Attention

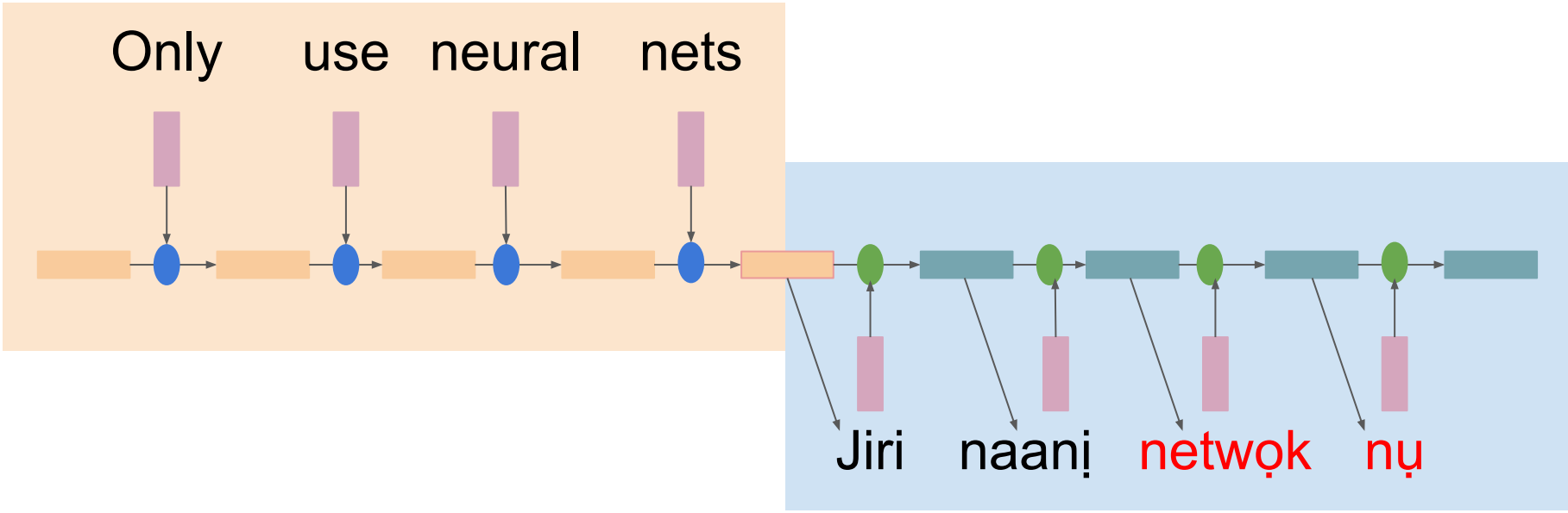
References:

- http://phontron.com/class/nn4nlp2017/assets/slides/nn4nlp-09-attention.pdf
- https://nlp.stanford.edu/~johnhew/public/14-seq2seq.pdf
- http://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/slides/lec16.pdf
- Neural Machine Translation by Jointly Learning to Align and Translate (Bahdanau et. al, 2015)
- Attention Is All You Need (Vaswani *et. al*, 2017)
- Hierarchical Attention Networks for Document Classification

Why Attention is Needed?

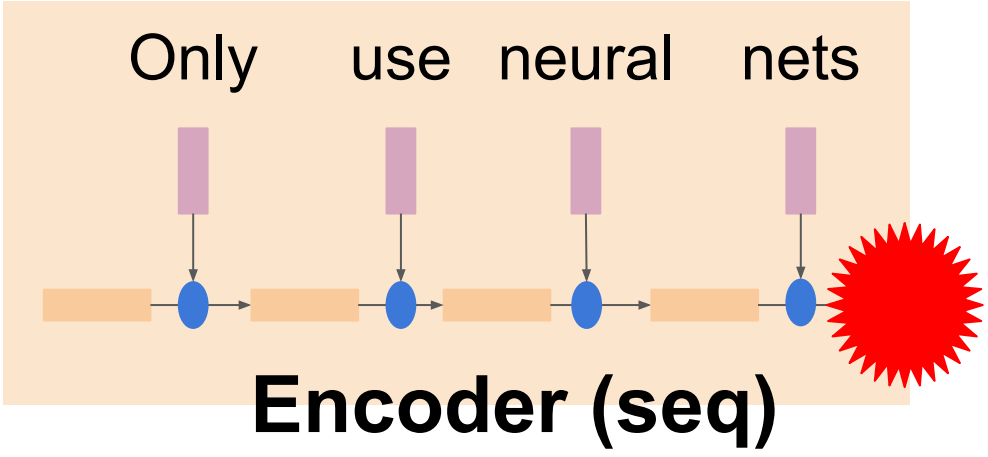
The information bottleneck and latent structure

Given the diagram below, what problem do you foresee when translating progressively longer sentences?



The information bottleneck and latent structure

We are trying to encode *variable-length* structure (e.g., variable-length sentences) in a fixed-length memory (e.g., only the 300 dimensions of your hidden state.)



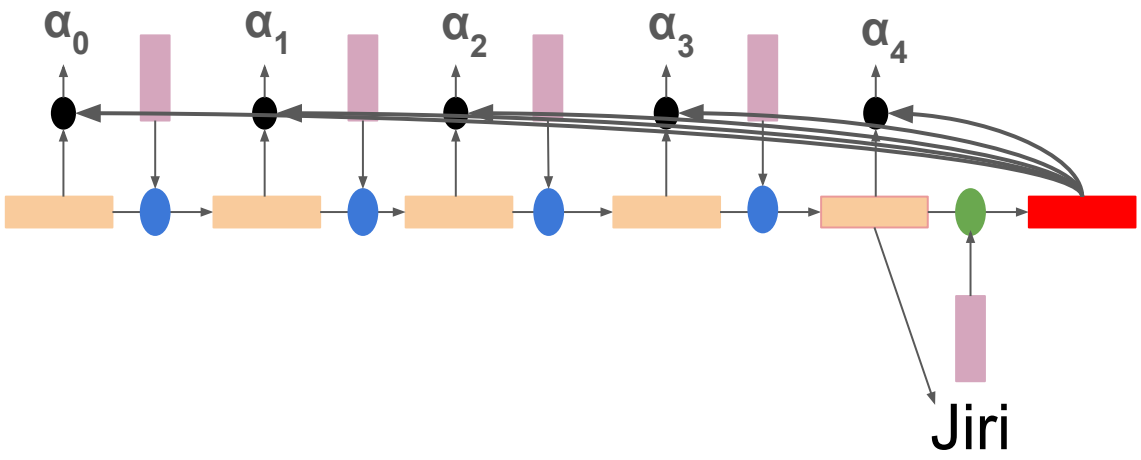
The last encoder hidden state is the bottleneck -- all information in the source sentence must pass through it to get to the decoder.

Finding a solution to this problem was the final advance that made neural MT competitive with previous approaches.

Learning to pay attention

Attention **summarizes the encoder**, focusing on specific parts/words.

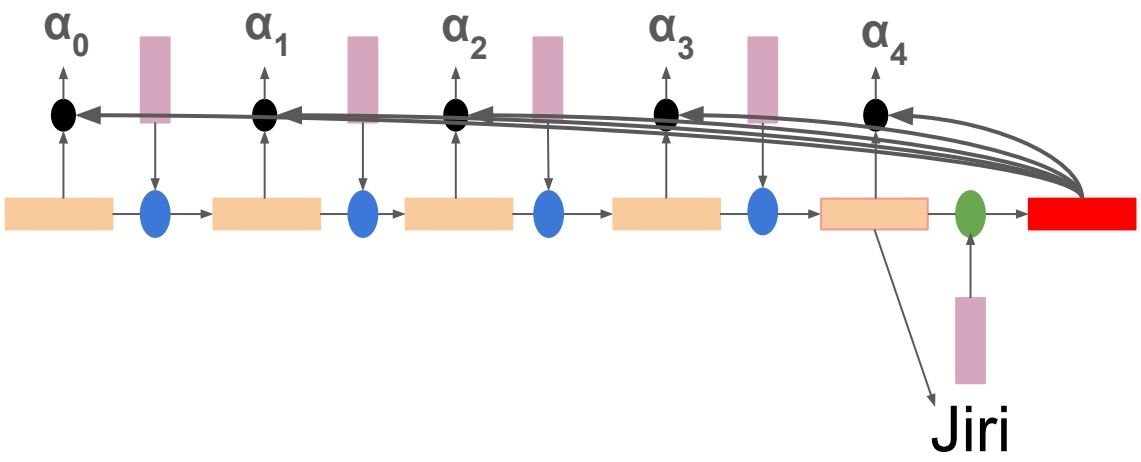
Step 1: Take the decoder state, and compute an *affinity* α_i with all encoder states.



Learning to pay attention

Attention **summarizes the encoder**, focusing on specific parts/words.

Step 1: Take the decoder state, and compute an *affinity* α_i with all encoder states.



