# CS 466/566
# Introduction to Deep Learning

Lecture 8 – Back Propagation

Compiled from various resources

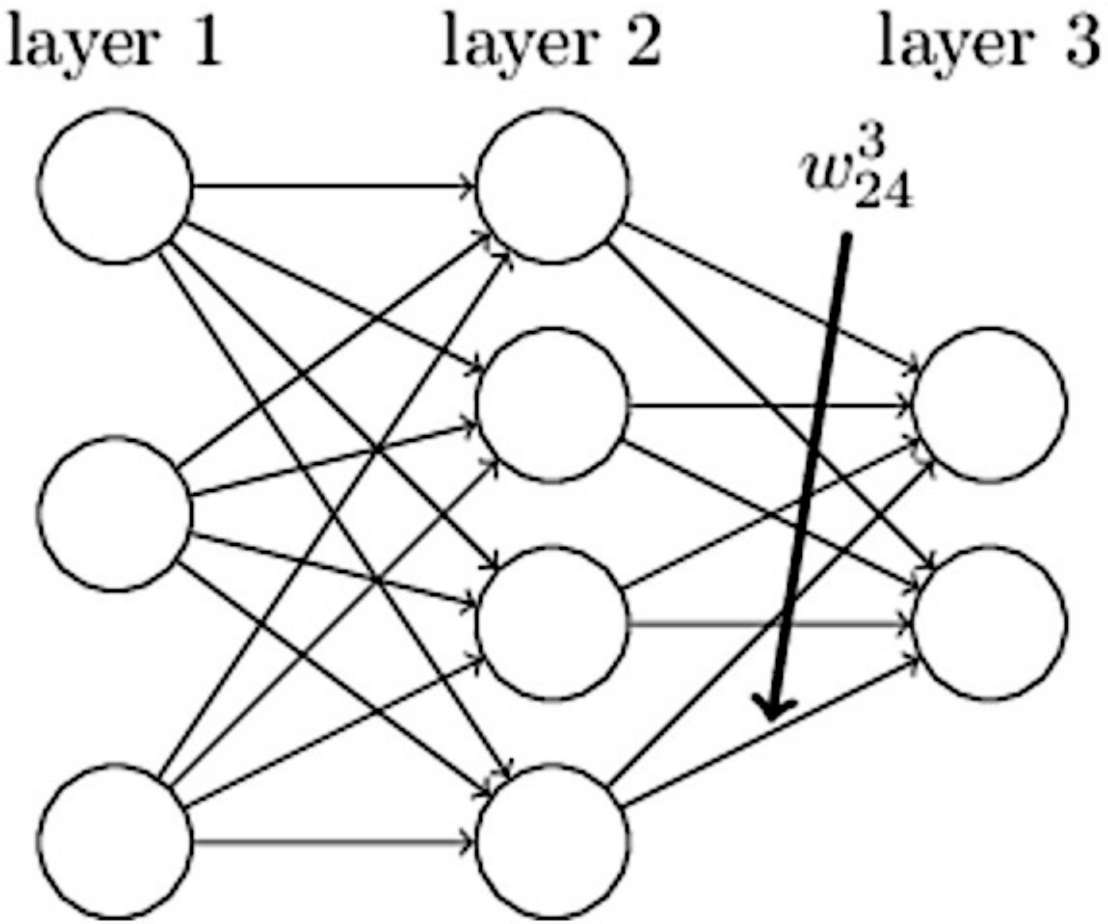
# Neural Network Notation

- For each neuron $\boldsymbol{j}$, its output $\boldsymbol{a_j}$ is defined as

$$a_j = \sigma(z_j) = \sigma(\sum_{k=1}^{n} w_{jk} a_k)$$

- where
  - $\boldsymbol{z_j}$ is the input to a neuron: weighted sum of outputs of previous neurons
  - $\boldsymbol{a_j}$ is the activated value of $\boldsymbol{z_j}$.
  - $\boldsymbol{n}$ is the number of input units to the neuron.
  - $\boldsymbol{w_{ij}}$ denotes the weight between neuron $\boldsymbol{i}$ and neuron $\boldsymbol{j}$.

# Neural Network Notation




$w^l_{jk}$ is the weight from the $k^{th}$ neuron in the $(l-1)^{th}$ layer to the $j^{th}$ neuron in the $l^{th}$ layer

# A naïve approach for updating weights

- Imagine that back-propagation hasn't been derived yet.
- You want to use gradient descent for learning.
- You need a way of computing the gradient of the cost function.
- You think back to your knowledge of calculus, and decide to see if you can use the chain rule to compute the gradient.
- But after playing around a bit, the algebra looks complicated, and you get discouraged.
- You decide to regard the cost as a function of the weights $C = C(w)$ alone.

