NLP Fundamentals

Tutorial on Legal IR and NLP -- ECIR 2023

Pawan Goyal
Department of CSE, IIT Kharagpur
http://cse.iitkgp.ac.in/~pawang

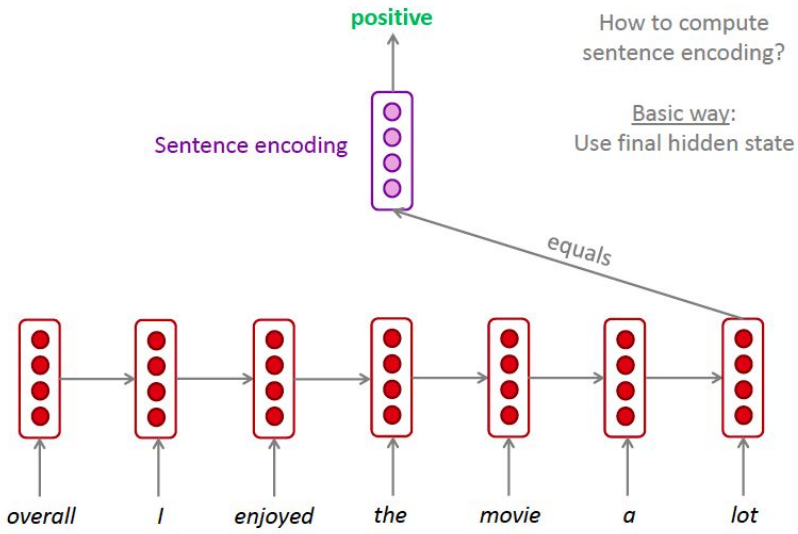
What is NLP?

- Making computers understand what we write
- Making computers write

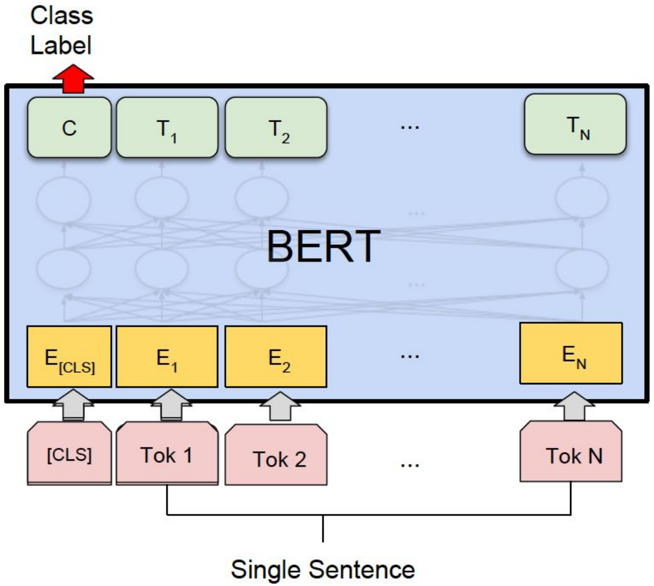
<u>We try to map NLP problems to various paradigms</u>

- Next word prediction, sentence completion → *language modeling*
- Sentiment Analysis, news article groupings, etc. → *text classification*
- Named entity recognition, language identification (code-mixed), parts-of-speech tagging, etc. → *Sequence Labeling*
- Machine Translation, abstractive summarization, chatbots, etc. → *Text generation*
- Question Answering (Reading Comprehension) → *Span Prediction*

Some Popular Deep Learning Methods



RNNs/LSTMs for Sentence classification



BERT (Pretrained Transformer) for Sentence classification

Quick Recap

- How to represent individual tokens
  - Word vectors
- Popular architectures for text (sequence) input:
  - RNNs (LSTMs)
  - Transformers
- Petraining
  - ELMo
  - BERT
  - T5, BART
  - GPT

Word Representation

In traditional NLP / IR, words are treated as discrete symbols (one-hot)

Why is this a problem?

- Vector dimension = number of words in vocabulary (e.g., 500,000)! Huge!
- No natural notion of similarity between two one-hot vectors!

Distributions Hypothesis:

Words that occur in the same contexts tend to have similar meanings." (Zellig Harris, 1968)

Word2Vec: Distributional Representation

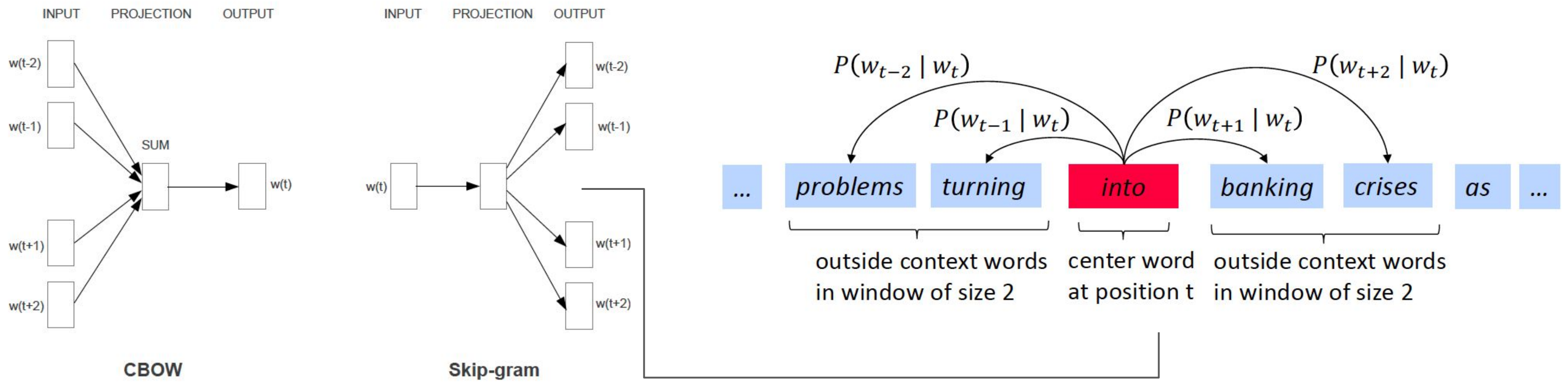
Distributional representation – word embedding?

