AIML231/DATA302 —Techniques in Machine Learning

Week 9 Neural Networks (2)

Backpropagation

Dr Qi Chen

School of Engineering and Computer Science

Victoria University of Wellington

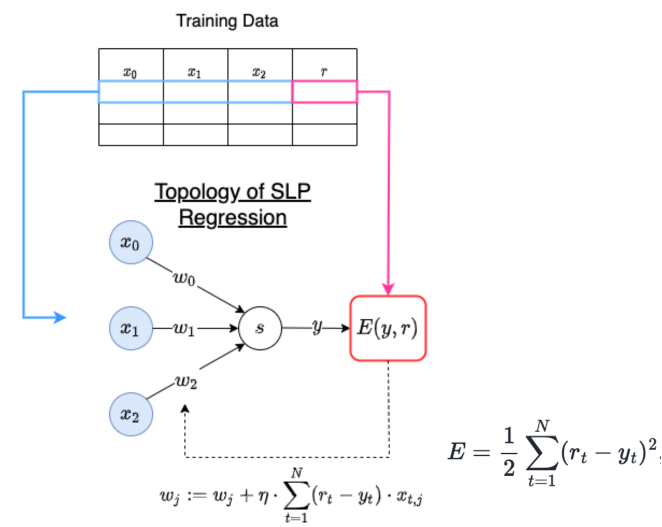
Qi.Chen@vuw.ac.nz

Outline

- Walkthrough of Backpropagation
  - -Recap of Gradient Descent and Gradient
  - -What is Backpropagation
  - -How does Backpropagation work
  - -Derivatives and the Chain Rule

Gradient Descent

- GD is an optimisation algorithm to update the weights in NN



- the update rule of GD

$$\boldsymbol{W_{i+1} = W_i - \eta \nabla L(W_i)}$$

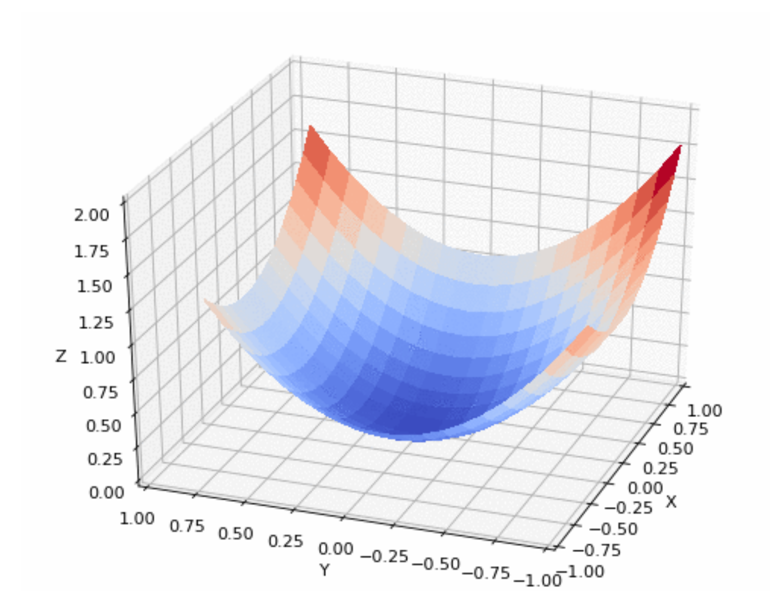
- $\nabla L(W_i)$ is the gradient of the loss function at the current parameters $\nabla L(W_i) = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \dots, \frac{\partial L}{\partial w_p}\right]$, partial derivatives

How sensitive the loss function is to the weight w_p or

how much the loss will be reduced by changing the weight

for example: $\nabla L(W_i) = [2.1, \dots, 0.1, \ \dots]$

w1 w2



given the same change on w1 and w2,
the change cause by w1 to loss function will be 21 times greater than that of w2

Backpropagation algorithm

- Central algorithm in network learning

How …

- Let η be the learning rate
- Set all weights to smaller random values
- Until total error is small enough, repeat
  - For each input example
    - Feed forward pass to get predicted outputs
    - Compute $\beta_z = d_z - o_z$ for each output node
    - Compute $\beta_j = \sum_k w_{j\to k} o_k (1 - o_k) \beta_k$
    - Compute the weight changes $\Delta w_{i\to j} = \eta o_i o_j \left(1 - o_j\right) \beta_j$
  - Add up weight changes for all input examples
  - Change weights according to the update rule of GD

Backpropagation

Simplify Notes

- Simplify notation: let $x_0 = 1$, $b = w_0 = w_0 x_0$

$$y = \sigma(W \cdot X + b) = \sigma\left(w_1 x_1 + w_2 x_2 + \cdots + w_p x_p + b\right)$$

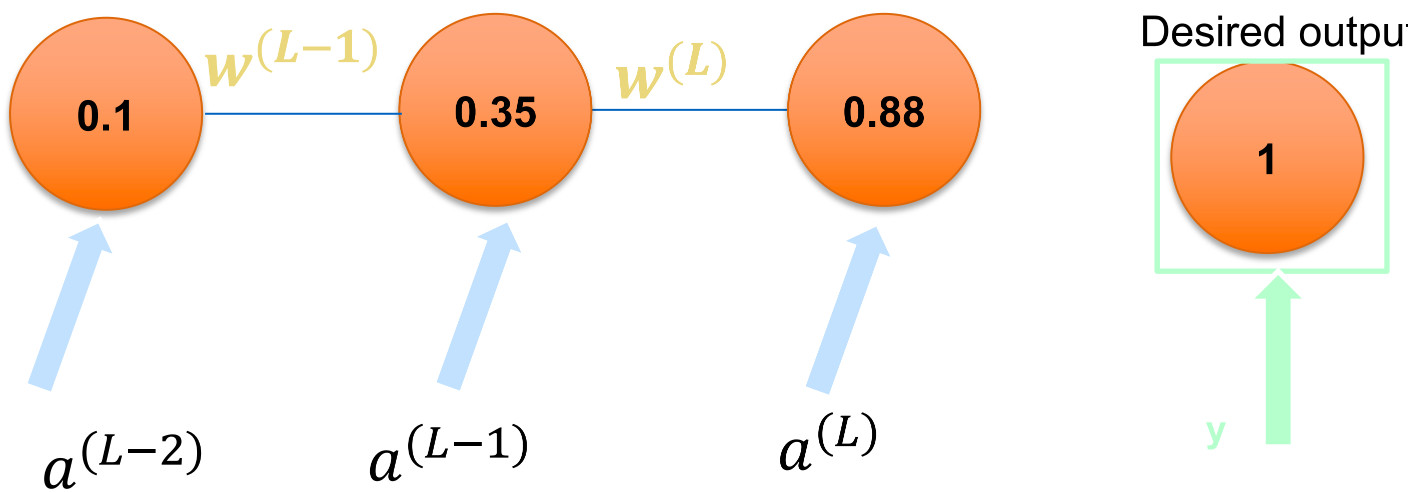
$$\Downarrow$$

$$y = \sigma(W \cdot X + b) = \sigma\left(w_0 x_0 + w_1 x_1 + \cdots + w_p x_p\right)$$

- So we have one block of code for changing all the "weights", rather than changing weights and biases separately

Backpropagation

- How to calculate contribution of W to the loss function C?



A simple NN

$$C_i = \left(a^{(L)} - y\right)^2$$

In this example $=(0.88 - 1)^2$

$a^{(L)} = \sigma\left(w^{(L)} a^{(L-1)}\right)$

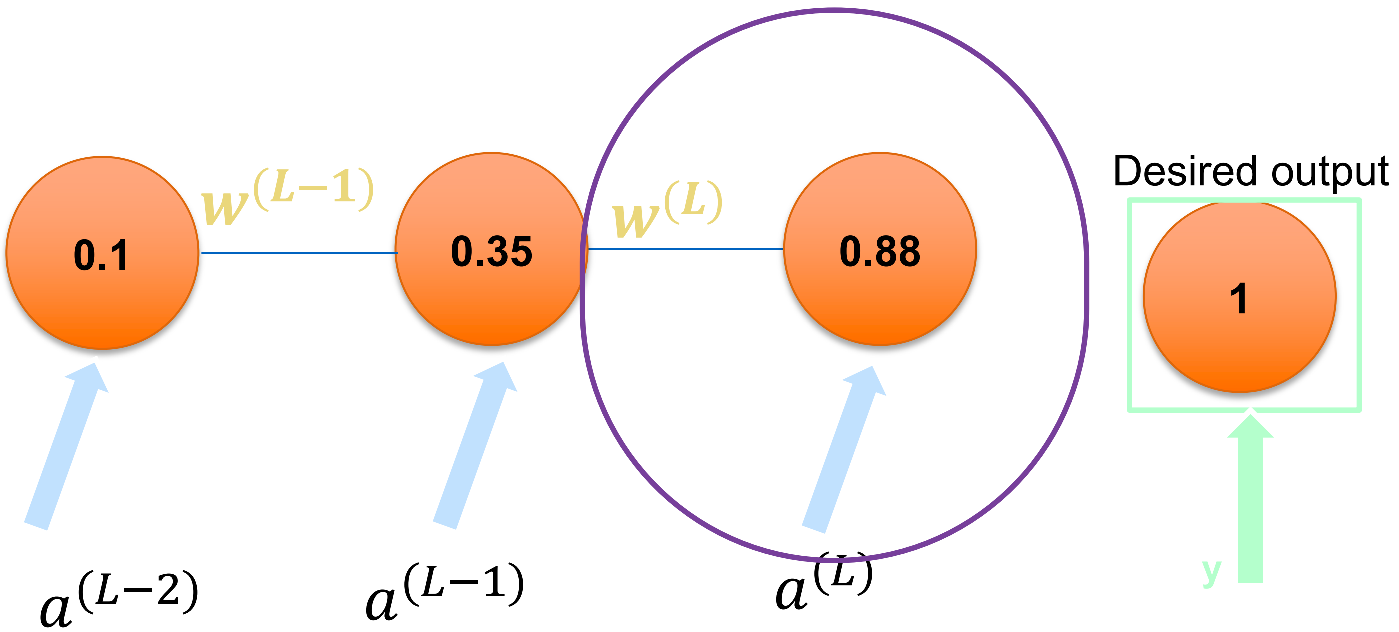
give $z^{(L)} = w^{(L)} a^{(L-1)}$

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

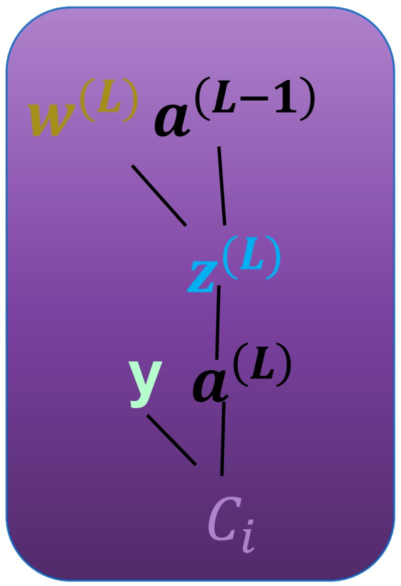
Where σ is the activation function

Backpropagation-Output layer first

- How sensitive the loss C_i is to small changes (e.g., 0.001) in the weight $w^{(L)}$, i.e., the derivative $\frac{\partial C_i}{\partial w^{(L)}}$

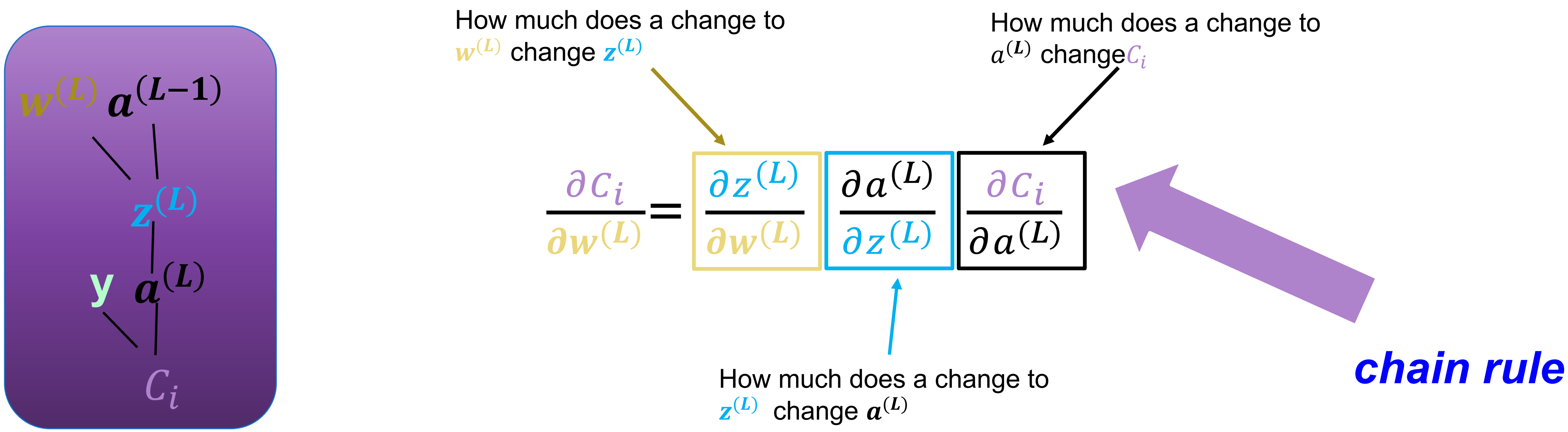


A simple NN



- this tiny change to $w^{(L)}$ causes some change to $z^{(L)}$, (as $z^{(L)} = w^{(L)} a^{(L-1)}$)
- which in turn causes some change to $a^{(L)}$, ($a^{(L)} = \sigma(z^{(L)})$)
- which directly influences the loss C_i ($C_i = (a^{(L)} - y)^2$)

Chain Rule



- the chain rule states how to compute the derivative of a composite function
- the chain rule allows to efficiently compute how small changes in the weight of one layer affect the loss function, by breaking down the computation into smaller, more manageable steps

The Constituent Derivative

- break down $\frac{\partial C_i}{\partial w^{(L)}}$ into separate derivatives according to chain rule

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

- now just need to compute the values of the three individual derivatives
- To compute each derivative, we'll use some relevant formula from the way we've defined our neural network

$$z^{(L)} = w^{(L)} a^{(L-1)} \rightarrow \frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$

$$a^{(L)} = \sigma(z^{(L)}) \rightarrow \frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

$$C_i = (a^{(L)} - y)^2 \rightarrow \frac{\partial C_i}{\partial a^{(L)}} = 2(a^{(L)} - y)$$

Rules	Function	Derivative
Multiplication by constant	cf	cf′
Power Rule	x^n	nx^{n-1}
Sum Rule	f + g	f′ + g′
Difference Rule	f - g	f′ − g′
Product Rule	fg	f g′ + f′ g
Quotient Rule	f/g	$\frac{f'g - g'f}{g^2}$
Reciprocal Rule	1/f	$-f'/f^2$

- It is straightforward once you know which equation to start from

Putting it all together

- Putting constituent derivatives together

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

$$= a^{(L-1)} \sigma'(z^{(L)})\ 2(a^{(L)} - y)$$

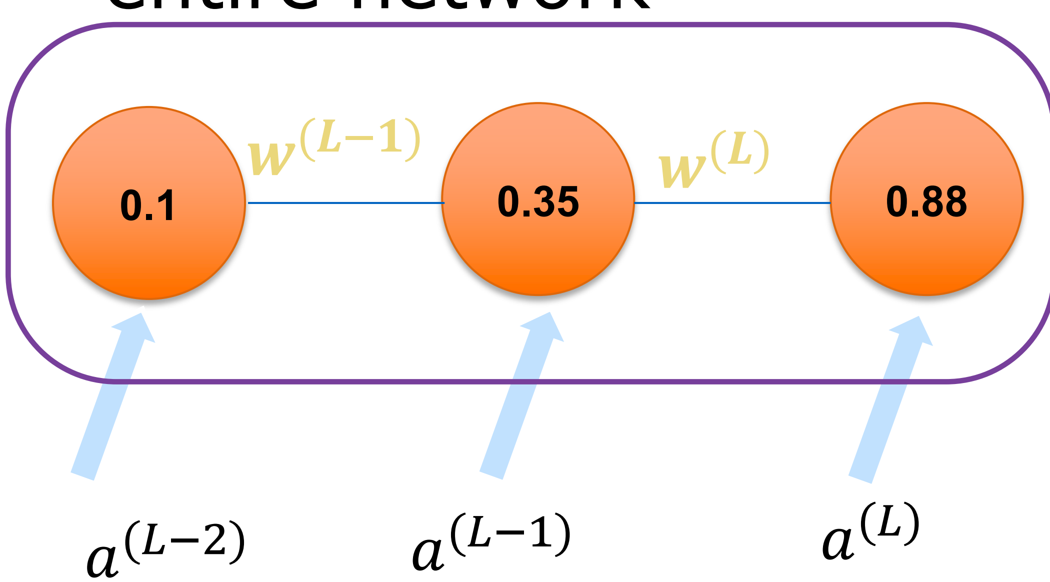
- This formula tells us how a change to that *one particular weight* in the last layer will affect the loss for that *one particular training example*

More …

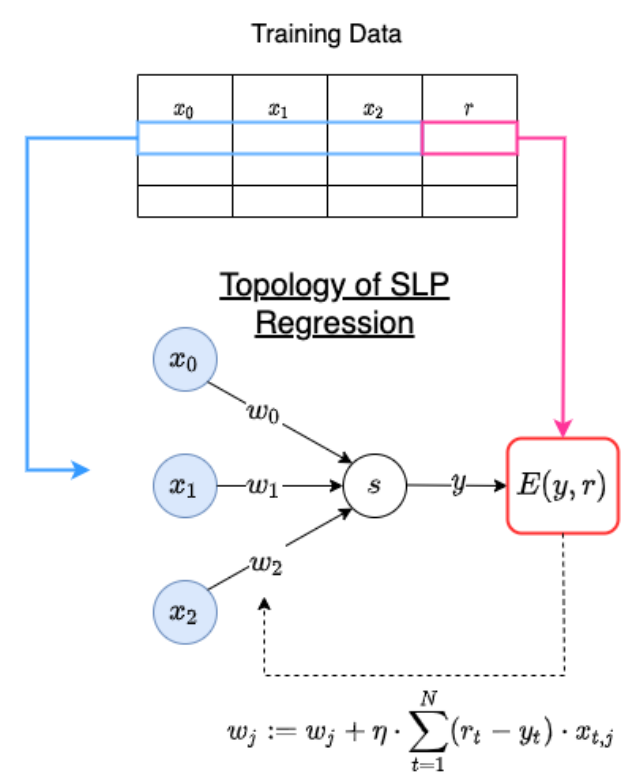
- The full loss function for the network is the average all the individual lost for each training $C = \frac{1}{n}\sum_{i=0}^{n-1} C_i$
- To get the derivative of C with respect to the weight

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{n}\sum_{i=0}^{n-1} \frac{\partial C_i}{\partial w^{(L)}}$$

- Also, to compute the full gradient, we will also need all the other derivatives with respect to all the other weights in the entire network

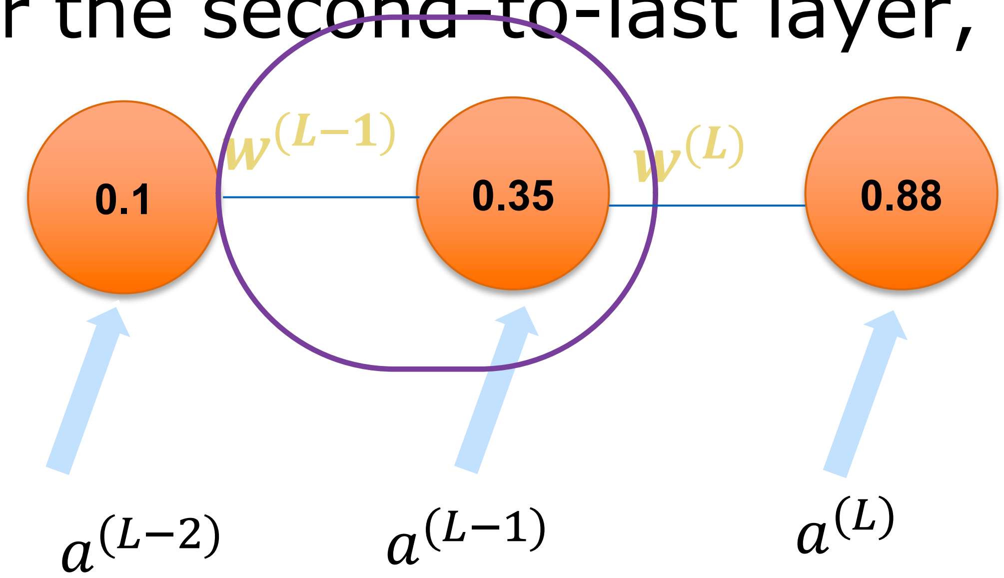


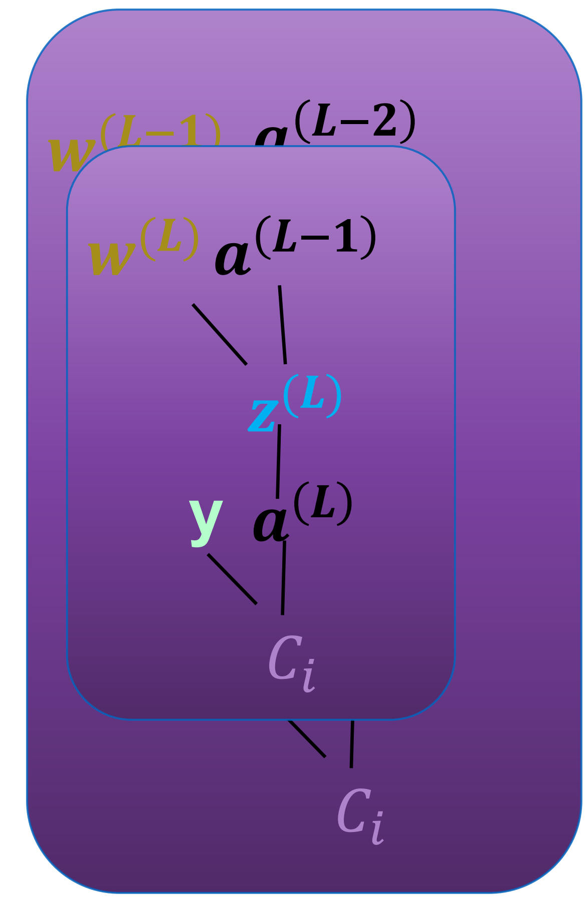
$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial w^{(2)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \end{bmatrix}$$



Previous Layers' Weights

- For other weights lie in earlier layers of the network
  - For the second-to-last layer,





  - consider $\frac{\partial C_i}{\partial a^{(L-1)}}$ first

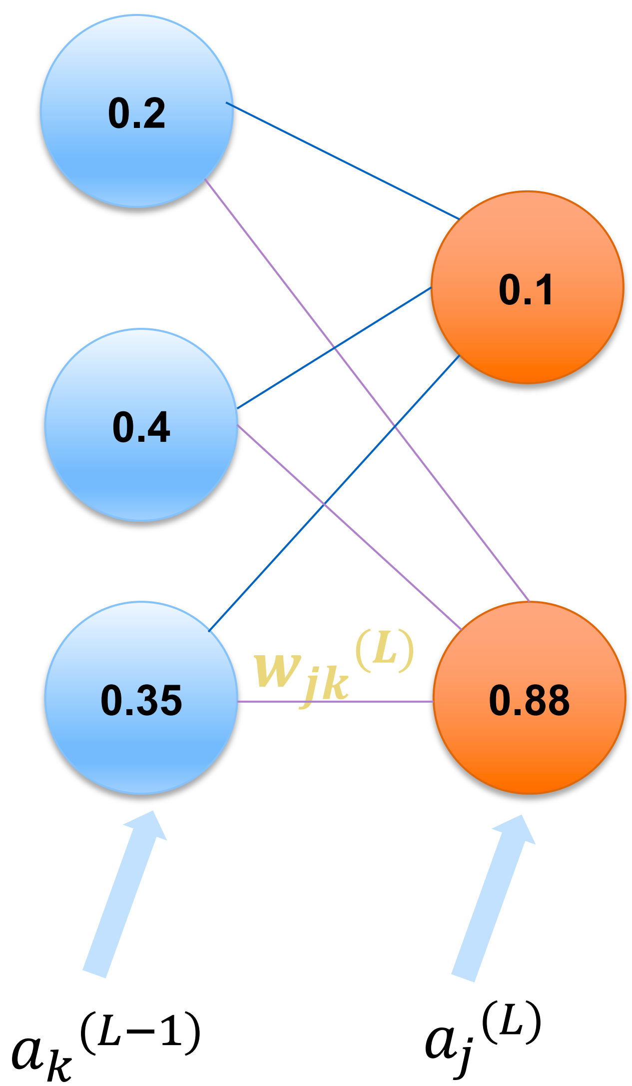
$$\frac{\partial C_i}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_i}{\partial a^{(L)}}$$

  - the derivative of the loss with respect to $w^{(L-1)}$ looks very similar with that of $w^{(L)}$

$$\frac{\partial C_i}{\partial w^{(L-1)}} = \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_i}{\partial a^{(L)}} = \frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial C_i}{\partial a^{(L-1)}}$$

More Complicated Networks

- Each layer has more than one neuron
- Loss of the NN $C_i = \sum\left(a_j^{(L)} - y_j\right)^2$



$$\frac{\partial C_i}{\partial w_{jk}^{(L)}} \quad ?$$

$$z_j^{(L)} = w_{j0}^{(L)} a_0^{(L-1)} + w_{j1}^{(L)} a_1^{(L-1)} + w_{j2}^{(L)} a_2^{(L-1)}$$

$$\frac{\partial C_i}{\partial w_{jk}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_i}{\partial a_j^{(L)}}$$

