Redes Neuronales para Lenguaje Natural

2024

Grupo de Procesamiento de Lenguaje Natural
Instituto de Computación

RNN como Modelos de Lenguaje

Modelos de Lenguaje

Debido a las copiosas P=0.8 *lluvias* *de las últimas horas …*

P=0.09 *nevadas* *y avalanchas …*

P=0.0000001 *árbol*

- Predecir la probabilidad de una secuencia
- Predecir la siguiente palabra dado un prefijo

P(lluvias | debido a las copiosas)

$$P(w_{1:k}) = \prod_{i=1}^{k} P(w_i|w_{<i})$$

P(<s>debido a las copiosas lluvias</s>) = P(<s>) P(debido|<s>)
P(a | <s> debido) P(las | <s> debido a) … P(</s> | <s>debido a las copiosas lluvias)

Modelos de Lenguaje

P(<s>debido a las copiosas lluvias</s>) = P(<s>) P(debido|<s>)
P(a | <s> debido) P(las | <s> debido a) … P(</s> | <s>debido a las copiosas lluvias)

Históricamente se resuelve mediante conteo de N-gramas

$$P(w_i|w_{<i}) \approx P(w_i|w_{i-N+1:i-1})$$
$$= P(w_i|w_{i-N+1}w_{i-N+2}...w_{i-1})$$

el valor *N* de N-grama

El ejemplo anterior con tri-gramas:

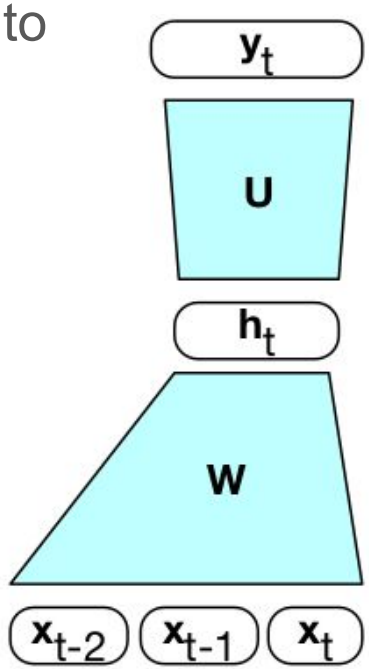
P(<s>debido a las copiosas lluvias</s>) = P(<s>) P(debido|<s>)
P(a | <s> debido) P(las | debido a) P(copiosas | a las)
P(lluvias | las copiosas) P(</s> | copiosas lluvias)

Problemas?

Redes Neuronales como Modelo de Lenguaje

Podemos sustituir los N-gramas por una red feedforward

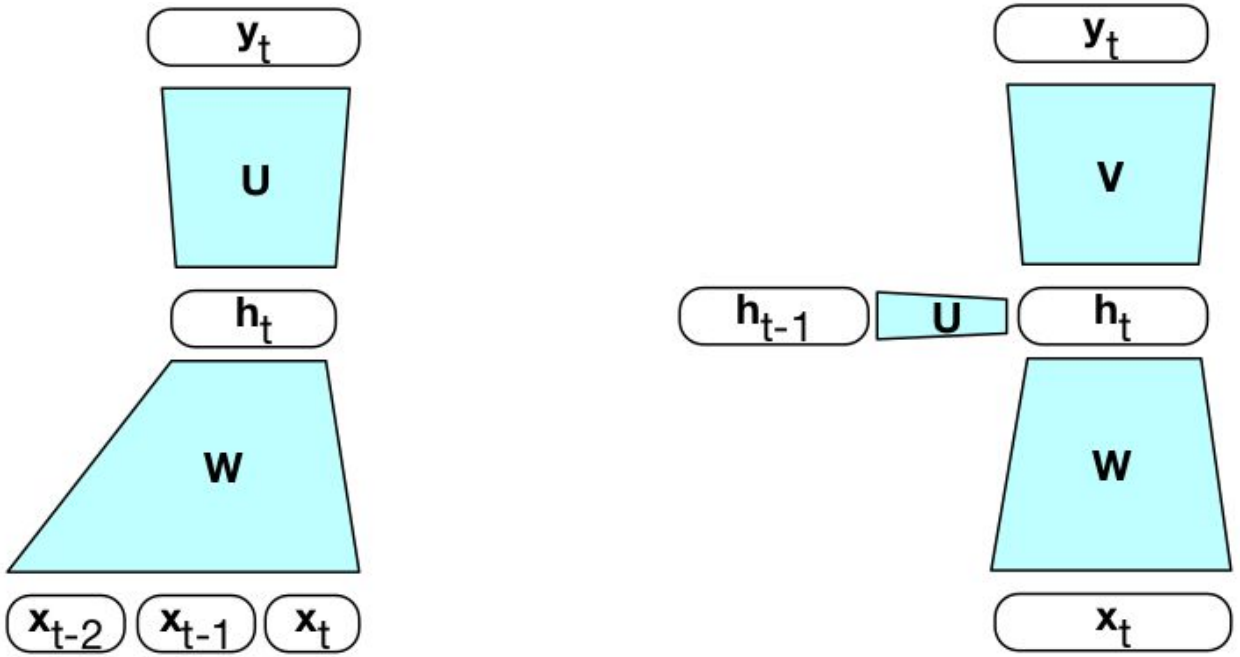
Mejora la generalización a ejemplos de N-gramas no vistos durante entrenamiento



