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

- ARUN

Content

Transformer Model

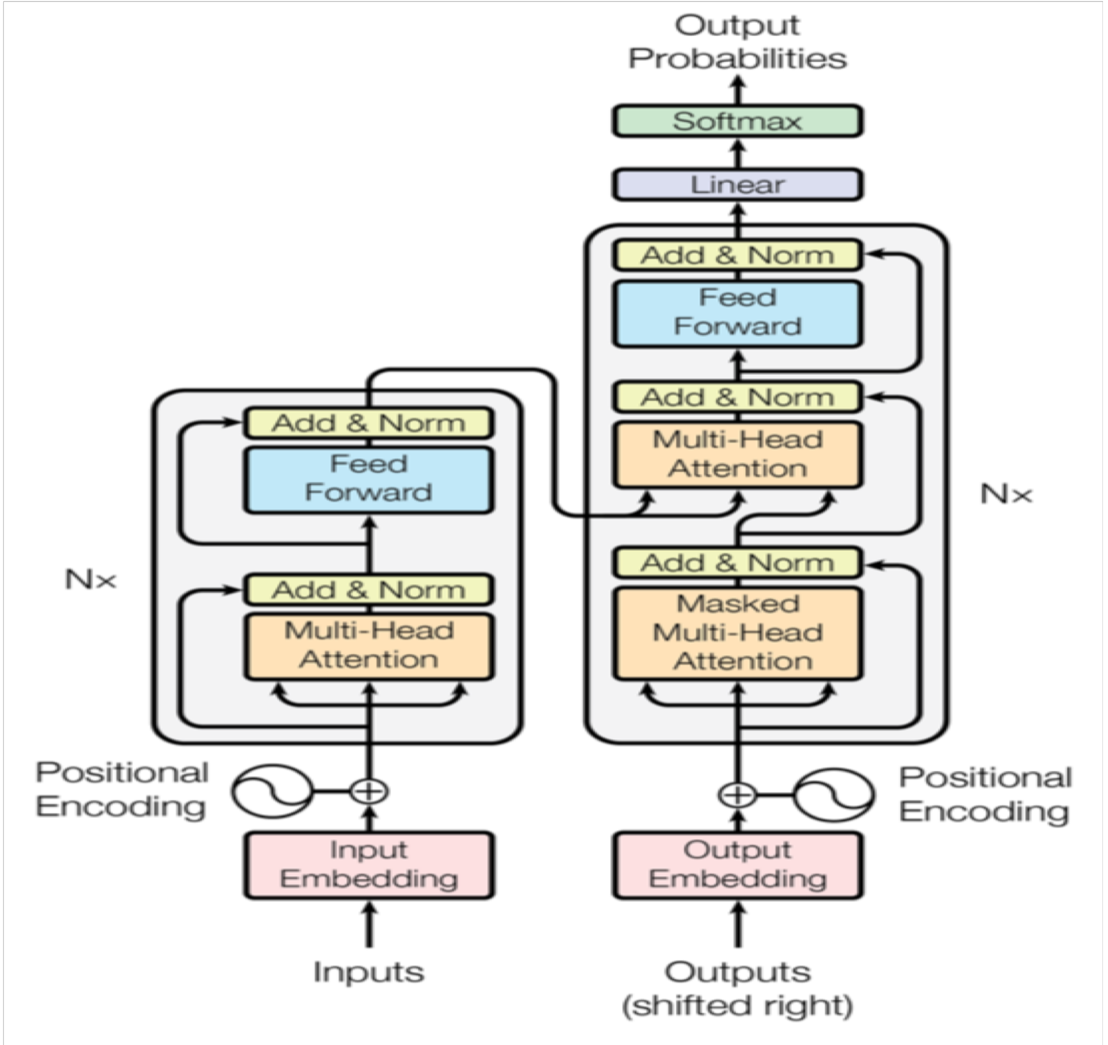


Figure 1: The Transformer - model architecture.

