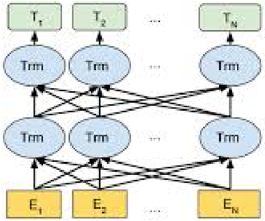
BERT - Bidirectional Encoder Representations from Transformers

Jaideep Ganguly

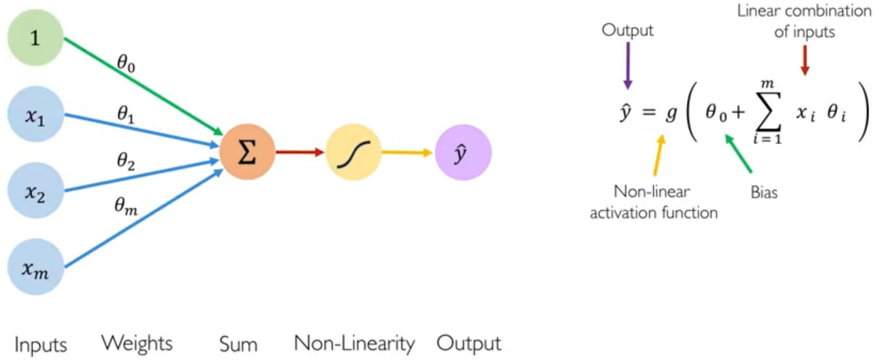
September 29, 2020



Agenda

1. Elementary & Basic Stuff - 9 slides
   1. Neural Net, Activation Functions
   2. One Hot Encoding, Word Embedding, Softmax
   3. Loss Function, Back Propagation
   4. Information & Uncertainty, Kulback-Leiber Divergence
2. BERT - 17 slides
   1. What is BERT?
   2. A brief history of NLP using Deep Learning
   3. Attention is all you need!
   4. Masked Language Model (MLM), GELU
   5. Next Sentence Prediction (NSP)
   6. BERT Architecture

Neural Net



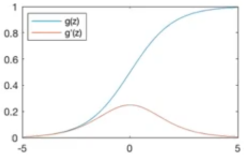
$$\hat{y} = g\left(\theta_0 + \sum_{i=1}^{m} x_i\,\theta_i\right)$$

Figure: Forward Propagation

Source: MIT 6.S191 (2018) - Introduction to Deep Learning

Activation Function

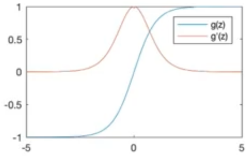
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

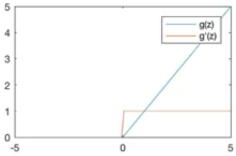
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

Figure: Activation Functions

One Hot Encoding

1. The string encoded variable is replaced with new variables of boolean type. For example, a particular feature, say color, can have "red", "green" and "blue" as possible values. This feature color, will be replaced by 3 features, red, blue and green which hold the values 1 or 0. We can then encode color, in terms of red, blue and green as follows:

red	green	blue
1	0	0
0	1	0
0	0	1

Note that if the boolean value of the feature is 1, it must be 0 for all the other features

Word Embedding

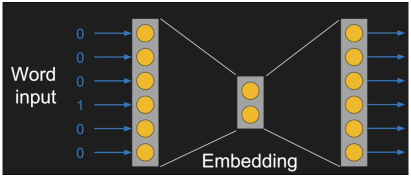


Figure: Word Embedding

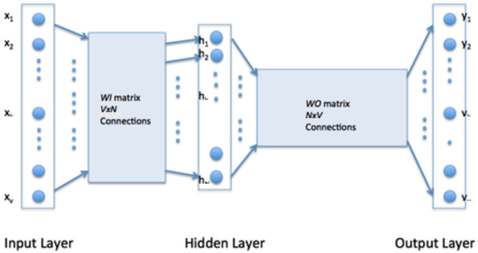


Figure: Hidden Layers

SoftMax

1. Input to neuron as well as output from neurons needs to be **normalized**. Some of these values can also be negative and the components may not add up to 1.

