Word Representations and Attention Models

Applied Machine Learning
Derek Hoiem
Today’s Lecture

• Representing natural language text as integers
  – Byte pair encoding
  – WordPiece

• Representing text tokens with continuous vectors
  – Word2Vec

• Masking Language Models and Attention
  – “Attention is all you need” transformers
Each pixel means little, but images can be interpreted by grouping and recognizing patterns in groups of groups of groups of pixels
CNNs iteratively process pixels->edges/colors->textures->sub-parts->parts->objects/scenes

Examples of strongly activating regions
But in text, the meaning is already in the words... right?
Which of these is more similar?

He sat on the chair, and it broke.

The chair says the department is broke.

After sitting, the seat is broken.
Which of these is more similar?

He sat on the chair, and it broke.

The chair says the department is broke.

After sitting, the seat is broken.
He sat on the chair, and it broke.

The chair says the department is broke.

After sitting, the seat is broken.

• Same word (character sequence) may mean different things

• Different words may mean similar things

• Word meaning depends on surrounding words
To analyze text, need to convert text to tokens

“Token”: an integer or vector that represents a data element, a unit of processing

• With integer tokens, the values are not continuous (e.g. 5 is no closer to 10 than 5000)
• With vector tokens, similarity/distance (typically L2, dot product or cosine) is meaningful
Word → Integer

• Each unique space-delimited character string is assigned to a different integer
  – To limit vocabulary size, assign only the most frequent words to integers
  – Others are <unk> (unknown)

• Pros and cons
  – Simple
  – Possible to compare/retrieve documents based on count of tokens
  – Many words map to unknown (e.g. 1298, Bart’s, Area-52, anachronism, ...)
  – Large vocabulary needed
  – Does not model similarity of related words like broke/broken

He sat on the chair, and it broke.

The chair says the department is broke.

After sitting, the seat is broken.
Character $\rightarrow$ Integer

• Each character is assigned to a unique integer

• Pros and cons
  – Simple
  – Every document within alphabet can be fully modeled
  – Small vocabulary (< 100 integers needed for English)
  – Sometimes, similar words will have similar sequences (broke/n)
  – Count of tokens is not meaningful
  – Character sequences are long
Subword $\rightarrow$ Integer

• Common sequences of characters are assigned to unique integers

• Pros and cons
  – Every document within alphabet can be fully modeled
  – Vocabulary size is flexible (e.g. 30K for BERT, 50K for GPT-3)
  – Sometimes, similar words will have similar sequences (broke/n)
  – Need to solve for good subword tokenization
<table>
<thead>
<tr>
<th>Character</th>
<th>Subword</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Chair is broken”</td>
<td>c, h, a, i, r, …</td>
<td>ch##, ##air, is, brok##, ##en</td>
</tr>
</tbody>
</table>

- **Vocabulary Size**: ~ 100, 4K-40K, > 30K
- **Completeness**: Perfect, Perfect, Incomplete
- **Independent Meaningfulness**: Bad, OK, Good
- **Sequence Length**: Long, Medium, A little shorter (e.g., 1.4 tokens per word)
- **Encodes word similarity**: Somewhat, A little better, Not at all
Subword Tokenizers: Byte Pair Encoding

1. Start with each character assigned to a unique token
2. Iteratively assign a new token to the most common pair of consecutive tokens, until max_tokens is reached

Initial array of 4 characters

<table>
<thead>
<tr>
<th>Replace aa by Z</th>
<th>Replace ab by Y</th>
<th>Replace ZY by X</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZabdZabac</td>
<td>ZYdZYYa</td>
<td>XdXac</td>
</tr>
<tr>
<td>Z=aa</td>
<td>Y=ab</td>
<td>X=ZY</td>
</tr>
<tr>
<td></td>
<td>Z=aa</td>
<td>Y=ab</td>
</tr>
</tbody>
</table>

