Bert Family Tree

Maury Courtland (PhD; www.maury.science)

March 10, 2020

Talk Overview

Talk Overview

- BERT review

Talk Overview

- BERT review
- RoBERTa

Talk Overview

- BERT review
- RoBERTa
- DistilBERT

Talk Overview

- BERT review
- RoBERTa
- DistilBERT
- ALBERT

Devlin et al. 2018: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Background and Approach

- Pre-trained LMs improve performance on many NLP tasks (both sentence and token level)

Background and Approach

- Pre-trained LMs improve performance on many NLP tasks (both sentence and token level)
- 2 current approaches:

Background and Approach

- Pre-trained LMs improve performance on many NLP tasks (both sentence and token level)
- 2 current approaches:
  1) feature-based: use (frozen) pre-trained models as feature encoders and let downstream models learn the task, similar to embeddings

Background and Approach

- Pre-trained LMs improve performance on many NLP tasks (both sentence and token level)
- 2 current approaches:
  1) feature-based: use (frozen) pre-trained models as feature encoders and let downstream models learn the task, similar to embeddings
  2) fine-tuning: fine-tune pre-trained model parameters directly on downstream task

Background and Approach

- Pre-trained LMs improve performance on many NLP tasks (both sentence and token level)
- 2 current approaches:
    1) feature-based: use (frozen) pre-trained models as feature encoders and let downstream models learn the task, similar to embeddings
    2) fine-tuning: fine-tune pre-trained model parameters directly on downstream task
    - same pre-train objective: "unidirectional" LMs, which limits architecture choices

Strengths

- Defines new training objective: “masked language model” (inspired by Cloze)

Strengths

- Defines new training objective: “masked language model” (inspired by Cloze)
  - Allows new architectures (here, deep bidirectional Transformer) given freedom from unidirectionality LM constraint (**BIG DEPARTURE** from past approaches)

Strengths

- Defines new training objective: “masked language model” (inspired by Cloze)
  - Allows new architectures (here, deep bidirectional Transformer) given freedom from unidirectionality LM constraint (**BIG DEPARTURE** from past approaches)
  - Randomly masks some input tokens (15%), task is to predict vocab ID of those tokens

Strengths

- Defines new training objective: "masked language model" (inspired by Cloze)
  - Allows new architectures (here, deep bidirectional Transformer) given freedom from unidirectionality LM constraint (**BIG DEPARTURE** from past approaches)
  - Randomly masks some input tokens (15%), task is to predict vocab ID of those tokens
  - Ex. from original Taylor, 1953: "Chickens cackle and ______ quack"

Strengths

- Defines new training objective: “masked language model” (inspired by Cloze)
  - Allows new architectures (here, deep bidirectional Transformer) given freedom from unidirectionality LM constraint (**BIG DEPARTURE** from past approaches)
  - Randomly masks some input tokens (15%), task is to predict vocab ID of those tokens
  - Ex. from original Taylor, 1953: “Chickens cackle and ______ quack”
- Avoids training and engineering time for task-specific architectures

Design Philosophy

- Have minimal architectural changes from pre-trained to downstream task (minimal parameters learned from scratch)

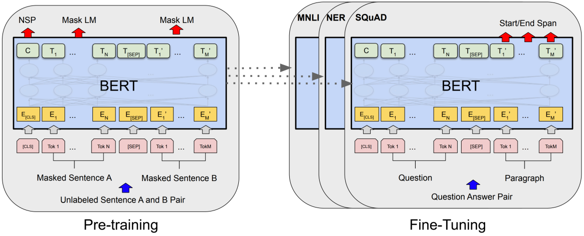

Figure 1: Applying pre-trained BERT to downstream tasks

Methods

- Built out of transformers (from "Attention Is All You Need" (2017)):

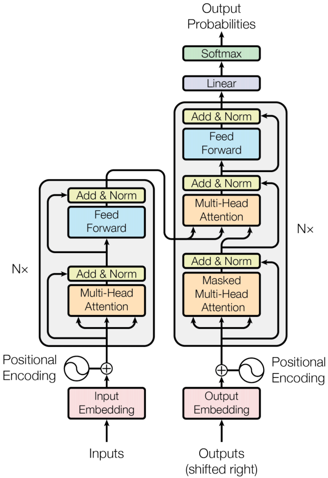

Figure 2: Transformer architecture

Inputs

- Use WordPiece embeddings (essentially sub-word features) with 30k vocab, denoted E_{token}

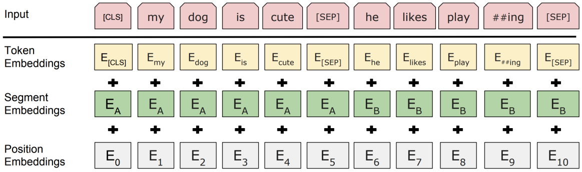

Figure 3: BERT input representation

Inputs

- Use WordPiece embeddings (essentially sub-word features) with 30k vocab, denoted E_{token}
  - Add segment embedding (E_A or E_B) and position embeddings (E_i) to token embeddings

