

# **Deep Learning**

**INF-396**

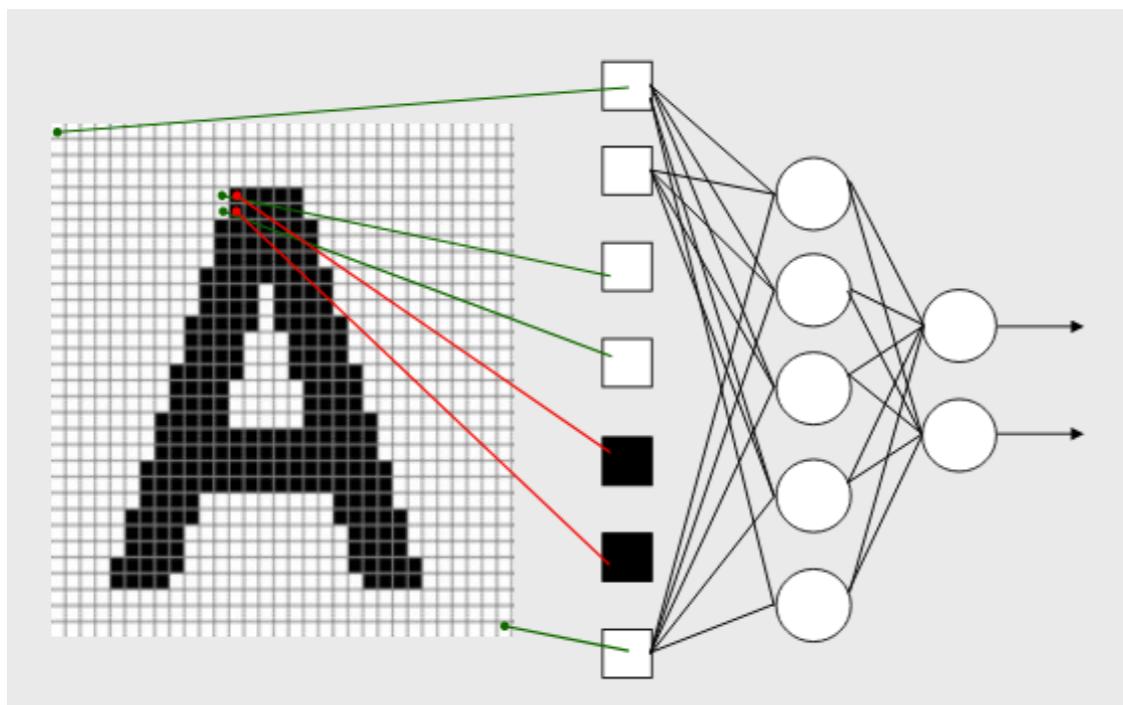
**Prof: Juan G. Pavez S.**

# Redes Convolucionales para el procesamiento de Imágenes

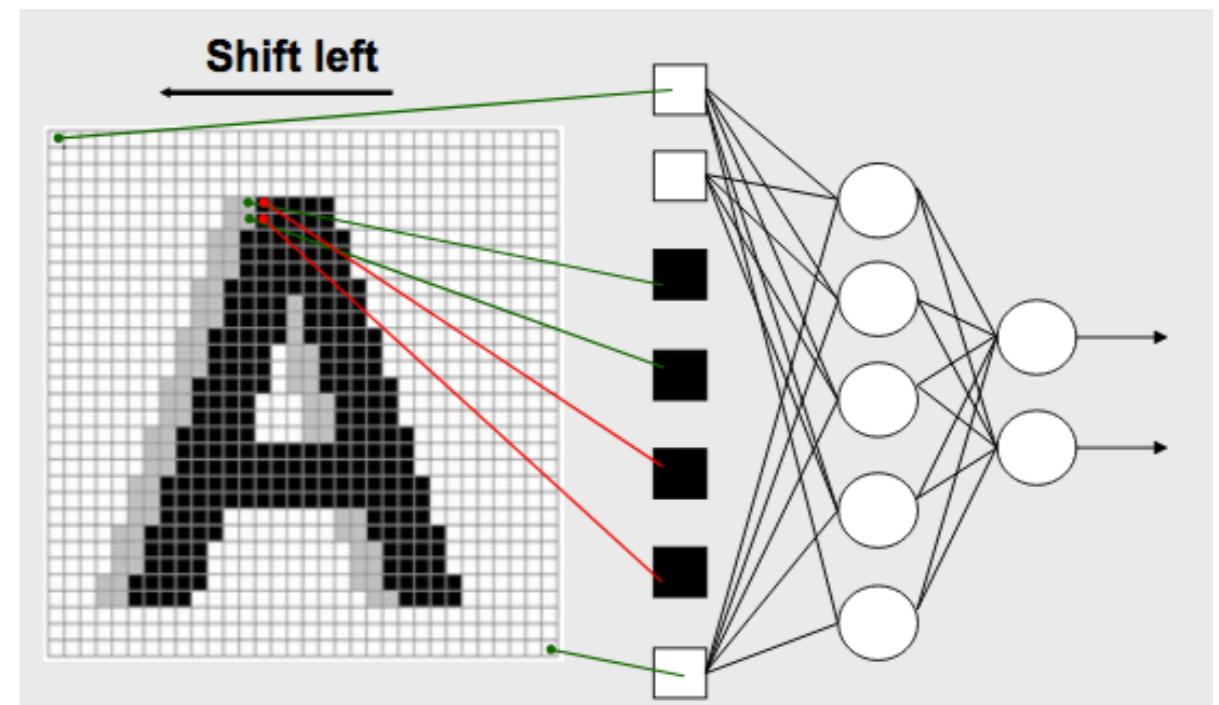
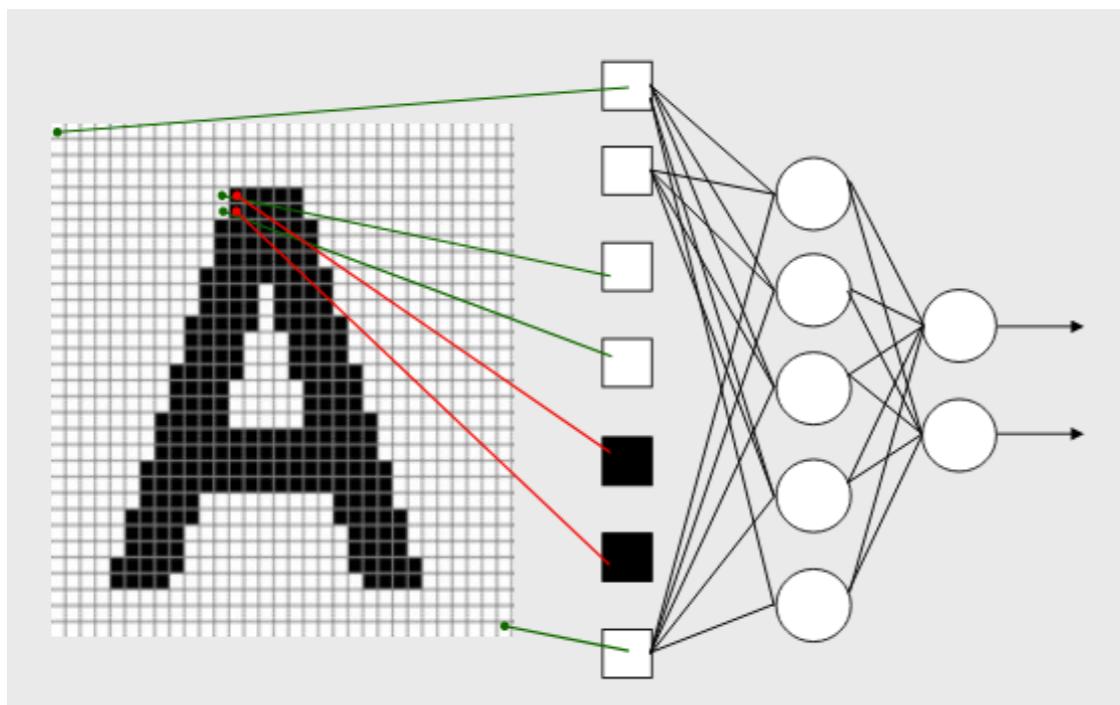
# Redes Neuronales Convolucionales

- Considere una imagen como entrada a una red neuronal densa.
- Por ejemplo una imagen de 200x200 (asumiendo que cada pixel tiene un sólo valor).
- Considere que la red tiene 500 unidades en la capa escondida. Entonces hay  $200 \times 200 \times 500 = 20M$  pesos que aprender en la primera capa.
- Para eso necesitamos MUCHOS datos.
- Ahora considerar que la imagen se mueve un poco.

# Redes Neuronales Convolucionales



# Redes Neuronales Convolucionales



# Redes Neuronales Convolucionales

- Hay algunas propiedades que podríamos querer cuando trabajamos con imágenes:
  - **Invarianza a la translación:** Si la imagen es movida, aún puede ser reconocida.
  - **Invarianza a la rotación:** Si la imagen es rotada, aún puede ser reconocida.
- Queremos incluir estas propiedades en nuestro modelo para hacerlo más eficiente: **Información a Priori.**

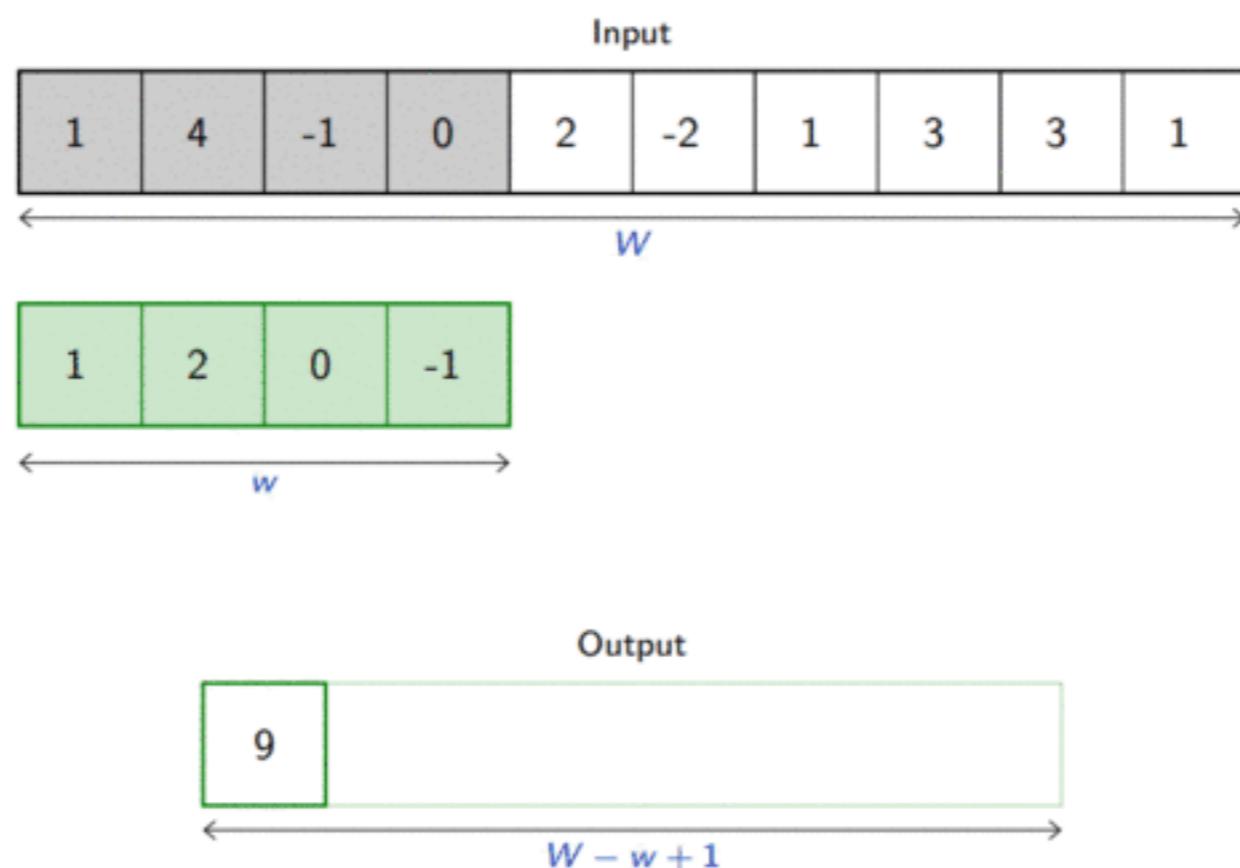
# Redes Neuronales Convolucionales

- **Las redes neuronales convolucionales** incluyen algunas de esas propiedades en la representación aprendida.
- Para eso usa la operación de **convolución (correlation)**:

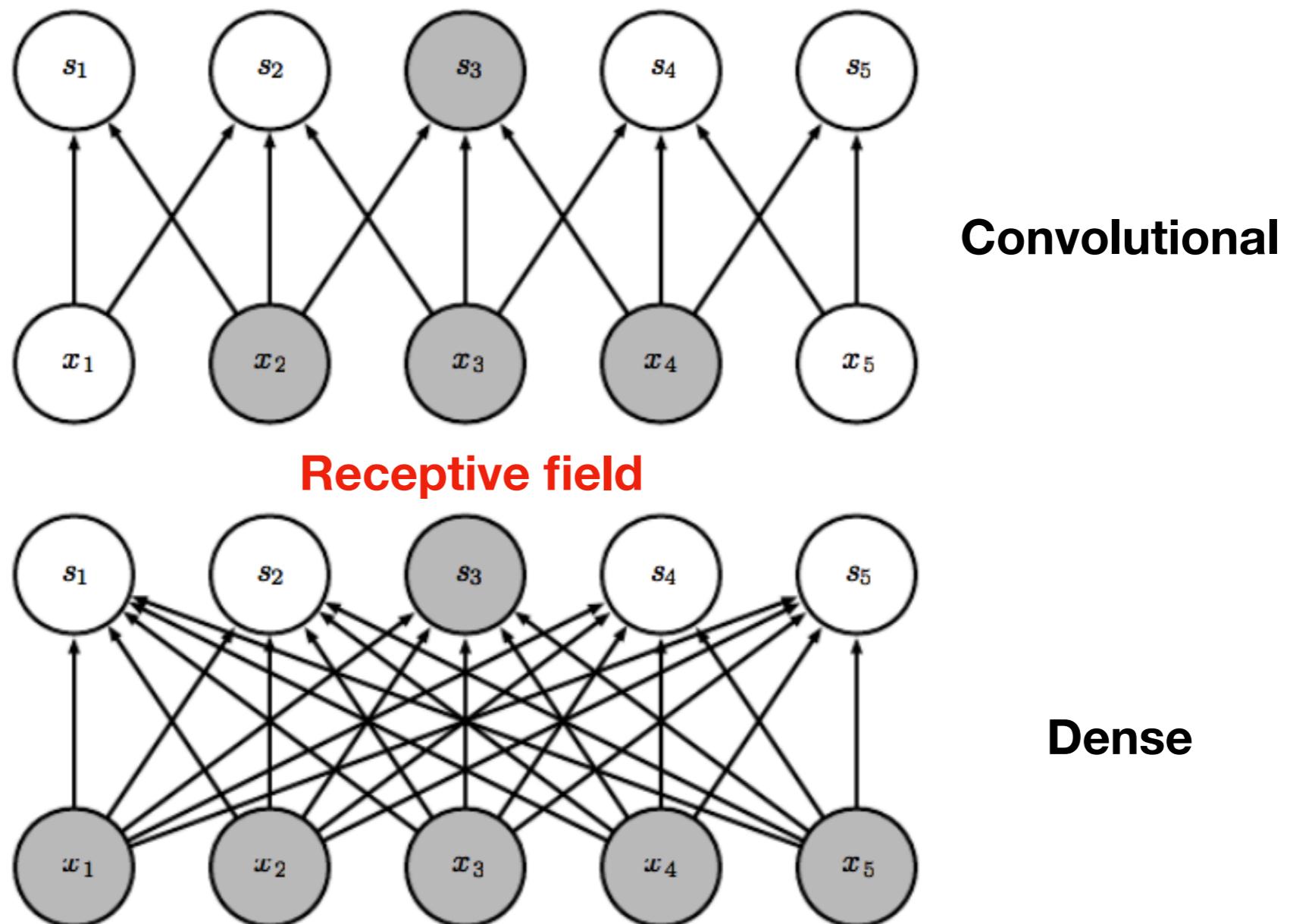
$$(x * k)_i = \sum_m x_{i+m-1} k_m$$

# Redes Neuronales Convolucionales

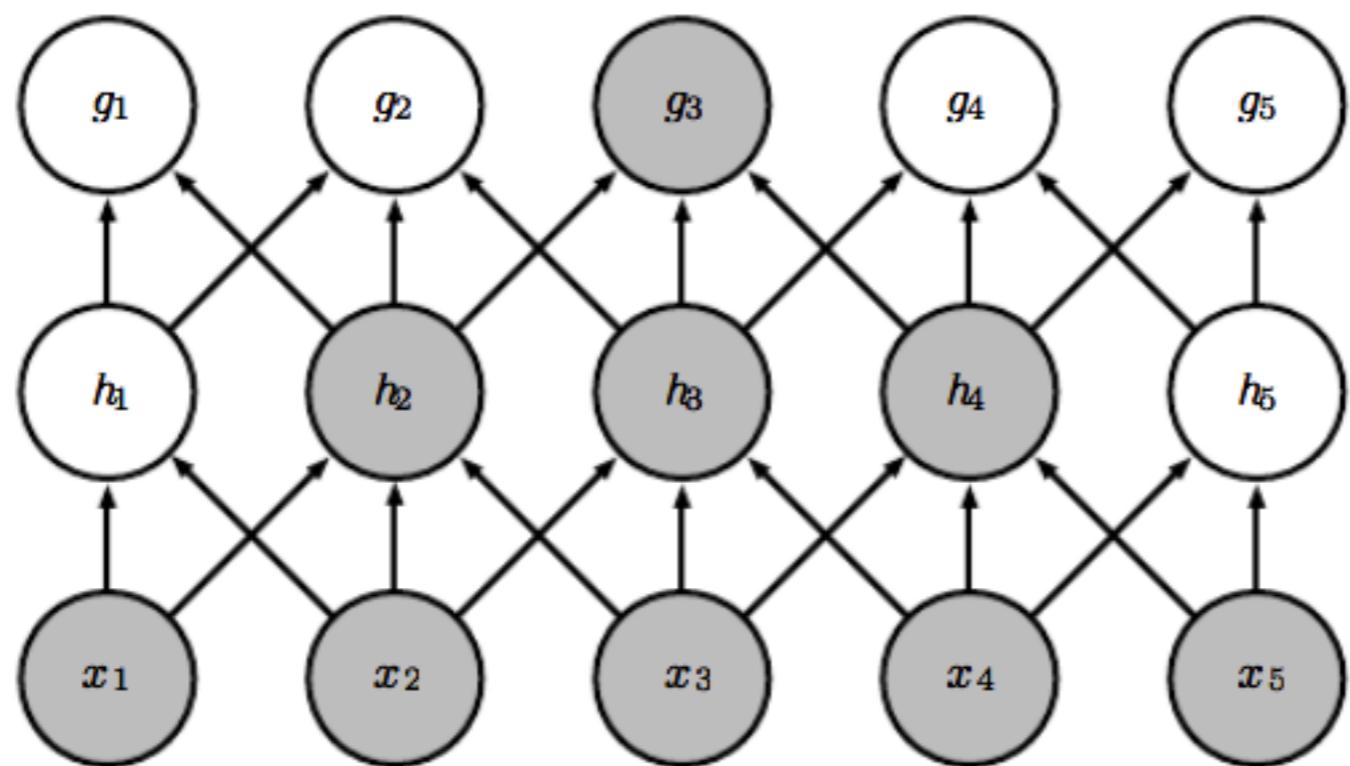
$$(x * k)_i = \sum_m x_{i+m-1} k_m$$



# Redes Neuronales Convolucionales



# Redes Neuronales Convolucionales



**Deep Convolutional**

**Receptive field**

# Redes Neuronales Convolucionales

- La convolución permite:
- **Compartir parámetros:** La misma red es usada en diferentes partes de la imagen. Menos requerimientos de memoria. Más eficiente (sí los supuestos están correctos).
- **Equivarianza a la translación:**  $f$  es equivariante a  $g$  sí:

$$f(g(x)) = g(f(x))$$

- Sí la imagen es movida un pixel a la derecha entonces la salida es la misma que aplicándola a la imagen movida.
- Si movemos la entrada, la salida se moverá.

# Redes Neuronales Convolucionales

- En general tenemos múltiples canales para cada posición de pixel, así que trabajamos con representaciones 3D.
- Por ejemplo una imagen tendrá canales RGB.
- Considere un kernel  $K$  con elementos  $K_{i,j,k,l}$  que conecta el canal  $i$  de la salida al canal  $j$  de la entrada, con offset de  $k$  filas y  $l$  columnas entre entrada y salida.

