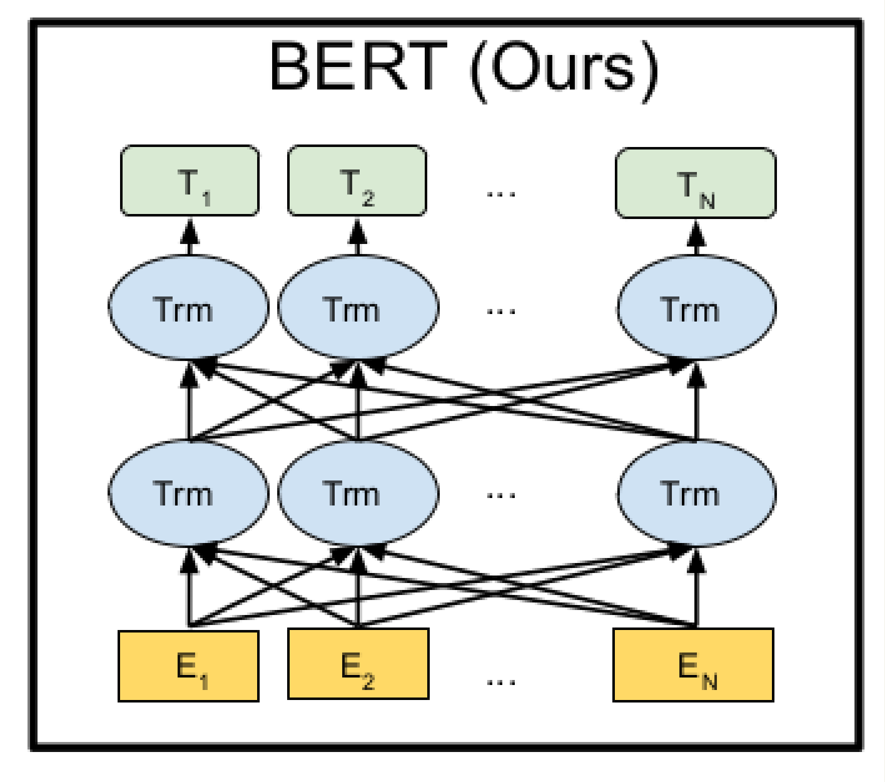


BERT: PRE-TRAINING OF DEEP BIDIRECTIONAL TRANSFORMERS FOR LANGUAGE UNDERSTANDING

Agenda

Background

Masked Language Model

Right to left Language Model

Left to Right with Right to Left Language Model

Transformers

Agenda

- BERT
- BERT Architecture
- Input Representation
- Input Embedding
- Pre-training
- Masked Language Model (MLM)
- Next Sentence Prediction (NSP)

Agenda

Using BERT

BEERT Fine-tuning

Experiments

- GLUE
- SQuAD v1.1
- SQuAD v2.0
- SWAG

Ablation Studies

- Effect of Pre-training Tasks
- Effect of Model Size
- Effect of number of training steps
- Models' comparisons

Conclusion

Background
Language Modeling

- **Language Modeling (LM)** means predicting the missing word in a sentence.
- **Masked Language Model (MLM)** means to mask (cover) a word in a sentence and train our LM to predict it

Example

- Suppose we have this input sentence:
- "The sky is blue"

- We will mask the verb "**is**": "The sky ---- blue"

Left To Right LM

- Traditional Techniques is to use RNNs with Linear layer and softmax to predict the missing word

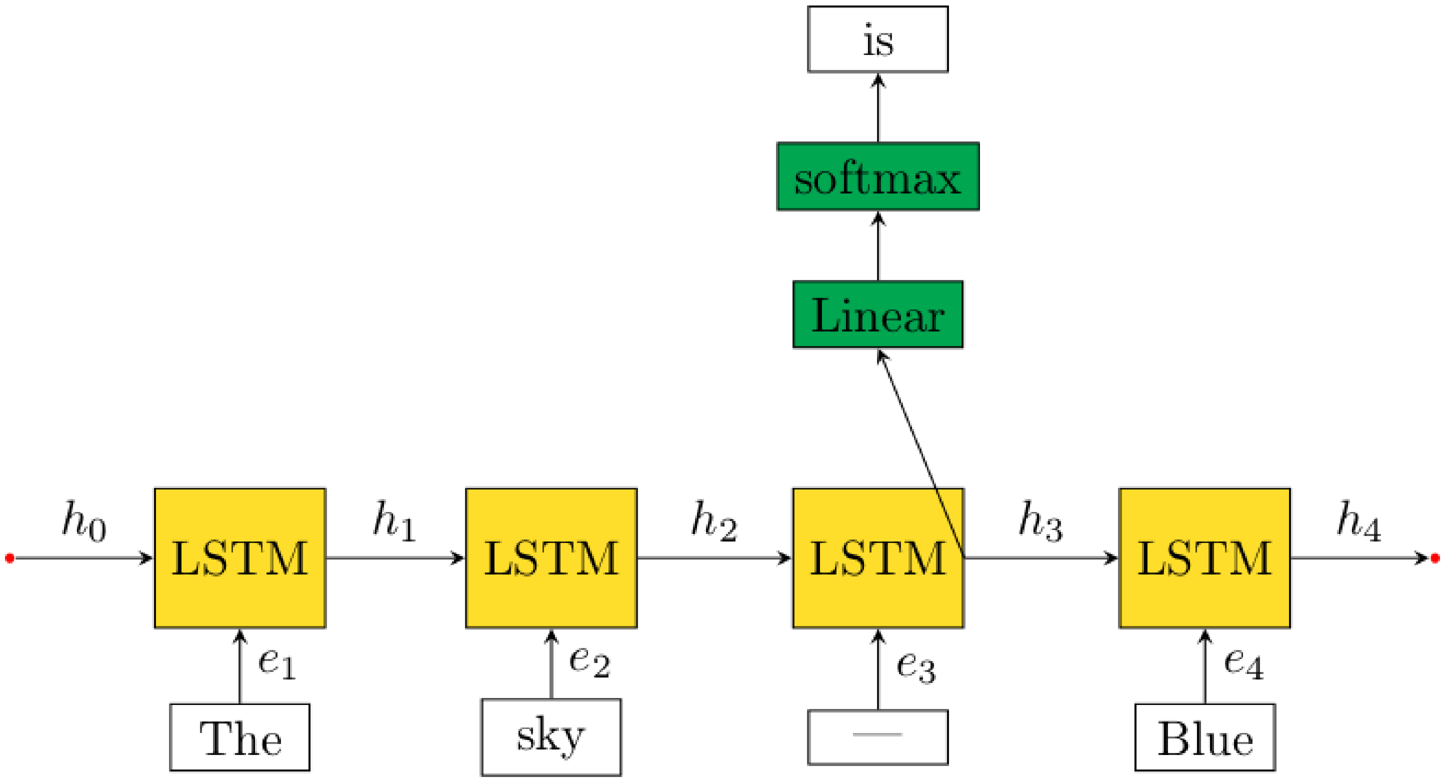


Figure 1: Unidrectional Masked Language Model

WE have problems:

- The model has no way to make prediciton is "goes" instead of "is"
- "The sky goes blue"
- because he cannot see the right context (I.e: "Blue" which is adjective)

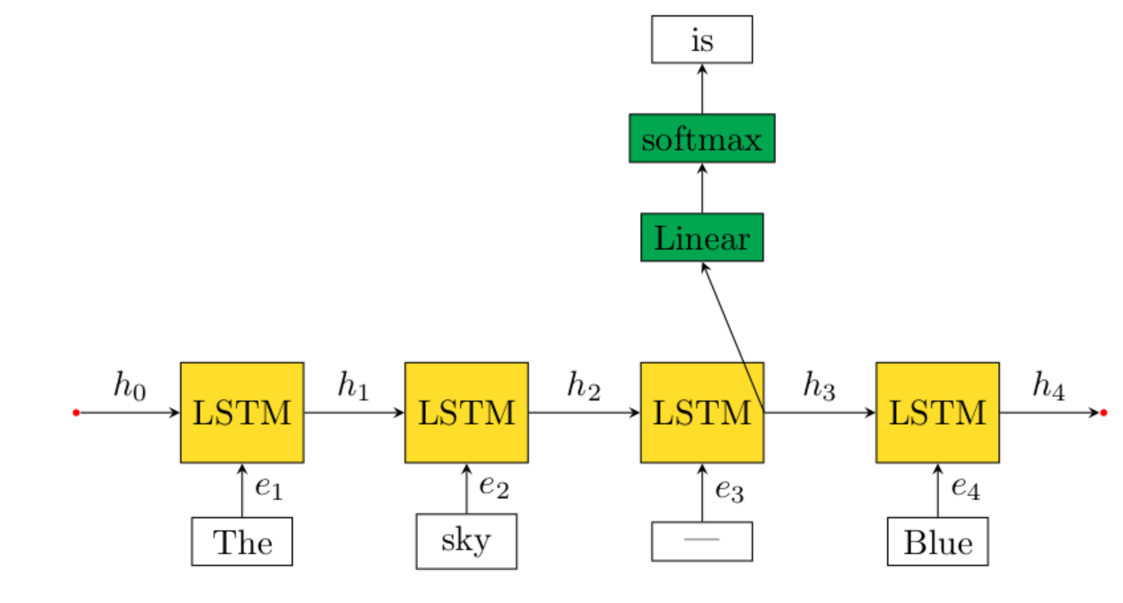


Figure 1: Unidrectional Masked Language Model

RTL with LTR LM

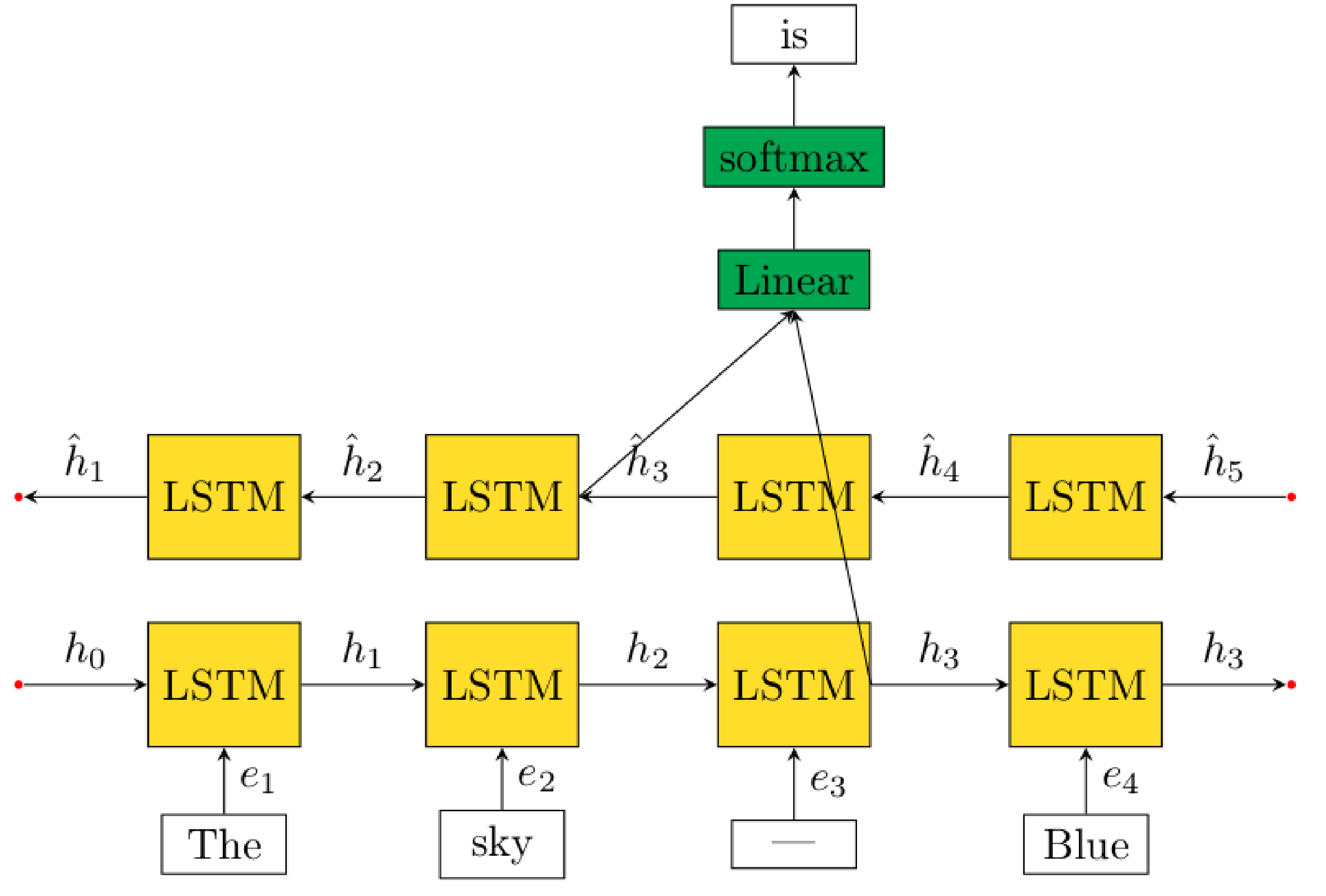


Figure 2: Bidirectional Masked Language Model

- To solve this problem, we add another RNNs that take care of the Left to right context
- Then we concatenate the RTL, and LTR
- Then Linear and softmax layers

RTL with LTR LM

- Successfully We predict the correct word "is":
- "The sky is blue"

