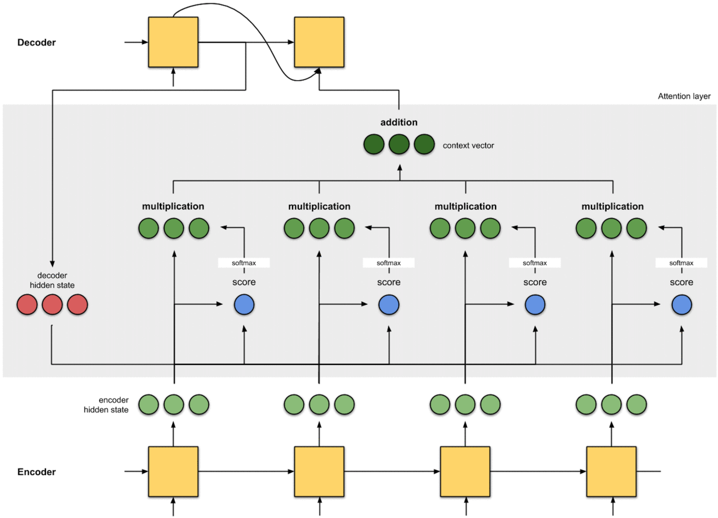
CS391L: Machine Learning WB

Transformers and Attention

Inderjit Dhillon
UT Austin

April 7, 2025

Last time: Attention + RNN in NMT



(Figure from https://towardsdatascience.com/

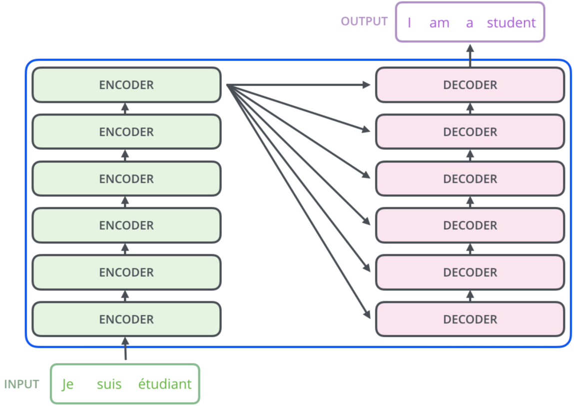
Transformer

Transformer

- An architecture that relies entirely on attention without using CNN/RNN
- A brief history:
  - "Attention Is All You Need" (Vaswani et al., 2017)
    First Transformer for machine translation
  - BERT (Jacob et al., 2018)
    Tranformer + Pretraining achieves SOTA on many other NLP tasks.
  - Vision Transformer (Dosovitskiy et al., 2020)
    Transformer outperforms ResNet on ImageNet

Transformer for machine translation

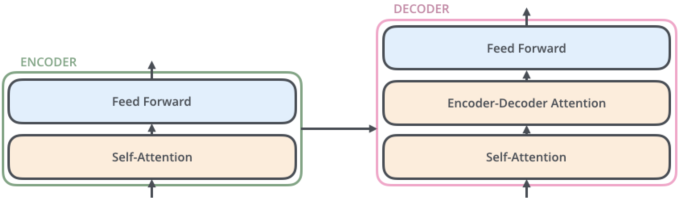
- Pass all input tokens to encoder simultaneously
- Passing through several Transformer blocks



(Vaswani et al., 2017)

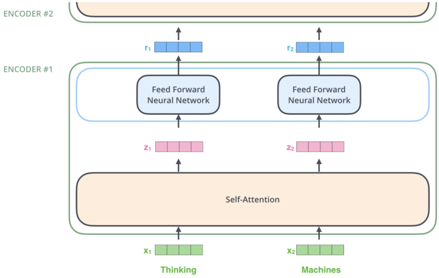
Encoder and Decoder

- Self attention layer: the main architecture used in Transformer
- Decoder: will have another attention layer to help it focus on relevant parts of input sentences.



