Lecture 6:
word2vec contd. + Feedforward Nets

Instructor: Swabha Swayamdipta

USC CSCI 444 NLP

Sep 17, 2025

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

Announcements + Logistics

- Teams!
  - Pinru Wang + YANG, Yihan: The Emoji Interpreter
  - Henry Chen + Jiang, Mo: Redirecting the latent space of unsafe prompts for pretrained LLMs
  - Alp Inangu + Gamez Hernandez, Valeria: Spike_Trains_to_Speech (Spike2vec)
  - Yuhui Zhang + Moumtzis, Alexios: Green Verifier
  - Avi Choudhary + Mahamuni, Aditya: Using NLP to analyze semiconductor data (could be changed)
  - Shane Yokota + Velayuthan, Vish: DSP Guitar Pedal
  - Kayal Bhatia + Panjwani, Naina: FirstLight - Early Mental Health Detection
- Today: HW1 due. This week: HW2 release
- Next week: Quiz 2 + Project Proposal:
  - ~1-page proposal (using the *CL paper submission template) for their project.
    - state and motivate the problem / **task definition** (preferably with example inputs and expected outputs),
    - situate the problem within **related work**,
      - Related work: publications, start by looking in the ACL anthology
      - References do not count towards page limit, but please follow the correct format
    - **state a hypothesis** to be verified and how to verify it (evaluation framework), and
    - provide a **brief description of the approach** (such as proposed models and baselines).
  - Think about the five key ingredients of supervised learning: data, model, loss function, optimization algorithm and inference / evaluation

Lecture Outline

- Recap: word2vec
- GloVe
- Evaluating Word Embeddings
- Feedforward Neural Nets
- Feedforward Net Language Models

Recap: 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

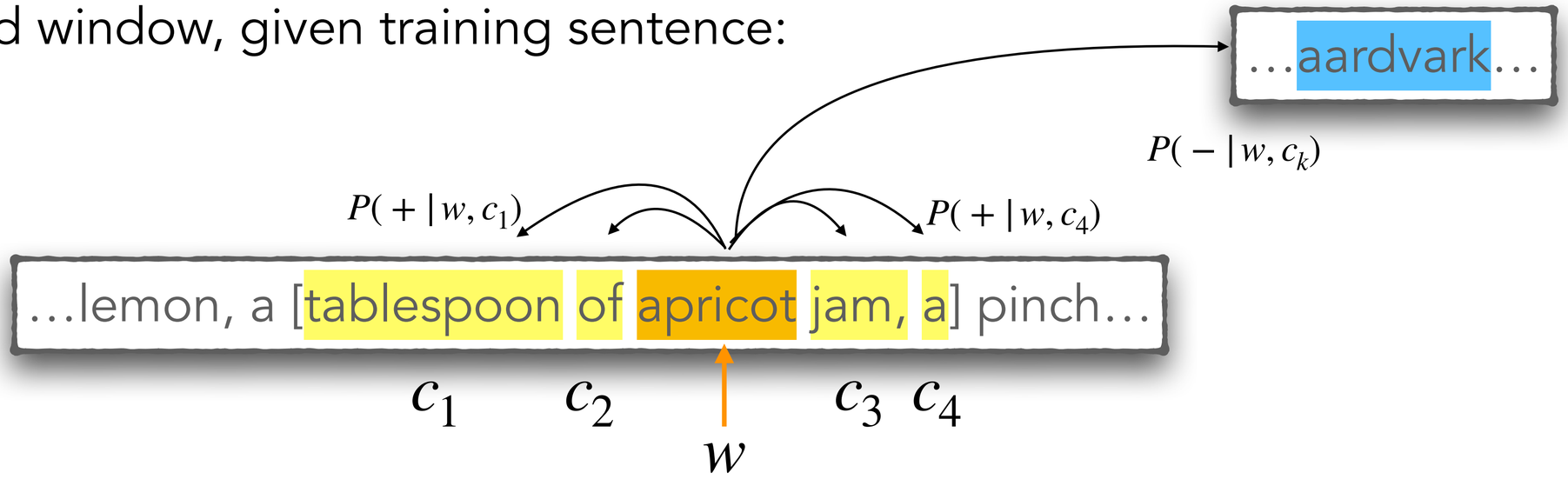
What happens to the problem of polysemy?

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: Goal

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



Goal: train a classifier that is given a candidate (word, context) pair:

(apricot, jam)

(apricot, aardvark)

…

And assigns each pair a probability:

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

Turning dot products into probabilities

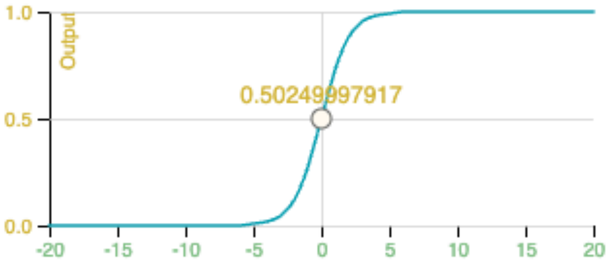
Similarity:

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

Sigmoid

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

Turn into a probability using the sigmoid function:



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

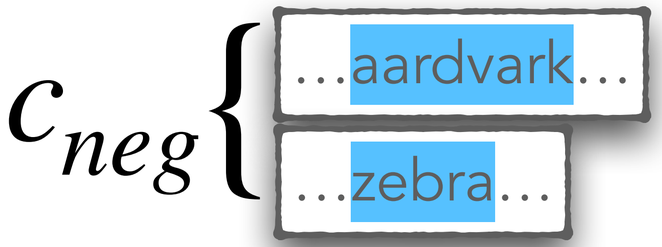
$$P(-|w, c) = 1 - P(+|w, c)$$

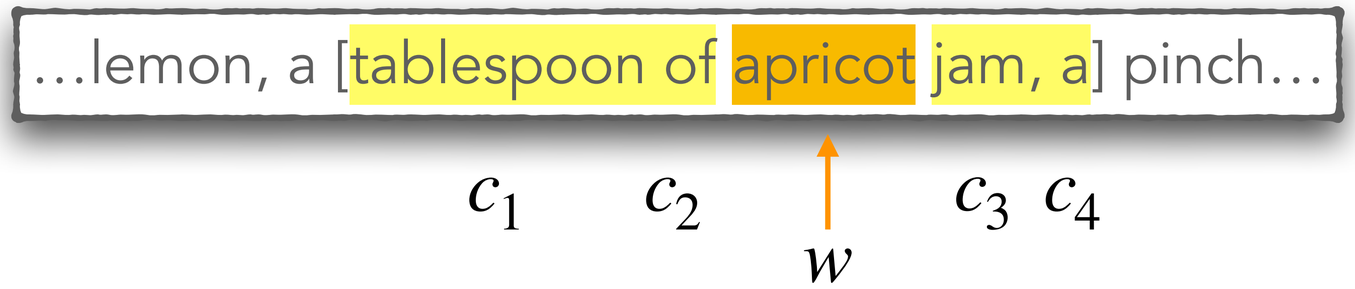
$$= \sigma(-\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1 + \exp(\mathbf{c} \cdot \mathbf{w})}$$

Logistic Regression!

