Recurrent Neural Networks (RNN)

Sylvain Caillou

Sylvain.caillou@l2it.in2p3.fr

Overview

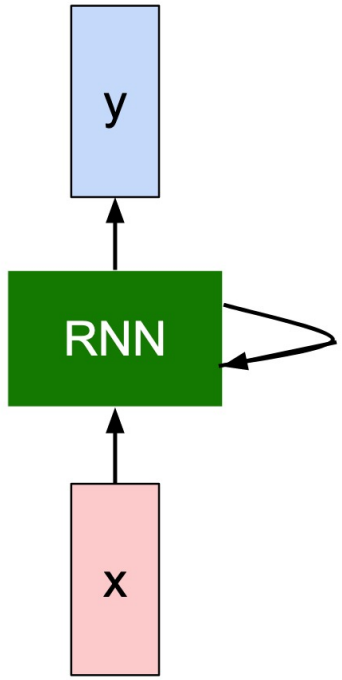
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

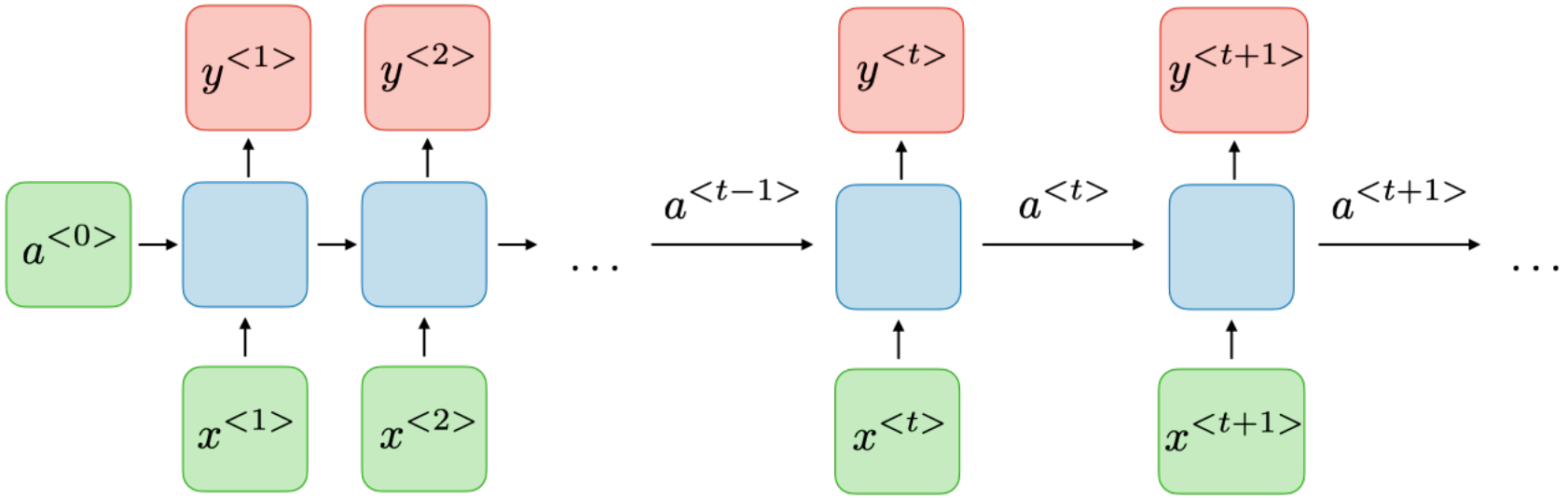
input vector at some time step



Notice: the same function and the same set of parameters are used at every time step.

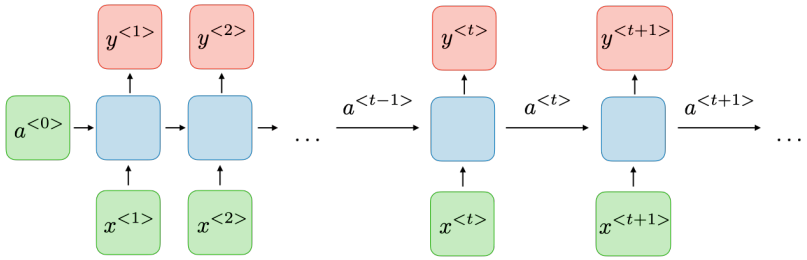
Overview

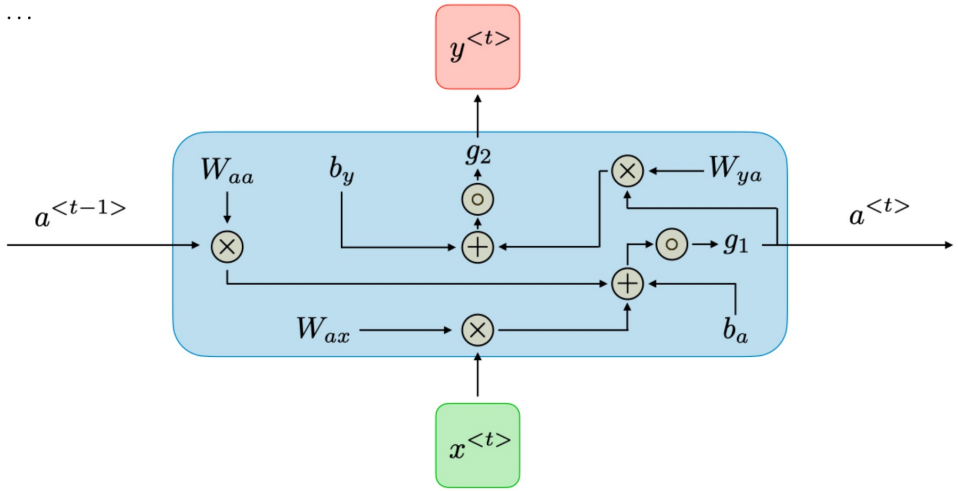
Architecture of a traditional RNN



Overview

Architecture of a traditional RNN





$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.

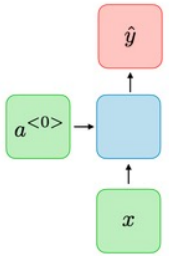
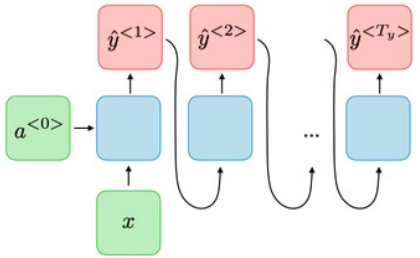
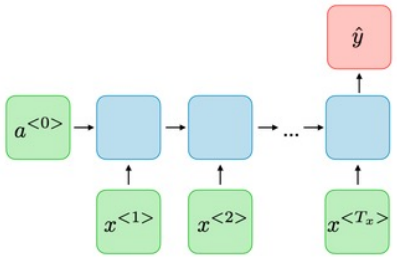
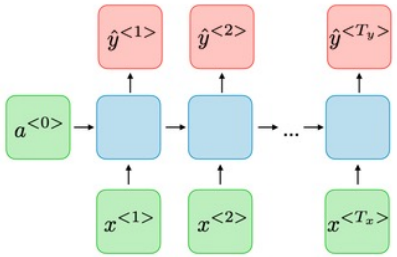
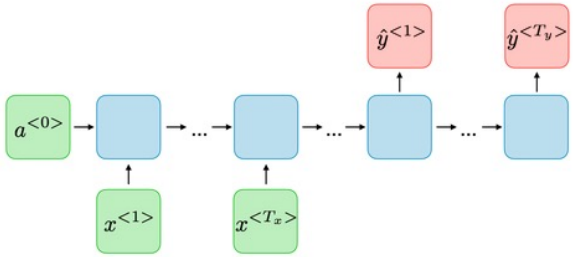
Overview

Architecture of a traditional RNN

- **Advantages**
  - RNN can process inputs of any length.
  - An RNN model is modeled to remember each information throughout the time which is very helpful in any time series predictor.
  - Even if the input size is larger, the model size does not increase.
  - The weights can be shared across the time steps.
- **Disadvantages**
  - Due to its recurrent nature, the computation is slow.
  - Training of RNN models can be difficult.
  - Difficulty of accessing information from a long time ago
  - Prone to problems such as exploding and gradient vanishing.