Example from Wikipedia

aaabdaaabac

XZd $\rightarrow$ ZYZd $\rightarrow$ aaabaad
WordPiece Tokenizer (Sennrich et al., Wu et al. 2016)

- **Word:** Jet makers feud over seat width with big orders at stake
- **wordpieces:** _Jet_makers_feud_over_seat_width_with_big_orders_at_stake

---

**Algorithm 1** Learn BPE operations

```python
import re, collections
def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i]+symbols[i+1]] += freq
    return pairs
def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(''.join(pair))
    p = re.compile(r'''
        (?<=%s)       \s*   # previous space
        (?=%s)       \s*   # next space
    ''' % (re.escape(pair[0]), re.escape(pair[1])), re.VERBOSE | re.IGNORECASE)
    for word in v_in:
        w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
    return v_out
vocab = ['low', 'lowest', 'newer', 'wider']
nv = get_stats(vocab)
for i in range(num_merges):
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
print(vocab)
```

For each merge:
1. Count token pair frequencies in dataset
2. Select most frequent pair
3. Merge that “best” pair
   a. Assign best pair to new token
   b. Replace all instances of best pair in dataset with that token

Figure 1: BPE merge operations learned from dictionary {'low', 'lowest', 'newer', 'wider'}.
Try it

Do first two merges of:
Your cat cannot do the can-can, can he?

For each merge:
1. Count token pair frequencies in dataset
2. Select most frequent pair
3. Merge that “best” pair
   a. Assign best pair to new token
   b. Replace all instances of best pair in dataset with that token
How can we better encode word similarity?

• Different words are related to each other

• Encode “meaning” in a continuous vector

• Learn these vectors based on surrounding words
Word2Vec (Mikolov et al. 2013)

For each word, solve for a continuous vector representation:

• CBOW: predict center word as average of surrounding words (after projecting each word to a vector)
• Skip-Gram: each word (after projecting to a vector) predicts each surrounding word with a linear model

Train by gradient descent

- At the end, each word integer can be replaced by a fixed-length continuous vector
- These vectors can predict word relationships

Table 1: Examples of five types of semantic and nine types of syntactic questions in the Semantic-Syntactic Word Relationship test set.

<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, Astana</td>
<td>Greece, Kazakhstan</td>
</tr>
<tr>
<td>All capital cities</td>
<td>Astana, Angola</td>
<td>kwanza, Illinois</td>
</tr>
<tr>
<td>Currency</td>
<td>Chicago, brother</td>
<td>sister</td>
</tr>
<tr>
<td>City-in-state</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Man-Woman</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adjective to adverb</td>
<td>apparent, possibly</td>
<td>rapidly, unethical</td>
</tr>
<tr>
<td>Opposite</td>
<td>great, possibly</td>
<td>ethical, tough</td>
</tr>
<tr>
<td>Comparative</td>
<td>easy, possibly</td>
<td>lucky, read</td>
</tr>
<tr>
<td>Superlative</td>
<td>think, possibly</td>
<td>Cambodiaswimming</td>
</tr>
<tr>
<td>Present Participle</td>
<td>Switzerland, walking</td>
<td>Swiss</td>
</tr>
<tr>
<td>Nationality adjective</td>
<td>mouse</td>
<td>walked</td>
</tr>
<tr>
<td>Past tense</td>
<td>work</td>
<td>mice</td>
</tr>
<tr>
<td>Plural nouns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plural verbs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Comparison of models trained using the DistBelief distributed framework. Note that training of NNLM with 1000-dimensional vectors would take too long to complete.

<table>
<thead>
<tr>
<th>Model</th>
<th>Vector Dimensionality</th>
<th>Training words</th>
<th>Accuracy [%]</th>
<th>Training time [days x CPU cores]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Semantic</td>
<td>Syntactic</td>
</tr>
<tr>
<td>NNLM</td>
<td>100</td>
<td>6B</td>
<td>34.2</td>
<td>64.5</td>
</tr>
<tr>
<td>CBOW</td>
<td>1000</td>
<td>6B</td>
<td>57.3</td>
<td>68.9</td>
</tr>
<tr>
<td>Skip-gram</td>
<td>1000</td>
<td>6B</td>
<td>66.1</td>
<td>65.1</td>
</tr>
</tbody>
</table>
Word2Vec predicted relationship examples

Table 8: Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).

