Backpropagation – calculus and computation graphs

CS 4740 (and crosslists): Introduction to Natural Language Processing
https://courses.cs.cornell.edu/cs4740/2025sp

Slides developed by:
Magd Bayoumi, Claire Cardie, Tanya Goyal, Dan Jurafsky, Lillian Lee, James Martin, Marten van Schijndel

Announcements

- Hwk2

  Milestone due this Weds 11:59pm

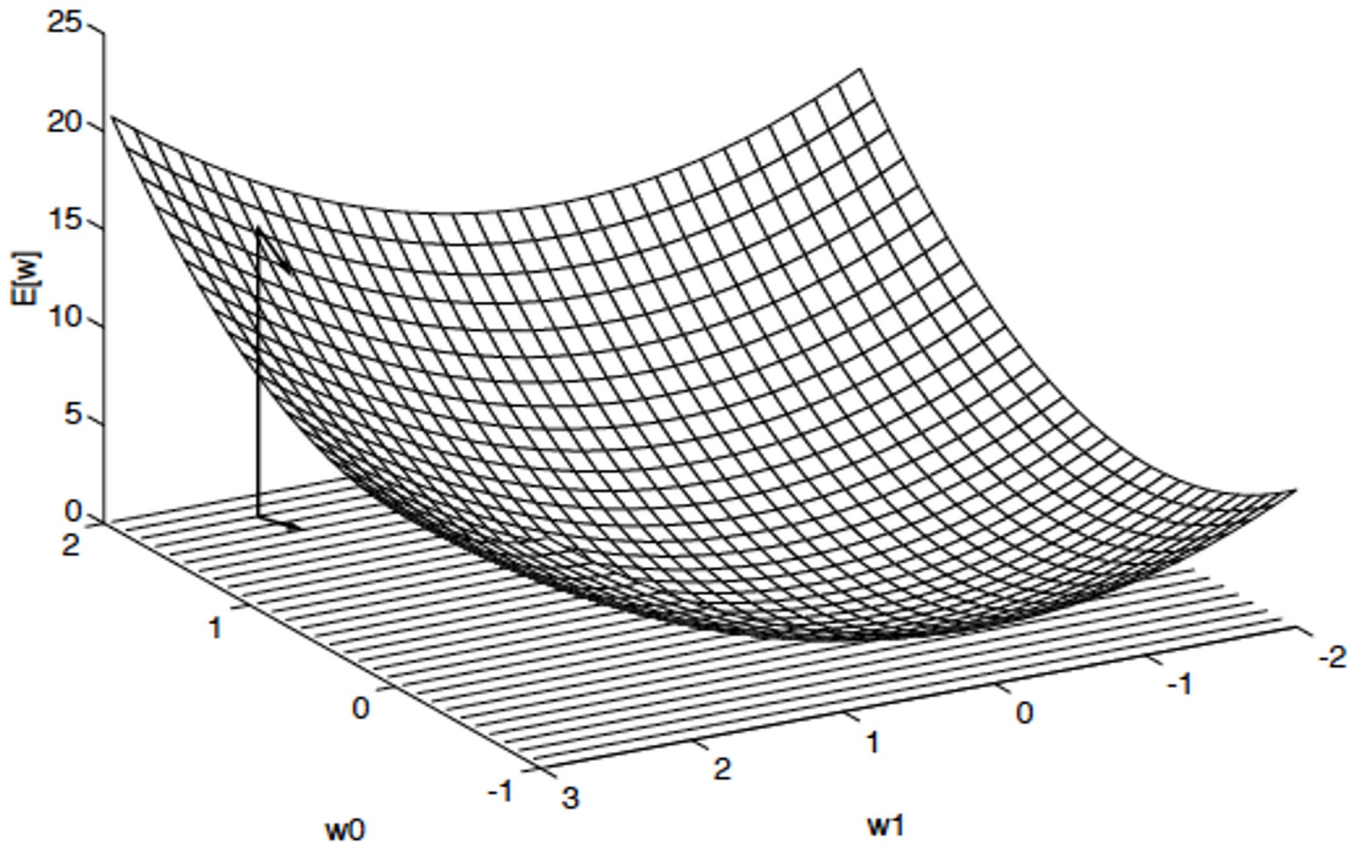
Today

- Calculus version of error backpropagation
  - Including some cool visualizations
  - Employs computation graphs

Recall the goal: Updating the weights

- Need an optimization algorithm for iteratively updating the weights so as to minimize the loss with respect to one or more training examples.
- The standard algorithm for this is **gradient descent**.
- The gradient of a function of many variables is a vector pointing in the direction of the greatest **increase** in function value.
- We want to **minimize** error, so…we will find the gradient of the loss function at the current point and move in the opposite direction.

Gradient Descent Through Weight Space



Understanding the gradient

Magnitude of each component in the gradient tells us how *sensitive* the cost function is to each weight and bias

↓

loss

Recap: Backpropagation

Input: *network* with randomly initialized weights
training *examples*

repeat
for each *e* in *examples*
feedforward pass to compute output
compute *error (i.e. loss)* at output layer
update weights in *network* to reduce *error*
until stopping criterion is reached
return *network*

Recall: Computing the Gradient

- Requires the partial derivative of the loss function with respect to each parameter.
- **Challenge:** We need to compute the derivative with respect to weight parameters that appear in the very early layers of the network, even though the loss is computed only at the very end of the network.
- **Solution:** error backpropagation
  - Computationally efficient algorithm (automatic differentiation)
  - Implemented using computation graphs

