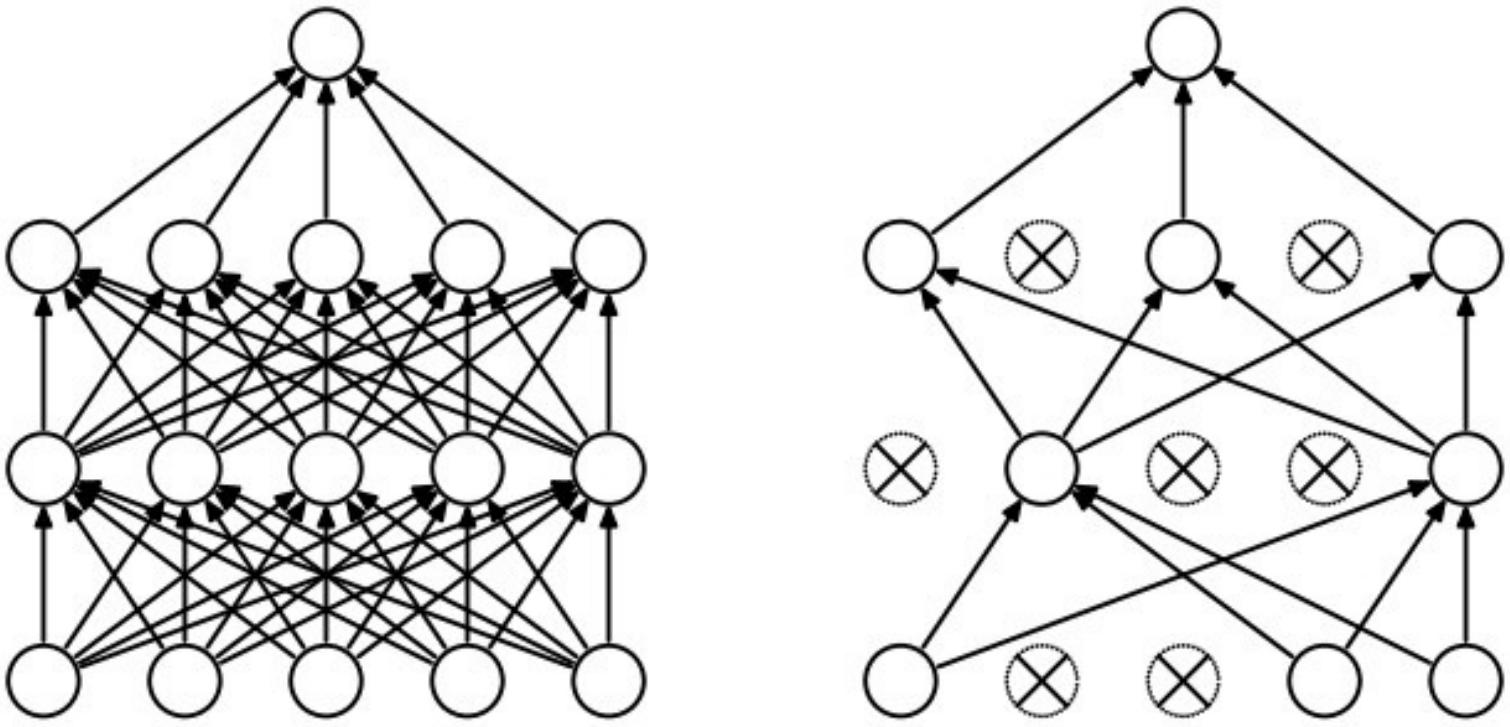
- **Idea**: promediar las predicciones de múltiples modelos entrenados independientemente para resolver el mismo problema.

- **Problema**: es carísimo!

- **Solución**: hacer que un mismo modelo sea a la vez "múltiples" modelos.

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

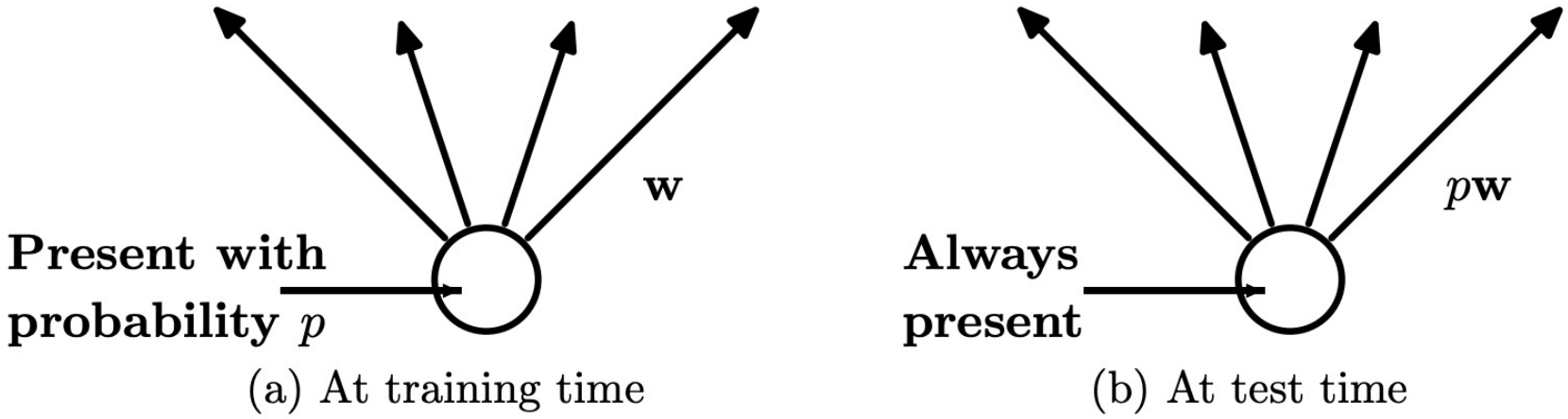


Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training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

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

Regularización: Dropout

the weights are multiplied by p. The output at test time is same as the expected output at training time.