Overview

Applications of RNNs

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification
Many-to-many $T_x = T_y$		Name entity recognition
Many-to-many $T_x \neq T_y$		Machine translation

Overview

Loss function and Backpropagation

- Loss function In the case of a recurrent neural network, the loss function $\mathcal{L}$ of all time steps is defined based on the loss at every time step as follows:

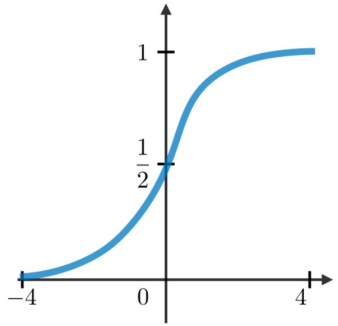
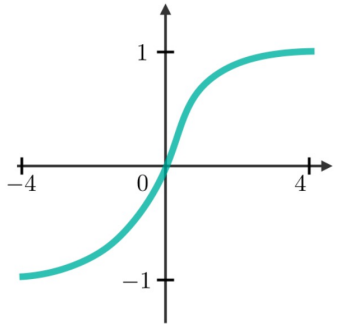
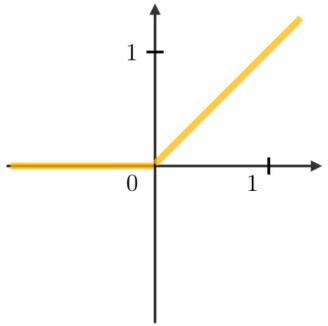
$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

- Backpropagation through time Backpropagation is done at each point in time. At timestep T, the derivative of the loss $\mathcal{L}$ with respect to weight matrix W is expressed as follows:

$$\frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^{T} \left. \frac{\partial \mathcal{L}^{(T)}}{\partial W} \right|_{(t)}$$

Handling long term dependencies

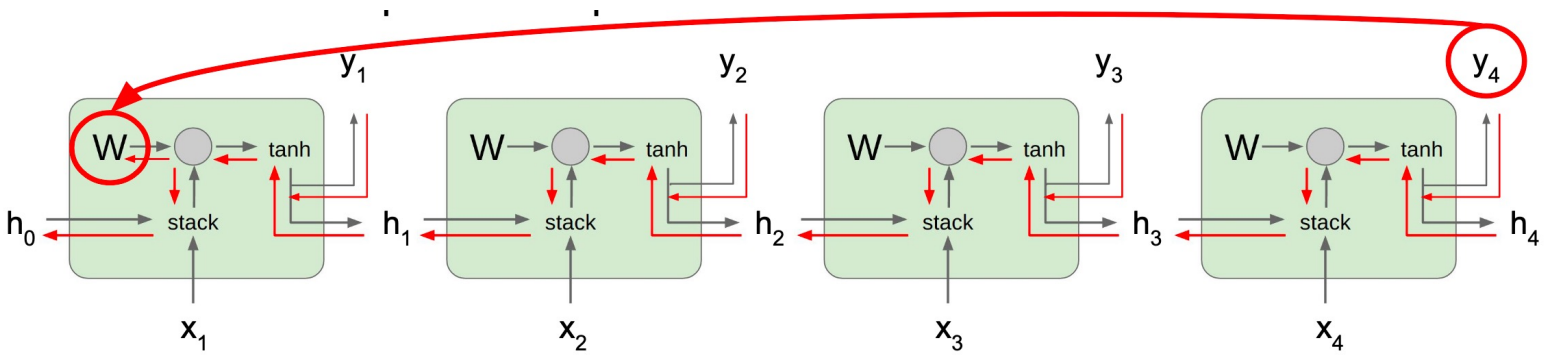
Commonly used activation functions

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z) = \max(0, z)$
		

Handling long term dependencies

Vanishing/exploding gradient

- Vanishing and exploding gradient phenomena are often encountered in the context of RNNs.
- It is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.



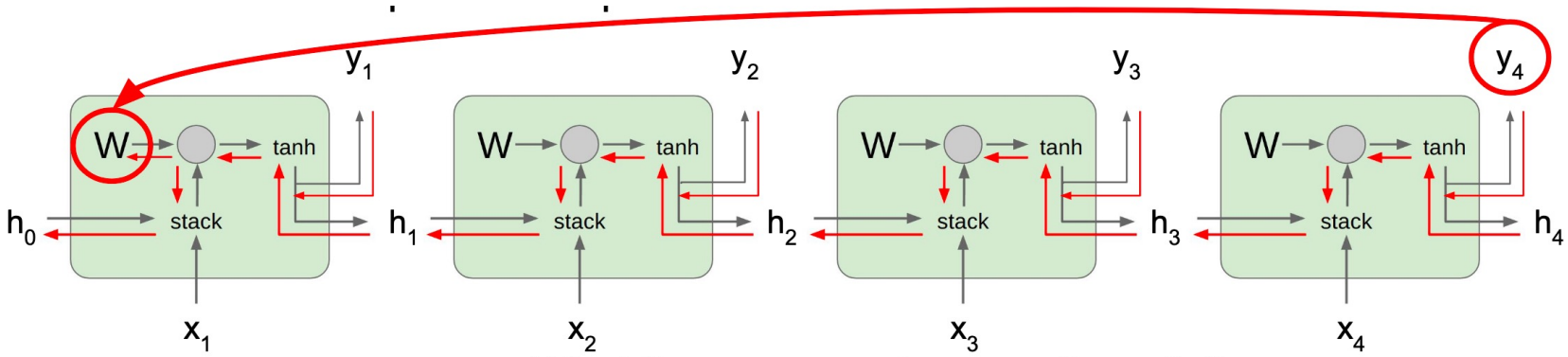
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}$$

Handling long term dependencies

Vanishing/exploding gradient

- Vanishing and exploding gradient phenomena are often encountered in the context of RNNs.
- It is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.



What if we assumed no non-linearity?

