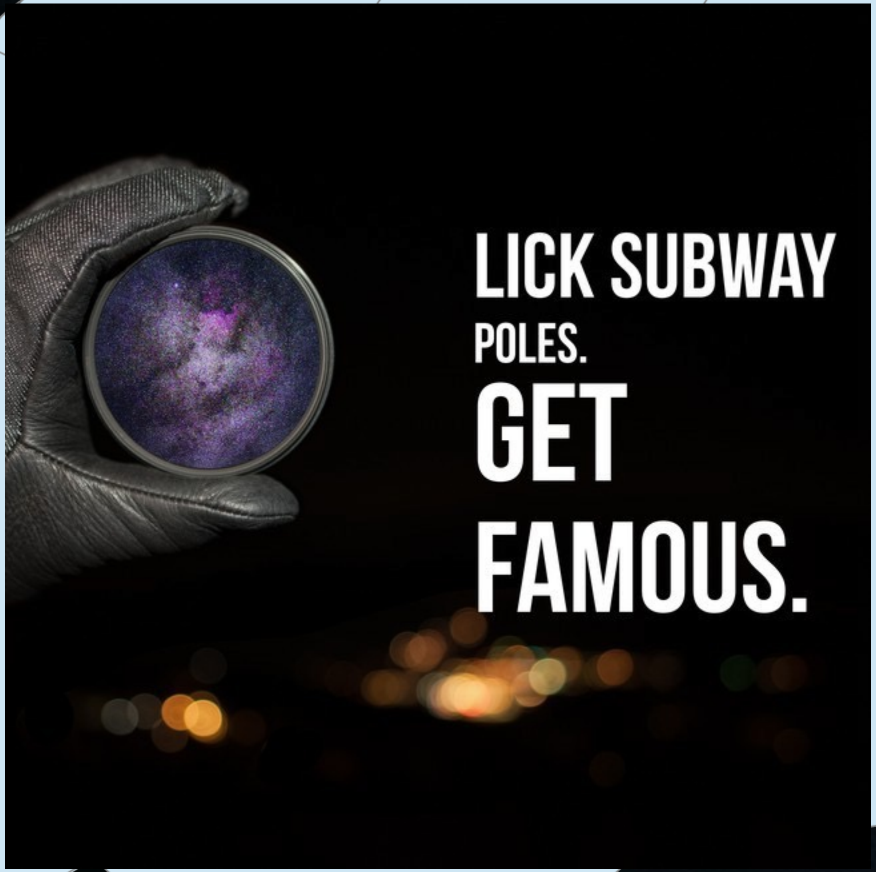
Daily Inspiration



Today

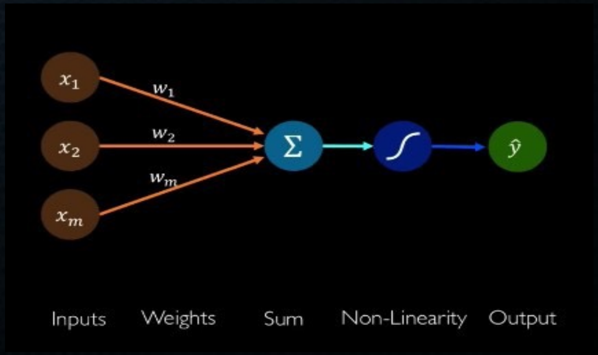
- Recap yesterday
- Neural network backpropagation
- Convolutional neural networks

Yesterday

- Logistic regression:
  - use the sigmoid function to turn the tools of regression into classification
  - Logistic regression cost function:

$$\mathrm{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1-h_\theta(x))$$

  - Multiclass (*i* classes)*:* train *i* binary classifiers
- Neural networks:
  - Hierarchically ordered units that calculate ever more complex functions using simpler calculated features
  - Universal approximation
  - Forward propagation
  - Multiclass: turn training samples into *i*-dimensional vectors



Exam

- Value of this course is in the practicals, in (trying to) do it yourself.
- Exam will ask mostly basic understanding:
  - I have these predictions and these Y, what is the MSE?
  - Showing that you understood how to get linear regression predictions with linear algebra
  - Calculating a few Euclidean distances
  - Showing that you know how nested cross-validation works and why we use it

Exam

- Since we worked with them extensively, you should be able to write down the partial derivatives of a simple linear regression
- For a simple neural network image: should be able to write down in linear algebra how a certain layer calculates its activations.
- Should be able to write down how backprop works for a NN
- Bit of *pseudocode*: I will probably give you part of the steps for a certain function you implemented. You complete it by writing down the logical steps you would need to do in words (doesn't need to be working code, but the right steps).

Before we start

$J(neuron_1, neuron_2, neuron_3 \ldots neuron_n)$ = network predictions – real values (actually use binary cross-entropy)

Activations this neuron gets
from other neurons

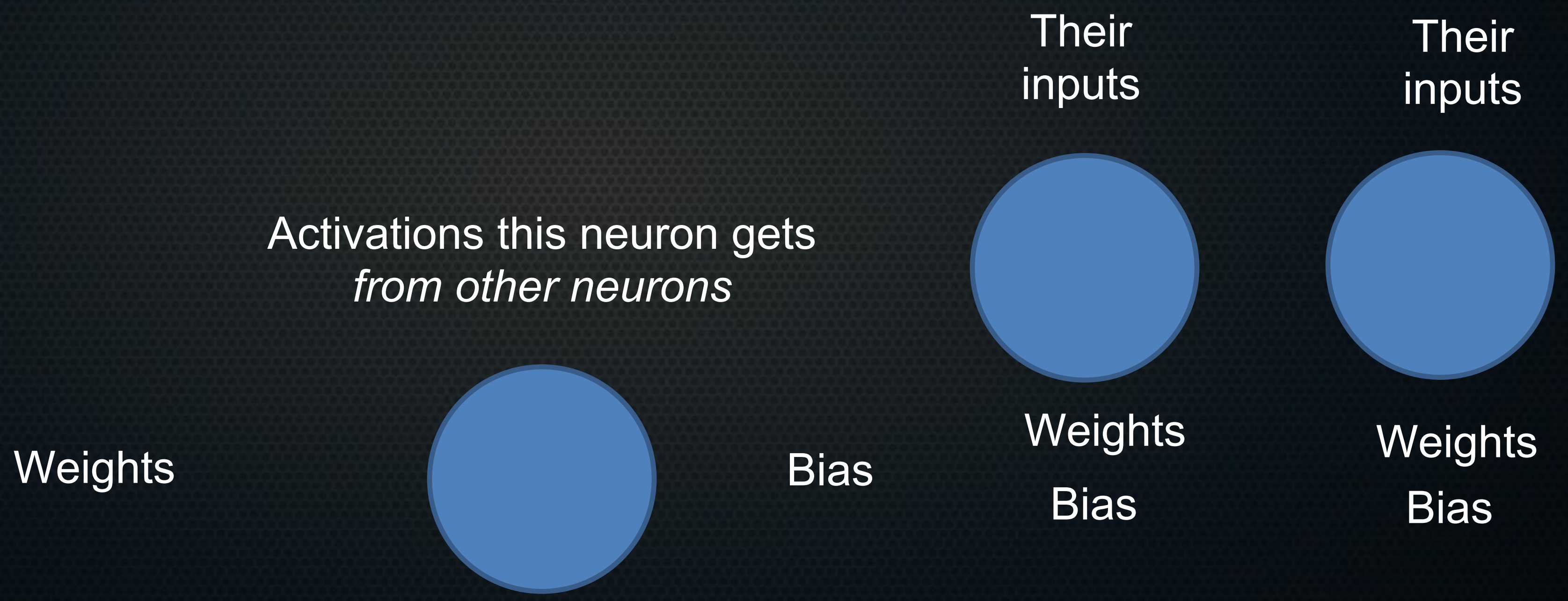
Weights

Bias

What knobs can we
turn in our function?

Before we start

$J(neuron_1, neuron_2, neuron_3 \ldots neuron_n)$ = network predictions – real values (actually use binary cross-entropy)



What knobs can we turn in our function?

Before we start

$J(neuron_1, neuron_2, neuron_3 \dots neuron_n)$ = network predictions – real values (actually use binary cross-entropy)

a(b(c(d(e(f(g(h(i(j(k(x)))))))))))

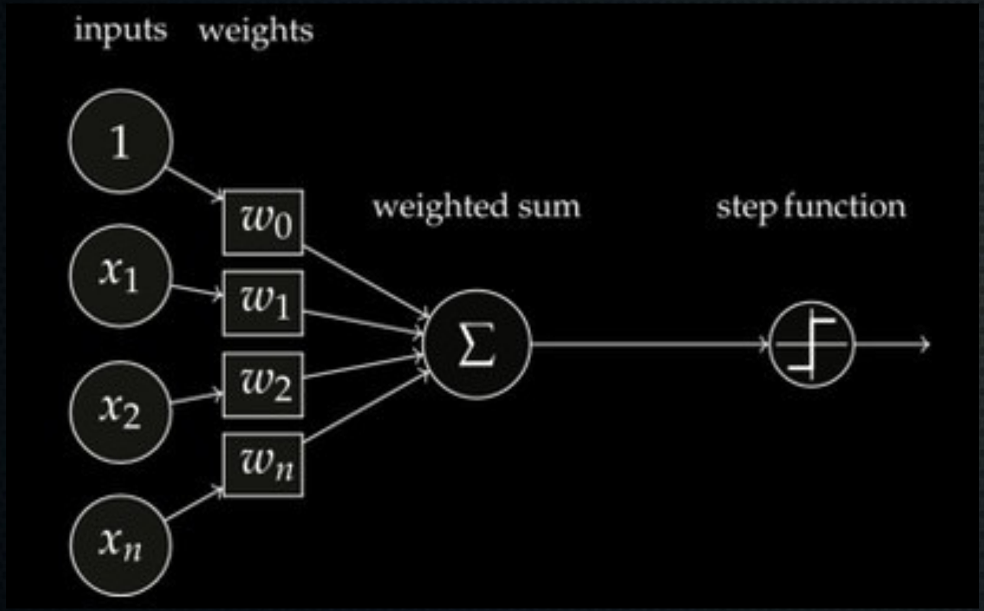
Lots of chain rules for the partial derivatives!

Before we start

- This is going to get quite mathy.
- Implementing this yourself is a clear *stretch goal* . I want you to understand how backpropagation works, and the basic partial derivatives you use and chain for it. Actually making it yourself with linear algebra is the dream, but probably not the reality.

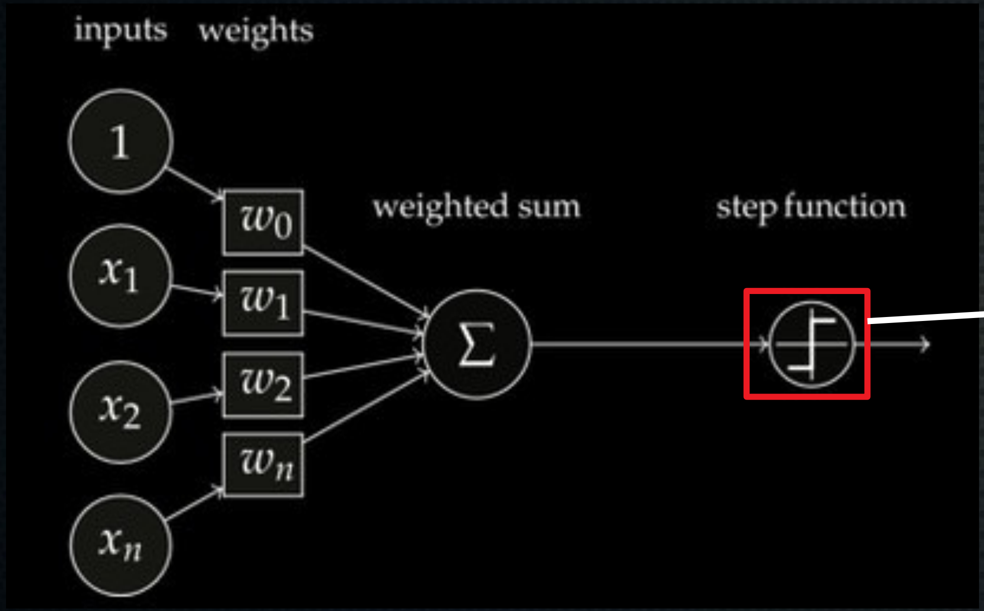
Backpropagation

- Some history: neural networks started out as perceptrons.



Backpropagation

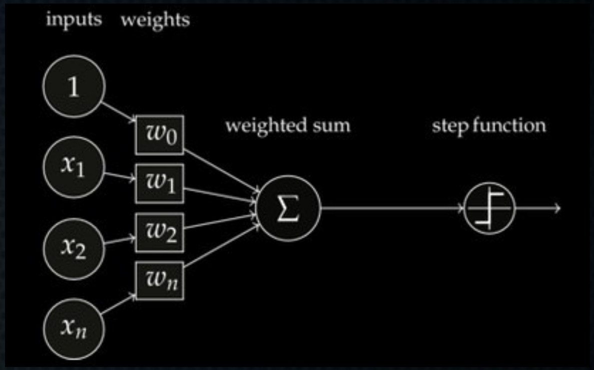
- Some history: neural networks started out as perceptrons.



Output is either 1 or 0. No in-between like the sigmoid!

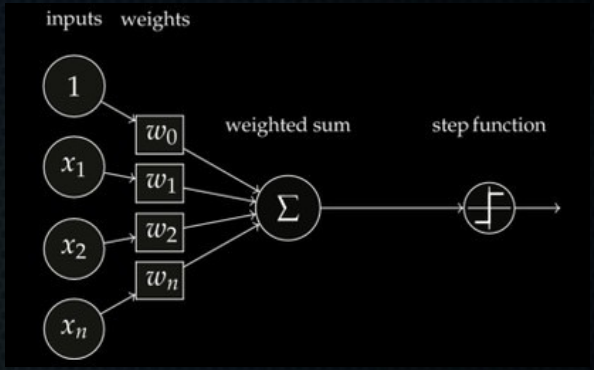
Backpropagation

- Some history: neural networks started out as perceptrons.
- In 1969, a paper was published by Marvin Minsky and Seymour Papert that showed that a single perceptron could not learn XOR



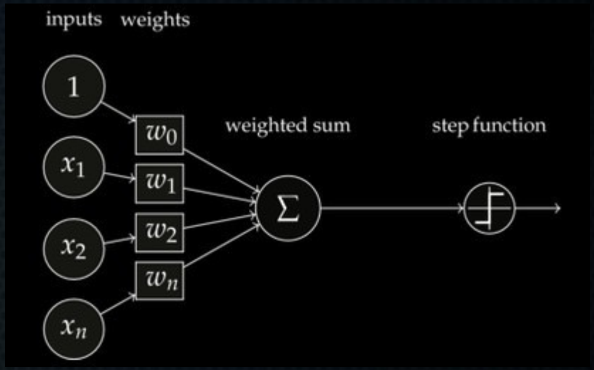
Backpropagation

- Some history: neural networks started out as perceptrons.
- In 1969, a paper was published by Marvin Minsky and Seymour Papert that showed that a single perceptron could not learn XOR
- While it was known that multi-layer perceptrons could, the perceptron learning rule was not good at learning multi-layer networks



Backpropagation

- Some history: neural networks started out as perceptrons.
- In 1969, a paper was published by Marvin Minsky and Seymour Papert that showed that a single perceptron could not learn XOR
- While it was known that multi-layer perceptrons could, the perceptron learning rule was not good at learning multi-layer networks
- Unfortunately, this mostly killed neural network research for a decade (!)