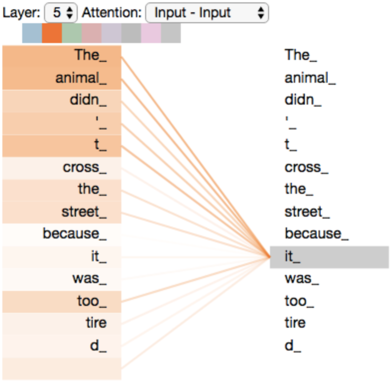
Encoder

- Each word has a corresponding "latent vector" (initially the word embedding for each word)
- Each layer of encoder:
  - Receive a list of vectors as input
  - Passing these vectors to a self-attention layer
  - Then passing them into a feed-foward layer
  - Output a list of vectors



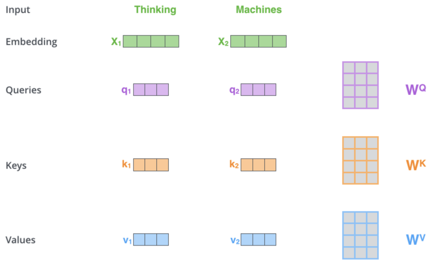
Self-attention layer

- Main idea: The actual meaning of each word may be related to other words in the sentence
- The actual meaning (latent vector) of each word is a weighted (attention) combination of other words (latent vectors) in the sentence



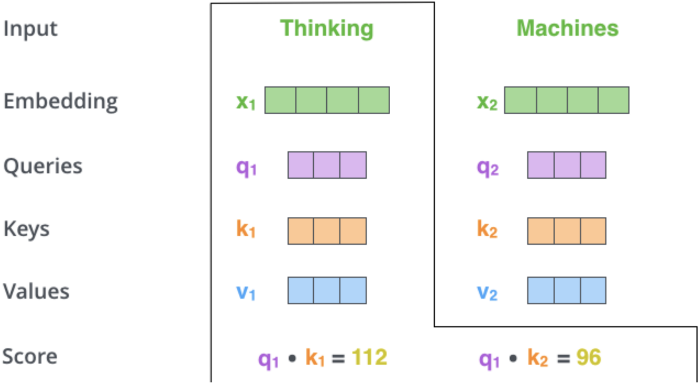
Self-attention layer

- Input latent vectors: $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$
- Self-attention parameters: W^Q, W^K, W^V (weights for query, key, value)
- For each word i, compute
  - Query vector: $\boldsymbol{q}_i = \boldsymbol{x}_i W^Q$
  - Key vector: $\boldsymbol{k}_i = \boldsymbol{x}_i W^K$
  - Value vector: $\boldsymbol{v}_i = \boldsymbol{x}_i W^V$



Self-attention layer

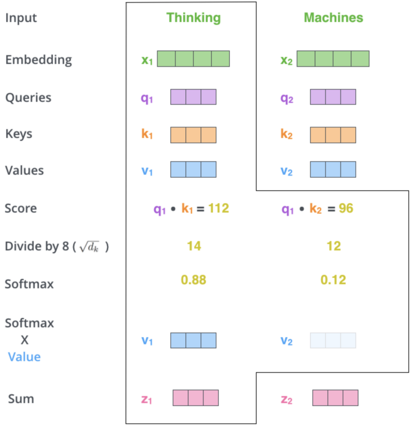
- For each word i, compute the scores to determine how much focus to place on other input words
  - The attention score for word j to word i: $\boldsymbol{q}_i^T \boldsymbol{k}_j$



Self-attention layer

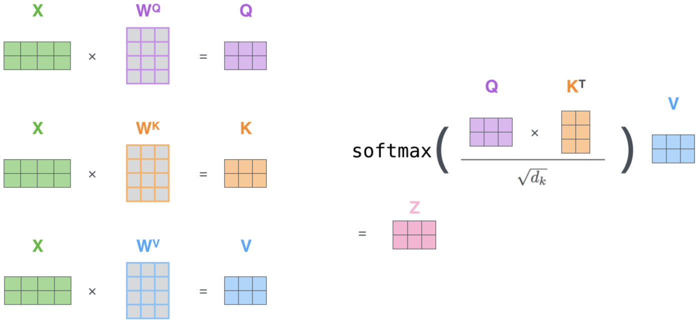
- For each word i, the output vector

$$\sum_j s_{ij} \boldsymbol{v}_j, \quad \boldsymbol{s}_i = \text{softmax}(\boldsymbol{q}_i^T \boldsymbol{k}_1, \ldots, \boldsymbol{q}_i^T \boldsymbol{k}_n)$$