Applying dropout to a neural network amounts to sampling a "thinned" network from it. The thinned network consists of all the units that survived dropout (Figure 1b). A neural net with n units, can be seen as a collection of 2^n possible thinned neural networks. These networks all share weights so that the total number of parameters is still $O(n^2)$, or less. For each presentation of each training case, a new thinned network is sampled and trained. So training a neural network with dropout can be seen as training a collection of 2^n thinned networks with extensive weight sharing, where each thinned network gets trained very rarely, if at all.

At test time, it is not feasible to explicitly average the predictions from exponentially many thinned models. However, a very simple approximate averaging method works well in

Regularización: Dropout

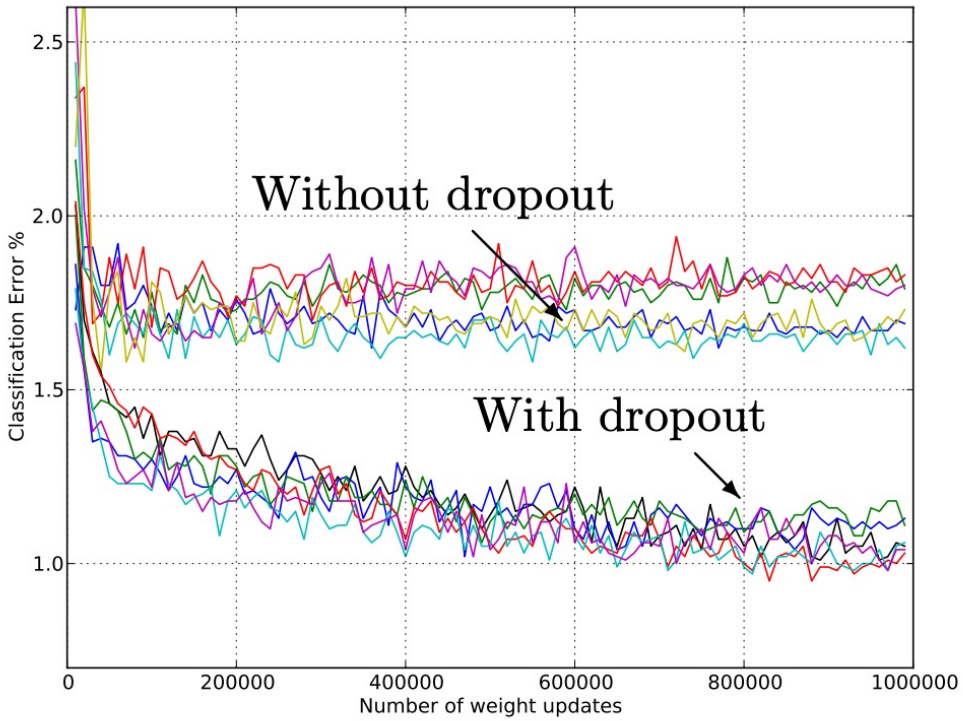


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

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

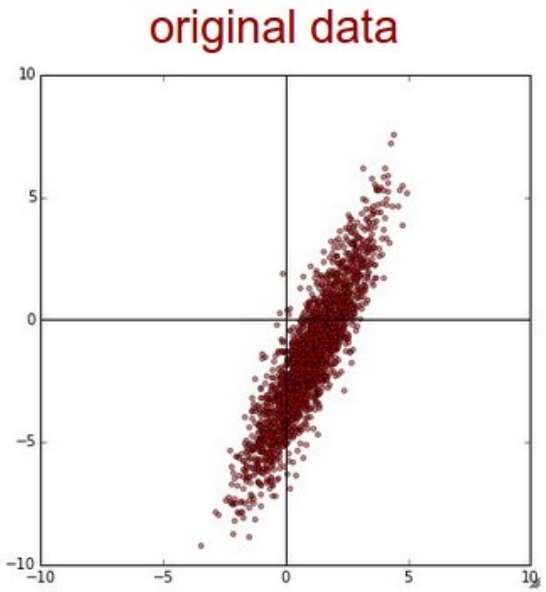
Normalización de los datos de entrada

Normalización de las entradas

Cuando cada dimensión se mueve en un rango de valores diferente

Matriz de datos de entrenamiento
[N x 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

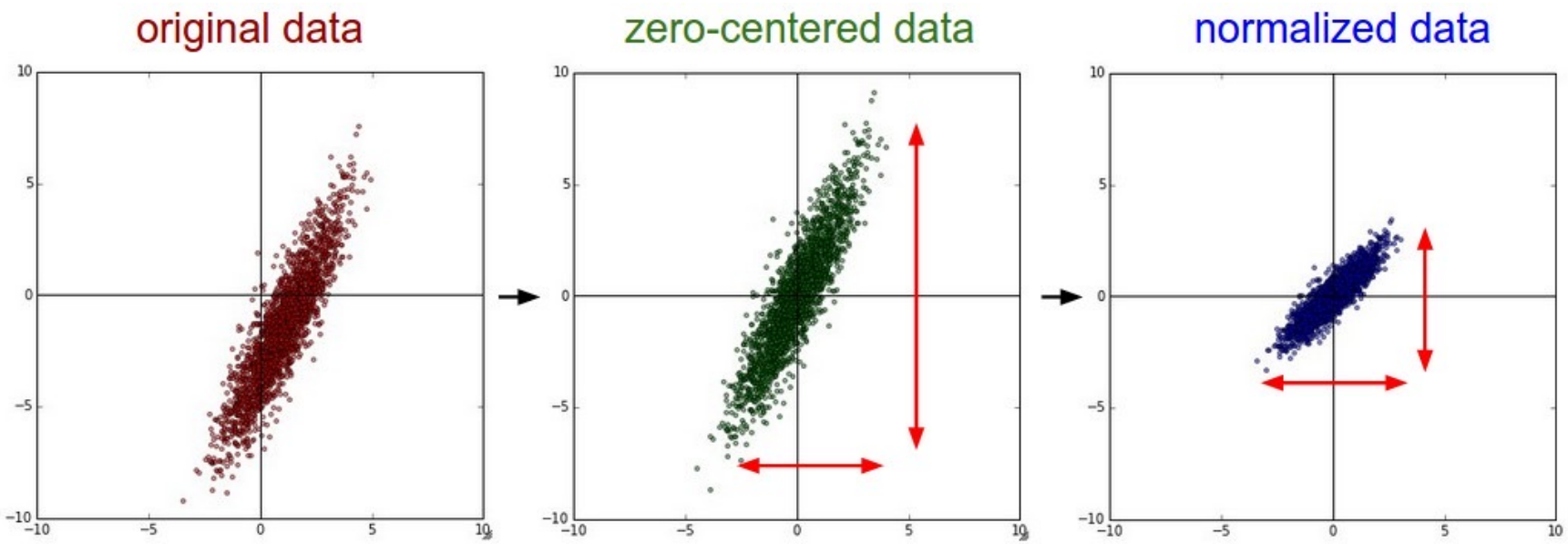
Cuando cada dimensión se mueve en un rango de valores diferente

