EEP590 Spring 2022

Deep Learning for Embedded Real Time Intelligence

Lecture 6: Neural Network Architectures for Timed Data Modeling: DTW, LSTM, GRU, Encoder-Decoder Seq2Seq, Attention, Transformer, BERT, GPT, ….

Prof. Richard Shi
Department of Electrical and Computer Engineering
(cjshi@uw.edu)

ELECTRICAL & COMPUTER ENGINEERING

Source Acknowledgement:

- **Deng Li, LSTM and GRU, Private Communication, 2017**
- **Nikolas Adaloglou, How Attention works in Deep Learning: understanding the attention mechanism in sequence models, blog** on2020-11-19·12 mins: https://theaisummer.com/attention/
- Vaswani et al. Attention is all you need. 2017
- https://nlp.seas.harvard.edu/2018/04/03/attention.html (Excellent explanation of transformer model with codes.)
- Jay Alammar, The illustrated transformer http://jalammar.github.io/illustrated-transformer/
- **Nir Arbel, How LSTM networks solve the problem of vanishing gradients:** A simple, straightforward mathematical explanation: https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577
- Daniel Gordon, Joseph Redmon and Ali Farhadi at UW CSE Course notes.
- https://databricks.com/blog/2019/04/30/understanding-dynamic-time-warping.html

ELECTRICAL & COMPUTER ENGINEERING

Source Acknowledgement:

- OpenAI GPT-2 implementation: https://github.com/openai/gpt-2
- BERT paper: J. Devlin et al, BERT, pretraining of deep bidirectional transformers for language understanding. Oct. 2018.
- ELMo paper: M. Peters, et al, Deep contextualized word representation, 2018
- ULM-FiT paper: Universal language model fine-tuning for text classification. J. Howeard, S. Ruder., 2018
- Jay Alammar, The illustrated GPT-2, https://jalammar.github.io/illustrated-gpt2/
- Esmaeil Alizadeh, **An Illustrative Introduction to Dynamic Time Warping:** **https://towardsdatascience.com/an-illustrative-introduction-to-dynamic-time-warping-36aa98513b98**
- ***Elena Tsiporkova,*** **Dynamic Time Warping Algorithm for Gene Expression Time Series**

ELECTRICAL & COMPUTER ENGINEERING

Outline

1. DTW: Similarity Measure of Time Series Data
2. RNN
3. LSTM and GRU
4. Encoder-Decoder
5. Attention
6. Transfomer
7. BERT
8. GPT

What you have talked about so far

- Basic Regression
- Image Classification
- Semantic Segmentation
- Object Detection

Something is missing...Time!

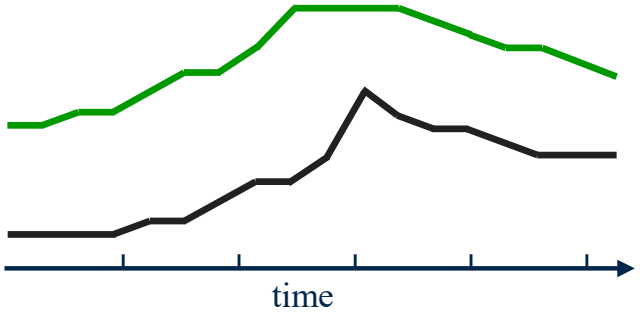
- Video Data
- Audio Data (Speech)
- Language Data
- Stock Markets???

How to Deal with Time: Some Options

- Summarization - Activity Detection?
  - Average/Max over all outputs
- Fixed Length Sequence? - Stock Market -- Keyword Spotting, Sezuire Detection
  - Sliding window
  - Padding
- Recurrent Neural Networks: RNNs!
- First, Let us Examine a Timed-Data Feature Extrator: Dynamic Time Warping (DTW)

Dynamic Time Warping (DTW)

- To compare two -usually temporal- sequences that do not sync up perfectly. For example, gene expression time series are expected to vary not only in terms of expression amplitudes, but also in terms of time progression since biological processes may unfold with different rates in response to different experimental conditions or within different organisms and individuals.



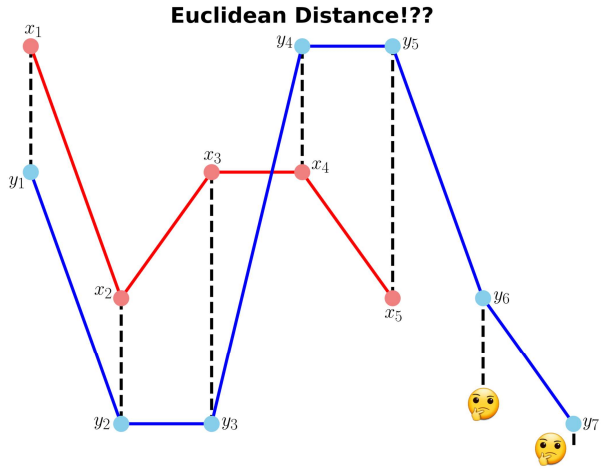
- To calculate the optimal matching between two sequences
- Useful in many domains such as speech recognition, data mining, genetic sequence, audio synchronization, speaker recognition, financial markets, etc
- Used in data mining to measure the distance between two time-series

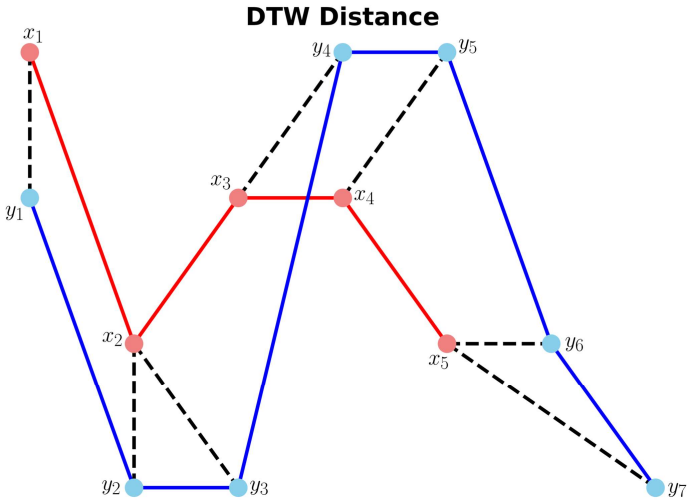
How to Measure the Similarity of Two Time Series?

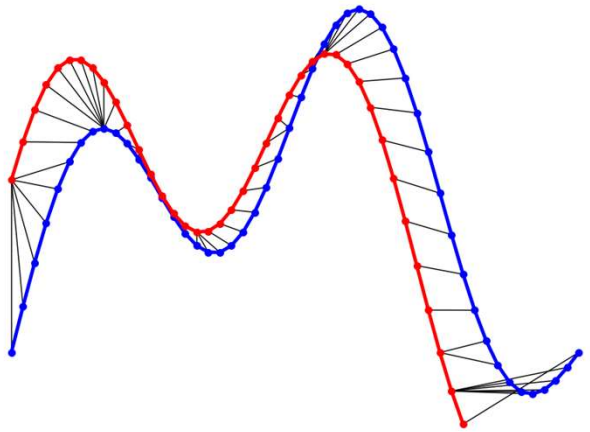
Two Sequences (X,Y):

$X=x[1], x[2], \ldots, x[i], \ldots, x[n]$

$Y=y[1], y[2], \ldots, y[j], \ldots, y[m]$

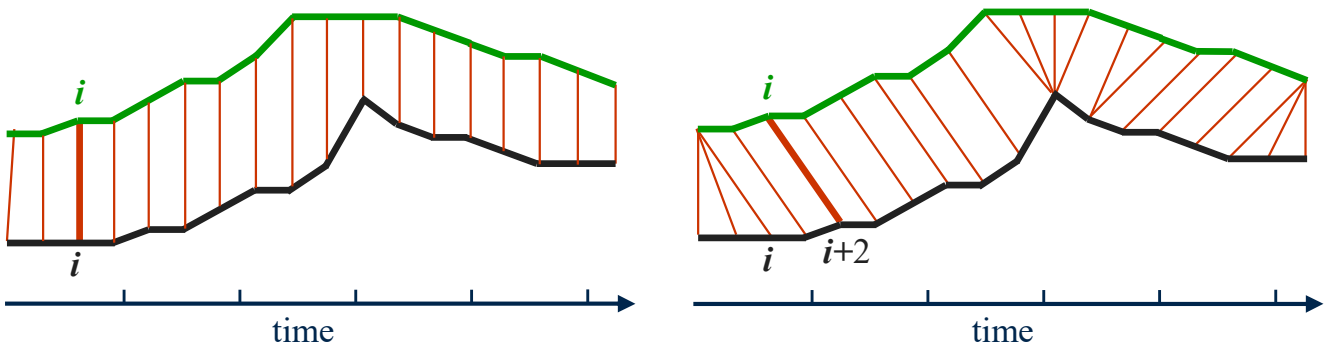






ELECTRICAL & COMPUTER ENGINEERING

Why DTW for Measuring the Time Serie Similarity?



Any distance (Euclidean, Manhattan, …) which aligns the *i*-th point on one time series with the *i*-th point on the other will produce a poor similarity score.

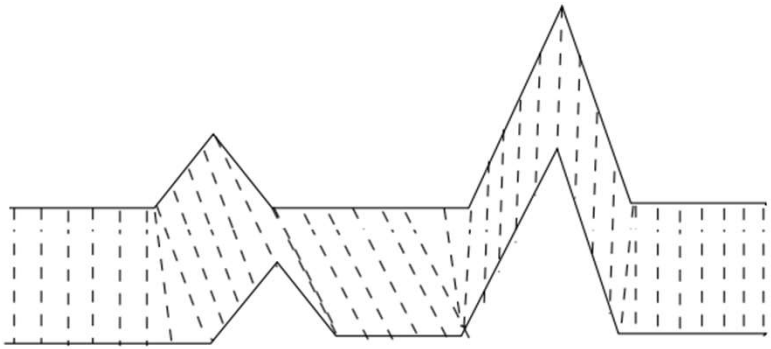
A non-linear (elastic) alignment produces a more intuitive similarity measure, allowing similar shapes to match even if they are out of phase in the time axis.

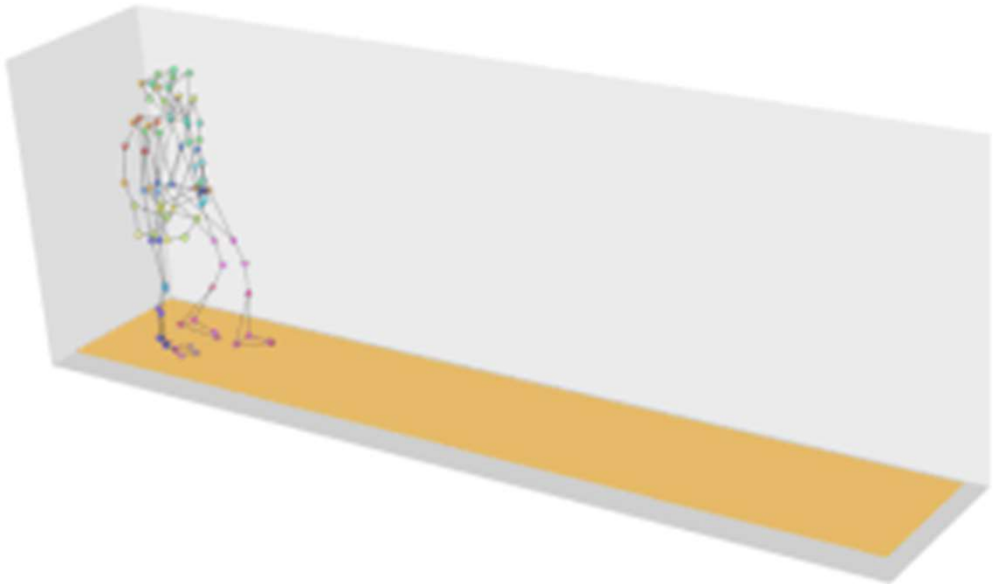
How to Measure the Similarity of Two Time Series?

Two Sequences (X,Y):

$X=x[1], x[2], \ldots, x[i], \ldots, x[n]$

$Y=y[1], y[2], \ldots, y[j], \ldots, y[m]$



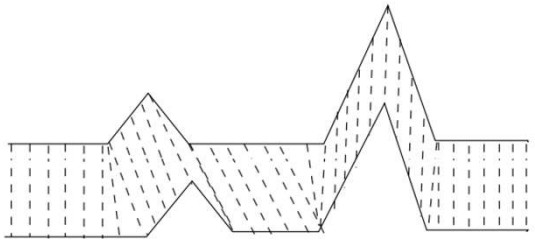


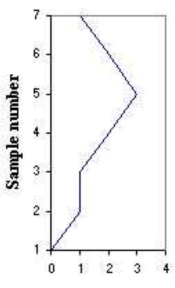
Two repetitions of a walking sequence recorded using a motion-capture system. While there are differences in walking speed between repetitions, the spatial paths of limbs remain highly similar.[1]
Pavel Senin, Dynamic Time Warping Algorithm Review

How to Measure the Similarity of Two Time Series?

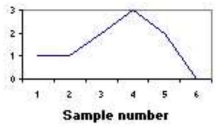
Divide the two series into equal points.

1. Calculate the euclidean distance between the first point in the first series and every point in the second series. Store the minimum distance calculated. (this is the 'time warp' stage)
2. Move to the second point and repeat 2. Move step by step along points and repeat 2 till all points are exhausted.
3. Repeat 2 and 3 but with the second series as a reference point.
4. Add up all the minimum distances that were stored and this is a true measure of similarity between the two series.





1	0	0	1	2	1	-1
2	-1	-1	0	1	0	-2
3	-2	-2	-1	0	-1	-3
2	-1	-1	0	1	0	-2
1	0	0	1	2	1	-1
1	0	0	1	2	1	-1
0	1	1	2	3	2	0
	1	1	2	3	2	0



x[t]

y[t]

1	0	0	1	4	1	1
2	1	1	0	1	0	4
3	4	4	1	0	1	9
2	1	1	0	1	0	4
1	0	0	1	4	1	1
1	0	0	1	4	1	1
0	1	1	4	9	4	0
	1	1	2	3	2	0

x[t]

ELECTRICAL & COMPUTER ENGINEERING

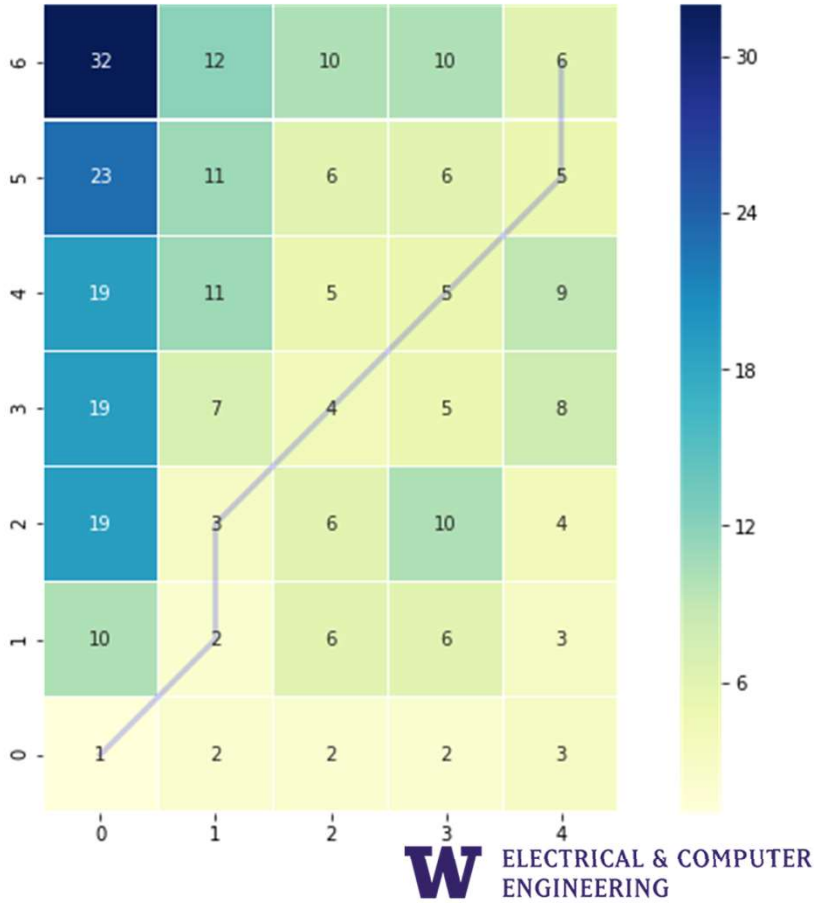
How to Measure the Similarity of two Time Series?

- The sequences X and Y can be arranged to form an n-by-m grid, where each point (i, j) is the alignment between $x[i]$ and $y[j]$.
- A warping path W maps the elements of X and Y to minimize the *distance* between them. W is a sequence of grid points (i, j).
- The Optimal path to (i_k, j_k) can be computed by dynamic programming:

$$D_{min}(i_k, j_k) = \min_{i_{k-1}, j_{k-1}} D_{min}(i_{k-1}, j_{k-1}) + d(i_k, j_k | i_{k-1}, j_{k-1})$$

- Path cost: $D = \sum_k d(i_k, j_k)$

x = [3, 1, 2, 2, 1]
y = [2, 0, 0, 3, 3, 1, 0]



ELECTRICAL & COMPUTER ENGINEERING

How to Measure the Similarity of two Time Series?

$$D_{min}(i_k, j_k) = \min_{i_{k-1}, j_{k-1}} D_{min}(i_{k-1}, j_{k-1}) + d(i_k, j_k | i_{k-1}, j_{k-1})$$

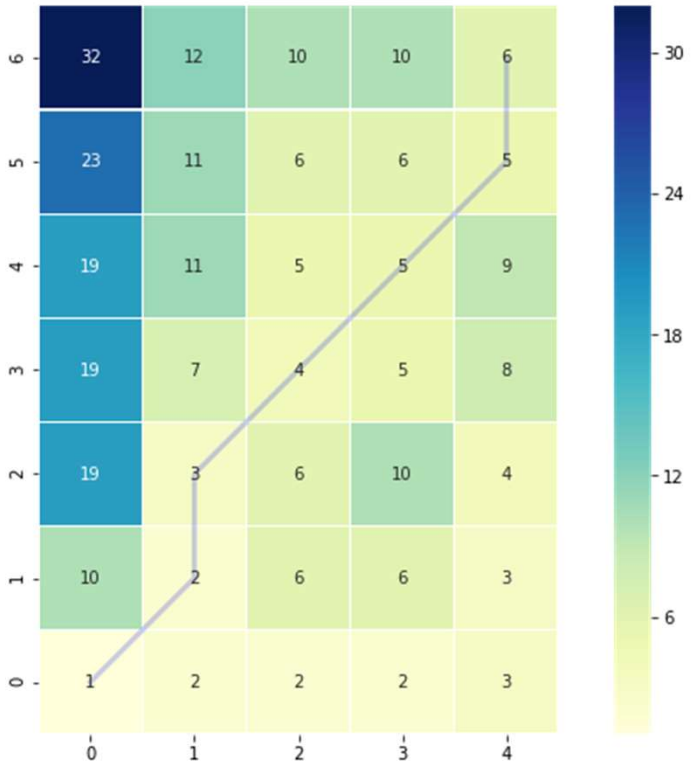
This example illustrates the implementation of the dynamic time warping algorithm when the two sequences *s* and *t* are strings of discrete symbols. For two symbols *x* and *y*, `d(x, y)` is a distance between the symbols, e.g. `d(x, y)` = .

```
int DTWDistance(s: array [1..n], t: array [1..m])
{
    DTW := array [0..n, 0..m]
    for i := 0 to n
        for j := 0 to m
            DTW[i, j] := infinity
     DTW[0, 0] := 0
     for i := 1 to n
        for j := 1 to m
            cost := d(s[i], t[j])
            DTW[i, j] := cost + minimum(DTW[i-1, j ], // insertion
                                        DTW[i , j-1], // deletion
                                         DTW[i-1, j-1]) // match

     return DTW[n, m]
}
```

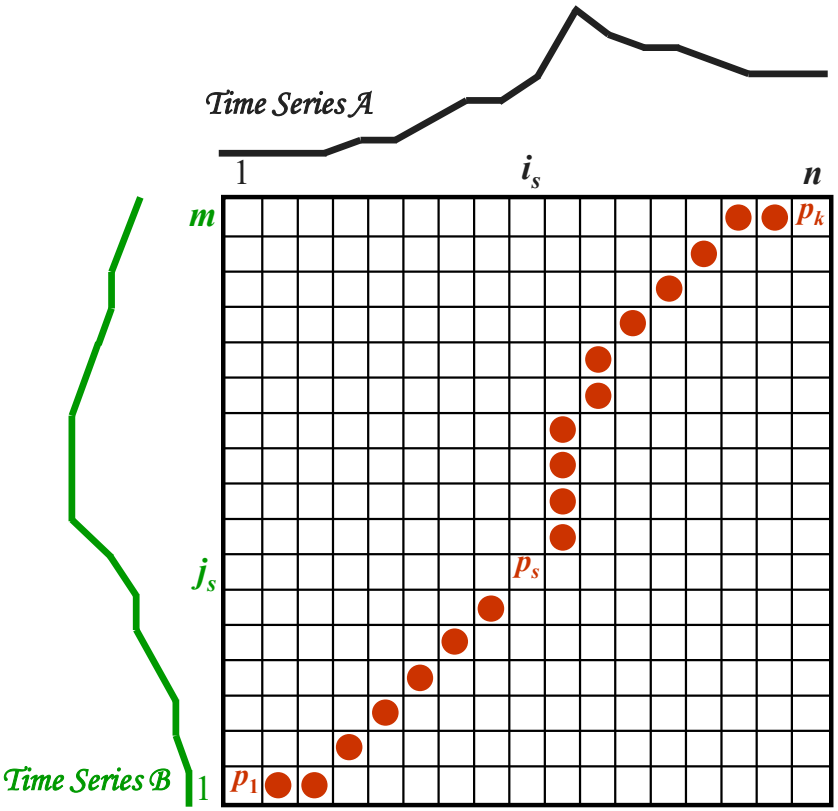
x = [3, 1, 2, 2, 1]

y = [2, 0, 0, 3, 3, 1, 0]



ELECTRICAL & COMPUTER ENGINEERING

Warping Function



To find the *best alignment* between A and B one needs to find the path through the grid

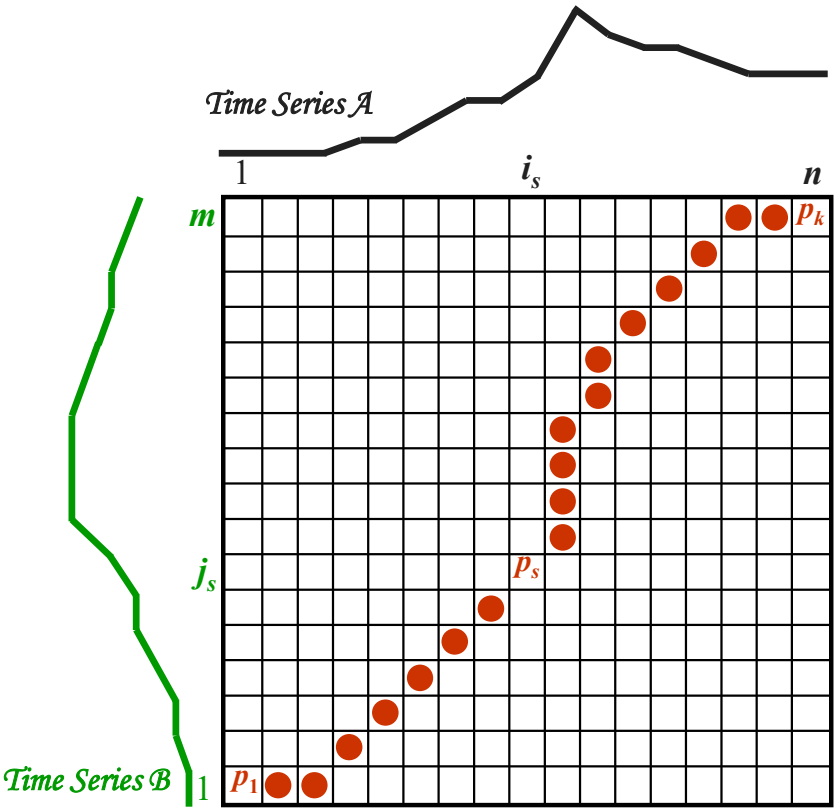
$P = p_1, \ldots, p_s, \ldots, p_k$

$p_s = (i_s, j_s)$

which *minimizes* the total distance between them.

P is called a *warping function*.

Time-Normalized Distance Measure



Time-normalized distance between A and B:

$$D(A, B) = \left[\frac{\sum_{s=1}^{k} d(p_s) \cdot w_s}{\sum_{s=1}^{k} w_s} \right]$$

$d(p_s)$: distance between i_s and j_s

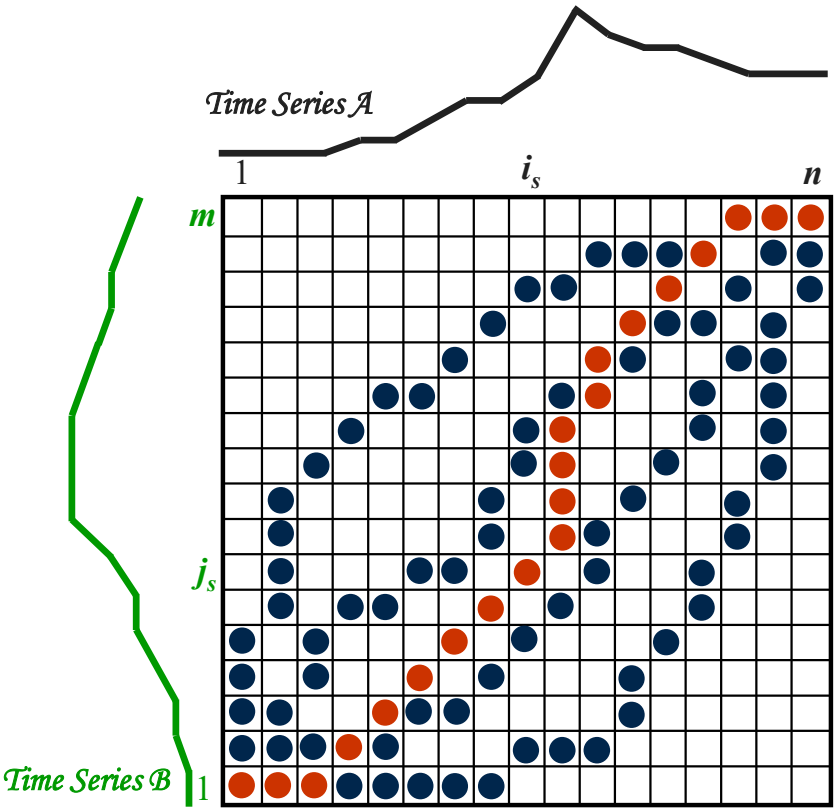
$w_s > 0$: weighting coefficient.

Best alignment path between A and B:

$P_0 = \arg\min_P (D(A, B))$.

ELECTRICAL & COMPUTER ENGINEERING

Optimisations to the DTW Algorithm



The number of possible warping paths through the grid is exponentially explosive!

reduction of the search space

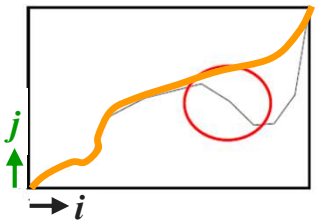
Restrictions on the warping function:

- monotonicity
- continuity
- boundary conditions
- warping window
- slope constraint.

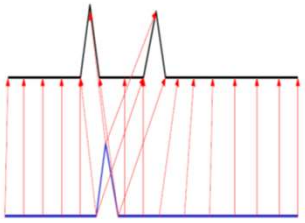
Restrictions on the Warping Function

Monotonicity: $i_{s-1} \le i_s$ and $j_{s-1} \le j_s$.

The alignment path does not go back in "*time*" index.

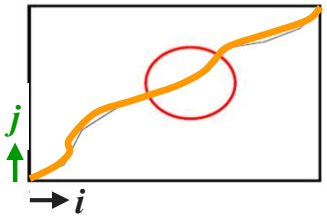


Guarantees that features are not repeated in the alignment.

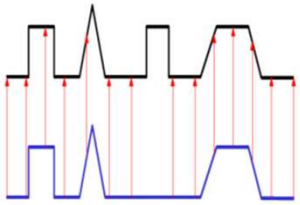


Continuity: $i_s - i_{s-1} \le 1$ and $j_s - j_{s-1} \le 1$.

The alignment path does not jump in "*time*" index.



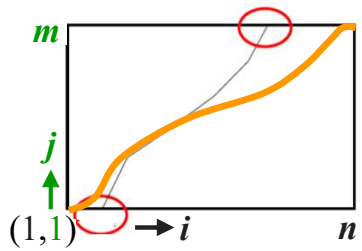
Guarantees that the alignment does not omit important features.



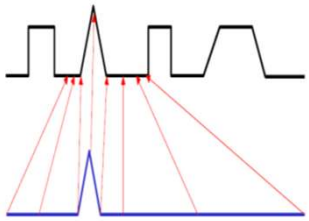
Restrictions on the Warping Function

Boundary Conditions: $i_1 = 1$, $i_k = n$ and $j_1 = 1$, $j_k = m$.

The alignment path starts at the bottom left and ends at the top right.

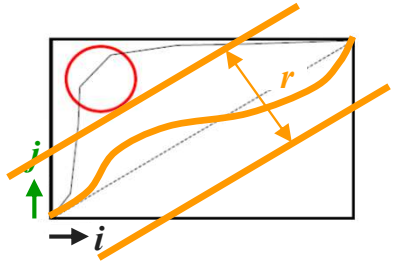


Guarantees that the alignment does not consider partially one of the sequences.

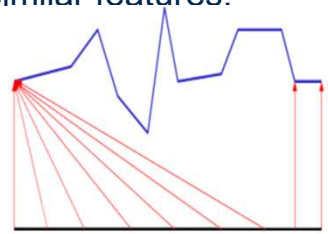


Warping Window: $|i_s - j_s| \leq r$, where $r > 0$ is the window length.

A good alignment path is unlikely to wander too far from the diagonal.



Guarantees that the alignment does not try to skip different features and gets stuck at similar features.

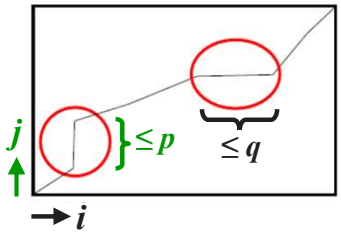


Restrictions on the Warping Function

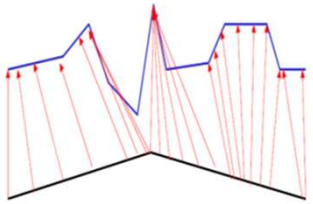
Slope Constraint: $(j_{s_p} - j_{s_0}) / (i_{s_p} - i_{s_0}) \leq p$ and $(i_{s_q} - i_{s_0}) / (j_{s_q} - j_{s_0}) \leq q$, where $q \geq 0$ is the number of steps in the x-direction and $p \geq 0$ is the number of steps in the y-direction.

After q steps in x one must step in y and vice versa: $S = p / q \in [0, \infty]$.

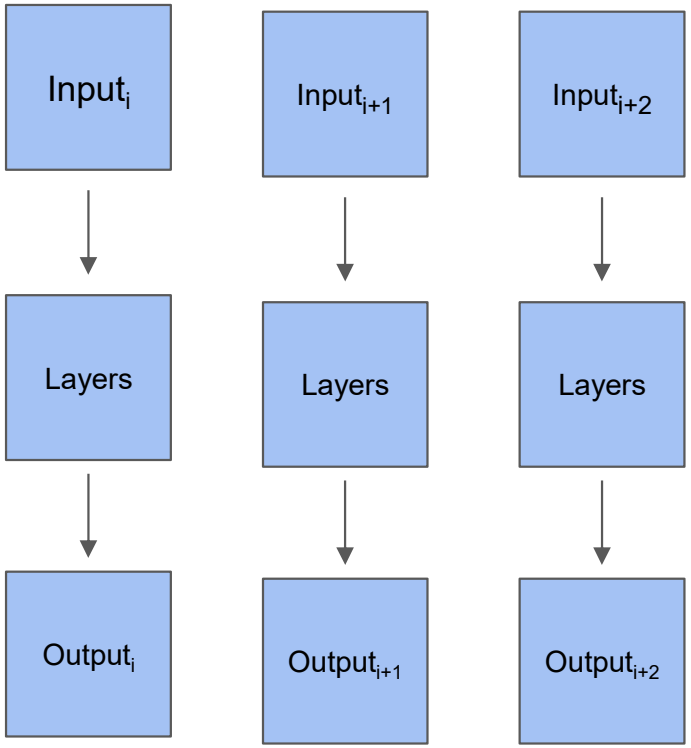
The alignment path should not be too steep or too shallow.



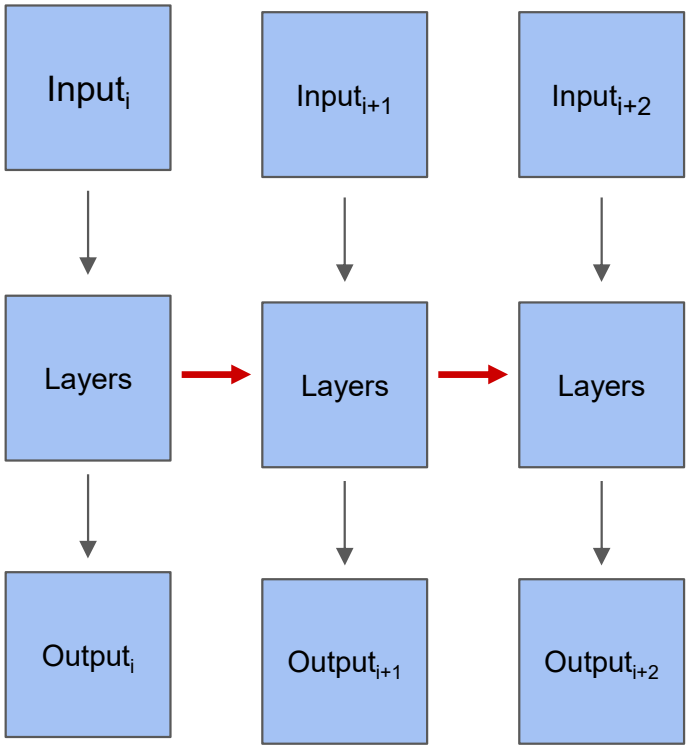
Prevents that very short parts of the sequences are matched to very long ones.



