AIRI × Университет Сириус

Language modeling

Alsu Sagirova

Agenda

01 Language Models Evolution

02 Sequence-to-Sequence Modeling

03 Attention Intro

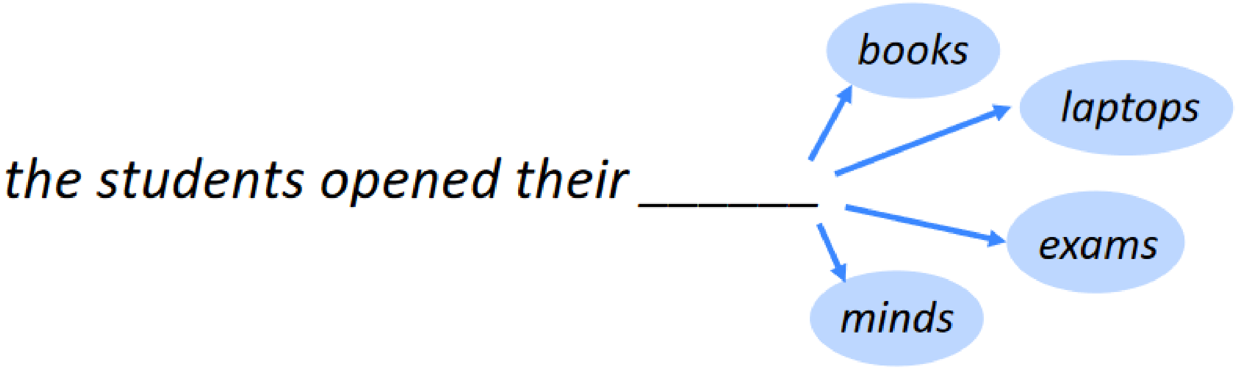
Slides & materials repo (feel free to zoom in the slides)

01

Language modeling

Language Modeling

Language Modeling is the task of predicting what word comes next

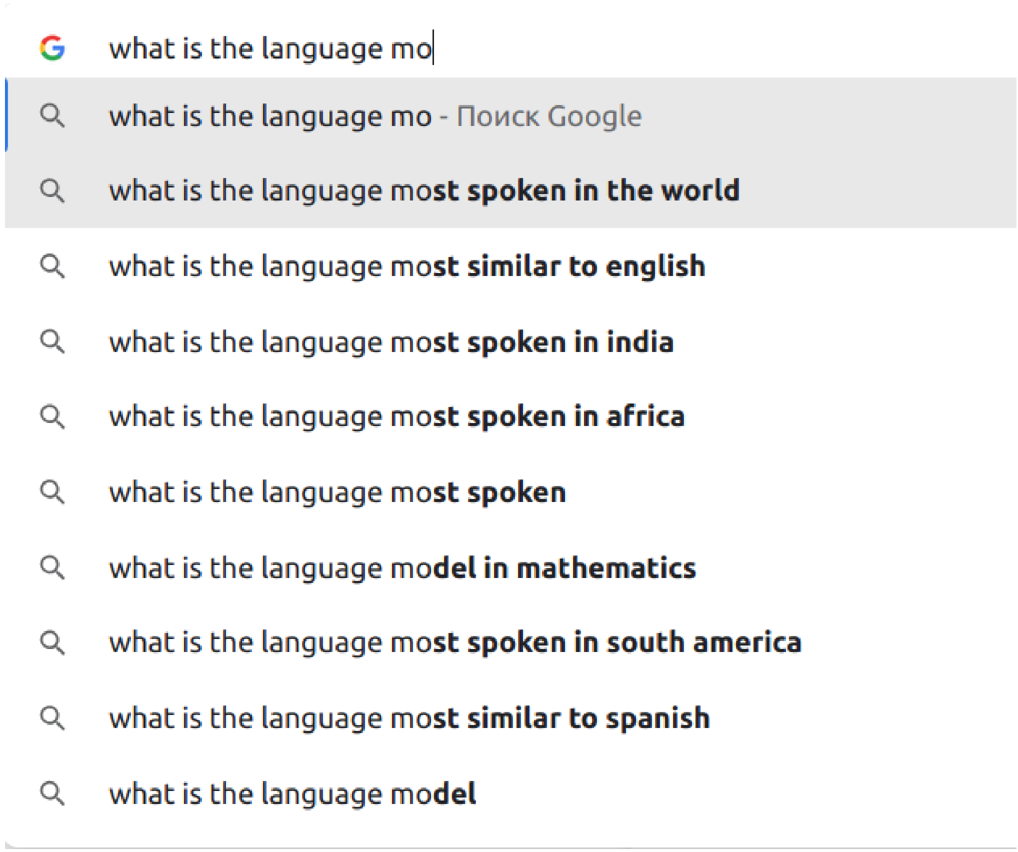


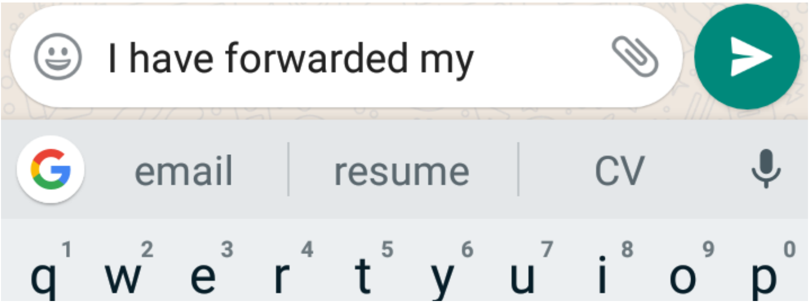
Formally: given a vocabulary $V = \{v_1, v_2, \dots v_{|V|}\}$ and a sequence of words/characters $x_1, x_2, x_3, \dots, x_t$ from the vocabulary, compute the probability distribution of the next word/character x_{t+1}:

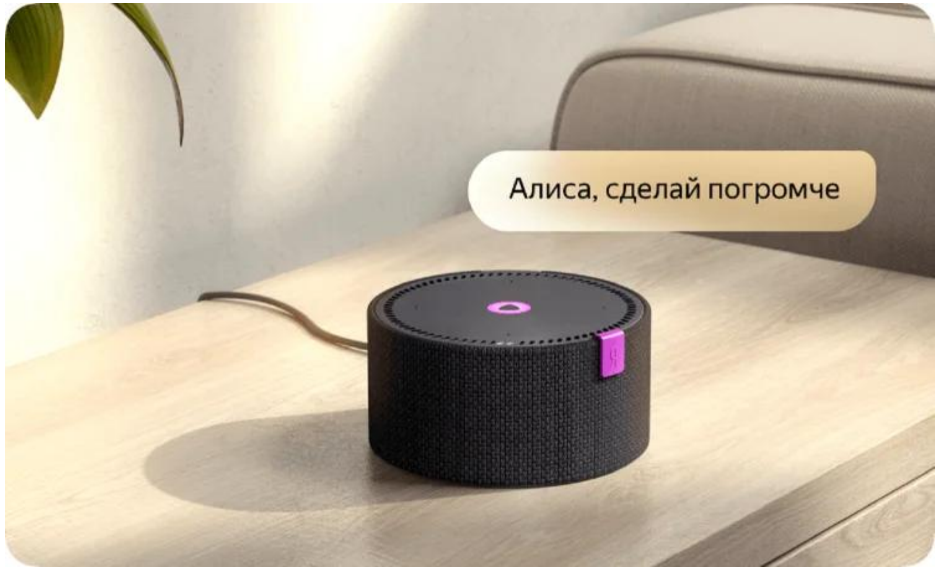
$$P(x_{t+1} | x_1, \dots, x_t).$$

Such predicting system is called a ***Language Model***

Language Models are everywhere



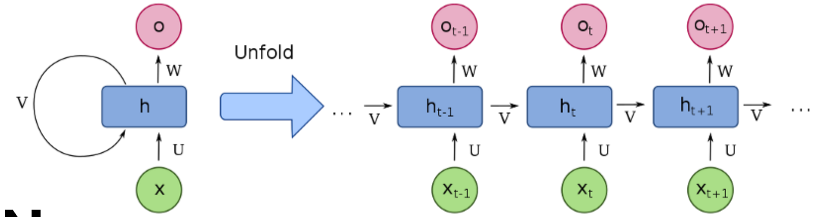




Language Models Evolution

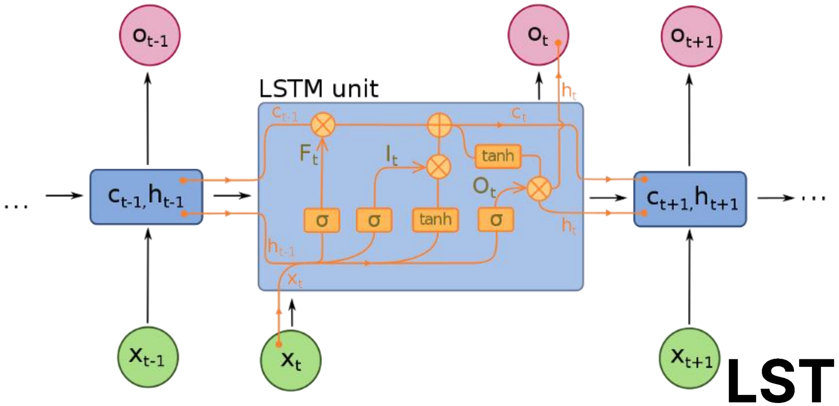
The neural architecture that takes the input sequence of any length and outputs the probability distribution for the next element of the sequence (e.g. word):

N-gram models 1940s



1980-1990 **RNN**

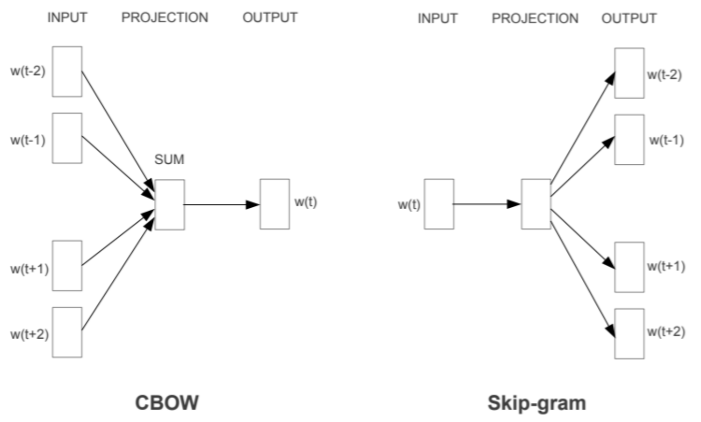
- In 1982 John Hopfield introduced the RNN to be used for operations on sequence data (text or voice).
- By 1986, Geoffrey Hinton presented first ideas of representing words as vectors (Hinton et al.,1986 Rumelhart et al., 1986).



LSTM 1997-2000

- Hochreiter and Schmidhuber, 1997
- Gers, Schmidhuber and Cummins, 2000

Language Models Evolution



Word2Vec & RNN/LSTM/GRU for NLP

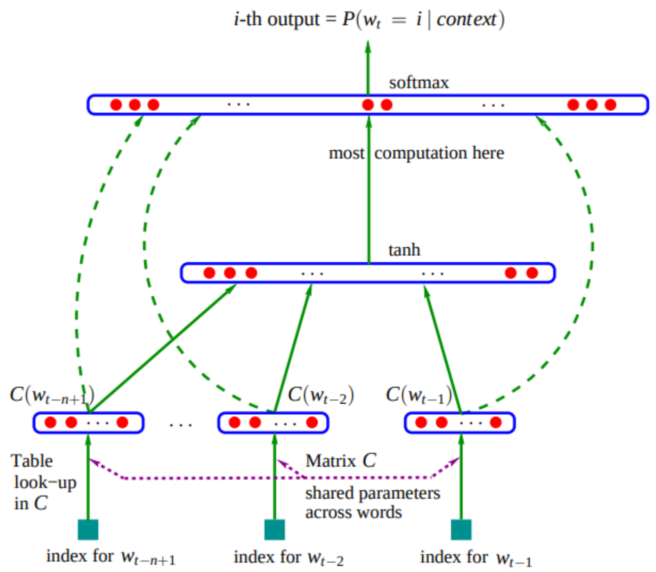
2013

[Mikolov et al., 2013](#) introduced Continuous bag-of-words and Skip-Gram to learn high-quality word embeddings that were transferable across NLP applications.

2003

First feed-forward neural network LM

[Bengio et al., 2003](#) firstly applied the Word Embedding concept.



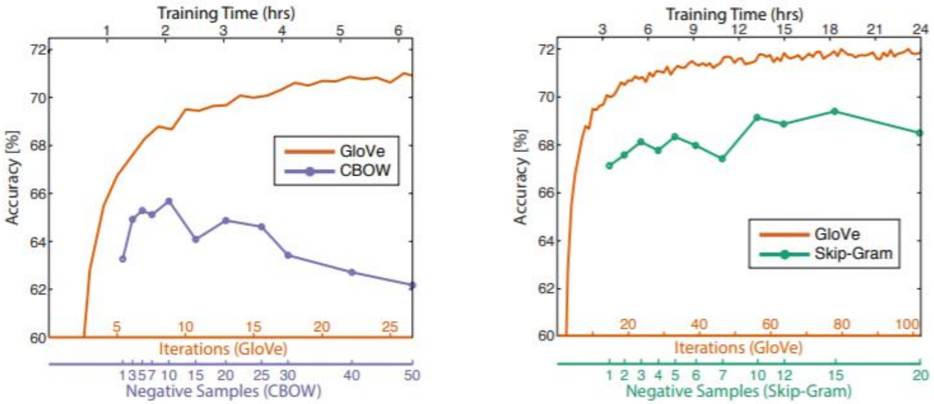
AIRI × Университет Сириус

Language Models Evolution

2014

GloVe

Pennington et al., 2014 found word embeddings could be learned by co-occurrence matrices and proved GloVe outperforms Word2Vec on word similarity tasks and NER.

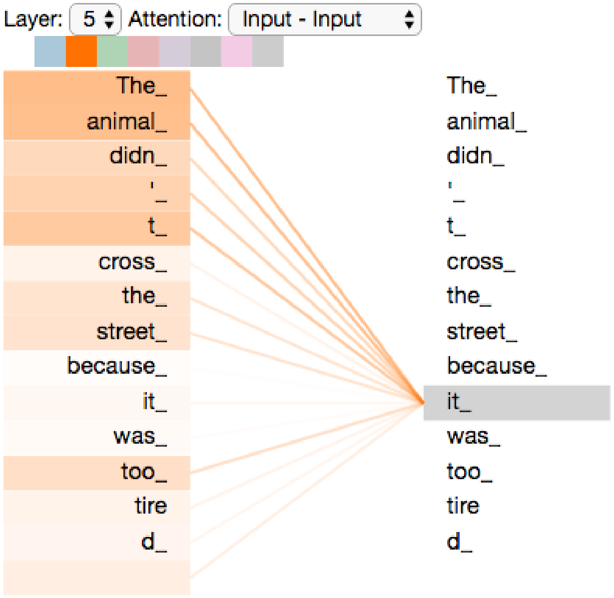


(a) GloVe vs CBOW (b) GloVe vs Skip-Gram

2015

Attention Models

Bahdanau et al., 2015 proposed the attention mechanism to address the Neural Machine Translation with RNNs long-term dependencies problem.



