

Sequence2Sequence

—

L101: Machine Learning for Language Processing
Andreas Vlachos



Structured prediction reminder

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

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}} \text{score}(\mathbf{x}, \mathbf{y})$$

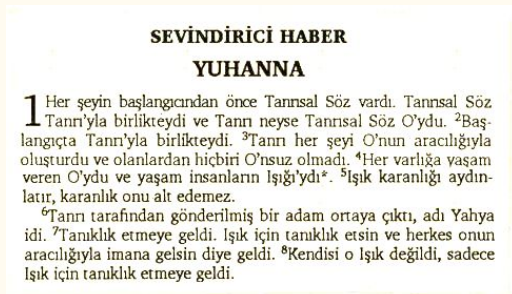
Where \mathcal{Y} is rather large and often depends on the input (e.g. $L^{|\mathbf{x}|}$ in PoS tagging)

Various approaches:

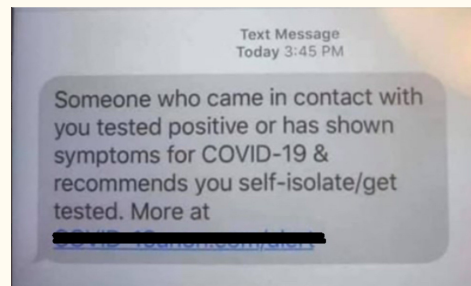
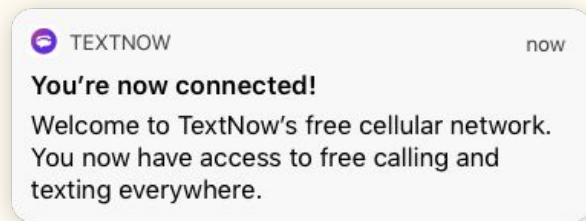
- Linear models (structured perceptron)
- Probabilistic linear models (conditional random fields)
- Generative models (hidden Markov models)

Most common structures

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.

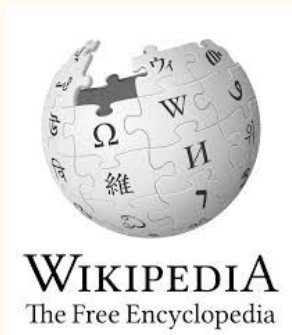
Problem setup

We want to learn a model that gives us:

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

Training?

As much text as we can get!



Language modelling

Decompose the probability of the sentence \mathbf{x} into conditional probabilities of each word given the previous ones:

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

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

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

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:

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

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

A language generator:

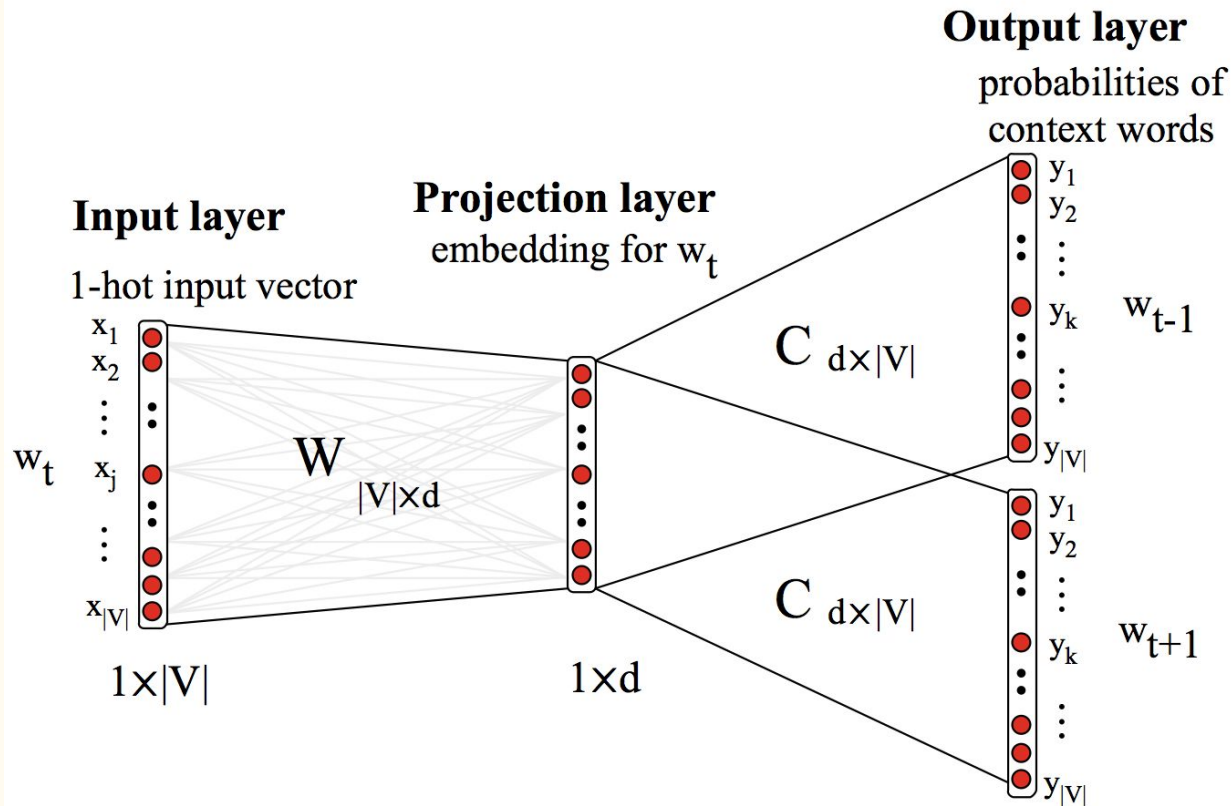
- Sample repeatedly from $p(\mathbf{x})$, each time adding the words in the context
- Stop when the $\langle \text{END} \rangle$ of the sentence token is sampled

Skipgram word embeddings reminder

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

$$P(w_{t-1} | w_t) = \frac{\exp(\mathbf{c}_{t-1} \cdot \mathbf{w}_t)}{\sum_{c' \in V} \exp(c' \cdot \mathbf{w}_t)}$$

(each word has two embeddings)

