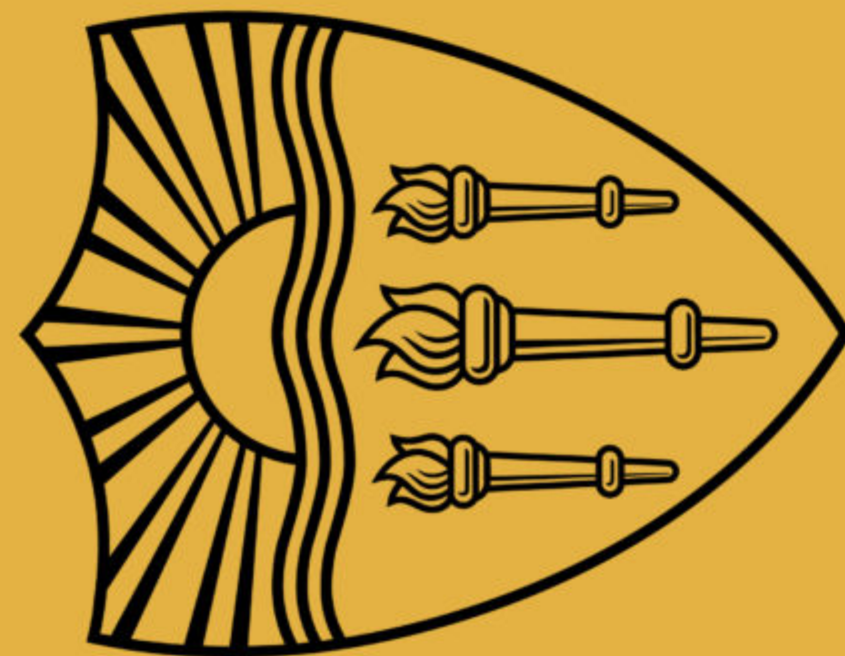
Lecture 8: Feed-Forward Neural Nets

Instructor: Swabha Swayamdipta

USC CSCI 499 LMs in NLP

Feb 12, 2024 Spring



Slides mostly adapted from Dan Jurafsky, some from Mohit Iyyer

Logistics / Announcements

- HW2 due on Monday, 2/26
  - Start early!
- Feedback for Project Proposal will be provided in a week
- Today: Quiz 2

Lecture Outline

- Recap: Logistic Regression and word2vec
- Quiz 2
- Feed-forward Neural Networks
- Feed-forward Language Models
- Training Feed-forward Neural Networks
- Computation Graphs and Backprop

Recap:
Logistic Regression +
word2vec

USC Viterbi

Ingredients of Supervised Machine Learning

I. **Data** as pairs $(x^{(i)}, y^{(i)})$ s.t $i \in \{1 \ldots N\}$
   - $x^{(i)}$ usually represented by a feature vector $\mathbf{x}^{(i)} = [x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)}]$
     - e.g. word embeddings

II. **Model**
   - A classification function that computes $\hat{y}$, the estimated class, via $p(y|x)$
     - e.g. sigmoid function: $\sigma(z) = 1/(1 + \exp(-z))$

III. **Loss**
   - An objective function for learning
     - e.g. cross-entropy loss, L_{CE}

IV. **Optimization**
   - An algorithm for optimizing the objective function
     - e.g. stochastic gradient descent

V. **Inference** / Evaluation

Case 1: Sentiment Analysis

Case 2: Word2Vec

Learning Phase

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

Word embedding itself is the learned parameter!

word2vec: Data

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

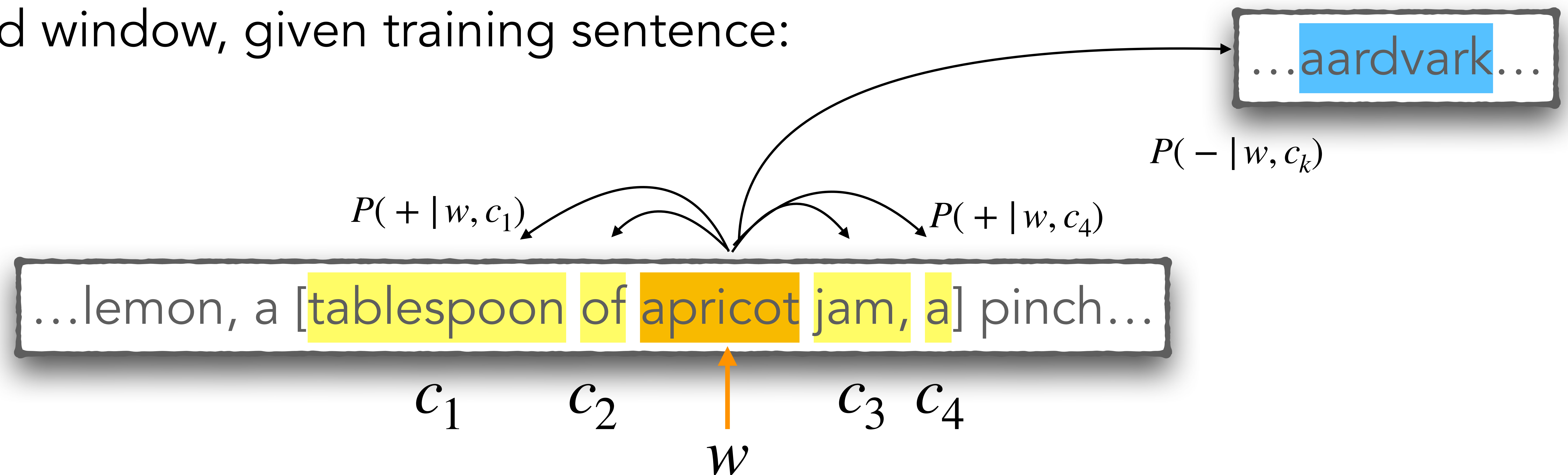
- Positive examples: A word c that occurs near "cherry" in the corpus acts as the gold "correct answer" for supervised learning
- Negative examples: randomly sampled words outside of the context window for target word
- No need for human labels!

Self-supervision

Bengio et al. (2003); Collobert et al. (2011)

word2vec: Goal

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



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

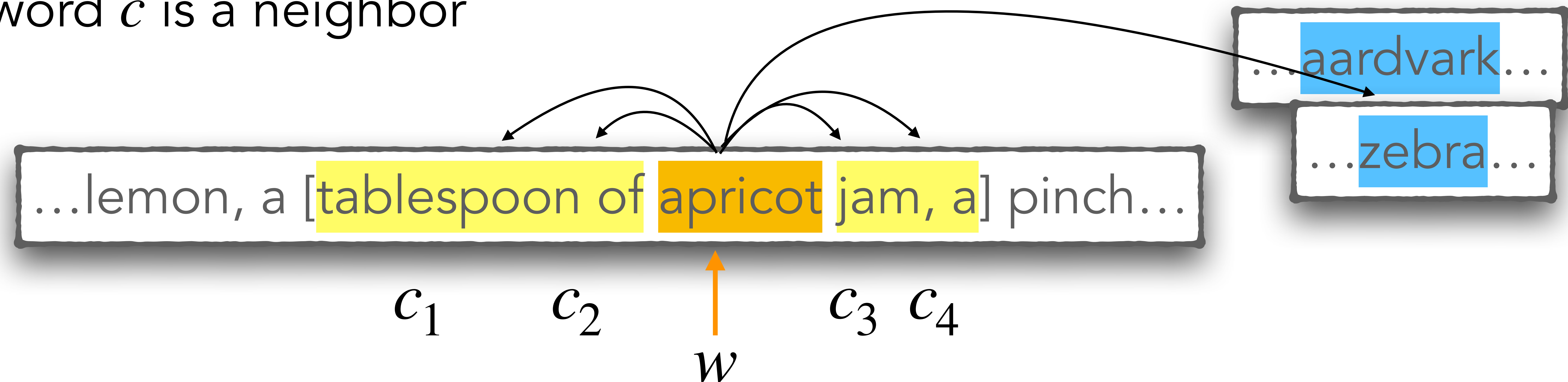
(apricot, jam)
(apricot, aardvark)
…

And assigns each pair a probability:

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

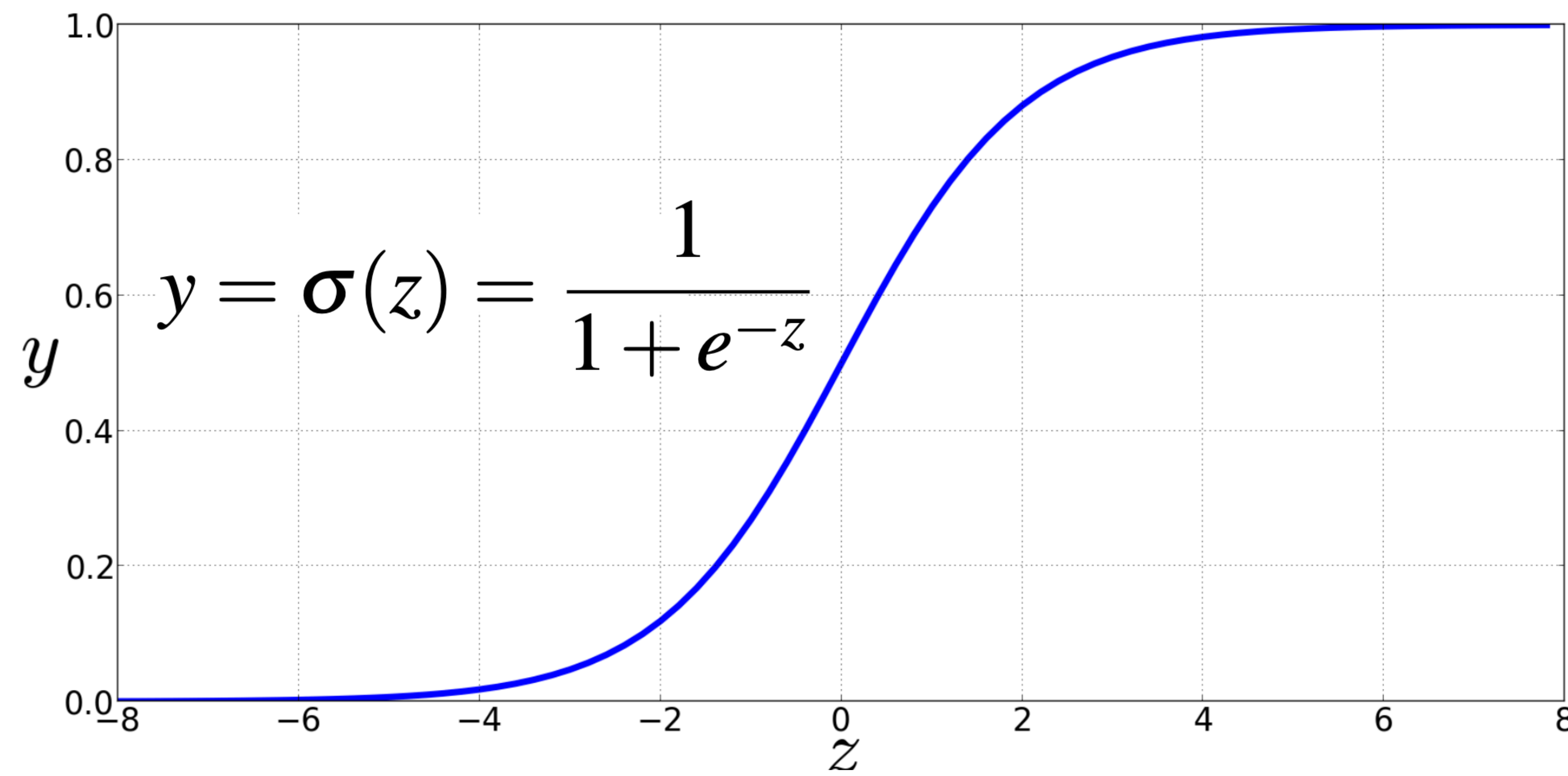
word2vec: Pseudocode

Predict if candidate word c is a neighbor



1. Treat the target word w and a neighboring context word c as positive examples.
2. Randomly sample other words in the lexicon to get **negative examples**
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

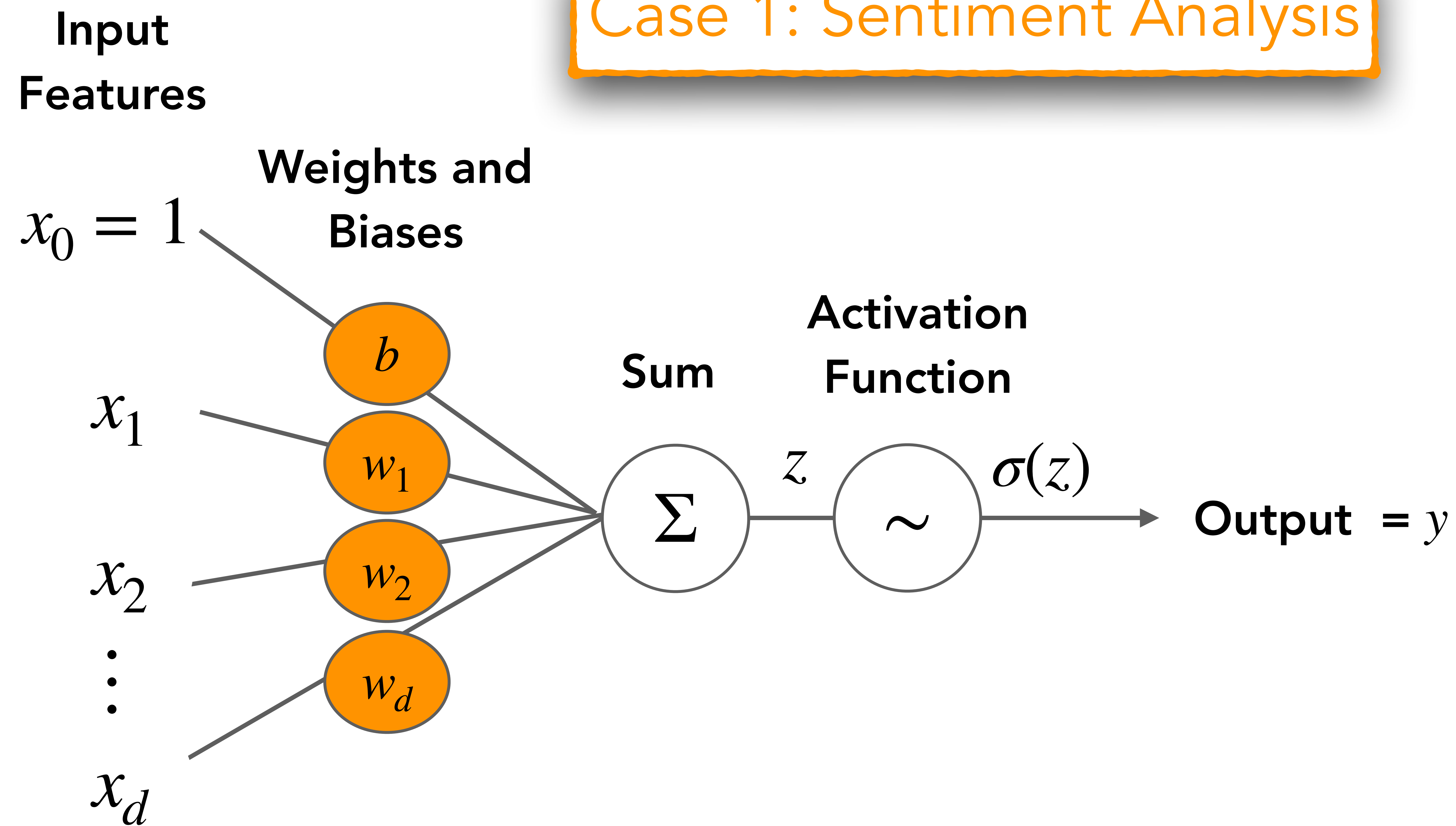
$$z = \left(\sum_d w_d x_d + b \right) = \mathbf{w} \cdot \mathbf{x} + b$$



$$P(y = 1 \mid \mathbf{x}; \theta) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$$
$$= \frac{1}{1 + \exp(-(\mathbf{w} \cdot \mathbf{x} + b))}$$

$$P(y = 0 \mid \mathbf{x}; \theta) = 1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b)$$

Case 1: Sentiment Analysis



Minimizing negative log likelihood

Goal: maximize probability of the correct label $p(y \mid \mathbf{x})$

Case 1: Sentiment Analysis

$$\log p(y \mid x) = \log(\hat{y}^y (1 - \hat{y})^{1-y})$$
$$= y \log \hat{y} + (1 - y)\log(1 - \hat{y})$$

Maximize:

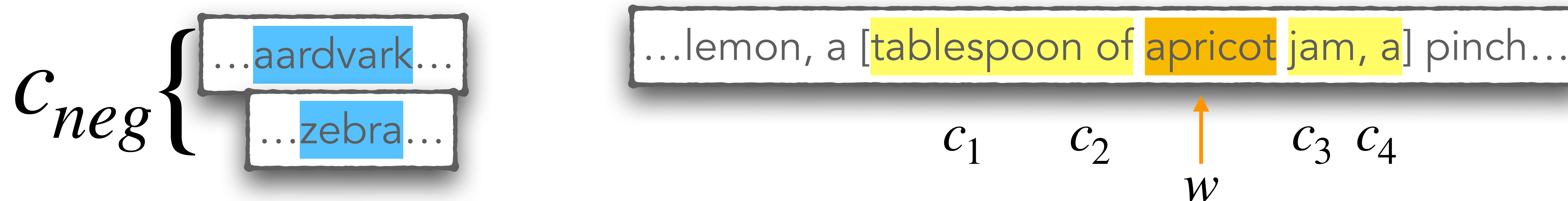
For something to minimize (we minimize the loss / cost), just flip the sign

Minimize:

$$L_{CE}(y, \hat{y}) = -\log p(y \mid x) = -[y \log \hat{y} + (1 - y)\log(1 - \hat{y})]$$
$$= -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1 - y)\log \sigma[-(\mathbf{w} \cdot \mathbf{x} + b)]]$$

