Sequence2Sequence

—

L101: Machine Learning for Language Processing
Andreas Vlachos

Structured prediction reminder

Given an input **x** (e.g. a sentence) predict **y** (e.g. a PoS tag sequence, cf lecture 5):

$$\hat{y} = \underset{y \in \mathcal{Y}}{\arg\max}\; score(x, y)$$

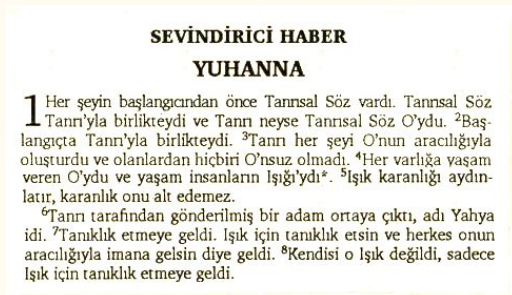Where $\boldsymbol{Y}$ is rather large and often depends on the input (e.g. $L^{|\boldsymbol{x}|}$ in PoS tagging)
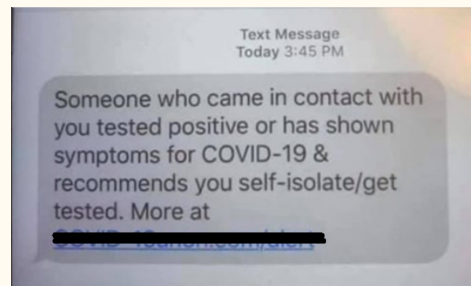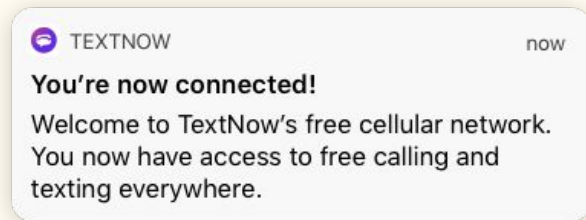
Various approaches:
- Linear models (structured perceptron)
- Probabilistic linear models (conditional random fields)
- Generative models (hidden Markov models)

Most common structures

Dies ist ein Blindtext. An ihm lässt sich vieles über die Schrift ablesen, in der er gesetzt ist. Auf den ersten Blick wird der Grauwert der Schriftfläche sichtbar. Dann kann man prüfen, wie gut die Schrift zu lesen ist und wie sie auf den Leser wirkt. Dies ist ein Blindtext. An ihm lässt sich vieles über die Schrift ablesen, in der er gesetzt ist. Auf den ersten Blick wird der Grauwert der Schriftfläche sichtbar. Dann kann man prüfen, wie gut die Schrift zu lesen ist und wie sie auf den Leser wirkt.

As input?

As output?

Natural language, i.e. sequences of character, words, sentences!

Today: focus on language-to-language methods, a.k.a. seq2seq, encoder-decoder

Language modelling

> How likely is that a sequence of words comes from a particular language (e.g. English)?

Odd sounding problem. Applications:

- speech recognition
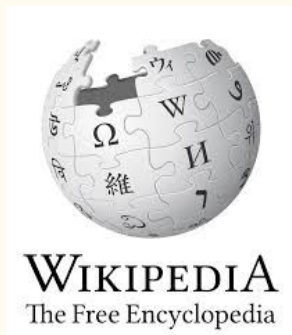- machine translation
- grammatical error detection
- etc.

Problem setup

We want to learn a model that gives us:

$$P(\mathbf{x}), \textbf{for } \forall \mathbf{x} \in V^{maxN}$$

Training?

As much text as we can get!

Language modelling

Decompose the probability of the sentence **x** into conditional probabilities of each word given the previous ones:

$$P(\mathbf{x}) = P(x_1)P(x_2|x_1)P(x_3|x_2,x_1)\ldots P(x_N|x_1,\ldots,x_{N-1})$$

These are typically (until 2010) estimated with maximum likelihood:

$$P(x_n|x_{n-1}\ldots x_1) = \frac{counts(x_1\ldots x_{n-1},x_n)}{counts(x_1\ldots x_{n-1})}$$

Any problems?

Sparsity! Solutions:

- Markov assumption, i.e. N-gram language models
- smoothing: interpolation between models, Kneser-Ney, stupid back-off, etc.

What is a language model?

A giant logistic regression classifier over words:

$$p(x_n = k | x_{n-1} \ldots x_1) = \frac{\exp(\mathbf{w}_k \cdot \phi(x_{n-1} \ldots x_1))}{\sum_{k'=1}^{|\mathcal{V}|} \exp(\mathbf{w}_{k'} \cdot \phi(x_{n-1} \ldots x_1))}$$
$$= softmax(\mathbf{W} \cdot \phi(x_{n-1} \ldots x_1))$$

But terribly inefficient using counts/one-hot encoding as features!

A language generator:

- Sample repeatedly from *p(x)*, each time adding the words in the context
- Stop when the $<$END$>$ of the sentence token is sampled

Skipgram word embeddings reminder

Skipgram ([Mikolov et al. 2013](#)) is a giant word-given-word classifier with learned features:

