Redes Neuronales para PLN

Parte 2

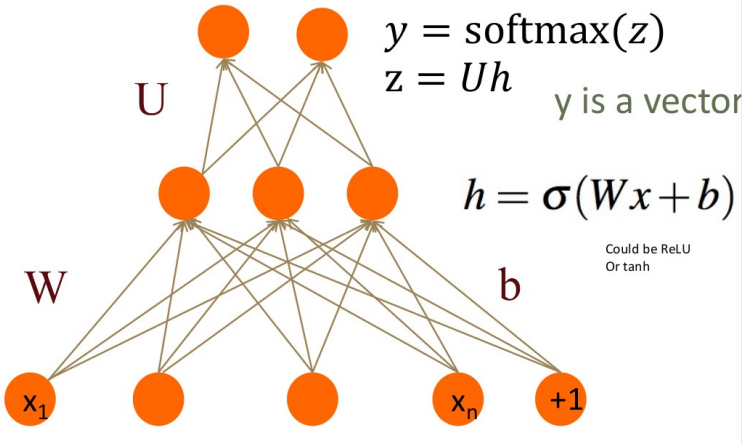
CDLN 2022

Jurafsky and Martin 3rd edition. Caps 7,9,11. https://web.stanford.edu/~jurafsky/slp3/

FFN o MLP (multi-layer perceptron)
(Multi-layer fully-connected feed forward Network)

Cada **capa** => secuencia de **neuronas artificiales**

Neurona - recibe vector, devuelve escalar
- pesos ajustables, func. activación



Un MLP es una secuencia de capas apiladas

Entrada → capas ocultas → capa de salida

Sucecivas transformaciones de la entrada

... es **<u>composición</u> de funciones diferenciables**

Regla de la cadena => gradiente de los pesos por capa (**backpropagation**)

Función objetivo (MAE, MSE, Cross-entropy, contrastive, triplet)

Se usan técnicas basadas en el gradiente y mini-batches (adam, rmsprop, sgd)

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

Algunos conceptos/prácticas generales

Corpus (train, validation/development, test)

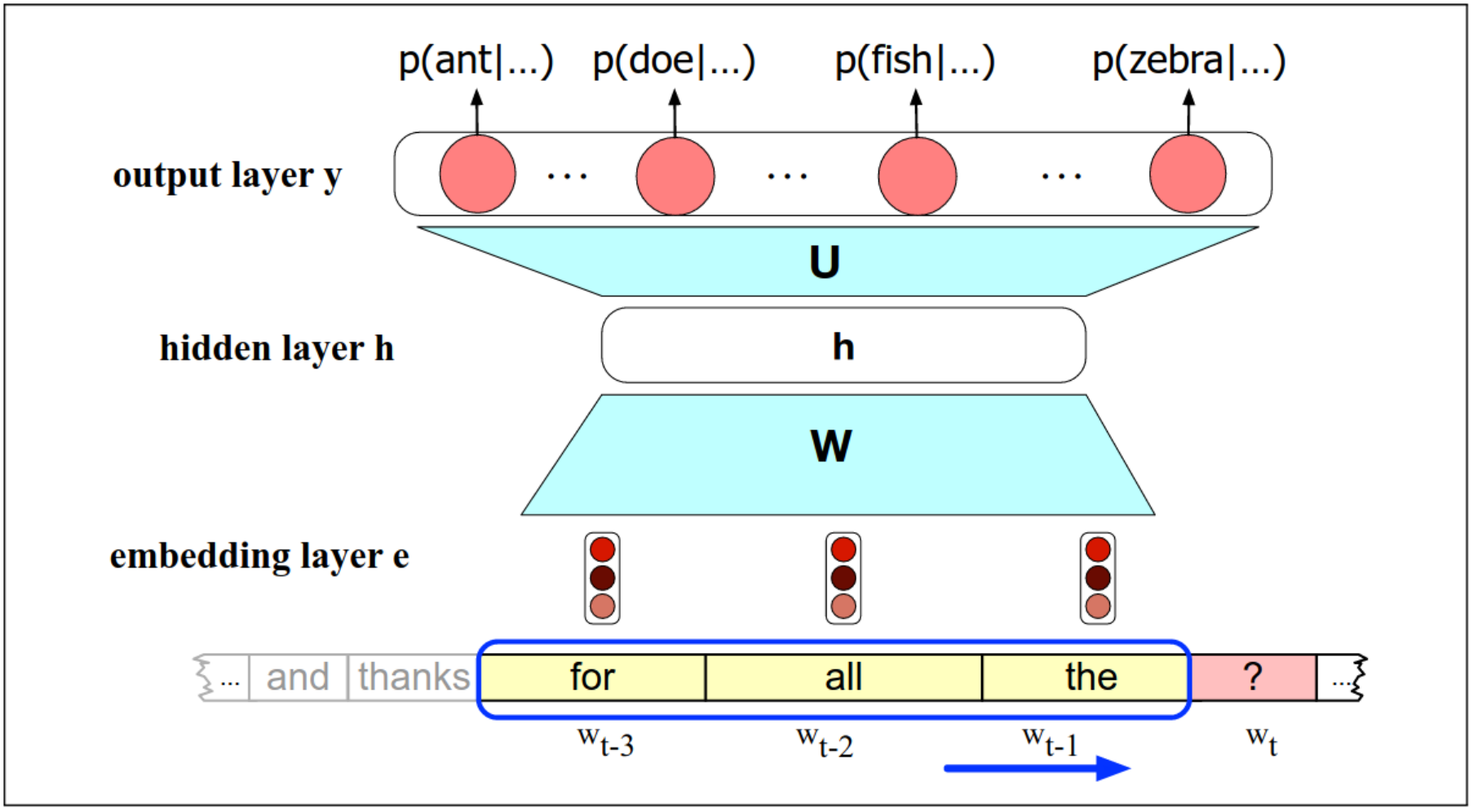
Early Stopping (usando validation)

Regularizations (dropout,noise, normalization)

Hyperparameter Search (manual, grid, random)

Embedding layer (look-up/embedding matrix)

Y si la entrada es una secuencia?



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

Seq2Vec

Ej. Text classification, Análisis de Sentimiento, Detección de Humor, Hatespeech,

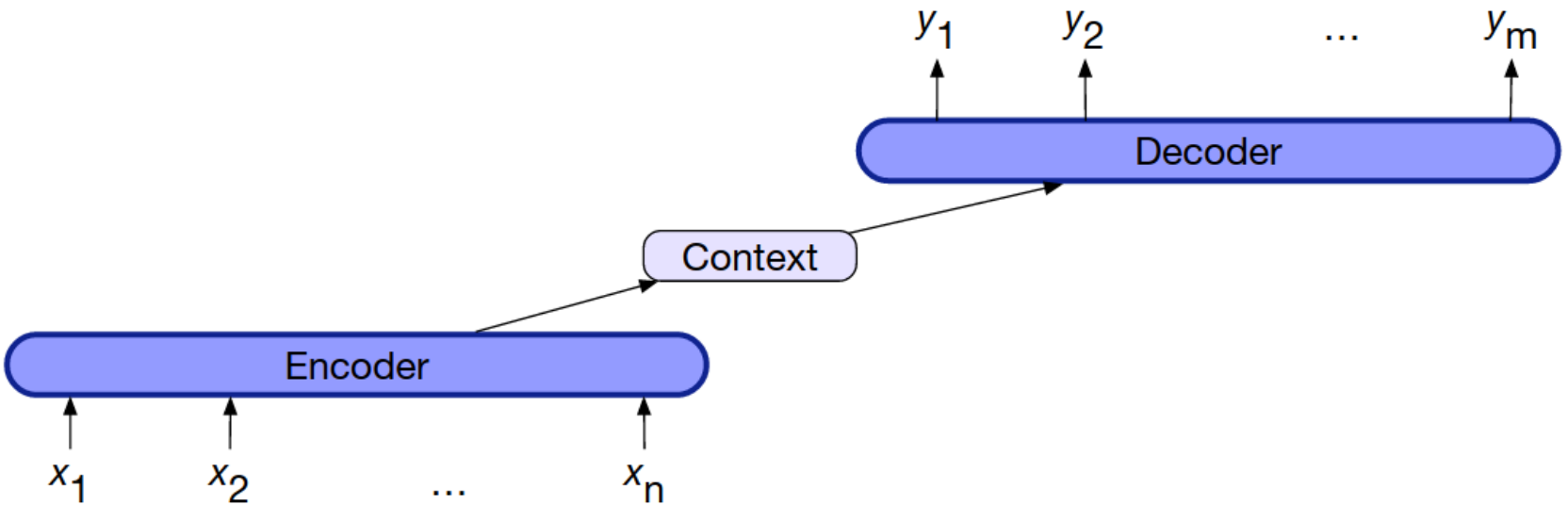
Vec2Seq

Ej. Generación de texto, Image captioning

Seq2Seq (encoder-decoder)

Ej. Traductores automáticos,
Resumen abstractivo, Question-answering,
Generative chatbots,