Matriz de datos de entrenamiento
[N x D]

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

Restar la media por feature calculada sobre todo el dataset

Restar la media y dividir por el desvío por feature calculado sobre todo el dataset



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.

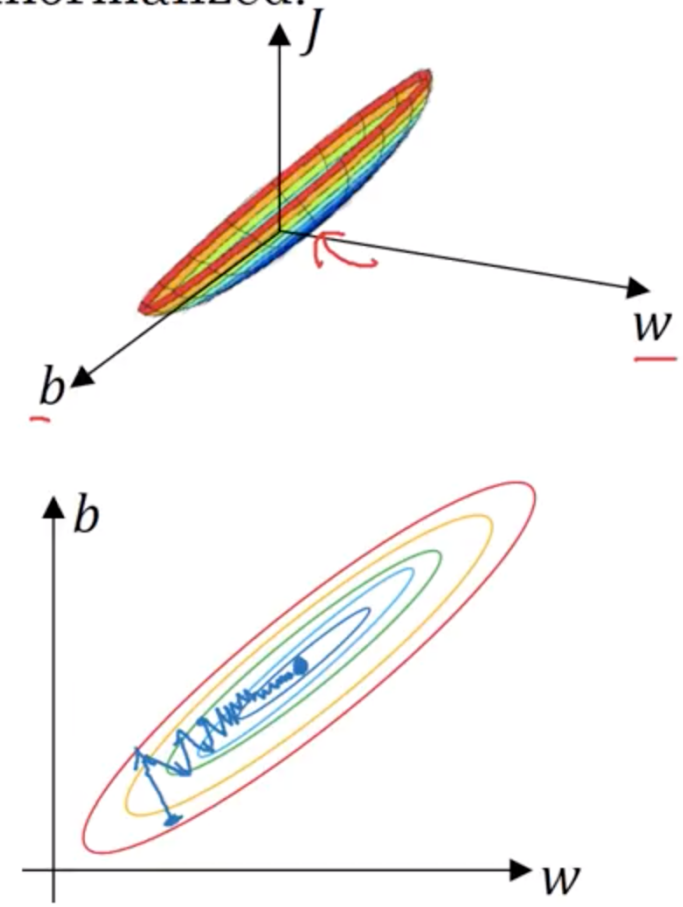
Imagen extraída de http://cs231n.github.io/neural-networks-2/

Normalización de las entradas

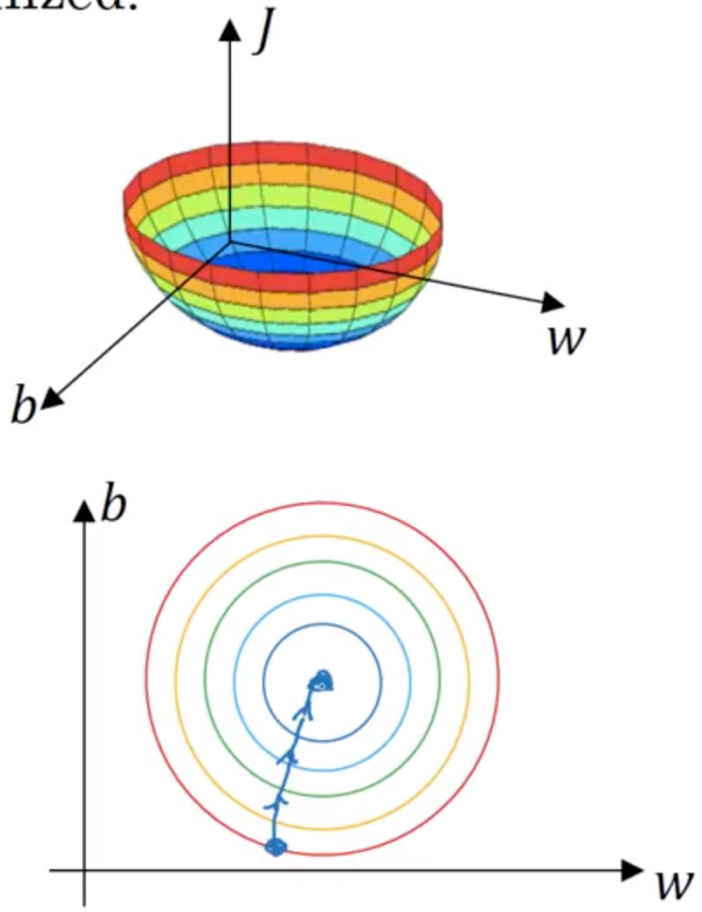
Por qué normalizar los datos?

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

Unnormalized:



Normalized:



Crédito: Andrew Ng

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.

