


THIS IS AI4001

**GCR : t37g47w**

A man with a beard, wearing a colorful patterned shirt and a dark cap, is on the left side of the frame. On the right side, a red Elmo puppet is standing on a wooden bench. The background is a plain blue wall. Four speech bubbles are overlaid on the image, containing a conversation about word embeddings.

Hey ELMo, what's the embedding  
of the word "stick"?

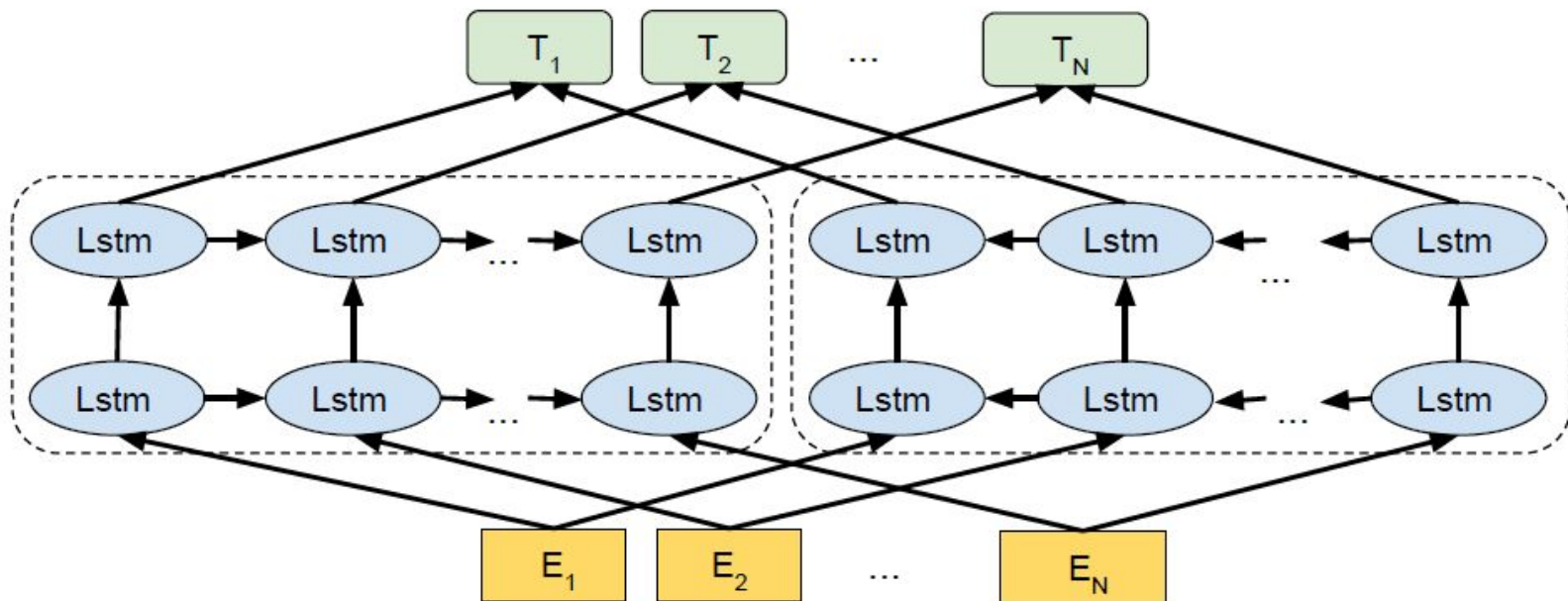
There are multiple possible  
embeddings! Use it in a sentence.

Oh, okay. Here:  
"Let's stick to improvisation in this  
skit"

Oh in that case, the embedding is:  
-0.02, -0.16, 0.12, -0.1 ....etc

# EMBEDDINGS FROM LANGUAGE MODELS

## ELMo

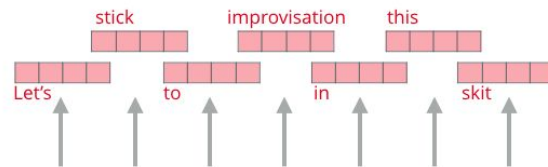


# ELMo

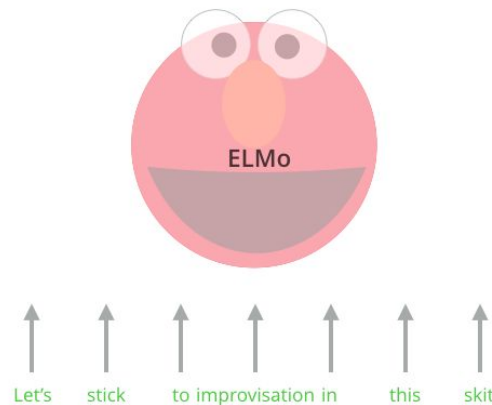
Instead of using a fixed embedding for each word, ELMo looks at the entire sentence before assigning each word in it an embedding.

It uses a bi-directional LSTM trained on a specific task to be able to create those embeddings.

