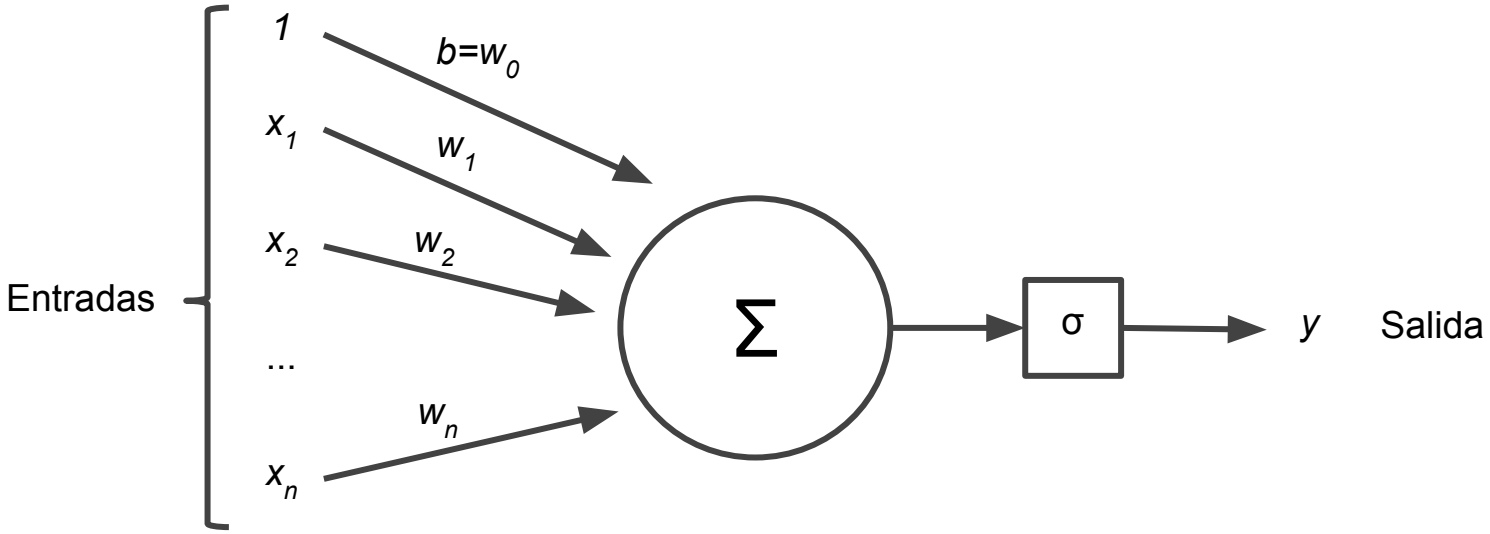
Redes Neuronales para Lenguaje Natural

2024

Grupo de Procesamiento de Lenguaje Natural
Instituto de Computación

Neurona Artificial



$$\hat{x} = [1, x_1, x_2, ..., x_n]$$
$$\hat{w} = [w_0, w_1, w_2, ..., w_n]$$

$$y = \sigma(\hat{x}.\hat{w})$$

Descenso por gradiente estocástico

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

where: L is the loss function
f is a function parameterized by θ
x is the set of training inputs $x^{(1)}$, $x^{(2)}$,..., $x^{(m)}$
y is the set of training outputs (labels) $y^{(1)}$, $y^{(2)}$,..., $y^{(m)}$

$\theta \leftarrow 0$
repeat til done
  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)
    1. Optional (for reporting): # How are we doing on this tuple?
       Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?
       Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?
    2. $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss?
    3. $\theta \leftarrow \theta - \eta\, g$ # Go the other way instead
return θ

Funciones de Activación

- Función logística o sigmoide
- ReLU
- Tangente hiperbólica
- Softmax

Otras…

- Identidad (salida lineal)
- Arcotangente
- Leaky ReLU
- SiLU
- Swish

…

Ojo! No se usa! Luego veremos por qué

¿Qué tienen que cumplir?

Derivable

Monótona ?

No lineal

Funciones de Loss

- Entropía cruzada
- Entropía cruzada categórica
- Error cuadrático medio

Otras…

¿Qué tienen que cumplir?

Derivable

Valores más bajos cuando los resultados de la red se parecen más a los esperados

Redes Multicapa

Problema del XOR

Perceptron de Rosenblatt (1958)

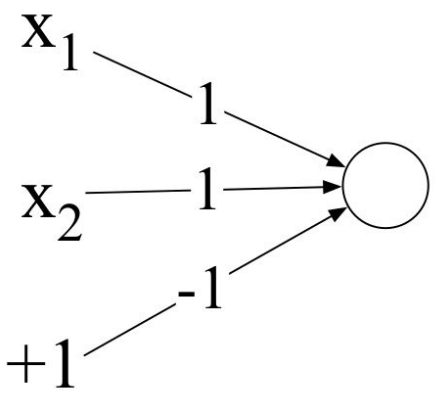
Una neurona como la que vimos, pero donde la función de activación es la función escalón

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

¿Podemos modelar los operadores lógicos usando esta neurona?

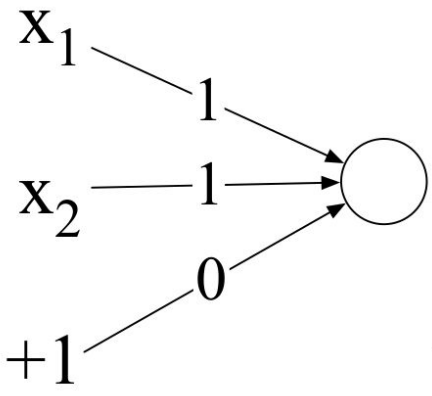
AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