<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>

E.g., Paris – France + Italy = Rome
Word2Vec demos

https://www.cs.cmu.edu/~dst/WordEmbeddingDemo/

https://remykarem.github.io/word2vec-demo/

2 Minute Break: think of other relations to try
A new type of data processing

• Linear: output is sum of weights times features

• Convolution: output at each position is linear function of features within a window

• Attention: given a set of <key, value> pairs and a <query>, output is sum of values weighted by key-query similarity
Cross-Attention

<key $k$, value $v$>: a data element, in which key is used for matching and value is used to output

<query $q$>: used to match keys and accumulate values

\[
\text{out}(q) = \left[ \sum_i s(k_i, q)v_i \right] / \left[ \sum_i s(k_i, q) \right]
\]

Similarity of $i$th key and query $i$th value Make similarities sum to 1
Cross-attention simple example

\[
\text{out}(q) = \left[ \sum_i s(k_i, q)v_i \right] / \left[ \sum_i s(k_i, q) \right]
\]

\[
s(k,q) = \frac{1}{(k-q)^2}
\]

<key, value> pairs: <1, 1>, <7, -1>, <5, -1>

query: 4
\[
\text{out} = \left( \frac{1}{3^2} \cdot 1 + \frac{1}{3^2} \cdot 1 + \frac{1}{1^2} \cdot 1 \right) / \left( \frac{1}{3^2} + \frac{1}{3^2} + \frac{1}{1^2} \right)
\]
\[
= -0.818
\]

query: 0
\[
\text{out} = \left( \frac{1}{1^2} \cdot 1 + \frac{1}{6^2} \cdot (-1) + \frac{1}{4^2} \cdot (-1) \right) / \left( \frac{1}{1^2} + \frac{1}{6^2} + \frac{1}{4^2} \right)
\]
\[
= 0.834
\]
Self-attention

- Key=value
- Each key is also a query

\[ S(k,q) = \frac{1}{(k-q)^2 + 1} \]

Input: 1, 7, 5

Output: \((\frac{1}{1} + \frac{1}{6+1} \cdot 7 + \frac{1}{4+1} \cdot 5) / (1 + \frac{1}{6+1} + \frac{1}{4+1}) = 1.37, 6.54, 5.13)\)

Apply again: \((1.76, 6.06, 5.29)\)

Apply again: \((2.19, 5.64, 5.42)\)
Another example of self attention

\[ s(k,q) = \frac{1}{(k-q)^2 + 1} \]

<table>
<thead>
<tr>
<th>Input (k,q,v)</th>
<th>iter 1</th>
<th>iter 2</th>
<th>iter 3</th>
<th>iter 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000</td>
<td>1.497</td>
<td>1.818</td>
<td>1.988</td>
<td>2.147</td>
</tr>
<tr>
<td>9.000</td>
<td>8.503</td>
<td>8.182</td>
<td>8.012</td>
<td>7.853</td>
</tr>
<tr>
<td>8.000</td>
<td>8.128</td>
<td>8.141</td>
<td>8.010</td>
<td>7.853</td>
</tr>
<tr>
<td>2.000</td>
<td>1.872</td>
<td>1.859</td>
<td>1.990</td>
<td>2.147</td>
</tr>
</tbody>
</table>
Attention

• Cross-Attention is an instance-based regression
• Self-Attention is a soft clusterer/aggregator and can be stacked
  – Adding multi-dimensional vectors can overlay multiple types of information, not just blend or replace