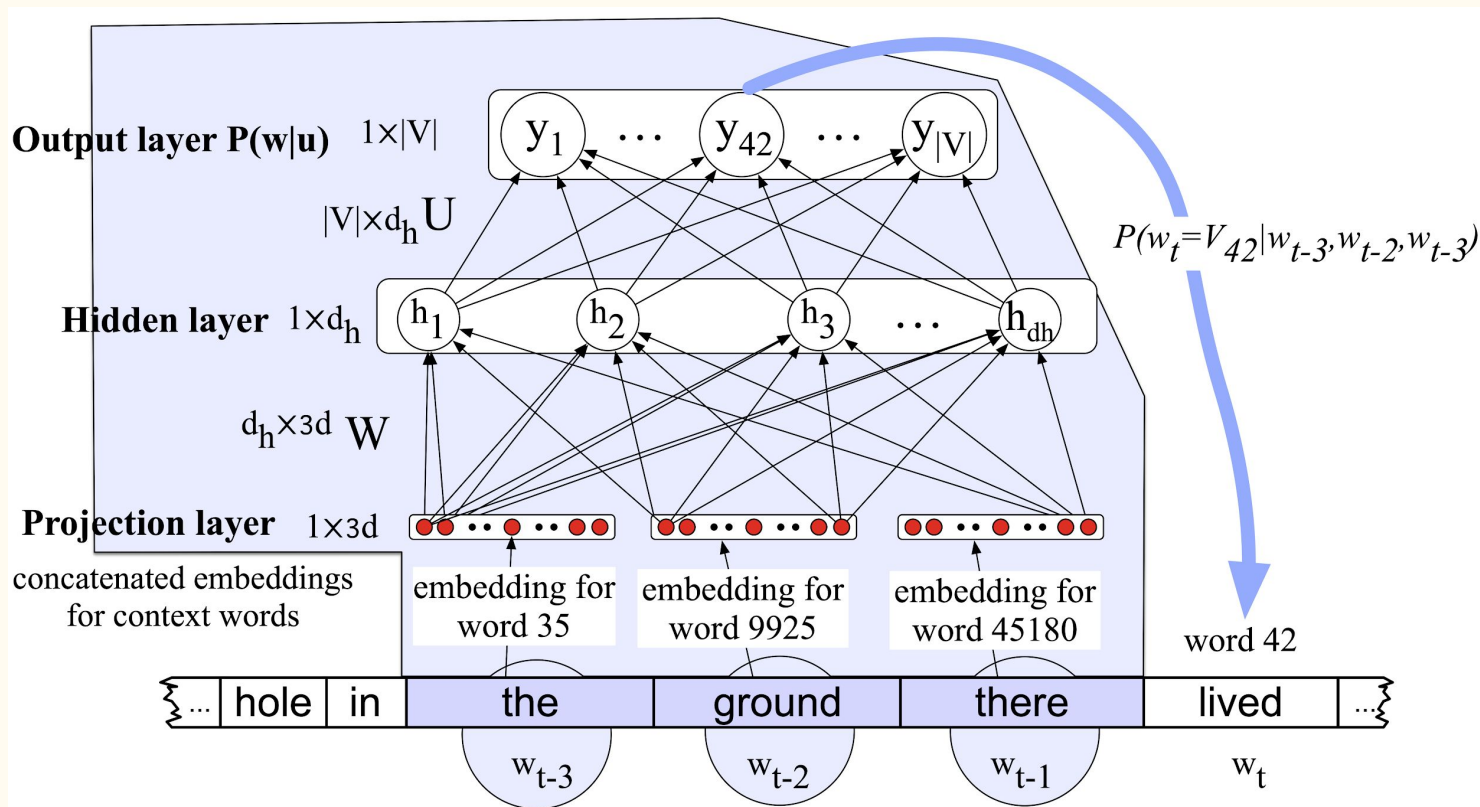


A Feedforward NN N-gram Language model

Using 3 words
as context
instead of 1.
Limitations?

