Lecture 7: Feed-forward Neural Nets

Instructor: Swabha Swayamdipta

USC CSCI 544 Applied NLP

Sep 17, Fall 2024

Some slides adapted from Dan Jurafsky and Chris Manning and Xuezhe Ma

Announcements + Logistics

- Thu: HW1 due / HW2 released
- Today: Quiz 2
- Next Thu: Project Proposal Due
- For queries, please contact all TAs / graders / myself through the mailing list:
  - Not just me
  - 2024f-csci544-staff@googlegroups.com

Lecture Outline

- Announcements
- Recap: Sparse and Dense Word Embeddings, word2vec
- GloVe
- Word Embeddings: Properties and Evaluation
- Quiz 2
- Feed-forward Neural Nets

Recap:
Sparse Word Embeddings

Term-document matrix

Let us consider a collection of documents and count how frequently a word (**term**) appears in each. A document could be a play or a Wikipedia article. In general, documents can be anything; we often call each paragraph a document!

Each **document** is represented by a vector of words

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word-word co-occurrence matrix

Context Window

is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet

Two words are similar in meaning if their context vectors are similar

Words, not documents

	aardvark	...	**computer**	**data**	**result**	**pie**	**sugar**	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Pointwise Mutual Information (PMI)

$$PMI(w_1, w_2) = \log \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

- **PMI between two words:**
  - Do words w_1 and w_2 co-occur more than if they were independent?

- PMI ranges from $-\infty$ to $+\infty$
  - Negative values are problematic: words are co-occurring less than we expect by chance
  - Only reliable under an enormous corpora
    - Imagine w_1 and w_2 whose probability of occurrence is each 10^{-6}
    - Hard to be sure $P(w_1, w_2)$ is significantly different than 10^{-12}
  - So we just replace negative PMI values by 0

- **Positive PMI**

$$PPMI(w_1, w_2) = \max\left(0, \log \frac{P(w_1, w_2)}{P(w_1)P(w_2)}\right)$$

Church & Hanks 1989

Computing PPMI on a term-context matrix

Context c

Term w

	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

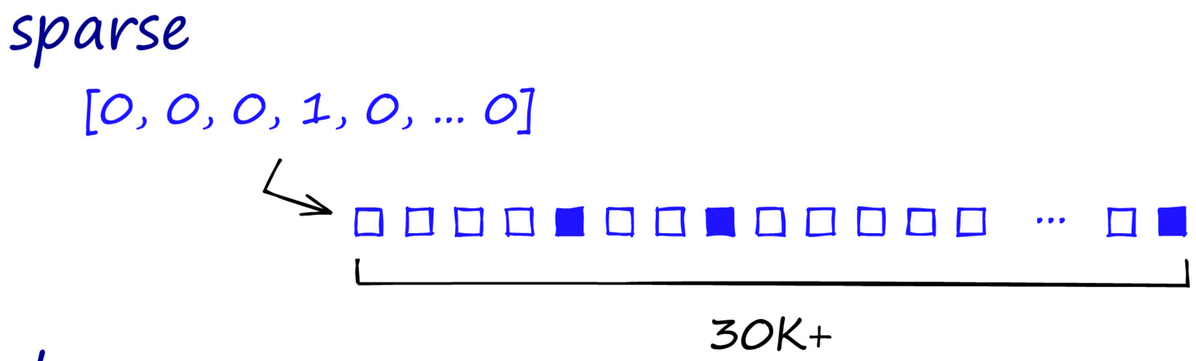
Frequency $f(w_i, c_j)$ or f_{ij} is the # times w_i occurs in context c_j

$$P_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{i,j}} \qquad P_i = \sum_{j=1}^{C} P_{ij} \qquad P_j = \sum_{i=1}^{W} P_{ij}$$

$$PPMI(w_i, c_j) = PPMI_{i,j} = \max\left(0, \log \frac{P_{ij}}{P_i P_j}\right)$$

The problem…

- PMI vectors are
  - long (length |V|= 20,000 to 50,000)
  - sparse (most elements are zero)



- Alternative: learn vectors which are
  - short (length 50-1000)
  - dense (most elements are non-zero)
    - Therefore, easier to use (fewer parameters)

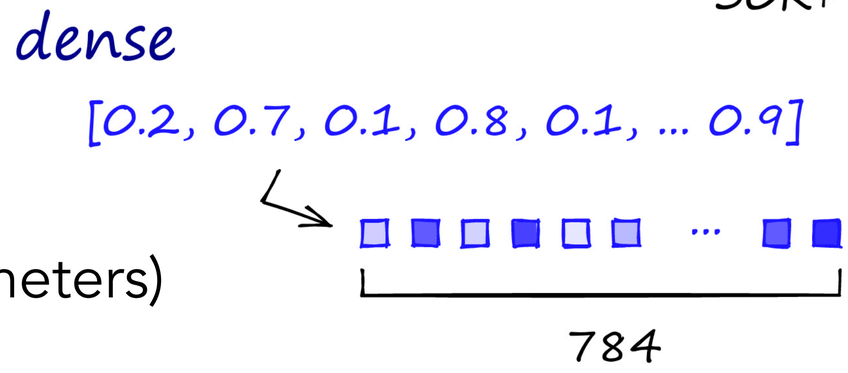


Image Credit: Pinecone

Recap: Dense Word Embeddings (word2vec)

word2vec

- Short, dense vector or embedding
- Static embeddings
  - One embedding per word type
  - Does not change with context change
- Two algorithms for computing:
  - Skip-Gram with Negative Sampling or SGNS
  - CBOW or continuous bag of words
  - But we will study a slightly different version…
- Efficient training
- Easily available to download and plug in

Mikolov et al., ICLR 2013. Efficient estimation of word representations in vector space.

Mikolov et al., NeurIPS 2013. Distributed representations of words and phrases and their compositionality.

word2vec : Intuition

is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet

Instead of counting how often each word w occurs near another, e.g. "cherry"

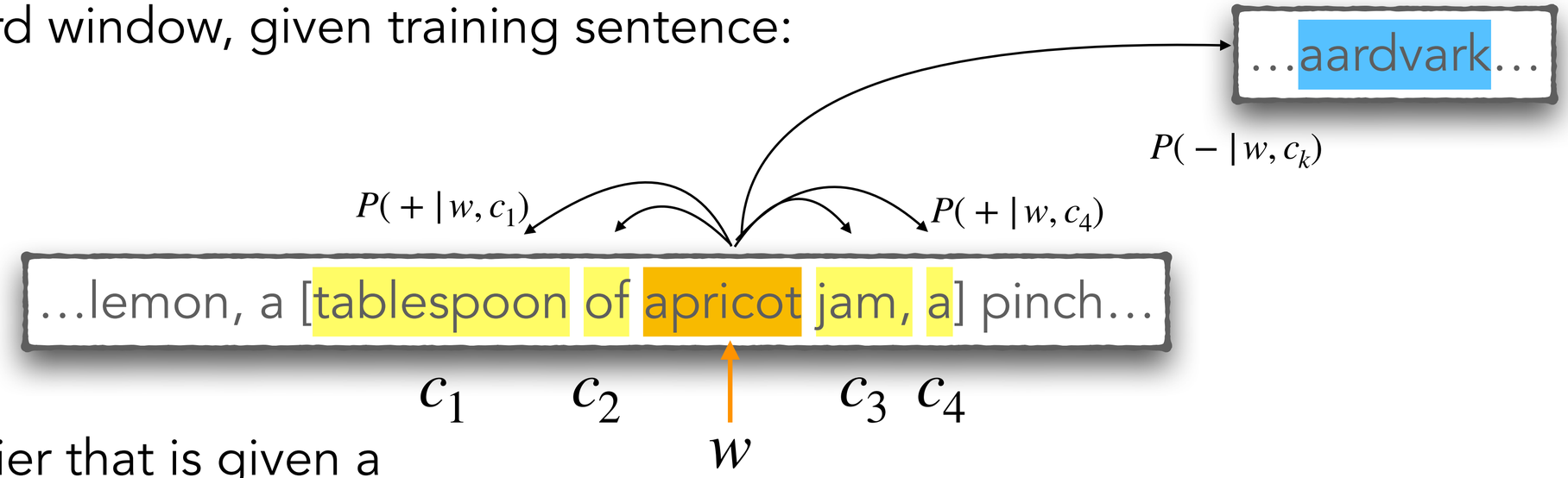
- Train a classifier on a binary prediction task:
  - Is w likely to show up near "cherry"?
- We don't actually care about this task!!!
  - But we'll take the learned classifier weights as the word embeddings

What is $\mathbf{x}$? What is y?

Word embedding itself is the learned parameter!

word2vec: Goal

Assume a +/- 2 word window, given training sentence:



Goal: train a classifier that is given a candidate (word, context) pair, and corresponding label about co-occurrence :

(apricot, jam), 1
(apricot, tablespoon), 1
(apricot, aardvark), 0
…

And assigns each pair a probability:

$$P(+|w, c)$$
$$P(-|w, c) = 1 - P(+|w, c)$$

Turning dot products into probabilities

Similarity:

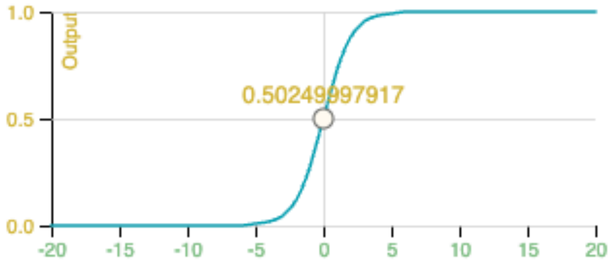
$$\text{sim}(w, c) \approx \mathbf{w} \cdot \mathbf{c}$$

Sigmoid

$$f(0.01) = \frac{1}{1 + e^{-(0.01)}} = 0.50249997917$$

Turn into a probability using the sigmoid function:

$$P(+|w, c) = \sigma(\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{c} \cdot \mathbf{w})}$$



$$P(-|w, c) = 1 - P(+|w, c)$$

$$= \sigma(-\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1 + \exp(\mathbf{c} \cdot \mathbf{w})}$$

Logistic Regression!

