Deep Learning

Tristan Cazenave

Tristan.Cazenave@dauphine.psl.eu

Backpropagation

A spectrum of machine learning tasks

Typical Statistics------------Artificial Intelligence

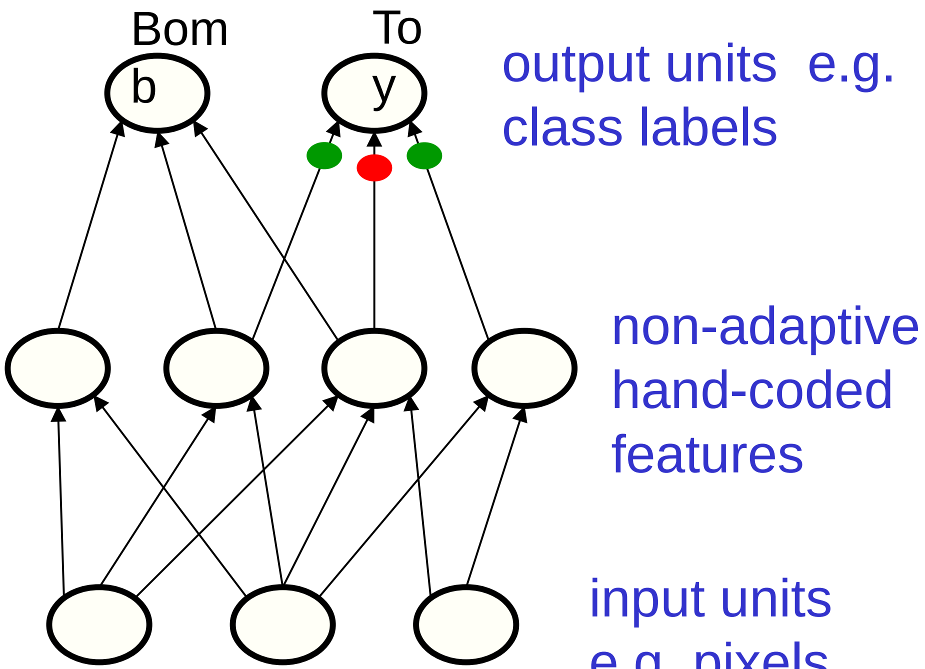
- Low-dimensional data (e.g. less than 100 dimensions)
- Lots of noise in the data
- There is not much structure in the data, and what structure there is, can be represented by a fairly simple model.
- The main problem is distinguishing true structure from noise.

- High-dimensional data (e.g. more than 100 dimensions)
- The noise is not sufficient to obscure the structure in the data if we process it right.
- There is a huge amount of structure in the data, but the structure is too complicated to be represented by a simple model.
- The main problem is figuring out a way to represent the complicated structure so that it can be learned.

Historical background:

First generation neural networks

- Perceptrons (~1960) used a layer of hand-coded features and tried to recognize objects by learning how to weight these features.
  - There was a neat learning algorithm for adjusting the weights.
  - But perceptrons are fundamentally limited in what they can learn to do.

