Natural Language Processing and Transformers 🤖

John Giorgi

March 8th, 2022

CSC413/2516

The big picture

- Natural language (& speech) are central to human intelligence
- **N**atural **l**anguage ***p****rocessing* (NLP) attempts to capture some of this intelligence algorithmically and is of huge practical importance
  - machine translation, chatbots, automatic fact checking, …
- NLP has seen several transformative shifts in the last few years

Goal of this tutorial

- Build basic *NLP literacy* by looking at **language models**
- Get up to date with recent developments
    - **BERT**, **GPT**, **S**elf-**S**upervised **L**earning (**SSL**)
- Know where to look if you're starting an NLP project

Language Models

Language models

Language models (LMs) assign **probabilities** to sequences of words

- **Speech recognition**: $P(\textit{I will be back soonish}) > P(\textit{I will be bassoon dish})$
- **Spell checkers**: $P(\textit{There are two midterms}) > P(\textit{Their are two midterms})$
- **Machine translation**:

他 向 记者 介绍了 主要 内容

$P(\textit{He to reporters introduced main content}) <$

$P(\textit{He briefed reporters on the main contents of the statement}) <$

Language models

Is the product of the conditional probability of each word and its history (chain rule)

$$P(w_{1:n}) = \prod_{k=1}^{n} P(w_k | w_{<k})$$

The probability of a sequence of n words

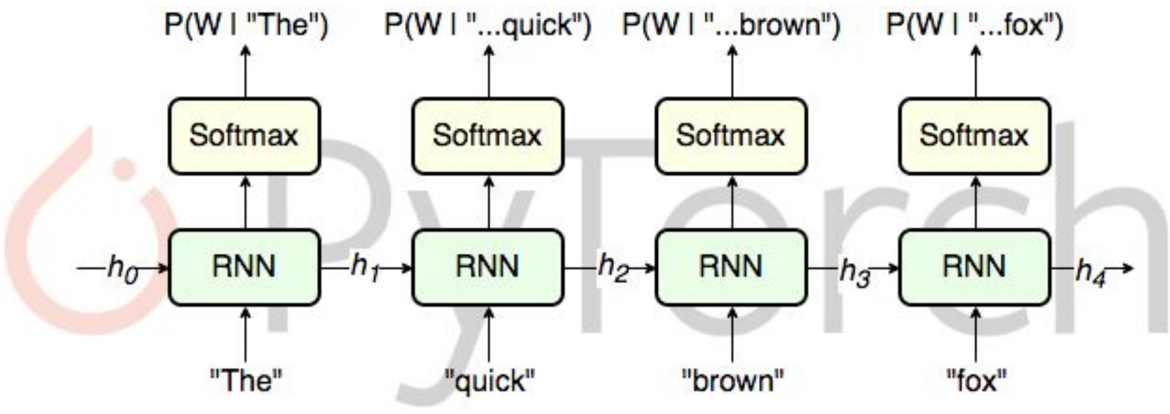
$$= \sum_{k=1}^{n} \log P(w_k | w_{<k})$$

How do we compute the conditional probability?

In practice, we take the log sum

Language models

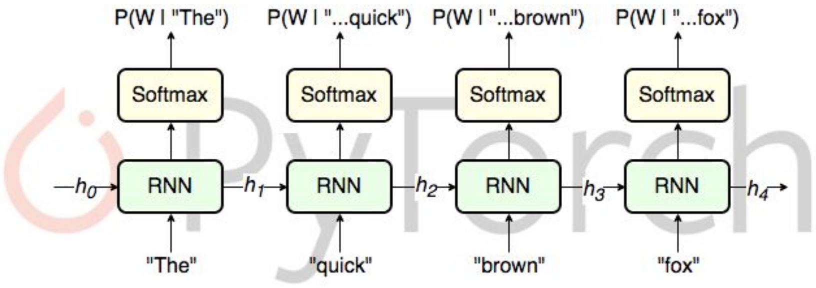
- We can use **recurrent** neural networks (RNNs)! E.g. LSTMS, GRUs
- Work well for variable length inputs, like sentences



Language models

Recurrent neural networks have some shortcomings:

