Recall: Feedforward NNs

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

- \( f(x) \) to \( V \)
- \( d \times n \) matrix
- \( n \) features
- \( V \) to \( g \)
- Nonlinearity: tanh, relu, ...
- \( g \) to \( z \)
- \( d \) hidden units
- \( z \) to \( W \)
- \( num \_classes \times d \) matrix
- \( W \) to softmax
- softmax to \( P(y|x) \)

Recall: Backpropagation

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

- \( f(x) \) to \( V \)
- \( d \times n \) matrix
- \( n \) features
- \( V \) to \( g \)
- Nonlinearity: tanh, relu, ...
- \( g \) to \( z \)
- \( d \) hidden units
- \( z \) to \( W \)
- \( num \_classes \times d \) matrix
- \( W \) to softmax
- softmax to \( P(y|x) \)

\[ \frac{\partial}{\partial W} \text{err}(\text{root}) \]

This Lecture

- Training
- Word representations
- word2vec
- Evaluating word embeddings
Training Basics

- Basic formula: compute gradients on batch, use first-order opt. method
- 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?

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

- 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

- How do we initialize V and W? What consequences does this have?
- Nonconvex problem, so initialization matters!
### 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
- 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)

### Optimizer

- Adam (Kingma and Ba, ICLR 2015) is very widely used
- Adaptive step size like Adagrad, incorporates momentum
  
  ![IMDB BoW Feature Logistic Regression](image1)
  ![MNIST Logistic Regression](image2)

- Wilson et al. NIPS 2017: adaptive methods can actually perform badly at test time (Adam is in pink, SGD in black)
- Check dev set periodically, decrease learning rate if not making progress
  
  ![Training Progress](image3)
  ![Development Progress](image4)
Structured Prediction

- Four elements of a machine learning method:
  - Model: feedforward, RNNs, CNNs can be defined in a uniform framework
  - Objective: many loss functions look similar, just changes the last layer of the neural network
  - Inference: define the network, your library of choice takes care of it (mostly...)
  - Training: lots of choices for optimization/hyperparameters

Word Representations

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

Discrete Word Representations

- Brown clusters: hierarchical agglomerative hard clustering (each word has one cluster, not some posterior distribution like in mixture models)
- Maximize $P(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)$
- Useful features for tasks like NER, not suitable for NNs Brown et al. (1992)
**Word Embeddings**

- Part-of-speech tagging with FFNNs
  
  \[ f(x) \]

- Fed raises interest rates in order to ...
  
  previous word
  
  curr word
  
  next word
  
  other words, feats, etc.

- What properties should these vectors have?

- Want a vector space where similar words have similar embeddings

  \[
  \text{the movie was great} \approx \text{the movie was good}
  \]

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

**Continuous Bag-of-Words**

- Predict word from context

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

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

**word2vec**

- Mikolov et al. (2013)
Skip-Gram

- Predict one word of context from word
  \[ \text{gold} = \text{dog} \]
  \[ P(w'|w) = \text{softmax}(We(w)) \]

- Another training example: \textit{bit} -> \textit{the}
- Parameters: \( d \times |V| \) vectors, \( |V| \times d \) output parameters (W) (also usable as vectors!)

Hierarchical Softmax

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

- Matmul + softmax over \( |V| \) is very slow to compute for CBOW and SG
- Huffman encode vocabulary, use binary classifiers to decide which branch to take
- \( \log(|V|) \) binary decisions

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
  \( (\textit{bit}, \textit{the}) \Rightarrow +1 \)
  \( (\textit{bit}, \textit{cat}) \Rightarrow -1 \)
  \( (\textit{bit}, \textit{a}) \Rightarrow -1 \)
  \( (\textit{bit}, \textit{fish}) \Rightarrow -1 \)

- Parameters: \( d \times |V| \) vectors, \( d \times |V| \) context vectors (same # of params as before)
- Objective = \( \log P(y = 1|w, c) - \frac{1}{k} \sum_{i=1}^{n} \log P(y = 0|w_i, c) \)