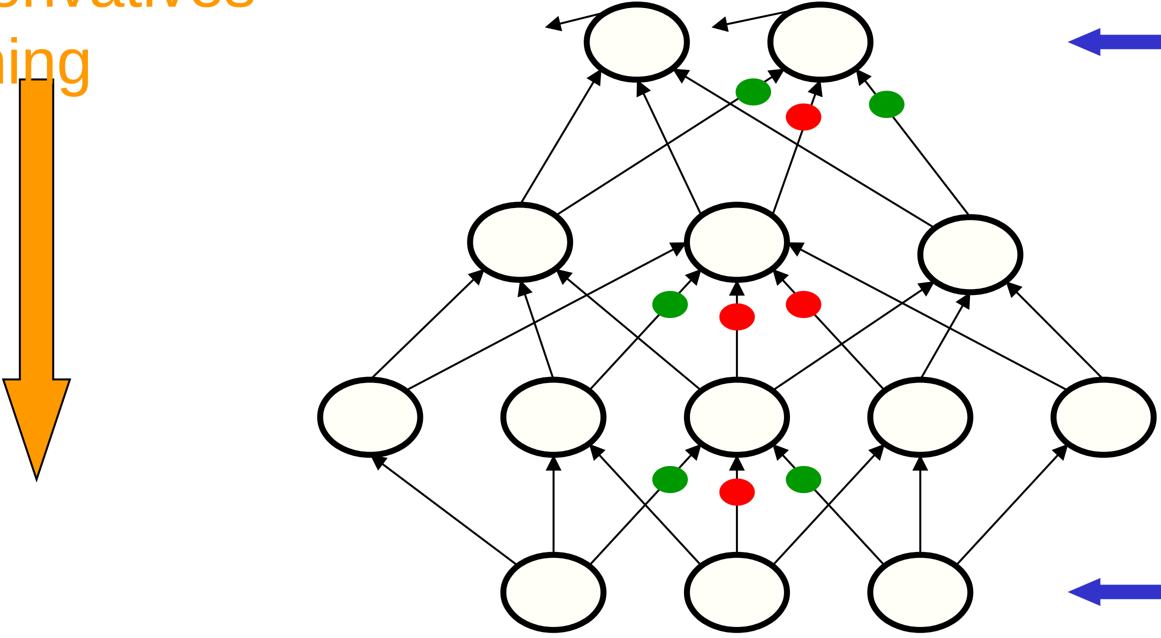

Bom
To
b
y
output units e.g. class labels
non-adaptive hand-coded features
input units e.g. pixels

Sketch of a typical perceptron from the 1960's

Second generation neural networks (~1985)

Compare outputs with correct answer to get error signal

Back-propagate error signal to get derivatives for learning



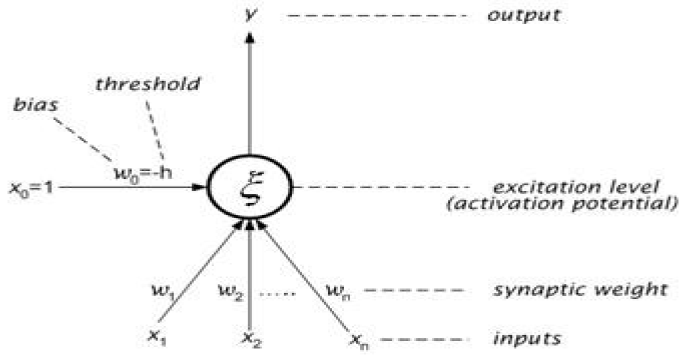
outputs

hidden layers

input vector

Formalization of a neural network

- Each neuron computes its output y as a linear combinations of its inputs x_i followed by an activation function :



Formalization of a neural network

- Linear combination : $\xi = \sum_{i=1}^{n} w_i x_i$

- Activation function :

$$y = \sigma(\xi) = \begin{cases} 1 \; if \; \xi \geq 0 \\ 0 \; if \; \xi < 0 \end{cases} \; where \; \xi = \sum_{i=0}^{n} w_i x_i$$

Formalization of a neural network

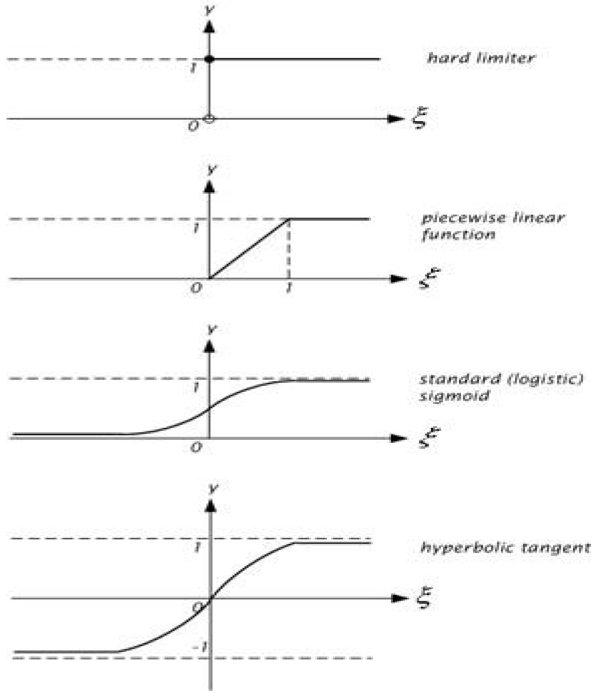
- Activation functions :