Attention

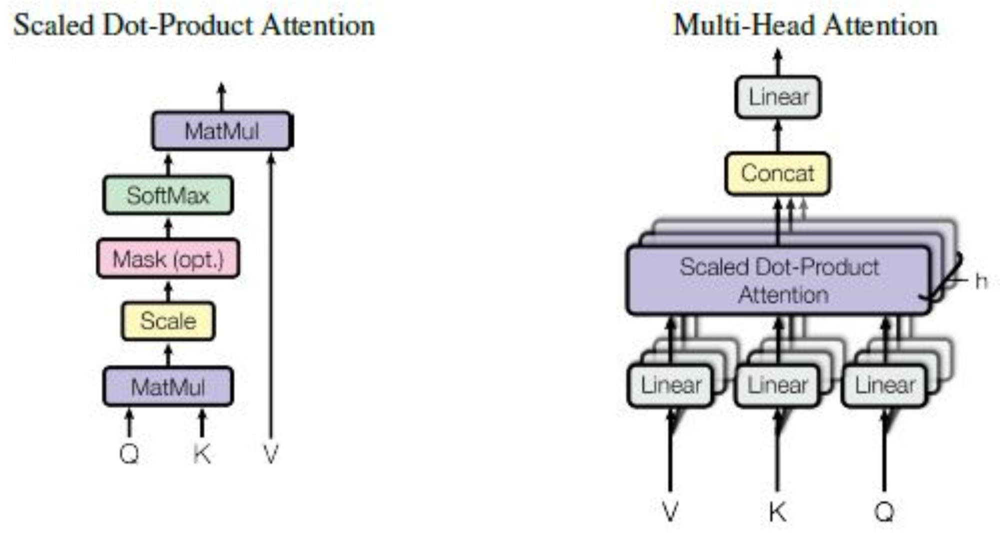


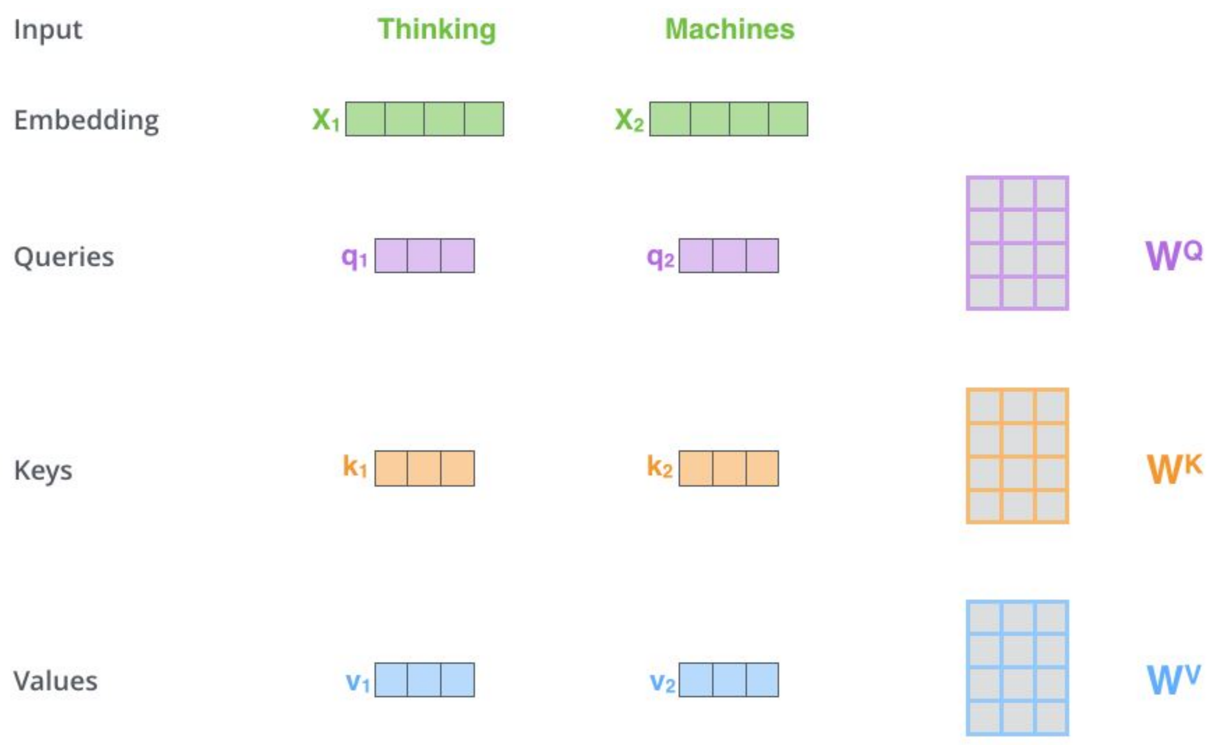
Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Types of Attention

- Additive Attention vs Dot product attention
- Scaled Dot product attention
  - Faster and space efficient (optimized matrix multiplication code)
- Scaling to increase performance for large values

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

Key,Value,Query



lultiplying 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.

Multi head attention

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

- dmodel = 512 ; dk = dv = dmodel/h
- Jointly attend to information from different representation subspaces at different positions
- Reduced dimension of each head, the total computational cost is similar to single-head attention with full dimensionality

Applications

1) Encoder – Decoder Attention

=> Key, Value from Input & Query from Output

2) Encoder - Self attention

=> Key,Value,Query all from Input

3) Decoder - Masked Self attention

=> Key, Value, Query all from Output

=> Prevent Leftward information flow , autoregressive property

=> Masking out all values in the input of the Softmax which correspond to

illegal connections

Positional Encoding

- sine and cosine functions of different frequencies
- Can also use learned positional embeddings
- sinusoidal over learned positional embeddings as it allows model to extrapolate

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

Model

- Embeddings + Positional Encoding
- Encoder

  - Multi head attention + Fully Positionwise FFN
  - Implements residual connections with batch normalization
- Decoder

  - Masked Multi Head Attention : to prevent positions from attending to subsequent positions
  - Encoder Decoder Multi Head Attention + Fully Positionwise FFN
- Two Linear Layers & Softmax

Model Comparison

- ELMo (Peters et al. NAACL 2018)
  - BiLSTM Pretrained using LM
  - Embeddings from Hidden Layers
- ULMFiT (Howard & Ruder ACL 2018)
  - LSTM Pretrained using LM & Fine Tuned on Specific Task
  - Add Classification layer for Predictions
- OpenAI GPT (Radford et al. 2018)
  - Transformer Decoder Pretrained using LM(Left to Right)
  - Add Classification layer for Predictions
- BERT (Devlin et al. 2018)
  - Transformer encoder Pretrained using MLM & NPS
  - Add Classification layer for Predictions