Cross-Entropy Loss

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

word2vec: 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

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

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

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

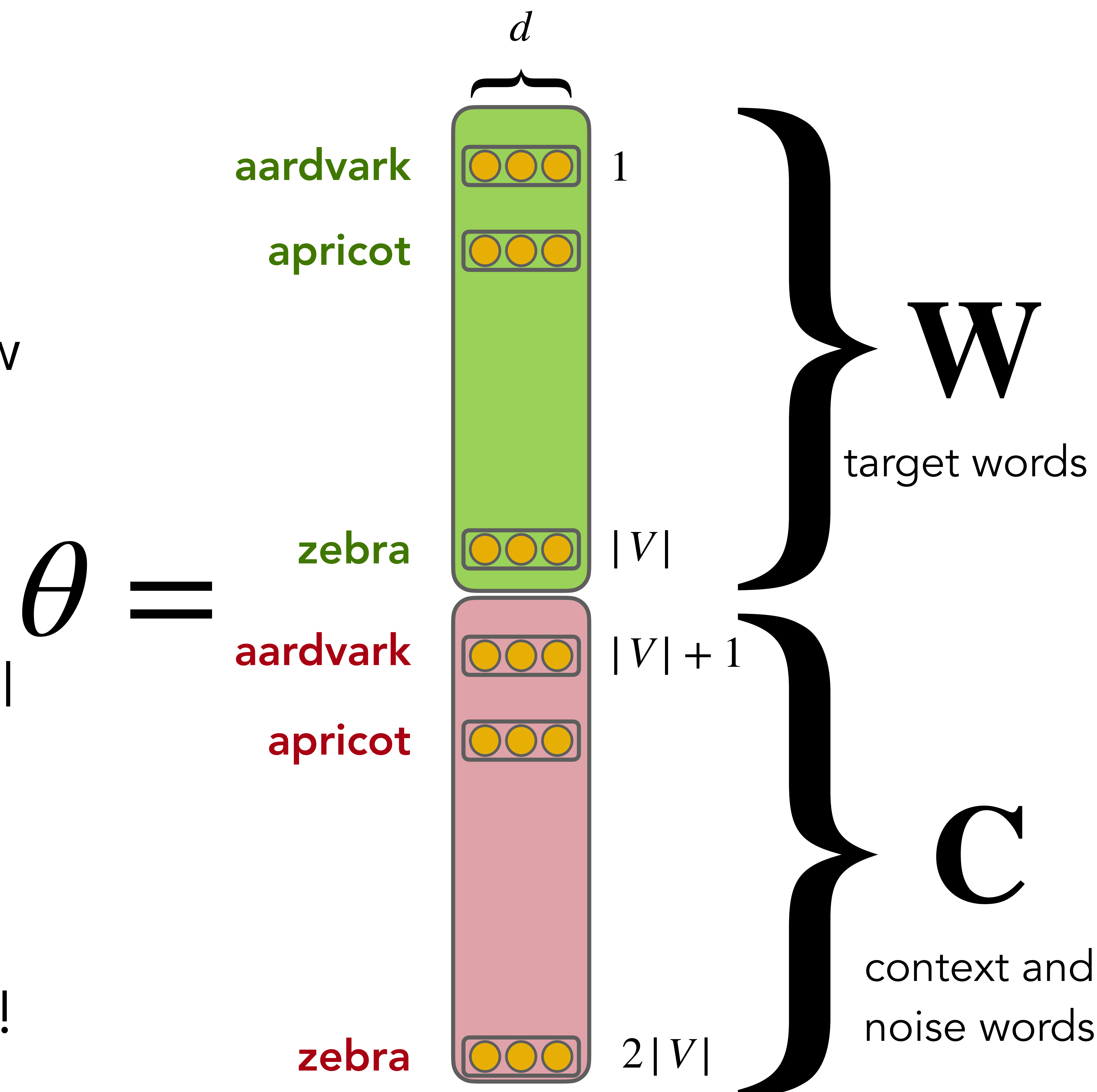
$$= -\left[\log \sigma(\mathbf{w}\cdot\mathbf{c}_{pos}) + \sum_{j=1}^{K} \log \sigma(-\mathbf{w}\cdot\mathbf{c}_{neg_j})\right]$$

USC Viterbi

word2vec classifier: Summary

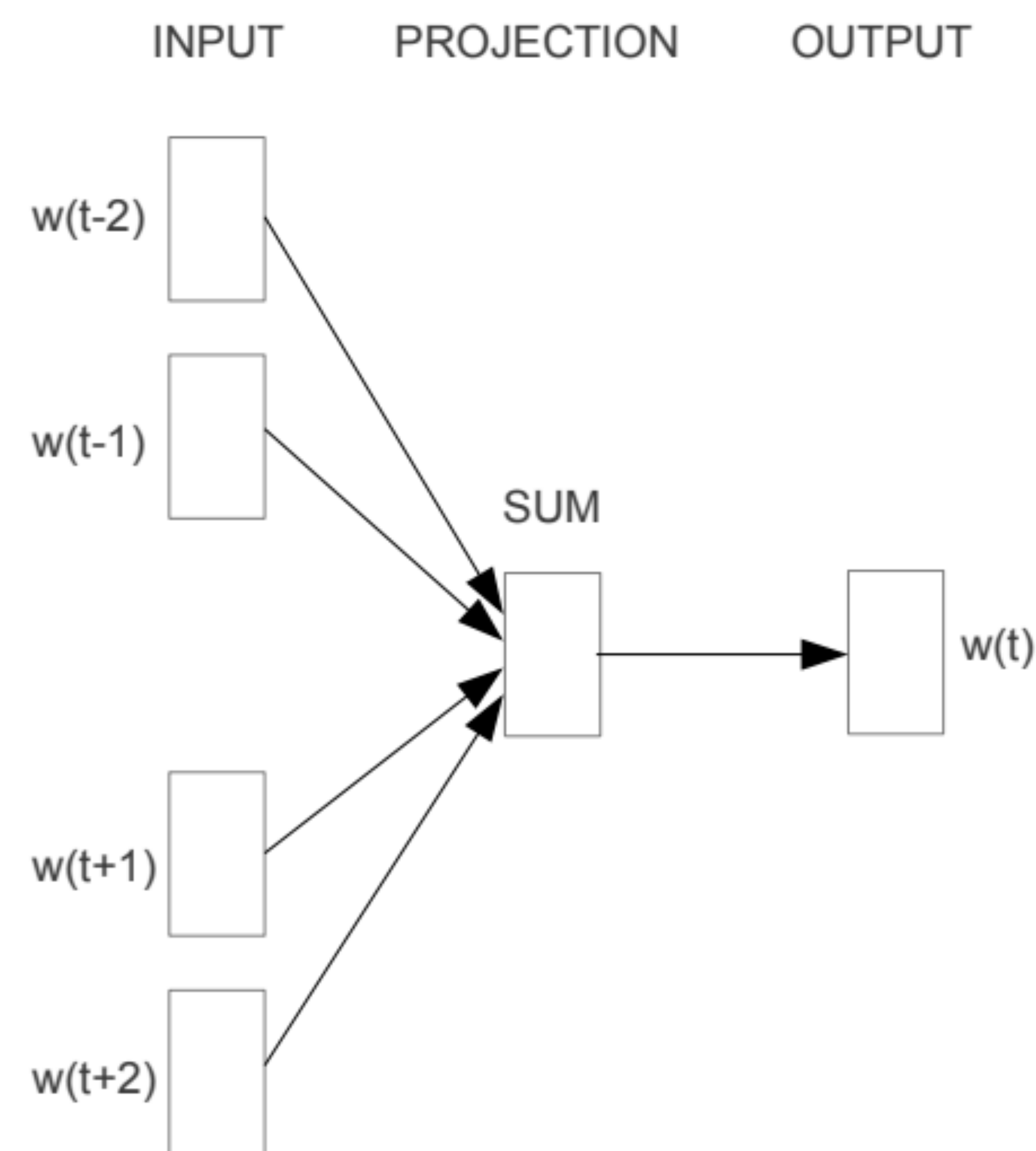
- A probabilistic classifier, given
  - a test target word w
  - its context window of L words $c_{1:L}$
- Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings)

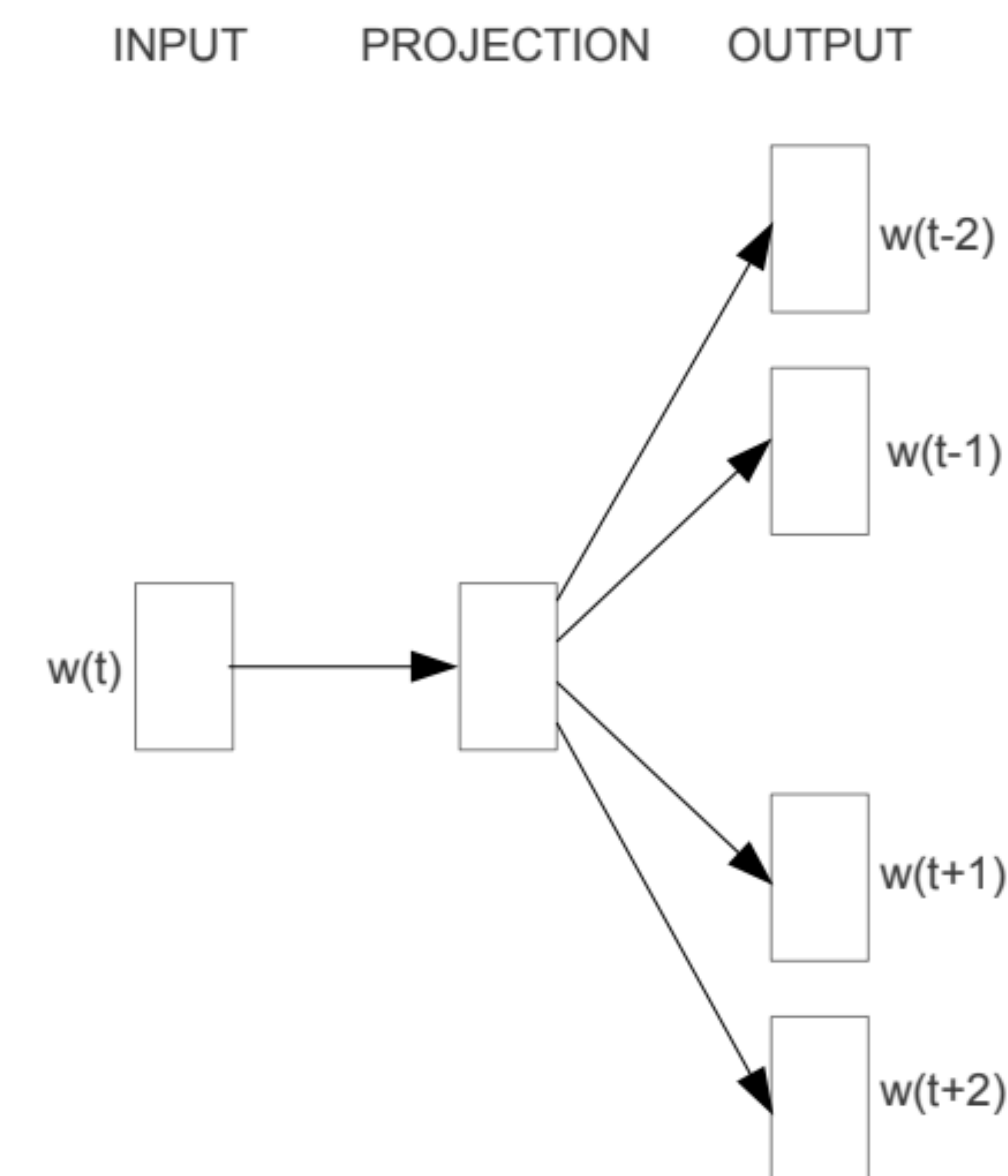
- To compute this, we just need embeddings for all the words
  - Separate representations for targets and contexts
  - Same as the parameters we need to estimate!



word2vec Variants: 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





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

Reminder: Gradient Descent

$$w_{t+1} = w_t - \eta \frac{\partial}{\partial w} L(f(x; w), y^*)$$

At each step of gradient descent, we update the parameter w

- Direction: We move in the reverse direction from the gradient of the loss function
- Magnitude: we move the value of this gradient $\frac{\partial}{\partial w} L(f(x; w), y^*)$, weighted by a learning rate η
- Higher learning rate means move w faster

The gradient vector expresses the directional components of the sharpest slope along each of the d dimensions for $\mathbf{w} \in \mathbb{R}^d$

USC Viterbi

Gradients for Logistic Regression

The cross-entropy loss for logistic regression

Case 1: Sentiment Analysis

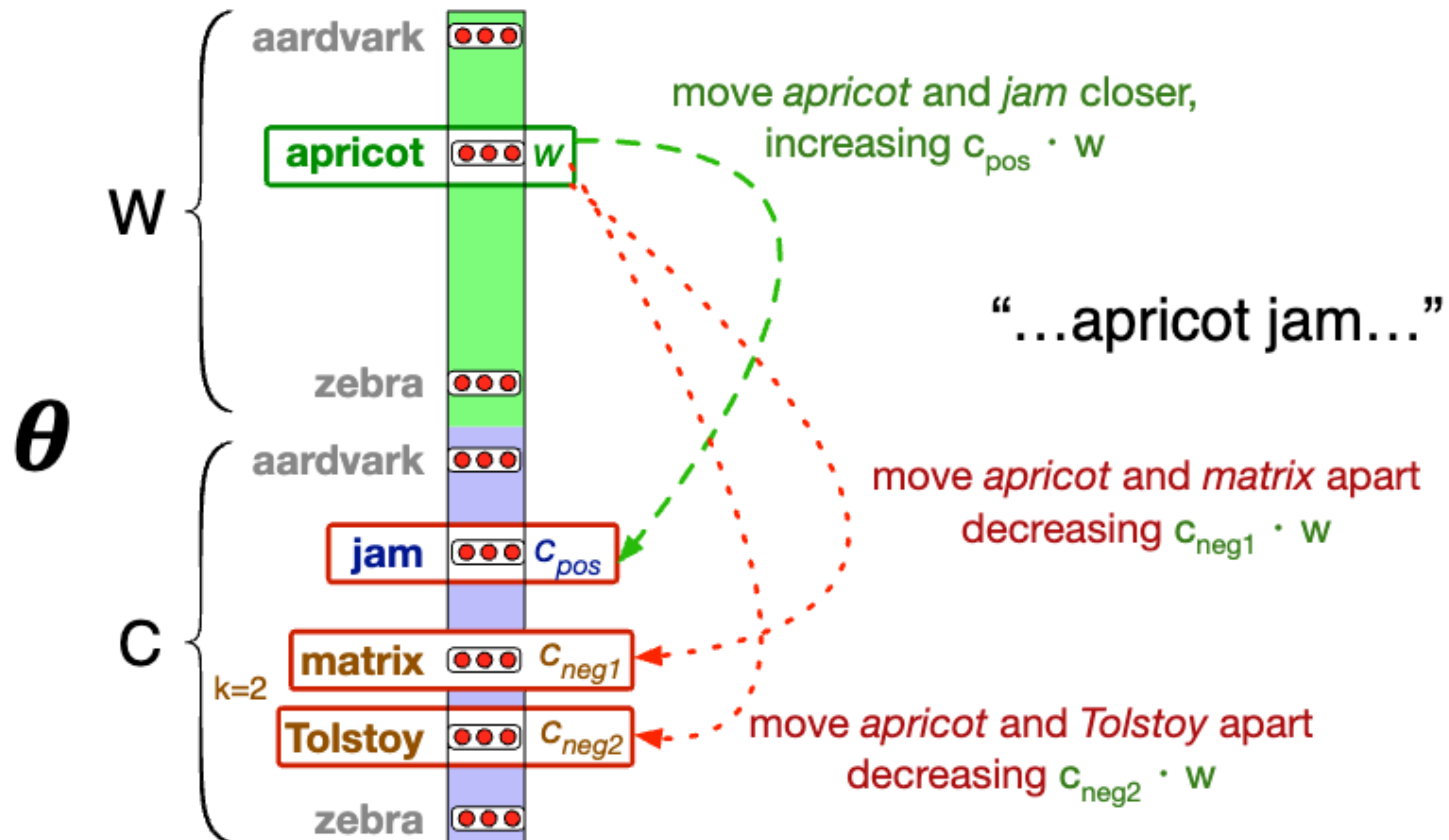
$$L_{CE}(\hat{y}, y) = -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1 - y)\log(\sigma(-\mathbf{w} \cdot \mathbf{x} + b))]$$

Derivatives have a closed form solution:

$$\frac{\partial L_{CE}(\hat{y}, y)}{\partial w_j} = [\sigma(\mathbf{w} \cdot \mathbf{x} + b) - y]x_j$$

USC Viterbi

Intuition of one step of gradient descent



Gradients for word2vec

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

3 different parameters

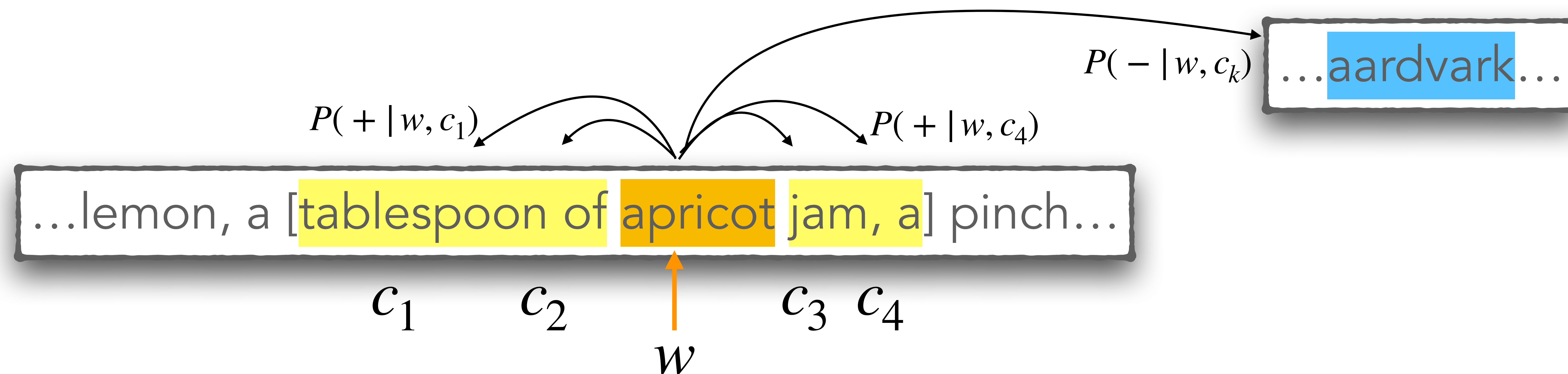
$$\frac{\partial L_{CE}}{\partial \mathbf{c}_{pos}} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{w}$$

