Lecture 03:
Feed Forward Neural Networks & RNNs

Xiang Ren

USC CSCI 662 Advanced NLP

Spring 2026

Some slides adapted from Chris Manning and Dan Jurafsky

Announcements

Announcements + Logistics

- Project team formation due today 11:59pm PST!
    - Teaming form: https://forms.gle/sT5M6zb7Eq3L4jNG6
- Project Pitch in next class:
    - Add your slides to the master deck by next Tue 11:59pm.
- Paper selection for presentation & learning discussions
    - Due: Feb 4 11:59 PST.

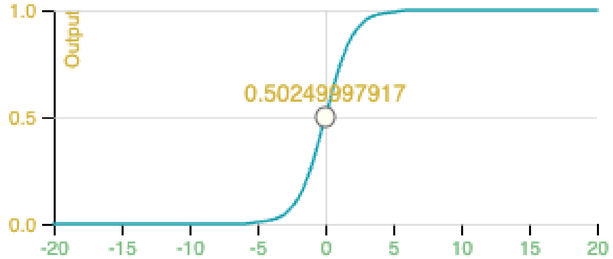
Paper Presentation

- list of suggested papers on 10 different timely topics in NLP/LLMs from last year.
- You can start "claiming" your paper to present (or) by commenting on that paper entry with your name+your claim (presenting, or leading discussion).
- As introduced earlier, each student will complete **one in-depth research paper presentation** (15 mins presentation + 5 mins discussion) during the semester using slides, and will also serve as a **designated discussion lead for two additional paper presentations**.
- The presentation will be assessed on the student's ability to clearly explain the paper's motivation, technical approach, and key results; critically evaluate its assumptions, limitations, and contributions; and situate the work within the broader LLM and NLP research landscape (learning objective O2 & O3).
- Students will also be evaluated on the quality of the discussion they facilitate—through insightful questions, engagement with peers, and ability to surface open problems and future research directions—both during their own presentation and when leading discussions for others.
- **The slides of the presentation need to be shared to the class a day before the presentation.**
- Will shuffle the order and release the order this Friday.

Supervised Machine Learning 101

Binary Text Classification

- Goal: Given an input, predict label or class from a discrete set
  - e.g. Predict the sentiment (positive or negative) for a sentence
- Input: x represented by feature vector of size d, given by $\mathbf{x} \in \mathbb{R}^d$
- Output: $y \in \{0,1\}$ for binary classification
- Suffices to learn conditional probabilities
  - Parameterized by $\theta \in \mathbb{R}^d$
- Could estimate by cooccurrence counts, but a single feature
  - Better option: dot product (assigning a weight to every feature)
  - Returns a real value: $z \in \mathbb{R}$
- How to get a probability?
  - Consider the Sigmoid function:
- Argmax for prediction:



Logistic Regression

$$P(y|\mathbf{x};\theta)$$

$$z = \theta \cdot \mathbf{x}$$

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

$$P(y=1|\mathbf{x};\theta) = \sigma(\theta \cdot \mathbf{x})$$

$$P(y=0|\mathbf{x};\theta) = 1 - \sigma(\theta \cdot \mathbf{x}) = \sigma(-\theta \cdot \mathbf{x})$$

$$\hat{y} = \arg\max_{y' \in \{0,1\}} P(y'|\mathbf{x};\theta)$$

Ingredients of Supervised Machine Learning

I. Data as pairs $(x^{(i)}, y^{(i)}) \quad s.t \;\; i \in \{1 \ldots N\}$

- $x^{(i)}$ usually represented by a feature vector $\mathbf{x}^{(i)} = [x_1, x_2, \ldots, x_d]$,
  - 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

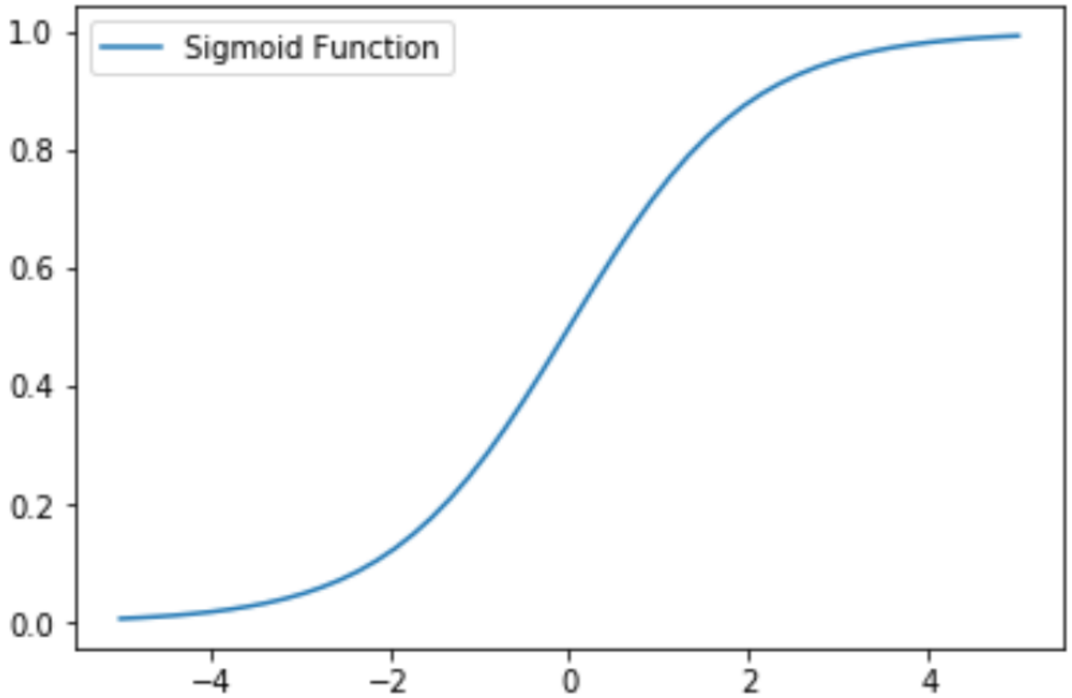
Learning Phase

Learning vs. Inference

- Learning: we learn parameter weights by minimizing the loss function using an optimization algorithm
- Inference: Given a test example x_{test} we compute $p(y|x)$ using learned weights and return whichever label receives higher probability
- Distinct from training and evaluation
  - Evaluation only contains inference; no parameters are updated
  - Training contains both learning and inference

Simple Example: Logistic Regression

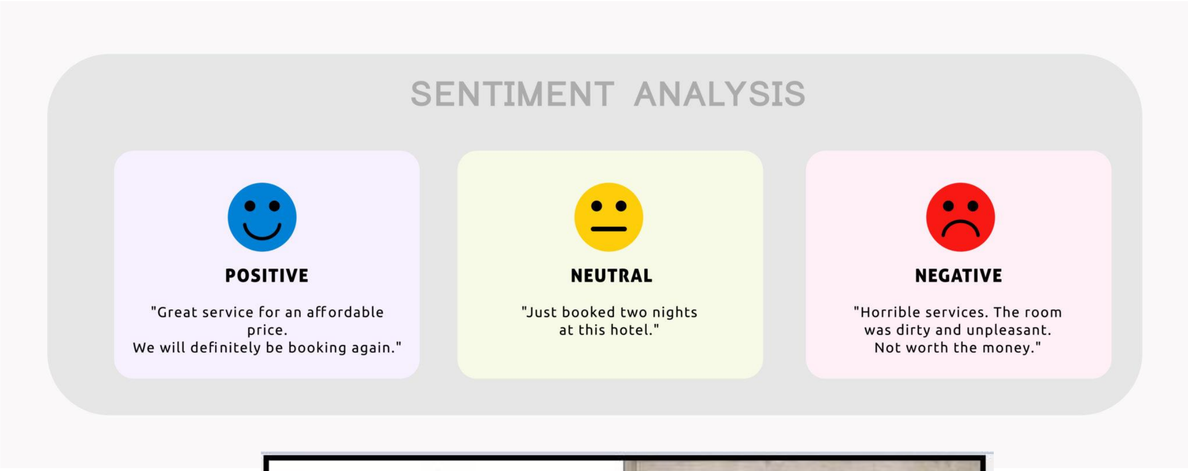
- Important analytic tool in natural and social sciences
- Baseline supervised machine learning tool for classification
- Is also the foundation of neural networks
- Logistic regression is a discriminative classifier
  - Learn a model that can (given the input) distinguish between different classes

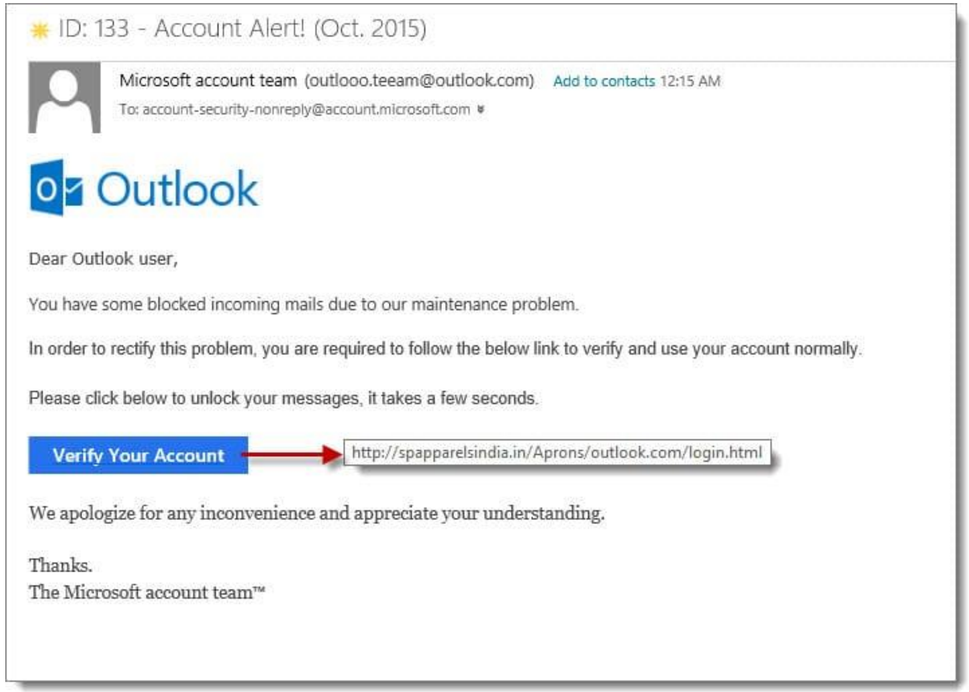


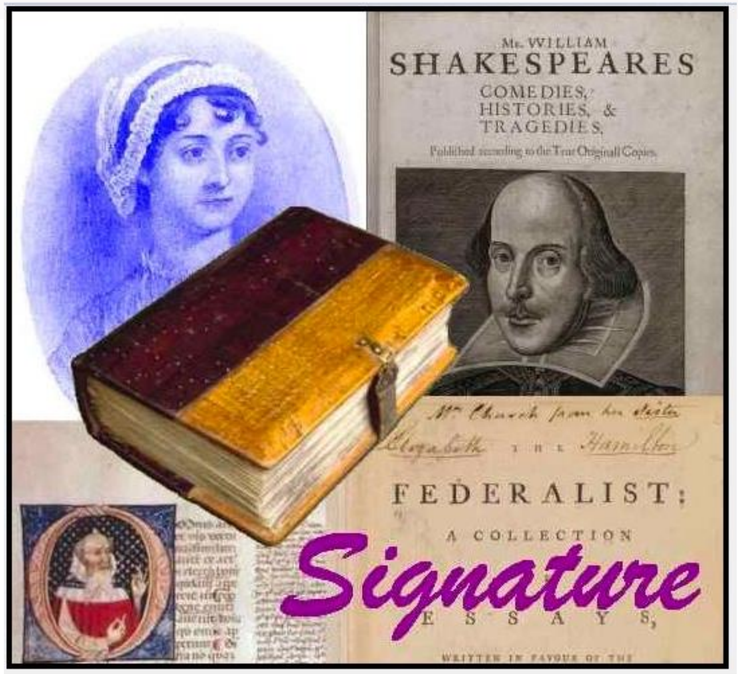
Is language modeling a classification task?

I. Data

Examples of Classification Tasks







Not just NLP, Logistic Regression is a general ML technique often applied across a wide variety of prediction tasks!

Text Classification Setup

- Input:
  - a document x
    - Each observation $x^{(i)}$ is represented by a feature vector $\mathbf{x}^{(i)} = [x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)}]$
  - a label y from a fixed set of classes $C = c_1, c_2, \ldots, c_J$
- Output: a predicted class $\hat{y} \in C$
- Setting for Binary Classification: given a series of input / output pairs:
  - $(x^{(i)}, y^{(i)})$ where label $y^{(i)} \in C = \{0,1\}$
- Goal of Binary Classification
  - At test time, for input x^{test}, compute an output: a predicted class $\hat{y}^{test} \in \{0,1\}$

Features in Classification

- Examples of feature x_i
  - x_i ="review contains 'awesome'"
  - x_j ="review contains 'abysmal'"
  - x_k ="review contains 'mediocre'"

$$w_i = +10$$
$$w_j = -10$$
$$w_k = -2$$

- Each x_i is associated with a weight w_i which determines how important x_i is
  - (For predi

Can you guess the w for x_l = "review contains 'restaurant'"?

II. Model: Logistic Regression

How to get the right y?

- For each feature x_i, introduce a weight w_i, which determines the importance of x_i
  - Plus we will have a bias term b
  - Together, all parameters can be termed as $\theta = [w; b]$
- We consider the weighted sum of all features and the bias

$$z = \left(\sum_d w_d x_d + b\right)$$

$$= \mathbf{w} \cdot \mathbf{x} + b$$

If high, $\hat{y} = 1$ If low, $\hat{y} = 0$

But how to determine the threshold?

Probabilistic Models!

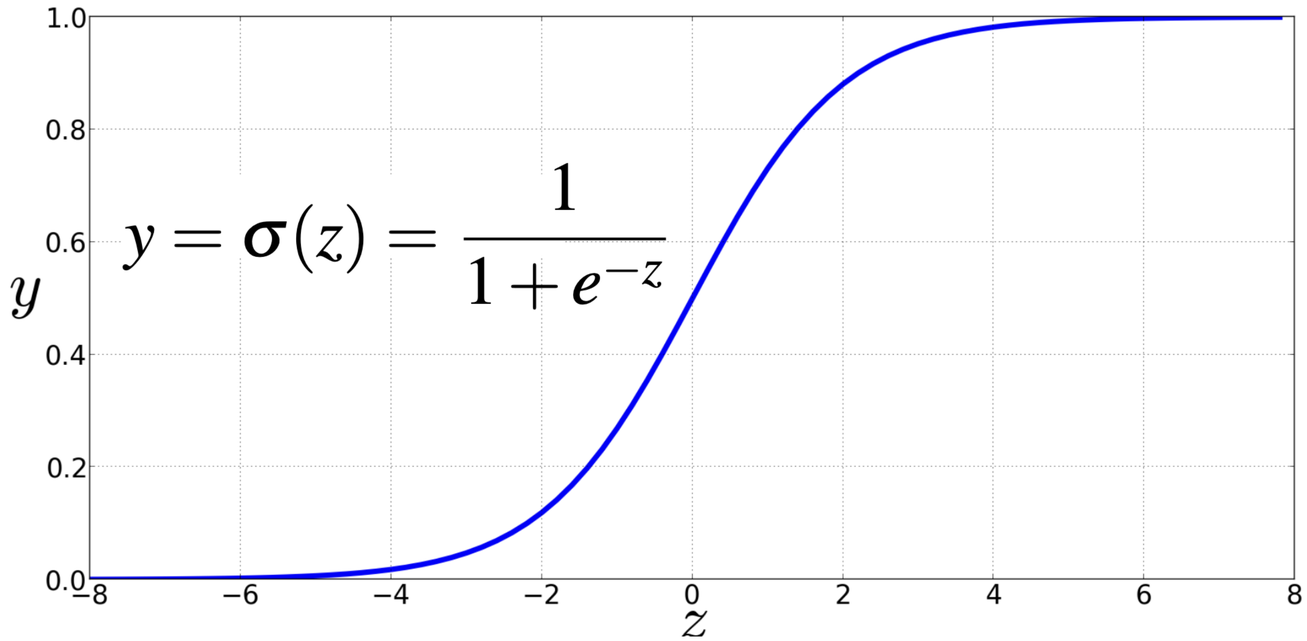
$$P(y = 1|\mathbf{x}; \theta)$$

$$P(y = 0|\mathbf{x}; \theta)$$

Solution: Squish it into the 0-1 range

$$z = \mathbf{w} \cdot \mathbf{x} + b \qquad z \in \mathbb{R}$$

- Sigmoid Function, $\sigma(\cdot)$
  - Non-linear!
- Compute z and then pass it through the sigmoid function
- Treat it as a probability!



Sigmoids and Probabilities

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

$$= \frac{1}{1 + \exp(-(\mathbf{w}\cdot\mathbf{x} + b))}$$

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

$$= 1 - \frac{1}{1 + \exp(-(\mathbf{w}\cdot\mathbf{x} + b))}$$

$$= \frac{\exp(-(\mathbf{w}\cdot\mathbf{x} + b))}{1 + \exp(-(\mathbf{w}\cdot\mathbf{x} + b))}$$

$$= \frac{1}{1 + \exp(\mathbf{w}\cdot\mathbf{x} + b)}$$

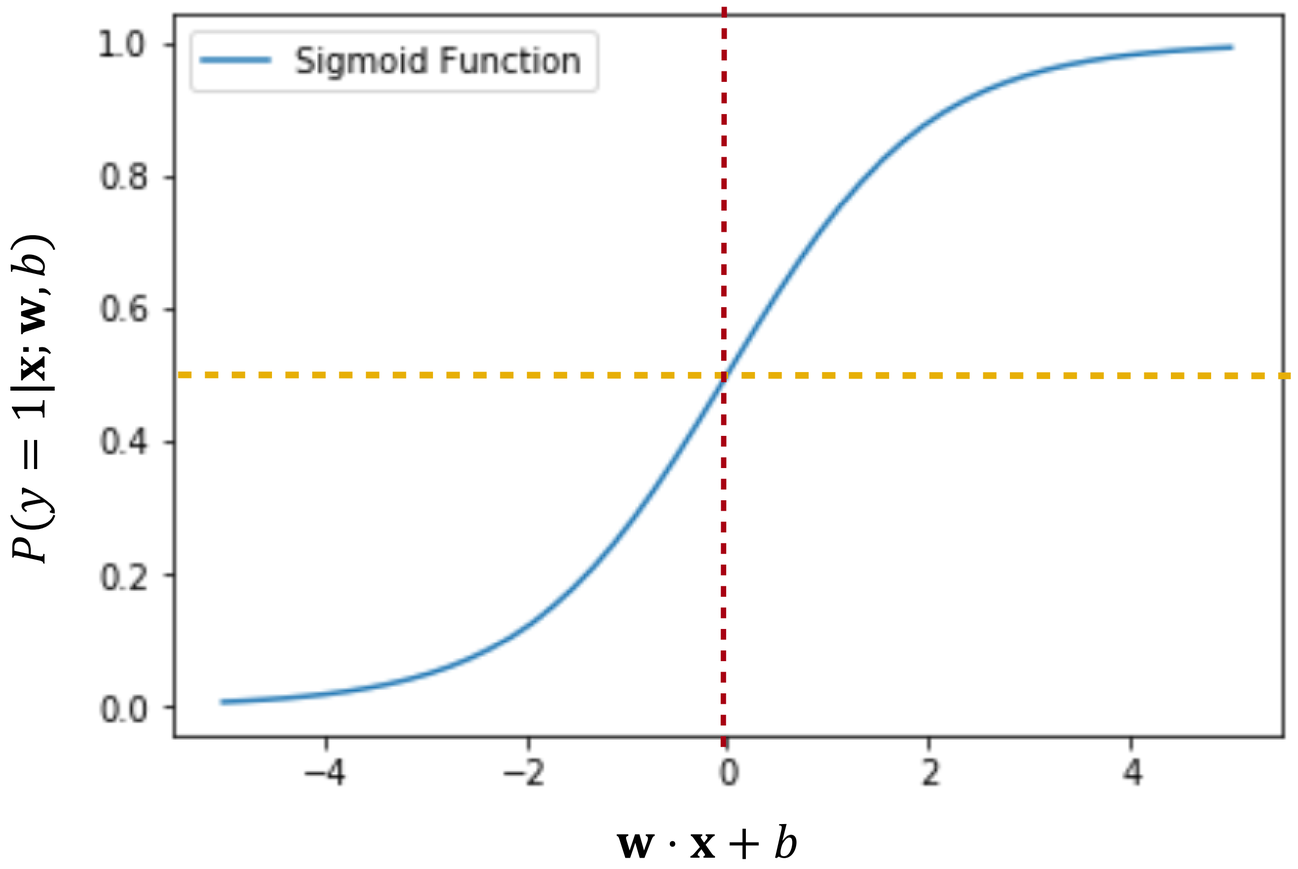
$$= \sigma(-(\mathbf{w}\cdot\mathbf{x} + b))$$

Classification Decision

$$\hat{y} = \begin{cases} 1 & \text{if } p(y = 1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Decision Boundary

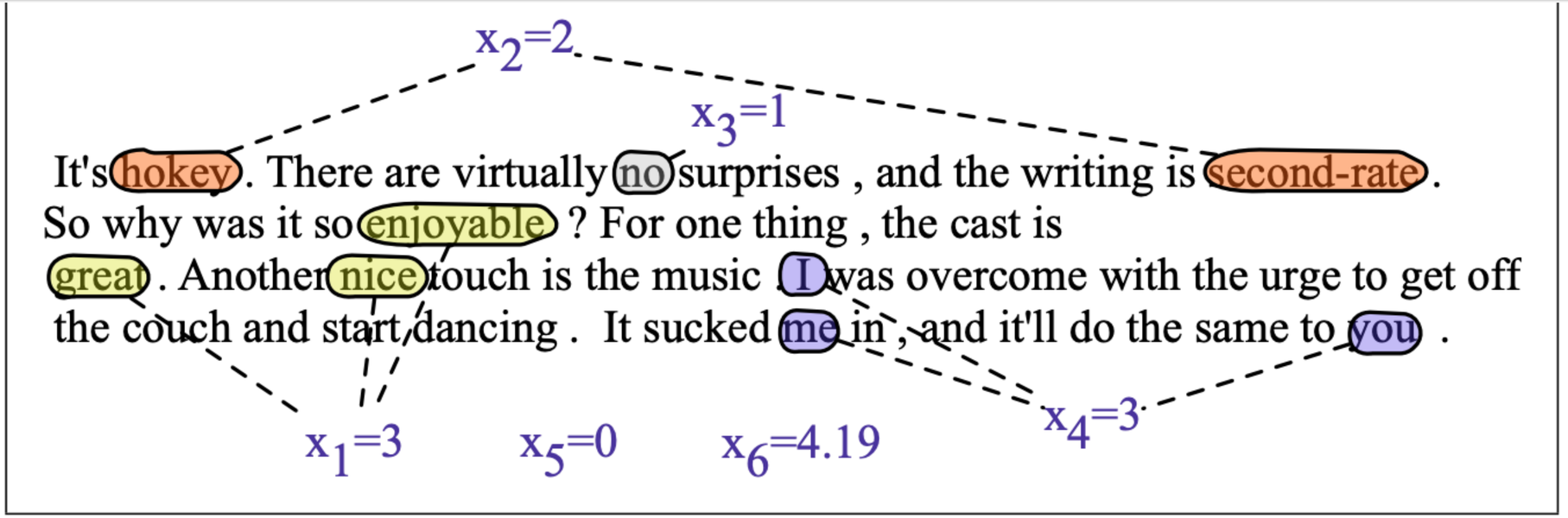
$$\hat{y} = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \end{cases}$$



Example: Sentiment Classification

It's hokey. There are virtually no surprises, and the writing is second-rate. So why was it so enjoyable? For one thing, the cast is great. Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you.

Is $y = 1$ or $y = 0$?



Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if } \text{“no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

Example: Classifying Sentiment

Var	Definition	Val
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if ``no'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if ``!'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

5.2

Suppose $w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$

$b = 0.1$

Example: Classifying Sentiment

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\
&= \sigma(.833) \\
&= 0.70
\end{aligned}
$$

$$
\begin{aligned}
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.30
\end{aligned}
$$

It's hokey. There are virtually no surprises, and the writing is second-rate. So why was it so enjoyable? For one thing, the cast is great. Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you.

Applying LR to other tasks

- Example: Period Disambiguation: Does a period correspond to the end of sentence?
  - "We saw many algorithms in class, e.g. word2vec."

$$x_1 = \begin{cases} 1 & \text{if } \text{“}Case(w_i) = \text{Lower”} \\ 0 & \text{otherwise} \end{cases}$$

$$x_2 = \begin{cases} 1 & \text{if } \text{“}w_i \in \text{AcronymDict”} \\ 0 & \text{otherwise} \end{cases}$$

$$x_3 = \begin{cases} 1 & \text{if } \text{“}w_i = \text{St. \& } Case(w_{i-1}) = \text{Cap”} \\ 0 & \text{otherwise} \end{cases}$$

Different tasks need different features; manually designed features must be task specific!

But where do the $\mathbf{w}$'s and the b's come from?

- Supervised Classification:
  - We know the correct label y (either 0 or 1) for each x
  - But what the system produces is an estimate, $\hat{y}$
- Set $\mathbf{w}$ and b to minimize the distance between our estimate $\hat{y}^{(i)}$ and the true $y^{(i)}$
  - We need a distance estimator: a loss function or a cost function
  - We need an optimization algorithm to update $\mathbf{w}$ and b to minimize the loss.

Loss function

Optimization Algorithm

III. Loss: Cross-Entropy

The distance between $\hat{y}$ and y

- We want to know how far is the classifier output:
  - $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$
  - (Not exactly, but it's reducible to $\hat{y}$)
- From the true (ground truth / gold standard) label:
  - $y \in \{0,1\}$
- This difference is called the loss or cost
  - $L(\hat{y}, y)$ = how much $\hat{y}$ differs from y
  - In other words, how much would you lose if you mispredicted
  - Or how much would it cost you to mispredict

Remember maximum likelihood?

- Here: conditional maximum likelihood estimation
- We choose the parameters $\mathbf{w}, b$ that maximize
  - the log probability
    - of the true y labels in the training data
    - given the observations x

$$\max \log p(y|x)$$

Suppose we flip the coin four times and see (H, H, H, T). What is p?

$p = 3/4 = 0.75$ maximizes the probability of data sequence (H,H,H,T)

maximum likelihood

Maximizing conditional likelihood

For a single observation

- Goal: maximize probability of the correct label $p(y|x)$
- Since there are only 2 discrete outcomes (0 or 1) we can express the probability $p(y|\mathbf{x})$ from our classifier (the thing we want to maximize) as

$$p(y|x;\mathbf{w},b) = \hat{y}^{y}(1-\hat{y})^{1-y}$$

$$\hat{y} = \sigma(\mathbf{w}\cdot\mathbf{x}+b)$$

Bernoulli Distribution

Only extremes!

	$\hat{y}=0$	$\hat{y}=1$
$y=0$	1	0
$y=1$	0	1

Maximizing conditional likelihood

- Goal: maximize probability of the correct label $p(y|\mathbf{x})$
- Maximize: $p(y|x) = \hat{y}^{y}(1-\hat{y})^{1-y}$
- Now take the log of both sides… to avoid underflow issues…

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

- Whatever values maximize $\log p(y|x)$ will also maximize $p(y|x)$

Minimizing negative log likelihood

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

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

- Now flip the sign for something to minimize (we minimize the loss / cost)
- Minimize: $L_{CE}(y,\hat{y}) = -\log p(y|x) = -[y\log\hat{y} + (1-y)\log(1-\hat{y})]$

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

Measures how well the training data matches the proposed model distribution and how good the model distribution is

Cross-Entropy Loss

Loss for sentiment classification

Case 1: Sentiment Analysis

- We want loss to be:
- smaller if the model estimate is close to correct
- bigger if model is confused

- Let's first suppose the true label of this is $y = 1$ (positive)

It's hokey. There are virtually no surprises , and the writing is second-rate. So why was it so enjoyable? For one thing, the cast is great. Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you.

Sentiment Example

True value is y=1. How well is our model doing?

$$\begin{aligned} p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\ &= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\ &= \sigma(.833) \\ &= 0.70 \end{aligned}$$

Pretty well! What's the loss?

$$\begin{aligned} L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1-y) \log (1 - \sigma(w \cdot x + b))] \\ &= -[\log \sigma(w \cdot x + b)] \\ &= -\log(.70) \\ &= .36 \end{aligned}$$

Sentiment Example: Contd

Now, suppose true value is $y = 0$. How well is our model doing?

$$\begin{aligned} p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\ &= 0.30 \end{aligned}$$

What's the loss?

$$\begin{aligned} L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1-y) \log (1 - \sigma(w \cdot x + b))] \\ &= -[\log (1 - \sigma(w \cdot x + b))] \\ &= -\log (.30) \\ &= 1.2 \end{aligned}$$

Sentiment Example: Summary

- The loss when the model is right (if true $y = 1$):

$$
\begin{aligned}
L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))] \\
&= -[\log \sigma(w \cdot x + b)] \\
&= -\log(.70) \\
&= .36
\end{aligned}
$$

Loss is bigger when the model is wrong!

..is lower than the loss when the model was wrong (if true $y = 0$)

$$
\begin{aligned}
L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))] \\
&= -[\log (1 - \sigma(w \cdot x + b))] \\
&= -\log (.30) \\
&= 1.2
\end{aligned}
$$

Loss in word2vec

Case 2: Word2Vec

c_{neg} } …aardvark…
…zebra…

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

c_1 c_2 w c_3 c_4

- 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

Case 2: Word2Vec

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

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

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

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

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

Regularization

Overfitting

- A model that perfectly match the training data has a problem.
- It will also overfit to the data, modeling noise
  - A random word that perfectly predicts y (it happens to only occur in one class) will get a very high weight.
  - Failing to generalize to a test set without this word.

Why?

A good model should be able to generalize

What happens when a feature only occurs with one class?

e.g. word "wow" for positive reviews

Overfitting: Features

This movie drew me in, and it'll do the same to you.

Useful or harmless features

x_1 = "this"
x_2 = "movie
x_3 = “hated”
x_4 = "drew me in"

I can't tell you how much I hated this movie. It sucked.

4gram features that just "memorize" training set and might cause problems

x_5 = "the same to you"
x_6 = "tell you how much”

Overfitting

- 4-gram model on tiny data will just memorize the data
  - 100% accuracy on the training set
- But it will be surprised by the novel 4-grams in the test data
  - Low accuracy on test set
- Models that are too powerful can overfit the data
  - Fitting the details of the training data so exactly that the model doesn't generalize well to the test set

How to avoid overfitting?

Regularization in logistic regression

Dropout in neural networks

Regularization

- A solution for overfitting: Add a regularization term $R(\theta)$ to the loss function
  - (for now written as maximizing logprob rather than minimizing loss)

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \sum_{i=1}^{n} \log P(y^{(i)}|\mathbf{x}^{(i)}) - \alpha R(\theta)$$

- Idea: choose an $R(\theta)$ that penalizes large weights
  - fitting the data well with lots of big weights not as good as
  - fitting the data a little less well, with small weights

L2 / Ridge Regularization

- The sum of the squares of the weights
- The name is because this is the (square of the) L2 norm $\| \theta \|_2^2$, = Euclidean distance of θ to the origin.

$$R(\boldsymbol{\theta}) = \| \boldsymbol{\theta} \|_2^2 = \sum_{j=1}^{d} \theta_j^2$$

L2 regularized objective function:

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \sum_{i=1}^{n} \log P(y^{(i)}|x^{(i)}) - \alpha \sum_{j=1}^{d} \theta_j^2$$

L1 / Lasso Regularization

- The sum of the (absolute value of the) weights
- Named after the L1 norm $\| \theta \|_1$ = sum of the absolute values of the weights = Manhattan distance

$$R(\theta) = \| \theta \|_1 = \sum_{j=1}^{d} |\theta_j|$$

L1 regularized objective function:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \log P(y^{(i)}|x^{(i)}) - \alpha \sum_{j=1}^{d} |\theta_j|$$

IV. Optimization: Stochastic Gradient Descent

Our goal: minimize the loss

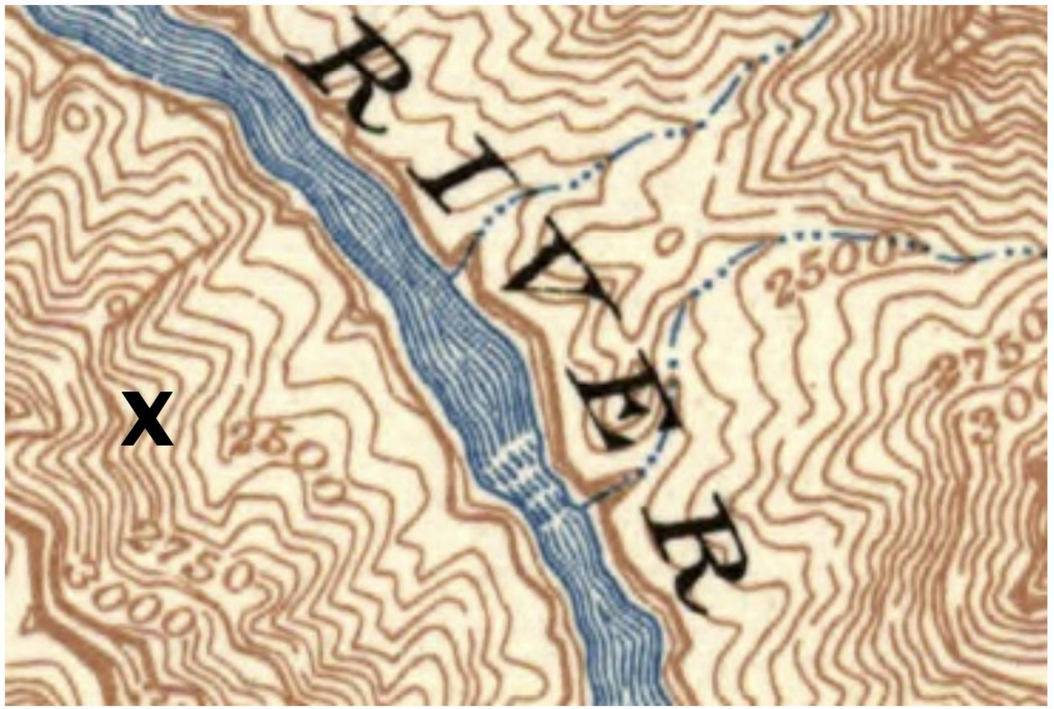
- Loss function is parameterized by weights: $\theta = [\mathbf{w}; b]$
- We will represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on θ more obvious
- We want the weights that minimize the loss, averaged over all examples:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{\text{CE}}(f(x^{(i)}; \theta), y^{(i)})$$

Intuition for gradient descent

How to get to the bottom of the river canyon?

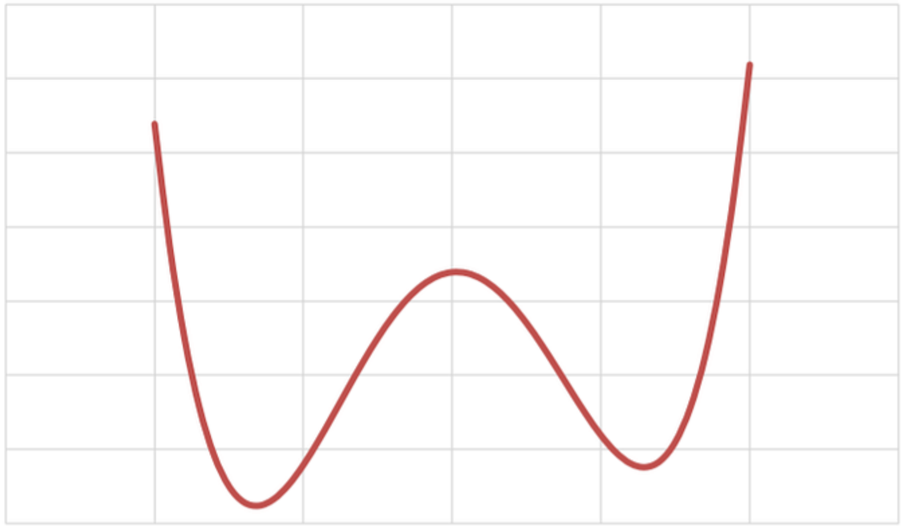
- Look around 360°
- Find the direction of steepest slope down
- Go that way



What if multiple equally good alternatives?

Logistic Regression: Loss

Convex function



Non-convex function

- Has only one option for steepest gradient
  - Or one minimum
- Gradient descent starting from any point is guaranteed to find the minimum

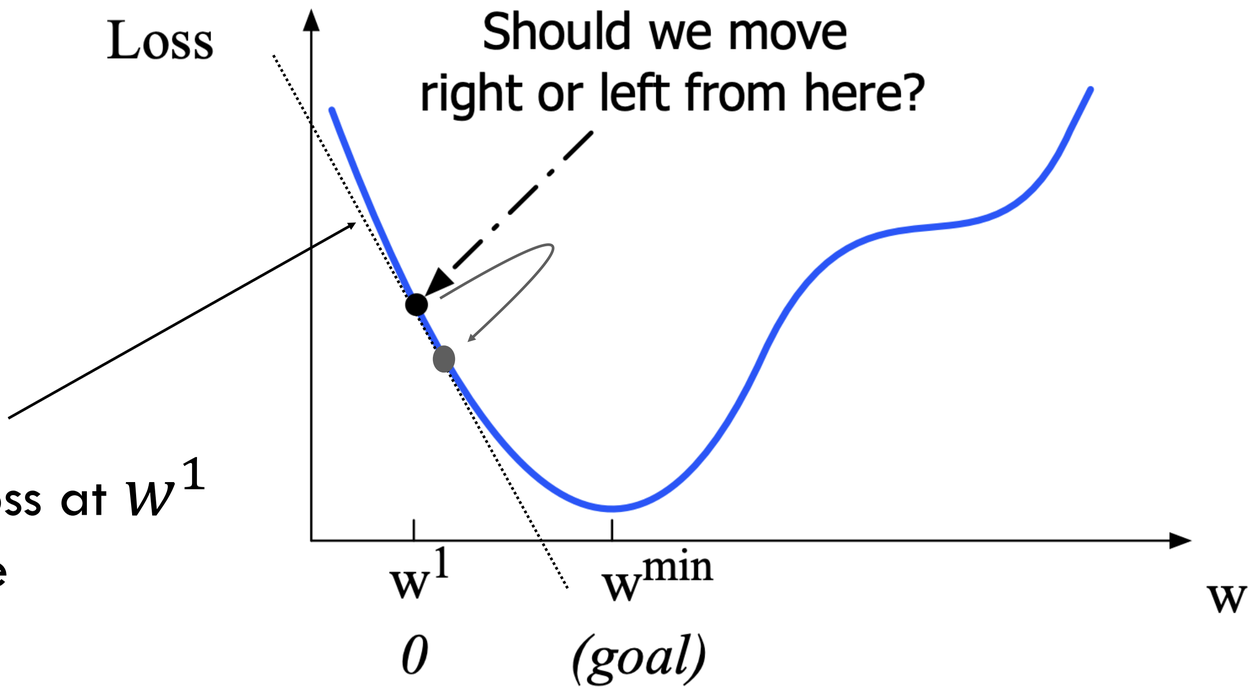
Neural Networks - multiple alternatives

Image Credit: Medium

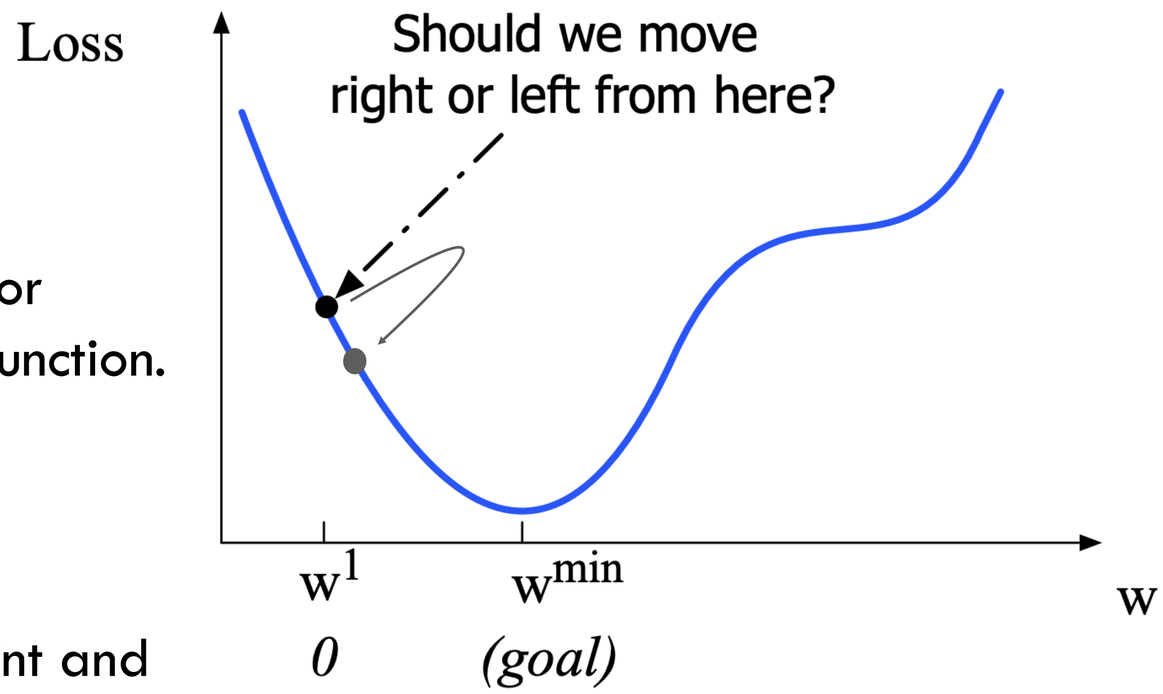
Consider: a single scalar w

Given current w, should we make it bigger or smaller?

Move w in the reverse direction from the slope of the function



Gradients



- The gradient of a function of many variables is a vector pointing in the direction of the greatest increase in a function.

Gradient Descent

- Find the gradient of the loss function at the current point and move in the opposite direction.

But by how much?

Gradient Updates

- Move the value of the gradient $\frac{\partial}{\partial w} L(f(x; w), y^*)$, weighted by a learning rate η
- Higher learning rate means move w faster

Too high: the learner will take big steps and overshoot

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

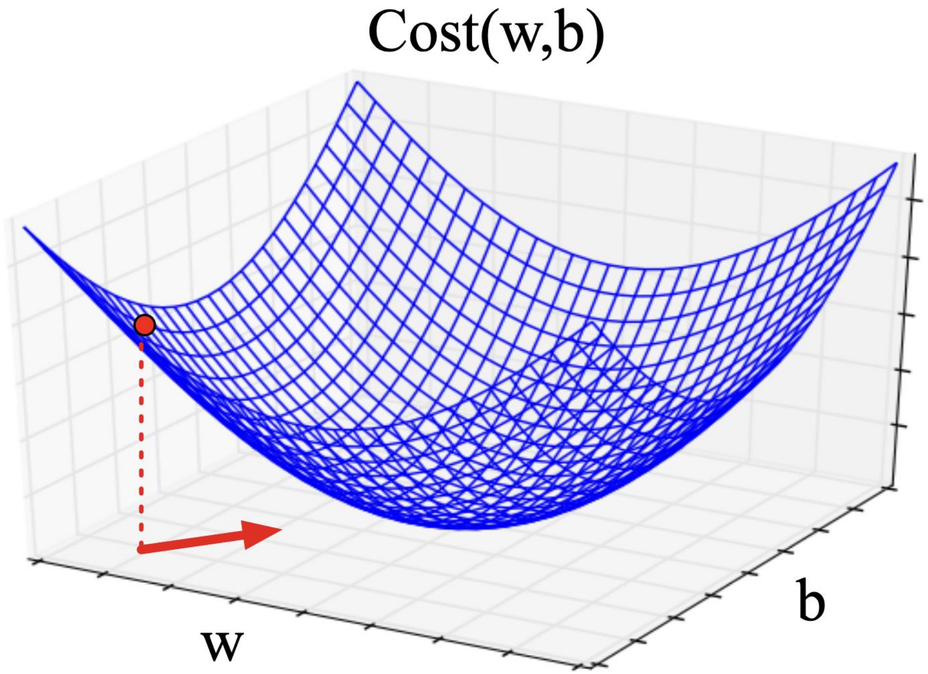
Too low: the learner will take too long

If parameter θ is a vector of d dimensions:

The gradient is just such a vector; it expresses the directional components of the sharpest slope along each of the d dimensions.

Under 2 dimensions

- Consider 2 dimensions, w and b:
- Visualizing the gradient vector at the red point
- It has two dimensions shown in the x-y plane



Real-life gradients

- Are much longer; lots and lots of weights!
- For each dimension θ_i the gradient component i tells us the slope with respect to that variable
  - "How much would a small change in θ_i influence the total loss function L?"
  - We express the slope as a partial derivative ∂ of the loss $\partial\theta_i$
  - The gradient is then defined as a vector of these partials

Real-life gradients

We will represent $\hat{y}$ as $f(x;\theta)$ to make the dependence on θ more obvious

$$\nabla_\theta L(f(x;\boldsymbol{\theta}),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\boldsymbol{\theta}),y) \\ \frac{\partial}{\partial w_2} L(f(x;\boldsymbol{\theta}),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\boldsymbol{\theta}),y) \end{bmatrix}$$

The final equation for updating θ at time step $t+1$ based on the gradient is thus

$$\theta_{t+1} = \theta_t - \eta \frac{\partial}{\partial \theta} L(f(x;\theta),y)$$

Gradients for Logistic Regression

Case 1: Sentiment Analysis

Recall: the cross-entropy loss for logistic regression

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

Derivatives have a closed form solution:

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

Gradients: word2vec

Case 2: Word2Vec

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

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

$$w_{t+1} = w_t - \eta[[\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{c}_{pos} + \sum_{j=1}^{K} [\sigma(\mathbf{c}_{neg_j} \cdot \mathbf{w})]\mathbf{c}_{neg_j}]$$

Pseudocode

- function STOCHASTIC GRADIENT DESCENT ($L(), f(), x, y$) returns θ
  - # where: L is the loss function
  - # f is a function parameterized by θ
  - # x is the set of training inputs $x^{(1)}, x^{(2)}, \ldots x^{(N)}$
  - # y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots y^{(N)}$
  - $\theta \leftarrow 0$ (or randomly initialized)
  - repeat till done
  - for each training tuple $(x^{(i)}, y^{(i)})$: (in random order)

Stochastic Gradient Descent

1. Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}^{(i)}$?
2. Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$ from the true output $y^{(i)}$?
3. $g \leftarrow \nabla L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss?
4. $\theta \leftarrow \theta - \eta g$ # Go the other way instead

  - return θ

Mini-Batching

function STOCHASTIC GRADIENT DESCENT $(L(), f(), x, y, m)$ returns θ

where: L is the loss function

f is a function parameterized by θ

x is the set of training inputs $x^{(1)}, x^{(2)}, \ldots x^{(N)}$

y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots y^{(N)}$ and m is the mini-batch size

$\theta \leftarrow 0$ (or randomly initialized)

repeat till done

for each randomly sampled minibatch of size m:

1. for each training tuple $(x^{(i)}, y^{(i)})$ in the minibatch: (in random order)
   i. Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}^{(i)}$?
   ii. Compute the loss $L_{mini} \leftarrow L_{mini} + L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$ from the true output $y^{(i)}$?
2. $g \leftarrow \frac{1}{m} \nabla L_{mini}(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss?
3. $\theta \leftarrow \theta - \eta g$ # Go the other way instead

return θ

Multinomial Logistic Regression

Multinomial Logistic Regression

- Often we need more than 2 classes
  - Positive / negative / neutral sentiment of a document
  - Parts of speech of a word (noun, verb, adjective, adverb, preposition, etc.)
  - Actionable classes for emergency SMSs
- If >2 classes we use multinomial logistic regression
  - = Softmax regression
  - = Multinomial logit
  - = (defunct names : Maximum entropy modeling or MaxEnt)
- So "logistic regression" will just mean binary (2 output classes)

Multinomial Logistic Regression

The probability of everything must still sum to 1

$$P(+|x) + P(-|x) + P(\sim |x) = 1$$

Need a generalization of the sigmoid called the softmax

- Takes a vector $\mathbf{z} = [z_1, z_2, \ldots, z_K]$ of K arbitrary values
- Outputs a probability distribution
  - each value in the range $[0,1]$
  - all the values summing to 1

Softmax

The Softmax Function

Turns a vector $\mathbf{z} = [z_1, z_2, \ldots, z_K]$ of K arbitrary values into probabilities

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

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

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

Softmax: Example

Turns a vector $\mathbf{z} = [z_1, z_2, \ldots, z_K]$ of K arbitrary values into probabilities

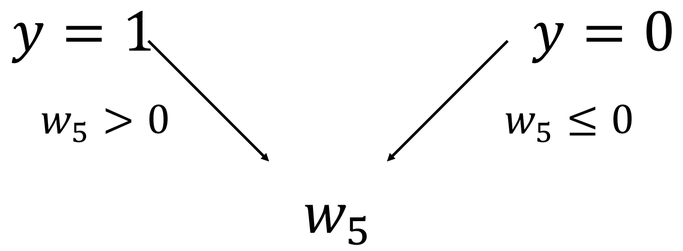
$$\mathbf{z} = [0.6,1.1,1.5,1.2,3.2,1.1]$$

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

$$softmax(\mathbf{z}) = [0.055,0.090,0.0067,0.10,0.74,0.010]$$

Binary versus Multinomial

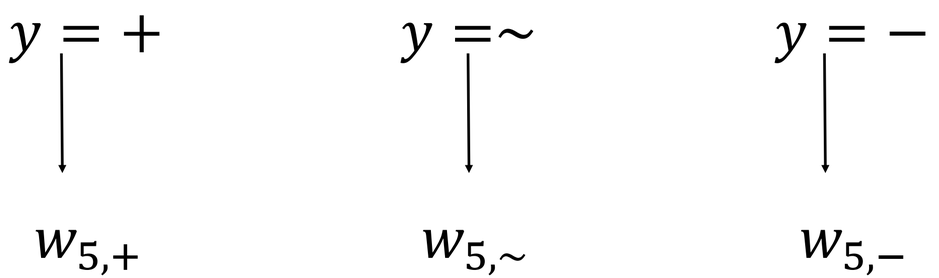
Binary Logistic Regression



$$x_5 = \begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases} \qquad w_5 = 3.0$$

Why do we not need a different weight for each class in binary logistic regression?

Multinomial Logistic Regression



separate weights for each class

Feature	Definition	$w_{5,+}$	$w_{5,-}$	$w_{5,0}$
$f_5(x)$	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	3.5	3.1	−5.3

Softmax in multinomial logistic regression

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

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

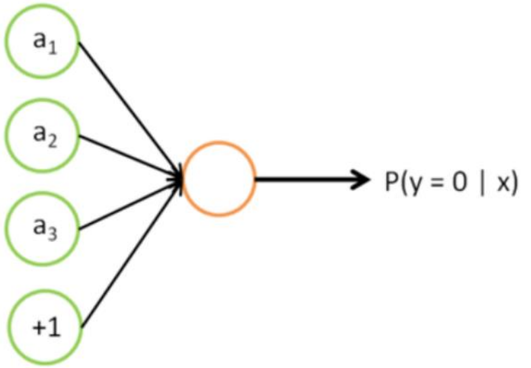
- Input is still the dot product between weight vector $\mathbf{w}_c$ and input vector $\mathbf{x}$, offset by b
- But separate weight vectors for each of the K classes, each of dimension d

Multinomial LR Loss:

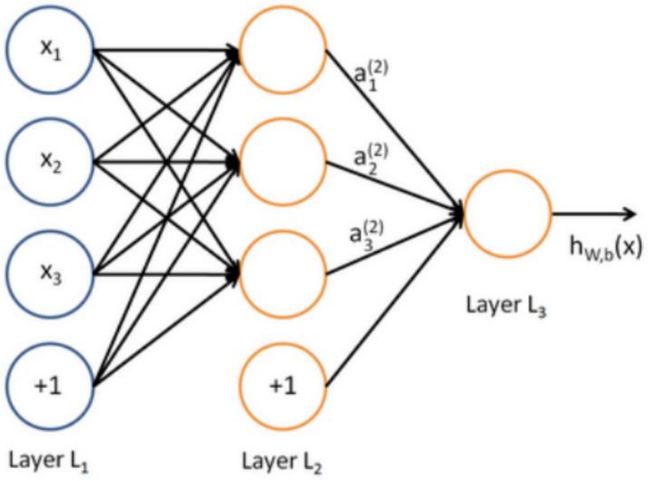
$$L_{CE} = -\log P(y = c|\mathbf{x}; \theta) = -(\mathbf{w}_c \cdot \mathbf{x} + b) + \log[\sum_{j=1}^{K} \exp(\mathbf{w}_j \cdot \mathbf{x} + b)]$$

Concluding Thoughts

- Logistic Regression
  - Running Example 1: Sentiment Classification
  - Running Example 2: word2vec
  - A very simple neural network
- Next Class: Let's add some hidden layers
  - Feed-Forward Neural Nets



Logistic Regression

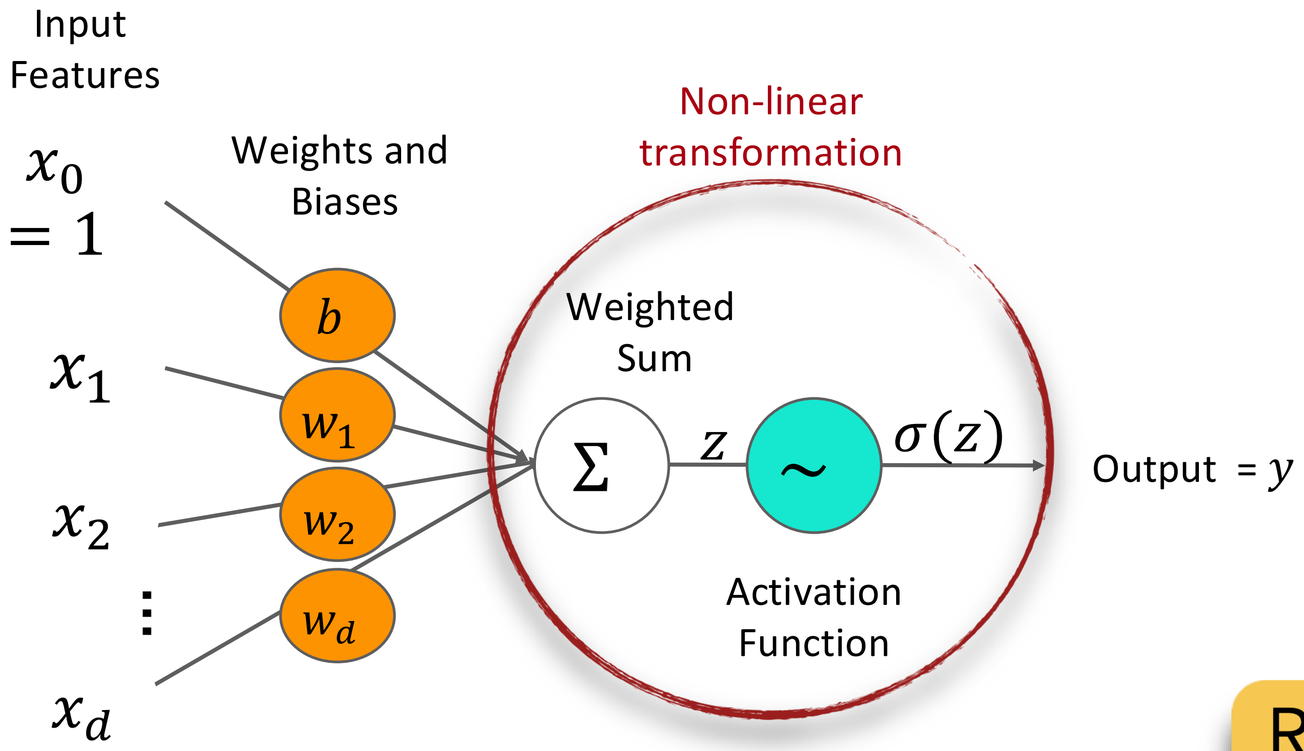


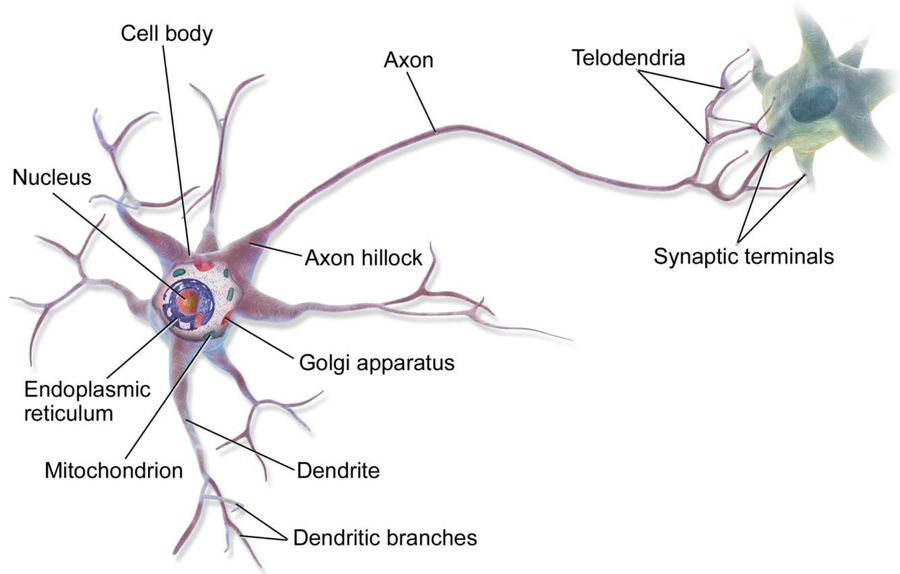
Neural Network

Feed-Forward Neural Networks

Neural Network Unit

Logistic Regression is a very simple neural network

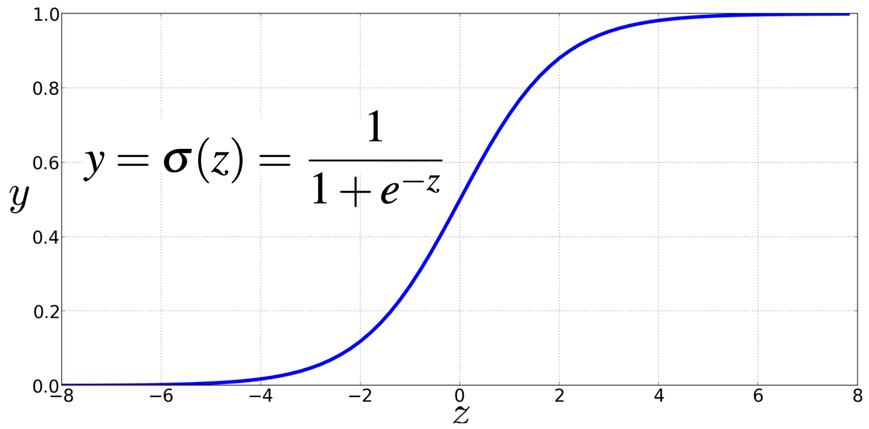


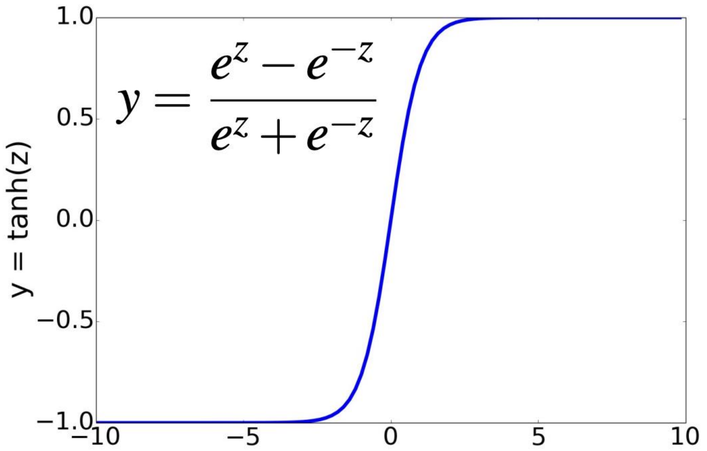


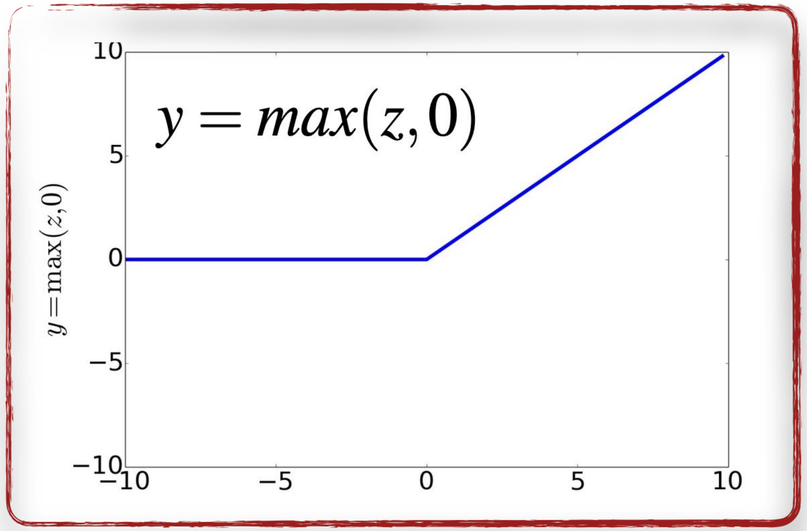
Resembles a neuron in the brain!

Non-Linear Activation Functions

Most common!







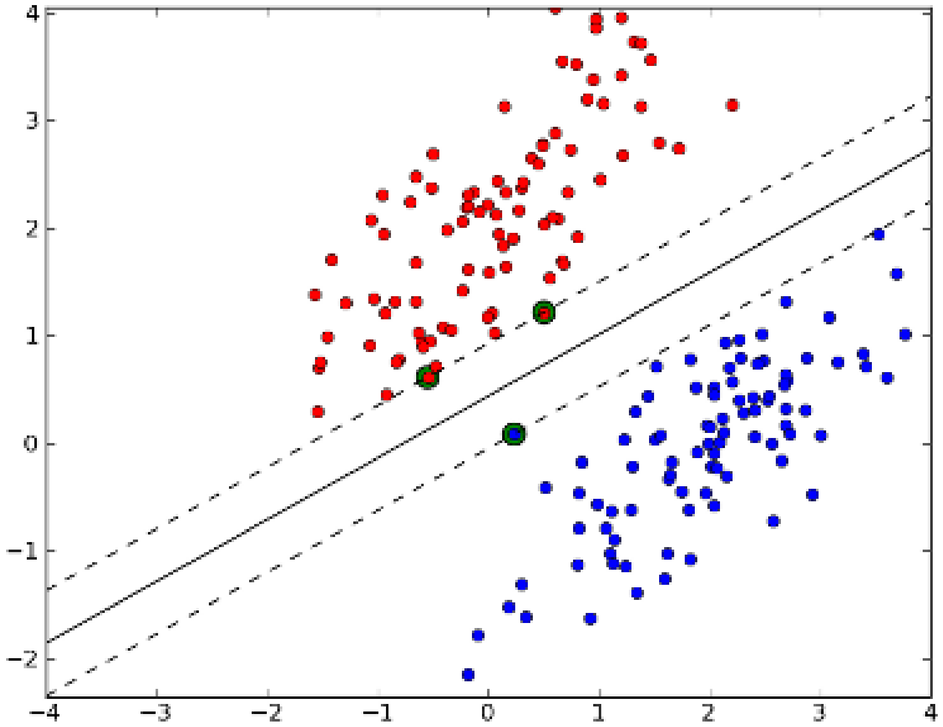
sigmoid

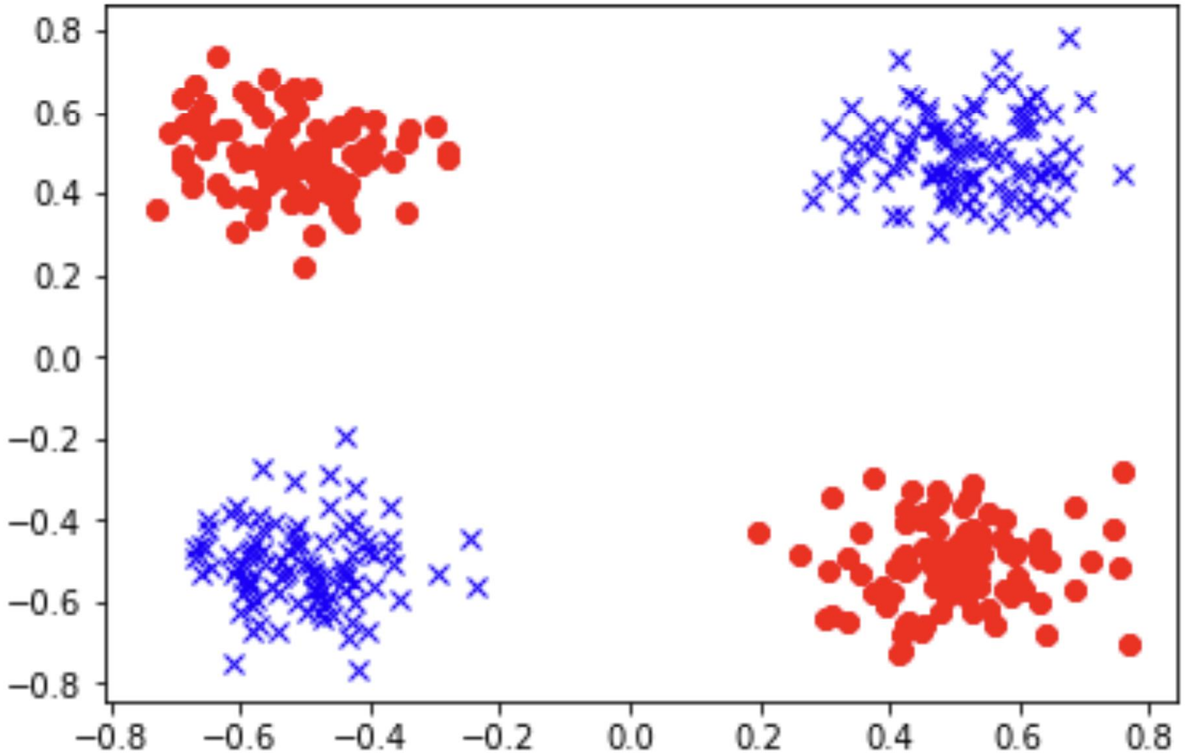
tanh

relu (Rectified Linear Unit)

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

Linear vs. Non-linear Functions

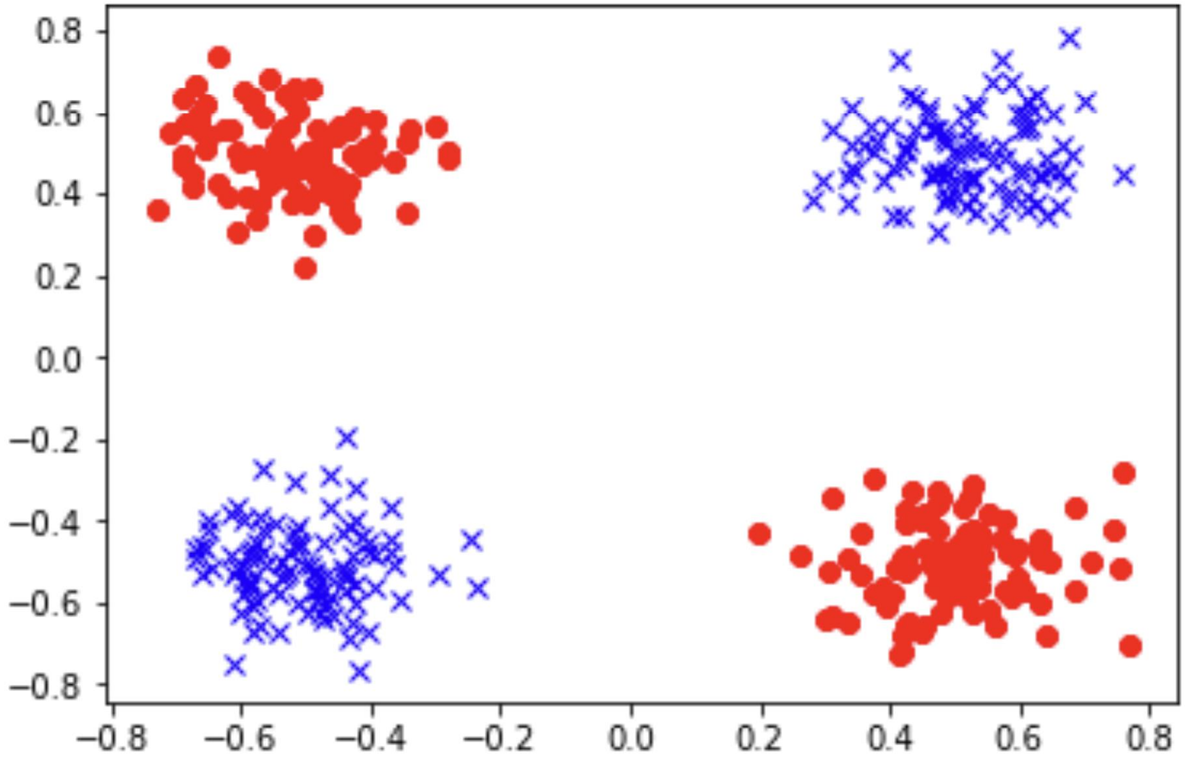




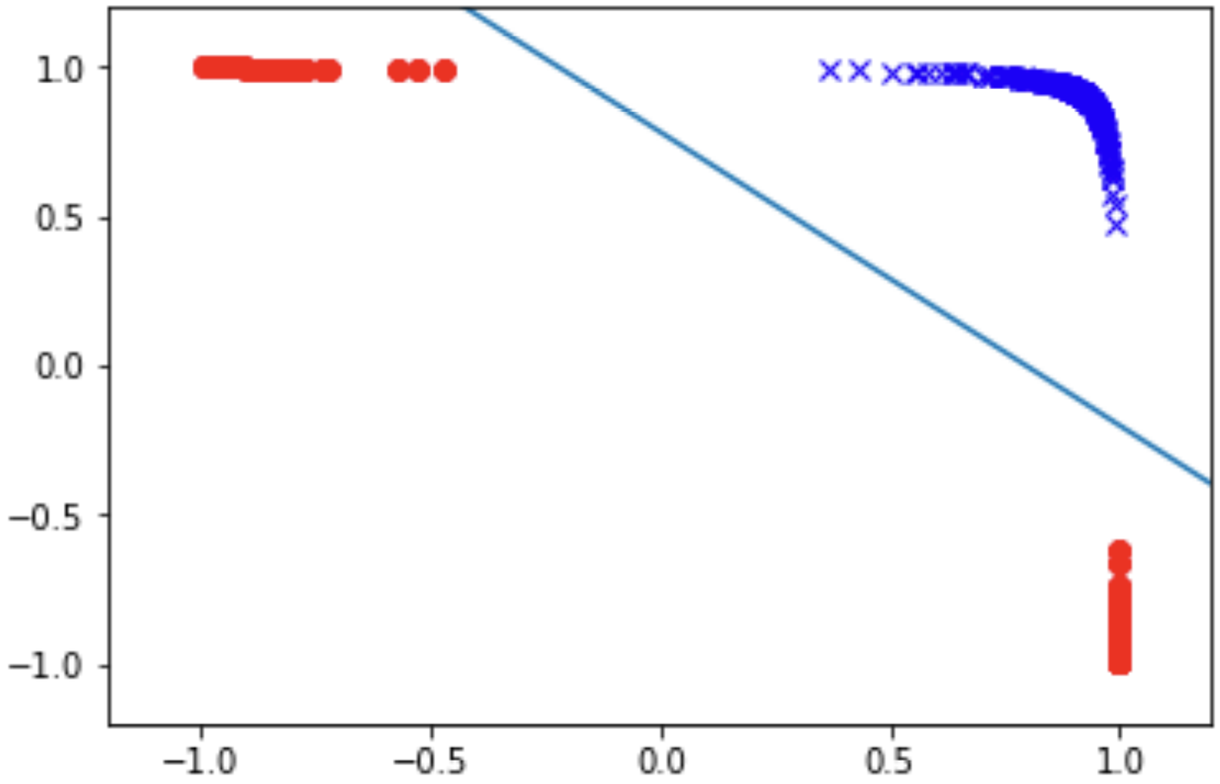
Linearly inseparable

Power of non-linearity

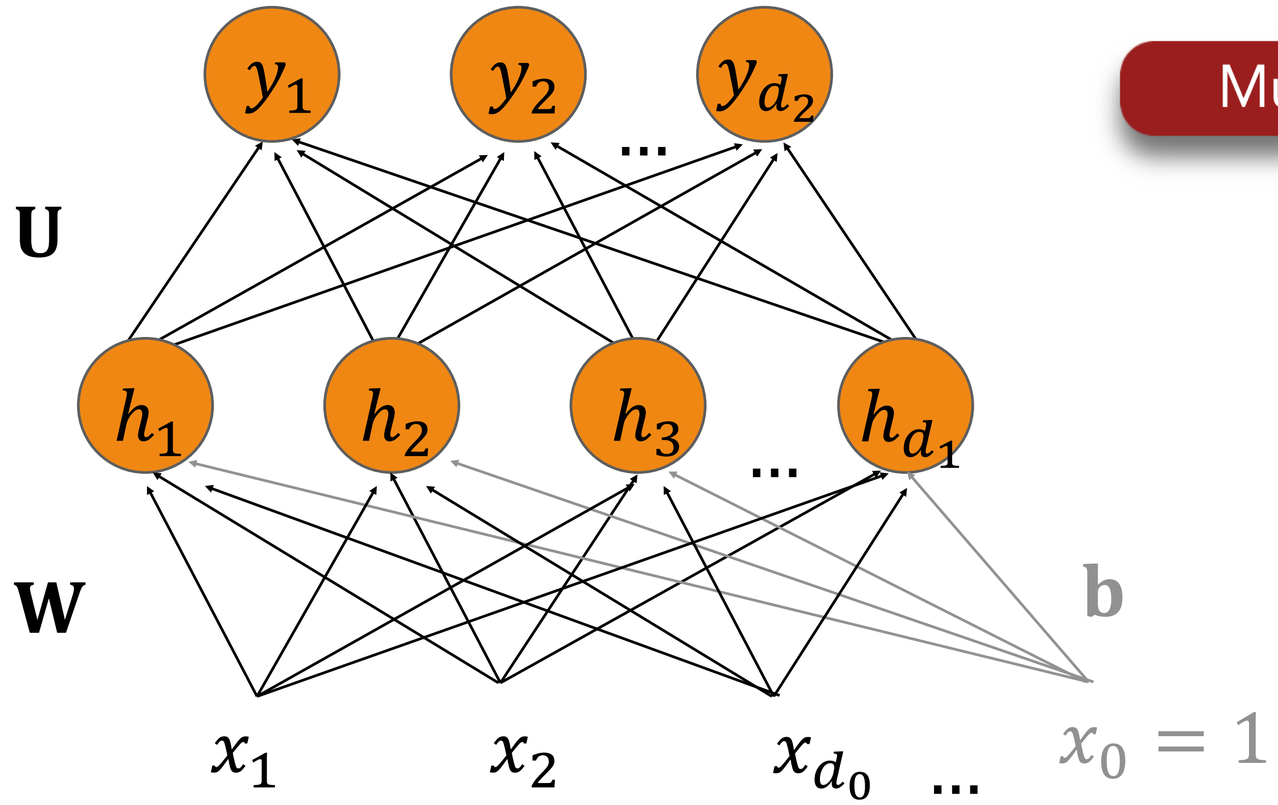
$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



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



Feedforward Neural Nets

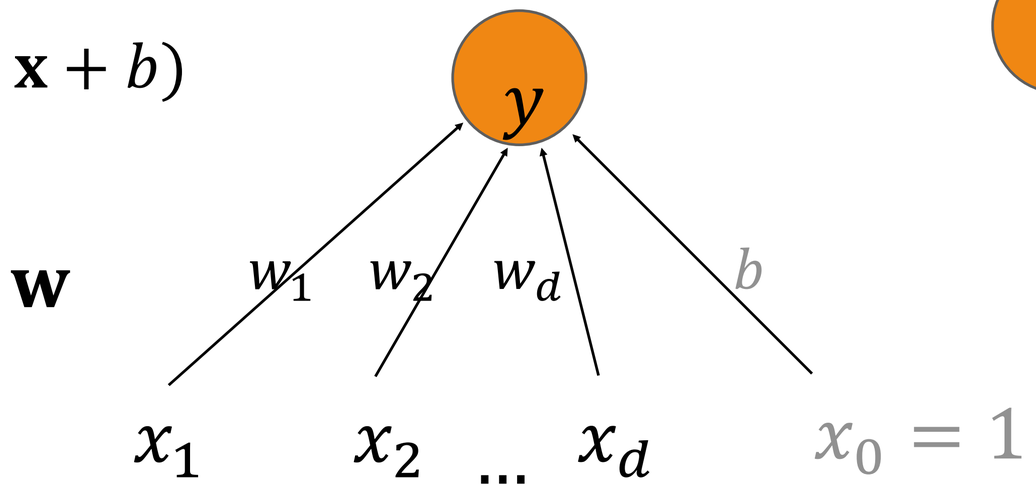


Multilayer Perceptron

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

Binary Logistic Regression

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



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

1-layer Network

we don't count the input layer in counting layers!

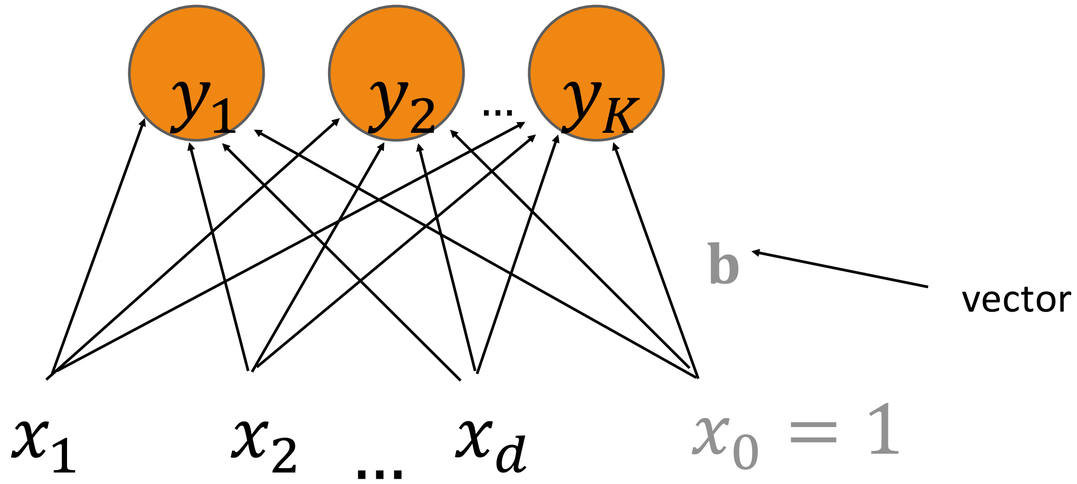
Multinomial Logistic Regression

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

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

matrix → $\mathbf{W}$

Input layer: vector $\mathbf{x}$



1-layer Network

Fully connected single layer network

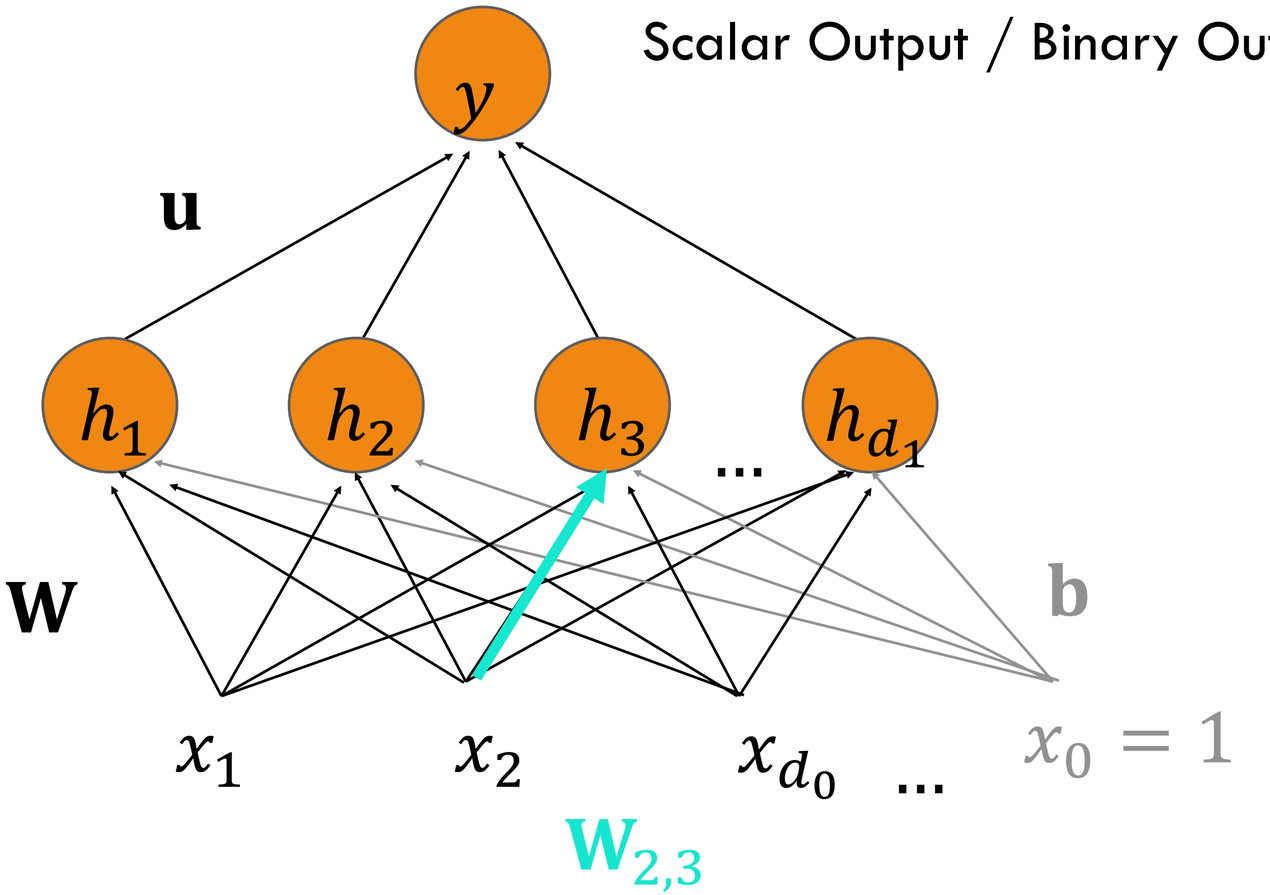
Two-layer Feedforward Network

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

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

Usually ReLU or tanh

Input layer: vector $\mathbf{x}$



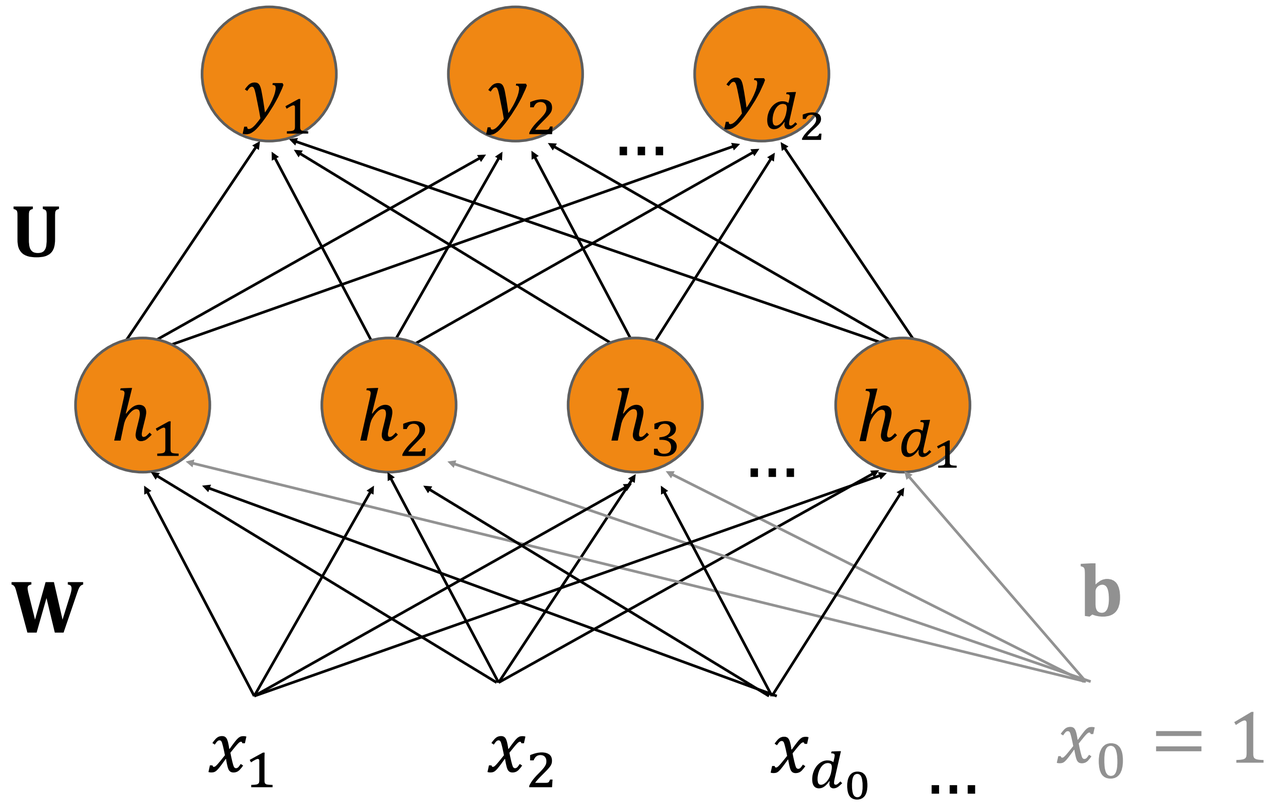
Two-layer Feedforward Network with Softmax Output

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

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

Usually ReLU or tanh

Input layer: vector $\mathbf{x}$



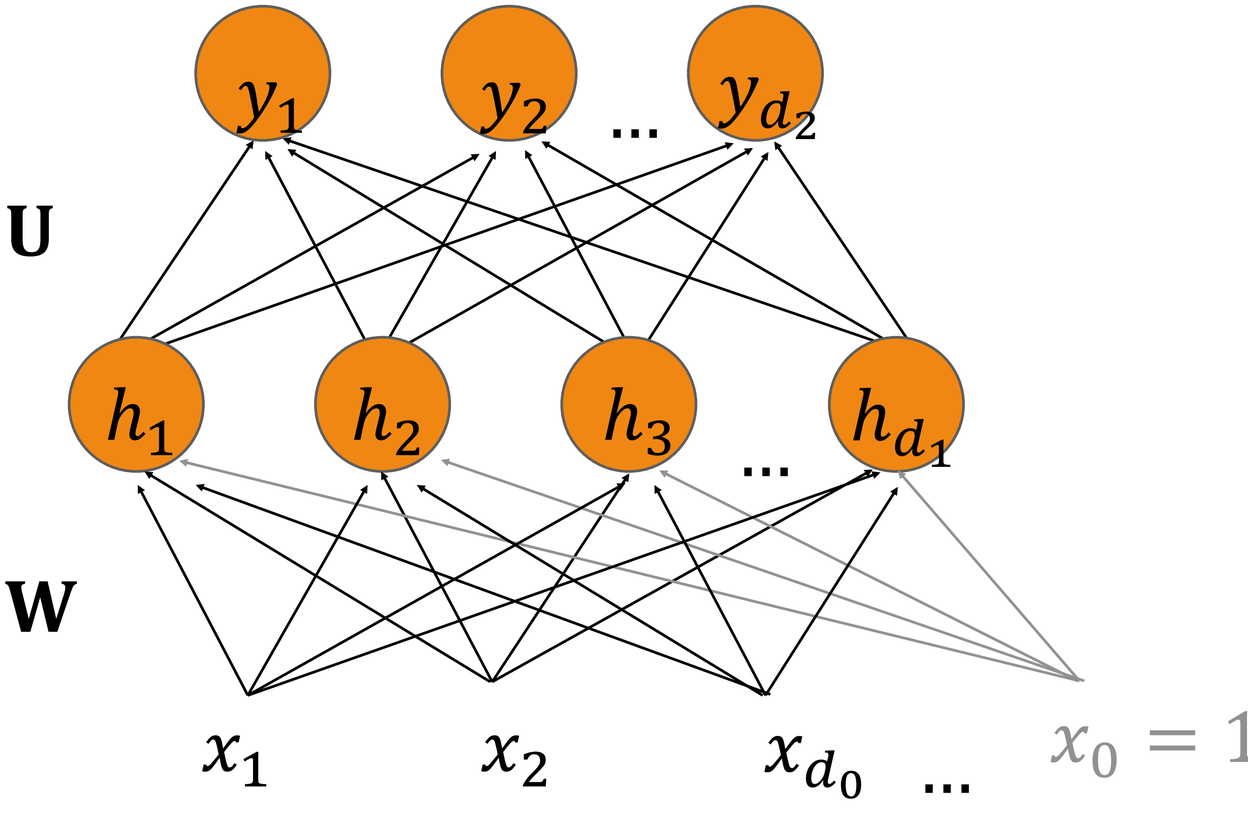
Two-layer FFNN: Notation

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

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

Usually ReLU or tanh

Input layer: vector $\mathbf{x}$



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

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
  - Remember the overfitting problem in n-gram LMs?
  - In general, count-based methods can never do as well as optimization-based ones

Why?

- 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!

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….

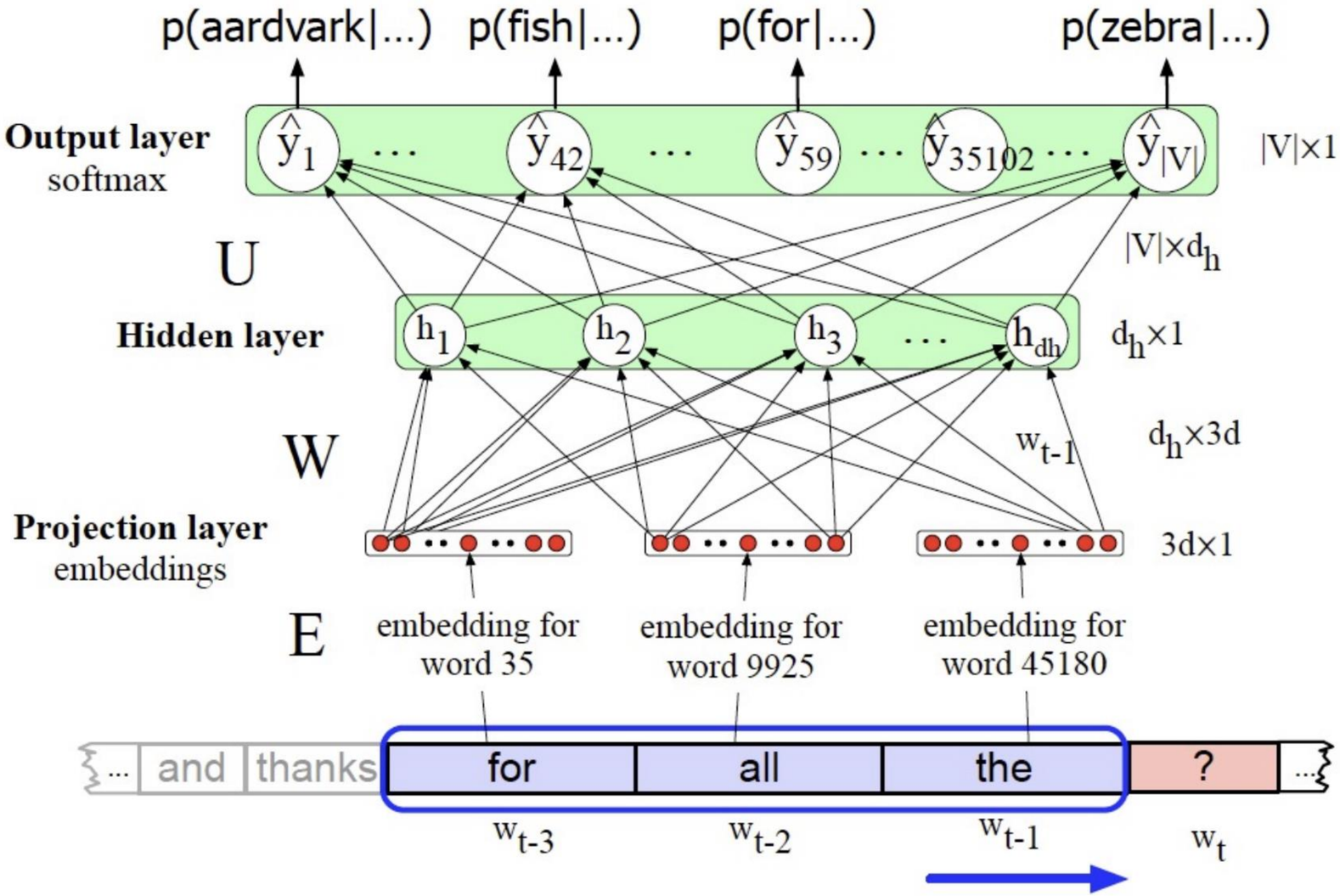
Solution: Sliding windows (of fixed length)

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

Where does this come from?

Feedforward Neural LM

- Sliding window of size 3
- Every feature in the embedding vector connected to every single hidden unit



Training FFNNs and Backprop

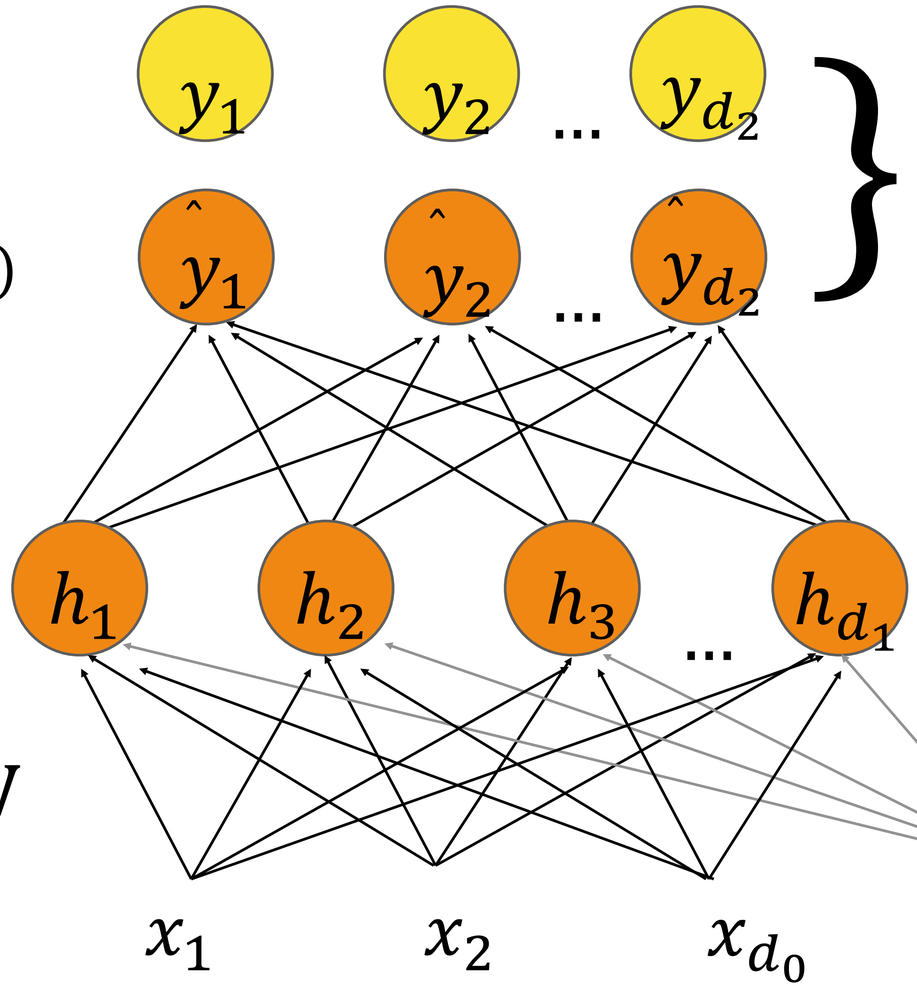
Intuition: Training a 2-layer Network

Training instance $\mathbf{y}$

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

Forward Pass

Training instance $\mathbf{x}$



Loss function $L(\hat{\mathbf{y}}, \mathbf{y})$

Backward Pass

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!

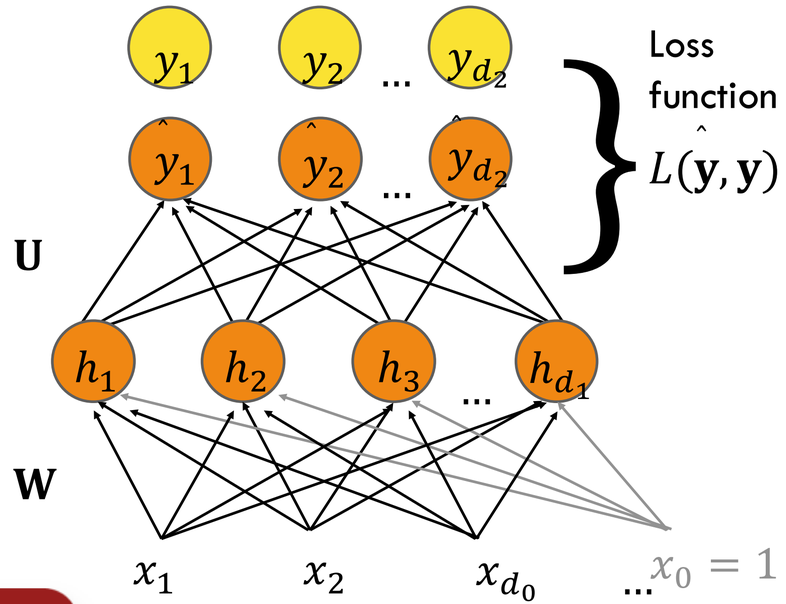
computation graphs

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

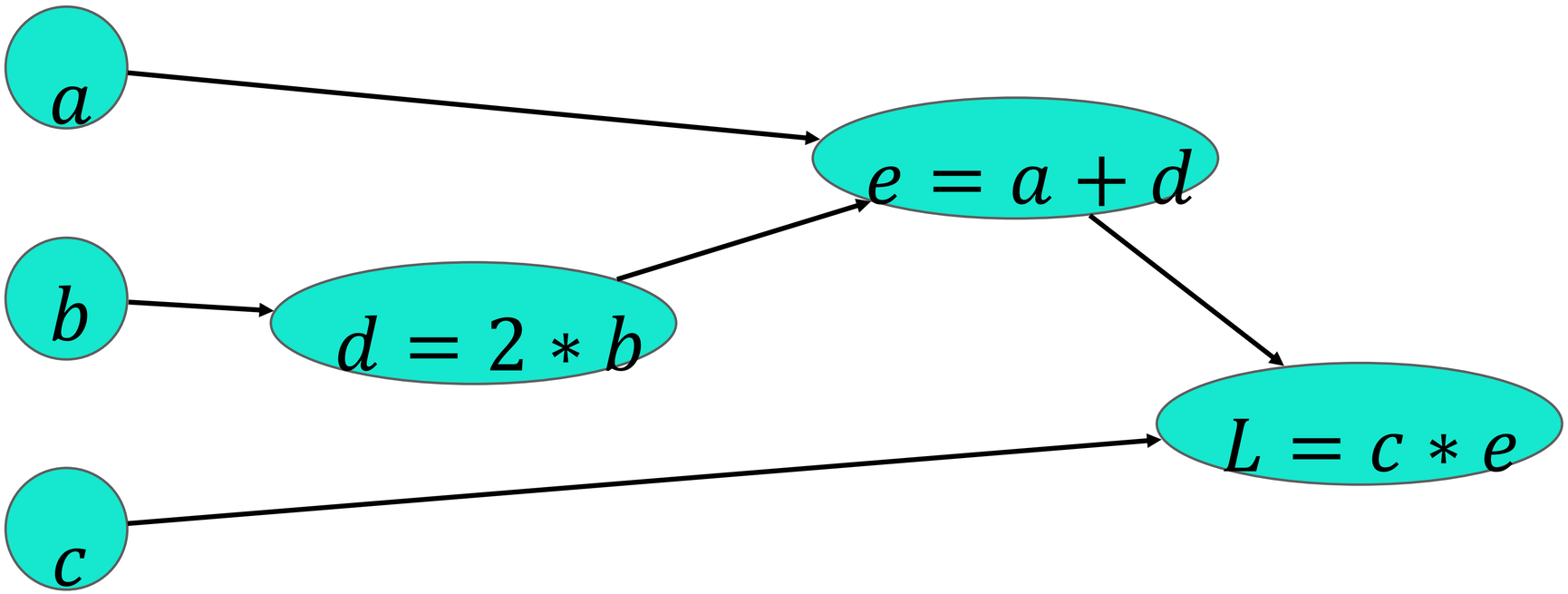
Rumelhart, Hinton, Williams, 1986

Example: Computation Graph

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

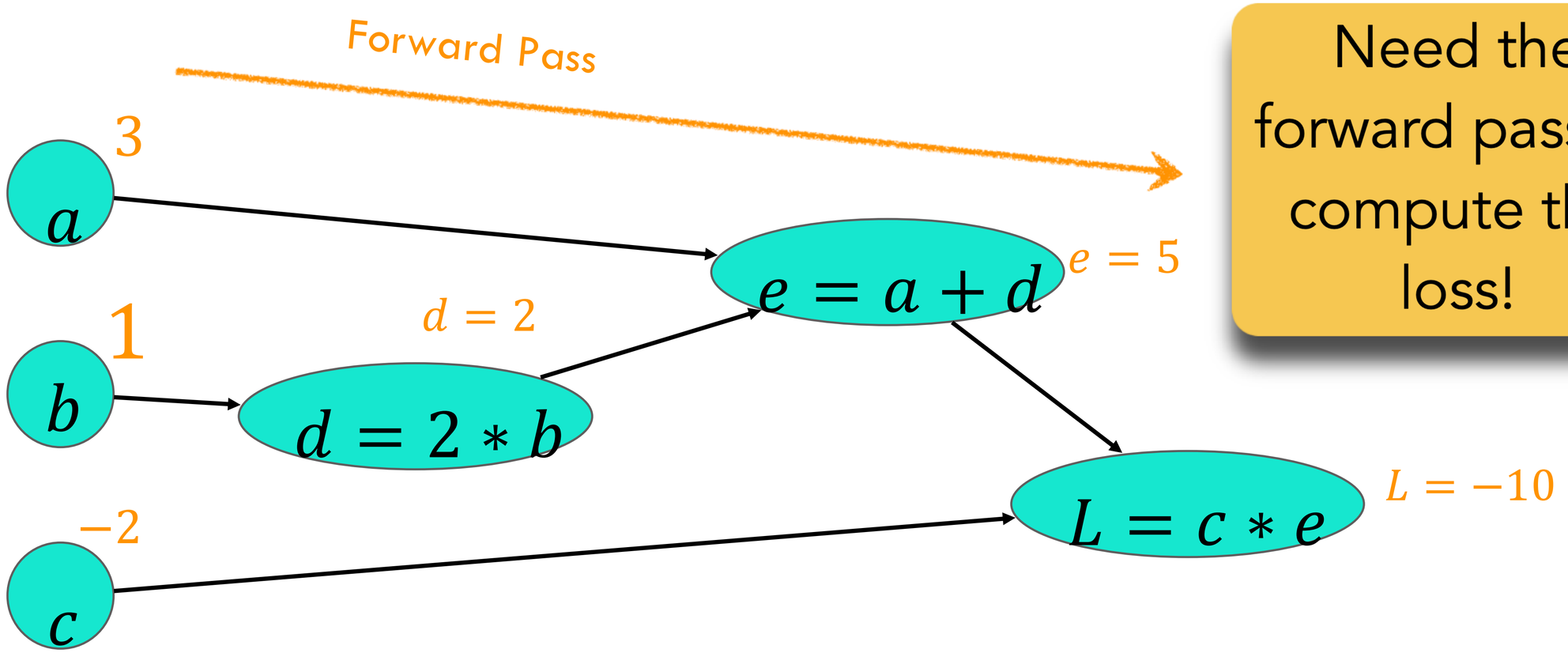


Example: Forward Pass

$d = 2 * b$

$e = a + d$

$L = c * e$



Need the forward pass to compute the loss!

But how to compute parameter updates?

Example: Backward Pass Intuition

$$d = 2 * b$$

$$e = a + d$$

$$L = c * e$$

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

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

$$\frac{\partial L}{\partial b} = ?$$

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

} Input Layer Gradients

Hidden Layer Gradients }

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

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

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

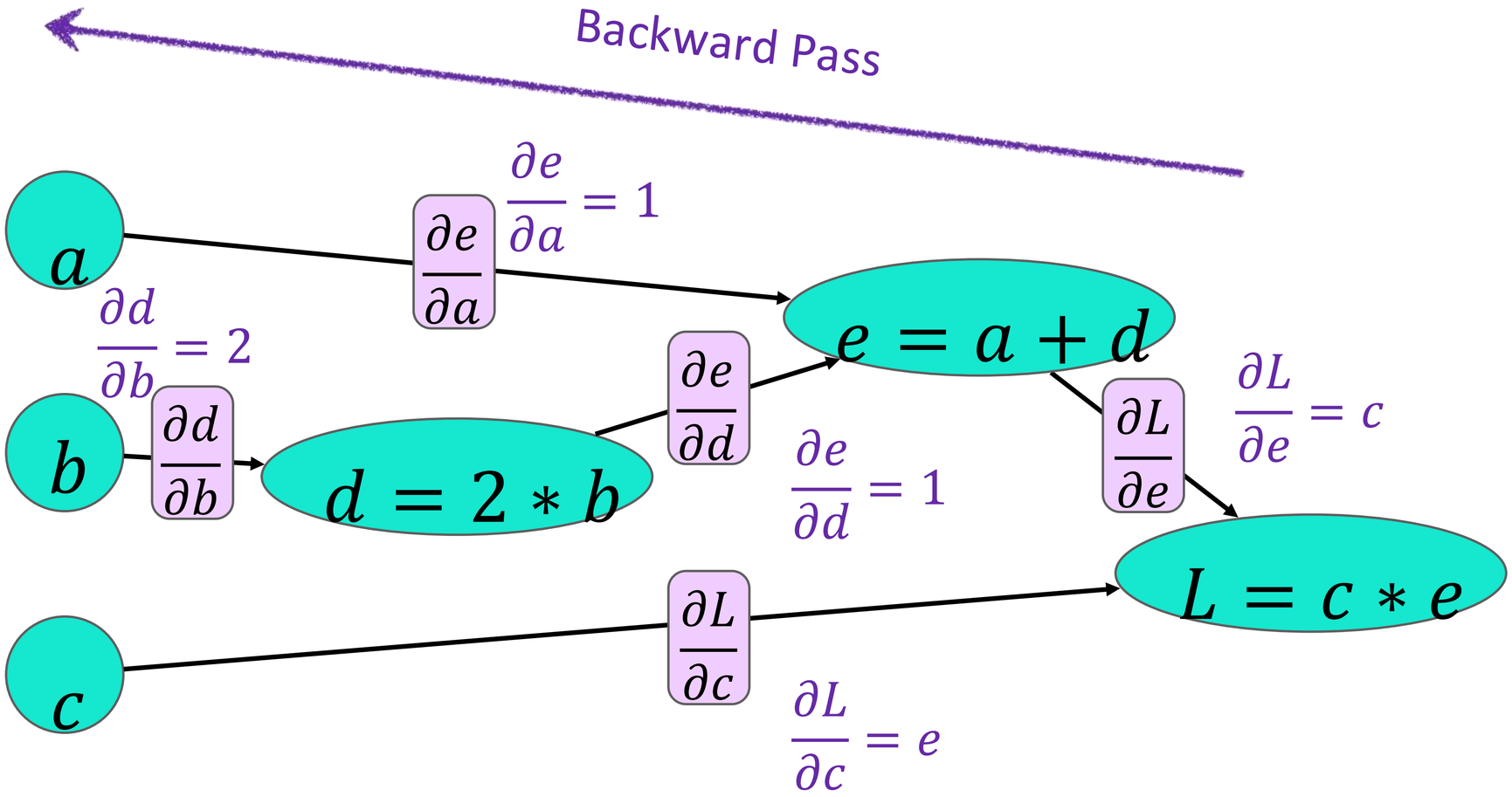
But we need the gradients of the loss with respect to parameters…

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

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

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

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



Example

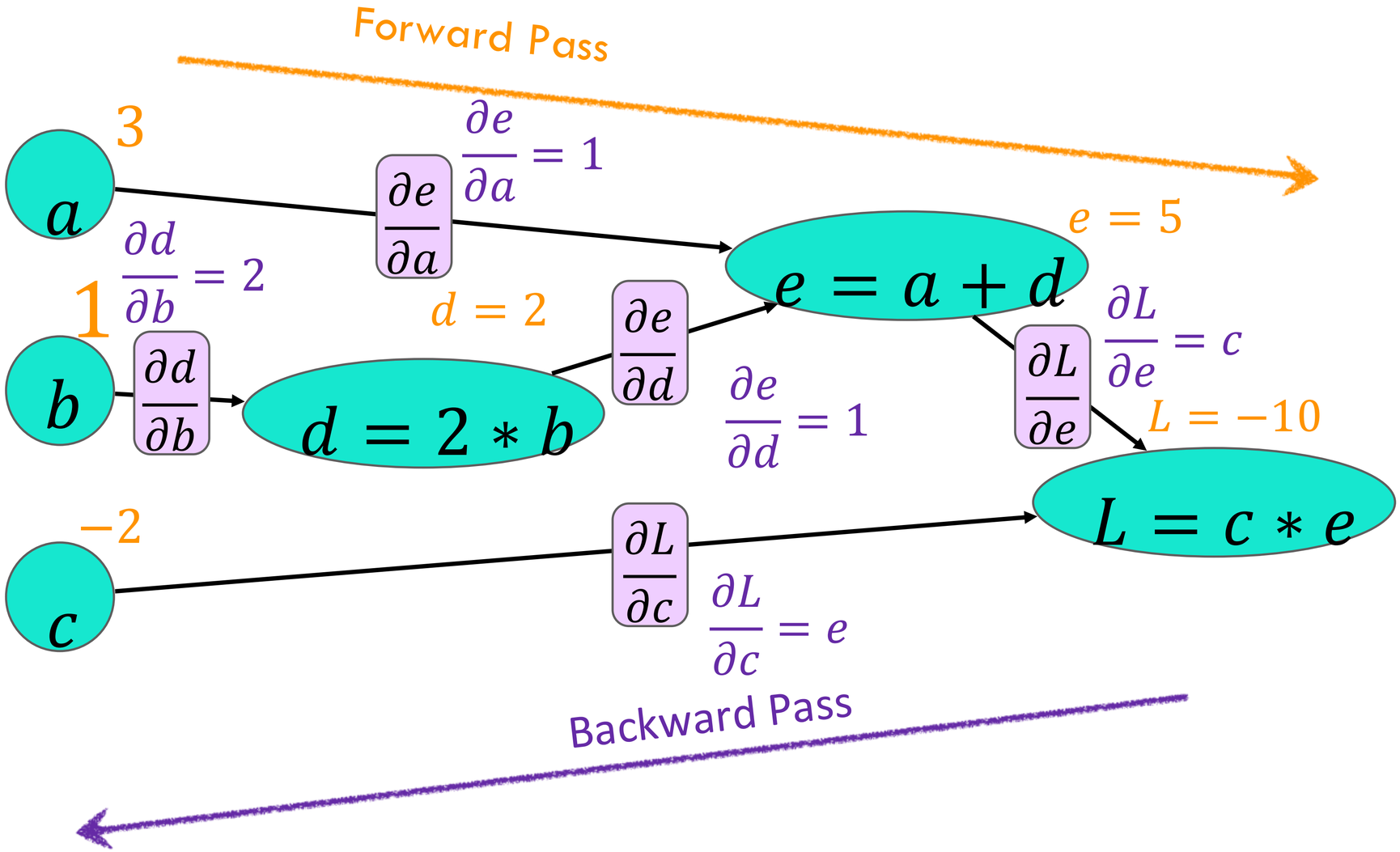
$$\frac{\partial L}{\partial e} = c = -2$$

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

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

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

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



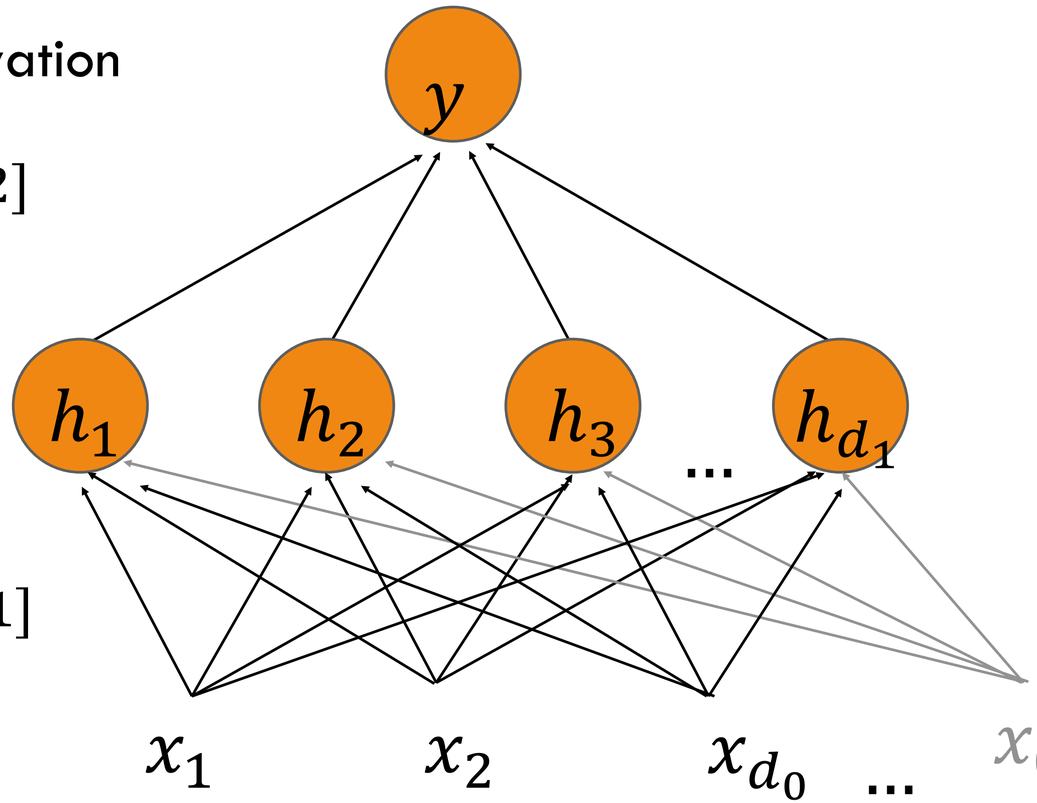
Backward Differentiation on a 2-layer MLP

Softmax Activation

$\mathbf{w}^{[2]}$

ReLU Activation

$\mathbf{W}^{[1]}$



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

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

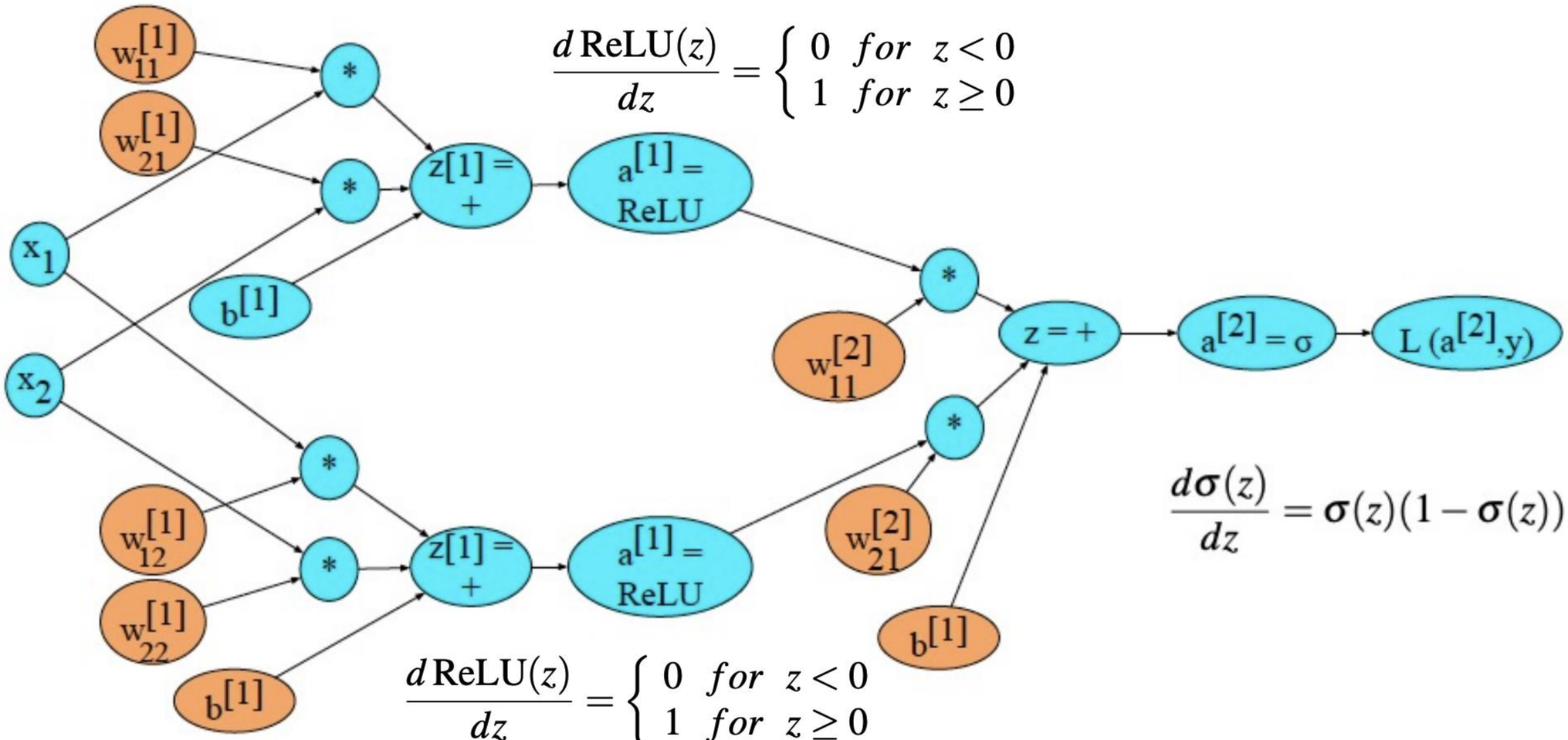
$$\mathbf{h}^{[1]} = ReLU(\mathbf{z}^{[1]})$$ Element-wise

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

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

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

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 $z^{[2]}$)

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

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

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

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

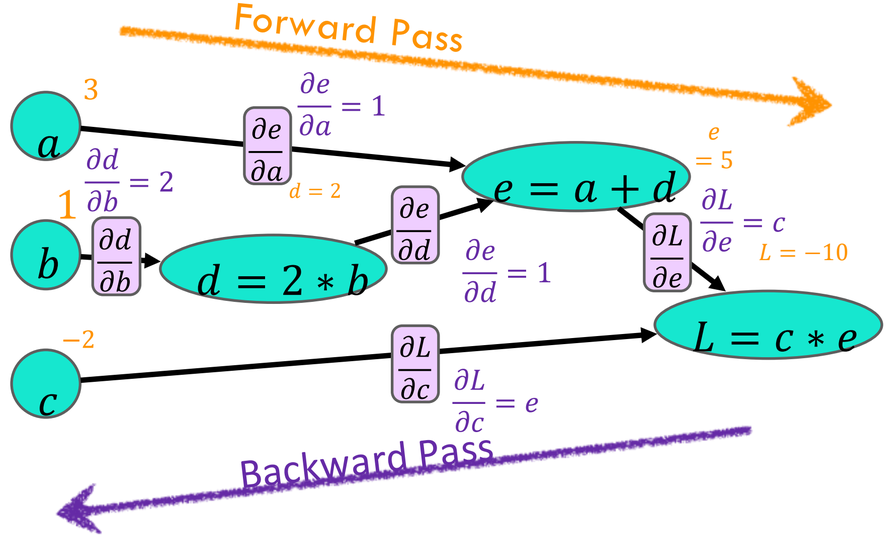
$$\frac{\partial L}{\partial a} = -\left(\left(y\frac{\partial \log(a)}{\partial a}\right) + (1-\mathrm{y})\frac{\partial \log(1-a)}{\partial a}\right)$$

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

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

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()`

Recurrent Neural Nets

Recurrent Neural Networks

- Recurrent Neural Networks processes sequences one element at a time:
  - Contains one hidden layer $\mathbf{h}_t$ per time step! Serves as a memory of the entire history…
  - Output of each neural unit at time t based both on
    - the current input at t and
    - the hidden layer from time $t-1$
- As the name implies, RNNs have a recursive formulation
    - dependent on its own earlier outputs as an input!
- RNNs thus don't have
  - the limited context problem that n-gram models have, or
  - the fixed context that feedforward language models have,
  - since the hidden state can in principle represent information about all of the preceding words all the way back to the beginning of the sequence

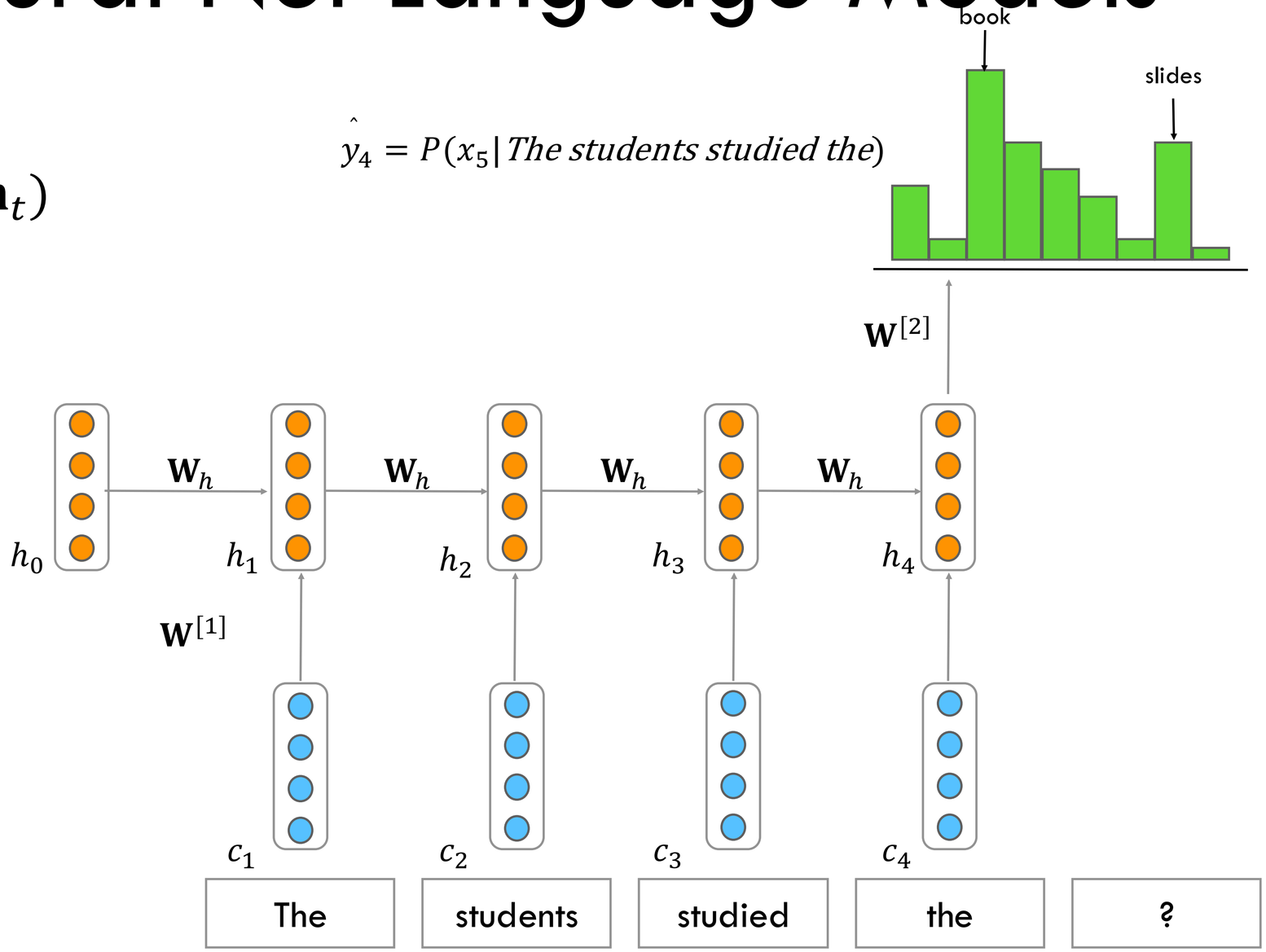
Recurrent Neural Net Language Models

Output layer: $\hat{\mathbf{y}}_t = softmax(\mathbf{W}^{[2]}\mathbf{h}_t)$

Hidden layer: $\mathbf{h}_t$
$= g(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}^{[1]}\mathbf{c}_t)$

Initial hidden state: $\mathbf{h}_0$

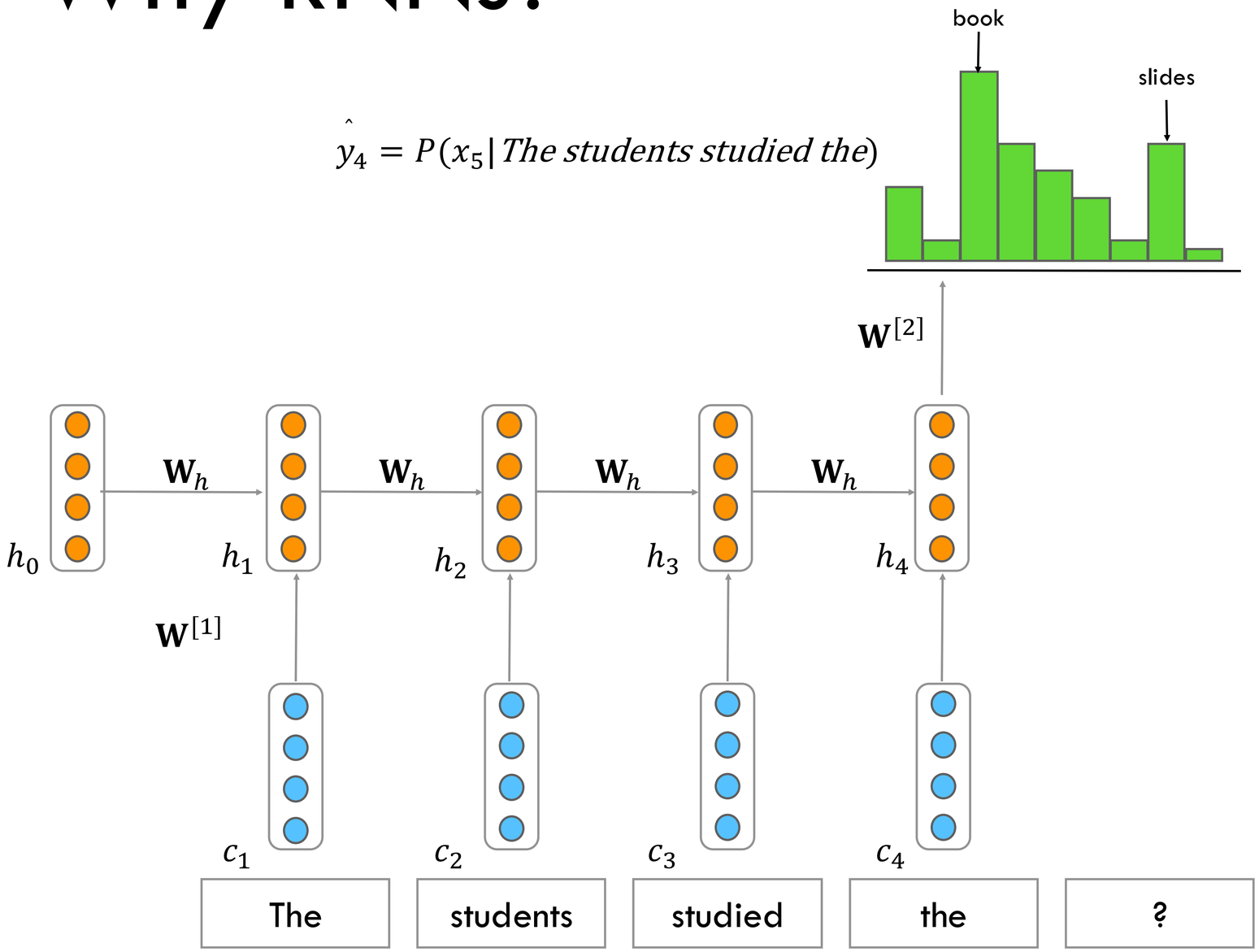
Word Embeddings, $\mathbf{c}_i$



Why RNNs?

RNN Advantages:

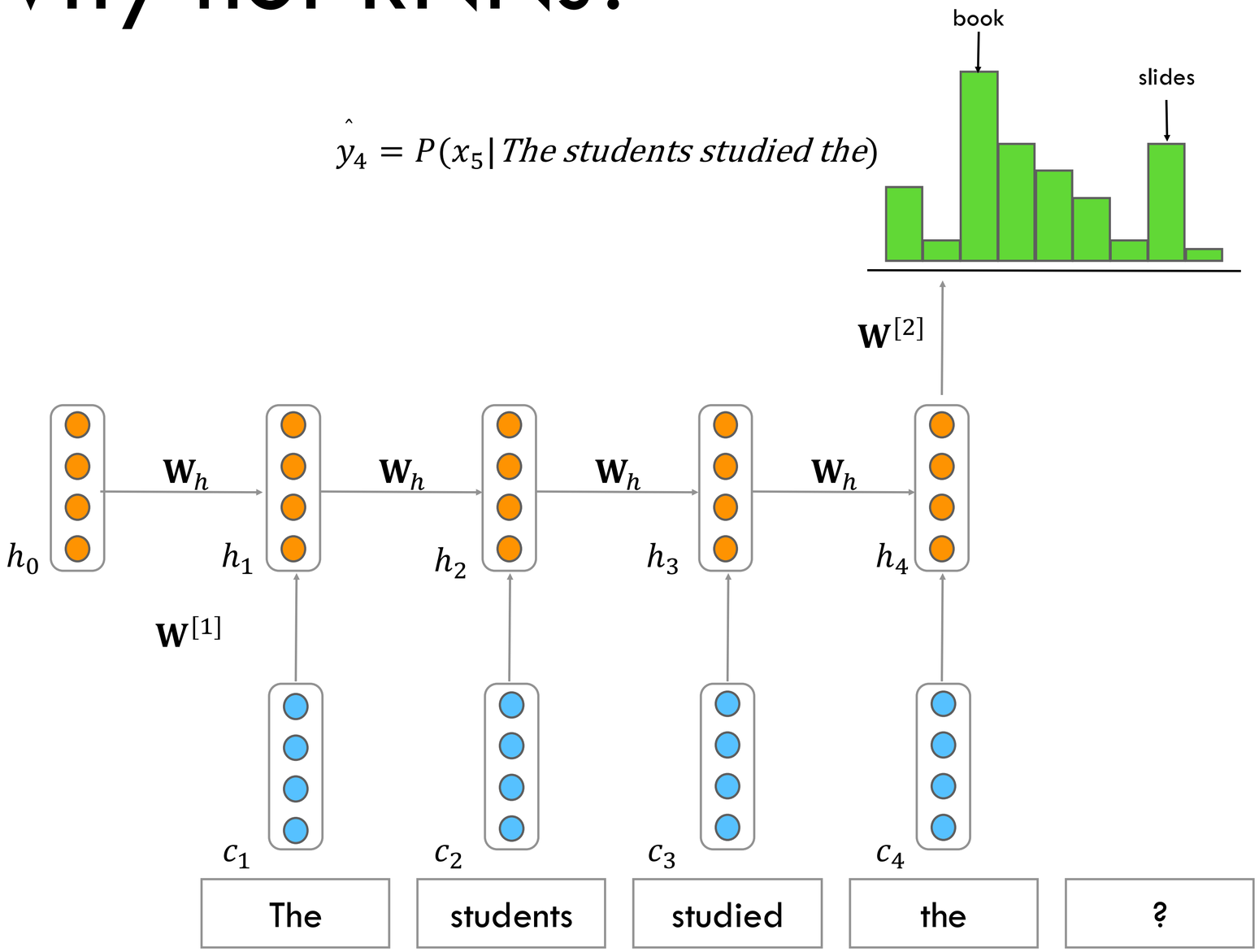
- Can process any length input
- Model size doesn't increase for longer input
- Computation for step t can (in theory) use information from many steps back
- Weights $\mathbf{W}^{[1]}$ are shared (tied) across timesteps → Condition the neural network on all previous words



Why not RNNs?

RNN Disadvantages:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back



Training RNNLMs

Training Outline

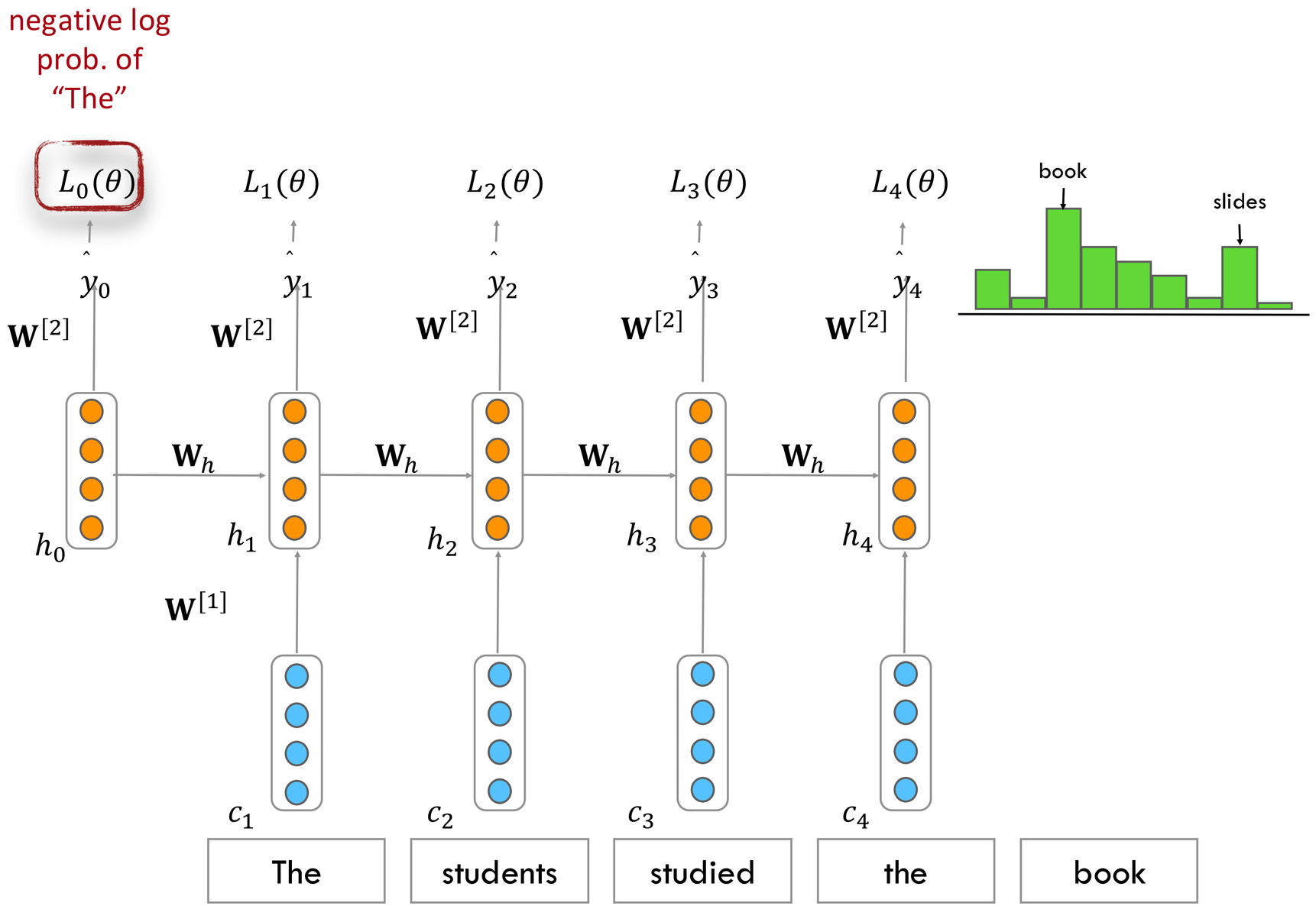
- Get a big corpus of text which is a sequence of words $x_1, x_2, \ldots x_T$
- Feed into RNN-LM; compute output distribution $\hat{y}_t$ for every step t
  - i.e. predict probability distribution of every word, given words so far
- Loss function on step t is usual cross-entropy between our predicted probability distribution $\hat{y}_t$, and the true next word $y_t = x_{t+1}$:

$$L_{CE}(\hat{y}_t, y_t; \theta) = -\sum_{v \in V} \mathbb{I}[y_t = v] \log \hat{y}_t = -\log p_\theta(x_{t+1} | x_{\leq t})$$

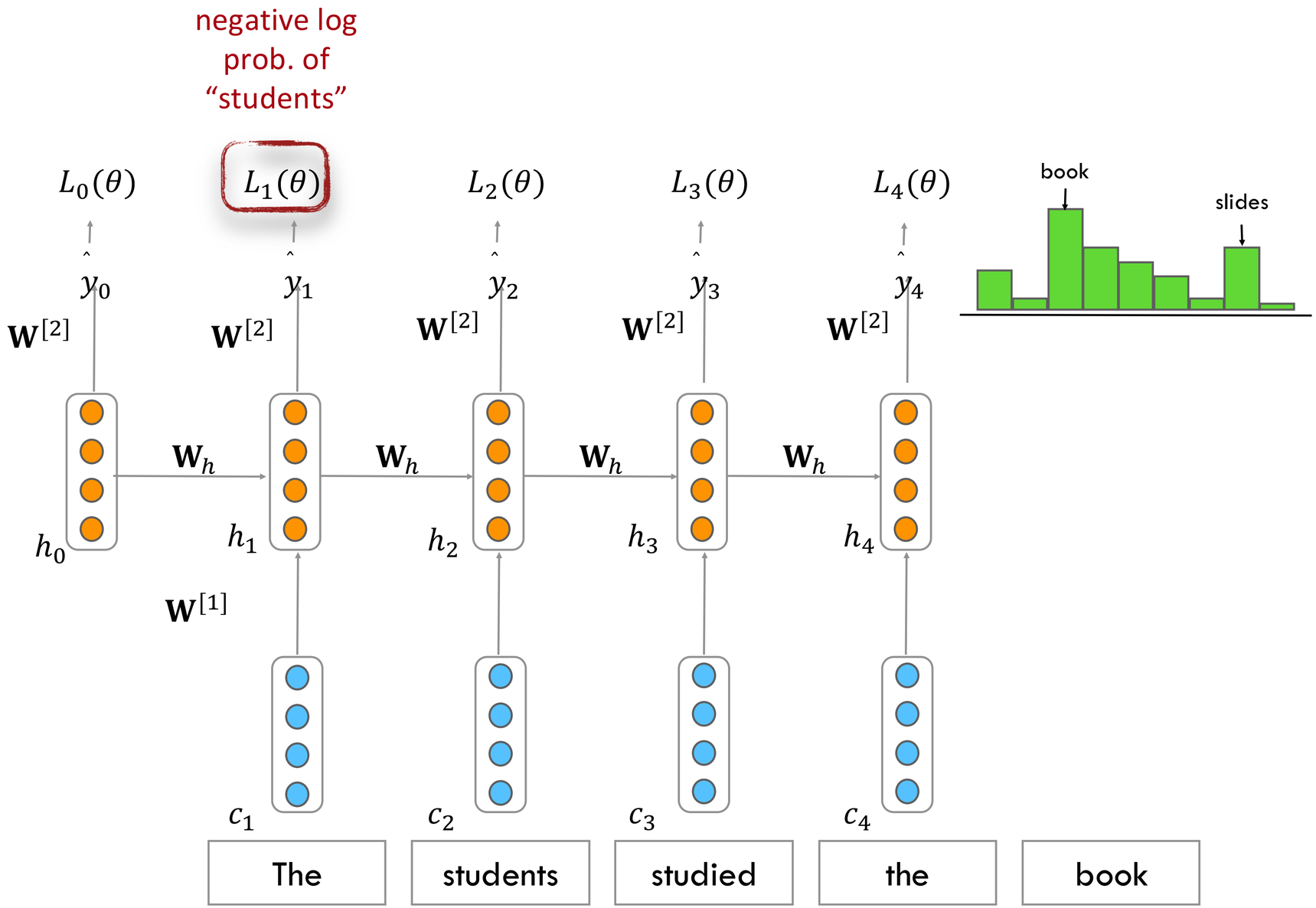
- Average this to get overall loss for entire training set:

$$L(\theta) = \frac{1}{T} \sum_{t=1}^{T} L_{CE}(\hat{y}_t, y_t)$$

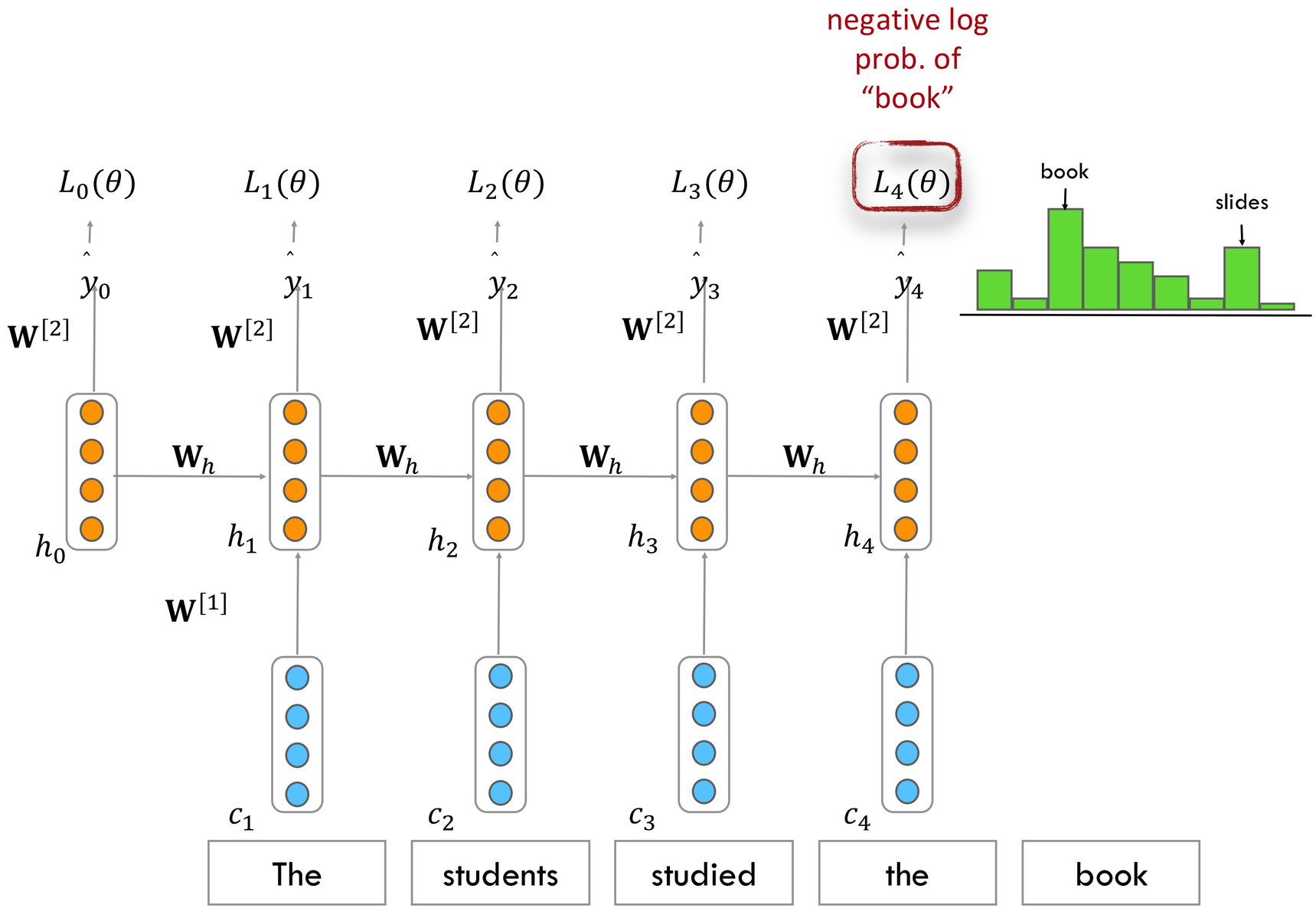
Loss


negative log prob. of "The"
$L_0(\theta)$
$L_1(\theta)$
$L_2(\theta)$
$L_3(\theta)$
$L_4(\theta)$
$\hat{y}_0$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_4$
$\mathbf{W}^{[2]}$
$\mathbf{W}_h$
$\mathbf{W}^{[1]}$
h_0
h_1
h_2
h_3
h_4
c_1
c_2
c_3
c_4
The
students
studied
the
book
book
slides

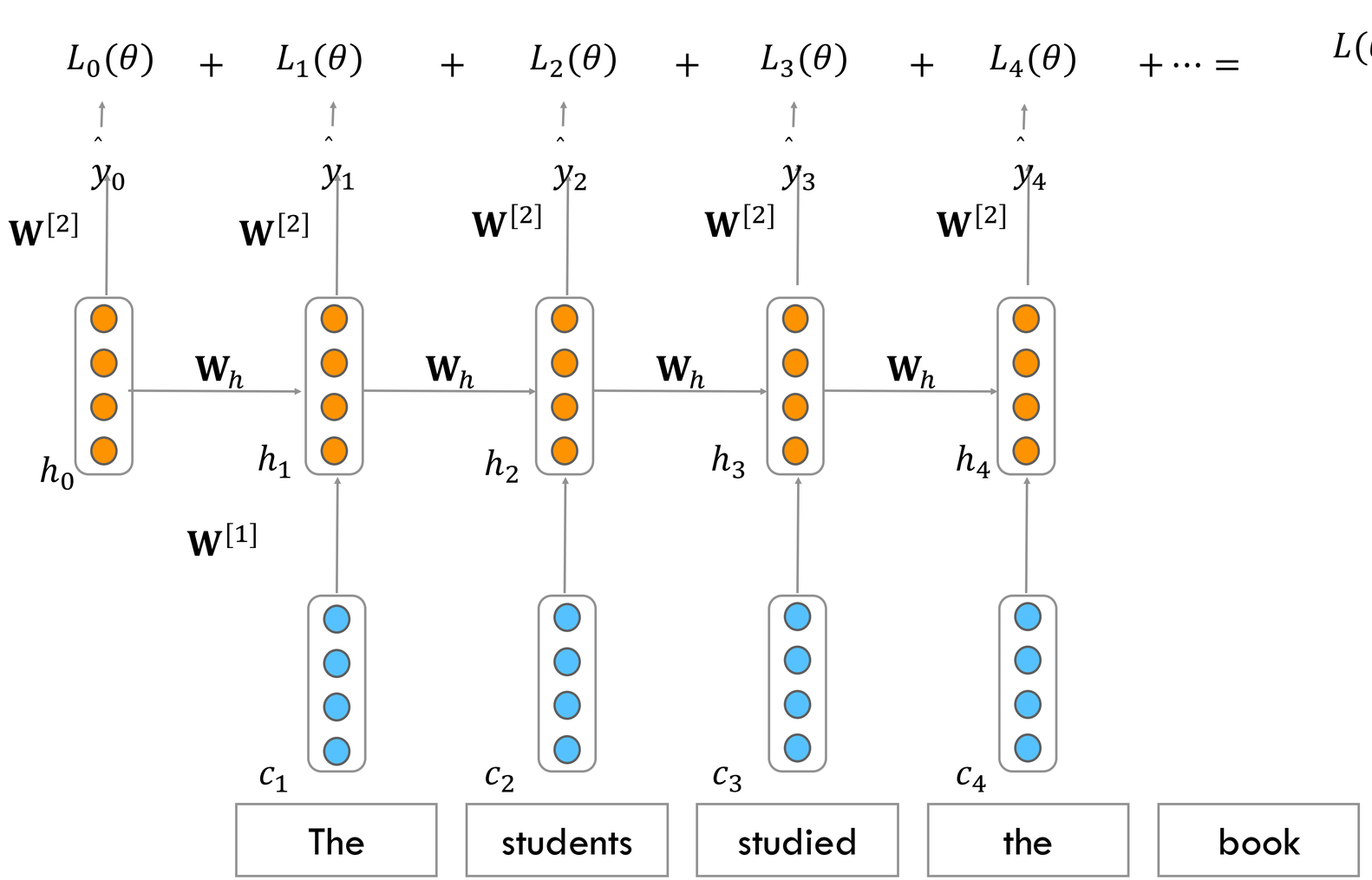
Loss


negative log prob. of "students"
$L_0(\theta)$
$L_1(\theta)$
$L_2(\theta)$
$L_3(\theta)$
$L_4(\theta)$
$\hat{y}_0$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_4$
$\mathbf{W}^{[2]}$
$\mathbf{W}_h$
$\mathbf{W}^{[1]}$
h_0
h_1
h_2
h_3
h_4
c_1
c_2
c_3
c_4
The
students
studied
the
book
book
slides

Loss


negative log prob. of "book"
$L_0(\theta)$
$L_1(\theta)$
$L_2(\theta)$
$L_3(\theta)$
$L_4(\theta)$
book
slides
$\hat{y}_0$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_4$
$\mathbf{W}^{[2]}$
$\mathbf{W}_h$
$\mathbf{W}^{[1]}$
h_0
h_1
h_2
h_3
h_4
c_1
c_2
c_3
c_4
The
students
studied
the
book

Loss


$L_0(\theta)$ + $L_1(\theta)$ + $L_2(\theta)$ + $L_3(\theta)$ + $L_4(\theta)$ + ⋯ =
$L(\theta) = \frac{1}{T}\sum_{t=1}^{T} L_t(\theta)$
$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$
$\mathbf{W}^{[2]}$
$\mathbf{W}_h$
h_0 h_1 h_2 h_3 h_4
$\mathbf{W}^{[1]}$
c_1 c_2 c_3 c_4
The students studied the book

Training RNNs is hard

- Multiply the same matrix at each time step during forward propagation
- Ideally inputs from many time steps ago can modify output y
- This leads to something called the vanishing gradient problem

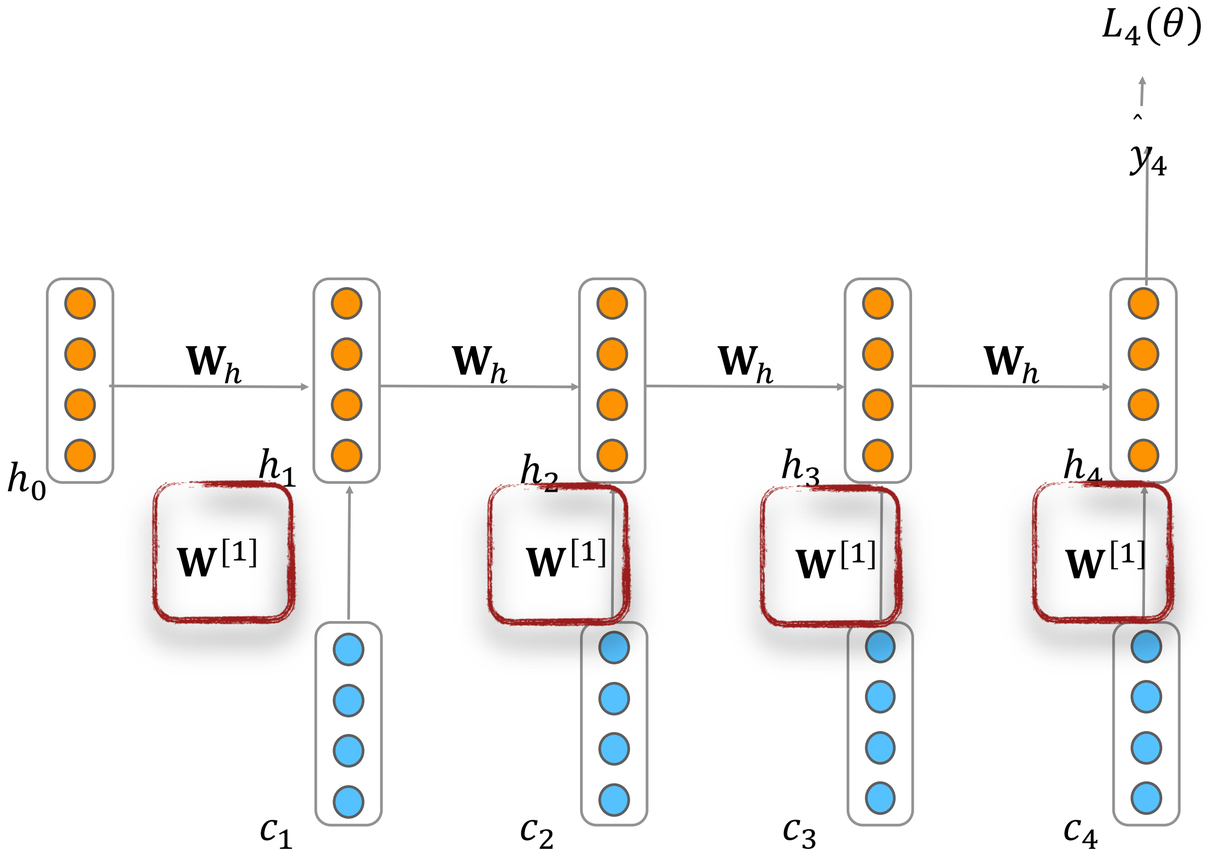


Table 2. *Comparison of different neural network architectures on Penn Corpus (1M words) and Switchboard (4M words).*

	Penn Corpus		Switchboard	
Model	NN	NN+KN	NN	NN+KN
KN5 (baseline)	-	141	-	92.9
feedforward NN	141	118	85.1	77.5
RNN trained by BP	137	113	81.3	75.4
RNN trained by BPTT	123	106	77.5	72.5

Next Class: Transformer Language Models!

T. **Mikolov**, S. Kombrink, L. Burget, J. Černocký and S. Khudanpur, "Extensions of recurrent neural network language model," *2011 IEEE ICASSP*, doi: 10.1109/ICASSP.2011.5947611.

Welcome

CSCI 662 Spring 2026: NLP

🐝 Spring 2026 ⏰ Wed 2:00 - 5:20pm 📍 WPH 102

- TODOs for you
  - Finalize project teams (due on Jan 28)
  - Brainstorm project ideas
    - Project pitch on Feb 4

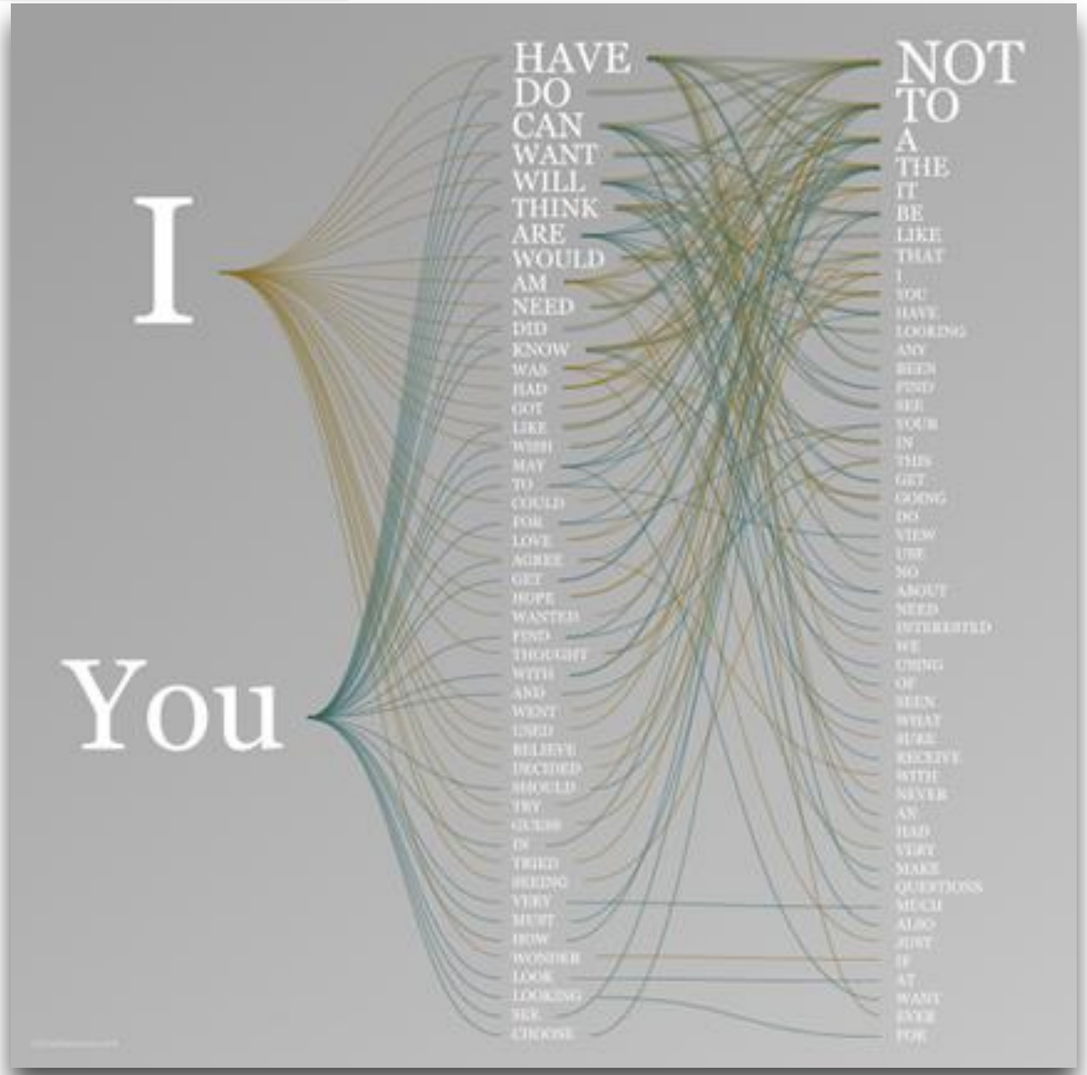


Image Courtesy: Chris Harrison