Backpropagation

- 1986 to the rescue!*

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

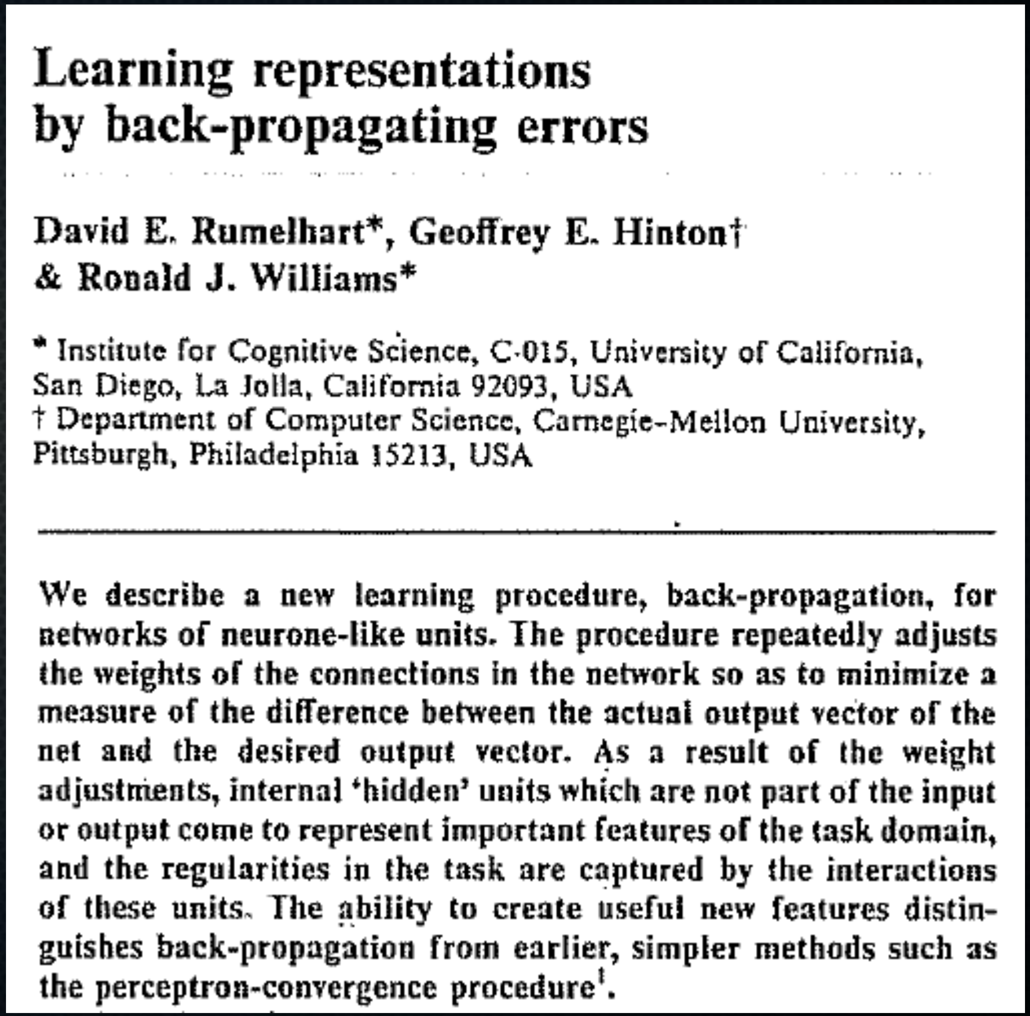
* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

*Not completely true, approach stemmed from the 70s, and someone else even published on it in control systems in 1960, but this paper really showed the *power* of the approach and renewed interest!

Backpropagation

- 1986 to the rescue!
- Idea: rather than the step-function we use a smooth, *differentiable* function (sigmoid or other).
- We know the error in the last layer (we know true classes and we know the vector that our NN outputs)
- Due to this, we can take the error, change the parameters of a layer, then go back a layer and change the parameters there, etc.



Learning representations by back-propagating errors

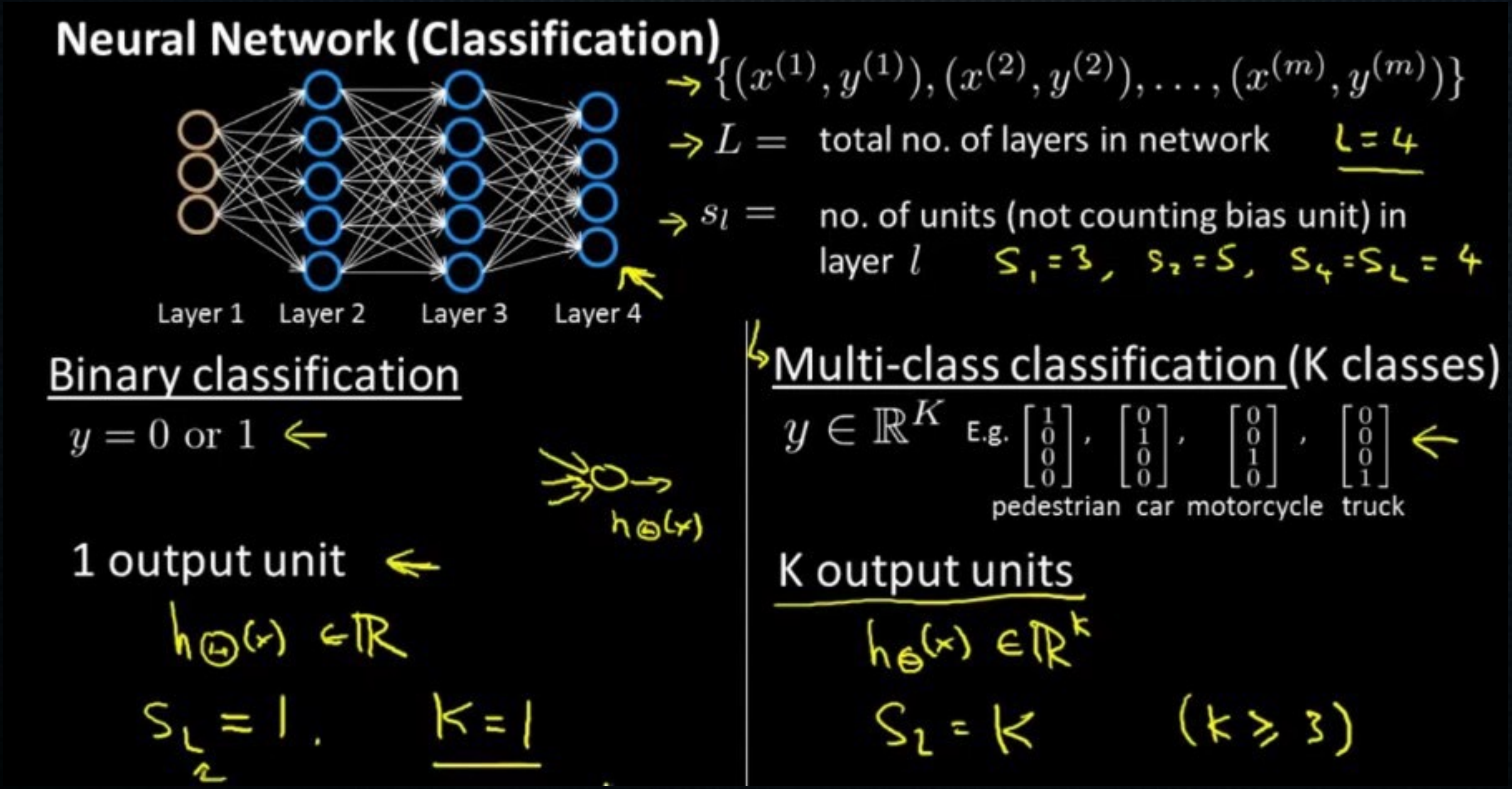
David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

Backpropagation - terminology

- How does that work?



Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\mathrm{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1-h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\mathrm{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1-h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

$$\mathrm{Cost}(x) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right]$$

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\text{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1 - h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

$$\text{Cost}(x) = -\frac{1}{m}\Big[\underbrace{\sum_{i=1}^{m}}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\Big]$$

We have m training samples and calculate the cost over each sample *i*

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\text{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1 - h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

$$\text{Cost}(x) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\theta(x^{(i)}))_k)\right]$$

We have m training samples and calculate the cost over each sample *i*

We have K classes and sum the cost of each entry *k* in the class vector

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\mathrm{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1-h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

$$\mathrm{Cost}(x) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\theta(x^{(i)}))_k)\right]$$

We have m training samples and calculate the cost over each sample *i*

We have K classes and sum the cost of each entry *k* in the class vector

$$y^{(3)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad h_\theta(x^{(3)}) = \begin{bmatrix} 0.234 \\ 0.678 \\ 0.102 \\ 0.020 \\ 0.800 \end{bmatrix}$$

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\text{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1-h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

$$\text{Cost}(x) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right]$$

We have m training samples and calculate the cost over each sample *i*

We have K classes and sum the cost of each entry *k* in the class vector

$$\begin{array}{r} k=1 \\ \\ y^{(3)} = \\ \\ \\ \end{array} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad h_\theta(x^{(3)}) = \begin{bmatrix} 0.234 \\ 0.678 \\ 0.102 \\ 0.020 \\ 0.800 \end{bmatrix}$$

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\mathrm{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1-h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

$$\mathrm{Cost}(x) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\theta(x^{(i)}))_k)\right]$$

We have m training samples and calculate the cost over each sample *i*

We have K classes and sum the cost of each entry *k* in the class vector

$$\begin{array}{rc} & \\ k = 2 & \\ y^{(3)} = & \end{array}\begin{bmatrix} 0 \\ \boxed{1} \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad h_\theta(x^{(3)}) = \begin{bmatrix} 0.234 \\ \boxed{0.678} \\ 0.102 \\ 0.020 \\ 0.800 \end{bmatrix}$$

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\mathrm{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1-h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

$$\mathrm{Cost}(x) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right]$$

We have m training samples and calculate the cost over each sample *i*

We have K classes and sum the cost of each entry *k* in the class vector

$$y^{(3)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad h_\theta(x^{(3)}) = \begin{bmatrix} 0.234 \\ 0.678 \\ 0.102 \\ 0.020 \\ 0.800 \end{bmatrix}$$

k = *K* = 5

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\mathrm{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1-h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

$$\mathrm{Cost}(x) = -\frac{1}{m}\Big[\underbrace{\sum_{i=1}^{m}}\underbrace{\sum_{k=1}^{K}} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\Big]$$

We have m training samples and calculate the cost over each sample *i*

We have K classes and sum the cost of each entry *k* in the class vector

$$y^{(3)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \boxed{0} \end{bmatrix} \qquad h_\theta(x^{(3)}) = \begin{bmatrix} 0.234 \\ 0.678 \\ 0.102 \\ 0.020 \\ 0.800 \end{bmatrix}$$

k = *K* = 5

$$\mathrm{Cost}(x^{(3)}; \text{5th neuron}) = -\Big[\boxed{y_5^{(3)}}\log(h_\theta(x^{(3)}))_5 + (1-y_5^{(3)})\log(1-(h_\theta(x^{(3)}))_5)\Big]$$

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\mathrm{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1 - h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

$$\mathrm{Cost}(x) = -\frac{1}{m}[\sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\theta(x^{(i)}))_k)]$$

We have m training samples and calculate the cost over each sample *i*

We have K classes and sum the cost of each entry *k* in the class vector

$$y^{(3)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \boxed{0} \end{bmatrix} \qquad h_\theta(x^{(3)}) = \begin{bmatrix} 0.234 \\ 0.678 \\ 0.102 \\ 0.020 \\ 0.800 \end{bmatrix}$$

k = *K* = 5

$$\mathrm{Cost}(x^{(3)}; \text{5th neuron}) = -[\boxed{0} + (1 - y_5^{(3)}) \log(1 - (h_\theta(x^{(3)}))_5)]$$

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\mathrm{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1-h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

$$\mathrm{Cost}(x) = -\frac{1}{m}\Big[\underbrace{\sum_{i=1}^{m}}\underbrace{\sum_{k=1}^{K}} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\Big]$$

We have m training samples and calculate the cost over each sample *i*

We have K classes and sum the cost of each entry *k* in the class vector

$$y^{(3)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \boxed{0} \end{bmatrix} \qquad h_\theta(x^{(3)}) = \begin{bmatrix} 0.234 \\ 0.678 \\ 0.102 \\ 0.020 \\ \boxed{0.800} \end{bmatrix}$$

k = *K* = 5

$$\mathrm{Cost}(x^{(3)}; 5\text{th neuron}) = -[\ \boxed{0}\ + (1-\boxed{y_5^{(3)}})\log(1-\boxed{(h_\theta(x^{(3)}))_5})]$$

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\mathrm{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1 - h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

$$\mathrm{Cost}(x) = -\frac{1}{m}\Big[\underbrace{\sum_{i=1}^{m}}\underbrace{\sum_{k=1}^{K}} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\theta(x^{(i)}))_k)\Big]$$

We have m training samples and calculate the cost over each sample *i*

We have K classes and sum the cost of each entry *k* in the class vector

$$y^{(3)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \boxed{0} \end{bmatrix} \qquad h_\theta(x^{(3)}) = \begin{bmatrix} 0.234 \\ 0.678 \\ 0.102 \\ 0.020 \\ \boxed{0.800} \end{bmatrix}$$

k = *K* = 5

$$\mathrm{Cost}(x^{(3)}; \text{5th neuron}) = -[\ 0\ + (1 - \boxed{0}) \log(1 - \boxed{0.8})]$$

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\mathrm{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1-h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

$$\mathrm{Cost}(x) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right]$$

We have m training samples and calculate the cost over each sample *i*

We have K classes and sum the cost of each entry *k* in the class vector

$$y^{(3)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad h_\theta(x^{(3)}) = \begin{bmatrix} 0.234 \\ 0.678 \\ 0.102 \\ 0.020 \\ 0.800 \end{bmatrix}$$

k = *K* = 5

$$\mathrm{Cost}(x^{(3)}; \text{5th neuron}) = -[\log(0.2)] \approx 0.69897$$

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\mathrm{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1-h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

$$\mathrm{Cost}(x) = -\frac{1}{m}[\underbrace{\sum_{i=1}^{m}}_{\text{We have m training samples and calculate the cost over each sample } i} \underbrace{\sum_{k=1}^{K}}_{\text{We have K classes and sum the cost of each entry } k \text{ in the class vector}} \underbrace{y_k^{(i)} \log(h_\theta(x^{(i)}))_k}_{\text{Error if y = 1 for this position in the class vector}} + \underbrace{(1-y_k^{(i)}) \log(1-(h_\theta(x^{(i)}))_k)}_{\text{Error if y = 0 for this position in the class vector}}]$$