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

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

Update the parameters by subtracting respective η-weighted gradients

word2vec: Summary

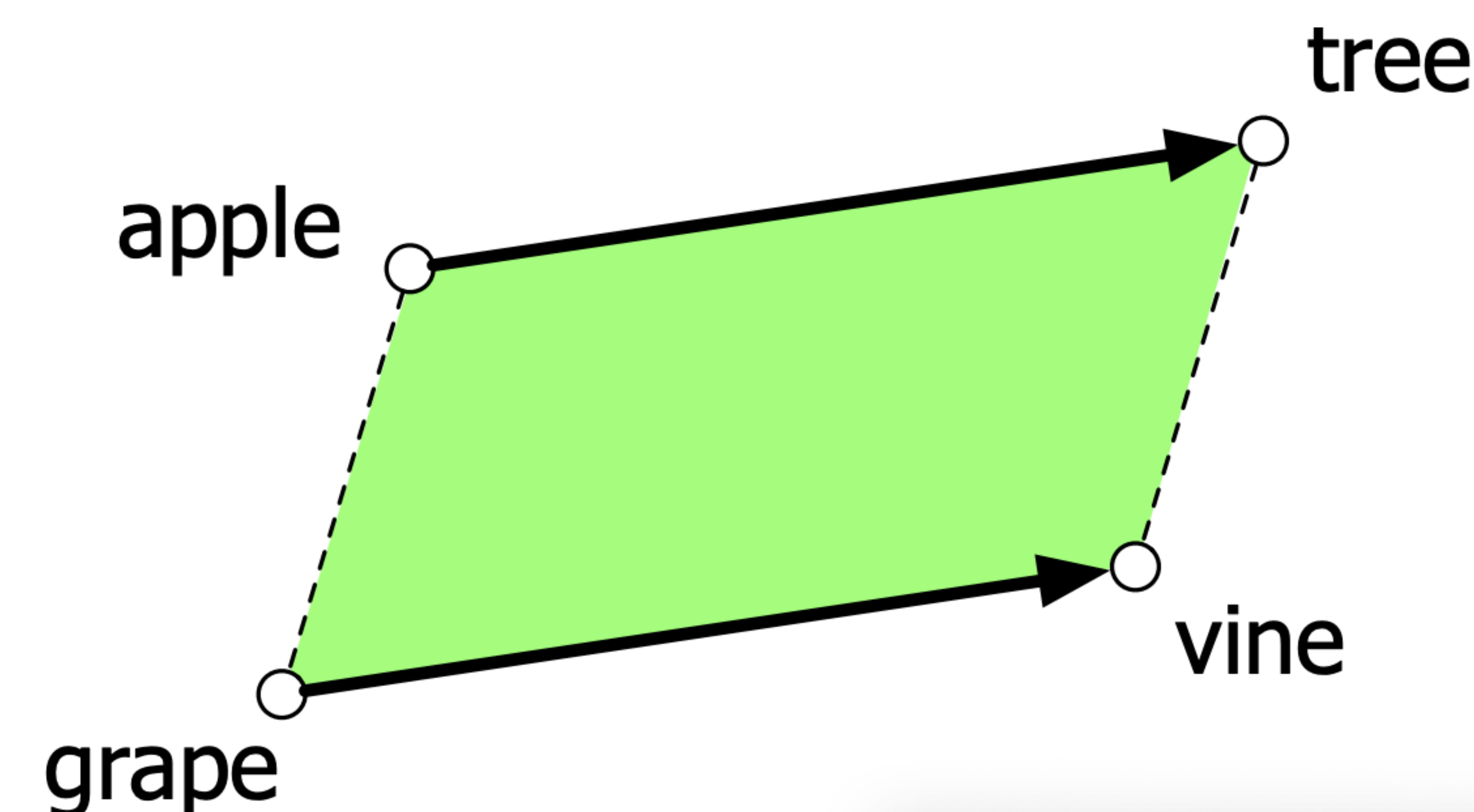


- Start with $2|V|$ random d-dimensional vectors as initial embeddings
- Train a classifier based on embedding similarity
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish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

Evaluation for word2vec: 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 …"



Rumelhart and Abrahamson, 1973

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

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

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

Multinomial Logistic Regression

Softmax Function

K different possible ground truths

$$\textbf{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)} \quad 1 \leq i \leq K$$

Vector $\mathbf{z} = [z_1, z_2, \ldots, z_K]$ of K and respective probabilities:

$$\textbf{softmax}(\mathbf{z}) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{K} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{K} \exp(z_i)}, \ldots, \frac{\exp(z_K)}{\sum_{i=1}^{K} \exp(z_i)} \right]$$

The denominator $\sum_{i=1}^{K} \exp(z_i)$ is used to normalize all the values into probabilities

Softmax is a generalization of the sigmoid function

Softmax in multinomial logistic regression

Parameters are now a matrix $\mathbf{W} \in \mathbb{R}^{d \times K}$ and $b \in \mathbb{R}^1$

$$P(y = c \mid \mathbf{x}; \theta) = \frac{\exp(\mathbf{w}_c \cdot \mathbf{x} + b)}{\sum_{j=1}^{K} \exp(\mathbf{w}_j \cdot \mathbf{x} + b)}$$

Could also have a separate b for each class!

- Input is still the dot product between weight vector $\mathbf{w}_c$ and input vector $\mathbf{x}$, offset by b
- But separate weight vectors for each of the K classes, each of dimension d

Multinomial LR Loss:

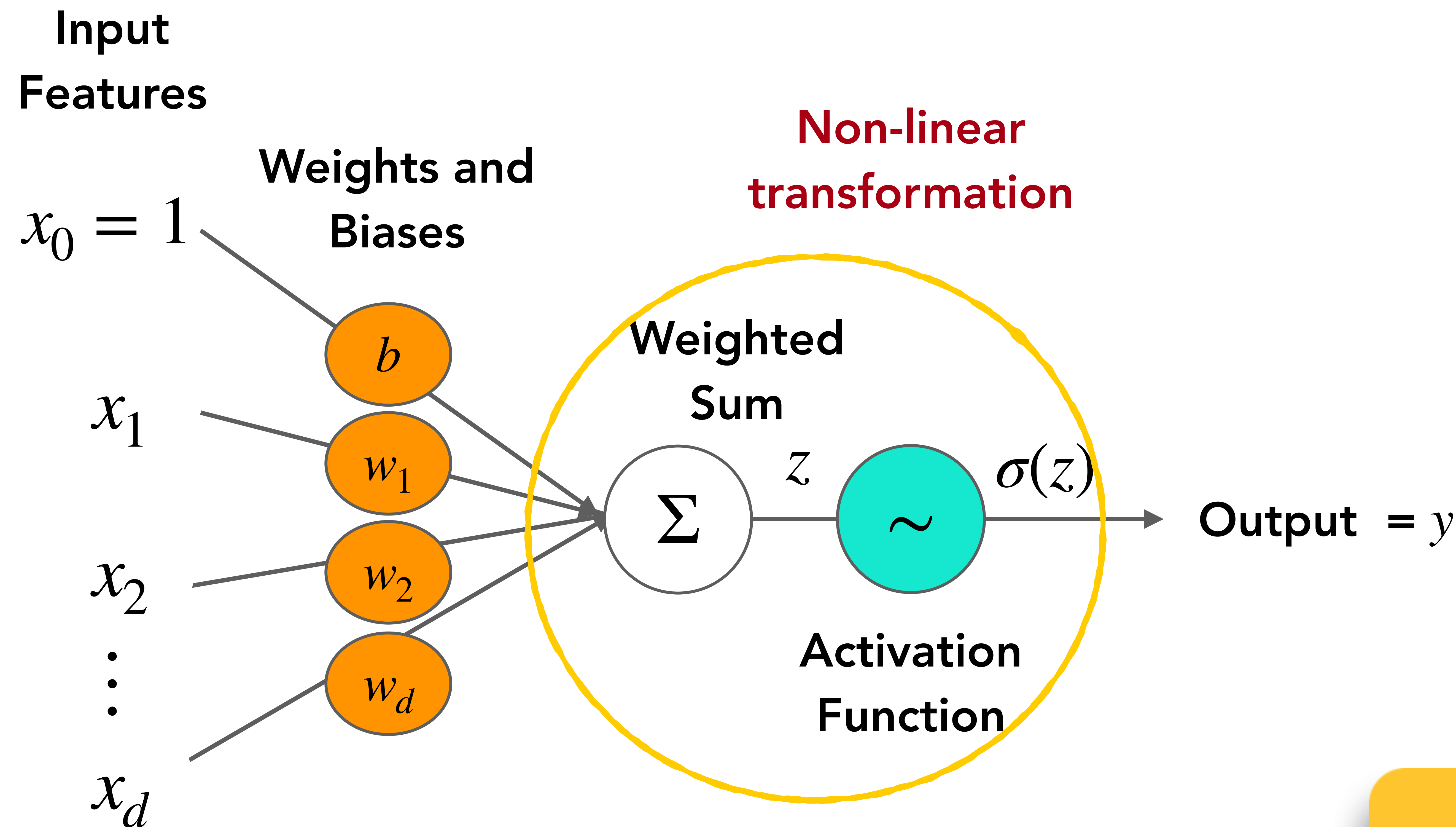
$$L_{CE} = -\log P(y = c \mid \mathbf{x}; \theta) = -(\mathbf{w}_c \cdot \mathbf{x} + b) + \log\left[\sum_{j=1}^{K} \exp(\mathbf{w}_j \cdot \mathbf{x} + b)\right]$$

Quiz 2!