$$\sigma(\xi) = \begin{cases} 1 \; if \; \xi \geq 0 \\ 0 \; if \; \xi < 0 \end{cases}$$

$$\sigma(\xi) = \begin{cases} 1 & \xi > 1 \\ \xi & 0 \leq \xi \leq 1 \\ 0 & \xi < 0 \end{cases}$$

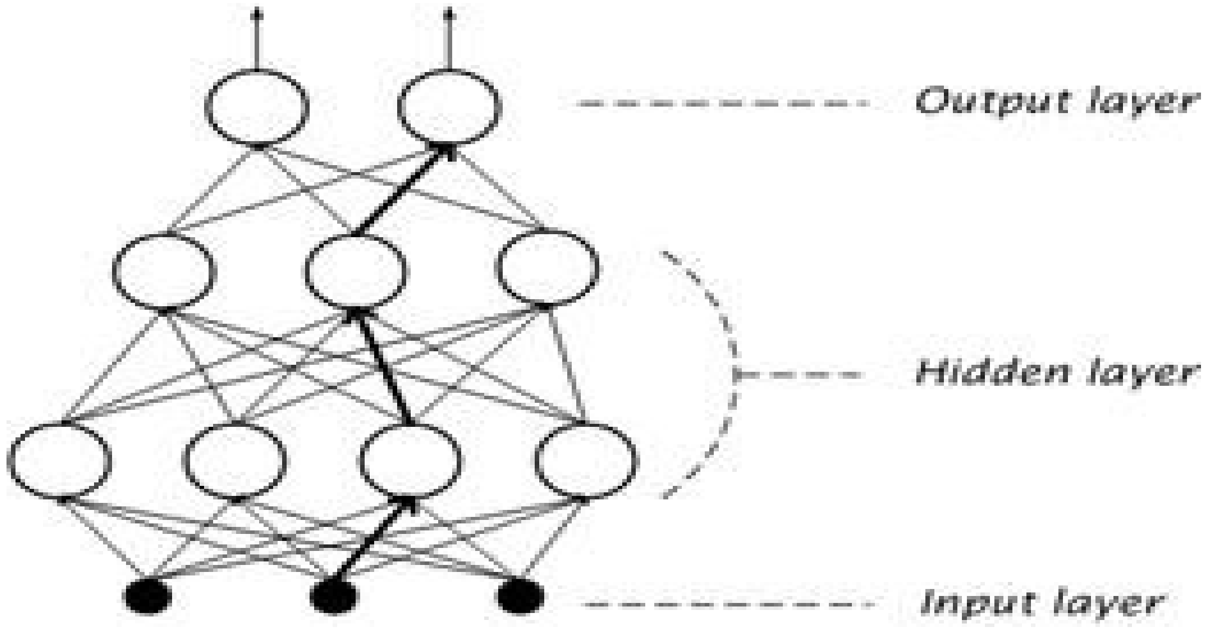
$$\sigma(\xi) = \frac{1}{1 + e^{-\xi}}$$

$$\sigma(\xi) = \tanh\left(\frac{1}{2}\xi\right) = \frac{1 - e^{-\xi}}{1 + e^{-\xi}}$$



Formalization of a neural network

- Architecture :