Pero sigue el problema de que las palabras del comienzo pierden relevancia

Redes Neuronales como Modelo de Lenguaje

Alternativa: Usar una RNN, de esta forma todo el contexto anterior se puede ir manteniendo en el estado oculto



Recurrent Neural Network based Language Model
Mikolov et al. (2010)

RNN como Modelo de Lenguaje

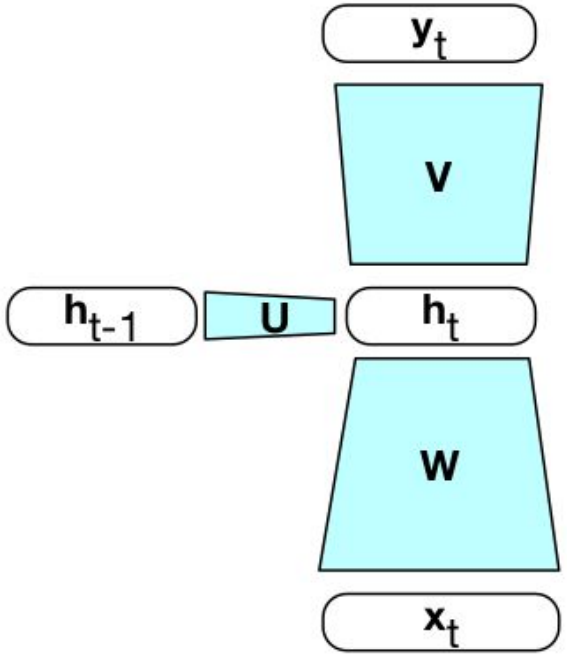
Tomamos la tabla de embeddings E

Matrices de pesos U, V y W

$$e_t = Ex_t$$
$$h_t = g(Uh_{t-1} + We_t)$$
$$y_t = \text{softmax}(Vh_t)$$

y_t es un vector del tamaño del vocabulario

La salida en el tiempo *t* es la distribución de probabilidad de la posible siguiente palabra *t+1*



