Backpropagation, Self-Attention, Text Representations Through Language Modeling

Karl Stratos

Rutgers University
Dropout

- Form of regularization for RNNs (and any NN in general)
- **Idea:** “Handicap” NN by removing hidden units **stochastically**
  - set each hidden unit in a layer to 0 with probability $p$ during training ($p = 0.5$ usually works well)
  - scale outputs by $1/(1 - p)$
  - hidden units forced to learn more general patterns
- **Test time:** Simply compute identity

(Slide credit Danqi Chen & Karthik Narasimhan)
Unidirectional vs Bidirectional RNN
Agenda

1. Backpropagation

2. Self-attention in NLP

3. Representation learning through language modeling
Backpropagation: Input and Output

- A technique to automatically calculate $\nabla J(\theta)$ for any definition of scalar-valued loss function $J(\theta) \in \mathbb{R}$.

  **Input**: loss function $J(\theta) \in \mathbb{R}$, parameter value $\hat{\theta}$

  **Output**: $\nabla J(\hat{\theta})$, the gradient of $J(\theta)$ at $\theta = \hat{\theta}$

- Calculates the gradient of an arbitrary differentiable function of parameter $\theta$

  Including neural networks
For the most part, we will consider (differentiable) function $f : \mathbb{R} \to \mathbb{R}$ with a single 1-dimensional parameter $x \in \mathbb{R}$.

The gradient of $f$ with respect to $x$ is a function of $x$

$$\frac{\partial f(x)}{\partial x} : \mathbb{R} \to \mathbb{R}$$

The gradient of $f$ with respect to $x$ evaluated at $x = a$ is written as

$$\left. \frac{\partial f(x)}{\partial x} \right|_{x=a} \in \mathbb{R}$$
Given any differentiable functions $f, g$ from $\mathbb{R}$ to $\mathbb{R}$,

$$\frac{\partial g(f(x))}{\partial x} = \frac{\partial g(f(x))}{\partial f(x)} \times \frac{\partial f(x)}{\partial x}$$

easy to calculate

“Proof”: Linearization of linearization of $g(z)$ around $f(x)$ around $a$

$$g(f(x)) \approx g(f(a)) + g'(f(a))f'(a)(x - a)$$

$$\left.\frac{\partial g(f(x))}{\partial x}\right|_{x=a}$$
Exercises

At $x = 42$,

- What is the value of the gradient of $f(x) := 7$?
- What is the value of the gradient of $f(x) := 2x$?
- What is the value of the gradient of $f(x) := 2x + 99999$?
- What is the value of the gradient of $f(x) := x^3$?
- What is the value of the gradient of $f(x) := \exp(x)$?
- What is the value of the gradient of $f(x) := \exp(2x^3 + 10)$?
- What is the value of the gradient of $f(x) := \log(\exp(2x^3 + 10))$?
Let $f_1 \ldots f_m$ denote any differentiable functions from $\mathbb{R}$ to $\mathbb{R}$.

If $g : \mathbb{R}^m \rightarrow \mathbb{R}$ is a differentiable function from $\mathbb{R}^m$ to $\mathbb{R}$,

$$
\frac{\partial g(f_1(x), \ldots, f_m(x))}{\partial x} = \sum_{i=1}^{m} \frac{\partial g(f_1(x), \ldots, f_m(x))}{\partial f_i(x)} \times \frac{\partial f_i(x)}{\partial x}
$$

easy to calculate

Calculate the gradient of $x + x^2 + yx$ with respect to $x$ using the chain rule.
A directed acyclic graph (DAG) is a directed graph $G = (V, A)$ with a topological ordering: a sequence $\pi$ of $V$ such that for every arc $(i, j) \in A$, $i$ comes before $j$ in $\pi$.

