Lecture 7: Word Embeddings
Mini 1 grades out tonight or tomorrow

Project 1 due Tuesday
Clarification: Forward-Backward

- Forward-backward slides showed forward-backward in the HMM case (emission scores were probabilities $P(x_i | y_i)$)

- For CRFs: use transition/emission potentials (computed from features + weights) instead of probabilities

- Lecture 5 notes updated with F-B on CRFs
Recall: Feedforward NNs

\[ P(y|x) = \text{softmax}(Wg(Vf(x))) \]

- \( f(x) \)\n- \( V \)\n- \( g \)\n- \( z \)\n- \( W \)\n- softmax
- \( P(y|x) \)

**Diagonal Elements:**
- \( n \) features
- \( d \times n \) matrix
- \( d \) hidden units
- nonlinearity (tanh, relu, ...)
- \( \text{num}\_\text{classes} \times d \) matrix
- \( \text{num}\_\text{classes} \) probs
Recall: Backpropagation

\[ P(y|x) = \text{softmax}(W_g(V_f(x))) \]
This Lecture

- Training tips
- Word representations
- word2vec/GloVe
- Evaluating word embeddings
Training Tips
Batching data gives speedups due to more efficient matrix operations

Need to make the computation graph process a batch at the same time

```python
# input is [batch_size, num_feats]
# gold_label is [batch_size, num_classes]
def make_update(input, gold_label):
    ...  
    probs = ffnn.forward(input)  # [batch_size, num_classes]
    loss = torch.sum(torch.neg(torch.log(probs)).dot(gold_label))
    ...
```

Batch sizes from 1-100 often work well
Training Basics

- Basic formula: compute gradients on batch, use first-order optimization method (SGD, Adagrad, etc.)

- How to initialize? How to regularize? What optimizer to use?

- This lecture: some practical tricks. Take deep learning or optimization courses to understand this further
How does initialization affect learning?

\[ P(y|x) = \text{softmax}(Wg(Vf(x))) \]

- How do we initialize \( V \) and \( W \)? What consequences does this have?
- *Nonconvex* problem, so initialization matters!
How does initialization affect learning?

- Nonlinear model...how does this affect things?

- If cell activations are too large in absolute value, gradients are small.

- ReLU: larger dynamic range (all positive numbers), but can produce big values, can break down if everything is too negative.
Initialization

1) Can’t use zeroes for parameters to produce hidden layers: all values in that hidden layer are always 0 and have gradients of 0, never change

2) Initialize too large and cells are saturated

- Can do random uniform / normal initialization with appropriate scale

- Glorot initializer: $U \left[-\sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}}, +\sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}}\right]$

  - Want variance of inputs and gradients for each layer to be the same

- Batch normalization (Ioffe and Szegedy, 2015): periodically shift+rescale each layer to have mean 0 and variance 1 over a batch (useful if net is deep)
Dropout

- Probabilistically zero out parts of the network during training to prevent overfitting, use whole network at test time

- Form of stochastic regularization

- Similar to benefits of ensembling: network needs to be robust to missing signals, so it has redundancy

- One line in Pytorch/Tensorflow

Srivastava et al. (2014)
Adam (Kingma and Ba, ICLR 2015): very widely used. Adaptive step size + momentum

Wilson et al. NIPS 2017: adaptive methods can actually perform badly at test time (Adam is in pink, SGD in black)

One more trick: gradient clipping (set a max value for your gradients)
Word Representations
Word Representations

- Neural networks work very well at continuous data, but words are discrete.
- Continuous model <-> expects continuous semantics from input.
- “You shall know a word by the company it keeps” Firth (1957)

[Finch and Chater 92, Shuetze 93, many others]
Brown clusters: hierarchical agglomerative \textit{hard} clustering (each word has one cluster, not some posterior distribution like in mixture models)

\begin{itemize}
  \item Maximize $P(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)$
  \item Useful features for tasks like NER, not suitable for NNs
\end{itemize}

Brown et al. (1992)
Word Embeddings

- Part-of-speech tagging with FFNNs
- Fed raises **interest** rates in order to ...
- Word embeddings for each word form input
- What properties should these vectors have?