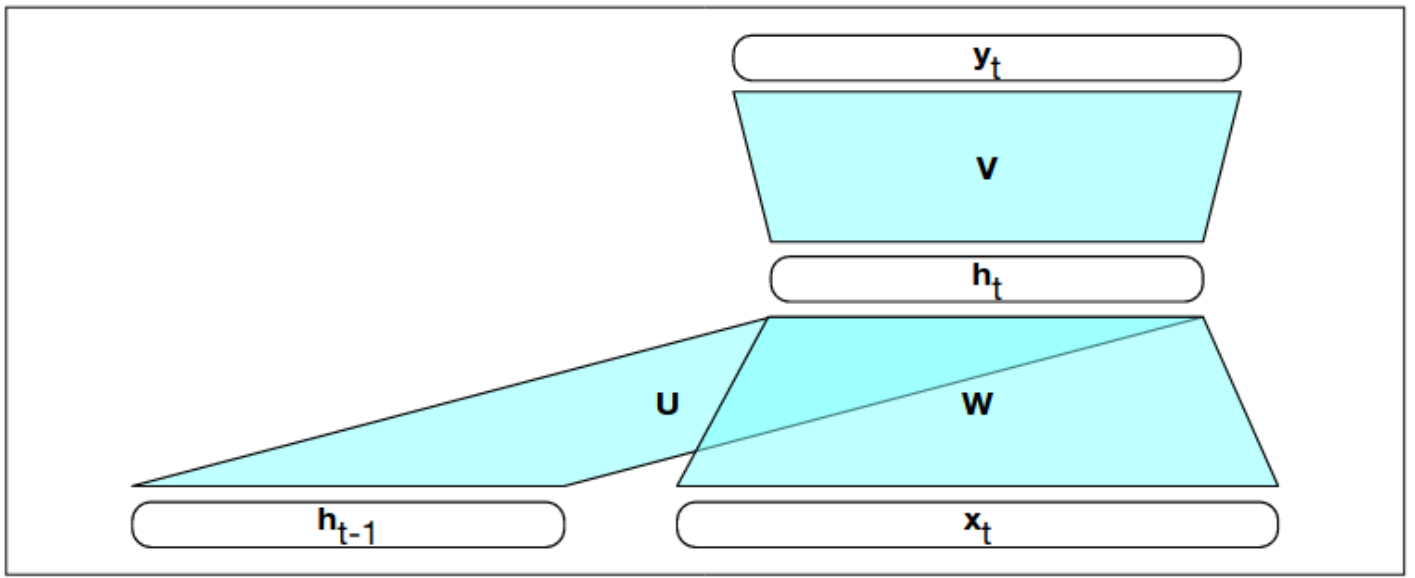
Encoder-decoder



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

Recurrent Neural Networks

Simple Recurrent Network (Elman, 1990)



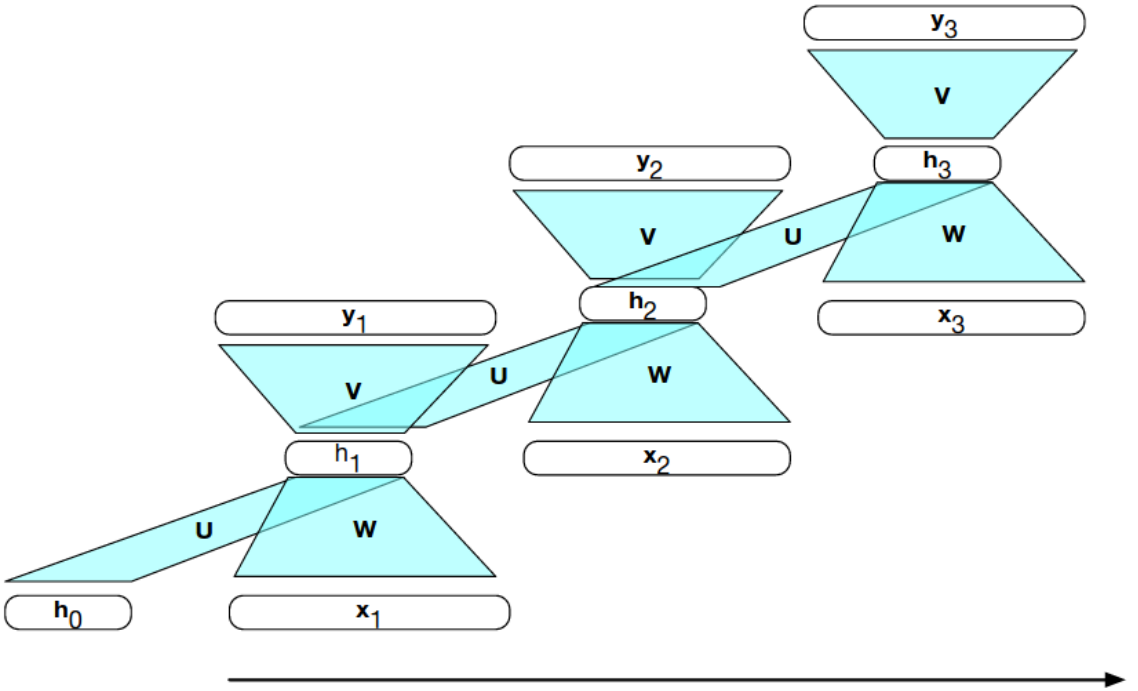
$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$$
$$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$$

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

Simple Recurrent Network (Elman, 1990)

function FORWARDRNN($\mathbf{x}$, *network*) **returns** output sequence $\mathbf{y}$

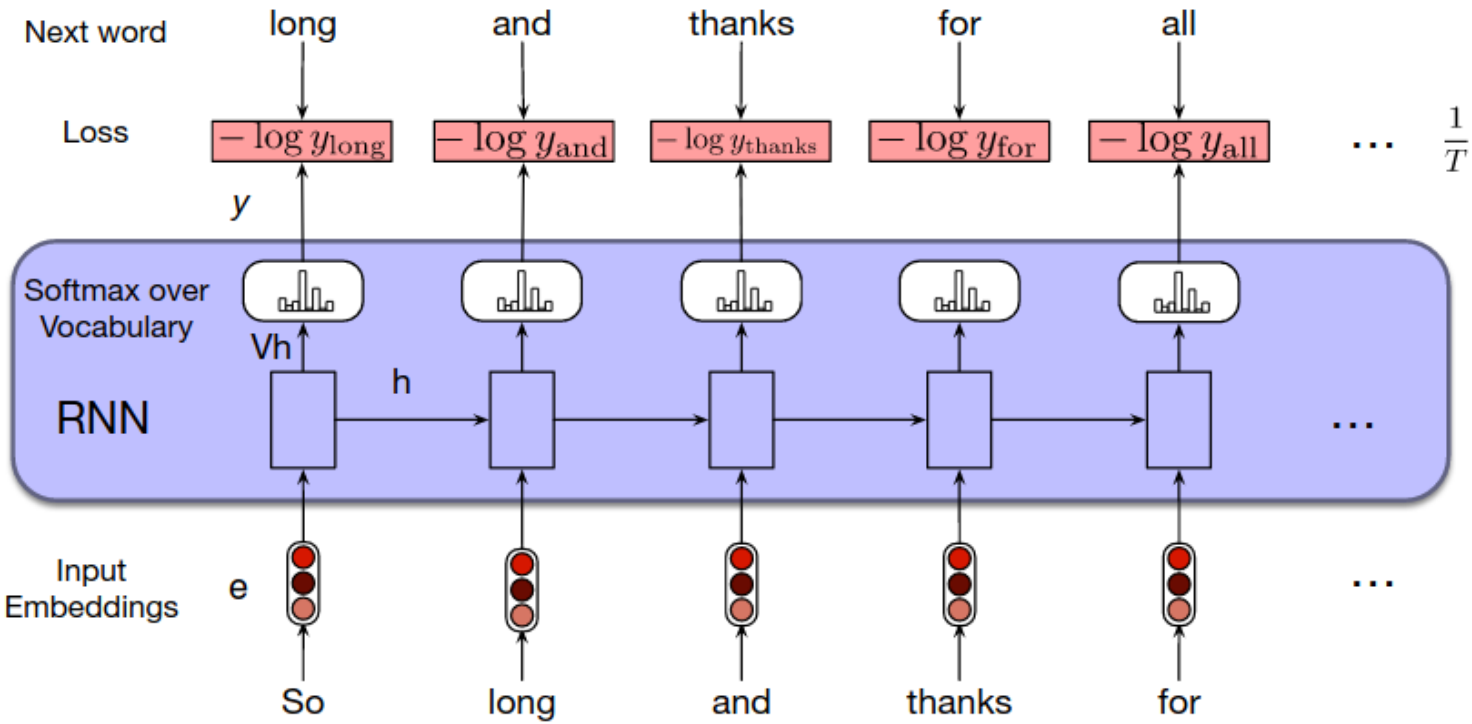
$\mathbf{h}^0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH($\mathbf{x}$) **do**
  $\mathbf{h}_i \leftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$
  $\mathbf{y}_i \leftarrow f(\mathbf{V}\mathbf{h}_i)$
return y



Entrenamiento
Cada paso es una composición (Backpropagation Through Time, BPTT)

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

RNN as Language Models (Mikolov, 2010)



Teacher forcing
Al entrenar cada paso se usa la secuencia correcta de entrada, se ignoran las predicciones del modelo.

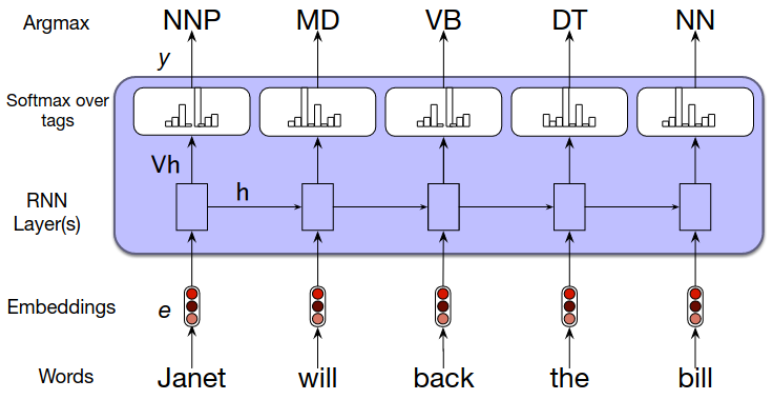
Weights tying
Las matrices E (embeddings) y V (softmax) pueden ser la misma, reduciendo la cantidad de parámetros del modelo.

$$\mathbf{e}_t = \mathbf{E}\mathbf{x}_t$$
$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t)$$
$$\mathbf{y}_t = \text{softmax}(\mathbf{E}^{intercal}\mathbf{h}_t)$$

