Redes Neuronales para Lenguaje Natural

2023

Grupo de Procesamiento de Lenguaje Natural
Instituto de Computación

Redes Neuronales Completamente Conectadas

- Regresión Logística
- **Neurona Artificial, Redes Multicapa, Backpropagation**
- **Aplicaciones, Clasificación, Modelo de lenguaje**
- Implementación

Jurafsky and Martin 3rd edition. Cap 7. https://web.stanford.edu/~jurafsky/slp3/

Redes Neuronales Completamente Conectadas (Perceptrón Multicapa)

Redes Neuronales (Perceptrón Multicapa)

Neurona Artificial

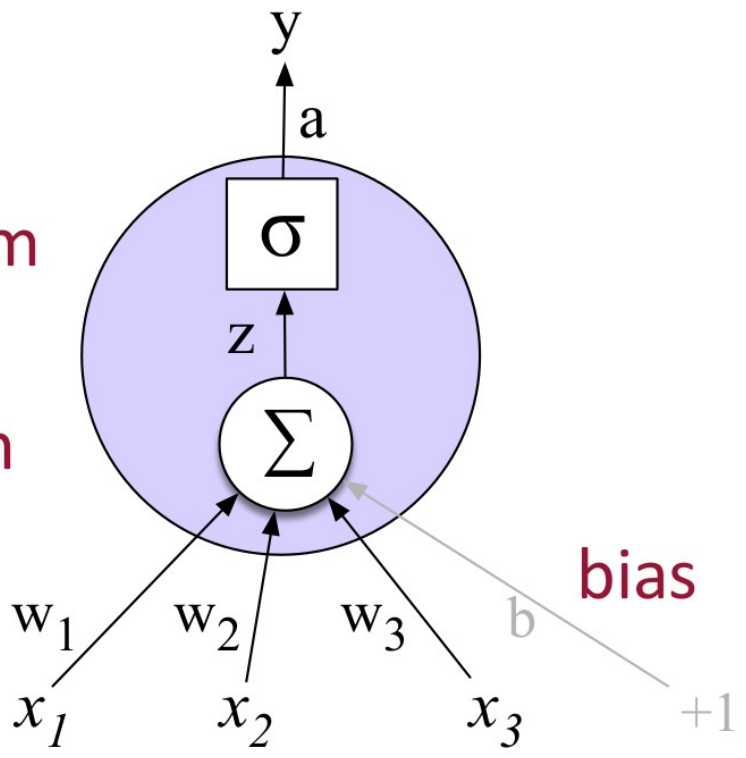
Output value

Non-linear transform

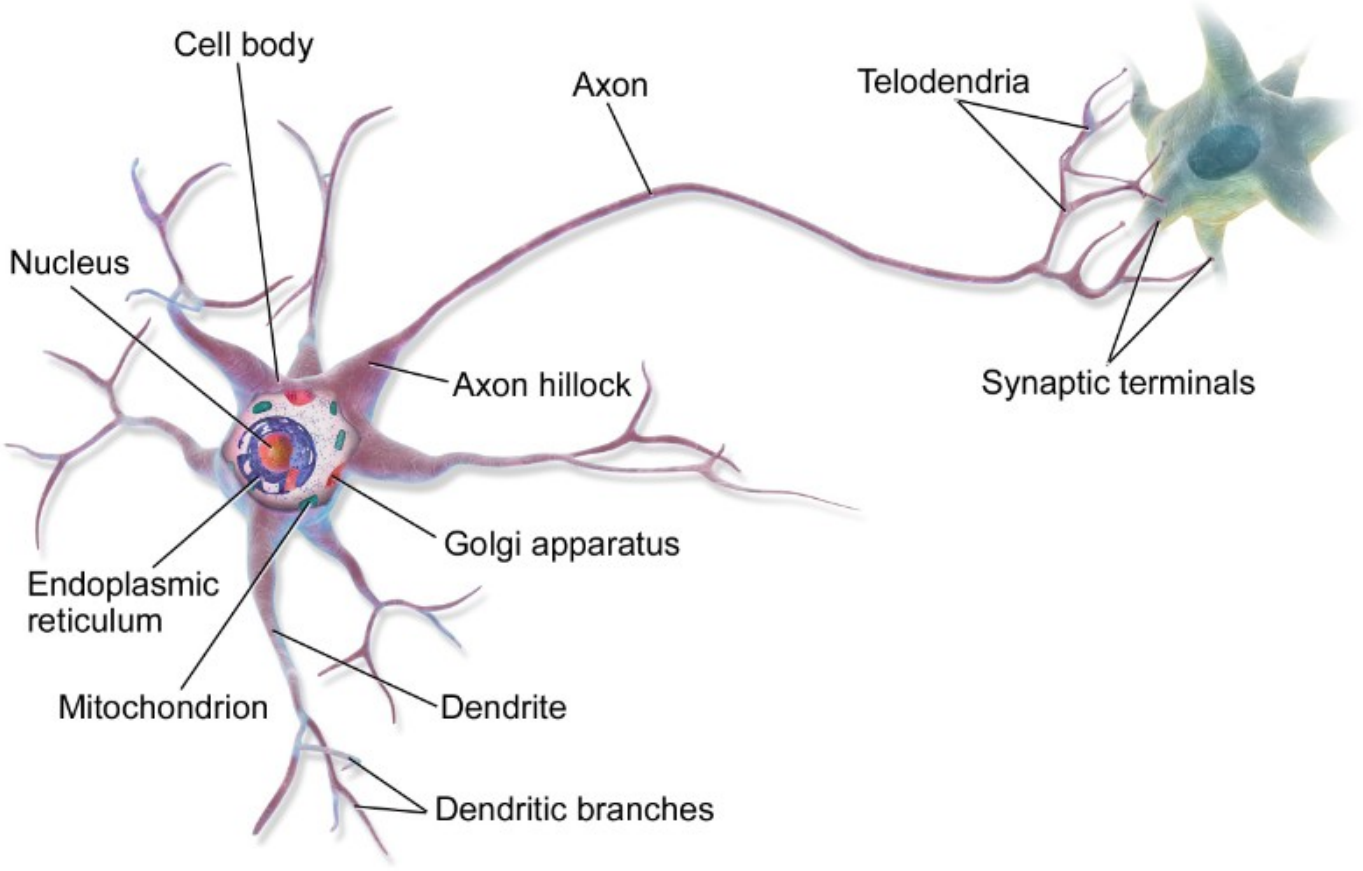
Weighted sum

Weights

Input layer



Inspiración biológica



Neurona Artificial

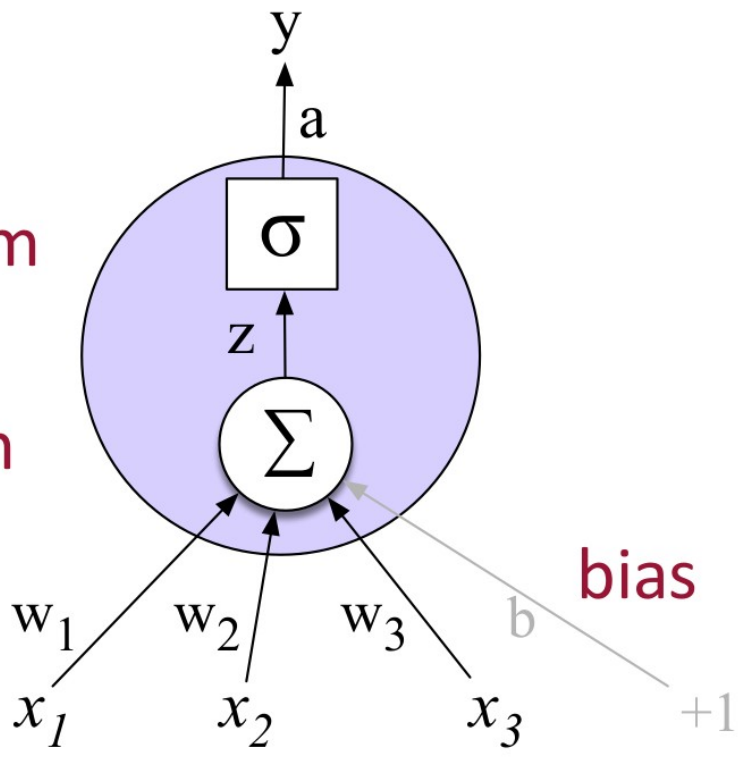
Output value

Non-linear transform

Weighted sum

Weights

Input layer



Neurona Artificial

$$z = b + \sum_i w_i x_i$$

$$z = w \cdot x + b$$

$$y = a = f(z)$$

Neurona Artificial

$$z = b + \sum_i w_i x_i$$

$$z = w \cdot x + b$$

$$y = a = f(z)$$

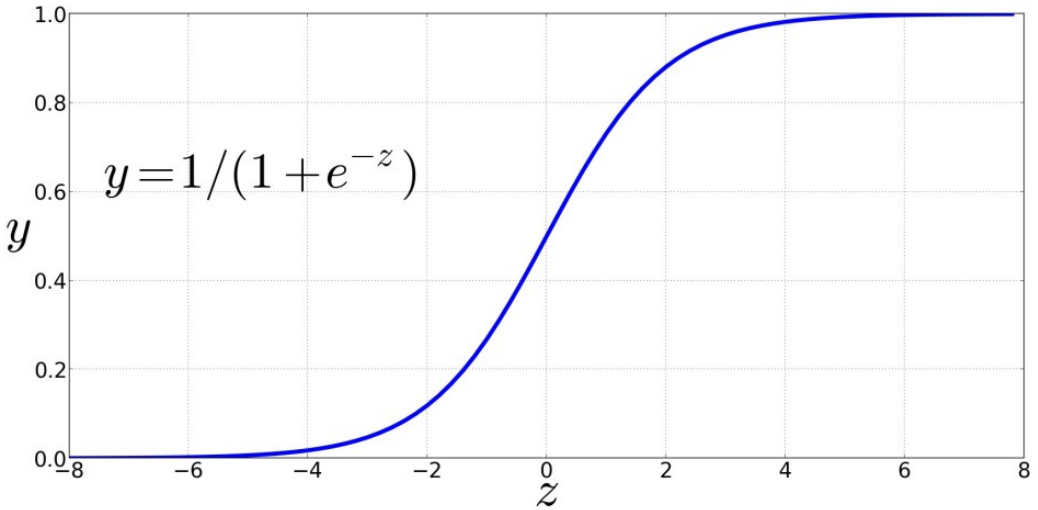
- Recibe un vector como entrada
- Devuelve un escalar
- A f le llamamos **función de activación**