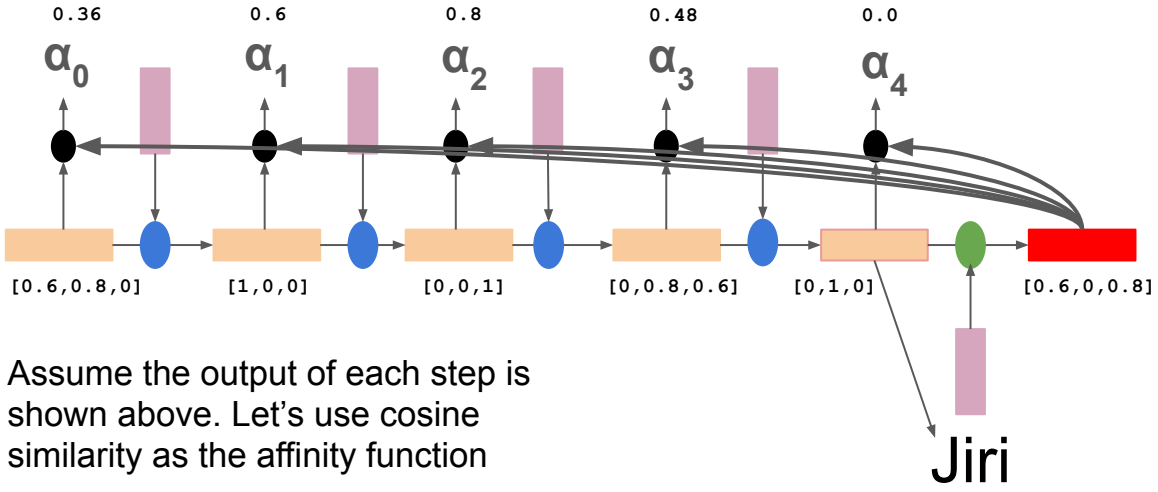
The affinity function, ● , is a dot product, or something similar.

● : = α_i

Learning to pay attention

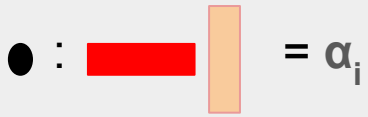
Attention **summarizes the encoder**, focusing on specific parts/words.

Step 1: Take the decoder state, and compute an *affinity* α_i with all encoder states.



Assume the output of each step is shown above. Let's use cosine similarity as the affinity function

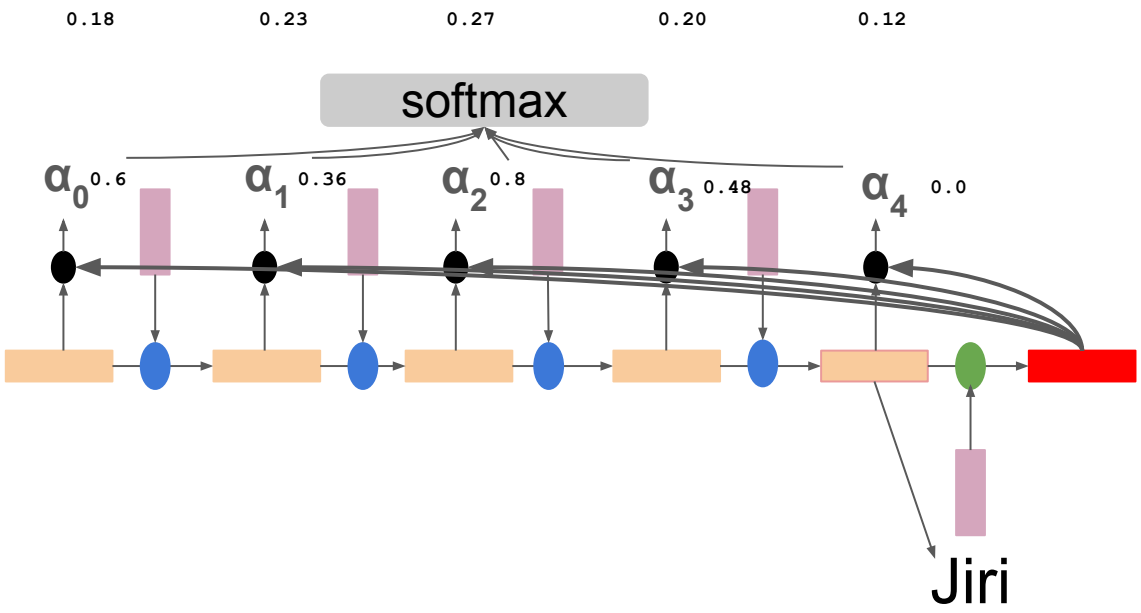
The affinity function, ● , is a dot product, or something similar.



Learning to pay attention

Attention **summarizes the encoder**, focusing on specific parts/words.

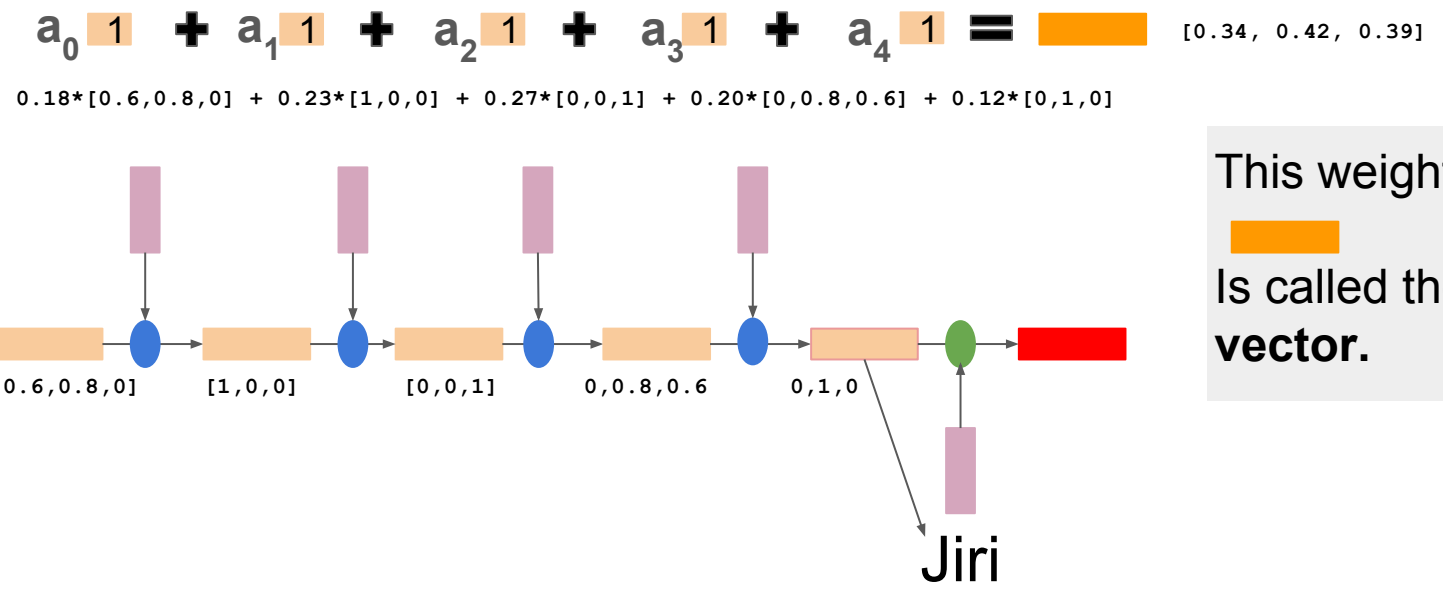
Step 2: Normalize the scores to sum to 1 by the softmax function.



Learning to pay attention

Attention **summarizes the encoder**, focusing on specific parts/words.

Step 3: Average the encoder states, weighted by the **a** distribution.



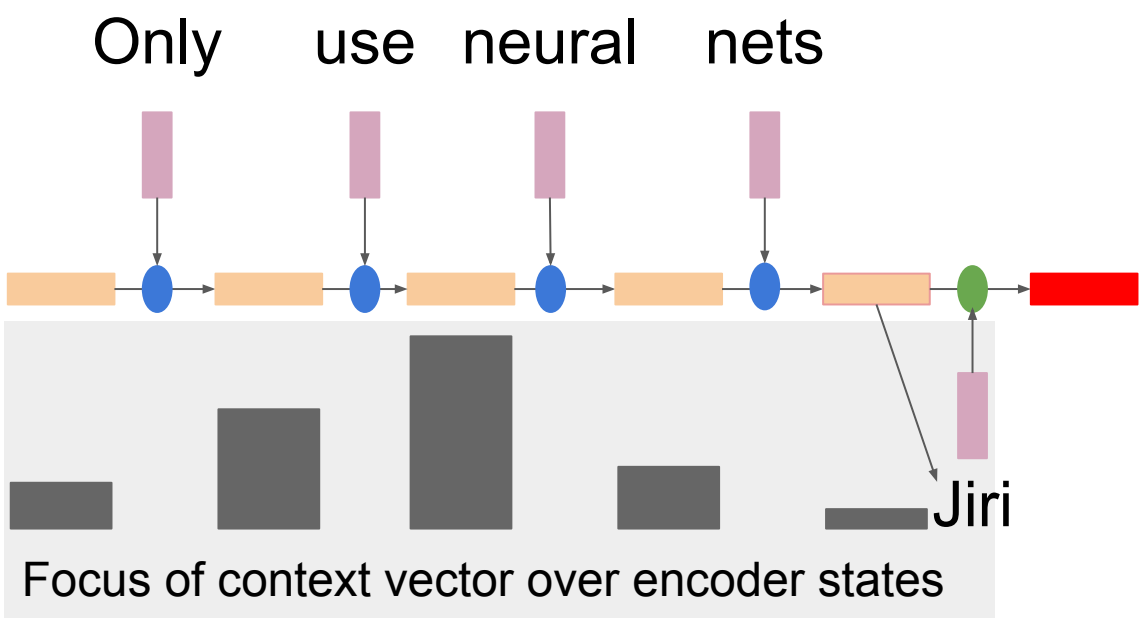
This weighted average

Is called the **context vector.**

Learning to pay attention

Attention **summarizes the encoder**, focusing on specific parts/words.