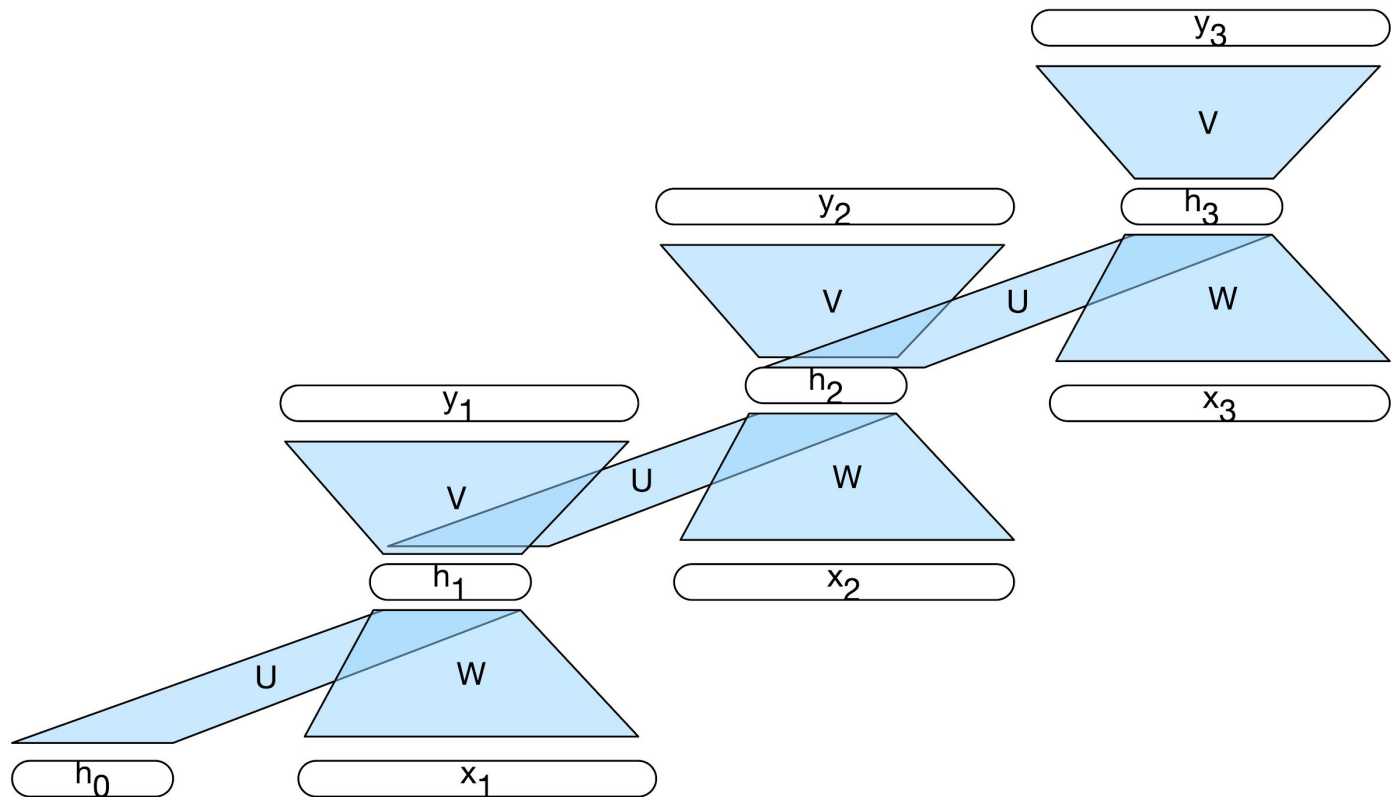
Context used
still limited

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



Recurrence

Each word is processed using the same weight matrices.



Recurrent Neural Network

$$\begin{aligned} p(x_n | x_{n-1} \dots x_1) &= \textit{softmax}(\mathbf{V} \cdot h_n) \\ &= \textit{softmax}(\mathbf{V} \cdot \phi(x_{n-1} \dots x_1)) \\ h_n &= g(\mathbf{U} \cdot h_{n-1} + \mathbf{W} \cdot x_n) \end{aligned}$$

- \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-1}

Backpropagation through time

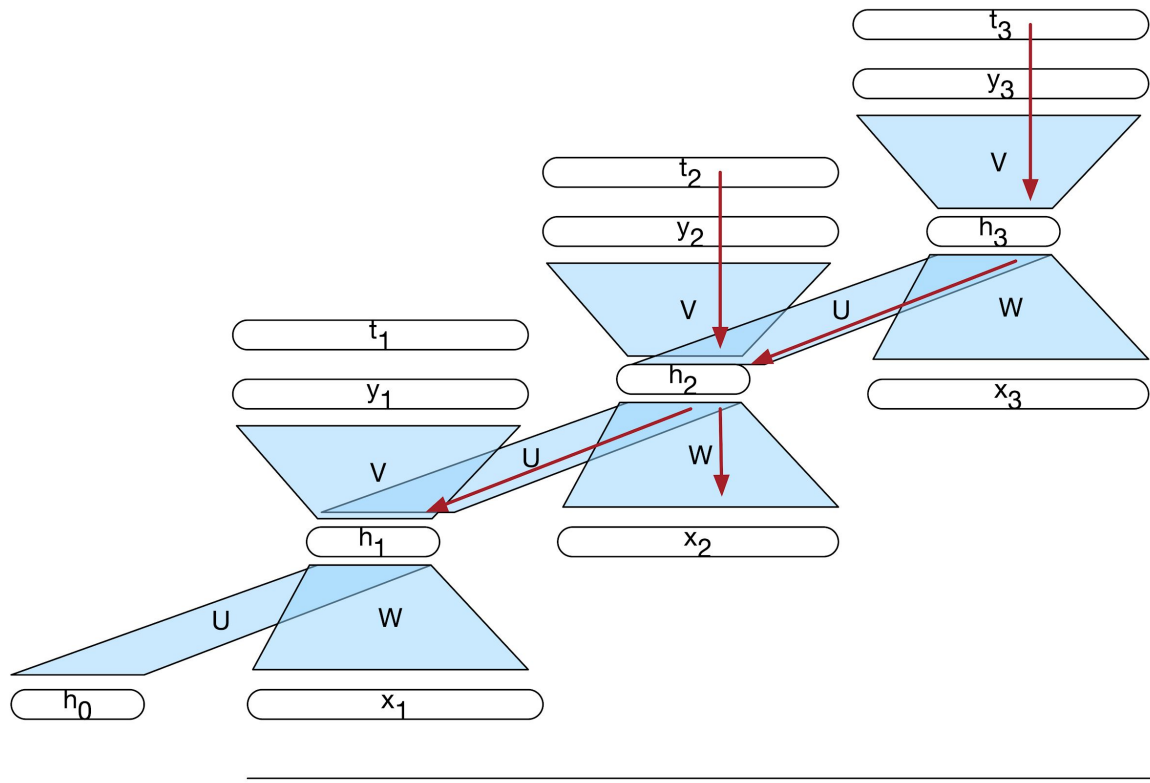
Why not simple backpropop?