2. The softmax function takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities in the interval 0...1 with the components adding up to 1.

$$\sigma(z)_i = \frac{e_i^z}{\sum_{j=1}^{K} e^{z_j}} \tag{1}$$

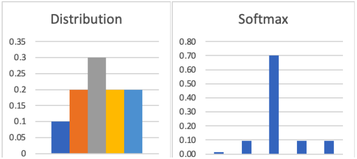


Figure: Probability distribution

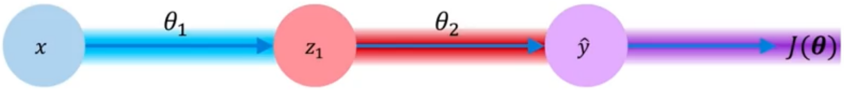
Loss Function

The objective is to find a good model that fits the training data. For that, we need to minimize some loss function over the training data **D**. The total loss **J** corresponding to **r** set of data is given by:

$$J = Squared\ Error\ Loss = \frac{1}{2} \times \sum_{i=1}^{r} (\hat{y}_i - y_i)^2 \tag{2}$$

where $\hat{y}_i$ is the observed value of the ith outcome and n is the number of features. The factor $1/2$ is introduced to cancel out 2 post differentiation.

Back Propagation



$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} = \frac{\partial J(\boldsymbol{\theta})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial \theta_1}$$

Figure: Loss Minimization

θ is the weight. Repeat this for every weight in the network using gradients from later layers.

Information & Uncertainty

1. In Clause Shannon's information theory, one bit of information reduces the uncertainty by 2. Similarly, if 3 bits of information are sent, then the reduction in uncertainly by 2^3, i.e., 8. This is intuitive. With 3 bits, there could be 8 possible values and so if a particular set of bits are transmitted, 8 possibilities are eliminated with 1 certainty. **Information Content** When the information is probabilistic, the self-information I_x, or Information Content of *measuring a random variable X as outcome x is defined as:*

$$I_x = log\left(\frac{1}{p(x)}\right) = -log(p(x)) \tag{3}$$

where $p(x)$ is probability mass function. **Shannon Entropy** of the random variable X is defined as:

$$H(X) = \sum_x -p(x)log(p(x)) = E(I_x) \tag{4}$$

It is the *expected information content* of the measurement of X.

Cross Entropy - Kullback–Leibler (KL) divegence

1. **Cross-Entropy** is defined as:

$$H(p, q) = -\sum_i p(i) log_2 q(i) \tag{5}$$

where p is the true distribution and q is the predicted distribution. If the predictions are perfect, then the cross-entropy is same as the entropy. If the prediction differs, then there is a divergence which is known as *Kullback–Leibler (KL) divergence*. Hence,

$$Cross\ Entropy = Entropy + KL\ Divergence$$

or,

$$KL\ Divergence = H(p, q) - H(p)$$

Hence, if the predicted distribution is closer to true distribution when KL divergence is low.

What is BERT?

1. **Bidirectional Encoder Representations from Transformers**.
2. As opposed to directional models, which read the text input sequentially (left-to-right or right-to-left), the Transformer encoder reads the entire sequence of words at once. Therefore it is considered bidirectional, or rather non-directional. This characteristic allows the model to learn the context of a word based on all of its surroundings.
3. BERT uses the **Transformer**, **an attention mechanism that learns contextual relations between words in a text**. Since BERT's goal is to generate a language model, only the encoder mechanism is necessary.
4. A pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks.

