BST 261: Data Science II
Lecture 15

Attention Models Continued

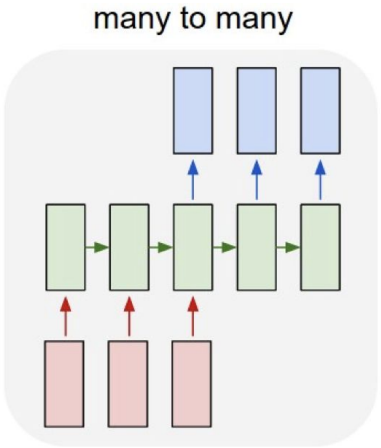
Heather Mattie
Harvard T.H. Chan School of Public Health
Spring 2 2021

Recipe of the Day!

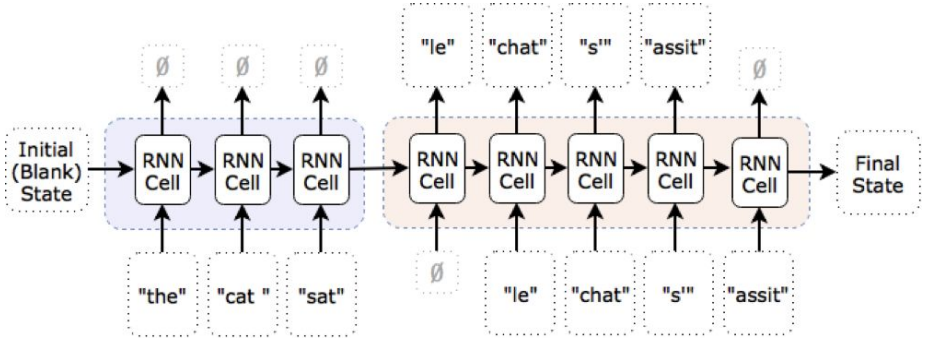
One Bowl Chocolate Chip Banana Bread

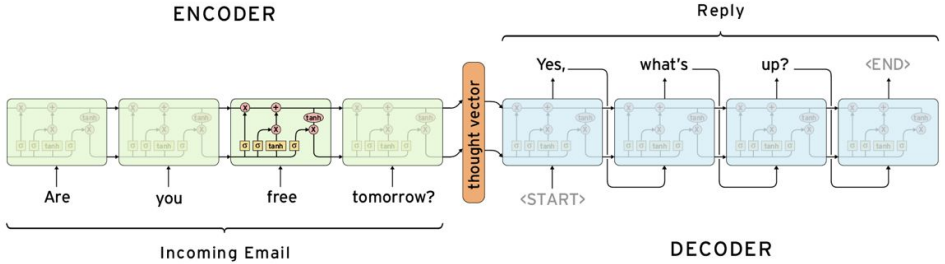
Seq2seq Learning

Recall: Many to Many RNNs



Ex:
Translation,
automated
response





Seq2seq Models

- Sequence-to-sequence models have been successful for tasks like machine translation, text summarization and image captioning

- Both the input and output are sequences
  - Words, letters, features of an image, etc.

- We'll focus on machine translation
  - Input will be a sequence of words and the output will also be a sequence of words

All videos in lecture from amazing post about attention and seq2seq models

Seq2seq Models

- Model consists of an **encoder** (RNN) and a **decoder** (RNN)
  - Encoder processes each item in the input sequence and compiles information into a vector called the **context**
  - The encoder sends the context to the decoder and the decoder produces the output sequence one item at a time

Seq2seq Models

- The context is really a hidden state that is passed to the decoder once

Seq2seq Models

- Major drawback of seq2seq models
  - The context vector acts as a bottleneck
  - These models struggle with very long sequences

- In 2014 and 2015, “**Attention**” was proposed as a solution
  - Attention allows the model to focus on the relevant parts of the input sequence, allowing for longer sequences to be used

Attention

Attention

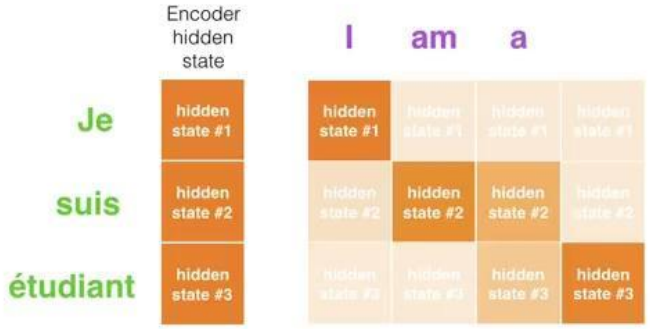
- Rather than passing 1 context vector (the last hidden state) to the decoder, the encoder passes **all** of the hidden states to the decoder
- This means that for each output that the decoder makes, it **has access to the entire input sequence and can selectively pick out specific elements from that sequence to produce the output**.

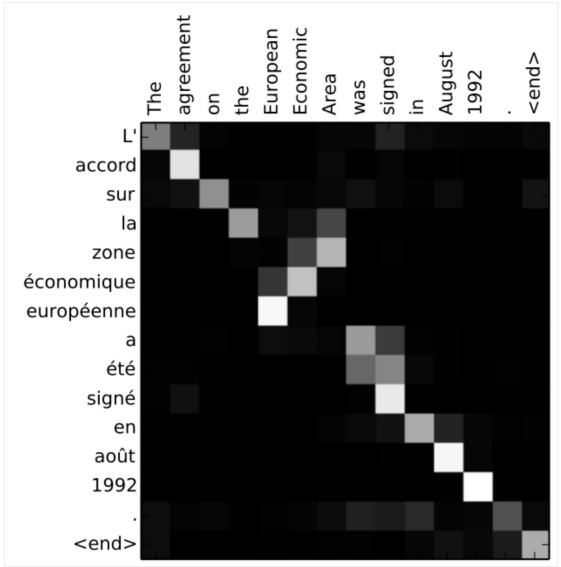
Attention

- An attention decoder does an extra step before producing its output
  - Looks at each of the hidden states from the encoder
  - Scores each of the hidden states
  - Multiplies each hidden state by its softmax score
    - Hidden states with **higher scores** will be **amplified**
    - Hidden states with **smaller scores** will be **dampened** and ignored
  - Sums the weighted vectors into a context vector for that time step

Attention

- Scoring is done by the decoder at each time step
  - For each output word, scoring maps important / relevant words from the input sequence - higher weight means more relevance
  - This helps with the accuracy of the output prediction