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

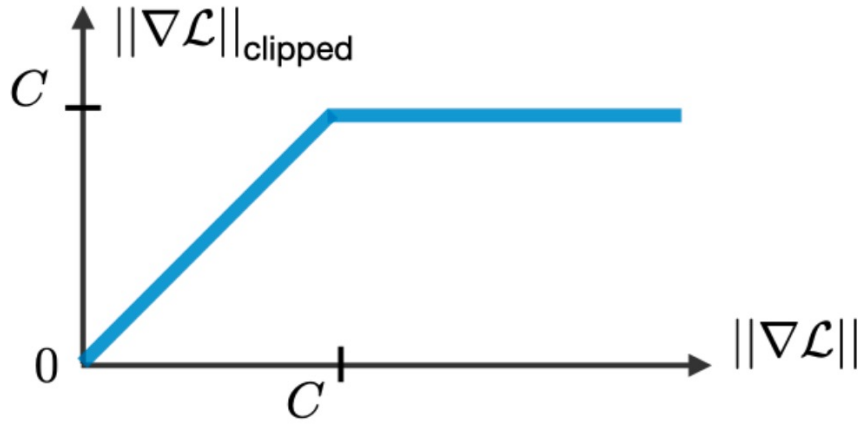
Largest singular value > 1:
Exploding gradients

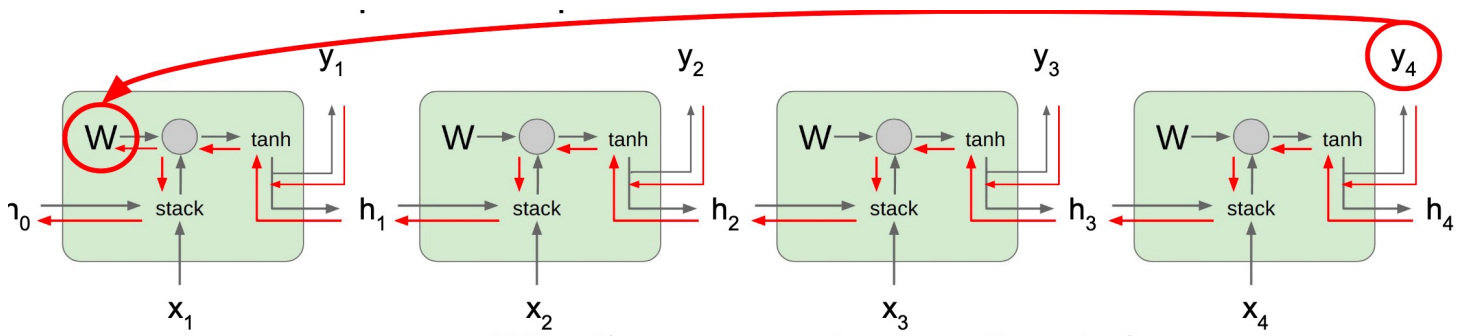
Largest singular value < 1:
Vanishing gradients

Handling long term dependencies

Gradient clipping

- Technique used to cope with the exploding gradient problem sometimes encountered when performing backpropagation.
- By capping the maximum value for the gradient, this phenomenon is controlled in practice.





What if we assumed no non-linearity?

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Largest singular value > 1: **Exploding gradients**

→ **Gradient clipping**: Scale gradient if its norm is too big

Largest singular value < 1: **Vanishing gradients**

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Handling long term dependencies

Types of gates

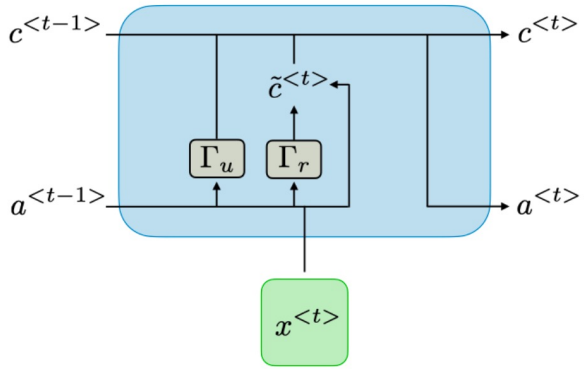
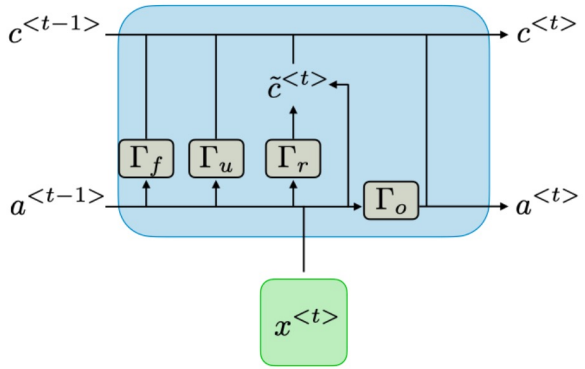
- In order to remedy the vanishing gradient problem, specific gates are used in some types of RNNs and usually have a well-defined purpose.

$$\boxed{\Gamma = \sigma(W x^{<t>} + U a^{<t-1>} + b)}$$

Type of gate	Role	Used in
Update gate Γ_u	How much past should matter now?	GRU, LSTM
Relevance gate Γ_r	Drop previous information?	GRU, LSTM
Forget gate Γ_f	Erase a cell or not?	LSTM
Output gate Γ_o	How much to reveal of a cell?	LSTM

Handling long term dependencies
GRU/LSTM

- Gated Recurrent Unit (GRU) and Long Short-Term Memory units (LSTM) deal with the vanishing gradient problem encountered by traditional RNNs
- LSTM being a generalization of GRU

Characterization	**Gated Recurrent Unit** (GRU)	**Long Short-Term Memory** (LSTM)
$\tilde{c}^{<t>}$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$
$c^{<t>}$	$\Gamma_u \star \tilde{c}^{<t>} + (1 - \Gamma_u) \star c^{<t-1>}$	$\Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$
$a^{<t>}$	$c^{<t>}$	$\Gamma_o \star c^{<t>}$
Dependencies		

Handling long term dependencies

LSTM

Vanilla RNN

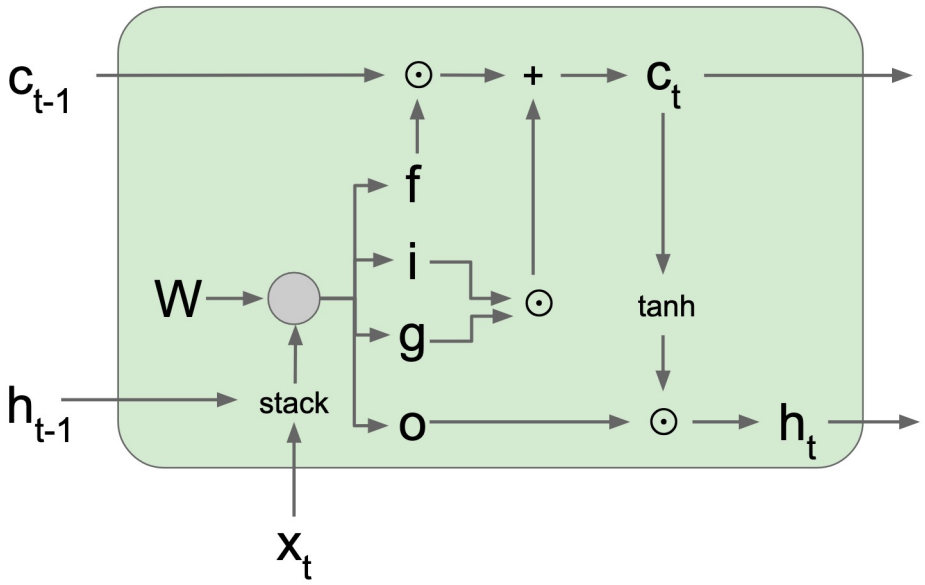
$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$