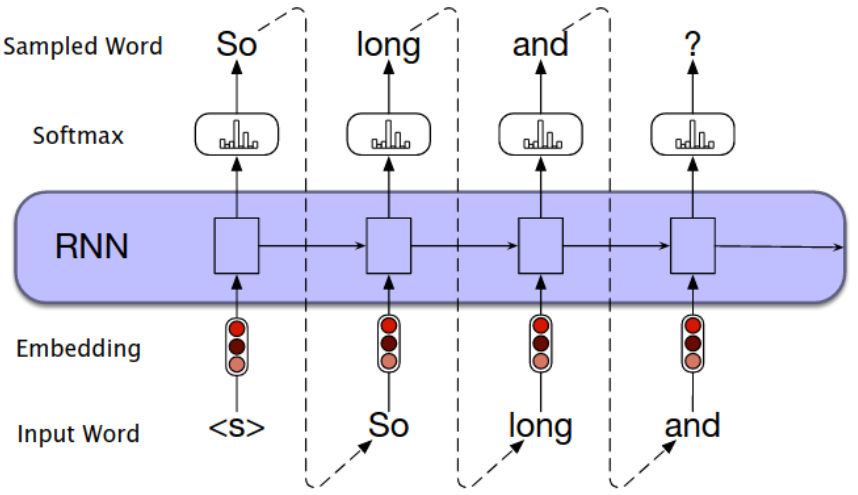
Basado en https://web.stanford.edu/~jurafsky/slp3/

RNN for Other NLP Tasks

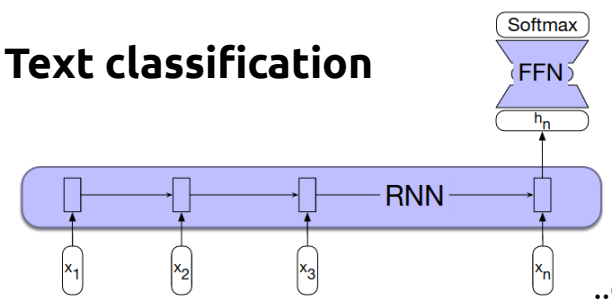
Sequence labeling



Text generation



Text classification



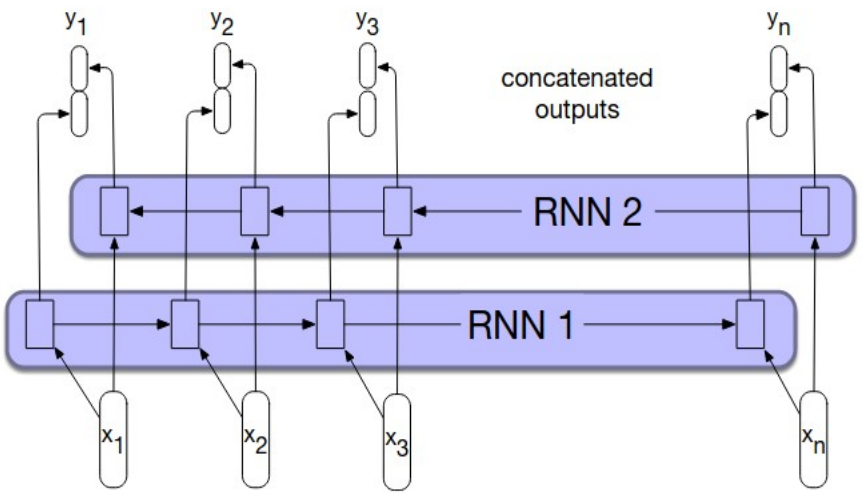
..o usar una combinación de todos los estados ocultos

$$\mathbf{h}_{mean} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{h}_i$$

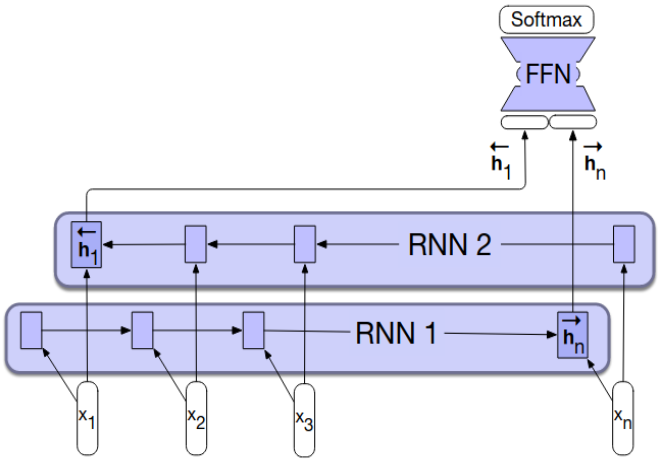
Basado en https://web.stanford.edu/~jurafsky/slp3/

Bidirectional RNNs

Sequence labeling

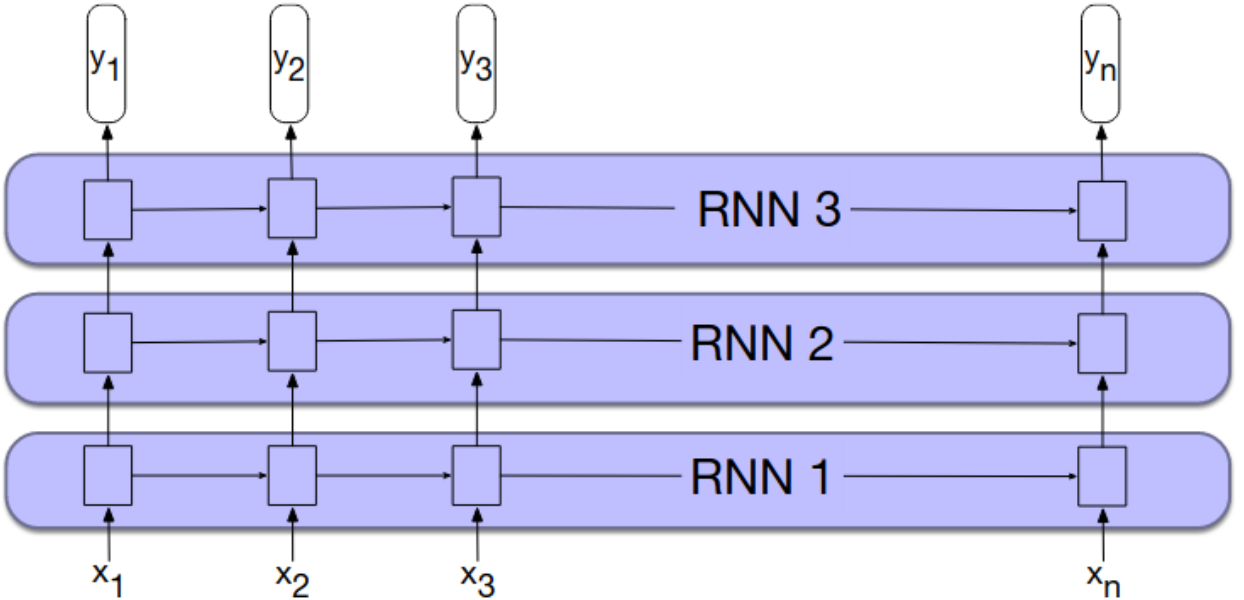


Sequence classification



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

Stacked RNNs



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

RNNs Vanishing Gradient Problem

Las RNNs tienden a degradar su funcionamiento con la información más alejada (ej. dependencias de largo alcance)

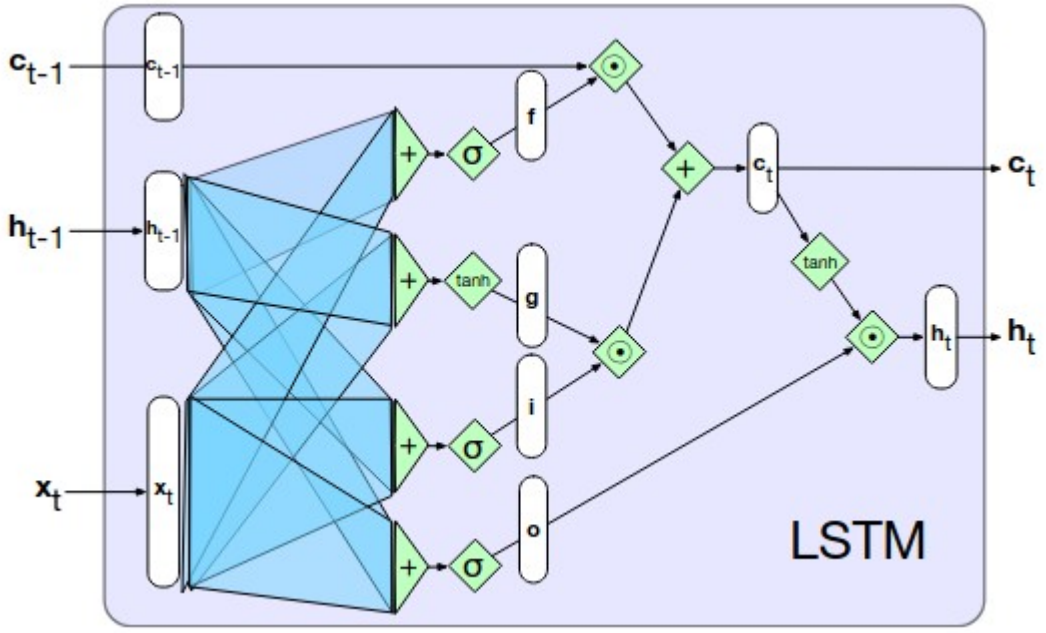
Vanishing Gradients Problem

Los *steps* en una RNN (ej. cada token de la secuencia) resulta en multiplicaciones de gradientes que eventualmente se van a 0, dependiendo de su cercanía a 0 y el largo de la secuencia

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

Long short-term memory (Hochreiter and Schmidhuber, 1997)