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

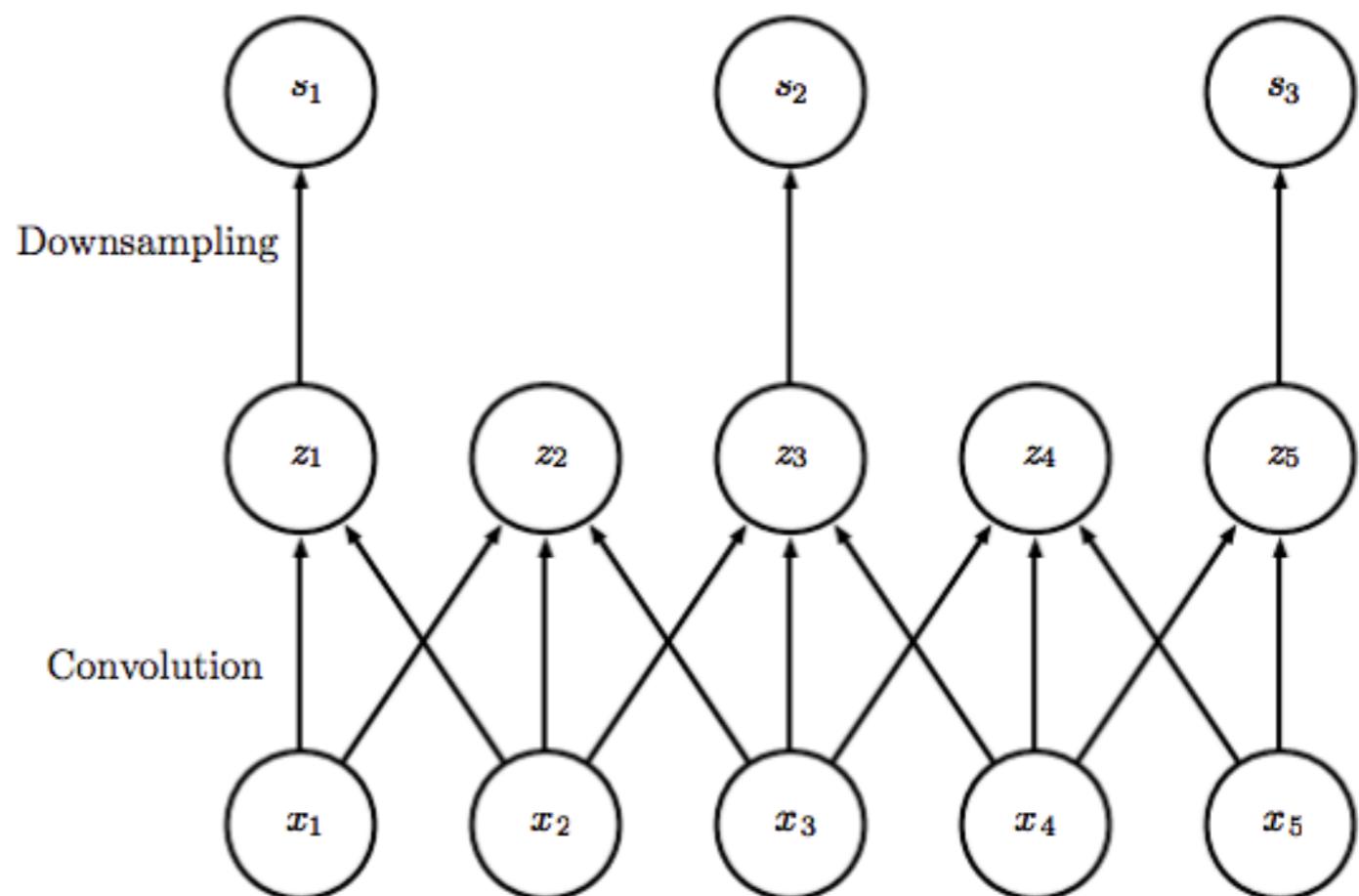
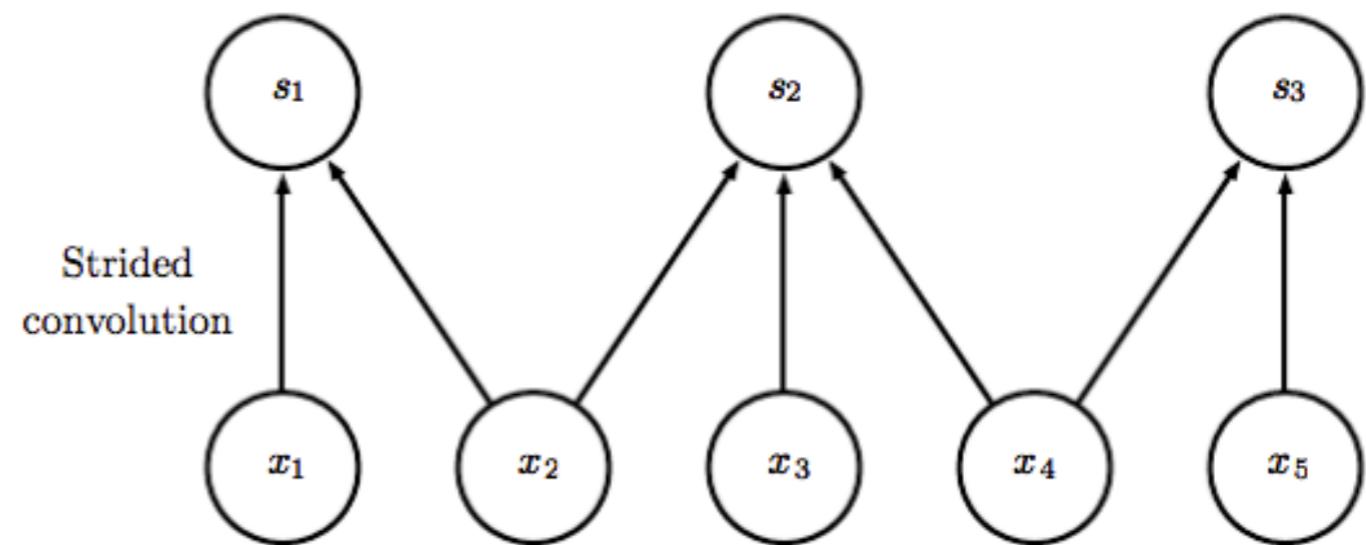
# Redes Neuronales Convolucionales

- 

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

# Redes Neuronales Convolucionales

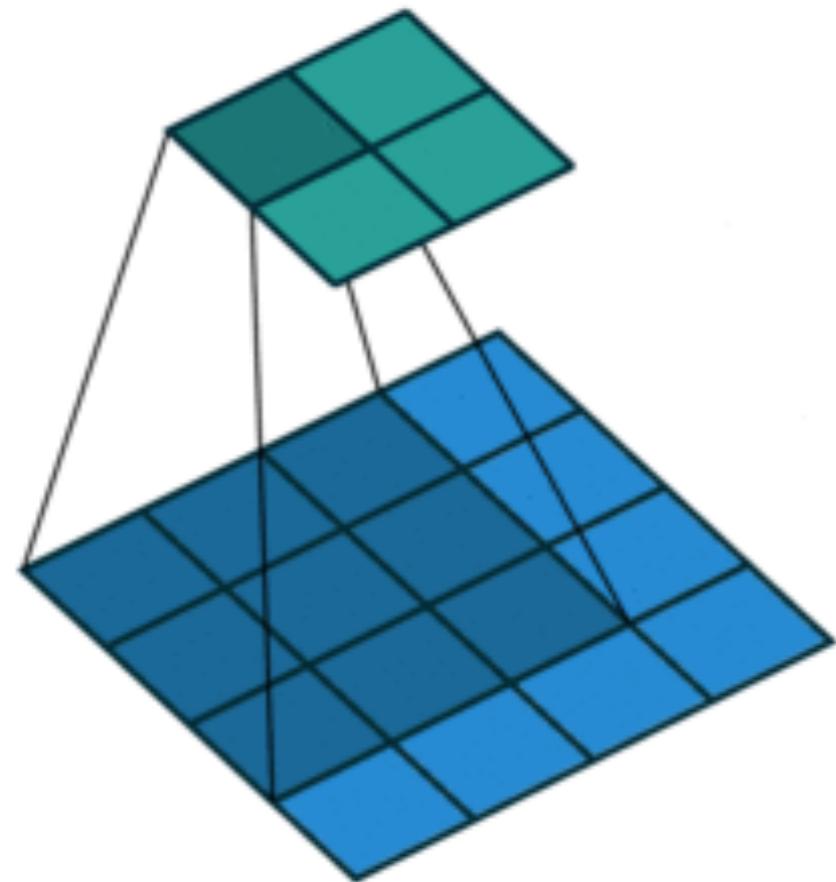
- También, uno podría querer saltar algunas de las posiciones con el kernel, para reducir al costo computacional.
- Esto se puede ver como submuestrear la entrada.
- **Stride:** Número de posiciones que saltar.



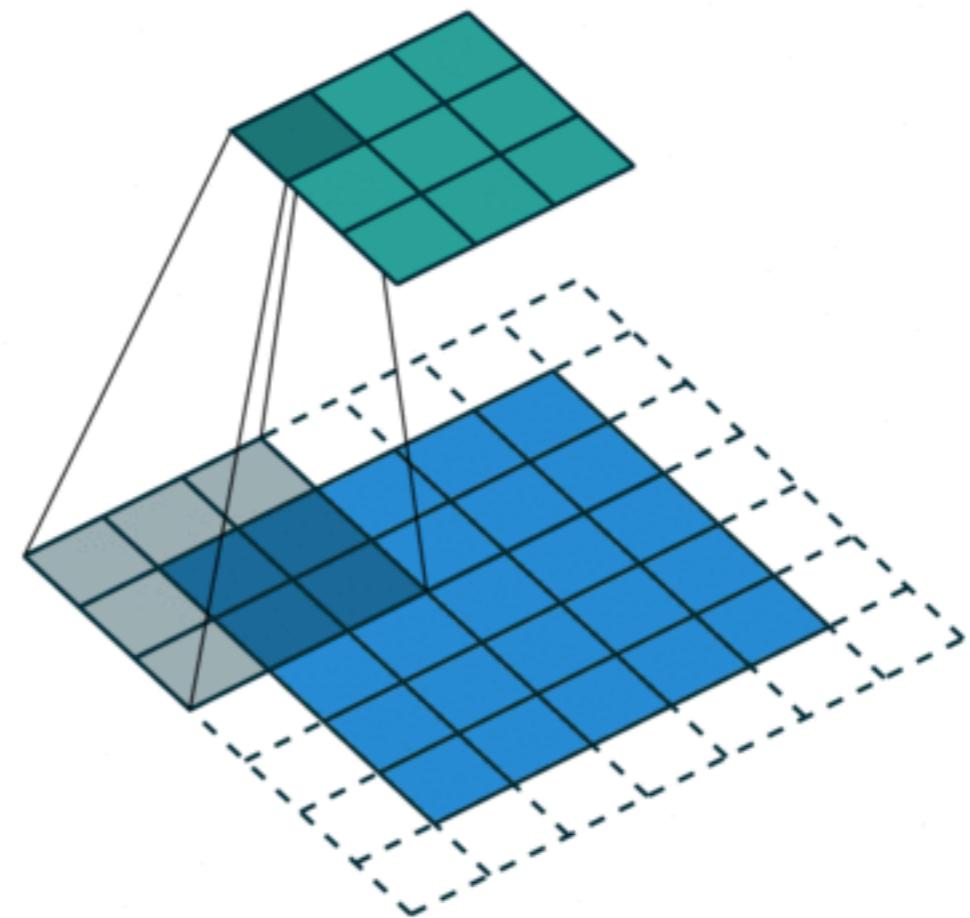
# Redes Neuronales Convolucionales

- La convolución reduce el tamaño del vector de salida a  $m - k + 1$ . Para evitar eso podemos agregar valores a los lados (**padding**).
- **valid**: Reduce tamaño (no usar padding).
- **same**: Agregar ceros a los lados, para mantener el tamaño de entrada

# Redes Neuronales Convolucionales

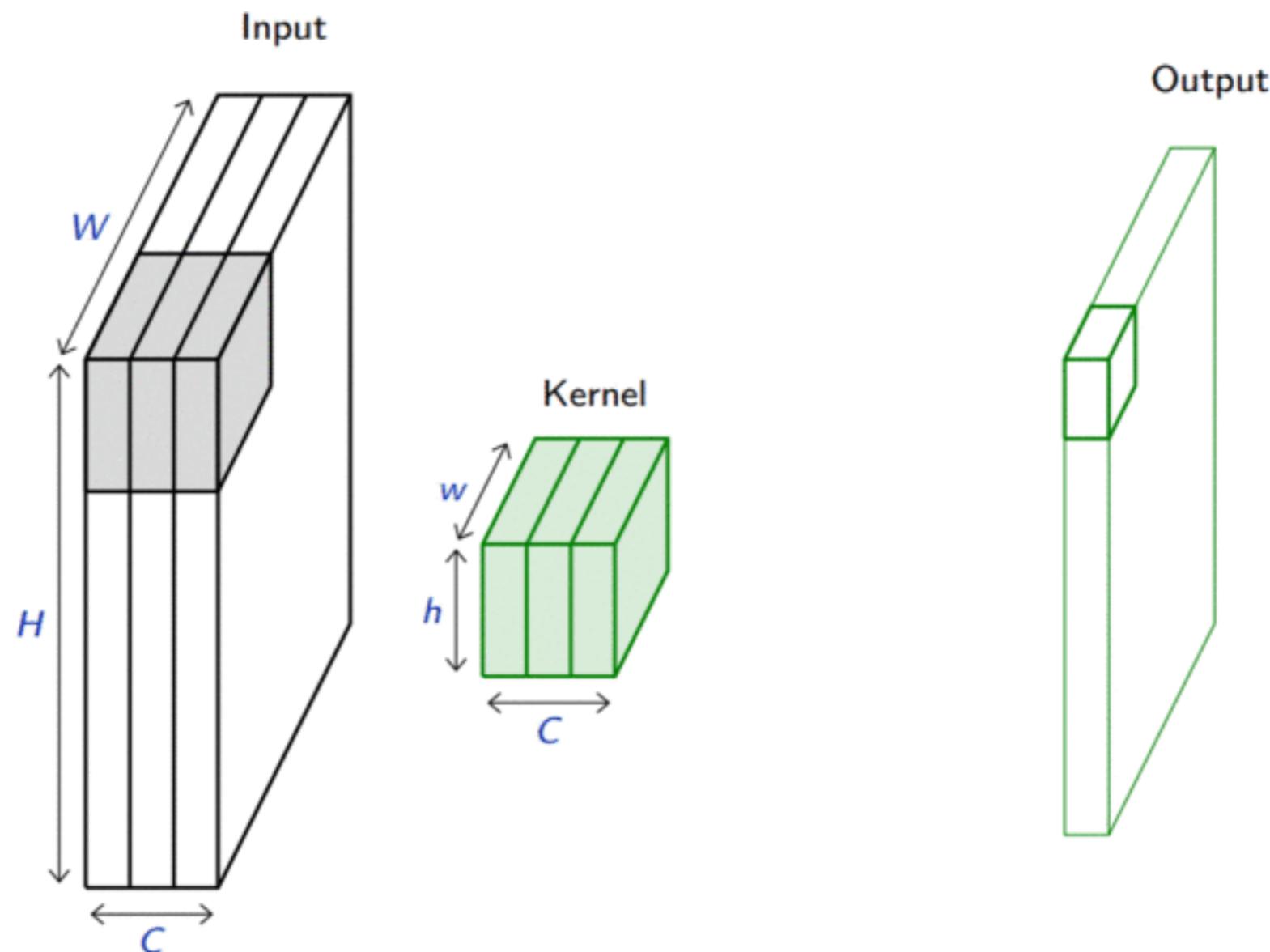


No padding  
Stride = 1



Padding  
Stride = 2

# Redes Neuronales Convolucionales

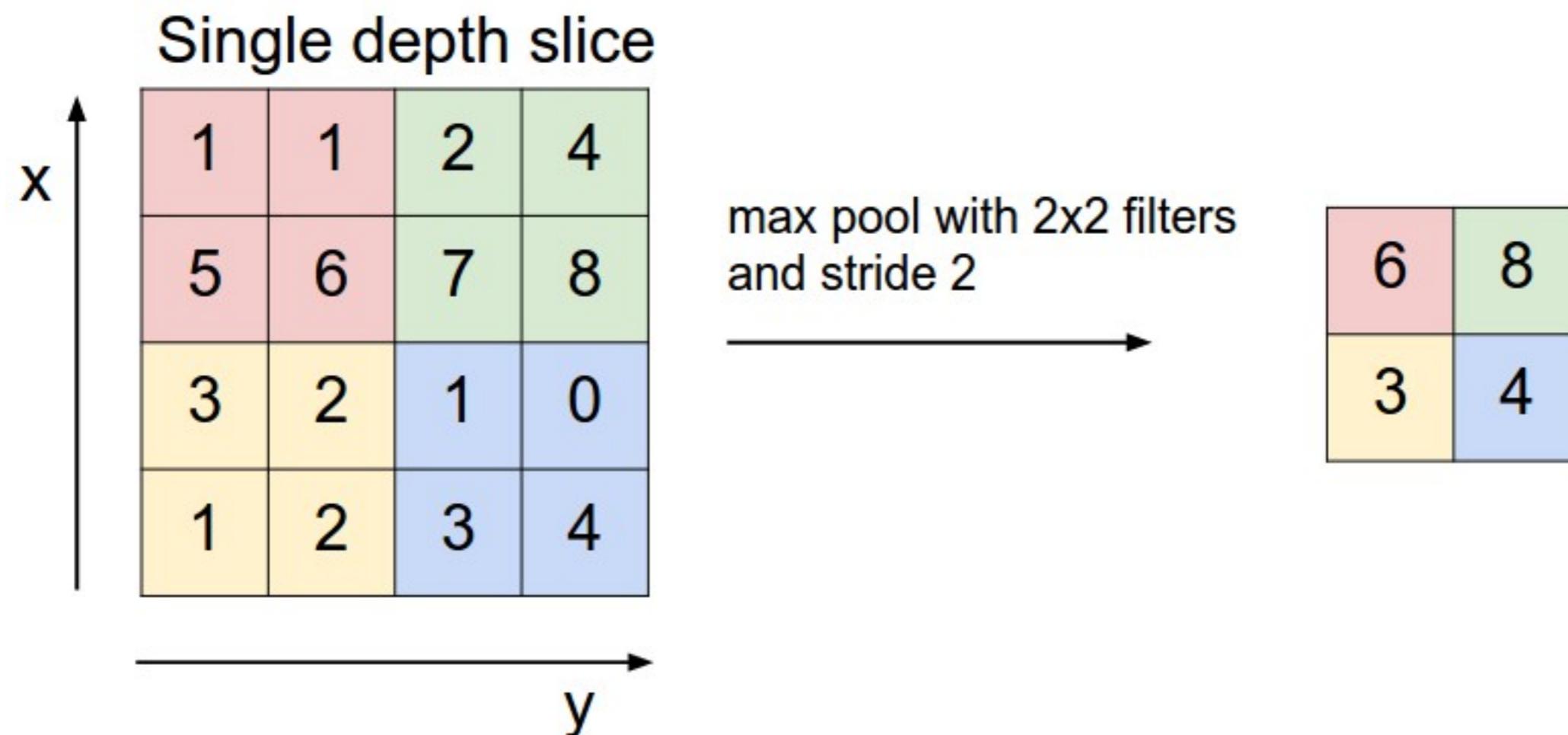


$$\text{Output Size} = (\text{Input Size} + 2 * \text{Padding} - \text{Receptive Field}) / \text{Stride} + 1$$

# Redes Neuronales Convolucionales

- **Pooling:** Reemplaza la salida en ciertas posiciones con un sumario de las salidas cercanas.
- Reduce la dimensionalidad de la salida, preservando estructuras globales.
- Algunas operaciones comunes de sumario: **Max, Mean, Weighted Mean.**
- **Stride:** Cuantas unidades mover la región de pooling.
- **Pooling Region:** El tamaño de los vecinos que tomar en la operación de pooling.

# Redes Neuronales Convolucionales

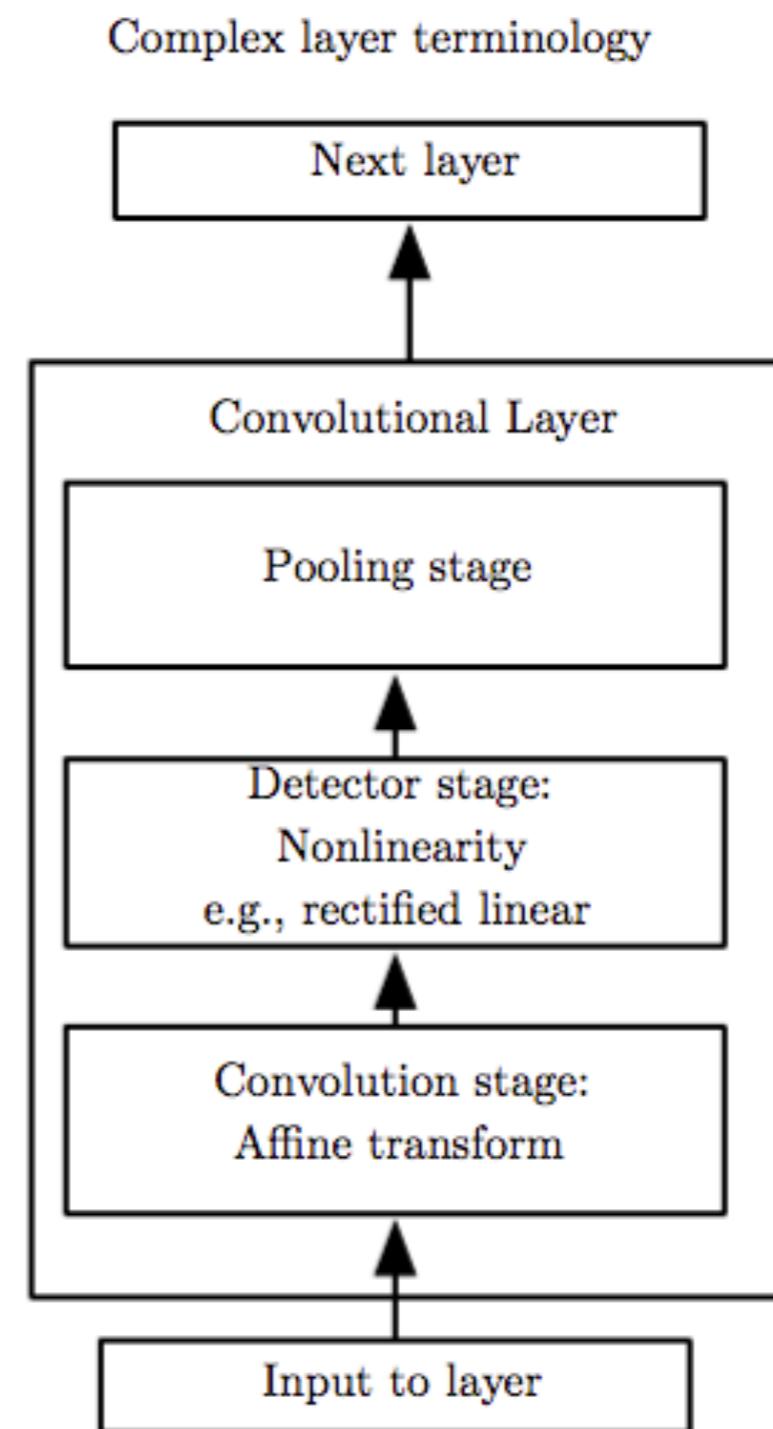


# Redes Neuronales Convolucionales

- **Pooling:** Reemplaza la salida en ciertas posiciones con un sumario de las salidas cercanas.
- **Invarianza a translaciones pequeñas:** La salida (casi) no cambia si trasladamos un poco la entrada.
- Útil si nos preocupamos más acerca de la presencia de ciertos patrones más que la posición.