ELMo  
Embeddings



Words to embed



Possible classes:  
All English words

0.1%	Aardvark
...	...
10%	Improvisation
...	...
0%	Zyzzzyva

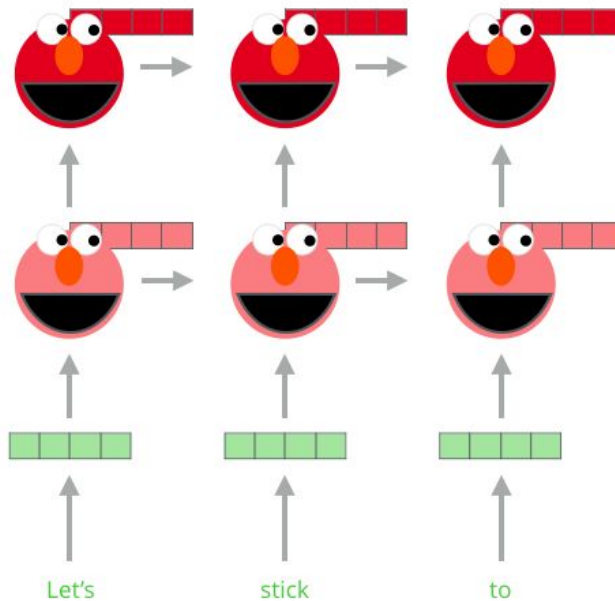
Output  
Layer

FFNN + Softmax

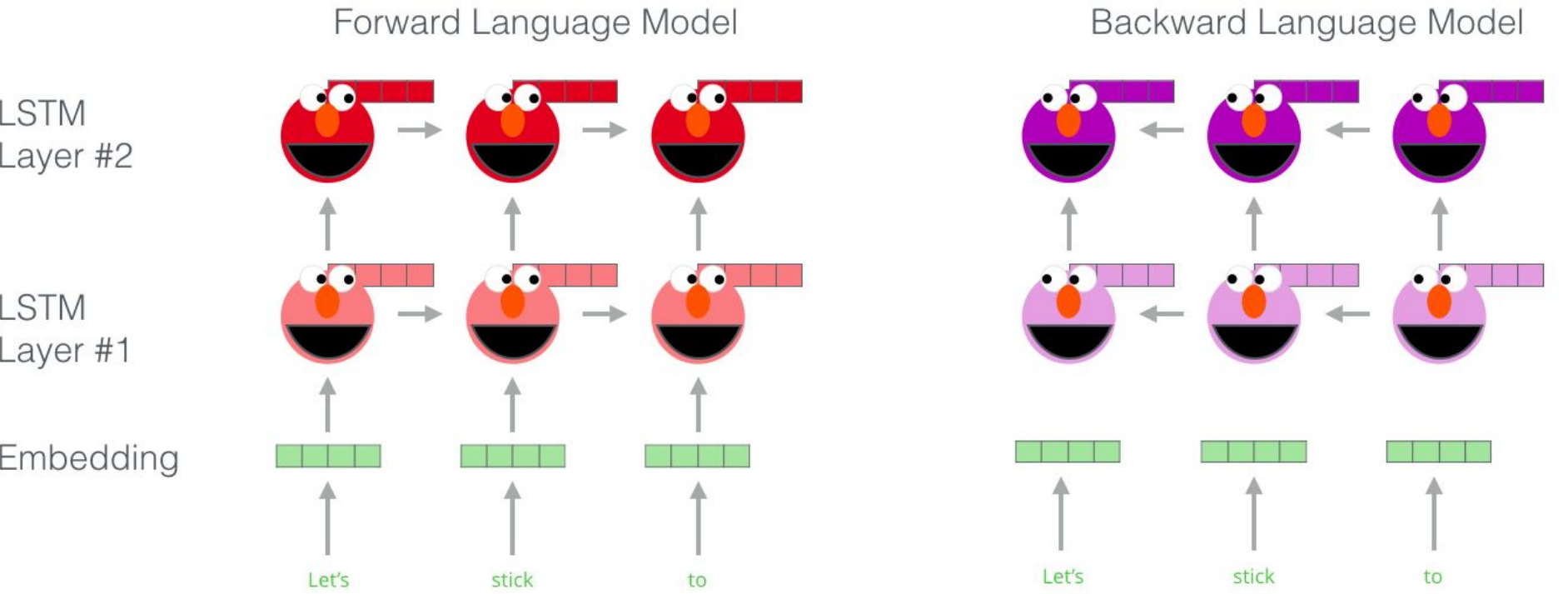
LSTM  
Layer #2

LSTM  
Layer #1

Embedding



# Embedding of “stick” in “Let’s stick to” - Step #1

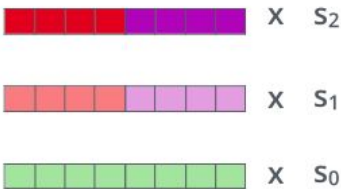


# Embedding of “stick” in “Let’s stick to” - Step #2

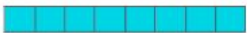
1- Concatenate hidden layers



2- Multiply each vector by a weight based on the task

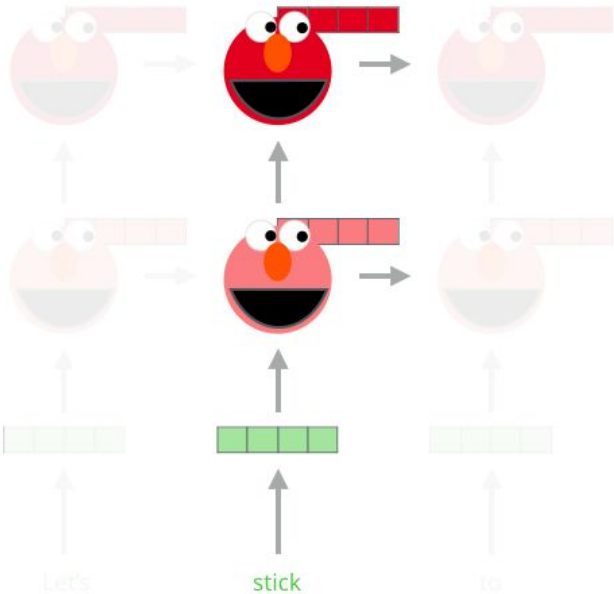


3- Sum the (now weighted) vectors

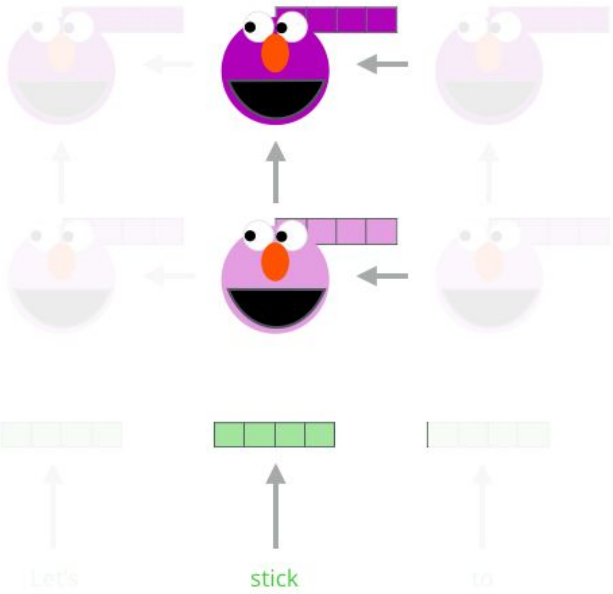


ELMo embedding of “stick” for this task in this context

Forward Language Model



Backward Language Model



# ELMO

- The architecture uses a character-level convolutional neural network (CNN) to represent words of a text string into raw word vectors
- These raw word vectors act as inputs to the first layer of biLM
- The forward pass contains information about a certain word and the context (other words) before that word
- The backward pass contains information about the word and the context after it
- This pair of information, from the forward and backward pass, forms the intermediate word vectors
- These intermediate word vectors are fed into the next layer of biLM
- The final representation (ELMo) is the weighted sum of the raw word vectors and the 2 intermediate word vectors





Google  
BERT



# Hi There! - I'm BERT

**Full Name:** Bidirectional Encoder Representations from Transformers

**Company:** Google

**Age:** 5 Years

**Work:** Understanding Natural Language Processing

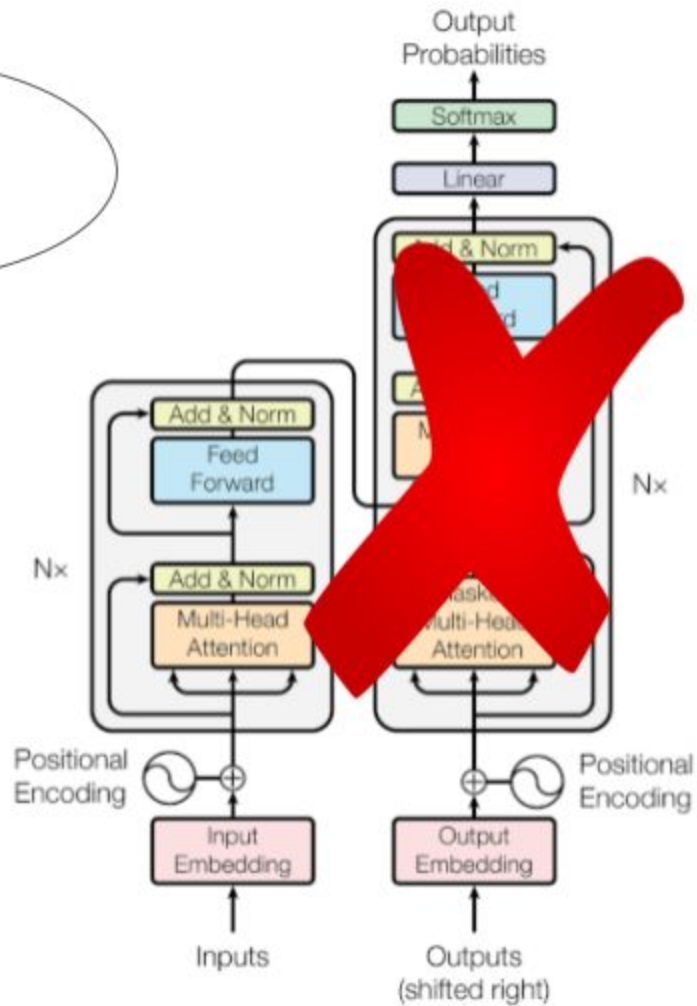
**Strength:** Deep Learning Algorithm

**Mind:** Complex

**Language:** English



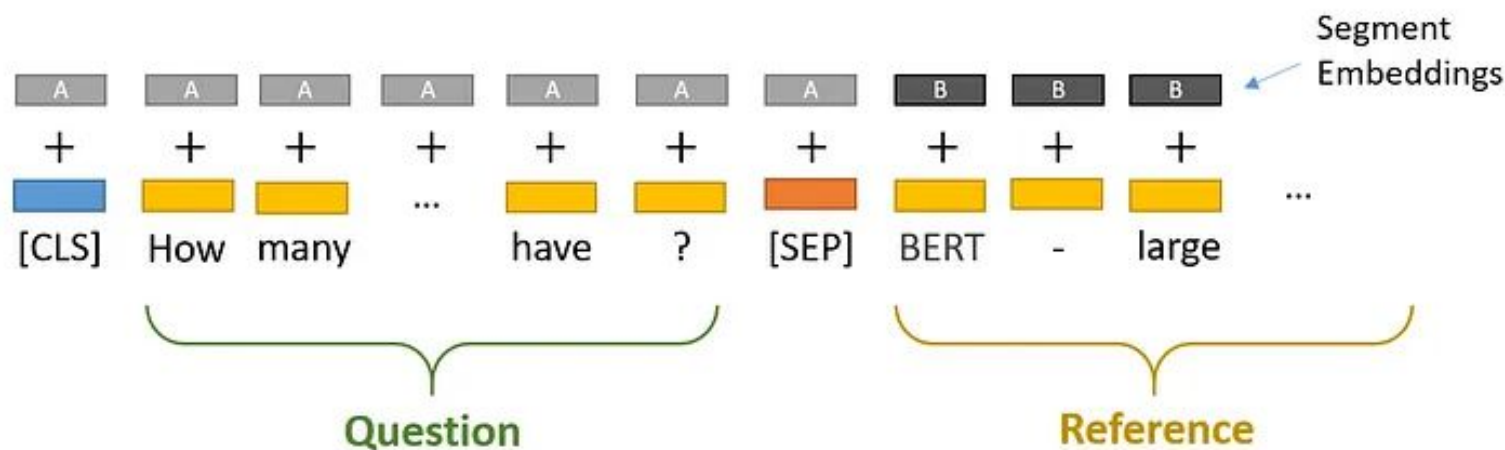
I only need the  
encoder part of  
the network



# BERT INSPIRATION

BERT builds on top of a number of clever ideas :

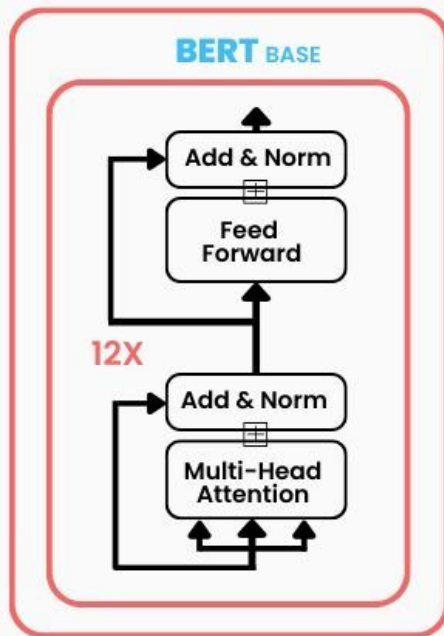
- Semi-supervised Sequence Learning (by Andrew Dai and Quoc Le)
- ELMo (by Matthew Peters and researchers from AI2 and UW CSE),
- ULMFiT (by fast.ai founder Jeremy Howard and Sebastian Ruder),
- OpenAI transformer (by OpenAI researchers Radford, Narasimhan, Salimans, and Sutskever)
- Transformer (Vaswani et al)



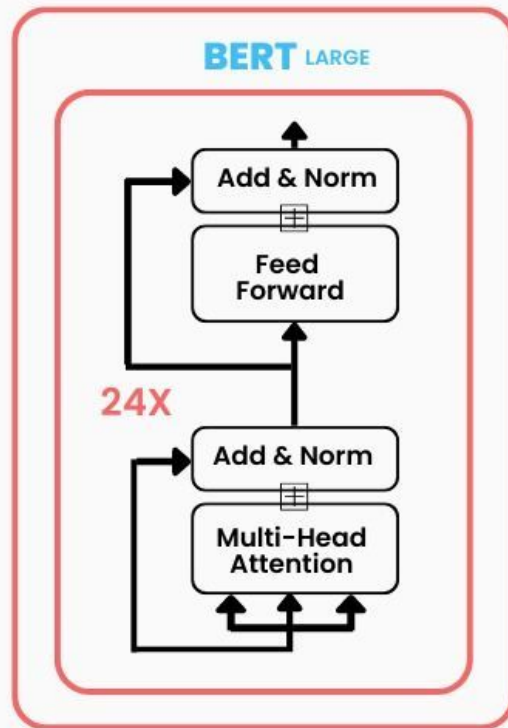
**Question:** How many parameters does BERT-large have?

**Reference Text:** BERT-large is really big... it has 24 layers and an embedding size of 1,024, for a total of 340M parameters! Altogether it is 1.34GB, so expect it to take a couple minutes to download to your Colab instance.

# BERT Size & Architecture



110M Parameters



340M Parameters

# ARCHITECTURE DETAILS

BERT-Base: 12-layer, 768-hidden-nodes, 12-attention-heads, 110M parameters

BERT-Large: 24-layer, 1024-hidden-nodes, 16-attention-heads, 340M parameters

Fun fact: BERT-Base was trained on 4 cloud TPUs for 4 days

BERT-Large was trained on 16 TPUs for 4 days!



# PRE-TRAINING & FINE-TUNING

BERT comes with an aim to learn 'language models' & no specific problem for the sake of generalization. For any specific problem in a certain language, minor tweaks are required.

Pre-training refers to the training where BERT learns the 'language model' of a particular language while Fine-tuning is the training done for any specific task (say Q&A or classification or any XYZ task) using Transfer Learning.

Hence, pre-training is something that has to be done once in a lifetime. And according to different problems, the same model can be fine-tuned. The major savings is on hardware resources & latency as the model doesn't need to learn all features from scratch for every problem but a few new ones according to the problem.