Any word w_i in the corpus is given a distributional representation by an embedding

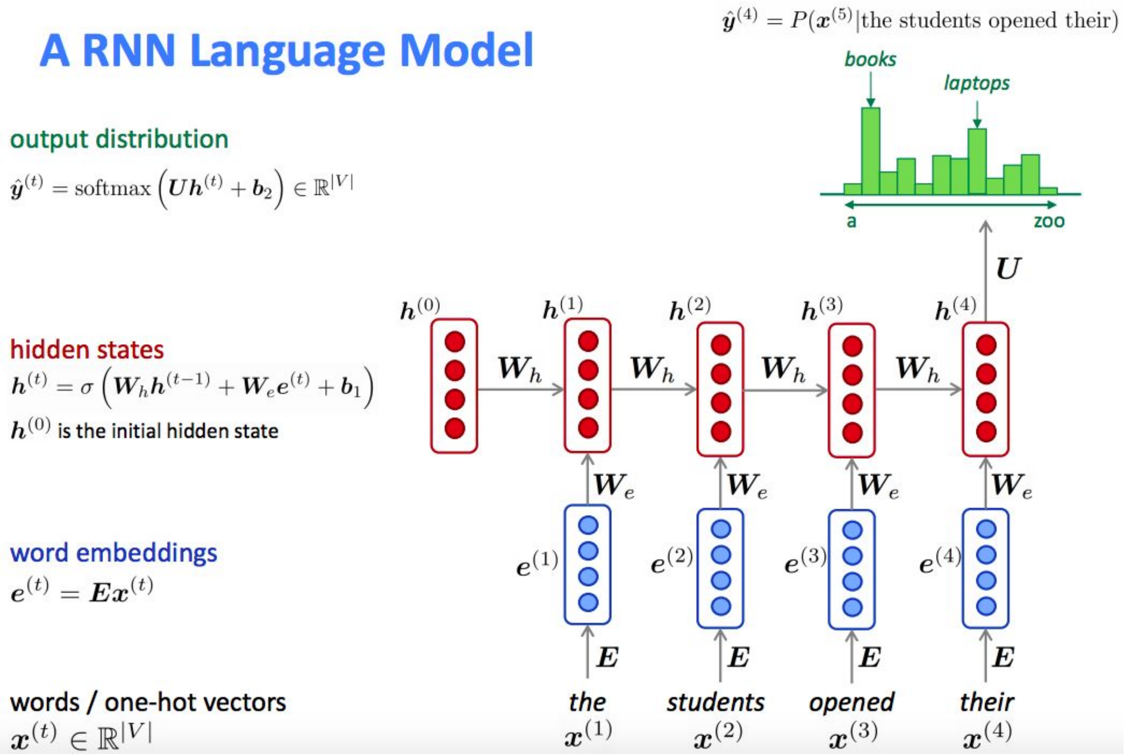
$$w_i \in R^d$$

i.e., a $d-$dimensional vector, which is mostly learnt!

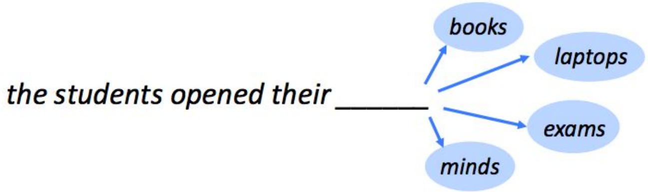
$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$



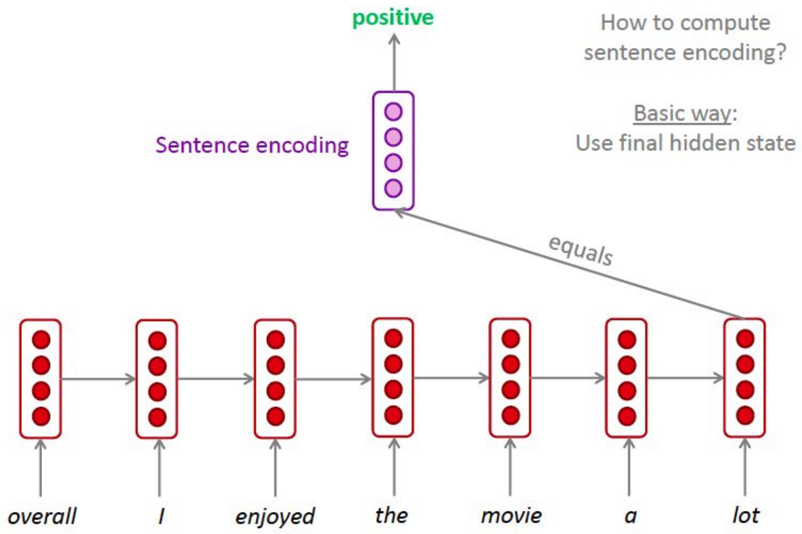
Recurrent Neural Networks with word embeddings