For backpropagation: usually assume have many roots and 1 leaf.
Notation

\[ V = \{1, 2, 3, 4, 5, 6\} \]
\[ V_I = \{1, 2\} \]
\[ V_N = \{3, 4, 5, 6\} \]
\[ A = \{(1, 3), (1, 5), (2, 4), (3, 4), (4, 6), (5, 6)\} \]
\[ \text{pa}(4) = \{2, 3\} \]
\[ \text{ch}(1) = \{3, 5\} \]
\[ \Pi_G = \{(1, 2, 3, 4, 5, 6), (2, 1, 3, 4, 5, 6)\} \]
Computation Graph

- DAG $G = (V, E)$ with a single output node $\omega \in V$.

- Every node $i \in V$ is equipped with a value $x^i \in \mathbb{R}$:
  1. For input node $i \in V_I$, we assume $x^i = a^i$ is given.
  2. For non-input node $i \in V_N$, we assume a differentiable function $f^i : \mathbb{R}^{|pa(i)|} \rightarrow \mathbb{R}$ and compute

$$x^i = f^i((x^j)_{j \in pa(i)})$$

- Thus $G$ represents a function: it receives multiple values $x^i = a^i$ for $i \in V_I$ and calculates a scalar $x^\omega \in \mathbb{R}$.
  - We can calculate $x^\omega$ by a forward pass.
Forward Pass

**Input:** computation graph $G = (V, A)$ with output node $\omega \in V$

**Result:** populates $x^i = a^i$ for every $i \in V$

1. Pick some topological ordering $\pi$ of $V$.
2. For $i$ in order of $\pi$, if $i \in V_N$ is a non-input node, set

$$x^i \leftarrow a^i := f^i((a^j)_{j \in \text{pa}(i)})$$

Why do we need a topological ordering?
Exercise

Construct the computation graph associated with the function

\[ f(x, y) := (x + y)xy^2 \]

Compute its output value at \( x = 1 \) and \( y = 2 \) by performing a forward pass.
For Notational Convenience... 

- Collectively refer to **all input slots** by \( x_I = (x^i)_{i \in V_I} \).
- Collectively refer to **all input values** by \( a_I = (a^i)_{i \in V_I} \).

- At \( i \in V \):
  - Refer to its **parental slots** by \( x^i_I = (x^j)_{j \in \text{pa}(i)} \).
  - Refer to its **parental values** by \( a^i_I = (a^j)_{j \in \text{pa}(i)} \).

Two equally valid ways of viewing any \( a^i \in \mathbb{R} \) as a function:

- A “global” function of \( x_I \) evaluated at \( a_I \).
- A “local” function of \( x^i_I \) evaluated at \( a^i_I \).
Computation Graph: Gradients

- Now for every node \( i \in V \), we introduce an additional slot \( z^i \in \mathbb{R} \) defined as

\[
z^i := \left. \frac{\partial x^\omega}{\partial x^i} \right|_{x_I = a_I}
\]

- **The goal of backpropagation** is to calculate \( z^i \) for every \( i \in V \).
  - Why are we done if we achieve this goal?
Key Ideas of Backpropagation

- Chain rule on the DAG structure

\[ z^i := \left. \frac{\partial x^\omega}{\partial x^i} \right|_{x_I=a_I} \]
Key Ideas of Backpropagation

- Chain rule on the DAG structure

\[ z^i := \left. \frac{\partial x^\omega}{\partial x^i} \right|_{x_I = a_I} = \sum_{j \in \text{ch}(i)} \left. \frac{\partial x^\omega}{\partial x^j} \right|_{x_I = a_I} \times \left. \frac{\partial x^j}{\partial x^i} \right|_{x_I = a_I} \]

- Easy to calculate
  - If we compute \( z^i \) in a reverse topological ordering, then we will have already computed \( z^j \) for all \( j \in \text{ch}(i) \).