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

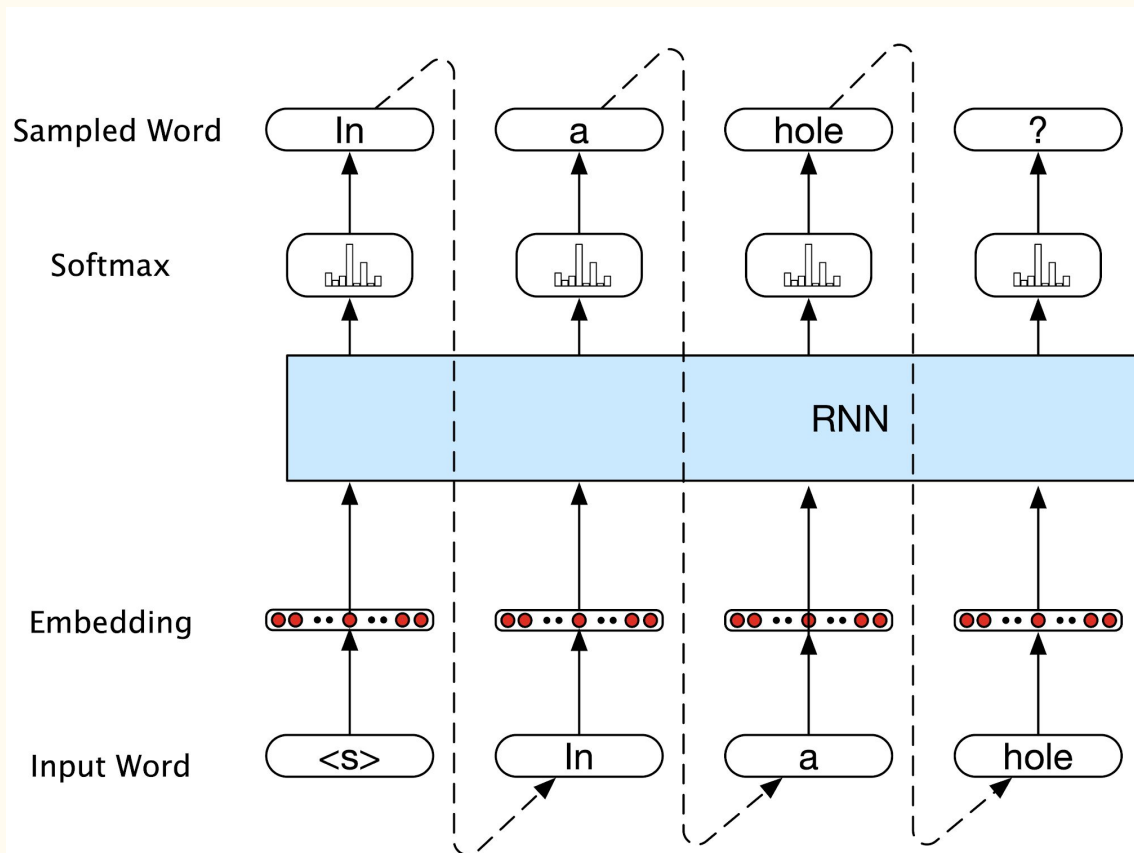


Decoding, i.e. generation

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

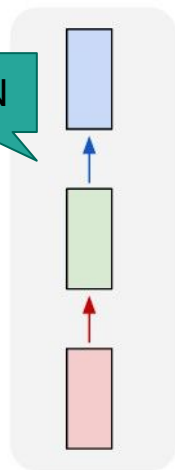
Many options for decoding

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

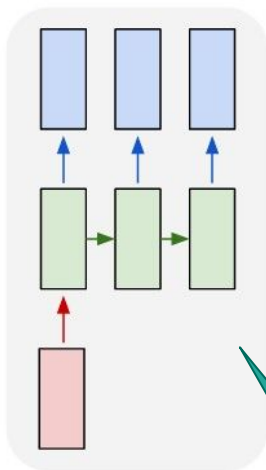


Typical combinations

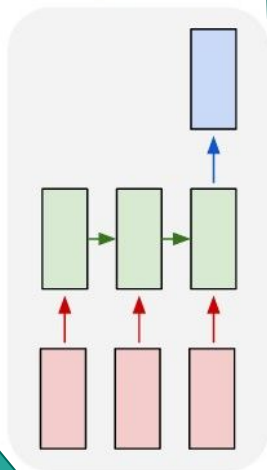
one to one



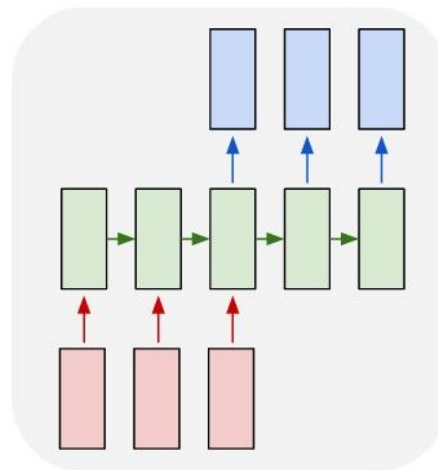
one to many



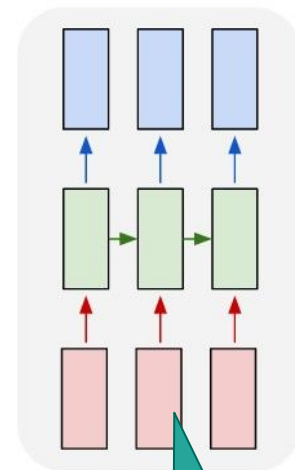
many to one



many to many



many to many



Sequence encoder and classifier

Seq2seq, encoder-decoder, language generator conditioned on sequence

Conditional language generator

Sequence encoder, token tagger

Which tasks fit which variant?

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

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

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:

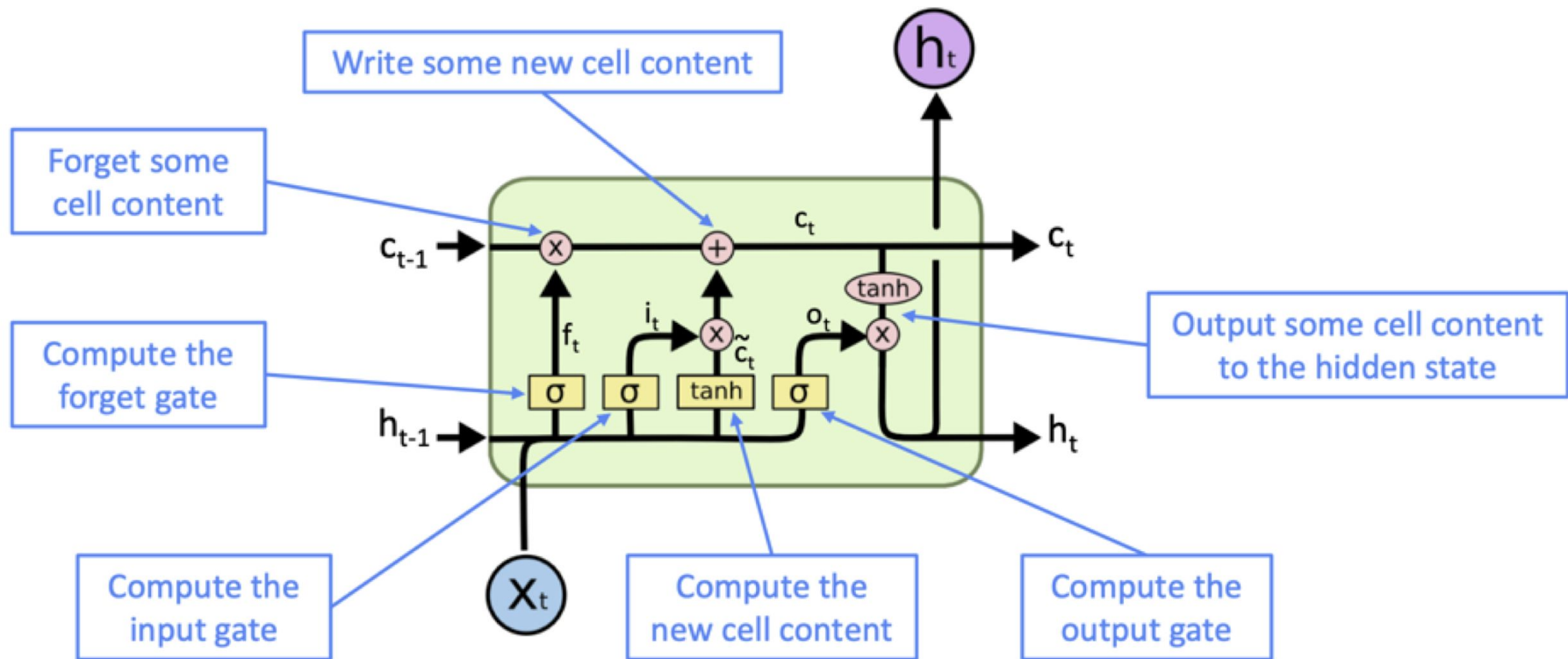
- introduce a **memory cell**, running parallel to the hidden state
- **forget gate** that decides which part of the memory to drop
- **input gate** that decides which part of the input to add to the memory
- **output gate** that decides which part of the memory to use in the hidden state