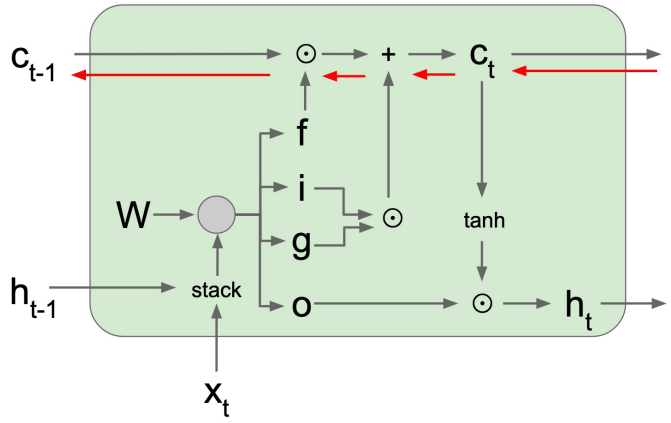
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

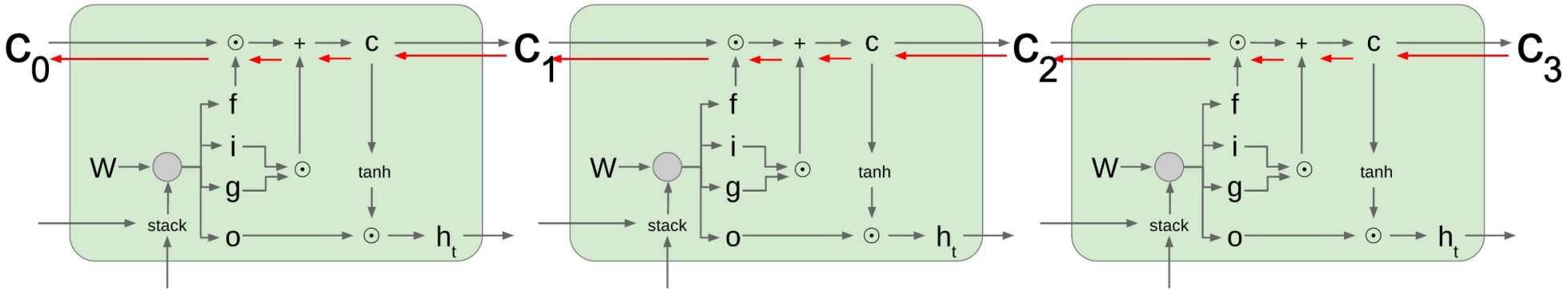
$$h_t = o \odot \tanh(c_t)$$

Handling long term dependencies

LSTM



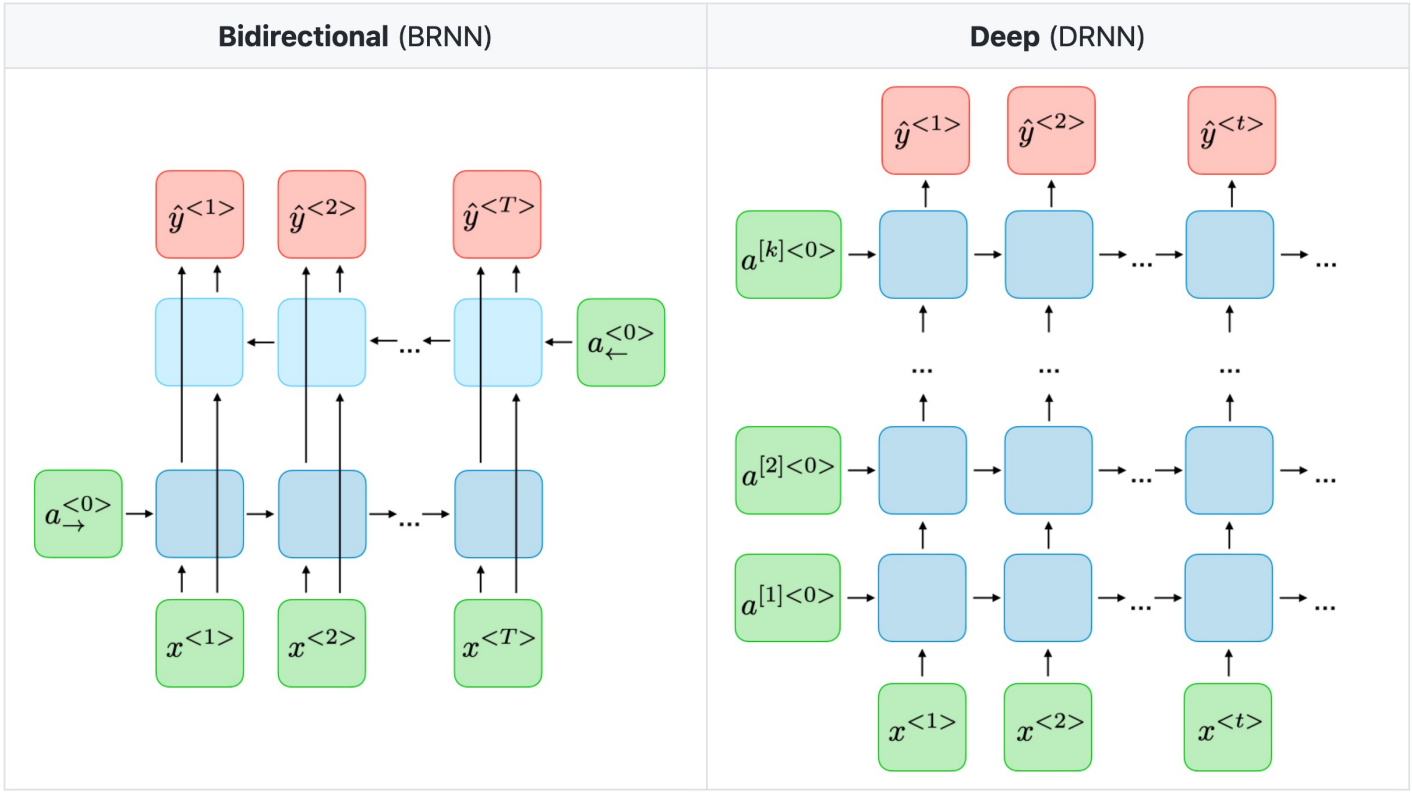
Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W



Handling long term dependencies

Variants of RNNs

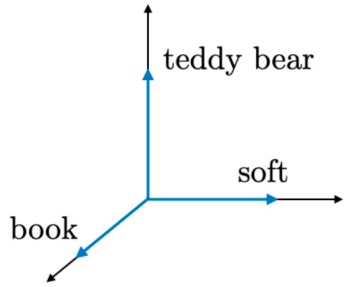
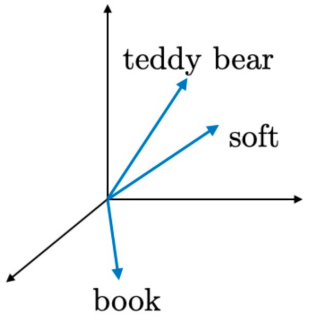
- Other commonly used RNN architectures



Learning word representation

Representation techniques

- The two main ways of representing words are summed up in the table below:

1-hot representation	Word embedding
	
• Noted o_w • Naive approach, no similarity information	• Noted e_w • Takes into account words similarity

