

# CS6120

Jiaji Huang

<https://jiaji-huang.github.io>

# About myself

- PhD in electrical engineering from Duke University
- Thesis: statistical signal processing and machine learning
- Worked at Baidu Research on NLP and Speech
- Now at AWS, on Large Language Models

# Announcement

- Discovery Cluster ready
- Check the 3<sup>rd</sup> link on syllabus

## Computers

1. Your laptop; will need to install python (and more)
2. Koury Cloud: [this](#) and [this](#)
3. [Discovery Cluster](#)
4. [GitHub](#) (please request an account)

# Motivating Task: Positive or Negative Review?



Roll over image to zoom in



Cyphis



**One Star is Too Much for This Product**

Reviewed in the United States on September 7, 2012

I don't know if this is a scam or if mine was broken, but it doesn't work and I am still getting abducted by UFO's on a regular basis.



Amazon Customer



**it works**

Reviewed in the United States on March 21, 2018

**Verified Purchase**

it works great



Marcie G.



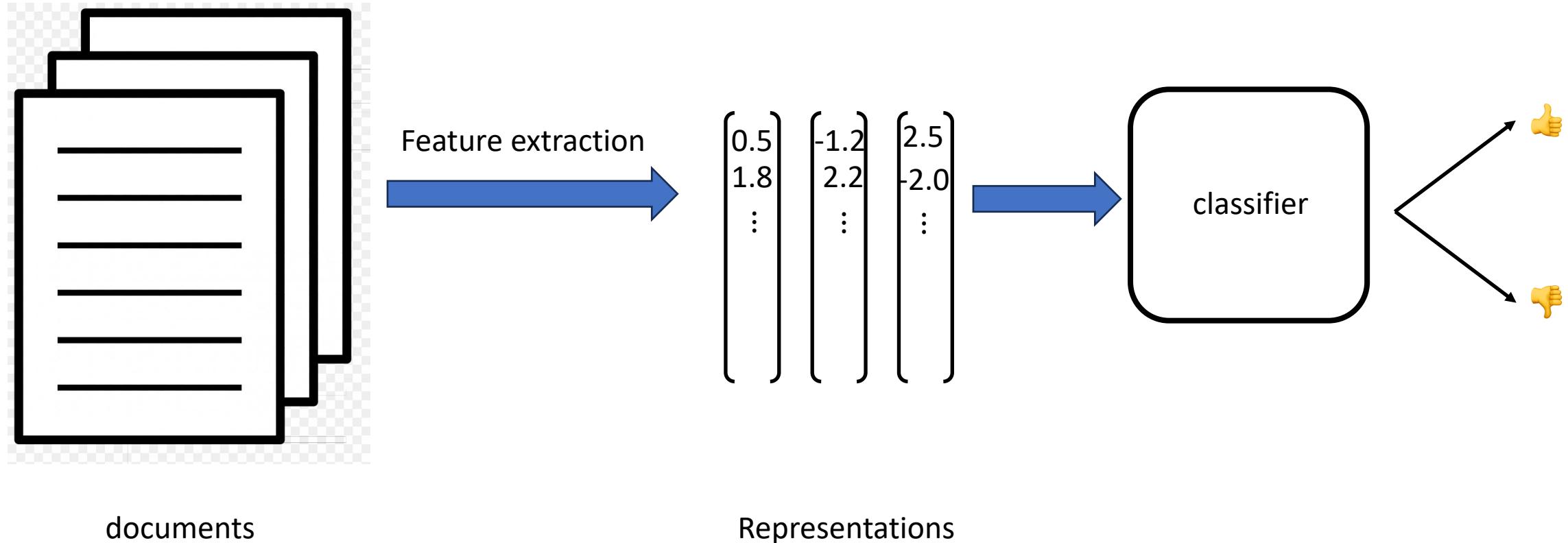
**Freedom!**

Reviewed in the United States on December 12, 2013

Knowing when "they" are in the area this allows me to take off my tin foil hat a little more frequently. I've even peeked out the foil drapes occasionally, but still not that often.

I gave it 4 stars as I believe it could alert you sooner. I was almost teleported during one of my peeking episodes.

# A Machine Learning Approach



# How to build the pipeline

- Classical:
  - Hand crafted features,
    - e.g., bag of words (NLP), sift (computer vision), MEL filter banks (speech)
  - Learn the classifier on training features
- Modern:
  - End-to-end  
Learn a (deep) network that extracts feature together with a classifier
- Cutting-edge
  - Pretrain + (supervised) finetune + RLHF + task specific adaptation

this lecture



# Roadmap

- Document representation (classical)
  - Bag of words
  - tf-idf
- Word representation (modern)
  - Word2vec and PMI
  - Applications
- Classifier
  - Naïve Bayesian classifier
  - Softmax classifier

# Roadmap

- Document representation (classical)
  - Bag of words
  - tf-idf
- Word representation (modern)
  - Word2vec and PMI
  - Applications
- Classifier
  - Naïve Bayesian classifier
  - Softmax classifier

# The Bag of Words Representation

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



# tf-idf

- Words have different importances, overlooked by simple count
- tf: term frequency (multiple ways to define)

$$tf_{t,d} = \frac{count(t, d)}{\sum_t count(t, d)}$$
$$tf_{t,d} = \log[1 + count(t, d)]$$

- idf: inverse document frequency

$$idf_t = \log\left(\frac{\# total\ docs}{\#docs\ that\ have\ term\ t}\right)$$

- tf-idf for word  $t$  in document  $d$ :  $tf_{t,d} \times idf_t$

# Implementation\*

```
import math
from textblob import TextBlob as tb

def tf(word, blob):
    return blob.words.count(word) / len(blob.words)

def n_containing(word, bloblist):
    return sum(1 for blob in bloblist if word in blob.words)

def idf(word, bloblist):
    return math.log(len(bloblist) / (1 + n_containing(word,
bloblist)))

def tfidf(word, blob, bloblist):
    return tf(word, blob) * idf(word, bloblist)
```

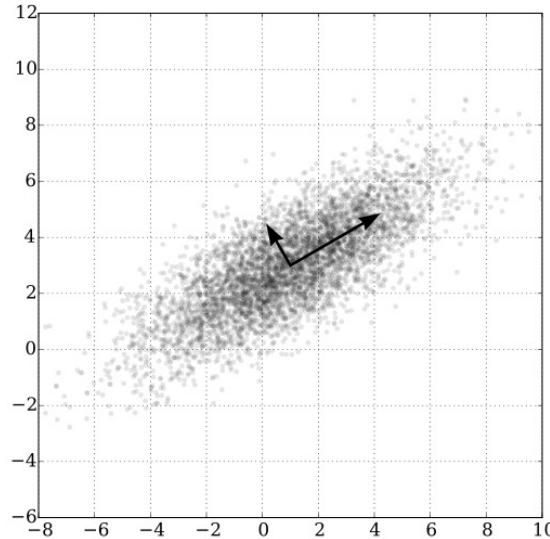
Showing a python demo ...