AND y OR



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

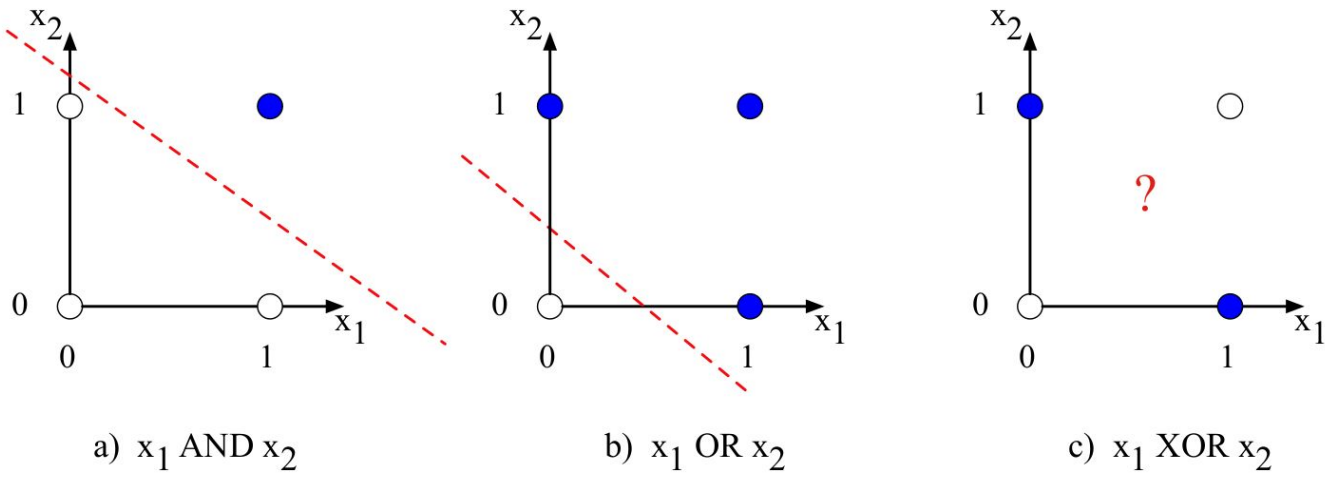
¿Qué pasa con el XOR?

La ecuación del perceptrón deja definida la frontera de decisión como una recta:

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

O sea:

$$x_2 = (-w_1/w_2)x_1 + (-b/w_2)$$



El problema con el XOR es que no es una función linealmente separable

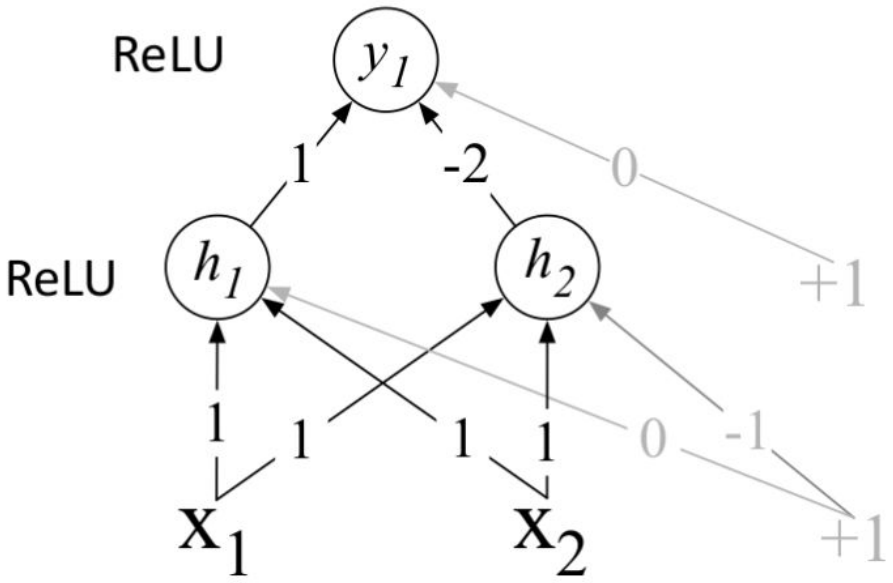
Perceptrons (Minsky & Papert, 1969)

¿Cómo lo resolvemos?

El XOR no se puede calcular con una sola unidad de perceptrón

¿Pero si pusiéramos varias unidades en cascada?

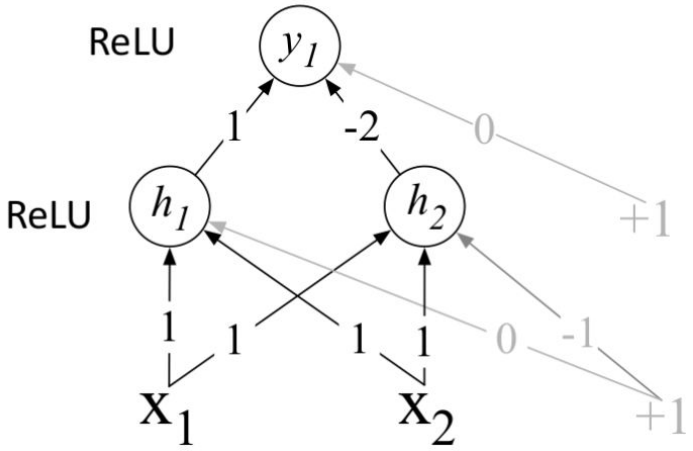
XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

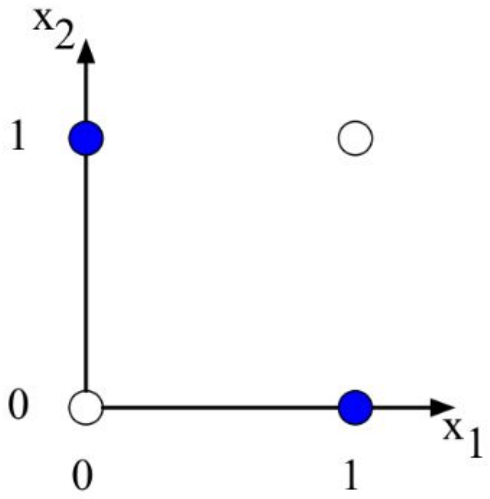


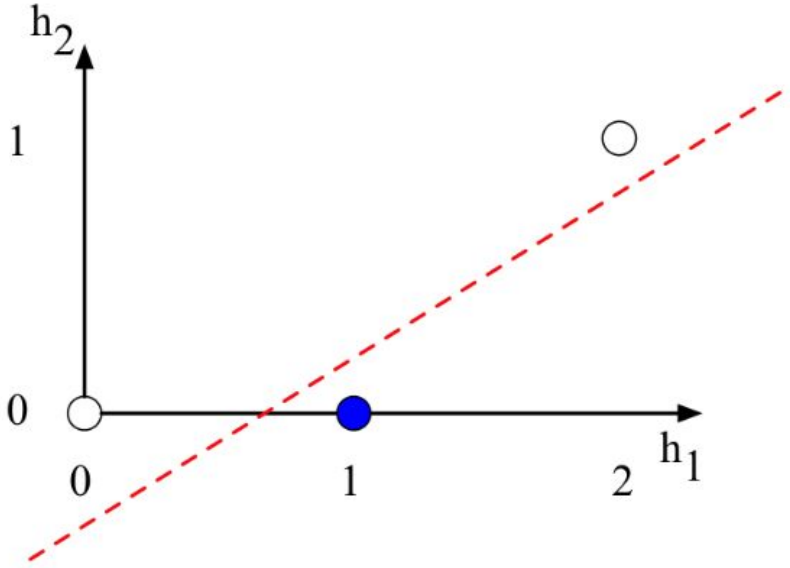
Resolviendo el XOR

La primera capa de unidades realiza una transformación de los valores x en los valores h