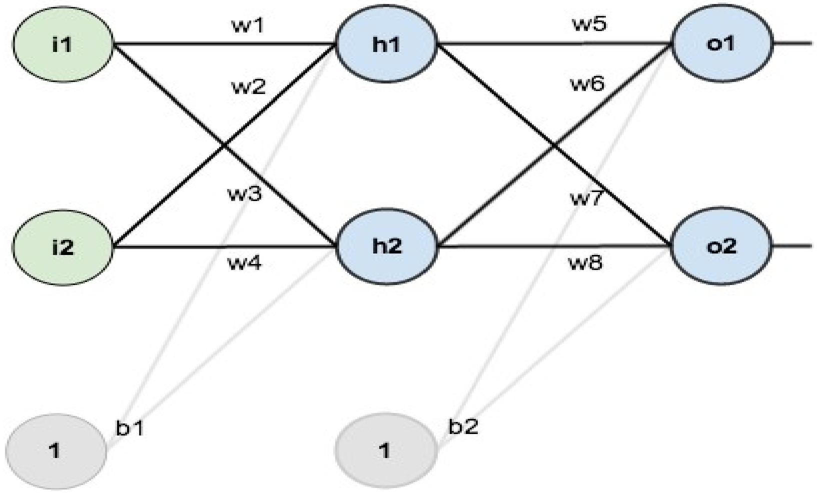


Backpropagation

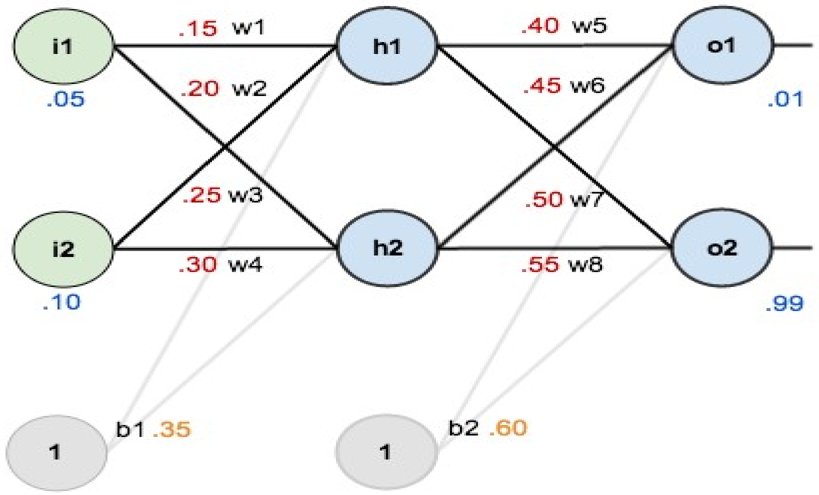
- We are going to explain backpropagation on a simple example.

- We take as example a network with two inputs, two outputs and two hidden neurons .

Backpropagation



Backpropagation



Backpropagation

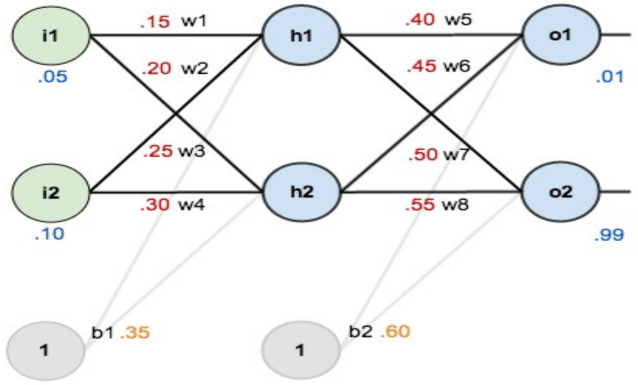
- The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.

- We are going to work with a single training set: given inputs 0.05 and 0.10, we want the neural network to output 0.01 and 0.99.

The Forward Pass

- We figure out the total net input to each hidden layer neuron.
- Squash the total net input using an activation function (here we use the sigmoid function).
- Repeat the process with the output layer neurons.

The Forward Pass



- Here's how we calculate the total net input for h1:

  $net_{h1} = w1 * i1 + w2 * i2 + b1 * 1$

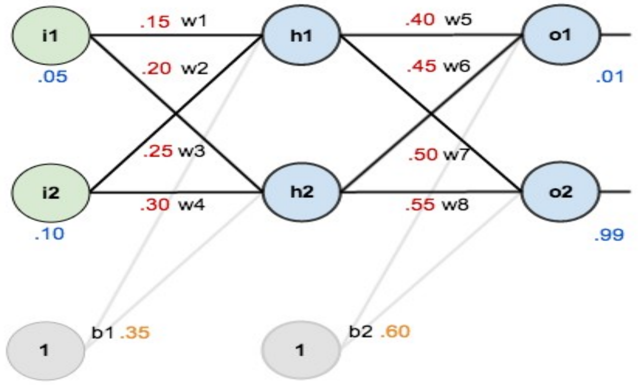
  $net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$