Related Architectures

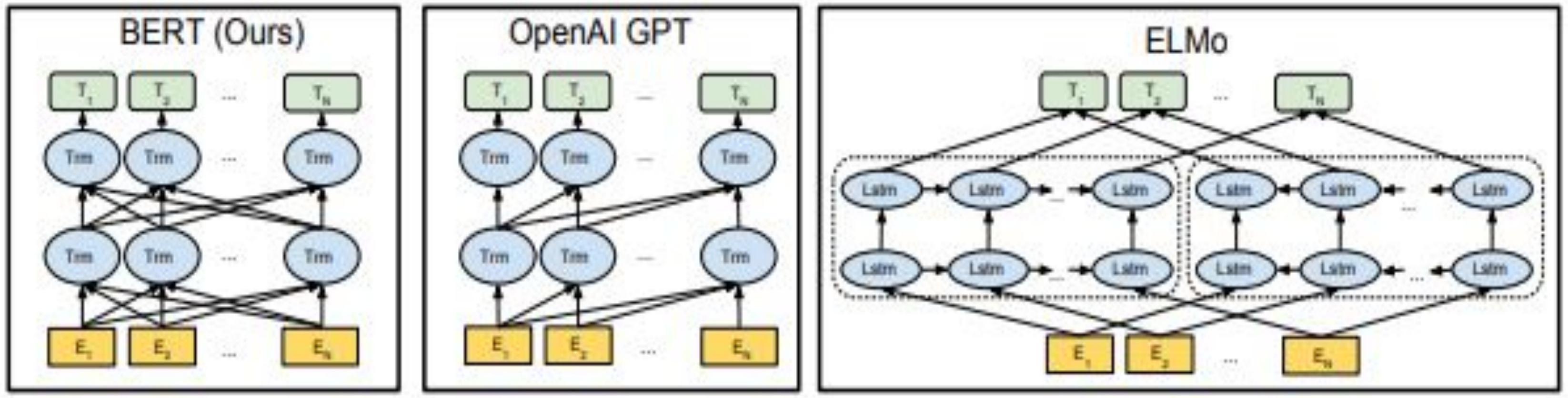


Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

Model Architecture

- Let No of Transformer blocks L, the hidden size as H, & no of self-attention heads as A
  - BERT BASE: L=12, H=768, A=12, Total Parameters=110M
  - BERT LARGE: L=24, H=1024, A=16, Total Parameters=340M
- WordPiece embeddings (Wu et al., 2016) with a 30,000 token vocabulary.
- Split word pieces with ##.
- Learned positional embeddings with max sequence length of 512 tokens.
- First token is special classification embedding ([CLS])
- Sentence pairs are packed together into 1 sequence with learned sentence embeddings

Input Representation

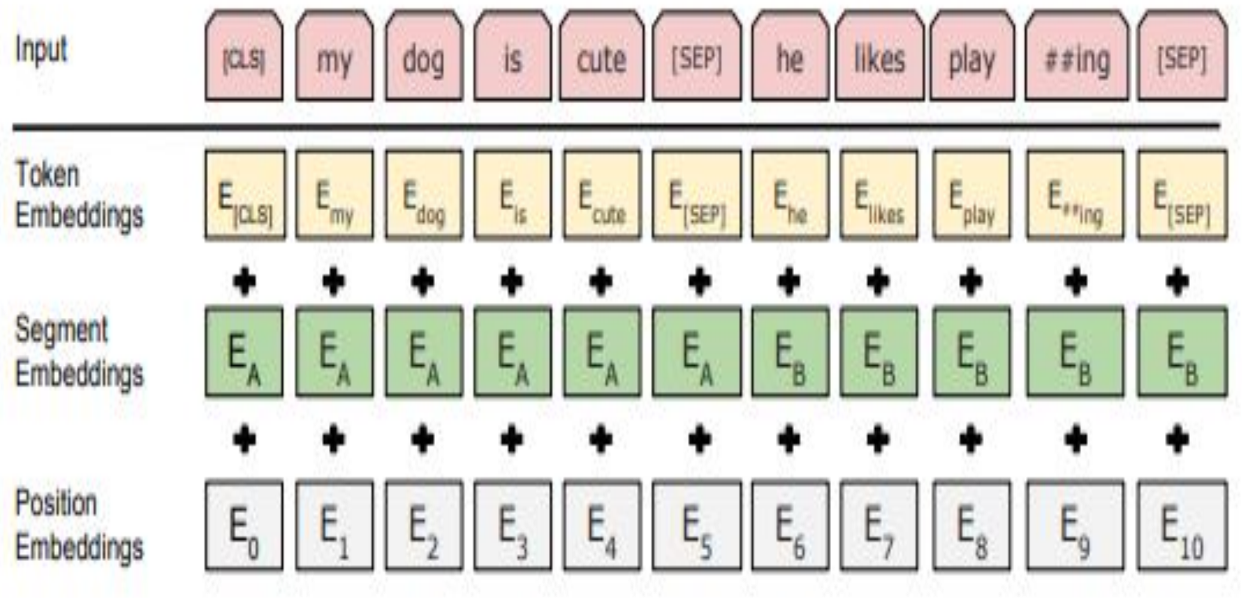
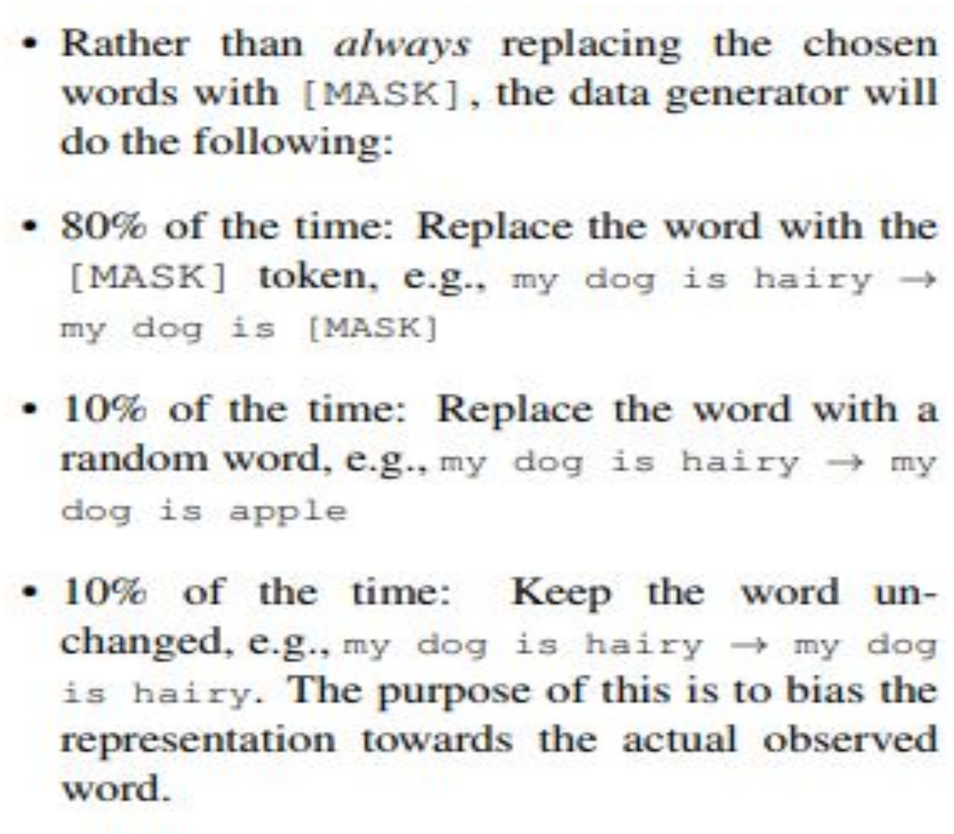