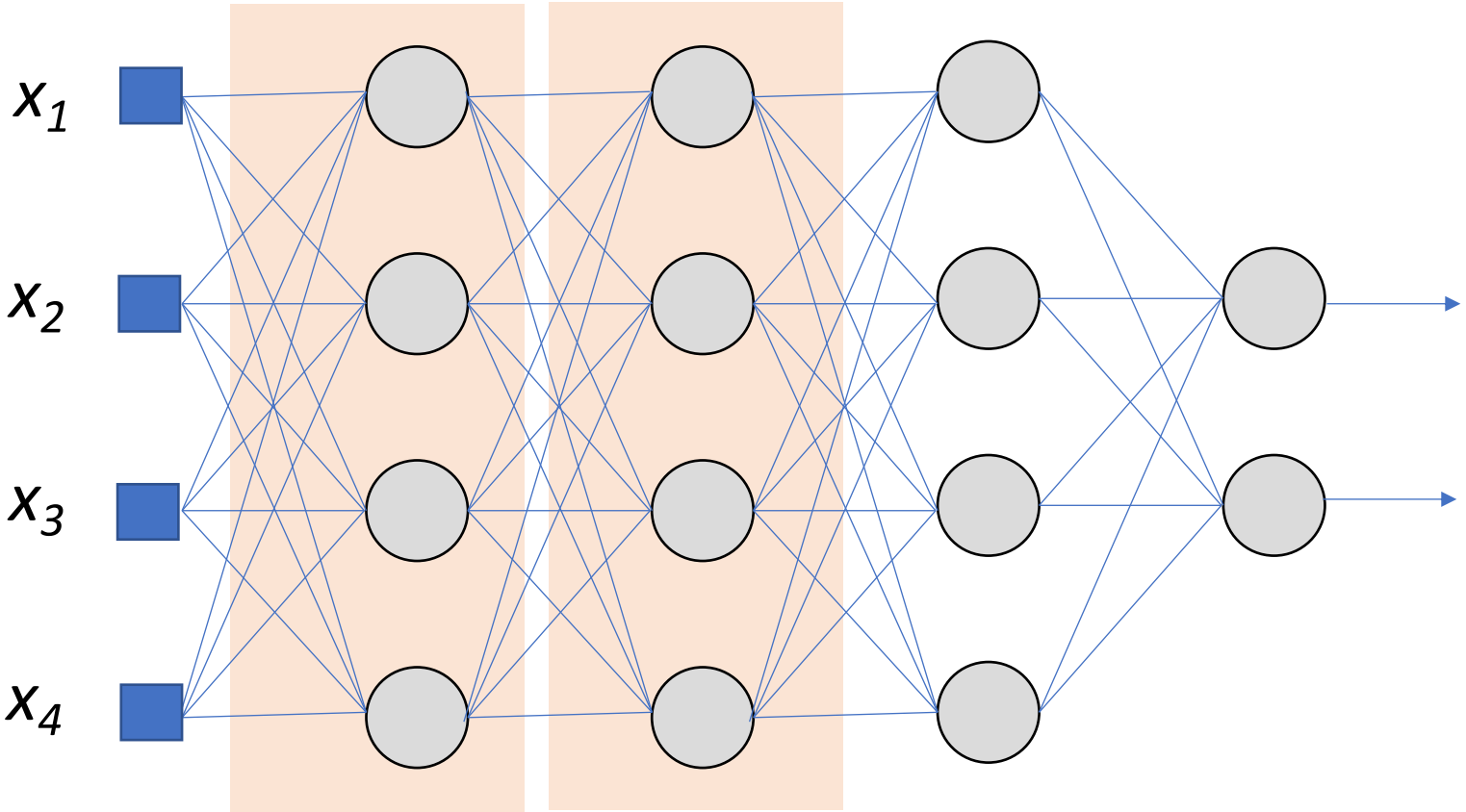
Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more 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.

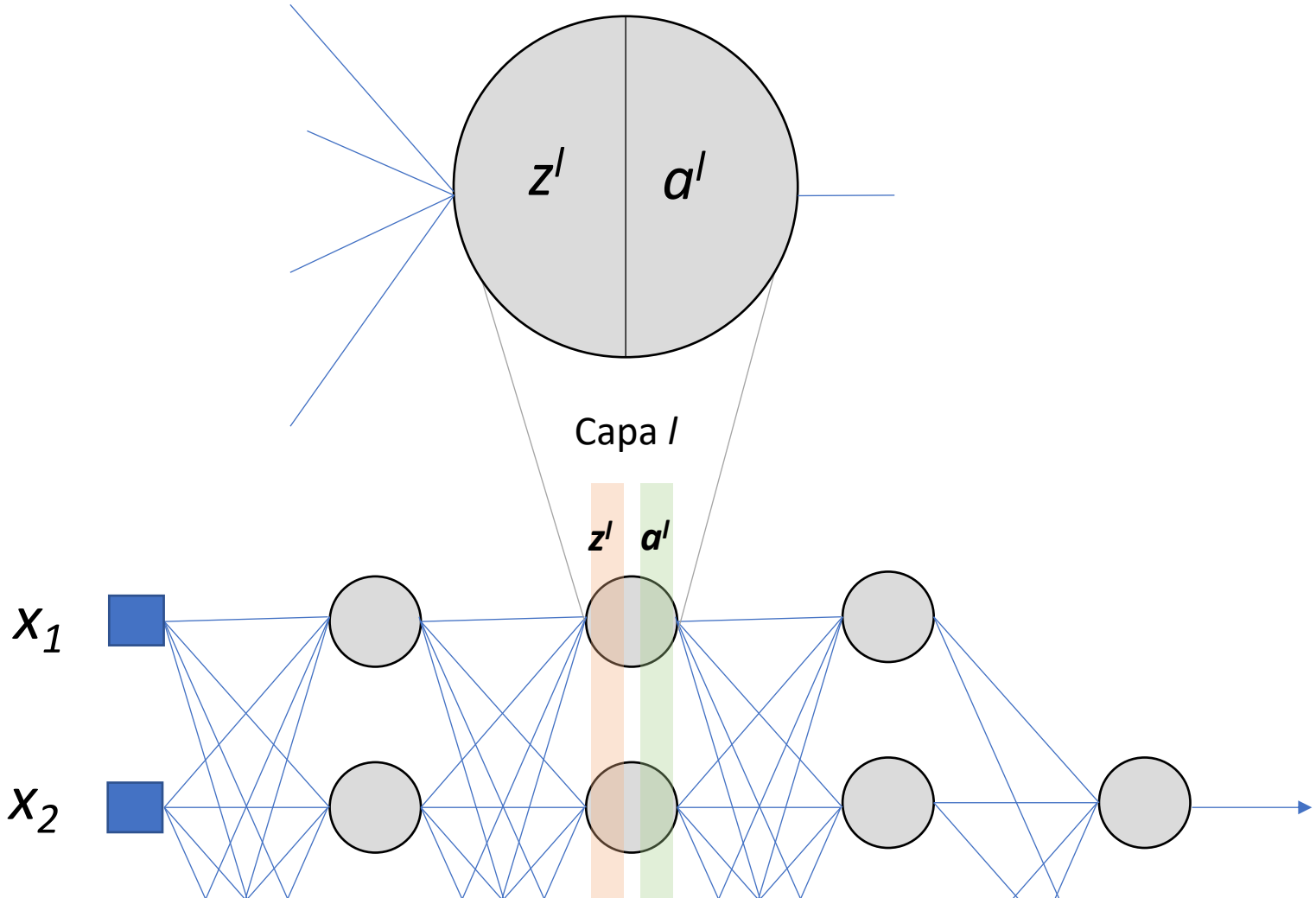
The change in the distributions of layers' inputs