$$P(w_{t-1}|w_t) = \frac{\exp(\mathbf{c_{t-1}} \cdot \mathbf{w_t})}{\sum_{c' \in V} \exp(\mathbf{c'} \cdot \mathbf{w_t})}$$
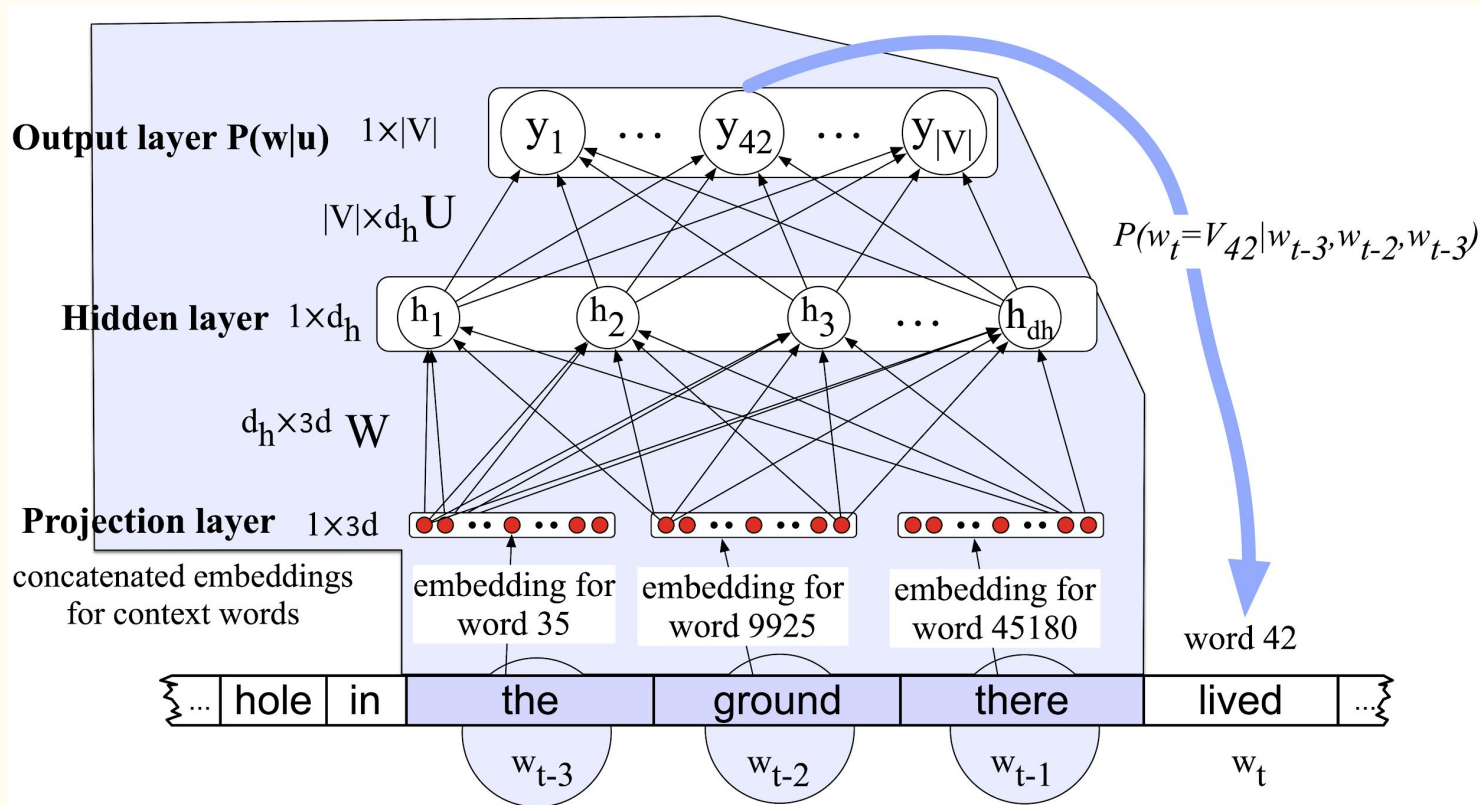
(each word has two embeddings)
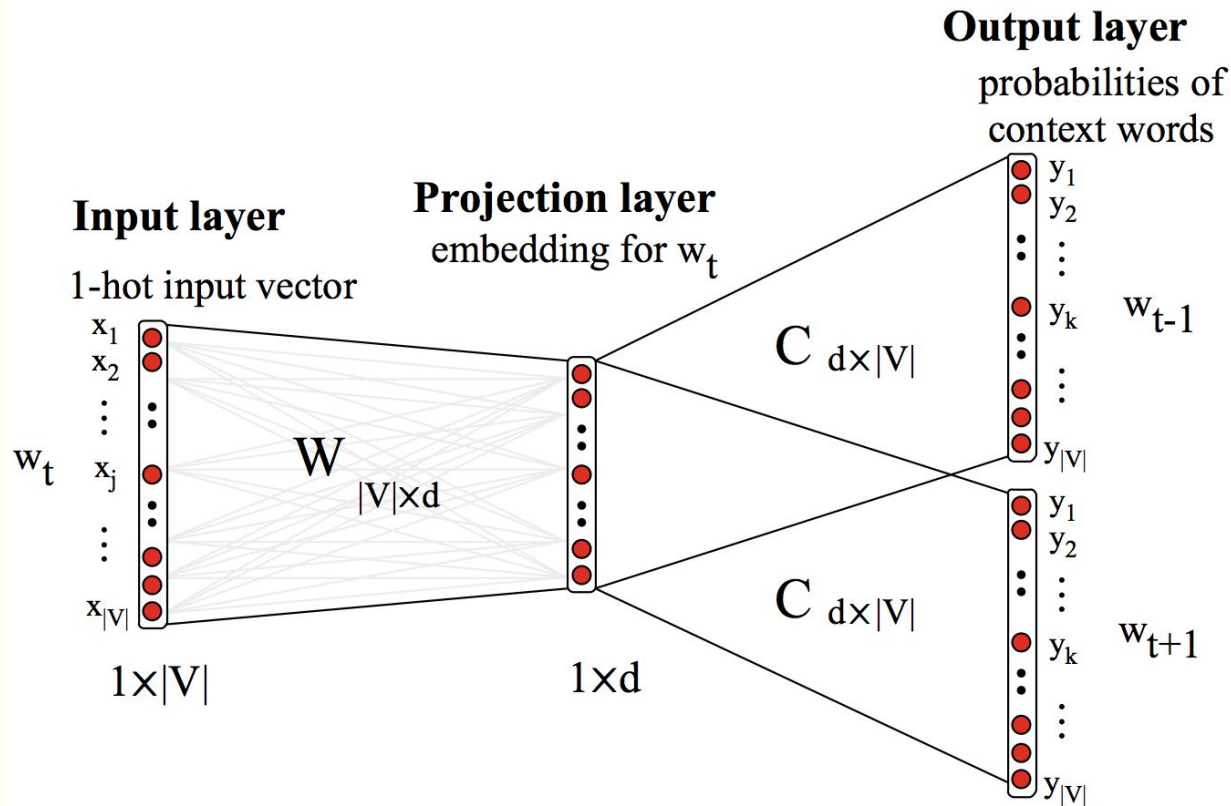
A Feedforward NN N-gram Language model