Step 3: Average the encoder states, weighted by the **a** distribution.

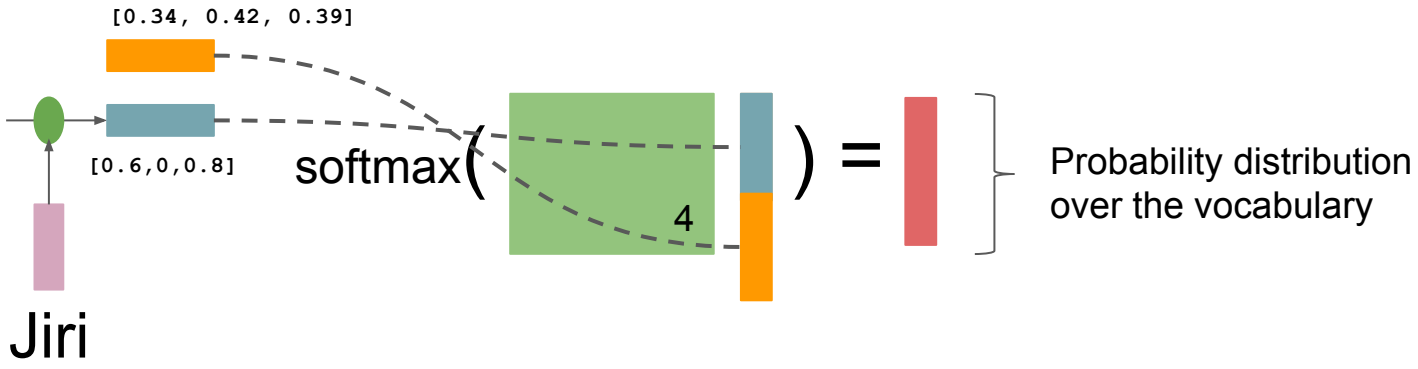


In this example, since “Jiri” means “use”, the attention will focus on the vectors around “use”.

Learning to pay attention

Attention **summarizes the encoder**, focusing on specific parts/words.

Step 4: Use the context vector at prediction, concatenating it to the decoder state.



This vector has the current decoder information, , but also a focused summary of the encoder, .

Attention in NLP Translation:

Neural Machine Translation by Jointly Learning to Align and Translate (Bahdanau et. al, 2015)

Attention-Based Machine Translation

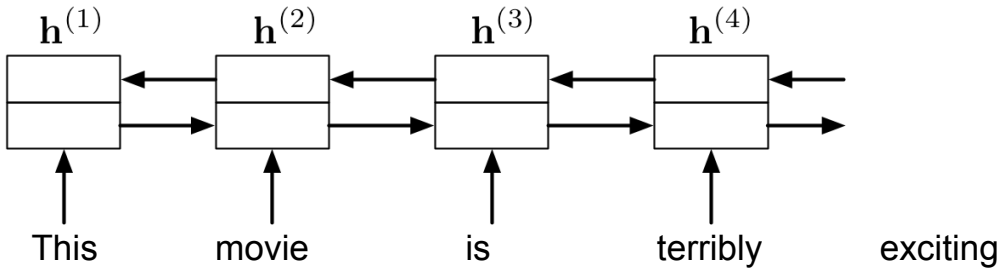
- We'll look at the translation model from the classic paper:

  Bahdanau et al., Neural machine translation by jointly learning to align and translate. ICLR, 2015.

- Basic idea: each output word comes from one word, or a handful of words, from the input. Maybe we can learn to attend to only the relevant ones as we produce the output.

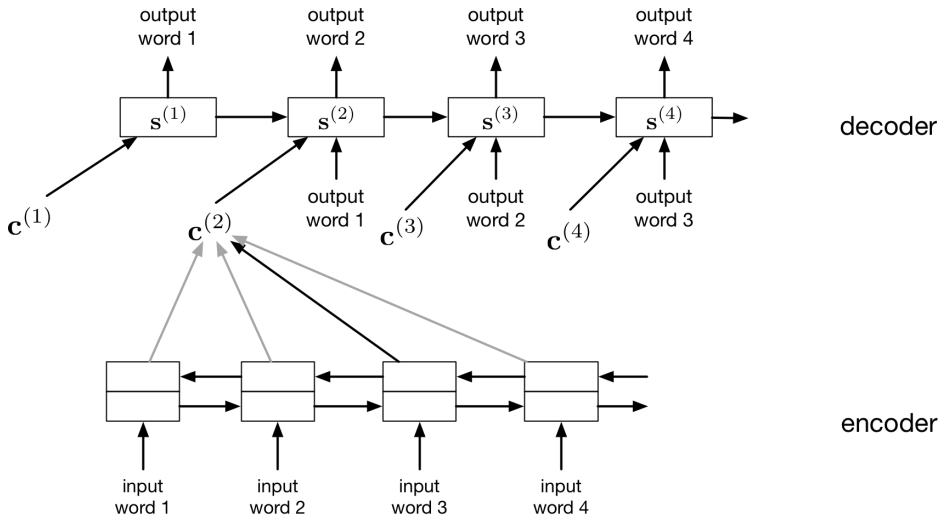
Attention-Based Machine Translation

- The model has both an encoder and a decoder. The encoder computes an annotation of each word in the input.
- It takes the form of a bidirectional RNN. This just means we have an RNN that runs forwards and an RNN that runs backwards, and we concantenate their hidden vectors.
  - The idea: information earlier or later in the sentence can help disambiguate a word, so we need both directions.
  - The RNN uses an LSTM-like architecture called gated recurrent units.



Attention-Based Machine Translation

- The decoder network is also an RNN. Like the encoder/decoder translation model, it makes predictions one word at a time, and its predictions are fed back in as inputs.
- The difference is that it also receives a context vector $\mathbf{c}^{(t)}$ at each time step, which is computed by attending to the inputs.



Attention-Based Machine Translation

- The context vector is computed as a weighted average of the encoder's annotations.

$$\mathbf{c}^{(i)} = \sum_j \alpha_{ij} h^{(j)}$$

- The attention weights are computed as a softmax, where the inputs depend on the annotation and the decoder's state:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = a(\mathbf{s}^{(i-1)}, \mathbf{h}^{(j)})$$

- Note that the attention function depends on the annotation vector, rather than the position in the sentence. This means it's a form of content-based addressing.
  - My language model tells me the next word should be an adjective. Find me an adjective in the input.

Context Vector and Attention

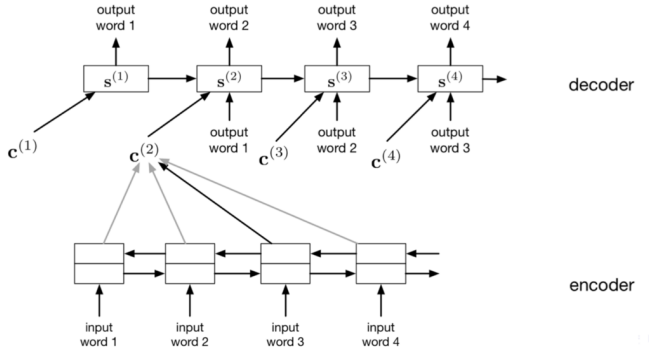
$$\mathbf{c}^{(i)} = \sum_j \alpha_{ij} h^{(j)}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = a(\mathbf{s}^{(i-1)}, \mathbf{h}^{(j)})$$

$$e_{ij} = v_a^\top \tanh\left(W_a s_{i-1} + U_a h_j\right) \quad v_a \in \mathbb{R}^{n'}, W_a \in \mathbb{R}^{n' \times n}\ U_a \in \mathbb{R}^{n' \times 2n}$$

