

Tutorial 6

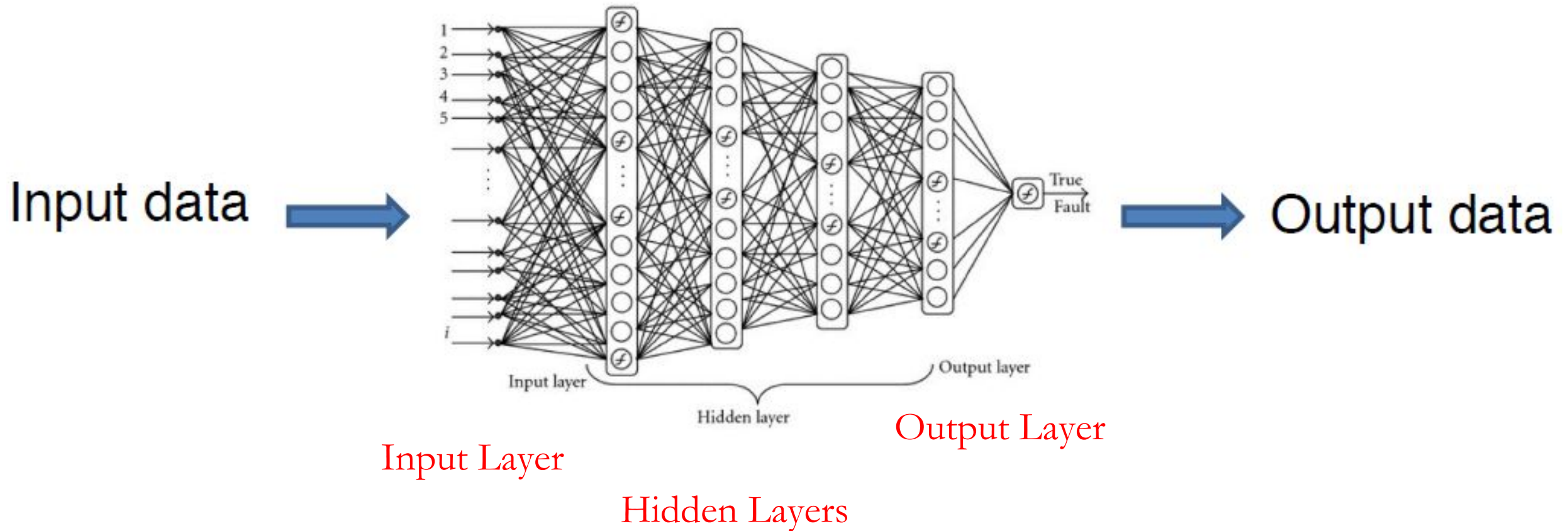
Neural Network: Backpropagation

Agenda

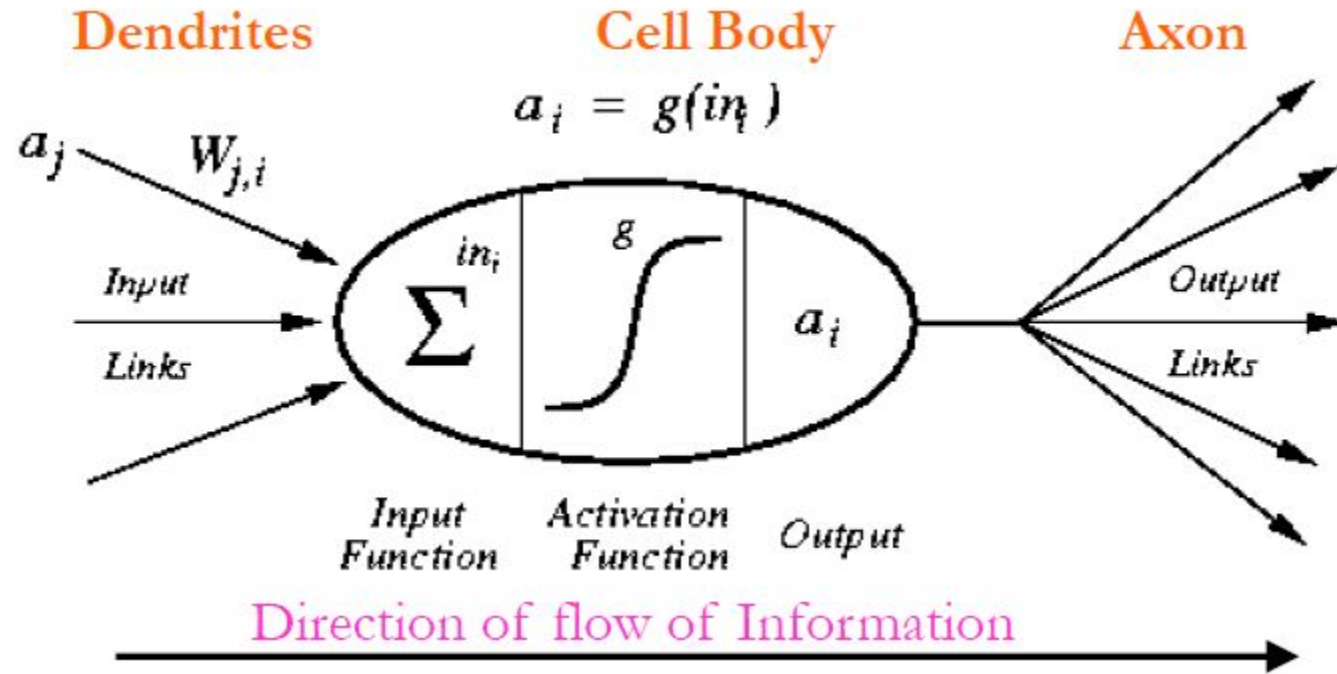
- Understand FeedFoward and Backpropagation
- Discussion about Programming Assignment 5
 - Build Neural Network model
 - Multi-class classification and prediction
- Python Implementation
 - Tensorflow and Keras

Neural Network

NN (perceptron) consists of three layers:



Neuron in Neural Network

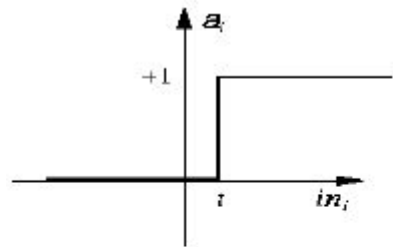


Each unit does a local computation based on inputs from its neighbours & compute a new activation level – sends along each of its output links

- a_j : Activation value of unit j
- $w_{j,i}$: Weight on the link from unit j to unit i
- in_i : Weighted sum of inputs to unit i
- a_i : Activation value of unit i
- g : Activation function.

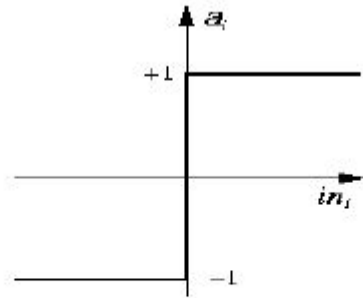
Hidden Layer: Activation Functions

- Common Choices:

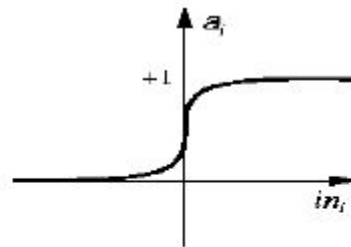


(a) Step function

threshold function



(b) Sign function



(c) Sigmoid function

logistic function

$$\begin{aligned}\text{Step}(x) &= 1 \text{ if } x \geq 0, \text{ else } 0 \\ \text{Sign}(x) &= 1 \text{ if } x \geq 0, \text{ else } -1 \\ \text{Sigmoid}(x) &= 1/(1+e^{-x})\end{aligned}$$

Simply put, activation function calculates a “weighted sum” of its input, and then decides whether it should be “fired” or not

$$Y = \sum (\text{weight} * \text{input}) + \text{bias}$$

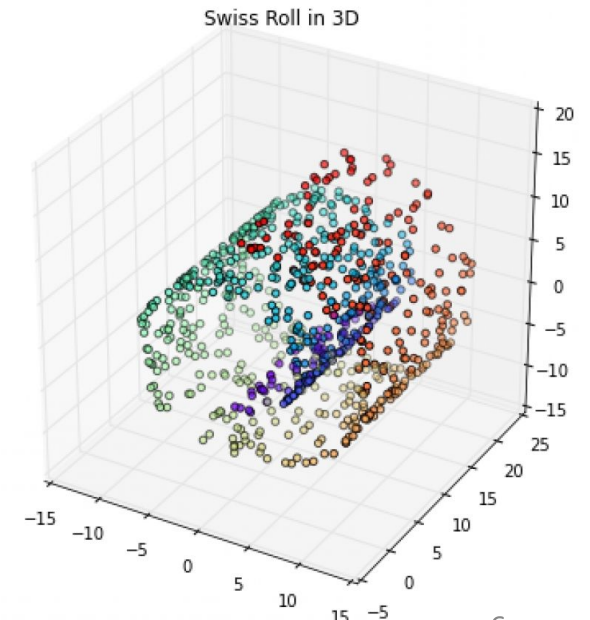
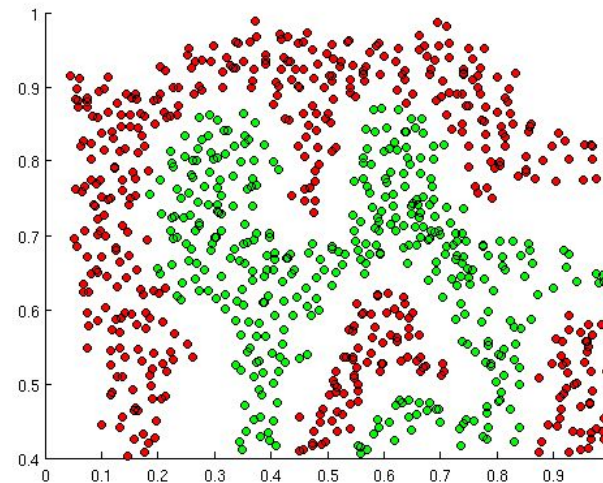
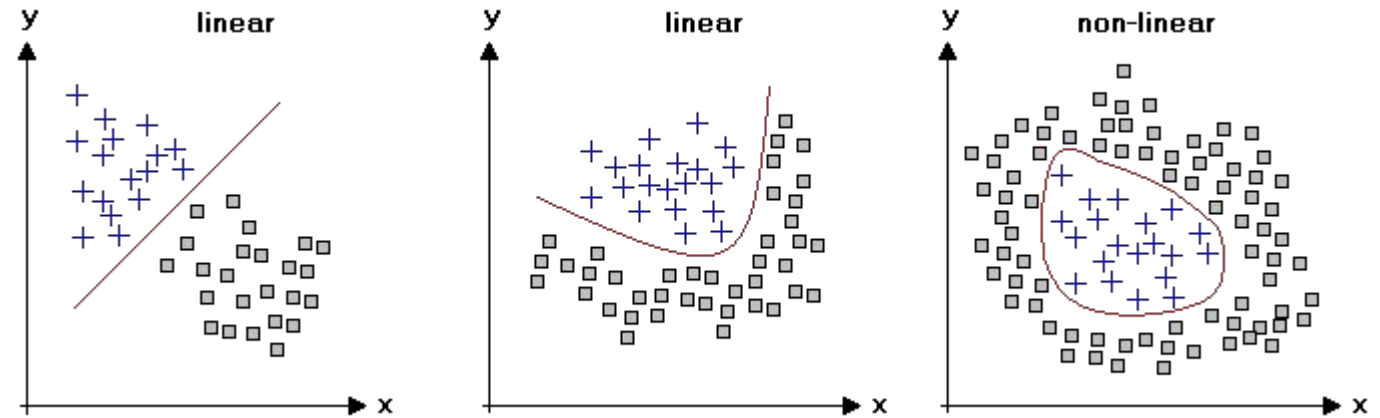
Question: Why do we need it?

Activation functions are really important for NN to learn and make sense of something really **Complicated** and **Non-linear complex functional mappings between the inputs and outputs**. They introduce **non-linear properties** to our NN.

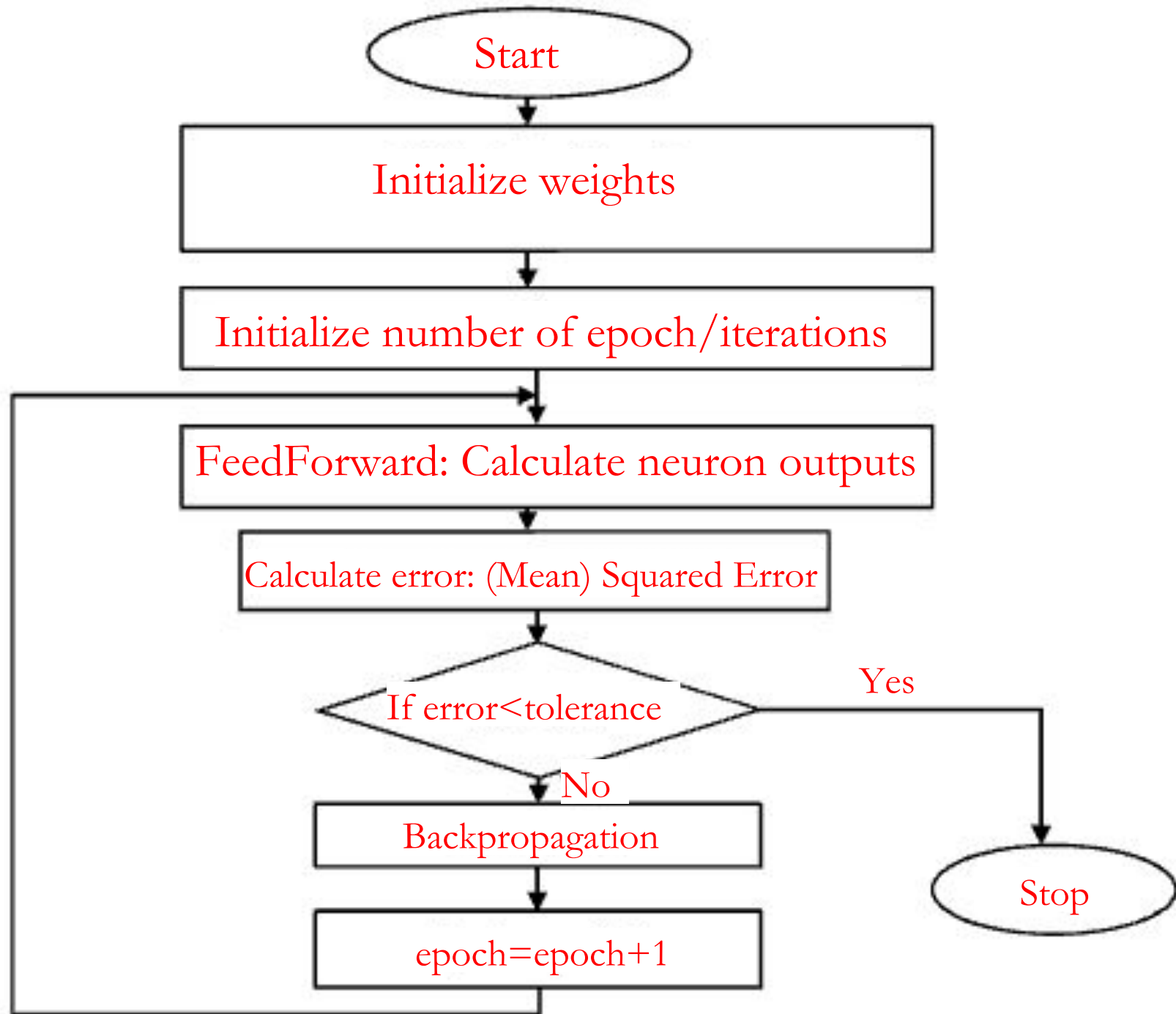
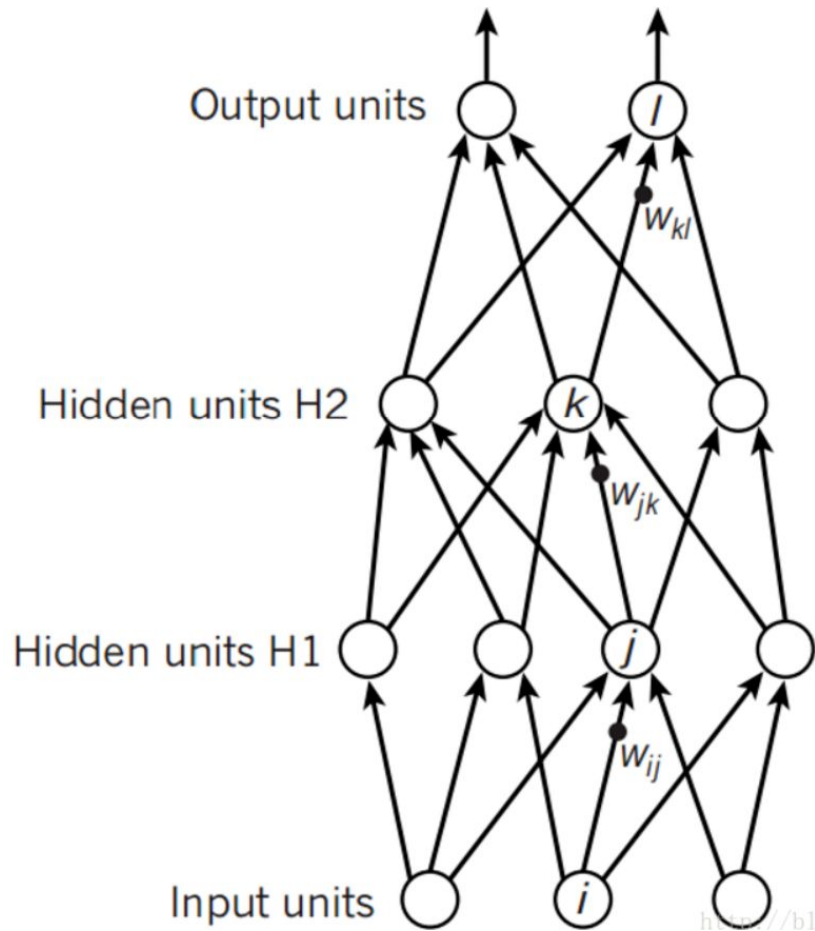
Hidden Layer: Activation Functions

Question: Why do we need it?