Accounting for a context window

$$P(+|w,c) = \sigma(\mathbf{c}\cdot\mathbf{w}) = \frac{1}{1+\exp(-\mathbf{c}\cdot\mathbf{w})}$$

Single Context Word

…lemon, a [tablespoon of apricot jam, a] pinch…

c_1 c_2 w c_3 c_4

But we have lots of context words

- Depends on window size, L
- We'll assume independence and just multiply them

Same with negative context words!

$$P(+|w,c_{1:L}) = \prod_{i=1}^{L} \sigma(\mathbf{c}_i\cdot\mathbf{w})$$

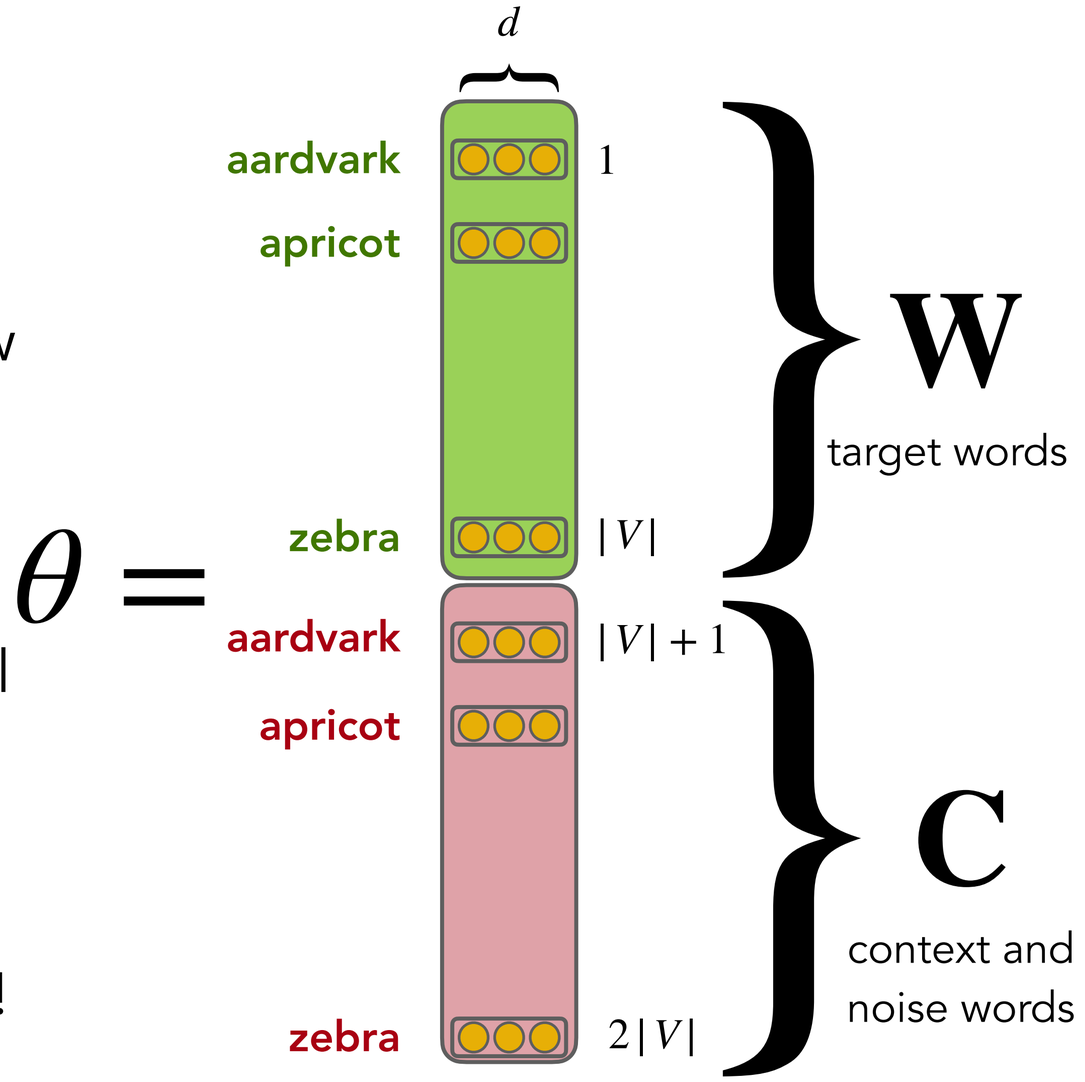
$$\log P(+|w,c_{1:L}) = \sum_{i=1}^{L} \log\sigma(\mathbf{c}_i\cdot\mathbf{w})$$

c_{neg} { …aardvark… …zebra…

$$\log P(-|w,c_{neg}) = \sum_{c'\in c_{neg}} \log\sigma(-\mathbf{c}'\cdot\mathbf{w})$$

word2vec classifier: Summary

- A probabilistic classifier, given
  - a test target word w
  - its context window of L words $c_{1:L}$
- Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings)
- To compute this, we just need embeddings for all the words
  - Separate representations for targets and contexts
  - Same as the parameters we need to estimate!



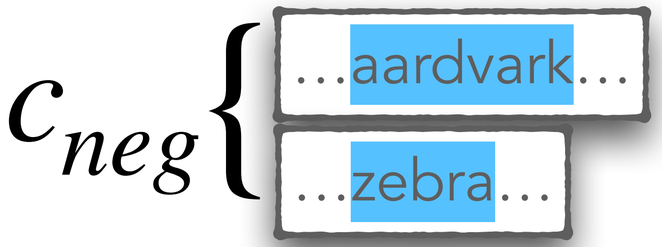
Lecture Outline

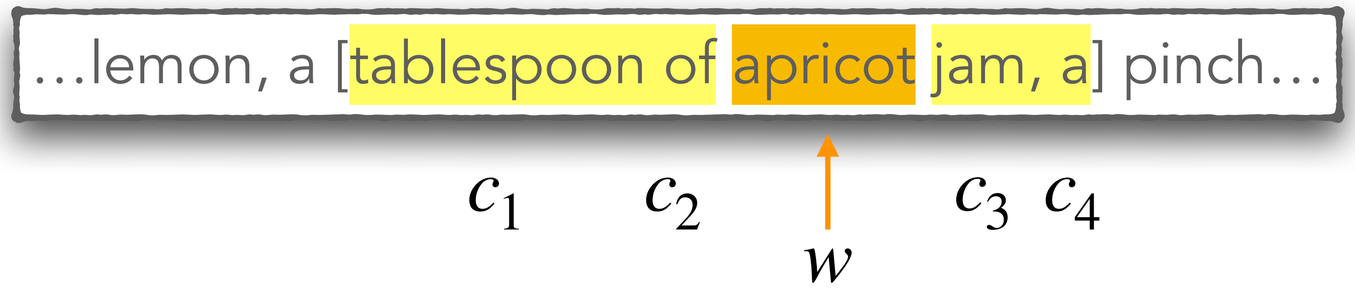
- Announcements
- Recap: Sparse and Dense Word Embeddings, word2vec
- GloVe
- Word Embeddings: Properties and Evaluation
- Quiz 2
- Feed-forward Neural Nets

Learning word2vec embeddings

Word2vec: Training Data

For each positive example we'll grab a set of negative examples, sampling by weighted unigram frequency





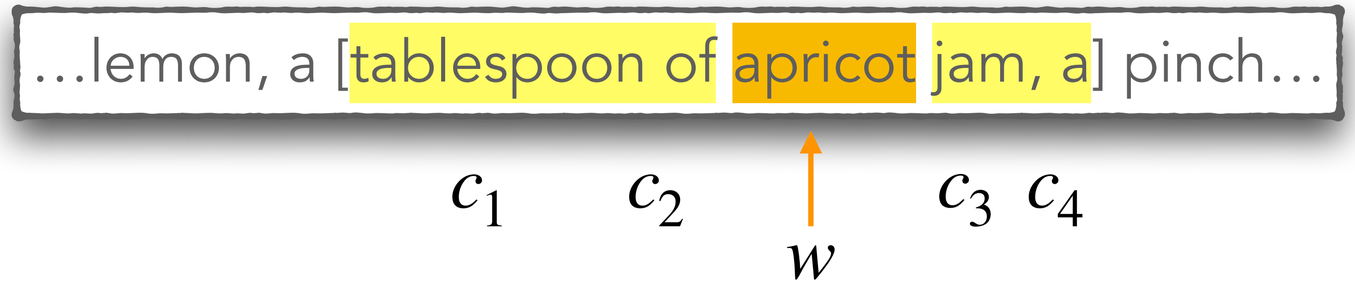
Negative examples

w	c_{neg}
apricot	aardvark
apricot	zebra
apricot	where
apricot	adversarial

Positive examples

w	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Word2vec: Learning Problem



Given

- the set of positive and negative training instances, and
- a set of randomly initialized embedding vectors of size $2|V|$,

the goal of learning is to adjust those word vectors such that we:

- **Maximize** the similarity of the target word, context word pairs $(w, c_{1:L})$ drawn from the positive data
- **Minimize** the similarity of the (w, c_{neg}) pairs drawn from the negative data

Loss function

Maximize the similarity of the target with the actual context words in a window of size L, and minimize the similarity of the target with the $K > L$ negative sampled non-neighbor words

For every word, context pair…

Cross Entropy

$$L_{CE} = -\log[P(+|\mathbf{w}, \mathbf{c}_{pos})P(-|\mathbf{w}, \mathbf{c}_{neg})]$$

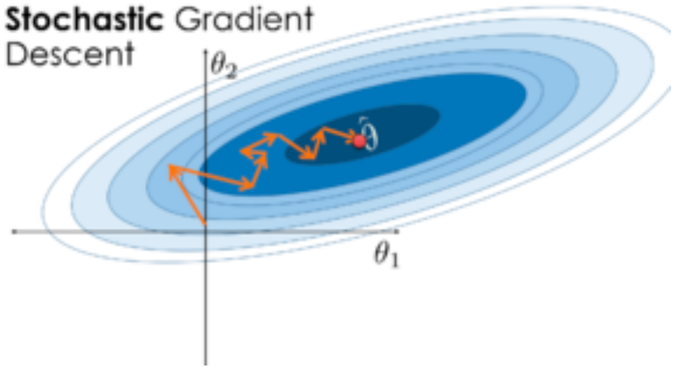
$$= -\left[\log P(+|\mathbf{w}, \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log P(-|\mathbf{w}, \mathbf{c}_{neg_j})\right]$$

$$= -\left[\log P(+|\mathbf{w}, \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log(1 - P(+|\mathbf{w}, \mathbf{c}_{neg_j})\right]$$