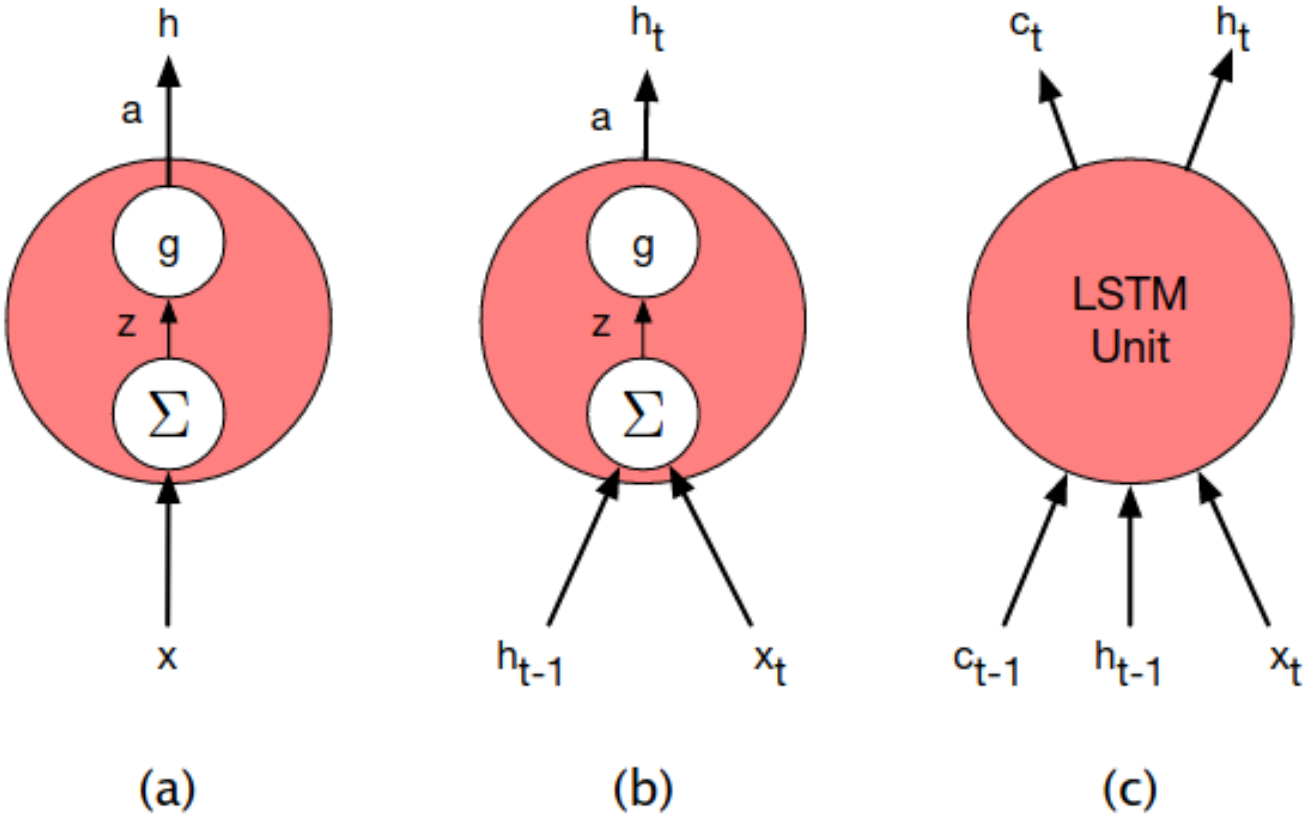
LSTM es un tipo de capa recurrente con un vector de contexto (c_t) que puede ser "propagado".
El diseño se basa en **compuertas** con capas feedforward, activaciones sigmoid y producto element-wise (Hadamard) (funciona como una máscara binaria)



$$\mathbf{f}_t = \sigma(\mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t)$$
$$\mathbf{k}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t$$
$$\mathbf{g}_t = \tanh(\mathbf{U}_g \mathbf{h}_{t-1} + \mathbf{W}_g \mathbf{x}_t)$$
$$\mathbf{i}_t = \sigma(\mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{W}_i \mathbf{x}_t)$$
$$\mathbf{j}_t = \mathbf{g}_t \odot \mathbf{i}_t$$
$$\mathbf{c}_t = \mathbf{j}_t + \mathbf{k}_t$$
$$\mathbf{o}_t = \sigma(\mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{W}_o \mathbf{x}_t)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Este tipo de formulación de la capa recurrente no es única, por ejemplo una GRU (Gated Recurrent Unit) es un tipo de capa recurrente con ideas similares, comparable a LSTM.

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

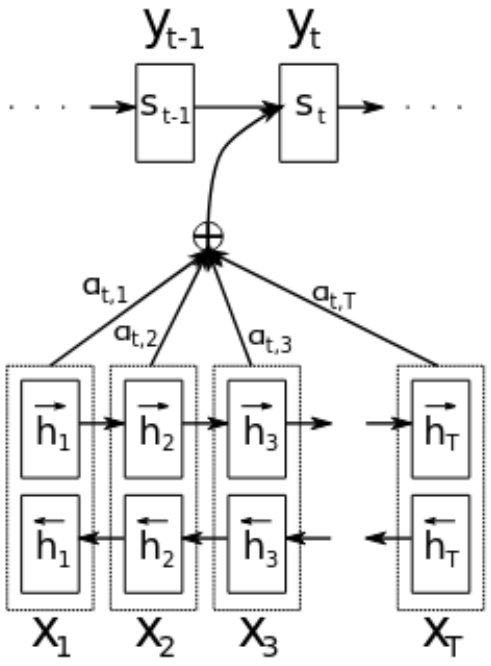


Extraído de https://web.stanford.edu/~jurafsky/slp3/

LSTMs y GRUs mejoran el manejo de información más distante pero no resuelven el problema

RNN + Attention (soft-search)
Bahdanau, D., Cho, K., & Bengio, Y. (2014)

Planteado para RNN encoder-decoder. Al generar, en cada paso, se considera como contexto un promedio ponderado (usando softmax y ffn) de los estados ocultos (anotaciones) de la entrada.



$$p(y_i|y_1,\ldots,y_{i-1},\mathbf{x}) = g(y_{i-1},s_i,c_i),$$

where s_i is an RNN hidden state for time i, computed by

$$s_i = f(s_{i-1},y_{i-1},c_i).$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$

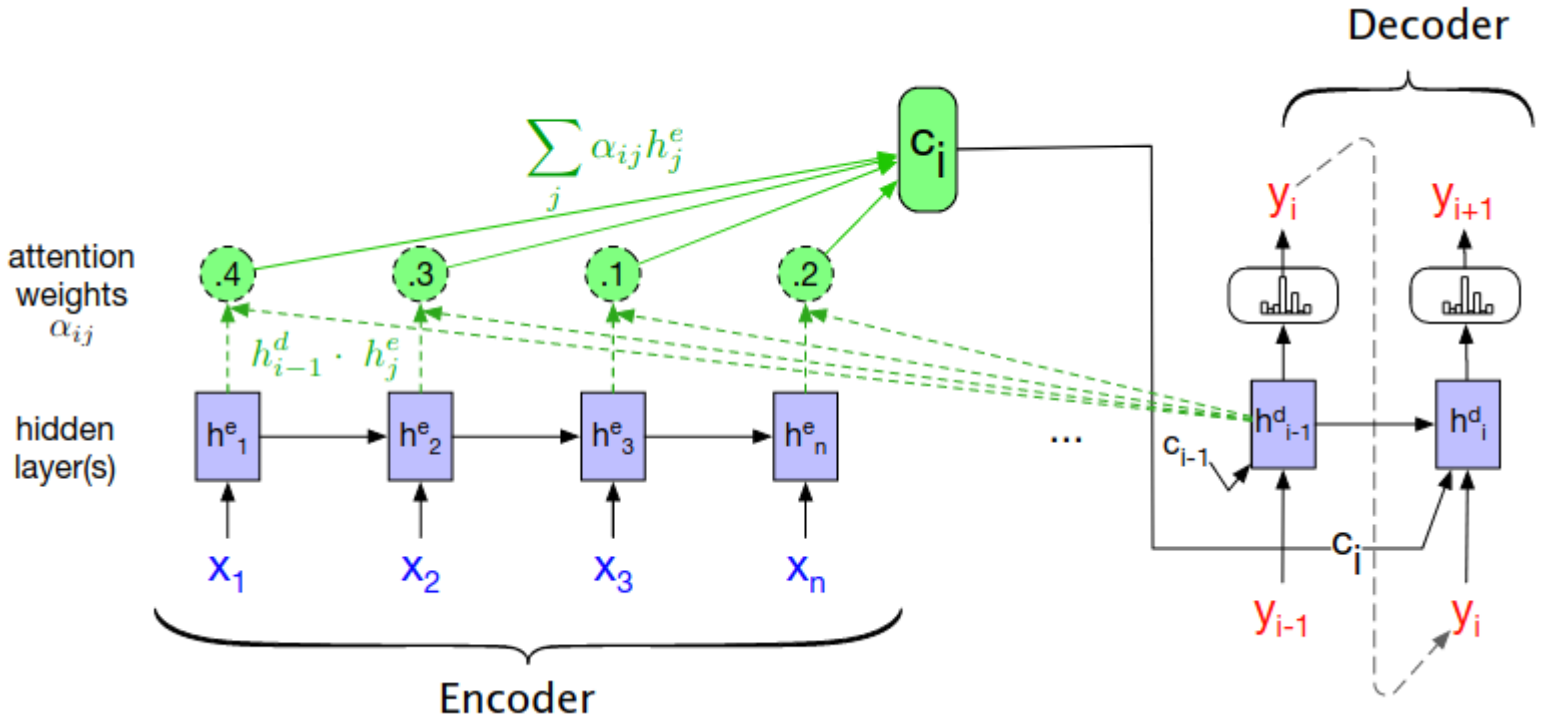
The weight α_{ij} of each annotation h_j is computed by

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

where

$$e_{ij} = a(s_{i-1},h_j)$$

RNN + Attention



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

ELMo (representaciones contextualizadas)

Peters, Matthew E. et al. “Deep Contextualized Word Representations.” NAACL (2018).

