BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Ming-Wei Chang, Google AI Language
Natural Language Processing (NLP)

- Enabling computers to process natural language
  - E.g. sentiment analysis, question answering, ....
- Example: Question Answering

**Q:** The traveling salesman problem is an example of what type of problem?

**P:** A function problem is a computational problem ... Notable examples include the traveling salesman problem and the integer factorization problem.

**A:** A function problem is a computational problem where a single output ... Notable examples include the traveling salesman problem and the integer factorization problem.
The Language Representation Problem

- Q: How to represent text for machine learning models?
- Many machine learning models (such as neural networks) expect continuous vectors as input
- The representations should capture the semantic meaning
How Should We Represent Words?

- A one-hot vector? Traditional NLP

  ![Vector representation of king and queen](image)

- Distances between any two words ...
  - are always the same!
  - However, “queen” should be more related to “king” compared to “headphone”

Can we do better?
Continuous Representations of Words

- Representing words with continuous values

  ![Diagram showing inner product between word embeddings](image)

- Word embeddings (`word2vec`, `GloVe`) are often pre-trained on unlabeled text corpus from co-occurrence statistics

  ![Diagram showing word embeddings](image)

Word2Vec [mikolov et-al, 13], PLSI [Hoffman, 99], GloVe [Pennington et-al, 14],
Contextual Representations

- **Problem** of word embeddings: context-independent

  \[
  \text{open a bank account} \quad \text{on the river bank}
  \quad [0.3, 0.2, -0.8, \ldots]
  \]

- Ideally, representations should be contextual

  \[
  \begin{align*}
  \text{open a bank account} & : [0.9, -0.2, 1.6, \ldots] \\
  \text{on the river bank} & : [-1.9, -0.4, 0.1, \ldots]
  \end{align*}
  \]
Training Contextual Representations

Can we do better?

- Pre-training contextual representations:
  - Semi-Supervised Sequence Learning, Google, 2015
  - ELMo, AI2, 2017
  - Generative Pre-Training, OpenAI, 2018
  - ULMFit, fast.ai, 2018

- Training language models on text corpus:
  - Generate contextual representations. Single direction.

\[
P(\text{open a bank account}) = P(\text{open}) \times P(a|\text{open}) \times P(\text{bank}|\text{open, a}) \times P(\text{account}|\text{open, a, bank})
\]
BERT!

- Deep **Bi-directional** Pre-training
  - Using Transformer Blocks
- Learning contextual representations
  - With unlabeled data
  - Not just embeddings; Model initialization.
- Pre-training is very powerful
  - State-of-the-art performance for 11 tasks
  - With little task-specific engineering
Input and Output for BERT
BERT Model

- Every token will be translated into a representation vector

- Bi-directional Transformer [Vaswani et al. 17]

- What is the input/output for BERT?
Unified Input Representation

Always represent the input as a long sequence

- For text pair, pack two sentences in one sequence
- For single text task (such as) classification or tagging, pack one sentence in one sequence
Sentence Pair Classification Tasks

Class Label

BERT

Sentence A
Sentence B
Single Sentence Classification Tasks

Class Label

BERT

E_{[CLS]}  E_1  E_2  \ldots  E_N

[CLS]  Tok 1  Tok 2  \ldots  Tok N

Single Sentence
Sequence Tagging

Single Sentence

BERT

[CLS] Tok 1 Tok 2 ... Tok N

E_{[CLS]} E_1 E_2 ... E_N

O B-PER ... O

C T_1 T_2 ... T_N
BERT for Question Answering
Task List

- Different tasks have different input/output

<table>
<thead>
<tr>
<th>Input \ Output</th>
<th>Classification</th>
<th>Token-level Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Text Sequence</td>
<td>e.g. Sentiment Classification</td>
<td>e.g. Named Entity Recognition</td>
</tr>
<tr>
<td>Text Sequence Pairs</td>
<td>e.g. Entailment</td>
<td>e.g. Question Answering</td>
</tr>
</tbody>
</table>

- The [CLS] and [SEP] tokens and the unified input format make it is possible to use the same architecture
**Input Representation Details**

<table>
<thead>
<tr>
<th>Input</th>
<th>[CLS]</th>
<th>my</th>
<th>dog</th>
<th>is</th>
<th>cute</th>
<th>[SEP]</th>
<th>he</th>
<th>likes</th>
<th>play</th>
<th>#ing</th>
<th>[SEP]</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Token Embeddings</th>
<th>$E_{[CLS]}$</th>
<th>$E_{my}$</th>
<th>$E_{dog}$</th>
<th>$E_{is}$</th>
<th>$E_{cute}$</th>
<th>$E_{[SEP]}$</th>
<th>$E_{he}$</th>
<th>$E_{likes}$</th>
<th>$E_{play}$</th>
<th>$E_{#ing}$</th>
<th>$E_{[SEP]}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment Embeddings</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_B$</td>
<td>$E_B$</td>
<td>$E_B$</td>
<td>$E_B$</td>
<td>$E_B$</td>
</tr>
<tr>
<td>Position Embeddings</td>
<td>$E_0$</td>
<td>$E_1$</td>
<td>$E_2$</td>
<td>$E_3$</td>
<td>$E_4$</td>
<td>$E_5$</td>
<td>$E_6$</td>
<td>$E_7$</td>
<td>$E_8$</td>
<td>$E_9$</td>
<td>$E_{10}$</td>
</tr>
</tbody>
</table>