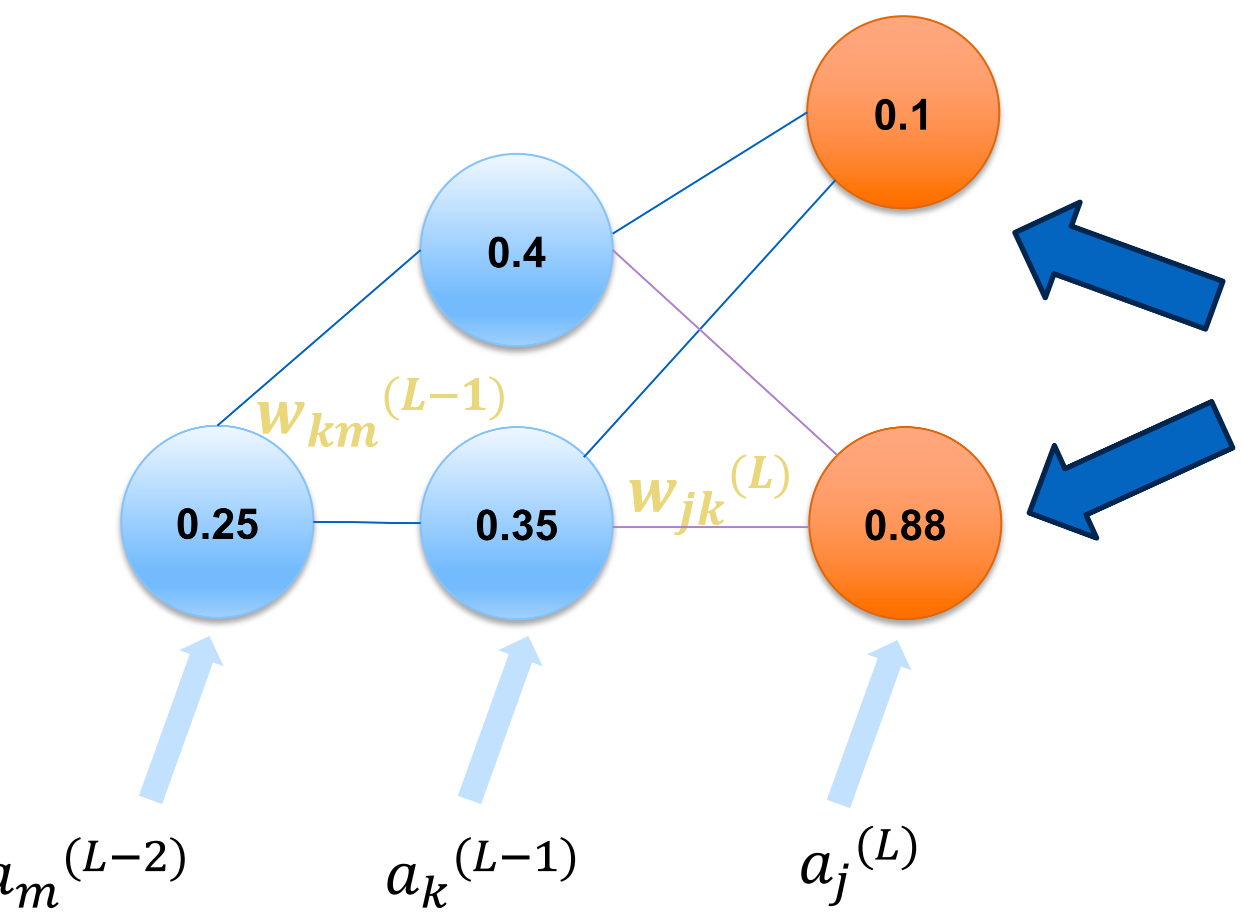
Essentially identical to what we have before,
Only difference is we keep track of more indices, *j* and *k*

More Complicated Networks

- How about $\frac{\partial C_i}{\partial w_{km}{}^{(L-1)}}$?

$$\frac{\partial C_i}{\partial w_{km}{}^{(L-1)}} = \frac{\partial z_k{}^{(L-1)}}{\partial w_{km}{}^{(L)}} \frac{\partial a_k{}^{(L-1)}}{\partial z_k{}^{(L-1)}} \frac{\partial C_i}{\boldsymbol{\partial a_k{}^{(L-1)}}}$$

$$\frac{\partial C_i}{\partial w_{km}{}^{(L-1)}} = a_m{}^{(L-2)}\ \sigma'(z_j{}^{(L-1)}) \sum_{j=0}^{n_L-1} w_{jk}{}^{(L)} \sigma'(z_j{}^{(L)}) \frac{\partial C_i}{\partial a_j{}^{(L)}}$$



$a_k{}^{(L-1)}$ influence the loss function through multiple paths

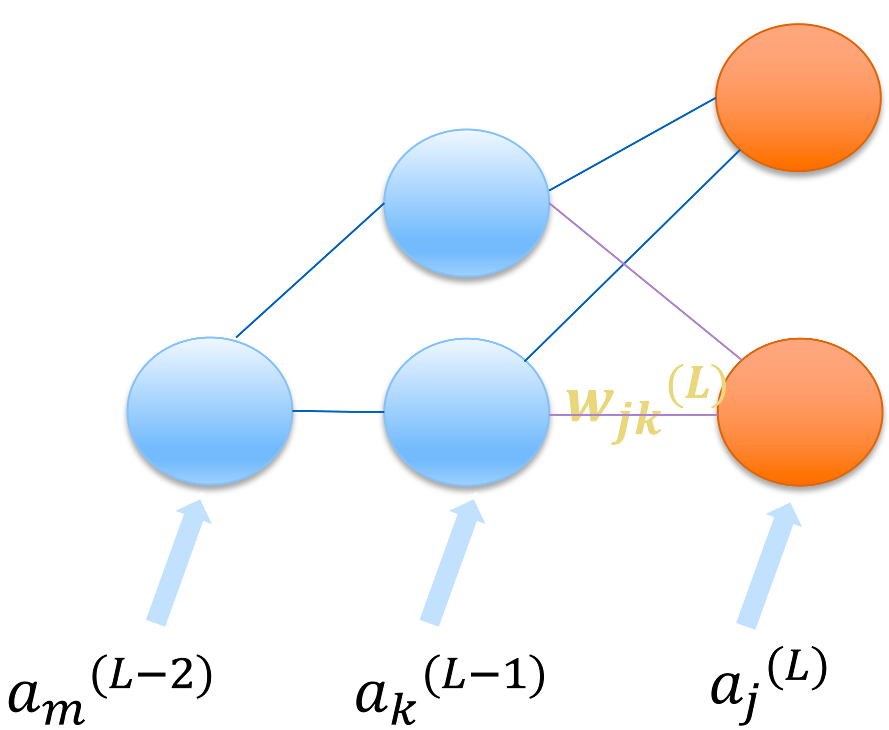
$$\frac{\partial C_i}{\boldsymbol{\partial a_k{}^{(L-1)}}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j{}^{(L)}}{\boldsymbol{\partial a_k{}^{(L-1)}}} \frac{\partial a_j{}^{(L)}}{\partial z_j{}^{(L)}} \frac{\partial C_i}{\partial a_j{}^{(L)}}$$