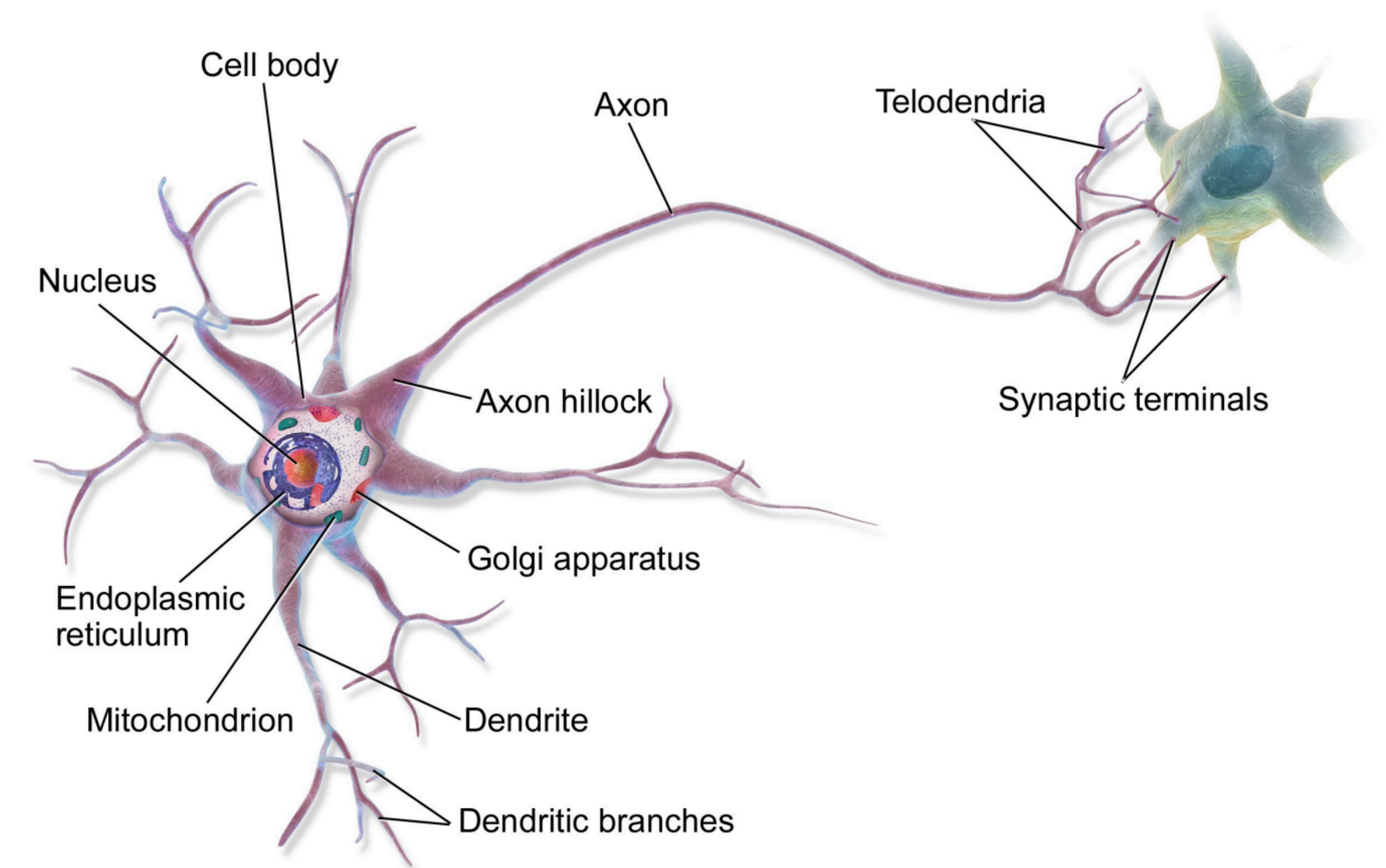
Feed-Forward Neural Networks

Neural Network Unit

Logistic Regression is a very simple neural network



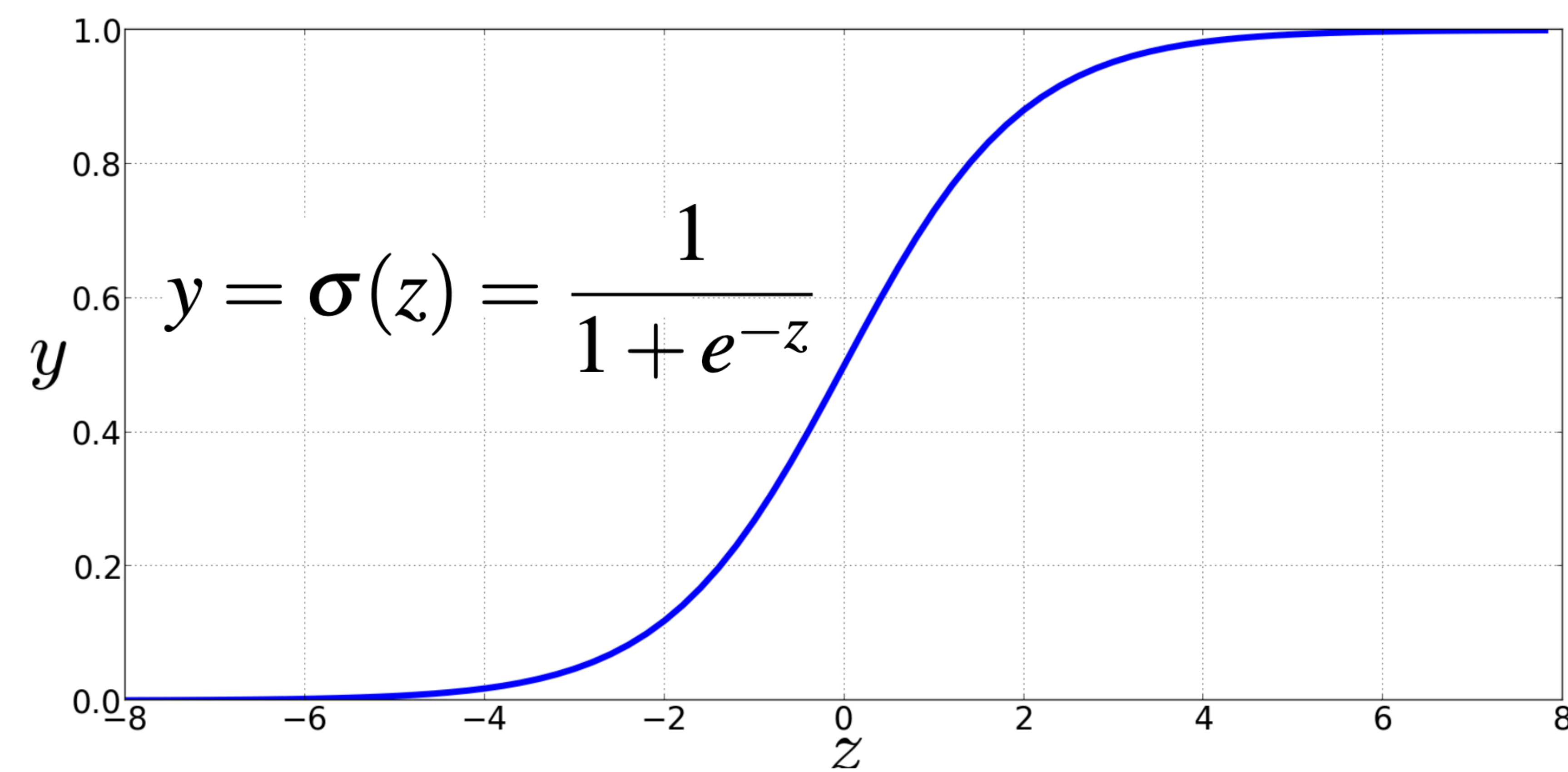
Output $= y$

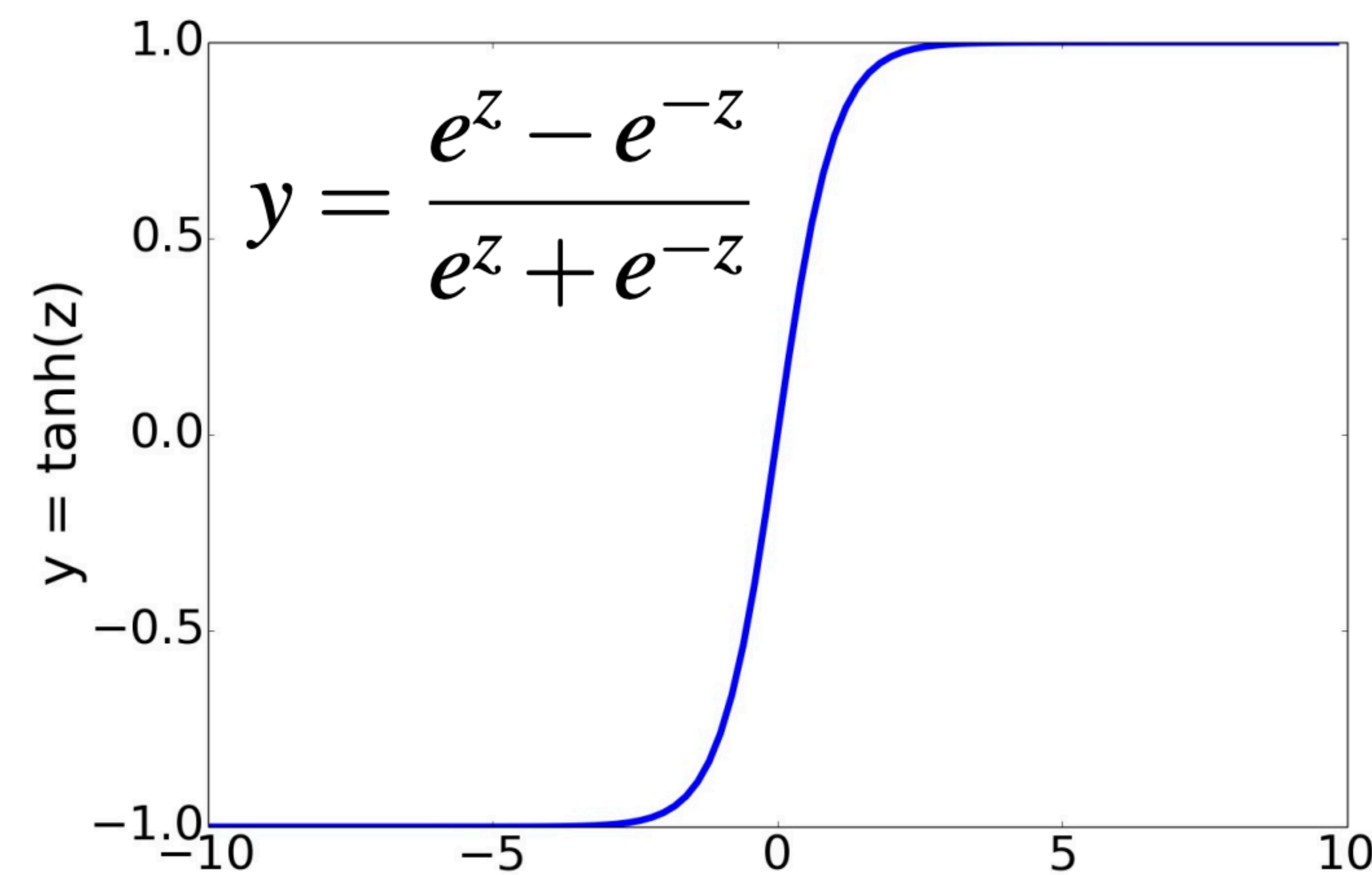


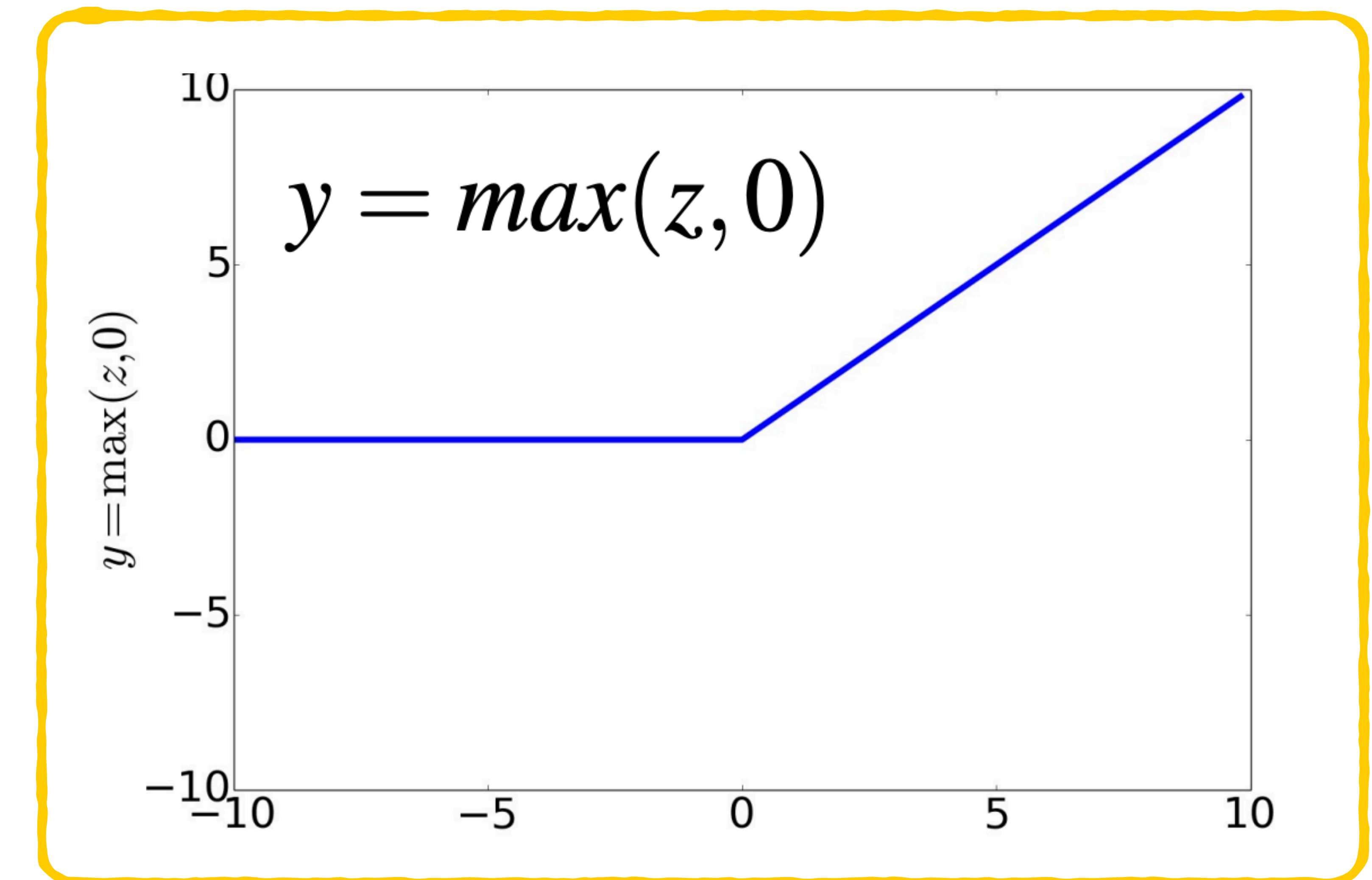
Resembles a neuron in the brain!

Non-Linear Activation Functions

Most common!







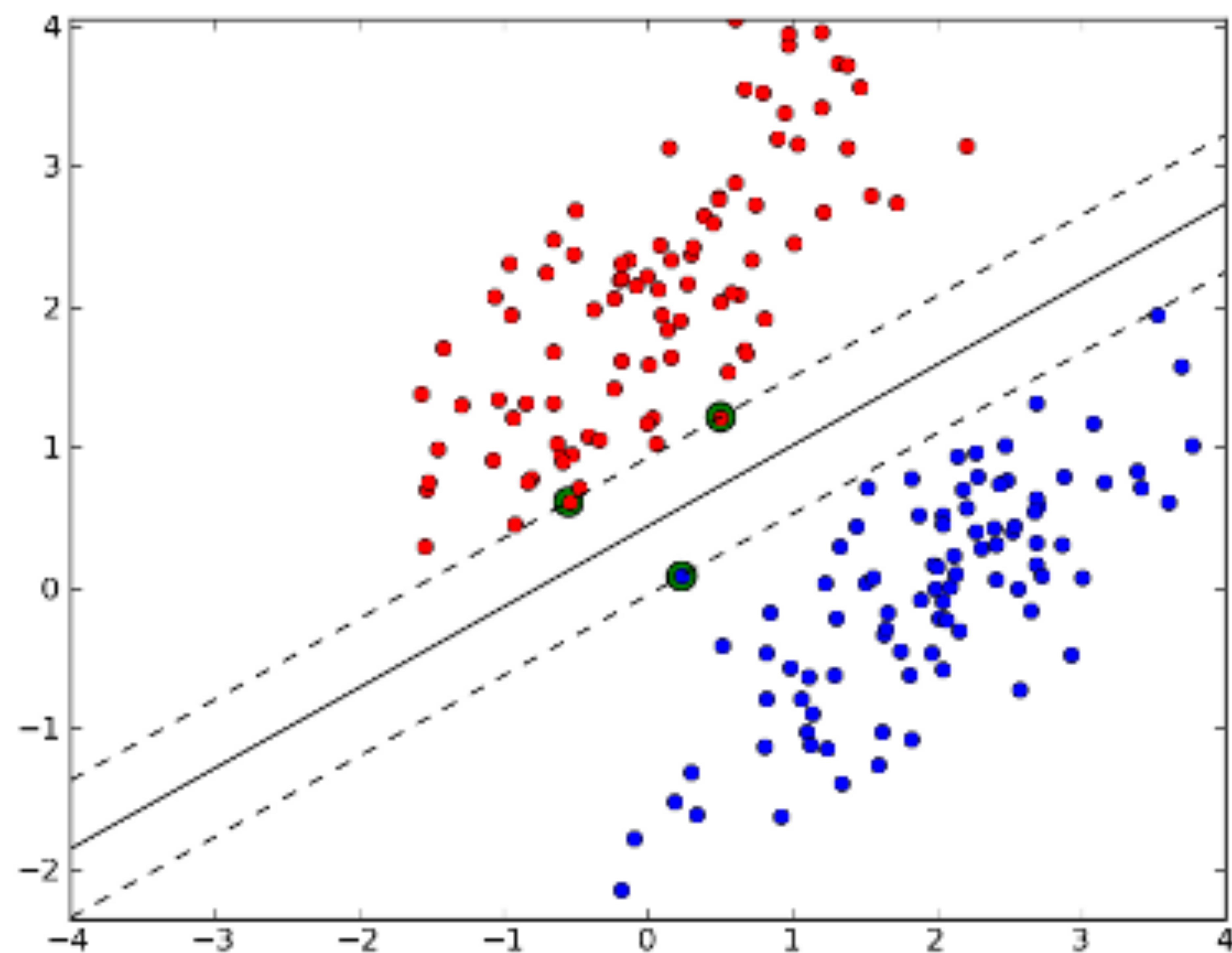
sigmoid

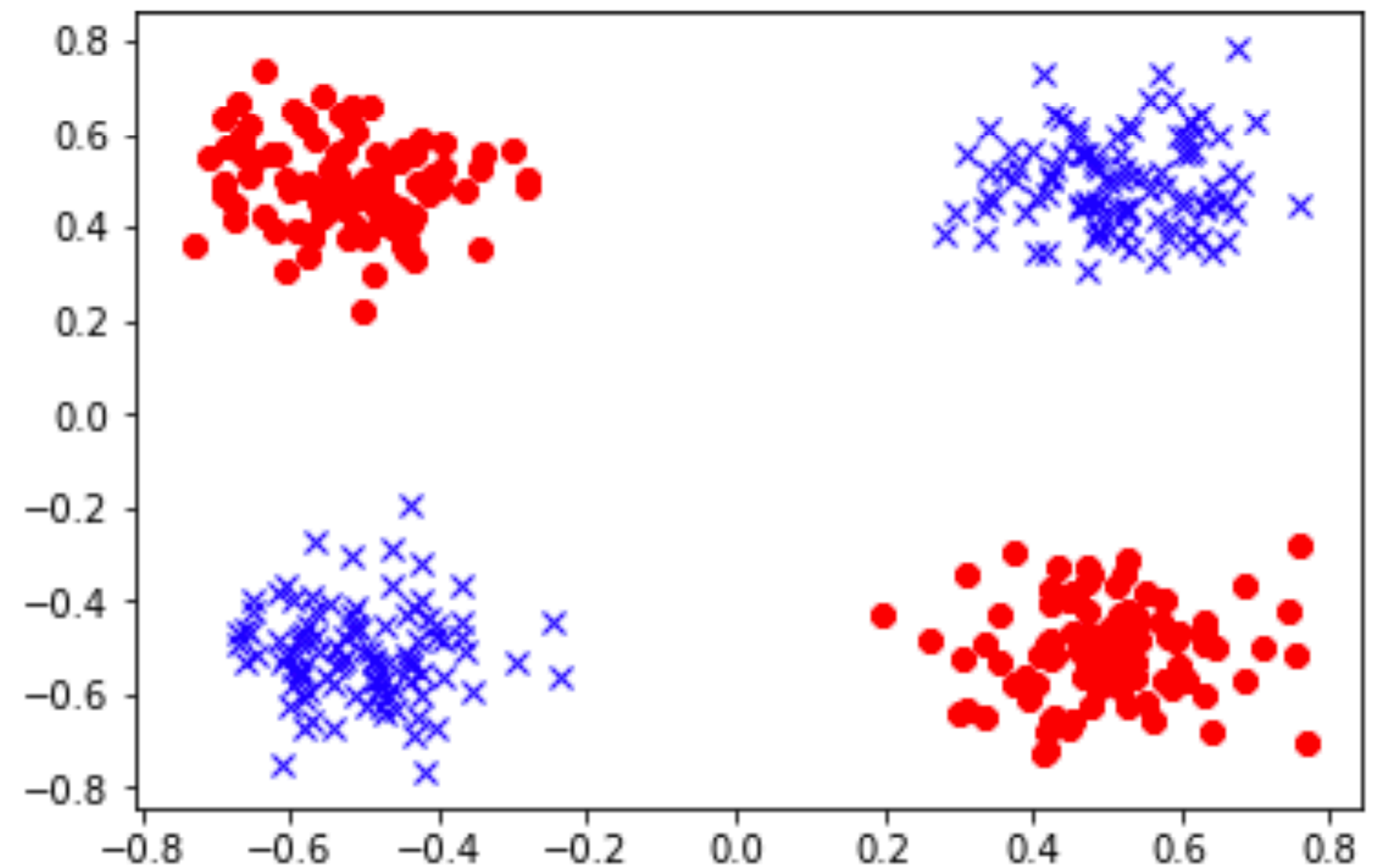
tanh

relu (Rectified Linear Unit)