Este nuevo espacio sí es linealmente separable



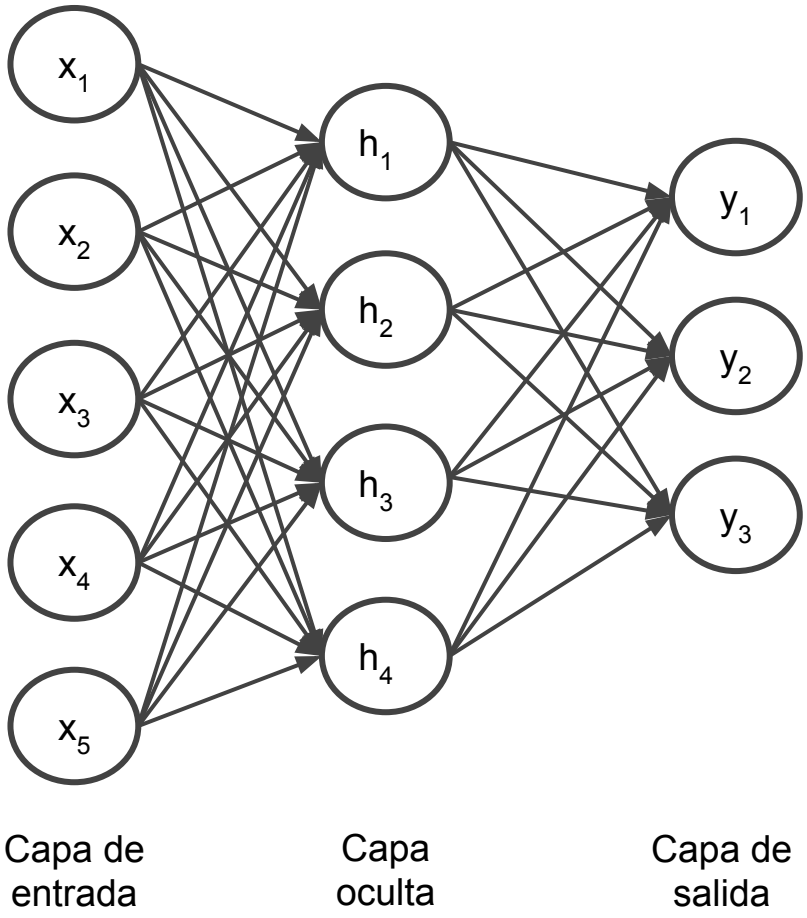




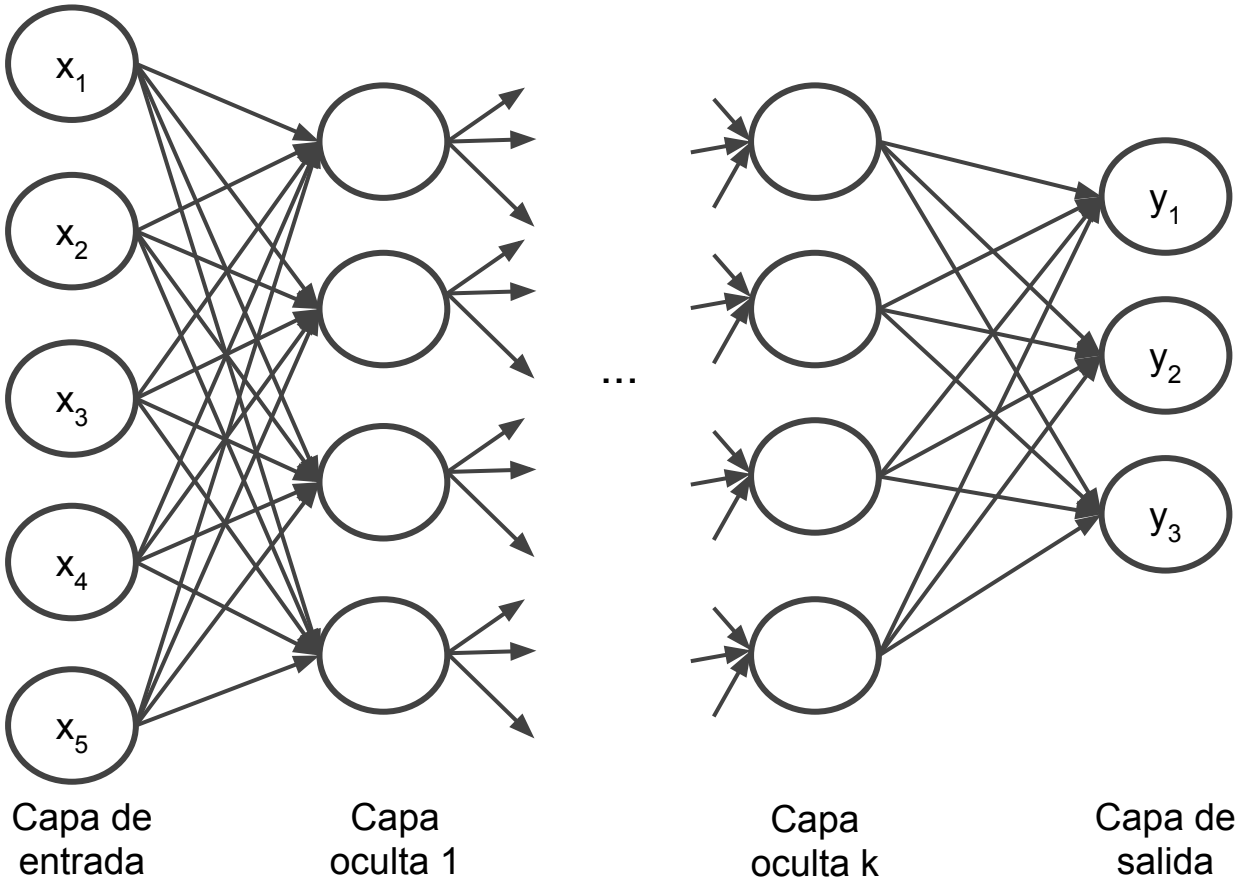
Capa oculta

Para poder modelar funciones más complejas necesitamos al menos una capa más que haga una transformación