# Redes Neuronales Convolucionales

- Una capa convolucional completa tiene:
  - **Convolution stage:** Varias convoluciones aplicadas a la entrada.
  - **Detector stage:** Cada salida es pasada por una operación no lineal (comúnmente **ReLU**).
  - **Pooling stage:** Se Pooling sobre la salida.



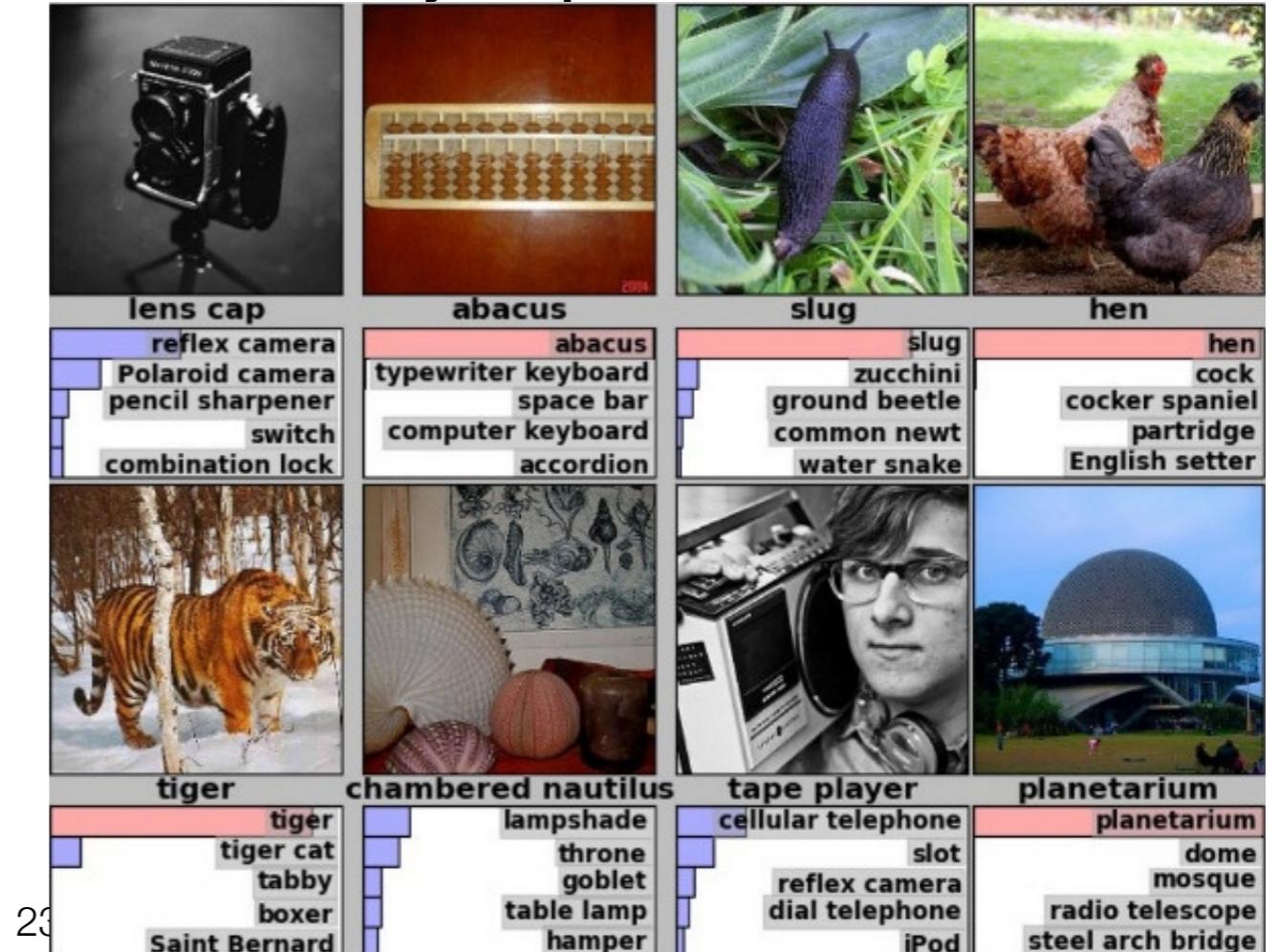
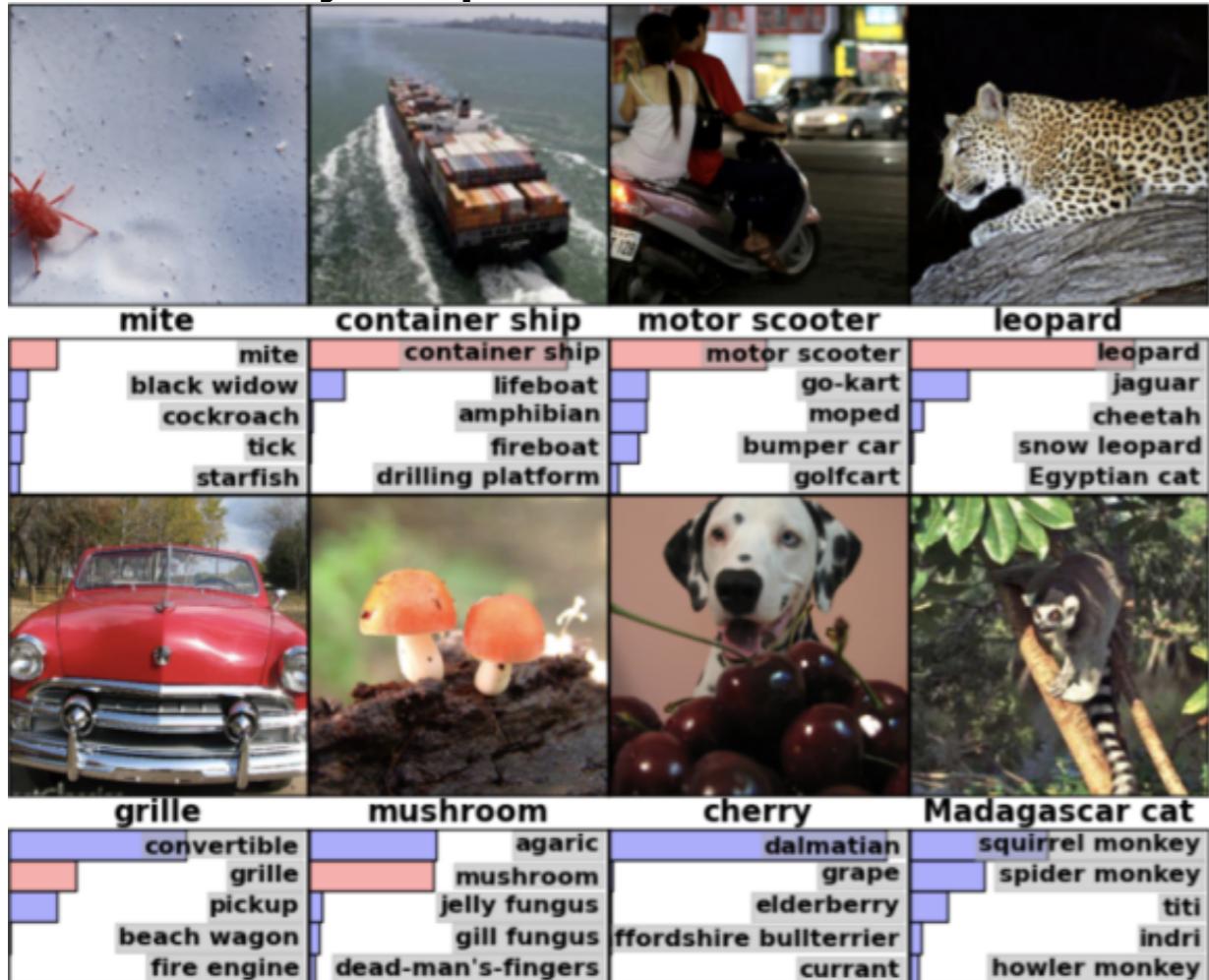
# Redes Neuronales Convolucionales

- En la capa final la salida de la convolución completa se ‘aplana’ y se pasa por una **red neuronal densa**.
- Las arquitecturas comúnmente siguen el siguiente patrón:
- **INPUT -> [[CONV -> RELU]<sup>N</sup> -> POOL]<sup>M</sup>->Flattening->[FC->RELU]<sup>K</sup>->FC->OUTPUT**

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

# Redes Neuronales Convolucionales

- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is un benchmark importante para evaluar algoritmos de visión computacional.
- Está compuesta de 1000 clases de objetos, 1200000 ejemplos de entrenamiento 100000 ejemplos de test.



# Redes Neuronales Convolucionales

## LeNet-5

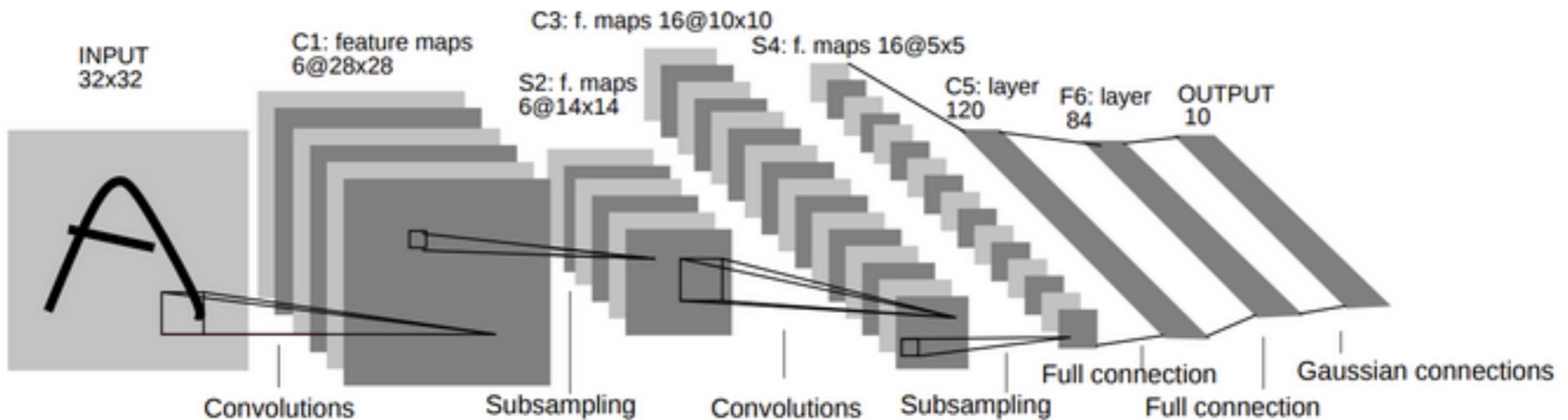


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

**LeCun et al, 1998.**

# Redes Neuronales Convolucionales

## AlexNet

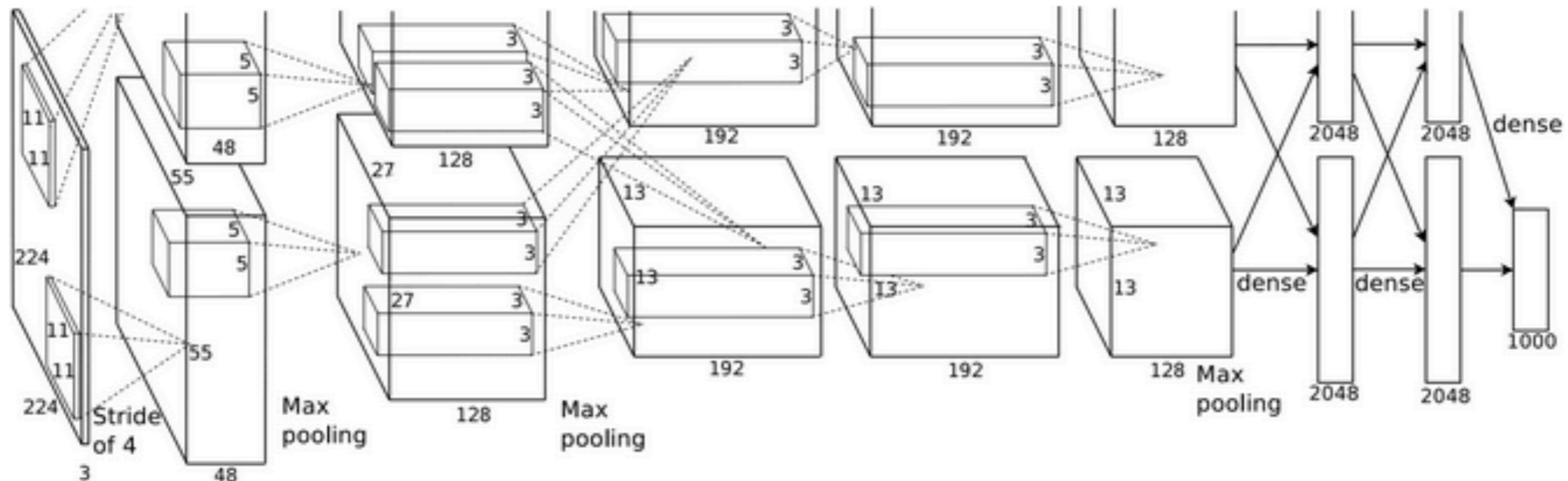
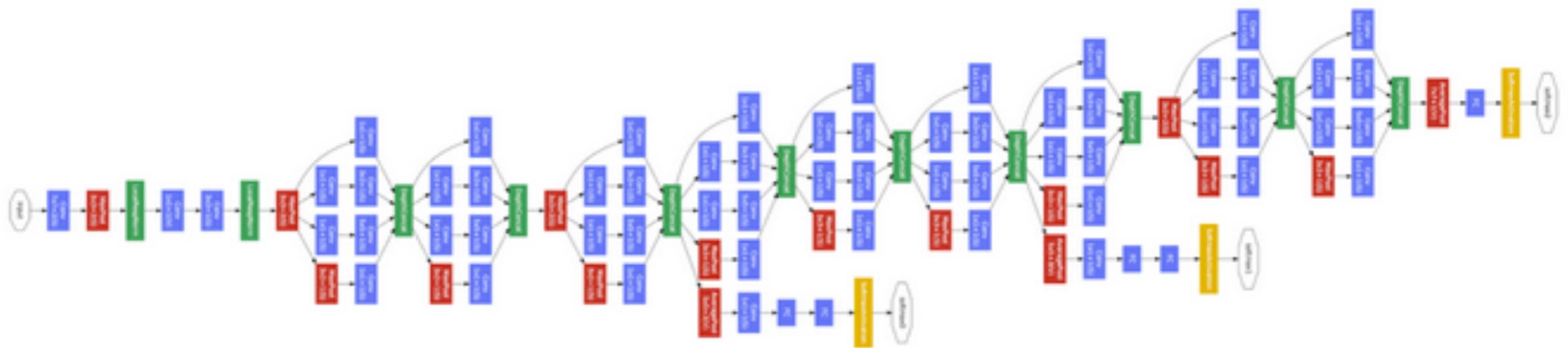


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

*Krizhevsky et al, 2012.*

# Redes Neuronales Convolucionales

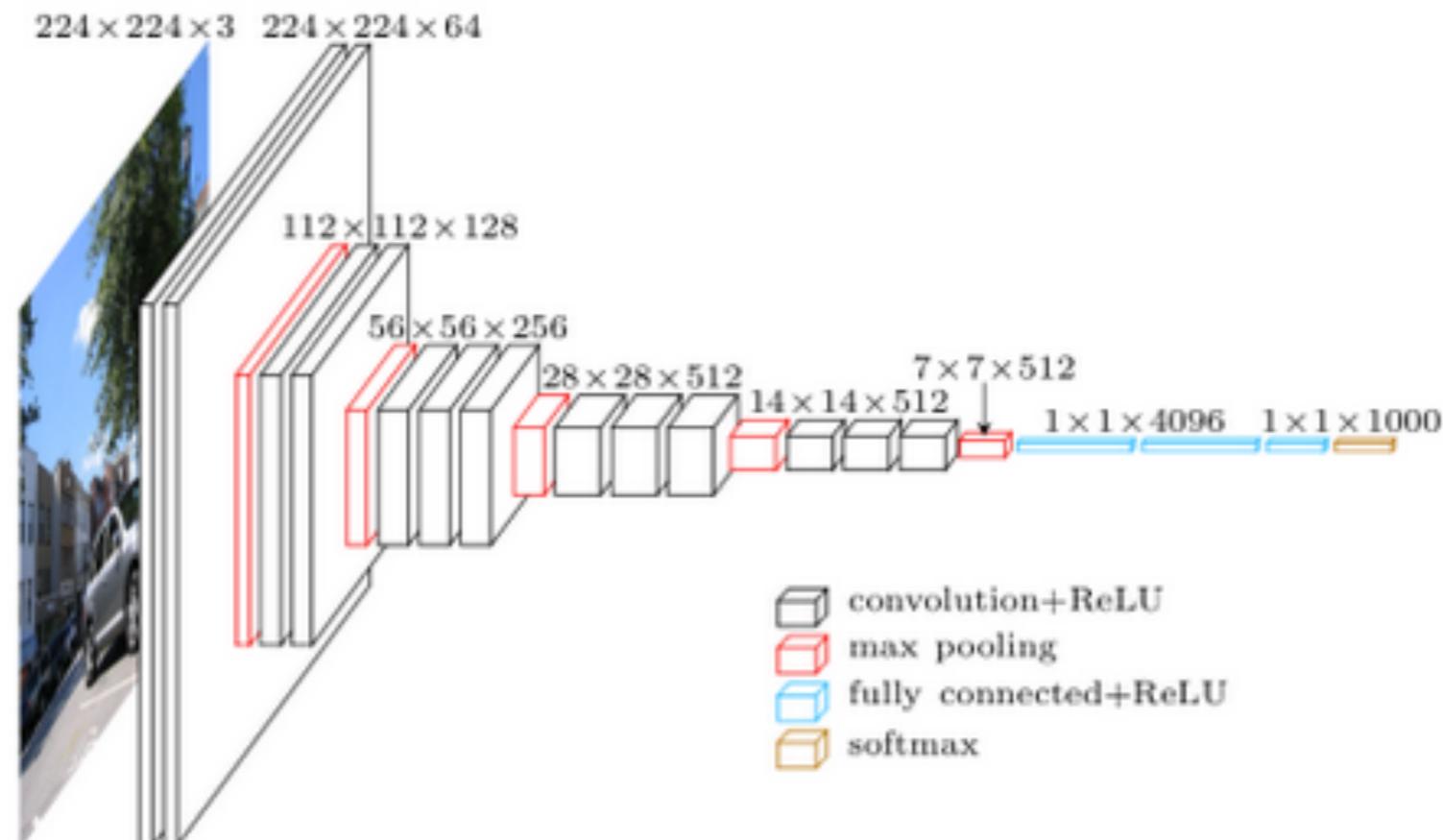
## GoogleNet



Szegedy et al, 2014.

