

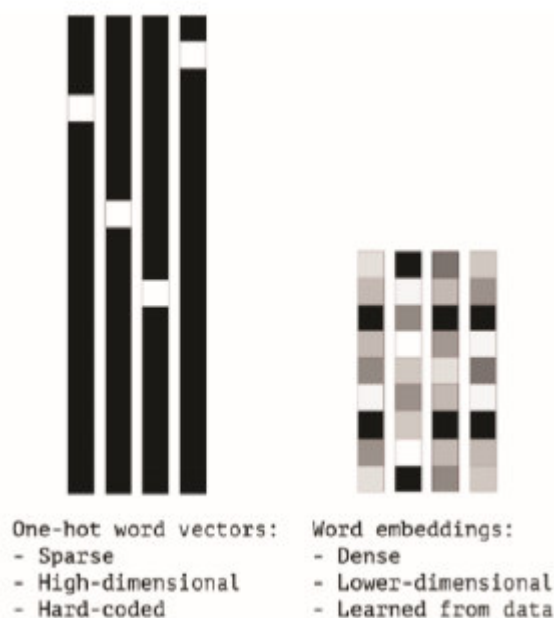
Escuela Latinoamericana de Aprendizaje Profundo

Curso de Lenguaje

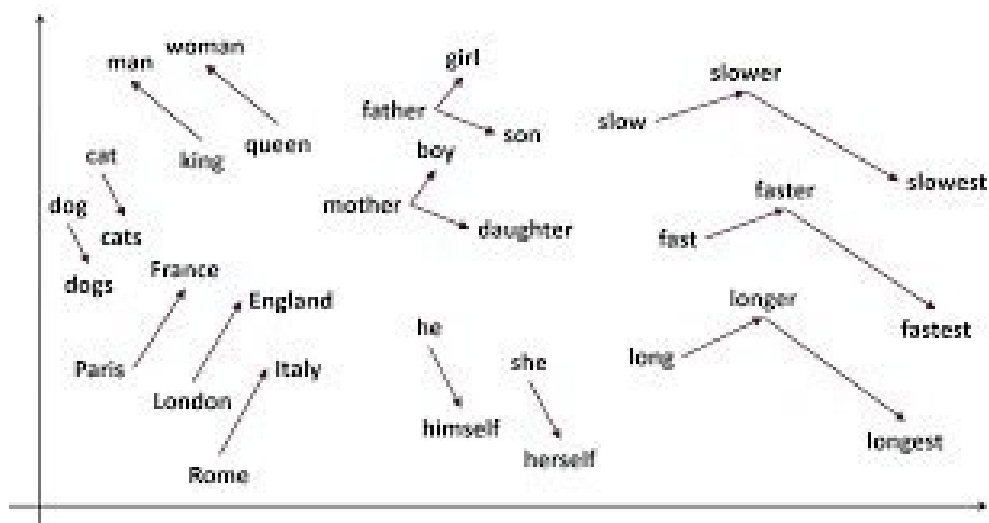
Alexandra González Eras

Recapitulando embeddings

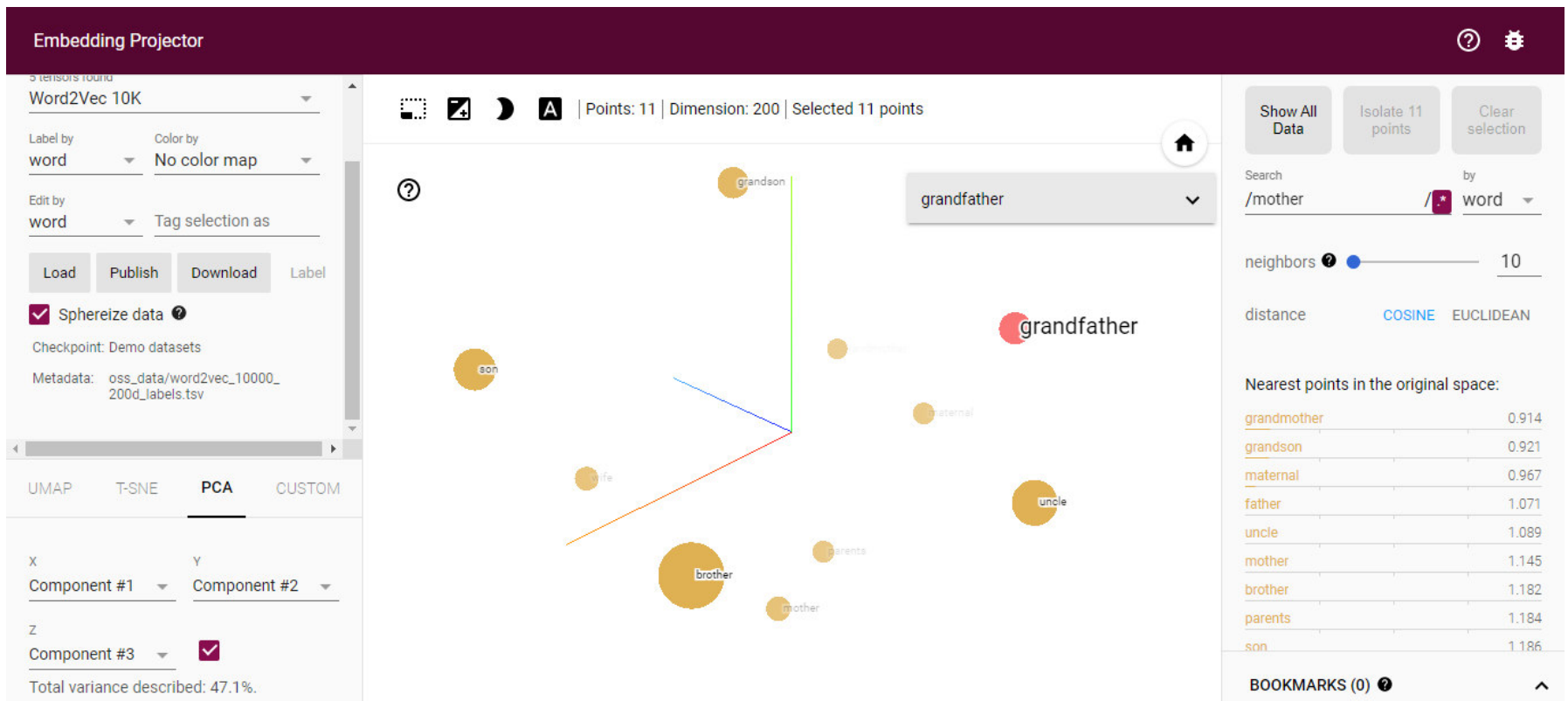
En **Natural Language Processing** (NLP) los Embeddings logran hacer que palabras sobre sentimientos positivos -“alegría”, “felicidad”- queden cercanas pero distanciadas de las que significan sentimientos negativos “odio”, “tristeza”.



Embedding es un vector de alta dimensionalidad de palabras que tienen similitudes entre si. Cada palabra se sitúa en un espacio n -dimensional y se agrupa con otras de acuerdo a un valor de similitud.



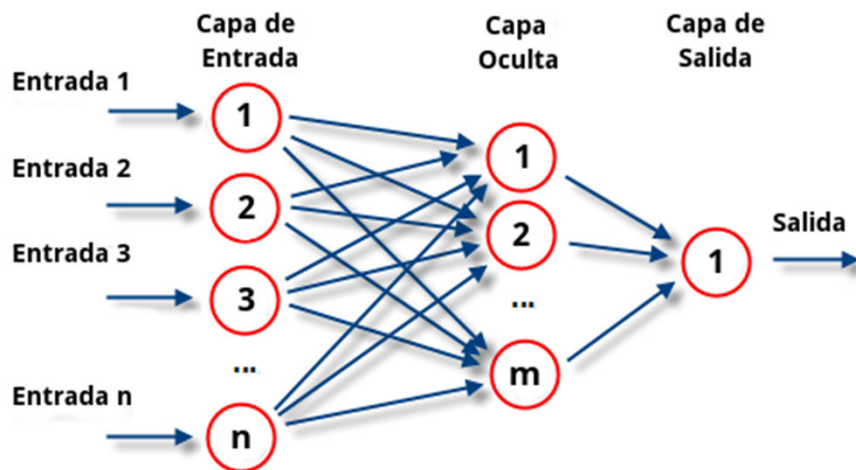
Recapitulando embeddings



[Demo de embeddings](#)

Modelos Neuronales

Las redes neuronales están compuestas de neuronas, que a su vez se agrupan en capas interconectadas. En cada neurona, se realizarán una serie de operaciones las cuales, al optimizar, consiguen que la red aprenda.

