Richer Contexts

Maury Courtland (PhD; www.maury.science)

November 12, 2019

Talk Overview

Talk Overview

- 30k foot background

Talk Overview

- 30k foot background
- ELMo and deep representations

Talk Overview

- 30k foot background
- ELMo and deep representations
- BERT and pre-trained models

30k foot background: The benefits of pre-training and more information

30k foot background: The benefits of pre-training and more information

- Pre-training avoids repeated, costly spin-up time (and core-hours)

30k foot background: The benefits of pre-training and more information

- Pre-training avoids repeated, costly spin-up time (and core-hours)
  - Arguably what allowed CV to take off (AlexNet/ImageNet)

30k foot background: The benefits of pre-training and more information

- Pre-training avoids repeated, costly spin-up time (and core-hours)
  - Arguably what allowed CV to take off (AlexNet/ImageNet)
- Fixed models provide a common out-of-the-box starter model to extend (like a base-class module)

30k foot background: The benefits of pre-training and more information

- Pre-training avoids repeated, costly spin-up time (and core-hours)
  - Arguably what allowed CV to take off (AlexNet/ImageNet)
- Fixed models provide a common out-of-the-box starter model to extend (like a base-class module)
  - Helps researchers have a shared common ground of tools

30k foot background: The benefits of pre-training and more information

- Pre-training avoids repeated, costly spin-up time (and core-hours)
  - Arguably what allowed CV to take off (AlexNet/ImageNet)
- Fixed models provide a common out-of-the-box starter model to extend (like a base-class module)
  - Helps researchers have a shared common ground of tools
- What could be wrong with *more* information (**a little troll-y**)

30k foot background: The benefits of pre-training and more information

- Pre-training avoids repeated, costly spin-up time (and core-hours)
  - Arguably what allowed CV to take off (AlexNet/ImageNet)
- Fixed models provide a common out-of-the-box starter model to extend (like a base-class module)
  - Helps researchers have a shared common ground of tools
- What could be wrong with *more* information (**a little troll-y**)
  - More signal (as long as it's worth the extra learning time and complexity) can immensely boost performance

Peters et al. 2018: Deep contextualized word representations (aka ELMo)

Background and Approach

- Wants to capture both syntax and semantics as previously done

Background and Approach

- Wants to capture both syntax and semantics as previously done
- Wants to capture variance across contexs (i.e. polysemy) *not* previously done

Background and Approach

- Wants to capture both syntax and semantics as previously done
- Wants to capture variance across contexs (i.e. polysemy) *not* previously done
  - Tokens are assigned representations based on entire sentence

Background and Approach

- Wants to capture both syntax and semantics as previously done
- Wants to capture variance across contexs (i.e. polysemy) *not* previously done
  - Tokens are assigned representations based on entire sentence
  - Uses bidirectional LSTM with language modeling (LM) objective

Background and Approach

- Wants to capture both syntax and semantics as previously done
- Wants to capture variance across contexs (i.e. polysemy) *not* previously done
  - Tokens are assigned representations based on entire sentence
  - Uses bidirectional LSTM with language modeling (LM) objective
  - Thus, ELMo (***E***mbeddings from ***L***anguage ***Mo***dels)

Strengths

- ELMo representations are deep (a function of all biLM layers, not just top layer)

Strengths

- ELMo representations are deep (a function of all biLM layers, not just top layer)
- Allows distributed meaning encoding

Strengths

- ELMo representations are deep (a function of all biLM layers, not just top layer)
- Allows distributed meaning encoding
    - Higher states capture context-dependent meaning (good for WSD)

Strengths

- ELMo representations are deep (a function of all biLM layers, not just top layer)
- Allows distributed meaning encoding
  - Higher states capture context-dependent meaning (good for WSD)
  - Lower states capture aspects of syntax (good for POS tagging)

Strengths

- ELMo representations are deep (a function of all biLM layers, not just top layer)
- Allows distributed meaning encoding
  - Higher states capture context-dependent meaning (good for WSD)
  - Lower states capture aspects of syntax (good for POS tagging)
  - With access to all this information, downstream models can select the relevant dimensions of information for the task (i.e. they are transferable)

Method

- Start with forward and backward LMs

Method

- Start with forward and backward LMs
  - Forward model: $p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2, ..., t_{k-1})$

Method

- Start with forward and backward LMs
  - Forward model: $p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2, ..., t_{k-1})$
  - Backward model: $p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, t_{k+2}, ...t_N)$

Method

- Start with forward and backward LMs
  - Forward model: $p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2, ..., t_{k-1})$
  - Backward model: $p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, t_{k+2}, ...t_N)$
- Jointly maximize their log likelihoods:

Method

- Start with forward and backward LMs
  - Forward model: $p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2, ..., t_{k-1})$
  - Backward model: $p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, t_{k+2}, ...t_N)$
- Jointly maximize their log likelihoods:
  -

$$\sum_{k=1}^{N} \Big(\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \Big)$$

ELMo's Magic

- ...

