BERT Model

1. What is BERT:

BERT	-	Bidirectional Encoder Representations from Transformers
Applications	-	Sentiment Analysis, Predict Next Word, Language Understanding etc....

Let's break down bidirectional and unidirectional training in the context of BERT and Transformers:

1.1. Unidirectional Training:

- *Explanation:* In unidirectional training, the model processes the input sequence from one direction only, typically from left to right or right to left.
- *Example:* In traditional language models like LSTM-based models or autoregressive Transformers (like GPT), each token is predicted based only on the tokens to its left. For example, when predicting the next word in a sentence, the model considers only the words that precede it in the sequence.
- *Pros and Cons:*
  - *Pros:* Unidirectional models are simpler to train because they process the text sequentially.
  - *Cons:* They may struggle with capturing relationships between words that are far apart in the sequence, as they can't directly access information from the future context.

1.2. Bidirectional Training:

- *Explanation:* In bidirectional training, the model processes the input sequence in both directions simultaneously, capturing information from both past and future context.
- *Example:* BERT (Bidirectional Encoder Representations from Transformers) uses bidirectional training. It employs a technique called "masked language modeling" where some of the input tokens are randomly masked, and the model is trained to predict them based on both the left and right contexts.
- *Pros and Cons:*
  - *Pros:* Bidirectional models have a better understanding of context since they can leverage information from both directions.
  - *Cons:* They are more computationally expensive and require more complex training procedures because they need to consider both past and future context during training.

1.3. BERT Model Variants:

1. BERT Base
2. BERT Large

1.4. Difference Between BERT and Transformer Architectures:

BERT	*Transformer*
It has only encoder part	It has both encoder and decoder parts
It is a bidirectional training model.	It is a unidirectional training model.
One model sufficient to perform multiple of tasks (Questioning, answering, generating text).	Multiple models required for multiple tasks
Tokenization happing internally	Tokenization should do explicitly
BERT Base Model: Embedding dimensionality: 784 d Multi-heads: 12 Number of encoders: 12 BERT Large Model: Embedding dimensionality: 1024 d Multi-heads: 16 Number of encoders: 24	Embedding dimensionality: 512 d Multi-heads: 8 Number of encoders: 6

2. BERT Training:

The BERT model training divided into two parts are:

1. Pre-training
2. Fine tuning

3. Pretraining:

Imagine you're teaching a robot to understand language. Before it learns specific tasks, it needs to learn about language in general. So, during pre-training, the model is fed lots of text without specific instructions. It learns patterns and relationships within the text.

1. Input embedding
2. MLM
3. Sequence training

4. Input Embedding:

1. Adding CLS & SEP to every sentence.
2. Sequence, Segment, Token embedding.

4.1.1. Adding CLS & SEP to Every Sentence:

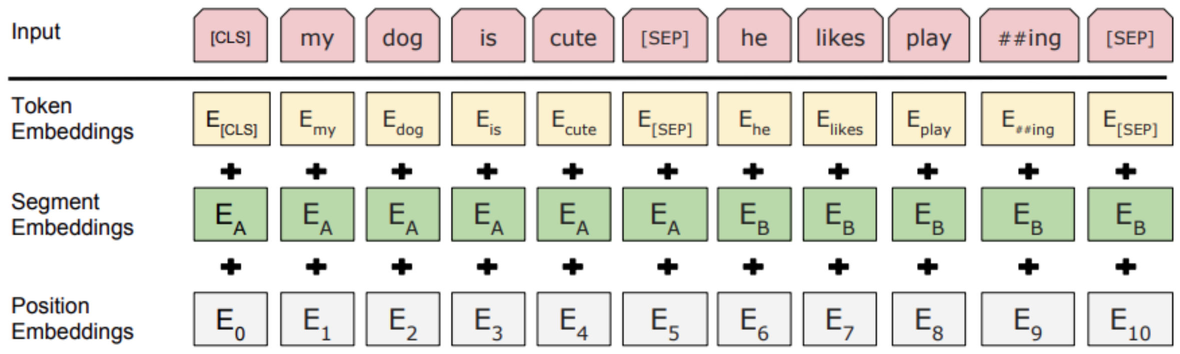
My dog is cute. He likes playing.	[CLS] My dog is cute.[SEP] He likes playing.[SEP]
Rama likes tea	[CLS]Rama likes tea.[SEP]

There are three benefits of adding CLS and SEP:

- ***Sentence Representation:*** [CLS] token helps BERT represent the entire sentence for tasks like deciding if a sentence is positive or negative.
- ***Sentence Separation:*** [SEP] token shows where one sentence ends and another begins, helping BERT understand the structure of text.
- ***Next Sentence Prediction:*** BERT learns if two sentences are related or not by predicting if they come one after the other. The [CLS] and [SEP] tokens help BERT do this effectively.