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\mathrm{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1-h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)

$$\mathrm{Cost}(x) = \underbrace{-\frac{1}{m}}_{\text{Average over all training samples } m} [\underbrace{\sum_{i=1}^{m}}_{\text{We have m training samples and calculate the cost over each sample } i} \underbrace{\sum_{k=1}^{K}}_{\text{We have K classes and sum the cost of each entry } k \text{ in the class vector}} \underbrace{y_k^{(i)} \log(h_\theta(x^{(i)}))_k}_{\text{Error if y = 1 for this position in the class vector}} + \underbrace{(1-y_k^{(i)}) \log(1-(h_\theta(x^{(i)}))_k)}_{\text{Error if y = 0 for this position in the class vector}}]$$

Backpropagation

- **We (may) want to regularise all weights and biases for each layer in the network**

$$\text{Cost}(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right]$$

$$\boxed{+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\theta_{ij}^{(l)})^2}$$

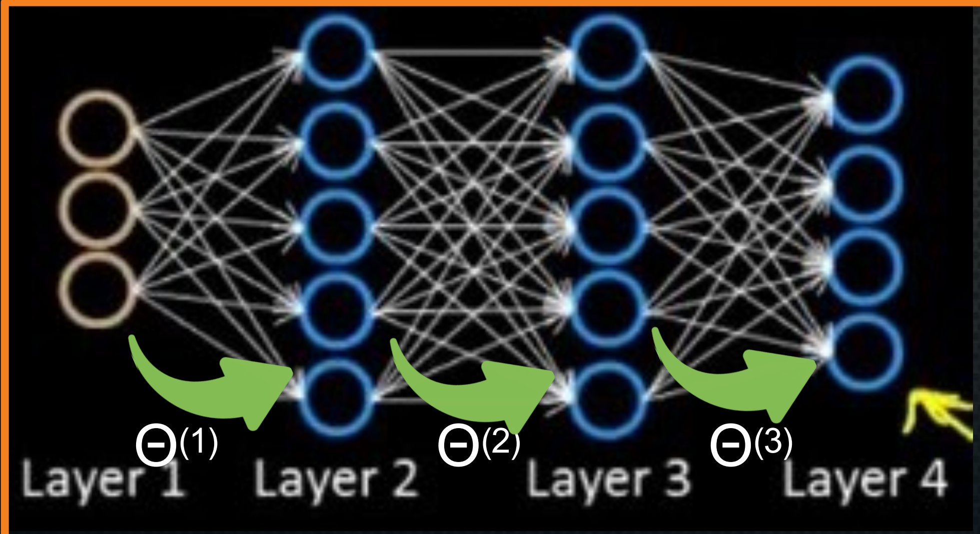
l = which layer's parameters are we looking at?
i = which unit in that layer (row in theta matrix) are we looking at?
j = what parameter of that unit (column) are we looking at?

Backpropagation

$$+\frac{\lambda}{2m}\boxed{\sum_{l=1}^{L-1}}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\theta_{ij}^{(l)}\right)^2$$

l = which layer's parameters are we looking at?
i = which unit in that layer (row in theta matrix) are we looking at?
j = what parameter of that unit (column) are we looking at?

$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$



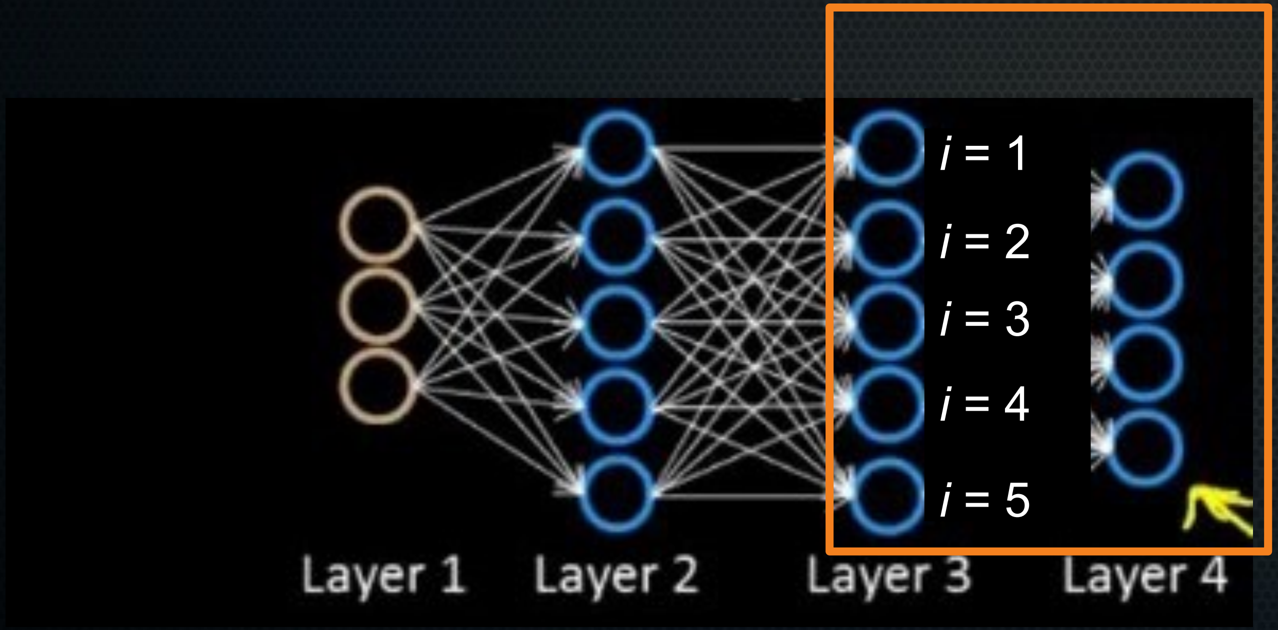
Have 3 matrices of parameters.

$\Theta^{(1)}$ = parameters layer 2 uses to calculate values with layer 1 as input.

Backpropagation

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\boxed{\sum_{i=1}^{s_l}}\sum_{j=1}^{s_{l+1}}(\theta_{ij}^{(l)})^2$$

l = which layer's parameters are we looking at?
i = which unit in that layer (row in theta matrix) are we looking at?
j = what parameter of that unit (column) are we looking at?

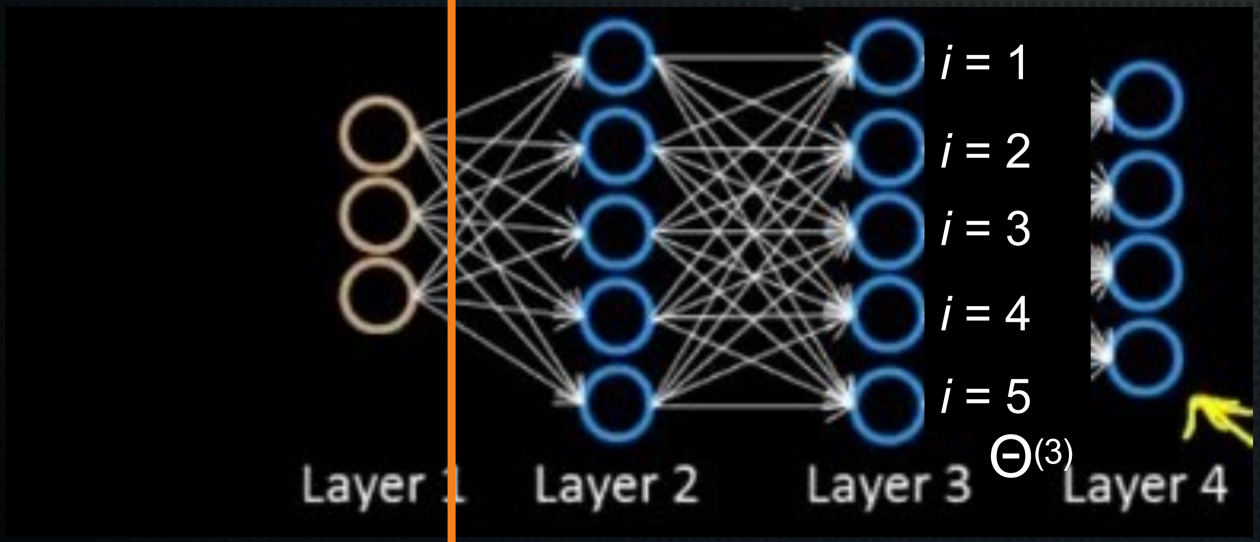


Each unit's parameters are in a certain row of that layer's Θ matrix.

Backpropagation

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\boxed{\sum_{j=1}^{s_{l+1}}}(\theta_{ij}^{(l)})^2$$

l = which layer's parameters are we looking at? ($\Theta^{(3)}$ = params layer 4)
i = which unit in that layer (row in theta matrix) are we looking at?
j = what parameter of that unit (column) are we looking at?



Every neuron has multiple parameters, so which one of those are we looking at?

$$\begin{array}{cc} & j \longrightarrow \\ i \downarrow & \begin{pmatrix} b_1 & w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ b_2 & w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ b_3 & w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \\ b_4 & w_{41} & w_{42} & w_{43} & w_{44} & w_{45} \end{pmatrix} \end{array}$$

Backpropagation

- First, we need a cost function. For logistic regression we used:

$$\mathrm{Cost}(x) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1-h_\theta(x))$$

- We use the same thing, only generalised for the fact that:
  - Our output is a vector (not a 1 or 0 as for logistic regression)
  - We (may) want to regularise all weights and biases for each layer in the network

$$\mathrm{Cost}(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\theta_{ij}^{(l)})^2$$

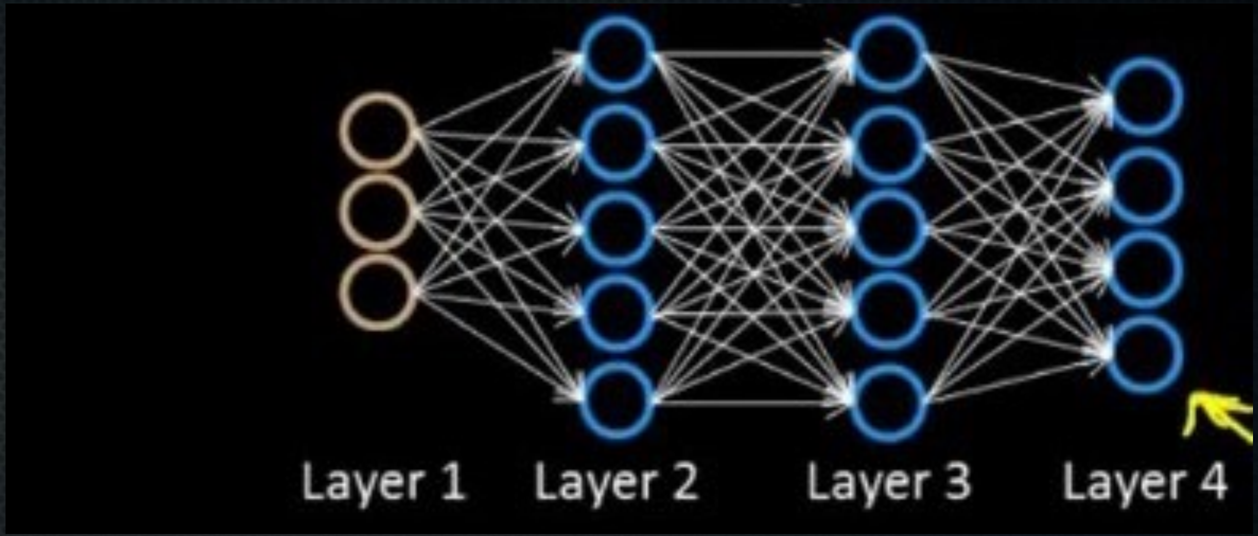
l = which layer‘s parameters are we looking at?
i = which unit in that layer (row in theta matrix) are we looking at?
j = what parameter of that unit (column) are we looking at?

How do we use this cost to update parameters?

Given one training example (x, y):

Forward propagation:

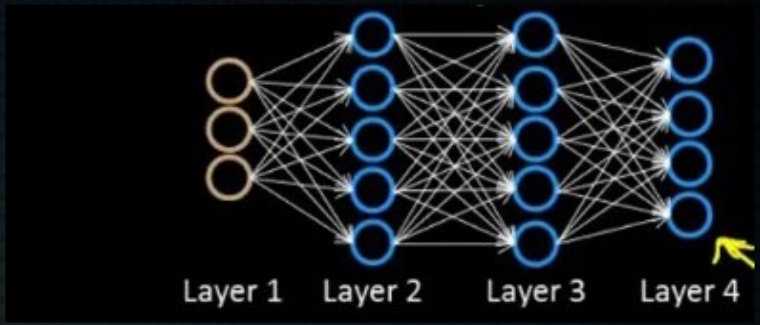
$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\Theta(x) = g(z^{(4)})
\end{aligned}
$$



$$z = \left(\sum_{i=1}^{n_{\text{input neurons}}} w_i \cdot x_i \right) + b$$

$$g = sigmoid = \frac{1}{1+e^{-z}}$$

How do we use this cost to update parameters?



- Scheme as follows:

Calculate cost of the current network

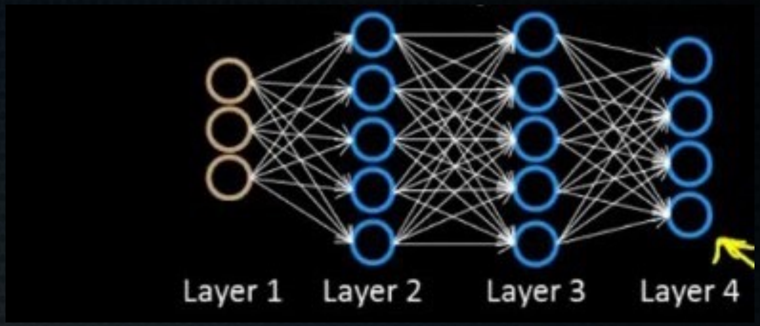
Calculate partial derivative of the cost w.r.t. each parameter

Take a small step in this direction

Gradient descent

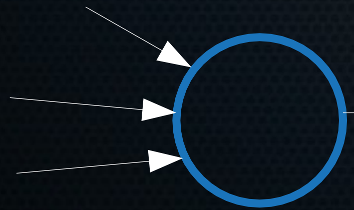
How do we use this cost to update parameters?

- With all these neurons in all these layers, we have a cost function that is dependent on *many* parameters.
- To tackle this complexity: let‘s first see how one *neuron* influences the cost function.