CNN


Input$_i$
Input$_{i+1}$
Input$_{i+2}$
Layers
Layers
Layers
Output$_i$
Output$_{i+1}$
Output$_{i+2}$

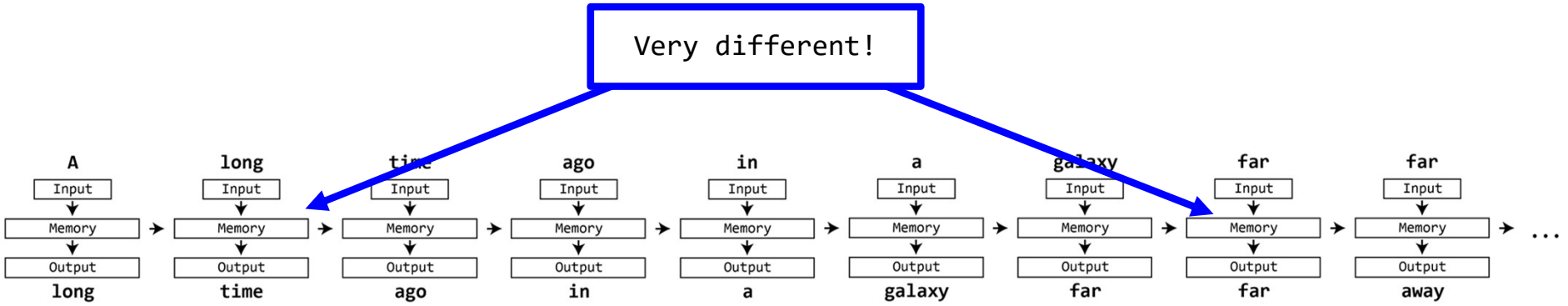
RNN


Input$_i$
Input$_{i+1}$
Input$_{i+2}$
Layers
Layers
Layers
Output$_i$
Output$_{i+1}$
Output$_{i+2}$

Problems?

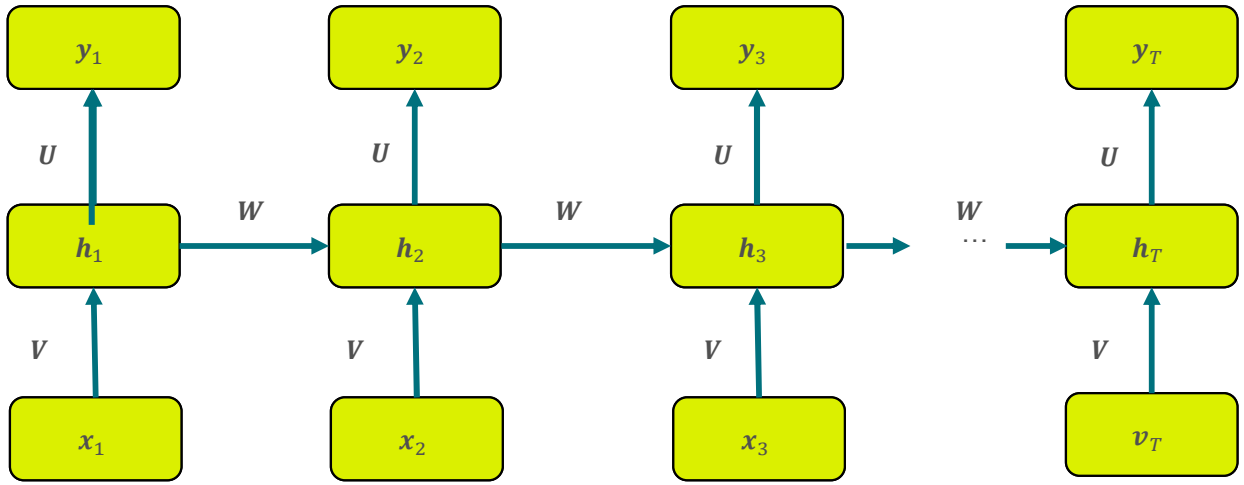
Memory computed from scratch every round

Hard to remember things for a long time.



Basic RNN Architecture

h_t is the hidden layer that carries the information from time $0 \sim t$
where x_t: *the input word* , y_t: *the output tag*
$y_t = SoftMax(U \cdot h_t), where\ h_t = \sigma(W \cdot h_{t-1} + V \cdot x_t)$

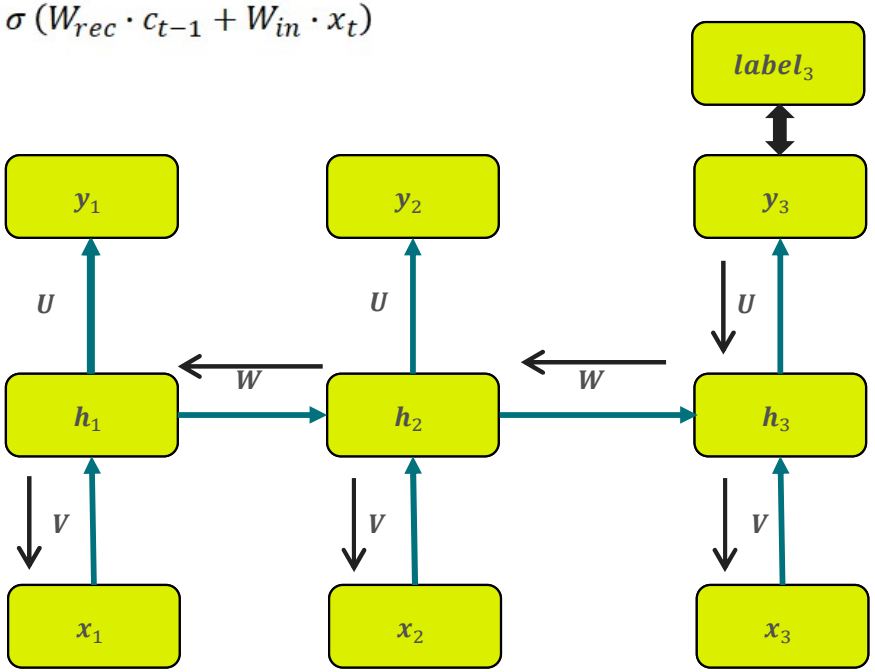


Used for slot filling in SLU

[Mesnil, He, Deng, Bengio, 2013; Yao, Zweig, Hwang, Shi, Yu, 2013]

Back-Propagation Through Time (BPTT)

$$c_t = \sigma\,(W_{rec} \cdot c_{t-1} + W_{in} \cdot x_t)$$



$$W \leftarrow W - \alpha \frac{\partial E}{\partial W}$$

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

at time $t = 3$

1. Forward propagation
2. Generate output
3. Calculate error
4. Back propagation
5. Back prop. through time

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}\cdots\frac{\partial c_2}{\partial c_1}\frac{\partial c_1}{\partial W}$$

$$= \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}\left(\prod_{t=2}^{k}\frac{\partial c_t}{\partial c_{t-1}}\right)\frac{\partial c_1}{\partial W} \qquad (1)$$

$$\frac{\partial C_t}{\partial C_{t-1}} = \sigma'(W_{rec} \cdot c_{t-1} + W_{in} \cdot x_t) \cdot \frac{\partial}{\partial C_{t-1}}[\,W_{rec} \cdot c_{t-1} + W_{in} \cdot x_t]$$

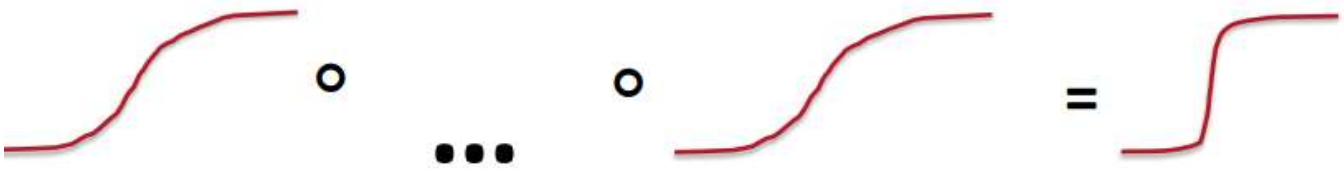
$$= \sigma'(W_{rec} \cdot c_{t-1} + W_{in} \cdot x_t) \cdot W_{rec} \qquad (2)$$

$$\prod_{t=2}^{k} \sigma'(W_{rec} \cdot c_{t-1} + W_{in} \cdot x_t) \cdot W_{rec} \to 0 \qquad \frac{\partial E_k}{\partial W} \to 0$$

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} \to 0$$

Some Early Attempts to Examine Difficulties in Learning RNNs

- Yoshua Bengio's Ph.D. thesis at McGill University (1991)
- Based on attractor properties of nonlinear dynamic systems
- His recent, more intuitive explanation --- in extreme of nonlinearity, discrete functions and gradients vanish or explode:



Problems?

Gradient of tanh(x)

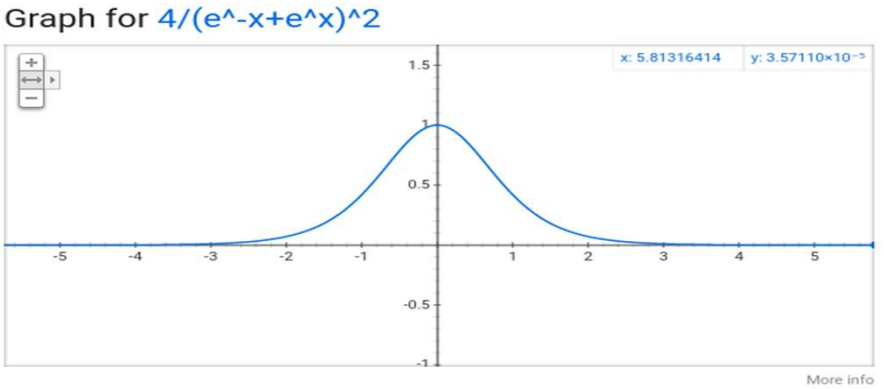
Between 0 and 1

d/dx $\tanh^{n}(x) = 0$

ReLU(x)
Gradient = 0 or 1

x^2

Gradient = 2x



Vanishing Gradient

Long multiplication of (smaller than 1) terms

Exploding Gradient

ABS(x)
Gradient = -1 or 1

ELECTRICAL & COMPUTER ENGINEERING

What about Residual Connections

Yes! Sort of.

Separate piece of memory which is never squashed

Control the flow of this memory with gates

An Important Gaget: Gating

f(x) = a * x where a is between 0 (forgot) and 1

f(x) = a * x + (1 - a) * m

Allows for partial updates of memory

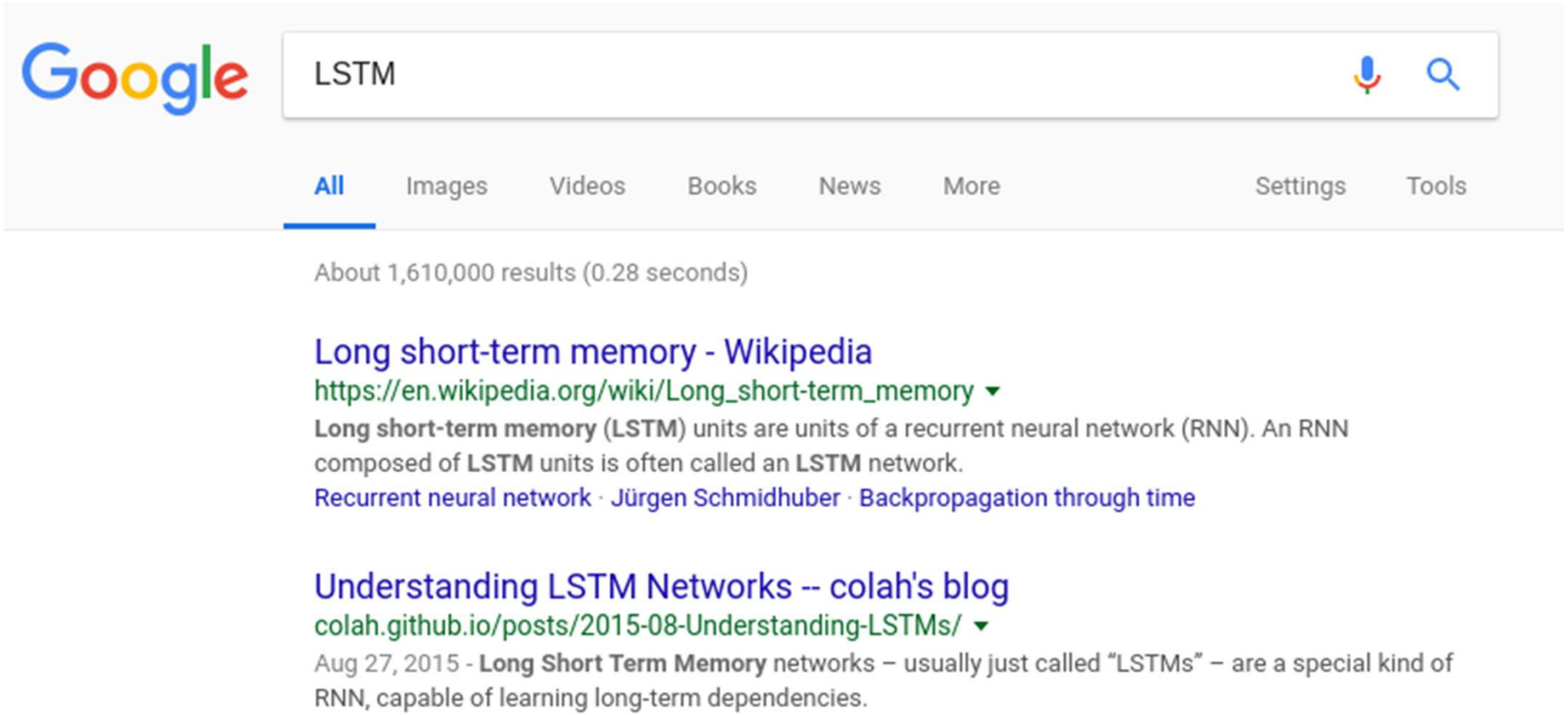
No squashing function

Deravative not approaching 0

How to Rescue: LSTM

- Use of LSTM (long short-term memory) cells
  - Sepp Hochreiter & Jürgen Schmidhuber (1997). "Long short-term memory" *Neural Computation* **9** (8): 1735–1780.
  - Many earlier-to-read materials, especially after 2013
- The best way so far to train RNNs well
  - Increasingly popular in speech/language processing
  - Attracted big attention from ASR community at ICASSP-2013's DNN special session (Graves et al.)
  - Huge progress since then

Long Short Term Memory



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

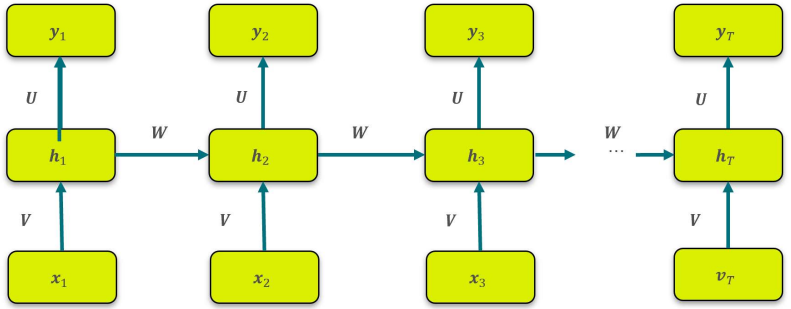
LSTM

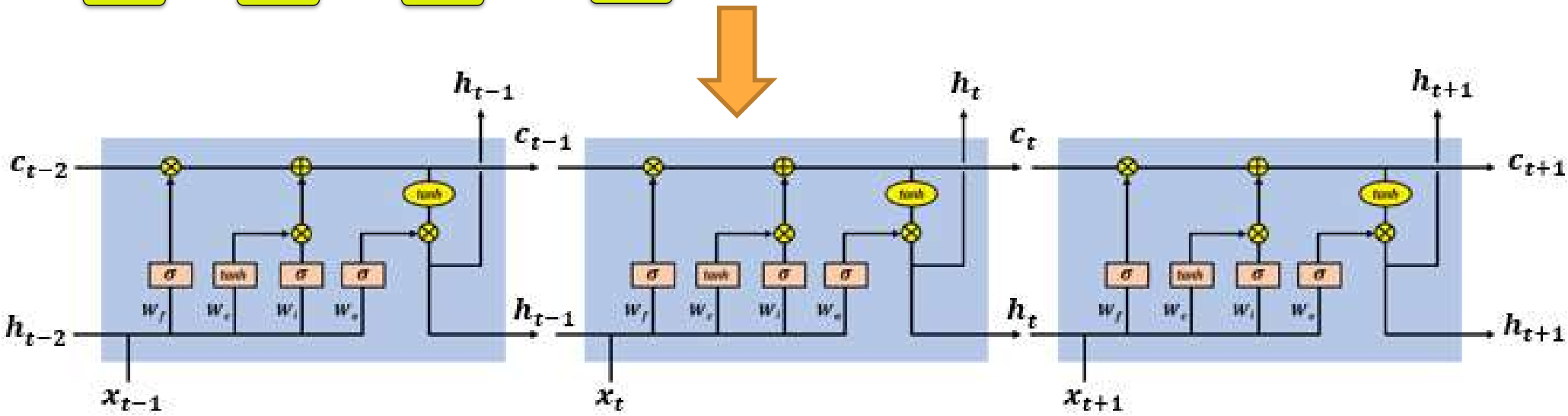
From the Vanina RNN to LSTM Architecture

h_t is the hidden layer that carries the information from time $0 \sim t$

where x_t: *the input word* , y_t: *the output tag*

$y_t = SoftMax(U \cdot h_t)$, *where* $h_t = \sigma(W \cdot h_{t-1} + V \cdot x_t)$

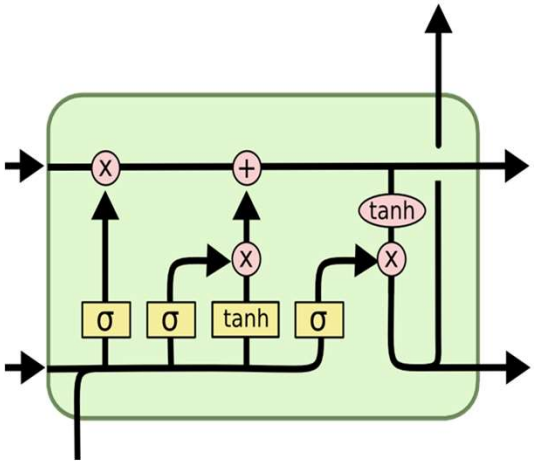




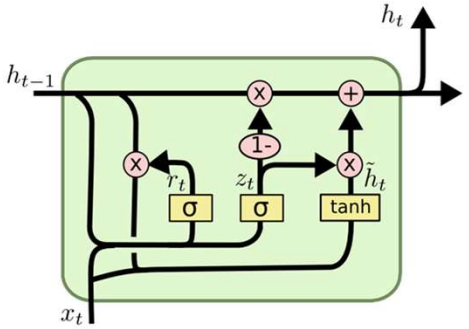
LSTM and GRU: The Same Idea using “Gates”

LSTM: Long Short Term Memory

GRU: Gated Recurrent Unit: Roughly equivalent performance, with about half parameters



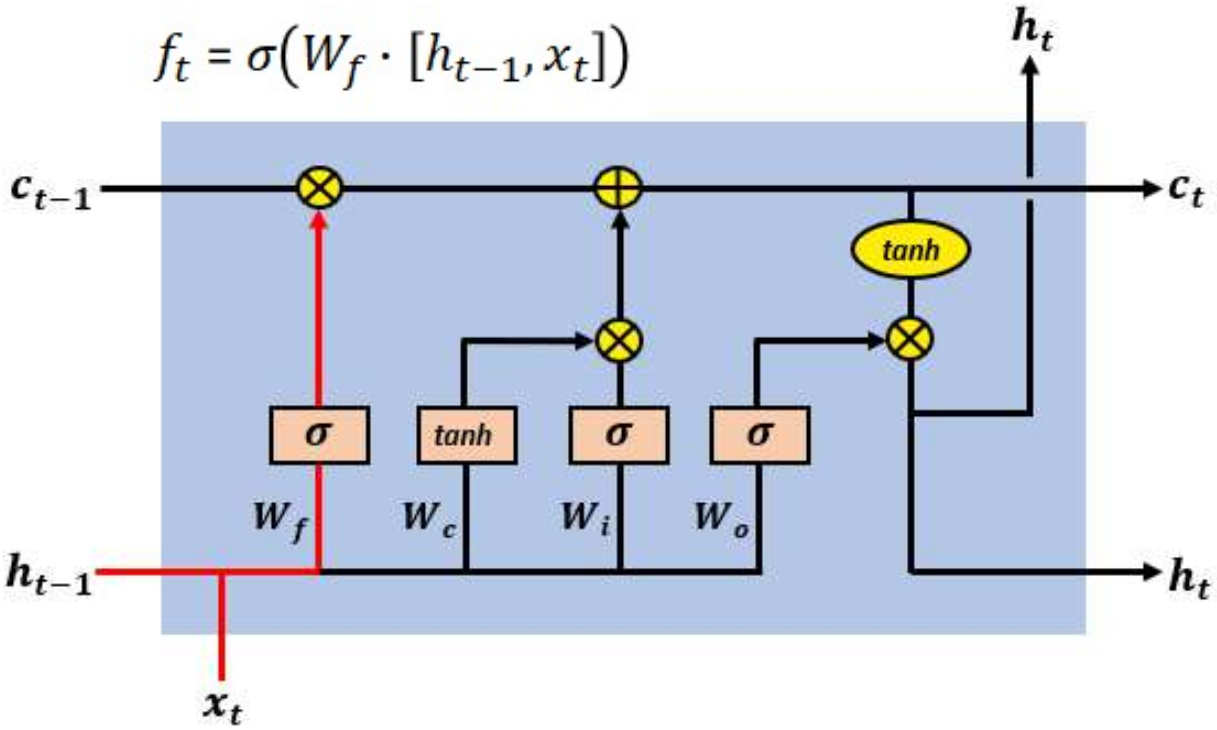
$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$



$$
\begin{aligned}
z_t &= \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \\
r_t &= \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \\
\tilde{h}_t &= \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned}
$$

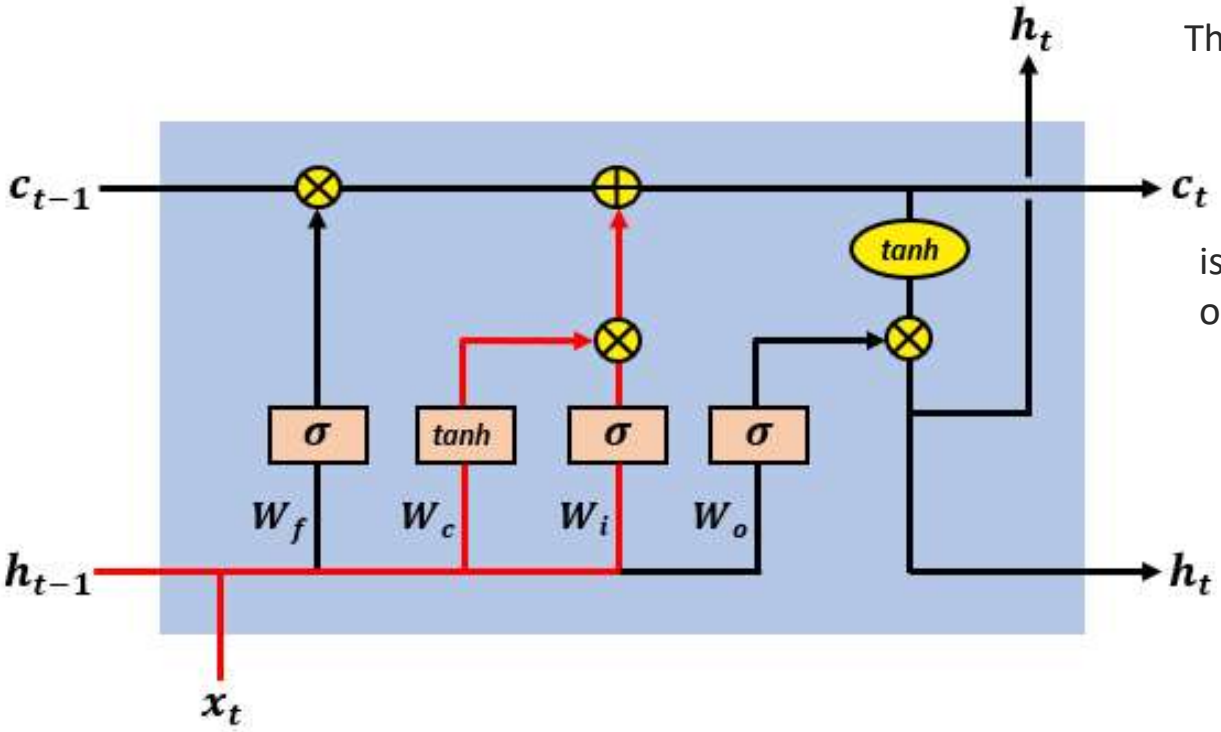
Demystify the LSTM Cell: Forget Gate

The forget gate controls what information in the cell state to forget, given new information than entered the network.



Demystify the LSTM Cell: Input Gate

is equal to the element-wise product of the outputs of the two fully connected layers:



The input gate's output has the form:

$$tanh(W_c \cdot [h_{t-1}, x_t]) \otimes \sigma(W_i \cdot [h_{t-1}, x_t])$$

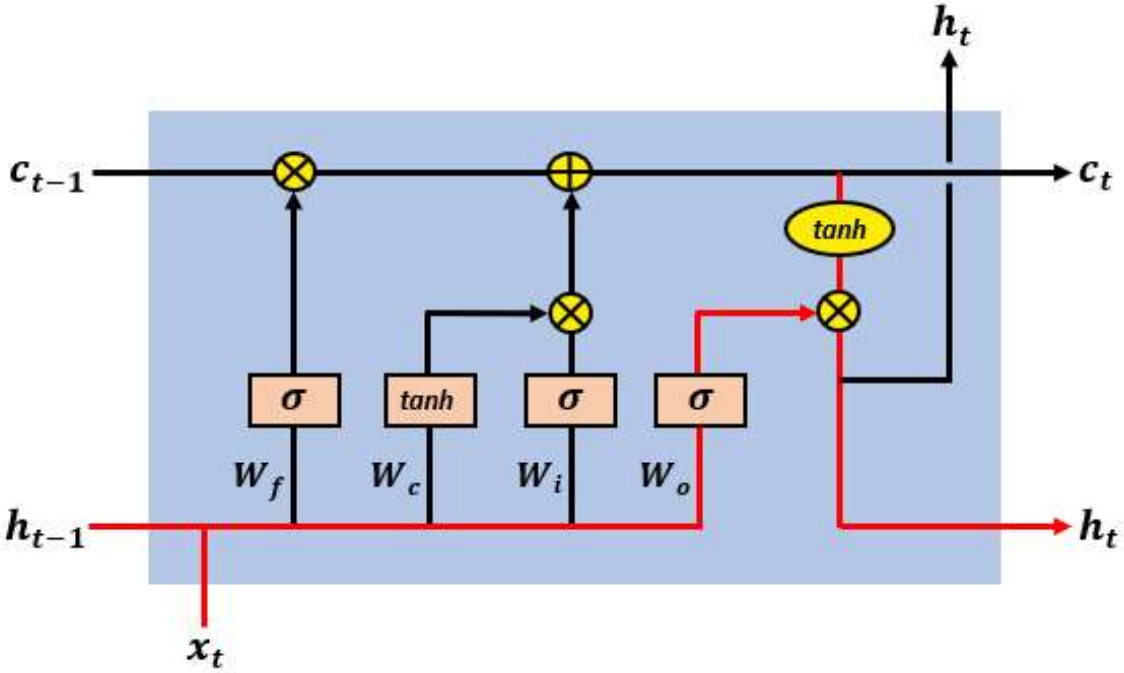
is equal to the element-wise product of the outputs of the two fully connected layers:

$$\tilde{c}_t = tanh\ (W_c \cdot [h_{t-1}, x_t])$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t])$$

Demystify the LSTM Cell: Output Gate

The output gate controls what information encoded in the cell state is sent to the network as input in the following time step, this is done via the output vector h(t).



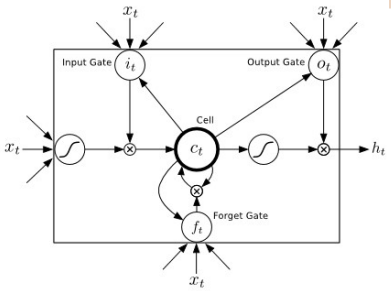
The output gate's activations are given by:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t])$$

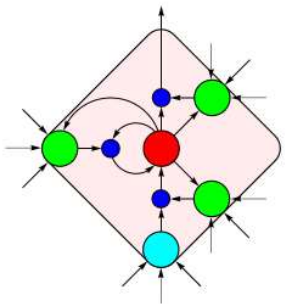
and the cell's output vector is given by:

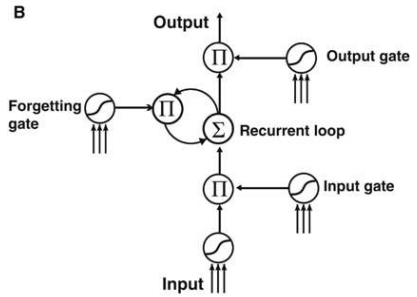
$$h_t = o_t \otimes tanh(c_t)$$

Many Ways to Show an LSTM Cell

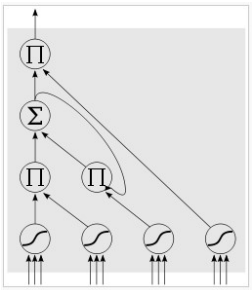


$$
\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \\
c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right) \\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$





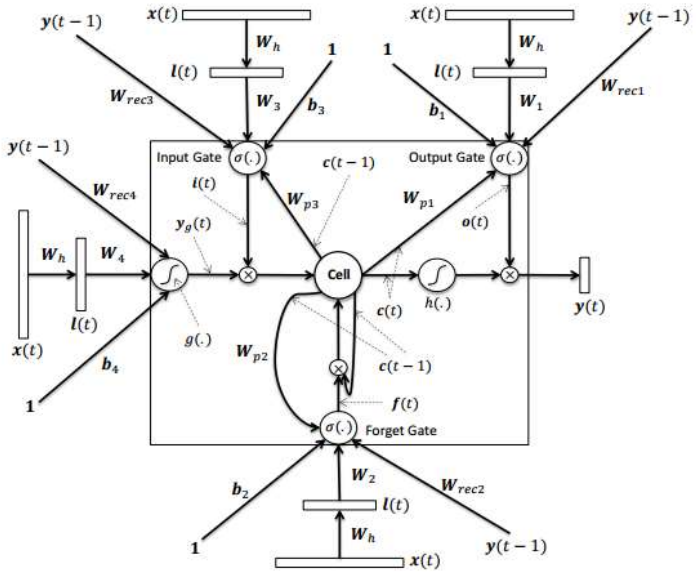
- Long Short-Term Memory (LSTM)[52,53]
- Provides vanishing gradient problem solution
- Input, Output and Forget[54] gates flow information through
- Linear memory cell (called Constant Error Carousel – CEC)


A typical implementation of an LSTM block.

[52] Hochreiter:95fki207r.
[53] Hochreiter:97lstm.
[54] Gers:99a.