Word2vec: Training Data

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





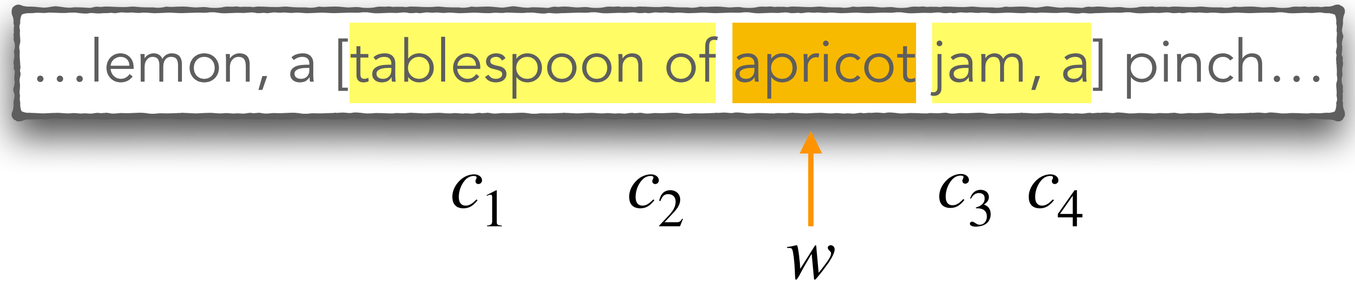
Negative examples

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

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

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

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

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

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

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

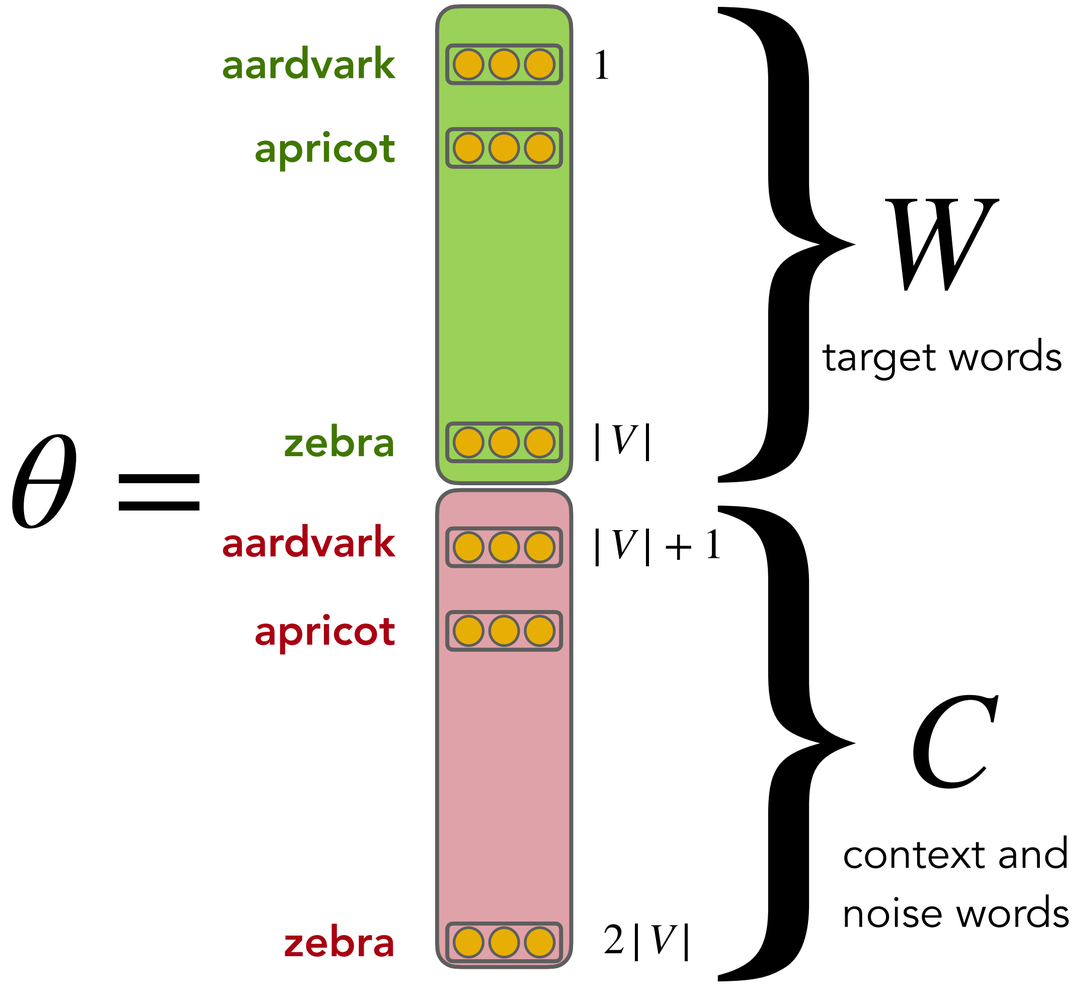
$$\frac{\partial L_{CE}}{\partial \mathbf{c}_{neg_j}} = [\sigma(\mathbf{c}_{neg_j} \cdot \mathbf{w})]\mathbf{w}$$

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

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 $\mathbf{w}_i + \mathbf{c}_i$



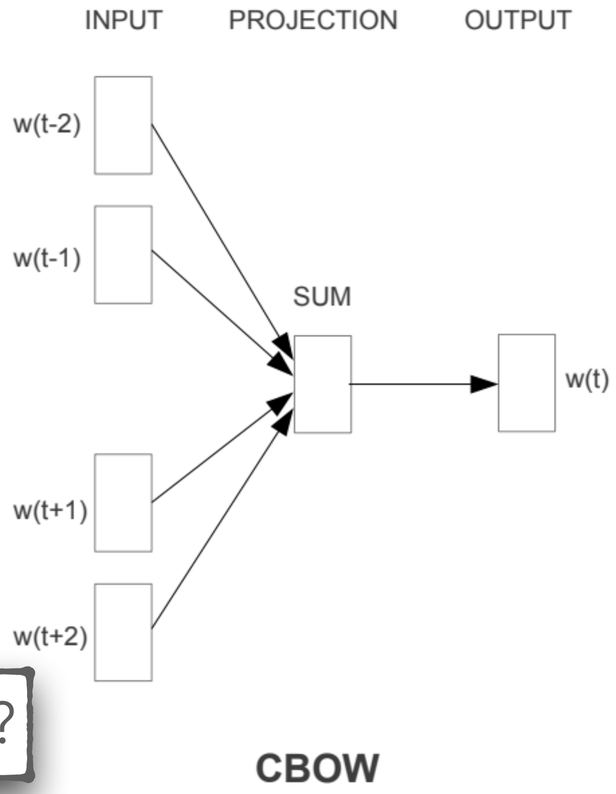
CBOW and Skipgram

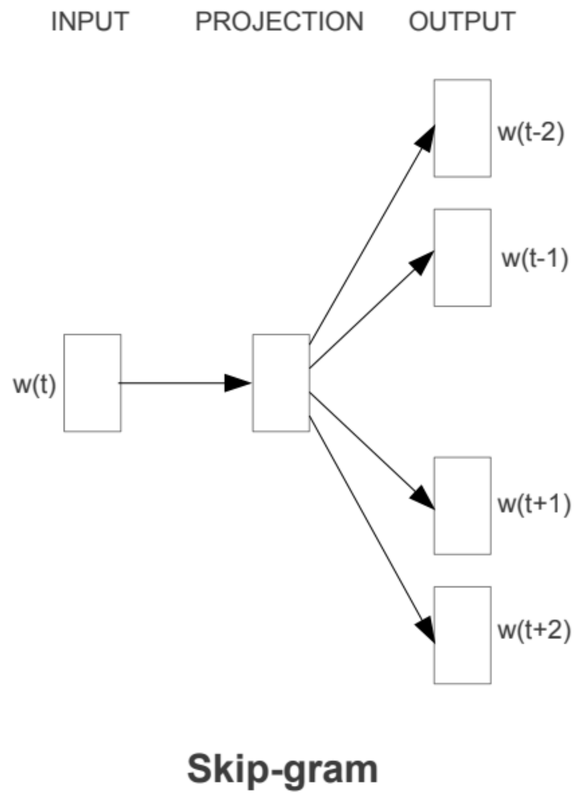
- **CBOW**: continuous bag of words - given context, predict which word might be in the target position
- **Skip-gram**: given word, predict which words make the best context
- CBOW is faster than Skip-gram

Why?

