Natural Language Processing with Deep Learning

Contextual Word Representations and Pretraining (ELMO, BERT, Transformer)
Representations for a word

• Up until now, we’ve basically said that we have one representation of words:
  • The word vectors that we learned about at the beginning
    • Word2vec, GloVe, fastText
Pre-trained word vectors: The early years
Collobert, Weston, et al. 2011 results

<table>
<thead>
<tr>
<th></th>
<th>POS WSJ (acc.)</th>
<th>NER CoNLL (F1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>State-of-the-art*</td>
<td>97.24</td>
<td>89.31</td>
</tr>
<tr>
<td>Supervised NN</td>
<td>96.37</td>
<td>81.47</td>
</tr>
<tr>
<td>Unsupervised pre-training</td>
<td>97.20</td>
<td>88.87</td>
</tr>
<tr>
<td>followed by supervised NN**</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ hand-crafted features***</td>
<td>97.29</td>
<td>89.59</td>
</tr>
</tbody>
</table>

* Representative systems: POS: (Toutanova et al. 2003), NER: (Ando & Zhang 2005)
** 130,000-word embedding trained on Wikipedia and Reuters with 11 word window, 100 unit hidden layer – for 7 weeks! – then supervised task training
*** Features are character suffixes for POS and a gazetteer for NER
Pre-trained word vectors: Current sense (2014–)

• We can just start with random word vectors and train them on our task of interest
• But in most cases, use of pre-trained word vectors helps, because we can train them for more words on much more data

• Chen and Manning (2014) Dependency parsing
• Random: uniform(-0.01, 0.01)
• Pre-trained:
  • PTB (C & W): +0.7%
  • CTB (word2vec): +1.7%

CoNLL Dependency, Stanford Dependency
Tips for unknown words with word vectors

• Simplest and common solution:
• Train time: Vocab is \{words occurring, say, ≥ 5 times\} ∪ \{<UNK>\}
• Map all rarer (< 5) words to <UNK>, train a word vector for it
• Runtime: use <UNK> when out-of-vocabulary (OOV) words occur

• Problems:
  • No way to distinguish different UNK words, either for identity or meaning

• Solutions:
  Hey, we just learned about char-level models to build vectors! Let’s do that!
Tips for unknown words with word vectors

- Especially in applications like question answering
  - Where it is important to match on word identity, even for words outside your word vector vocabulary

- Try these tips (from Dhingra, Liu, Salakhutdinov, Cohen 2017)
  a. If the <UNK> word at test time appears in your unsupervised word embeddings, use that vector as is at test time.
  b. Additionally, for other words, just assign them a random vector, adding them to your vocabulary

- a. definitely helps a lot; b. may help a little more

- Another thing you can try:
  - Collapsing things to word classes (like unknown number, capitalized thing, etc. and having an <UNK-class> for each
Representations for a word

- Up until now, we’ve basically had one representation of words:
  - The word vectors that we learned about at the beginning
    - Word2vec, GloVe, fastText

- These have two problems:
  - Always the same representation for a word type regardless of the context in which a word token occurs
    - We might want very fine-grained word sense disambiguation
  - We just have one representation for a word, but words have different aspects, including semantics, syntactic behavior, and register/connotations
Did we all along have a solution to this problem?

- In an NLM, we immediately stuck word vectors (perhaps only trained on the corpus) through LSTM layers.
- Those LSTM layers are trained to predict the next word.
- But those language models are producing context-specific word representations at each position!
Peters et al. (2017): TagLM – “Pre-ELMo”


• Idea: Want meaning of word in context, but standardly learn task RNN only on small task-labeled data (e.g., NER)

• Why don’t we do semi-supervised approach where we train NLM on large unlabeled corpus, rather than just word vectors?
Tag LM

**Step 1:** Pretrain word embeddings and language model.

**Step 2:** Prepare word embedding and LM embedding for each token in the input sequence.

**Step 3:** Use both word embeddings and LM embeddings in the sequence tagging model.

Two representations of the word “York”
Tag LM

\[ h_{k,1} = [h_{k,1}^\rightarrow; h_{k,1}^\leftarrow; h_{k,1}^{LM}] \]
A very important NLP sub-task: find and classify names in text, for example:

