[COMSE6998-015] Fall 2024
Introduction to Deep Learning and LLM based Generative AI Systems

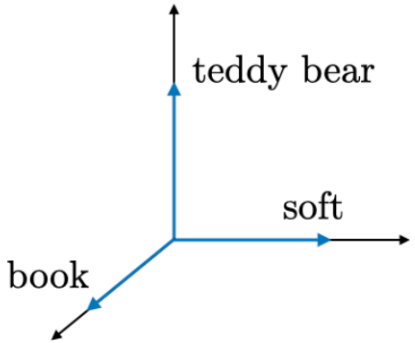
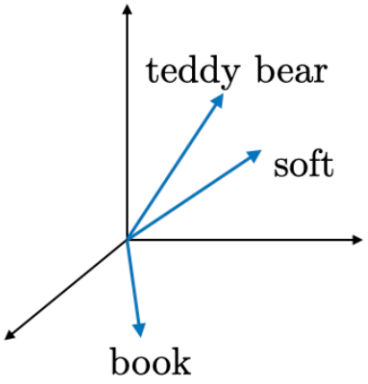
Parijat Dube and Chen Wang

Lecture 5 10/08/24

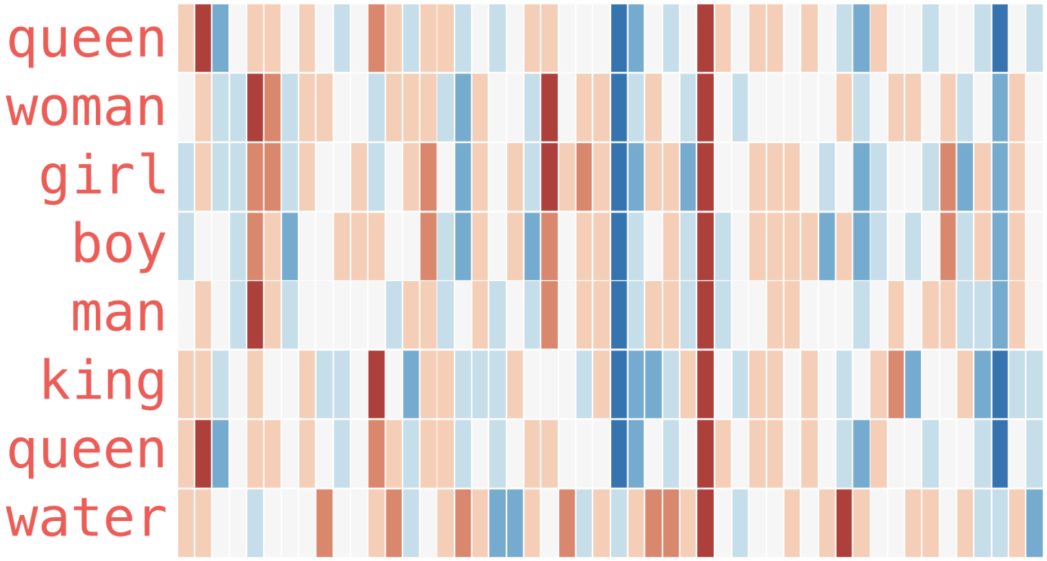
Agenda

- Word embeddings
- Attention mechanism
- Transformer architecture
- Transfer Learning in NLP

Word Embeddings

1-hot representation	Word embedding
	
• Noted o_w • Naive approach, no similarity information	• Noted e_w • Takes into account words similarity

Word2Vec



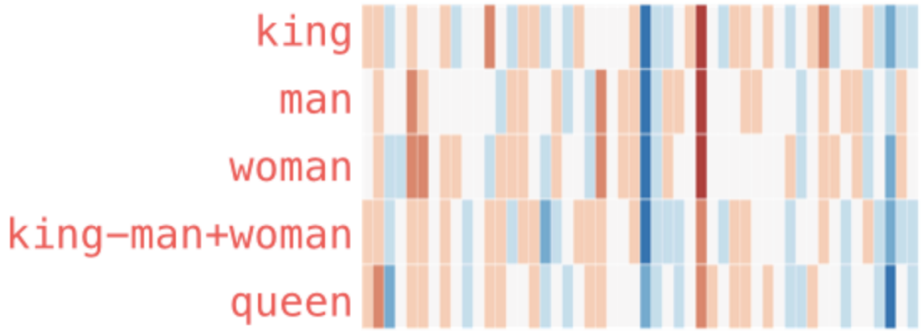
[The Illustrated Word2vec]

Operations using word embeddings

```
model.most_similar(positive=["king","woman"], negative=["man"])
```

```
[('queen', 0.8523603677749634),
 ('throne', 0.7664333581924438),
 ('prince', 0.7592144012451172),
 ('daughter', 0.7473883032798767),
 ('elizabeth', 0.7460219860076904),
 ('princess', 0.7424570322036743),
 ('kingdom', 0.7337411642074585),
 ('monarch', 0.721449077129364),
 ('eldest', 0.7184862494468689),
 ('widow', 0.7099430561065674)]
```

king − man + woman ~= queen

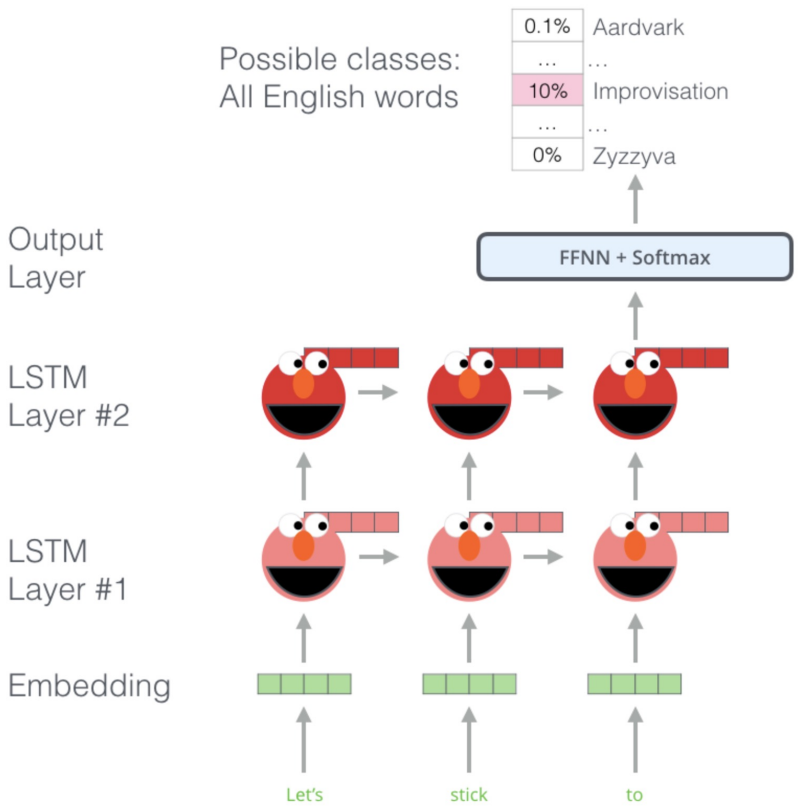


ELMo: Context Matters

- A word can have different meanings depending on the context
  - Give me the stick
  - Let's stick to improving our work
- ELMo generates *contextualized* embeddings for a word
- Contextualized word-embeddings can give words different embeddings based on the meaning they carry in the context of the sentence
- LSTM based architecture

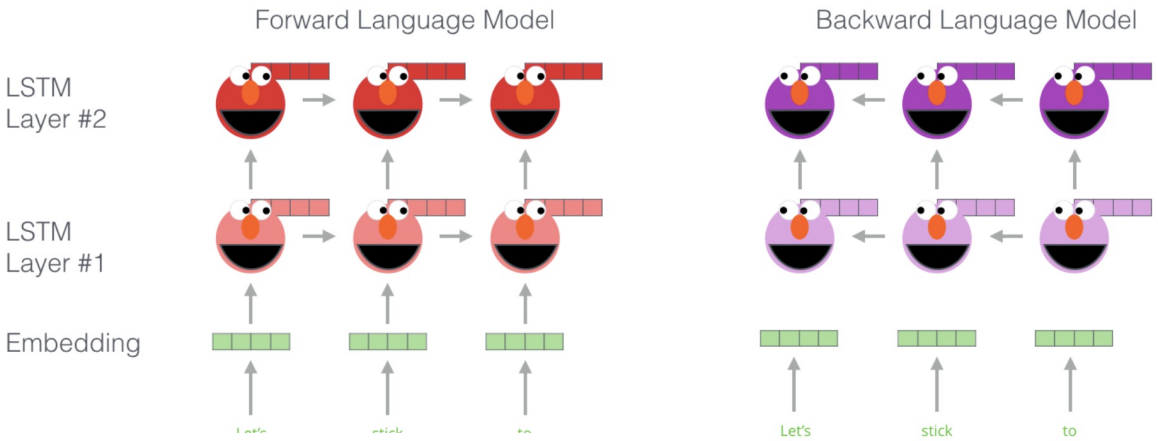
ELMo Training

- Predict next word in a sequence of words
- Self-supervised learning, without any need for labels



Embedding generation in ELMo: Step 1

Embedding of “stick” in “Let’s stick to” - Step #1



Training objective: Jointly maximizes the log likelihood of the forward and backward directions