- Skip-gram generally works better

Why?





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

Effects of Context Window Size

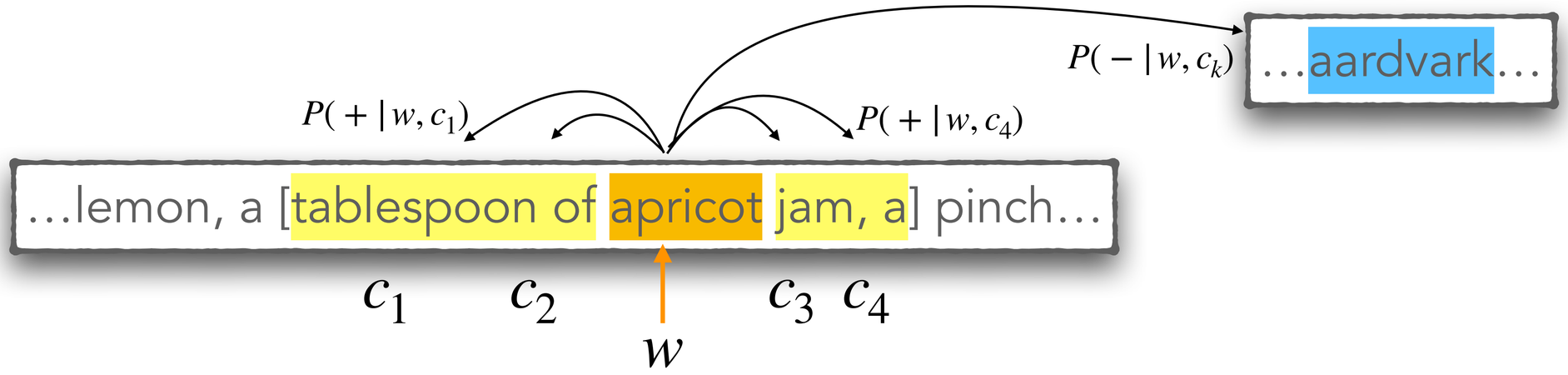
Both sparse and dense vectors

- 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?

word2vec: Summary



- 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.

GloVe: Global Vectors

- Another very widely used static embedding model
  - model is based on capturing global corpus statistics
    - 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

Lecture Outline

- Recap: word2vec
- GloVe
- Evaluating Word Embeddings
- Feedforward Neural Nets
- Feedforward Net Language Models

Evaluating Word Embeddings

Evaluating Word Embeddings

- Related to general evaluation in NLP:
  - Intrinsic vs. extrinsic
  - Intrinsic:
    - Evaluation on a specific/intermediate subtask
    - Fast to compute
    - Helps to understand that system
    - Not clear if it's helpful unless correlation to real task is established
  - Extrinsic:
    - Evaluation on a real task
    - Can take a long time to compute accuracy
    - Unclear if the subsystem is the problem or its interaction or other subsystems
    - If replacing exactly one subsystem with another improves accuracy → Winning!

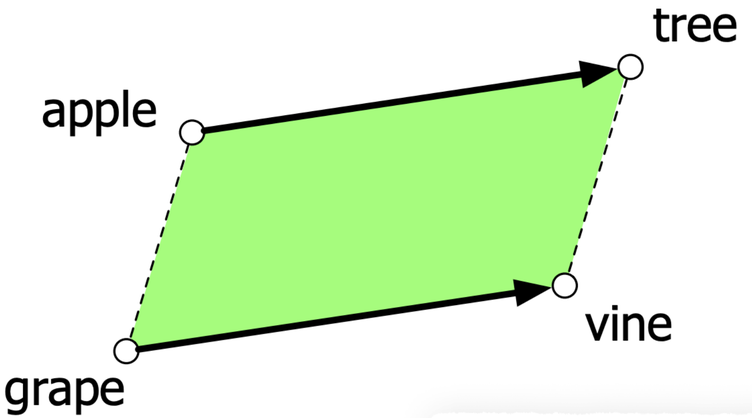
Intrinsic Evaluation: Analogy Relations

Both sparse and dense vectors

- The classic parallelogram model of analogical reasoning
- Word analogy problem:
  - "Apple is to tree as grape is to …"

Add $(\mathbf{w}_{tree} - \mathbf{w}_{apple})$ to $\mathbf{w}_{grape}$ …

Should result in $\mathbf{w}_{vine}$



Rumelhart and Abrahamson, 1973

For a problem $a : a^* :: b : b^*$, the parallelogram method is:

$$\hat{b}^* = \arg\max_{\mathbf{w}} sim(\mathbf{w}, \mathbf{b} - \mathbf{a} + \mathbf{a}^*)$$

Maximize similarity = minimize distance

Problem: What if the information is there but is not linear?

Intrinsic Evaluation: Meaning Similarity

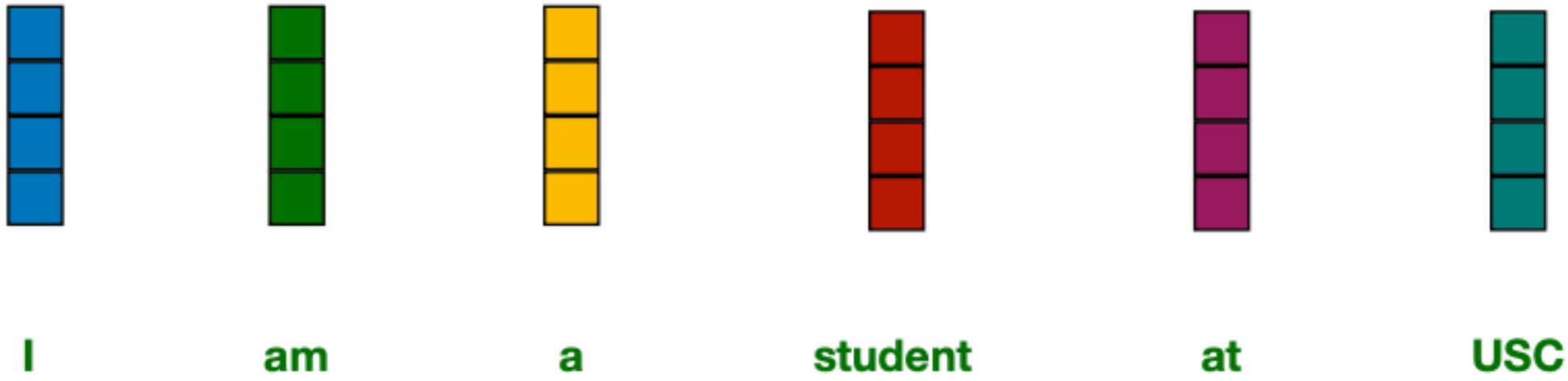
- Word vector distances and their correlation with human judgments

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW†	6B	57.2	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	**75.9**	**83.6**	**82.9**	**59.6**	**47.8**
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

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

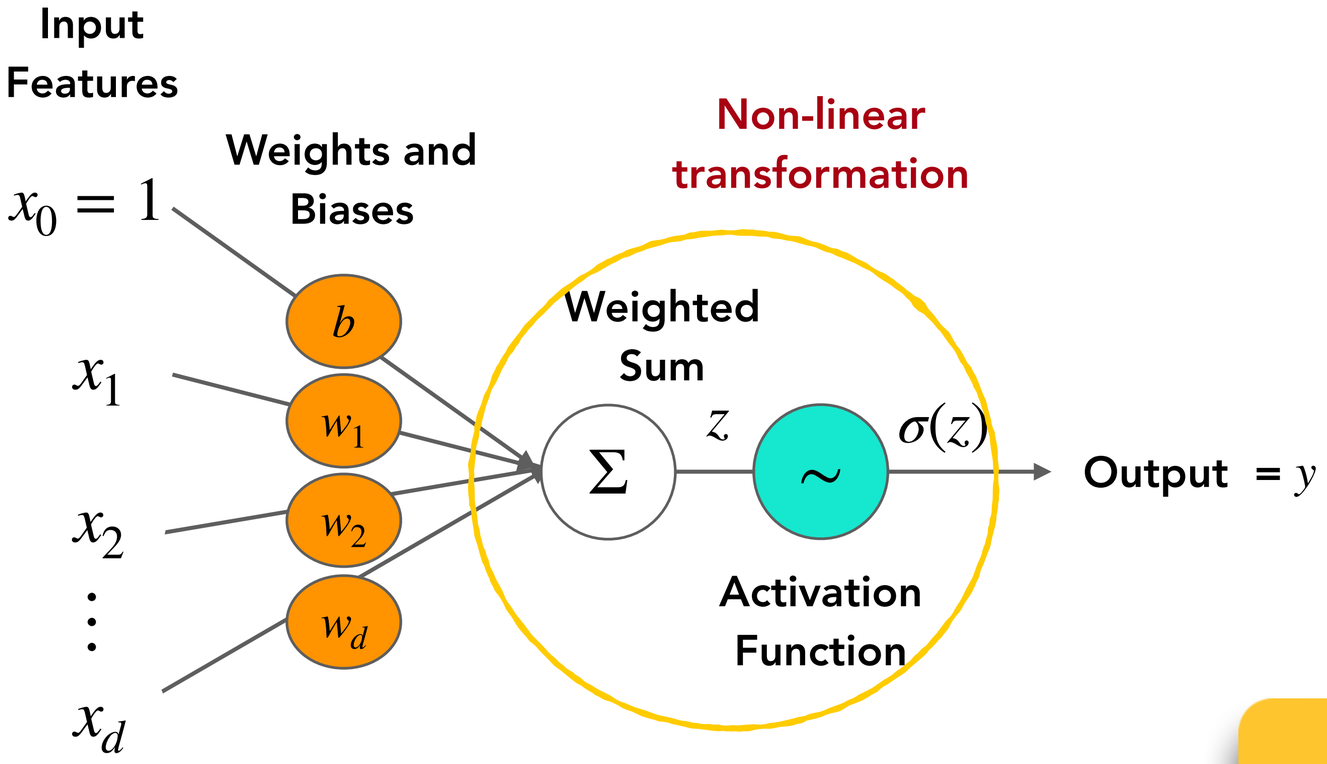
Lecture Outline

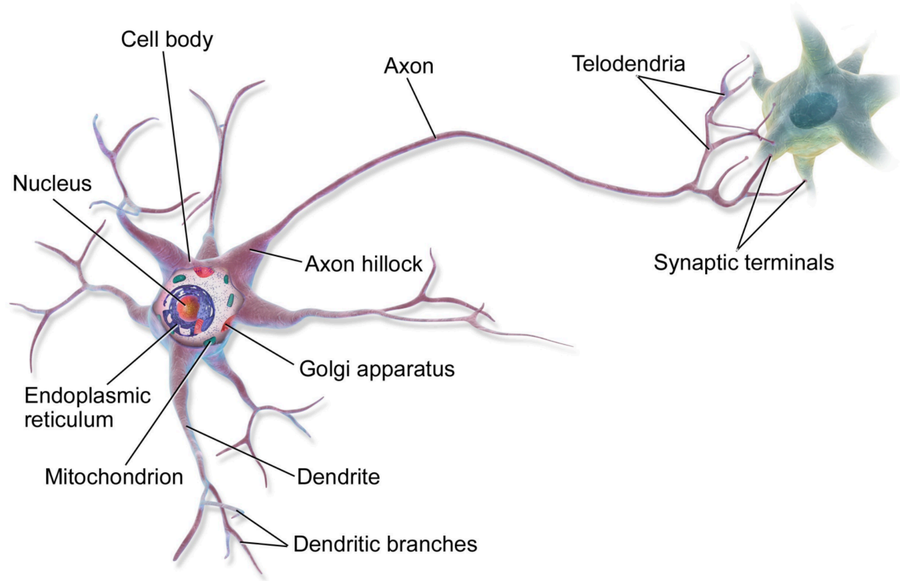
- Recap: word2vec
- GloVe
- Evaluating Word Embeddings
- Feedforward Neural Nets
- Feedforward Net Language Models

Feed-Forward Neural Networks

Neural Network Unit

Logistic Regression is a very simple neural network

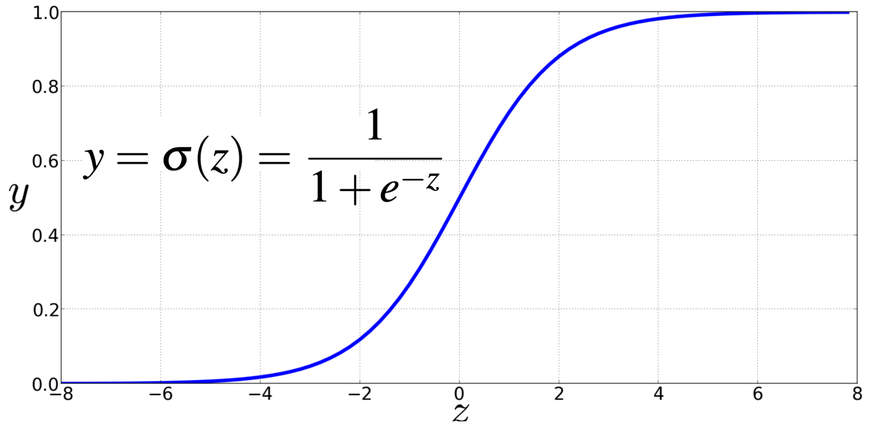




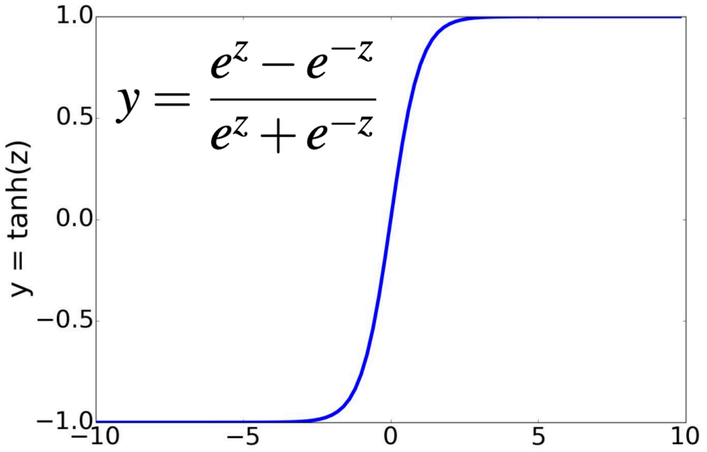
Resembles a neuron in the brain!

Non-Linear Activation Functions

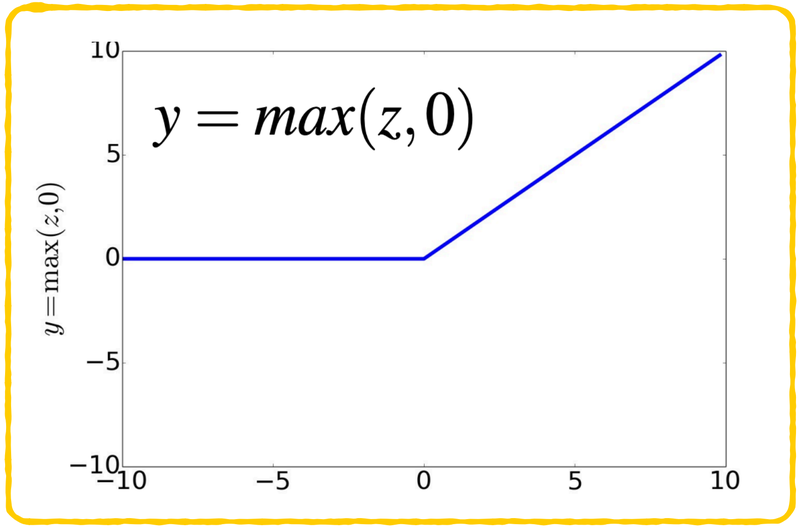
Most common!



sigmoid



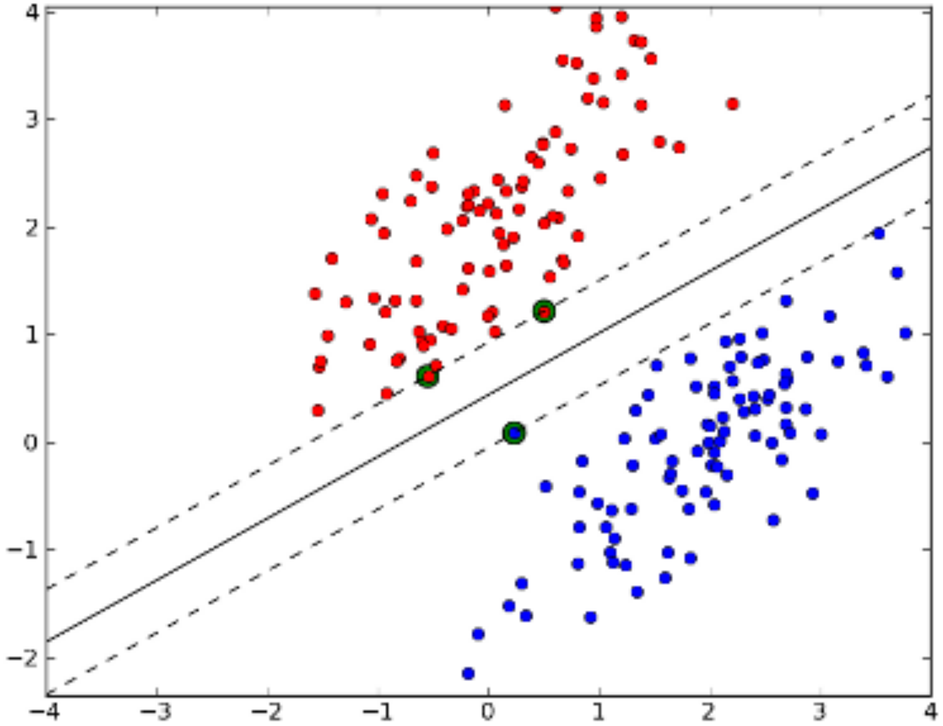
tanh

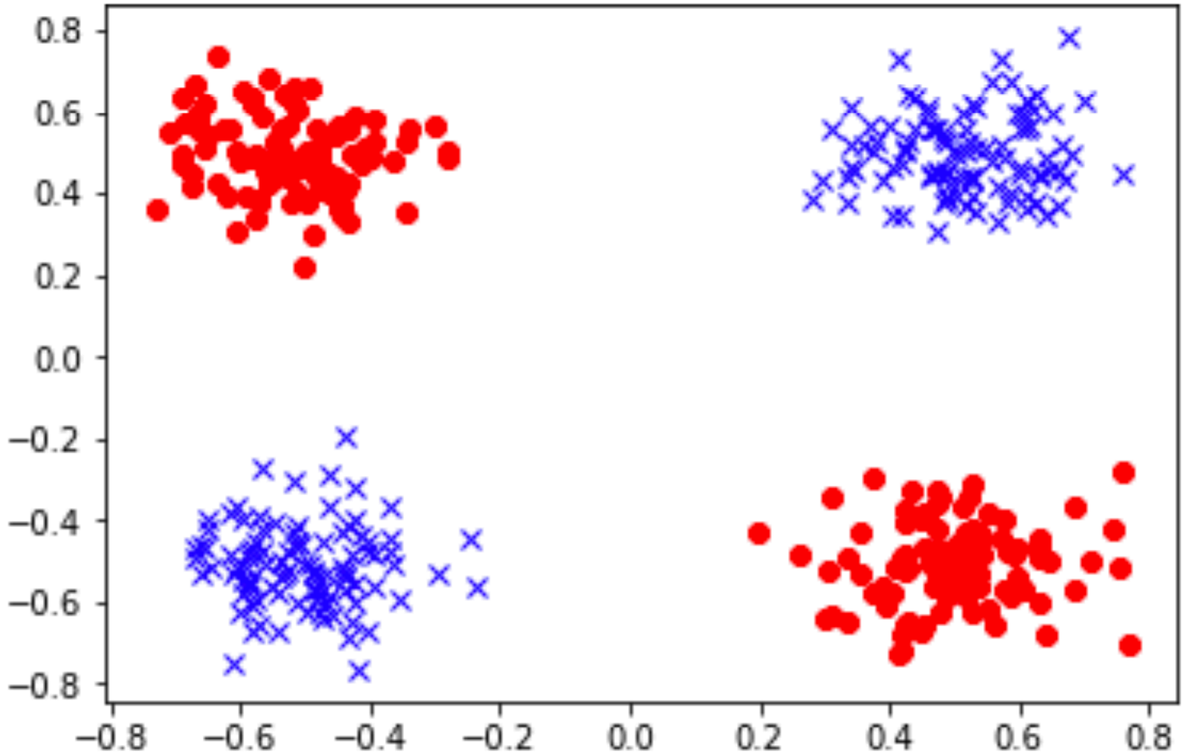


relu (Rectified Linear Unit)