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

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

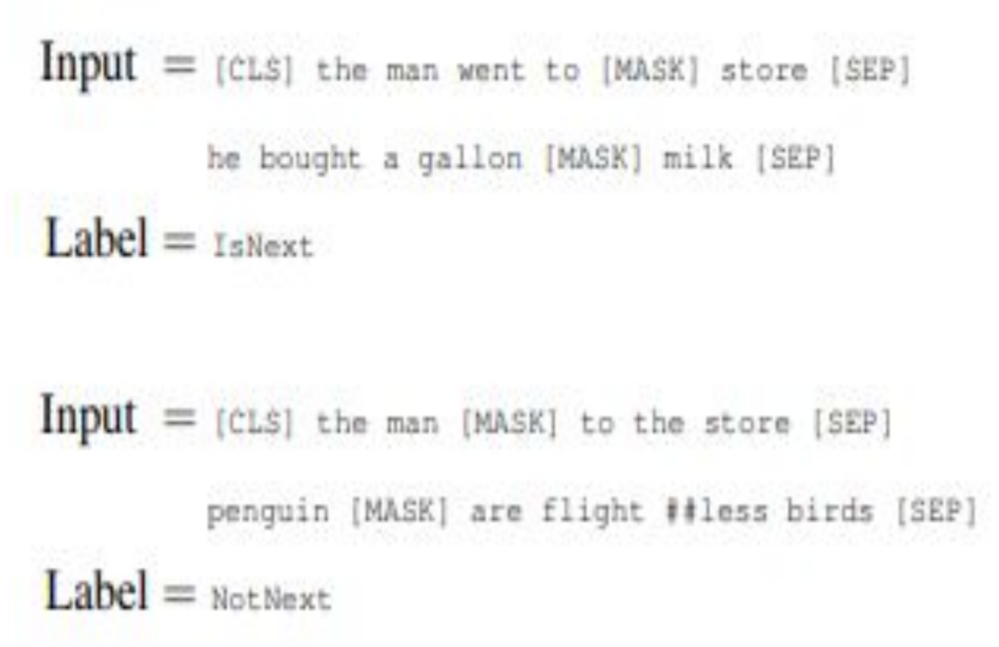
Masked Language Model



- Rather than *always* replacing the chosen words with `[MASK]`, the data generator will do the following:
- 80% of the time: Replace the word with the `[MASK]` token, e.g., `my dog is hairy` → `my dog is [MASK]`
- 10% of the time: Replace the word with a random word, e.g., `my dog is hairy` → `my dog is apple`
- 10% of the time: Keep the word unchanged, e.g., `my dog is hairy` → `my dog is hairy`. The purpose of this is to bias the representation towards the actual observed word.

- ➢ Why not <MASK> everywhere ?
  - ➢ Might not predict well during fine tuning
  - ➢ Might learn good contextual representation of only <MASK>
- ➢ Why leave some sentences intact?
  - ➢ Biasing to learn masked tokens better
  - ➢ Helps model learn representation for all the tokens
- ➢ Will random words confuse the model?
  - ➢ Yes. Hence, only small percentage
  - ➢ Didn’t affect model performance

Next Sentence Prediction



Input = [CLS] the man went to [MASK] store [SEP]

he bought a gallon [MASK] milk [SEP]

Label = IsNext

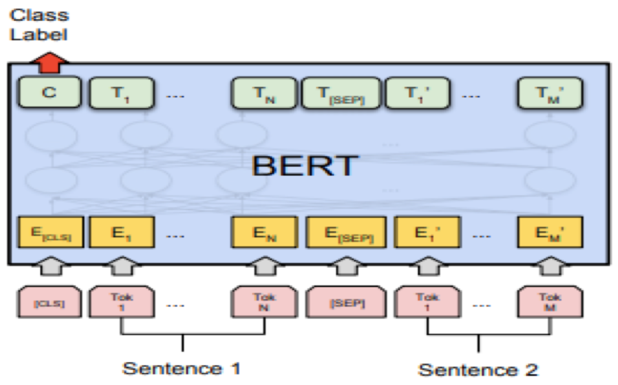
Input = [CLS] the man [MASK] to the store [SEP]

penguin [MASK] are flight ##less birds [SEP]

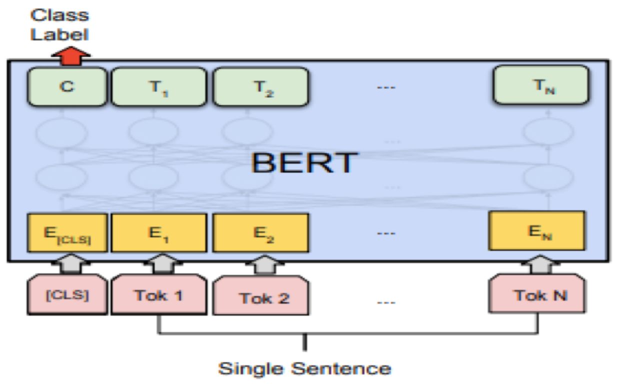
Label = NotNext

- Helps understanding the relationship between two text sentences
- 50% of time B is the actual next sentence that follows A, & other 50% random sentence from corpus
- 97%-98% accuracy at this task