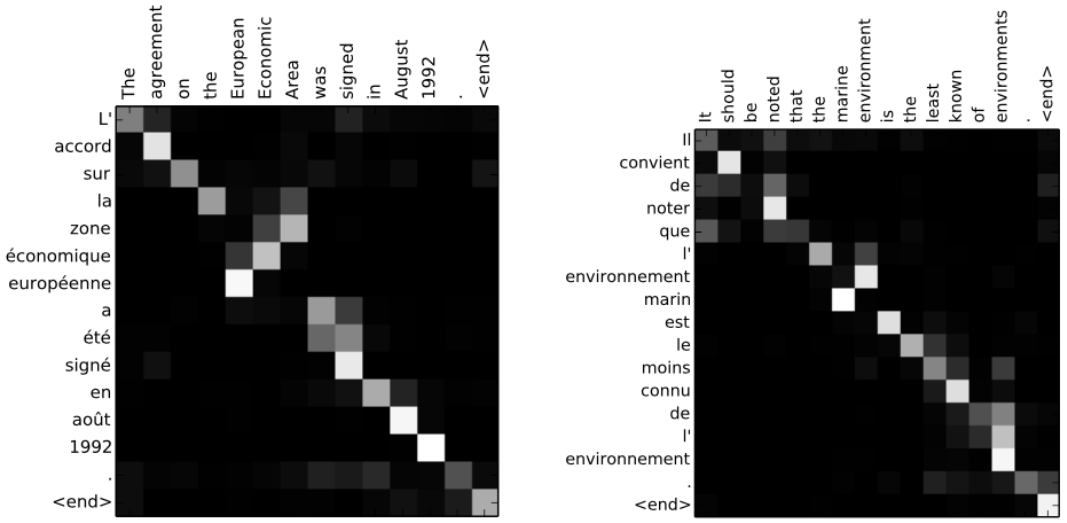
Value Query Key



- e_{ij}: Affinity between s_{i-1} and h_j, e.g. similarity.
- However, s_{i-1} and h_j may have different dimension. Then s_{i-1} and h_j can be transformed into a common space
- Generalized form: Attention (**Query**, **Key**, **Value**), in a similar vein as a retrieval system
- Use attention in any part of the model you like !

Attention-Based Machine Translation

- Here's a visualization of the attention maps at each time step.



- Nothing forces the model to go linearly through the input sentence, but somehow it learns to do it.
  - It's not perfectly linear — e.g., French adjectives can come after the nouns.

Attention Score Functions

Attention Score Functions (1)

- $\boldsymbol{q}$ is the query and $\boldsymbol{k}$ is the key
- **Multi-layer Perceptron** (Bahdanau et al. 2015)

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{w}_2^\intercal \tanh(W_1[\boldsymbol{q}; \boldsymbol{k}])$$

  - Flexible, often very good with large data

- **Bilinear** (Luong et al. 2015)

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^\intercal W \boldsymbol{k}$$

Attention Score Functions (2)

- **Dot Product** (Luong et al. 2015)

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^\intercal \boldsymbol{k}$$

  - No parameters! But requires sizes to be the same.

- **Scaled Dot Product** (Vaswani et al. 2017)

  - Problem: scale of dot product increases as dimensions get larger

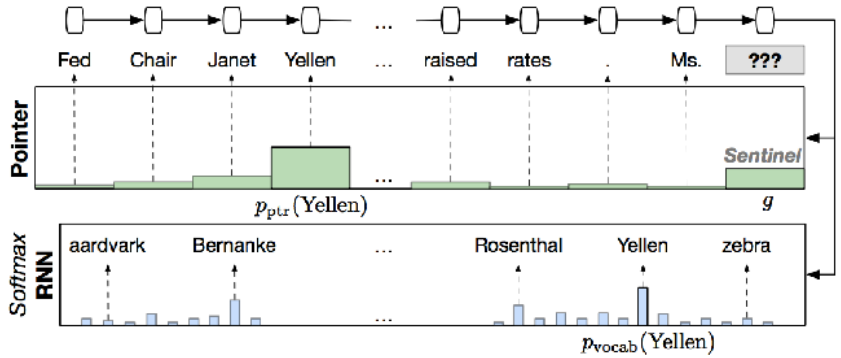
  - Fix: scale by size of the vector

$$a(\boldsymbol{q}, \boldsymbol{k}) = \frac{\boldsymbol{q}^\intercal \boldsymbol{k}}{\sqrt{|\boldsymbol{k}|}}$$

What do we Attend To?

Previously Generated Things

- In language modeling, attend to the previous words (Merity et al. 2016)

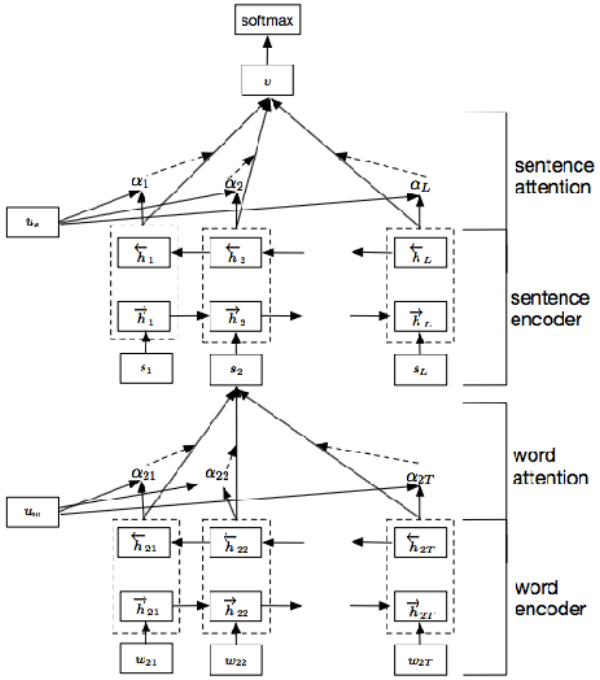


- In translation, attend to either input or previous output (Vaswani et al. 2017)

Hierarchical Structures

(Yang et al. 2016)

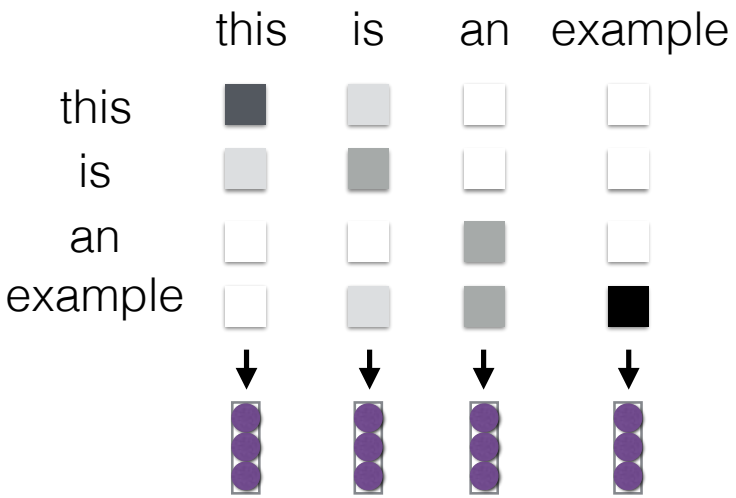
- Encode with attention over each sentence, then attention over each sentence in the document



Intra-Attention / Self Attention

(Cheng et al. 2016)

- Each element in the sentence attends to other elements → context sensitive encodings!



Contextual representations: BERT

- https://jalammar.github.io/illustrated-transformer/
- https://nlp.stanford.edu/seminar/details/jdevlin.pdf

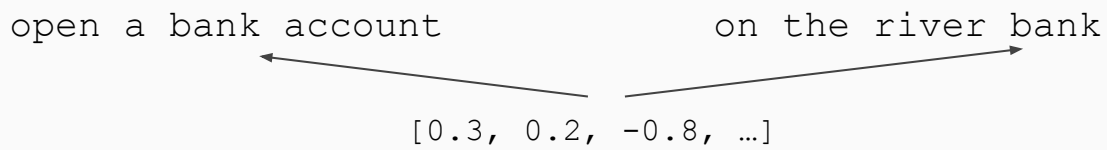
Pre-training in NLP

- Word embeddings are the basis of deep learning for NLP

- Word embeddings (`word2vec`, `GloVe`) are often *pre-trained* on text corpus from co-occurrence statistics

Contextual Representations

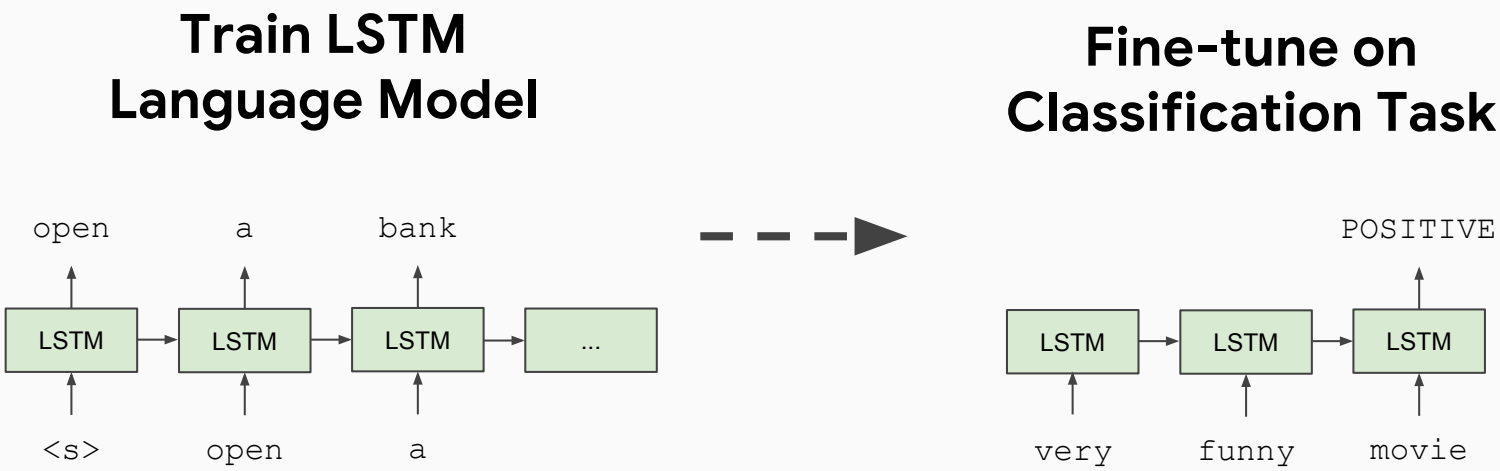
- **Problem**: Word embeddings are applied in a context free manner