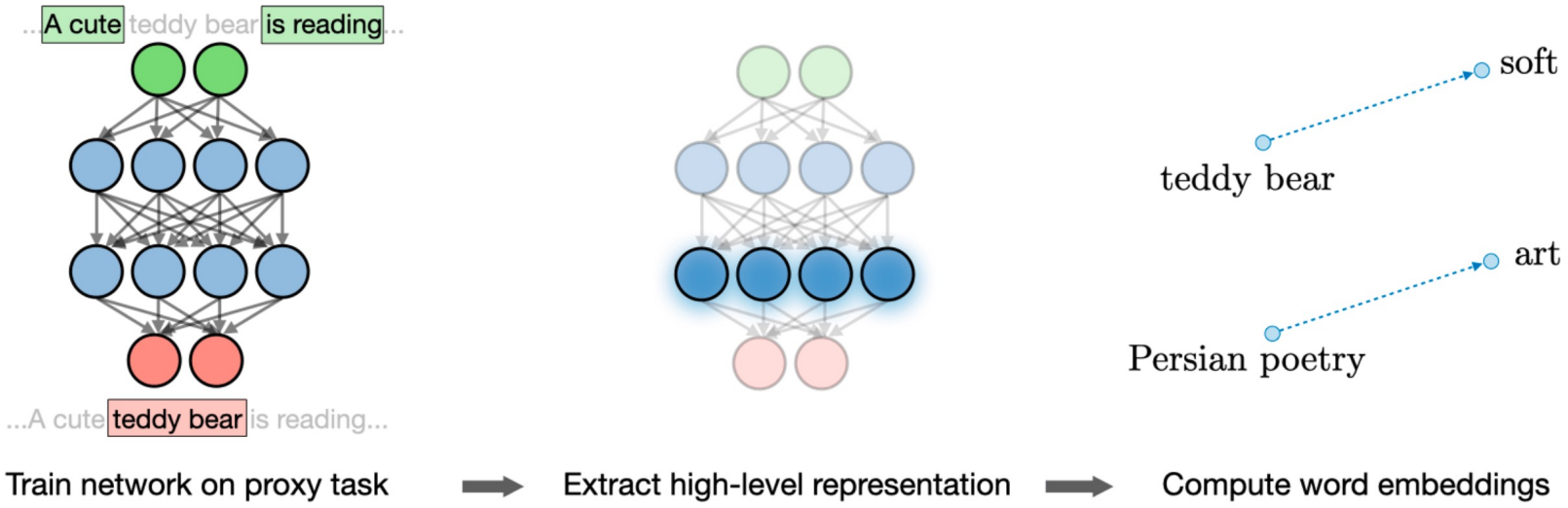
- For a given word w, the embedding matrix E is a matrix that maps its 1-hot representation o_w to its embedding e_w as follows:

$$e_w = Eo_w$$

Remark: learning the embedding matrix can be done using target/context likelihood models.

Parmi les premiers modèles permettant le plogement lexical (word embeddings) entrainné sur de larges corpus (millions de mots)

- Word2vec is a framework aimed at learning word embeddings by estimating the likelihood that a given word is surrounded by other words.
- First mode
- Popular models include skip-gram, negative sampling and CBOW.



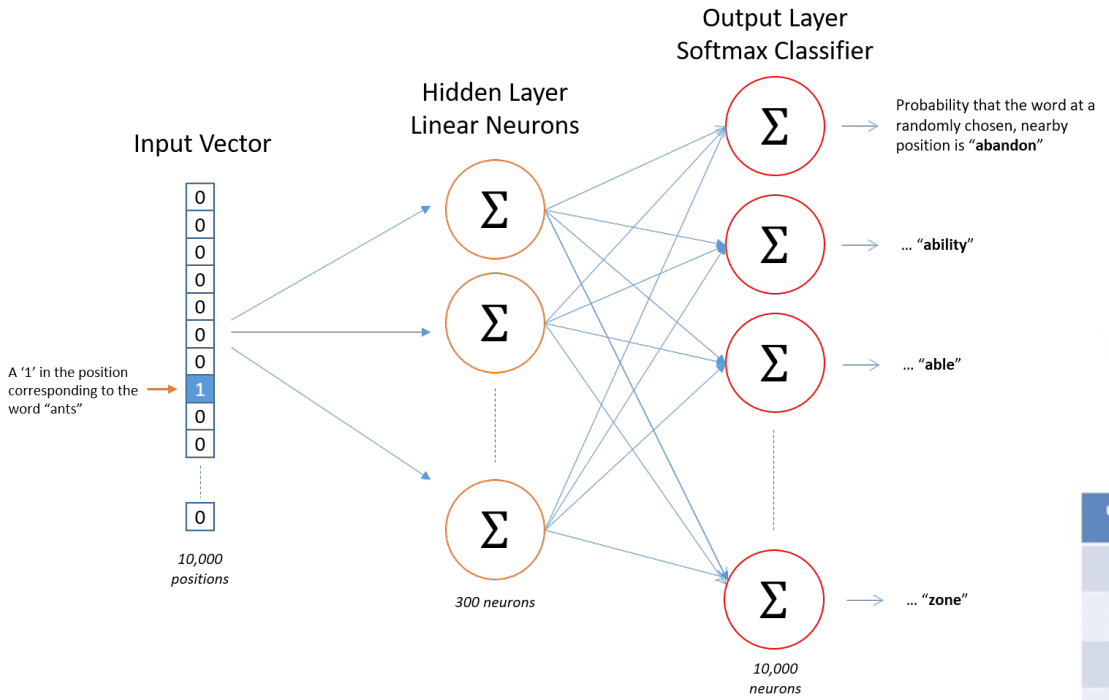
Learning word representation
Word embeddings : Skip-gram

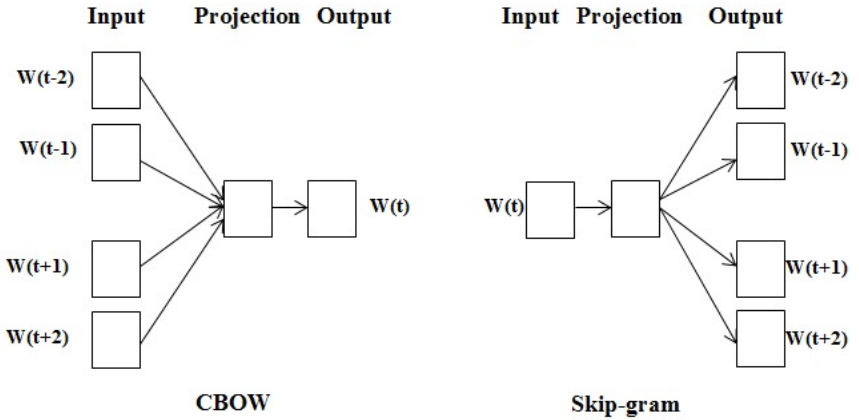
- The skip-gram word2vec model is a supervised learning task that learns word embeddings by assessing the likelihood of any given target word t happening with a context word c. By noting θ_t a parameter associated with t, the probability $P(t|c)$ is given by:

$$P(t|c) = \frac{\exp(\theta_t^T e_c)}{\sum_{j=1}^{|V|} \exp(\theta_j^T e_c)}$$

Remark: summing over the whole vocabulary in the denominator of the softmax part makes this model computationally expensive. CBOW is another word2vec model using the surrounding words to predict a given word.

