BERT

Bidirectional Encoder Representations from Transformers

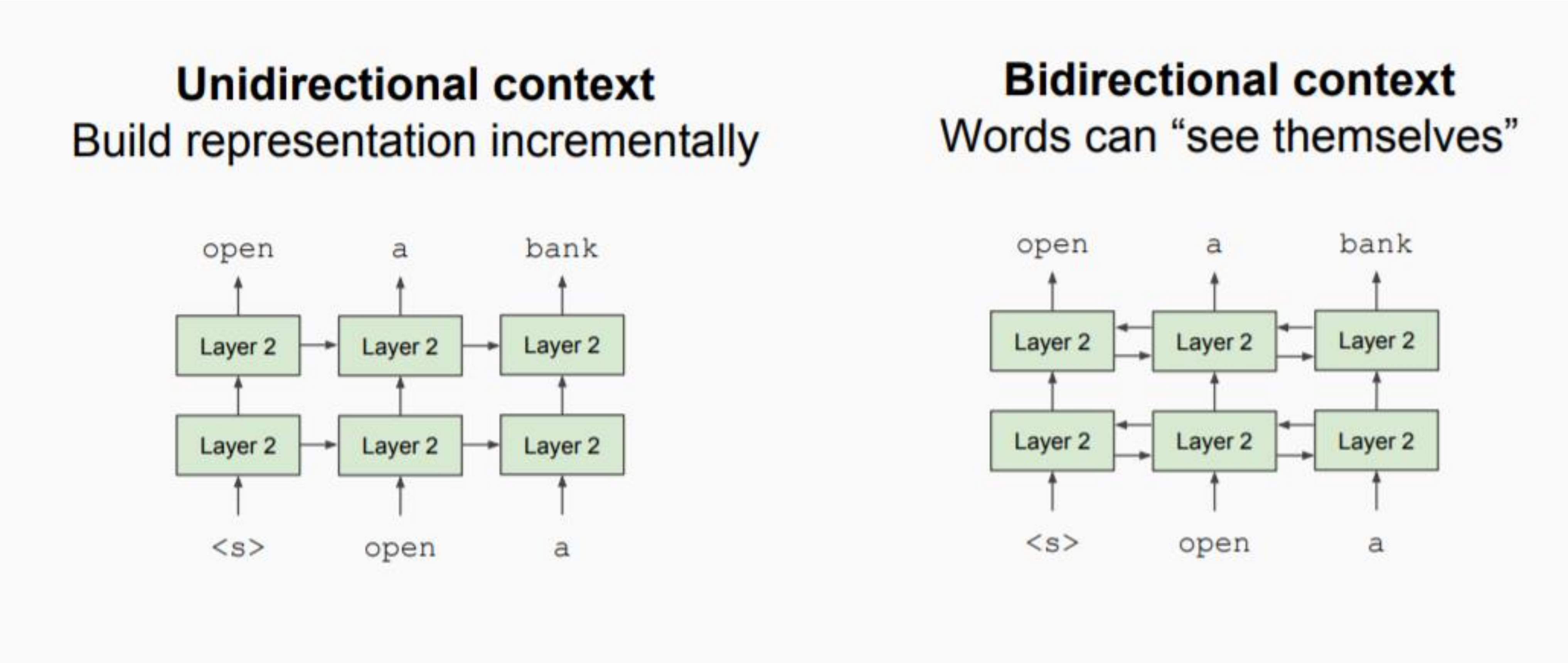
Introduction – What is BERT?

- Latest language representational model
- BERT is conceptually simple and empirically powerful.
- One of the biggest challenges in natural language processing (NLP) is the shortage of training data.
- most task-specific datasets contain only a few thousand or a few hundred thousand human-labelled training examples.
- anyone in the world can train their own state-of-the-art question answering system (or a variety of other models) in a few hours.