2016

From NMT to Image Captioning with Attention

Attention begins to be widely used for NLP applications.

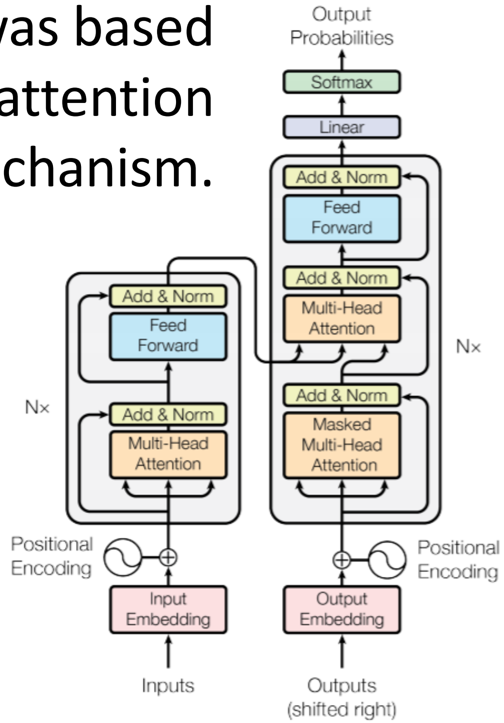
A dog is standing on a hardwood floor.

AIRI × Университет Сириус

Language Models Evolution

Transformer 2017

Vaswani et al., 2017 proposed a new simple network architecture known as the Transformer that was based solely on the attention mechanism.



2018-Present

Fun fact: BERT is currently used by the Google Search Engine.

Pretrained Language Models

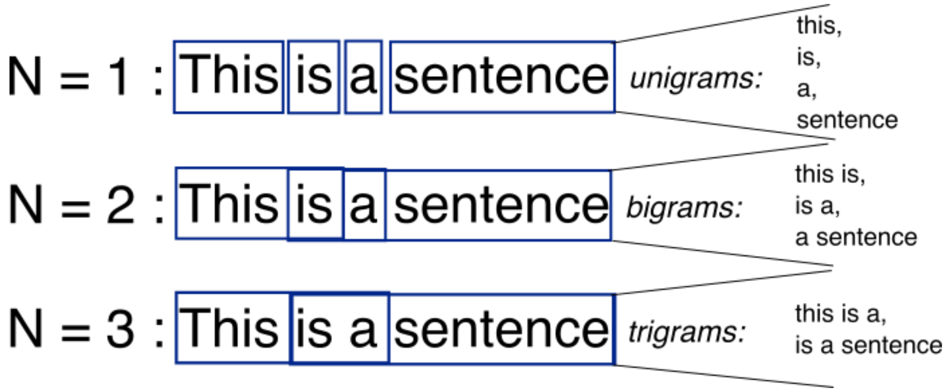
Pretrained Word Embeddings had evolved to Pretrained Language Models.

- Devlin et al., 2018 BERT: Bidirectional Encoder Representations from Transformers
- Lan et al., 2019 ALBERT: A Lite BERT for Self-supervised Learning of Language Representations
- Liu et al., 2019 RoBERTa: Robustly Optimized BERT Pretraining Approach
- Peters et al., 2018 ELMo : Embeddings from Language Models
- Sun et al., 2019 ERNIE: Enhanced Representation through Knowledge Integration
- Radford et al., 2019 GPT-2: Language Models are Unsupervised Multitask Learners
- Raffel et al., 2019: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

First Language Models

The basic example of a language model is the ***n-gram model****

An n-gram is a chunk of n consequtive elements of a sequence from a given sample of text.



Model Markov assumption:

$$P(x_{t+1}|x_1, x_2, \dots, x_t) = P(x_{t+1}|\underbrace{x_{t-n+2}, \dots, x_t}_{\text{n-1 elements}}) = \frac{P(x_{t+1}, x_{t-n+2}, \dots, x_t)}{P(x_{t-n+2}, \dots, x_t)},$$

where the probabilities are counts of given sequences in some large corpus.

* The first references of N-grams came from Claude Shannon's paper "A Mathematical Theory of Communications" (1948).

N-gram model disadvantages

Consider 4-gram modelss

~~This is a sentence~~ expalaining the model ***x***

$$P(x|explain\ the\ model) = \frac{count(explain\ the\ model\ x)}{count(explain\ the\ model)}$$

- Sparsity Problems
  - ***expalaining the model x*** never occurred in data so **x** has probability 0!
    Do *smoothing* by adding small δ to the count for every **x** in the vocabulary
  - ***expalaining the model*** never occurred in data so we cannot calculate the probability for any **x**!
    Do *backoff* by conditioning on ***the model*** instead.
  - increasing n makes sparsity problems worse. Typically, we cannot have n bigger than 5.
- Storage Problems
  - need to store counts for all n-grams occured in the corpus
  - increasing n or increasing corpus increases model size!

Neural network with fixed window

A neural LM with fixed window

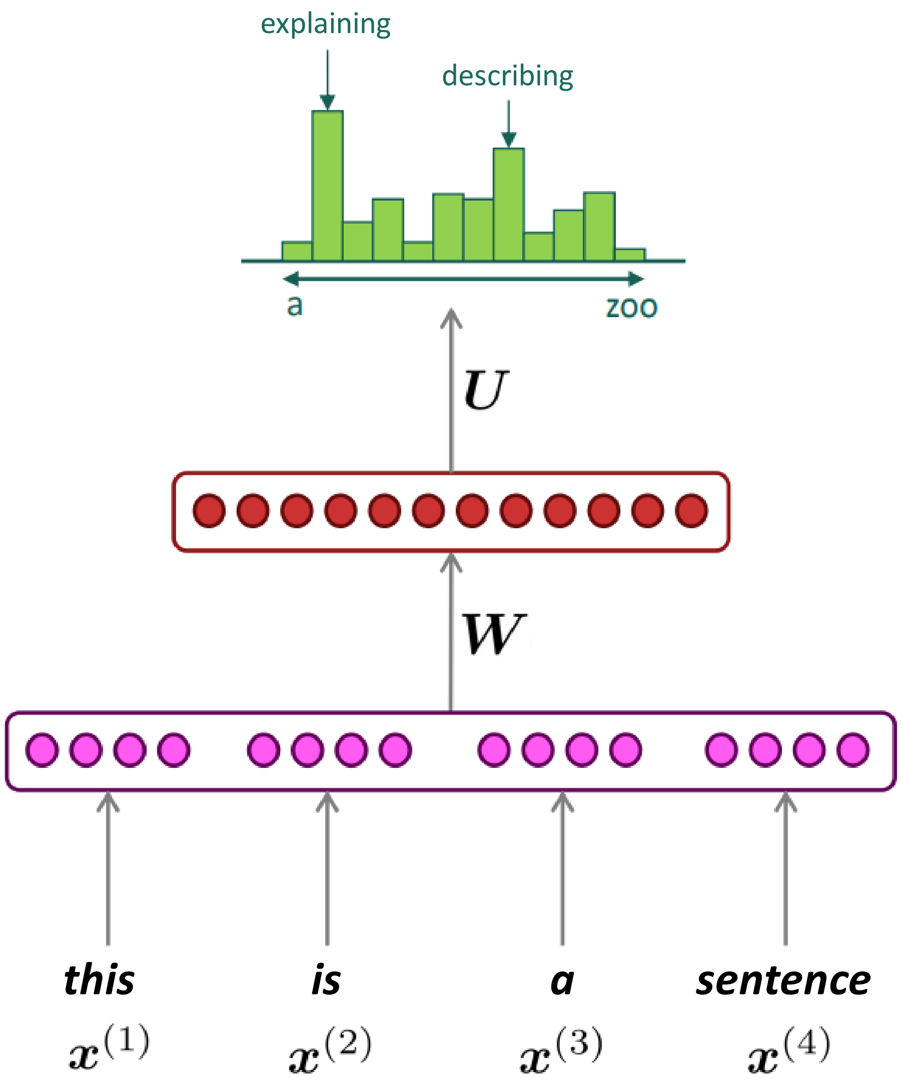
Pros:

- No sparsity problem
- Don’t need to store all observed n-grams

Cons:

- Fixed window is *too small*
- Enlarging window enlarges the model size
- Window will never be large enough!
- Different parts of input are miltiplied by different model weights

Solution: **RNN**



RNN

Pros:
- The same weights applied repeatedly
- Input sequence can have any length
- Computation for step t can (in theory) use information from many steps back
- Model size doesn’t increase for longer input context

Cons:
- Recurrent computation is slow
- In practice, difficult to access information from many steps back
- Vanishing & exploding gradients

output word representations

$$y^{(t)} = soft^{m} \quad (Uh^{(t)} + b_2)$$

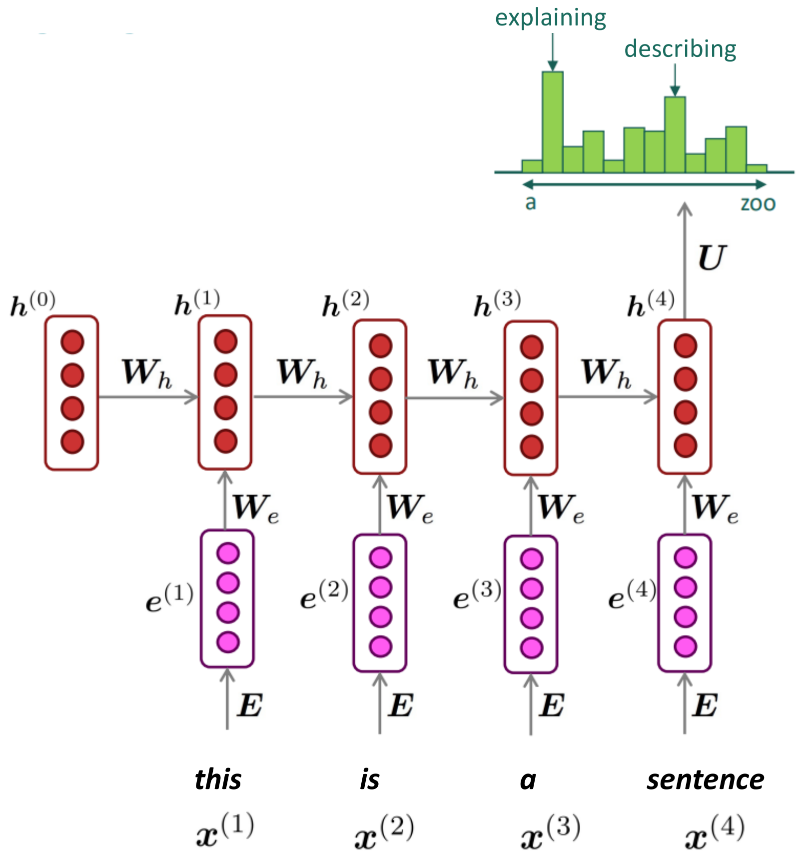
hidden states

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_e e^{(t)} + b_1\right)$$

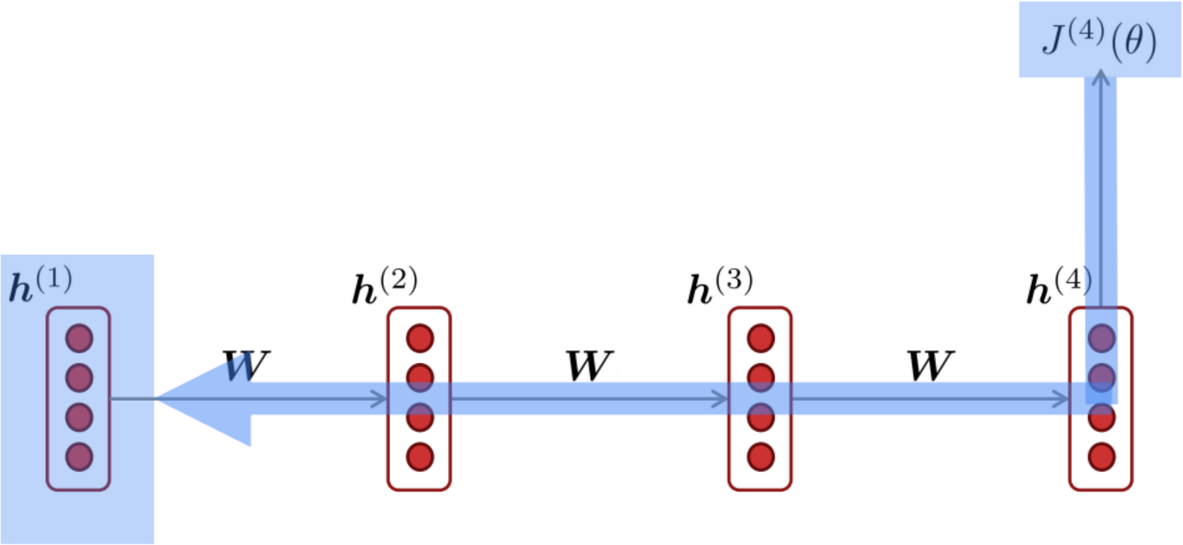
input embeddings

$$e^{(t)} = Ex^{(t)}$$

words/tokens $x^{(t)}$



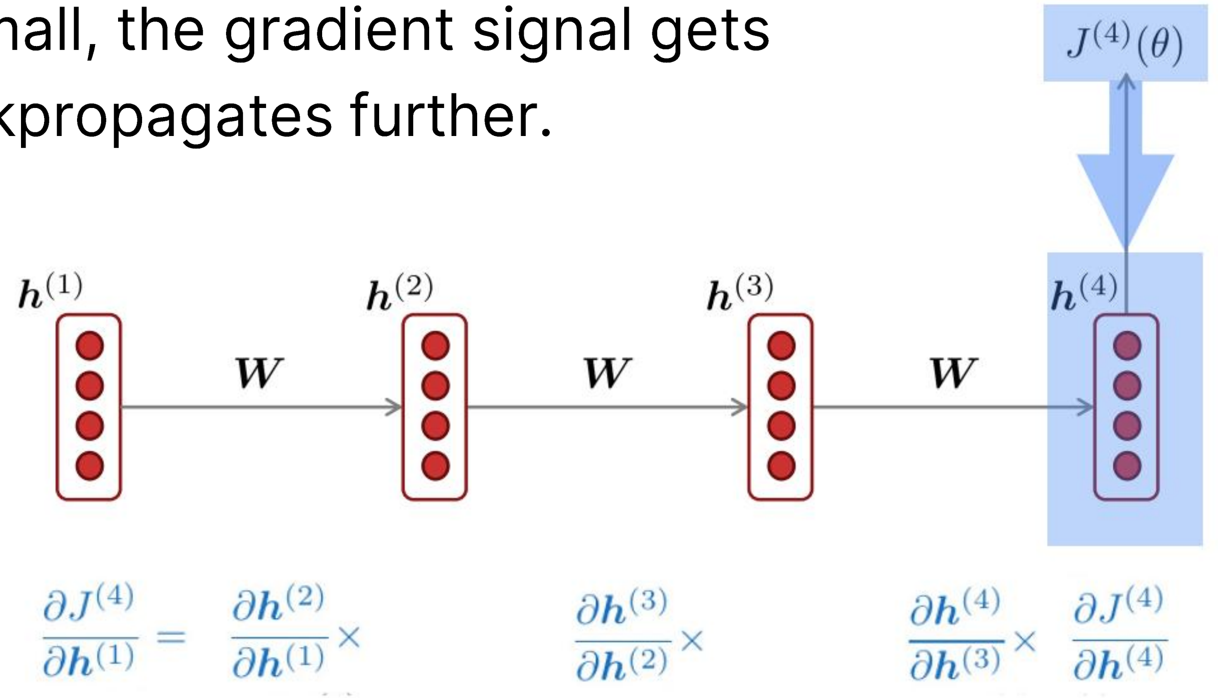
Vanishing gradients



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \; ?$$

Vanishing gradients

When $\frac{\partial \boldsymbol{h}^{(j)}}{\partial \boldsymbol{h}^{(k)}}$ are small, the gradient signal gets smaller as it backpropagates further.



So gradient signal from far away is lost because it's much smaller than gradient signal from close-by. So, model weights are updated only with respect to near effects.

The model can't learn long-term dependencies during training so it is unable to predict long-distance dependencies at test time.

Vanishing gradients: proof scetch

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_e e^{(t)} + b_1\right)$$

Let's consider the linear activation $\sigma(x) = x$.

$$\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \operatorname{diag}\left(\sigma\left(W_h h^{(t-1)} + W_e e^{(t)} + b_1\right)\right) W_h = I W_h = W_h$$

So the loss on step *i* w.r.t. The hidden state on step *j*<*i*

$$\frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \prod_{j<t\leq i} \frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \prod_{j<t\leq i} W_h$$

For small W_h values, the product gets exponentially problematic while the distance between *i* and *j* grows.

Source: [“On the difficulty of training recurrent neural networks”, Pascanu et al, 2013]() and supplemental [materials]()

AIRI × Университет Сириус

Vanishing gradients: proof scetch

Consider $\prod_{j<t\leq i} W_h$.

If the eigenvalues of W_h are all less than 1:

$\frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} \prod_{j<t\leq i} W_h = \sum_{i=1,\dots,n} c_i {\lambda_i}^{i-j} \boldsymbol{q}_i \approx 0$ for large distances between i and j so the gradient vanishes.

