W4995 Applied Machine Learning

Word Embeddings

04/10/19

Andreas C. Müller

Beyond Bags of Words

Limitations of bag of words:

- Semantics of words not captured
- Synonymous words not represented
- Very distributed representation of documents

Last Time

- Latent Semantic Analysis
- Non-negative Matrix Factorization
- Latent Dirichlet Allocation
- All embed documents into a continuous, corpus specific space.
- Today: Embed words in a “general” space (mostly).

Idea

- Unsupervised extraction of semantics using large corpus (wikipedia etc)
- Input: one-hot representation of word (as in BoW).
- Use auxiliary task to learn continuous representation.

Skip-Gram models

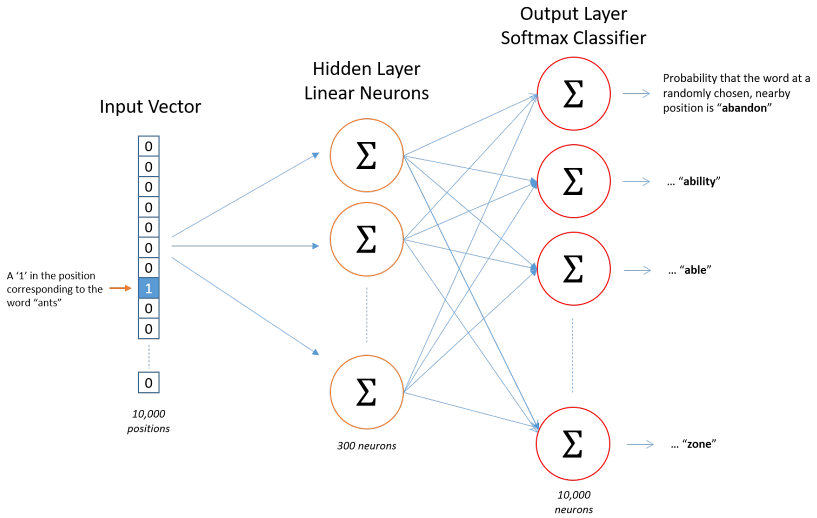
- Given a word, predict surrounding word
- Supervised task, each document yields many examples
- Not interested in performance for this task, just want to learn representations.

Example

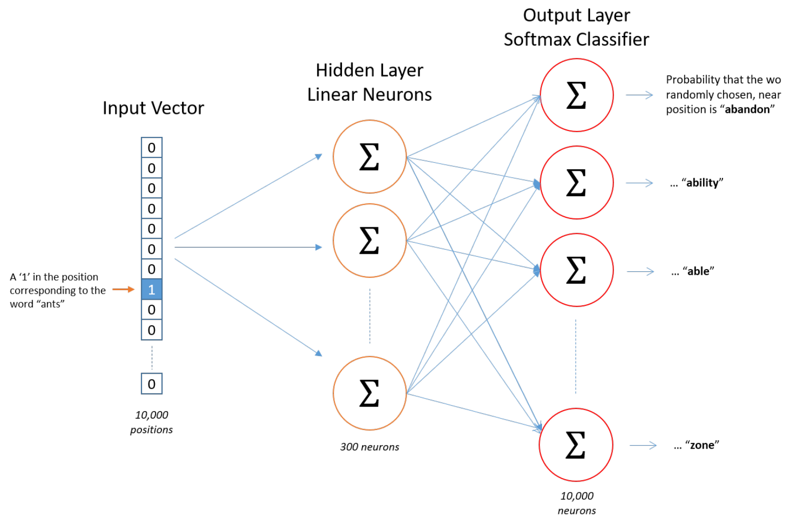
```
[“What is my purpose?”, “You pass the butter.”]
```

