



# MLPs + Backpropagation

CMSC 389A: Lecture 3

---

Sujith Vishwajith

University of Maryland

# Agenda

1. Recap
2. Intro to Neural Networks
3. Multi-layer Perceptron (MLP)
4. Backpropagation
5. Announcements

# Recap

---

# Recap

## Perceptrons

Linear classifier.

Either 1 or 0 (no probability indication).

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

Updates:

$$\mathbf{b} \leftarrow \mathbf{b} + \alpha * \mathbf{e}$$

$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha * \mathbf{e} * \mathbf{x}_i$$

## Logistic Regression

Linear classifier.

Indication of confidence (probability).

$$f(x) = \frac{1}{1 + e^{-x}}$$

Updates:

$$\mathbf{b} \leftarrow \mathbf{b} + \alpha * \mathbf{e} * \hat{y} * (1 - \hat{y})$$

$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha * \mathbf{e} * \mathbf{x}_i * \hat{y} * (1 - \hat{y})$$

# Intro to Neural Networks

---

# What are Neural Networks

Computing systems inspired by the biological brain

Learns by approximating functions that are unknown

Composed of artificial neurons (most basic one is the perceptron)

Input layers processed then passed to next layer one by one

# Types of Neural Networks

**Feedforward** (outputs only move in forward through the network):

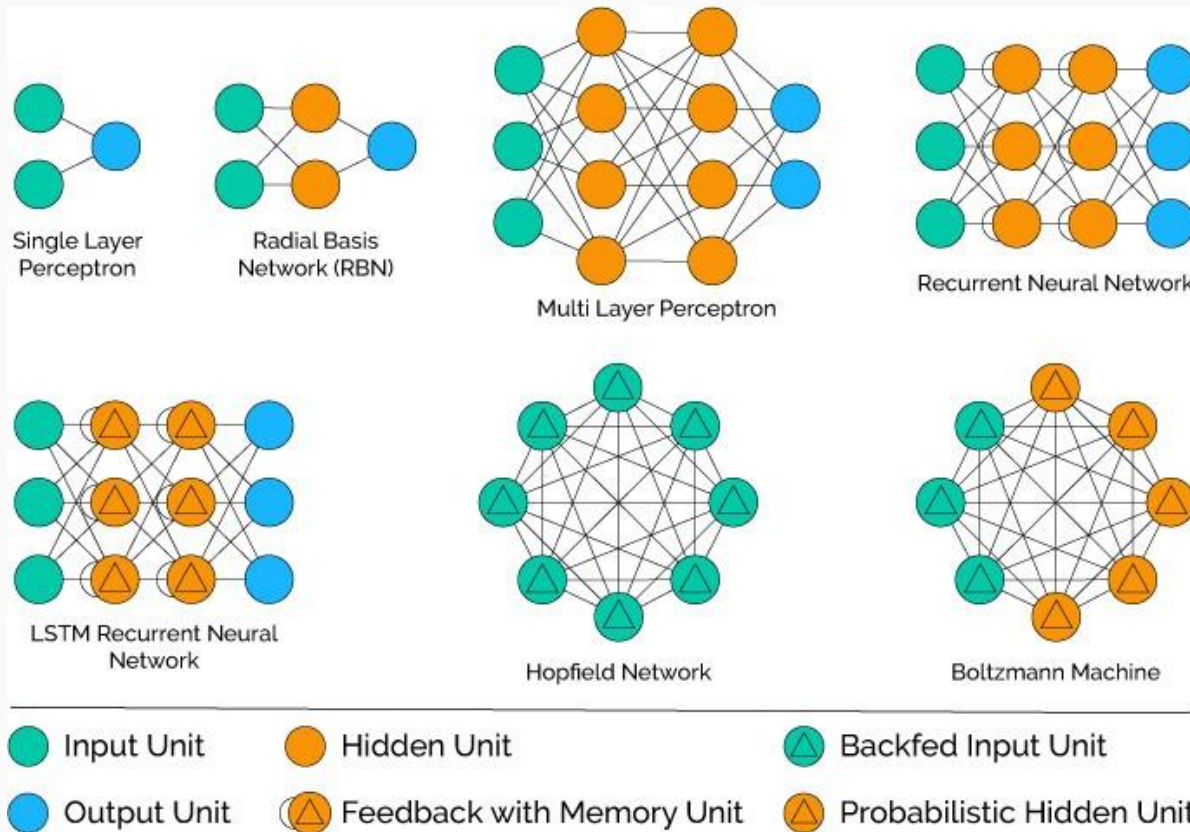
- Single Layer Network
- Multi-layer Perceptron
- Autoencoder
- Convolutional Neural Network