The decision by the independent MP Andrew Wilkie to withdraw his support for the minority Labor government sounded dramatic but it should not further threaten its stability. When, after the 2010 election, Wilkie, Rob Oakeshott, Tony Windsor and the Greens agreed to support Labor, they gave just two guarantees: confidence and supply.
Peters et al. (2017): TagLM – “Pre-ELMo”

Language model is trained on 800 million training words of “Billion word benchmark”

Language model observations
• An LM trained on supervised data does not help
• Having a bidirectional LM helps over only forward, by about 0.2
• Having a huge LM design (ppl 30) helps over a smaller model (ppl 48) by about 0.3

Task-specific BiLSTM observations
• Using just the LM embeddings to predict isn’t great: 88.17 F1
  • Well below just using an BiLSTM tagger on labeled data
Also in the air: McCann et al. 2017: CoVe


- Also has idea of using a trained sequence model to provide context to other NLP models
- Idea: Machine translation is meant to preserve meaning, so maybe that’s a good objective?
- Use a 2-layer bi-LSTM that is the encoder of seq2seq + attention NMT system as the context provider
- The resulting CoVe vectors do outperform GloVe vectors on various tasks
- But, the results aren’t as strong as the simpler NLM training described in the rest of these slides so seems abandoned
  - Maybe NMT is just harder than language modeling?
  - Maybe someday this idea will return?
Peters et al. (2018): ELMo: Embeddings from Language Models


- Breakout version of **word token vectors** or **contextual word vectors**
- Learn word token vectors using long contexts not context windows (here, whole sentence, could be longer)
- Learn a deep Bi-NLM and use all its layers in prediction
Peters et al. (2018): ELMo: Embeddings from Language Models

- Train a bidirectional LM
- Aim at performant but not overly large LM:
  - Use 2 biLSTM layers
  - Use character CNN to build initial word representation (only)
    - 2048 char n-gram filters and 2 highway layers, 512 dim projection
  - User 4096 dim hidden/cell LSTM states with 512 dim projections to next input
  - Use a residual connection
  - Tie parameters of token input and output (softmax) and tie these between forward and backward LMs
Peters et al. (2018): ELMo: Embeddings from Language Models

- ELMo learns task-specific combination of biLM representations
- This is an innovation that improves on just using top layer of LSTM stack

\[
R_k = \{ x^{LM}_k, \overrightarrow{h}^{LM}_{k,j}, \overleftarrow{h}^{LM}_{k,j} \mid j = 1, \ldots, L \}
\]

\[
= \{ h^{LM}_{k,j} \mid j = 0, \ldots, L \},
\]

\[
\text{ELMo}^{task}_k = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s^{task}_j h^{LM}_{k,j}
\]

- \( \gamma^{task} \) scales overall usefulness of ELMo to task;
- \( s^{task} \) are softmax-normalized mixture model weights
Peters et al. (2018): ELMo: Use with a task

- First run biLM to get representations for each word
- Then let (whatever) end-task model use them
  - Freeze weights of ELMo for purposes of supervised model
  - Concatenate ELMo weights into task-specific model
    - Details depend on task
      - Concatenating into intermediate layer as for TagLM is typical
      - Can provide ELMo representations again when producing outputs, as in a question answering system
ELMo used in a sequence tagger

\[ h_{k,1} = [\overrightarrow{h}_{k,1}, \overleftarrow{h}_{k,1}, h_{k,LM}] \]
ELMo results: Great for all tasks

<table>
<thead>
<tr>
<th>TASK</th>
<th>PREVIOUS SOTA</th>
<th>OUR BASELINE</th>
<th>ELMo + BASELINE</th>
<th>INCREASE (ABSOLUTE/RELATIVE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQuAD</td>
<td>Liu et al. (2017)</td>
<td>84.4</td>
<td>81.1</td>
<td>85.8</td>
</tr>
<tr>
<td>SNLI</td>
<td>Chen et al. (2017)</td>
<td>88.6</td>
<td>88.0</td>
<td>88.7 ± 0.17</td>
</tr>
<tr>
<td>SRL</td>
<td>He et al. (2017)</td>
<td>81.7</td>
<td>81.4</td>
<td>84.6</td>
</tr>
<tr>
<td>Coref</td>
<td>Lee et al. (2017)</td>
<td>67.2</td>
<td>67.2</td>
<td>70.4</td>
</tr>
<tr>
<td>NER</td>
<td>Peters et al. (2017)</td>
<td>91.93 ± 0.19</td>
<td>90.15</td>
<td>92.22 ± 0.10</td>
</tr>
<tr>
<td>SST-5</td>
<td>McCann et al. (2017)</td>
<td>53.7</td>
<td>51.4</td>
<td>54.7 ± 0.5</td>
</tr>
</tbody>
</table>
ELMo: Weighting of layers