- Use 30,000 WordPiece vocabulary on input.
- Each token is sum of three embeddings
Pre-training BERT
Review: Supervised Training

- Labeled data: (input, output) pairs
Pre-training for BERT
Terminologies

- **Pre-training**
  - Training BERT with a large amount of unlabeled data

- **Fine-tuning**
  - Training BERT with a small amount of task-specific labeled data

- In BERT, we use pre-training as a way to initialize the whole model. This is different from just using fixed word embeddings.
Basic BERT Overall Framework

Pre-Training

Masked Sentence A

Masked Sentence B

Fine-Tuning

Start/End Span
Notes in the BERT Framework

- We always use the same model architecture
- We initialize all parameters from pre-training model
- We fine-tune all parameters in the fine-tuning stages
Pre-training: Masked LM

- **Solution**: Mask out $k\%$ of the input words, and then predict the masked words
  - We always use $k = 15\%$

- Too little masking: Too expensive to train
- Too much masking: Not enough context

```
the man went to the [MASK] to buy a [MASK] of milk
```

- `store`    `gallon`
  - ↑

Example sentence with masked words: "the man went to the [MASK] to buy a [MASK] of milk"
Pre-training: Next Sentence Prediction

- To learn *relationships* between sentences, predict whether Sentence B is actual sentence that follows Sentence A, or a random sentence

**Sentence A** = The man went to the store.
**Sentence B** = He bought a gallon of milk.
**Label** = IsNextSentence

**Sentence A** = The man went to the store.
**Sentence B** = Penguins are flightless.
**Label** = NotNextSentence
Transfer Learning; Not Multi-task Learning

Pre-Training

Fine-Tuning

Masked Sentence A

Masked Sentence B

Start/End Span

Question

Paragraph
Model Details

- Public BERT: Train on 3.3B words for 40 epochs
- BERT-Base: 12-layer, 768-hidden, 12-head
- BERT-Large: 24-layer, 1024-hidden, 16-head
- Trained on TPU for 4 days
- Pre-Trained models released and ready to use!
RESULTS
GLUE Results

### MultiNLI

**Premise:**
At the other end of Pennsylvania Avenue, people began to line up for a White House tour.

**Hypothesis:** People formed a line at the end of Pennsylvania Avenue.

**Label:** Entailment

### CoLA

**Sentence:** The wagon rumbled down the road.

**Label:** Acceptable

**Sentence:** The car honked down the road.

**Label:** Unacceptable

### Table

<table>
<thead>
<tr>
<th>System</th>
<th>MNLI-(m/mm)</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>392k</td>
<td>363k</td>
<td>108k</td>
<td>67k</td>
<td>8.5k</td>
<td>5.7k</td>
<td>3.5k</td>
<td>2.5k</td>
<td></td>
</tr>
<tr>
<td>Pre-OpenAI SOTA</td>
<td>80.6/80.1</td>
<td>66.1</td>
<td>82.3</td>
<td>93.2</td>
<td>35.0</td>
<td>81.0</td>
<td>86.0</td>
<td>61.7</td>
<td>74.0</td>
</tr>
<tr>
<td>BiLSTM+ELMo+Attn</td>
<td>76.4/76.1</td>
<td>64.8</td>
<td>79.9</td>
<td>90.4</td>
<td>36.0</td>
<td>73.3</td>
<td>84.9</td>
<td>56.8</td>
<td>71.0</td>
</tr>
<tr>
<td>OpenAI GPT</td>
<td>82.1/81.4</td>
<td>70.3</td>
<td>88.1</td>
<td>91.3</td>
<td>45.4</td>
<td>80.0</td>
<td>82.3</td>
<td>56.0</td>
<td>75.2</td>
</tr>
<tr>
<td>BERT&lt;sub&gt;BASE&lt;/sub&gt;</td>
<td>84.6/83.4</td>
<td>71.2</td>
<td>90.1</td>
<td>93.5</td>
<td>52.1</td>
<td>85.8</td>
<td>88.9</td>
<td>66.4</td>
<td>79.6</td>
</tr>
<tr>
<td>BERT&lt;sub&gt;LARGE&lt;/sub&gt;</td>
<td>86.7/85.9</td>
<td>72.1</td>
<td>91.1</td>
<td>94.9</td>
<td>60.5</td>
<td>86.5</td>
<td>89.3</td>
<td>70.1</td>
<td>81.9</td>
</tr>
</tbody>
</table>
SQuAD 1.1 Results