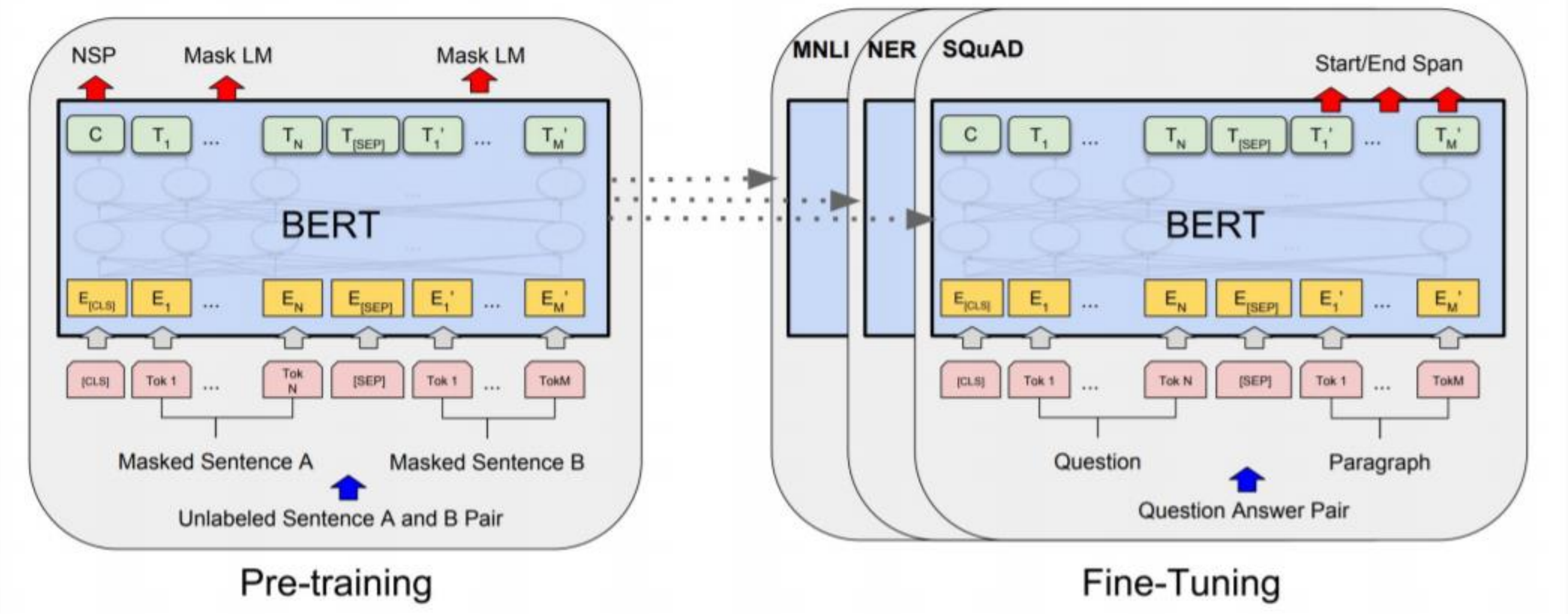
What makes BERT different?

- BERT builds upon recent work in pre-training contextual representations — including ELMo, Generative Pre-Training (OPENAI-GPT)
- These previous models are unidirectional.
- BERT is the first *deeply bidirectional*, *unsupervised* language representation, multilingual model.
- In BERT they have improved the fine tuning approach by introducing two new pre-training objectives, i.e. the Masked Language Model and the Next sentence prediction task.