# Redes Neuronales Convolucionales

## VGGNet



*Simonyan et al, 2014.*

# Redes Neuronales Convolucionales

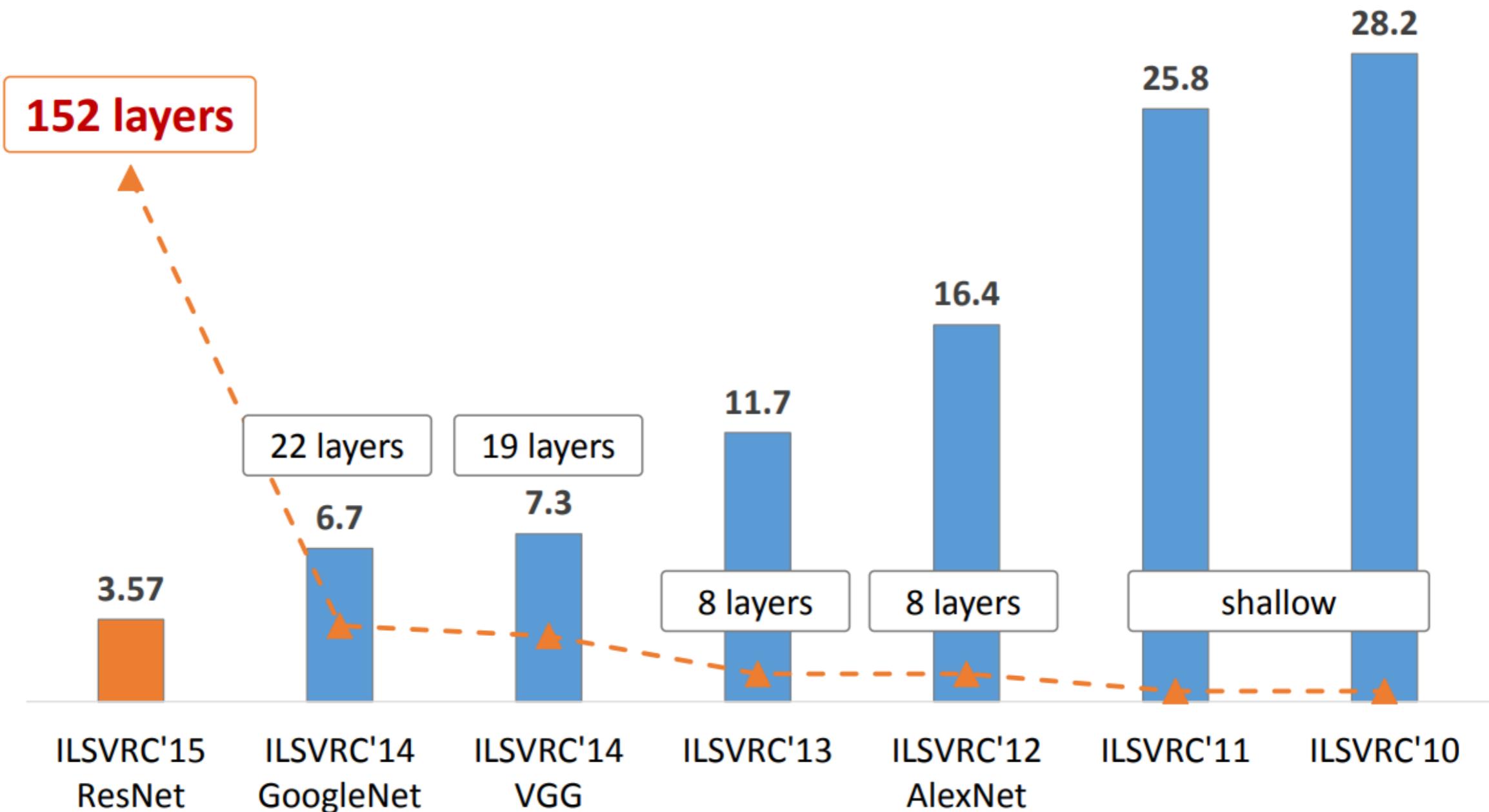
## ResNet



*He et al, 2015.*

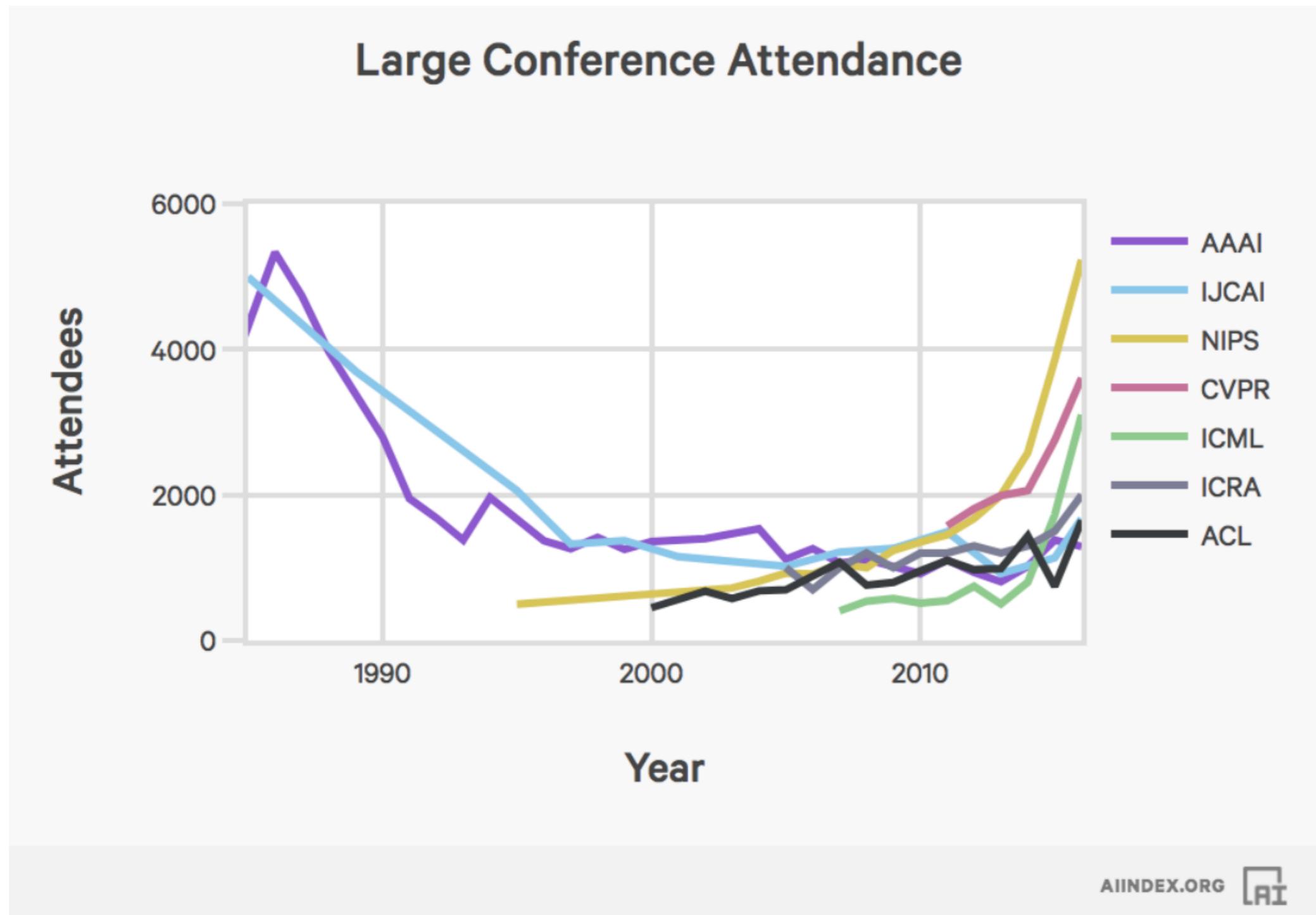
# Redes Neuronales Convolucionales

## ImageNet



# Redes Neuronales Convolucionales

## Interes

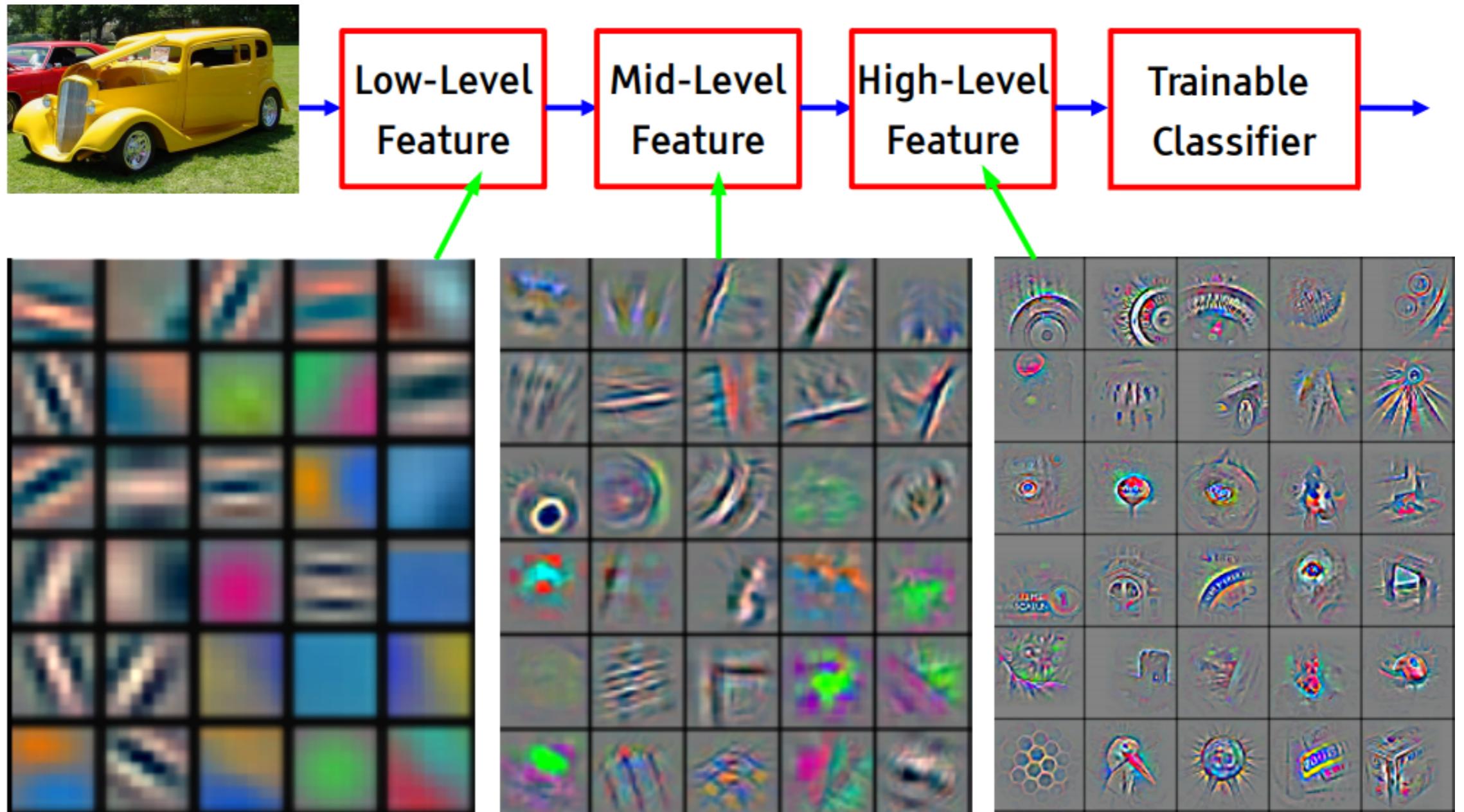


# Redes Neuronales Convolucionales

## ImageNet

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88
DenseNet121	33 MB	0.745	0.918	8,062,504	121
DenseNet169	57 MB	0.759	0.928	14,307,880	169
DenseNet201	80 MB	0.770	0.933	20,242,984	201

# Redes Neuronales Convolucionales



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

# **Redes Neuronales recurrentes para el procesamiento de secuencias**

# Redes Neuronales Recurrentes

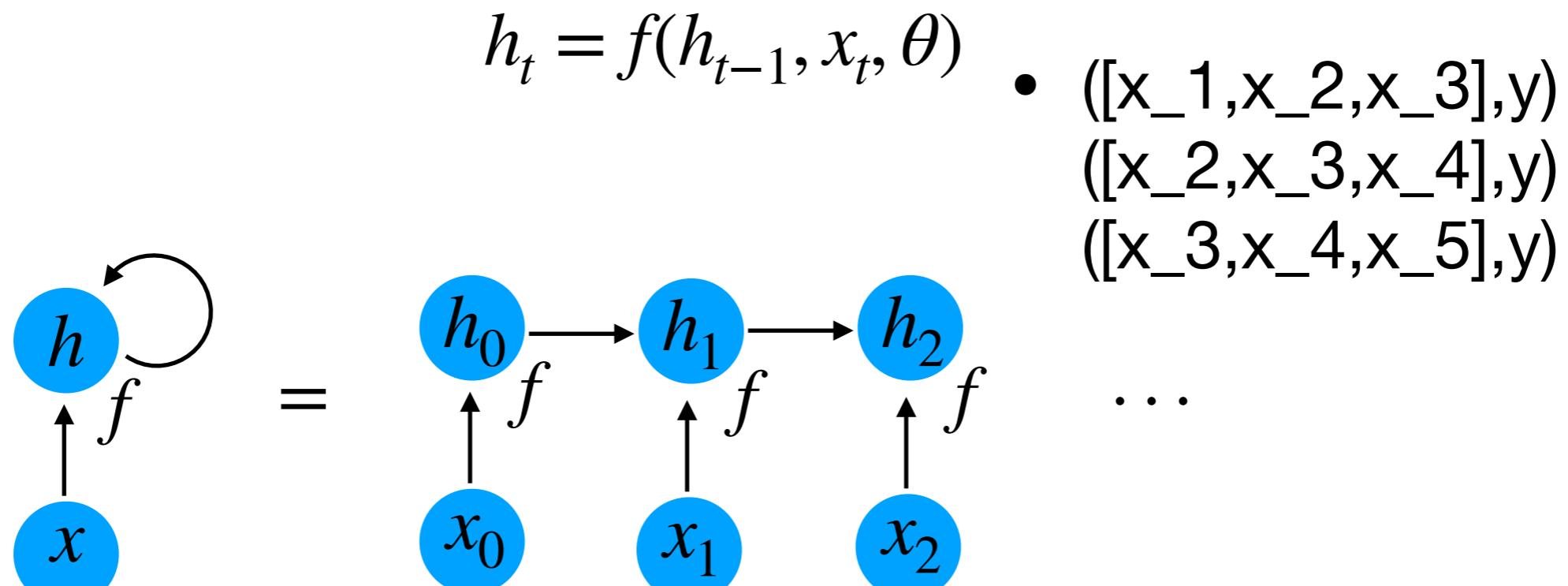
- **Las redes neuronales recurrentes** son como redes neuronales convolucionales pero especializadas en secuencias (texto, series de tiempo, etc.)
- Permiten **compartir parámetros** para reducir el número de parámetros que aprender.
- Permite analizar secuencias de **cualquier tamaño**, sin tener que cambiar la arquitectura.

# Redes Neuronales Recurrentes

- Considerar una **secuencia de entrada**  $\{x_1, x_2, x_3, \dots, x_N\}$  por ejemplo una secuencia de palabras en una oración o una serie de precios en una serie de tiempo de precios de acciones.
- La idea de la red neuronal recurrente es usar una función única para procesar cada valor de la secuencia.
- Pero también mantiene un **estado oculto** que es actualizado **recurrentemente**.

# Redes Neuronales Recurrentes

- Considerar la secuencia de entrada  $\{x_1, x_2, x_3, \dots, x_N\}$



Representación desenvuelta

# Redes Neuronales Recurrentes

- Considerar una secuencia de entrada  $\{x_1, x_2, x_3, \dots, x_N\}$

$$h_t = f(h_{t-1}, x_t, \theta)$$

- Vamos a parametrizar  $f$  con una red neuronal simple

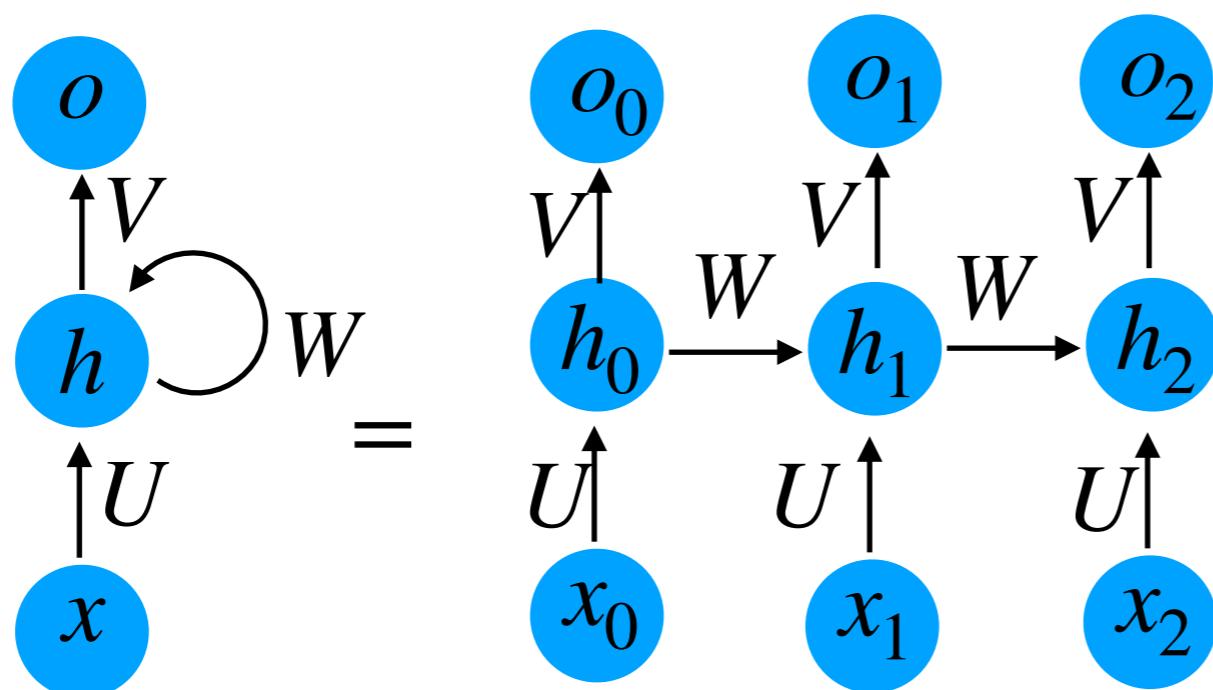
$$\begin{aligned}f(h_{t-1}, x_t, \theta) &= g(b + Wh_{t-1} + Ux_t) \\&= \tanh(b + Wh_{t-1} + Ux_t)\end{aligned}$$

# Redes Neuronales Recurrentes

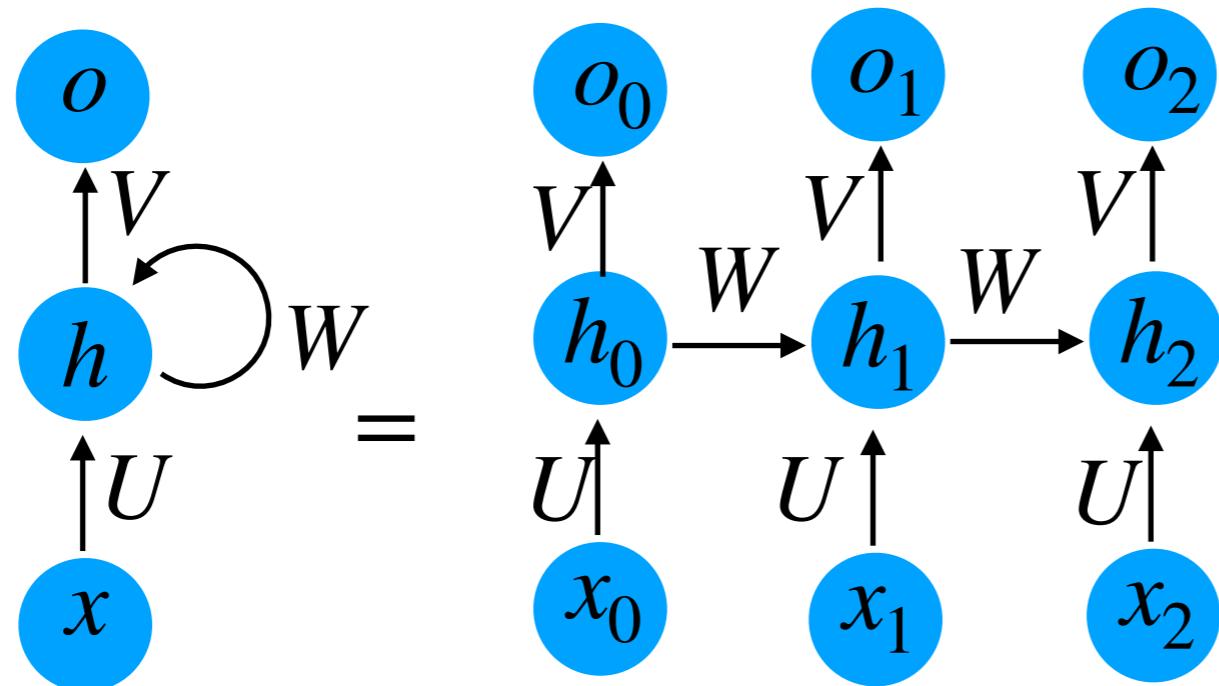
- Considerar una secuencia de entrada  $\{x_1, x_2, x_3, \dots, x_N\}$  y una secuencia de salida que queremos predecir:  
 $\{y_1, y_2, y_3, \dots, y_N\}$
- Ejemplos:
  - Próxima palabra en una secuencia.
  - Comprar o no comprar en una serie de tiempo de precios de acciones.
  - Sí la palabra es un insulto o no.