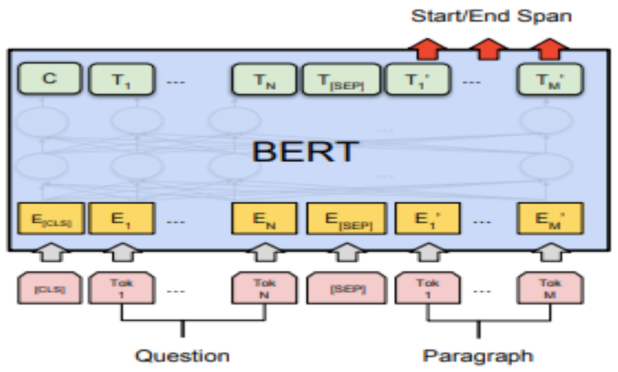
Fine Tuning



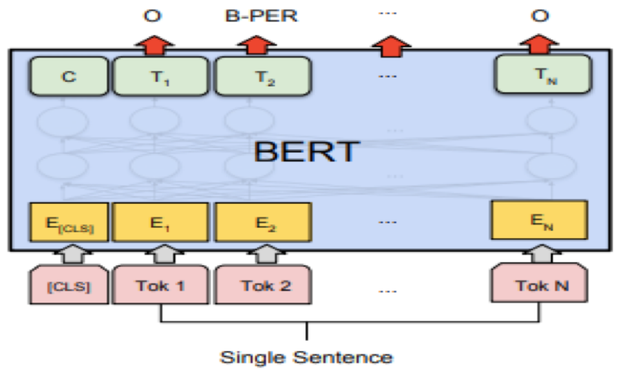
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



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



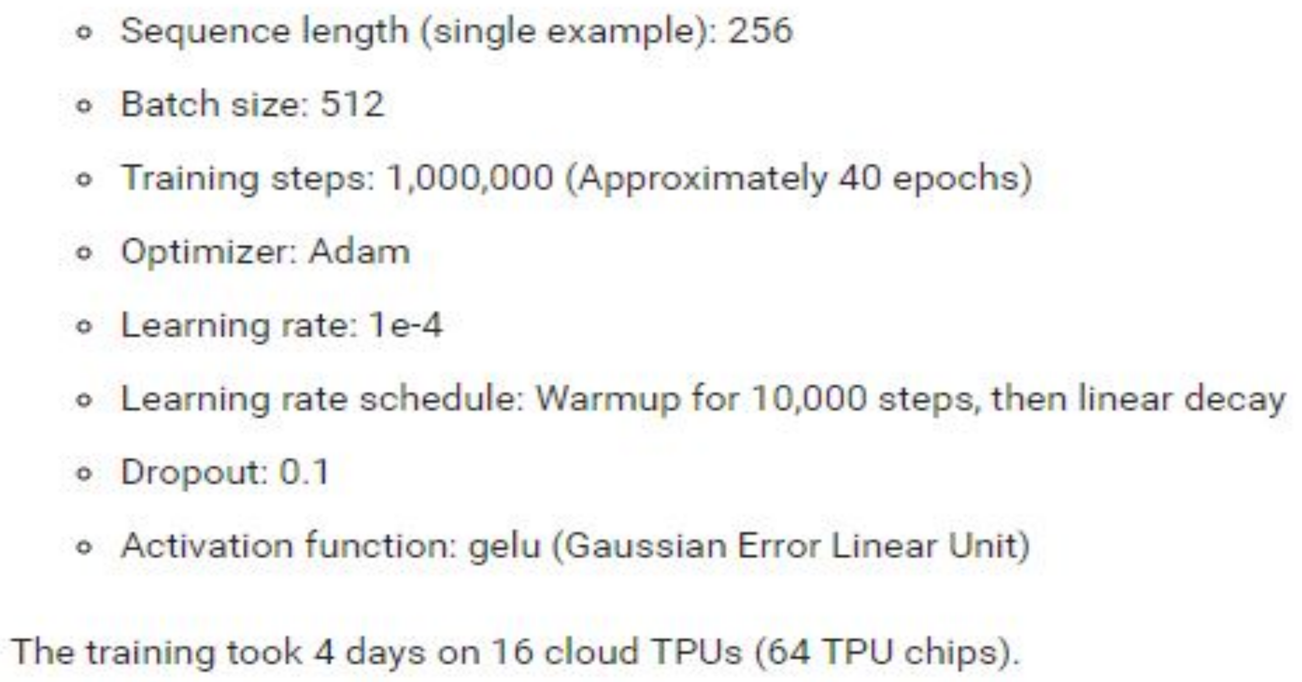
(c) Question Answering Tasks: SQuAD v1.1



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

Hyperparameters

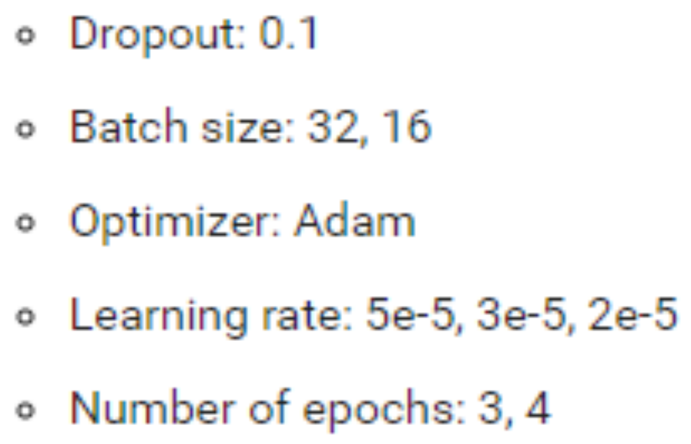
Pre Training



- Sequence length (single example): 256
- Batch size: 512
- Training steps: 1,000,000 (Approximately 40 epochs)
- Optimizer: Adam
- Learning rate: 1e-4
- Learning rate schedule: Warmup for 10,000 steps, then linear decay
- Dropout: 0.1
- Activation function: gelu (Gaussian Error Linear Unit)

