

Imperial College
London

May 1st, 2019

Feed-Forward Neural Networks

Olivier Dubrulle/Lukas Mosser

Imperial College
London

Objectives of the Day

- Focus on Supervised Learning
- Present Logistic Regression as a Single Neuron Operation
- Generalize to Feed-Forward Neural Networks
- Illustrate the Back-Propagation Algorithm
- Introduce Stochastic Gradient Descent
- Show Examples

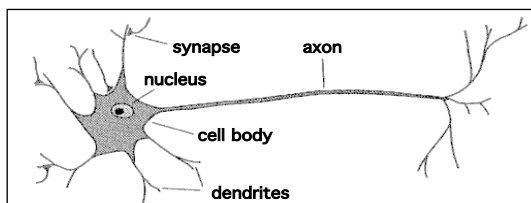
Imperial College
London

Feed-Forward Neural Networks

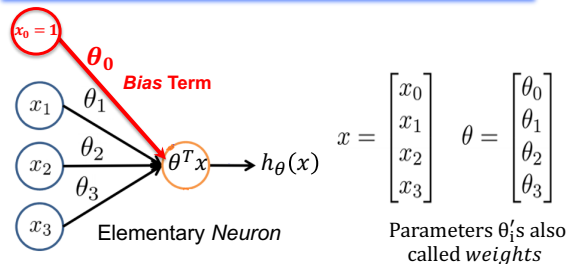
1. From Logistic Regression to Single Neuron Representation
2. Feed-Forward Neural Networks
3. Back-Propagation
4. Batch, Stochastic and Mini-Batch Gradient Descent
5. Work-Flow and Examples

Imperial College
London

Logistic Regression: Analogy with Human Neuron



- A neuron has
 - Branching input (dendrites)
 - Branching output (the axon)

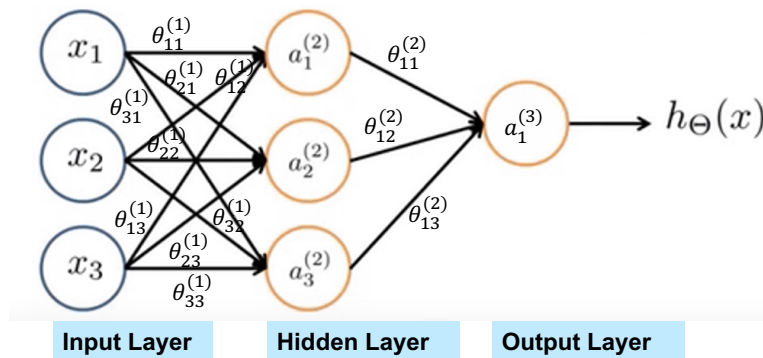


$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

g also called *activation function*

Imperial College
London

From Single Neuron to Feed-forward Neural Network



Historically this is a generalization of the Multi-Layer Perceptron or MLP, which was the very first network capable of learning its weights itself (Rosenblatt, 1961).

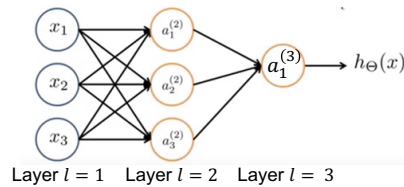
Imperial College
London

Feed-Forward Neural Networks

1. From Logistic Regression to Single Neuron Representation
2. Feed-Forward Neural Networks
3. Back-Propagation
4. Batch, Stochastic and Mini-Batch Gradient Descent
5. Work-Flow and Examples

Imperial College
London

A Simple Neural Network in Matrix Form



$a^{(l)}$ = Activation vector of layer l

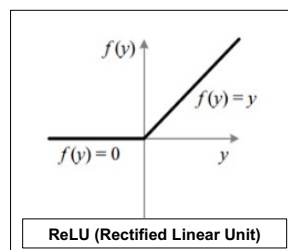
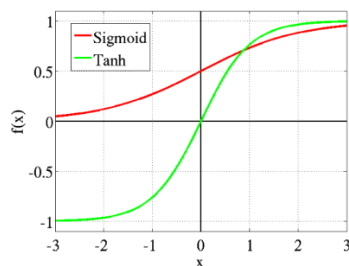
$\theta^{(l)}$ = Matrix of weights controlling mapping from layer l to layer $l+1$
 $\theta_{jk}^{(l)}$ = weight from neuron k in layer (l) to neuron j in layer $(l+1)$

For example:

$$a^{(2)} = \begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{pmatrix} = g(\theta^{(1)} a^{(1)}) = g(\theta^{(1)} x) = g \left(\begin{pmatrix} \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)} \\ \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)} \\ \theta_{30}^{(1)} & \theta_{31}^{(1)} & \theta_{32}^{(1)} & \theta_{33}^{(1)} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \right)$$

Imperial College
London

Possible Choices for the Non-Linear Activation Function g



Tanh has mean of zero (instead of mean of 0.5 for sigmoid function).