Botha et al. (2017)
Word Embeddings

- Want a vector space where similar words have similar embeddings

  \textit{the movie was great} \\
  \sim \\
  \textit{the movie was good}

- Goal: come up with a way to produce these embeddings

- For each word, want “medium” dimensional vector (50-300 dims) representing it
word2vec/GloVe
Continuous Bag-of-Words

- Predict word from context
  
  \[ P(w | w_{-1}, w_{+1}) = \text{softmax} \left( W \left( c(w_{-1}) + c(w_{+1}) \right) \right) \]

- Parameters:
  - \( d \times |V| \) (one \( d \)-length context vector per voc word),
  - \( |V| \times d \) output parameters \( (W) \)

**the dog bit the man**

- Mikolov et al. (2013)
Skip-Gram

- Predict one word of context from word

\[ \text{the dog bit the man} \]

\[ P(w' | w) = \text{softmax}(W e(w)) \]

- Another training example: \textit{bit} -> \textit{the}

- Parameters: \( d \times |V| \text{ vectors}, |V| \times d \text{ output parameters (W) (also usable as vectors!)} \)

Mikolov et al. (2013)
Hierarchical Softmax

\[ P(w|w_{-1}, w_{+1}) = \text{softmax} \left( W(c(w_{-1}) + c(w_{+1})) \right) \quad P(w'|w) = \text{softmax}(We(w)) \]

- Matmul + softmax over \(|V|\) is very slow to compute for CBOW and SG

- Standard softmax: 
  \[ [|V| \times d] \times d \]

- Hierarchical softmax:
  \[ \log(|V|) \text{ dot products of size } d, \]
  \[ |V| \times d \text{ parameters} \]

- Huffman encode vocabulary, use binary classifiers to decide which branch to take

- \(\log(|V|)\) binary decisions

Mikolov et al. (2013)
Skip-Gram with Negative Sampling

- Take (word, context) pairs and classify them as “real” or not. Create random negative examples by sampling from unigram distribution.

  \[(bit, the) \Rightarrow +1\]
  \[(bit, cat) \Rightarrow -1\]
  \[(bit, a) \Rightarrow -1\]
  \[(bit, fish) \Rightarrow -1\]

  \[P(y = 1|w, c) = \frac{e^{w \cdot c}}{e^{w \cdot c} + 1}\]

- \(d \times |V|\) vectors, \(d \times |V|\) context vectors (same # of params as before)

- Objective = \(\log P(y = 1|w, c) + \frac{1}{k} \sum_{i=1}^{n} \log P(y = 0|w_i, c)\)

Mikolov et al. (2013)
Connections with Matrix Factorization

- Skip-gram model looks at word-word co-occurrences and produces two types of vectors

- Looks almost like a matrix factorization...can we interpret it this way?

Levy et al. (2014)
Skip-Gram as Matrix Factorization

\[ M_{ij} = \text{PMI}(w_i, c_j) - \log k \]

\[ \text{PMI}(w_i, c_j) = \frac{P(w_i, c_j)}{P(w_i)P(c_j)} = \frac{\text{count}(w_i, c_j)}{D} \cdot \frac{1}{\text{count}(w_i)} \cdot \frac{1}{\text{count}(c_j)} \]

Skip-gram objective exactly corresponds to factoring this matrix:

- If we sample negative examples from the uniform distribution over words
- ...and it’s a weighted factorization problem (weighted by word freq)

Levy et al. (2014)
GloVe (Global Vectors)

- Also operates on counts matrix, weighted regression on the log co-occurrence matrix

- Objective = \( \sum_{i,j} f(\text{count}(w_i, c_j)) \left( w_i^\top c_j + a_i + b_j - \log \text{count}(w_i, c_j) \right)^2 \)

- Constant in the dataset size (just need counts), quadratic in voc size

- By far the most common word vectors used today (5000+ citations)

Pennington et al. (2014)
fastText: Sub-word Embeddings

- Same as SGNS, but break words down into n-grams with n = 3 to 6

  where:
  - 3-grams: <wh, whe, her, ere, re>
  - 4-grams: <whe, wher, here, ere>,
  - 5-grams: <wher, where, here>,
  - 6-grams: <where, where>

- Replace $w \cdot c$ in skip-gram computation with $\left( \sum_{g \in \text{ngrams}} w_g \cdot c \right)$

- Advantages?

Bojanowski et al. (2017)
Using Word Embeddings

- Approach 1: learn embeddings as parameters from your data
  - Often works pretty well
- Approach 2: initialize using GloVe, keep fixed
  - Faster because no need to update these parameters
- Approach 3: initialize using GloVe, fine-tune
  - Works best for some tasks
Context-dependent Embeddings

- How to handle different word senses? One vector for *balls*
  - Train a neural language model to predict the next word given previous words in the sentence, use its internal representations as word vectors
  - *Context-sensitive* word embeddings: depend on rest of the sentence
  - *Huge* improvements across nearly all NLP tasks over GloVe

Peters et al. (2018)
Compositional Semantics

- What if we want embedding representations for whole sentences?