**Recurrent** (outputs are passed forwards and backward):

- Long-short term memory
- Bi-directional

Not limited to these examples

# Types of Neural Networks (cont.)





# Multi-layer Perceptron

---

# Inspiration

Remember that Perceptrons could only linearly separate data

Can we use more than one artificial neuron unit like a perceptron together?

What if we created multiple layers of such units?

Could we learn XOR and non-linear functions?

# Intro to MLPs

**M**ulti-**l**ayer **P**erceptron -> MLP

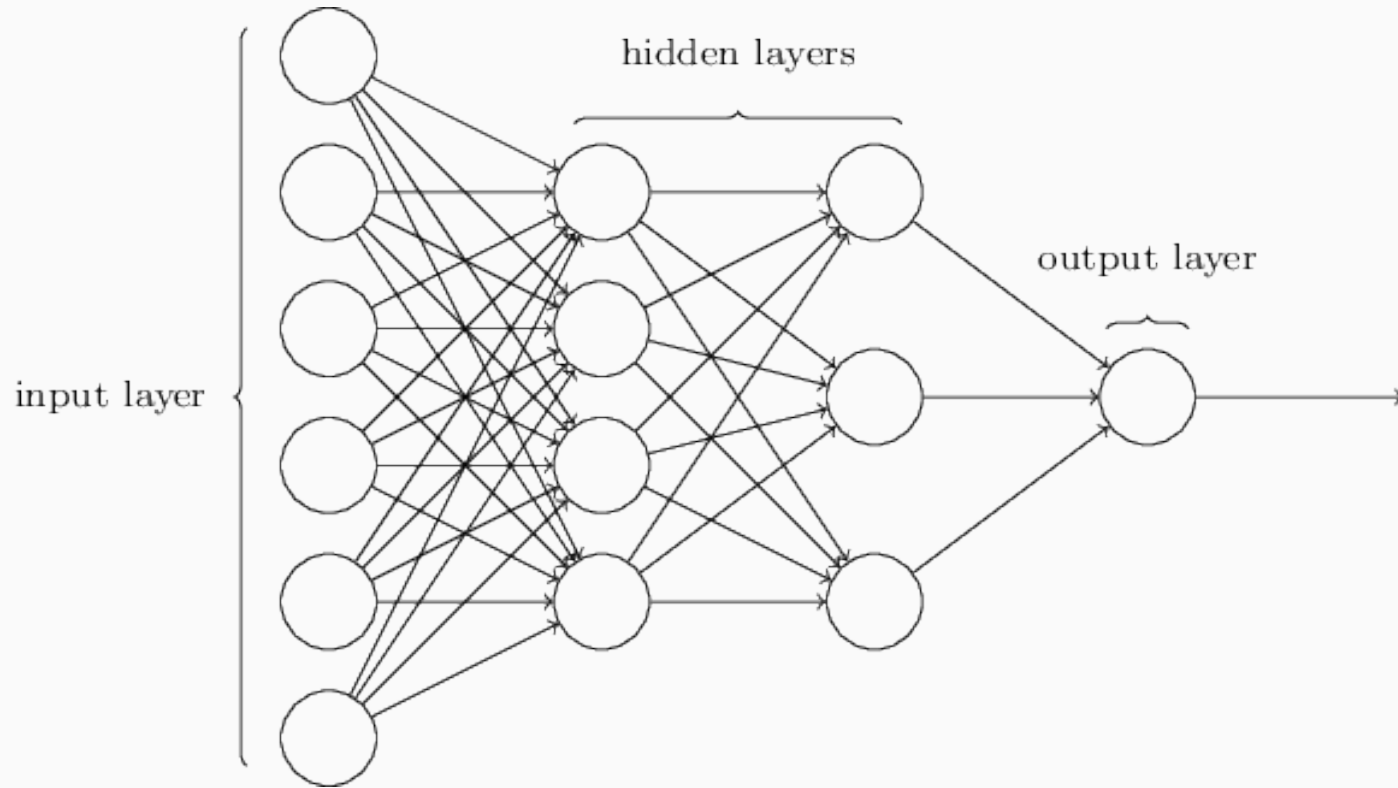
As the name suggests, MLPs are feed-forward neural networks with multiple layers of neurons