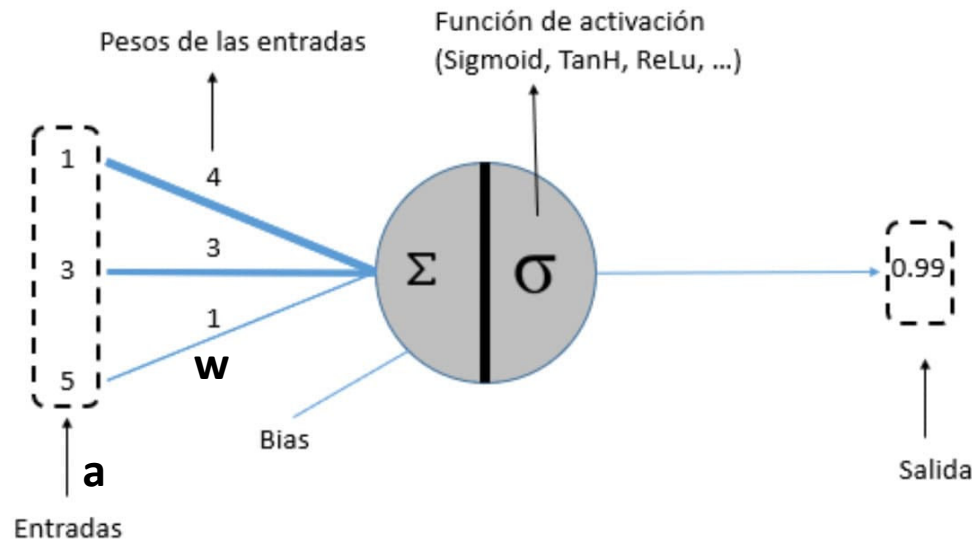


Capa de entrada: recibe las entradas. Los valores vendrán definidos por los datos de entrada, mientras que el resto de capas recibirán el resultado de la capa anterior (Densas o interconectadas)

Función de propagación depende del conjunto de entradas a cada neurona y de la función de activación de la neurona

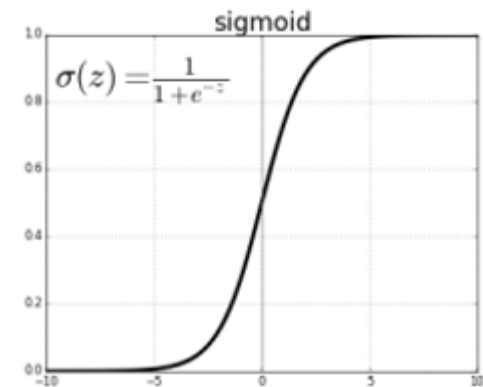
Capa de salida que se representa como una función $f(x)$ (clasificador binario)

Función de propagación



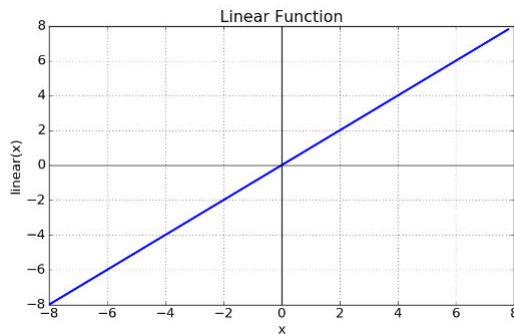
1. La neurona recibe la sumatoria de las entradas ponderadas (multiplicadas por los pesos w), agregado el valor del bias (sesgo)

2. Al resultado de la suma pondera y el bias se le aplica la función de activación



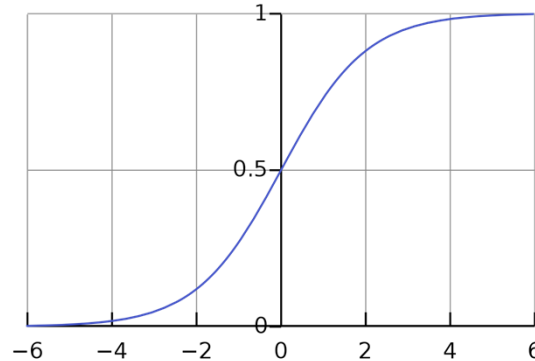
3. El resultado es una función $f(x) = y$, que dependiendo de la función de activación de la neurona tendrá por ejemplo rango entre cero y uno

Funciones de activación



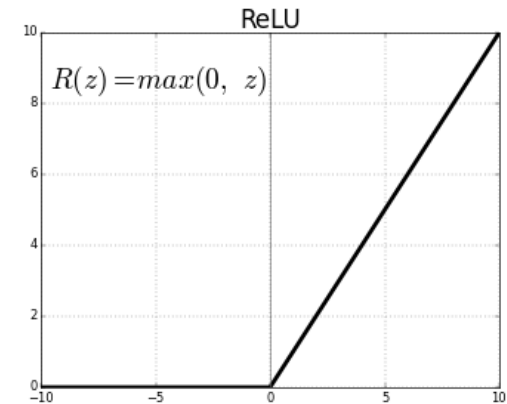
Función lineal (regresión)

La entrada es igual a la salida
la red neuronal genera un valor único
Predicción del valor de un número de ventas.



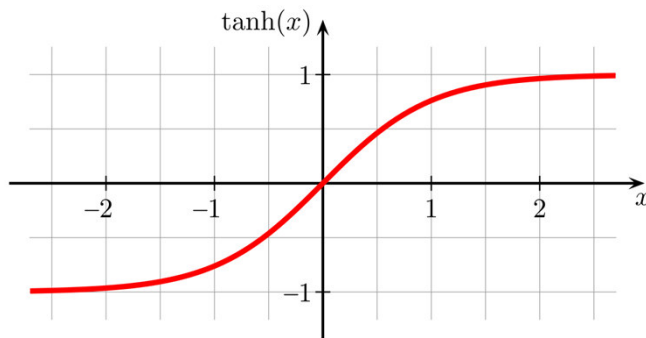
Función Sigmoide (probabilidad)

Valores de salida entre cero y uno
Se usa en la última capa
para clasificar en dos categorías.
Tiende a generar
desvanecimiento de gradiente



Función Relu

Para valores negativos, la función devuelve cero.
Para valores positivos, la función devuelve el mismo valor.
La más usada evita el desvanecimiento del gradiente



Función Tanh transforma los valores introducidos a una escala (-1,1).

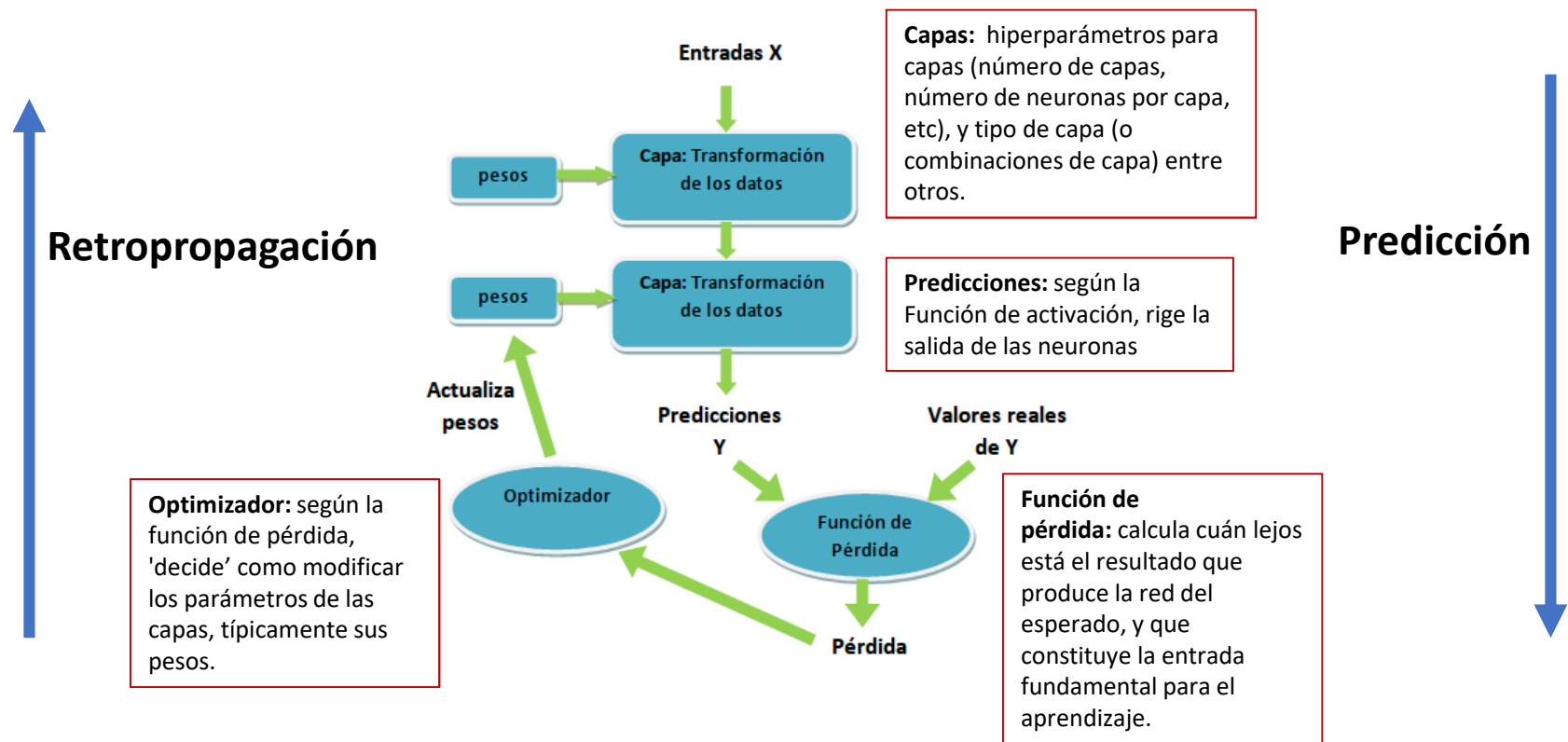
Aprendizaje de una red

En el entrenamiento una red neuronal aprende de datos clasificados (etiquetados), ajustando el error en la retropropagación, donde el ajuste de los pesos de cada neurona es uniformes. De este modo consideramos un Aprendizaje Supervisado con las siguientes etapas:

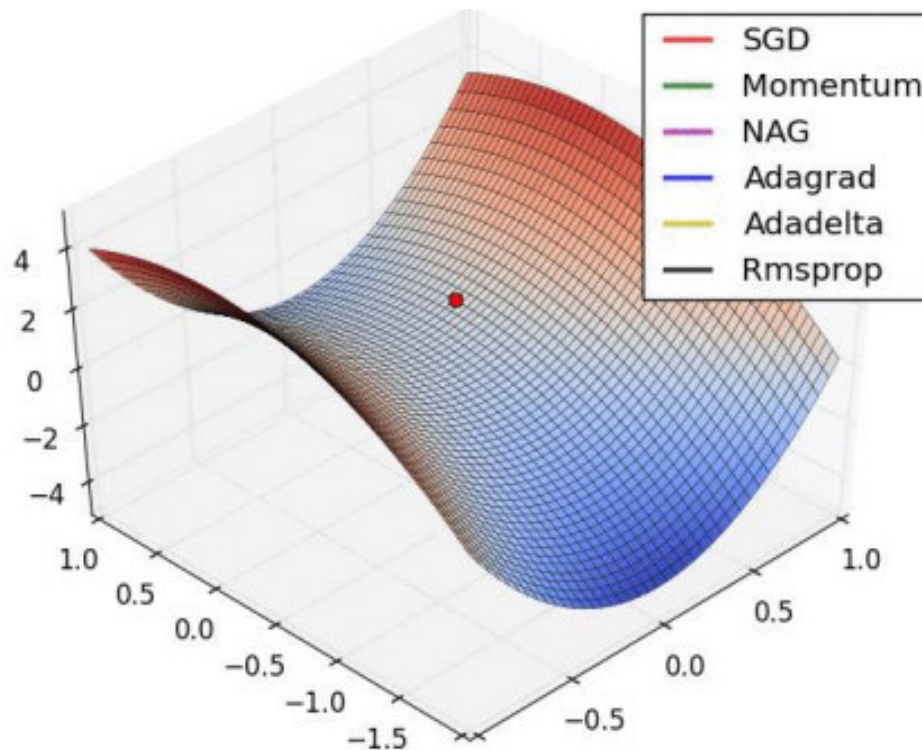
- **Separación:** Dividir los datos de entrada en conjunto de entrenamiento y conjunto de prueba. Es común usar también un conjunto de validación.
- **Entrenamiento:** A partir de los casos (x,y) conocidos, modificar los pesos para minimizar el error entre y y y' (retropropagación)
- **Prueba:** Durante el aprendizaje (validación) o a posteriori, verificar el ajuste de la red ante nuevos casos (x,y) no usados para el entrenamiento
- **Operación:** Una vez que la red ha aprendido correctamente, según los resultados de la fase de prueba y validación, colocar la red en ejecución.

Aprendizaje en Deep Learning

En DL además de las diferentes funciones de activación, también hay varios tipos de funciones de pérdida y una gran variedad de versiones de descenso de gradiente (optimizadores) para parametrizar las implementaciones. En líneas generales el proceso se puede resumir como:



Optimizadores



SGD: minimizar la pérdida de un modelo predictivo con respecto a un conjunto de datos de entrenamiento

ADAGRAD: controla la disminución excesiva de tasa de aprendizaje.

RMSProp: es una variación de AdaGrad en la que utiliza el concepto de "ventana" para considerar solo los gradientes más recientes.

Funciones de pérdida

La Función de pérdida es aquella que se debe minimizar, y para ello se calcula su gradiente por ser una función multidimensional.

Se pueden usar dos clases de error: el error medio cuadrático (MSE) o el error medio absoluto MAE

$$MSE = \frac{1}{n} \sum_{j=1}^n (y_l - \hat{y}_l)^2 \qquad MAE = \frac{\sum_{j=1}^n |e_j|}{n}$$

Para operaciones de clasificación se usan funciones dependientes de la entropía. La más conocida es entropía cruzada, que puede ser:

Categórica: si se trata de un clasificación multiclases (categorical_crossentropy)

Binaria: si es un clasificador de dos clases (binary_crossentropy)

Tipo de problema	Activación última capa	Función de pérdida o error
Clasificación binaria	Sigmoide	binary_crossentropy
Clasificación multiclase exclusiva	Softmax	categorical_crossentropy
Clasificación multiclase no exclusiva	Sigmoide	binary_crossentropy
Regresión para valores arbitrarios	Lineal	Error cuadrático medio (mse)
Regresión para valores entre 0 y 1	Sigmoide	Error cuadrático medio (mse) o binary_crossentropy

Desvanecimiento explosión de gradiente

Desvanecimiento de gradiente: cuando los pesos tienen valores menores a 1 y muchas capas, el gradiente irá disminuyendo a valores muy pequeños (ineficacia de ajuste de pesos)

Explosión de gradiente: cuando la derivada de la función de activación toma valores muy grandes, los pesos aumentan exponencialmente (inestabilidad de la red), provocada en redes de muchas capas

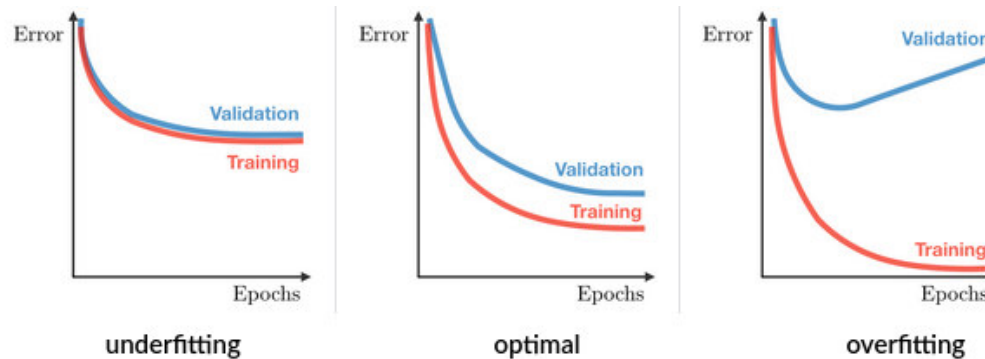
Tener muchas capas y datos de entrenamiento, provocar ajustar muchas veces los pesos. Se puede definir lotes para realiza el cálculo por cada lote

Ejemplo en Keras

```
model.fit(train_images, train_labels, epochs=5, batch_sizes=128)
```

Si se tiene 60.000 ejemplos de entrada, para lotes de 128 por época, entonces se calcularían, sólo 469 corridas de retropropagación por época → 2345 fases de retropropagación en total

Overfitting, Underfitting, Optimal fit

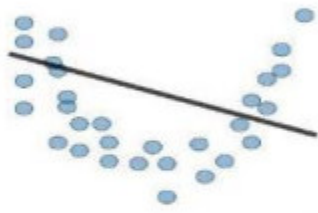


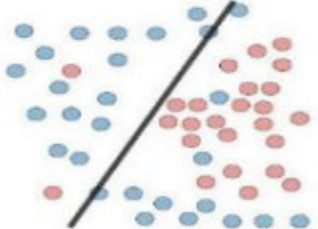
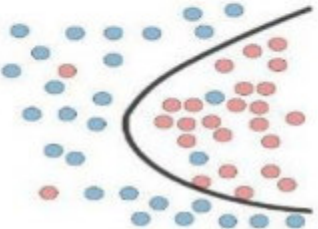
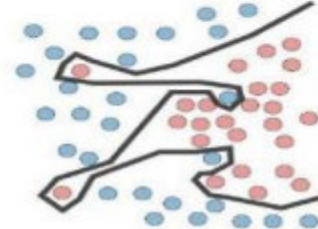
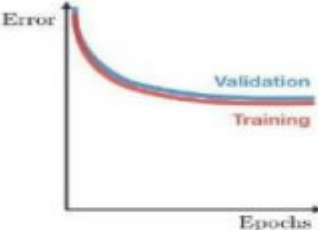
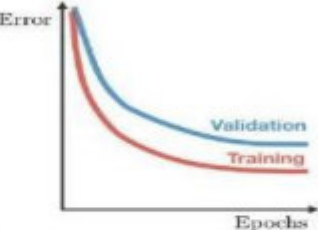
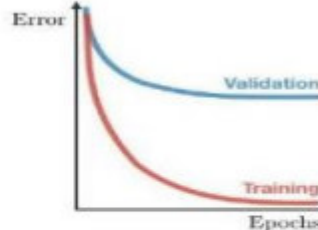


Underfitting (o sub ajuste): cuando el modelo es incapaz de obtener resultados correctos por falta de entrenamiento o de más muestras. Se reconoce visualmente cuando existe altos valores de pérdida tanto en el set de entrenamiento como en el de validación.

Overfitting (o sobre ajuste): cuando el modelo se ajusta solo a los datos de entrenamiento y se vuelve incapaz de reconocer nuevos datos. Es visualmente reconocido cuando existe una separación importante entre las pérdidas del set de entrenamiento y el set de validación.

Optimal fit: cuando el modelo logra captar el comportamiento general de los datos. Se puede identificar al obtener valores de pérdidas en los datos de validación que no difieren demasiado de las pérdidas de los datos de entrenamiento.