# Redes Neuronales Recurrentes

- Considerar una secuencia de entrada  $\{x_1, x_2, x_3, \dots, x_N\}$ . y considerar una secuencia de salida que queremos predecir:  $\{y_1, y_2, y_3, \dots, y_N\}$



# Redes Neuronales Recurrentes



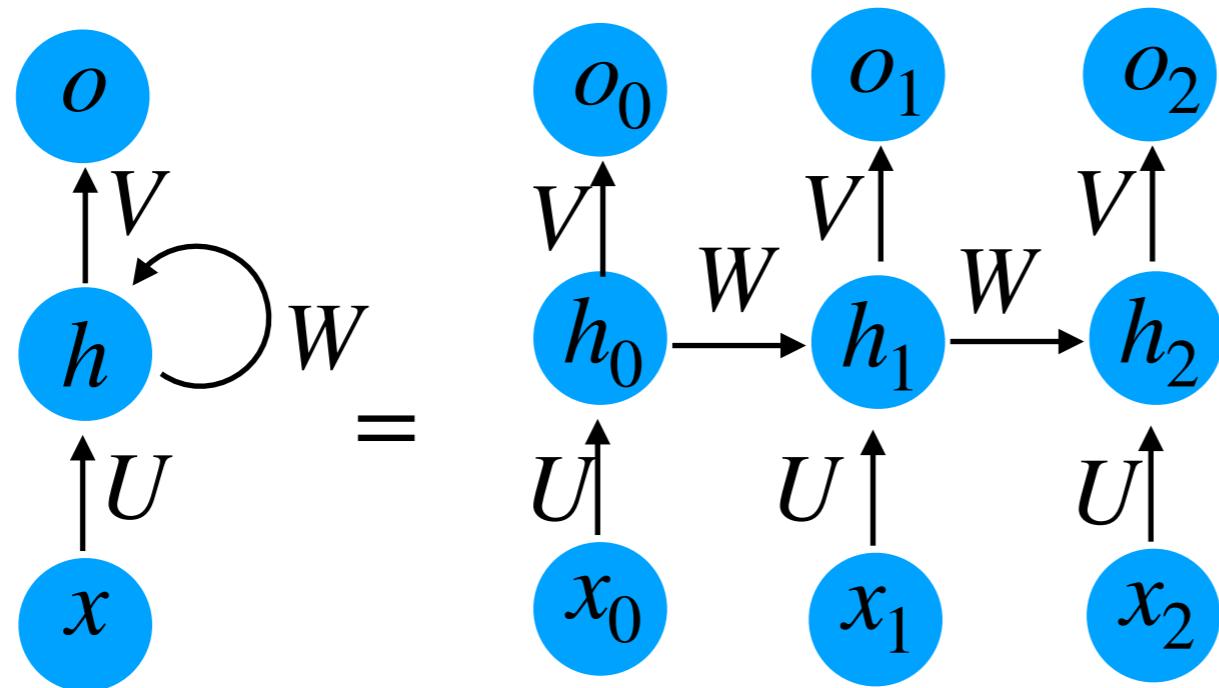
•

$$h_t = \tanh(b + Wh_{t-1} + Ux_t)$$

$$o_t = c + Vh_t$$

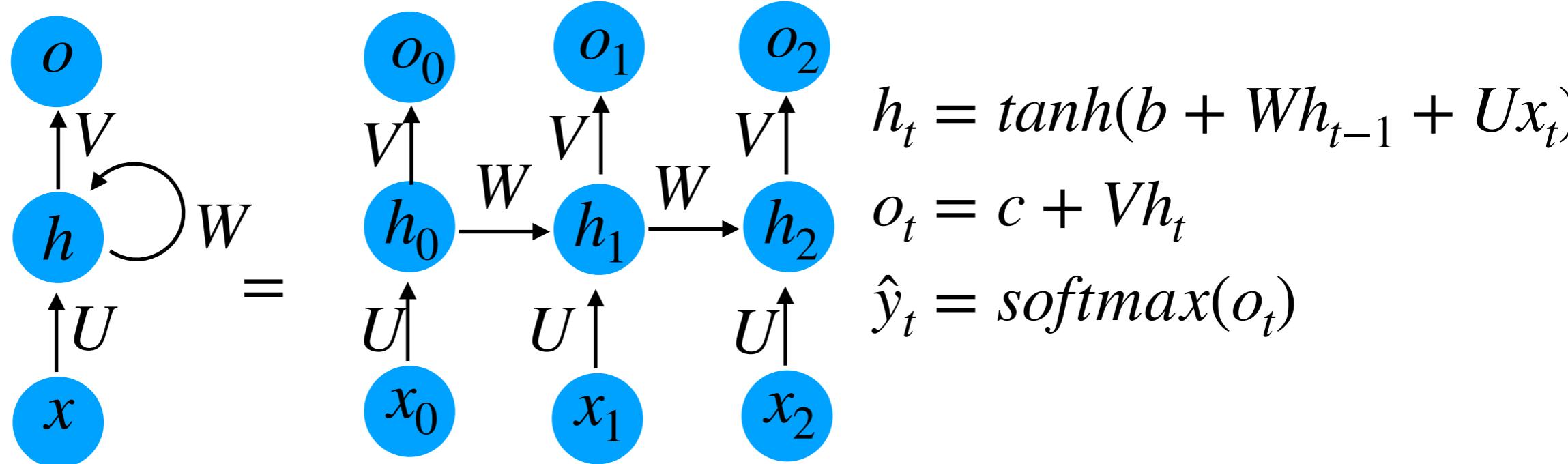
$$\hat{y}_t = \text{softmax}(o_t)$$

# Redes Neuronales Recurrentes



- $$h_t = \tanh(b + Wh_{t-1} + Ux_t)$$
$$o_t = c + Vh_t$$
$$\hat{y}_t = softmax(o_t)$$
$$L(x_1, \dots, x_T, y_1, \dots, y_T) = \sum_t^T L(\hat{y}_t, y_t)$$

# Redes Neuronales Recurrentes



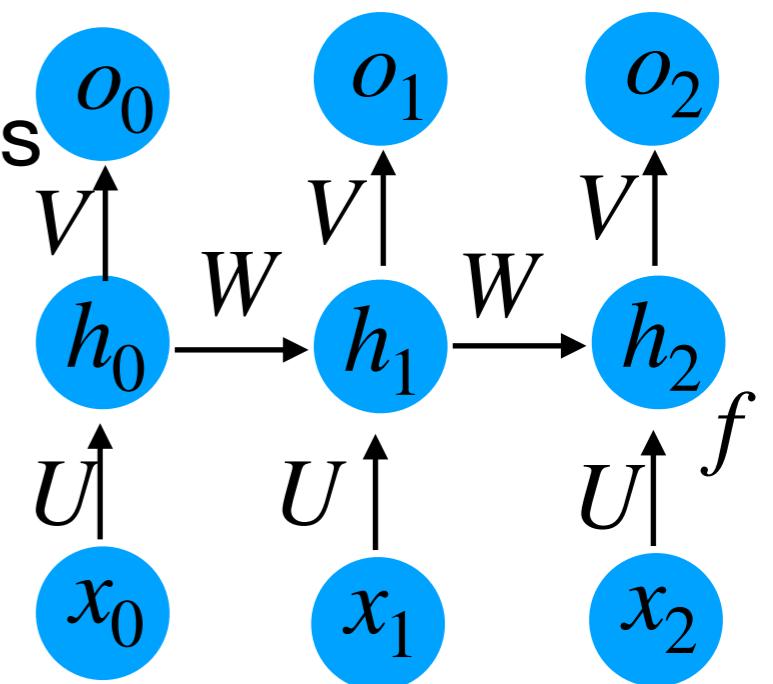
- $$L(x_1, \dots, x_T, y_1, \dots, y_T) = \sum_t^T L(\hat{y}_t, y_t)$$
$$= - \sum_t I(y_i = c) \log \hat{y}_{tc}$$

# Redes Neuronales Recurrentes

- La pregunta es como calcular los gradientes de los pesos de una RNN.
- Se vemos la red como desenvuelta en el tiempo, se observa que la red es una red neuronal profunda con varias capas (igual al número de pasos) y podemos usar backpropagation. Esto es conocido como **backpropagation through time**.

- En cada paso actualizamos los mismos pesos por lo que el error se debe acumular

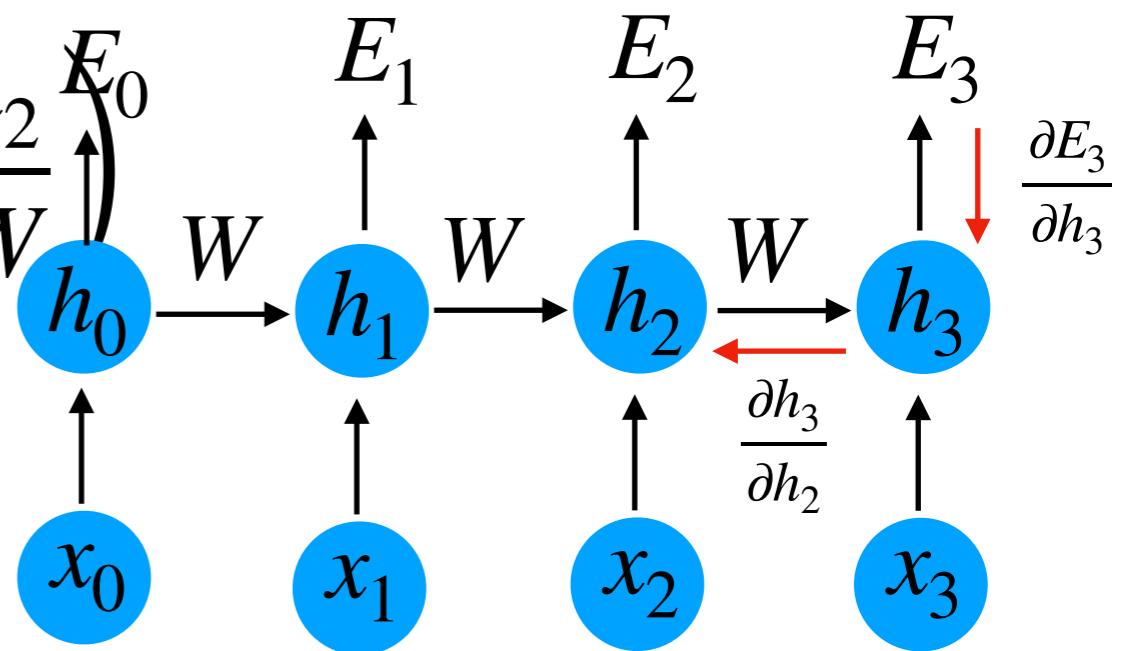
$$\frac{\partial E}{\partial W} = \sum_{t=1}^T \frac{\partial E_t}{\partial W}$$



# Redes Neuronales Recurrentes

- Vanishing/Exploding Gradient

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \left( \frac{\partial h_3}{\partial W} + \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial W} \right)$$



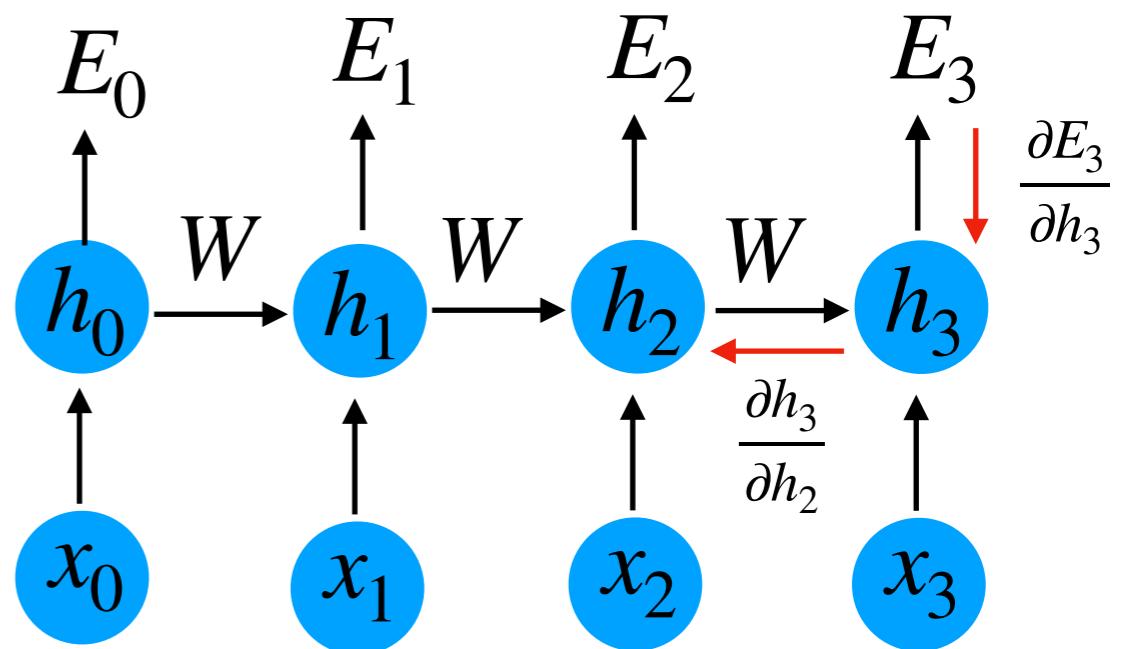
$$h_3 = Wh_2 + Ux_3$$

$$h_2 = Wh_1 + Ux_2$$

# Redes Neuronales Recurrentes

- Vanishing/Exploding Gradient

$$\frac{\partial E_3}{\partial W} =$$



$$h_3 = Wh_2 + Ux_3$$

$$h_2 = Wh_1 + Ux_2$$

# Redes Neuronales Recurrentes

- Vanishing/Exploding Gradient

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^t \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^t \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^t W^T \text{diag}[g'(h_{i-1})]$$

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| \leq \|W^T\| \| \text{diag}[g'(h_{i-1})] \| \leq \gamma_W \gamma_\phi$$

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| \leq (\gamma_W \gamma_\phi)^{t-k}$$

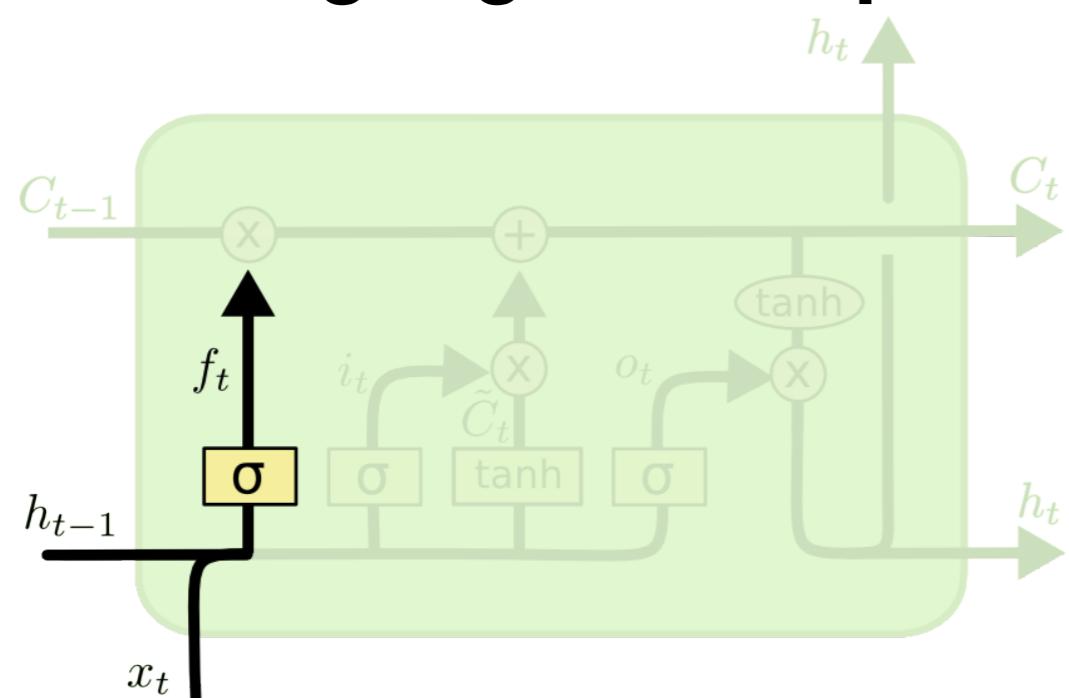
On the difficulty of training Recurrent Neural Networks, Pascanu et al.

# Redes Neuronales Recurrentes

- **LSTM (Long-Short Term Memory)**
- Las LSTM fueron introducidas para resolver el problema de vanishing/exploding gradient.
- Pueden controlar la cantidad de data que es ‘guardada’ en memoria, y en esa forma pueden controlar la cantidad de gradiente que ‘pasa’ por la red.
- Para eso añade un **estado de celda** que la red aprende a actualizar.

# Redes Neuronales Recurrentes

- **LSTM (Long-Short Term Memory)**
- En cada paso la unidad LSTM recibe el estado oculto anterior  $h_{t-1}$  y una entrada  $x_t$  y actualiza el estado de la celda  $C_t$
- 1. **Forget gate computation:**

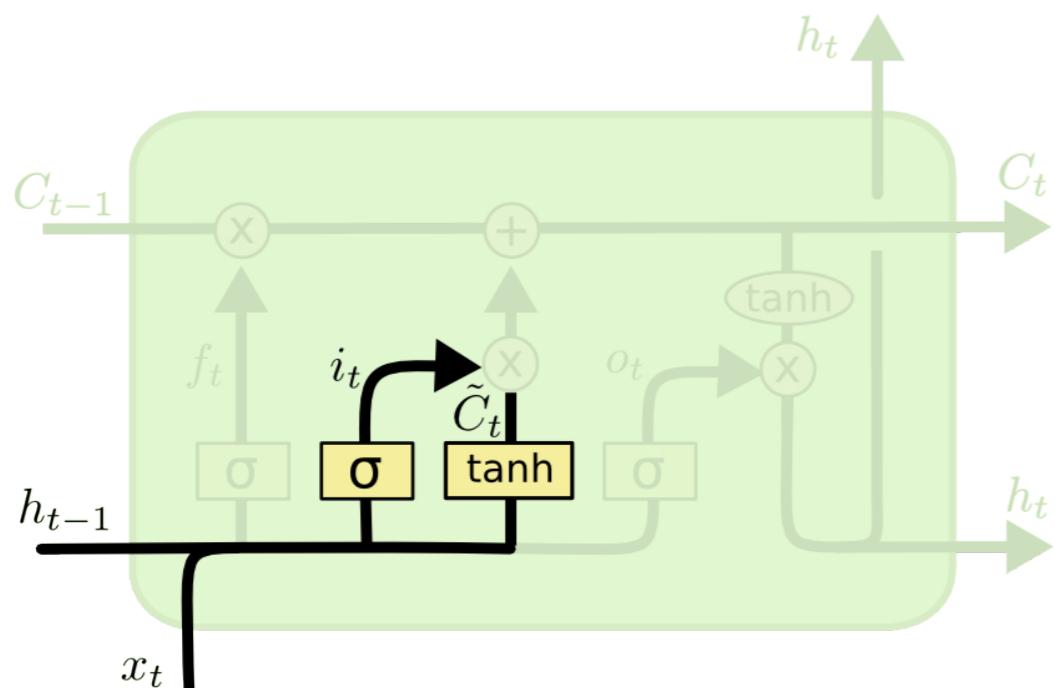


$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

*Credit: Understanding LSTM Networks. Colah's blog*

# Redes Neuronales Recurrentes

- **LSTM (Long-Short Term Memory)**
- En cada paso la unidad LSTM recibe el estado oculto anterior  $h_{t-1}$  y una entrada  $x_t$  y actualiza el estado de la celda  $C_t$
- **1. Input gate computation and new cell state:**

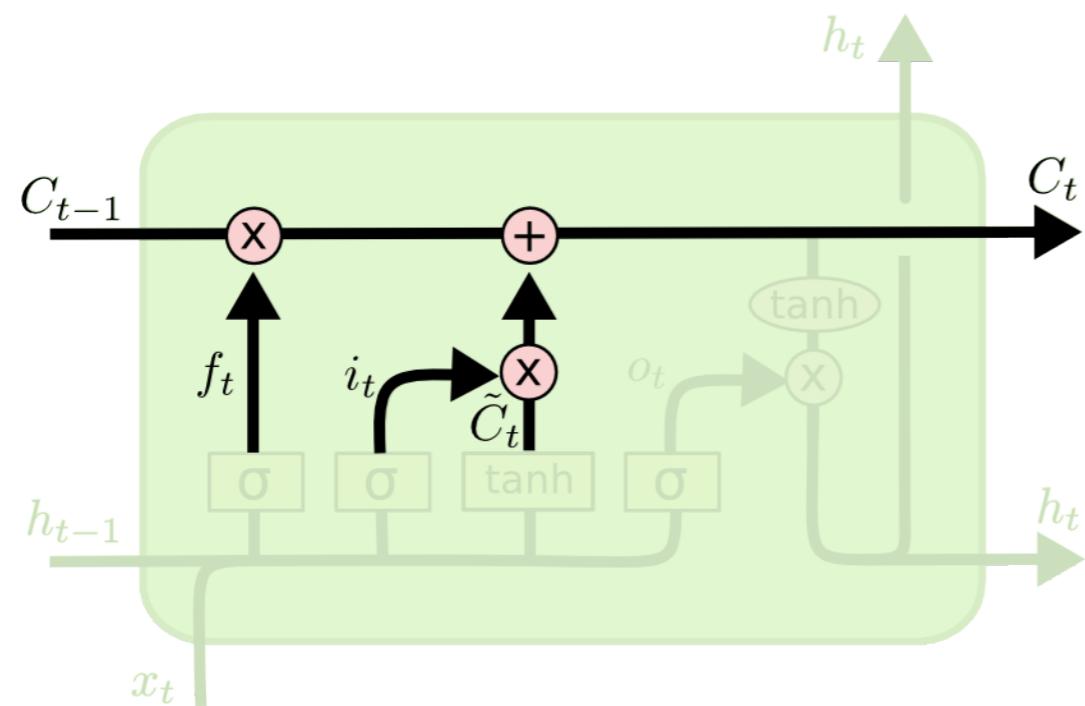


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

*Credit: Understanding LSTM Networks. Colah's blog*

# Redes Neuronales Recurrentes

- **LSTM (Long-Short Term Memory)**
- En cada paso la unidad LSTM recibe el estado oculto anterior  $h_{t-1}$  y una entrada  $x_t$  y actualiza el estado de la celda  $C_t$
- **Cell State Update:**

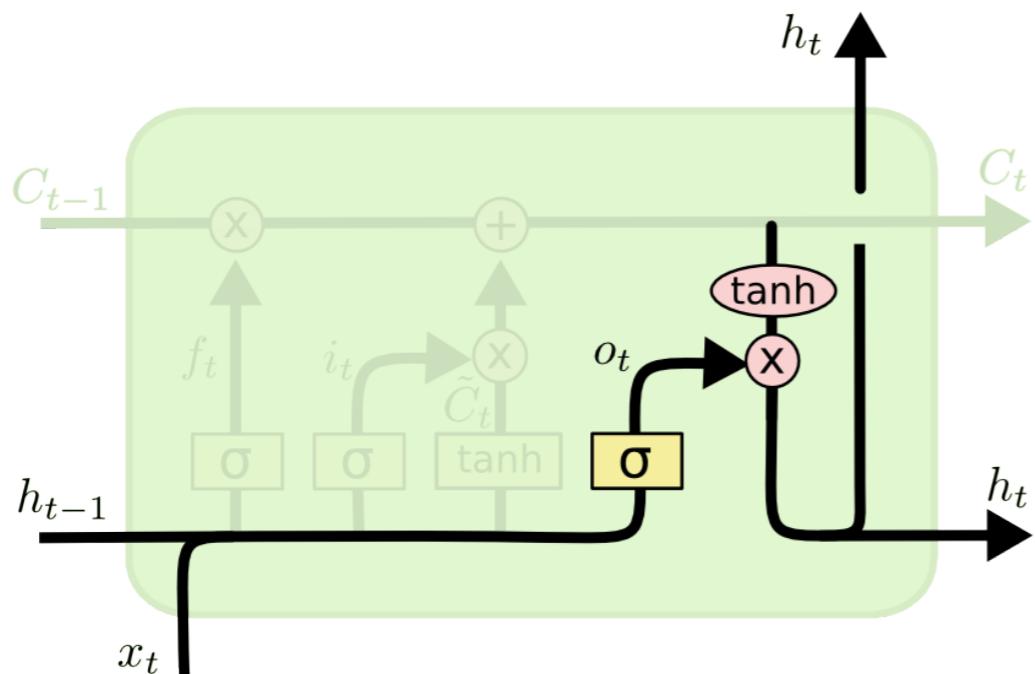


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

*Credit: Understanding LSTM Networks. Colah's blog*

# Redes Neuronales Recurrentes

- **LSTM (Long-Short Term Memory)**
- En cada paso la unidad LSTM recibe el estado oculto anterior  $h_{t-1}$  y una entrada  $x_t$  y actualiza el estado de la celda  $C_t$
- **1. Output and hidden state computation:**



$$o_t = \sigma (W_o [ h_{t-1}, x_t ] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

*Credit: Understanding LSTM Networks. Colah's blog*

# Batch Normalization

- **Batch Normalization**
- En redes neuronales normalizamos las entradas, para facilitar el trabajo. Sin embargo a medida que los datos pasan por las capas, la normalización se pierde
  - -> Explosión de pesos o vanishing de pesos
- Para solucionarlo vamos a normalizar cada una de las salidas en cada capa (features).