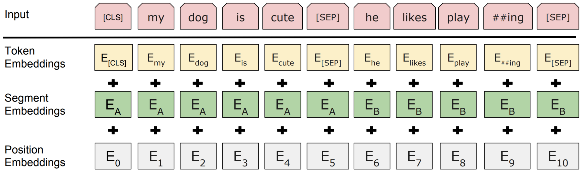

Figure 3: BERT input representation

Inputs

- Use WordPiece embeddings (essentially sub-word features) with 30k vocab, denoted E_{token}
  - Add segment embedding (E_A or E_B) and position embeddings (E_i) to token embeddings
- Use "aggregate sequence representation for classification tasks" token [CLS] (final hidden vector C)

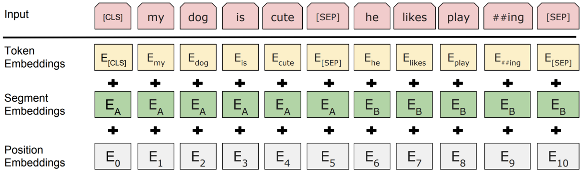

Figure 3: BERT input representation

Inputs

- Use WordPiece embeddings (essentially sub-word features) with 30k vocab, denoted E_{token}
  - Add segment embedding (E_A or E_B) and position embeddings (E_i) to token embeddings
- Use "aggregate sequence representation for classification tasks" token [CLS] (final hidden vector C)
- Can handle either a single sentence or a pair (e.g. <Q, A>), separated by [SEP] token (final hidden vector $T_{[SEP]}$)

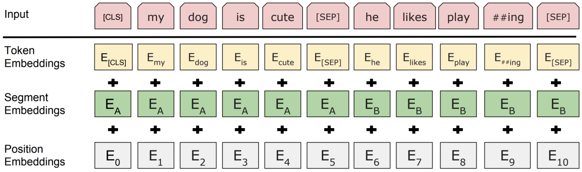

Figure 3: BERT input representation

Outputs (training objective): Masked LM

- Final hidden vectors for masked tokens at final transformer layer

Outputs (training objective): Masked LM

- Final hidden vectors for masked tokens at final transformer layer
- Fed into output softmax over vocab (like normal) and minimize cross-entropy loss

Outputs (training objective): Masked LM

- Final hidden vectors for masked tokens at final transformer layer
- Fed into output softmax over vocab (like normal) and minimize cross-entropy loss
- *BUT*, this creates mismatch between pre-training and fine-tuning (directly against design philosophy)

Outputs (training objective): Masked LM

- Final hidden vectors for masked tokens at final transformer layer
- Fed into output softmax over vocab (like normal) and minimize cross-entropy loss
- *BUT*, this creates mismatch between pre-training and fine-tuning (directly against design philosophy)
  - Solution: don't always [MASK] tokens

Outputs (training objective): Masked LM

- Final hidden vectors for masked tokens at final transformer layer
- Fed into output softmax over vocab (like normal) and minimize cross-entropy loss
- *BUT*, this creates mismatch between pre-training and fine-tuning (directly against design philosophy)
  - Solution: don't always [MASK] tokens
  - To remedy, they use 80% [MASK], 10% random token, 10% unmasked correct token

Outputs: Next Sentence Prediction

- Masked LM is good for token-level tasks, but downstream fine-tuning tasks have relationships between sentences (e.g. question answering, natural language inference, etc.)

Outputs: Next Sentence Prediction

- Masked LM is good for token-level tasks, but downstream fine-tuning tasks have relationships between sentences (e.g. question answering, natural language inference, etc.)
- Therefore we need a way to evaluate and embed sentence relationships

Outputs: Next Sentence Prediction

- Masked LM is good for token-level tasks, but downstream fine-tuning tasks have relationships between sentences (e.g. question answering, natural language inference, etc.)
- Therefore we need a way to evaluate and embed sentence relationships
- Create an output hidden vector from a special appended [CLS] token and (binary) classify whether the 2 training sentences are related or not (final accuracy is 97+% on NSP)

Outputs: Next Sentence Prediction

- Masked LM is good for token-level tasks, but downstream fine-tuning tasks have relationships between sentences (e.g. question answering, natural language inference, etc.)
- Therefore we need a way to evaluate and embed sentence relationships
- Create an output hidden vector from a special appended [CLS] token and (binary) classify whether the 2 training sentences are related or not (final accuracy is 97+% on NSP)
  - 50% of the time, sample adjacent sentences from the corpus (i.e. related)

