FastBERT

Khalil Mrini, Henry Mao, Bodhisattwa Majumder
Motivation
Motivation

- Transformer models are really good at long term sequence modeling
- BERT has achieved astounding performance in the GLUE benchmark
Problem

- Transformer models are big and slow
- BERT has 110M parameters
- BERT Base is trained for 4 days on 16 TPUs (= 64 NVIDIA Titan Xp GPUs)
- BERT Large is trained for 4 days on 64 TPUs (= 256 NVIDIA Titan Xp GPUs)
- Maximum batch size of approximately 8 on a single GPU for any downstream tasks
Goal

A faster and more compact version of BERT with a minimal accuracy loss on downstream tasks.
Background
Transformers & BERT
Original Transformer Architecture

INPUT: Je suis étudiant

OUTPUT: I am a student
Original Transformer Architecture

INPUT: Je suis étudiant

OUTPUT: I am a student
An Encoder Layer

ENCODER

Feed Forward Neural Network

Self-Attention
Self-Attention

Multiplying $x_1$ by the $W_Q$ weight matrix produces $q_1$, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.
Self-Attention: In Detail

Input
- Embedding
- Queries
- Keys
- Values

Score
- Divide by $8 \cdot \sqrt{d_k}$

Softmax
- Softmax

Value
- Value

Sum

Thinking
- $x_1$
- $q_1$
- $k_1$
- $v_1$
- $q_1 \cdot k_1 = 112$
- $\frac{112}{8} = 14$
- $0.88$
- $v_1$
- $z_1$

Machines
- $x_2$
- $q_2$
- $k_2$
- $v_2$
- $q_1 \cdot k_2 = 96$
- $\frac{96}{8} = 12$
- $0.12$
- $v_2$
- $z_2$
Self-Attention: The Result
BERT: Bi-Encoder Rep. from Transformers

<table>
<thead>
<tr>
<th>Name</th>
<th>BERT\textsubscript{BASE}</th>
<th>BERT\textsubscript{LARGE}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layers</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>Hidden size</td>
<td>768</td>
<td>1024</td>
</tr>
<tr>
<td>Self-Attention Heads</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>Total Parameters</td>
<td>110M</td>
<td>340M</td>
</tr>
</tbody>
</table>

![Diagram of BERT models](image)
BERT: a Masked Language Model

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

Possible classes:
- 0.1% Aardvark
- 0.1% Improvisation
- 0.0% Zyzyva

FFNN + Softmax

Randomly mask 15% of tokens

Input

BERT's clever language modeling task masks 15% of words in the input and asks the model to predict the missing word.
BERT: Next Sentence Prediction

The second task BERT is pre-trained on is a two-sentence classification task. The tokenization is oversimplified in this graphic as BERT actually uses WordPieces as tokens rather than words --- so some words are broken down into smaller chunks.
BERT: Task-Specific Representations
Knowledge Distillation
Overview

Large model  Ensemble  Small model
Objective Function

\[ \mathcal{L}_{KD}(W_S) = \mathcal{H}(y_{true}, P_S) + \lambda \mathcal{H}(P_T^\tau, P_S^\tau) \]

\[ P_T^\tau = \text{softmax} \left( \frac{a_T}{\tau} \right), \quad P_S^\tau = \text{softmax} \left( \frac{a_S}{\tau} \right) \]
Soft Targets

\[ \mathcal{L}_{KD}(W_S) = \mathcal{H}(y_{true}, P_S) + \lambda \mathcal{H}(P_T^\tau, P_S^\tau) \]

\[ P_T^\tau = \text{softmax} \left( \frac{a_T}{\tau} \right), \quad P_S^\tau = \text{softmax} \left( \frac{a_S}{\tau} \right) \]
An Extension: FitNets

(a) Teacher and Student Networks

(b) Hints Training

(c) Knowledge Distillation

\[ \mathcal{L}_{HT}(W_{Guided}, W_r) = \frac{1}{2} \| u_h(x; W_{Hint}) - r(v_g(x; W_{Guided}); W_r) \|^2, \]
Experiments
Speed Breakdown of BERT

- cls: Mean 1.28%
- bert: Mean 98.68%
- bert.encoder: 92.94%
- bert.encoder.layer: 7.21% (x 12)
- bert.encoder.layer.attention: 4.14%
- bert.encoder.layer.pointwise feedforward: 3.03%

Attention took the most time but Pointwise Feedforward was a low hanging fruit for optimization.
FastBERT

Exactly the same as BERT but

- 768 embedding => 512
- 3072 pointwise feedforward => 2048
- 110 M parameters => 50 M parameters

