Announcements

• Project Milestone 2 due **Wednesday, November 9 at 8pm**
  • Will open GradeScope submission tonight
  • **GPU option:** AWS SageMaker Studio

• Quiz 9 is due **Thursday, November 10 at 8pm**

• HW 5 due **Wednesday, November 16**
  • Please start early!
Word Embeddings, Ctd

Osbert Bastani and Zachary G. Ives
CIS 4190/5190 – Fall 2022
Recall: Similar Words Are Used in Similar Contexts

“I buy food for my cat at the pet store”

vs

“I buy food for my dog at the pet store”

vs

“My car guzzles gas”

Intuition: we can “semantically cluster” words based on vectors describing the contexts of their occurrences
Capturing Context in a Vector

Term-Frequency model:
- For each term, count # occurrences in each document in a corpus
- Vector is \(|\text{terms}|\) by \(|\text{documents}|\)

<table>
<thead>
<tr>
<th>Words</th>
<th>Article</th>
<th>Cat</th>
<th>Dog</th>
<th>Apple Inc.</th>
<th>Apple (fruit)</th>
<th>Microsoft Inc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td>377</td>
<td>370</td>
<td>842</td>
<td>231</td>
<td>286</td>
</tr>
<tr>
<td>the</td>
<td></td>
<td>929</td>
<td>787</td>
<td>1690</td>
<td>503</td>
<td>872</td>
</tr>
<tr>
<td>apple</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1091</td>
<td>166</td>
<td>14</td>
</tr>
</tbody>
</table>

A “windows” term-term model:
- For each term, count # co-occurrences with other words within an n-word window
- Vector is \(|\text{terms}|\) by \(|\text{terms}|\) but sparse (n non-zero entries)

<table>
<thead>
<tr>
<th>Words</th>
<th>Wikipedia</th>
<th>pet</th>
<th>play</th>
<th>tire</th>
<th>engine</th>
<th>run</th>
</tr>
</thead>
<tbody>
<tr>
<td>dog</td>
<td>872</td>
<td>649</td>
<td>1</td>
<td>7</td>
<td>378</td>
<td></td>
</tr>
<tr>
<td>cat</td>
<td>789</td>
<td>831</td>
<td>5</td>
<td>0</td>
<td>285</td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>12</td>
<td>4</td>
<td>290</td>
<td>927</td>
<td>562</td>
<td></td>
</tr>
</tbody>
</table>

These are huge vectors, likely with lots of zeros.

Can we get a more compact representation?
Why not *learn* a way of mapping to a reduced number of dimensions?

We’ll do this in a surprising (?) way:

- **Train a NN classifier** to predict words that will co-occur in the context by mapping them through a hidden layer with fewer dimensions

- Take the *learned weights* as a compact vector space representation!
Word2Vec Neural Network, Sketched

- Words: dog, car, cat
- Hidden layer:
  - \( V \)-dimensional one-hot encoding as input (\# unique words)
  - \( N \)-dimensional intermediate layer (200-300 dim)
- Context:
  - \( V \)-dimensional one-hot encoding as output (\# unique words)

The hidden layer has a smaller number of dimensions – we’ll learn \( N \) features useful in predicting context.
Word2Vec Training Data

“The quick brown fox jumped over the lazy dog” (n=2)

<table>
<thead>
<tr>
<th>Word</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick</td>
<td>[the, brown]</td>
</tr>
<tr>
<td>brown</td>
<td>[quick, fox]</td>
</tr>
<tr>
<td>fox</td>
<td>[brown, jumped]</td>
</tr>
<tr>
<td>jumped</td>
<td>[fox, over]</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
“The quick brown fox jumped over the lazy dog.”

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick</td>
<td>the</td>
</tr>
<tr>
<td>quick</td>
<td>brown</td>
</tr>
<tr>
<td>brown</td>
<td>quick</td>
</tr>
<tr>
<td>brown</td>
<td>fox</td>
</tr>
<tr>
<td>fox</td>
<td>brown</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Millions of training input-output pairs, from parsing huge, unlabeled datasets (e.g., all of Wikipedia)
Word2Vec Classifier

One-Hot Encoding for the Input Word

One-Hot Encoding for the Output Word

Source: https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html
Word2Vec Classifier

Input

\[ x_1, x_2, \ldots, x_i, \ldots, x_V \]

Vector of word \( i \)

Matrix \( W \)

Embedding matrix

\[ N \times N \text{-dimension vector} \]

Hidden

\[ h_1, h_2, h_3, \ldots, h_N \]

\[ V = X \]

Matrix \( W' \)

Context matrix

Output softmax

\[ y_1, y_2, \ldots, y_j, \ldots, y_V \]

\[ N = V \]

quick

\[ \text{brown} \]

\( N \) hidden units
where \( N \ll V \)

© 2019-22 D. Jayaraman, O. Bastani, Z. Ives
Word2Vec Classifier

Has $N$ columns, $V$ (vocabulary size) rows.
Each row corresponds to a word.