(Slide revised from: Koutnik & Schmidhuber, 2015)

Many Ways to Show an LSTM Cell



Deep Sentence Embedding Using Long Short-Term Memory Networks

Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, Rabab Ward

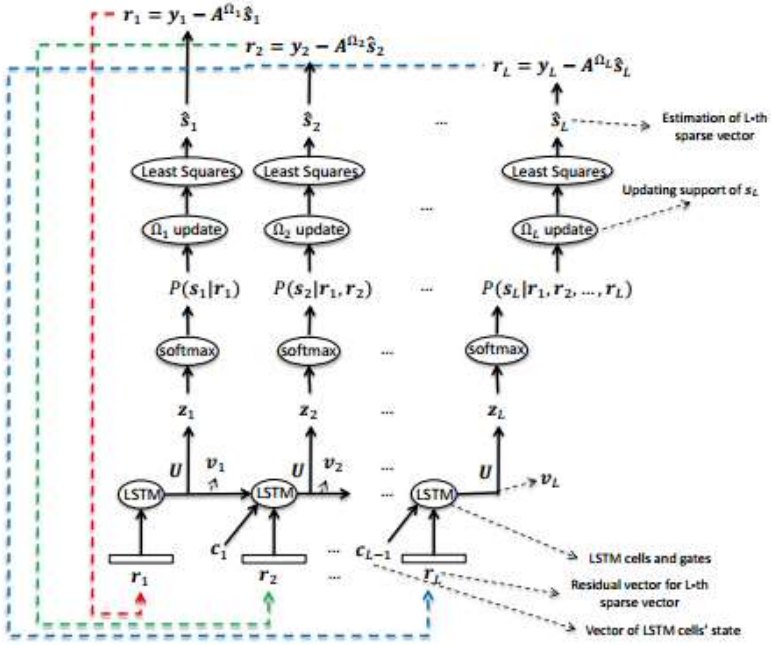
Jul 2015

***Abstract*—This paper develops a model that addresses sentence embedding, a hot topic in current natural language processing research, using recurrent neural networks (RNN) with Long Short-Term Memory (LSTM) cells. The proposed LSTM-RNN model sequentially takes each word in a sentence, extracts its information, and embeds it into a semantic vector. Due to its ability to capture long term memory, the LSTM-RNN accumulates increasingly richer information as it goes through the sentence, and when it reaches the last word, the hidden layer of the network** embedding is learned using sentence pairs. As a result, sentence embedding can better discover salient words and topics in a sentence, and thus is more suitable for tasks that require computing semantic similarities between text strings. By mapping texts into a unified semantic representation, the embedding vector can be further used for different language processing applications, such as machine translation [1], sentiment analysis [2], and information retrieval [3]. In machine translation

$$
\begin{aligned}
\mathbf{y}_g(t) &= g(\mathbf{W}_4\mathbf{l}(t) + \mathbf{W}_{\text{rec4}}\mathbf{y}(t-1) + \mathbf{b}_4) \\
\mathbf{i}(t) &= \sigma(\mathbf{W}_3\mathbf{l}(t) + \mathbf{W}_{\text{rec3}}\mathbf{y}(t-1) + \mathbf{W}_{p3}\mathbf{c}(t-1) + \mathbf{b}_3) \\
\mathbf{f}(t) &= \sigma(\mathbf{W}_2\mathbf{l}(t) + \mathbf{W}_{\text{rec2}}\mathbf{y}(t-1) + \mathbf{W}_{p2}\mathbf{c}(t-1) + \mathbf{b}_2) \\
\mathbf{c}(t) &= \mathbf{f}(t) \circ \mathbf{c}(t-1) + \mathbf{i}(t) \circ \mathbf{y}_g(t) \\
\mathbf{o}(t) &= \sigma(\mathbf{W}_1\mathbf{l}(t) + \mathbf{W}_{\text{rec1}}\mathbf{y}(t-1) + \mathbf{W}_{p1}\mathbf{c}(t) + \mathbf{b}_1) \\
\mathbf{y}(t) &= \mathbf{o}(t) \circ h(\mathbf{c}(t)) \qquad (2)
\end{aligned}
$$

Fig. 2. The basic LSTM architecture used for sentence embedding

Many Ways to Show an LSTM Cell



20 Aug 2015

Distributed Compressive Sensing: A Deep Learning Approach

Hamid Palangi, Rabab Ward, Li Deng

***Abstract*—We address the problem of compressed sensing with Multiple Measurement Vectors (MMVs) when the structure of sparse vectors in different channels depend on each other. *The sparse vectors are not necessarily joint sparse*. We capture this dependency by computing the conditional probability of each entry of each sparse vector to be non-zero given "residuals" of all previous sparse vectors. To compute these probabilities, we propose to use Long Short-Term Memory (LSTM) [1], a bottom up data driven model for sequence modelling. To compute model parameters we minimize a cross entropy cost function. We propose a greedy solver that uses above probabilities at**

where $\mathbf{y} \in \Re^{M\times 1}$ is the known measured vector and $\mathbf{\Phi} \in \Re^{M\times N}$ is a random measurement matrix. An important assumption needed by the decoder to uniquely recover $\mathbf{x}$ given $\mathbf{y}$ and $\mathbf{\Phi}$, is that $\mathbf{x}$ is sparse in a given basis $\mathbf{\Psi}$. This means that

$$\mathbf{x} = \mathbf{\Psi}\mathbf{s} \tag{2}$$

where $\mathbf{s}$ is $K-sparse$, i.e., $\mathbf{s}$ has at most K non-zero elements. The basis $\mathbf{\Psi}$ can be complete; i.e., $\mathbf{\Psi} \in \Re^{N\times N}$, or over-complete; i.e., $\mathbf{\Psi} \in \Re^{N\times N_1}$ where $N < N_1$ over-complete dictionaries is

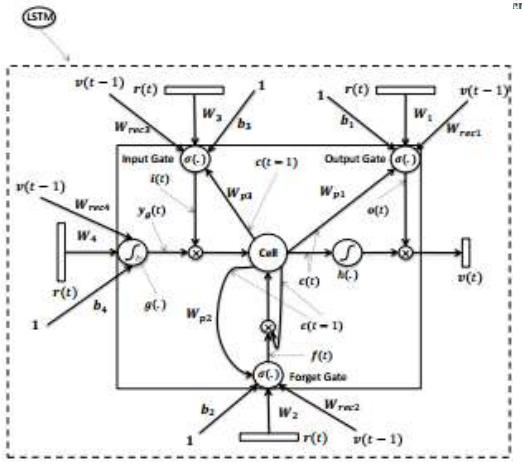
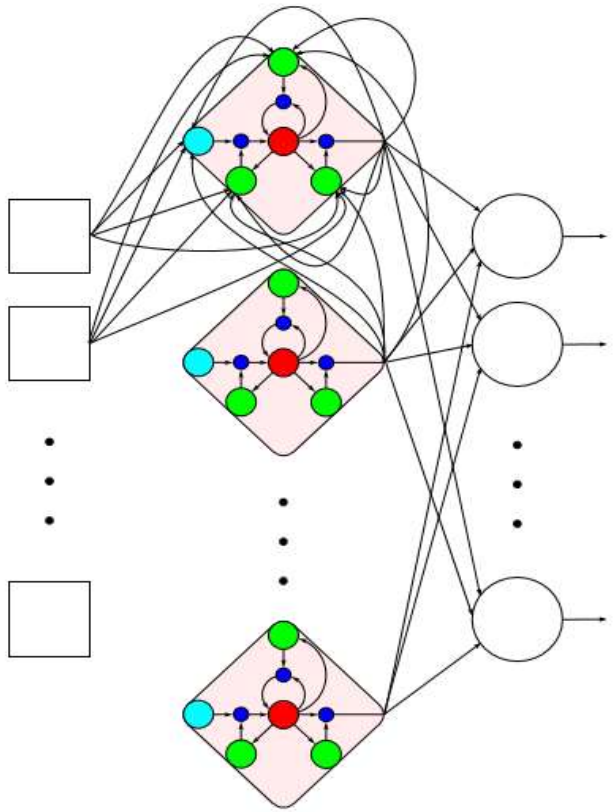
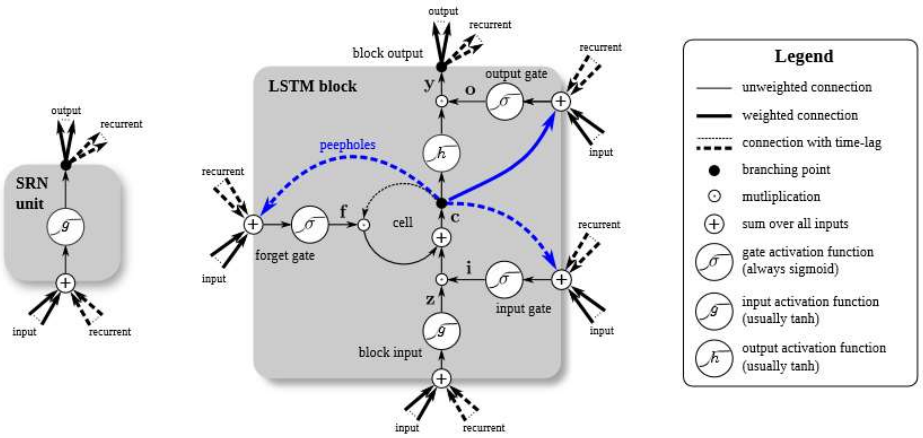


Fig. 2. Block diagram of the Long Short-Term Memory (LSTM).

LSTM Cells in an RNN



LSTM Cell with Peepholes



$$\mathbf{z}^t = g(\mathbf{W}_z\mathbf{x}^t + \mathbf{R}_z\mathbf{y}^{t-1} + \mathbf{b}_z) \qquad \textit{block input}$$

$$\mathbf{i}^t = \sigma(\mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i) \qquad \textit{input gate}$$

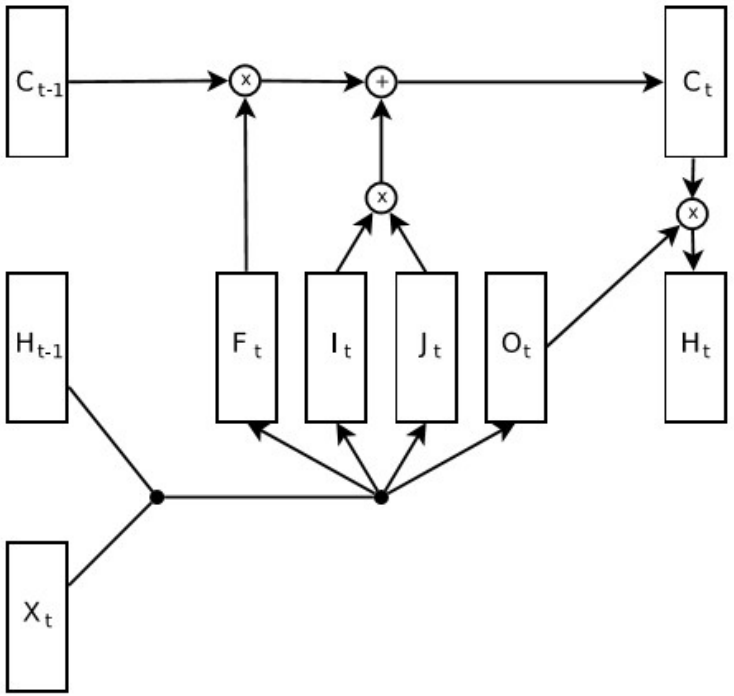
$$\mathbf{f}^t = \sigma(\mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f) \qquad \textit{forget gate}$$

$$\mathbf{c}^t = \mathbf{i}^t \odot \mathbf{z}^t + \mathbf{f}^t \odot \mathbf{c}^{t-1} \qquad \textit{cell state}$$

$$\mathbf{o}^t = \sigma(\mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o) \qquad \textit{output gate}$$

$$\mathbf{y}^t = \mathbf{o}^t \odot h(\mathbf{c}^t) \qquad \textit{block output}$$

LSTM Cell Unfolding over Time



$$
\begin{aligned}
i_t &= \tanh(W_{\mathrm{xi}} x_t + W_{\mathrm{hi}} h_{t-1} + b_{\mathrm{i}}) \\
j_t &= \mathrm{sigm}(W_{\mathrm{xj}} x_t + W_{\mathrm{hj}} h_{t-1} + b_{\mathrm{j}}) \\
f_t &= \mathrm{sigm}(W_{\mathrm{xf}} x_t + W_{\mathrm{hf}} h_{t-1} + b_{\mathrm{f}}) \\
o_t &= \tanh(W_{\mathrm{xo}} x_t + W_{\mathrm{ho}} h_{t-1} + b_{\mathrm{o}}) \\
c_t &= c_{t-1} \odot f_t + i_t \odot j_t \\
h_t &= \tanh(c_t) \odot o_t
\end{aligned}
$$

Figure 1. The LSTM architecture. The value of the cell is increased by $i_t \odot j_t$, where $\odot$ is element-wise product. The LSTM's output is typically taken to be h_t, and c_t is not exposed. The forget gate f_t allows the LSTM to easily reset the value of the cell.

(Jozefowics, Zarembe, Sutskever, ICML 2015)

Backpropagation Through Time in LSTMs

The long term dependencies and relations are *encoded* in the cell state vectors and it's the cell state derivative that can prevent the LSTM gradients from vanishing. The LSTM cell state has the form:

$$c_t = c_{t-1} \otimes \sigma(W_f \cdot [h_{t-1}, x_t]) \oplus$$

$$tanh\,(W_c \cdot [h_{t-1}, x_t]) \otimes \sigma(W_i \cdot [h_{t-1}, x_t])$$

$$\frac{\partial c_t}{\partial c_{t-1}} = \sigma'(W_f \cdot [h_{t-1}, x_t]) \cdot W_f \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot c_{t-1}$$

$$+ f_t$$

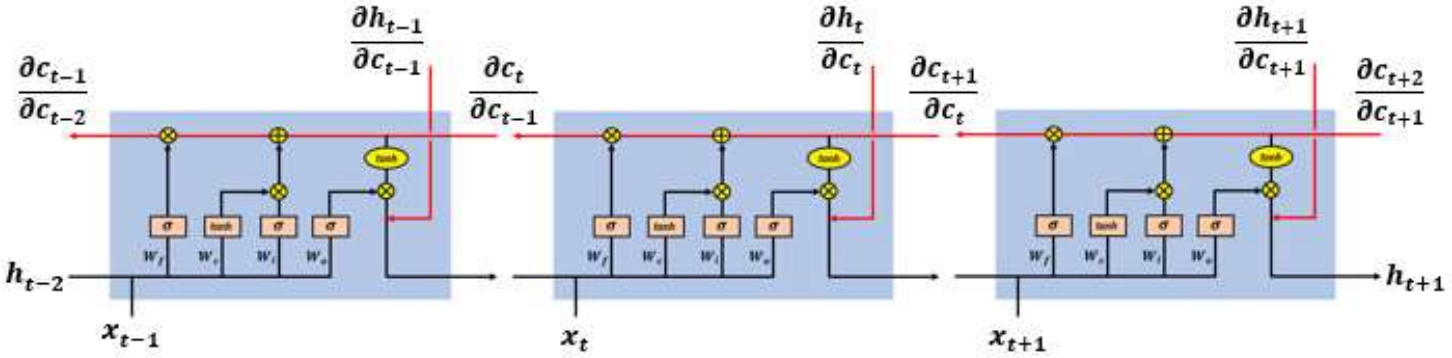
$$+ \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot \tilde{c}_t$$

$$+ \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot i_t$$

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k} \cdots \frac{\partial c_2}{\partial c_1}\frac{\partial c_1}{\partial W}$$

$$= \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}\left(\prod_{t=2}^{k} \frac{\partial c_t}{\partial c_{t-1}}\right)\frac{\partial c_1}{\partial W} \quad (4)$$

$$c_t = c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t$$



➔ **The gradient contains the forget gate's vector of activations, which allows the network to better control the gradients values, at each time step, using suitable parameter updates of the forget gate**

➔ **Addictive: less likely all 0s**

How to Train

Exact Forward, approximate backward

Clipped Gradients

Sequences of fixed length

Padded sequences - normalize loss value to actual sequence length

Initialize with zero vector

Use output of network as part of next input (student forcing)

Use ground truth as next input (teacher forcing)

Predict deltas

ELECTRICAL & COMPUTER ENGINEERING

GRU: Gated Recurrent Unit

x_t = input

h_t = hidden state (memory)

z_t = "update gate"

r_t = "reset gate"

σ = sigmoid function (gate)

ϕ = tanh function

$\otimes$ = Matrix Multiplication

* = Elementwise Multiplication

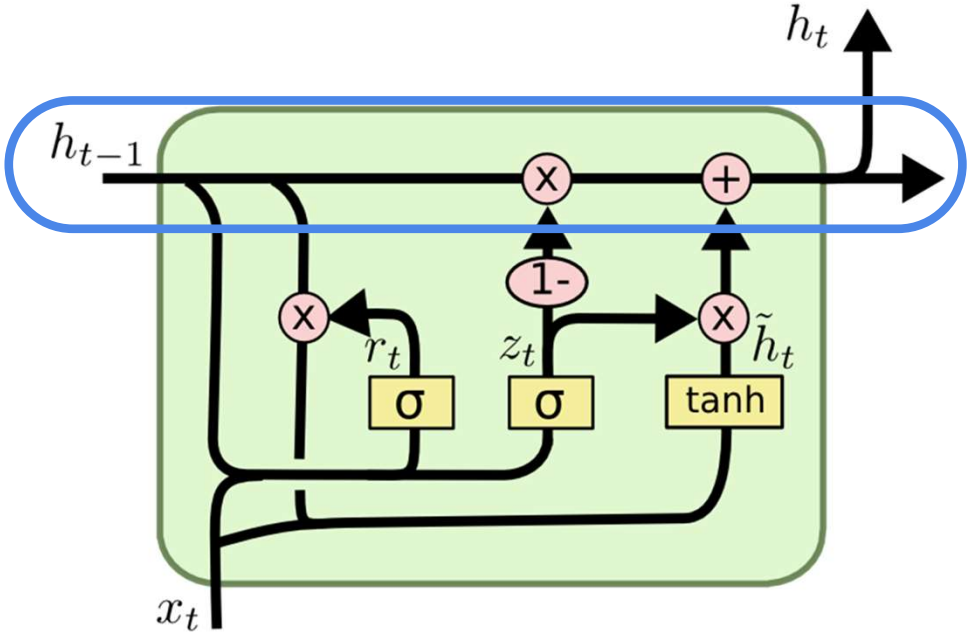
$z_t = \sigma(W_{zh} \otimes h_{t-1} + W_{zx} \otimes x_t)$

$r_t = \sigma(W_{rh} \otimes h_{t-1} + W_{rx} \otimes x_t)$

$g_t = \mathbf{r_t} * h_{t-1}$

$h'_t = \phi(W_{hg} \otimes g_t + W_{hx} \otimes x_t)$

$h_t = (1 - z_t) * h_{t-1} + z_t * h'_t$



Gated Recurrent Unit (GRU)

(simpler than LSTM; no output gates)

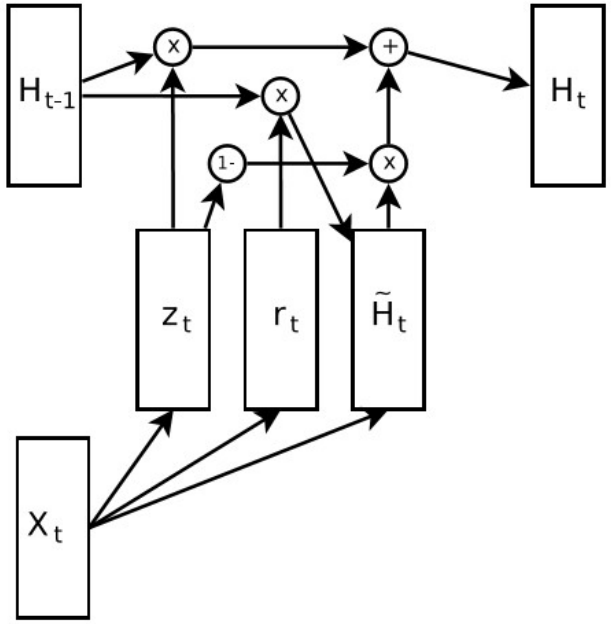


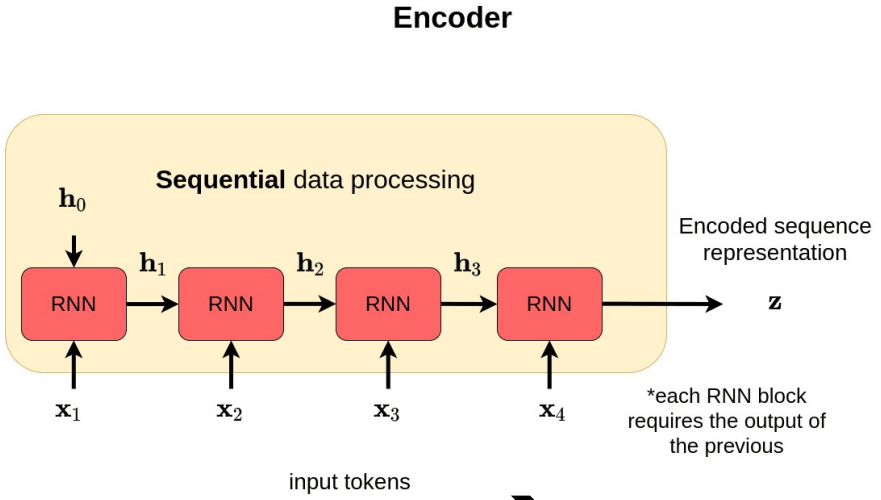
Figure 2. The Gated Recurrent Unit. Like the LSTM, it is hard to tell, at a glance, which part of the GRU is essential for its functioning.

$$
\begin{aligned}
r_t &= \text{sigm}\left(W_{\text{xr}}x_t + W_{\text{hr}}h_{t-1} + b_{\text{r}}\right) \\
z_t &= \text{sigm}(W_{\text{xz}}x_t + W_{\text{hz}}h_{t-1} + b_{\text{z}}) \\
\tilde{h}_t &= \tanh(W_{\text{xh}}x_t + W_{\text{hh}}(r_t \odot h_{t-1}) + b_{\text{h}}) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
\end{aligned}
$$

(Jozefowics, Zarembe, Sutskever, ICML 2015; Google
Kumar et al., arXiv, July, 2015; Metamind)

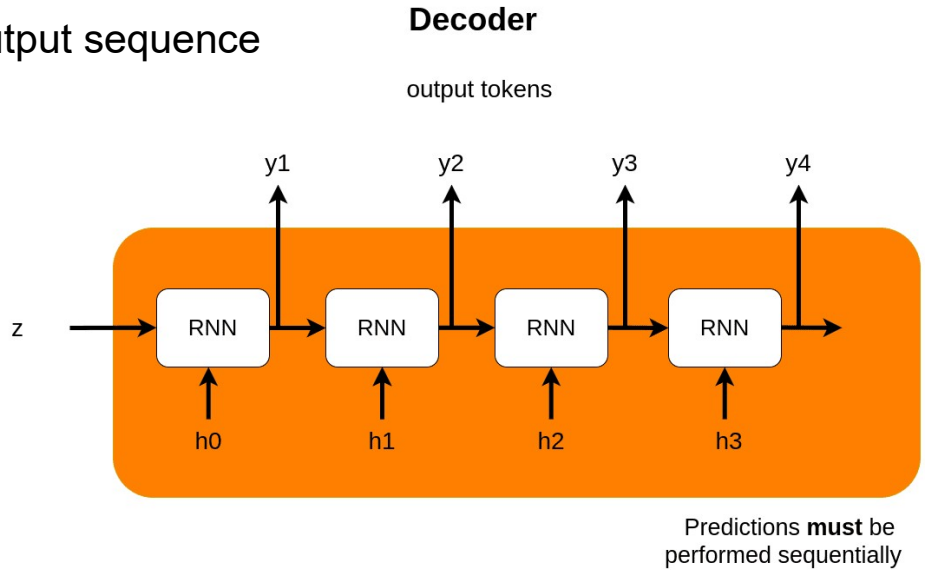
Encoder and Decoder for Sequence-to-Sequence Modeling

- RNN: LSTM/GRU



From an Input sequence

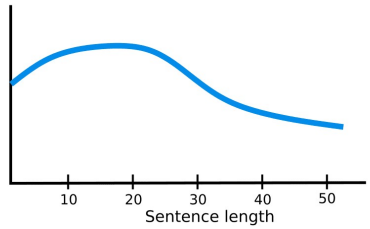
To predict an output sequence



➔ Work well for short sequences (<20 time steps): reference window/reception field

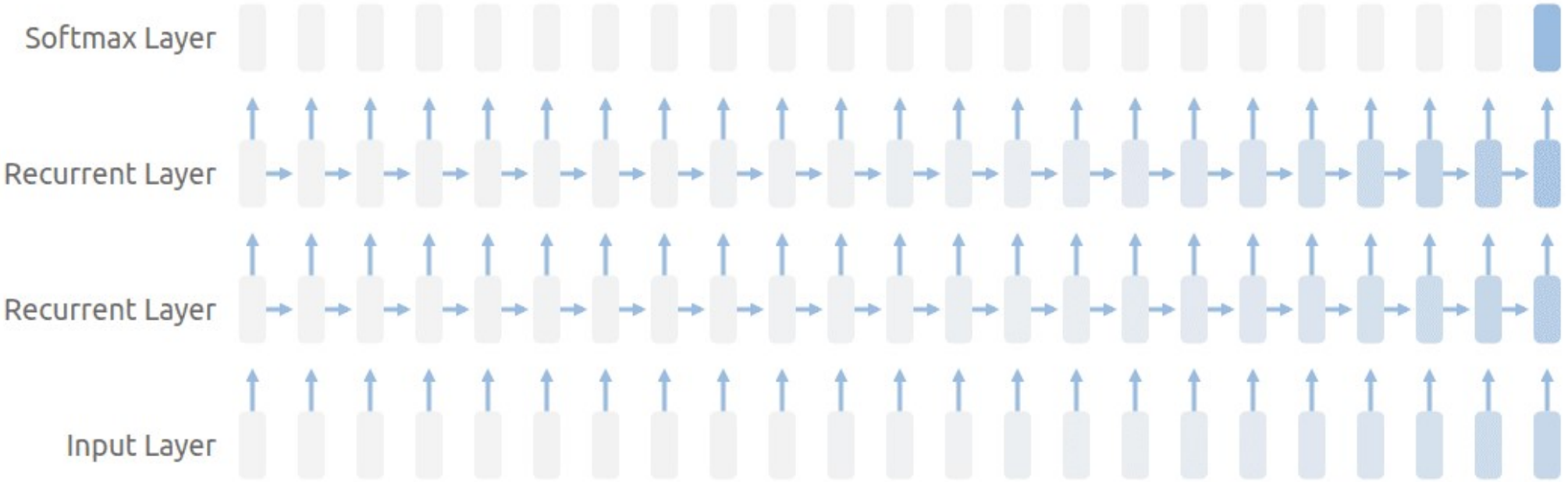
➔ Sequence can be long, and z can remember well the last token;

➔ Gradient vanishing



ELECTRICAL & COMPUTER ENGINEERING

Vanishing Gradient in the Stacked RNN



Vanishing Gradient: where the contribution from the earlier steps becomes insignificant in the gradient for the vanilla RNN unit.

Need “Attention”

- The core idea is that the context vector *z* should have access to all parts of the input sequence instead of just the last one: Need to form a **direct connection** with each timestamp.
- Larochelle and Hinton: Glimpse at Different Parts of an Image
- Look at all the different words at the same time and learn to “pay attention“ to the correct ones depending on the task at hand.
- This is what we now call **attention**, which is simply a notion of **memory**, gained from attending at ***multiple inputs through time***.

Memory is **attention through time**. ~ Alex Graves 2020

Implicit Attention

- One way to visualize implicit attention is by looking at the partial derivatives with respect to the input. In math, this is the Jacobian matrix
- By asking the network to **'weigh' its sensitivity to the input based on memory from previous inputs,** we introduce **explicit attention**
- Differential (soft) vs Hard Attention

Attention in the Encoder-Decoder Model for Seq2Seq

In the encoder-decoder RNN case, given previous state in the decoder as $\mathbf{y}_{i-1}$ and the the hidden state $\mathbf{h} = h_1, h_2, h_n$, we have something like this:

$$\mathbf{e}_i = \text{attention}_{\text{net}}\left(y_{i-1}, \mathbf{h}\right) \in R^n$$

The index i indicates the prediction step. Essentially, we define a score between the hidden state of the decoder and all the hidden states of the encoder.

More specifically, for each hidden state (denoted by j) $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_n$ we will calculate a scalar:

$$e_{ij} = \text{attention}_{\text{net}}\left(\mathbf{y}_{i-1}, h_j\right)$$

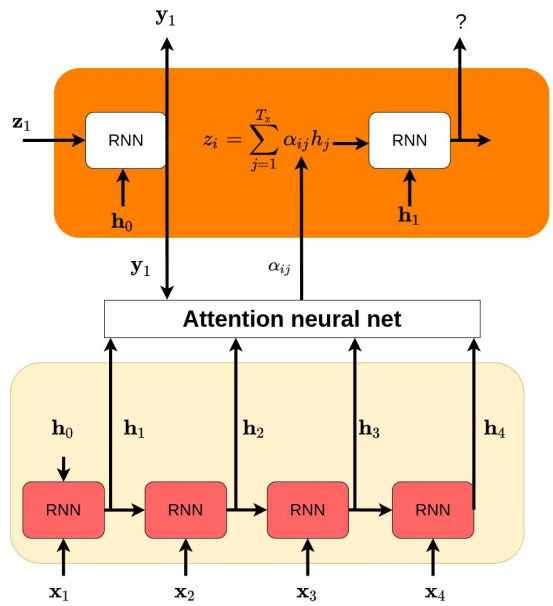
Because, we want some extra properties: a) to make it a probability distribution and b) to make the scores to be far from each other. The latter results in having more confident predictions and is nothing more than our well known softmax.

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

Finally, here is where the new magic will happen:

$$z_i = \sum_{j=1}^{T} \alpha_{ij} \mathbf{h}_j$$

In theory, attention is defined as the weighted average of values. But this time, **the weighting is a learned function**! Intuitively, we can think of α_{ij} as **data-dependent dynamic weights**. Therefore, it is obvious that we need a notion of **memory**, and as we said **attention weight store the memory** that is gained through time



ELECTRICAL & COMPUTER ENGINEERING

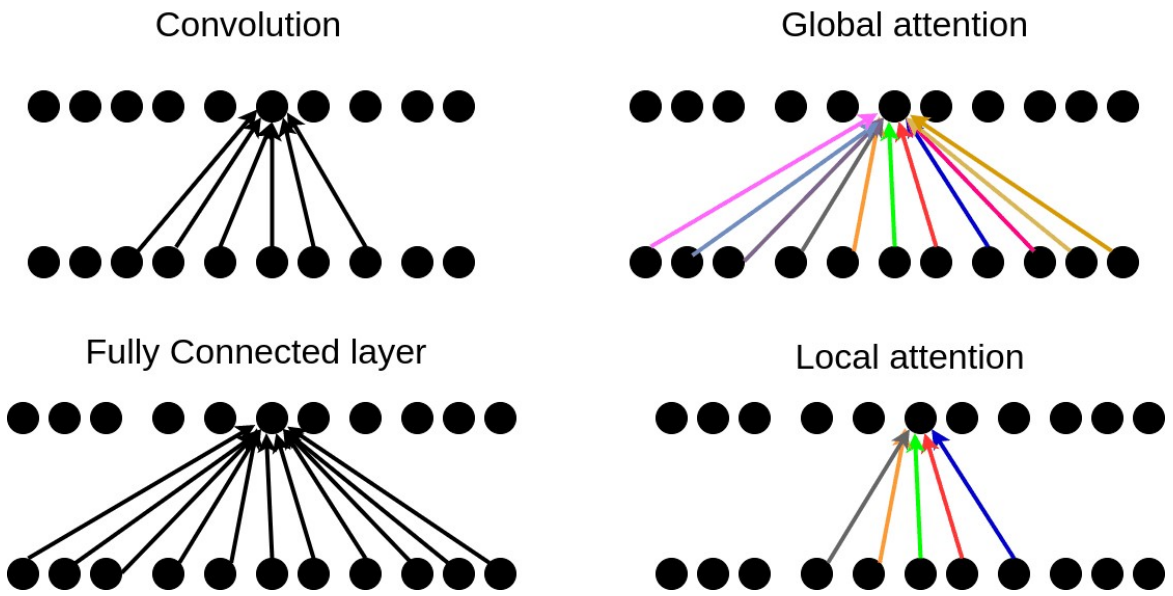
How to Compute Attention

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

A Widely Used Attention Network

- Bahdanau et al. [2]: **They parametrize attention as a small fully connected neural network**.
- “Intuitively, this implements a mechanism of attention in the decoder. The decoder decides parts of the source sentence to pay attention to. By letting the decoder have an attention mechanism, we **relieve the encoder from the burden of having to encode all information in the source sentence into a fixed-length vector**. With this new approach, the information can be spread throughout the sequence of annotations, which can be selectively retrieved by the decoder accordingly.” ~ Neural machine translation by jointly learning to align and translate
- We have another neural network to train and we need to have O(T^2)*O*(*T*2) weights (where T*T* is the length of both the input and output sentence).

Global/Local Attention vs Convolution/FC



- The colors in the attention indicate that these weights are constantly changing while in convolution and fully connected layers they are slowly changing by gradient descent.

