Ollscoil na Gaillimhe
University of Galway

Deep Learning

Topic 02: Fundamentals of Neural Networks, Part 1

Prof. Michael Madden

Chair of Computer Science
Head of Machine Learning Group
University of Galway

Learning Objectives

After successfully completing Topics 2 and 3,
you will be able to…

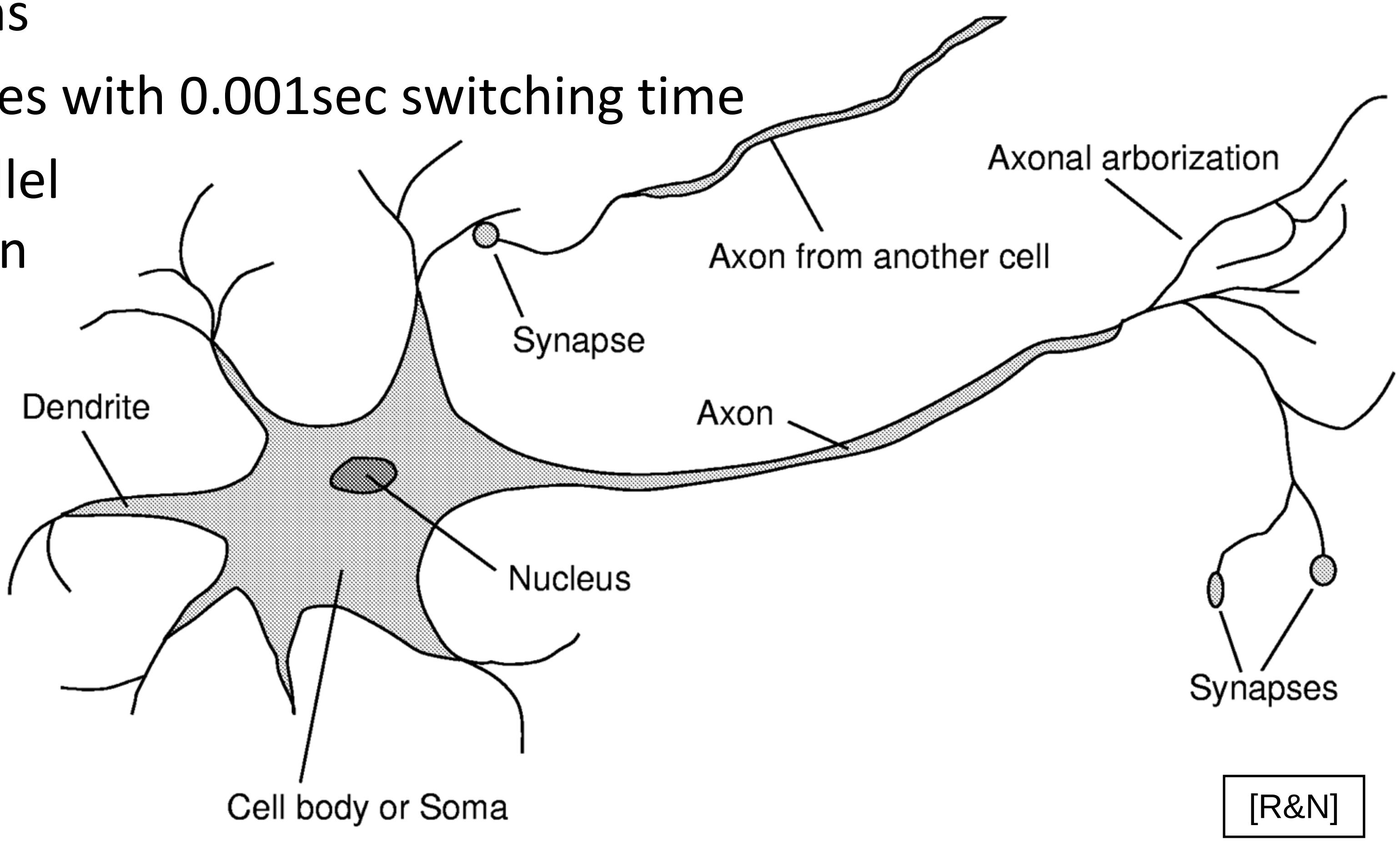
- Explain how logistic regression and stacked classifiers can be implemented as feed-forward neural networks
- Explain neural network notation and perform calculations for forward-propagation and backward-propagation
- Explain and implement the gradient descent algorithm
- Implement neural networks for supervised machine learning tasks, from first principles

Recommended Reading

- There are many tutorials on the web
  - Challenging because they use different notations and often even different assumptions
  - Many that are not peer-reviewed contain errors!
  - Below are ones I found useful
- Russell & Norvig's chapter on Neural Networks
- Goodfellow Deep Learning Book, Chapter 6: https://www.deeplearningbook.org/contents/mlp.html
- Andrew Ng and colleagues in Stanford:
  http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/
- Quoc Le of Google:
  http://www.trivedigaurav.com/blog/quoc-les-lectures-on-deep-learning/

Neural Nets: Biological Inspiration

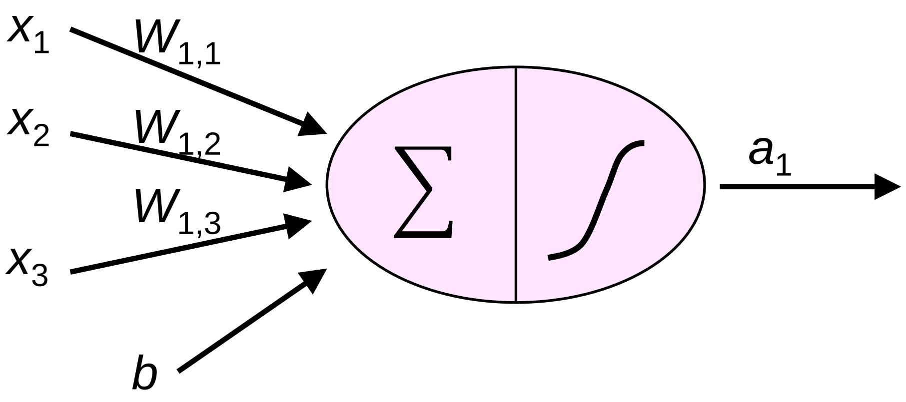
- Human Brain:
  - 10^{11} neurons
  - 10^{14} synapses with 0.001sec switching time
  - Highly parallel computation