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

Linear vs. Non-linear Functions



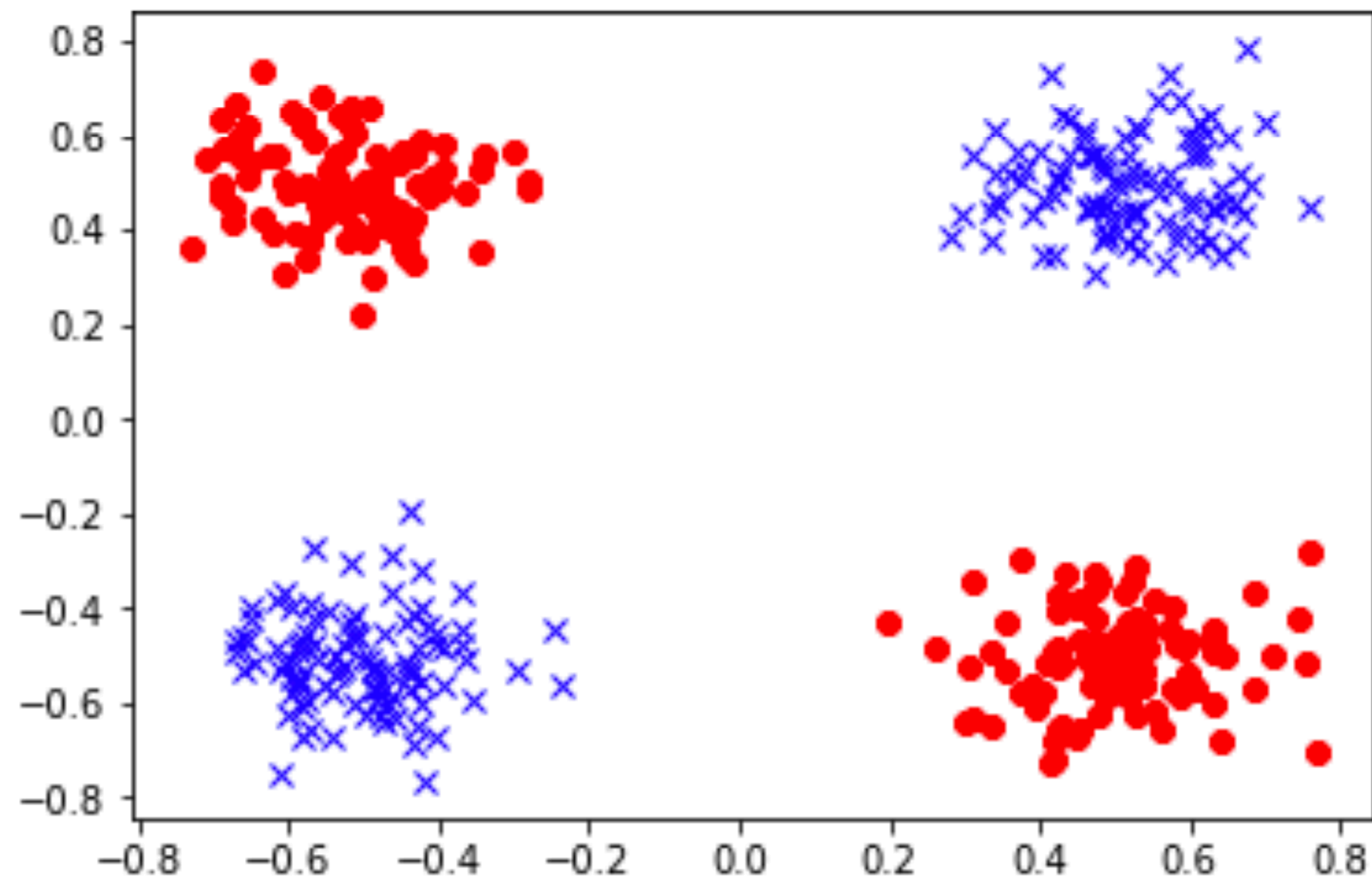


Linearly inseparable

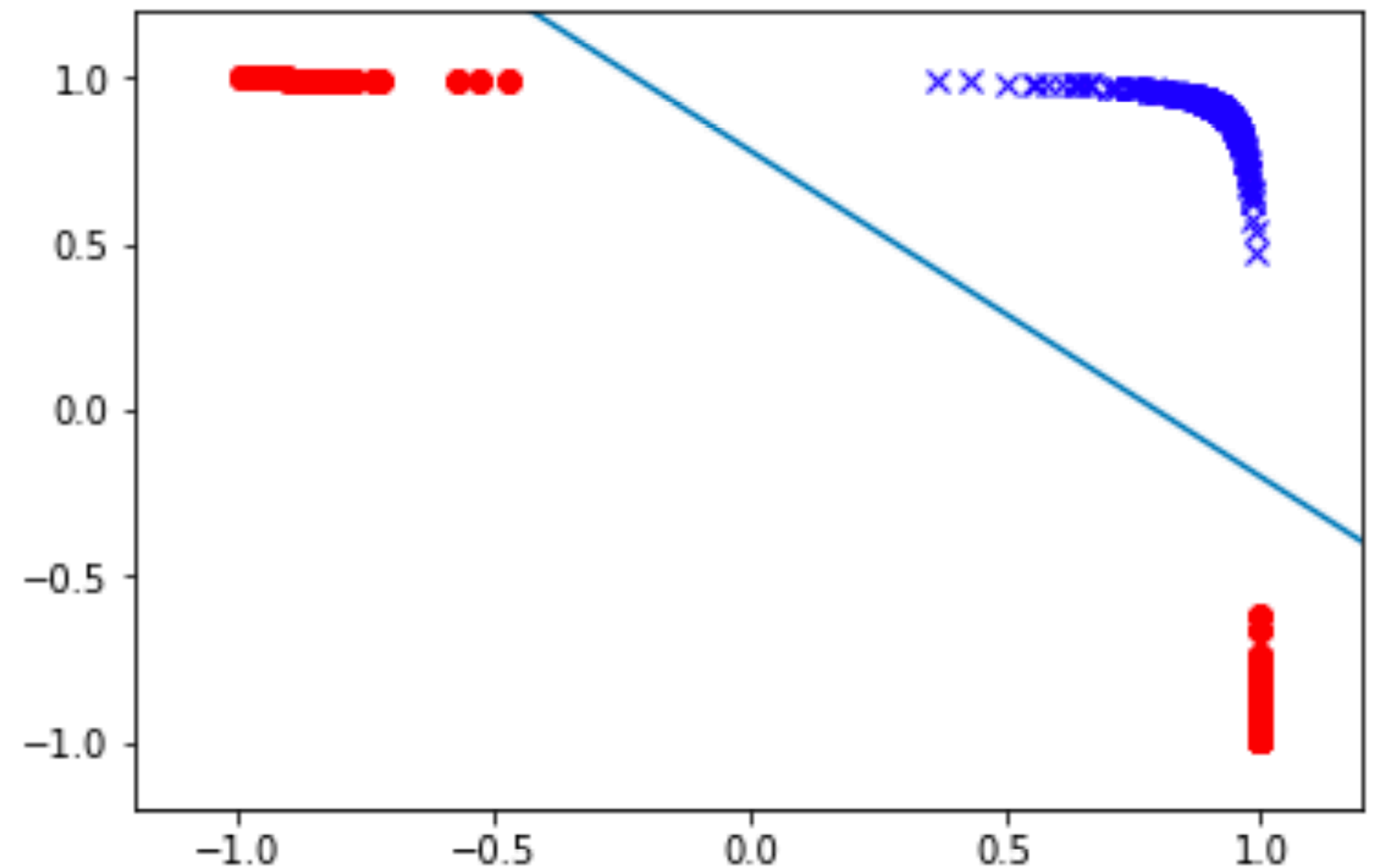
USC Viterbi

Power of non-linearity

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



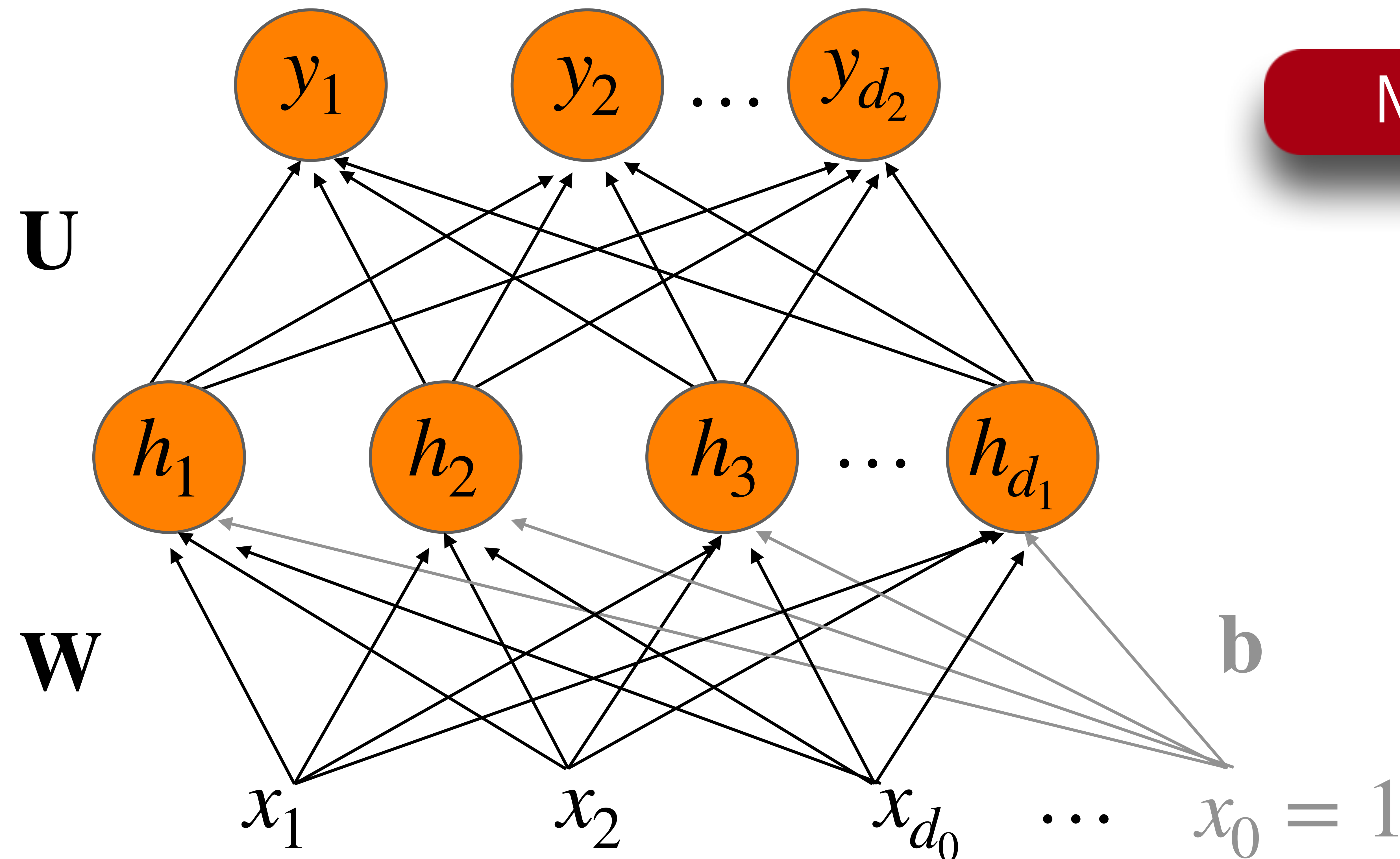
After a $\tanh(\cdot)$ transformation:



Feedforward Neural Nets

Multilayer Perceptron

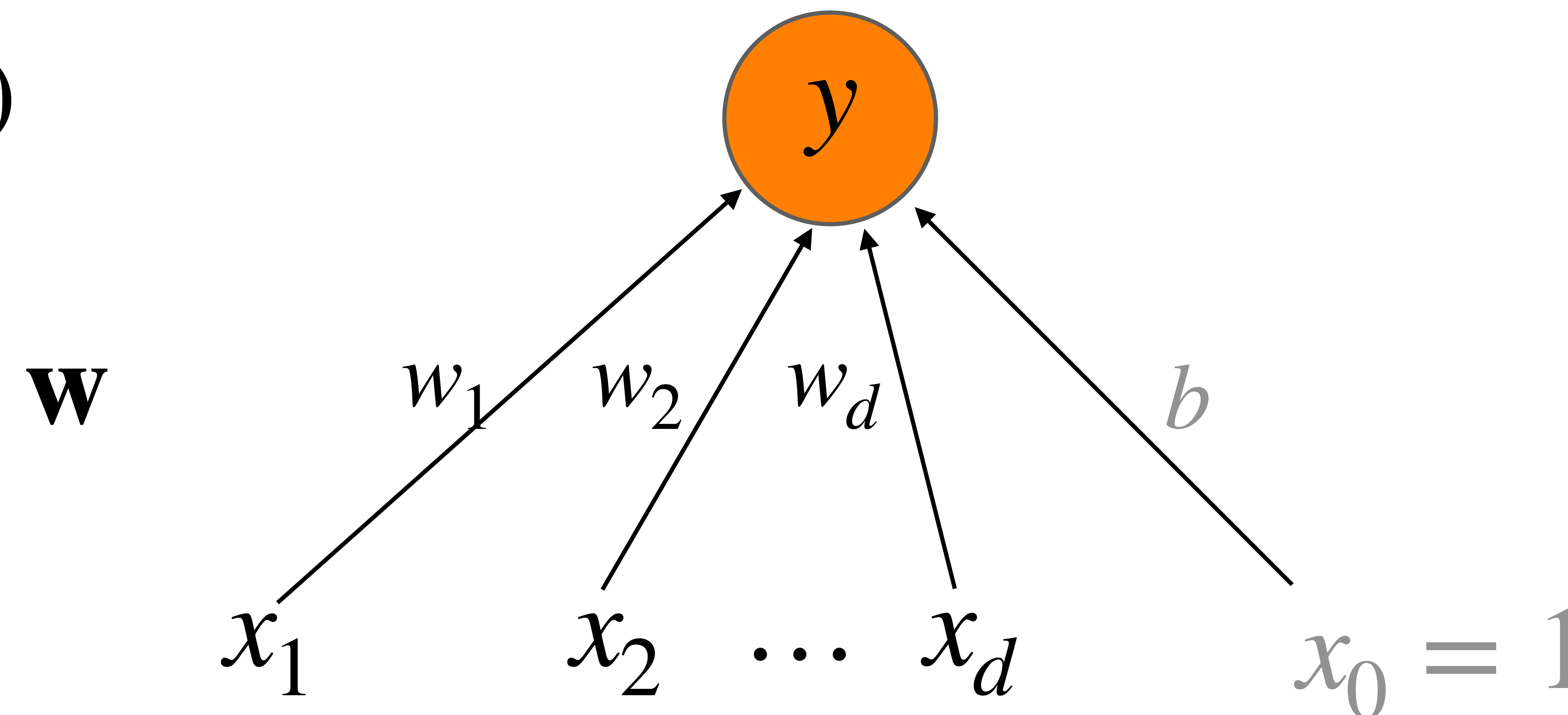
Technically, can learn any function!



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

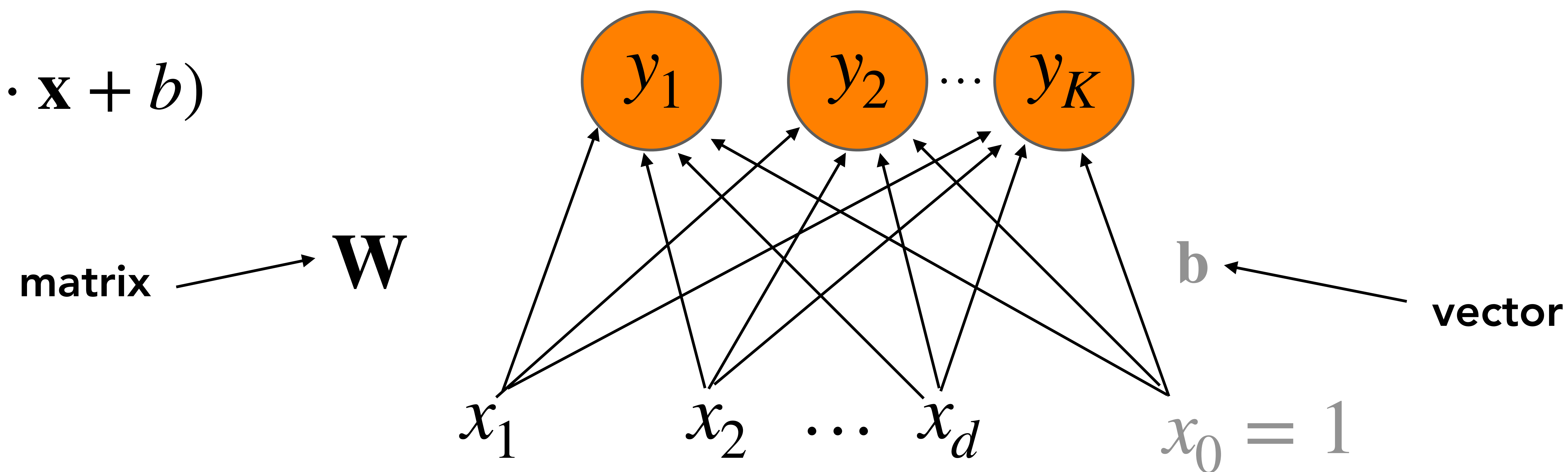
1-layer Network

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

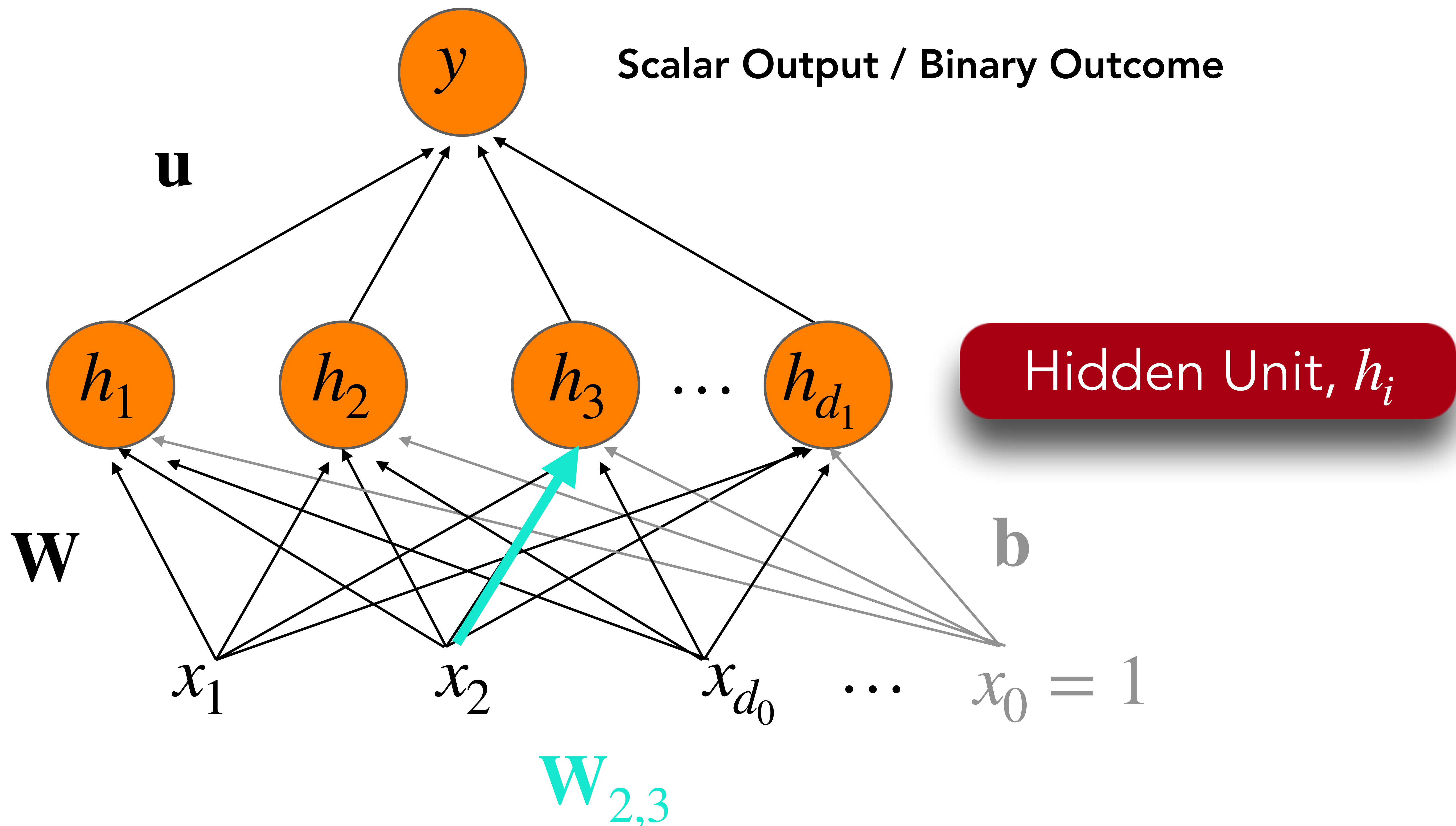
Two-layer Feedforward Network

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

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

Usually ReLU or tanh

Input layer: vector $\mathbf{x}$



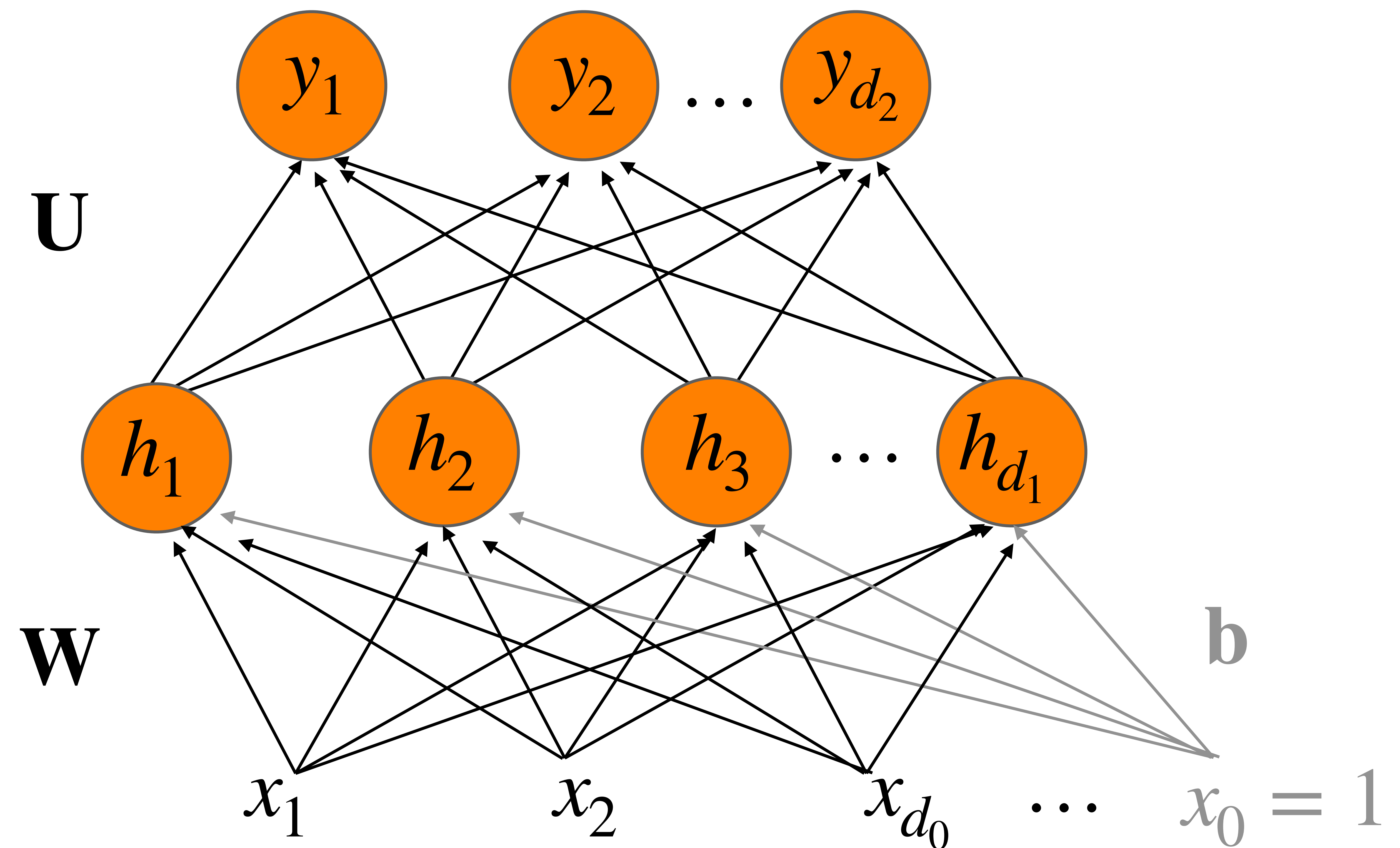
Two-layer Feedforward Network with Softmax Output

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

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

Usually ReLU or tanh

Input layer: vector $\mathbf{x}$



What is $\mathbf{y}$?

