Word Vectors - First stage for language processors

- For each word, you have a giant vector of fixed size dimension for it
    - You precompute the embedding tables, or you use word-2-vec or gl0ve
- For each word in the sequence, you get the vector for it
- **Limitation of Word Vectors**
    - As time goes on words can change meaning
    - Words can change meaning depending on the context of the words around them

ELMo - substitute for word vectors

- Uses **2 LSTMs**
    - One goes left, one goes right
- Takes the input sequence one by one
- At each step, it generates hidden states that are results of the current input token and previous states
- The hidden states are now the embeddings for the tokens
- The word vectors are no longer one vector per word, you need the whole sequence to generate embeddings
- Each word has a unique (but maybe similar) embeddings depending on context
- You concatenate the two embeddings from the two LSTMs
- By combining them you get one final word vector for each token
- **Limitation of ELMo -**
    - You have information from the left and the right but it's very shallow because it's just a concatenation of two LSTMs that don't have anything to do with each other
    - You basically have two half-blind models that are looking in separate directions and concatenating output

OpenAI GPT - Is a **left-to-right transformer**

- It goes step-by-step but uses attention at each stepEach
- This means that each attention intermediate steps can attend to the left of it, i.e. what came before it
- This means we can enter the sequence as it is generated
- **Limitation -** The context that we have is only to the left of the token, kind of like reading text, it tries to make sense of the text as it goes
- It only uses the **decoder** part of a transformer
- The reason for only looking left is the kind of problems GPT is designed for
- **Language Modelling -** a task where you are given *a, b, c, d* and are asked to **predict** what comes next
    - By definition of the task, you can only look left
    - **Language Prediction** for generating language (can't do this with BERT)

BERT - A language model that generates text by predicting what comes next

- Can be trained unsupervised, only needs text, doesn't need any labels
- **Bidirectional -**
- Only an **encoder**
- BERT claims to be looking at each layer of the model, at each token, at all of the contexts of every single input
- Very similar to GPT, but it looks in both directions
- **Pretty good at all NLP problems except language prediction**
- Pre-trained on a large corpus

Conditional Language Model - a model that predicts the a word *based on the sequence generated so far*

Tasks BERT can be used for:

- **Masked Language Modelling -** replace some words by a **mask token** and then try to predict it
    - Make use of words from the left and from the right
    - Replace some words with a token, have two sentences and ask if they flow on from one to another
    - This is pre-training that can be done **unsupervised on a large corpus**

How BERT Works

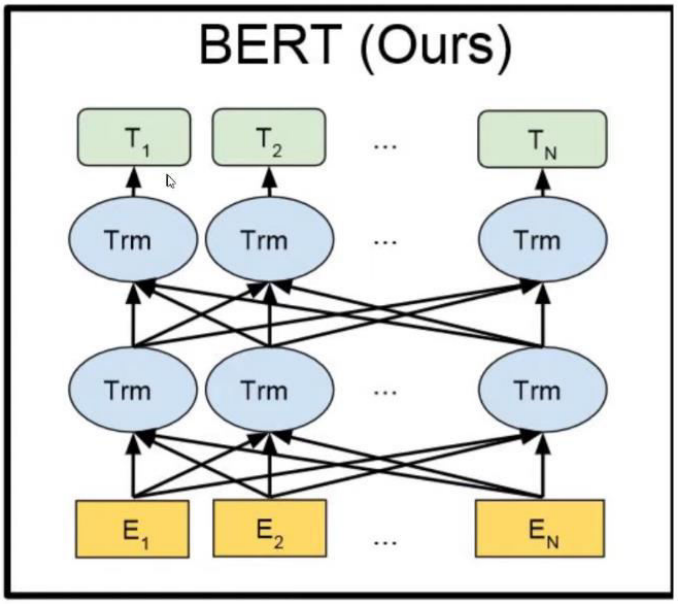
- You have **CLS**, **SEP**, and **##** tags
  - **CLS -** start
  - **SEP -** separate sentences
  - **##ing -** used to represent two **word pieces** (so you can do things like split postfixes)
    - It means you don't need to have all different modifications of a word a separate entry in table
  - *(alternative)* **OOV -** out of vocabulary tokens
  - **Character Level -** Represent your data per character
    - A character by itself doesn't really have any meaning
  - **Word Level -** have a table of all words and include **OOV** (undesirable)
    - Hard to represent every word, i.e. names
  - **Word Piece Level -** have an embedding for **all characters**, **most common words**, **common prefixes**
- **Token Embedding -** get word vectors for each token to get into vector spaces, i.e. with precomputed embeddings
- **Segment Embedding -** two possible tags $\mathbf{E_A}$ for first sentence and $\mathbf{E_B}$ for second sentence
- **Positional Embedding -** The model doesn't go step by step, so the model finds it hard to know how far two tokens are from each other so the positional embeddings help the model decide how close two tokens are to each other
- The input into the network is then created by concatenating the three embeddings for your inputs
- The model, after pre-training, is really good at predicting masked words
- We can then **fine-tune** the model for **11 different tasks** (SOTA at the time)
- **Classification**
  - label to text
  - **Entailment classification -** given a pair of sentences, predict whether second sentence is an **entailment**, **contradiction**, or **neutral** w.r.t the first one
  - Train by getting output token for the **CLS** token and use **logistic regression** to get output class probabilities
    - You only have to learn the **weights for the logistic regression**
  - **SQUAD - question>answer -** given a question and a paragraph, mark the start and end of an answer
    - You enter question and paragraph separated by a **SEP** token
    - For each output token for the second paragraph, you **classify each token** as a start/end/span token
    - You get a distribution for each token the probability that it is a certain type of a token
  - **Named Entity Recognition -** you have a sentence and you have to identify named entities
    - Similar to squad
    - For each output, you classify whether it's part of a named entity or not
      - You can have different types of entities
- **Reason for performance**
  - They are very sure it's because they get the **left AND right context** of a given token
  - If you don't do the **next sentence prediction** task the performance drops a lot
  - If you additionally only do **left-to-right training**, the performance drops

How to use BERT

- Pre-train **semi-supervised(/unsupervised)** with masked words
- Pre-train **semi-supervised(/unsupervised)** with sentence entailment
- Fine-tune **supervised** on a specific task with a labelled dataset
  - 11 different tasks, with different ways of fine tuning

Architecture

- BERT, opposite to GPT, only uses **encoder blocks**
  - **Encoder blocks** don't have a mapping effect and can be bidirectional, they can look in the past and ahead



BERT Variations

- **DistilBERT -** downscaled version of original BERT
    - Comparable performance to BERT but signitifantly smaller