Recurrent Neural Networks

Roberto Basili, Danilo Croce

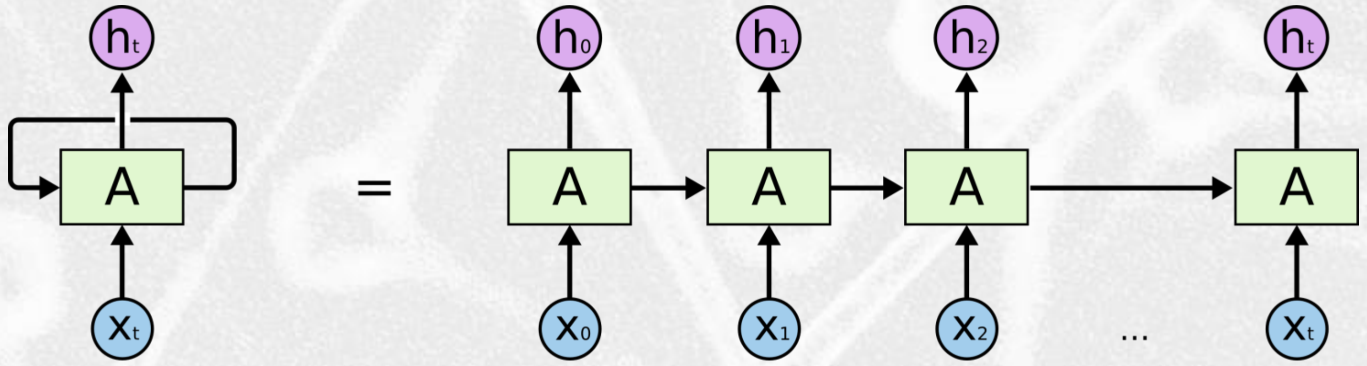
Machine Learning, Web Mining & Retrieval 2021/2022

Outline

- Recurrent and Recursive Networks
- Training Recurrent Networks
- LSTMS
- Applications to Language Processing
- Perspectives

Recurrent Neural Networks

- Used mainly to model sequences
  - naturally applied to textual and speech problems
- A representation at time step i is made dependent on the representations of the preceding steps
  - connections between units form a directed cycle



Recurrent Neural Networks

- Commons tasks are
  - ***language models***: predict the next word in a sentence given the already seen word
  - ***speech recognition***: predict a word given the current wave form and the preceding words
  - ***machine translation***: produce a sequence in a target language given an input sequence in a source language
- The most famous and effective model of RNNs are the Long-Short Term Memory (LSTM) Networks (Sepp Hochreiter and Jürgen Schmidhuber, 1997)
  - they are meant to better deal with long-range dependencies

Neural Networks for Natural Language Processing

- Linguistic features have been highly enriched since NN language models have been introduced
  - Words, *n*-grams as well as sentences, paragraphs have been modeled through efficient and highly robust neural learners
- Representation are usually dense Embeddings
- Making explicit Use of the contexts: Recurrent Networks
- Beyond Classification: Transducing, Ranking, Encoding, Decoding

Recurrent Neural Networks

- For example, consider the classifcal form of a dynamical system

$$s^{(t)} = f(s^{(t-1)}; \theta),$$

- Its corresponding unfolded computational graph is as follows

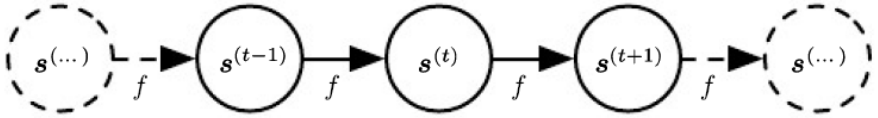
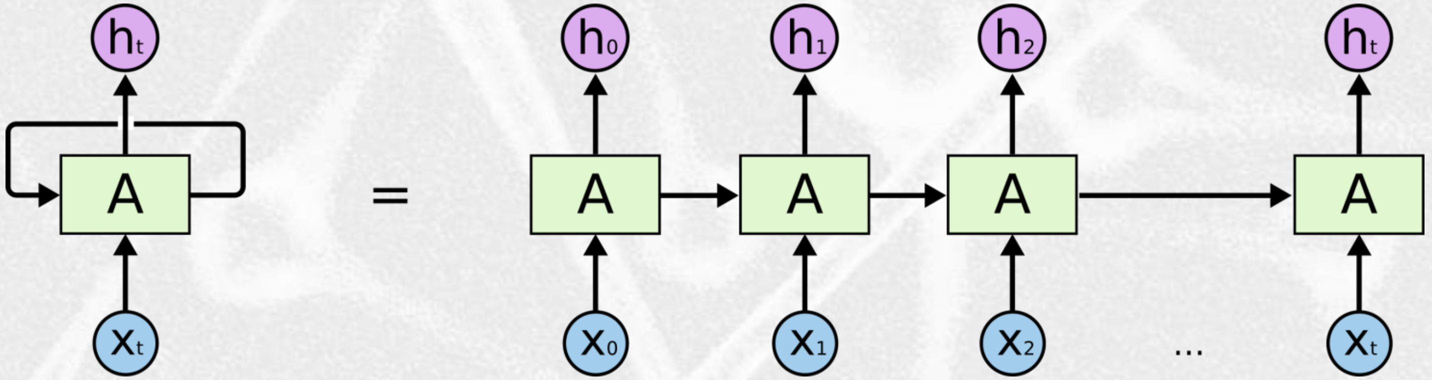


Figure 10.1: The classical dynamical system described by equation 10.1, illustrated as an unfolded computational graph. Each node represents the state at some time t, and the function f maps the state at t to the state at $t+1$. The same parameters (the same value of θ used to parametrize f) are used for all time steps.



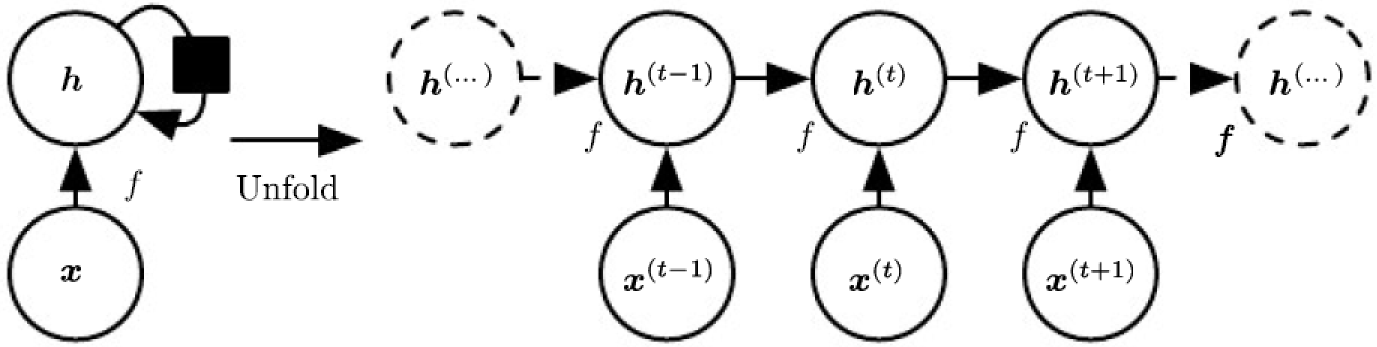
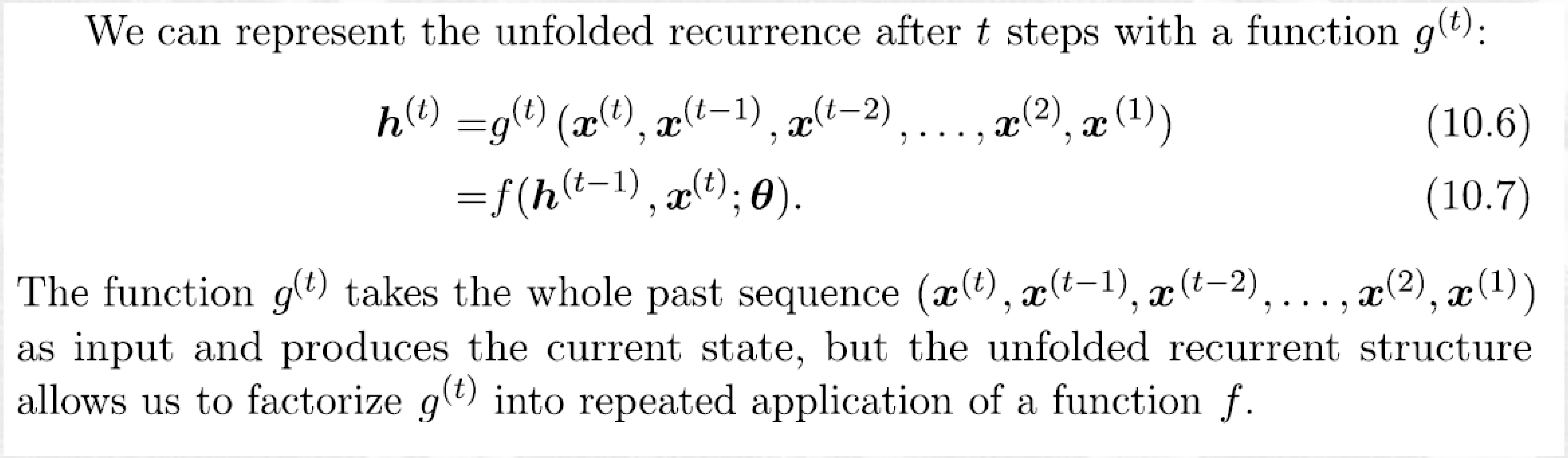


Figure 10.2: A recurrent network with no outputs. This recurrent network just processes information from the input $\boldsymbol{x}$ by incorporating it into the state $\boldsymbol{h}$ that is passed forward through time. *(Left)* Circuit diagram. The black square indicates a delay of a single time step. *(Right)* The same network seen as an unfolded computational graph, where each node is now associated with one particular time instance.

Many recurrent neural networks use equation 10.5 or a similar equation to define the values of their hidden units. To indicate that the state is the hidden units of the network, we now rewrite equation 10.4 using the variable $\boldsymbol{h}$ to represent the state,

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}), \tag{10.5}$$

illustrated in figure 10.2; typical RNNs will add extra architectural features such as output layers that read information out of the state $\boldsymbol{h}$ to make predictions.



We can represent the unfolded recurrence after t steps with a function $g^{(t)}$:

$$\boldsymbol{h}^{(t)} = g^{(t)}(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}) \tag{10.6}$$

$$= f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}). \tag{10.7}$$

The function $g^{(t)}$ takes the whole past sequence $(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ as input and produces the current state, but the unfolded recurrent structure allows us to factorize $g^{(t)}$ into repeated application of a function f.

Using a RNN

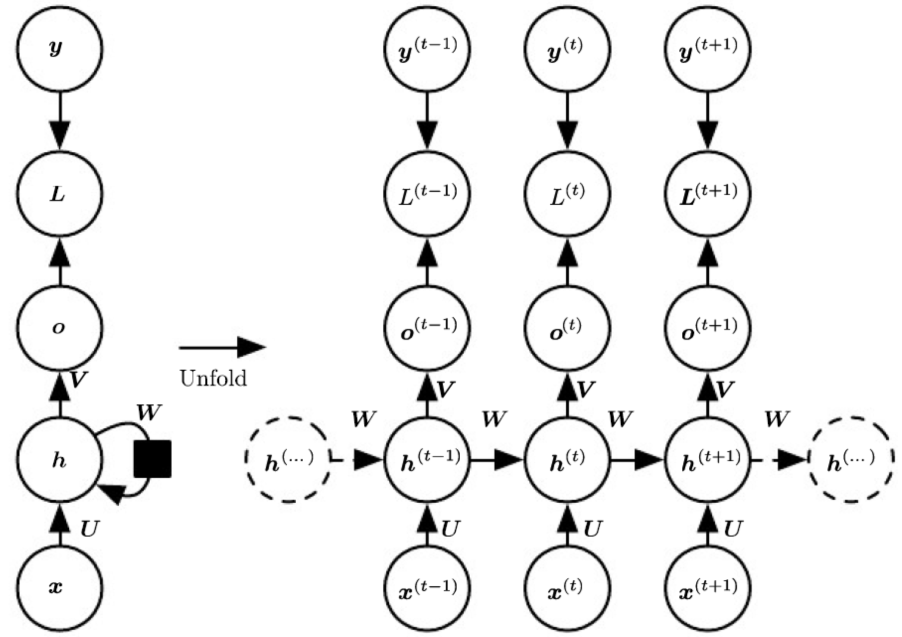
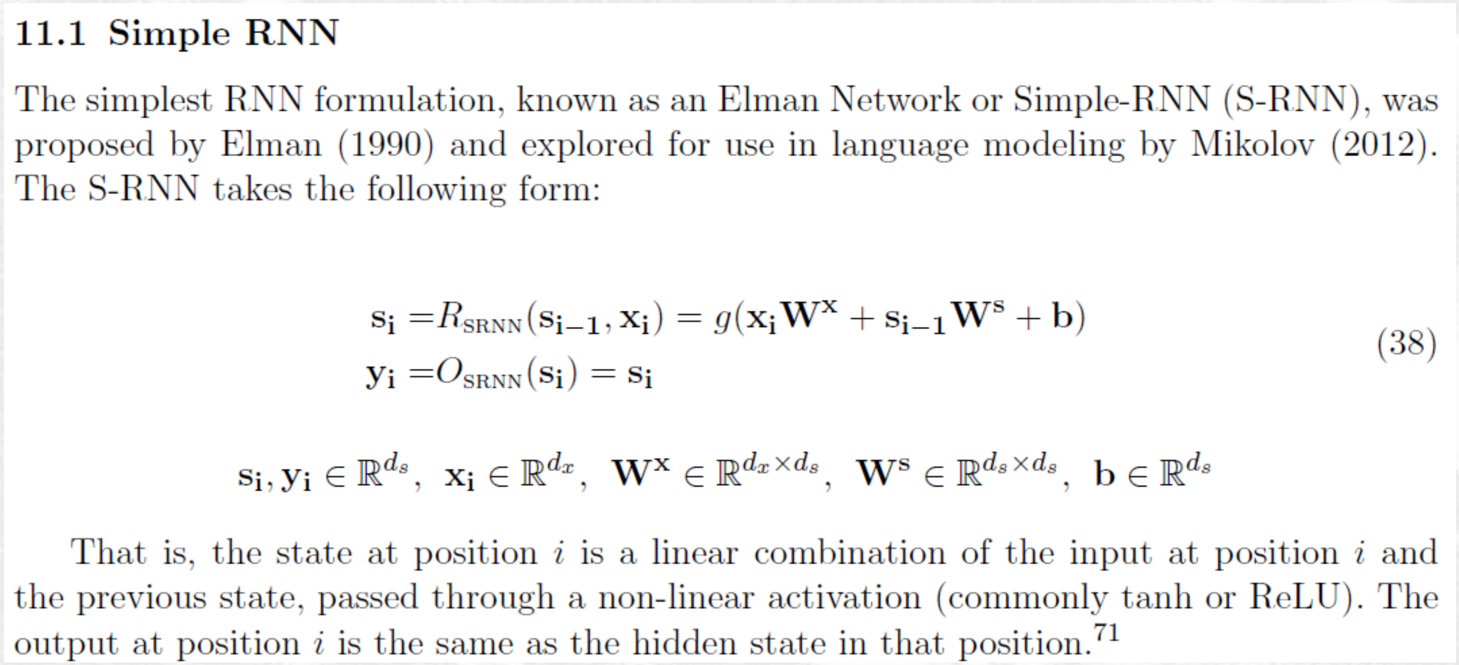


Figure 10.3: The computational graph to compute the training loss of a recurrent network that maps an input sequence of $\boldsymbol{x}$ values to a corresponding sequence of output $\boldsymbol{o}$ values. A loss L measures how far each $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. When using softmax outputs, we assume $\boldsymbol{o}$ is the unnormalized log probabilities. The loss L internally computes $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{o})$ and compares this to the target $\boldsymbol{y}$. The RNN has input to hidden connections parametrized by a weight matrix $\boldsymbol{U}$, hidden-to-hidden recurrent connections parametrized by a weight matrix $\boldsymbol{W}$, and hidden-to-output connections parametrized by a weight matrix $\boldsymbol{V}$. Equation 10.8 defines forward propagation in this model. *(Left)* The RNN and its loss drawn with recurrent connections. *(Right)* The same seen as a time-unfolded computational graph, where each node is now associated with one particular time instance.

Simple RNN



11.1 Simple RNN

The simplest RNN formulation, known as an Elman Network or Simple-RNN (S-RNN), was proposed by Elman (1990) and explored for use in language modeling by Mikolov (2012). The S-RNN takes the following form:

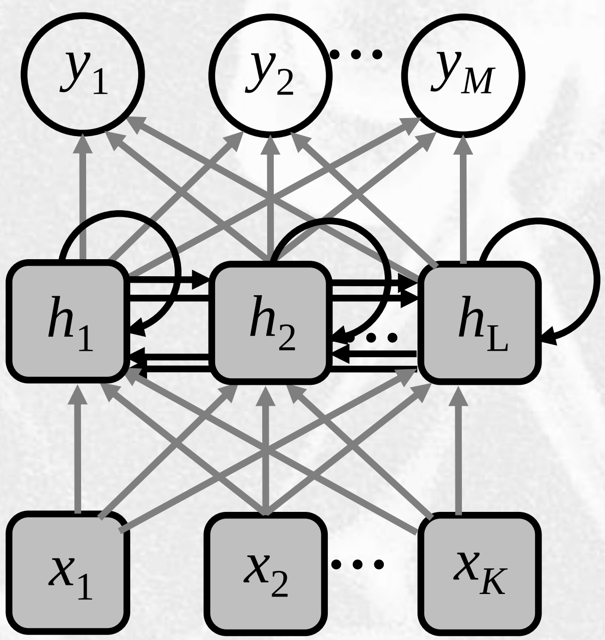
$$\begin{aligned} \mathbf{s_i} &= R_{\text{SRNN}}(\mathbf{s_{i-1}}, \mathbf{x_i}) = g(\mathbf{x_i}\mathbf{W^x} + \mathbf{s_{i-1}}\mathbf{W^s} + \mathbf{b}) \\ \mathbf{y_i} &= O_{\text{SRNN}}(\mathbf{s_i}) = \mathbf{s_i} \end{aligned} \tag{38}$$

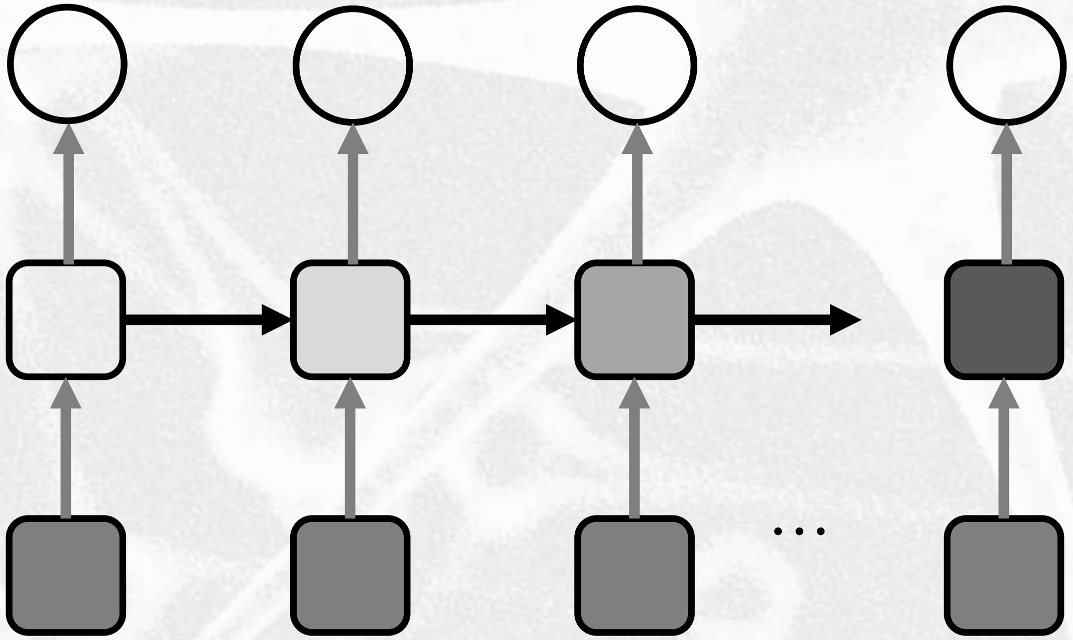
$$\mathbf{s_i}, \mathbf{y_i} \in \mathbb{R}^{d_s}, \;\; \mathbf{x_i} \in \mathbb{R}^{d_x}, \;\; \mathbf{W^x} \in \mathbb{R}^{d_x \times d_s}, \;\; \mathbf{W^s} \in \mathbb{R}^{d_s \times d_s}, \;\; \mathbf{b} \in \mathbb{R}^{d_s}$$

That is, the state at position i is a linear combination of the input at position i and the previous state, passed through a non-linear activation (commonly tanh or ReLU). The output at position i is the same as the hidden state in that position.[71]

Recurrent neural networks (RNNs)

- An RNN can be unwrapped and implemented using the same weights and biases at each step to link units over time as shown below
- The resulting unwrapped RNN is similar to a hidden Markov model, but keep in mind that the hidden units in RNNs are not stochastic





Recurrent neural networks (RNNs)

- Recurrent neural networks apply linear matrix operations to the current observation and the hidden units from the previous time step, and the resulting linear terms serve as arguments of activation functions act():

$$\mathbf{h}_t = \mathrm{g}(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\mathbf{o}_t = \mathrm{f}(\mathbf{W}_o \mathbf{h}_t + \mathbf{b}_o)$$

- The same matrix $\mathbf{U}_h$ is used at each time step
- The hidden units in the previous step $\mathbf{h}_{t-1}$ influence the computation of $\mathbf{h}_t$ where the current observation $\mathbf{x}_t$ contributes to a $\mathbf{W}_h\mathbf{x}_t$ term that is combined with $\mathbf{U}_h\mathbf{h}_{t-1}$ and bias $\mathbf{b}_h$ terms
- Both $\mathbf{W}_h$ and $\mathbf{b}_h$ are typically replicated over time
- The output layer is modeled by a classical neural network activation function applied to a linear transformation of the hidden units, the operation is replicated at each step.