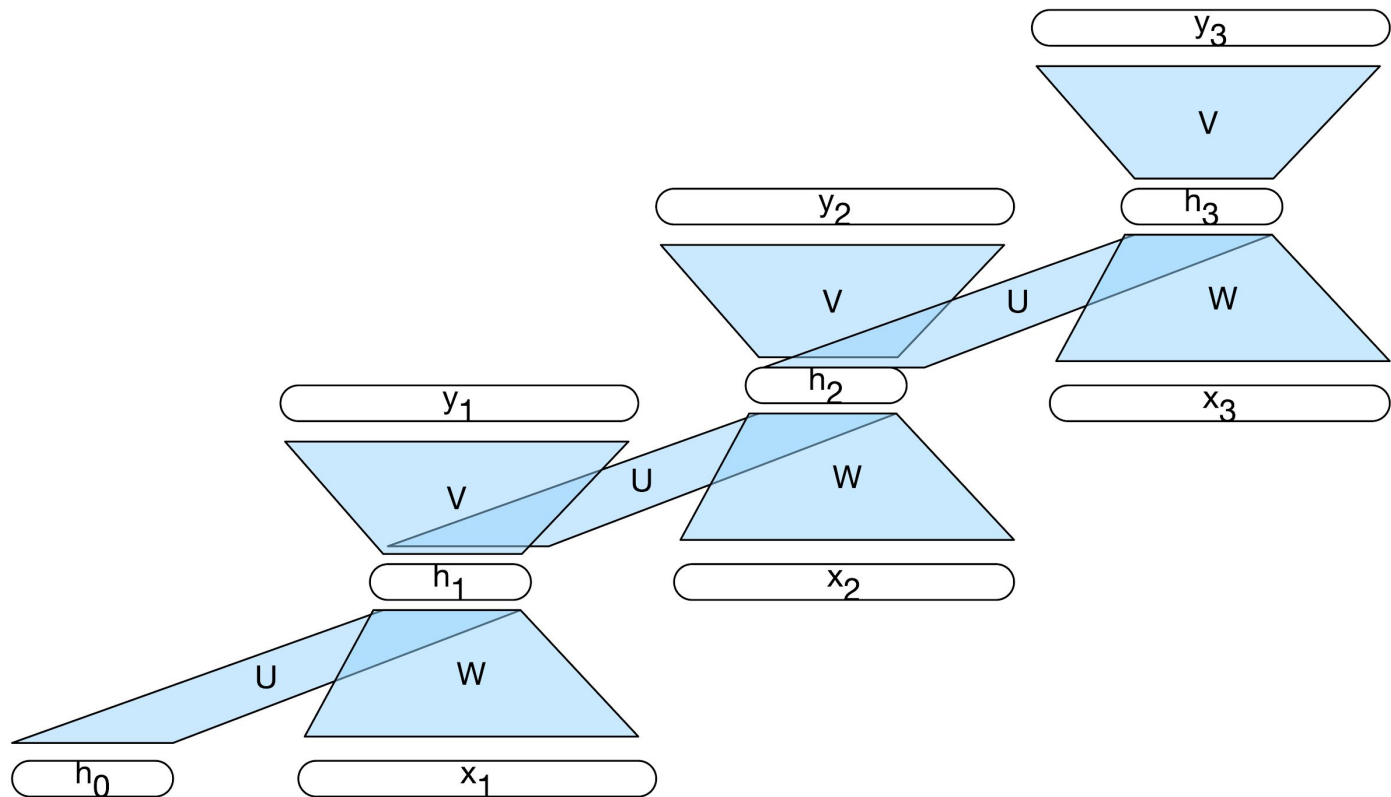Using 3 words as context instead of 1. Limitations?

Context used still limited

NN has to learn how to use the same word in each context place separately

Output layer P(w|u) $1 \times |V|$

y_1 $\cdots$ y_{42} $\cdots$ $y_{|V|}$

$|V| \times d_h$ U

$P(w_t = V_{42} | w_{t-3}, w_{t-2}, w_{t-3})$

Hidden layer $1 \times d_h$

h_1 h_2 h_3 $\cdots$ h_{dh}

$d_h \times 3d$ W

Projection layer $1 \times 3d$

concatenated embeddings for context words

embedding for word 35

embedding for word 9925

embedding for word 45180

word 42

$\cdots$ hole | in | the | ground | there | lived | $\cdots$

w_{t-3} w_{t-2} w_{t-1} w_t

Recurrence

Each word is processed using the same weight matrices.

Recurrent Neural Network

$$p(x_n | x_{n-1} \ldots x_1) = softmax(\mathbf{V} \cdot h_n)$$
$$= softmax(\mathbf{V} \cdot \phi(x_{n-1} \ldots x_1))$$
$$h_n = g(\mathbf{U} \cdot h_{n-1} + \mathbf{W} \cdot x_n)$$

- $\mathbf{V}$ is the output layer, like the weights of logistic regression
- $\mathbf{W}$ is the word embeddings dictionary (can be pre-trained/fine-tuned)
- g is a nonlinear function, e.g. tanh
- $\mathbf{U}$ determines how to use the representation of the context $\mathbf{h}_{t\text{-}1}$

Backpropagation through time

Why not simple backpropop?

Each $\mathbf{h}_t$ affects $\mathbf{y}_t$ and every $\mathbf{y}_{t'>t}$ afterwards

The loss calculation needs to take all into account

Unroll the network for a fixed number of steps

Still uses unlimited context during testing
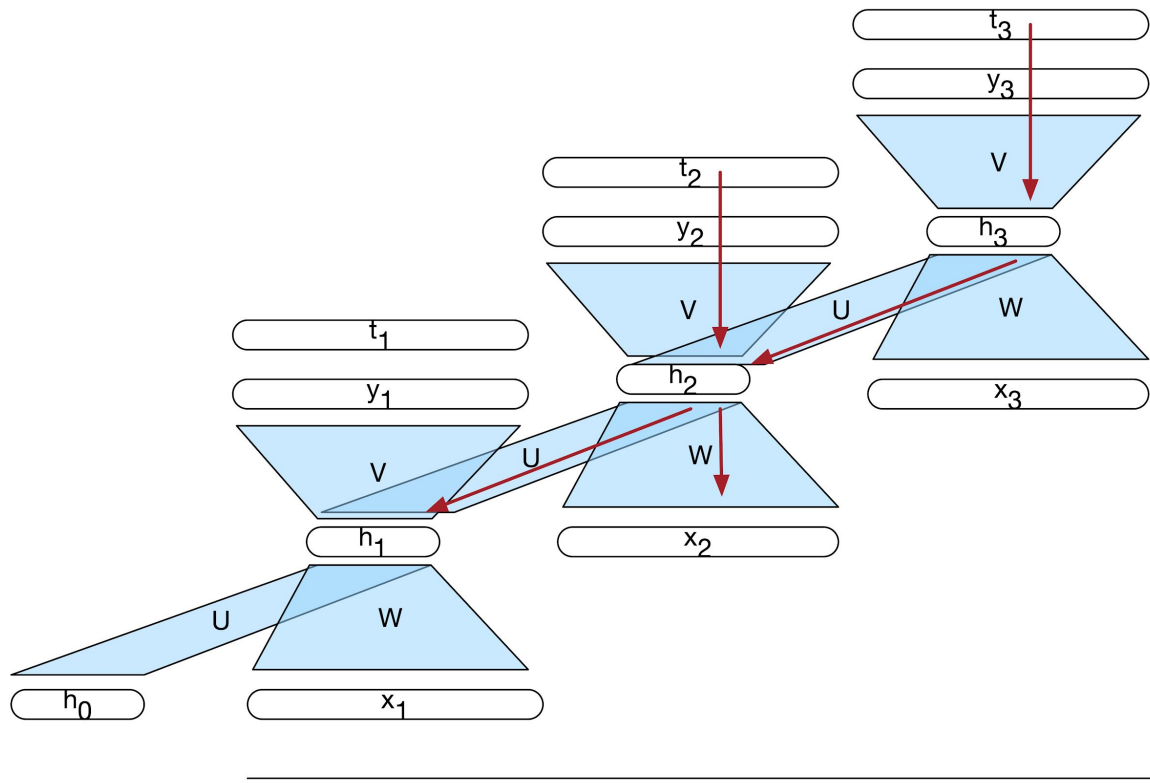
Decoding, i.e. generation