ELMo's Magic

- ...
- Just combine all the representations computed for each token into a single vector!

each token t_k, a L-layer biLM computes a set of $2L + 1$ representations

$$\begin{aligned} R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\ &= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\}, \end{aligned}$$

where $\mathbf{h}_{k,0}^{LM}$ is the token layer and $\mathbf{h}_{k,j}^{LM} = [\overrightarrow{\mathbf{h}}_{k,j}^{LM}; \overleftarrow{\mathbf{h}}_{k,j}^{LM}]$, for each biLSTM layer.

ELMo's Magic

- ...
- Just combine all the representations computed for each token into a single vector!

each token t_k, a L-layer biLM computes a set of $2L + 1$ representations

$$
\begin{aligned}
R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\
&= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\},
\end{aligned}
$$

where $\mathbf{h}_{k,0}^{LM}$ is the token layer and $\mathbf{h}_{k,j}^{LM} = [\overrightarrow{\mathbf{h}}_{k,j}^{LM}; \overleftarrow{\mathbf{h}}_{k,j}^{LM}]$, for each biLSTM layer.

- Previous approaches for contextually aware embeddings use a forward LM and take the last layer of the hidden state to be the embedding

Choosing a Useful Representation

- Simplest case is to simply select the top layer (as in previous work)

Choosing a Useful Representation

- Simplest case is to simply select the top layer (as in previous work)
- ***Or***, for each task, adjust the weighting function of each layer:

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}.$$

Choosing a Useful Representation

- Simplest case is to simply select the top layer (as in previous work)
- ***Or***, for each task, adjust the weighting function of each layer:

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}.$$

  - where s^{task} are softmax-normalized weights and γ^{task} allows the downstream model to scale the ELMo vector.

Choosing a Useful Representation

- Simplest case is to simply select the top layer (as in previous work)
- ***Or***, for each task, adjust the weighting function of each layer:

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}.$$

  - where s^{task} are softmax-normalized weights and γ^{task} allows the downstream model to scale the ELMo vector.
  - sometimes it helps to apply layer normalization before weighting

Applying the Representations Downstream

- Use the pre-trained biLM and a specified architecture for target task

Applying the Representations Downstream

- Use the pre-trained biLM and a specified architecture for target task
  1) Run pre-trained (frozen) biLM and store representations for each word

Applying the Representations Downstream

- Use the pre-trained biLM and a specified architecture for target task
  1) Run pre-trained (frozen) biLM and store representations for each word
  2) Let target task model learn a linear combination of these representations

Applying the Representations Downstream

- Use the pre-trained biLM and a specified architecture for target task
  1) Run pre-trained (frozen) biLM and store representations for each word
  2) Let target task model learn a linear combination of these representations
- Can include (by concatenation) ELMo representations both in the input (concatenated with a context-independent embedding) and output (concatenated with the top hidden layer)

Applying the Representations Downstream

- Use the pre-trained biLM and a specified architecture for target task
  1) Run pre-trained (frozen) biLM and store representations for each word
  2) Let target task model learn a linear combination of these representations
- Can include (by concatenation) ELMo representations both in the input (concatenated with a context-independent embedding) and output (concatenated with the top hidden layer)
- Authors find it helpful to add “a moderate amount” of dropout

Applying the Representations Downstream

- Use the pre-trained biLM and a specified architecture for target task
  1) Run pre-trained (frozen) biLM and store representations for each word
  2) Let target task model learn a linear combination of these representations
- Can include (by concatenation) ELMo representations both in the input (concatenated with a context-independent embedding) and output (concatenated with the top hidden layer)
- Authors find it helpful to add "a moderate amount" of dropout
- Also some cases regularization helped (adding $\lambda||w||_2^2$ to the loss making it close to an average of all layers)

Results: SOTA!

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Table 1: Test set comparison of ELMo enhanced neural models with state-of-the-art single model baselines across six benchmark NLP tasks. The performance metric varies across tasks – accuracy for SNLI and SST-5; F_1 for SQuAD, SRL and NER; average F_1 for Coref. Due to the small test sizes for NER and SST-5, we report the mean and standard deviation across five runs with different random seeds. The "increase" column lists both the absolute and relative improvements over our baseline.