\* Example is taken from [here](#)

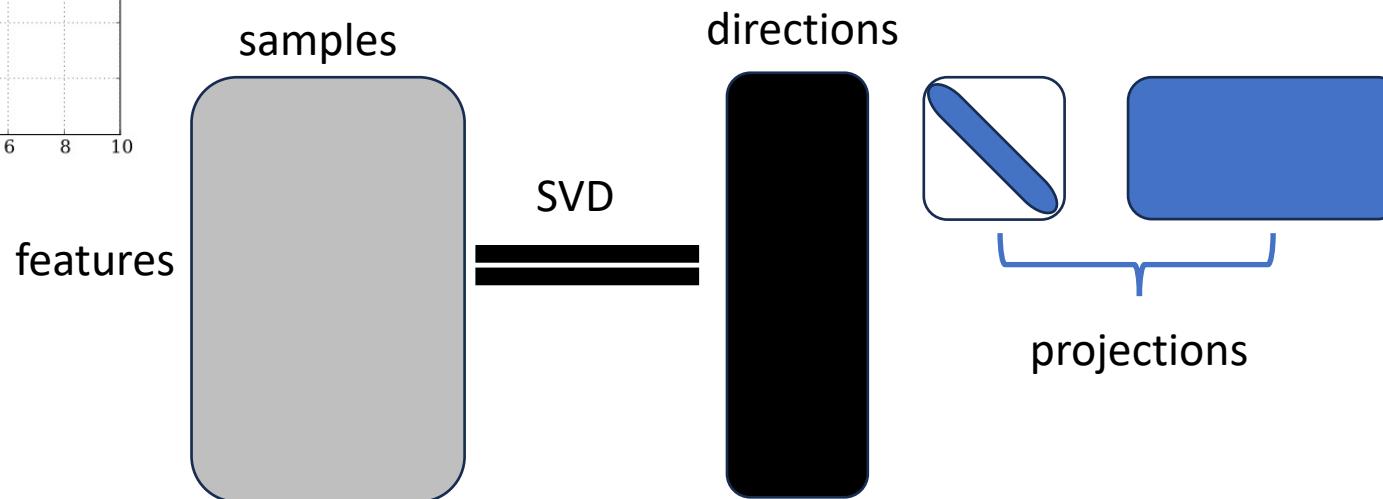
# Brain storming

- Ways to improve bag of words representation?
  - Word order information: counts of tuple of n words (n-gram)
    - tf is unigram probability in document
  - For “new” word: sub-word counts
  - More efficient: dimension reduction, PCA etc...
  - ...

# Dimension Reduction



- For  $i=1, \dots, \text{reduced\_dimension}$ :
  - Find next orthogonal direction with biggest variance
  - Project feature to this direction
- Easily solved by singular value decomposition (SVD)



# Roadmap

- Document representation (classical)
  - Bag of words
  - tf-idf
- Word representation (modern)
  - Word2vec and PMI
  - Applications
- Classifier
  - Naïve Bayesian classifier
  - Softmax classifier

# Word representation

- Endow each word a vector
- Most naïve way: one-hot vector, length equal to vocab size
- A better way, ensure

*Similar words are “nearby in semantic space”*

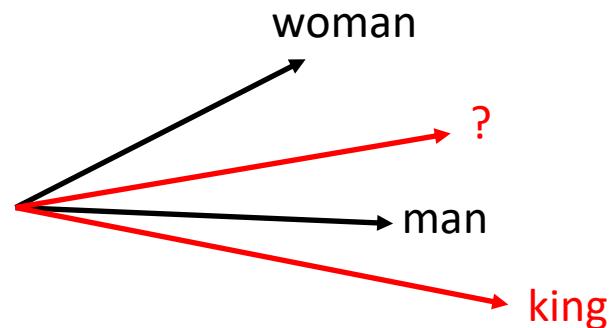
# Word representation



- Aka “word embedding”, as it’s embedded into a space
- Why represent them as vectors?

# One use case: Word Analogy

- Allows us to use linear algebra
- Man to woman is like king to ?



$$\text{man} - \text{woman} = \text{king} - ?$$

$$\Rightarrow ? = \text{king} - \text{man} + \text{woman}$$

$\Rightarrow$  Take ? as the nearest neighbor of RHS

# word2vec

## Efficient Estimation of Word Representations in Vector Space

by T Mikolov · 2013 · Cited by 37327 — Download a PDF of the paper titled Efficient Estimation of Word Representations in Vector Space, by Tomas Mikolov and 3 other authors.

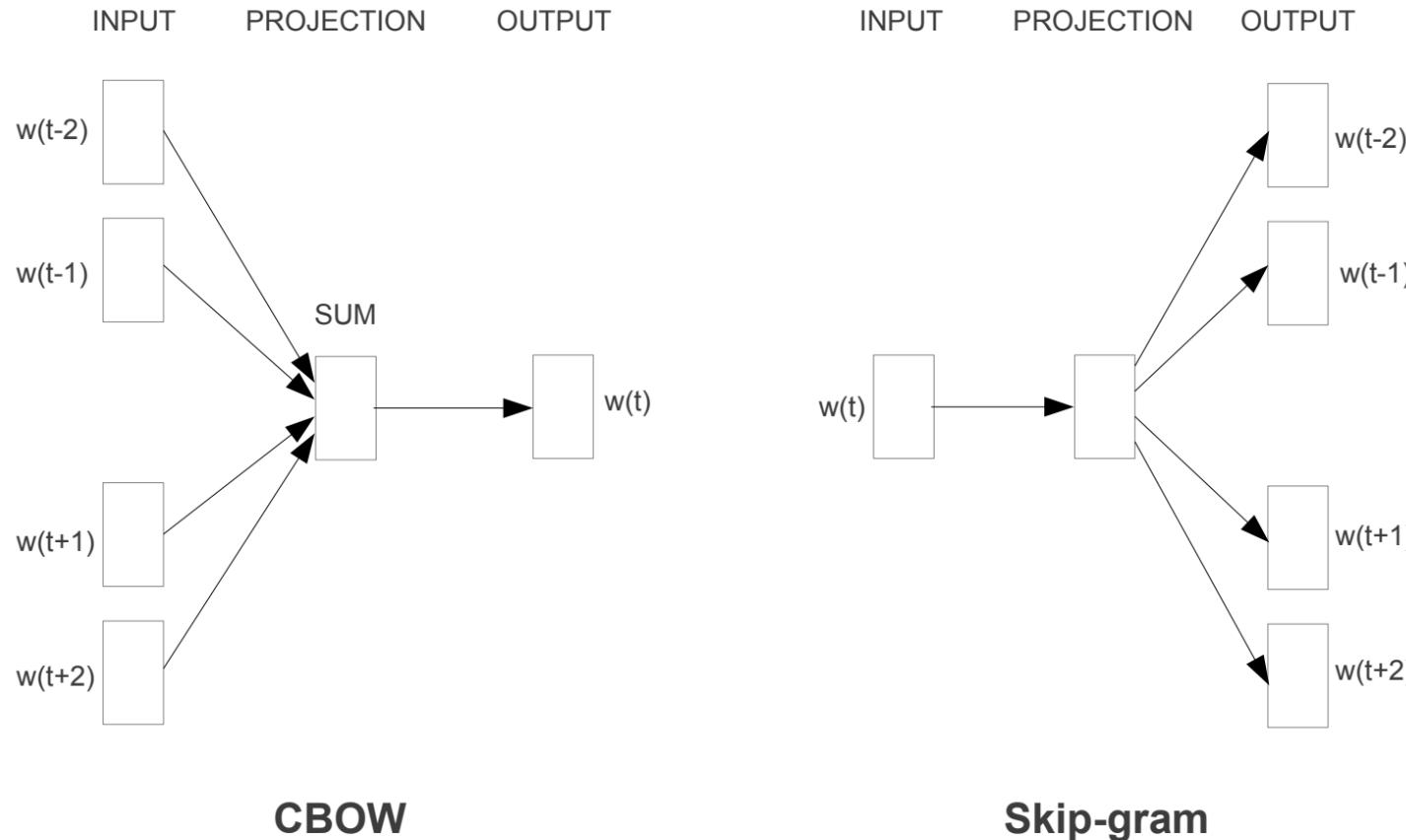


Figure 1: New model architectures. The **CBOW** architecture predicts the current word based on the context, and the **Skip-gram** predicts surrounding words given the current word.