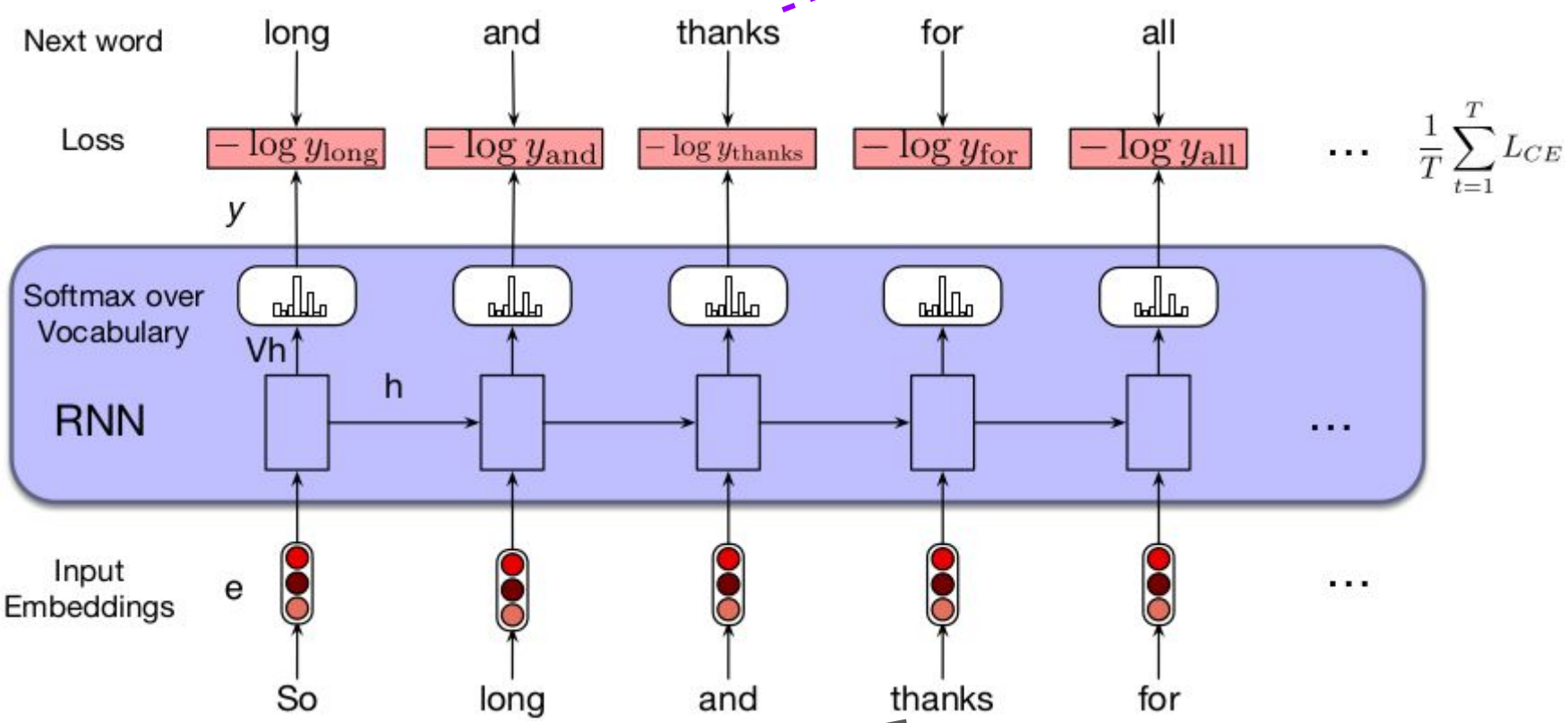
Entrenamiento

- Autosupervisado: usamos oraciones de un corpus conocido y no necesitamos etiquetas
- Se busca minimizar el error de predecir la siguiente palabra dadas todas las anteriores
- El cross-entropy loss en este caso considera que la salida esperada es un one-hot donde solo la siguiente palabra correcta vale 1

$$L_{CE} = -\log \hat{y}_t[w_{t+1}]$$

Entrenamiento

La palabra predicha podría ser la correcta, o no



Pero durante el entrenamiento siempre se presenta la correcta a continuación *(teacher forcing)*

Redes Neuronales como Modelo de Lenguaje

Otras denominaciones: modelo *autorregresivo* de lenguaje, modelo de lenguaje *causal*

Nos permiten predecir la siguiente palabra

…e iterar y predecir toda una continuación de un *prompt*

También calcular la probabilidad de toda una secuencia

Lo podemos usar para otros tipos de tareas?

Tipos de problemas

1 palabra (o un par) → 1 categoría

frío y *caliente* son sinónimos?
antónimos?

MLP

n palabras → 1 categoría

este tweet tiene sentimiento positivo?
este tweet es un chiste?
este mail es spam?

MLP, CNN, RNN

n palabras → 0..n categorías

de qué temas habla este texto?
qué emociones presenta este tweet?

MLP, CNN, RNN

n palabras → n categorías

POS-tagging, NER, chunking,
parsing, SRL

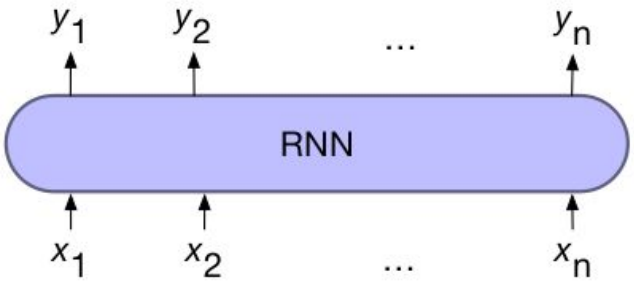
CNN, RNN

n palabras → m palabras

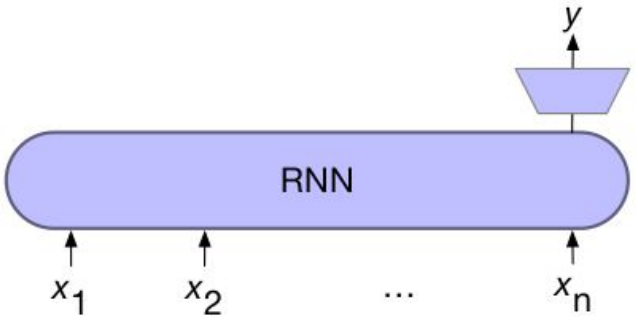
traducción automática
respuestas a preguntas
resúmenes automáticos
chatbots…

Encoder-Decoder (RNN, Transformer)