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 → 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

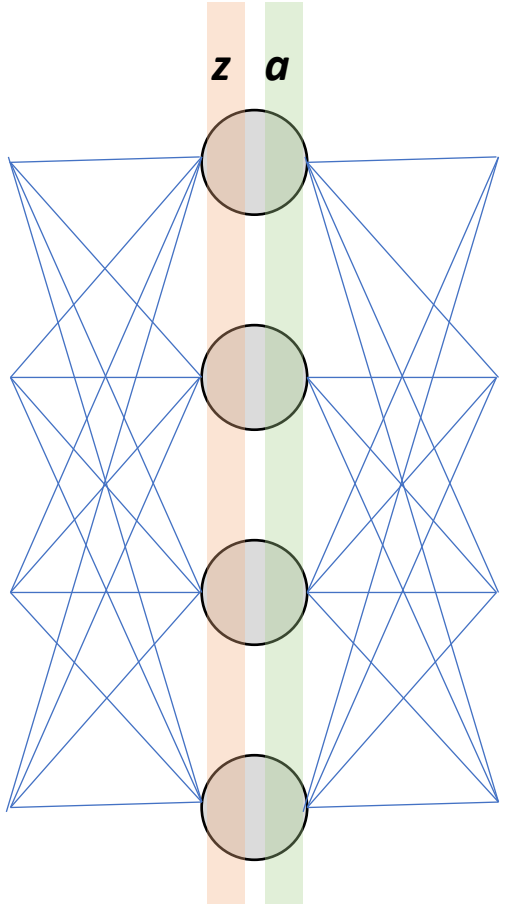
$$\hat{\mathbf{z}}_n = \frac{\mathbf{z}_n - \mu}{\sqrt{\sigma^2 + \boxed{\varepsilon}}}$$

Evita inestabilidad

4. Re-escalo

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

Parámetros entrenables (por capa)