Used to be computationally difficult to train

In 1986 the Backpropagation algorithm was invented allowing us to train such networks

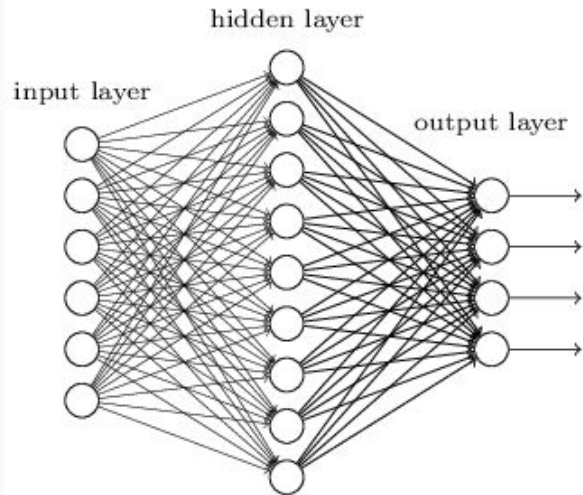
Called "deep" usually if multiple hidden layers

# Visualization

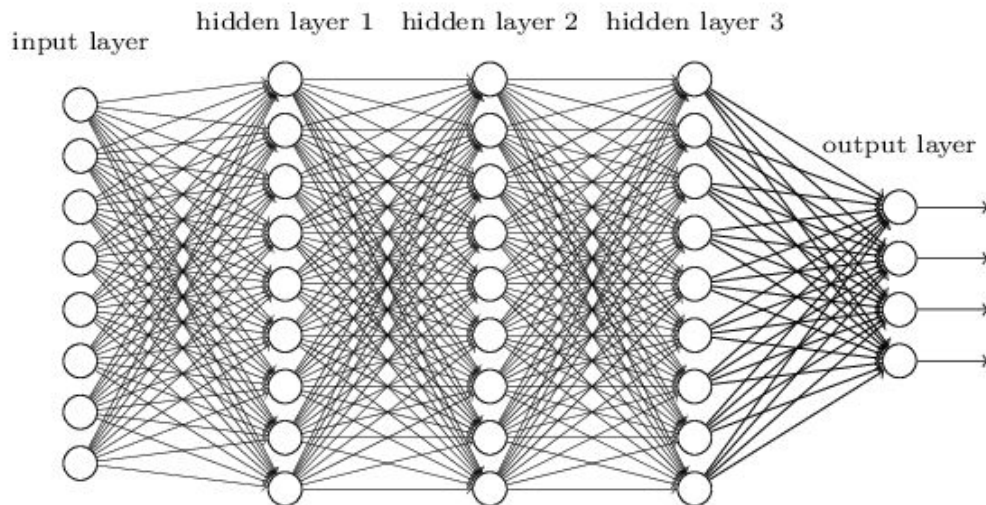


# Depth of Neural Network

"Non-deep" feedforward neural network



Deep neural network



# MLP Terminology

**Input Layer:** First layer which is the data input. Number of neurons is 1-1 with number of input features.

**Hidden Layer:** Layers between input layer and output layer used to encode and map the to the output.

**Output Layer:** Result of the network. Usually number of output values corresponds with the number of the classes or values you want. For example, having 10 output nodes could refer to probabilities of each class at an index.