Outputs: Next Sentence Prediction

- Masked LM is good for token-level tasks, but downstream fine-tuning tasks have relationships between sentences (e.g. question answering, natural language inference, etc.)
- Therefore we need a way to evaluate and embed sentence relationships
- Create an output hidden vector from a special appended [CLS] token and (binary) classify whether the 2 training sentences are related or not (final accuracy is 97+% on NSP)
  - 50% of the time, sample adjacent sentences from the corpus (i.e. related)
  - 50% of the time, sample random sentences from the corpus (i.e. unrelated)

Outputs: Next Sentence Prediction

- Masked LM is good for token-level tasks, but downstream fine-tuning tasks have relationships between sentences (e.g. question answering, natural language inference, etc.)
- Therefore we need a way to evaluate and embed sentence relationships
- Create an output hidden vector from a special appended [CLS] token and (binary) classify whether the 2 training sentences are related or not (final accuracy is 97+% on NSP)
  - 50% of the time, sample adjacent sentences from the corpus (i.e. related)
  - 50% of the time, sample random sentences from the corpus (i.e. unrelated)
- Simple fix, but boosts performance across the board (esp. for QNLI: 3.5% and SQuAD:.6%)

Fine-tuning

- Attention mechanisms make BERT easily extensible and can capture many tasks (rather than learning representations from scratch, just change weighting of subspace features)

Fine-tuning

- Attention mechanisms make BERT easily extensible and can capture many tasks (rather than learning representations from scratch, just change weighting of subspace features)
- Just plug-n-play: use whichever task-specific inputs and outputs you want and fine-tune **all** the parameters

Fine-tuning

- Attention mechanisms make BERT easily extensible and can capture many tasks (rather than learning representations from scratch, just change weighting of subspace features)
- Just plug-n-play: use whichever task-specific inputs and outputs you want and fine-tune **all** the parameters
  - Input: Use A/B sentence pairs (e.g. sentence and paraphrase, hypothesis and premise, etc.) OR just a dummy null B sentence (e.g. text classification, sequence tagging)

Fine-tuning

- Attention mechanisms make BERT easily extensible and can capture many tasks (rather than learning representations from scratch, just change weighting of subspace features)
- Just plug-n-play: use whichever task-specific inputs and outputs you want and fine-tune **all** the parameters
  - Input: Use A/B sentence pairs (e.g. sentence and paraphrase, hypothesis and premise, etc.) OR just a dummy null B sentence (e.g. text classification, sequence tagging)
  - Output for token-tasks: use token hidden vectors as input for token-level tasks (e.g. sequence tagging, QA)

Fine-tuning

- Attention mechanisms make BERT easily extensible and can capture many tasks (rather than learning representations from scratch, just change weighting of subspace features)
- Just plug-n-play: use whichever task-specific inputs and outputs you want and fine-tune **all** the parameters
  - Input: Use A/B sentence pairs (e.g. sentence and paraphrase, hypothesis and premise, etc.) OR just a dummy null B sentence (e.g. text classification, sequence tagging)
  - Output for token-tasks: use token hidden vectors as input for token-level tasks (e.g. sequence tagging, QA)
  - Output for sentence-tasks: Use C for sentence-level tasks (e.g. entailment, sentiment analysis)

Fine-tuning

- Attention mechanisms make BERT easily extensible and can capture many tasks (rather than learning representations from scratch, just change weighting of subspace features)
- Just plug-n-play: use whichever task-specific inputs and outputs you want and fine-tune **all** the parameters
  - Input: Use A/B sentence pairs (e.g. sentence and paraphrase, hypothesis and premise, etc.) OR just a dummy null B sentence (e.g. text classification, sequence tagging)
  - Output for token-tasks: use token hidden vectors as input for token-level tasks (e.g. sequence tagging, QA)
  - Output for sentence-tasks: Use C for sentence-level tasks (e.g. entailment, sentiment analysis)
- Relatively inexpensive (all paper tasks fine-tuned in a few hours on 1 GPU)

Results: Grand-slam SOTA

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Figure 4: SOTA across the board with some impressive gains

Ablation studies takeaways

- NSP fairly important (especially for QA and NLI)

Ablation studies takeaways

- NSP fairly important (especially for QA and NLI)
- Deep bidirectionality very important

Ablation studies takeaways

- NSP fairly important (especially for QA and NLI)
- Deep bidirectionality very important
  - Also cheaper than separate unidirectional models

Ablation studies takeaways

- NSP fairly important (especially for QA and NLI)
- Deep bidirectionality very important
  - Also cheaper than separate unidirectional models
  - And strictly more powerful than unidirectional models

Ablation studies takeaways

- NSP fairly important (especially for QA and NLI)
- Deep bidirectionality very important
  - Also cheaper than separate unidirectional models
  - And strictly more powerful than unidirectional models
- More parameters = better (sometimes *much* = 8%, sometimes *a bit* .4%)