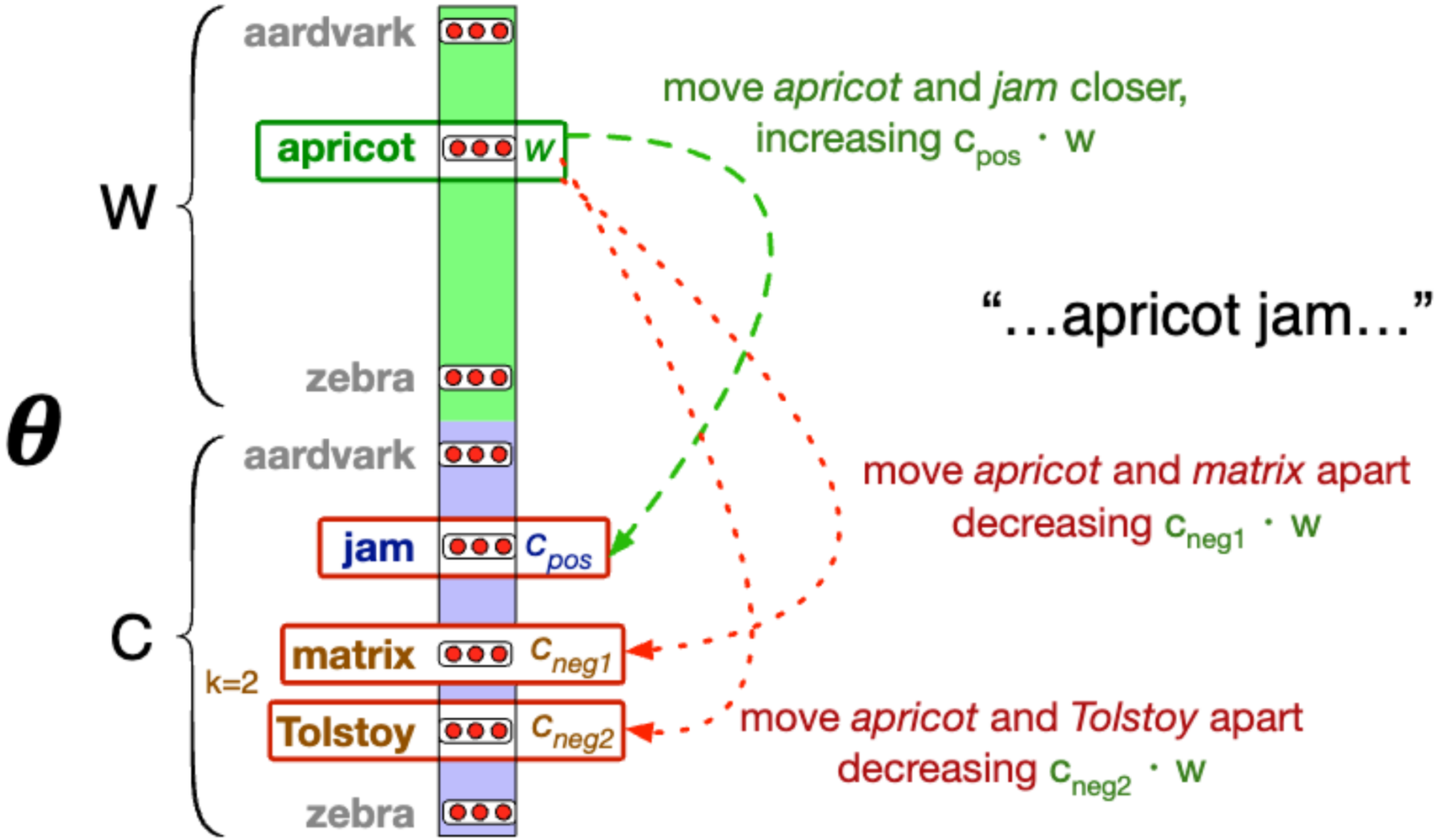
$$= -\left[\log \sigma(\mathbf{w} \cdot \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log \sigma(-\mathbf{w} \cdot \mathbf{c}_{neg_j})\right]$$

Learning the classifier

- How to learn?
  - Stochastic gradient descent!
  - Iterative process
  - Start with randomly initialized weights
    - Update the parameters by computing gradients of the loss w.r..t. parameters
    - Stop when the parameters (or, the loss) do not change much…
- We'll adjust the word weights to
  - make the positive pairs more likely
  - and the negative pairs less likely,
  - over the entire training set.



Intuition of one step of gradient descent



SGD: Derivates

$$L_{CE} = -\left[\log \sigma(\mathbf{w} \cdot \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log \sigma(-\mathbf{w} \cdot \mathbf{c}_{neg_j})\right]$$

3 different parameters

$$\frac{\partial L_{CE}}{\partial \mathbf{c}_{pos}} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{w}$$

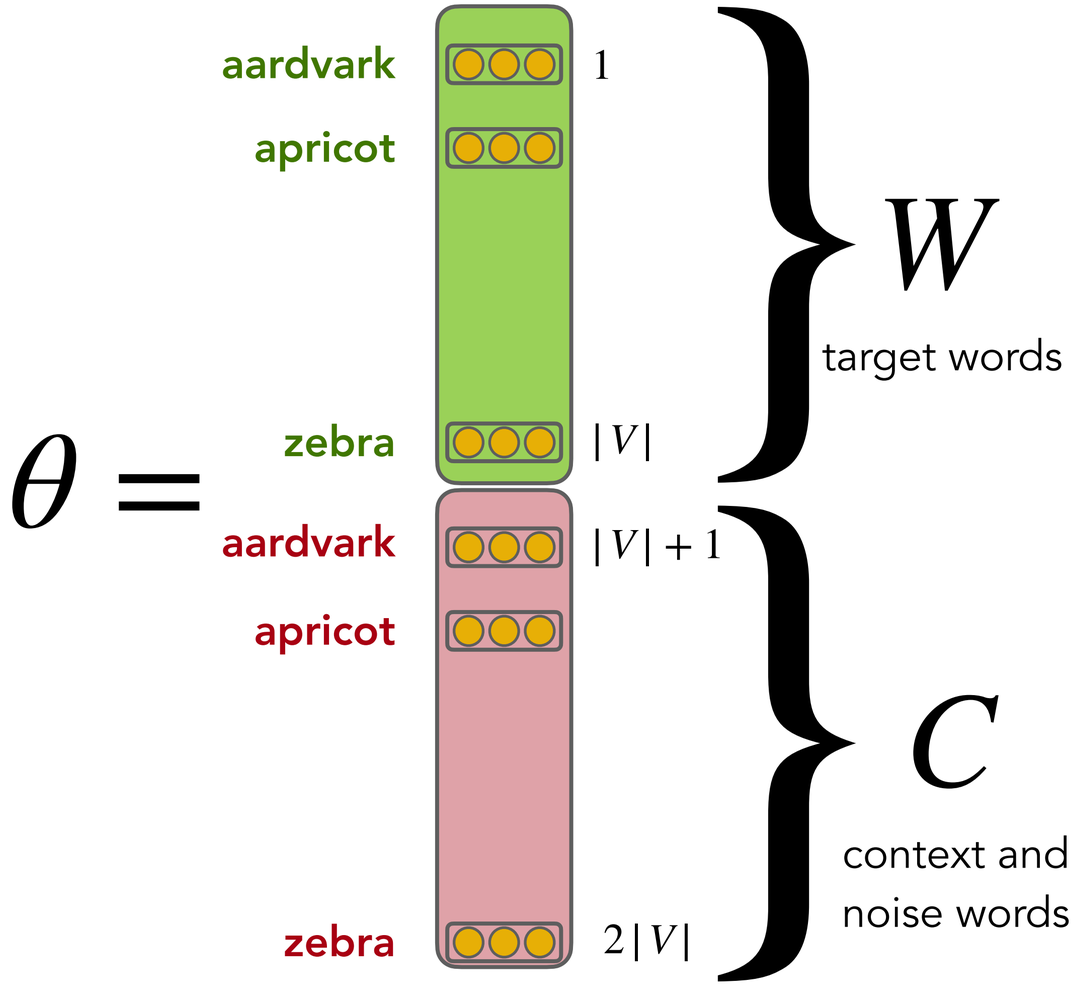
$$\frac{\partial L_{CE}}{\partial \mathbf{c}_{neg_j}} = [\sigma(\mathbf{c}_{neg_j} \cdot \mathbf{w})]\mathbf{w}$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{c}_{pos} + \sum_{j=1}^{K} [\sigma(\mathbf{c}_{neg_j} \cdot \mathbf{w})]\mathbf{c}_{neg_j}$$

Update the parameters by subtracting respective η-weighted gradients

word2vec: Learned Embeddings

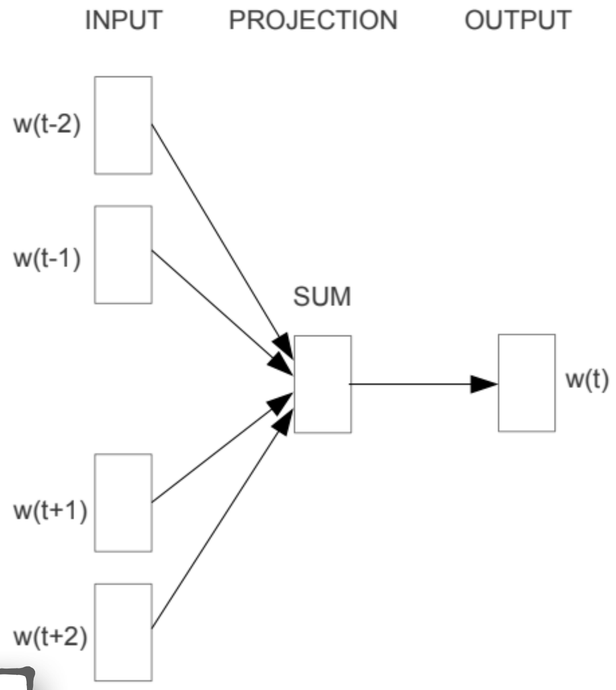
- word2vec learns two sets of embeddings:
  - Target embeddings matrix $\mathbf{W}$
  - Context embedding matrix $\mathbf{C}$
- It's common to just add them together, representing word i as the vector $\mathbf{w}_i + \mathbf{c}_i$



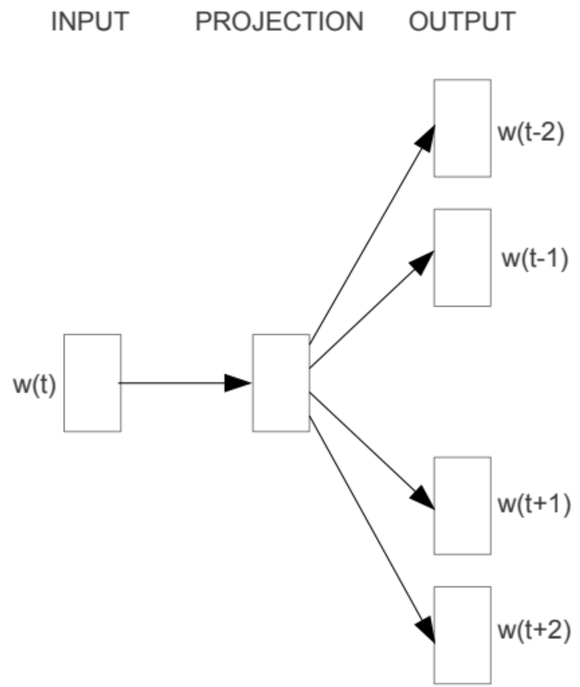
What do the different dimensions in dense embeddings like word2vec represent?