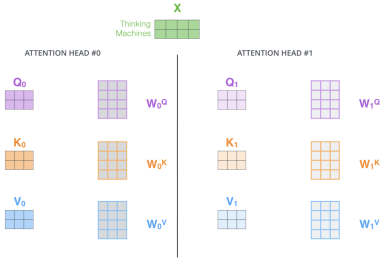
Matrix form

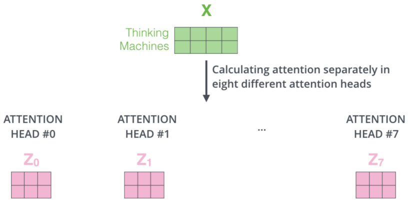
$$Q = XW^Q,\;\; K = XW^K,\;\; V = XW^V,\quad Z = \mathrm{softmax}(QK^T)V$$



Multiple heads

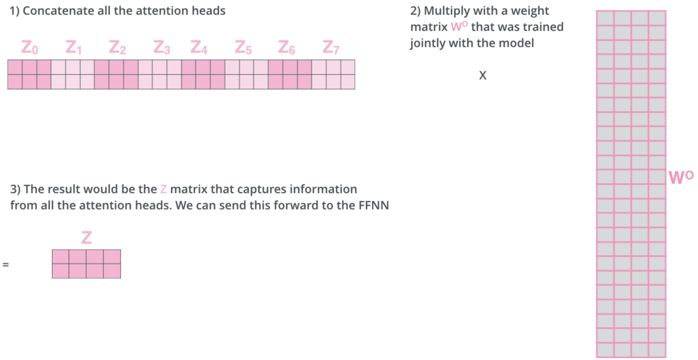
- Multi-headed attention: use multiple set of $(key, value, query)$ weights
- Each head will output a vector Z_i



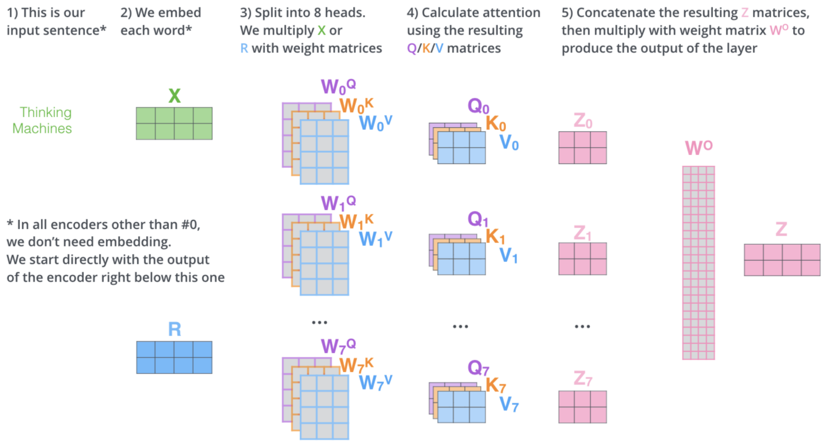


Multiply with weight matrix to reshape

- Gather all the outputs $Z_1, \ldots, Z_k$
- Multiply with a weight matrix to reshape
- Then pass to the next fully connected layer

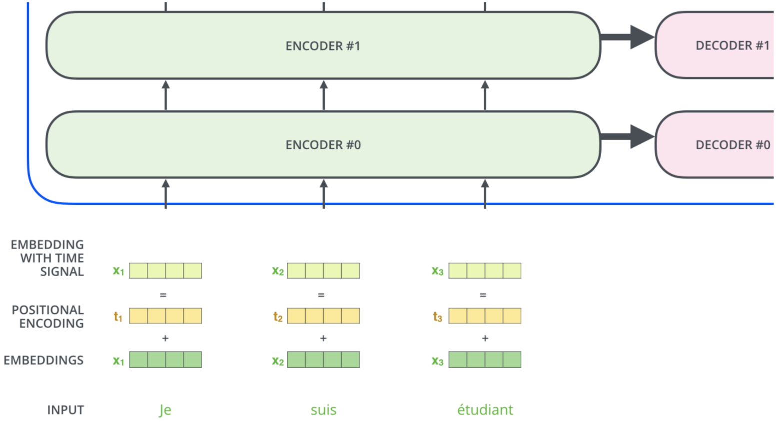


Overall architecture



Position Encoding

- The above architecture ignores the sequential information
- Add a positional encoding vector to each $\boldsymbol{x}_i$ (according to i)



