Applications: NLP
Designing, Visualizing and Understanding Deep Neural Networks

CS W182/282A

Instructor: Sergey Levine
UC Berkeley
The Big Idea: Unsupervised Pretraining

Deep learning works best when we have a lot of data

**The big challenge:** how can we use **freely available** and **unlabeled** text data to help us apply deep learning methods to NLP?

**Good news:** there is plenty of data of text out there!

**Bad news:** most of it is unlabeled

1,000s of times more data **without** labels (i.e., valid English text in books, news, web) vs. labeled/paired data (e.g., English/French translations)
What can we do with unlabeled data?

How can we represent these... representations?

- local non-contextual representations
- word embeddings
- sentence embeddings
- global context-dependent representations
- pretrained language models
Start simple: how do we represent *words*?

\[
x_{1,1} = \begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
1 \\
\end{bmatrix}
\]

**dimensionality** = number of possible words

**index of this word**

not great, not terrible...

The pixels mean something!
Not much (not a great metric space), but they mean something

This means basically nothing by itself

**Maybe** if we had a more meaningful representation of words, then learning downstream tasks would be much easier!

**Meaningful** = vectors corresponding to *similar* words should be close together
Some examples of **good** embeddings
How do we learn embeddings?

**Basic idea:** the meaning of a word is determined by what *other* words occur in close proximity to it.

**Put another way:** the more interchangeable words are, the more similar they are.

**Example:** Seattle hotel vs. Seattle motel

---

**Basic principle:** pick a representation for each word such that its neighbors are “close” under this representation.

---

Examples & Images: Christopher Manning, CS224n
More formally...

Can we **predict** the neighbors of a word from its **embedding value**?

![Diagram showing the prediction of neighbors using embedding values](image)

- "context word" \( o \)
- "center word" \( c \)
- \( u \) and \( v \) vectors for all possible words
- \( u \) and \( v \) vectors for all possible words
- all possible \( c \) and \( o \) combinations
- e.g., for each word \( c \), pick all words \( o \) that are within 5 steps

\[
p(o|c) = \frac{\exp(u^T_o v_c)}{\sum_{w \in V} \exp(u^T_w v_c)}
\]

(learned) vector representation of \( c \)
(learned) vector representation of \( c \)

looks a bit like a **logistic regression** model

how to train? \[ \arg \max_{u_1, \ldots, u_n, v_1, \ldots, v_n} \sum_{c, o} \log p(o|c) \]

Examples & Images: Christopher Manning, CS224n
word2vec

\[
\arg_{u_1, \ldots, u_n, v_1, \ldots, v_n} \max \sum_{c,o} \log p(o|c) \quad \quad p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}
\]

\[\theta = \begin{bmatrix}
  v_{aardvark} \\
v_{a} \\
\ldots \\
v_{zebra} \\
u_{aardvark} \\
u_{a} \\
\ldots \\
u_{zebra}
\end{bmatrix}\]

- Why two vectors? Makes optimization easier
- What to do at the end? Average them
- This then gives us a representation of words!
Making word2vec tractable

\[
\arg \max_{u_1, \ldots, u_n, v_1, \ldots, v_n} \sum_{c, o} \log p(o|c) \quad p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}
\]

**Problem:** the vocabulary might be **huge**

**Another idea:** what if we instead have a **binary** classification problem ("is this the right word or not")?

\[
p(o \text{ is the right word}|c) = \sigma(u_o^T v_c) = \frac{1}{1 + \exp(-u_o^T v_c)}
\]

This is not enough! Why?

\[
p(o \text{ is the wrong word}|c) = \sigma(-u_o^T v_c) = \frac{1}{1 + \exp(u_o^T v_c)}
\]

\[
\arg \max_{u_1, \ldots, u_n, v_1, \ldots, v_n} \sum_{c, o} \left( \log p(o \text{ is right}|c) + \sum_{w} \log p(w \text{ is wrong}|c) \right)
\]

randomly chosen "negatives"