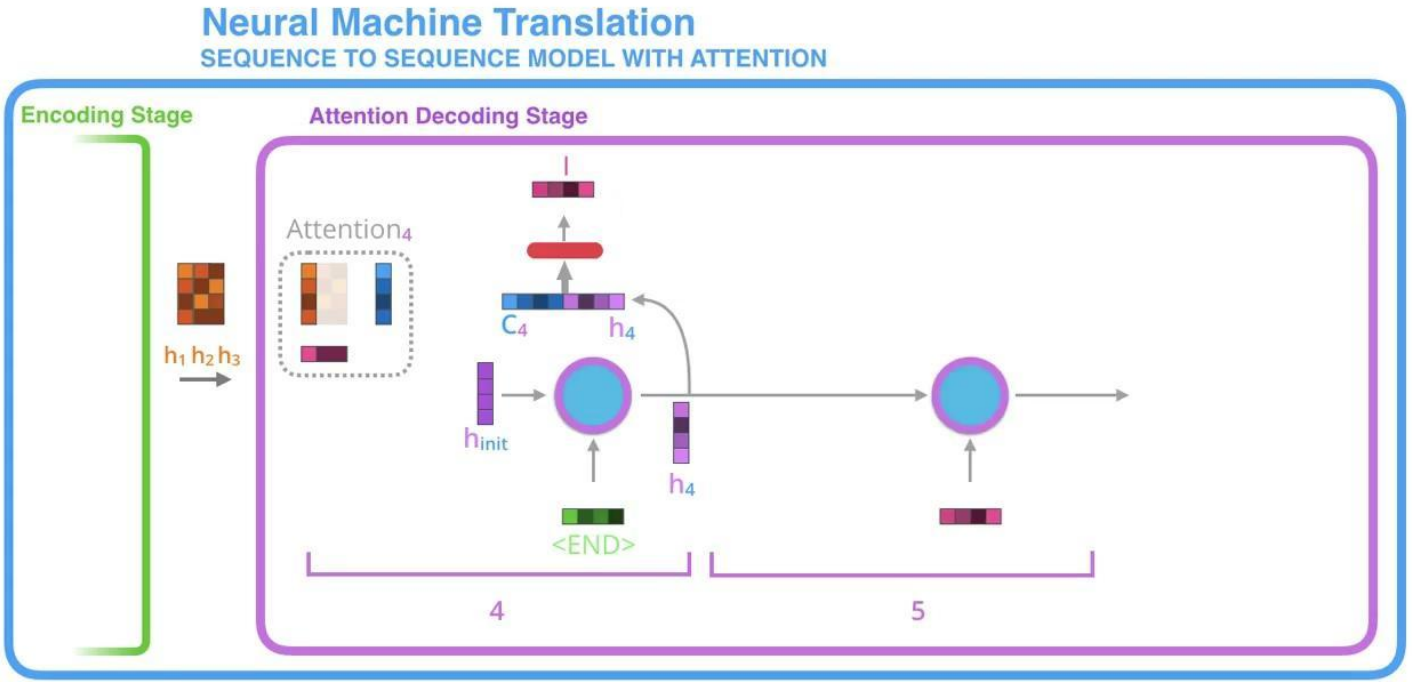


You can see how the model paid attention correctly when outputing "European Economic Area". In French, the order of these words is reversed ("européenne économique zone") as compared to English. Every other word in the sentence is in similar order.

Attention Process

- Send input sequence through encoder
  - Keep track of hidden state at each time point
- Send all hidden states to decoder along with the final hidden state from the encoder
- Use the hidden states from the encoder to calculate the context vector for this time step and concatenate it with the final hidden state from the encoder
- Pass the vector through a feedforward neural network (dense network)
- The output of the dense network is the output word for this time step
- Repeat for the next time steps

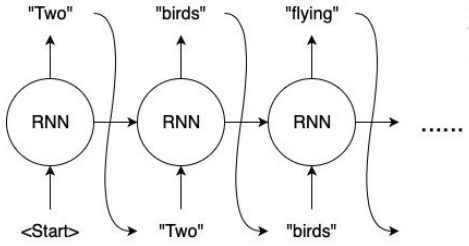
Attention Process



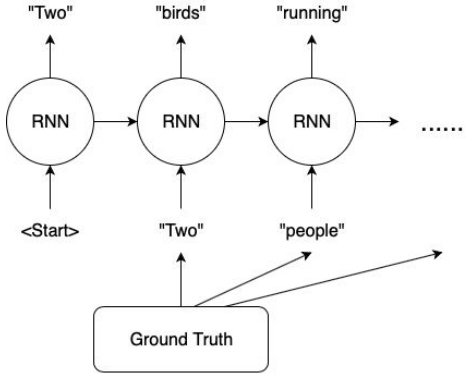
Teacher Forcing

After every prediction, the model is shown the ground truth from the known output and uses this as the input for the next time step - rather than the output generated from the previous time step.

- Short post
- Tutorial
- Book section 10.2.1
- Professor forcing



Without Teacher Forcing



With Teacher Forcing

Teacher Forcing

Pros

- Training converges faster
  - Early stage predictions are usually very bad - teacher forcing speeds up the process of making better predictions

Cons

- Exposure bias
  - During testing we don’t have the ground truth so the predictions the model makes are used as the ground truth
  - A bad/wrong prediction in the beginning can affect the rest of the predictions made and cause a decrease in performance and stability

Transformers

Recall

Problems with RNNs (even LSTM and GRU layers)

- Loss of information (very long sequences)
- Vanishing gradient problem
- No parallel computing
  - Values calculated at the end depend on all previous calculations in all previous time steps
  - Very computationally expensive

Recall

Problems with RNNs (even LSTM and GRU layers)

- Loss of information (very long sequences)
- Vanishing gradient problem
- No parallel computing
  - Values calculated at the end depend on all previous calculations in all previous time steps
  - Very computationally expensive

⇦ Attention helps with these

Recall

Problems with RNNs (even LSTM and GRU layers)

- Loss of information (very long sequences)
- Vanishing gradient problem
- No parallel computing
  - Values calculated at the end depend on all previous calculations in all previous time steps
  - Very computationally expensive

Transformers to the rescue!

Transformers

- Solve all of the problems with classic RNNs
  - Allow for parallel computing
  - Use **attention**
    - Helps with loss of information problem

- [Attention is all you need paper]
  - December 2017
  - Huge breakthrough in NLP

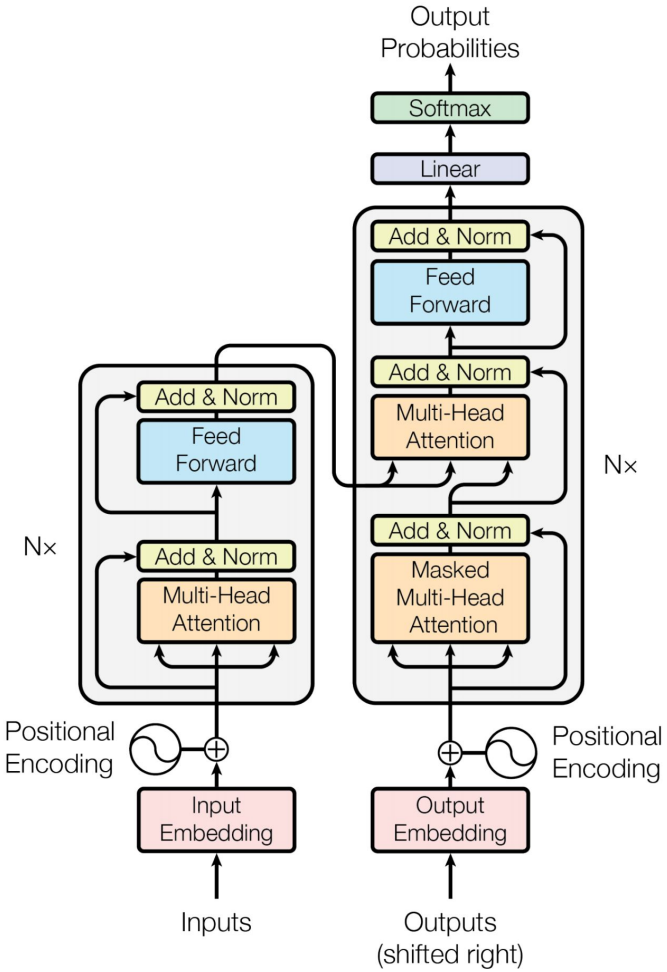


Figure 1: The Transformer - model architecture.