**Activation Functions:** At each layer, you have a choice of what activation function the neurons on those layer should use.

# Artificial Neuron Unit

Most basic computational unit in a neural network

Similar to a logistic regression model or perceptron

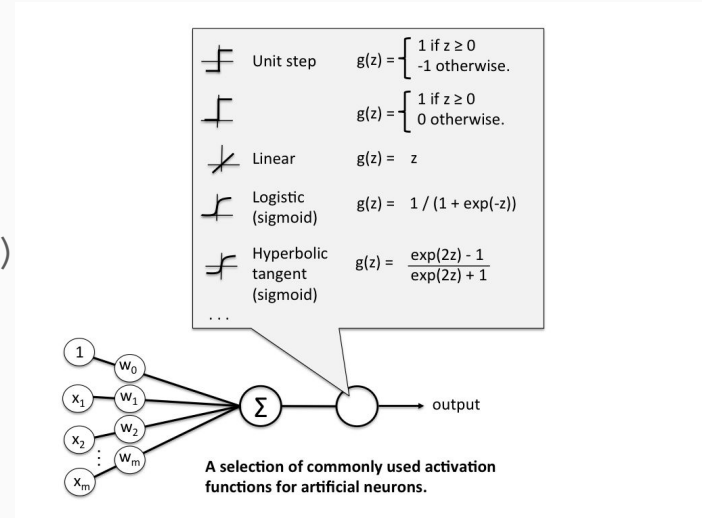
Takes in a series of inputs and bias node

Outputs a value between 0 (signal off) and 1 (signal on)

Bias node is always 1

Output is computed by an **activation function**

Activation function computes strength of signal



# Activation Functions

**Steps-side:** What we used for the perceptron. Simple to understand, signals are either on or off, and easy to train. However, not the best performance and lack of non-linearity.


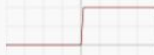


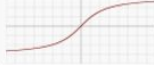




**Sigmoid:** What we used for logistic regression. Outputs a value between 0 and 1 based on an S-curve distribution. Allows for signals to be in the partially on phase.

**Hyperbolic Tangent:** Also known as tanh. A function which returns a value between -1 and 1 based on an S curve as well.

**ReLU:** Most popularly used activation function due to success. Not bounded under 1 unlike before.



# Activation Functions (cont.)

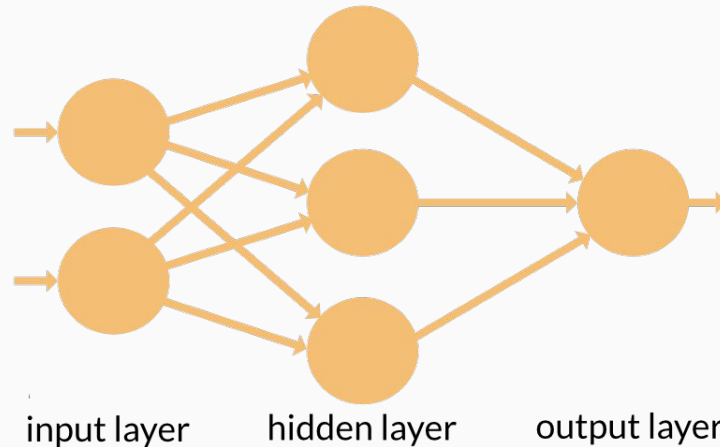
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) <sup>[2]</sup>		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) <sup>[3]</sup>		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

# Making Predictions

Each layers' neurons take in the output of the previous layer neuron as the **inputs**

We multiply the **inputs** and **bias** by the **weights** of the neuron

Pass the resulting value through an **activation function** and spit out an **output** which is then passed to the next layer



# Training

Initially the weights are randomly initialized to small random values

Normalize input data between 0 and 1

Goal is to learn weights  $\mathbf{w}$  for the network such that a mapping is learnt between the input  $\mathbf{x}$  and the desired output vector  $\mathbf{y}$ .

