Machine Learning

Agenda

- Data Representation
  - Feature Extraction
- Machine Learning Problems
  - Regression
  - Classification
  - Clustering
- Machine Learning Algorithms
  - Gradient Descent
  - KNN
  - Neural Networks
  - K-means

Data Representation

- Critical first step in many learning problems
- How to represent real world objects or concepts
  - Extract numerical information (**features**) from them

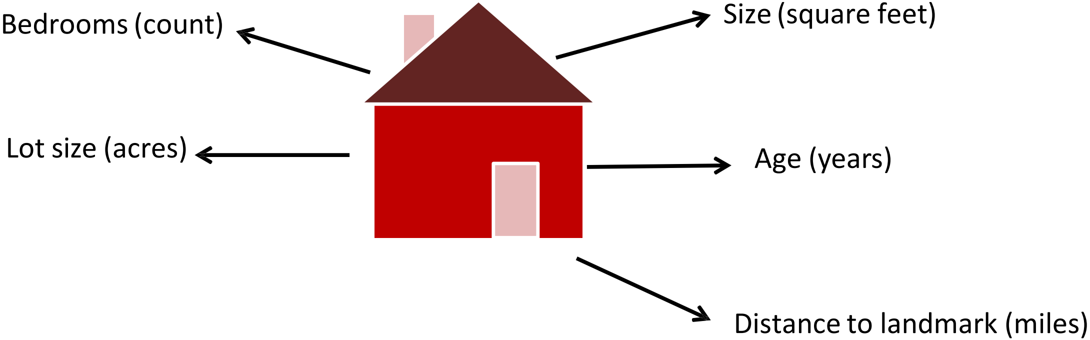
Feature Extraction

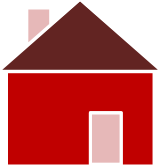
Suppose we want to:

- Cluster houses
- Predict home values
- Classify houses

Need to represent houses as collection of **features**

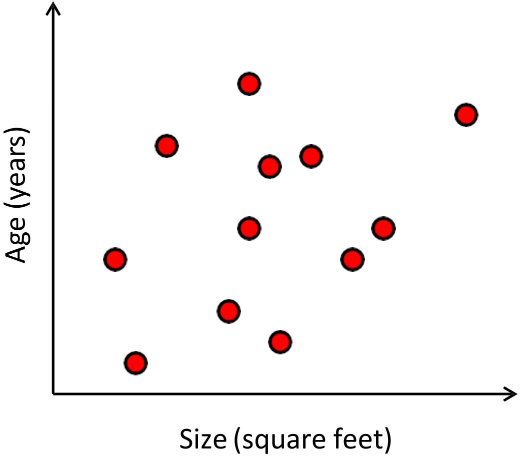
Feature Extraction





Feature Space

House	Size	Age	Lot Size	# BR
1	1400	15	0.5	2
2	800	4	0.25	1
3	2300	35	0.2	4
4	1700	8	0.5	2
…				



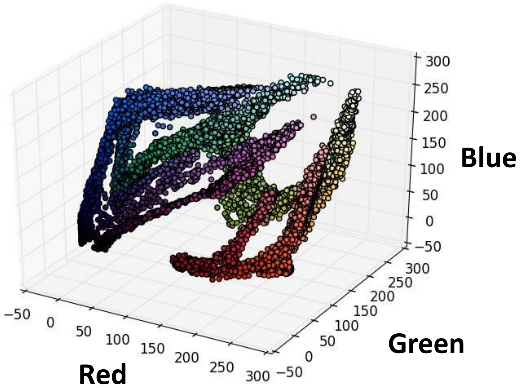
Feature Extraction

Suppose we want to:

- Find similar regions in an image
  - Called **image segmentation**
  - Primitive for higher level learning
  - Cluster pixels

Image from https://flic.kr/p/ahLKUz

Feature Extraction



Data Representation Recap

- Feature selection is critical
- Some preprocessing usually required
  - Scale features
  - Reduce dimensionality (e.g., PCA)
- Once we have data in features space, we can apply ML algorithms

Machine Learning Problems

- **Regression**
  - Fit model (e.g., function) to existing data
  - Several input variables, one response (e.g., output) variable
  - Predict stock prices, home values, etc.
- **Classification**
  - Place items into one of N bins/classes
  - Document / Text classification (e.g., spam vs. not spam, positive tweet vs. negative tweet, etc.)
- **Clustering**
  - Group common items (e.g., documents, tweets, images, people) together
  - Product recommendation

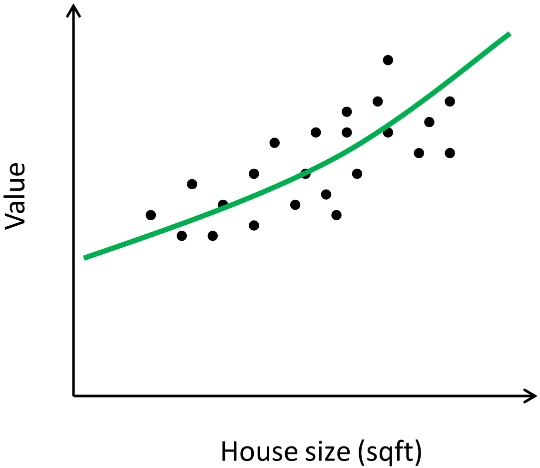
Regression

- Select features
- Embed data in feature space

Regression

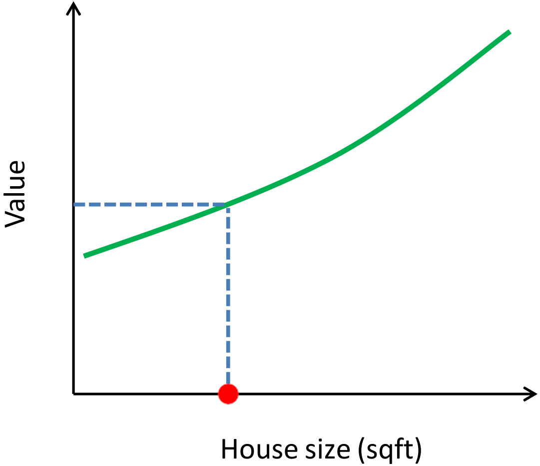
- Select features
- Embed data in feature space
- Predict house values

Regression



- Select features
- Embed data in feature space
- Predict house values
- Run regression algorithm

Regression



- Select features
- Embed data in feature space
- Predict house values
- Run regression algorithm
- Predict values

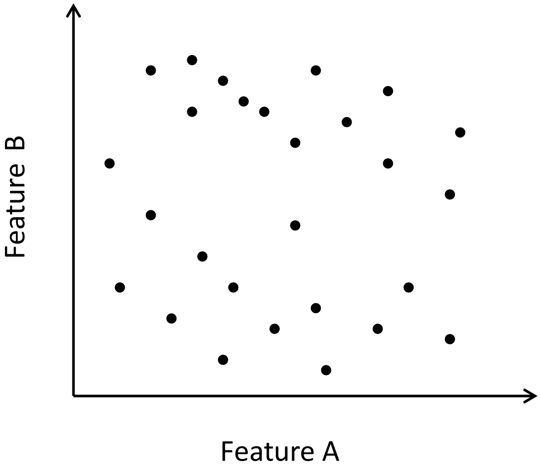
Machine Learning Problems

- **Regression**
  - Fit model (e.g., function) to existing data
  - Several input variables, one response (e.g., output) variable
  - Predict stock prices, home values, etc.
- **Classification**
  - Place items into one of N bins/classes
  - Document / Text classification (e.g., spam vs. not spam, positive tweet vs. negative tweet, etc.)
- **Clustering**
  - Group common items (e.g., documents, tweets, images, people) together
  - Product recommendation

Classification

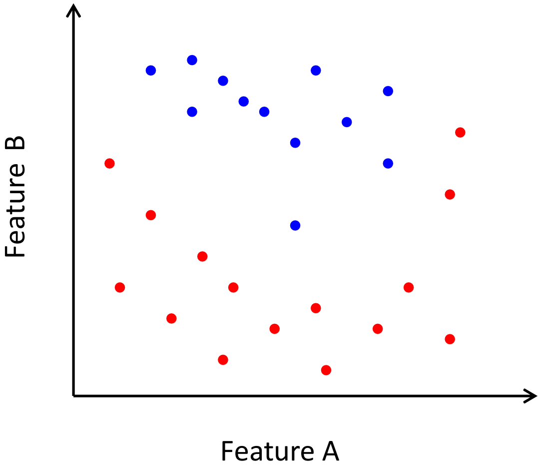
- Sentiment classification
- Face detection
- Medial diagnosis
- Spam detection

Classification



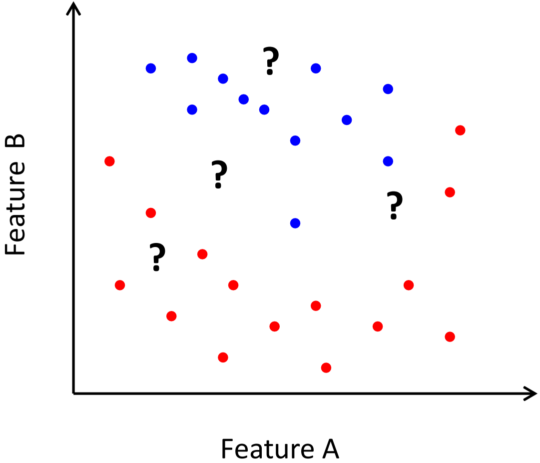
- Select features
- Embed data in feature space

Classification 2D Example



- Select features
- Embed data in feature space
- Label data
  - E.g., **spam** vs. **not spam**

Classification 2D Example



- Select features
- Embed data in feature space
- Label data
  - E.g., **spam** vs. **not spam**
- Classify new observations

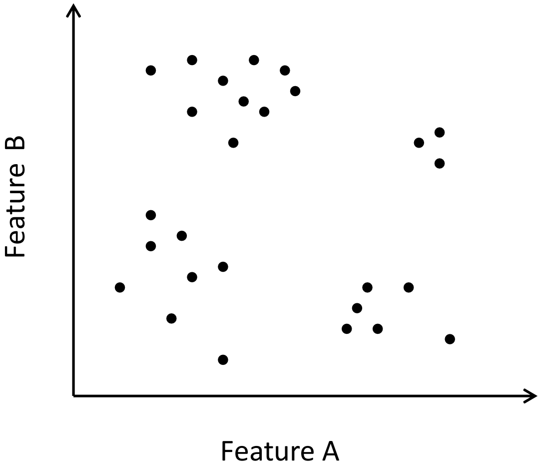
Machine Learning Problems

- **Regression**
  - Fit model (e.g., function) to existing data
  - Several input variables, one response (e.g., output) variable
  - Predict stock prices, home values, etc.
- **Classification**
  - Place items into one of N bins/classes
  - Document / Text classification (e.g., spam vs. not spam, positive tweet vs. negative tweet, etc.)
- **Clustering**
  - Group common items (e.g., documents, tweets, images, people) together
  - Product recommendation

Clustering

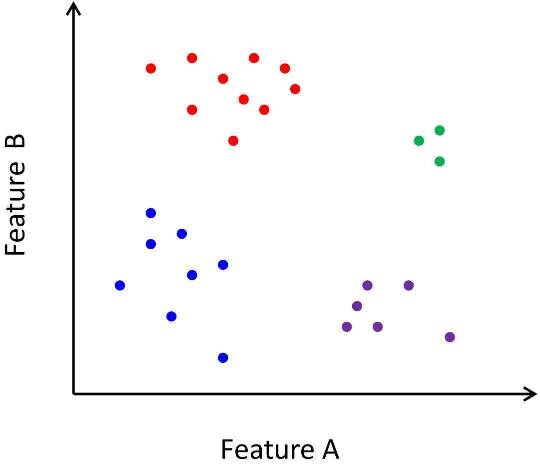
- Group common items
  - documents, tweets, images, people
- Customer segmentation

Clustering



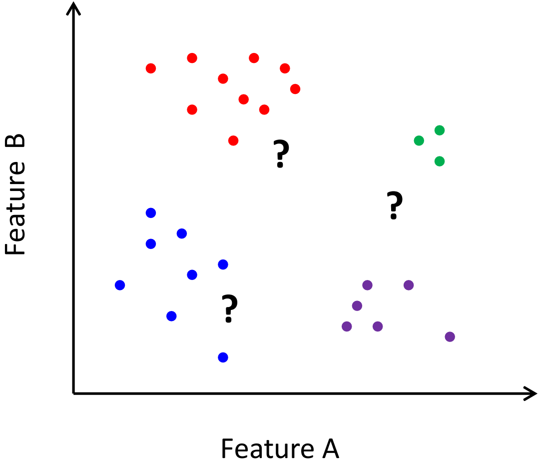
1. Select features
2. Embed data in feature space