  (función “escalón”, ej. Sigmoid)
- Los pesos **w** y **b** se aprenden

Función de Activación

Sigmoid

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

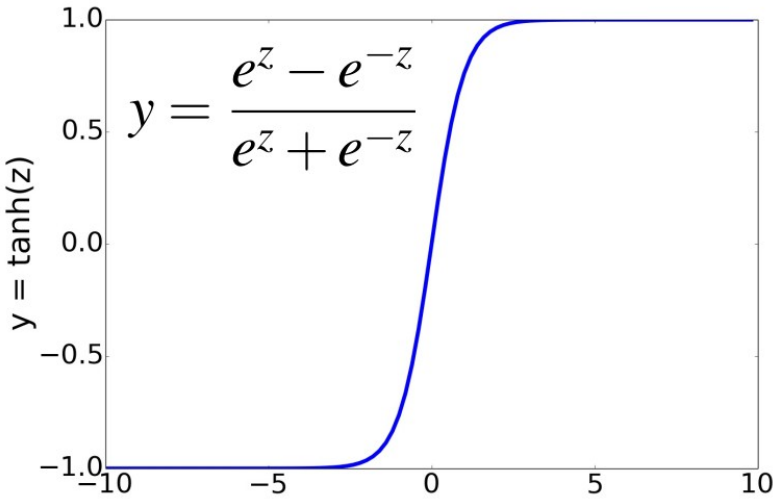


La función logística (sigmoid) como función de activación:

$$y = \sigma(w \cdot x + b) = \frac{1}{1+\exp(-(w \cdot x + b))}$$

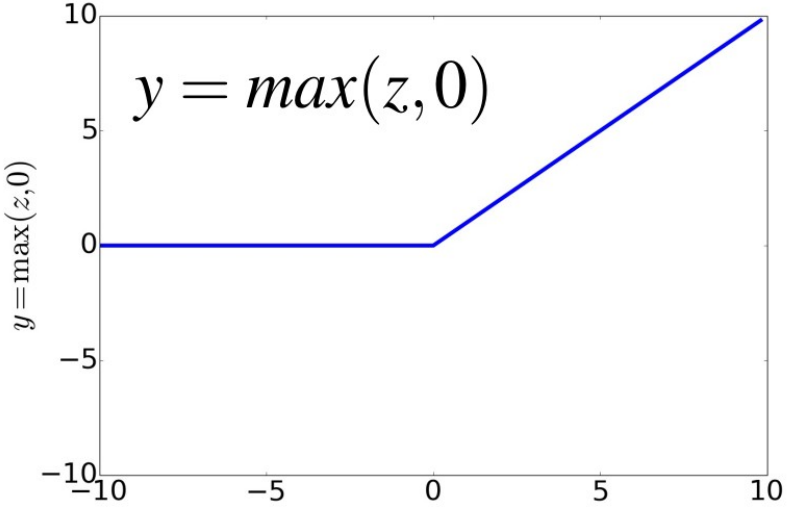
Otras funciones de Activación

Non-Linear Activation Functions besides sigmoid



tanh

Most Common:



ReLU
Rectified Linear Unit

Basado en https://web.stanford.edu/~jurafsky/slp3/

Neurona Artificial (Resumen)

- Función que **recibe un vector** y **devuelve un escalar**
- **Secuencia de neuronas** para **salidas vectoriales**
  - **softmax** para problemas multiclase (regresión multinomial)
- Entrenamos con **descenso por gradiente**
  - de a **mini-batches**

The XOR problem (Minsky and Papert, 1969)

Logic Functions as Artificial Neurons

Minsky and Papert (1969)

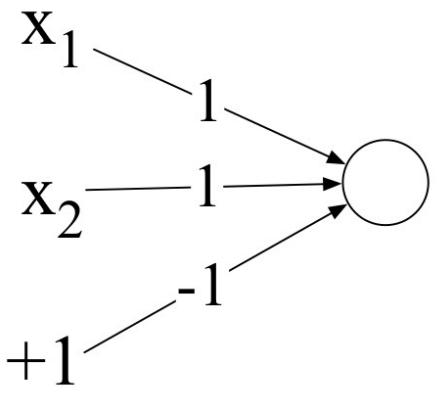
Can neural units compute simple functions of input?

AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

AND y OR como una neurona

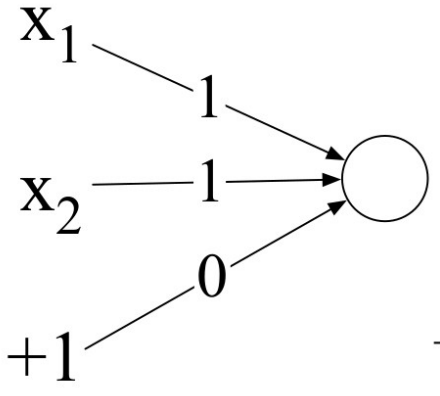
Perceptron

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$



AND

AND		
x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1



OR

OR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

¿Como construimos el XOR?

XOR Problem

Perceptron equation given x_1 and x_2, is the equation of a line

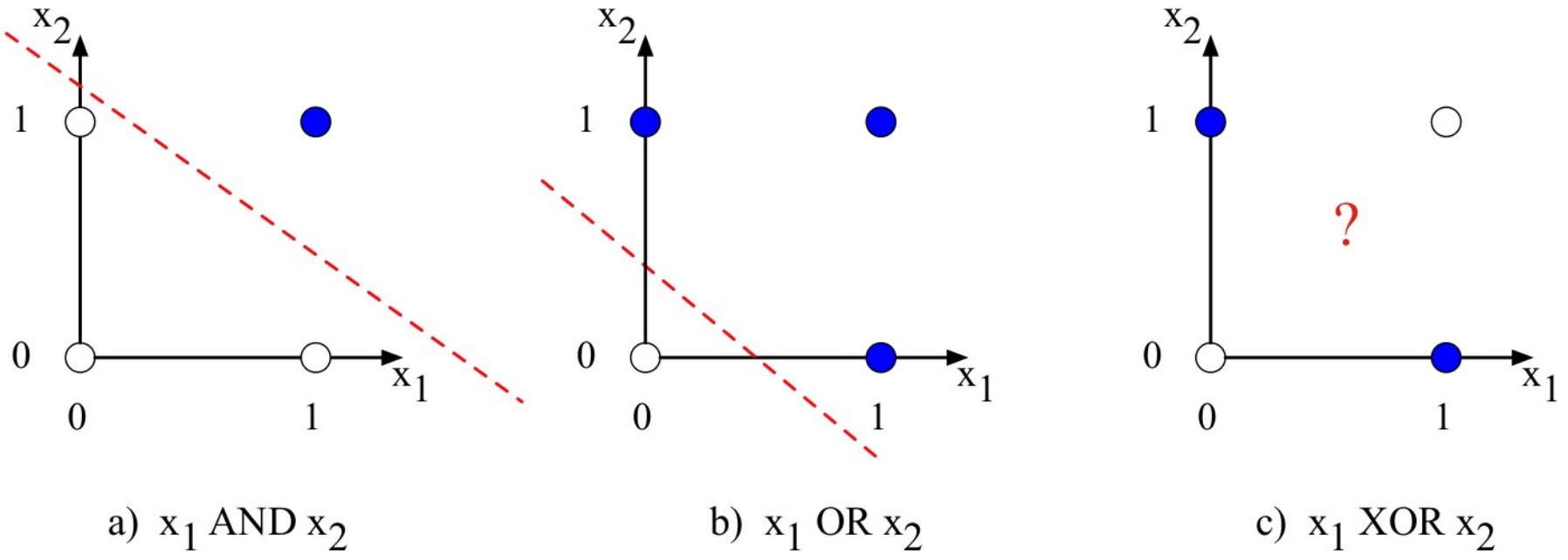
$$w_1 x_1 + w_2 x_2 + b = 0$$

(in standard linear format: $x_2 = (-w_1/w_2)x_1 + (-b/w_2)$)