Trained using backpropagation.

# Backpropagation

---

# Stochastic Gradient Descent vs. Gradient Descent

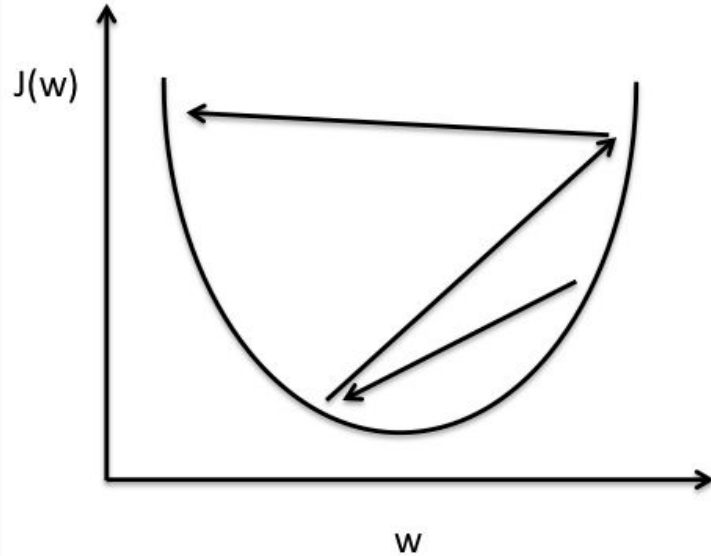
## Gradient Descent:

- Computes the gradients for every training example first
- Makes an update based off of the averaged gradient
- Since learning rate is small, it has to do this many times to converge
- Infeasible for large training data

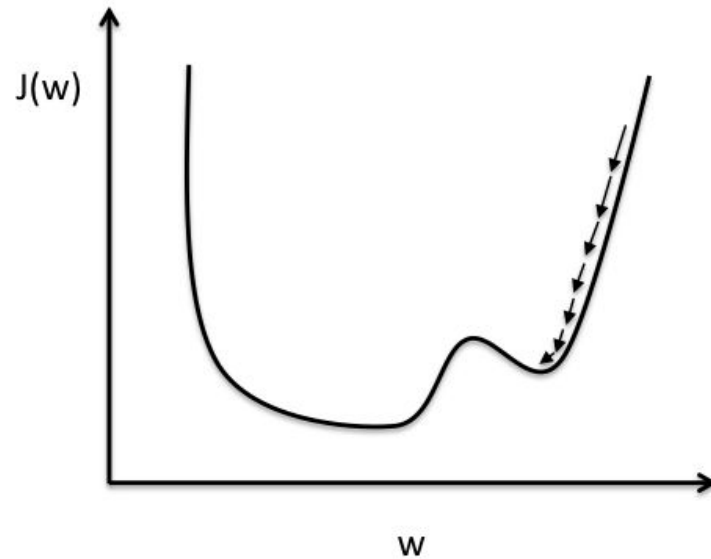
## Stochastic Gradient Descent:

- Computes the gradients for a random **batch** of examples
- Batch size can range from 1 to the size of training data
- Intuition is that a batch of training examples is representative enough of training data to make an informed learning step
- **Online learning** is a batch size of 1 while **Batch learning** is size  $> 1$