1. Sample a word
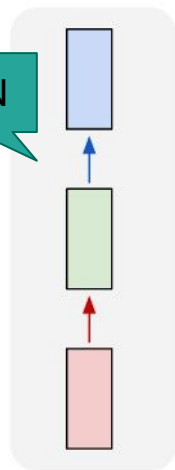2. Feed its embedding
3. Repeat until end of sentence

Many options for decoding

- max
- random
- top-k
- nucleus (Holtzman et al.)
- temperature scaling
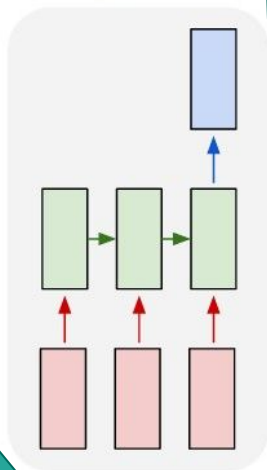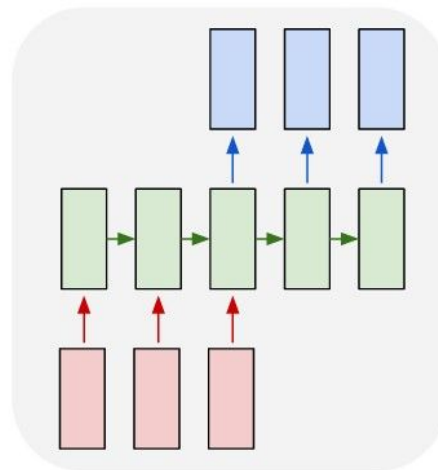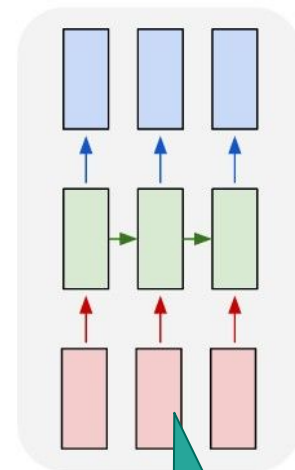- beam search (next lecture)
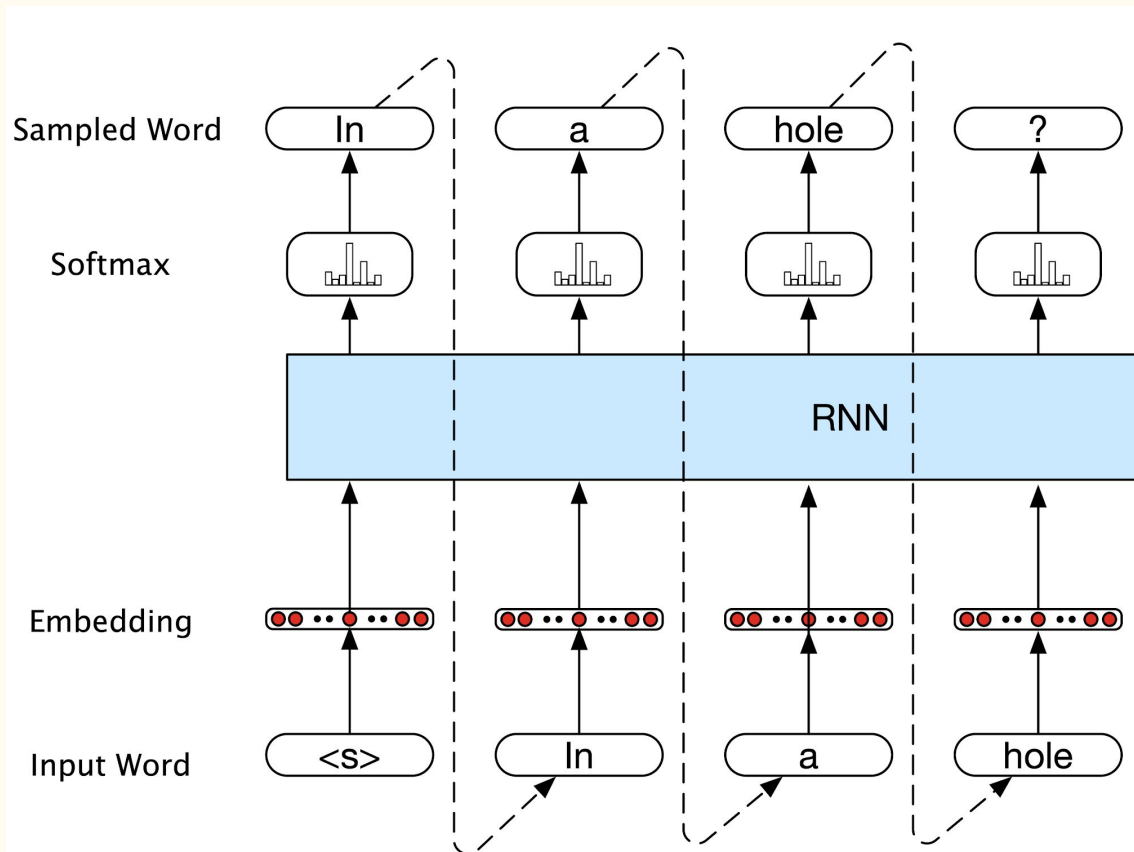
Typical combinations

one to one   one to many   many to one   many to many   many to many
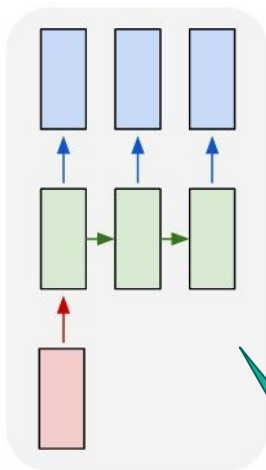
FFNN

Sequence encoder and classifier

Seq2seq, encoder-decoder, language generator conditioned on sequence

Which tasks fit which variant?

- PoS tagging?
- Machine translation?
- Image captioning?