$i^{th}$ row = a vector representation for word $i$

“Target Embedding”
Word2Vec Classifier

Has V (vocabulary size) columns, N rows.
Each column corresponds to a word.

\[ i^{th} \text{ column} = \text{another vector representation for word } \#i \]

“Context Embedding”
After training, we can make our final word vector a concatenation of the two embeddings, or just use $W$. 
Standard softmax loss, then train the neural network.

\[
p(w_o | w_{in}) = \frac{\exp(v'_{w_o}^T v_{w_{in}})}{\sum_{k=1}^{V} \exp(v'_{w_k}^T v_{w_{in}})}
\]

Computing this denominator will be expensive. Remember that the vocabulary size \( V \) is of the order of millions of words!
Scaling Word2Vec Training

Simple Trick: Sample some random $K \ll V$ negative example words for each sample. e.g., $K=2, 5, 20$ etc. [“Negative sampling”]

Also means we need to update many fewer weights during each iteration of gradient descent.
Using Word2Vec Predictions

CBOW
Predict word from bag-of-words context

Skip-gram
Predict context from word

© 2019-22 D. Jayaraman, O. Bastani, Z. Ives
From Words to Documents

- Sentence2Vec, Paragraph2Vec scale these Word2Vec ideas to learn direct embeddings for sentences / paragraphs.
- However, much more common to treat as a sequence of words, and represent each word by its word2vec-style representation:

Simple “sequence-to-sequence” models like these produced huge advances in machine translation in 2014.

“I have a dog”

“j'ai un chien”
Properties of Word2Vec

Words that co-occur have vector representations that are close together (Euclidean distance).

“sofa” and “couch” (synonyms) will be close together, but also things like “hot” and “cold” (antonyms)

People say “It’s ____ outside today” for both

- “hot” and “cold” co-occur with the same words often in sentences.
Properties of Word2Vec

Vector operations (vector addition and vector subtraction) on word vectors often capture the semantic relationships of their words.

man : king :: woman: ?

Use in Practice

GLoVe is an alternative word vector embedding similar to word2vec

Available freely, and often used off-the-shelf:
• English word2vec weights trained on Google News data
• GloVe vectors trained on the Common Crawl dataset and a Twitter dataset.

If you have a lot of training data or a very different / niche domain (e.g., medical text), you might want to train your own word vectors on your dataset!
Summary So Far

• The journey to compact vector representations of words by their context:
  ▪ term frequency vectors
  ▪ term-term vectors
  ▪ word embeddings (learned by NN)

• We can use measures of vector similarity (e.g., cosine similarity, L1 or L2 distance, others) to find related terms
Words in Context

• While word2vec is trained based on context, after training, it is applied independently to each word
  • E.g., train linear regression of sum of word vectors, or n-grams

• **Why is this problematic?**
  • “He ate a tasty apple”
  • “He wrote his essay on his Apple computer”

• Both use the same embedding!
Recurrent Neural Networks

• Handle inputs/outputs that are sequences

• Naïve strategy
  • Pad inputs to fixed length and use feedforward network
  • Ignores temporal structure

• Recurrent neural networks (RNNs): Process input sequentially
Recurrent Neural Networks
Recurrent Neural Networks

\[ z^{(1)} \xrightarrow{f_{W_2}} z^{(2)} \xrightarrow{f_{W_4}} z^{(3)} \xrightarrow{f_{W_6}} z^{(4)} \xrightarrow{f_{W_8}} z^{(5)} \xrightarrow{f_{\beta}} \hat{y} \]

\[ x_1 \xrightarrow{f_{W_1}} \]
\[ x_2 \xrightarrow{f_{W_3}} \]
\[ x_3 \xrightarrow{f_{W_5}} \]
\[ x_4 \xrightarrow{f_{W_7}} \]
\[ x_5 \xrightarrow{f_{W_9}} \]
Recurrent Neural Networks

\[ z^{(1)} \xrightarrow{f_U} z^{(2)} \xrightarrow{f_U} z^{(3)} \xrightarrow{f_U} z^{(4)} \xrightarrow{f_U} z^{(5)} \xrightarrow{f_\beta} \hat{y} \]

\[ \begin{align*}
  x_1 &\xleftarrow{f_W} \\
  x_2 &\xleftarrow{f_W} \\
  x_3 &\xleftarrow{f_W} \\
  x_4 &\xleftarrow{f_W} \\
  x_5 &\xleftarrow{f_W}
\end{align*} \]
Recurrent Neural Networks

• Initialize $z^{(0)} = \vec{0}$

• Iteratively compute (for $t \in \{1, \ldots, T\}$):

$$z^{(t)} = g(Wx_t + Uz^{(t-1)})$$

• Compute output:

$$y = \beta^T z^{(T)}$$
Recurrent Neural Networks

Image captioning

Sentiment prediction

Machine translation

Video captioning

Fei-Fei Li, Justin Johnson, Serena Yeung
Recurrent Neural Networks

• Backpropagation works as before
  • For shared parameters, overall gradient is sum of gradient at each usage