ReLU is such that gradient does not vanish for large or small values.








Sigmoid $\sigma(z)$ between 0 and 1, mostly used for output layer of binary classification problems.

Imperial College
London

Why Logistic Regression is not Enough?

- There are some complex mathematical functions (such as the “Exclusive OR”) that Logistic Regression cannot represent

Logic Gates

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR																																																																																																
Alg. Expr.	\overline{A}	AB	\overline{AB}	$A + B$	$\overline{A + B}$	$A \oplus B$	$\overline{A \oplus B}$																																																																																																
Symbol																																																																																																							
Truth Table	<table><tr><th>A</th><th>X</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	X	0	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	0	0	1	0	1	0	0	1	1	1	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	1	0	1	1	1	0	1	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	1	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	1
A	X																																																																																																						
0	1																																																																																																						
1	0																																																																																																						
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	1																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					

<https://medium.com/autonomous-agents/how-to-teach-logic-to-your-neuralnetworks-116215c71a49>

Imperial College
London

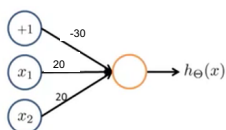
Single Neurons Approximate Some Logical Functions

The activation function used is the sigmoid.



Show that $h_{\theta}(x)$ models the « NOT » Logical Function

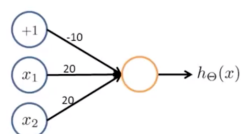
x_1	$h_{\theta}(x)$
0	
1	



Show that $h_{\theta}(x)$ models the « AND » Logical Function

x_2	0	1
x_1	0	0

x_1	x_2	$h_{\theta}(x)$
0	0	
0	1	
1	0	
1	1	



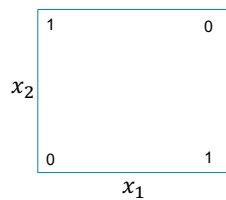
Show that $h_{\theta}(x)$ models the « OR » Logical Function

x_2	1	1
x_1	0	1

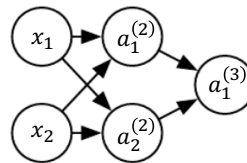
x_1	x_2	$h_{\theta}(x)$
0	0	
0	1	
1	0	
1	1	

Modelling the XOR Function with a Neural Network (1)

The "Exclusive OR" or XOR function: when one of the input x_1, x_2 values is equal to 1 and the other to 0, XOR returns the value 1, otherwise 0.



This cannot be approximated by a single neuron, as we saw that the Logistic Regression Decision Boundary was a line. We need a hidden layer! Let us try the simplest possible neural network:



Inspired from Goodfellow et al, 2016

Modelling the XOR Function with a Neural Network (2)

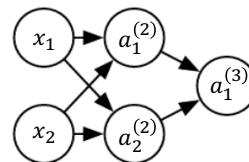
Using previously defined notations, let us assume that we have a Bias term and the parameters of the first layer are:

$$\theta^{(1)} = \begin{pmatrix} 0 & 1 & 1 \\ -1 & 1 & 1 \end{pmatrix}$$

And that the parameters of the second layer are:

$$\theta^{(2)T} = (0 \quad 1 \quad -2)$$

The activation function is ReLU.



Exercise: Calculate the network output for each of the four input points $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$:

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Inspired from Goodfellow et al, 2016

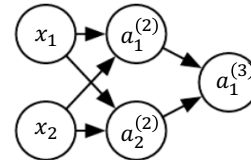
Modelling the XOR Function with a Neural Network (3)

We have $\begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = \text{ReLU} \left(\theta^{(1)} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \right)$

with $\theta^{(1)} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 \\ -1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 + x_1 + x_2 \\ -1 + x_1 + x_2 \end{pmatrix}$

And we have $a_1^{(3)} = \text{ReLU} \left(\theta^{(2)T} \begin{pmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} \right)$

with $\theta^{(2)T} \begin{pmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = \begin{pmatrix} 0 & 1 & -2 \end{pmatrix} \begin{pmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = (a_1^{(2)} - 2a_2^{(2)})$



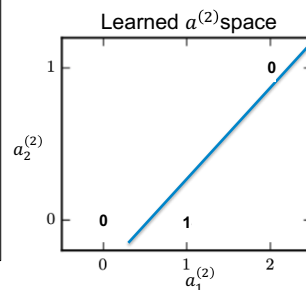
Inspired from Goodfellow et al, 2016

Modelling the XOR Function with a Neural Network (4)

It is easy to see that :

$$\begin{aligned} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} &\Rightarrow \begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = \text{ReLU} \begin{pmatrix} 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} &\Rightarrow \begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = \text{ReLU} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} &\Rightarrow \begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = \text{ReLU} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} &\Rightarrow \begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = \text{ReLU} \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \end{aligned}$$

The first layer has changed the position of the four points such that they can now be linearly separated by the second layer!

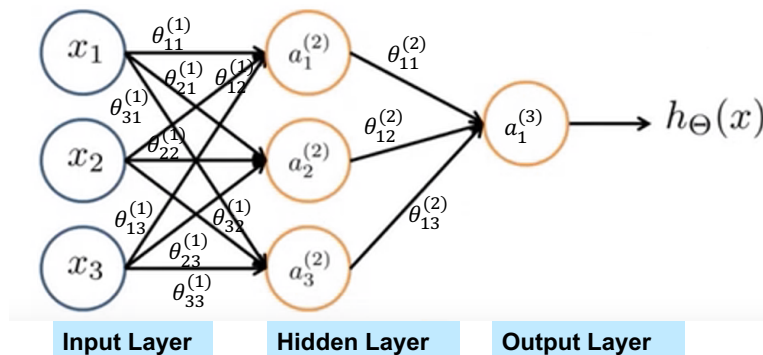


..and the third layer $a_1^{(3)} = \text{ReLU} (a_1^{(2)} - 2a_2^{(2)})$ gives XOR function for the four points!

Inspired from Goodfellow et al, 2016

Imperial College
London

From Single Neuron to Feed-forward Neural Network



Historically this is a generalization of the Multi-Layer Perceptron or MLP, which was the very first network capable of learning its weights itself (Rosenblatt, 1961).

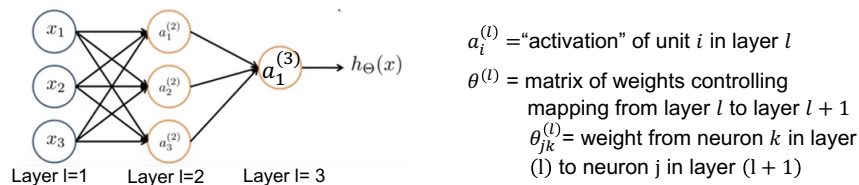
Imperial College
London

Neural Nets or Logistic Regression

- There are some complex mathematical functions (such as the “Exclusive OR”) that Logistic Regression cannot represent.
- Linear Logistic Regression represents a single neuron and is the shallowest form of neural network. The input features are given and the Decision Boundary is linear. But you may need to “massage” the input features to help this linear Decision Boundary separate the input data properly in the output layer. The Neural Network, by transforming the initial variable at each layer, does this “massaging”.
- The “Universal Approximation Theorem” (Hornik et al, 1989) states that a Neural Network with at least one hidden layer can approximate arbitrarily well any function from any finite dimensional space to another, provided that the network is given enough neurons.

Imperial College
London

Neural Network: Notations and Definitions



$$a_1^{(2)} = g \left(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3 \right)$$

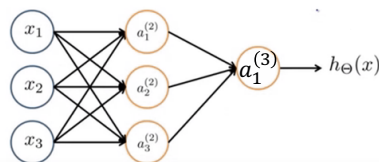
$$a_2^{(2)} = g \left(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3 \right)$$

$$a_3^{(2)} = g \left(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3 \right)$$

$$h_{\theta}(x) = a_1^{(3)} = g \left(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)} \right)$$

Imperial College
London

The Number of Parameters of a Neural Network



Number of parameters (or weights) $\theta_{jk}^{(l)}$, assuming bias term for each layer?

On this example:

(Number of nodes in input layer + 1) \times Number of nodes in hidden layer +
 (Number of nodes in hidden layer + 1) \times Number of nodes in output layer

$$= (3+1) \times 3 + (3+1) \times 1 = \mathbf{16}$$

Imperial College
London

Example of Neural Network With Two Hidden Layers

Forward propagation

$$a^{(1)} = x$$

$$z^{(2)} = \theta^{(1)} a^{(1)}$$

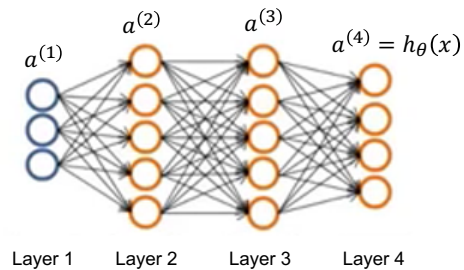
$$a^{(2)} = g(z^{(2)})$$

$$z^{(3)} = \theta^{(2)} a^{(2)}$$

$$a^{(3)} = g(z^{(3)})$$

$$z^{(4)} = \theta^{(3)} a^{(3)}$$

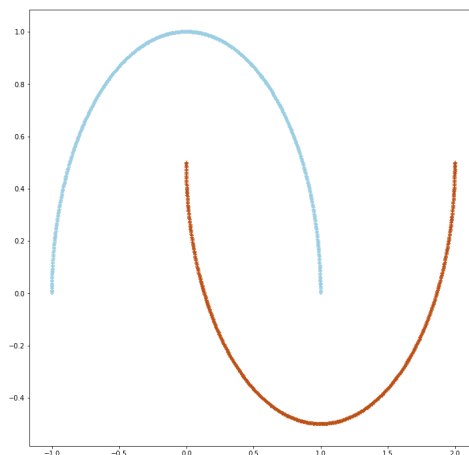
$$a^{(4)} = g(z^{(4)}) = h_{\theta}(x)$$



Number of parameters (assuming bias terms)?
 $(3+1) \times 5 + (5+1) \times 5 + (5+1) \times 4 = 74$

Imperial College
London

Neural Network on Logistic Regression Example

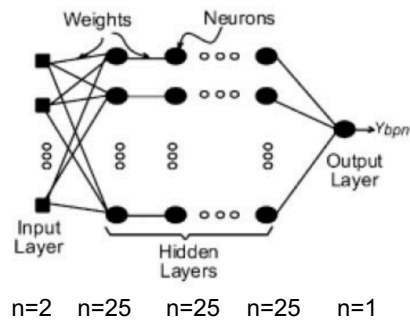


The $m = 1000$ data points $(x_1^i, x_2^i)_{i=1,1000}$ are in the plane.
 A group of data are one color,
 the other group another color.

Question: predict the color at each location of the plane.

Imperial College
London

A Simple Neural Network Example (2)



Neural Network Architecture

Input Layer: Two Neurons (x_1 and x_2)

Three Hidden Layers Each 25 Neurons

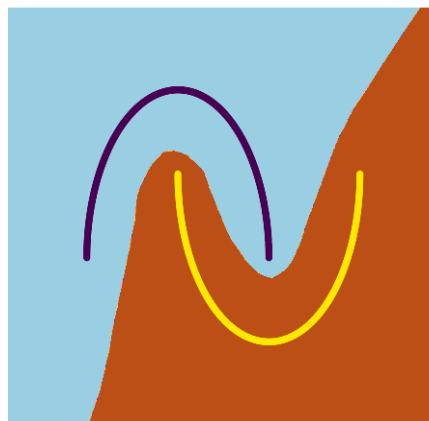
Output Layer: One neuron: Probability of Being Blue

Total Number of Weights (if bias terms):
 $(2+1) \times 25 + (25+1) \times 25 + (25+1) \times 25 + (25+1) \times 1 = 1401$

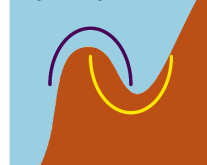
Imperial College
London

A Simple Neural Network Example (3)

Result of
Neural
Network



Logistic Regression



Imperial College
London

More Simple Neural Network Examples

<https://playground.tensorflow.org>

Imperial College
London

More Simple Neural Network Classification Examples

<https://playground.tensorflow.org>

Four classification examples are discussed.

The third one can be addressed by logistic regression, that is with no hidden layer at all and just one output neuron.

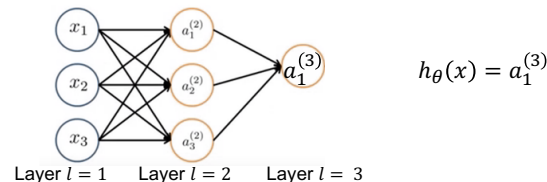
The second one cannot be addressed by logistic regression, because it is clearly non linear and it is similar to the XOR function. We need at least one hidden layer.

The first one is the circle. With just one hidden layer and 4 neurons in it we reproduce the circle (<https://www.youtube.com/watch?v=ru9dXF04iSE>). But of course if we also use x_1^2 and x_2^2 as input we do not even need a hidden layer.

Imperial College
London

Cost Function in the Binary Case

Binary Classification problem: $y = 0$ or 1



Since we have m training examples in the training set, the cost function is the cross-entropy already used in the Logistic Regression case:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[y_i \log a_{1i}^{(3)} + (1 - y_i) \log (1 - a_{1i}^{(3)}) \right]$$

We will see in the third week that cross-entropy is equal to (minus) the likelihood of a Bernoulli distribution.

Imperial College
London

Multi-Class Classification

The “One vs Rest” Approach. Example of 3 Classes.

- Calculate the Neural Network for three problems:
 - Merge classes 2 and 3 as class “zero” and calculate parameters θ_1 of function:

$$h_{\theta_1}^1(x) = P(y = 1|x, \theta_1)$$
 - Merge classes 1 and 3 as class “zero” and calculate parameters θ_2 of function:

$$h_{\theta_2}^2(x) = P(y = 2|x, \theta_2)$$
 - Merge classes 1 and 2 as class “zero” and calculate parameters θ_3 of function:

$$h_{\theta_3}^3(x) = P(y = 3|x, \theta_3)$$
- For each new unlabeled point x , calculate the three probabilities above and pick as the predicted class the one corresponding to the highest of the three probabilities.

Imperial College
London

Multi-Class Classification with Softmax

Suppose the output of the last layer of the neural network is $\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_i \\ \vdots \\ z_{n-1} \\ z_n \end{pmatrix}$ of size n

The **Softmax** function transforms it into an output probability vector:

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \\ p_i \\ \vdots \\ p_{n-1} \\ p_n \end{pmatrix} = \begin{pmatrix} \frac{e^{z_1}}{\sum_{j=1}^n e^{z_j}} \\ \frac{e^{z_2}}{\sum_{j=1}^n e^{z_j}} \\ \vdots \\ \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}} \\ \vdots \\ \frac{e^{z_{n-1}}}{\sum_{j=1}^n e^{z_j}} \\ \frac{e^{z_n}}{\sum_{j=1}^n e^{z_j}} \end{pmatrix} \text{ then the class with the highest probability is chosen}$$

Imperial College
London

A SoftMax Function Calculation Example

$$\begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix} = \begin{pmatrix} 5 \\ -1 \\ 2 \\ -4 \end{pmatrix} \qquad \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} = \begin{pmatrix} \frac{148.41}{156.19} \\ \frac{0.37}{156.19} \\ \frac{7.39}{156.19} \\ \frac{0.02}{156.19} \end{pmatrix} = \begin{pmatrix} 0.950 \\ 0.003 \\ 0.047 \\ 0.000 \end{pmatrix}$$

SoftMax Function does two things:

- Transform the vector into probabilities that are between 0 and 1 and add up to 1
- Transform the largest value into a value close to 1 and the lowest into one close to 0

Imperial College
London

Example of Neural Network With Two Hidden Layers

Forward propagation

$$a^{(1)} = x$$

$$z^{(2)} = \theta^{(1)} a^{(1)}$$

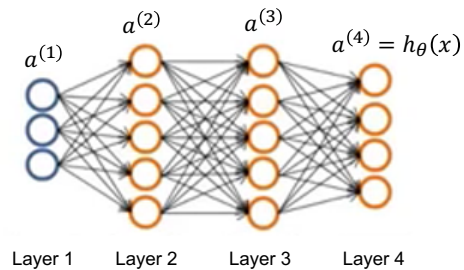
$$a^{(2)} = g(z^{(2)})$$

$$z^{(3)} = \theta^{(2)} a^{(2)}$$

$$a^{(3)} = g(z^{(3)})$$

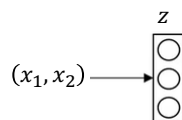
$$z^{(4)} = \theta^{(3)} a^{(3)}$$

$$a^{(4)} = \text{Softmax}(z^{(4)}) = h_{\theta}(x)$$



Imperial College
London

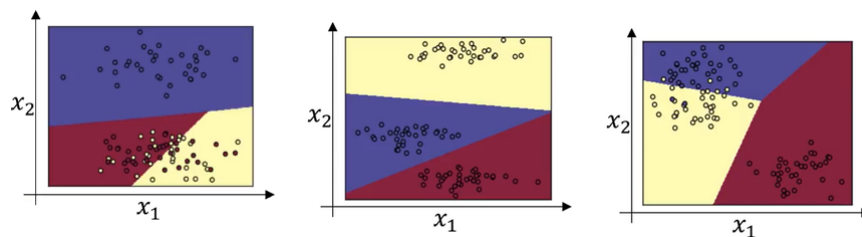
Example of Classifying 2-D Data into 3 Classes



$$a_1^{(1)} = \text{Softmax}(z_1) = \text{Softmax}(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2)$$

$$a_2^{(1)} = \text{Softmax}(z_2) = \text{Softmax}(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2)$$

$$a_3^{(1)} = \text{Softmax}(z_3) = \text{Softmax}(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2)$$



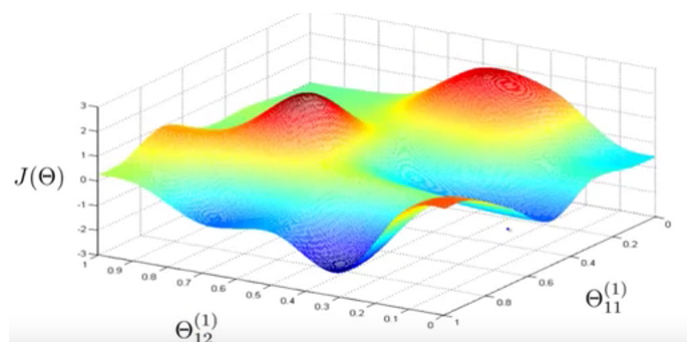
Imperial College
London

Feed-Forward Neural Networks

1. From Logistic Regression to Single Neuron Representation
2. Feed-Forward Neural Networks
3. Back-Propagation
4. Batch, Stochastic and Mini-Batch Gradient Descent
5. Work-Flow and Examples

Imperial College
London

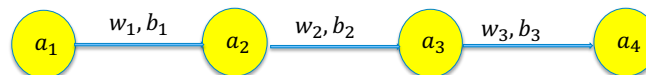
How to Minimize the Cost Function with Neural Nets?



Imperial College
London

A Look at Back-Propagation Using a Simple Example (1)

Let us take a very simple network of $L=4$ layers with one neuron in each.
We have six parameters, three weights w_i and three biases b_i .



Suppose we have just one sample (x, y) (hence $a_1 = x$) and a regression neural network.

The L_2 cost function is $C = \frac{1}{2}(a_4 - y)^2$.

We have: $a_2 = g(z_2)$ with $z_2 = w_1 a_1 + b_1$
 $a_3 = g(z_3)$ with $z_3 = w_2 a_2 + b_2$
 $a_4 = g(z_4)$ with $z_4 = w_3 a_3 + b_3$

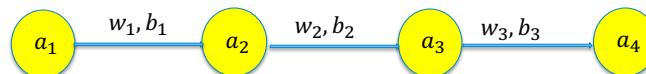
*How to calculate the
derivative of the Cost
Function according to
each of the six
coefficients?*

How many coefficients do we have to calculate?

<https://www.youtube.com/watch?v=tleHLnjs5U8>

Imperial College
London

A Look at Back-Propagation Using a Simple Example (2)



The L_2 cost function is $C = \frac{1}{2}(a_4 - y)^2$.

We have: $a_2 = g(z_2)$ with $z_2 = w_1 a_1 + b_1$
 $a_3 = g(z_3)$ with $z_3 = w_2 a_2 + b_2$
 $a_4 = g(z_4)$ with $z_4 = w_3 a_3 + b_3$

Using the chain rule, the derivatives according to w_3 and b_3 are easy;

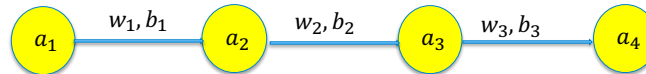
$$\frac{\partial C}{\partial w_3} = \frac{\partial C}{\partial a_4} \frac{\partial a_4}{\partial z_4} \frac{\partial z_4}{\partial w_3} = (a_4 - y)g'(z_4)a_3 \quad \frac{\partial C}{\partial b_3} = \frac{\partial C}{\partial a_4} \frac{\partial a_4}{\partial z_4} \frac{\partial z_4}{\partial b_3} = (a_4 - y)g'(z_4)$$

and we also have:
$$\frac{\partial C}{\partial a_3} = \frac{\partial C}{\partial a_4} \frac{\partial a_4}{\partial z_4} \frac{\partial z_4}{\partial a_3} = (a_4 - y)g'(z_4)w_3$$

<https://www.youtube.com/watch?v=tleHLnjs5U8>

Imperial College
London

A Look at Back-Propagation Using a Simple Example (3)



Thanks to the formula : $\frac{\partial C}{\partial a_3} = \frac{\partial C}{\partial a_4} \frac{\partial a_4}{\partial z_4} \frac{\partial z_4}{\partial a_3} = (a_4 - y)g'(z_4)w_3$

We can keep moving backwards and now obtain the derivatives of C in w_2 and b_2

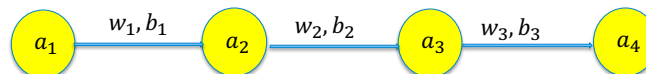
$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial a_3} \frac{\partial a_3}{\partial z_3} \frac{\partial z_3}{\partial w_2} = (a_4 - y)g'(z_4)w_3g'(z_3)a_2$$

$$\frac{\partial C}{\partial b_2} = \frac{\partial C}{\partial a_3} \frac{\partial a_3}{\partial z_3} \frac{\partial z_3}{\partial b_2} = (a_4 - y)g'(z_4)w_3g'(z_3)$$

<https://www.youtube.com/watch?v=tleHLnjs5U8>

Imperial College
London

A Look at Back-Propagation Using a Simple Example (4)



We can show that : $\frac{\partial C}{\partial a_2} = \frac{\partial C}{\partial a_3} \frac{\partial a_3}{\partial z_3} \frac{\partial z_3}{\partial a_2} = (a_4 - y)g'(z_4)w_3g'(z_3)w_1$

And again we can keep moving backwards and obtain the derivatives of C in w_1 and b_1

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial w_1} = (a_4 - y)g'(z_4)w_3g'(z_3)w_1g'(z_2)x$$

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial b_1} = (a_4 - y)g'(z_4)w_3g'(z_3)w_1g'(z_2)$$

So we have obtained the six partial derivatives by back-propagation!

<https://www.youtube.com/watch?v=tleHLnjs5U8>

Imperial College
London

Feed-Forward Neural Networks

1. From Logistic Regression to Single Neuron Representation
2. Feed-Forward Neural Networks
3. Back-Propagation
4. Batch, Stochastic and Mini-Batch Gradient Descent
5. Work-Flow and Examples

Imperial College
London

Batch vs Stochastic Gradient Descent

"Batch" Gradient Descent uses all m training set data $(x_i, y_i)_{i=1, \dots, m}$ at each gradient descent iteration. When m is very large (say m is in the 100,000's), the number of calculations is thus very large, and this is just to calculate one gradient on all the parameters $\theta_{ij}^{(l)}$.

Instead of summing over all the training set, then iterating, we can iterate only on the basis of gradients at individual data points. With *Stochastic Gradient Descent*: every iteration works on one data point at a time.



Stochastic gradient descent

1. Randomly shuffle (reorder) training examples
2. Repeat {
 - for $i := 1, \dots, m$ {
 - $\theta_j := \theta_j - \alpha(h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}$
(for every $j = 0, \dots, n$)

Imperial College
London

Mini Batch Gradient Descent

Rather than the two extremes of *Batch* and *Stochastic Gradient Descent*, one often chooses the intermediate *Mini-Batch Gradient Descent*, where the gradient is calculated on a small subset of data.

Mini-batch gradient descent

Say $b = 10, m = 1000$.

Repeat {

for $i = 1, 11, 21, 31, \dots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_{\theta}(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every $j = 0, \dots, n$)

}

}

Andrew NG, 17.3

Imperial College
London

Gradient Descent: Different Ways to use the Data

Batch (also called Full-Batch) Gradient Descent:

Using all m training set data $(x_i, y_i)_{i=1, \dots, m}$ at each gradient descent iteration

Stochastic Gradient Descent:

Use one single data (x_i, y_i) at each gradient descent iteration

Mini-Batch Gradient Descent:

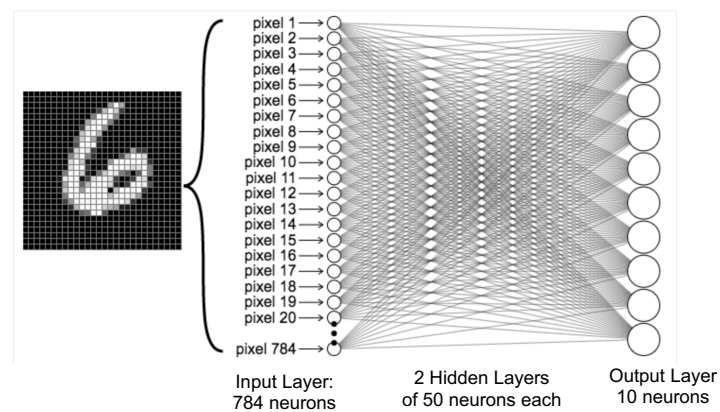
Use small number (say a few tens or hundreds) of data (x_i, y_i) at each gradient descent iteration

An **epoch** is a training iteration over the whole training set. It is thus composed of one single gradient descent iteration in the Batch case, and as many gradient descent iterations as there are training data in the Stochastic Gradient Descent case.

Basic Neural Network Training Algorithm (one Epoch)

1. Initialize the training with random parameters $\theta_{jk}^{(l)}$
2. Calculate $h_{\theta}(x^{(i)})$ for new sub-group of training set data $x^{(i)}$
3. Calculate cost/loss function $J(\theta)$
4. Calculate gradients by back-propagation
5. Modify parameters $\theta_{jk}^{(l)}$ by gradient descent

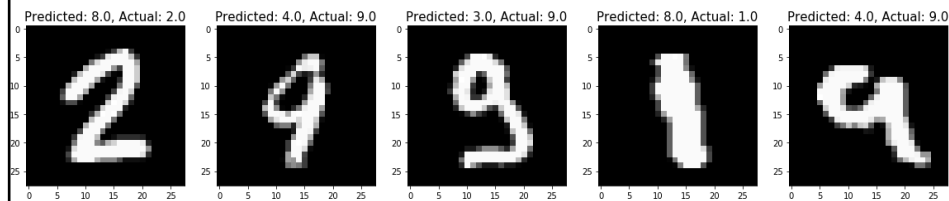
MNIST Example of Neural Network Architecture



Imperial College
London

MNIST Example of Neural Network Architecture

Examples of Misclassified Images



Imperial College
London

Feed-Forward Neural Networks on MNIST

The Results:

On the 60000 Training Images

Mean Accuracy: 0.95

Misclassified Images: 2859 (4.8%)

On the 10000 Test Images

Mean Accuracy: 0.94

Misclassified Images: 618 (6.2%)

Imperial College
London

Feed-Forward Neural Networks

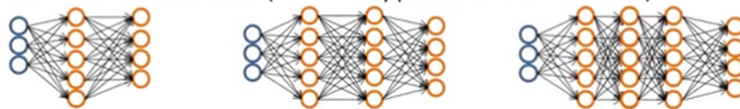
1. From Logistic Regression to Single Neuron Representation
2. Feed-Forward Neural Networks
3. Back-Propagation
4. Batch, Stochastic and Mini-Batch Gradient Descent
5. Work-Flow and Examples

Imperial College
London

Architecture of a Feed-Forward Neural Network

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



1. Number of input and output units determined by number of features and number of outputs.
2. One hidden layer is a good default
3. If several hidden layers, good to start with the same number of units
4. Take number of hidden units about 1x, 2x or 3x number of input units

How can the above choices be made more objective?

Imperial College
London

What have we Learnt?

- Logistic Regression as a Single Neuron
- Feed-Forward Neural Networks allow a « messaging » of the features at each layer
- Back-Propagation Algorithm allows calculation of all gradients
- Batch, Stochastic and MiniBatch Gradient Descent for speeding-up convergence
- Neural Network Workflow