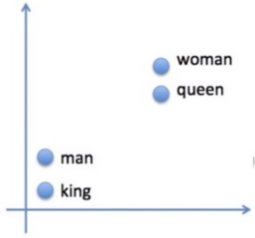
Learning word representation
Word embeddings : Word2vec

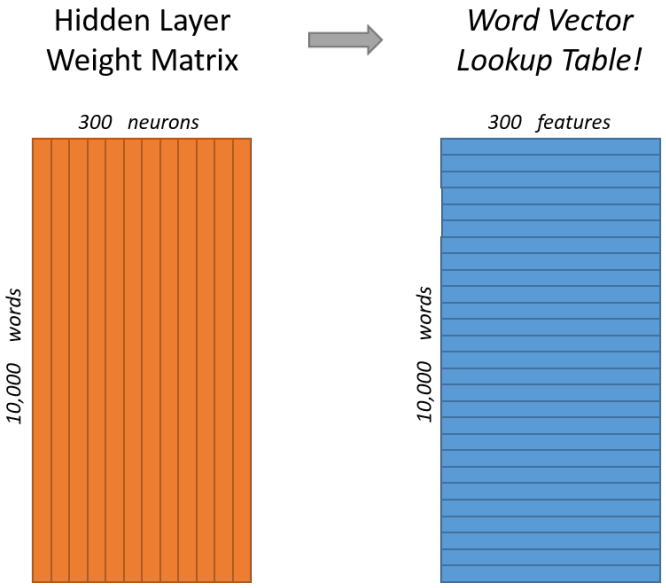




Word2Vec gives similarity in vector representation

unique word	encoding	word2vec embedding
king	[1, 0, 0, 0, 0, 0]	[1, 1]
man	[0, 0, 0, 1, 0, 0]	[1, 3]
queen	[0, 0, 0, 1, 0, 0]	[5, 5]
woman	[0, 0, 0, 0, 0, 1]	[5, 7]





Learning word representation

Word embeddings : Negative sampling

- Set of binary classifiers using logistic regressions that aim at assessing how a given context and a given target words are likely to appear simultaneously
- The models being trained on sets of k negative examples and 1 positive example.
- Given a context word c and a target word t, the prediction is expressed by:

$$P(y = 1|c, t) = \sigma(\theta_t^T e_c)$$

Learning word representation
Word embeddings : GloVe

- The GloVe model, short for global vectors for word representation, is a word embedding technique that uses a co-occurence matrix X where each $X_{i,j}$ denotes the number of times that a target i occurred with a context j.
- Its cost function j is as follows:

$$J(\theta) = \frac{1}{2}\sum_{i,j=1}^{|V|} f(X_{ij})(\theta_i^T e_j + b_i + b'_j - \log(X_{ij}))^2$$

$$e_w^{(\text{final})} = \frac{e_w + \theta_w}{2}$$

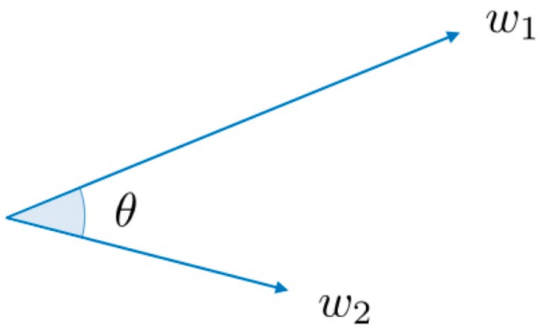
Learning word representation
Word embeddings : Word2vec

- Interrests:
  - Represents the distributional similarity between words
  - First models of embeddings trained on large corpus (millions of words)
- Limitations:
  - Multiple sense for word is not captured (ex « prison cells » vs « biological cells » vs « phone cells »)
  - Can not treat words if they were not in the trained vocabulary (need to re-train if new word)

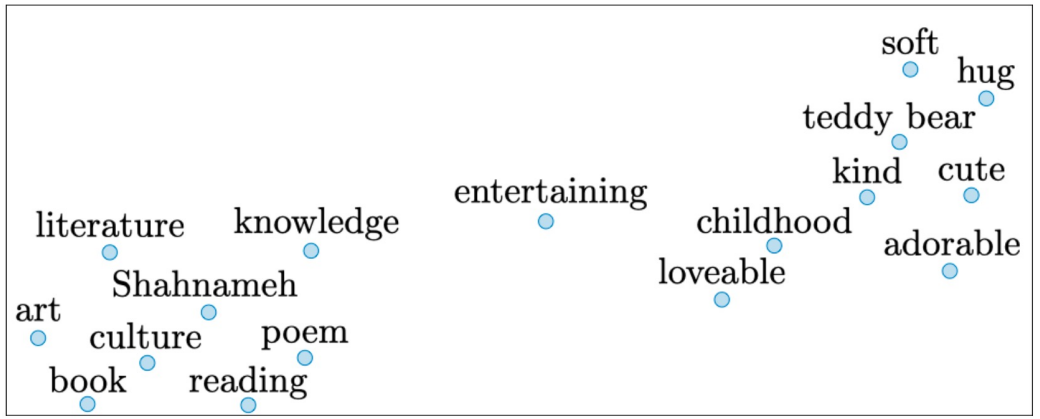
Comparing words

- Cosine similarity

$$\text{similarity} = \frac{w_1 \cdot w_2}{||w_1||\ ||w_2||} = \cos(\theta)$$



- t-SNE (ttt-distributed Stochastic Neighbor Embedding)
  - technique aimed at reducing high-dimensional embeddings into a lower dimensional space.
  - In practice, it is commonly used to visualize word vectors in the 2D space.