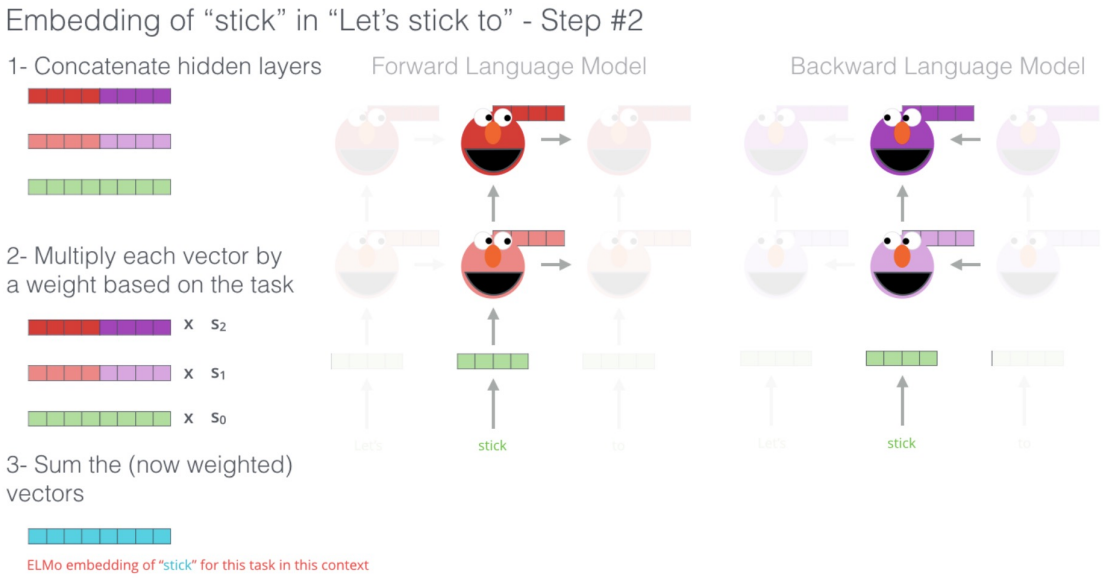
$$\sum_{k=1}^{N} \left(\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \right)$$

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_1, t_2, \ldots, t_{k-1})$$

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_{k+1}, t_{k+2}, \ldots, t_N)$$

Embedding generation in ELMo: Step 2

- Contextualized embedding through grouping together the hidden states (and initial embedding)



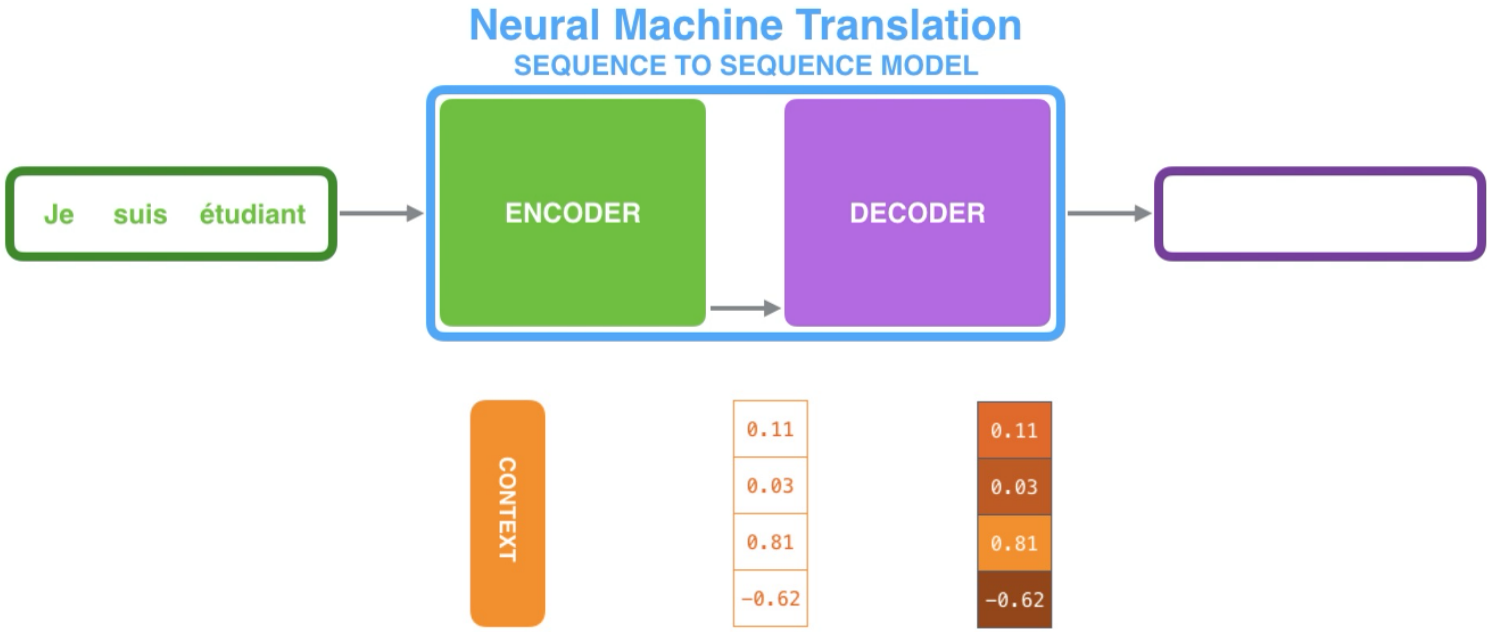
each token t_k, a L-layer biLM computes a set of $2L + 1$ representations

$$\begin{aligned} R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\ &= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\}, \end{aligned}$$

Task specific ELMo embedding

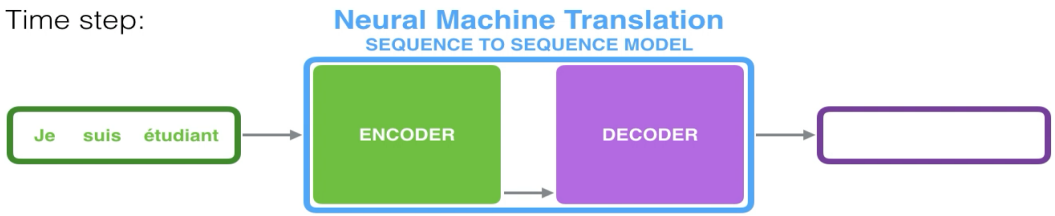
$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}$$

Context in Seq2Seq model



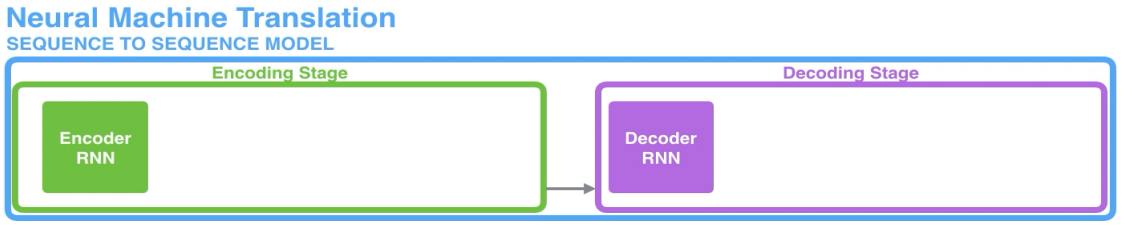
Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)

Hidden states as context



The last hidden state of the encoder forms the context we pass along to the decoder.

The decoder also maintains a hidden state that it passes from one time step to the next.

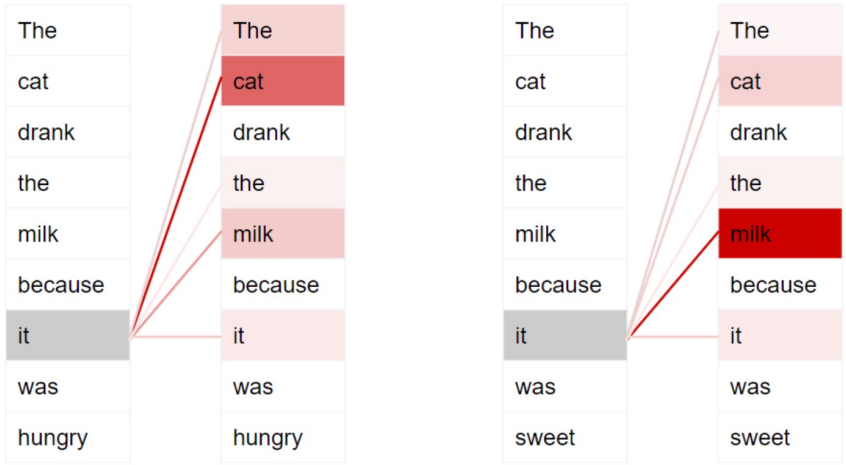


Je suis étudiant

[Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)]

Motivating Attention

- Capture long-range dependence in context
- Example:
  1. The *cat* drank the milk because **it** was hungry.
  2. The cat drank the *milk* because **it** was sweet.