[R&N]

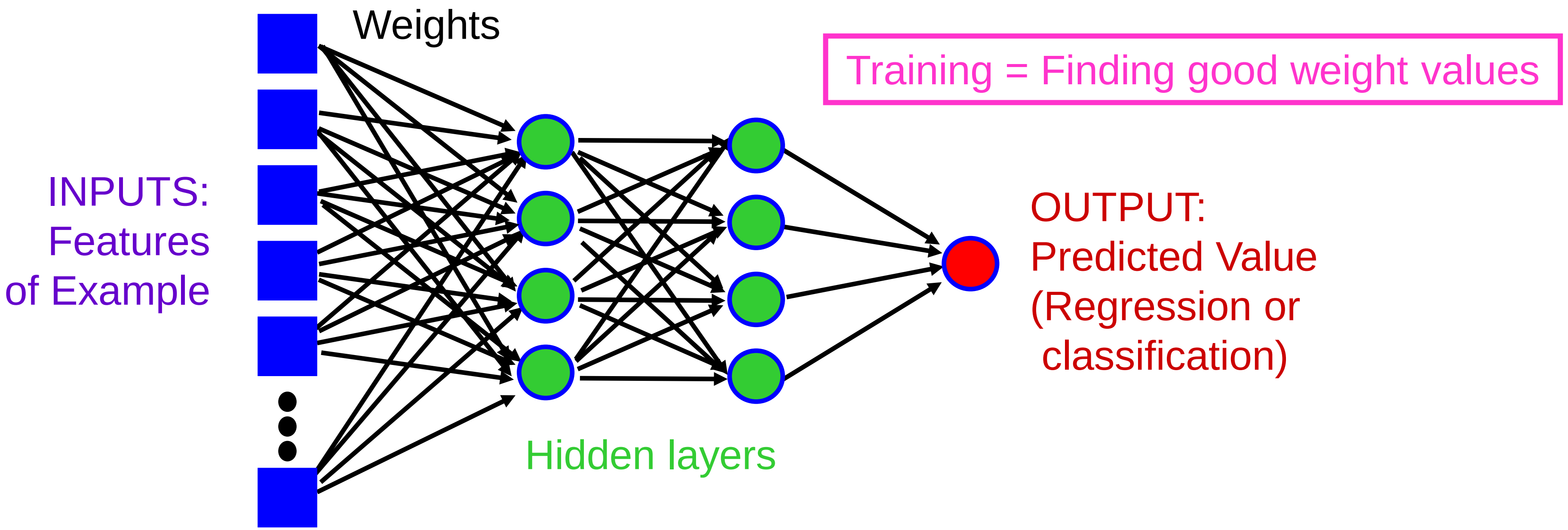
McCulloch-Pitts Neuron (1943)

- Extremely unrealistic model of brain neuron
  - Weights: positive or negative
  - Activation: 0/1, soft threshold; nonlinear; differentiable
    Commonly: sigmoid function
  - Output: "Squashed" linear function of inputs



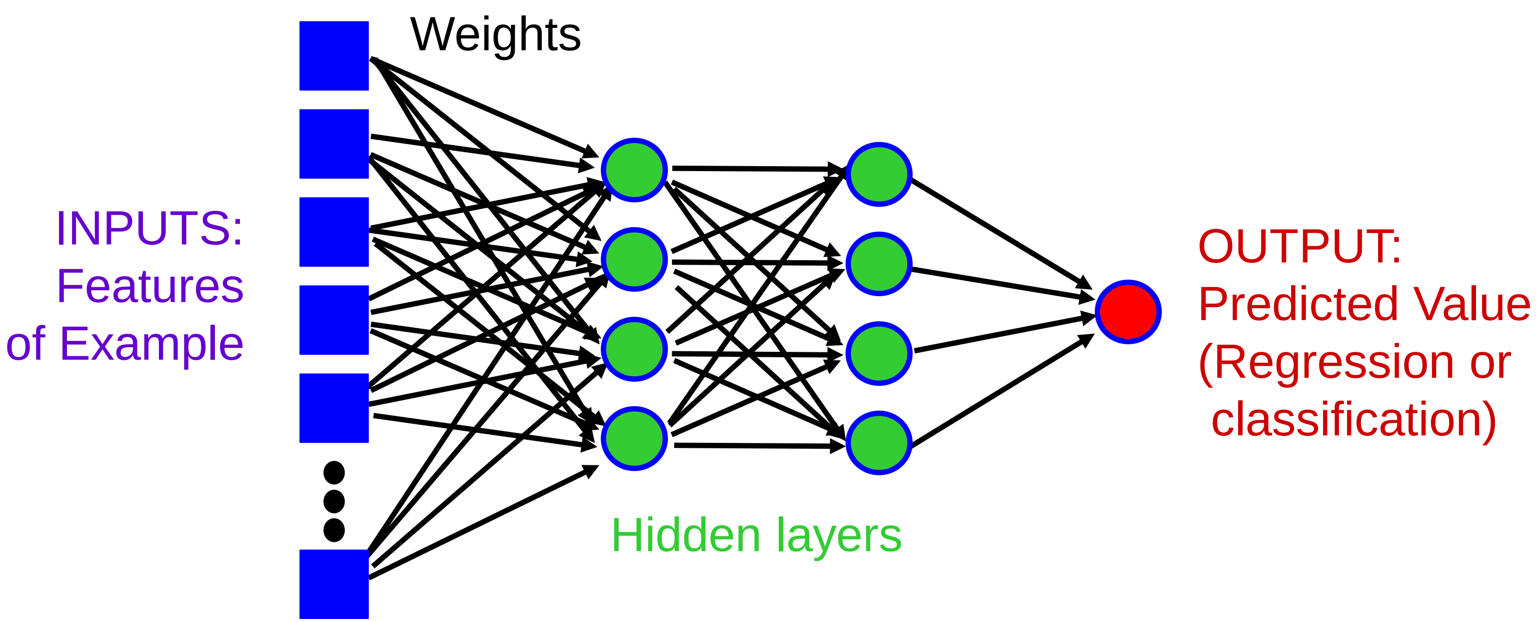
- **A single neuron gives us a simple linear classifier …**