• The two biLSTM NLM layers have differentiated uses/meanings
  • Lower layer is better for lower-level syntax, etc.
    • Part-of-speech tagging, syntactic dependencies, NER
  • Higher layer is better for higher-level semantics
    • Sentiment, Semantic role labeling, question answering, SNLI

• This seems interesting, but it’d seem more interesting to see how it pans out with more than two layers of network
Also around: ULMfit


- Same general idea of transferring NLM knowledge
- Here applied to text classification
ULMfit

Train LM on big general domain corpus (use biLM)
Tune LM on target task data
Fine-tune as classifier on target task
ULMfit emphases

Use reasonable-size “1 GPU” language model not really huge one  
A lot of care in LM fine-tuning  
Different per-layer learning rates  
Slanted triangular learning rate (STLR) schedule  
Gradual layer unfreezing and STLR when learning classifier  
Classify using concatenation $[h_T, \text{maxpool}(\mathbf{h}), \text{meanpool}(\mathbf{h})]$
# ULMfit performance

- Text classifier error rates

<table>
<thead>
<tr>
<th>IMDb</th>
<th>Model</th>
<th>Test</th>
<th>Model</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoVe (McCann et al., 2017)</td>
<td>8.2</td>
<td>CoVe (McCann et al., 2017)</td>
<td>4.2</td>
<td></td>
</tr>
<tr>
<td>oh-LSTM (Johnson and Zhang, 2016)</td>
<td>5.9</td>
<td>TBCNN (Mou et al., 2015)</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>Virtual (Miyato et al., 2016)</td>
<td>5.9</td>
<td>LSTM-CNN (Zhou et al., 2016)</td>
<td>3.9</td>
<td></td>
</tr>
<tr>
<td>ULMFiT (ours)</td>
<td><strong>4.6</strong></td>
<td>ULMFiT (ours)</td>
<td><strong>3.6</strong></td>
<td></td>
</tr>
</tbody>
</table>
Let’s scale it up!

ULMfit
Jan 2018
Training:
1 GPU day

GPT
June 2018
Training
240 GPU days

BERT
Oct 2018
Training
256 TPU days
~320–560 GPU days

GPT-2
Feb 2019
Training
~2048 TPU v3
days according to
a reddit thread
In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

The scientist named the population, after their distinctive horn, Ovid’s Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. ...
Elon Musk’s scientists have announced the creation of a terrifying artificial intelligence that’s so smart they refused to release it to the public.

OpenAI’s GPT-2 is designed to write just like a human and is an impressive leap forward capable of penning chillingly convincing text.

It was ‘trained’ by analysing eight million web pages and is capable of writing large tracts based upon a ‘prompt’ written by a real person.

But the machine mind will not be released in its fully-fledged form because of the risk of it being used for ‘malicious purposes’ such as generating fake news, impersonating people online, automating the production of spam or churning out ‘abusive or faked content to post on social media’.