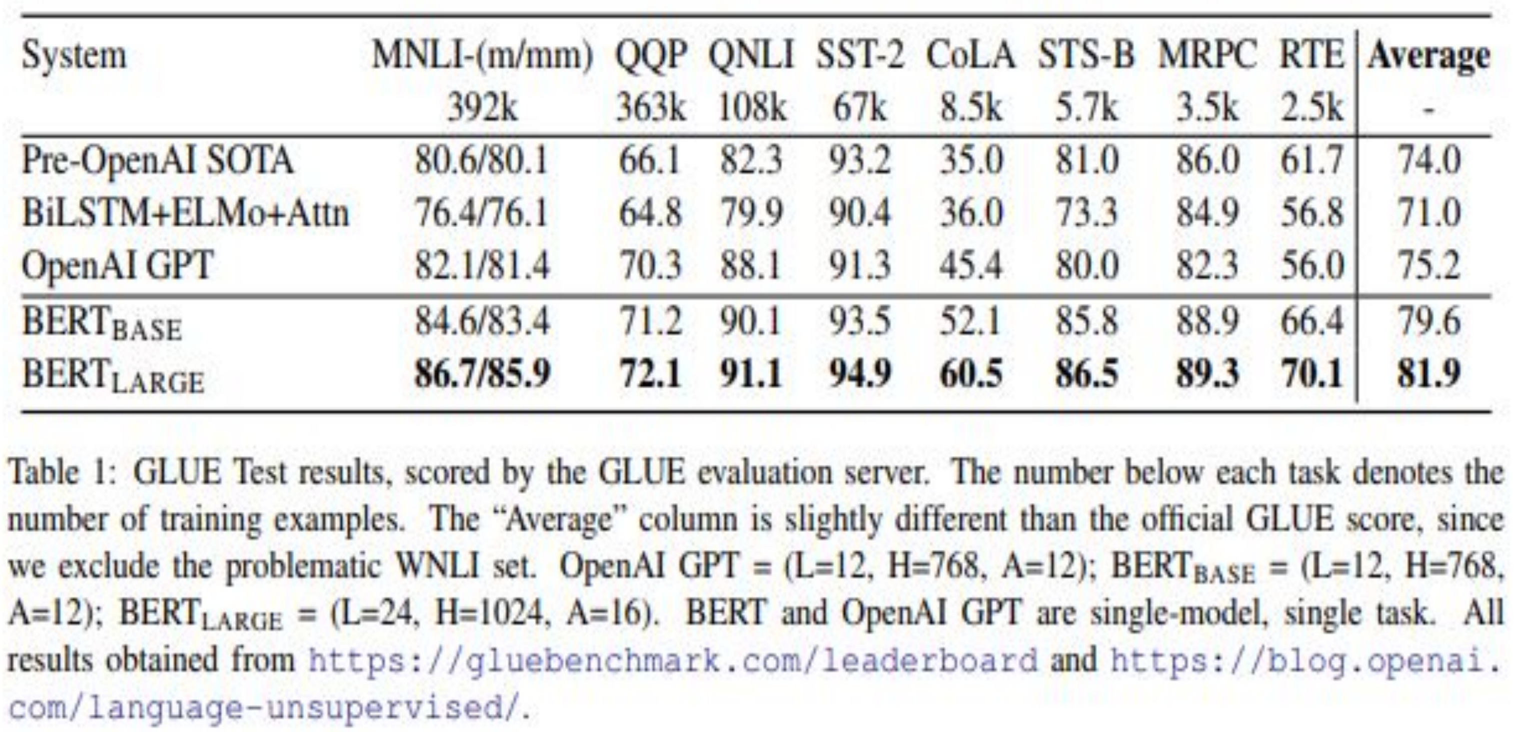
The training took 4 days on 16 cloud TPUs (64 TPU chips).

Fine Training



- Dropout: 0.1
- Batch size: 32, 16
- Optimizer: Adam
- Learning rate: 5e-5, 3e-5, 2e-5
- Number of epochs: 3, 4

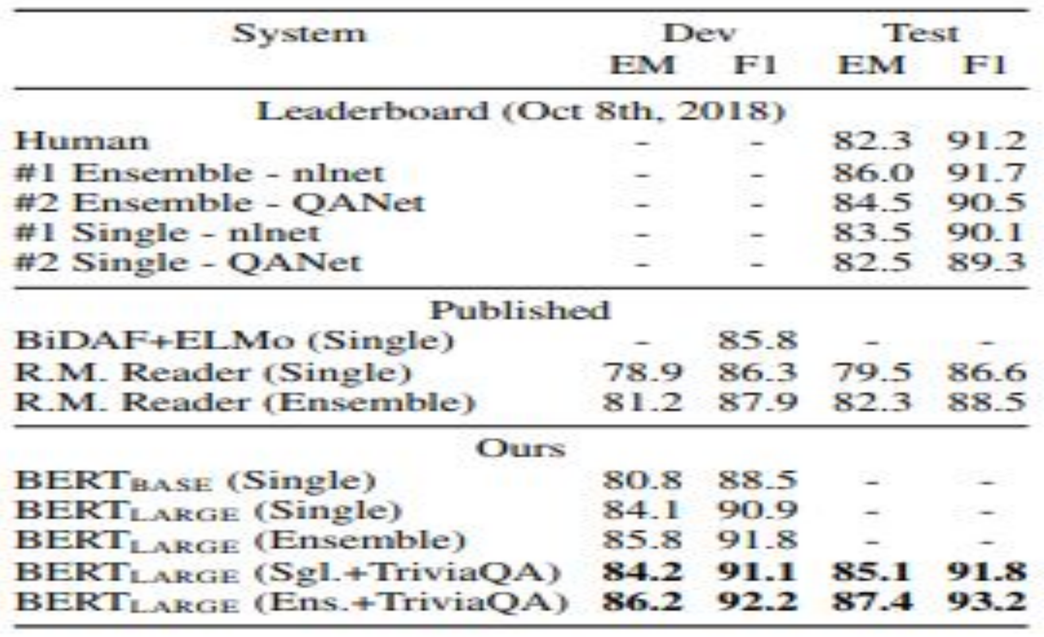
GLUE Datasets



System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
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
BERT$_{BASE}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT$_{LARGE}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