Fully Connected Feed-Forward Neural Network (1)



- Also known as a *Multi-Layer Perceptron*
- Simplest architecture, widely used

Fully Connected Feed-Forward Neural Network (2)



- Neurons connected in layers
  - Family of functions parameterised by weights
  - No internal state
- High-dimensional non-linear interpolation
- Network is a distributed model of the data

Review: Binary Classification

- Is unknown sample X or O ?
  - Goal: Find model that correctly classifies a new sample, $\mathbf{x}^{(i)}$

- X's are assigned : +1
- O's are assigned : 0 (or -1)
- These will be our values for $y^{(i)}$

- Training Data – ($\mathbf{x}^{(i)}$: $y^{(i)}$)

  $\mathbf{s_1}$: (11, 3: 0) **O**

  $\mathbf{s_2}$: (3, 10: 0) **O**

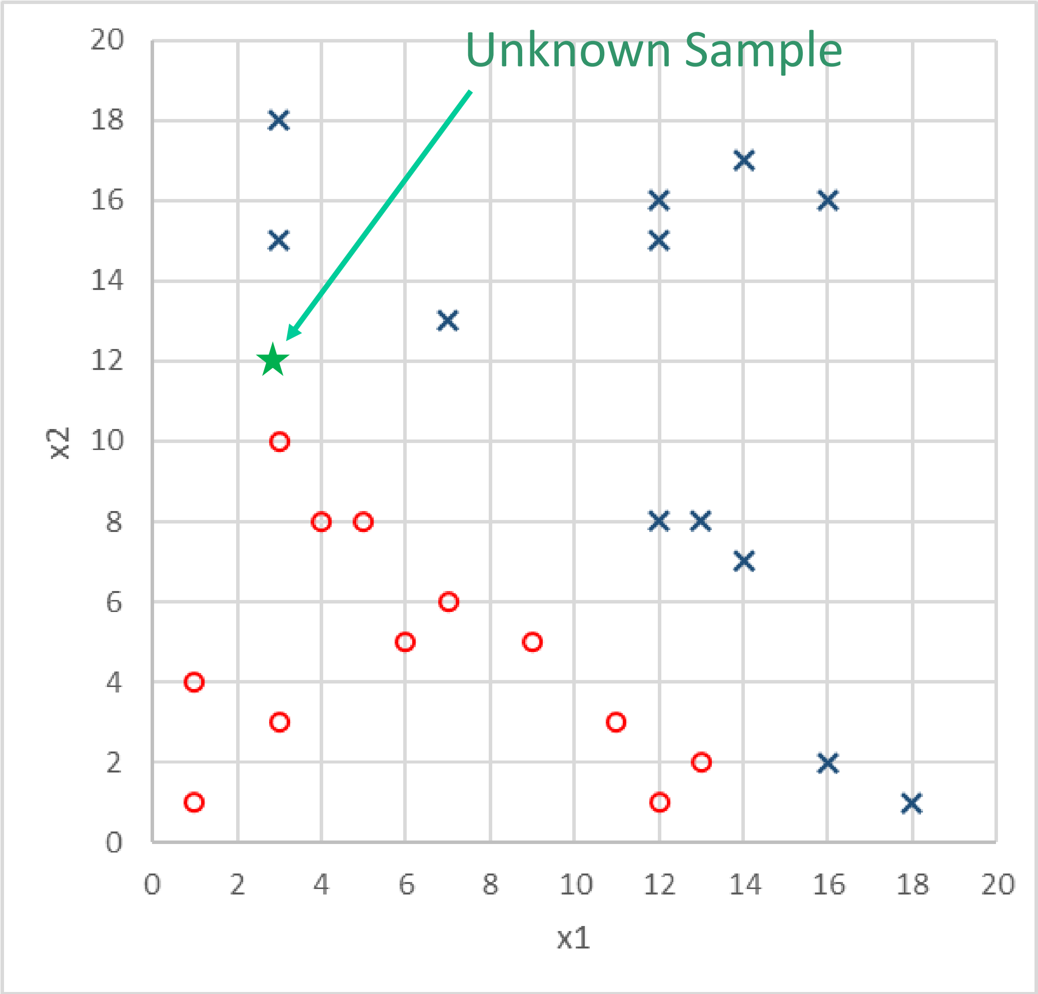
  $\mathbf{s_3}$: (14, 17: +1) **X**

  $\mathbf{s_4}$: (16, 16: +1) **X**

  …

Given an unknown sample s = (3, 12: ?)

What class does this belong to?