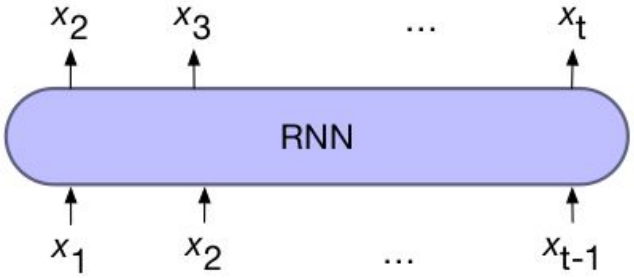
Usos de las redes recurrentes



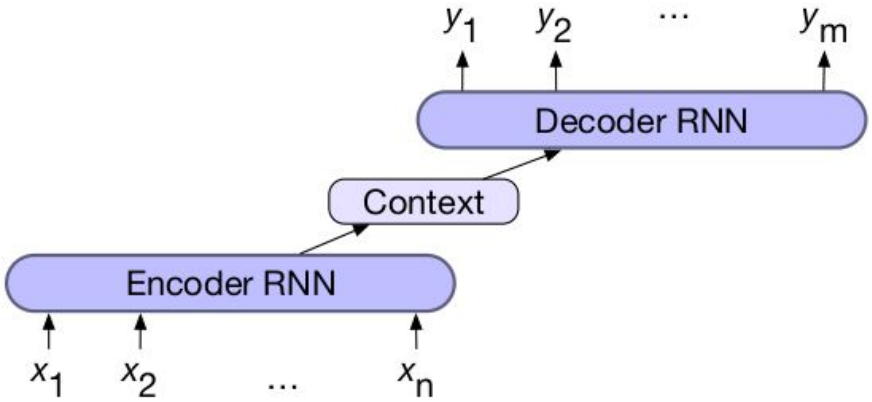
a) etiquetado secuencial



b) clasificación de secuencias



c) modelos de lenguaje



d) modelo encoder-decoder

Arquitectura Encoder-Decoder

Arquitectura Encoder-Decoder

Toma una secuencia de entrada

Devuelve una secuencia de salida

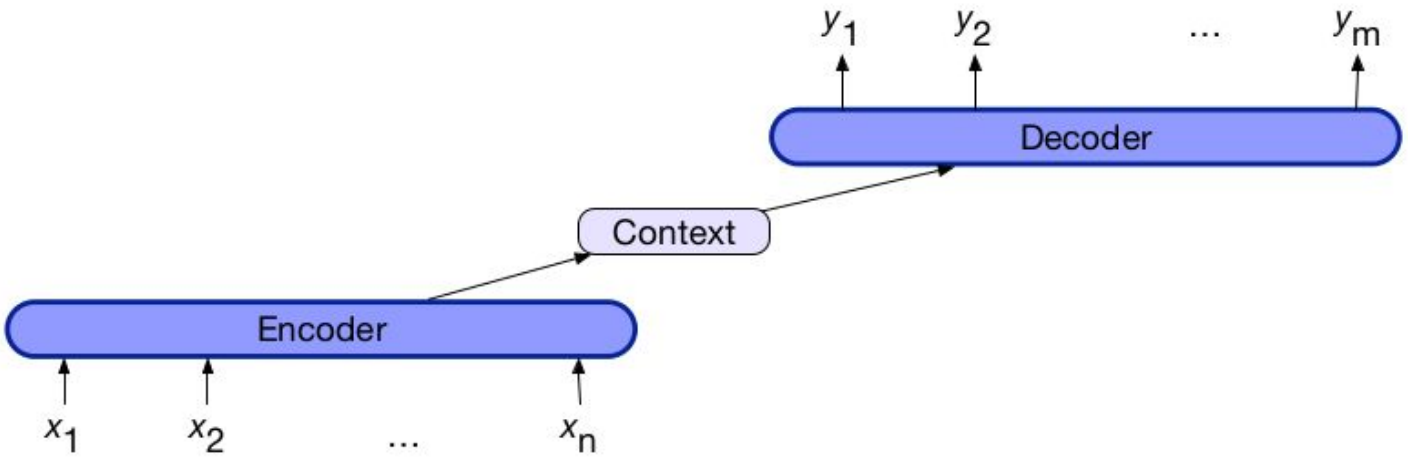
La cantidad de elementos de entrada y de salida pueden ser diferentes

Los tokens posibles también pueden ser diferentes de cada lado, por ejemplo en distinto idioma (traducción automática)

Arquitectura Encoder-Decoder

Una red compuesta por dos subredes:

- **Encoder**: red que codifica la entrada
- **Decoder**: red que decodifica (y construye) la salida
- **Context vector**: “embedding” de toda la secuencia de entrada



Arquitectura Encoder-Decoder

Secuencia de entrada x_1^n

- **Encoder**: toma x_1^n y genera una secuencia de representaciones contextualizadas h_1^n. Se puede hacer con LSTMs, convolucionales, transformers (redes secuenciales).
- **Context vector**: este vector *c* se construye a partir de las h_1^n y representa “toda la semántica” de la entrada
- **Decoder**: a partir de *c*, genera una tira de estados ocultos h_1^m con los que se construye las salidas de la red y_1^m