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

Linear vs. Non-linear Functions

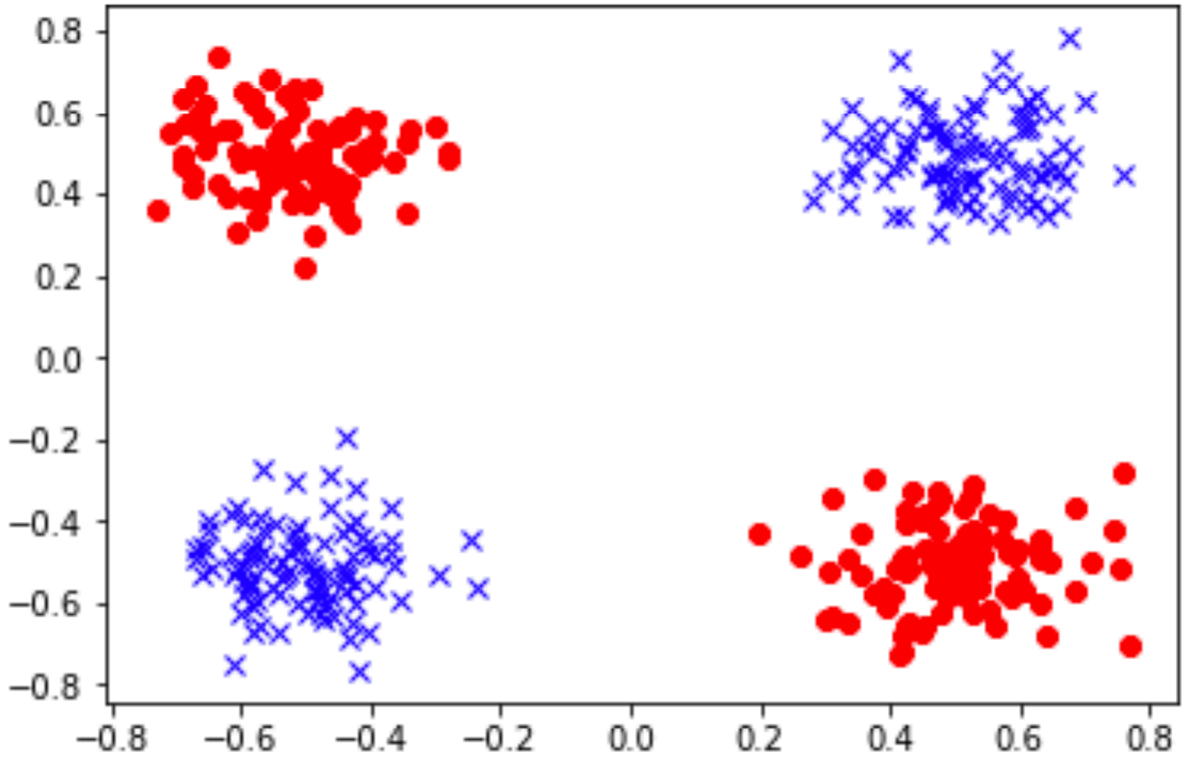




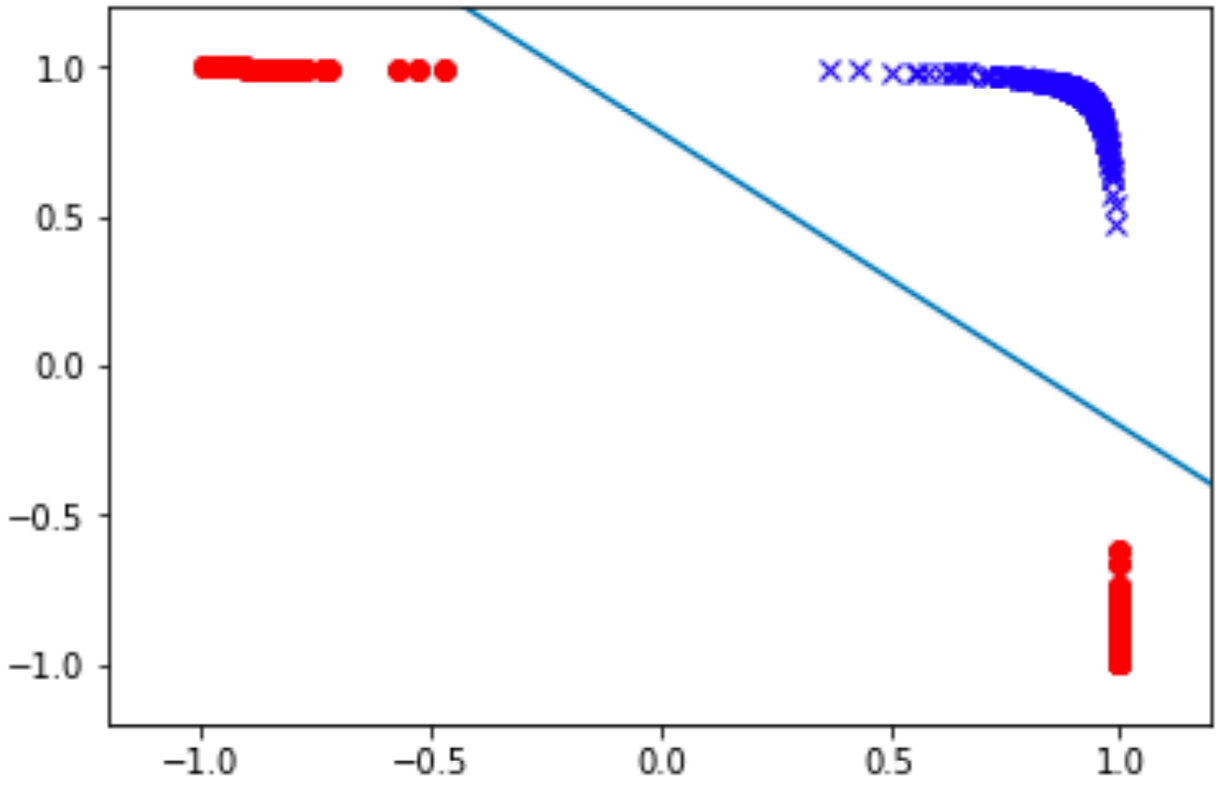
Linearly inseparable

Power of non-linearity

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



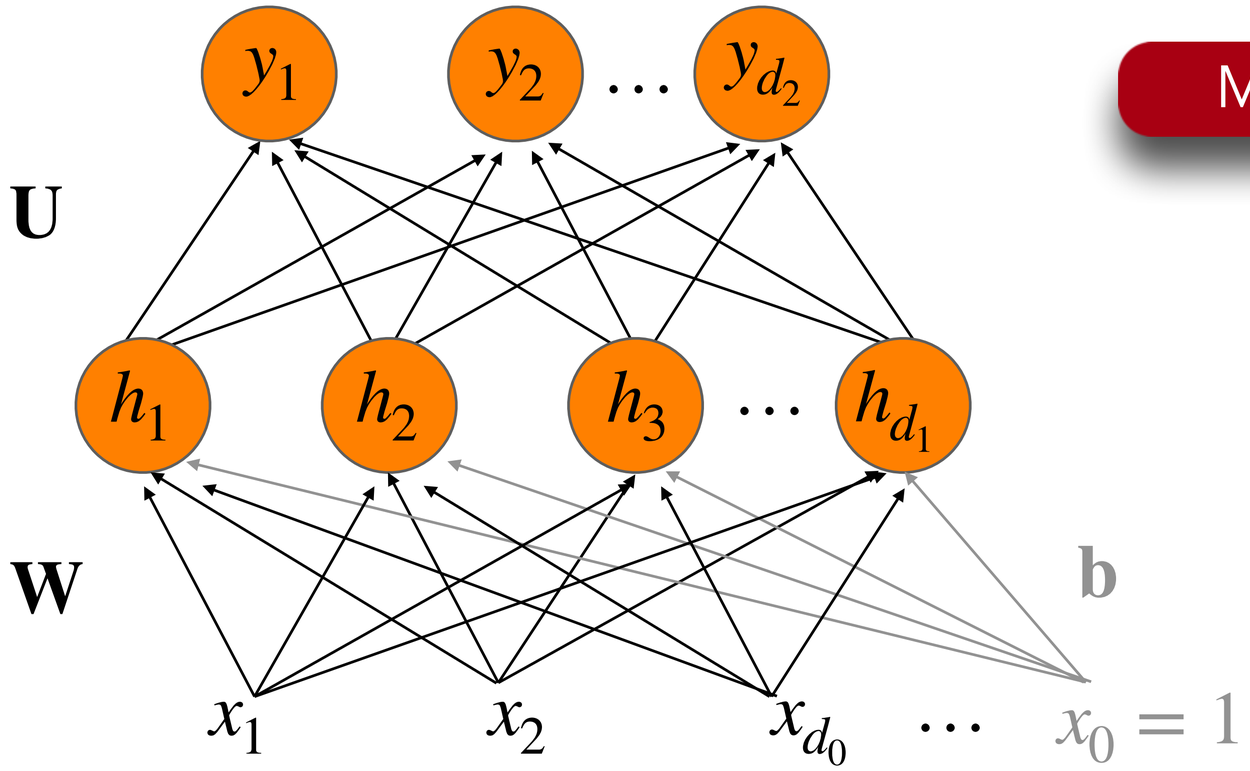
After a $\tanh(\cdot)$ transformation:



Feedforward Neural Nets

Multilayer Perceptron

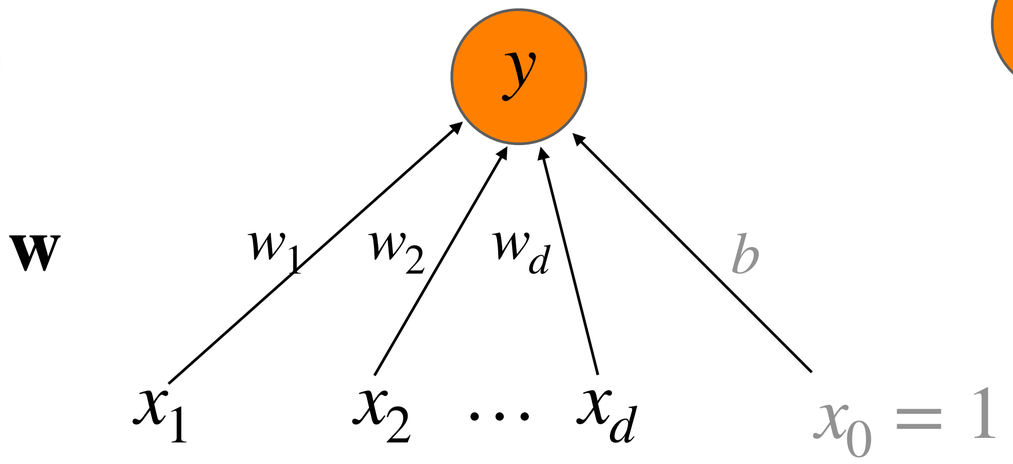
Technically, can learn any function!



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

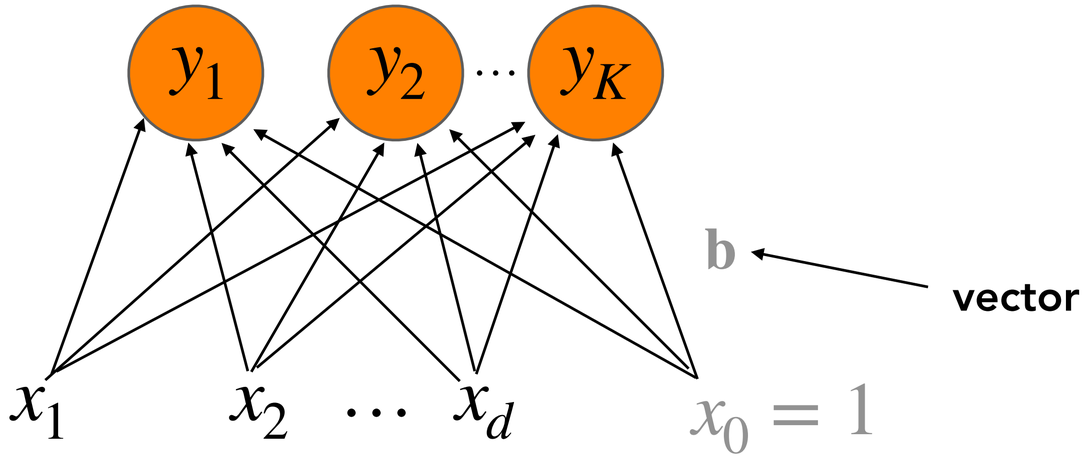
1-layer Network

Don't count the input layer in counting layers!