OpenAI wrote: ‘Due to our concerns about malicious applications of the technology, we are not releasing the trained model.'
Transformer models

<table>
<thead>
<tr>
<th>Model</th>
<th>Year</th>
<th>Training Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>ULMfit</td>
<td>Jan 2018</td>
<td>Training: 1 GPU day</td>
</tr>
<tr>
<td>GPT</td>
<td>June 2018</td>
<td>240 GPU days</td>
</tr>
<tr>
<td>BERT</td>
<td>Oct 2018</td>
<td>256 TPU days; ~320–560 GPU days</td>
</tr>
<tr>
<td>GPT-2</td>
<td>Feb 2019</td>
<td>Training; ~2048 TPU v3 days according to a reddit thread</td>
</tr>
</tbody>
</table>

All of these models are Transformer architecture models ... so maybe we had better learn about Transformers?
The Motivation for Transformers

- We want **parallelization** but RNNs are inherently sequential

- Despite GRUs and LSTMs, RNNs still need an attention mechanism to deal with long range dependencies – **path length** between states grows with sequence otherwise

- But if **attention** gives us access to any state... maybe we can just use attention and don’t need the RNN? 😐
Transformer Overview

Attention is all you need. 2017. Aswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin
• Non-recurrent sequence-to-sequence encoder-decoder model
• Task: machine translation with parallel corpus
• Predict each translated word
• Final cost/error function is standard cross-entropy error on top of a softmax classifier

This and related figures from paper ↑
Transformer Basics

• Learning about transformers on your own?
  • Key recommended resource:
    • [http://nlp.seas.harvard.edu/2018/04/03/attention.html](http://nlp.seas.harvard.edu/2018/04/03/attention.html)
    • The Annotated Transformer by Sasha Rush
  • An Jupyter Notebook using PyTorch that explains everything!

• For now: Let’s define the basic building blocks of transformer networks: first, new attention layers!
Dot-Product Attention

- Inputs: a query $q$ and a set of key-value (k-v) pairs to an output
- Query, keys, values, and output are all vectors

- Output is weighted sum of values, where
- Weight of each value is computed by an inner product of query and corresponding key
- Queries and keys have same dimensionality $d_k$ value have $d_v$

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$
Dot-Product Attention – Matrix notation

• When we have multiple queries \( q \), we stack them in a matrix \( Q \):

\[
A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i
\]

• Becomes:

\[
A(Q, K, V) = \text{softmax}(QK^T)V
\]

\[
[|Q| \times d_k] \times [d_k \times |K|] \times [|K| \times d_v]
\]

softmax row-wise

\[
= [|Q| \times d_v]
\]
**Scaled Dot-Product Attention**

- **Problem:** As $d_k$ gets large, the variance of $q^T k$ increases $\rightarrow$ some values inside the softmax get large $\rightarrow$ the softmax gets very peaked $\rightarrow$ hence its gradient gets smaller.

- **Solution:** Scale by length of query/key vectors:

$$A(Q, K, V) = \text{softmax} \left( \frac{Q K^T}{\sqrt{d_k}} \right) V$$
Self-attention in the encoder

• The input word vectors are the queries, keys and values

• In other words: the word vectors themselves select each other

• Word vector stack = Q = K = V

• We’ll see in the decoder why we separate them in the definition
Multi-head attention

- Problem with simple self-attention:
- Only one way for words to interact with one-another
- Solution: Multi-head attention
- First map Q, K, V into h=8 many lower dimensional spaces via W matrices
- Then apply attention, then concatenate outputs and pipe through linear layer

\[
\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O
\]

where \(\text{head}_i = \text{Attention}(QW^Q_i, KW^K_i, VW^V_i)\)
Complete transformer block

- Each block has two “sublayers”
  1. Multihead attention
  2. 2-layer feed-forward NNet (with ReLU)

Each of these two steps also has:
Residual (short-circuit) connection and LayerNorm
LayerNorm(x + Sublayer(x))

LayerNorm changes input to have mean 0 and variance 1, per layer and per training point (and adds two more parameters)

\[
\mu^l = \frac{1}{H} \sum_{i=1}^{H} a_i^l
\]
\[
\sigma^l = \sqrt{\frac{1}{H} \sum_{i=1}^{H} (a_i^l - \mu^l)^2}
\]
\[
h_i = f\left(\frac{g_i}{\sigma_i} (a_i - \mu_i) + b_i\right)
\]

Encoder Input

- Actual word representations are byte-pair encodings

- Also added is a **positional encoding** so same words at different locations have different overall representations:

\[
PE_{(pos,2i)} = \sin \left( \frac{pos}{10000^{2i/d_{\text{model}}}} \right)
\]

\[
PE_{(pos,2i+1)} = \cos \left( \frac{pos}{10000^{2i/d_{\text{model}}}} \right)
\]
Complete Encoder