Ablation studies takeaways

- NSP fairly important (especially for QA and NLI)
- Deep bidirectionality very important
  - Also cheaper than separate unidirectional models
  - And strictly more powerful than unidirectional models
- More parameters = better (sometimes *much* = 8%, sometimes *a bit* .4%)
- ELMo-ish approach of concatenating parameters as features is worse (though not by much, best linear combo is .3 worse on 96.1 F1) than a fine-tuning approach (which again, is generally cheaper and more portable)

Liu et al. 2019: RoBERTa: A Robustly Optimized BERT Pretraining Approach

Background and Approach

- Lots of other models have been beating BERT's SOTA results

Background and Approach

- Lots of other models have been beating BERT's SOTA results
- These models differ along many dimensions

Background and Approach

- Lots of other models have been beating BERT's SOTA results
- These models differ along many dimensions
- Therefore, the results comparisons are apples to oranges and BERT needs a fair chance

Tweaks from BERT

- Dynamically change masking pattern each time a sentence is encountered rather than generating a few variations of what is masked for each input.

Tweaks from BERT

- Dynamically change masking pattern each time a sentence is encountered rather than generating a few variations of what is masked for each input.
- Remove NSP task (unnecessary in their findings)

Tweaks from BERT

- Dynamically change masking pattern each time a sentence is encountered rather than generating a few variations of what is masked for each input.
- Remove NSP task (unnecessary in their findings)
- Train on max-length (or almost) sequences during the whole training, rather than beginning on short sentences and only training on long sentences at the very end

Tweaks from BERT

- Dynamically change masking pattern each time a sentence is encountered rather than generating a few variations of what is masked for each input.
- Remove NSP task (unnecessary in their findings)
- Train on max-length (or almost) sequences during the whole training, rather than beginning on short sentences and only training on long sentences at the very end
- Trains model much longer with much bigger batches (shown to be very beneficial) and with more data (*always* good)

Dynamic Masking Results

- Slightly better than static (though not across the board), **BUT** worth noting that it prevents overfitting so with dynamic masking the model probably could have kept training

Masking	SQuAD 2.0	MNLI-m	SST-2
reference	76.3	84.3	92.8
Our reimplementation:			
static	78.3	84.3	92.5
dynamic	78.7	84.0	92.9

What to choose for input samples?

- SEGMENT-PAIR+NSP (BERT clone): Segments (possibly many sentences) $<=$ 512 tokens with an NSP loss

What to choose for input samples?

- SEGMENT-PAIR+NSP (BERT clone): Segments (possibly many sentences) $<=$ 512 tokens with an NSP loss
- SENTENCE-PAIR+NSP: Same as above, but with a pair of natural sentences. Given much shorter input length, increase batch size.

What to choose for input samples?

- SEGMENT-PAIR+NSP (BERT clone): Segments (possibly many sentences) $<=$ 512 tokens with an NSP loss
- SENTENCE-PAIR+NSP: Same as above, but with a pair of natural sentences. Given much shorter input length, increase batch size.
- FULL-SENTENCES: Pack contiguous full sentences (possibly across doc boundaries with an extra [SEP] token) $<=$ 512 tokens. No NSP loss.

What to choose for input samples?

- SEGMENT-PAIR+NSP (BERT clone): Segments (possibly many sentences) $<=$ 512 tokens with an NSP loss
- SENTENCE-PAIR+NSP: Same as above, but with a pair of natural sentences. Given much shorter input length, increase batch size.
- FULL-SENTENCES: Pack contiguous full sentences (possibly across doc boundaries with an extra [SEP] token) $<=$ 512 tokens. No NSP loss.
- DOC-SENTENCES: Same as above, but inputs cannot cross document boundaries

Input Encoding Results

- NSP doesn't seem to make a large difference. Longer inputs are definitely better.

Model	SQuAD 1.1/2.0	MNLI-m	SST-2	RACE
Our reimplementation (with NSP loss):				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
Our reimplementation (without NSP loss):				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6
$\text{BERT}_{\text{BASE}}$	88.5/76.3	84.3	92.8	64.3
$\text{XLNet}_{\text{BASE}}$ (K = 7)	–/81.3	85.8	92.7	66.1
$\text{XLNet}_{\text{BASE}}$ (K = 6)	–/81.0	85.6	93.4	66.7

Results of Batch Size (Goldilocks)

bsz	steps	lr	ppl	MNLI-m	SST-2
256	1M	1e-4	3.99	84.7	92.7
2K	125K	7e-4	**3.68**	**85.2**	**92.9**
8K	31K	1e-3	3.77	84.6	92.8

Table 3: Perplexity on held-out training data (*ppl*) and development set accuracy for base models trained over BOOKCORPUS and WIKIPEDIA with varying batch sizes (*bsz*). We tune the learning rate (*lr*) for each setting. Models make the same number of passes over the data (epochs) and have the same computational cost.

