"Machine learning is the last invention that humanity will ever need to make"

Deep Learning - 1

Abhinav Gupta

May 24, 2021

Robotics Research Center
IIIT Hyderabad

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Robotics Research Center

Today's Lecture

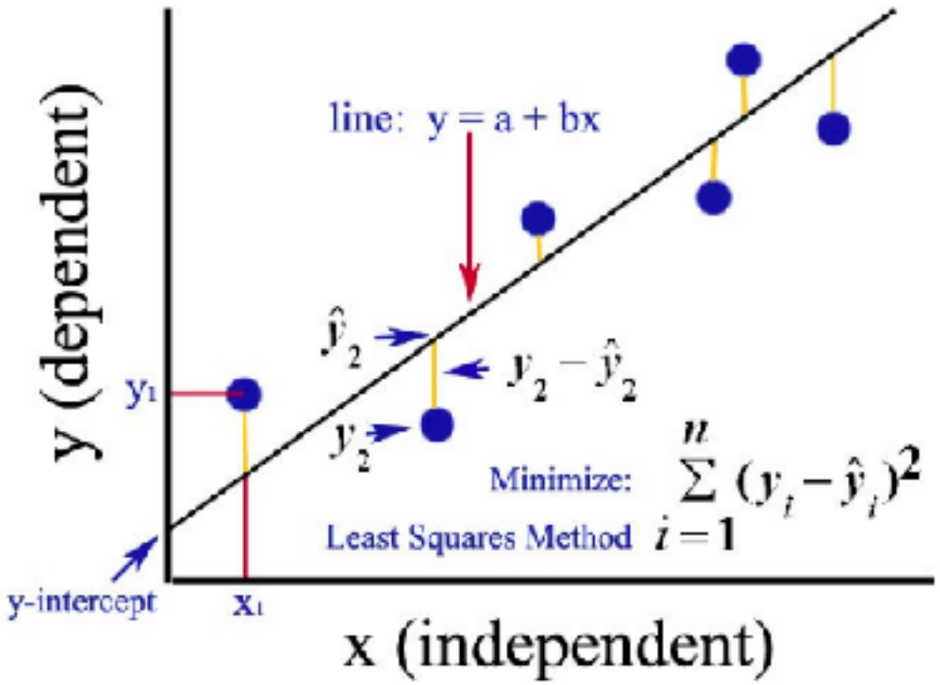
- Logistic Regression
- Gradient Descent
- Neural Networks

Robotics Research Center

So far…

- We've looked at **least squares** and **linear regression.**

- We've also seen what a gradient is and its significance in finding the optimum value.



$$\nabla f(x, y) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \cdots \\ \cdots \\ \cdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