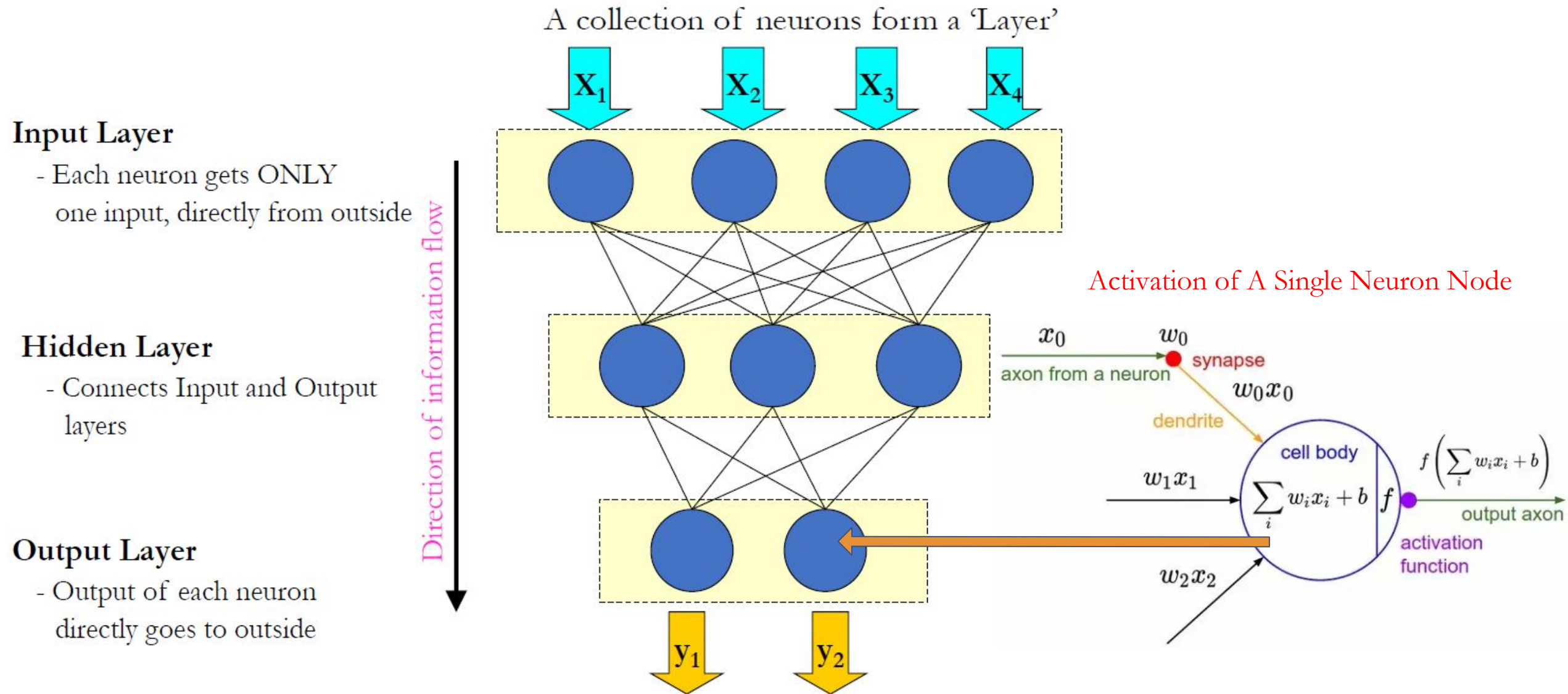
Activation functions are really important for NN to learn and make sense of something really **Complicated** and **Non-linear** complex functional mappings between the inputs and outputs. They introduce **non-linear properties** to our NN.



Neural Network: Feed Forward and Backpropagation



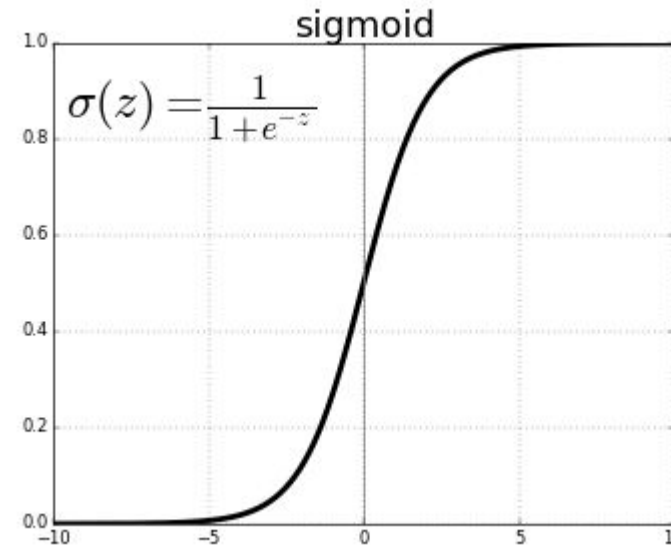
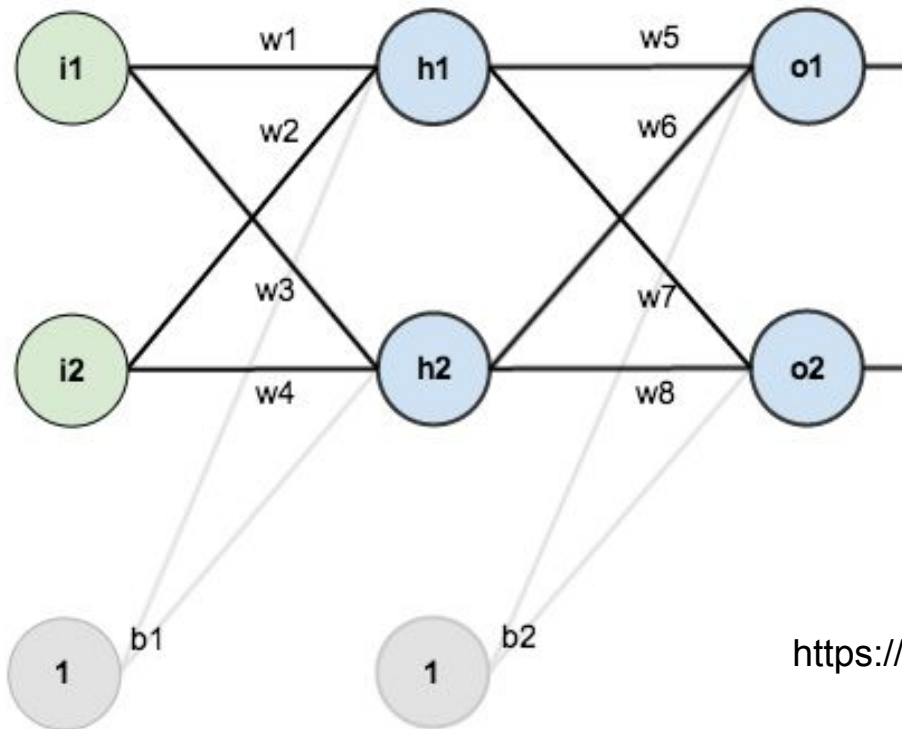
Feed Forward



Feed Forward: Example

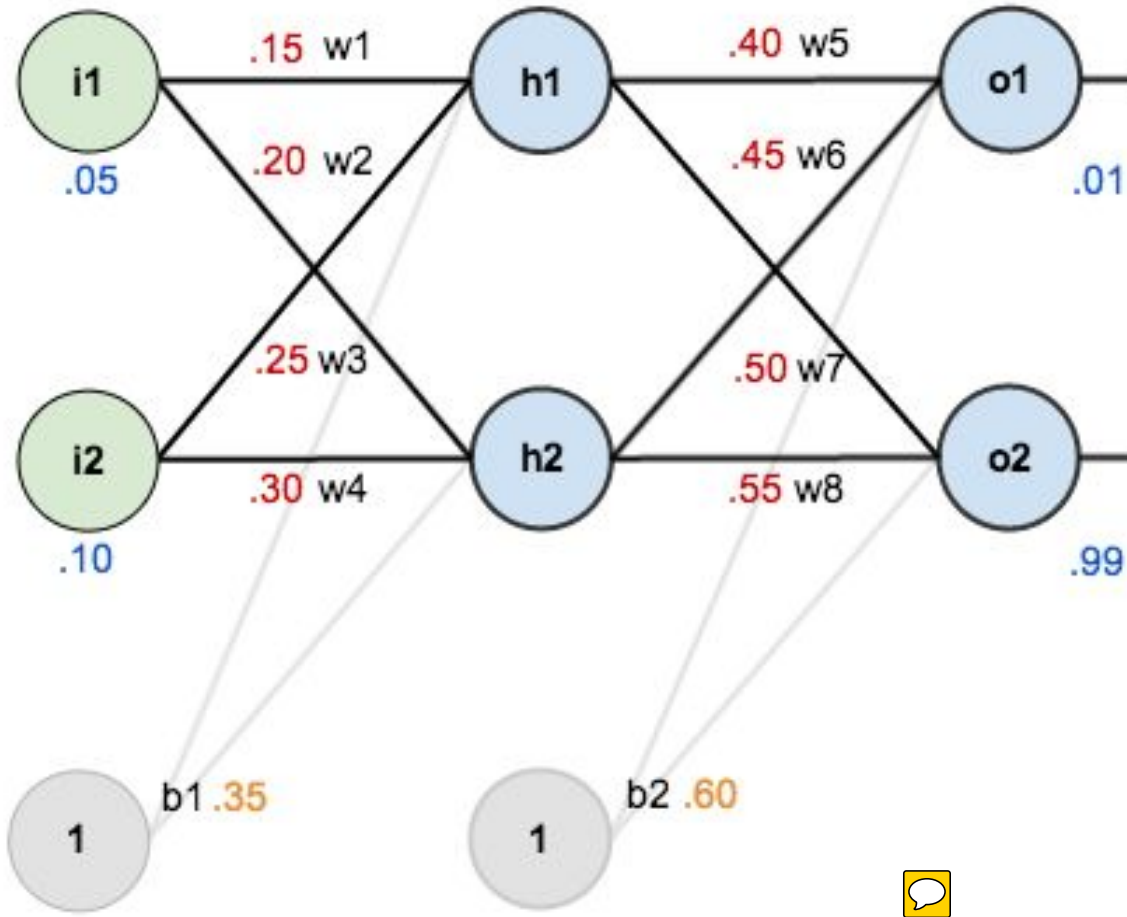
Suppose we have neural network. It has 2 inputs in the input layer, 1 hidden layer with 2 neurons, and 2 outputs in the output layer.

Suppose Activation function is sigmoid function (logistic function)



<https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>

Feed Forward: Example

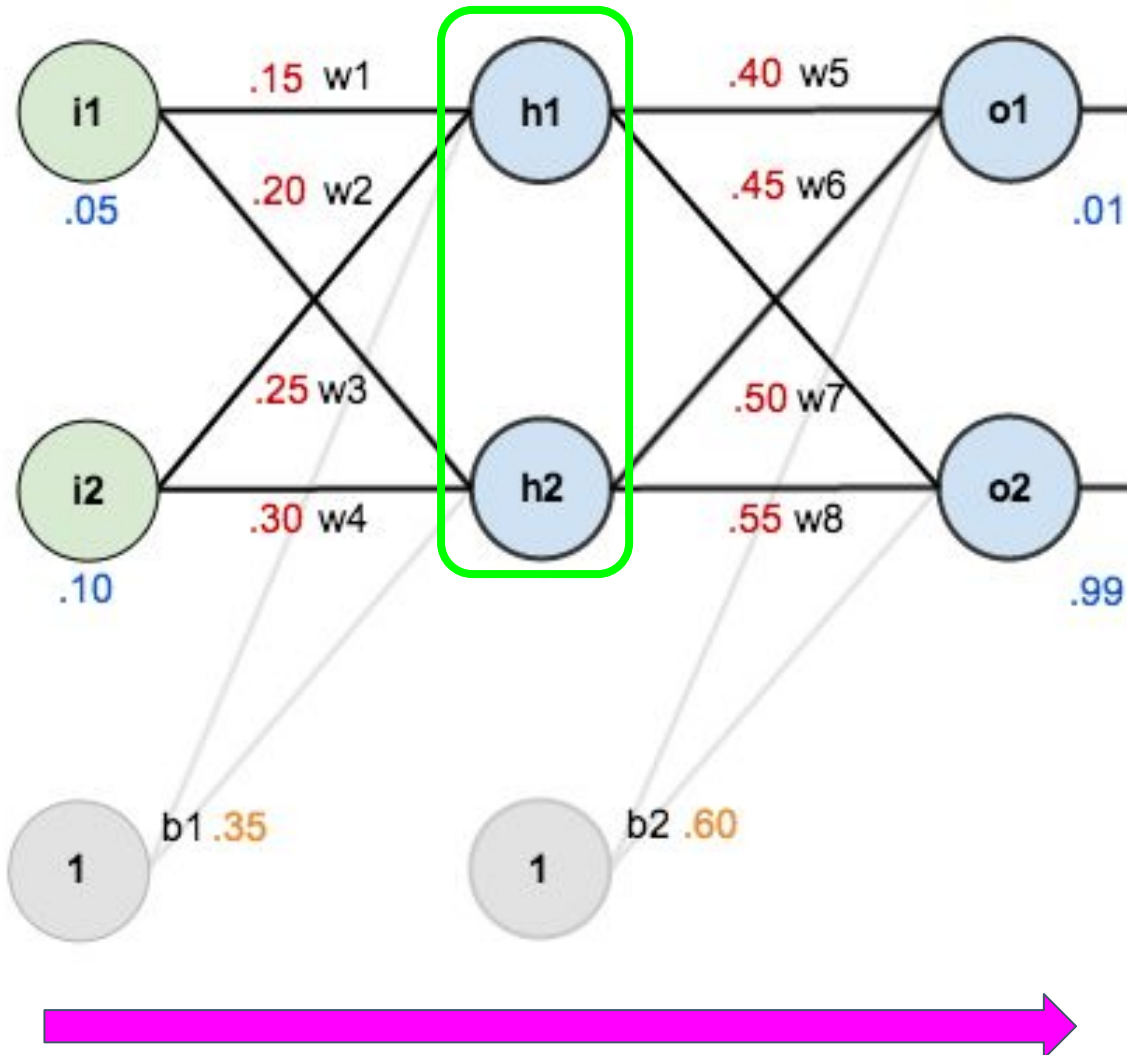


Step-1: Initialize weights/parameters

- In practice, set random initial weights
 - uniform distribution (0,1)
 - normal distribution $N(0,1)$
- Do not set initial weights too high (e.g., 100) or too low (e.g., all 0s), otherwise gradient descent is very slow, you may get poor solutions



Feed Forward: Example



Step-2: Forward pass to hidden layers

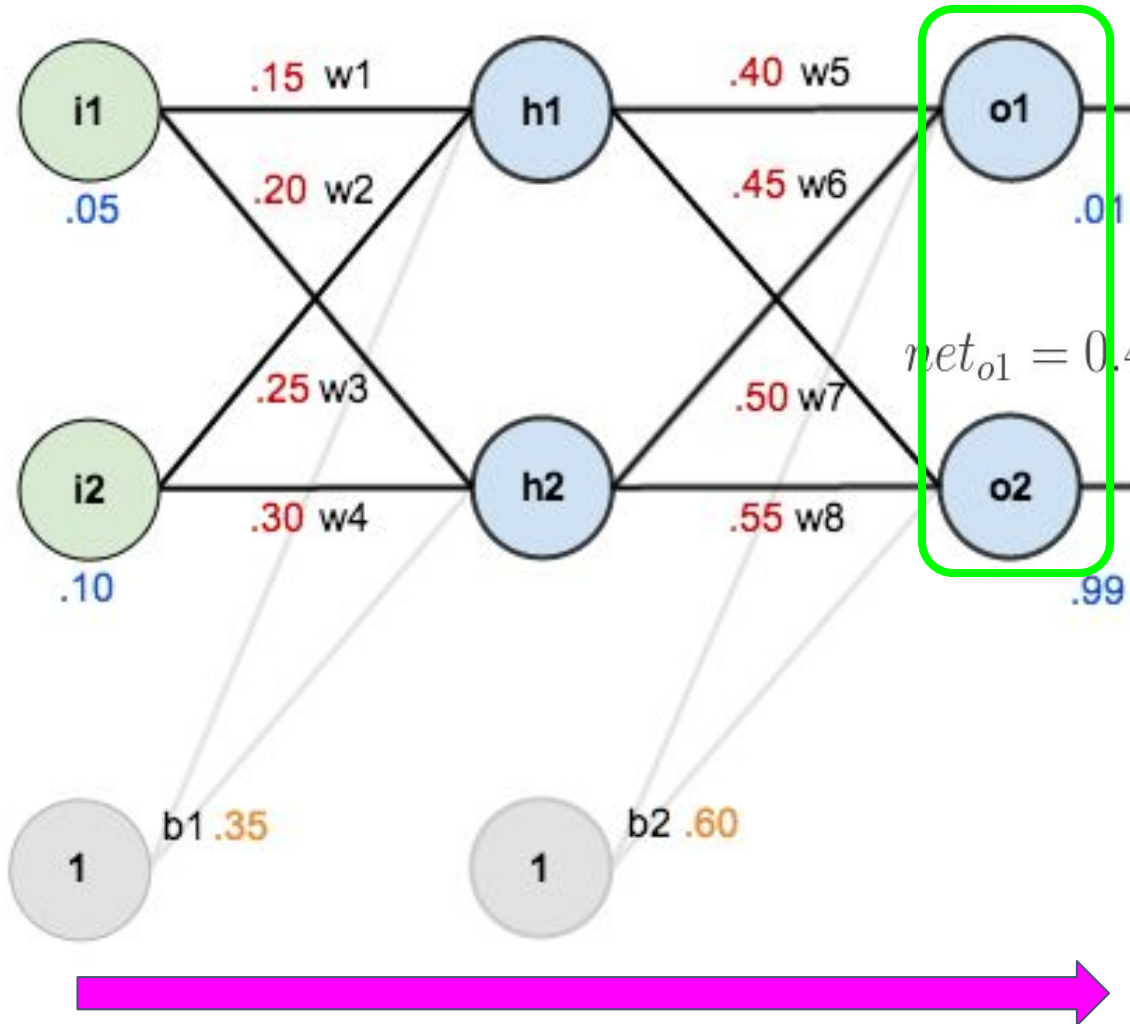
- Input layer: $i1=0.05$, $i2=0.1$
- Network **input** for neuron **h1** in hidden layer:
- $net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$
- $net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$
- Network **output** for neuron **h1** in hidden layer:

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

- Network **output** for neuron **h2** in hidden layer:

$$out_{h2} = 0.596884378$$

Feed Forward: Example



Step-3: Forward pass to output layer

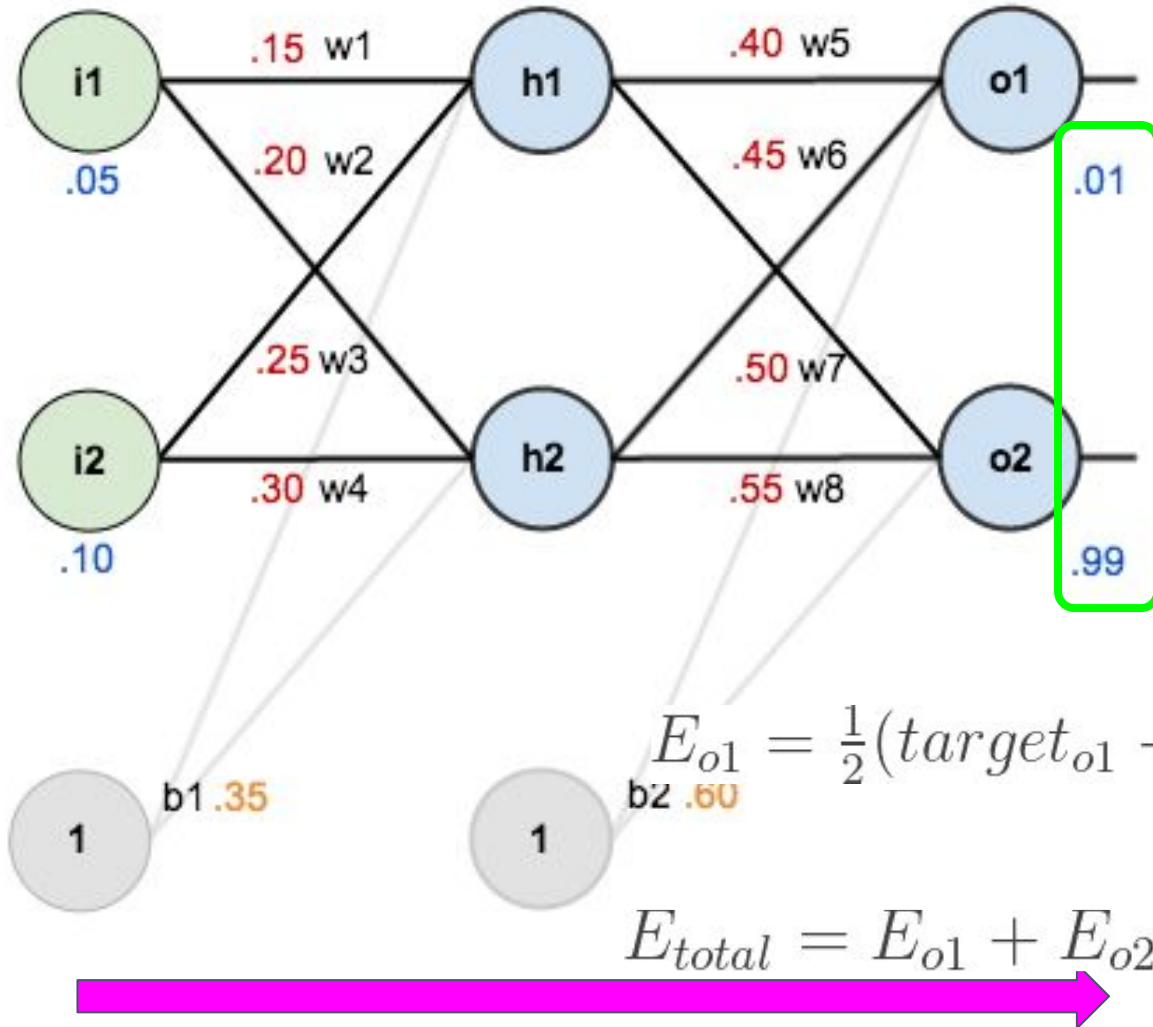
- Network **input** for neuron **o1** in output layer:
- $net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$
 $net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$
- Network **output** for neuron **o1** in output layer:

$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}} = \frac{1}{1 + e^{-1.105905967}} = 0.75136507$$

- Network **output** for neuron **o2** in output layer:

$$out_{o2} = 0.772928465$$

Feed Forward: Example



Step-4: Calculate errors

- Actual output for **o1** is 0.01; Actual output for **o2** is 0.99
- Predicted output for **o1** is 0.75; Predicted output for **o2** is 0.77
- We need to reduce errors
- Overall/Total error is:

$$E_{total} = \sum \frac{1}{2} (target - output)^2$$

$$E_{o1} = \frac{1}{2} (target_{o1} - out_{o1})^2 = \frac{1}{2} (0.01 - 0.75136507)^2 = 0.274811083$$

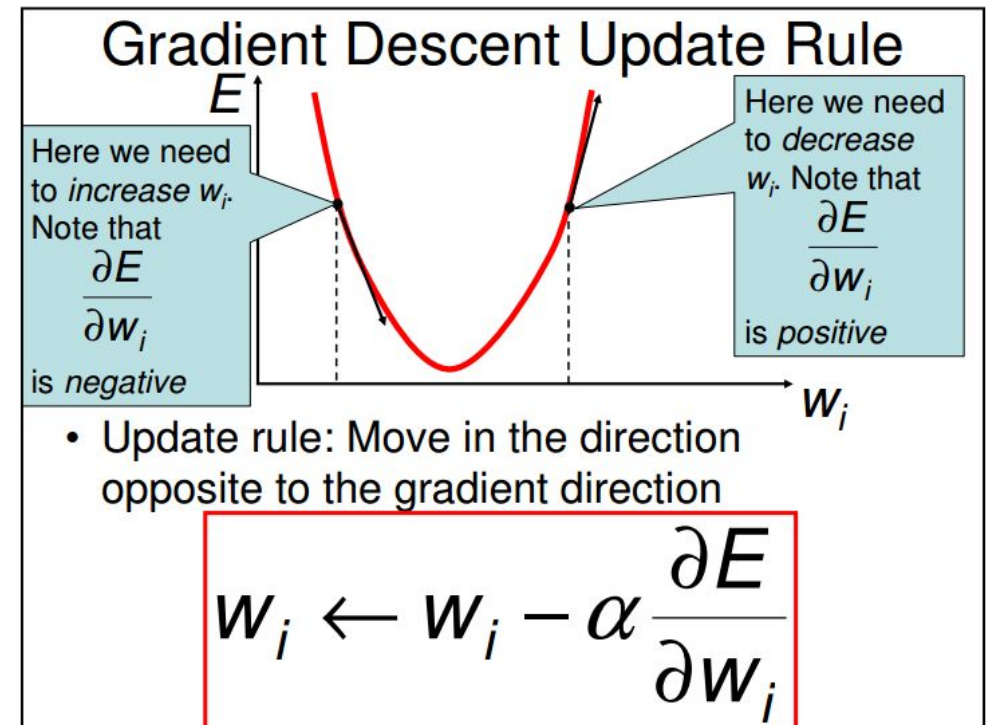
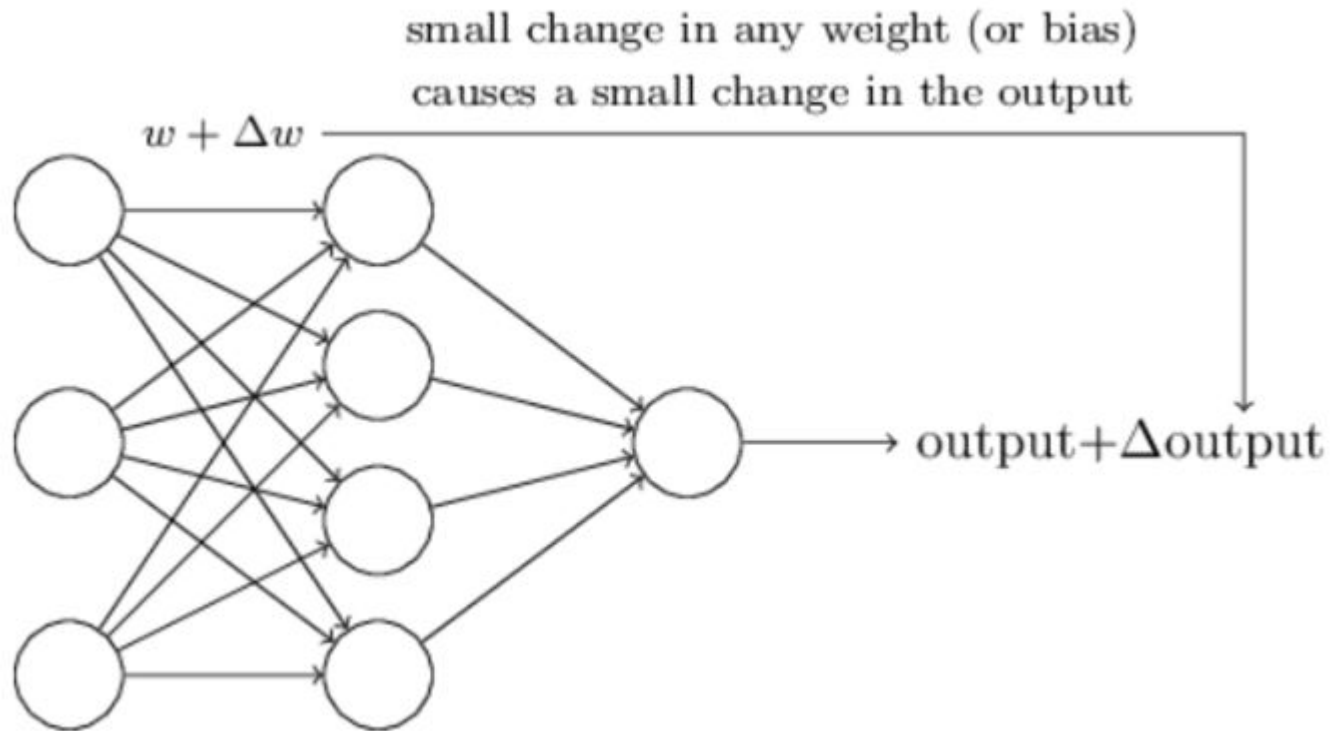
$$E_{o2} = 0.023560026$$

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

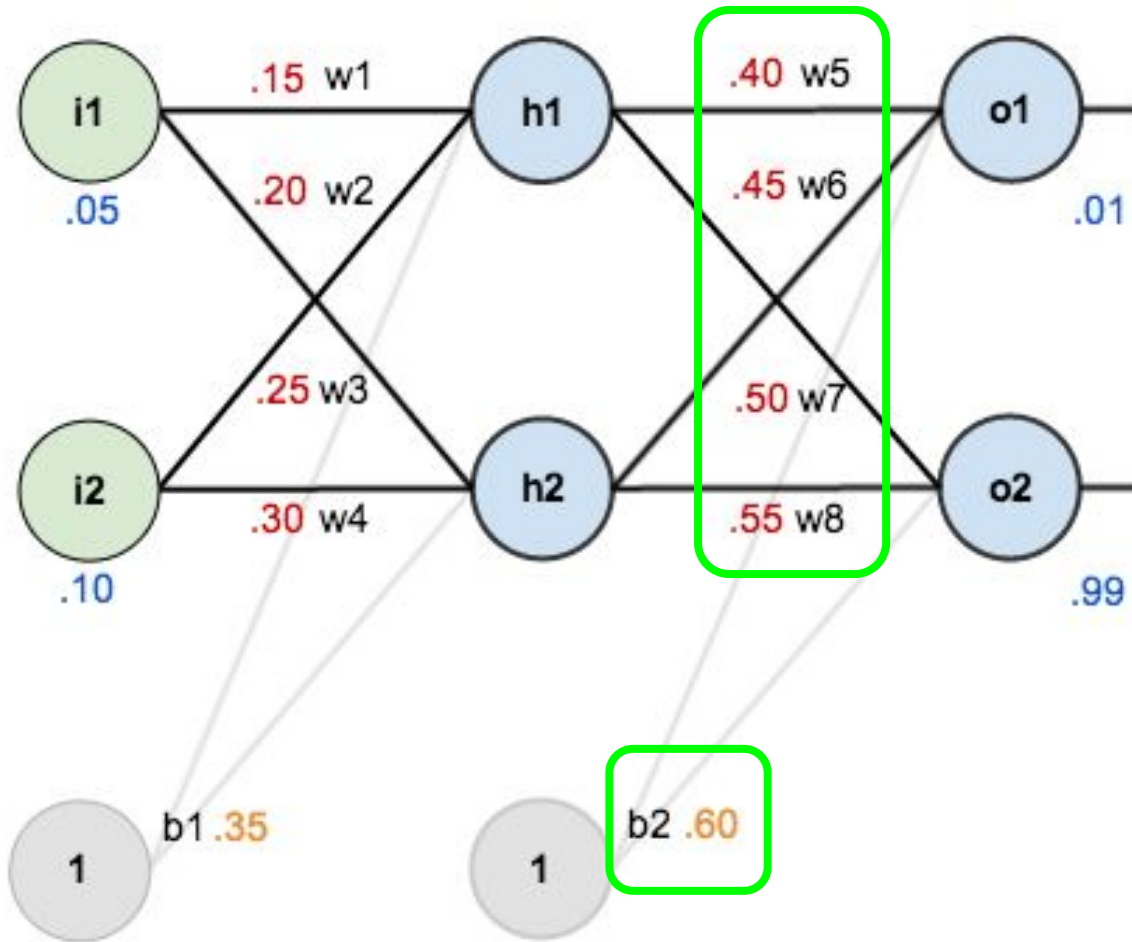
Backward Propagation (Backpropagation)

Why do we need backpropagation?

- Reduce overall/total errors after updating weights
- Go back to update weights/parameters (**Gradient Descent**)



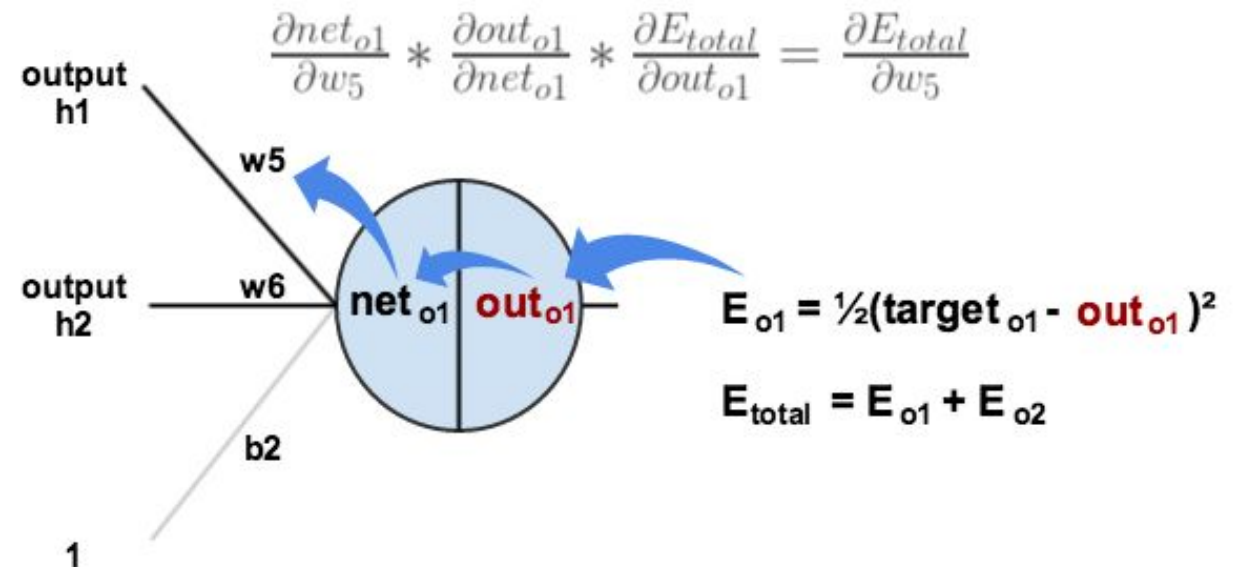
Backpropagation: Example



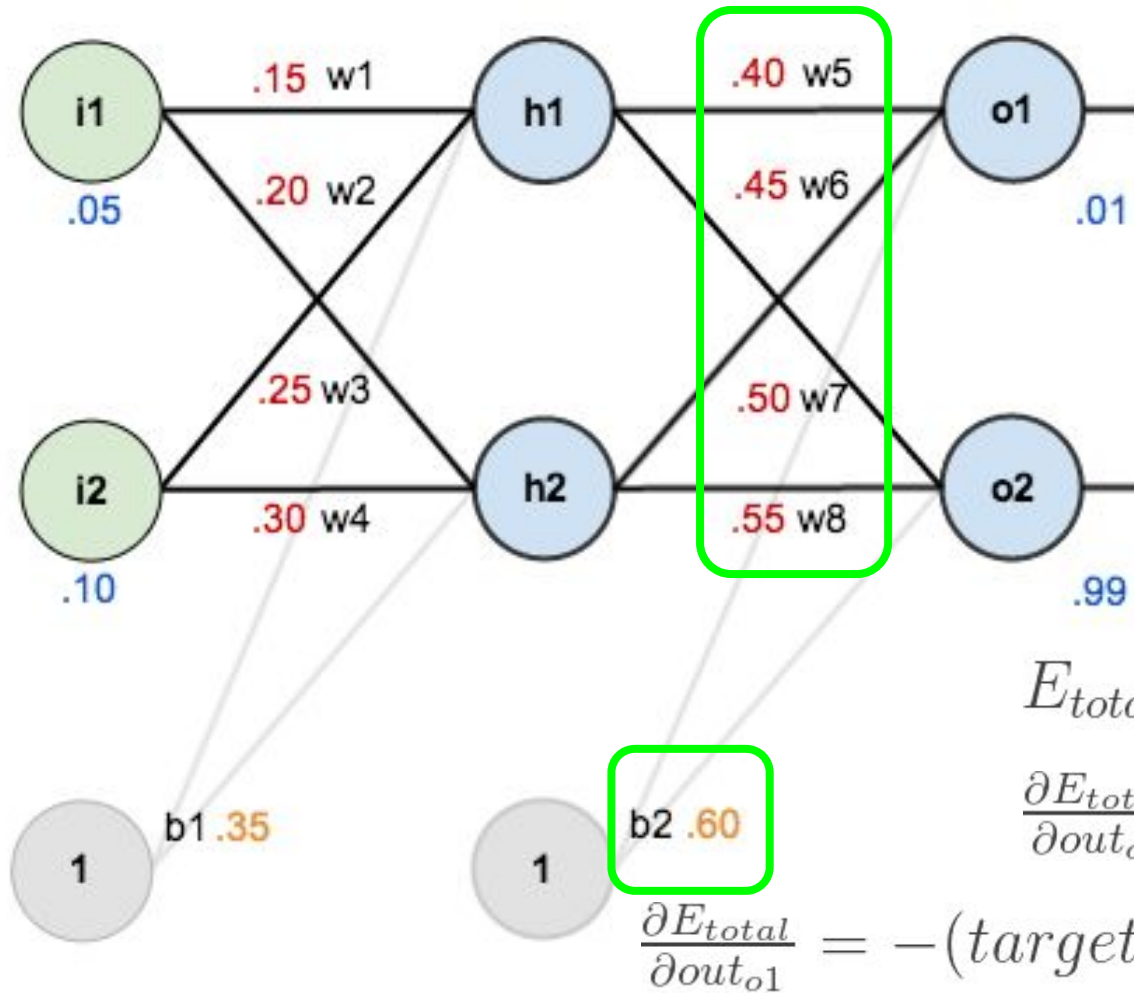
Step-1: Calculate derivatives on weights