# Learning Rate



**Large learning rate: Overshooting.**



**Small learning rate: Many iterations until convergence and trapping in local minima.**

# Backpropagation Intuition

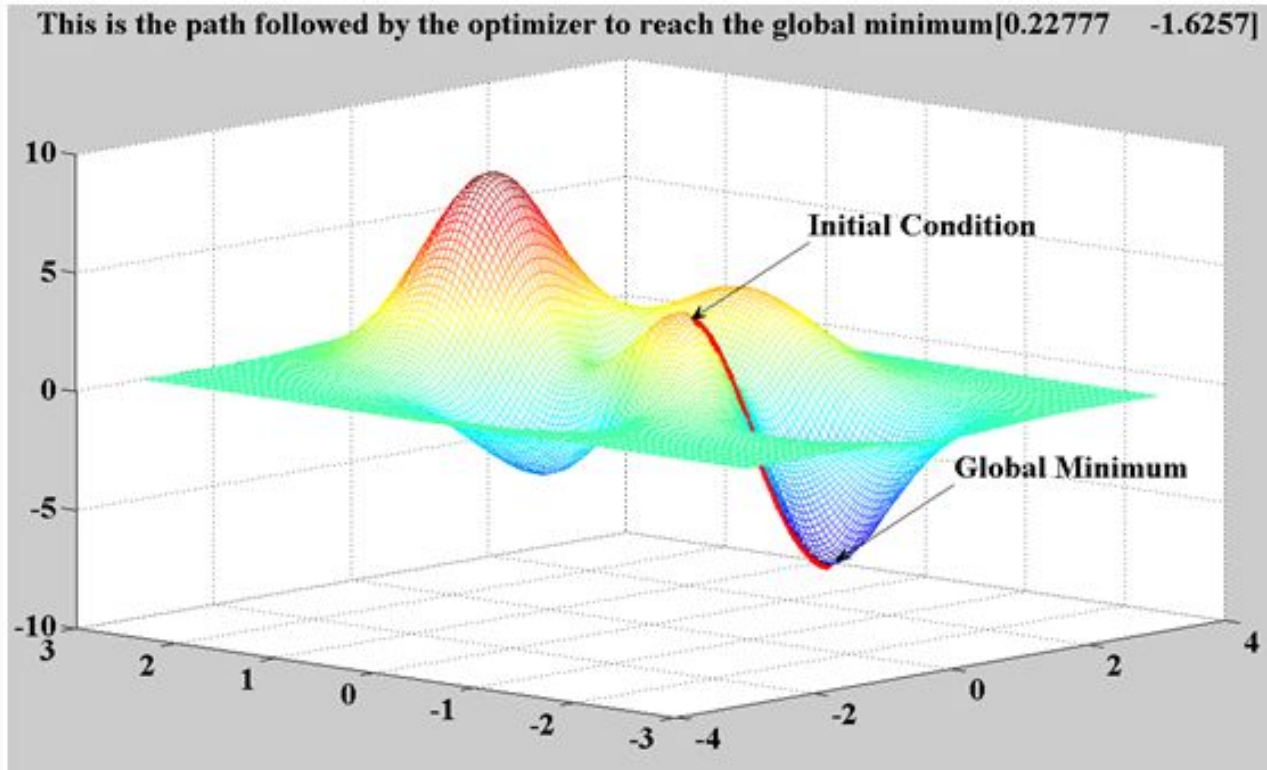
Goal: Propagate weight updates for each layer to minimize a loss function

We want to compare our output with the error and update the weights backward through the network.

The neurons closest to the output have the most impact so their weight is adjusted more compared to the initial neurons.

Based off of the chain rule since predictions follow a chain of neurons whose change in weights affect each other neurons in the chain

# Visualization





# Single Hidden Layer Update

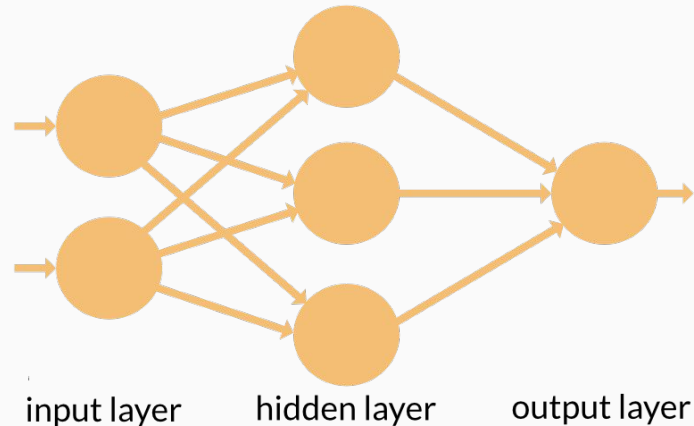
Output nodes: **error = (expected - output) \* derivative (output)**