What about nonlinear activations (i.e., what we use?)

Vanishing gradients: proof scetch

Consider $\prod_{j<t\leq i} W_h$.
If the eigenvalues of W_h are all less than 1:
$\frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} \prod_{j<t\leq i} W_h = \sum_{i=1,\ldots,n} c_i {\lambda_i}^{i-j} \boldsymbol{q}_i \approx 0$ for large distances between i and j so the gradient vanishes.

What about nonlinear activations (i.e., what we use?)
- The same except the moment that the proof requires eigenvalues to be smaller than some fixed γ value specific to the dimensionality and activation choice.

Exploding gradients

If the gradient grows large, then the SGD update step grows:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

So if we take too large step size, we reach a bad parameter configuration with large loss value.
In the worst case, this will result in Inf or NaN loss values, leading to the training restart.

Solution: **Gradient Clipping -** if the gradient norm is bigger than given threshold, we scale the gradient down before applying SGD update. So we take a smaller step in the chosen direction.

LSTM

In a vanilla RNN, the hidden state is constantly being rewritten

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$$

Let's add cell state $\boldsymbol{c}^{(t)}$ that will store long-term information. The model will be able to read, write and rewrite cell information.

To operate with memory three dynamic **gates** will be used:

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \boxed{\sigma}\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

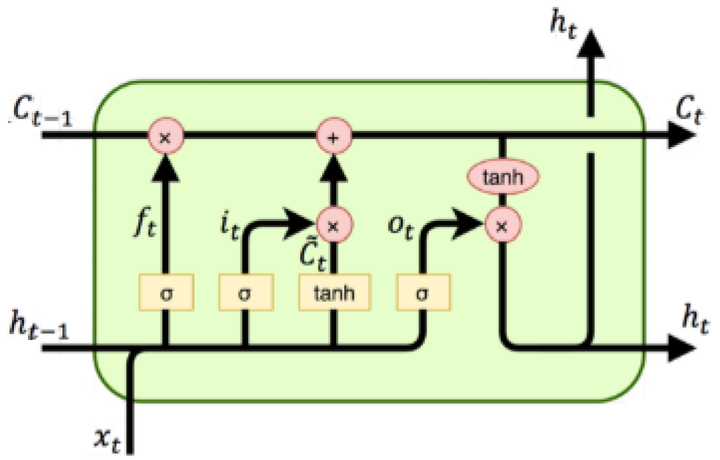
$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

All these are vectors of same length n

Gates are applied using element-wise (or Hadamard) product: $\odot$



GRU

A simpler altermative to LSTM without cell state, proposed in 2014*

Update gate: controls what parts of hidden state are updated vs preserved

Reset gate: controls what parts of previous hidden state are used to compute new content

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

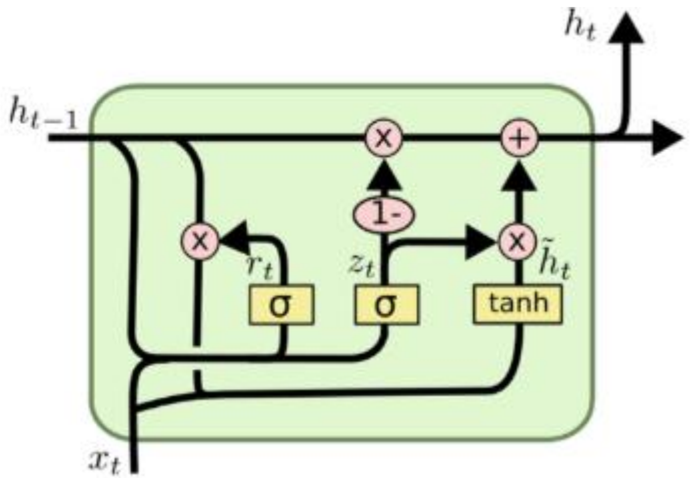
Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{W}_u \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u \boldsymbol{x}^{(t)} + \boldsymbol{b}_u\right)$$

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h (\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$



How does this solve vanishing gradient?
Like LSTM, GRU makes it easier to retain info long-term (e.g., by setting update gate to 0)