- **Solution**: Train *contextual* representations on text corpus

History of Contextual Representations

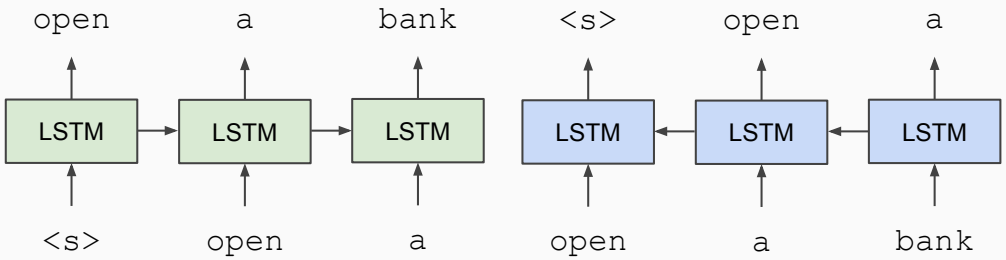
- *Semi-Supervised Sequence Learning*, Google, 2015



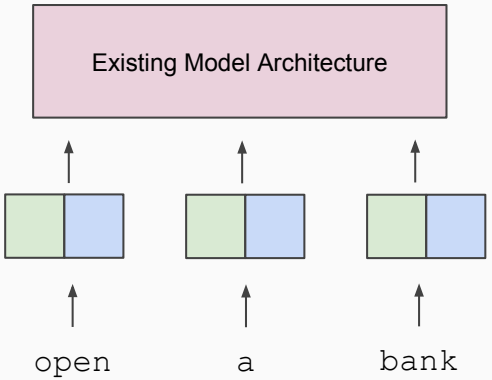
History of Contextual Representations

- *ELMo: Deep Contextual Word Embeddings*, AI2 & University of Washington, 2017

Train Separate Left-to-Right and Right-to-Left LMs



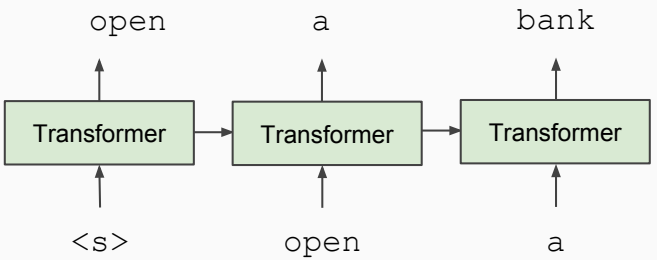
Apply as "Pre-trained Embeddings"



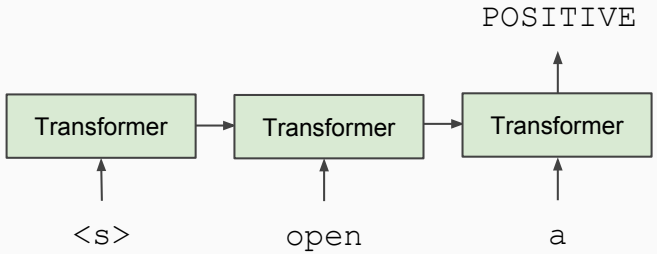
History of Contextual Representations

- *Improving Language Understanding by Generative Pre-Training*, OpenAI, 2018 **(BERT)**

Train Deep (12-layer) Transformer LM



Fine-tune on Classification Task



How good is BERT? GLUE Results

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
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	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

MultiNLI

Premise: Hills and mountains are especially sanctified in Jainism.
Hypothesis: Jainism hates nature.
Label: Contradiction

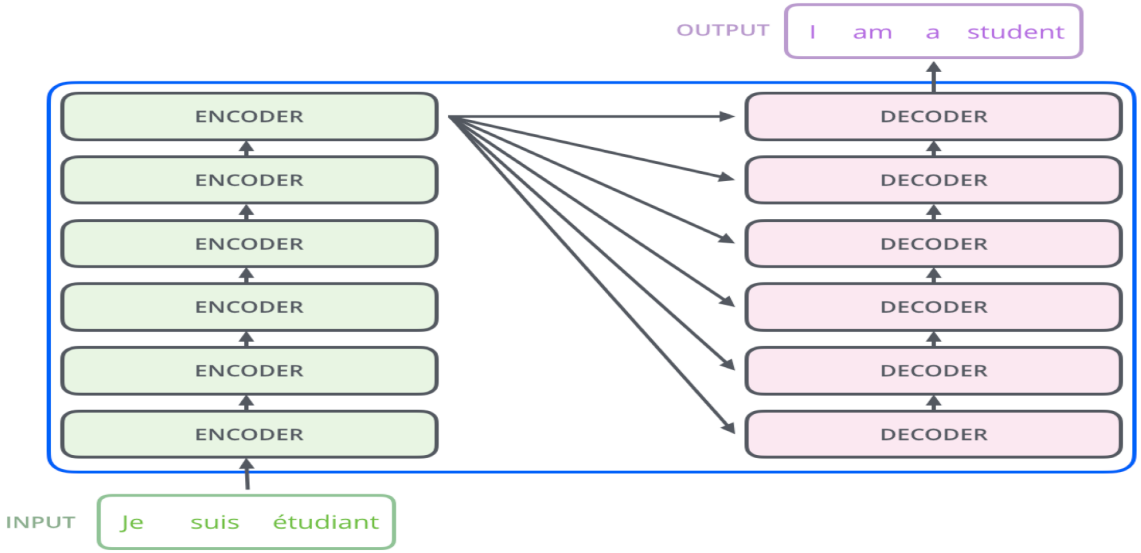
CoLa

Sentence: The wagon rumbled down the road.
Label: Acceptable

Sentence: The car honked down the road.
Label: Unacceptable

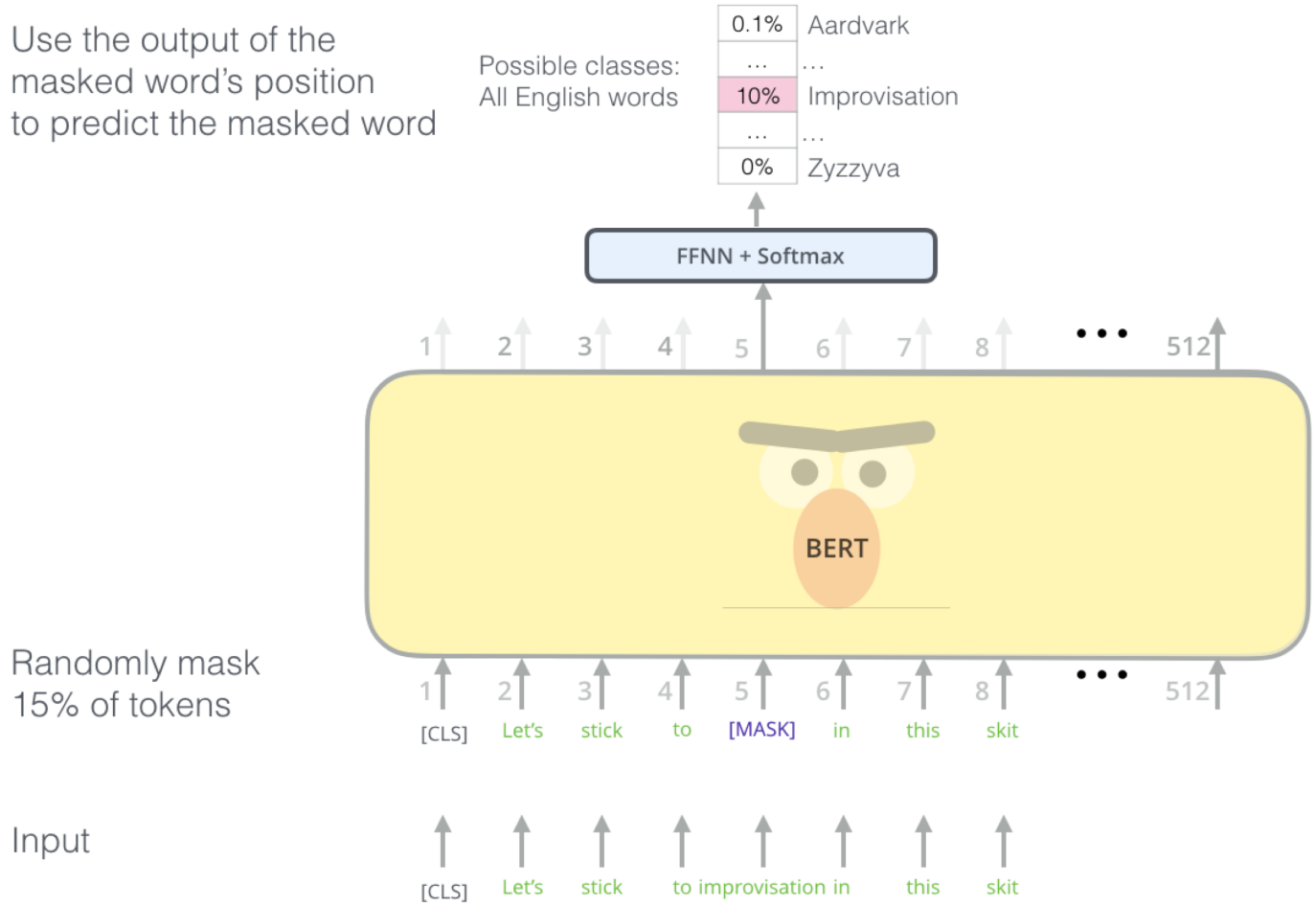
GLUE: The General Language Understanding Evaluation (GLUE) benchmark is a collection of resources for training, evaluating, and analyzing natural language understanding systems