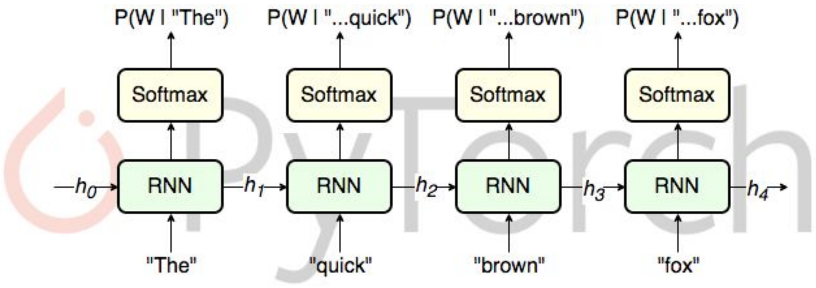
- Not parallelizable within training examples
- Difficult to optimize due to vanishing gradients
- Difficulty modelling long range dependencies



Language models

We'd like an architectural primitive that:

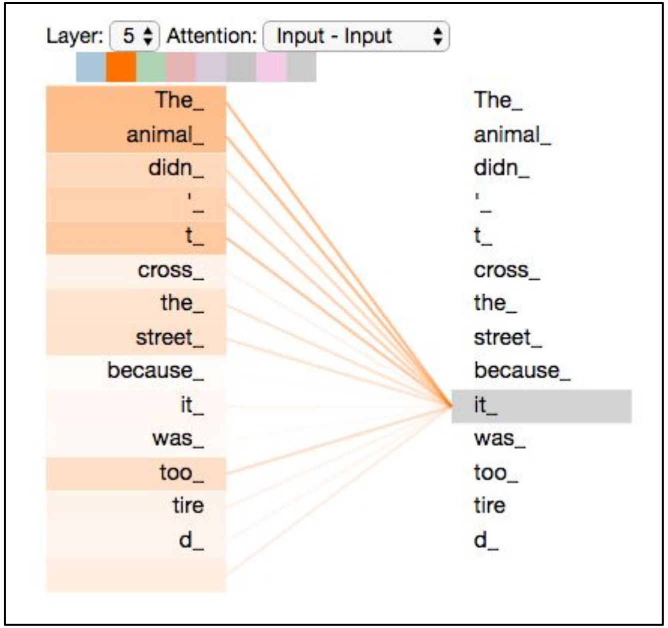
- Is parallelizable within training examples
- Directly facilities interactions between tokens
- Better models long range dependencies
- Attention to the rescue?



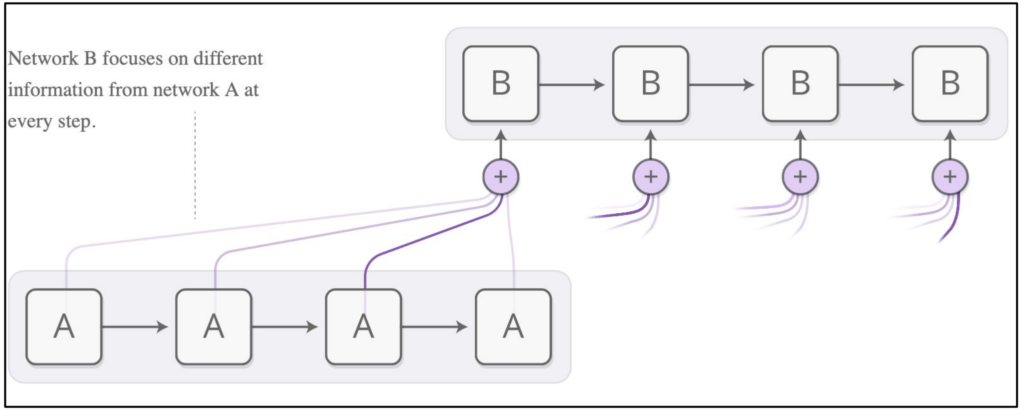
Attention

Attention

Self-attention



Cross-attention



https://jalammar.github.io/illustrated-transformer/
https://distill.pub/2016/augmented-rnns/

Attention

- Many flavours of attention have been proposed
- We will focus on the most common, (**scaled**) **dot-product** attention
- Scaled dot-product attention is the backbone of **transformers**
- Like any attention mechanism, we need to make two decisions:
    - How to compute similarity? → **dot-product**
    - How do we normalize the similarity score? → **softmax**