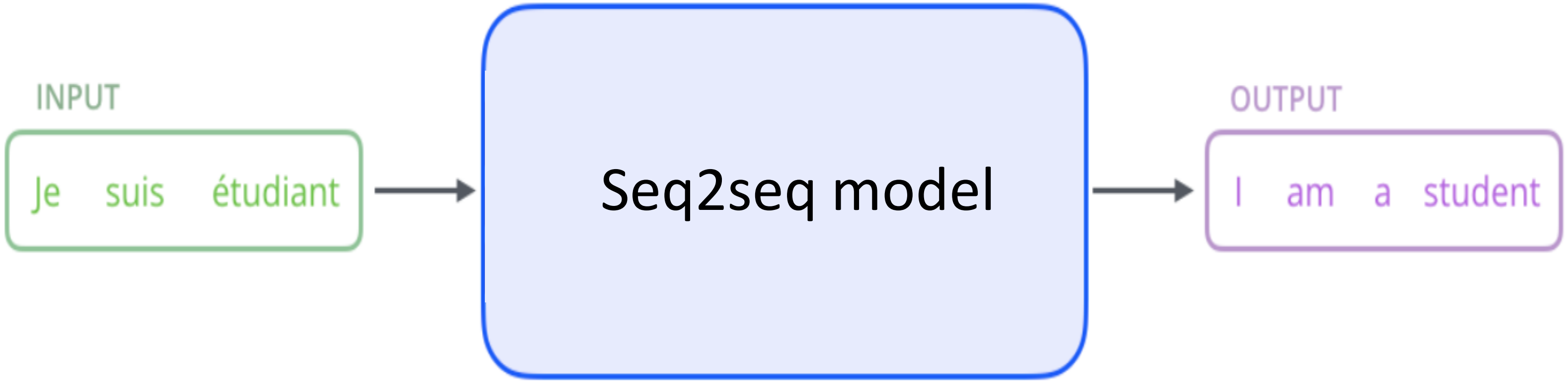
*"Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", Cho et al., 2014

02

Seq2seq

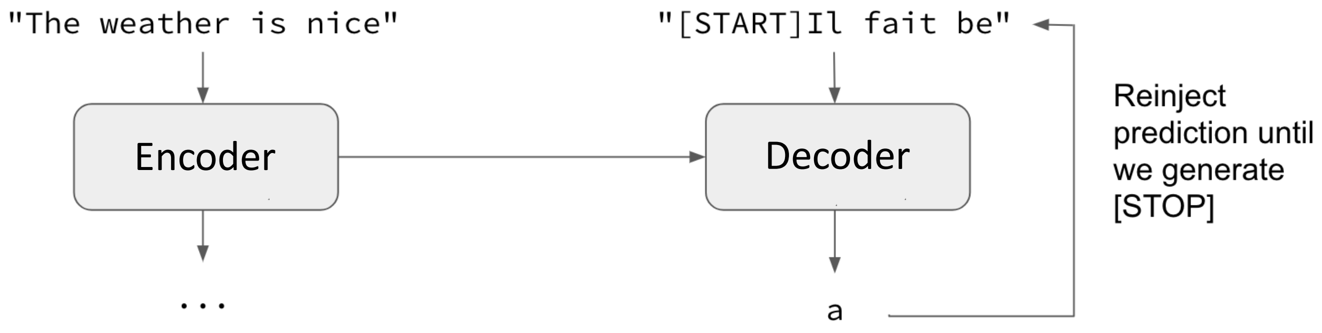
Seq2seq

Sequence-to-sequence learning (Seq2Seq) is the technique of training models to convert sequences from one domain (e.g. sentences in English) to sequences in another domain (e.g. the same sentences translated to French).



Seq2seq

Seq2seq learning, first introduced for machine translation task, can be applied any time you need to generate text (summarization, dialogue generation, question answering, code generation...)



- Encoder - converts the input sequence to corresponding hidden vectors. Each vector represents the current word and its local context.
- Decoder - takes as input the encoder hidden states and the already generated part of the output sequence to predict the next word.

First seq2seq models for neural machine translation used one- or multi-layer RNNs for encoder and decoder.

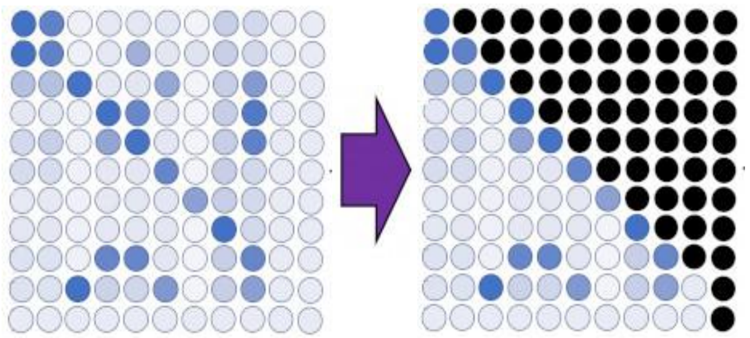
Seq2seq training

- Targets are divided into two parts

```
tar_inp = 'SOS A lion in the jungle is sleeping'
```

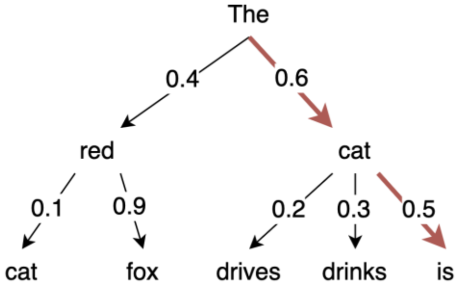
```
tar_real = 'A lion in the jungle is sleeping EOS'
```

- Decoder learns to generate targets offset one timestep in the future
- Teacher forcing is applied during training: the true output is passed to the next time step regardless of model prediction at the current time step
- Self-attention allows the model to look at the previous input tokens to better predict the next token
- To prevent the model from peeking at the expected output the model uses a look-ahead mask
- Seq2seq is optimized as a single system. Backpropagation operates «end-to-end»



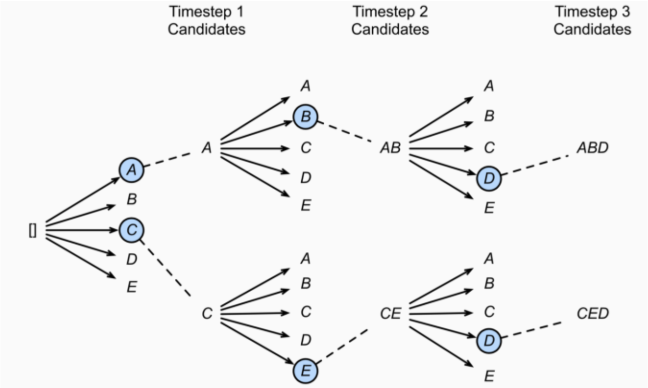
Decoding strategies

Greedy (best path) decoding




```
Context:             Try this cake. I baked it myself.
Optimal Response   : This cake tastes great.
Generated Response: This is okay.
```

Beam search (beam width=2)




```
Context:     Try this cake. I baked it myself.
Response A: That cake tastes great.
Response B: Thank you.
```

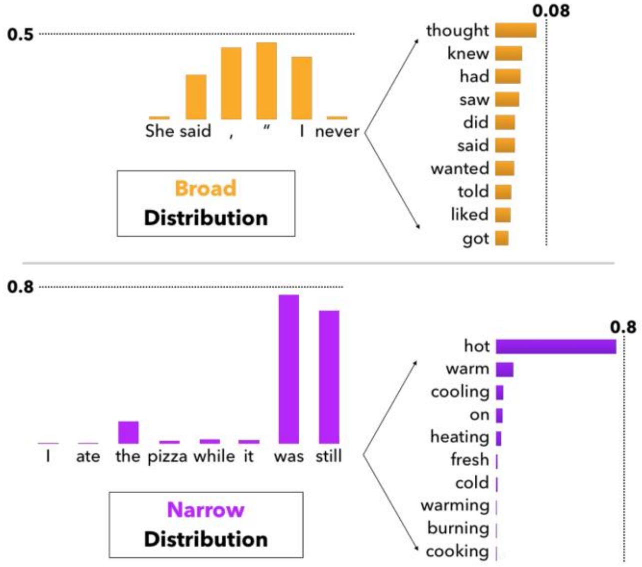
...

```
Context:     ...
Response B: Thank you.
```

...