## 1 - Semi-supervised training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.

### Semi-supervised Learning Step

**Model:**



**Dataset:**



**Objective:**

Predict the masked word  
(language modeling)

## 2 - Supervised training on a specific task with a labeled dataset.

### Supervised Learning Step

**Model:**  
(pre-trained  
in step #1)

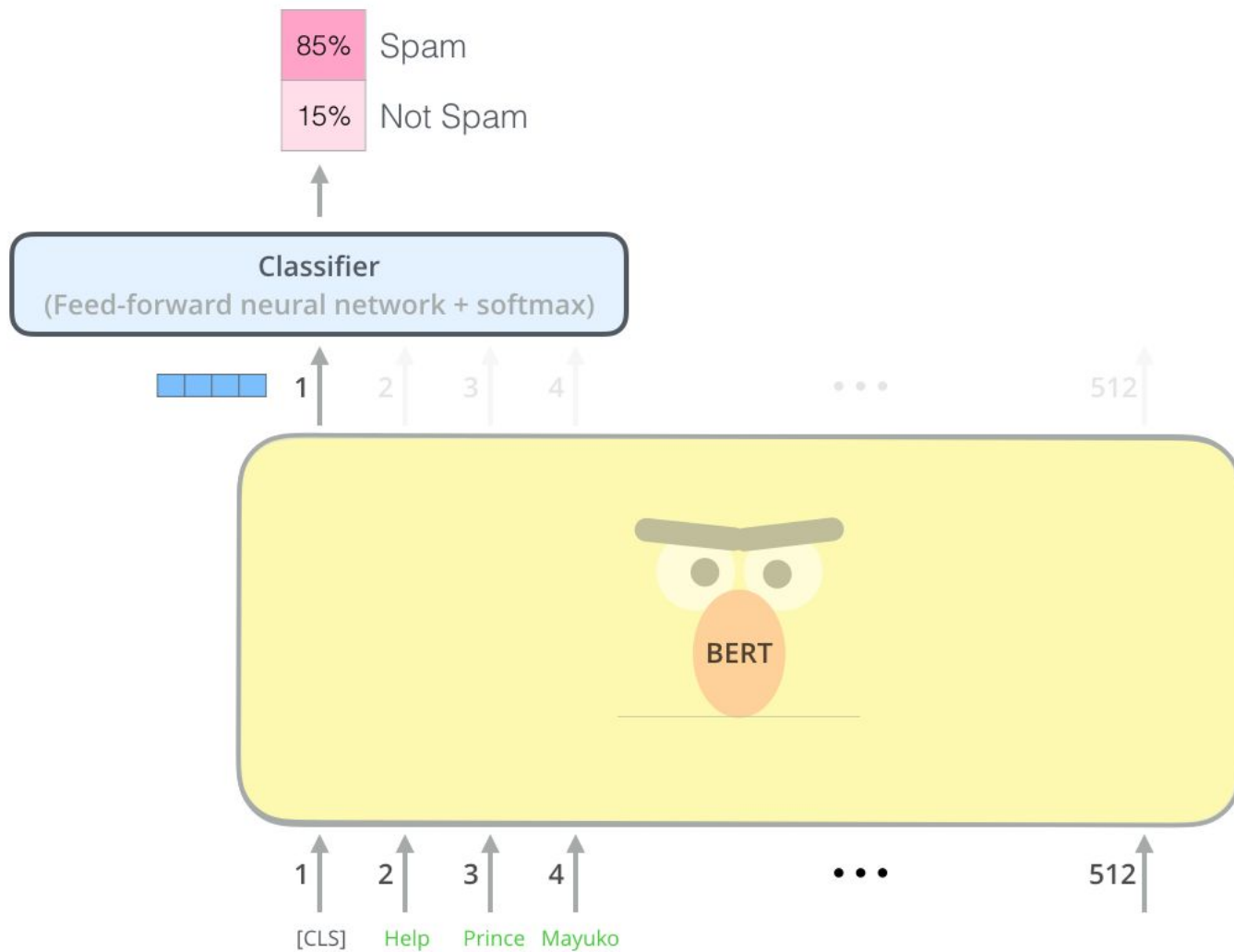


**Classifier**

75% Spam  
25% Not Spam

**Dataset:**

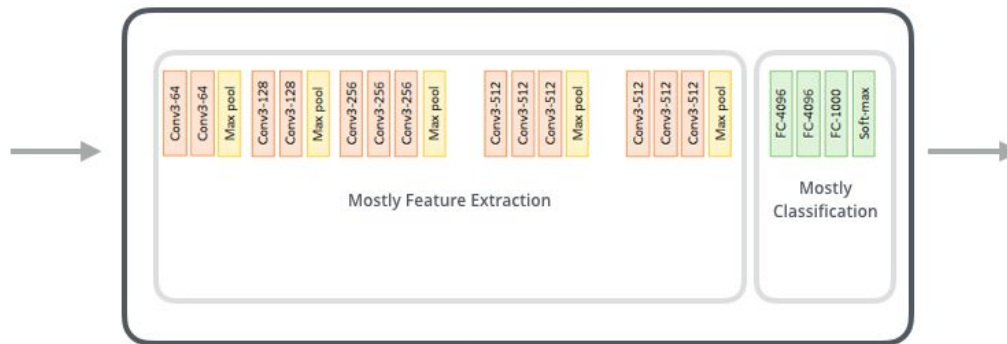
Email message	Class
Buy these pills	Spam
Win cash prizes	Spam
Dear Mr. Atreides, please find attached...	Not Spam



Input  
Features



VGG-16



Output  
Prediction

0.2%	Kit fox
0.1%	English setter
95%	Egyptian cat
1%	Great Dane
	...
0%	Hotdog

# BERT: FROM DECODERS TO ENCODERS

The openAI transformer gave us a fine-tunable pre-trained model based on the Transformer. But something went missing in this transition from LSTMs to Transformers.

ELMo's language model was bi-directional, but the openAI transformer only trains a forward language model.

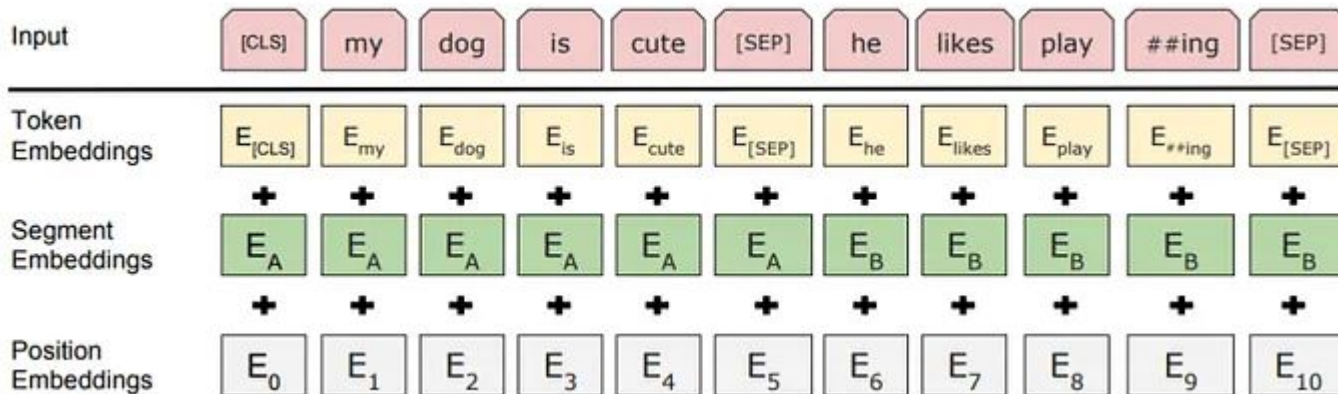
Could we build a transformer-based model whose language model looks both forward and backwards (in the technical jargon - “is conditioned on both left and right context”)?

# EMBEDDINGS

Token embeddings: A [CLS] token is added to the input word tokens at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.

Segment embeddings: A marker indicating Sentence A or Sentence B is added to each token. This allows the encoder to distinguish between sentences.

Positional embeddings: A positional embedding is added to each token to indicate its position in the sentence.



# 1. MASKED LM (MLM)

The idea here is “simple”: Randomly mask out 15% of the words in the input – replacing them with a [MASK] token – run the entire sequence through the BERT attention based encoder and then predict only the masked words, based on the context provided by the other non-masked words in the sequence.

The model only tries to predict when the [MASK] token is present in the input, while we want the model to try to predict the correct tokens regardless of what token is present in the input. To deal with this issue, out of the 15% of the tokens selected for masking:

- 80% of the tokens are actually replaced with the token [MASK].
- 10% of the time tokens are replaced with a random token.
- 10% of the time tokens are left unchanged.

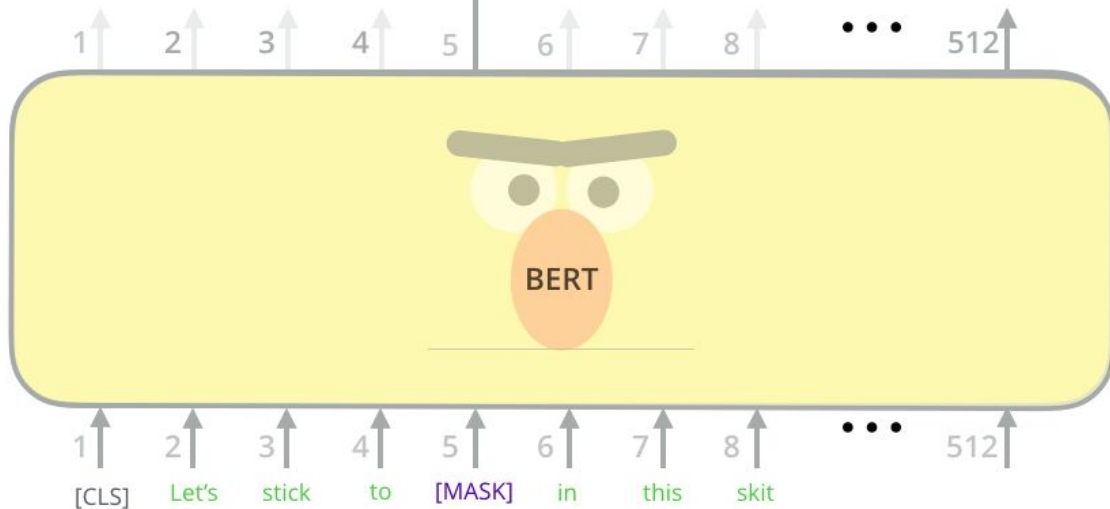
While training the BERT loss function considers only the prediction of the masked tokens and ignores the prediction of the non-masked ones. This results in a model that converges much more slowly than left-to-right or right-to-left models.