Clustering



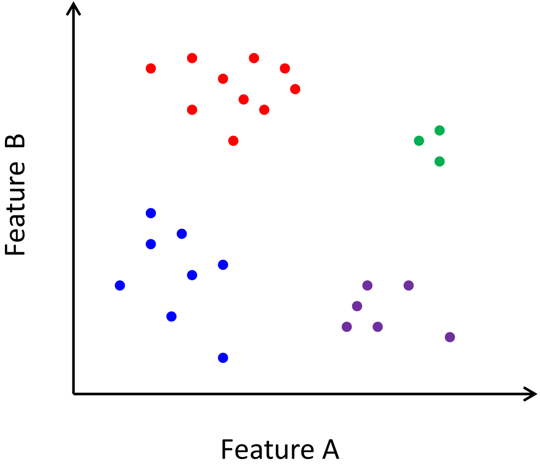
1. Select features
2. Embed data in feature space
3. Apply clustering algorithm

Clustering



1. Select features
2. Embed data in feature space
3. Apply clustering algorithm
4. Can be used to classify new observations

Clustering

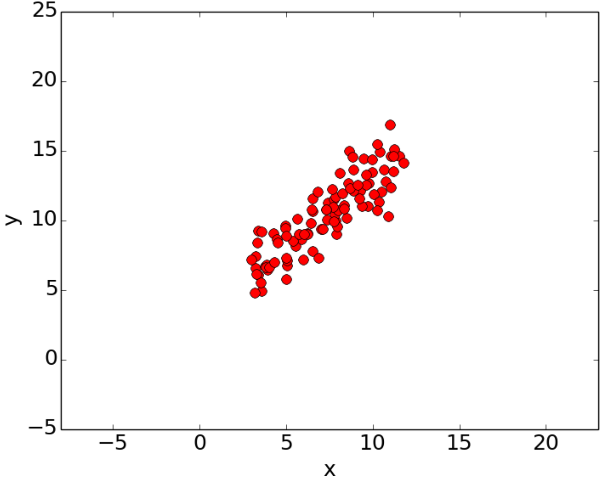


1. Select features
2. Embed data in feature space
3. Apply clustering algorithm
4. Can be used to classify new observations
5. Many other applications

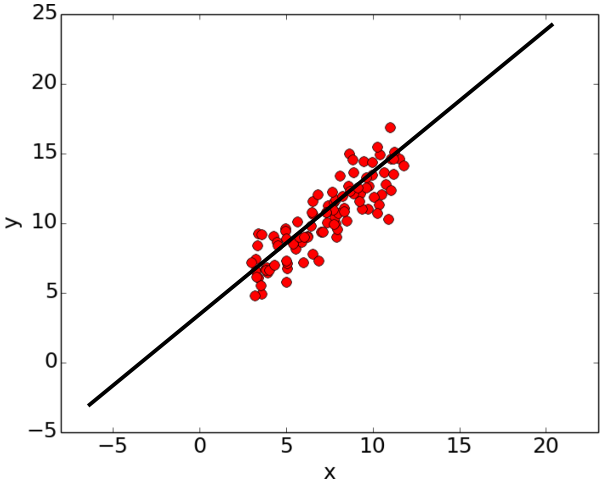
Algorithms

- **Regression**
  - Gradient Descent
- **Classification**
  - K-Nearest Neighbors
  - Neural Networks
- **Clustering**
  - K-means

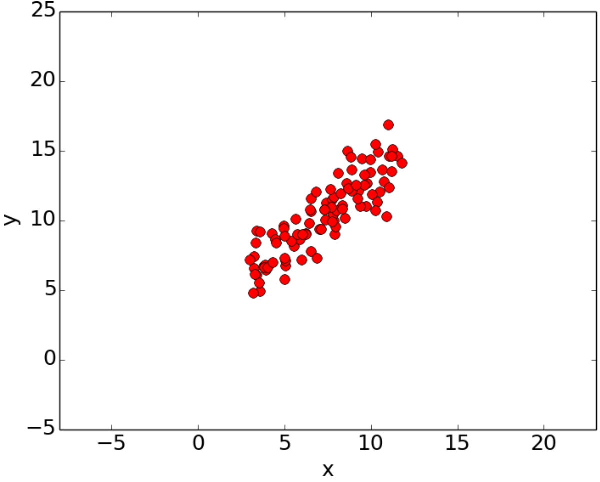
Gradient Descent for Linear Regression



Gradient Descent for Linear Regression



Regression Example

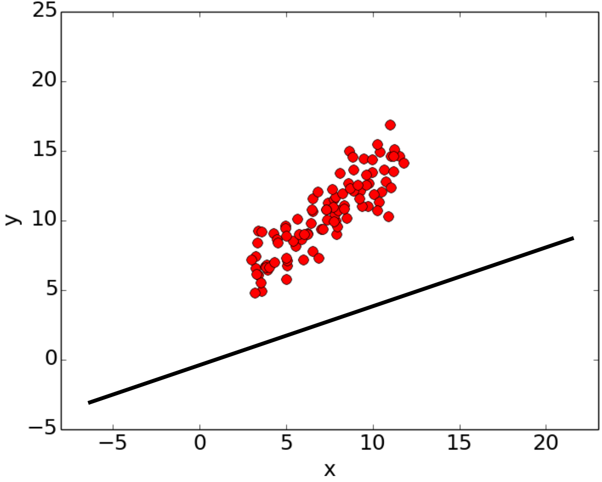


$$y = mx + c$$

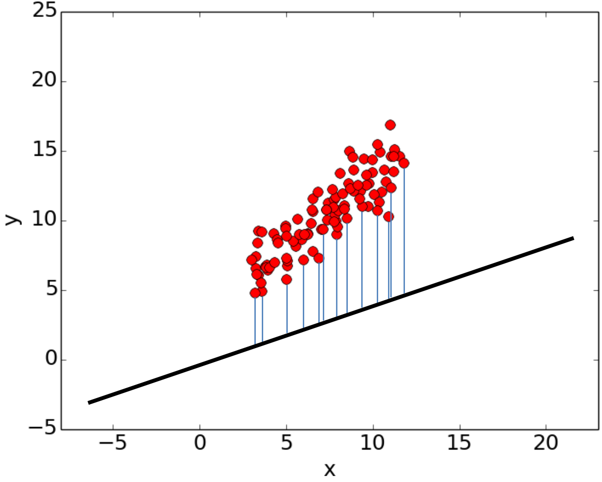
Regression Example

- Need to score candidate lines (m,c) pairs
- Choose the best one

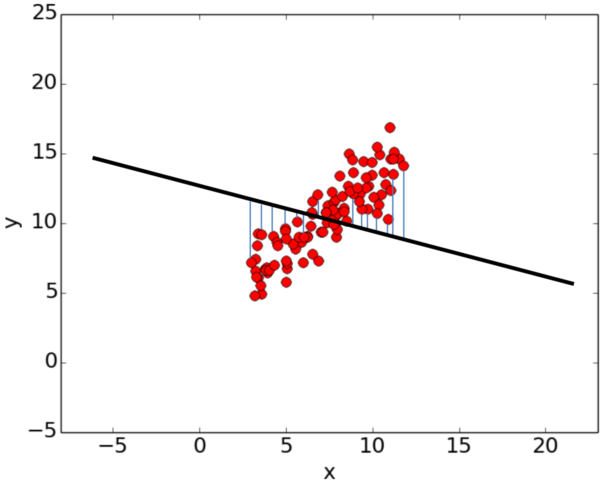
Regression Example - Error



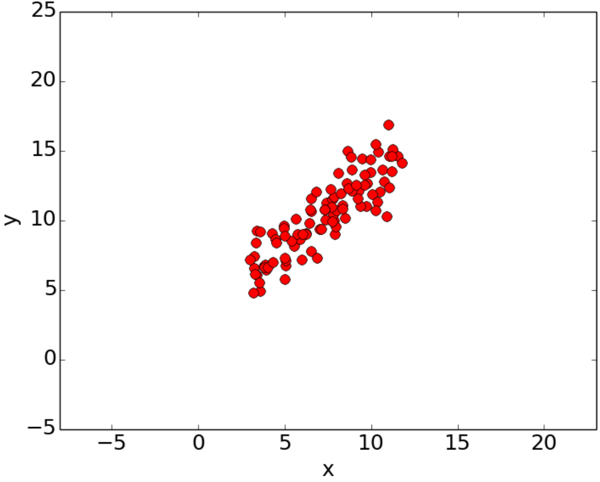
Regression Example - Error



Regression Example - Error



Regression Example - Error



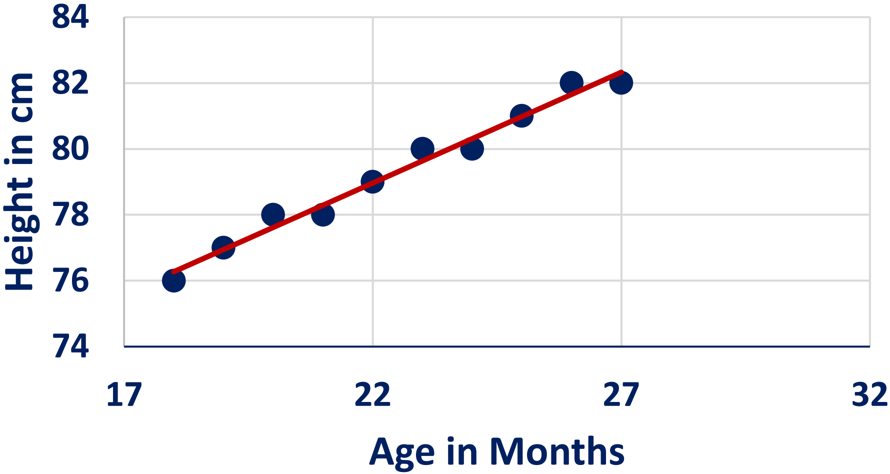
$$y = mx + c$$

$$error = \sum_{i=1}^{N} \left(y_i - (mx_i + c)\right)^2$$

Compute Gradient

Search for solution

Linear Regression



Minimize the Cost Function

Gradient Descent

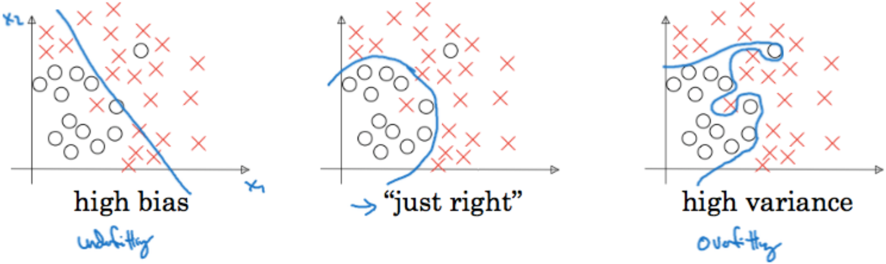
Gradient Descent is an optimization algorithm used to find the values of a function's parameters (m,c) that minimize a cost function as far as possible.

Demo: Gradient Descent Workedout.xlsx

Overfitting

Suppose we want to find the price of a house. We trained the model and model is giving 99% accuracy score on training data.
And then we test the model performance on test data and we get the accuracy score=50%.
Why this much difference? The reason is that the model is overfitted and that's why its performing good on training data and bad on testing data.

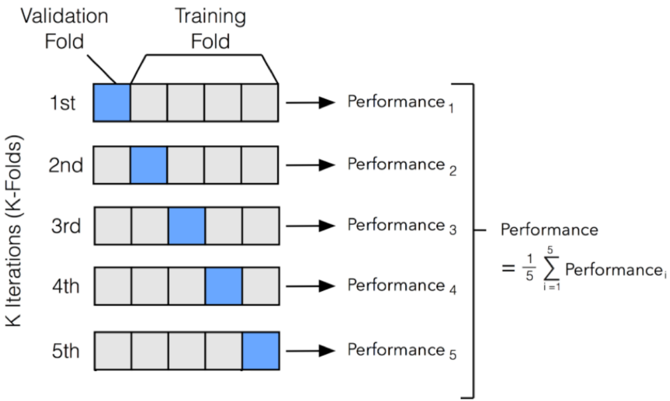
Bias and Variance



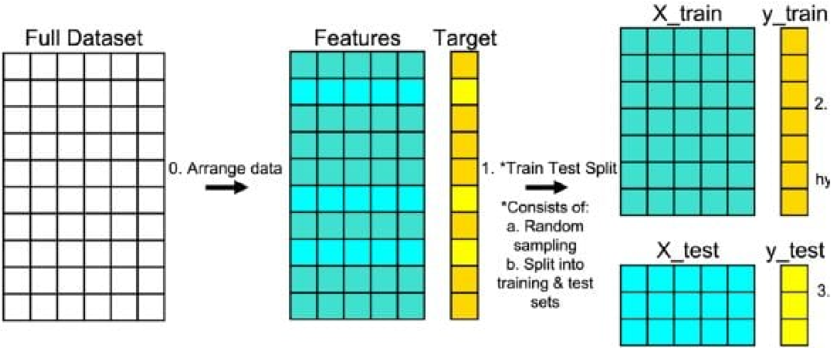
How Does Overfitting can be handled in Machine Learning?

So how can you avoid this happening? By using a technique called cross-validation. This helps limit the amount of data the machine learning algorithm has access to, reducing the chance of Overfitting.

Testing – Cross

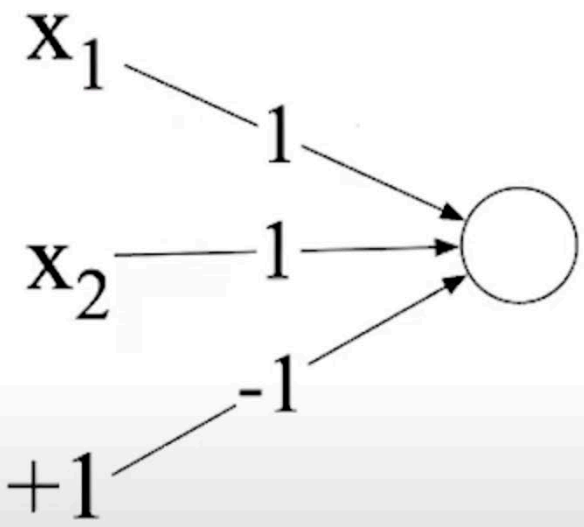


Split the Dataset



XOR Problem

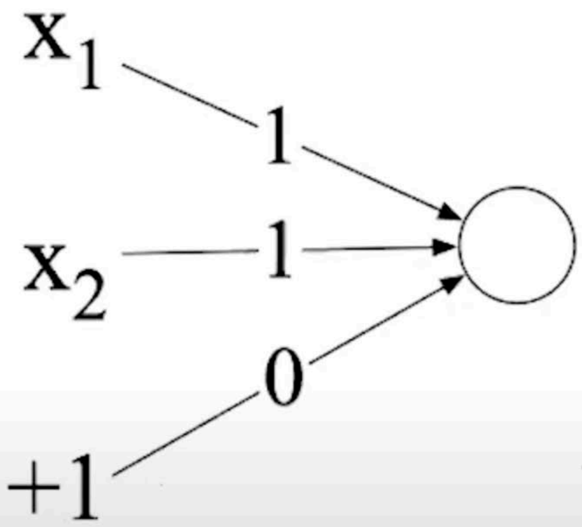
$$y = \begin{cases} 0, & \text{if } w \cdot x + b \le 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$



AND

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

AND



OR

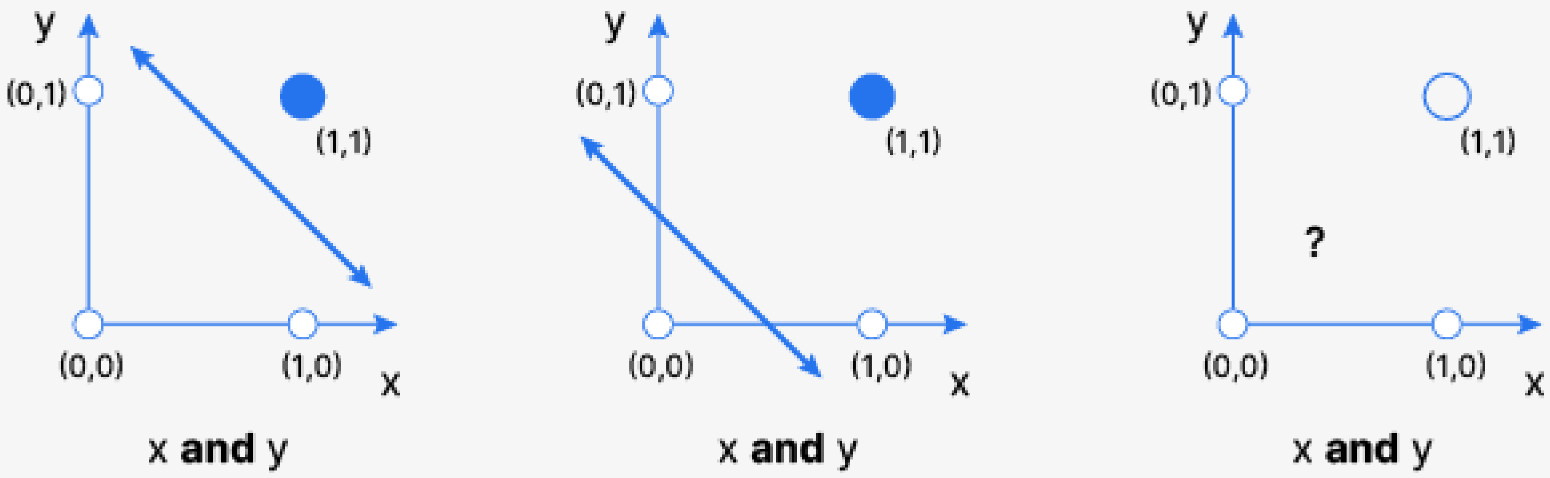
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

OR

Perceptron equation given x_1 and x_2, is the equation of a line

$$w_1x_1 + w_2x_2 + b = 0$$

Linear separability



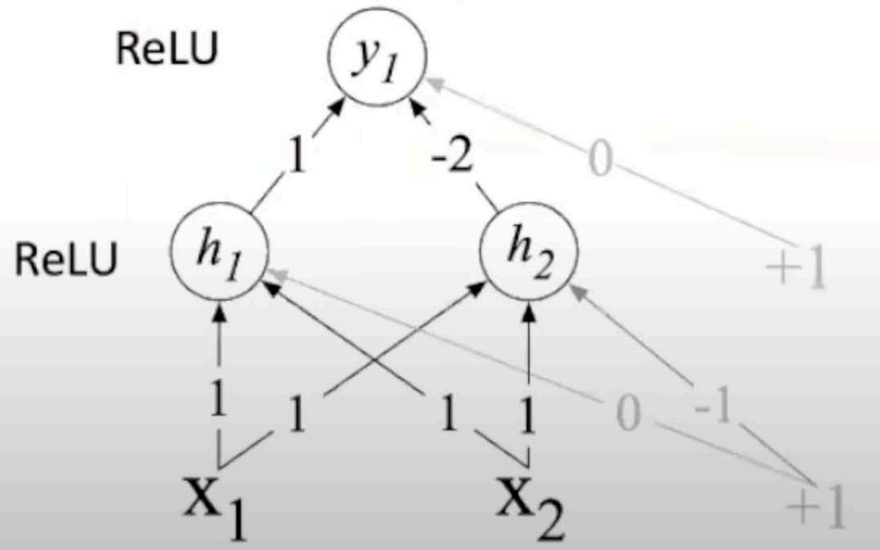
XOR is not linearly separable problem

XOR **can't** be calculated by a single perceptron

XOR **can** be calculated by a layered network of units.

XOR

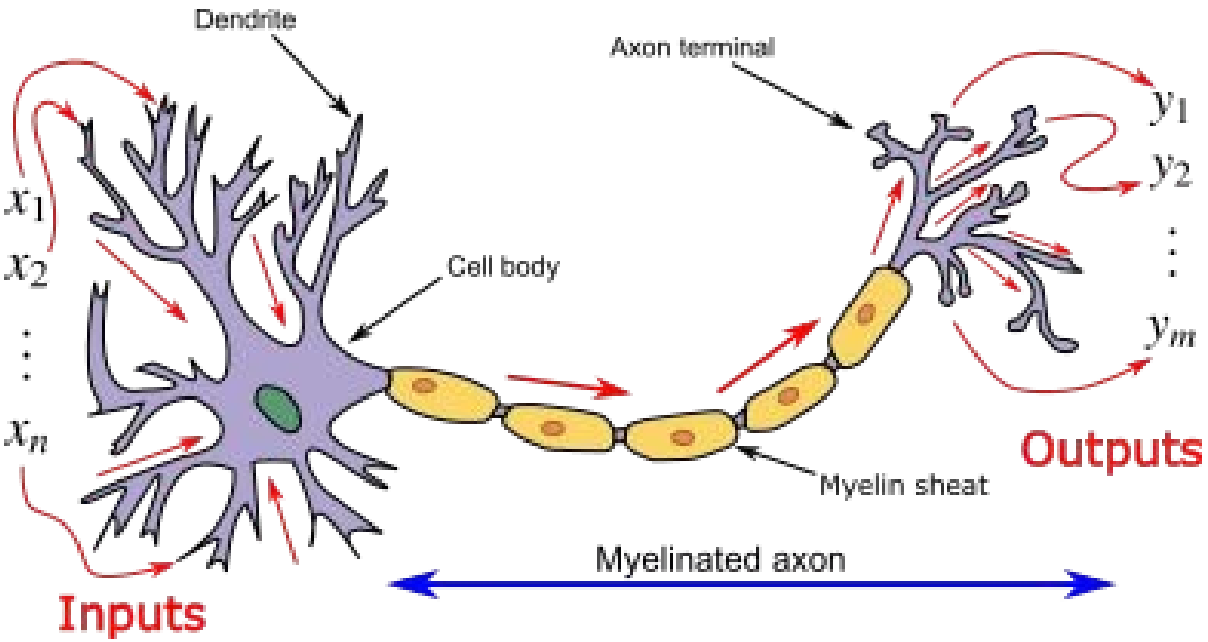
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



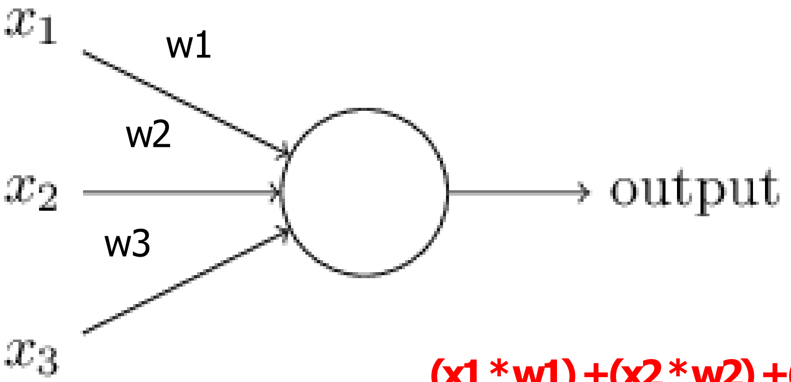
Stanford

Artificial Neural Networks

Neurons


Dendrite
Axon terminal
x_1
x_2
$\vdots$
x_n
Cell body
y_1
y_2
$\vdots$
y_m
Outputs
Myelin sheat
Myelinated axon
Inputs