Scaled dot-product attention

Similarity is simply the dot product between Q and K

Scaled dot-product attention takes three matrices as input

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

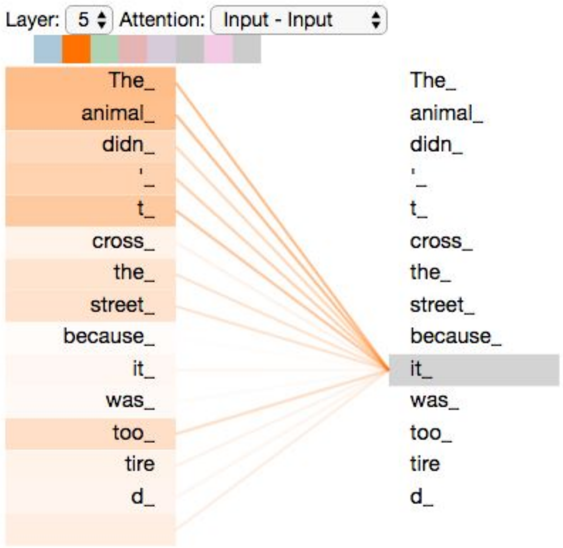
The output is simply a scaling of V

A softmax normalizes similarities → [0, 1]

Queries, keys and what?

These will change depending on how the attention mechanism is used

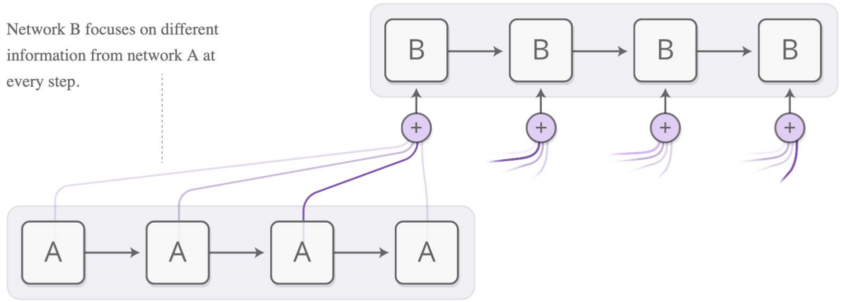
- In **self**-attention, Q == K == V
- *We are updating the representation of each token based on the other tokens in the sequence*



Queries, keys and what?

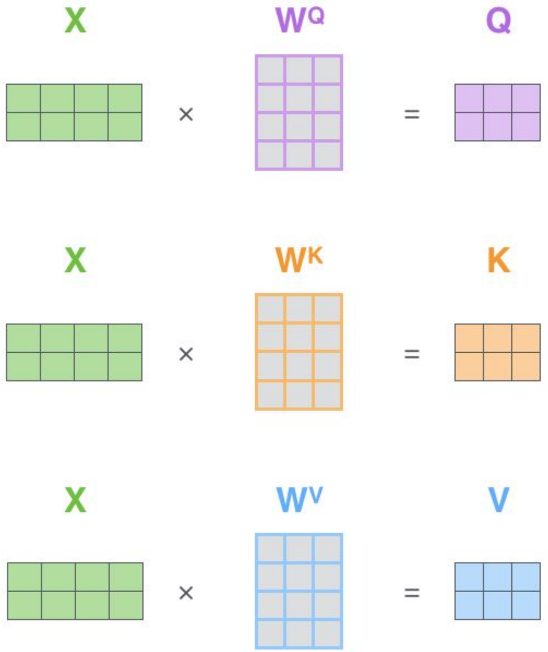
These will change depending on how the attention mechanism is used

- In **cross**-attention, K == V and come from the encoder. Q comes from the decoder.
- *The decoder "focuses" on certain tokens in the encoders output*



Queries, keys and what?

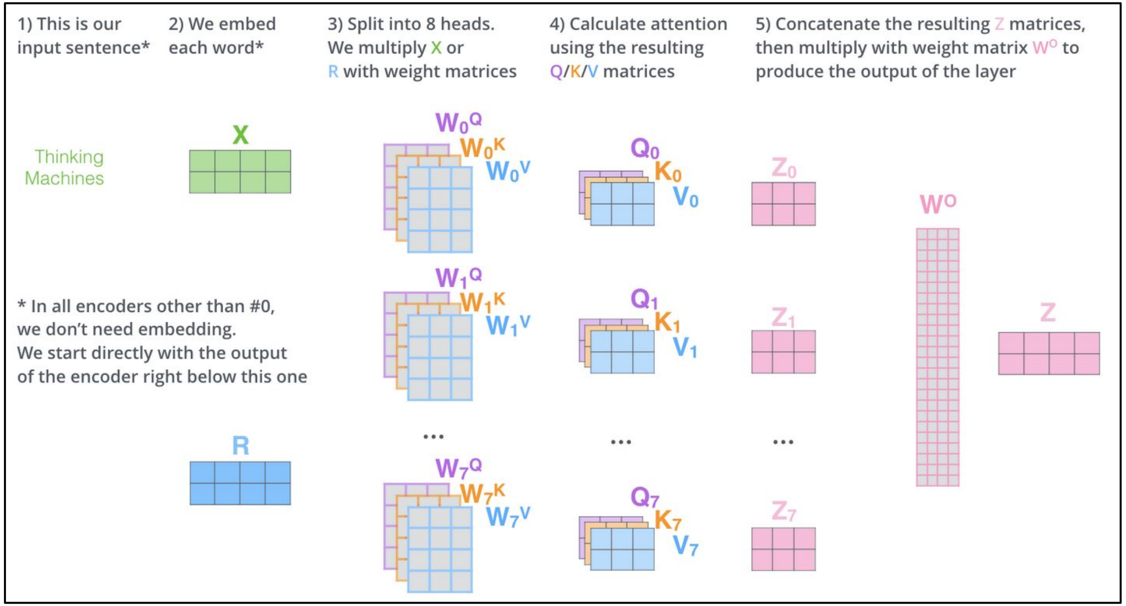
- Q, K & V are *projections* of embedded tokens
- If this is a multi-layered network (e.g. a transformer), they are outputs of the previous layer