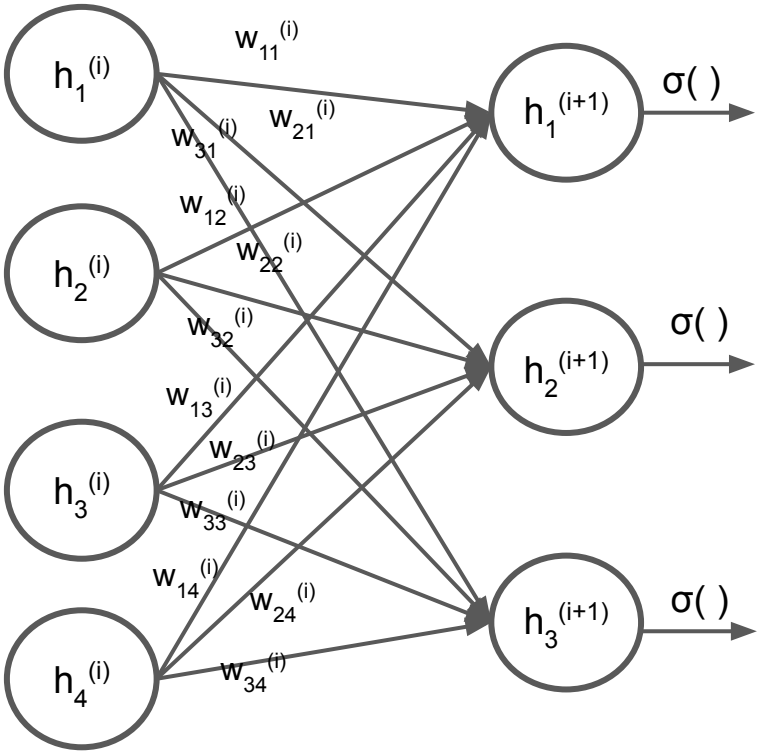
Llamamos capa oculta (*hidden layer*) a la capa que está luego de la entrada pero antes de la capa de salida



Red Feedforward



Red Feedforward



Entrada: $h^{(i)} = [\, h_1^{(i)}, h_2^{(i)}, h_3^{(i)}, h_4^{(i)} \,]$

Salida: $h^{(i+1)} = [\, h_1^{(i+1)}, h_2^{(i+1)}, h_3^{(i+1)} \,]$

$$W^{(i)} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix}$$

$$h^{(i+1)} = \sigma(W^{(i)} h^{(i)})$$

Perceptrón Multicapa

Multilayer Perceptron (MLP)

Otro nombre para estas redes feedforward

Se debe elegir

- Cantidad de capas
- Cantidad de unidades por capa
- Función de activación de cada capa

¿La función de activación de la última capa de qué depende?

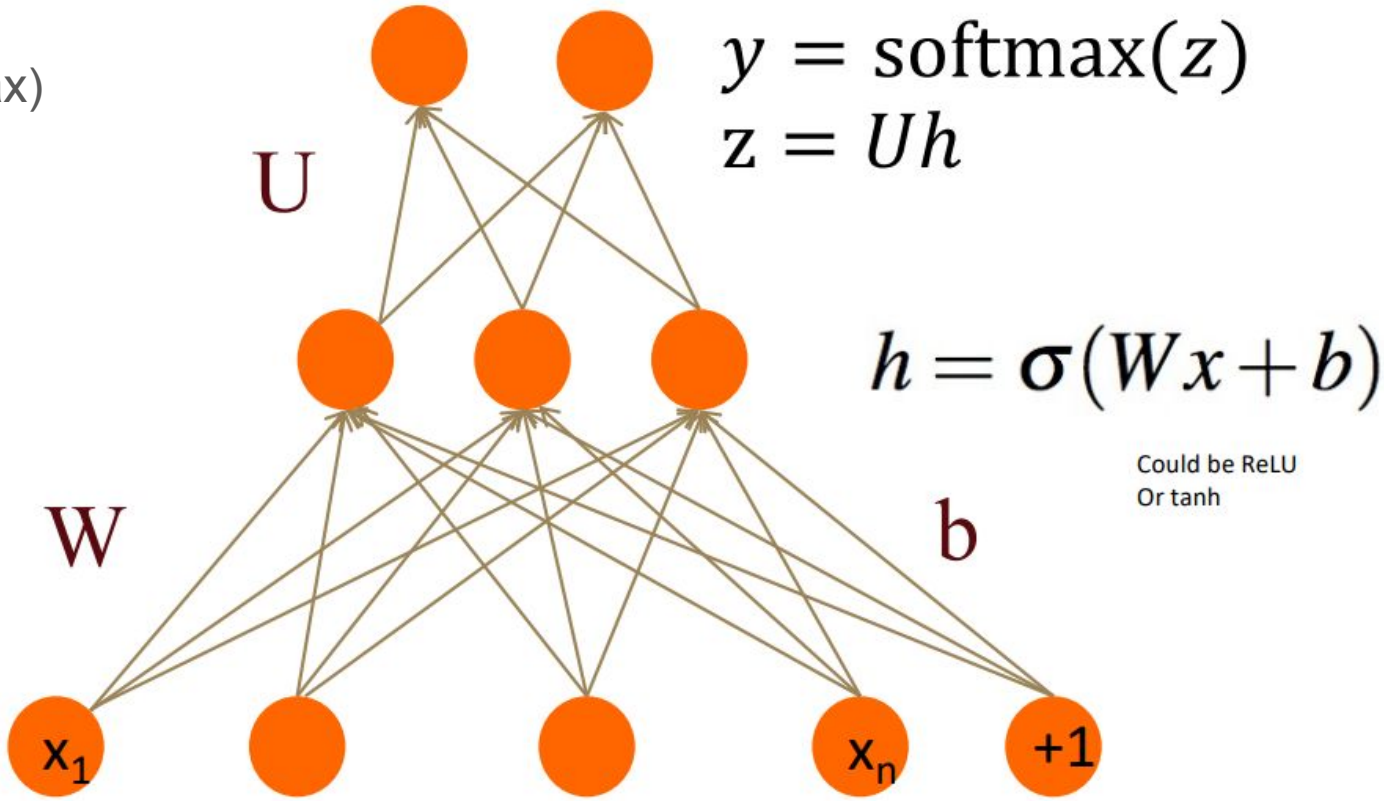
¿Y la función de loss?