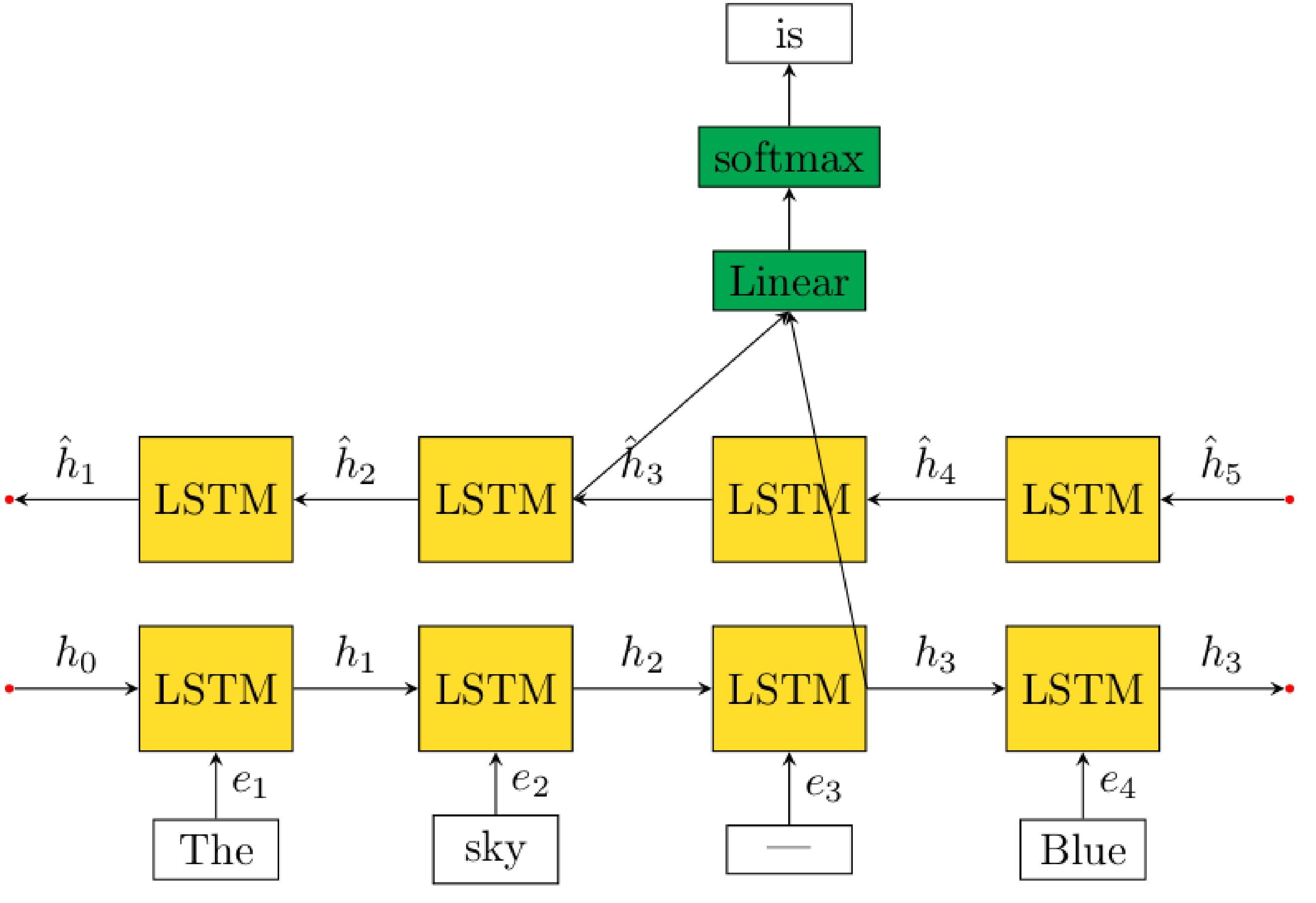


Figure 2: Bidirectional Masked Language Model

RTL with LTR LM Problem

- Suppose the sentence is too long :
- "The ball that children plays with in school is blue"
- We mask "is"
- "The ball that children plays with in school ----- blue"

This Photo by Unknown author is licensed under CC BY-NC.

- "The ball that children plays with in school ----- blue"
- Does our model predict "is" -> ball, or "are" -> children ?

The answer is **NO**

Because the Children is closer to the masked word with this architecture

LTR with RTL with Attention

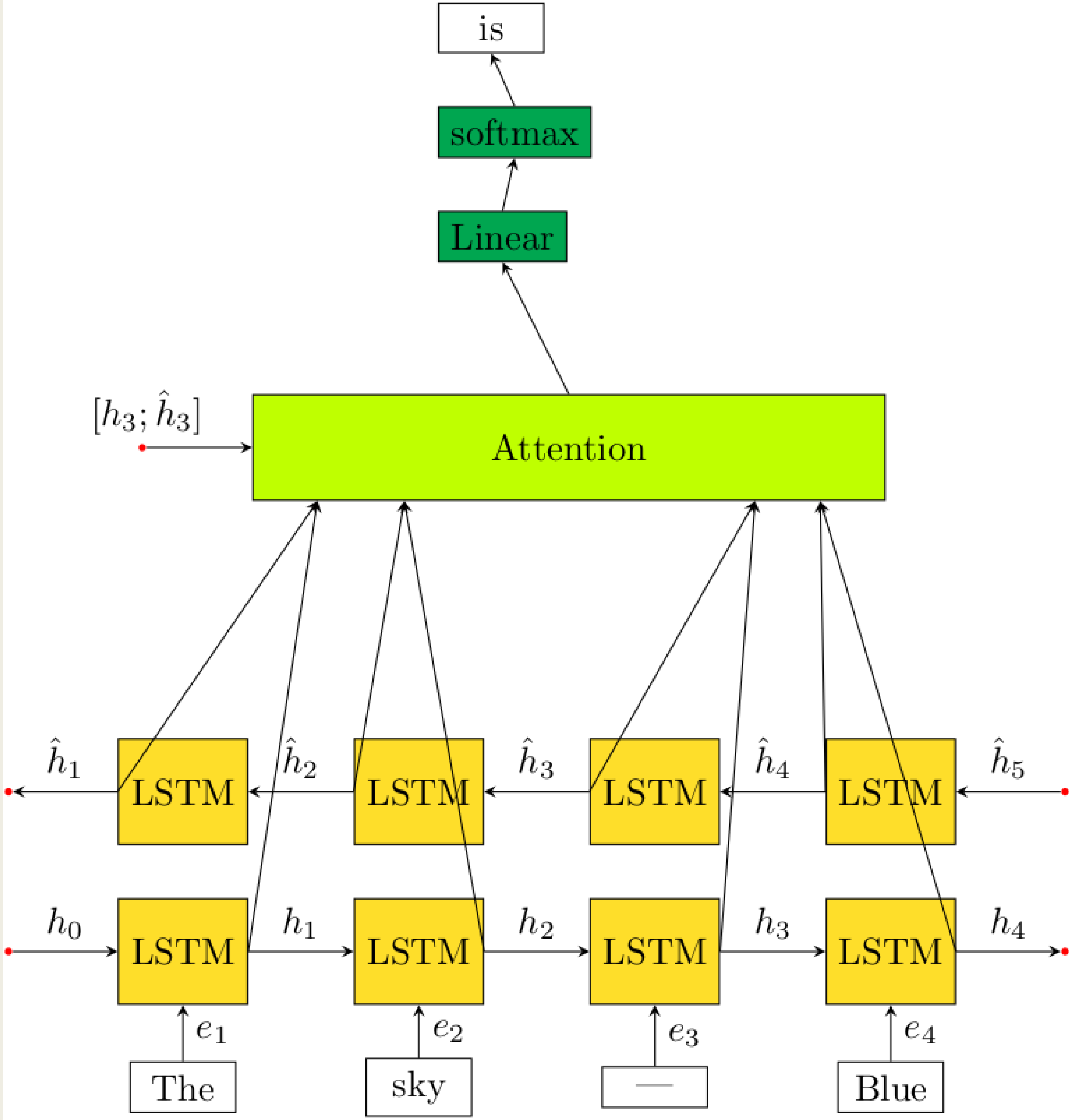


Figure 4: Bidirectional Masked Language Model with Attention

Attention

- We multiply every hidden state by a Contant **αi**, then we sums all the like this:
- Then Linear and softmax

$$\alpha_i = h_{masked} \cdot h_i$$

$$d_{masked} = \sum_{i}^{N} \alpha_i h_i$$

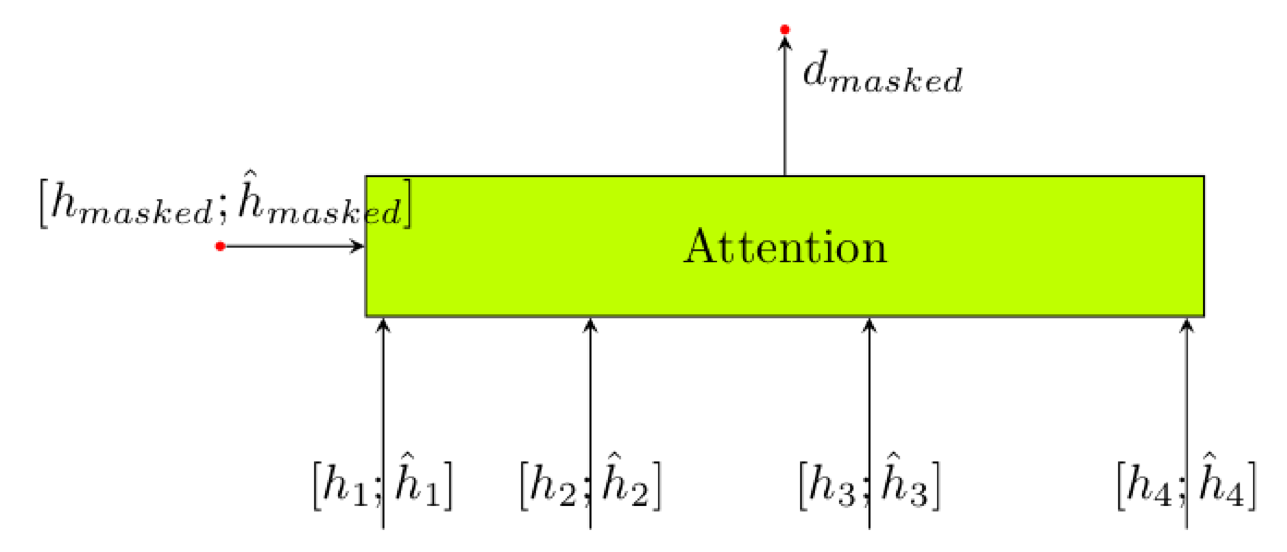


Figure 3: Attention block diagram

LTR with RTL RNNs

- Again, we have a problem:
- RNNs are sequential in nature, but we have GPUs right ?
- Unfortunately, We GPUs loves parallel tasks not sequential task.
- What to do ?
- What if we take this attention block and replace it with the bidirectional RNNs.
- WE introduce "Self Attention" aka: Transformers

Selft Attention

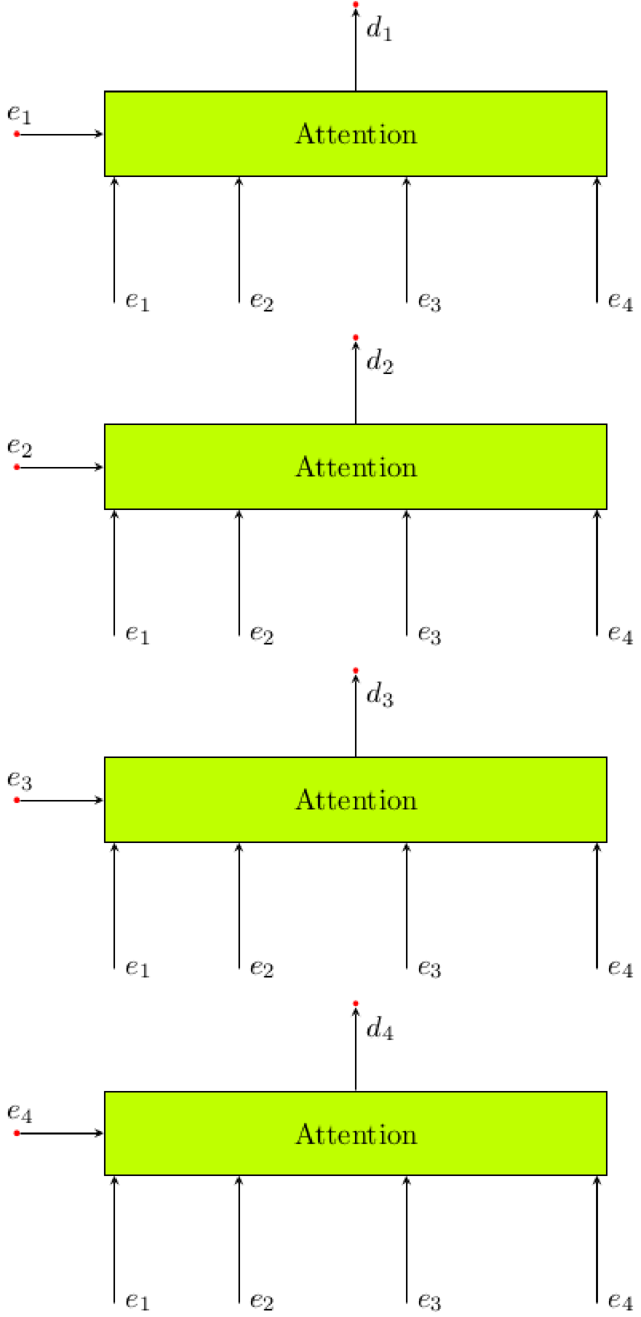
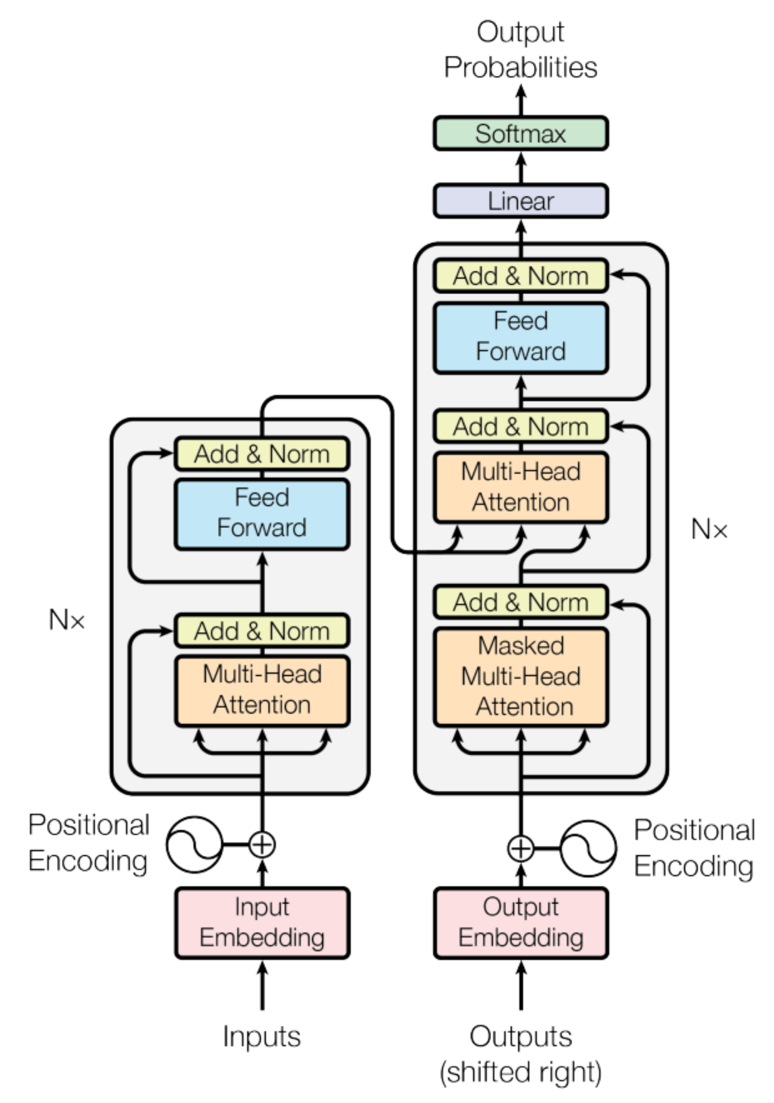


Figure 5: Self Attention block diagram
Attention for every token (i.e: word). The new d will replace h as in Figure 4.