- Gradient **w5**: Applying **Chain Rule**

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

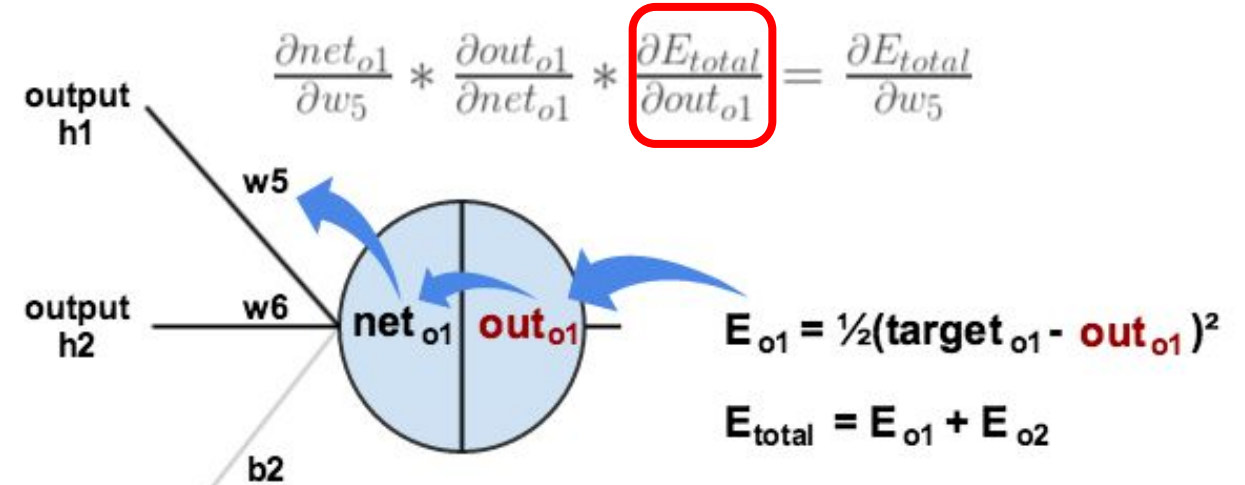


Backpropagation: Example



Step-1: Calculate derivatives on weights

- Gradient **w5**: Applying **Chain Rule**

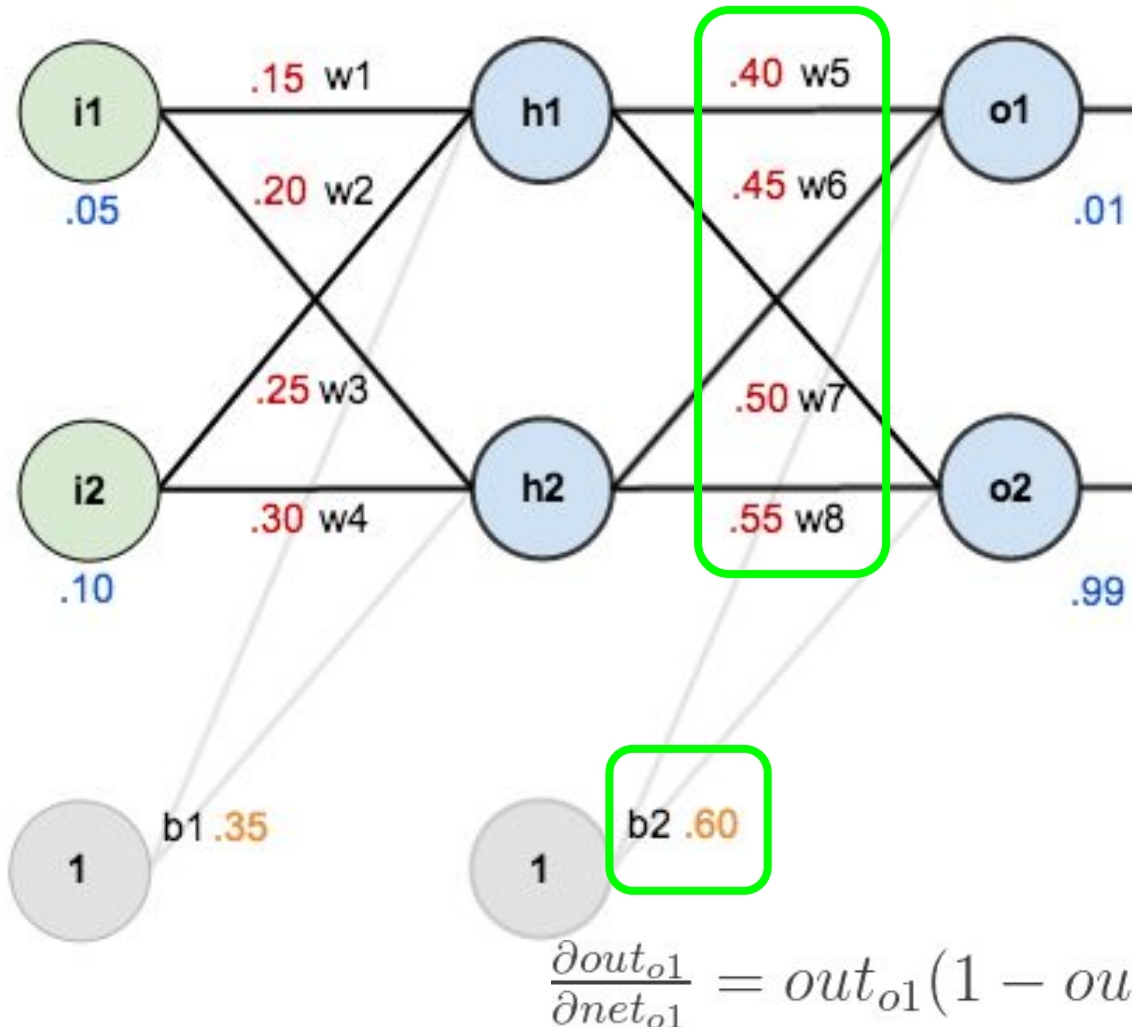


$$E_{\text{total}} = \frac{1}{2}(\text{target}_{o1} - \text{out}_{o1})^2 + \frac{1}{2}(\text{target}_{o2} - \text{out}_{o2})^2$$

$$\frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} = 2 * \frac{1}{2}(\text{target}_{o1} - \text{out}_{o1})^{2-1} * -1 + 0$$

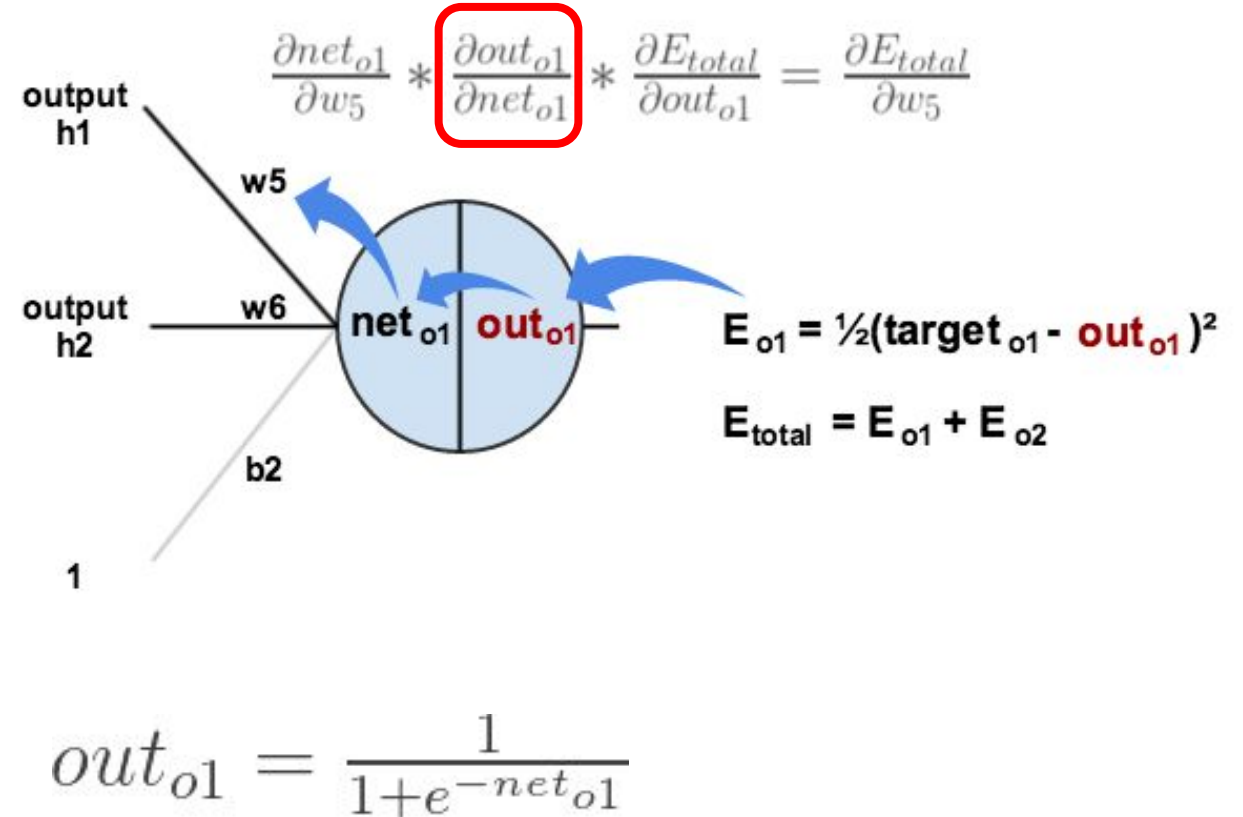
$$\frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} = -(\text{target}_{o1} - \text{out}_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

Backpropagation: Example

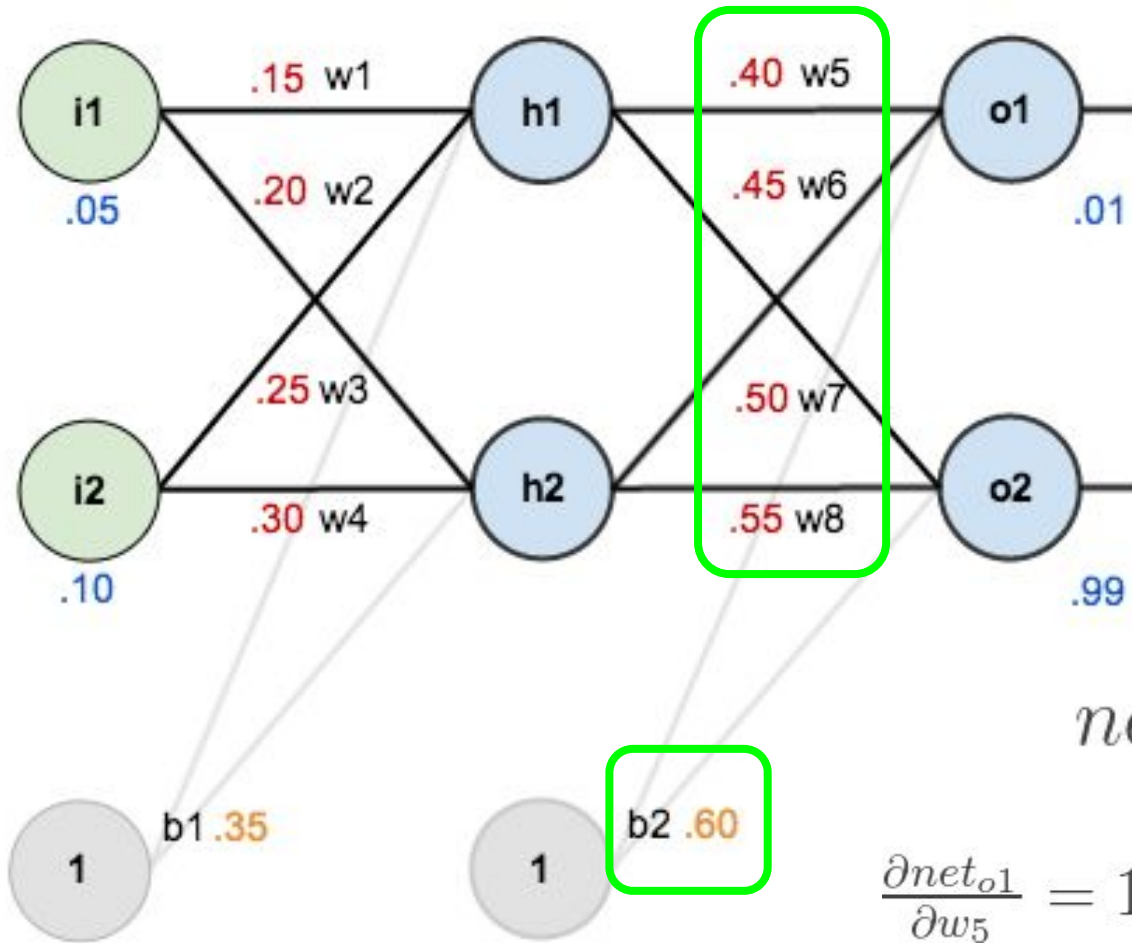


Step-1: Calculate derivatives on weights

- Gradient **w5**: Applying **Chain Rule**

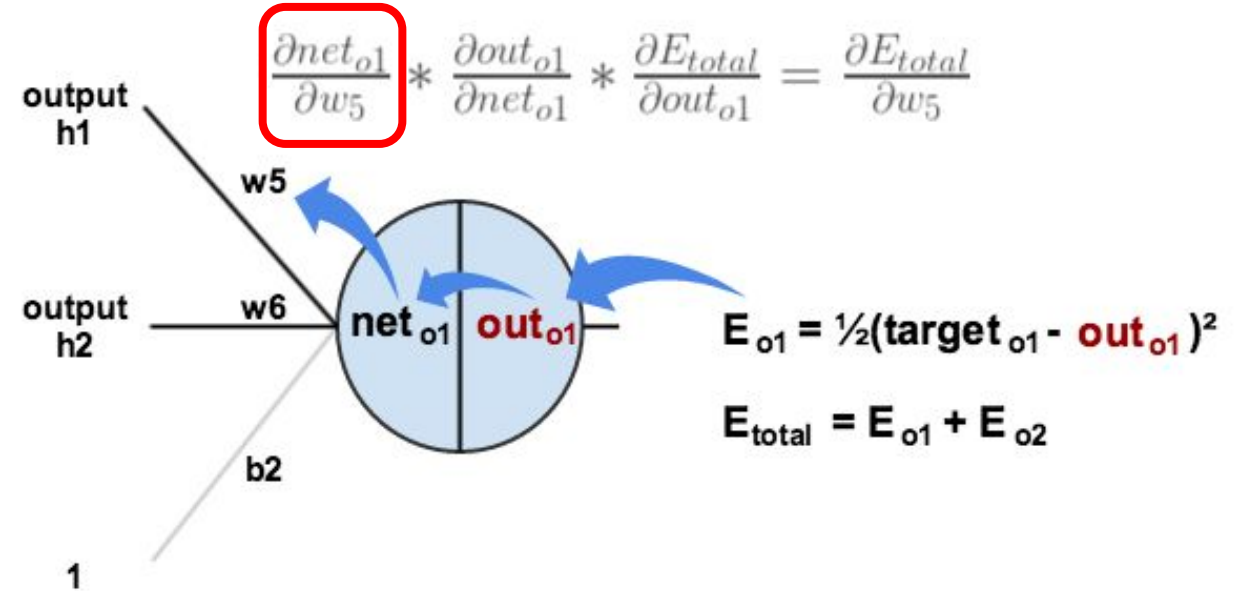


Backpropagation: Example



Step-1: Calculate derivatives on weights

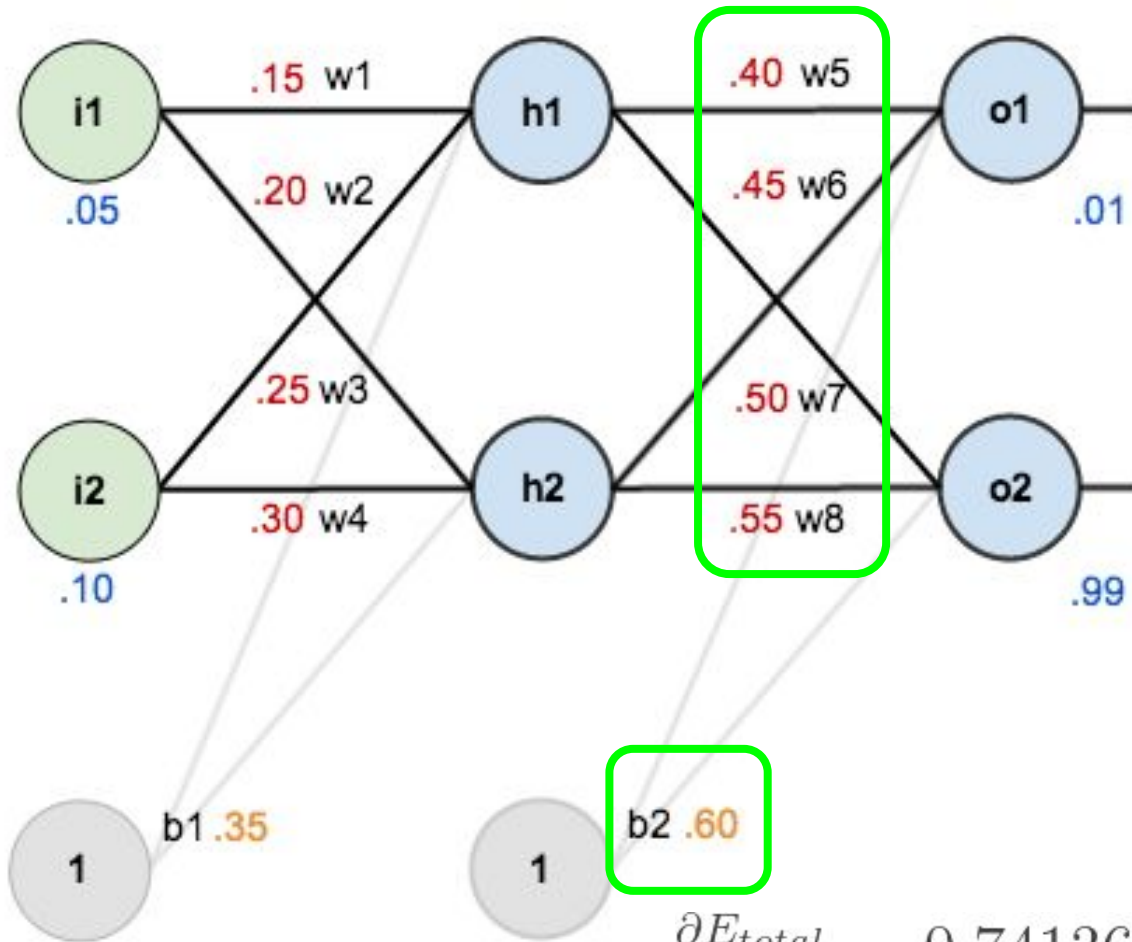
- Gradient **w5**: Applying **Chain Rule**