This line acts as a **decision boundary**

- 0 if input is on one side of the line
- 1 if on the other side of the line

XOR Problem



a) x_1 AND x_2

b) x_1 OR x_2

c) x_1 XOR x_2

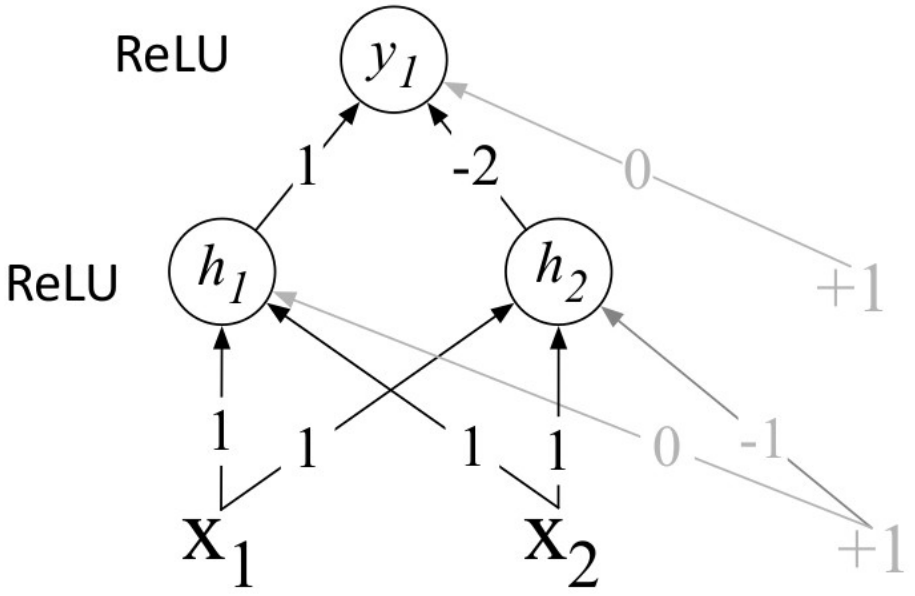
XOR is not a **linearly separable** function!

XOR Problem

XOR **can't** be calculated by a single perceptron

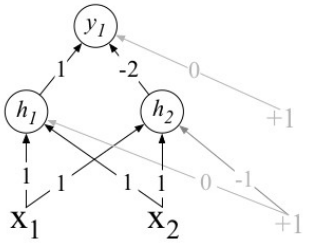
XOR **can** be calculated by a layered network of units.

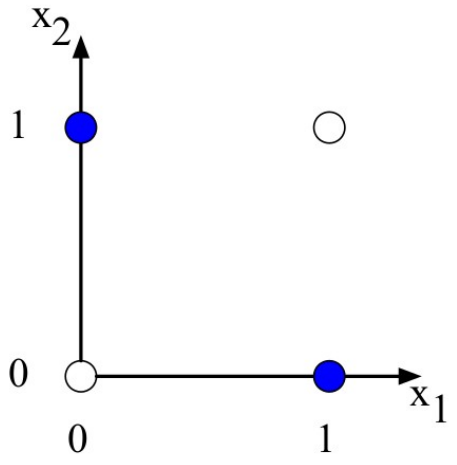
XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



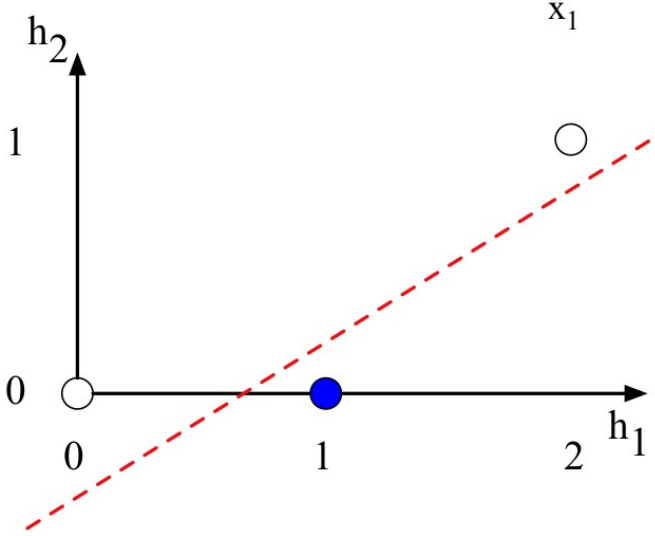
XOR Problem

The hidden representation h





a) The original *x* space



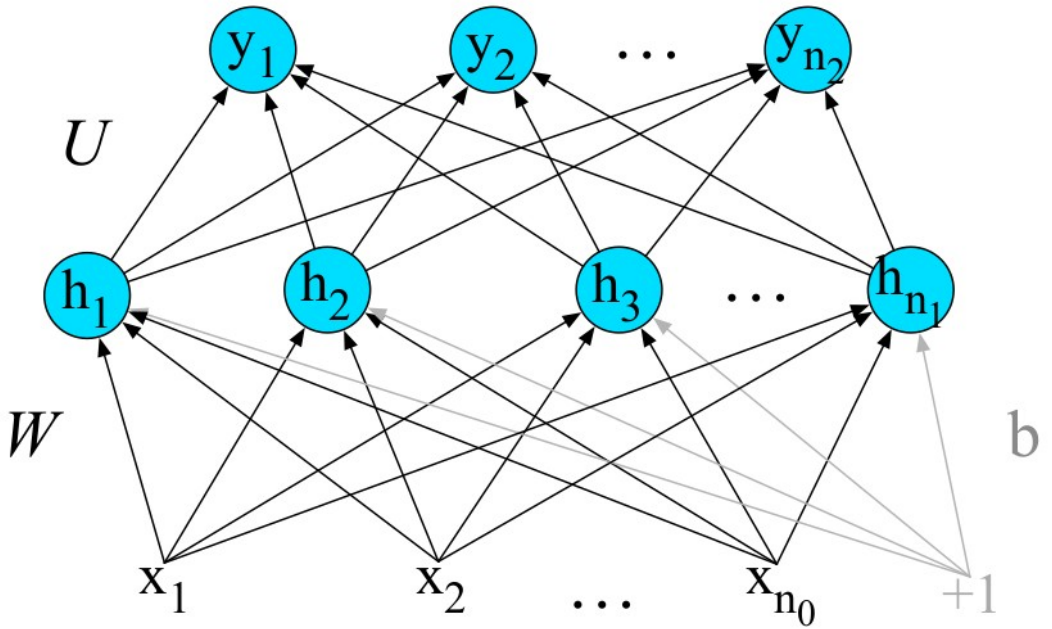
b) The new (linearly separable) *h* space

(With learning: hidden layers will learn to form useful representations)

Feedforward Fully-Connected Networks

Feedforward Network (Multilayer Perceptron)

Can also be called **multi-layer perceptrons** (or **MLPs**) for historical reasons

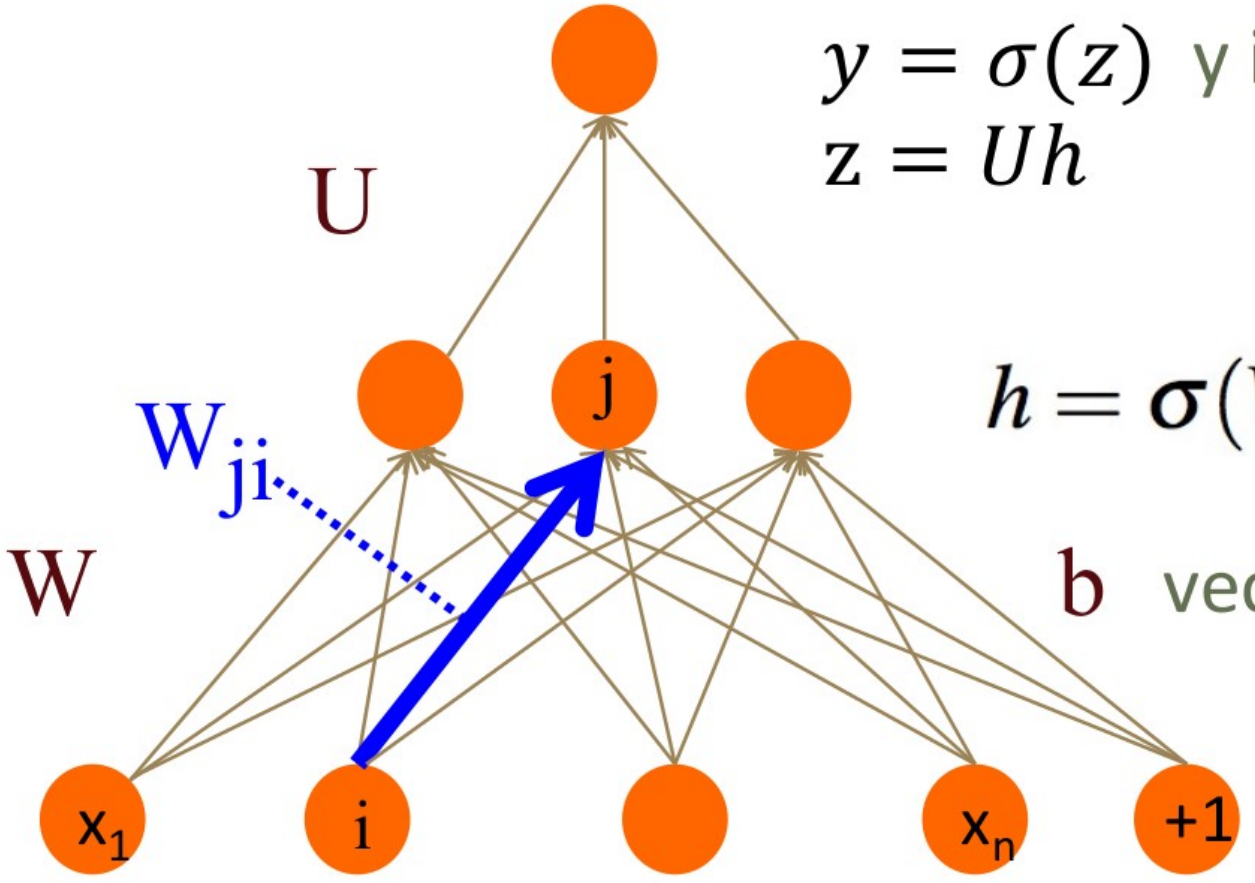


Two-layer Network Example (scalar output)