Computation Graphs

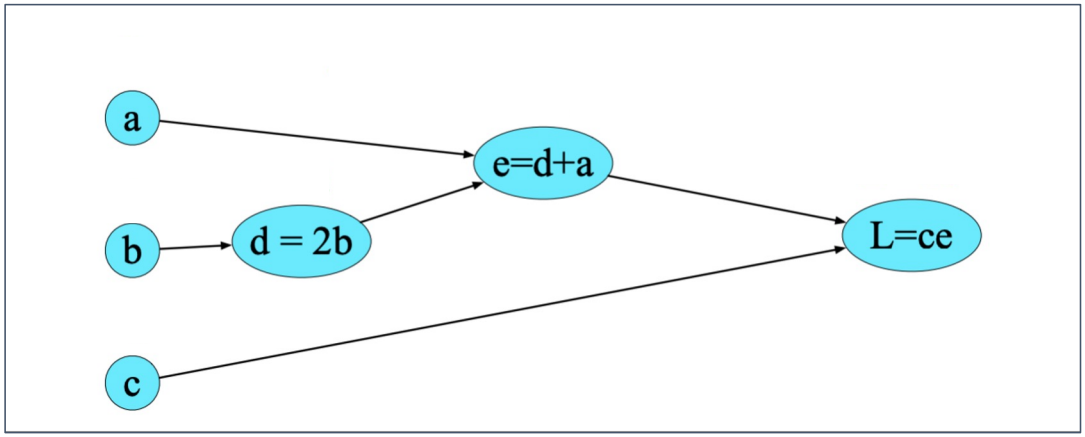
- A representation of the process of computing a mathematical expression.
- Each operation is modeled as a node in a graph.
- A node with an incoming edge is a function of that edge's tail node.
- Useful to think of computation graphs as modeling data dependencies

Computation Graph Construction

via function decomposition

$$L(a,b,c) = c(a+2b).$$

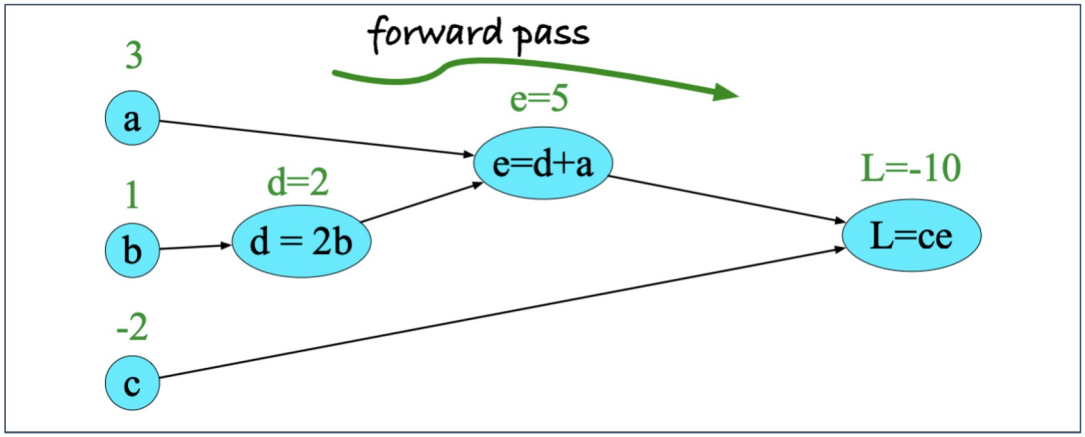
$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$



Forward Pass in Computation Graphs

$$L(a,b,c) = c(a+2b).$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$



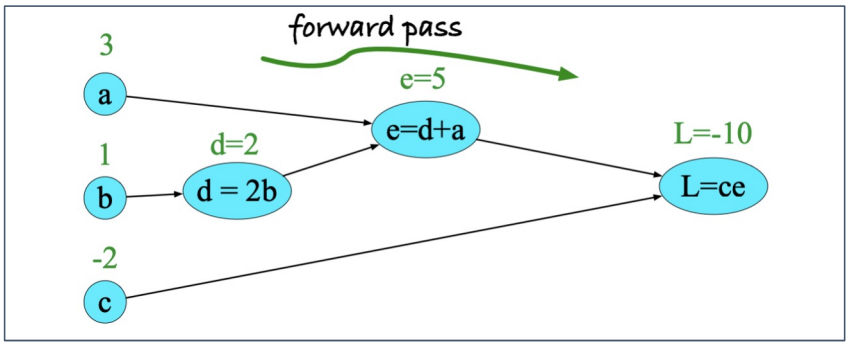
Forward pass to compute the result L(3,1,−2) = - 10

Backward Differentiation on Computation Graphs

$$L(a,b,c) = c(a+2b).$$

How can we compute the derivative of *L* w.r.t. a, b and c?

$$\frac{\partial L}{\partial a}, \frac{\partial L}{\partial b}, \text{and } \frac{\partial L}{\partial c}$$

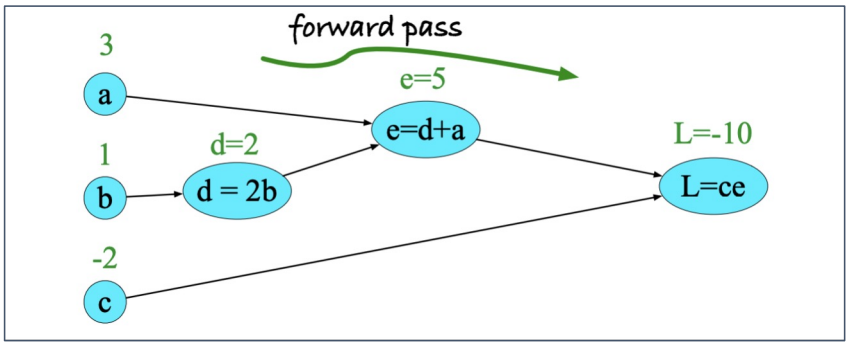


The backward pass will compute the derivatives that we'll need for the weight updates.