# Deep dive into skip-gram

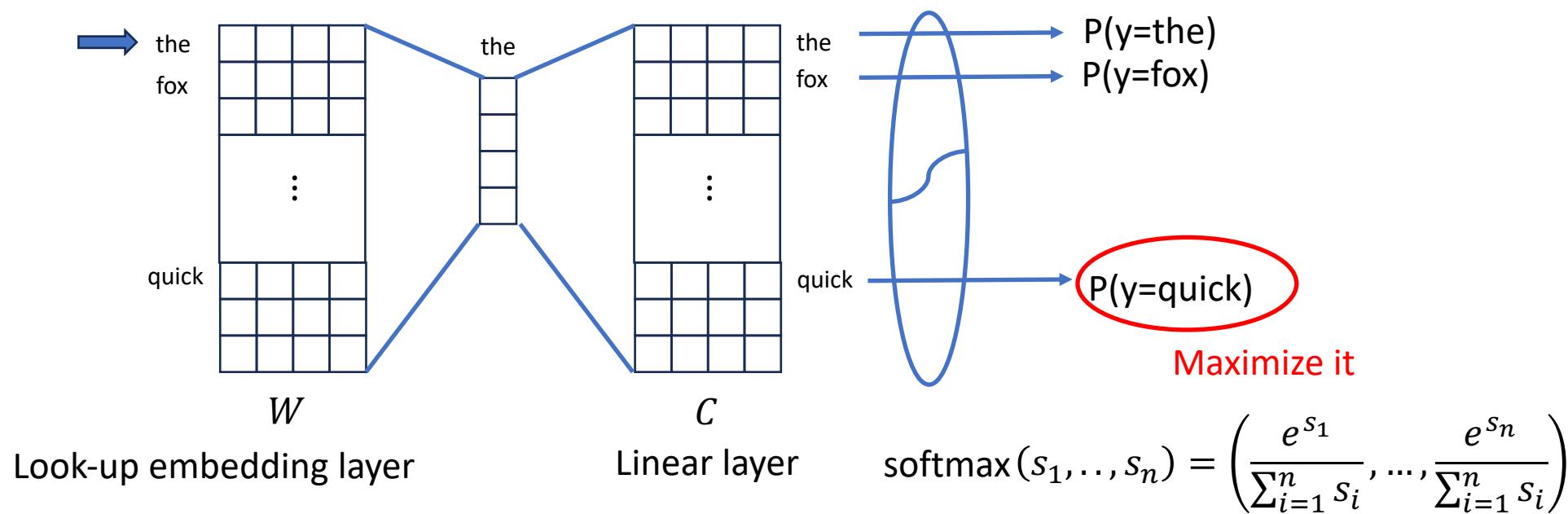
How do we train it? Suppose a window size of 2

Source Text	Training Samples
The quick brown fox jumps over the lazy dog. ➔	(the, quick) (the, brown)
The quick brown fox jumps over the lazy dog. ➔	(quick, the) (quick, brown) (quick, fox)
The quick brown fox jumps over the lazy dog. ➔	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox jumps over the lazy dog. ➔	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

Figure taken from [blogpost](#)

# Deep dive into skip-gram

- Viewed as 2-layer network
- e.g., receive a training sample ( $x=\text{the}$ ,  $y=\text{quick}$ )



# Why softmax here

- The outputs after linear layer are not probabilities
- i.e., they don't add to 1
- Softmax converts them to probabilities
- e.g., consider

$$[s_1, \dots, s_6] = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

We will revisit this later

$$\begin{aligned} softmax(s_1, \dots, s_6) &= \left[ \frac{e^{s_1}}{\sum_{i=1}^6 s_i}, \dots, \frac{e^{s_6}}{\sum_{i=1}^6 s_i} \right] \\ &= [0.055, 0.090, 0.0067, 0.10, 0.74, 0.010] \end{aligned}$$

# Deep dive into skip-gram

- The overall training objective is

$$\min_{\Theta} \sum_{x,y} -\log p_{\Theta}(y|x)$$

where  $\Theta$  are parameters in the network

- Typically, we use  $W$  as word representation
- Trained by stochastic gradient descent (SGD)
- Details in last part of this lecture!

# skip-gram with negative sampling



Neural Information Processing Systems

<https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-in-a-common-vector-space.pdf>

⋮

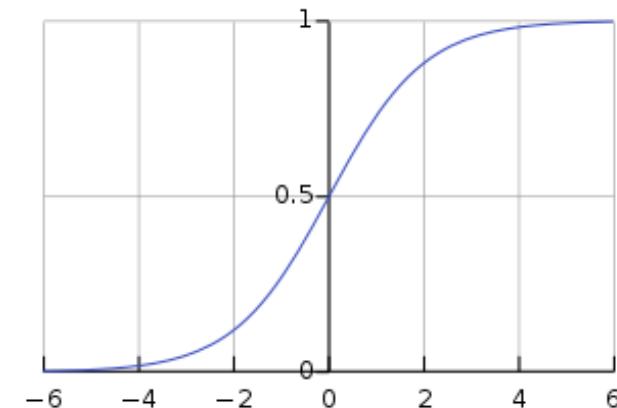
[Distributed Representations of Words and Phrases ...](#)

by T Mikolov · 2013 · Cited by 40525 — An inherent limitation of word representations is their indifference to word order and their inability to represent idiomatic phrases. For example, the...

- Vocab can be huge
- huge summation on softmax denominator!
- An efficient variant: contrast with negative samples
- $P(x \text{ and } y \text{ co-occur}) = \sigma(W_x^T C_y)$ ,

where  $\sigma(s) = \frac{1}{1+e^{-s}}$ , sigmoid function

Will revisit this later



# skip-gram with negative sampling

- Denote negative samples as  $y_i$ ,  $i = 1, \dots, k$  from  $q(y)$
- Maximize:

$$\log \sigma(W_x^T C_y) + \sum_{i=1}^k [\log \sigma(-W_x^T C_{y_i})]$$

- In practice,  $q(y)$  is uni-gram probability
- Is this a good approximation for softmax?

# Packages

- native c [implementation](#)
- handy python package: [genism](#), [fastText](#), ...