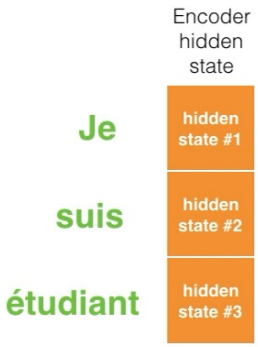
Attention

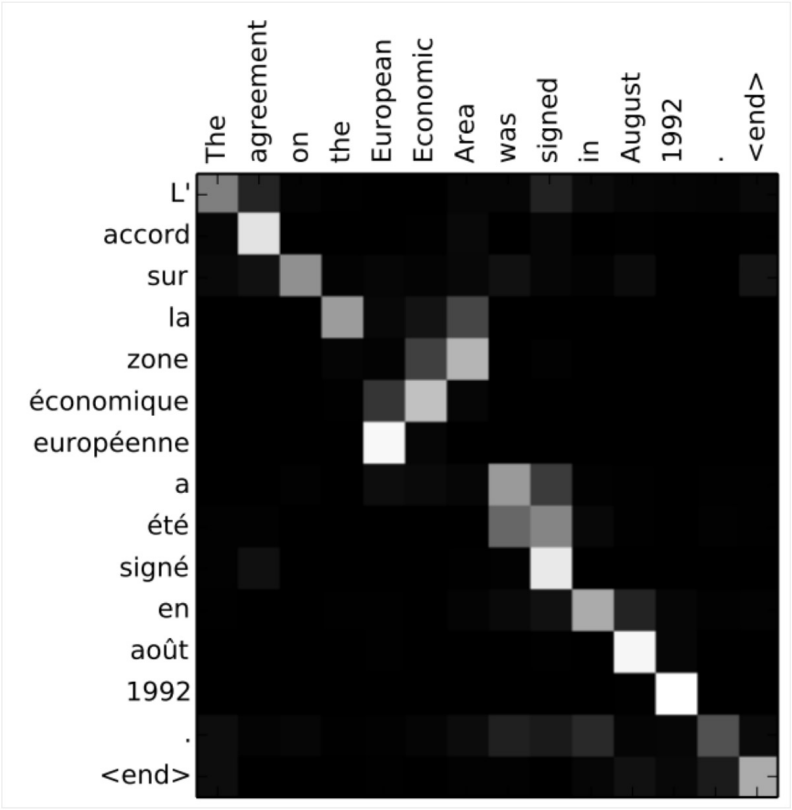
- The context vector turned out to be a bottleneck for Seq2Seq models
- "Attention" mechanism was first proposed in Bahdanau et al., 2014 and Luong et al., 2015
- Attention allows a Seq2Seq model to focus on the relevant parts of the input sequence when decoding
- Attention model differs from a classic Seq2Seq model:
  1. The encoder passes *all* the hidden states to the decoder
  2. The decoder gives different scores to different hidden states from the encoder

Attention Mechanism Steps

Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)

Attention example

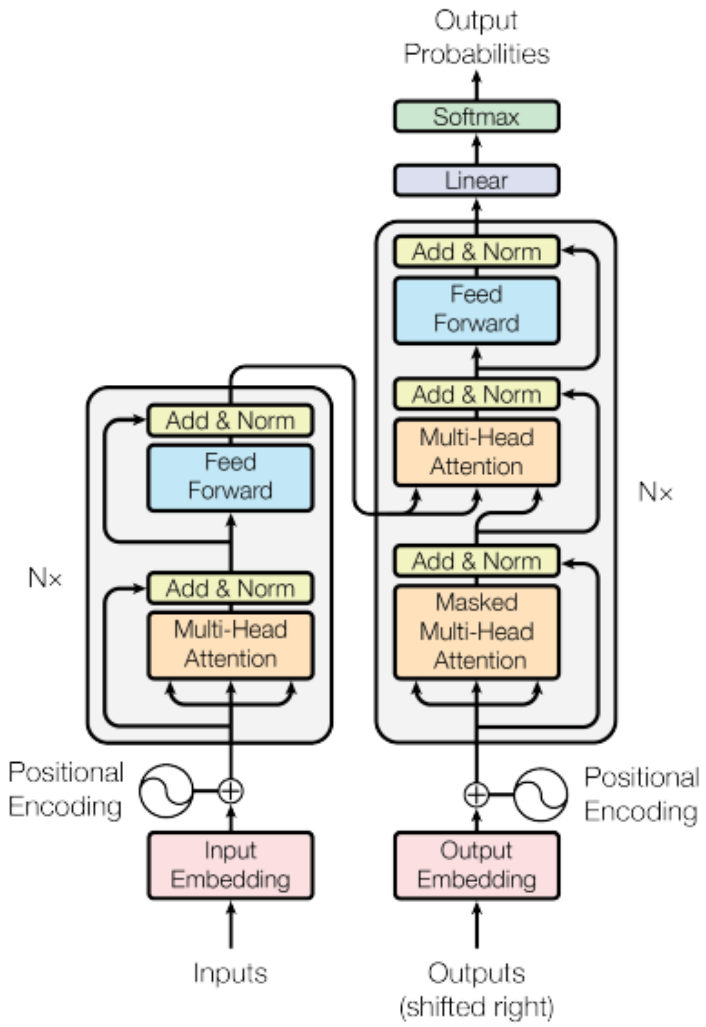




You can see how the model paid attention correctly when outputing "European Economic Area". In French, the order of these words is reversed ("européenne économique zone") as compared to English. Every other word in the sentence is in similar order.

Transformer

- Attention is all you need
- Encoder—Decoder architecture
- The Illustrated Transformer



Multi-Head Attention

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

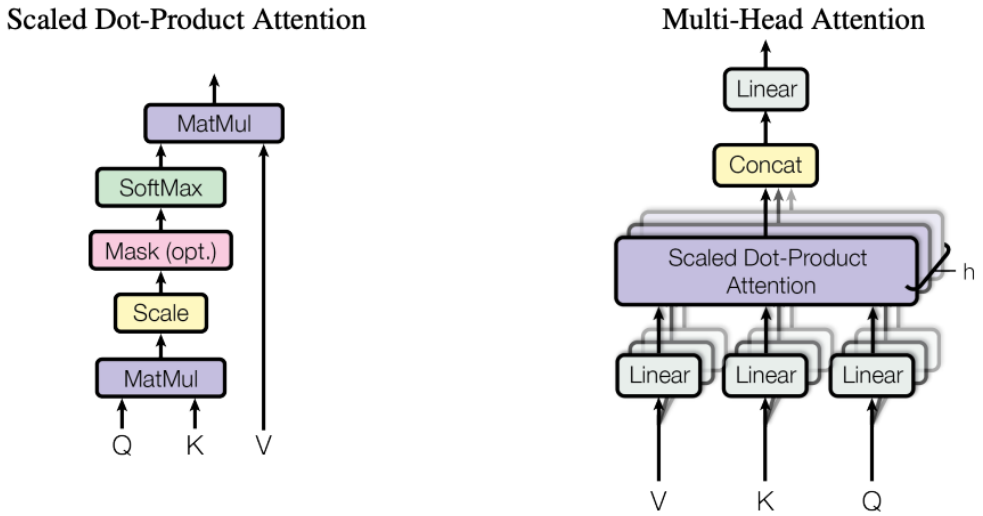
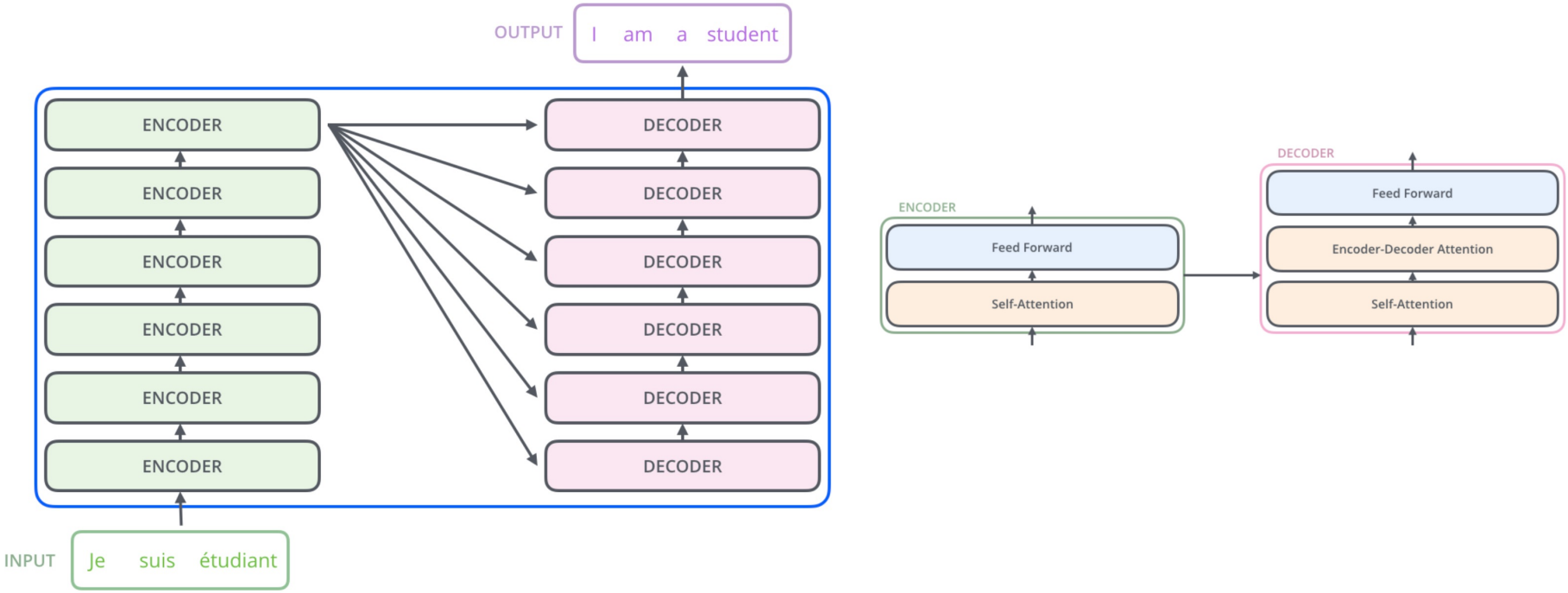


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Attention in Transformer models