Backward Differentiation on Computation Graphs

$$L(a,b,c) = c(a+2b).$$

How can we compute the derivative of *L* w.r.t. a, b and c?



$$\frac{\partial L}{\partial a}, \frac{\partial L}{\partial b}, \text{ and } \frac{\partial L}{\partial c}$$

Main idea: Use chain rule

$$f(x) = u(v(x)) \qquad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$

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

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

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

$$L(a,b,c) = c(a+2b).$$

Main idea: Use chain rule

$$f(x) = u(v(x)) \qquad \frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$

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

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

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

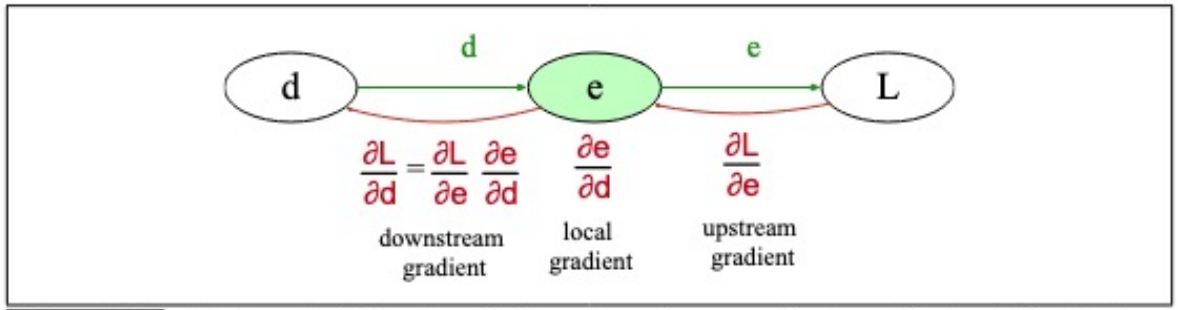


Figure 7.13 Each node (like *e* here) takes an upstream gradient, multiplies it by the local gradient (the gradient of its output with respect to its input), and uses the chain rule to compute a downstream gradient to be passed on to a prior node. A node may have multiple local gradients if it has multiple inputs.

$$L(a,b,c) = c(a+2b).$$

Main idea: Use chain rule

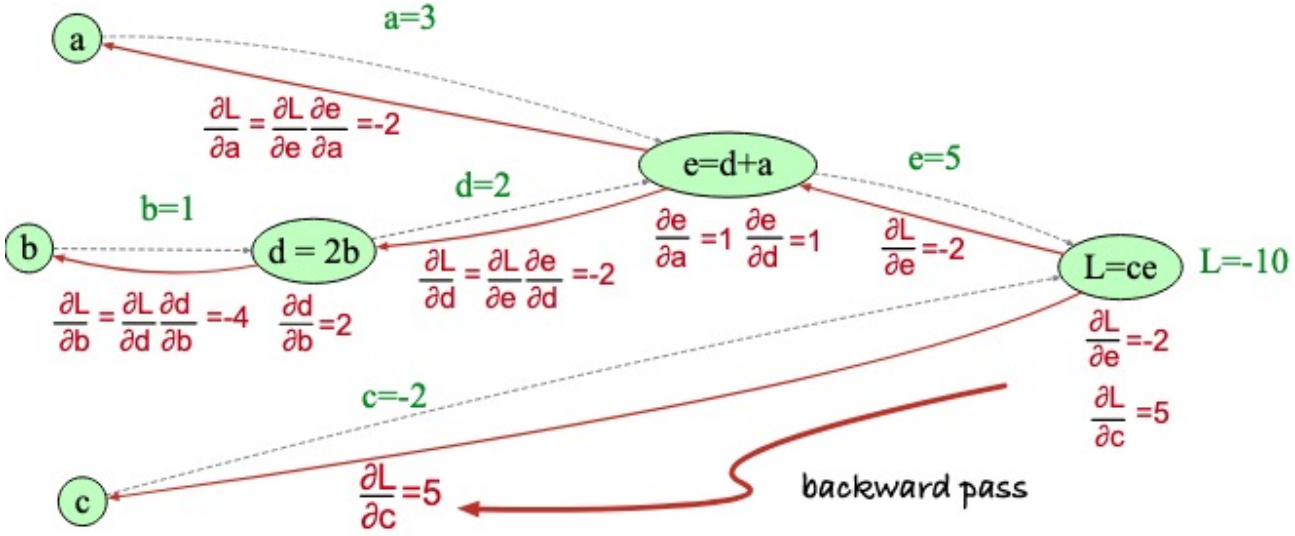
$$f(x) = u(v(x)) \qquad \frac{df}{dx} = \frac{du}{dv}\cdot\frac{dv}{dx}$$

$$\frac{\partial L}{\partial c} = e \qquad \frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \qquad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

$$L = ce \;:\quad \frac{\partial L}{\partial e} = c,\ \frac{\partial L}{\partial c} = e$$

$$e = a + d \;:\quad \frac{\partial e}{\partial a} = 1,\ \frac{\partial e}{\partial d} = 1$$

$$d = 2b \;:\quad \frac{\partial d}{\partial b} = 2$$



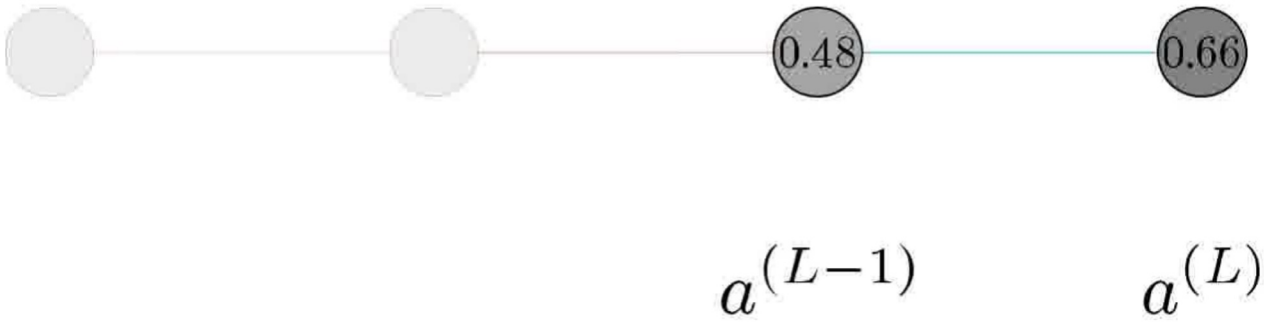
Computing the Gradient with Backpropagation