Red de dos capas con Softmax (Jurafsky)

Capa de salida
(activación softmax)

Capa oculta
(activación σ)

Capa de entrada
(vector)



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

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

Could be ReLU
Or tanh


Entrenamiento

Descenso por gradiente estocástico

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

where: L is the loss function
f is a function parameterized by θ
x is the set of training inputs $x^{(1)}$, $x^{(2)}$,..., $x^{(m)}$
y is the set of training outputs (labels) $y^{(1)}$, $y^{(2)}$,..., $y^{(m)}$

~~$\theta \leftarrow 0$~~

Inicializar aleatoriamente con valores pequeños (uniforme, normal, otras…)

repeat til done

For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)

1. Optional (for reporting): # How are we doing on this tuple?
   Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?
   Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?
2. $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss?
3. $\theta \leftarrow \theta - \eta\, g$ # Go the other way instead

return θ

Cuándo dar por terminado? Ahora no hay garantía de llegar al mínimo global

Gradiente

$$g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$$

Función de:

- $x^{(i)}$: datos del corpus

- $y^{(i)}$: valores esperados

- θ: pesos de la red

Derivada respecto a θ, la x es fija

¿Cómo calcularla?

Backpropagation: intuición

- Para entrenar necesitamos el gradiente de la función de *loss*
- 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
- Backward pass: se propaga el gradiente desde la capa de salida hacia la capa de entrada

Regla de la cadena

$$f(g(x))' = f'(g(x)) \cdot g'(x)$$

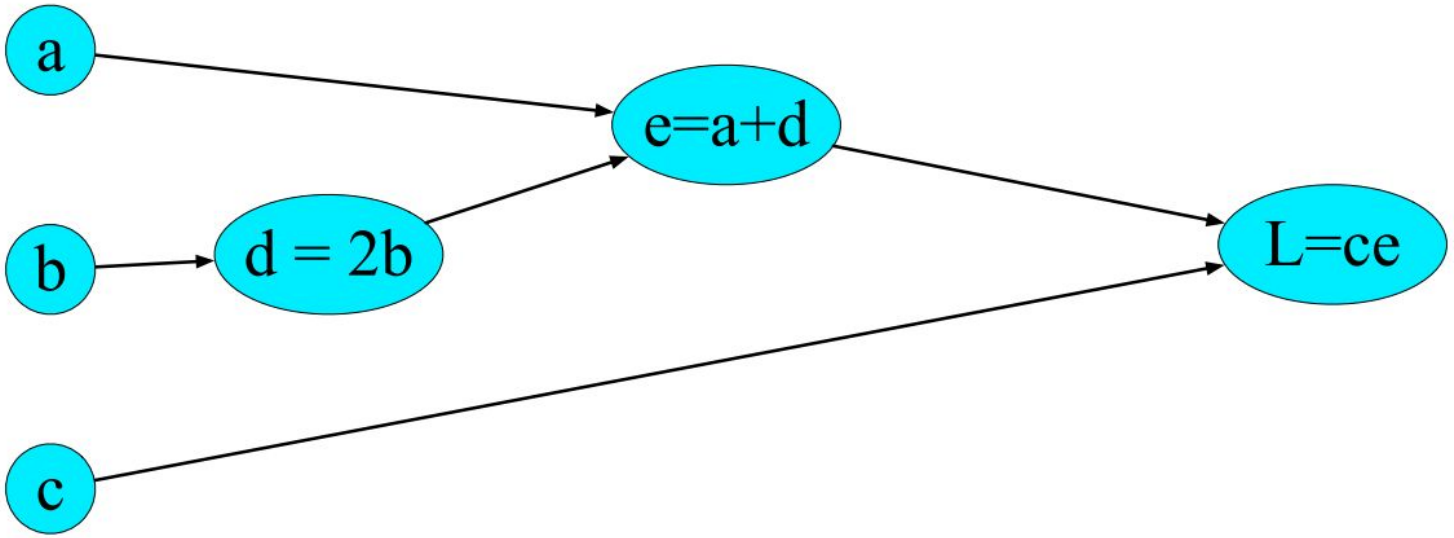
$$\frac{\partial f \circ g}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}$$

Learning internal representations by error propagation. (Rumelhart et al., 1986)