# A naïve approach for updating weights

- An obvious way of doing that is to use the numerical approximation of the derivative at a specific weight.

$$\frac{\partial C}{\partial w_j} \approx \frac{C(w + \epsilon e_j)}{\epsilon}$$

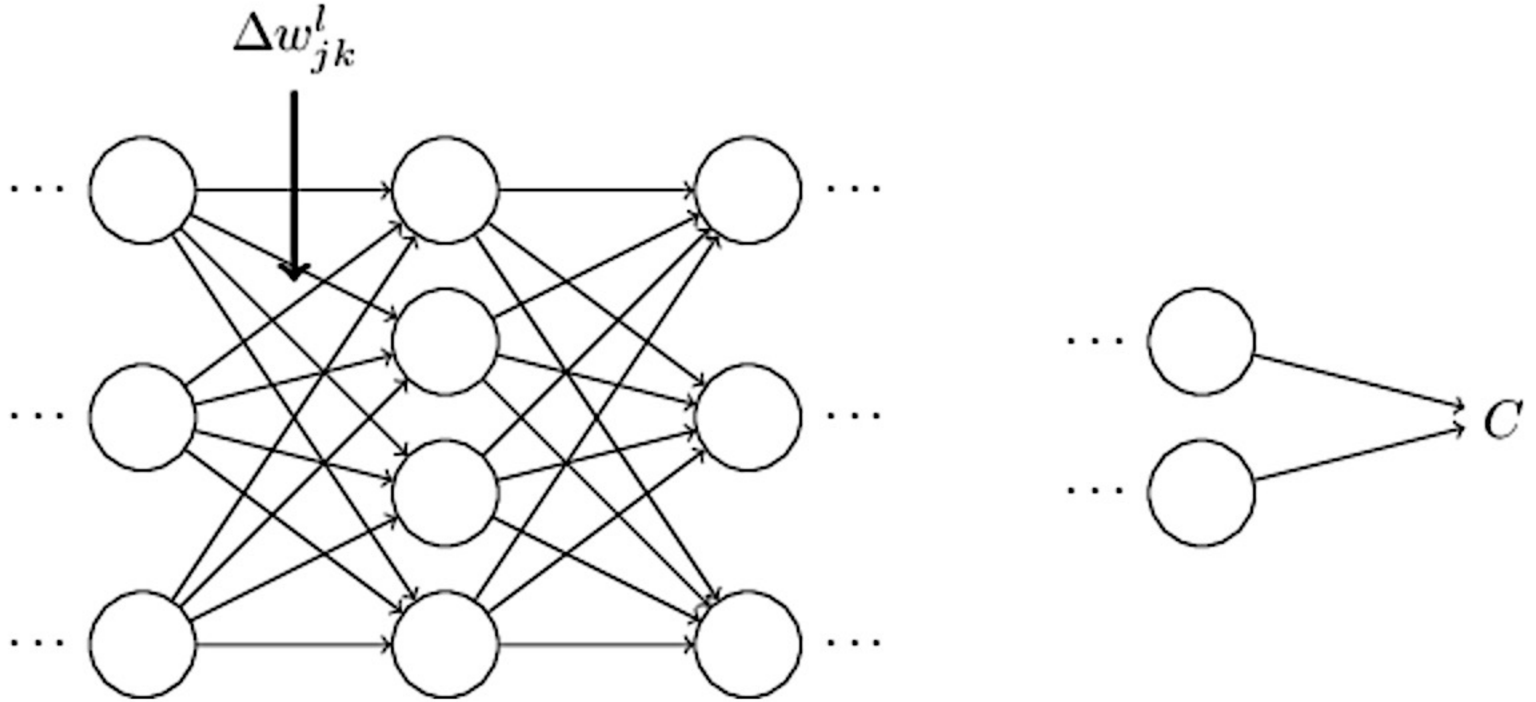
- where $\boldsymbol{\epsilon} > \mathbf{0}$ is a small positive number, and $\boldsymbol{e_j}$ is the unit vector in the $\boldsymbol{j^{th}}$ direction.
- This approach looks very promising.
- It's simple conceptually, and extremely easy to implement, using just a few lines of code.
- Certainly, it looks much more promising than the idea of using the chain rule to compute the gradient!

# A naïve approach for updating weights

- Unfortunately, while this approach appears promising, when you implement the code it turns out to be extremely slow.
- To understand why, imagine we have a **million weights** in our network.
- Then for each distinct weight $w_j$ we need to compute $C(w+\epsilon e_j)$ in order to compute its partial effect on ultimate cost.
- That means that to compute the gradient we need to compute the cost function a million different times, requiring **a million forward passes** through the network (**per training example**)!
- We need a more efficient strategy.