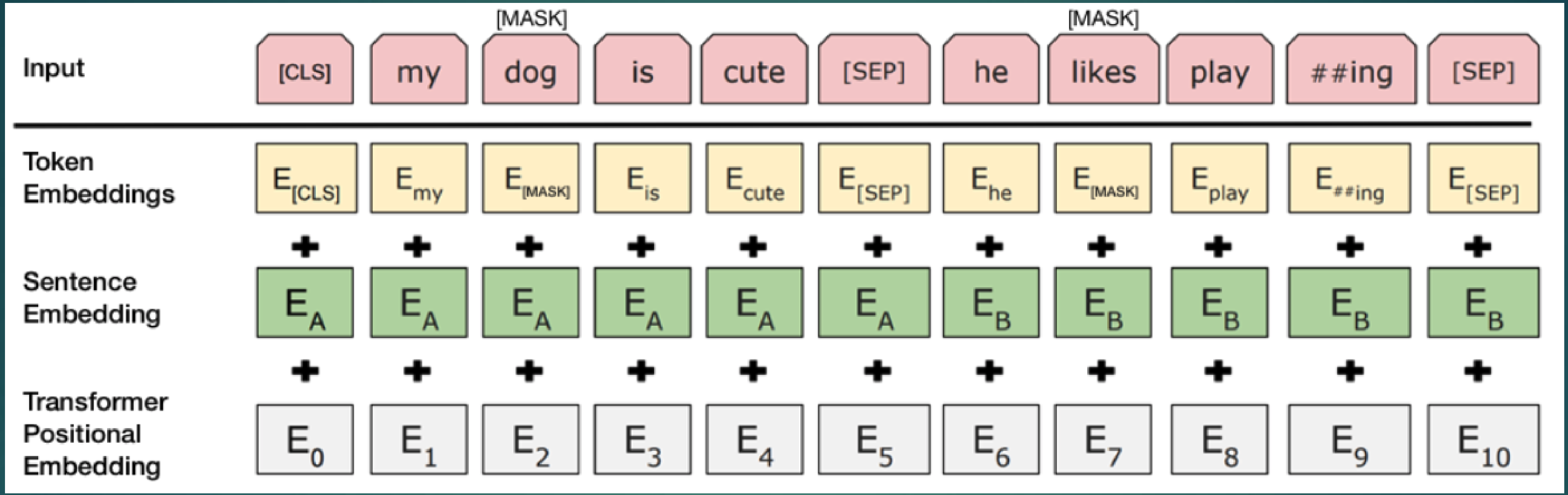
Unidirectional vs Bidirectional



Pre-Training and Fine-Tuning



Model Architecture



- Token Embeddings: Uses pretrained WordPiece embeddings (supports sequence lengths up to 512 tokens)
- The first token of every sequence is always the special classification embedding ([CLS])
- Sentences are separated using a special token [SEP]
- Learned sentence A embedding is added to every token of the first sentence and a sentence B embedding to every token of the second sentence

Task#1: Masked LM

- 15% of the words are masked at random and the task is to predict the masked words based on its left and right context
- Not all tokens were masked in the same way (example sentence "My dog is hairy")
    - 80% are replaced by the token: "My dog is [MASK] "
    - 10% are replaced by a random token: "My dog is apple"
    - 10% are left intact: "My dog is hairy"

The BERT loss function takes into consideration only the prediction of the masked values and ignores the prediction of the non-masked words.

Task#2: Next Sentence Prediction

- Many downstream tasks are based on understanding the relationship between two text sentences
  - Question Answering (QA) and Natural Language Inference (NLI)
  - Language modeling does not directly capture that relationship.
- The task is pre-training binarized next sentence prediction task.

Input = [CLS] the kid [MASK] all the ice-cream [SEP] he [MASK] not hungry anymore [SEP]

Label = isNext

Input = [CLS] the kid [MASK] all the ice-cream [SEP] I think I [MASK] buy the red car [SEP]

Label = NotNext

Fine-Tuning task for SQuAD

- **INPUT QUESTION**

  Where do water droplets collide with ice crystals to form precipitation?

