BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

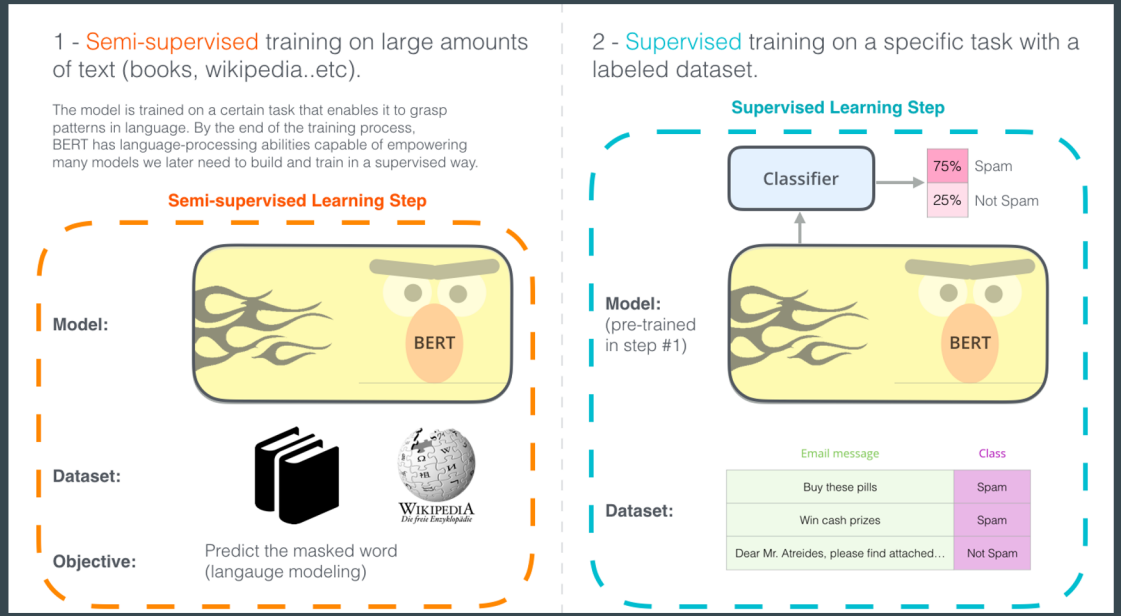
...

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova

arXiv:1810.04805

What is BERT?

- Bidirectional Encoder Representations from Transformers
- A contextualized language model that can be fine-tuned for a variety of tasks



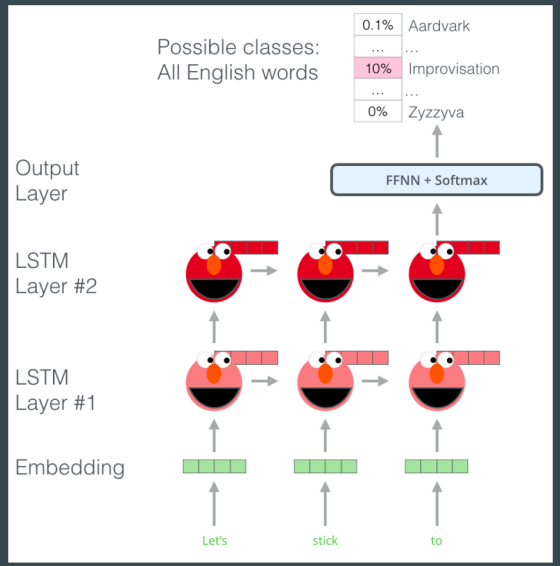
The two steps of how BERT is developed. You can download the model pre-trained in step 1 (trained on un-annotated data), and only worry about fine-tuning it for step 2.