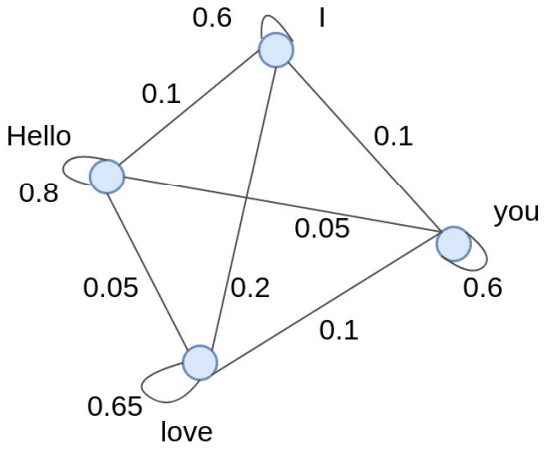
Self Attention

Self-attention
Probability score matrix

	Hello	I	love	you
Hello	0.8	0.1	0.05	0.05
I	0.1	0.6	0.2	0.1
love	0.05	0.2	0.65	0.1
you	0.2	0.1	0.1	0.6

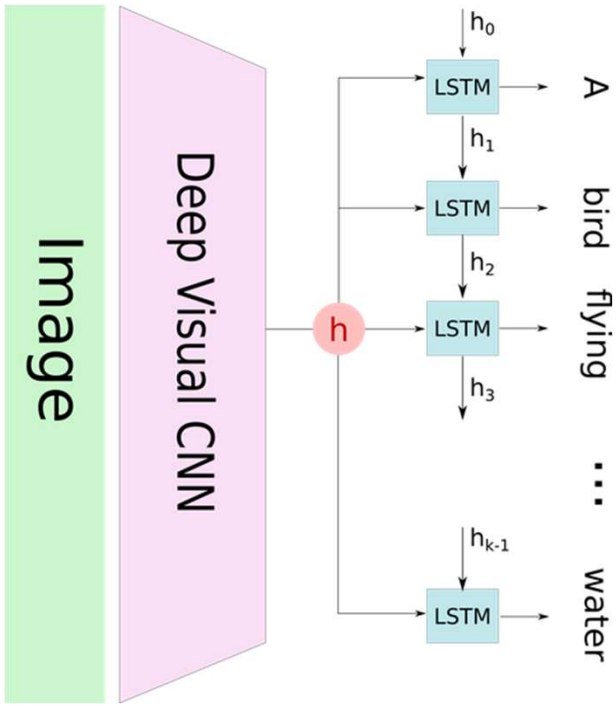
Softmax(Attention)
equation

Self-attention as a
undirected weighted graph



The self-attention can be computed in any mentioned trainable way.
The end goal is to create a meaningful representation of the sequence before transforming to another

What to train? Image Captioning

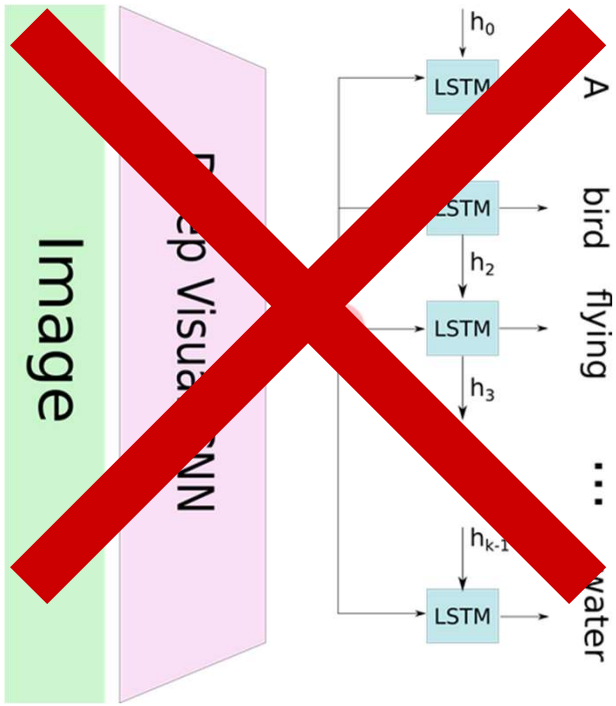


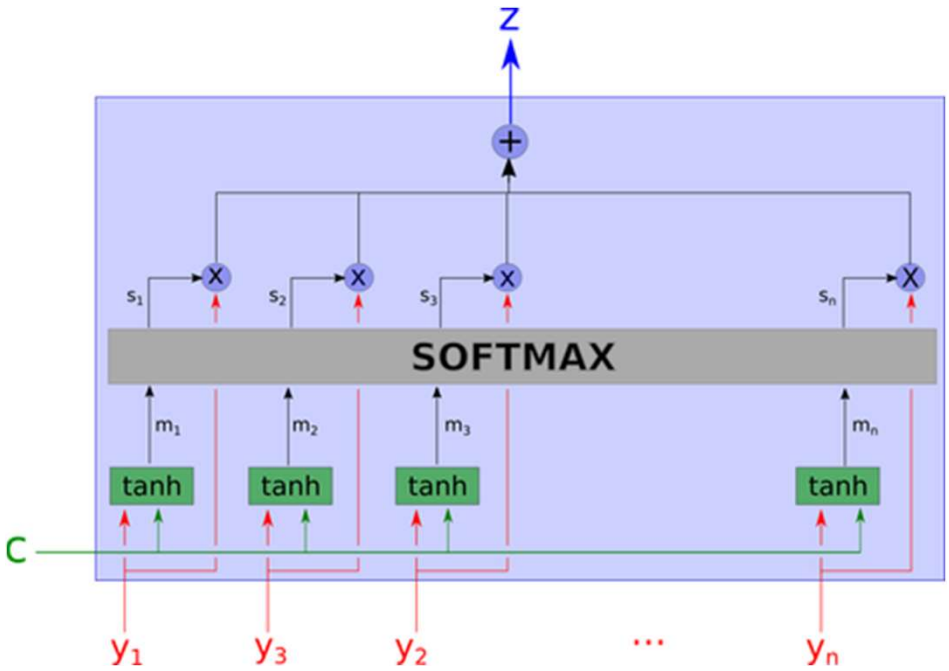
https://blog.heuritech.com/2016/01/20/attention-mechanism/

ELECTRICAL & COMPUTER ENGINEERING

Attention Mechanism

Gate a static input based on a dynamic input

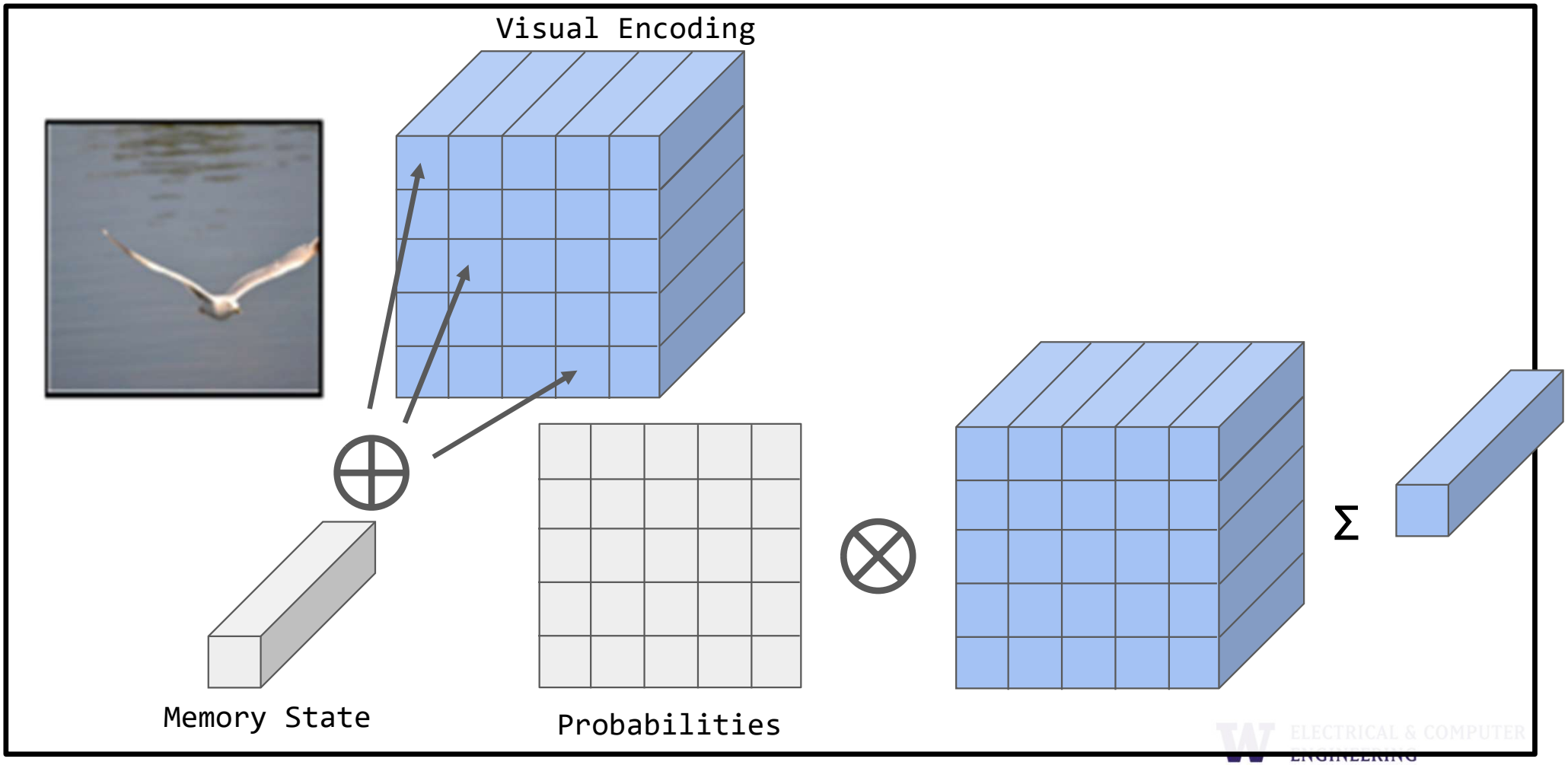




https://blog.heuritech.com/2016/01/20/attention-mechanism/

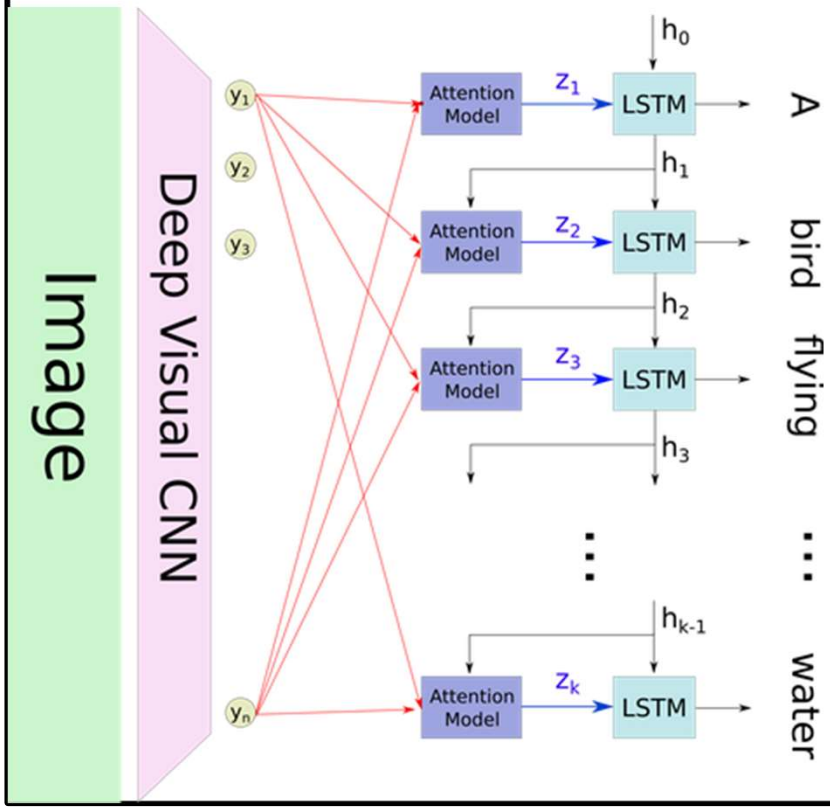
ELECTRICAL & COMPUTER ENGINEERING

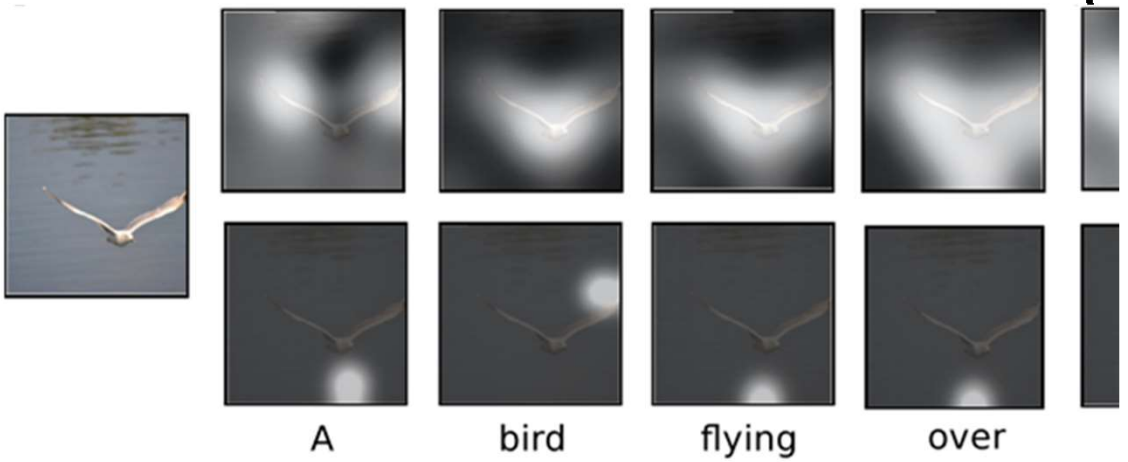
Visual Attention



What to train? Image Captioning, VQA

Show, Attend and Tell:





A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

Solution: VQA

Who is wearing glasses?

man

woman

Where is the child sitting?

fridge

arms

Is the umbrella upside down?

yes

no

How many children are in the bed?

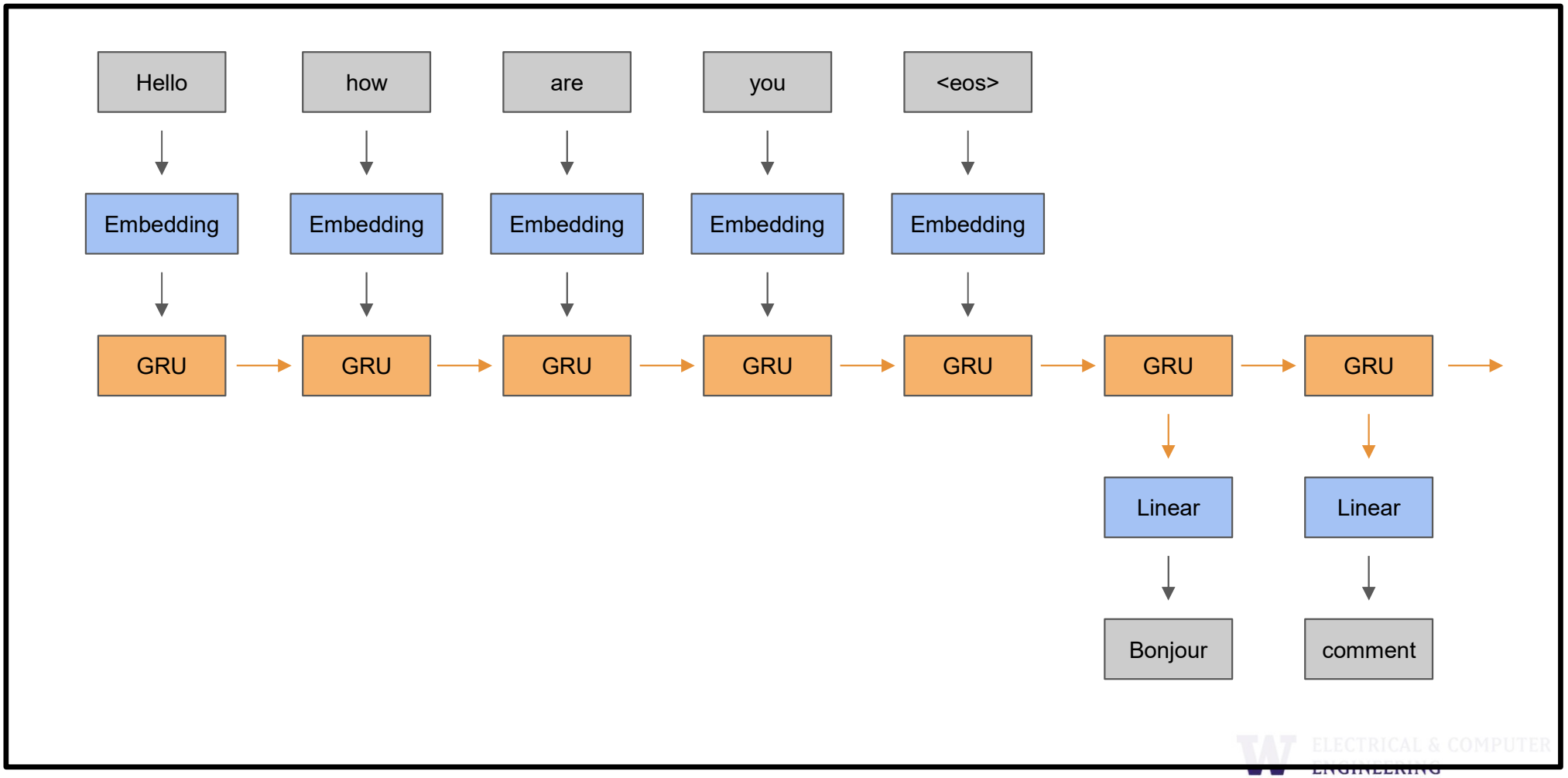
2

1

http://www.visualqa.org/challenge.html

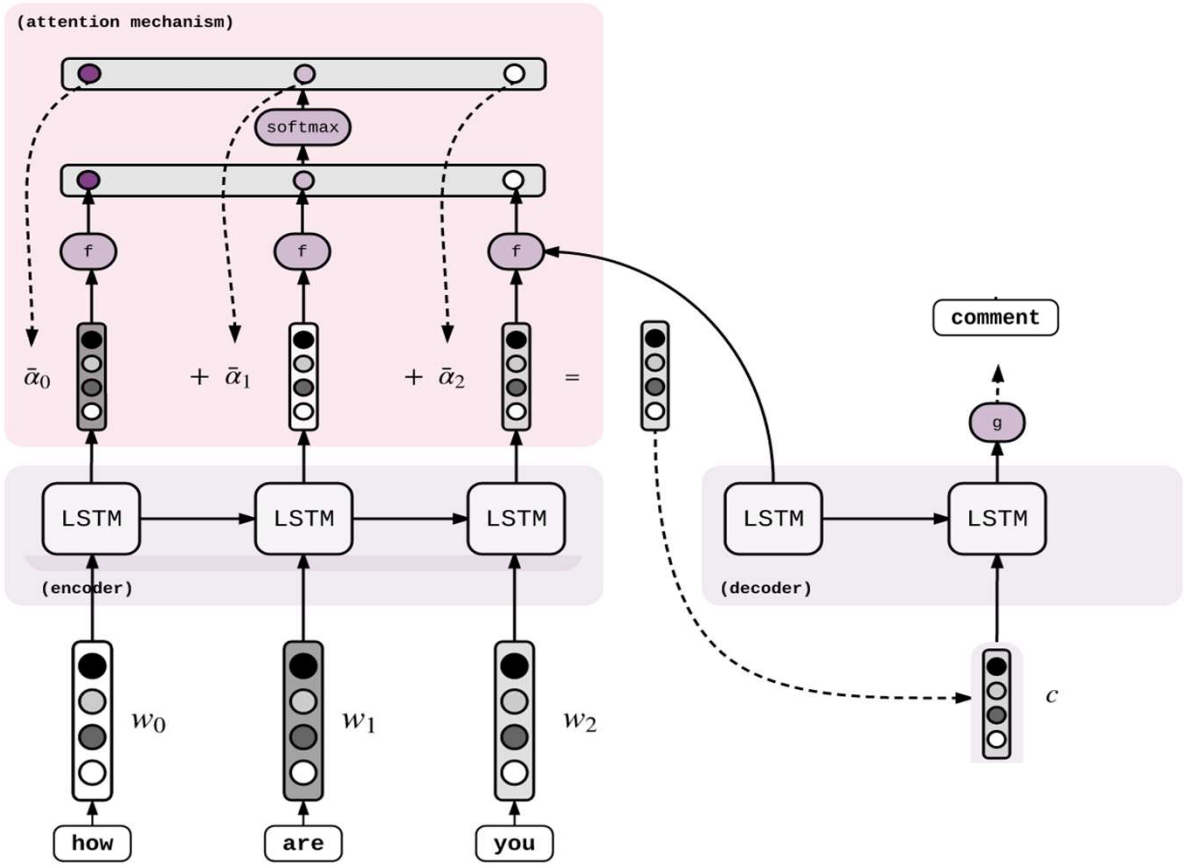
ELECTRICAL & COMPUTER ENGINEERING

Sequence To Sequence



ELECTRICAL & COMPUTER ENGINEERING

What to Train? Machine Translation with Attention



https://guillaumegenthial.github.io/sequence-to-sequence.html

ELECTRICAL & COMPUTER ENGINEERING

RNN Difficulties

Slow to converge

Don’t parallelize as well as CNNs

Vanishing Gradient

Long-term dependencies

Need more data

Hard to understand what they learn

ELECTRICAL & COMPUTER ENGINEERING

When do people use RNNs

Variable length sequences where order matters

When medium/long term dependencies are important

ELECTRICAL & COMPUTER ENGINEERING

Outline

1. DTW: Similarity Measure of Time Series Data
2. RNN
3. LSTM and GRU
4. Encoder-Decoder
5. Attention
6. Transfomer
7. BERT
8. GPT

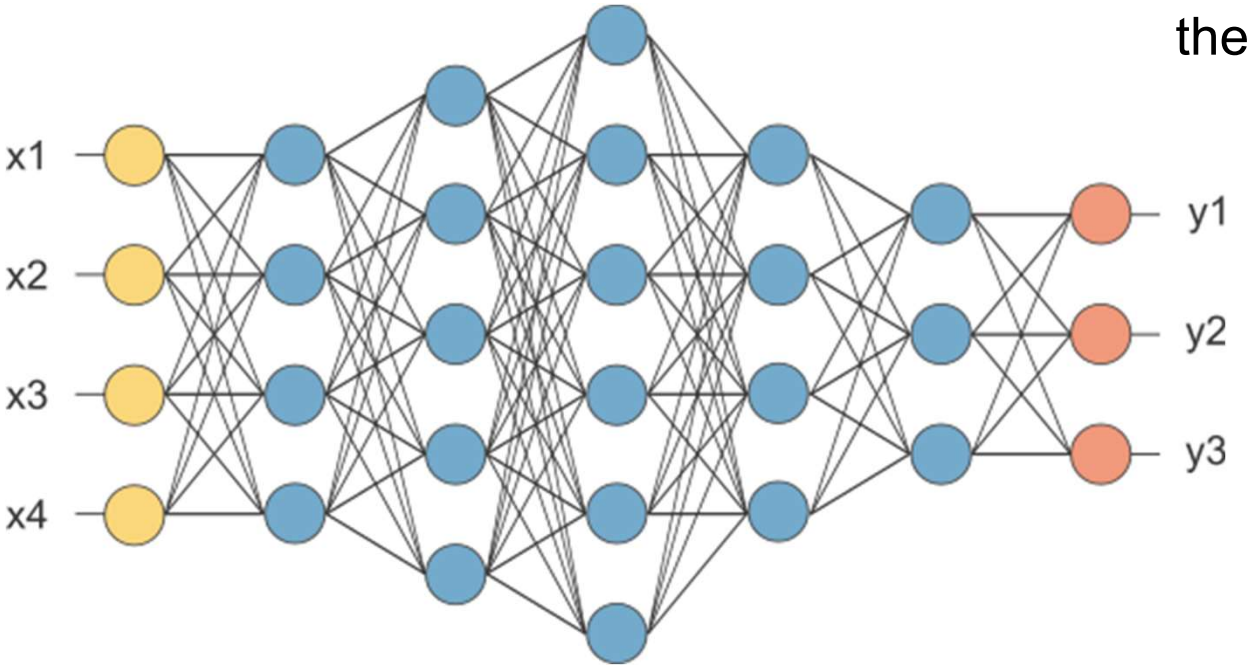
Attention and Transformers

Plan: We trace back history to see how attention and transformers have emerged

1. Basic models, related to transduction models and attention.
2. Encoder-Decoder model, using recurrent networks such as LSTM.
3. Transformer models are general models sufficient for almost all biotech applications (graph models may be treated to be special cases too).
4. For example, DeepMind AlphaFold2 uses depends on a transformer architecture to train an end-to-end model.
5. The transformer model also makes it easy for large scale biological data (pre)training.

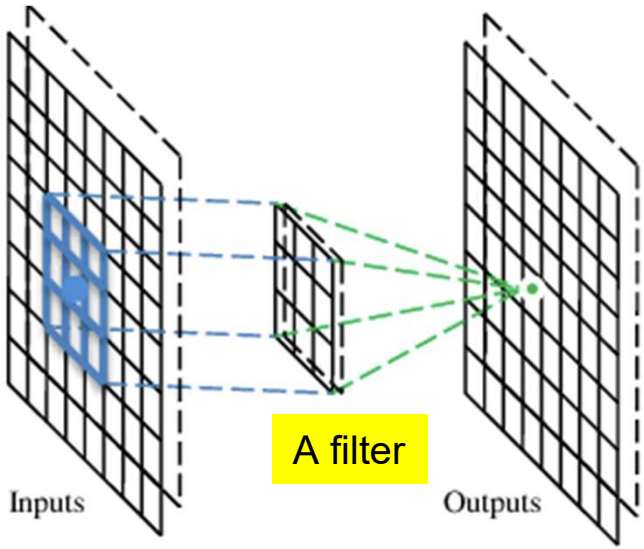
1. Fully connected network, feedforward network

To learn the weights on the edges



2. CNN

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that do convolutional operation.



Convolutional layer

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input

These are the network parameters to be learned.

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

Convolution Operation

1	-1	-1
-1	1	-1
-1	-1	1

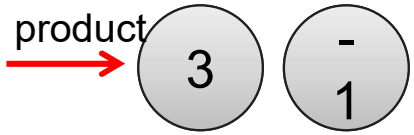
Filter 1

stride=1

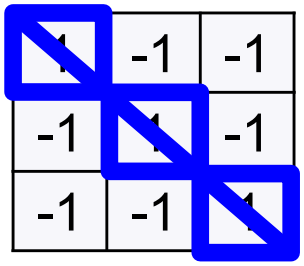
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input