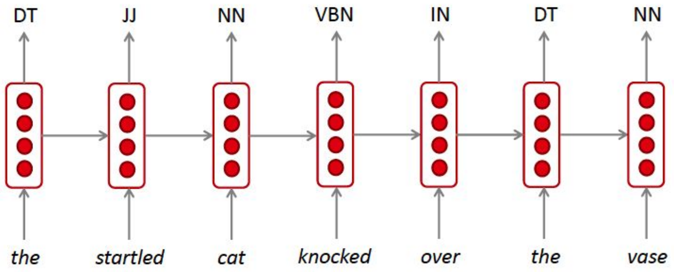


Language Modeling is the task of predicting what word comes next

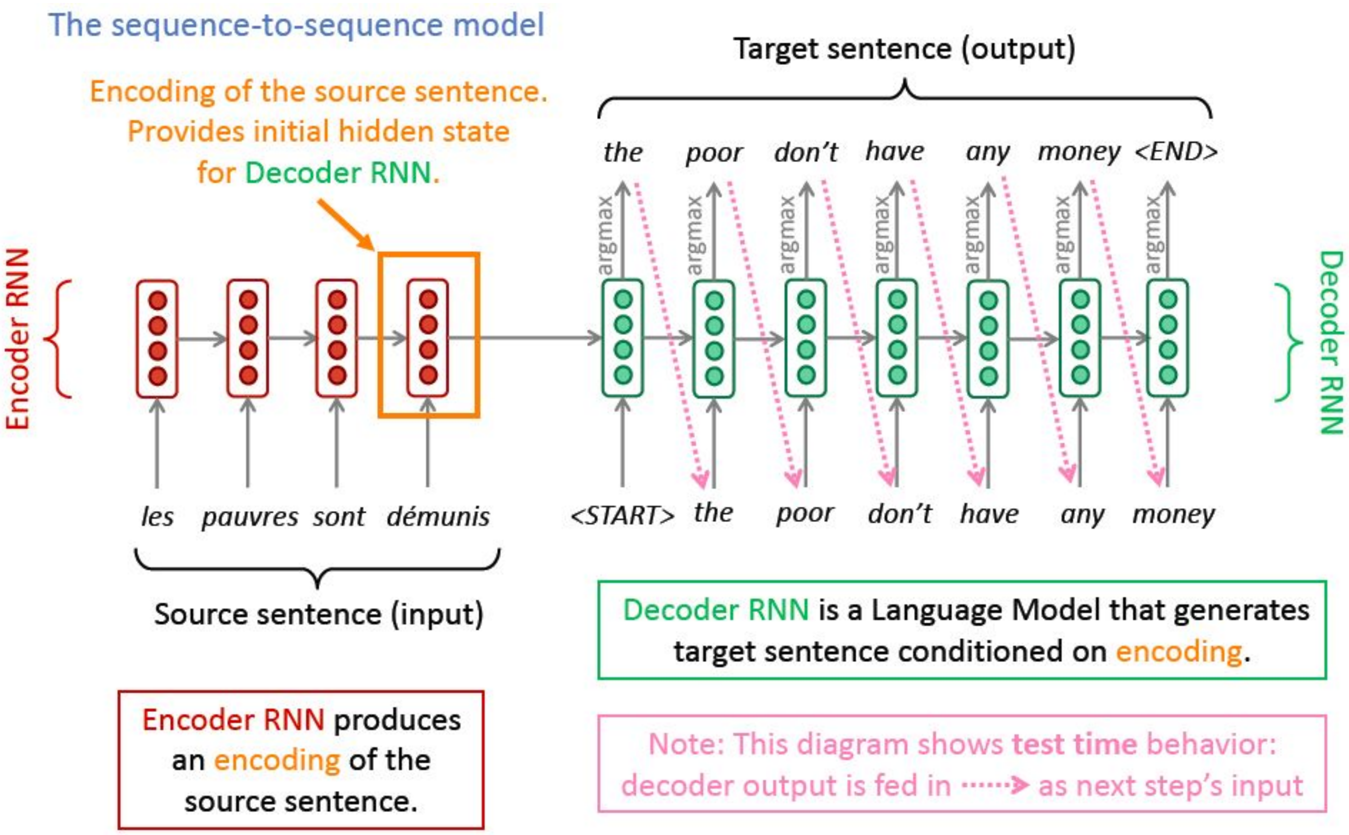


RNNs can be used for classification, sequence labeling





RNNs can be used for text generation



Sequence to Sequence (seq2seq) is optimized as a single system → <u>back-propagation operates end-to-end</u>

The need for fancy units: GRUs, LSTMs

Vanishing Gradient Problem with RNNs

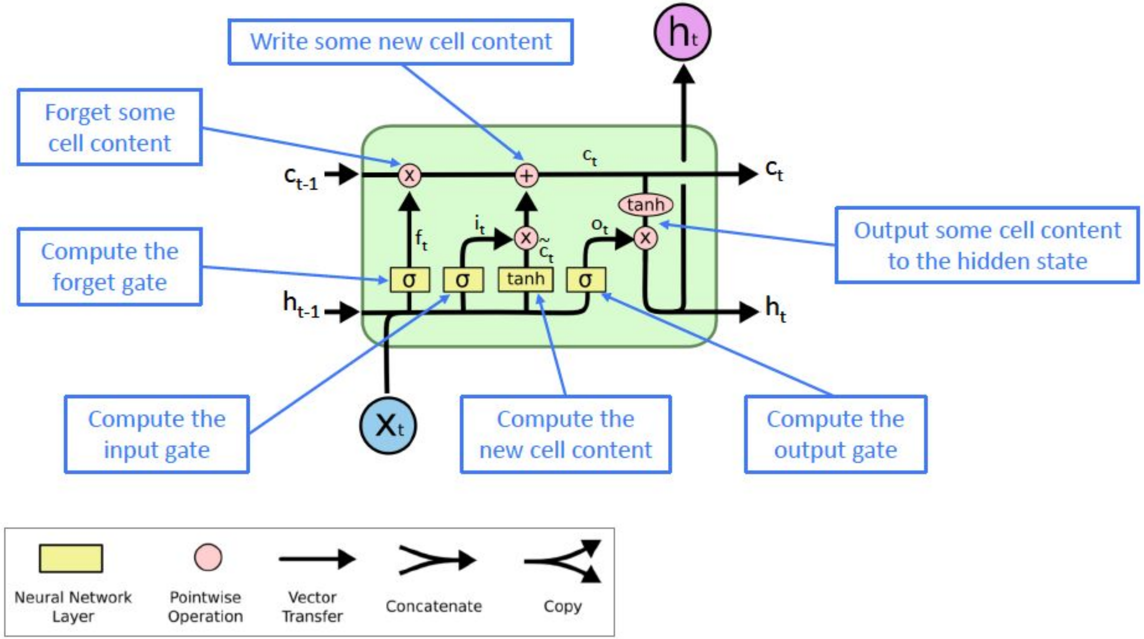
- The main problem is that it is too difficult for the RNN to learn to preserve information over many timesteps.
- In a vanilla RNN, the hidden state is constantly being rewritten

$$h^{(t)} = tanh(Wh^{(t-1)} + Ux^{(t)} + b)$$

Use Gates: GRUs, LSTMs

- The gates are also vectors. On each timestep, each element of the gates can be open (1), close (0) or somewhere in-between.
- The gates are dynamic: their value is computed based on the current context.