CBOW and Skipgram

- **CBOW**: continuous bag of words - given context, predict which word might be in the target position: $p(w \mid c)$
- **Skip-gram**: given word, predict which words make the best context: $p(c \mid w)$
- CBOW is faster than Skip-gram Why?
- Skip-gram generally works better Why?



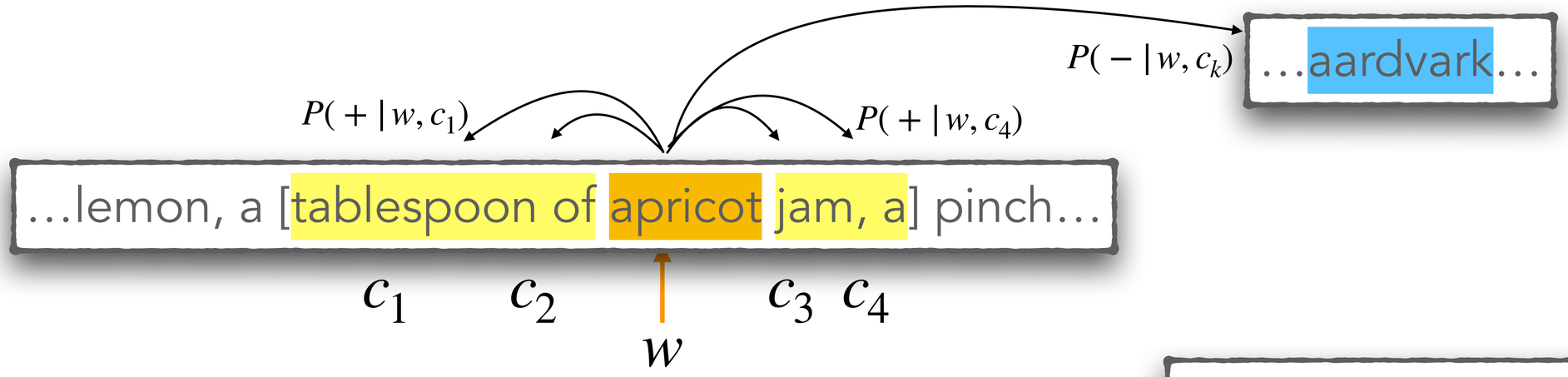
CBOW



Skip-gram

Mikolov et al., 2013. Exploiting Similarities among Languages for Machine Translation.

word2vec: Summary



Can we learn only one embedding per token?

- Start with $2|V|$ random d-dimensional vectors as initial embeddings
- Train a classifier based on embedding similarity
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - Throw away the classifier code and keep the embeddings.

Lecture Outline

- Announcements
- Recap: Sparse and Dense Word Embeddings, word2vec
- GloVe
- Word Embeddings: Properties and Evaluation
- Quiz 2
- Feed-forward Neural Nets

GloVe

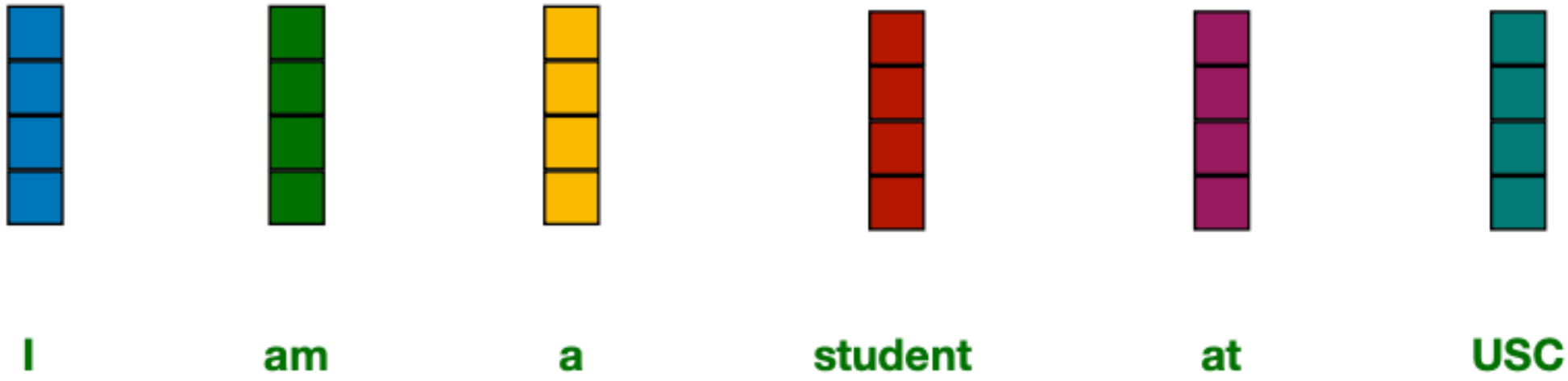
GloVe: Global Vectors

- Another very widely used static embedding model
  - model is based on capturing global corpus statistics (not just a narrow window)
    - based on ratios of probabilities from the word-word co-occurrence matrix,
    - intuitions of count-based models like PPMI
- Builds on matrix factorization
  - Idea: store most of the important information in a fixed, small number of dimensions: a dense vector
  - Goal: Create a low-dimensional matrix for the embedding while minimizing reconstruction loss (error in going from low to high dimension)
- Fast training, scalable to huge corpora

Pennington et al., 2014

Applying Word Embeddings to Classification Tasks

- We need a single feature vector to feed into ML classifiers



$$\mathbf{x_{doc}} = pool_{i \in |len(\mathbf{doc})|} \mathbf{w_i}; \quad \mathbf{x}, \mathbf{w} \in \mathbb{R}^d$$

Pooling may happen via dimension-specific averaging or max operations

Lecture Outline

- Announcements
- Recap: Sparse and Dense Word Embeddings, word2vec
- GloVe
- Word Embeddings: Properties and Evaluation
- Quiz 2
- Feed-forward Neural Nets

Properties and Evaluation of Word Embeddings

Effects of Context Window Size

Both sparse and dense vectors

- Small windows (C= +/- 2) : nearest words are syntactically similar words in same taxonomy (semantics and syntax)
  - Hogwarts nearest neighbors are other fictional schools
    - Sunnydale, Evernight, Blandings

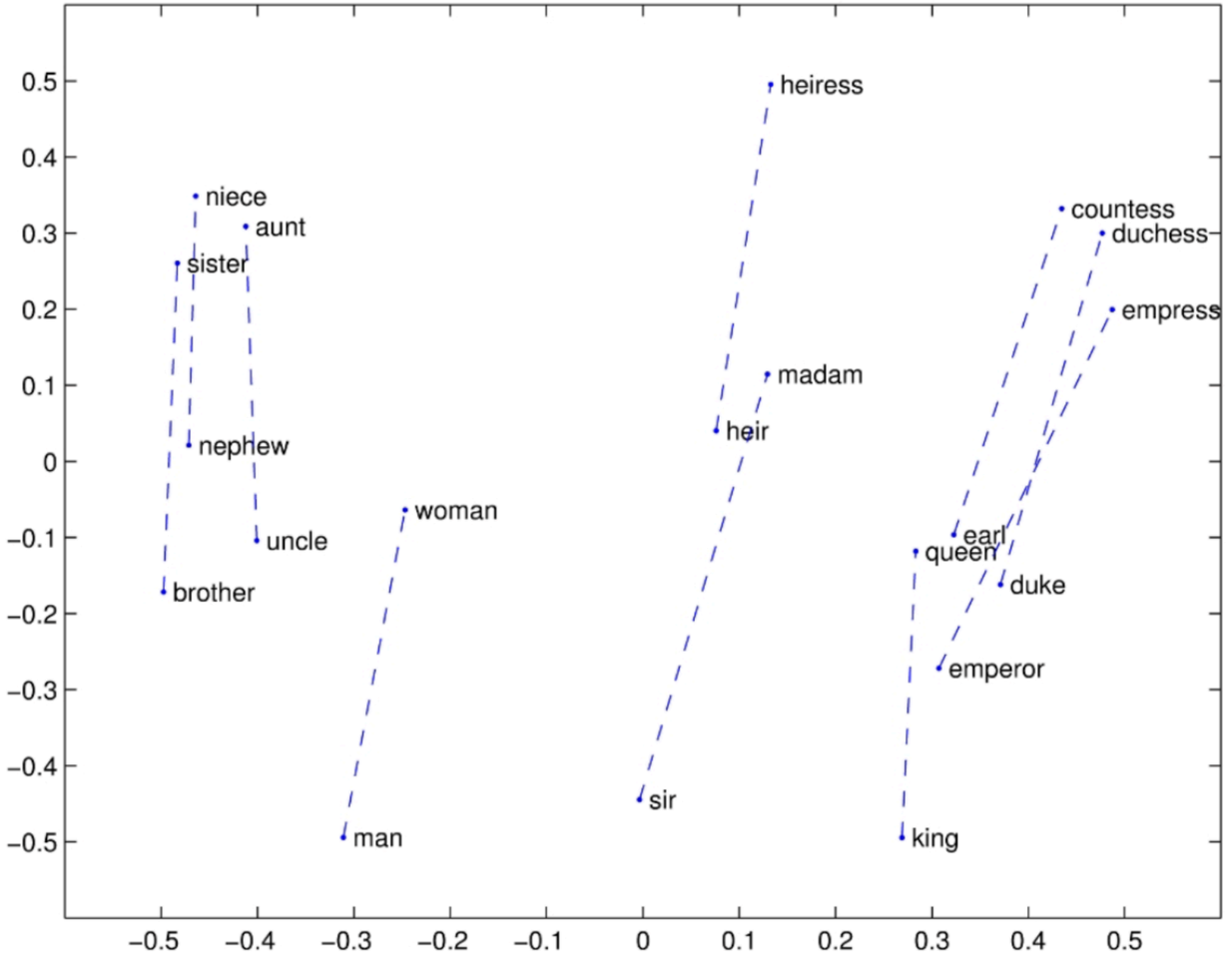
- Large windows (C= +/- 5) : nearest words are related words in same topic
  - Hogwarts' nearest neighbors are in the Harry Potter world:
    - Dumbledore, half-blood, Malfoy

Why?