Motivating Example: Movie Reviews

- Example from tutorial by Quoc Le
- Want to predict whether I will like a movie
- Two critics, Mary and John, have rated it
- They have also rated (1-5) other movies that I saw, and I know whether or not I liked those movies, so I can try to generalise from those
  - Each instance: a movie with ratings
  - Each attribute: its ratings by Mary and John
  - Dependent variable: I like it Yes/No

Simple Classifier: Logistic Regression (1)

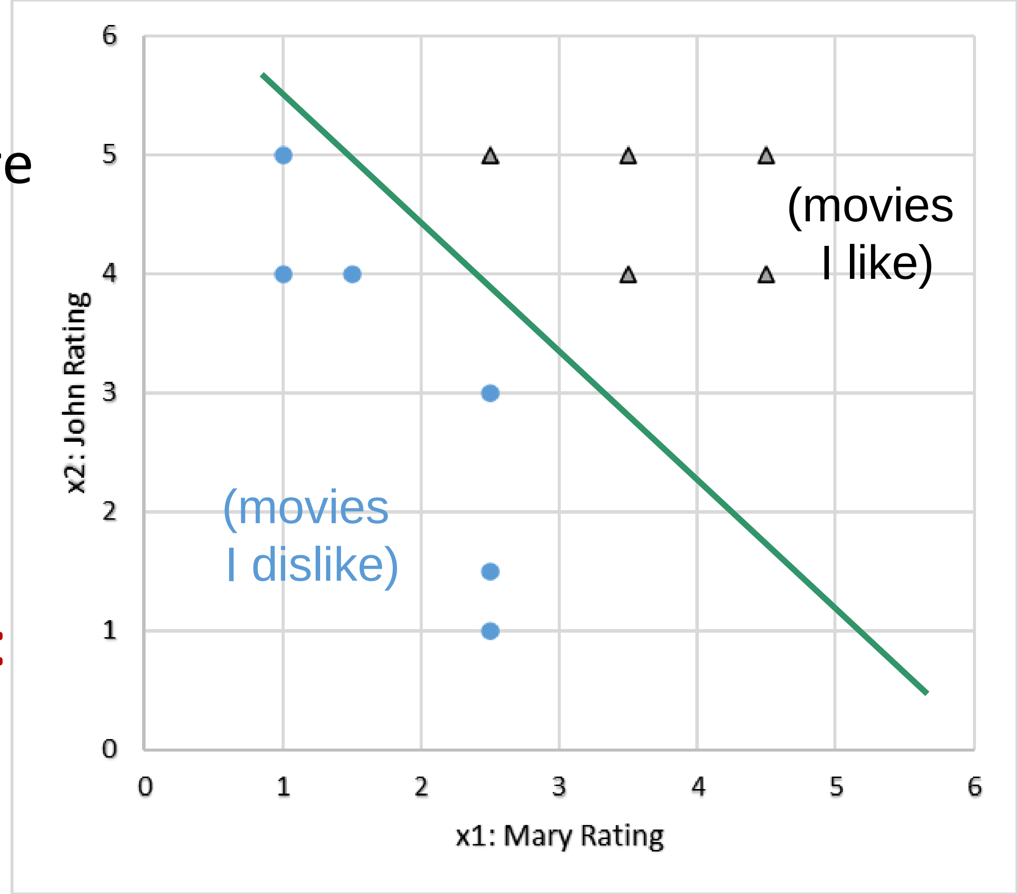
- Goal is to find a **dividing line** between classes
  - Contrast with Linear Regression, where goal is to find best line **through** points
- We can define a straight line:

$$z(\boldsymbol{x}) = b + w_1x_1 + w_2x_2$$
$$= b + \sum w_i x_i$$
$$= \boldsymbol{w} \cdot \boldsymbol{x} + b$$

- Notation changes from ML module:

  W rather than θ for weights

  Represent intercept with b, rather than fixed x_0=1 and weight θ_0



Simple Classifier: Logistic Regression (2)

- Need to apply an activation function (aka threshold function) to convert straight line equation into 0/1 values for classifier
  - For gradient descent, it needs to be differentiable

- Require Activation Function $a(x)$ to take on values in range [0,1]
  - Have it switch rapidly from 0 to 1 (almost a step function)
  - Can also write $a(x)$ as $\hat{y}$ since its goal is to approximate y

- Common NN activation functions are sigmoid, tanh, ReLU, Leaky ReLU: for logistic regression, mainly use sigmoid

Sigmoid Function and Output of Logistic Regression

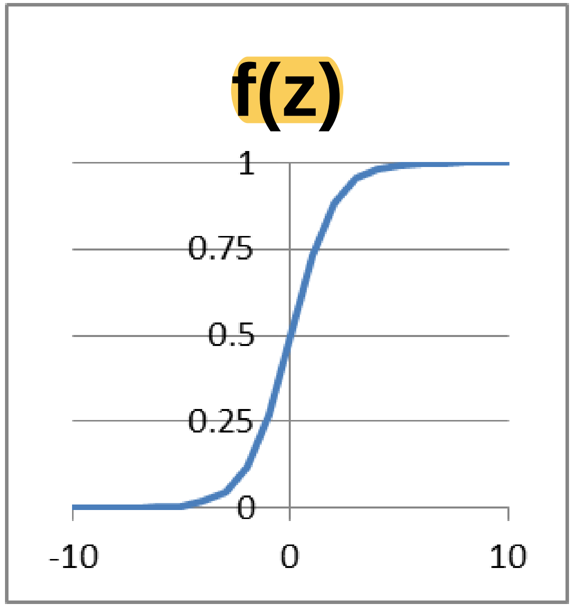
- Go from the linear regression formula:

  $z_{\boldsymbol{w},b}(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b$

- To this:

  $\hat{y} = a_{\boldsymbol{w},b}(\boldsymbol{x}) = f(\boldsymbol{w} \cdot \boldsymbol{x} + b)$ where