Long Short Term Memory (LSTM)



Sigmoid function: all gate values are between 0 and 1

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

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

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

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

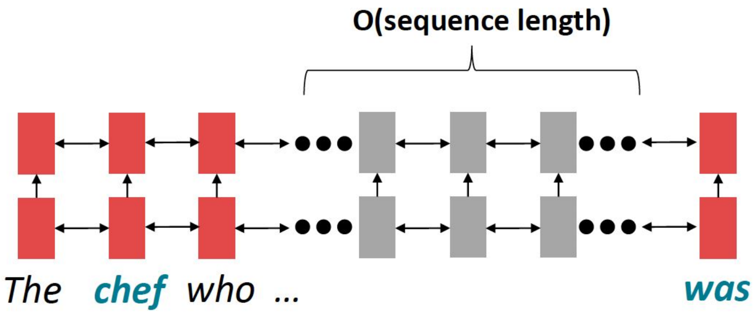
$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

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

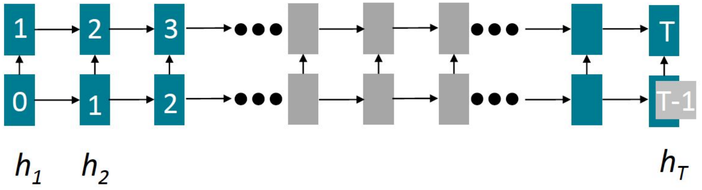
All these are vectors of same length *n*

Gates are applied using element-wise product

Issues with Recurrent Models



RNNs take *O(sequence length)* steps for distant word pairs to interact.



Numbers indicate min # of steps before a state can be computed

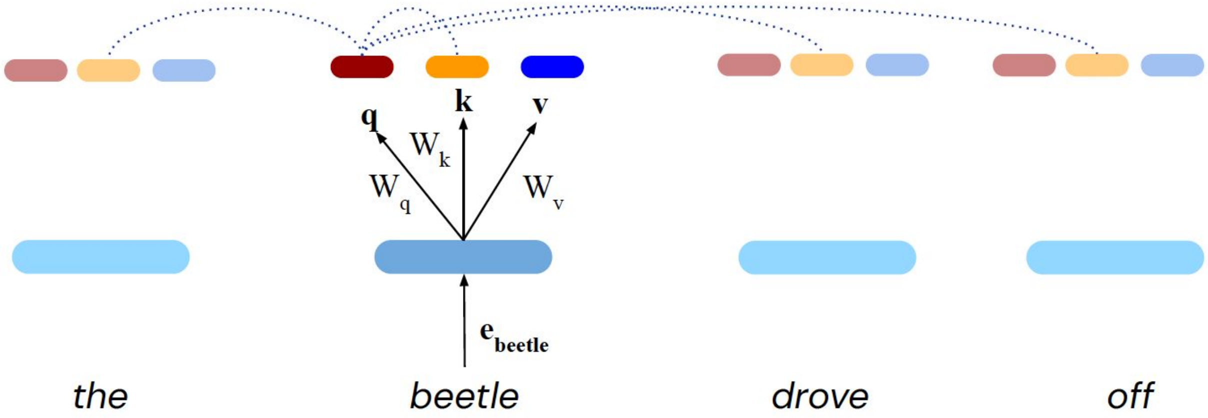
Lack of parallelizability: Future RNN hidden states can't be computed in full before past RNN hidden states have been computed

Transformers: Self-attention over input embeddings

$$\mathbf{q} = \mathbf{e}_{beetle}\, W_q$$

$$\mathbf{k} = \mathbf{e}_{beetle}\, W_k$$

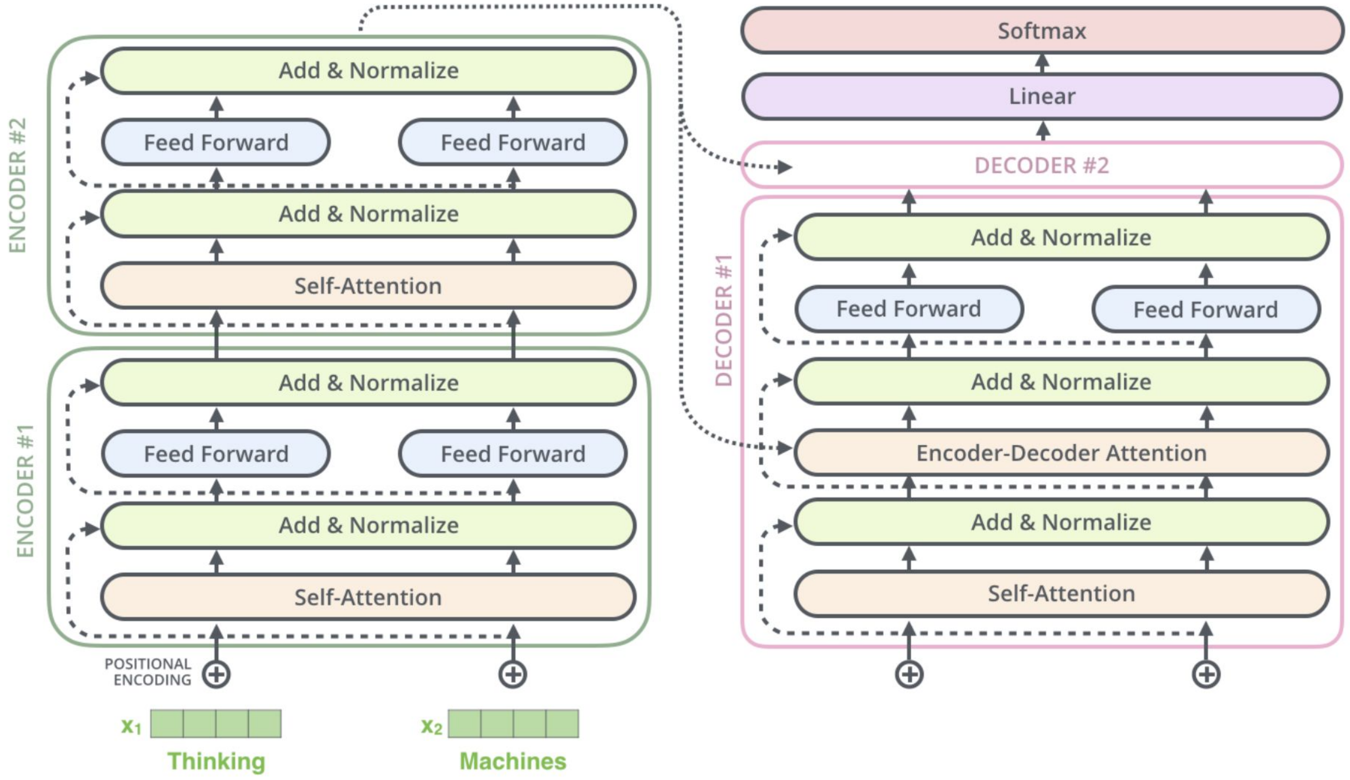
$$\mathbf{V} = \mathbf{e}_{beetle}\, W_v$$