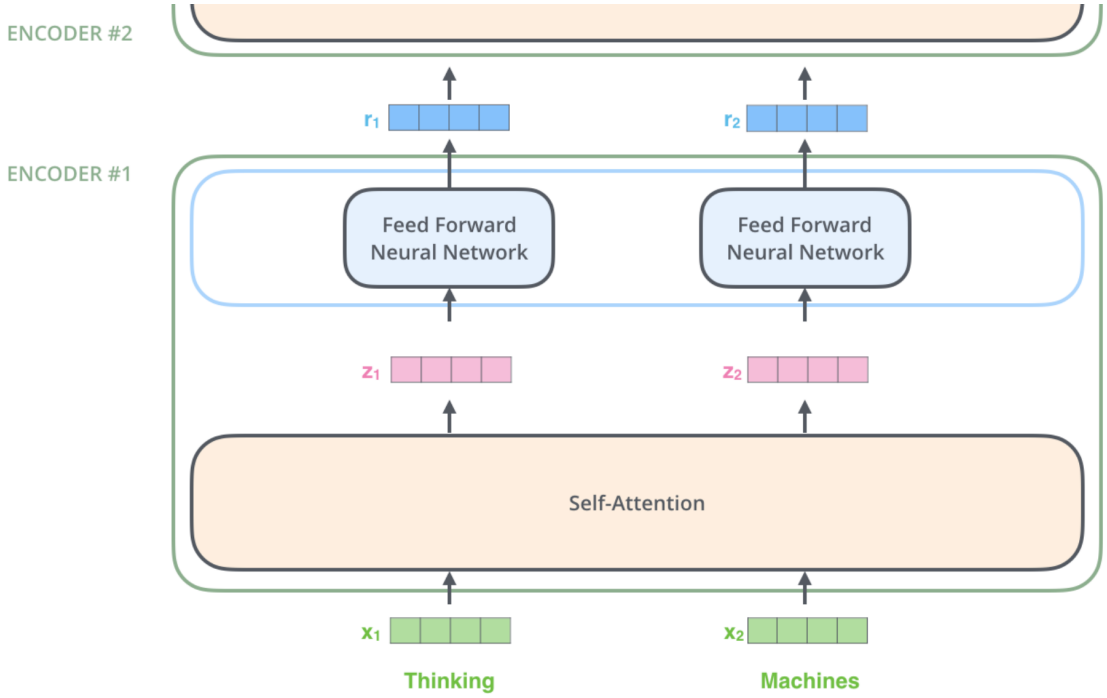
- The Transformer uses multi-head attention in three different ways:
- In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models.
- The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections.

Transformer – a parallelizable encoder-decoder architecture



[The Illustrated Transformer – Part2]

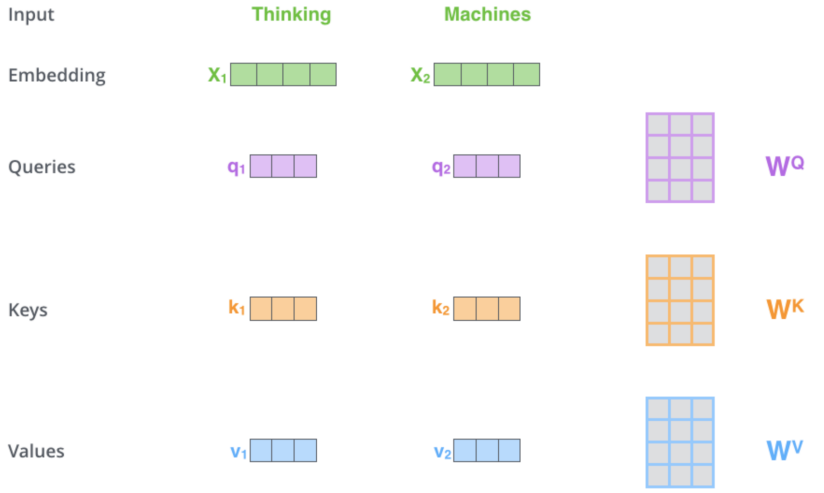
Flow of Tensors in Transformers



The word at each position passes through a self-attention process. Then, they each pass through a feed-forward neural network -- the exact same network with each vector flowing through it separately.

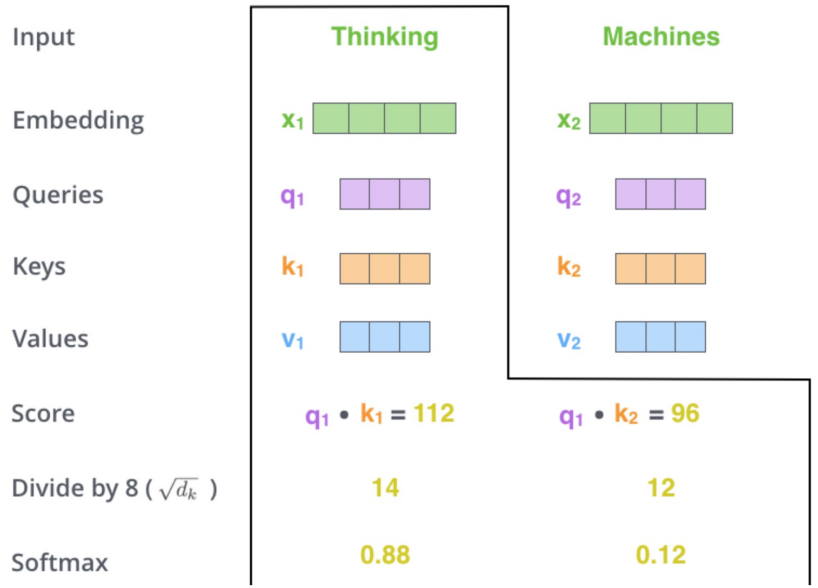
Self-Attention Steps

1. create three vectors (query, key, value) from each of the encoder's input vectors



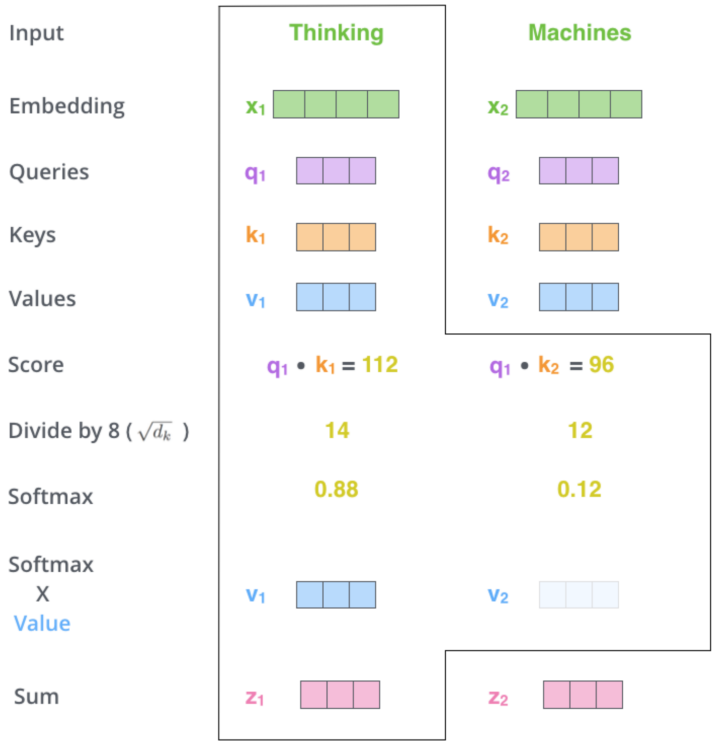
Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

2. create a score for each input vector and pass normalized score through a softmax

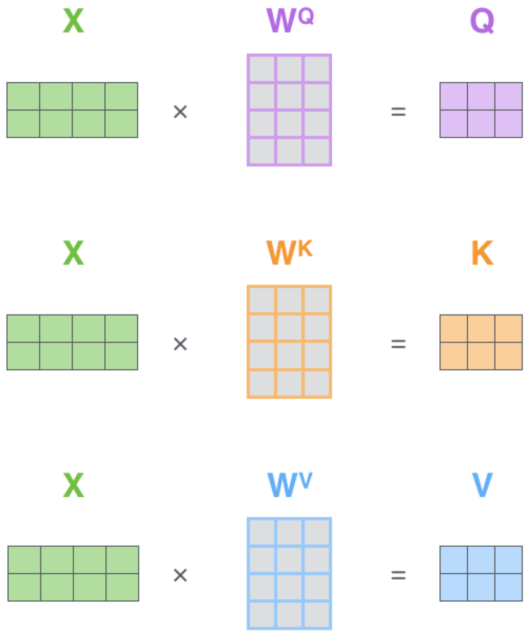


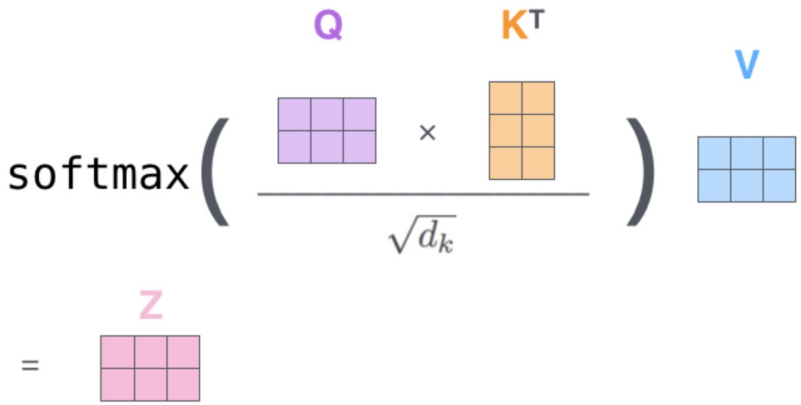
Self-Attention Steps

3. multiply each value vector by the softmax score and sum the weighted value vectors