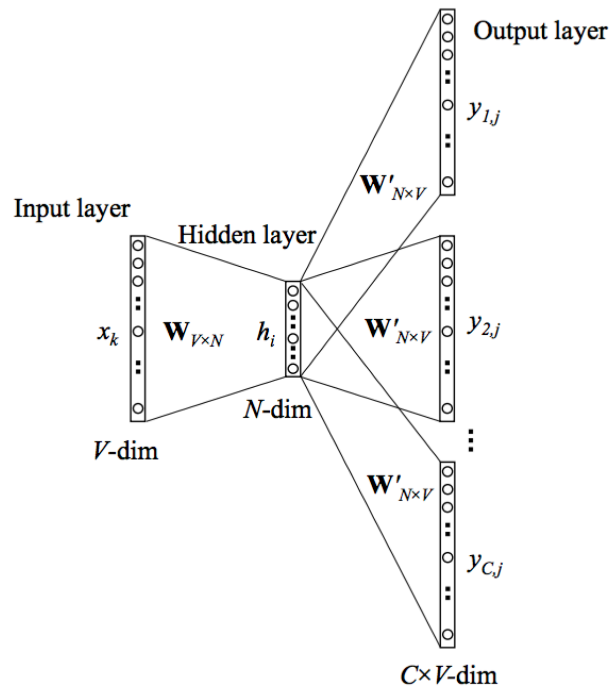
word	context
“is”	[“what”, “my”]
“my”	[“is”, “purpose”]
“pass”	[“you”, “the”]
“the”	[“pass”, “butter”]

Using context windows of size 1 (in practice 5 or 10):



http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/





http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

8 / 48

Softmax Training

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c,j\neq 0}\log p(w_{t+j}|w_t)$$

Output weights

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^\top v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^\top v_{w_I}\right)}$$

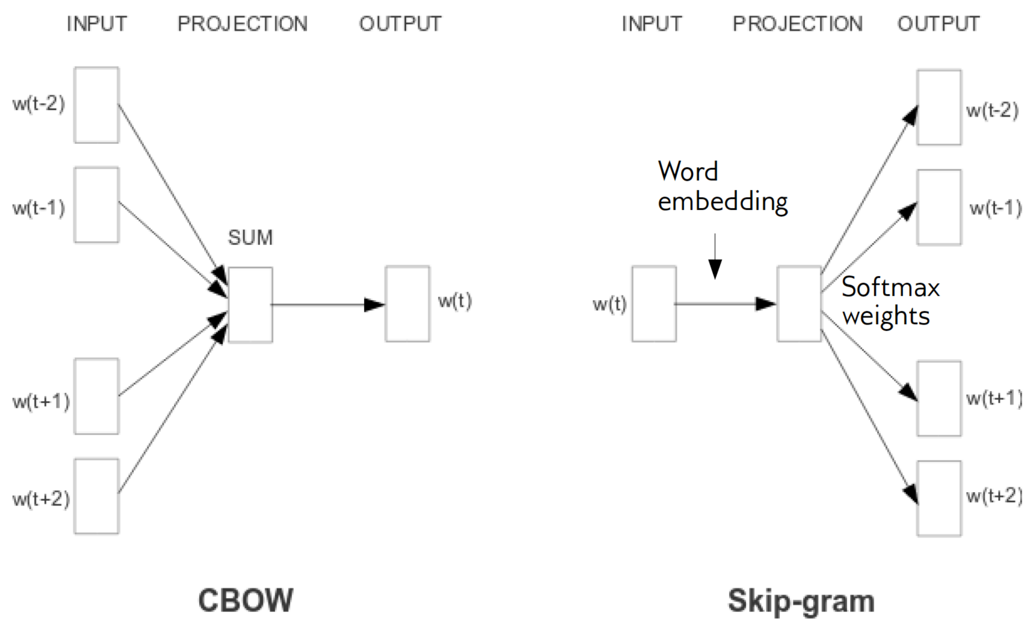
Word embedding

Normalize over whole vocabulary!
[We want to do stochastic gradient descent / minibatch learning]
Monte-Carlo estimate: use some "noise words"

[Mikolov et. al. - Distributed Representations of Words and Phrases and their Compositionality (2013)]

9 / 48

CBOW vs Skip-gram



Efficient Estimation of Word Representations in Vector Space https://arxiv.org/pdf/1301.3781.pdf

Implementations

- Gensim
- Word2vec
- Tensorflow
- fasttext
- ...
- Don't train yourself

Gensim - topic models for humans

- Multiple Latent Dirichlet Allocation implementations
- Wrappers for Mallet and vopal wabbit
- Tools for analyzing topic models
- No supervised learning
- Uses list-of-tuples instead of sparse matrices to store documents.