Hidden nodes: **error = (weight<sub>k</sub> - error<sub>j</sub>) \* derivative (output)**

Where **weight<sub>k</sub>** is the weight of **k<sup>th</sup>** neuron to current neuron and **error<sub>j</sub>** is error signal propagated back from **j<sup>th</sup>** output neuron

Update: **weight += α \* error \* input**

Derivative of sigmoid is **1 \* (1-output)**

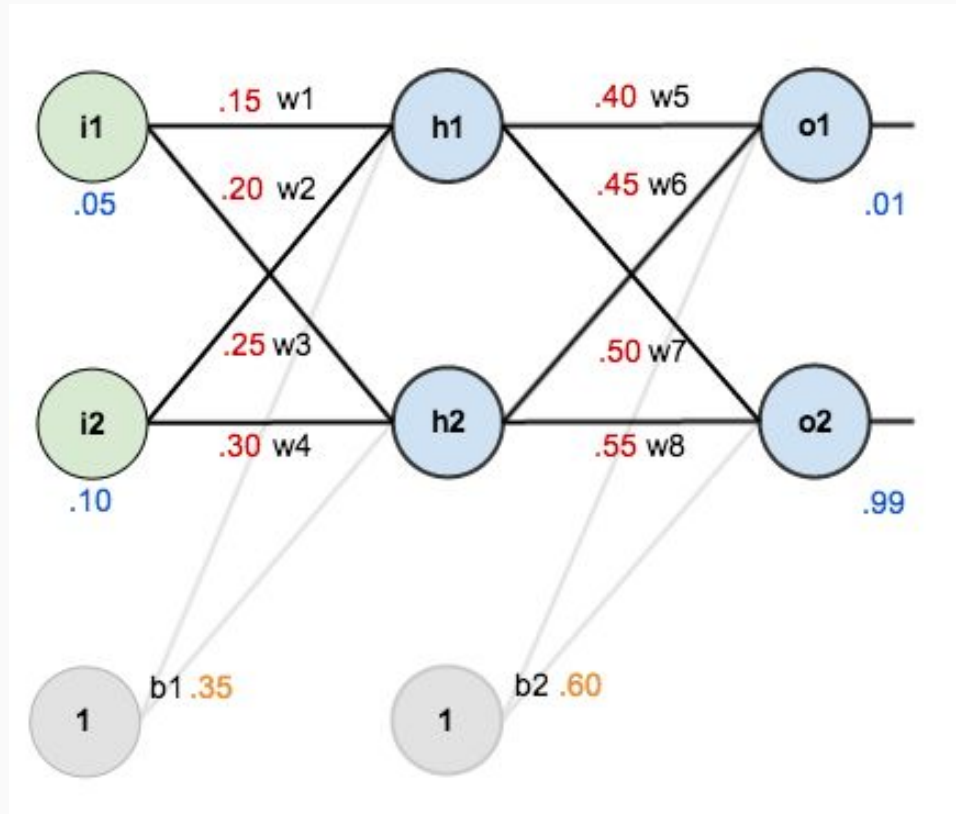


# Let's do an example!

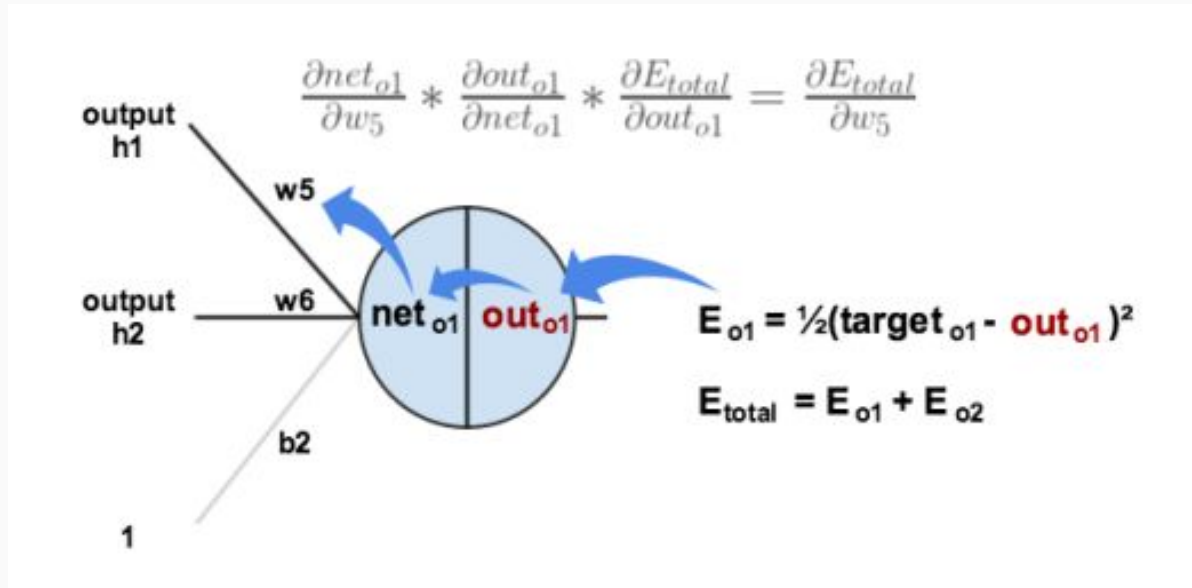