Overfitting, Underfitting, Optimal fit

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none">- High training error- Training error close to test error- High bias	<ul style="list-style-type: none">- Training error slightly lower than test error	<ul style="list-style-type: none">- Low training error- Training error much lower than test error- High variance
Regression			
Classification			
Deep learning			
Remedies	<ul style="list-style-type: none">- Complexify model- Add more features- Train longer		<ul style="list-style-type: none">- Regularize- Get more data

Reducción de la red

- Una vez que se ha detectado el *overfitting*, una primera solución es reducir la talla de la red

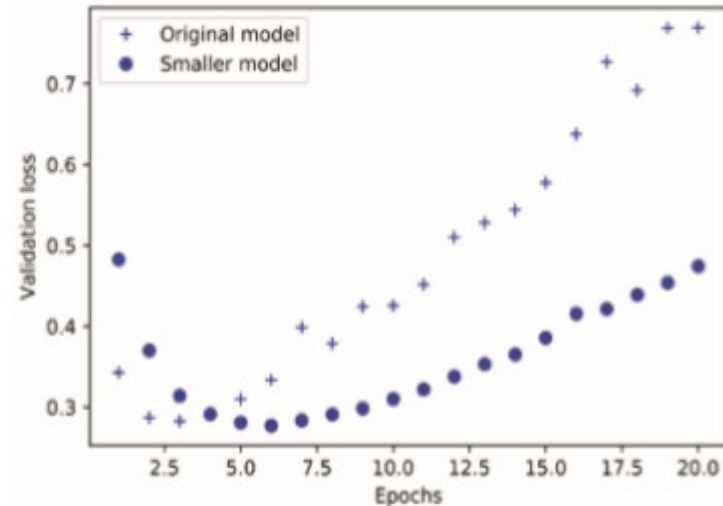
Modelo inicial (+)

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Modelo reducido (•)

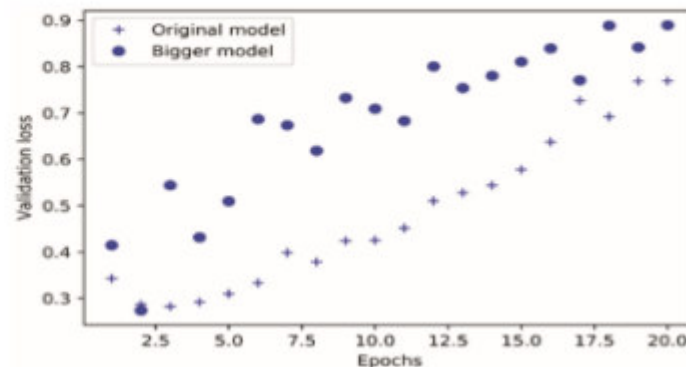
```
model = models.Sequential()
model.add(layers.Dense(4, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```



- Si se agranda el modelo, el *overfitting* empeora y la pérdida es más errática ...

Modelo agrandado (•)

```
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```



Penalizaciones de la red

- El objetivo es ajustar el error con un factor de castigo para que se altere el cálculo de los pesos.
Por ejemplo, sea el error cuadrático medio para ajustar una red:

$$E = \frac{1}{2}(y - \hat{y})^2 \quad \text{Se agrega el castigo} \quad E = \frac{1}{2}(y - \hat{y})^2 + L_Norma$$

Lo común es agregar la L1-Norma y L2-Norma para las regularizaciones L1 y L2 respectivamente

$$L1 = \lambda \sum_{i=1}^n |w_i| \quad L2 = \lambda \sum_{i=1}^n w_i^2 \quad \text{donde } \lambda \text{ es el parámetro de regularización}$$

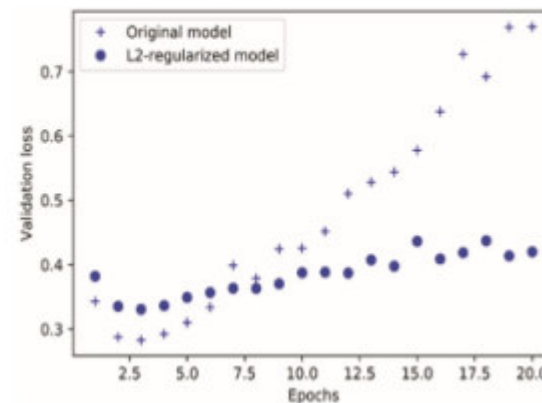
Visto más general:

$$E' = \frac{1}{2}(y - \hat{y})^2 + \lambda \sum_{i=1}^n |w_i| \quad E' = \frac{1}{2}(y - \hat{y})^2 + \lambda \sum_{i=1}^n w_i^2$$

- En consecuencia al calcular los nuevos pesos con $w^{k+1} = w^k + \eta \frac{\partial E'}{\partial w^k}$ será sobre el error regularizado ...

```
from keras import regularizers

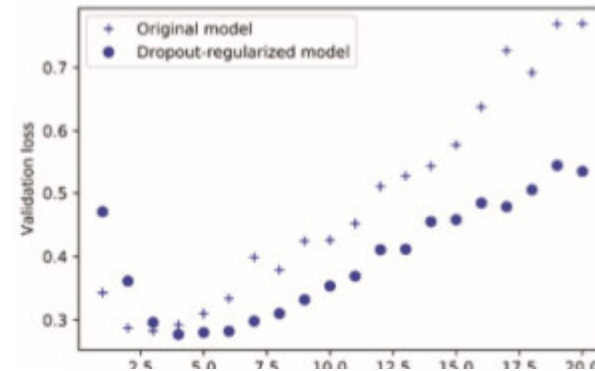
model = models.Sequential()
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                        activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                        activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```



Dropout

- Al igual que la regularización, se aplica durante la fase de entrenamiento. El objetivo es deshabilitar, aleatoriamente, algunas neuronas, en una o varias capas, durante las fases *forward* y *backward*. La tasa de *dropout*, para anular pesos, más común es entre 0.2 y 0.5, es decir, entre 20% a 50% de los pesos de la capa siguiente se llevan a cero.