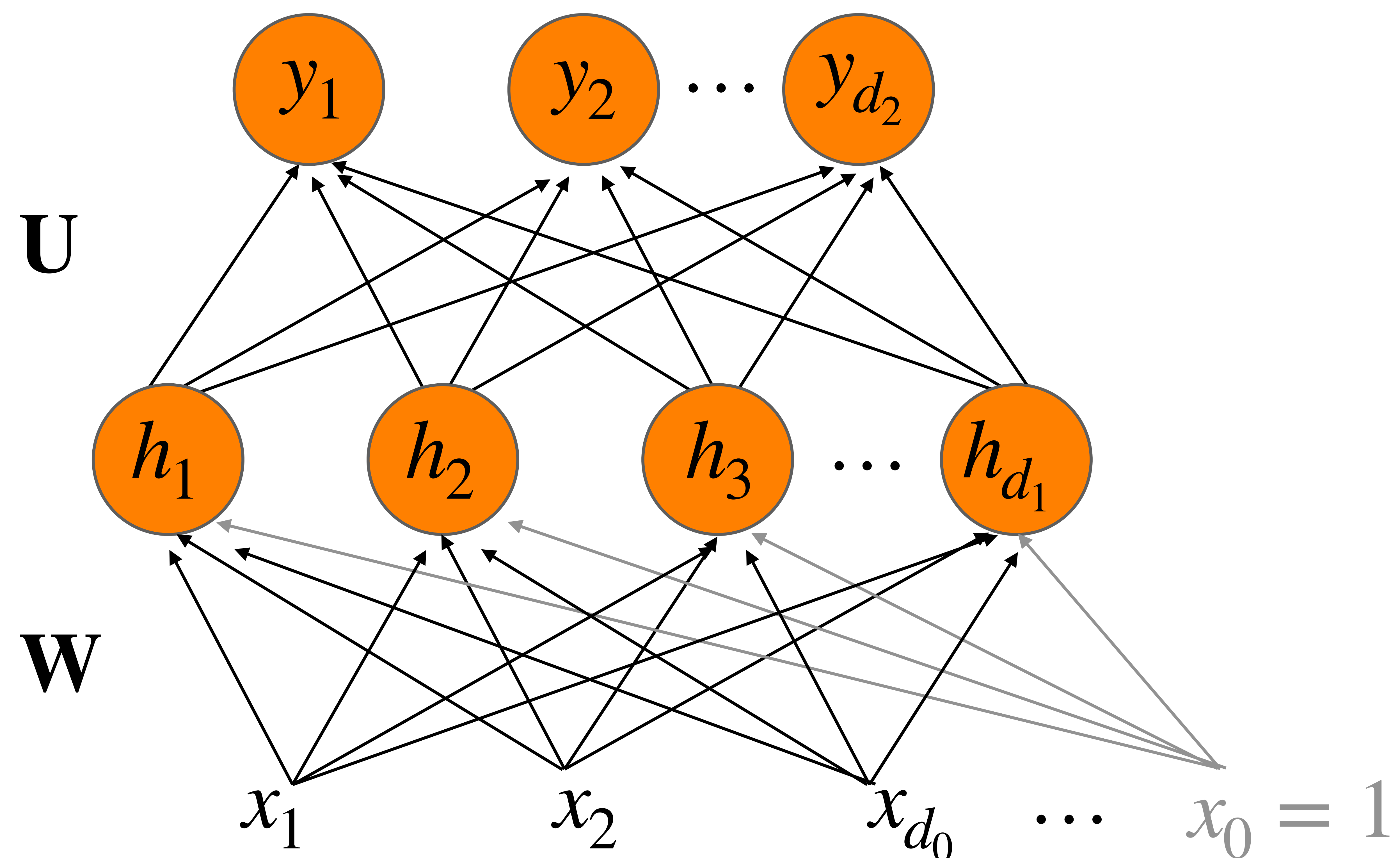
Two-layer FFNN: Notation

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: Logistic Regression and word2vec
- Quiz 2
- Feed-forward Neural Networks
- Feed-forward Language Models
- Training Feed-forward Neural Networks
- Computation Graphs and Backprop

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** can do almost as well!

Why?

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

USC Viterbi

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

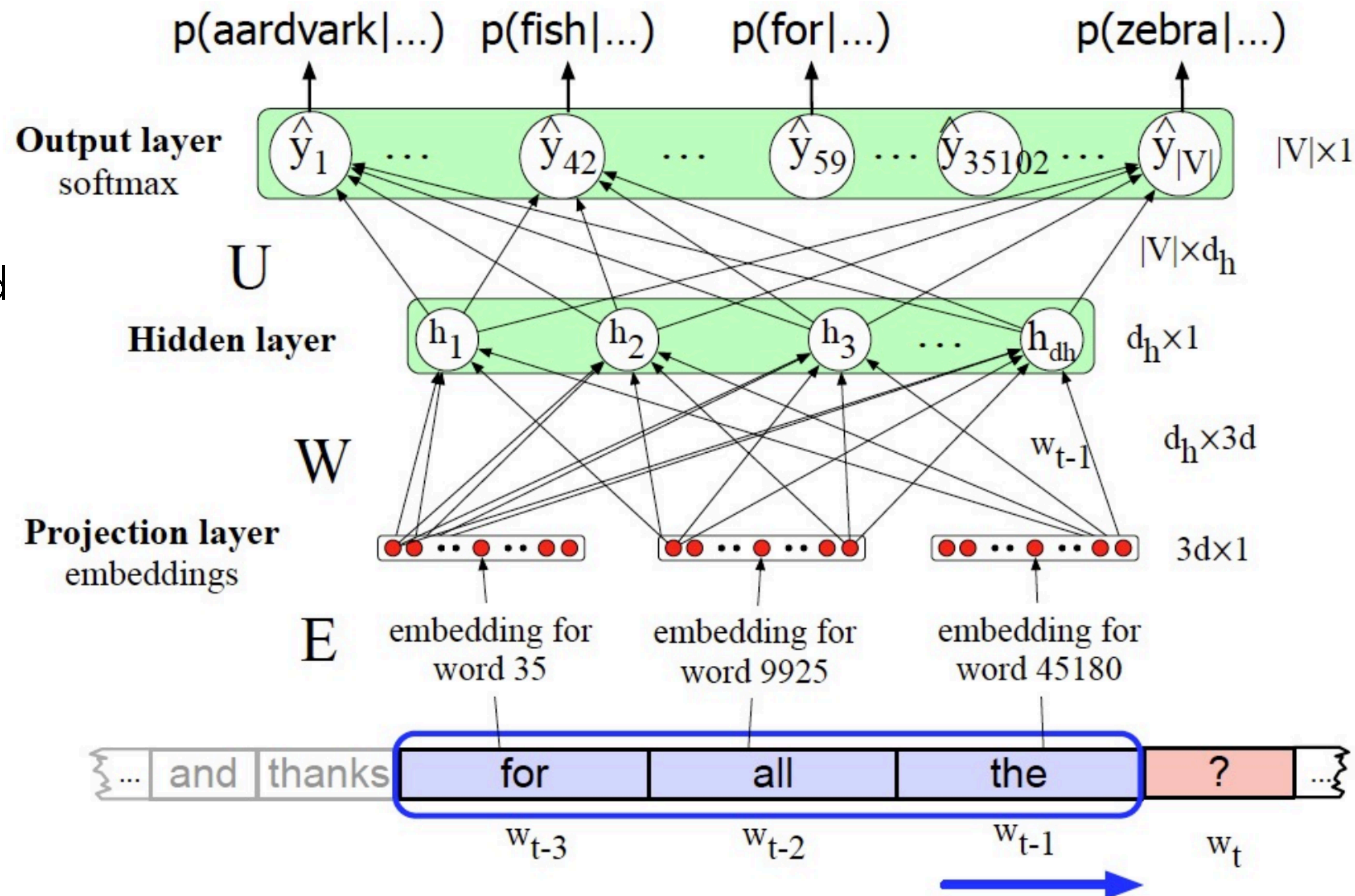
- Self-supervised
- Computation is divided into time steps t, where different sliding windows are considered
- $x_t = (w_{t-1}, \ldots, w_{t-M+1})$ for the context
    - represent words in this prior context by their embeddings, rather than just by their word identity as in n-gram LMs
    - allows neural 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

USC Viterbi

Feedforward Neural LM

- Sliding window of size 4 (including the target word)
- Every feature in the embedding vector connected to every single hidden unit
- Projection / embedding layer is a kind of input layer
  - This is where we plug in our word2vec embeddings
  - May or may not update embedding weights



Simplified Representation

p(aardvark|...) p(fish|...) p(for|...) p(zebra|...)

Output layer softmax: $\hat{y}_1$ ... $\hat{y}_{42}$ ... $\hat{y}_{59}$ ... $\hat{y}_{35102}$ ... $\hat{y}_{|V|}$ — $|V|\times1$

U — $|V|\times d_h$

Hidden layer: h_1 h_2 h_3 ... h_{dh} — $d_h\times1$

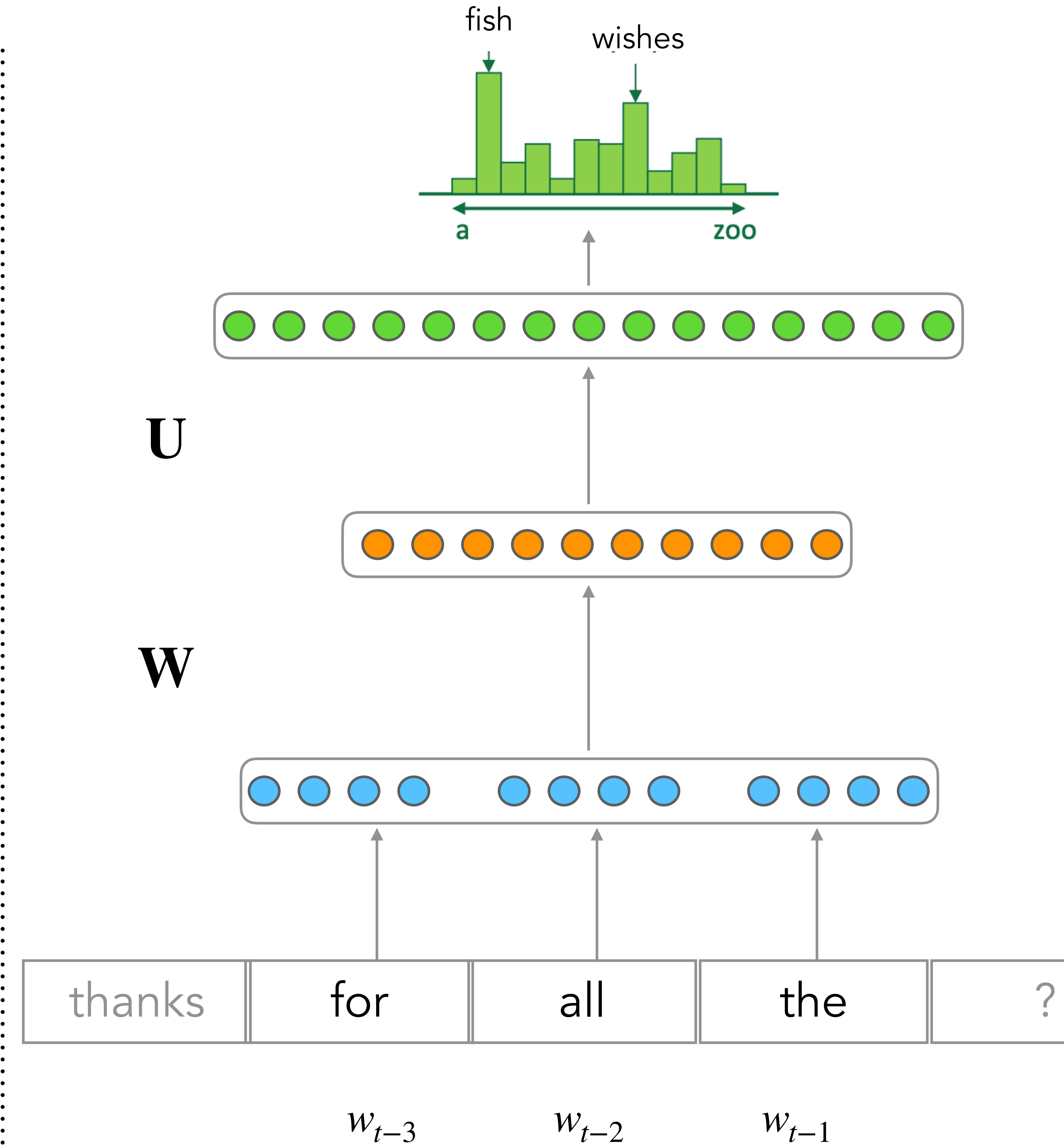
W — $d_h\times3d$

Projection layer embeddings — $3d\times1$

E: embedding for word 35, embedding for word 9925, embedding for word 45180

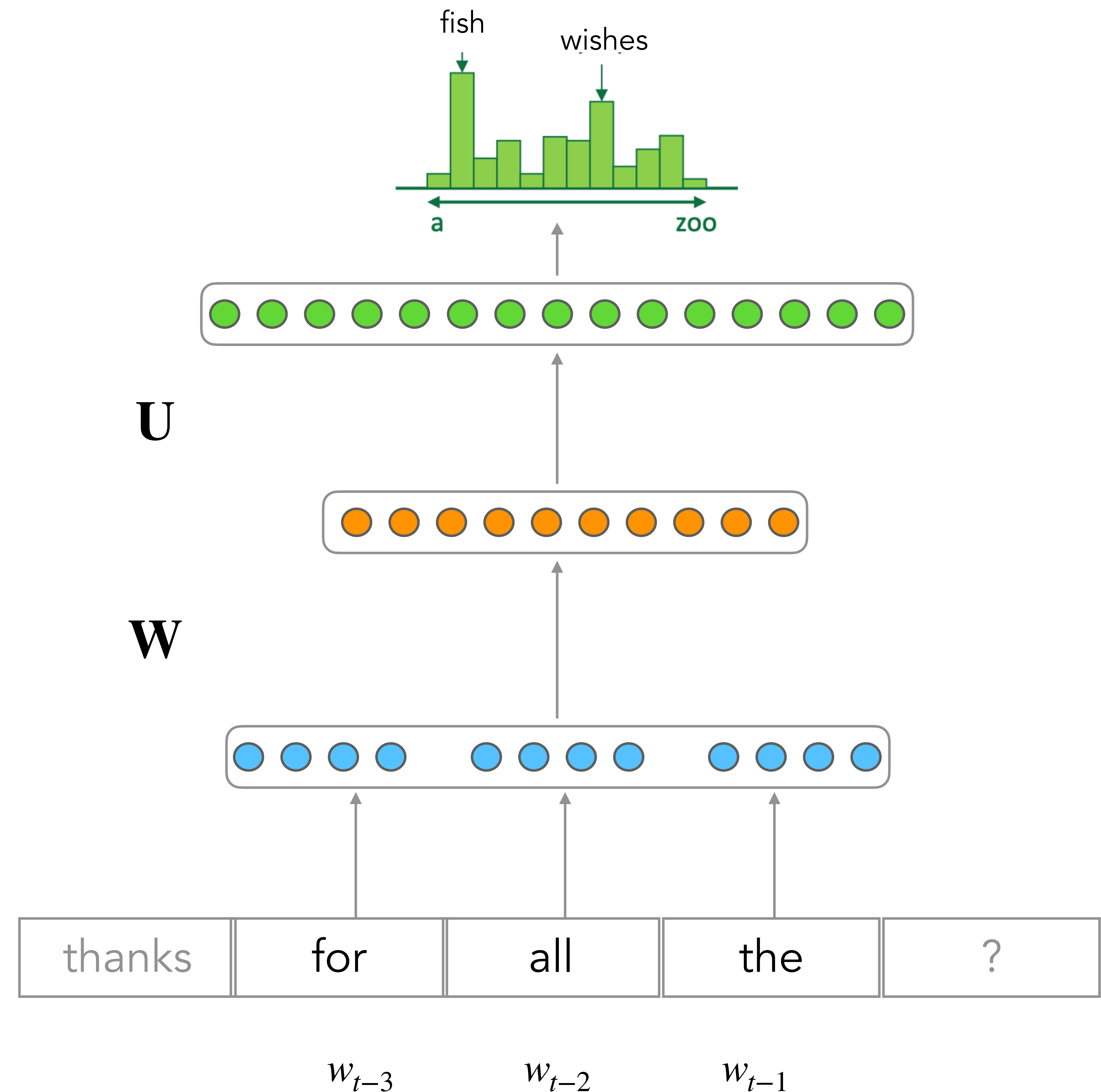
... and thanks | for | all | the | ? | ...

w_{t-3} w_{t-2} w_{t-1} w_t



Feedforward LMs: Windows

- The goodness of the language model depends on the size of the sliding window!
- Fixed window can be too small
- Enlarging window enlarges **W**
- Each word uses different rows of **W**. We don't share weights across the window.
- Window can never be large enough!