ULMFiT (fine-tuning)

Howard, Jeremy and Sebastian Ruder. “Fine-tuned Language Models for Text Classification.” ArXiv abs/1801.06146 (2018): n. pag.

Transformer

Transformer

La naturaleza recurrente de las RNNs dificulta el procesamiento paralelo.

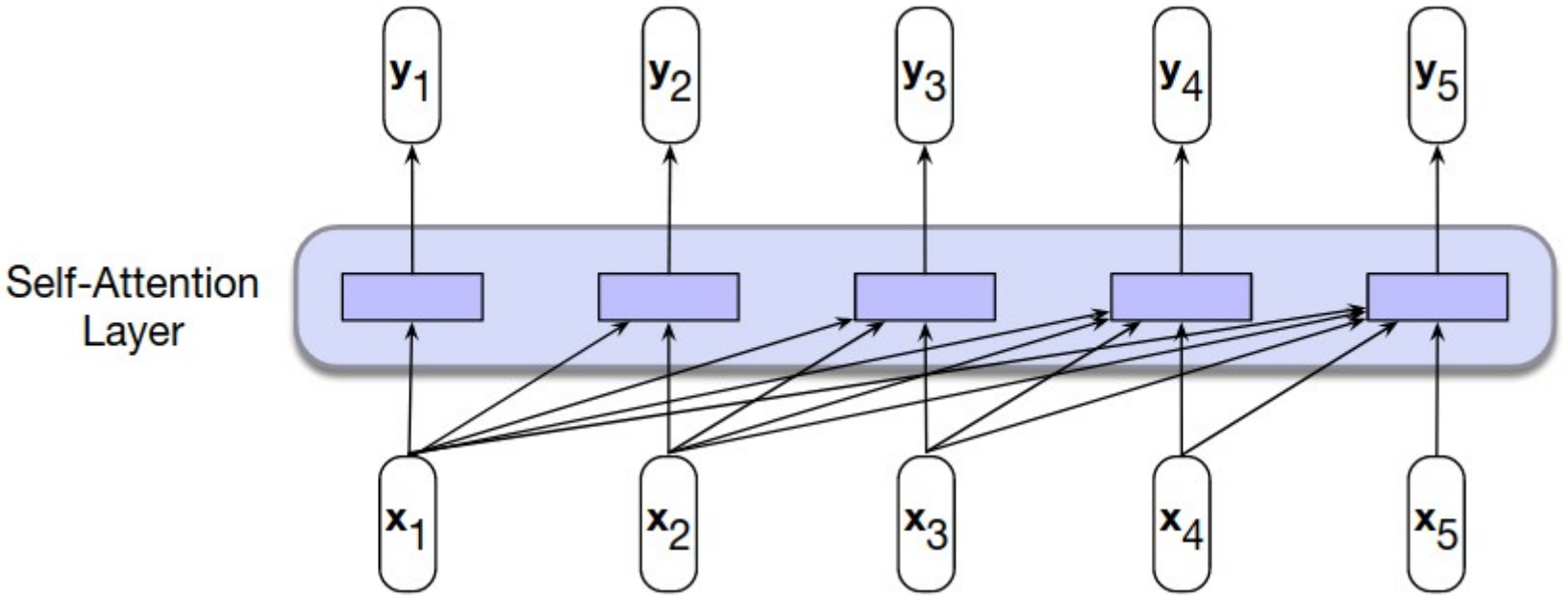
Reúne varias ideas exitosas

- self-attention
- positional embeddings
- contextualized embeddings
- fine-tuning
- masked language modeling
- next sentence prediction

El transformer puede procesar todas las entradas en paralelo.

Transformer: Self-Attention

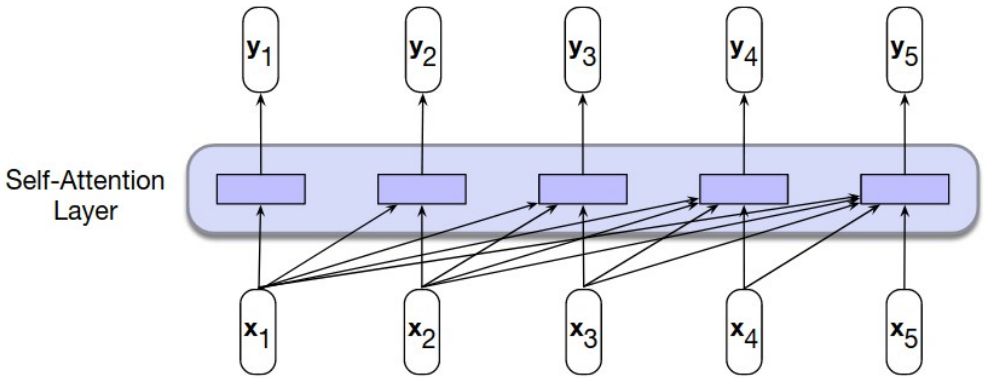
Transformer: Self-Attention



A diferencia de una RNN, cada paso es independiente (se puede paralelizar).

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

Transformer: Self-Attention



$$\mathbf{y}_i = \sum_{j \le i} \alpha_{ij} \mathbf{x}_j$$

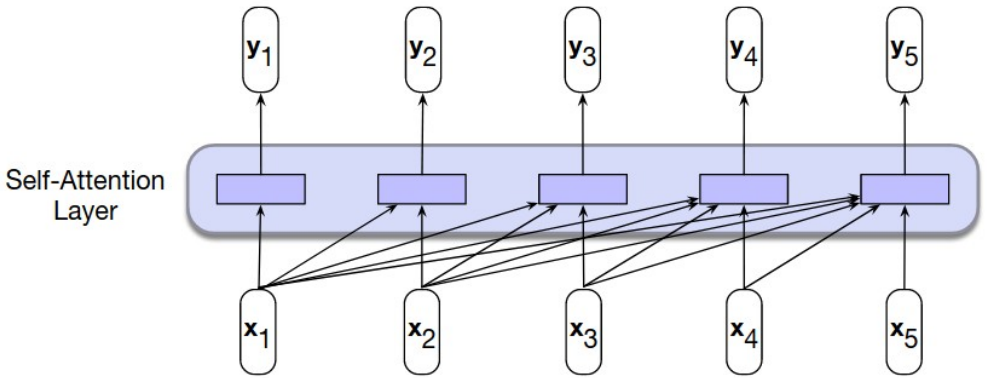
$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \;\; \forall j \le i$$

$$= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \;\; \forall j \le i$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \; ?$$

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

Transformer: Self-Attention



Se plantea *self-attention* con 3 proyecciones de cada x_i (**query**, **key** y **value**)

$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i; \;\; \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i; \;\; \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

Los vectores $\mathbf{q}_i$ y $\mathbf{k}_j$ computan **score** de j para i

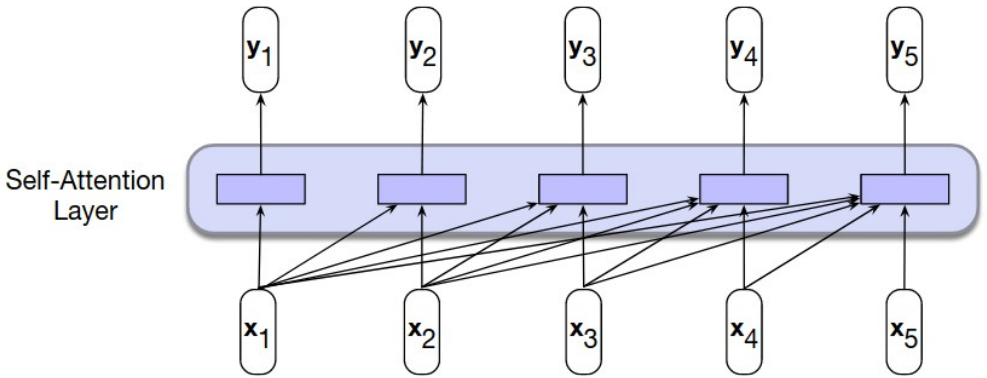
$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j$$

Con softmax se suman los $\mathbf{v}_i$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

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

Transformer: Self-Attention



Se plantea *self-attention* con 3 proyecciones de cada x_i (**query**, **key** y **value**)

$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i; \;\; \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i; \;\; \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