```
model = models.Sequential()  
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))  
model.add(layers.Dropout(0.5))  
model.add(layers.Dense(16, activation='relu'))  
model.add(layers.Dropout(0.5))  
model.add(layers.Dense(1, activation='sigmoid'))
```



- Recapitulando, las tres maneras de corregir el *overfitting* son:
 1. Reducir el tamaño de la red neuronal, con ajuste, gracias al conjunto de validación
 2. Modificar el cálculo del error de la predicción, mediante L-Normas, para ajustar los pesos durante el *backpropagation*, de tal manera que eviten el *overfitting*.
 3. Eliminar neuronas, aleatoriamente, por lo que los pesos en la siguiente capa se desactivan y se reduce la capacidad de memorización de la red neuronal.

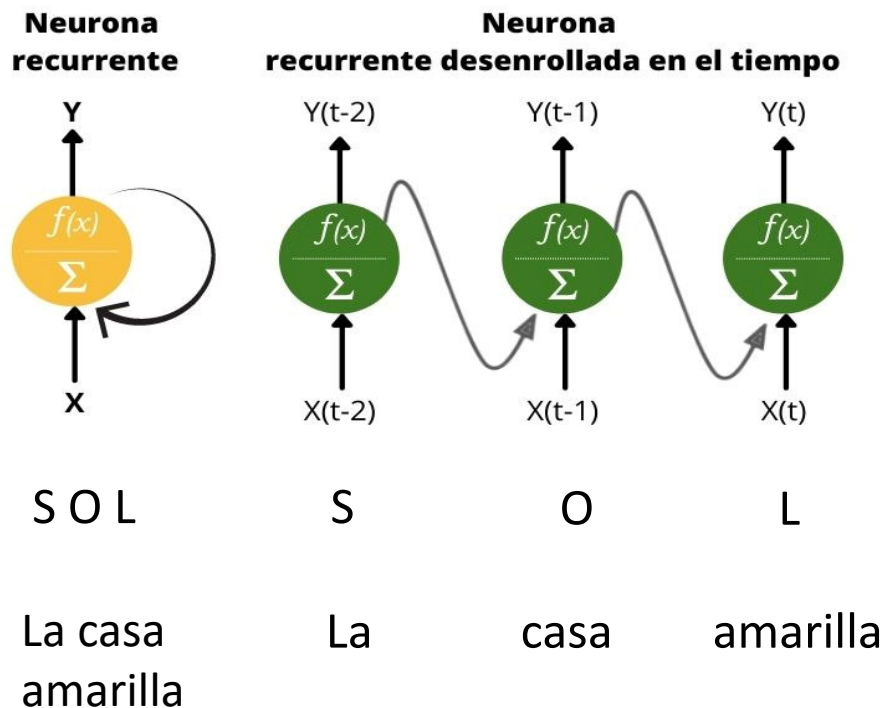
Consideraciones de las NN

- Las redes de neuronas están diseñadas para tener el mismo tamaño de entrada y salida. Sin embargo, **en los problemas secuenciales, los inputs y los outputs pueden tener tamaños muy diferentes dependiendo de la observación**. Por ejemplo en procesos de traducción, las entradas y las salidas varían en su longitud.
- Las redes de neuronas no comparten características entre las diferentes posiciones. Es decir, **las NN asumen que cada input (y output) es independiente uno del otro**.



Redes de Neuronas Recurrentes

La idea de las RRN es hacer uso de información secuencial, que presenta una dependencia de los datos procesados anteriormente por ejemplo, a la izquierda se encuentra la representación de la neurona recurrente



Las redes de neuronas recurrentes pueden aprender una función de probabilidad

P= (Palabra | palabras previas)

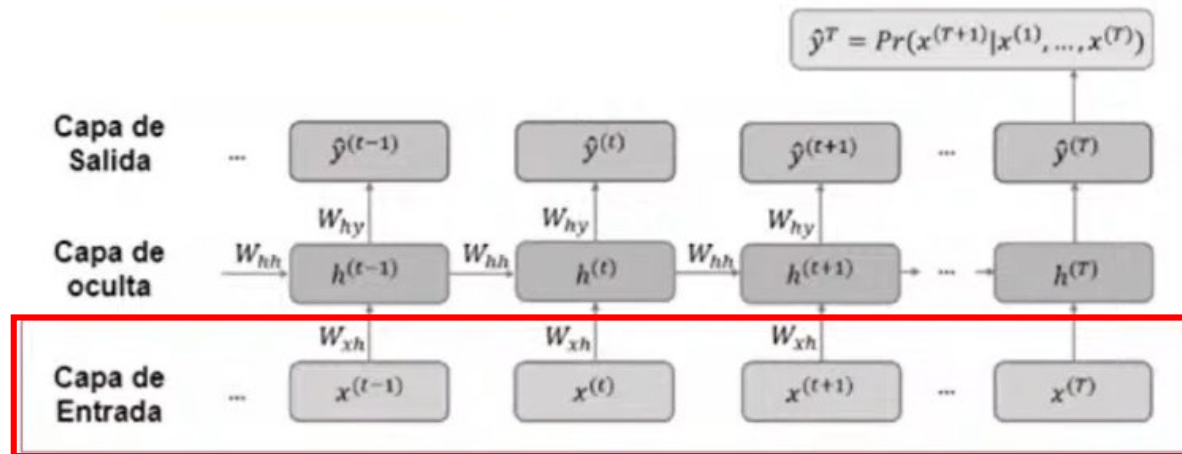
¿Por favor podrías venir ahora?

Información previa

Predicción

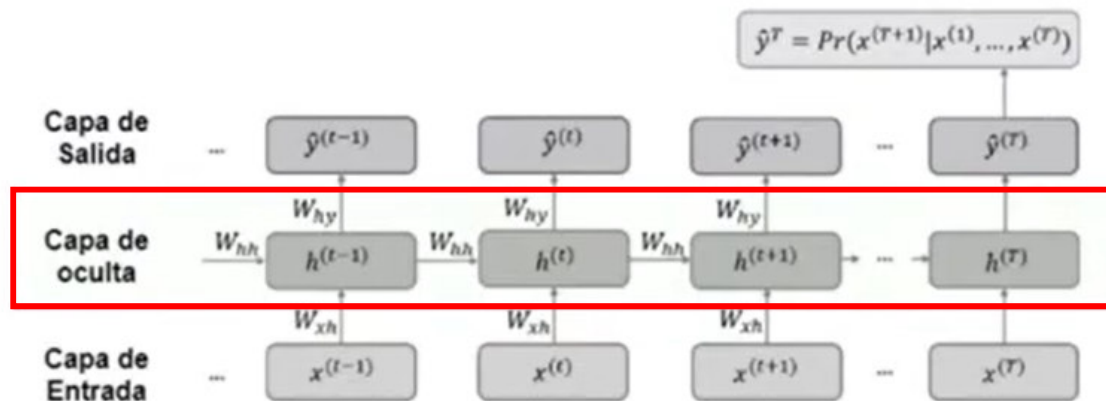
RNN Simple

Cada capa de entrada se corresponde con la información de entrada de la red
Las entradas deben codificarse mediante un vector de valores numéricos pudiendo ser one-hot o embeddings



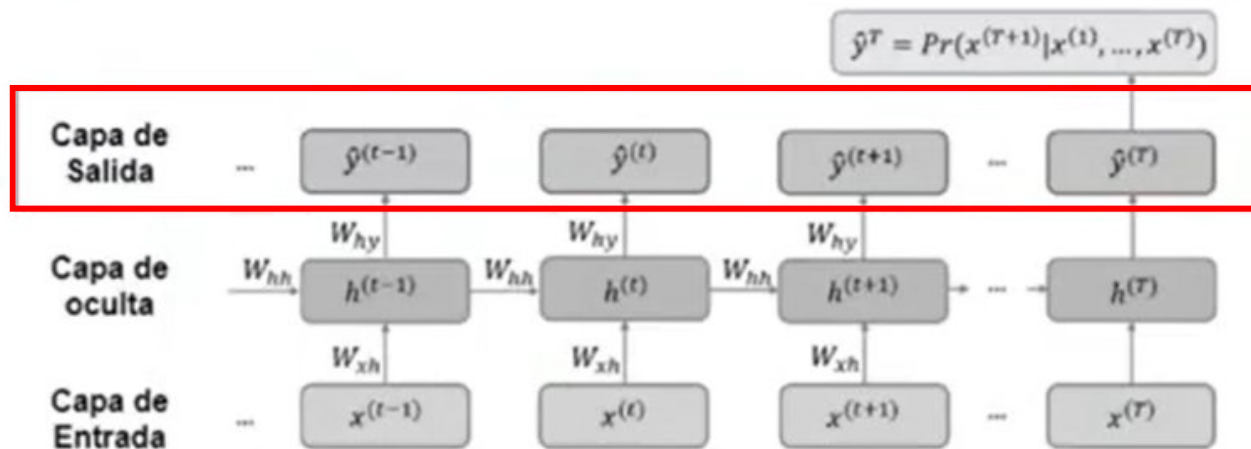
RNN Simple

Compuesta por un número finito de capas ocultas que corresponden a las capas intermedias de la red donde están los estados ocultos $h(t)$, que corresponden con la salida de la capa oculta que será usada como entrada de la capa siguiente y como entrada a la propia capa con el objetivo de tener memoria a corto plazo.



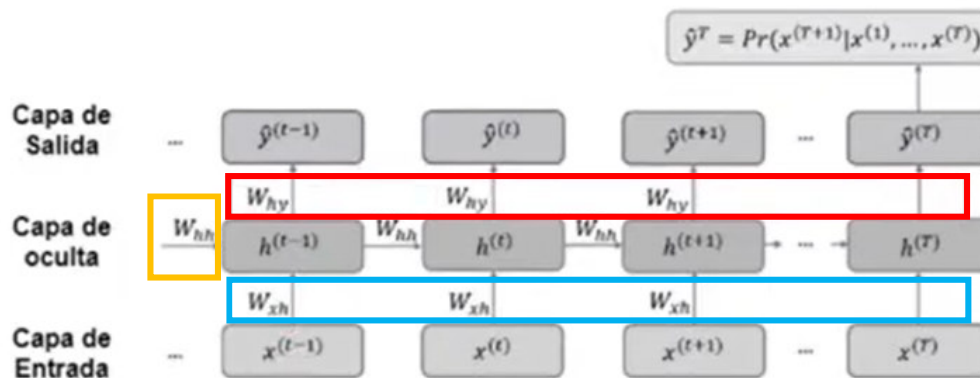
La entrada de las neuronas de la capa oculta combina la información de entrada $x(t)$ y el valor del estado oculto $h(t-1)$ que es la salida en el instante de tiempo inmediatamente anterior

RNN Simple



La capa de salida se utiliza para modelar una distribución de probabilidad para el instante de tiempo t

RNN Simple



Why Las funciones de activación de las diferentes capas ocultas se corresponden con funciones no lineales que normalmente son de tipo sigmoidal

Estas funciones dependen del tipo de salida a emplear, como las de procesos de clasificación

W_{xh} es la matriz de pesos que conecta las neuronas de la capa de entrada con las neuronas de la capa oculta

W_{hh} es la matriz de peso que conecta las neuronas de la capa oculta consigo mismas, simulando el proceso de memoria a corto plazo

RNN Simple

Una vez importada las clases necesarias y cargada la base de conocimiento para el entrenamiento de la red neuronal, se definen las capas y cantidad de neuronas por capas de la red

```
from keras.layers import SimpleRNN
from keras.models import Sequential

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

```
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
```

```
#Compile el modelo
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(X_train, y_train,
                   epochs=10,
                   batch_size=60,
                   validation_split=0.2)

#Para evaluar el modelo
score = model.evaluate(X_test, y_test, verbose=1)
```

RNN Simple

Ventajas

- Procesan entradas de orden secuencial, lo cual permite su uso en análisis de texto
- Procesan entradas (y salidas) de cualquier longitud, sin que estas longitudes afecten al tamaño del modelo
- Los pesos se comparten a lo largo del tiempo

Desventajas

- No consideran entradas futuras para el estado actual
- **Difícil acceso a información de estados muy antiguos.**
- **Desaparición/explosión del gradiente:** es complicado capturar las dependencias a largo plazo debido a la disminución/aumento exponencial del gradiente a medida que se crean más capas.

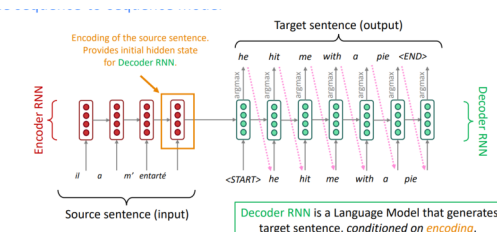
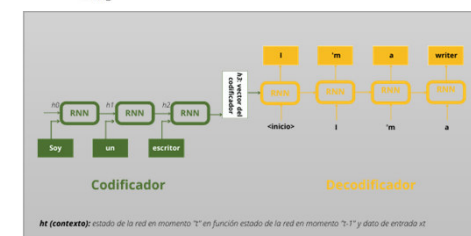
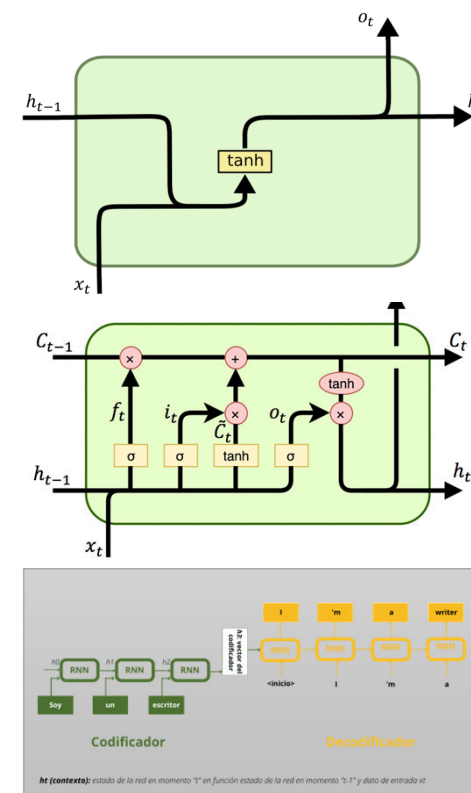
Redes de Neuronas Recurrentes

Las redes de neuronas RNN se clasifican en

- Redes recurrentes simples
- Redes recurrentes complejas

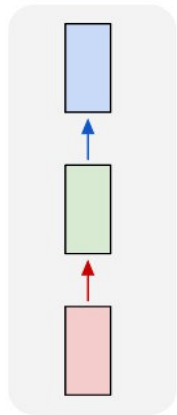
Existen modelos más complejos que utilizan combinaciones de redes recurrentes para resolver otros problemas: generación de texto, traducción