Bidirectional Encoder Representations from Transformers (BERT)



- A trained autoencoder model with many layers (12 or 24) of **Transformer** blocks as encoders/decoders.
- Each transformer is a feedforward-networks with multiple attention heads (12 and 16)
- Trained on Wikipedia (2.5B words) + BookCorpus (800M words) with different NLP tasks:
  - Masked word prediction
  - Next sentence prediction
- Provides pre-trained contextual word representations for downstream NLP tasks

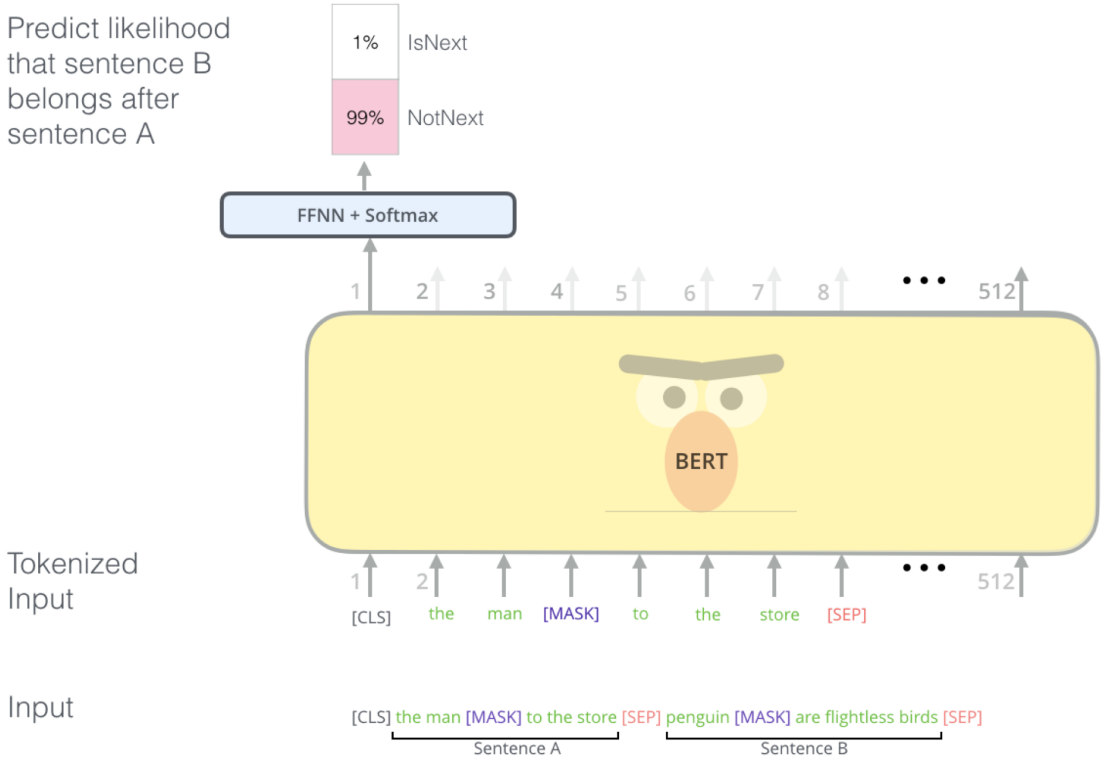
Masked Language Model



Two-sentence Tasks

Some tasks require the model to say something intelligent about two sentences

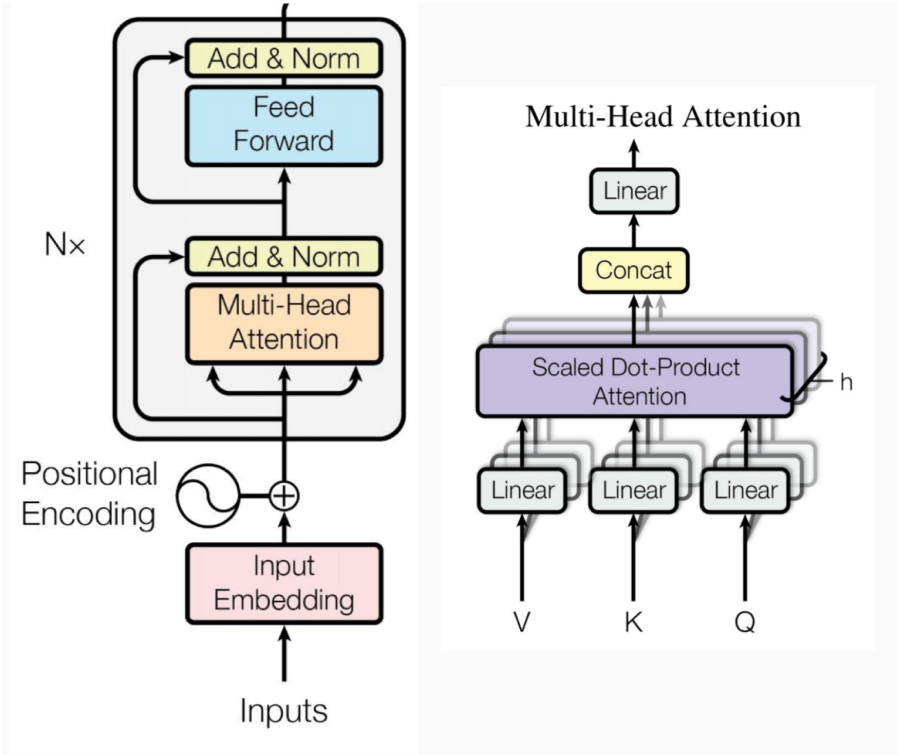
- Duplicates detection
- Q&A



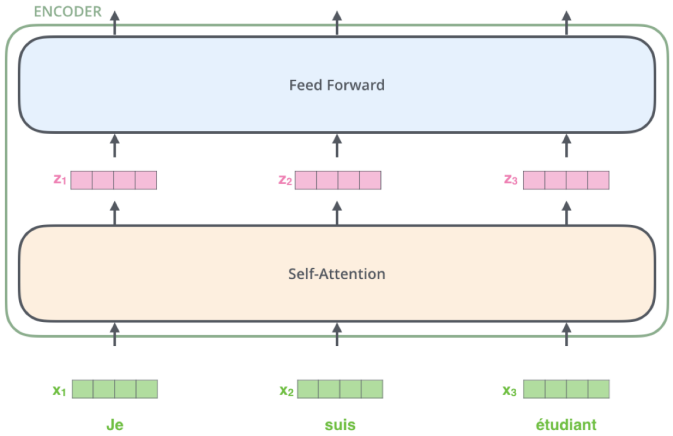
- To make BERT better at handling relationships between multiple sentences, the pre-training process includes an additional task: **Given two sentences (A and B), is B likely to be the sentence that follows A, or not?**