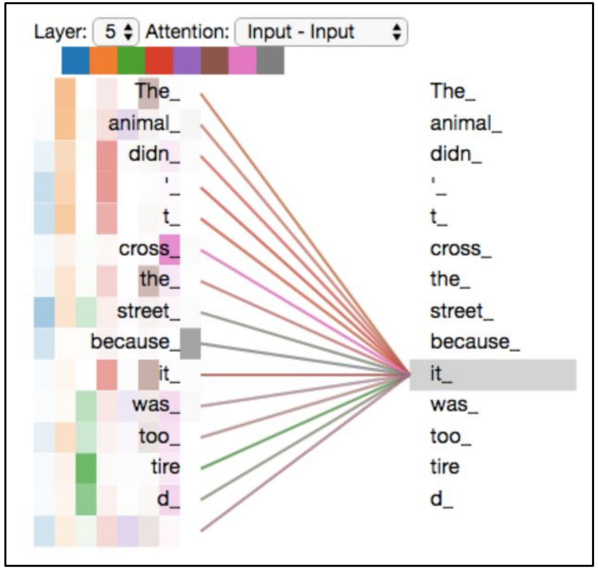


https://jalammar.github.io/illustrated-transformer/

The beast with many heads

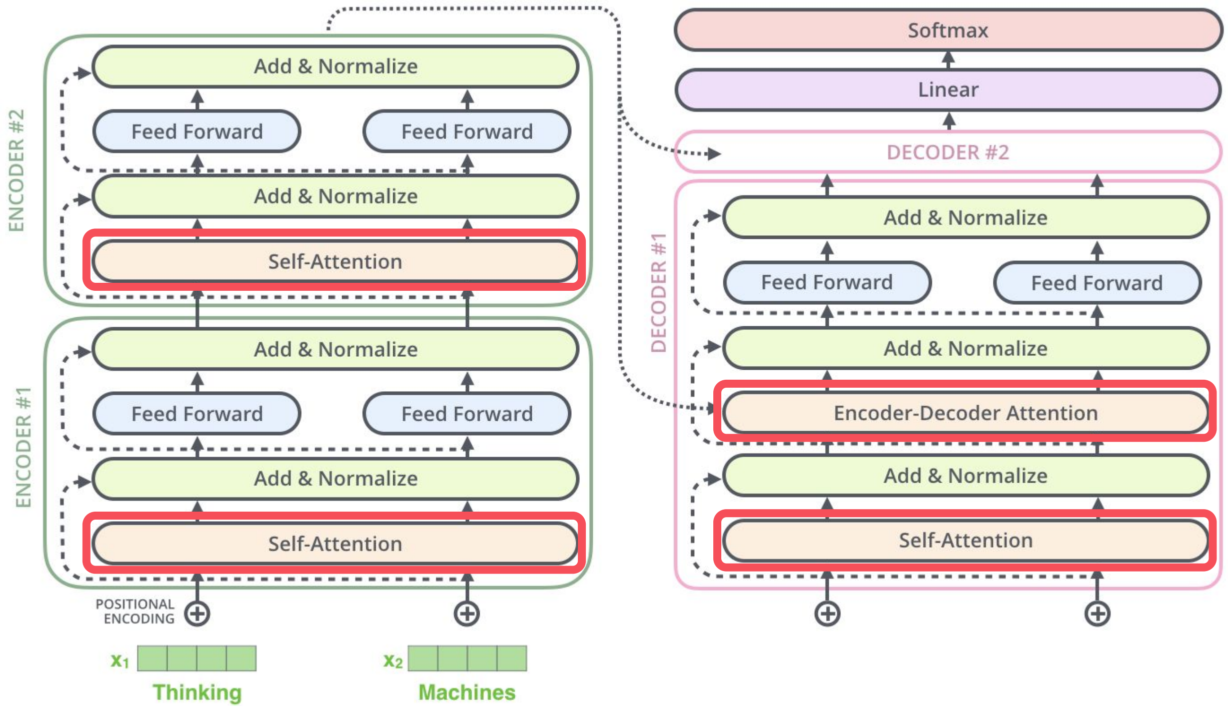
Usually, we use **multi-head** scaled dot-product attention