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?
Skip-Gram as Matrix Factorization

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

PMI\((w_i, c_j) = \frac{P(w_i, c_j)}{P(w_i)P(c_j)} = \frac{\text{count}(w_i, c_j)}{\sum_{i} \text{count}(w_i)} \frac{\text{count}(c_j)}{\sum_{j} \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

- Also operates on counts matrix, weighted regression on the log co-occurrence matrix (weights \(f\))
- Objective = \[ \sum_{i,j} f(\text{count}(w_i, c_j)) \left( w_i^T c_j + a_i + b_j - \log \text{count}(w_i, c_j) \right)^2 \]
- 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)

Preview: 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)

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 ???

### 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><strong>.793</strong></td>
<td>.691</td>
<td><strong>.778</strong></td>
<td>.666</td>
<td><strong>.514</strong></td>
<td>.432</td>
</tr>
<tr>
<td>SGNS</td>
<td>.793</td>
<td>.685</td>
<td>.774</td>
<td><strong>.693</strong></td>
<td>.470</td>
<td><strong>.438</strong></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

---

### Hypernymy Detection

- Hypernyms: detective *is a* person, dog *is a* animal
- Do word vectors encode these relationships?

<table>
<thead>
<tr>
<th>Dataset</th>
<th>TM14</th>
<th>Kotlerman 2010</th>
<th>HypeNet</th>
<th>WordNet</th>
<th>Avg (10 datasets)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>52.0</td>
<td>30.8</td>
<td>24.5</td>
<td>55.2</td>
<td>23.2</td>
</tr>
<tr>
<td>Word2Vec + C</td>
<td>52.1</td>
<td><strong>39.5</strong></td>
<td>20.7</td>
<td><strong>63.0</strong></td>
<td>25.3</td>
</tr>
<tr>
<td>GE + C</td>
<td>53.9</td>
<td>36.0</td>
<td>21.6</td>
<td>58.2</td>
<td>26.1</td>
</tr>
<tr>
<td>GE + KL</td>
<td>52.0</td>
<td>39.4</td>
<td>23.7</td>
<td>54.4</td>
<td>25.9</td>
</tr>
<tr>
<td>DIVE + C: ΔS</td>
<td><strong>57.2</strong></td>
<td>36.6</td>
<td><strong>32.0</strong></td>
<td>60.9</td>
<td><strong>32.7</strong></td>
</tr>
</tbody>
</table>

- word2vec (SGNS) works barely better than random guessing here

---

### 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
Analogies

<table>
<thead>
<tr>
<th>Method</th>
<th>Google</th>
<th>MSR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Add / Mul</td>
<td>Add / Mul</td>
</tr>
<tr>
<td>PPMI</td>
<td>.553 / .679</td>
<td>.306 / .535</td>
</tr>
<tr>
<td>SVD</td>
<td>.554 / .591</td>
<td>.408 / .468</td>
</tr>
<tr>
<td>SGNS</td>
<td>.676 / .688</td>
<td>.618 / .645</td>
</tr>
<tr>
<td>GloVe</td>
<td>.569 / .596</td>
<td>.533 / .580</td>
</tr>
</tbody>
</table>

- These methods can perform well on analogies on two different datasets using two different methods

Maximizing for $b$:

\[
\text{Add} = \cos(b, a_2 - a_1 + b_1) \quad \text{Mul} = \frac{\cos(b_2, a_2) \cos(b_2, b_1)}{\cos(b_2, a_1) + \epsilon}
\]

Levy et al. (2015)

Using Semantic Knowledge

- Structure derived from a resource like WordNet
- Doesn’t help most problems

Faruqui et al. (2015)

Using Word Embeddings

- Approach 1: learn embeddings as parameters from your data
  - Often works pretty well
- Approach 2: initialize using GloVe/ELMo, keep fixed
  - Faster because no need to update these parameters
- Approach 3: initialize using GloVe, fine-tune
  - Works best for some tasks, but not used for ELMo

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
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