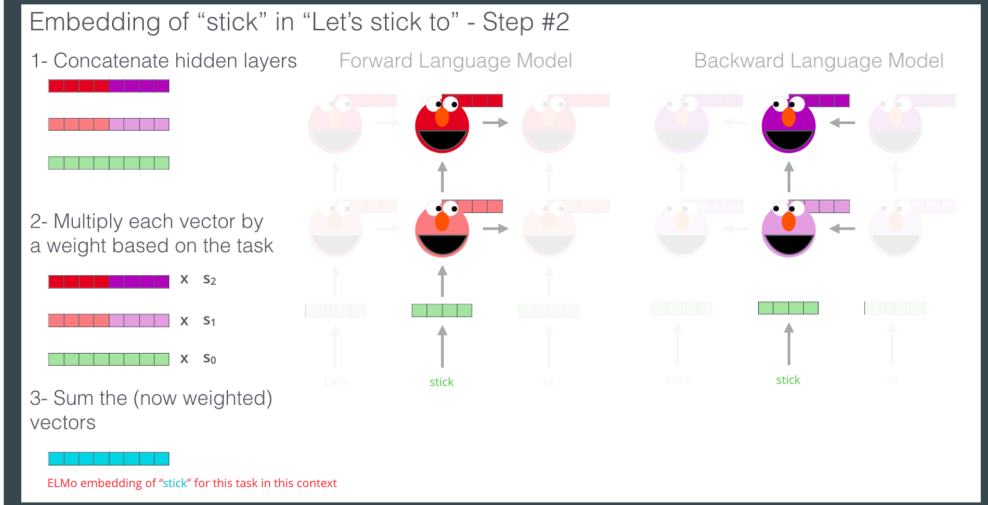
BERT

- **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
- Bidirectional - Uses attention
  - Attention considers content before and after a word rather than sequentially like an RNN
- Encoder Representations
  - Condenses text into a vector that represents its meaning and components (features)
- Transformers
  - Layered neural networks which use attention to encode the representations

Previous Models: ELMo

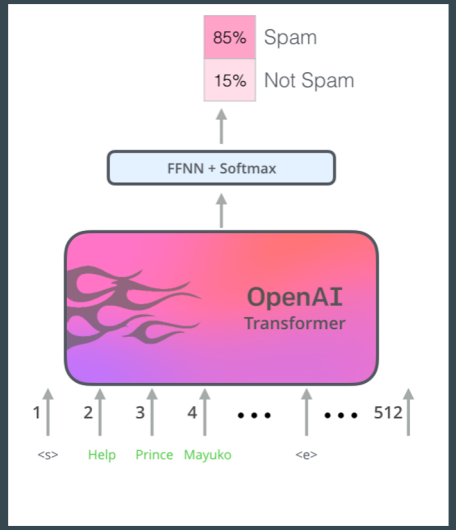
- Creates embeddings based on the context it's used in (contextualized word-embeddings)
- It is a pair of LSTMs trained to take input vectors and predict the next word in a sequence
- ELMo looks at the entire sentence before assigning each word an embedding
- These vectors are then used as features in a downstream task

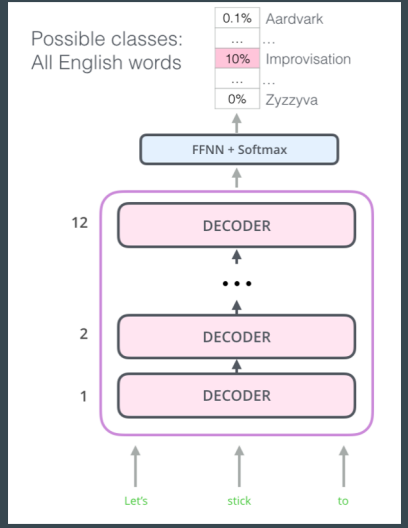


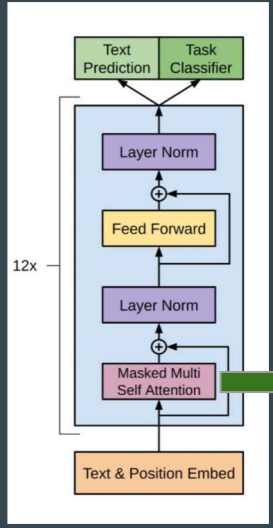


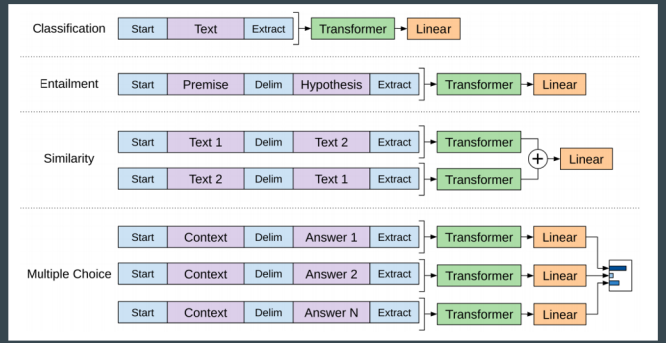
Previous Models: OpenAI Transformer

- Often sentences will rely on information significantly distant from each other
- An RNN may fail to capture this so instead an attention based system is used
- The OpenAI Transformer is a stack of Decoders which are trained to predict the next word in a sequence
  - Decoders sequential data and mask all future information past the target token
- Transfer learning is done by formatting different problems as if they were sequential and adding a final classifier layer