Use the output of the masked word's position to predict the masked word

Possible classes:  
All English words

0.1%	Aardvark
...	...
10%	Improvisation
...	...
0%	Zyzyva

FFNN + Softmax



Randomly mask  
15% of tokens

Input

[CLS] Let's stick to improvisation in this skit

# MASKED LM

Random tokens are masked

- I am [MASKED] boy
- She loved [MASKED] with chocolate [MASKED]
- She is at[MASKED]. I just texted me.



# MASKED LM

Replace any token with any random token from vocabulary (10%)

- He is 'cake' best friend
- 'Bus' love each other

No alterations in the remaining 10% of the input sequence

- He is my best friend
- They love each other

# TRANSFORMER VS BERT

Transformer: I am a \_\_\_

BERT: I am a \_\_\_. I like playing Cricket

# TRANSFORMER VS BERT

In transformers we always predict the end token of the sequence. The transformers model is trained in a unidirectional way i.e. either left-right or right-left as the tokens after the token which has to be predicted are always masked in the decoder.

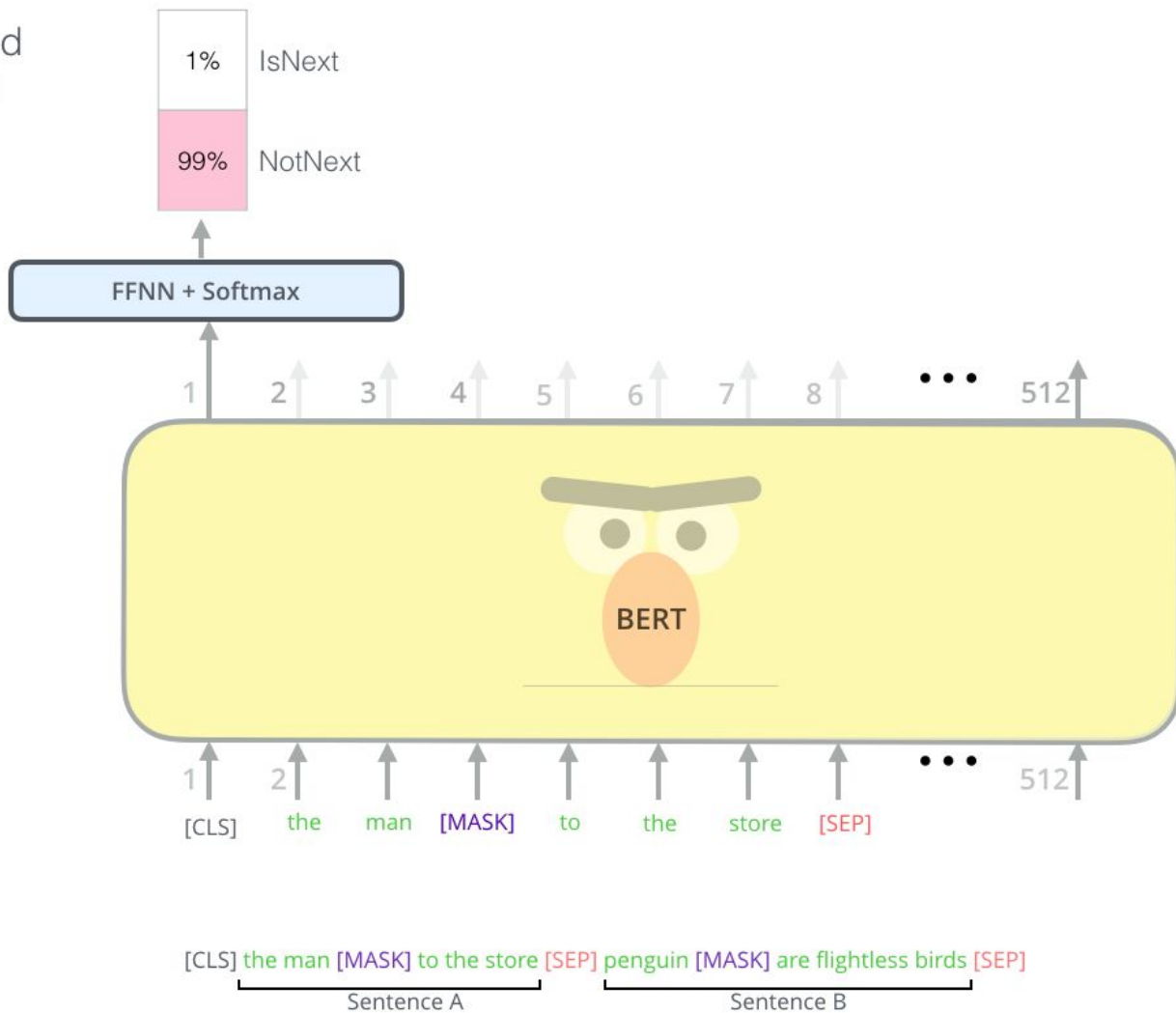
In BERT random token is masked & has to be predicted, the rest of the tokens are available & hence a bidirectional training happens i.e. the model can take context from both sides together for this token prediction.

## 2. NEXT SENTENCE PREDICTION (NSP)

In order to understand relationship between two sentences, BERT training process also uses next sentence prediction. During training the model gets as input pairs of sentences and it learns to predict if the second sentence is the next sentence in the original text as well.

- 50% of the time the second sentence comes after the first one.
- 50% of the time it is a random sentence from the full corpus.

Predict likelihood  
that sentence B  
belongs after  
sentence A



## 2. NEXT SENTENCE PREDICTION (NSP)

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

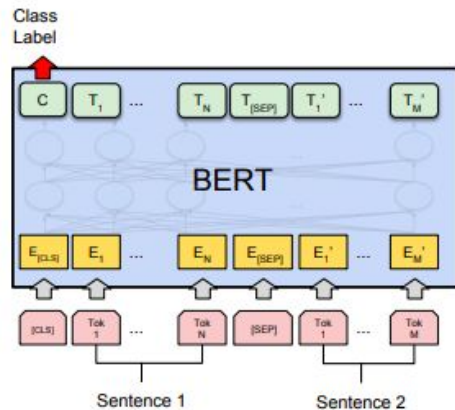
he bought a gallon [MASK] milk [SEP]

Label = IsNext

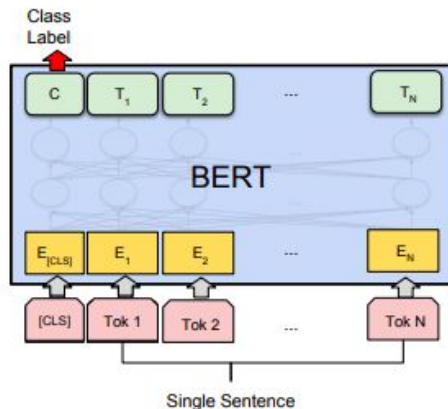
Input = [CLS] the man [MASK] to the store [SEP]

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

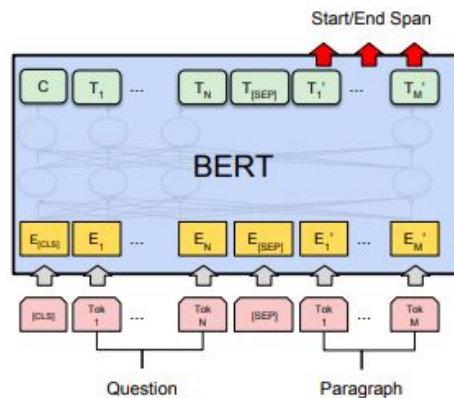
Label = NotNext



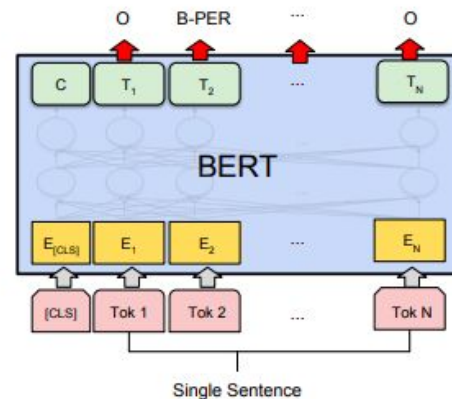
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



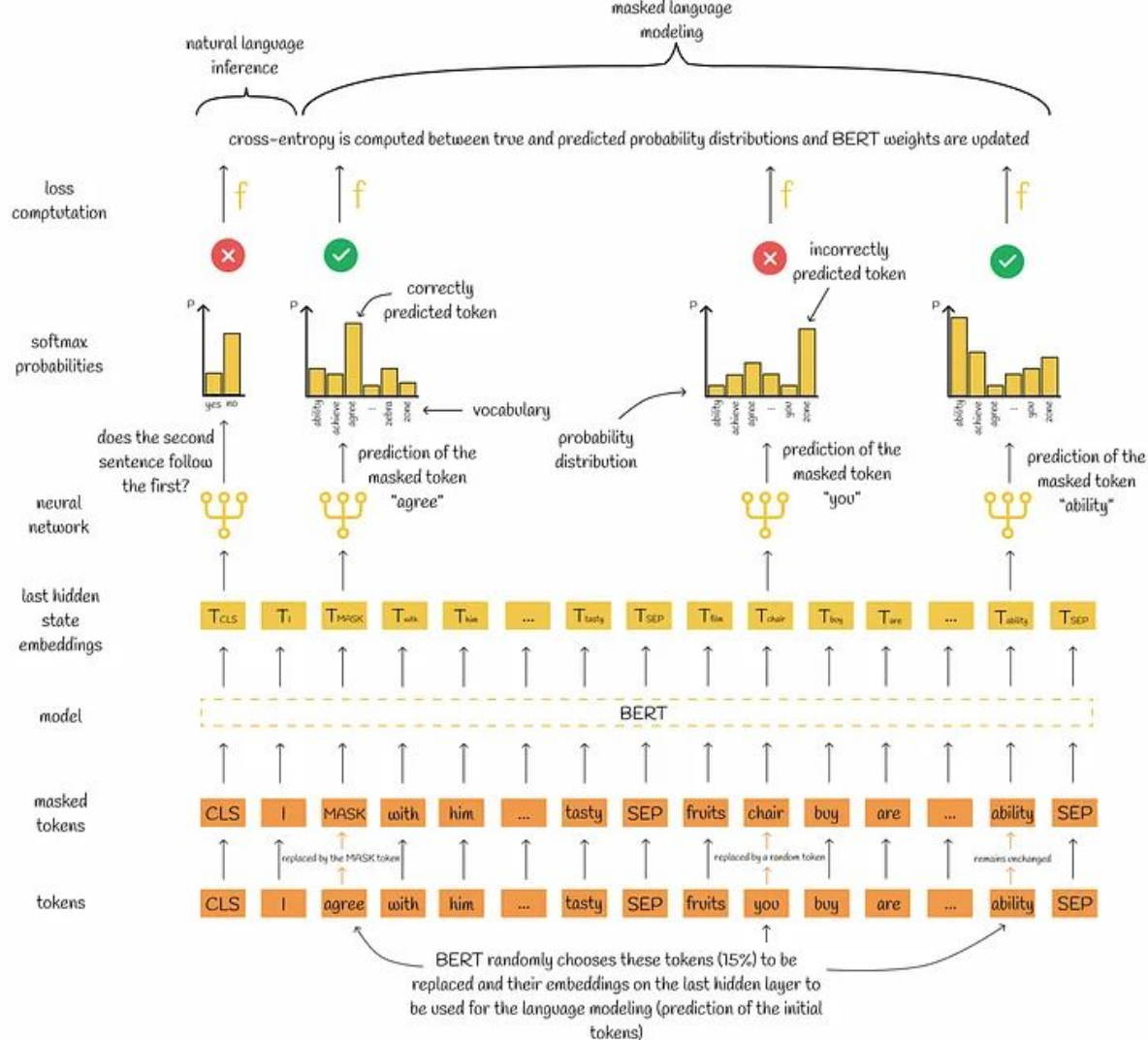
(b) Single Sentence Classification Tasks:  
SST-2, CoLA