# Batch Normalization

- **Batch Normalization**
- Para solucionarlo vamos a normalizar cada una de las salidas en cada capa (features).
- Para evitar tener que calcular media y varianza en todo el dataset utilizaremos los datos del mini Batch.

$$\mu_b = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\cdot \sigma_b^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_b)^2$$

# Batch Normalization

- **Batch Normalization**

$$\mu_b = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\bullet \quad \sigma_b^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_b)^2$$

- Finalmente normalizaremos la entrada y agregaremos unos parámetros ‘scale and shift’  $\gamma, \beta$  que se pueden aprender.

# Batch Normalization

- **Batch Normalization**

- 

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

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

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

# Batch Normalization

- **Batch Normalization**

$$\mu_b = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\cdot \sigma_b^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_b)^2$$

- En el **test set** estas estadísticas podrían ser problemáticas, ya que podemos tener Batch Size incluso de 1. Para resolverlo se usa una estimación de la media y la varianza obtenida como un promedio de las medias y varianzas de cada Batch.

# Layer Normalization

- **Layer Normalization**
- Para evitarnos los problemas con los Batch, calculamos la media y la varianza no por cada feature (sumada sobre el batch), si no que para cada dato (sumada sobre todas las features).

$$\mu_H = \frac{1}{H} \sum_{i=1}^H x_i$$

$$\sigma_H^2 = \frac{1}{H} \sum_{i=1}^H (x_i - \mu_H)^2$$

- Donde H es el número de features (hidden units) en la capa.
- Luego se aplica scale and shift.
- Reduce variability of gradients

# Transformers

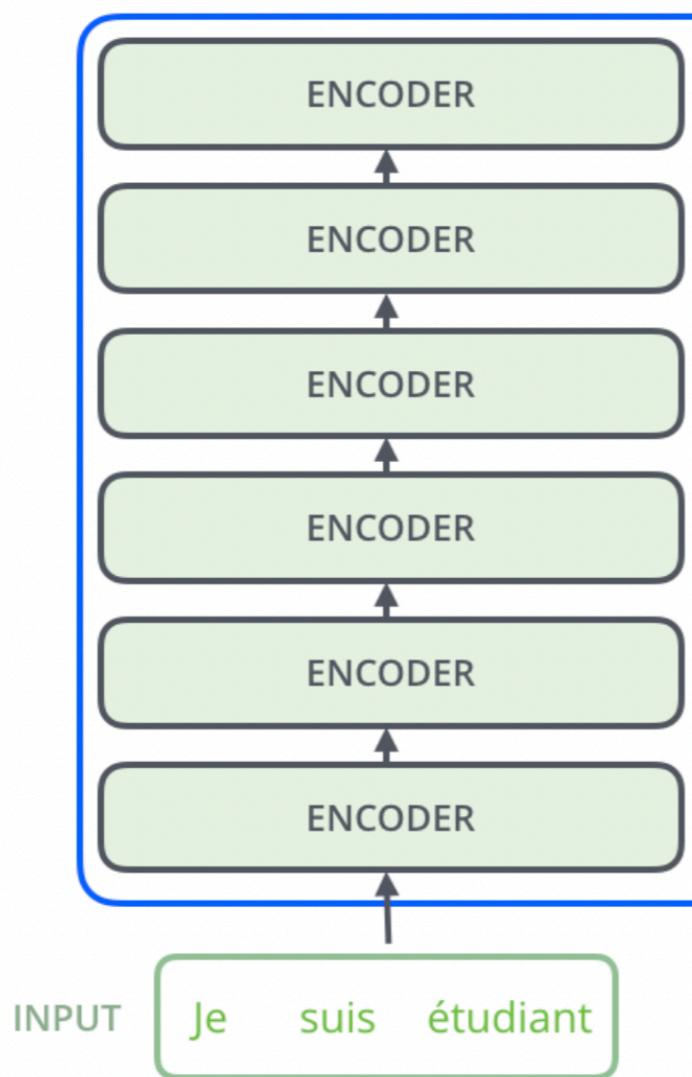
- **Transformers**
- Este es el modelo más nuevo que veremos, se originó en 2016 (Vaswani et al.) y se ha convertido en el modelo más popular, usado en modelos importantes como BERT, GPT-3, y otros.
- Ha comenzado a ganar popularidad también en el área de computer visión.
- Word embeddings
- Hola -> [0.2,0.1,-0.1] Representaciones de tamaño fijo
- Perro -> [-0.1,0.2, 0.5]

*Credit: Illustrated Transformer - <https://jalammar.github.io/illustrated-transformer/>*

# Transformers

- **Transformers**

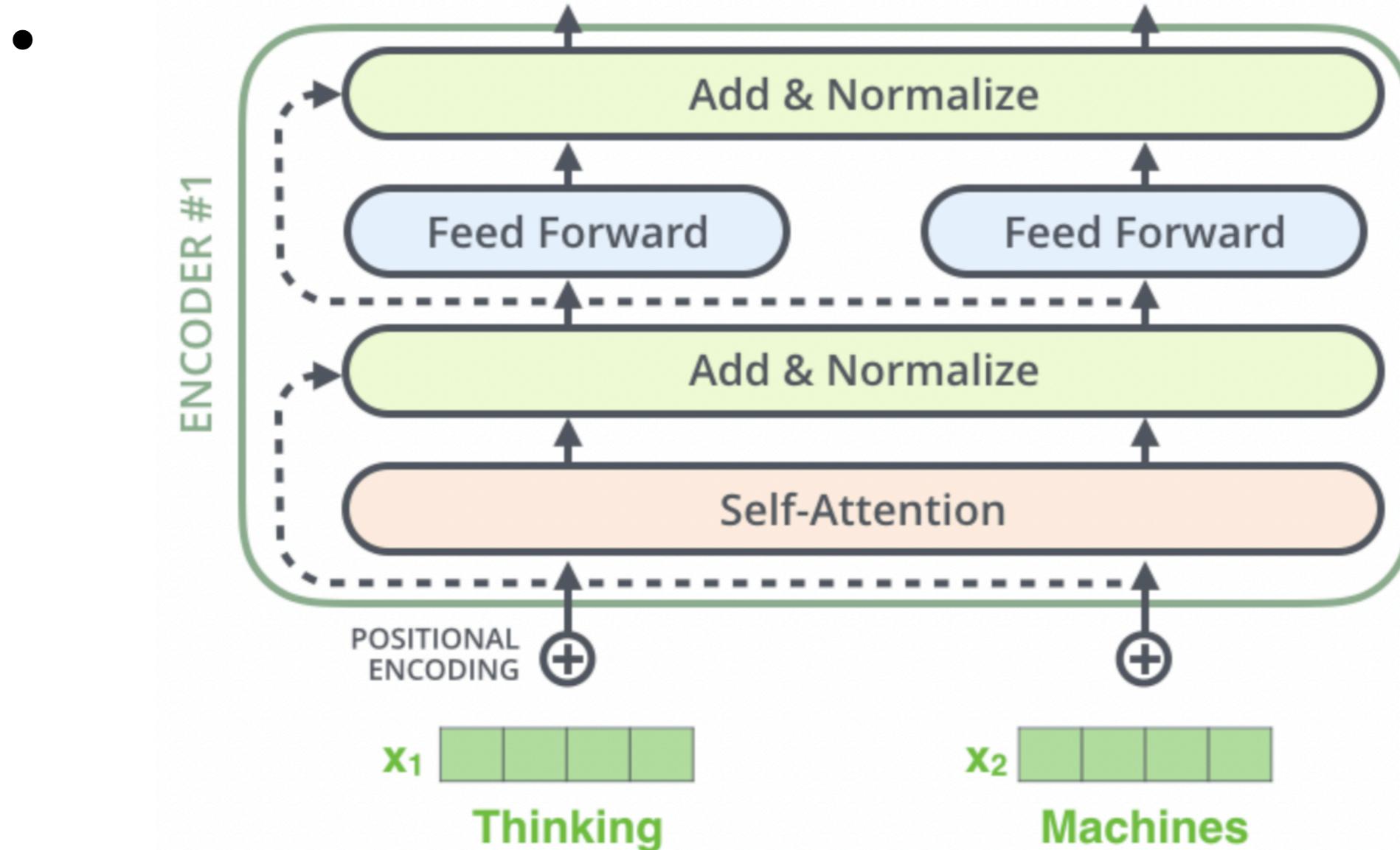
- 



Credit: Illustrated Transformer - <https://jalammar.github.io/illustrated-transformer/>

# Transformers

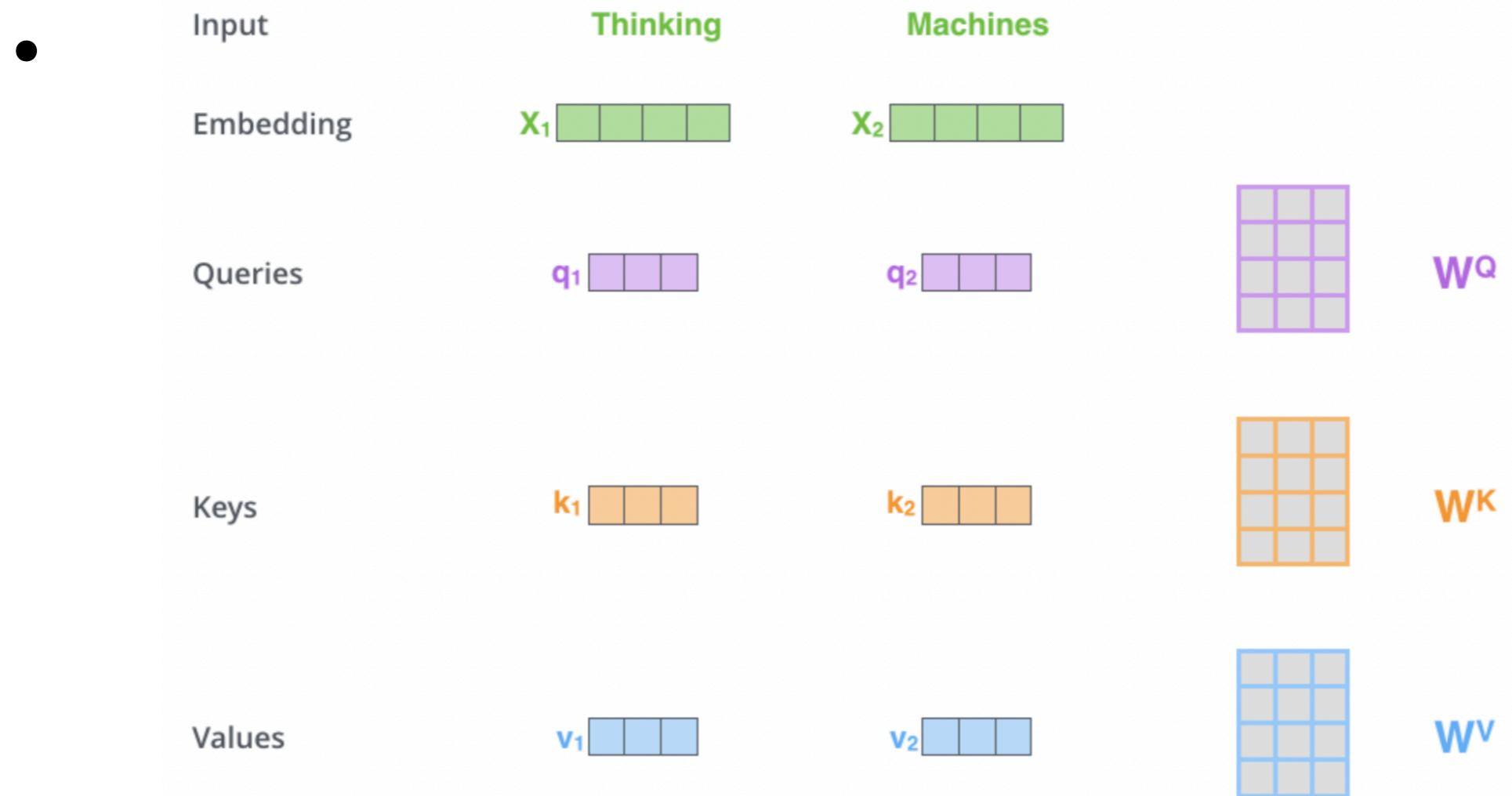
- **Transformers**



Credit: Illustrated Transformer - <https://jalammar.github.io/illustrated-transformer/>

# Transformers

- **Transformers**



Credit: Illustrated Transformer - <https://jalammar.github.io/illustrated-transformer/>

# Transformers

- **Transformers**

- 

$$\mathbf{X} \times \mathbf{W}^Q = \mathbf{Q}$$

A diagram illustrating a linear transformation. On the left, a green matrix labeled  $\mathbf{X}$  is shown with four columns and two rows. In the middle, a multiplication symbol ( $\times$ ) is followed by a purple weight matrix labeled  $\mathbf{W}^Q$ , which is also 4x4. To the right of the multiplication is an equals sign (=). On the far right, the resulting matrix labeled  $\mathbf{Q}$  is shown, which is 2x2 and colored purple.

$$\mathbf{X} \times \mathbf{W}^K = \mathbf{K}$$

A diagram illustrating a linear transformation. On the left, a green matrix labeled  $\mathbf{X}$  is shown with four columns and two rows. In the middle, a multiplication symbol ( $\times$ ) is followed by an orange weight matrix labeled  $\mathbf{W}^K$ , which is 4x4. To the right of the multiplication is an equals sign (=). On the far right, the resulting matrix labeled  $\mathbf{K}$  is shown, which is 2x2 and colored orange.

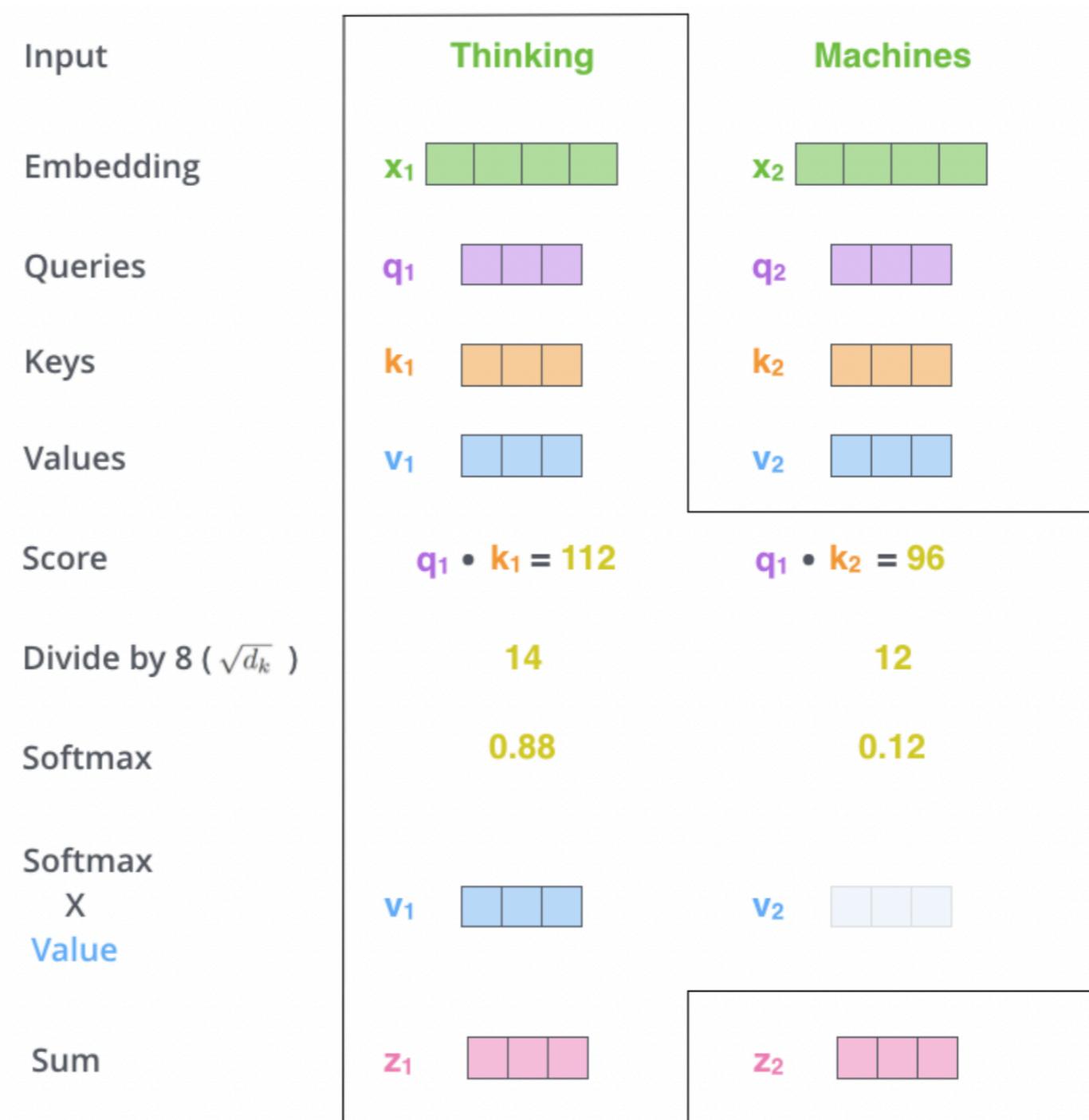
$$\mathbf{X} \times \mathbf{W}^V = \mathbf{V}$$

A diagram illustrating a linear transformation. On the left, a green matrix labeled  $\mathbf{X}$  is shown with four columns and two rows. In the middle, a multiplication symbol ( $\times$ ) is followed by a blue weight matrix labeled  $\mathbf{W}^V$ , which is 4x4. To the right of the multiplication is an equals sign (=). On the far right, the resulting matrix labeled  $\mathbf{V}$  is shown, which is 2x2 and colored blue.