- What's the base case \( z^\omega \)?
Key Ideas of Backpropagation

- Chain rule on the DAG structure

\[ z^i := \left. \frac{\partial x^\omega}{\partial x^i} \right|_{x^I = a_I} = \sum_{j \in \text{ch}(i)} \left. \frac{\partial x^\omega}{\partial x^j} \right|_{x^I = a_I} \times \left. \frac{\partial x^j}{\partial x^i} \right|_{x^j = a_I} \]

\[ = \sum_{j \in \text{ch}(i)} z^j \times \left. \frac{\partial f^j(x^I_j)}{\partial x^i} \right|_{x^I_j = a_I} \]

\(z^i\) is easy to calculate
Key Ideas of Backpropagation

- Chain rule on the DAG structure

\[ z^i := \left. \frac{\partial x^\omega}{\partial x^i} \right| \quad x_I = a_I = \sum_{j \in \text{ch}(i)} \left. \frac{\partial x^j}{\partial x^i} \right| x_I = a_I \times \left. \frac{\partial x^j}{\partial x^i} \right| x_I = a_I \]

\[ = \sum_{j \in \text{ch}(i)} z^j \times \left. \frac{\partial f^j(x_I^j)}{\partial x^i} \right|_{x_I^j = a_I} \]

- If we compute \( z^i \) in a reverse topological ordering, then we will have already computed \( z^j \) for all \( j \in \text{ch}(i) \).
Key Ideas of Backpropagation

- **Chain rule on the DAG structure**

$$ z^i := \left. \frac{\partial x^\omega}{\partial x^i} \right|_{x_I = a_I} = \sum_{j \in \text{ch}(i)} \left. \frac{\partial x^\omega}{\partial x^j} \right|_{x_I = a_I} \times \left. \frac{\partial x^j}{\partial x^i} \right|_{x_I = a_I} \times \left. \frac{\partial f^j(x_I^j)}{\partial x^i} \right|_{x_I^j = a_I} \quad \text{easy to calculate} $$

- If we compute $z^i$ in a **reverse topological ordering**, then we will have already computed $z^j$ for all $j \in \text{ch}(i)$.

- What’s the base case $z^\omega$?
Backpropagation

**Input:** computation graph $G = (V, A)$ with output node $\omega \in V$ whose value slots $x^i = a^i$ are already populated for every $i \in V$

**Result:** populates $z^i$ for every $i \in V$

1. Set $z^\omega \leftarrow 1$.
2. Pick some topological ordering $\pi$ of $V$.
3. For $i$ in reverse order of $\pi$, set

$$z^i \leftarrow \sum_{j \in \text{ch}(i)} z^j \times \frac{\partial f^j(x^j_I)}{\partial x^i} \Bigg|_{x^j_I = a^j_I}$$
Exercise

Calculate the gradient of

\[ f(x, y) := (x + y)xy^2 \]

with respect to \( x \) at \( x = 1 \) and \( y = 2 \) by performing backpropagation. That is, calculate the scalar

\[ \frac{\partial f(x, y)}{\partial x} \bigg|_{(x, y) = (1, 2)} \]
Answer
Implementation

- Each type of function \( f \) creates a child node from parent nodes and initializes its gradient to zero.
  - “Add” function creates a child node \( c \) with two parents \((a, b)\) and sets \( c.z \leftarrow 0 \).

- Each node has an associated **forward** function.
  - Calling forward at \( c \) populates \( c.x = a.x + b.x \) (assumes parents have their values).

- Each node also has an associated **backward** function.
  - Calling backward at \( c \) “broadcasts” its gradient \( c.z \) (assumes it’s already calculated) to its parents

\[
\begin{align*}
  a.z &\leftarrow a.z + c.z \\
  b.z &\leftarrow b.z + c.z
\end{align*}
\]
Implementation (Cont.)

- Express your loss $J_B(\theta)$ on minibatch $B$ at $\theta = \hat{\theta}$ as a computation graph.