Start with a very simple network

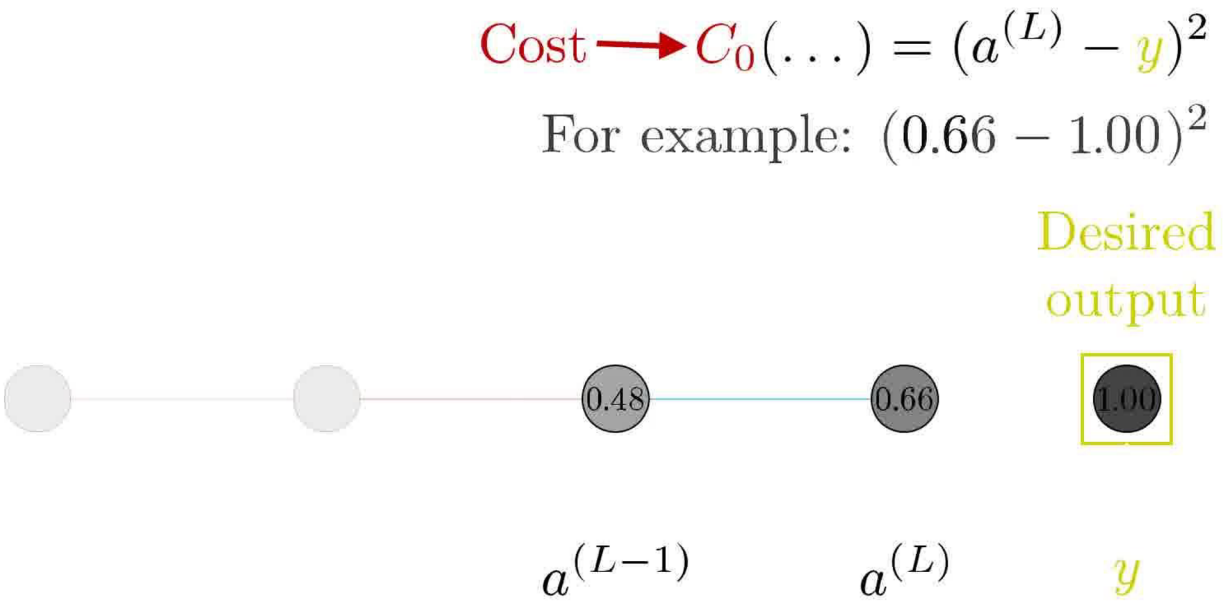
Computing the Gradient with Backpropagation

$$w_1 \quad b_1 \quad w_2 \quad b_2 \quad w_3 \quad b_3$$

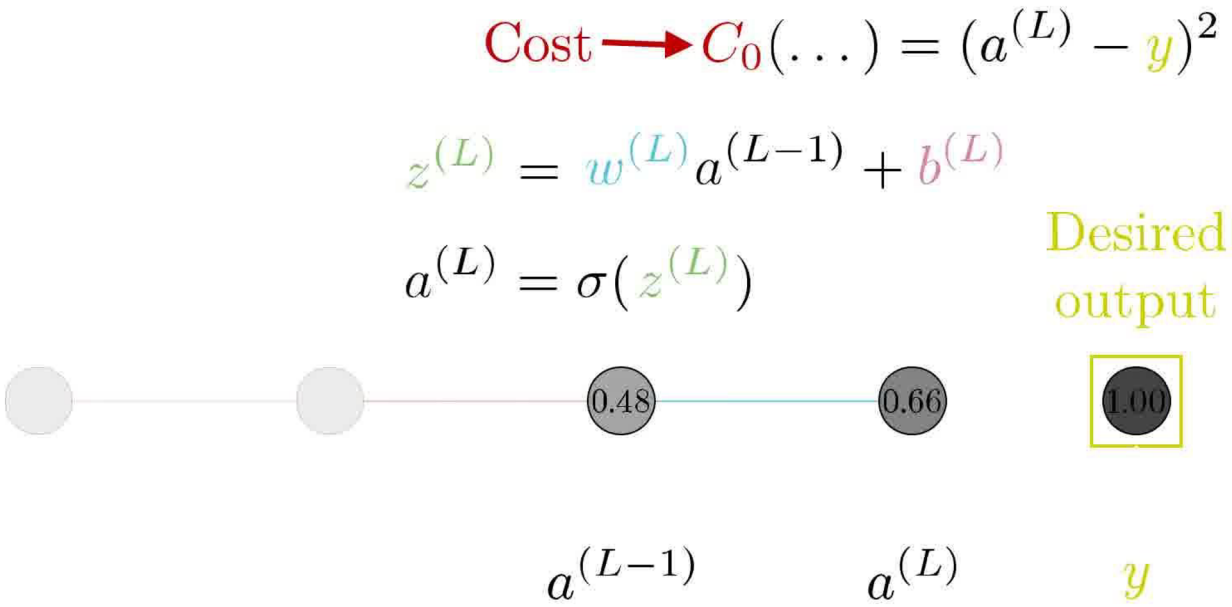
Computing the Gradient with Backpropagation