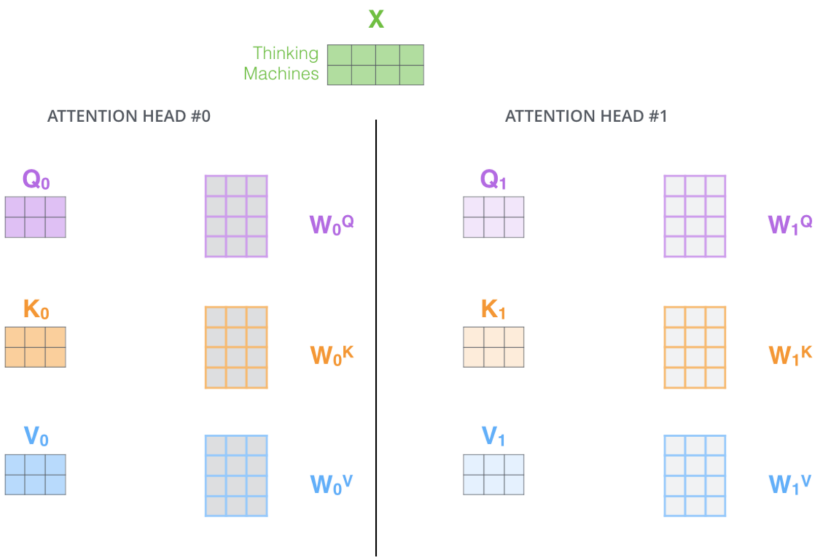


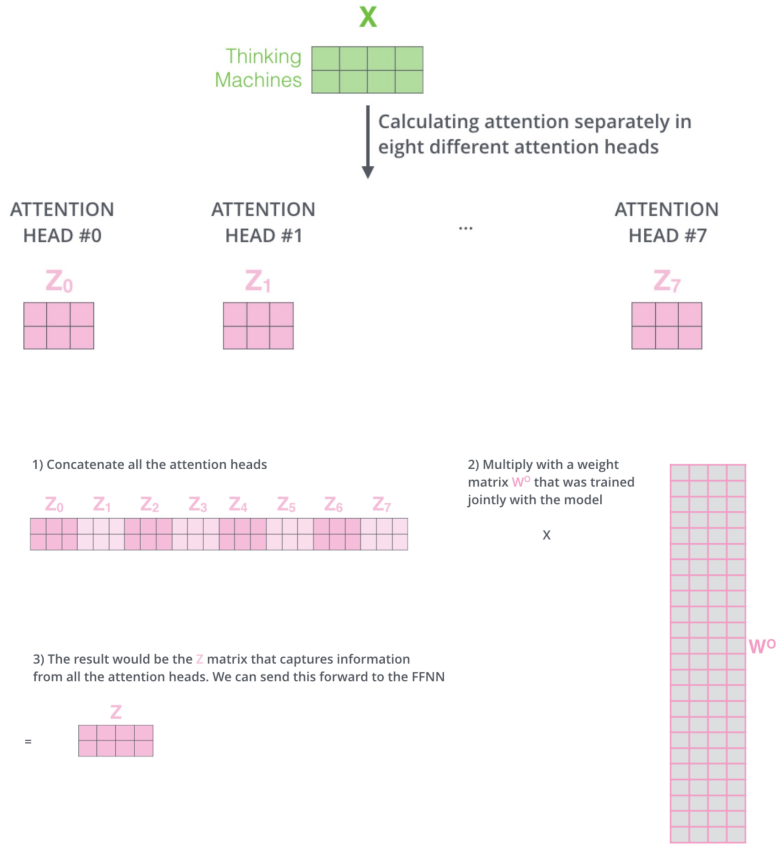
Matrix calculation of self-attention



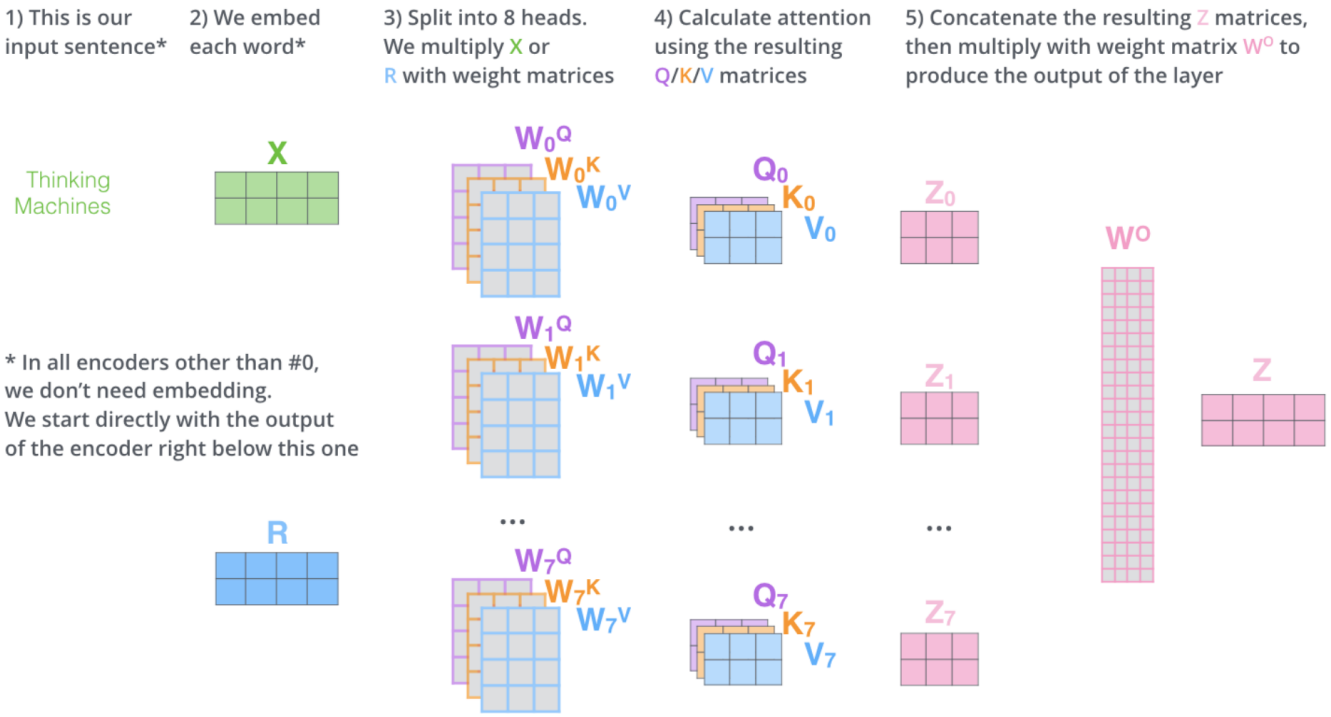


Multi-head Attention

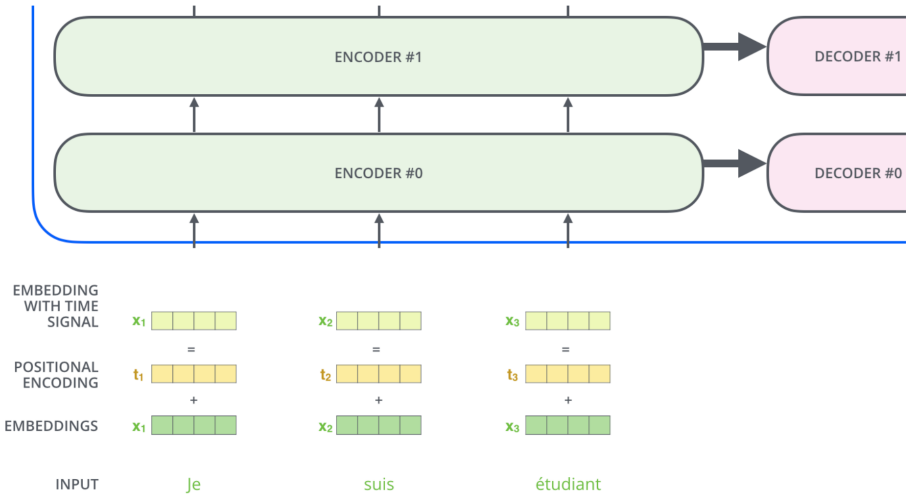


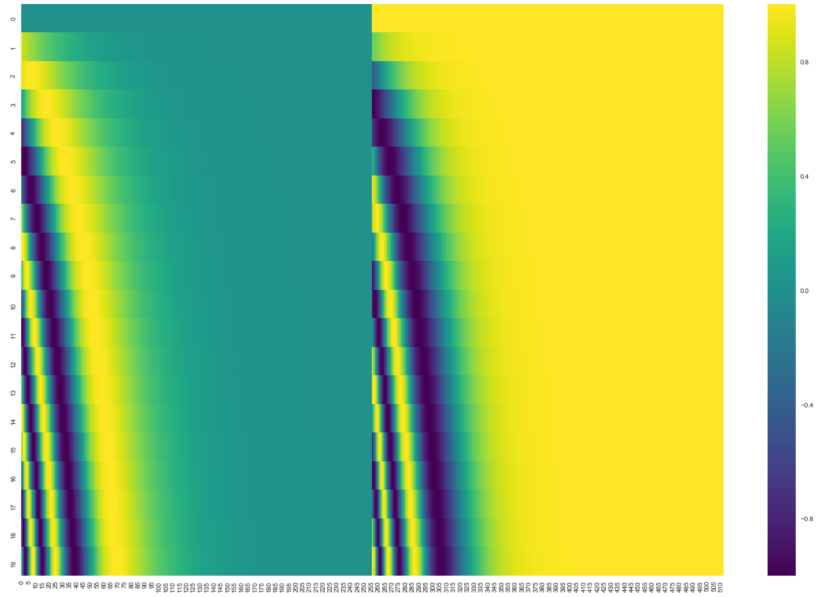


Self attention summary



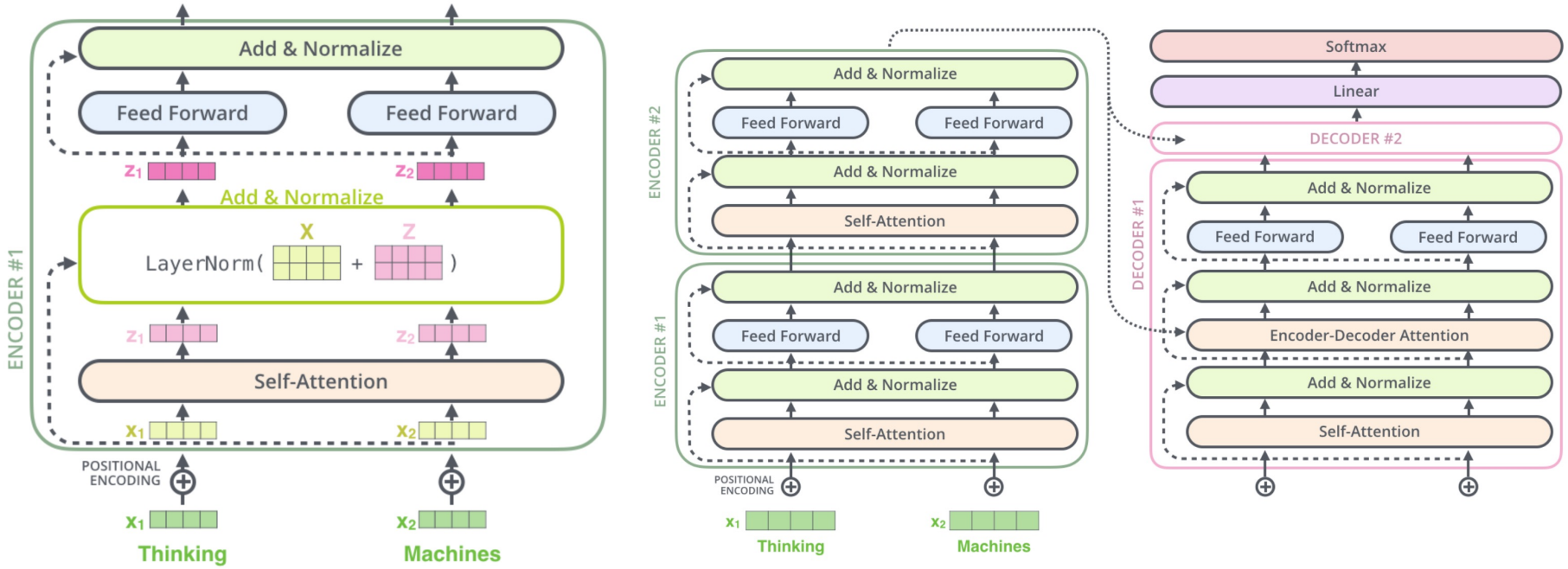
Positional Encoding



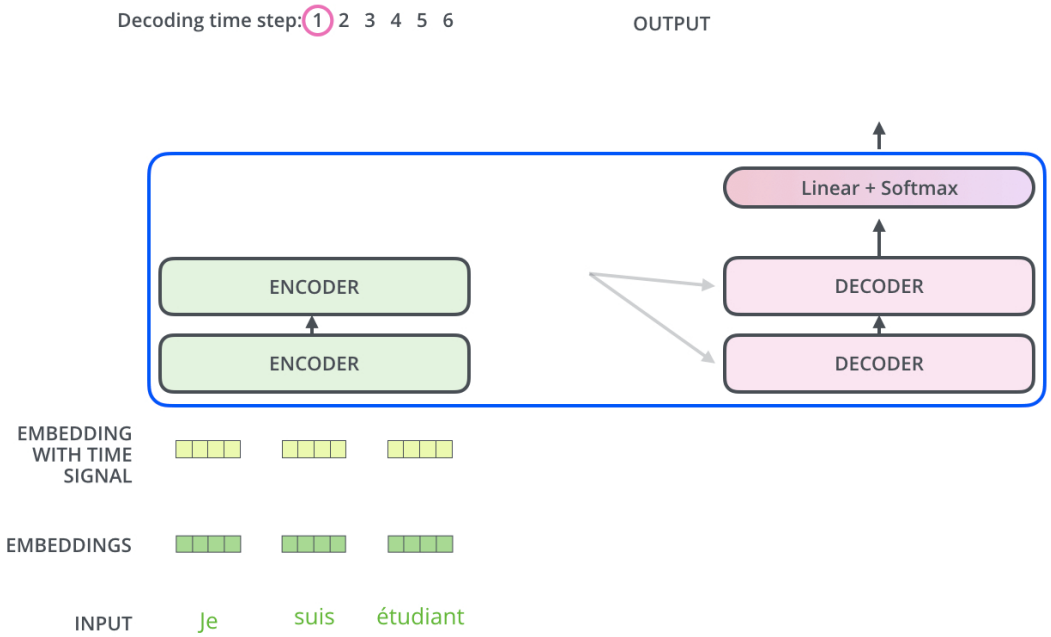


A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns). You can see that it appears split in half down the center. That's because the values