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Conditional language generator

Sequence encoder, token tagger

A few more pointers

- Bidirectional RNN encoders are commonly used
  - Sentence representations avoiding being biased by the last tokens
  - Word representations taking into account context left and right
- Multiple hidden layers are commonly used (stacked RNNs)
  - More demanding computationally
  - Able to learn more high level features
- Batching (for GPU) requires padding
- Convolutional Neural Networks are also used relying on multiple layers to mitigate the effect of the fixed window
- While using words as the modelling unit, using characters and subwords helps deal with rare/unknown words

Long-range dependencies

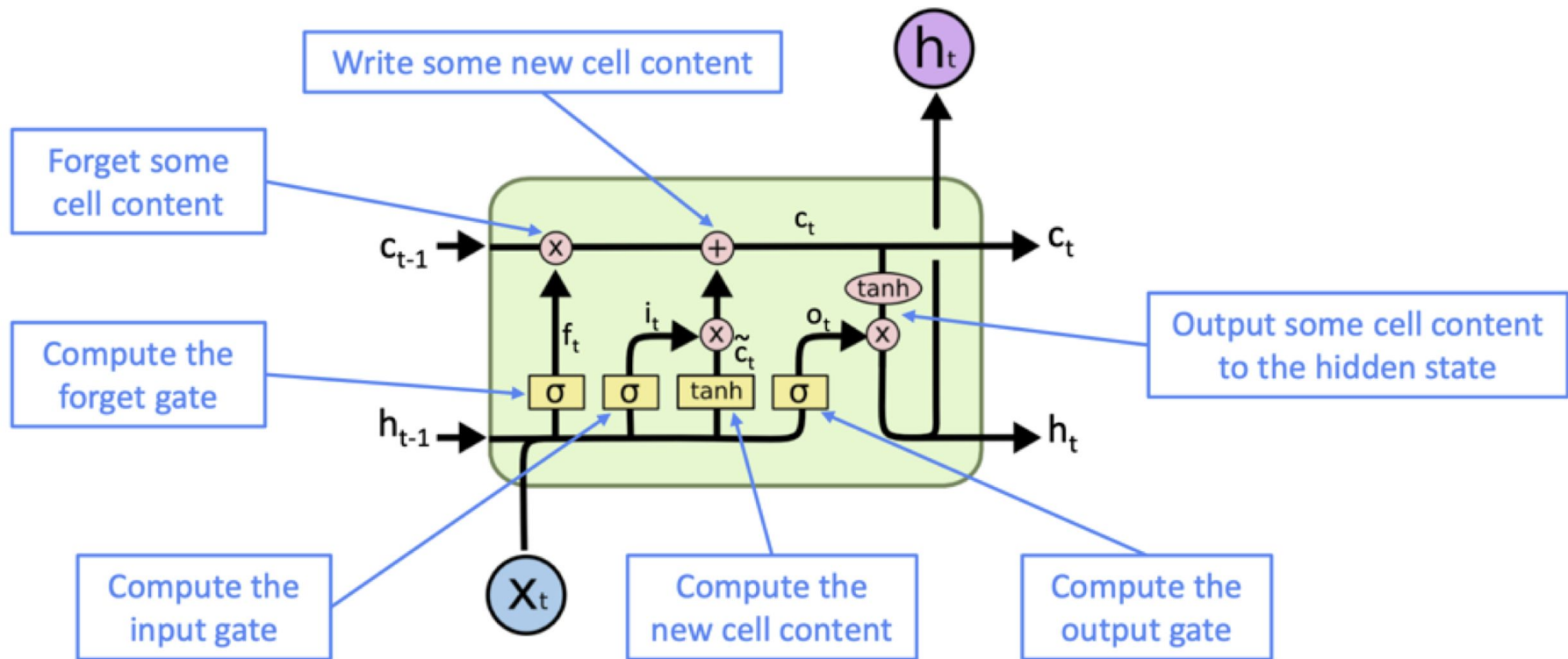RNNs don't handle them well, but Long-Short Term Memory Networks (Hochreiter and Schmidhuber, 1997) do:
- introduce a **memory cell**, running parallel to the hidden state
- **forget gate** that decides which part of the memory to drop
- **input gate** that decides which part of the input to add to the memory
- **output gate** that decides which part of the memory to use in the hidden state

Advantages:
- Memory cell allows us to keep information not immediately needed
- Addresses the issue of vanishing/exploding gradients
  - see gradient clipping as an alternative

Main disadvantage: more parameters to learn, but usually worth it
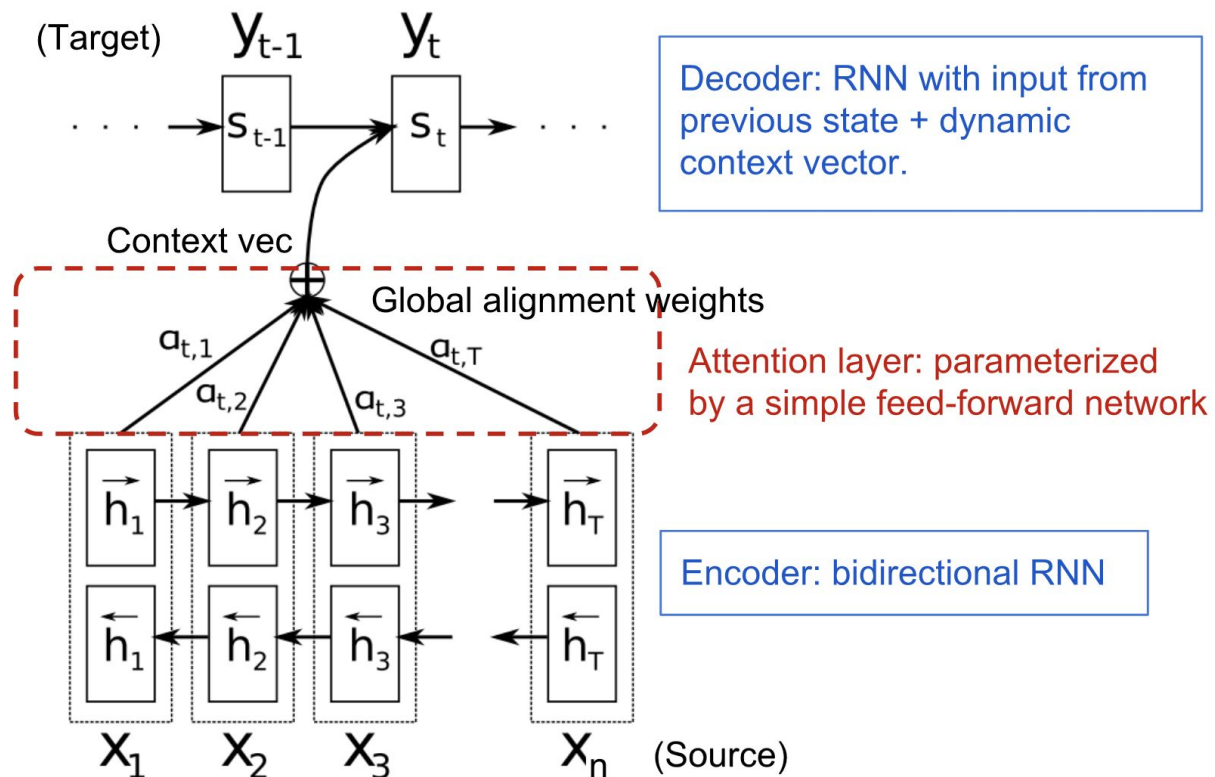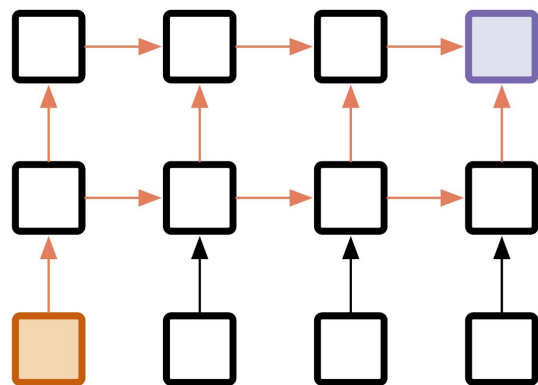
Long-short term memory networks