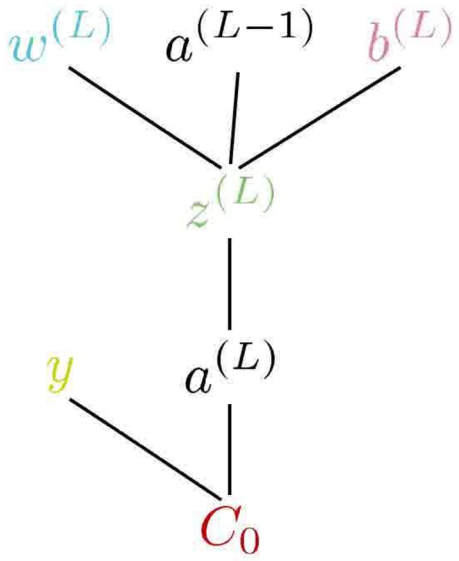
Computing the Gradient with Backpropagation



Computing the Gradient with Backpropagation



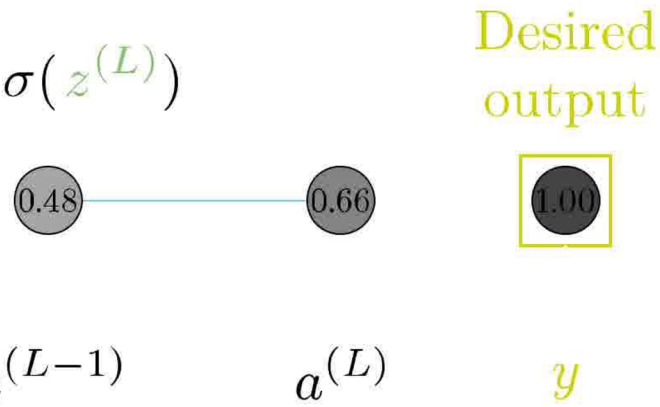
Computing the Gradient with Backpropagation



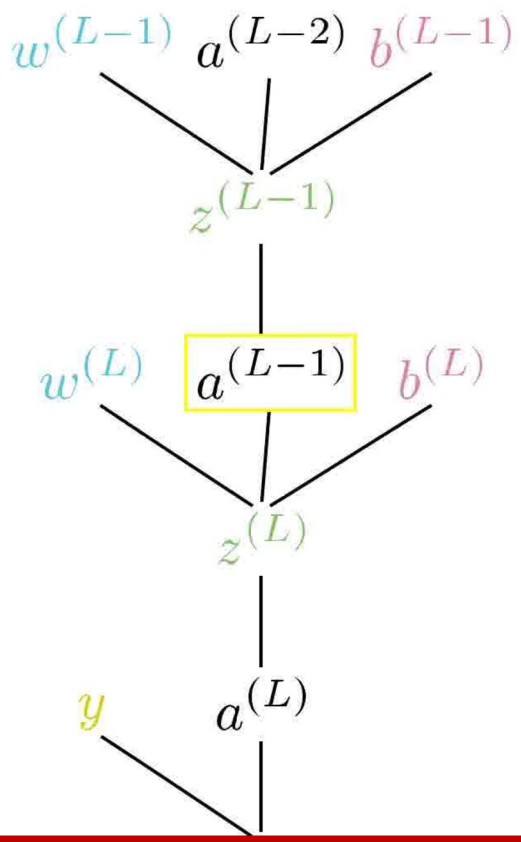
$$\text{Cost} \rightarrow C_0(\dots) = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$



Computing the Gradient with Backpropagation

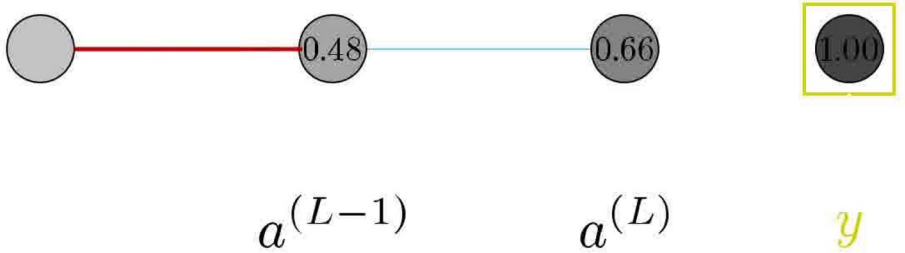


$$\text{Cost} \rightarrow C_0(\dots) = (a^{(L)} - y)^2$$

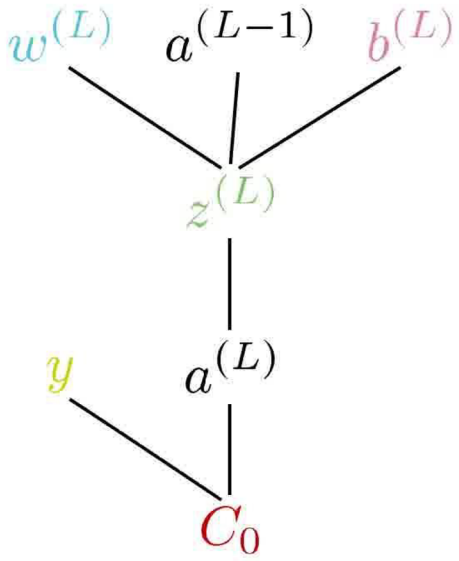
$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$

Desired output



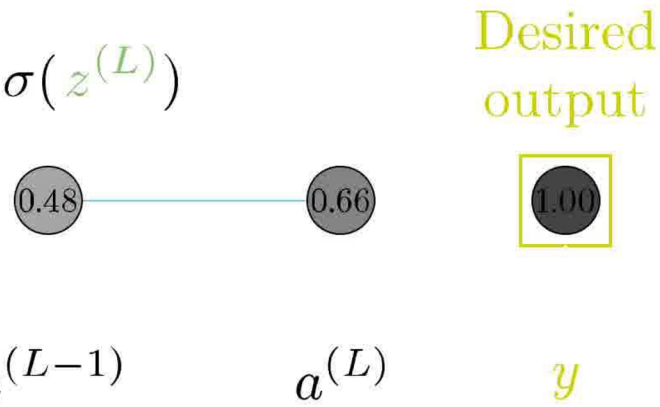
Computing the Gradient with Backpropagation