Results of Training Corpus and Length

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

Figure 6: Bigger and Longer Training is Better

Downstream Fine-tuning Results

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
$BERT_{LARGE}$	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
$XLNet_{LARGE}$	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	**90.2/90.2**	**94.7**	**92.2**	**86.6**	**96.4**	**90.9**	**68.0**	**92.4**	**91.3**	-
Ensembles on test (from leaderboard as of July 25, 2019)										
ALICE	88.2/87.9	95.7	**90.7**	83.5	95.2	92.6	**68.6**	91.1	80.8	86.3
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2/89.8	98.6	90.3	86.3	**96.8**	**93.0**	67.8	91.6	**90.4**	88.4
RoBERTa	**90.8/90.2**	**98.9**	90.2	**88.2**	96.7	92.3	67.8	**92.2**	89.0	**88.5**

- SQuAD and RACE SOTA too!

Sanh et al. 2018: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter

Background and Approach

- BERT is pretty big and slow

Background and Approach

- BERT is pretty big and slow
- Network sizes are getting ridiculous (looking at you Microsoft's Turing-NLG: **17B** parameters)

Background and Approach

- BERT is pretty big and slow
- Network sizes are getting ridiculous (looking at you Microsoft's Turing-NLG: **17B** parameters)
  - Are all those parameters really necessary?

Background and Approach

- BERT is pretty big and slow
- Network sizes are getting ridiculous (looking at you Microsoft's Turing-NLG: **17B** parameters)
  - Are all those parameters really necessary?
- Maybe we can "compress" the performance with knowledge distillation (Hinton)

Background and Approach

- BERT is pretty big and slow
- Network sizes are getting ridiculous (looking at you Microsoft's Turing-NLG: **17B** parameters)
  - Are all those parameters really necessary?
- Maybe we can "compress" the performance with knowledge distillation (Hinton)
  - Smaller size would be good for edge computing

Knowledge Distillation

- Rather than predict on 1-hot classification, predict to match distribution of "teacher" model

Knowledge Distillation

- Rather than predict on 1-hot classification, predict to match distribution of "teacher" model
  - Calculates distillation loss as: $L_{ce} = \sum_i t_i * log(s_i)$, where t_i and s_i are prob. of teacher and student

Knowledge Distillation

- Rather than predict on 1-hot classification, predict to match distribution of "teacher" model
  - Calculates distillation loss as: $L_{ce} = \sum_i t_i * log(s_i)$, where t_i and s_i are prob. of teacher and student
  - Gives richer prediction signal than just 1-hot answer (continuous targets $\in (0, 1)$ for **all** labels)

Knowledge Distillation

- Rather than predict on 1-hot classification, predict to match distribution of "teacher" model
  - Calculates distillation loss as: $L_{ce} = \sum_i t_i * log(s_i)$, where t_i and s_i are prob. of teacher and student
  - Gives richer prediction signal than just 1-hot answer (continuous targets $\in (0, 1)$ for **all** labels)
  - Allows a "student" network to mimic the "teacher" model (monkey see, monkey do)

Knowledge Distillation

- Rather than predict on 1-hot classification, predict to match distribution of "teacher" model
  - Calculates distillation loss as: $L_{ce} = \sum_i t_i * log(s_i)$, where t_i and s_i are prob. of teacher and student
  - Gives richer prediction signal than just 1-hot answer (continuous targets $\in (0, 1)$ for **all** labels)
  - Allows a "student" network to mimic the "teacher" model (monkey see, monkey do)
- Can be done just during training of the base MLM task (madlib) or during the downstream task as well (e.g. NER, NLI, etc.)

Knowledge Distillation

- Rather than predict on 1-hot classification, predict to match distribution of "teacher" model
  - Calculates distillation loss as: $L_{ce} = \sum_i t_i * log(s_i)$, where t_i and s_i are prob. of teacher and student
  - Gives richer prediction signal than just 1-hot answer (continuous targets $\in (0, 1)$ for **all** labels)
  - Allows a "student" network to mimic the "teacher" model (monkey see, monkey do)
- Can be done just during training of the base MLM task (madlib) or during the downstream task as well (e.g. NER, NLI, etc.)
- Add in Cosine Similarity loss between the final hidden state vectors before the classifier

Architecture and Approach

- Same general architecture as BERT but

Architecture and Approach

- Same general architecture as BERT but
  - Take *every other* attention layer for warm start (reducing network size by $\approx \frac{1}{2}$)

Architecture and Approach

- Same general architecture as BERT but
  - Take *every other* attention layer for warm start (reducing network size by $\approx \frac{1}{2}$)
  - Get rid of token-type embeddings for input (i.e. segment "A" or "B")