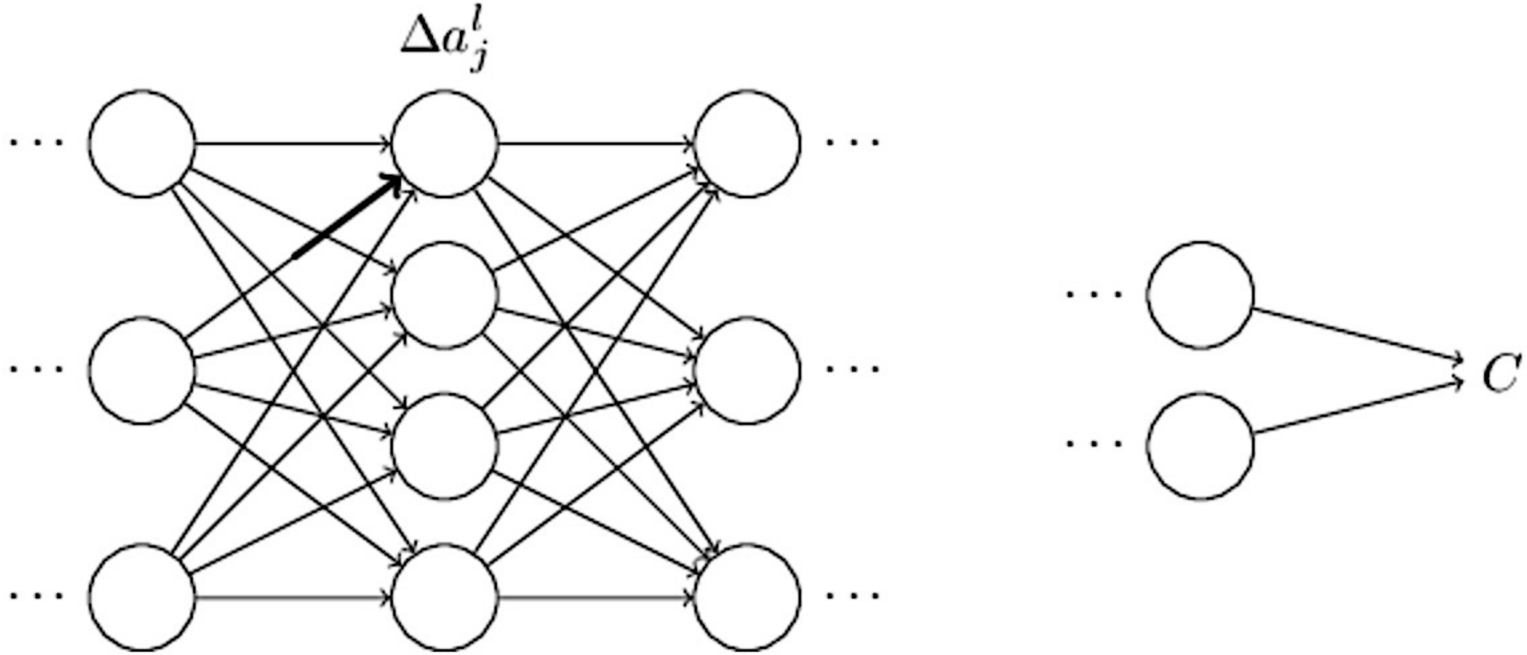
# Intuition before derivation of back propagation

- To improve our intuition about what the algorithm is doing, let's imagine that we've made a small change to some weight in the network:



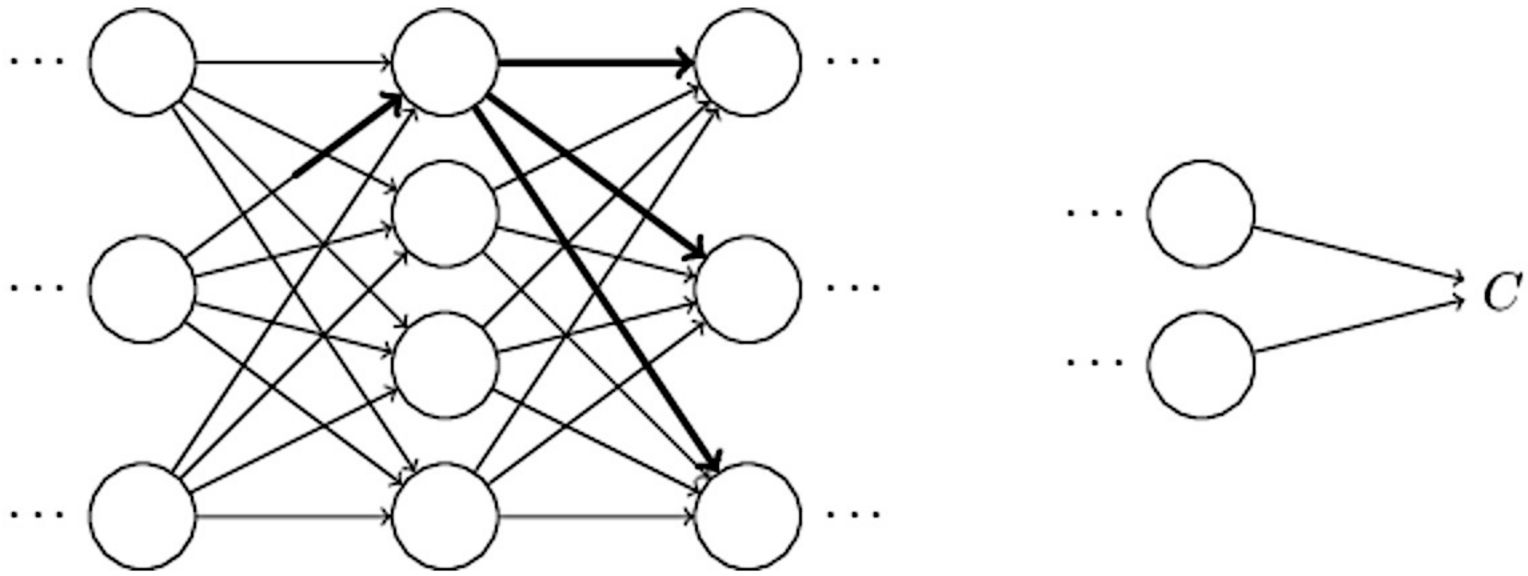
# Intuition before derivation of back propagation

- That change in weight will cause a change in the output activation from the corresponding neuron:



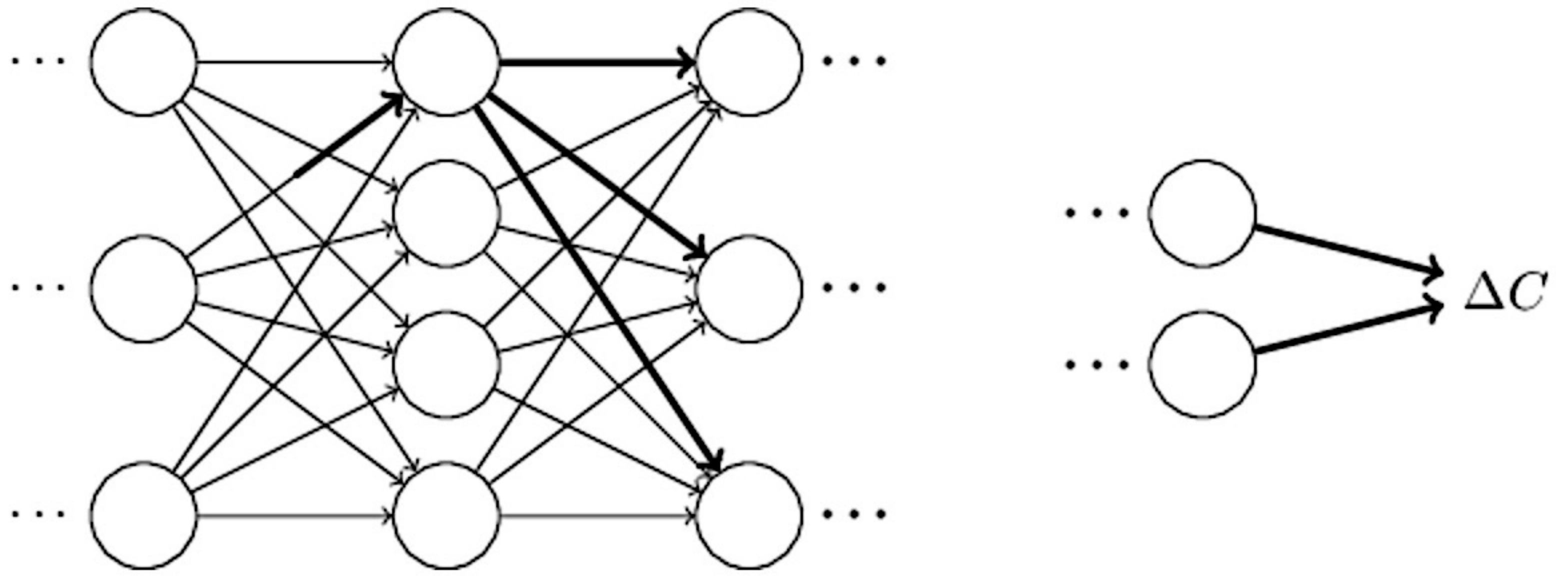
# Intuition before derivation of back propagation

- That, in turn, will cause a change in ***all*** the activations in the next layer:



# Intuition before derivation of back propagation

- Those changes will in turn cause changes in the next layer, and then the next, and so on all the way through to causing a change in the final layer, and then in the cost function:



# Intuition before derivation of back propagation

- The change $\boldsymbol{\Delta C}$ in the cost is related to the change in the weight by the equation:

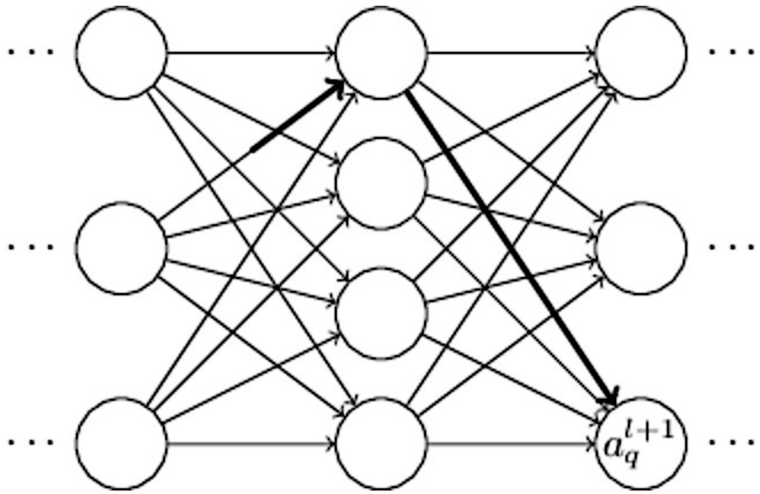
$$\Delta C \approx \frac{\partial C}{\partial w^l_{jk}} \Delta w^l_{jk}$$

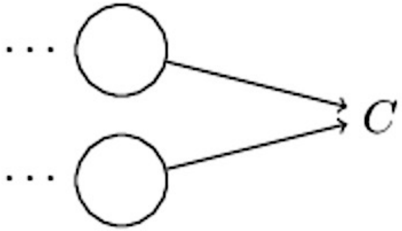
- Let's try to carry this out. The change in $\boldsymbol{w}$ causes a small change in the activation of the $\boldsymbol{j^{th}}$ neuron in the $\boldsymbol{l^{th}}$ layer. This change is given by:

$$\boxed{\Delta a^l_j \approx \frac{\partial a^l_j}{\partial w^l_{jk}} \Delta w^l_{jk}}$$

# Intuition before derivation of back propagation

- The change in activation will cause changes in ***all*** the activations in the next layer, i.e., the *(l+1)^th^* layer.
- We'll concentrate on the way **just a single one of those activations** is affected




$$\Delta a_j^l \approx \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

- in fact,
  it'll cause a change
  like this:

$$\Delta a_q^{l+1} \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \Delta a_j^l$$

# Intuition before derivation of back propagation

- We have:

$$\boxed{\Delta a_j^l} \approx \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

$$\Delta a_q^{l+1} \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \boxed{\Delta a_j^l}$$

- Substituting

$$\boxed{\Delta a_q^{l+1} \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l}$$

# Intuition before derivation of back propagation

- This change will, in turn, cause changes in the activations in the next layer.
- In fact, we can imagine a path all the way through the network from ***our initial weight*** to $\boldsymbol{C}$, with each change in activation causing a change in the next activation, and, finally, a change in the **cost** at the output.
- If the path goes through some activations then the resulting expression can be:

$$\Delta C \approx \frac{\partial C}{\partial a_m^L}\frac{\partial a_m^L}{\partial a_n^{L-1}}\frac{\partial a_n^{L-1}}{\partial a_p^{L-2}}\ldots\frac{\partial a_q^{l+1}}{\partial a_j^l}\frac{\partial a_j^l}{\partial w_{jk}^l}\Delta w_{jk}^l$$

- This represents the change in $\boldsymbol{C}$ due to changes in the activations along this particular path through the network.

# Intuition before derivation of back propagation

- Of course, there's many paths by which a change in **our weight** can propagate to affect the cost, and **we've been considering just a single path**.
- To compute the total change in $\boldsymbol{C}$ it is plausible that we should sum over all the possible paths between the weight and the final cost, i.e.,

$$\Delta C \approx \sum_{mnp\ldots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

- where we've summed over all possible choices for the intermediate neurons along the path. But remember how we started all this:

$$\Delta C \approx \frac{\partial C}{\partial w_{jk}^l} \Delta w_{jk}^l$$

# Intuition before derivation of back propagation

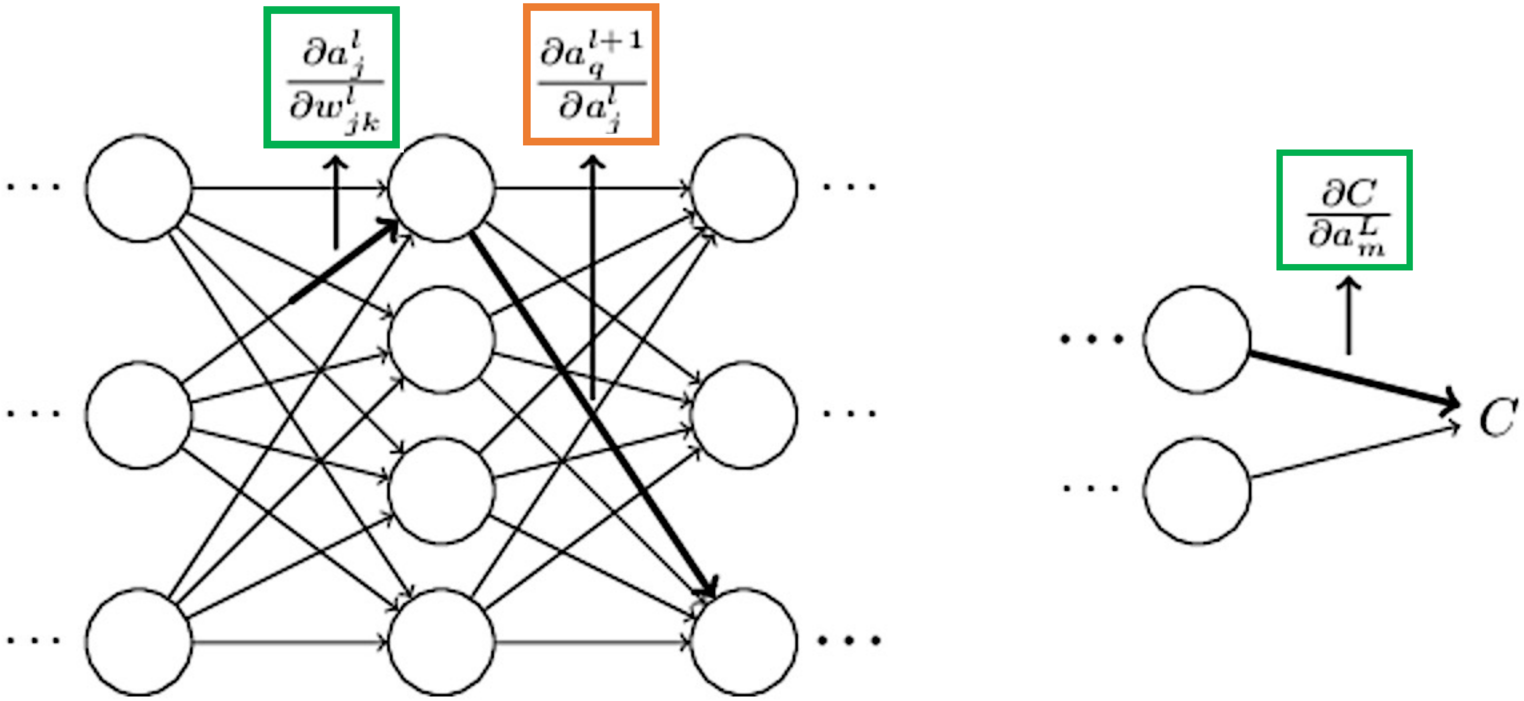
- At last, we have:

$$\frac{\partial C}{\partial w^l_{jk}} = \sum_{mnp\ldots q} \boxed{\frac{\partial C}{\partial a^L_m}} \boxed{\frac{\partial a^L_m}{\partial a^{L-1}_n}} \boxed{\frac{\partial a^{L-1}_n}{\partial a^{L-2}_p}} \cdots \boxed{\frac{\partial a^{l+1}_q}{\partial a^l_j}} \boxed{\frac{\partial a^l_j}{\partial w^l_{jk}}}$$

- This looks complicated! However, it has a nice intuitive interpretation.
- We're computing the rate of change of $C$ with respect to **a weight** in the network.
- What the equation tells us is that *every edge between two neurons in the network is associated with a rate factor which is just the partial derivative of one neuron's activation with respect to the other neuron's activation*.

# Intuition before derivation of back propagation

$$\frac{\partial C}{\partial w^l_{jk}} = \sum_{mnp\ldots q} \frac{\partial C}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} \cdots \frac{\partial a^{l+1}_q}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{jk}}$$



# Back propagation

Now that we have an intuition about the effect of changing a single weight in our network,

Let’s start the real derivation;

starting from the output layer,

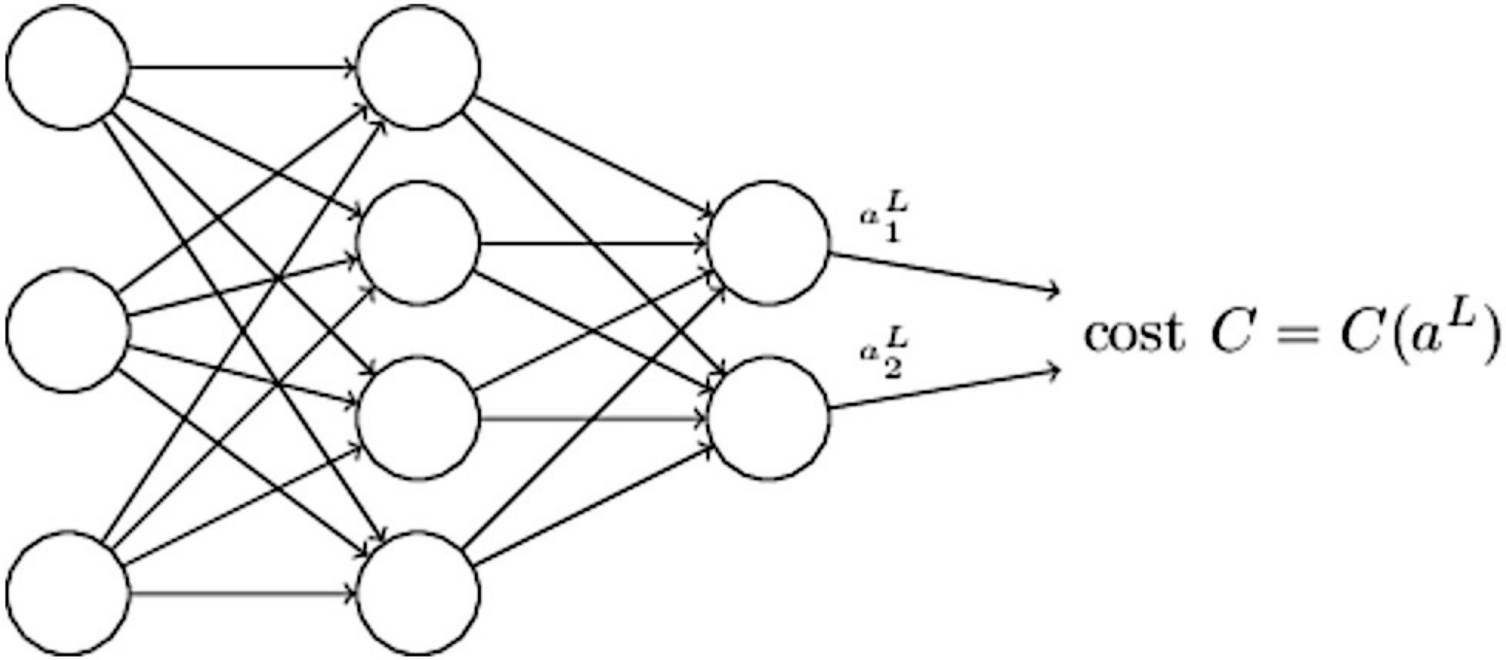
going towards the input layer

# Back propagation: Cost function assumptions First Assumption

- Cost must be written as an average over cost functions for individual training examples.
- The reason for this assumption is that the backpropagation algorithm calculates the gradient of the error function for a single training example, which needs to be generalized to the overall error function.
- In practice, training examples are placed in batches, and the error is averaged at the end of the batch, which is then used to update the weights.

# Back propagation: Cost function assumptions
# Second Assumption

- Cost must be written as a function of the outputs from the neural network.



# Back propagation: Idea

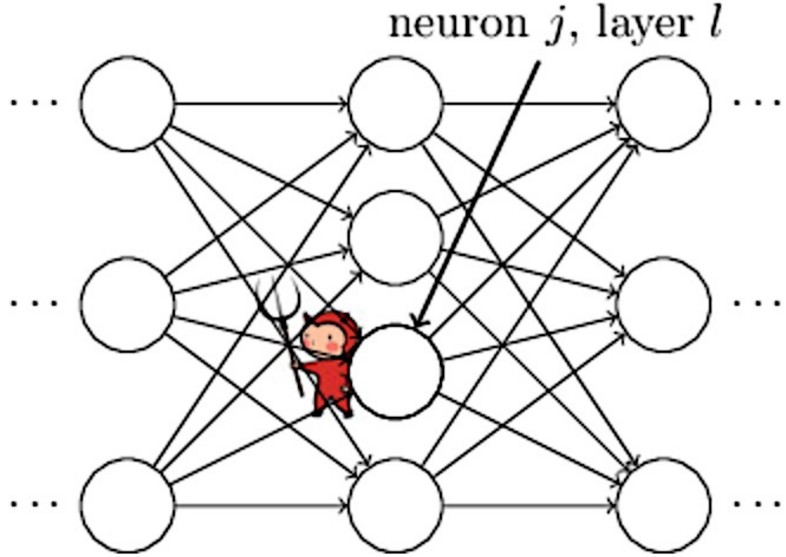
- Backpropagation is about calculating how changing the weights and biases in a network changes the cost function, ***effectively***.
- Ultimately, this will mean computing the partial derivatives.
- But to compute those, we first introduce an intermediate quantity which we call the ***local error*** in the $j^{th}$ neuron in the $l^{th}$ layer.

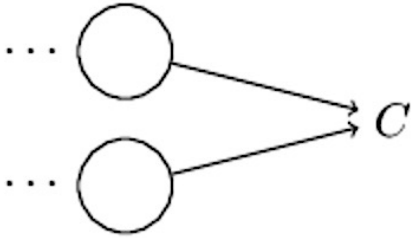
$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}}$$

- Backpropagation will give us a procedure to compute the error and then will relate it to the ultimate cost function.

# Back propagation: Idea

- To understand how the ***local error*** is defined, imagine there is a **demon** in our neural network:




$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}}$$

$$\sigma(z_j^l + \Delta z_j^l)$$

$$\frac{\partial C}{\partial z_j^{(l)}} \Delta z_j^l$$

- The **demon** sits at the $j^{th}$ neuron in layer $l$. As the input to the neuron comes in, the demon messes with the neuron's operation. It adds a little change to the neuron's weighted input $z_j$.
- This change propagates through later layers in the network, finally causing the overall cost to change by an amount.

# Back propagation: Regression with squared cost

- Since backpropagation uses the gradient descent method, one needs to calculate the derivative of the squared cost function with respect to the weights of the network.
- Assuming one output neuron, the squared cost function is:

$$C = \frac{1}{2}(t - y)^2$$

- where
  - $\boldsymbol{C}$ is the squared cost,
  - $\boldsymbol{t}$ is the target output for a training sample
  - $\boldsymbol{y}$ is the actual output of the output neuron.