Write some new cell content

Forget some cell content

Compute the forget gate

Compute the input gate

Compute the new cell content

Compute the output gate

Output some cell content to the hidden state

Abigail See, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Attention

Generating a sentence based on one vector suboptimal: not all inputs relevant to every output

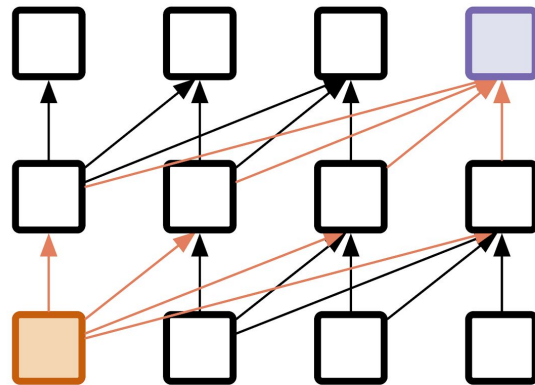Allow the model to use hidden states of the input apart from the last one

Attention intuitively works as an alignment mechanism, but it is not clear how/why

(Target) y_{t-1} y_t

Decoder: RNN with input from previous state + dynamic context vector.

Context vec

Global alignment weights

$a_{t,1}$ $a_{t,2}$ $a_{t,3}$ $a_{t,T}$

Attention layer: parameterized by a simple feed-forward network

$\vec{h}_1$ $\vec{h}_2$ $\vec{h}_3$ $\vec{h}_T$

$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$

Encoder: bidirectional RNN

x_1 x_2 x_3 x_n (Source)

https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Self-attention instead of recurrence

RNN
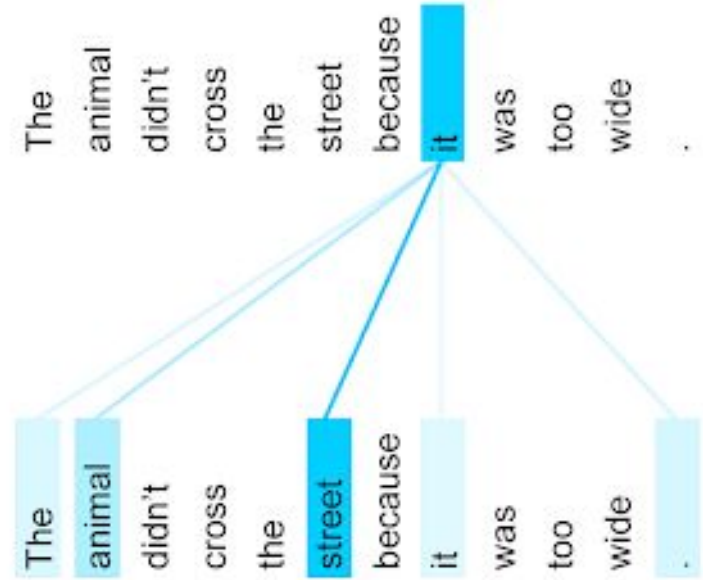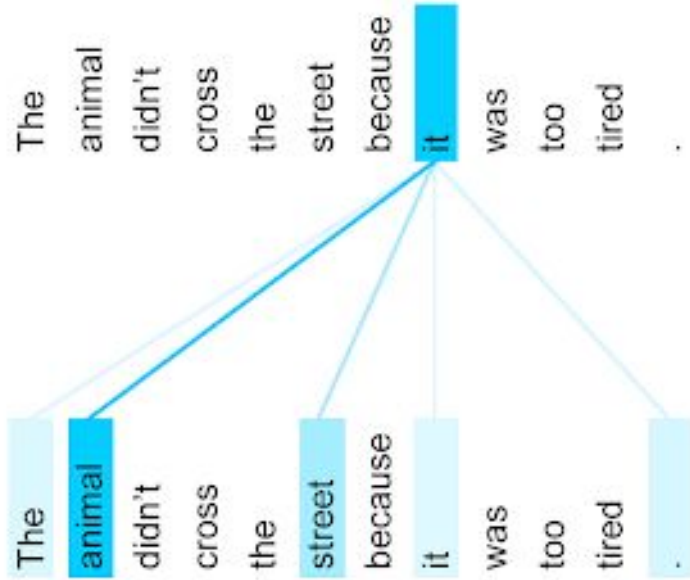
Transformer

Arriana Bisazza
(AthNLP 2019)

Key idea behind the Transformers (Vaswani et al. 2017)
- Better parallelization, i.e. faster, more data, etc.
- Can be seen as a fully connected graph neural network