Making word2vec tractable: summary

\[ p(o \text{ is the right word}|c) = \sigma(u_o^T v_c) = \frac{1}{1 + \exp(-u_o^T v_c)} \]

\[ p(o \text{ is the wrong word}|c) = \sigma(-u_o^T v_c) = \frac{1}{1 + \exp(u_o^T v_c)} \]

\[
\underset{u_1, \ldots, u_n, v_1, \ldots, v_n}{\arg \max} \sum_{c, o} \left( \log p(o \text{ is right}|c) + \sum_w \log p(w \text{ is wrong}|c) \right) 
\]

\[
\underset{u_1, \ldots, u_n, v_1, \ldots, v_n}{\arg \max} \sum_{c, o} \left( \log \sigma(u_o^T v_c) + \sum_w \log \sigma(-u_w^T v_c) \right) 
\]

**Intuition:** push \( v_c \) toward \( u_o \) and away from other vectors \( u_w \)

word2vec examples

\[
\arg \max_{u_1,\ldots,u_n,v_1,\ldots,v_n} \sum_{c,o} \log p(o|c) \quad p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}
\]

Algebraic relations:

\[\text{vec(“woman”)–vec(“man”) \approx vec(“aunt”)–vec(“uncle”)}\]
\[\text{vec(“woman”)–vec(“man”) \approx vec(“queen”)–vec(“king”)}\]

This is a little bit idealized, most relationships are not nearly this “nice”
word2vec examples

\[ \arg_{u_1, \ldots, u_n, v_1, \ldots, v_n} \max \sum_{c,o} \log p(o|c) \quad \text{subject to} \quad \sum_{w \in V} \exp(u_w^T v_c) \neq 0 \]

\[ p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \]

Word2vec model computed from 6 billion word corpus of news articles

<table>
<thead>
<tr>
<th>Type of relationship</th>
<th>Word Pair 1</th>
<th>Word Pair 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common capital city</td>
<td>Athens, Greece</td>
<td>Oslo, Norway</td>
</tr>
<tr>
<td>All capital cities</td>
<td>Astana, Kazakhstan</td>
<td>Harare, Zimbabwe</td>
</tr>
<tr>
<td>Currency</td>
<td>Angola, kwanza</td>
<td>Iran, rial</td>
</tr>
<tr>
<td>City-in-state</td>
<td>Chicago, Illinois</td>
<td>Stockton, California</td>
</tr>
<tr>
<td>Man-Woman</td>
<td>brother, sister</td>
<td>grandson, granddaughter</td>
</tr>
<tr>
<td>Adjective to adverb</td>
<td>apparently, impossibly</td>
<td>rapidly, unethical</td>
</tr>
<tr>
<td>Opposite</td>
<td>possibly, great</td>
<td>ethical, tough</td>
</tr>
<tr>
<td>Comparative</td>
<td>easy, easiest</td>
<td>lucky, read</td>
</tr>
<tr>
<td>Superlative</td>
<td>think, thinking</td>
<td>read, reading</td>
</tr>
<tr>
<td>Present Participle</td>
<td>Switzerland, Swiss</td>
<td>Cambodia, swan</td>
</tr>
<tr>
<td>Nationality adjective</td>
<td>walking, walked</td>
<td>swimming, swan</td>
</tr>
<tr>
<td>Past tense</td>
<td>mouse, mice</td>
<td>dollar, dollars</td>
</tr>
<tr>
<td>Plural nouns</td>
<td>work, works</td>
<td>speak, speaks</td>
</tr>
<tr>
<td>Plural verbs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

word2vec examples

\[
\arg \max_{u_1, \ldots, u_n, v_1, \ldots, v_n} \sum_{c, o} \log p(o|c) \quad p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}
\]