A Brief History of NLP using Deep Learning

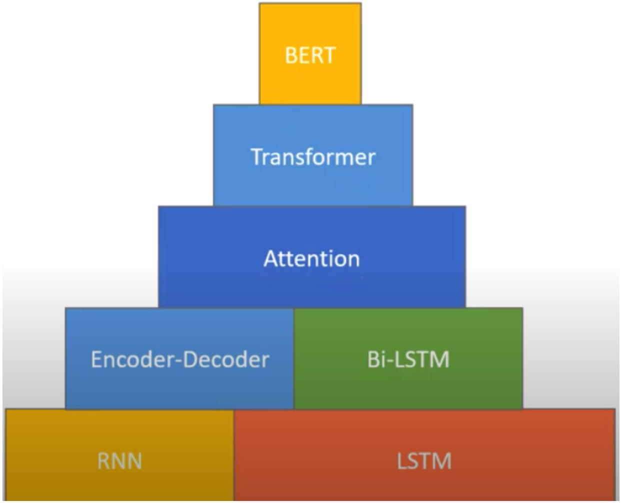


Figure: Abridged history of NLP using Deep Learning

Attention is all you need!

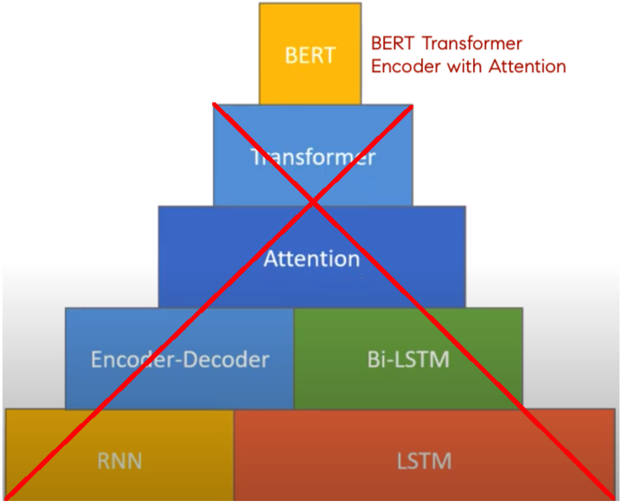


Figure: Attention is all you need!

Masked Language Model

1. **Masked Language Model** - Predict the masked word. 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence. This is done by:
    1. Adding a classification layer on top of the encoder output.
    2. Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
    3. Calculate probability of each word in the vocabulary with **softmax**.
2. BERT loss function takes into consideration only the prediction of the masked values and ignores the prediction of non-masked words. As a consequence, the model converges slower than directional models, a characteristic which is offset by its increased context awareness.

MLM Contd.

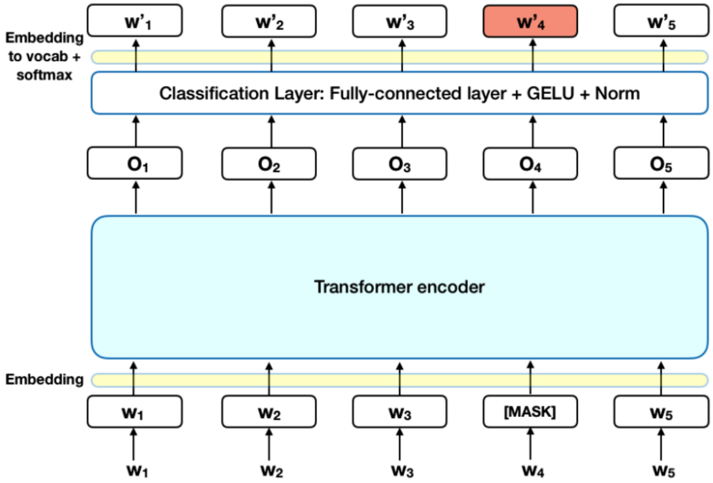

w'1
w'2
w'3
w'4
w'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
w1
w2
w3
[MASK]
w5
w1
w2
w3
w4
w5

Figure: Masked Language Modeling (MLM)

GELU - Gaussian Error Linear Unit

$$\mathrm{GELU}(x) := x\mathbb{P}(X \le x) = x\Phi(x)$$
$$= 0.5x\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$