- **INPUT PARAGRAPH**

  .... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. …

- **OUTPUT ANSWER**

  Within a cloud

Fine-Tuning task for SQuAD

- Represent the input question and paragraph as a single packed sequence.
  - The question uses the A embedding and the paragraph uses the B embedding
- New parameters to be learned in fine-tuning are start vector $S \in R^H$ and end vector $E \in R^H$
- Calculate the probability of word & being the start of the answer span

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

- The training objective is the log-likelihood the correct and end positions

Prediction in SQuAD(using final hidden layer of BERT and its weights)

```python
def create_model(bert_config, is_training, input_ids, input_mask, segment_ids,
                 use_one_hot_embeddings):
  """Creates a classification model."""
  model = modeling.BertModel(
      config=bert_config,
      is_training=is_training,
      input_ids=input_ids,
      input_mask=input_mask,
      token_type_ids=segment_ids,
      use_one_hot_embeddings=use_one_hot_embeddings)

  final_hidden = model.get_sequence_output()

  final_hidden_shape = modeling.get_shape_list(final_hidden, expected_rank=3)
  batch_size = final_hidden_shape[0]
  seq_length = final_hidden_shape[1]
  hidden_size = final_hidden_shape[2]

  output_weights = tf.get_variable(
      "cls/squad/output_weights", [2, hidden_size],
      initializer=tf.truncated_normal_initializer(stddev=0.02))

  output_bias = tf.get_variable(
      "cls/squad/output_bias", [2], initializer=tf.zeros_initializer())

  final_hidden_matrix = tf.reshape(final_hidden,
                                   [batch_size * seq_length, hidden_size])
  logits = tf.matmul(final_hidden_matrix, output_weights, transpose_b=True)
  logits = tf.nn.bias_add(logits, output_bias)

  logits = tf.reshape(logits, [batch_size, seq_length, 2])
  logits = tf.transpose(logits, [2, 0, 1])

  unstacked_logits = tf.unstack(logits, axis=0)

  (start_logits, end_logits) = (unstacked_logits[0], unstacked_logits[1])

  return (start_logits, end_logits)
```

Calling the Above Create model Function

```
(start_logits, end_logits) = create_model(
    bert_config=bert_config,
    is_training=is_training,
    input_ids=input_ids,
    input_mask=input_mask,
    segment_ids=segment_ids,
    use_one_hot_embeddings=use_one_hot_embeddings)
```

Computation of Loss

```
def compute_loss(logits, positions):
  one_hot_positions = tf.one_hot(
      positions, depth=seq_length, dtype=tf.float32)
  log_probs = tf.nn.log_softmax(logits, axis=-1)
  loss = -tf.reduce_mean(
      tf.reduce_sum(one_hot_positions * log_probs, axis=-1))
  return loss

start_positions = features["start_positions"]
end_positions = features["end_positions"]

start_loss = compute_loss(start_logits, start_positions)
end_loss = compute_loss(end_logits, end_positions)

total_loss = (start_loss + end_loss) / 2.0
```

EXPERIMENTS

- GLUE (General Language Understanding Evaluation) benchmark
  1. MNLI: Multi-Genre Natural Language Inference
  2. QQP: Quora Question Pairs
  3. QNLI: Question Natural Language Inference
  4. SST-2: Stanford Sentiment Treebank
  5. CoLA: The corpus of Linguistic Acceptability
  6. STS-B: The Semantic Textual Similarity Benchmark
  7. MRPC: Microsoft Research Paraphrase Corpus
  8. RTE: Recognizing Textual Entailment
  9. WNLI: Winograd NLI
- SQuAD v1.1