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



$y = \sigma(z)$ y is a scalar

$z = Uh$

$h = \sigma(Wx + b)$

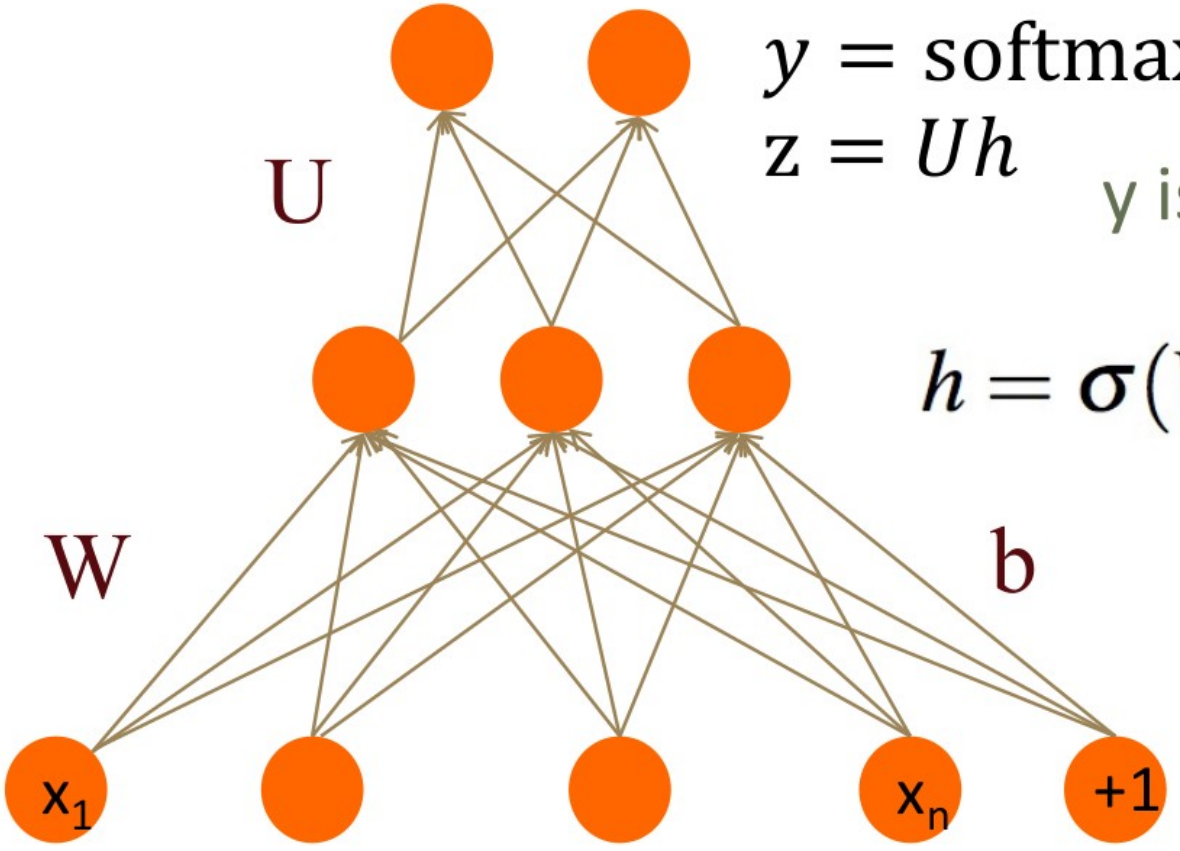
b vector

Two-layer Network Example (softmax)

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



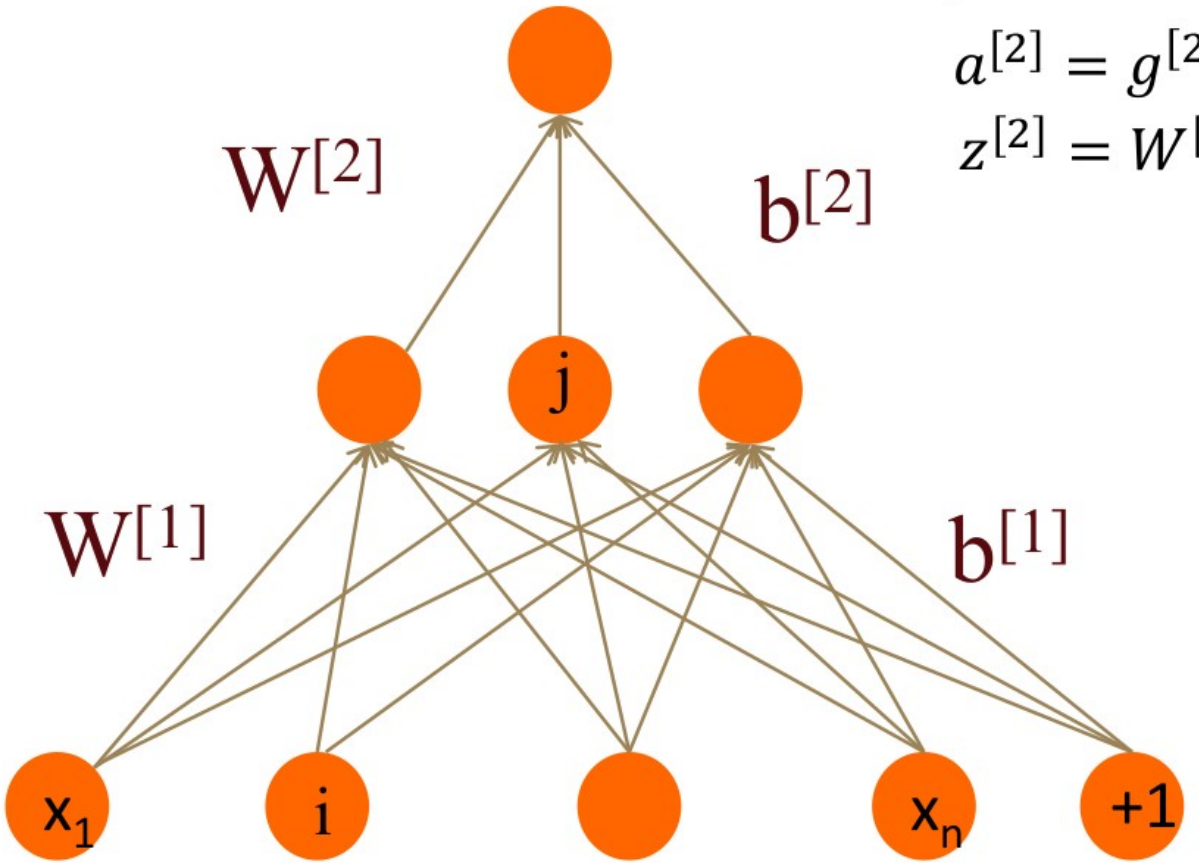
$$y = \text{softmax}(z)$$
$$z = Uh$$

y is a vector

$$h = \sigma(Wx + b)$$

Could be ReLU
Or tanh

Two-layer Network Example



$$y = a^{[2]}$$

$$a^{[2]} = g^{[2]}(z^{[2]})$$ sigmoid or softmax

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[1]} = g^{[1]}(z^{[1]})$$ ReLU

$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

Backpropagation (Network Training)