$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

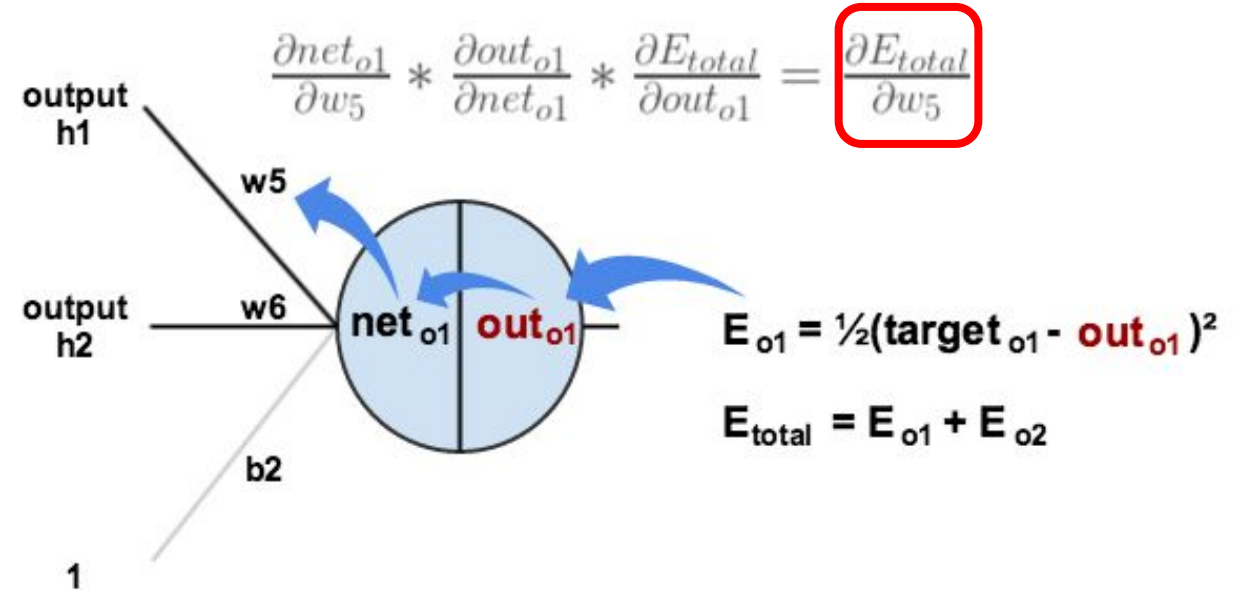
Backpropagation: Example



$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

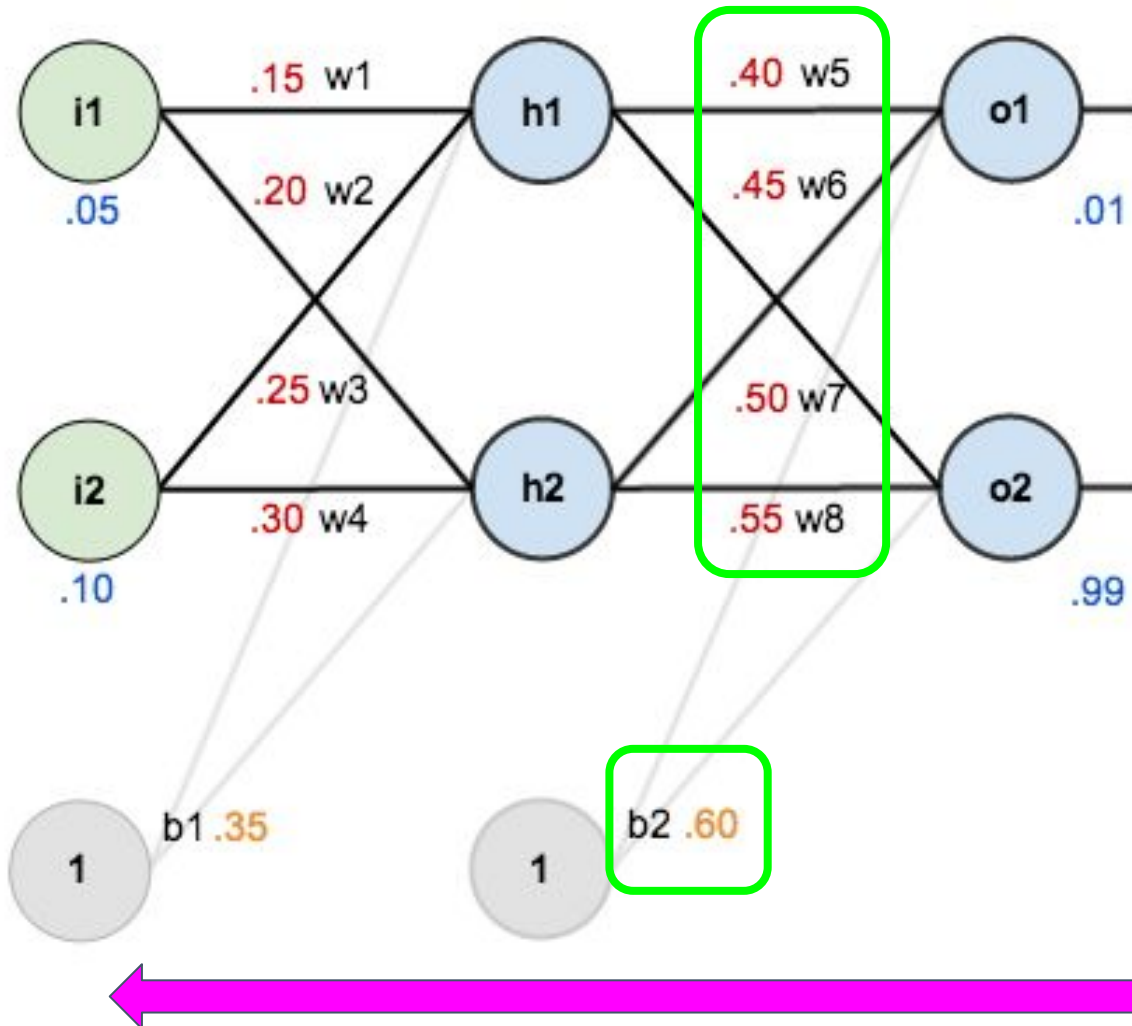
Step-1: Calculate derivatives on weights

- Gradient **w5**: Applying **Chain Rule**



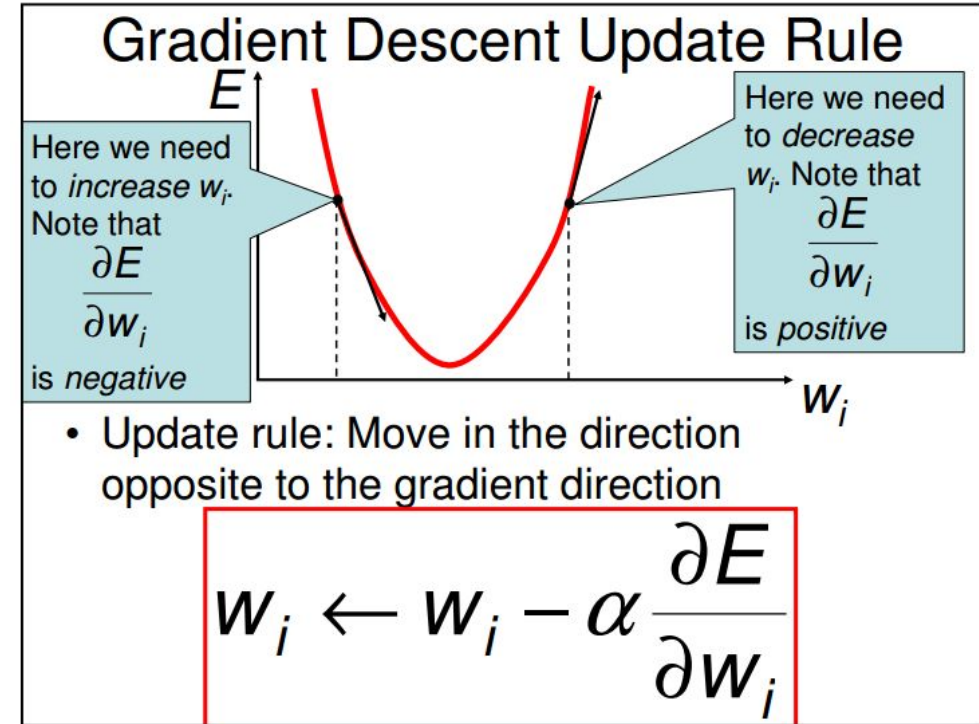
$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Backpropagation: Example



Step-2: Update weights

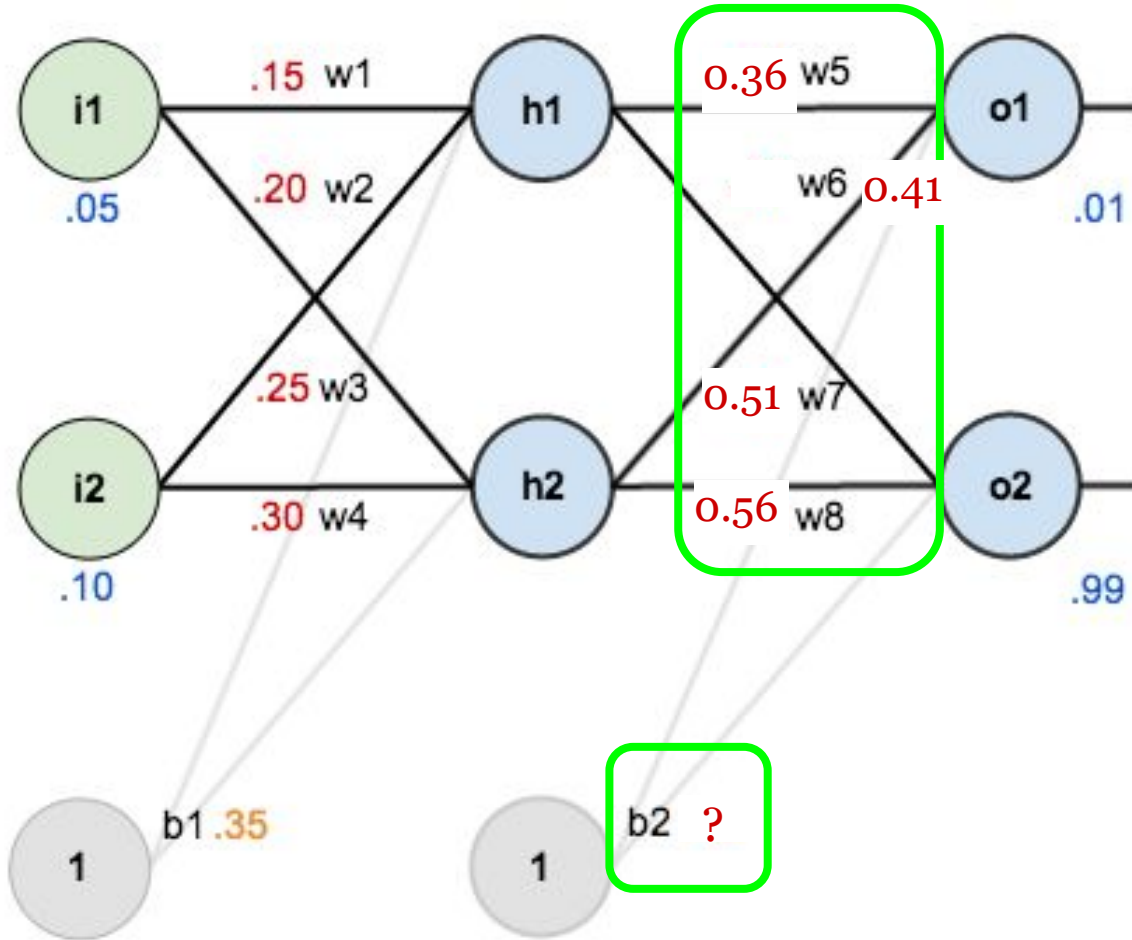
- Update **w5**: **Gradient Descent**
- Remember **Tutorial 2**:



- $w_5 \leftarrow w_5 - \text{stepsize} \times (\partial E_{\text{total}} / \partial w_5)$

$$w_5^+ = w_5 - \eta * \frac{\partial E_{\text{total}}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

Backpropagation: Example



Step-2: Update weights

- Update w_6, w_7, w_8, b_2 : **Gradient Descent**

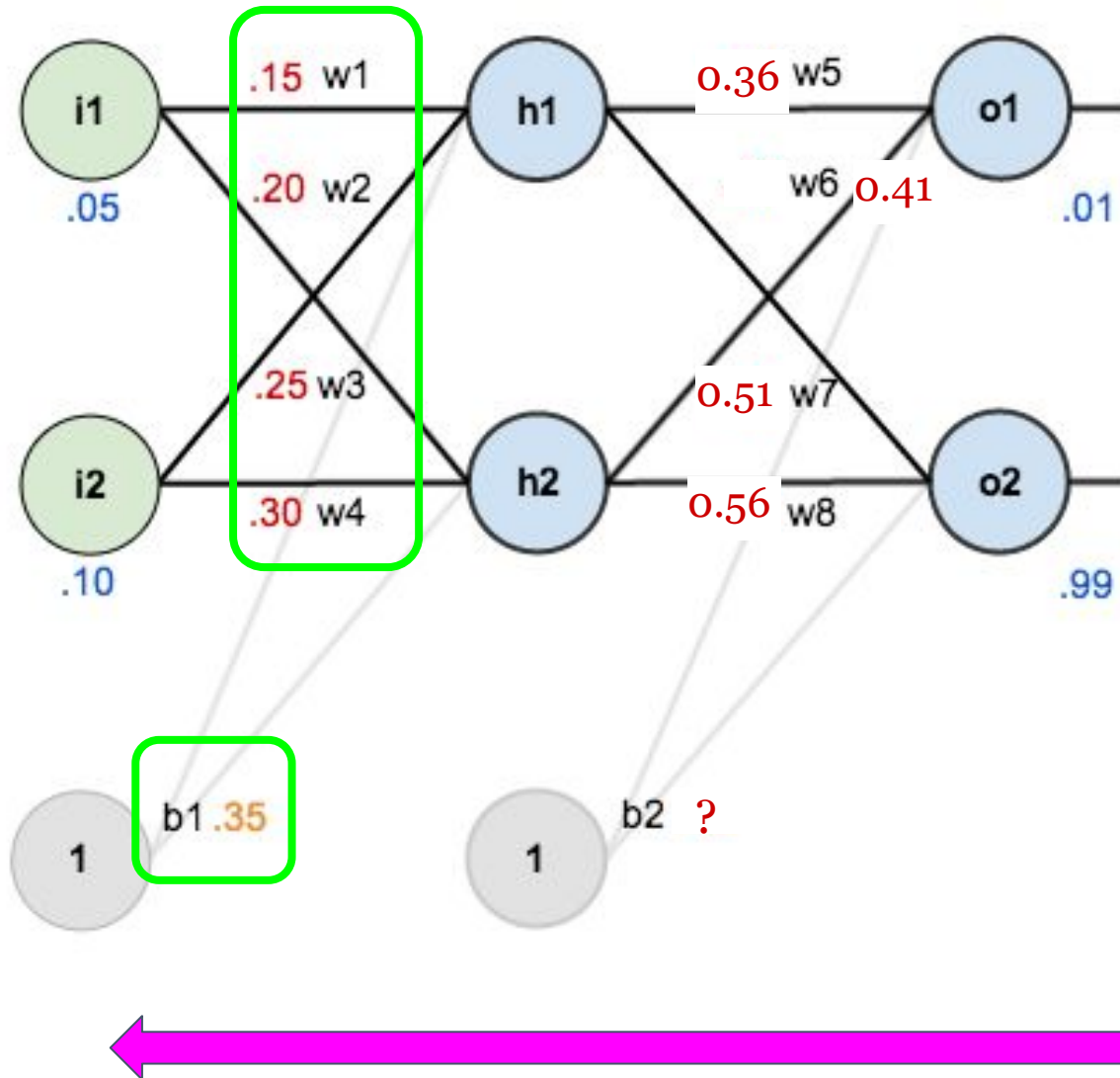
$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