<u>Many other tricks:</u>

- Multi-headed self-attention
- Positional encoding
- Skip-connections
- Layer normalization
- Masked self-attention

Seq2Seq with Transformers

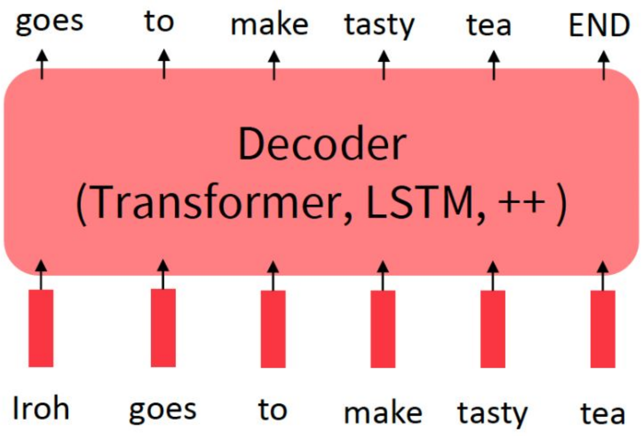


Enter the pretrain/finetune paradigm!

Pretraining can improve NLP applications by serving as parameter initialization.

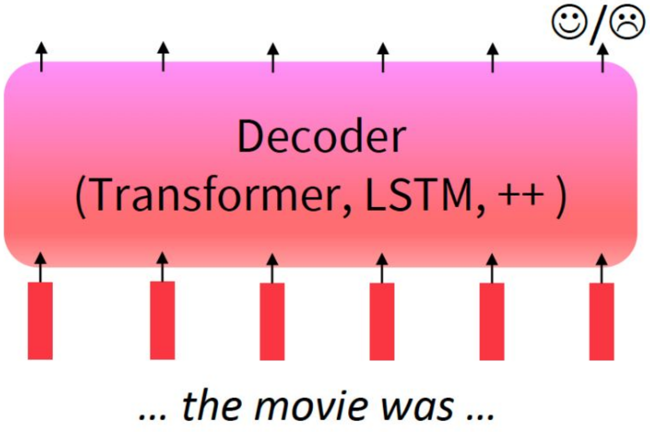
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

Not many labels; adapt to the task!