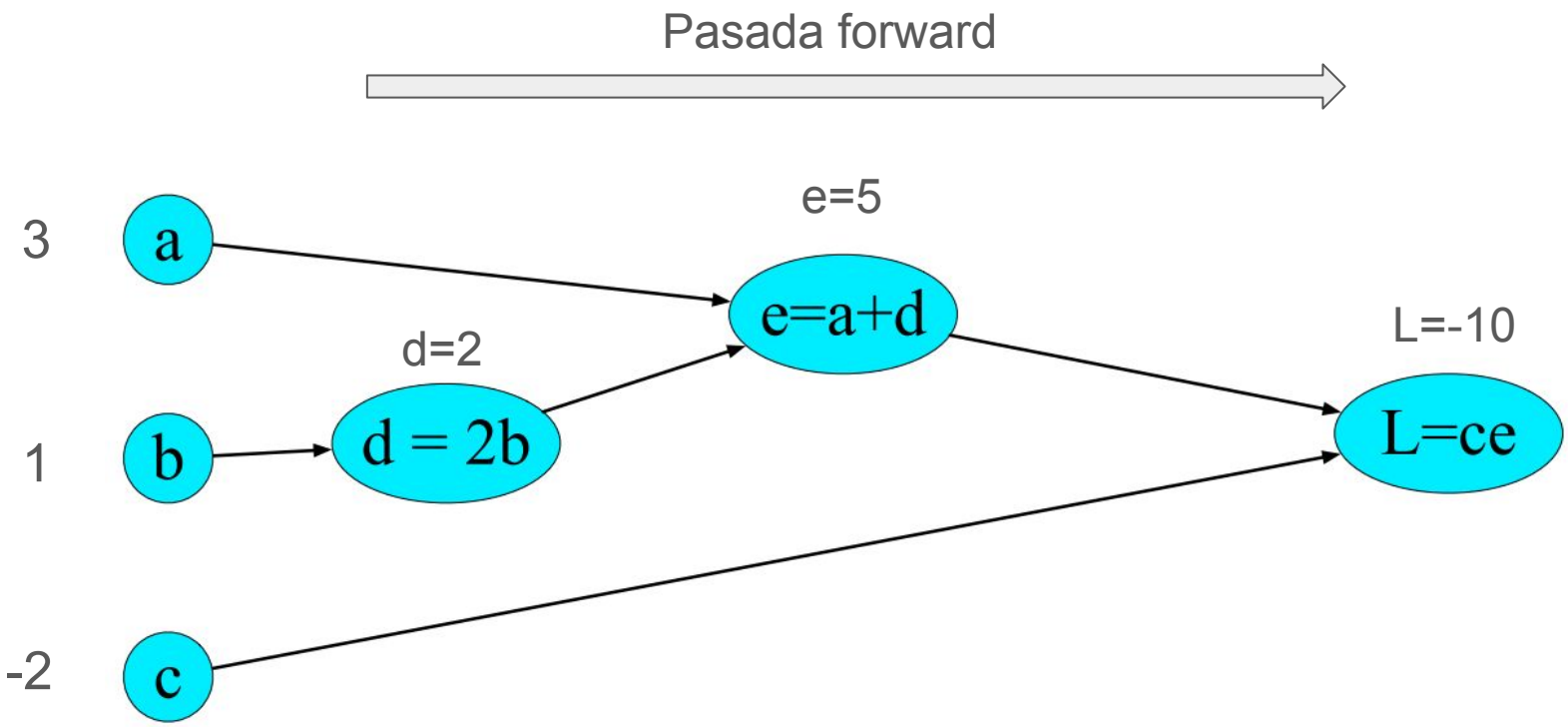
Grafo de computación

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

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



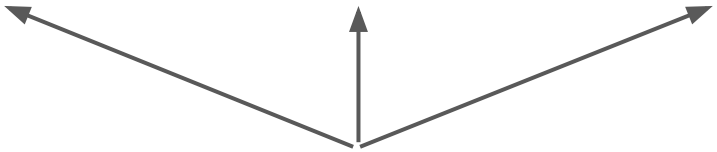
Grafo de computación



Grafo de computación

Nos interesa calcular las derivadas respecto a cada entrada

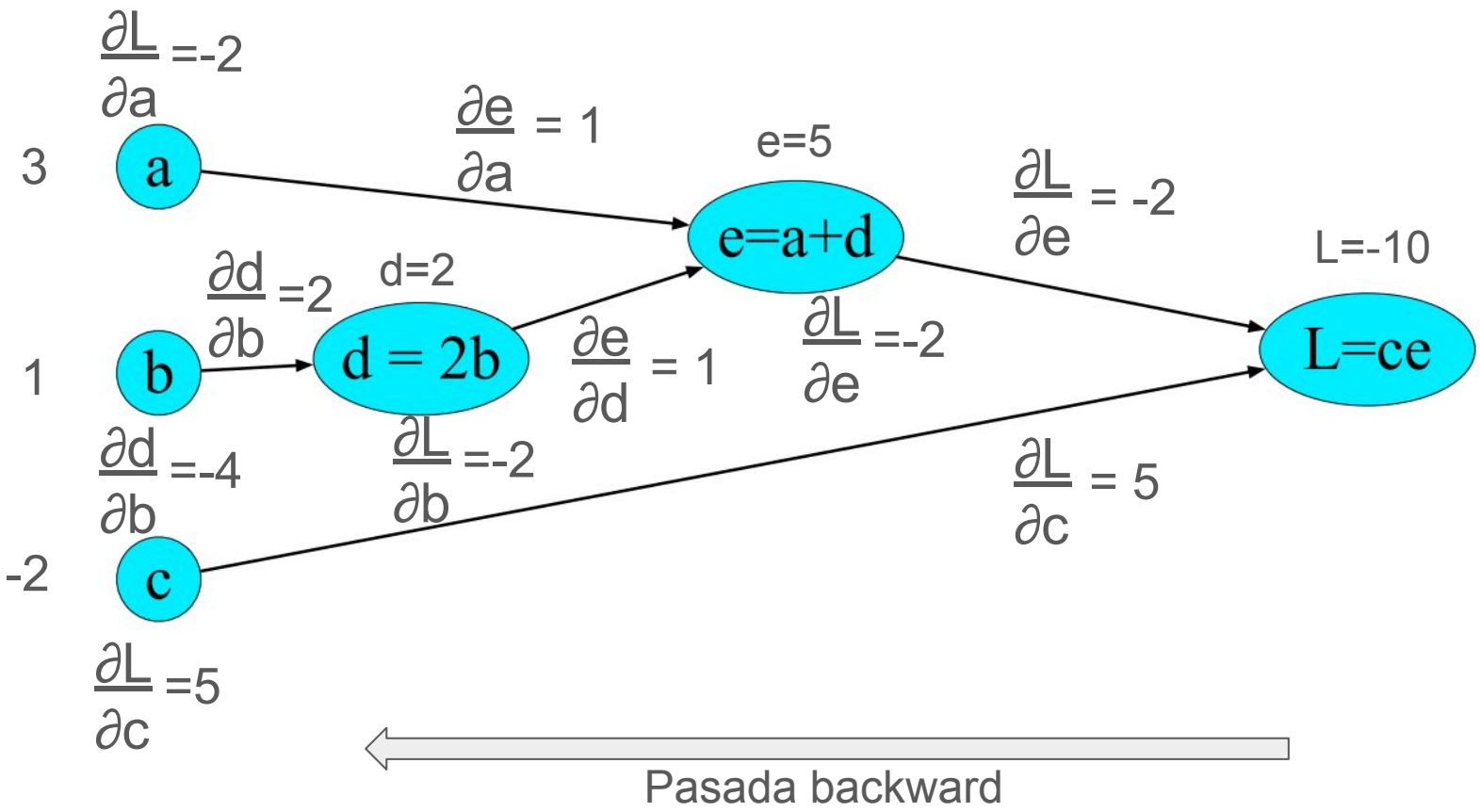
$$\frac{\partial L}{\partial a} \qquad \frac{\partial L}{\partial b} \qquad \frac{\partial L}{\partial c}$$