The Perceptron

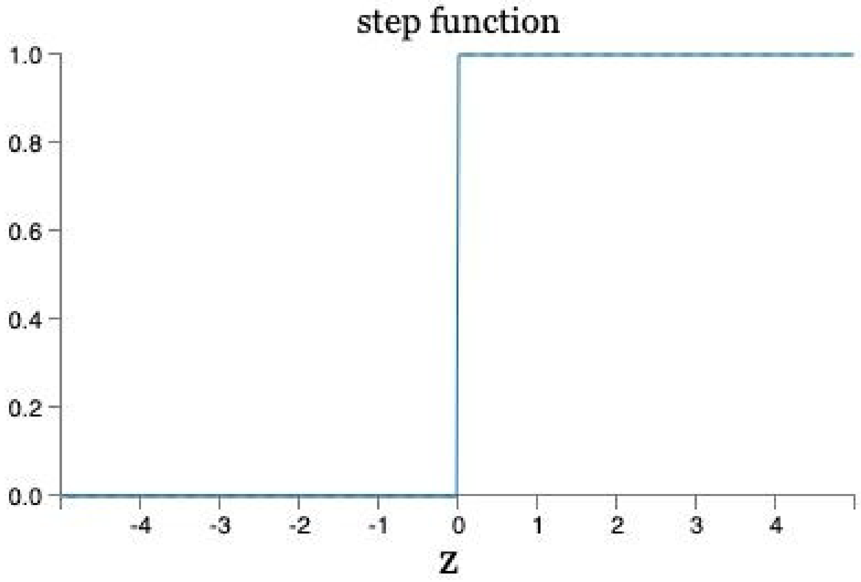


$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

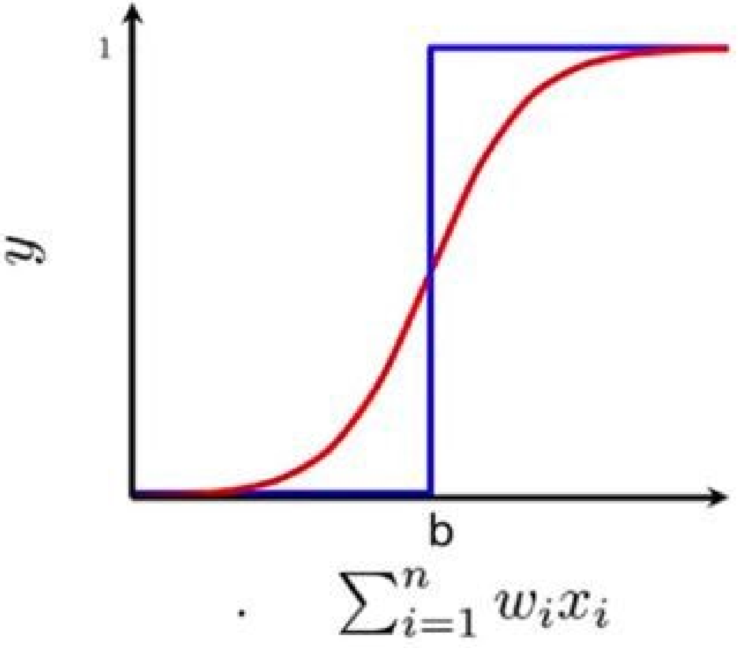
(x1*w1)+(x2*w2)+(x3* w3) >=threshold

((x1*w1)+(x2*w2)+(x3* w3))+bias >0

Activation function



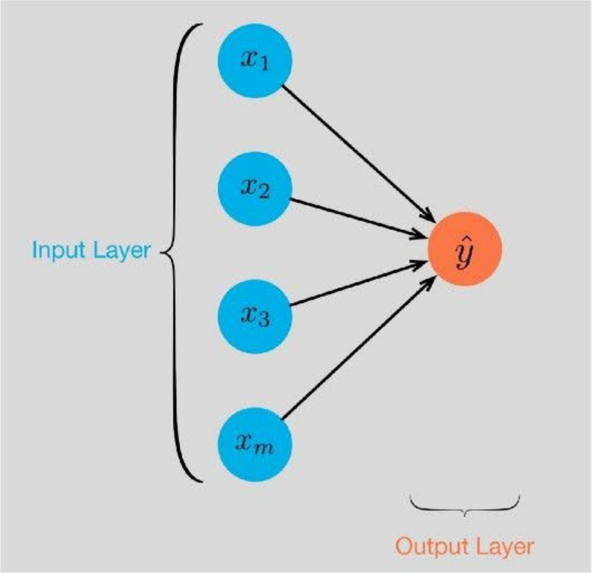
Activation function - Sigmoid

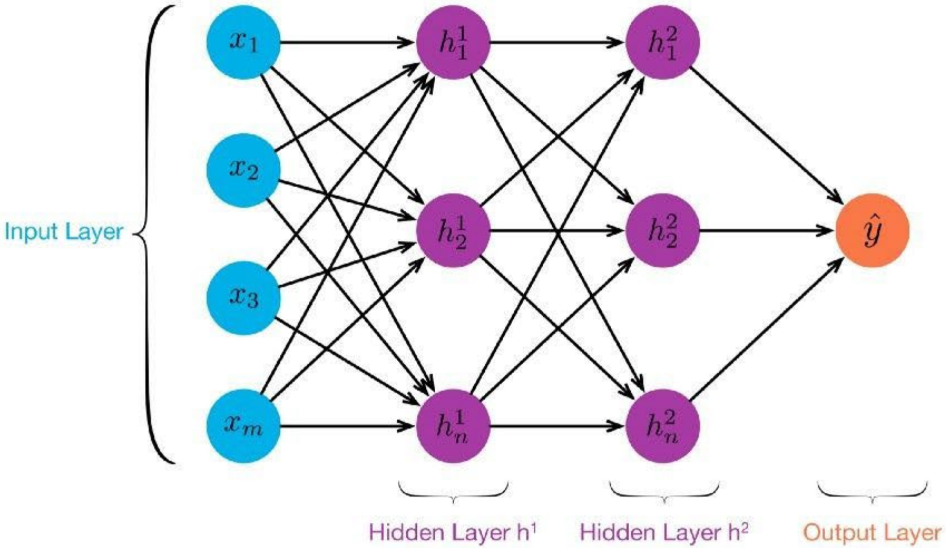


$$y = \frac{1}{1+e^{-(w^T x+b)}}$$

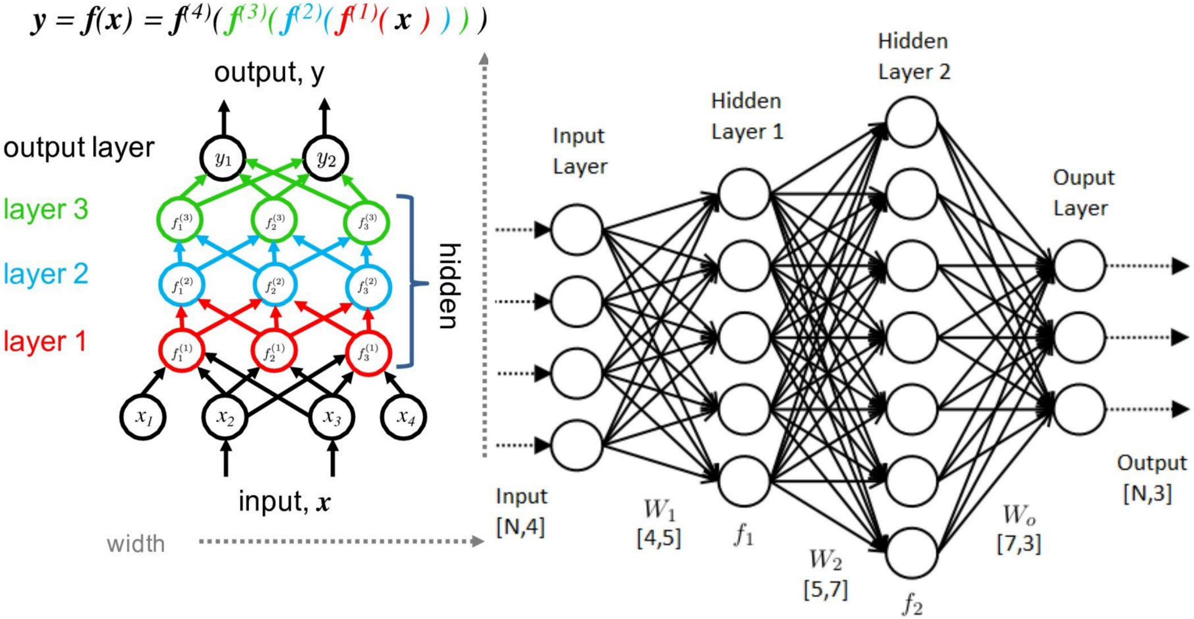
Sum of weights times inputs, plus bias

Neural networks - Structure

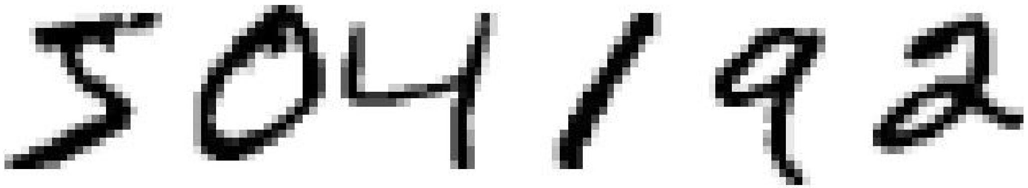




Neural networks?



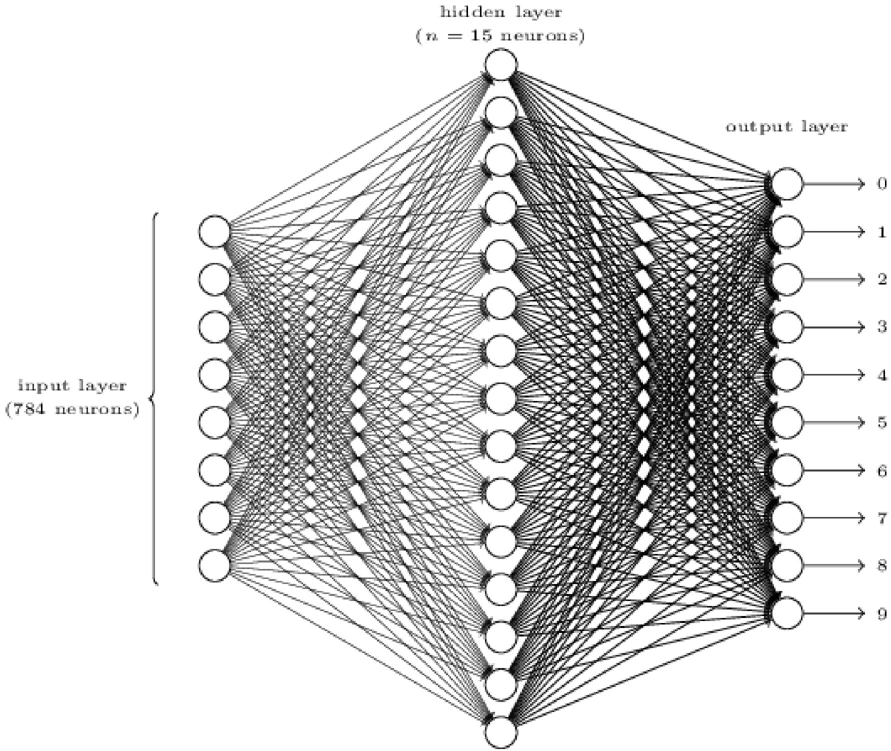
Recognising handwritten numbers



how does it work?

- Start with random numbers for all weights and biases.
- Train the network with training examples
- Assess how well it did by comparing actual output and desired using a **cost function (or loss function)** to compute the error.
- Try and reduce this error by tuning the weights and biases in the network

Cost function - outputs

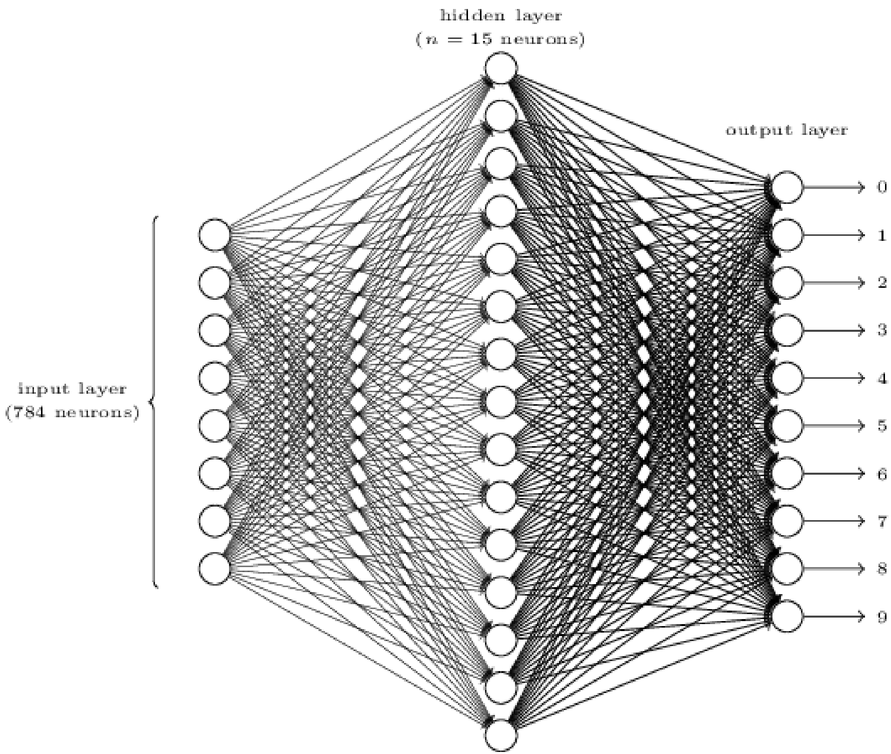


Training a network to recognise a 6

Desired

0
0
0
0
0
0
1
0
0
0
0

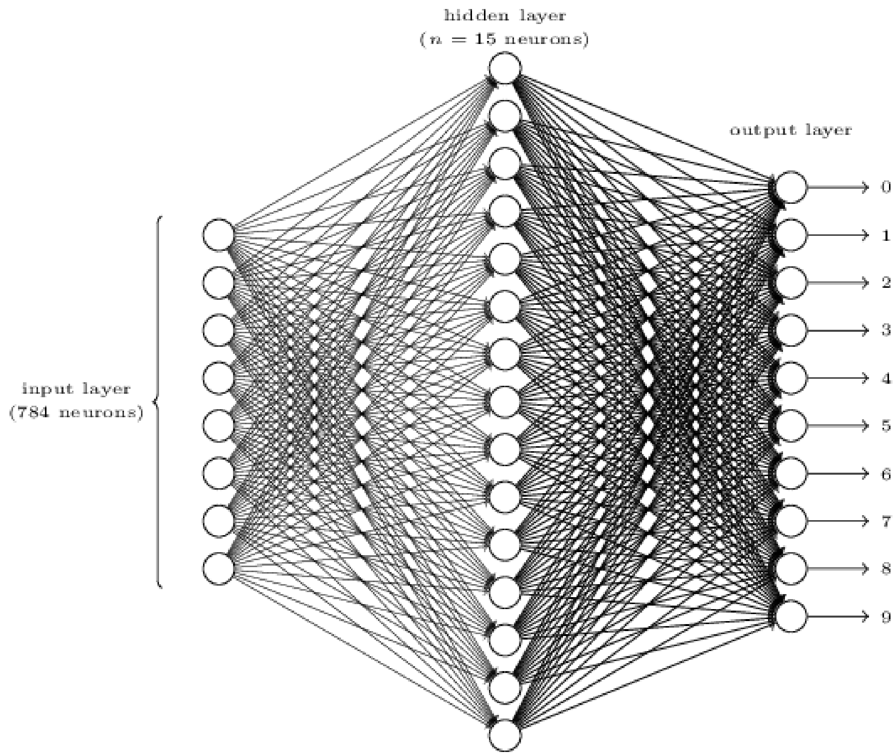
Cost function - outputs



Training a network to recognise a 6

Desired	Actual	|Difference|
0	0.3	0.3
0	0	0
0	0.5	0.5
0	0.2	0.2
0	0	0
0	0.1	0.1
1	0.3	0.7
0	0	0
0	0	0
0	0.9	0.9
0	0.8	0.8