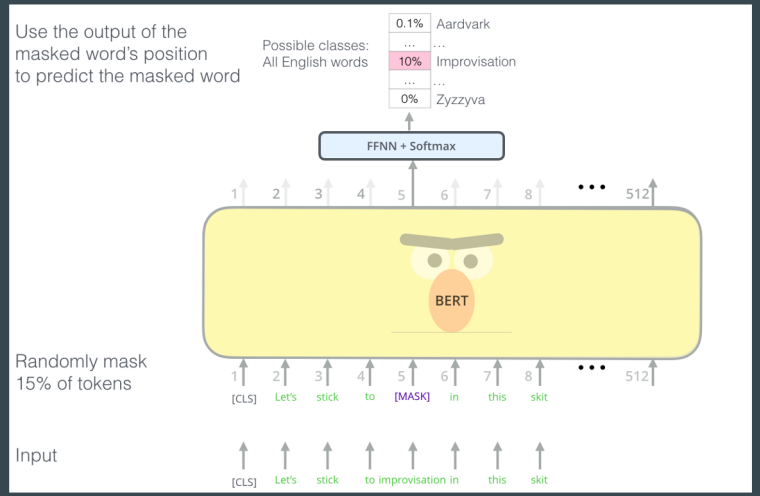


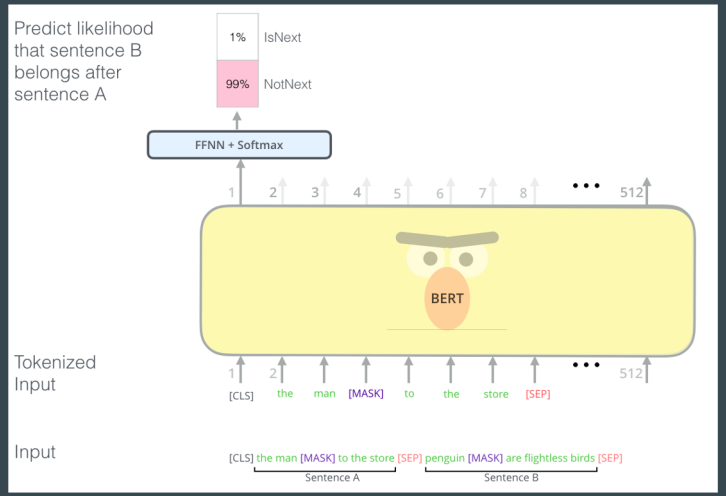




BERT: The Next Step

- OpenAI, by using decoders, goes back to only considering information in 1-direction
- BERT is instead a stack of encoders that looks at attention in both directions simultaneously
  - To prevent the model from automatically recognizing the target word, it selectively masks (or replaces) 15% of the tokens and is trained to predict the missing term
- BERT is also trained on predicting likelihood sentence B follows sentence A
  - This likelihood is based on the output vector for a given class label which is appended to the sequence