BPPTT

- For training a recurrent network, a solution is to unfold the recurrent structure and expand it as a feedforward neural network with a certain number of time steps: then apply traditional backpropagation onto this unfolded neural network.
- This solution is known as **Backpropagation through Time** (BPTT), independently invented by several researchers including (Robinson and Fallside, 1987; Werbos, 1988; Mozer, 1989)

The loss, exploding and vanishing gradients

- The loss for a particular sequence in the training data can be computed either at each time step or just once, at the end of the sequence.
- In either case, predictions will be made after many processing steps and this brings us to an important problem.
- The gradient for feedforward networks decomposes the gradient of parameters at layer *l* into a term that involves the product of matrix multiplications of the form $\delta^{(l)}\mathbf{W}^{T(l+1)}$ (remind lessons on backpropagation in feedforward network)
- A recurrent network uses the same matrix at each time step, and over many steps the gradient can very easily either diminish to zero or explode to infinity—just as the magnitude of any number other than one taken to a large power either approaches zero or increases indefinitely

BPTT: the algorithm

1. Present a sequence of *k1* timesteps of input and output pairs to the network.
2. Unroll the network then calculate and accumulate errors across *k2* timesteps.
3. Roll-up the network and update weights.
4. Repeat

- The TBPTT algorithm requires the consideration of two parameters:
  - **k1**: The *number of forward-pass timesteps between updates*.
    - this influences how slow or fast training will be, given how often weight updates are performed.
  - **k2**: The *number of timesteps to which to apply BPTT*.
    - it should be large enough to capture the temporal structure in the problem for the network to learn.
    - Too large a value results in vanishing gradients

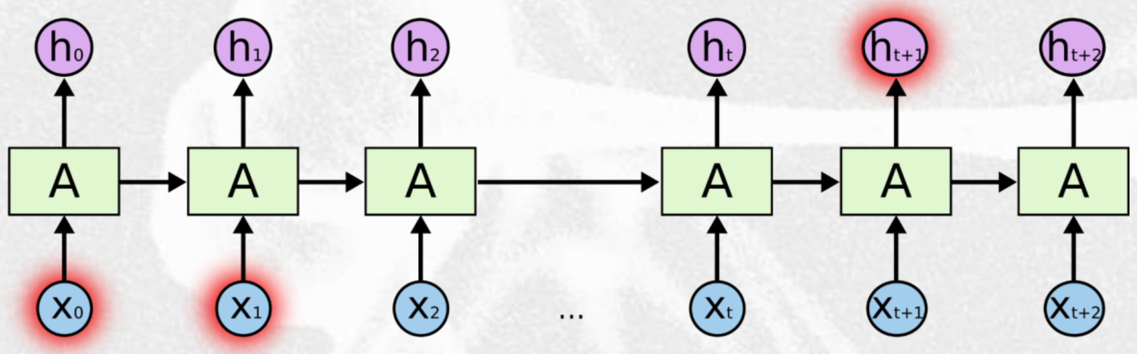
Dealing with exploding gradients

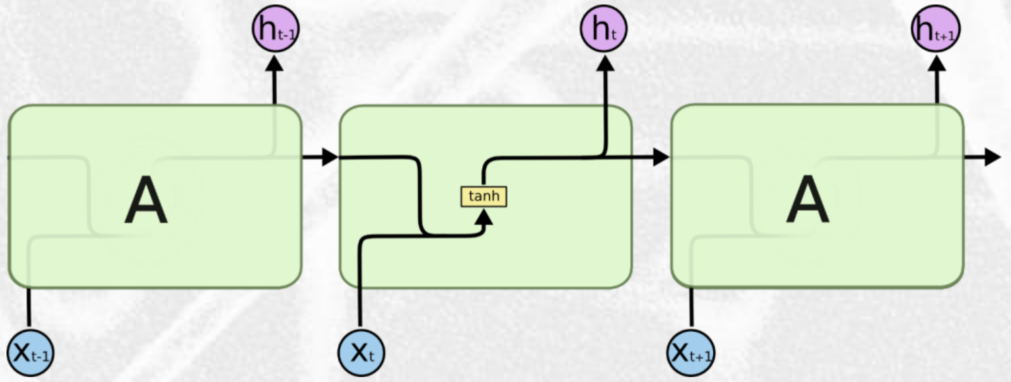
- The use of L1 or L2 regularization can mitigate the problem of exploding gradients by encouraging weights to be small.
- Another strategy is to simply detect if the norm of the gradient exceeds some threshold, and if so, scale it down.
- This is sometimes called ***gradient (norm) clipping*** where for a gradient vector *g* and threshold *T*,

$$\text{if} \quad \|\mathbf{g}\| \geq T \quad \text{then} \quad \mathbf{g} \leftarrow \frac{T}{\|\mathbf{g}\|}\mathbf{g}$$

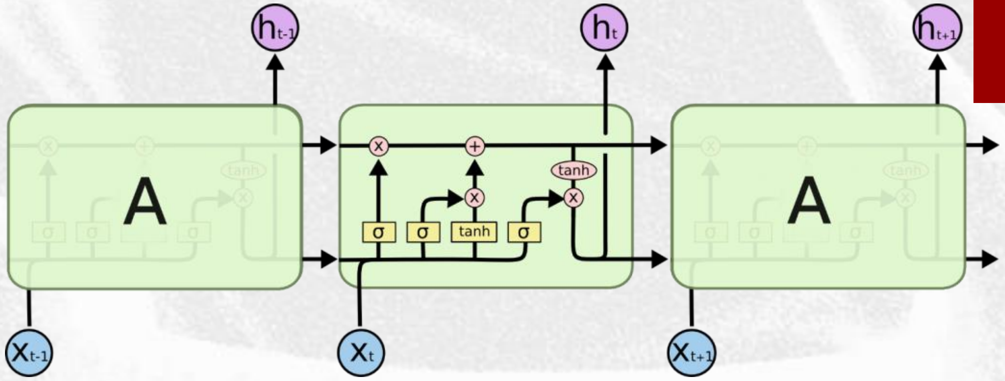
  - where T is a hyperparameter, which can be set to the average norm over several previous updates where clipping was not used.

Long-term Dependencies with one single layer



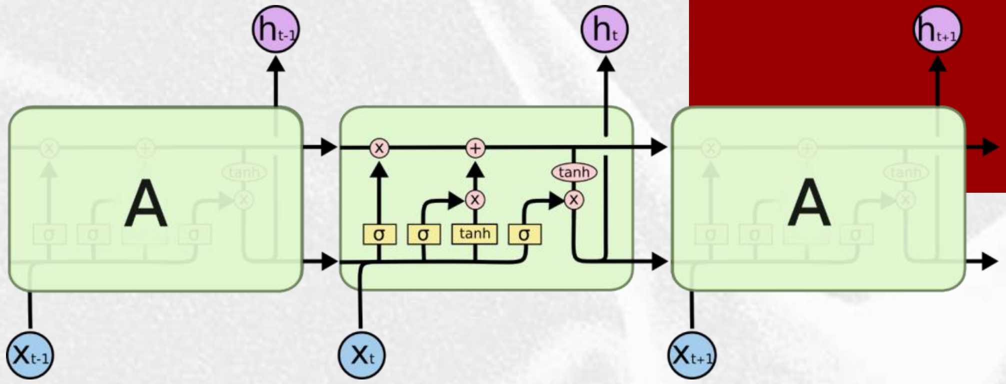


LSTMS (Hochreiter & Schmidhuber, 1997)



- The **Long Short-Term Memory (LSTM)** architecture (Hochreiter & Schmidhuber, 1997) was designed to solve the vanishing gradients problem.
- Main idea: to introduce as part of the state representation also specialized **memory cells** (a vector C) that can preserve gradients across time.
- Access to the memory cells is controlled by gating components, i.e. smooth mathematical functions that simulate logical gates.

LSTMS

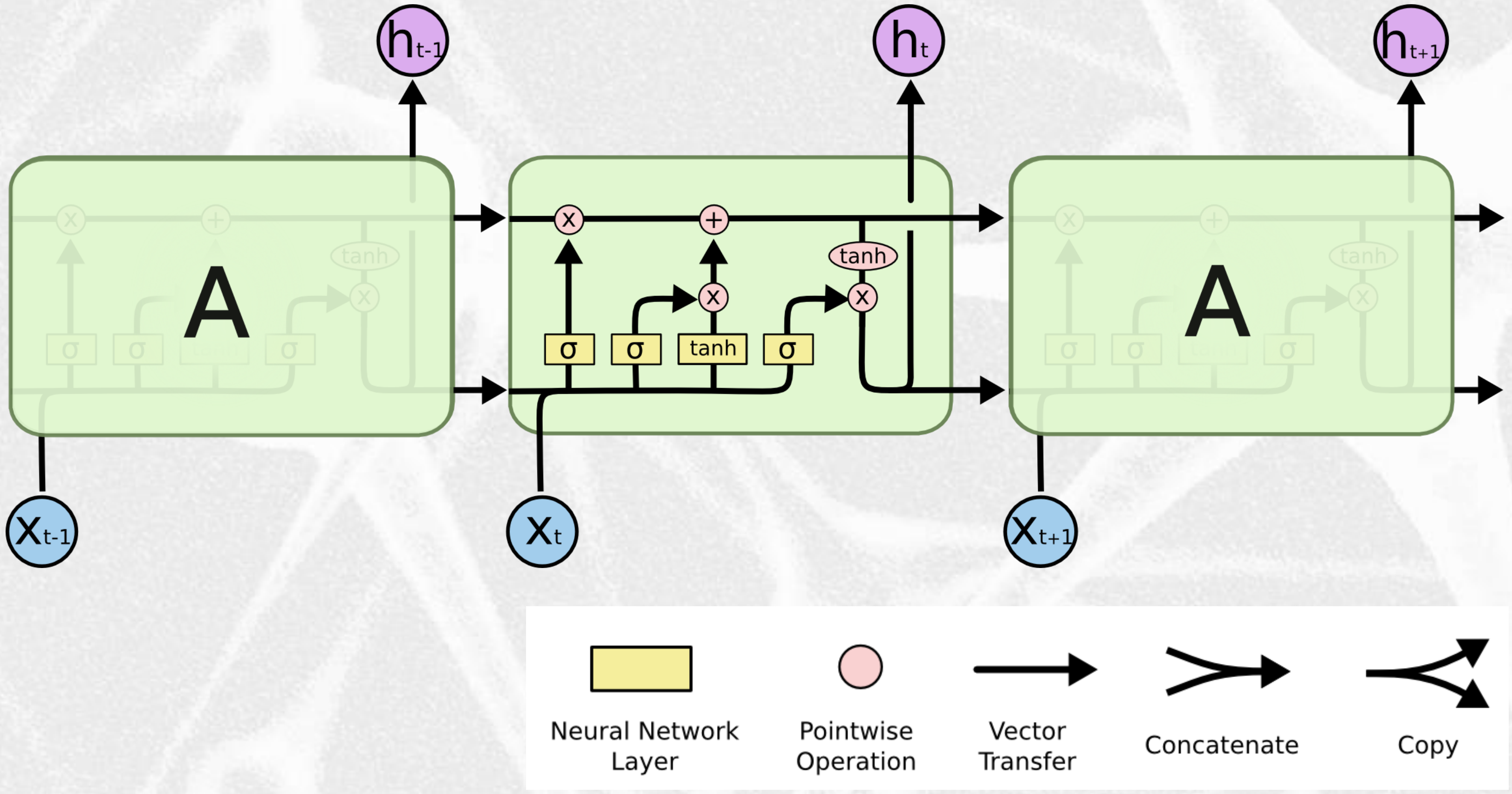


- At each input state, a gate is used to decide:
  - **how much of the new input** should be written to the memory cell,
  - **how much of the current content** of the memory cell should be **forgotten**.

- Concretely, a gate g in $[0;1]^n$ is a vector of values in the range [0; 1] that is multiplied **component-wise** with another vector C in R^n, and the result is then added to another vector.

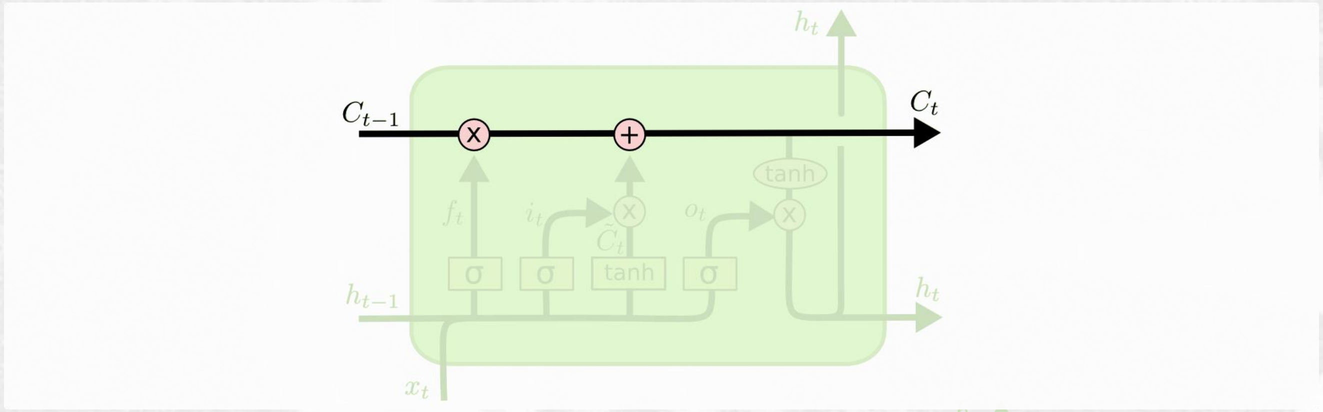
- Indices in C corresponding to near-one values in *g* are allowed to pass, while those corresponding to near-zero values are blocked.

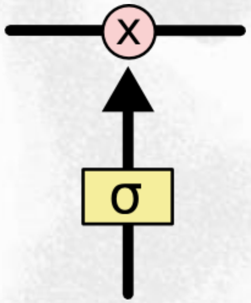
4 layer RNNS



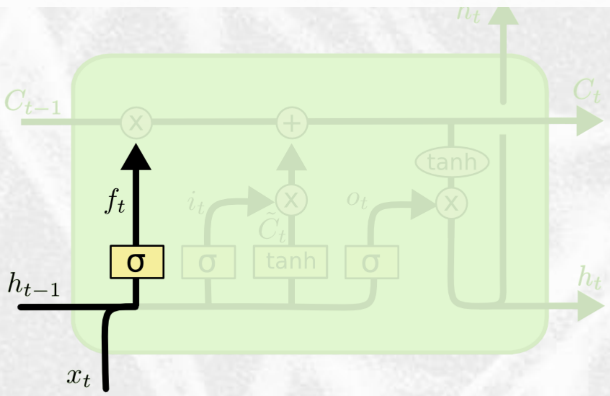
h_{t-1}
h_t
h_{t+1}
A
A
x
+
tanh
x
x
σ
σ
tanh
σ
X_{t-1}
X_t
X_{t+1}
Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy

… The memory component and the gates



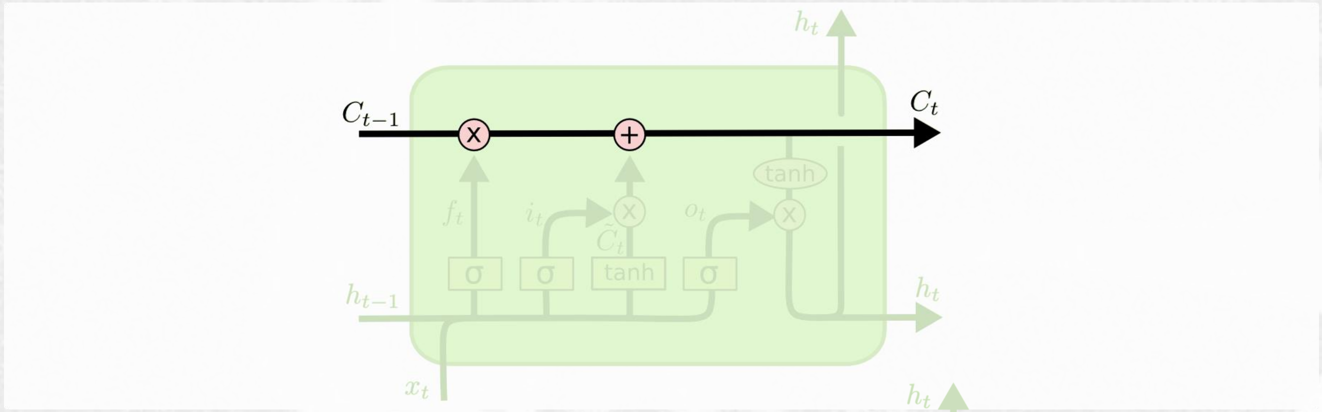


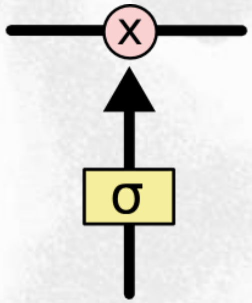
- The FORGET gate

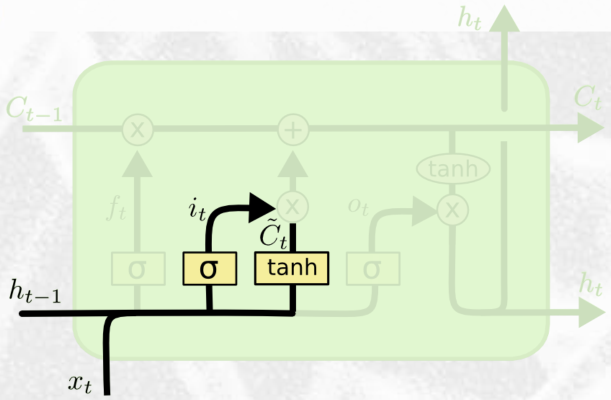


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

… The memory component and the gates



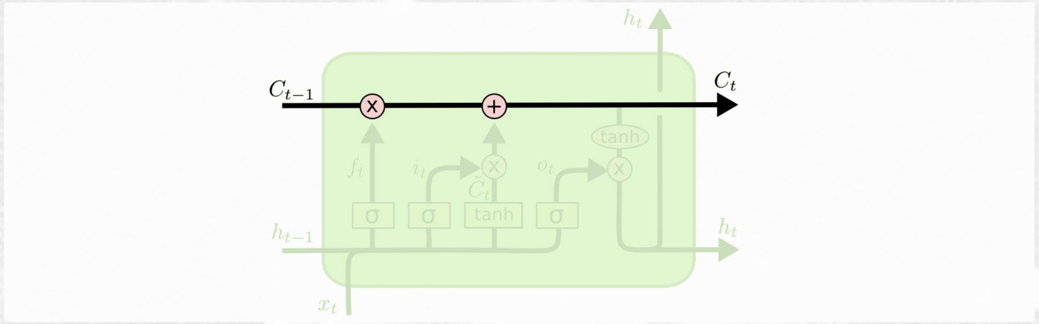


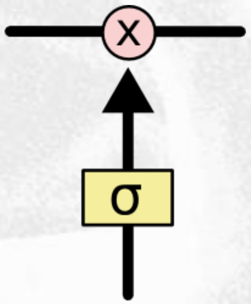


- The INPUT gate

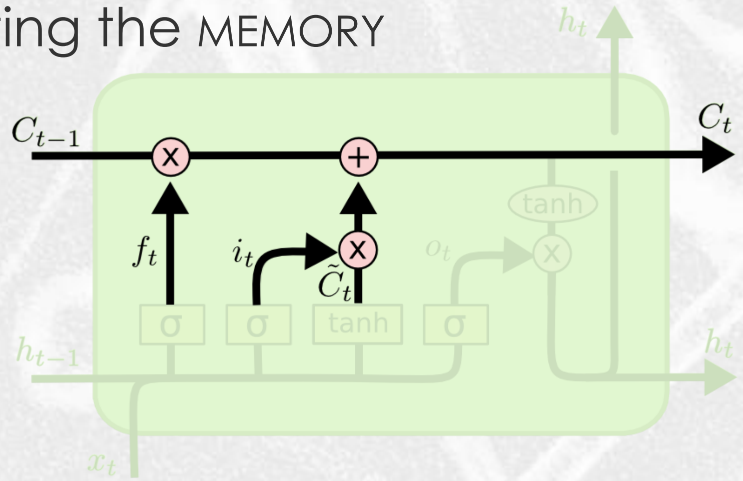
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

… The memory component and the gates



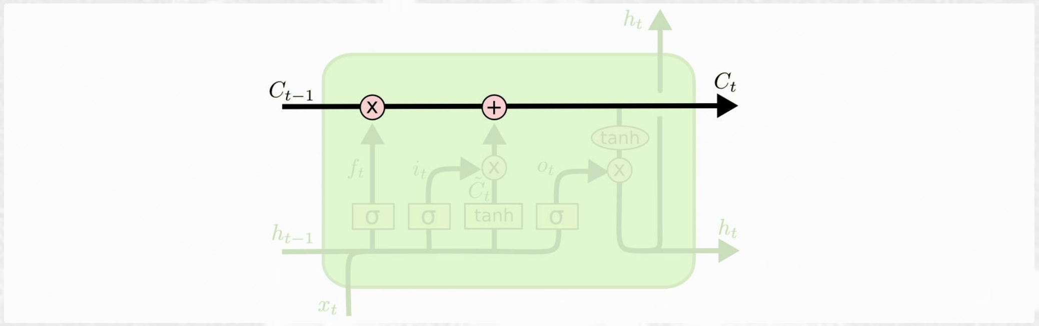


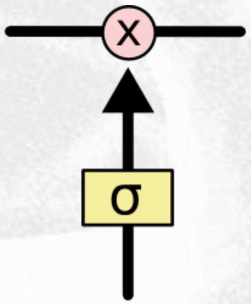
- Updating the MEMORY



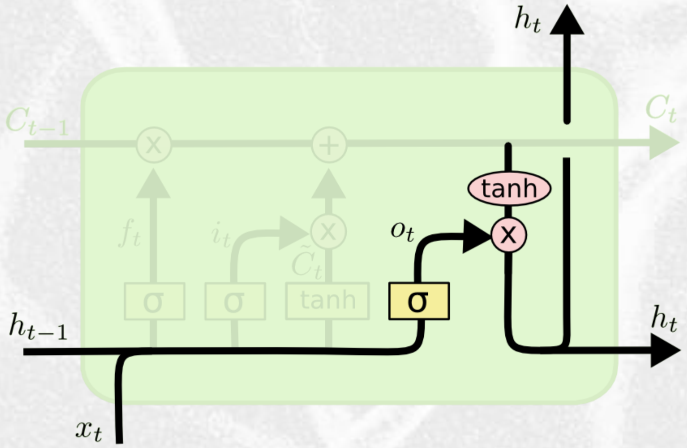
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

… The memory component and the gates



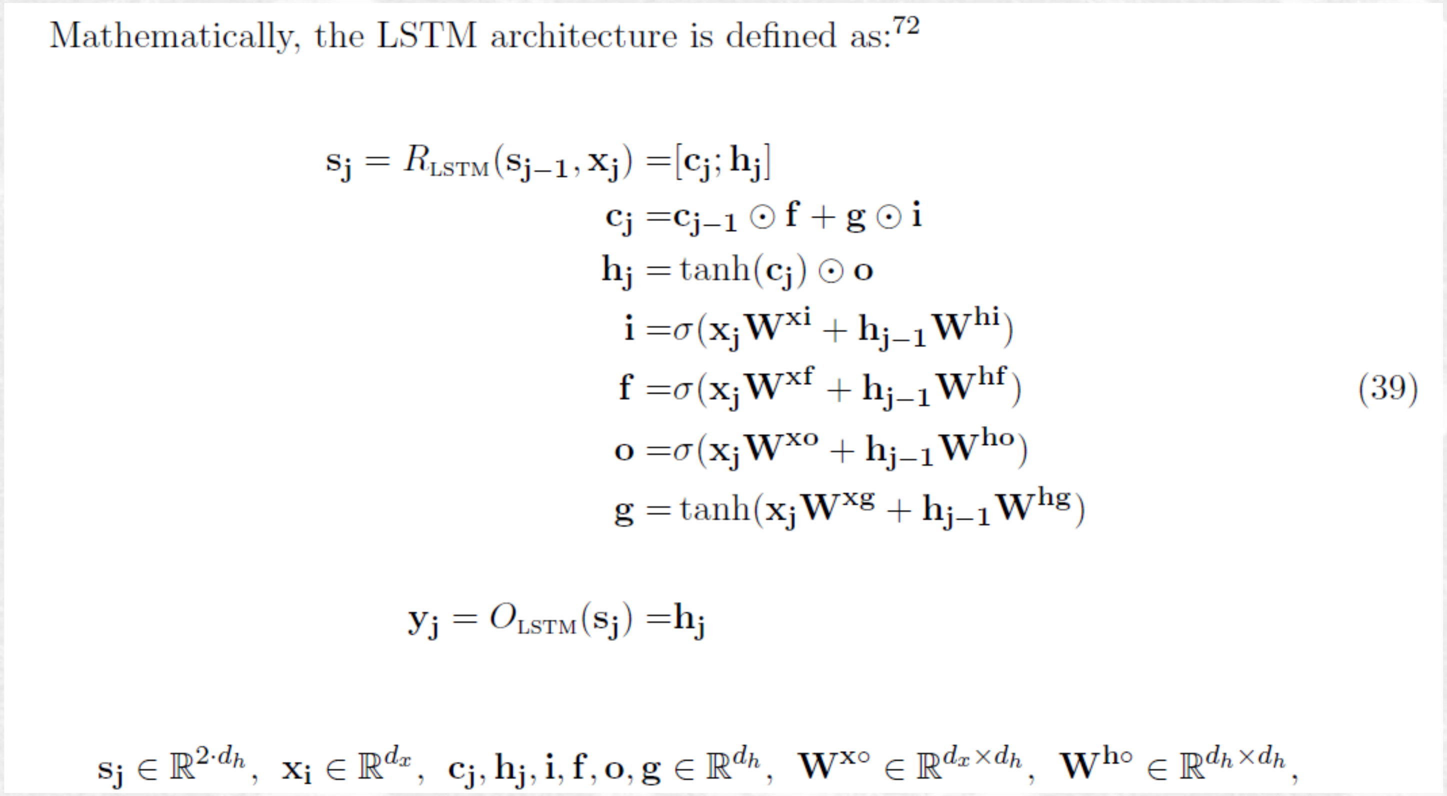


- Computing the OUTPUT



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

LSTMS



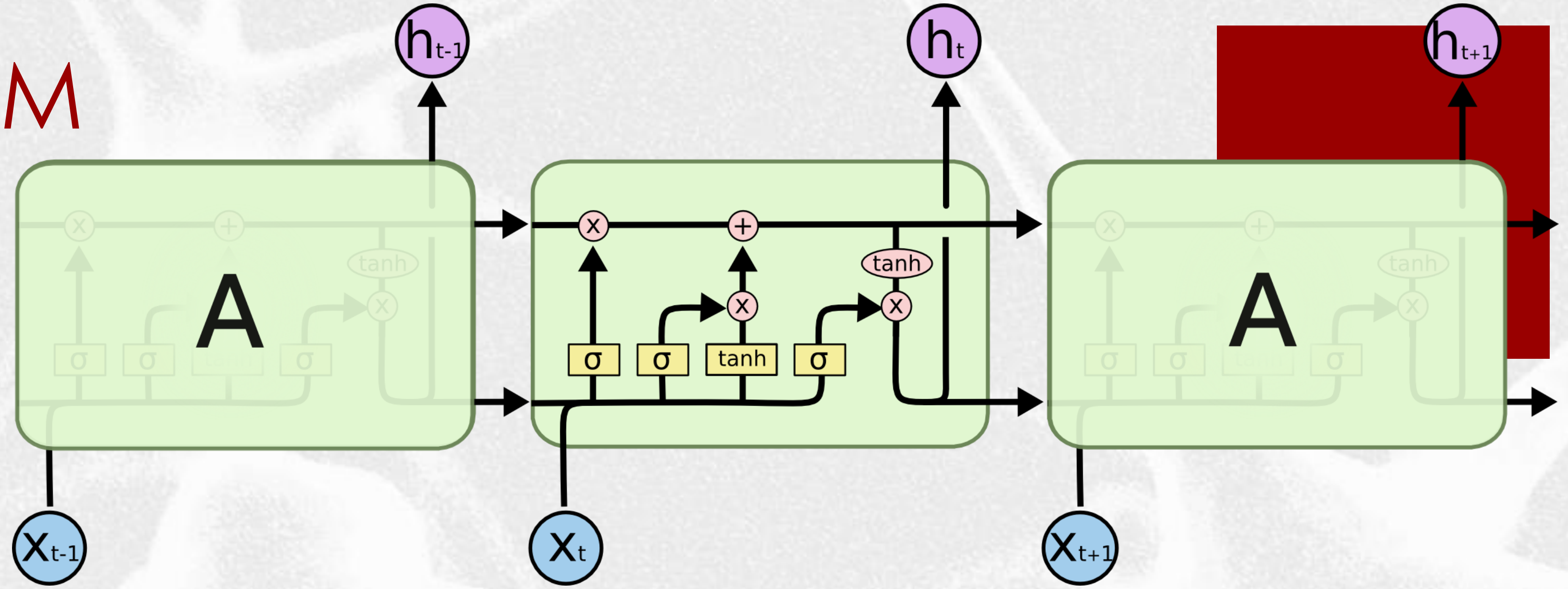
Mathematically, the LSTM architecture is defined as:[72]

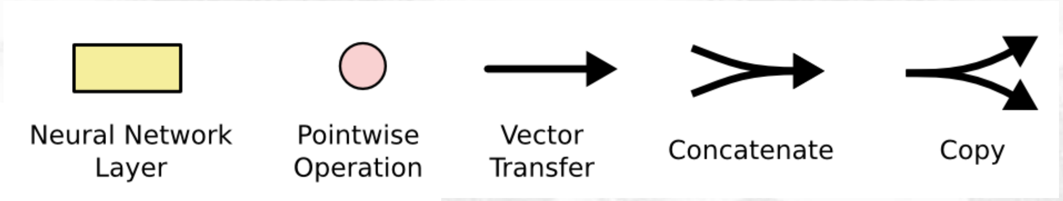
$$
\begin{aligned}
\mathbf{s_j} = R_{\text{LSTM}}(\mathbf{s_{j-1}}, \mathbf{x_j}) =& [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} =& \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} =& \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{i} =& \sigma(\mathbf{x_j}\mathbf{W^{xi}} + \mathbf{h_{j-1}}\mathbf{W^{hi}}) \\
\mathbf{f} =& \sigma(\mathbf{x_j}\mathbf{W^{xf}} + \mathbf{h_{j-1}}\mathbf{W^{hf}}) \\
\mathbf{o} =& \sigma(\mathbf{x_j}\mathbf{W^{xo}} + \mathbf{h_{j-1}}\mathbf{W^{ho}}) \\
\mathbf{g} =& \tanh(\mathbf{x_j}\mathbf{W^{xg}} + \mathbf{h_{j-1}}\mathbf{W^{hg}})
\end{aligned}
\tag{39}
$$

$$
\mathbf{y_j} = O_{\text{LSTM}}(\mathbf{s_j}) = \mathbf{h_j}
$$

$$
\mathbf{s_j} \in \mathbb{R}^{2 \cdot d_h}, \;\; \mathbf{x_i} \in \mathbb{R}^{d_x}, \;\; \mathbf{c_j}, \mathbf{h_j}, \mathbf{i}, \mathbf{f}, \mathbf{o}, \mathbf{g} \in \mathbb{R}^{d_h}, \;\; \mathbf{W^{x\circ}} \in \mathbb{R}^{d_x \times d_h}, \;\; \mathbf{W^{h\circ}} \in \mathbb{R}^{d_h \times d_h},
$$

LSTM





Mathematically, the LSTM architecture is defined as:[72]

$$
\begin{aligned}
\mathbf{s_j} = R_{\text{LSTM}}(\mathbf{s_{j-1}}, \mathbf{x_j}) =& [\mathbf{c_j}; \mathbf{h_j}] \\
\mathbf{c_j} =& \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{h_j} =& \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{i} =& \sigma(\mathbf{x_j}\mathbf{W^{xi}} + \mathbf{h_{j-1}}\mathbf{W^{hi}}) \\
\mathbf{f} =& \sigma(\mathbf{x_j}\mathbf{W^{xf}} + \mathbf{h_{j-1}}\mathbf{W^{hf}}) \\
\mathbf{o} =& \sigma(\mathbf{x_j}\mathbf{W^{xo}} + \mathbf{h_{j-1}}\mathbf{W^{ho}}) \\
\mathbf{g} =& \tanh(\mathbf{x_j}\mathbf{W^{xg}} + \mathbf{h_{j-1}}\mathbf{W^{hg}})
\end{aligned}
\tag{39}
$$

$$\mathbf{y_j} = O_{\text{LSTM}}(\mathbf{s_j}) = \mathbf{h_j}$$

$\mathbf{s_j} \in \mathbb{R}^{2 \cdot d_h}$, $\mathbf{x_i} \in \mathbb{R}^{d_x}$, $\mathbf{c_j}, \mathbf{h_j}, \mathbf{i}, \mathbf{f}, \mathbf{o}, \mathbf{g} \in \mathbb{R}^{d_h}$, $\mathbf{W^{x\circ}} \in \mathbb{R}^{d_x \times d_h}$, $\mathbf{W^{h\circ}} \in \mathbb{R}^{d_h \times d_h}$,

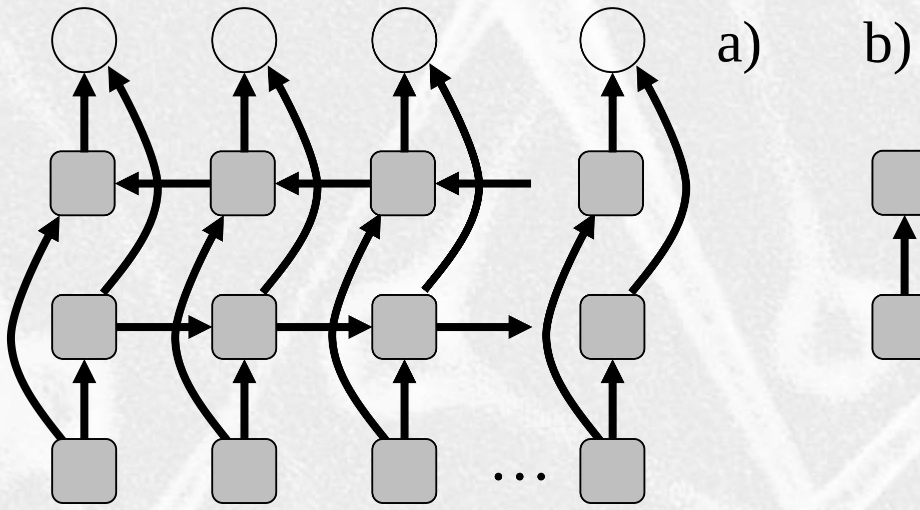
LSTMs and vanishing gradients

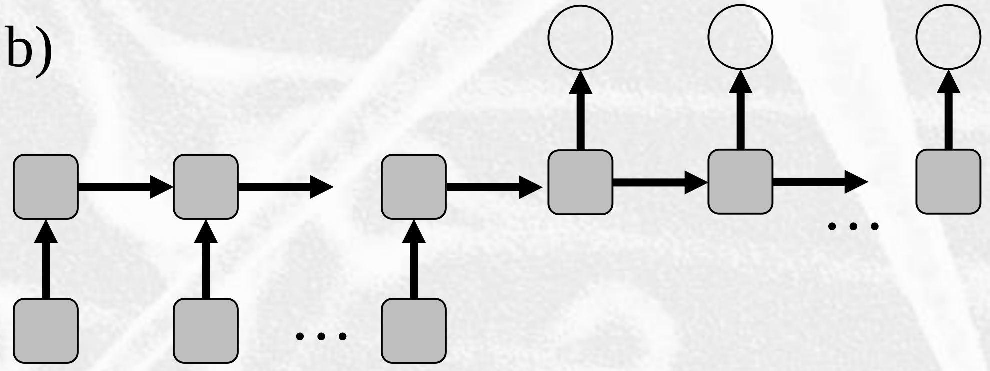
- The so-called "long short term memory" (LSTM) RNN architecture was specifically created to address the vanishing gradient problem.
- Uses a combination of hidden units, elementwise products and sums between units to implement gates that control "memory cells".
- Memory cells are designed to retain information without modification for long periods of time.
- They have their own input and output gates, which are controlled by learnable weights that are a function of the current observation and the hidden units at the previous time step.
- As a result, backpropagated error terms from gradient computations can be stored and propagated backwards without degradation.

