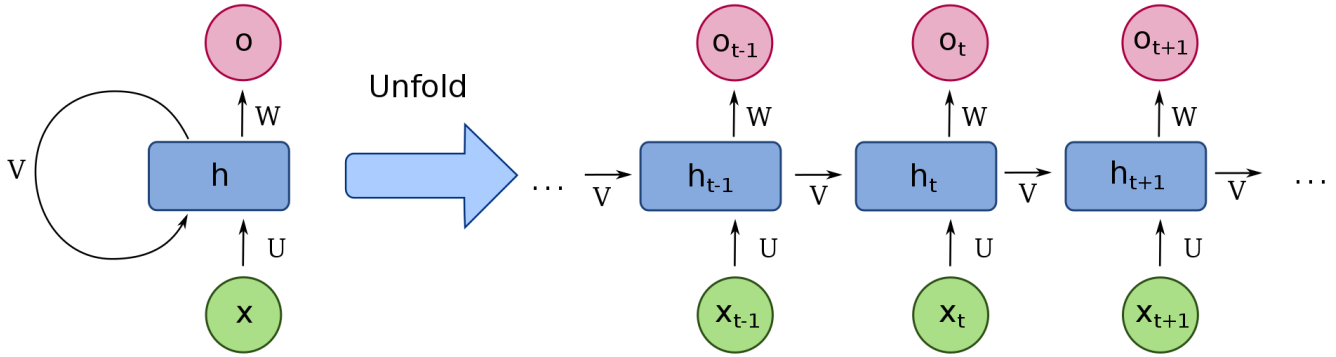
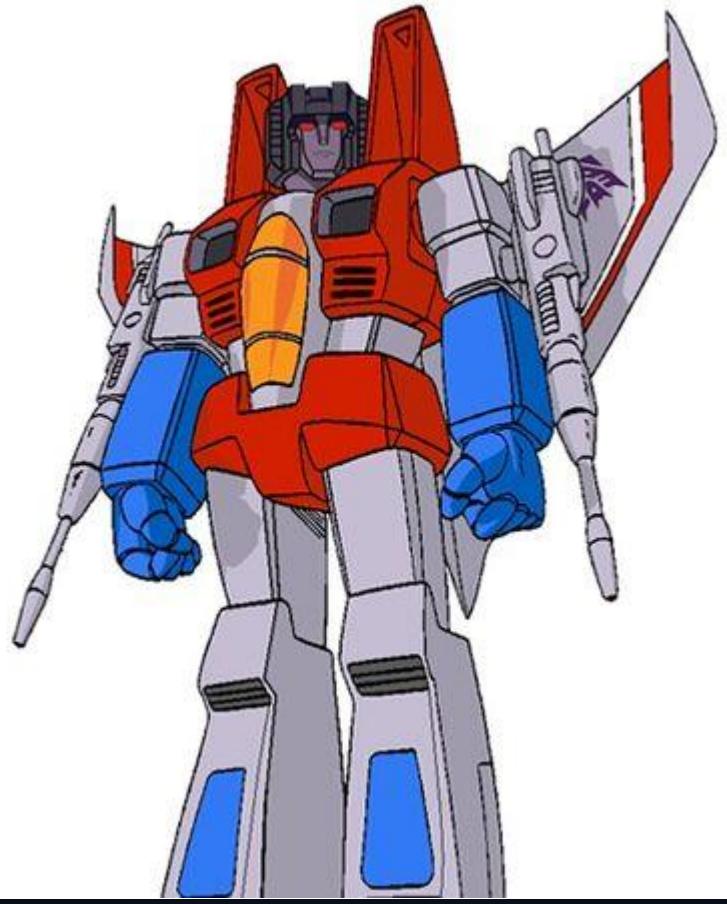
Redes Neuronales

LSTM, GRU, Bidireccioneales. Attention y Transformers.

Agenda

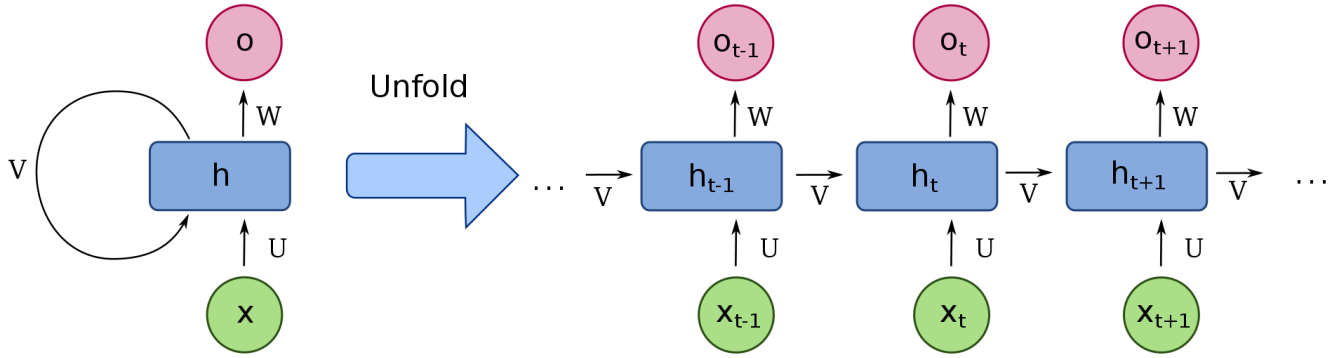
- Problemas recurrentes.
- Redes LSTM, GRU y bidereccionales.
- Attention Is All You Need (it is actually the paper title: https://arxiv.org/abs/1706.03762)
- Transformers





Redes recurrentes

- Las redes recurrentes son buenas cuando nuestras intancias muestran un comportamiento “temporal”
- El texto es un buen ejemplo del problema: el sentido de la oración no viene dado solo por las palabras, sino también por el orden.
- Hasta ahora, perdíamos esa información.