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

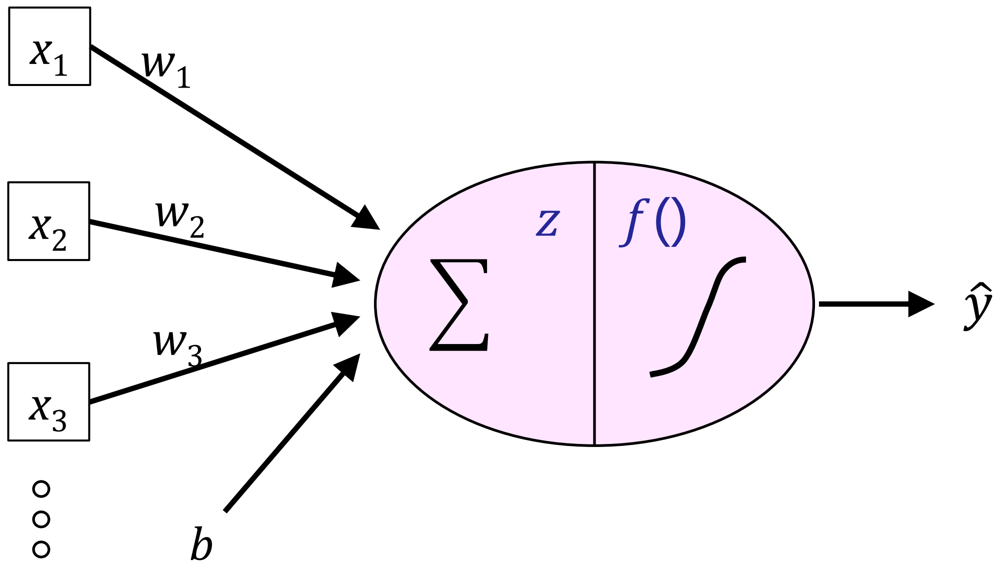


- $f(z)$ is called the sigmoidal or logistic function

- We interpret $\hat{y}$ as the probability that the actual output y is 1, given a set of inputs x: $\hat{y} = \mathrm{P}(y{=}1 \mid \boldsymbol{x};\, \boldsymbol{w}, b)$

Binary Logistic Regression Corresponds to a Single-Node Neural Network

$$\hat{y} = a_{\mathbf{w},b}(\mathbf{x}) = f(\mathbf{w} \cdot \mathbf{x} + b)$$



Logistic Regression Decision Boundary

- To find decision boundary, we need to find values for ***w*** and *b* such that:

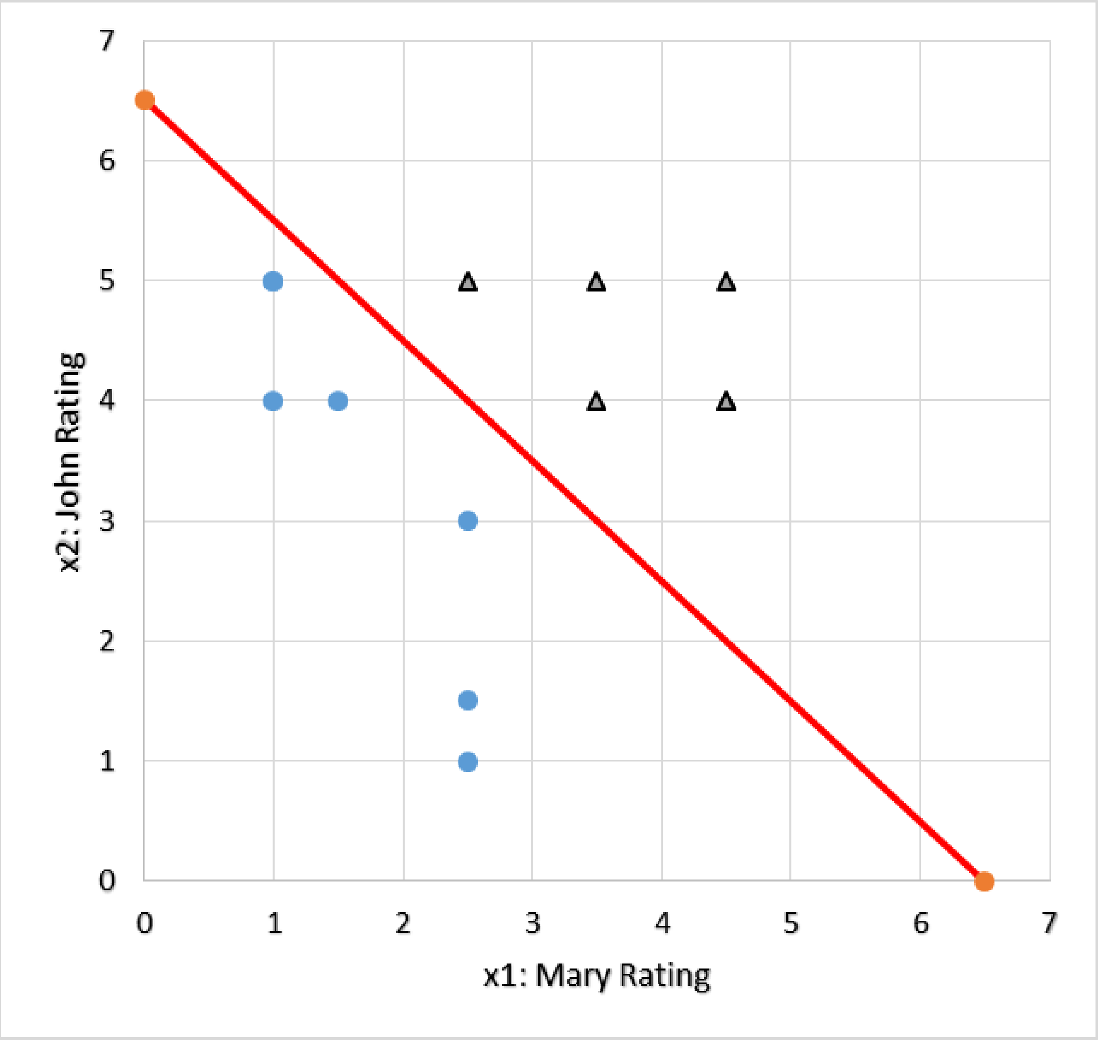
  Class is **Yes** (I like = △)

  $y = 1,\ \hat{y} = a_{w,b}(\boldsymbol{x}) \cong 1$

  Class is **No** (I dislike = ●) :

  $y = 0,\ \hat{y} = a_{w,b}(\boldsymbol{x}) \cong 0$

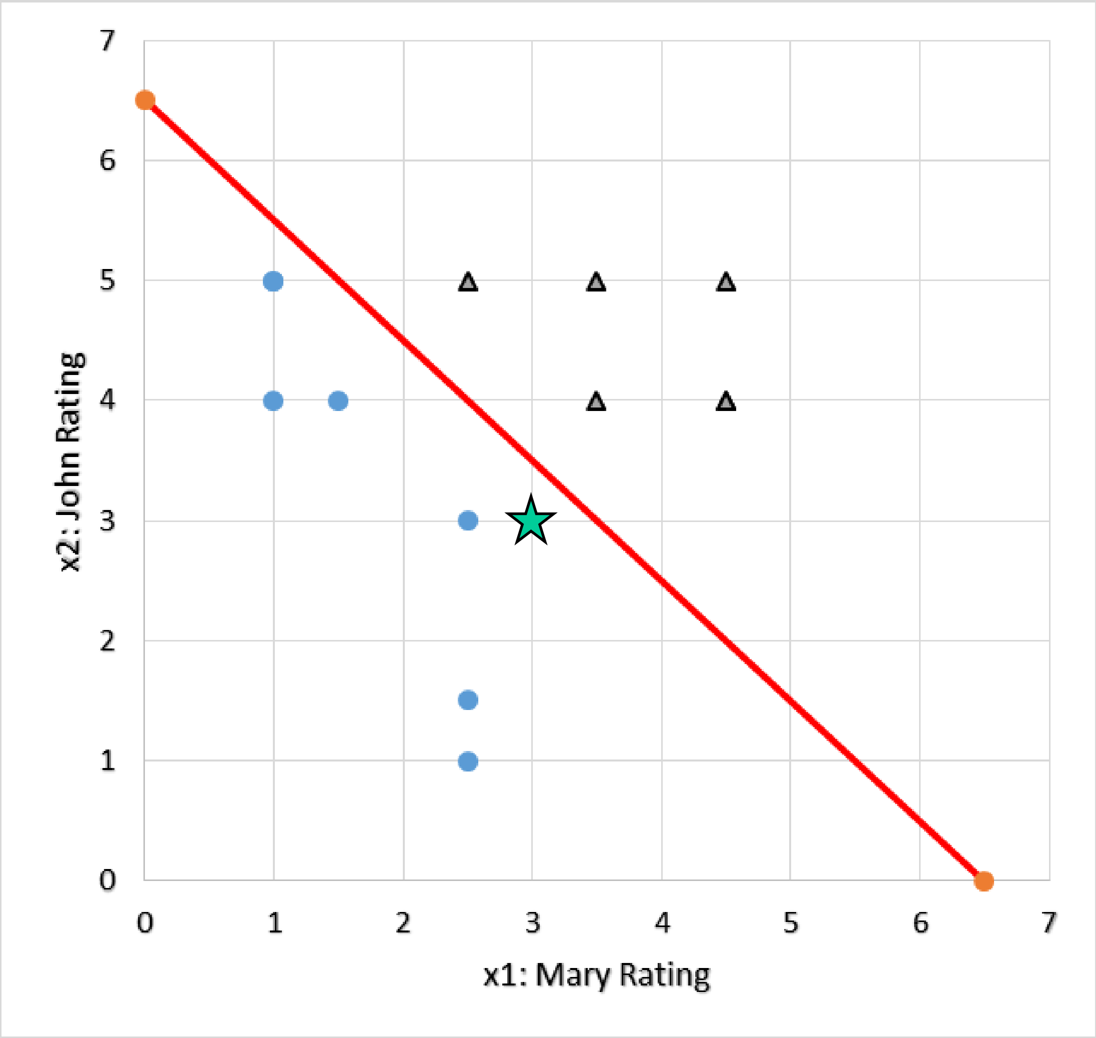
- See MovieRating.xlsx for illustration



Logistic Regression Decision Boundary

- There is another movie, Gravity, that I'm not sure whether or not I will like

- John and Mary gave it a (3,3), which means that I probably won't like it!



Learning the Parameters w and b

- Training a neural network entails finding values for $\boldsymbol{w}$ and b such that, for all training cases $\{\boldsymbol{x}^{(i)}, y^{(i)}\}$, the network outputs values $\hat{y}^{(i)}$ that are as close as possible to $y^{(i)}$
- To implement this (details on following slides):
  1. Define a loss function $L()$ for a single case
  2. From that, define a cost function $J()$ over all training cases
- Then use Gradient Descent (optimisation algorithm) to search for weights $\boldsymbol{w}$ and b that **minimise** the cost for a given training set:

$$\min_{\boldsymbol{w},b} J(\boldsymbol{w}, b)$$

Logistic Regression Loss Function

- **For a single case** $\{\boldsymbol{x}^{(i)}, y^{(i)}\}$, probability that label $y^{(i)}$=1 (Positive Class) for inputs $\boldsymbol{x}^{(i)}$ is given by:

  $$P(y^{(i)} = 1 \mid \boldsymbol{x}^{(i)}; \boldsymbol{w}, b) = a_{\boldsymbol{w},\boldsymbol{b}}(\boldsymbol{x}^{(i)}) = \hat{y}^{(i)}$$

- Therefore, probability that label $y^{(i)}$=0 (Negative Class) is $1 - \hat{y}^{(i)}$

  $$P(y^{(i)} = 0 \mid \boldsymbol{x}^{(i)}; \boldsymbol{w}, b) = 1 - \hat{y}^{(i)}$$