Other RNN architectures

a) Recurrent networks can be made bidirectional, propagating information in both directions
   - They have been used for a wide variety of applications, including protein secondary structure prediction and handwriting recognition

b) An "encoder-decoder" network creates a fixed-length vector representation for variable-length inputs, the encoding can be used to generate a variable-length sequence as the output
   - Particularly useful for machine translation





Training different Types of RNNs

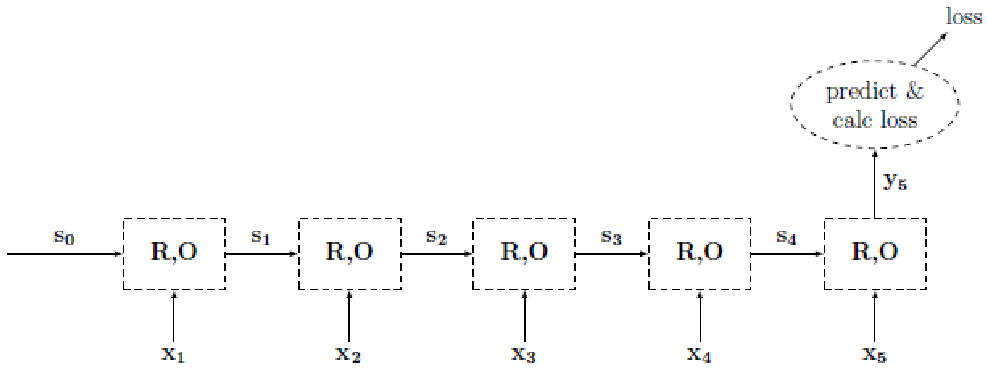


Figure 7: Acceptor RNN Training Graph.

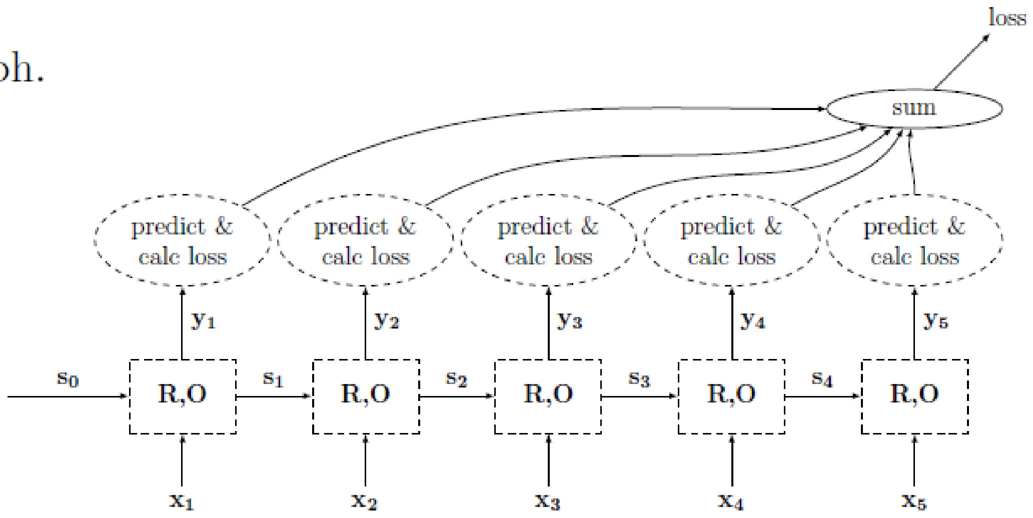
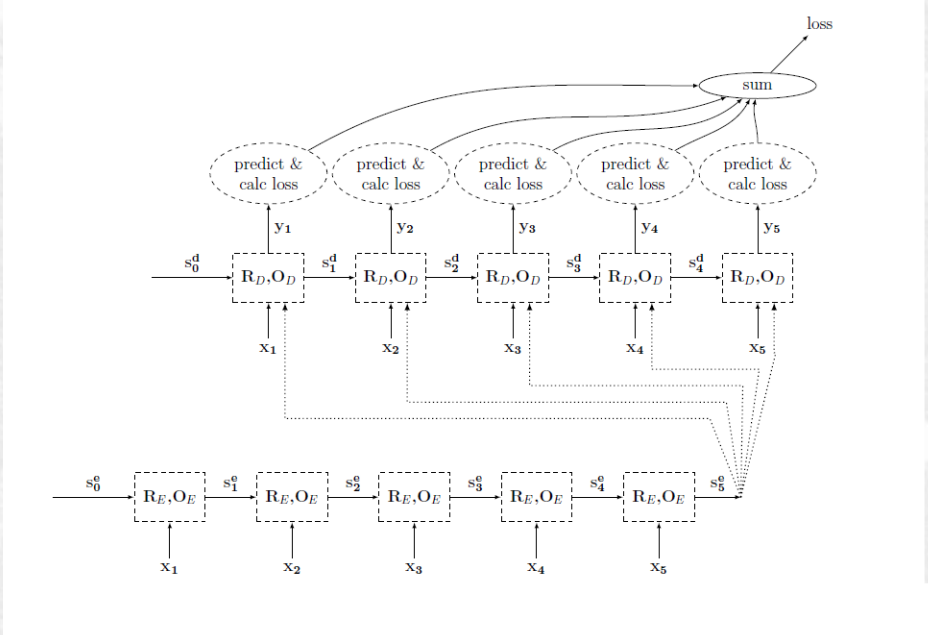


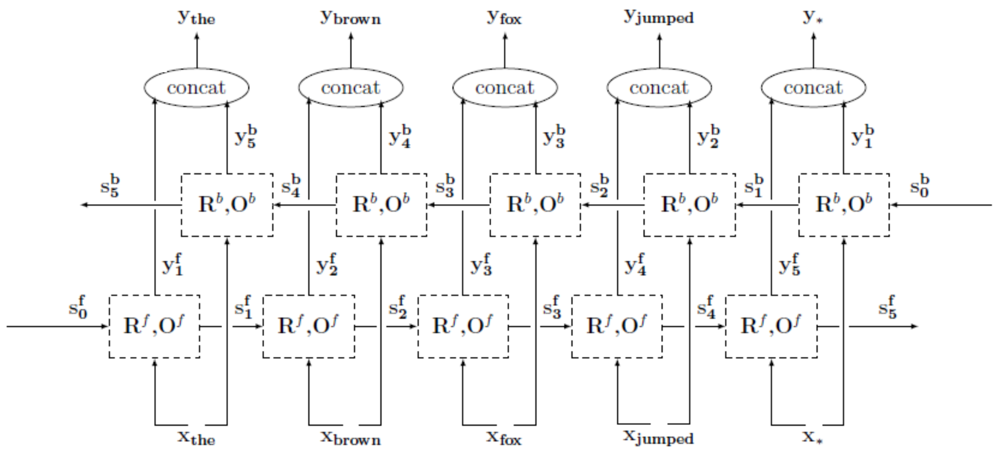
Figure 8: Transducer RNN Training Graph.

Training different Types of RNNs



loss
sum
predict & calc loss
y_1 y_2 y_3 y_4 y_5
s_0^d s_1^d s_2^d s_3^d s_4^d
R_D,O_D
x_1 x_2 x_3 x_4 x_5
s_0^e s_1^e s_2^e s_3^e s_4^e s_5^e
R_E,O_E

Figure 9: Encoder-Decoder RNN Training Graph.



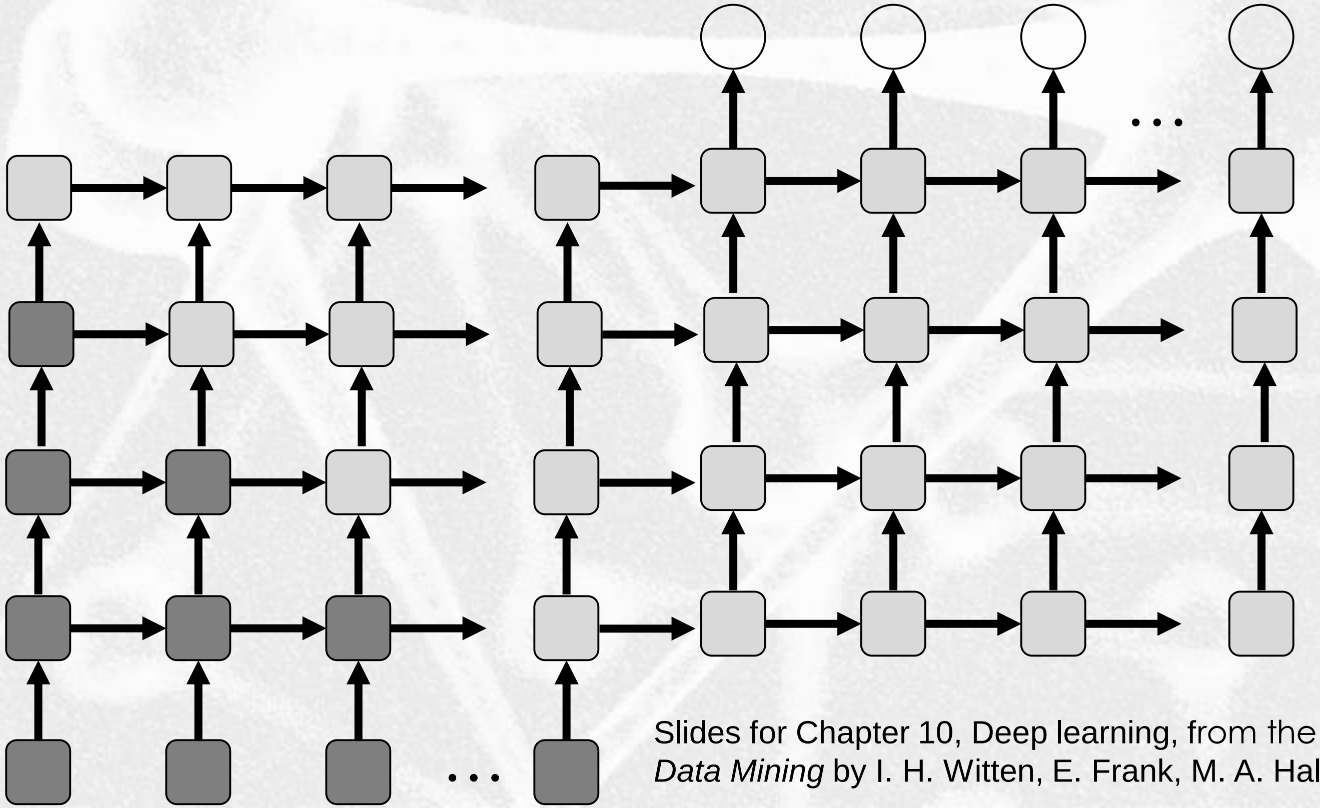
y_{the} y_{brown} y_{fox} y_{jumped} y_*
concat
y_5^b y_4^b y_3^b y_2^b y_1^b
s_5^b s_4^b s_3^b s_2^b s_1^b s_0^b
R^b,O^b
y_1^f y_2^f y_3^f y_4^f y_5^f
s_0^f s_1^f s_2^f s_3^f s_4^f s_5^f
R^f,O^f
x_{the} x_{brown} x_{fox} x_{jumped} x_*

Figure 11: biRNN over the sentence "the brown fox jumped .".

Encoder-decoder deep architectures

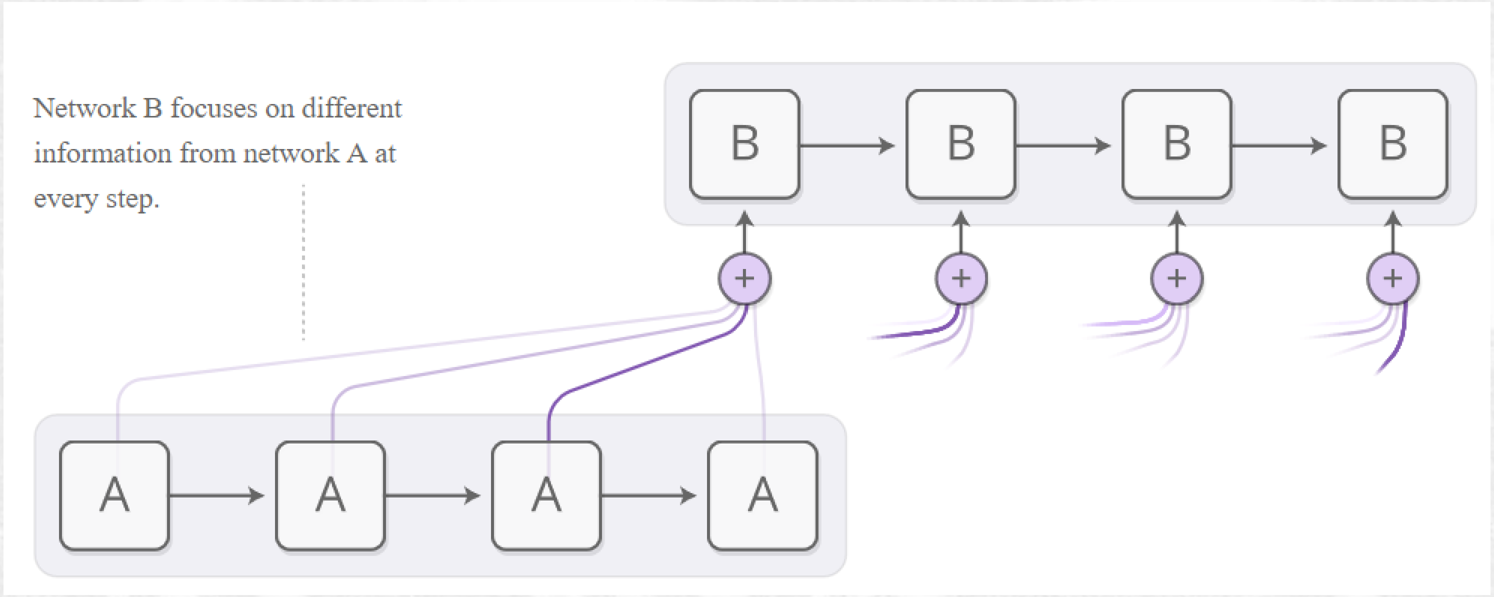
- Given enough data, a deep encoder-decoder architecture (see below) can yield results that compete with hand-engineered translation systems.

- The connectivity structure means that partial computations in the model can flow through the graph in a wave (darker nodes in fig.)



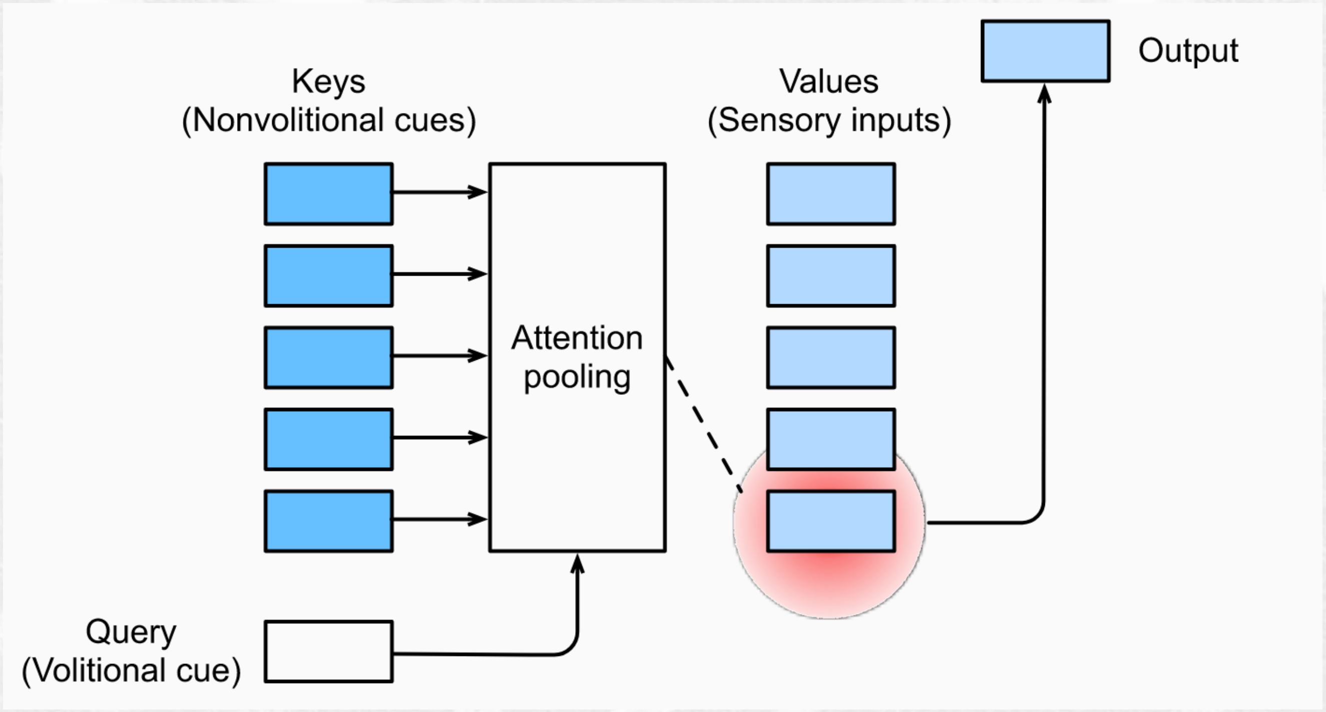
Attention-based RNNs

- A NN (e.g. B) is used to attend the outcome of a second network A, e.g. (Vaswani et al., 2017)

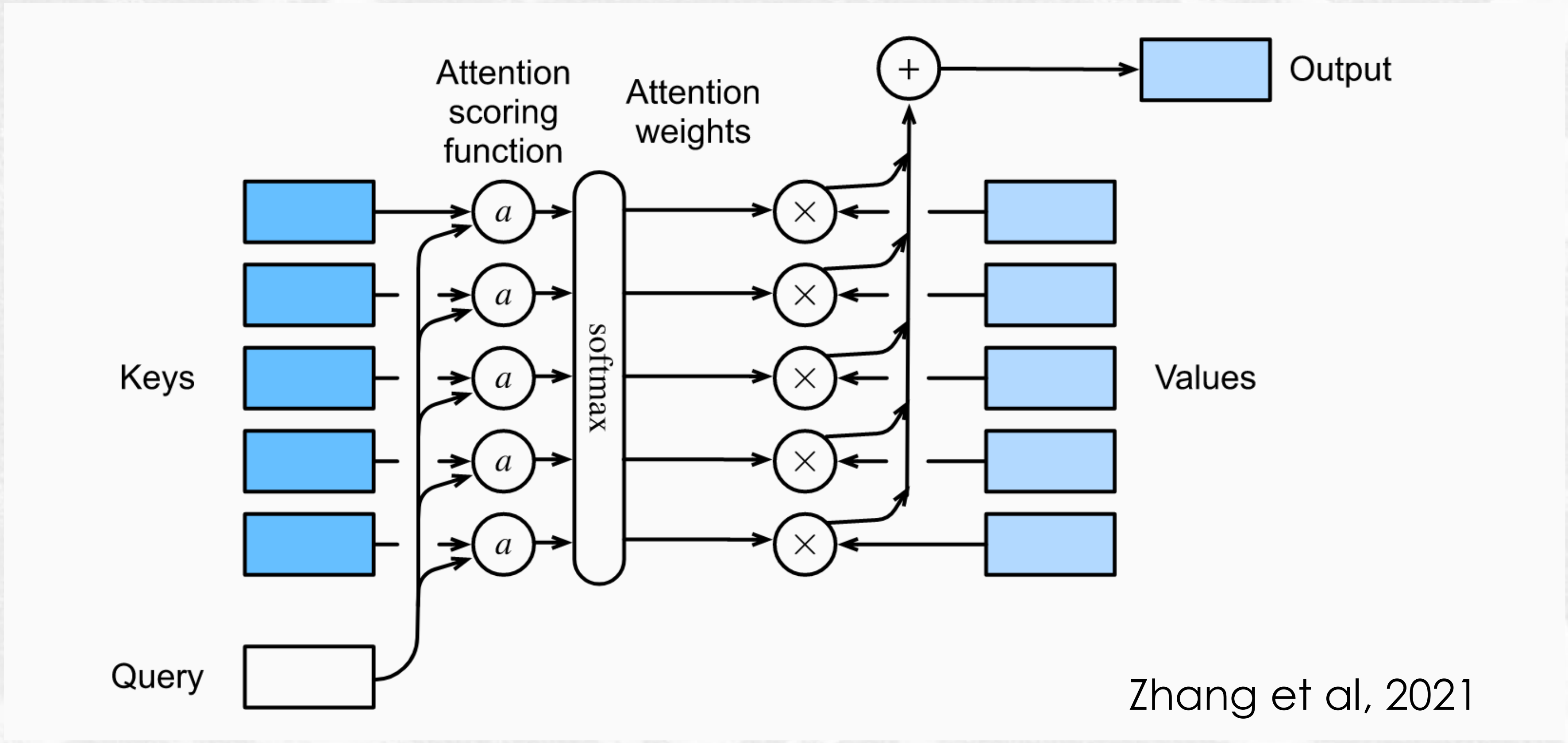

Network B focuses on different information from network A at every step.
B
B
B
B
+
+
+
+
A
A
A
A

Attention: motivations

- From (*Dive into Deep Learning*, Zhang, Aston and Lipton, Zachary C. and Li, Mu and Smola, Alexander J., 2021).