Modelo de lenguaje autorregresivo

Queremos generar la secuencia de salida $y = y_1^m$

$$p(y) = p(y_1)p(y_2|y_1)p(y_3|y_1, y_2)...p(y_m|y_1, ..., y_{m-1})$$

Los estados ocultos h_t de una red se calculan usando el estado oculto en el paso anterior, y el nuevo token en el tiempo t

$$h_t = g(h_{t-1}, x_t)$$

La salida en el paso t se obtiene a partir del estado oculto h_t

$$y_t = f(h_t)$$

Generar salida a partir de una entrada

Una forma de modelarlo: consideremos que estamos haciendo un modelo autorregresivo sobre las secuencias origen y destino separados por un token especial <s>

$$x_1 x_2 \dots x_n \langle s \rangle y_1 y_2 \dots y_m$$

Sea $x = x_1^n \langle s \rangle$, entonces:

$$p(y|x) = p(y_1|x)p(y_2|y_1, x)p(y_3|y_1, y_2, x)...p(y_m|y_1, ..., y_{m-1}, x)$$

Es el modelo autorregresivo con un generador condicionado al estado oculto de la codificación de *x* (secuencia de entrada)

Generar salida a partir de una entrada

Supongamos que estamos construyendo un sistema que traduce del inglés al español

En nuestra red usaremos la secuencia:

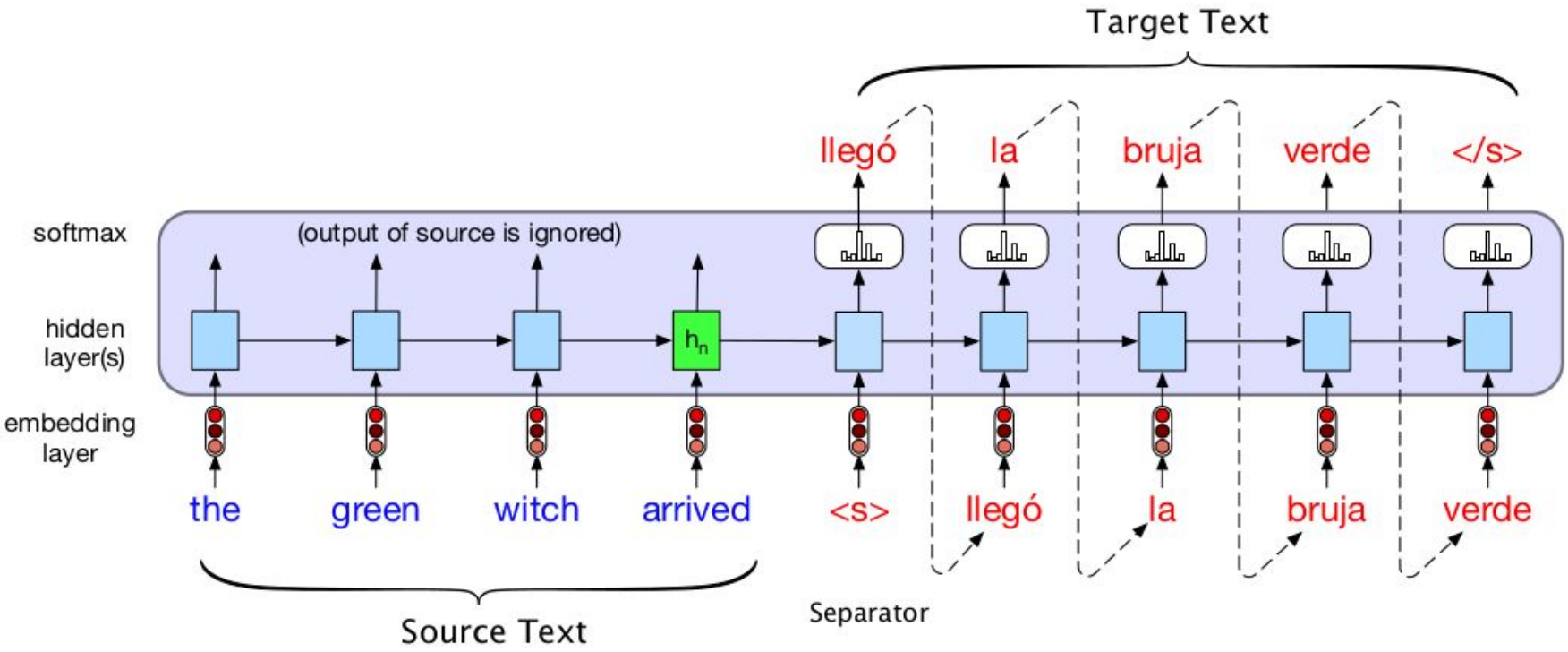
(salida esperada)

the green witch arrived <s> llegó la bruja verde

Alimentamos la red con los tokens en inglés hasta el token <s>, generando el estado oculto *(forward inference)*

Luego pasamos a usarlo como modelo de lenguaje generativo para obtener las palabras en español *(autoregressive generation)*