Credit: Illustrated Transformer - <https://jalammar.github.io/illustrated-transformer/>

# Transformers

- **Transformers**



Credit: Illustrated Transformer - <https://jalammar.github.io/illustrated-transformer/>

# Transformers

- **Transformers**

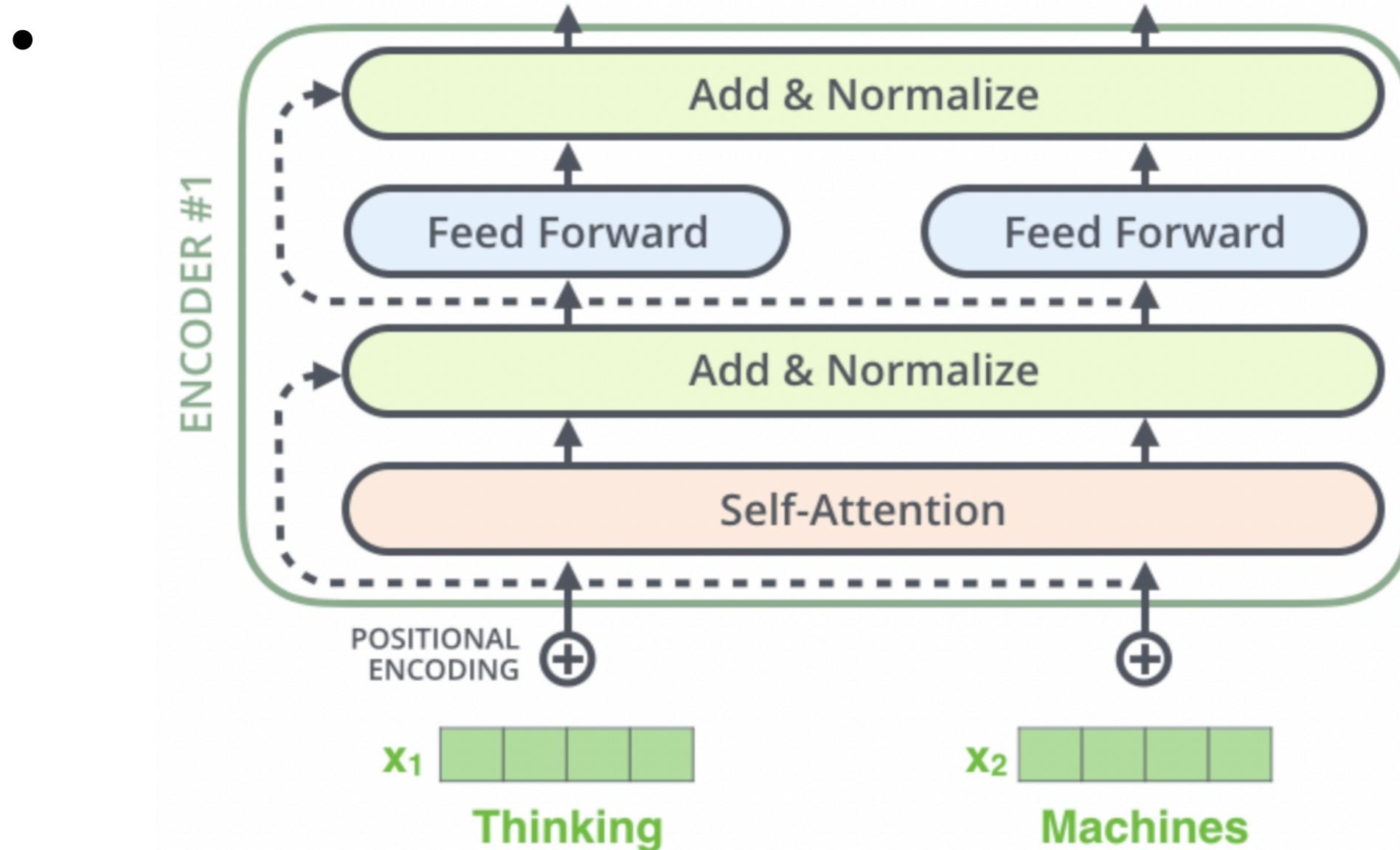
- 

$$\text{softmax} \left( \frac{\begin{array}{c} \text{Q} \quad \text{K}^T \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \quad \begin{array}{|c|c|} \hline & \\ \hline & \\ \hline & \\ \hline \end{array} \\ \times \end{array}}{\sqrt{d_k}} \right) \text{V}$$
$$= \begin{array}{c} \text{Z} \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \end{array}$$

Credit: Illustrated Transformer - <https://jalammar.github.io/illustrated-transformer/>

# Transformers

- **Transformers**



Credit: Illustrated Transformer - <https://jalammar.github.io/illustrated-transformer/>

# Deep Learning NLP

## Language Modeling

- “*I think I am, actually humble. I think I'm much more humble than you would understand.*”
- Cuan probable es esta oración?:

$$p(x_1, x_2, \dots, x_T) = ?$$

$$p(x_1, x_2, \dots, x_T) = \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1})$$

$$\sim \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-k})$$

**Supuesto  
Markoviano**

# Deep Learning NLP

## Language Modeling

- Una forma de resolver esto es contando el número de veces que la palabra viene luego de la oración:

$$p(x_t | x_{t-k}, \dots, x_{t-1}) = \frac{\text{count}(x_{t-k}, \dots, x_{t-1}, x_t)}{\text{count}(x_{t-k}, \dots, x_{t-1})}$$

$$k = 1$$

$$p(x_t | x_{t-1}, \dots, x_1) = \frac{\text{count}(x_{t-1}, x_t)}{\text{count}(x_{t-1})}$$

# Deep Learning NLP

## Language Modeling

- Una forma de resolver esto es contando el número de veces que la palabra viene luego de la oración:

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

< s > I am Sam < /s >  
< s > Sam I am < /s >  
< s > I do not like green eggs and ham < /s >

$$P(I | < s >) = \frac{2}{3} = .67$$

$$P(< /s > | Sam) = \frac{1}{2} = 0.5$$

$$P(Sam | < s >) = \frac{1}{3} = .33$$

$$P(Sam | am) = \frac{1}{2} = .5$$

$$P(am | I) = \frac{2}{3} = .67$$

$$P(do | I) = \frac{1}{3} = .33$$

# Deep Learning NLP

## Language Modeling

- **Problema:** Muy mala generalización.  
Probabilidades en el conjunto de entrenamiento deben ser muy similares a las del conjunto de test.
- Training set:  
I want two beers  
I want three beers
- Test set:  
I want four beers

$$p(\text{beer} \mid \text{want four}) = 0$$

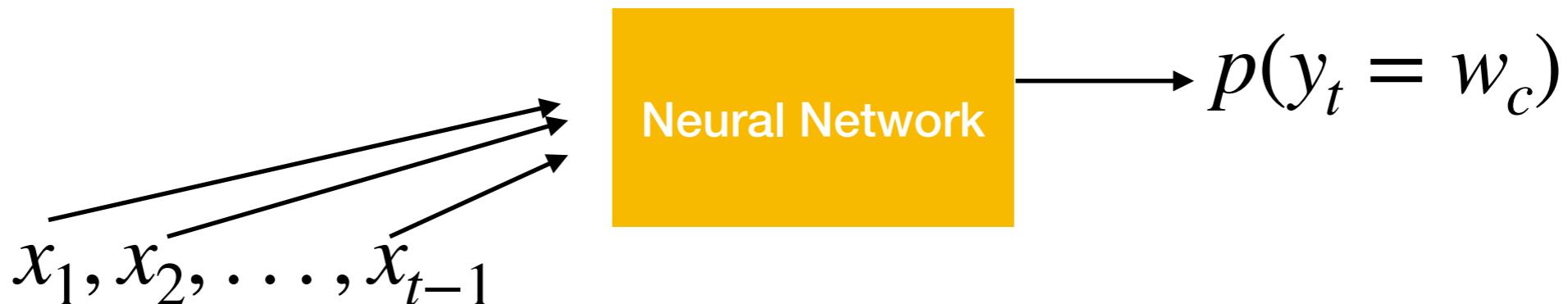
# Deep Learning NLP

## Language Modeling

- **Idea:** Reemplazar el conteo por una función que calcule las probabilidades (sí, una red neuronal)

$$p(x_t | x_{t-k}, \dots, x_{t-1}) = \frac{\text{count}(x_{t-k}, \dots, x_{t-1}, x_t)}{\text{count}(x_{t-k}, \dots, x_{t-1})}$$

$$p(x_t | x_{t-k}, \dots, x_{t-1}) = f_W(x_t, x_{t-1}, \dots, x_{t-k})$$



# Deep Learning NLP

**Language Modeling:** Dado un conjunto de palabras calcular la probabilidad de la próxima.

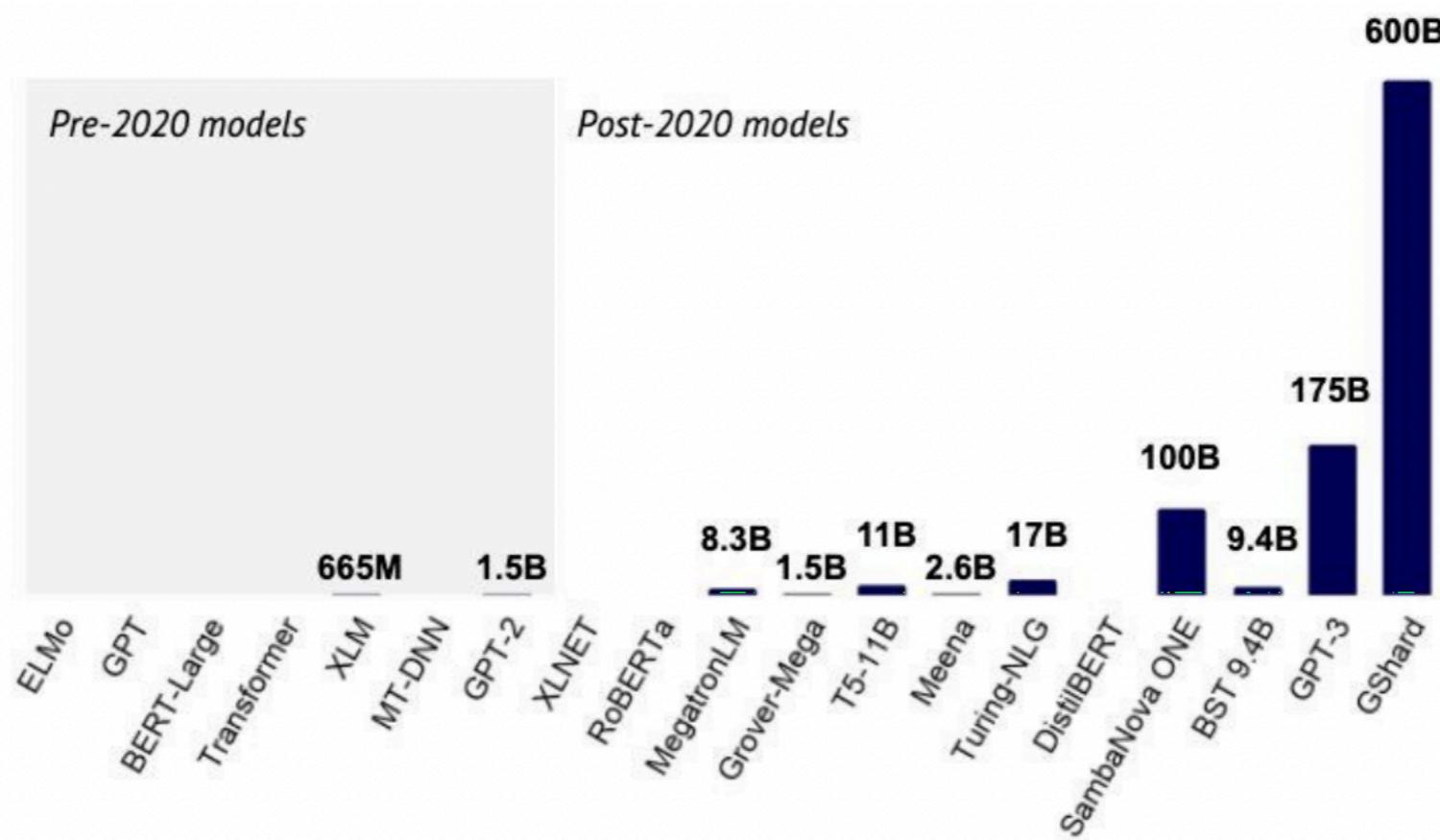
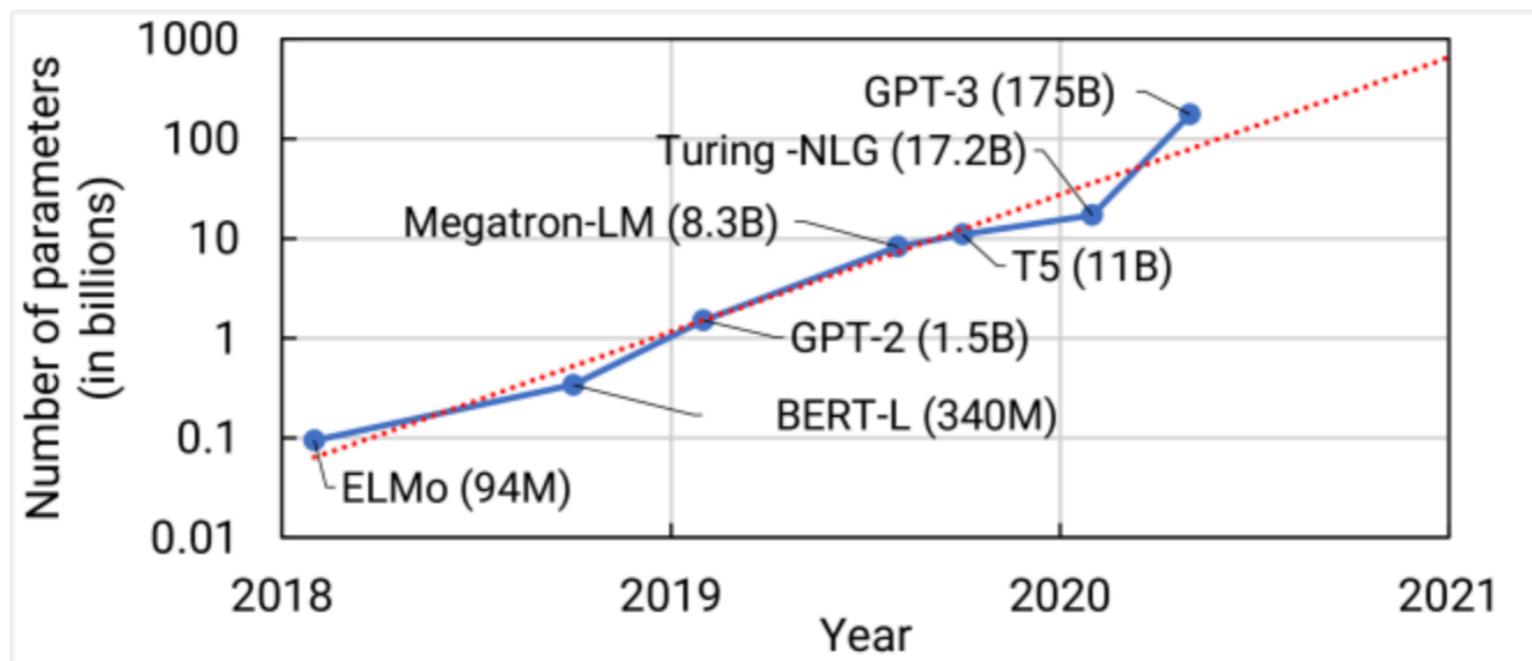
- Necesitamos representar oraciones de largo variable.
- Recurrent Neural Networks: **Read**, **Update**, **Predict**

$$h_t = \tanh(Wx_t + Uh_{t-1})$$

$$p(x_t^w = 1 | x_{<t}) = \text{softmax}(V_w^T h_t) \quad V \in R^{O \times |V|}$$

$$p(x_t^w = 1 | x_{<t}) = \exp(V_w^T h_{t-1}) / \sum_i \exp(V_i^T h_{t-1})$$

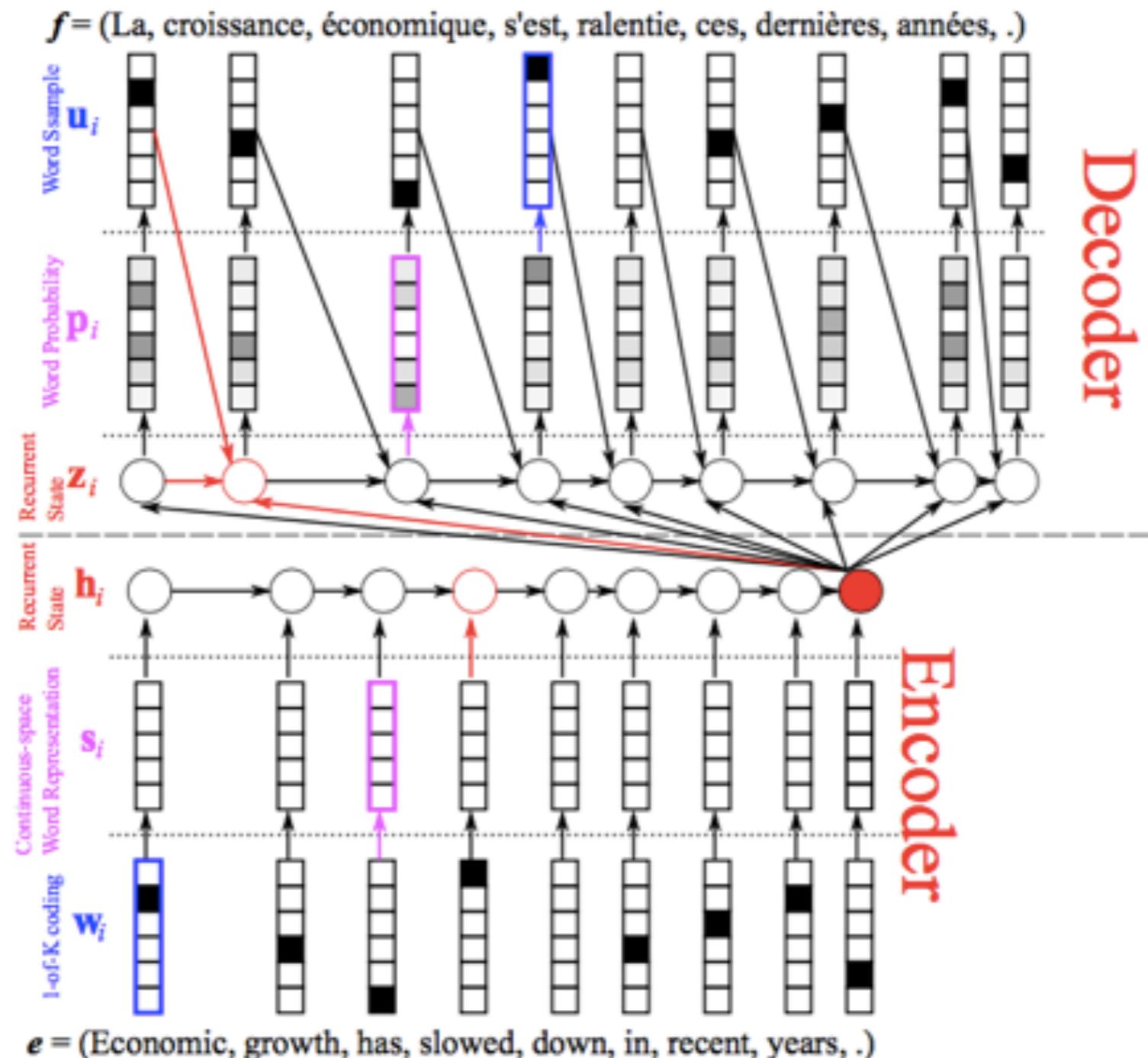
# Deep Learning NLP



# Redes Neuronales Recurrentes

**Neural Machine Translation:** Traducir una entrada en lenguaje X a lenguaje Y.

**Encoder-Decoder Architecture:** **Codifica** la oración en el lenguaje original en un vector usando una RNN y **Decodifica** el vector codificado en la traducción usando una RNN.



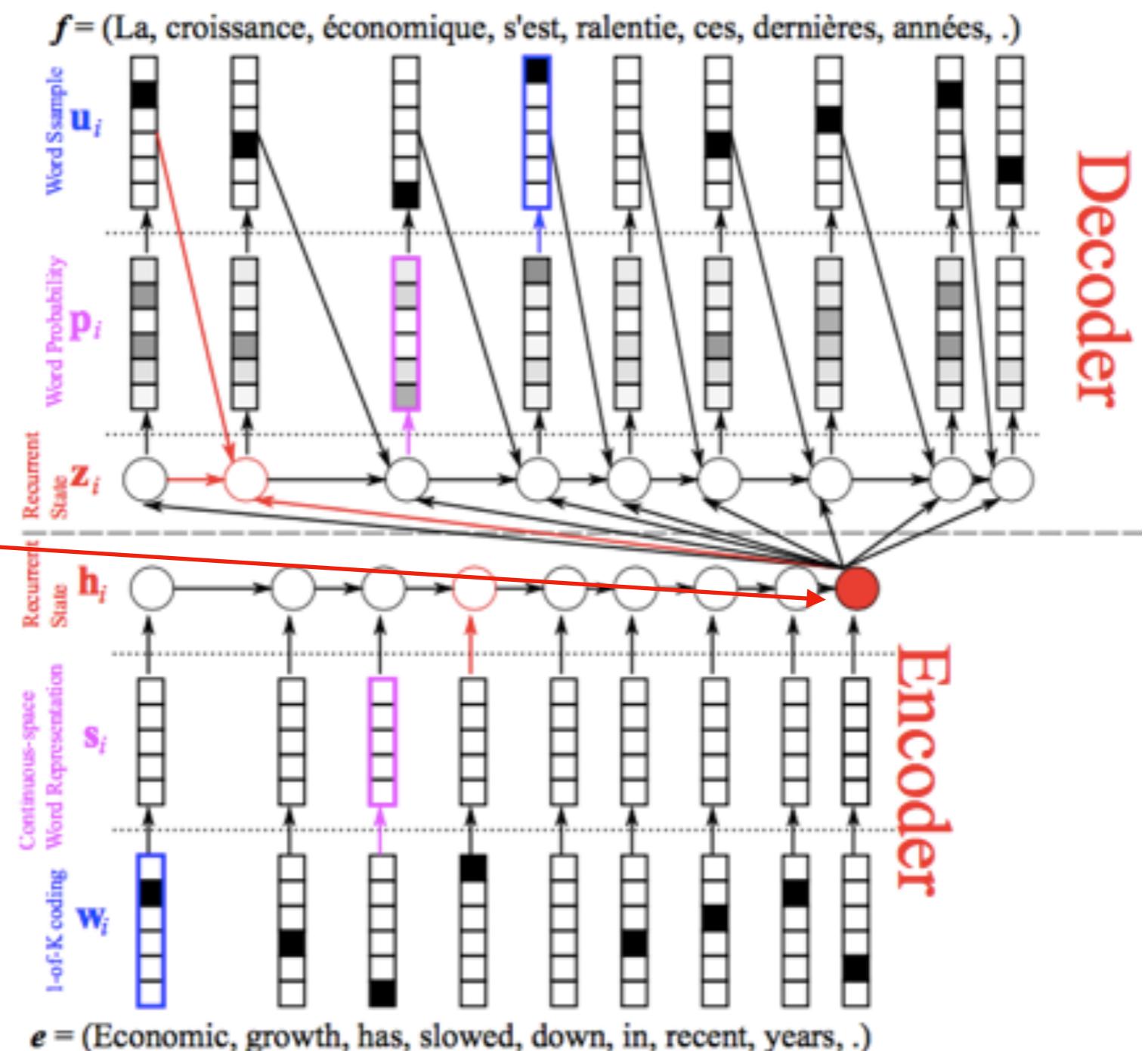
Credit: Kyunghyun Cho et al. 2014

# Redes Neuronales Recurrentes

## Neural Machine Translation

Sin embargo estamos reduciendo el significado de toda la oración en un **vector de largo fijo** (mala idea).