Attention functions



Zhang et al, 2021

Attention functions: examples (1)

- In general, when queries and keys are vectors of different lengths, we can use additive attention as the scoring function. Given a query $\mathbf{q} \in \mathbb{R}^q$ and a key $\mathbf{k} \in \mathbb{R}^k$, the *additive attention* scoring function

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{w}_v^\top \tanh(\mathbf{W}_q\mathbf{q} + \mathbf{W}_k\mathbf{k}) \in \mathbb{R},$$

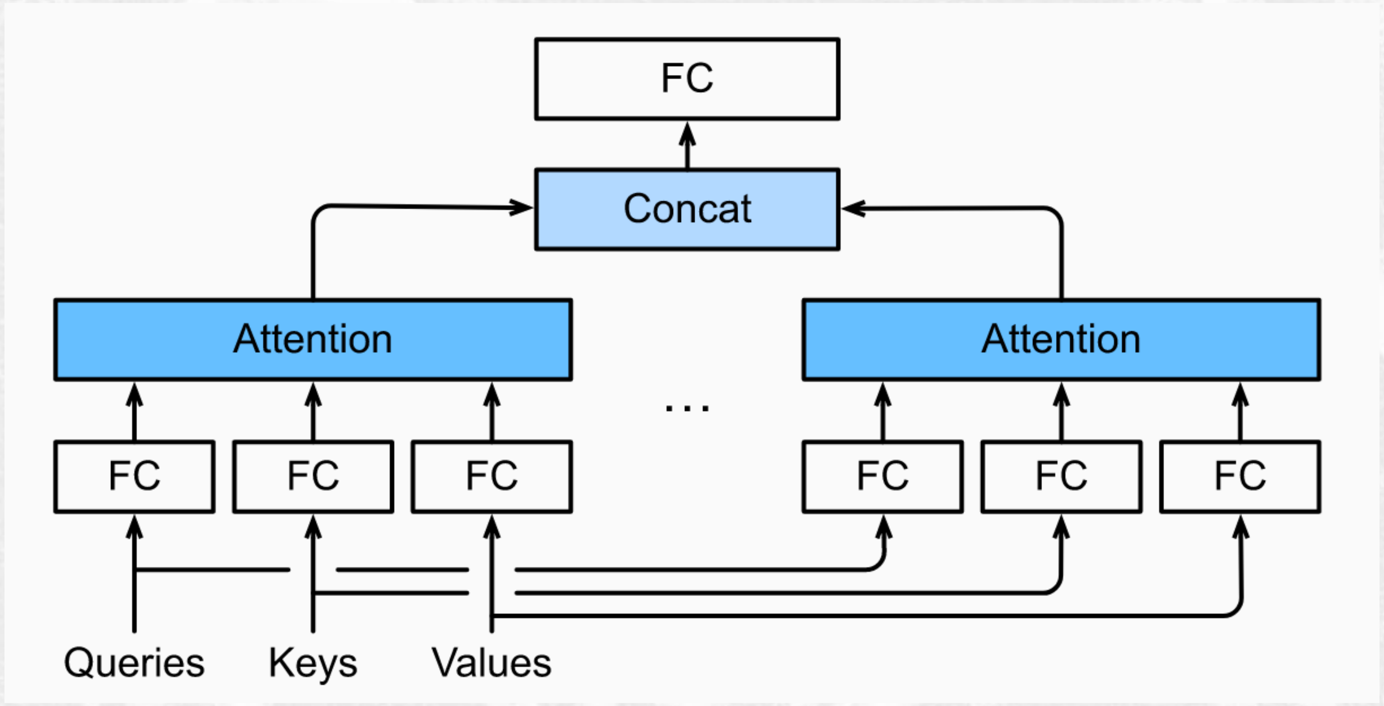
- where learnable parameters $\mathbf{W}_q \in \mathbb{R}^{h\times q}$, $\mathbf{W}_k \in \mathbb{R}^{h\times k}$ and $\mathbf{w}_v \in \mathbb{R}^h$.
- In a learnable setting, the query and the key are concatenated and fed into an MLP with a single hidden layer whose number of hidden units is *h*, a hyperparameter. By using as the activation function and disabling bias terms, we implement additive attention in the following

Attention functions: scaled dot-product (2)

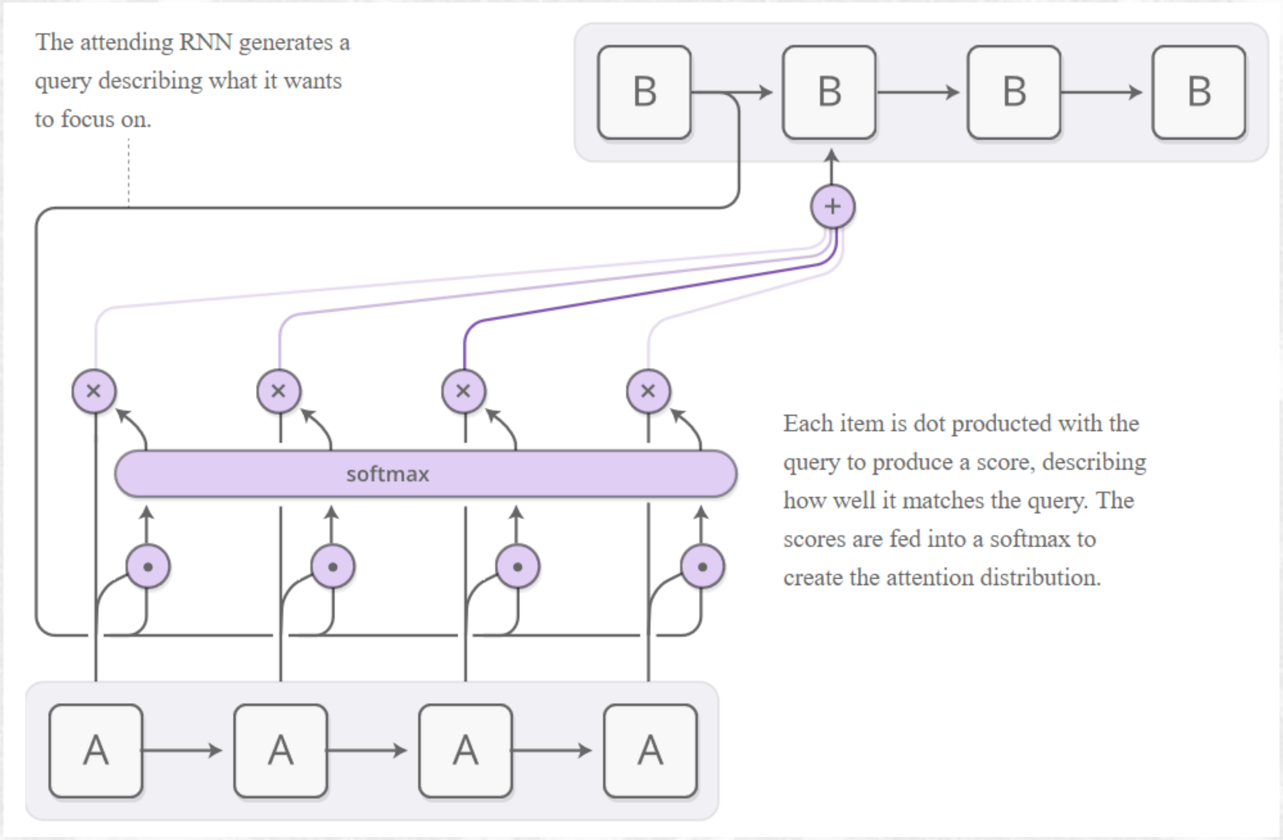
- When *q* and *k* are *d*-dimensional vectors whose independent dimensions have mean=0 and variance=1, their dot product has mean = 0 and a variance = *d*. To ensure that the variance of the dot product still remains one regardless of vector length, the *scaled dot-product attention* scoring function is adopted $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^\top \mathbf{k} / \sqrt{d}$
- It divides the dot product by $\sqrt{d}$. In practice, we often think in minibatches for efficiency, such as computing attention for *n* queries and *m* key-value pairs, where queries and keys are of length *d* and values are of length v. The scaled dot-product attention of queries $\mathbf{Q} \in \mathbb{R}^{n \times d}$, keys $\mathbf{K} \in \mathbb{R}^{m \times d}$, and values $\mathbf{V} \in \mathbb{R}^{m \times v}$ is

$$\mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V} \in \mathbb{R}^{n \times v}.$$

Attention: multihead



Attention-based RNNs


The attending RNN generates a query describing what it wants to focus on.

Each item is dot producted with the query to produce a score, describing how well it matches the query. The scores are fed into a softmax to create the attention distribution.

Attention mechanisms in Machine Translation

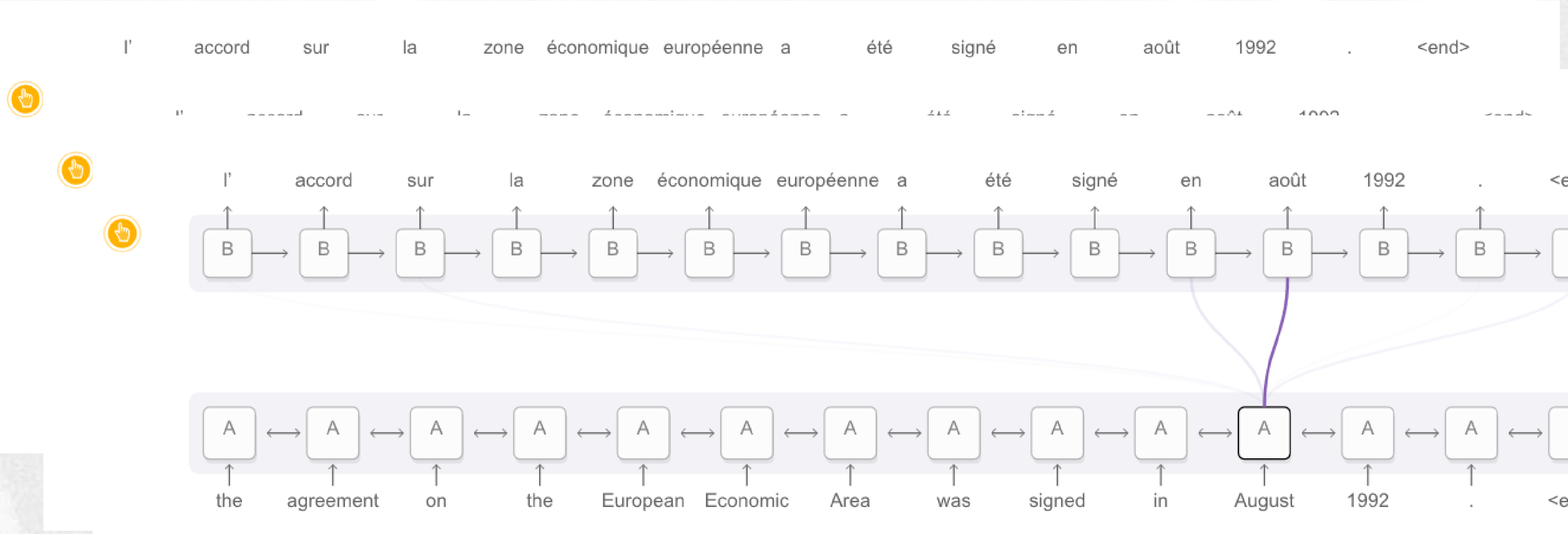
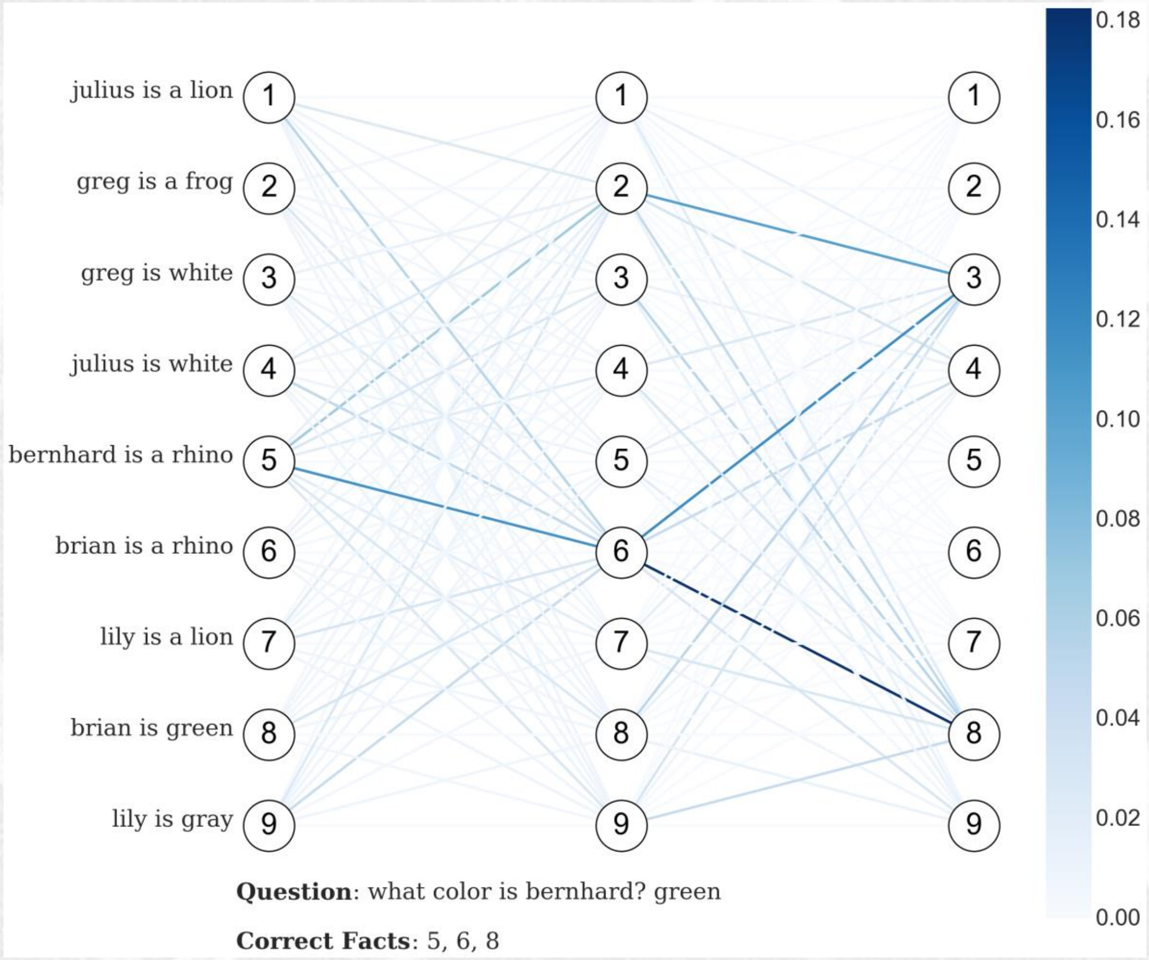

l' accord sur la zone économique européenne a été signé en août 1992 . <end>
l' accord sur la zone économique européenne a été signé en août 1992 .
B
A
the agreement on the European Economic Area was signed in August 1992 .

Diagram derived from Fig. 3 of Bahdanau, *et al.* 2014

Visualization of the attention distribution in QA

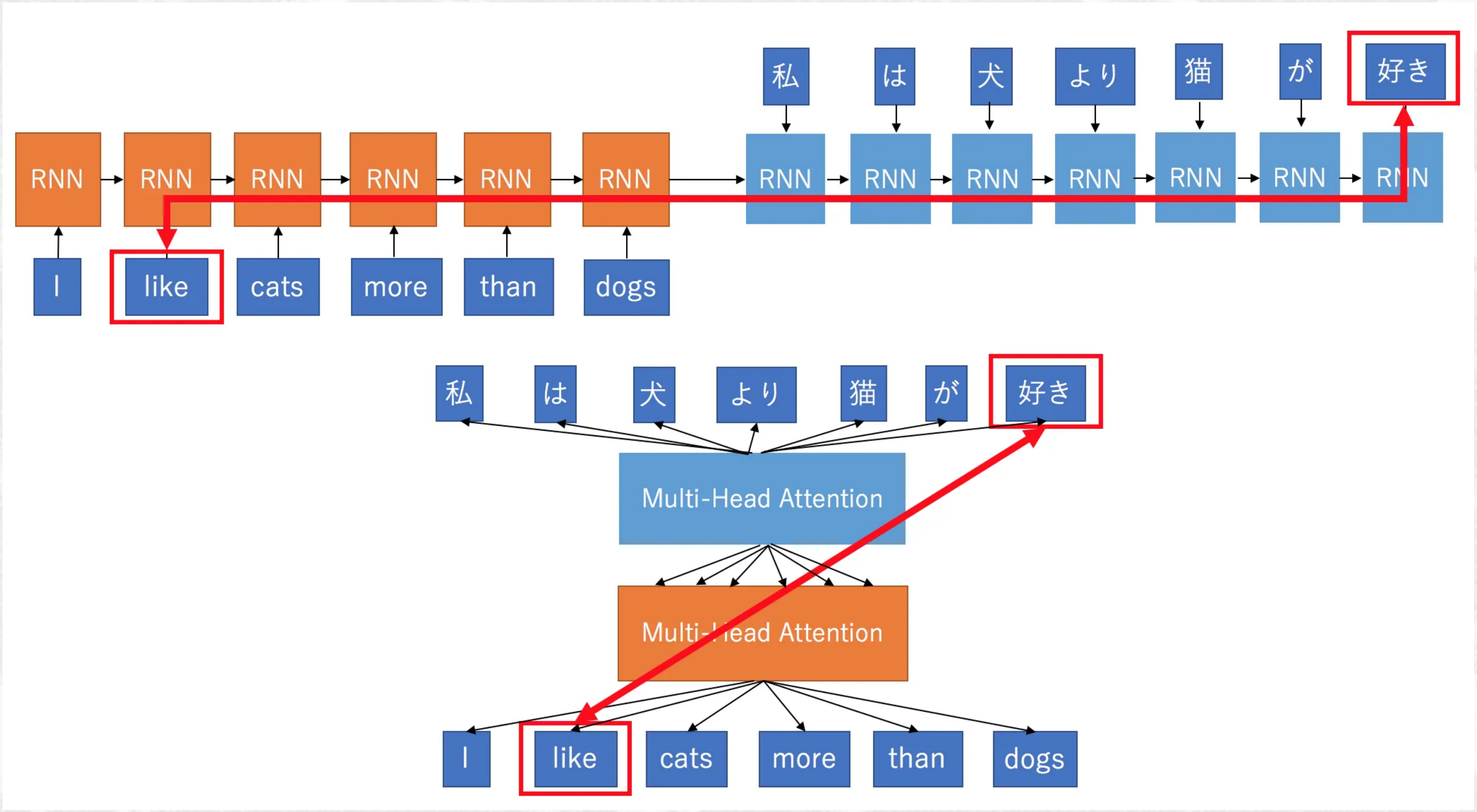
- Supporting fact sequences for an example question
- On the right the attentions over facts for individual sequences
  - Each sequence is mapped into a Markov process