- We can combine these equations to cover both y=1 and y=0:

  $$P(y \mid \boldsymbol{x}; \boldsymbol{w}, b) = (\hat{y})^{y} (1 - \hat{y})^{1-y}$$

- Finally, we can get log of this without affecting optimisation, as log function is strictly monotonically increasing: then referred to as **Log Loss**

Logistic Regression Cost Function

- The cost function is the **log loss averaged over all cases**:

$$J(\boldsymbol{w}, b) = -\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}\log\left(\hat{y}^{(i)}\right) + (1 - y^{(i)})\log\left(1 - \hat{y}^{(i)}\right)\right)$$

- Behaviour:
  - As $\hat{y}$ tends to the correct value (either y=1 or y=0), $J()$ tends to 0
  - As $\hat{y}$ tends to the wrong value, $J()$ tends towards infinity

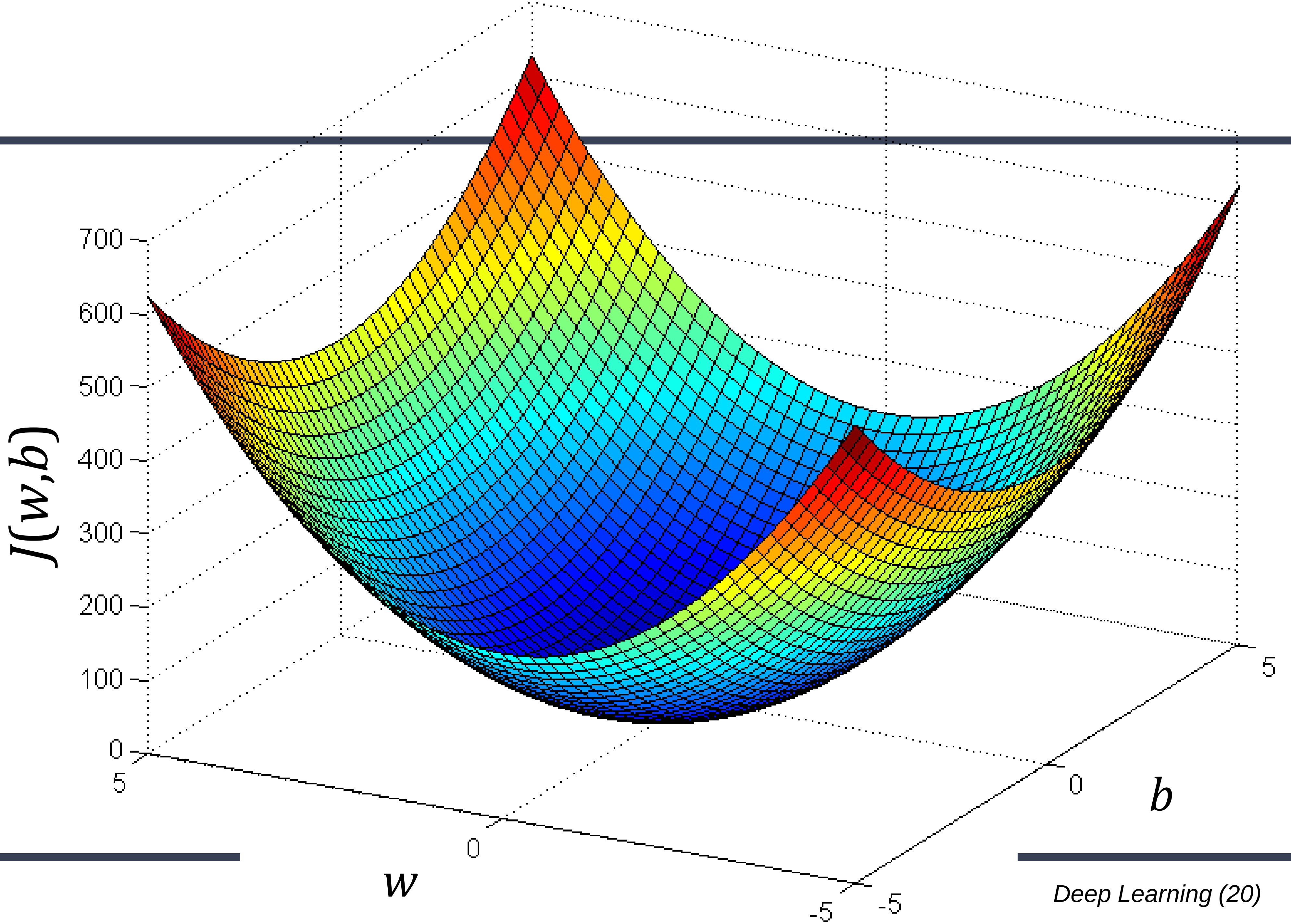
- Will use Gradient Descent to search for weights ***w*** and ***b*** that **minimise** the cost for all cases in a given training set:

$$\min_{\boldsymbol{w},b} \; J(\boldsymbol{w}, b)$$

Overview of Gradient Descent

- General-purpose optimization algorithm to minimize an objective function [in this case, J] by varying some values that can be changed [in this case, $\mathbf{w}$ and b]
- In this case:
  - objective function is our cost function J
  - Values that can be changed are $\mathbf{w}$ and b
- Basic idea:
  - Make initial guesses for $\mathbf{w}$ and b;
  - take incremental steps 'downhill' with step size controlled by learning rate α
  - Keep going until little/no change

Gradient Descent Illustration



Gradient Descent to Learn LR Parameters

We will use the **Stochastic Gradient Decent** algorithm:

initialise $\boldsymbol{w}$, b to a set of valid initial values

repeat until convergence (or until max number of iterations):

  select a single example $\{\boldsymbol{x}^{(i)}, y^{(i)}\}$ at random

  simultaneously for each w_j in $\boldsymbol{w}$ and for b do:

$$w_j := w_j - \alpha \boxed{\frac{\partial}{\partial w_j} J(\mathbf{w}, b)} \qquad \Delta w_j$$

$$b := b - \alpha \boxed{\frac{\partial}{\partial b} J(\mathbf{w}, b)} \qquad \Delta b$$

LR Stochastic Gradient Descent: Partial Derivatives

$$J(\mathbf{w}, b) = -\cancel{\frac{1}{N}}\cancel{\sum_{i=1}^{N}}\left(y^{(i)} \log\left(\hat{y}^{(i)}\right) + (1 - y^{(i)}) \log\left(1 - \hat{y}^{(i)}\right)\right)$$

The partial derivatives are:

$$\Delta w_j = \frac{\partial J}{\partial w_j} = \left(\hat{y}^{(i)} - y^{(i)}\right) x_j^{(i)}$$

$$\Delta b = \frac{\partial J}{\partial b} = \hat{y}^{(i)} - y^{(i)}$$

Where $\hat{y}^{(i)} = f\left(\mathbf{w} \cdot \mathbf{x}^{(i)} + b\right)$ and $f(z) = \dfrac{1}{1+e^{-z}}$

Stopping Criteria: Convergence or Max Number of Iterations

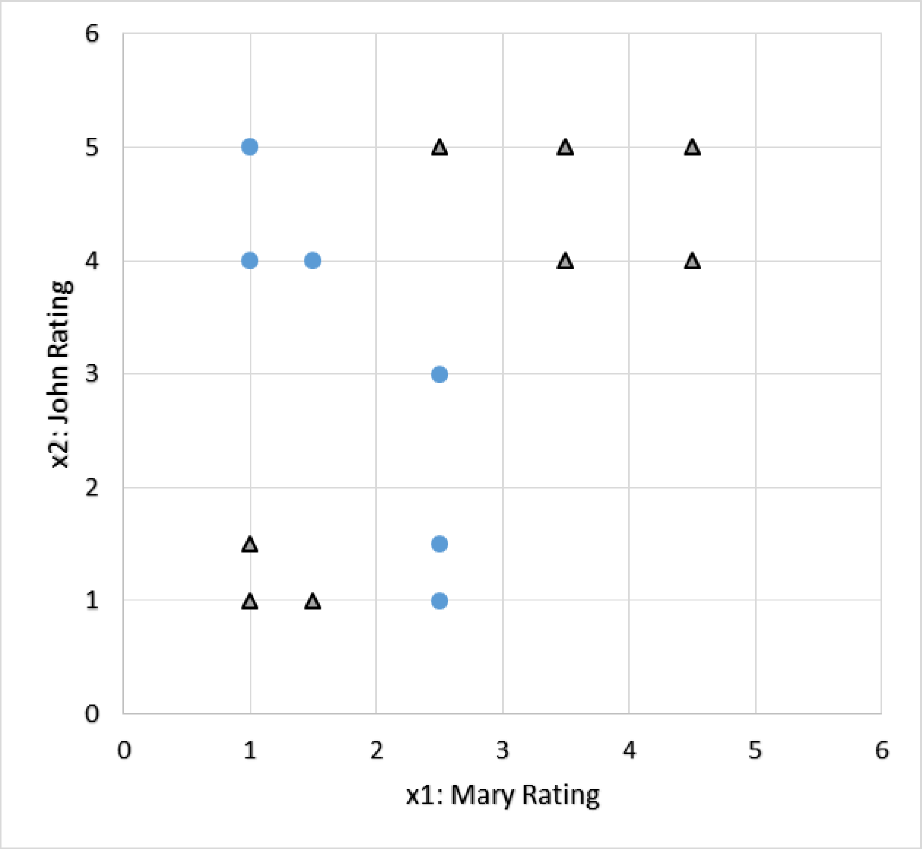
- After a batch of iterations, compare the value of the cost function with its previous value:
  - At the start, define a threshold value (e.g 10^{-4})
  - In each iteration, add the cost function to a running total
  - After N iterations (usually N = size of training dataset):
    - Check if the change in cost function between these N and previous N iterations is below the threshold
    - If so, convergence is reached: can stop iterating
- In case it taking too long to converge, possibly because learning rate is too small:
  - Stop when a maximum number of iterations is reached
  - Should let the user know that it finished without converging
- Will get into greater detail in the next topic

Full Logistic Regression Learning Algorithm: Forward Pass and Stochastic Gradient Descent

```
# Initialisation
Choose values for α, max_iterations, threshold, N
Set w and b to a set of valid initial values (w is same size as x^(i))
Set stopping = False; J_running = 0; J_running_prev = 0; iteration = 0
While not stopping:
    Set {x, y} = single example from training set selected at random
    # Forward propagation stage
    Calculate y_hat from x, w, b
    Calculate J_current from y, y_hat
    # Gradient Descent stage:
    Loop over the j elements of w: Calculate Δw_j
    Calculate Δb
    Loop again over the j elements of w: w_j −= α * Δw_j
    b −= α * Δb
    # Check stopping criteria
    iteration += 1
    J_running += J_current
    if iteration > max_iterations: stopping = True      # Failed to converge
    if (iteration mod N) == 0:    # Have done N iterations: test for convergence on the batch
        Compare J_running with J_running_prev: if less than threshold, stopping = True
        Set J_running_prev = J_running; J_running = 0
```

Logistic Regression Limitations

- Classes must be linearly separable – what about this?
  - Added 3 more movies
  - I like them, but critics don't
- Can we use stacking to combine outputs of 2 separate LR classifiers?
  - Yes!
  - See "MoreMovies" tab of MovieRating.xlsx
- Essentially this is operating like a neural net with 1 hidden layer



- Next topic: will see how to tackle this properly with NN learning.

Combining Logistic Regressors - Illustration

End of Topic 2