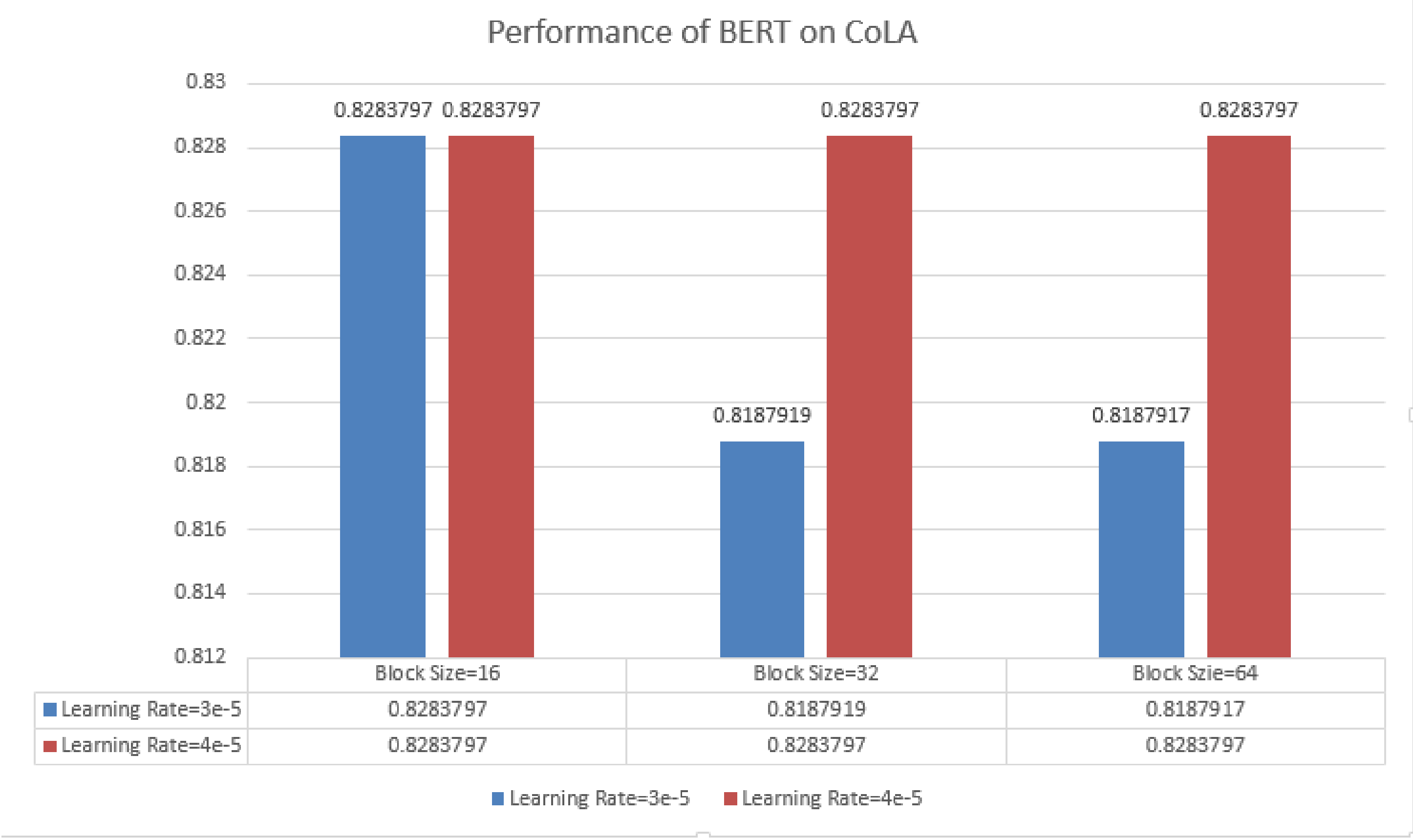
EXPERIMENTS Cont.

- BERT-BASE pre trained model that contains 12 layers (Transformer blocks), 768 hidden layers, 12 heads and 110M parameters.
- Range of Hyperparameters:
  - Batch Size: 16,32
  - Learning rate: 5e-5, 4e-5, 3e-5, 2e-5
  - Number of epochs: 3, 4