4.1.2. Sequence, Segment, Token Embedding:

Effectively


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}

4. MLM:

Masked language modeling (MLM)

- Randomly masking some words and training the model to predict the masked word.

Preparing training data in Sequence training:

- In all data 80% data got masked at a specific token and 10% got replaced with random tokens and last 10% data keep as it is.

5. Sequence Training:

Next Sentence Prediction (NSP):

- In addition to MLM, BERT also undergoes pre-training with a task called Next Sentence Prediction (NSP).
- Here, the pairs of sentences pass as input to model and is trained to predict whether the second sentence follows (depends on) the first one in the original text or not.
- NSP helps BERT understand relationships between sentences and enables it to capture broader context beyond individual sentences.

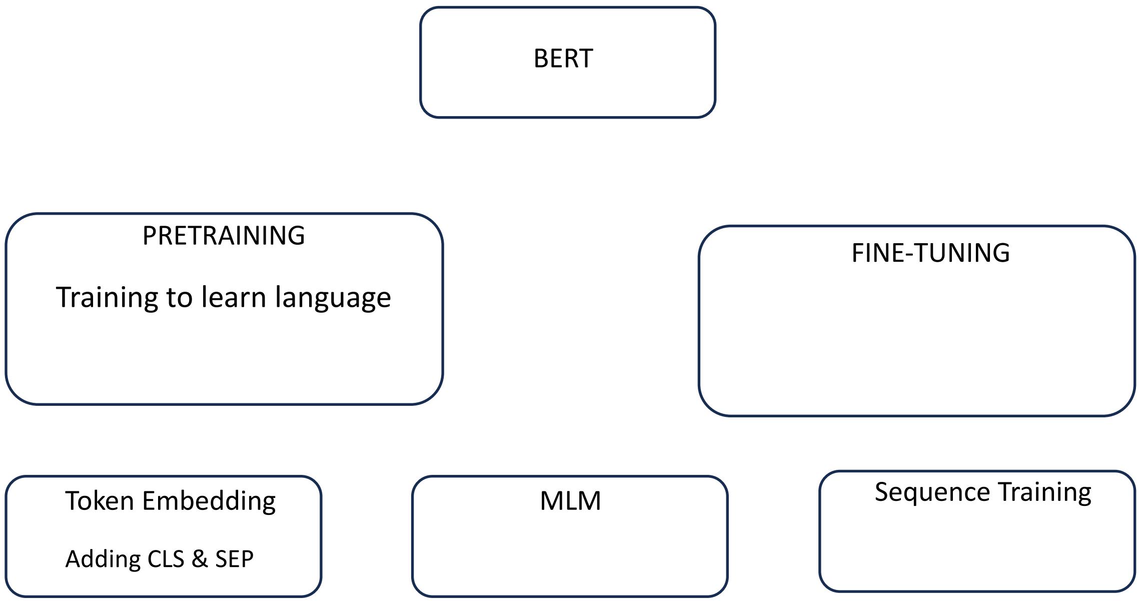
Preparing training data in Sequence training:

- There are two types of data generate for sequential training.
- First selecting sequence of two sentences from document data and label them as 'yes'.

- Second one selecting random sentences from different documents or data document label them as ‘no’.

6. Fine-tuning:

Once the robot has a good understanding of language from pre-training, it's time to teach it specific tasks, like answering questions or summarizing text. To do this, we start with what the robot already knows from pre-training (its pre-trained parameters). Then, we give it labeled examples of the specific tasks we want it to perform. The robot adjusts its understanding based on these examples, fine-tuning its parameters to become better at the specific tasks. 2.1.1. Input embedding:



CHAPTER – 2

GPT

1. What is GPT:

GPT – Generative pretrained transformers

The Transformers needs tons of sequence labels data for training. But GPT overcomes this problem by transfer learning with small size data.