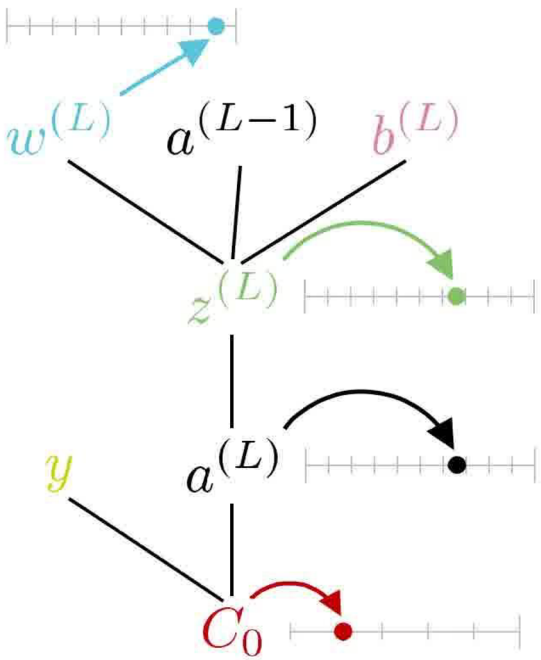
$$\text{Cost} \rightarrow C_0(\dots) = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$



Computing the Gradient with Backpropagation

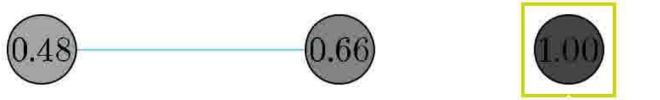


$$\text{Cost} \rightarrow C_0(\dots) = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

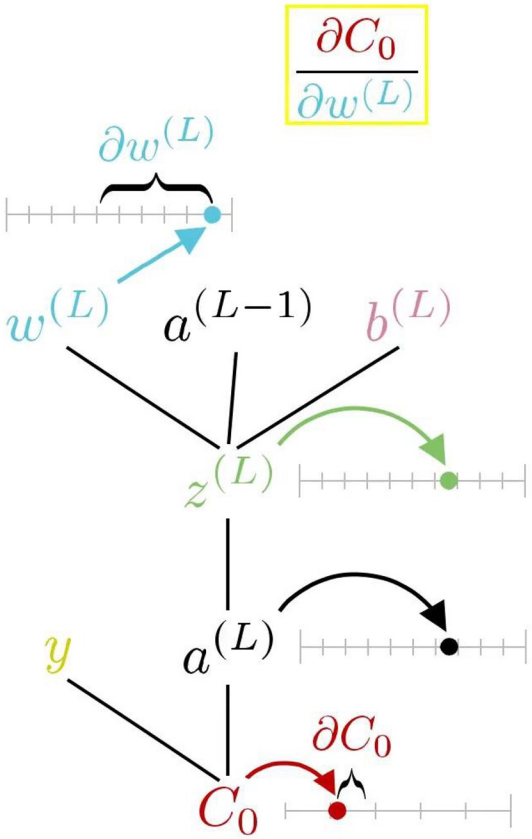
$$a^{(L)} = \sigma(z^{(L)})$$

Desired output



$a^{(L-1)}$ $a^{(L)}$ y

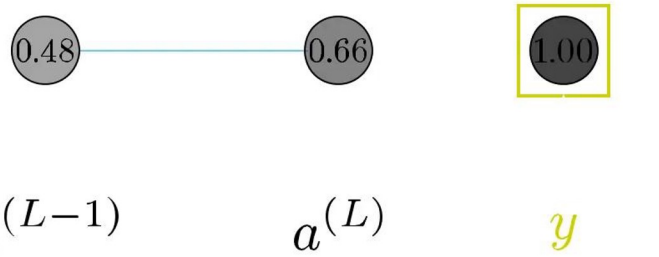
Computing the Gradient with Backpropagation



$$C_0(\dots) = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

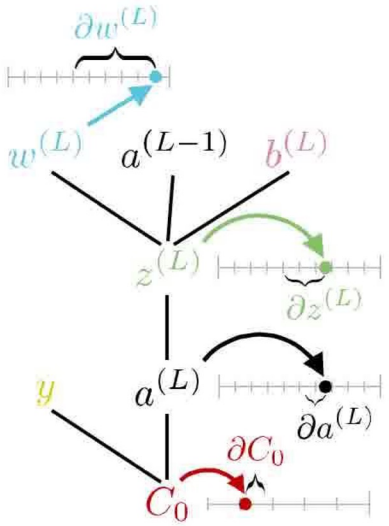
$$a^{(L)} = \sigma(z^{(L)})$$



Computing the Gradient with Backpropagation

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial C0}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}}$$

Chain rule

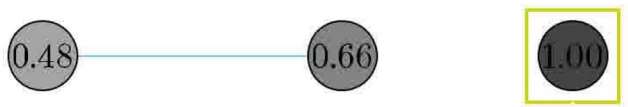


$$C_0(\dots) = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$

Desired output

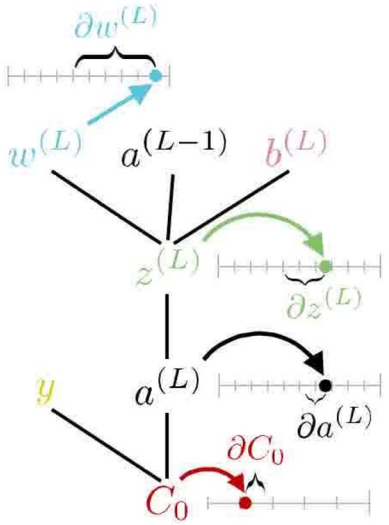


$a^{(L-1)}$ $a^{(L)}$ y

Exercise: What is $\partial C_0/\partial w^{(L-1)}$ using chain rule?