# Why so successful

- See Prof. Church's [opinion piece](#)

## Word2Vec

Published online by Cambridge University Press: **16 December 2016**

The screenshot shows a digital journal article page. At the top, it displays the author's name, KENNETH WARD CHURCH, and a link to 'Show author details'. Below this is a navigation bar with tabs: 'Article' (which is selected and highlighted in black), 'Figures', and 'Metrics'. Underneath the navigation bar are four buttons: 'Save PDF', 'Share', 'Cite', and 'Rights & Permissions'. The main content area is titled 'Abstract' and contains a detailed paragraph about the success of Word2Vec.

### Abstract

My last column ended with some comments about Kuhn and word2vec. Word2vec has racked up plenty of citations because it satisfies both of Kuhn's conditions for emerging trends: (1) a few initial (promising, if not convincing) successes that motivate early adopters (students) to do more, as well as (2) leaving plenty of room for early adopters to contribute and benefit by doing so. The fact that Google has so much to say on 'How does word2vec work' makes it clear that the definitive answer to that question has yet to be written. It also helps citation counts to distribute code and data to make it that much easier for the next generation to take advantage of the opportunities (and cite your work in the process).

- A few initial successes
- Plenty of room to improve
- Publish code and data

# PMI

## Word Association Norms, Mutual Information, and ...

by K Church · 1990 · Cited by 6485 — Anthology ID: J90-1003; Volume: Computational Linguistics, Volume 16, Number 1, March 1990; Month: Year: 1990; Address: Venue: CL; SIG:...

- The PMI (point wise mutual information) between two words  $w_1, w_2$

$$PMI(w_1; w_2) \triangleq \log_2 \frac{p(w_1, w_2)}{p(w_1)p(w_2)} = \log_2 \frac{p(w_1|w_2)}{p(w_1)}$$

- Consider  $w_2$  as context word
- 2D count table

	$w_2$					
	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

# PMI

	$w_2$	computer	data	result	pie	sugar	count(w)
<b>cherry</b>		2	8	9	442	25	486
<b>strawberry</b>		0	0	1	60	19	80
<b>digital</b>		1670	1683	85	5	4	3447
<b>information</b>		3325	3982	378	5	13	7703
<b>count(context)</b>		4997	5673	473	512	61	11716

- $p(w_1) = \frac{count(w_1)}{\sum_{w_1} counts(w_1)}$
- $p(w_1, w_2) = \frac{count(w_1, w_2)}{\sum_{w_1, w_2} count(w_1, w_2)}$
- Practically:  
How associated are two words?

# Positive PMI

- PMI ranges from  $-\infty$  to  $+\infty$
- Negative values are problematic as
  - Unreliable without enormous corpora
    - Suppose  $w_1$  and  $w_2$  each occur with probability  $10^{-6}$
    - Hard to be sure  $p(w_1, w_2)$  is significantly different than  $10^{-12}$
- So we just replace negative PMI with 0

$$PPMI(w_1; w_2) = \max(PMI(w_1; w_2), 0)$$

# Example

	$w_2$					
	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

$$p(w_1 = \text{information}, w_2 = \text{data}) = \frac{3982}{111716} = 0.3399$$

$$p(w_1 = \text{information}) = \frac{7703}{11716} = 0.6575$$

$$p(w_2 = \text{data}) = \frac{5673}{11716} = 0.4842$$

$$PMI(\text{information}, \text{data}) = \log_2 \frac{0.3399}{0.6575 \times 0.4842} = 0.0944$$

# Example

Resulting PPMI matrix (negatives replaced by 0)

	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

# Connections



Neural Information Processing Systems  
<https://proceedings.neurips.cc/paper/5477> · [...⋮](#)

## Neural Word Embedding as Implicit Matrix Factorization

by O Levy · Cited by 2384 — We show that using a sparse Shifted Positive **PMI** word-context matrix to represent **words** improves results on two word similarity tasks and one of two...

- Do some math on whiteboard ...

# Generalize: \*2vec

- Create pairwise associations between entities, call it matrix  $M$
- Then factorize  $M$ !

## [node2vec: Scalable Feature Learning for Networks](#)

by A Grover · 2016 · Cited by 10317 — Here we propose **node2vec**, an algorithmic framework for learning continuous feature representations for nodes in networks. In **node2vec** ...

## [\[1902.03545\] Task2Vec: Task Embedding for Meta-Learning](#)

by A Achille · 2019 · Cited by 250 — Abstract: We introduce a method to provide vectorial representations of visual classification tasks which can be used to reason about the ...

# Measure word similarity with embeddings

- Dot product

$$v^T w = \sum_{i=1}^d v_i w_i$$

- Big if their angle is small
- But also favors long vectors, i.e.,  $\|v\| = \sqrt{\sum_{i=1}^d v_i^2}$  big
- Frequent words have long vectors (why so?)

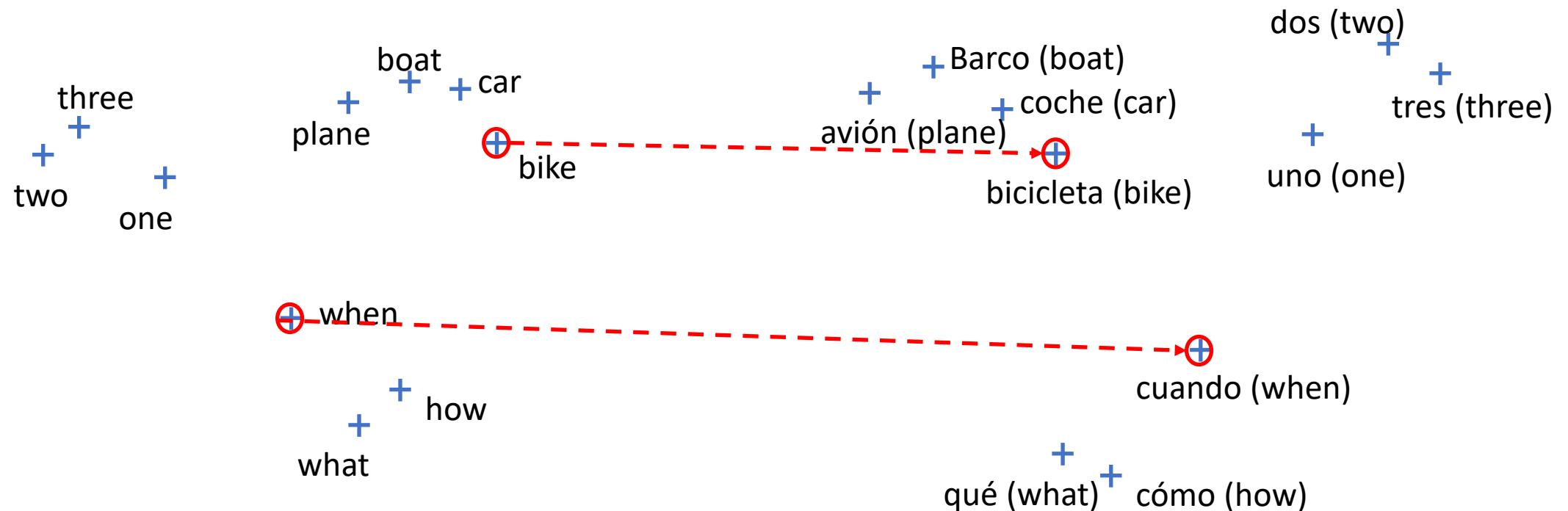
# Measure word similarity with embeddings

- Normalize by length

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

# Application: Bilingual Lexicon Induction (BLI)

- Translate words without aligned bilingual text
- Introduce some “anchors”



# Application: Bilingual Lexicon Induction (BLI)

- English anchors:  $X = [x_1, \dots x_n]$ , Spanish anchors:  $Y = [y_1, \dots y_n]$
- Learn a rotation matrix  $R$  that aligns them
$$\min_R \|RX - Y\|^2$$
- Nearest neighbor search for  $Rx$  in the space of  $y$ 's
- Extension: zero-shot learning

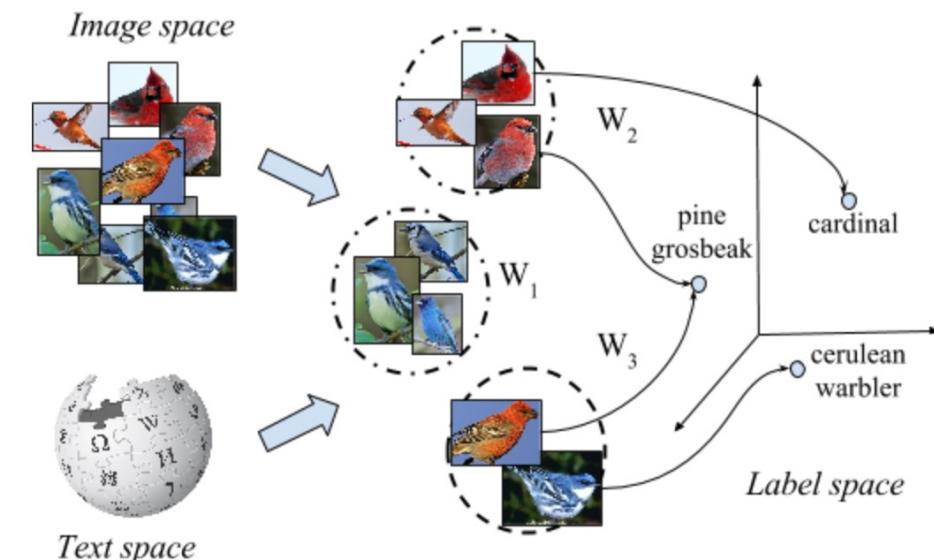


Image taken from [here](#)

# Connection with BERT

- Masked Language modeling
- Randomly mask some word
- Ask to predict based on context words, e.g.

The [mask] brown fox [mask] over [mask] lazy dog



arXiv

<https://arxiv.org/> > cs



[BERT: Pre-training of Deep Bidirectional Transformers for ...](#)

by J Devlin · 2018 · Cited by 77914 — We introduce a new language representation model called **BERT**, which stands for **Bidirectional Encoder Representations from Transformers**....



Wikipedia

<https://en.wikipedia.org> > wiki > BERT\_(language\_mo...



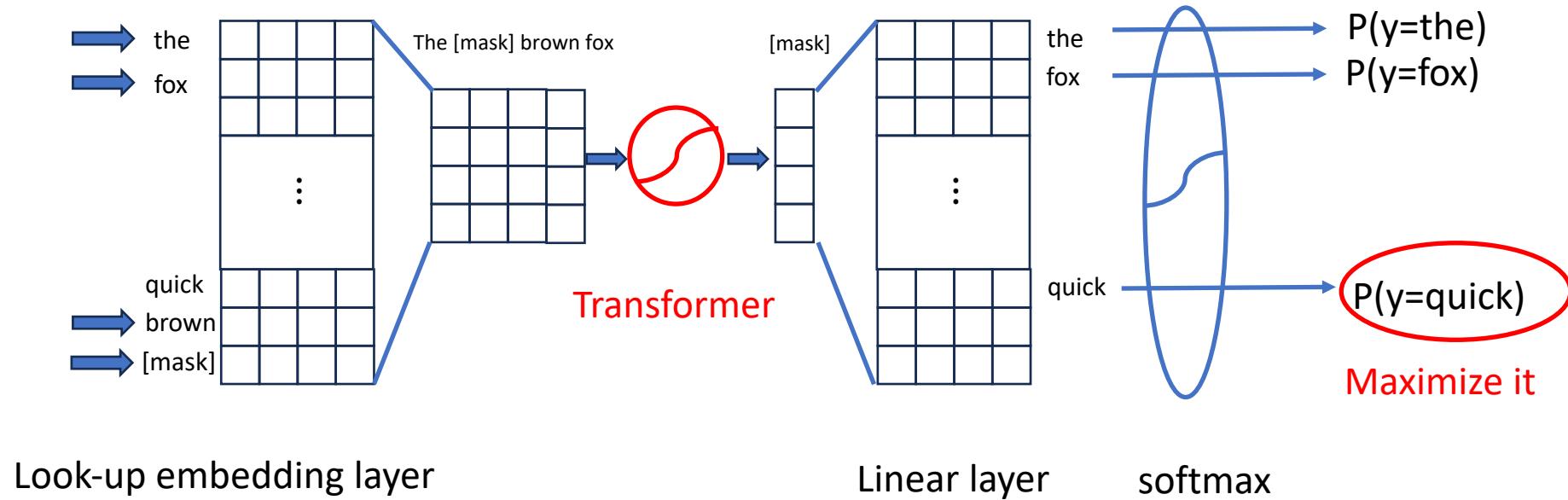
[BERT \(language model\)](#)

**Bidirectional Encoder Representations from Transformers (BERT)** is a family of language models introduced in 2018 by researchers at Google.

[Architecture](#) · [Performance](#) · [Analysis](#) · [History](#)

# Connection with BERT

## The BERT architecture



# Roadmap

- Document representation (classical)
  - Bag of words
  - tf-idf
- Word representation (modern)
  - Word2vec and PMI
  - Applications
- Classifier
  - Naïve Bayesian classifier
  - Softmax classifier

# Naïve Bayesian Classifier

- Feature  $f \in \mathbb{R}^d$ , label  $y \in \{1, \dots, c\}$ . Invoke Bayesian rule:
- $p(y|f) = \frac{p(y)p(f|y)}{p(f)}$
- Only interested in denominator
- Assume **conditional independence** among feature dimensions

$$p(y|f) \propto p(y) \prod_i p(f_i|y)$$

Parameterized, e.g., Gaussian, multinomial

# Start Slides from Last term

Caveat: notation inconsistency, but we will explain

# Training NBC for document classification

Maximum Likelihood Estimates: just use the frequencies in the training set

$$\hat{P}(c_j) = \frac{\text{doccount}(C = c_j)}{N_{\text{doc}}}$$

$$\hat{P}(w_i | c_j) = \frac{\text{count}(w_i, c_j)}{\sum_{w \in V} \text{count}(w, c_j)}$$

To calculate  $\text{count}(w_i, c_j)$ : concatenate all documents in class  $c$  into a mega-document and use the frequencies in that mega-document

# Problem with Maximum Likelihood

- What if we have seen no training documents with the word ***fantastic*** and classified in the topic **positive (*thumbs-up*)**?