Example from:

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

# Initial Network



# Output Layer Update

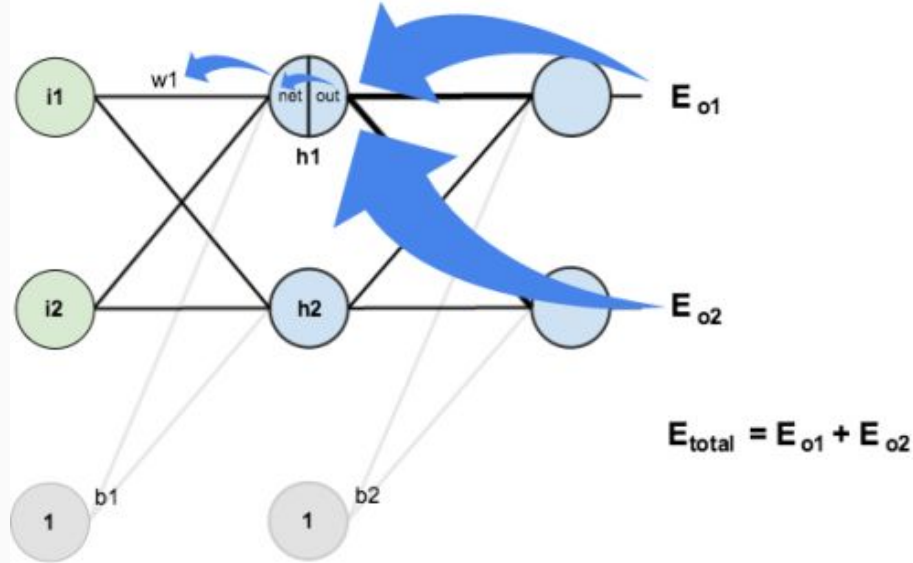


# Hidden Layer Update

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



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



# Announcements

---

# Announcements

Please complete weekly feedback.

No class next week!

Practical 1 is due next week on **February 16th** at **11:59 p.m.**

Questions?