Dot product

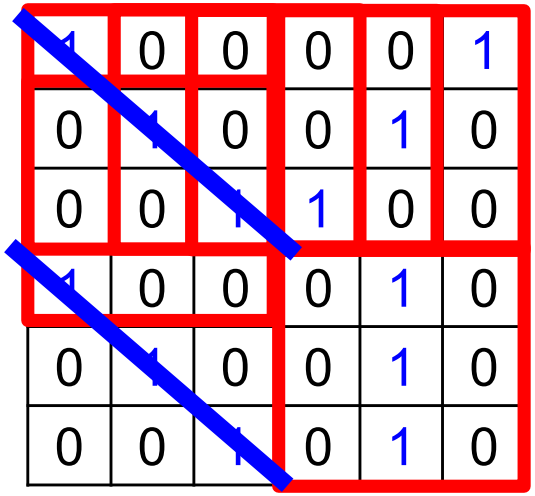


Convolution

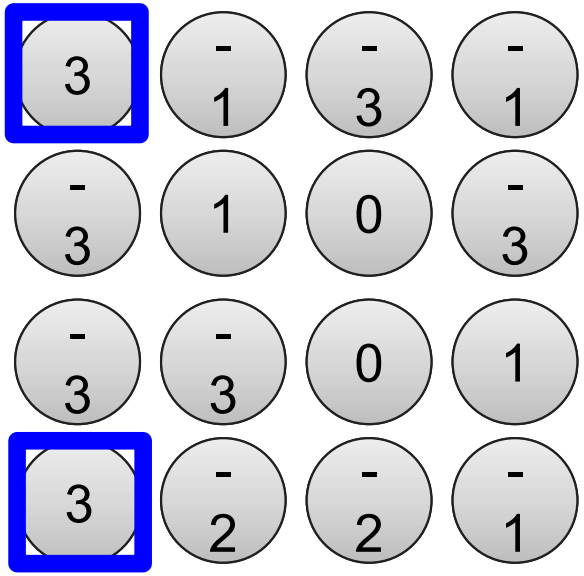


Filter 1

stride=1

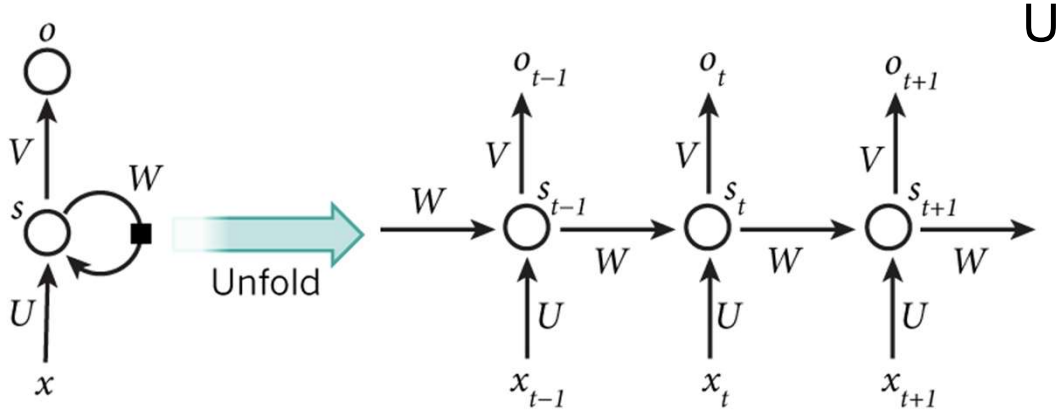


Input

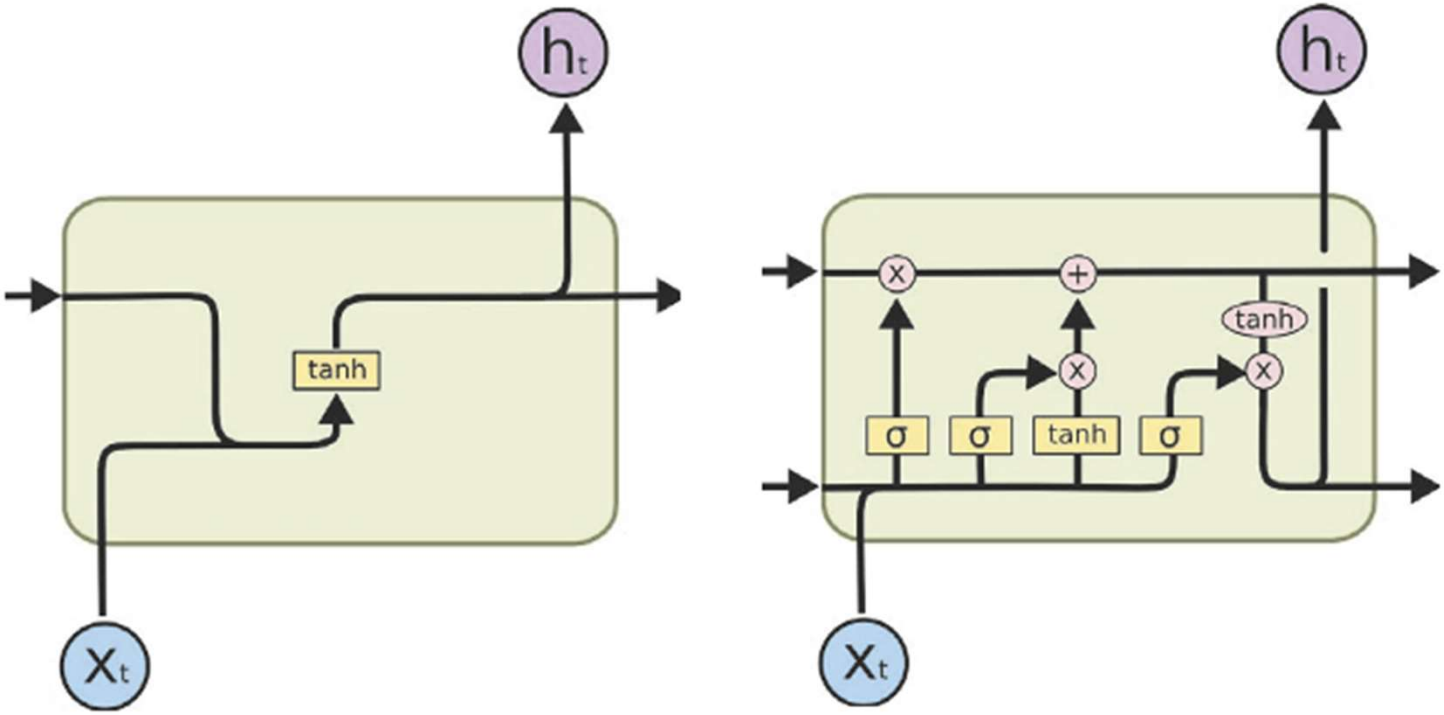


3. RNN

Parameters to be learned:
U, V, W

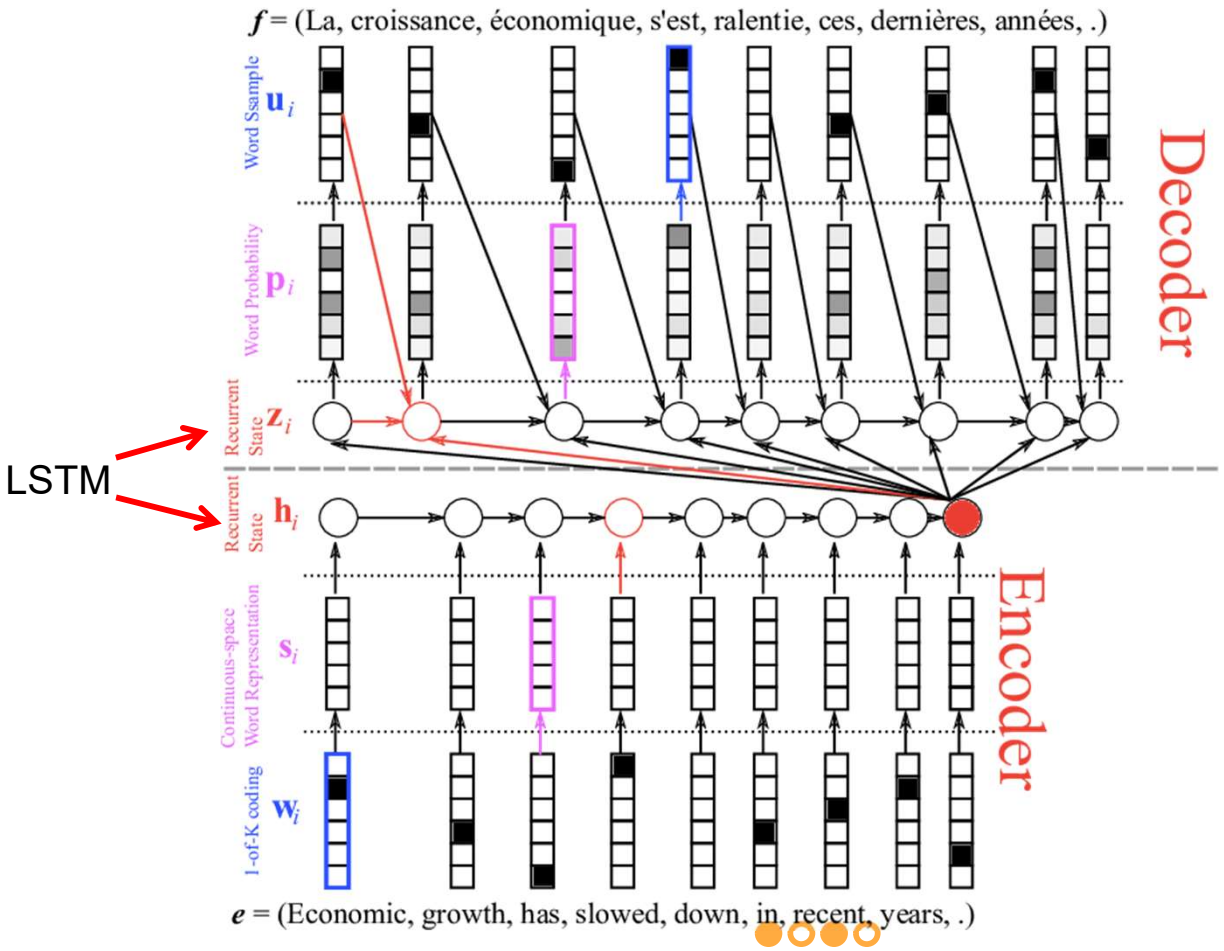


Simple RNN vs LSTM

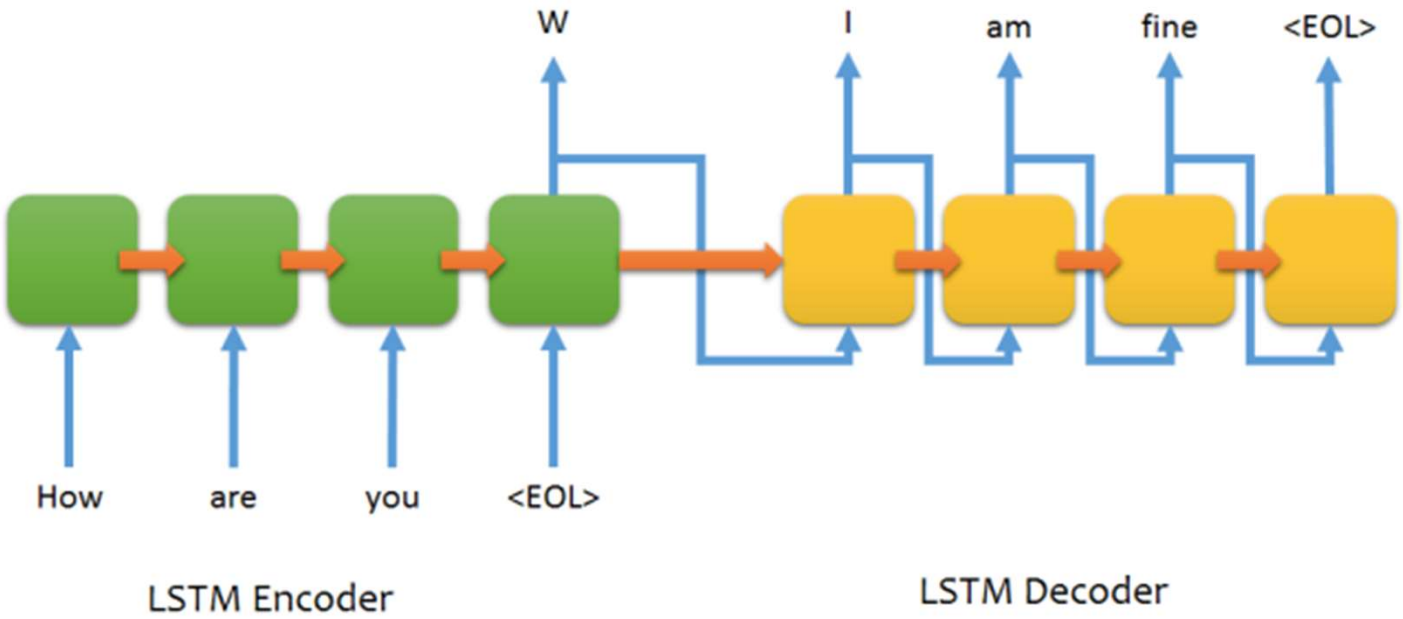


(a) RNN (b) LSTM

Encoder-Decoder machine translation



Encoder-Decoder LSTM structure for chatting



Attention

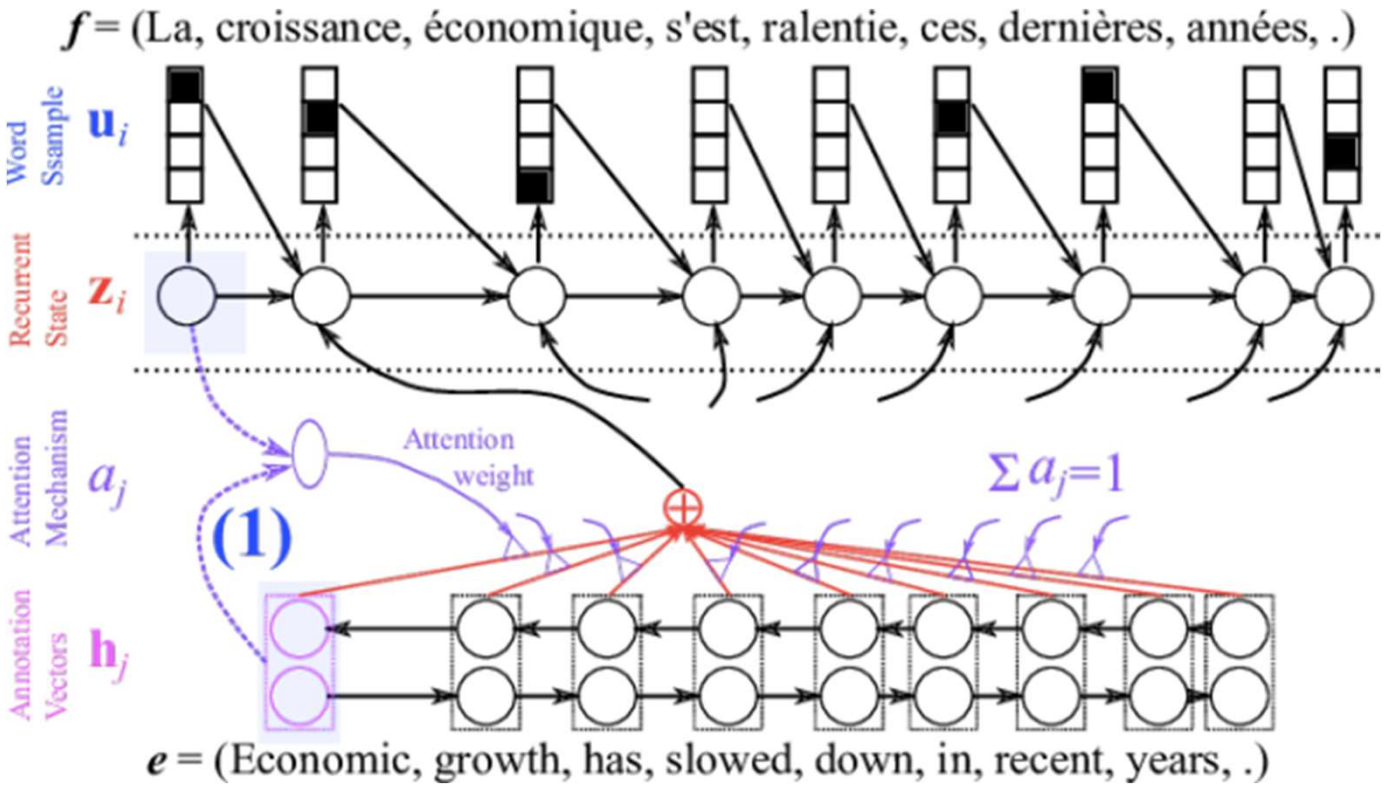
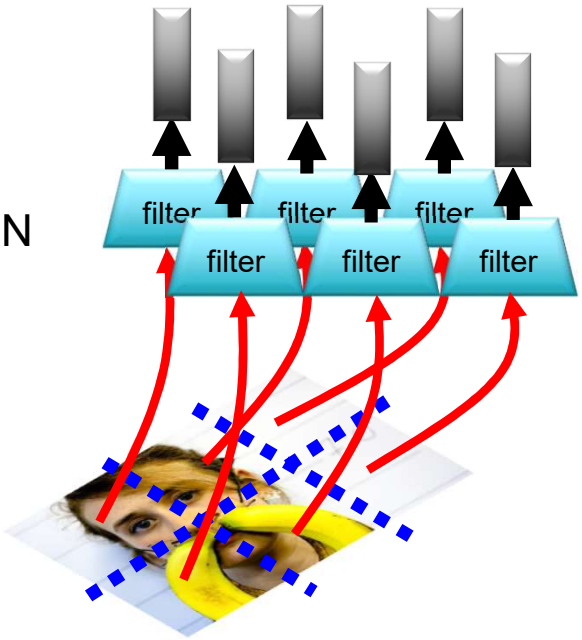

f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample
u_i
Recurrent State
z_i
Attention Mechanism
a_j
Attention weight
$\Sigma a_j=1$
(1)
Annotation Vectors
h_j
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Image caption generation using attention

A vector for each region

CNN



z^0 is initial parameter, it is also learned

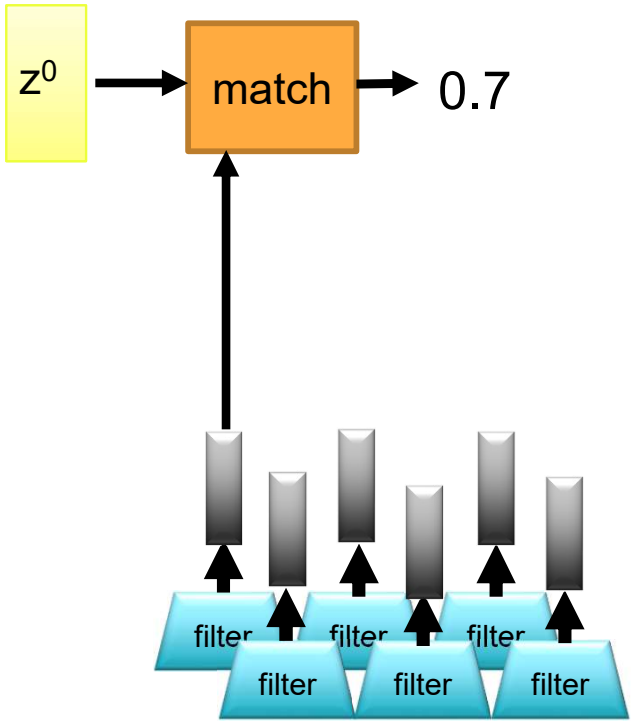


Image caption generation using attention

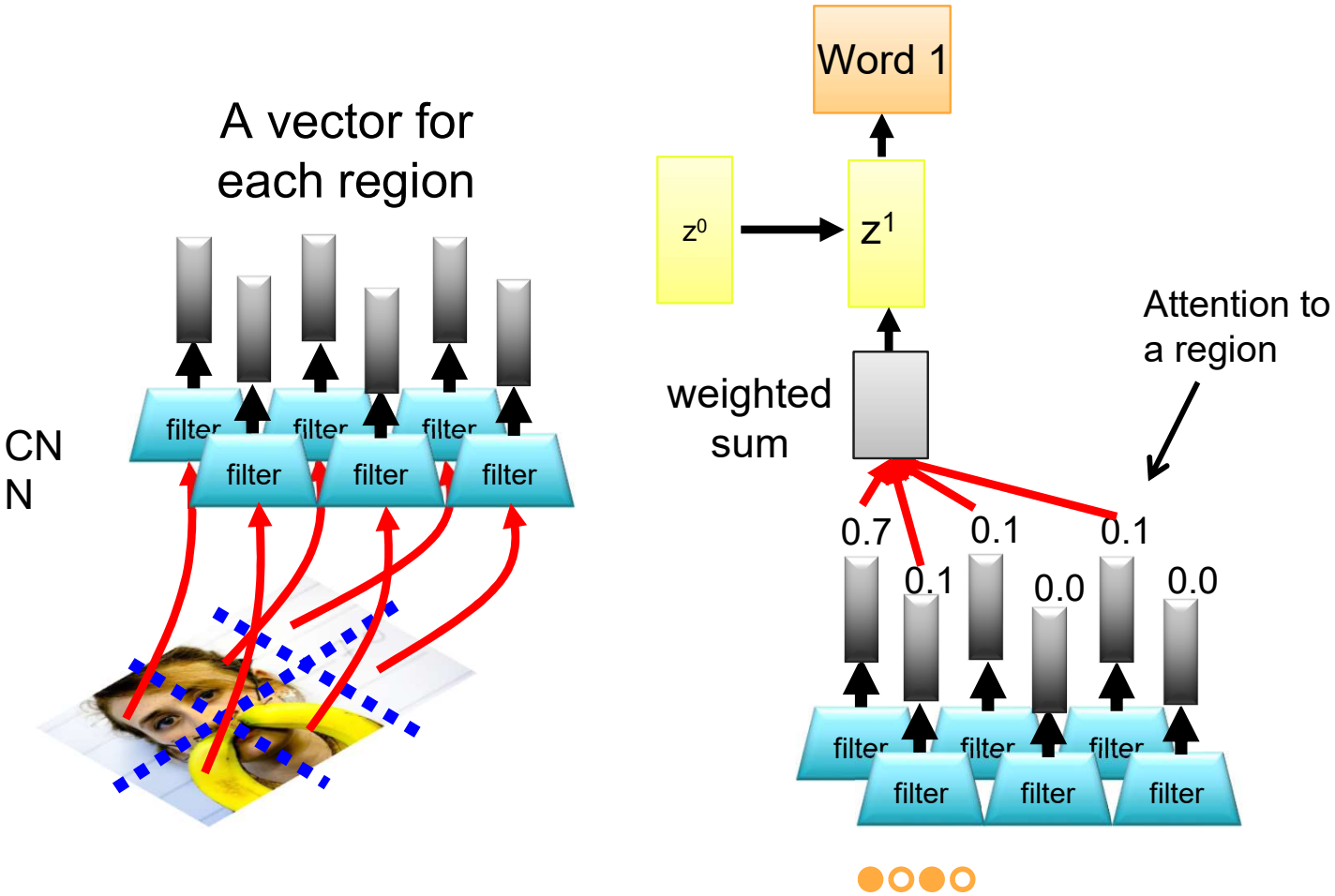


Image caption generation using attention

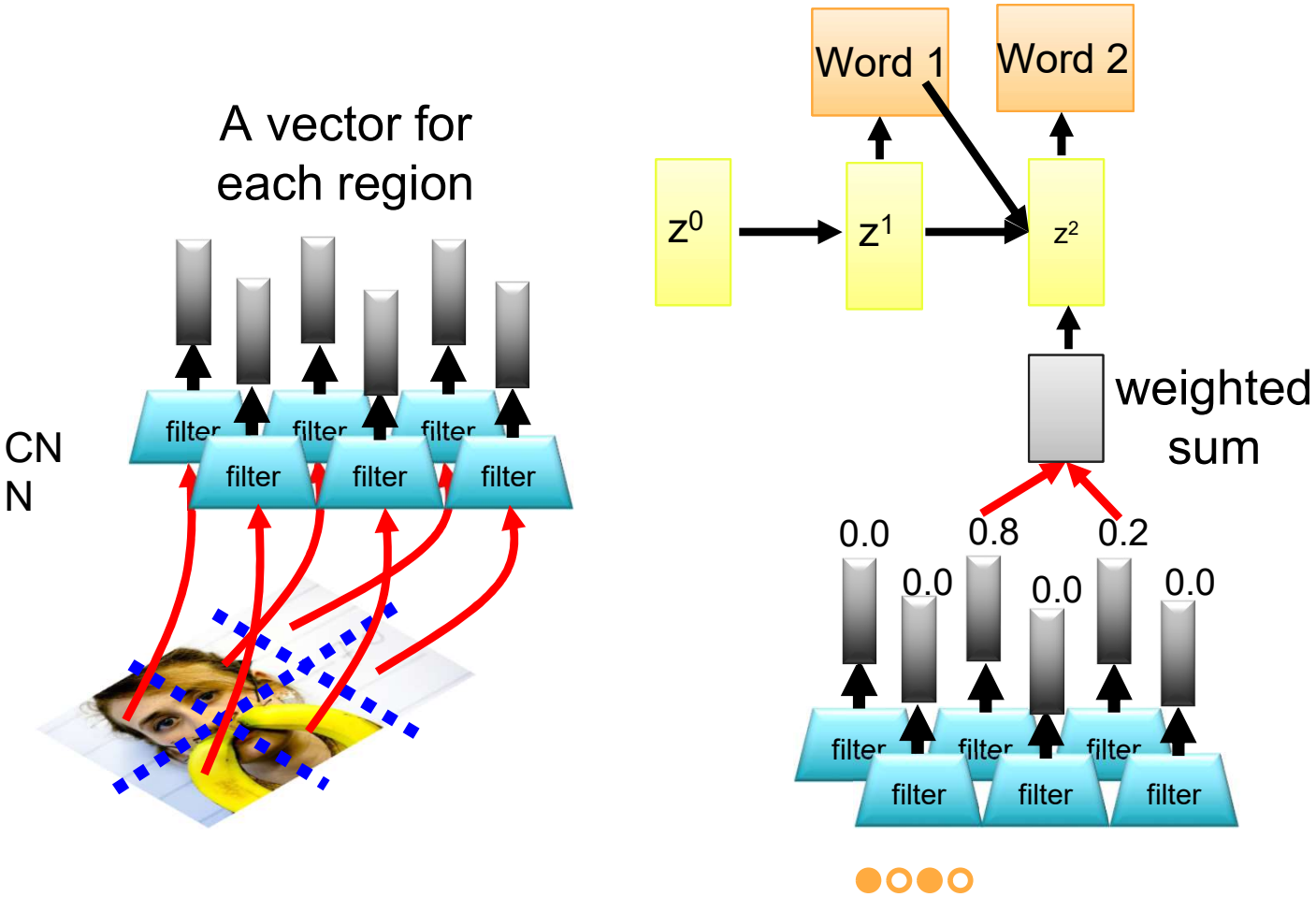


Image caption generation using attention

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

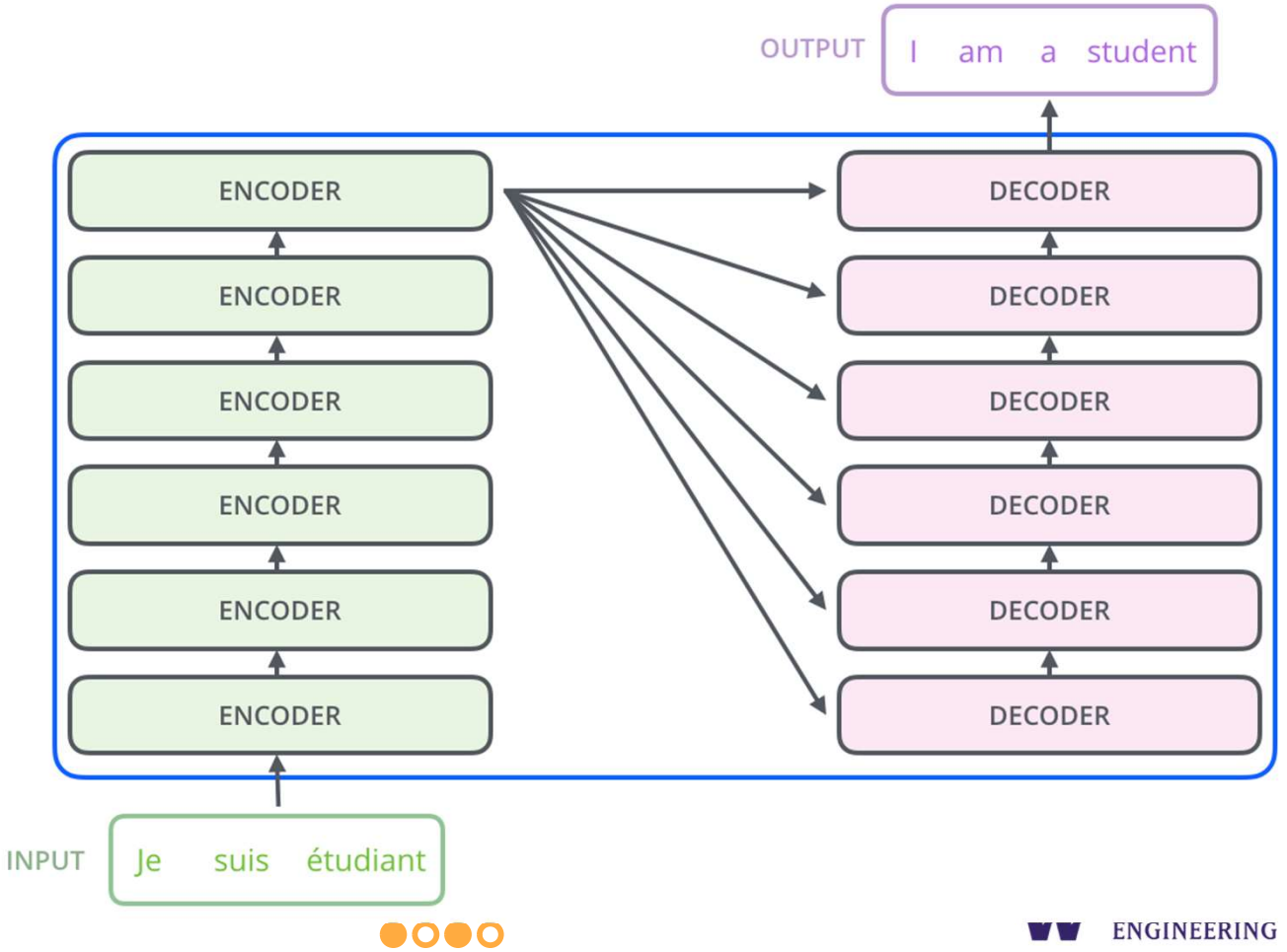
A giraffe standing in a forest with trees in the background.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, Yoshua Bengio, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML, 2015

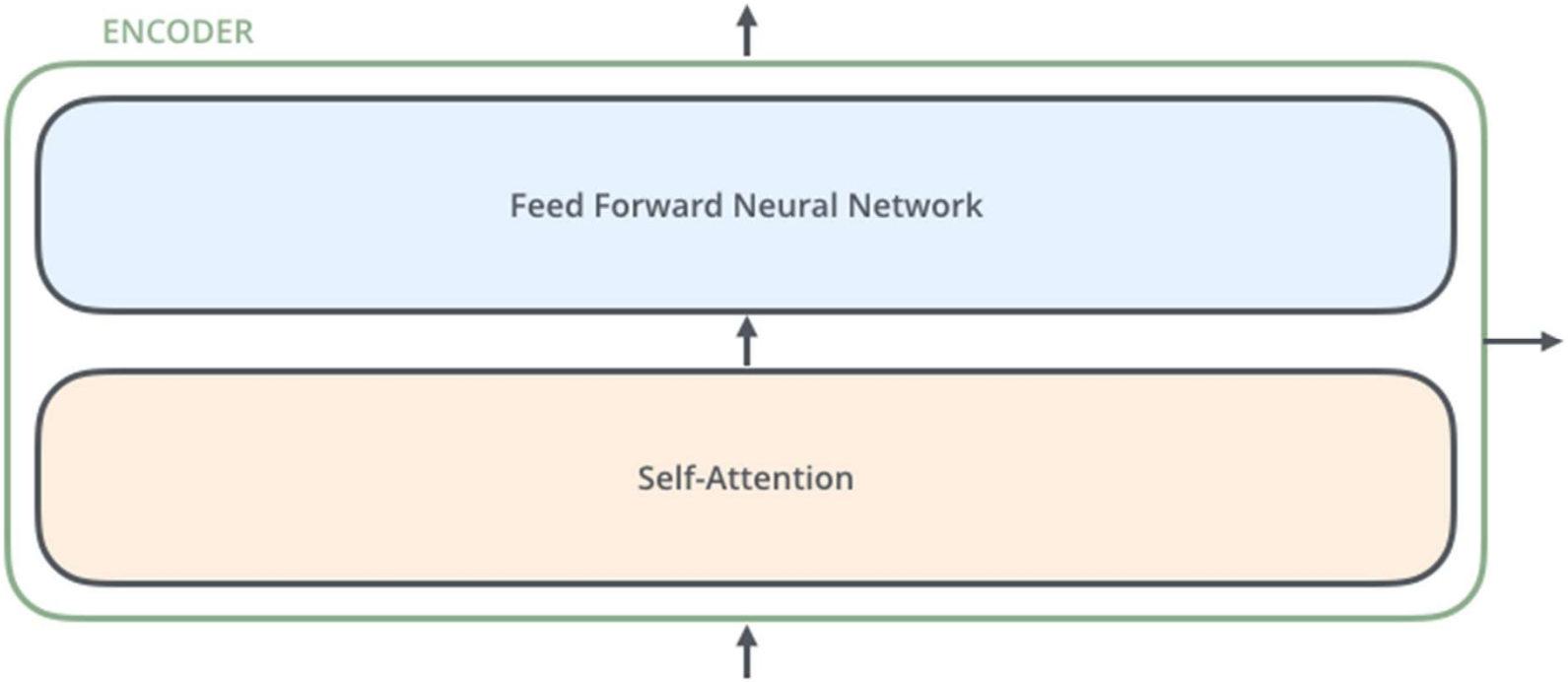
More new ideas:

1. ULM-FiT, pre-training, transfer learning in NLP
2. Recurrent models require linear sequential computation, hard to parallelize. ELMo, bidirectional LSTM.
3. In order to reduce such sequential computation, several models based on CNN are introduced, such as ConvS2S and ByteNet. Dependency for ConvS2S needs linear depth, and ByteNet logarithmic.
4. The transformer is the first transduction model relying entirely on self-attention to compute the representations of its input and output without using RNN or CNN.

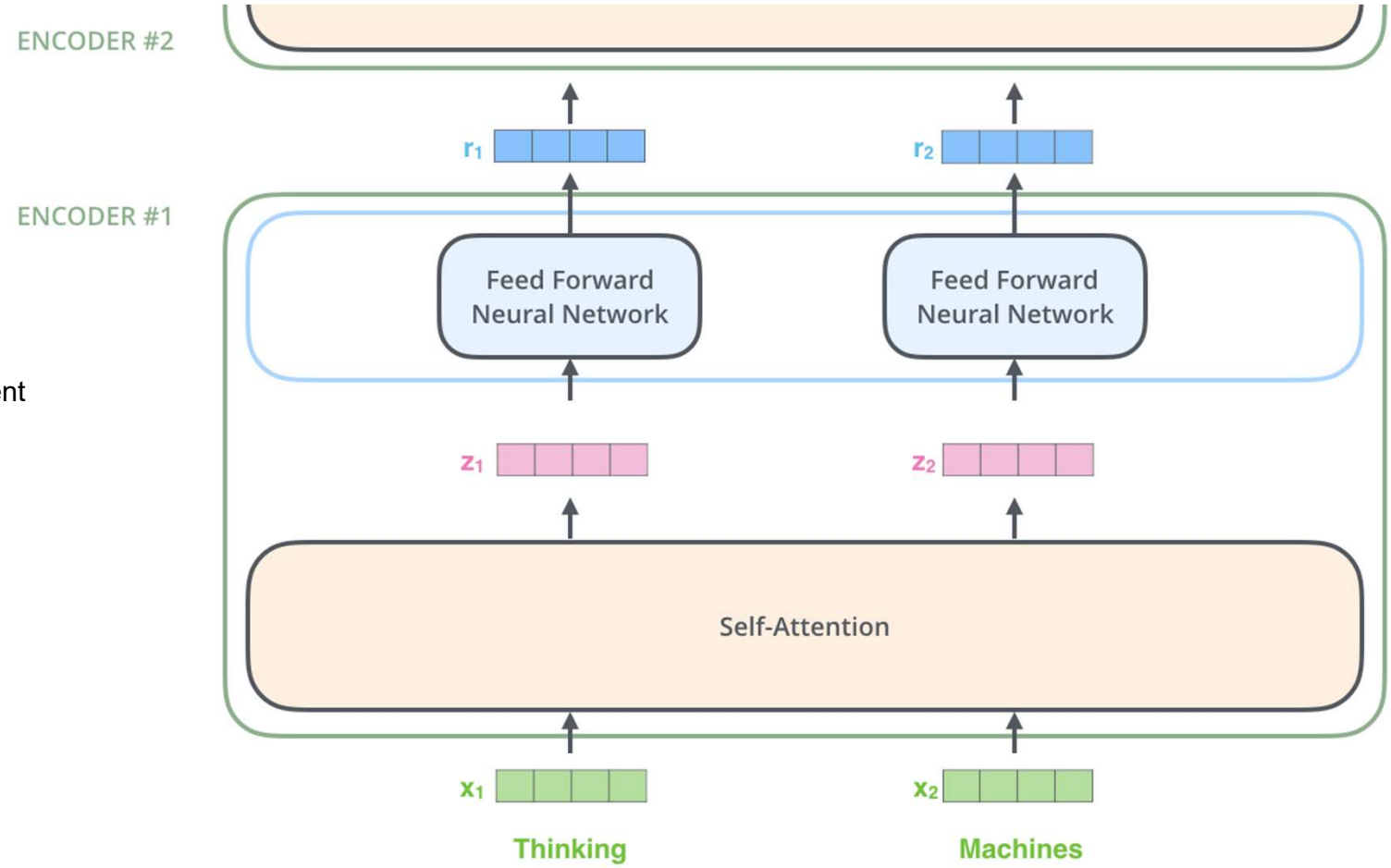
Transformer



An Encoder Block: same structure, different parameters

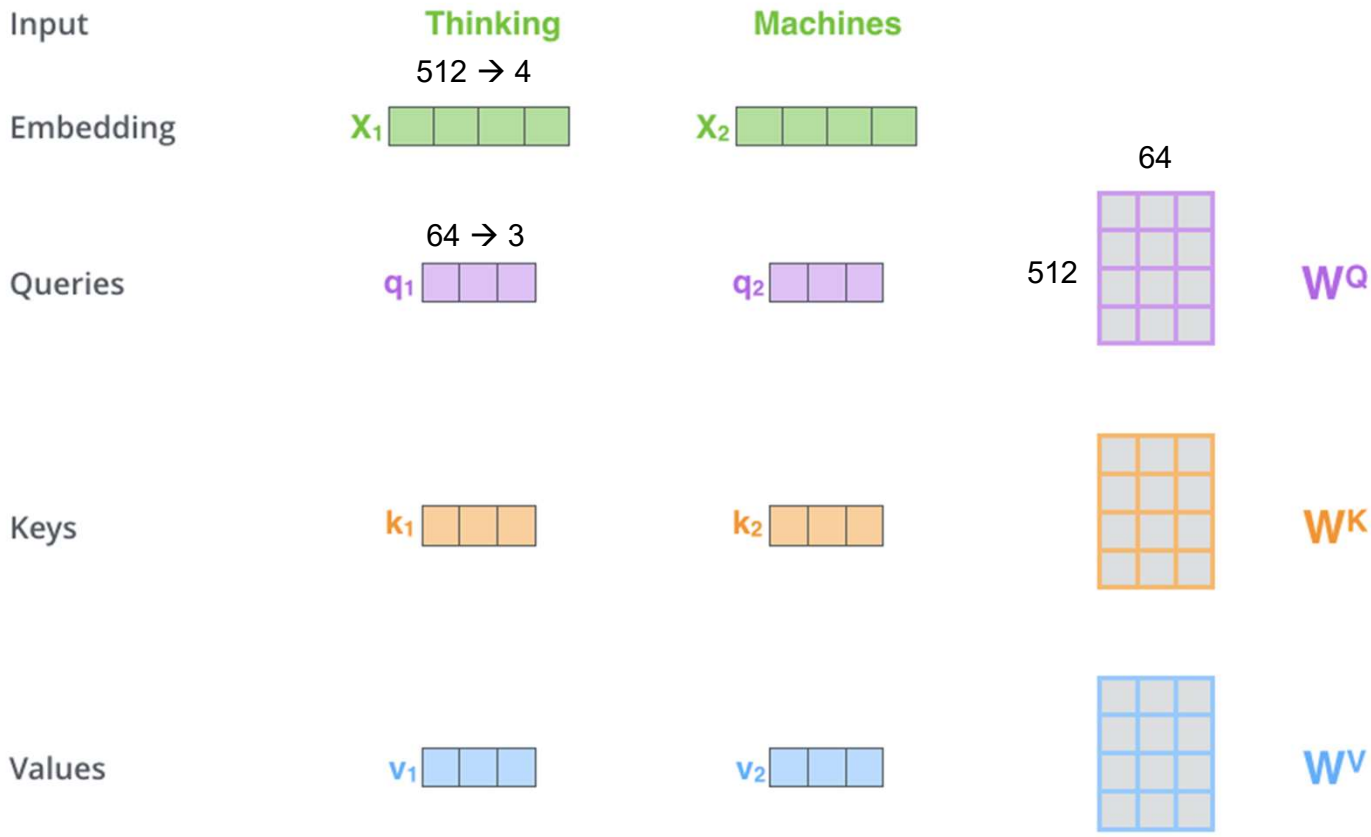


Encoder



Note: The ffnn is independent for each word.
Hence can be parallelized.

Self Attention

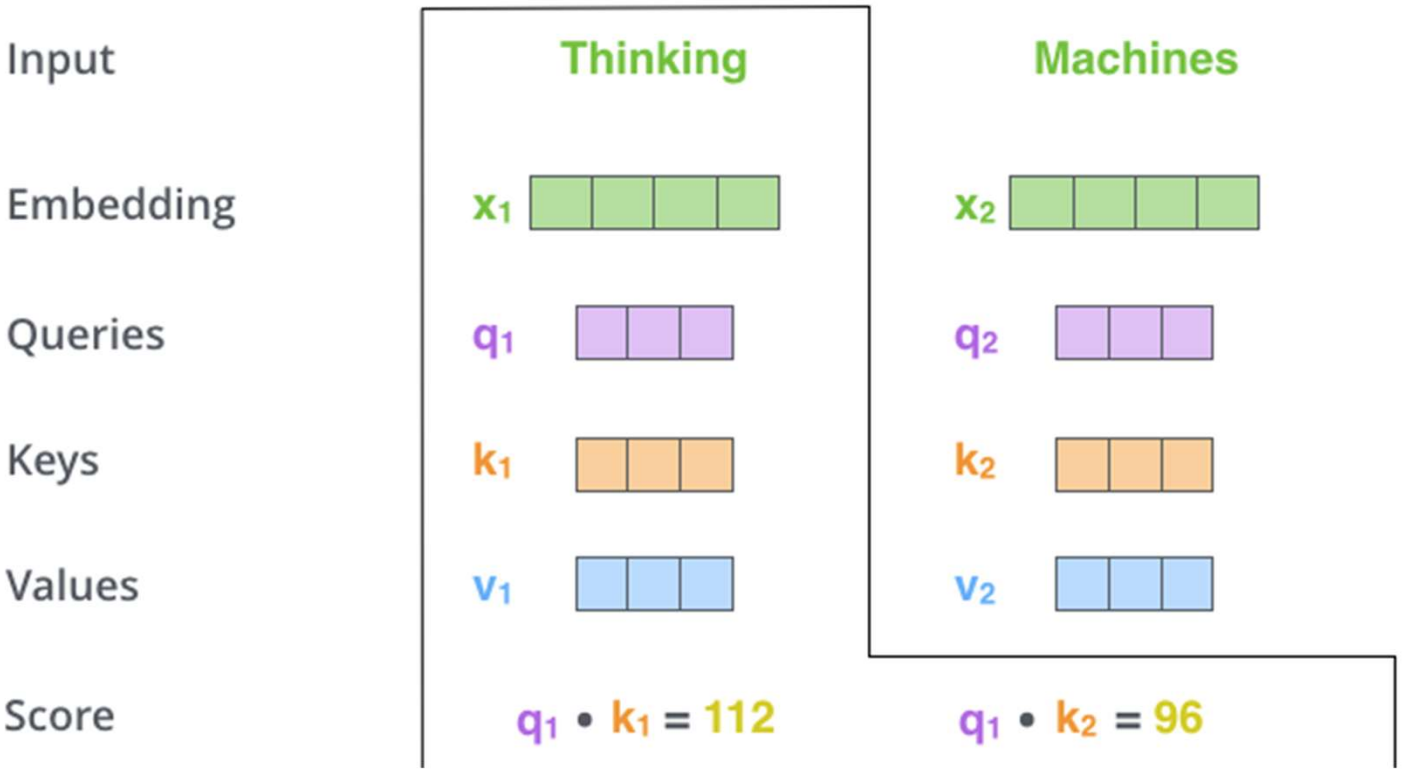


First we create three vectors by multiplying input embedding (1x512) x_i with three matrices (64x512):

$q_i = x_i W^Q$
$K_i = x_i W^K$
$V_i = x_i W^V$

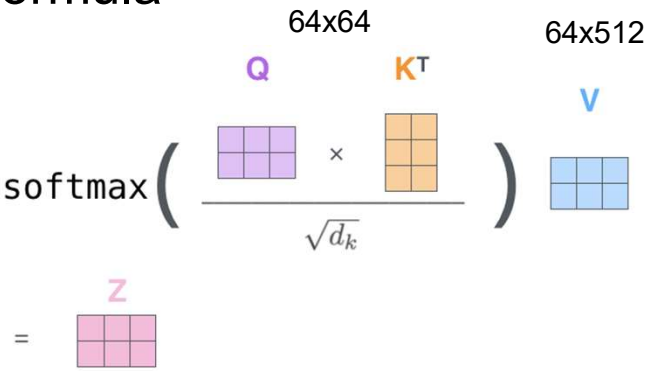
Self Attention

Now we need to calculate a score to determine how much focus to place on other Parts of the input.