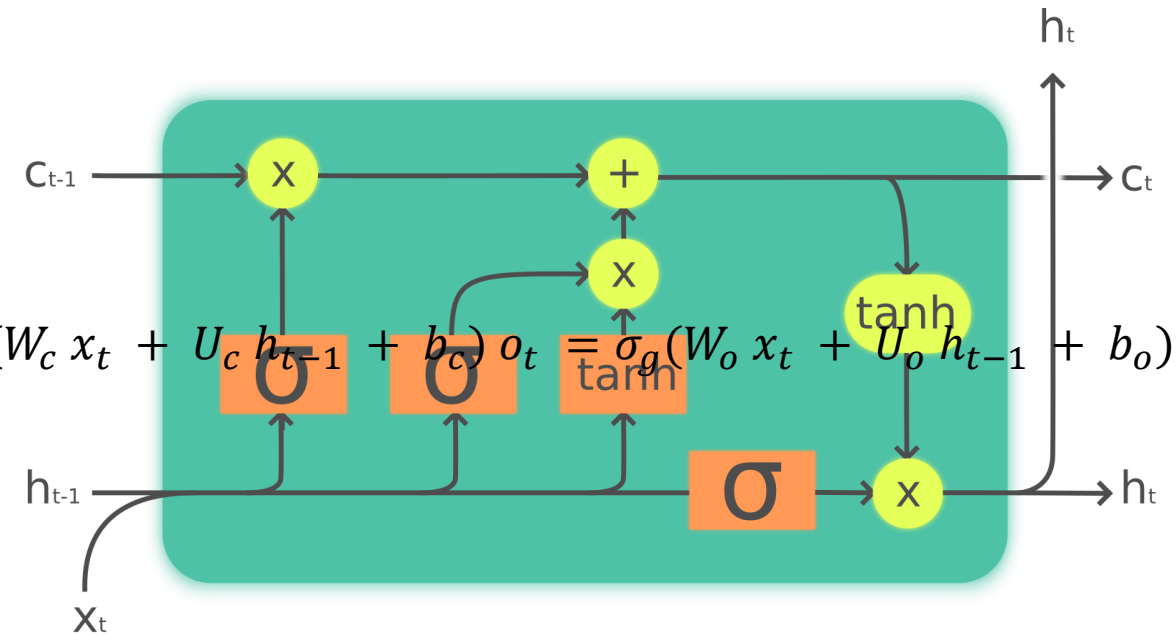


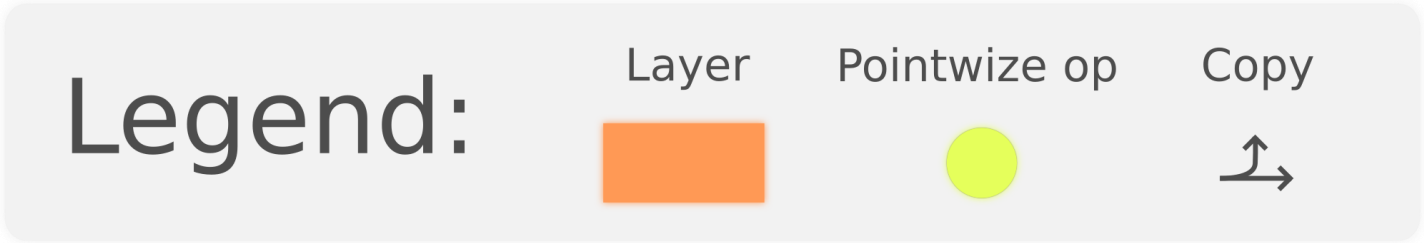
Redes Recurrentes: LSTM-GRU

- LSTM y GRU son redes neuronales recurrentes que resuelven el problema del exploding gradiente. Este problema se da porque los pesos son aplicados de forma repetidas.
- Lo hacen a través de algo llamado “forget gate” y “update gate”

GRU

$$h_t = o_t \circ \sigma_h(c_t) c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \; \tilde{c}_t = \sigma_h(W_c x_t + U_c h_{t-1} + b_c) \; o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \; i_t = \sigma_g($$

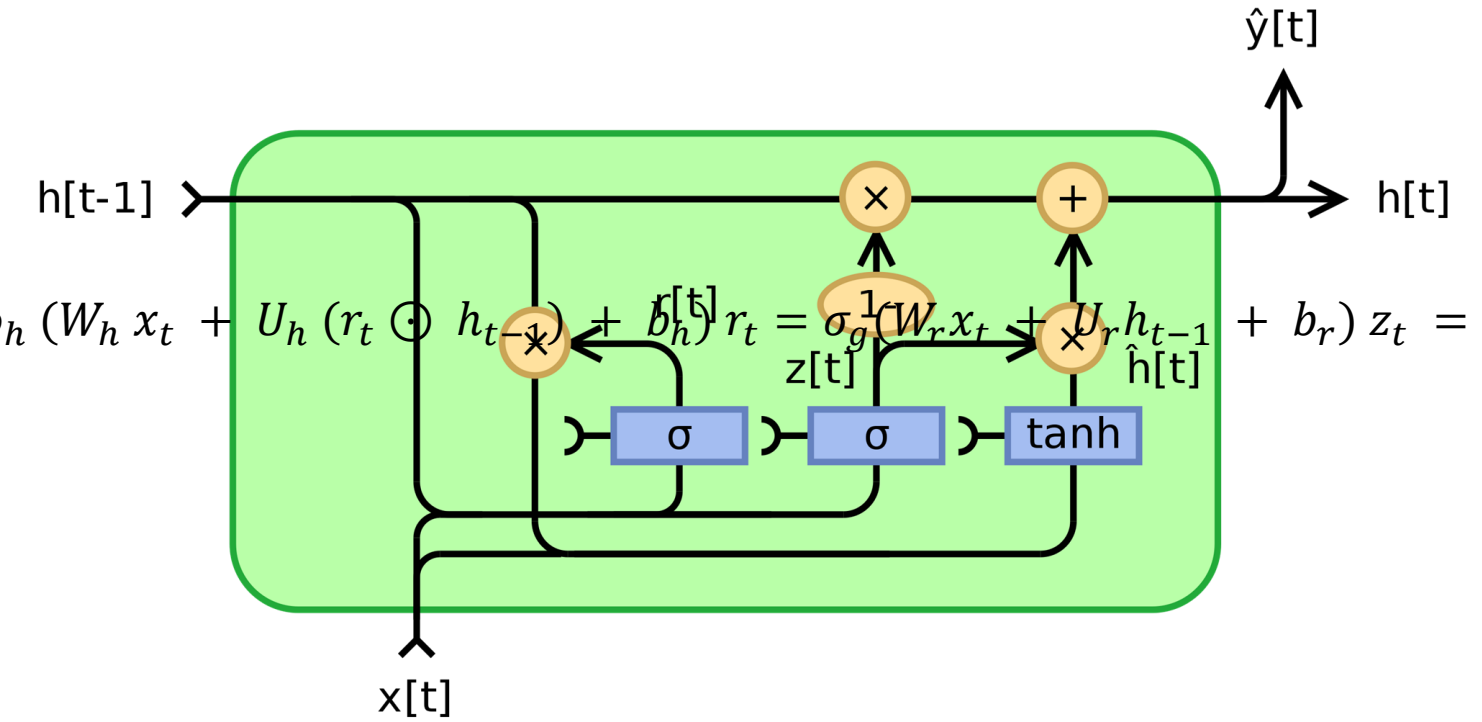




https://en.wikipedia.org/wiki/Long_short-term_memory

GRU

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t \hat{h}_t \& = \phi_h (W_h x_t + U_h (r_t \odot h_{t-1}) + b_h) r_t = \sigma_g (W_r x_t + U_r h_{t-1} + b_r) z_t = \sigma$$



https://en.wikipedia.org/wiki/Long_short-term_memory

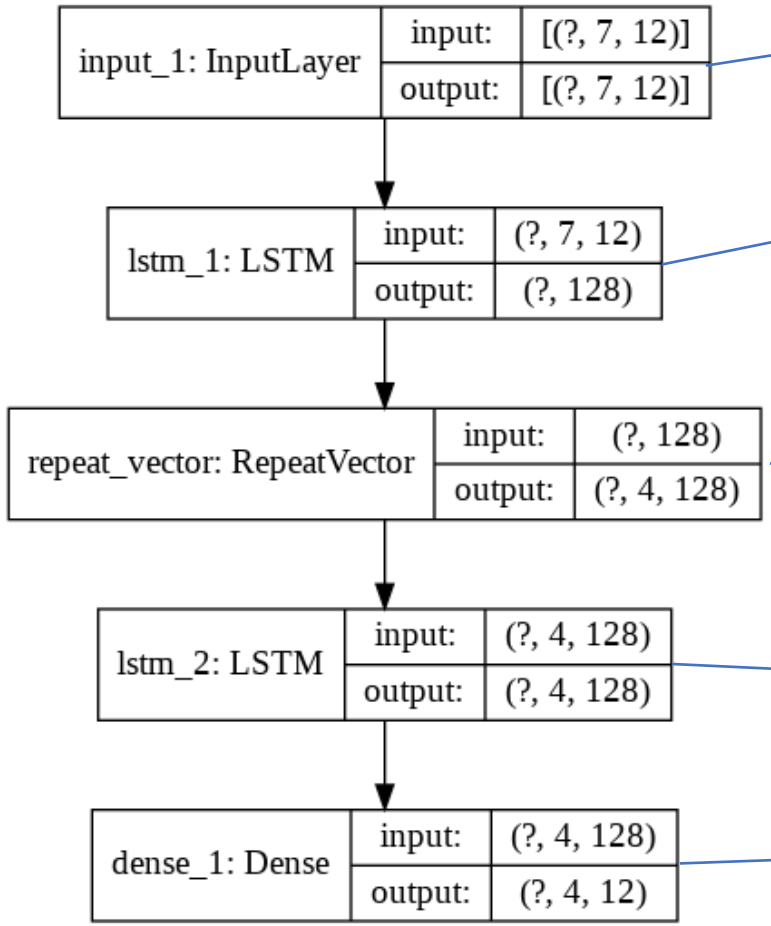
LSTM Ejemplo: Sequence to Sequence

- Supongamos que tenemos un dataset con operaciones de suma:
  - 12+3 -> 15
  - 129+124 -> 253
- Las sumas y los resultados están expresados como cadenas de caracteres.
- El objetivo es a partir de la cadena de caracteres, encontrar el valor de la suma.