b2: You can try to update b2 by yourself

Backpropagation: Example



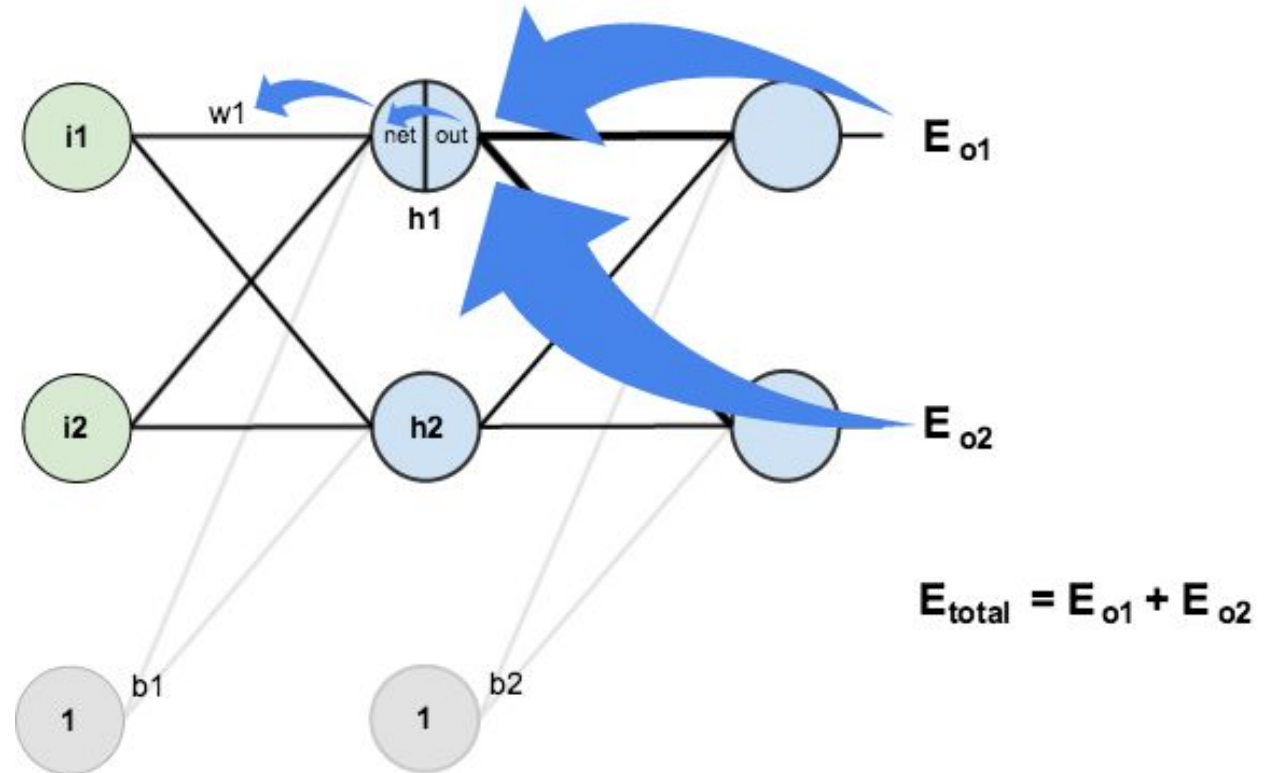
Step-3: Repeat updating weights

- Gradient **w1**: **Chain Rule**

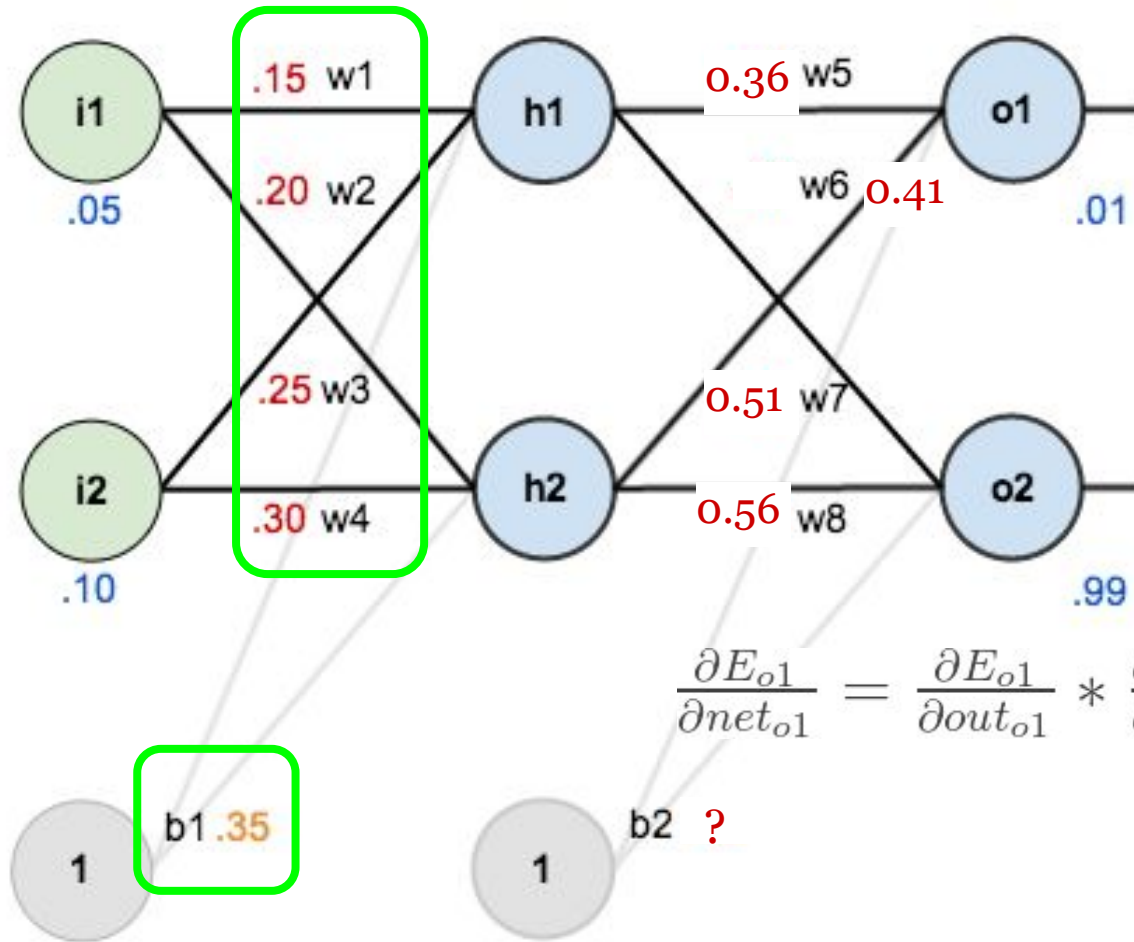
$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

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



Backpropagation: Example



Step-3: Repeat updating weights

- Gradient **w1**: **Chain Rule**

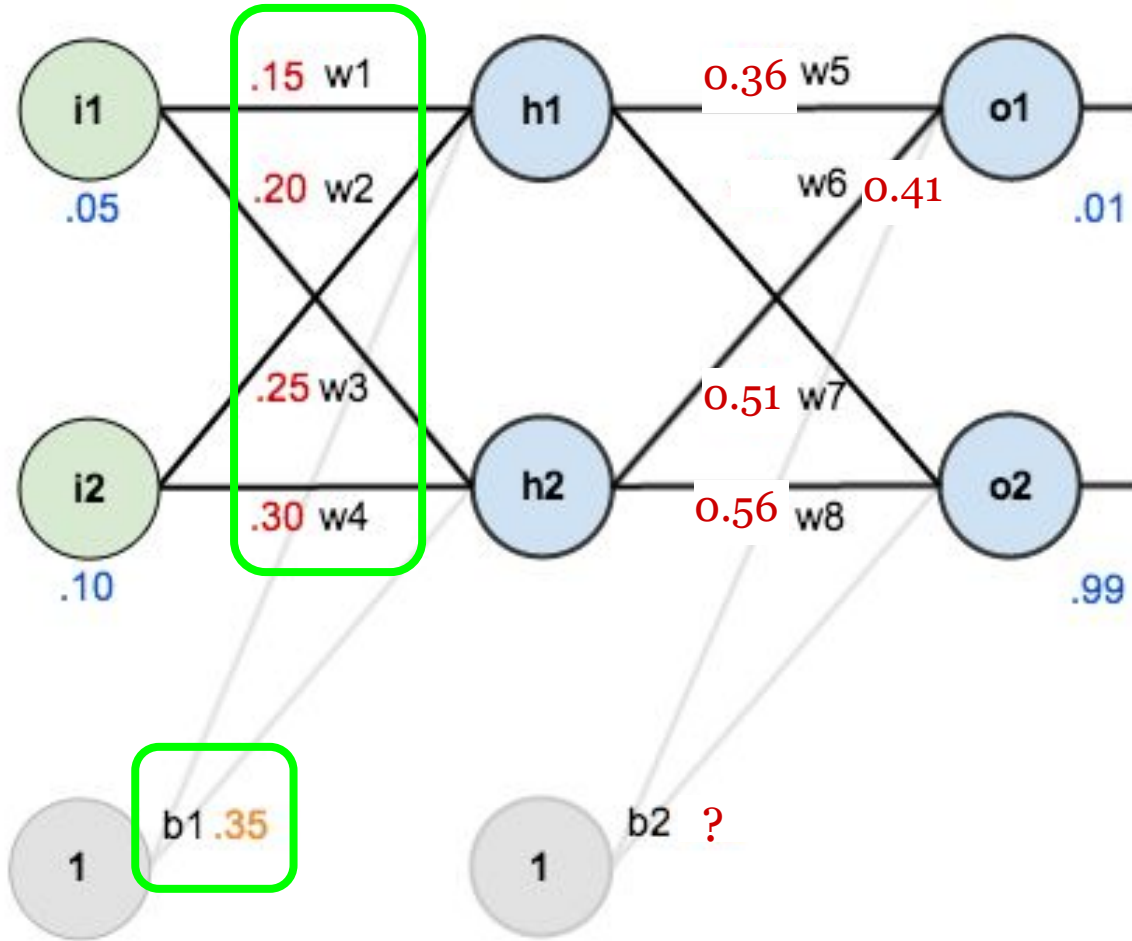
$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

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

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$

Backpropagation: Example



Step-3: Repeat updating weights

- Gradient **w1**: **Chain Rule**

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

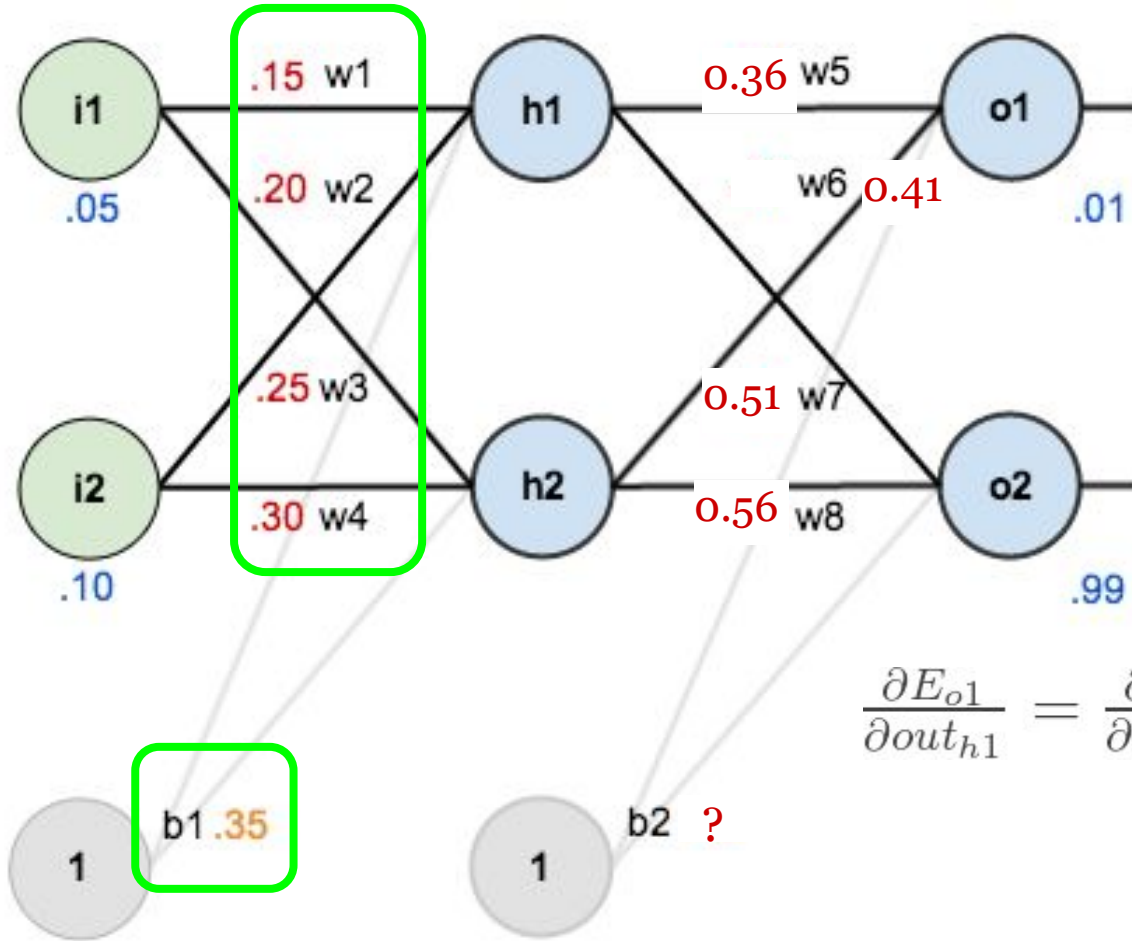
$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

Backpropagation: Example



Step-3: Repeat updating weights

- Gradient **w1**: **Chain Rule**

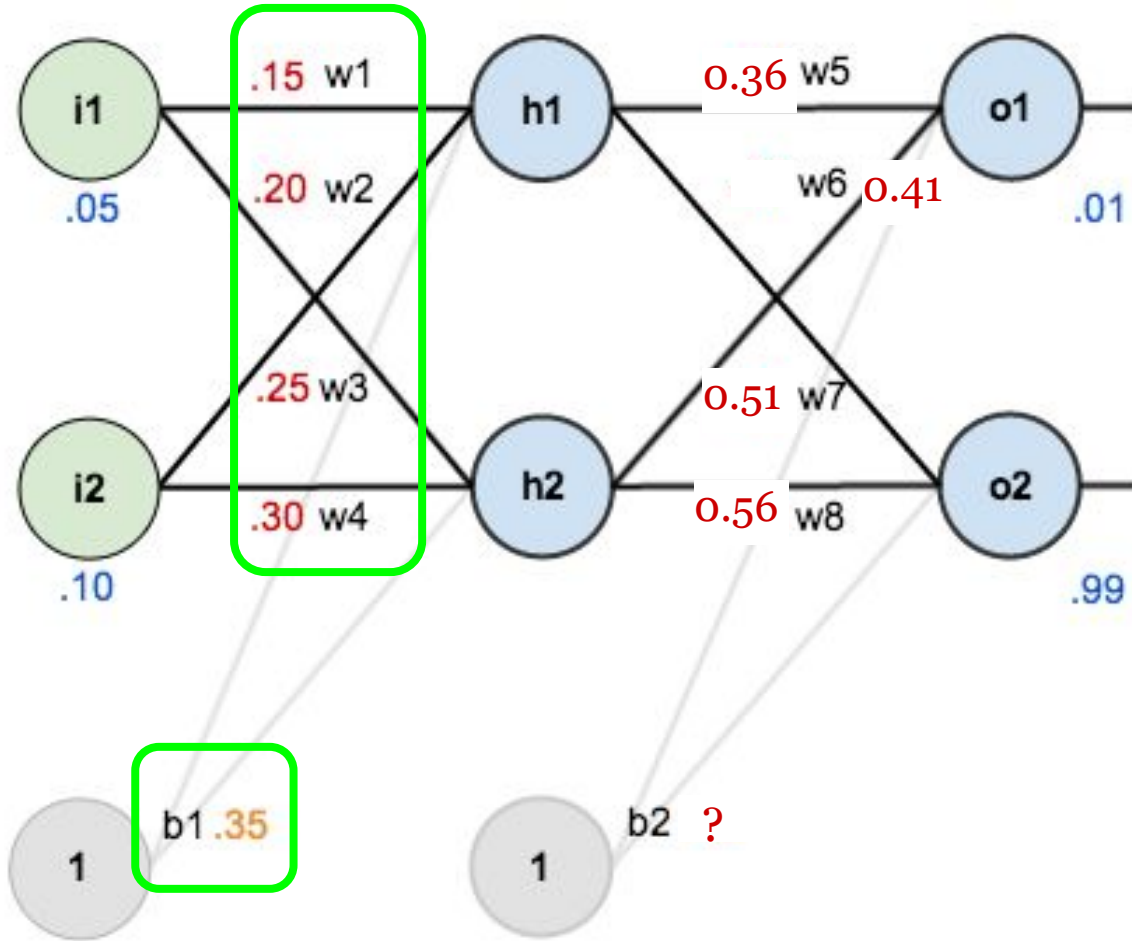
$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