TRANSFORMS

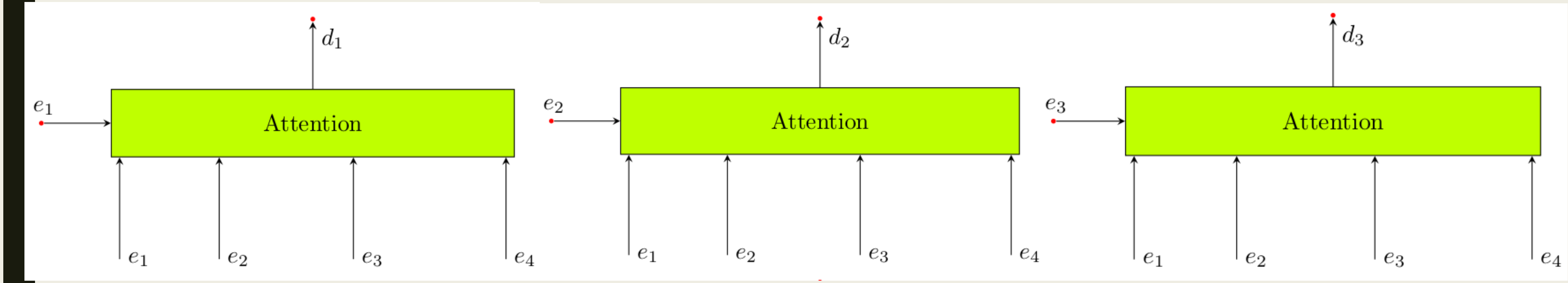
Not this

Transformers

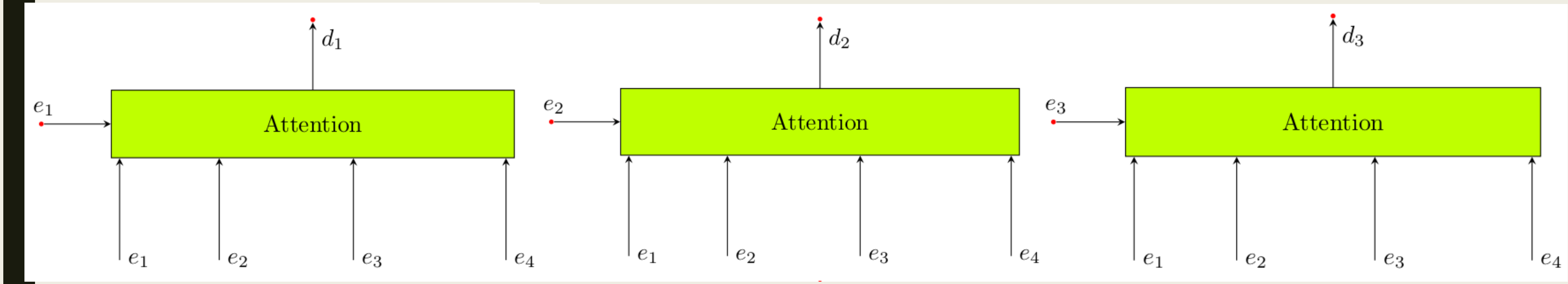


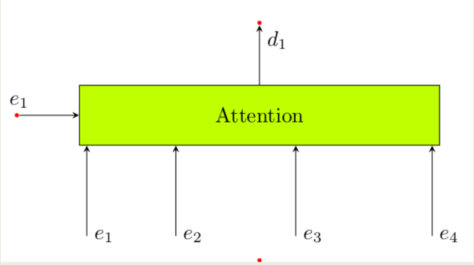
But This

Attention layer

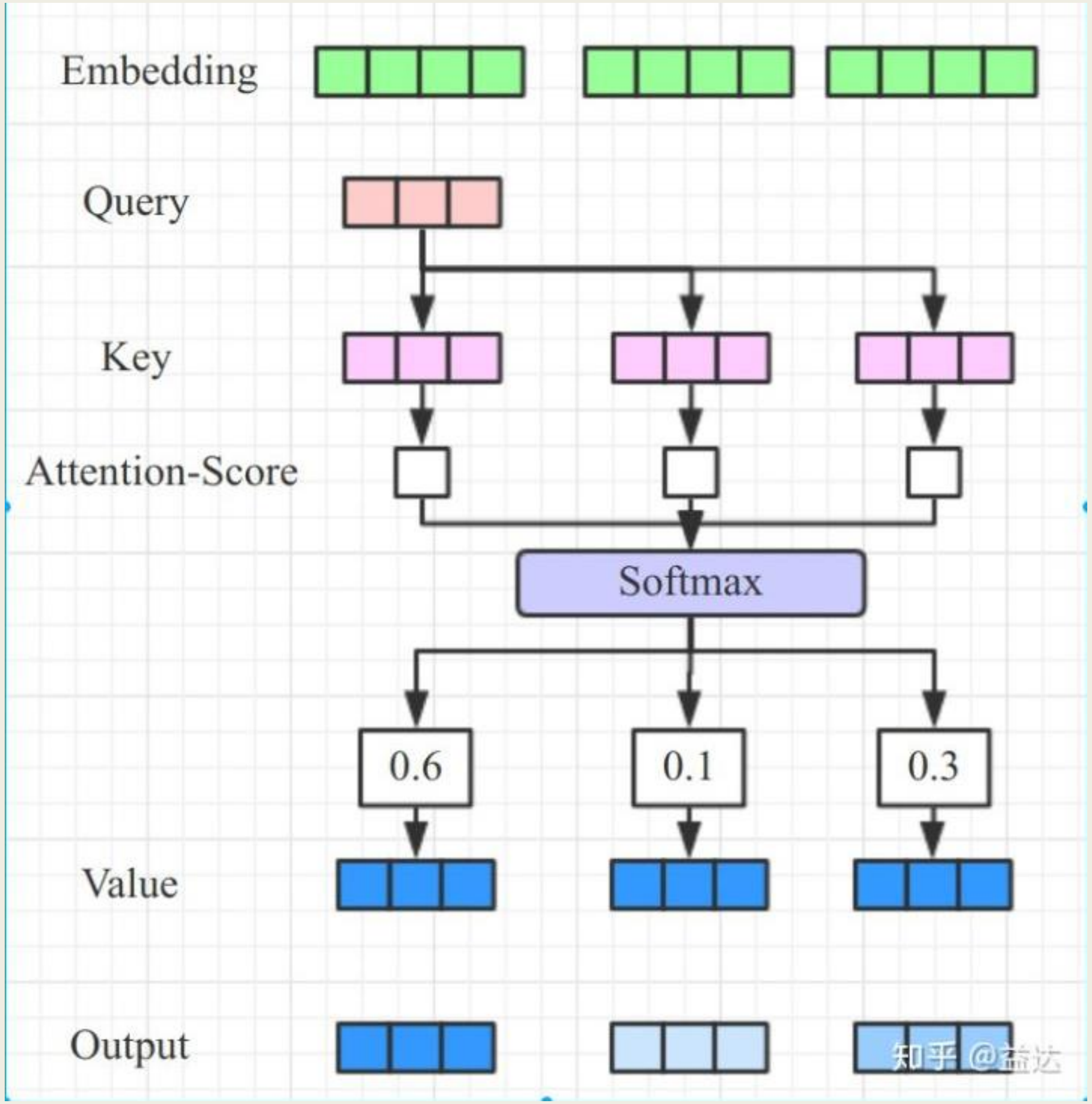


Can Attention handle variable length sequence ?

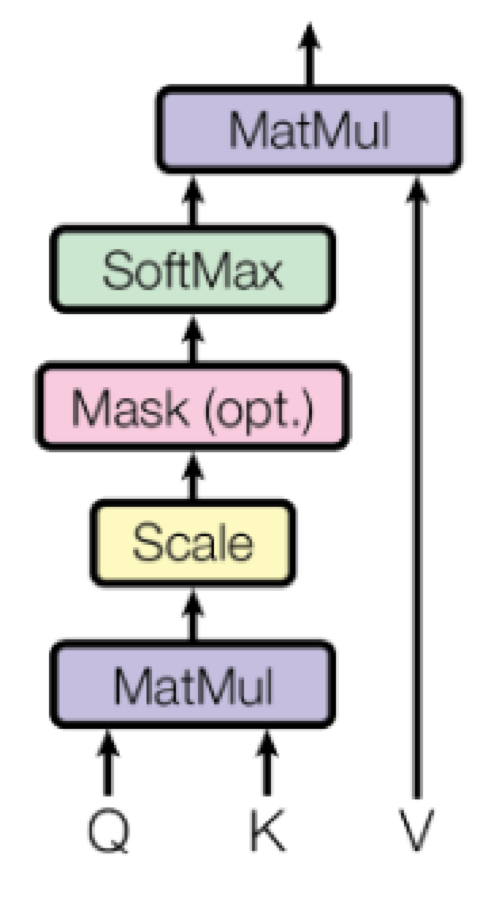




- Let's dive into attention block



- We have three matrices shared across all self-attention block:
- Wq, Wk, Wv
- query= e_i Wq
- Key = e_i Wk
- Value = e_i Wv

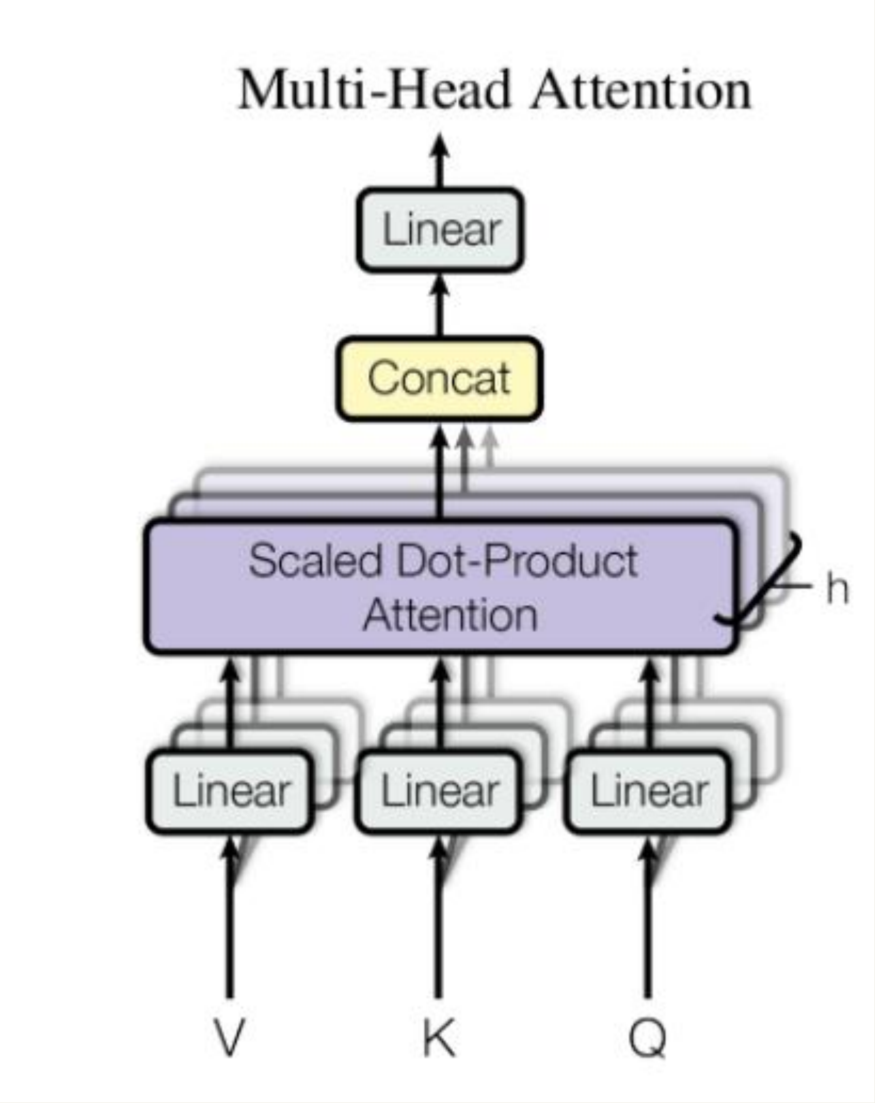


SELFT ATTENTION BLOCK

We can generalize as matrix multiplication (more parallel)