Generar salida a partir de una entrada



Encoder-Decoder

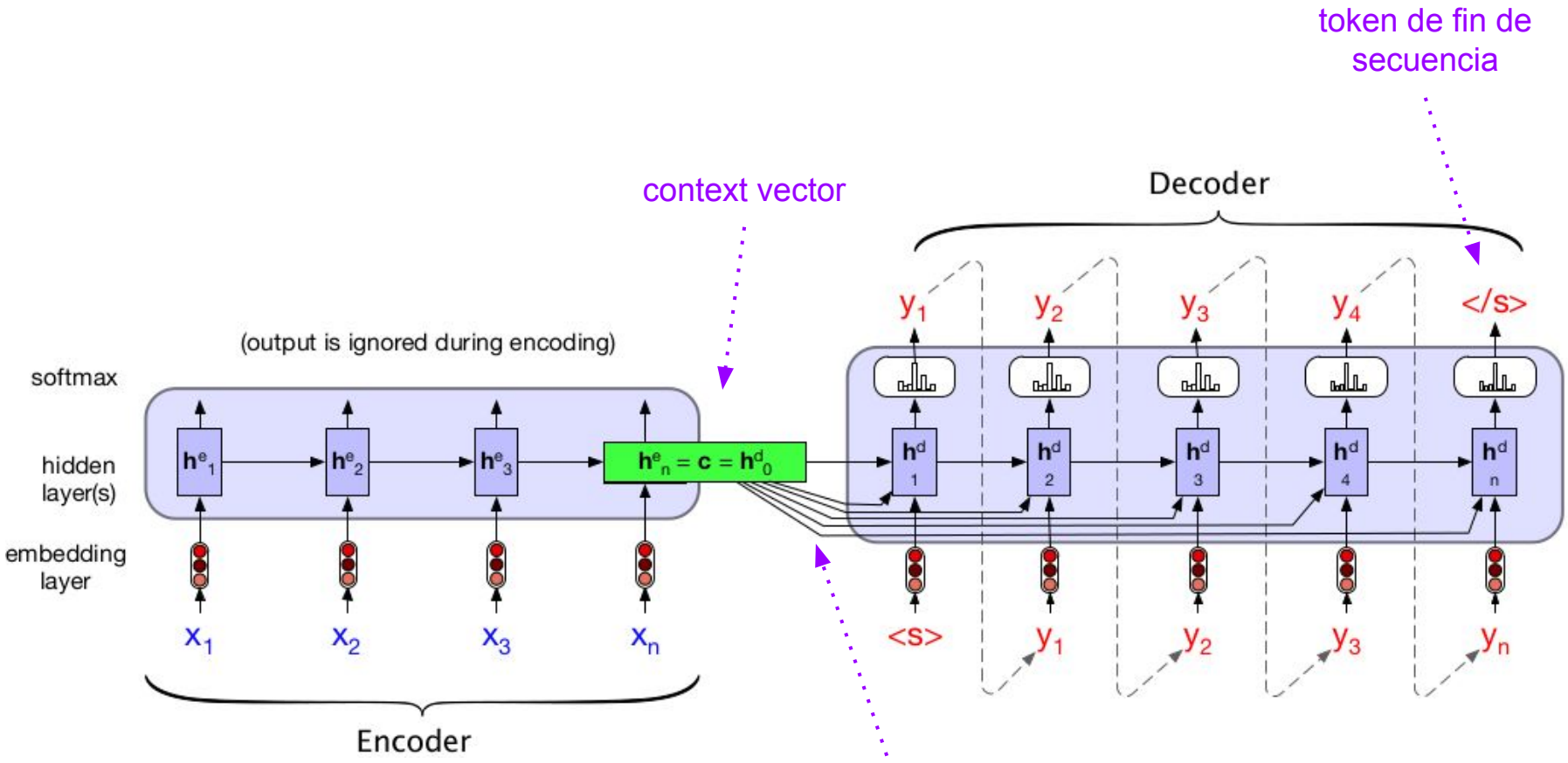
Formalizaremos la idea como composición de dos subredes

El encoder genera los estados ocultos h^e, el decoder los estados ocultos h^d

El último estado oculto del encoder será el vector de contexto, que además es el vector inicial con que se alimenta el decoder

$$h^e_n = c = h^d_0$$

Encoder-Decoder



debido a que la influencia de c sobre la decodificación de la secuencia puede ir disminuyendo, se suele incluir en cada paso

Encoder-Decoder

Codificación de la secuencia de entrada

$$c = h_n^e$$

Estado inicial del decodificador

$$h_0^d = c$$

Cada paso del decoder usa el estado oculto anterior, el token decodificado y el contexto

$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c)$$

El estado oculto se pasa por una nueva red feed forward y se obtiene el siguiente token con un softmax

$$z_t = f(h_t^d)$$

$$y_t = \text{softmax}(z_t)$$

Cómo computar la siguiente salida:

$$\hat{y}_t = \text{argmax}_{w \in V} P(w|x, y_1 \ldots y_{t-1})$$

Encoder-Decoder: entrenamiento

Se entrenan con pares de secuencias (entrada - salida)