It is a decoder-based transformer and it contains only decoder part.

2. GPT Training:

The GPT training will divide into two parts:

1. Pretraining
2. Fine tuning

Pretraining:

Training to GPT architecture to understand what language is

Fine tuning:

Use transfer learning to make GPT architecture perform well on a specific task.

Can transfer learning first recent a sequence of words and it will generate a specific length with randomly initialized values and these values will update for every training based on how a word close to other word.

3. Issues in Fine Tuning:

1. Still too much data required: In fine tuning, we required hundreds and thousands of examples to train the model.
2. Overfit is easy: If we use very less data in fine tuning, the model may overfit with pretrained data. In this case it might not be work properly with expected task.
3. Not how humans learn: GPT is not like a human because human can learn with small set of examples but GPT needs thousands of examples.
4. Failed to recognize complex sentence context: If the sentence contains any mathematical operations or numbers GPT fails to recognize that entity.

These all issues weaken overcome by using GPT-2 meta learning.

4. GPT-2 Meta Learning:

The pre training is as same as GPT, but it will use zero-shot learning technique instead of fine-tuning.

It trained with 1.5 billion parameters.

Zero-shot learning:

Ten zero shot learning it performs a specific task when given just an instruction (prompt) along with input (Sequence of words)

5. GPT-3 Meta Learning:

GPT-3 is the third-generation model in GPT.

The GPT-3 pretraining is as like as GPT-2, but the main difference of GPT-2 and GPT-3 is architecture and meta-learning.

The GPT-3 meta learning and involve three types of learnings:

1. Zero-shot learning: input text + prompt
2. One-shot learning: input text + prompt + one example what we want
3. Few-short learning: input text + prompt + multiple example what we want

GPT-3 works on meta-learning with updated GPT-2 architecture.

It trained with 175 billion parameters.

CHAPTER – 3

XLNET Model

XLNET	-	eXtreme Language NETwork.
Applications	-	Sentiment Analysis, Predict Next Word, Language Understanding etc....

1. Autoencoding and Autoregressive Learning:

Autoencoding Learning: Autoencoding learning predicts the masked word without considering the previously predicted word in the same sequence. The BERT model uses this technique.

Autoregressive Learning: Autoregressive learning predicts the masked or next word in the sequence by using previously predicted words.

2. What is XLNET:

- XLNet is a *Permutation Language Modeling* (PLM) which is implemented with BERT (Autoencoding learning technique) and Autoregressive learning.
- Xlnet uses the ***bidirectional learning*** from BERT model, and ***autoregressive learning*** to avoid the ***disadvantages (Independent learning of each masked word)*** in BERT model.

3. Problem With BERT:

BERT uses a technique called Masked Language Modeling (MLM), where it randomly masks some of the tokens in a sentence and then trains the model to predict those masked tokens based on their context. The challenge arises when multiple words are masked in a sentence, BERT predicts each masked word independently, without considering the relationship between them.

3.1. Example with BERT:

Consider the sentence: "Rama goes to the mall and purchases clothes."

Training with Masking: "Rama goes to the [MASK] and purchases [MASK]."

During training, BERT learns to predict each masked token ([MASK]) independently:

It Might Predict: "Rama goes to the cinema and purchases clothes."

Here, BERT treats each mask independently, which can result in incoherent predictions like *"Rama goes to the **cinema** and purchases **clothes**,"* where "***cinema***" and "***clothes***" do not make sense together.

XLNet addresses this issue by using a permutation-based autoregressive approach. Instead of masking tokens, XLNet considers all possible permutations of the input sequence during training. This allows the model to capture dependencies between words more effectively, as it is trained to predict tokens based on their surrounding context in multiple permutations.

4. Problem With Autoregressive Learning:

Auto regressive learning predicts the future word in the sequence by using only left context of the sentence. The problem is, it not considers the right context of the masked word.

This problem we can overcome by using BERT model, because it considers both left and right context while learning the language.

5. Difference Between BERT And XLNET Model Training Prediction:

$$\tau_{BERT} = \log\, p(New \;/\; is\ a\ city) \;+ \log\, p(York \;/\; is\ a\ city)$$

$$\tau_{XLNET} = \log\, p(New \;/\; is\ a\ city) \;+ \log\, p(York \;/\; New, is\ a\ city)$$

6. XLNet Learning process: