MSA 8650 Deep Learning

fall Semester, 2019

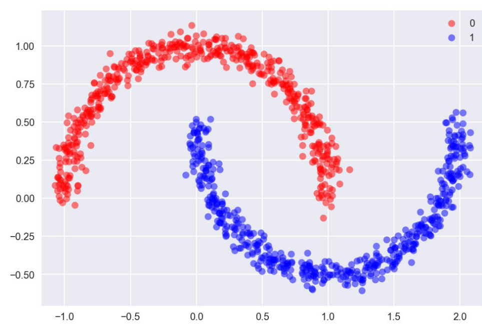
1

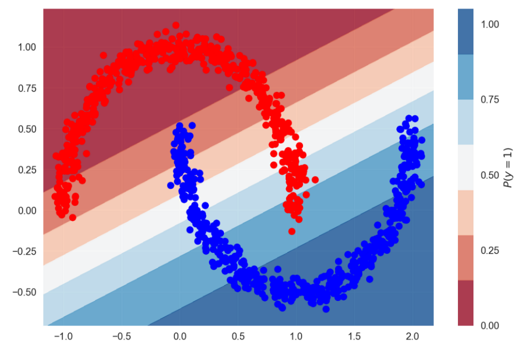
Feedforward Neural Networks

2

Robinson

Moon Data Binary Classification



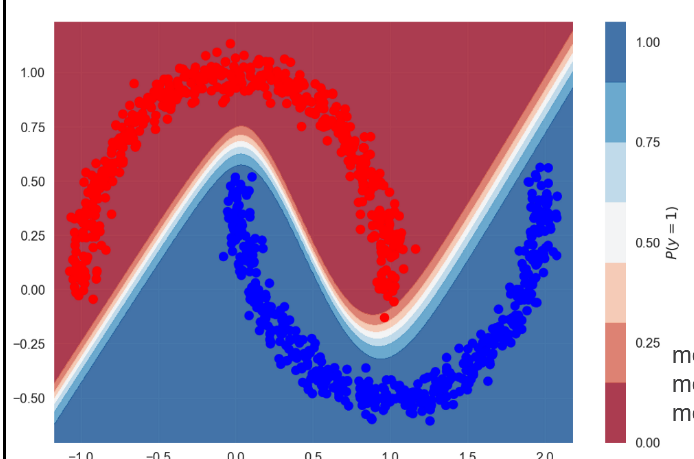


Logistic regression model on this dataset can get 86% accuracy.

https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

Moon Data Binary Classification

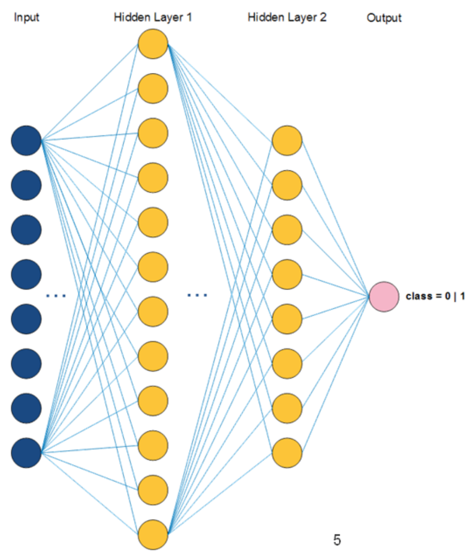
A neural network with 3 layers can achieve 100% accuracy.



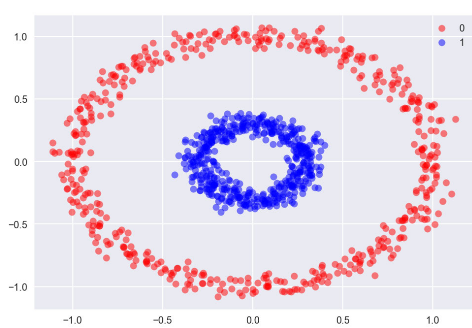

```
model.add(Dense(4, input_shape=(2,), activation='tanh'))
model.add(Dense(2, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
```

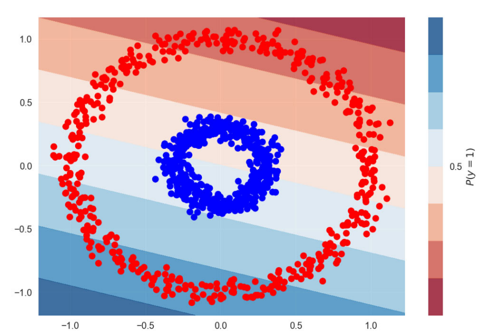

https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6 model.add(Dense(12, input_dim=8, init='uniform', activation='relu'))

It means 8 input parameters, with 12 neurons in the FIRST hidden layer.



Circles Data Binary Classification

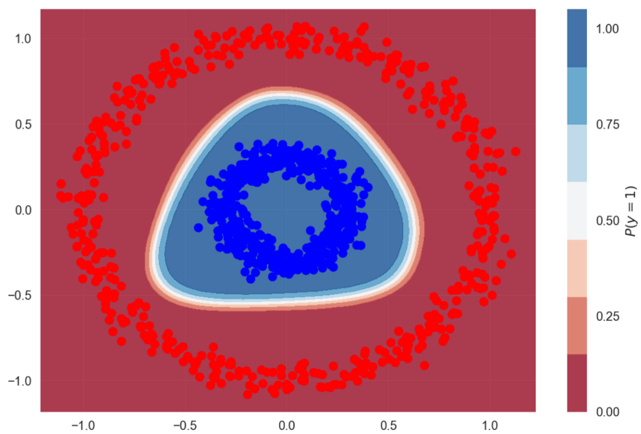




Logistic regression model on this dataset can get 50% accuracy.

https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

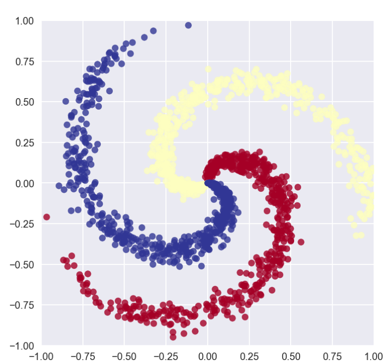
Circles Data Binary Classification



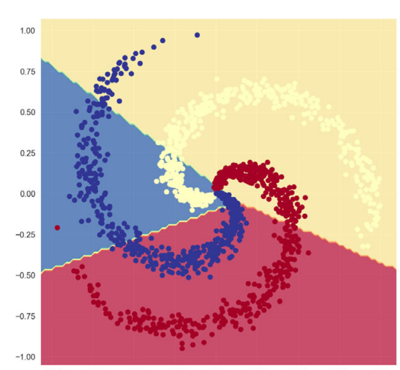
A neural network with 3 layers can achieve 100% accuracy.

```
model.add(Dense(4, input_shape=(2,), activation='tanh'))
model.add(Dense(2, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
```


Multiclass Classification

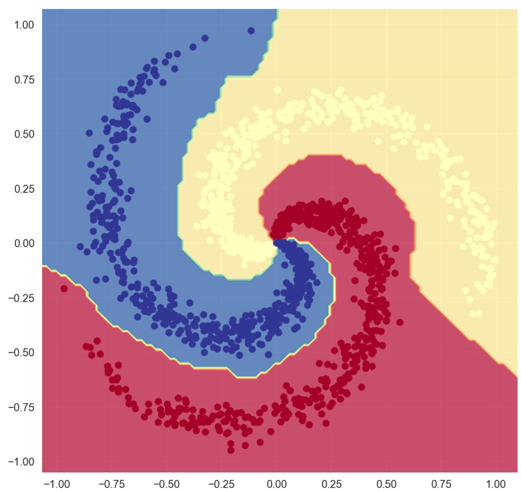


Classification for 3 classes



Softmax regression model on this dataset can get about 50% accuracy.

Multiclass Classification



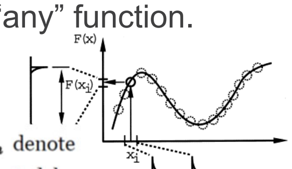
A neural network with 3 layers can achieve 99% accuracy

```
model.add(Dense(64, input_shape=(2,), activation='tanh'))
model.add(Dense(32, activation='tanh'))
model.add(Dense(16, activation='tanh'))
model.add(Dense(3, activation='softmax'))
```


https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

Multilayer Neural Network

- Without knowing the shape and other details of high dimensional input, multilayer neural networks can be used as they can compute "any" function.



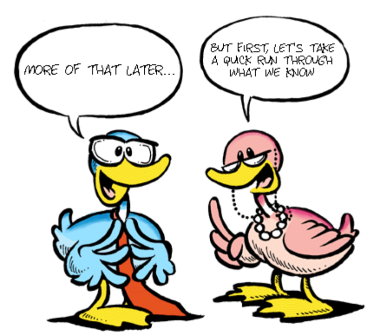
- Universal approximation theorem:

Let $\varphi(\cdot)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let I_m denote the m-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$, where $i = 1, \cdots, N$, such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \varphi\left(w_i^T x + b_i\right)$$

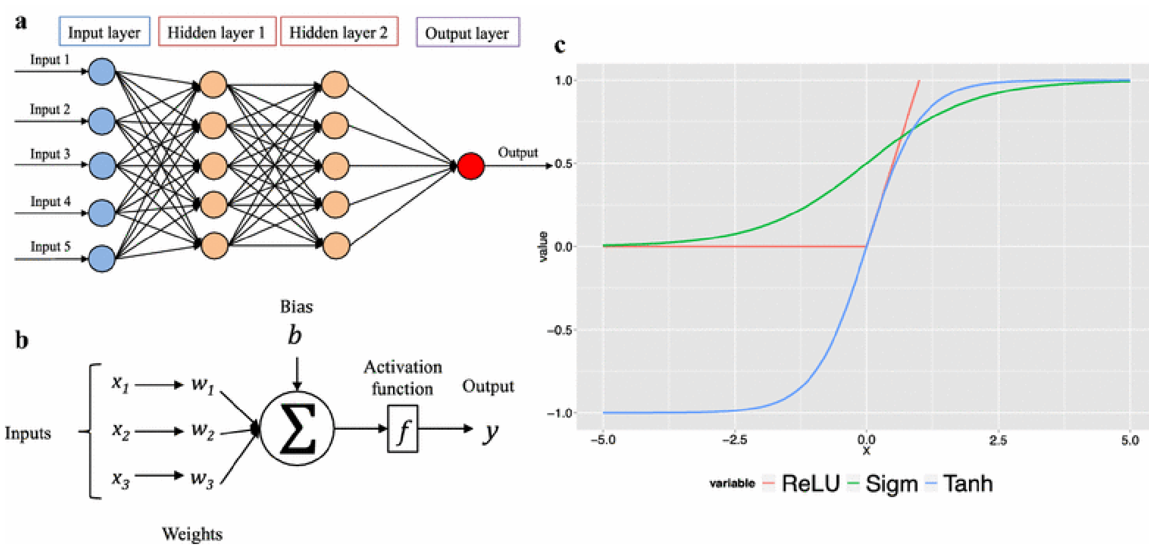
as an approximate realization of the function f where f is independent of φ; that is,

$$|F(x) - f(x)| < \varepsilon$$



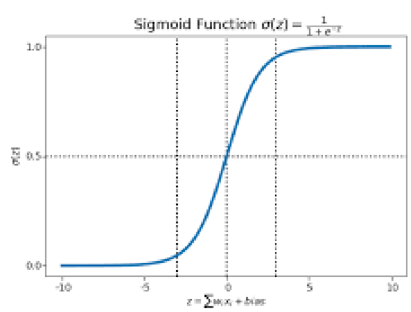
https://en.wikipedia.org/wiki/Universal_approximation_theorem
http://neuralnetworksanddeeplearning.com/chap4.html

How to Generate Nonlinear Functions?

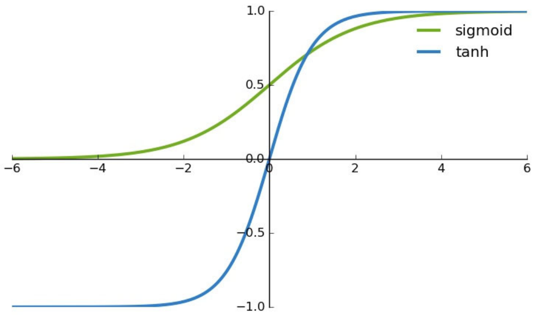


Activation Function: Sigmoid

- Takes a real-valued number and “squashes” it into range between 0 and 1.
- Sigmoid neurons saturate and kill gradients, thus NN will barely learn when the neuron’s activation are 0 or 1 (saturate)
  - ❖ gradient at these regions almost zero
  - ❖ almost no signal will flow to its weights
  - ❖ if initial weights are too large then most neurons would saturate



Activation Function: Tanh



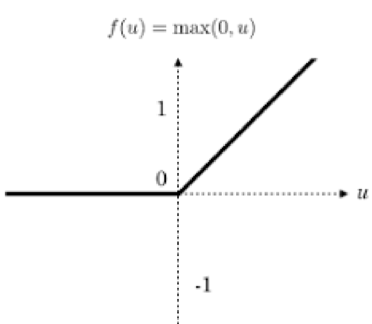
$$\sinh x = \frac{e^x - e^{-x}}{2}$$

$$\cosh x = \frac{e^x + e^{-x}}{2}$$

$$\tanh x = \frac{\sinh}{\cosh} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Takes a real-valued number and "squashes" it into range between -1 and 1.
- Like sigmoid, $\tanh$ neurons saturate
- Unlike sigmoid, output is zero-centered
- Tanh is a scaled sigmoid: $\tanh(x) = 2\sigma(2x) - 1$

Activation Function: ReLU



Rectified Linear Unit

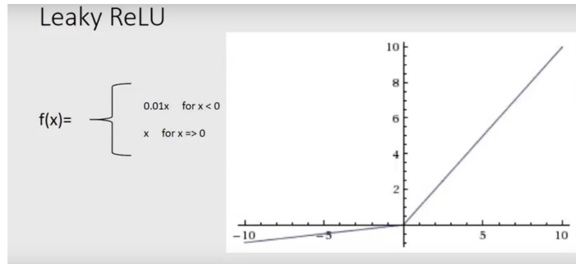
Takes a real-valued number and thresholds it at zero

Most Deep Networks use ReLU nowadays

- Trains much **faster**
  - ❖ accelerates the convergence of SGD
  - ❖ due to linear, non-saturating form
- Less expensive operations
  - ❖ compared to sigmoid/tanh (exponentials etc.)
  - ❖ implemented by simply thresholding a matrix at zero
- Prevents the **gradient vanishing problem**

Activation Function: Leaky ReLU

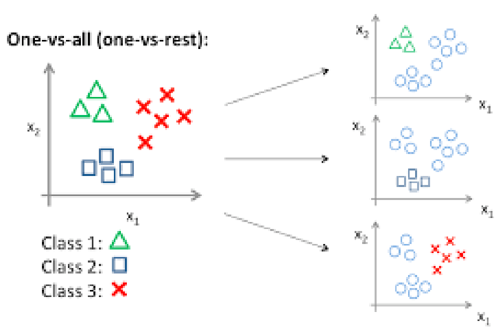
The only difference between **ReLu** and **Leaky ReLu** is that it does not completely vanishes the negative part, it just lower its magnitude.



Leaky ReLU above won't have the issue of units "die" as there is no active point (negative).

Although (leaky) ReLU is not differentiable at zero, it is not a problem since we can return one sided derivatives at zero and gradient based optimization is subject to numerical error anyway.

Activation Function: SoftMax



Multiple classes

$$\boxed{\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}}$$

All Activation Functions

A complete list of activation functions can be found as follows,

https://stats.stackexchange.com/questions/115258/comprehensive-list-of-activation-functions-in-neural-networks-with-pros-cons

Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \le -\frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} \le z \le \frac{1}{2} \\ 1 & z \ge \frac{1}{2} \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

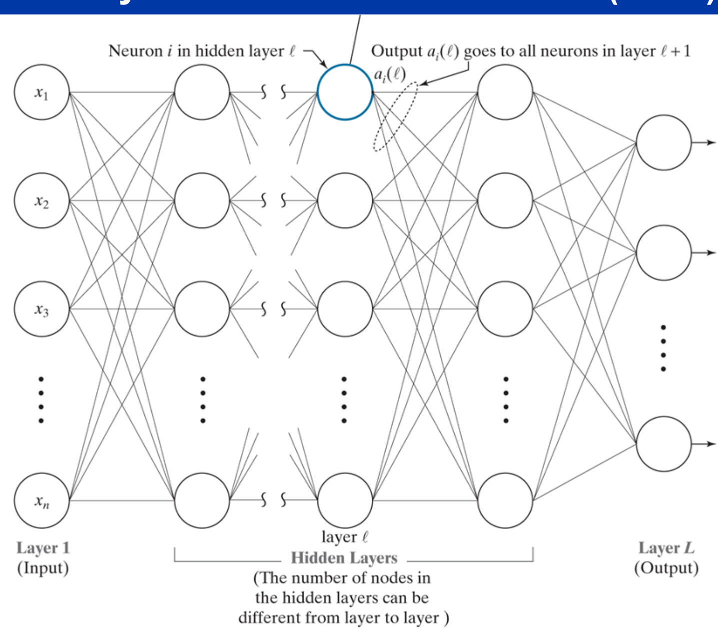
Georgia State University J. Mack Robinson College of Business

Robinson

Multilayer Feedforward Neural Net(work)

General model of a feedforward, fully connected neural net

how the output of each neuron goes to the input of all neurons in the following layer, hence the name fully connected for this type of architecture.

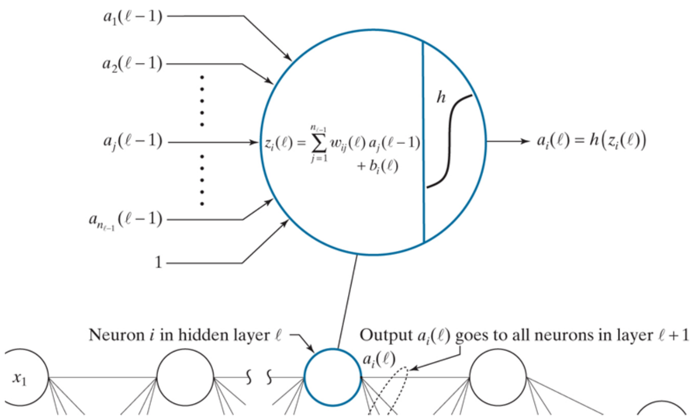


we call a neural net with a single hidden layer a shallow neural network, and refer to network with two or more hidden layers as a **deep** neural network

Georgia State University J. Mack Robinson College of Business

Robinson

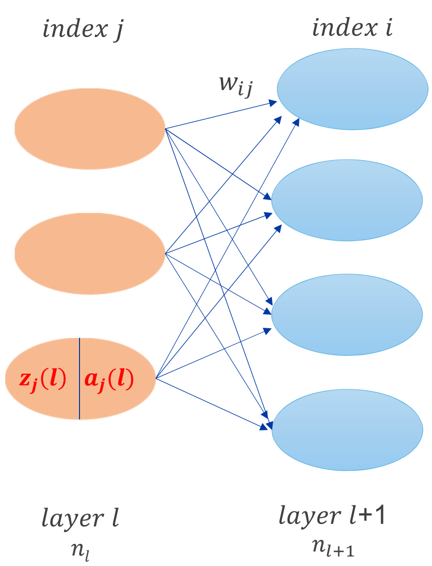
Multilayer Feedforward Neural Net(work)



- w_{ij} the weight that associates the link connecting the output of neuron j to the input of neuron i.
- That is, the first subscript denotes the neuron that receives the signal, and the second refers to the neuron that sends the signal.
- We use the notation as stated for easy matrix representation.

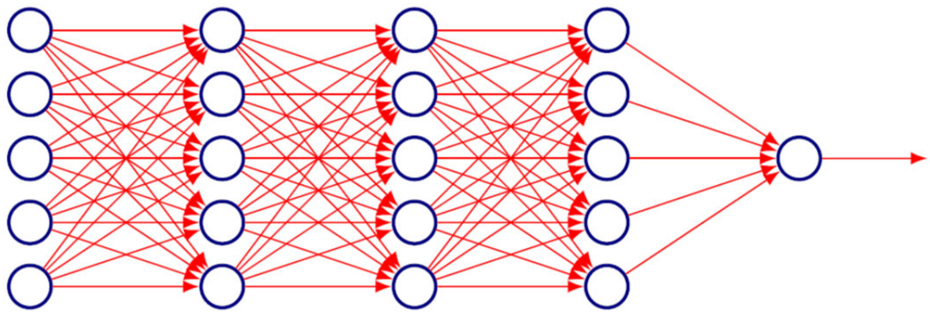
The biases depend only on the neuron containing it, a single subscript that associates a bias with a neuron is sufficient. For example, we use b_i to denote the bias value associated with the ith neuron in a given layer of the network.

Multilayer Feedforward Neural Net(work)



- Let l denote a layer in the network, for $l = 1, 2, \dots, L$.
- $l = 1$ denotes the input layer, $l = L$ is the output layer, and all other values of l denote hidden layers.
- The number of neurons in layer l is denoted n_l. We have two options to include layer indexing in the parameters of a neural network. We can do it as a superscript, for example, w_{ij}^l and b_i^l; or we can use the notation $w_{ij}(l)$ and $b_i(l)$.
- Using the second notation, the output (activation value) of neuron j in layer l is denoted $a_j(l)$.

Multilayer Feedforward Neural Networks

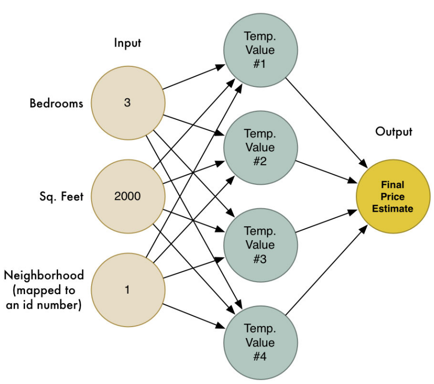


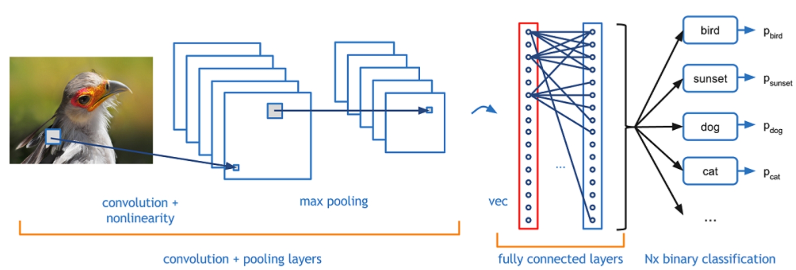
Function f is a composition of many different functions

e.g. $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$

$f^{(1)}$ is the first layer, f^2 the second layer and so on.

Multilayer Feedforward Neural Networks

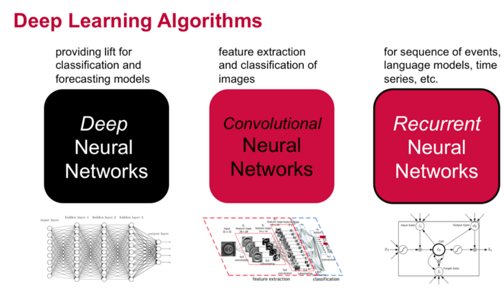




Different output types

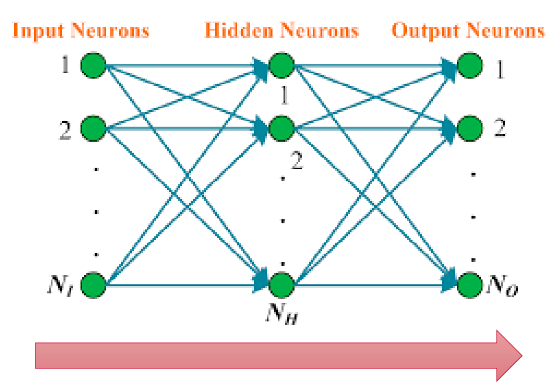
How to Solve a Deep Neural Network?

- Forward propagate to get the output and compare it with the real value to get the error.
- to minimize the error, you propagate backwards by finding the derivative of error with respect to each weight and then subtracting this value from the weight value.



Multilayer Feedforward Neural Net(work)—Forward Pass

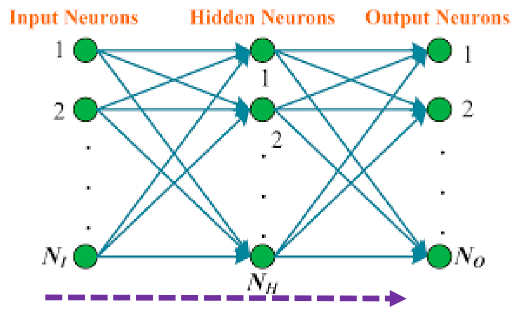
A forward pass through a neural network maps the input layer (i.e., values of x) to the output layer. The values in the output layer are used for determining the class of an input vector or make predictions.



Robinson

Multilayer Feedforward Neural Net(work)—Forward Pass

The outputs of the layer 1 are the components of input vector x: $a_j(1) = x_j$ $j = 1,2,\dots,n_1$ where $n_1 = n$ is the dimensionality of x.



The value of network output node i is

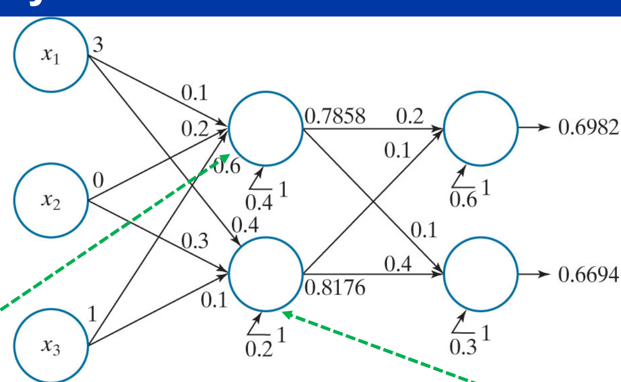
$a_{i(L)} = h(z_{i(L)})$
$i = 1,2,\dots,n_L$

The computation performed by neuron i in layer l is given by

- $z_i(l) = \sum_{j=1}^{n_{l-1}} w_{ij}(l)a_j(l-1) + b_i(l)$ for $i = 1,2,\dots,n_l$ and $l = 2,\dots,L$.
- $z_i(l)$ is formed using all outputs from layer $l-1$ and is called the net (or total) input to neuron i in layer l, The output (activation value) of neuron i in layer l is given by $a_i(l) = h(z_i(l))$ $i = 1,2,\dots,n_l$ where h is an activation function.

Multilayer Feedforward Neural Network: Forward Pass

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix}$$



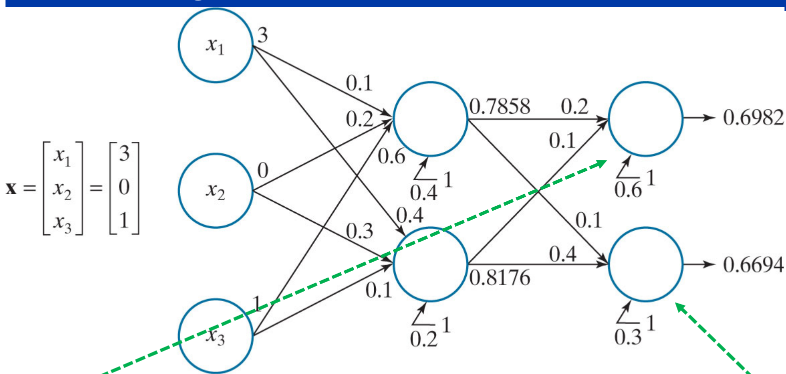
example

A small, fully connected, feedforward net with labeled weights, biases, and outputs. The activation function is a sigmoid.

$$z_1(2) = (0.1)\times 3 + (0.2)\times 0 + (0.6)\times 1 + 0.4 = 1.3$$
$$a_1(2) = h(z_1(2)) = \frac{1}{1+e^{-1.3}} = 0.7858$$

$$z_2(2) = (0.4)\times 3 + (0.3)\times 0 + (0.1)\times 1 + 0.2 = 1.5$$
$$a_2(2) = h(z_2(2)) = \frac{1}{1+e^{-1.5}} = 0.8176$$

Multilayer Feedforward Neural Network: Forward Pass



$$z_1(3) = (0.2) \times 0.7858 + (0.1) \times 0.8176 + 0.6 = 0.8389$$

$$a_1(3) = h\big(z_1(3)\big) = \frac{1}{1 + e^{-0.8389}} = 0.6982$$

$$z_2(3) = (0.1) \times 0.7858 + (0.4) \times 0.8176 + 0.3 = 0.7056$$

$$a_2(3) = h\big(z_2(3)\big) = \frac{1}{1 + e^{-0.7056}} = 0.6694$$

Multilayer Feedforward Neural Network: Forward Pass

Matrix Representation

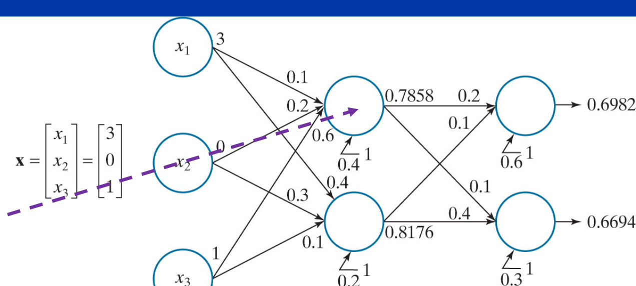
$\mathbf{a}(1) = \mathbf{x}$

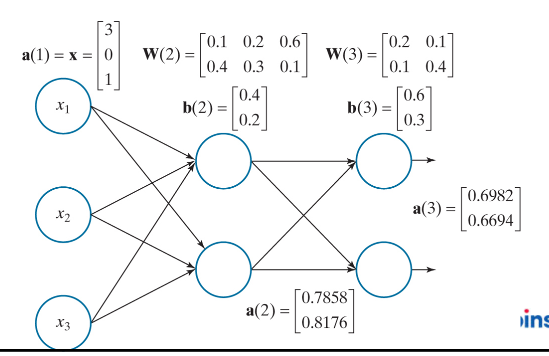
Each row corresponding a feeding layer,
each column corresponding the feeded layer

$$\mathbf{W}(l) = \begin{bmatrix} w_{11}(l) & w_{12}(l) & \cdots & w_{1n_{l-1}}(l) \\ w_{21}(l) & w_{22}(l) & \cdots & w_{2n_{l-1}}(l) \\ \vdots & \vdots & & \vdots \\ w_{n_1 1}(l) & w_{n_1 2}(l) & \cdots & w_{n_1 n_{l-1}}(l) \end{bmatrix}$$

$$\mathbf{z}(l) = \mathbf{W}(l)\mathbf{a}(l-1) + \mathbf{b}(l) \quad l = 2, 3, \cdots, L$$

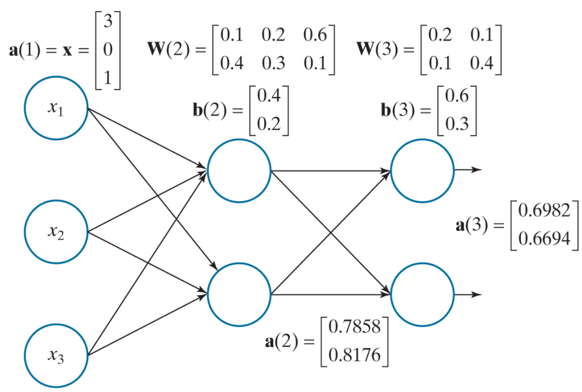
$$\mathbf{a}(l) = h[\mathbf{z}(l)] = \begin{bmatrix} h(z_1(l)) \\ h(z_2(l)) \\ \vdots \\ h(zl(l)) \end{bmatrix}$$





Multilayer Feedforward Neural Network: Forward Pass

Matrix Operations



$$\mathbf{z}(2) = \mathbf{W}(2)\mathbf{a}(1) + \mathbf{b}(2)$$
$$= \begin{bmatrix} 0.1 & 0.2 & 0.6 \\ 0.4 & 0.3 & 0.1 \end{bmatrix} \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.4 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 1.3 \\ 1.5 \end{bmatrix}$$

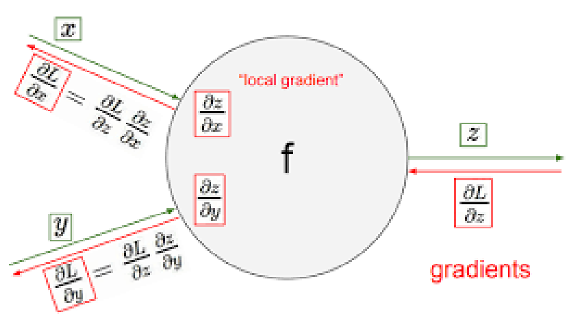
$$\mathbf{a}(2) = h[\mathbf{z}(2)] = \begin{bmatrix} h(1.3) \\ h(1.5) \end{bmatrix} = \begin{bmatrix} 0.7858 \\ 0.8176 \end{bmatrix}$$

$$\mathbf{z}(3) = \mathbf{W}(3)\mathbf{a}(2) + \mathbf{b}(3)$$
$$= \begin{bmatrix} 0.2 & 0.1 \\ 0.1 & 0.4 \end{bmatrix} \begin{bmatrix} 0.7858 \\ 0.8176 \end{bmatrix} + \begin{bmatrix} 0.6 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 0.8389 \\ 0.7056 \end{bmatrix}$$

$$\mathbf{a}(3) = h[\mathbf{z}(3)] = \begin{bmatrix} h(0.8389) \\ h(0.7056) \end{bmatrix} = \begin{bmatrix} 0.6982 \\ 0.6694 \end{bmatrix}$$

Multilayer Feedforward Neural Net(work)- Backpropagation

A neural network is defined completely by its weights, biases, and activation function. Training a neural network refers to using one or more sets of training patterns (inputs) to estimate these parameters.



The Chain Rule

$f = f(g)\,;\ \ g = g(x)$

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}$$

Main Tool

Multilayer Feedforward Neural Net(work)- Backpropagation

The activation values of neuron j in the output layer is $a_j(L)$. We define the error of that neuron as

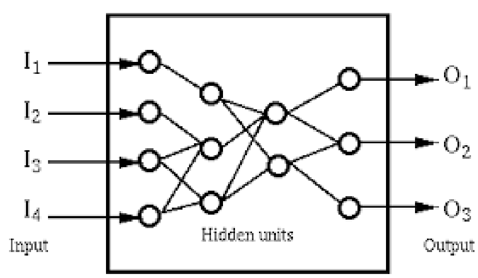
$$E_j = \frac{1}{2}\left(r_j - a_j(L)\right)^2$$

for $j = 1, 2, \ldots, n_L$, where r_j is the desired response of output neuron $a_j(L)$ for a given input $\mathbf{x}$.

The output error with respect to a single $\mathbf{x}$ is the sum of the errors of all output neurons with respect to that vector:

$$E = \sum_{j=1}^{n_L} E_j = \frac{1}{2}\sum_{j=1}^{n_L}\left(r_j - a_j(L)\right)^2 = \frac{1}{2}\|\mathbf{r} - \mathbf{a}(L)\|^2$$

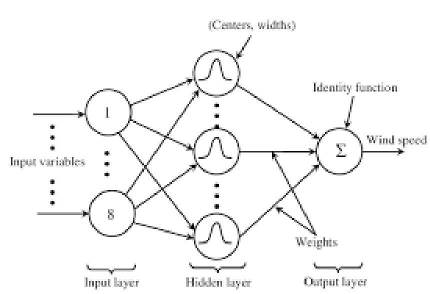
Multilayer Feedforward Neural Net(work)- Backpropagation



$$E = \sum_{j=1}^{n_L} E_j = \frac{1}{2}\sum_{j=1}^{n_L}\left(r_j - a_j(L)\right)^2 = \frac{1}{2}\|\mathbf{r} - \mathbf{a}(L)\|^2$$

$$a_j(L) = h(z_j(L))$$

This is an example ☺



For the output layer:

$$\delta_j(L) = \frac{\partial E}{\partial z_j(L)} = \frac{\partial E}{\partial a_j(L)}\frac{\partial a_j(L)}{\partial z_j(L)} = \frac{\partial E}{\partial a_j(L)}\frac{\partial h(z_{j(L)})}{\partial z_j(L)}$$
$$= (a_j(L) - r_j)h'(z_j(L))$$

Multilayer Feedforward Neural Net(work)- Backpropagation

$$E = \sum_{j=1}^{n_L} E_j = \frac{1}{2}\sum_{j=1}^{n_L}(r_j - a_j(L))^2 = \frac{1}{2}\|\mathbf{r} - \mathbf{a}(L)\|^2$$

$$z_i(l) = \sum_{j=1}^{n_{l-1}} w_{ij}(l)a_j(l-1) + b_i(l) \quad i = 1, 2, \cdots, n_l$$

$$a_i(l) = h\big(z_i(l)\big) \quad i = 1, 2, \cdots, nl$$

$$f'(x) = (\,g(h(x))\,)' = g'(\,h(x)\,)\,h'(x)$$

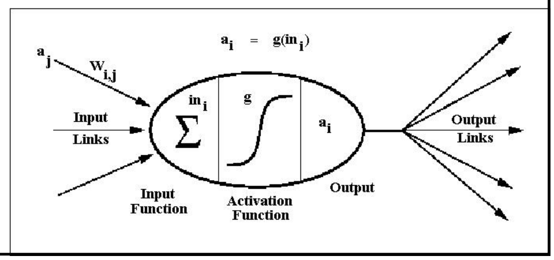
- keep the inside
- take derivative of outside

multiply by derivative of the inside

For the previous layers: $l = L-1, L-2, \cdots, 2$

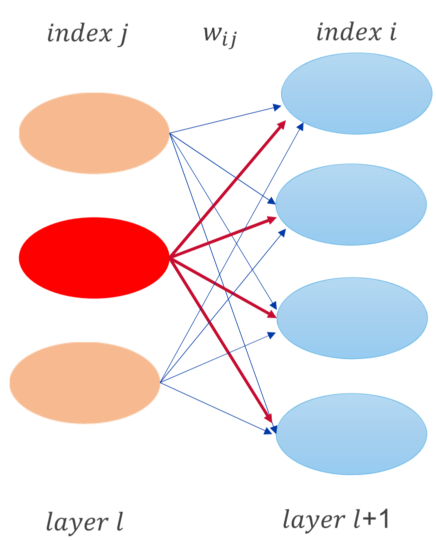
$$\frac{\partial E}{\partial w_{ij}(l)} = \frac{\partial E}{\partial z_i(l)}\frac{\partial z_i(l)}{\partial w_{ij}(l)} = \delta_i(l)a_j(l-1)$$

$$\frac{\partial E}{\partial b_i(l)} = \frac{\partial E}{\partial z_i(l)}\frac{\partial z_i(l)}{\partial b_i(l)} = \delta_i(l)$$



Georgia State University | J. MACK ROBINSON COLLEGE OF BUSINESS

Multilayer Feedforward Neural Net(work)- Backpropagation



$$z_i(l) = \sum_{j=1}^{n_{l-1}} w_{ij}(l)a_j(l-1) + b_i(l) \quad i = 1, 2, \cdots, n_l$$

$$a_i(l) = h\big(z_i(l)\big)$$

$$\delta_j(l) = \frac{\partial E}{\partial z_j(l)} = \sum_i \frac{\partial E}{\partial z_i(l+1)}\frac{\partial z_i(l+1)}{\partial a_i(l)}\frac{\partial a_i(l)}{\partial z_j(l)}$$

$$= \sum_i \frac{\partial E}{\partial z_i(l+1)}\frac{\partial z_i(l+1)}{\partial a_i(l)}\frac{\partial a_i(l)}{\partial z_j(l)}$$

$$= \sum_i \delta_i(l+1)w_{ij}(l+1)h'(z_j(l))$$

Georgia State University | J. MACK ROBINSON COLLEGE OF BUSINESS

34

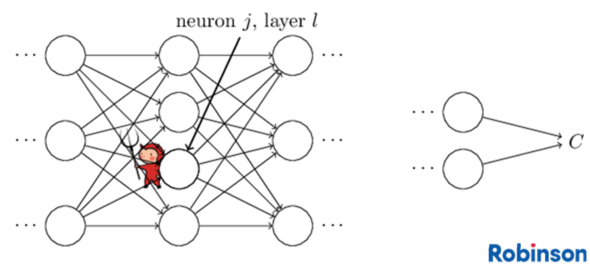
Robinson

Multilayer Feedforward Neural Net(work)- Backpropagation

- Michael Nielsen's book: http://neuralnetworksanddeeplearning.com/chap2.html

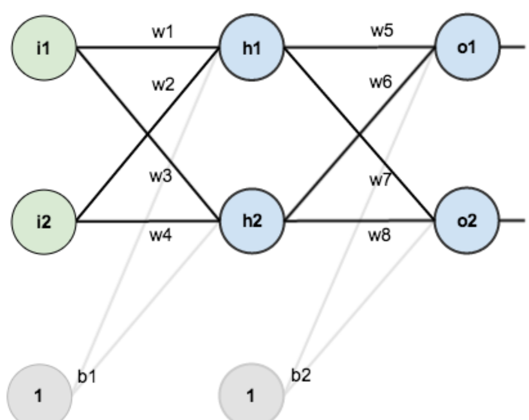
CHAPTER 2

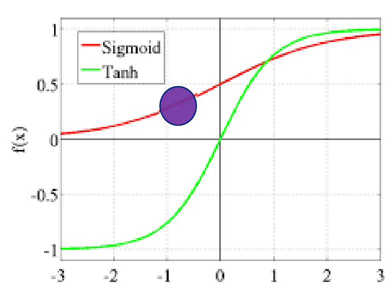
How the backpropagation algorithm works



Backpropagation Example

The neural network has two inputs, two hidden neurons, two output neurons. Additionally, the hidden and output neurons will include a bias.

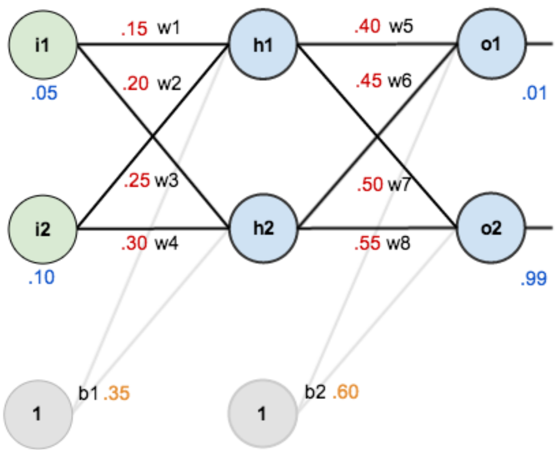




https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Backpropagation Example

In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:



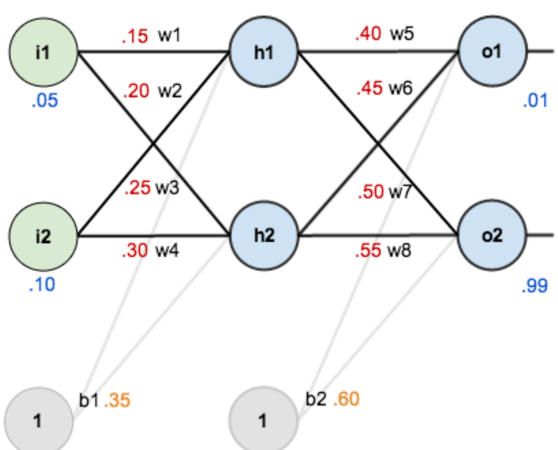
The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.

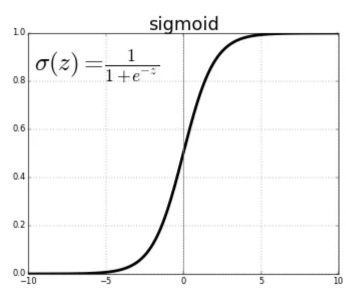
As an example, we work with a single training set: given inputs 0.05 and 0.10 (**X**), we want the neural network to output 0.01 and 0.99 (**Y**).

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

Backpropagation Example

The Forward Pass

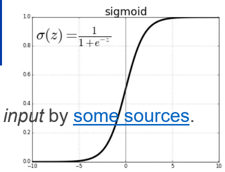




Exercises: find the outputs for hidden neurons and output neurons

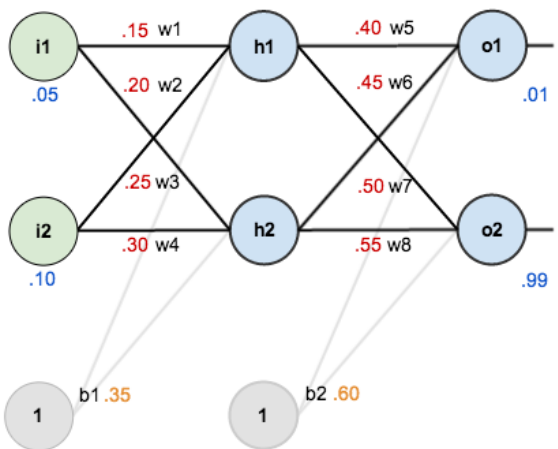
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Backpropagation Example



The Forward Pass

Total net input is also referred to as just *net input* by some sources.



Here's how we calculate the total net input for h_1:

$$net_{h1} = w_1 \times x_1 + w_2 \times x_2 + b_1$$
$$net_{h1} = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775$$

Using the logistic function as the activation function to get the output of h_1:

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593$$

Carrying out the same process for h_2 we get:

$$out_{h2} = 0.597$$

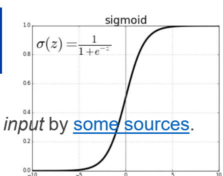
Georgia State University | J. Mack Robinson College of Business

39

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

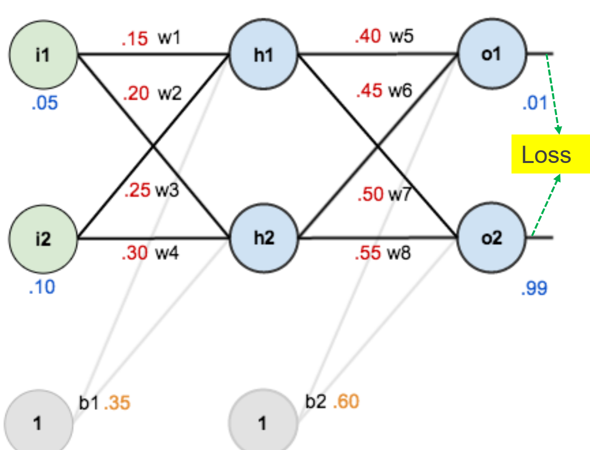
Robinson

Backpropagation Example



The Forward Pass

Total net input is also referred to as just *net input* by some sources.



We repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

Here's the output for o_1:

$$net_{o1} = w_5 \times out_{h1} + w_6 \times out_{h2} + b_2$$
$$net_{o1} = 0.4 \times 0.593 + 0.45 \times 0.597 + 0.6 = 1.106$$

Using the logistic function as the activation function to get the output of o_1:

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.106}} = 0.7513$$

Carrying out the same process for o_2 we get:

$$out_{o2} = 0.773.$$

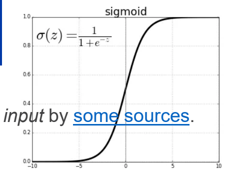
Georgia State University | J. Mack Robinson College of Business

40

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

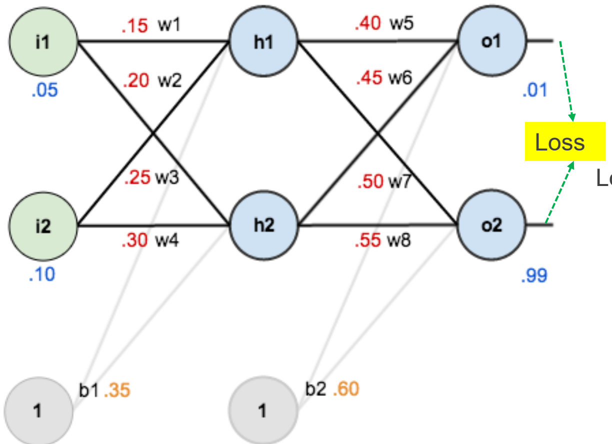
Robinson

Backpropagation Example



The Forward Pass

Total net input is also referred to as just *net input* by some sources.



$$out_{o1} = 0.7513$$

$$out_{o2} = 0.773.$$

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

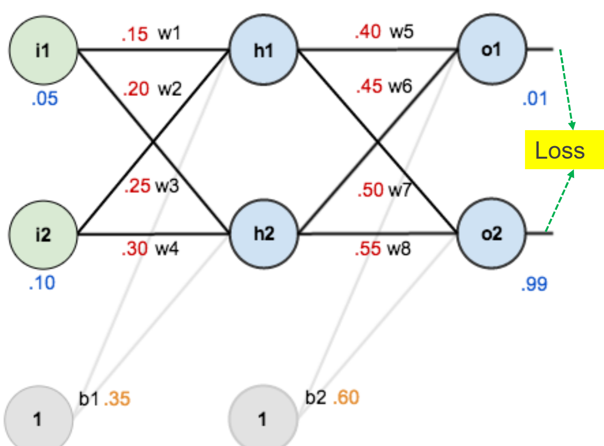
$$= \frac{1}{2}(0.01 - 0.7513)^2 + \frac{1}{2}(0.99 - 0.773)^2 = 0.2984$$

41

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

Backpropagation Example

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.

42

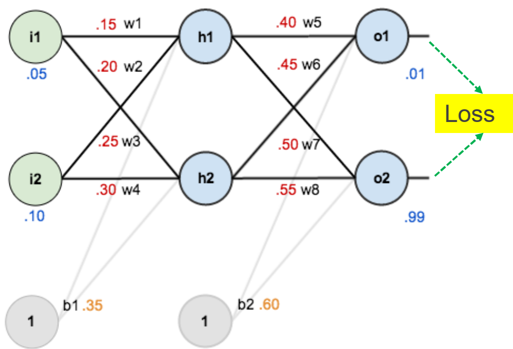
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

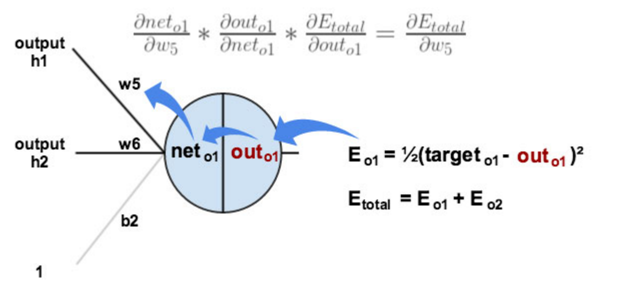
Backpropagation Example

The Backwards Pass

Consider w_5. We want to know how much a change in w_5 affects the total error/loss. That is $\frac{\partial E_{total}}{\partial w_5}$.

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





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

Backpropagation Example

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5} = 0.7413 \times 0.1868 \times 0.593 = 0.082$$

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

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

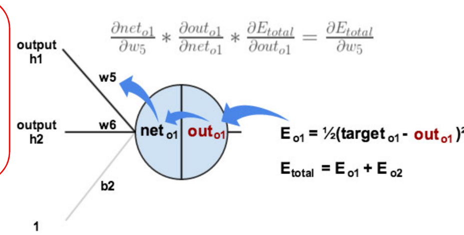
$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

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

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

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

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



http://cs231n.github.io/optimization-2/

Backpropagation Example

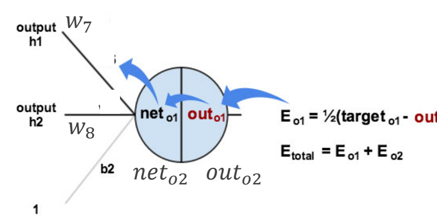
In class exercise: calculate $\frac{\partial E_{total}}{\partial w_7}$=?

$$target_{o_2} = 0.99$$
$$out_{h_1} = 0.593$$
$$out_{o_2} = 0.773$$

$$\frac{\partial E_{total}}{\partial out_{o2}} \times \frac{\partial out_{o2}}{\partial net_{o2}} \times \frac{\partial net_{o2}}{\partial w_7} = -0.218 \times 0.175 \times 0.593 = -0.023$$

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

$$\frac{\partial E_{total}}{\partial out_{o2}} = -(target_{o2} - out_{o2}) = \quad -(0.99 - 0.773) = -0.218$$



$$out_{o2} = \frac{1}{1 + e^{-net_{o2}}}$$

$$\frac{\partial out_{o2}}{\partial net_{o2}} = out_{o2}(1 - out_{o2}) = 0.773 \times (1 - 0.773) = 0.175$$

$$net_{o2} = w_7 \times out_{h1} + w_8 \times out_{h2} + b_2$$

$$\frac{\partial net_{o2}}{\partial w_7} = out_{h1} = 0.593$$

Robinson
http://cs231n.github.io/optimization-2/

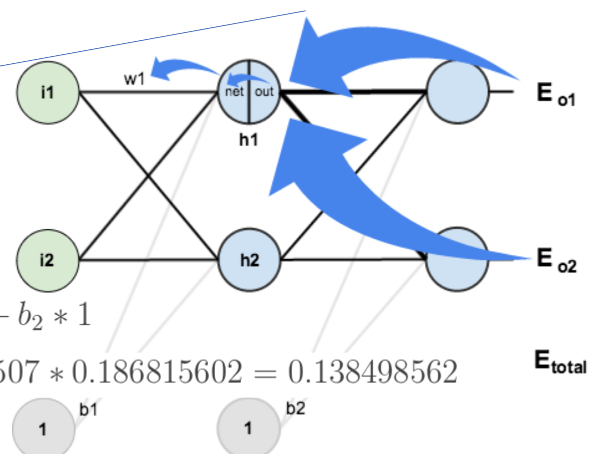
Backpropagation

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

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

Hidden layer

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



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

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

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

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

E total = E o1 + E o2

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

Robinson
http://cs231n.github.io/optimization-2/

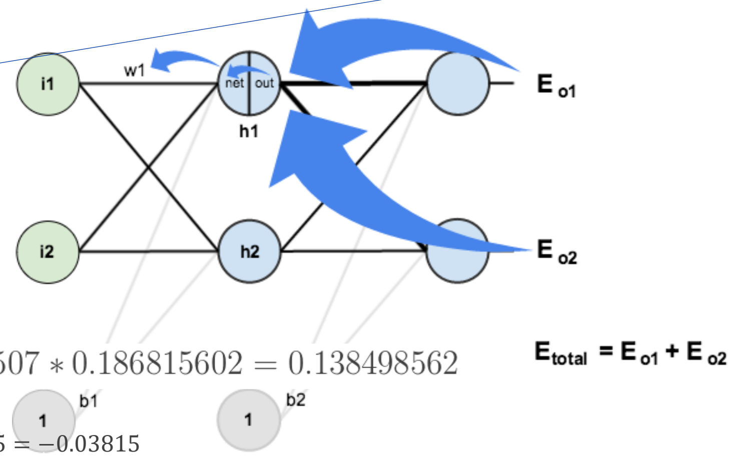
Backpropagation

Hidden layer

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

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

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



$$\frac{\partial E_{o2}}{\partial out_{h1}} = \frac{\partial E_{o2}}{\partial net_{o2}} \times \frac{\partial net_{o2}}{\partial out_{h1}} = -0.0190$$

$$\frac{\partial net_{o2}}{\partial out_{h1}} = w_7 = 0.50$$

$$net_{o2} = w_7 \times out_{h1} + w_8 \times out_{h2} + b_2$$

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

$$\frac{\partial E_{o2}}{\partial net_{o2}} = \frac{\partial E_{o2}}{\partial out_{o2}} \times \frac{\partial out_{o2}}{\partial net_{o2}} = -0.218 \times 0.175 = -0.03815$$

Robinson

http://cs231n.github.io/optimization-2/

Backpropagation

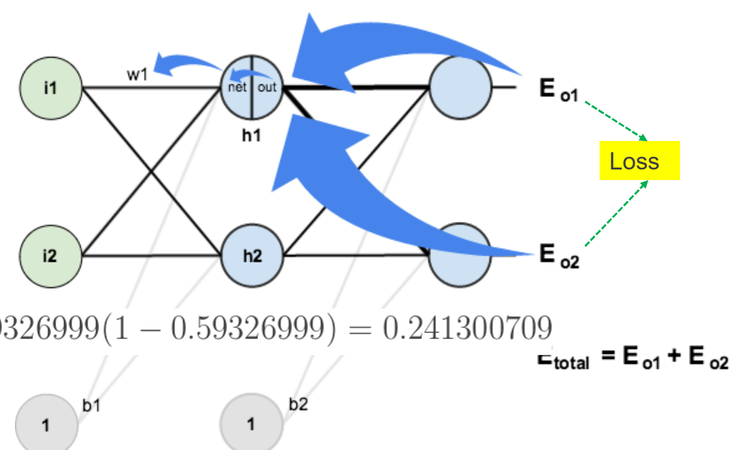
Hidden layer

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

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

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

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



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

Robinson

http://cs231n.github.io/optimization-2/

Backpropagation

Hidden layer

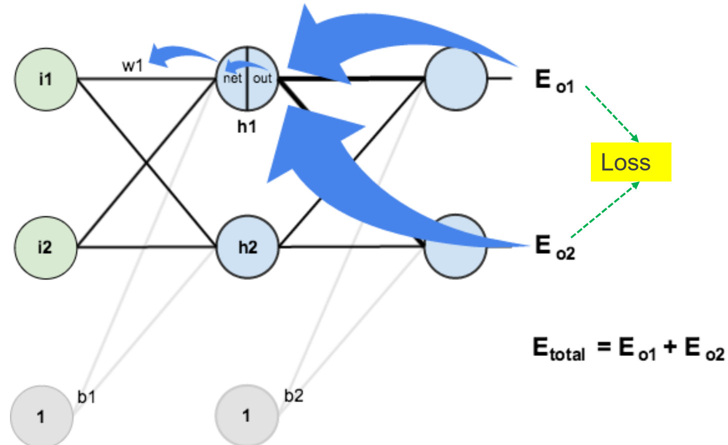
$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

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

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

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

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.0554 + (-0.0190) = 0.0364$$



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

Putting it all together:

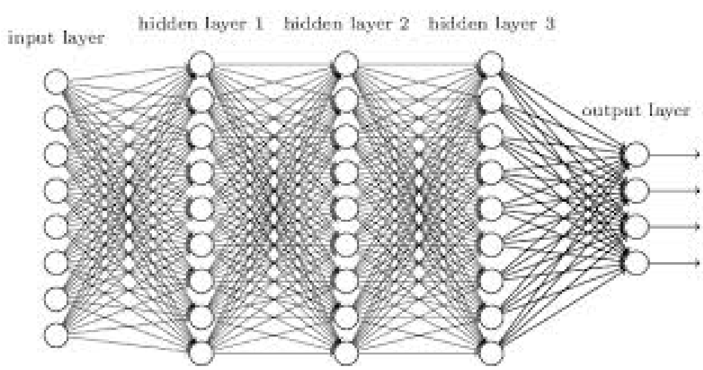
$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

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

http://cs231n.github.io/optimization-2/

Backpropagation

See the gradient change



Georgia State University | J. MACK ROBINSON COLLEGE OF BUSINESS

50

Robinson

Multilayer Feedforward Neural Net(work)- Backpropagation

Once we have the rate of change of E with respect to the network weights and biases in terms of quantities we can compute. The network parameters can be updated using gradient descent:

$$w_{ij}(l) = w_{ij}(l) - \alpha \frac{\partial E(l)}{\partial w_{ij}(l)}$$
$$b_i(l) = b_i(l) - \alpha \frac{\partial E(l)}{\partial b_i(l)}$$

α is the learning rate constant used in gradient descent. There are numerous approaches that attempt to find optimal learning rates, but ultimately this is a problem-dependent parameter that involves experimenting. A reasonable approach is to start with a small value of α (e.g., 0.01), then experiment with vectors from the training set to determine a suitable value in a given application.

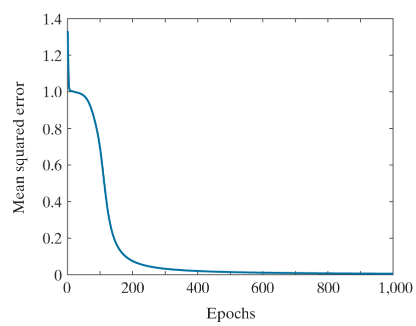
GeorgiaState University | J. MACK ROBINSON COLLEGE OF BUSINESS

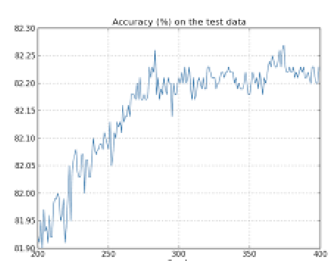
Robinson

Multilayer Feedforward Neural Net(work)- Backpropagation

Summary of one epoch:

Step	Description
Step 1	Input the training data
Step 2	Forward pass to find activations
Step 3	Backpropagaton to find the derivatives
Step 4	Update weight and biases





GeorgiaState University | J. MACK ROBINSON COLLEGE OF BUSINESS

Robinson

Deep Neural Network Implementation Considerations

Robinson

Parameter Initiation

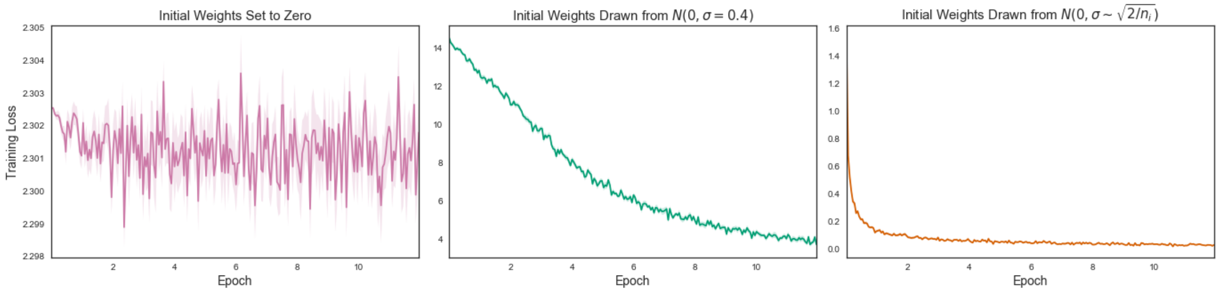
- Weight initialization can actually have a profound impact on both the convergence rate and final quality of a network.
- Still a current active research area

Robinson

Parameter Initiation

Can we set weights to zero?

- If weights are set to 0.0, the equations of the learning algorithm would fail to make any changes to the network weights, and the model will be stuck.
- The bias weight in each neuron is set to zero by default, not a small random value.



Training loss of a simple CNN over 12 epochs.
Figure from https://intoli.com/blog/neural-network-initialization/

Parameter Initiation

Can we set weights to zero?

- Nodes that are side-by-side in a hidden layer connected to the same inputs must have different weights for the learning algorithm to update the weights.
- This is often referred to as the need to break symmetry during training.