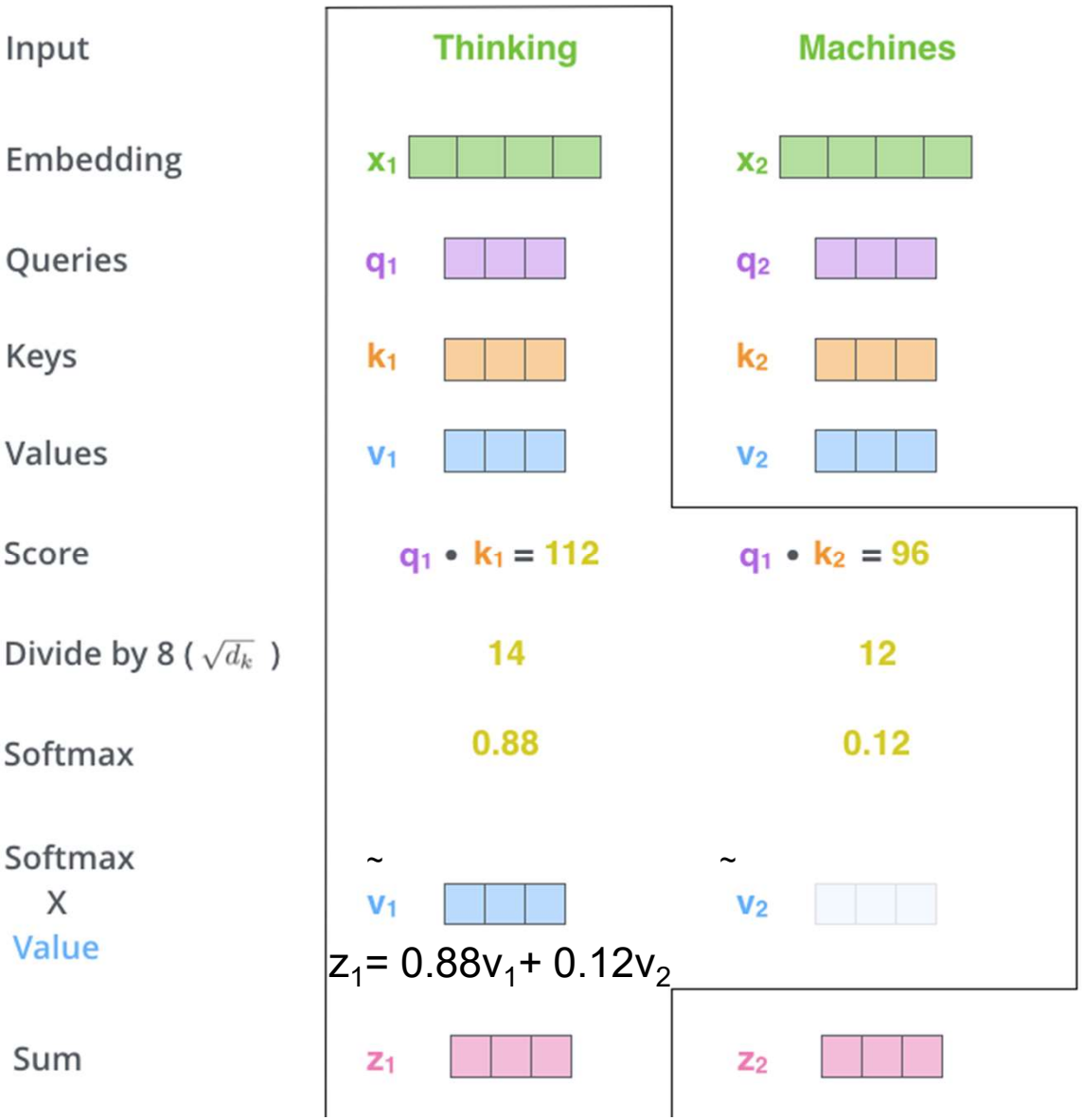


Self Attention

Formula

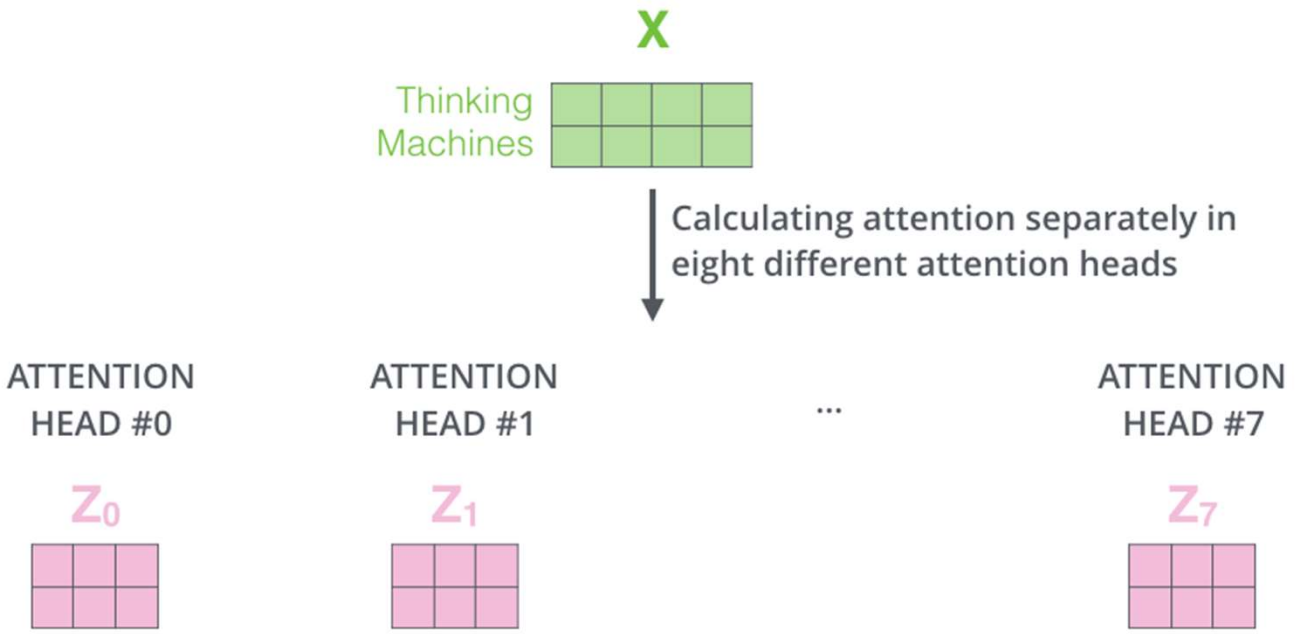


d_k=64 is dimension of key vector



Multiple heads

1. It expands the model's ability to focus on different positions.
2. It gives the attention layer multiple "representation subspaces"

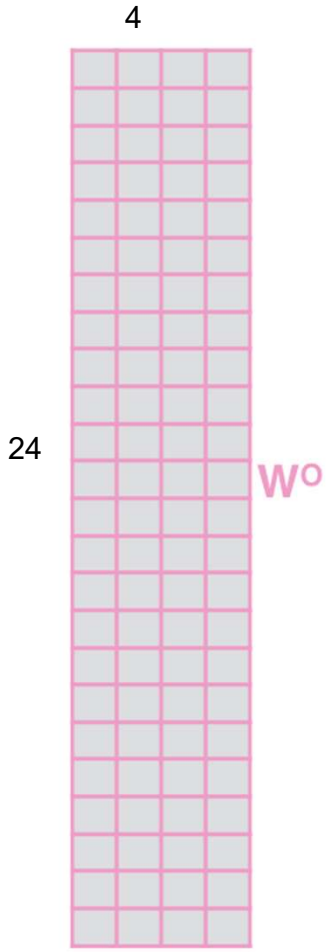


The output is expecting only a 2x4 matrix, hence,

1) Concatenate all the attention heads

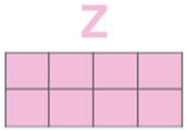
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=

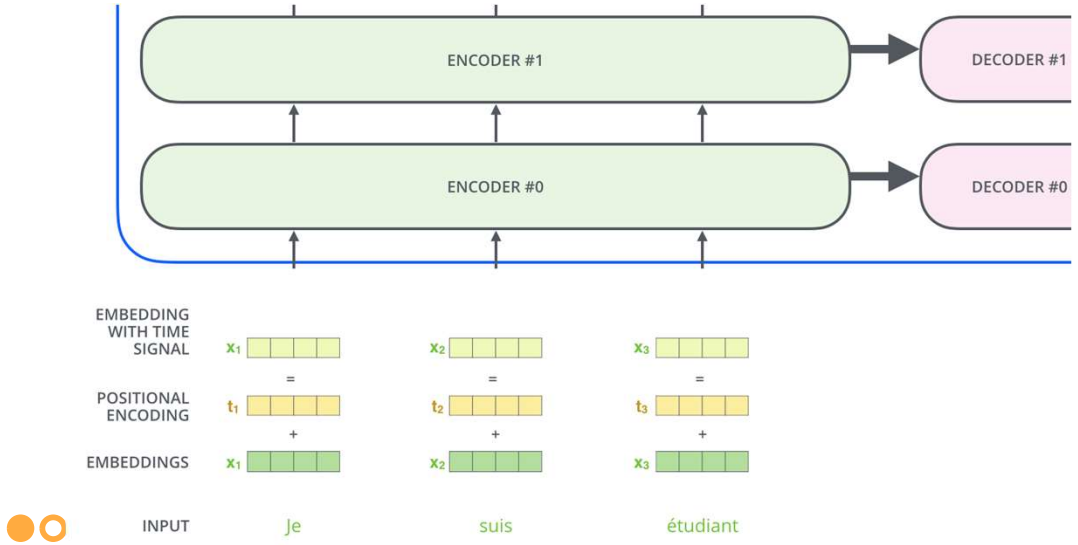


If you want some more intuition on attention: watch https://www.youtube.com/watch?v=-9vVhYEXeyQ

Representing the input order (positional encoding)

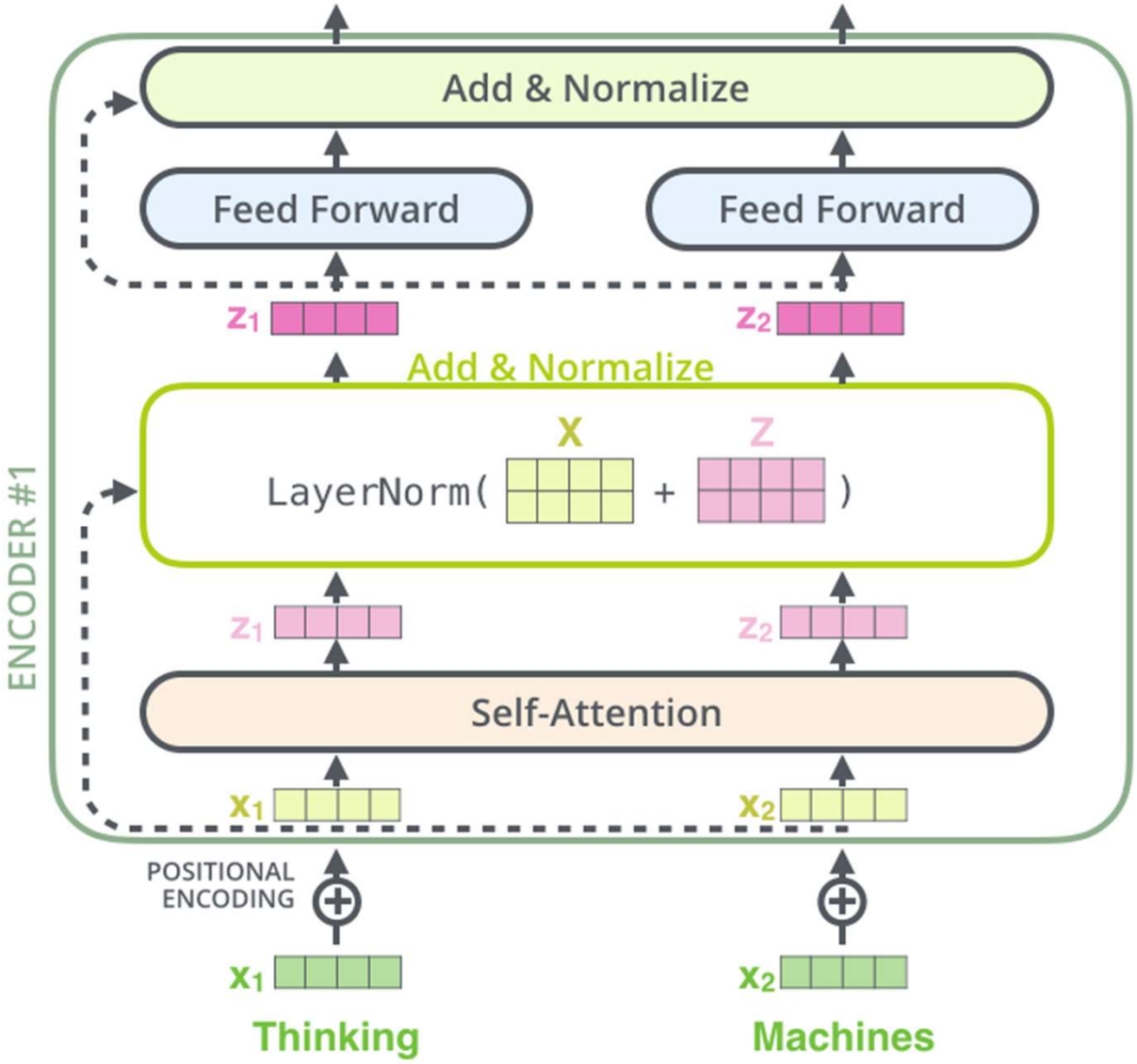
The transformer adds a vector to each input embedding. These vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence. The intuition here is that adding these values to the embedings provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention.

More on positional encoding:
https://kazemnejad.com/blog/transformer_architecture_positional_encoding/



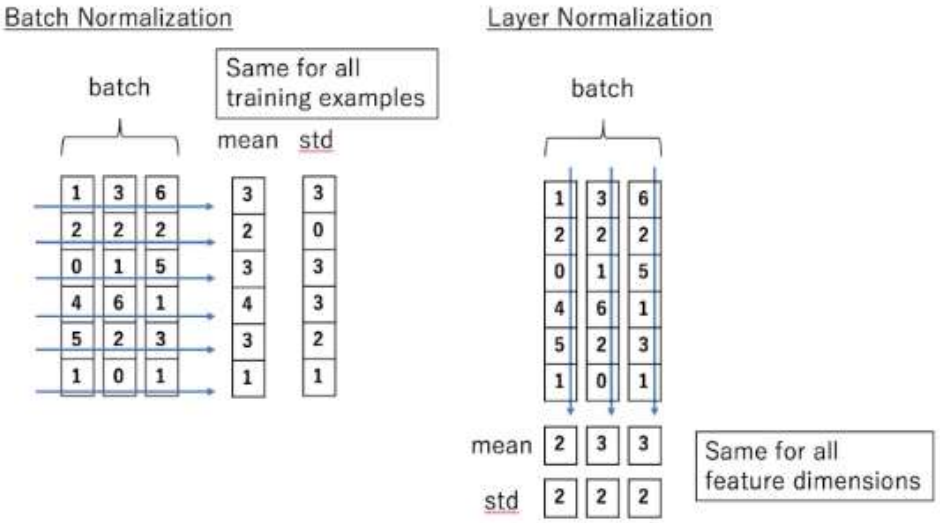
Add and Normalize

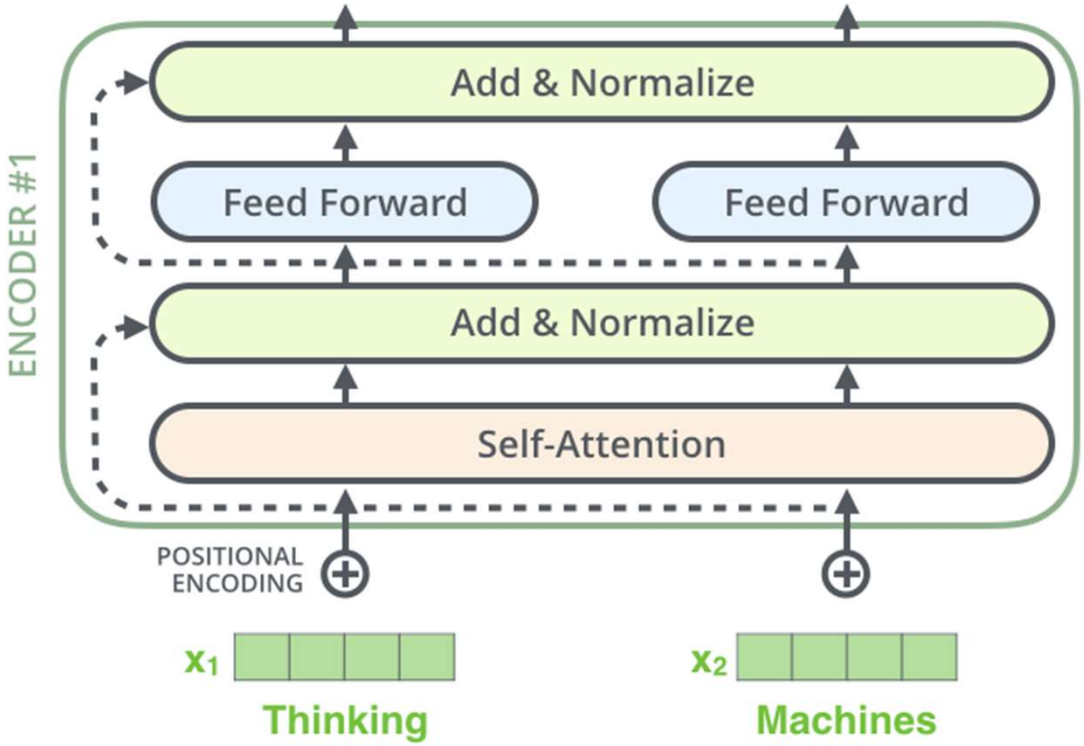
In order to regulate the computation, this is a normalization layer so that each feature (column) have the same average and deviation.



Layer Normalization (Hinton)

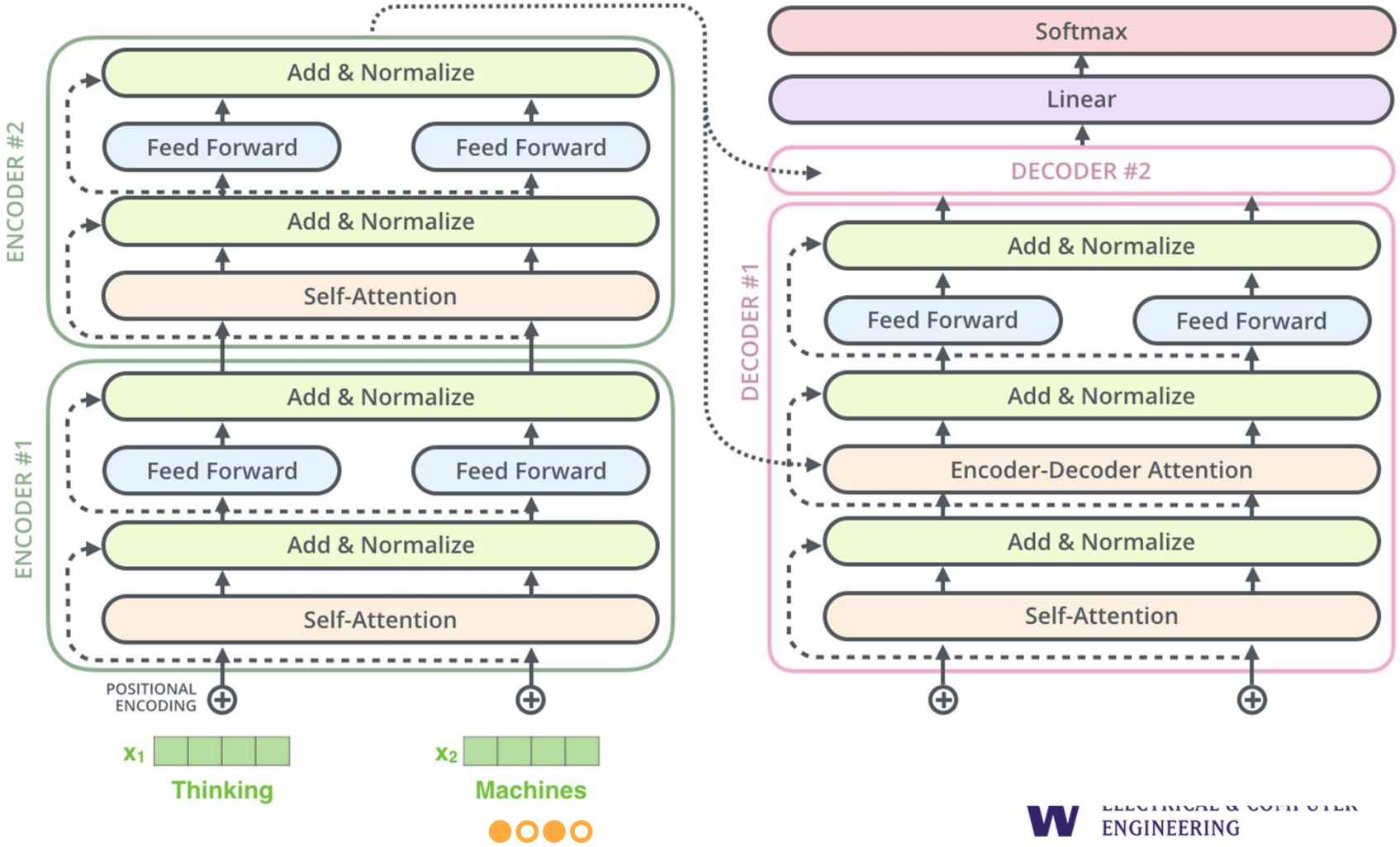
Layer normalization normalizes the inputs across the features.



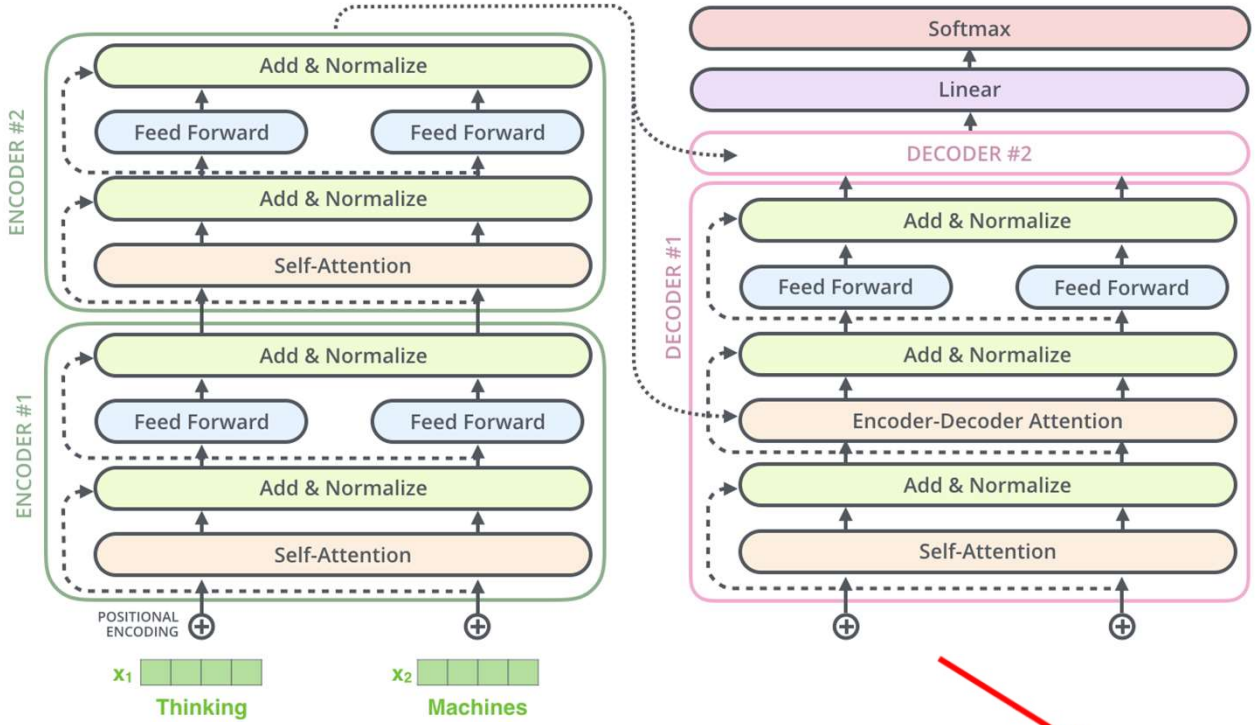


The Complete Transformer

The encoder-decoder attention is just like self attention, except it uses K, V from the top of encoder output, and its own Q



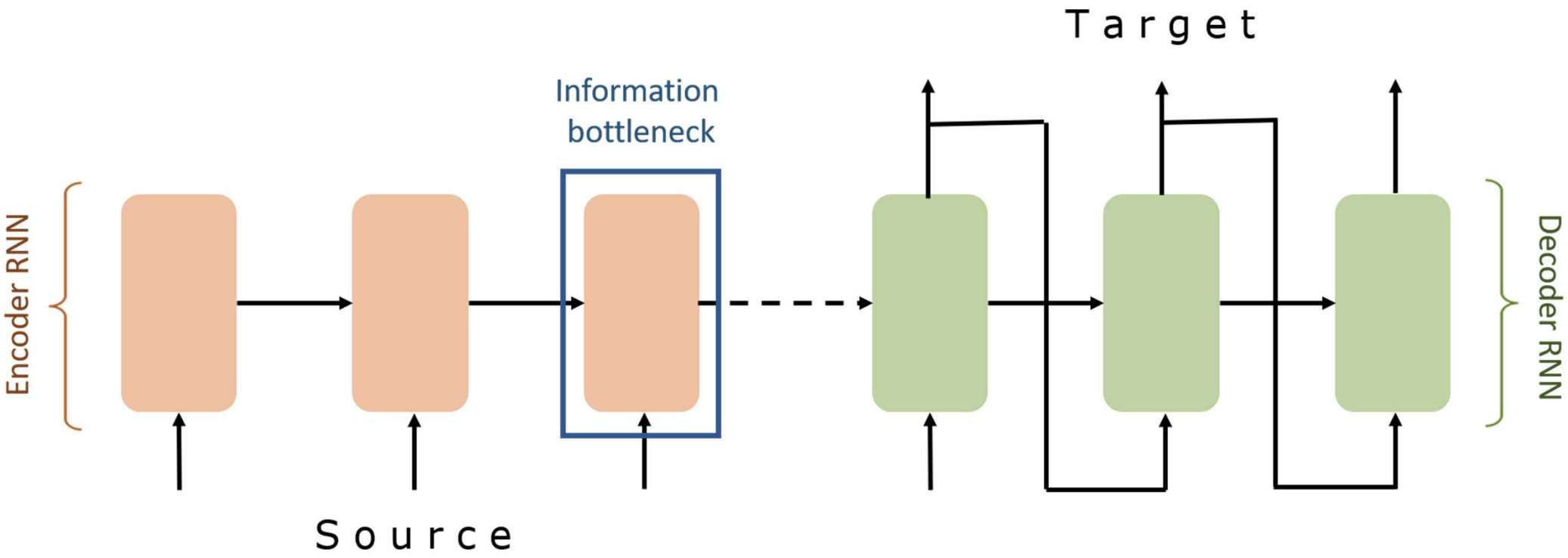
Next: Pretraining with BERT/GPT



GPT: Generative Pretrained Transformer

BERT: Bidirectional Encoder Representation from Transformers

Avoiding Information Bottleneck

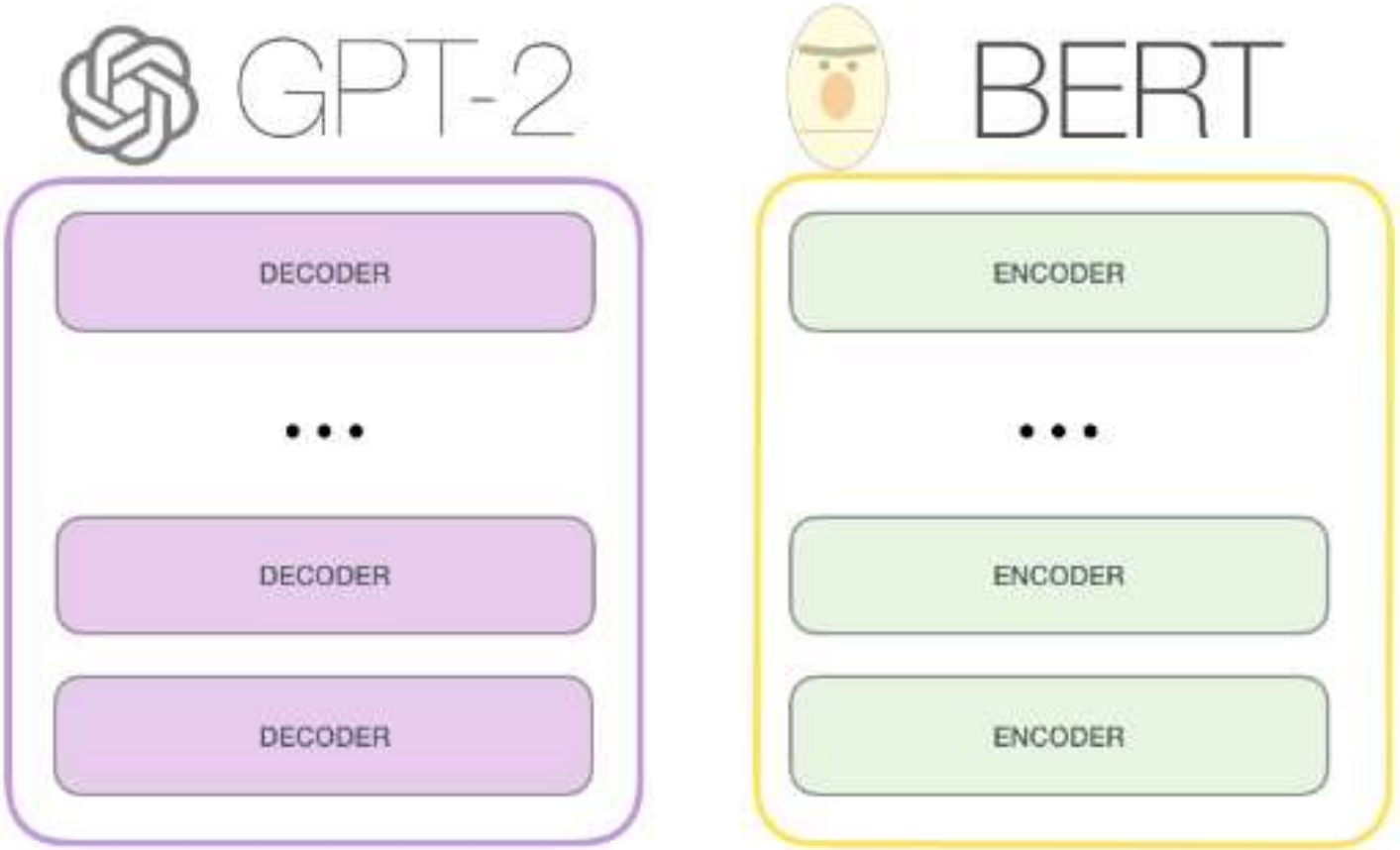


Transformers, GPT-2, and BERT

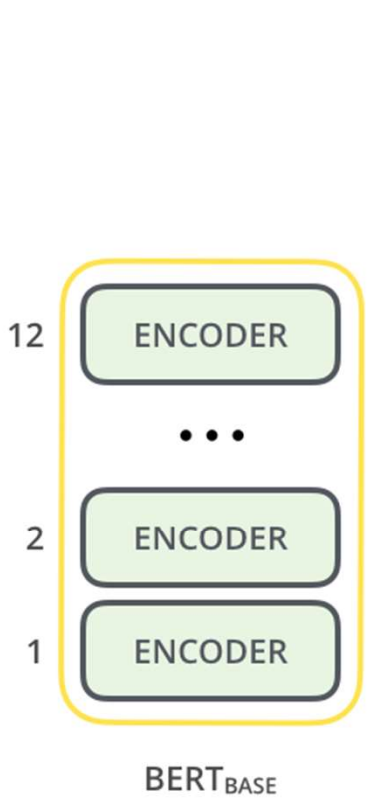
1. A transformer uses Encoder stack to model input, and uses Decoder stack to model output (using input information from encoder side).
2. But if we do not have input, we just want to model the "next word", we can get rid of the Encoder side of a transformer and output "next word" one by one. This gives us GPT.
3. If we are only interested in training a language model for the input for some other tasks, then we do not need the Decoder of the transformer, that gives us BERT.