Difference Between Architectures

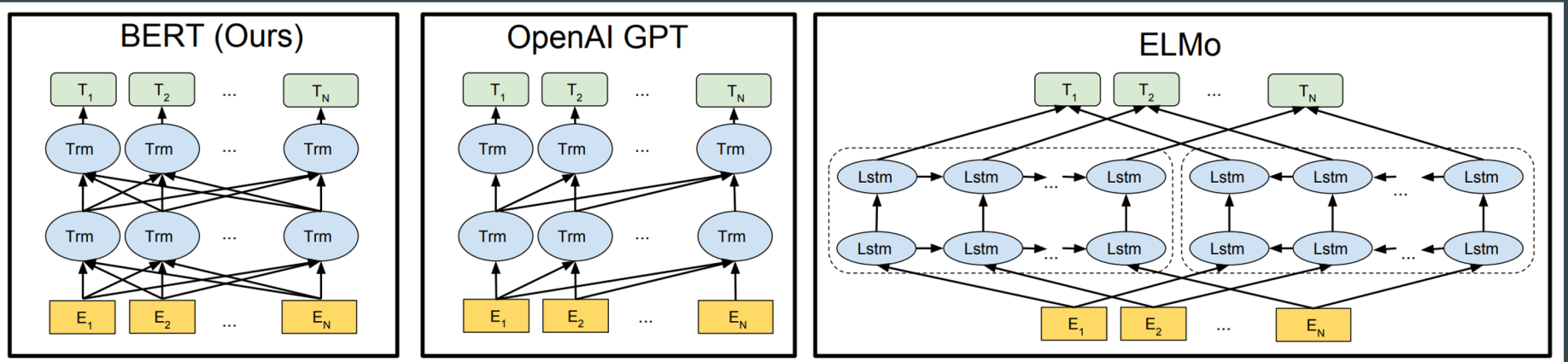
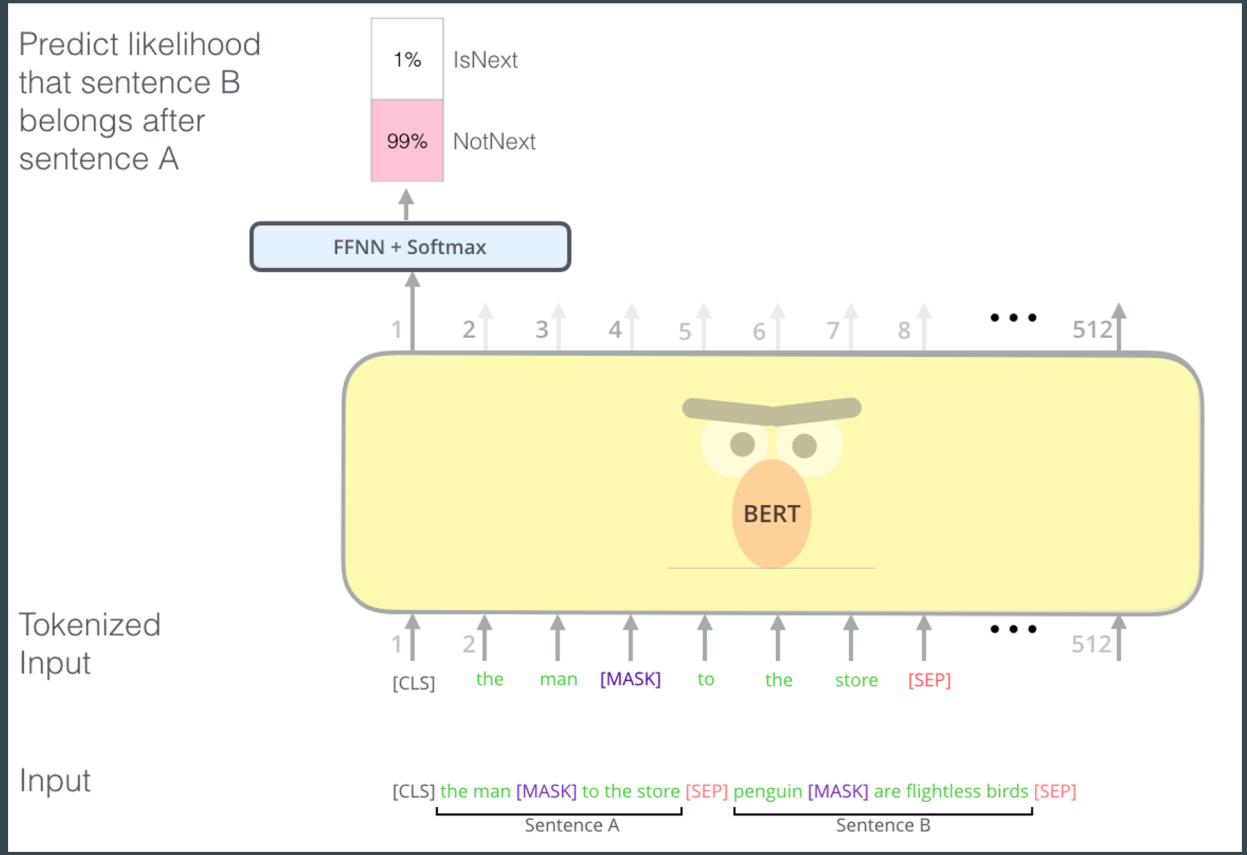


Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT 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.