- We then squash it using the sigmoid function to get the output of h1 :

  $out_{h1} = 1/(1+e\text{^}-net_{h1}) = 1/(1+e\text{^}-0.3775) = 0.593269992$

- Carrying out the same process for h2 we get : $out_{h2} = 0.596884378$

The Forward Pass



- We repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.
- Here's the output for o1: net_{o1} = w5 * out_{h1} + w6 * out_{h2} + b2 * 1

  net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967

  out_{o1} = 1/(1+e^-net_{o1}) = 1/(1+e^-1.105905967) = 0.75136507
- And carrying out the same process for o2 we get: out_{o2} = 0.772928465

The Error

- We can now calculate the error for each output neuron using the squared error function and sum them to get the total error:

  $E_{total} = \Sigma\ 1/2\ (\text{target} – \text{output})^2$

- For example, the target output for o1 is 0.01 but the neural network output 0.75136507, therefore its error is:

  $E_{o1} = 1/2\ (\text{target}_{o1} - \text{out}_{o1})^2 = 1/2\ (0.01 - 0.75136507)^2 = 0.274811083$

- Repeating this process for o2 we get: $E_{o2} = 0.023560026$
- The total error for the neural network is the sum of these errors:

  $E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$

The Backwards Pass

- Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

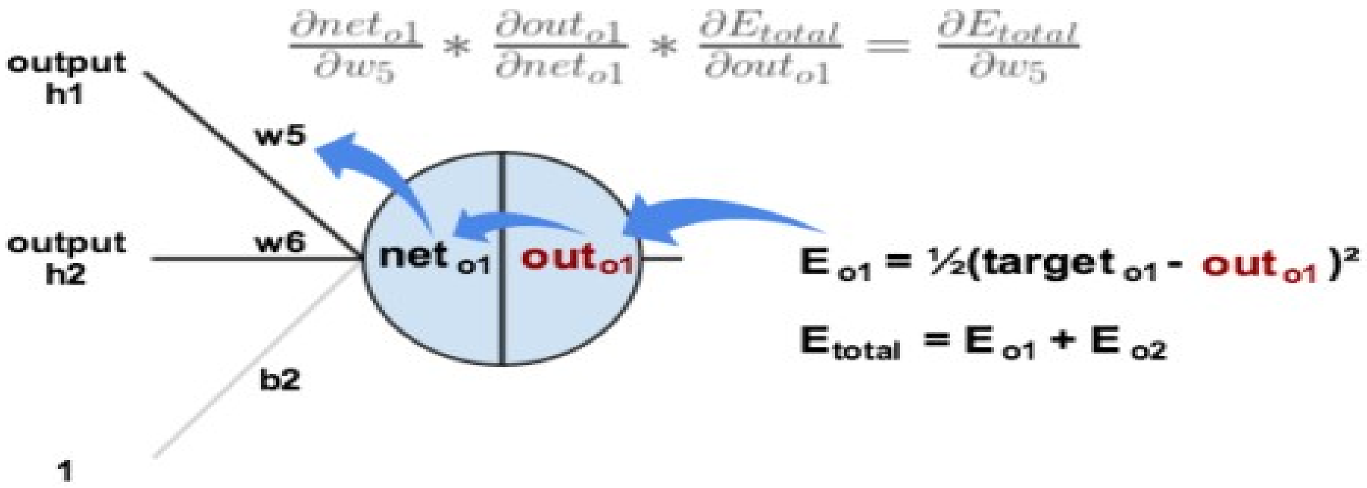
Output Layer

- Consider w5.
- We want to know how much a change in w5 affects the total error:

  $\delta E_{total} / \delta w5$
- $\delta E_{total} / \delta w5$ is read as “the partial derivative of E_{total} with respect to w5“.
- You can also say “the gradient with respect to w5“.
- By applying the chain rule we know that:

  $\delta E_{total} / \delta w5 = \delta E_{total} / \delta out_{o1} * \delta out_{o1} / \delta net_{o1} * \delta net_{o1} / \delta w5$

Output Layer



$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2$$

$$E_{total} = E_{o1} + E_{o2}$$

Output Layer

- We need to figure out each piece in this equation.
- First, how much does the total error change with respect to the output?