Evaluation: via Analogy Relations

- Relational properties of the GloVe vector space, shown by projecting vectors onto two dimensions
- $\mathbf{w}_{king} - \mathbf{w}_{man} + \mathbf{w}_{woman}$ is similar to $\mathbf{w}_{queen}$

- Caveats: Only works for frequent words, small distances and certain relations (relating countries to capitals, or parts of speech), but not others
  - Understanding analogy is an open area of research
- Better Evaluation: Performance on Downstream Tasks (Extrinsic Evaluation)



Embeddings reflect cultural bias!

Offensive Content Warning

- Ask "Paris : France :: Tokyo : x"
  - x = Japan
- Ask "father : doctor :: mother : x"
  - x = nurse
- Ask "man : computer programmer :: woman : x"
  - x = homemaker

Allocational Harms

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

Bolukbasi et al., NeurIPS 2016. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings."

Embeddings as a tool to study cultural bias!

- Compute a gender or ethnic bias for each adjective: e.g., how much closer the adjective is to "woman" synonyms than "man" synonyms, or names of particular ethnicities
  - Embeddings for competence adjective (smart, wise, brilliant, resourceful, thoughtful, logical) are biased toward men, a bias slowly decreasing 1960-1990
  - Embeddings for dehumanizing adjectives (barbaric, monstrous, bizarre) were biased toward Asians in the 1930s, bias decreasing over the 20th century
- These match the results of old surveys done in the 1930s

1910	1950	1990
Irresponsible	Disorganized	Inhibited
Envious	Outrageous	Passive
Barbaric	Pompous	Dissolute
Aggressive	Unstable	Haughty
Transparent	Effeminate	Complacent
Monstrous	Unprincipled	Forceful
Hateful	Venomous	Fixed
Cruel	Disobedient	Active
Greedy	Predatory	Sensitive
Bizarre	Boisterous	Hearty

Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. Proceedings of the National Academy of Sciences 115(16), E3635–E3644.

Representational Harms

Quiz 2
(Password: embedding)
Time: 15 mins

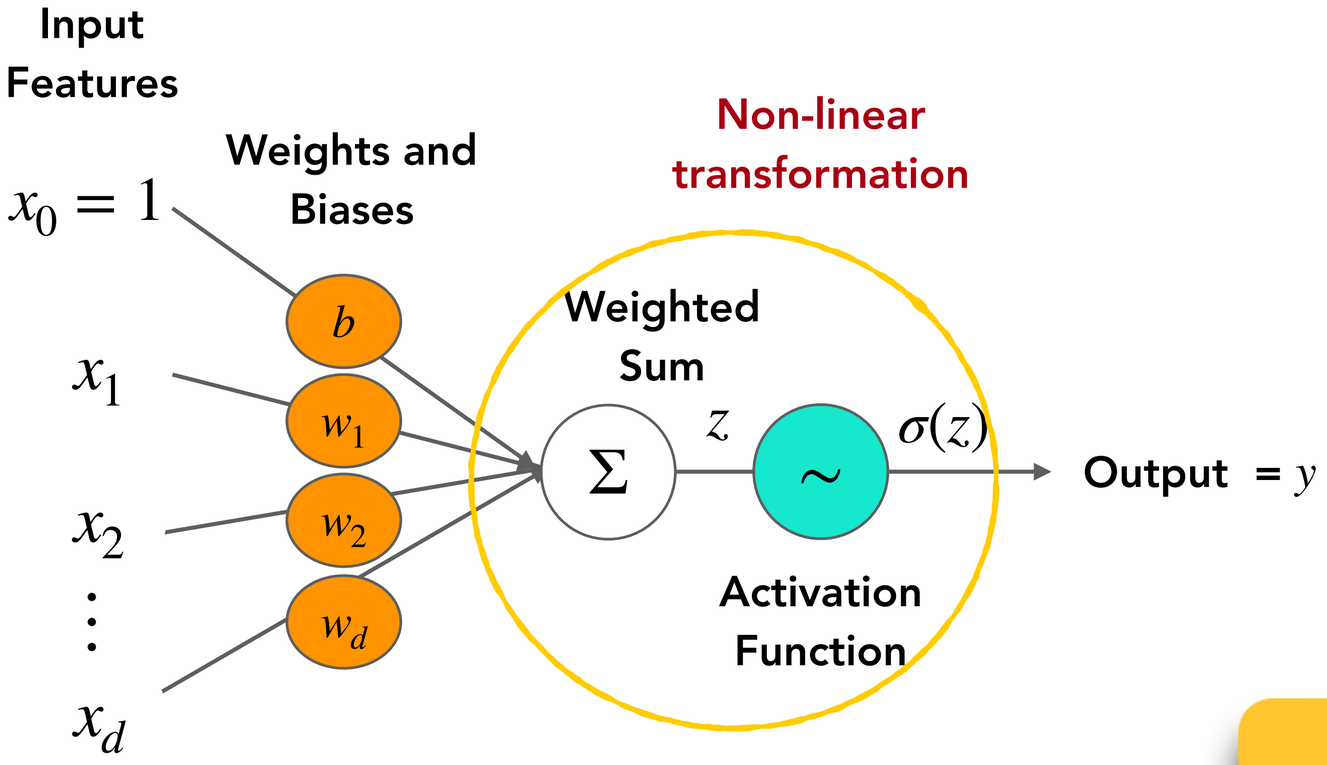
Lecture Outline

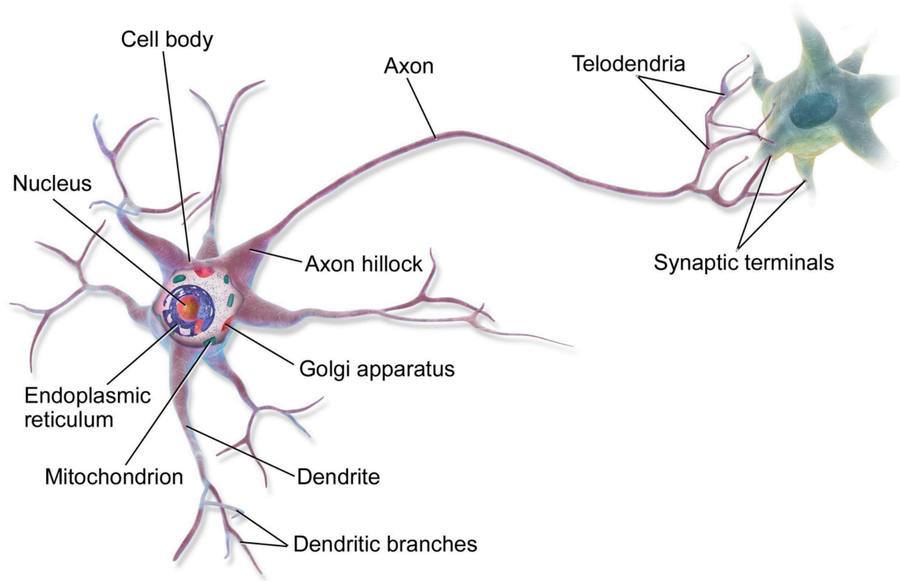
- Announcements
- Recap: Sparse and Dense Word Embeddings, word2vec
- GloVe
- Word Embeddings: Properties and Evaluation
- Quiz 2
- Feed-forward Neural Nets
  - Language Models

Feed-Forward Neural Networks

Neural Network Unit

Logistic Regression is a very simple neural network

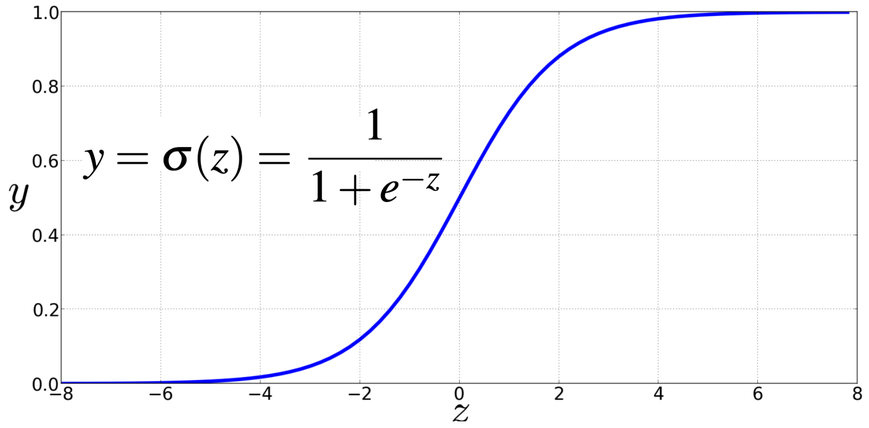


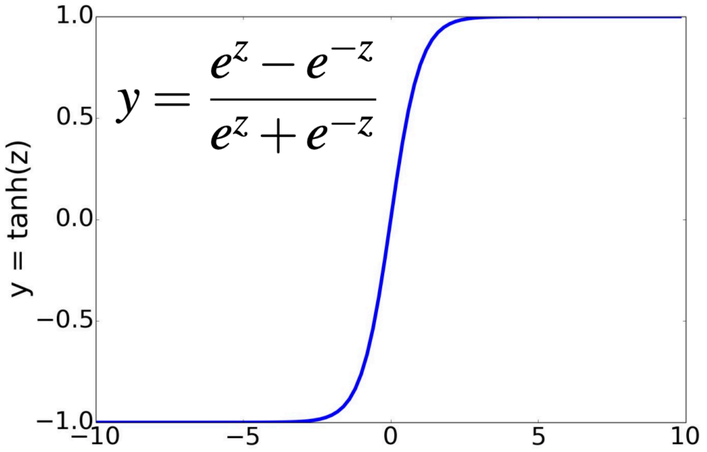


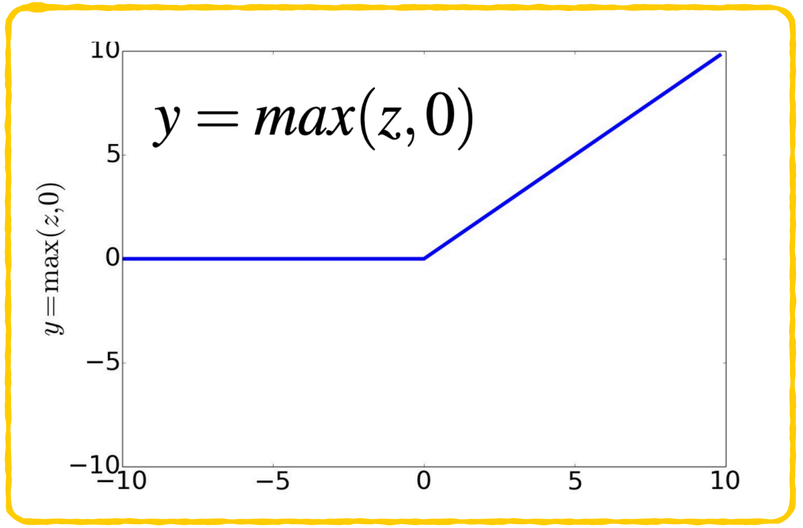
Resembles a neuron in the brain!