Los vectores $\mathbf{q}_i$ y $\mathbf{k}_j$ computan **score** de j para i

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

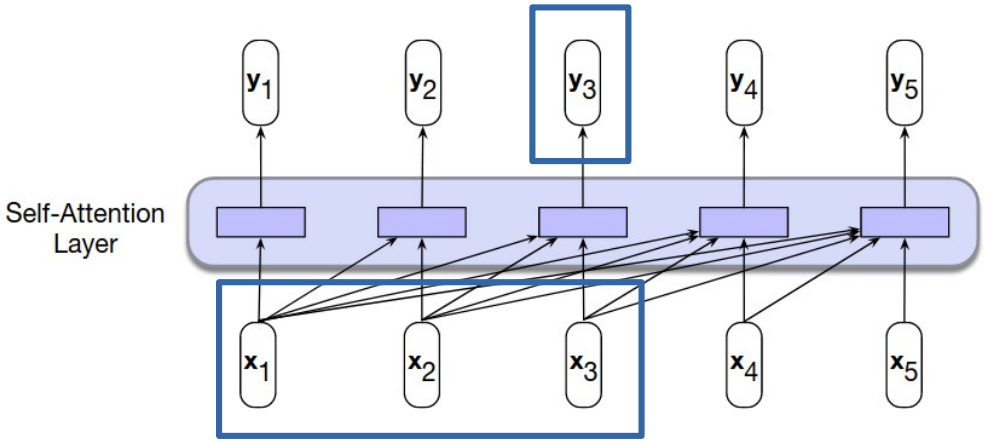
⇦ para suavizar valores grandes

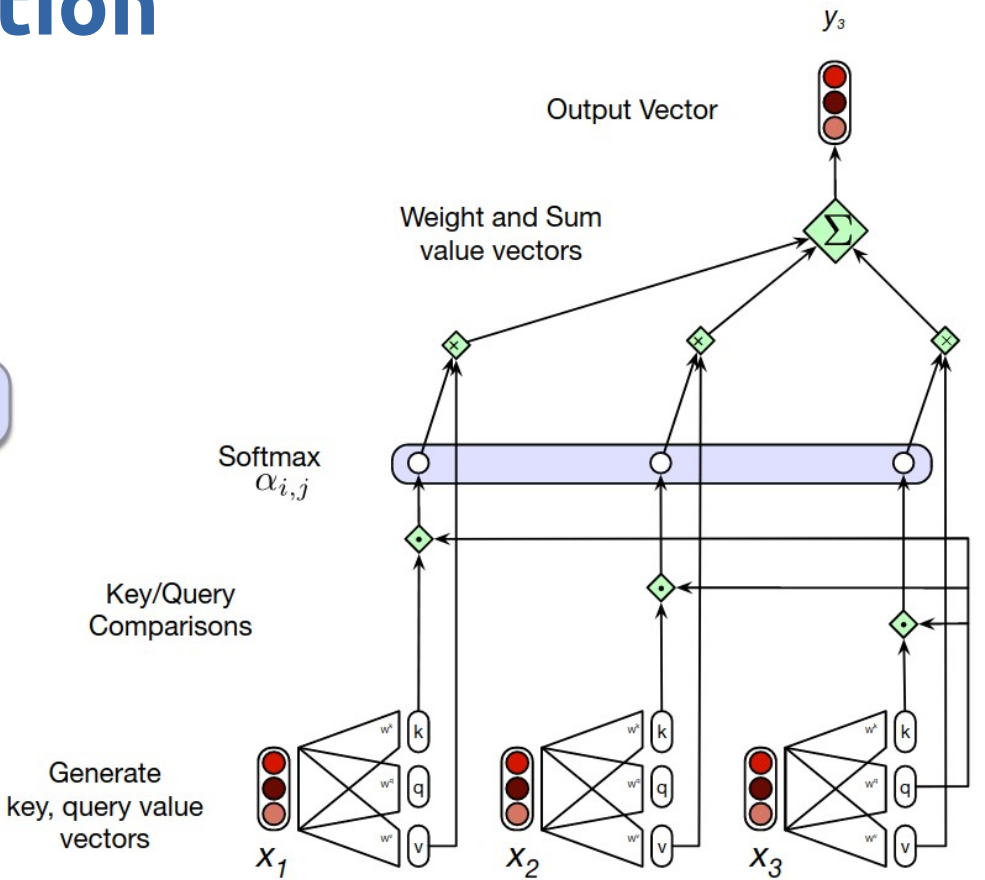
Con softmax se suman los $\mathbf{v}_i$

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

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

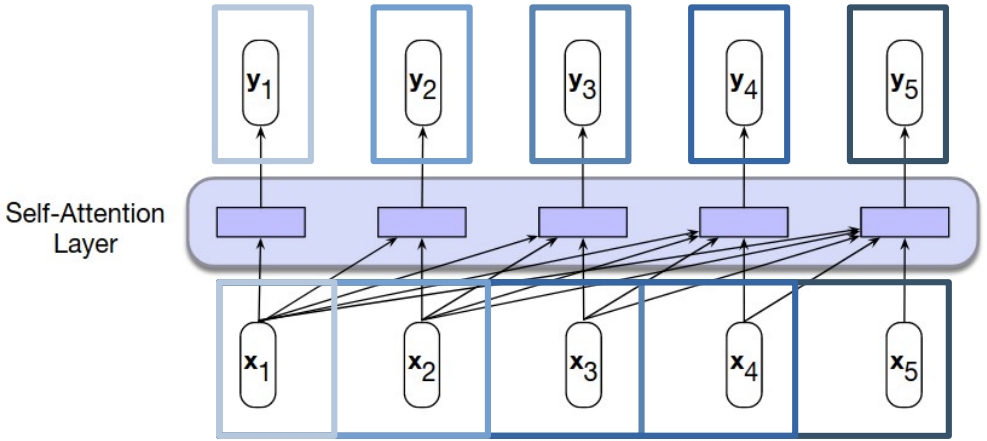
Transformer: Self-Attention





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

Transformer: Self-Attention



$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i;\ \ \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i;\ \ \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

Como cada paso es independiente podemos hacerlos en paralelo

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}};\ \ \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}};\ \ \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

… quedando

$$\mathit{SelfAttention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathsf{T}}}{\sqrt{d_k}}\right)\mathbf{V}$$

QK^T ⇨

q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

N

Se rellena con -∞ (se vuelven 0 con softmax)

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

Multihead Attention

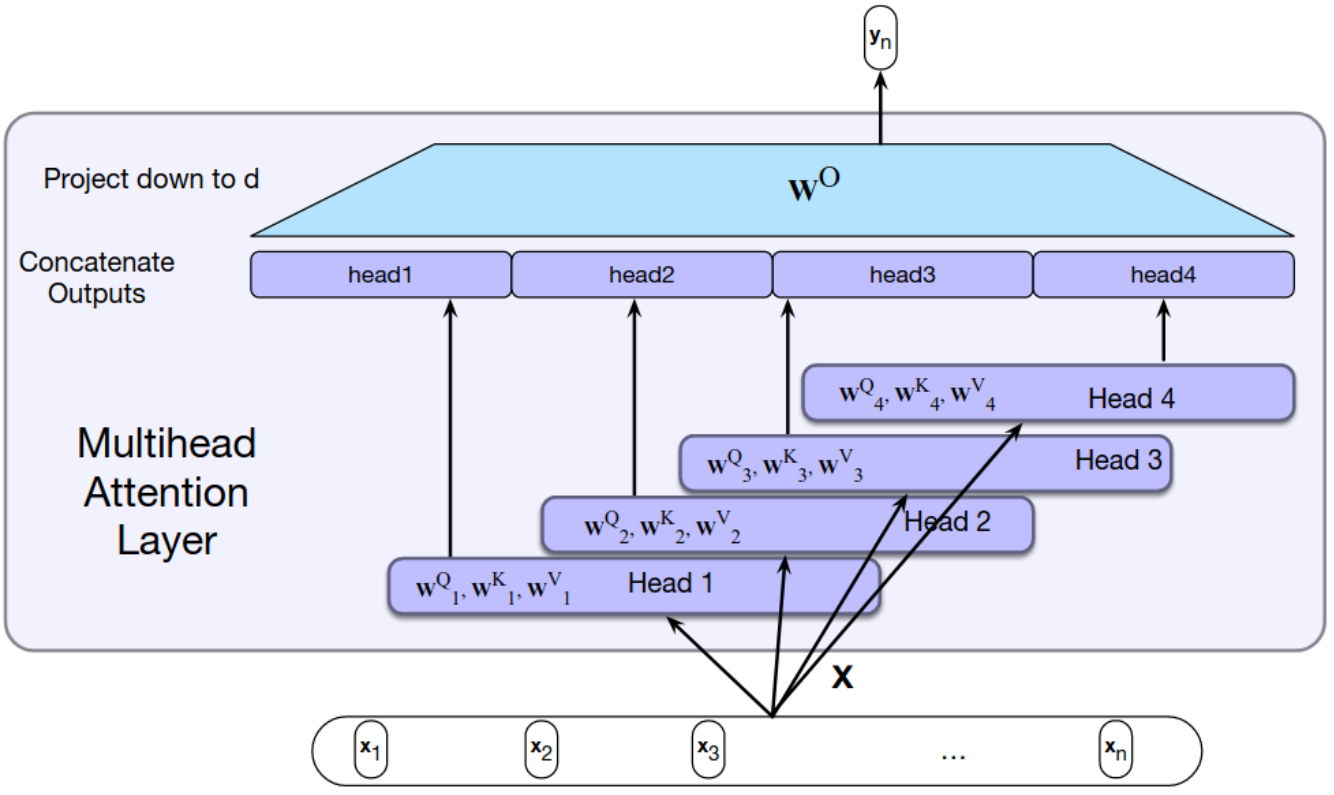
Multihead Attention

$$\mathbf{W}_i^Q \in \mathbb{R}^{d \times d_k},\ \mathbf{W}_i^K \in \mathbb{R}^{d \times d_k},\ \mathbf{W}_i^V \in \mathbb{R}^{d \times d_v}$$

$$MultiHeadAttn(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_h)\mathbf{W}^O$$

$$\mathbf{Q}_i = \mathbf{X}\mathbf{W}_i^Q\ ;\ \mathbf{K}_i = \mathbf{X}\mathbf{W}_i^K\ ;\ \mathbf{V}_i = \mathbf{X}\mathbf{W}_i^V$$