Calculate cost of the current network

Gradient descent:
- Calculate partial derivative of the cost w.r.t. each parameter
- Take a small step in this direction



Incoming activations from layer l-1 (including a bias neuron)

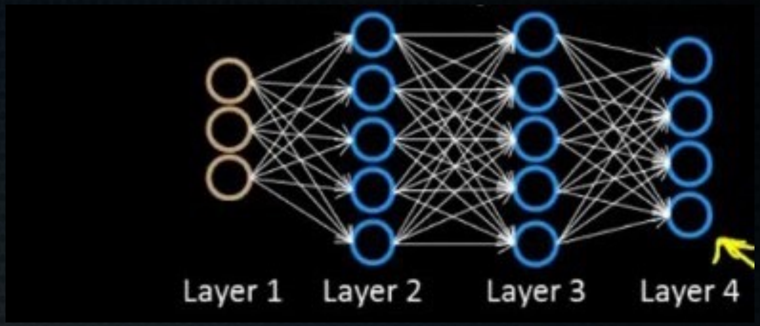
Layer l

$$z_j^{(l)} = \Theta^{(l-1)} \cdot a^{(l-1)} \rightarrow \text{sigmoid}$$

sigmoid → Next layer

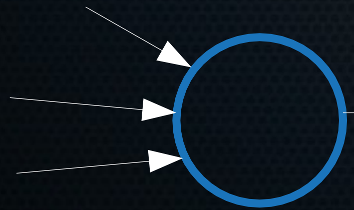
How do we use this cost to update parameters?

- Partial derivative w.r.t. this neuron's z = $\dfrac{\partial C}{\partial z_j^{(l)}}$
- If we somehow change this neuron's z with small $\Delta z_j^{(l)}$, then total cost will change with $\dfrac{\partial C}{\partial z_j^{(l)}} \cdot \Delta z_j^{(l)}$



Calculate cost of the current network

Gradient descent:
- Calculate partial derivative of the cost w.r.t. each parameter
- Take a small step in this direction



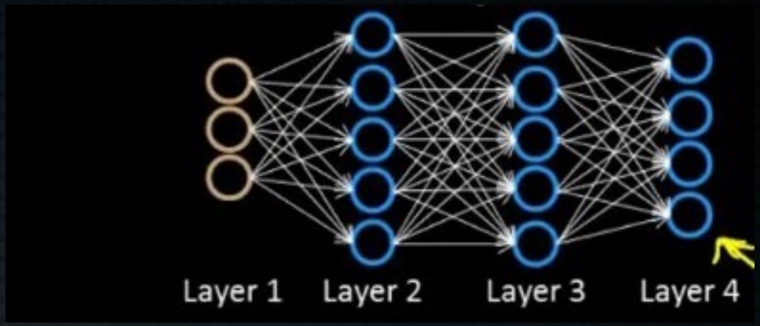
$z_j^{(l)} = \Theta^{(l-1)} \cdot a^{(l-1)}$ → sigmoid → Next layer

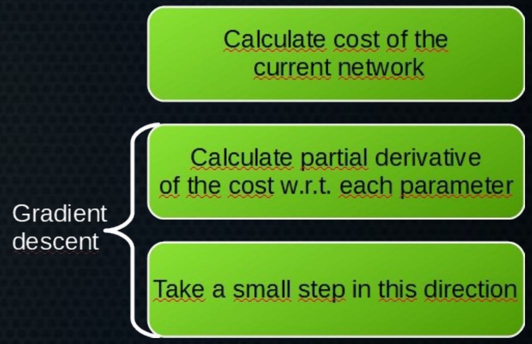
Incoming activations from layer l-1 (including a bias neuron)

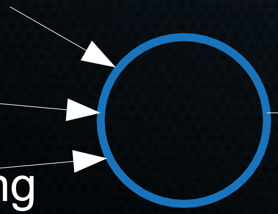
Layer l

How do we use this cost to update parameters?

- Partial derivative w.r.t. this neuron‘s z = $\frac{\partial C}{\partial z_j^{(l)}}$
- If we somehow change this neuron‘s z with small $\Delta z_j^{(l)}$, then total cost will change with $\frac{\partial C}{\partial z_j^{(l)}} \cdot \Delta z_j^{(l)}$
- If $\frac{\partial C}{\partial z_j^{(l)}}$ is 20, then a small nudge of -0.1 gives a change in total cost of -2.



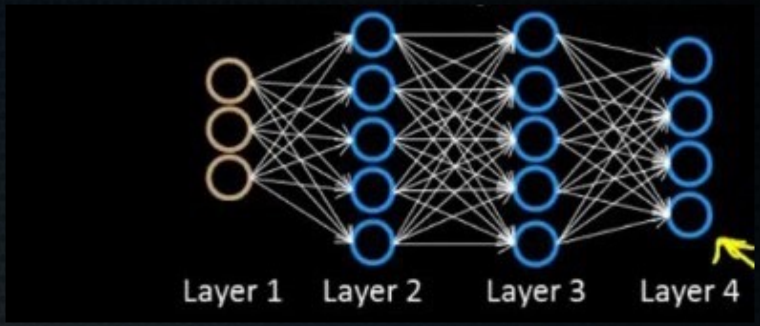


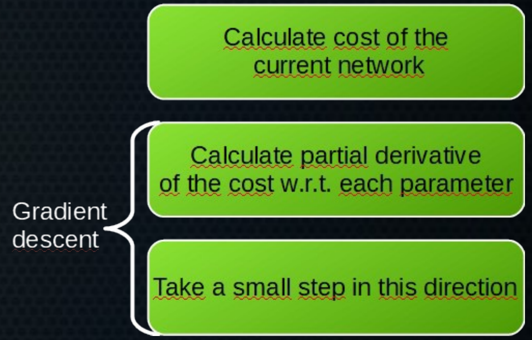


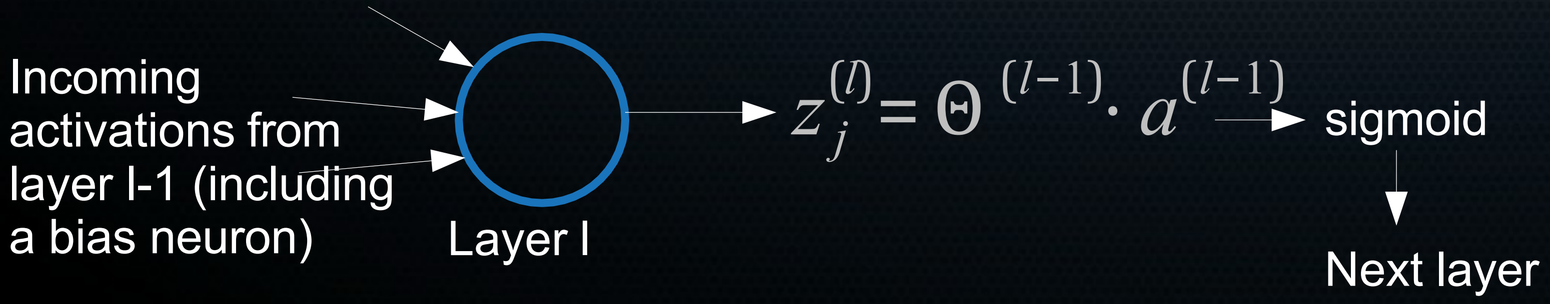
$z_j^{(l)} = \Theta^{(l-1)} \cdot a^{(l-1)}$ → sigmoid → Next layer

How do we use this cost to update parameters?

- Partial derivative w.r.t. this neuron's z = $\dfrac{\partial C}{\partial z_j^{(l)}}$
- If we somehow change this neuron's z with small $\Delta z^{(l)}_j$, then total cost will change with $\dfrac{\partial C}{\partial z_j^{(l)}} \cdot \Delta z_j^{(l)}$
- If $\dfrac{\partial C}{\partial z_j^{(l)}}$ is 0.01, then a small nudge of -0.1 gives a change in cost of -0.001.

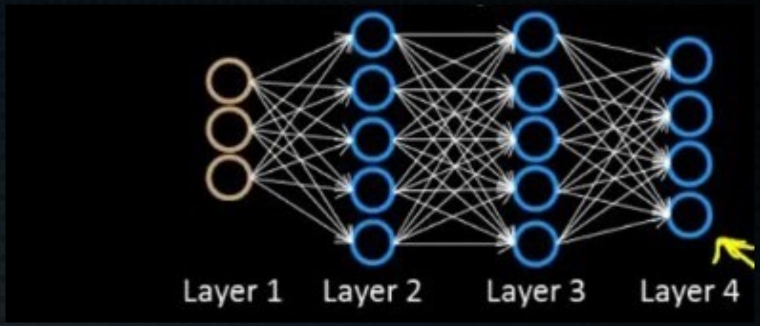


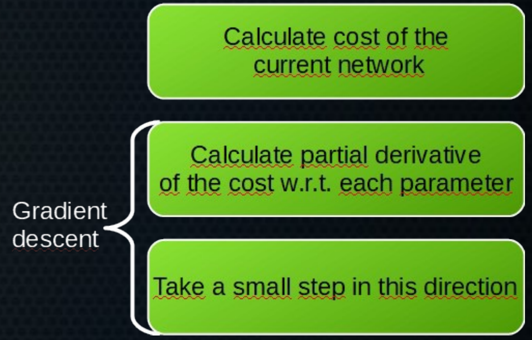




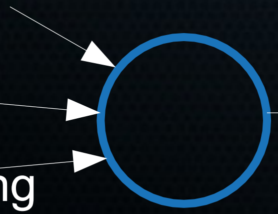
How do we use this cost to update parameters?

- Partial derivative w.r.t. this neuron‘s z = $\boxed{\dfrac{\partial C}{\partial z_j^{(l)}}}$
- If we somehow change this neuron‘s z with small $\Delta z_j^{(l)}$, then total cost will change with $\dfrac{\partial C}{\partial z_j^{(l)}} \cdot \Delta z_j^{(l)}$
- Hence, we see that this quantity shows heuristically how wrong a neuron is.





Incoming activations from layer l-1 (including a bias neuron)



Layer l

$z_j^{(l)} = \Theta^{(l-1)} \cdot a^{(l-1)}$ → sigmoid

↓

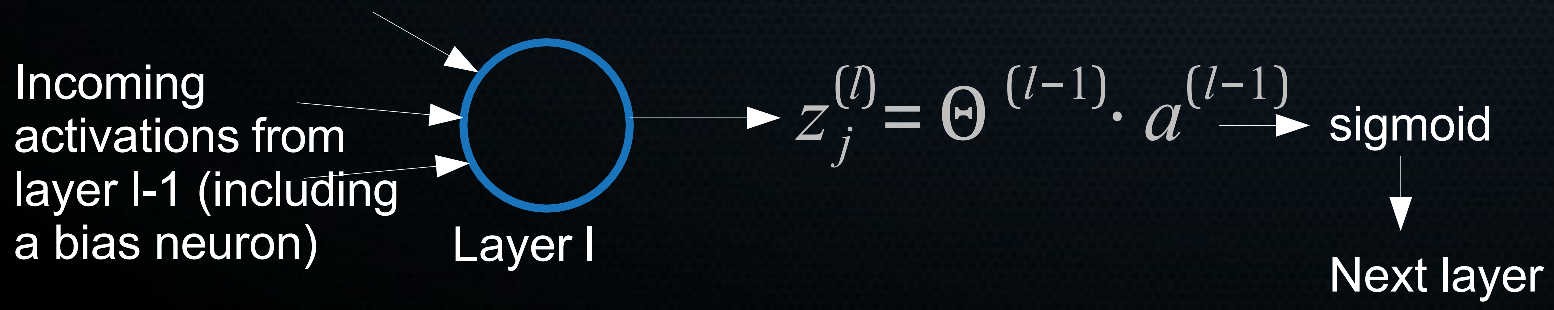
Next layer

How do we use this cost to update parameters?

- So, the error for a neuron:
  $$\delta_j^{(l)} = \text{error of node j in layer l} = \frac{\partial C}{\partial z_j^{(l)}}$$

- Remember, we want to change the weights and biases, that is, calculate all $\frac{\partial C}{\partial w_{jk}^{(l)}}$ $\frac{\partial C}{\partial b_j^{(l)}}$

- Then we can take a small step and update.



How do we use this cost to update parameters?

- So, the error for a neuron:

$$\delta_j^{(l)} = \text{error of node j in layer l} = \frac{\partial C}{\partial z_j^{(l)}}$$

- If we can :
  - Find this error for every neuron
  - Relate this quantity to how to change the weights and biases

We are set.

How do we use this cost to update parameters?

- So, the error for a neuron:

$$\delta_j^{(l)} = \text{error of node j in layer l} = \frac{\partial C}{\partial z_j^{(l)}}$$

~~Ab~~using the chain rule:

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \frac{\delta_j^{(l)} \cdot \partial \left(\sum_{k=1}^{n_{weights}} w_{jk}^{(l)} \cdot a^{(l-1)} + b_j^{(l)} \right)}{\partial b_j^{(l)}}$$

How do we use this cost to update parameters?

- So, the error for a neuron:

$$\delta_j^{(l)} = \text{error of node j in layer l} = \frac{\partial C}{\partial z_j^{(l)}}$$

Using the chain rule:

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\boxed{\partial z_j^{(l)}}} \cdot \frac{\boxed{\partial z_j^{(l)}}}{\partial b_j^{(l)}} = \frac{\delta_j^{(l)} \cdot \sum_{k=1}^{n_{weights}} w_{jk}^{(l)} \cdot a^{(l-1)} + b_j^{(l)}}{\partial b_j^{(l)}}$$

These cancel out, giving what we want: how to change the bias of a neuron

How do we use this cost to update parameters?

- So, the error for a neuron:

$$\boxed{\delta_j^{(l)}} = \text{error of node j in layer l} = \frac{\partial C}{\partial z_j^{(l)}}$$

Using the chain rule:

$$\frac{\partial C}{\partial b_j^{(l)}} = \boxed{\frac{\partial C}{\partial z_j^{(l)}}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \boxed{\delta_j^{(l)} \cdot} \frac{\sum_{k=1}^{n_{weights}} w_{jk}^{(l)} \cdot a^{(l-1)} + b_j^{(l)}}{\partial b_j^{(l)}}$$

We just defined this delta term as shorthand for the error of each neuron‘s product of the weights with its inputs plus its bias

How do we use this cost to update parameters?

- So, the error for a neuron:

$$\delta_j^{(l)} = \text{error of node j in layer l} = \frac{\partial C}{\partial z_j^{(l)}}$$

Using the chain rule:

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \cdot \frac{\partial \boxed{\left(\sum_{k=1}^{n_{weights}} w_{jk}^{(l)} \cdot a^{(l-1)} + b_j^{(l)}\right)}}{\partial b_j^{(l)}}$$

This is just how a neuron calculates the weighted sum of the inputs (+ bias): weights * activations previous layer + bias

How do we use this cost to update parameters?