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

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

Backpropagation: Example



Step-3: Repeat updating weights

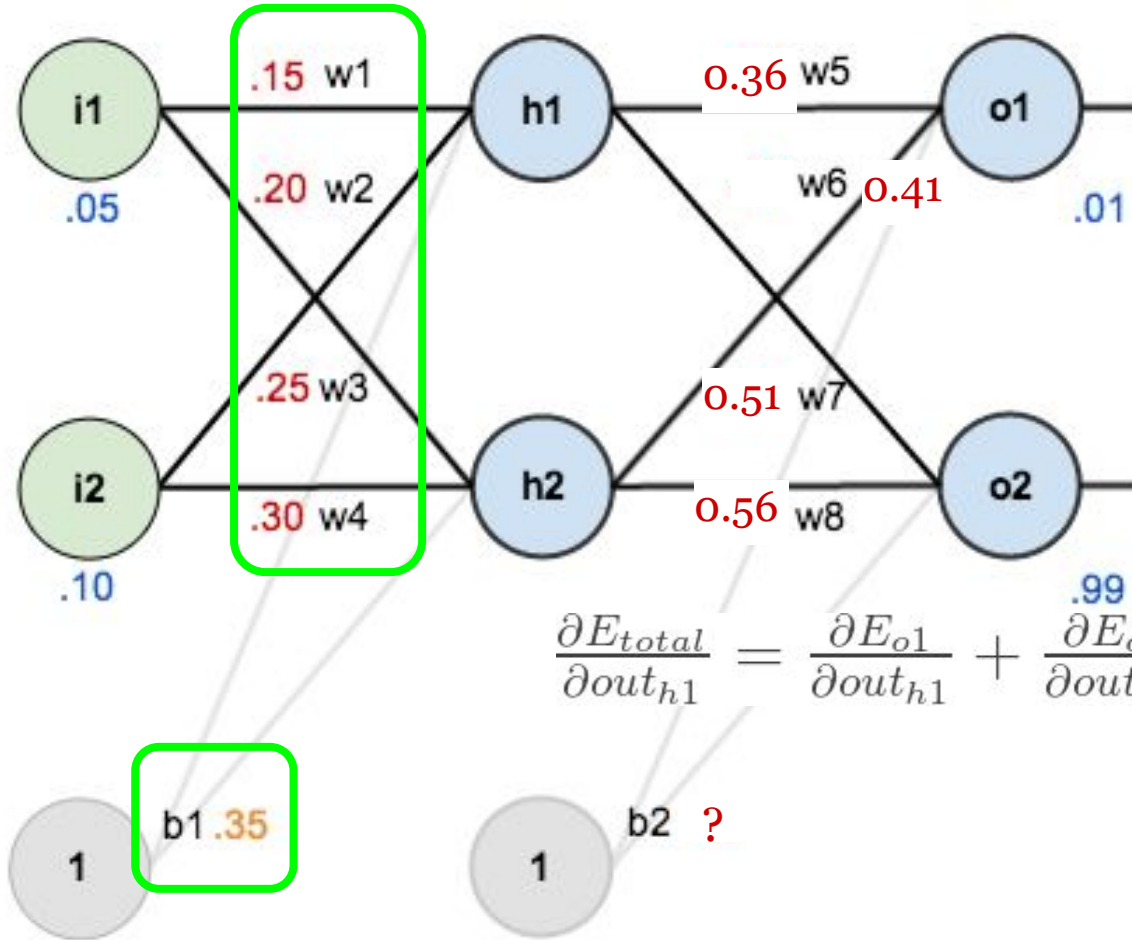
- Gradient **w1**: **Chain Rule**

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

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

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

Backpropagation: Example



Step-3: Repeat updating weights

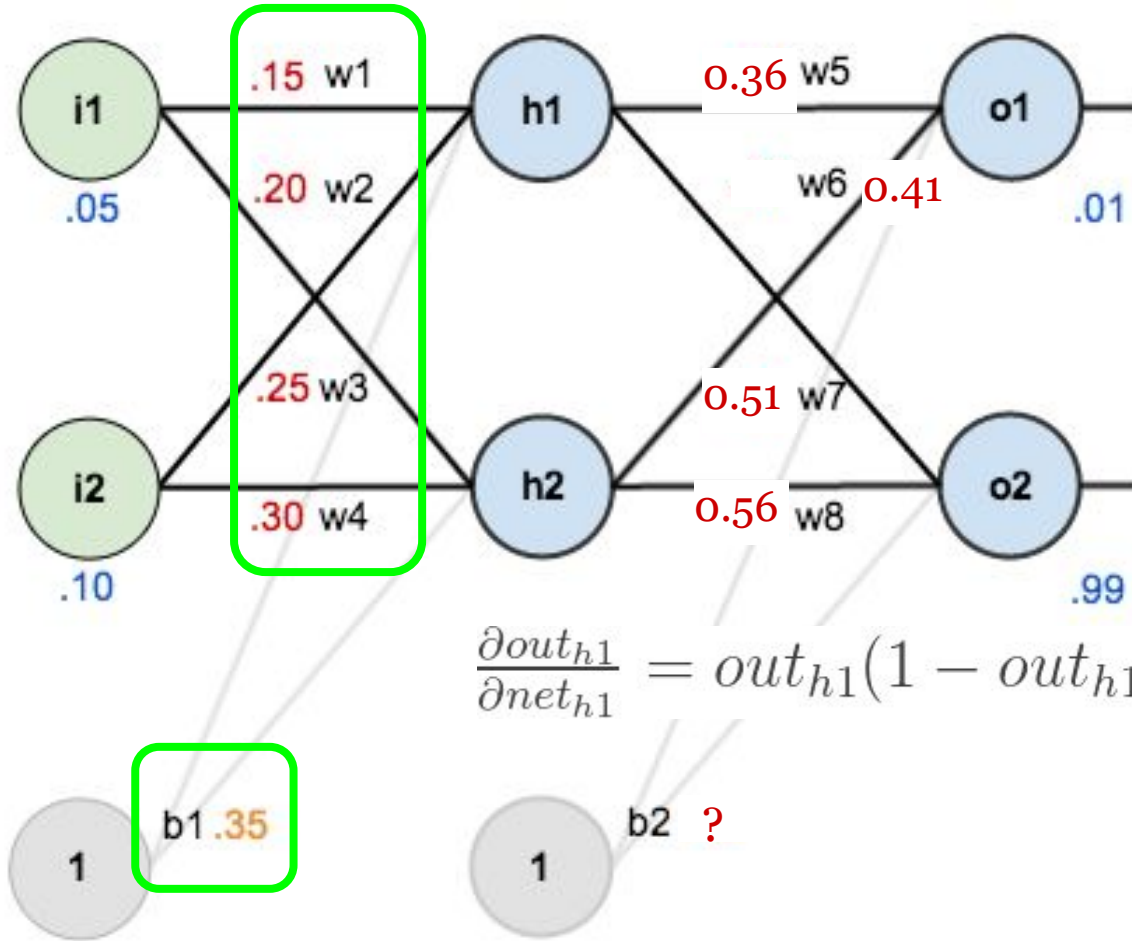
- Gradient **w1**: **Chain Rule**

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

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

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

Backpropagation: Example



Step-3: Repeat updating weights

- Gradient **w1**: **Chain Rule**

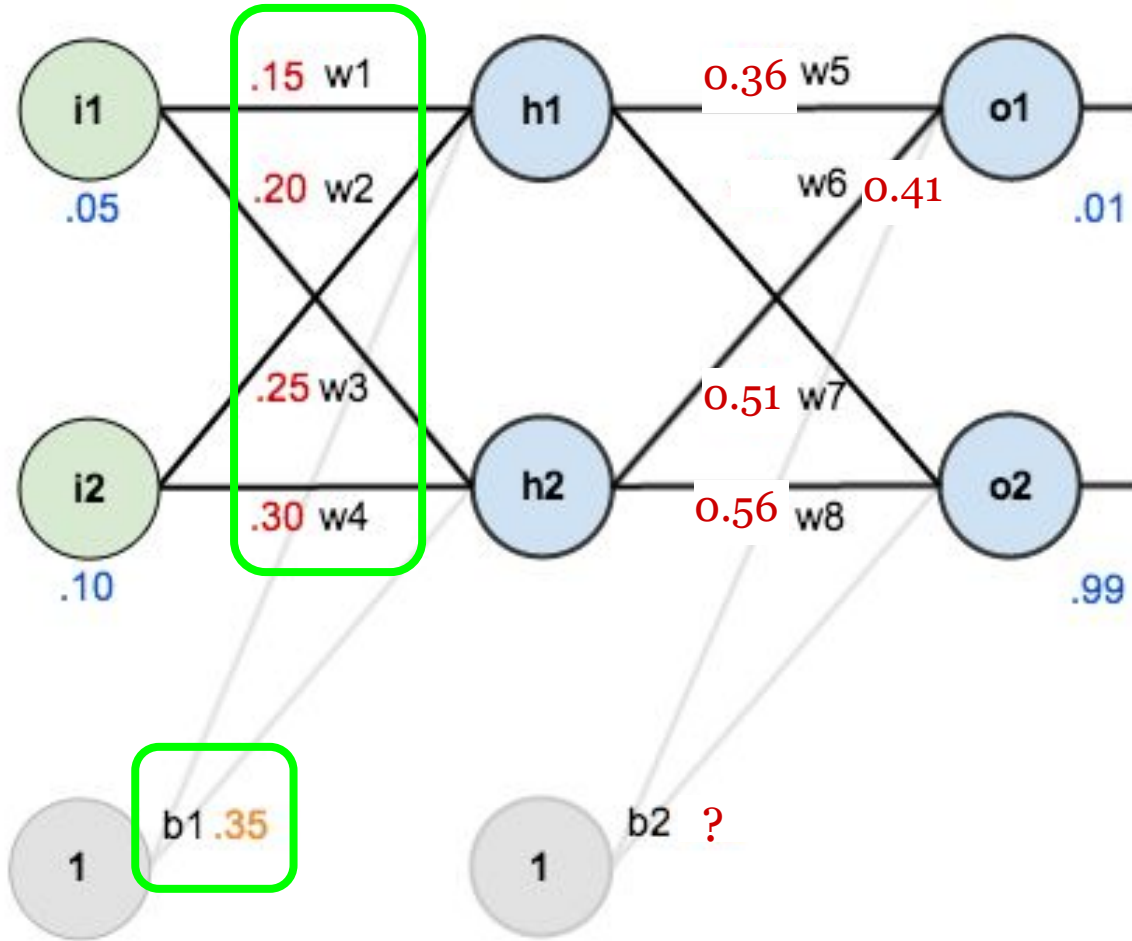
$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$



Backpropagation: Example



Step-3: Repeat updating weights

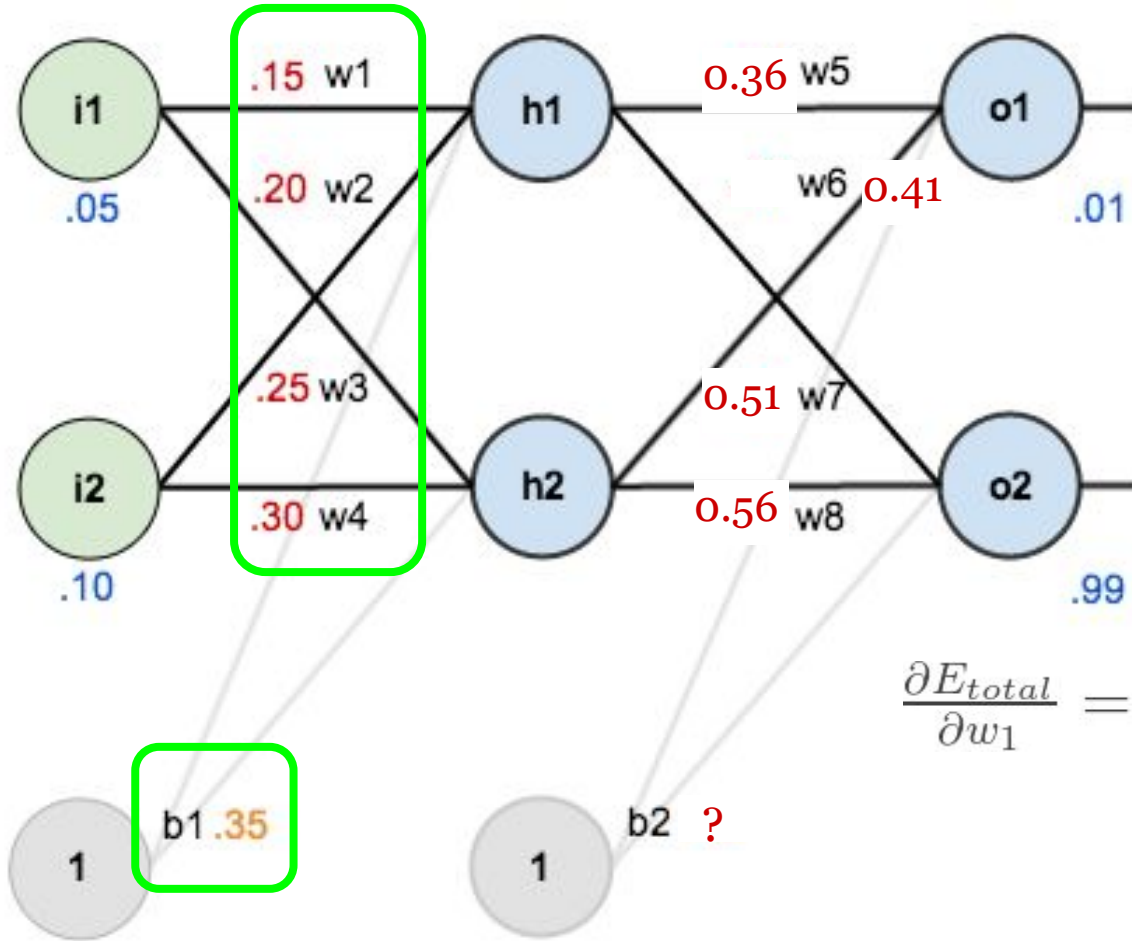
- Gradient **w1**: **Chain Rule**

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

$$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

Backpropagation: Example



Step-3: Repeat updating weights

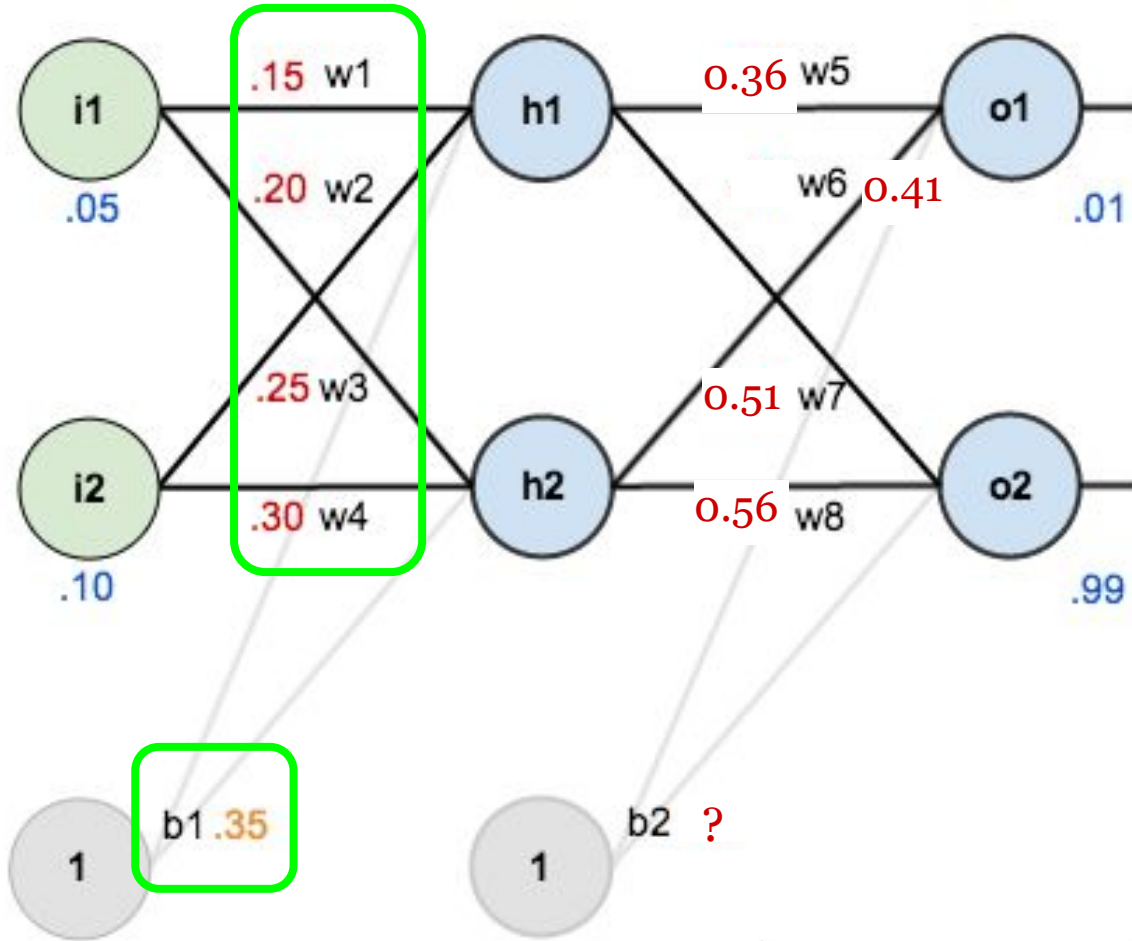
- Gradient **w1**: **Chain Rule**

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

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

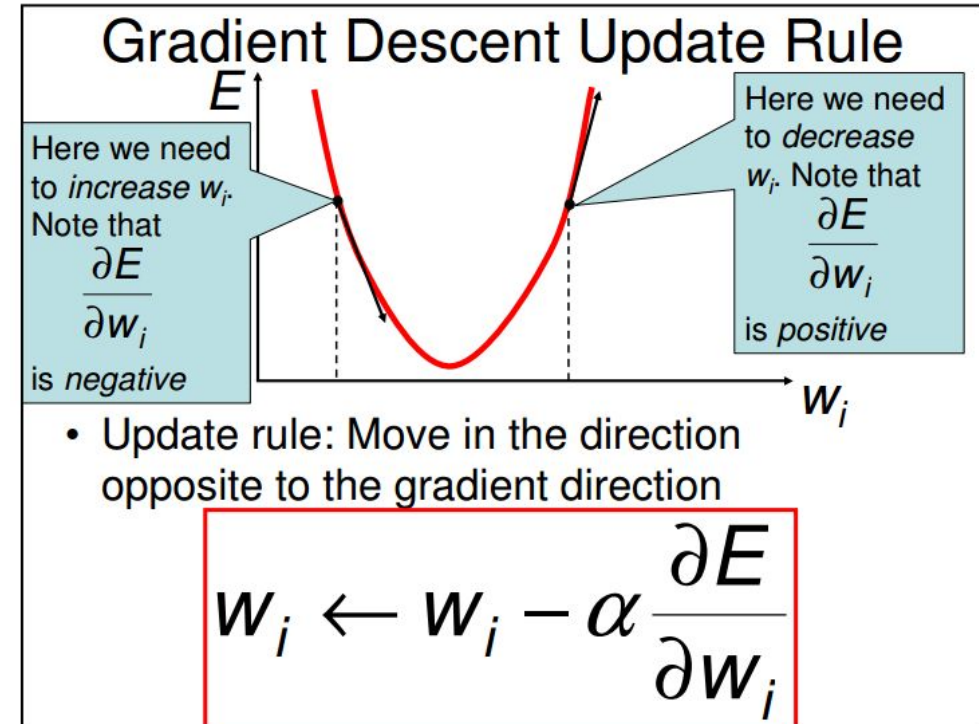
$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