Table 1: GLUE Test results, scored by the GLUE evaluation server. The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. OpenAI GPT = (L=12, H=768, A=12); BERT$_{BASE}$ = (L=12, H=768, A=12); BERT$_{LARGE}$ = (L=24, H=1024, A=16). BERT and OpenAI GPT are single-model, single task. All results obtained from https://gluebenchmark.com/leaderboard and https://blog.openai.com/language-unsupervised/.

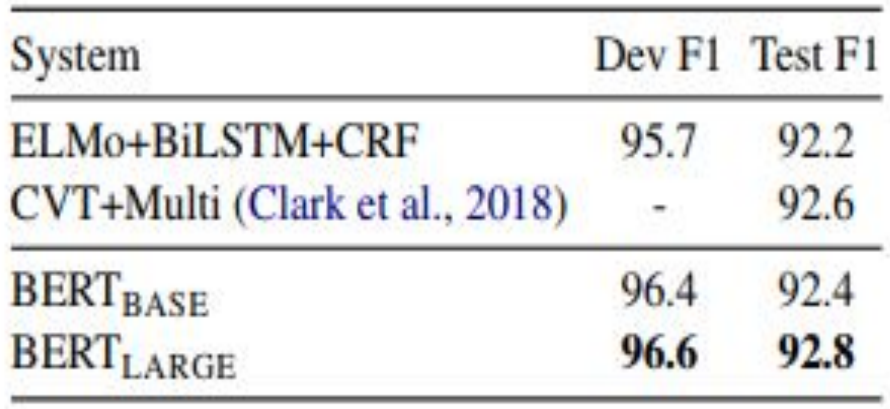
SQuAD Dataset



System	Dev		Test	
	EM	F1	EM	F1
Leaderboard (Oct 8th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
#1 Single - nlnet	-	-	83.5	90.1
#2 Single - QANet	-	-	82.5	89.3
Published				
BiDAF+ELMo (Single)	-	85.8	-	-
R.M. Reader (Single)	78.9	86.3	79.5	86.6
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**

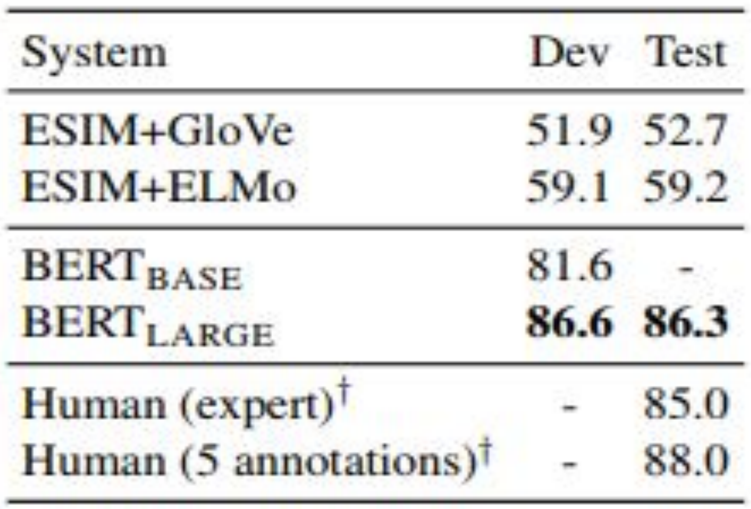
Table 2: SQuAD results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

NER & SWAG



System	Dev F1	Test F1
ELMo+BiLSTM+CRF	95.7	92.2
CVT+Multi (Clark et al., 2018)	-	92.6
$\text{BERT}_{\text{BASE}}$	96.4	92.4
$\text{BERT}_{\text{LARGE}}$	**96.6**	**92.8**

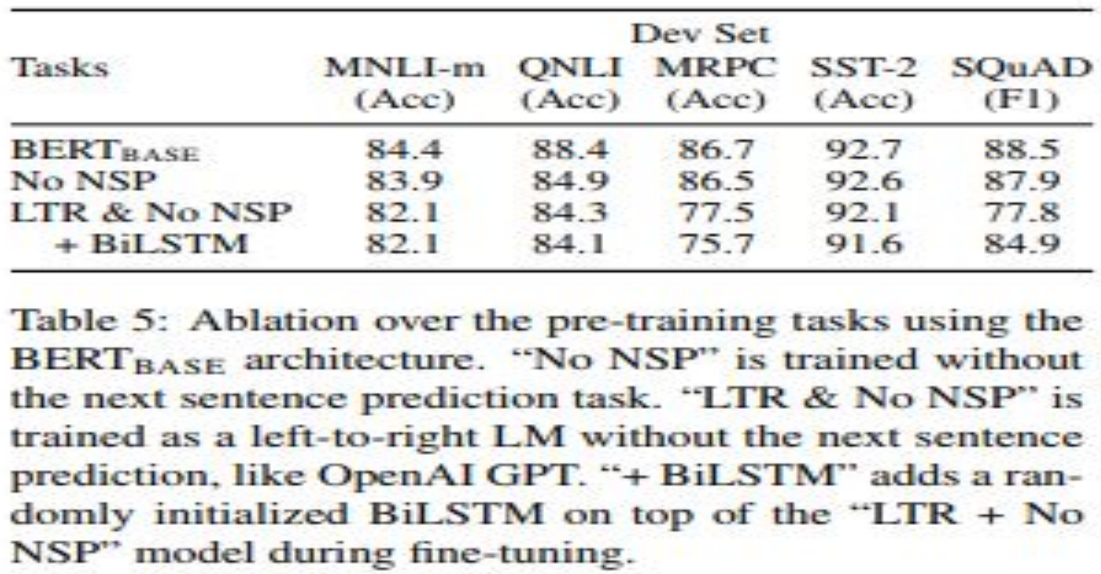
Table 3: CoNLL-2003 Named Entity Recognition results. The hyperparameters were selected using the Dev set, and the reported Dev and Test scores are averaged over 5 random restarts using those hyperparameters.