- Encoders – Decoders
- Modelos de atención
- Transformers



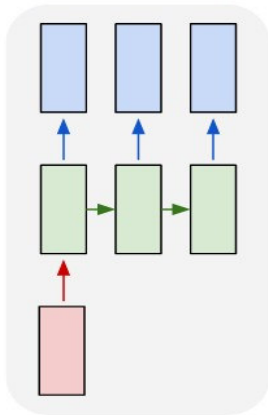
Arquitecturas RNN

one to one



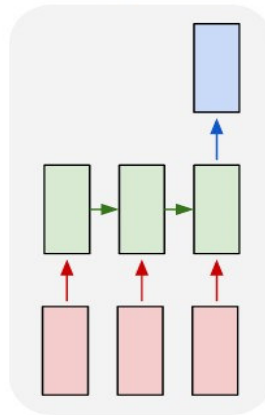
RNN simple
Tareas de
clasificación

one to many



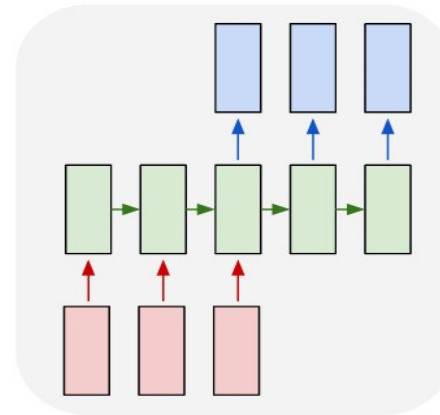
Etiquetado de imágenes,
Entrada:
imagen
Salida:
secuencia de
palabras

many to one



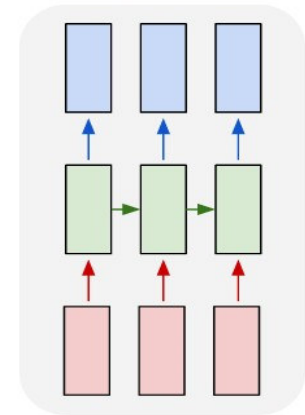
**Identificación del
sentimiento
(texto)**
Entrada: texto
Salida: valor
numérico entre 0 y
1 (negativo -
positivo)

many to many



**Simétrica: traducción
automática entre
lenguajes,**
Entrada: secuencia de
palabras
Salida: secuencia de
palabras en otro idioma.
NER

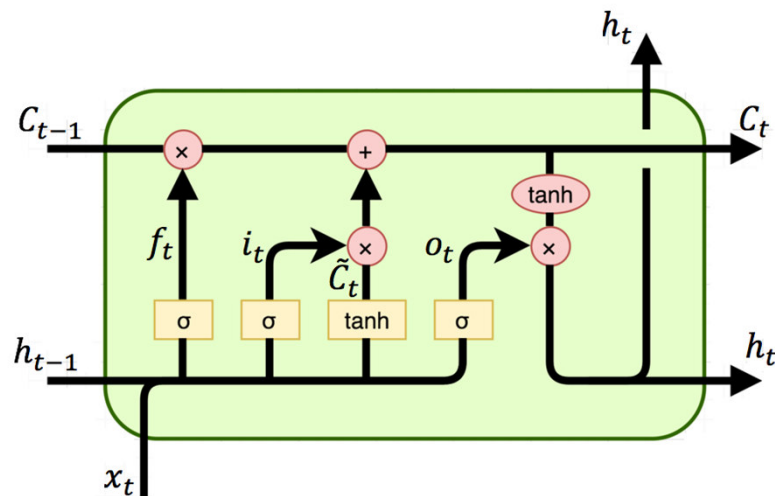
many to many



Sincronizado
Etiquetado de
video frame a
frame
Traducción en
tiempo real

Long Short Term Memory LSTM

Las neuronas LSTM combinan estados ocultos internos que permiten almacenar información simulando el proceso de estado a largo y corto plazo. Además, las unidades ocultas ahora se amplían mediante puertas especiales de entrada, olvido y salida, que ayudan a controlar el flujo de información

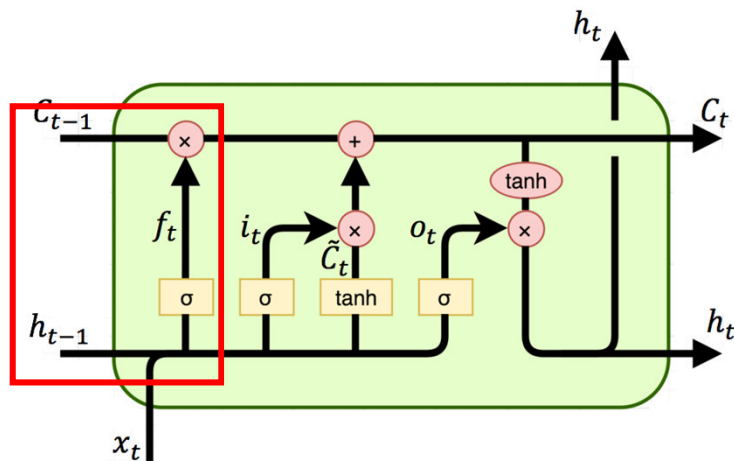


Entradas y salidas

- x_t : entrada en el tiempo t .
- h_{t-1} es la salida de la celda en tiempo $t-1$, la salida anterior.
- c_{t-1} : es la información (memoria) que es pasada por la etapa anterior.
- c_t es la información que es pasada a la siguiente etapa.

Long Short Term Memory LSTM

Las neuronas LSTM combinan estados ocultos internos que permiten almacenar información simulando el proceso de estado a largo y corto plazo. Además, las unidades ocultas ahora se amplían mediante puertas especiales de entrada, olvido y salida, que ayudan a controlar el flujo de información



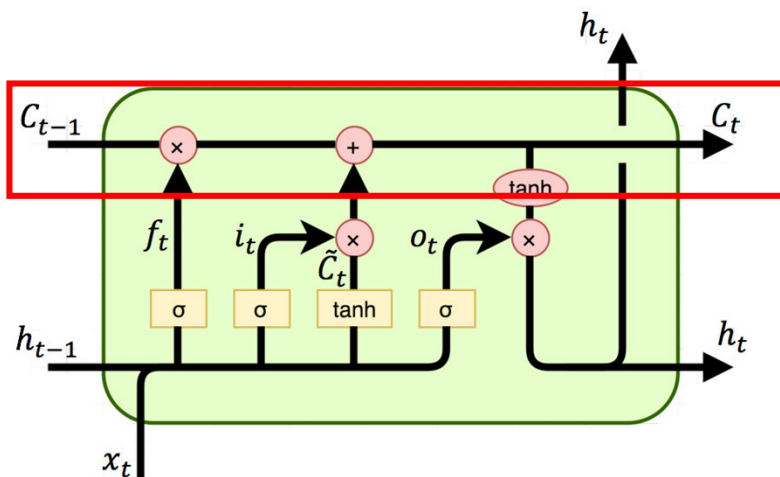
Puerta de olvido o *forget gate*

Esta puerta decide que información permanece y cual se olvida. Esto se consigue con la función sigmoide la cual tiene un dominio de 0 a 1. Cuanto más cerca del 0 menos importante es y cuanto más cerca del 1 más importante es.

$f(t)$ selecciona aquellos valores que corresponden con la salida anterior $h(t-1)$ para que sean olvidados

Long Short Term Memory LSTM

Las neuronas LSTM combinan estados ocultos internos que permiten almacenar información simulando el proceso de estado a largo y corto plazo. Además, las unidades ocultas ahora se amplían mediante puertas especiales de entrada, olvido y salida, que ayudan a controlar el flujo de información



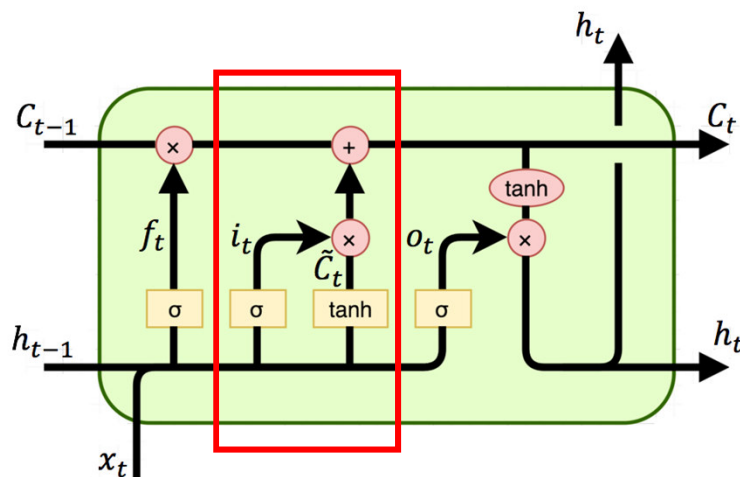
Celda de estado

La puerta de estado define que información será almacenada en la celda de estado interno $c(t)$ para usarse en las siguientes iteraciones

1. El estado previo $c(t-1)$ se multiplica por el valor $f(t)$ para olvidar información no útil

Long Short Term Memory LSTM

Las neuronas LSTM combinan estados ocultos internos que permiten almacenar información simulando el proceso de estado a largo y corto plazo. Además, las unidades ocultas ahora se amplían mediante puertas especiales de entrada, olvido y salida, que ayudan a controlar el flujo de información