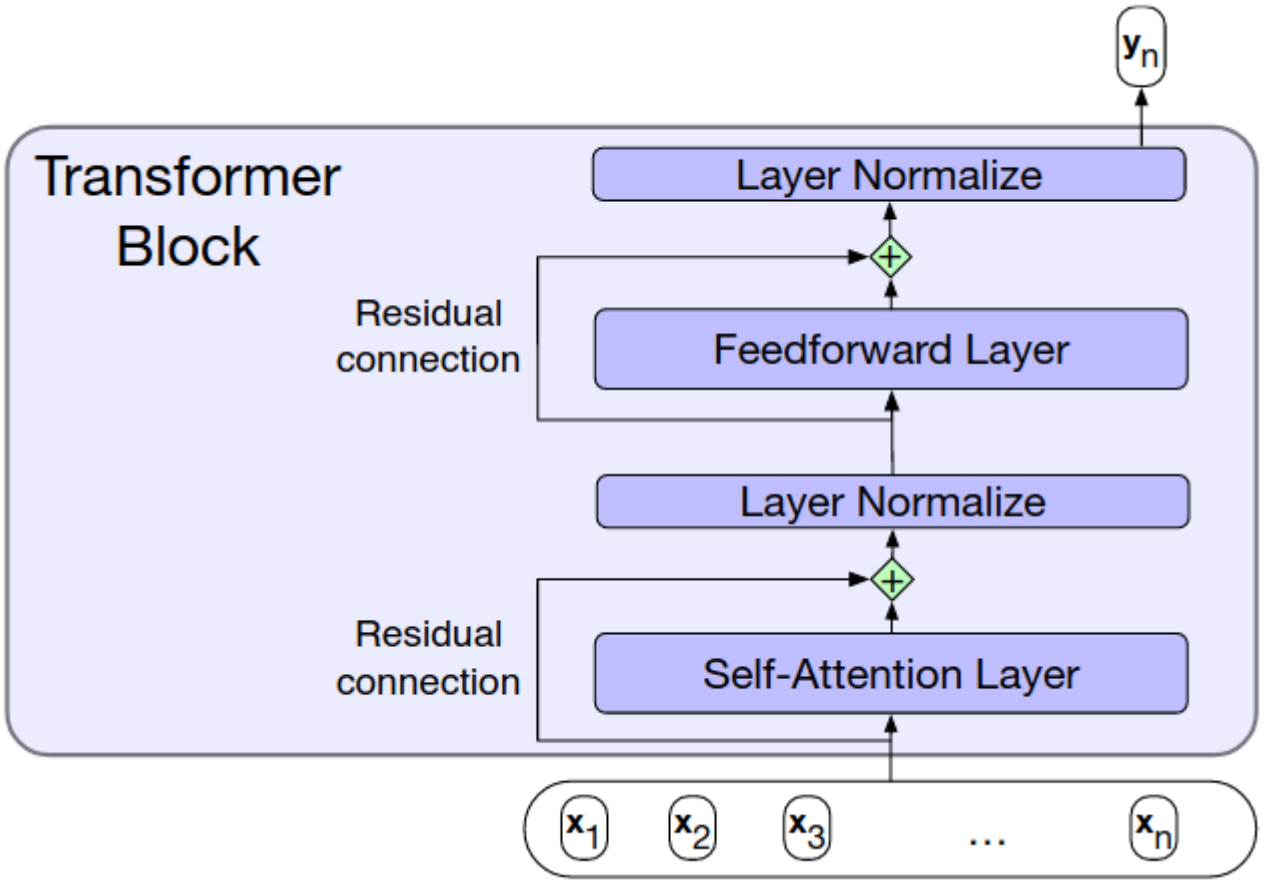
$$\mathbf{head}_i = SelfAttention(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i)$$



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

Transformer Block

Transformer Block



$$\mathbf{z} = \text{LayerNorm}(\mathbf{x} + \text{SelfAttn}(\mathbf{x}))$$
$$\mathbf{y} = \text{LayerNorm}(\mathbf{z} + \text{FFNN}(\mathbf{z}))$$

Layer Normalization

$$\mu = \frac{1}{d_h}\sum_{i=1}^{d_h} x_i$$

$$\sigma = \sqrt{\frac{1}{d_h}\sum_{i=1}^{d_h}(x_i - \mu)^2}$$

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

(γ y β se aprenden)

$$LayerNorm = \gamma\,\hat{\mathbf{x}} + \beta$$

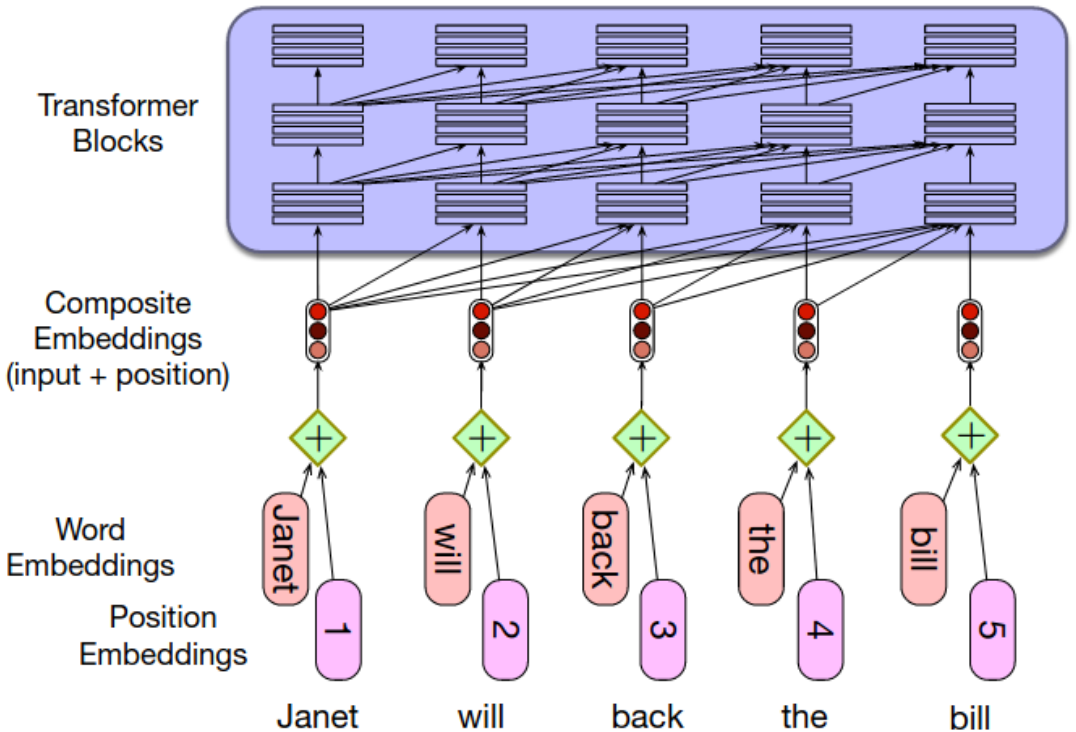
Basado en https://web.stanford.edu/~jurafsky/slp3/

Positional Embeddings

Word Order: Positional Embeddings

Hasta este punto no hay noción de orden en la entrada.

Solucion: Agregar a cada x_i un vector que represente su posición (**positional embeddings)**



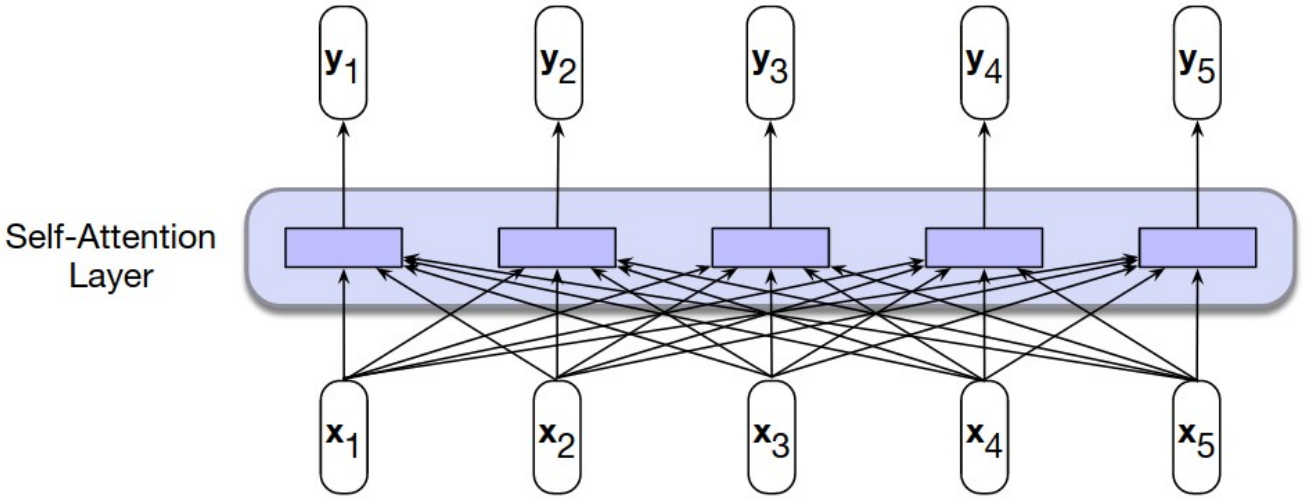
Pueden ser entrenados pero las primeras posiciones van a ser más comunes (por los diferentes largos), dificultando el entrenamiento.

Una altarnativa es que sean estáticos. Una combinación de senos y cosenos es usada en el transformer original.

Mejorar los positional embeddings es un área actual de investigación.