Attention & enconding

- IN a decoding process (e.g. machine translation) there are **three** kinds of dependencies for neural architectures
- Dependencies can establish between
- (1) the ***input and output*** tokens
- (2) the ***input tokens themselves***
- (3) the ***output tokens themselves***
- Examples:
  - MT
  - QA where the query the answer paragraph is the input and the matched answer is the output

Attention in MT:
long distance dependencies



From RNNs to Transformers

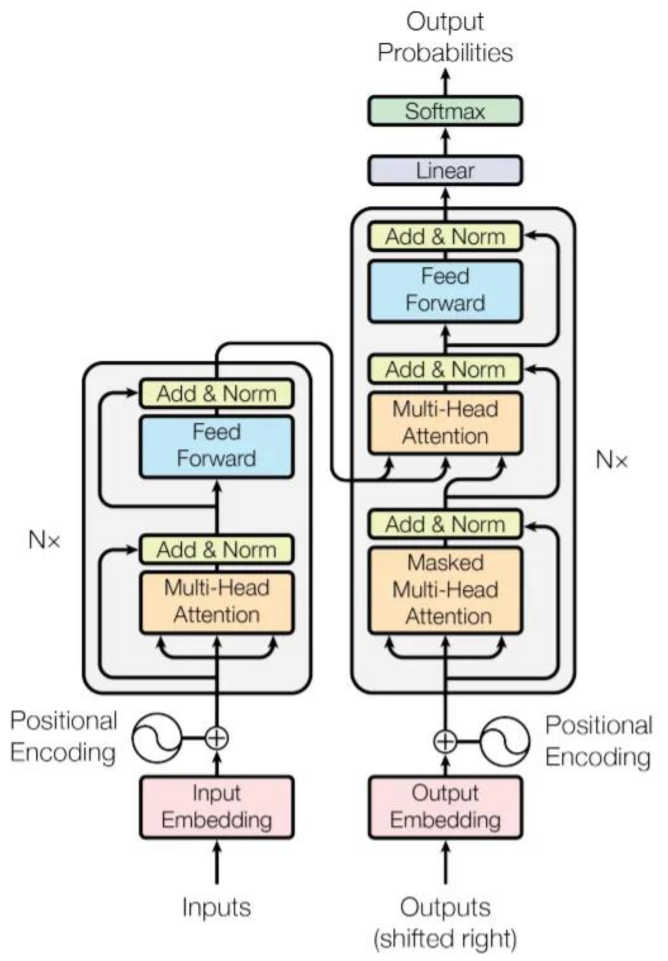
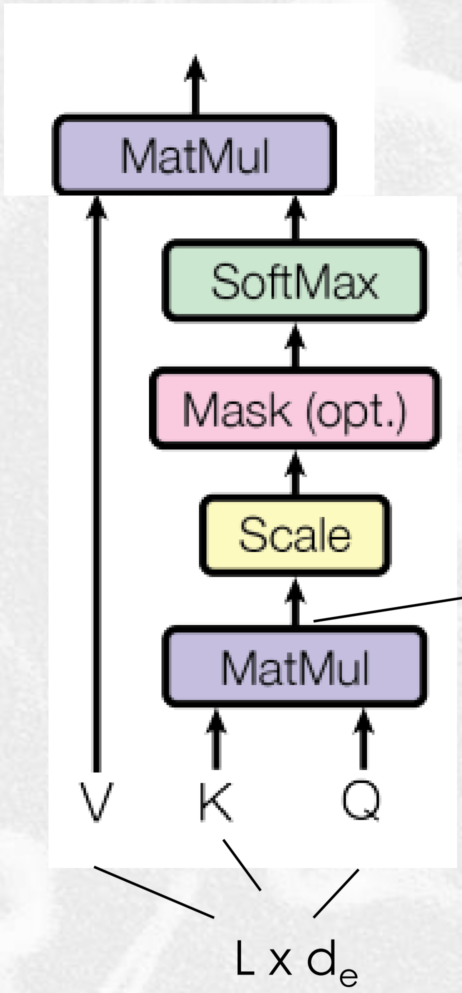


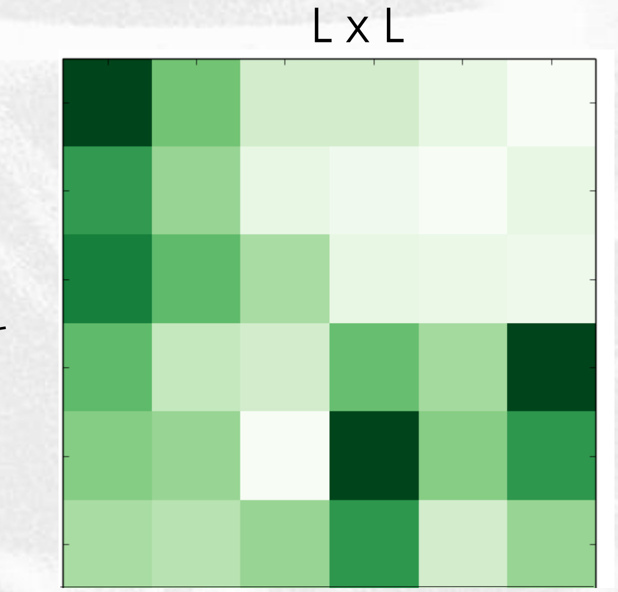
Figure 1: The Transformer - model architecture.

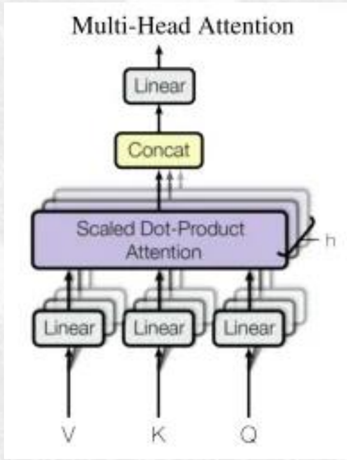
Bidirectional Encoder Representations from **BERT** - Transformers (Devlin et al. '18)

Scaled Dot-Product Attention

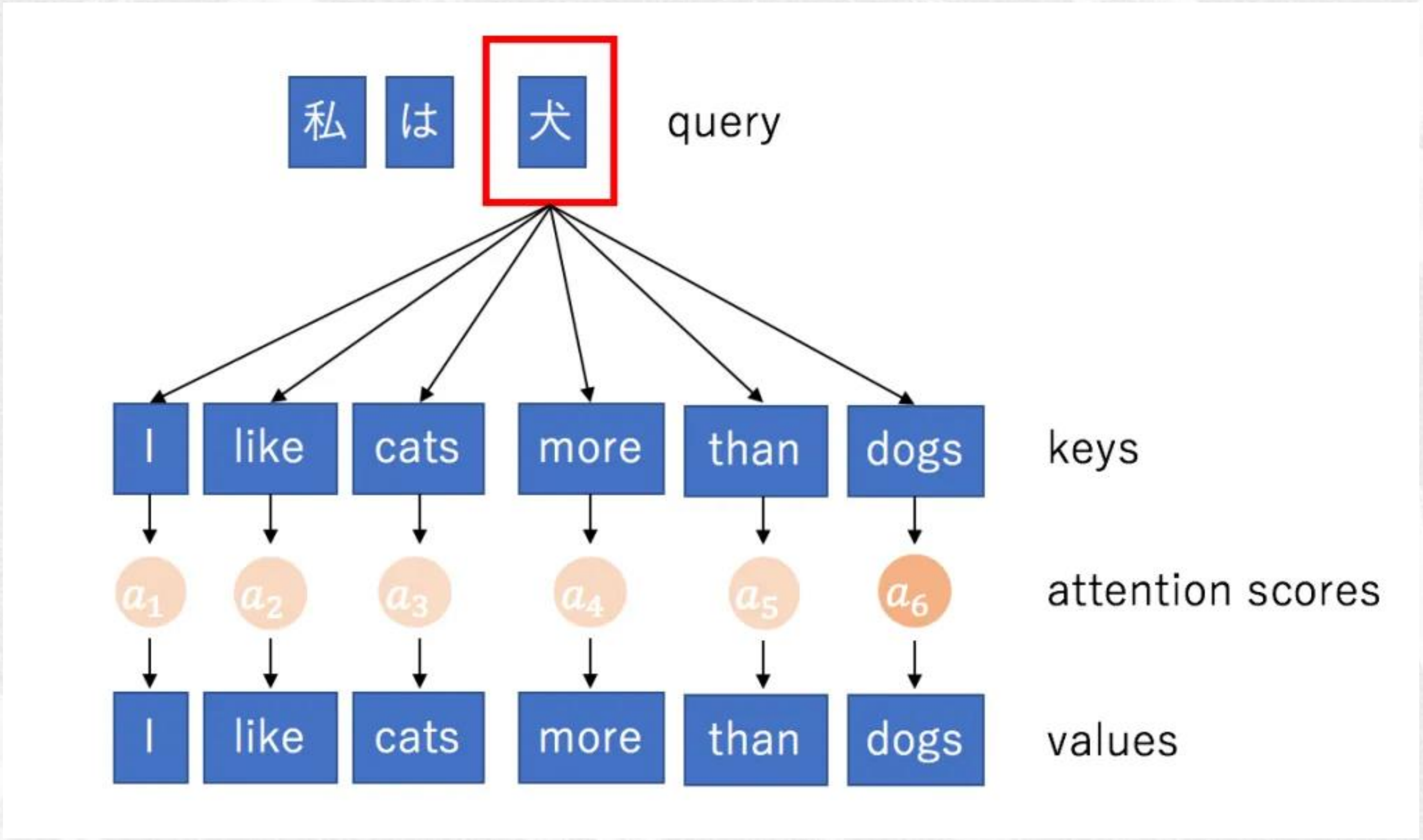
Attention is a function that maps a query Q and a set of key-value pairs <K,V> to an output





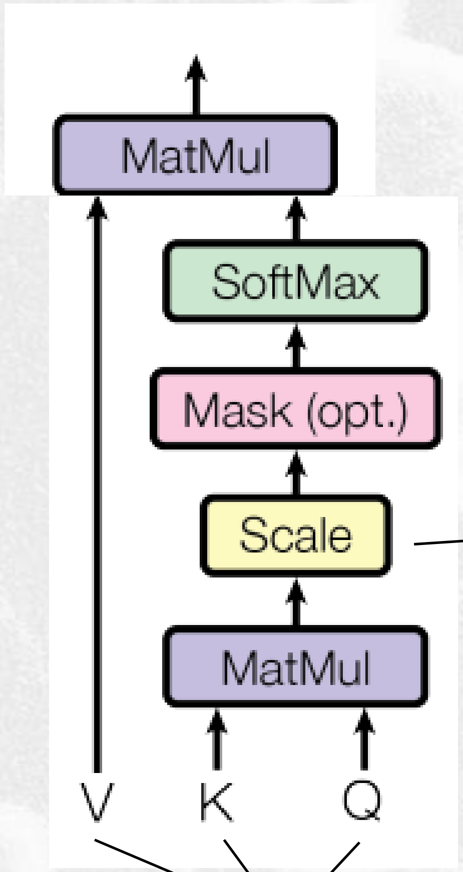


Input-Output Attention



BERT (Devlin et al. '18)

Scaled Dot-Product Attention

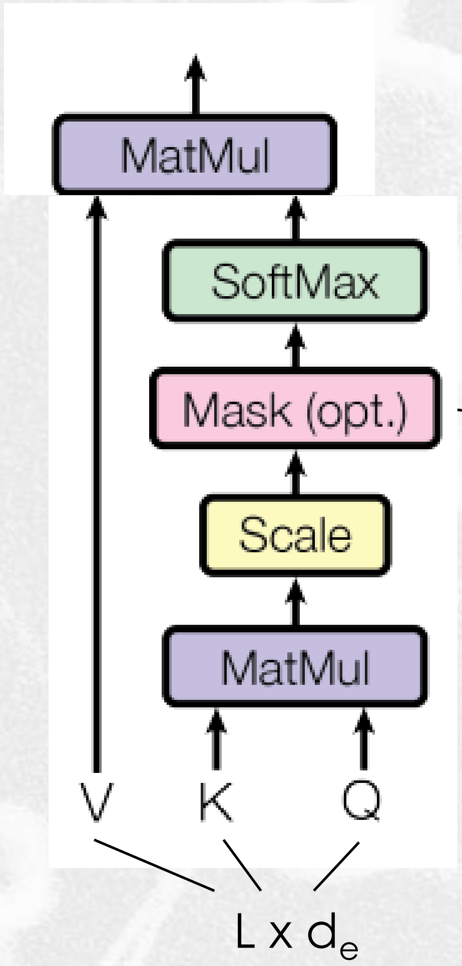


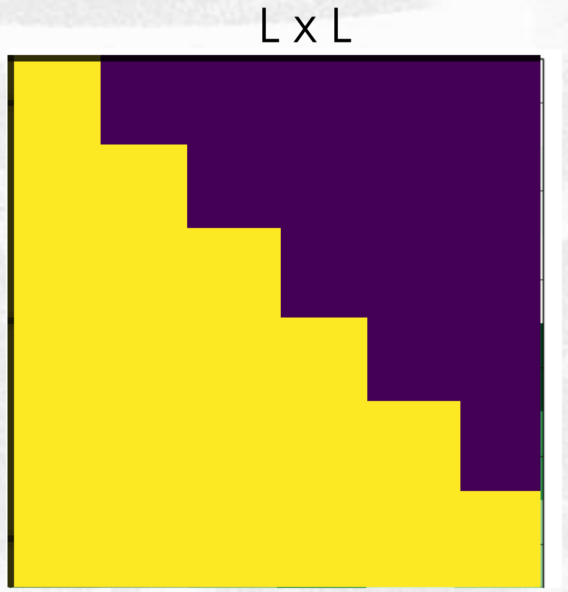
$L \times d_e$

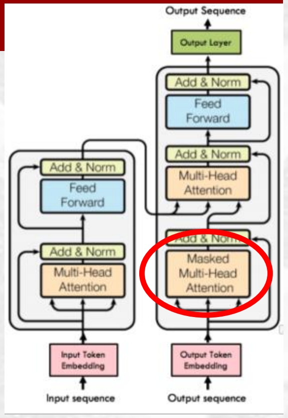
Division by $\sqrt{d_e}$
Only for numerical stability

BERT (Devlin et al. '18)

Scaled Dot-Product Attention

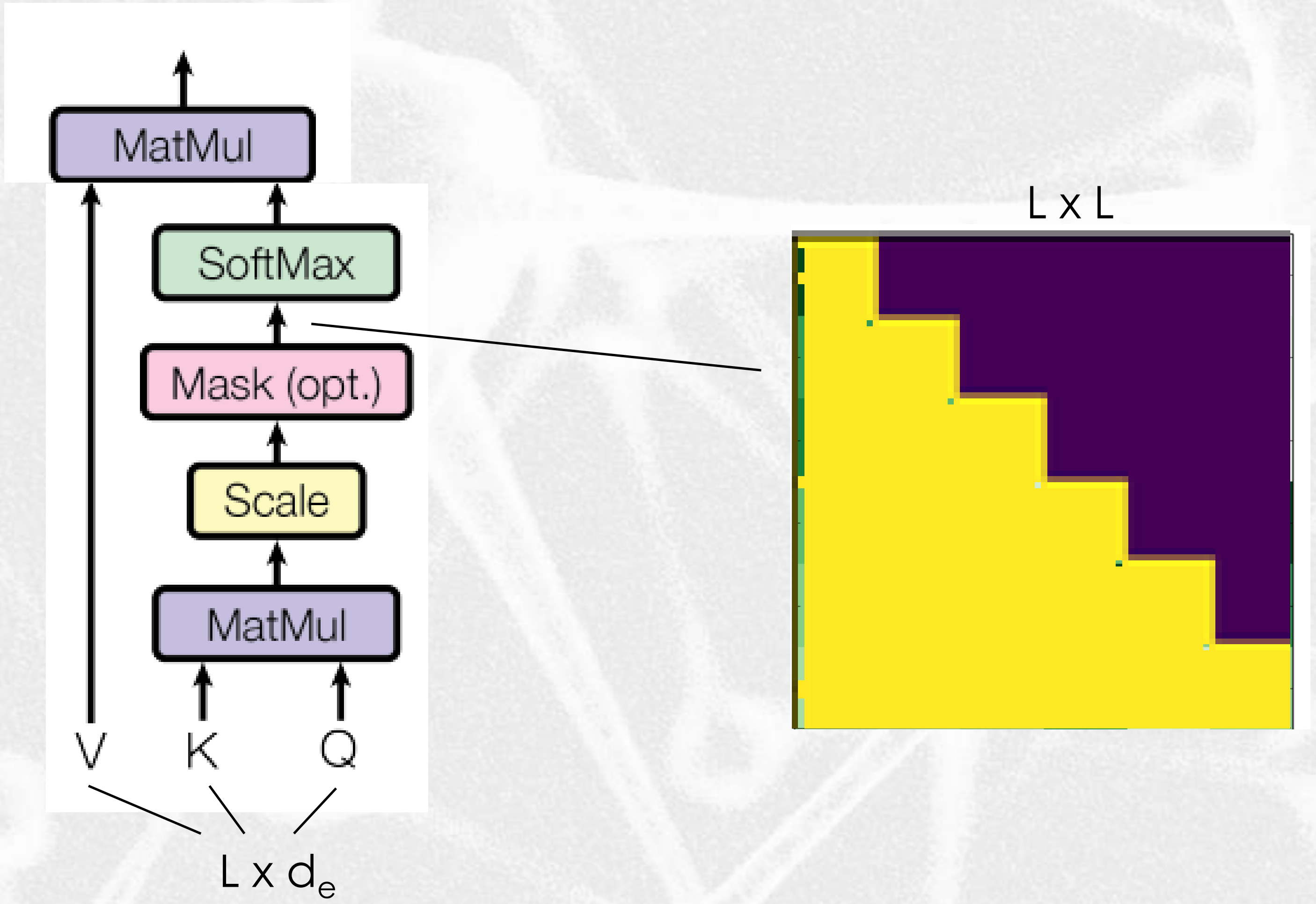


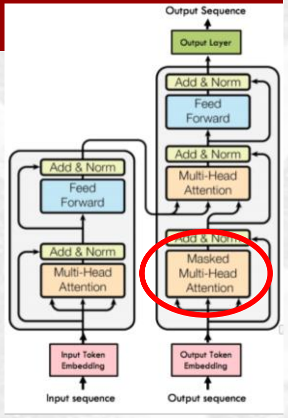




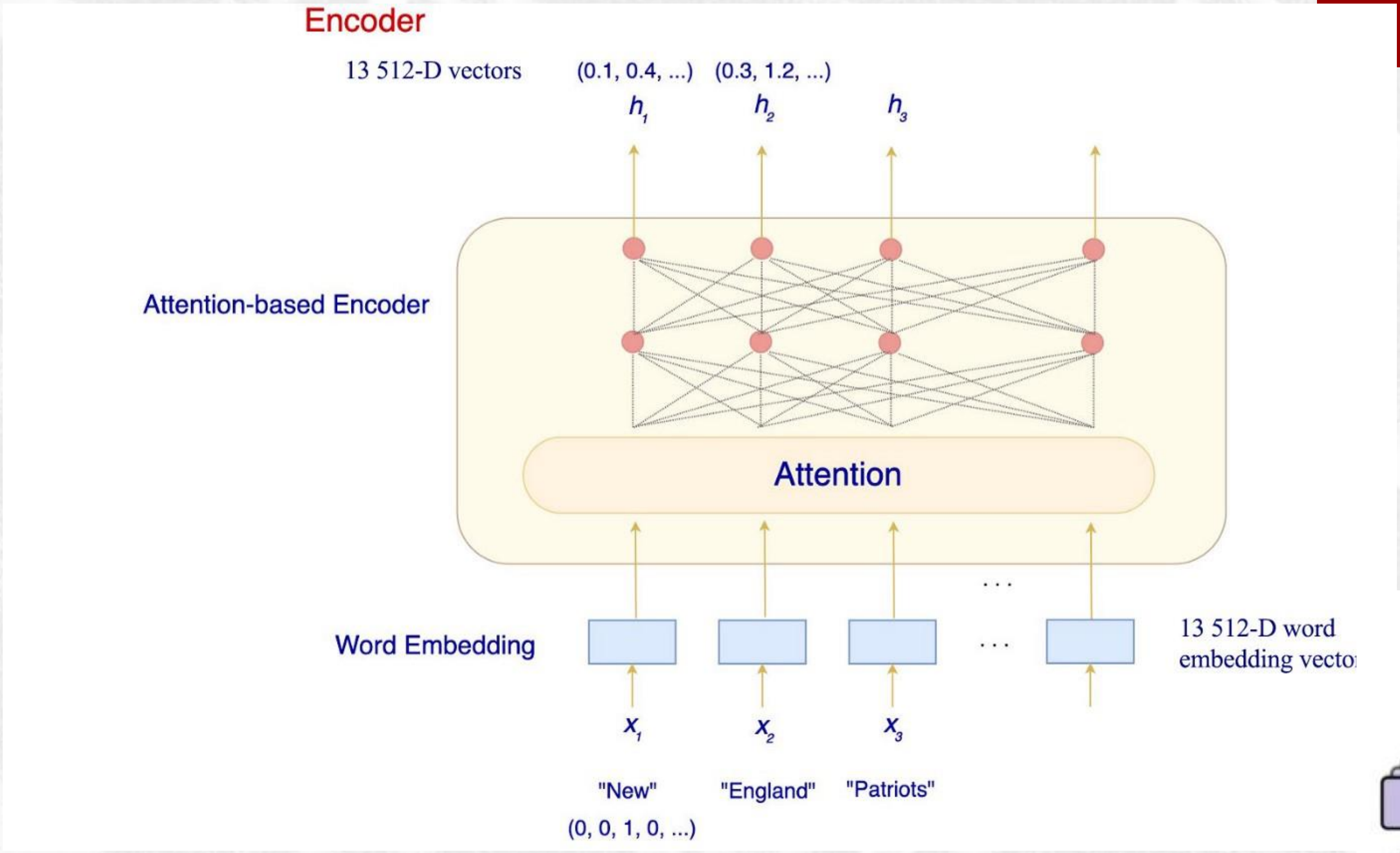
BERT (Devlin et al. '18)