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

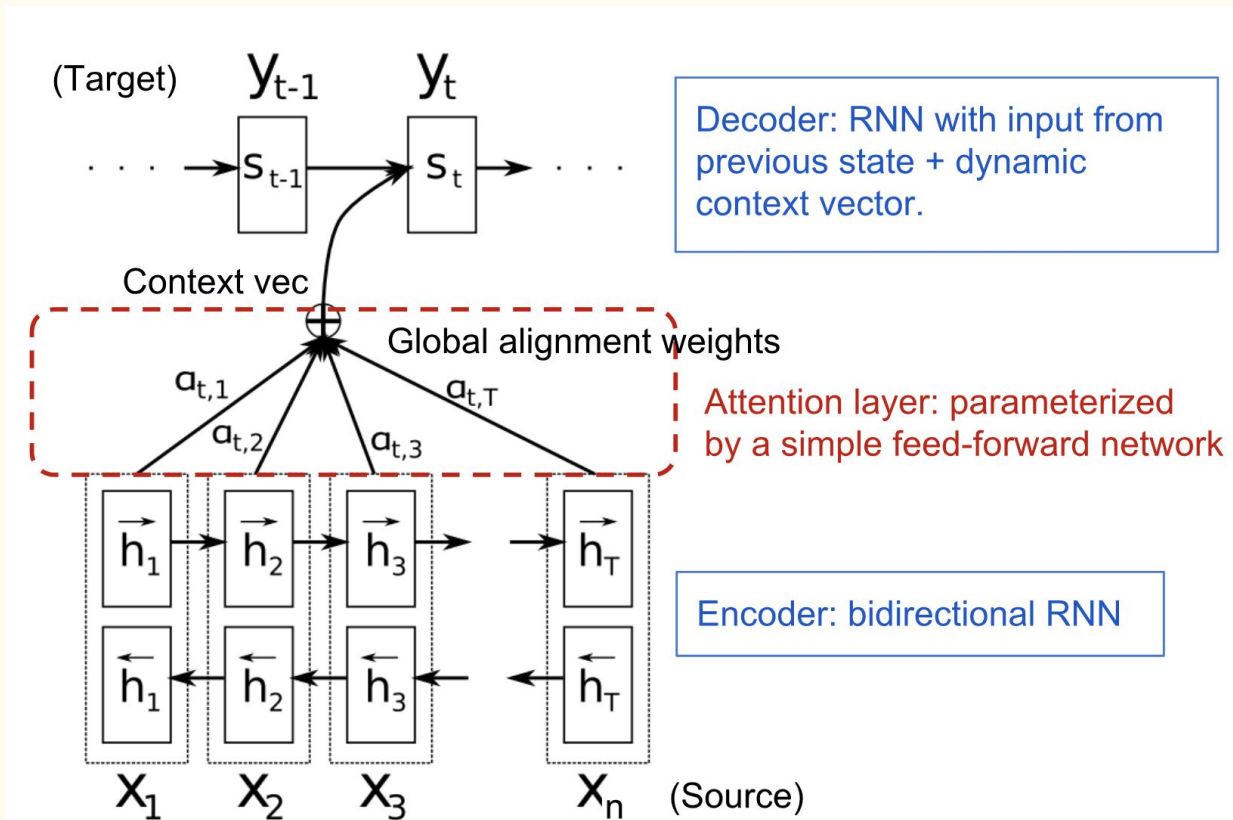


Attention

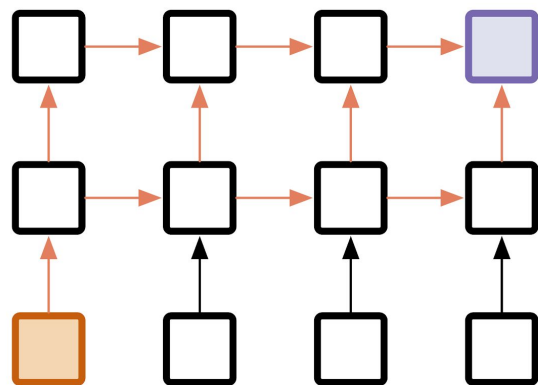
Generating a sentence based on one vector suboptimal: not all inputs relevant to every output