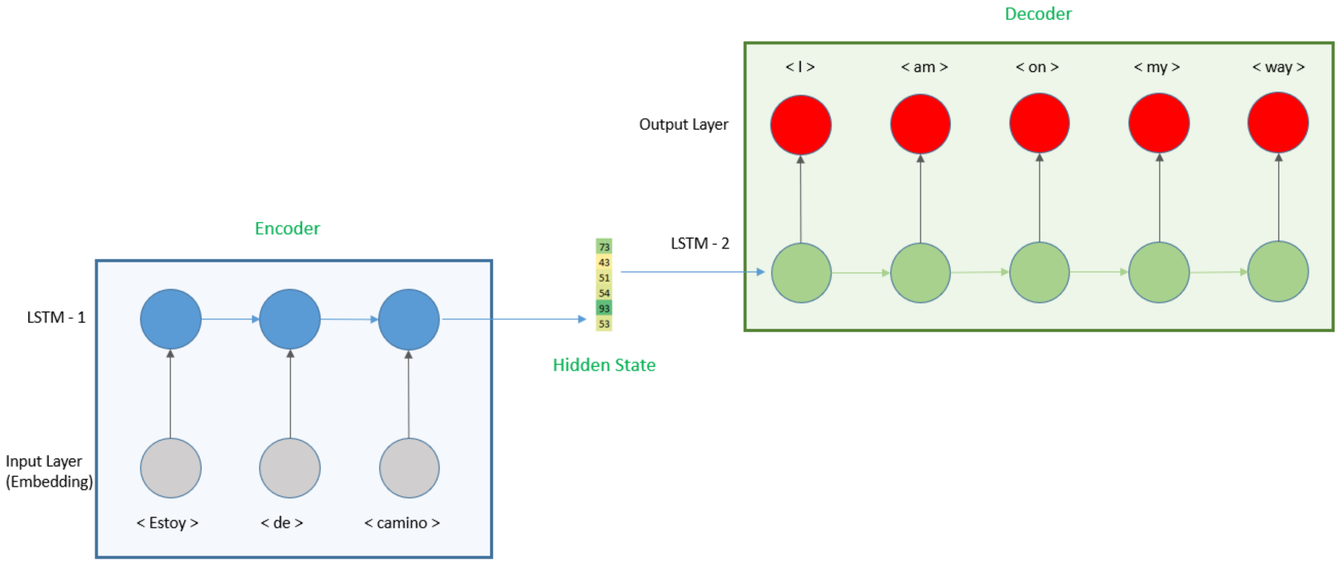
```
Context:     ...
Response B: Thank you.
```

Nucleus sampling



Seq2seq: the bottleneck problem

The encoder input meaning is captured in one vector, so **as the input length increases, the more difficult it gets for the model to capture the information in this vector.** Consequently, its performance decreases with long sentences as it tends to forget parts of it, **the hidden vector becomes a bottleneck.**