Pretrained LSTMs: ELMo

1- Concatenate hidden layers

2- Multiply each vector by a weight based on the task

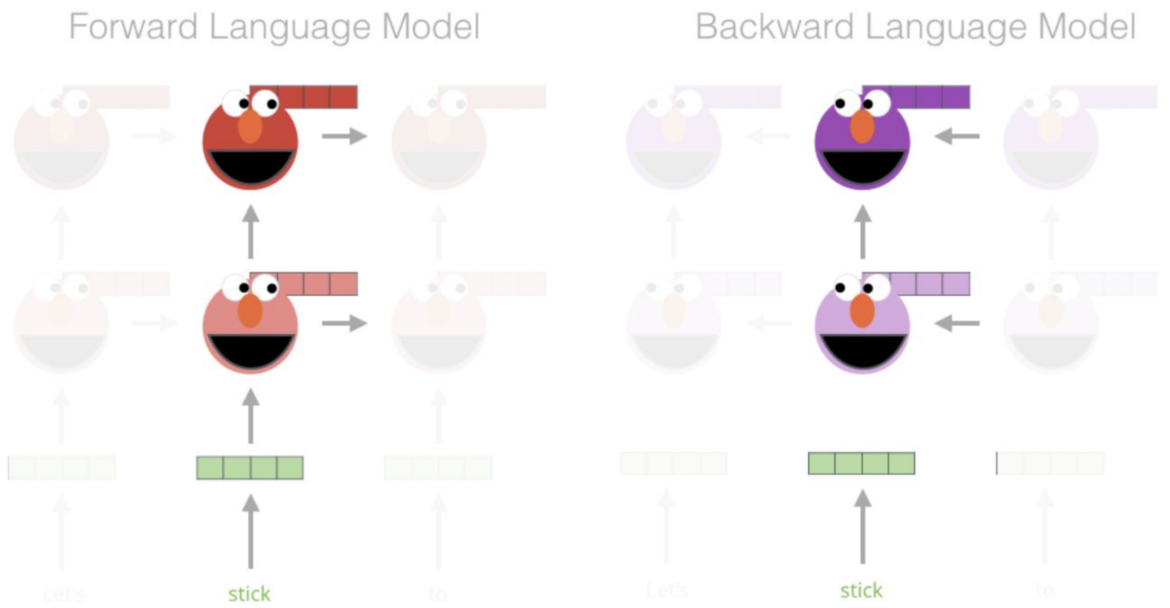
x s_2

x s_1

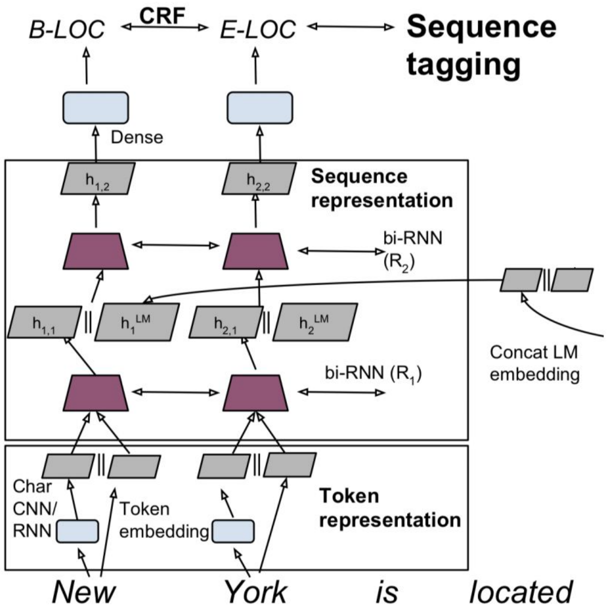
x s_0

3- Sum the (now weighted) vectors

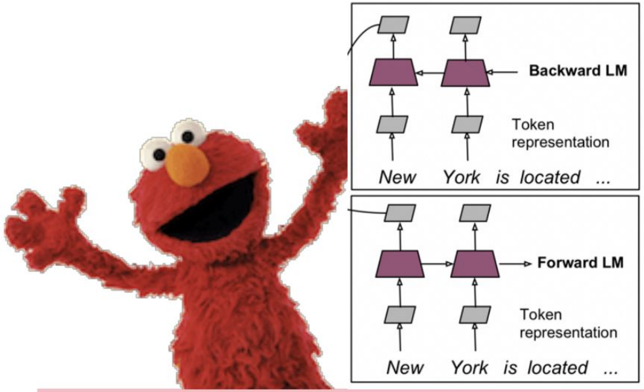
ELMo embedding of "stick" for this task in this context



Using ELMo for a task



$$\mathbf{h}_{k,1} = [\overrightarrow{\mathbf{h}}_{k,1}; \overleftarrow{\mathbf{h}}_{k,1}; \mathbf{h}_k^{LM}].$$

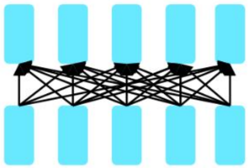


ELMo representation:
A deep bidirectional neural LM

Use learned, task-weighted average of (2) hidden layers

Pretraining Transformers

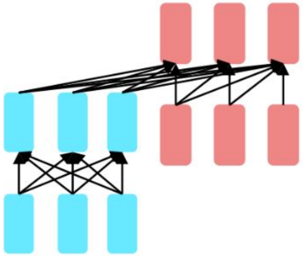
The neural architecture influences the type of pretraining, and natural use cases.



Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?