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

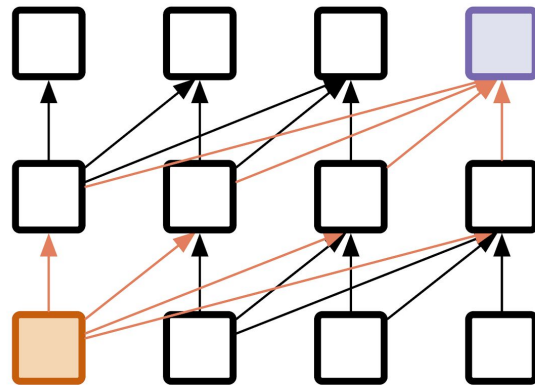
Attention intuitively works as an alignment mechanism, but it is not clear how/why



Self-attention instead of recurrence



RNN



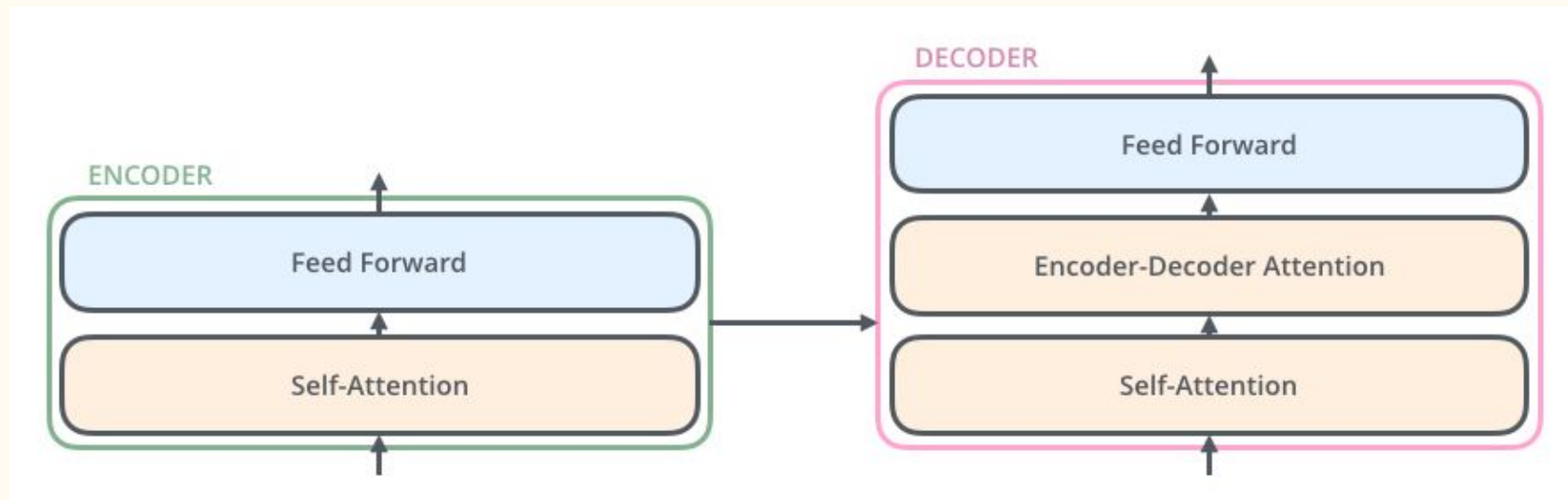
Transformer

Arriana Bisazza
(AthNLP 2019)

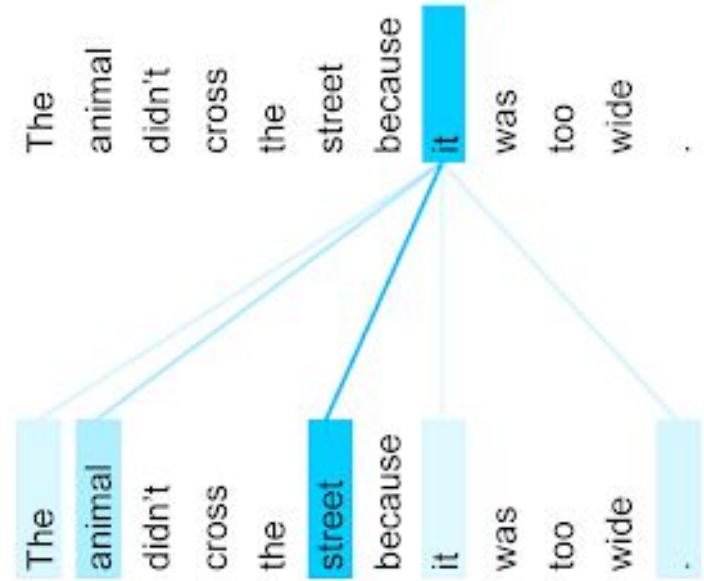
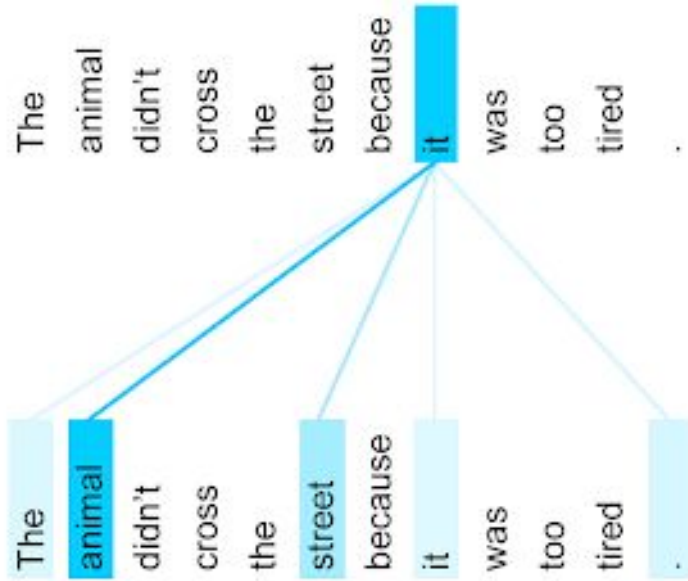
Key idea behind the Transformers (Vaswani et al. 2017)