En vez, podemos enfocarnos en partes importantes de la entrada, cuando traducimos cada palabra.



Credit: Kyunghyun Cho et al. 2014

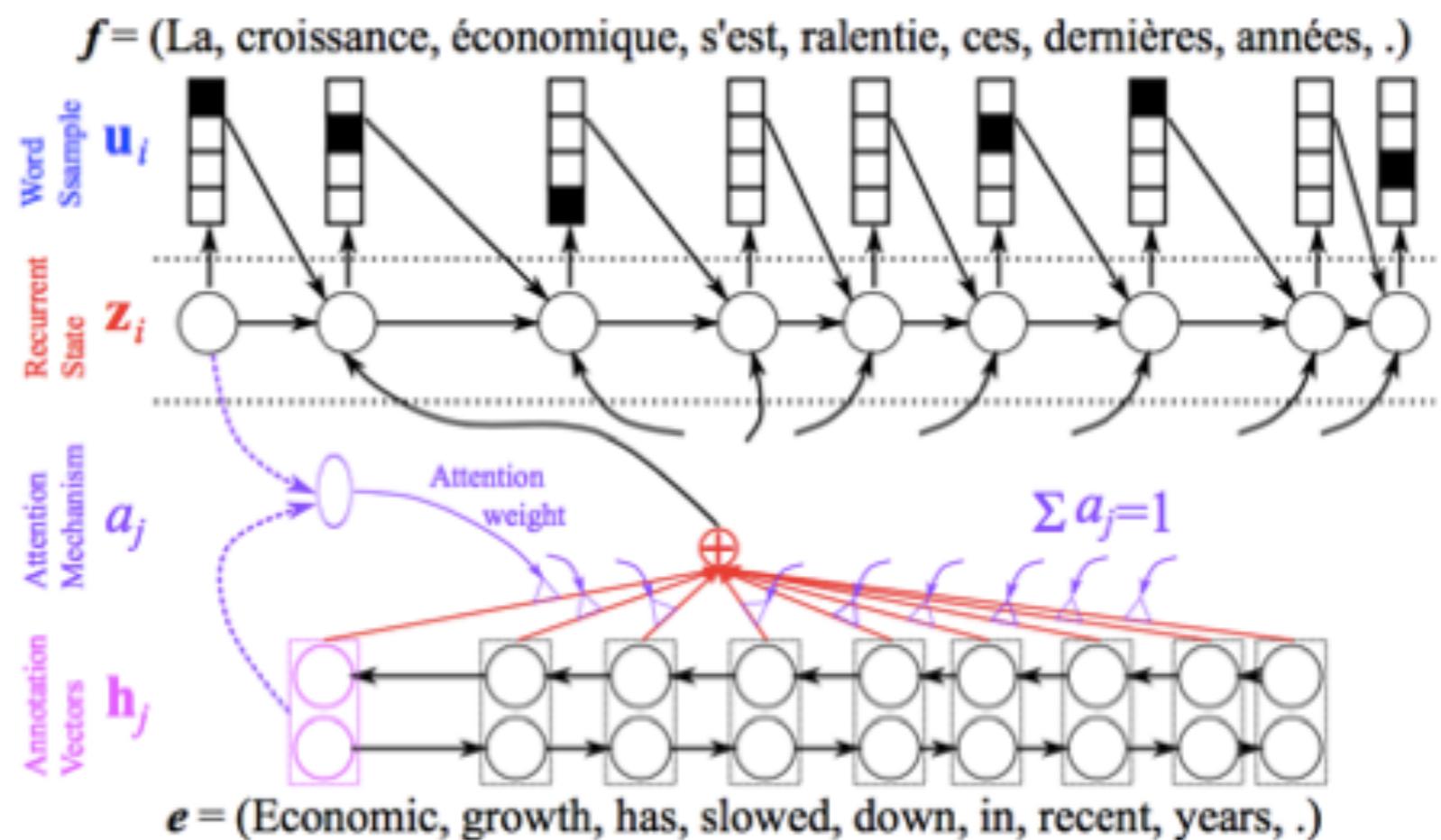
# Redes Neuronales Recurrentes

## Neural Machine Translation

**Attention based-decoder:** En cada paso calcular un peso de cada palabra y usar una suma con pesos para calcular el vector.

$$\alpha_{i,j} = \frac{\exp(f_W(z_{i-1}, h_j))}{\sum_k f_W(z_{i-1}, h_k)}$$

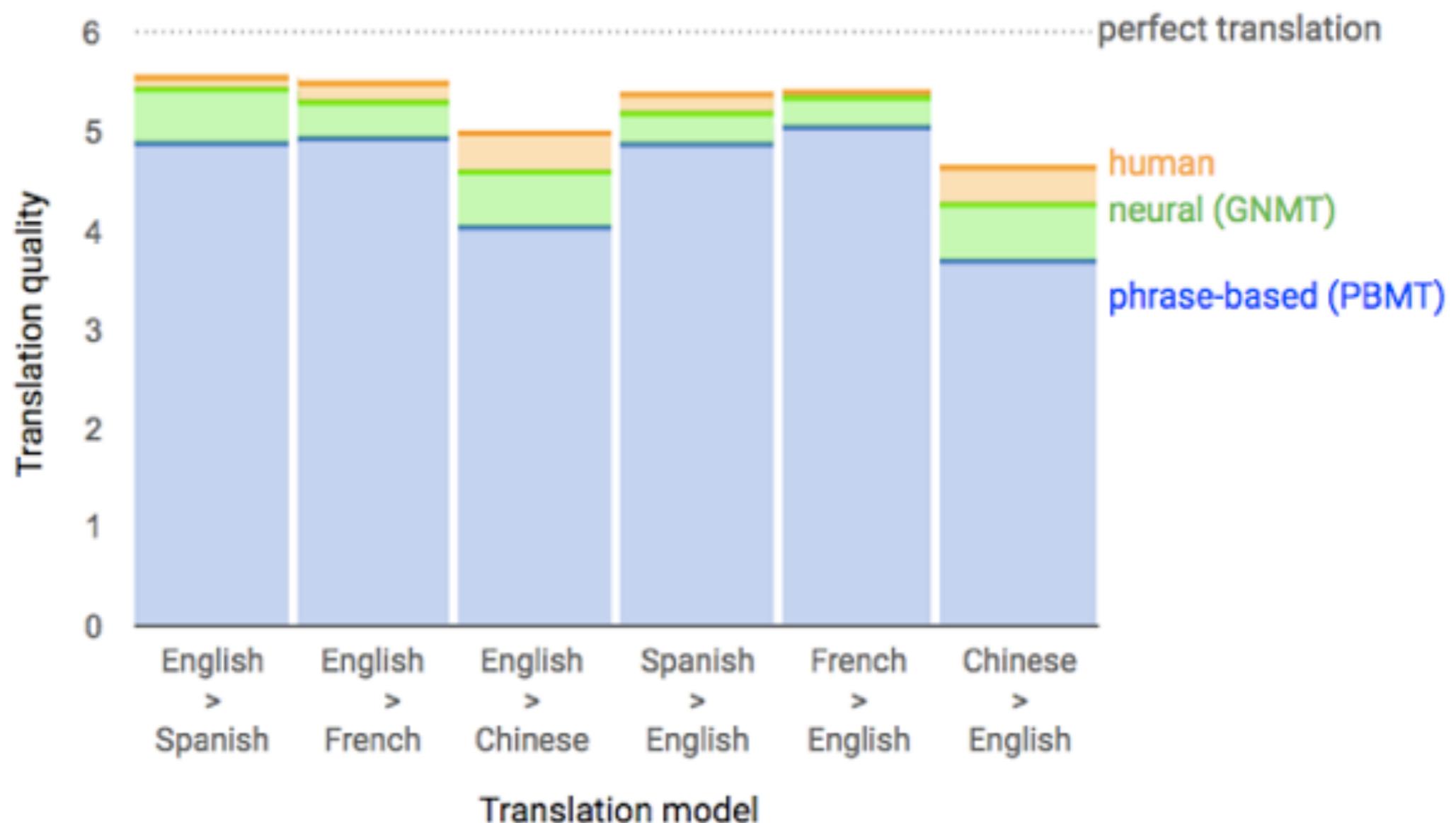
$$c_i = \sum_k \alpha_{i,k} h_k$$



Credit: Kyunghyun Cho et al. 2014

# Deep Learning for NLP

## Neural Machine Translation



# Machine Learning en la Práctica

## El tradeoff sesgo-varianza

- Considere el error cuadrático:

$$\bar{y} = E[\hat{y}]$$

$$\begin{aligned}E[(\hat{y} - y)^2] &= E[((\hat{y} - \bar{y}) + (\bar{y} - y))^2] \\&= E[(\hat{y} - \bar{y})^2] + 2E(\hat{y} - \bar{y})(\bar{y} - y) + (\bar{y} - y)^2 \\&= E[(\hat{y} - \bar{y})^2] + (\bar{y} - y)^2 \\&= Var[\hat{y}] + Bias^2(\hat{y})\end{aligned}$$

# Machine Learning en la Práctica

## El tradeoff sesgo-varianza

- Considere el error cuadrático:

$$\bar{y} = E[\hat{y}]$$

$$\begin{aligned}E[(\hat{y} - y)^2] &= E[((\hat{y} - \bar{y}) + (\bar{y} - y))^2] \\&= E[(\hat{y} - \bar{y})^2] + 2E(\hat{y} - \bar{y})(\bar{y} - y) + (\bar{y} - y)^2 \\&= E[(\hat{y} - \bar{y})^2] + (\bar{y} - y)^2 \\&= Var[\hat{y}] + Bias^2(\hat{y})\end{aligned}$$

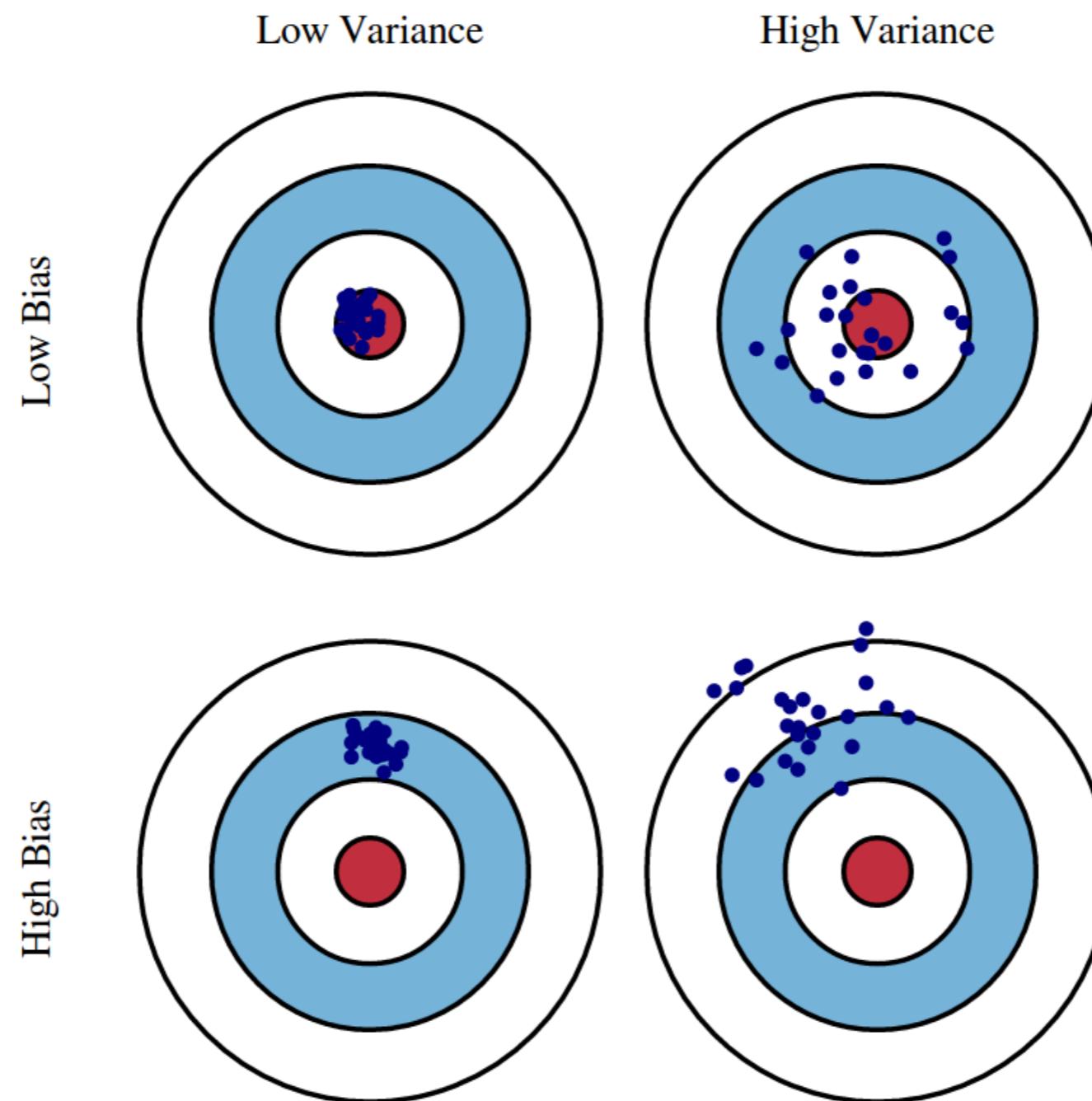
Tener estimadores sesgados con baja varianza es de utilidad.

Por ejemplo: La regresión ridge es sesgado pero tiene menor varianza.

# Machine Learning en la Práctica

## El tradeoff sesgo-varianza

- 



# Machine Learning en la Práctica

## Proceso de entrenamiento - testing

Entrenamiento ~0.7

Test ~0.3



- **Entrenamiento:** Donde elegir los parámetros (optimizar).  
**Test:** Donde medir la performance del modelo.

# Machine Learning en la Práctica

## Proceso de entrenamiento - testing

Entrenamiento ~0.7

Test ~0.3

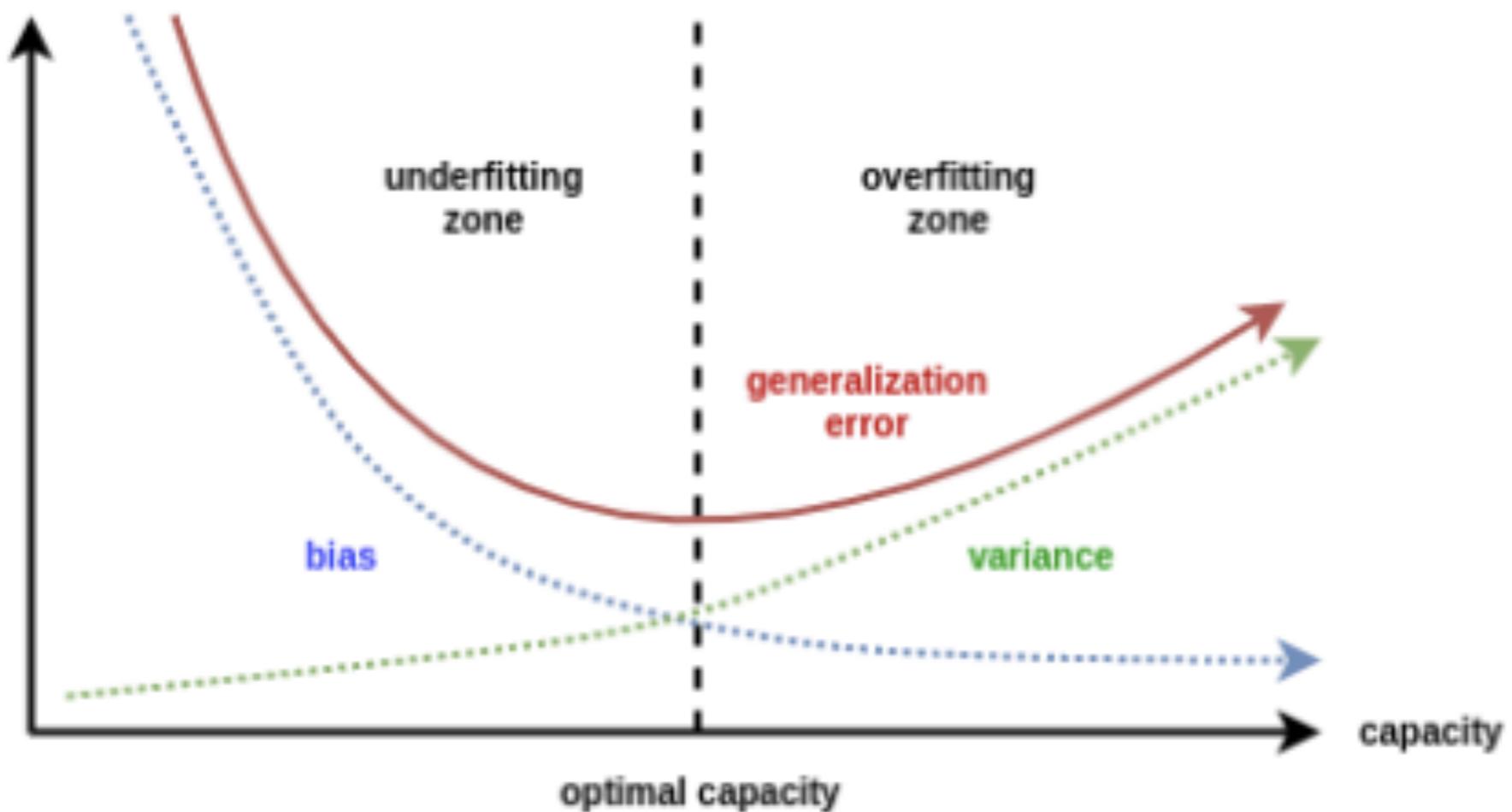


- **Entrenamiento:** Donde elegir los parámetros (optimizar).
- Test:** Donde medir la performance del modelo.
- Validación:** Donde elegir los **hiperparámetros**.

Entrenamiento ~0.7   Valid~0.1   Test ~0.2



# Machine Learning en la Práctica



# Machine Learning en la Práctica

Deep Learning es un caso diferente, aún se está comprendiendo este fenómeno

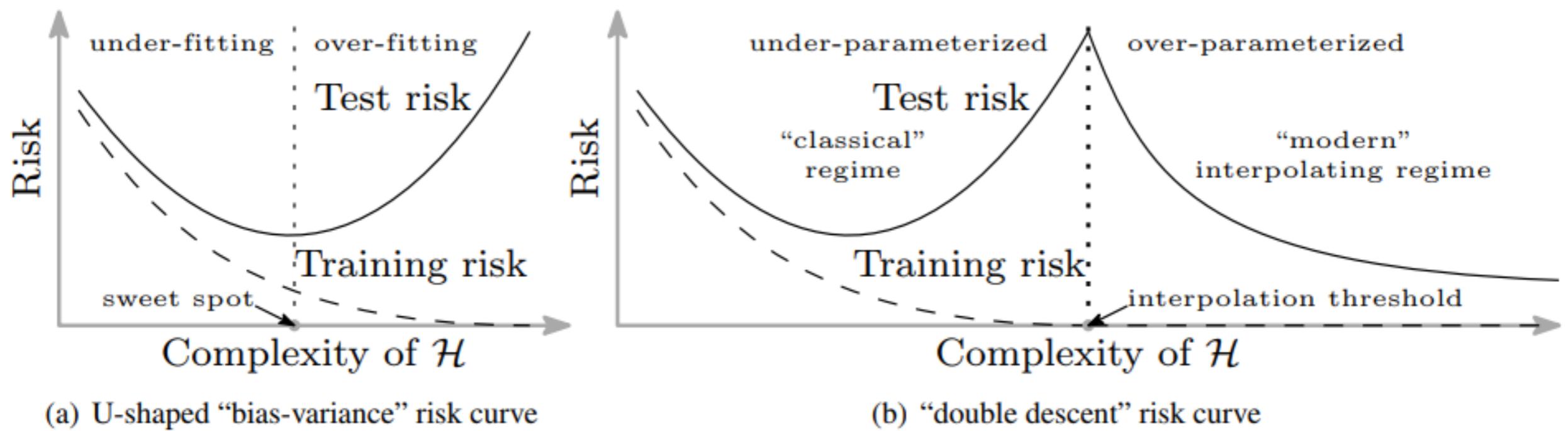


Figure 1: Curves for training risk (dashed line) and test risk (solid line). (a) The classical *U-shaped risk curve* arising from the bias-variance trade-off. (b) The *double descent risk curve*, which incorporates the U-shaped risk curve (i.e., the “classical” regime) together with the observed behavior from using high complexity function classes (i.e., the “modern” interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

Imagen de Belkin et al. 2018

# Machine Learning en la Práctica

## Cross Validation:

Sea todo el conjunto de datos de entrenamiento D. Se divide D en CV submuestras.

For k in CV

- Entrenar en  $D_{-k}$  evaluar en  $D_k$

Elegir mejores hiperparámetros de repetir proceso anterior.

Entrenar en todo D

Evaluar en el conjunto de test.



# Machine Learning en la Práctica

## Evaluación:

No necesariamente siempre evaluamos con la función de pérdida.

		True condition			
		Condition positive	Condition negative	Prevalence = $\frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
Predicted condition	Predicted condition positive	True positive	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
	Predicted condition negative	False negative, Type II error	True negative	False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR-}}$
		False negative rate (FNR), Miss rate $= \frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Negative likelihood ratio (LR-) = $\frac{\text{FNR}}{\text{TNR}}$	