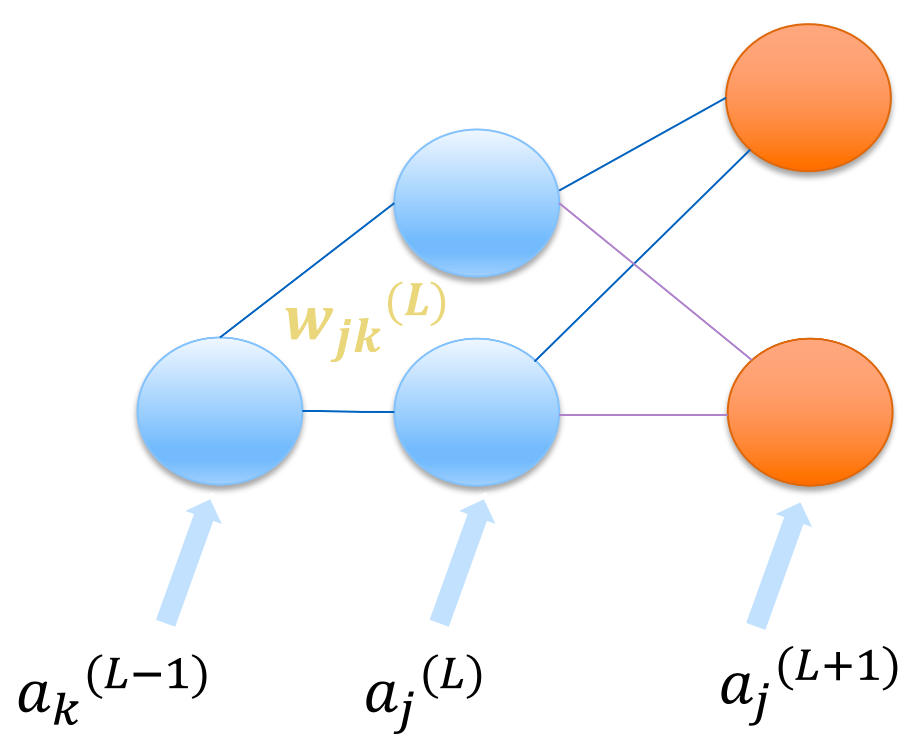
$$= \sum_{j=0}^{n_L-1} w_{jk}{}^{(L)} \sigma'(z_j{}^{(L)}) \frac{\partial C_i}{\partial a_j{}^{(L)}}$$

Partial Derivatives

Varied for different loss functions

$$\frac{\partial C_i}{\partial w_{jk}^{(L)}} = \begin{cases} a_k^{(L-1)} \sigma'(z_j^{(L)}) 2(a_j^{(L)} - y_i), & \boldsymbol{w_{jk}\ for\ the\ last\ layer} \\ a_k^{(L-1)} \sigma'(z_j^{(L)}) \sum_{j=0}^{n_{L+1}-1} w_{jk}^{(L+1)} \sigma'(z_j^{(L+1)}) \frac{\partial C_i}{\partial a_j^{(L+1)}}, & \boldsymbol{others} \end{cases}$$





Again, more …

- To get the derivative of L with respect to the weight, take the average over all training data

$$\frac{\partial L}{\partial w^{(L)}} = \frac{1}{n}\sum_{i=0}^{n-1} \frac{\partial L_i}{\partial w^{(L)}}$$

- also need all the other derivatives with respect to all the other weights in the entire network

$$\nabla L(W_i) = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \dots, \frac{\partial L}{\partial w_p}\right]$$

- Then update weights **with** $\boldsymbol{W_{i+1} = W_i - \eta \nabla L(W_i)}$

Summary

- Backpropagation is a fundamental algorithm in neural network training, used to train NNs by adjusting the weights of connections
- Gradient descent is an optimization technique that updates the weights by moving in the direction of the steepest descent of the loss function
- Derivatives and gradients give us a concrete way to find a minimum loss
- The chain rule - decompose a complicated network of influences to understand how sensitive that cost function is to each and every weight and bias