Architecture and Approach

- Same general architecture as BERT but
  - Take *every other* attention layer for warm start (reducing network size by $\approx \frac{1}{2}$)
  - Get rid of token-type embeddings for input (i.e. segment "A" or "B")
  - Get rid of pooler layer (linear $tanh()$ square-matrix layer post-encoder pre-classifier)

Architecture and Approach

- Same general architecture as BERT but
  - Take *every other* attention layer for warm start (reducing network size by $\approx \frac{1}{2}$)
  - Get rid of token-type embeddings for input (i.e. segment "A" or "B")
  - Get rid of pooler layer (linear $tanh()$ square-matrix layer post-encoder pre-classifier)
- Train using improvements of RoBERTa but with original BERT corpus

Results

- Reduce size (and storage footprint) by **40%** and make inference time **60%** faster

Results

- Reduce size (and storage footprint) by **40%** and make inference time **60%** faster
- Retain up to **97%** of performance

Model	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

Ablation Studies

- Distillation loss and a warm start are **key**

Ablation	**Variation on GLUE macro-score**
$\emptyset$ - L_{cos} - L_{mlm}	-2.96
L_{ce} - $\emptyset$ - L_{mlm}	-1.46
L_{ce} - L_{cos} - $\emptyset$	-0.31
Triple loss + random weights initialization	-3.69

Ablation Studies

- Distillation loss and a warm start are **key**
- Cosine similarity loss is fairly important

Ablation	**Variation on GLUE macro-score**
$\emptyset$ - L_{cos} - L_{mlm}	-2.96
L_{ce} - $\emptyset$ - L_{mlm}	-1.46
L_{ce} - L_{cos} - $\emptyset$	-0.31
Triple loss + random weights initialization	-3.69

Ablation Studies

- Distillation loss and a warm start are **key**
- Cosine similarity loss is fairly important
- MLM is not that important

Ablation	**Variation on GLUE macro-score**
$\emptyset$ - L_{cos} - L_{mlm}	-2.96
L_{ce} - $\emptyset$ - L_{mlm}	-1.46
L_{ce} - L_{cos} - $\emptyset$	-0.31
Triple loss + random weights initialization	-3.69

Lan et al. 2020: ALBERT: A Lite BERT for Self-supervised Learning of Language Representations

Background and Approach

- Network sizes are getting ridiculous (*still* looking at you Microsoft's Turing-NLG's **17B** parameters) but show good performance

Background and Approach

- Network sizes are getting ridiculous (*still* looking at you Microsoft's Turing-NLG's **17B** parameters) but show good performance
  - Are all those parameters really necessary?

Background and Approach

- Network sizes are getting ridiculous (*still* looking at you Microsoft's Turing-NLG's **17B** parameters) but show good performance
  - Are all those parameters really necessary?
- Maybe we can share parameters across layers and factor some large matrices

Background and Approach

- Network sizes are getting ridiculous (*still* looking at you Microsoft's Turing-NLG's **17B** parameters) but show good performance
  - Are all those parameters really necessary?
- Maybe we can share parameters across layers and factor some large matrices
  - Smaller size would be good for edge computing

Tweaks from BERT

1. Factorizing embedding matrices (which is the largest matrix in the network, other than its mirror in the classifier)

Tweaks from BERT

1. Factorizing embedding matrices (which is the largest matrix in the network, other than its mirror in the classifier)
2. Parameter sharing across layers (which saves several square matrices each time)

Tweaks from BERT

1. Factorizing embedding matrices (which is the largest matrix in the network, other than its mirror in the classifier)
2. Parameter sharing across layers (which saves several square matrices each time)
3. NSP swap out for Sentence Order Prediction (bring back the sentence relationship training objective!)

Embedding matrix factorization

- BERT (and RoBERTa) tie embedding size (E) to hidden space dimensionality (H, generally 768)

Embedding matrix factorization

- BERT (and RoBERTa) tie embedding size (E) to hidden space dimensionality (H, generally 768)
- Conceptually, conflates 2 concepts:

Embedding matrix factorization

- BERT (and RoBERTa) tie embedding size (E) to hidden space dimensionality (H, generally 768)
- Conceptually, conflates 2 concepts:
    1. The word embedding which is meant to capture a *context-indepedent* representation

Embedding matrix factorization

- BERT (and RoBERTa) tie embedding size (E) to hidden space dimensionality (H, generally 768)
- Conceptually, conflates 2 concepts:
  1. The word embedding which is meant to capture a *context-indepedent* representation
  2. Hidden layer embeddings are meant to capture a *context-dependent* representation