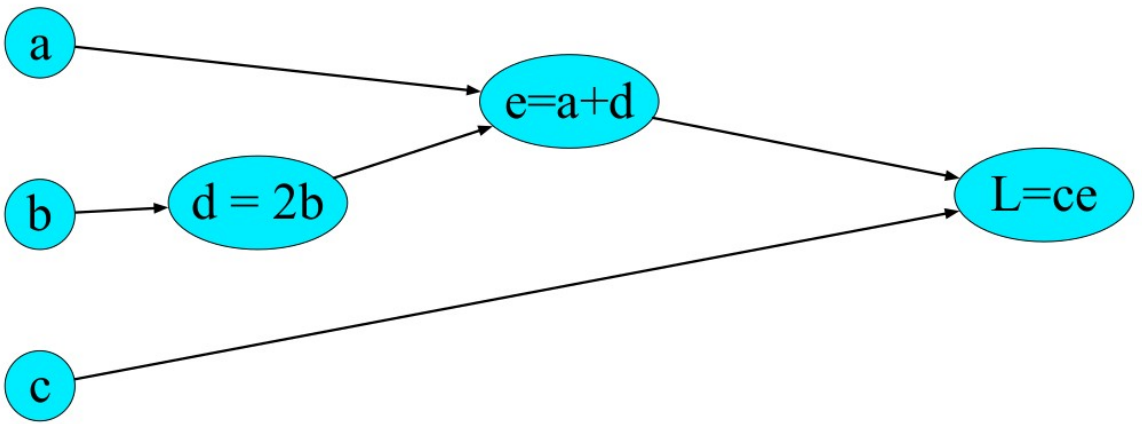
- Para **entrenar** necesitamos el **gradiente** de la **función de pérdida.** Las **capas apiladas** son **composiciones de funciones** (regla de la cadena)
- Dos pasadas para computar el gradiente
  - **Forward pass**: se ejecuta la red para una entrada
  - **Bakcward pass**: se propaga el gradiente desde la capa de salida hacia la capa de entrada

Grafo de computación

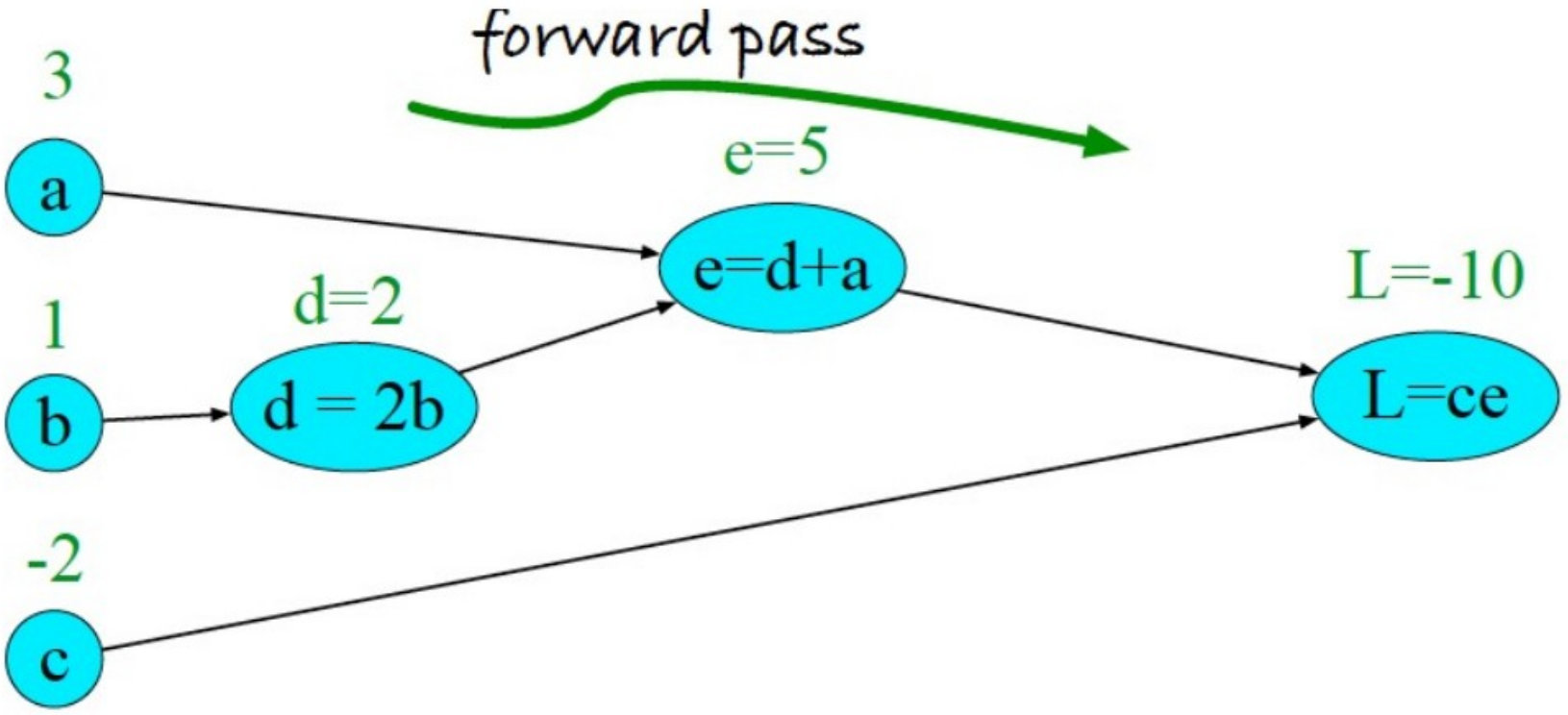
Example: $L(a,b,c) = c(a+2b)$

Computations:

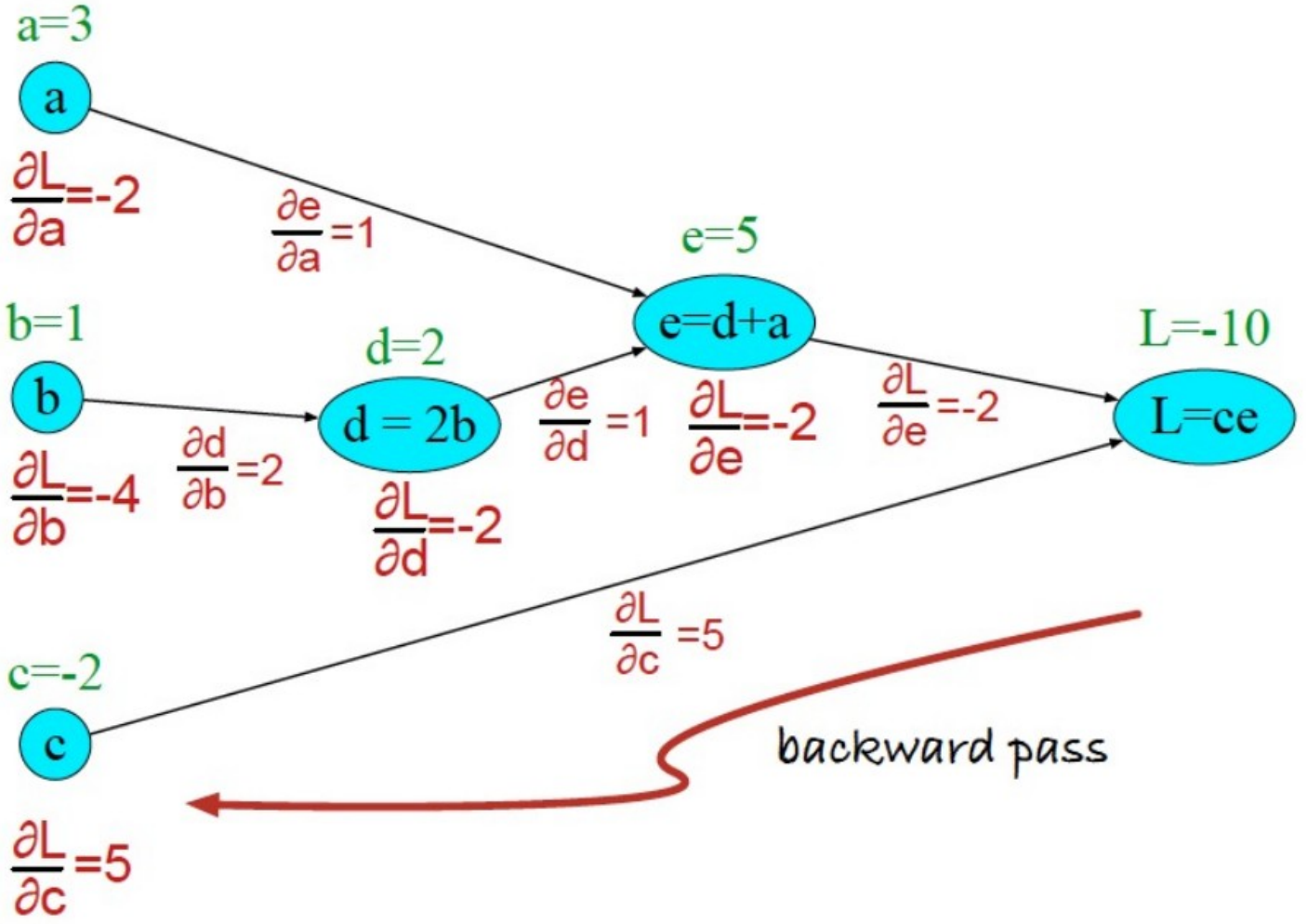
$$d = 2*b$$
$$e = a+d$$
$$L = c*e$$



Grafo de computación



Grafo de computación



Backpropagation

- El **grafo de computación** permite la **diferenciación automática**
- Las **operaciones tensoriales** se computan más rápido en hardware especifico (Ej. GPUs, TPUs)
- En la práctica se utilizan **variantes de SGD**
  - **AdaGrad (Adaptative Gradient Algorithm)**
  - **AdaDelta (AdaGrad extension)**
  - **RMSProp (Root Mean Square Propagation)**
  - **Adam (Adaptive Moment Estimation)**