- 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



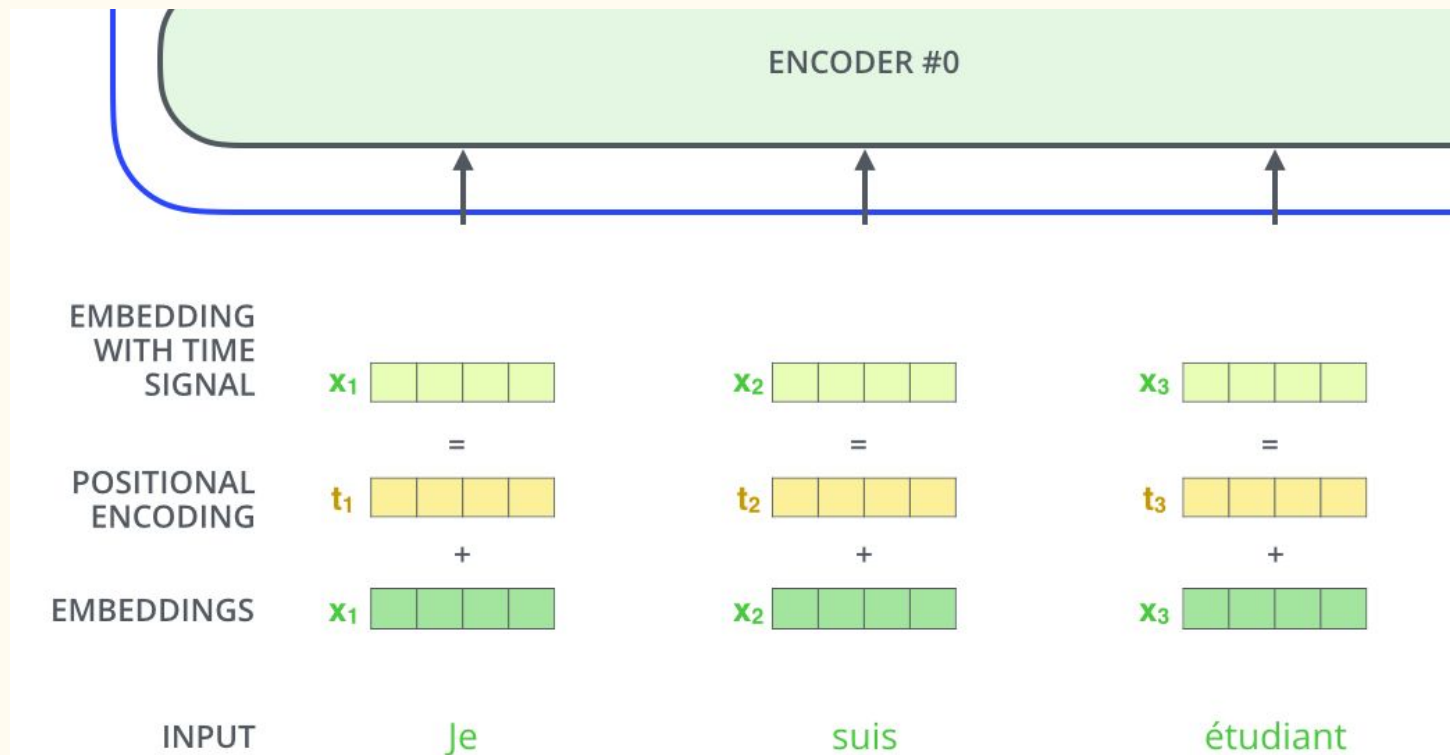
Each layer has multiple attention heads which can be pruned after training

<https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>

Transformers - positional encoding

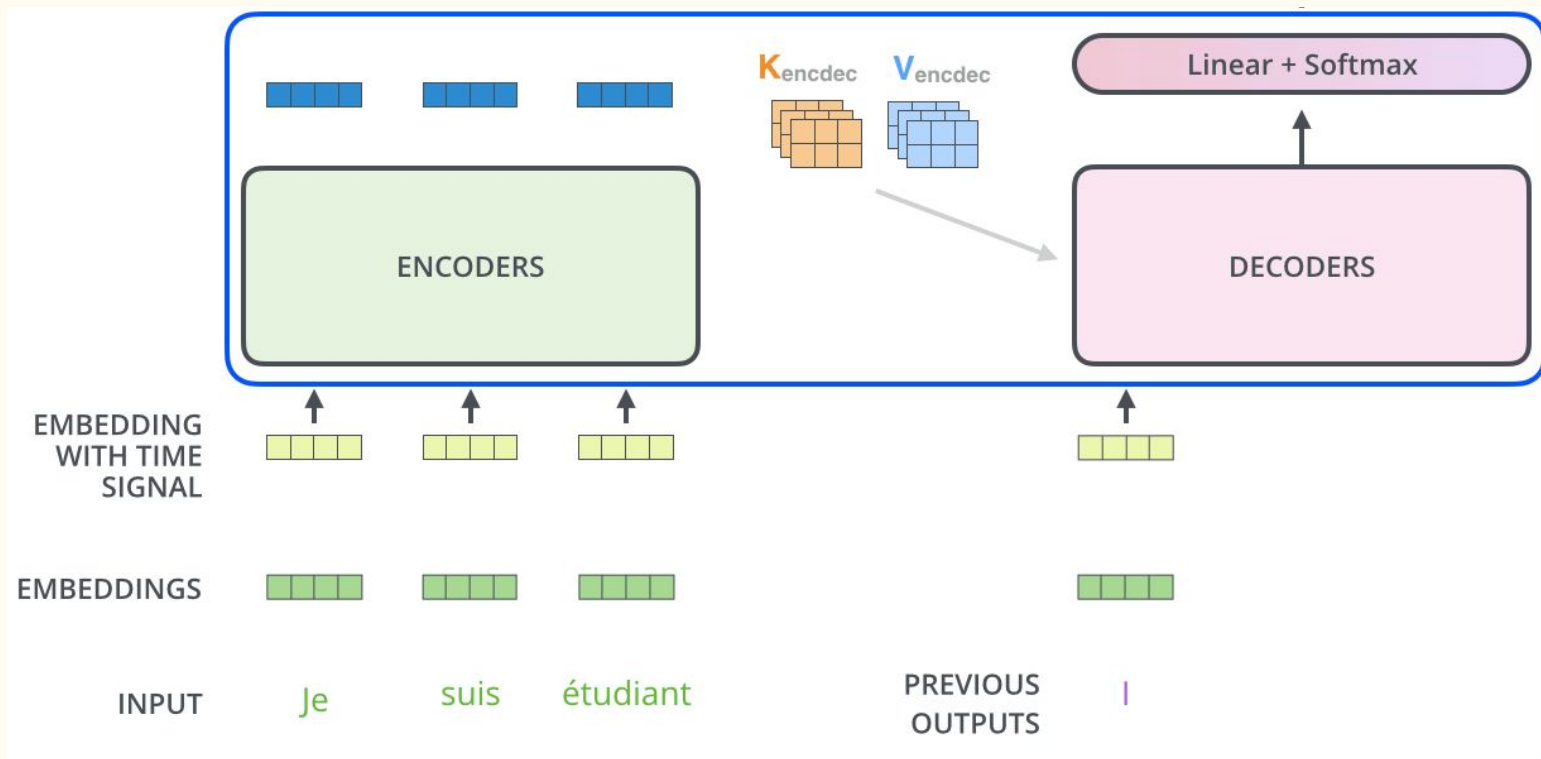
No recurrence, but order is taken into account via the positional embeddings