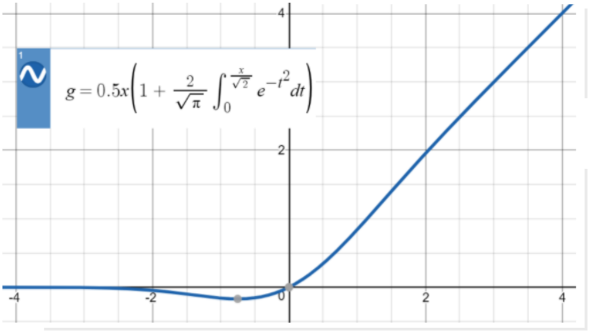


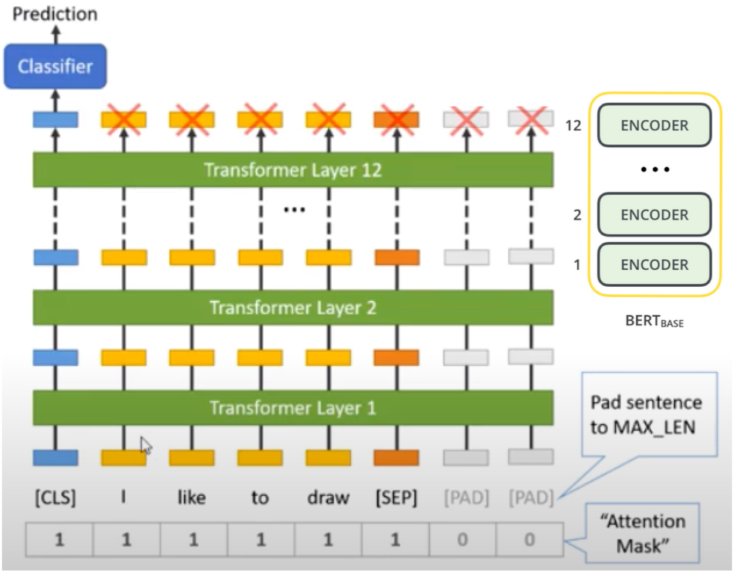
Figure: Masked Language Modeling (Gaussian Error Linear Unit)

Next Sentence Prediction (NSP)

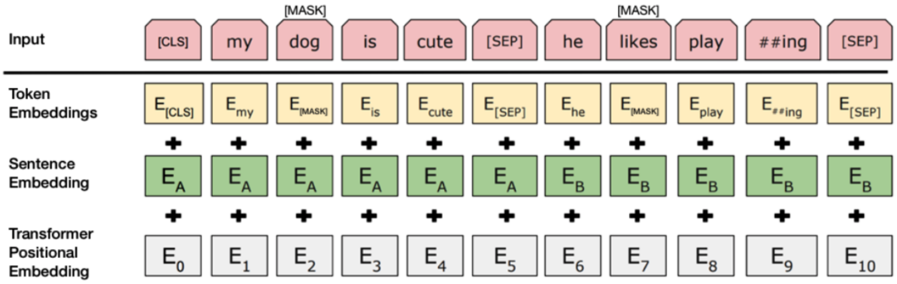
Next Sentence Prediction - The model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document. In the other 50% a random sentence from the corpus is chosen as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence. Before entering the model:

1. A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
2. A sentence embedding indicating Sentence A or Sentence B is added to each token.
3. A positional embedding is added to each token to indicate its position in the sequence.

BERT Architecture (Parallel)



BERT Architecture (Parallel)