Performance gain on NMT

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

(Figure from Vaswani et al., 2017)

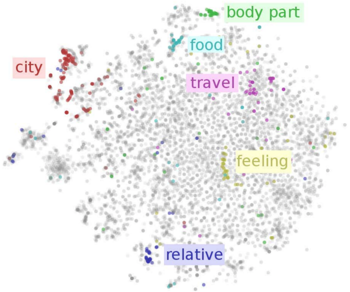
Unsupervised pretraining for NLP

Motivation

- There is a huge amount of unlabeled NLP data available but very little labeled data
- Can we use large amount of unlabeled data to obtain meaningful representations of words/sentences?

Learning word embeddings

- Use large (unlabeled) corpus to learn a useful word representation
  - Learn a vector for each word based on the corpus
  - Hopefully the vector represents some semantic meaning
  - Can be used for many tasks
    - Replace the word embedding matrix for DNN models for classification/translation
- Two different perspectives but led to similar results:
  - Glove (Pennington et al., 2014)
  - Word2vec (Mikolov et al., 2013)



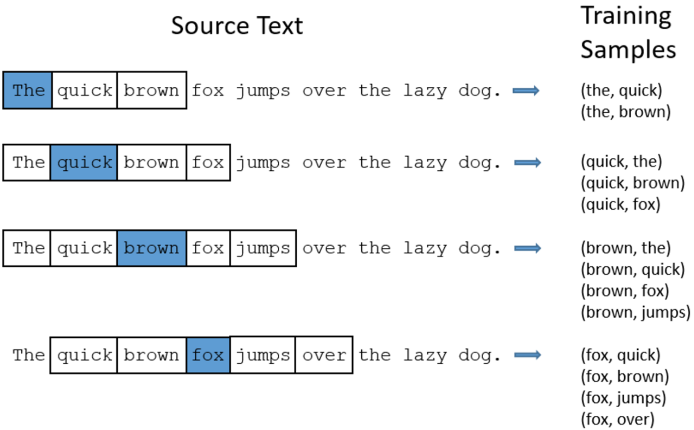
Context information

- Given a large text corpus, how to learn low-dimensional features to represent a word?
- For each word w_i, define the "context" of the word as the words surrounding it in an L-sized window:

$$w_{i-L-2}, w_{i-L-1}, \underbrace{w_{i-L}, \cdots, w_{i-1}}_{\text{context of } w_i}, w_i, \underbrace{w_{i+1}, \cdots, w_{i+L}}_{\text{context of } w_i}, w_{i+L+1}, \cdots$$

- Get a collection of (word, context) pairs, denoted by D.

Examples



(Figure from `http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/`)

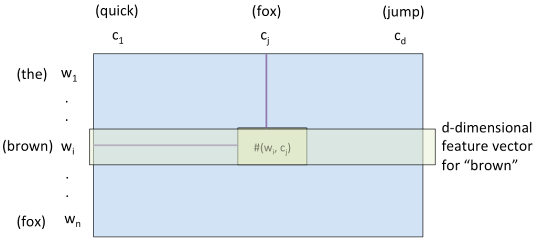
Use bag-of-word model

- Idea 1: Use the bag-of-word model to "describe" each word
- Assume we have context words $c_1, \cdots, c_d$ in the corpus, compute

$$\#(w, c_i) := \text{ number of times the pair } (w, c_i) \text{ appears in } D$$

- For each word w, form a d-dimensional (sparse) vector to describe w

$$\#(w, c_1), \cdots, \#(w, c_d),$$



PMI/PPMI Representation

- Similar to TF-IDF: Need to consider the frequency of each word and each context
- Instead of using co-ocurrence count $\#(w, c)$, we can define pointwise mutual information:

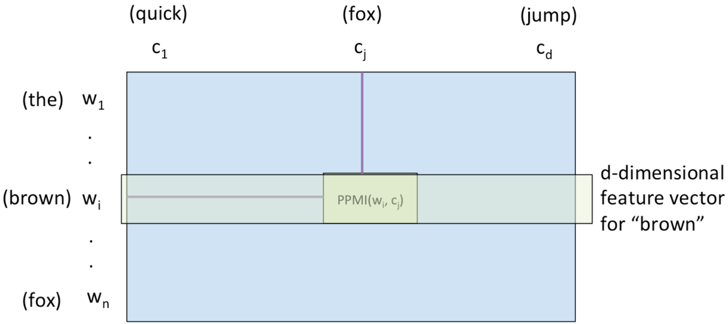
$$\text{PMI}\,(w, c) = \log\left(\frac{\hat{P}(w, c)}{\hat{P}(w)\hat{P}(c)}\right) = \log \frac{\#(w, c)|D|}{\#(w)\#(c)},$$

  - $\#(w) = \sum_c \#(w, c)$: number of times word w occurred in D
  - $\#(c) = \sum_w \#(w, c)$: number of times context c occurred in D
  - $|D|$: number of pairs in D
- Positive PMI (PPMI) usually achieves better performance:

$$\text{PPMI}(w, c) = \max(\text{PMI}(w, c), 0)$$

- M^{PPMI}: a n by d word feature matrix, each row is a word and each column is a context

PPMI Matrix



(quick)
(fox)
(jump)
c_1
c_j
c_d
(the) w_1
(brown) w_i
(fox) w_n
PPMI(w_i, c_j)
d-dimensional feature vector for "brown"

Generalized Low-rank Embedding

- SVD basis will minimize

$$\min_{W,V} \|M^{\text{PPMI}} - WV^T\|_F^2$$

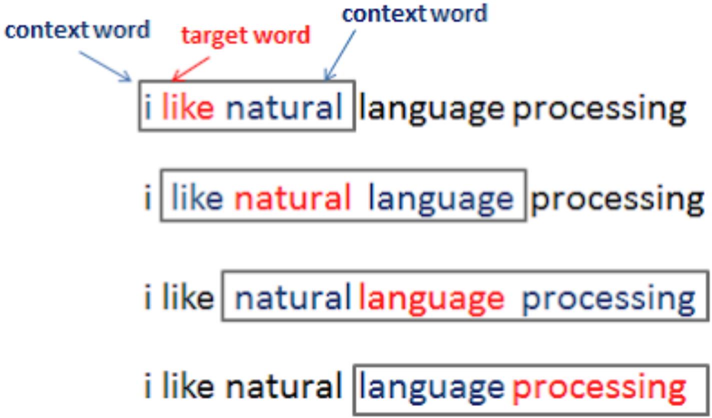
- Glove (Pennington et al., 2014)
  - Negative sampling (less weights to 0s in M^{PPMI})
  - Adding bias term:

$$M^{\text{PPMI}} \approx WV^T + \boldsymbol{b}_w \boldsymbol{e}^T + \boldsymbol{e}\boldsymbol{b}_c^T$$

- Use W as the word embedding matrix

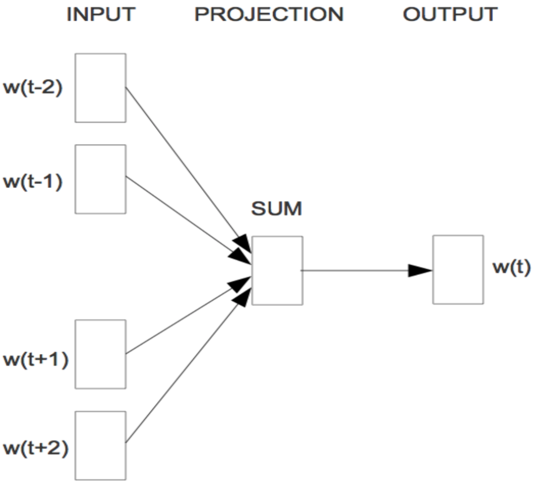
Word2vec (Mikolov et al., 2013)

- A neural network model for learning word embeddings
- Main idea:
  - Predict the target words based on the neighbors (CBOW)
  - Predict neighbors given the target words (Skip-gram)



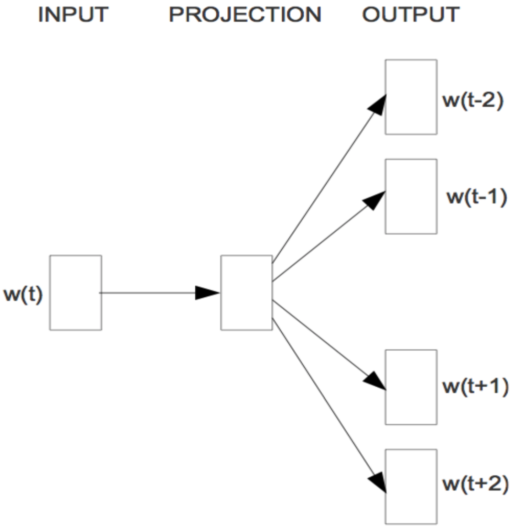
CBOW (Continuous Bag-of-Word model)

- Predict the target words based on the neighbors



Skip-gram

- Predict neighbors using target word



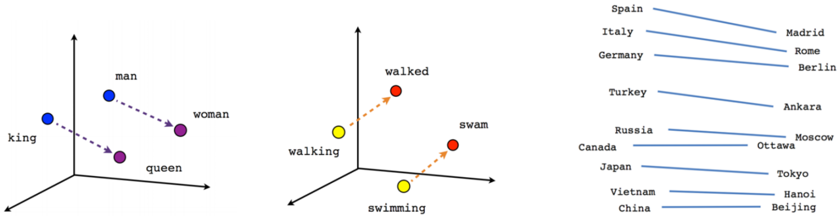
More on skip-gram

- Learn the probability $P(w_{t+j}|w_t)$: the probability to see w_{t+j} in target word w_t's neighborhood
- Every word has two embeddings:
  - v_i serves as the role of target
  - u_i serves as the role of context
- Model probability as softmax:

$$P(o|c) = \frac{e^{u_o^T v_c}}{\sum_{w=1}^{W} e^{u_w^T v_c}}$$

Results

The low-dimensional embeddings are (often) meaningful:



Male-Female Verb tense Country-Capital

(Figure from `https://www.tensorflow.org/tutorials/word2vec`)

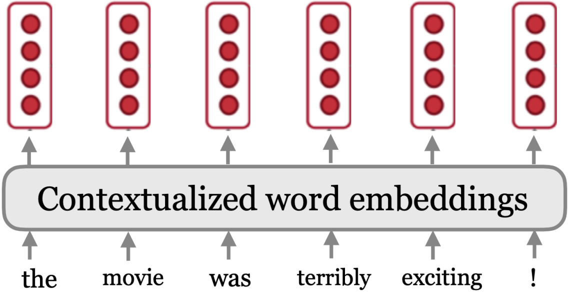
Contextual embedding

Contextual world representation

- The semantic meaning of a word should depend on its context

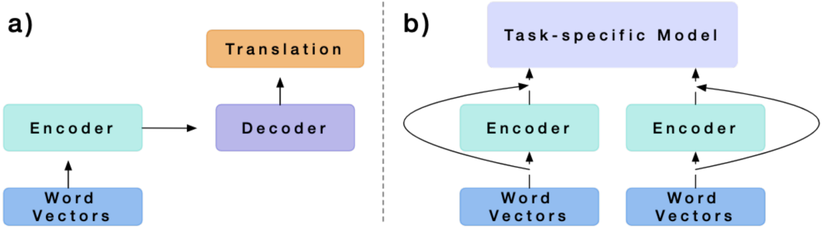
```
[0.9, -0.2, 1.6, …]              [-1.9, -0.4, 0.1, …]
        ↑                                  ↑
open a bank account              on the river bank
```

- Solution: Train a model to extract contextual representations on text corpus



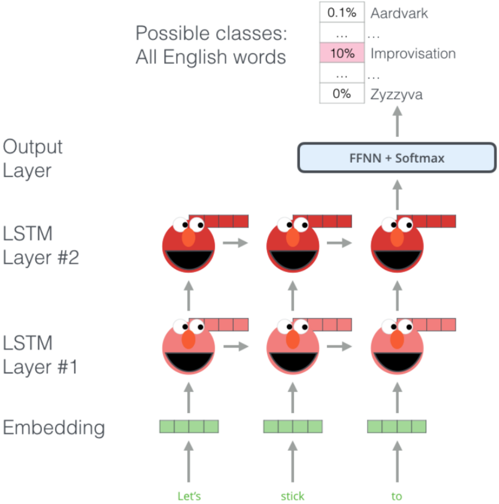
CoVe (McCann et al., 2017)