Representación del problema

- Caracteres válidos (12 en total): 0, 1, 2, …, 9, +, “ “.
- Entrada suma de números de hasta 3 cifras. Se necesitan 7 caracteres máximo.
- Los caracteres se pueden representar con representación one-hot.
- El conjunto de entrenamiento se puede representar con matrices de 3 dimensiones: instancias X pasos de tiempo X características (caracteres codificados en one-hot en este caso).
- La salida de la red puede ser hasta 4 caracteres, en este caso, se representa la distribución de probabilidad para cada carácter. El conjunto de salida se representa como: instancias X tamaño resultante X caracteres.
- En caso de que las secuencias sean más cortas, se completa con el carácter “ “.

Sequence to Sequence



- Entrada
- LSTM: nos quedamos con la última salida que códifica la secuencia original
- Repetimos la codificación para generar la nueva secuencia
- LSTM: genera la nueva secuencia
- Densa: decodifica la secuencia y aplica softmax sobre la última dimensión

Sequence to Sequence

```
i = Input((MAXLEN, len(chars)))
d = LSTM(128, return_sequences=False)(i)
d = RepeatVector(DIGITS+1)(d)
d = LSTM(128, return_sequences=True)(d)
d = Dense(len(chars), activation='softmax')(d)

model = Model(i, d)
model.compile(loss="categorical_crossentropy",
optimizer="adam", metrics=["accuracy"])
model.summary()
```

Sequence to Sequence

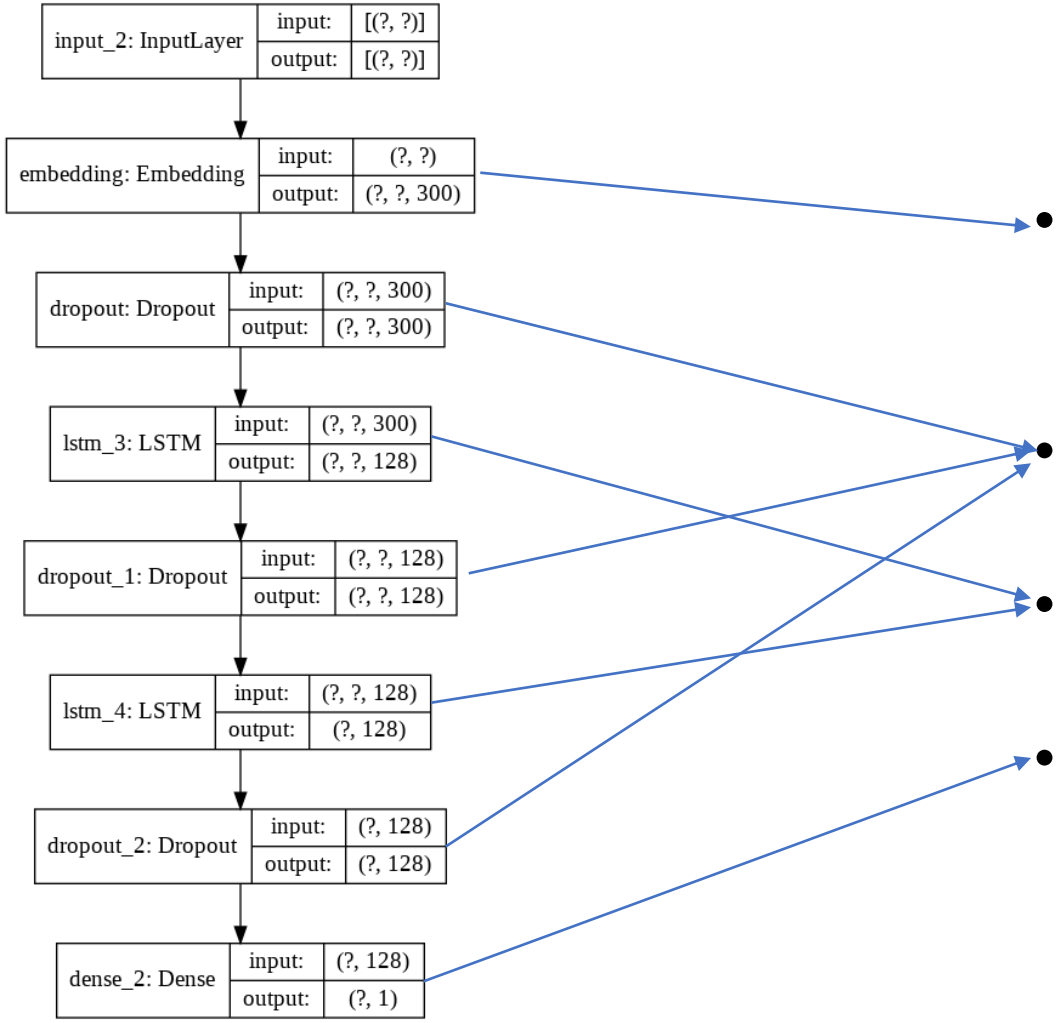
```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 7, 12)]           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               72192
_________________________________________________________________
repeat_vector (RepeatVector) (None, 4, 128)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 4, 128)            131584
_________________________________________________________________
dense_1 (Dense)              (None, 4, 12)             1548
=================================================================
Total params: 205,324
Trainable params: 205,324
Non-trainable params: 0
_________________________________________________________________
```

Clasificación de texto

Las redes recurrentes (LSTM/GRU) pueden utilizarse para clasificar texto.

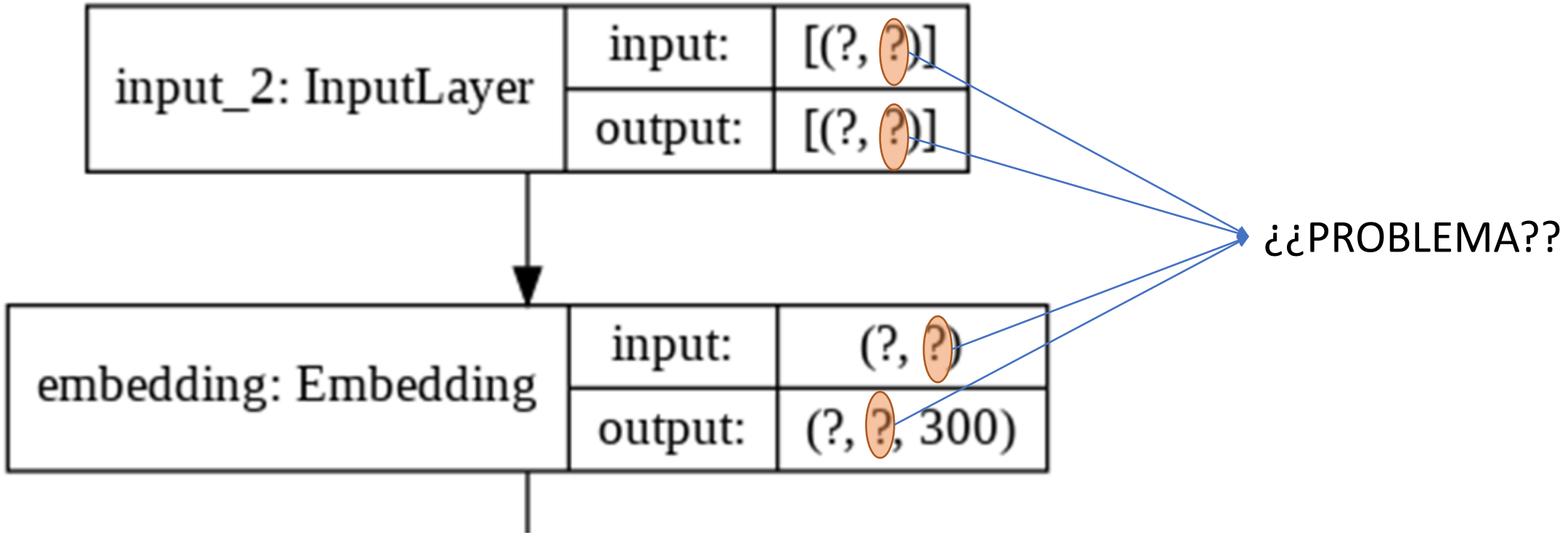
- Se suele utilizar una capa de embedding para facilitar la representación de texto.
- Se suelen usar una o dos capas recurrentes. Más capas pueden tener problemas de “vanishing gradient”.
- Se puede integrar dropout dentro de las LSTM para evitar el overfitting interno.