Non-Linear Activation Functions

Most common!







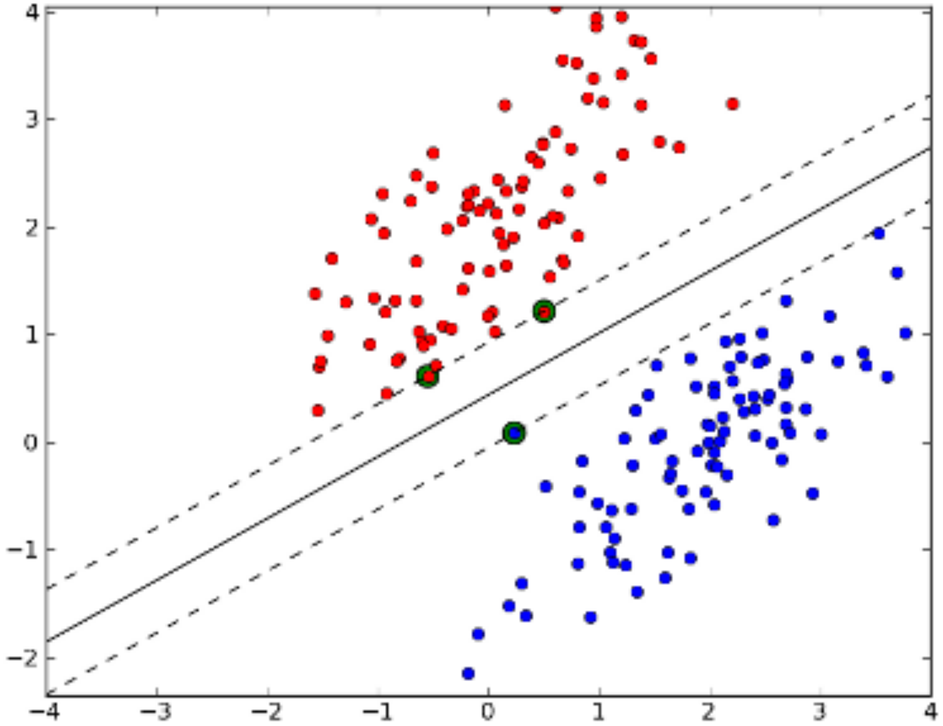
sigmoid

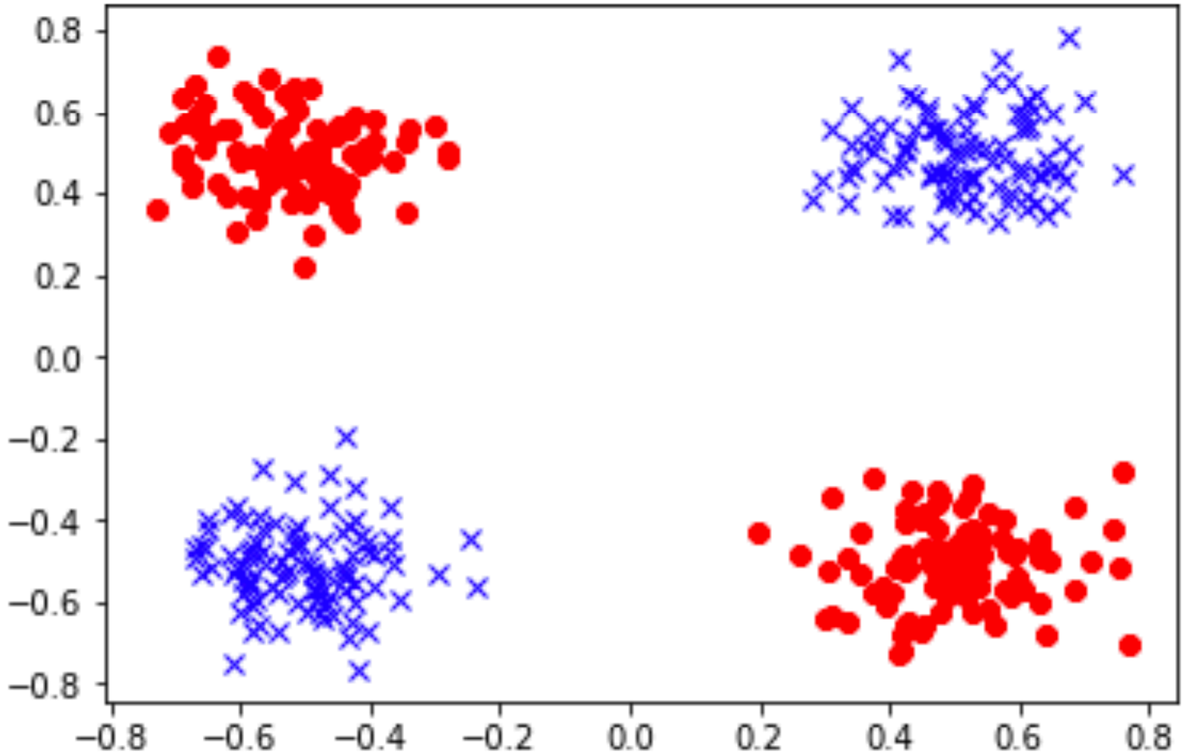
tanh

relu (Rectified Linear Unit)