Cómo pequeños cambios en cada variable impactan el resultado final de L

En vez del cálculo analítico, lo calcularemos con el grafo de computación

Grafo de computación



Backpropagation

$$g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$$

- Antes del entrenamiento desarrollamos el grafo de computación de la red y la función de *loss*
- Composición de funciones “elementales”
- Pasada forward: Presentamos las entradas a la red y computamos todos los valores de activación intermedios hasta la salida
- Pasada backward: Vamos calculando desde la salida y hacia atrás los valores de todas las derivadas

Backpropagation

El grafo de computación permite la diferenciación automática

Las operaciones tensoriales se computan más rápido en hardware específico (Ej. GPUs, TPUs)

Seguimos utilizando SGD con mini-batches

Términos a tener en cuenta:

- Step: un episodio de actualización del gradiente
  - Aplicar la red a un mini-batch
  - Realizar backpropagation y actualizar los pesos
- Epoch: pasar una vez por el corpus de entrenamiento entero
  - Generalmente implica varios steps

Backpropagation

En la práctica se utilizan variantes de SGD:

- AdaGrad (Adaptative Gradient Algorithm)
- AdaDelta (AdaGrad extension)
- RMSProp (Root Mean Square Propagation)
- Adam (Adaptive Moment Estimation)

…

Introducen nuevos hiperparámetros


Sobreajuste

Sobreajuste

Las redes neuronales son una familia de funciones con mucha flexibilidad y capacidad de representación

Podemos representar casi cualquier función matemática con una sola capa oculta[1]

Por consiguiente tienden a sobreajustar muy fácilmente!

- Regularización
- Dropout
- Early stopping

[1] *Approximation by Superpositions of a Sigmoidal Function (Cybenko 1989)*

Dropout

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 la red

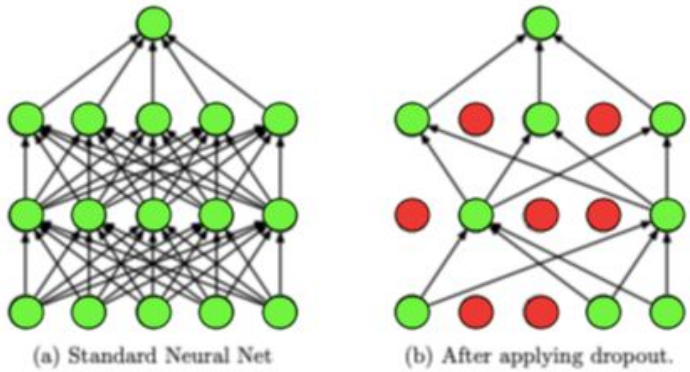
sin anular neuronas

Se puede interpretar como que se

aprenden varias sub-redes que son

utilizadas conjuntamente luego del

entremiento



(a) Standard Neural Net

(b) After applying dropout.

Early Stopping

¿Cuándo paramos de entrenar?

- Cantidad fija de épocas (ej. 20)
- Cuando la diferencia entre iteraciones no supera cierto ε
- Early stopping

Monitoreamos la evolución del entrenamiento del modelo (en el conjunto de dev u otro conjunto separado) y paramos cuando no se observan mejoras

Hiperparámetros: variación mínima, paciencia o tolerancia

Se retorna el mejor modelo en dev (model checkpoint)

Early Stopping

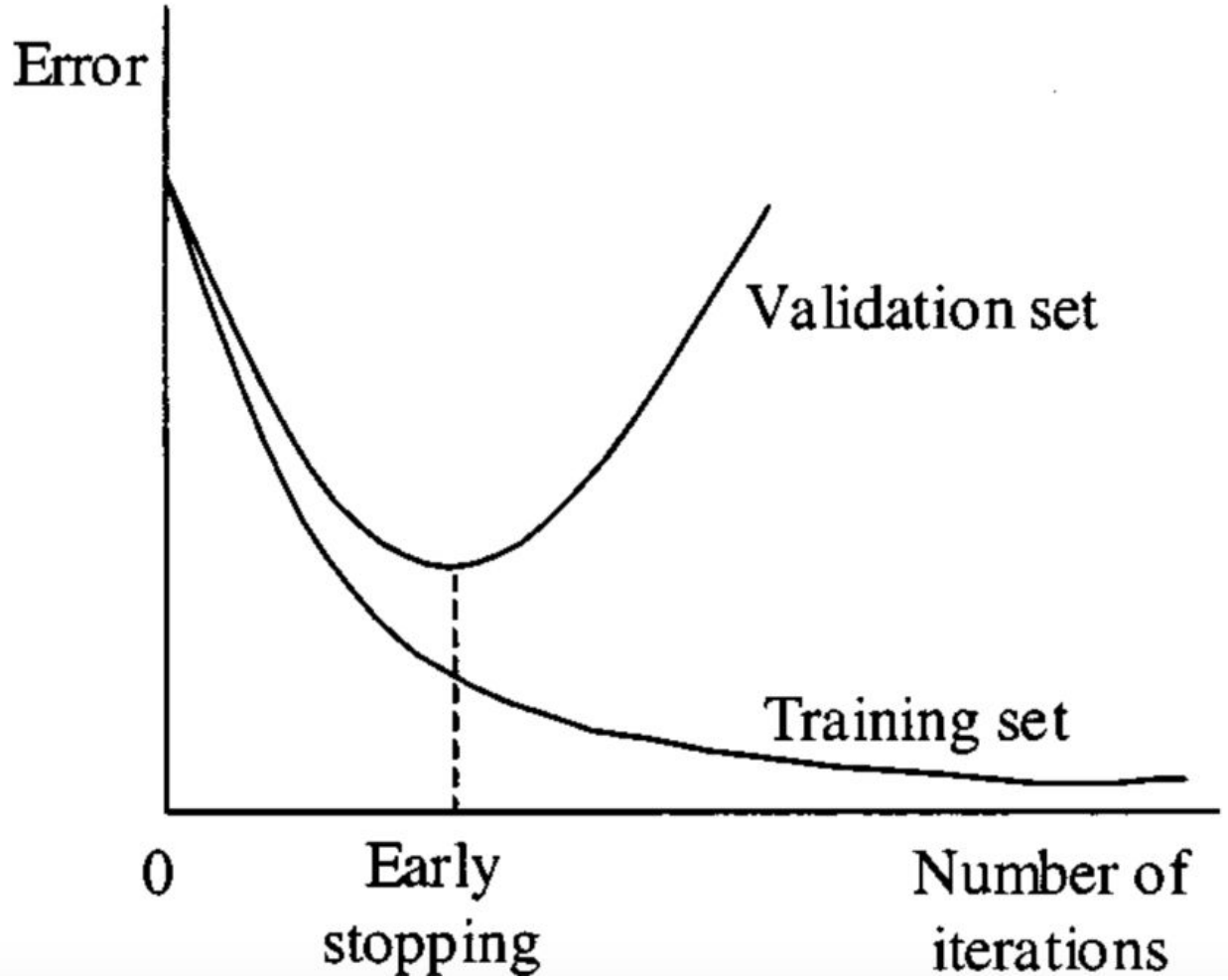


Imagen obtenida de https://paperswithcode.com/method/early-stopping

Búsqueda de Hiperparámetros

- Learning rate
- Tamaño del mini-batch
- Cantidad de capas
- Cantidad de unidades por capa
- Función de activación de cada capa
- Probabilidad de dropout
- Tolerancia de Early stopping

… (cada vez habrá más hiperparámetros!)