Multi head Attention

- We can add multiple parallel "self attention" blocks (A) with different: Wq, Wk, Wv
- Then we have a decoder (has multiple parallel heads I.e: "self attentions" blocks)



- The complete transformer

  Stacking multiple Encoders -> then we have a full Transformer decoder

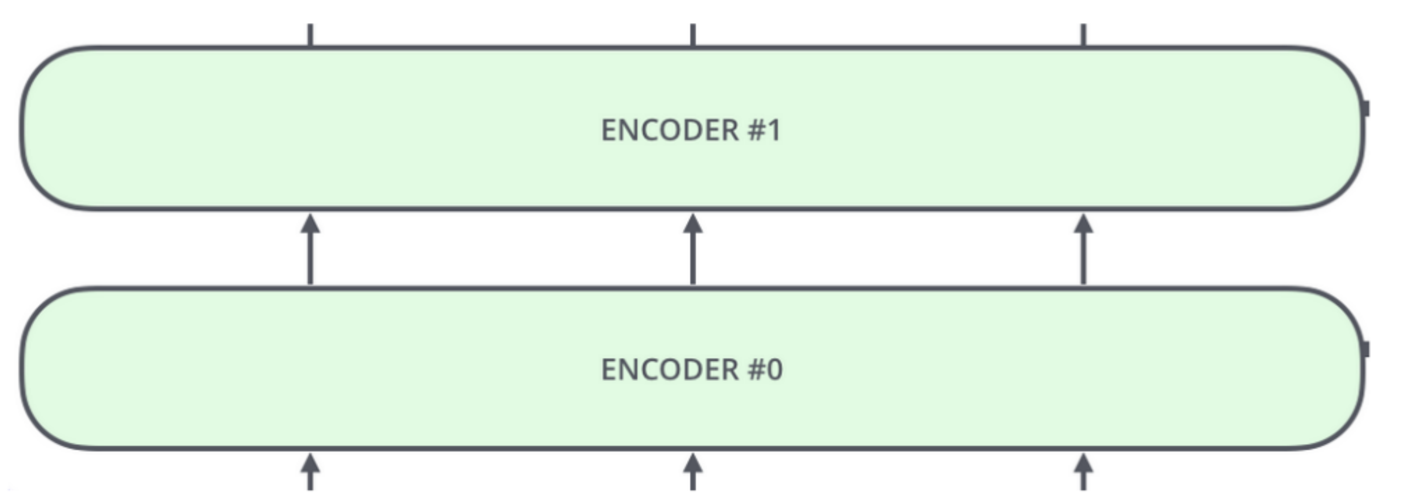


Figure 8: The full Encoder of the Transformer.
Every $Encoder_i$ has (A) Heads (i.e: "Self Attention" blocks) as in Figure 7

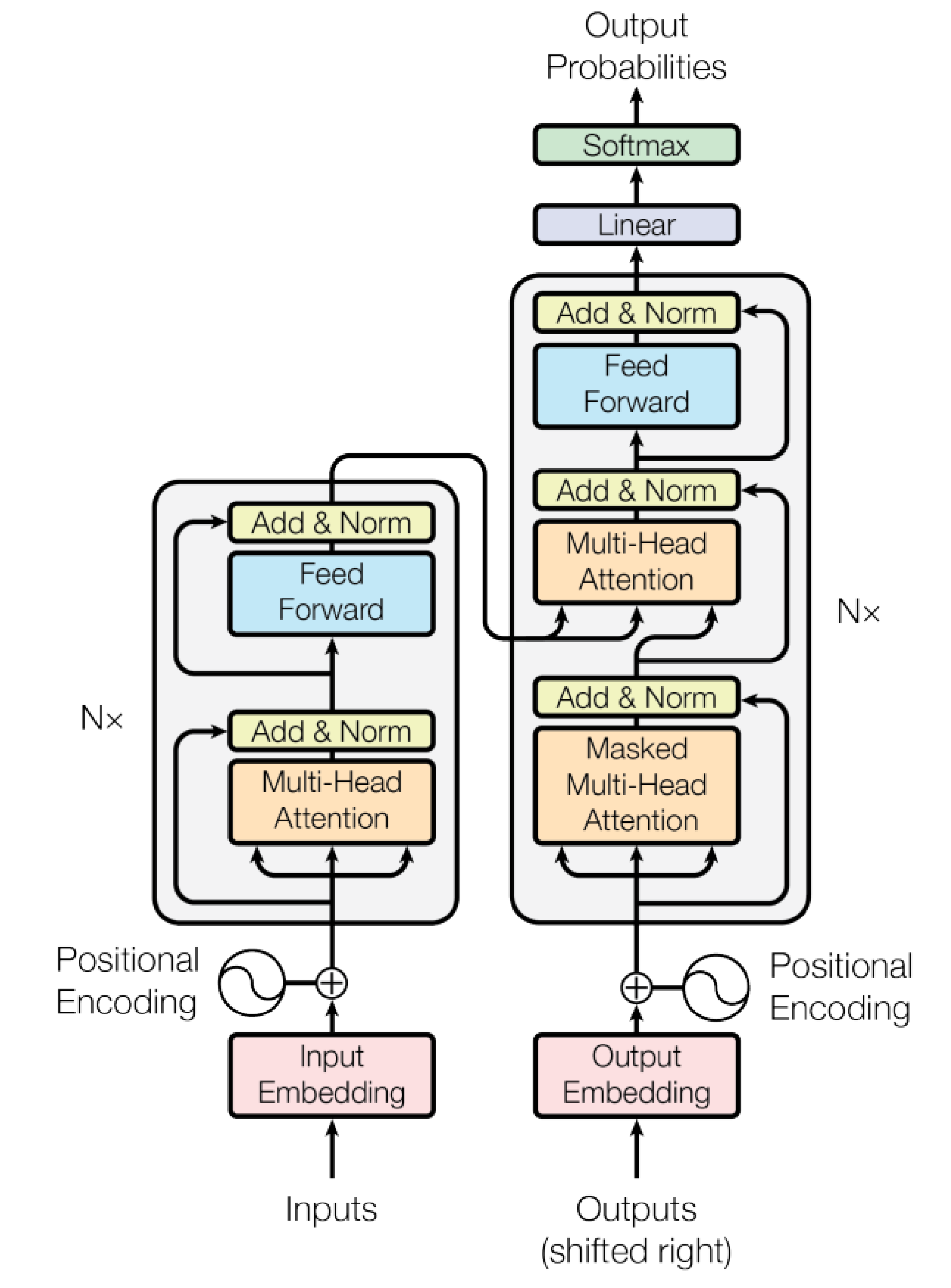
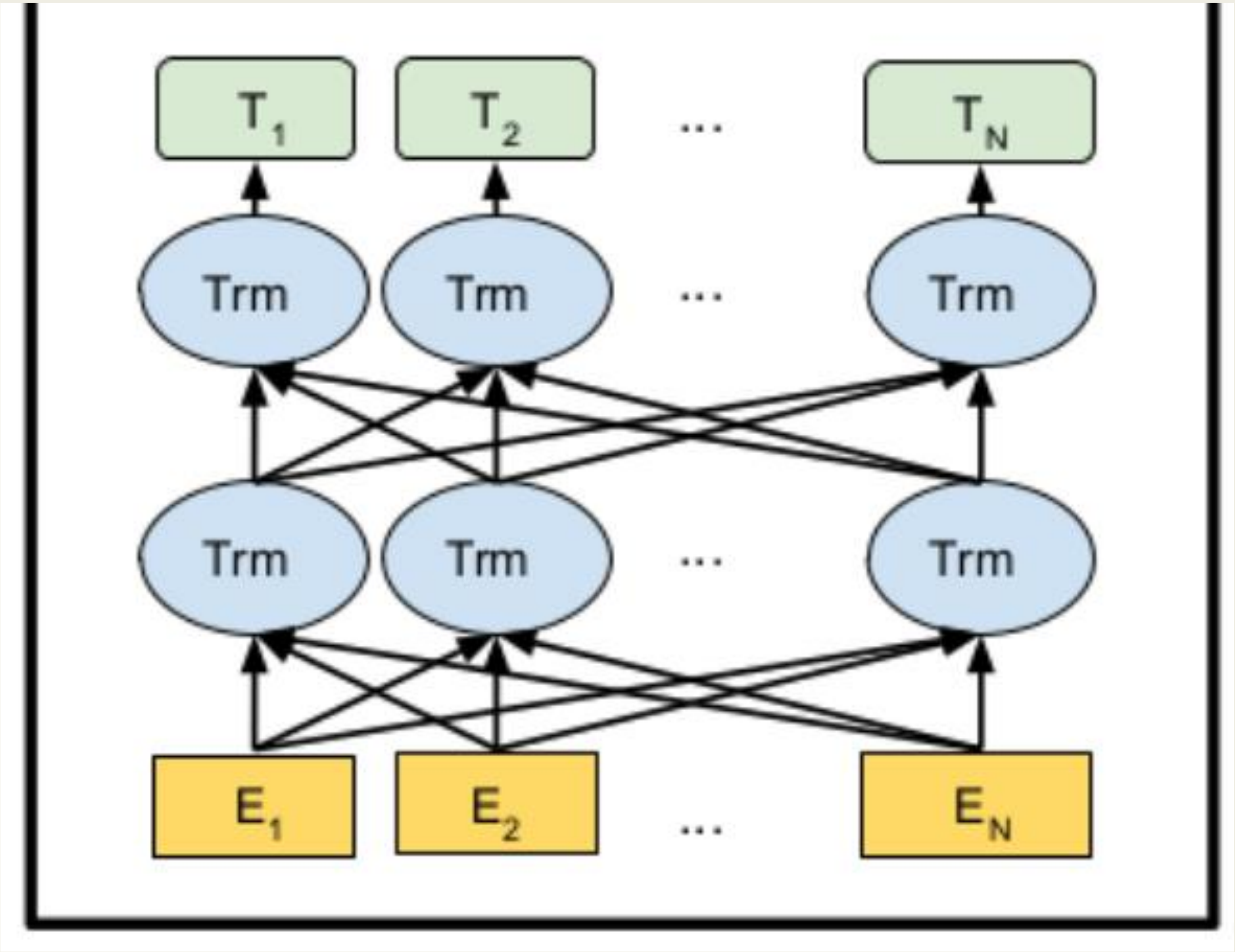


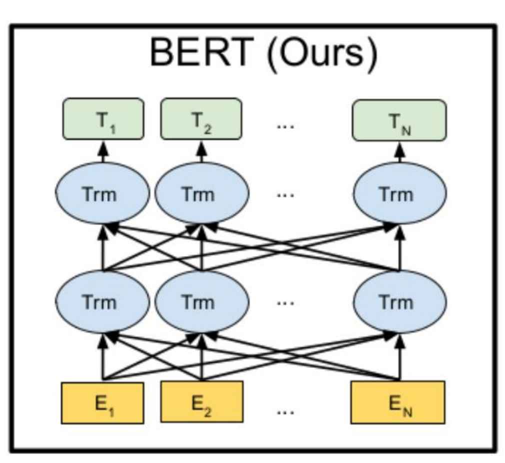
Figure 1: The Transformer - model architecture.

The transformer Encoder

- Every output has contextual representation of all the inputs



BERT



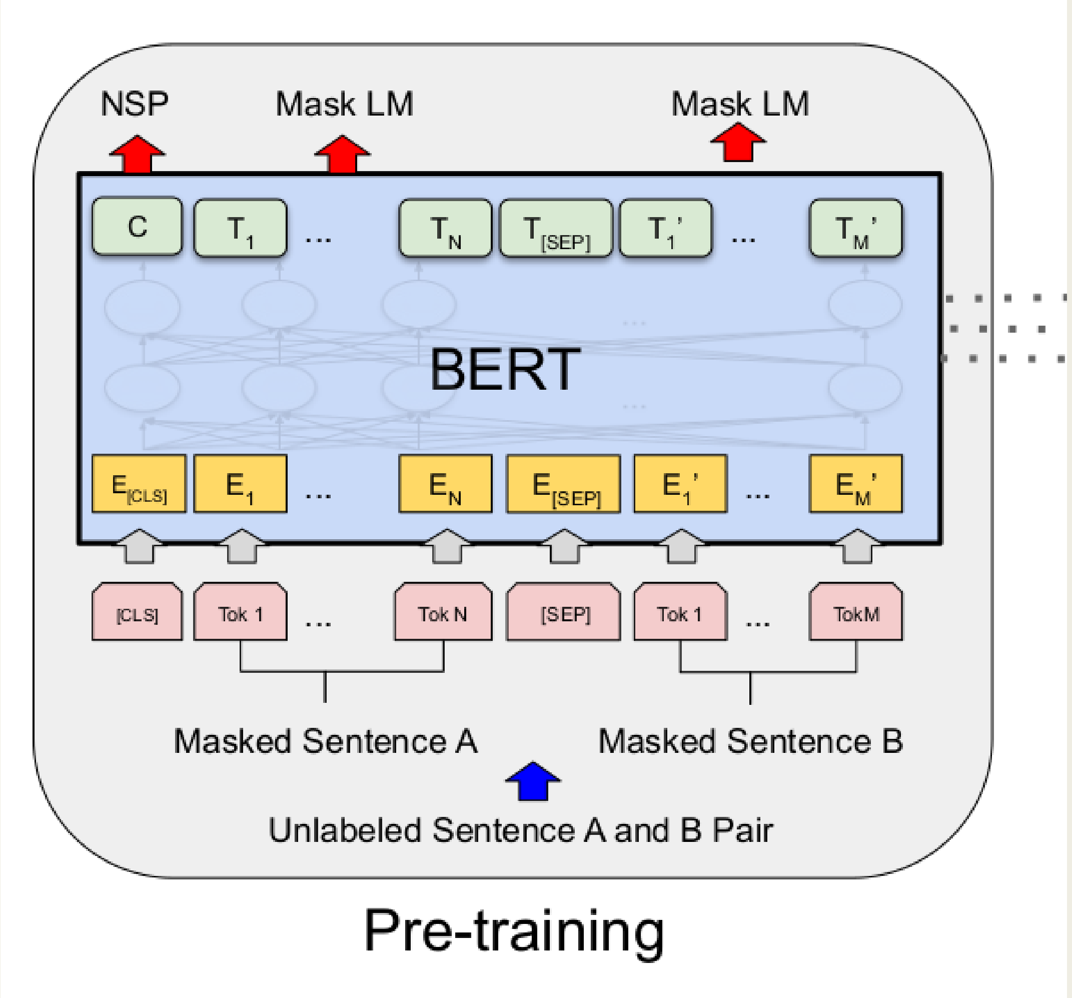
- Finally, we head to our paper goal which is BERT
- BERT: **B**idirectional **E**ncoder **R**epresentation form **T**ransformers.
- Why is it called ImageNet of the NLP ?
- BERT was pre-trained on 800M word corpus, 2,500M Wkippedia, and achieved state of the art in 11 NLP tasks !!!!!
- BERT Introduces a generic model that can be fine-tuned (2 to 3 epochs) a single hour on TPU, and we can get the state of the are results.
- It has more than 40K citations !!!!!