Clasificación de texto



- Embedding: mapea una secuencia de id a sus correspondientes vectores.
- Dropout: previenen overfitting.
- LSTM: Procesan la secuencia.
- Dense: Realiza la clasificación

Clasificación de texto



Clasificación de texto

Cuando no conocemos el tamaño de las secuencias de entrada tenemos dos opciones:

- Padding: la más usada. Consiste en quedarse con los últimos N elementos de la secuencia. Si la secuencia es más corta que N, se completan los elementos con ceros.

- Bucketing: Se generan los mini-batchs con todas las secuencias del mismo tamaño. Pueden haber batchs de distintos tamaños. Requiere implementar un Sequence en keras.

Clasificación de texto

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
MAXLEN = 50

x_train = pad_sequences(x_train, maxlen=MAXLEN)
x_test = pad_sequences(x_test, maxlen=MAXLEN)
```

Clasificación de texto

```
from tensorflow.keras.layers import Input, LSTM, Dropout, Dense, Embedding
from tensorflow.keras.models import Model

i = Input((None,))
d = Embedding(len(words_id) + 1, 300, mask_zero=True)(i)
d = Dropout(0.5)(d)
d = LSTM(128, return_sequences=True)(d)
d = Dropout(0.5)(d)
d = LSTM(128, return_sequences=False)(d)
d = Dropout(0.5)(d)
d = Dense(1, activation='sigmoid')(d)

model = Model(i, d)
model.summary()
model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['bin
ary_accuracy'])
```

Clasificación de texto: Embeddings preentrenados

```
emb = Embedding(len(words_id) + 1, 300, mask_zero=True, trainable=False)
i = Input((None,))
d = emb(i)
d = Dropout(0.5)(d)
d = LSTM(128, return_sequences=True)(d)
d = Dropout(0.5)(d)
d = LSTM(128, return_sequences=False)(d)
d = Dropout(0.5)(d)
d = Dense(1, activation='sigmoid')(d)

model = Model(i, d)
```

Clasificación de texto: Embeddings preentrenados

```
base_emb = np.zeros(emb.embeddings.shape)

for w, i in words_id.items():
    if w in glove:
        base_emb[i, :] = (glove[w] / np.sum(glove[w]))

K.set_value(emb.embeddings, base_emb)
```

Clasificación de texto: Embeddings preentrenados

- Puede reducir el tiempo de entrenamiento.
- Permite utilizar vocabulario que no está en el conjunto de entrenamiento. En otro caso, se puede eliminar o utilizar un valor <unk>.
- Los embeddings pueden no ajustarse a la tarea.
- Los embeddings pueden dejarse fijos o entrenarse.

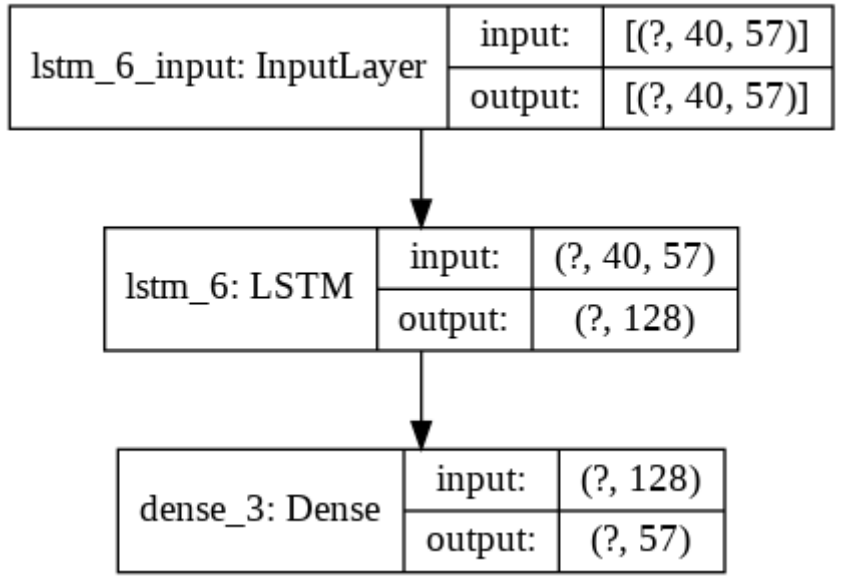
Generación de texto

- En el ejemplo generaremos texto aleatorio carácter a carácter. (Se puede hacer a nivel palabra pero requiere más entrenamiento)

- Trataremos como problema de clasificación: “Dado una secuencia de caracteres cual es el siguiente”.

Generación de texto

```
i = Input((maxlen, len(chars)))
d = (LSTM(128)(i)
d = Dense(len(chars), activation='softmax')(d)

model = Model(i, d)

optimizer = 'adam'
model.compile(loss='categorical_crossentropy',
 optimizer=optimizer, metrics=['accuracy'])
```



Generación de texto: Generación

```
generated = ''
sentence = text[start_index: start_index + maxlen]
generated += sentence

for i in range(400):
    x_pred = np.zeros((1, maxlen, len(chars)))
    for t, char in enumerate(sentence):
       x_pred[0, t, char_indices[char]] = 1.

    preds = model.predict(x_pred, verbose=0)[0]
    next_index = sample(preds, diversity)
    next_char = indices_char[next_index]

    generated += next_char
    sentence = sentence[1:] + next_char
```

Generación de texto: Generación

```
def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
```

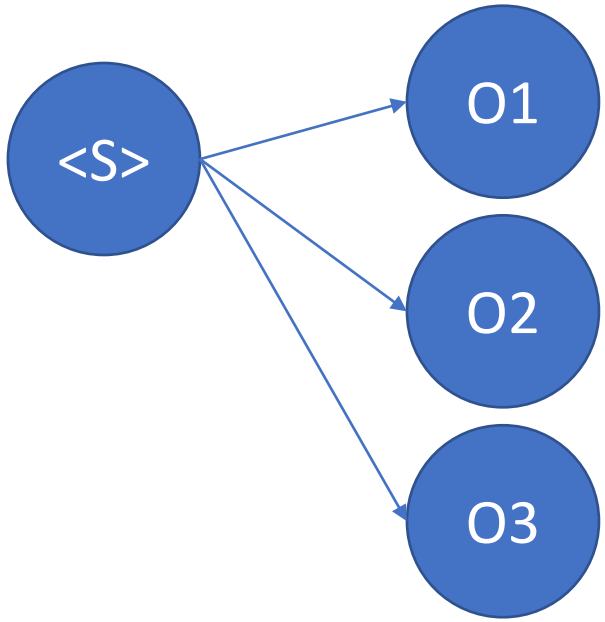
Generación de texto: Generación