Introduction to gensim

```
docs = ["What is my purpose", "You bring butter"]
texts = [[token for token in doc.lower().split()] for doc in docs]
print(texts)
```

```
[['what', 'is', 'my', 'purpose'], ['you', 'bring', 'butter']]
```

```
from gensim import corpora
dictionary = corpora.Dictionary(texts)
print(dictionary)
```

```
Dictionary(7 unique tokens: ['butter', 'you', 'is', 'purpose', 'my']...)
```

```
new_doc = "what butter"
dictionary.doc2bow(new_doc.lower().split())
```

```
[(3, 1), (5, 1)]
```

```
corpus = [dictionary.doc2bow(text) for text in texts]
corpus
```

```
[[(0, 1), (1, 1), (2, 1), (3, 1)], [(4, 1), (5, 1), (6, 1)]]
```

Converting to/from sparse matrix

```
import gensim
corpus
```

```
[[(0, 1), (1, 1), (2, 1), (3, 1)], [(4, 1), (5, 1), (6, 1)]]
```

```
gensim.matutils.corpus2csc(corpus)
```

```
<7x2 sparse matrix of type '<class 'numpy.float64'>'
  with 7 stored elements in Compressed Sparse Column format>
```

```
X = CountVectorizer().fit_transform(docs)
X
```

```
sparse_corpus = gensim.matutils.Sparse2Corpus(X.T)
print(sparse_corpus)
print(list(sparse_corpus))
```

```
<gensim.matutils.Sparse2Corpus object at 0x7fd6d776b438>
[[(4, 1), (3, 1), (2, 1), (5, 1)], [(1, 1), (0, 1), (6, 1)]]
```

Corpus Transformations

```
tfidf = gensim.models.TfidfModel(corpus)
tfidf[corpus[0]]
```

```
[(0, 0.5), (1, 0.5), (2, 0.5), (3, 0.5)]
```

```
print(tfidf[corpus])
print(list(tfidf[corpus]))
```

```
<gensim.interfaces.TransformedCorpus object at 0x7fd6d776b2b0>
[[(0, 0.5), (1, 0.5), (2, 0.5), (3, 0.5)], [(4, 0.577), (5, 0.577), (6, 0.577)]]
```

Word2Vec in Gensim

```
from gensim import models
w = models.KeyedVectors.load_word2vec_format(
    '../GoogleNews-vectors-negative300.bin', binary=True)
w['queen'].shape
```

```
(300,)
```

```
w.vectors.shape
```

```
(3000000, 300)
```

Cosine Similarity

$$\text{similarity}(v, w) = \cos(\theta) = \frac{v^T w}{\|v\| \|w\|}$$

Inspecting Semantics

```
w.most_similar(positive=['movie'], topn=5)
```

```
[('film', 0.867),
 ('movies', 0.801),
 ('films', 0.736),
 ('moive', 0.683),
 ('Movie', 0.669)]
```

```
w.most_similar(positive=['good'], topn=5)
```

```
w.most_similar(positive=['good'], topn=5)
```

```
[('great', 0.729),
 ('bad', 0.719),
 ('terrific', 0.688),
 ('decent', 0.683),
 ('nice', 0.683)]
```

```
w.most_similar(positive=['good'], topn=5)
```

```
[('great', 0.729),
 ('bad', 0.719),
 ('terrific', 0.688),
 ('decent', 0.683),
 ('nice', 0.683)]
```

```
w.most_similar(positive=['cute', 'dog'], topn=5)
```

```
[('puppy', 0.764),
 ('chihuahua', 0.720),
 ('adorable_puppy', 0.710),
 ('yorkie', 0.701),
 ('Shitzu', 0.700)]
```

Represent doc by average

```
X_train = np.vstack([np.mean(w[doc], axis=0) for doc in docs])
X_train.shape
```

```
(18750, 300)
```

```
docs_val = vect_w2v.inverse_transform(vect_w2v.transform(text_val))
X_val = np.vstack([np.mean(w[doc], axis=0) for doc in docs_val])
```

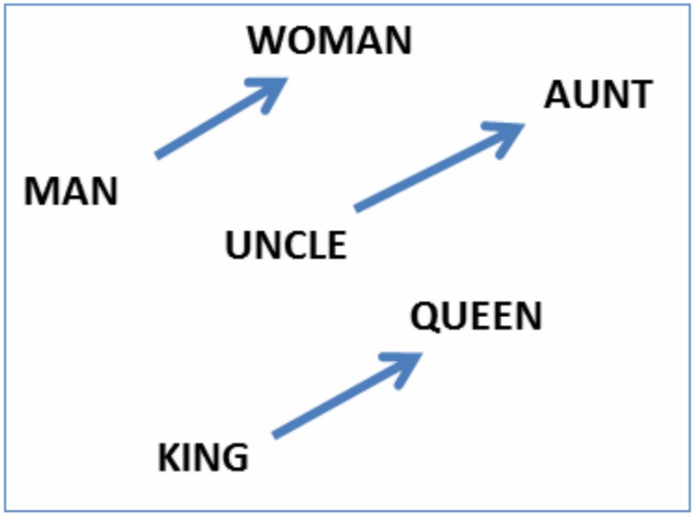
```
lr_w2v = LogisticRegression(C=100).fit(X_train, y_train_sub)
lr_w2v.score(X_train, y_train_sub)
```

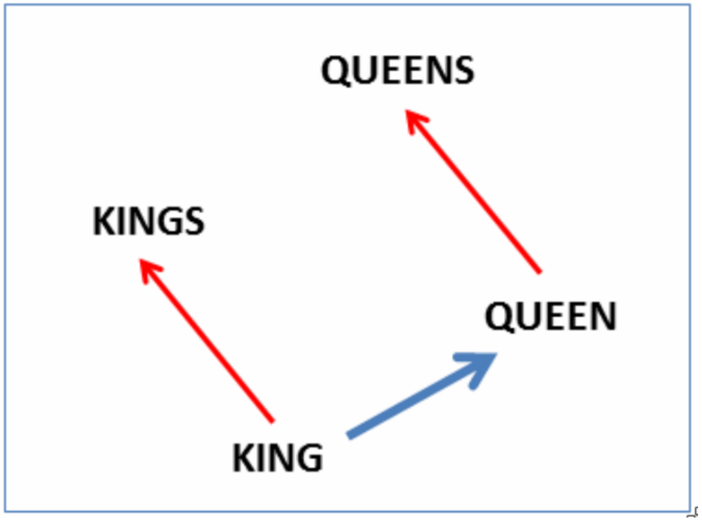
```
0.867
```

```
lr_w2v.score(X_val, y_val)
```

```
0.857
```

Analogues and Relationships

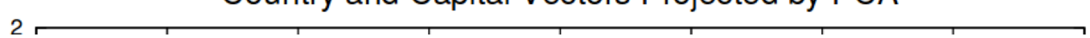




Answer “King is to Kings as Queen is to ?”:
Find closest vector to vec(“Queen”) + (vec(“Kings”) - vec(“King”))

[Mikolov et. al. Linguistic Regularities in Continuous Space Word Representations (2013)](https://amueller.github.io/COMS4995-s19/slides/aml-19-word-embeddings/#1)

Country and Capital Vectors Projected by PCA



Finding Relations

$$a : b :: c :?$$

$$d = \arg\max_i \frac{(\text{vec}(b) - \text{vec}(a) + \text{vec}(v))^T vec_i}{\|\text{vec}(b) - \text{vec}(a) + \text{vec}(v)\| \|\text{vec}_i\|}$$