$$\hat{P}(\text{"fantastic"} \mid \text{positive}) = \frac{\text{count}(\text{"fantastic"}, \text{positive})}{\sum_{w \in V} \text{count}(w, \text{positive})} = 0$$

# Problem with Maximum Likelihood

- What if we have seen no training documents with the word ***fantastic*** and classified in the topic **positive (thumbs-up)**?

$$\hat{P}(\text{"fantastic"} \mid \text{positive}) = \frac{\text{count}(\text{"fantastic"}, \text{positive})}{\sum_{w \in V} \text{count}(w, \text{positive})} = 0$$

- Zero probabilities cannot be conditioned away, no matter the other evidence!

$$c_{MAP} = \operatorname{argmax}_c \hat{P}(c) \prod_i \hat{P}(x_i \mid c)$$

# Laplace (add-1) smoothing for Naïve Bayes

$$\hat{P}(w_i | c) = \frac{\text{count}(w_i, c) + 1}{\sum_{w \in V} (\text{count}(w, c) + 1)}$$
$$= \frac{\text{count}(w_i, c) + 1}{\left( \sum_{w \in V} \text{count}(w, c) \right) + |V|}$$

# Multinomial Naïve Bayes: Learning

- From training corpus, extract *Vocabulary*
  - Calculate  $P(c_j)$  terms
    - For each  $c_j$  in  $C$  do  
 $docs_j \leftarrow$  all docs with class =  $c_j$
  - Calculate  $P(w_k | c_j)$  terms
    - $Text_j \leftarrow$  single doc containing all  $docs_j$
    - For each word  $w_k$  in *Vocabulary*  
 $n_k \leftarrow$  # of occurrences of  $w_k$  in  $Text_j$
- $$P(c_j) \leftarrow \frac{|docs_j|}{|\text{total \# documents}|}$$
- $$P(w_k | c_j) \leftarrow \frac{n_k + \alpha}{n + \alpha |\text{Vocabulary}|}$$

$$\hat{P}(c) = \frac{N_c}{N}$$

$$\hat{P}(w|c) = \frac{\text{count}(w,c)+1}{\text{count}(c)+|V|}$$

**Priors:**

$$P(c) = \frac{3}{4}$$

$$P(j) = \frac{1}{4}$$

	Doc	Words	Class
Training	1	Chinese Beijing Chinese	c
	2	Chinese Chinese Shanghai	c
	3	Chinese Macao	c
	4	Tokyo Japan Chinese	j
Test	5	Chinese Chinese Chinese Tokyo Japan	?

**Choosing a class:**

$$P(c|d5) \propto 3/4 * (3/7)^3 * 1/14 * 1/14 \\ \approx 0.0003$$

**Conditional Probabilities:**

$$P(\text{Chinese}|c) = (5+1) / (8+6) = 6/14 = 3/7$$

$$P(\text{Tokyo}|c) = (0+1) / (8+6) = 1/14$$

$$P(\text{Japan}|c) = (0+1) / (8+6) = 1/14$$

$$P(\text{Chinese}|j) = (1+1) / (3+6) = 2/9$$

$$P(\text{Tokyo}|j) = (1+1) / (3+6) = 2/9$$

$$P(j|d5) \propto 1/4 * (2/9)^3 * 2/9 * 2/9 \\ \approx 0.0001$$

$$41 \quad P(\text{Japan}|j) = (1+1) / (3+6) = 2/9$$

# Underflow Prevention: log space

- Multiplying lots of probabilities can result in floating-point underflow.
- Since  $\log(xy) = \log(x) + \log(y)$ 
  - Better to sum logs of probabilities instead of multiplying probabilities.
- Class with highest un-normalized log probability score is still most probable.

$$c_{NB} = \operatorname{argmax}_{c_j \in C} \log P(c_j) + \sum_{i \in positions} \log P(x_i | c_j)$$

- Model is now just max of sum of weights

End Slides from Last term

# Questions

- How did we parameterize the likelihood?  
Multinomial
- How about tf-idf?  
Sure, just replace word count with it
- Packages? [Sklearn](#)
- ...

# Binary Classification

- Positive v.s. negative reviews
- Feature  $f \in \mathbb{R}^d$ , label  $y \in \{0, 1\}$
- Come up with  $P(Y = 1|f)$  or  $P(Y = 0|f) = 1 - P(Y = 1|f)$

# Logistic Regression

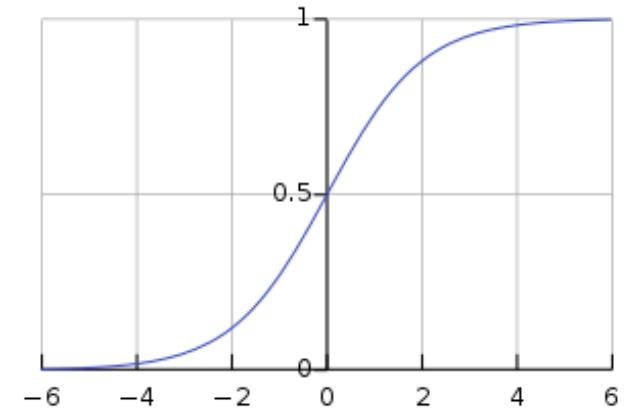
- Logit (raw score)  $s = \alpha^T f + \beta$ , where  $\alpha \in \mathbb{R}^d$ ,  $\beta \in \mathbb{R}$
- $p(Y = 1|f) = \sigma(s) = \frac{1}{1+e^{-s}}$  (sigmoid function)
- $p(Y = 0|f) = 1 - \sigma(s) = \sigma(-s)$
- Training loss

$$\min_{\alpha, \beta} \sum_{(f, y)} -\log p(Y = y|f)$$

- Note: more compact way is to define

$$\tilde{f} = [f; 1], \text{ and } \tilde{\alpha} = [\alpha; \beta]$$

$$\text{Then } s = \alpha^T f + \beta = \tilde{\alpha}^T \tilde{f}$$



# Generalize to Multi-class

- Sentiment classification: Excellent / Good / fair / poor / terrible
- POS tagging: noun / verb/ adjective / ...
- Multinomial logistic regression
- Aka softmax classifier

# Softmax classifier

- Feature  $f \in \mathbb{R}^d$ , label  $y \in \{1, \dots, c\}$
- Linear layer  $A = [a_1, \dots, a_c]$  to be learned
- Logits (raw scores)  $s_i = a_i^T f$
- Probabilities  $p_i(f) = \frac{e^{s_i}}{\sum_{j=1}^c e^{s_j}}$
- Loss function:

$$\min_A \frac{1}{\#\text{training samples}} \sum_{(f,y)} \{\ell(f, y) \triangleq -\log p_y(f)\}$$