BERT: Input Representation



BERT: Input Representation

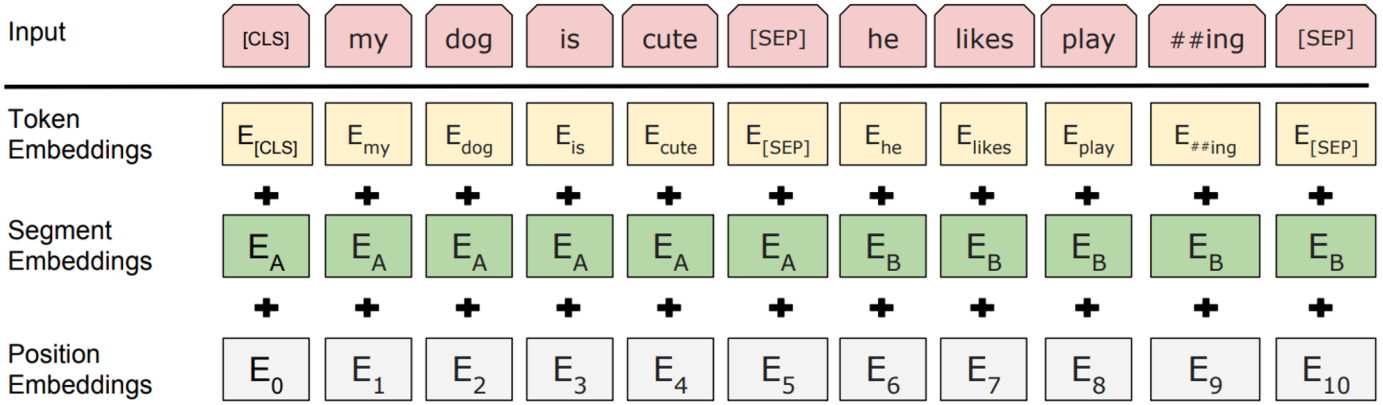


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Positional Embedding

- When using attention, word order is not retained unlike with sequential NN
- This can lead to ambiguity when comparing tokens within a sentence
  - I do not like the story of the movie, but I do like the cast
  - I do like the story of the movie, but I do not like the cast
  - Same words but different order completely changes sentiment

- BERT encodes positional embeddings into tokens
  - This encoding uses the same formula as presented in Attention is All You Need
  - pos = position
  - i = dimension
  - d_{model} = dimension size of model

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Subword Embeddings

- In the pretraining step, Bert uses WordPiece tokenization
  - This encodes sub-word information into the language model so that in testing/usage an Out-Of-Vocabulary (OOV) token can still have an embedding

The following algorithm is how WordPiece works:

1. Create a corpus
2. Define a desired subword vocabulary size
3. Split word to sequence of characters
4. Build a language model based on step 3 data
5. Choose the new word unit out of all the possible ones that increases the likelihood on the training data the most when added to the model.
6. Repeating step 5 until reaching subword vocabulary size which is defined in step 2 or the likelihood increase falls below a certain threshold.

Imagine a corpus of ‘walker’, ‘walks’, ‘walked’

1. w|a = wa
2. l|k = lk
3. wa|lk = walk

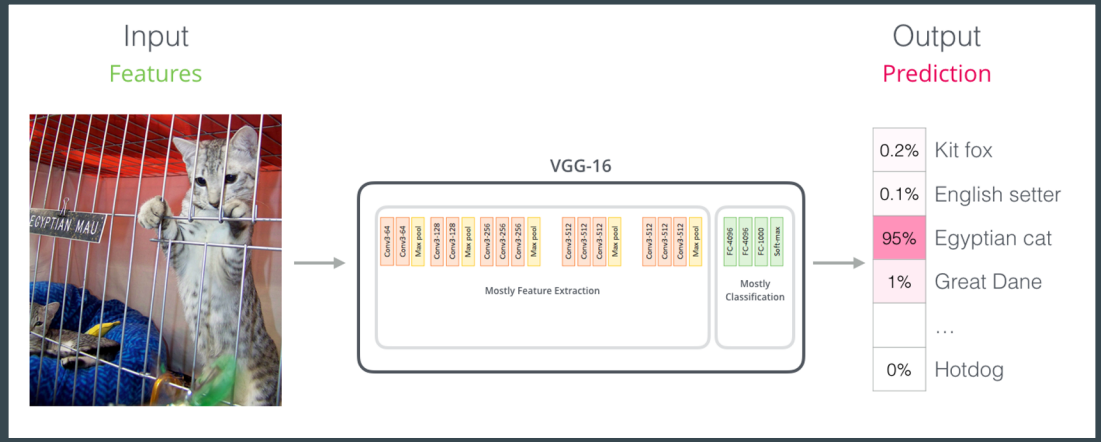
These subwords are selected since they are common in the corpus and thus likely. Now each component, including letters, subwords, and entire words, are included in the dictionary

Given OOV “walking” the model can use ‘walk’ and ‘ing’ (which it learns from other words)

BERT: Transfer Learning

- In a neural network, each layer gathers different information
  - Earlier layers extract more general features while later layers get more specific/nuanced features