The key ingredient of a neural network is the non-linear activation function

Linear vs. Non-linear Functions

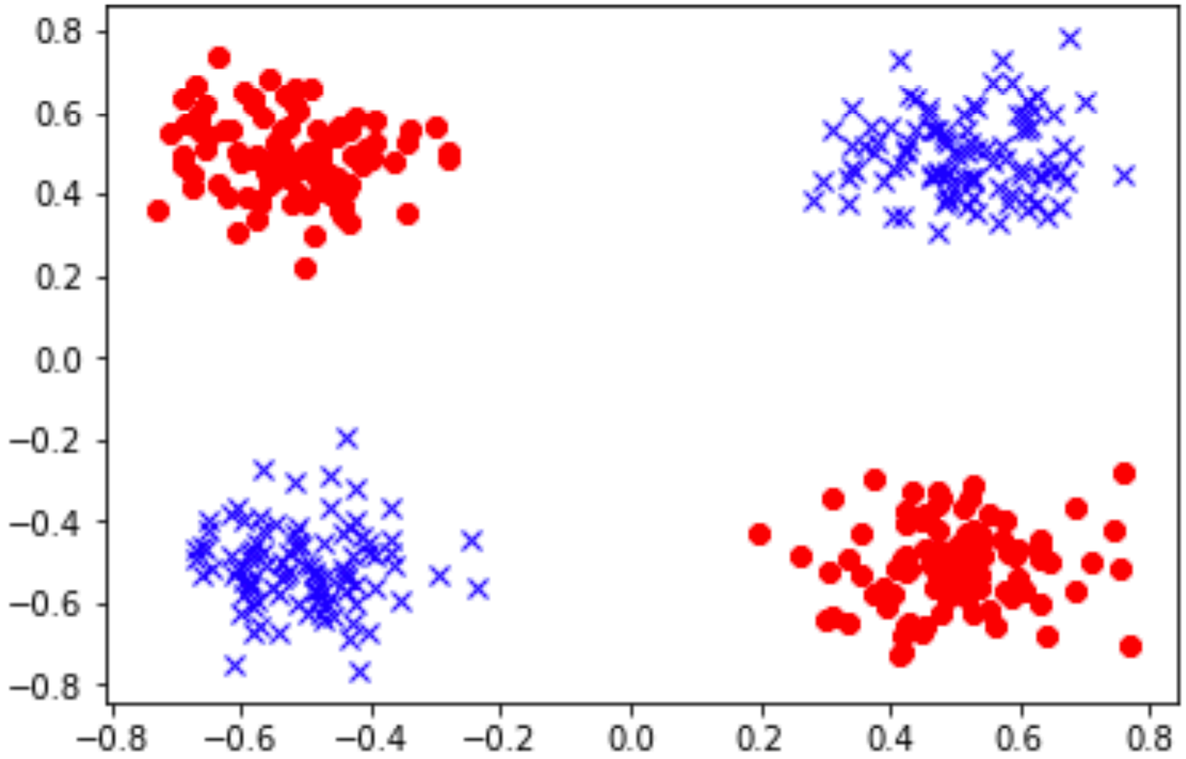




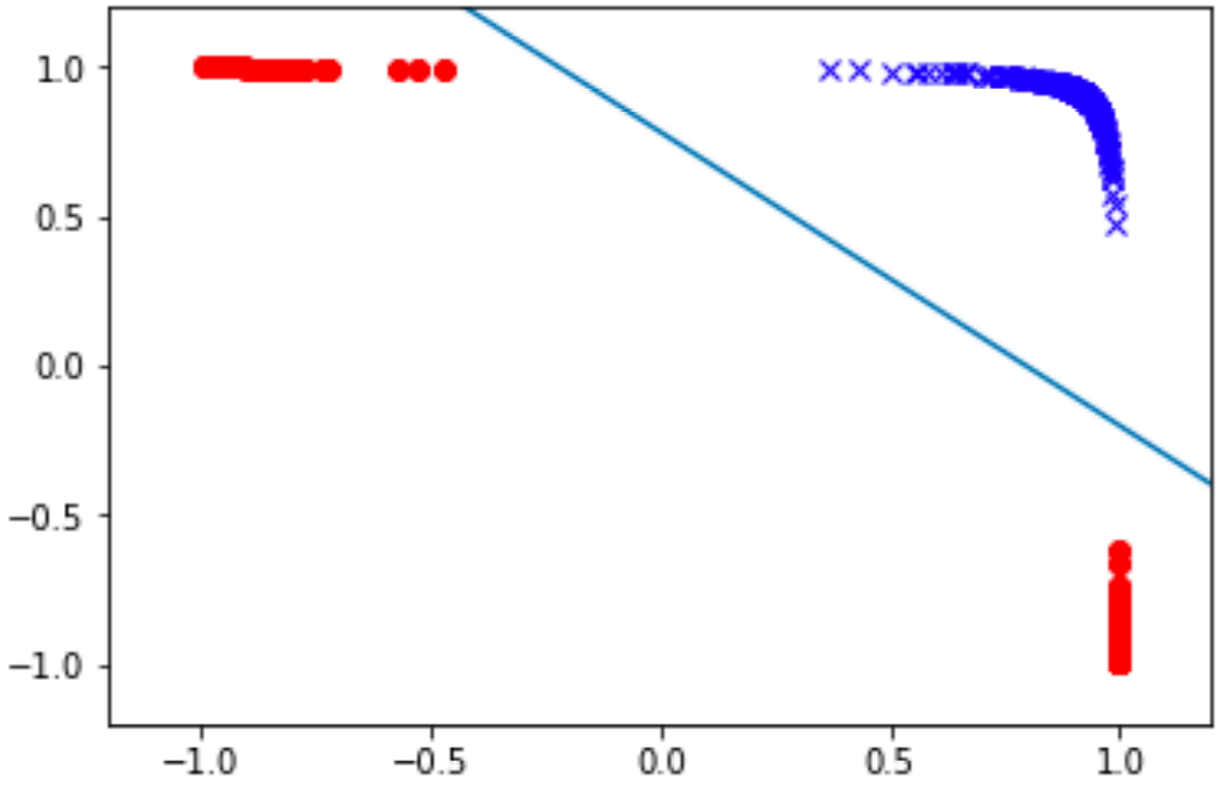
Linearly inseparable

Power of non-linearity

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



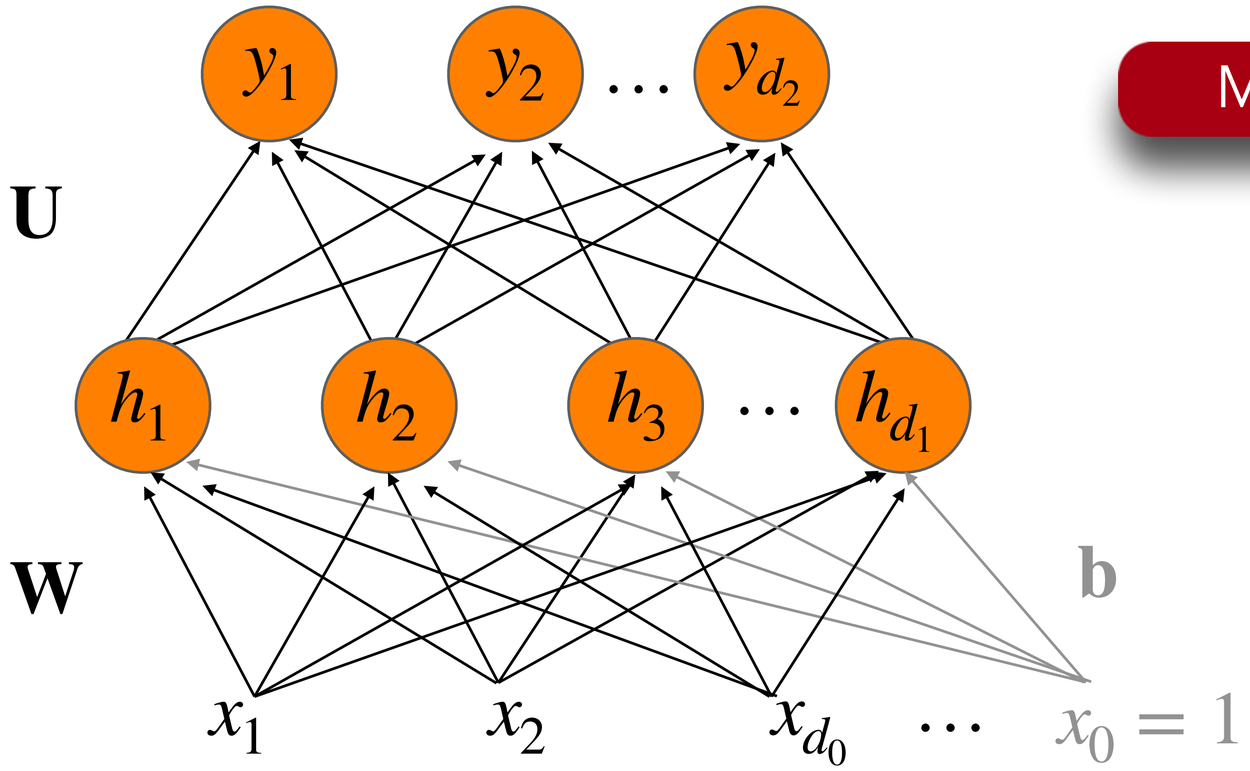
After a $\tanh(\cdot)$ transformation:



Feedforward Neural Nets

Multilayer Perceptron

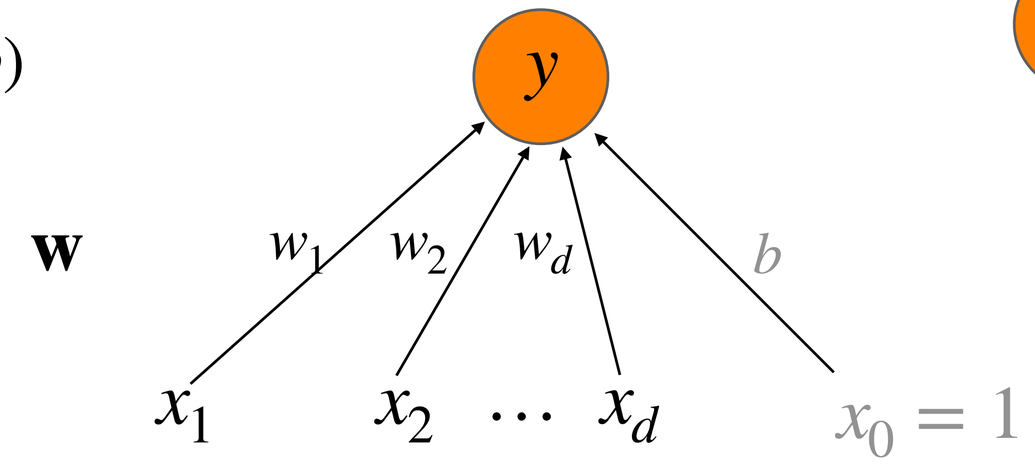
Technically, can learn any function!



Let's break it down by revisiting our logistic regression model

Binary Logistic Regression

Output layer: $y = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$



Input layer: vector $\mathbf{x}$

Weighted sum of all incoming, followed by a non-linear activation

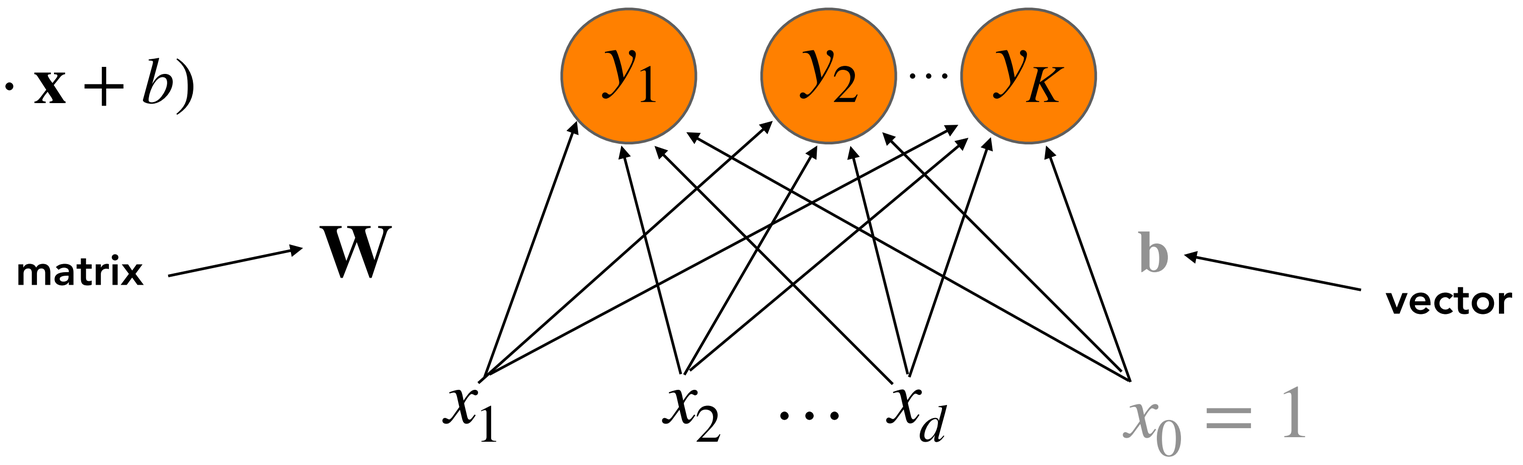
1-layer Network

Don't count the input layer in counting layers!