- **Forward pass.** For each node $a$ in a topological ordering,

  $a$.forward()

- **Backward pass.** For each node $a$ in a reverse topological ordering,

  $a$.backward()

- The gradient of $J_B(\theta)$ at $\theta = \hat{\theta}$ is stored in the input nodes of the computation graph.
General Backpropagation

- Computation graph in which input values that are vectors
  
  \[ x^i \in \mathbb{R}^{d^i} \quad \forall i \in V \]

  But the output value \( x^\omega \in \mathbb{R} \) is always a scalar!

- The corresponding gradients are also vectors of the same size
  
  \[ z^i \in \mathbb{R}^{d^i} \quad \forall i \in V \]

- Backpropagation has exactly the same structure using the generalized chain rule

  \[
  z^i = \sum_{j \in \text{ch}(i)} \frac{\partial x^\omega}{\partial x^j} \bigg|_{x^I = a^I} \times \frac{\partial x^j}{\partial x^i} \bigg|_{x^j = a^j_I} \\
  1 \times d^j \\
  d^j \times d^i
  \]

  where second term is \textbf{Jacobian} of \( f^j \) wrt \( x^i \) evaluated at \( a^I \)
Agenda

1. Backpropagation

2. Self-attention in NLP

3. Representation learning through language modeling
Recurrent vs Self-Attention

\[ \frac{\partial h}{\partial x} \]
Attention: General Form

Input
- $Q \in \mathbb{R}^{d \times T}$: $T$ query vectors of the “asker”
- $K \in \mathbb{R}^{d \times T'}$: $T'$ key vectors of the “answerer”
- $V \in \mathbb{R}^{d \times T'}$: $T'$ value vectors of the “answerer”

Output
- $A \in \mathbb{R}^{d \times T}$: $T$ answer vectors of the “asker” after asking

$$A = V \text{softmax} (K^\top Q)$$
Example: Attention-Based Seq2Seq

**Input**
- \( Q = Y \): target LSTM encodings
- \( K = X \): source LSTM encodings
- \( V = X \): source LSTM encodings

**Output**
- \( A = \text{Attention}(Y, X, X) \): new target encodings

\[
A = X_{\text{softmax}} \left( X^\top Y \right)
\]
Scaled Attention

Useful when $d$ is large

$$A = V \text{softmax} \left( \frac{K^\top Q}{\sqrt{d}} \right)$$

Exercise: $k, q \in \mathbb{R}^d$ elementwise independent, mean 0, variance 1

- $\text{var} \left( k^\top q \right)$?
- $\text{var} \left( \frac{k^\top q}{\sqrt{d}} \right)$?
Multi-Head Attention

Same input

Parameters

- $W^Q_i \in \mathbb{R}^{(d/H) \times d}$ for $i = 1 \ldots H$: query projectors
- $W^K_i \in \mathbb{R}^{(d/H) \times d}$ for $i = 1 \ldots H$: key projectors
- $W^V_i \in \mathbb{R}^{(d/H) \times d}$ for $i = 1 \ldots H$: value projectors
- $W \in \mathbb{R}^{d \times d}$

$$A = W \begin{bmatrix}
\text{Attention} \left( W^Q_1 Q, W^K_1 K, W^V_1 V \right) \\
\vdots \\
\text{Attention} \left( W^Q_H Q, W^K_H K, W^V_H V \right)
\end{bmatrix}$$
Multi-Head Attention with Residual (or Skip) Connection

Plus regularization: dropout, layer normalization \((\text{Ba et al., 2016})\)

\[
A = \text{MultiHeadAttention}(Q, K, V)
\]

\[
A' = \text{LayerNorm} \left( \text{Drop} \left( A \right) + Q \right)
\]

Henceforth \(\text{ResMHA}(Q, K, V)\)
Transformer Encoder (Vaswani et al., 2017)