Cost function



Training a network

Desired **Actual** **|Difference|**

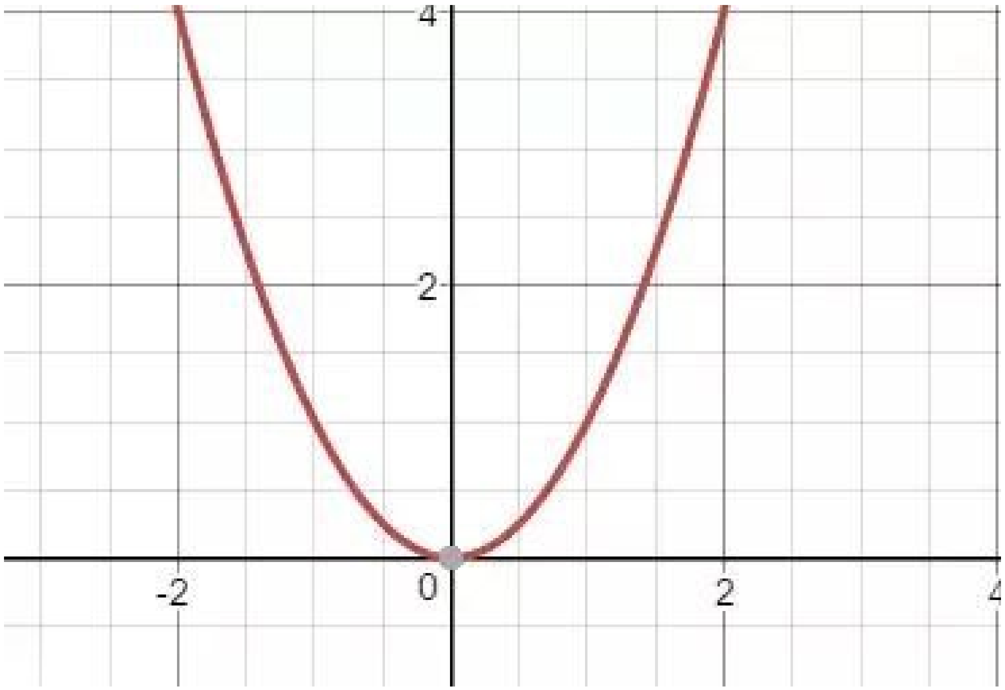
$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

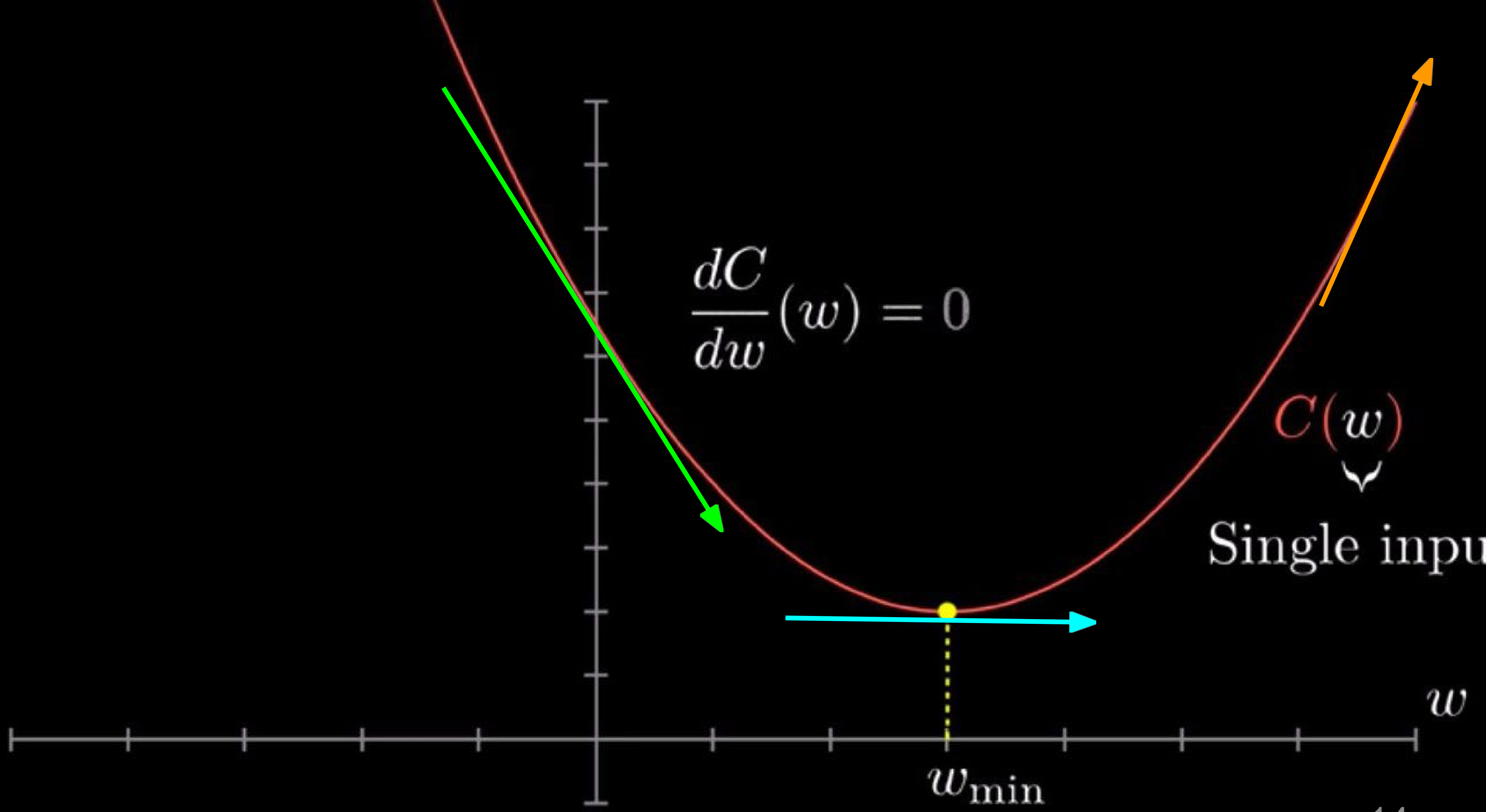
weights

biases

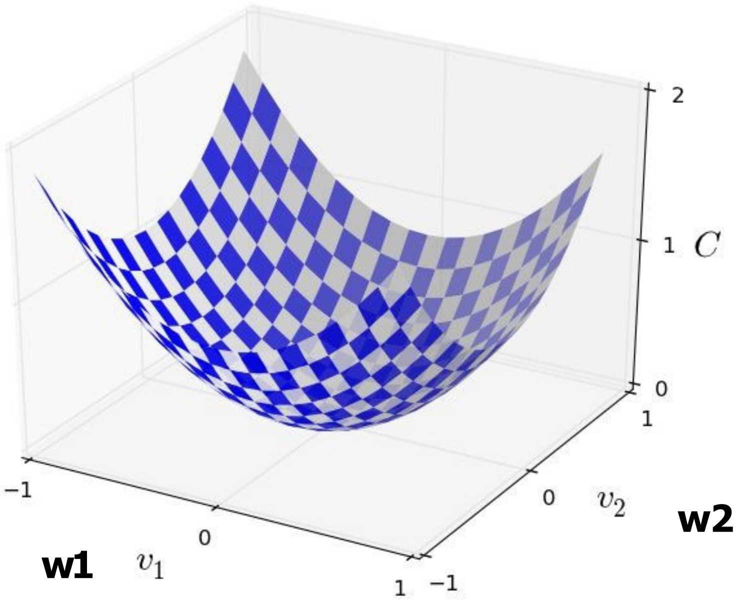
Number of training inputs

How to minimise a function?




$\frac{dC}{dw}(w) = 0$
$C(w)$
Single inpu
w
$w_{\min}$

How to minimise a function? C(w1,w2)



- Two variables = 3D graph
- 3+ variables = ??? puny human brain. But that's fine, we can use the derivative.
- Use partial differentiation to understand derivative of a function with multiple inputs

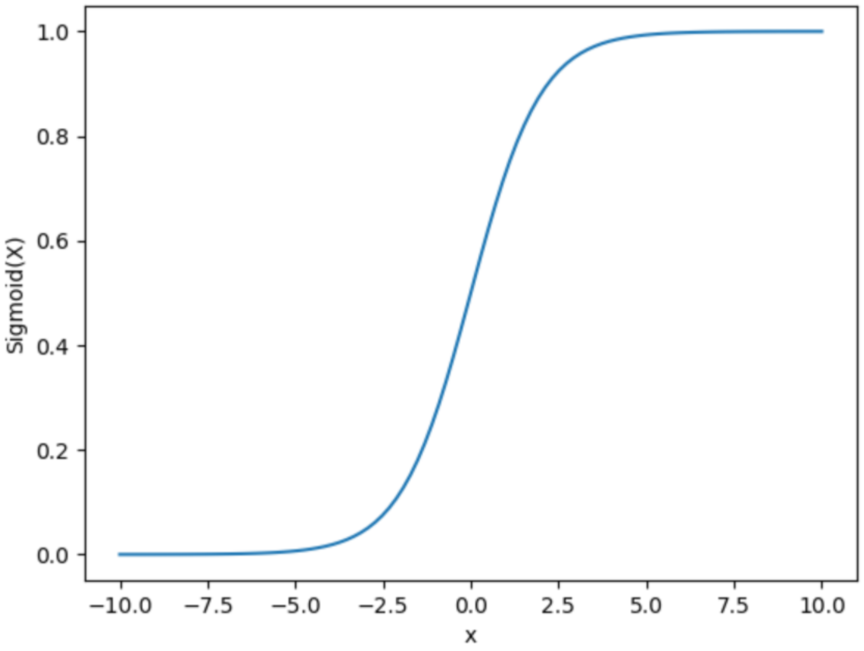
$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2.$$

Activation Functions

What is an activation function and why use them?

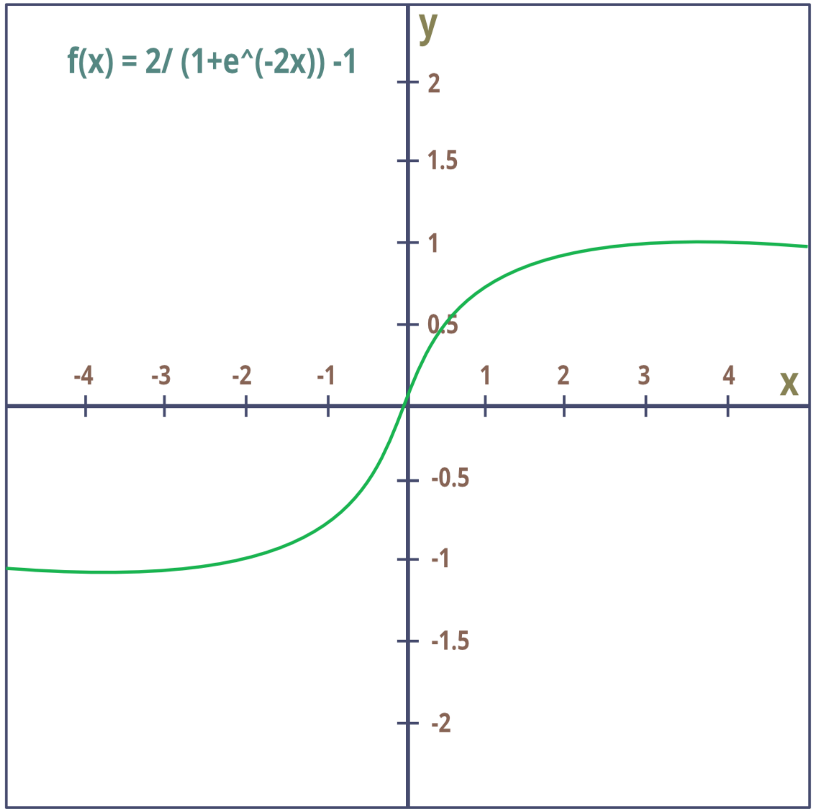
The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

Sigmoid Function



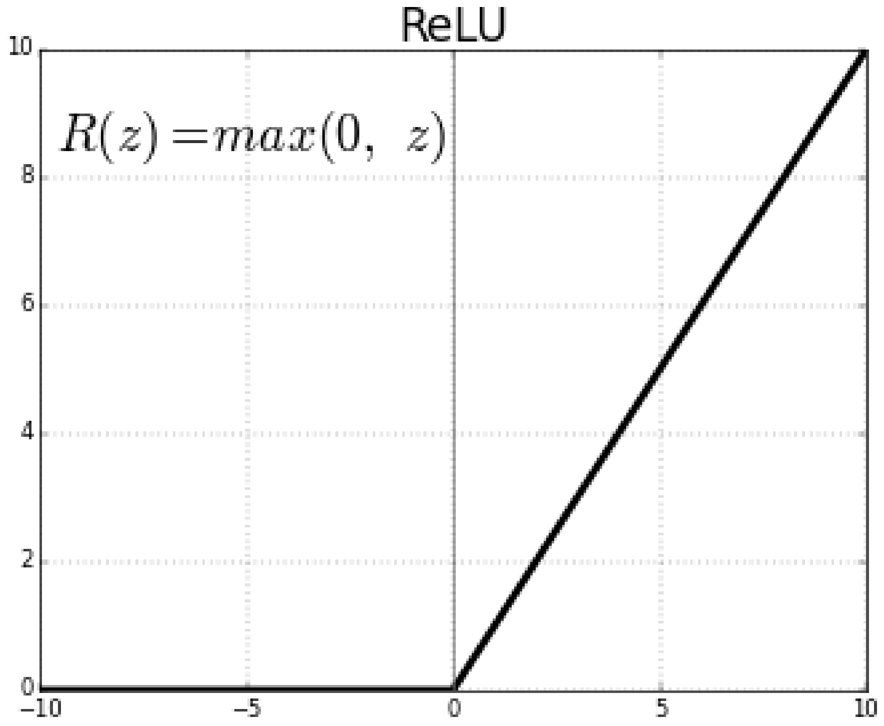
- It is a function which is plotted as **‘S’** shaped graph.
- **Equation :** $A = 1/(1 + e^{-x})$
- **Nature :** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **Value Range :** 0 to 1
- **Uses :** Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.

Tanh Function



- The activation that works almost always better than sigmoid function is Tanh function also known as **Tangent Hyperbolic function**. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
- **Equation :-**
f(x) = tanh(x) = 2/(1 + e-2x) – 1
OR
tanh(x) = 2 * sigmoid(2x) – 1
- **Value Range :-** -1 to +1
- **Nature :-** non-linear
- **Uses :-** Usually used in hidden layers of a neural network as it's values lies between **-1 to 1** hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.

RELU Function



•It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.

•**Equation :-** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.

•**Value Range :-** [0, inf)

•**Nature :-** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.

•**Uses :-** ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

Softmax Function

The softmax function is also a type of sigmoid function but is handy when we are trying to handle multi- class classification problems.

Training the Neural Network Model

- The best values of weights and biases to be identified.

This is done called as the training phase.

Forward and Backward Propagation

Forward propagation: We work forward through the network producing an output result for a current given input from the dataset. A loss function is then evaluated that tells us how well the network did at predicting the correct outputs.

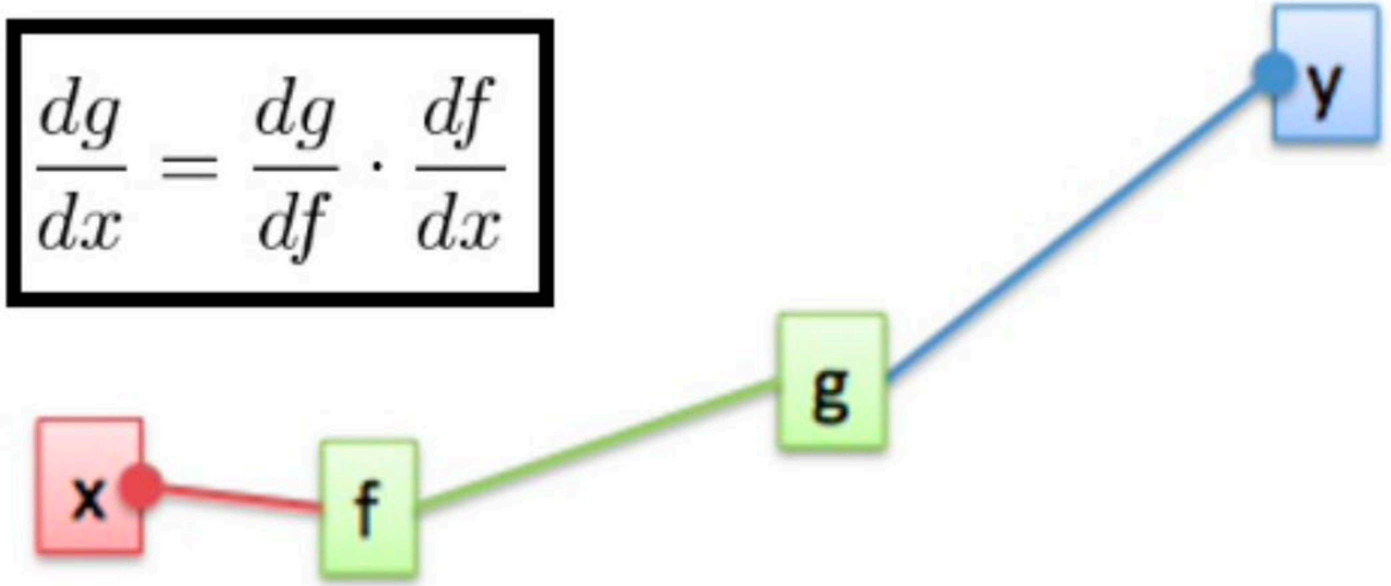
Backward propagation: We work backward through the network calculating the impact each weight had on producing the current loss of the network.

Backpropagation and the chain rule

If you want to know the influence of *x* on the function *g*, we just multiply the influence of *f* on g by the influence of *x* on *f*

$$y = g(f(x))$$

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

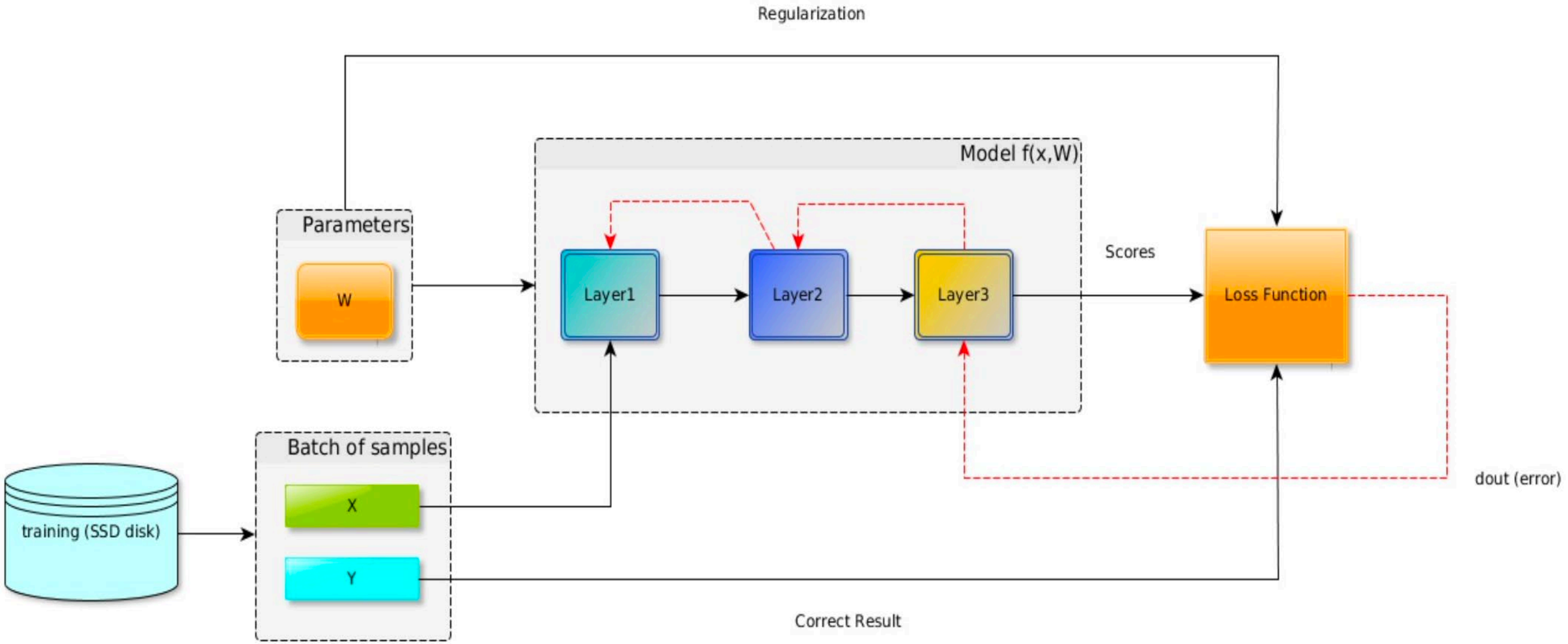


Batches

During training, they divide the dataset into small pieces, named mini batches (or commonly just batches). Then, in turn, each mini batch is loaded and fed to the network where the backpropagation and gradient descent algorithms will be calculated and weights then updated. This is then repeated for each mini batch until you have gone through the dataset completely.

Loss functions

- **Log Loss** - Classification tasks (returning a label from a finite set) with only two possible outcomes
- **Cross-Entropy Loss** - Classification tasks (returning a label from a finite set) with more than two outcomes
- **L1 Loss** - Regression tasks (returning a real valued number)
- **L2 Loss** - Regression tasks (returning a real valued number)


Regularization
Model f(x,W)
Parameters
W
Layer1
Layer2
Layer3
Scores
Loss Function
Batch of samples
X
Y
training (SSD disk)
dout (error)
Correct Result

The optimizer and its hyperparameters

- Gradient descent with momentum
- RMSProp
- Adam

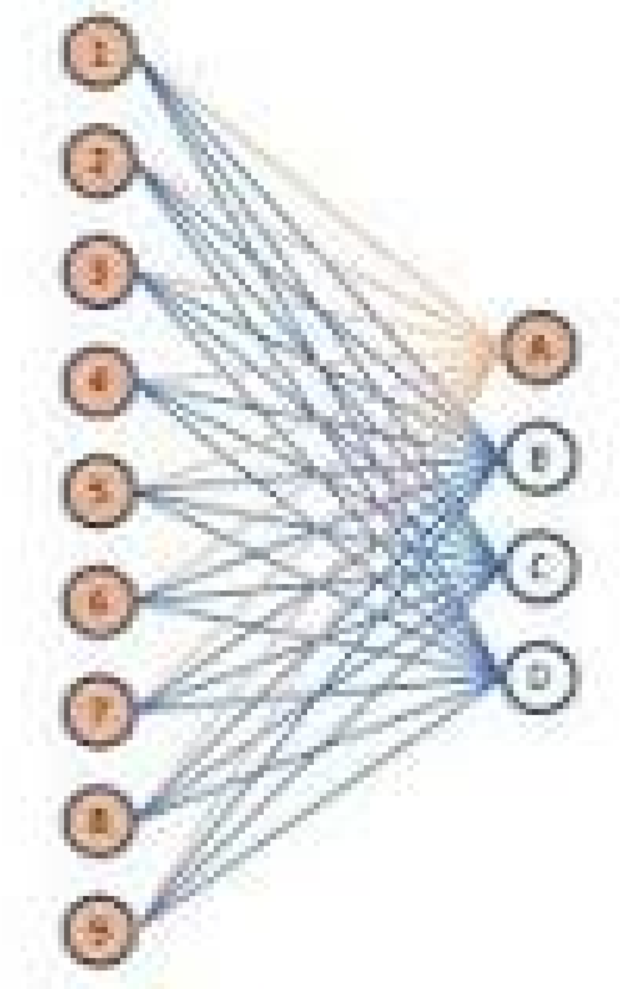
Underfitting versus overfitting

When designing a neural network to solve a specific problem, have to take care of:

- Preparing your dataset
- Choosing the number of layers/number of neurons
- Choosing optimizer hyper-parameters

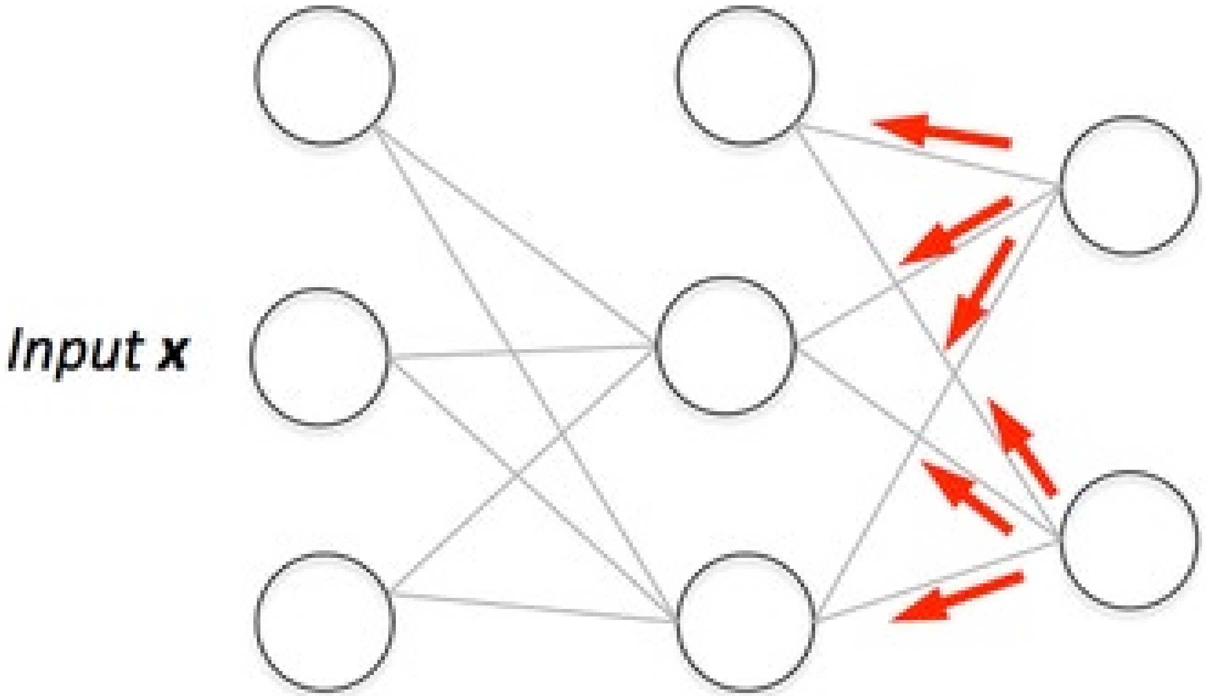
Fully connected layers

- A fully connected layer refers to a neural network in which each input node is connected to each output node.
- In a convolutional layer, not all nodes are connected.

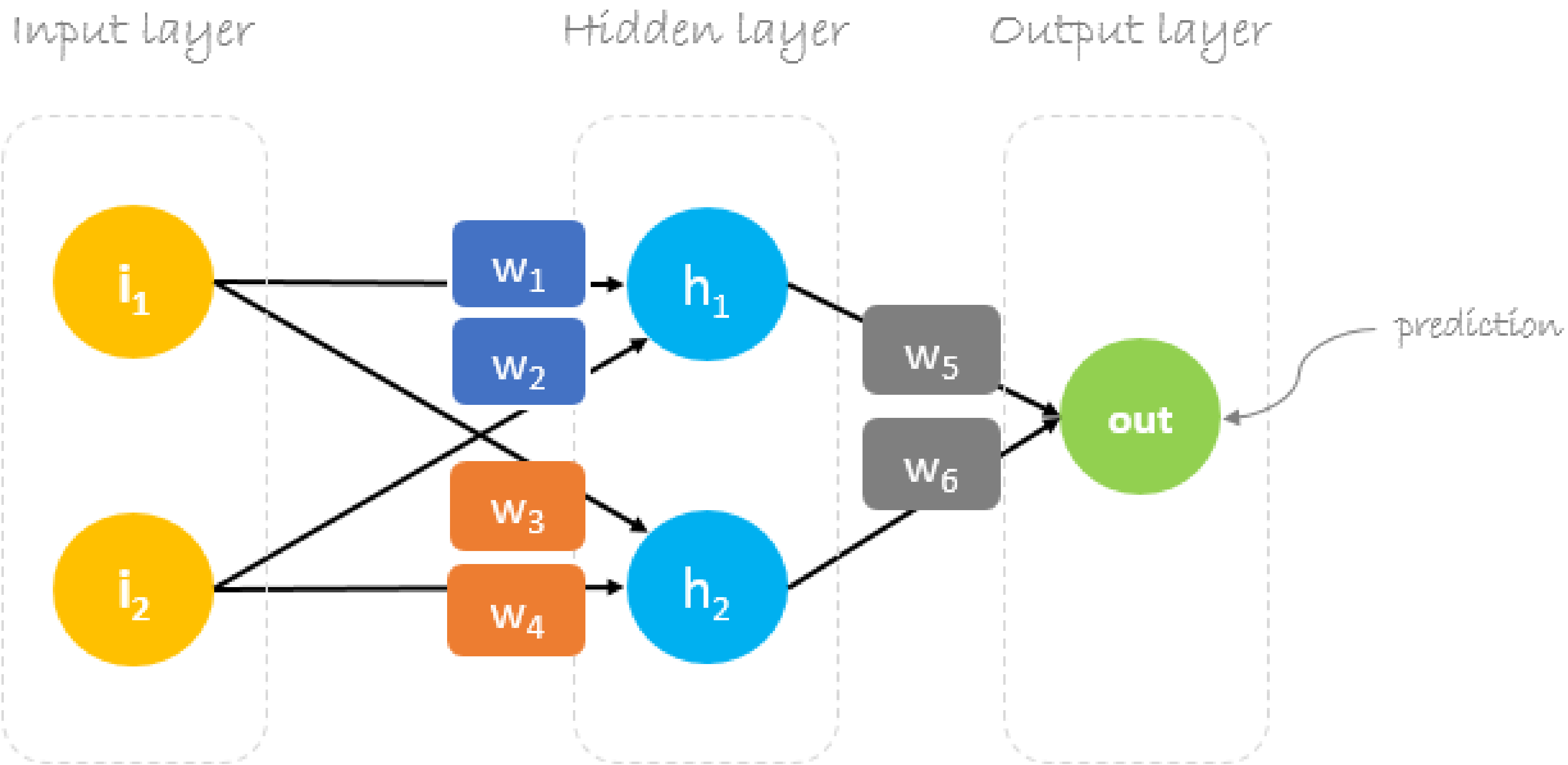


Backpropagation

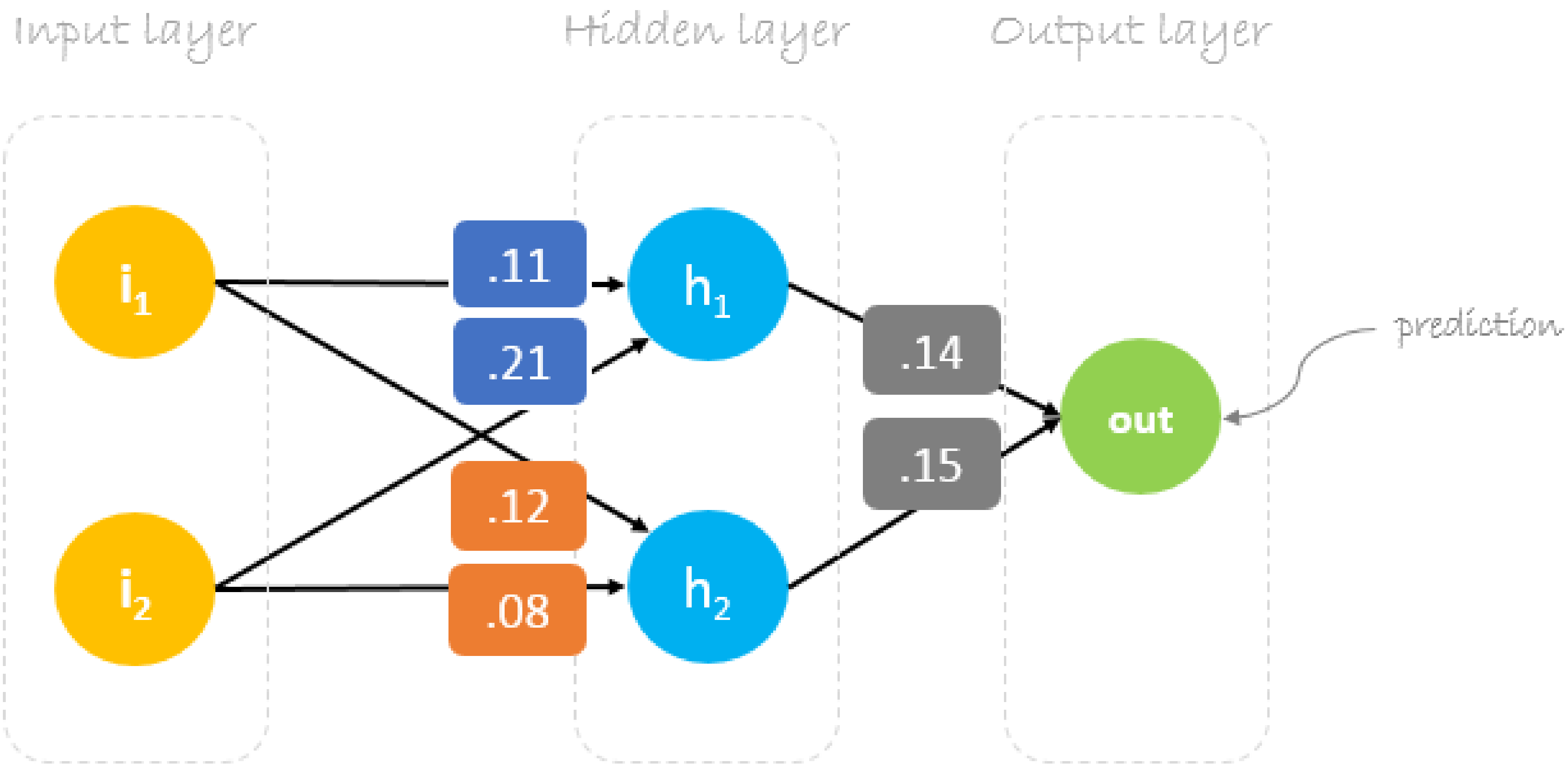
https://hmkcode.com/ai/backpropagation-step-by-step/