https://jalammar.github.io/illustrated-transformer/

Transformers (covered in lecture)



https://jalammar.github.io/illustrated-transformer/

The payoff

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

Before we move on...

- LMs assign **probabilities to sequences** and are the "workhorse" of NLP
- Typically implemented with RNNs; being replaced with **Transformers**
- **Multi**-head, **scaled** dot-product attention the backbone of Transformers
  - Allows us to learn long range dependencies and parallelize computation within training examples
- *How do we train Transformers as language models?*

Pretrained Language Models & Self-Supervised Learning

Wishlist

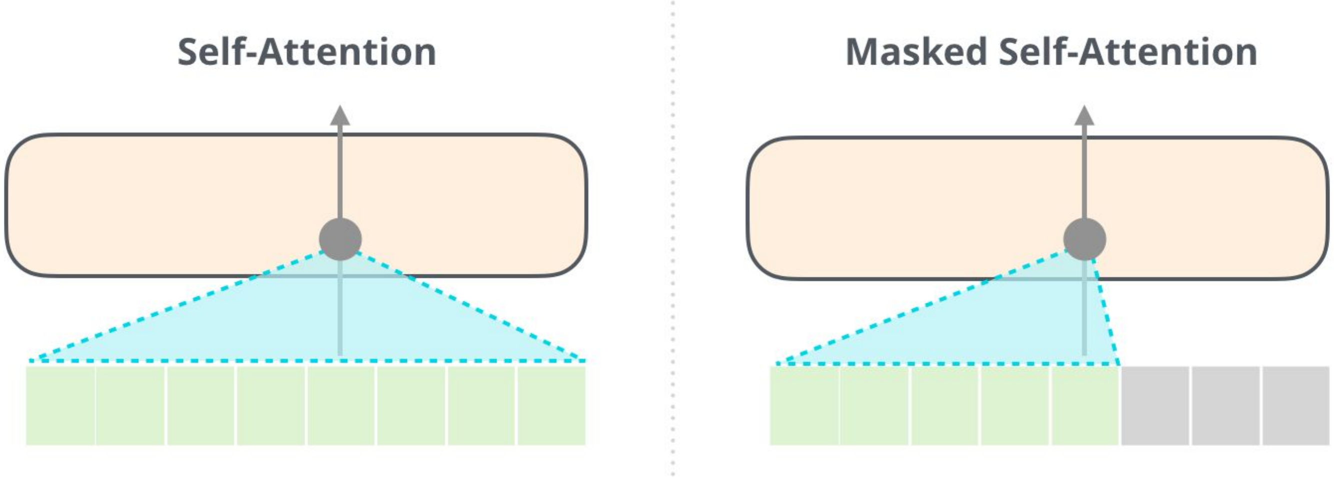
- Want to train transformers as language models → *learn general properties of language that can be transferred to downstream tasks*
- Ideally, we will use unlabeled text → *leverage unsupervised or self-supervised learning (SSL)*
- (At least) two paradigms have emerged:
  - **G**enerative **P**retrained **T**ransformer (**GPT**) → *next-token prediction, decoder only transformer*
  - **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (**BERT**) → *masked language modelling, encoder only transformer*

Generative Pretrained Transformer (GPT)

- **GPT** is a *decoder only* transformer pretrained on huge amounts of text
- The latest version, GPT-3 is trained on 45TB of unlabelled text
- The (pre)training objective is simply to predict the next token
- For this, we will need to slightly tweak the self-attention…

