Lecture 7: Tricks + Word Embeddings

Alan Ritter

(many slides from Greg Durrett)
Recall: Feedforward NNs

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

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

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

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

\[ f(x) \]

\[ V \]

def \(d \times n\) matrix

\(n\) features
Recall: Feedforward NNs

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

- **n features**: \( f(x) \) entering the network
- **d x n matrix**: \( V \) transforming the features
- **nonlinearity**: \( g \) applying tanh, relu, etc.
Recall: Feedforward NNs

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

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

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

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

\[ n \] features

\[ d \times n \text{ matrix} \]

\( f(x) \)

\[ d \text{ hidden units} \]

\[ g \]

\[ z \]

\[ W \]

\[ \text{softmax} \]

\[ P(y|x) \]

\[ \text{num_classes} \]

\[ \text{probs} \]

\[ \text{num_classes x d matrix} \]

\( \text{tanh, relu, ...} \)
Recall: Backpropagation

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

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

- Training
- Word representations
- word2vec/GloVe
- Evaluating word embeddings
Training Tips
Training Basics

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

- Basic formula: compute gradients on batch, use first-order opt. method
- How to initialize? How to regularize? What optimizer to use?
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?

\[ P(y|x) = \text{softmax}(W g(V f(x))) \]
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?
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?
How does initialization affect learning?

- Nonlinear model...how does this affect things?
How does initialization affect learning?

- Nonlinear model...how does this affect things?
How does initialization affect learning?

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

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

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

- Xavier 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
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
- Xavier 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.

Srivastava et al. (2014)
 Dropout

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

- Form of stochastic regularization

Srivastava et al. (2014)
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

Srivastava et al. (2014)
**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)
Optimizer

- Adam (Kingma and Ba, ICLR 2015) is very widely used
- Adaptive step size like Adagrad, incorporates momentum
Adam (Kingma and Ba, ICLR 2015) is very widely used

Adaptive step size like Adagrad, incorporates momentum
Wilson et al. NIPS 2017: adaptive methods can actually perform badly at test time (Adam is in pink, SGD in black)
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
Structured Prediction

- Four elements of a machine learning method:
Structured Prediction

- Four elements of a machine learning method:
- Model: feedforward, RNNs, CNNs can be defined in a uniform framework
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
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...)
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
Word Representations

- Neural networks work very well at continuous data, but words are discrete.
Word Representations

- Neural networks work very well at continuous data, but words are discrete.
- Continuous model <-> expects continuous semantics from input.

slide credit: Dan Klein
Word Representations

- Neural networks work very well at continuous data, but words are discrete.
- Continuous model $\Longleftrightarrow$ 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 *hard* clustering (each word has one cluster, not some posterior distribution like in mixture models)

Brown et al. (1992)
Discrete Word Representations

- Brown clusters: hierarchical agglomerative *hard* clustering (each word has one cluster, not some posterior distribution like in mixture models)

```
   0   1
```

Brown et al. (1992)
Brown clusters: hierarchical agglomerative *hard* clustering (each word has one cluster, not some posterior distribution like in mixture models)

Brown et al. (1992)
Discrete Word Representations

- Brown clusters: hierarchical agglomerative *hard* clustering (each word has one cluster, not some posterior distribution like in mixture models)

Brown et al. (1992)
Discrete Word Representations

- Brown clusters: hierarchical agglomerative *hard* clustering (each word has one cluster, not some posterior distribution like in mixture models)

![Tree diagram with nodes labeled as follows: 0, 1, cat, fish, is, go, great, enjoyable, dog, ...]

- Maximize $P(w_i | w_{i-1}) = P(c_i | c_{i-1}) P(w_i | c_i)$

Brown et al. (1992)
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
  \[ f(x) \]
  \[ \text{emb}(\text{raises}) \]
  \[ \text{emb}(	ext{interest}) \]
  \[ \text{emb}(	ext{rates}) \]

- Word embeddings for each word form input

Fed raises *interest* rates in order to ...

prev word

curr word

next word

other words, feats, etc.