- So, the error for a neuron:

  $\delta_j^{(l)} = \text{error of node j in layer l} = \frac{\partial C}{\partial z_j^{(l)}}$

  Using the chain rule:

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \cdot \boxed{\frac{\partial \left(\sum_{k=1}^{n_{weights}} w_{jk}^{(l)} \cdot a^{(l-1)} + b_j^{(l)} \right)}{\partial b_j^{(l)}}}$$

The whole thing is the partial derivative of
z w.r.t. the bias.
How should we nudge the bias such that
the cost C decreases?
How does the cost change if we keep
everything the same but increase or
decrease this bias?

How do we use this cost to update parameters?

- So, the error for a neuron:

$$\delta_j^{(l)} = \text{error of node j in layer l} = \frac{\partial C}{\partial z_j^{(l)}}$$

Using the chain rule:

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \cdot \boxed{\frac{\partial \left(\sum_{k=1}^{n_{weights}} w_{jk}^{(l)} \cdot a^{(l-1)} + b_j^{(l)} \right)}{\partial b_j^{(l)}}}$$

Who can tell me what the partial derivative is?

Hint: $f(x,k) = 12x - 3k$

$$\frac{\partial f(x,k)}{\partial k} = -3$$

How do we use this cost to update parameters?

- So, the error for a neuron:

$$\delta_j^{(l)} = \text{error of node j in layer l} = \frac{\partial C}{\partial z_j^{(l)}}$$

Using the chain rule:

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \cdot \frac{\partial\left(\sum_{k=1}^{n_{weights}} \boxed{w_{jk}^{(l)} \cdot a^{(l-1)}} + b_j^{(l)}\right)}{\partial b_j^{(l)}}$$

Contains no $b_j^{(l)}$, so part. derivative w.r.t. $b_j^{(l)} = 0$.

How do we use this cost to update parameters?

- So, the error for a neuron:

  $$\delta_j^{(l)} = \text{error of node j in layer l} = \frac{\partial C}{\partial z_j^{(l)}}$$

  Using the chain rule:

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \cdot \frac{\partial\left(\sum_{k=1}^{n_{weights}} w_{jk}^{(l)} \cdot a^{(l-1)} + \boxed{b_j^{(l)}}\right)}{\partial b_j^{(l)}}$$

Just one, like the derivative of f(x) = x
equals 1.

How do we use this cost to update parameters?

- So, the error for a neuron:

$$\delta_j^{(l)} = \text{error of node j in layer l} = \frac{\partial C}{\partial z_j^{(l)}}$$

Using the chain rule:

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \cdot 1 = \delta_j^{(l)}$$

How do we use this cost to update parameters?

- So, the error for a neuron:

$$\delta_j^{(l)} = \text{error of node j in layer l} = \frac{\partial C}{\partial z_j^{(l)}}$$

Using the chain rule:

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \cdot 1 = \delta_j^{(l)}$$

Now for the weights:

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot \frac{\partial \left(\sum_{k=1}^{n_{weights}} w_{jk}^{(l)} \cdot a^{(l-1)} + b_j^{(l)} \right)}{\partial w_{jk}^{(l)}}$$

How do we use this cost to update parameters?

- So, the error for a neuron:

$$\delta_j^{(l)} = \text{error of node j in layer l} = \frac{\partial C}{\partial z_j^{(l)}}$$

Using the chain rule:

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \cdot 1 = \delta_j^{(l)}$$

Now for the weights:

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot \frac{\partial \left(\sum_{k=1}^{n_{weights}} w_{jk}^{(l)} \cdot a^{(l-1)} + b_j^{(l)} \right)}{\partial w_{jk}^{(l)}}$$

What is this partial derivative?

How do we use this cost to update parameters?

- So, the error for a neuron:

$$\delta_j^{(l)} = \text{error of node j in layer l} = \frac{\partial C}{\partial z_j^{(l)}}$$

Using the chain rule:

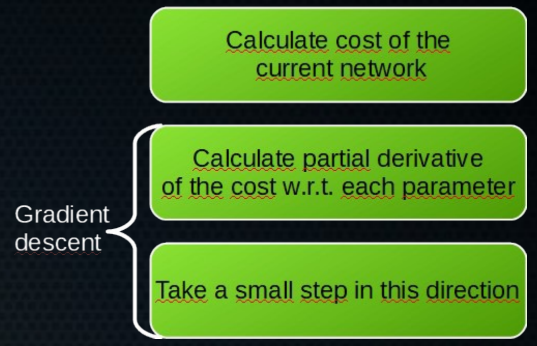
$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \cdot 1 = \delta_j^{(l)}$$

Now for the weights:

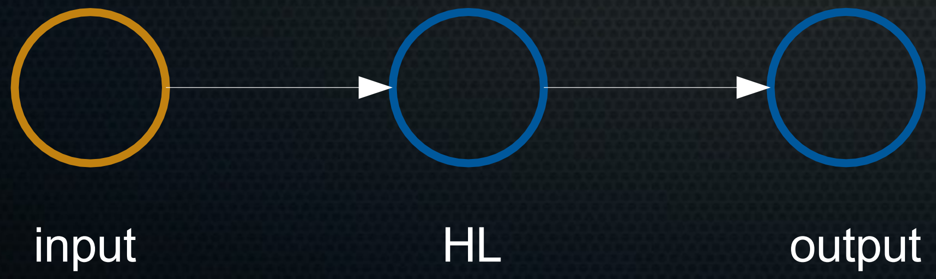
$$\frac{\partial C}{\partial w_{jk}^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot \frac{\partial \left(\sum_{k=1}^{n_{weights}} \overbrace{w_{jk}^{(l)} \cdot a^{(l-1)}}^{a^{(l-1)}} + \overbrace{b_j^{(l)}}^{0} \right)}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$

How do we use this cost to update parameters?

- This is a lot of math. Why are we doing this again?



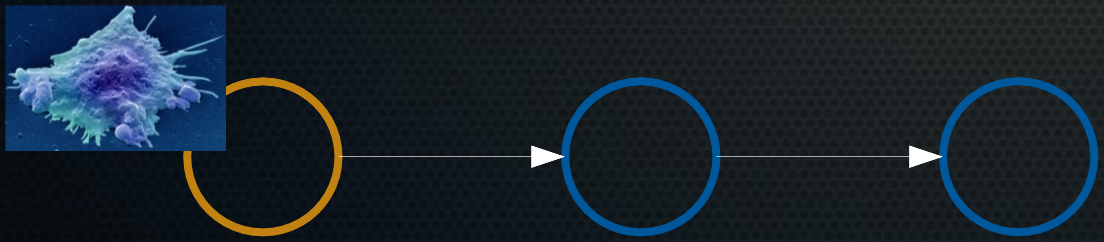
$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Big| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Big| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$



How do we use this cost to update parameters?

- This is a lot of math. Why are we doing this again?

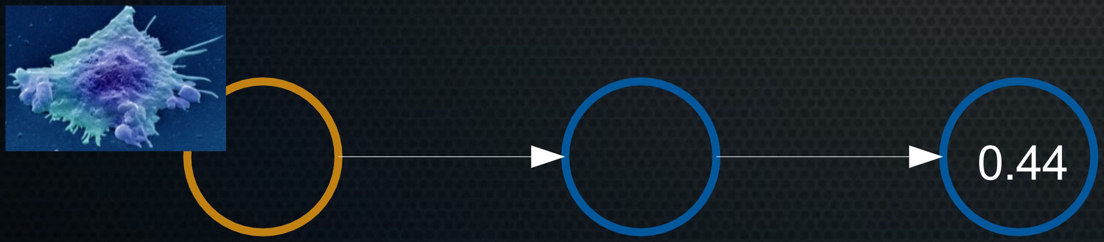
$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$



How do we use this cost to update parameters?

- This is a lot of math. Why are we doing this again?

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$



Colon cancer cell

input

HL

output

Is it a cancer cell?

0 = no
1 = yes

How do we use this cost to update parameters?

- This is a lot of math. Why are we doing this again?

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$

Colon cancer cell

input

HL

output

Is it a cancer cell?

0 = no
1 = yes

We want it to give 1!

How do we use this cost to update parameters?

- This is a lot of math. Why are we doing this again?

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$

Colon cancer cell

input

HL

output

Is it a cancer cell?

0 = no
1 = yes

$$MSE = \frac{1}{2}(y - a^{(L)})^2$$

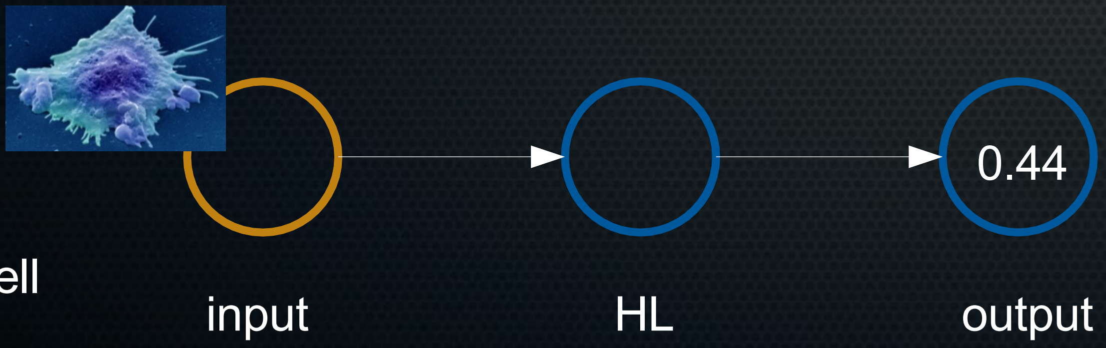
$$MSE = \frac{1}{2}(1 - 0.44)^2 = 0.1568$$

So we calculate a cost.
(For ease of calculation I use MSE, but in reality you would use the cost function explained earlier)

How do we use this cost to update parameters?

- This is a lot of math. Why are we doing this again?

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$



Colon cancer cell

Is it a cancer cell?

0 = no
1 = yes

$$MSE = \frac{1}{2}(y - a^{(L)})^2$$

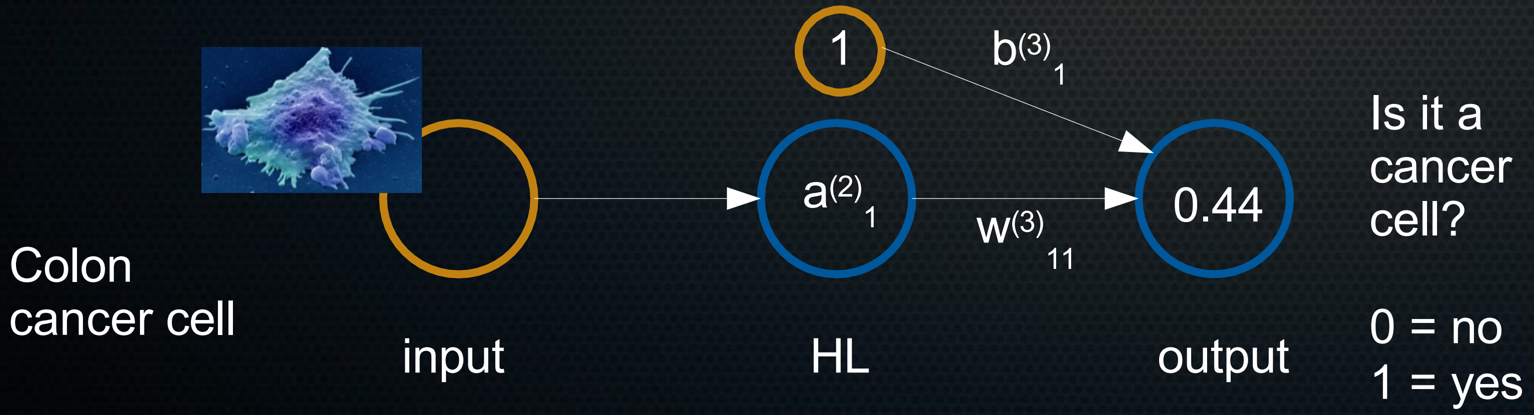
$$MSE = \frac{1}{2}(1 - 0.44)^2 = 0.1568$$

Okay, we know how wrong we are (for this one example). What next?

How do we use this cost to update parameters?

- This is a lot of math. Why are we doing this again?

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Big| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Big| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$



$$MSE = \frac{1}{2}\left(y - a^{(L)}\right)^2$$

$$MSE = \frac{1}{2}(1 - 0.44)^2 = 0.1568$$

Well, the cost hinges on the activation (0.44). The activation hinges on the weight(s) and bias(es), and on the incoming activations from the previous layer.

How do we use this cost to update parameters?

- This is a lot of math. Why are we doing this again?

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$

Is it a cancer cell?

0 = no
1 = yes

$$MSE = \frac{1}{2}(1-0.44)^2 = 0.1568$$

$$\boxed{\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}}}$$

We can calculate the partial derivative of the cost with respect to:
-the bias of the final neuron
-the weight of the final neuron
-the activation of the previous layer that the final neuron takes in

How do we use this cost to update parameters?

- This is a lot of math. Why are we doing this again?

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$

Is it a cancer cell?

0 = no
1 = yes

$$MSE = \frac{1}{2}(1-0.44)^2 = 0.1568$$

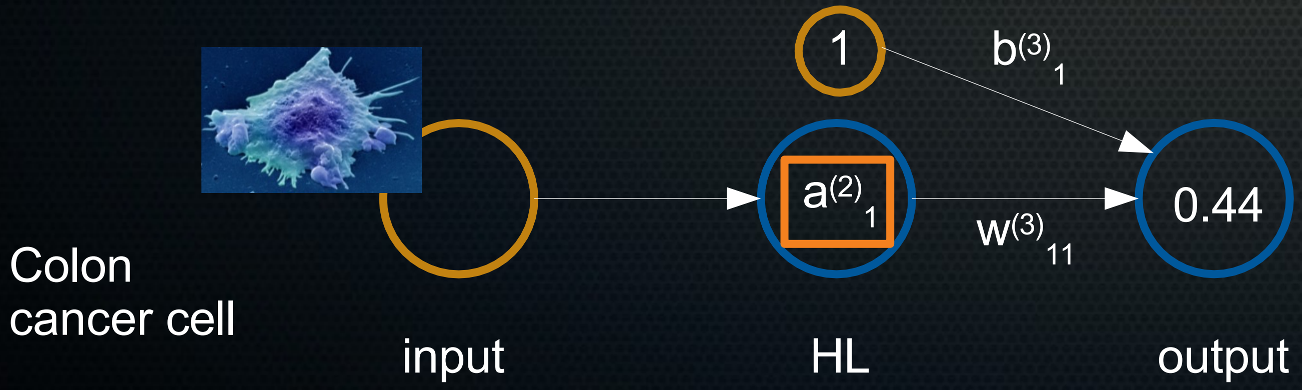
$$\boxed{\frac{\partial C}{\partial w_{jk}^{(l)}} = \frac{\partial C}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}}}$$

We can calculate the partial derivative of the cost with respect to:
-the bias of the final neuron
-the weight of the final neuron
-the activation of the previous layer that the final neuron takes in

How do we use this cost to update parameters?