of the left half are generated by one function (which uses sine), and the right half is generated by another function (which uses cosine). They're then concatenated to form each of the positional encoding vectors.

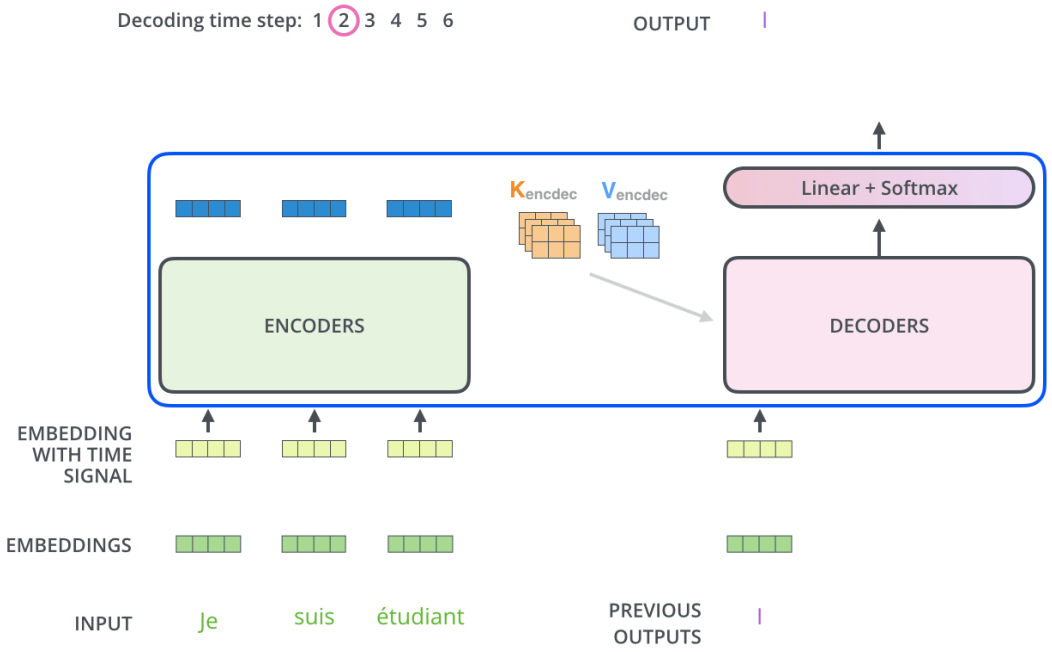
Residual connection



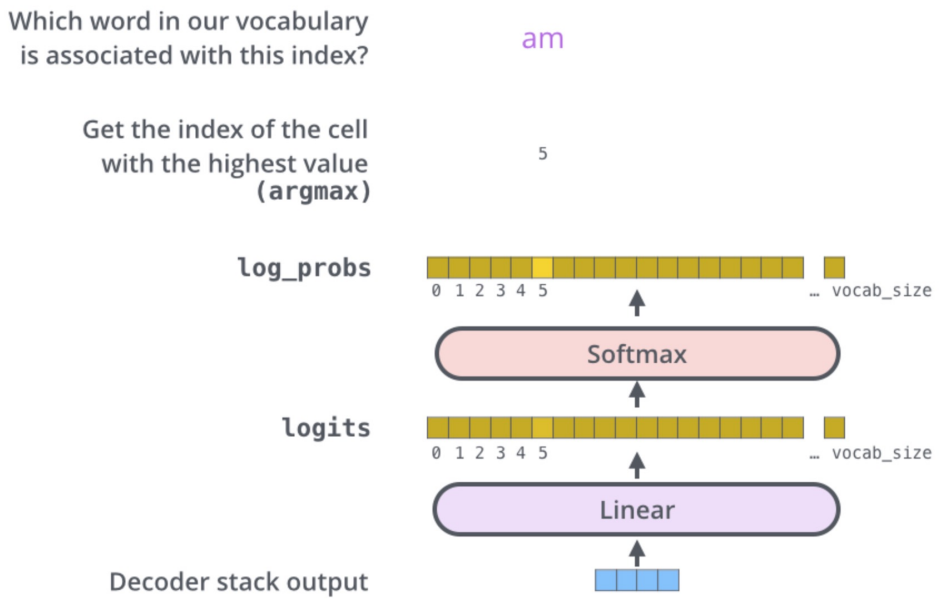
Decoder at first time step



Decoder over time steps



Final linear layer and softmax



This figure starts from the bottom with the vector produced as the output of the decoder stack. It is then turned into an output word.