Botha et al. (2017)
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

dog
bad

great
good
enjoyable

is
Word Embeddings

- Want a vector space where similar words have similar embeddings
Want a vector space where similar words have similar embeddings

- the movie was great
- the movie was good
Word Embeddings

- Want a vector space where similar words have similar embeddings

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

- Goal: come up with a way to produce these embeddings
word2vec/GloVe
Continuous Bag-of-Words

- Predict word from context

the\textcolor{blue}{dog} bit the\textcolor{blue}{man}

Mikolov et al. (2013)
Continuous Bag-of-Words

- Predict word from context

  $d$-dimensional word embeddings

Mikolov et al. (2013)
Continuous Bag-of-Words

- Predict word from context

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

- d-dimensional word embeddings

Mikolov et al. (2013)
Continuous Bag-of-Words

- Predict word from context

\[
\text{dog} \quad \text{Multiplying by } W \quad \text{softmax}
\]

\[
d\text{-dimensional word embeddings}
\]

\[
\text{the} \quad + \quad \text{size } d \quad \text{size } |V| \times d
\]

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

Mikolov et al. (2013)
Continuous Bag-of-Words

- Predict word from context

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

- \( d \)-dimensional word embeddings

- Multiply by \( W \)

- Softmax

\( d \)-dimensional word embeddings

Mikolov et al. (2013)
Continuous Bag-of-Words

- Predict word from context

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

\( d \)-dimensional word embeddings

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

\[ \text{gold label} = \text{bit}, \text{ no manual labeling required!} \]

Mikolov et al. (2013)
Continuous Bag-of-Words

- Predict word from context

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

Mikolov et al. (2013)
Continuous Bag-of-Words

- Predict word from context

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

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

Mikolov et al. (2013)
Skip-Gram

- Predict one word of context from word

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

Mikolov et al. (2013)
Skip-Gram

- Predict one word of context from word

$\text{the dog bit the man}$

$\text{gold} = \text{dog}$

Mikolov et al. (2013)
Skip-Gram

- Predict one word of context from word

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

Gold = dog

Mikolov et al. (2013)
Skip-Gram

- Predict one word of context from word

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

- Another training example: \( \text{bit} \rightarrow \text{the} \)

\[ \text{gold} = \text{dog} \]
Skip-Gram

- Predict one word of context from word
  - The dog bit the man

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

- Another training example: bit -> the

- $P(w'|w) = \text{softmax}(W e(w))$

Mikolov et al. (2013)
Hierarchical Softmax

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

Mikolov et al. (2013)
Hierarchical Softmax

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

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

Mikolov et al. (2013)
Hierarchical Softmax

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

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

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

Mikolov et al. (2013)
Hierarchical Softmax

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

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

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

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

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

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

Mikolov et al. (2013)
Hierarchical Softmax

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

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

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

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:
  \[ [|V| \times d] \times d \]

Mikolov et al. (2013)
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

- Standard softmax:
  \[
  \text{[[}|V| \times d]| \times d
  \]
Hierarchical Softmax

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

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

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

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

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

Mikolov et al. (2013)
Hierarchical Softmax

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

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

\[ P(w|w_{0}) = \text{softmax}(W e(w)) \]

- Huffman encode vocabulary, use binary classifiers to decide which branch to take
- \(\log(|V|)\) binary decisions

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

- Hierarchical softmax:
Hierarchical Softmax

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

- 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

- Standard softmax: \([|V| \times d] \times d\)
- Hierarchical softmax: \(\log(|V|)\) dot products of size \(d\)

Mikolov et al. (2013)
Hierarchical Softmax

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

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

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

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

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

- \(\text{log}(|V|)\) binary decisions

---

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

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.
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.

  $(bit, the) \Rightarrow +1$

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

  \[(\text{bit, the}) \Rightarrow +1\]

  \[(\text{bit, cat}) \Rightarrow -1\]
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.

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

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

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

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.

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

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

- Words in similar contexts select for similar \(c\) vectors.

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.

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

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

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

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:
  - $(bit, the) \Rightarrow +1$
  - $(bit, cat) \Rightarrow -1$
  - $(bit, a) \Rightarrow -1$
  - $(bit, fish) \Rightarrow -1$

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

- Words in similar contexts select for similar $c$ vectors

- $d \times |V|$ 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)
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.
  - \((bit, the) \Rightarrow +1\)
  - \((bit, cat) \Rightarrow -1\)
  - \((bit, a) \Rightarrow -1\)
  - \((bit, fish) \Rightarrow -1\)

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

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

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

words in similar contexts select for similar \(c\) vectors

Mikolov et al. (2013)
Connections with Matrix Factorization

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

Levy et al. (2014)
Connections with Matrix Factorization

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

```
|V|  |V|
|---|---|
|   | word pair counts |
```
Connections with Matrix Factorization

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

![Diagram showing connections between word pair counts and word vectors.](image-url)
Skip-gram model looks at word-word co-occurrences and produces two types of vectors.