Training FFNNs

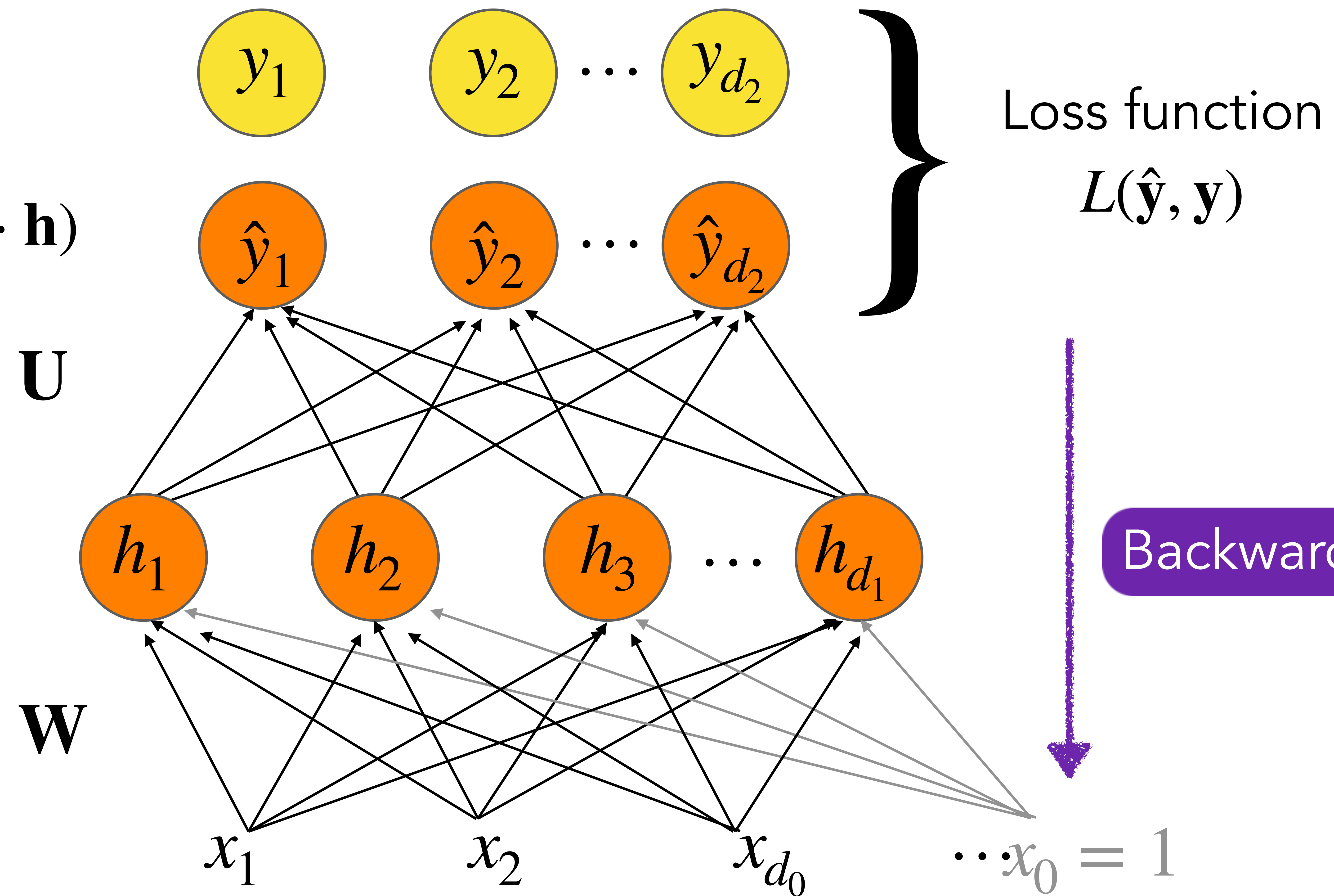
Intuition: Training a 2-layer Network

Training instance $\mathbf{y}$

Model Output $\hat{\mathbf{y}} = \text{softmax}(\mathbf{U} \cdot \mathbf{h})$

Forward Pass

Training instance $\mathbf{x}$



Intuition: Training a 2-layer network

For every training tuple (x, y)

- Run forward computation to find our estimate $\hat{y}$
- Run backward computation to update weights:
    - For every output node
        - Compute loss L between true y and the estimated $\hat{y}$
        - For every weight w from hidden layer to the output layer
            - Update the weight
        - For every hidden node
            - Assess how much blame it deserves for the current answer
            - For every weight w from input layer to the hidden layer
                - Update the weight

LR and FFNN: Similarities and Differences

Cross Entropy Loss again!

$$L_{CE}(y, \hat{y}) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$
$$= -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1-y)\log(\sigma(-\mathbf{w} \cdot \mathbf{x} + b))]$$

Gradient Update

$$\frac{\partial L_{CE}(\hat{y}, y)}{\partial w_j} = [\sigma(\mathbf{w} \cdot \mathbf{x} + b) - y]x_j$$

Only one parameter!

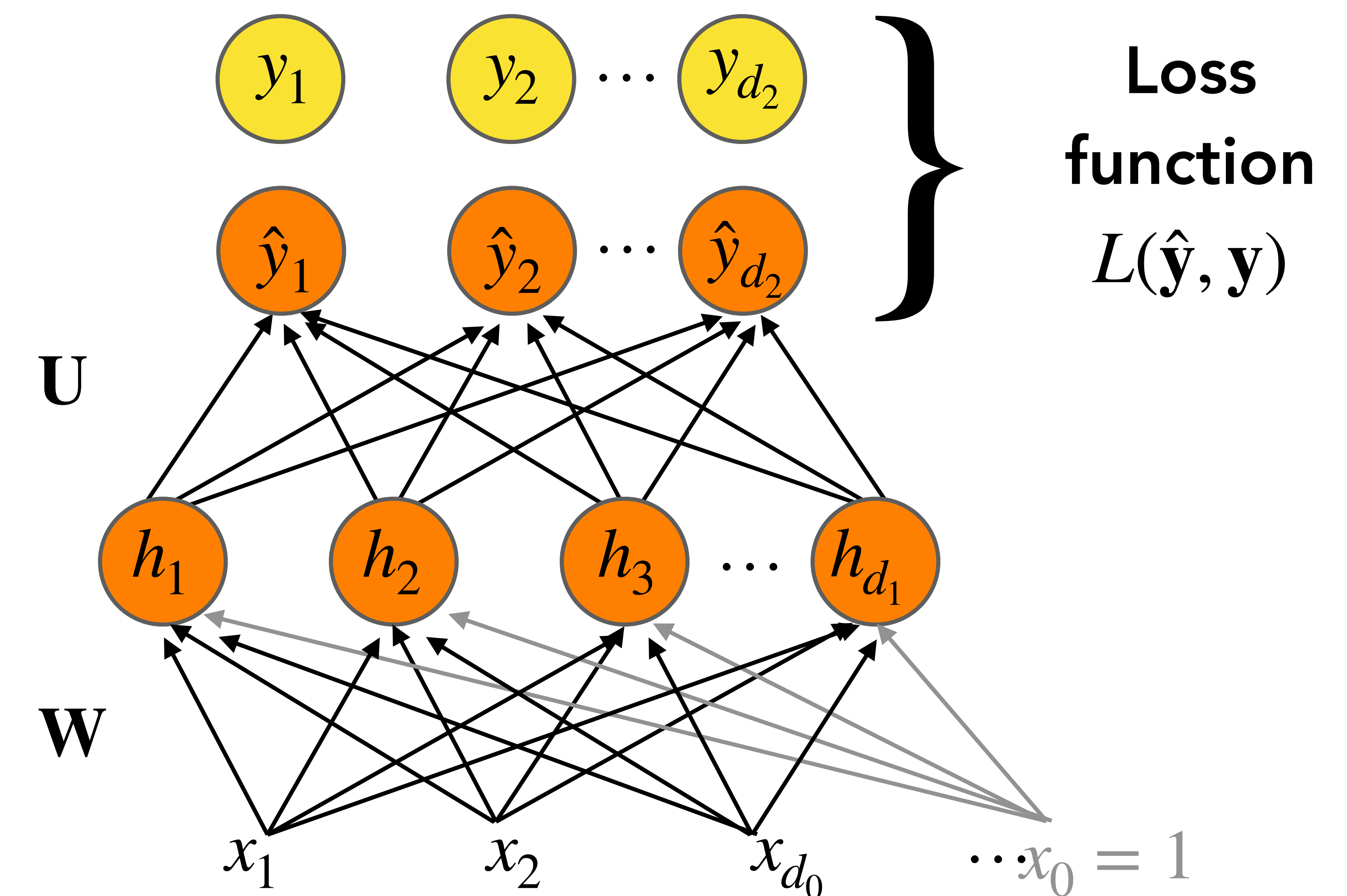
Computation Graphs

As (multiple) hidden layers are introduced, there will be many more parameters to consider, not to mention activation functions!

Computation Graphs and Backprop

Why Computation Graphs?

- For training, we need the derivative of the loss with respect to each weight in every layer of the network
  - But the loss is computed only at the very end of the network!
- Solution: error backpropagation or backward differentiation
  - Backprop is a special case of backward differentiation
    - Which relies on computation graphs



Backprop

Graph representing the process of computing a mathematical expression

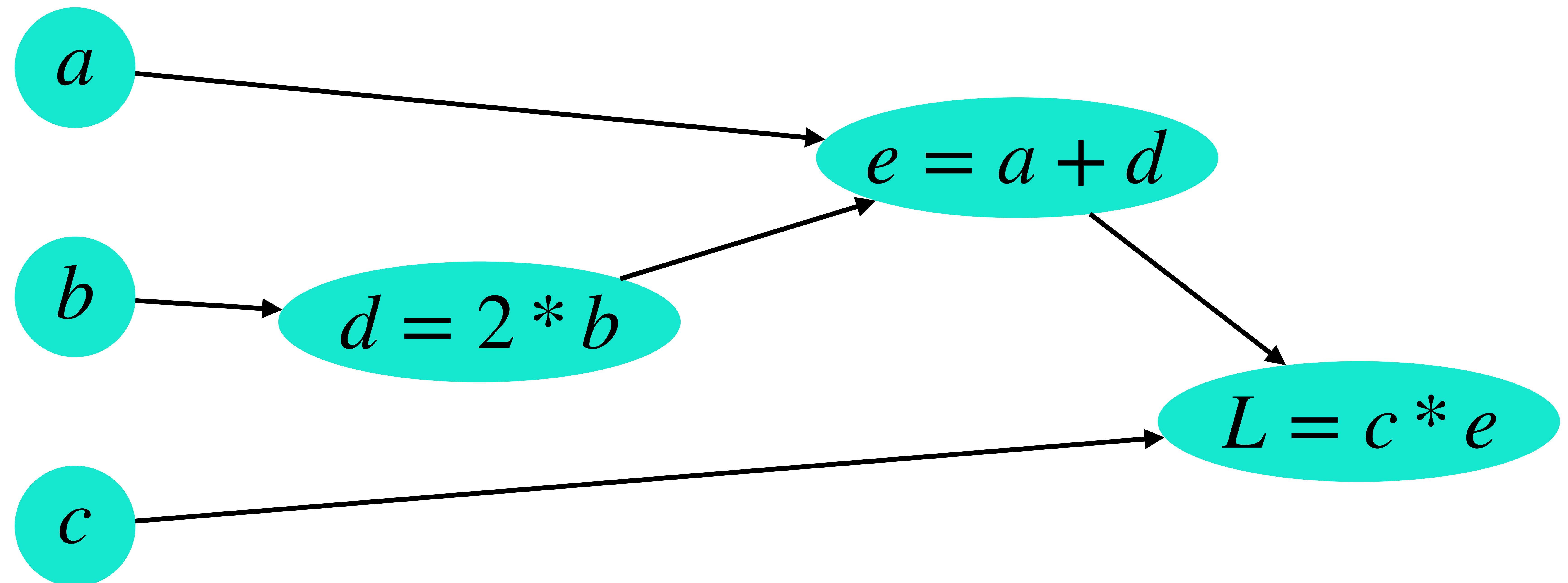
Rumelhart, Hinton, Williams, 1986

Example: Computation Graph

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$



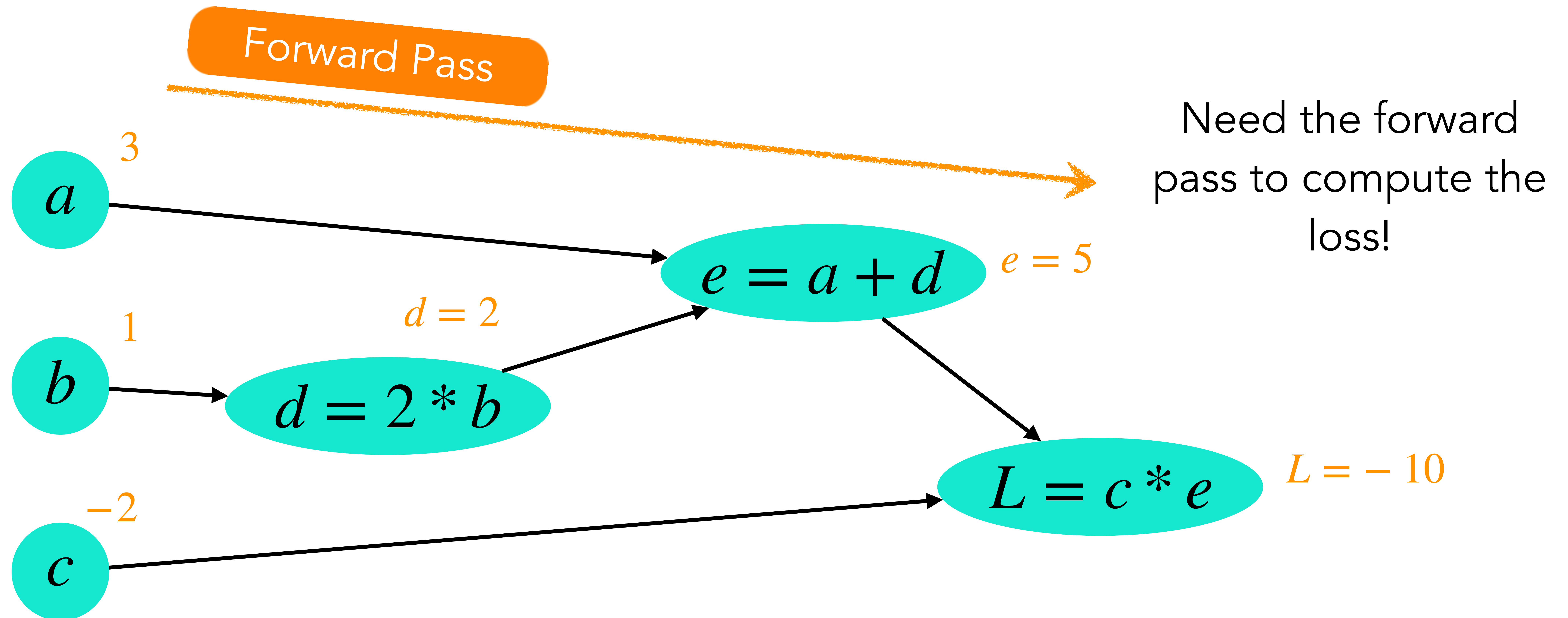
USC Viterbi

Example: Forward Pass

$d = 2 * b$

$e = a + d$

$L = c * e$



Need the forward pass to compute the loss!

But how to compute parameter updates?

Example: Backward Pass Intuition

- The importance of the computation graph comes from the **backward pass**
- Used to compute the derivatives needed for the weight updates

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

$$\frac{\partial L}{\partial a} = ? \quad \frac{\partial L}{\partial b} = ? \quad \frac{\partial L}{\partial c} = ?$$

Input Layer Gradients

Hidden Layer Gradients

$$\frac{\partial L}{\partial d} = ? \quad \frac{\partial L}{\partial e} = ?$$

Chain Rule of Differentiation!

The Chain Rule

Computing the derivative of a composite function:

$$f(x) = u(v(x)) \qquad \frac{\partial f}{\partial x} = \frac{\partial u}{\partial v}\frac{\partial v}{\partial x}$$

$$f(x) = u(v(w(x))) \qquad \frac{\partial f}{\partial x} = \frac{\partial u}{\partial v}\frac{\partial v}{\partial w}\frac{\partial w}{\partial x}$$

Example: Applying the chain rule

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

$$\frac{\partial L}{\partial c} = e$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial d} \frac{\partial d}{\partial b}$$

$$\frac{\partial L}{\partial e} = c$$

$$\frac{\partial L}{\partial d} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial d}$$

Cannot do all at once, need to follow an order…

Example: Backward Pass

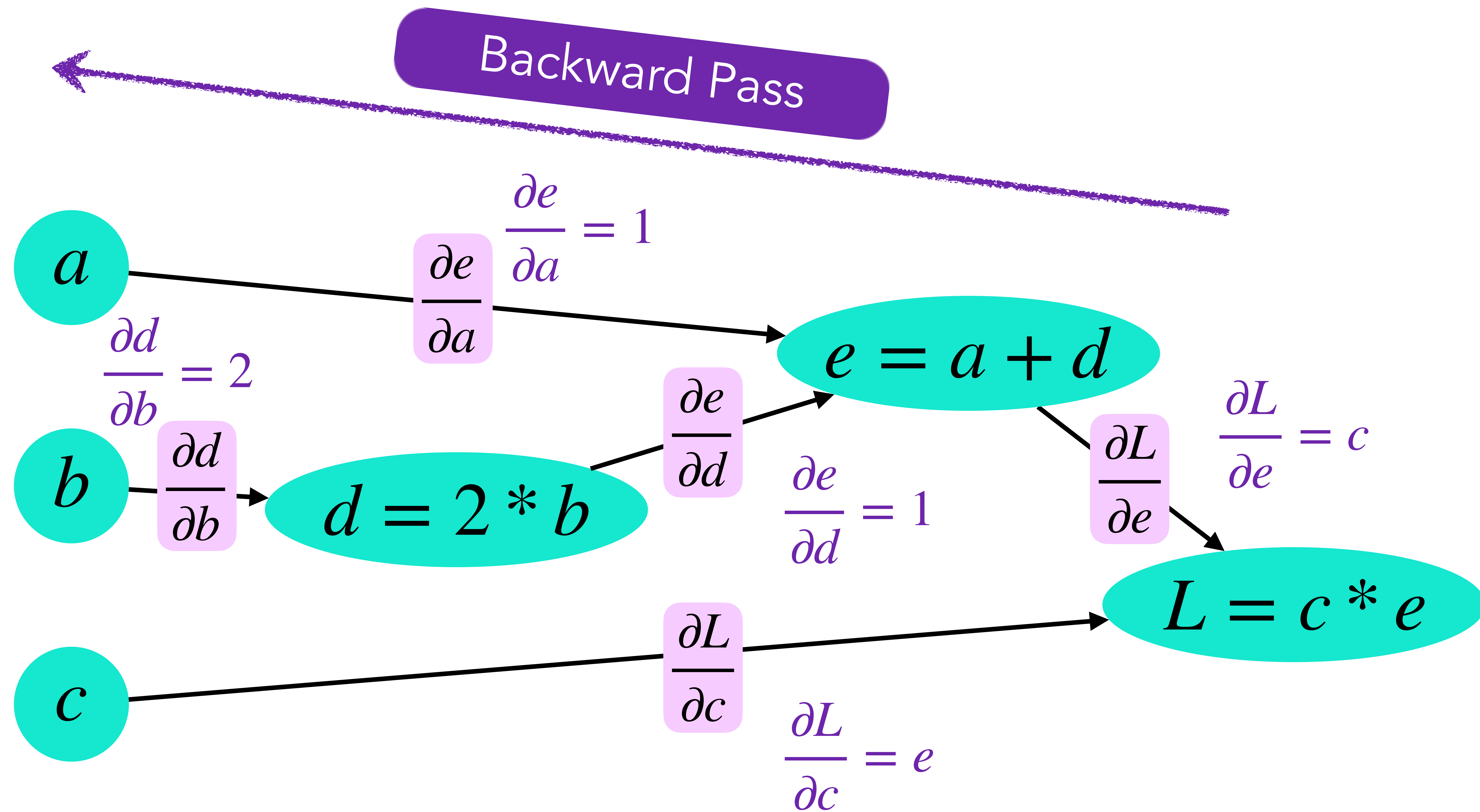
But we need the gradients of the loss with respect to parameters...