Multinomial Logistic Regression

Output layer: $\mathbf{y} = \text{softmax}(\mathbf{w} \cdot \mathbf{x} + b)$

Input layer: vector $\mathbf{x}$



1-layer Network

Fully connected single layer network

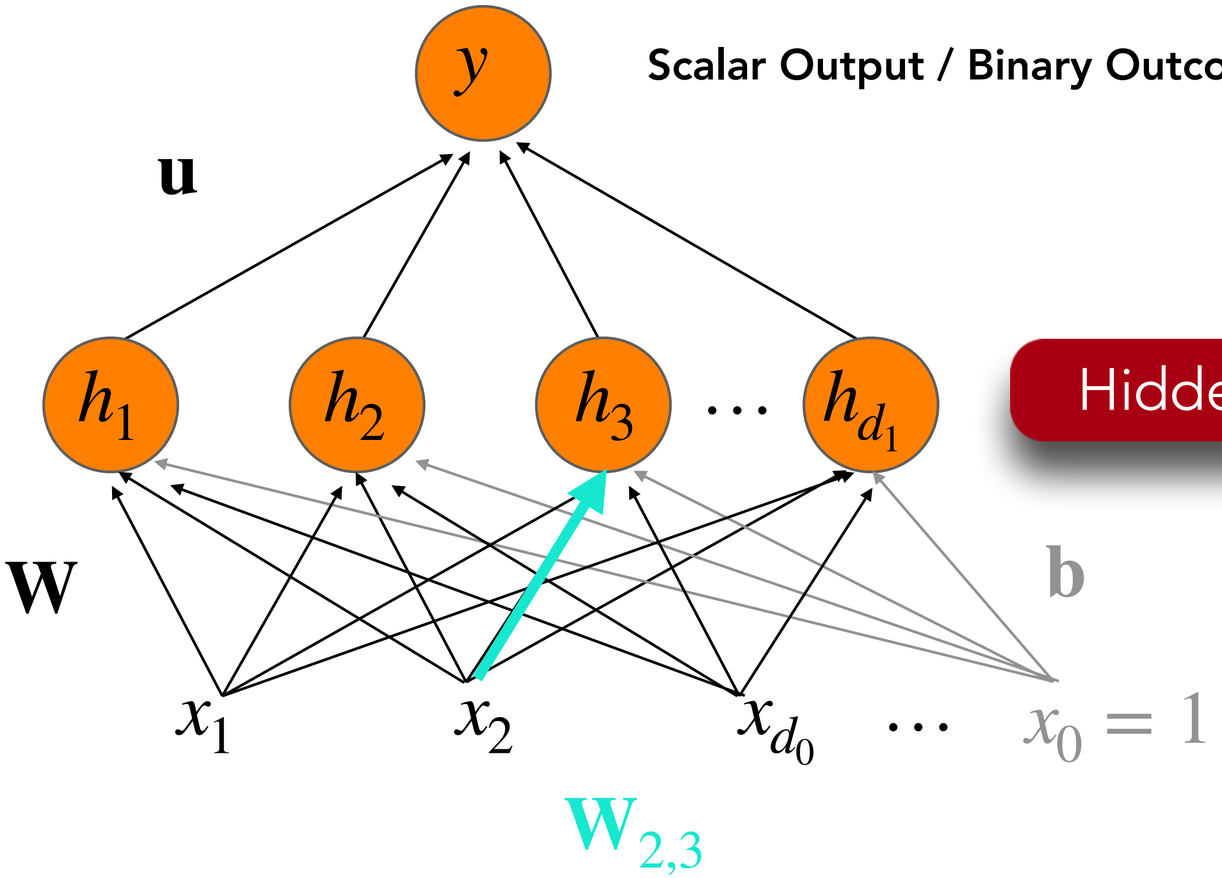
Two-layer Feedforward Network

Output layer: $y = \sigma(\mathbf{uh})$

Hidden layer: $\mathbf{h} = g(\mathbf{Wx} + \mathbf{b})$

Usually ReLU or tanh

Input layer: vector $\mathbf{x}$



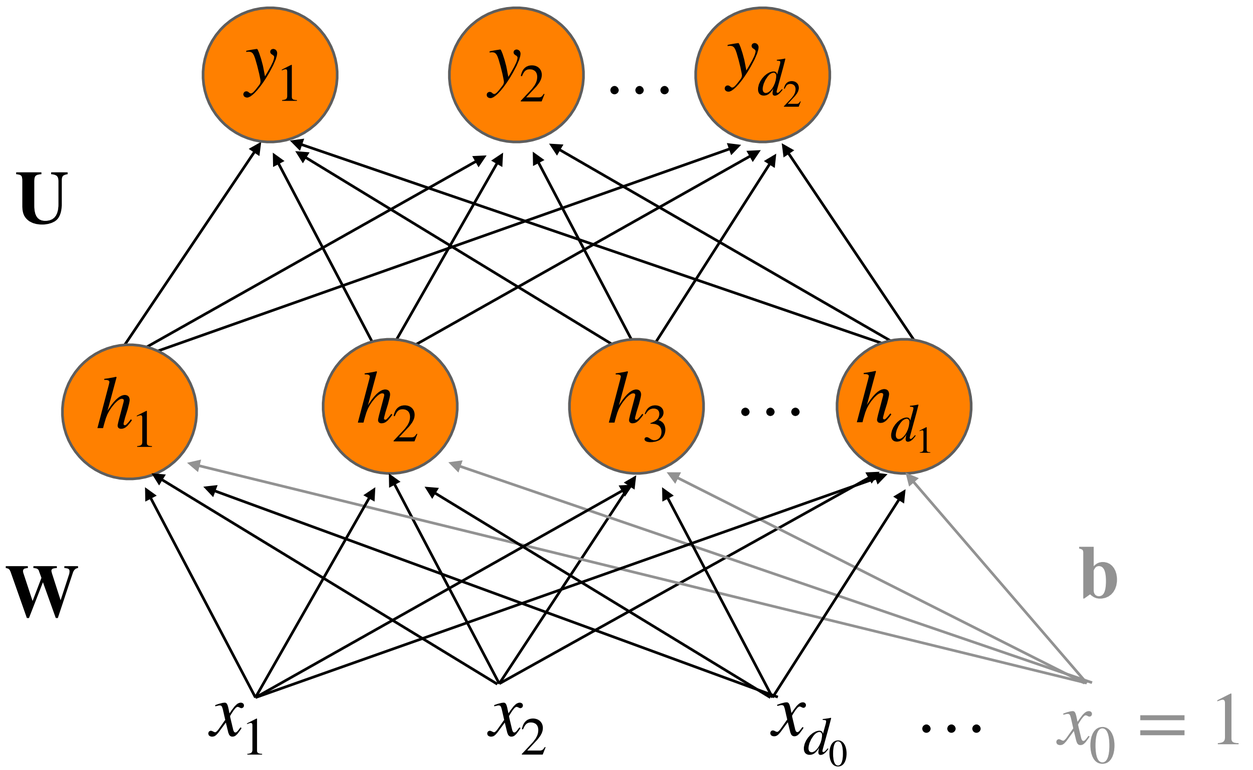
Two-layer Feedforward Network with Softmax Output

Output layer: $\mathbf{y} = \text{softmax}(\mathbf{U} \cdot \mathbf{h})$

Hidden layer: $\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$

Usually ReLU or tanh

Input layer: vector $\mathbf{x}$



What is $\mathbf{y}$?

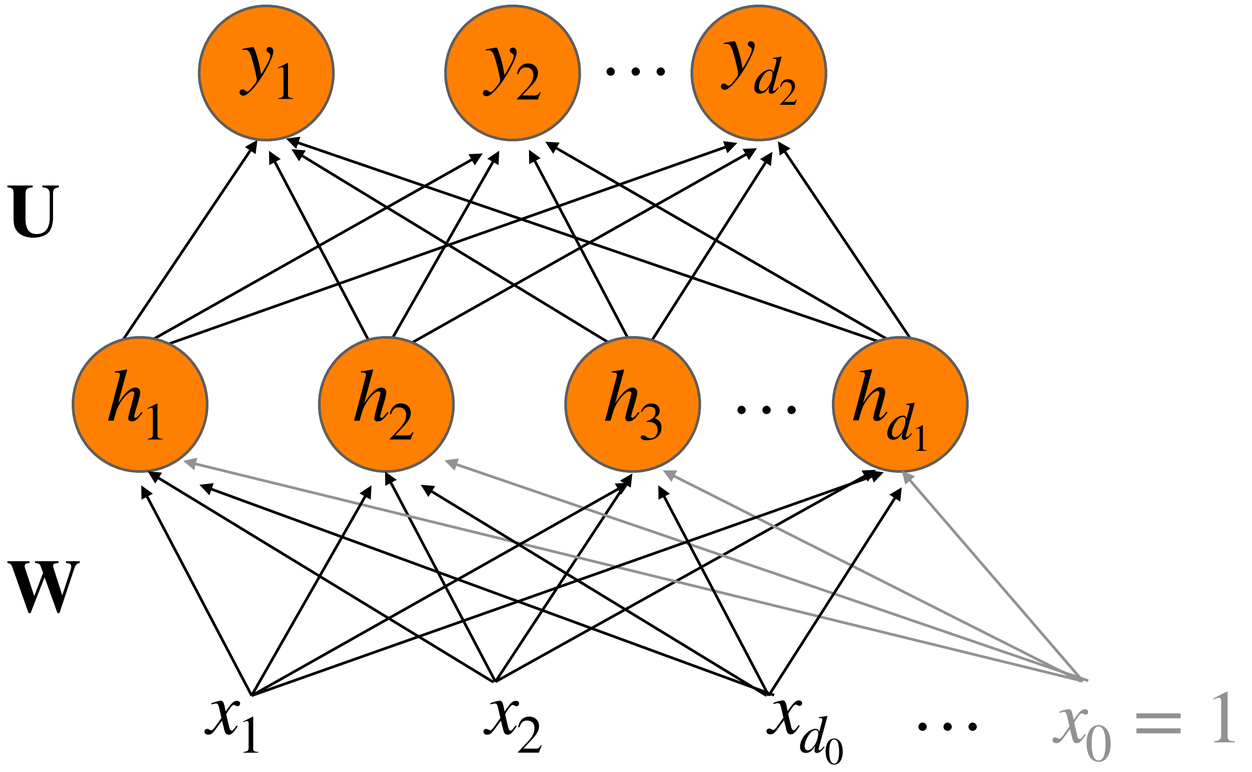
Two-layer FFNN: Notation

Output layer: $\mathbf{y} = \text{softmax}(\mathbf{U} \cdot \mathbf{h})$

Hidden layer: $\mathbf{h} = g(\mathbf{W}\mathbf{x}) = g\left(\sum_{i=0}^{d_0} \mathbf{W}_{ji}\mathbf{x}_i\right)$

Usually ReLU or $\tanh$

Input layer: vector $\mathbf{x}$



We usually drop the $\mathbf{b}$ and add one dimension to the $\mathbf{W}$ matrix

Concluding Thoughts

Next Class:

- More on Feedforward neural nets
- Backpropagation