BERT Architecture

- BERT is the Encoder block of the transformer
- L: is the # encoders, A #heads, H the length of hidden vector

Model	**#L**	**#A**	**#H**	**#Prameters**
$\text{BERT}_{\text{base}}$	12	12	768	110M
$\text{BERT}_{\text{large}}$	24	16	1024	340M

BERT Architecture Input Representation

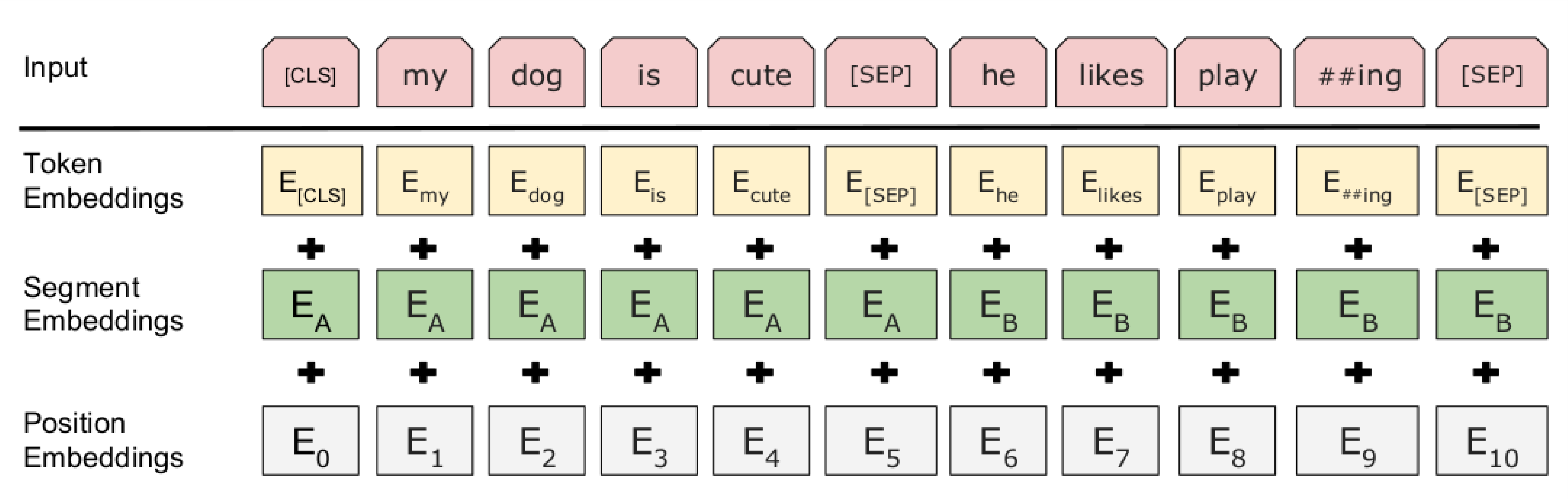


- The model input is a sequence of two sentences: **sentenceA** and **sentenceB.**
- Sentence here means: a sequence word not the linguistic sentence
- We have 3 special tokens:
  - **[cls]**: (class) token donates the start of sequence is ouput is C
  - **[sep]**: end of sentences: sentenceA, and sentenceB
  - **[mask]**: a token to the hidden sentence

BERT Architecture
Input Embedding

- We use wordPiece embedding with 300,000 words as input.
- The second layer is the positional encoding form transformer
- The third layer is sentence embedding for ever sentence, sentenceA, and sentenceB

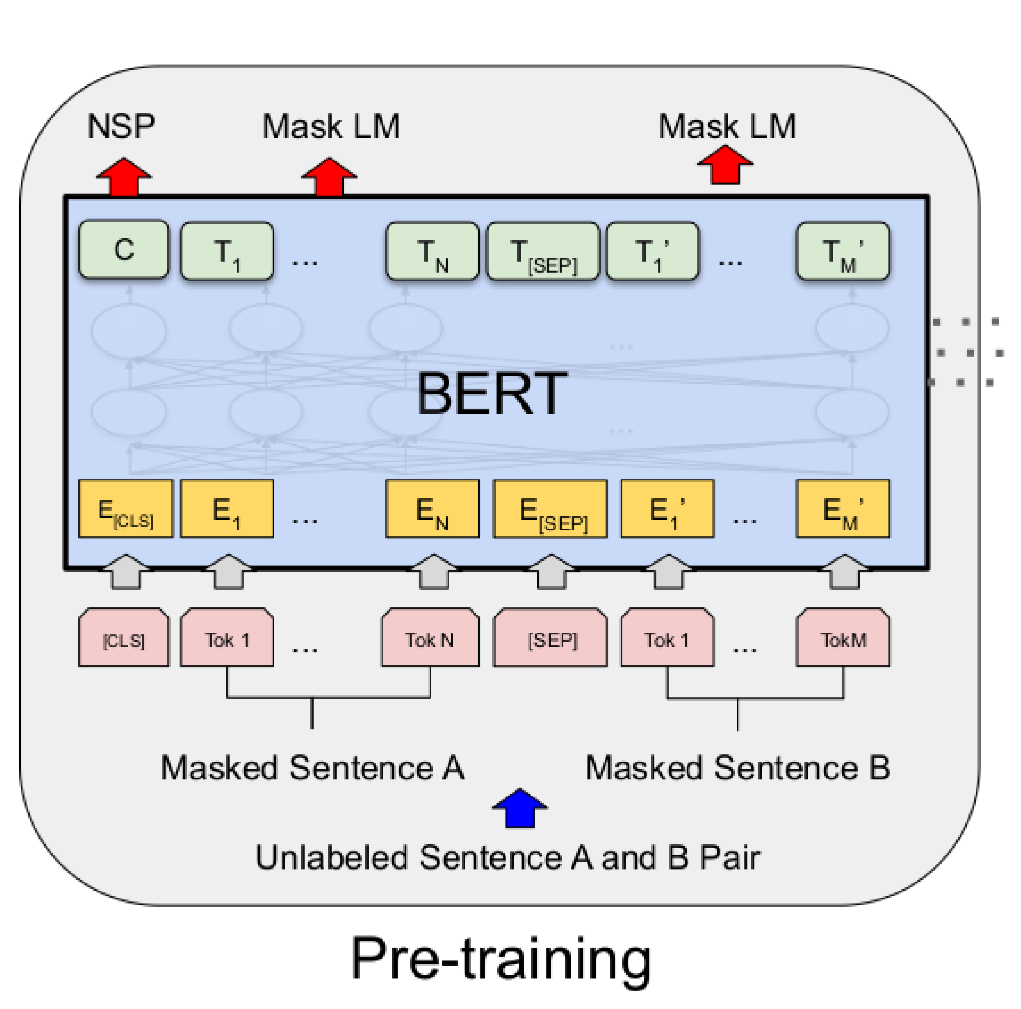
BERT Architecture
Input Embedding



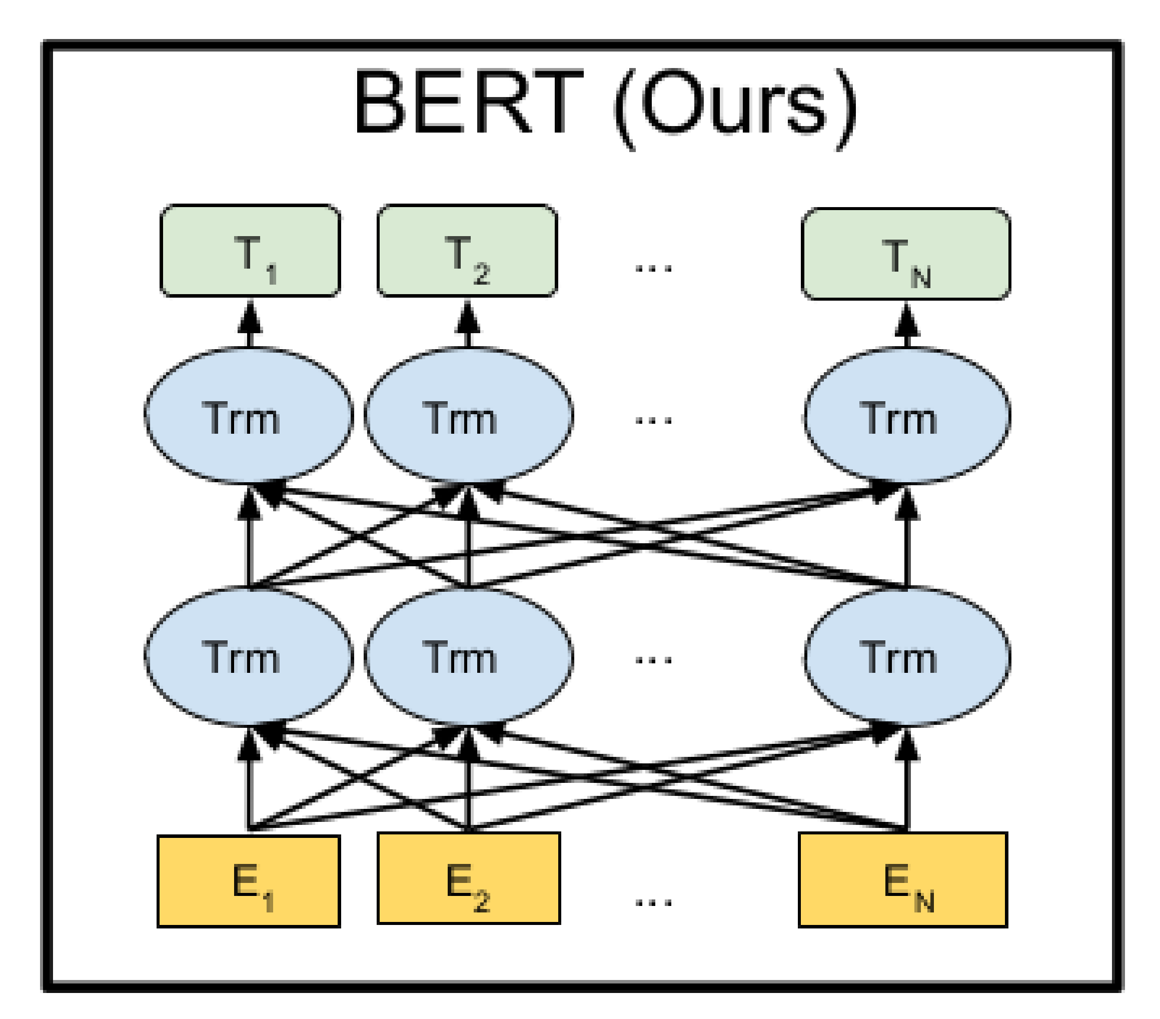
Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Token Embeddings	$E_{[CLS]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[SEP]}$	E_{he}	E_{likes}	E_{play}	$E_{\#\#ing}$	$E_{[SEP]}$
	+	+	+	+	+	+	+	+	+	+	+
Segment Embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	E_B
	+	+	+	+	+	+	+	+	+	+	+
Position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

BERT Architecture

BERT is the encoder of the Transformer



Pre-training



BERT
Pre-training

- We pre-train our model on two things:
  - Masked Language Model (MLM)
  - Next Sentence Prediction (NSP)

BERT Pre-training
Masked Language Model

- We will mask 15% of the input sequence the masked words will be replaced by:
  - • *the [mask] token 80% of the time*
  - • *a random word (i.e: token) 10% of the time.*
  - • *the word itself 10% of the time.*

BERT Pre-training
Masked Language Model

- 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.

BERT Pre-training
Next Sentence Prediction

- In many application we have more than one sentence like: Question Answer
- We pre-train our model to learn relation between sentences by the following:
- we train a barbarized next sentence predictor (NSP) that predicts whether the SentenceB follows SentenceA or not. At training we put:
  - *50% of the time SentenceB is actually following sentenceA with labeled as **IsNext**.*
  - 50% of the time SentenceB is NOT following sentenceA with labeled as **NotNext**.
- This binary classification is done throw the output token **(C)** the corresponds to input token **[cls]**

HOW TO USE THE PRE-TRAINED MODEL ?