$E_{total} = 1/2\ (target_{o1} - out_{o1})^2 + 1/2\ (target_{o2} - out_{o2})^2$

$\delta E_{total} / \delta out_{o1} = -(target_{o1} - out_{o1})$

$\delta E_{total} / \delta out_{o1} = -(0.01 - 0.75136507) = 0.74136507$

Output Layer

- Next, how much does the output of o1 change with respect to its total net input?

  $out_{o1} = 1/(1+e\hat{}(-net_{o1}))$

  $\delta out_{o1} / \delta net_{o1} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$

- Finally, how much does the total net input of o1 change with respect to w5?

  $net_{o1} = w5 * out_{h1} + w6 * out_{h2} + b2 * 1$

  $\delta net_{o1} / \delta w5 = out_{h1} = 0.593269992$

Output Layer

- Putting it all together:

  $\delta E_{total} / \delta w5 = \delta E_{total} / \delta out_{o1} * \delta out_{o1} / \delta net_{o1} * \delta net_{o1} / \delta w5$

  $\delta E_{total} / \delta w5 = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$

- To decrease the error, we then subtract this value from the current weight (optionally multiplied by some learning rate, η, which we'll set to 0.5):

  $w5 = w5 - \eta * \delta E_{total} / \delta w5 = 0.4 - 0.5 * 0.082167041 = 0.35891648$

Output Layer

- We can repeat this process to get the new weights w6, w7, and w8:

  w6 = 0.408666186

  w7 = 0.511301270

  w8 = 0.561370121

- We perform the actual updates in the neural network after we have the new weights leading into the hidden layer neurons (ie, we use the original weights, not the updated weights, when we continue the backpropagation algorithm below).

Hidden Layer

- Next, we’ll continue the backwards pass by calculating new values for w1, w2, w3, and w4.

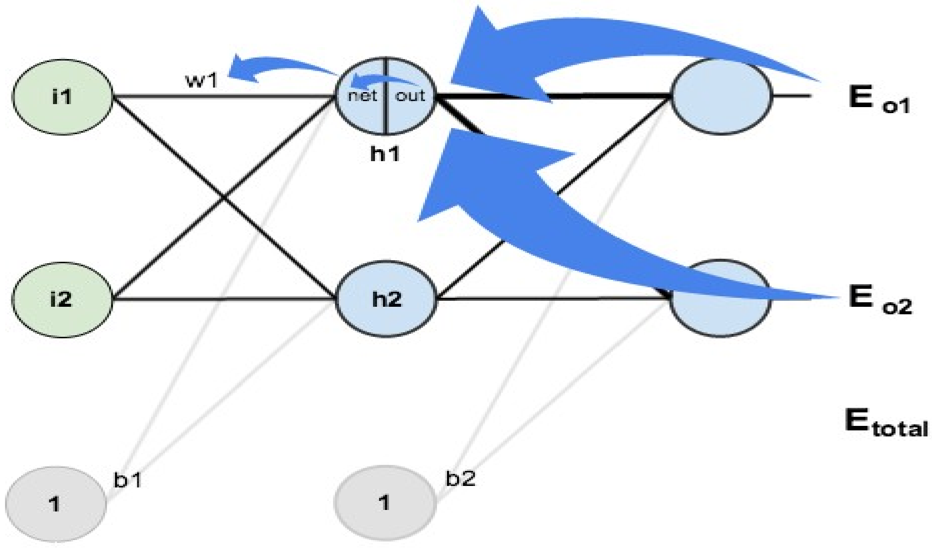
- Big picture, here’s what we need to figure out:

$$\delta E_{total} / \delta w1 = \delta E_{total} / \delta out_{h1} * \delta out_{h1} / \delta net_{h1} * \delta net_{h1} / \delta w1$$