Transformers

- State of the art models
  - GPT-2, GPT-3
  - BERT
  - T5

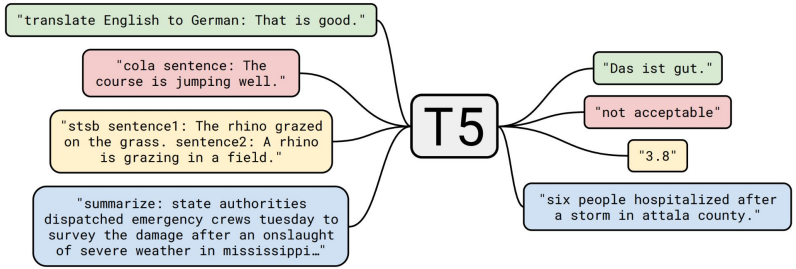
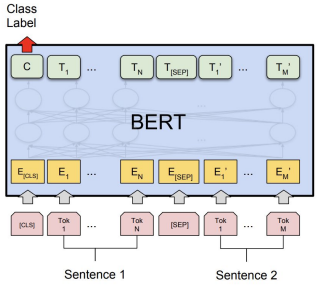
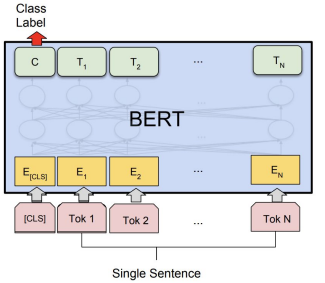


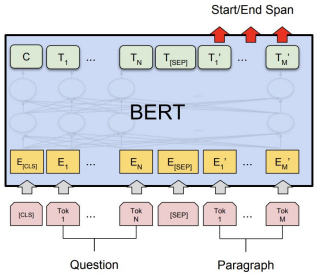
Figure 1: A diagram of our text-to-text framework. Every task we consider—including translation, question answering, and classification—is cast as feeding our model text as input and training it to generate some target text. This allows us to use the same model, loss function, hyperparameters, etc. across our diverse set of tasks. It also provides a standard testbed for the methods included in our empirical survey. "T5" refers to our model, which we dub the "**T**ext-**t**o-**T**ext **T**ransfer **T**ransformer".



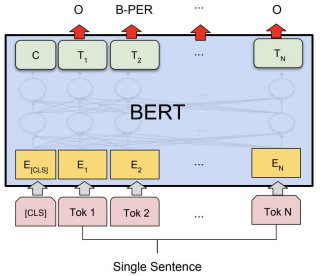
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



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



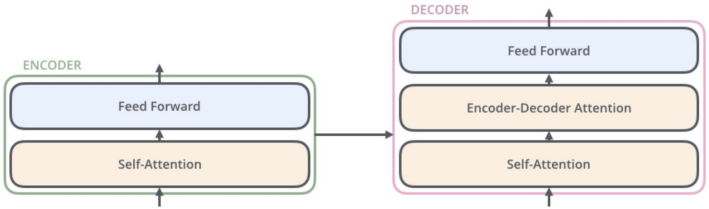
(c) Question Answering Tasks: SQuAD v1.1

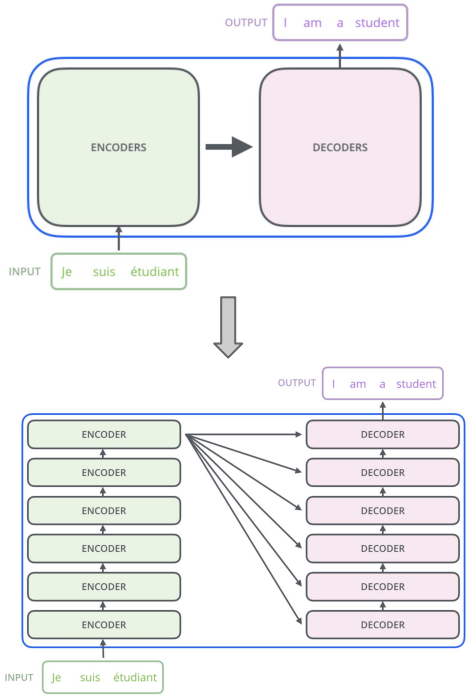


(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Transformers

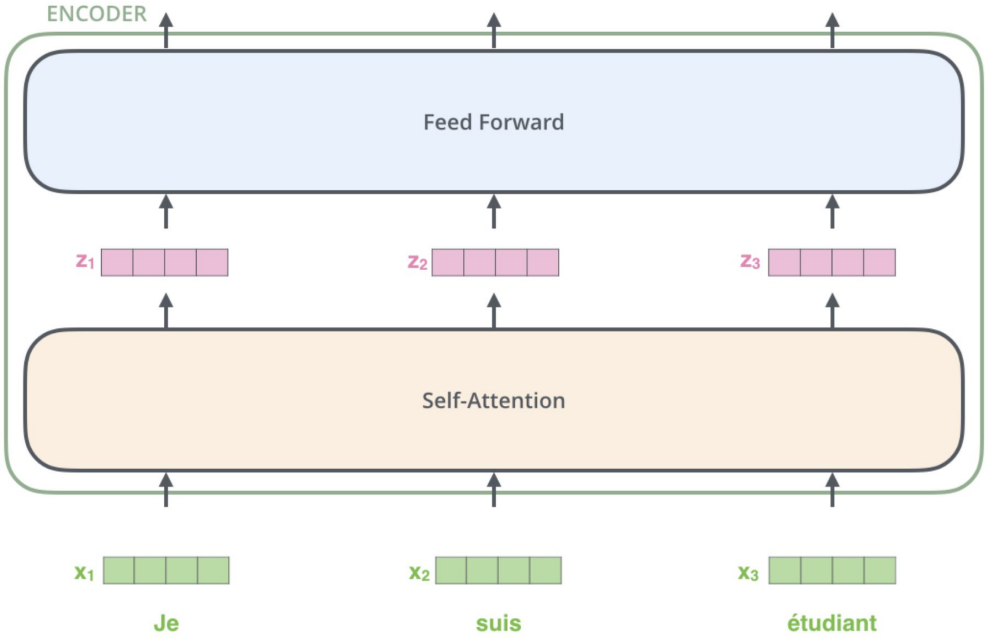
- Consist of “transformer blocks”
  - There are encoder and decoder transformer blocks
    - 6 layers each in the original paper
  - Each encoder block contains a self-attention layer and a dense (feedforward) network
  - Each decoder block contains a self-attention layer, an encoder-decoder attention layer, and a dense (feedforward) network