Using BERT pretrained model

- WE have two approaches:
  - *Fine-tuning*
  - *Feature Based*

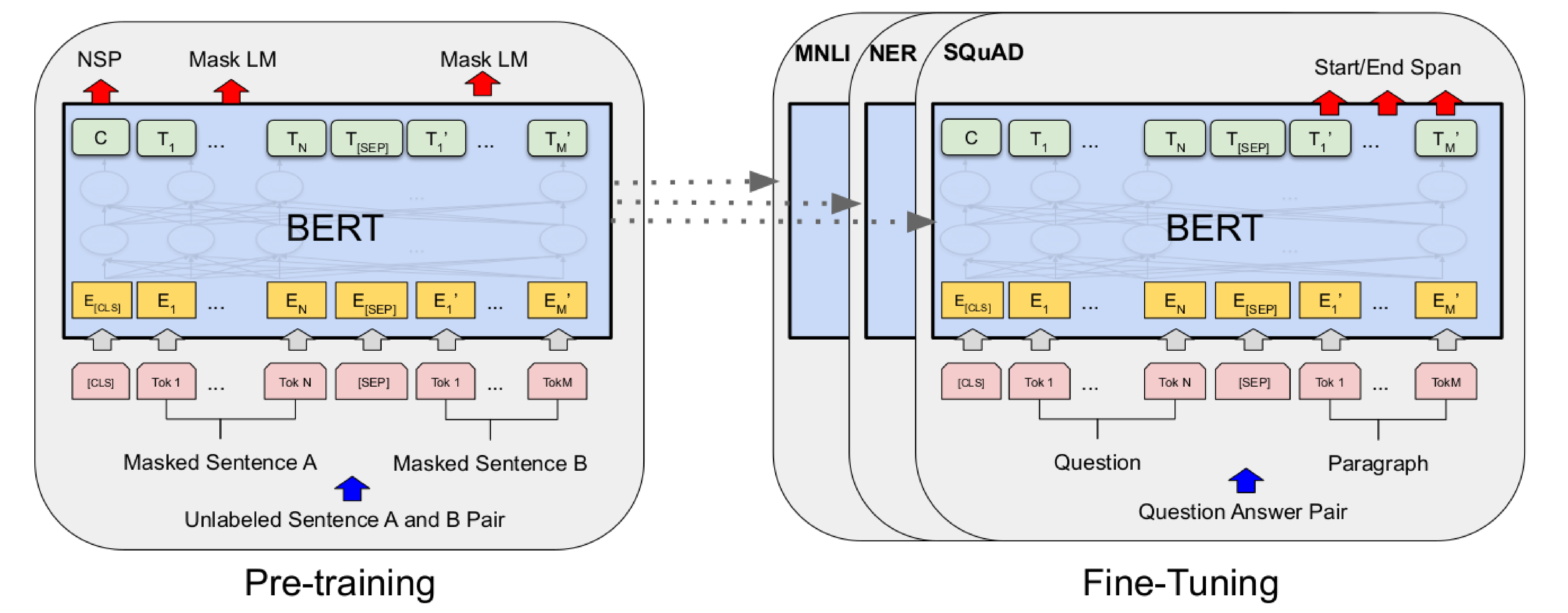
BERT Fine-tuning

- Fine-tuning means make small changes at the BERT model with weighs initialized form the pre-training
- And train form small epochs (2 to 4)
- These small changes includes:
  - Manipulating BERT input
  - Manipulating BERT output

BERT Fine-tuning
Manipulating Input

- While manipulating input: we have two sentences: sentenceA, sentenceB to the
- BERT input. The **sentenceA**, **sentenceB** are:
- • Sentencing paris in praphrasing task.
- • hypothesis, and premises (e.g: Mohamed goes to school by bike, Mohamed can ride bike) in instalment task.
- • question, and sentence that have the answer in Question Answer Task.
- • input sentence, and degenerate textφ in text classification tasks.

BERT Fine-tuning Manipulating Output



BERT Fine-tuning Manipulating Output

- Binary Classification: (like Sentiment classification, or entailment classification) we use the output **(C)** that corresponds to **[cls]** token as.
- Downstream Tasks that involves manipulating more than one output token **(T_i)** like; Question Answer

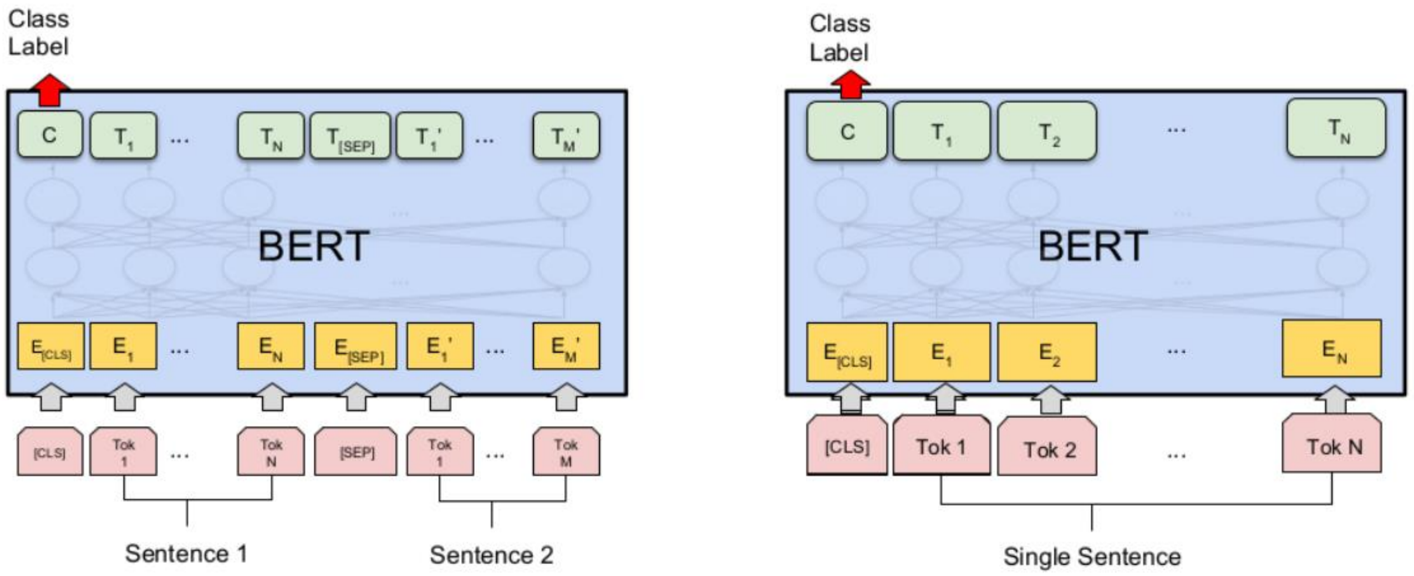
Experiments
GLUE Benchmark

The General Understanding Evaluation Benchmark (GLUE) a set of several data sets of different classification tasks.

We used $C \in R^{H}$ that corresponding the aggregate representation for the input sequence, and a classification layer $W \in R^{KxH}$, where K is the number of classes.

We used standard classification loss i.e: log(sof tmax(CW^{T})).

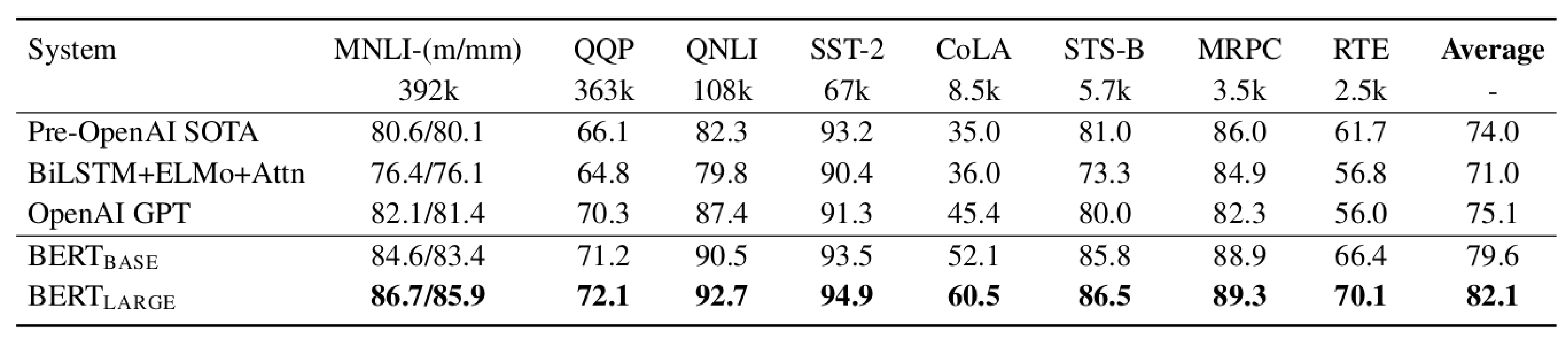
Experiments
GLUE Benchmark



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

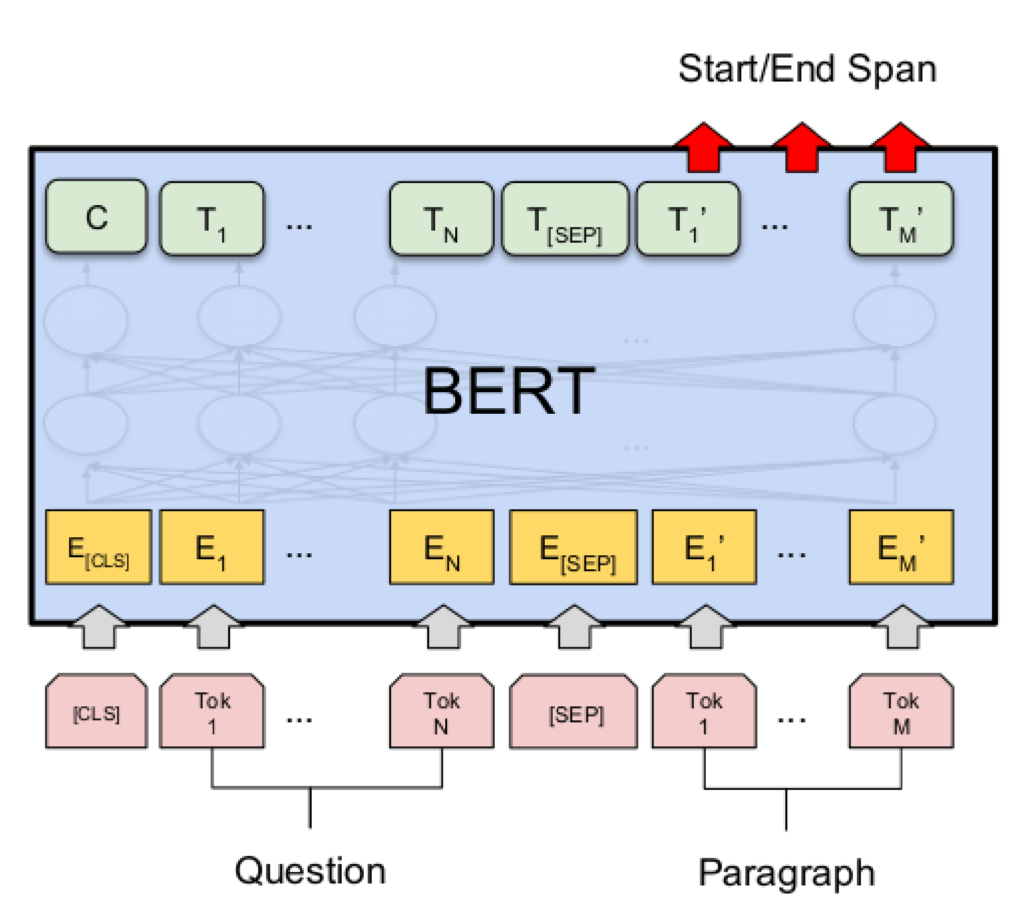
(b) Single Sentence Classification Tasks: SST-2, CoLA

Experiments
GLUE Benchmark



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	79.8	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**

Experiments SQuAD v1.1



(c) Question Answering Tasks:
SQuAD v1.1

- The Stanford Question Answering Dataset (SQuAD v1.1) is a collection of 100k crowd-sourced question/answer pairs. In QA we have two sentences:
  - *The question Sentence, that will be sentenceA in BERT.*
  - *The Answer Sentence that have the answer, that will be sentenceB in BERT.*
- Our goal is to find the start and the end of the answer in the answer sentence.

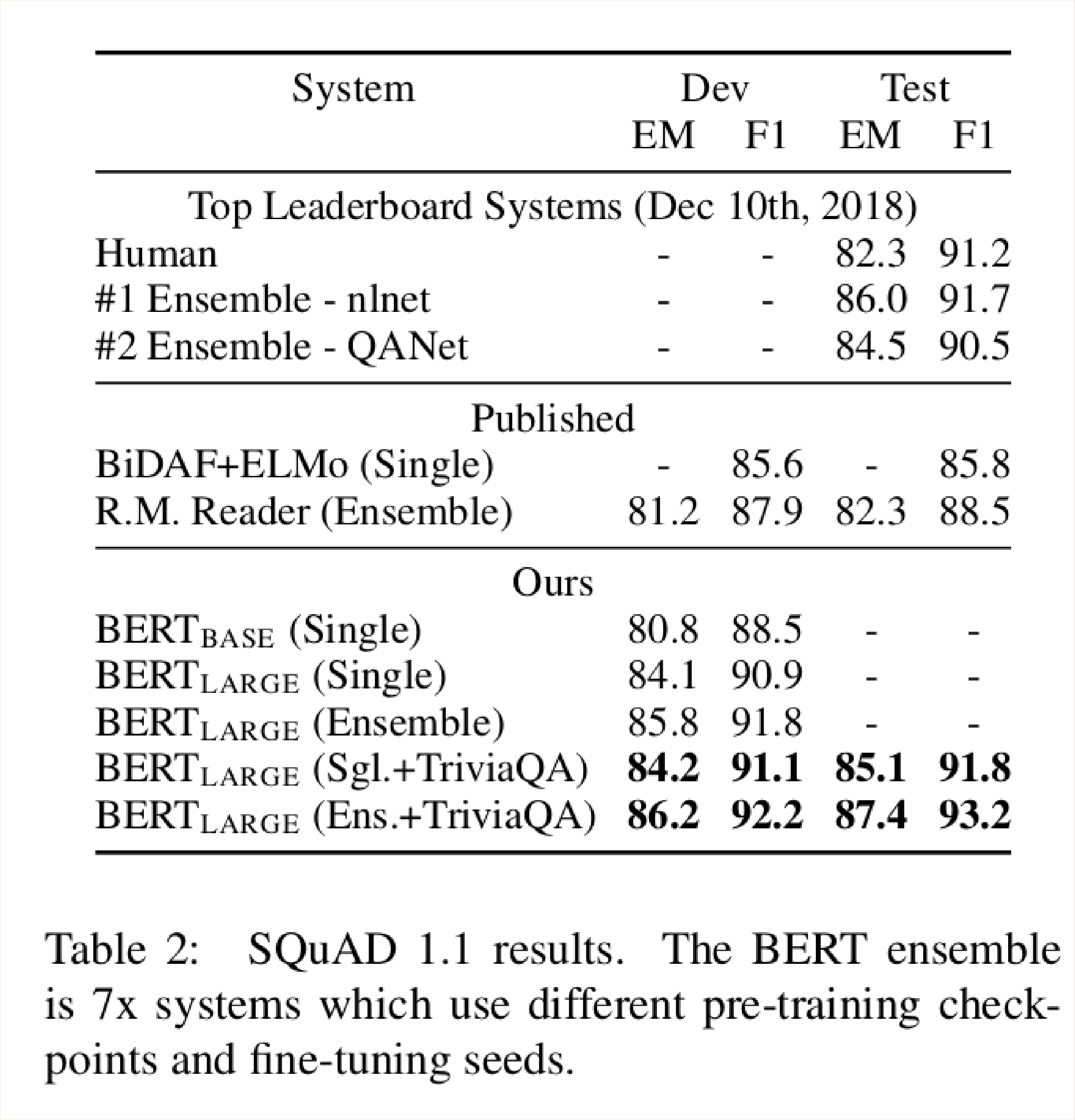
Experiments
SQuAD v1.1

- As we goal is to find the start of answer token $T_i \in R^H$, and end of answer token $T_i \in R^H$.
- We introduce start vector ($S \in R^H$, then we do product S with every output token of sentenceB T_i ,
- Then we take softmax among words in sentenceB as in equation.
- Similar to start we introduce end vector ($E \in R^H$, the end of answer is calculated as in where i is the start token, j is the end.
- **The loss** is the log-likelihood of correct start and end positrons