Transfer Learning in NLP

- Used in standard NLP tasks
  - Question answering
  - Text generation
  - Text summarization
  - Named Entity Recognition
  - Key value pair identification
  - …
- Transfer learning in NLP through
  - Finetuning language models for down stream tasks
  - Generating contextualized word embeddings

Pre-Training Model Architectures in NLP

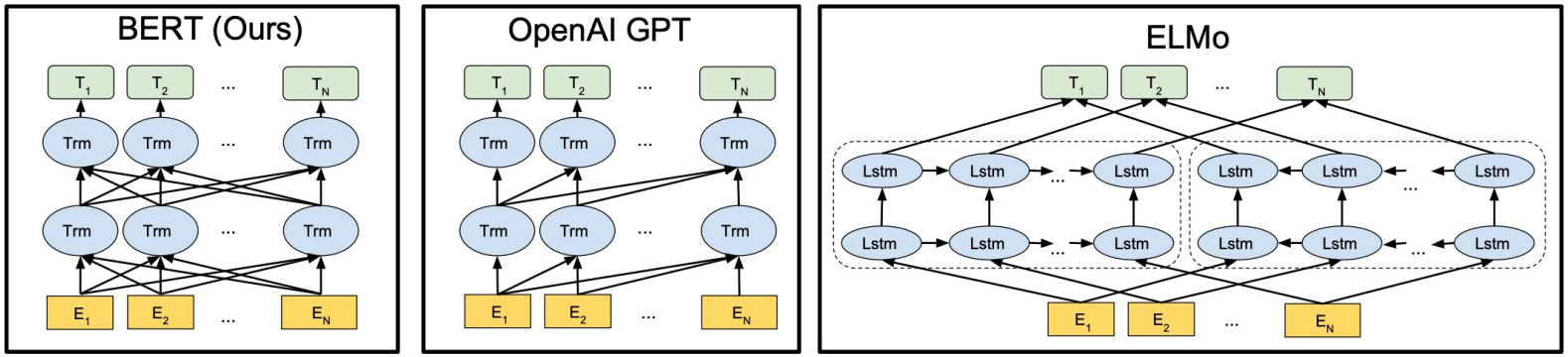


Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

Using Transformer for Transfer Learning in NLP

- Transformers handle long-term dependencies better than LSTMs
- Encoder-Decoder structure of the transformer made it perfect for machine translation
- How to pre-train a language model using transformer-based architecture (with attention mechanisms and faster training due to parallelization) that can be finetuned for downstream supervised learning tasks?

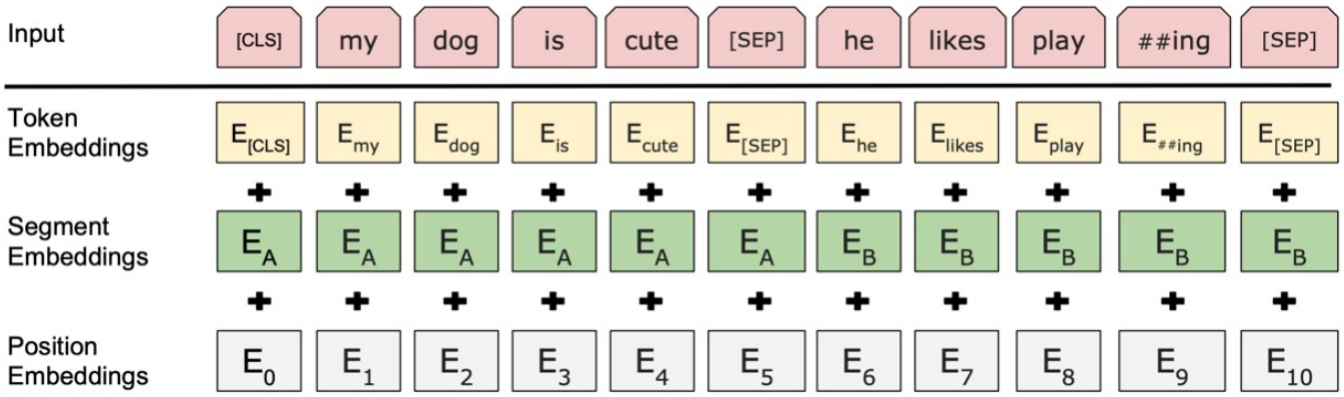
BERT (Bidirectional Encoder Representations from Transformers)

- OpenAI Transformer only trains a forward language model while ELMo's model was bidirectional
- BERT is a transformer-based model whose language model is conditioned on both left and right contexts
- BERT is basically a trained Transformer Encoder stack

BERT Features

- Training corpus was comprised of two entries
  - Toronto Book Corpus (800M words), and
  - English Wikipedia (2,500M words)
- Two versions of BERT (L stands for the number of layers, H stands for the hidden size, A stands for the number of self-attention heads)
  - BERT-Base: L = 12, H = 768, A = 12, Total parameters = 110M
  - BERT-Large: L = 24, H = 1024, A = 16, Total parameters = 340M

Input Representation of BERT

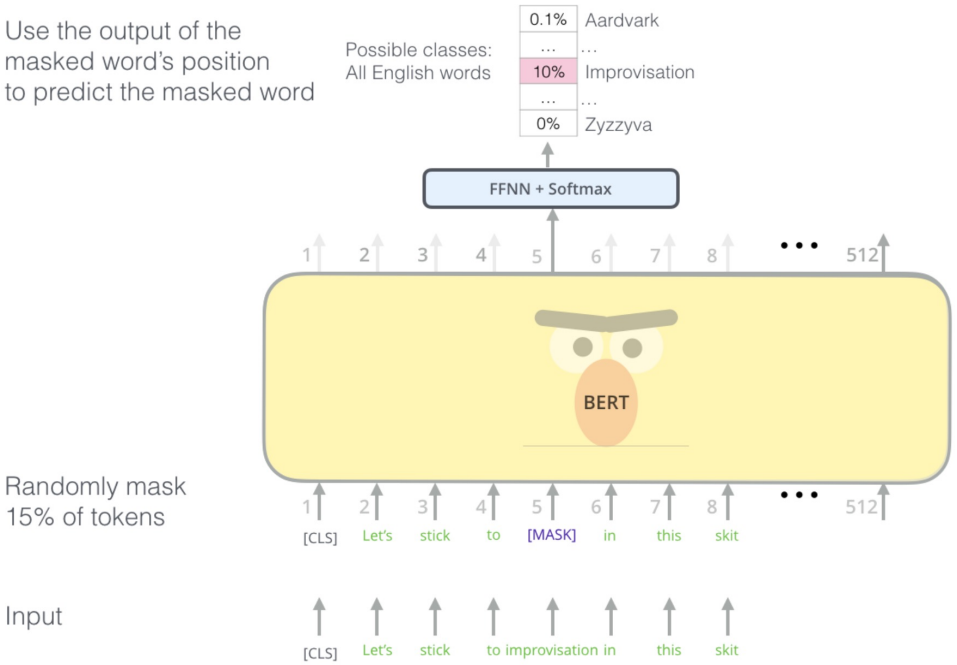


BERT Training – Predicting word at a position

- Masked language model (Masked LM)

That's `[mask]` she `[mask]` -> That's what she said

- Randomly masks 15% of tokens in the input and model is trained to predict the masked word



BERT's clever language modeling task masks 15% of words in the input and asks the model to predict the missing word.

BERT Training – Two sentence tasks

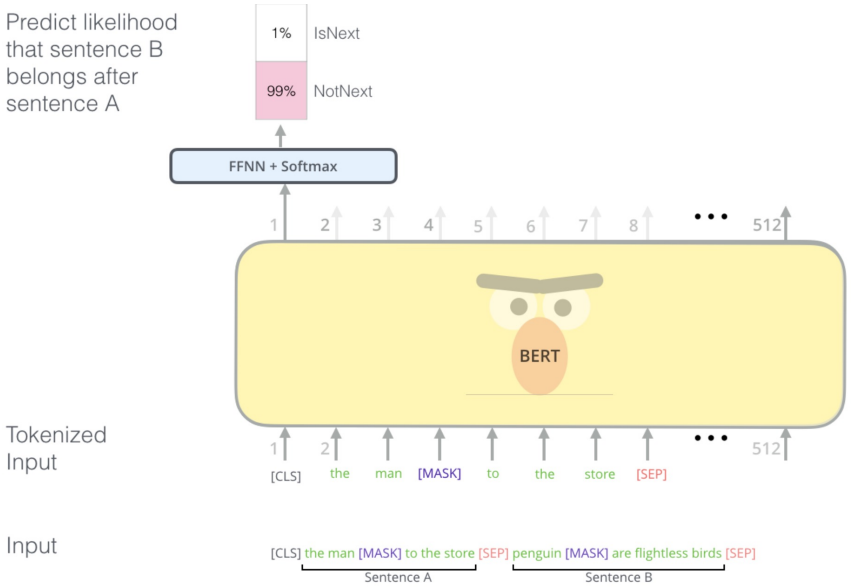
- Next sentence prediction (NSP)
- Task: Given two sentences (A and B), is B likely to be the sentence that follows A, or not? (binary classification)

Input = `[CLS]` That's `[mask]` she `[mask]` . [SEP] Hahaha, nice! [SEP]

Label = *IsNext*

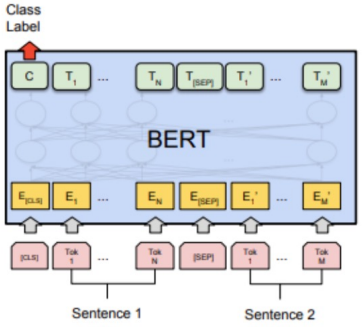
Input = `[CLS]` That's `[mask]` she `[mask]` . [SEP] Dwight, you ignorant `[mask]` ! [SEP]

Label = *NotNext*

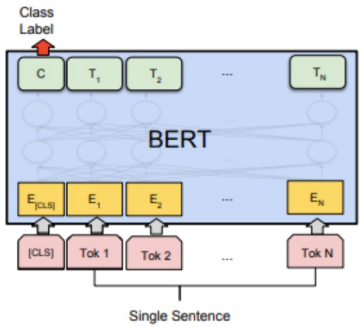


The second task BERT is pre-trained on is a two-sentence classification task. The tokenization is oversimplified in this graphic as BERT actually uses WordPieces as tokens rather than words --- so some words are broken down into smaller chunks.