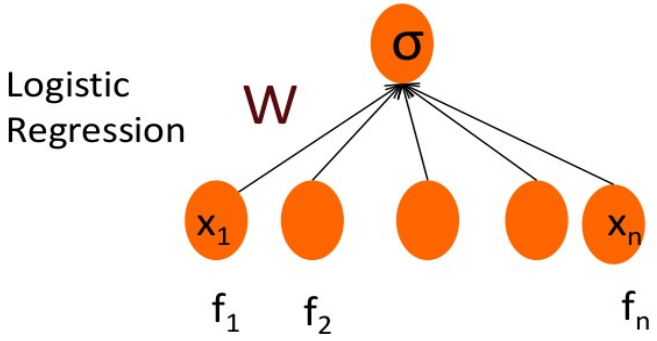
Red Completamente Conectada (Resumen)

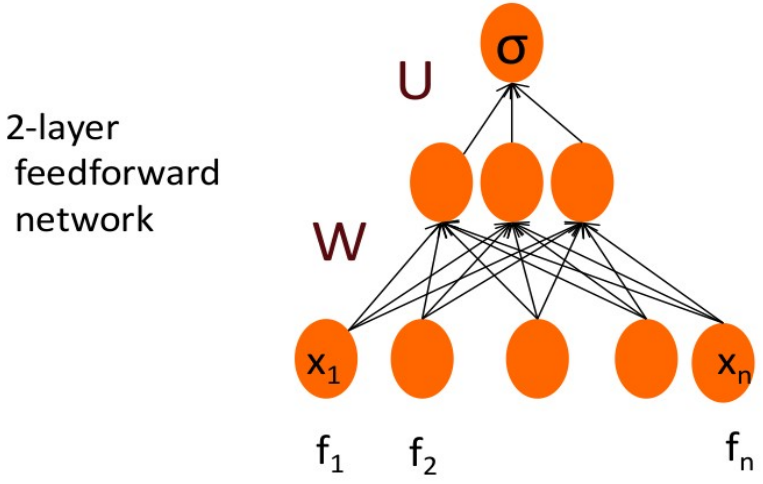
- N **capas de neuronas apiladas** (N es la profundidad de la red)
  - La red recibe como entrada un vector y devuelve un vector
    - Cada capa devuelve un vector (representaciones de la entrada)
  - Cada neurona recibe como entrada la salida de la capa anterior
  - Entrenamos con **backpropagation** por **mini-batches**

- **Varias capas** permite representaciones **linealmente separables** de entradas que originalmente no lo eran

- **¿Cómo la usamos?**

Como una regresión logística (o multinomial)

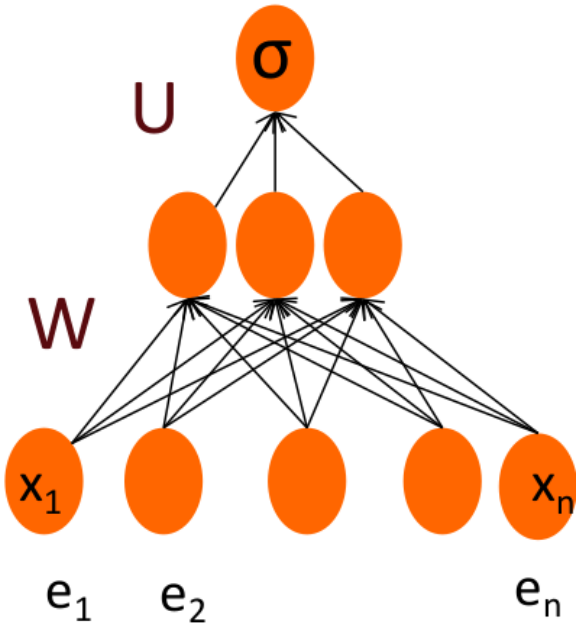




Var	Definition
x_1	count(positive lexicon) $\in$ doc)
x_2	count(negative lexicon) $\in$ doc)
x_3	$\begin{cases} 1 & \text{if } \text{“no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

Representation Learning

- Este tipo de modelos permite **aprender representaciones**
- Sin necesidad de **ingeniería de atributos**, las representaciones se aprenden automáticamente
- **Ej. Word Embeddings!**



Hyperparametrs, Dropout, Early Stopping

Búsqueda de Hiperparámetros

Muchos hiperparámetros

- Cantidad de capas
- Cantidad de neuronas en cada capa
- Funciones de activación
- Tamaño de mini-batch
- Algoritmo de optimización
  - con sus hiperparámetros (ej. Learning rate)
- Otros hiperparámetros
  - Prob. De dropout
  - Early Stopping

Búsqueda de Hiperparámetros

- Partimos el conjunto de entrenamiento en 3
  - train
  - val (o dev)
  - test
- Utilizamos train y val para ajustar hiperparámetros
- Evaluamos nuestro modelo final en test

Grid Search y Random Search

- Definimos un **espacio de búsqueda** para cada hiperparámetro
  - Ej. Cant. Capas -> [2,3,4, 5] , LR -> [0.01, 0.001, 0.0001], etc.

- **Grid Search**
  - Se ejecutan todas las configuraciones de hiperparámetros para los espacios definidos
- **Random Search**
  - Se ejecutan N configuraciones de hiperparámetros tomados aleatoriamente en cada espacio

Otras funciones de activación

Leaky ReLU

Parametric ReLU

ELU (Exponential Linear Unit)

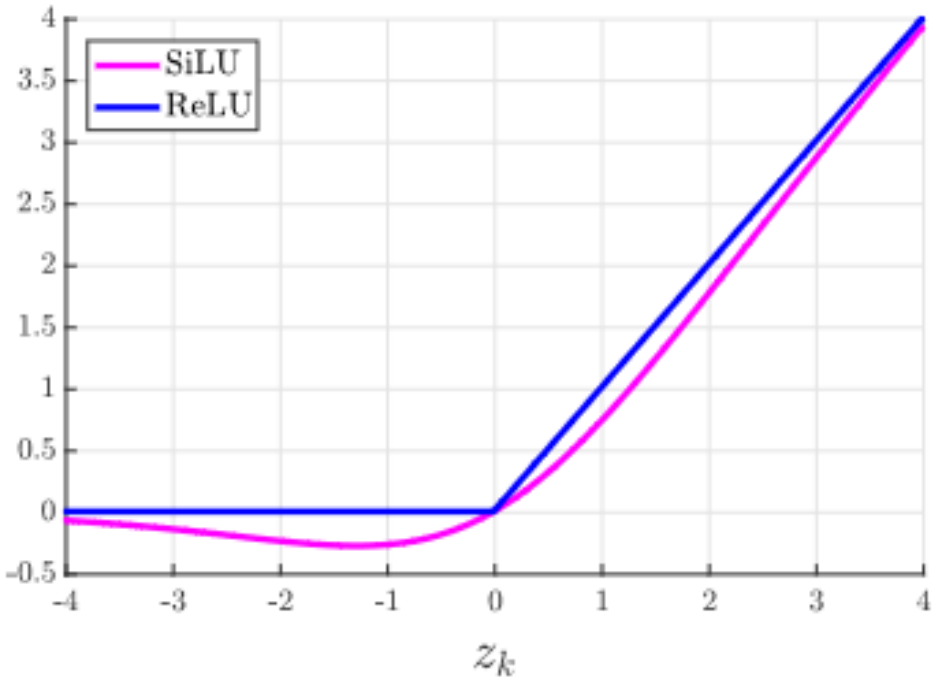
GELU (Gaussian Error Linear Unit)

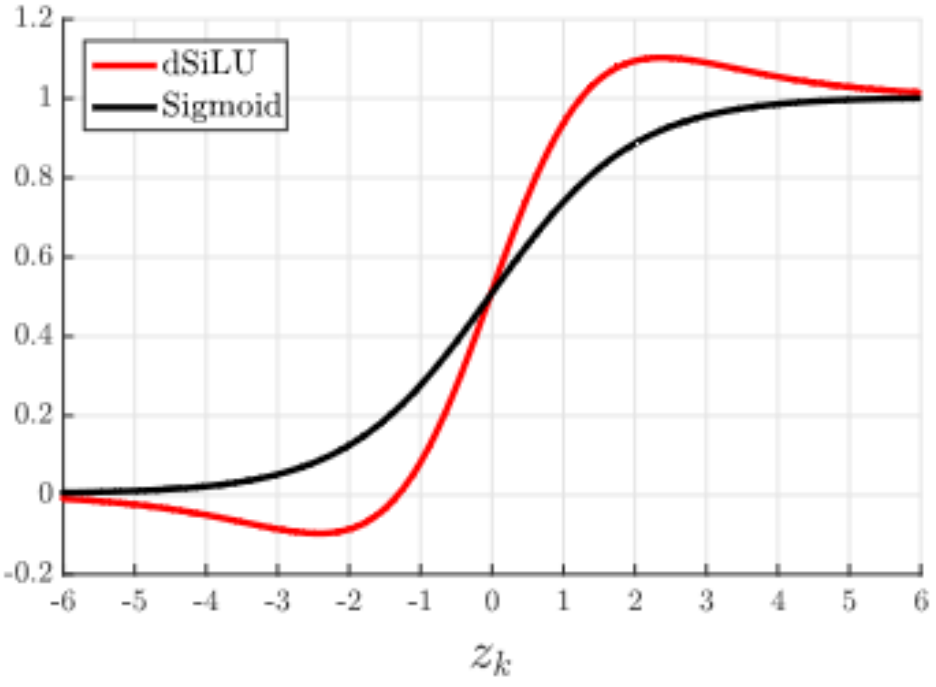
Swish o SiLU (Sigmoid Linear Unit)

dSiLU (derivative SiLU)

...