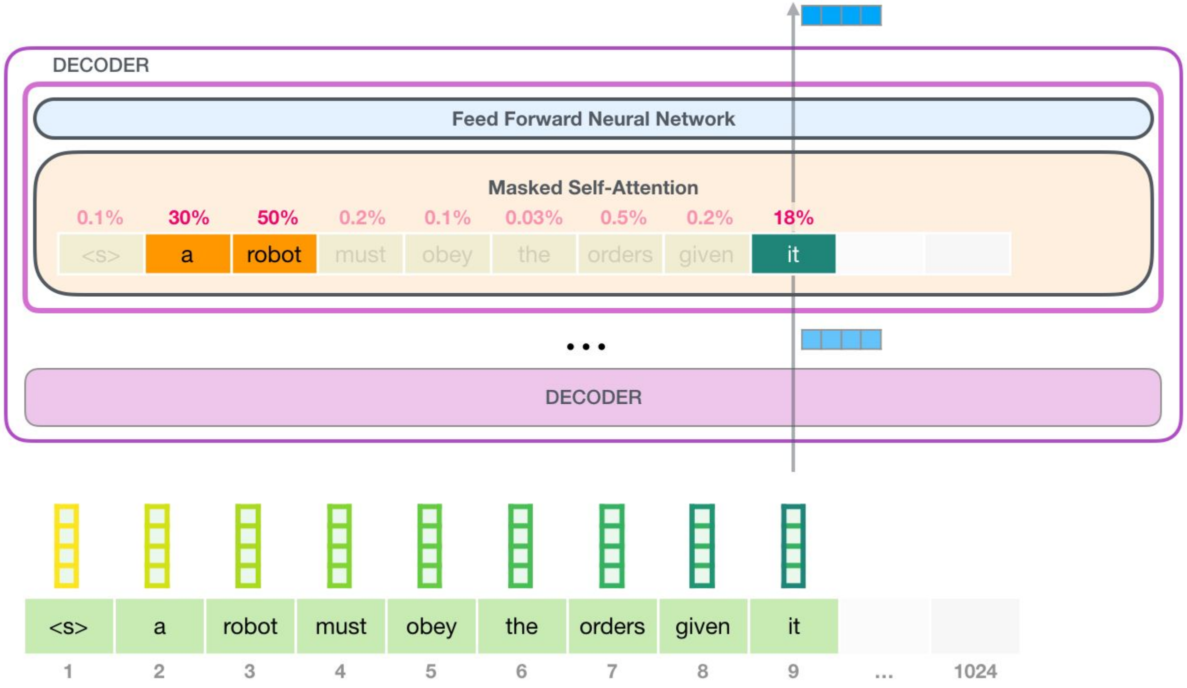
Masked self-attention

Future timesteps are **masked** to prevent decoder from "peaking"



https://jalammar.github.io/illustrated-gpt2/

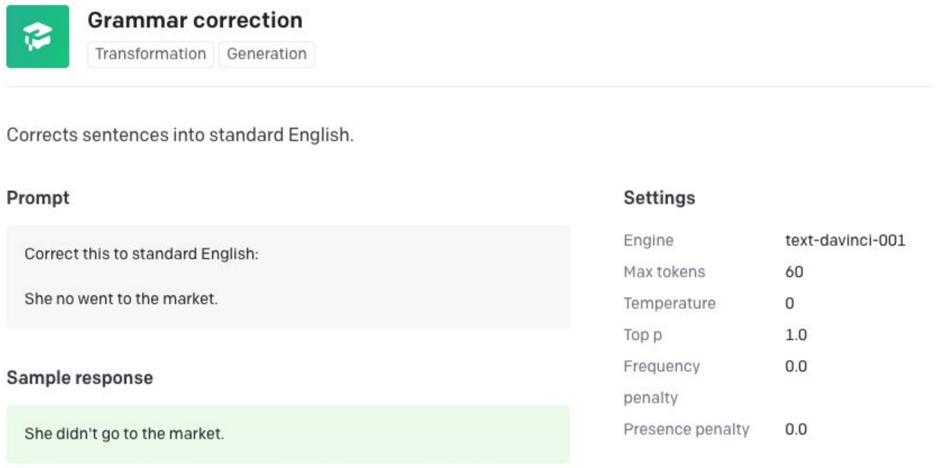
Masked self-attention



https://jalammar.github.io/illustrated-gpt2/

Generative Pretrained Transformer (GPT)