Por ejemplo, corpus paralelos en traducción automática: oraciones origen en inglés y oraciones destino en español

La red *encoder* codifica la entrada, se ignoran las salidas intermedias y se mantiene solo el vector de contexto

A partir del vector de contexto se decodifica la salida con el *decoder* de forma autorregresiva

Se entrena *end-to-end*, minimizando el *loss* de acertar a las palabras correctas

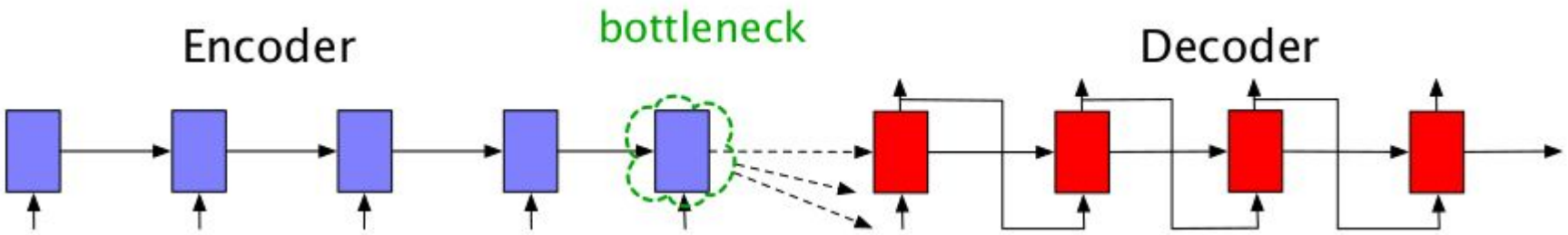
Se utiliza *teacher forcing* al entrenar la decodificación de la salida


Mecanismo Atencional

Problema del vector de contexto

El encoder genera un estado oculto para cada token de la secuencia $h^e_1 \ldots h^e_n$

Toda esta información se condensa en el *context vector*



Esto es un problema, porque es un vector de tamaño finito que tiene que poder representar la semántica de cualquier secuencia posible (cuello de botella)

Problema del vector de contexto

Solución: ¿qué tal si en cada paso del decoder se utilizara un vector de contexto distinto?

$$c_i = f(h_1^e ... h_n^e)$$

Cada vector de contexto es calculado en función de los estados ocultos del encoder para cada token de la entrada

La cantidad de tokens es variable, y la cantidad de estados ocultos también, lo que haremos es crear un vector de tamaño fijo usando pesos para combinar los distintos $h^e_1 \ldots h^e_n$

Mecanismo Atencional

Los estados ocultos del decoder ahora se calculan:

$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$$

Los pesos que usamos para combinar los h^e_j dan una idea de qué tan importante es cada token *j* al momento de decodificar el paso *i*

Pesos más grandes implican que el modelo le está “prestando más atención” a ese token en ese paso de decodificación

Le asignaremos un *score* a cada par (h^d_{i-1}, h^e_j), el esquema atencional que usaremos definirá cómo se calcula ese score

Cálculo de scores

El score que le damos al par (h^d_{i-1}, h^e_j) representa qué tanta atención se le presta a h^e_j al generar la salida *i*

Una primera idea es medir qué tan similares son el estado oculto anterior y el estado oculto del token *j*

$$\text{score}(h^d_{i-1}, h^e_j) = h^d_{i-1} \cdot h^e_j$$

Esta es la atención más simple de todas: atención de producto punto (*dot product attention*), y no introduce parámetros nuevos

Effective Approaches to Attention-based Neural Machine Translation
Luong, Pham, Manning, 2015

Cálculo del vector de contexto

Una vez que tenemos definida una función de score, pasamos definir cómo se calcula el vector de contexto para el paso *i*