Transformers - overview
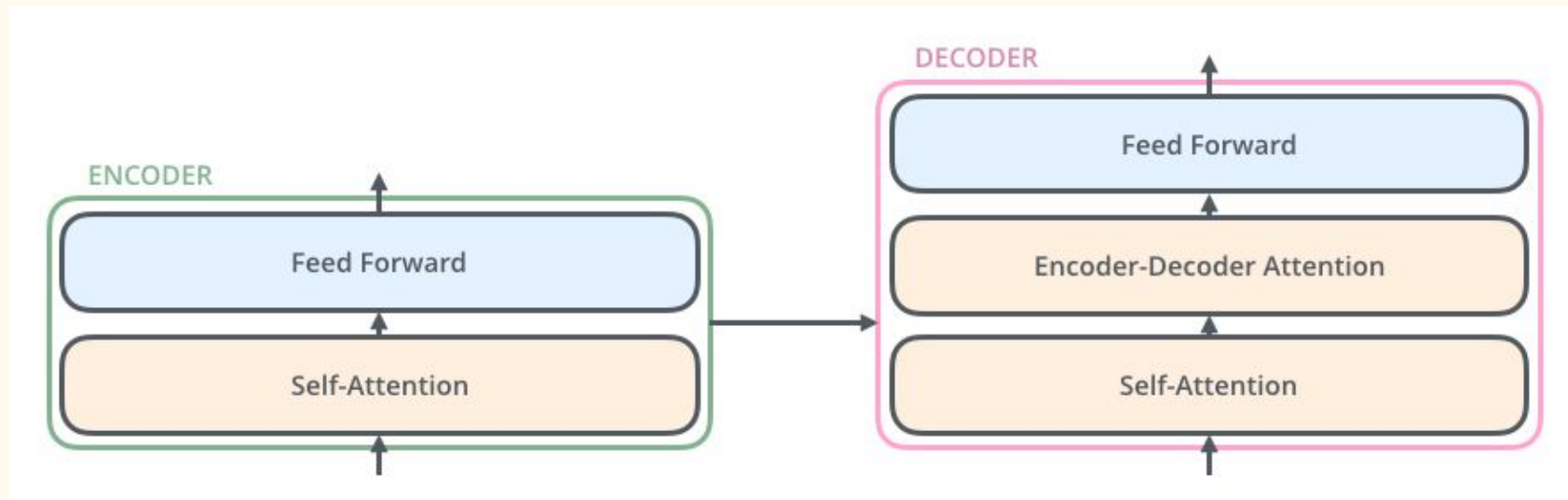
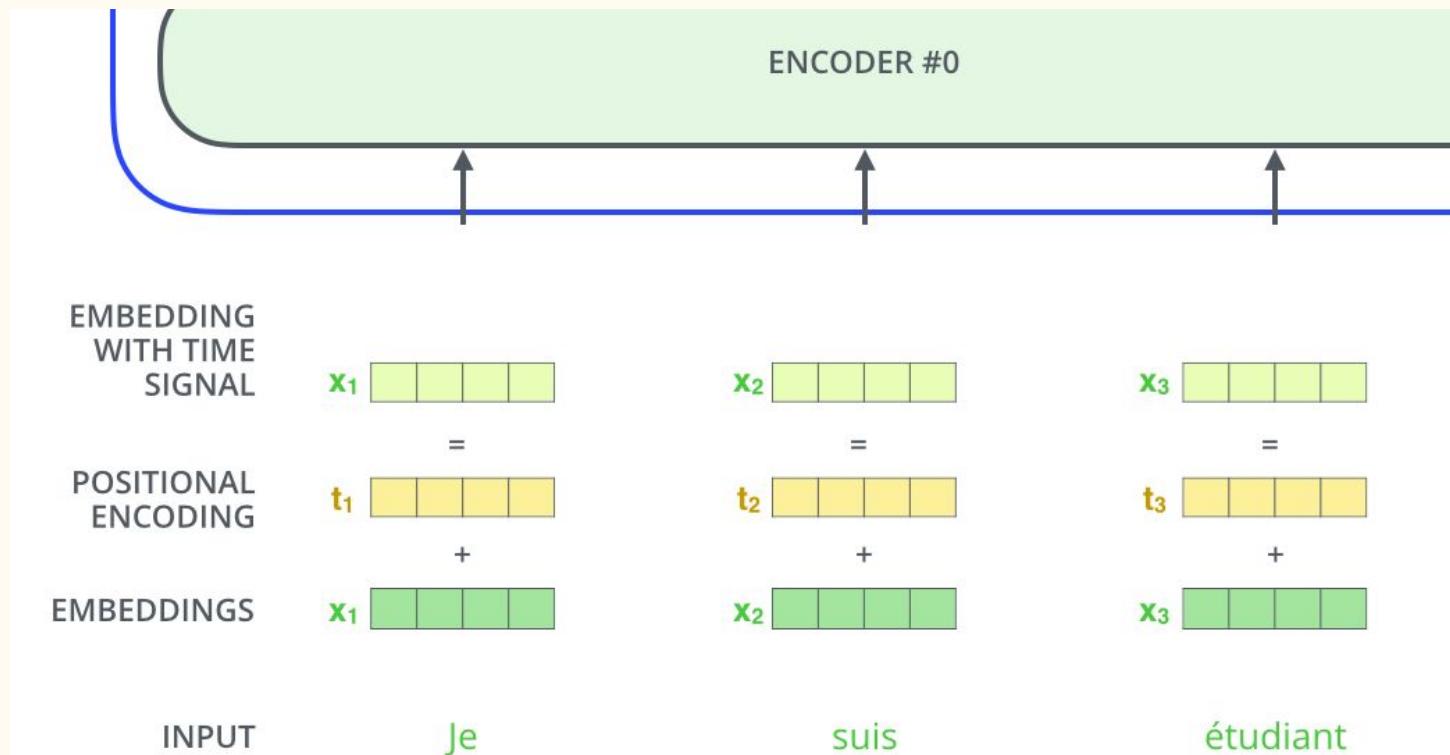# Transformers - multiple layers of self-attention

Each layer has multiple attention heads which can be pruned after training
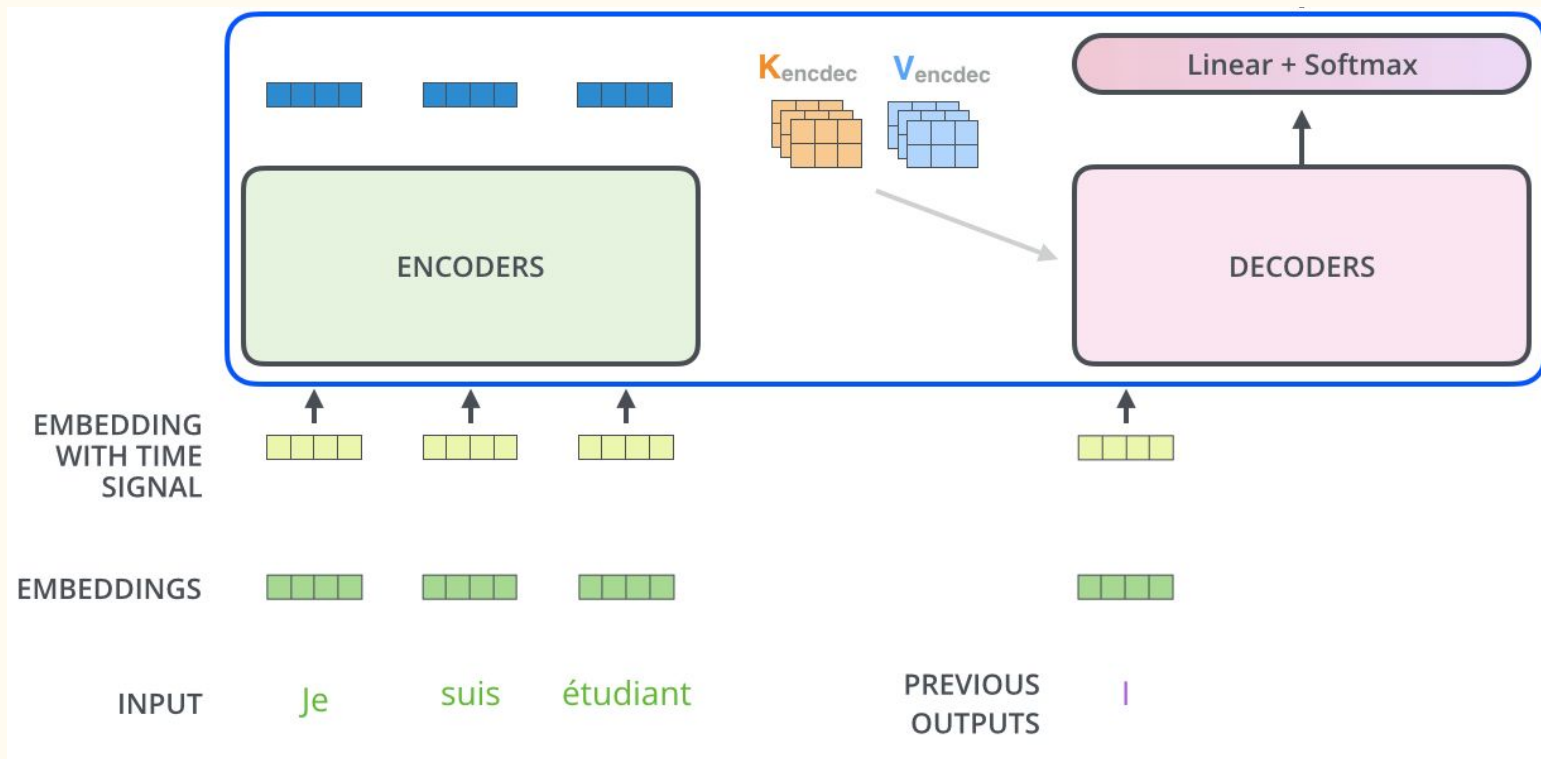
Transformers - positional encoding