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial C0}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}}$$

Chain rule

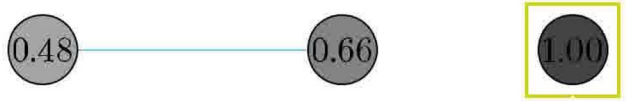


$$C_0(\dots) = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

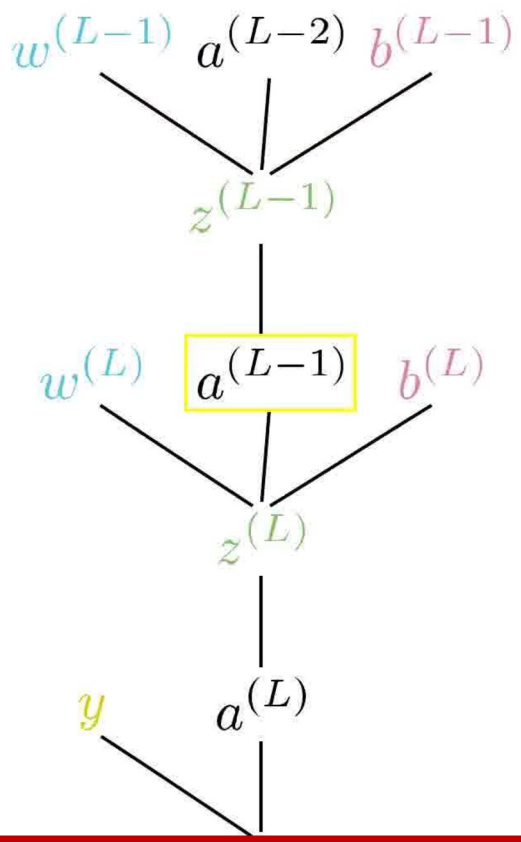
$$a^{(L)} = \sigma(z^{(L)})$$

Desired output



$a^{(L-1)}$ $a^{(L)}$ y

Exercise: What is $\partial C_0/\partial w^{(L-1)}$ using chain rule?

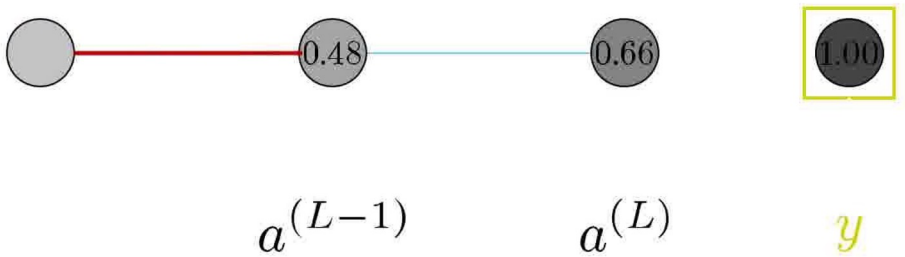


$$\text{Cost} \rightarrow C_0(\dots) = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$

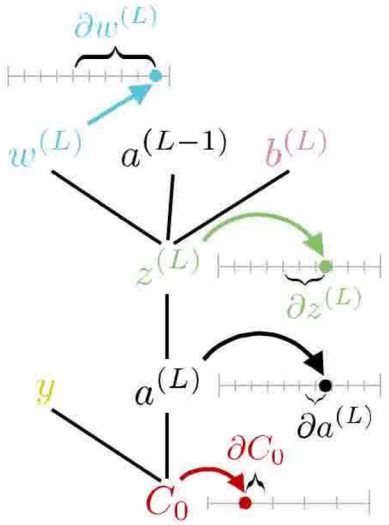
Desired output



What is $\partial C_0/\partial w^{(L)}$ using formulas below?

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial C0}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}}$$

Chain rule

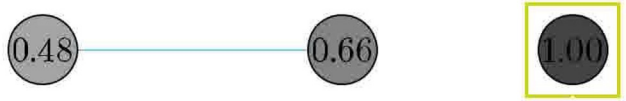


$$C_0(\dots) = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$

Desired output



$a^{(L-1)}$ $a^{(L)}$ y

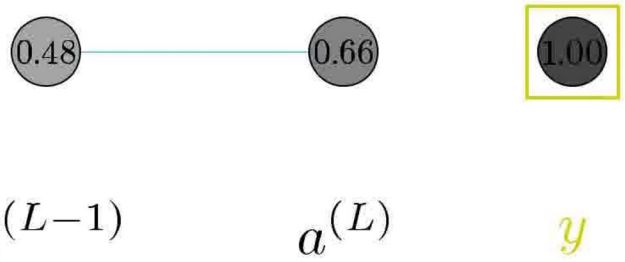
What is $\partial C_0/\partial w^{(L)}$ using formulas below?

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial C0}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

$$C_0 = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

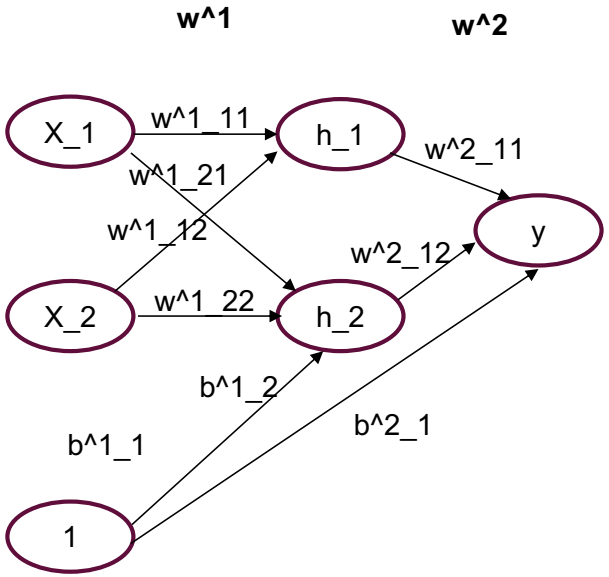
$$a^{(L)} = \sigma(z^{(L)})$$



Putting it all together + bias update

Isn’t this example a bit too simple….?