- This is a lot of math. Why are we doing this again?

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$



Is it a cancer cell?

0 = no
1 = yes

$$MSE = \frac{1}{2}(1-0.44)^2 = 0.1568$$

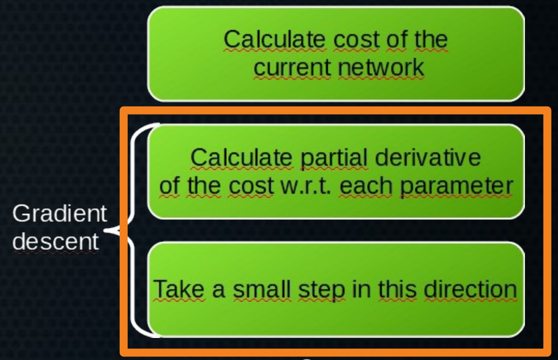
$$\boxed{\frac{\partial C}{\partial a_j^{(l-1)}} = \frac{\partial C}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial a_j^{(l-1)}}}$$

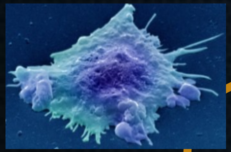
We can calculate the partial derivative of the cost with respect to:
-the bias of the final neuron
-the weight of the final neuron
-the activation of the previous layer that the final neuron takes in

How do we use this cost to update parameters?

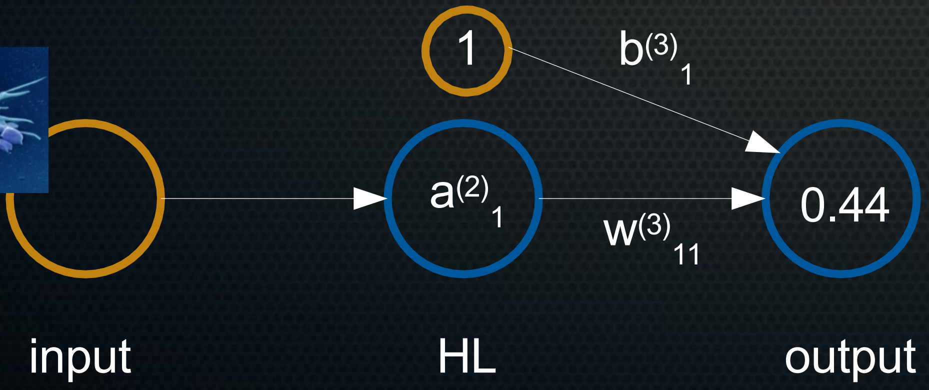
- This is a lot of math. Why are we doing this again?

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$





Colon cancer cell



Is it a cancer cell?

0 = no
1 = yes

$$MSE = \frac{1}{2}(1-0.44)^2 = 0.1568$$

$$\frac{\partial C}{\partial a_j^{(l-1)}} = \frac{\partial C}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial a_j^{(l-1)}}$$

We can calculate the partial derivative of the cost with respect to:

-the bias of the final neuron

-the weight of the final neuron

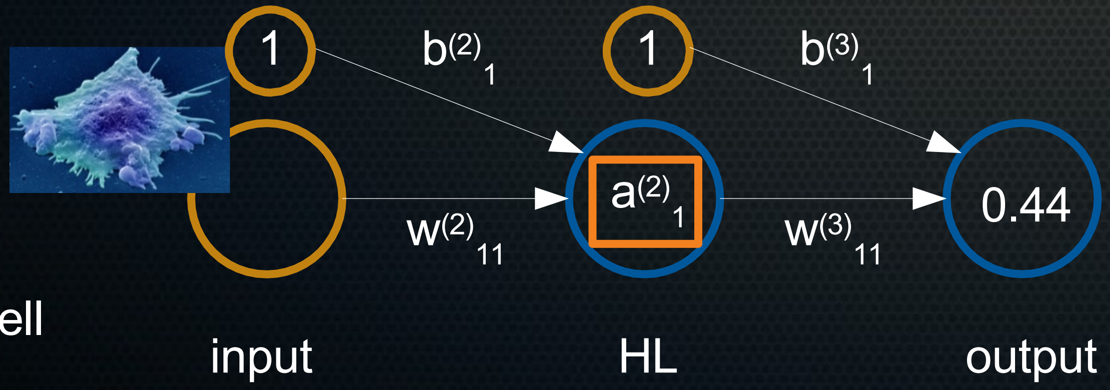
Here we can take a small step in the direction that minimises it

-the activation of the previous layer that the final neuron takes in

How do we use this cost to update parameters?

- This is a lot of math. Why are we doing this again?

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$



Colon cancer cell

Is it a cancer cell?

0 = no
1 = yes

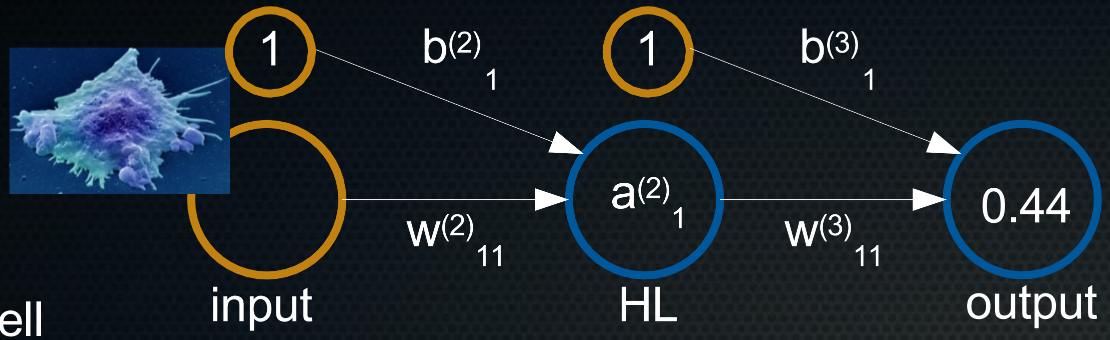
$$MSE = \frac{1}{2}(1-0.44)^2 = 0.1568$$

This term we can propagate back!

We know *that* neuron's activation hinges on *its* weights and biases and the previous layer's activation.

We can calculate the partial derivative of the cost with respect to:
-the bias of the final neuron
-the weight of the final neuron
-the activation of the previous layer that the final neuron takes in

How do we use this cost to update parameters?



Colon
cancer cell

Is it a
cancer
cell?

0 = no
1 = yes

$$MSE = \frac{1}{2}(1-0.44)^2 = 0.1568$$

We can calculate the partial derivative of the cost with respect to:

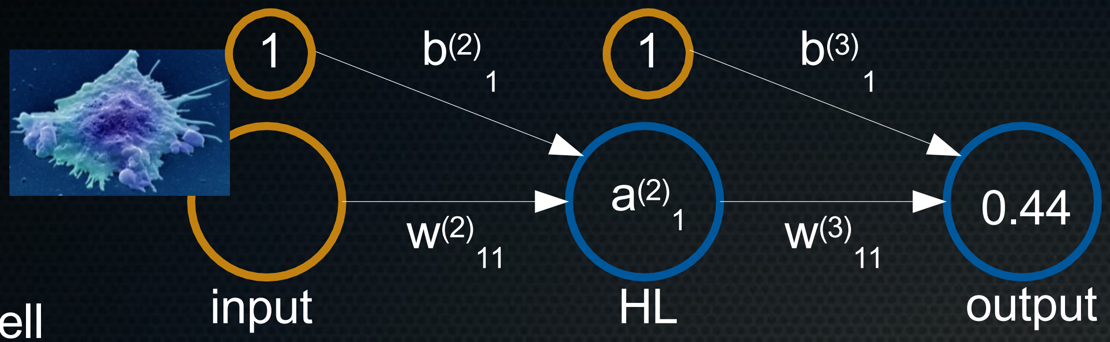
-the bias of the HL neuron

-the weight of the HL neuron

~~-the activation of the previous layer that the HL neuron takes in (but we don't do this for the input!)~~

$$\frac{\partial C}{\partial w_{11}^{(2)}} = \frac{\partial C}{\partial a_1^{(3)}} \cdot \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \cdot \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}}$$

How do we use this cost to update parameters?



Colon
cancer cell

Is it a
cancer
cell?

0 = no
1 = yes

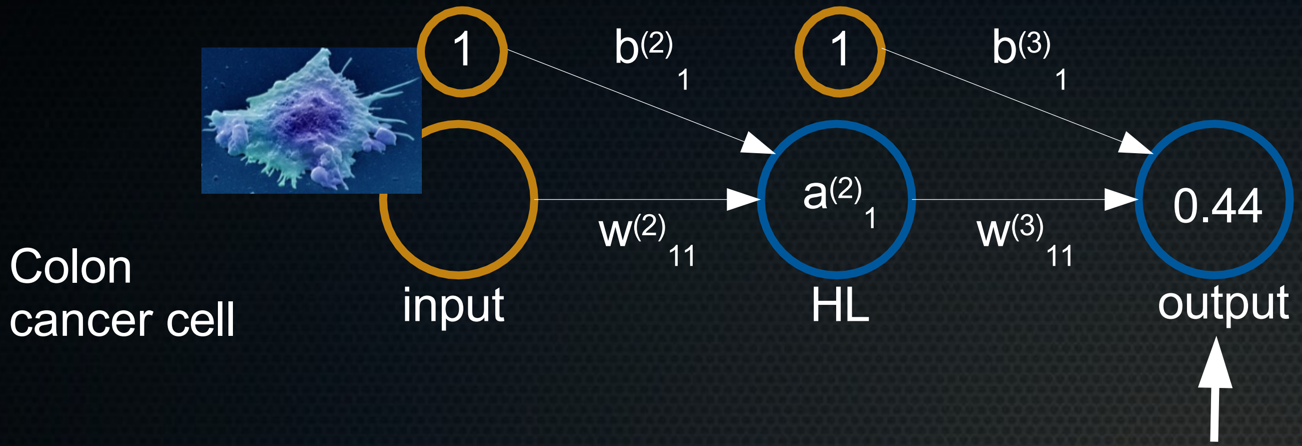
$$MSE = \frac{1}{2}(1-0.44)^2 = 0.1568$$

We can calculate the partial derivative of the cost with respect to:
-the bias of the HL neuron
-the weight of the HL neuron

$$\boxed{\frac{\partial C}{\partial w_{11}^{(2)}} = \frac{\partial C}{\partial a_1^{(3)}} \cdot \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \cdot \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}}}$$

In words: how the total cost depends on the weight of the HL neuron is:
-how the cost depends on the activation of the final layer (0.44), times
-how the activation of the final layer depends on the weighted sum + bias ($z^{(3)}_1$), times
-how that weighted sum depends on the activation of the HL ($a^{(2)}_1$), times
-how the activation of the HL depends on the weighted sum + bias ($z^{(2)}_1$), times
-how that weighted sum depends on the weight from the input.

How do we use this cost to update parameters?



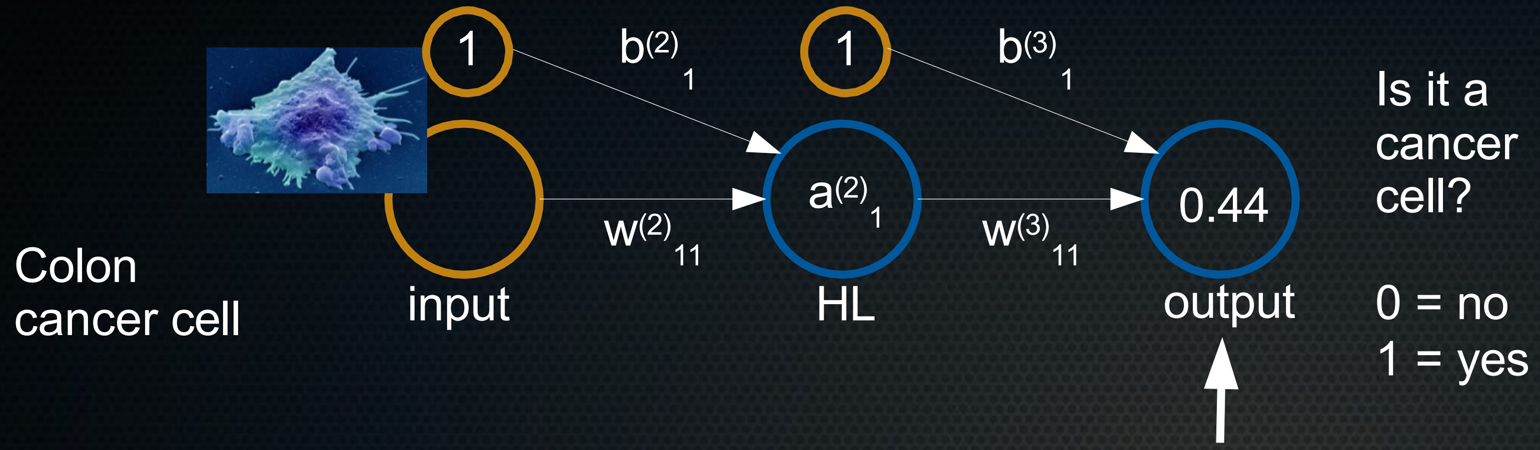
Is it a cancer cell?

0 = no
1 = yes

$$MSE = \frac{1}{2}(1-0.44)^2 = 0.1568$$

- It might seem complex, but the core idea is simple:
  - We can calculate the cost in the final layer
  - We can calculate how that cost hinges on the parameters of that layer
  - → remember, for gradient descent we want to take a small step in *every* parameter of the network to decrease the cost and make it perform better.

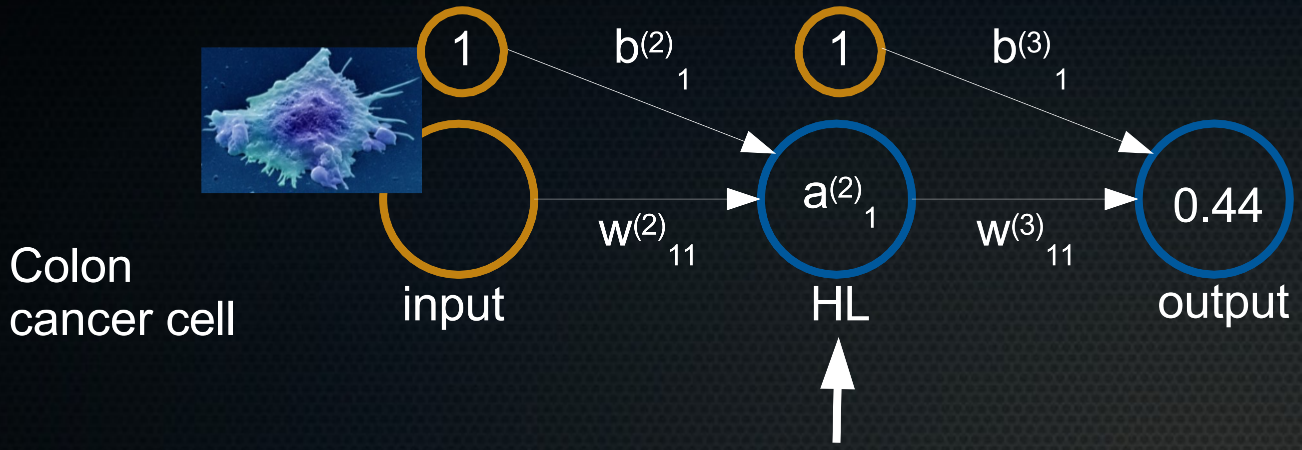
How do we use this cost to update parameters?



$$MSE = \frac{1}{2}(1-0.44)^2 = 0.1568$$

- It might seem complex, but the core idea is simple:
  - We can calculate the cost in the final layer
  - We can calculate how that cost hinges on the parameters of that layer
  - We can also calculate how the cost hinges on the activations of the previous layer

How do we use this cost to update parameters?