No recurrence, but order is taken into account via the positional embeddings

Transformers - decoder

Same as encoding but can only self-attend to tokens already generated and attends to the input too

BERT (Devlin et al., 2018)

BERT: Bidirectional Encoder Representations from Transformers, i.e.:
- take the transformer encoder stack (self-attention, positional encodings, etc.)
- download a lot of text (sub-word tokenized)
- add special tokens for the sentence beginning and separators
- train two models: left-to-right ($\cong$GPT-2) and right-to-left using two objectives:
  - Masked language modelling: predict words missing at random from the text
  - Next sentence prediction: predict whether the next sentence was the one in the text or not

Typically considered the baseline method to beat:
- use it pre-trained as an input encoder/feature extractor
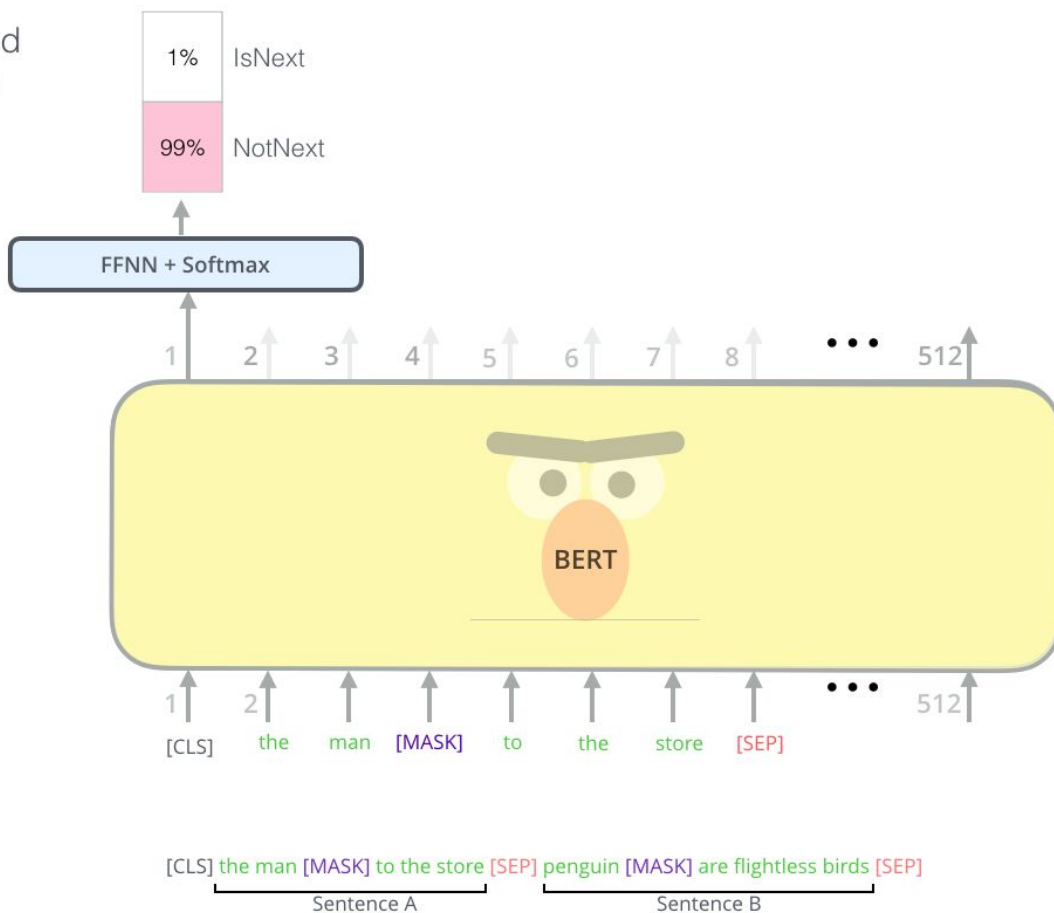- fine-tune it to produce token/sentence embeddings for the task

BERT

Task 1: masked language modelling

Task 2: next sentence prediction

http://jalammar.github.io/illustrated-bert/

Predict likelihood that sentence B belongs after sentence A

1% IsNext
99% NotNext

FFNN + Softmax

1   2   3   4   5   6   7   8   ...   512

BERT

Tokenized Input

1   2   ...   512

[CLS]   the   man   [MASK]   to   the   store   [SEP]

Input

[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]

Sentence A   Sentence B

Bibliography

- [Jurafsky and Martin chapter on RNNs](#) (RNN figures are from there unless otherwise stated)
- This [blog](#) explains RNNs and BPTT with code
- The [deep learning book](#), chapter 10
- [Two](#) [blogs](#) about transformers
- [BERT survey](#)
- Noah Smith's [introduction to contextual word embeddings](#)
- Seq2Seq can be used for generating sequences that are not words (only) such as [semantic parsing](#)

We have addressed the problem of sparse N-grams

But we replaced it with the problem of empty translations (next lecture!)