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>France - Paris</td>
<td>Italy: Rome</td>
<td>Japan: Tokyo</td>
<td>Florida: Tallahassee</td>
</tr>
<tr>
<td>big - bigger</td>
<td>small: larger</td>
<td>cold: colder</td>
<td>quick: quicker</td>
</tr>
<tr>
<td>Miami - Florida</td>
<td>Baltimore: Maryland</td>
<td>Dallas: Texas</td>
<td>Kona: Hawaii</td>
</tr>
<tr>
<td>Einstein - scientist</td>
<td>Messi: midfielder</td>
<td>Mozart: violinist</td>
<td>Picasso: painter</td>
</tr>
<tr>
<td>Sarkozy - France</td>
<td>Berlusconi: Italy</td>
<td>Merkel: Germany</td>
<td>Koizumi: Japan</td>
</tr>
<tr>
<td>copper - Cu</td>
<td>zinc: Zn</td>
<td>gold: Au</td>
<td>uranium: plutonium</td>
</tr>
<tr>
<td>Berlusconi - Silvio</td>
<td>Sarkozy: Nicolas</td>
<td>Putin: Medvedev</td>
<td>Obama: Barack</td>
</tr>
<tr>
<td>Microsoft - Windows</td>
<td>Google: Android</td>
<td>IBM: Linux</td>
<td>Apple: iPhone</td>
</tr>
<tr>
<td>Microsoft - Ballmer</td>
<td>Google: Yahoo</td>
<td>IBM: McNealy</td>
<td>Apple: Jobs</td>
</tr>
<tr>
<td>Japan - sushi</td>
<td>Germany: bratwurst</td>
<td>France: tapas</td>
<td>USA: pizza</td>
</tr>
</tbody>
</table>
What do we do with this?

➢ Use as word representation in place of one-hot vectors

➢ Much more meaningful for downstream applications

➢ Can train word embeddings on large unlabeled corpus, and then use it as input into a supervised model trained on a much smaller corpus

➢ Could think of it as a simple type of **unsupervised pretraining**

---

Pretrained Language Models
Contextual representations

**Word embeddings** associate a vector with each word. This can make for a much **better** representation than just a one-hot vector! However, the vector **does not** change if the word is used in different ways!

Let’s play baseball

I saw a play yesterday

same word2vec representation, even though they mean different things!

Can we learn representations that depend on context?

**High level idea:**

1. Train a **language model**
2. Run it on a sentence
3. Use its **hidden state**

   use the hidden state as the representation for downstream tasks

---

**Question 1:** how to train the best language model for this?

**Question 2:** how to use this language model for downstream tasks?
The age of Sesame Street characters

**ELMo**: bidirectional LSTM model used for context-dependent embeddings

**BERT**: transformer language model used for context-dependent embeddings

Both the forward and backward LM are trained as language models.

Predict the next (or previous) word.

Together, all these hidden states form a representation of the word “cute.”
Using ELMo

simple version: $\text{ELMO}_t = [h_{t,2}^{\text{fwd}}, h_{t,2}^{\text{bwd}}]$

complex version: $\text{ELMO}_t = \gamma \sum_{i=1}^{L} w_i [h_{t,i}^{\text{fwd}}, h_{t,i}^{\text{bwd}}]$
Using ELMo

This is just an example, the actual ELMo paper does not test on translation, but does test:
- Question answering
- Textual entailment
- Semantic role labeling
- Coreference resolution
- Named entity extraction
- Sentiment analysis

And LMs help with all of these!
ELMo Summary

- Train **forward** and **backward** language models on a large corpus of **unlabeled** text data.
- Use the (concatenated) forward and backward LSTM states to represent the word **in context**.
- Concatenate the **ELMo representation** to the word embedding (or one-hot vector) as an **input** into a downstream task-specific sequence model.
  - This provides a **context specific** and **semantically meaningful** representation of each token.
BERT and Friends
The age of Sesame Street characters

**ELMo**: bidirectional LSTM model used for context-dependent embeddings

**BERT**: transformer language model used for context-dependent embeddings
Can we use a transformer instead?

Before:
- Repeated $N_x$ position-wise softmax

Now:
- $\hat{y}_1$, $\hat{y}_2$, $\hat{y}_3$, $\hat{y}_4$
- Position-wise nonlinear network
- Masked self-attention
- Position-wise encoder

- This model has a direction (forward, depends on masking used in self-attention)
- ELMo is bidirectional, this isn’t
- We could train two transformers, and make “transformer ELMo”
- But is there a better way? Can we simply remove the mask in self-attention and have one transformer?
  - What would go wrong?