```
----- Generating with seed: "ve--is not that just endeavouring to be “
 ve--is not that just endeavouring to be the moral nature the fark which
may a therath and have to the condicianing and long the most is a there
is them there is respless of the spority, there is to the spirit of the
contient deed is not is there in the same--which may in the conterned to
the for themselves as a reand of also in innorment in the strangly and
the some deverys and stand of the them about and and they are soul the
present t
```

Beam Search

Otra técnica utilizada para la generación de secuencias es el beam search.
En vez de gererar aleatoriamente se va generando las “más” probables:

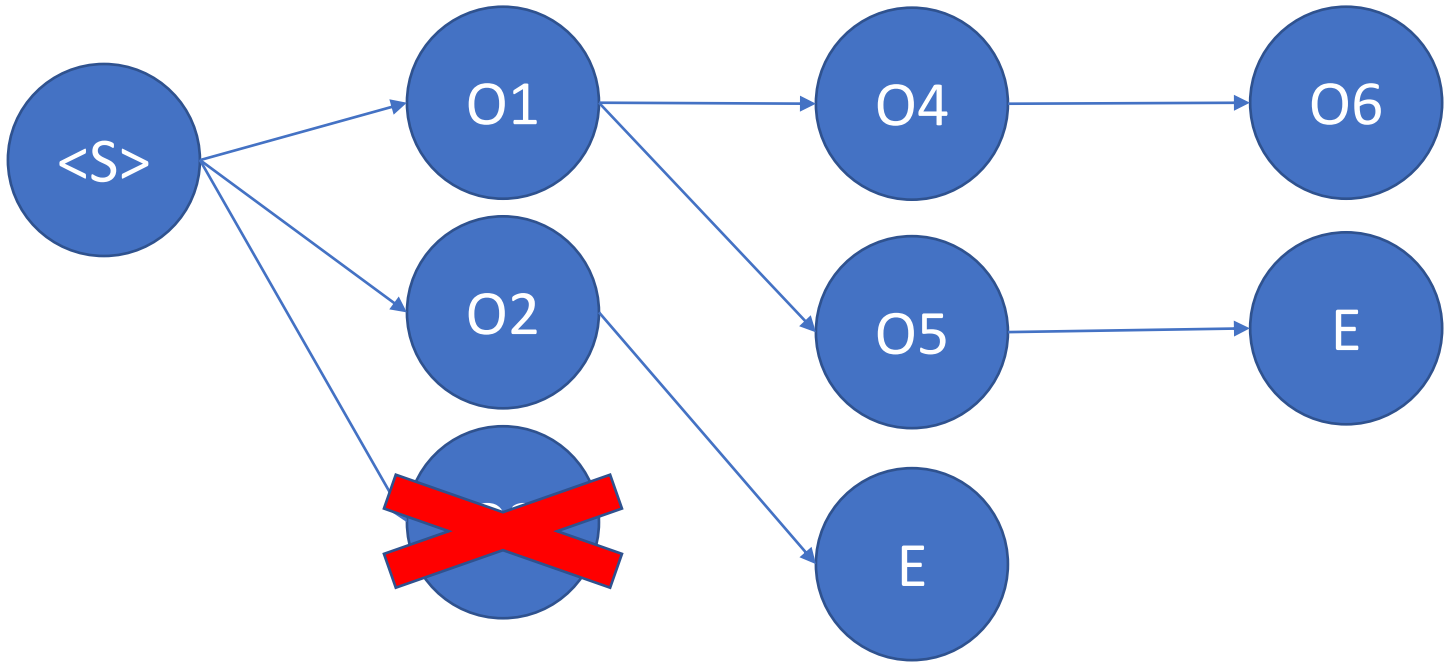
- Tokens especiales: <s> inicio de sentencia, <e> fin de sentencia.



Beam Search

Otra técnica utilizada para la generación de secuencias es el beam search.
En vez de gererar aleatoriamente se va generando las “más” probables:

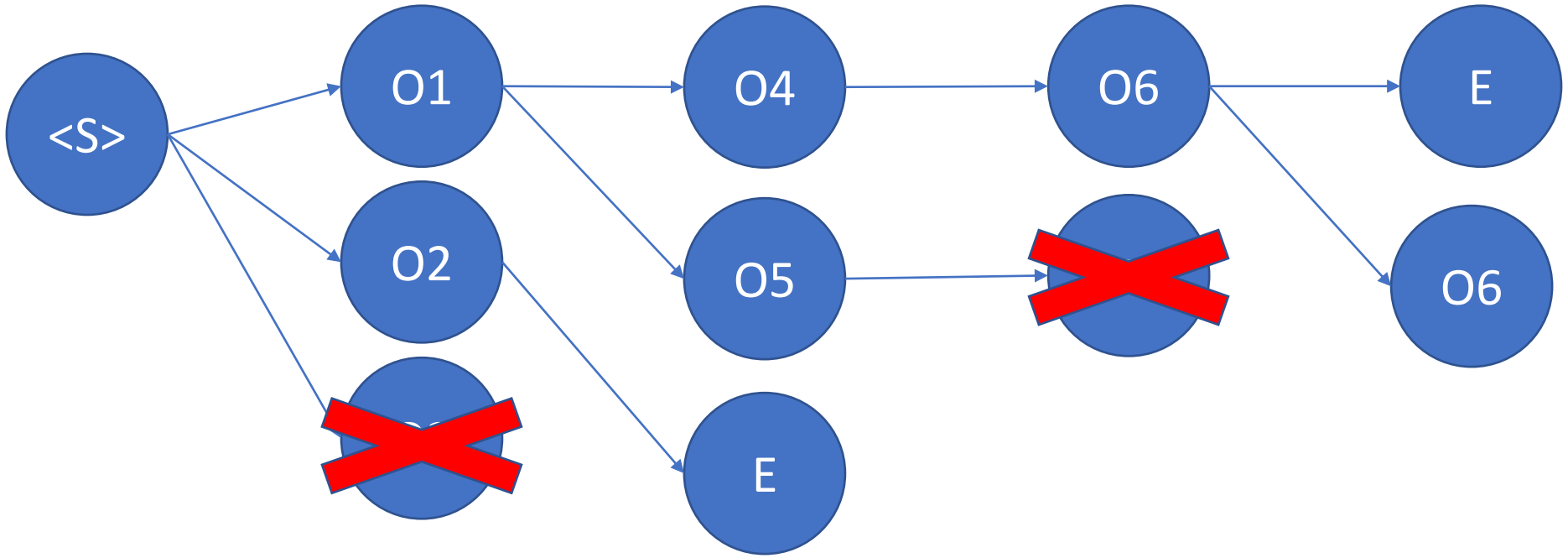
- Tokens especiales: <s> inicio de sentencia, <e> fin de sentencia.



Beam Search

Otra técnica utilizada para la generación de secuencias es el beam search.
En vez de gererar aleatoriamente se va generando las “más” probables:

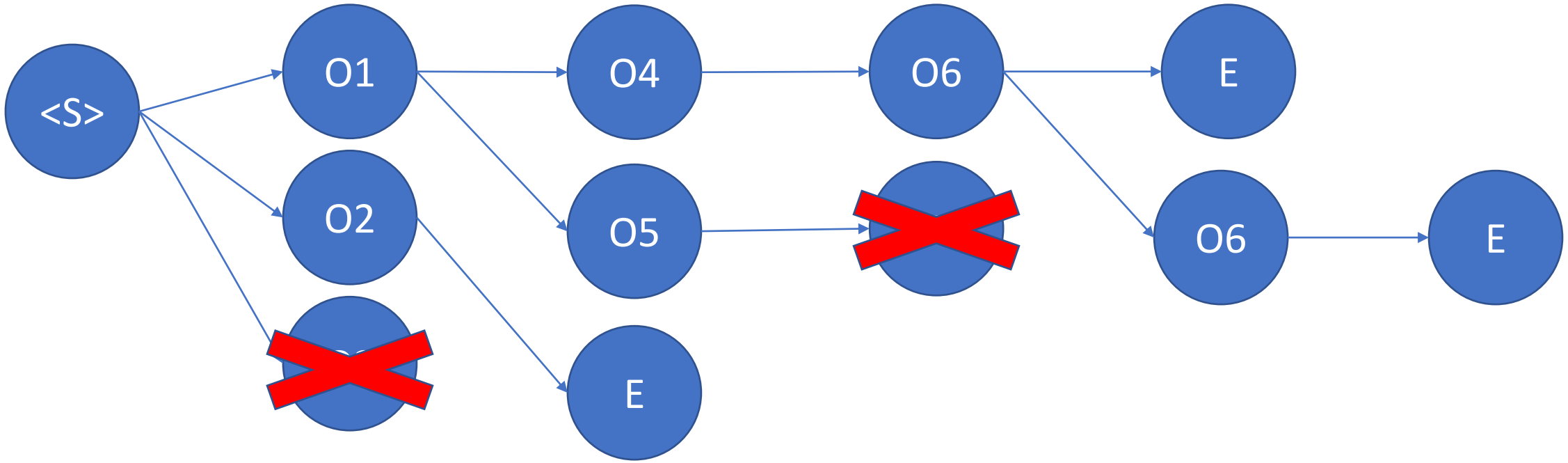
- Tokens especiales: <s> inicio de sentencia, <e> fin de sentencia.



Beam Search