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

$$S.T_i + E.T_j, \quad where j > i$$

Experiments
SQuAD v1.1



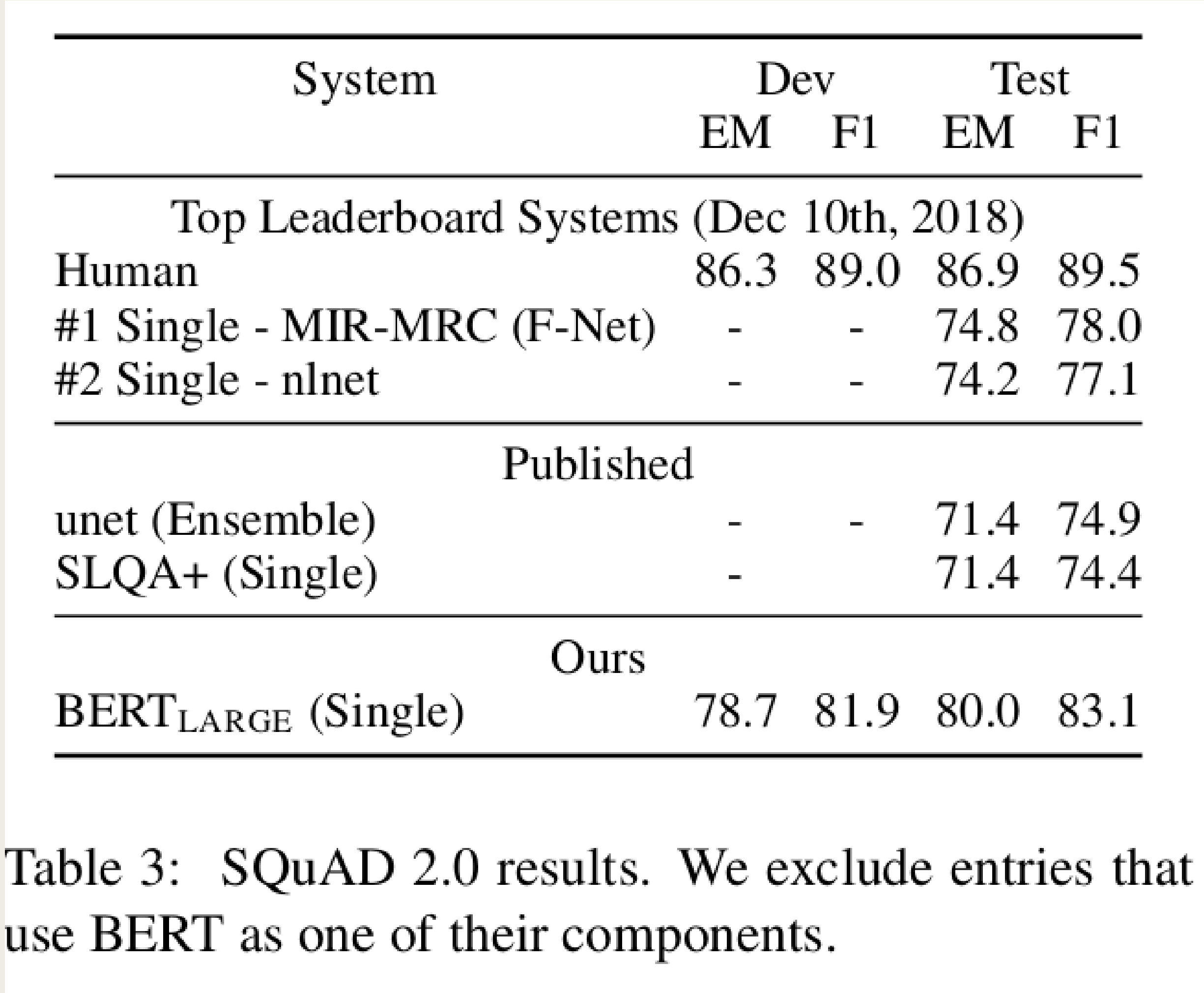
System	Dev EM	Dev F1	Test EM	Test F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
BERT$_{\text{BASE}}$ (Single)	80.8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

Experiments
SQuAD v2.0

- The data set is the same as SQuAD v1.1 but with a no answer.
- We solve the no answer as the start and the end of the answer at [cls] token. For Prediction we compare $s_{null} = S.C + E.C$ (null answer span) with:
- $\hat{s} = \max_{j>i} (S.T_i + E.T_j)$ (answer span).
- We predict a none null answer if $\hat{s} > s_{null} + taw$.

Experiments
SQuAD v2.0



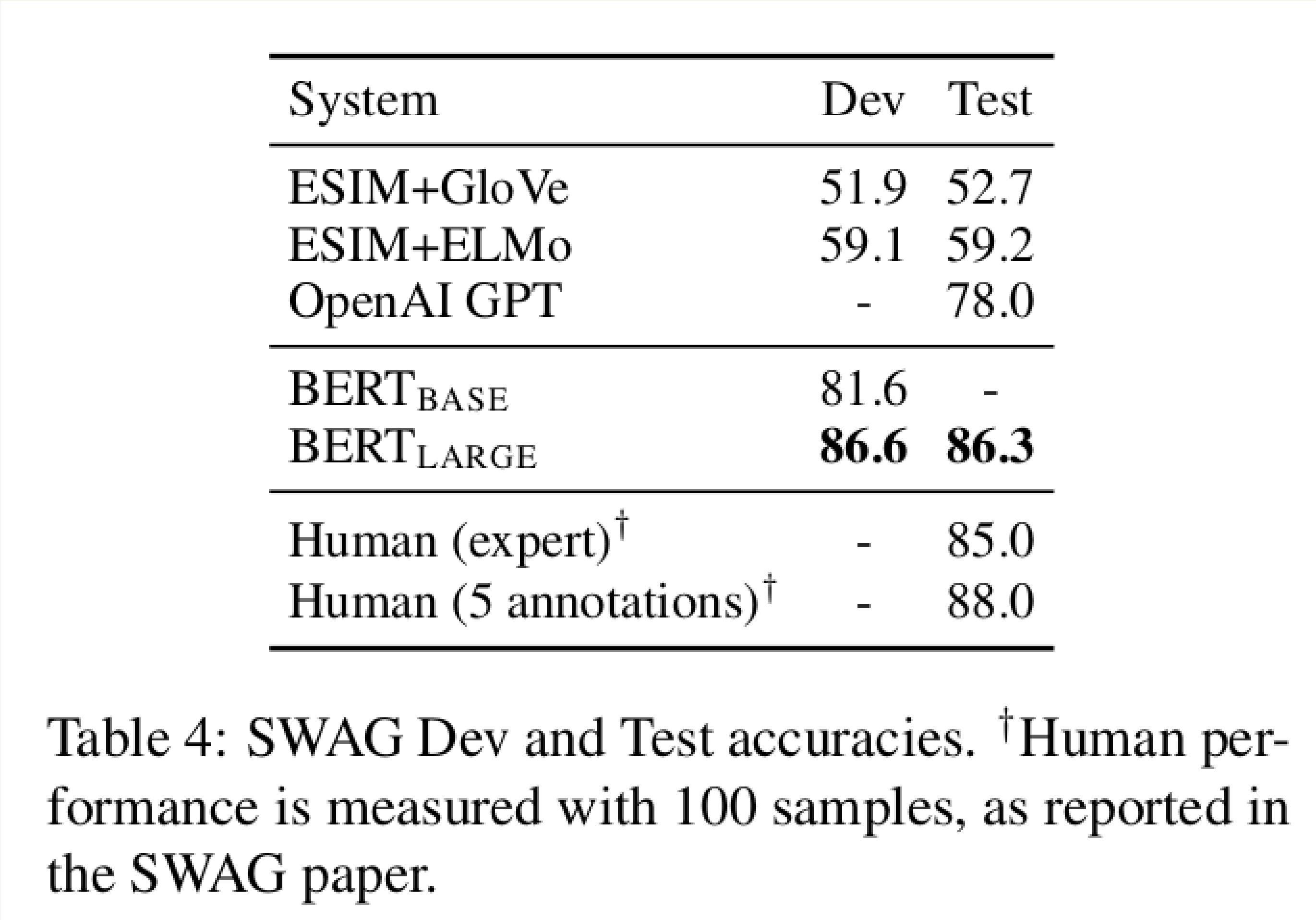
System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	86.3	89.0	86.9	89.5
#1 Single - MIR-MRC (F-Net)	-	-	74.8	78.0
#2 Single - nlnet	-	-	74.2	77.1
Published				
unet (Ensemble)	-	-	71.4	74.9
SLQA+ (Single)	-		71.4	74.4
Ours				
BERT$_{\text{LARGE}}$ (Single)	78.7	81.9	80.0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

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

Experiments
SWAG

- The Situations With Adversarial Generations (SWAG) dataset contains 113k sentence-pair completion examples that evaluate grounded common-sense inference.
- Given an input sentence the task is to find the most suitable answer among 4 choices.
- We fin-tune by constructing four output vector each one has input of:
  - *sentenceA is the input sentence*
  - sentecneB is one of the four choices
- The output of every choice is (C) (i.e: output of [cls]).
- The only task specific parameter we introduce is a vector ($V \in R^H$). We dot product V with C for every choice, Then we take softmax with the four choices.

Experiments SWAG



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

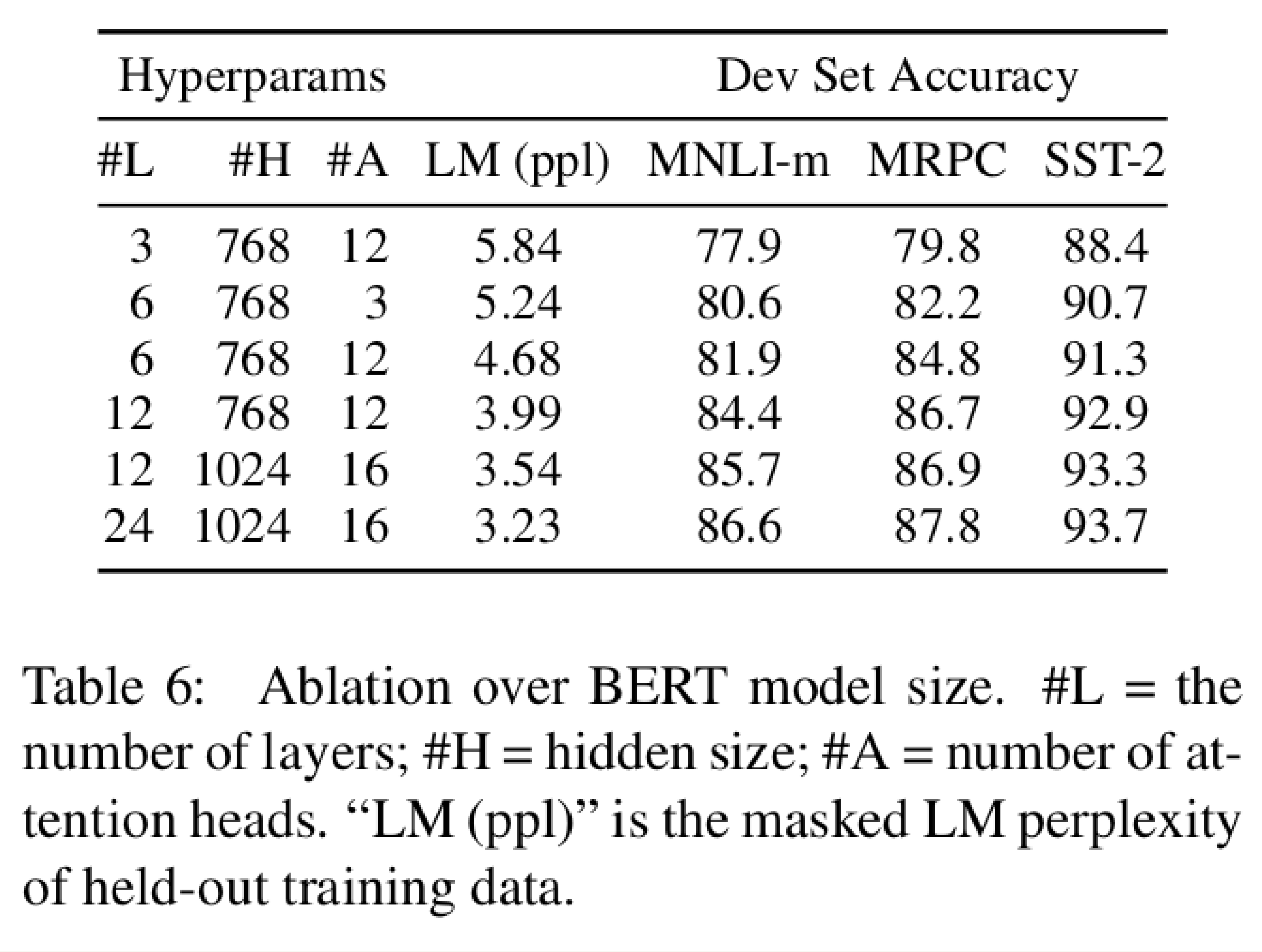
Table 4: SWAG Dev and Test accuracies. †Human performance is measured with 100 samples, as reported in the SWAG paper.