Batch Normalization

Dados *N* training samples en un minibatch → Batch Norm en la capa *l:*

Se computan a partir de los datos de cada minibatch

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. Normalizar

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

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

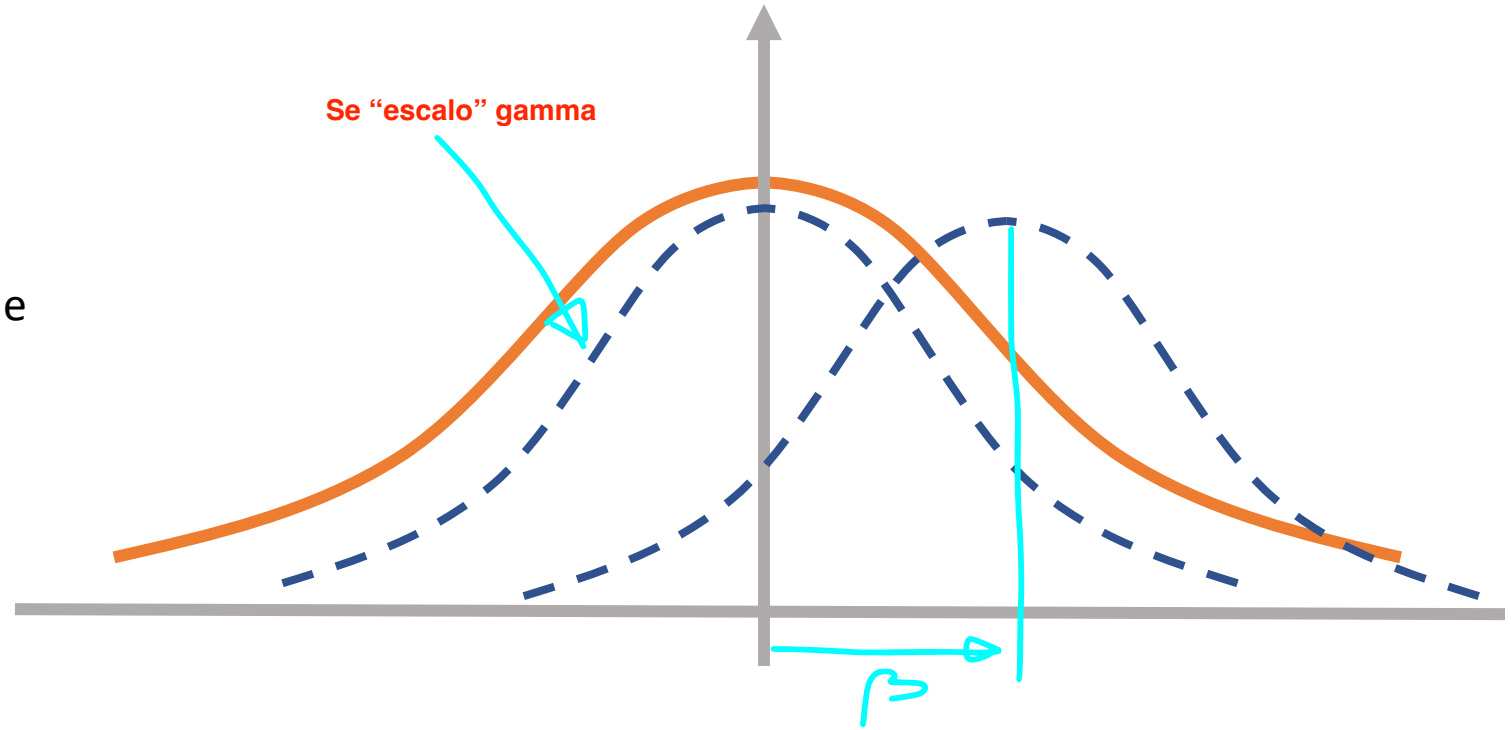
4. Re-escalar

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

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

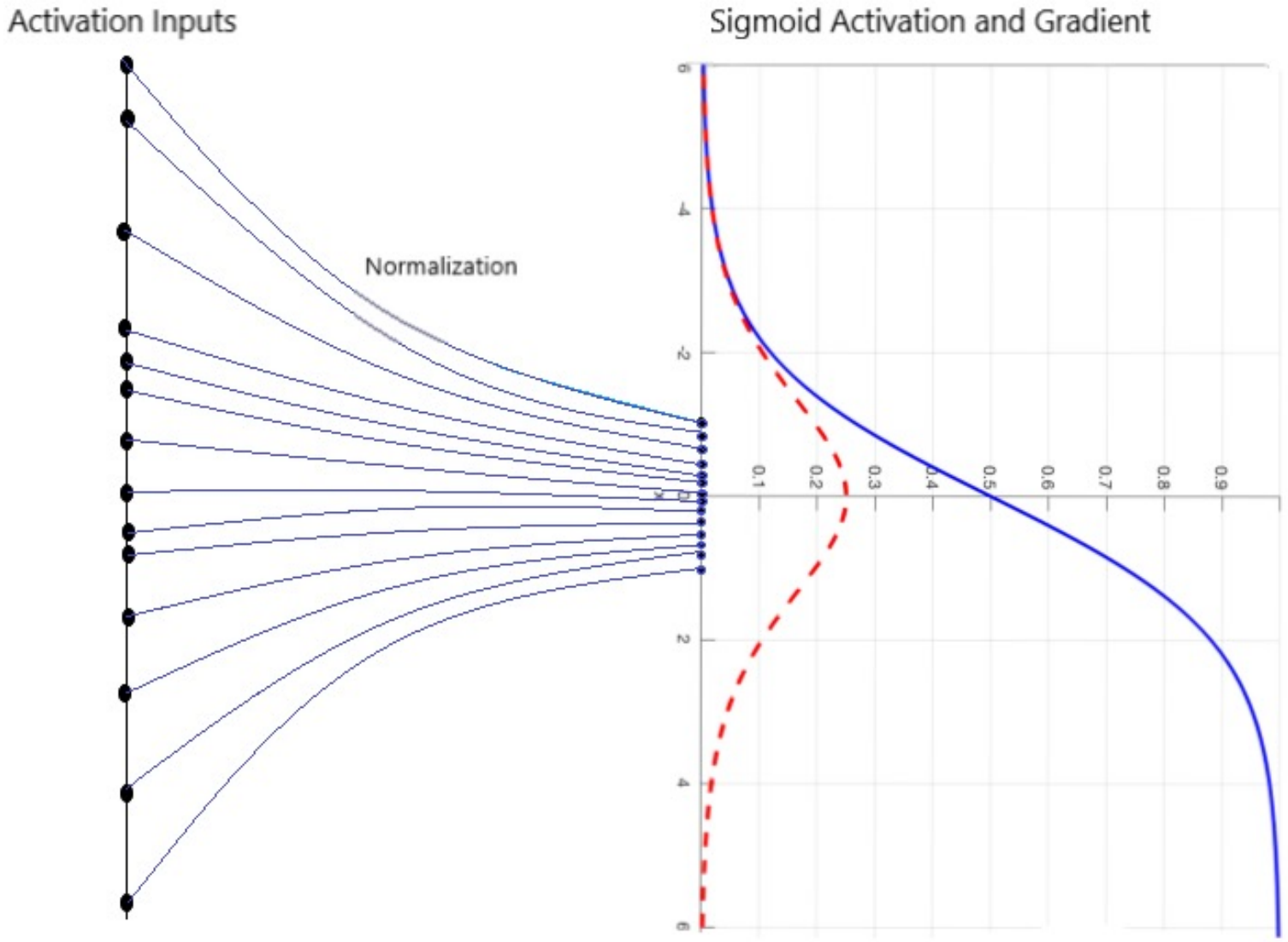
- **Efecto regularizador:** el cálculo de la media y varianza es por minibatch

  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.”*

Deep Learning with Pytorch, Página 222

Efectos obtenidos al aplicar Batch Norm



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

Imagen extraída de: https://towardsdatascience.com/intuit-and-implement-batch-normalization-c05480333c5b