Using $H = 8$ heads, $l = 0 \ldots 5$

\[
\tilde{X}^{(l)} = \text{ResMHA} \left( X^{(l)}, X^{(l)}, X^{(l)} \right)
\]

\[
X^{(l+1)} = \text{ResFF} \left( \tilde{X}^{(l)} \right)
\]

\[
X^{(0)} = \text{Drop}_{0.1}(E + \Pi)
\]
Transformer Decoder (Vaswani et al., 2017)

Using $H = 8$ heads, $l = 0 \ldots 5$

$$\tilde{Y}^{(l)} = \text{ResMHA} \left( Y^{(l)}, Y^{(l)}, Y^{(l)} \right)$$

$$\overline{Y}^{(l)} = \text{ResMHA} \left( \tilde{Y}^{(l)}, X^{(6)}, X^{(6)} \right)$$

$$Y^{(l+1)} = \text{ResFF} \left( \overline{Y}^{(l)} \right)$$

Prediction: $\text{softmax} \left( EY^{(6)} \right)$
<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>39.2</td>
<td>1.0 \cdot 10^{20}</td>
</tr>
<tr>
<td>GNMT + RL [38]</td>
<td>24.6</td>
<td>2.3 \cdot 10^{19}</td>
</tr>
<tr>
<td>ConvS2S [9]</td>
<td>40.46</td>
<td>1.4 \cdot 10^{20}</td>
</tr>
<tr>
<td>MoE [32]</td>
<td>26.03</td>
<td>9.6 \cdot 10^{18}</td>
</tr>
<tr>
<td></td>
<td>40.56</td>
<td>1.5 \cdot 10^{20}</td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [39]</td>
<td>40.4</td>
<td>2.0 \cdot 10^{19}</td>
</tr>
<tr>
<td>GNMT + RL Ensemble [38]</td>
<td>26.30</td>
<td>1.2 \cdot 10^{20}</td>
</tr>
<tr>
<td>ConvS2S Ensemble [9]</td>
<td>41.16</td>
<td>8.0 \cdot 10^{20}</td>
</tr>
<tr>
<td></td>
<td>41.29</td>
<td></td>
</tr>
<tr>
<td>Transformer (base model)</td>
<td>27.3</td>
<td>1.8 \cdot 10^{20}</td>
</tr>
<tr>
<td>Transformer (big)</td>
<td>38.1</td>
<td>1.1 \cdot 10^{21}</td>
</tr>
<tr>
<td></td>
<td>28.4</td>
<td>3.3 \cdot 10^{18}</td>
</tr>
<tr>
<td></td>
<td>41.8</td>
<td>2.3 \cdot 10^{19}</td>
</tr>
</tbody>
</table>
Self-Attention Visualization (Vaswani et al., 2017)

Layer 5 and 6, one of the “heads”

Different heads learn different weights
Agenda

1. Backpropagation

2. Self-attention in NLP

3. Representation learning through language modeling
1. Language models can be trained on a lot of text (e.g., the web)

2. They yield text representations generally useful for downstream tasks
Example Downstream Tasks

Sentence classification

- Binary sentiment classification

This film doesn't care about intelligent humor $\rightarrow \begin{bmatrix} 0.05 \\ 0.95 \end{bmatrix}$

or multi-class (e.g., 5 stars)

- Example datasets: SST-2, IMBb, Yelp Review, SemEval, CoLA

- Sentiment analysis results: http://nlpprogress.com/english/sentiment_analysis.html

- Other types of classification: grammatical vs ungrammatical (CoLA)
Example Downstream Tasks

**Sentence pair classification**, or natural language inference (NLI)

\[
\begin{pmatrix}
0.05 \\
0.03 \\
0.92
\end{pmatrix}
\]