- *Skip-thought* vectors (Kiros et al., 2015), similar to skip-gram generalized to a sentence level (more later)

- Is there a way we can compose vectors to make sentence representations? Summing?

- Will return to this in a few weeks as we move on to syntax and semantics
Evaluation
Evaluating Word Embeddings

- What properties of language should word embeddings capture?

- Similarity: similar words are close to each other

- Analogy:
  - good is to best as smart is to ???
  - Paris is to France as Tokyo is to ???

![Diagram showing word embeddings in a vector space with examples of similarity and analogy.]
## Similarity

<table>
<thead>
<tr>
<th>Method</th>
<th>WordSim Similarity</th>
<th>WordSim Relatedness</th>
<th>Bruni et al. MEN</th>
<th>Radinsky et al. M. Turk</th>
<th>Luong et al. Rare Words</th>
<th>Hill et al. SimLex</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPMI</td>
<td>.755</td>
<td>.697</td>
<td>.745</td>
<td>.686</td>
<td>.462</td>
<td>.393</td>
</tr>
<tr>
<td>SVD</td>
<td>.793</td>
<td>.691</td>
<td>.778</td>
<td>.666</td>
<td>.514</td>
<td>.432</td>
</tr>
<tr>
<td>SGNS</td>
<td>.793</td>
<td>.685</td>
<td>.774</td>
<td>.693</td>
<td>.470</td>
<td>.438</td>
</tr>
<tr>
<td>GloVe</td>
<td>.725</td>
<td>.604</td>
<td>.729</td>
<td>.632</td>
<td>.403</td>
<td>.398</td>
</tr>
</tbody>
</table>

- SVD = singular value decomposition on PMI matrix
- GloVe does not appear to be the best when experiments are carefully controlled, but it depends on hyperparameters + these distinctions don’t matter in practice

Levy et al. (2015)
Analogies

(king - man) + woman = queen

king + (woman - man) = queen

‣ Why would this be?

‣ woman - man captures the difference in the contexts that these occur in

‣ Dominant change: more “he” with man and “she” with woman — similar to difference between king and queen

‣ Can evaluate on this as well
What can go wrong with word embeddings?

- What’s wrong with learning a word’s “meaning” from its usage?
- What data are we learning from?
- What are we going to learn from this data?
What do we mean by bias?

- Identify *she - he* axis in word vector space, project words onto this axis

- Nearest neighbor of \((b - a + c)\)

---

**Bolukbasi et al. (2016)**

<table>
<thead>
<tr>
<th>Extreme <em>she</em> occupations</th>
<th>Extreme <em>he</em> occupations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. homemaker</td>
<td>2. nurse</td>
</tr>
<tr>
<td>2. nurse</td>
<td>3. receptionist</td>
</tr>
<tr>
<td>4. librarian</td>
<td>5. socialite</td>
</tr>
<tr>
<td>5. socialite</td>
<td>6. hairdresser</td>
</tr>
<tr>
<td>7. nanny</td>
<td>8. bookkeeper</td>
</tr>
<tr>
<td>8. bookkeeper</td>
<td>9. stylist</td>
</tr>
<tr>
<td>10. housekeeper</td>
<td>11. interior designer</td>
</tr>
<tr>
<td></td>
<td>12. guidance counselor</td>
</tr>
</tbody>
</table>

---

**Racial Analogies**

| black → homeless        | caucasian → servicemen |
| caucasian → hillbilly   | asian → suburban       |
| asian → laborer         | black → landowner      |

**Religious Analogies**

| jew → greedy            | muslim → powerless     |
| christian → familial    | muslim → warzone       |
| muslim → uneducated     | christian → intellectually |
Debiasing

- Identify gender subspace with gendered words
- Project words onto this subspace
- Subtract those projections from the original word

Bolukbasi et al. (2016)
Hardness of Debiasing

- Not that effective...and the male and female words are still clustered together.

- Bias pervades the word embedding space and isn’t just a local property of a few words.

(a) The plots for HARD-DEBIASED embedding, before (top) and after (bottom) debiasing.

Gonen and Goldberg (2019)
Takeaways

- Lots to tune with neural networks
  - Training: optimizer, initializer, regularization (dropout), ...
  - Hyperparameters: dimensionality of word embeddings, layers, ...
- Word vectors: learning word -> context mappings has given way to matrix factorization approaches (constant in dataset size)
- Lots of pretrained embeddings work well in practice, they capture some desirable properties
- Even better: context-sensitive word embeddings (ELMo)
- Next time: RNNs and CNNs