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

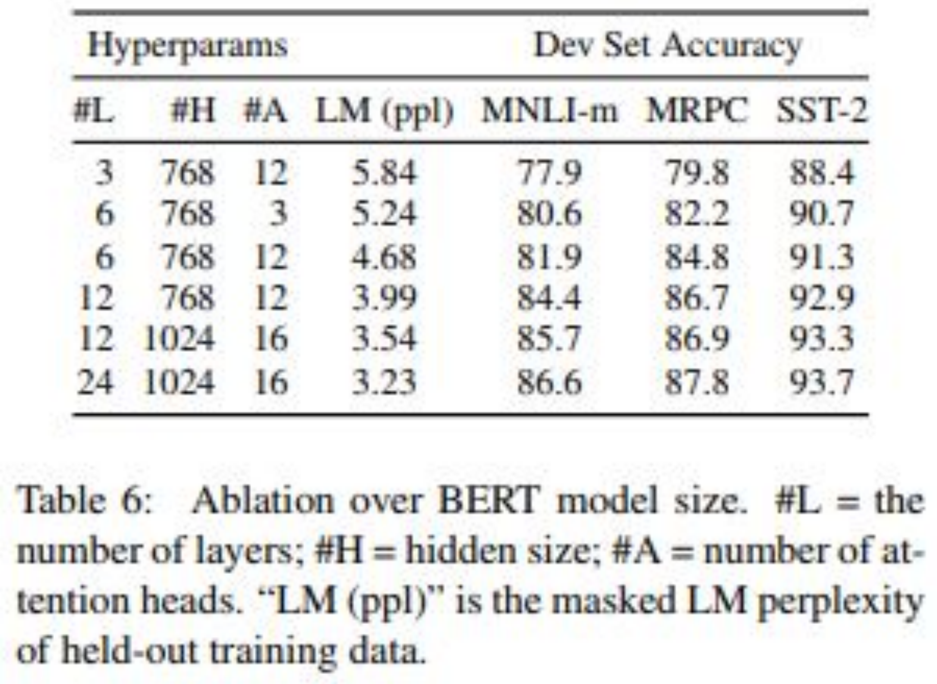
Table 4: SWAG Dev and Test accuracies. Test results were scored against the hidden labels by the SWAG authors. †Human performance is measure with 100 samples, as reported in the SWAG paper.

Ablation Studies/Findings



Tasks	Dev Set MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
BERT$_{BASE}$	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 BERT$_{BASE}$ 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.

Contd..

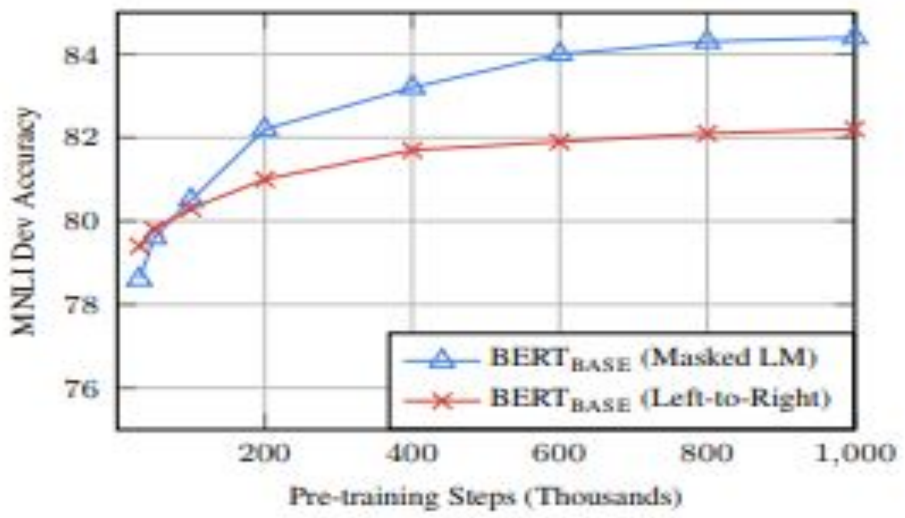


Figure 4: Ablation over number of training steps. This shows the MNLI accuracy after fine-tuning, starting from model parameters that have been pre-trained for k steps. The x-axis is the value of k.

Layers	Dev F1
Finetune All	96.4
First Layer (Embeddings)	91.0
Second-to-Last Hidden	95.6
Last Hidden	94.9
Sum Last Four Hidden	95.9
Concat Last Four Hidden	96.1
Sum All 12 Layers	95.5

Table 7: Ablation using BERT with a feature-based approach on CoNLL-2003 NER. The activations from the specified layers are combined and fed into a two-layer BiLSTM, without backpropagation to BERT.

References

- https://arxiv.org/pdf/1706.03762.pdf
- https://arxiv.org/pdf/1802.05365.pdf
- https://arxiv.org/pdf/1801.06146.pdf
- https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf
- https://medium.com/@hyponymous/paper-summary-bert-12af7c89d2e0
- https://medium.com/dissecting-bert/dissecting-bert-part2-335ff2ed9c73
- https://www.lyrn.ai/2018/11/07/explained-bert-state-of-the-art-language-model-for-nlp/
- http://jalammar.github.io/illustrated-bert/
- http://mlexplained.com/2019/01/07/paper-dissected-bert-pre-training-of-deep-bidirectional-transformers-for-language-understanding-explained/