- Key idea: Train a standard neural machine translation model
- Take the encoder directly as contextualized word embeddings
- Problems:
  - Translation requires paired (labeled) data
  - The embeddings are tailored to particular translation corpuses



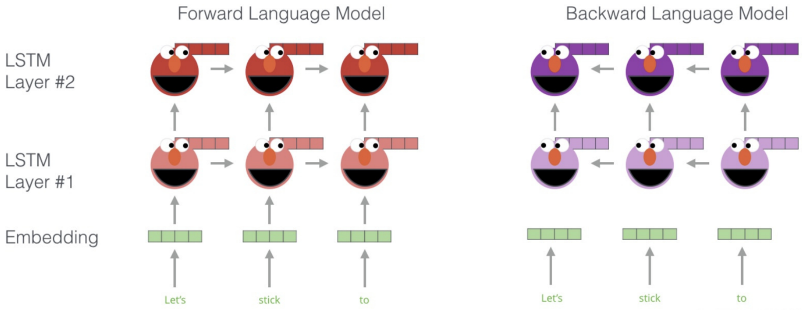
Language model pretraining task

- Predict the next word given the prefix
- Can be defined on any unlabeled document



ELMo (Peter et al., 2018)

- Key ideas:
  - Train a foward and backward LSTM language model on large corpus
  - Use the hidden states for each token to compute a vector representation of each word
  - Replace the word embedding by Elmo's embedding (with fixed Elmo's LSTM weights)



ELMo results

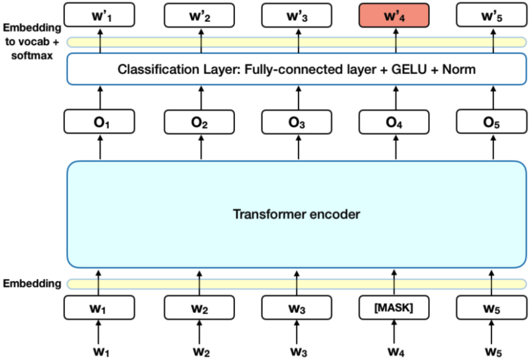
Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

BERT

- Key idea: replace LSTM by Transformer
- Define the generated pretraining task by masked language model
- Two pretraining tasks
- Finetune both BERT weights and task-dependent model weights for each task

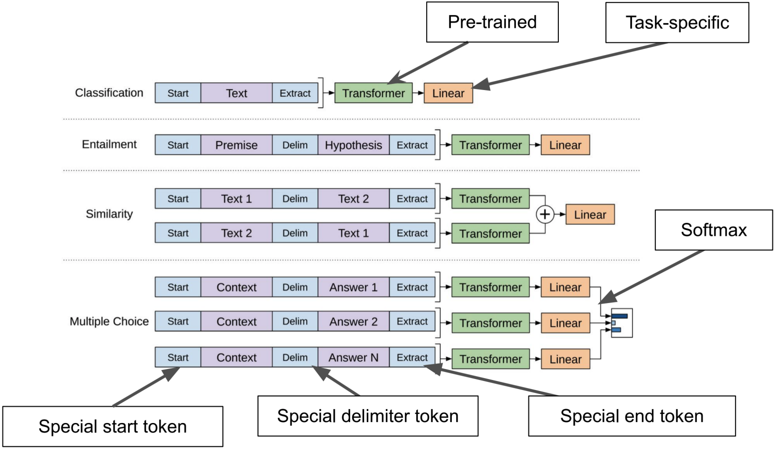
BERT pretraining loss

- Masked language model: predicting each word by the rest of sentence
- Next sentence prediction: the model receives pairs of sentences as input and learns to predict if the second sentence is the subsequent sentence in the original document.



BERT finetuning

- Keep the pretrained Transformers
- Replace or append a layer for the final task
- Train the whole model based on the task-dependent loss



BERT results

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

- BERT base: 110M parameters, BERT large: 340M parameters