Figure 1: ELMo's Improvements Across the board

Analysis

- ELMo improves over just last layer and is better when linear weights are allowed to vary $\lambda = .001$ (better than $\lambda = 1$)

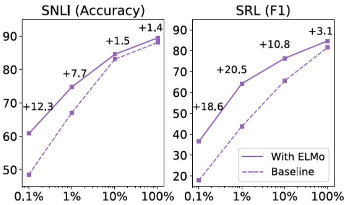

Figure 1: Comparison of baseline vs. ELMo performance for SNLI and SRL as the training set size is varied from 0.1% to 100%.

Figure 2: ELMo is very useful for small training corpora

Analysis

- ELMo improves over just last layer and is better when linear weights are allowed to vary $\lambda = .001$ (better than $\lambda = 1$)
- Isolation of layers as features for WSD and POS tagging tasks reveal different layers capture different information

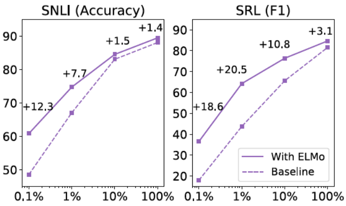

Figure 1: Comparison of baseline vs. ELMo performance for SNLI and SRL as the training set size is varied from 0.1% to 100%.

Figure 2: ELMo is very useful for small training corpora

Analysis

- ELMo improves over just last layer and is better when linear weights are allowed to vary $\lambda = .001$ (better than $\lambda = 1$)
- Isolation of layers as features for WSD and POS tagging tasks reveal different layers capture different information
- ELMo also converges (at higher performance) much quicker than previous approaches

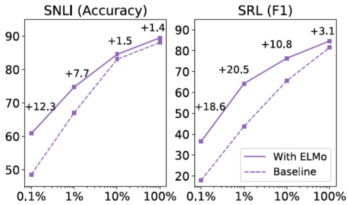

Figure 1: Comparison of baseline vs. ELMo performance for SNLI and SRL as the training set size is varied from 0.1% to 100%.

Figure 2: ELMo is very useful for small training corpora

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)

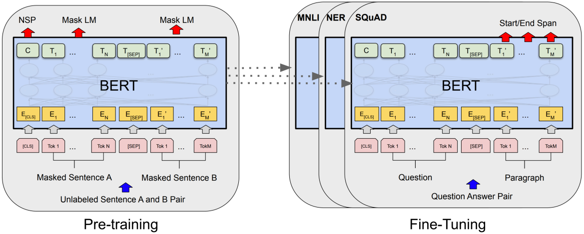

Figure 3: Applying pre-trained BERT to downstream tasks

Methods

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

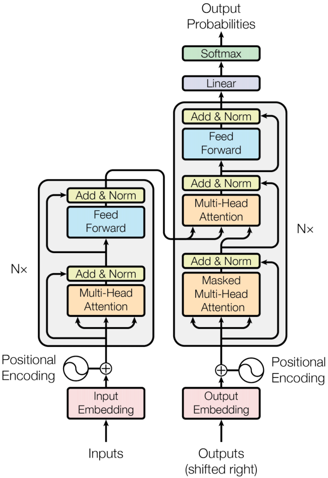

Figure 4: Transformer architecture

Inputs

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

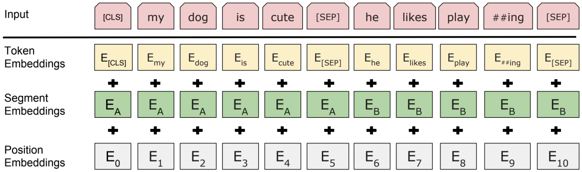

Figure 5: BERT input representation

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

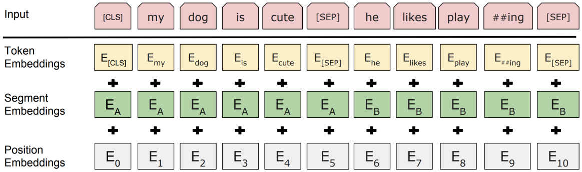

Figure 5: BERT input representation

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)

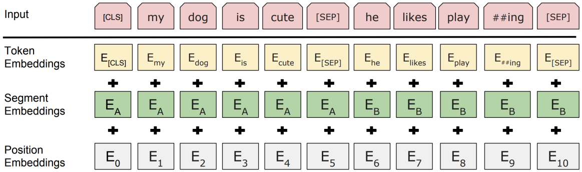

Figure 5: BERT input representation

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 $T_{[SEP]}$)

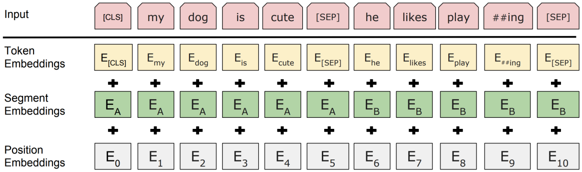

Figure 5: BERT input representation

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 6: 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)