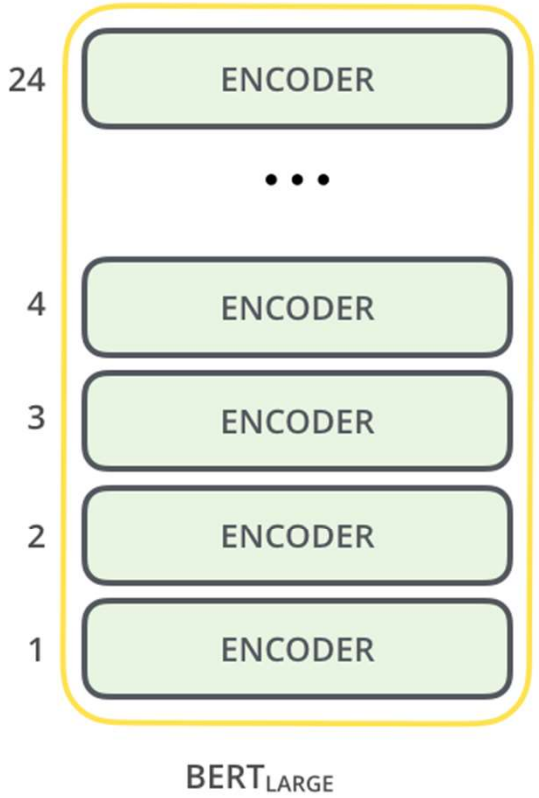
03

GPT-2, BERT


GPT-2
DECODER
DECODER
DECODER
BERT
ENCODER
ENCODER
ENCODER

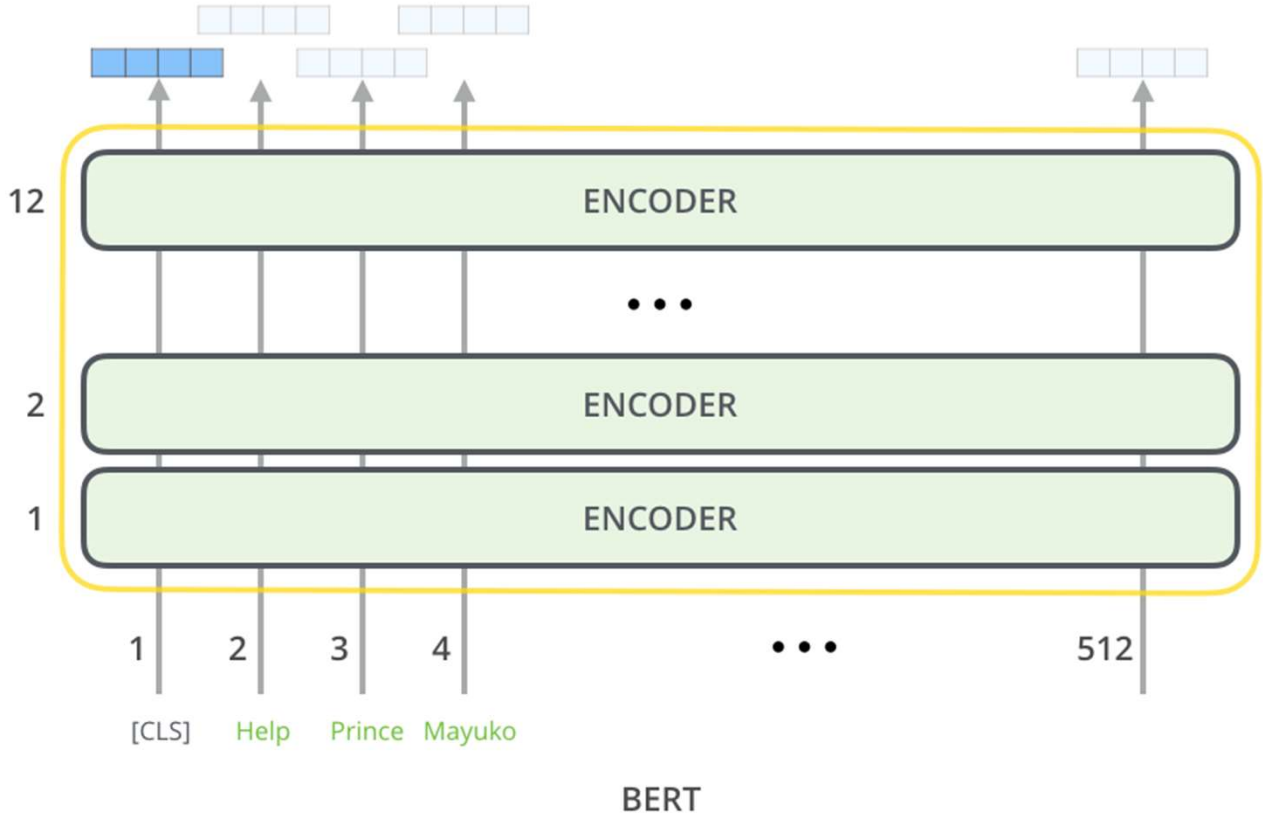
BERT (Bidirectional Encoder Representation from Transformers)





Model input dimension 512

Input and output vector size

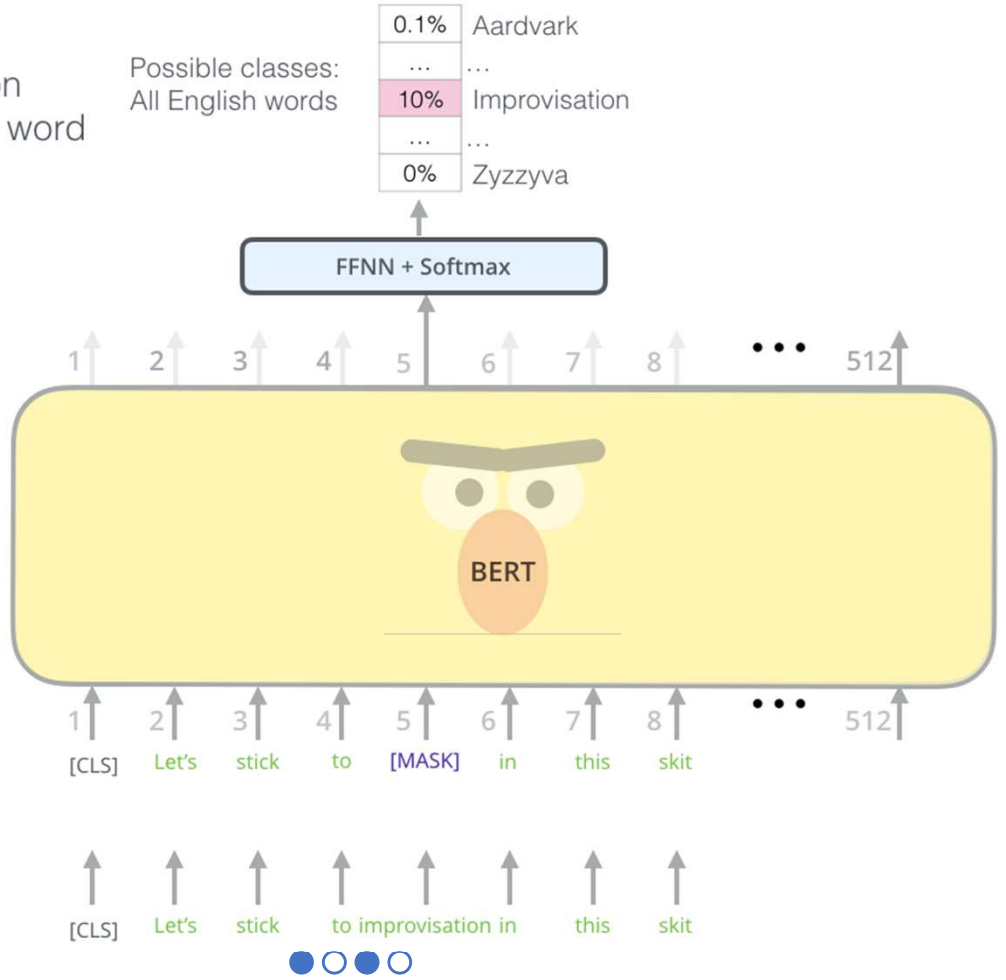


BERT Pretraining

ULM-FiT (2018): Pre-training ideas, transfer learning in NLP.
ELMo: Bidirectional training (LSTM)
Transformer: Although used things from left, but still missing from the right.
GPT: Use Transformer Decoder half.
BERT: Switches from Decoder to Encoder, so that it can use both sides in training and invented corresponding training tasks: masked language model

BERT Pretraining Task 1: Masked Words

Use the output of the masked word's position to predict the masked word

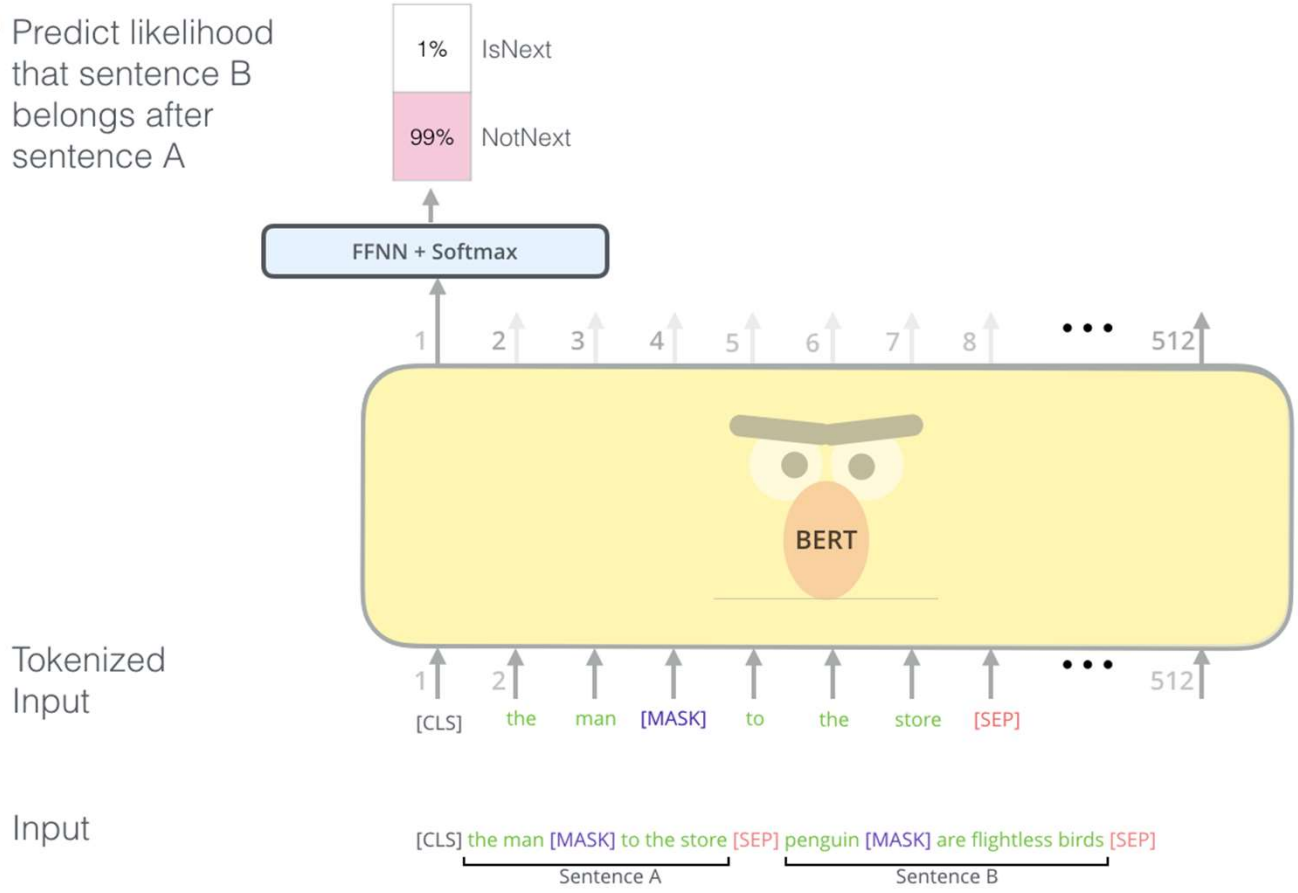


Out of this 15%,
80% are [Mask],
10% random words
10% original words

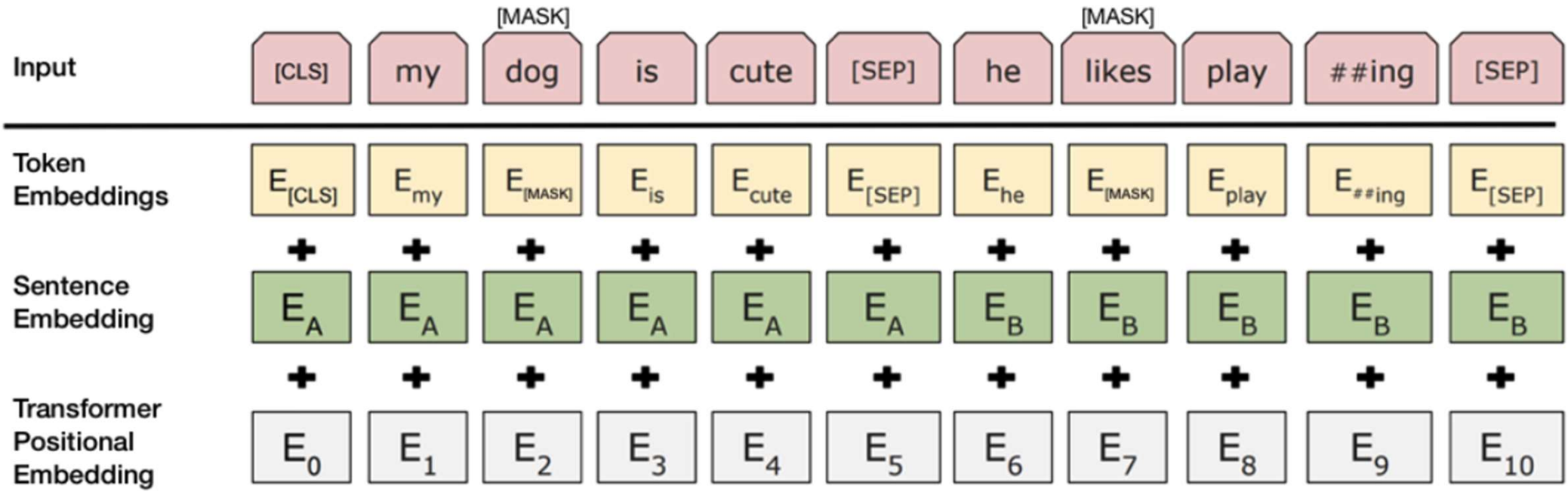
Randomly mask 15% of tokens

Input

BERT Pretraining Task 2: Two Sentences



BERT Pretraining Task 2: Two Sentences

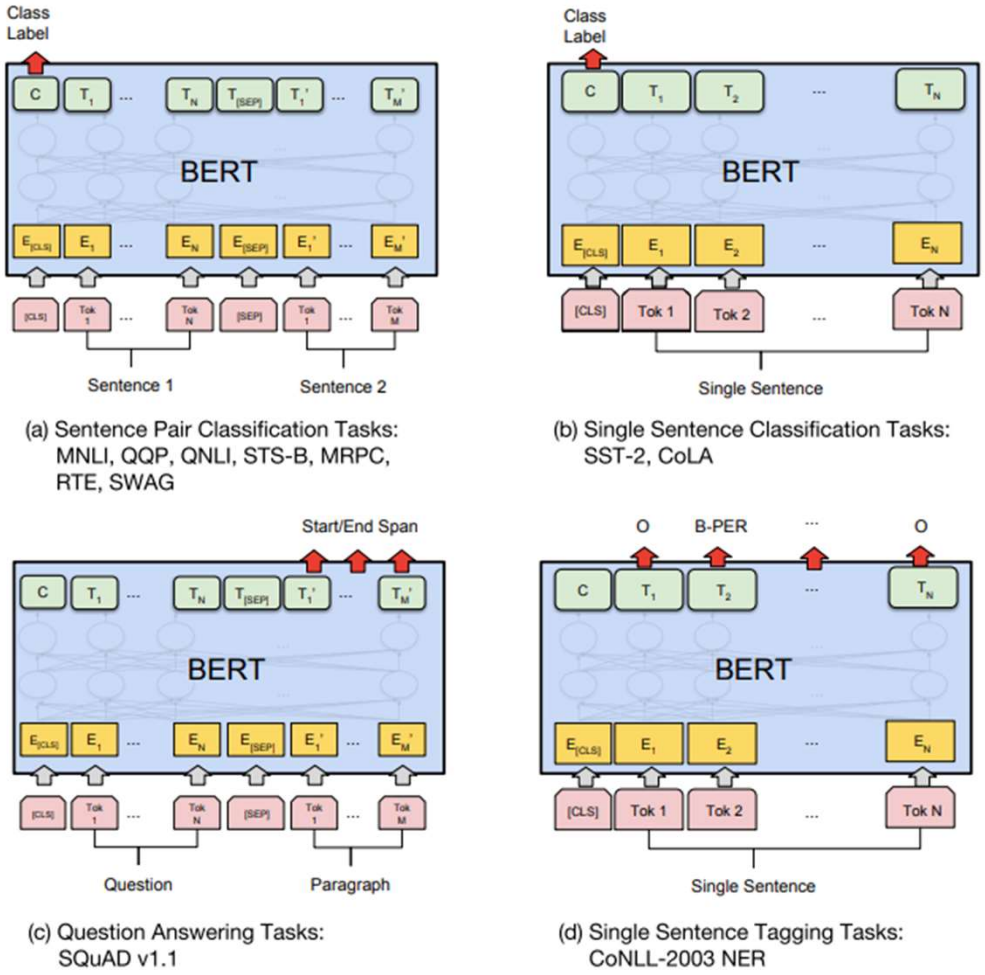


50% true second sentences

50% random second sentences

Fine-Tuning BERT for Specific Tasks

MNLI
QQP (Quaro Question Pairs)
Semantic equivalence)
QNLI (NL inference dataset)
STS-B (texture similarity)
MRPC (paraphrase, Microsoft)
RTE (textual entailment)
SWAG (commonsense inference)
SST-2 (sentiment)
CoLA (linguistic acceptability
SQuAD (question and answer)



SST (Stanford sentiment treebank): 215k phrases with fine-grained sentiment labels in the parse trees of 11k sentences.

GPT-2, BERT

NLP Tasks (SQuAD -- Stanford Question Answering Dataset):

Sample: Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California. As this was the 50th Super Bowl, the league emphasized the "golden anniversary" with various gold-themed initiatives, as well as temporarily suspending the tradition of naming each Super Bowl game with Roman numerals (under which the game would have been known as "Super Bowl L"), so that the logo could prominently feature the Arabic numerals 50.

Which NFL team represented the AFC at Super Bowl 50?

Ground Truth Answers: Denver Broncos

Which NFL team represented the NFC at Super Bowl 50?

Ground Truth Answers: Carolina Panthers

Add Indices for Sentences and Paragraphs

SegaTron/SegaBERT

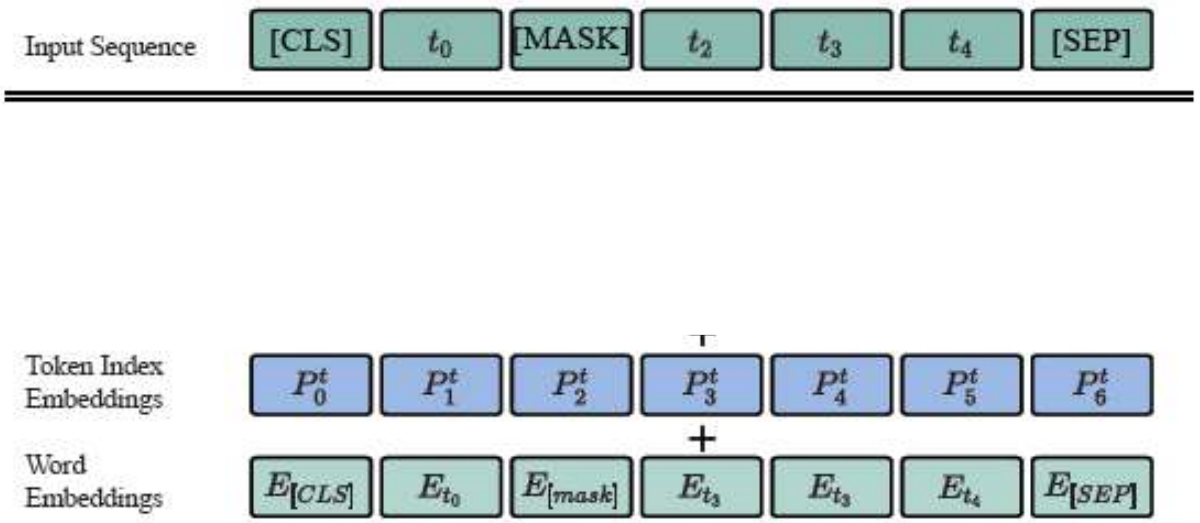
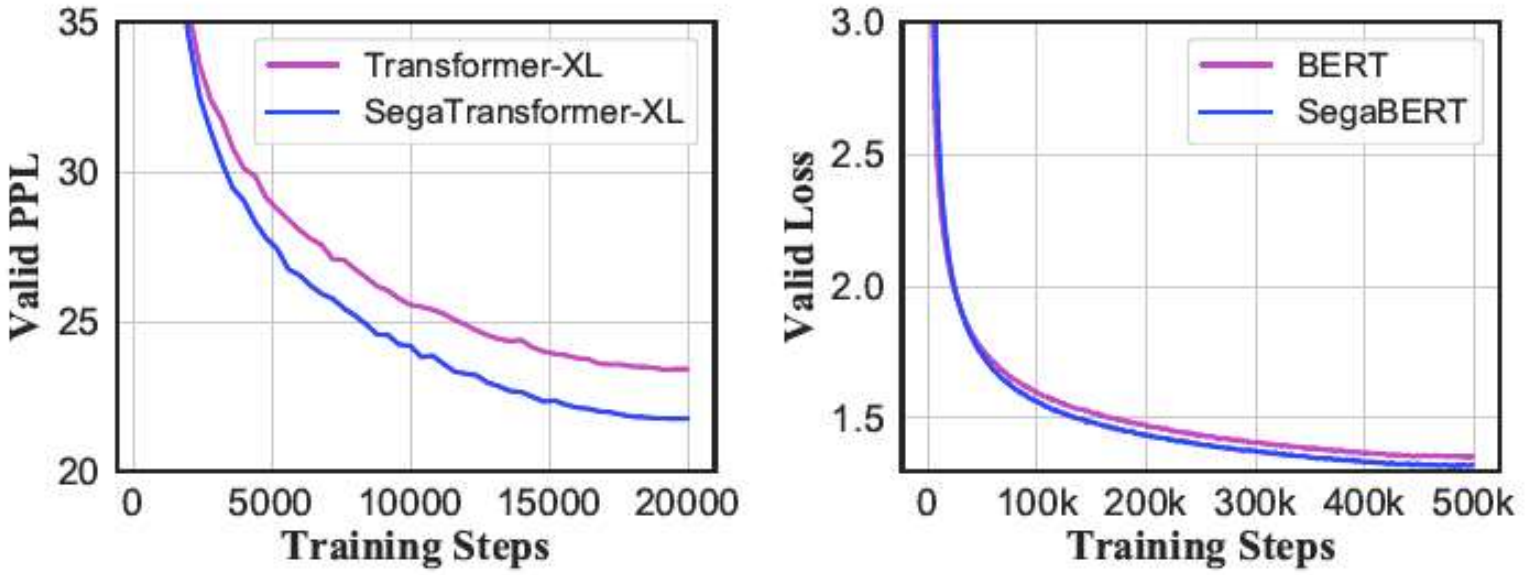


Figure 1: Input Representation of SegaBERT

H. Bai, S. Peng, J. Lin, L. Tan, K. Xiong, W. Gao, M. Li: SgaTron: Segment-aware transformer for language modeling and understanding. AAAI'2021

Conversion speed much faster ：



(a) Language modeling perplexities

(b) Pre-training losses

Figure 2: Valid perplexities and losses during the training processes of language modeling and pre-training.

Testing on GLUE dataset

Task(Metrics)	BASE model(wikipedia 500K steps)				LARGE model(wikibooks 1000K steps)			
	dev		test		dev		test	
	BERT	SegaBERT	BERT	SegaBERT	BERT	SegaBERT	BERT	SegaBERT
CoLA (Matthew Corr.)	**55.0**	54.7	43.5	**50.7**	60.6	**65.3**	60.5	**62.6**
SST-2 (Acc.)	91.3	**92.1**	91.2	**91.5**	93.2	**94.7**	**94.9**	94.8
MRPC (F1)	**92.6**	92.4	88.9	**89.3**	-	92.3	89.3	**89.7**
STS-B (Spearman Corr.)	88.9	**89.0**	83.9	**84.6**	-	90.3	86.5	**88.6**
QQP (F1)	86.5	**87.0**	70.8	**71.4**	-	89.1	72.1	**72.5**
MNLI-m (Acc.)	83.2	**83.8**	82.9	**83.5**	86.6	**87.6**	86.7	**87.9**
MNLI-mm (Acc.)	83.4	**84.1**	82.8	**83.2**	-	87.5	85.9	**87.7**
QNLI (Acc.)	90.4	**91.5**	90.1	**90.8**	92.3	**93.6**	92.7	**94.0**
RTE (Acc.)	68.3	**71.8**	65.4	**68.1**	70.4	**78.3**	70.1	**71.6**
Average	82.2	**82.9**	77.7	**79.2**	-	86.5	82.1	**83.3**

Table 2: The results on GLUE benchmark. All base models are pre-trained by this work. Every result of the dev set is the average score of 4 times finetuning with different random seeds. Scores of BERT large dev are from (Sun et al., 2019) and scores of BERT large test are from (Devlin et al., 2018).

H. Bai, S. Peng, J. Lin, L. Tan, K. Xiong, W. Gao, M. Li: SgaTron: Segment-aware transformer for language modeling and understanding. AAAI'2021

Reading Comprehesion – SQUAD Tasks

System	Dev	
	EM	F1
BERT base (Single)	80.8	88.5
BERT large (Single	84.1	90.9
BERT large (Single+DA)	84.2	91.1
KT-NET	85.2	91.7
StructBERT Large (Single)	85.2	92.0
SegaBERT base (Single)	83.2	90.2
SegaBERT large (Single)	85.3	92.4

Table 3: Evaluation results of SQUAD v1.1.

System	Dev	
	EM	F1
BERT base	72.3	75.6
BERT base (ours)	75.4	78.2
SegaBERT base	76.3	79.2
BERT large	78.7	81.9
BERT large wwm	80.6	83.4
SegaBERT large	81.8	85.2

Table 4: Evaluation results of SQUAD v2.0.

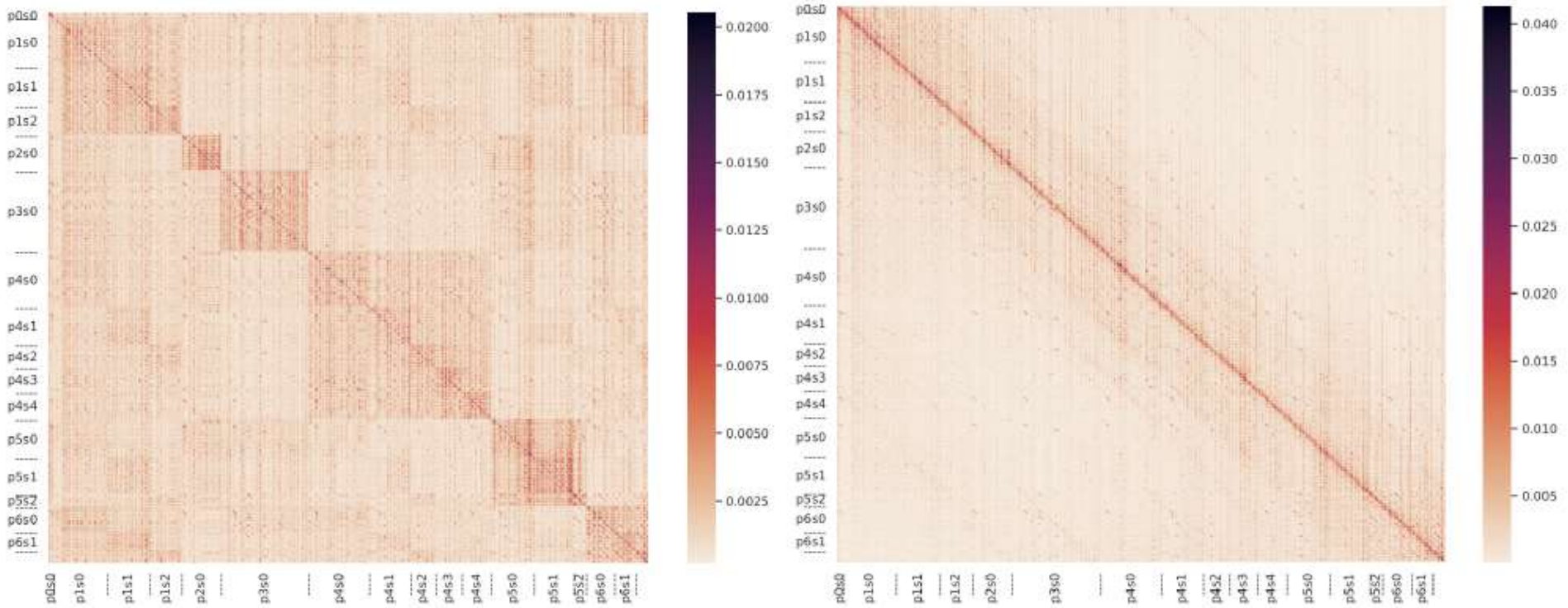
F1 = 2 (P*R) / (P+R), P is precision, R is recall, all in percentage, EM – exact match

Improving Transformer-XL

Model	#Param.	PPL
LSTM+Neural cache (Grave et al., 2017)	-	40.8
Hebbian+Cache (Rae et al., 2018)	-	29.9
Transformer-XL base, M=150 (Dai et al., 2019)	151M	24.0
Transformer-XL base, M=150 (ours)	151M	24.4
SegaTransformer-XL base, M=150	151M	**22.5**
Adaptive Input (Baevski and Auli, 2019)	247M	18.7
Transformer-XL large, M=384 (Dai et al., 2019)	257M	18.3
Compressive Transformer, M=1024 (Rae et al., 2020)	257M	17.1
SegaTransformer-XL large, M=384	257M	**17.1**

Table 1: Comparison with Transformer-XL and competitive baseline results on WikiText-103.