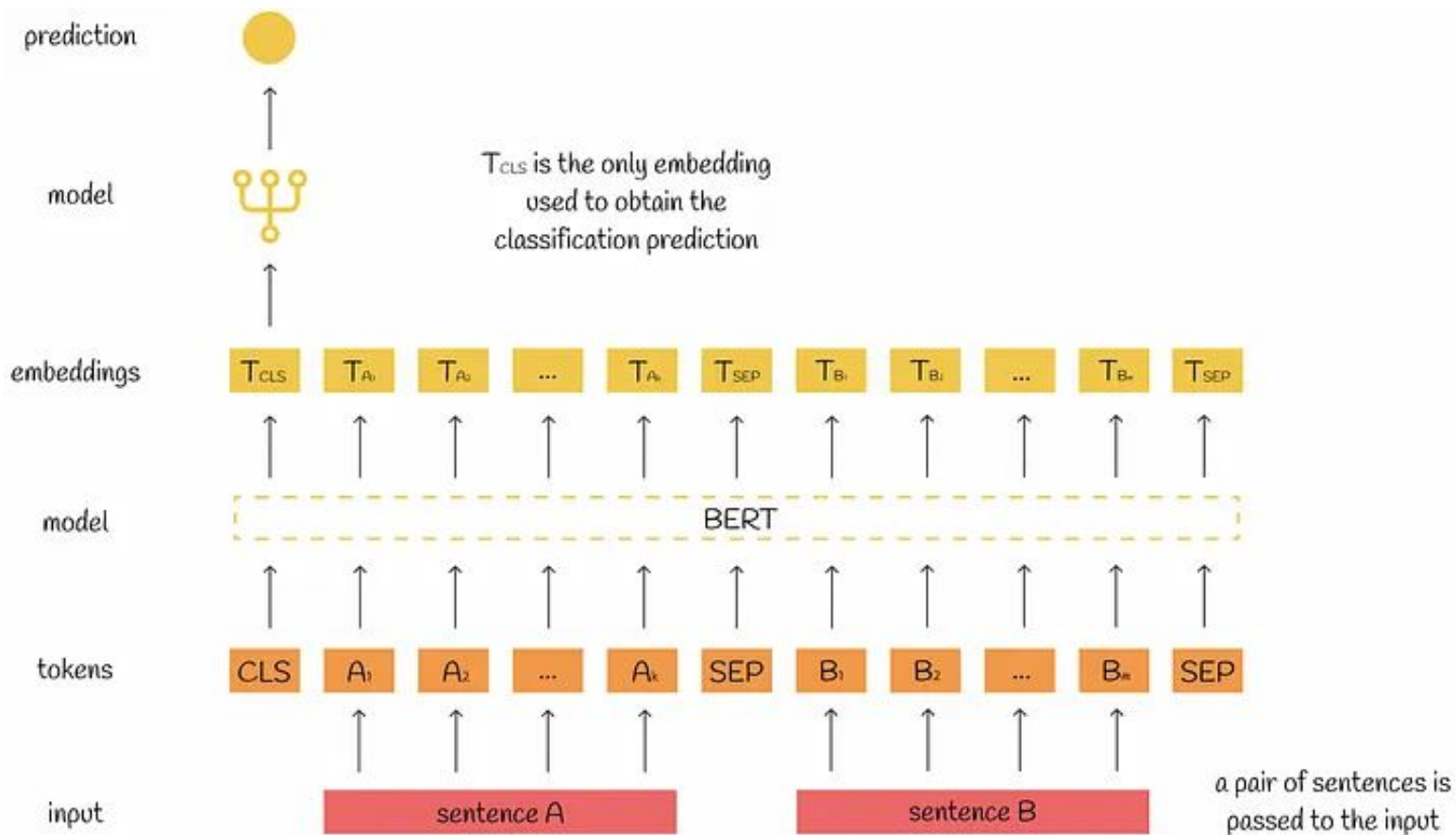
(c) Question Answering Tasks:  
SQuAD v1.1