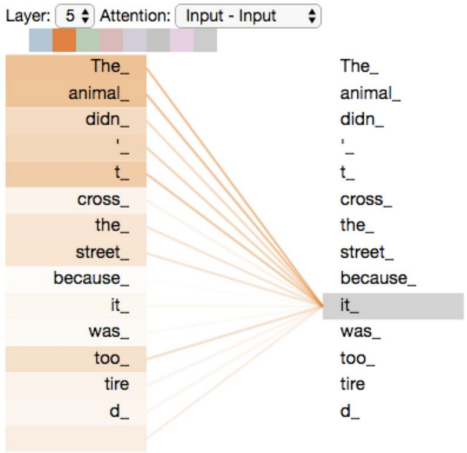
[All depictions from this amazing post]

Encoder Block



Self-Attention

- Self-attention is attention that takes place in one block - rather than between an encoder and decoder
- Uses the input to learn context and identify important words

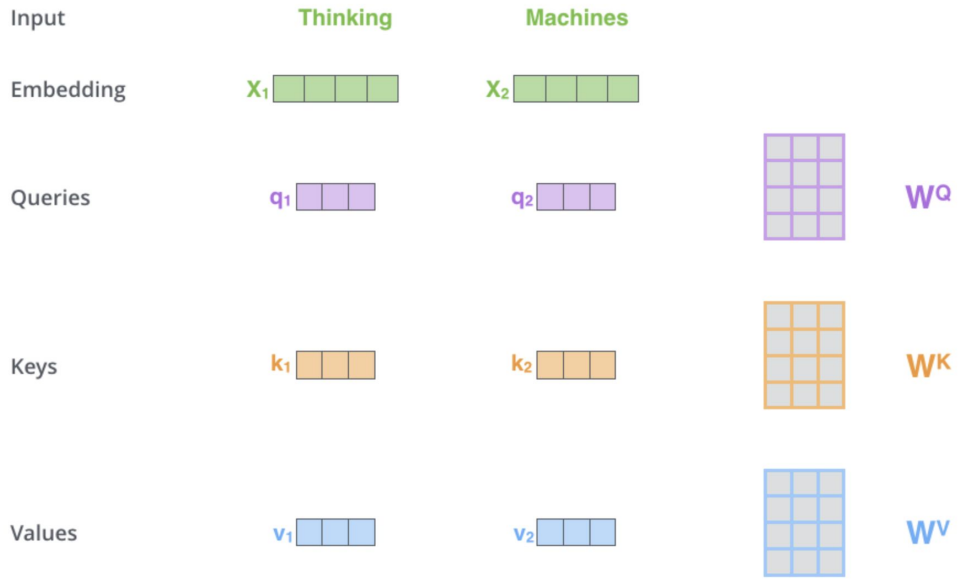


As we are encoding the word "it" in encoder #5 (the top encoder in the stack), part of the attention mechanism was focusing on "The Animal", and baked a part of its representation into the encoding of "it".

Self-Attention

Steps

1. Create 3 vectors from the embedding vector of each word in the input sequence - query, keys, and values vectors.
    - These are created by multiplying the embedding vector by the query, keys, and values matrices that are trained during the training process



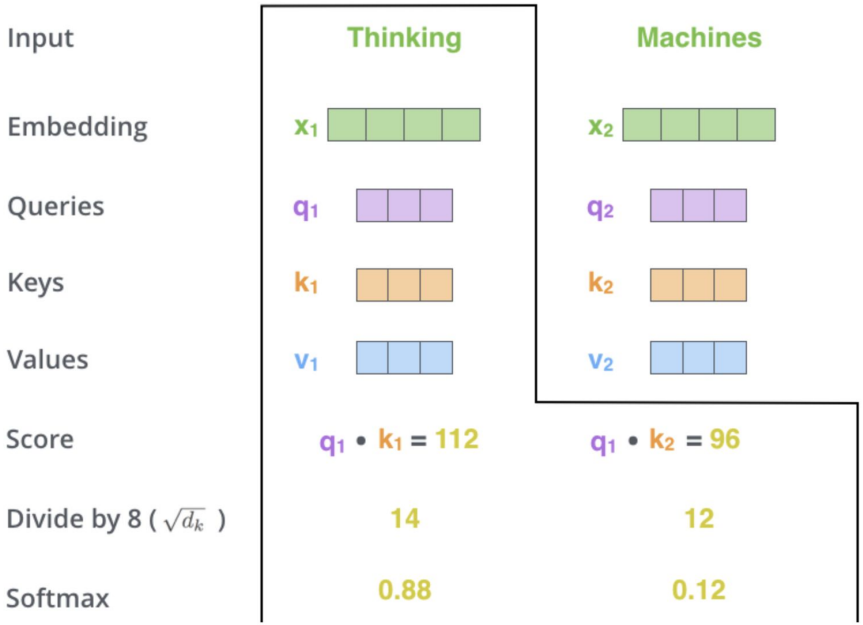
Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

Self-Attention

2. Calculate a score for each word
    - Dot product of the query and key vectors
    - Determines which other words in the input sequence to focus on

3. Divide scores by $\sqrt{d_k}$ (d_k is the dimension of the key vector)
    - Leads to more stable gradients

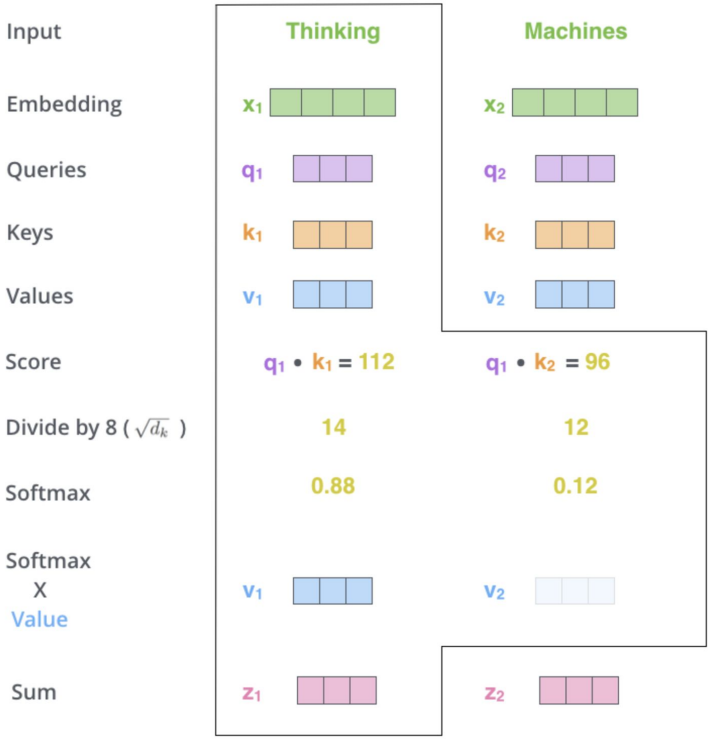
4. Pass the result through a softmax operation