<http://jalammar.github.io/illustrated-transformer/>



Transformers - decoder

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



<http://jalamar.github.io/illustrated-transformer/>

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

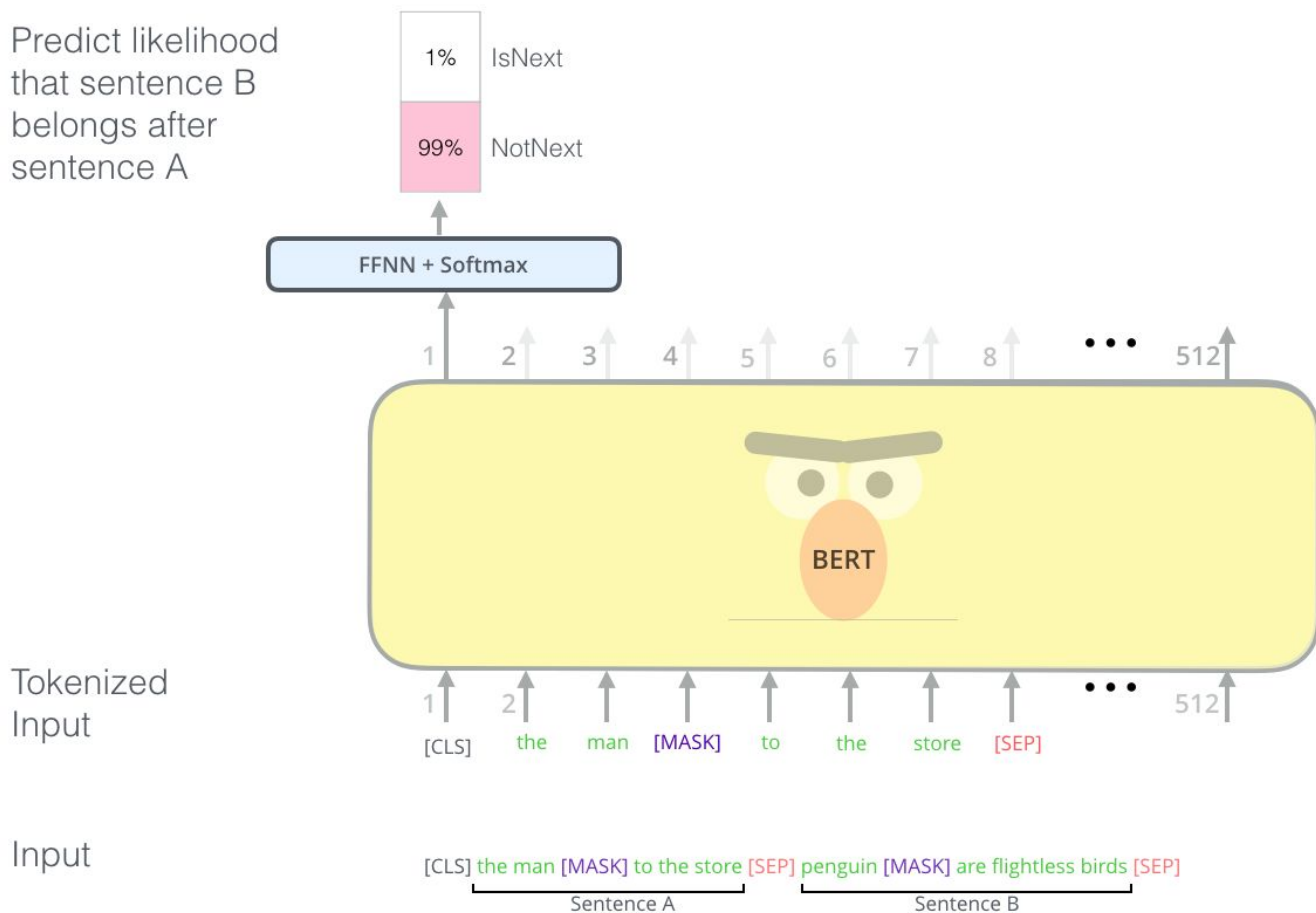
BERT

Task 1: masked language modelling

Task 2: next sentence prediction

<http://jalamar.github.io/illustrated-bert/>

Predict likelihood that sentence B belongs after sentence A



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](#)

Sometimes, I don't know if a sentence is grammatical, so I ask the Internet.

N-GRAM LANGUAGE MODELING, AS SHE IS IMPLEMENTED.

But storing the entire internet would require at least, like, 20 computers, which is more than the one I currently own. So instead, I ask the internet for ALL THE N-GRAMS.

We have addressed the problem of sparse N-grams

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

What are "n-grams"?

A sequence of n words in a sentence. For example, "quixotic generative model" would be a 3-gram.

But T-Rex, a sequence model can't possibly capture all the complexities of human language. We know that language is at least context-free, and in the case of Swiss German, we know that even a context-free grammar isn't powerful enough to model some constructions.

Sometimes, even a truly awesome n-gram might just not have been said yet.

Just the other day I asked the Google about the sentence "Batman high-fived Superman" and the Googs found nothing! I DO NOT TRUST ANY MODEL THAT DOUBTS THE GRAMMATICALITY OF THAT SENTENCE.

That's it?

grammar isn't powerful enough to model some constructions.

It's even worse than that!