Multinomial Logistic Regression

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

matrix → $\mathbf{W}$



Input layer: vector $\mathbf{x}$

1-layer Network

Fully connected single layer network

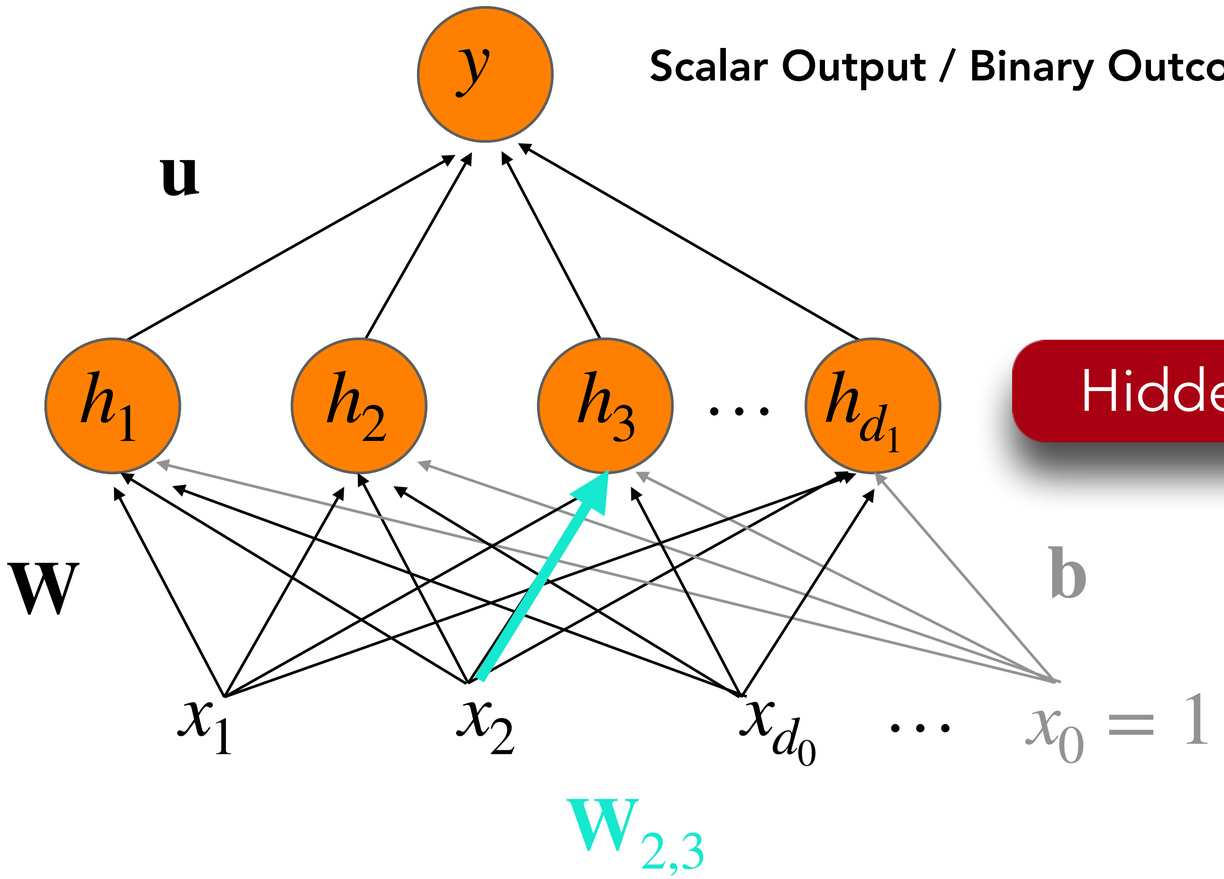
Two-layer Feedforward Network

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

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

Usually ReLU or tanh

Input layer: vector $\mathbf{x}$



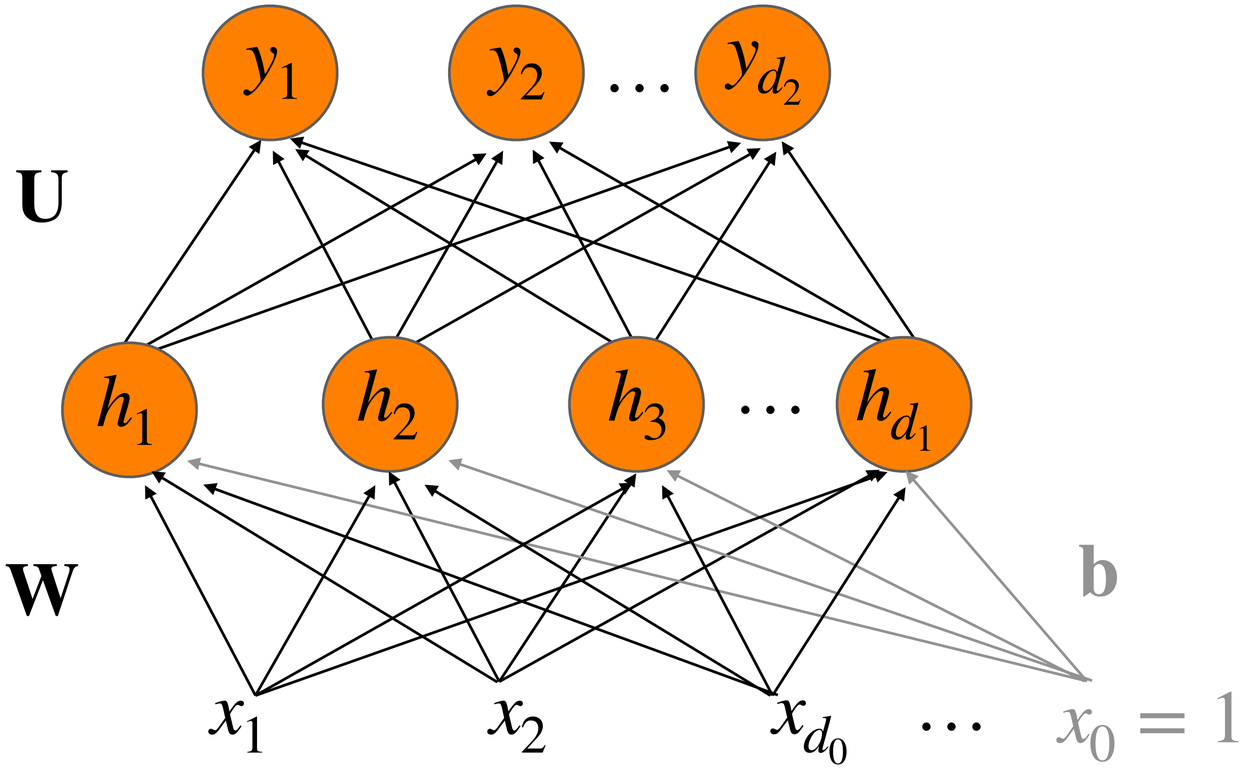
Two-layer Feedforward Network with Softmax Output

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 $\mathbf{y}$?

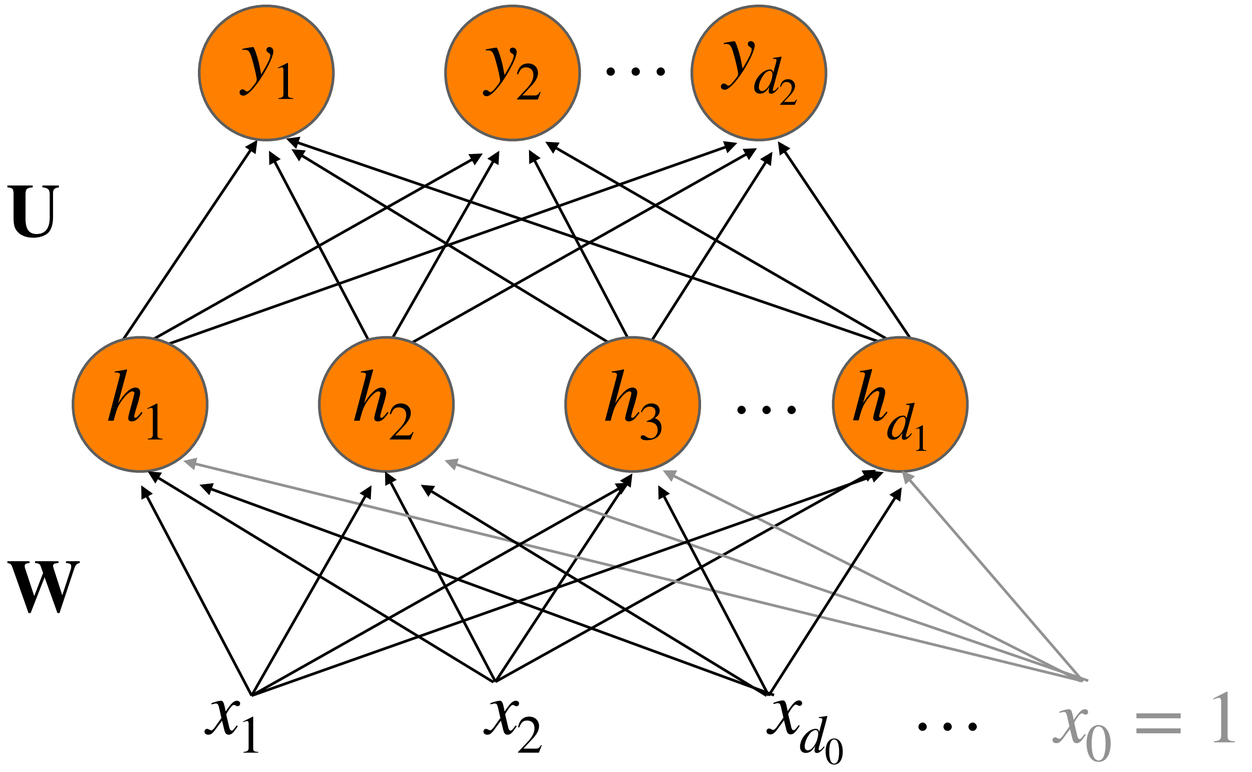
Two-layer FFNN: Notation

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

Lecture Outline

- Recap: word2vec
- GloVe
- Evaluating Word Embeddings
- Feedforward Neural Nets
- Feedforward Net Language Models

FFNN Language Models

Feedforward Neural Language Models

- Language Modeling: Calculating the probability of the next word in a sequence given some history.
- Compared to n-gram language models, neural network LMs achieve much higher performance
  - In general, count-based methods can never do as well as optimization-based ones
- State-of-the-art neural LMs are based on more powerful neural network technology like Transformers
- But **simple feedforward LMs** work well too!

Why?

Can neural LMs overcome the overfitting problem in n-gram LMs?

Simple Feedforward Neural LMs

Task: predict next word w_t given prior words $w_{t-1}, w_{t-2}, w_{t-3}, \ldots$

Problem: Now we are dealing with sequences of arbitrary length….

Solution: Sliding windows (of fixed length)

Basis of word embedding models!

$$P(w_t \mid w_{t-1}) \approx P(w_t \mid w_{t-1:t-M+1})$$

First introduced by Yoshua Bengio and colleagues in 2003

Data: Feedforward Language Model

- Self-supervised
- Computation is divided into time steps t, where different sliding windows are considered
- $x_t = (w_{t-1}, \ldots, w_{t-M+1})$ for the context
  - represent words in this prior context by their embeddings, rather than just by their word identity as in n-gram LMs
  - allows neural LMs to generalize better to unseen data / similar data
  - All embeddings in the context are concatenated
- $y_t = w_t$ for the next word
  - Represented as a one hot vector of vocabulary size where only the ground truth gets a value of 1 and every other element is a 0

One-hot vector