• Attention becomes extremely powerful and general when combined with learned similarity and non-linear feature transformations!
Transformer (Vaswani et al. 2017)

- Define similarity via linear projection with softmax
  \[ S(k_i, q) = \exp(k_i \cdot q) \]

Attention is all you need

\[
\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V
\]

Normalize by sqrt of dimensionality of keys
Transformer (Vaswani et al. 2017)

- One or more similarity functions can be learned with linear layers
  - If there are K similarities and D dimensions to input, each parallel linear layer outputs D/K values

Mathematical representation:

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

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

Attention is all you need
Transformers: general data processors

- **Input tokens can represent anything**: image patches, text tokens, audio, controls, etc.

- Invariant to order of tokens: **add positional embedding** to distinguish pos/type of input

- **Transformer block**:
  - Apply multi-head attention
  - Apply 2-layer MLP with ReLU to each token separately
  - Residual and layer norm (over all tokens) after each

- Can stack any number of transformer blocks

**Attention is all you need**
Positional encodings

- Transformer processing does not depend on position of token
  - This is kind of similar to convolution, as each “patch” or token vector is processed independently, but no overlap between patches
  - But to compare between tokens, relative position may be important
- Sinusoidal encoding (on right) is such that a dot product between encodings corresponds to positional similarity
- Learned or even fixed random encodings also work similarly in practice

\[
PE_{(\text{pos}, 2i)} = \sin(\text{pos}/10000^{2i/d_{\text{model}}}) \\
PE_{(\text{pos}, 2i+1)} = \cos(\text{pos}/10000^{2i/d_{\text{model}}})
\]
Language Transformer: Complete Architecture

- WordPiece tokens (integers) are mapped to learned 512-d vectors
- Positional encoding added to each vector
- N=6 transformer blocks applied to input
- Until <EOS> is output:
  - Process input + output so far
  - Output most likely word (after more attention blocks and linear model)

Attention is all you need
Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attention here shown only for the word ‘making’. Different colors represent different heads. Best viewed in color.
Figure 4: Two attention heads, also in layer 5 of 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word ‘its’ for attention heads 5 and 6. Note that the attentions are very sharp for this word.

Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.
Application to Translation

- English-German
  - 4.5M sentence pairs
  - 37K tokens
- English-French
  - 36M sentences
  - 32K tokens
- Base models trained on 8 P100s for 12 hours
- Big models (2x token dim, 3x training steps) trained for 3.5 days
- Adam optimizer: learning rate ramps up for 4K iterations, then down
- Regularization: drop-out, L2 weight, label smoothing

Attention is all you need
Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

<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>25.16</td>
<td>9.6 ( \cdot 10^{18}</td>
</tr>
<tr>
<td>MoE [32]</td>
<td>26.03</td>
<td>2.0 ( \cdot 10^{19}</td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [39]</td>
<td>40.4</td>
<td>8.0 ( \cdot 10^{20}</td>
</tr>
<tr>
<td>GNMT + RL Ensemble [38]</td>
<td>26.30</td>
<td>1.8 ( \cdot 10^{20}</td>
</tr>
<tr>
<td>ConvS2S Ensemble [9]</td>
<td>26.36</td>
<td>7.7 ( \cdot 10^{19}</td>
</tr>
<tr>
<td>Transformer (base model)</td>
<td>27.3</td>
<td>3.3 ( \cdot 10^{18}</td>
</tr>
<tr>
<td>Transformer (big)</td>
<td>28.4</td>
<td>2.3 ( \cdot 10^{19}</td>
</tr>
</tbody>
</table>
Things to remember

Sub-word tokenization based on byte-pair encoding is an effective way to turn natural text into a sequence of integers.

Learned vector embeddings of these integers model the relationships between words.

Attention is a general processing mechanism that regresses or clusters values.

Stacked transformer blocks are a powerful network architecture that alternates attention and MLPs.

Next class: Transformers in Language and Vision

- BERT
- ViT
- Unified-IO