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

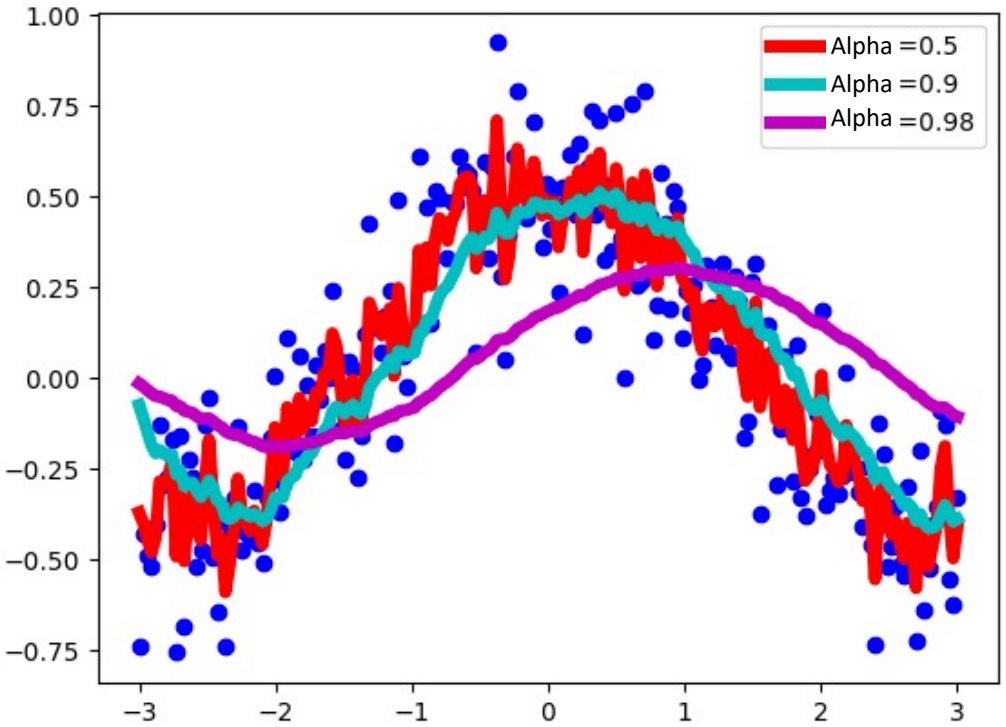
- Por cada capa *l* usamos los parámetros aprendidos: γ_l, β_l
- Tenemos solo un data sample → Qué usamos para μ, σ^2 ?

La media móvil exponencial de μ, σ^2
acumulada durante el entrenamiento

Media Móvil Exponencial (EMA)

Ej: Dada una serie de medias (una para cada minibatch) $\mu_1, \mu_2, \ldots \mu_k$

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



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

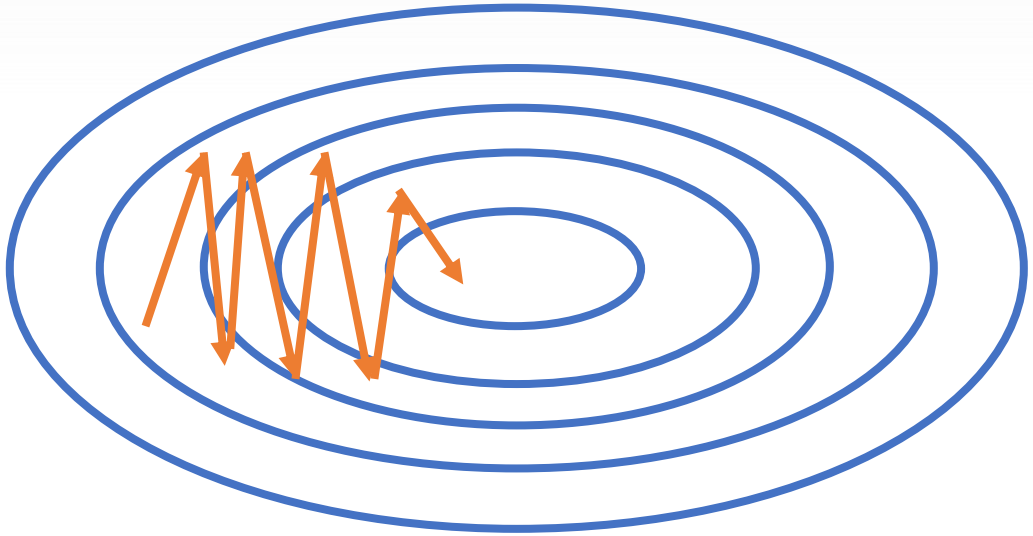
Variantes del gradiente descendiente

Variantes del gradiente descendiente

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

Gradiente descendiente con Momentum

Sin momentum

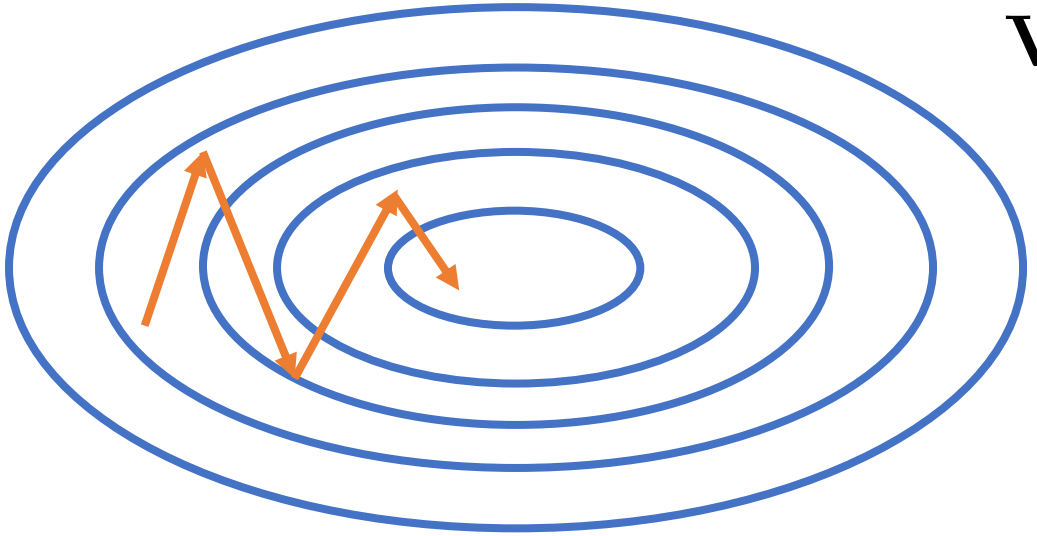


$$\mathbf{W} = \mathbf{W} - \delta \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W})$$

Alfa suele tomarse = 0.9 (promedio 10 últimos)

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

Con momentum



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

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

Gradiente descendiente con Momentum

- Suaviza los pasos dados por el gradiente descendiente en las dimensiones más oscilantes