**What was another term used for the oil crisis?**

*Ground Truth Answers: first oil shock shock first oil shock shock.*

*Prediction: shock*

---

The 1973 **oil crisis** began in October 1973 when the members of the Organization of Arab Petroleum Exporting Countries (OAPEC, consisting of the Arab members of OPEC plus Egypt and Syria) proclaimed an **oil embargo**. By the end of the embargo in March 1974, the price of oil had risen from US$3 per barrel to nearly $12 globally; US prices were significantly higher. The embargo caused an **oil crisis**, or "shock", with many short- and long-term effects on global politics and the global economy. It was later called the **first oil shock**, followed by the 1979 **oil crisis**, termed the **second oil shock**.

---

<table>
<thead>
<tr>
<th>Rank</th>
<th>Model</th>
<th>EM</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Human Performance</td>
<td>82.304</td>
<td>91.221</td>
</tr>
<tr>
<td></td>
<td>Stanford University (Rajpurkar et al. '16)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>BERT (ensemble)</td>
<td>87.433</td>
<td>93.160</td>
</tr>
<tr>
<td></td>
<td>Google AI Language</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>BERT (single model)</td>
<td>85.083</td>
<td>91.835</td>
</tr>
<tr>
<td></td>
<td>Google AI Language</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>nllnet (ensemble)</td>
<td>85.954</td>
<td>91.677</td>
</tr>
<tr>
<td></td>
<td>Microsoft Research Asia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>nllnet (single model)</td>
<td>83.468</td>
<td>90.133</td>
</tr>
<tr>
<td></td>
<td>Microsoft Research Asia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>QANet (ensemble)</td>
<td>84.454</td>
<td>90.490</td>
</tr>
<tr>
<td></td>
<td>Google Brain &amp; CMU</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SWAG Results

A girl is going across a set of monkey bars. She
(i) jumps up across the monkey bars.
(ii) struggles onto the bars to grab her head.
(iii) gets to the end and stands on a wooden plank.
(iv) jumps up and does a back flip.
Effect of Model Size

- Big models help a lot
- Going from 110M -> 340M params helps even on datasets with 3,600 labeled examples
- Bigger model might even help more!
BERT is Open Sourced

- Both code and pre-trained models are available
  - Over 14,000 github stars
  - Multilingual models are also released

- Many articles and blog posts about BERT

- Large impact on other NLP tasks
  - ARC, CoQA, SQuAD 2.0, MSMARCO, OpenBookQA, SciTail, parsing

https://github.com/google-research/bert
Using BERT on TfHub
Install and Config BERT

- Install BERT

  ```
  pip install bert-tensorflow
  ```

- Import BERT

  ```
  import bert
  from bert import run_classifier
  from bert import optimization
  from bert import tokenization
  ```
Prepare Input for BERT

- For this task, we have only one text sequence

```python
bert.run_classifier.InputExample(guid=None,
    text_a = x[DATA_COLUMN],
    text_b = None,
    label = x[LABEL_COLUMN])
```
Input Format

● Construct BERT input

```
bert.run_classifier.convert_examples_to_features(train_ 
InputExamples, label_list, MAX_SEQ_LENGTH, tokenizer)
```

```
INFO:tensorflow:tokens: [CLS] the performances were fault #less and outstanding . [SEP]
INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1
INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0
INFO:tensorflow:label: 1 (id = 1)
```
Use BERT TfHub Module

● Load BERT Tf-Hub Module

```python
bert_module = hub.Module(BERT_MODEL_HUB, trainable=True)
bert_inputs = dict(input_ids=input_ids,
                   input_mask=input_mask,
                   segment_ids=segment_ids)
bert_outputs = bert_module(inputs=bert_inputs,
                           signature="tokens",
                           as_dict=True)
```
Specify the Output Layer

- Add the standard output layer for classification

```python
with tf.variable_scope("loss"):
    logits = tf.layers.dense(output_layer, num_classes)
    loss = tf.losses.sparse_softmax_cross_entropy(labels, logits)
    predictions = tf.argmax(logits, -1)
```
Training and Evaluating with BERT

● Getting results

```python
estimator.train(input_fn=train_input_fn, max_steps=num_train_steps)
...

estimator.evaluate(input_fn=test_input_fn, steps=None)
```

```python
{'auc': 0.86659324,
 'eval_accuracy': 0.8664,
 'f1_score': 0.8659711,
```
Conclusion

- BERT is a novel language representation model
- The pretrained models and code are released!
  - https://github.com/google-research/bert
- Questions?