Need masking to have a proper language model.
Bidirectional transformer LMs

It’s trivially easy to get the right answer, since self-attention can access the “right answer” at time $t$ from the input at time $t+1$!
Bidirectional transformer LMs

**Main idea:** needing to predict missing words forces the model to “work hard” to learn a good representation.

Without the need for masked self-attention!

This makes it **bidirectional**

**Input:** I [MASK] therefore I [MASK]

**Output:** I think therefore I am

**BERT** is essentially the “encoder” part of a transformer with 15% of inputs replaced with [MASK]

randomly **mask out** some input tokens

**masked self-attention**

**position-wise encoder**

**position-wise nonlinear network**

**position-wise softmax**
Training BERT

input consists of two sentences why?
many downstream tasks require processing two sentences:
- question answering
- natural language inference

pairs of sentences in the data are transformed in two ways:
1. Randomly replace 15% of the tokens with [MASK]
2. Randomly swap the order of the sentences 50% of the time

binary classifier output:
does first sentence follow the second sentence, or precede it?

this forces learning sentence-level representations

reconstruct all tokens at each time step (must predict actual token in place of [MASK])
forces learning context-dependent word-level representations

input consists of two sentences why?
many downstream tasks require processing two sentences:
Using BERT

- **binary classifier output**: A before B vs. A after B
- **task classification output**: entailment classification, semantic equivalence (e.g., Quora question pair), sentiment classification

1. Put a cross-entropy loss on **only** the first output (replaces the sentence order classifier)
2. Finetune the **whole** model end-to-end on the new task
Using BERT

- Finetune named entity label for each position (person name, location, other categories)
- Highlight which span of paragraph contains answer
- Classification tasks
Using BERT to get features

We can also pull out features, just like with ELMo!

Credit: Jay Alammar: http://jalammar.github.io/illustrated-bert/
Using BERT to get features

We can also pull out features, just like with ELMo!

Credit: Jay Alammar: http://jalammar.github.io/illustrated-bert/
BERT results are extremely good

GLUE test result (battery of varied natural language understanding 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>Pre-OpenAI SOTA</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>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_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_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>

Since then, it has been applied to nearly every NLP task you can imagine, and often makes a **huge** difference in performance.
GPT et al.

➢ One-directional (forward) transformer models do have one **big** advantage over BERT. Can you guess what it is?

➢ *Generation* is not really possible with BERT, but a forward (masked attention) model can do it!

➢ GPT (GPT-2, GPT-3, etc.) is a classic example of this.
OpenAI GPT-2 generated text

**Input:** In a shocking finding, scientists 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.

**Output:** 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.
Pretrained language models summary

**BERT**
- bidirectional transformer
- + great representations
- - can’t generate text

**OpenAI GPT**
- one-directional transformer
- + can generate text
- + OK representations

**ELMo**
- bidirectional LSTM
- - OK representations
- (largely supplanted by BERT)
Pretrained language models summary

- Language models can be trained on very large and unlabeled datasets of text (e.g., Wikipedia text). Often these are 100s or even 1000s of millions of sentences!

- Internal learned representations depend on context: the meaning of a word is informed by the whole sentence!

- Can even get us representations of entire sentences (e.g., the first output token for BERT)

- Can be used to either extract representations to replace standard word embeddings...

- ...or directly finetuned on downstream tasks (which means we modify all the weights in the whole language model, rather than just using pretrained model hidden states)

This is very important in modern NLP because it works extremely well!