Self-Attention

5. Multiple each value vector by the softmax score
    - Keeps the values of important words intact and minimizes the values of non-informative words (words you can ignore)

6. Sum the weighted value vectors
    - This is the self-attention output



Multi-Head Attention

Main idea: self-attention, multiple times

- Improves performance of the self-attention layer
  - Expands the model's ability to focus on different positions in the input sequence
  - Gives the attention layer multiple "representation subspaces"
    - Multiple sets of query, key, and value matrices

Multi-Head Attention

1) This is our input sentence*

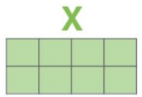
2) We embed each word*

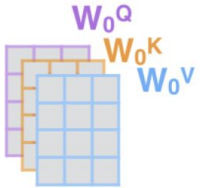
3) Split into 8 heads. We multiply X or R with weight matrices

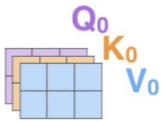
4) Calculate attention using the resulting Q/K/V matrices

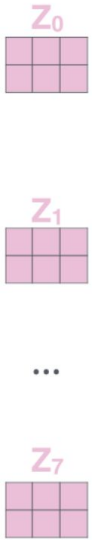
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

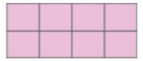
Thinking Machines





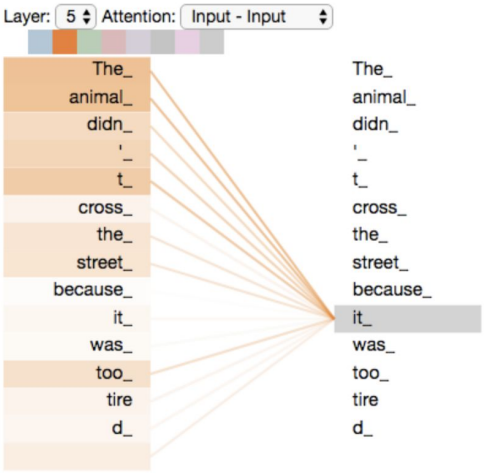




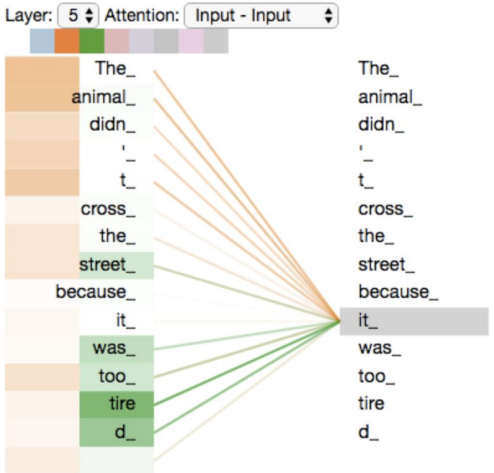


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

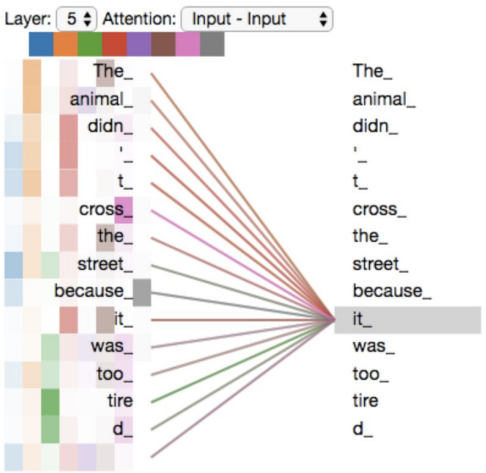
Multi-Head Attention



Self-attention



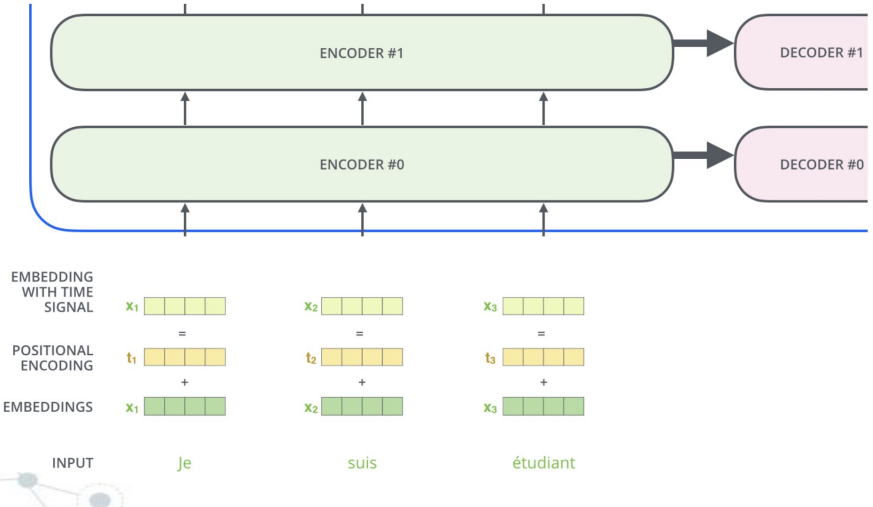
2-head Self-attention



8-head Self-attention

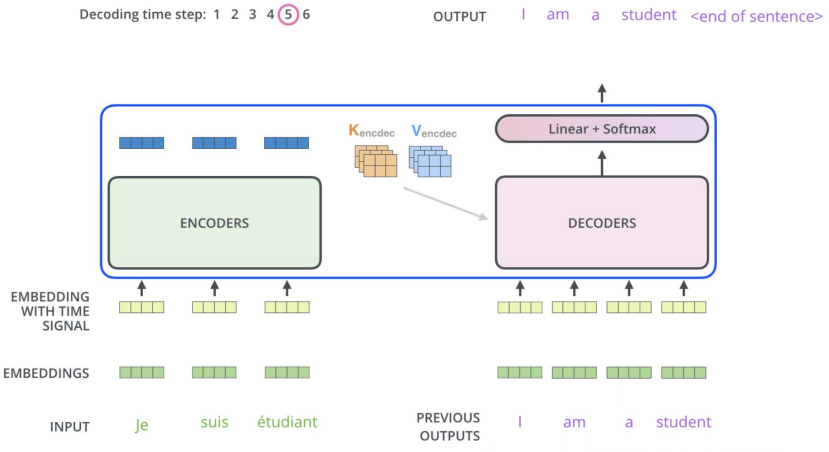
Positional Encoding

- Accounts for the ordering of the words in the input sequence
- A vector is added to the embedding vector
  - Model learns the position of the word in the sequence, as well as the distance between different words in the sequence