2-layer network for binary classification



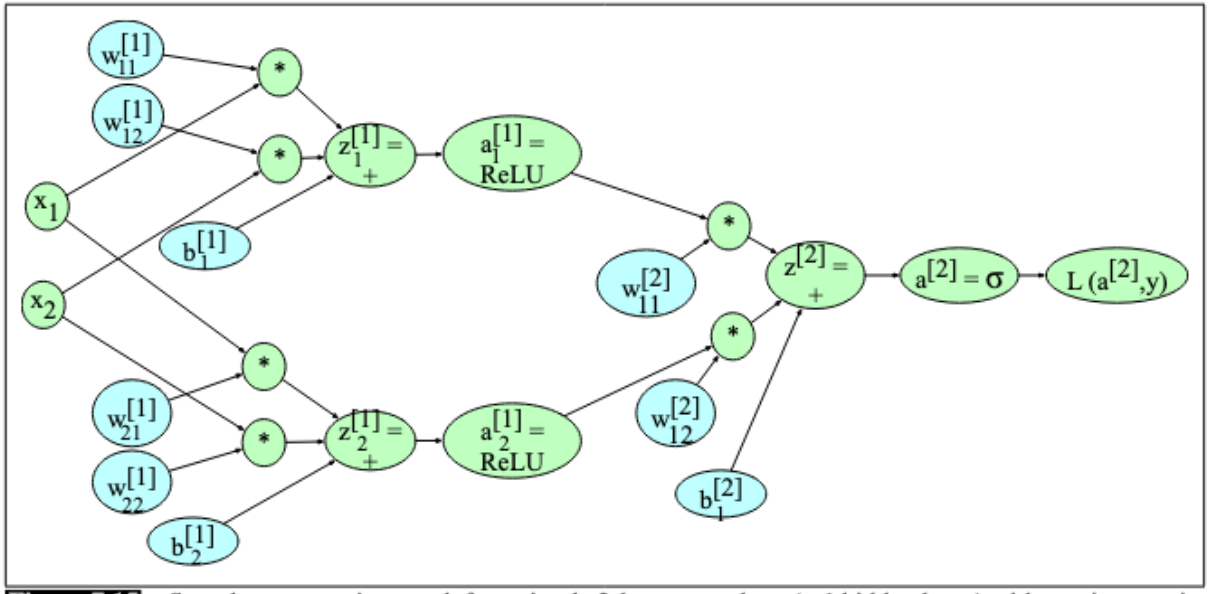
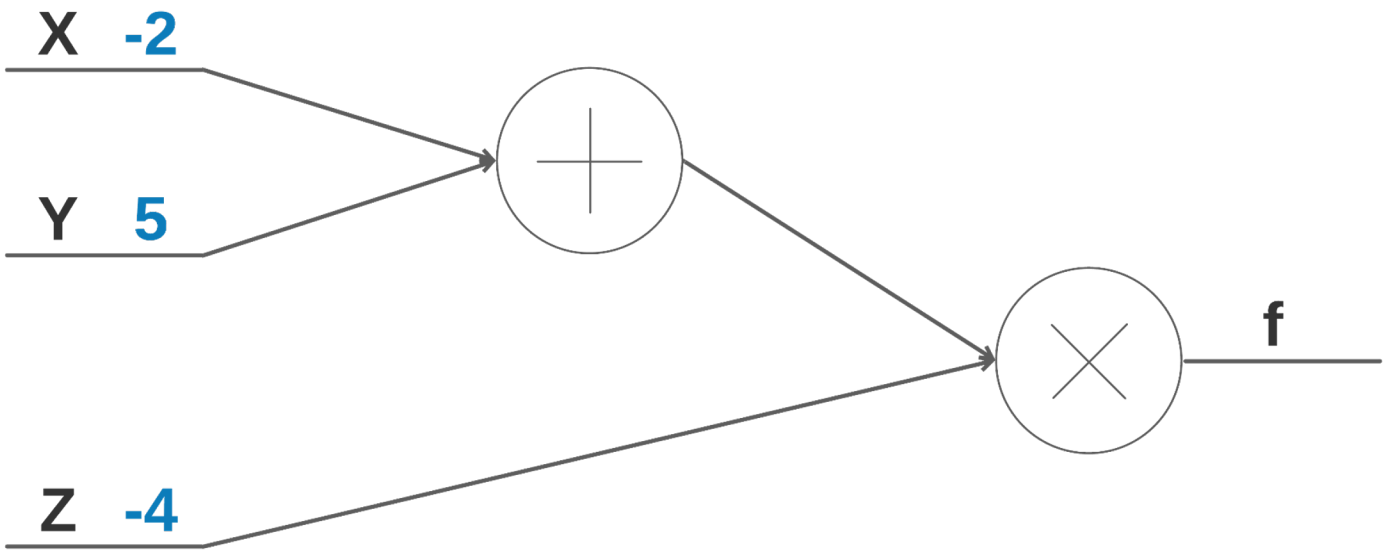


Figure 7.15 Sample computation graph for a simple 2-layer neural net (= 1 hidden layer) with two input units and 2 hidden units. We've adjusted the notation a bit to avoid long equations in the nodes by just mentioning

Exercise: Forward pass and backpropagation



$$f(x, y, z) = (x + y) \cdot z$$

Exercise: Forward pass and backpropagation