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

Bidirectional Transformer



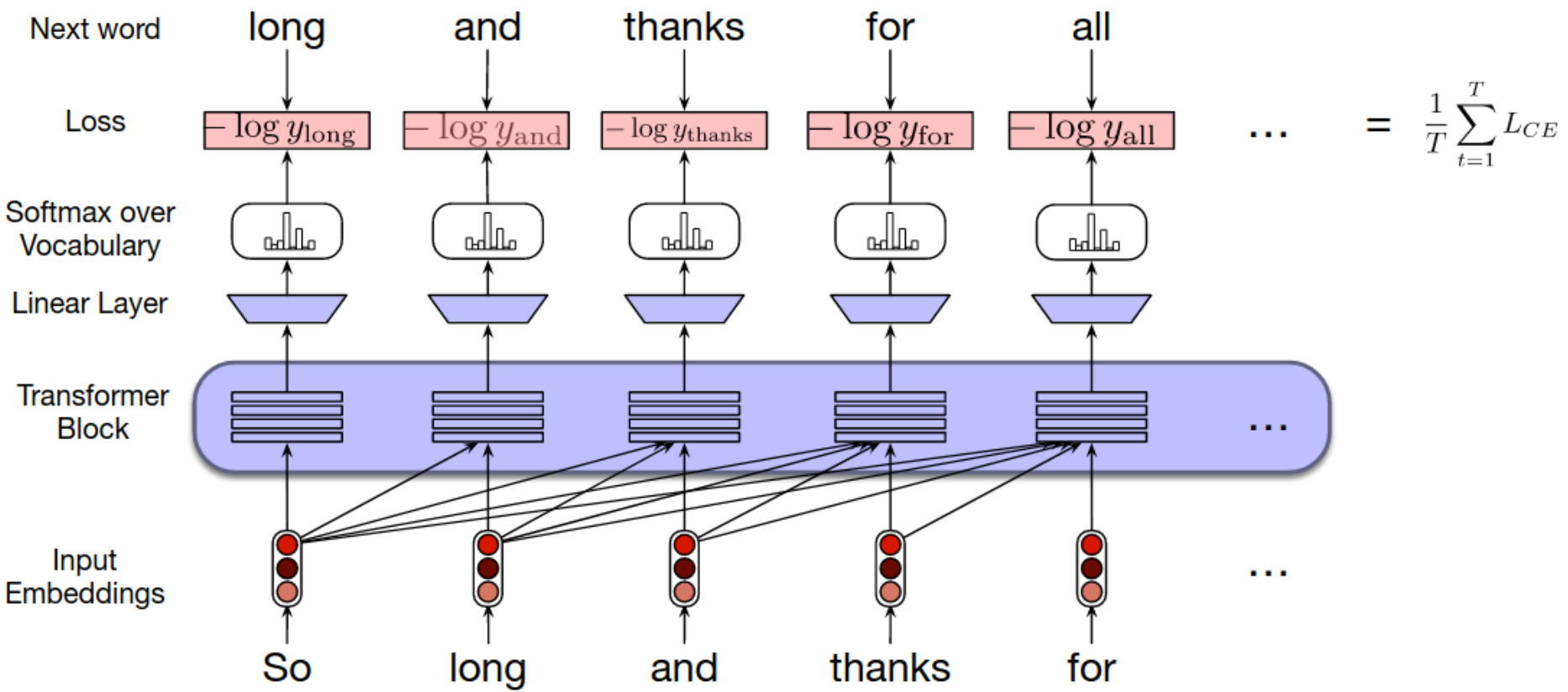
N

q1•k1	q1•k2	q1•k3	q1•k4	q1•k5
q2•k1	q2•k2	q2•k3	q2•k4	q2•k5
q3•k1	q3•k2	q3•k3	q3•k4	q3•k5
q4•k1	q4•k2	q4•k3	q4•k4	q4•k5
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

N

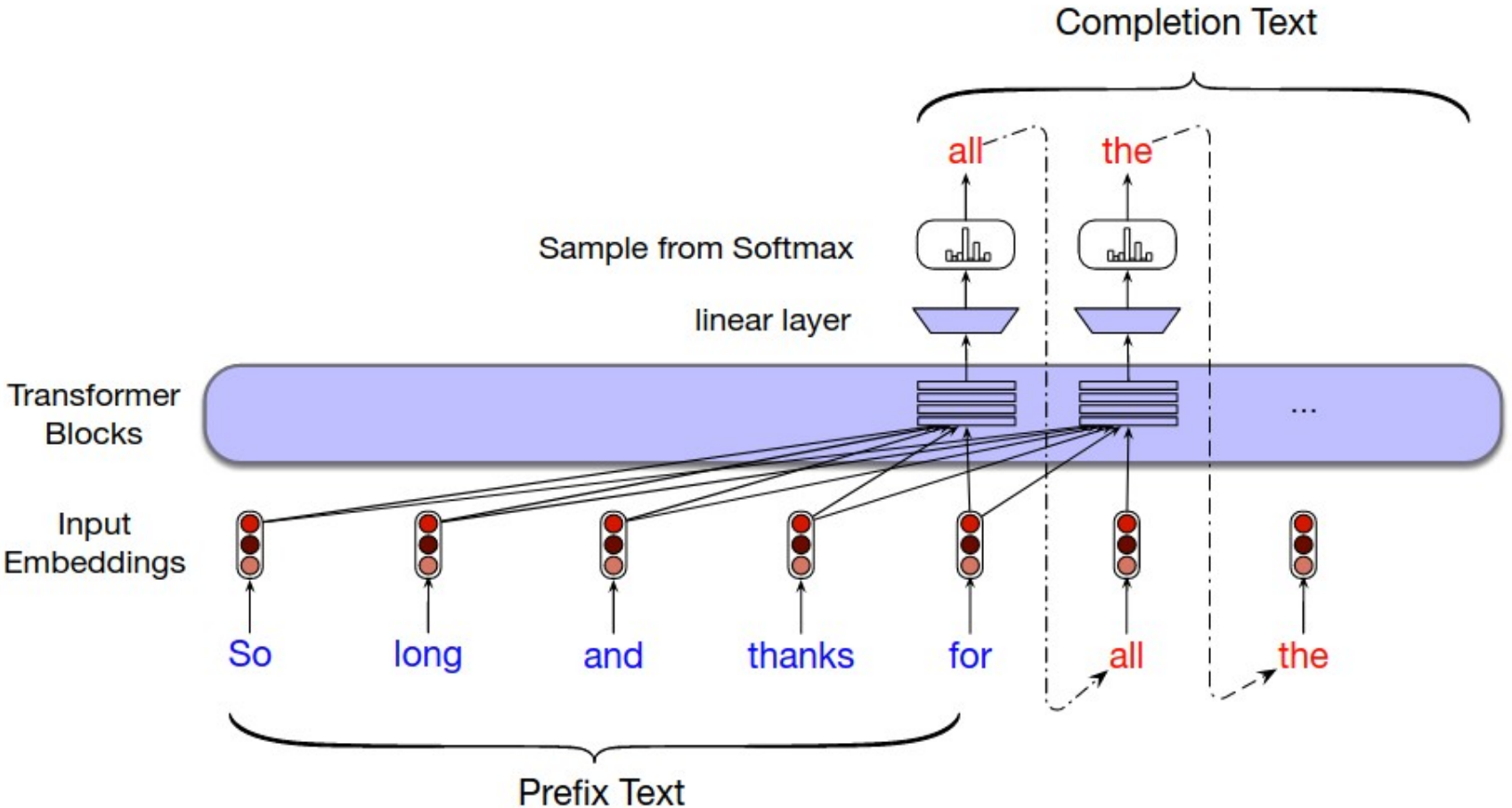
¿Cómo los usamos?

Language Model con Transformer



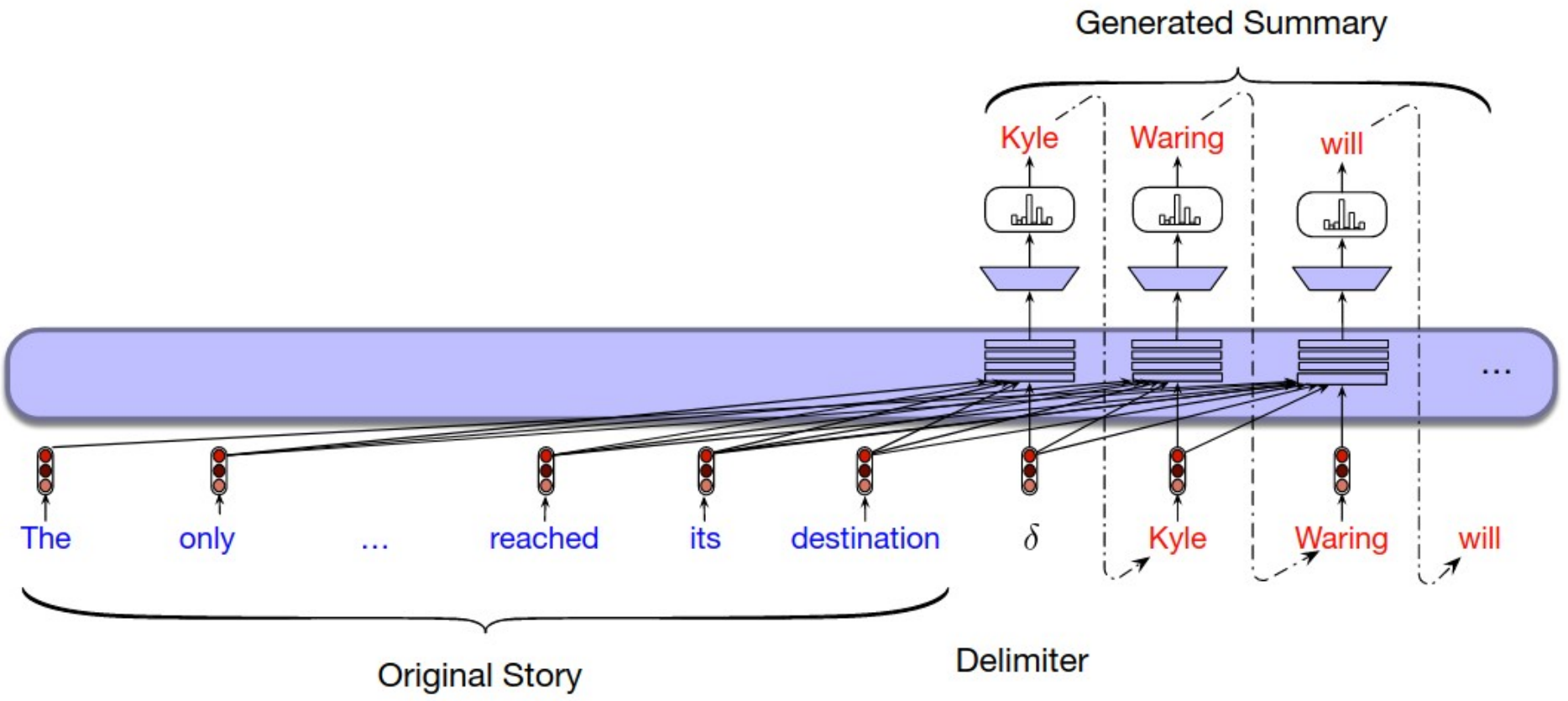
Basado en https://web.stanford.edu/~jurafsky/slp3/

Generar con Transformers



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

Conditioned Generation with Transformers



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

Pretraining y fine-tuning

Se preentrena un transformer como un modelo de lenguaje con un corpus grande **(pretraining)**

Luego, se agrega una capa feedforward que es ajustada con un corpus anotado **(fine-tuning)**