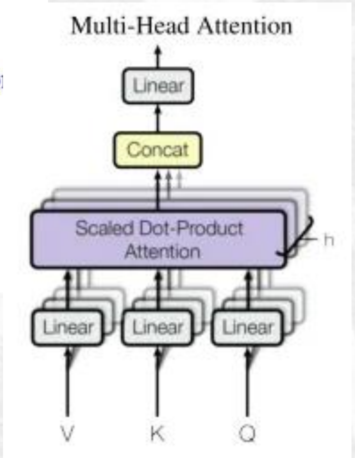
Scaled Dot-Product Attention





BERT & NLP

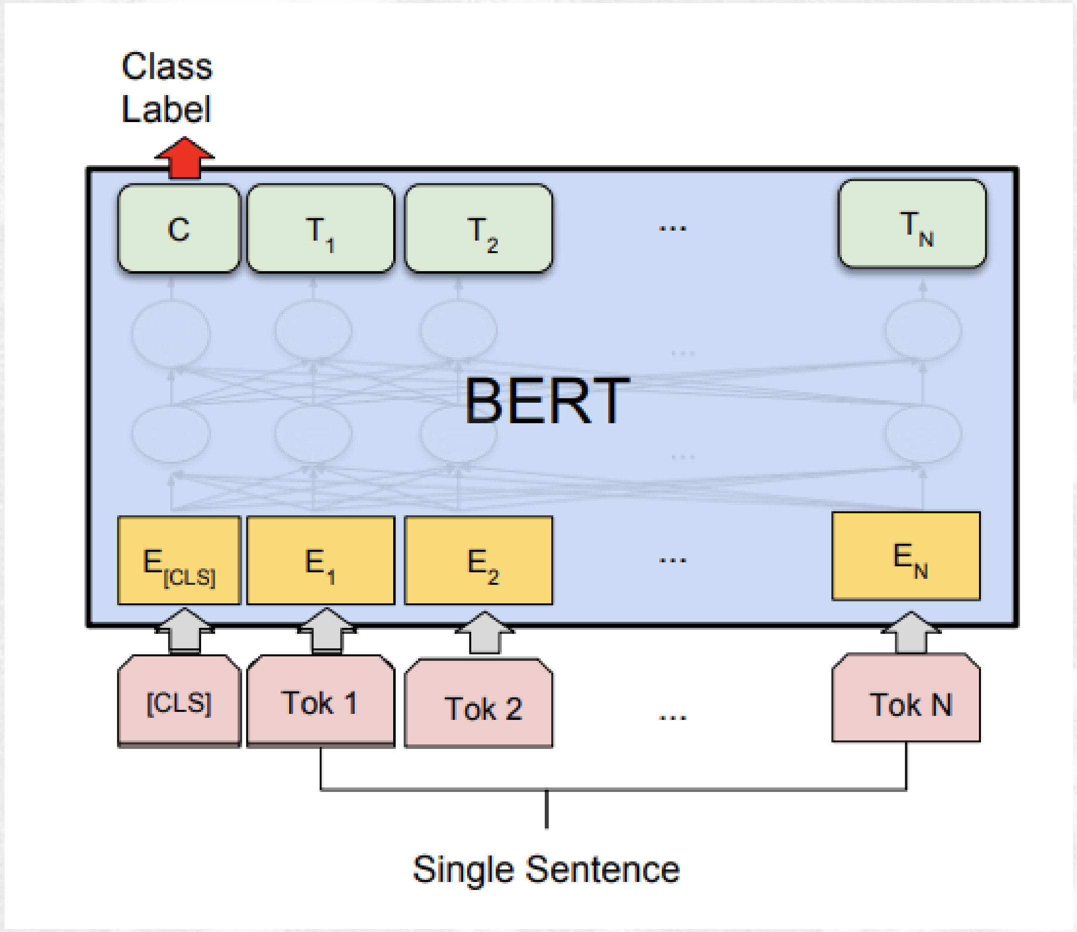

Encoder
13 512-D vectors
(0.1, 0.4, ...)
(0.3, 1.2, ...)
h_1
h_2
h_3
Attention-based Encoder
Attention
. . .
Word Embedding
. . .
13 512-D word
embedding vecto
x_1
x_2
x_3
"New"
"England"
"Patriots"
(0, 0, 1, 0, ...)


Multi-Head Attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q

BERT & NLP (2)

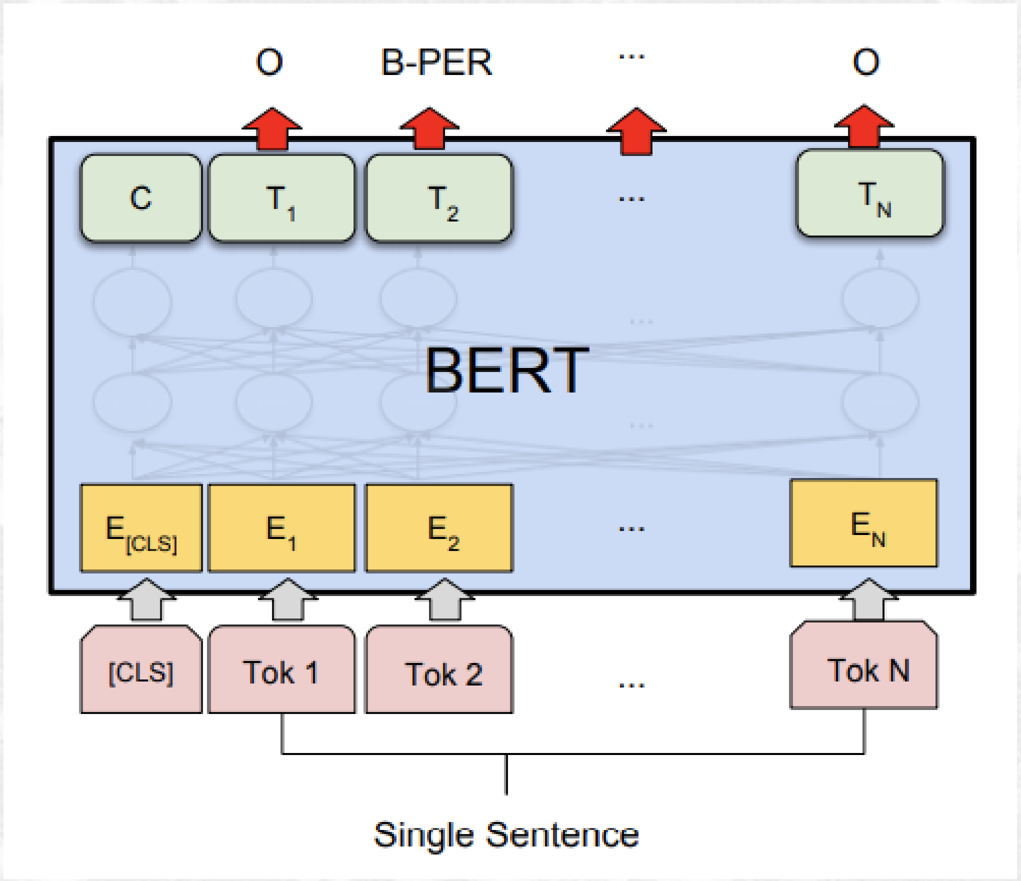
- How to optimize the encoding?
- General and complex tasks defined in (Devlin et al., 2018) are
  - Masked Language Modeling (15%)
    - Inpired by Distributional Hypothesis
    - Can be Simulated and does not require any labeling
  - Next Sentence Prediction
    - Inspired by Textual Inference tasks (e.g. Textual Entailment)
    - Can be Simulated and does not require any labeling
- Source Representations
  - Words? And why not subword (in the BERT jargon: word pieces)?
    - Useful to deal with out-of-vocabulary phenomena

BERT (Devlin et al. '18)



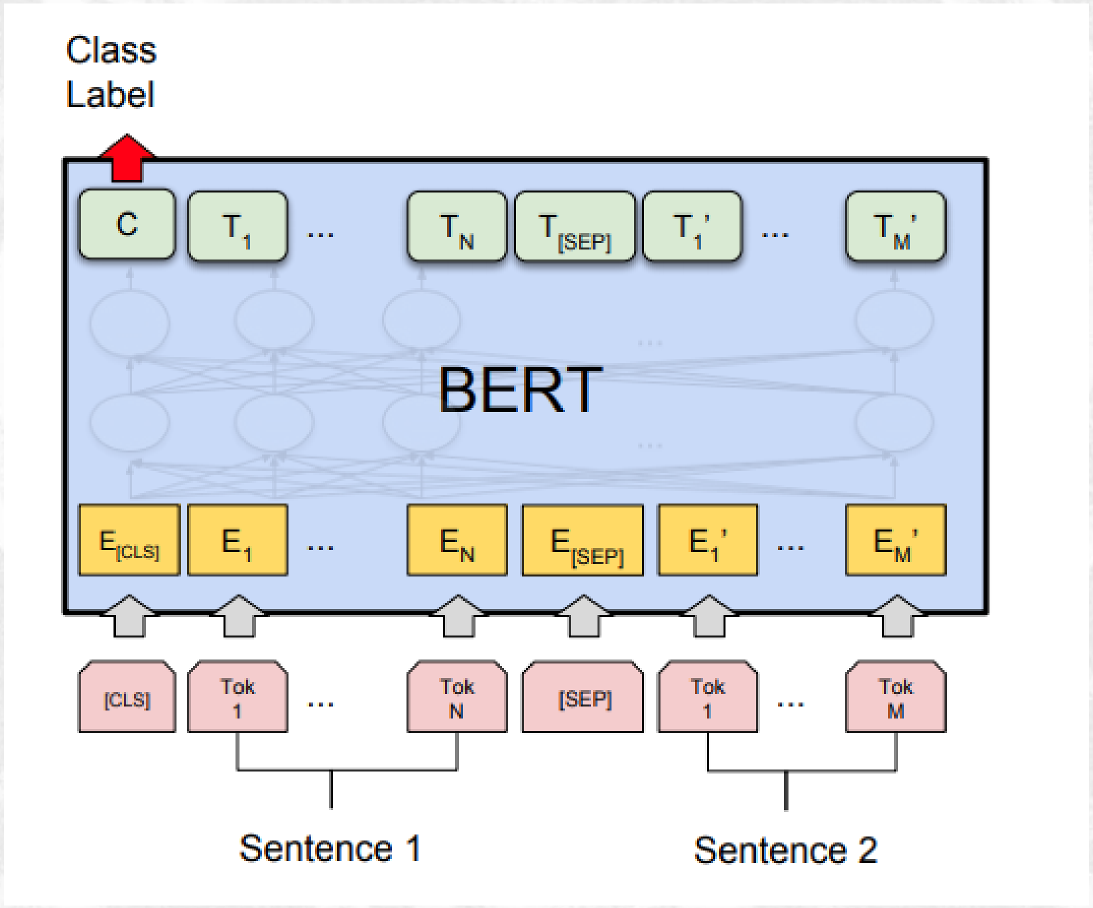
BERT for single sentence classification (Sentiment analysis, Intent Classification, etc.)

BERT (Devlin et al. '18)



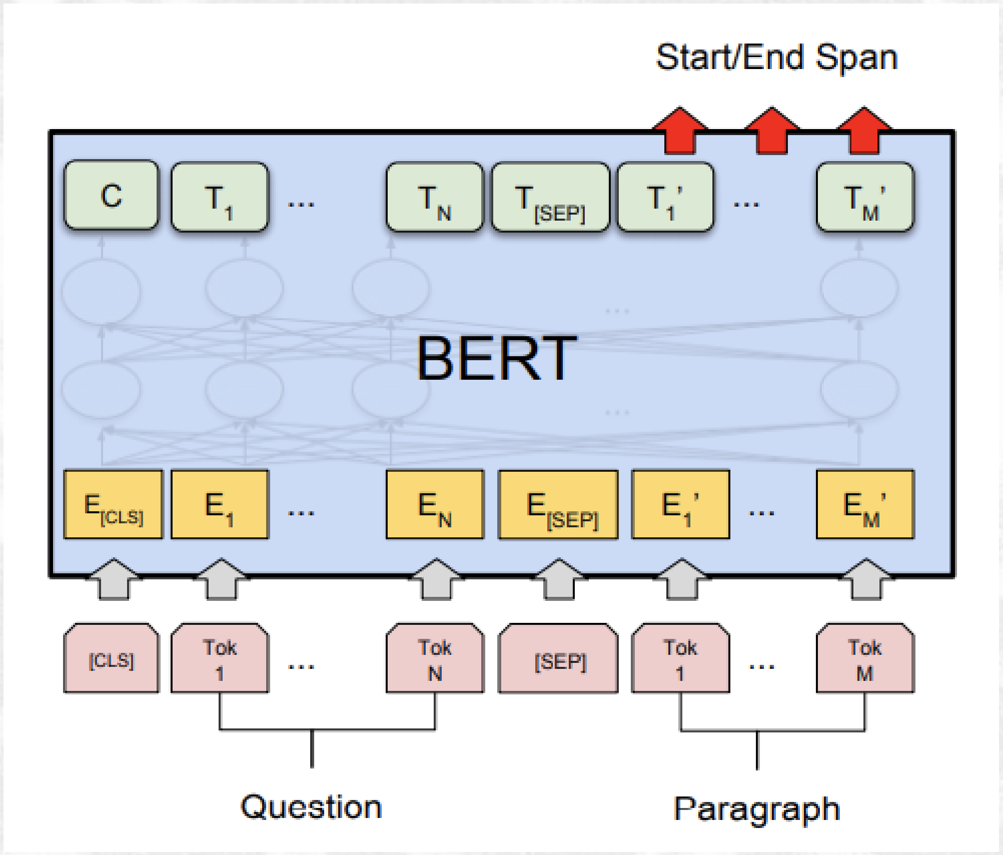
BERT for Sequence Tagging Tasks (e.g., POS tagging, Named Entity Recognition, etc.)

BERT (Devlin et al. '18)



BERT for sentence pairs classification (Paraphrase Identification, answer selection in QA, Recognizing Textual Entailment)

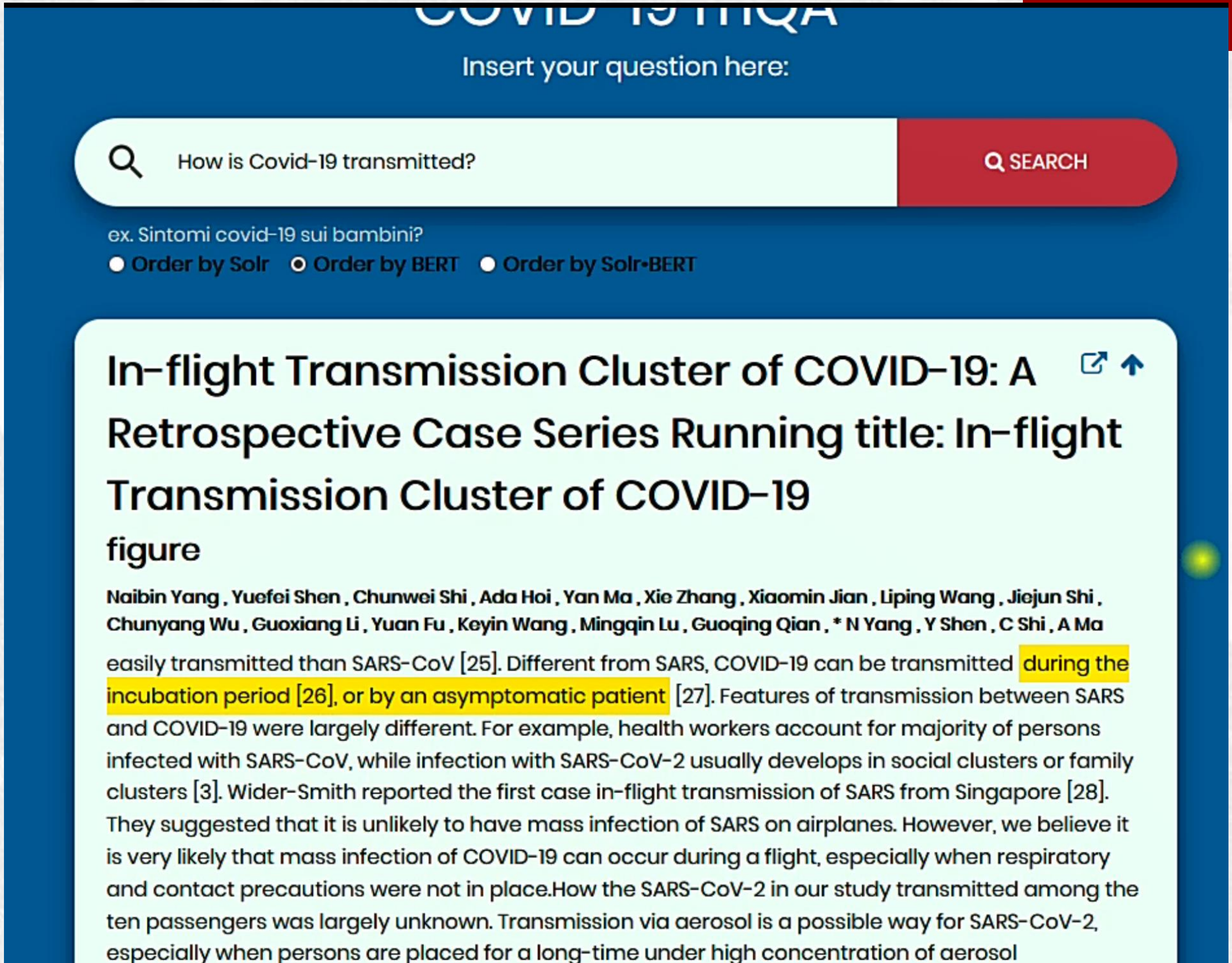
BERT (Devlin et al. '18)



BERT for Answer Span Selection in Question Answering

A QA example on SquAD

- Cross-lingual Question Answering



Insert your question here:

How is Covid-19 transmitted?

ex. Sintomi covid-19 sui bambini?