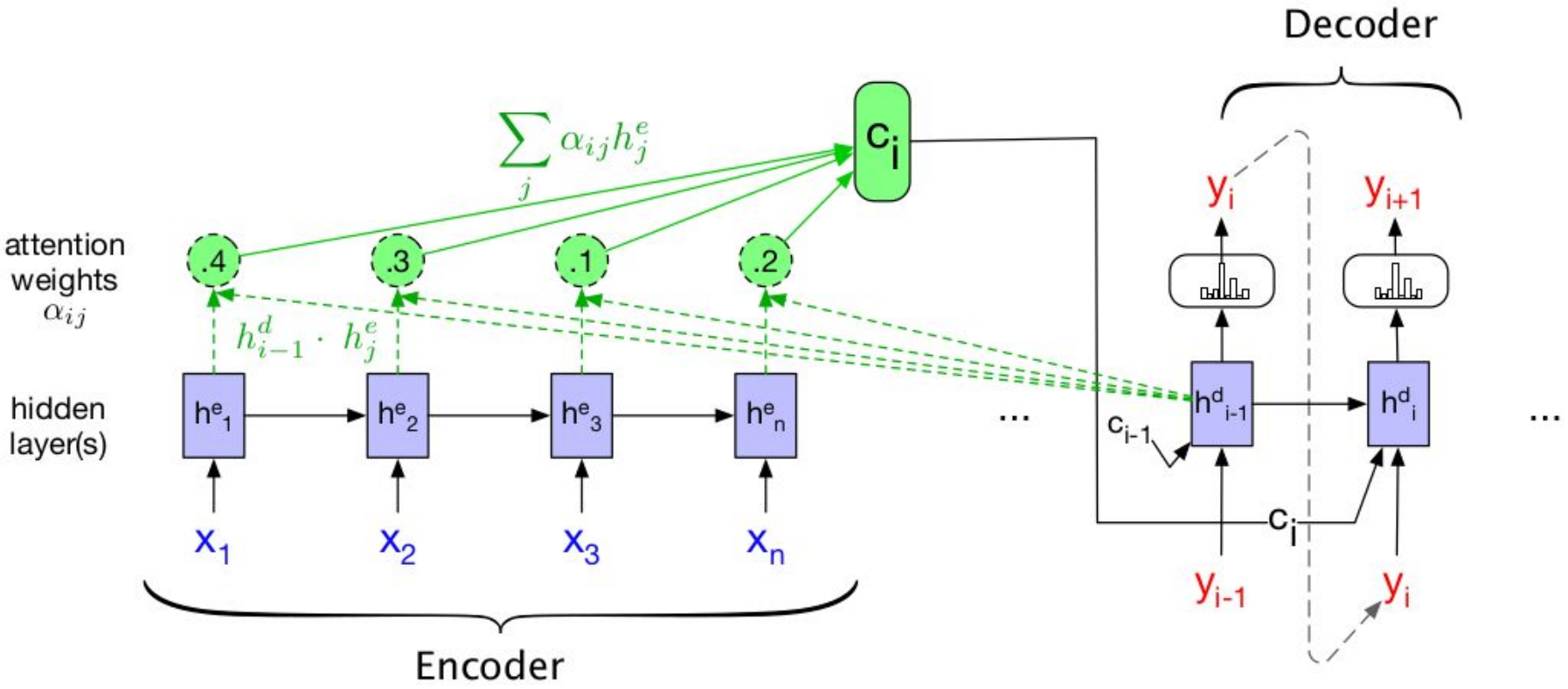
Softmax sobre todos los scores para normalizarlos a pesos

$$\begin{aligned}\alpha_{ij} &= \text{softmax}(\text{score}(h^d_{i-1}, h^e_j)) \\ &= \frac{\exp(\text{score}(h^d_{i-1}, h^e_j))}{\sum_k \exp(\text{score}(h^d_{i-1}, h^e_k))}\end{aligned}$$

Multiplicamos los pesos por los estados ocultos del encoder para generar el vector de contexto

$$c_i = \sum_j \alpha_{ij} h^e_j$$

Mecanismo Atencional



Otros scores

La *dot product attention* es simple, pero no permite ningún grado de expresividad extra porque no tiene parámetros

Se han desarrollado otros métodos de atención paramétricos que tienen mejores resultados

- Atención aditiva (Atención de Bahdanau)
- Atención multiplicativa (Atención de Luong)

Atención aditiva

Modela la función de score como una red feedforward

Aparecen parámetros nuevos:

- Matriz *W* que opera sobre la concatenación de los estados ocultos
- Vector *v* de pesos para definir la salida final

$$\text{score}(h_{i-1}^d, h_j^e) = v^T \tanh(W[h_j^e; h_{i-1}^d])$$

Luego sigue igual: hacemos softmax sobre los scores para cada token para obtener pesos y calculamos el vector de contexto

Neural Machine Translation by Jointly Learning to Align and Translate
Bahdanau, Cho, Bengio, 2014

Atención multiplicativa

En vez de modelar como red feedfoward y usar una matriz y un vector de pesos, utiliza una única matriz con producto bilineal

$$\text{score}(h_{i-1}^d, h_j^e) = h_{i-1}^d W h_j^e$$

Es similar a la *dot product attention*, pero agrega la matriz de pesos parametrizables

Luego sigue igual: hacemos softmax sobre los scores para cada token para obtener pesos y calculamos el vector de contexto

Effective Approaches to Attention-based Neural Machine Translation
Luong, Pham, Manning, 2015

Bibliografía

- Jurafsky & Martin, 3rd Ed. (draft) - Capítulos 8, 13 y 3
- Tutoriales sobre mecanismos atencionales de Stefania Cristina https://machinelearningmastery.com/
- Papers…