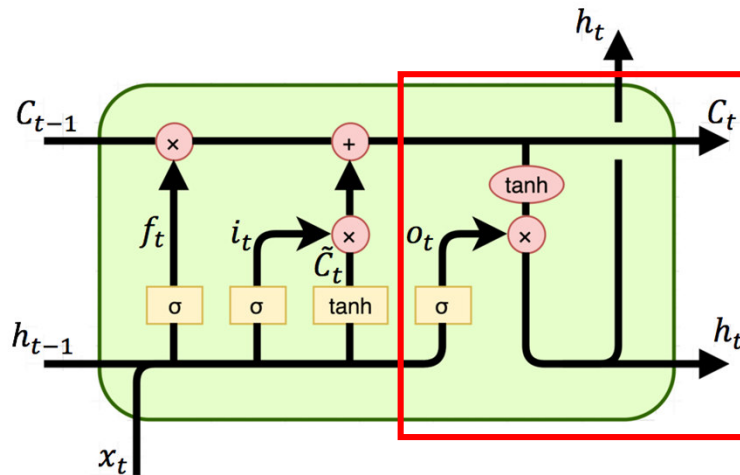
Puerta de estado

La puerta de estado define que información será almacenada en la celda de estado interno $c(t)$ para usarse en las siguientes iteraciones

1. El estado previo $c(t-1)$ se multiplica por el valor $f(t)$ para olvidar información no útil
2. El nuevo estado se actualiza multiplicándose con la información obtenida por la puerta de entrada a $i(t)$ y la salida anterior para añadir la nueva información de entrada

Long Short Term Memory LSTM

Las neuronas LSTM combinan estados ocultos internos que permiten almacenar información simulando el proceso de estado a largo y corto plazo. Además, las unidades ocultas ahora se amplían mediante puertas especiales de entrada, olvido y salida, que ayudan a controlar el flujo de información



Puerta de salida

La salida $h(t)$ es calculada mediante la utilización de la información almacenada en la puerta de salida $o(t)$ y que combina la entrada y el estado oculto anterior y el nuevo estado oculto

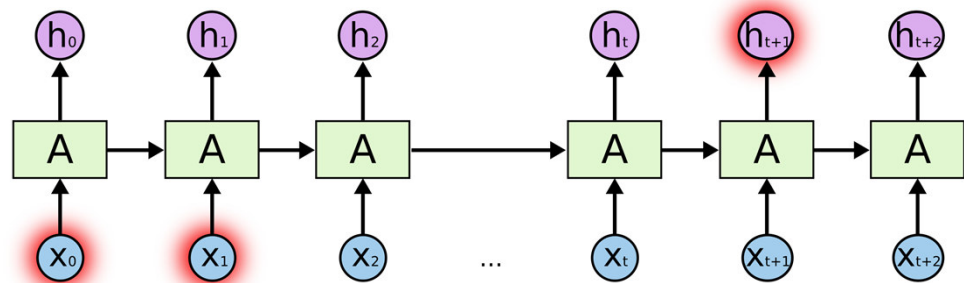
Long Short Term Memory LSTM

Ventaja

LSTM resuelve el problema de memoria a largo plazo

Para RNN es difícil acceder información de estados muy antiguos.

Un gran contexto para predecir, las RNN no son muy eficientes.

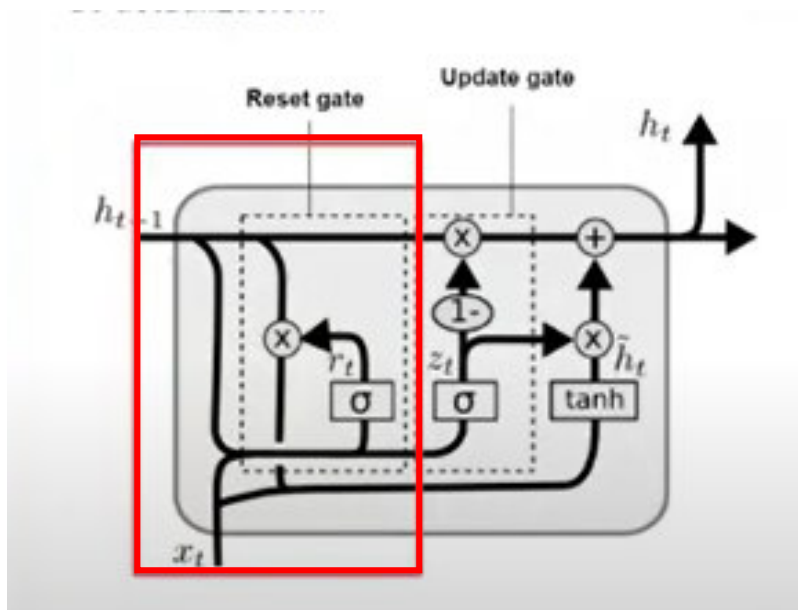


Desventaja

LSTM requiere mayor capacidad computacional
y mayor tiempo de entrenamiento

Gated Recurrent Units GRU

Las neuronas recurrentes de tipo cerrado (GRU) son neuronas de tipo recurrente con un comportamiento similar a las neuronas LSTM pero con una estructura mucho más simple formada solo por dos puertas una de reinicio y otra de actualización

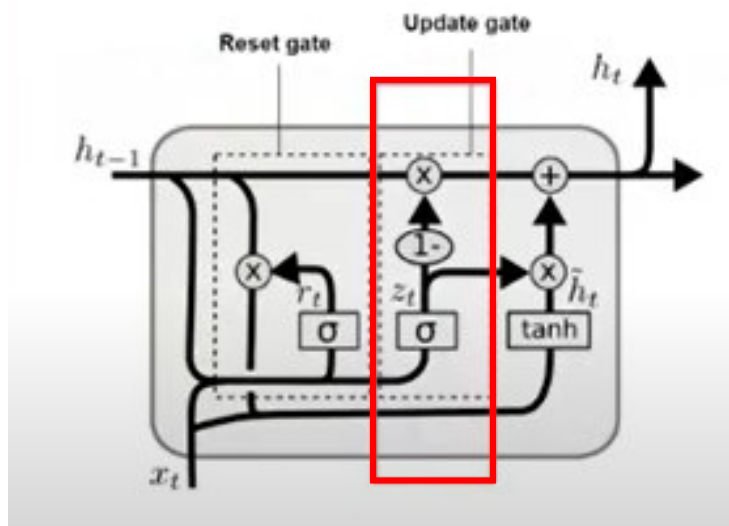


Puerta de Reinicio (Reset gate)

La puerta de reinicio $r(t)$ define cual es el nivel de reinicio del estado oculto previo $h(t-1)$. De manera que si el valor de $r(t)$ es muy cercano a cero, la información del estado oculto es prácticamente eliminada y reseteado con la información de la entrada $x(t)$

Gated Recurrent Units GRU

Las neuronas recurrentes de tipo cerrado (GRU) son neuronas de tipo recurrente con un comportamiento similar a las neuronas LSTM pero con una estructura mucho más simple formada solo por dos puertas una de reinicio y otra de actualización

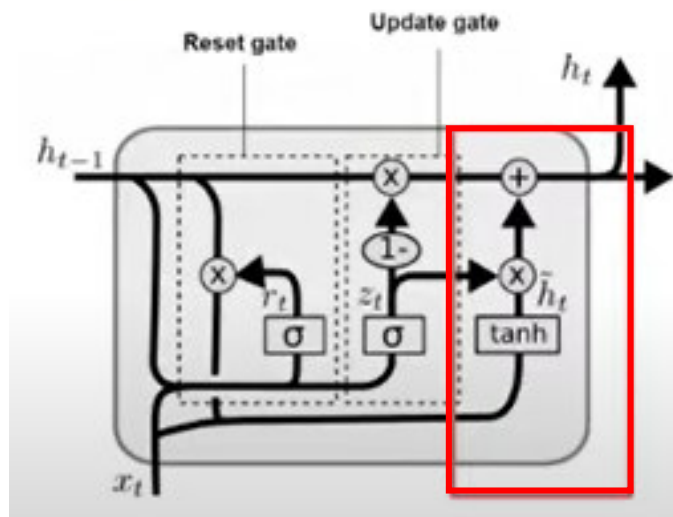


Puerta de Actualización (Update gate)

En la puerta de actualización se calcula el nuevo valor del estado oculto mediante la aplicación de tres operaciones, que combina la información de entrada, el antiguo valor oculto

Gated Recurrent Units GRU

Las neuronas recurrentes de tipo cerrado (GRU) son neuronas de tipo recurrente con un comportamiento similar a las neuronas LSTM pero con una estructura mucho más simple formada solo por dos puertas una de reinicio y otra de actualización



Puerta de salida

El nuevo valor de salida se calcula mediante el resultado de la celda de actualización $z(t)$, el valor anterior actualizado $h(t-1)$ y lo combina con la entrada y con el nivel de reinicio calculado previamente

Gated Recurrent Units GRU

Ventajas

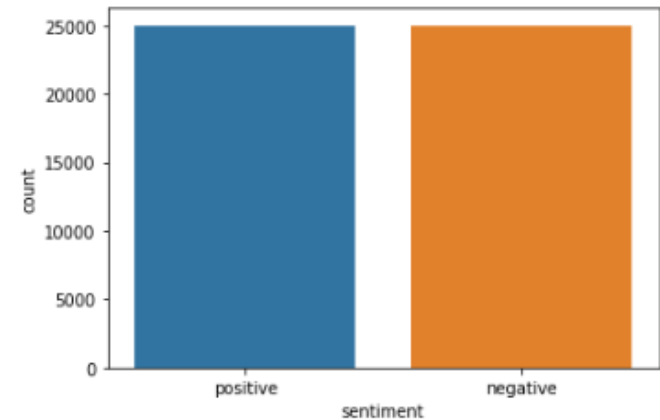
GRU ayuda a resolver los problemas de desaparición de gradiente a través de:

- La agregación de dos puertas: **Update** y **Reset**
- Mantienen un registro de la información más relevante para la predicción. Básicamente, son dos vectores que deciden qué información se pasa a la salida.
 - La **puerta de Update** determina **cuánta información** previa se debe transmitir **al siguiente paso** de tiempo.
 - La **puerta de Reset** determina **cuánta información** es irrelevante y **debe olvidarse**.