Example dataset: MNLI *(Williams et al., 2018)*

<table>
<thead>
<tr>
<th>Met my first girlfriend that way.</th>
<th>FACE-TO-FACE contradiction C C N C</th>
<th>I didn’t meet my first girlfriend until later.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 million in relief in the form of emergency housing.</td>
<td>GOVERNMENT neutral N N N N</td>
<td>The 8 million dollars for emergency housing was still not enough to solve the problem.</td>
</tr>
<tr>
<td>Now, as children tend their gardens, they have a new appreciation of their relationship to the land, their cultural heritage, and their community.</td>
<td>LETTERS neutral N N N N</td>
<td>All of the children love working in their gardens.</td>
</tr>
<tr>
<td>At 8:34, the Boston Center controller received a third transmission from American 11</td>
<td>9/11 entailment E E E</td>
<td>The Boston Center controller got a third transmission from American 11.</td>
</tr>
<tr>
<td>I am a lacto-vegetarian.</td>
<td>SLATE neutral N N E N</td>
<td>I enjoy eating cheese too much to abstain from dairy.</td>
</tr>
<tr>
<td>someone else noticed it and i said well i guess that’s true and it was somewhat melodious in other words it wasn’t just you know it was really funny</td>
<td>TELEPHONE contradiction C C C C</td>
<td>No one noticed and it wasn’t funny at all.</td>
</tr>
</tbody>
</table>
Example Downstream Tasks

**SQuAD-style question answering** *(Rajpurkar et al., 2016)*

Example dataset: SQuAD *(Rajpurkar et al., 2016)*

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, graupel and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called “showers”.

What causes precipitation to fall?
**gravity**

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
**graupel**

Where do water droplets collide with ice crystals to form precipitation?
**within a cloud**

Can be framed as predicting start/end index of the passage
Setting

- Each such downstream task provides only a limited amount of labeled data.

- Can we transfer a large-scale pretrained language model to improve performance in all these tasks simultaneously?

- Popular benchmarks (Wang et al, 2018):
  - GLUE: https://gluebenchmark.com/leaderboard
  - SuperGLUE: https://super.gluebenchmark.com/leaderboard
ELMo (Peters et al., 2018)

Trained 10 epochs on 1B Word Benchmark

Forward Language Model

Backward Language Model

# words in the sentence

$$\sum_{k=1}^{N} \left( \log p(t_k | t_1, \ldots, t_{k-1}; \Theta_x, \Theta_{LSTM}, \Theta_s) + \log p(t_k | t_{k+1}, \ldots, t_N; \Theta_x, \Theta_{LSTM}, \Theta_s) \right)$$

input

softmax
ELMo (Peters et al., 2018)

*Kitaev and Klein, ACL 2018  (see also Joshi et al., ACL 2018)
ELMo in Practice

1. ELMo layer: new representation of $i$-th token in a sequence

$$\text{ELMo}_i(\gamma, s_1 \ldots s_L) = \gamma \sum_{l=0}^{L} s_l \begin{cases} e^\text{ELMo}_i & \text{if } l = 0 \\
\begin{bmatrix} \overrightarrow{h}_{i,\text{ELMo},l} \\
\overleftarrow{h}_{i,\text{ELMo},l} \end{bmatrix} & \text{otherwise} \end{cases}$$