Otra técnica utilizada para la generación de secuencias es el beam search.
En vez de gererar aleatoriamente se va generando las “más” probables:

- Tokens especiales: <s> inicio de sentencia, <e> fin de sentencia.



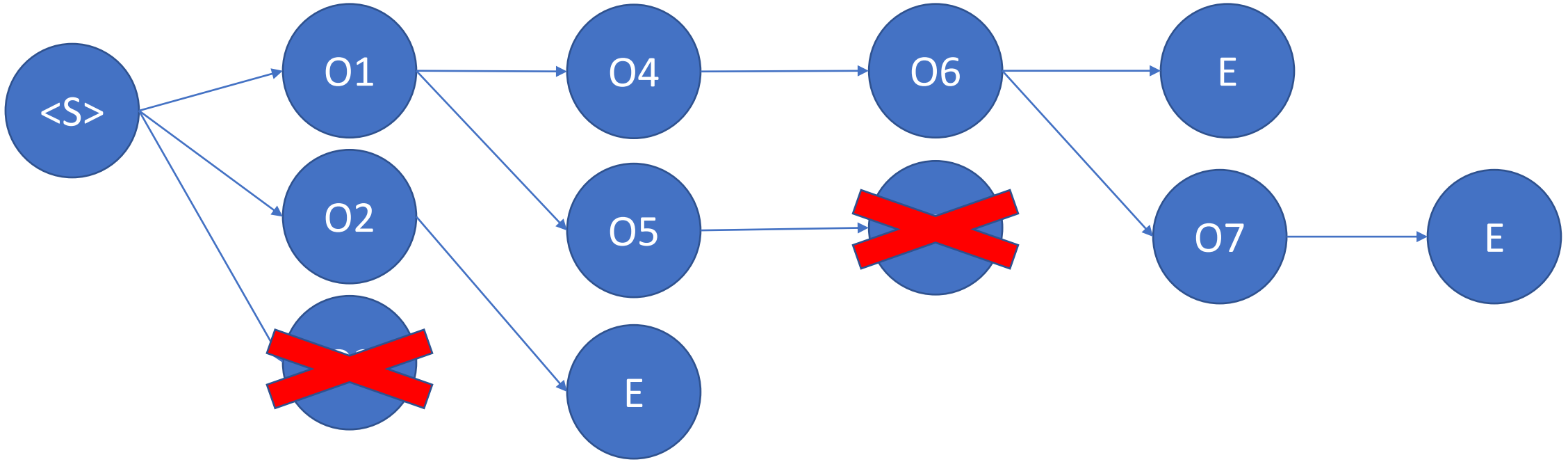
Beam Search

$P(O_1| < S >)$ $P(O_4|O_1 < S >)$ $P(O_6|O_4O_1 < S >)$ $P(E|O_6O_4O_1 < S >)$

$P(O_2| < S >)$ $P(O_5|O_1 < S >)$ $P(E|O_5O_1 < S >)$ $P(O_7|O_6O_5O_1 < S >)$

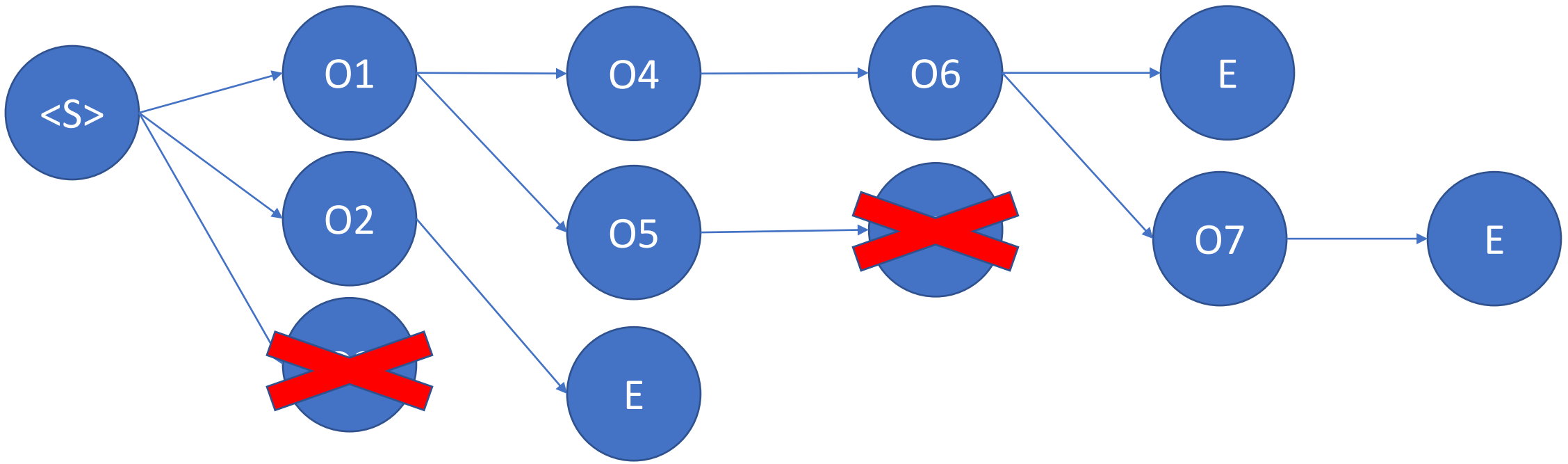
$P(O_3| < S >)$ $P(E|O_2 < S >)$

$P(E|O_7O_6O_5O_1 < S >)$



Beam Search

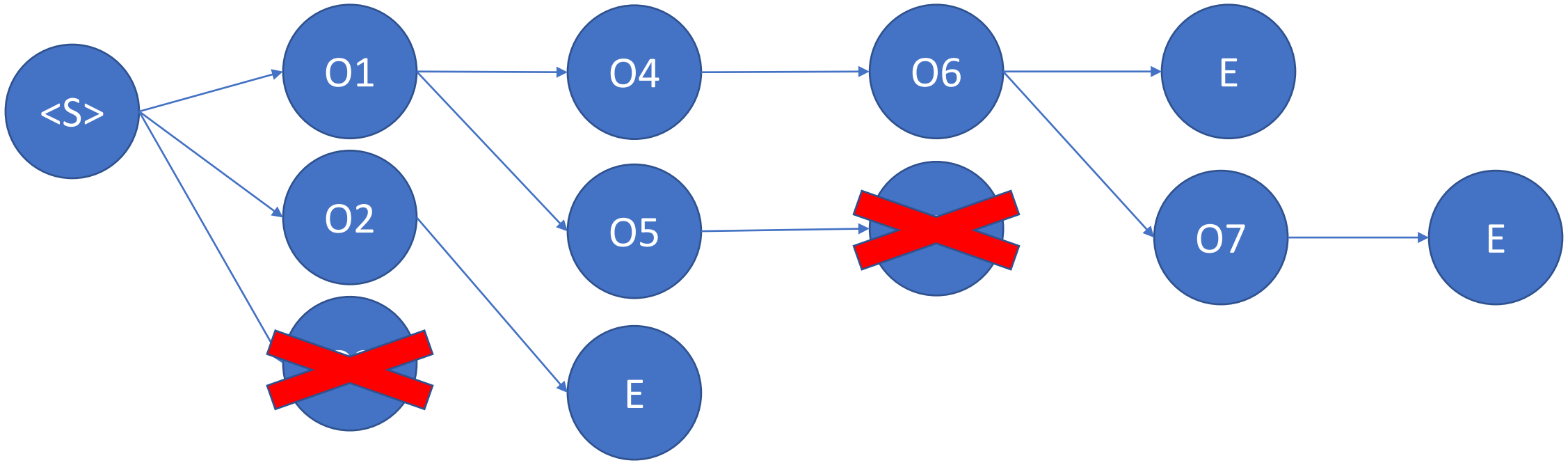
$$P(node_n) = P(node_{n-1}|\dots)P(node_{n-2}|\dots)\dots P(node_n|\dots)$$



Beam Search

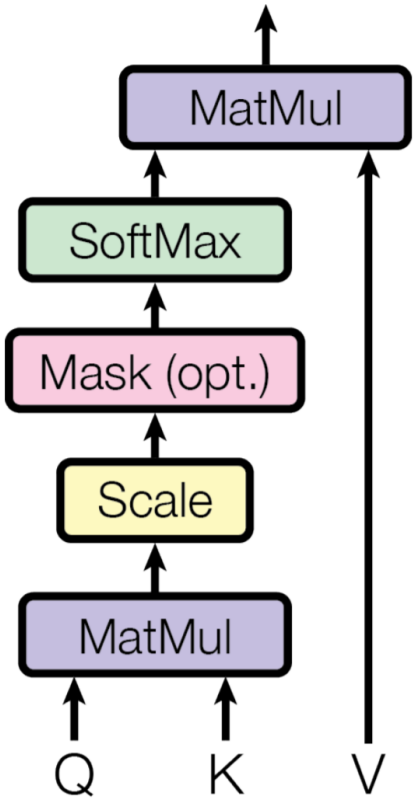
$$P(node_n, \dots, node_1) = P(node_{n-1}| \dots)P(node_{n-2}| \dots) \dots P(node_1| \dots)$$

$$\frac{1}{N}(log\big(P(node_{n-1}| \dots)\big) + \log\big(P(node_{n-2}| \dots)\big) + \cdots + \log\big(P(node_1| \dots)\big))$$



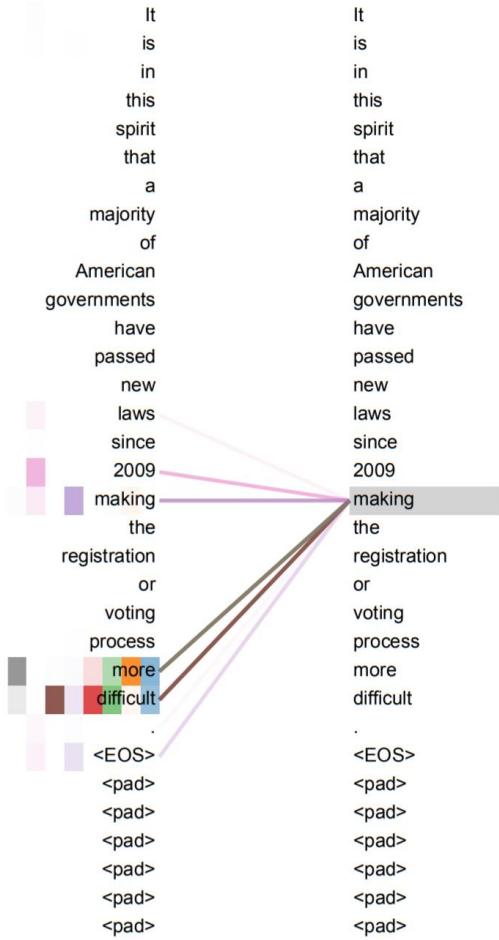
Attention

- Attention es el mecanismo básico de los Transformers.
- En vez de trabajar recurrentemente, comparan los elementos del texto todos contra todos.
- Attention recibe una query, key y values.
- Query es un vector [batch, Tq, dim]
- Key es un vector [batch, Tv, dim]
- Value es un vector [batch, Tv, dim]



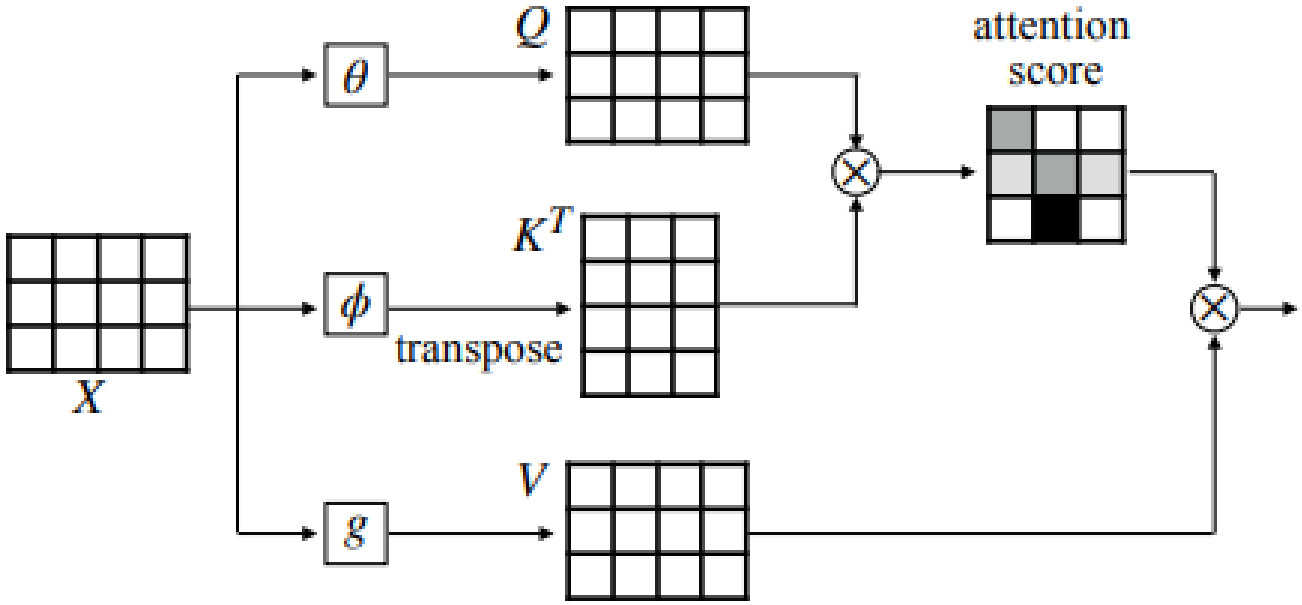
Attention