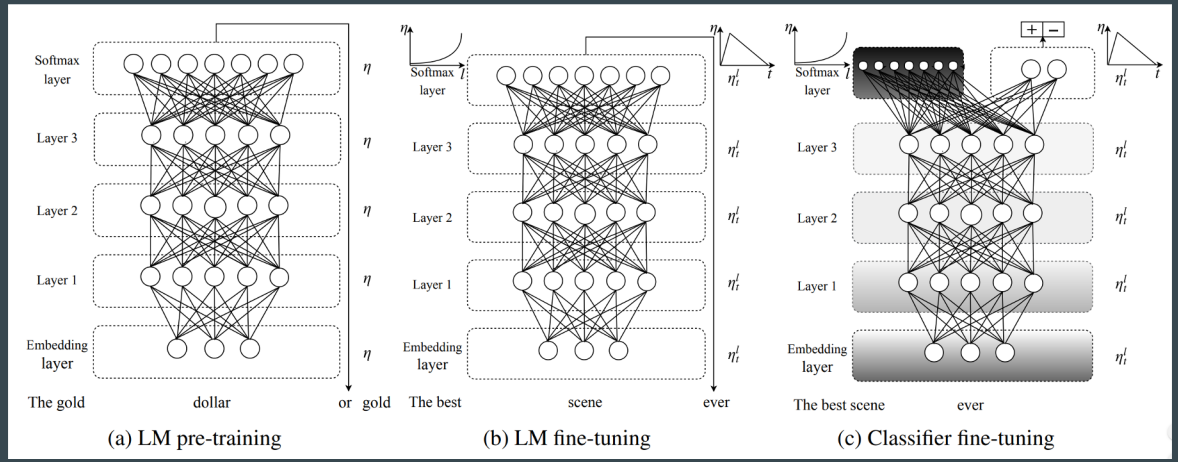
- Transfer learning reduces the need for in-domain data
  - Pre-train a model on a large, general dataset and then fine-tune it for a task using a smaller, specific dataset
  - Pre-training makes early layers good at extracting general features
  - Attach new task-specific classification layers to this model, fine-tuning them for a specific task
  - All previous parameters, which were initialized in pre-training, are then further trained at a fine-tuning learning rate
  - This is common in computer vision



Fine-Tuning : ULM-FiT

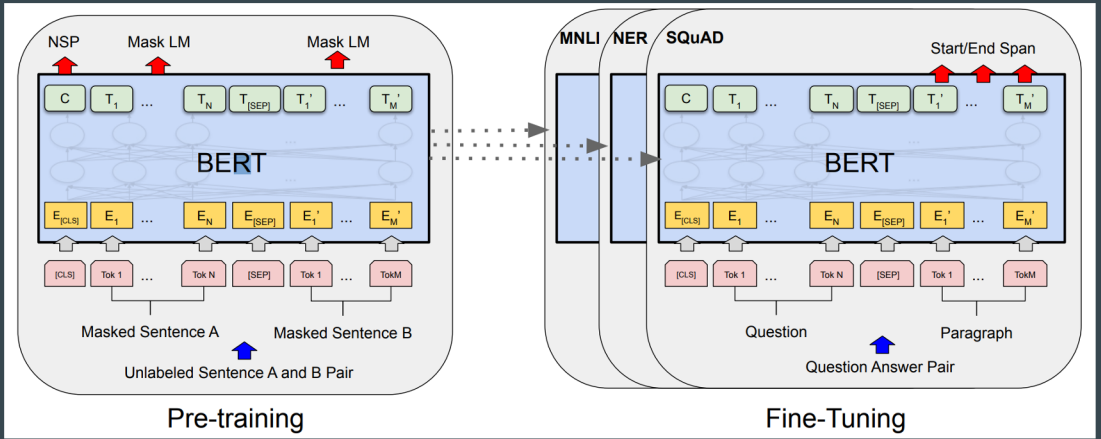
- ULM-FiT introduced convention for transfer learning in language modeling

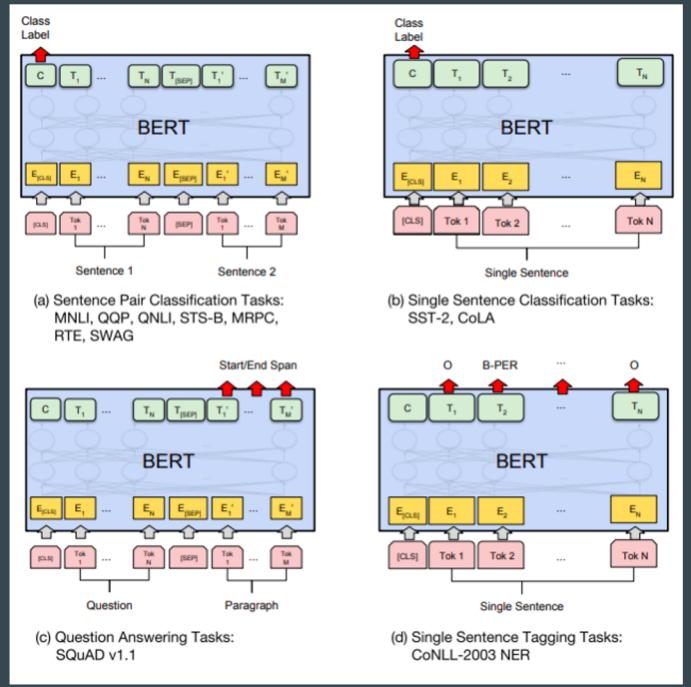
A. The model is trained over a large data set
B. Each layer of the model is fine tuned using parameters specific to that layer since each layer captures different information
    a. The triangular learning rate is more aggressive, allowing for less data to make a large impact
C. Fine-tunes model for classification task
    a. Concat Pooling: Concatenate as many layers as fit in memory to to retain detailed information
    b. Gradual Unfreezing: Fine tune a layer at a time to prevent ‘catastrophic forgetting’