2. In your downstream task, concatenate $\text{ELMo}_i(\gamma, s_1 \ldots s_L)$ to your $i$-th input embedding.

3. Train your original model AND $\gamma, s_1 \ldots s_L$ while keeping ELMo parameters fixed
Using ELMo

https://allennlp.org/elmo

Pre-trained ELMo Models

<table>
<thead>
<tr>
<th>Model</th>
<th>Link(Weights/Options File)</th>
<th># Parameters (Millions)</th>
<th>LSTM Hidden Size/Output size</th>
<th># Highway Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>weights options</td>
<td>13.6</td>
<td>1024/128</td>
<td>1</td>
</tr>
<tr>
<td>Medium</td>
<td>weights options</td>
<td>28.0</td>
<td>2048/256</td>
<td>1</td>
</tr>
<tr>
<td>Original</td>
<td>weights options</td>
<td>93.6</td>
<td>4096/512</td>
<td>2</td>
</tr>
<tr>
<td>Original (5.5B)</td>
<td>weights options</td>
<td>93.6</td>
<td>4096/512</td>
<td>2</td>
</tr>
</tbody>
</table>

```python
from allennlp.modules.elmo import Elmo, batch_to_ids


# Compute two different representation for each token.
# Each representation is a linear weighted combination for the
# 3 layers in ELMo (i.e., charcnn, the outputs of the two BiLSTM))

elmo = Elmo(options_file, weight_file, 2, dropout=0)

# use batch_to_ids to convert sentences to character ids

sentences = [['First', 'sentence', '.'], ['Another', '.']]

character_ids = batch_to_ids(sentences)

embeddings = elmo(character_ids)
```
Recurrent vs Self-Attention Encoding

not bidirectional until later

deploy bidirectional
For the purposes of representation learning, we don’t care about defining a proper language model which only conditions on previous history.

We want a prediction problem which conditions on entire context all the time, so that we can use deeply bidirectional encoders.

Solution: mask out words at random

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

Need to be careful

- Too little masking: too expensive to train
- Too much masking: not enough context
- Test time: no [MASK] input, so training should also handle no [MASK] input sometimes

BERT (Devlin et al., 2019)

Transformer (Vaswani et al., 2017)
BERT (Devlin et al., 2019)

<table>
<thead>
<tr>
<th>System</th>
<th>MNLI-(m/mm)</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>392k</td>
<td>363k</td>
<td>108k</td>
<td>67k</td>
<td>8.5k</td>
<td>5.7k</td>
<td>3.5k</td>
<td>2.5k</td>
<td></td>
</tr>
<tr>
<td>Pre-OpenAI SOTA</td>
<td>80.6/80.1</td>
<td>66.1</td>
<td>82.3</td>
<td>93.2</td>
<td>35.0</td>
<td>81.0</td>
<td>86.0</td>
<td>61.7</td>
<td>74.0</td>
</tr>
<tr>
<td>BiLSTM+ELMo+Attn</td>
<td>76.4/76.1</td>
<td>64.8</td>
<td>79.8</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>87.4</td>
<td>91.3</td>
<td>45.4</td>
<td>80.0</td>
<td>82.3</td>
<td>56.0</td>
<td>75.1</td>
</tr>
<tr>
<td>BERT\textsc{Base}</td>
<td>84.6/83.4</td>
<td>71.2</td>
<td>90.5</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\textsc{Large}</td>
<td>86.7/85.9</td>
<td>72.1</td>
<td>92.7</td>
<td>94.9</td>
<td>60.5</td>
<td>86.5</td>
<td>89.3</td>
<td>70.1</td>
<td>82.1</td>
</tr>
</tbody>
</table>

Number of parameters
- ELMo: 94 million
- BERT Base: 110 million
- BERT Large: 340 million
RoBERTa (Liu et al., 2019)

<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>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_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_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>
</tbody>
</table>

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 ($\S 3.2$) | 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    |

BERT_LARGE |
with BOOKS + WIKI | 13GB | 256 | 1M    | 90.9/81.8        | 86.6   | 93.7    |

XLNet_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    |

RoBERTa = BERT + more careful training + more data

https://github.com/pytorch/fairseq/tree/master/examples/roberta
Critical difference from ELMo: all BERT weights are fine-tuned for the target task (expensive but worth it)
BERT Applications

(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER
Currently in NLP

Explosion of pretrained contextualized word embedding models

- TagLM \cite{peters2017}
- CoVe \cite{mccann2017}
- ULMfit \cite{howard2018}
- ELMo \cite{peters2018}
- OpenAI GPT \cite{radford2018}
- BERT \cite{devlin2018}
- OpenAI GPT-2 \cite{radford2019}
- XLNet \cite{yang2019}
- SpanBERT \cite{joshi2019}
- RoBERTa \cite{liu2019a}
- AIBERT \cite{anonymous}
- T5 \cite{raffel2019}
- ...