Input	Result Produced
Chicago : Illinois : : Houston	Texas
Chicago : Illinois : : Philadelphia	Pennsylvania
Chicago : Illinois : : Phoenix	Arizona
Chicago : Illinois : : Dallas	Texas
Chicago : Illinois : : Jacksonville	Florida
Chicago : Illinois : : Indianapolis	Indiana
Chicago : Illinois : : Austin	Texas
Chicago : Illinois : : Detroit	Michigan
Chicago : Illinois : : Memphis	Tennessee
Chicago : Illinois : : Boston	Massachusetts

[Stanford CS 224D: Deep Learning for NLP]

Examples with Gensim

```
w.most_similar(positive=['woman', 'king'], negative=['man'], topn=3)
```

```
[('queen', 0.711),
 ('monarch', 0.618),
 ('princess', 0.590)]
```

```
w.most_similar(positive=['woman', 'he'], negative=['man'], topn=3)
```

```
[('she', 0.849),
 ('She', 0.632),
 ('her', 0.602)]
```

```
w.most_similar(positive=['Germany', 'pizza'], negative=['Italy'], topn=3)
```

```
[('bratwurst', 0.543),
 ('Domino_pizza', 0.513),
 ('donuts', 0.512)]
```

Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

Tolga Bolukbasi[1], Kai-Wei Chang[2], James Zou[2], Venkatesh Saligrama[1,2], Adam Kalai[2]
[1]Boston University, 8 Saint Mary's Street, Boston, MA
[2]Microsoft Research New England, 1 Memorial Drive, Cambridge, MA
tolgab@bu.edu, kw@kwchang.net, jamesyzou@gmail.com, srv@bu.edu, adam.kalai@microsoft.com

$$\overrightarrow{man} - \overrightarrow{woman} \approx \overrightarrow{king} - \overrightarrow{queen}$$

$$\overrightarrow{man} - \overrightarrow{woman} \approx \overrightarrow{computerprogrammer} - \overrightarrow{homemaker}$$

Going along he-she direction:

Gender stereotype *she*-*he* analogies.

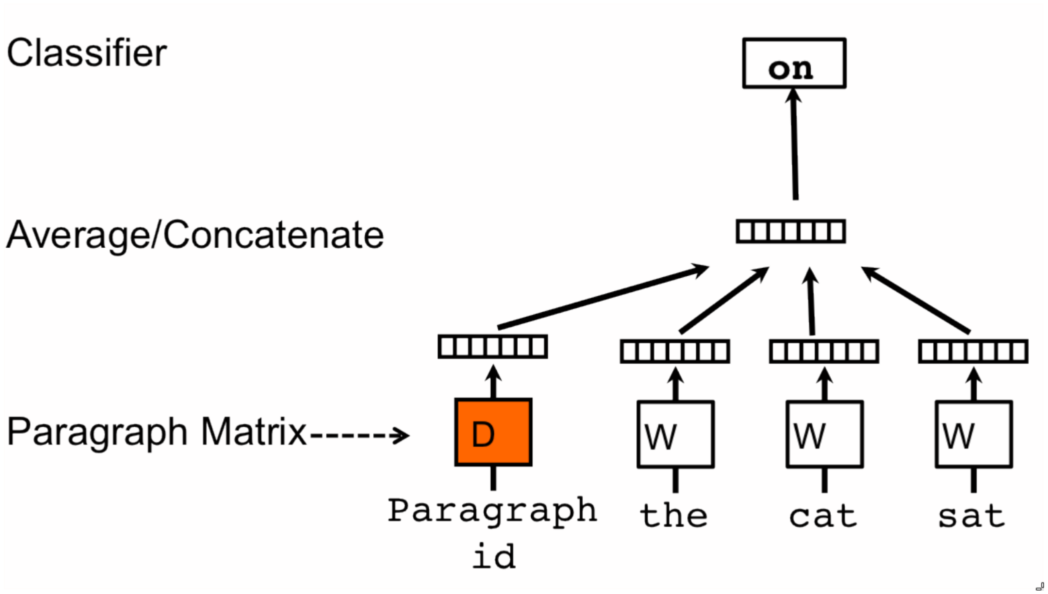
sewing-carpentry	register-nurse-physician	housewife-shopkeeper
nurse-surgeon	interior designer-architect	softball-baseball
blond-burly	feminism-conservatism	cosmetics-pharmaceuticals
giggle-chuckle	vocalist-guitarist	petite-lanky
sassy-snappy	diva-superstar	charming-affable
volleyball-football	cupcakes-pizzas	hairdresser-barber

Gender appropriate *she*-*he* analogies.

queen-king	sister-brother	mother-father
waitress-waiter	ovarian cancer-prostate cancer	convent-monastery

Paragaph Vectors

Doc2Vec



[Le, Mikolov: Distributed Representations of Sentences and Documents (2014)](https://amueller.github.io/COMS4995-s19/slides/aml-19-word-embeddings/#1)

Doc2Vec with gensim

```
def read_corpus(text, tokens_only=False):
    for i, line in enumerate(text):
        if tokens_only:
            yield gensim.utils.simple_preprocess(line)
        else:
            # For training data, add tags
            yield gensim.models.doc2vec.TaggedDocument(
                gensim.utils.simple_preprocess(line), [i])

train_corpus = list(read_corpus(text_train_sub))
test_corpus = list(read_corpus(text_val, tokens_only=True))

model = gensim.models.doc2vec.Doc2Vec(vector_size=50, min_count=2)
model.build_vocab(train_corpus)

model.train(train_corpus, total_examples=model.corpus_count, epochs=55)
```

https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/doc2vec-lee.ipynb

32 / 48

Validation of Word Vectors

```
model.wv.most_similar("movie")
```

```
[('film', 0.948),
 ('flick', 0.822),
 ('series', 0.715),
 ('programme', 0.703),
 ('sequel', 0.693),
 ('story', 0.677),
 ('show', 0.655),
 ('documentary', 0.653),
 ('picture', 0.642),
 ('thriller', 0.630)]
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=100).fit(X_train, y_train_sub)

lr.score(X_train, y_train_sub)
```

```
0.817
```

```
lr.score(X_val, y_val)
```

```
0.803
```

GloVe: Global Vectors for Word Representation

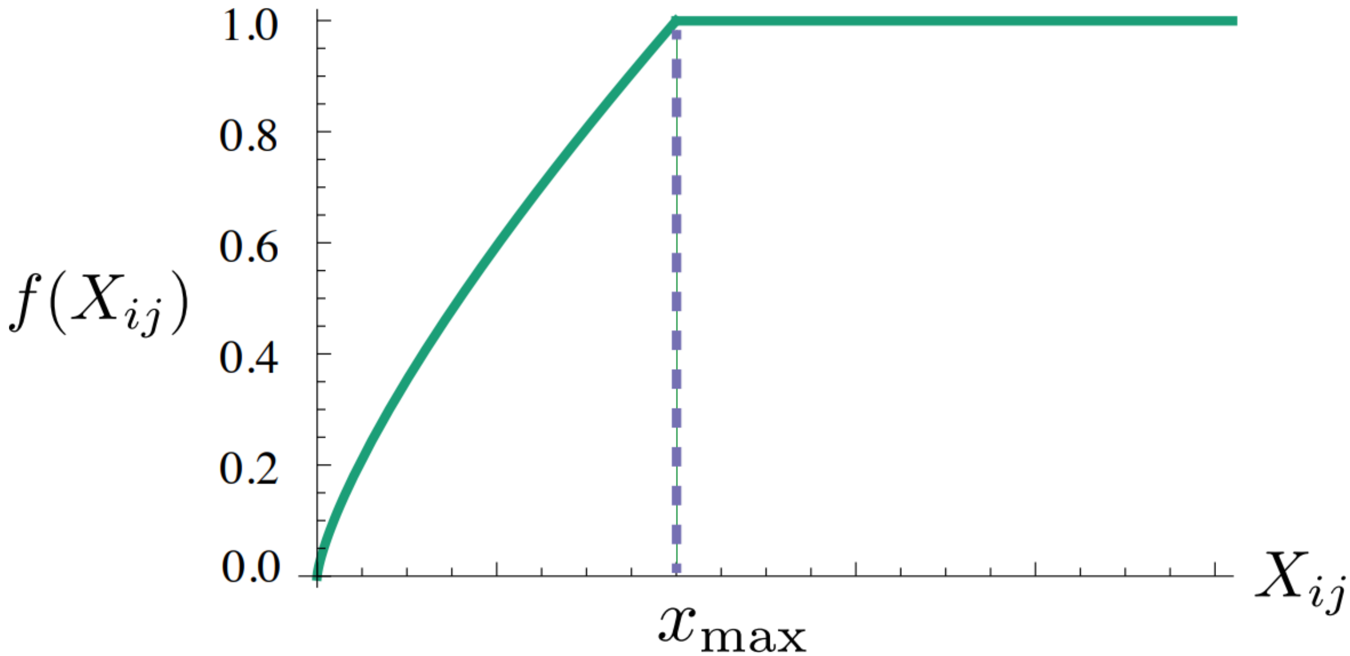
X_{ij} = How often does work j appear in context of word i