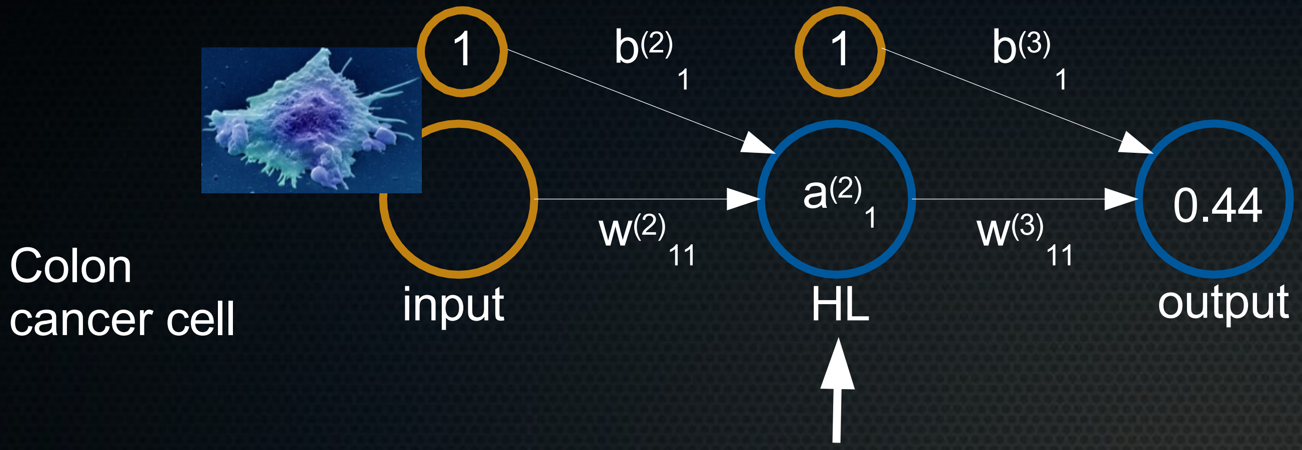
Is it a cancer cell?

0 = no
1 = yes

$$MSE = \frac{1}{2}(1-0.44)^2 = 0.1568$$

- It might seem complex, but the core idea is simple:
  - We can calculate the cost in the final layer
  - We can calculate how that cost hinges on the parameters of that layer
  - We can also calculate how the cost hinges on the activations of the previous layer
  - We can use *that* to go one step back: once we know how to decrease the cost for the activations of the previous layer, we can figure out how to tweak the parameters there to make that activation better.

How do we use this cost to update parameters?



Is it a cancer cell?

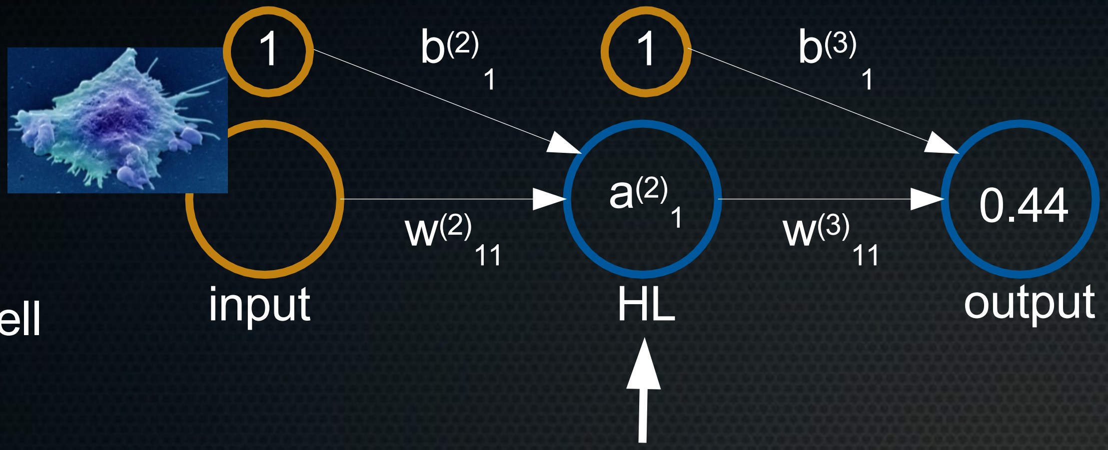
0 = no
1 = yes

$$MSE = \frac{1}{2}(1-0.44)^2 = 0.1568$$

- It might seem complex, but the core idea is simple:
  - We can calculate the cost in the final layer
  - We can calculate how that cost hinges on the parameters of that layer
  - We can also calculate how the cost hinges on the activations of the previous layer
  - We can use *that* to go one step back: once we know how to decrease the cost for the activations of the previous layer, we can figure out how to tweak the parameters there to make that activation better.
  - For this toy network, it ends there, at the input. With more layers, you can keep on chaining derivatives until you know how to change every network parameter for the better!

How do we use this cost to update parameters?

Colon cancer cell

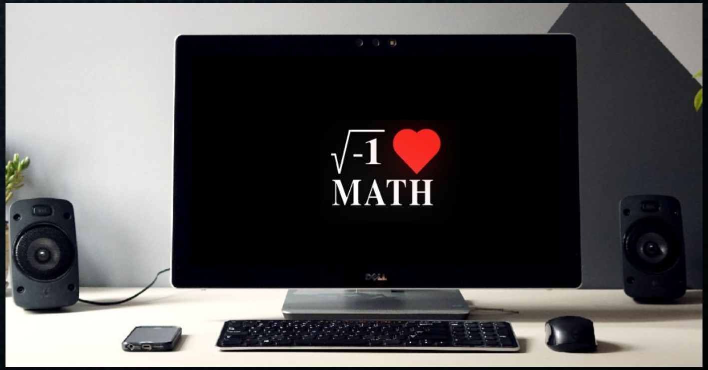


Is it a cancer cell?

0 = no
1 = yes

$$MSE = \frac{1}{2}(1-0.44)^2 = 0.1568$$

- It might seem complex, but the core idea is simple.
- It‘s a lot of number-crunching, but computers love that!



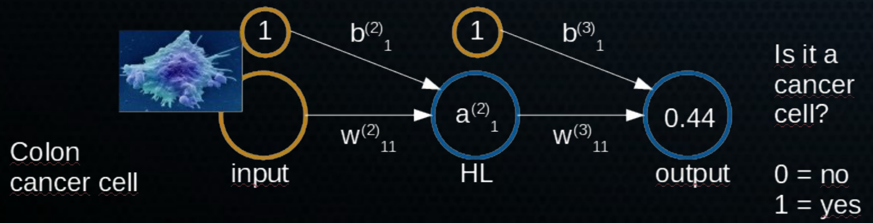
How do we use this cost to update parameters?

- We have almost all the ingredients:

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$

- What we still need is how the cost hinges on the activation, and how the activation hinges on its inputs (it was hidden in the left term):

$$\frac{\partial C}{\partial w_{11}^{(2)}} = \boxed{\frac{\partial C}{\partial a_1^{(3)}} \cdot \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}} \cdot \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}}$$

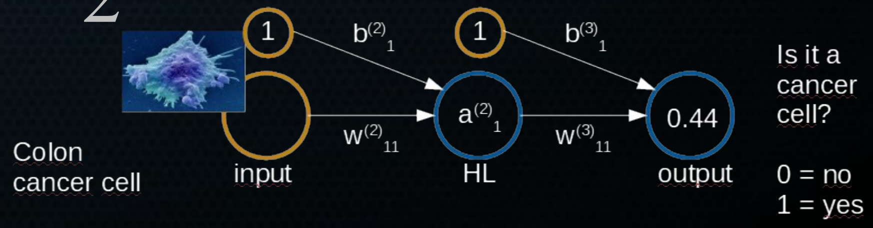


How do we use this cost to update parameters?

- What we still need is how the cost hinges on the activation, and how the activation hinges on its inputs (it was hidden in the left term):

$$\frac{\partial C}{\partial w_{11}^{(2)}} = \boxed{\frac{\partial C}{\partial a_1^{(3)}}} \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \cdot \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}} \qquad C = \frac{1}{2}(y - a_1^{(L)})^2$$

$$C = \frac{1}{2}(1-0.44)^2 = 0.1568$$



How do we use this cost to update parameters?

- What we still need is how the cost hinges on the activation, and how the activation hinges on its inputs (it was hidden in the left term):

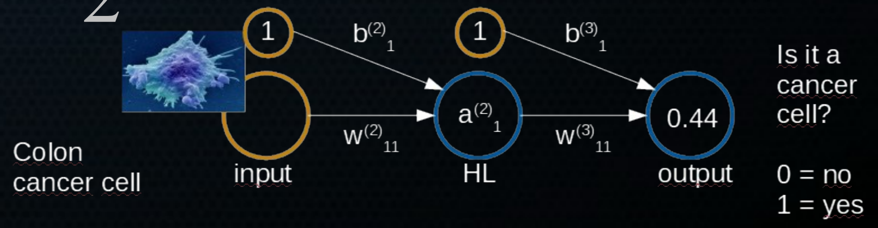
$$\frac{\partial C}{\partial w_{11}^{(2)}} = \boxed{\frac{\partial C}{\partial a_1^{(3)}}} \cdot \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \cdot \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}}$$

$$C = \frac{1}{2}\left(y - \boxed{a_1^{(L)}}\right)^2$$

$$C = \frac{1}{2}\left(y - \boxed{sigmoid\left(z_j^{(l)}\right)}\right)^2$$

$$C = \frac{1}{2}\left(y - \boxed{sigmoid\left(w_{11}^{(3)} \cdot a_1^{(2)} + b_1^{(3)}\right)}\right)^2$$

$$C = \frac{1}{2}(1 - 0.44)^2 = 0.1568$$



How do we use this cost to update parameters?

- What we still need is how the cost hinges on the activation, and how the activation hinges on its inputs (it was hidden in the left term):

$$\frac{\partial C}{\partial w^{(2)}_{11}} = \frac{\partial C}{\partial a^{(3)}_1} \cdot \frac{\partial a^{(3)}_1}{\partial z^{(3)}_1} \cdot \frac{\partial z^{(3)}_1}{\partial a^{(2)}_1} \cdot \frac{\partial a^{(2)}_1}{\partial z^{(2)}_1} \cdot \frac{\partial z^{(2)}_1}{\partial w^{(2)}_{11}}$$

$$\boxed{C = \frac{1}{2}(y - a^{(L)}_1)^2}$$

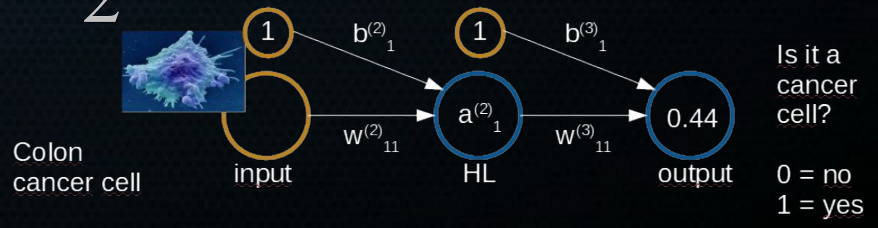
$$C = \frac{1}{2}(y - sigmoid(z^{(l)}_j))^2$$

$$C = \frac{1}{2}(y - sigmoid(w^{(3)}_{11} \cdot a^{(2)}_1 + b^{(3)}_1))^2$$

$$C = \frac{1}{2}(1 - 0.44)^2 = 0.1568$$

Take the derivative

$$\boxed{\frac{\partial C}{\partial a^{(3)}_1} = y - a^{(L)} = 1 - 0.44 = 0.56}$$



How do we use this cost to update parameters?

$$f(g(x)) \qquad \frac{dy}{dx} = f'(g(x)) \times g'(x)$$

- What we still need is how the cost hinges on the activation, and how the activation hinges on its inputs (it was hidden in the left term):

$$\frac{\partial C}{\partial w_{11}^{(2)}} = \frac{\partial C}{\partial a_1^{(3)}} \cdot \boxed{\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}} \cdot \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}}$$

$$C = \frac{1}{2}(y - a_1^{(L)})^2$$

$$C = \frac{1}{2}(y - \boxed{sigmoid(z_j^{(l)})})^2$$

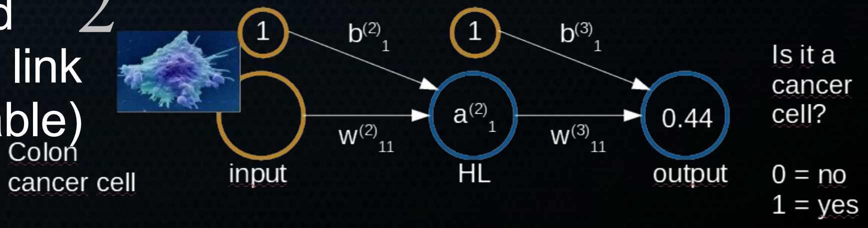
$$C = \frac{1}{2}(y - sigmoid(w_{11}^{(3)} \cdot a_1^{(2)} + b_1^{(3)}))^2$$

$$\frac{\partial C}{\partial a_1^{(3)}} = y - a^{(L)} = 1 - 0.44 = 0.56$$

$$C = \frac{1}{2}(1 - 0.44)^2 = 0.1568$$

$$\boxed{\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = \sigma'(z_1^{(3)}) = \sigma(z_1^{(3)}) \cdot (1 - \sigma(z_1^{(3)}))}$$

Derivative of sigmoid (complex derivation, link with extra info available)



How do we use this cost to update parameters?