Order by Solr [✓ Order by BERT] Order by Solr+BERT

In-flight Transmission Cluster of COVID-19: A Retrospective Case Series Running title: In-flight Transmission Cluster of COVID-19

figure

Naibin Yang , Yuefei Shen , Chunwei Shi , Ada Hoi , Yan Ma , Xie Zhang , Xiaomin Jian , Liping Wang , Jiejun Shi , Chunyang Wu , Guoxiang Li , Yuan Fu , Keyin Wang , Mingqin Lu , Guoqing Qian , * N Yang , Y Shen , C Shi , A Ma

easily transmitted than SARS-CoV [25]. Different from SARS, COVID-19 can be transmitted during the incubation period [26], or by an asymptomatic patient [27]. Features of transmission between SARS and COVID-19 were largely different. For example, health workers account for majority of persons infected with SARS-CoV, while infection with SARS-CoV-2 usually develops in social clusters or family clusters [3]. Wider-Smith reported the first case in-flight transmission of SARS from Singapore [28]. They suggested that it is unlikely to have mass infection of SARS on airplanes. However, we believe it is very likely that mass infection of COVID-19 can occur during a flight, especially when respiratory and contact precautions were not in place.How the SARS-CoV-2 in our study transmitted among the ten passengers was largely unknown. Transmission via aerosol is a possible way for SARS-CoV-2, especially when persons are placed for a long-time under high concentration of aerosol

BERT (Devlin et al. '18)

Pretraining on two unsupervised prediction tasks:

- **Masked Language Model**: given a sentence *s* with missing words, reconstruct *s*
  - Example: Amazon <MASK> amazing → Amazon is amazing
  - In BERT the language modeling is deeply Bidirectional, while in ELMo the forward and backward LMs were two independent branches of the NN

- **Next Sentence Prediction**: given two sentences s_1 and s_2, the task is to understand whether s_2 is the actual sentence that follows s_1
  - 50% of the training data are positive examples: s_1 and s_2 are actually consecutive sentences
  - 50% of the training data are negative examples: s_1 and s_2 are randomly chosen from the corpus

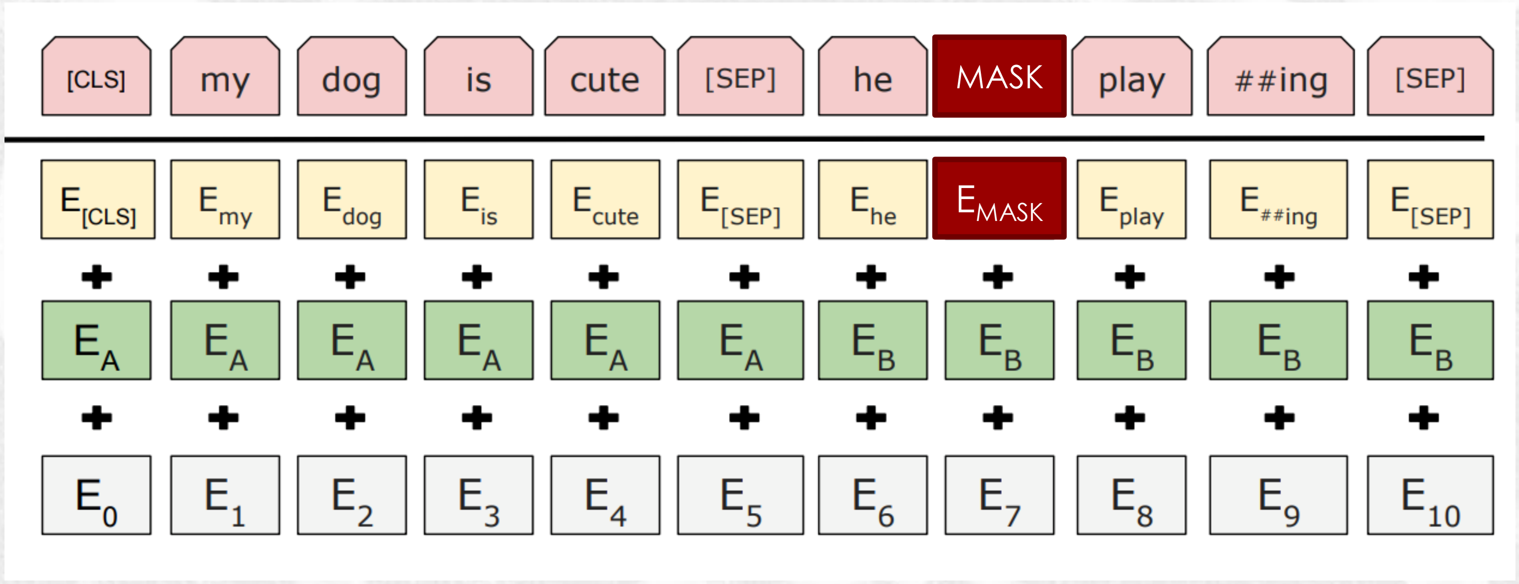
BERT pretraining:
Input representations

INPUT

WordPieces Embeddings

Sentence Embeddings

Position Embeddings


[CLS]
my
dog
is
cute
[SEP]
he
MASK
play
##ing
[SEP]
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{MASK} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

All these embeddings are learned during the (pre)training process

In pre-training 15% of the input tokens are masked for the masked LM task

Attention mechanisms in Speech Recognition

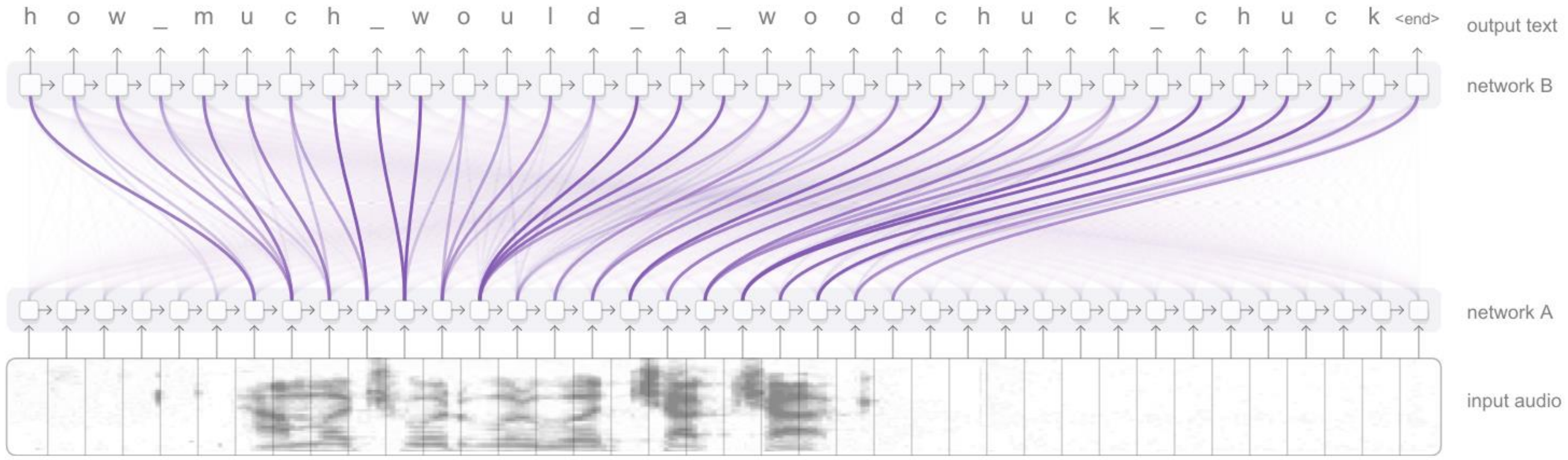
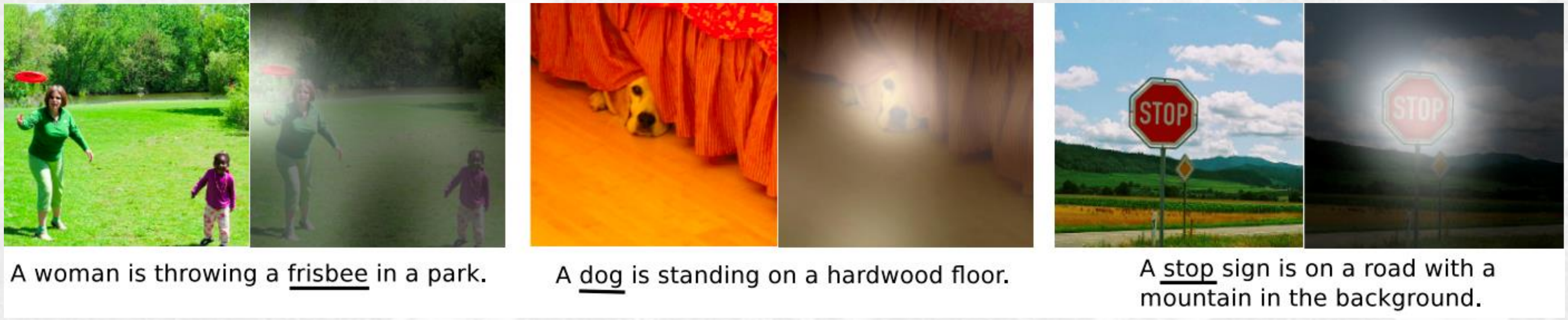

h o w _ m u c h _ w o u l d _ a _ w o o d c h u c k _ c h u c k <end>
output text
network B
network A
input audio

Figure derived from Chan, *et al.* 2015

https://arxiv.org/pdf/1508.01211.pdf

A complex application of LSTM (and recently Transformers): Image captioning



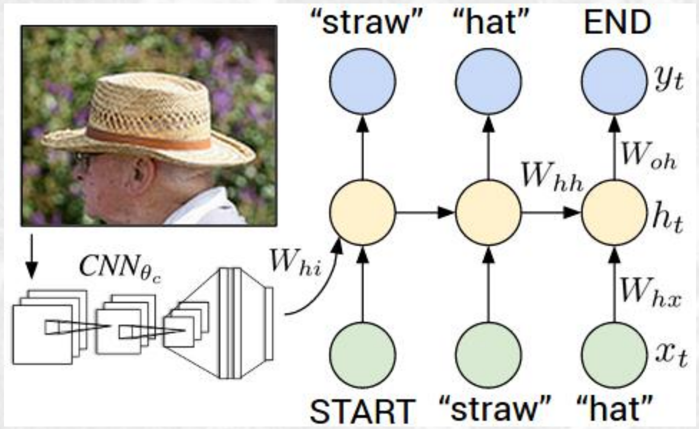
A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

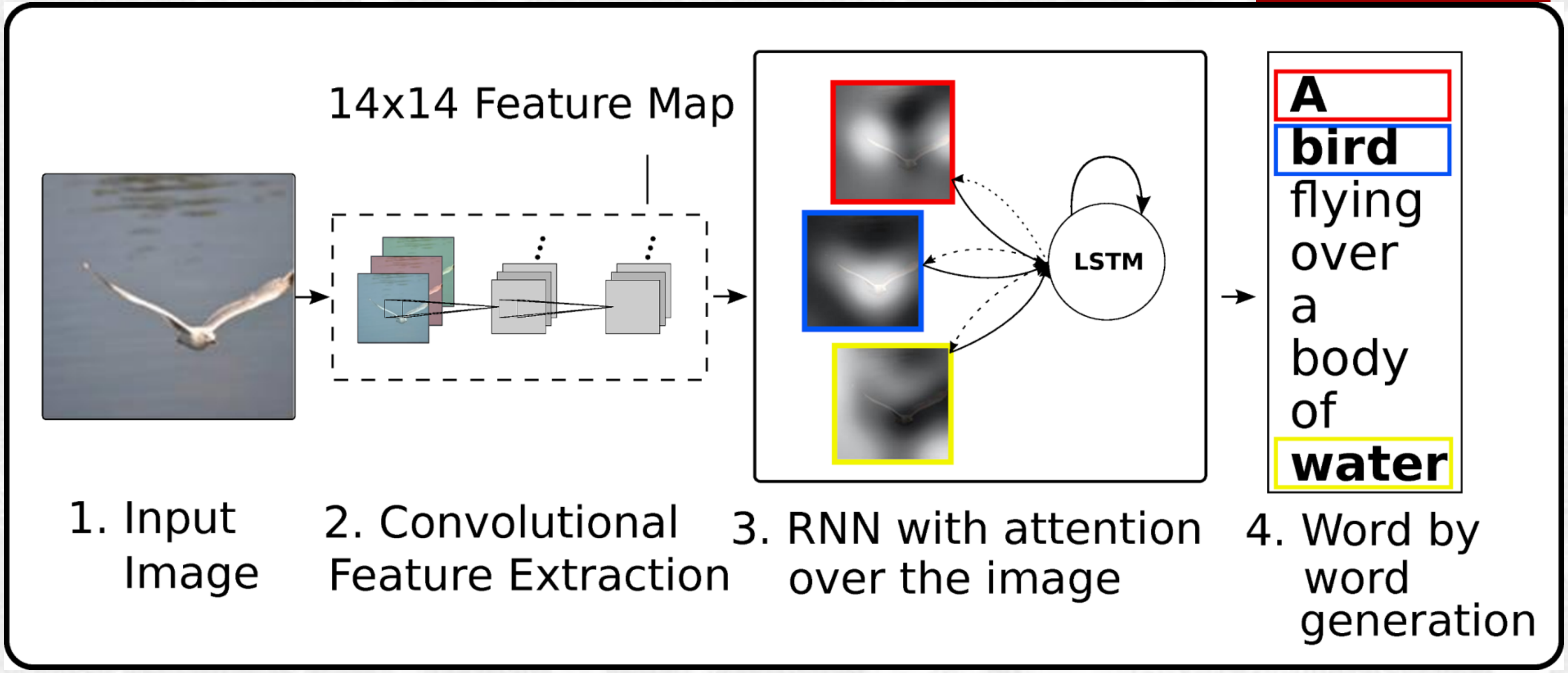
A stop sign is on a road with a mountain in the background.

Image Captioning

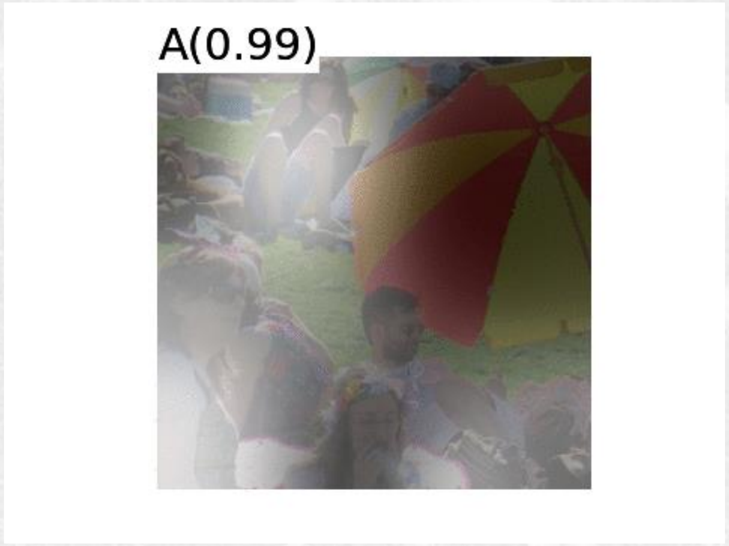
- Image to captions
  - Convolutional Neural Network to learn a representation of the image
  - (Bi-directional) Recurrent Neural Network to generate a caption describing the image
    - its input is the representation computed from the CNN
    - its output is a sequence of words, i.e. the caption

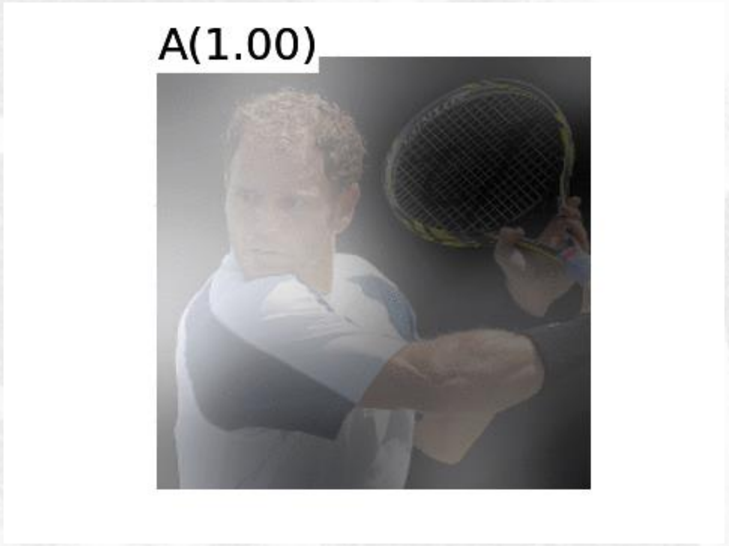


"baseball player is throwing ball in game."


14x14 Feature Map
LSTM
A
bird
flying
over
a
body
of
water
1. Input Image
2. Convolutional Feature Extraction
3. RNN with attention over the image
4. Word by word generation

Attention: a dynamic rendering





Perspectives

- Injecting bias (e.g. linguistic structures) within the learning architectures
- Making use of hybrid architectures integrating visual and linguistic knowledge
- Extending the epistemological transparency of current architectures: Explainable AI
- Making natural language data to work as a representation layer for different cognitive functions (e.g HRI in robotics but also vision)

RNNs - Bibliographic Notes & Further Readings

- Graves et al. (2009) demonstrate how recurrent neural networks are particularly effective at handwriting recognition,
- Graves et al. (2013) apply recurrent neural networks to speech.
- The form of gradient clipping presented above was proposed by Pascanu et al. (2013).
- Hochreiter and Schmidhuber (1997) is the seminal work on the "Long Short-term Memory" architecture for recurrent neural networks;
  - our explanation follows Graves and Schmidhuber (2005)'s formulation.
- Yoav Goldberg, A Primer on Neural Network Models for Natural Language Processing, Journal of Artificial Intelligence Research volume 57 pp 345-420, 2016
- Greff et al. (2015)'s paper "LSTM: A search space odyssey" explored a wide variety of variants and finds that:
  - none of them significantly outperformed the standard LSTM architecture; and
  - forget gates and the output activation function were the most critical components. Forget gates were added by Gers et al. (2000).

RNNs - Bibliographic Notes & Further Readings

- IRNNs were proposed by Le et al. (2015)
- Chung et al. (2014) proposed gated recurrent units
- Schuster and Paliwal (1997) proposed bidirectional recurrent neural networks
- Chen and Chaudhari (2004) used bi-directional networks for protein structure prediction; Graves et al. (2009) used them for handwriting recognition
- Cho et al. (2014) used encoder-decoder networks for machine translation, while Sutskever et al. (2014) proposed deep encoder-decoder networks and used them with massive quantities of data
- For further accounts of advances in deep learning and a more extensive history of the field, consult the reviews of LeCun et al. (2015), Bengio (2009), and Schmidhuber (2015)

Transformers

- (Vaswani 2017), Attention is all you need, https://arxiv.org/abs/1706.03762
- (Devlin et al 2018), BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, https://arxiv.org/abs/1810.04805
- Other Task specific works:
  - Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473, 2014.
  - Effective Approaches to Attention-based Neural Machine Translation, Minh-Thang Luong Hieu Pham Christopher D. Manning, 2015, https://arxiv.org/abs/1508.04025v5
  - Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured attention networks. In International Conference on Learning Representations, 2017.