BERT: Fine-Tuned for Different Tasks

- Using fine tuning, BERT can be geared towards a variety of tasks
- For a given task, tokens are appended into the sequence
  - The hidden state of this token is output for classification tasks





Evaluation: GLUE Tasks

- General Language Understanding Evaluation benchmark
- Collection of diverse language understanding tasks
  - MNLI: Multi-Genre Language Interference
    - Entailment classification (Determine if 2 sentence are entailment/contradiction/neutral
  - QQP: Quora Question Pairs
    - Determine if 2 questions asked on quora are semantically equivalent
  - QNLI: Question Natural Language Interference
    - Stanford Question Answering Dataset converted to classification task
  - SST-2: Stanford Sentiment Treebank
    - Single-sentence classification of movie review sentiment
  - CoLA: Corpus of Linguistic Acceptability
    - Single sentence classification to see if a sentence is linguistically english
  - STS-B: Semantic Textual Similarity Benchmark
    - Determine how similar 2 sentences are by semantic meaning
  - MRPC: Microsoft Research Paraphrase Corpus
    - Sentence pairs gathered from the news to determine if they are semantically equivalent
  - RTE: Recognizing Textual Entailment
    - Similar to MNLI but less training data
  - WNLI: Winograd NLI
    - Small natural language inference dataset

Evaluation: Glue Results

- Introduced softmax classification layer
- Batch size of 32 and 3 epochs
- Selected best fine-tuning learning rate on Dev set
- $\text{BERT}_{\text{Large}}$ was unstable on some small datasets
  - Performed random restarts that did different fine-tuning data shuffling and classifier layer initialization

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	79.8	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**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard).

Evaluation: SQuaD (1.1 & 2.0)

- Stanford Question Answering Dataset
  - Collection of 100k crowdsourced Q:A pairs
  - Given a question and passage from Wikipedia, the task is to predict the answer substring (text span) from the passage
- Introduce a Start [S] and End [E] vector in fine tuning
  - Gather probability of a given word T_i being the start or end token for the span by calculating a dot product followed by a softmax over all words in the paragraph

$$P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}$$

$$S \cdot T_i + E \cdot T_j$$

  - Score of span is calculated by the above formula where j>= i
  - Training objective is sum of log-likelihoods of the correct start and end positions
- Squad v2.0 introduces the chance that the text is not in the passage
  - A null answer has the start and end token at the [CLS] token
  - Compare the probability of null answer vs best non-null answer

System	Dev EM	Dev F1	Test EM	Test F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Table 2: SQuAD 1.1 results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

System	Dev EM	Dev F1	Test EM	Test F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	86.3	89.0	86.9	89.5
#1 Single - MIR-MRC (F-Net)	-	-	74.8	78.0
#2 Single - nlnet	-	-	74.2	77.1
Published				
unet (Ensemble)	-	-	71.4	74.9
SLQA+ (Single)	-		71.4	74.4
Ours				
$\text{BERT}_{\text{LARGE}}$ (Single)	78.7	81.9	80.0	83.1

Table 3: SQuAD 2.0 results. We exclude entries that use BERT as one of their components.

Evaluation: SWAG

- Situations With Adversarial Generations dataset
  - 113k sentence-pair completion examples
  - Given a sentence, the task is to choose the most plausible continuation among four choices

- In fine-tuning, they concatenated the given sentence with each of the 4 sentences and labelled accordingly

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
OpenAI GPT	-	78.0
$\text{BERT}_{\text{BASE}}$	81.6	-
$\text{BERT}_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

Table 4: SWAG Dev and Test accuracies.