Ejemplo

Vamos a trabajar con el caso del corpus de reseñas de películas IMDB

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive



Vamos a clasificar el sentimiento utilizando los siguientes modelos neuronales

1. Red neuronal
2. LSTM (Long Short Term Memory)
3. GRU

Adicionalmente, se da como tarea adicional usar un RNN

Red Neuronal

La Red neuronal del primer caso tiene las siguiente arquitectura e hiperparámetros:

```
model = Sequential()
embedding_layer = Embedding(vocab_size,100,weights=[embedding_matrix],
                           input_length=maxlen,trainable=False)
model.add(embedding_layer)
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

#Compilar el modelo usando el optimizador Adam
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=['acc'])

print(model.summary())
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 100)	9254700
flatten (Flatten)	(None, 10000)	0
dense (Dense)	(None, 1)	10001

```
=====
Total params: 9,264,701
Trainable params: 10,001
Non-trainable params: 9,254,700
```

```
#Entrenar el modelo
history = model.fit(X_train, y_train,
                   batch_size=128,
                   epochs=6,
                   verbose=1,
                   validation_split=0.2)
```

[Simulador de configuración](#)

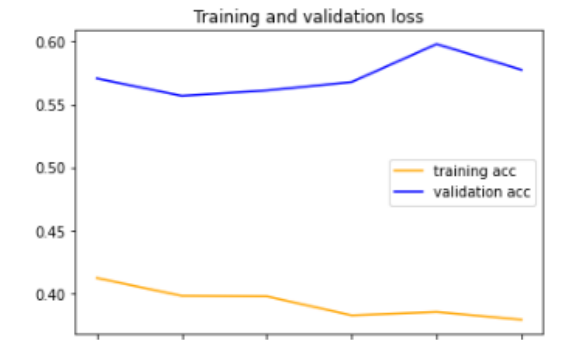
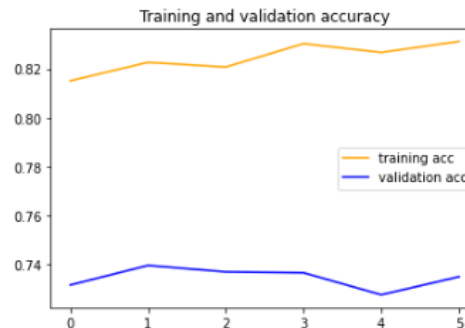
Red Neuronal

Los resultados obtenidos por el modelo son:

1. Accuracy **0.73**
2. Loss **0.59**

Agregando dos capas densas de 12 y 8 neuronas

1. Accuracy **0.72**
2. Loss **0.66**



Incrementando las neuronas de las dos capas a 32 no hay mejoría

CONCLUSIÓN:

Underfitting (o sub ajuste): cuando el modelo es incapaz de obtener resultados correctos por falta de entrenamiento o de más muestras. Se reconoce visualmente cuando existe altos valores de pérdida tanto en el set de entrenamiento como en el de validación.

Red Neuronal LSTM

La Red neuronal LSTM tiene las siguiente arquitectura e hiperparámetros:

```
from keras.layers import LSTM
model = Sequential()
embedding_layer = Embedding(vocab_size, 100,
                            weights=[embedding_matrix],
                            input_length=maxlen,
                            trainable=False)
model.add(embedding_layer)
model.add(LSTM(128))
model.add(Dense(1, activation='sigmoid'))
#Compilar el modelo
model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=['acc'])
print(model.summary())
```

Model: "sequential_9"

Layer (type)	Output Shape	Param #
embedding_9 (Embedding)	(None, 100, 100)	9254700
lstm_1 (LSTM)	(None, 128)	117248
dense_21 (Dense)	(None, 1)	129

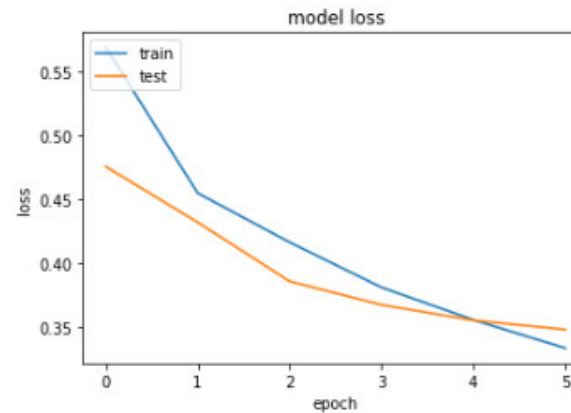
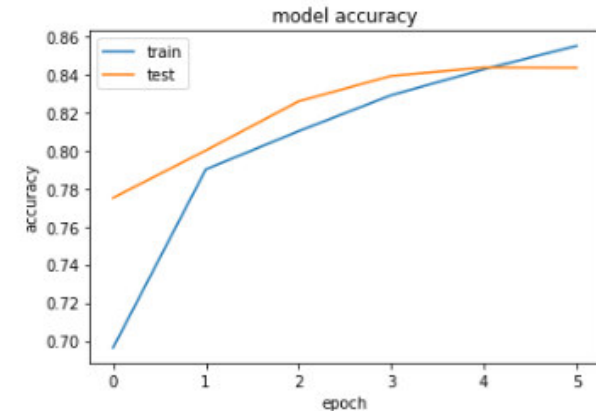
```
=====
Total params: 9,372,077
Trainable params: 117,377
Non-trainable params: 9,254,700
```

```
#entrenar el modelo en el conjunto de entrenamiento
history = model.fit(X_train, y_train,
                    batch_size=128,
                    epochs=6,
                    verbose=1,
                    validation_split=0.2)
#evalua desempeño en el conjunto de prueba.
score = model.evaluate(X_test, y_test, verbose=1)
```


Red Neuronal LSTM

Los resultados obtenidos por el modelo son:

1. Train Accuracy **0.86**
2. Train Loss **0.34**
3. **Validation Accuracy 0.85**
4. **Validation Loss 0.34**



Dropout?

•Optimal fit: cuando el modelo logra captar el comportamiento general de los datos. Se puede identificar al obtener valores de pérdidas en los datos de validación que no difieren demasiado de las pérdidas de los datos de entrenamiento.

Red Neuronal GRU

La Red neuronal GRU tiene la siguiente arquitectura e hiperparámetros:

```
from keras.models import Sequential
#from keras import layers
from keras.layers import Dense, GRU, Embedding
model = Sequential()
# Capa embedding
# input_dim : tamaño del vocabulario
# output_dim: dimensión del vector al que se mapea
model.add(Embedding(vocab_size, 100,
                    weights=[embedding_matrix],
                    input_length=maxlen,
                    trainable=False))
# comentar la siguiente línea para evaluar dropout
model.add(GRU(32))
# descomentar la siguiente línea para evaluar dropout
#model.add(GRU(32, dropout=.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
print(model.summary())
```

Model: "sequential_10"

Layer (type)	Output Shape	Param #
embedding_10 (Embedding)	(None, 100, 100)	9254700
gru_1 (GRU)	(None, 32)	12864
dense_22 (Dense)	(None, 1)	33

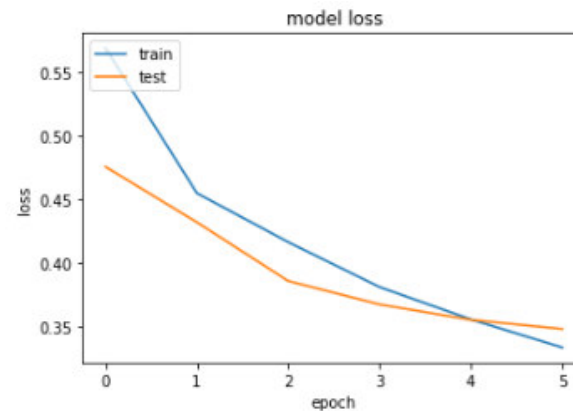
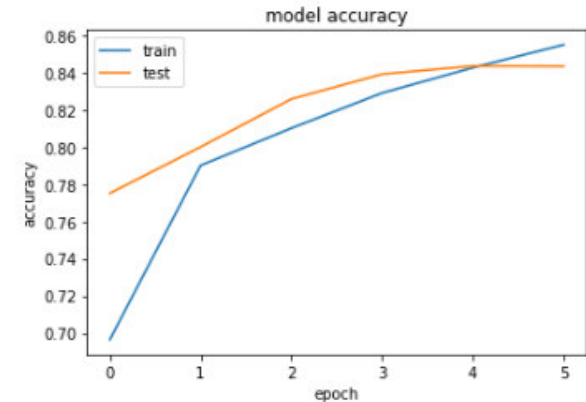
```
history_GRU = model.fit(input_train, y_train,
                        epochs=10,
                        batch_size=128,
                        validation_split=0.2,
                        verbose=2)
```

Red Neuronal GRU

Los resultados obtenidos por el modelo son:

1. Train Accuracy **0.86**
2. Train Loss **0.32**
3. Validation Accuracy **0.85**
4. Validation Loss **0.34**

¿Con Dropout?



•Optimal fit: cuando el modelo logra captar el comportamiento general de los datos. Se puede identificar al obtener valores de pérdidas en los datos de validación que no difieren demasiado de las pérdidas de los datos de entrenamiento.

<https://bth.diva-portal.org/smash/get/diva2:1454870/FULLTEXT02.pdf>

Consideraciones finales

1. Las redes neuronales NN no son adecuadas para la clasificación de las reseñas de IMDB, se observa que no se realiza ningún aprendizaje, ya que las NN no son capaces de utilizar la información del contexto de la entrada para realizar la clasificación
2. En cambio, las redes LSTM y GRU presentan mejores ventajas para la clasificación de las reseñas de películas que la red neuronal NN, por la característica de recordar el contexto previo.
3. El proceso de entrenamiento de un modelo neuronal, requiere de la definición adecuada de hiperparámetros, las funciones de activación de las capas ocultas y la capa de salida, el optimizador y la función de pérdida o error y para ello se realiza una ejercicio de prueba de acierto de combinaciones entre estos elementos