Transformer Architecture

Each encoder /decoder of BERT uses an architecture called transformer



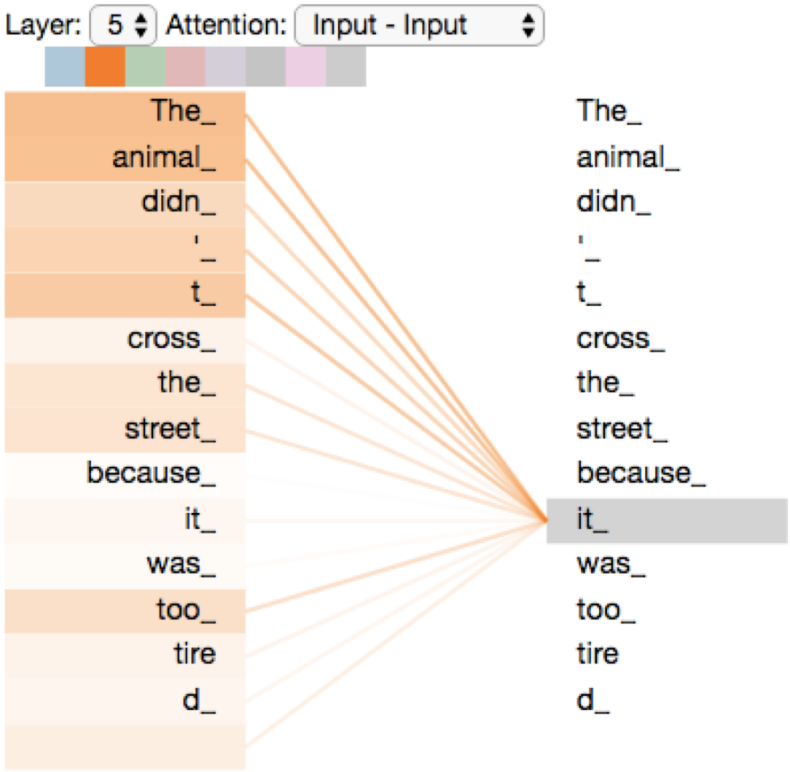
Complete Architecture



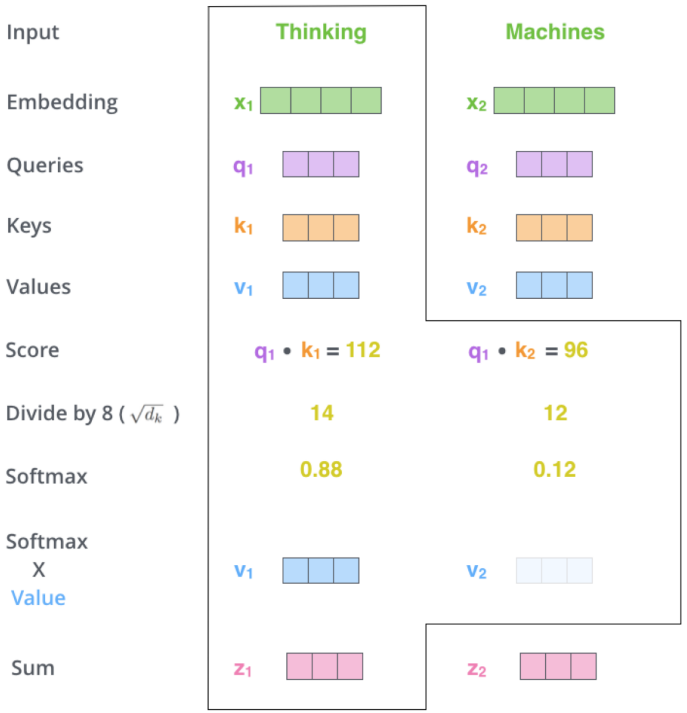
Simplified Architecture

Self Attention

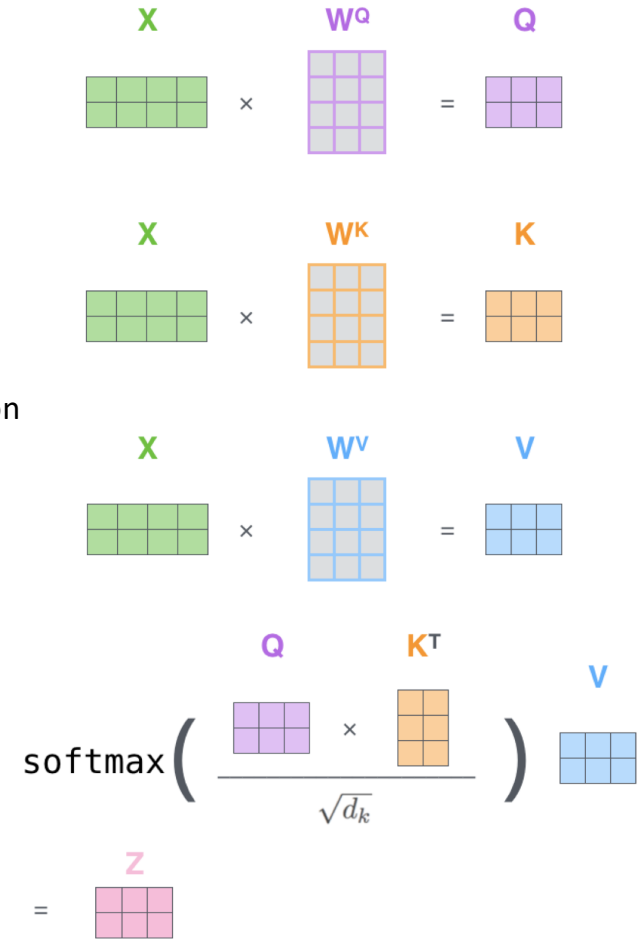
Transformer uses “self attention” to bake the “understanding” of other relevant words into the one currently being processed, i.e. context aware



Self Attention

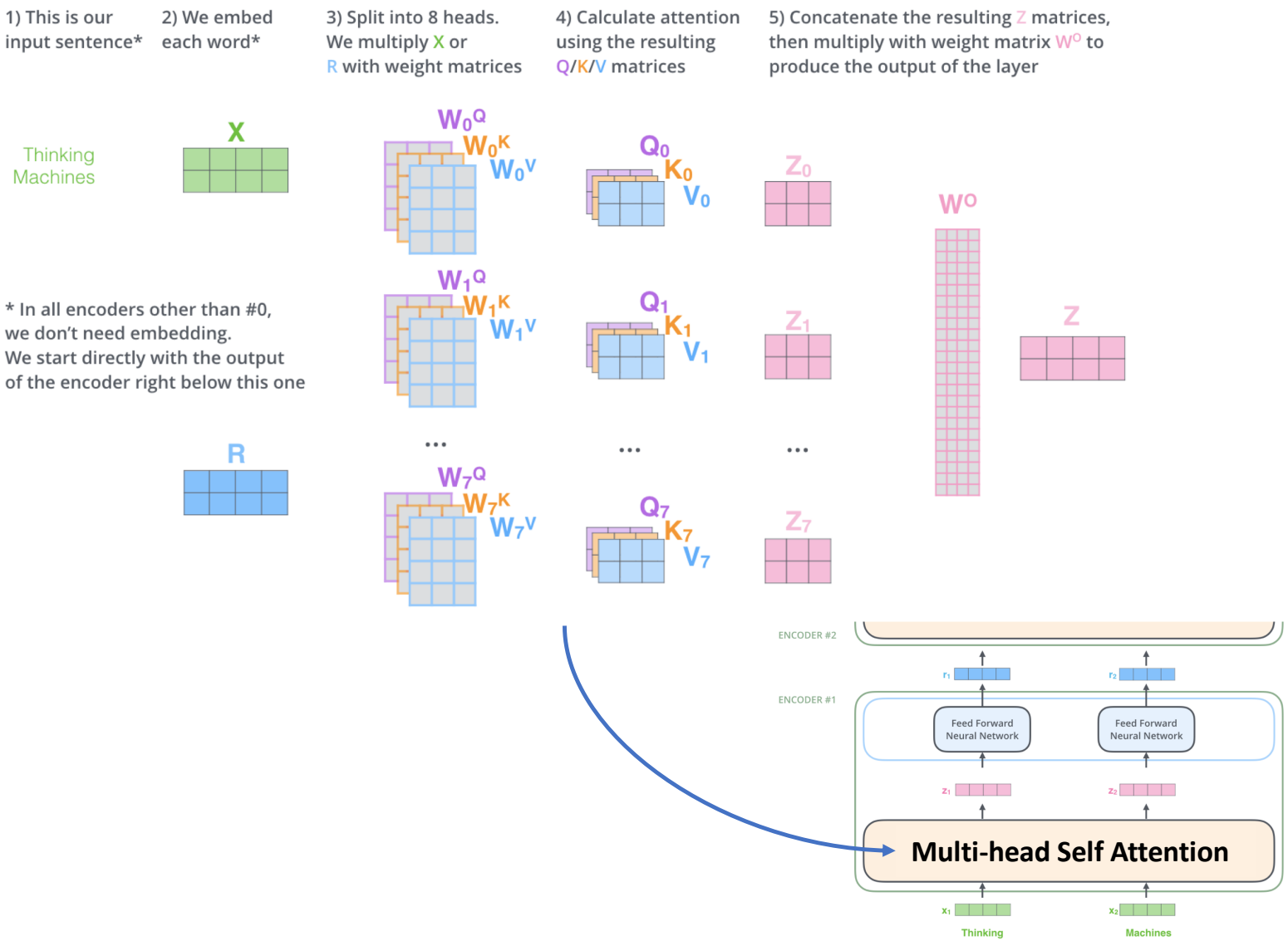


Matrix Operation

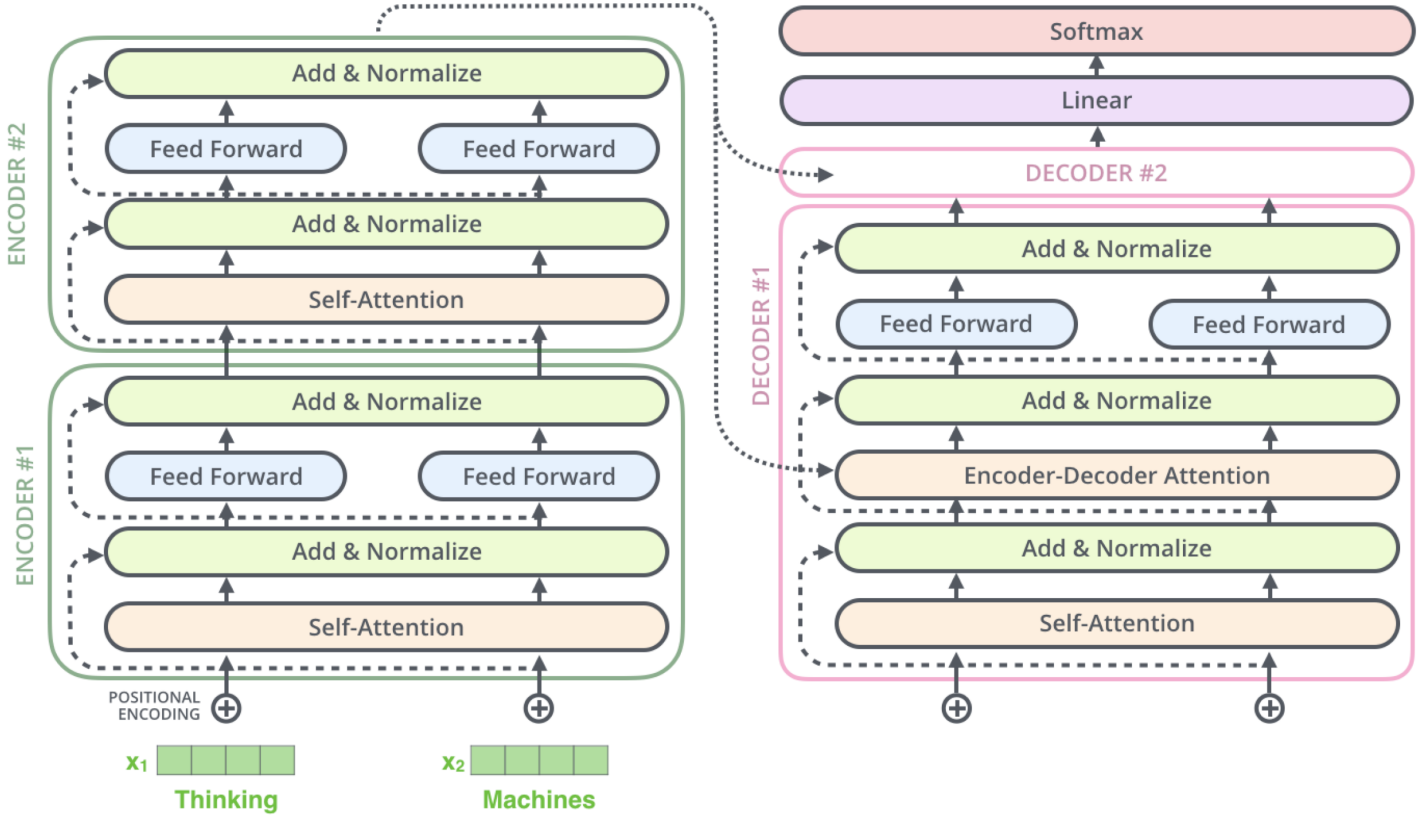


Note that this layer has no LSTM or recurrent construct, making computation more efficient

Multi-head Attention



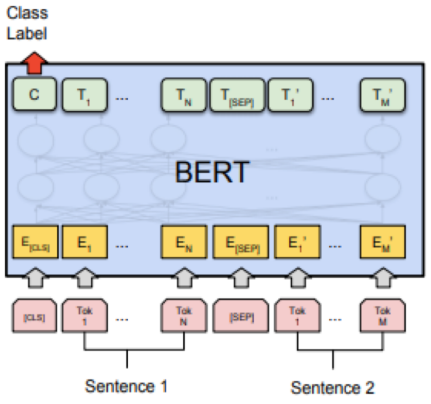
Stacked Transformer Blocks



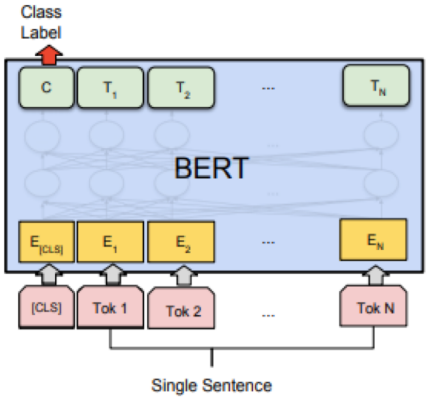
- The output of one block becomes input of the block above
- Residual network
- Layer Normalization
- Encoder-decoder attention

How to use pretrained BERT Model?

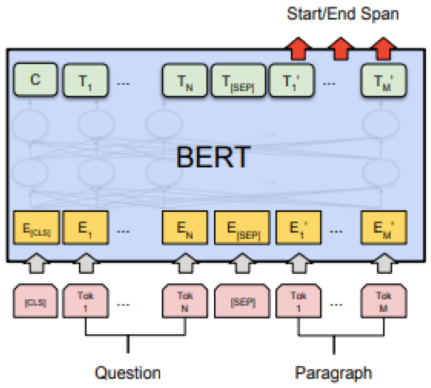
- Bert can be fine tuned for specific NLP tasks
- Process:
  - Remove the last layer (mask or sentence pair prediction)
  - Add task specific layers
  - Continue training with small learning rate



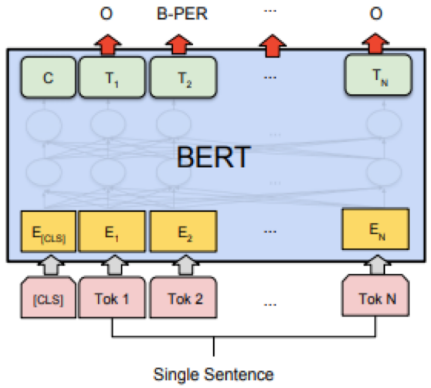
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



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



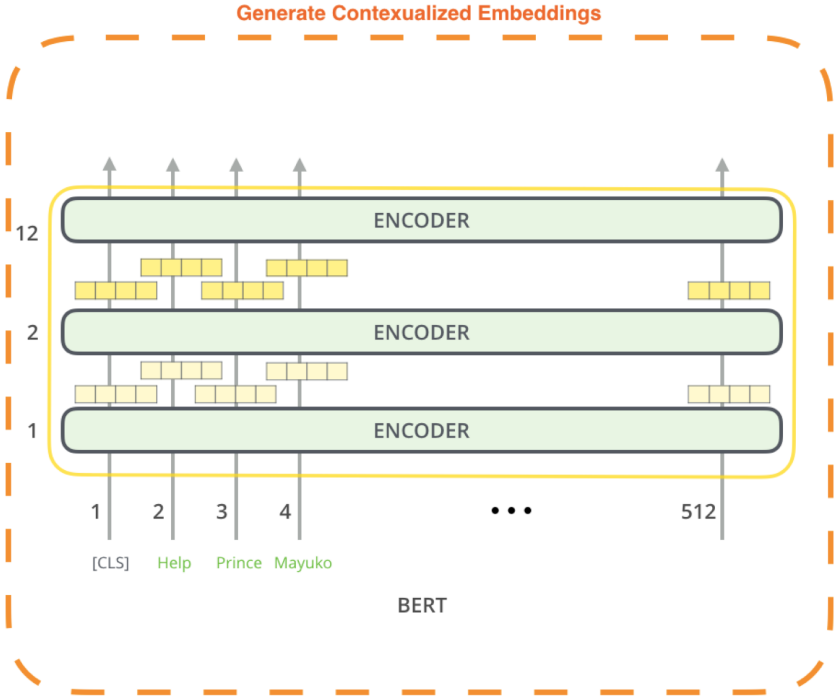
(c) Question Answering Tasks: SQuAD v1.1

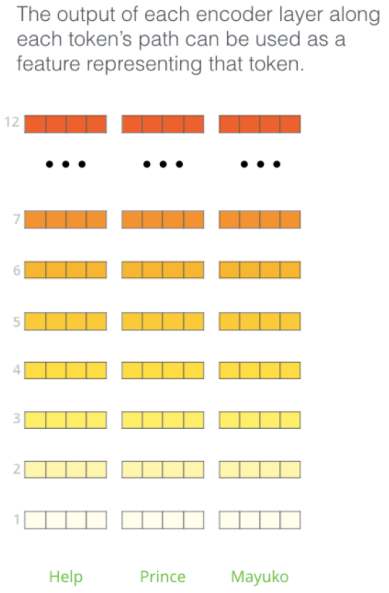


(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

BERT for feature extraction

- Feed your input text to pre-trained BERT model and then extract the output of each encoder
- These outputs can be used as contextualized word embeddings.
- Feed these embeddings to your model – a process the paper shows yield results not far behind fine-tuning BERT on a task such as named-entity recognition.





But which layer output should we use?

Which layer works best as a contextualized embedding?