Hidden Layer

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



$E_{total} = E_{o1} + E_{o2}$

Hidden Layer

- We’re going to use a similar process as we did for the output layer, but slightly different to account for the fact that the output of each hidden layer neuron contributes to the output (and therefore error) of multiple output neurons.
- We know that out_{h1} affects both out_{o1} and out_{o2} therefore the $\delta E_{total} / \delta out_{h1}$ needs to take into consideration its effect on the both output neurons:

$$\delta E_{total} / \delta out_{h1} = \delta E_{o1} / \delta out_{h1} + \delta E_{o2} / \delta out_{h1}$$

Hidden Layer

- Starting with $\delta E_{o1} / \delta out_{h1}$:

  $\delta E_{o1} / \delta out_{h1} = \delta E_{o1} / \delta net_{o1} * \delta net_{o1} / \delta out_{h1}$

- We can calculate $\delta E_{o1} / \delta net_{o1}$ using values we calculated earlier:

  $\delta E_{o1} / \delta net_{o1} = \delta E_{o1} / \delta out_{o1} * \delta out_{o1} / \delta net_{o1} = 0.74136507 * 0.186815602 = 0.138498562$

- And $\delta net_{o1} / \delta out_{h1}$ is equal to w5:

  $net_{o1} = w5 * out_{h1} + w6 * out_{h2} + b2 * 1$

  $\delta net_{o1} / \delta out_{h1} = w5 = 0.40$

- Plugging them in:

  $\delta E_{o1} / \delta out_{h1} = \delta E_{o1} / \delta net_{o1} * \delta net_{o1} / \delta out_{h1} = 0.138498562 * 0.40 = 0.055399425$

Hidden Layer

- Following the same process for $\delta E_{o2} / \delta out_{h1}$, we get:

  $\delta E_{o2} / \delta out_{h1} = -0.019049119$

- Therefore:

  $\delta E_{total} / \delta out_{h1} = \delta E_{o1} / \delta out_{h1} + \delta E_{o2} / \delta out_{h1} = 0.055399425 + -0.019049119 = 0.036350306$

- Now that we have $\delta E_{total} / \delta out_{h1}$, we need to figure out $\delta out_{h1} / \delta net_{h1}$ and then $\delta net_{h1} / \delta w$ for each weight:

  $out_{h1} = 1/(1+e\text{^}(-net_{h1}))$

  $\delta out_{h1} / \delta net_{h1} = out_{h1} (1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$

Hidden Layer

- We calculate the partial derivative of the total net input to h1 with respect to w1 the same as we did for the output neuron:

  $net_{h1} = w1 * i1 + w2 * i2 + b1 * 1$

  $\delta net_{h1} / \delta w1 = i1 = 0.05$

- Putting it all together:

  $\delta E_{total} / \delta w1 = \delta E_{total} / \delta out_{h1} * \delta out_{h1} / \delta net_{h1} * \delta net_{h1} / \delta w1$

  $\delta E_{total} / \delta w1 = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$

Hidden Layer

- We can now update w1:

  w1 = w1 - η * δE_{total} / δw1 = 0.15 - 0.5 * 0.000438568 = 0.149780716

- Repeating this for w2, w3, and w4 :

  w2 = 0.19956143

  w3 = 0.24975114

  w4 = 0.29950229

Backpropagation

- Finally, we’ve updated all of our weights!
- When we fed forward the 0.05 and 0.1 inputs originally, the error on the network was 0.298371109.
- After this first round of backpropagation, the total error is now down to 0.291027924.
- It might not seem like much, but after repeating this process 10,000 times, for example, the error plummets to 0.000035085.
- At this point, when we feed forward 0.05 and 0.1, the two outputs neurons generate 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target).

Backpropagation

- Network with two inputs, one output, one hidden layer of one neuron.
- Inputs are 0.1 and 0.5, desired output is 0.2.
- Write code for the forward pass.
- Compute the error.
- Backpropagate the error.
- Train the network.

Backpropagation

- Represent the connections between two layers with a matrix of weights.
- Train a network using matrices.