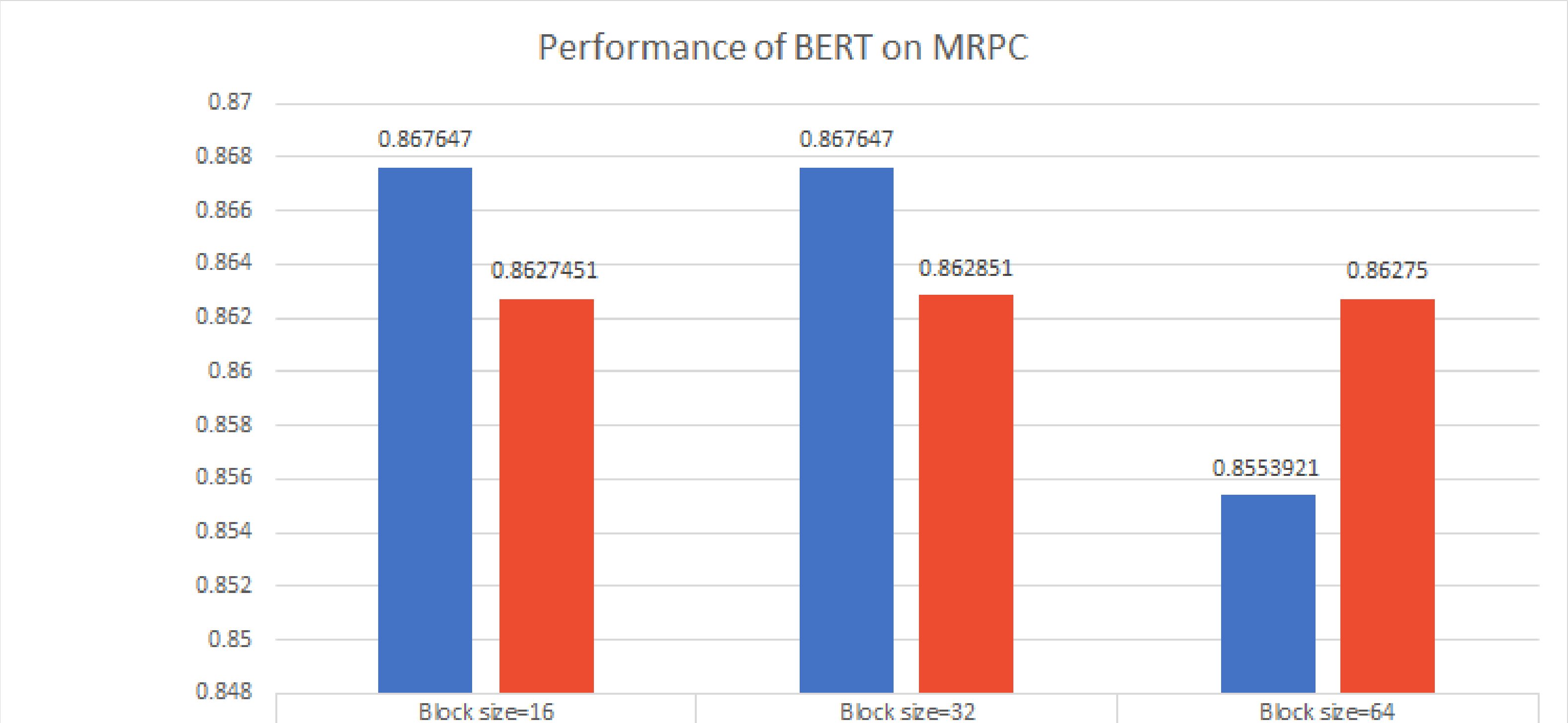
RESULTS

- We use 3 epochs for the above tasks and successfully reproduced the results to a satisfactory accuracy.
- CoLA (Corpus Linguistic Acceptability)
- MRPC (Microsoft Research Paraphrase Corpus)
- MNLI (Multi-Genre Natural Language inference)
- SQuAD v1.1
  - F1 score = 88.587

CoLA (Corpus Linguistic Acceptability)



MRPC (Microsoft Research Paraphrase Corpus)



	Block size=16	Block size=32	Block size=64
Learning rate=3e-5	0.867647	0.867647	0.8553921
Learning rate=4e-5	0.8627451	0.862851	0.86275

Future Work

- Many different adaptations, tests, and experiments have been left for the future due to lack of time (i.e. the experiments with large data sets are usually very time consuming, requiring even days to finish a single run).
- Deep analysis of the transformer, updations in transformer like change in the number of layers of Encoder and Decoder.