- **Word pair counts**
- **Context vectors**
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

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

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

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

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

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

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

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

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

Skip-gram objective exactly corresponds to factoring this matrix:

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

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

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

Skip-gram objective exactly corresponds to factoring this matrix:

- If we sample negative examples from the uniform distribution over words

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

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

PMI\((w_i, c_j)\) = \(\frac{P(w_i, c_j)}{P(w_i) P(c_j)} = \frac{\text{count}(w_i, c_j)}{D_{\text{count}(w_i)} D_{\text{count}(c_j)}}\)

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

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

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

Loss = \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

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

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

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

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

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

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

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

- 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)
How to handle different word senses? One vector for *balls*

- they dance at balls
- they hit the balls

Peters et al. (2018)
How to handle different word senses? One vector for *balls*

they dance at balls they hit the balls

Train a neural language model to predict the next word given previous words in the sentence, use its internal representations as word vectors
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.

Peters et al. (2018)
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

Peters et al. (2018)
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

Peters et al. (2018)
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?
Evaluating Word Embeddings

- What properties of language should word embeddings capture?
- Similarity: similar words are close to each other
Evaluating Word Embeddings

- What properties of language should word embeddings capture?
- Similarity: similar words are close to each other
- Analogy:
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 ???

Diagram:
- cat
- dog
- tiger
- wolf
- bad
- was
- is
- great
- enjoyable
- good
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><strong>.793</strong></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

Levy et al. (2015)
This page contains a table comparing various methods for word similarity, along with a brief commentary on their performance and limitations. The table is titled "Similarity" and includes a breakdown of performance across different metrics and datasets.

<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>0.755</td>
<td>0.697</td>
<td>0.745</td>
<td>0.686</td>
<td>0.462</td>
<td>0.393</td>
</tr>
<tr>
<td>SVD</td>
<td><strong>0.793</strong></td>
<td>0.691</td>
<td><strong>0.778</strong></td>
<td>0.666</td>
<td><strong>0.514</strong></td>
<td>0.432</td>
</tr>
<tr>
<td>SGNS</td>
<td><strong>0.793</strong></td>
<td>0.685</td>
<td>0.774</td>
<td><strong>0.693</strong></td>
<td>0.470</td>
<td><strong>0.438</strong></td>
</tr>
<tr>
<td>GloVe</td>
<td>0.725</td>
<td>0.604</td>
<td>0.729</td>
<td>0.632</td>
<td>0.403</td>
<td>0.398</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

Chang et al. (2017)
Hypernymy Detection

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

Chang et al. (2017)
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>

Chang et al. (2017)
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

Chang et al. (2017)
Analogies

king
queen
Analogies

king
queen
man
woman
Analogies

\[(king \ - \ man) \ + \ woman = queen\]
Analogies

\[(\text{king} - \text{man}) + \text{woman} = \text{queen}\]
Analogies

\[(\text{king} - \text{man}) + \text{woman} = \text{queen}\]
Analogies

\[(\text{king} - \text{man}) + \text{woman} = \text{queen}\]

\[\text{king} + (\text{woman} - \text{man}) = \text{queen}\]
Analogies

(king - man) + woman = queen

king + (woman - man) = queen

- Why would this be?
Analogies

(king - man) + woman = queen

king + (woman - man) = queen

- Why would this be?
Analogies

\[(\text{king} - \text{man}) + \text{woman} = \text{queen}\]

\[\text{king} + (\text{woman} - \text{man}) = \text{queen}\]

- Why would this be?
- woman - man captures the difference in the contexts that these occur in
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

LEVY ET AL. (2015)

<table>
<thead>
<tr>
<th>Method</th>
<th>Google Add / Mul</th>
<th>MSR Add / Mul</th>
</tr>
</thead>
<tbody>
<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 / <strong>.688</strong></td>
<td>.618 / <strong>.645</strong></td>
</tr>
<tr>
<td>GloVe</td>
<td>.569 / .596</td>
<td>.533 / .580</td>
</tr>
</tbody>
</table>

LEVY ET AL. (2015)
Analogies

<table>
<thead>
<tr>
<th>Method</th>
<th>Google Add / Mul</th>
<th>MSR Add / Mul</th>
</tr>
</thead>
<tbody>
<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.

Levy et al. (2015)
These methods can perform well on analogies on two different datasets using two different methods.

Maximizing for $b$: Add $= \cos(b, a_2 - a_1 + b_1)$, 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

Faruqui et al. (2015)
Using Semantic Knowledge

- Structure derived from a resource like WordNet

Faruqui et al. (2015)
Using Semantic Knowledge

Structure derived from a resource like WordNet

Original vector for false

Adapted vector for false

Faruqui 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
Using Word Embeddings

- Approach 1: learn embeddings as parameters from your data
  - Often works pretty well
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
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?
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)
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?
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
Takeaways

- Lots to tune with neural networks
  - Training: optimizer, initializer, regularization (dropout), ...
Takeaways

- Lots to tune with neural networks
  - Training: optimizer, initializer, regularization (dropout), ...
  - Hyperparameters: dimensionality of word embeddings, layers, ...

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)
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
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)
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