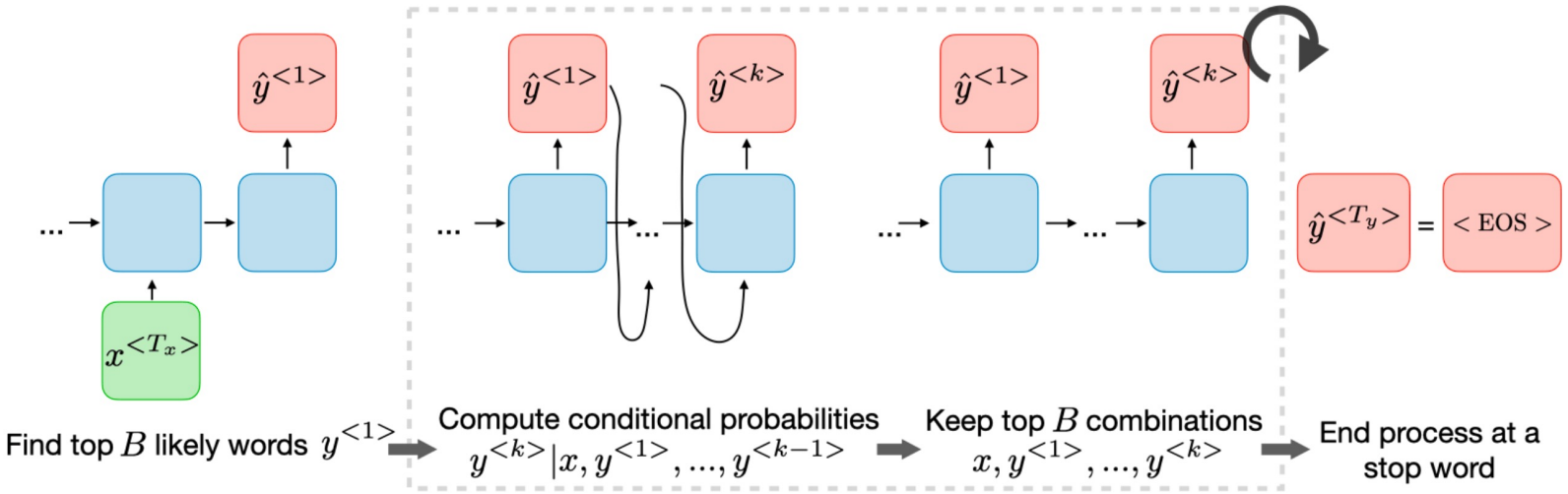


Language model

- A language model aims at estimating the probability of a sentence $P(y)$
- n-gram model naive approach aiming at quantifying the probability that an expression appears in a corpus by counting its number of appearance in the training data.
- Perplexity Language models are commonly assessed using the perplexity metric, also known as PP, which can be interpreted as the inverse probability of the dataset normalized by the number of words T.
- The perplexity is such that the lower, the better and is defined as follows:

$$\mathrm{PP} = \prod_{t=1}^{T} \left(\frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \hat{y}_j^{(t)}} \right)^{\frac{1}{T}}$$

Machine translation



The goal is to find a sentence y such that:

$$y = \underset{y^{<1>},...,y^{<T_y>}}{\arg\max}\ P(y^{<1>}, ..., y^{<T_y>}|x)$$

Attention

- This model allows an RNN to pay attention to specific parts of the input that is considered as being important, which improves the performance of the resulting model in practice.
- By noting $\alpha^{<t,t'>}$ the amount of attention that the output $y^{<t>}$ should pay to the activation $a^{<t'>}$ and $c^{<t>}$ the context at time t, we have:

$$\boxed{c^{<t>} = \sum_{t'} \alpha^{<t,t'>} a^{<t'>}} \quad \text{with} \quad \sum_{t'} \alpha^{<t,t'>} = 1$$

Remark: the attention scores are commonly used in image captioning and machine translation.

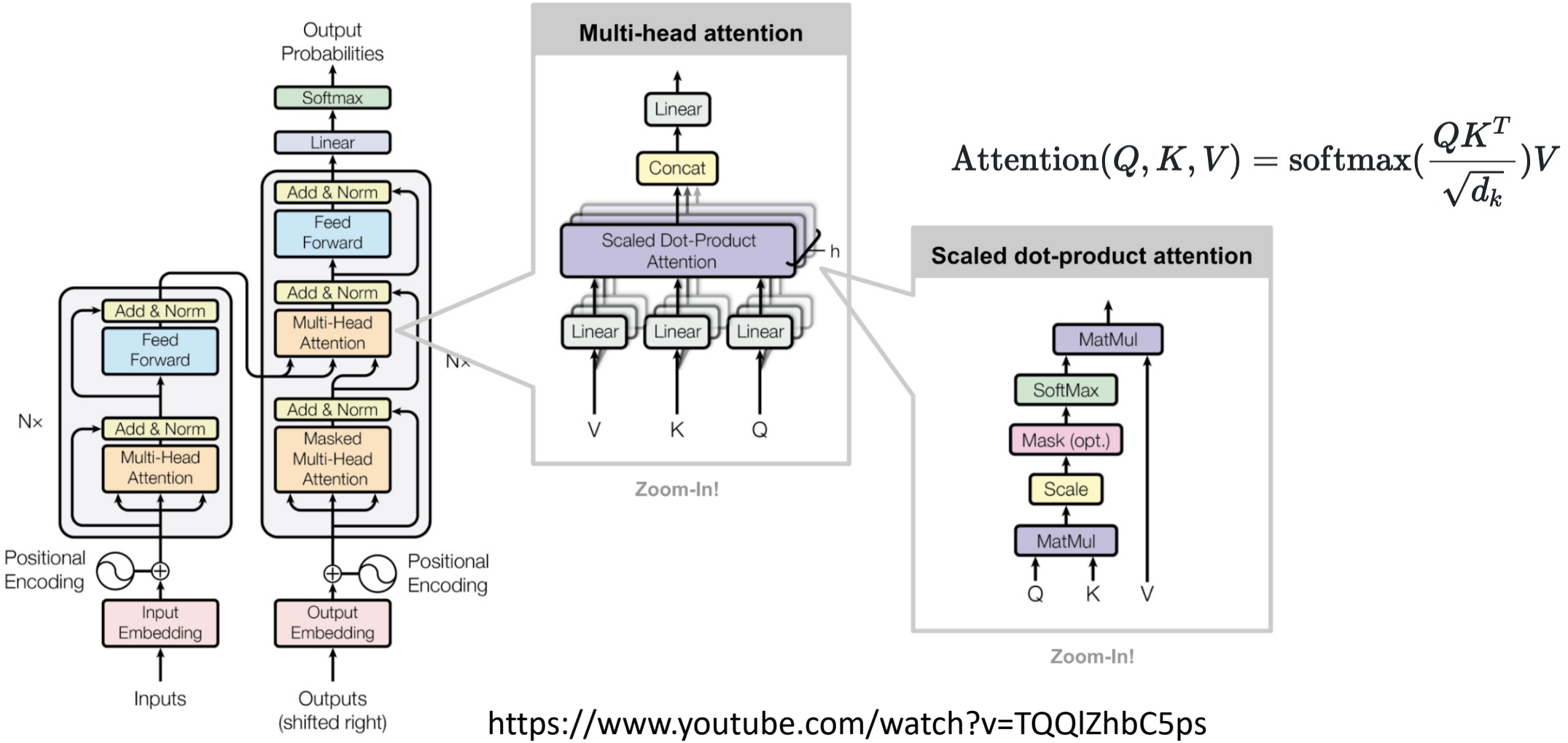
Attention

A cute teddy bear is reading Persian literature

A cute teddy bear is reading Persian literature

Attention

Transformer architecture



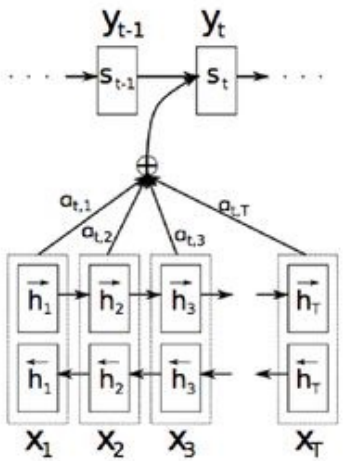
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

https://www.youtube.com/watch?v=TQQlZhbC5ps

Attention

Transformer architecture

Model Architecture



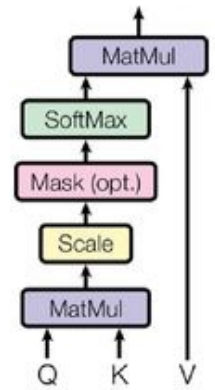
- **Attention**
- Attention in RNNsearch
  - Query(Q): S_{t-1}
  - Keys(K): $[h_1, h_2, ..., h_T]$
  - Values(V): $[h_1, h_2, ..., h_T]$
  - Attention(Q, K, V) = softmax(v*tanh(Q+K)) * V (*additive attention*)
- Scaled Dot-Product Attention (*dot-product attention*)