SiLU y dSiLU

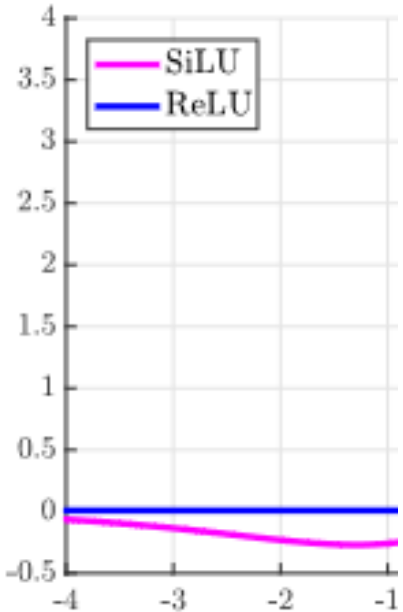


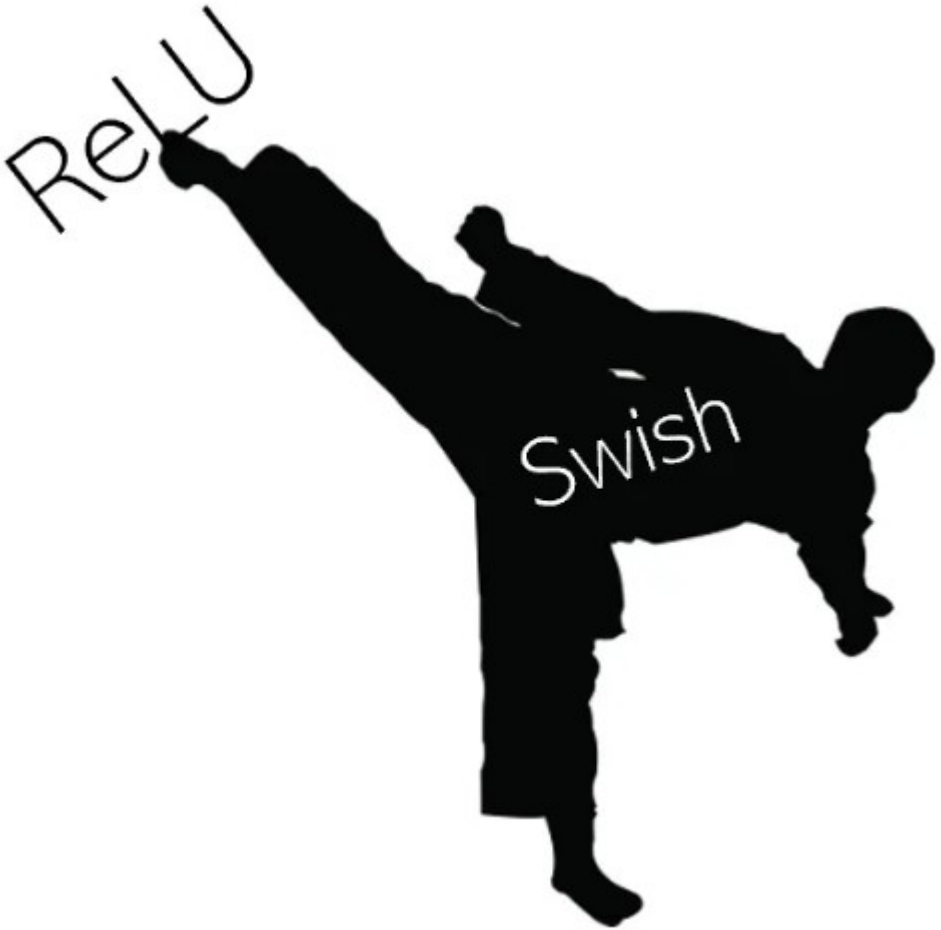


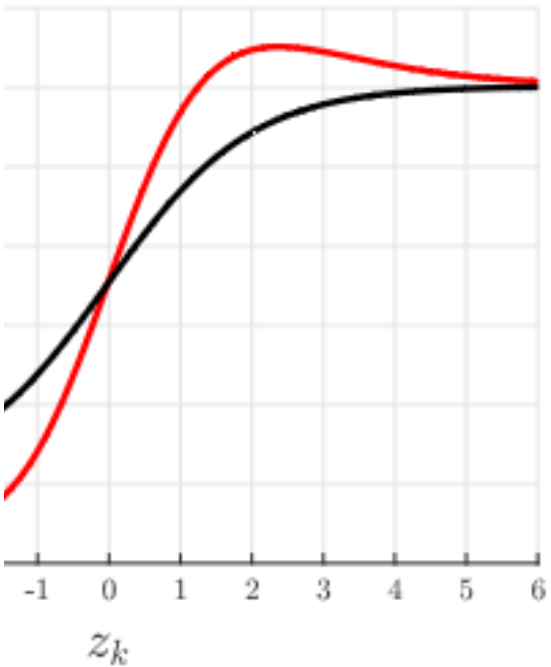
$$\text{silu}(x) = x * \sigma(x)$$

De “Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning” Elfwing et al. 2017

SiLU y dSiLU







De “Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning” Elfwing et al. 2017

Dropout

- Es una técnica de regularización
- Consiste en anular aleatoriamente un conjunto de neuronas de la red durante cada paso de entrenamiento
  - Las neuronas se anulan con una probabilidad P (hiperparámetro)
- Luego de entrenada se utiliza las red sin neuronas anuladas
- Se puede interpretar como que se aprenden varias sub-redes que son utilizadas conjuntamente luego del entremiento

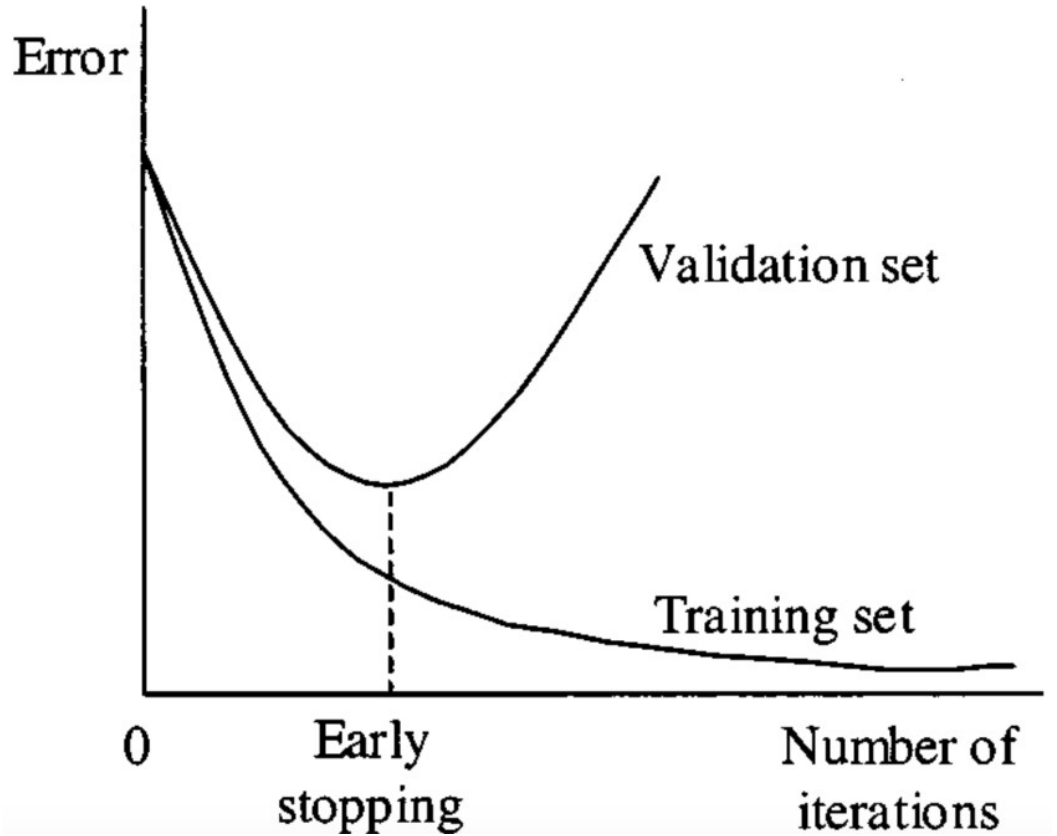
Early Stopping

- ¿Cuándo paramos de entrenar?
  - Cantidad fija de épocas (ej. 20)
  - Early stopping

- **Early stopping** es una técnica de regularización para evitar overfitting. Consiste en monitorear la evolución del entrenamiento del modelo (en el conjunto de validación) y se detiene cuando no se observan mejoras
  - Hiperparámetros: variación_mínima, paciencia