Experiments Hyper-parameters

Model	Adam Learning rate	Epochs	Batch size
GLUE	among(5e- 5, 4e− 5, 3e−5,and 2e−5)	3	32
SQuAD v1.1	5e-5	3	32
SQuAD v2.0	5e-5	2	48
SWAG	2e-5	3	16

Ablation Studies Effect of Model Size

- We used to know that using bigger model on small dateset will under-fit that model, so performance will degrade.
- But here we show that using larger model improves the accuracy even with small dataset like MPRC with 3,600 samples why ?
- this due to the pre-training process we used, we pre-trained our model with 3,300M wrod (our model has learned before this is only 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.

LM Comparisons
BERT vs Open AI GPT vs ELMo

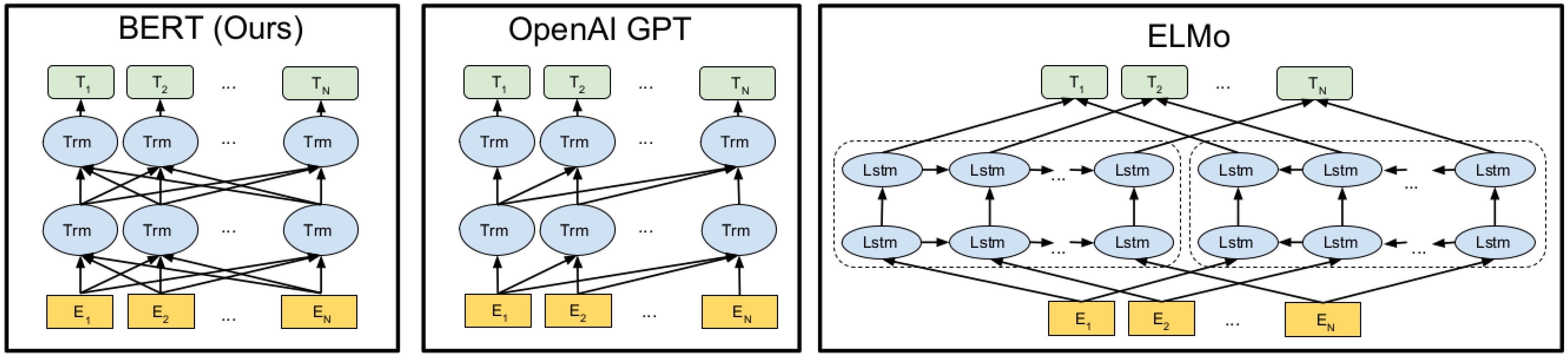
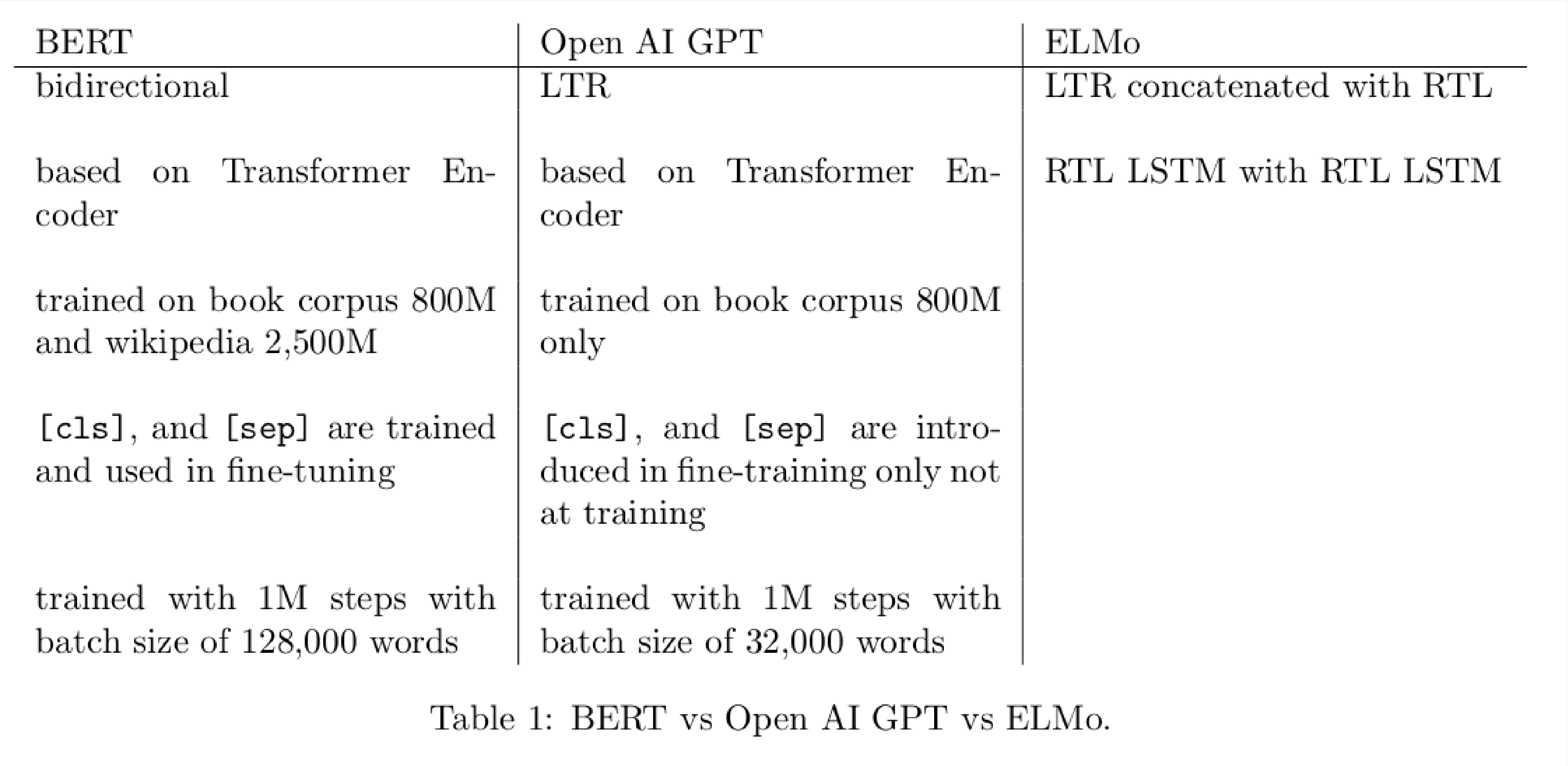


Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT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

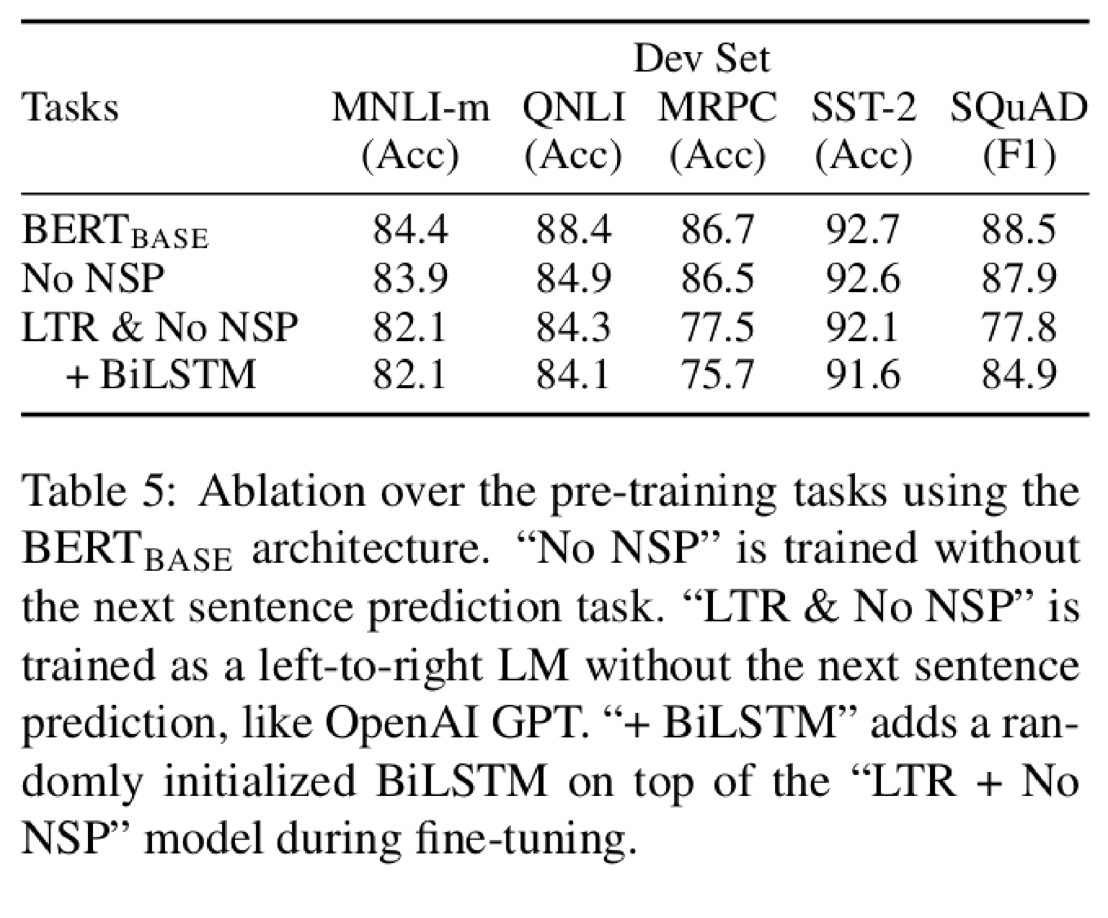
LM Comparisons
BERT vs Open AI GPT vs ELMo



BERT	Open AI GPT	ELMo
bidirectional	LTR	LTR concatenated with RTL
based on Transformer Encoder	based on Transformer Encoder	RTL LSTM with RTL LSTM
trained on book corpus 800M and wikipedia 2,500M	trained on book corpus 800M only	
[cls], and [sep] are trained and used in fine-tuning	[cls], and [sep] are introduced in fine-training only not at training	
trained with 1M steps with batch size of 128,000 words	trained with 1M steps with batch size of 32,000 words	

Table 1: BERT vs Open AI GPT vs ELMo.

ABLATION STUDIES EFFECT OF PRE-TRAINING TASKS



	Dev Set				
Tasks	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.

Ablation Studies Effect of Number of training steps

- BERT converges slower than LTR transformer like (I.e: Open AI GPT)

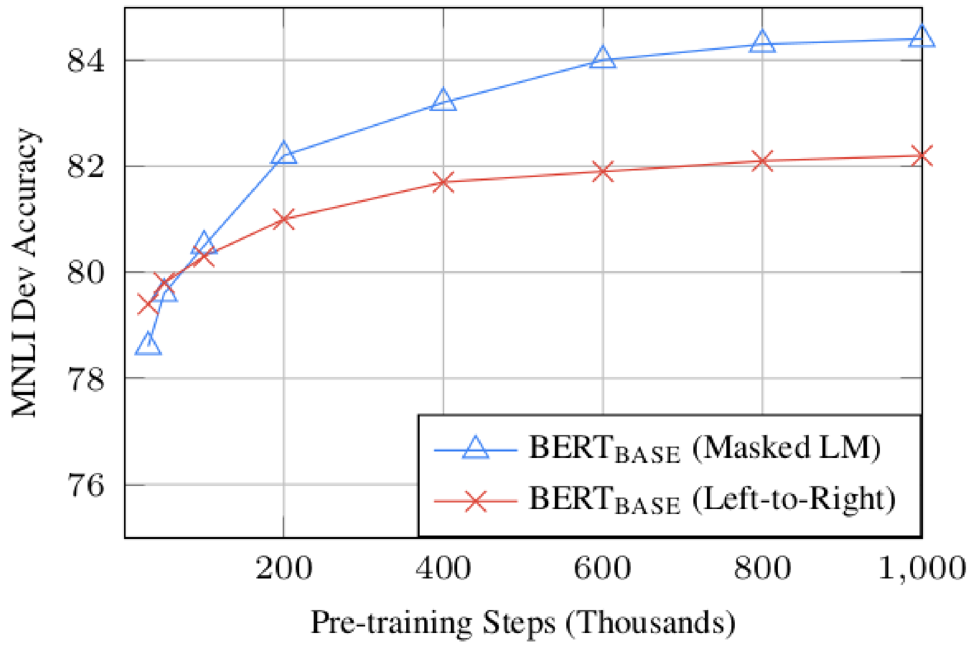


Figure 5: Ablation over number of training steps. This shows the MNLI accuracy after fine-tuning, starting from model parameters that have been pre-trained for k steps. The x-axis is the value of k.

Conclusion

We introduced a generalized model that can be used in many NLP tasks which proves the benefit of using transfer learning in language models, and bidirectional context representation.

Even if we fine-tune our model on small datasets we can get high performance thank to pre-training.

Resources to learn transformers and BERT

- Videos by Eng Ahmed Fathi very powerful. We built the background over it.
- "The Illustrated Transforms " blog post by Jay.
- The paper itself: Attention is all you need.
- Step by step explained Implementation of Transformers Notebook using Pytorch.
- "The Illustrated BERT " blog post by Jay.

QUESTIONS !

THANK YOU