• Exploding/vanishing gradients can be particularly problematic

• LSTM (“long short-term memory”) and GRU (“gated recurrent unit”) do clever things to better maintain hidden state
RNNs for Natural Language

• Apply RNN to sequence of words
  • **Encoding 1**: One-hot encoding of each word
  • **Encoding 2**: Sequence of word vectors

• **Unsupervised pretraining**
  • Train on dataset of text to predict next word (classification problem)
  • $x = w_1 w_2 \ldots w_t$ and $y = w_{t+1}$ (usually $y$ is one-hot even if $x$ is not)

• Finetune pretrained RNN on downstream task
“Transfer Learning” Strategy

• **Step 0:** Pretrained on a large *unlabeled* text dataset
  • Also called “self-supervised”
  • Trained using supervised learning, but labels are predicting data itself

• **Step 1:** Replace next-word prediction layer with new layer for task

• **Step 2:** Train new layer or finetune end-to-end
RNNs for Natural Language

• **Shortcomings**
  • Unidirectional information flow (must remember everything relevant)
  • Computation time is proportional to sequence length

• **Improvements/alternatives**
  • Bidirectional LSTMs
  • CNNs
  • Transformers
ELMo Model

- **Bidirectional LSTM**: Combine one LSTM to predict next word given previous words, another to predict previous word given later words
CNNs

• **Model**
  • 1D convolutional layers
  • Input is word embedding sequence
  • # channels is word embedding dimension
CNNs

• **Shortcomings**
  • Hard to reason about interactions between words that are far apart

Figure credit to d2l.ai
Transformers

• Composition of **self-attention layers**

• **Intuition**
  • Want sparse connection structure of CNNs, but with different structure
  • Can we **learn** the connection structure?
Self-Attention Layer

- Self-attention layer:
  \[ y[t] = \sum_{s=1}^{T} \text{attention}(x[s], x[t]) \cdot f(x[s]) \]

  - Input first processed by local layer \( f \)
  - All inputs can affect \( y[t] \)
  - But weighted by \( \text{attention}(x[s], x[t]) \)

- Resembles convolution but connection is learned instead of hardcoded

Figure credit to d2l.ai
Self-Attention Layer

• Self-attention layer:

\[
y[t] = \sum_{s=1}^{T} \text{softmax}([\text{query}(x[t])^\top \text{key}(x[s])]) \cdot \text{value}(x[s])
\]

• Here, we have (learnable parameters are \(W_Q\), \(W_K\), and \(W_V\)):

\[
\begin{align*}
\text{query}(x[s]) &= W_Q x[s] \\
\text{key}(x[s]) &= W_K x[s] \\
\text{value}(x[s]) &= W_V x[s]
\end{align*}
\]
Self-Attention Layer

- **Query vectors:** $v^Q$
- **Key vectors:** $v^K$
- **Value vectors:** $v^V$

**Equation:**

$$T \times T \text{ matrix } \text{attention}_{ij} = \text{softmax}(matrix_{ij})$$

$$matrix_{ij} = v_i^Q \cdot v_j^K$$

**Diagram:**

- Input sequence: $x[1], \ldots, x[T]$
- Query transformation: $W_Q$
- Key transformation: $W_K$
- Value transformation: $W_V$
- Attention calculation:
  $$\text{attention}_{ij} = \text{softmax}(matrix_{ij})$$
- Output sequence: $y[1], \ldots, y[T]$

**Note:** The diagram illustrates the interaction between query, key, and value vectors through matrix multiplication and softmax functions to compute self-attention scores.
Self-attention

input #1

1 0 1 0

input #2

0 2 0 2

input #3

1 1 1 1
Transformers

• Stack self-attention layers to form a neural network architecture

• **Examples:**
  • **BERT:** Bidirectional transformer similar to ELMo, useful for prediction
  • **GPT:** Unidirectional model suited to text generation

• **Aside:** Self-attention layers subsume convolutional layers
  • Use “positional encodings” as auxiliary input so each input knows its position
  • [https://d2l.ai/chapter_attention-mechanisms/self-attention-and-positional-encoding.html#](https://d2l.ai/chapter_attention-mechanisms/self-attention-and-positional-encoding.html#)
  • Then, the attention mechanism can learn convolutional connection structure
Visualizing Attention Outputs

As aliens entered our planet and began to colonized Earth, a certain group of extraterrestrials began to manipulate our society through their influences of a certain number of the elite to keep and iron grip over the populace.

https://transformer.huggingface.co/
Applications: Spam Detection

• “Bag of words” + SVMs for spam classification

• **Features:** Words like “western union”, “wire transfer”, “bank” are suggestive of spam
Applications: Search

• Use “bag of words” + TF-IDF to identify relevant documents for a search query
Applications: Virtual Assistants

• Use word vectors to predict intent of queries users ask
Applications: Question Answering

- Models like ELMo and BERT can be used to answer questions based on a given passage.
Applications: Generation

- Language models such as GPT can automatically generate text for applications such as video games.
Transformers for Computer Vision

Figure credit to “End-to-End Object Detection with Transformers”