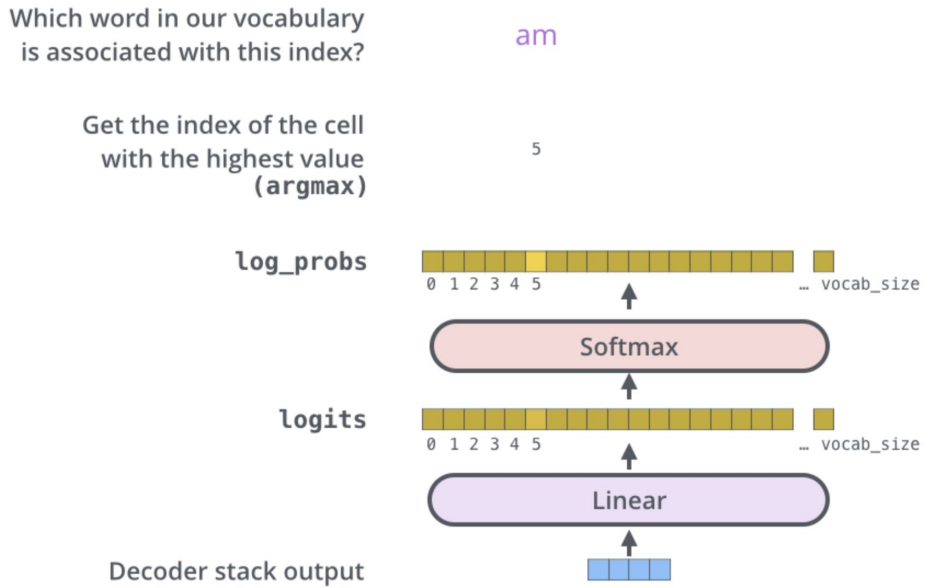
The Decoder

- The inner workings of the decoder are very similar to the encoder
  - Extra layer: “**encoder-decoder attention**” layer where the encoder sends information to the decoder to help with attention and prediction
    - Creates its query matrix from the layer below
    - Takes the keys and values matrices from the output of the encoder
  - Self-attention within the decoder blocks is a little different
    - Only allowed to attend to earlier positions in the output sequence



The Final Layer

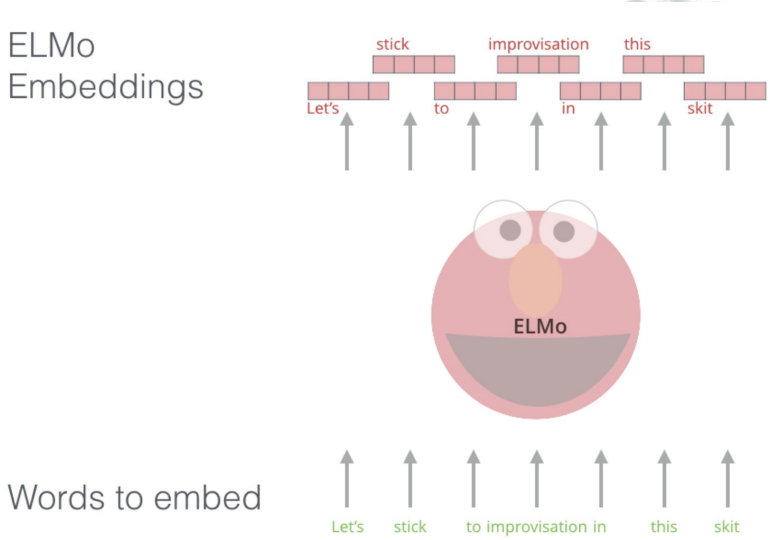
- The decoder outputs a vector that is transformed into a larger vector by a dense (linear) network
  - Output is called a logits vector
  - Each cell is the score of a particular word
- Pass the logits vector through a softmax function to get probabilities for each word
- The cell with the highest probability is chosen



ELMo

ELMo

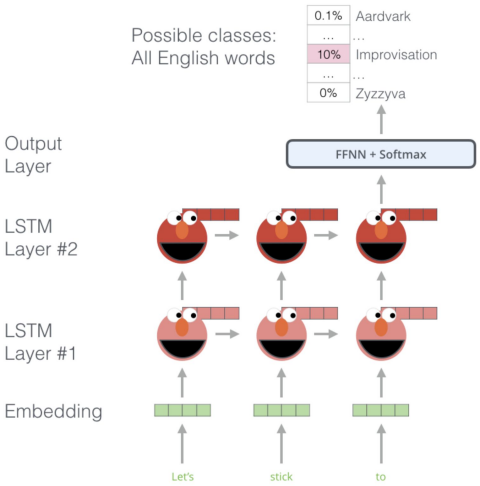
- Recall: word embeddings are vector representations of words
  - We've used GloVe in class and in lab
  - Only 1 embedding for each word (fixed embeddings)
- Problem: the same word can have different meanings depending on the **context** in which it is used
- ELMo looks at an entire sequence before assigning each word it in an embedding
- Uses a bi-directional LSTM trained on a specific task

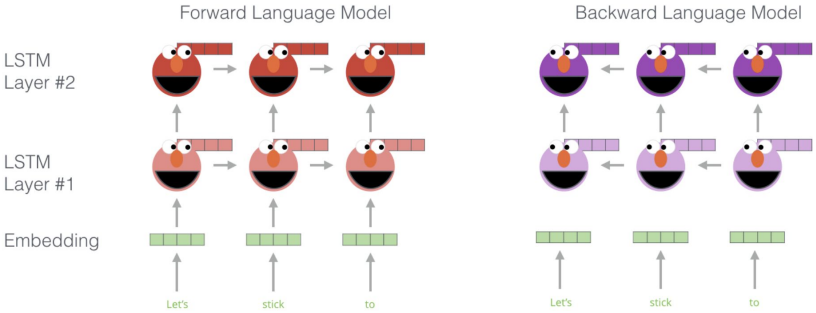


ELMo

- Provided a significant step towards pre-training in NLP
- Was trained to predict the next word in a sequence of words