BERT, RoBERTa

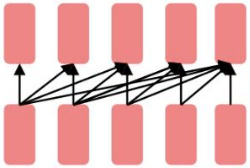


Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

T5, BART

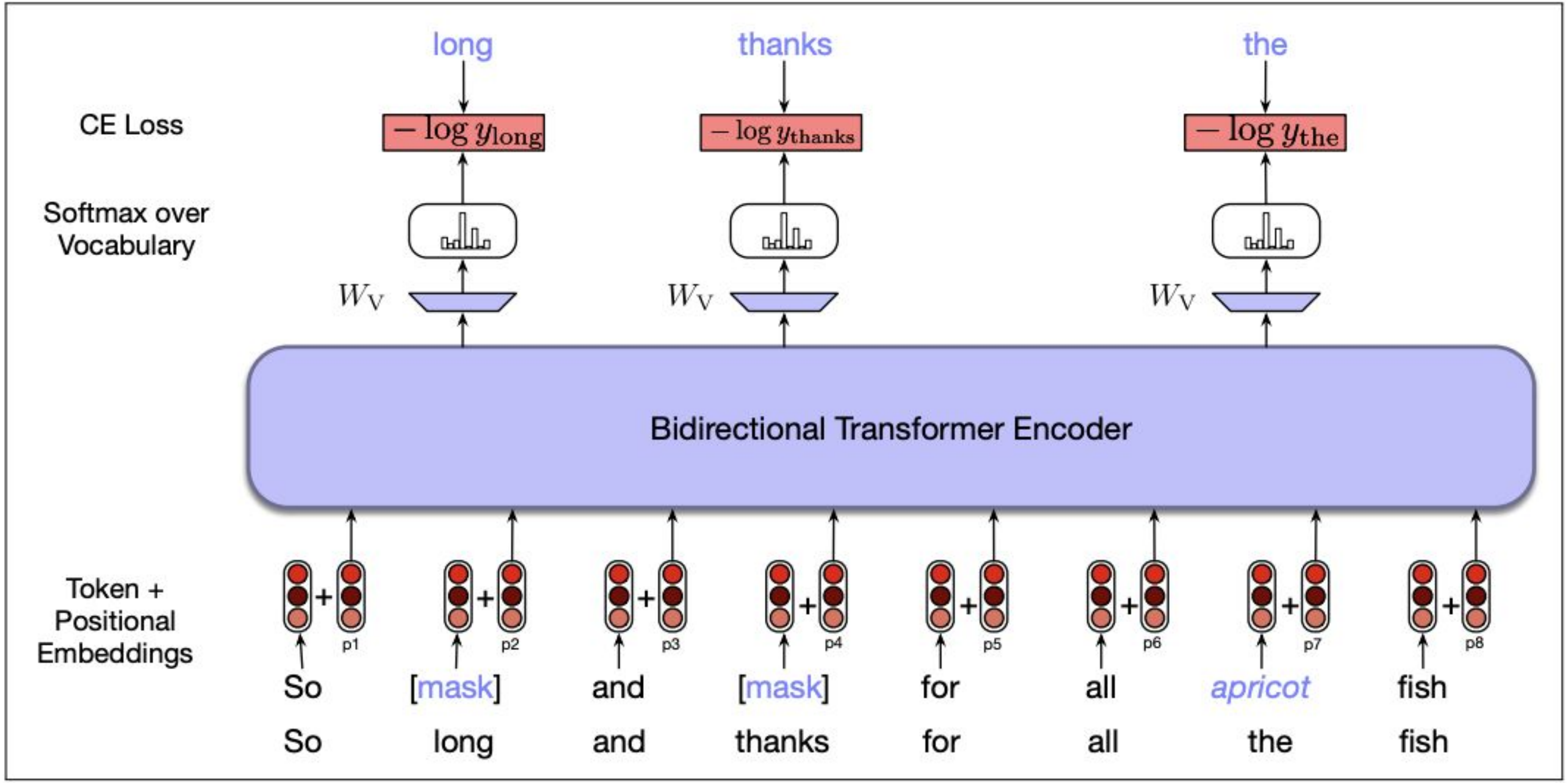
GPT family



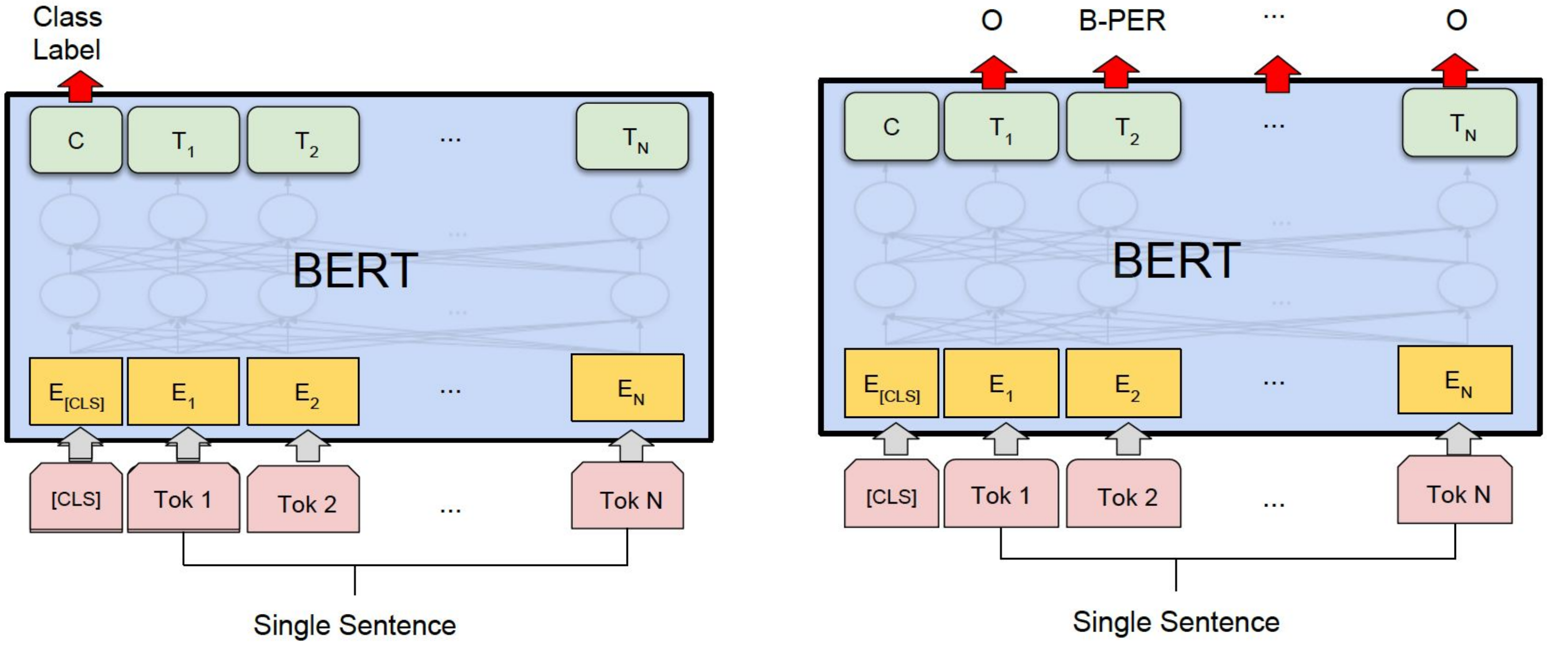
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

BERT (Pretrained Transformer Encoder)



BERT for Text Classification, Sequence Labeling



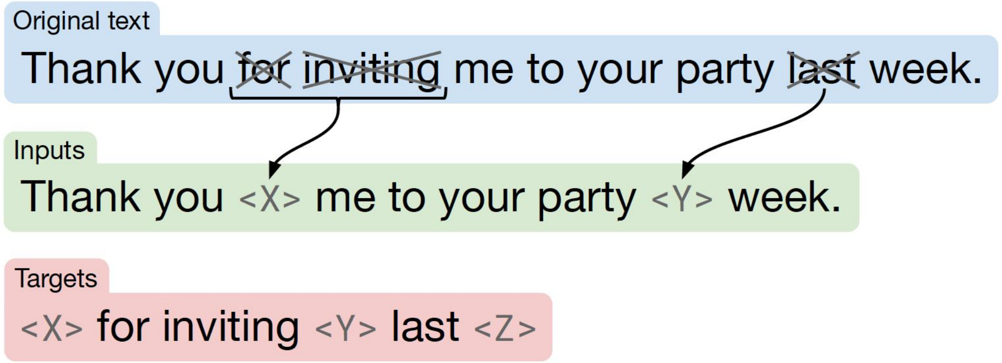
(b) Single Sentence Classification Tasks: SST-2, CoLA

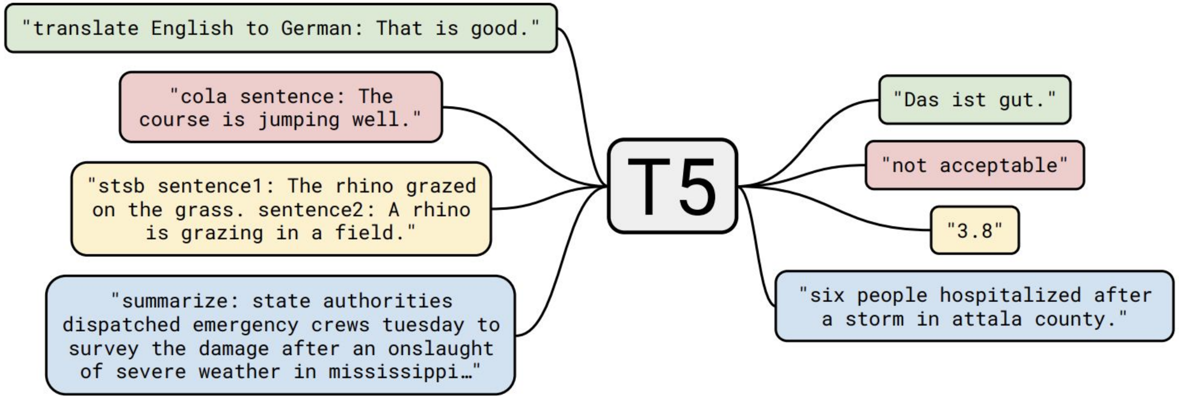
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

T5 (Pretrained Transformer Encoder-Decoder)