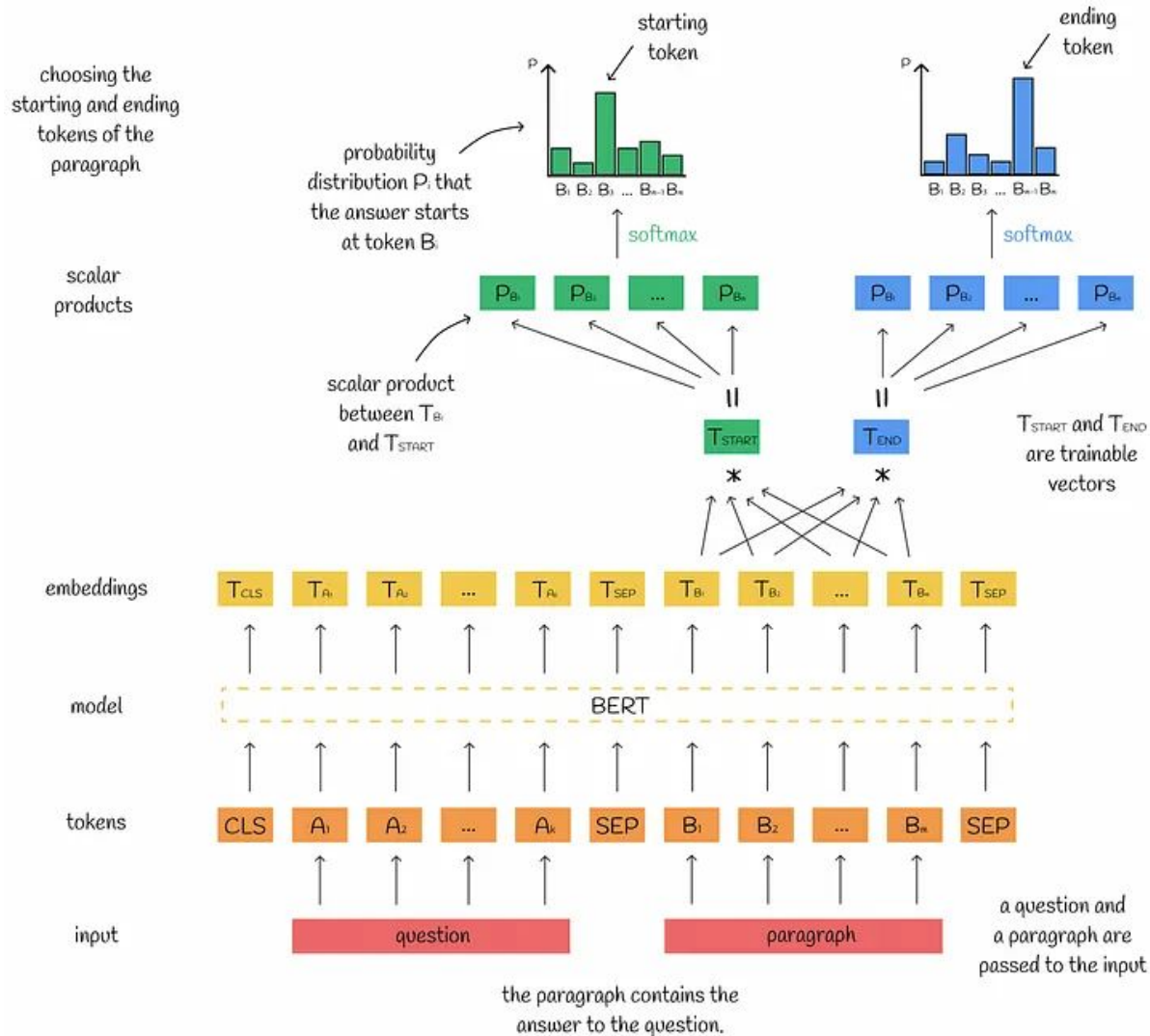


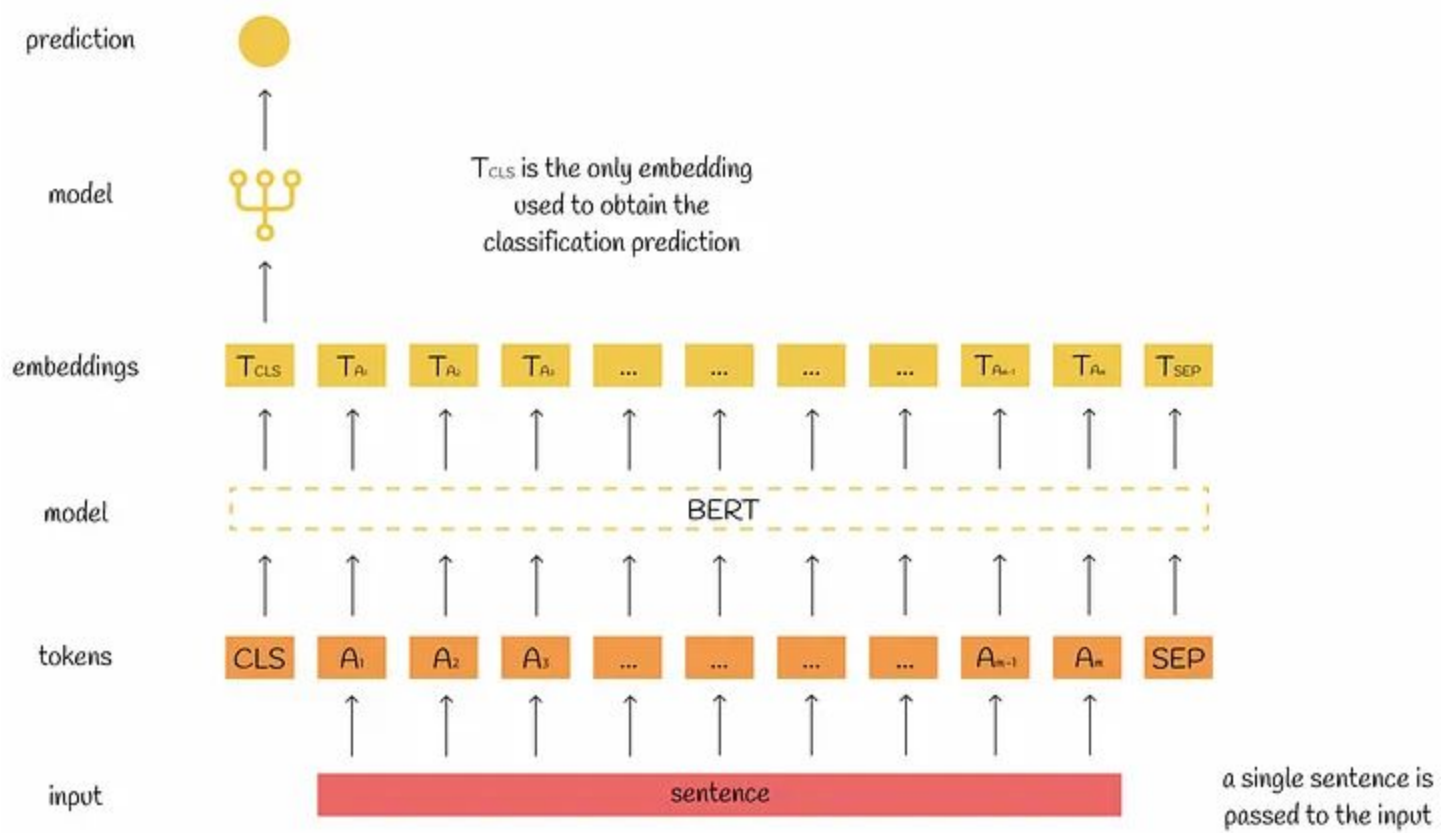
(d) Single Sentence Tagging Tasks:  
CoNLL-2003 NER

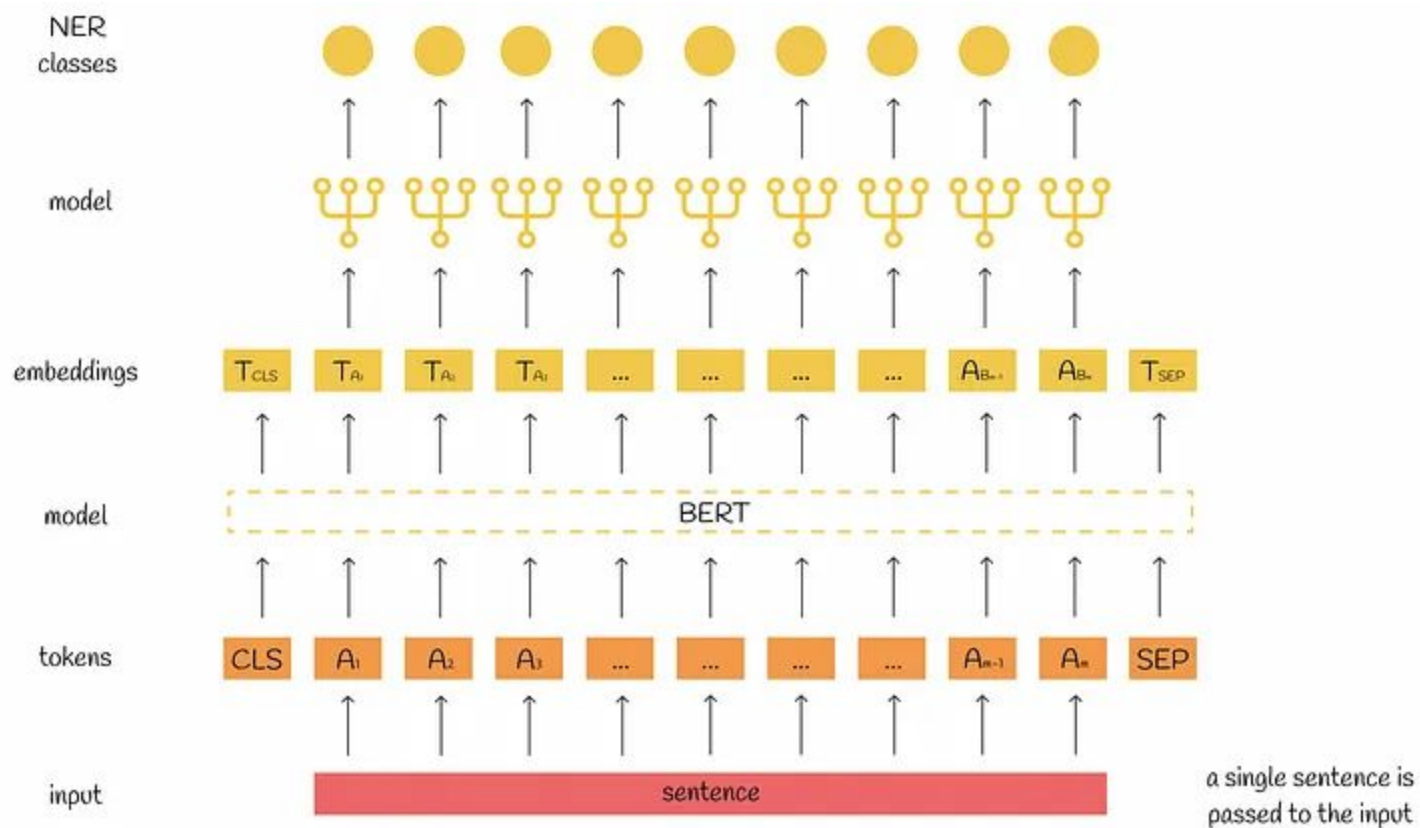












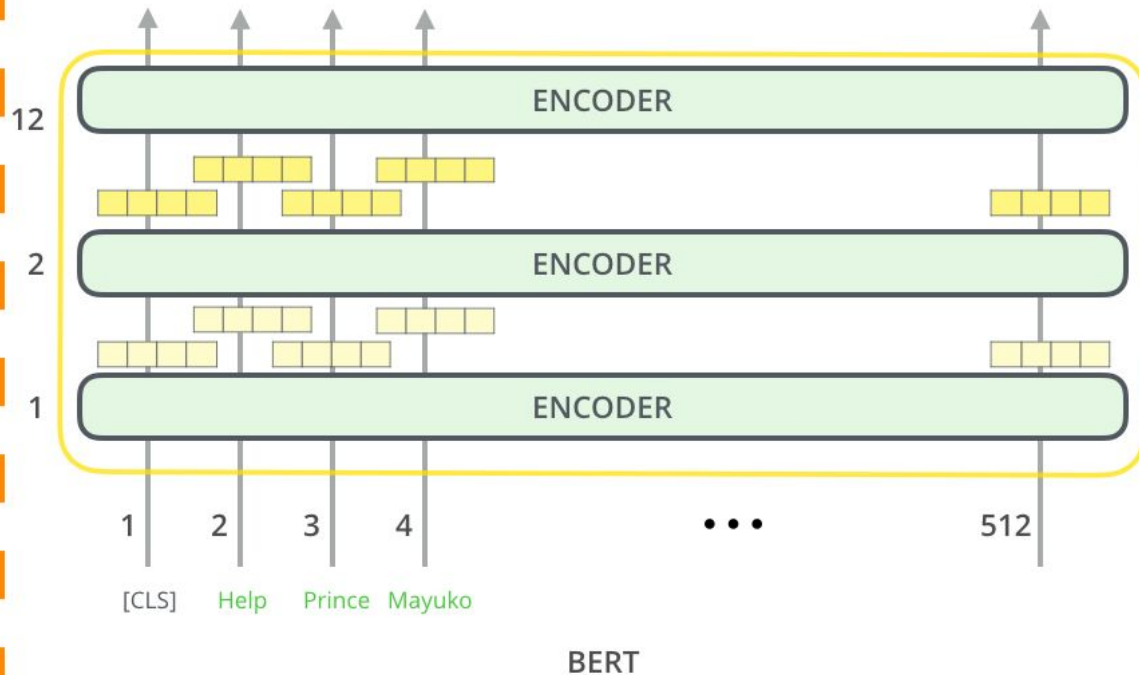
# BERT FOR FEATURE EXTRACTION

The fine-tuning approach isn't the only way to use BERT.

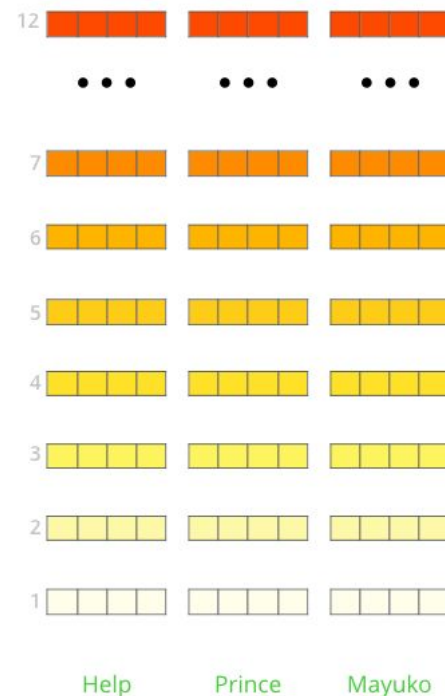
Just like ELMo, you can use the pre-trained BERT to create contextualized word embeddings.

Then you can feed these embeddings to your existing model – a process the paper shows yield results.

## Generate Contextualized Embeddings



The output of each encoder layer along each token's path can be used as a feature representing that token.