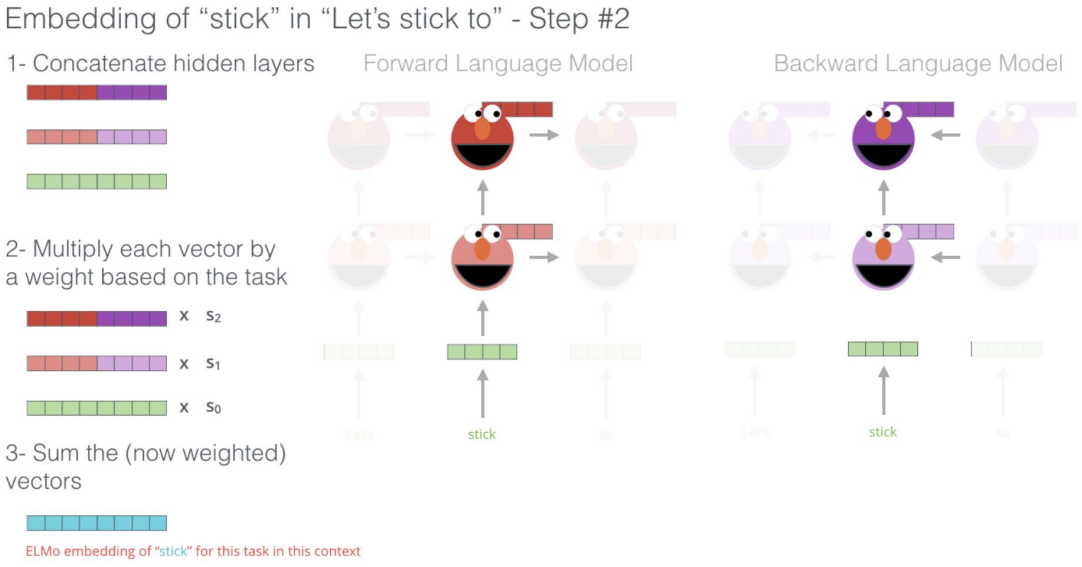
- Trains bi-directional LSTM
  - Has a sense of the next word as well as the previous word





ELMo

- Creates the contextualized embedding through grouping together he hidden states and initial embedding in a certain way



ULM-FiT

Universal Language Model Fine-tuning for Text Classification (ULM-FiT)

- What led to transfer learning for NLP
  - Original paper
- Introduced methods to utilize a lot of what the model learns during pre-training
  - Produced a language model that can be fine-tuned to perform various tasks

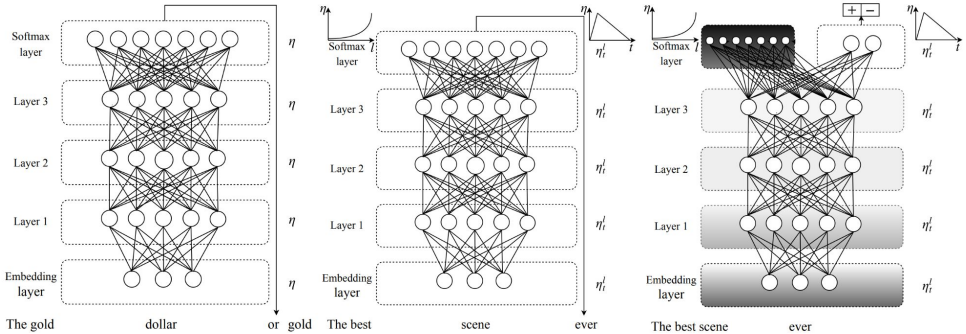
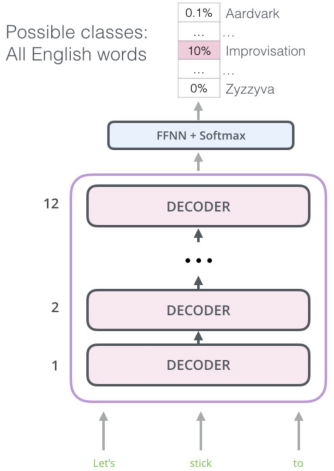


Figure 1: ULMFiT consists of three stages: a) The LM is trained on a general-domain corpus to capture general features of the language in different layers. b) The full LM is fine-tuned on target task data using discriminative fine-tuning ('*Discr*') and slanted triangular learning rates (STLR) to learn task-specific features. c) The classifier is fine-tuned on the target task using gradual unfreezing, '*Discr*', and STLR to preserve low-level representations and adapt high-level ones (shaded: unfreezing stages; black: frozen).

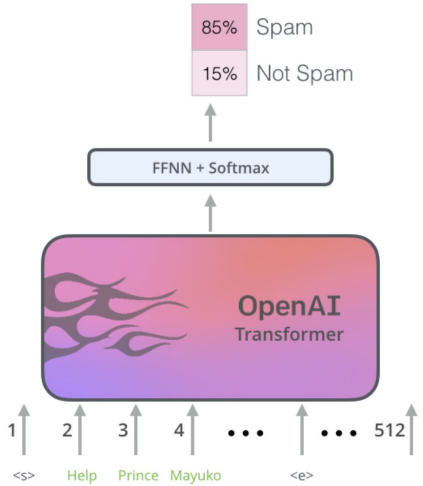
OpenAI Transformer

- Pre-training a transformer decoder for language modeling
  - Don’t need an entire encoder-decoder transformer - just the decoder
  - 12 decoder layers
  - No encoder-decoder attention layer since no encoder
  - Trained to predict the next word using **7,000 books** (BooksCorpus dataset)
  - Original paper

The OpenAI Transformer is made up of the decoder stack from the Transformer



Transfer Learning



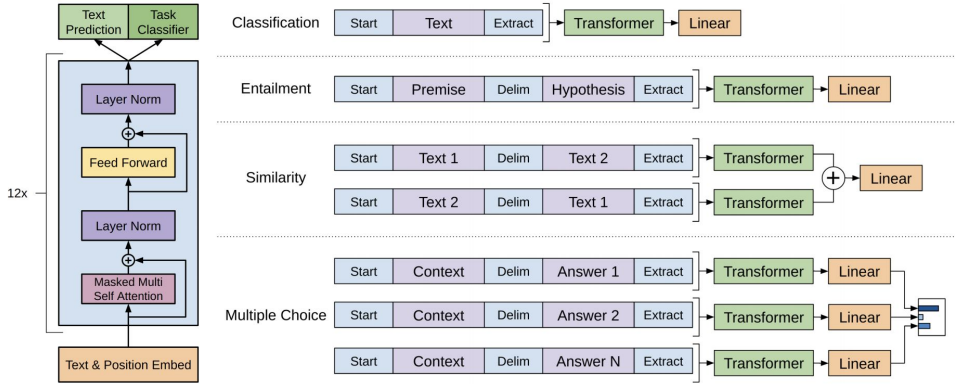
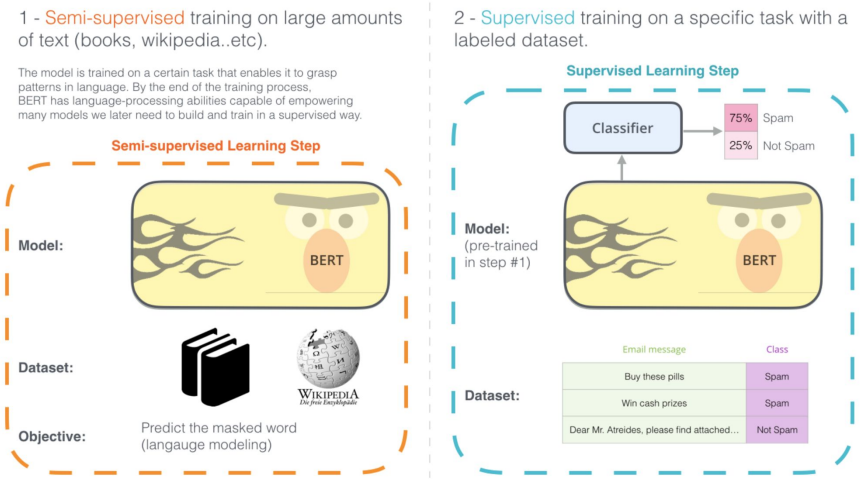