Span Corruption

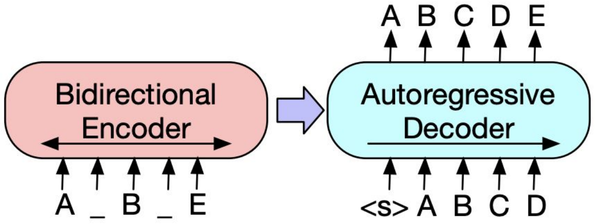
Replace different-length spans from the input with unique placeholders; decode out the spans that were corrupted

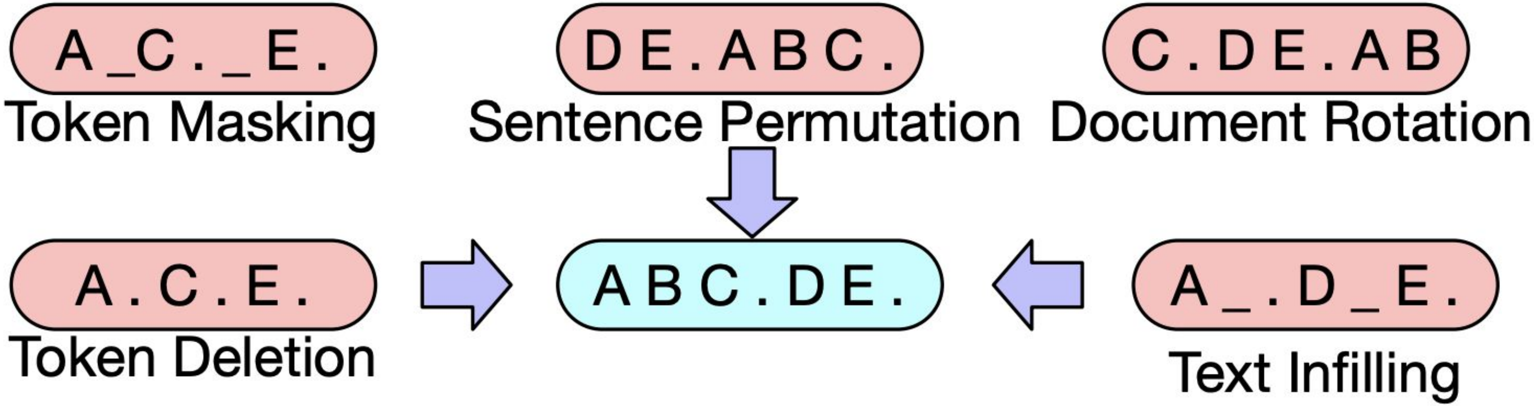




T5: The same model could be used for diverse set of tasks including classification, summarization, translation, etc.

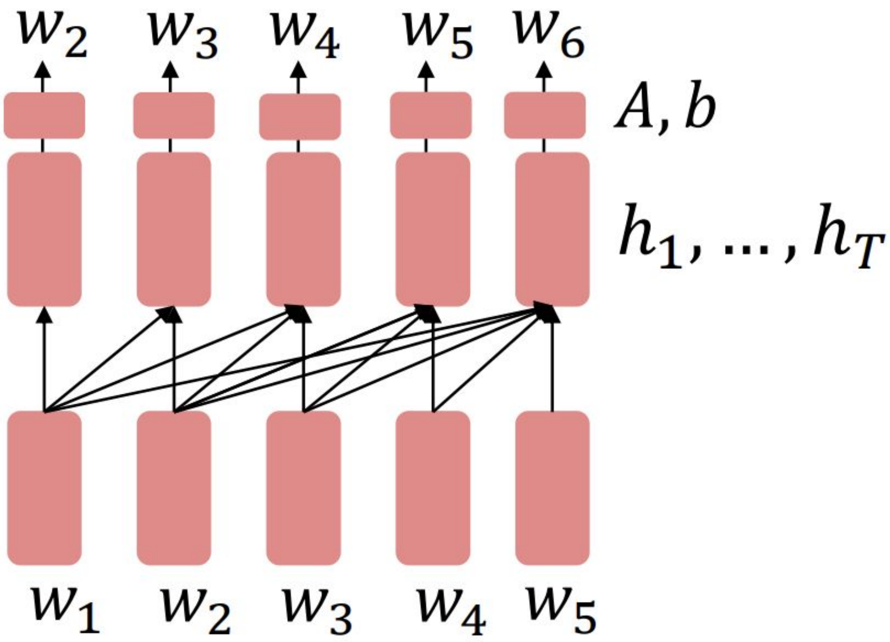
Bi-directional and Auto-regressive Transformers (BART)





Pretraining Transformer decoders (GPT)

It's natural to pretrain decoders as language models



Using GPT for generic tasks (Use of Prompt)

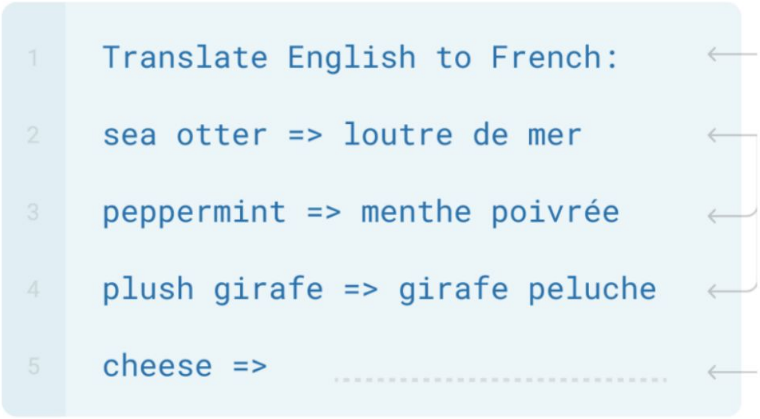
Any NLP task can be expressed in a probabilistic framework as estimating a conditional distribution p(*output*|input).

E.g., reading comprehension

(answer the question, document, question, ____*answer*_____)

Prompting as ‘Few-shot learning’ in GPT-3

Zero/few-shot prompting



Conclusions

- Pretraining on large corpus has been one of the main driving forces in representation learning for NLP
- Lot of interesting issues in adaptation to new domains, new languages, compute time efficiency, few-shot learning, etc.
- Prompting is one latest trend in this direction, where we do not need to fine-tune the pretrained model

References

https://web.stanford.edu/class/cs224n/

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

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