```
scores = tf.matmul(query, key,
                   transpose_b=True)
distribution = tf.nn.softmax(scores)
return tf.matmul(distribution, value)
```



Self-attention

Self-attention se utiliza cuando los vectores de entrada son de una solo sentencia.

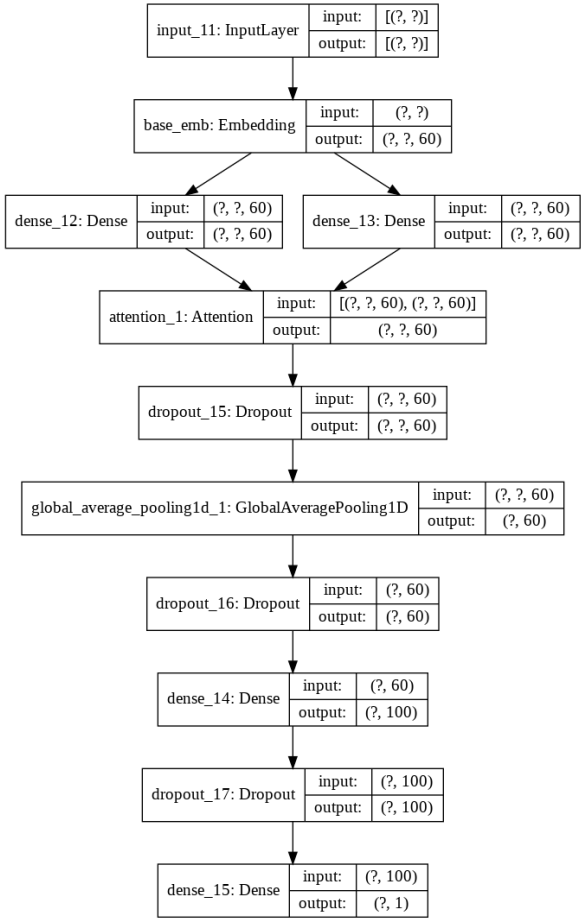


Clasificador con Self-Attention