- For encoder, at each block, we use the same Q, K and V from the previous layer

- Blocks are repeated 6 times
  - (in vertical stack)
Attention visualization in layer 5

- Words start to pay attention to other words in sensible ways
In 5\textsuperscript{th} layer. Isolated attentions from just the word ‘its’ for attention heads 5 and 6. Note that the attentions are very sharp for this word.
Transformer Decoder

- 2 sublayer changes in decoder
- Masked decoder self-attention on previously generated outputs:
- Encoder-Decoder Attention, where queries come from previous decoder layer and keys and values come from output of encoder

Blocks repeated 6 times also
Tips and tricks of the Transformer

Details (in paper and/or later lectures):

• Byte-pair encodings
• Checkpoint averaging
• ADAM optimizer with learning rate changes
• Dropout during training at every layer just before adding residual
• Label smoothing
• Auto-regressive decoding with beam search and length penalties

• Use of transformers is spreading but they are hard to optimize and unlike LSTMs don’t usually just work out of the box and they don’t play well yet with other building blocks on tasks.
## Experimental Results for MT

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU</th>
<th>Training Cost (FLOPs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EN-DE</td>
<td>EN-FR</td>
</tr>
<tr>
<td>ByteNet [18]</td>
<td>23.75</td>
<td></td>
</tr>
<tr>
<td>Deep-Att + PosUnk [39]</td>
<td></td>
<td>39.2</td>
</tr>
<tr>
<td>GNMT + RL [38]</td>
<td>24.6</td>
<td>39.92</td>
</tr>
<tr>
<td>ConvS2S [9]</td>
<td>25.16</td>
<td>40.46</td>
</tr>
<tr>
<td>MoE [32]</td>
<td>26.03</td>
<td>40.56</td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [39]</td>
<td></td>
<td>40.4</td>
</tr>
<tr>
<td>GNMT + RL Ensemble [38]</td>
<td>26.30</td>
<td>41.16</td>
</tr>
<tr>
<td>ConvS2S Ensemble [9]</td>
<td>26.36</td>
<td>41.29</td>
</tr>
<tr>
<td>Transformer (base model)</td>
<td>27.3</td>
<td>38.1</td>
</tr>
<tr>
<td>Transformer (big)</td>
<td>28.4</td>
<td>41.8</td>
</tr>
</tbody>
</table>

BERT (Bidirectional Encoder Representations from Transformers): Pre-training of Deep Bidirectional Transformers for Language Understanding

Based on slides from Jacob Devlin
• **Problem**: Language models only use left context or right context, but language understanding is bidirectional.
• Why are LMs unidirectional?
• Reason 1: Directionality is needed to generate a well-formed probability distribution.
  • We don’t care about this.
• Reason 2: Words can “see themselves” in a bidirectional encoder.

Unidirectional context
Build representation incrementally

Bidirectional context
Words can “see themselves”
**Solution**: Mask out \( k\% \) of the input words, and then predict the masked words

- They always use \( k = 15\% \)

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

- Too little masking: Too expensive to train
- Too much masking: Not enough context
BERT complication: Next sentence prediction

- To learn *relationships* between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence.
## BERT sentence pair encoding

<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>
<tbody>
<tr>
<td><strong>Token Embeddings</strong></td>
<td>E_{CLS}</td>
<td>E_{my}</td>
<td>E_{dog}</td>
<td>E_{is}</td>
<td>E_{cute}</td>
<td>E_{[SEP]}</td>
<td>E_{he}</td>
<td>E_{likes}</td>
<td>E_{play}</td>
<td>E_{#ing}</td>
<td>E_{[SEP]}</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>Segment Embeddings</strong></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>
<td>E_{B}</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>Position Embeddings</strong></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>

Token embeddings are word pieces
Learned segmented embedding represents each sentence
Positional embedding is as for other Transformer architectures
BERT model architecture and training

- Transformer encoder (as before)
- Self-attention ⇒ no locality bias
  - Long-distance context has “equal opportunity”
- Single multiplication per layer ⇒ efficiency on GPU/TPU