# Gradient Solver

- Gradient Descent, Consider 1D case

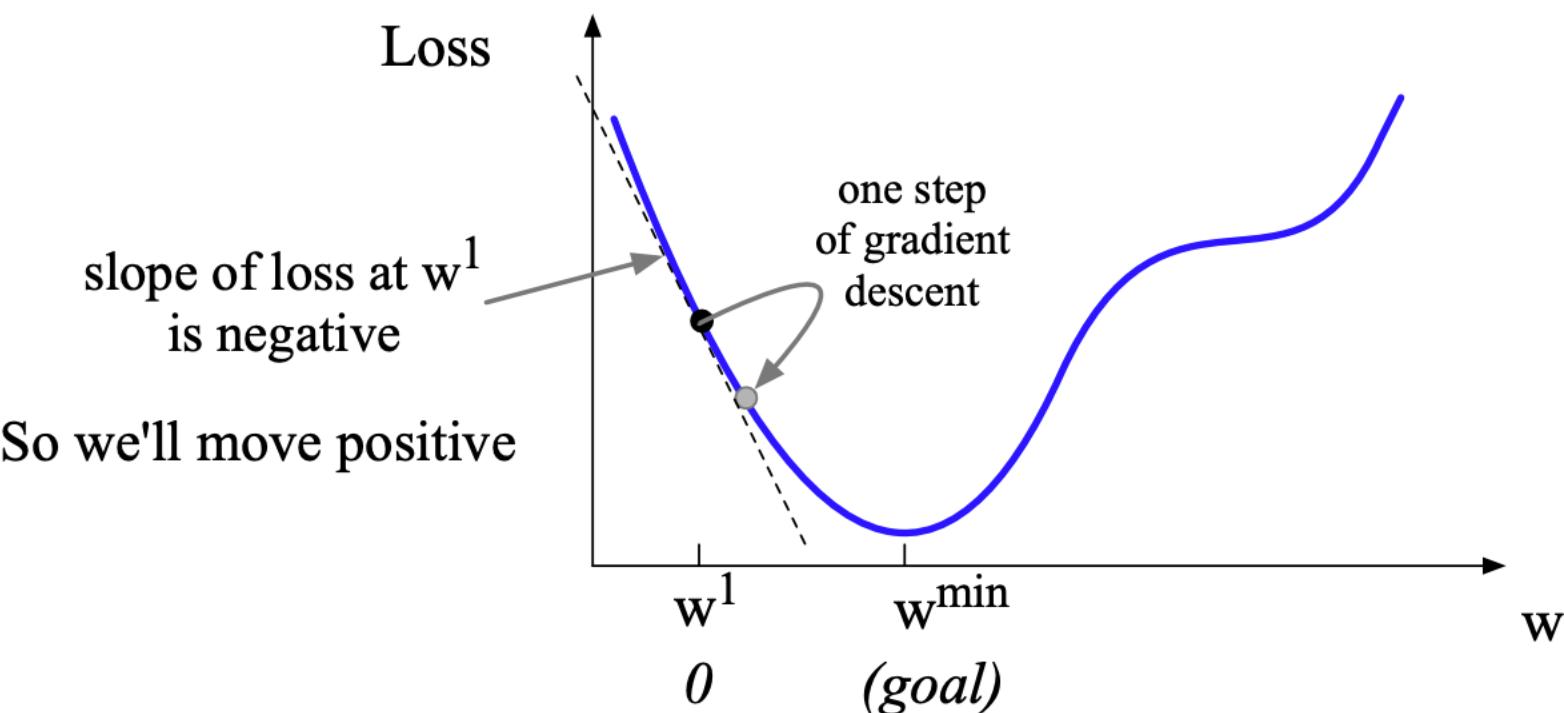
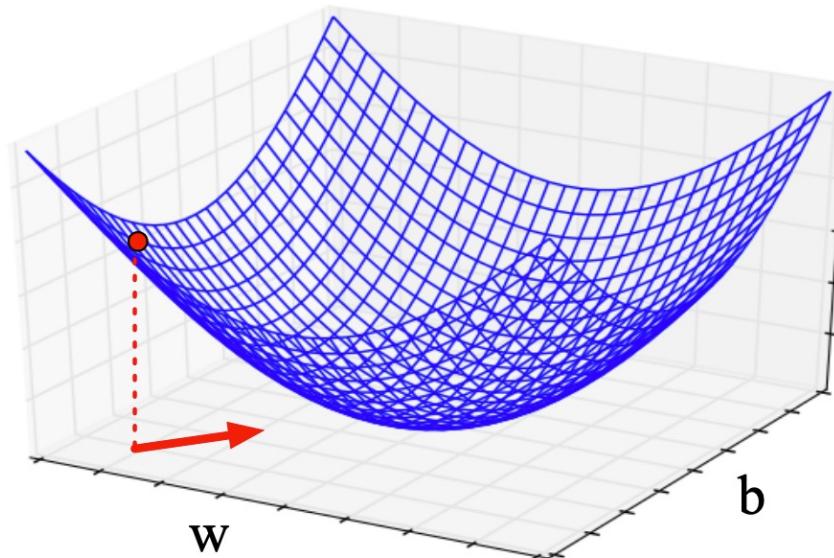


Image from [Stanford NLP class slides](#)

# Gradient Solver

- A loss function defined on  $\mathbb{R}^d$
- Gradient is a vector of  $\mathbb{R}^d$ , greatest increase of the function value
- So move along the opposite direction by some stepsize
- E.g., 2D



# Stochastic Gradient Solver

- Many loss functions are of the form

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^n \ell(x_i, \theta)$$

- Stochastic gradient descent (SGD):

Sample an  $x_i$  each time, move along  $-\frac{\partial \ell(x_i, \theta)}{\partial \theta}$

- Minibatch SGD:

Sample a batch of  $x_i$ 's each time, move along  $-\frac{1}{|\mathcal{B}|} \sum_{x_i \in \mathcal{B}} \frac{\partial \ell(x_i, \theta)}{\partial \theta}$

# Softmax classifier

- Search optimal  $A$  by **Stochastic** Gradient Descent (SGD)
- Derive gradient due to a sample  $(f, y)$

$$\frac{\partial \ell(f, y)}{\partial a_i} = \frac{\partial}{\partial a_i} \left\{ \log \sum_{j=1}^c e^{s_j} - s_y \right\} = \begin{cases} (p_y - 1)f, & i = y \\ p_i f, & i \neq y \end{cases}$$

Initialize  $A$  randomly

While stopping criteria not met:

sample a batch of  $(f, y)$ 's, call the batch  $\mathcal{B}$

$a_i \leftarrow \gamma \cdot \sum_{(f,y) \in \mathcal{B}} \frac{\partial \ell(f, y)}{\partial a_i}$  for  $i = 1, \dots, c$

# Questions

- Does initialization matter?  
No (because of convexity)
- How do we choose batch size?  
Trade-off between memory cost and time to convergence
- What if  $f$  also needs to be learned?  
chain rule (pytorch excels)
- What if  $c$  huge?
- ...