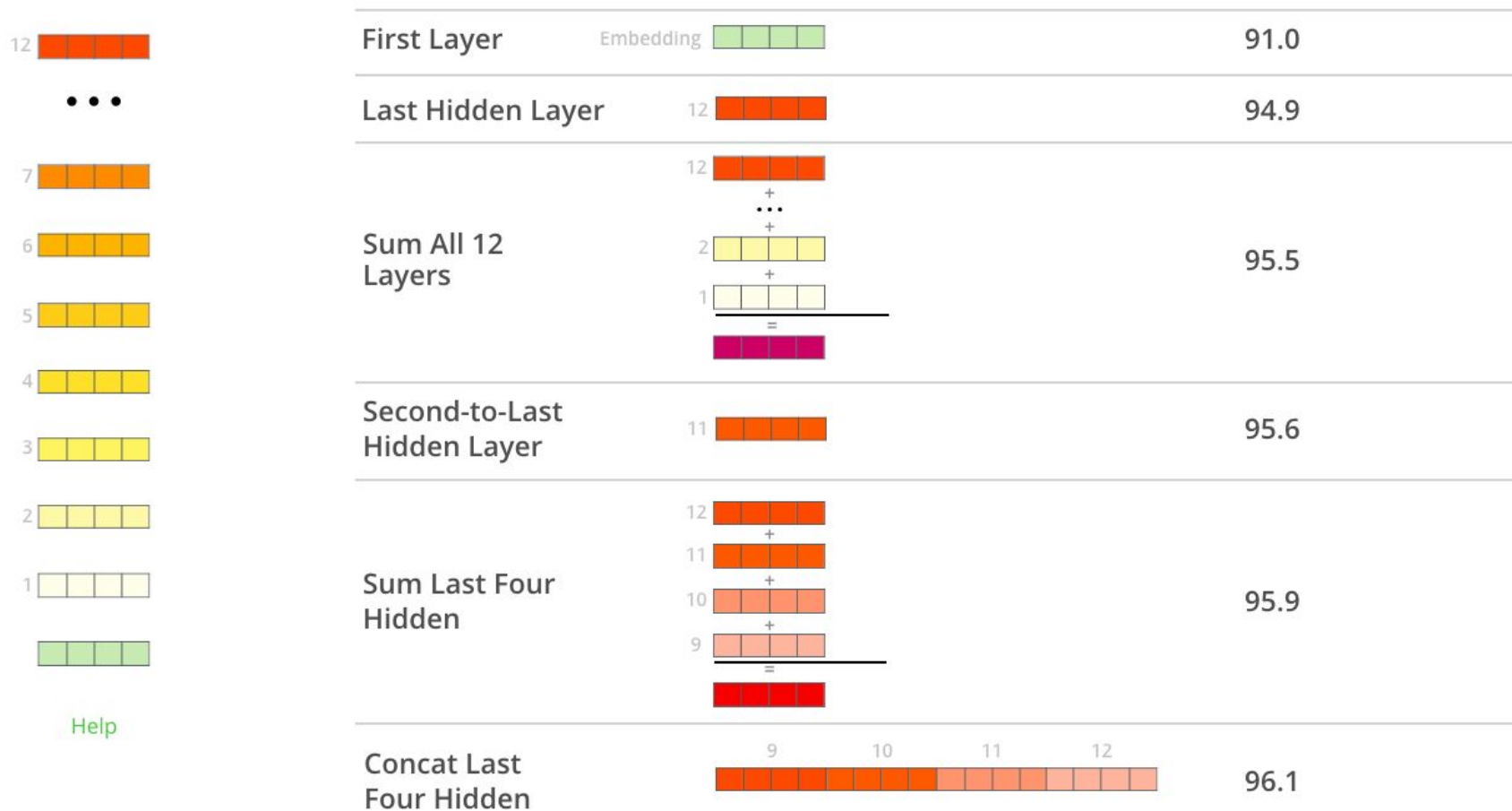


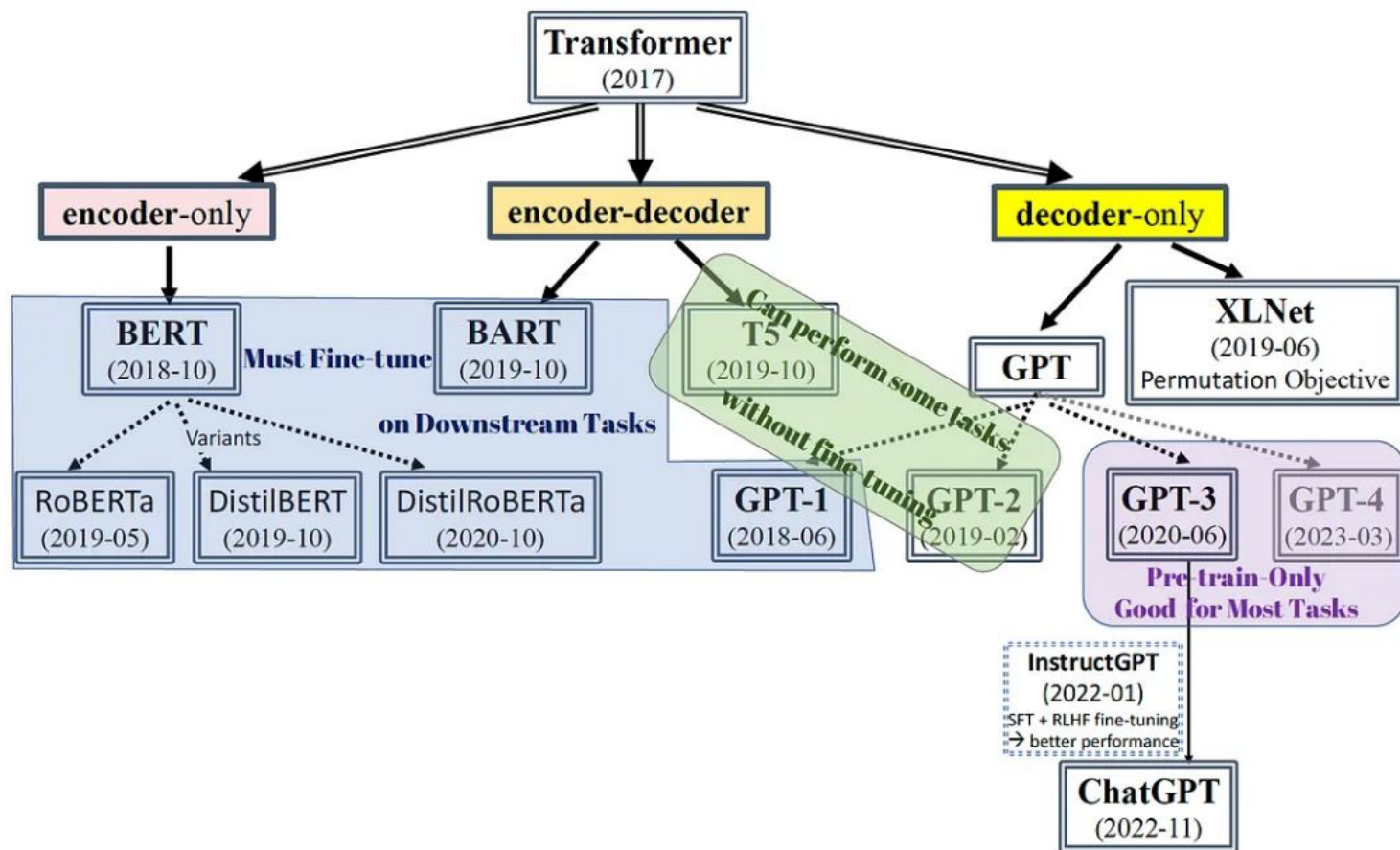
But which one should we use?

What is the best contextualized embedding for “Help” in that context?

For named-entity recognition task CoNLL-2003 NER

Dev F1 Score







# Semi-supervised Sequence Learning

context2Vec  
Pre-trained seq2seq



**ELMo**

**ULMFiT**

Multi-lingual

**MultiFiT**

Cross-lingual

Multi-task

**XLM**  
**UDify**

**MT-DNN**

Knowledge distillation

**MT-DNN<sub>KD</sub>**

**MASS**  
**UniLM**

Span prediction  
Remove NSP

Longer time  
Remove NSP  
More data

**SpanBERT**

**RoBERTa**

**XLNet**

Permutation LM  
Transformer-XL  
More data



**BERT**

Transformer

Bidirectional LM

+ Knowledge Graph

Transformer-XL  
More data



**ERNIE**  
**(Tsinghua)**

Neural entity linker

**KnowBert**

**GPT**

Larger model  
More data

**GPT-2**

Defense



**Grover**

Whole Word Masking

**VideoBERT**  
**CBT**

**ViLBERT**

**VisualBERT**

**B2T2**

**Unicoder-VL**

**LXMERT**

**VL-BERT**

**UNITER**



**ERNIE (Baidu)**  
**BERT-wwm**

# ISSUES WITH BERT

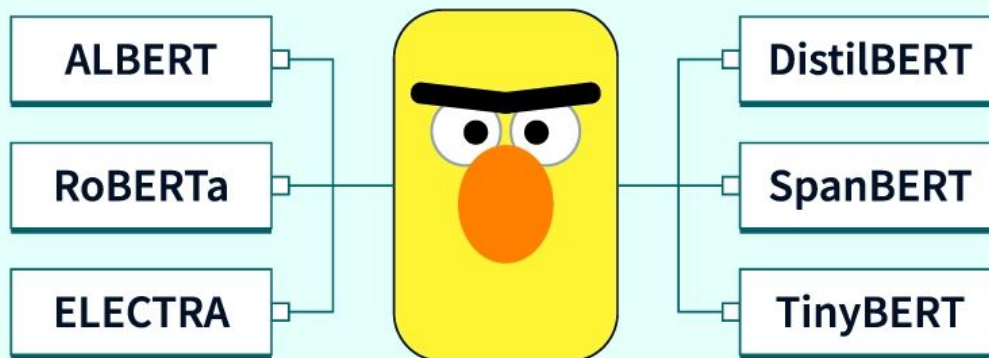
Consumes a lot of time & resources to train

Humongous in size due to 110 million parameters

High inference time

BERT is considered to be 'less efficiently' trained by some researchers. They think some better training techniques could have been used.

## Exploring Variants of BERT





# **ALBERT:** A Lite BERT

Parameters  
are:

## BERT

Unique

Encoder 12

...

Unique

Encoder 3

Unique

Encoder 2

Unique

Encoder 1

Parameters  
are:

## ALBERT

Identical

Encoder 12

...

Encoder 3

Encoder 2

Encoder 1

# ALBERT

ALBERT achieves performance increase by using several techniques such as cross-layer parameter sharing to share parameters across different layers of the model, reducing the number of parameters that need to be trained. It also uses a technique called factorized embedding parameterization to reduce the number of parameters in the embedding layer, which is the first layer of the model.

Using the above techniques, ALBERT brings down the total parameters from about 110 million to around 12 million (~1/10th of the original BERT Size) & hence better for real-world deployment & speedy by following the below strategies.

# ALBERT

By matrix factorization. The idea is simple. In Bert what we do:

Input Embedding (3000x768) → Encoder → Output(3000x768)

Now, in ALBERT, we would be doing instead

IE (3000xN) X Temp(Nx768) → Encoder → Output(3000x768)

# **Robustly optimized BERT approach**



# ROBERT

One key difference between RoBERTa and BERT is that RoBERTa is trained on a much larger dataset, which includes more than 160GB of text data, whereas BERT was originally trained on about 16GB of text data.