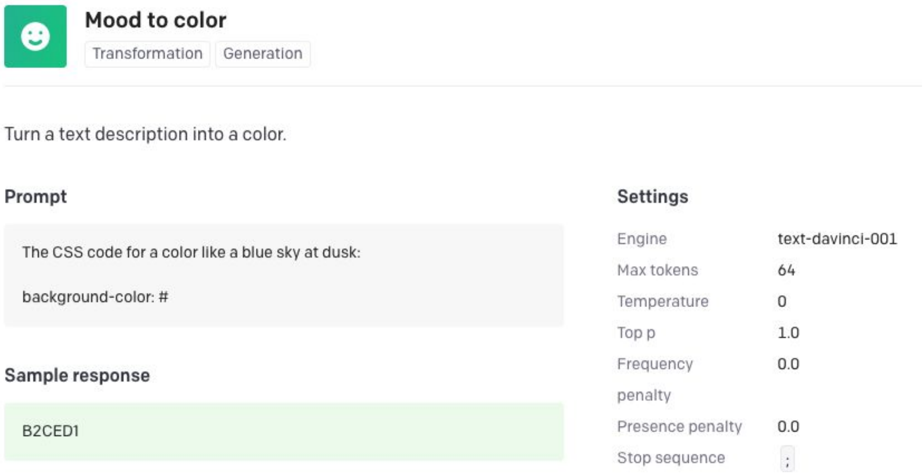
Once pretrained, GPT can be used for any "text in, text out" task



https://beta.openai.com/examples

Generative Pretrained Transformer (GPT)

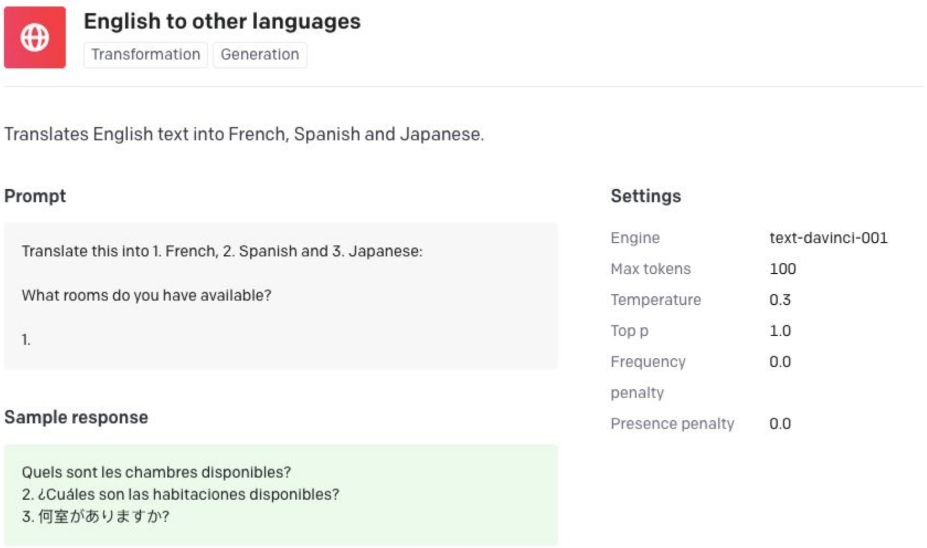
Once pretrained, GPT can be used for any "text in, text out" task



https://beta.openai.com/examples

Generative Pretrained Transformer (GPT)