Siempre terminaremos probando varios modelos

Importante! Partir en train, dev y test

Búsqueda de Hiperparámetros

Técnicas automáticas:

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

Representación del Texto

Ejemplo: Análisis de Sentimiento

Las entradas de mi red son números, y la salida también

¿Cómo hago para representar mi texto mediante números?

Alternativa 1: Extracción de features (manuales)

Lista de palabras positivas P={buenísima, buena, gustó, linda, recomiendo}

Lista de palabras negativas N={horrible, divague, mamarracho}

x_1 = cantidad de palabras de la lista P en el texto

x_2 = cantidad de palabras de la lista N en el texto

x_3 = la palabra “no” aparece en el texto

x_4 = el signo de exclamación “!” aparece en el texto

x_5 = cantidad de palabras en el texto

…

¿Cómo representamos un texto?

Problemas a resolver

- Cómo representar una palabra
- Cómo representar varias palabras

Representación de Palabras

One-hot encoding

- Vector del tamaño del vocabulario (e.g. 10.000 palabras más frecuentes)
- Vale 0 para todas las dimensiones excepto la de la palabra a representar

D = {perro, gato, árbol, saltar, correr, … australopithecus … sicómoro …}

v_{saltar} = [0, 0, 0, 1, 0, 0, …..]

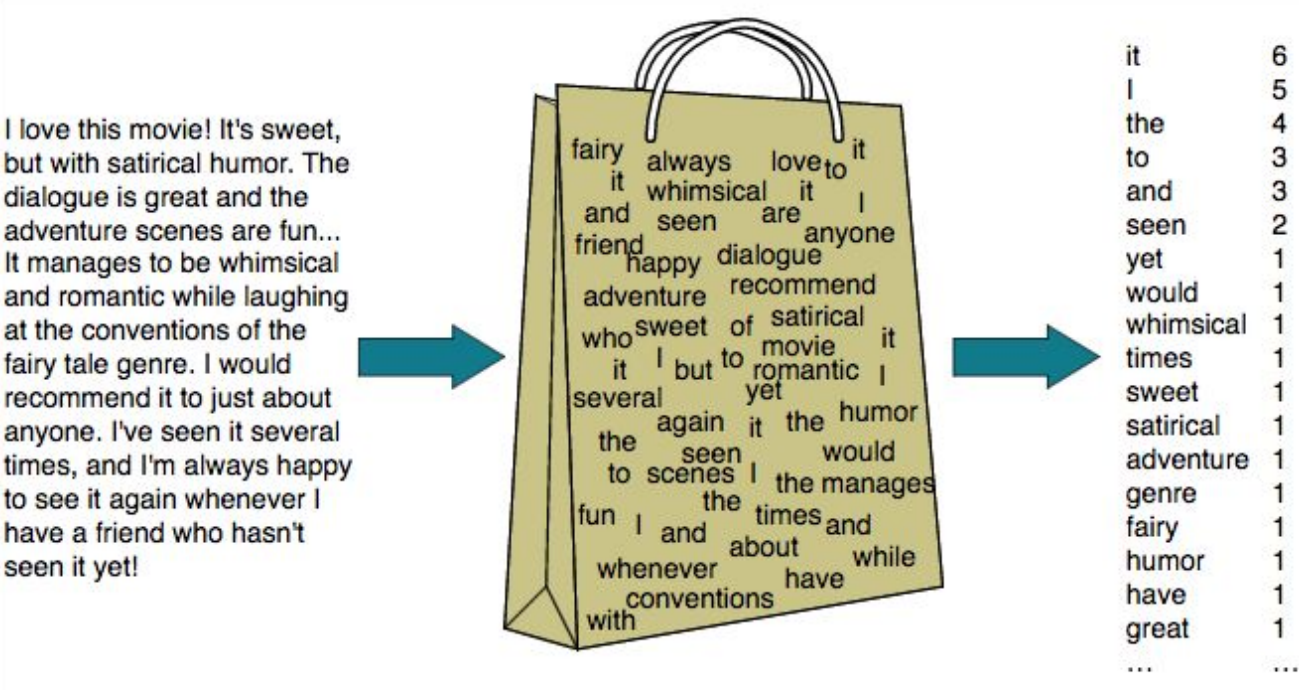
v_{gato} = [0, 1, 0, 0, 0, 0, …..]

Mucho mejor: Word Embeddings! (lo veremos más adelante)

Representación de Textos

- Representar cada una de las palabras concatenadas
  - Por ejemplo los vectores one-hot (o los embeddings)
- Problema: largo fijo
  - Largo máximo de un texto del corpus (?)
- Pesos de las palabras del principio se actualizan más que los del final
- Pero esta idea puede servir como base para otras (más adelante)

Representación de Textos



Bag of Words

Bag of Words

Vector de largo fijo (el vocabulario)

Cada componente representa la cantidad de veces que aparece la palabra en el documento (por ejemplo en un review)

Se puede calcular como la suma de los vectores *one-hot* del documento

- Es un método de agregación

Otro método de agregación muy usado: tf-idf

- Ponderar cada componente *k* por su idf: $\log(N/d_k)$

Mucho mejor: Agregación de Word Embeddings! (lo veremos más adelante)

Representación de Textos

Representaciones de agregación:

- Bag-of-words, tf-idf, bag-of-ngrams, promedio de embeddings…

Problema principal:

- Se pierde el orden de la entrada!

“la vi pero no me gusta” *“no la vi pero me gusta”*

Para solucionar esto veremos *arquitecturas secuenciales* (en unos días)