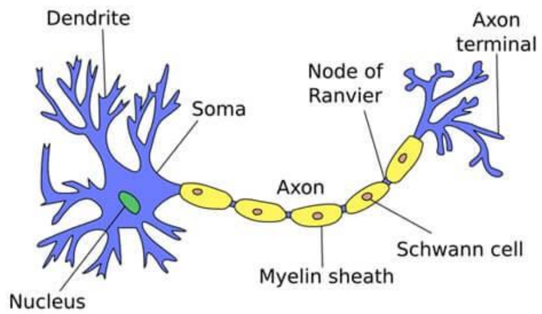


Solution: **attention mechanism**

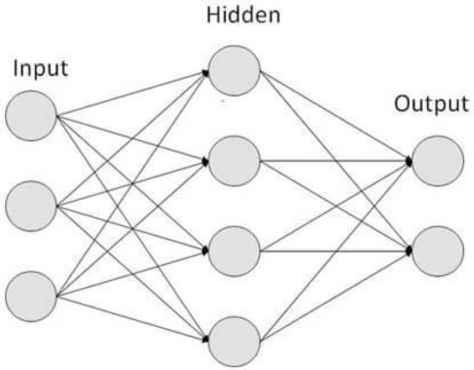
03

Attention intro

Attention

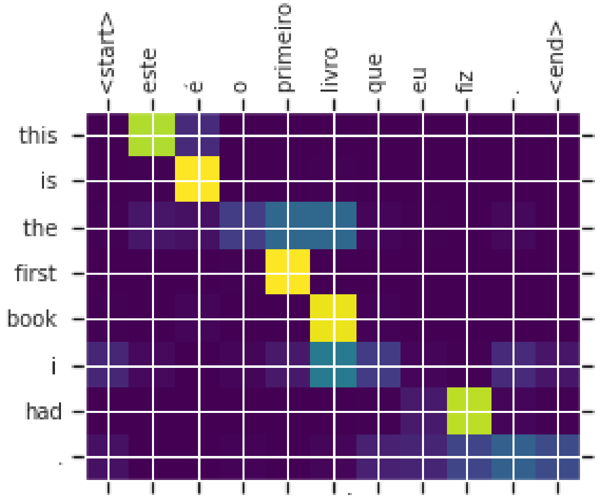


The biological neuron graph



The artificial neural network

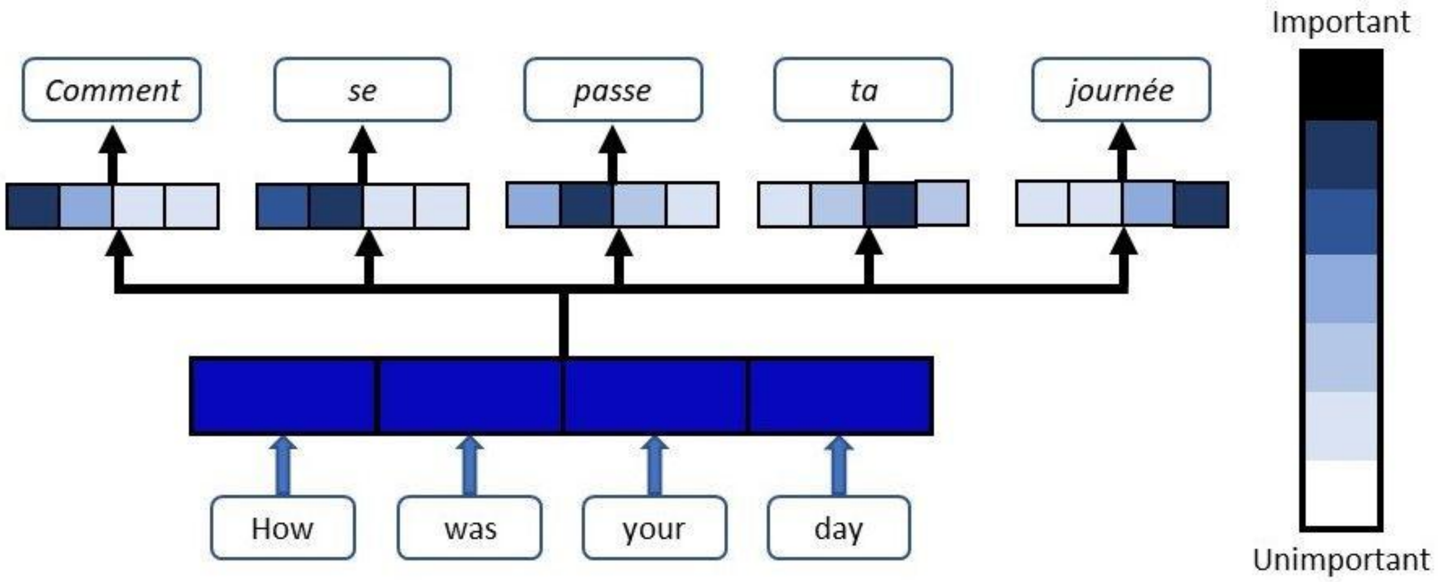
Visual attention map



LM attention map

Attention

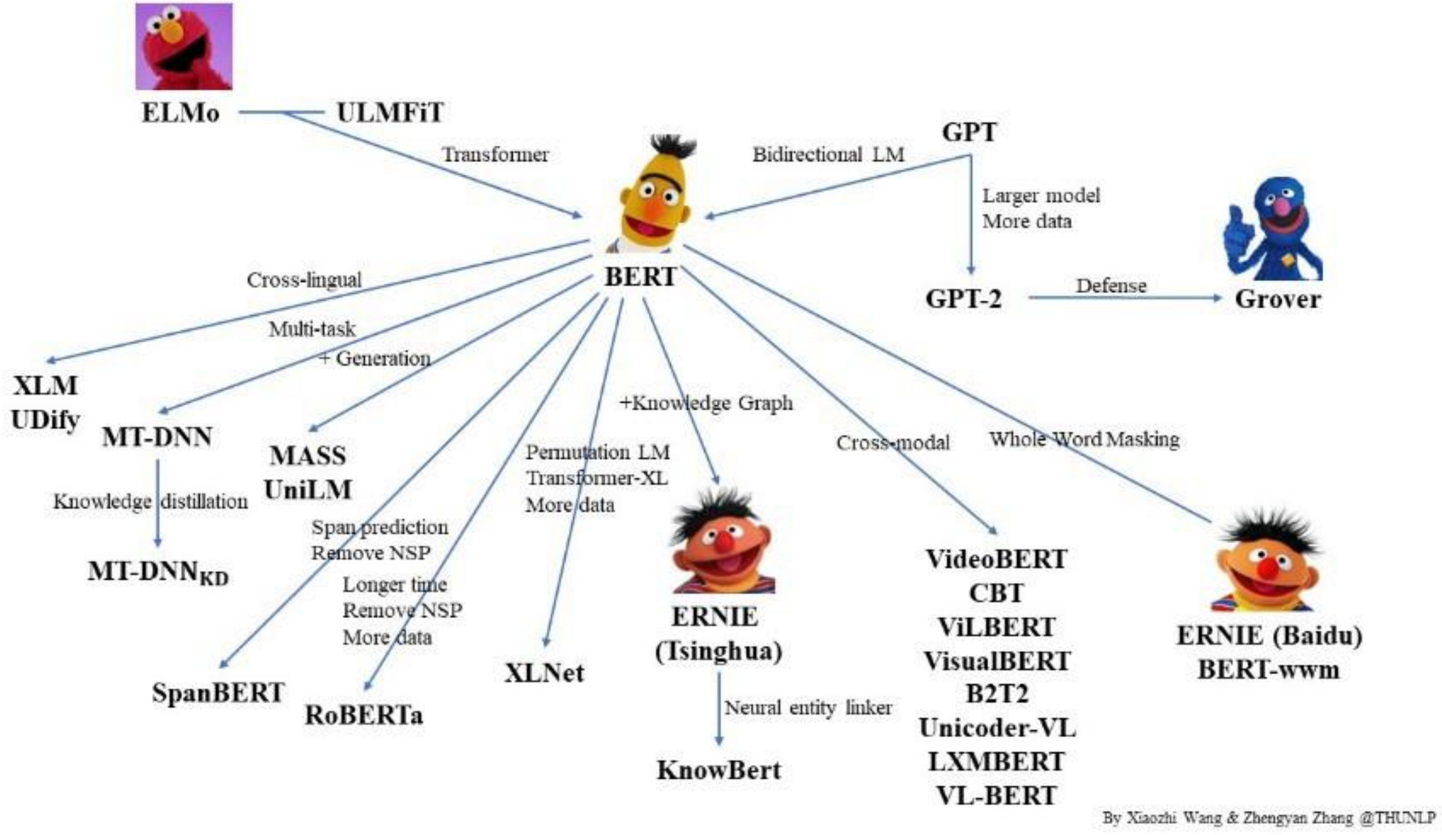
In 2015, Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio* proposed **attention mechanism** to to solve the seq2seq bottleneck problem.

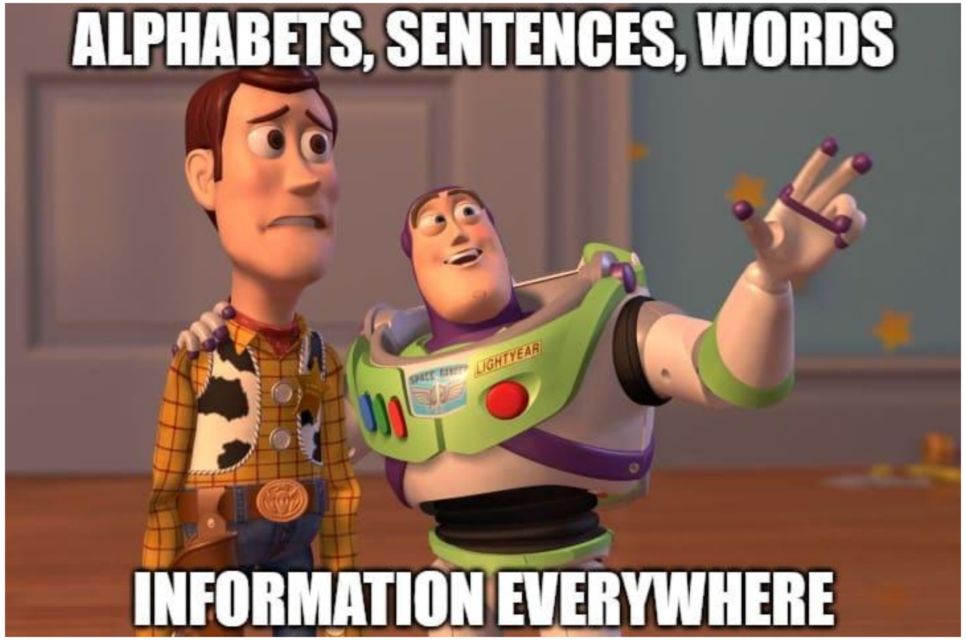


They suggested the context vector to take into account not only all the inputs, but also **a relative importance** of each input elemement.

This mechanism also deals with *another* seq2seq drawback: different parts of input information are needed and relevant to generate different parts of output sequence.

*"Neural Machine Translation by Jointly Learning to Align and Translate" Bahdanau et al., 2015


ELMo
ULMFiT
Transformer
BERT
Bidirectional LM
GPT
Larger model
More data
GPT-2
Defense
Grover
Cross-lingual
Multi-task
+ Generation
XLM
UDify
MT-DNN
MASS
UniLM
Knowledge distillation
MT-DNNKD
Span prediction
Remove NSP
Longer time
Remove NSP
More data
SpanBERT
RoBERTa
Permutation LM
Transformer-XL
More data
XLNet
+Knowledge Graph
ERNIE
(Tsinghua)
Neural entity linker
KnowBert
Cross-modal
VideoBERT
CBT
ViLBERT
VisualBERT
B2T2
Unicoder-VL
LXMBERT
VL-BERT
Whole Word Masking
ERNIE (Baidu)
BERT-wwm
By Xiaozhi Wang & Zhengyan Zhang @THUNLP


ALPHABETS, SENTENCES, WORDS
INFORMATION EVERYWHERE

Artificial Intelligence Research Institute

airi.net

- airi_research_institute
- AIRI Institute
- AIRI Institute
- AIRI_inst
- artificial-intelligence-research-institute