CS 1671 / CS 2071 / ISSP 2071
Human Language Technologies

Session 14: Neural networks part 2

Michael Miller Yoder

March 2, 2026

School of Computing and Information

Course logistics

- Project proposal presentations will be in class **this Wed Mar 4**
  - Add your group's slides to this shared presentation
  - Maximum 5 minutes, please
  - I will give you feedback on your proposal by then

Course logistics

- Homework 2 is due **Mar 17** (Tue after spring break)
  - Implement a logistic regression classifier with different features
- Class recordings are available at **Canvas > Panopto**

Review: neural networks

Discuss with a neighbor:

1. Describe the steps of computation that occurs in the “forward pass” of a neural network to make predictions
2. How are word vectors like word2vec trained to roughly represent word meaning?

Overview: neural networks part 2

- Feedforward neural networks with word embedding input
- Training neural networks
- Coding activity

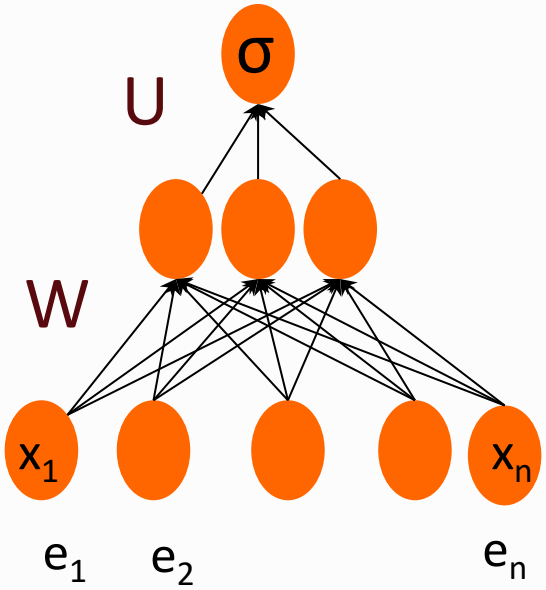
Feedforward neural networks with word embedding input

Even better: representation learning

The real power of deep learning comes from the ability to **learn** features from the data

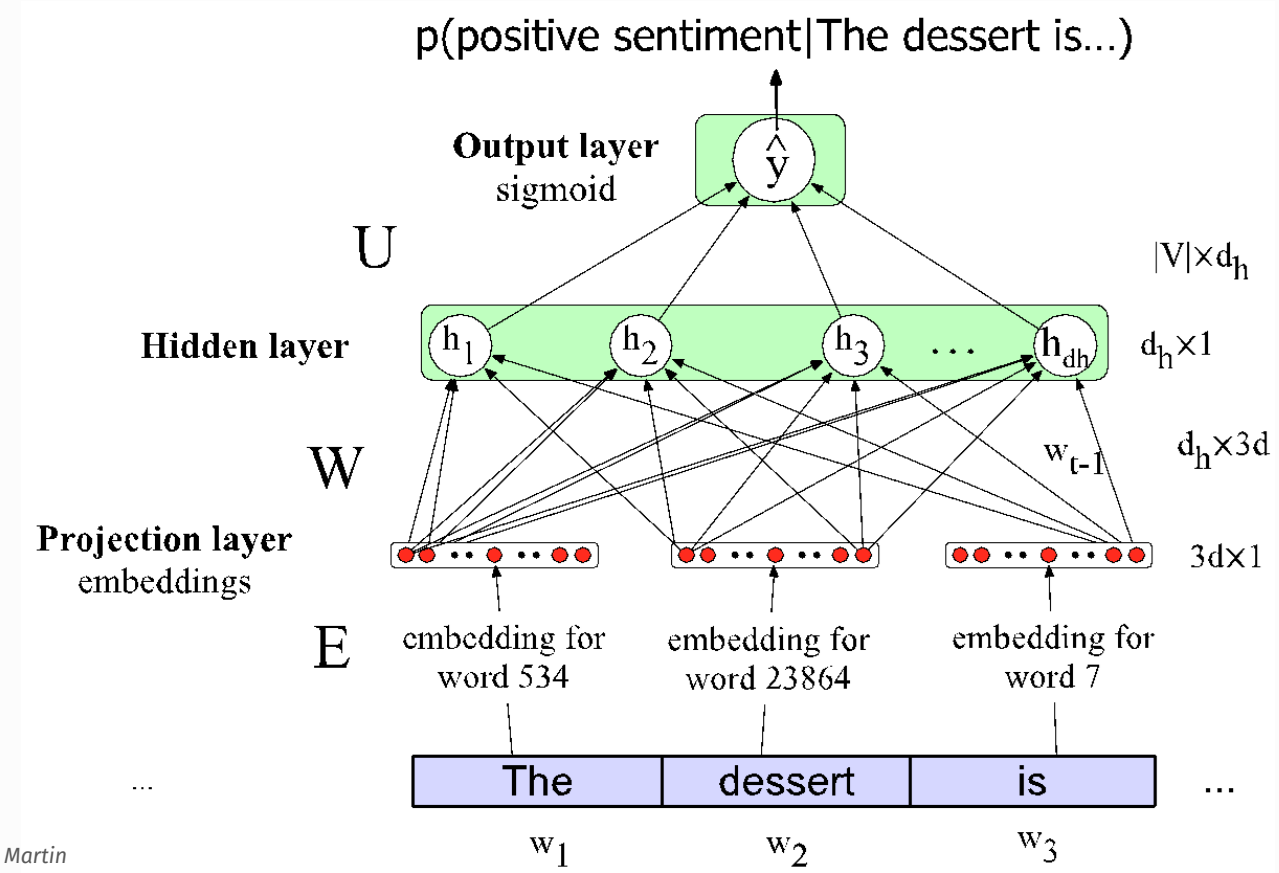
Instead of using hand-built human-engineered features for classification

Use learned representations like embeddings!



Slide adapted from Jurafsky & Martin

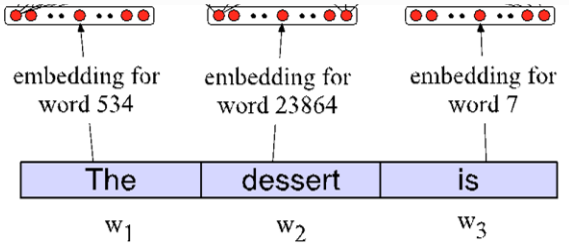
Neural net classification with embeddings as input features!



Slide adapted from Jurafsky & Martin

Issue: texts come in different sizes

This assumes a fixed size length (3)!
Kind of unrealistic.
Some simple solutions:



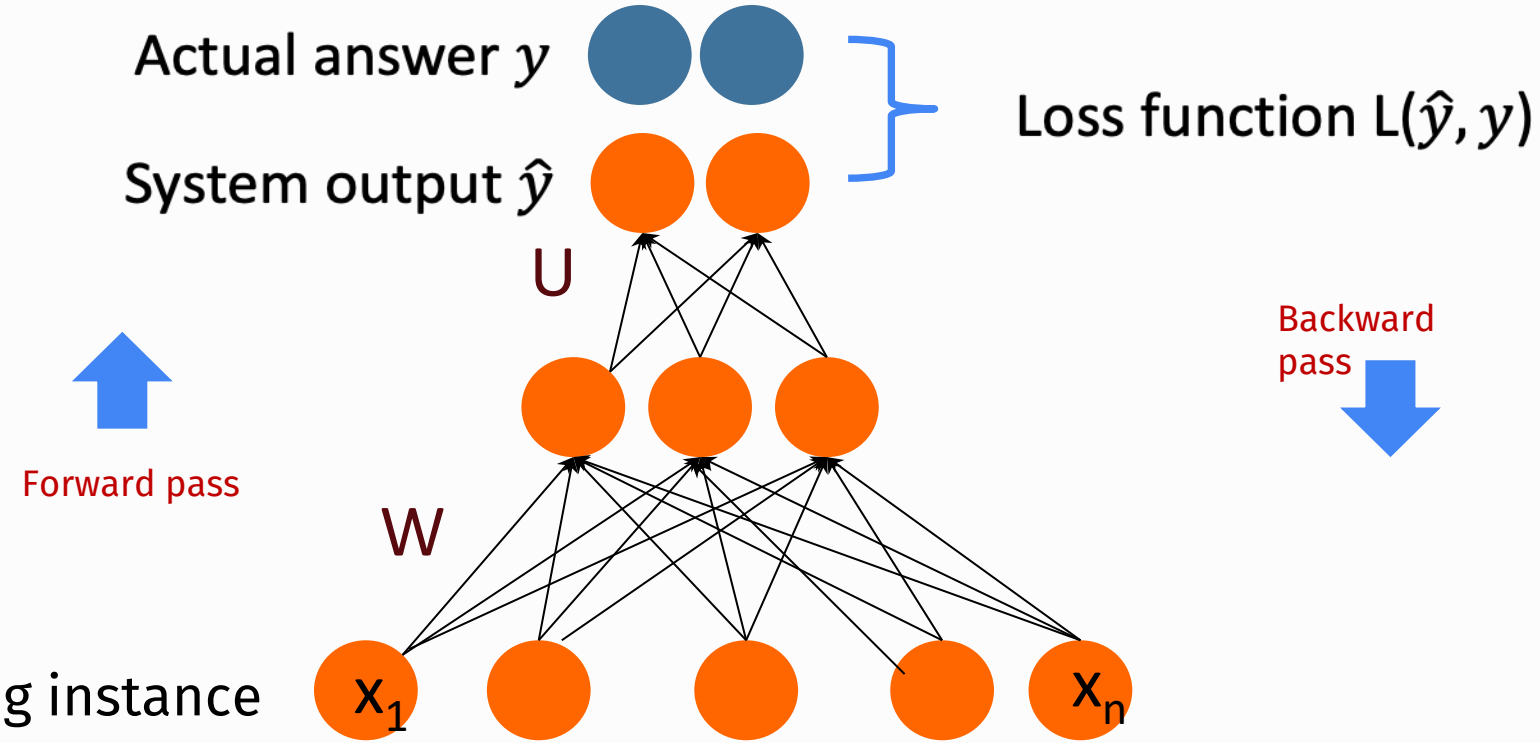
1. Make the input the length of the longest review
   - If shorter then pad with zero embeddings
   - Truncate if you get longer reviews at test time

2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
   - Take the mean of all the word embeddings
   - Take the element-wise max of all the word embeddings
     - For each dimension, pick the max value from all words

Slide adapted from Jurafsky & Martin

Training feedforward neural networks

Intuition: training a 2-layer Network



Slide adapted from Jurafsky & Martin

Remember stochastic gradient descent from the logistic regression lecture—find gradient and optimize

Slide credit: David Mortensen

The Intuition Behind Training a 2-Layer Network

For every training tuple (x, y)

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

Slide credit: David Mortensen

Computing the gradient requires finding the derivative of the loss with respect to each weight in every layer of the network.

Error backpropagation through computation graphs.

Slide credit: David Mortensen

Reminder: gradient descent for weight updates

Use the derivative of the loss function with respect to weights $\frac{d}{dw} L(f(x;w),y)$

To tell us how to adjust weights for each training item

- Move them in the opposite direction of the gradient

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{d}{dw} L_{CE}(f(\mathbf{x};\mathbf{w}),y)$$

- For logistic regression

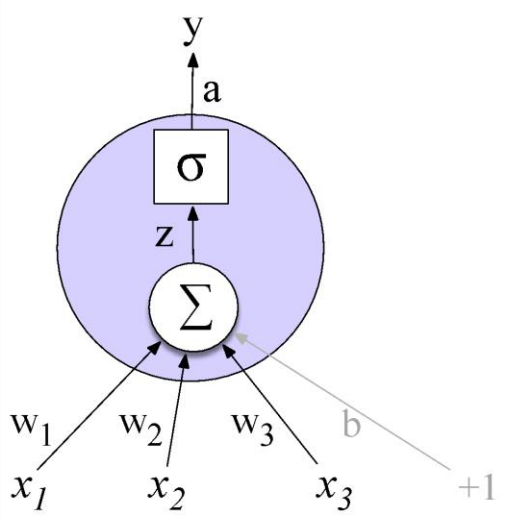
$$\frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

Slide adapted from Jurafsky & Martin

Where did that derivative come from?

Using the chain rule of derivatives! $f(x) = u(v(x))$

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$



Derivative of the weighted sum

Derivative of the Activation

Derivative of the Loss

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i}$$

Slide adapted from Jurafsky & Martin

How can I find that gradient for every weight in the network?

These derivatives on the prior slide only give the updates for one weight layer: the last one!

What about deeper networks? For training, we need the derivative of the loss with respect to each weight in every layer of the network

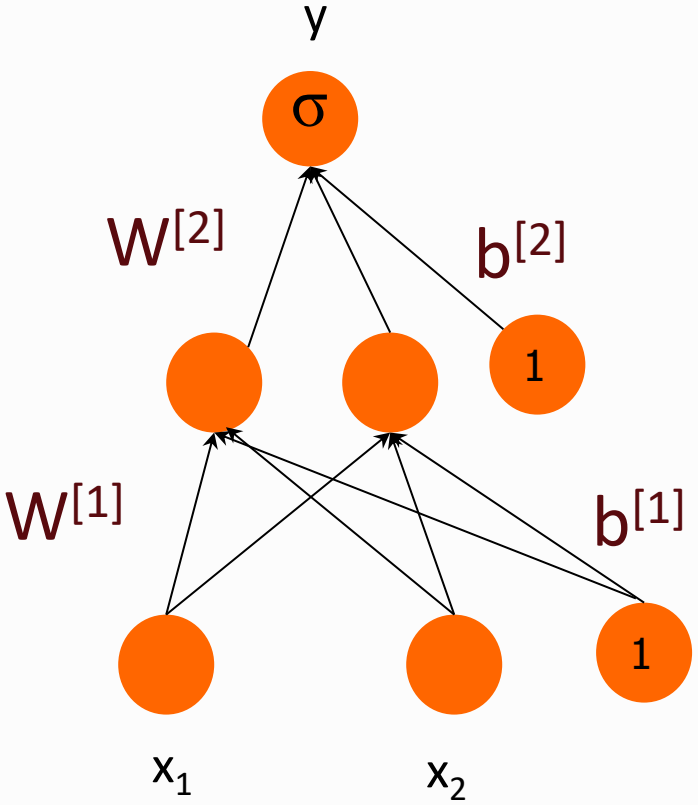
- Lots of layers, different activation functions?
- But the loss is computed only at the very end of the network!

Solution:

- Even more use of the chain rule!!
- This process is called **error backpropagation** (Rumelhart et al 1986)

Slide adapted from Jurafsky & Martin

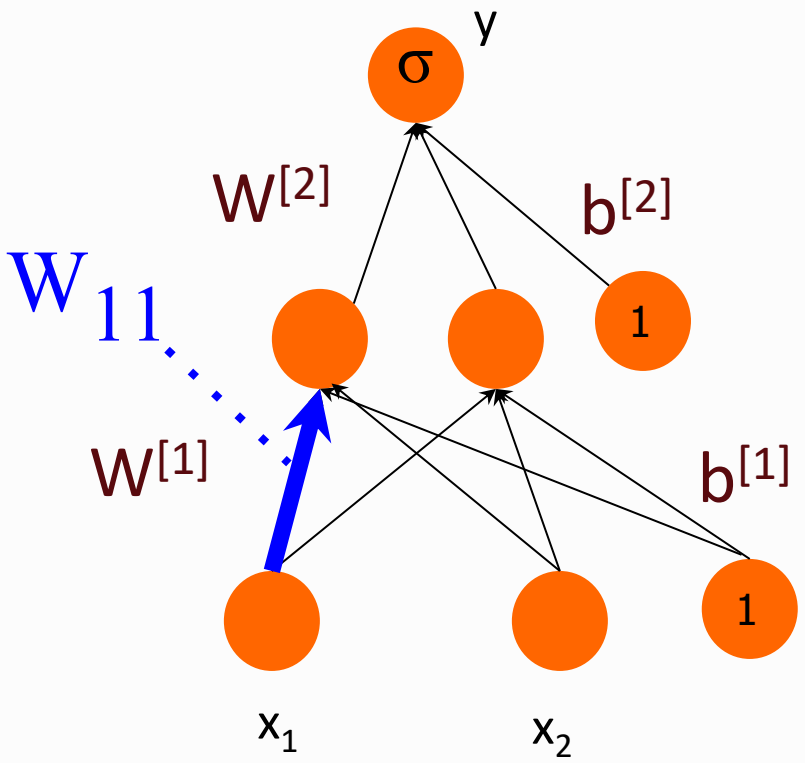
Backpropagation on a two layer network



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

Slide adapted from Jurafsky & Martin

Backpropagation on a two layer network

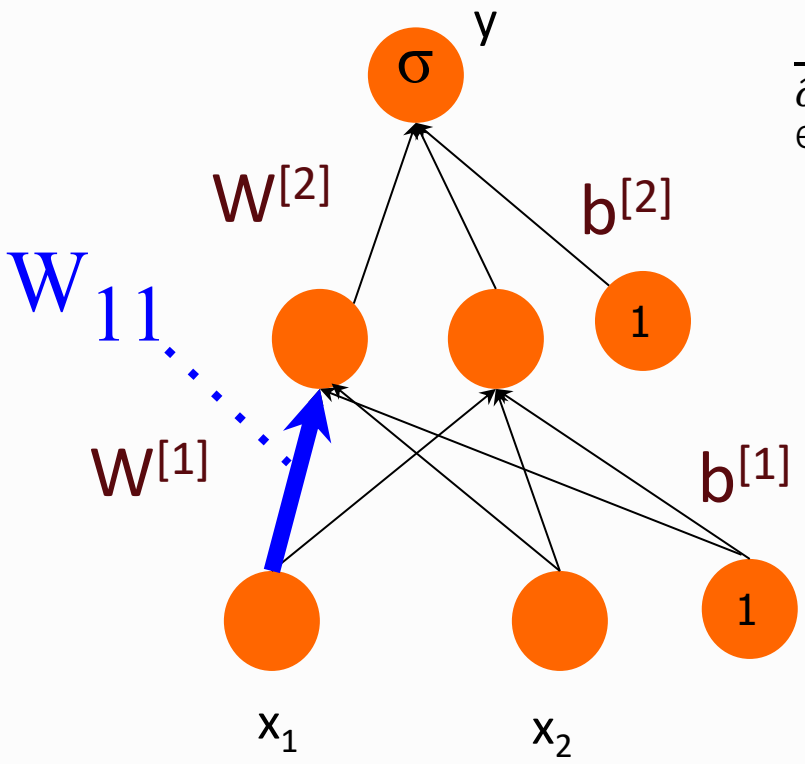


$$
\begin{aligned}
z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
a^{[1]} &= \text{ReLU}(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$

$$
\frac{\partial L}{\partial W_{11}} = \frac{\partial L}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial W_{11}^{[1]}}
$$

Slide adapted from Jurafsky & Martin

Backpropagation on a two layer network



$\frac{\partial L}{\partial a^{[2]}} = -\left(\frac{y}{a^{[2]}} + \frac{y-1}{1-a^{[2]}}\right)$ from derivative of cross-entropy loss

$\frac{\partial a^{[2]}}{\partial z^{[2]}} = a^{[2]}(1-a^{[2]})$ from derivative of sigmoid

$\frac{\partial z^{[2]}}{\partial a^{[1]}} = W_{11}^{[2]}$ from derivative of weighted sum

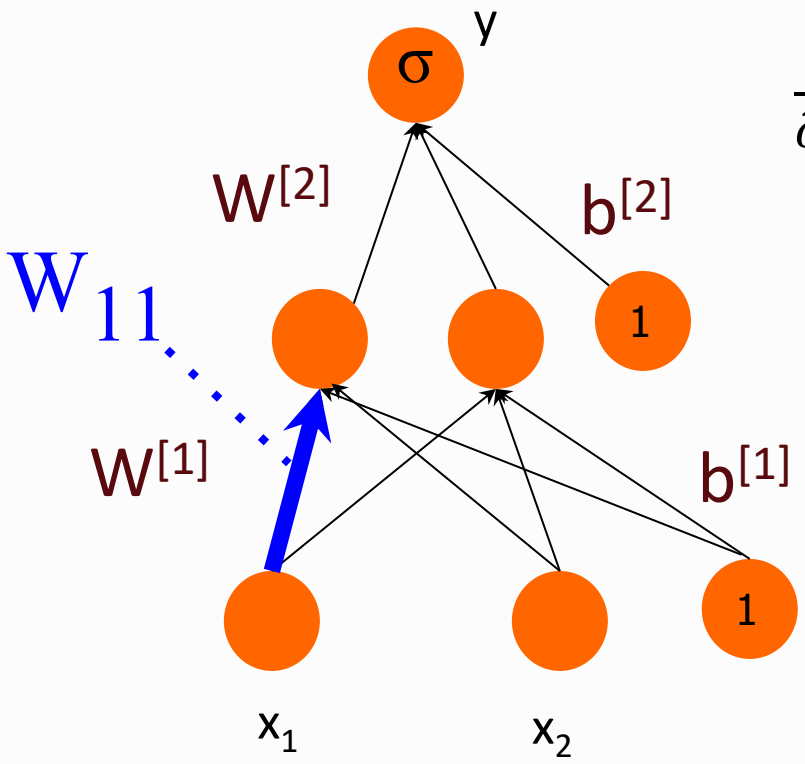
$\frac{\partial a^{[1]}}{\partial z^{[1]}} = \begin{cases} 0 \; for \; z^{[1]} < 0 \\ 1 \; for \; z^{[1]} \geq 0 \end{cases}$ from derivative of RELU

$\frac{\partial z^{[1]}}{\partial W_{11}^{[1]}} = x_1$ from derivative of product

$$\frac{\partial L}{\partial W_{11}} = \frac{\partial L}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial W_{11}^{[1]}}$$

Slide adapted from Jurafsky & Martin

Backpropagation on a two layer network



$$\frac{\partial L}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} = a^{[2]} - y$$

$$\frac{\partial a^{[2]}}{\partial z^{[2]}} = a^{[2]}(1 - a^{[2]})$$ from derivative of sigmoid

$$\frac{\partial z^{[2]}}{\partial a^{[1]}} = W_{11}^{[2]}$$ from derivative of weighted sum

$$\frac{\partial a^{[1]}}{\partial z^{[1]}} = \begin{cases} 0 \; for \; z^{[1]} < 0 \\ 1 \; for \; z^{[1]} \geq 0 \end{cases}$$ from derivative of RELU

$$\frac{\partial z^{[1]}}{\partial W_{11}^{[1]}} = x_1$$ from derivative of product

$$\frac{\partial L}{\partial W_{11}} = \frac{\partial L}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial W_{11}^{[1]}}$$

Slide adapted from Jurafsky & Martin

Backpropagation on a two layer network

$y = 1$
$a^{[2]} = 2.5$
$W_{11}^{[2]} = 1.5$
$z^{[1]} = 3$
$x_1 = 4.2$

$$\frac{\partial L}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} = a^{[2]} - y$$

$$\frac{\partial a^{[2]}}{\partial z^{[2]}} = a^{[2]}(1 - a^{[2]})$$ from derivative of sigmoid

$$\frac{\partial z^{[2]}}{\partial a^{[1]}} = W_{11}^{[2]}$$ from derivative of weighted sum

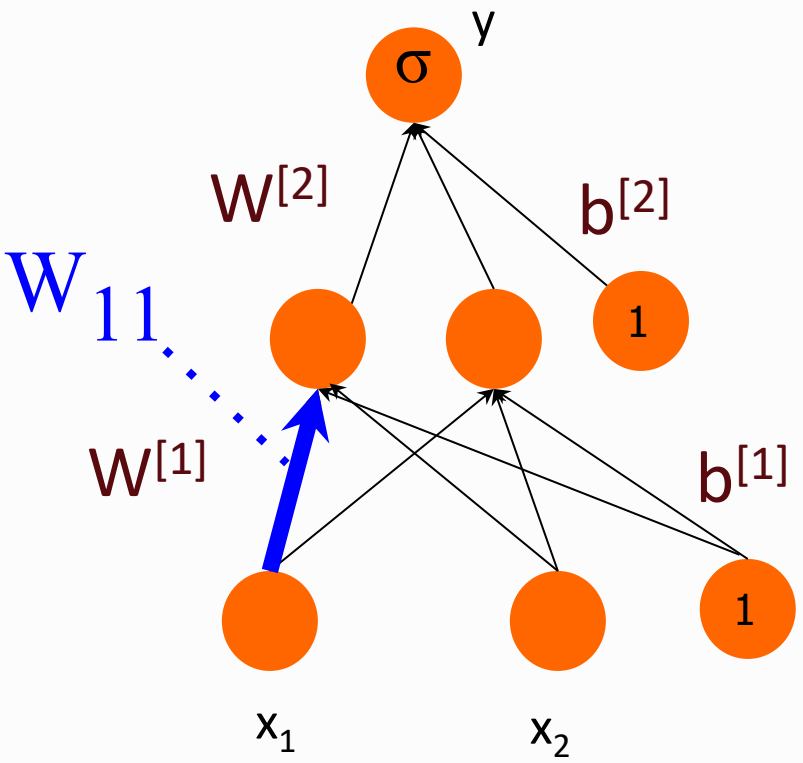
$$\frac{\partial a^{[1]}}{\partial z^{[1]}} = \begin{cases} 0 \; for \; z^{[1]} < 0 \\ 1 \; for \; z^{[1]} \geq 0 \end{cases}$$ from derivative of RELU

$$\frac{\partial z^{[1]}}{\partial W_{11}^{[1]}} = x_1$$ from derivative of product

$$\frac{\partial L}{\partial W_{11}} = \frac{\partial L}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial W_{11}^{[1]}}$$

Slide adapted from Jurafsky & Martin

Backpropagation on a two layer network



$$\frac{\partial L}{\partial W_{11}} = -22.125$$

$$W_{11,t+1} =$$

Slide adapted from Jurafsky & Martin

Summary

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: **backpropagation**

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

Muddiest point: Backprop is confusing!

With a neighbor, discuss what questions you have about backpropagation.

Coding activity

Notebook: feedforward neural network

1. Go to this nbgitpuller link (also available on course website)
2. Start a server with **TEACH – 6 CPUs, 48 GB**
3. Load custom environment at **/ix1/cs1671-2026s/class_env**
4. Open **session14_ffnn.ipynb**