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

BERT Encoder

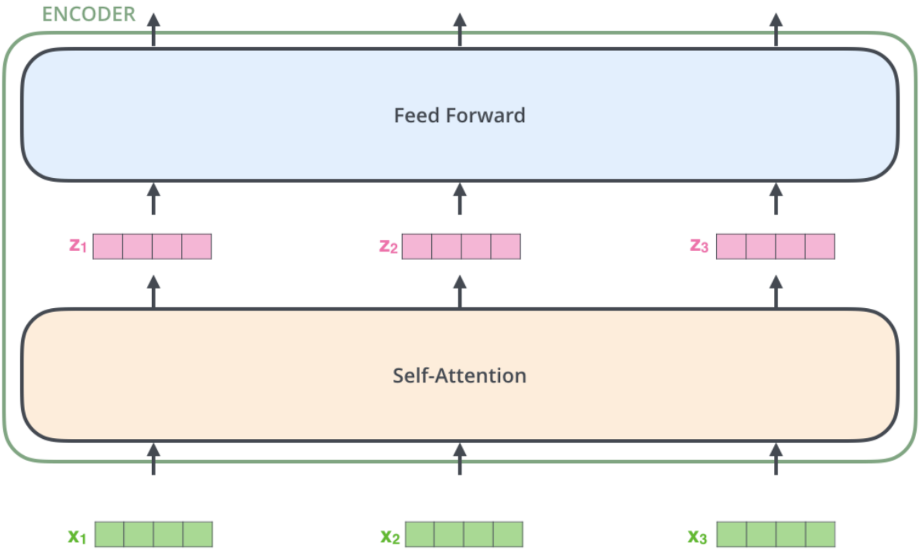


Figure: BERT Encoder

Attention

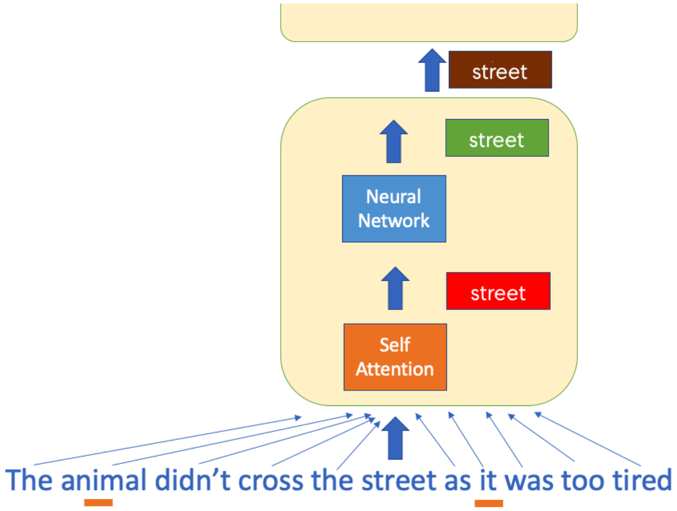
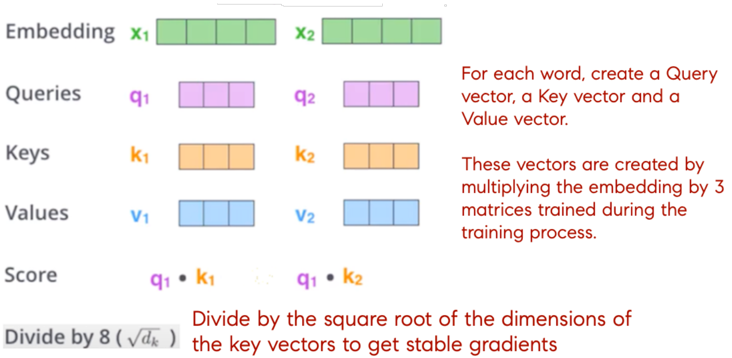


Figure: Attention Example

Self Attention



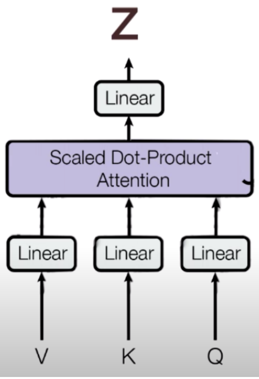
For each word, create a Query vector, a Key vector and a Value vector.

These vectors are created by multiplying the embedding by 3 matrices trained during the training process.