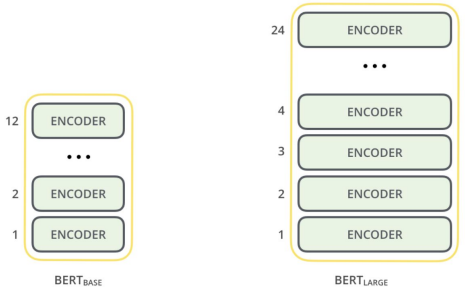
Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

BERT

Bidirectional Encoder Representations Transformer (BERT)

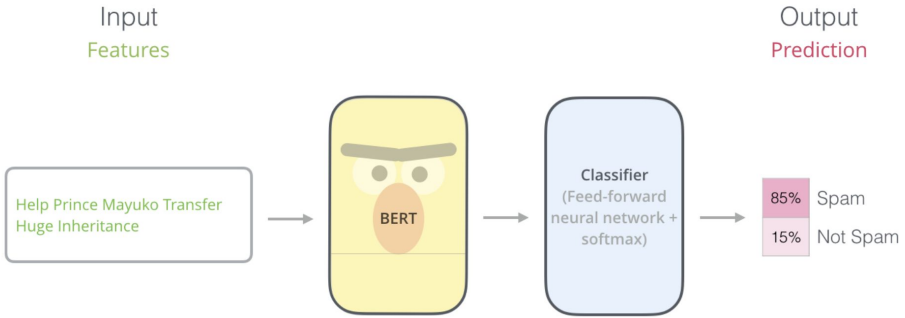
- Has been described as the “marking the beginning of a new era in NLP”
  - Has broken several records for language-based tasks
  - Has been trained on massive datasets
  - Was made available as a pre-trained network
  - Now available in multiple sizes
- [Original paper]
- [GitHub repository]
- [Great post]





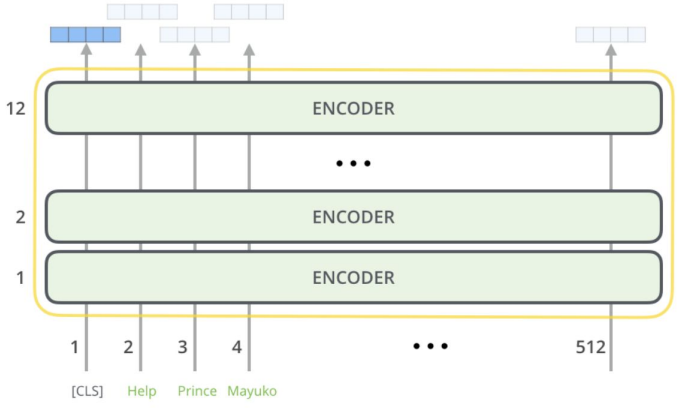
BERT

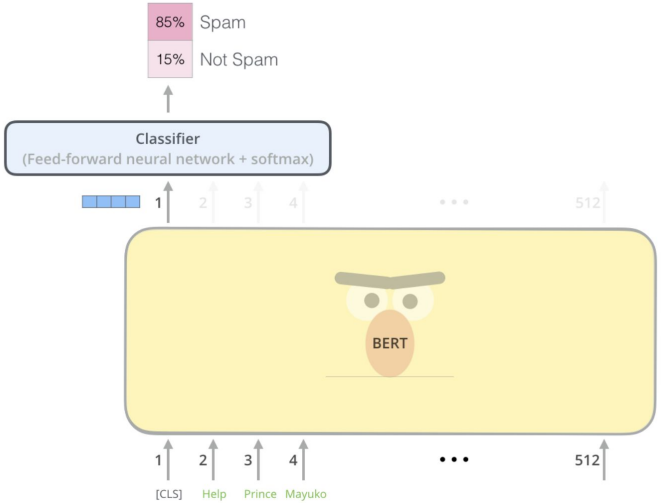
- BERT can complete several different kinds of tasks
  - Classification
  - Sentiment analysis
  - Question answering
  - Contextualized word embeddings
  - Etc.
- Uses only encoder transformer blocks



BERT

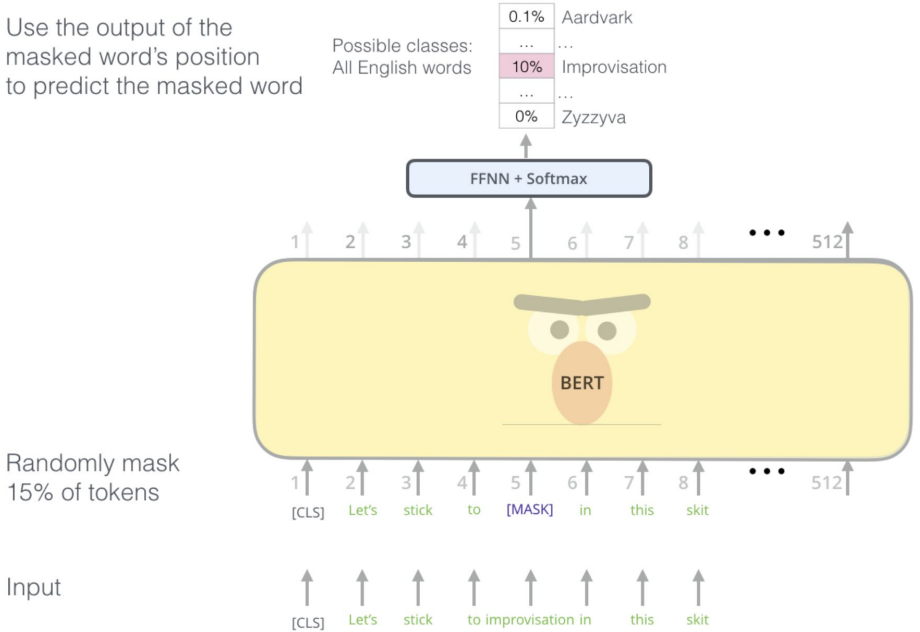
- The architecture is very similar to a regular encoder transformer
  - Sequences are put into encoder blocks that use self-attention
  - The outputs differ
    - Are vectors that can be used for as an input for a classifier





BERT

- The OpenAI Transformer gave us a fine-tunable pre-trained model based on the Transformer
  - Only trains a forward language model
- ELMo's model was bi-directional
- BERT combined the two using **masks**
  - Masks temporarily hide certain words in a sequence
  - By being masked, words wouldn't be able to "see themselves" when training



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