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

https://nlp.stanford.edu/projects/glove/

GloVe Weighting function f



36 / 48

Word analogies

Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2

37 / 48

(Stochastic) Gradient Descent

(see http://leon.bottou.org/projects/sgd and http://leon.bottou.org/papers/bottou-bousquet-2008 and http://scikit-learn.org/stable/modules/scaling_strategies.html)

Reminder: Gradient Descent

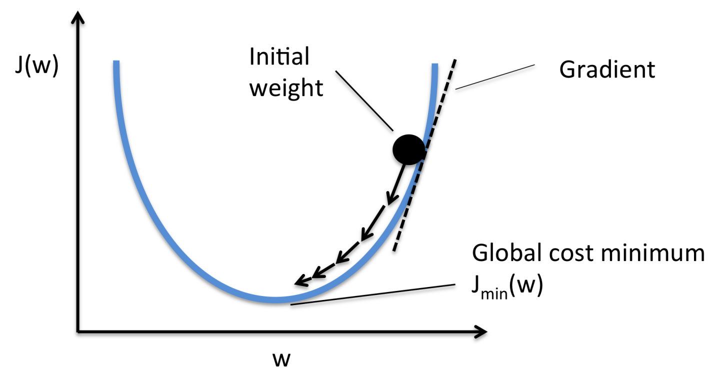
Want:

$$\arg\min_{w} F(w)$$

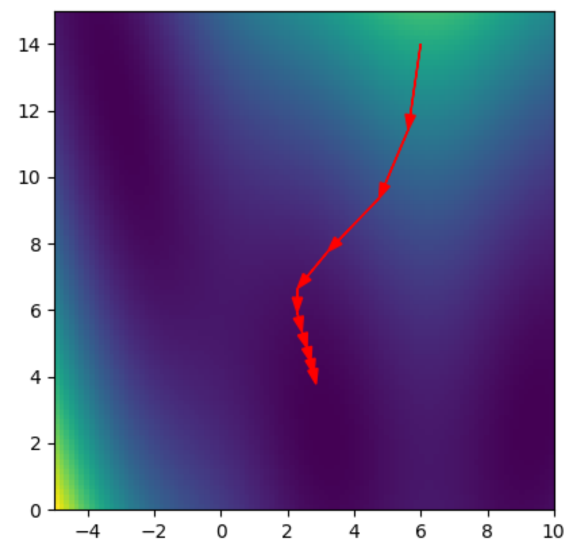
Initialize w_0

$$w^{(i+1)} \leftarrow w^{(i)} - \eta_i \frac{d}{dw} F(w^{(i)})$$

Converges to local minimum

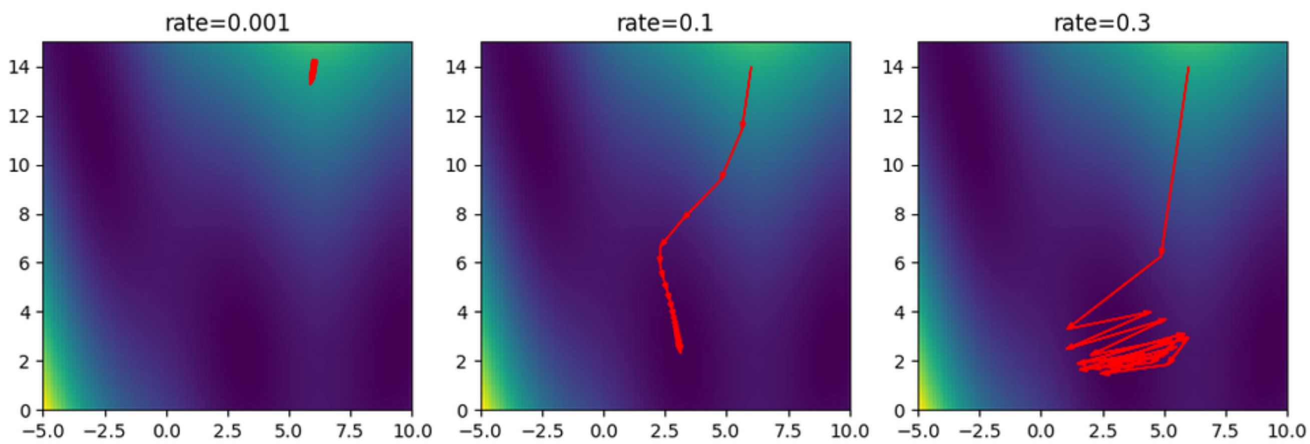


Reminder: Gradient Descent



$$w^{(i+1)} \leftarrow w^{(i)} - \eta_i \frac{d}{dw} F(w^{(i)})$$

Pick a learning rate



$$w^{(i+1)} \leftarrow w^{(i)} - \eta_i \frac{d}{dw} F(w^{(i)})$$

Batch vs stochastic optimization

Batch

$$W_i \leftarrow W_i - \eta \sum_{j=1}^{N} \frac{\partial l(x_j, y_j)}{\partial W_i}$$

Batch vs stochastic optimization

Batch

$$W_i \leftarrow W_i - \eta \sum_{j=1}^{N} \frac{\partial l(x_j, y_j)}{\partial W_i}$$

Online/Stochastic

$$W_i \leftarrow W_i - \eta \frac{\partial l(x_j, y_j)}{\partial W_i}$$

Batch vs stochastic optimization

Batch

$$W_i \leftarrow W_i - \eta \sum_{j=1}^{N} \frac{\partial l(x_j, y_j)}{\partial W_i}$$

Online/Stochastic

$$W_i \leftarrow W_i - \eta \frac{\partial l(x_j, y_j)}{\partial W_i}$$

Minibatch

$$W_i \leftarrow W_i - \eta \sum_{j=k}^{k+m} \frac{\partial l(x_j, y_j)}{\partial W_i}$$

Stochastic Gradient Descent

- Logistic Regression:

$$F(w) = -C \sum_{i=1}^{n} \log(\exp(-y_i w^T \mathbf{x}_i) + 1) + ||w||_2^2$$

- Pick x_i randomly, then

$$\frac{d}{dw}F(w) = \frac{d}{dw} - C \log(\exp(-y_i w^T \mathbf{x}_i) + 1) + \frac{1}{n}||w||_2^2$$

- In practice: just iterate over i.

SGD and partial_fit

- SGDClassifier(), SGDRegressor() fast on very large datasets
- Tuning learning rate and schedule can be tricky.
- partial_fit allows working with out-of-memory data!

```python
sgd = SGDClassifier()
for X_batch, y_batch in batches:
    sgd.partial_fit(X_batch, y_batch, classes=[0, 1, 2])
sgd.score(X_test, y_test)
```

```
0.815
```

```python
sgd = SGDClassifier()
for i in range(10):
    for X_batch, y_batch in batches:
        sgd.partial_fit(X_batch, y_batch, classes=[0, 1, 2])
sgd.score(X_test, y_test)
```

```
0.947
```

Questions ?