$$\frac{\partial L}{\partial c} = e \qquad \frac{\partial L}{\partial e} = c$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial d} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$



USC Viterbi

Example

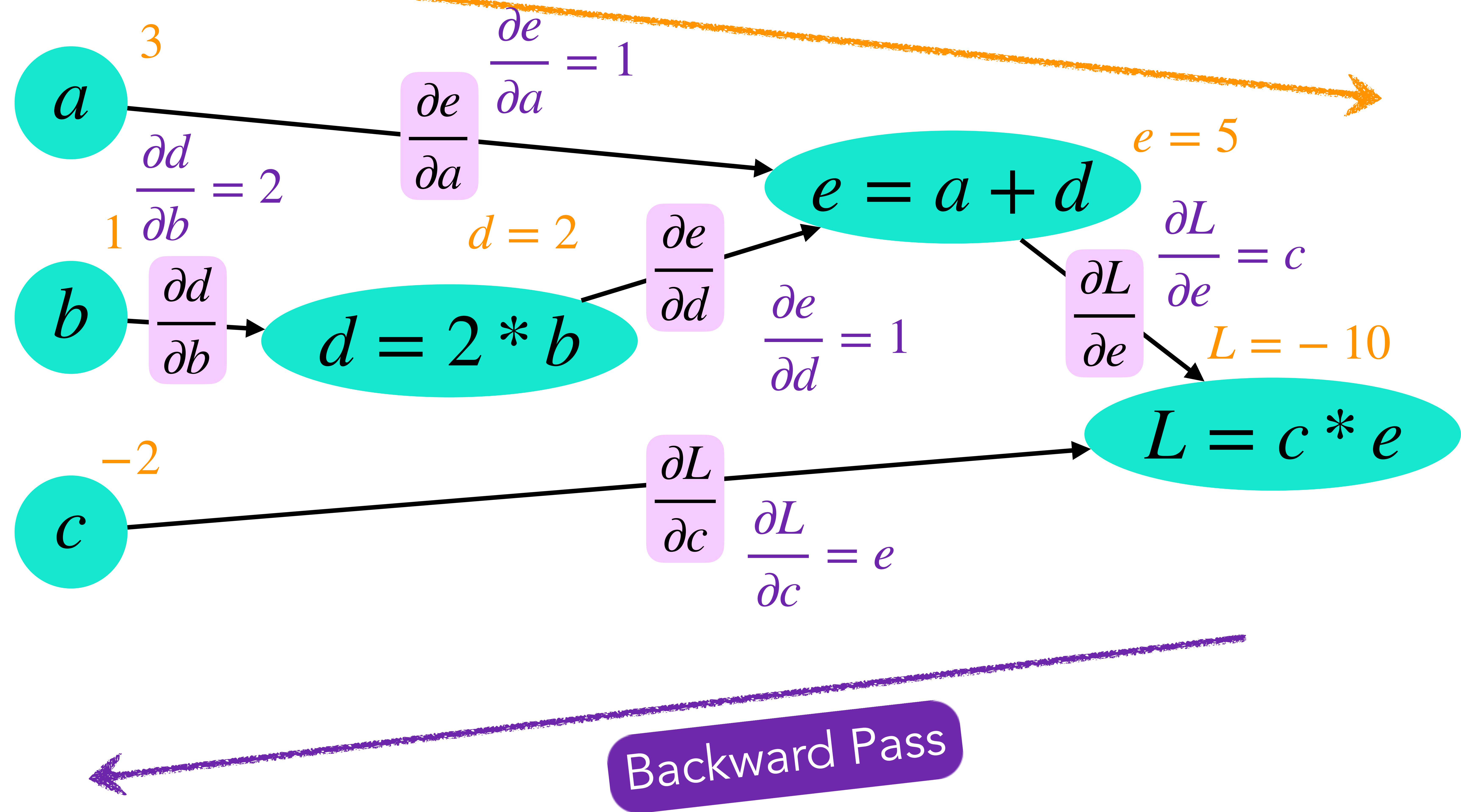
$$\frac{\partial L}{\partial e} = c = -2$$

$$\frac{\partial L}{\partial c} = e = 5$$

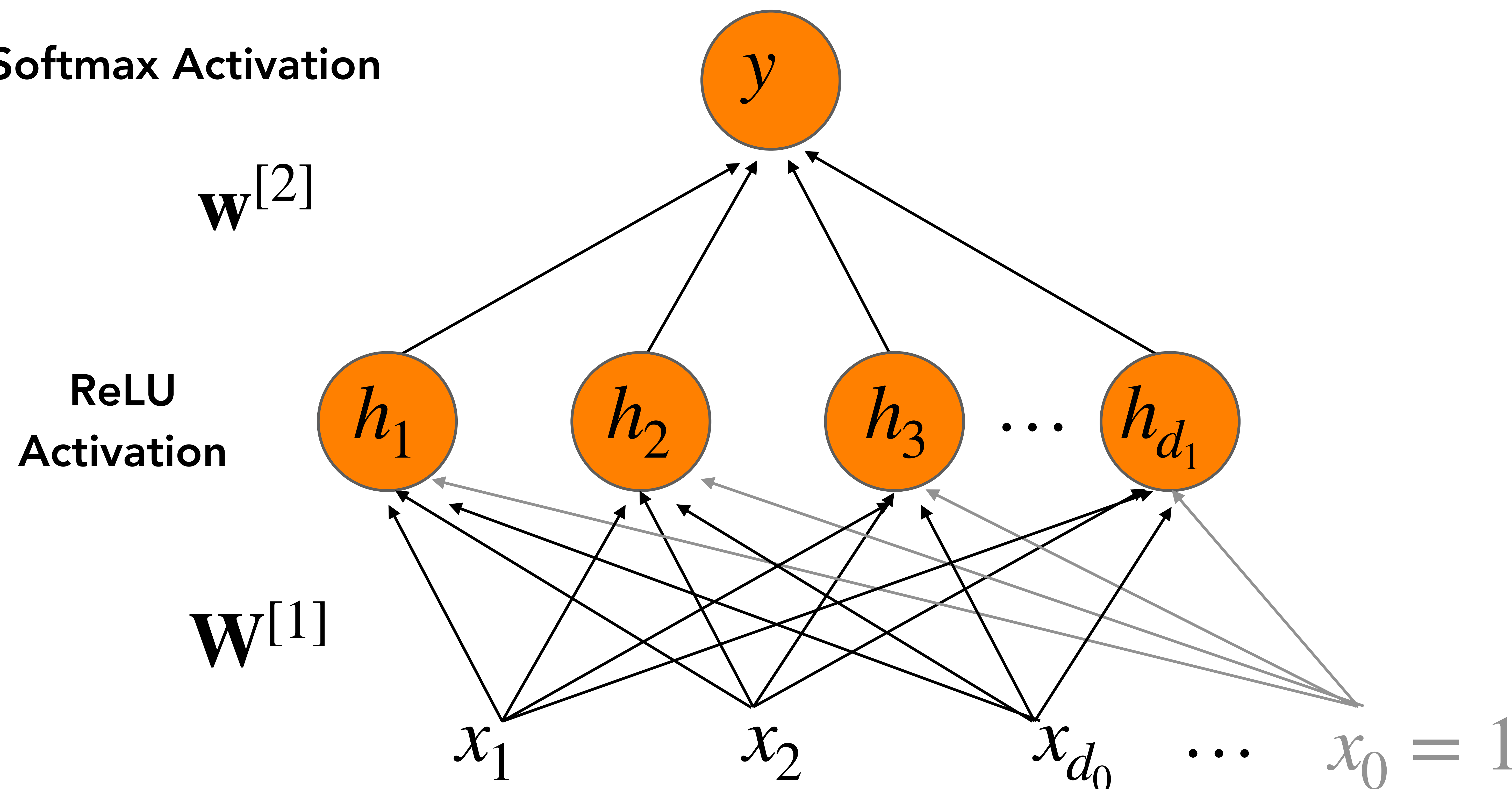
$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} = -2$$

$$\frac{\partial L}{\partial d} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d} = -2$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b} = -4$$



Backward Differentiation on a 2-layer MLP



$$\hat{y} = \sigma(z^{[2]})$$

$$z^{[2]} = \mathbf{w}^{[2]} \cdot \mathbf{h}^{[1]}$$

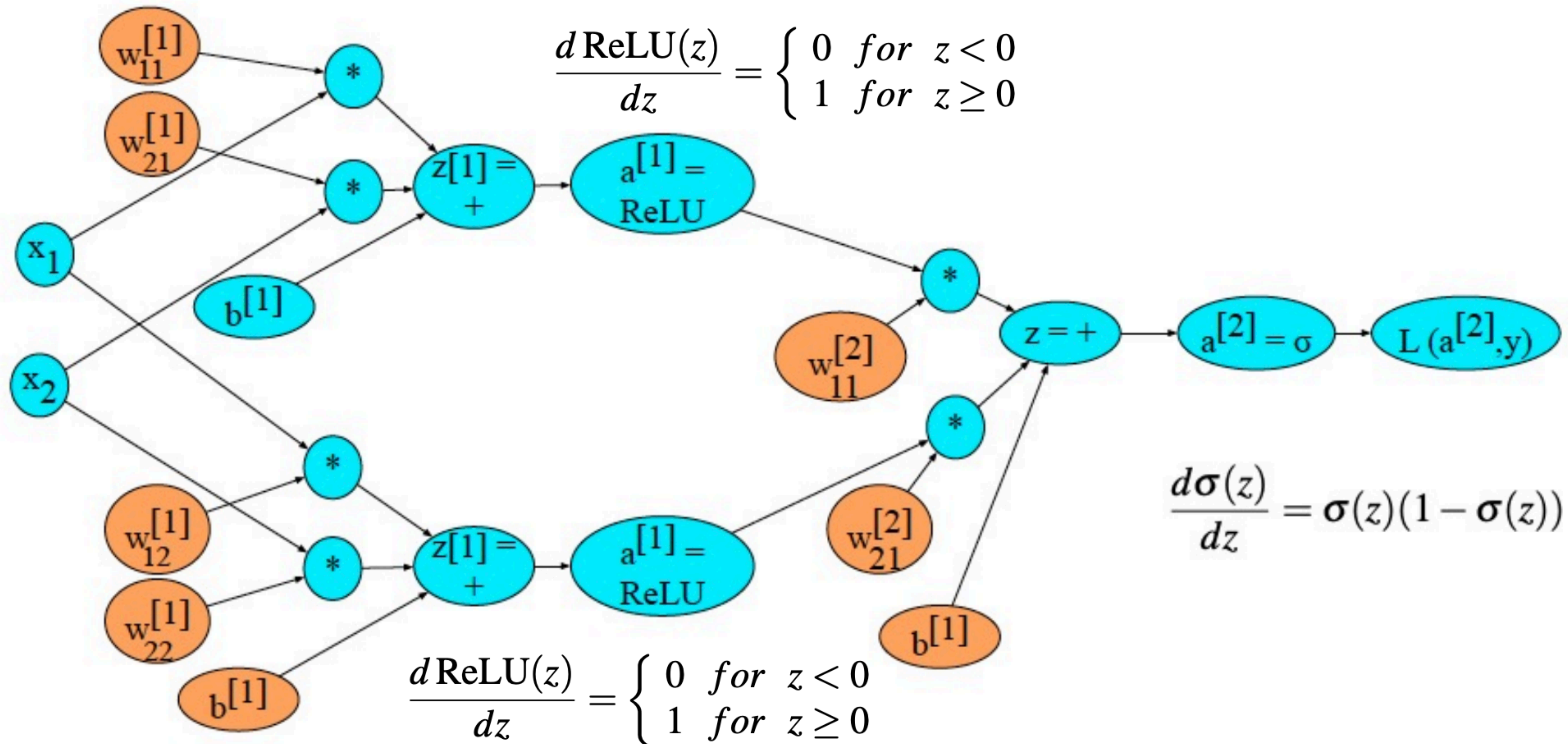
$$\mathbf{h}^{[1]} = \textbf{ReLU}(\mathbf{z}^{[1]})$$ Element-wise

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x}$$

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)\sigma(-z) = \sigma(z)(1 - \sigma(z))$$

$$\frac{d\,\text{ReLU}(z)}{dz} = \begin{cases} 0 & for \quad z < 0 \\ 1 & for \quad z \geq 0 \end{cases}$$

2 layer MLP with 2 input features



Starting off the backward pass: $\frac{\partial L}{\partial z}$

(I'll write a for $a^{[2]}$ and z for $z^{[2]}$)

$$\begin{aligned} z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\ a^{[1]} &= \text{ReLU}(z^{[1]}) \\ z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} &= \sigma(z^{[2]}) \\ \hat{y} &= a^{[2]} \end{aligned}$$

$$L(\hat{y}, y) = -(\mathrm{y} \log(\hat{y}) + (1 - y)\log(1 - \hat{y}))$$

$$L(a, y) = -(\mathrm{y} \log a + (1 - y)\log(1 - a))$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z}$$

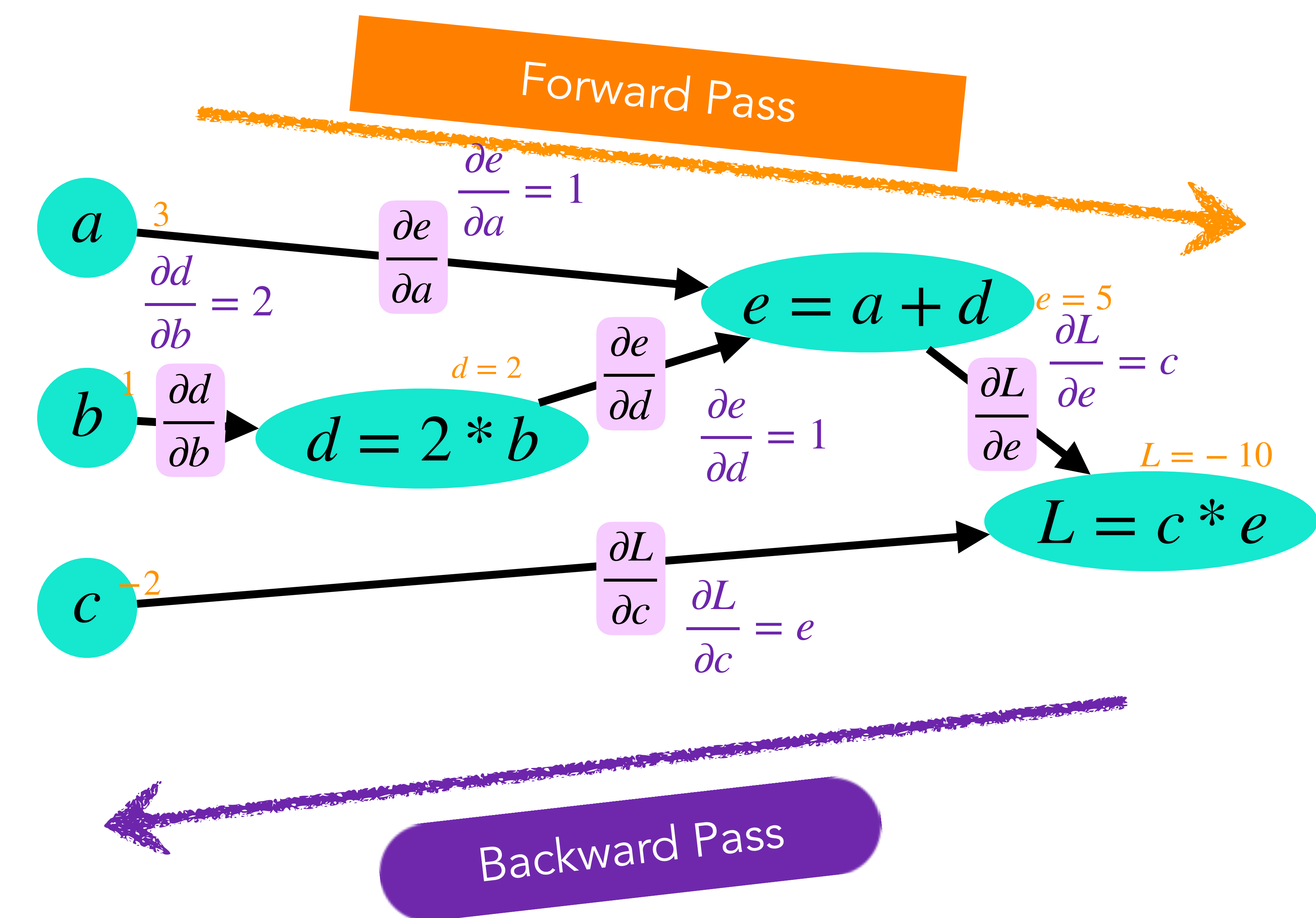
$$\frac{\partial L}{\partial a} = -\left(\left(y \frac{\partial \log(a)}{\partial a}\right) + (1 - \mathrm{y}) \frac{\partial \log(1 - a)}{\partial a}\right)$$

$$= -\left(\left(y \frac{1}{a}\right) + (1 - \mathrm{y}) \frac{1}{1 - a}(-1)\right) = -\left(\frac{y}{a} + \frac{y - 1}{1 - a}\right)$$

$$\frac{\partial a}{\partial z} = a(1 - a) \qquad \frac{\partial L}{\partial z} = -\left(\frac{y}{a} + \frac{y - 1}{1 - a}\right) a(1 - a) = \mathrm{a} - \mathrm{y}$$

Summary: Backprop / Backward Differentiation

- For training, we need the derivative of the loss with respect to weights in early layers of the network
  - But loss is computed only at the very end of the network!
- Solution: **backward differentiation**



Given a computation graph and the derivatives of all the functions in it we can automatically compute the derivative of the loss with respect to these early weights.

Libraries such as PyTorch do this for you in a single line: `model.backward()`