Embedding matrix factorization

- BERT (and RoBERTa) tie embedding size (E) to hidden space dimensionality (H, generally 768)
- Conceptually, conflates 2 concepts:
  1. The word embedding which is meant to capture a *context-indepedent* representation
  2. Hidden layer embeddings are meant to capture a *context-dependent* representation
- Practically, given the necessity of the hidden space to capture complex interactions between multiple words, we would want $H >> E$

Embedding matrix factorization

- BERT (and RoBERTa) tie embedding size (E) to hidden space dimensionality (H, generally 768)
- Conceptually, conflates 2 concepts:
  1. The word embedding which is meant to capture a *context-indepedent* representation
  2. Hidden layer embeddings are meant to capture a *context-dependent* representation
- Practically, given the necessity of the hidden space to capture complex interactions between multiple words, we would want $H >> E$
- Additionally, $|V|$ is really large so increasing E takes many parameters (only sparsely updated during training)

Embedding matrix factorization results

- So take the $V \times H$ matrix and decompose into $V \times E$ and $E \times H$ matrices

Model	E	Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
ALBERT base not-shared	64	87M	89.9/82.9	80.1/77.8	82.9	91.5	66.7	81.3
	128	89M	89.9/82.8	80.3/77.3	83.7	91.5	67.9	81.7
	256	93M	90.2/83.2	80.3/77.4	84.1	91.9	67.3	81.8
	768	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3
ALBERT base all-shared	64	10M	88.7/81.4	77.5/74.8	80.8	89.4	63.5	79.0
	128	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1
	256	16M	88.8/81.5	79.1/76.3	81.5	90.3	63.4	79.6
	768	31M	88.6/81.5	79.2/76.6	82.0	90.6	63.3	79.8

Table 3: The effect of vocabulary embedding size on the performance of ALBERT-base.

Embedding matrix factorization results

- So take the $V \times H$ matrix and decompose into $V \times E$ and $E \times H$ matrices
  - Reduces $O(V \times H)$ to $O(V \times E + E \times H)$

Model	E	Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
ALBERT base not-shared	64	87M	89.9/82.9	80.1/77.8	82.9	91.5	66.7	81.3
	128	89M	89.9/82.8	80.3/77.3	83.7	91.5	67.9	81.7
	256	93M	90.2/83.2	80.3/77.4	84.1	91.9	67.3	81.8
	768	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3
ALBERT base all-shared	64	10M	88.7/81.4	77.5/74.8	80.8	89.4	63.5	79.0
	128	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1
	256	16M	88.8/81.5	79.1/76.3	81.5	90.3	63.4	79.6
	768	31M	88.6/81.5	79.2/76.6	82.0	90.6	63.3	79.8

Table 3: The effect of vocabulary embedding size on the performance of ALBERT-base.

Embedding matrix factorization results

- So take the $V \times H$ matrix and decompose into $V \times E$ and $E \times H$ matrices
  - Reduces $O(V \times H)$ to $O(V \times E + E \times H)$
  - Can be significant with large H (especially with shared parameters)

Model	E	Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
ALBERT base not-shared	64	87M	89.9/82.9	80.1/77.8	82.9	91.5	66.7	81.3
	128	89M	89.9/82.8	80.3/77.3	83.7	91.5	67.9	81.7
	256	93M	90.2/83.2	80.3/77.4	84.1	91.9	67.3	81.8
	768	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3
ALBERT base all-shared	64	10M	88.7/81.4	77.5/74.8	80.8	89.4	63.5	79.0
	128	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1
	256	16M	88.8/81.5	79.1/76.3	81.5	90.3	63.4	79.6
	768	31M	88.6/81.5	79.2/76.6	82.0	90.6	63.3	79.8

Table 3: The effect of vocabulary embedding size on the performance of ALBERT-base.

Parameter sharing results

- All-shared is always non-optimal, but also has *way fewer* parameters (and trains faster)

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
ALBERT base E=768	all-shared	31M	88.6/81.5	79.2/76.6	82.0	90.6	63.3	79.8
	shared-attention	83M	89.9/82.7	80.0/77.2	84.0	91.4	67.7	81.6
	shared-FFN	57M	89.2/82.1	78.2/75.4	81.5	90.8	62.6	79.5
	not-shared	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3
ALBERT base E=128	all-shared	12M	89.3/82.3	80.0/77.1	82.0	90.3	64.0	80.1
	shared-attention	64M	89.9/82.8	80.7/77.9	83.4	91.9	67.6	81.7
	shared-FFN	38M	88.9/81.6	78.6/75.6	82.3	91.7	64.4	80.2
	not-shared	89M	89.9/82.8	80.3/77.3	83.2	91.5	67.9	81.6

Table 4: The effect of cross-layer parameter-sharing strategies, ALBERT-base configuration.