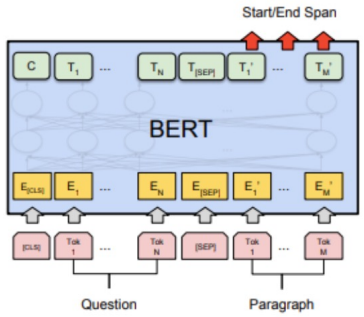
Using BERT for different Downstream Tasks



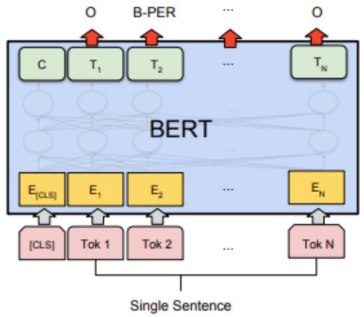
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



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



(c) Question Answering Tasks: SQuAD v1.1



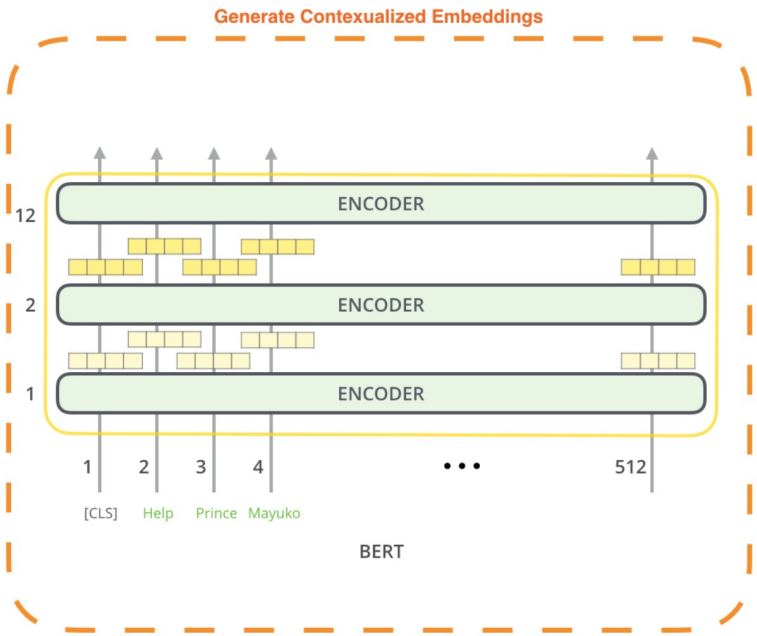
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

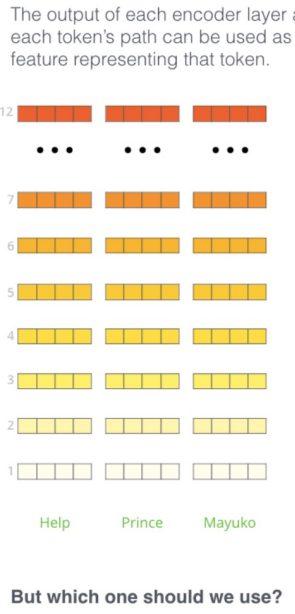
Fine-tuning BERT for Sentiment Analysis

- Sentiment Analysis with BERT colab tutorial

BERT for Feature Extraction

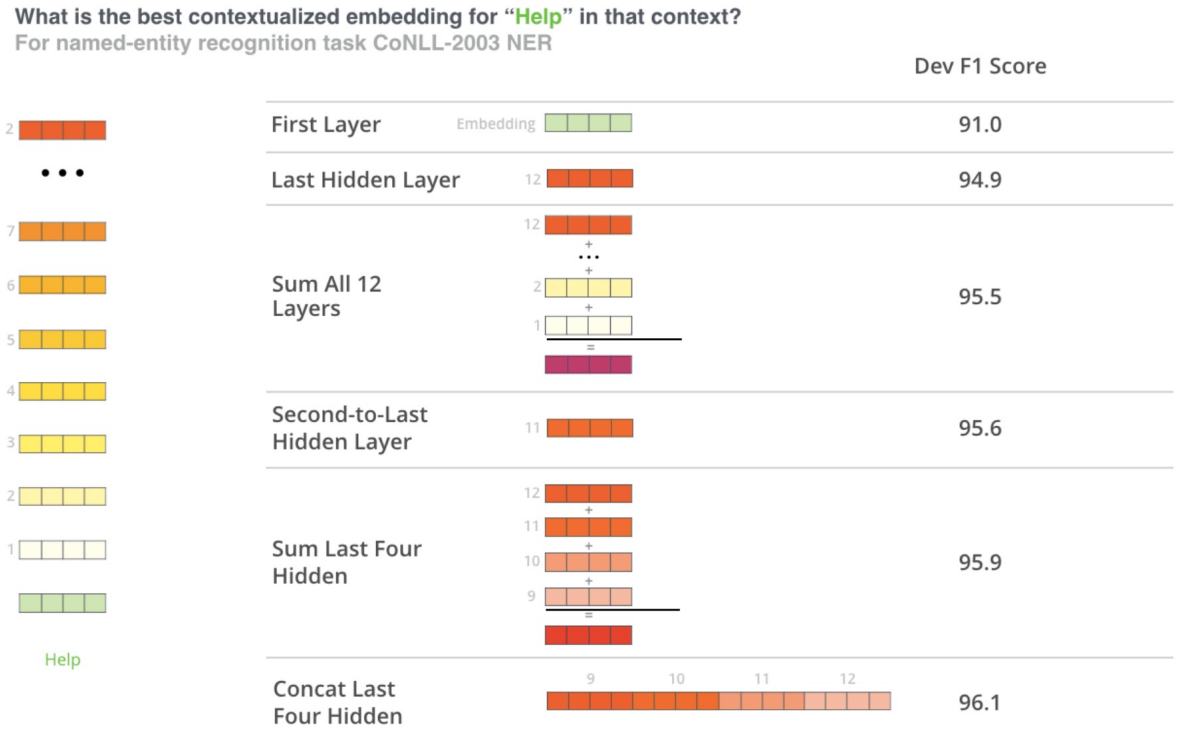
- Using BERT to create contextualized word embeddings





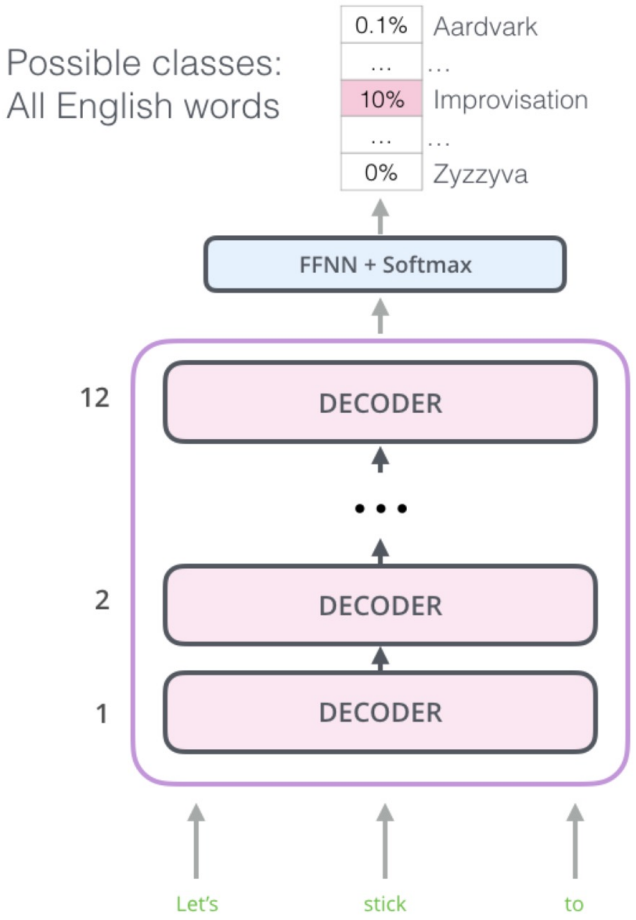
- 12 embedding vectors for each word from the output of 12 encoder layers
- Which embedding(s) to use for a given task?

Different ways to create contextualized embeddings for a word using BERT



OpenAI Transformer

- Pre-training a transformer decoder for language modeling
- Decoder masks future tokens so can be trained on next word prediction task
- Model has 12 decoder layers stacked
- Only masked self attention layers



Transfer Learning to Downstream Tasks