Ablation Studies

Tasks	Dev Set MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
$\text{BERT}_{\text{BASE}}$	84.4	88.4	86.7	92.7	88.5
No NSP	83.9	84.9	86.5	92.6	87.9
LTR & No NSP	82.1	84.3	77.5	92.1	77.8
+ BiLSTM	82.1	84.1	75.7	91.6	84.9

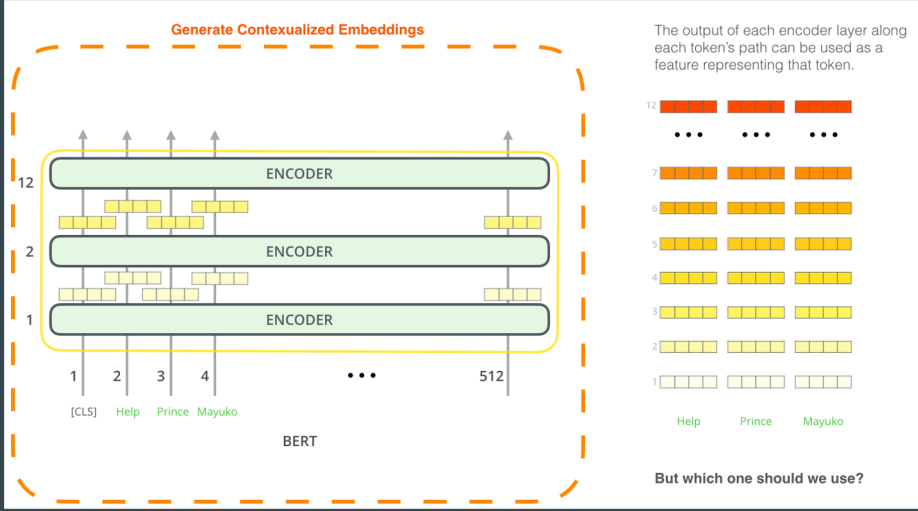
Table 5: Ablation over the pre-training tasks using the $\text{BERT}_{\text{BASE}}$ architecture. "No NSP" is trained without the next sentence prediction task. "LTR & No NSP" is trained as a left-to-right LM without the next sentence prediction, like OpenAI GPT. "+ BiLSTM" adds a randomly initialized BiLSTM on top of the "LTR + No NSP" model during fine-tuning.

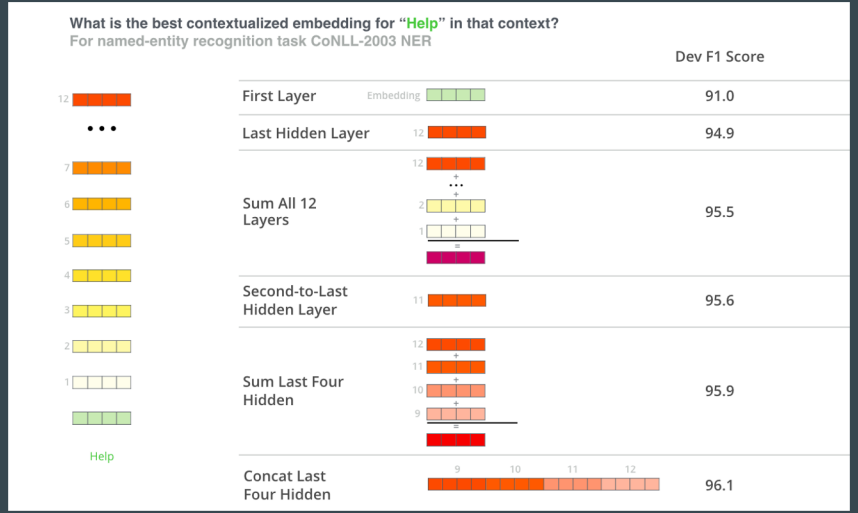
Hyperparams				Dev Set Accuracy		
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

Table 6: Ablation over BERT model size. #L = the number of layers; #H = hidden size; #A = number of attention heads. "LM (ppl)" is the masked LM perplexity of held-out training data.

BERT: Feature Extraction

- ELMo and Word2Vec both allowed for word vectors to be output and then used for a downstream task
- While fine tuning a model typically works better for an overall task, this feature extraction is useful for quick prototyping and for especially low resource tasks
- In BERT, we use a combination of hidden layers to create a vector representation
    - The best combination is dependent on the final task





Conclusion & Limitation

- BERT builds on several advancements in NLP in recent years

- BERT presents a new cutting edge for language models than can be quickly applied to a variety of tasks and domains
  - It especially opens up opportunities on working in low-resource tasks

- Base BERT has the limitation of only being able to handle 512 tokens at a time, which limits its ability to to work on data sets with larger sample components