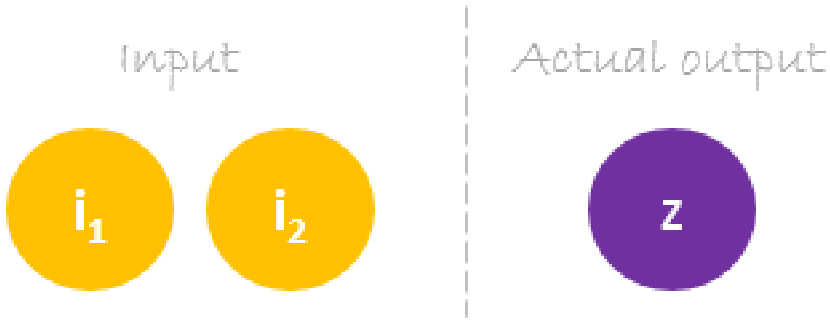
Backpropagation

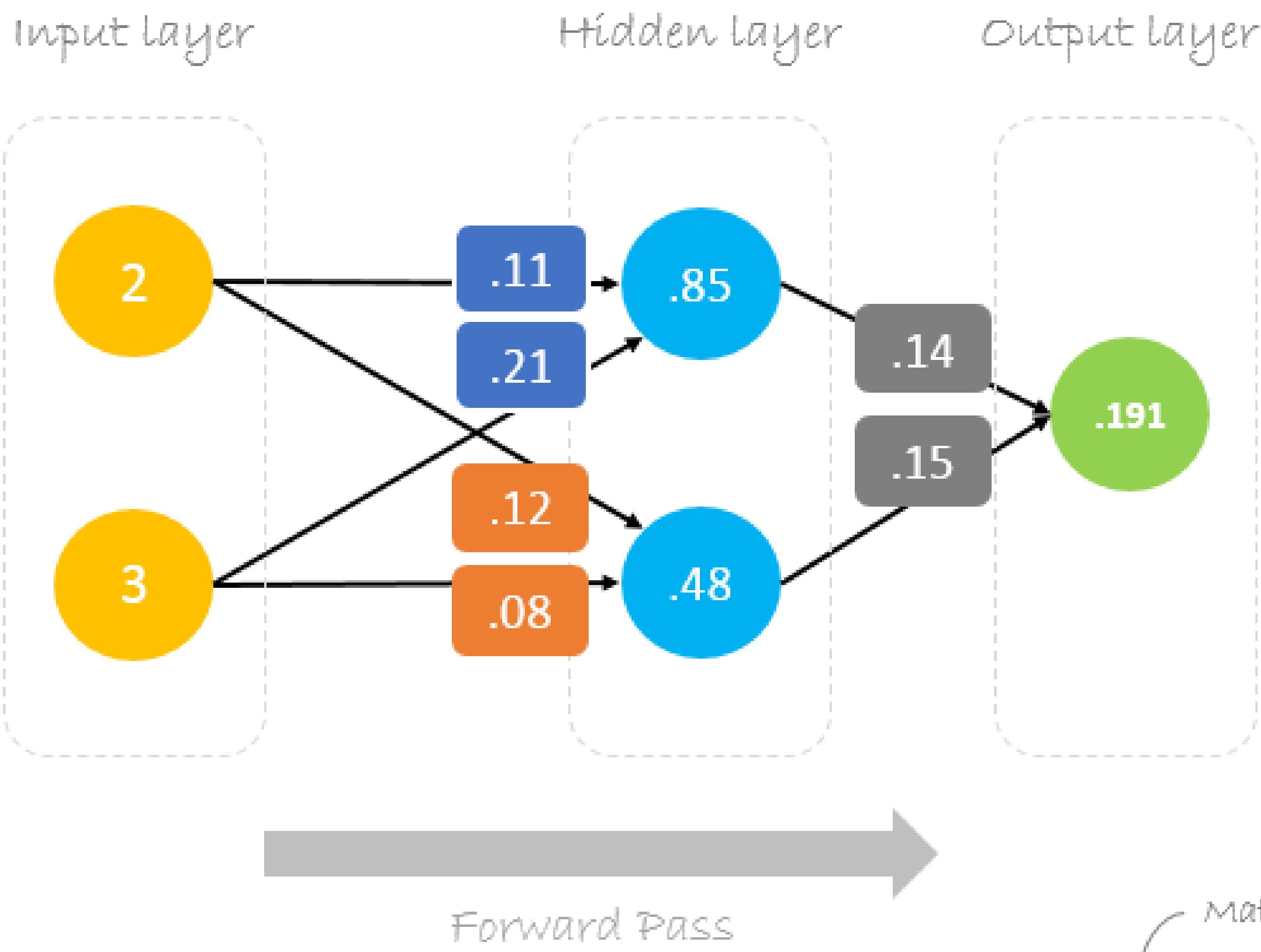


Backpropagation



Backpropagation





Matrix multiplication

$$[2 \quad 3] \cdot \begin{bmatrix}0.11 & 0.12\\ 0.21 & 0.08\end{bmatrix} = [0.85 \quad 0.48] \cdot \begin{bmatrix}0.14\\ 0.15\end{bmatrix} = [0.191]$$

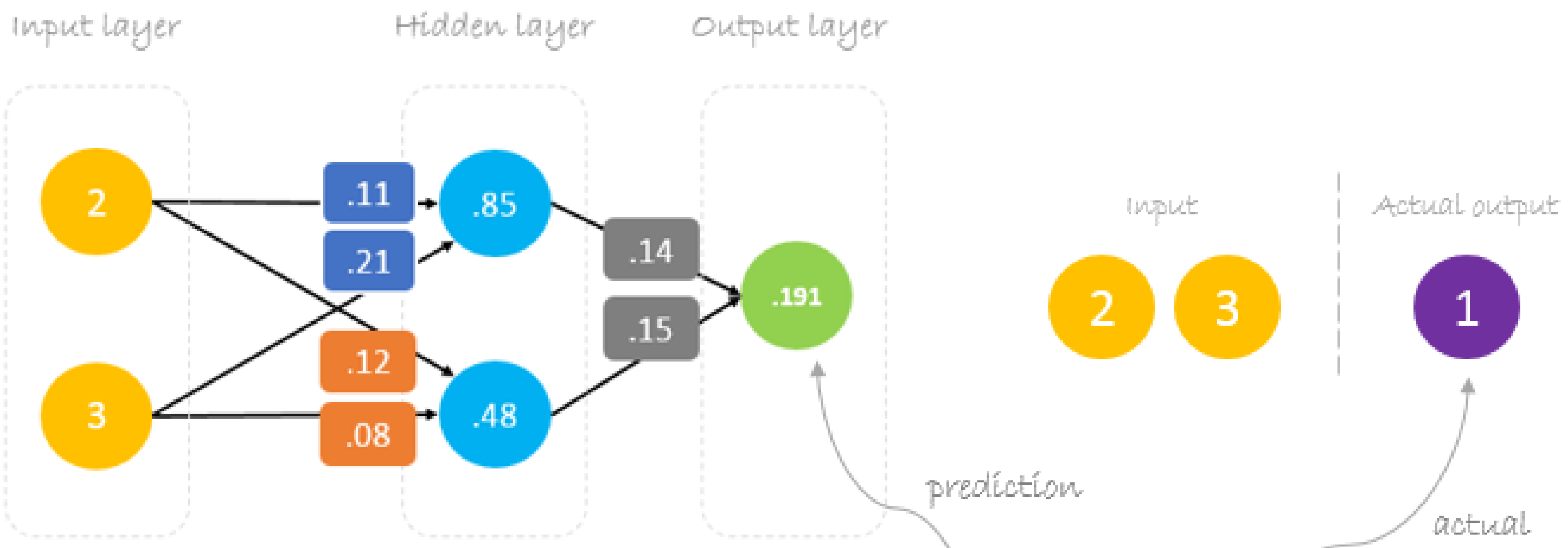
Details

2 x .11 + 3 x .21 = .85

2 x .12 + 3 x .08 = .48

.85 x .14 + .48 x .15 = .191

Calculating Error



Error = 0 , if prediction = actual

$$\boxed{\text{Error} = \frac{1}{2}(\text{prediction} - \text{actual})^2}$$

Error is always positive because of the square

$\frac{1}{2}$ is added to ease the calculation of the derivative

$$\boxed{\text{Error} = \frac{1}{2}(0.191 - 1.0)^2 = 0.327}$$

Reducing Error

$$\text{prediction} = \text{out}$$

$$\text{prediction} = (h_1)\, w_5 + (h_2)\, w_6$$

$$h_1 = i_1\, w_1 + i_2\, w_2$$

$$h_2 = i_1\, w_3 + i_2\, w_4$$

$$\text{prediction} = (i_1\, \boldsymbol{w_1} + i_2\, \boldsymbol{w_2})\, \boldsymbol{w_5} + (i_1\, \boldsymbol{w_3} + i_2\, \boldsymbol{w_4})\, \boldsymbol{w_6}$$

to change ***prediction*** value,
we need to change ***weights***

Backpropagation

Backpropagation, short for “backward propagation of errors”, is a mechanism used to update the **weights** using gradient descent. It calculates the gradient of the error function with respect to the neural network’s weights. The calculation proceeds backwards through the network.

Gradient descent is an iterative optimization algorithm for finding the minimum of a function; in our case we want to minimize the error function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point.

Backpropagation

Derivative of Error with respect to weight

Old weight

$$^*W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$$

New weight

Learning rate

$$^*W_6 = W_6 - a\ \Delta h_2$$

$$^*W_5 = W_5 - a\ \Delta h_1$$

Backpropagation

updated weights

$$^{*}w_6 = w_6 - a\,(h_2 \,.\, \Delta)$$

$$^{*}w_5 = w_5 - a\,(h_1 \,.\, \Delta)$$

$$^{*}w_4 = w_4 - a\,(i_2 \,.\, \Delta w_6)$$

$$^{*}w_3 = w_3 - a\,(i_1 \,.\, \Delta w_6)$$

$$^{*}w_2 = w_2 - a\,(i_2 \,.\, \Delta w_5)$$

$$^{*}w_1 = w_1 - a\,(i_1 \,.\, \Delta w_5)$$

Backpropagation

Formulas in Matrices

updated weights

$$^{*}w_6 = w_6 - a\,(h_2\,.\,\Delta)$$

$$^{*}w_5 = w_5 - a\,(h_1\,.\,\Delta)$$

$$^{*}w_4 = w_4 - a\,(i_2\,.\,\Delta w_6)$$

$$^{*}w_3 = w_3 - a\,(i_1\,.\,\Delta w_6)$$

$$^{*}w_2 = w_2 - a\,(i_2\,.\,\Delta w_5)$$

$$^{*}w_1 = w_1 - a\,(i_1\,.\,\Delta w_5)$$

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a\,\Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} a h_1 \Delta \\ a h_2 \Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a\,\Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} . \begin{bmatrix} w_5 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a\, i_1 \Delta w_5 & a\, i_1 \Delta w_6 \\ a\, i_2 \Delta w_5 & a\, i_2 \Delta w_6 \end{bmatrix}$$

Backpropagation
Backward Pass

$\Delta = 0.191 - 1 = -0.809$ ← **Delta** = prediction - actual

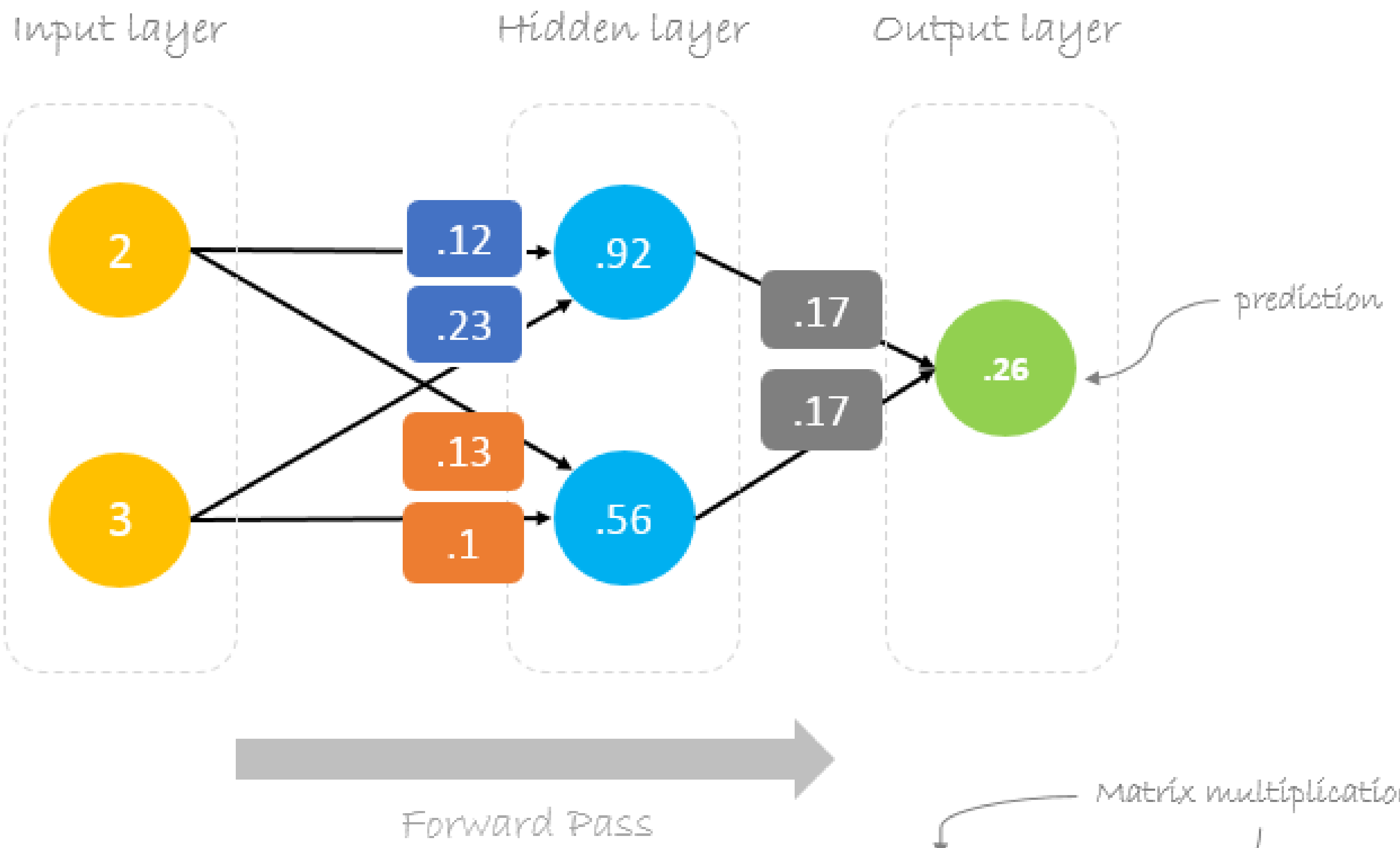
$a = 0.05$ ← **Learning rate,** we smartly guess this number

$$^{*}w_6 = w_6 - a\,(h_2\,.\,\Delta)$$

$$^{*}w_5 = w_5 - a\,(h_1\,.\,\Delta)$$

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 2 \\ 3 \end{bmatrix}.\begin{bmatrix} 0.14 & 0.15 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$



Matrix multiplication

$$[2 \quad 3] \cdot \begin{bmatrix} 0.12 & 0.13 \\ 0.23 & 0.10 \end{bmatrix} = [0.92 \quad 0.56] \cdot \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix} = [0.26]$$

Details

2 x .12 + 3 x .23 = .85

2 x .13 + 3 x .10 = .48

.92 x .17 + .56 x .17 = .26

Backpropagation

We can notice that the **prediction** 0.26 is a little bit closer to **actual output** than the previously predicted one 0.191. We can repeat the same process of backward and forward pass until **error** is close or equal to zero.