```
i = Input((None,))
e = Embedding(len(words_id) + 1, 60, mask_zero=True, name='base_emb')(i)

dq = Dense(60)(e)
dk = Dense(60)(e)

att = Attention()([dq, dk])
attd = Dropout(0.1)(att)

d = GlobalAveragePooling1D()(attd)
d = Dropout(0.1)(d)
d = Dense(100)(d)
d = Dropout(0.1)(d)
d = Dense(1, activation='sigmoid')(d)
```

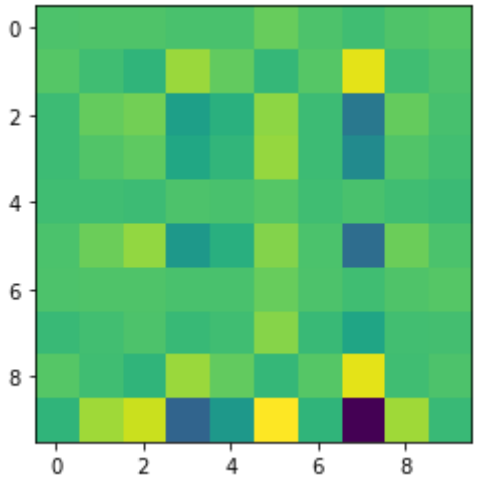
Parámetros [dq, dv, dk] o [dq, dv]

Clasificador con Self-Attention

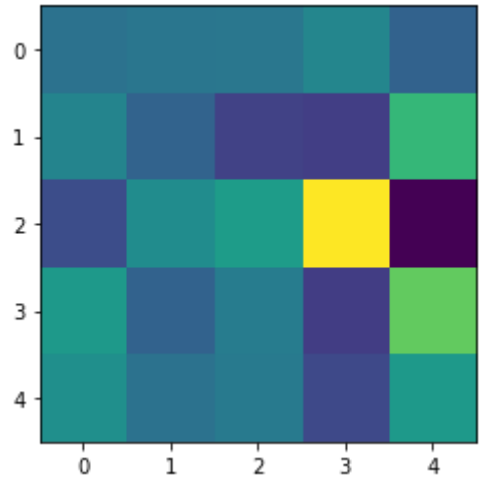


Clasificador con Self-Attention

'The movie was excelent. It is probably the best movie ever'

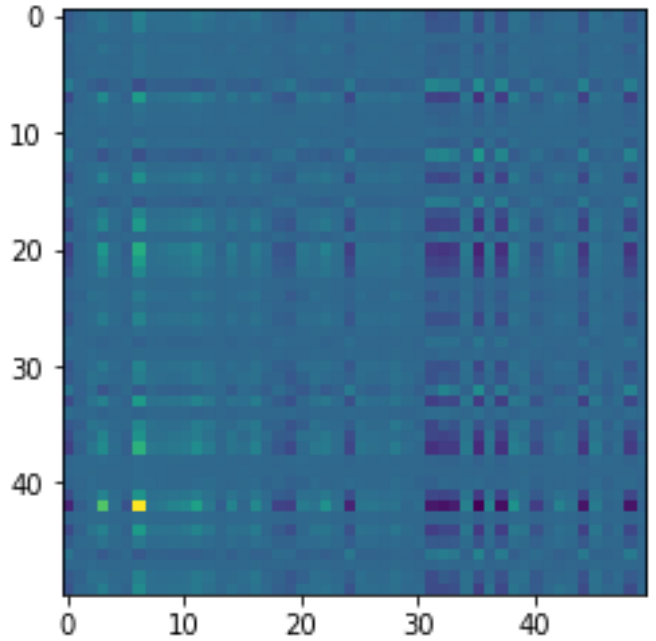


'The movie was not good'



Clasificador con Self-Attention

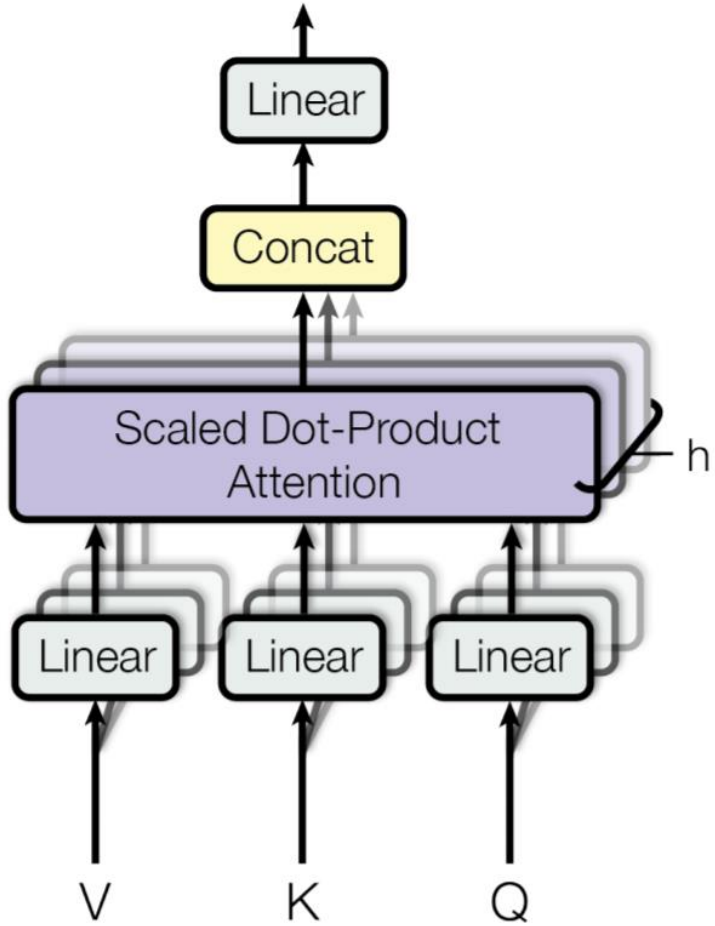
…interest and every line of dialog with sweet poison and cutting ambiguity. John C. McGinley, as Barry's long-suffering screener/technical producer Stu, turns in a hilarious, sharp performance, as does the great Michael Wincott. The film is a flawless, underrated masterpiece of superb writing, awesome acting and brutal, uncompromising direction. The Stewart Copeland score is brilliant, too.



Multi Headed Self-Attention

- Multi-headed Self-Attetion es una técnica que consiste en aplicar varias veces attention sobre diferrentes versiones de los vectores.

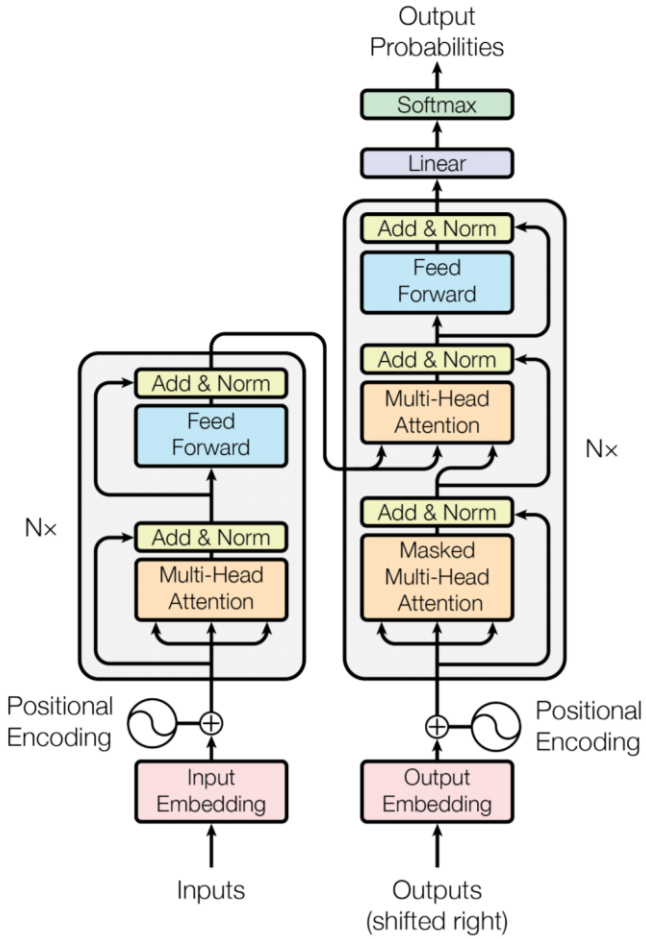
- Utiliza la versión escalada de attention del producto matricial de Q y K. Se escala por la raíz cuadrada de la cantidad de dimensiones.



Transformer

- Originalmente transforemer fue definido para traducción.
- Se utiliza una encoder/decoder.
  - La red de la izquierda codifica nuestra oración de entrada.
  - La red de la derecha la decodifica.
- Se agrega información posicional al encoder. Esta puede ser aprendida o inyectada.
- El encoder puede utilizarse parType equation here.a otras tareas.

$$PE(pos, 2i) = \sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \quad PE(pos, 2i+1) = \cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$



Conclusiones

- Las redes recurrentes son buenas para tratar datos con dependencia temporal.
- La utilización de redes recurrentes, junto con embeddings es una buena opción para NLP.
- Los Transformers son una nueva manera de realizar NLP.
- Transformes se entrenan más rápido y son más generalizables para crear embeddings de textos.