- Acelera la convergencia del método

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**

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

- Los **métodos adaptativos escalan el LR por cada parámetro**. Ejemplos:
  - RMSProp http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
  - 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

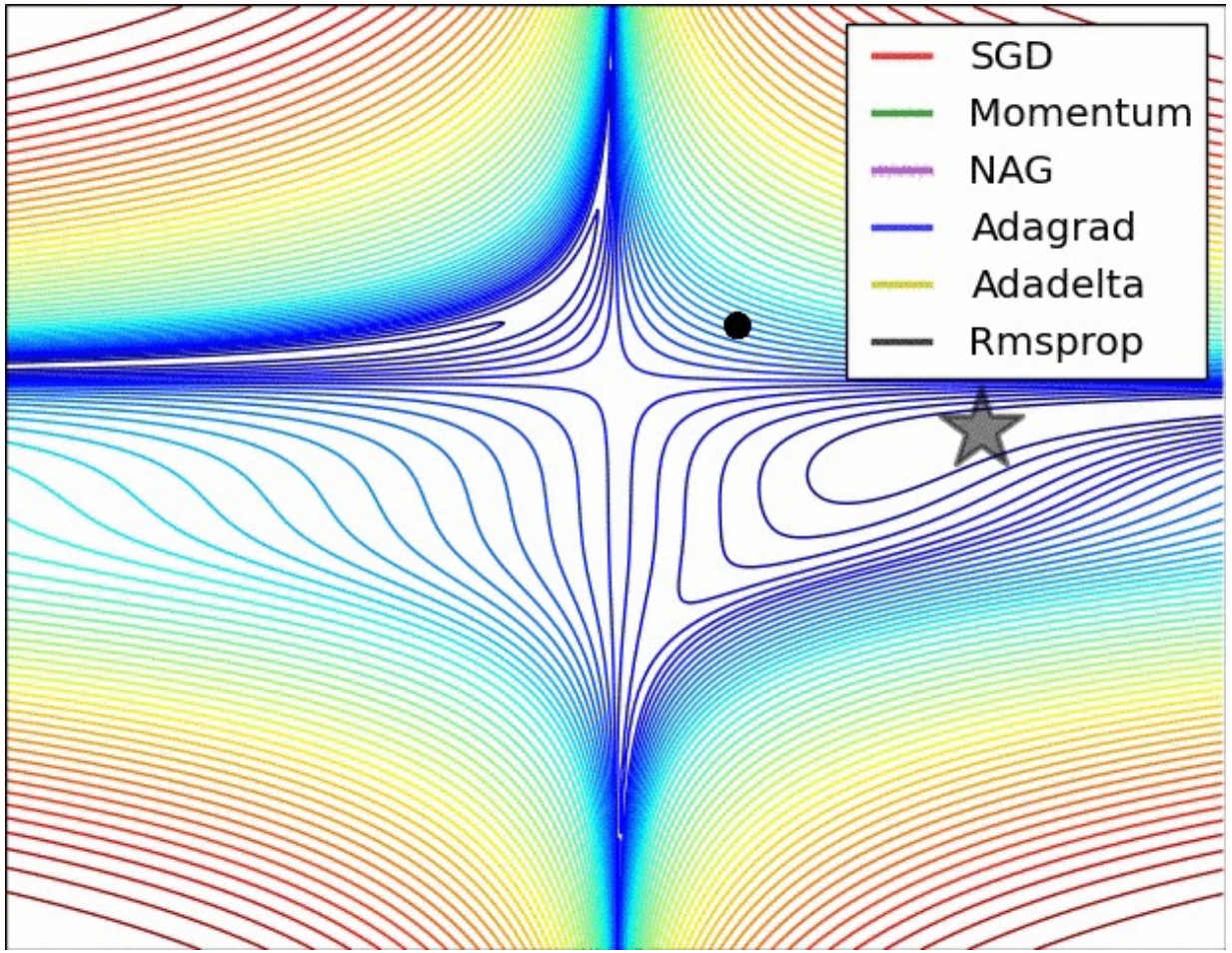


Imagen extraída de : http://ruder.io/optimizing-gradient-descent/index.html#fn13 (Muy buen review!)

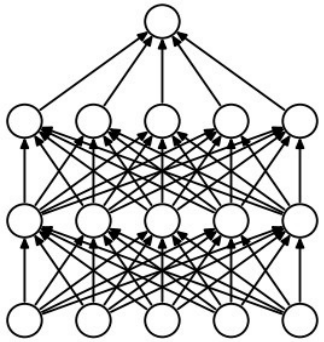
Clase 5

Entrenando una red neuronal

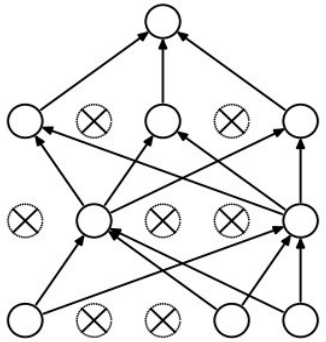
Enzo Ferrante

eferrante@sinc.unl.edu.ar

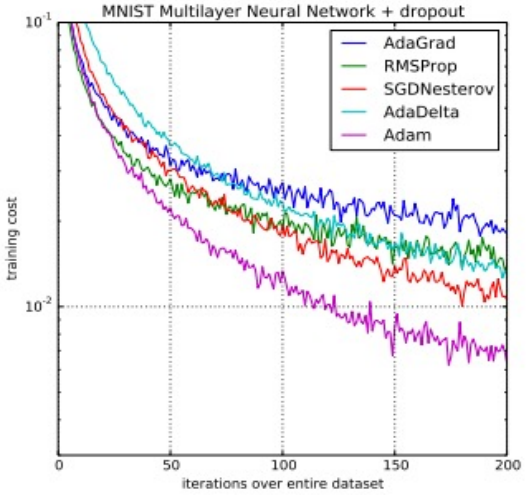
@enzoferrante



(a) Standard Neural Net



(b) After applying dropout.



Universidad de SanAndrés

UNL CONICET

s i n c (i)