Sentence Order Prediction Results

- SOP is the best objective and seems to implicitly learn NSP (to a degree)

	Intrinsic Tasks			Downstream Tasks					
SP tasks	MLM	NSP	SOP	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
None	54.9	52.4	53.3	88.6/81.5	78.1/75.3	81.5	89.9	61.7	79.0
NSP	54.5	90.5	52.0	88.4/81.5	77.2/74.6	81.6	**91.1**	62.3	79.2
SOP	54.0	78.9	86.5	**89.3/82.3**	**80.0/77.1**	**82.0**	90.3	**64.0**	**80.1**

Table 5: The effect of sentence-prediction loss, NSP vs. SOP, on intrinsic and downstream tasks.

Overall Results

- Train on BERT corpus with $<=$512 segment encoding and A/B embeddings with 30k vocabulary

Overall Results

- Train on BERT corpus with $<=$512 segment encoding and A/B embeddings with 30k vocabulary
- 18x fewer parameters

Overall Results

- Train on BERT corpus with $<=$512 segment encoding and A/B embeddings with 30k vocabulary
- 18x fewer parameters
- 1.7x faster training time

Overall Results

- Train on BERT corpus with <=512 segment encoding and A/B embeddings with 30k vocabulary
- 18x fewer parameters
- 1.7x faster training time
  - **BUT** slower inference time

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	**94.1/88.3**	**88.1/85.1**	**88.0**	**95.2**	**82.3**	**88.7**	0.3x

Takeaways

When to use each model

- Concerned about storage space but not runtime and want best results?

When to use each model

- Concerned about storage space but not runtime and want best results?
  - ALBERT

When to use each model

- Concerned about storage space but not runtime and want best results?
  - ALBERT
- Want good old fashioned BERT?

When to use each model

- Concerned about storage space but not runtime and want best results?
  - ALBERT
- Want good old fashioned BERT?
  - RoBERTa

When to use each model

- Concerned about storage space but not runtime and want best results?
  - ALBERT
- Want good old fashioned BERT?
  - RoBERTa
- Concerned about runtime?

When to use each model

- Concerned about storage space but not runtime and want best results?
  - ALBERT
- Want good old fashioned BERT?
  - RoBERTa
- Concerned about runtime?
  - DistilBERT

GLUE Leaderboard at time of ALBERT

Models	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
BERT-large	86.6	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
XLNet-large	89.8	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa-large	90.2	94.7	**92.2**	86.6	96.4	**90.9**	68.0	92.4	-	-
ALBERT (1M)	90.4	95.2	92.0	88.1	96.8	90.2	68.7	92.7	-	-
ALBERT (1.5M)	**90.8**	**95.3**	**92.2**	**89.2**	**96.9**	**90.9**	**71.4**	**93.0**	-	-
Ensembles on test (from leaderboard as of Sept. 16, 2019)										
ALICE	88.2	95.7	**90.7**	83.5	95.2	92.6	**69.2**	91.1	80.8	87.0
MT-DNN	87.9	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2	98.6	90.3	86.3	96.8	93.0	67.8	91.6	90.4	88.4
RoBERTa	90.8	98.9	90.2	88.2	96.7	92.3	67.8	92.2	89.0	88.5
Adv-RoBERTa	91.1	98.8	90.3	88.7	96.8	93.1	68.0	92.4	89.0	88.8
ALBERT	**91.3**	**99.2**	90.5	**89.2**	**97.1**	**93.4**	69.1	**92.5**	**91.8**	**89.4**

RACE and SQuAD at time of ALBERT

Models	SQuAD1.1 dev	SQuAD2.0 dev	SQuAD2.0 test	RACE test (Middle/High)
Single model (from leaderboard as of Sept. 23, 2019)				
BERT-large	90.9/84.1	81.8/79.0	89.1/86.3	72.0 (76.6/70.1)
XLNet	94.5/89.0	88.8/86.1	89.1/86.3	81.8 (85.5/80.2)
RoBERTa	94.6/88.9	89.4/86.5	89.8/86.8	83.2 (86.5/81.3)
UPM	-	-	89.9/87.2	-
XLNet + SG-Net Verifier++	-	-	90.1/87.2	-
ALBERT (1M)	94.8/89.2	89.9/87.2	-	86.0 (88.2/85.1)
ALBERT (1.5M)	**94.8/89.3**	**90.2/87.4**	**90.9/88.1**	**86.5 (89.0/85.5)**
Ensembles (from leaderboard as of Sept. 23, 2019)				
BERT-large	92.2/86.2	-	-	-
XLNet + SG-Net Verifier	-	-	90.7/88.2	-
UPM	-	-	90.7/88.2	
XLNet + DAAF + Verifier	-	-	90.9/88.6	-
DCMN+	-	-	-	84.1 (88.5/82.3)
ALBERT	**95.5/90.1**	**91.4/88.9**	**92.2/89.7**	**89.4 (91.2/88.6)**