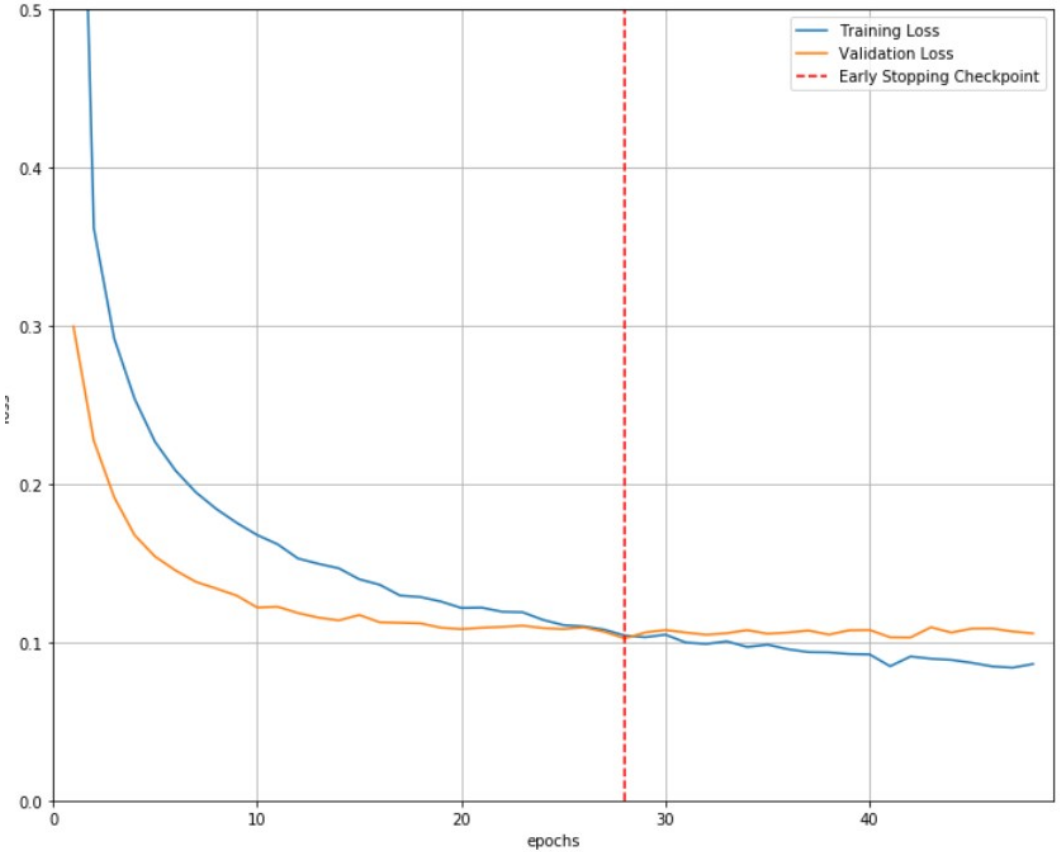
- Se retorna el mejor modelo en validación (**model checkpoint**)

Early Stopping



Imágen obtenida de https://paperswithcode.com/method/early-stopping

Early Stopping



Patience = 20

Imágen obtenida de https://github.com/Bjarten/early-stopping-pytorch

Feedforward Networks for NLP

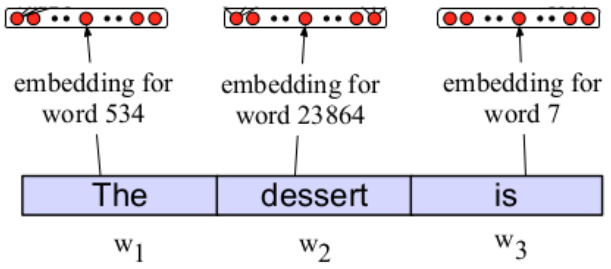
Clasificación de textos

- ¿Cómo vectorizamos la entrada? Aún podemos utilizar lo visto para regresión logística y multinomial
  - Ingeniería de features
  - Bag of words

- Este tipo de modelo permite **constuir representaciones** de la entrada
  - **Embedding layer** (look-up matrix)
    - se ajusta durante el entrenamiento como el resto de los parámetros
  - Se conocen como **word embedding** (vamos a volver sobre esto más adelante)

Clasificación de textos

- Asumir largo fijo de oraciones (Ej. 3)
  - No es muy realista



- Algunas soluciones simples (soluciones más sofisticadas luego)

1. Tamaño de la entrada del largo del review más largo

  - Si la entrada es más corta rellenamos con cero (padding)
  - Si es más larga truncamos (Ej. en test)

2. Si contamos con un conjunto de word embeddings (más adelante)

  - Tomar una representaciónd de largo fijo del texto. Ej. Promedio de las representaciones de las palabras (centroide) o element-wise max

Modelo de lenguaje neuronal

- **Modelo de lenguaje:** Calcular la probabilidad de la próxima palabra en una secuencia (dada cierta historia)
- Se puede hacer con **N-gramas** pero las redes neuronales funcionan mejor
- El estado del arte en modelo de lenguajes está basado en modelos como el **Transformer**
- Pero modelos simples de redes feedforward (completamente conectadas) también funcionan bastante bien!

Neural Language Model

Task: predict next word w_t

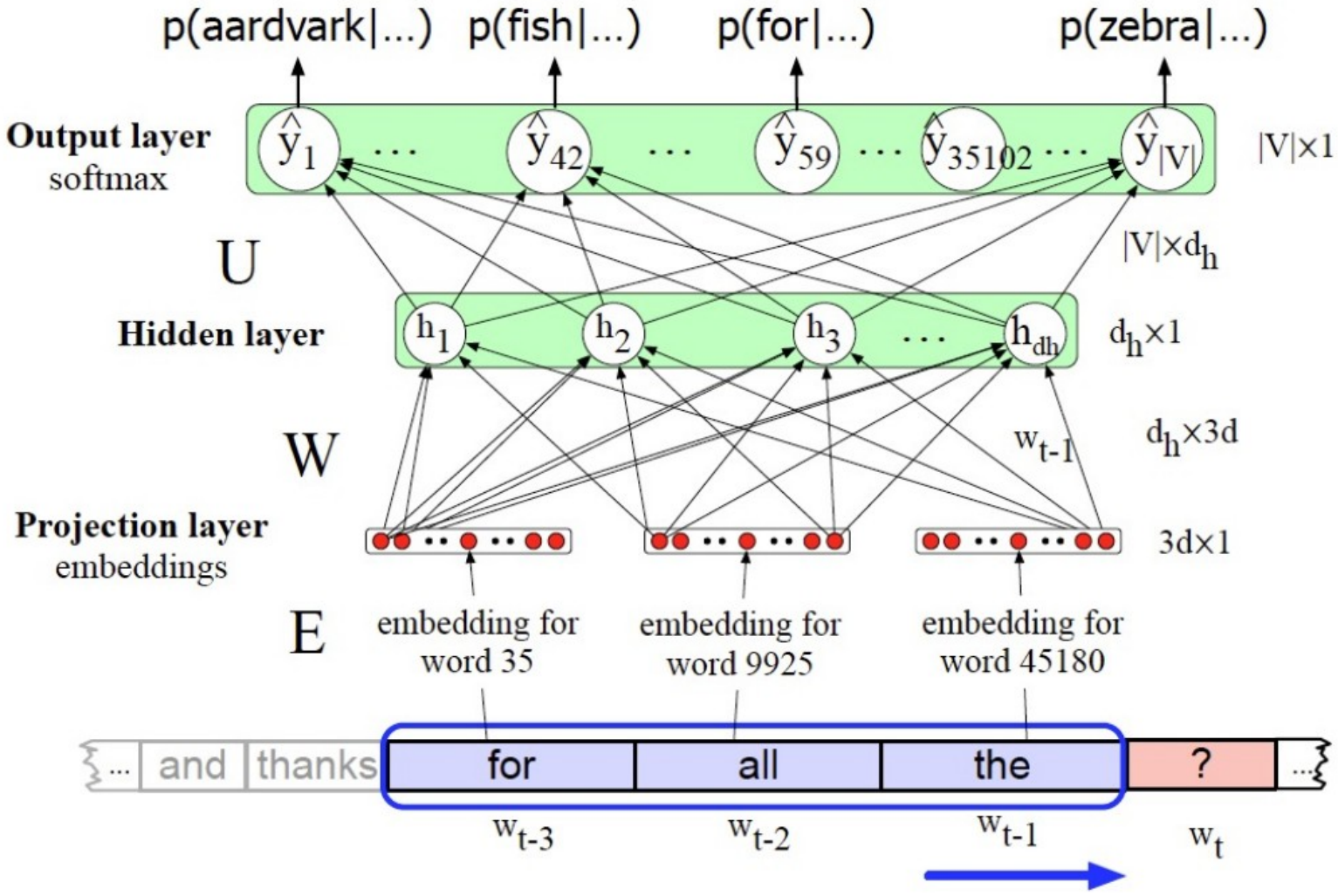
given prior words w_{t-1}, w_{t-2}, w_{t-3}, ...

Problem: Now we're dealing with sequences of arbitrary length.

Solution: Sliding windows (of fixed length)

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$$

Neural Language Model



Why Neural LMs perform better than n-gram LMs?

Training data:

We've seen: I have to make sure that the cat gets fed.

Never seen: dog gets fed

Test data:

I forgot to make sure that the dog gets ___

N-gram LM can't predict "fed"!

Neural LM can use similarity of "cat" and "dog" embeddings to generalize and predict “fed” after dog

Basado en https://web.stanford.edu/~jurafsky/slp3/