It mask different tokens in the input sequence for every epoch trained. Hence, the input sequence doesn't remain constant for all epochs

Sentence	Tokens
Sentence 1	[CLS], we, [MASK], at, the, airport, in, [MASK], [SEP]
Sentence 2	[CLS], we, arrived, [MASK], the, [MASK], in, time, [SEP]
⋮	⋮
Sentence 10	[CLS], we, arrived, at, [MASK], airport, [MASK], time, [SEP]

Epoch	Sentence
Epoch 1	Sentence 1
Epoch 2	Sentence 2
⋮	⋮
Epoch 10	Sentence 10
Epoch 11	Sentence 1
Epoch 12	Sentence 2
⋮	⋮
Epoch 40	Sentence 10

→ 100 epochs, 10 sentences

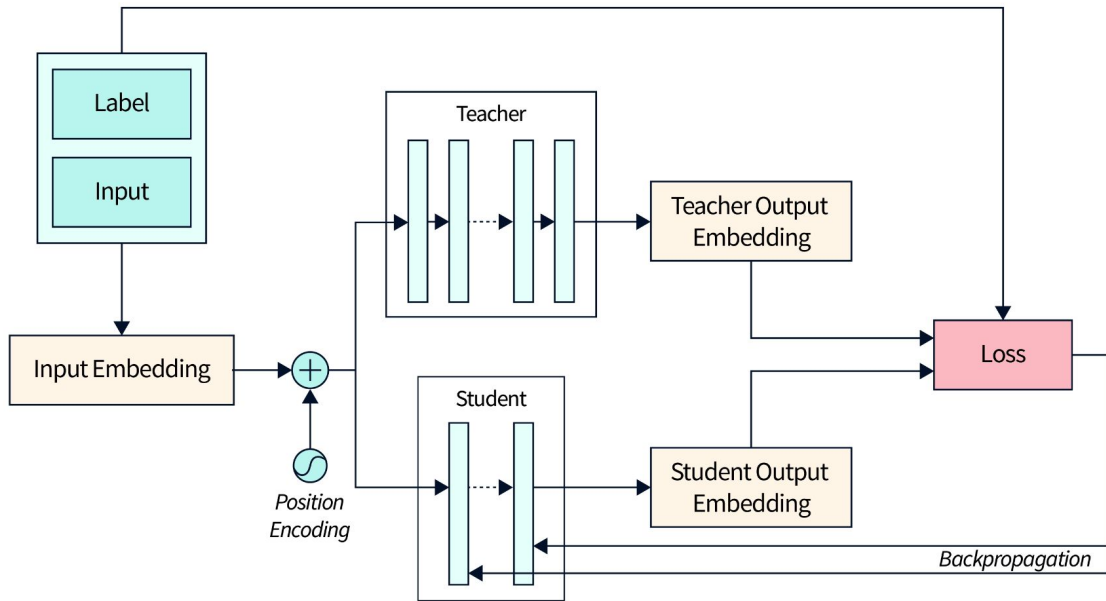
# DISTILBERT

DistilBERT uses Knowledge Distillation.



# DISTILBERT

The idea behind knowledge distillation is that the larger, pre-trained model has learned to solve the task in a general and robust way, and the smaller model can learn to mimic its behavior and achieve similar performance. This makes the smaller model more efficient and easier to deploy in certain situations, such as on mobile devices.



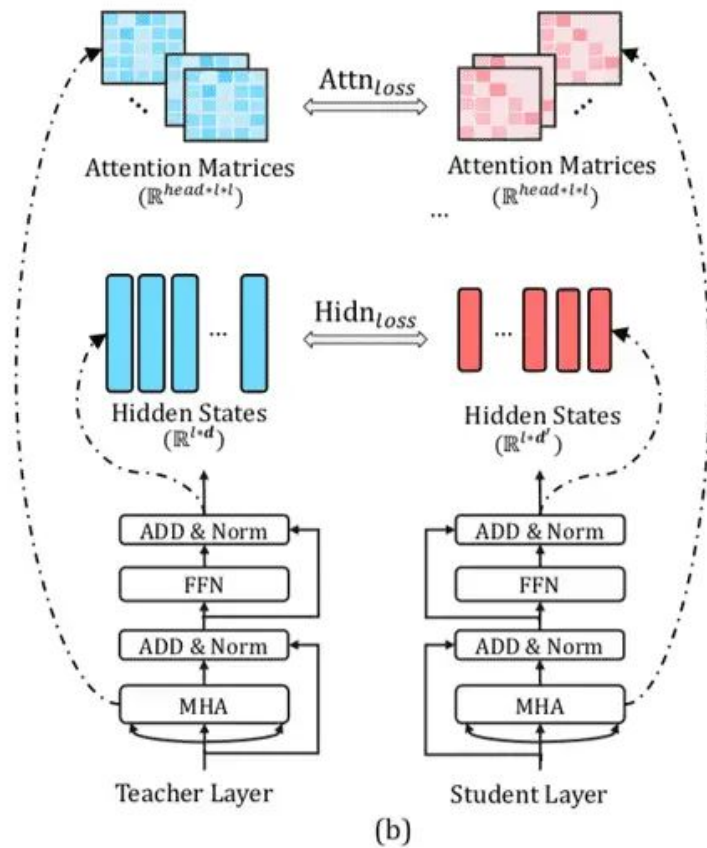
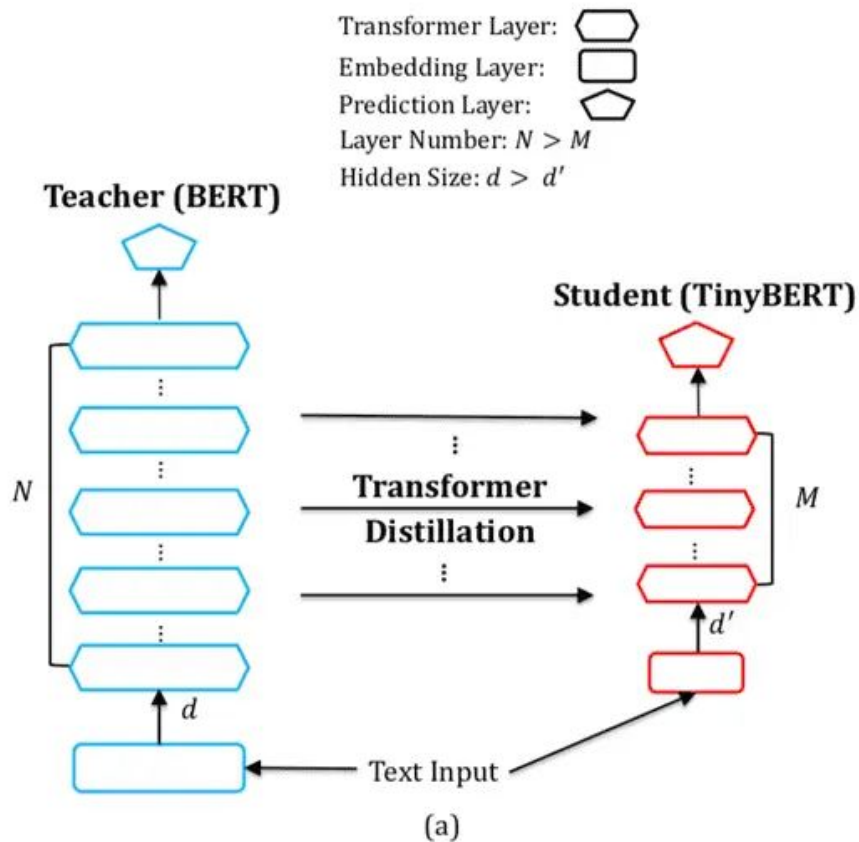
# DISTILBERT

One key difference between DistilBERT and BERT is that DistilBERT is a smaller and more efficient model. It has 40% fewer parameters than BERT, with only 66 million parameters, which makes it faster to train and easier to deploy in resource-constrained situations.

# TINYBERT

TinyBERT is a variant of BERT that uses a technique called knowledge distillation.

In addition to using the Teacher's output embedding (predictions), TinyBERT allows the Student to learn from each layer in the Teacher, such as the attention module and other layers.





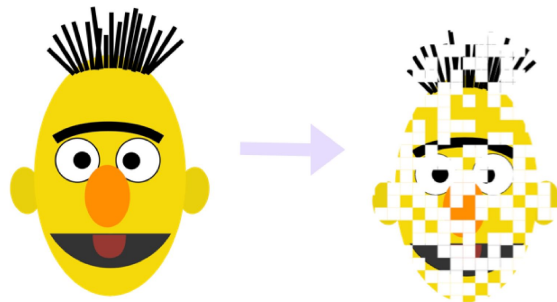
# TINY BERT

In the knowledge distillation process used by TinyBERT, the smaller "Student" model learns from the larger "Teacher" model at three different levels:

The Transformer layer

The Embedding layer

The Prediction layer



# TINYBERT

TinyBERT has only about 14.5 million parameters, which is more than 7 times smaller than the original BERT model, which has 110 million parameters.

Comparison	BERT October 11, 2018	RoBERTa July 26, 2019	DistilBERT October 2, 2019	ALBERT September 26, 2019
Parameters	Base: 110M Large: 340M	Base: 125 Large: 355	Base: 66	Base: 12M Large: 18M
Layers / Hidden Dimensions / Self-Attention Heads	Base: 12 / 768 / 12 Large: 24 / 1024 / 16	Base: 12 / 768 / 12 Large: 24 / 1024 / 16	Base: 6 / 768 / 12	Base: 12 / 768 / 12 Large: 24 / 1024 / 16
Training Time	Base: 8 x V100 x 12d Large: 280 x V100 x 1d	1024 x V100 x 1 day (4-5x more than BERT)	Base: 8 x V100 x 3.5d (4 times less than BERT)	[not given] Large: 1.7x faster
Performance	Outperforming SOTA in Oct 2018	88.5 on GLUE	97% of BERT-base's performance on GLUE	89.4 on GLUE
Pre-Training Data	BooksCorpus + English Wikipedia = 16 GB	BERT + CCNews + OpenWebText + Stories = 160 GB	BooksCorpus + English Wikipedia = 16 GB	BooksCorpus + English Wikipedia = 16 GB
Method	Bidirectional Trans- former, MLM & NSP	BERT without NSP, Using Dynamic Masking	BERT Distillation	BERT with reduced para- meters & SOP (not NSP)

# REFERENCES

<https://medium.com/@samia.khalid/bert-explained-a-complete-guide-with-theory-and-tutorial-3ac9ebc8fa7c>

<https://jalammar.github.io/illustrated-bert/>

<https://towardsdatascience.com/bert-3d1bf880386a>

<https://medium.com/data-science-in-your-pocket/understanding-bert-variants-part-1-740fee88616>

<https://www.scaler.com/topics/nlp/bert-variants/>