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

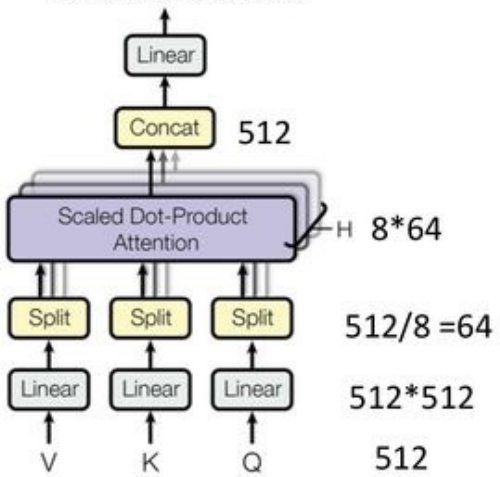
- Multi-Head Attention

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Scaled Dot-Product Attention



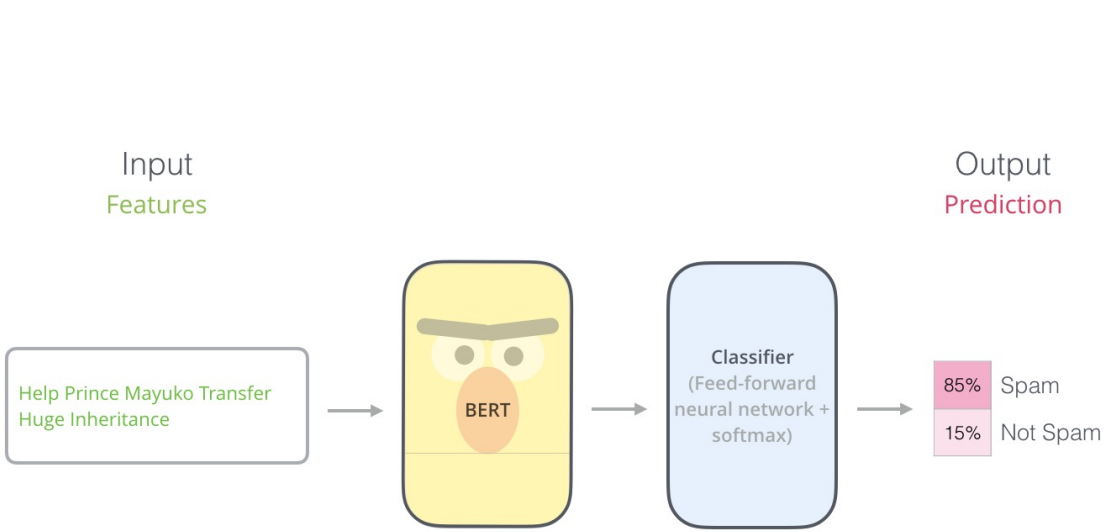
Multi-Head Attention

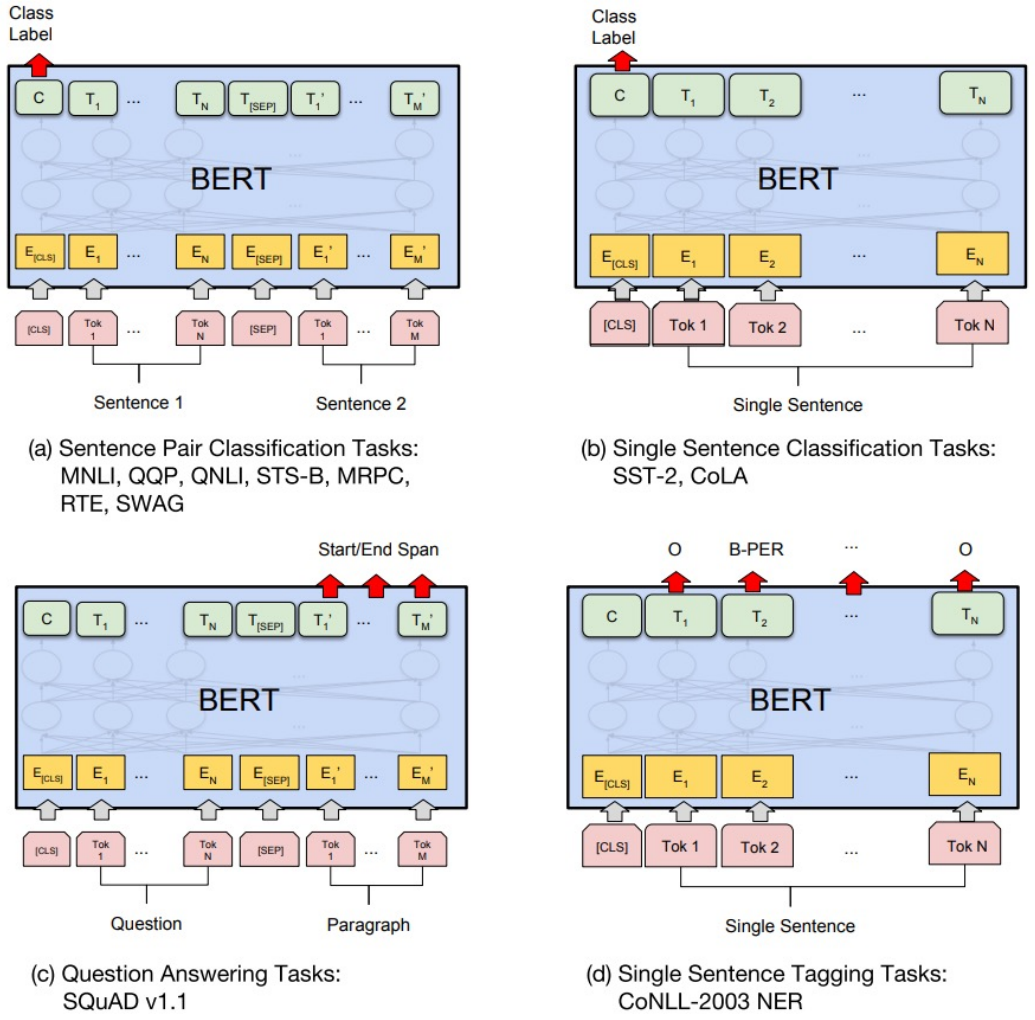


Deep Learning for NLP, a recent history

- 2013: word2Vec (« Distributed Representations of Words and Phrases and their Compositionality » T.Mikolov)
  - Representing words in the embedding layers of NN
  - Similarities between words
  - More efficient way to create word vector (no more one hot encoding)
- 2014 – 2015: RNNs and LSTMs (« The Unreasonable Effectiveness of Recurrent Neural Networks » A. Karpathy)
  - Processing of textual sequence data
  - LSTM / GRU: improvement of RNN for long sequence (remember what is important in the earlier inputs)
- 2015 – 2016: Attention based Networks
  - Focus on a specific subset of the data input
  - Network learns what is important to pay attention to
  - Breaking performance record for lot of NLP tasks (Machine Translation, Langage Modelling, Question Answering Problem)
- 2017: Transformer Model ("Attention is all you need" (2017) (A.Vaswani et al))
- 2018: BERT (Bidirectional Encoder Representations for Transfomers) (Google Research)
  - One the most advanced NLP models avalaibable
  - Context sensitive (different vectors to represent different contexts of the same word)
- 2019: XLNet and ERNIE

BERT (Bidirectional Encoder Representations for Transfomers)





Transfer learning with pretrained models