Divide by the square root of the dimensions of the key vectors to get stable gradients

Figure: Self Attention

Self Attention

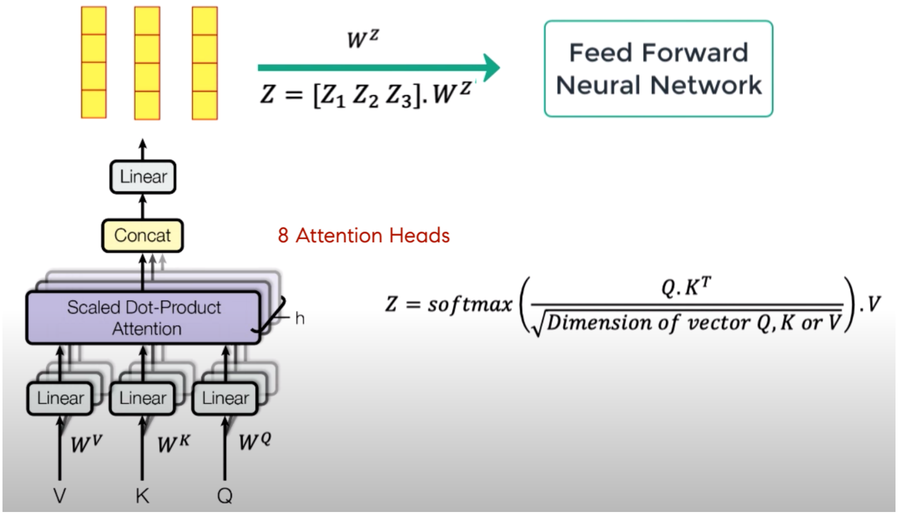


$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{QK}^\top}{\sqrt{n}})\mathbf{V}$$

$$Z = softmax\left(\frac{Q.K^T}{\sqrt{Dimension\ of\ vector\ Q, K\ or\ V}}\right).V$$

Figure: Self Attention

Multi Head Attention



$$Z = [Z_1\ Z_2\ Z_3].W^Z$$

$$Z = softmax\left(\frac{Q.K^T}{\sqrt{Dimension\ of\ vector\ Q, K\ or\ V}}\right).V$$

Figure: Multi Head Attention

NSP Contd.

Fine Tuning - Add a small layer to the core model.

1. In Named Entity Recognition (NER), the software receives a text sequence and is required to mark the various types of entities (Person, Organization, Date, etc) that appear in the text. Using BERT, a NER model can be trained by feeding the output vector of each token into a classification layer that predicts the NER label.

2. Classification tasks such as sentiment analysis are done similarly to Next Sentence classification, by adding a classification layer on top of the Transformer output for the [CLS] token.

3. In Question Answering tasks (e.g. SQuAD v1.1), the software receives a question regarding a text sequence and is required to mark the answer in the sequence. Using BERT, a Q&A model can be trained by learning two extra vectors that mark the beginning and the end of the answer.

BERT Fine Tuning

1. Model size matters, even at huge scale. BERT_large, with 345 million parameters, is the largest model of its kind. It is demonstrably superior on small-scale tasks to BERT_base, which uses the same architecture with "only" 110 million parameters.
2. With enough training data, more training steps == higher accuracy. For instance, on the MNLI task, the BERT_base accuracy improves by 1.0% when trained on 1M steps (128,000 words batch size) compared to 500K steps with the same batch size.
3. BERT's bidirectional approach (MLM) converges slower than left-to-right approaches (because only 15% of words are predicted in each batch) but bidirectional training still outperforms left-to-right training after a small number of pre-training steps.

RoBERTa

RoBERTa performs better than BERT by applying the following adjustments:

1. Bigger training data (16G vs 161G).
2. Using dynamic masking pattern (BERT use static masking pattern). Dynamic masking refers to "training data was duplicated 10 times so that each sequence is masked in 10 different ways over the 40 epochs of training." In Batch Gradient Descent, the **Batch Size** refers to the total number of training examples present in a single batch and **Epoch** means that each sample in the training dataset has had an opportunity to update all the model parameters.
3. Replacing the next sentence prediction training objective.
4. Training on longer sequences.

Questions?

Thank you!