# Back propagation

- The activation function $\sigma$ is in general non-linear and differentiable. A commonly used activation function is the logistic function:

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

- which has a nice derivative of:

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

**Recall: Derivative of division**

$$\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}$$

# Back propagation

- Finding the derivative of the cost:
  - Calculating the partial derivative of the cost with respect to a weight $\boldsymbol{w_{ij}}$ is done using the chain rule twice:

$$\frac{\partial C}{\partial w_{ij}} = \boxed{\frac{\partial C}{\partial a_i}} \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial w_{ij}}$$

  - $\boldsymbol{a_i}$ can be either at the output layer, or in an arbitrary inner layer of the network.

# Back propagation

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_i}\frac{\partial a_i}{\partial z_i}\boxed{\frac{\partial z_i}{\partial w_{ij}}}$$

- In the last factor of the right-hand side of the above, only one term depends on $\boldsymbol{w}_{ij}$, so that

$$\boxed{\frac{\partial z_i}{\partial w_{ij}}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{ik}a_k\right) = \boxed{a_j}$$

# Back propagation

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_i} \boxed{\frac{\partial a_i}{\partial z_i}} \frac{\partial z_i}{\partial w_{ij}}$$

- If the neuron is in the first layer after the input layer, $\boldsymbol{a_i}$ is just $\boldsymbol{x_i}$.

- The derivative of the output of neuron $\boldsymbol{j}$ ($\boldsymbol{a_i}$) with respect to its input ($\boldsymbol{z_i}$) is simply the partial derivative of the activation function (assuming here that the logistic function is used):

$$\boxed{\frac{\partial a_i}{\partial z_i}} = \frac{\partial}{\partial z_i}\sigma(z_i) = \boxed{\sigma(z_i)(1-\sigma(z_i))}$$

- This is the reason why backpropagation requires the activation function to be differentiable.

# Back propagation

$$\frac{\partial C}{\partial w_{ij}} = \boxed{\frac{\partial C}{\partial a_i}} \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial w_{ij}}$$

- The first factor is straightforward to evaluate if the neuron is in the output layer, because then $a_i = y$ and

$$\boxed{\frac{\partial C}{\partial a_i}} = \frac{\partial C}{\partial y} = \frac{\partial}{\partial y} \frac{1}{2} (t - y)^2 = y - t = \boxed{a_i - t}$$

- However, if $i$ is in an arbitrary inner layer of the network, it is less obvious.

# Back propagation

$$\frac{\partial C}{\partial w_{ij}} = \boxed{\frac{\partial C}{\partial a_i}} \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial w_{ij}}$$

- Considering $\boldsymbol{C}$ as a function of the inputs of all neurons $\boldsymbol{L=u, v, \ldots, w}$ receiving input from neuron $\boldsymbol{i}$,

$$\boxed{\frac{\partial C(a_i)}{\partial a_i}} = \frac{\partial C(z_u, z_v, \ldots, z_w)}{\partial a_i}$$

- and taking the **total derivative** with respect to $\boldsymbol{a_i}$, a recursive expression for the derivative is obtained:

$$\boxed{\frac{\partial C}{\partial a_i}} = \sum_{l \in L} \left( \frac{\partial C}{\partial z_l} \frac{\partial z_l}{\partial a_i} \right) = \boxed{\sum_{l \in L} \left( \frac{\partial C}{\partial a_l} \frac{\partial a_l}{\partial z_l} w_{li} \right)}$$

- Therefore, the derivative with respect to $\boldsymbol{a_i}$ can be calculated if all the derivatives with respect to the outputs $\boldsymbol{a_l}$ of the next layer are known.

# Back propagation

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial a_i} \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial w_{ij}}$$

- Putting it all together from previous slides:

$$\frac{\partial C}{\partial w_{ij}} = \delta_i a_j$$

- where

$$\delta_i = \frac{\partial C}{\partial a_i} \frac{\partial a_i}{\partial z_i} = \begin{cases} (a_i - t_i) a_i (1 - a_i) & \text{if } i \text{ is an output neuron} \\ (\sum_{l \in L} \delta_l w_{li}) a_i (1 - a_i) & \text{if } i \text{ is an inner neuron} \end{cases}$$

# Back propagation: Final

$$\frac{\partial C}{\partial w_{ij}} = \delta_i a_j$$

$$\delta_i = \frac{\partial C}{\partial a_i}\frac{\partial a_i}{\partial z_i} = \begin{cases} (a_i - t_i)a_i(1 - a_i) & \text{if } i \text{ is an output neuron} \\ (\sum_{l \in L} \delta_l w_{li})a_i(1 - a_i) & \text{if } i \text{ is an inner neuron} \end{cases}$$

- To update the weight $w_{ij}$ using gradient descent, one must choose a learning rate, $\boldsymbol{\alpha}$.
- The change in weight, which is added to the old weight, is equal to the product of the learning rate and the gradient, multiplied by −1:

$$\boxed{\Delta w_{ij} = -\alpha\frac{\partial C}{\partial w_{ij}} = \begin{cases} -\alpha a_j(a_i - t_i)a_i(1 - a_i) & \text{if } i \text{ is an output neuron} \\ -\alpha a_j(\sum_{l \in L} \delta_l w_{li})a_j(1 - a_i) & \text{if } i \text{ is an inner neuron} \end{cases}}$$