To compensate for smaller model capacity:

- No dropout
- No L2 weight decay
FastBERT Objective

\[ \mathcal{L}(W_{\text{student}}) = \tau^2 \mathcal{L}_{KD}(W_{\text{student}}) + \eta \mathcal{L}_{HT}(W_{\text{guided}}, W_r) \]

- Temperature = 3 decayed to 1
- Hint weight = 100 decayed to 0

Hyperparameters
Dataset for KD

- Original BERT was trained on Wikipedia and Bookcorpus (1 B words) datasets
- For knowledge distillation, we use a cleaned subset of wikipedia dump named Wikitext 103.
- Wikitext 103 is a collection over 100 million tokens
Experiments

Compared several baselines

1. Original BERT
2. 25% weight pruned BERT - We prune the lowest 25% weights in each layer to zero. This is to understand the amount of redundancy in the BERT architecture.
3. Pretraining FastBERT architecture - We pretrain using the same Wikitext 103 dataset from starch with the FastBERT architecture. This is an ablation study to see how KD is a better option for model compression.
4. FastBERT by KD - We use KD. Trained on single GPU for 40 hours.
Downstream tasks - GLUE Benchmark

- To analyze the efficacy of FastBERT, we consider baselines as the original performance by BERT on a set of downstream tasks
- GLUE is General Language Understanding Evaluation - popular benchmark for many NLU tasks
- For benchmarking, we use the GLUE datasets and consider following tasks -
  - Sentiment Analysis task - SST-2
  - Paraphrase comparison - Quora Question Pair (QQP), MRPC
  - Textual Entailment - MNLI-matched, RTE
- We keep the hyperparameters fixed for all the tasks while comparing FastBERT and BERT
## Accuracy comparison

<table>
<thead>
<tr>
<th>Task</th>
<th>Size</th>
<th>BERT</th>
<th>BERT 25% Weight Pruning</th>
<th>FastBERT - pretrained</th>
<th>FastBERT - KD</th>
</tr>
</thead>
<tbody>
<tr>
<td>SST-2</td>
<td>67K</td>
<td>92.08</td>
<td>91.90</td>
<td>78.44</td>
<td>90.06</td>
</tr>
<tr>
<td>QQP</td>
<td>364K</td>
<td>91.04</td>
<td>90.47</td>
<td>82.02</td>
<td>89.52</td>
</tr>
<tr>
<td>MRPC*</td>
<td>3.7K</td>
<td>66.49</td>
<td>65.81</td>
<td>66.49</td>
<td>77.21</td>
</tr>
<tr>
<td>RTE*</td>
<td>2.5K</td>
<td>47.29</td>
<td>47.05</td>
<td>47.29</td>
<td>60.61</td>
</tr>
<tr>
<td>MNLI</td>
<td>393K</td>
<td>83.99</td>
<td>82.87</td>
<td>63.02</td>
<td>79.51</td>
</tr>
<tr>
<td>Overall</td>
<td></td>
<td>76.178</td>
<td>75.62</td>
<td>67.452</td>
<td>79.382</td>
</tr>
</tbody>
</table>
## Speed comparison

<table>
<thead>
<tr>
<th>Task</th>
<th>BERT</th>
<th>FastBERT - KD</th>
<th>Speedup*</th>
</tr>
</thead>
<tbody>
<tr>
<td>SST-2</td>
<td>14 mins 18 secs</td>
<td>8 mins 05 secs</td>
<td>1.77</td>
</tr>
<tr>
<td>QQP</td>
<td>3hr 51 mins</td>
<td>2hr 13 mins</td>
<td>1.73</td>
</tr>
<tr>
<td>MRPC</td>
<td>3 min 21 secs</td>
<td>1 min 38 secs</td>
<td>2.05</td>
</tr>
<tr>
<td>RTE</td>
<td>1 min 38 secs</td>
<td>59 secs</td>
<td>1.66</td>
</tr>
<tr>
<td>MNLI</td>
<td>4hr 9 mins</td>
<td>2hr 24 mins</td>
<td>1.73</td>
</tr>
</tbody>
</table>

*Speedup factor varies across tasks due to varying time taken by data loading even if training FastBERT achieves same speedup
Loss stability with Hints

KD loss without hints

KD loss with hints

Hint loss

smoother!
Conclusions

- KD works fairly well on transformer architectures
- Hints actually help accelerate training
- Temperature decay makes convergence more stable
Future Work

- 2 weeks plan: Deeper but thinner networks
- 4 weeks plan: Optimized attention architectures
  - CSAN
  - Local attention
  - CNN + Attention