# Generalize

- Generally, we can write

$$\ell(f, y; \theta) = -s_y(\theta) + \log \sum_{j=1}^c e^{s_j(\theta)}$$
$$\frac{\partial \ell}{\partial \theta} = -\nabla s_y(\theta) + \sum_{\substack{j=1 \\ e^{s_j}}}^c p_j \nabla s_j(\theta)$$
$$p_j = \boxed{\sum_{j=1}^c e^{s_j}}$$

- Costly if  $c$  big

# Sampled softmax

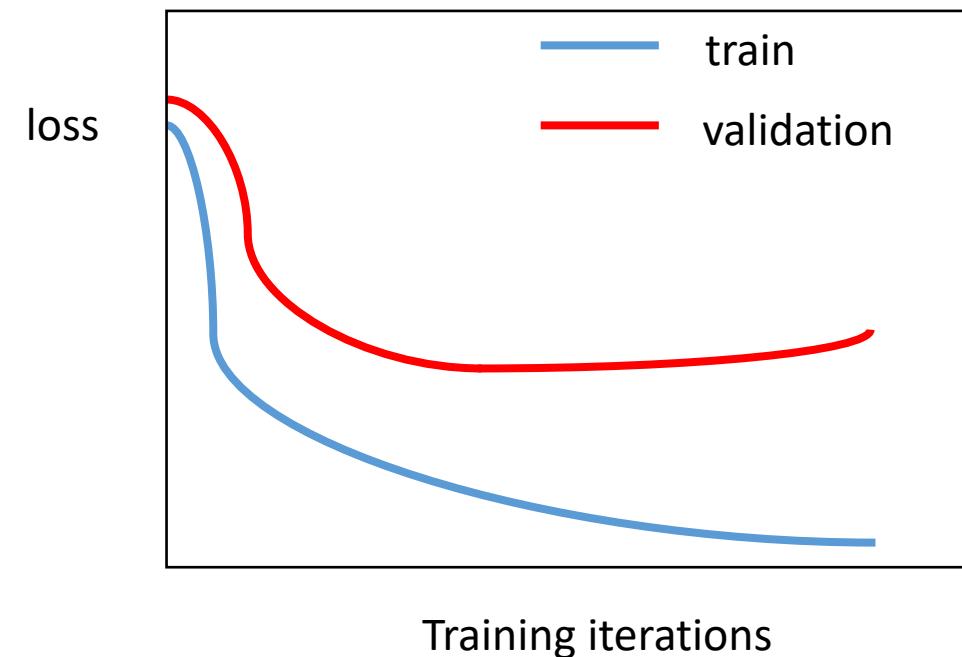
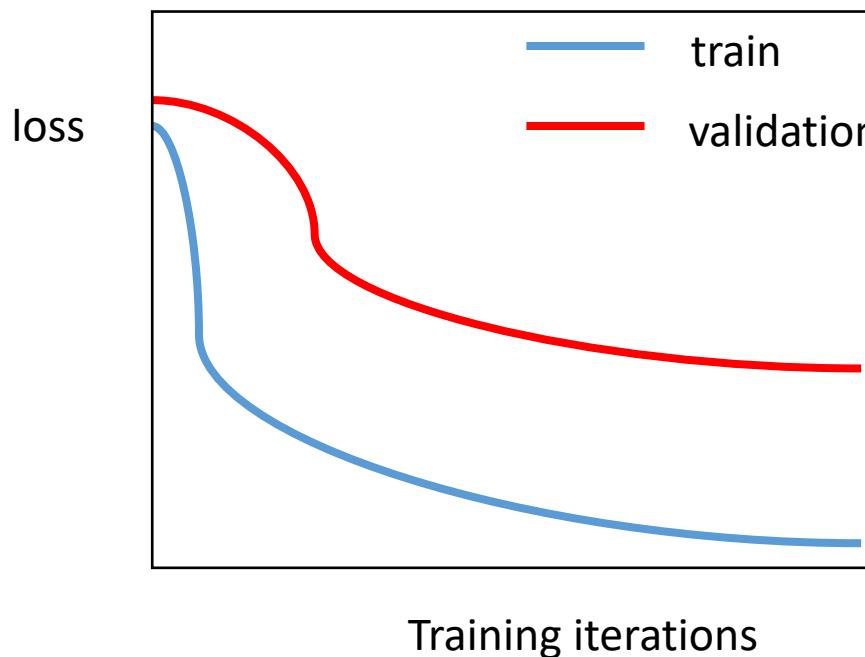
- Approximate  $\sum_{j=1}^c p_j \nabla s_j(\theta)$  with
$$\sum_{j=1}^{c'} q_j \nabla s_j(\theta), c' < c$$
- E.g.,  $q_j$ : unigram probability
- Speed up training, but not testing

# Discussion

- Strength and weakness of Naïve Bayesian v.s. softmax
- Small data: Naïve Bayesian
- Big data: softmax, integrates well with joint learning of features

# Overfitting

- Softmax classifier in small data regime
- Typical learning curve trends:



# Regularization

- A solution for overfitting
- Add a regularizer

$$\min_A \sum_{(f,y)} \ell(f, y) + \alpha R(A)$$

- Usually,  $R(A)$  encourages weights in  $A$  to be small

# Two typical $R(A)$ 's

- L2 regularization, aka ridge regression

$$R(A) = \|A\|^2 = \sum_{i,j} A_{i,j}^2$$

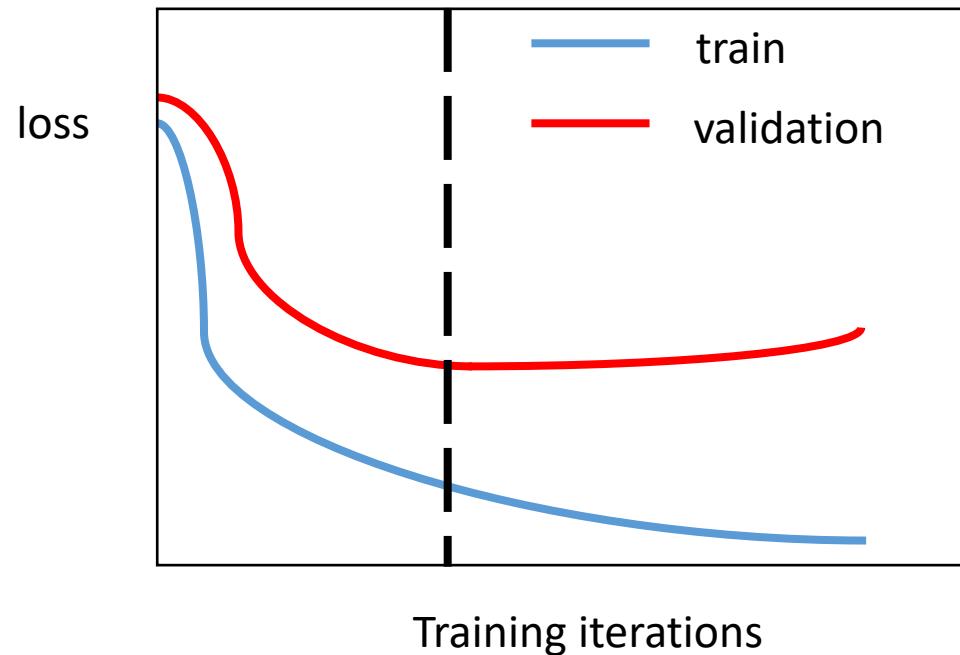
- L1 regularization, aka lasso

$$R(A) = \|A\|_1 = \sum_{i,j} |A_{i,j}|$$

# SGD with regularizers

- Additional term to  $\frac{\partial \ell}{\partial a_i}$
- L2 regularization:  $R(A) = \|A\|^2$   
$$\frac{\partial R(A)}{\partial A_{i,j}} = 2A_{i,j}$$
- L1 regularization:  $R(A) = \|A\|_1$   
$$\frac{\partial R(A)}{\partial A_{i,j}} = sign(A_{i,j})$$

# Early stopping



Stop training around here