Once pretrained, GPT can be used for any "text in, text out" task

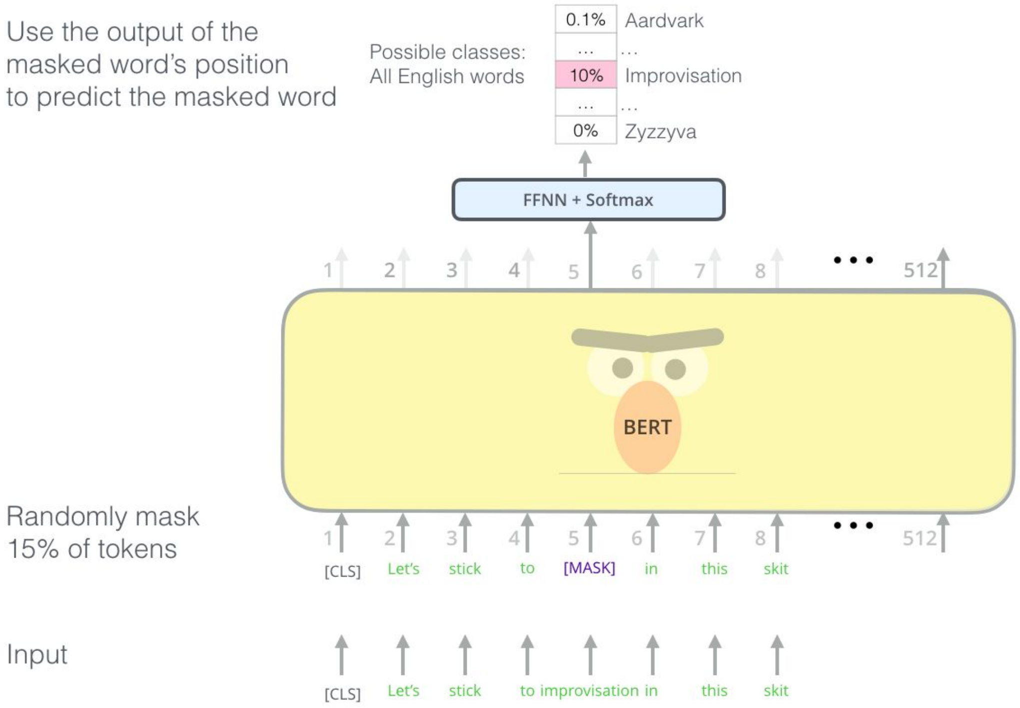


https://beta.openai.com/examples

Bidirectional Encoder Representations from Transformers (BERT)

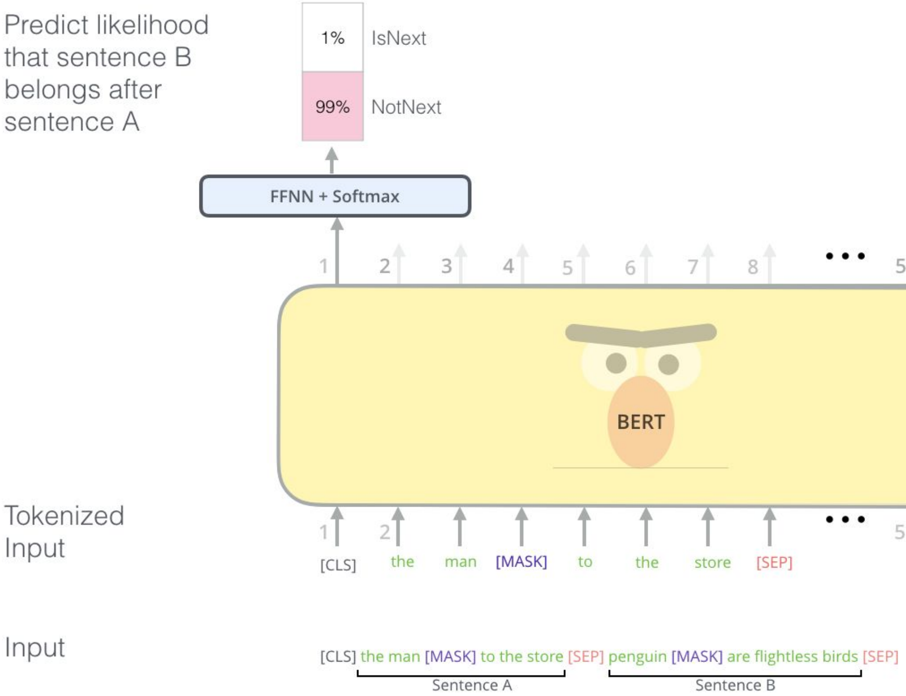
- GPT is a **unidirectional** LM, incorporating context from *previous* tokens
- This is likely sub-optimal for many token- or sentence-level tasks
- BERT proposes a **bidirectional** LM based on a transformer encoder
- BERT is pretrained with two self-supervised objectives:
  - **M**asked **L**anguage **M**odelling (**MLM**)
  - **N**ext **S**entence **P**rediction (**NSP**)

Masked language modelling



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

Next sentence prediction



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

BERT fine-tuning

- BERT can easily be fine-tuned on downstream tasks
  - Add & initialize a new layer on top of BERTs outputs
  - Use supervised learning to tune all parameters (pretrained & newly initialized)
- BERT has learned rich representations which encode **syntax** & **semantics** → fine-tuning BERT is the best performing approach across a wide range of tasks.
- Because BERT is pretrained, fine-tuning is typically cheap
  - 3-4 epochs on 100s or 1000s of labelled examples
  - Typically takes a few hours to fine-tune on GPU(s)

Resources

- For pretrained models, datasets try HuggingFace
- For NLP specific machine learning library, try AllenNLP
- For a great free textbook, try Speech and Language Processing
- For a great MOOC, try Sequence Models (free with UofT Coursera)
- For great blog posts illustrating these concepts, try https://jalammar.github.io/

Thank you for your attention! (get it?)

John Giorgi

March 8th, 2022

CSC413/2516