"….. If two hidden units with the same activation function are connected to the same inputs, then these units must have different initial parameters. If they have the same initial parameters, then a deterministic learning algorithm applied to a deterministic cost and model will constantly update both of these units in the same way." -- textbook

Parameter Initiation

Initiation Methods

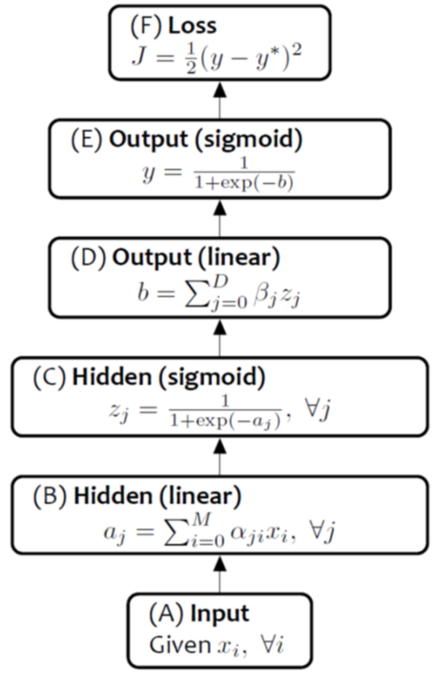
- Weight w_{ij} initialization
  - W = np.random.randn(., .) $\times 0.01$
    - Randomization is used to handle symmetry breaking problem
    - Multiplying a smaller number 0.01 is to let the z value to be small to avoid small gradient in activation functions such as sigmoid, tanh etc.
    - There are many heuristics to initialize the weight (Saxe et al. 2013)
  - $w_{ij} \sim U\left(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}\right)$ for a fully connected layer with m inputs and n outputs.
- Bias initialization
  - B = np.zeros(.,1)

Architecture Design

Neural Network Architectures

Even for a basic Neural Network, there are many design decisions to make:

1. # of hidden layers (depth)
2. # of units per hidden layer (width)
3. Type of activation function (nonlinearity)
4. Form of objective function



Neural Network Architectures

Architecture Design

- Advantage of Depth.

The test set accuracy consistently increases with increasing depth.

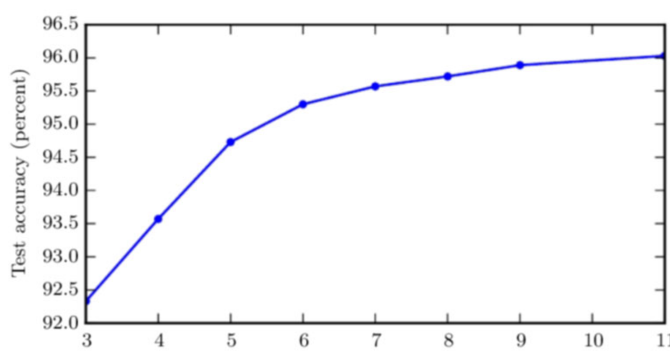


Figure: Goodfellow *et al.*, 2014

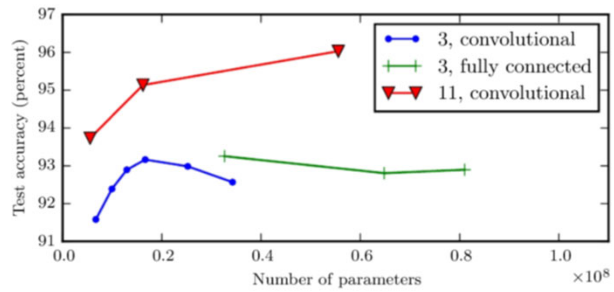
Very simple. Just keep adding layers until the test error does not improve anymore."

Neural Network Architectures

Architecture Design

- Advantage of Depth.

Deeper models tend to perform better. However, width and the number of parameters need to be consistent

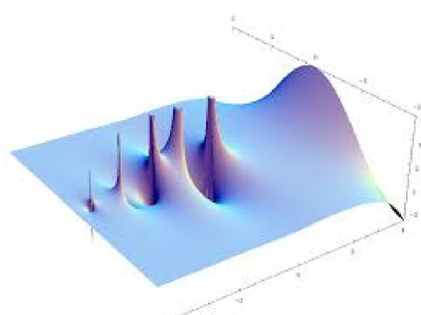


- Control experiments show that other increases to model size don't yield the same effect

Figure: Goodfellow *et al.*, 2014

Why Deep Multilayer Neural Networks?

- **Shallow representations are inefficient at representing highly varying functions**
  - ❖ Representation of complex functions needs many layers in the architecture

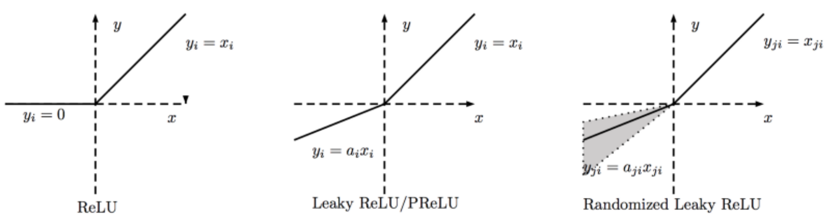


62

Robinson

Neural Network Architectures

Type of activation function (nonlinearity)



Activation functions are used to **introduce nonlinearity** to models, which allows deep learning models to learn nonlinear prediction boundaries.

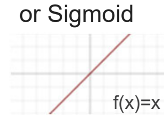
$$\begin{bmatrix}1.2\\0.9\\0.4\end{bmatrix} \rightarrow \text{Softmax} \rightarrow \begin{bmatrix}0.46\\0.34\\0.20\end{bmatrix}$$

- Generally, the **rectifier activation function** is the most popular.
- **Sigmoid** is used in the output layer while making **binary predictions.**
- **Softmax** is used in the output layer while making **multi-class predictions.**

63

Neural Network Architectures

Objective Functions for Neural Networks

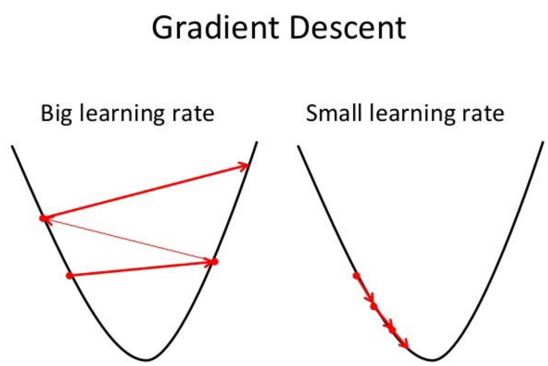
	Classification	Regression
Training examples	R^n x {class_1, ..., class_n} (one-hot encoding)	R^n x R^m
Output Layer	Soft-max [map R^n to a probability distribution] $P(y=j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^T \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^T \mathbf{w}_k}}$	Linear (Identity) or Sigmoid 
Cost (loss) function	Cross-entropy $J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\left[y_k^{(i)}\log\hat{y}_k^{(i)} + \left(1-y_k^{(i)}\right)\log\left(1-\hat{y}_k^{(i)}\right)\right]$	Mean Squared Error $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)}-\hat{y}^{(i)}\right)^2$ Mean Absolute Error $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left\lvert y^{(i)}-\hat{y}^{(i)}\right\rvert$

List of loss functions

Other Hyperparameters

Learning Rate

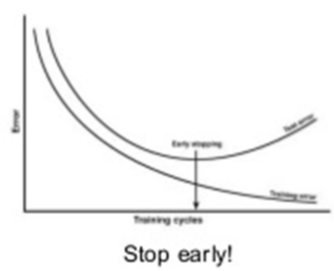
- The learning rate defines how quickly a network updates its parameters.
- Low learning rate slows down the learning process but converges smoothly.
- Larger learning rate speeds up the learning but may not converge.
- Usually a decaying Learning rate is preferred.



Other Hyperparameters

Number of Epochs

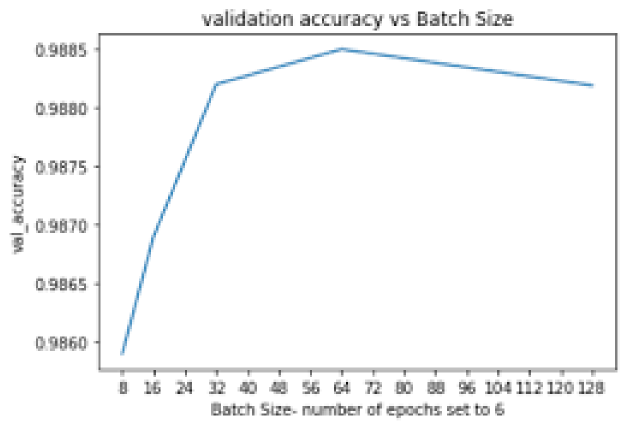
- Number of epochs is the number of times the whole training data is shown to the network while training.
- **Increase the number of epochs until the validation accuracy starts decreasing even when training accuracy is increasing(overfitting).**
- Be aware that when the number of epochs is increased to be too large, overfit can happen



Other Hyperparameters

Batch size

- Mini batch size is the number of sub samples given to the network after which parameter update happens.
- A good default for batch size might be 32. Also try 32, 64, 128, 256, and so on.

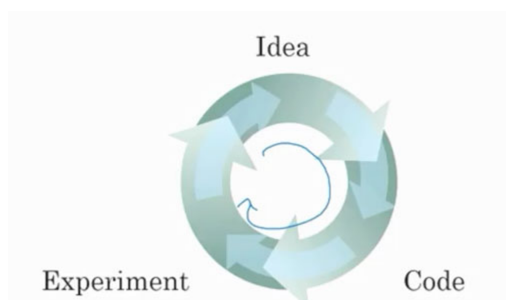


https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a

Other Hyperparameters

Methods used to find out Hyperparameters

- *Manual Search*
- *Grid Search* (http://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/)
- *Random Search*
- *Bayesian Optimization*



Deep Learning Development Cycle by Andrew Ng

https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a