Backpropagation: Example



Step-3: Repeat updating weights

- Update **w1**: **Gradient Descent**

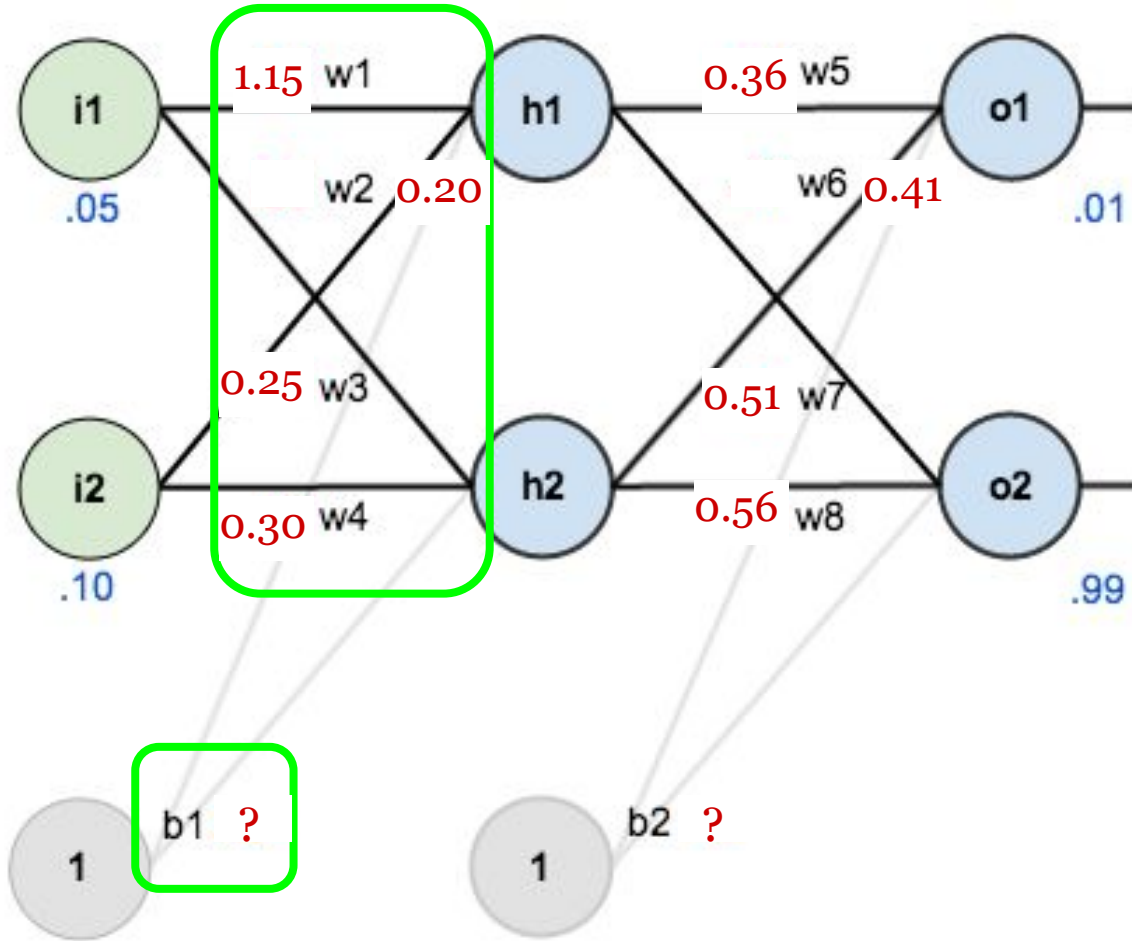


- $w_1 \leftarrow w_1 - \text{stepsize} \times (\partial E_{\text{total}} / \partial w_1)$

$$w_1^+ = w_1 - \eta * \frac{\partial E_{\text{total}}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$



Backpropagation: Example



Step-3: Repeat updating weights

- Update w_2, w_3, w_4, b_1 : **Gradient Descent**

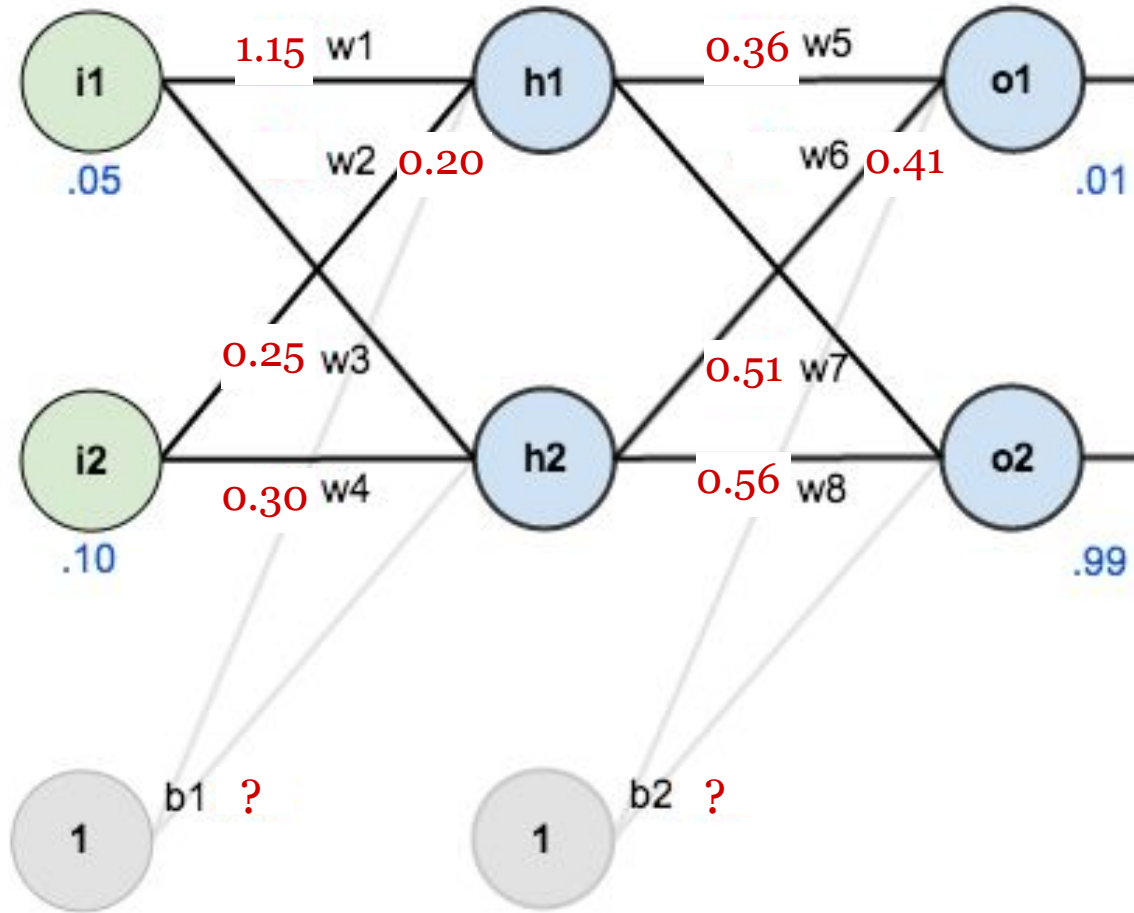
$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

b_1 : You can try to update it by yourself

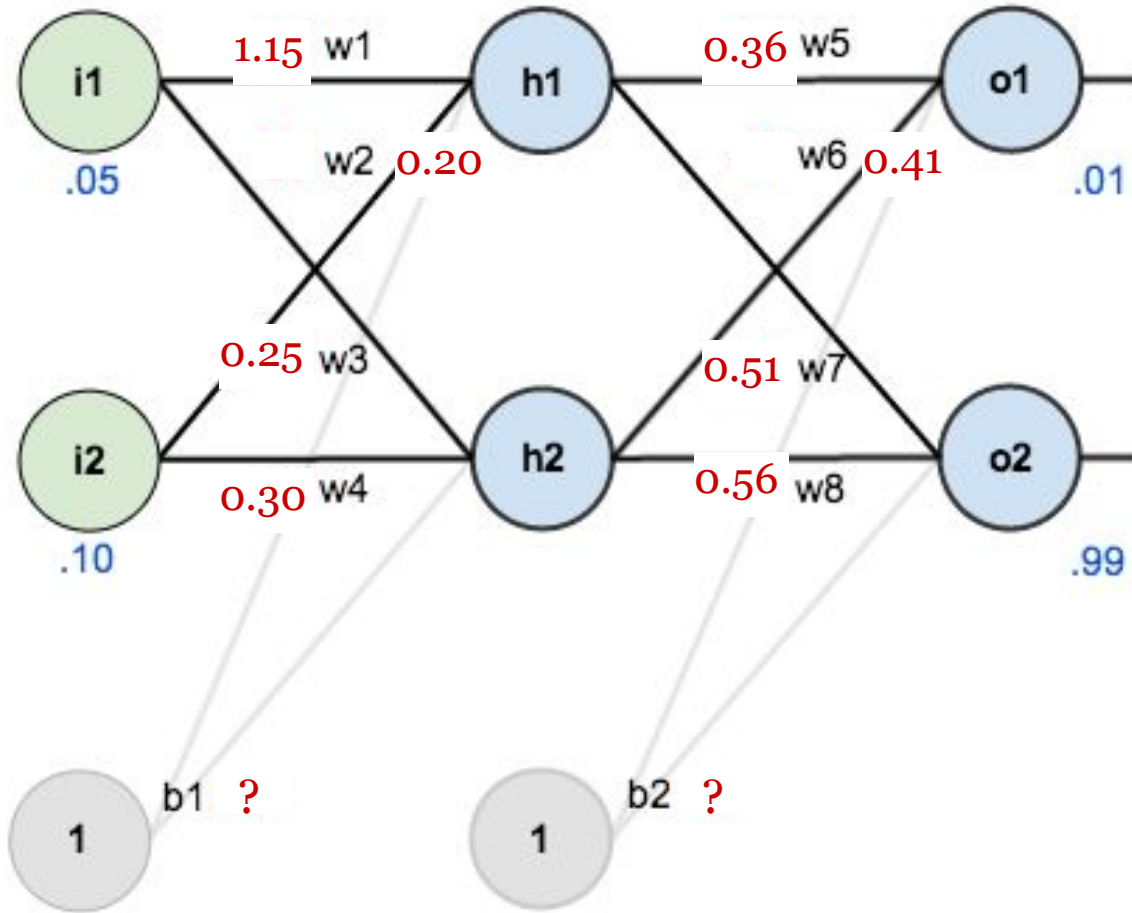
Repeat FeedForward and Backpropagation



Then **Feed Forward** Again

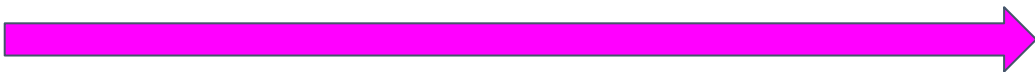
- Original error is about 0.298
- After 1st round update, error is about 0.291

Repeat FeedForward and Backpropagation

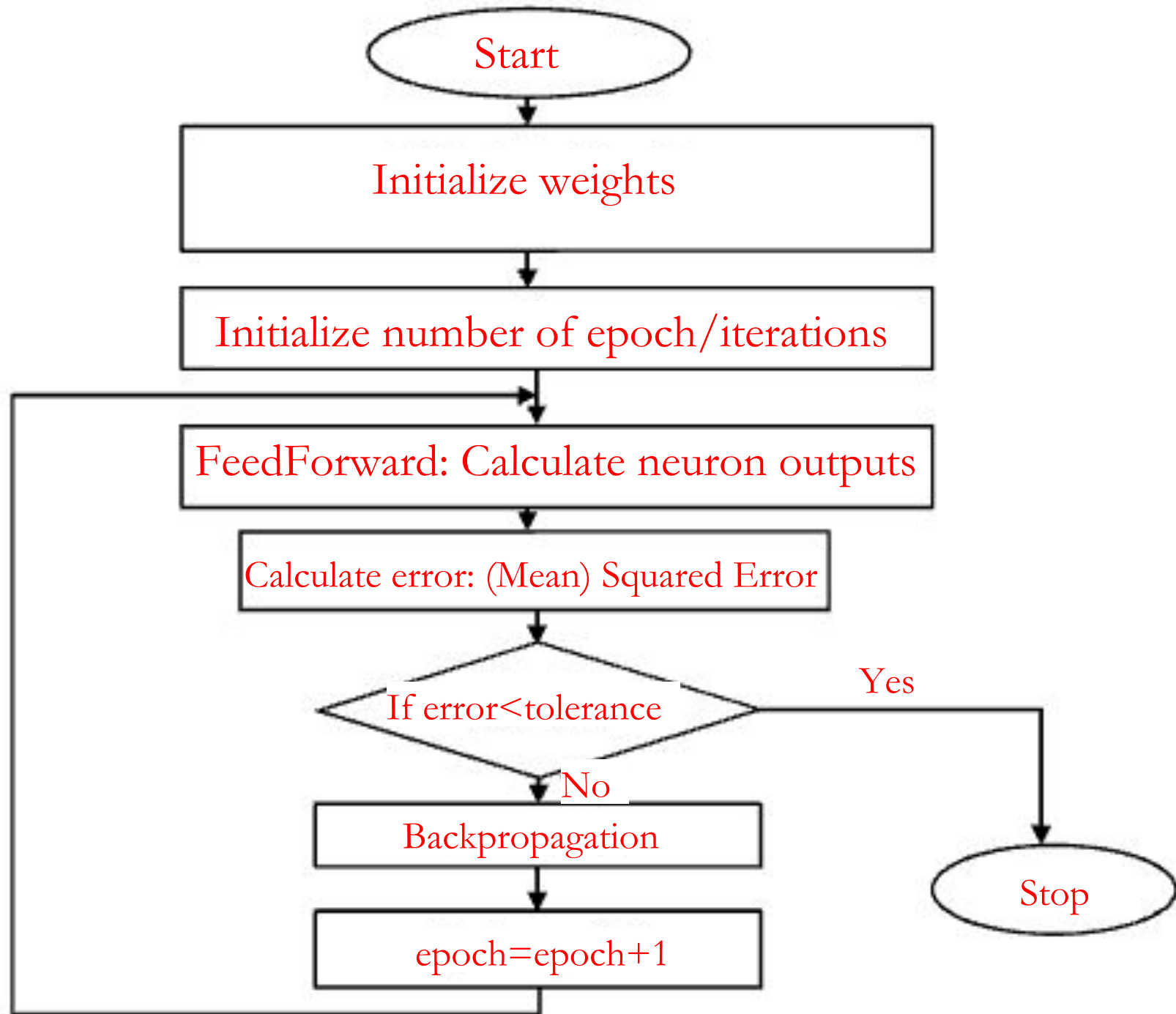
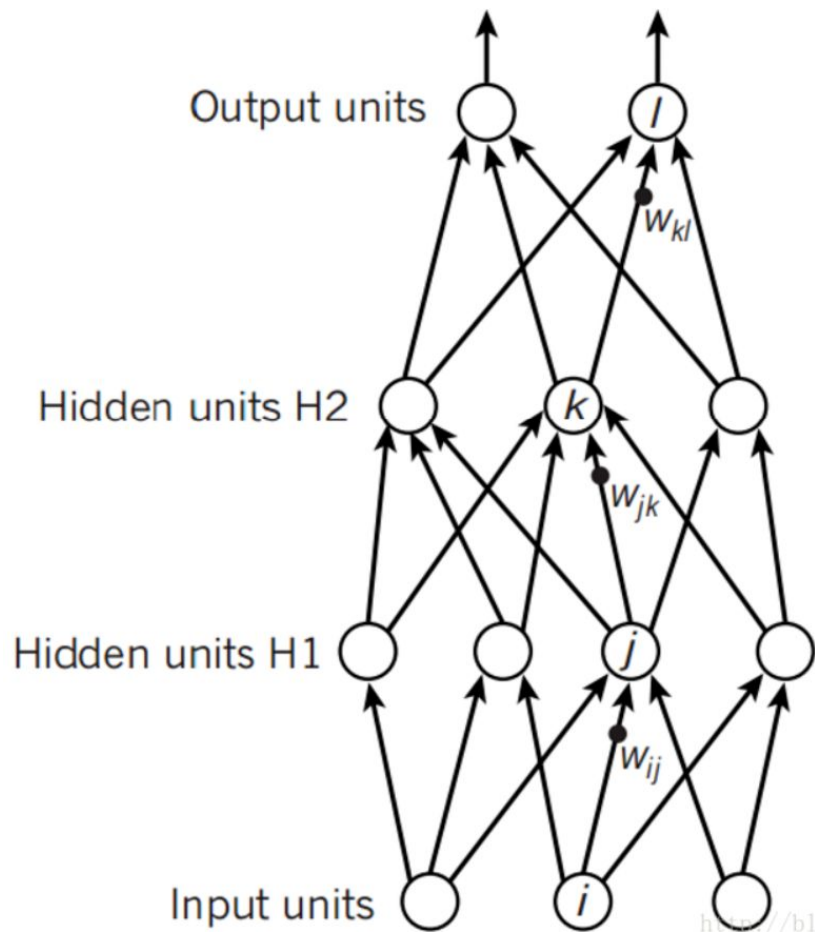


Repeat **FeedForward** and **Backpropagation**

- After 10,000 rounds update, error is only about $4e-5$
- Actual output for **o1** is 0.01;
Actual output for **o2** is 0.99
- Predicted output for **o1** is 0.016;
Predicted output for **o2** is 0.984



Neural Network: Feed Forward and Backpropagation



Programming Assignment 5

Make sure you install tensorflow and keras correctly in Python 3.5+ environment.

Programming Assignment 5

Using the BT2101 Tutorial 5 Notebook ([Deep Learning with Tensorflow and Keras.ipynb](#)), please answer the questions in the jupyter notebook

Answer all in the jupyter notebook.

Instructions

Submit Python Notebook to the submission folder and Named:
AXXXX_T5_program.ipynb

Include your answers in the jupyter notebook

- You need to show outputs, instead of just showing functions.

Submit by **Tuesday OCT-09** (by 12:00pm noon)

- Based on **Deep Learning with Tensorflow and Keras.ipynb**

Thank You!