- What we still need is how the cost hinges on the activation, and how the activation hinges on its inputs (it was hidden in the left term):

$$\frac{\partial C}{\partial w_{11}^{(2)}} = \frac{\partial C}{\partial a_1^{(3)}} \cdot \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \cdot \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}}$$

$$C = \frac{1}{2}(y - a_1^{(L)})^2$$

$$C = \frac{1}{2}(y - \boxed{sigmoid(z_j^{(l)})})^2$$

$$\frac{\partial C}{\partial a_1^{(3)}} = y - a^{(L)} = 1 - 0.44 = 0.56$$

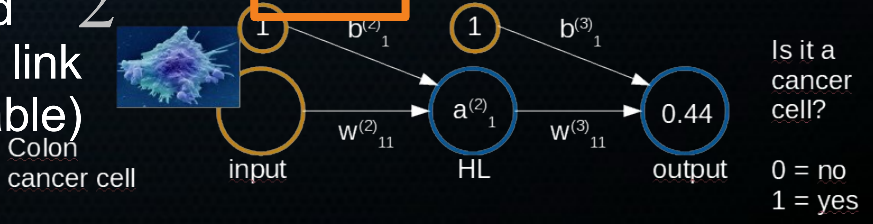
$$C = \frac{1}{2}(y - sigmoid(w_{11}^{(3)} \cdot a_1^{(2)} + b_1^{(3)}))^2$$

$$C = \frac{1}{2}(1 - \boxed{0.44})^2 = 0.1568$$

$$\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = \sigma'(z_1^{(3)}) = \sigma(z_1^{(3)}) \cdot (1 - \sigma(z_1^{(3)}))$$

Derivative of sigmoid (complex derivation, link with extra info available)

$$\boxed{\sigma(-0.241) \approx 0.44}$$



How do we use this cost to update parameters?

- What we still need is how the cost hinges on the activation, and how the activation hinges on its inputs (it was hidden in the left term):

$$\frac{\partial C}{\partial w_{11}^{(2)}} = \frac{\partial C}{\partial a_1^{(3)}} \cdot \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \cdot \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}}$$

$$C = \frac{1}{2}(y - a_1^{(L)})^2$$

$$C = \frac{1}{2}(y - \boxed{sigmoid(z_j^{(l)})})^2$$

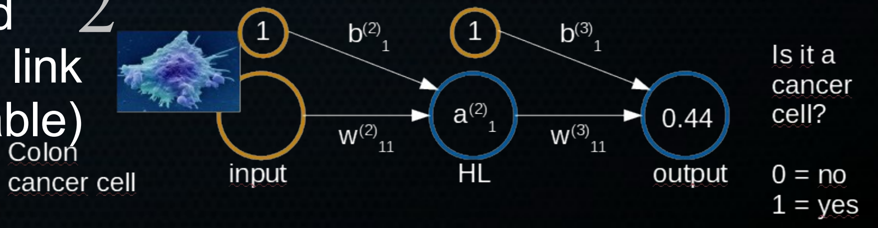
$$C = \frac{1}{2}(y - sigmoid(w_{11}^{(3)} \cdot a_1^{(2)} + b_1^{(3)}))^2$$

$$\frac{\partial C}{\partial a_1^{(3)}} = y - a^{(L)} = 1 - 0.44 = 0.56$$

$$C = \frac{1}{2}(1 - 0.44)^2 = 0.1568$$

$$\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = \sigma'(z_1^{(3)}) = \sigma(z_1^{(3)}) \cdot (1 - \sigma(z_1^{(3)}))$$

Derivative of sigmoid (complex derivation, link with extra info available)



$$\boxed{\sigma'(-0.241) = \sigma(-0.241) \cdot (1 - \sigma(-0.241)) \approx 0.2464}$$

How do we use this cost to update parameters?

- What we still need is how the cost hinges on the activation, and how the activation hinges on its inputs (it was hidden in the left term):

$$\frac{\partial C}{\partial w_{11}^{(2)}} = \frac{\partial C}{\partial a_1^{(3)}} \cdot \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \cdot \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \cdot \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}}$$

$$C = \frac{1}{2}(y - a_1^{(L)})^2$$

$$C = \frac{1}{2}(y - sigmoid(z_j^{(l)}))^2$$

$$\frac{\partial C}{\partial a_1^{(3)}} = y - a^{(L)} = 1 - 0.44 = 0.56$$

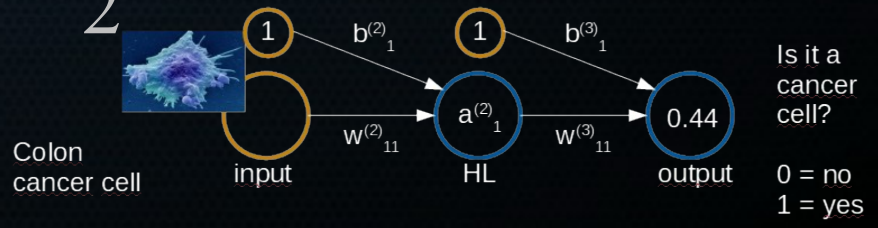
$$C = \frac{1}{2}(y - sigmoid(w_{11}^{(3)} \cdot a_1^{(2)} + b_1^{(3)}))^2$$

$$\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = 0.2464$$

$$C = \frac{1}{2}(1 - 0.44)^2 = 0.1568$$

So we can keep on calculating, calculating partial derivatives for weights and biases in the current layer,
and propagating the error back to get them for previous layers iteratively

$$\sigma(-0.241) \approx 0.44$$



How do we use this cost to update parameters?

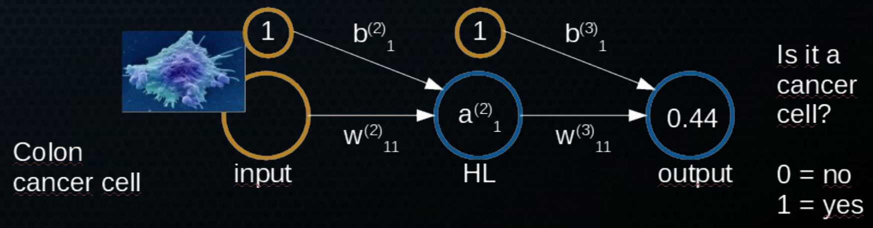
- Taken together:

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$

For the final layer

$$\delta_j^{(L)} = \frac{\partial C}{\partial z_j^{(L)}} = (a_j^{(L)} - y_j)\sigma(z_j^{(L)}) \cdot (1 - \sigma(z_j^{(L)}))^{*}$$

* (For cost function == MSE $(a^{(L)}_j - y_j)$, just $a_j^L - y_j$ for binary cross-entropy



How do we use this cost to update parameters?

- Taken together:

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$

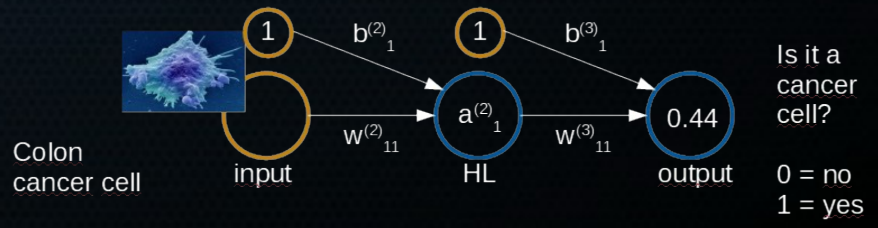
For the final layer

$$\delta_j^{(L)} = \frac{\partial C}{\partial z_j^{(L)}} = (a_j^{(L)} - y_j)\sigma(z_j^{(L)}) \cdot (1 - \sigma(z_j^{(L)}))^{*}$$

For the previous layer (applied iteratively)

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{weights}} w_{kj}^{(l+1)} \cdot \delta_k^{(l+1)} \cdot \sigma'(z_j^{(l)})$$

* (For cost function == MSE $(a^{(L)})_j - y_j)$, slightly more complex for binary crossentropy)



How do we use this cost to update parameters?

- Taken together:

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$

For the final layer

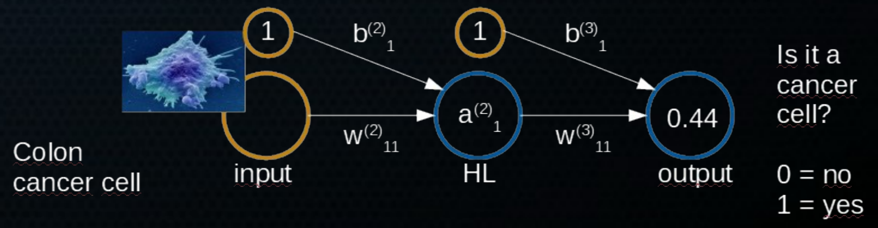
For the previous layer (applied iteratively)

$$\delta_j^{(L)} = \frac{\partial C}{\partial z_j^{(L)}} = (a_j^{(L)} - y_j)\,\sigma(z_j^{(L)}) \cdot (1 - \sigma(z_j^{(L)}))^{*}$$

Moves the error from the inputs of neurons in layer l+1 to the outputs of a neuron in the previous layer

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{weights}} \overbrace{w_{kj}^{(l+1)} \cdot \delta_k^{(l+1)}} \cdot \sigma'(z_j^{(l)})$$

* (For cost function == MSE $(a^{(L)}_j - y_j)$, slightly more complex for binary crossentropy)



How do we use this cost to update parameters?

- Taken together:

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \Bigg| \quad \frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \Bigg| \quad \frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}$$

For the final layer

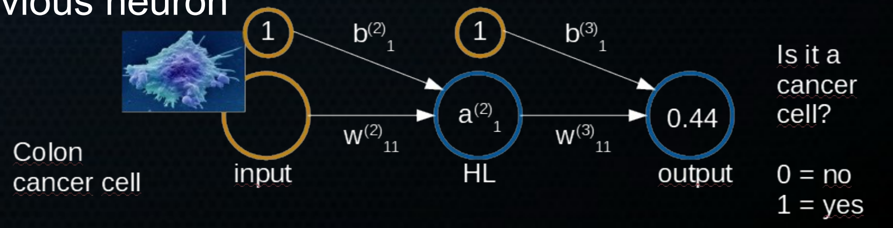
For the previous layer (applied iteratively)

$$\delta_j^{(L)} = \frac{\partial C}{\partial z_j^{(L)}} = \left(a_j^{(L)} - y_j\right)\sigma\left(z_j^{(L)}\right) \cdot \left(1 - \sigma\left(z_j^{(L)}\right)\right)^{*}$$

Moves error back through the activation function, so that we can calculate the partial derivative w.r.t. weights, bias, and input of previous neuron

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{weights}} w_{kj}^{(l+1)} \cdot \delta_k^{(l+1)} \cdot \overbrace{\sigma'\left(z_j^{(l)}\right)}$$

* (For cost function == MSE $(a^{(L)}_j – y_j)$, slightly more complex for binary crossentropy)



How do we use this cost to update parameters?

- Taken together:

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} \quad \left| \quad \boxed{\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)}} \quad \right| \quad \boxed{\frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \cdot a^{(l-1)}}$$

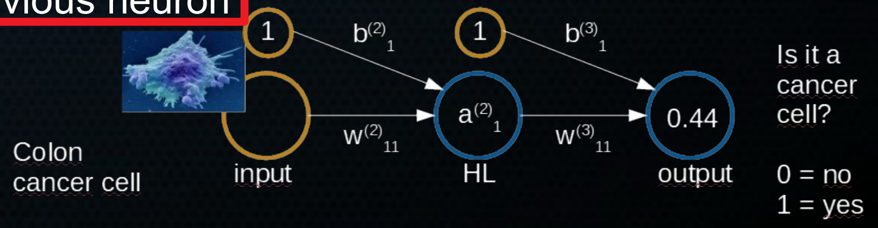
For the final layer

$$\delta_j^{(L)} = \frac{\partial C}{\partial z_j^{(L)}} = (a_j^{(L)} - y_j)\sigma(z_j^{(L)}) \cdot (1 - \sigma(z_j^{(L)}))$$

For the previous layer (applied iteratively)

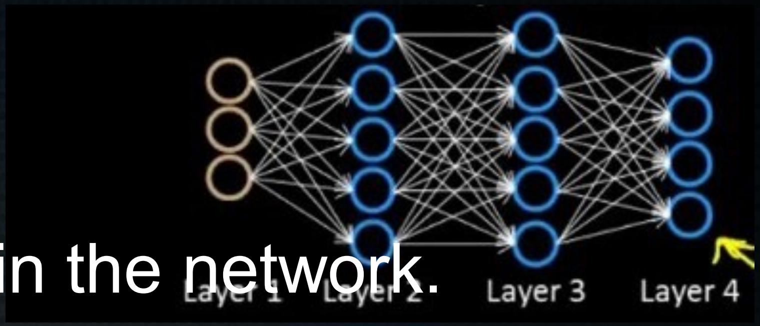
Moves error back through the activation function, so that we can calculate the partial derivative w.r.t. weights, bias, and input of previous neuron

$$\boxed{\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{weights}} w_{kj}^{(l+1)} \cdot \delta_k^{(l+1)} \cdot \overbrace{\sigma'(z_j^{(l)})}}$$



How do we use this cost to update parameters?

- The procedure we just discussed leads to calculating the partial derivatives of the cost function with respect to all the weights and biases in the network.

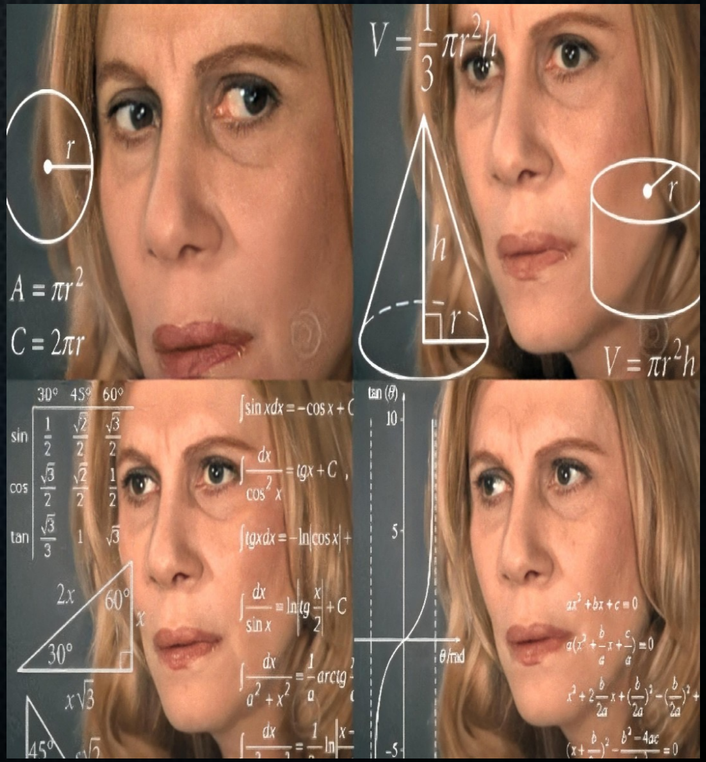


And now?

- Most equations I gave here were not vectorised, but they can all be vectorised. That makes them fast(er) to compute.
- In the example here, we had one training sample. In reality you have more, so you compute an average cost over all training samples and *then* update the network using backpropagation.

And now?

- You might feel like the lady in this distinctly non-spicy stale meme, though without trigonometrical or geometric concerns.
- I for sure did not get this quickly. It’s not simple!
- Modern libraries do all the heavy lifting for you.



And now?

- Let this sink in, and continue where you left off yesterday