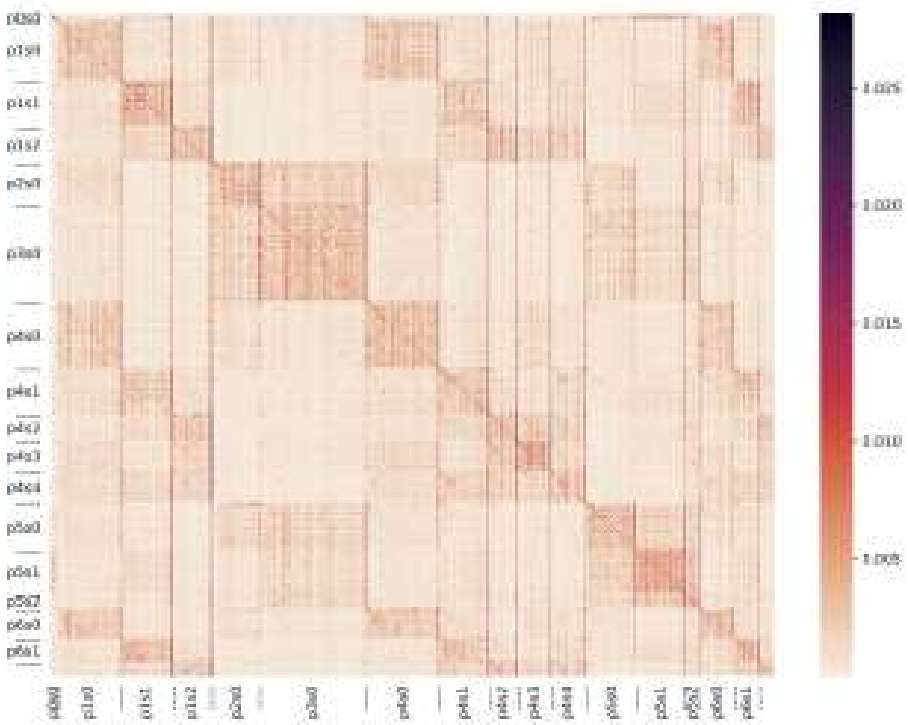
Looking at Attention



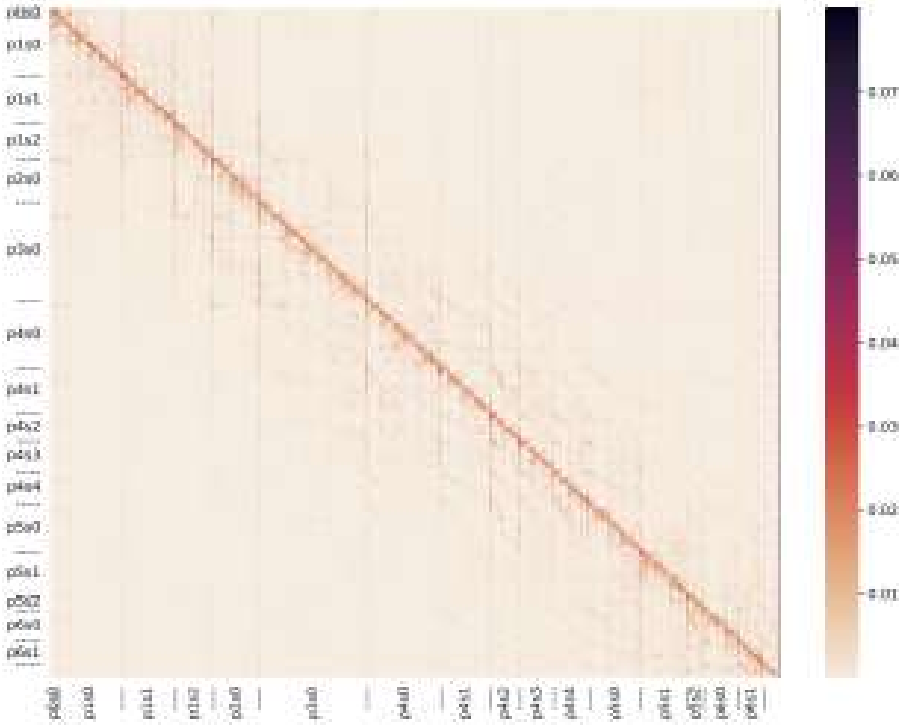
(a) SegaBERT-Layer 1

(b) BERT-Layer 1

Looking at Attention



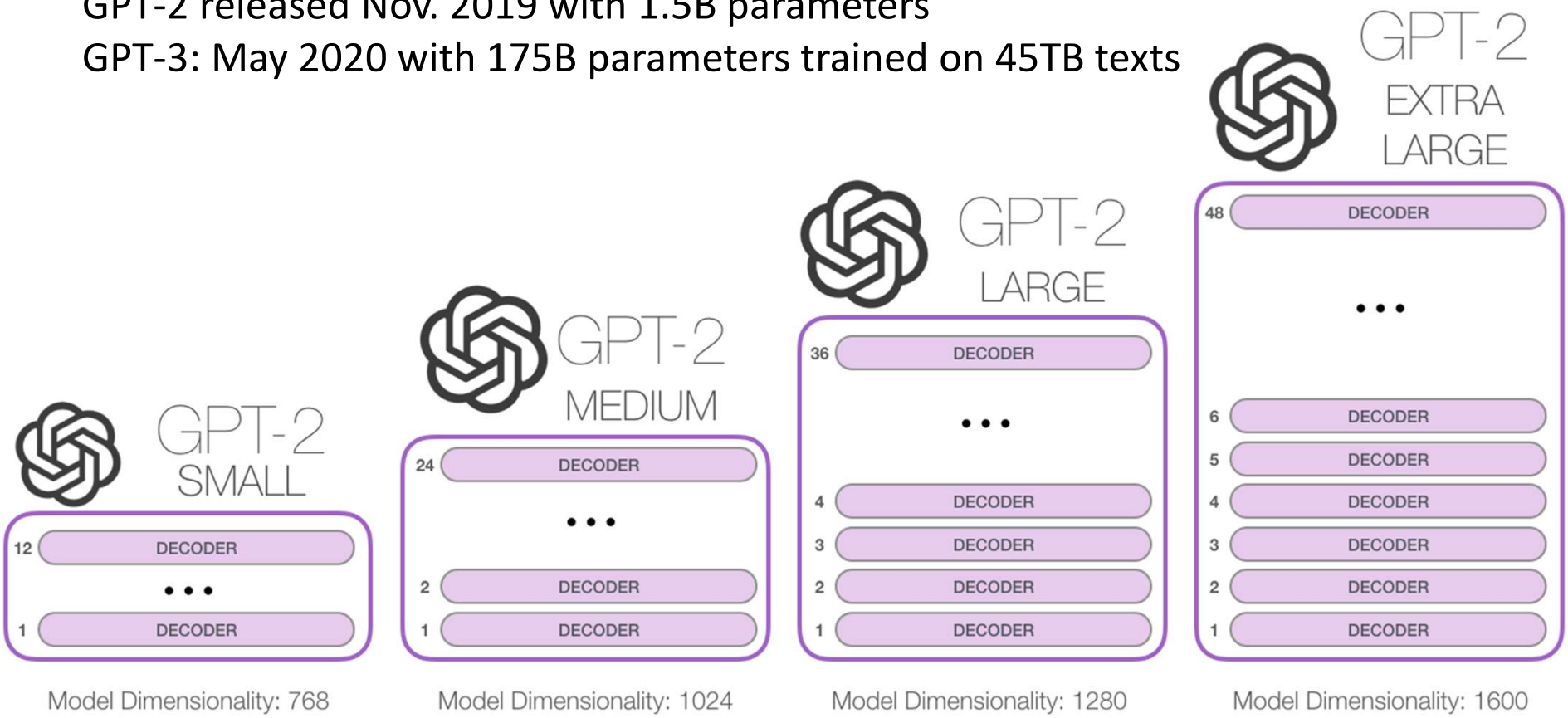
(a) SegaBERT-Layer 6



(b) BERT-Layer 6

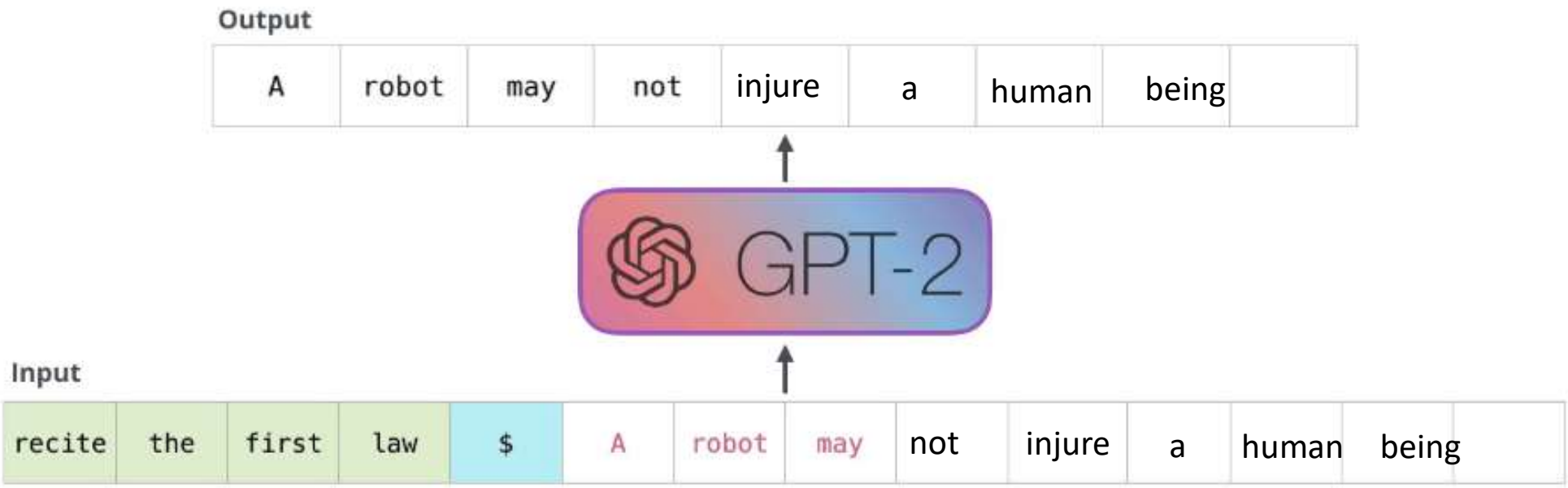
GPT-2, BERT

GPT released June 2018 (OpenAI)
GPT-2 released Nov. 2019 with 1.5B parameters
GPT-3: May 2020 with 175B parameters trained on 45TB texts



117M parameters 345M 762M 1542M

GPT-2 in action



Byte Pair Encoding (BPE)

Word embedding sometimes is too high level, pure character embedding too low level. For example, if we have learned

old older oldest

We might also wish the computer to infer

smart smarter smartest

But at the whole word level, this might not be so direct. Thus the idea is to break the words up into pieces like er, est, and embed frequent fragments of words.

GPT adapts this BPE scheme.

Byte Pair Encoding (BPE)

GPT uses BPE scheme. The subwords are calculated by:

1. Split word to sequence of characters (add </w> char)
2. Joining the highest frequency pattern.
3. Keep doing step 2, until it hits the pre-defined maximum number of sub-words or iterations.

Example (5, 2, 6, 3 are number of occurrences)

{‘l o w </w>’: 5, ‘l o w e r </w>’: 2, ‘n e w e s t </w>’: 6, ‘w i d e s t </w>’: 3 }

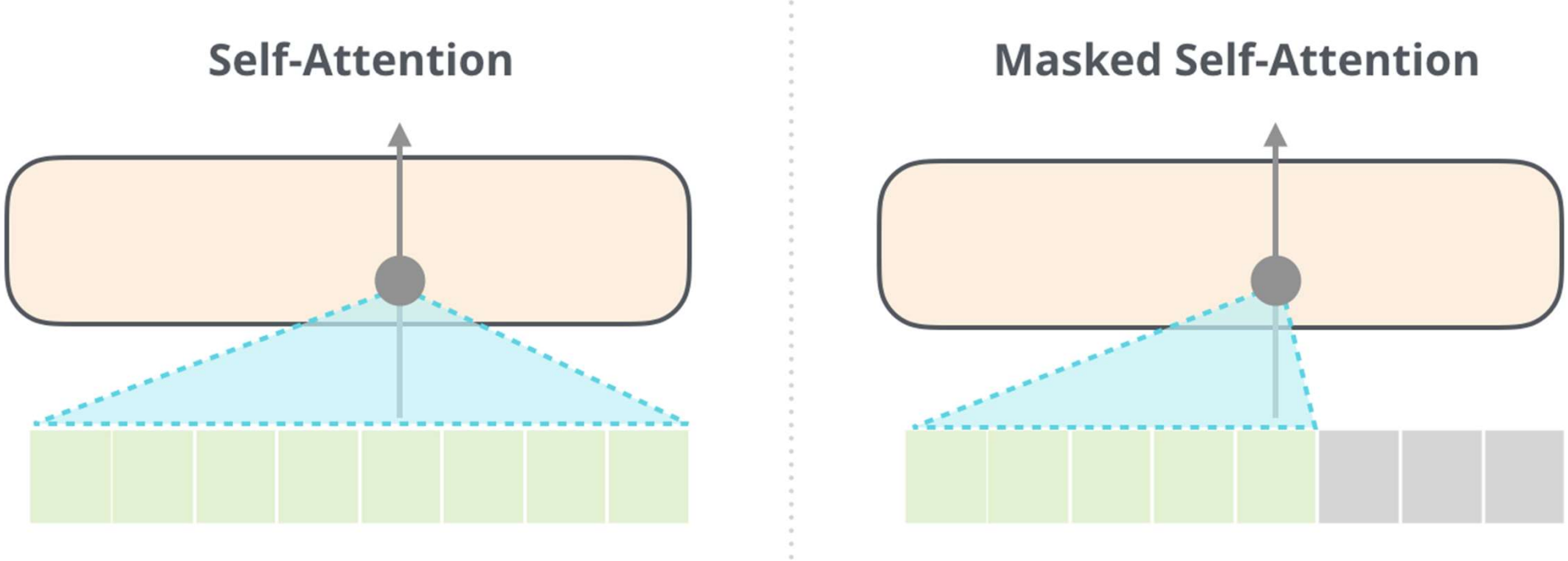
{‘l o w </w>’: 5, ‘l o w e r </w>’: 2, ‘n e w es t </w>’: 6, ‘w i d es t </w>’: 3 }

{‘l o w </w>’: 5, ‘l o w e r </w>’: 2, ‘n e w est </w>’: 6, ‘w i d est </w>’: 3 } (est freq. 9)

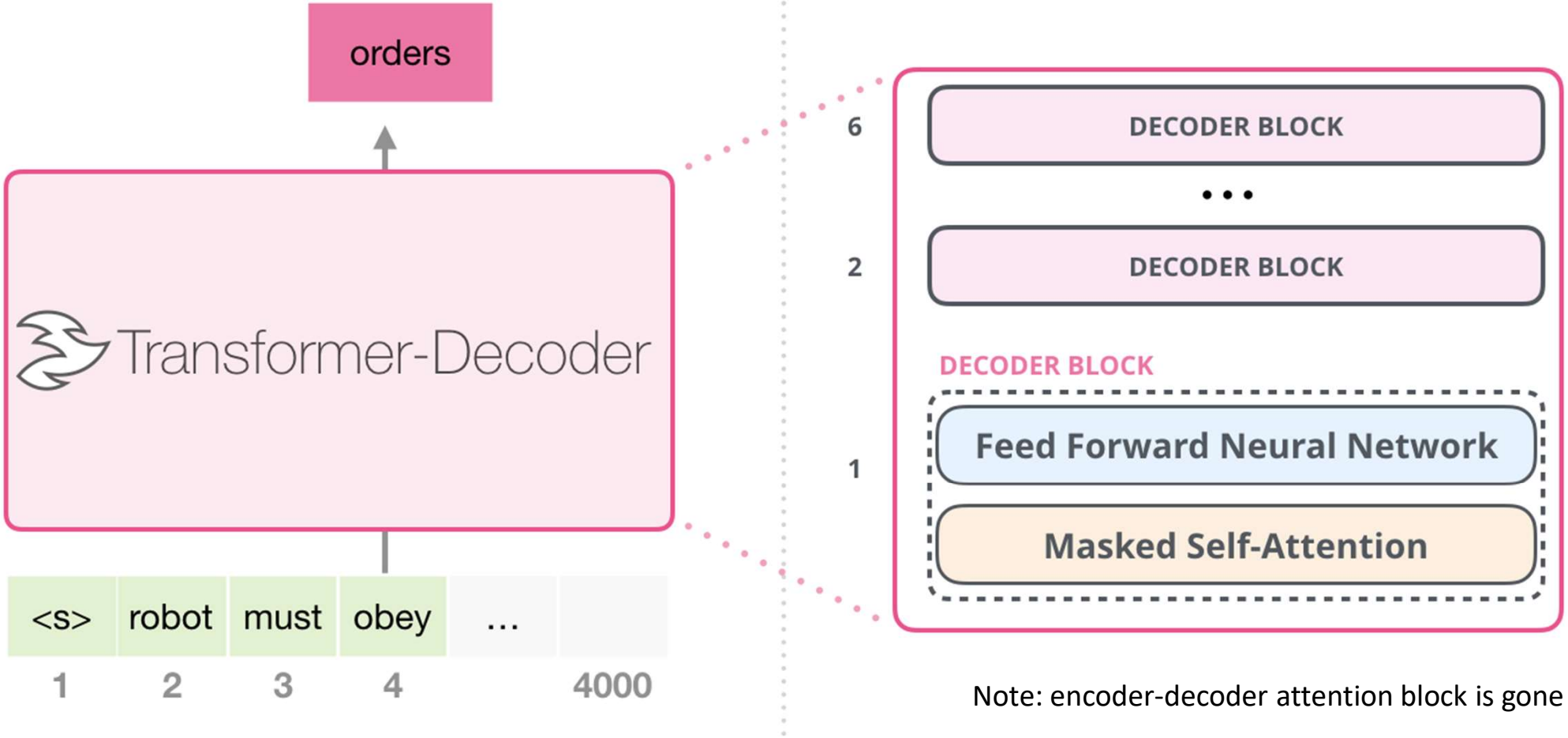
{‘lo w </w>’: 5, ‘lo w e r </w>’: 2, ‘n e w est</w>’: 6, ‘w i d est</w>’: 3 } (lo freq 7)

…..

Masked Self-Attention (to compute more efficiently)



Masked Self-Attention



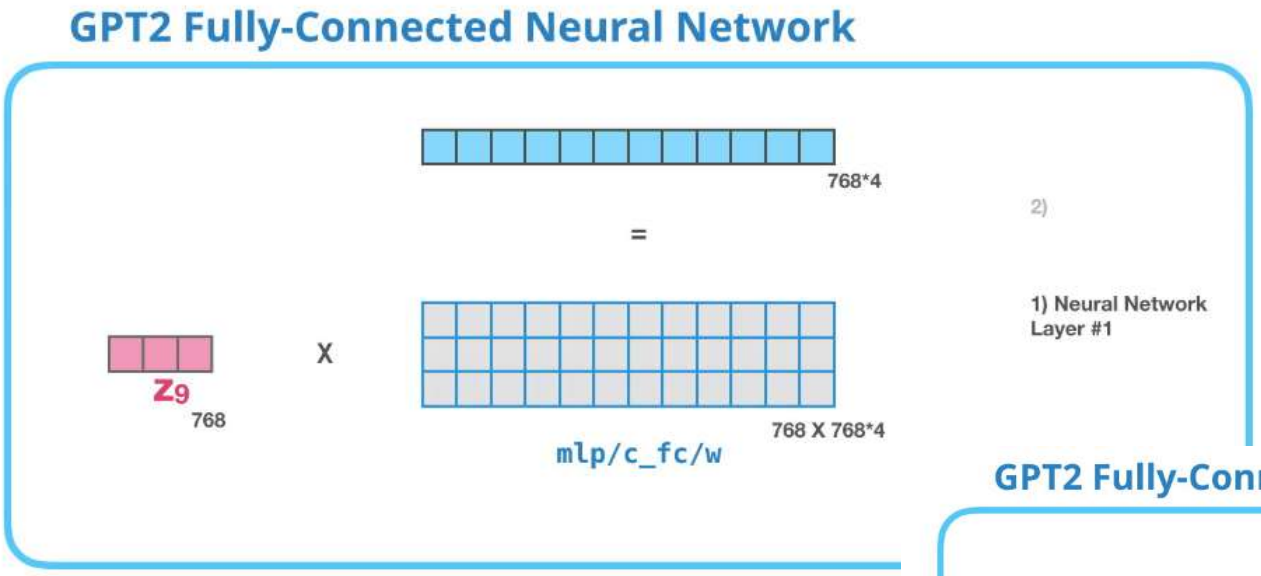
Note: encoder-decoder attention block is gone

Masked Self-Attention Calculation

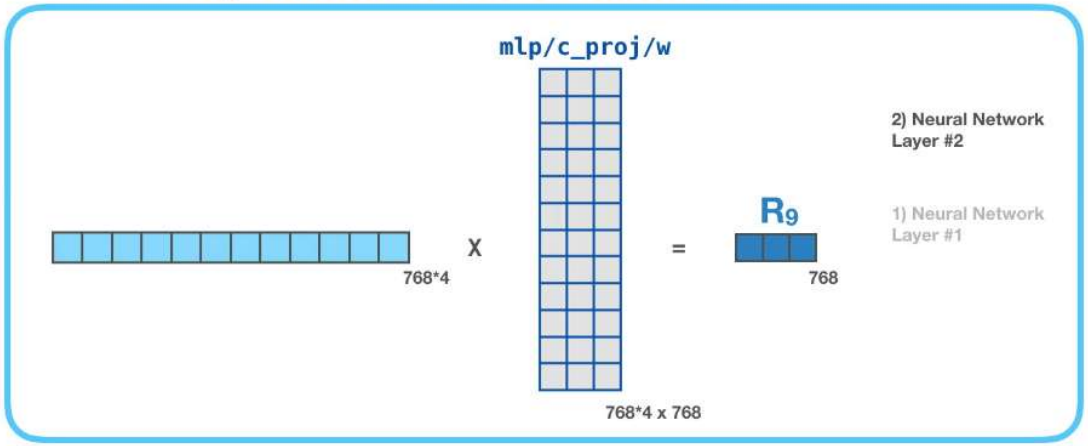
Re-use previous computation results: at any step, only need to results of q, k , v related to the new output word, no need to re-compute the others. Additional computation is linear, instead of quadratic.

Note: encoder-decoder attention block is gone

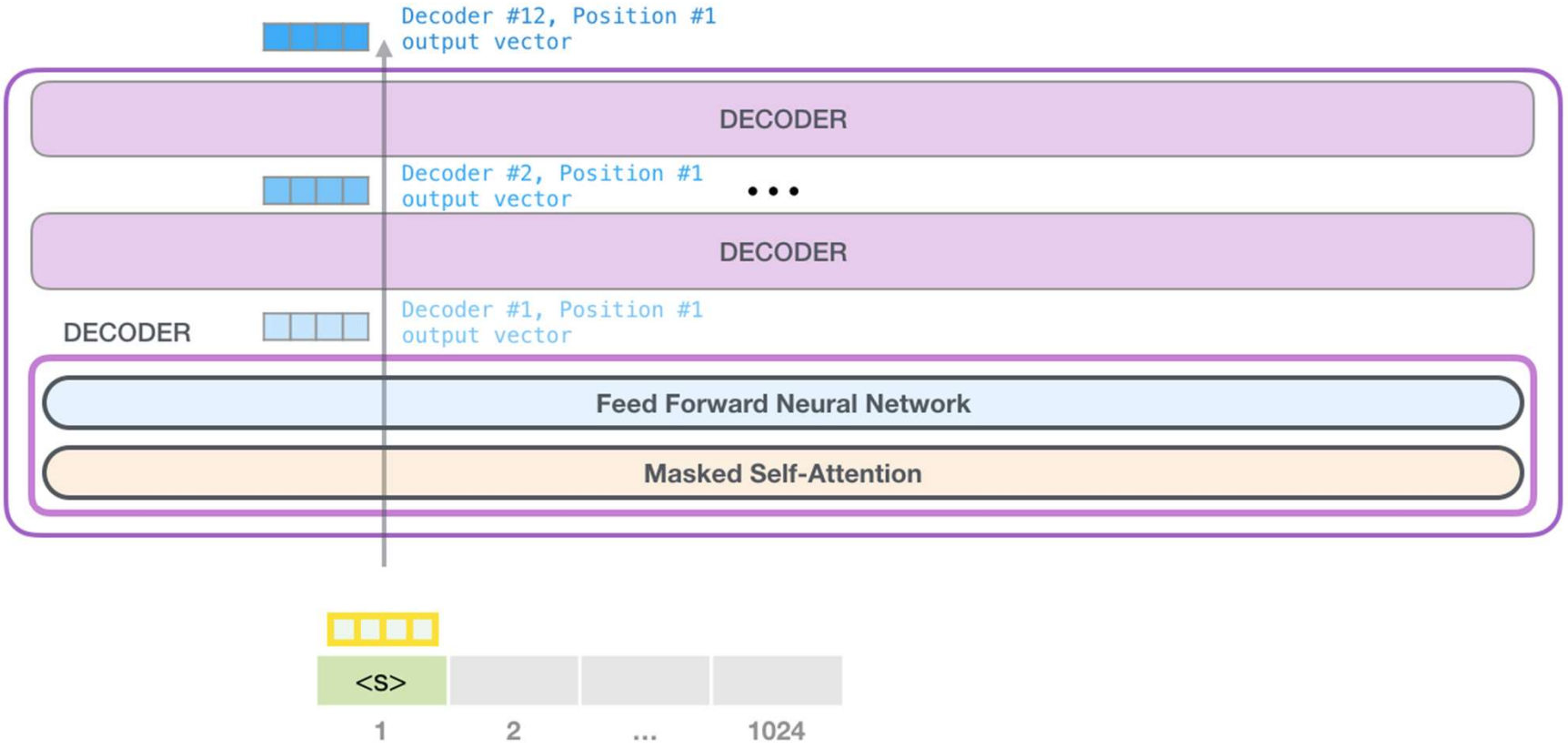
GPT-2 fully connected network has two layers (Example for GPT-2



768 is small model size



GPT-2 has a parameter top-k, so that we sample words from top k (highest probability from softmax) words for each output



This top-k parameter, if k=1, we would have output like:

The first time I saw the new version of the game, I was so excited. I was so excited to see the new version of the game, I was so excited to see the new version of the game, I was so excited to see the new version of the game, I was so excited to see the new version of the game, I was so excited to see the new version of the game, I was so excited to see the new version of the game, I was so excited to see the new version of the game, I was so excited to see the new version of the game, I was so excited to see the new version of the game, I was so excited to see the new version of the game, I was so excited to see the new version of the game, I was so excited to see the new version of the game, I was so excited to see the new version of the game, I was so excited to see the new version of the game,

GPT Training

GPT-2 uses unsupervised learning approach to training the language model.

There is no custom training for GPT-2, no separation of pre-training and fine-tuning like BERT.

A Story Generated by GPT-2

“The scientist named the population, after their distinctive horn, Ovid’s Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

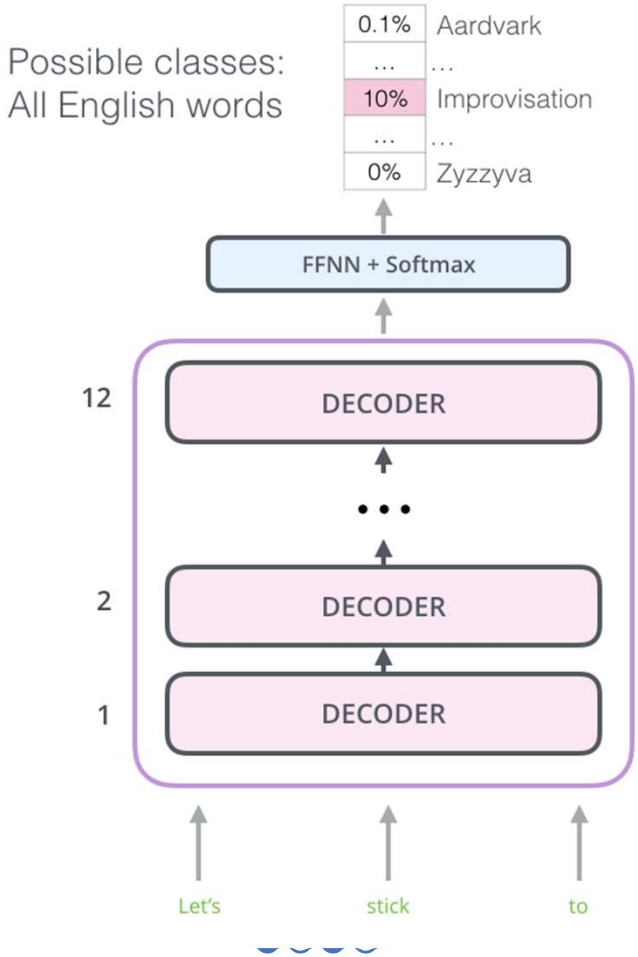
Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. `By the time we reached the top of one peak, the water looked blue, with some crystals on top,’ said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns."

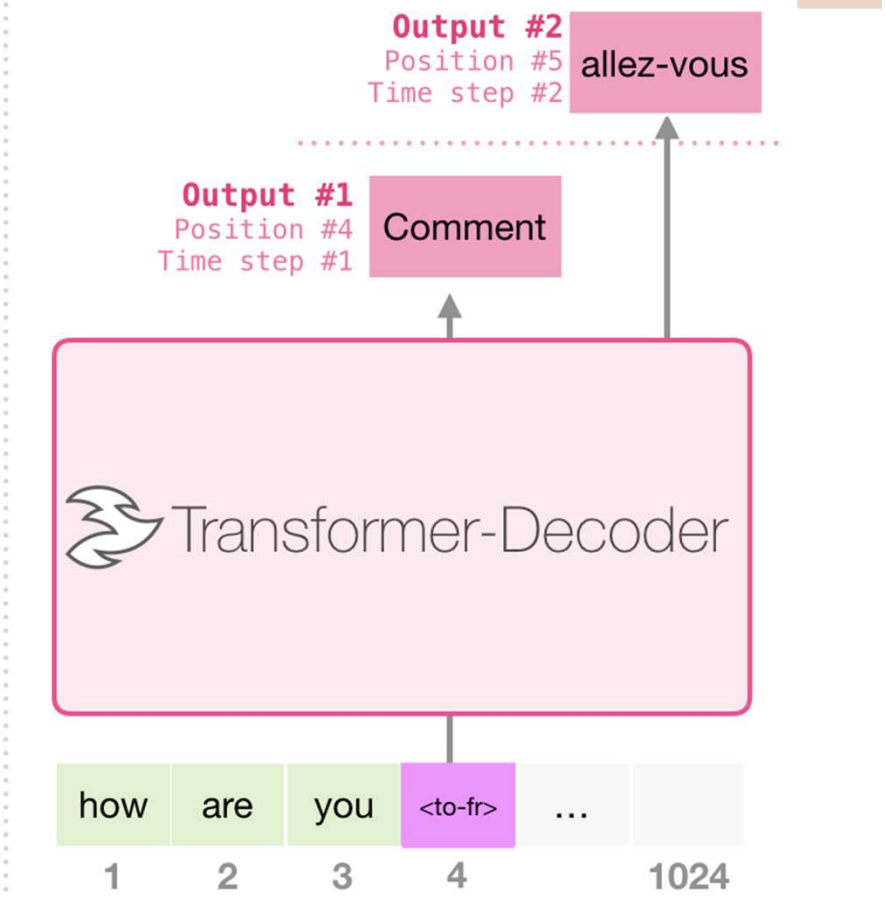
Transformer / GPT Prediction



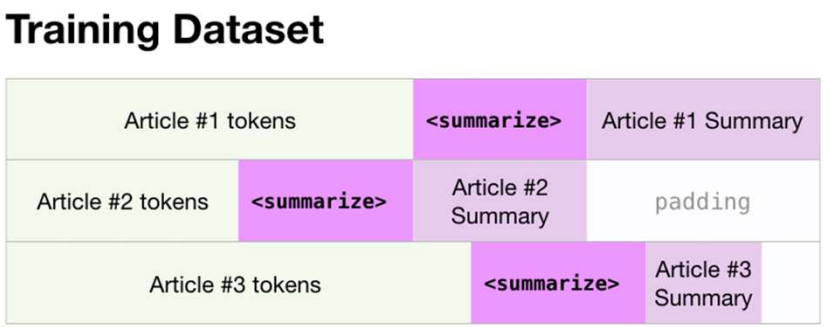
GPT-2 Application: Translation

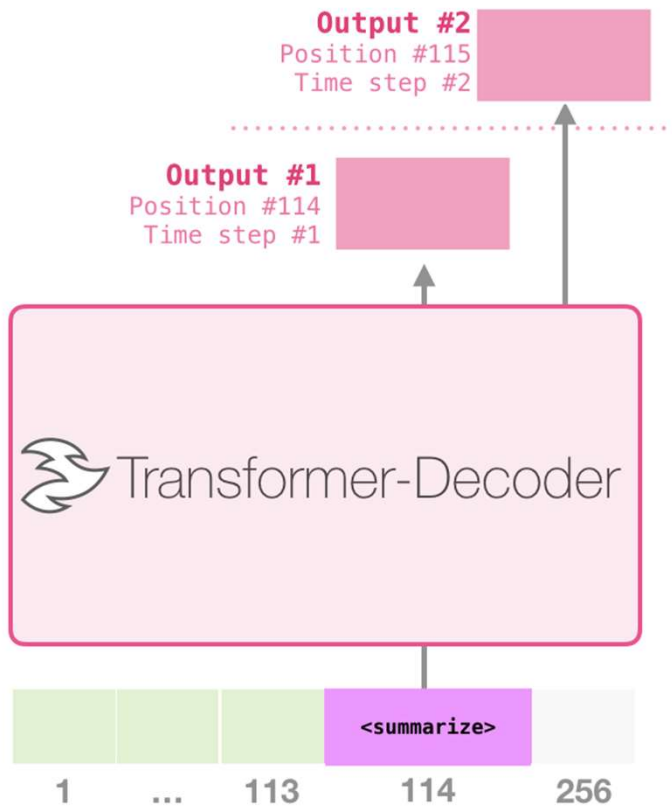
Training Dataset

I	am	a	student	<to-fr>	je	suis	étudiant
let	them	eat	cake	<to-fr>	Qu'ils	mangent	de
good	morning	<to-fr>	Bonjour				

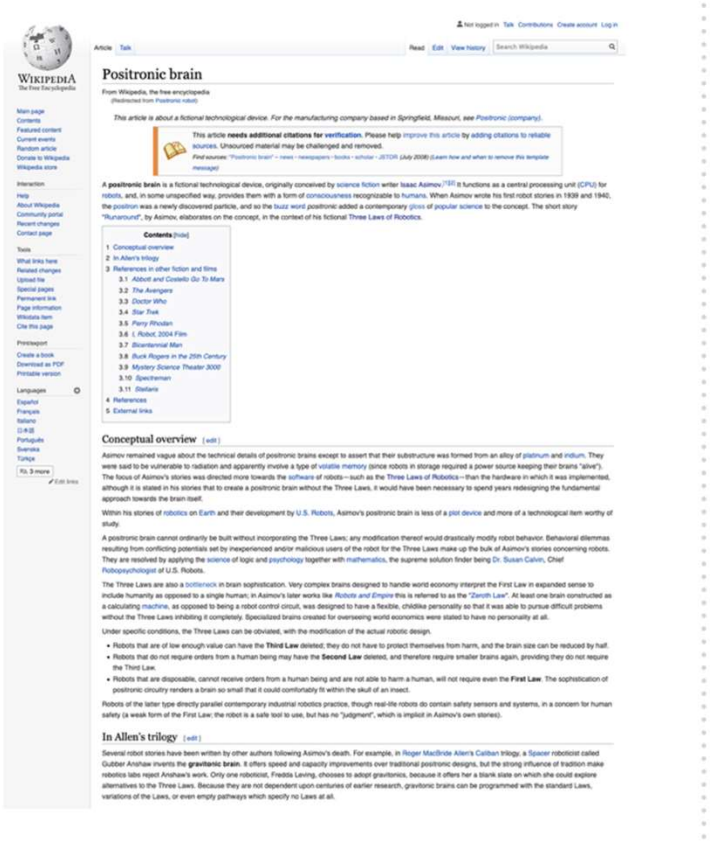


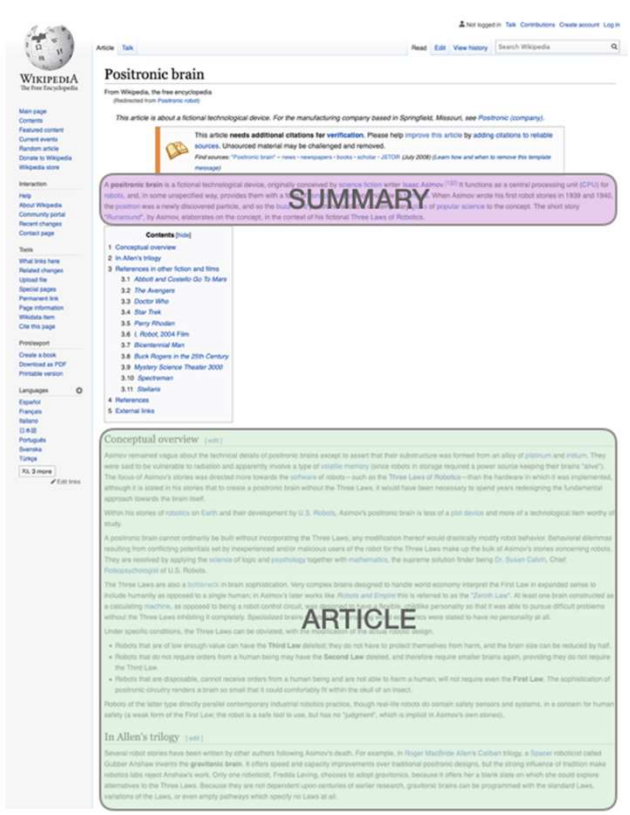
GPT-2 Application: Summarization





Using Wikipedia Data





What We have Learned

1. Model size matters (345 million parameters is better than 110 million parameters).

2. With enough training data, more training steps imply higher accuracy

3. Key innovation: Learning from unannotated data.

4. In biotechnology, we also have a lot of such data (for example meta-genomes).

Summary of Timed Data Modeling

1. DTW
2. LSTM
3. GRU
4. Attention
5. Transformer
6. BERT
7. GPT