- Train on Wikipedia + BookCorpus
- Train 2 model sizes:
  - BERT-Base: 12-layer, 768-hidden, 12-head
  - BERT-Large: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days
BERT model fine tuning

- Simply learn a classifier built on the top layer for each task that you fine tune for
BERT results on GLUE tasks

- GLUE benchmark is dominated by natural language inference tasks, but also has sentence similarity and sentiment

- **MultiNLI**
  - Premise: Hills and mountains are especially sanctified in Jainism.
  - Hypothesis: Jainism hates nature.
  - Label: Contradiction

- **CoLa**
  - Sentence: The wagon rumbled down the road. Label: Acceptable
  - Sentence: The car honked down the road. Label: Unacceptable
BERT results on GLUE tasks

<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>108k</td>
<td>67k</td>
<td>8.5k</td>
<td>5.7k</td>
<td>3.5k</td>
<td>2.5k</td>
<td></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\text{BASE}</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\text{LARGE}</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>
<tr>
<td>Name</td>
<td>Description</td>
<td>Year</td>
<td>F1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
<td>-----</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flair (Zalando)</td>
<td>Character-level language model</td>
<td>2018</td>
<td>93.09</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT Large</td>
<td>Transformer bidi LM + fine tune</td>
<td>2018</td>
<td>92.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVT Clark</td>
<td>Cross-view training + multitask learn</td>
<td>2018</td>
<td>92.61</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT Base</td>
<td>Transformer bidi LM + fine tune</td>
<td>2018</td>
<td>92.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ELMo</td>
<td>ELMo in BiLSTM</td>
<td>2018</td>
<td>92.22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TagLM Peters</td>
<td>LSTM BiLM in BiLSTM tagger</td>
<td>2017</td>
<td>91.93</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ma + Hovy</td>
<td>BiLSTM + char CNN + CRF layer</td>
<td>2016</td>
<td>91.21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tagger Peters</td>
<td>BiLSTM + char CNN + CRF layer</td>
<td>2017</td>
<td>90.87</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ratinov + Roth</td>
<td>Categorical CRF+Wikipedia+word cls</td>
<td>2009</td>
<td>90.80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finkel et al.</td>
<td>Categorical feature CRF</td>
<td>2005</td>
<td>86.86</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM Florian</td>
<td>Linear/softmax/TBL/HMM ensemble, gazettes++</td>
<td>2003</td>
<td>88.76</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stanford</td>
<td>MEMM softmax markov model</td>
<td>2003</td>
<td>86.07</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### BERT results on SQuAD 1.1

<table>
<thead>
<tr>
<th>Rank</th>
<th>Model</th>
<th>EM</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</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></td>
<td><a href="https://arxiv.org/abs/1810.04805">Link</a></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></td>
<td><a href="https://arxiv.org/abs/1810.04805">Link</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>nlnet (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>5</td>
<td>nlnet (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>

*Stanford University*

(Rajpurkar et al. '16)
### SQuAD 2.0 leaderboard, 2019-02-07

<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>86.831</td>
<td>89.452</td>
</tr>
<tr>
<td></td>
<td><em>Stanford University</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>(Rajpurkar &amp; Jia et al. '18)</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>BERT + MMFT + ADA (ensemble)</td>
<td>85.082</td>
<td>87.615</td>
</tr>
<tr>
<td></td>
<td><em>Microsoft Research Asia</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>BERT finetune baseline (ensemble)</td>
<td>83.536</td>
<td>86.096</td>
</tr>
<tr>
<td></td>
<td><em>Anonymous</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Lunet + Verifier + BERT (ensemble)</td>
<td>83.469</td>
<td>86.043</td>
</tr>
<tr>
<td></td>
<td><em>Layer 6 AI NLP Team</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>PAML+BERT (ensemble model)</td>
<td>83.457</td>
<td>86.122</td>
</tr>
<tr>
<td></td>
<td><em>PINGAN GammaLab</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Lunet + Verifier + BERT (single model)</td>
<td>82.995</td>
<td>86.035</td>
</tr>
<tr>
<td></td>
<td><em>Layer 6 AI NLP Team</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Effect of pre-training task
Size matters

- Going from 110M to 340M parameters helps a lot
- Improvements have not yet asymptoted