Robotics Research Center

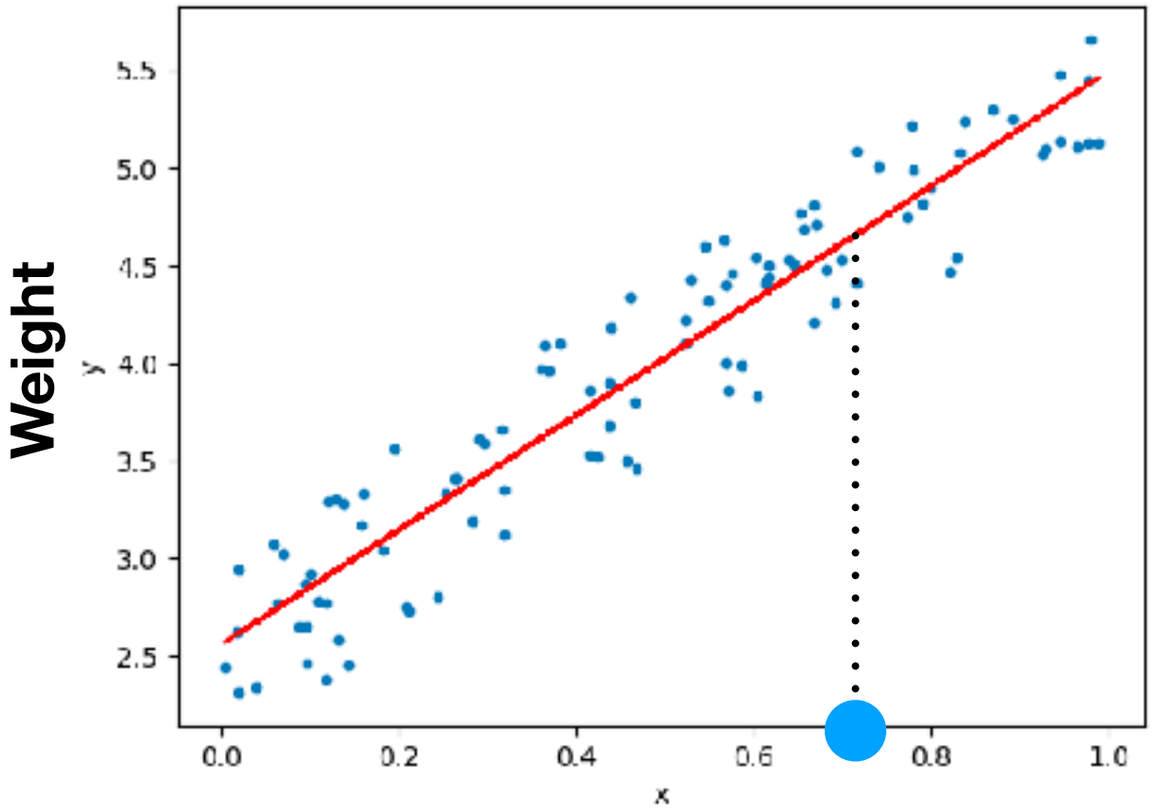
Linear Regression

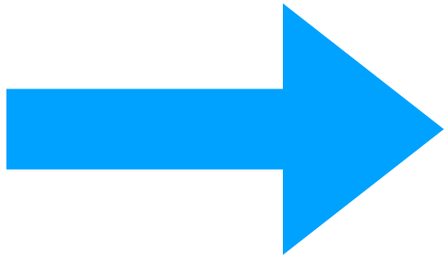
- In linear regression, we find a line (y=mx+c) that minimises the sum of the squares of the residuals. This can help us a lot for:

1. Determining whether height and weight are correlated by computing R^2

2. Calculate p-value to see if the correlation we get is statistically significant.

3. **Use the line to predict the weight, given height.**





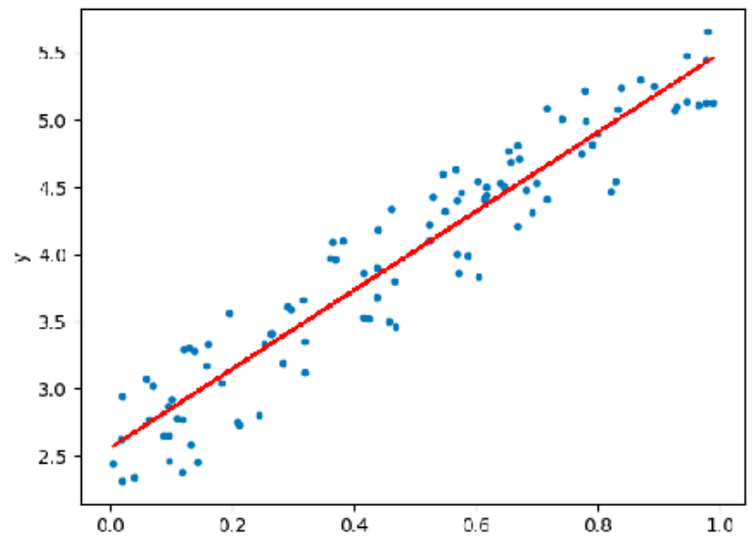
Machine Learning

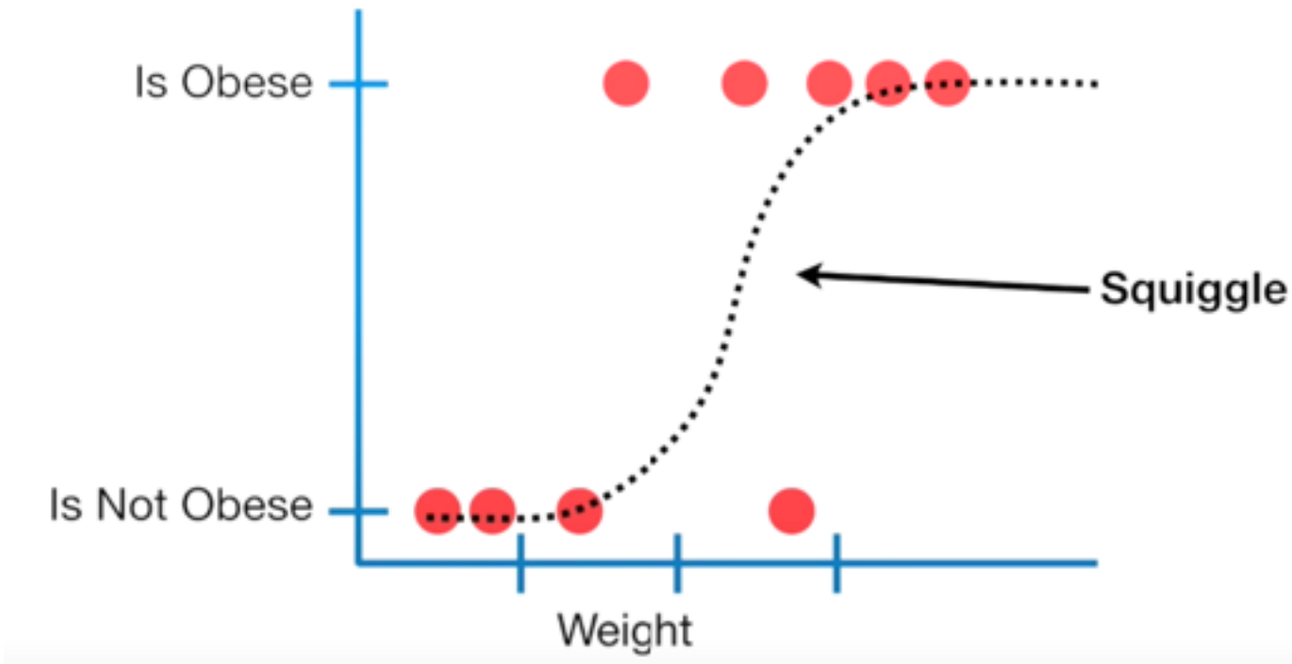
Robotics Research Center

Introduction

- In linear regression, we were optimising the intercept (c) and the slope (m) of the line that best fitted our data.

- In logistic regression, we optimise a **squiggle.** It is used commonly in both, traditional statistics and machine learning.





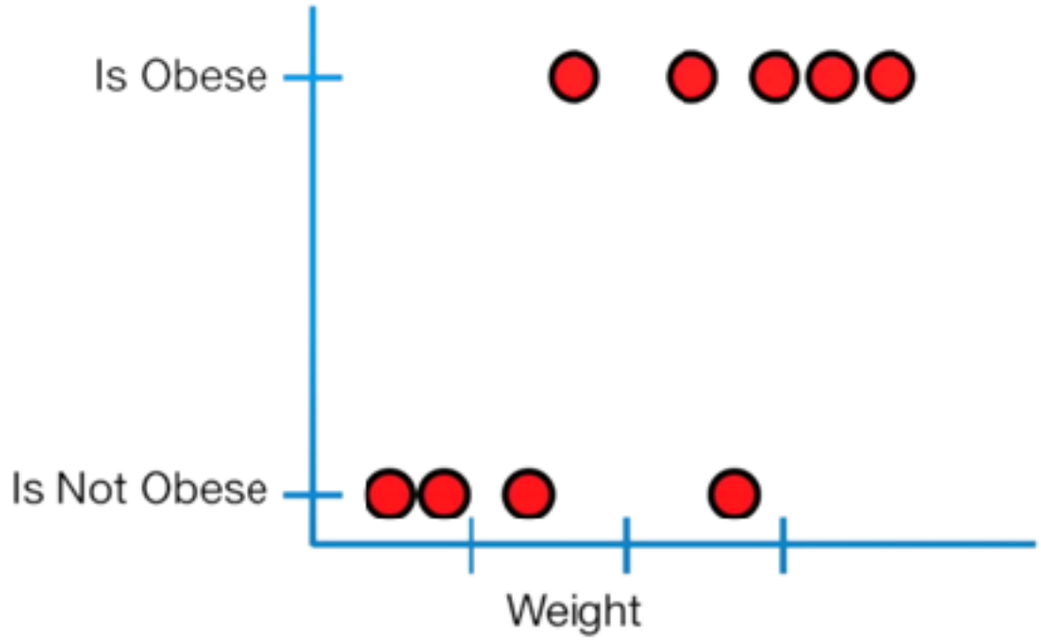
Robotics Research Center

Logistic Regression

- Logistic regression predicts if something is true or false. It helps us predict the probability of a human being, say obese or not obese.

- In this example, we have data for 9 human beings - 5 are obese and 4 aren't. We plot these points on the graph. The Y axis is just a categorical variable with two choices - obese or not obese.

- Now we wish to fit a line or a curve that can help in prediction for machine learning.



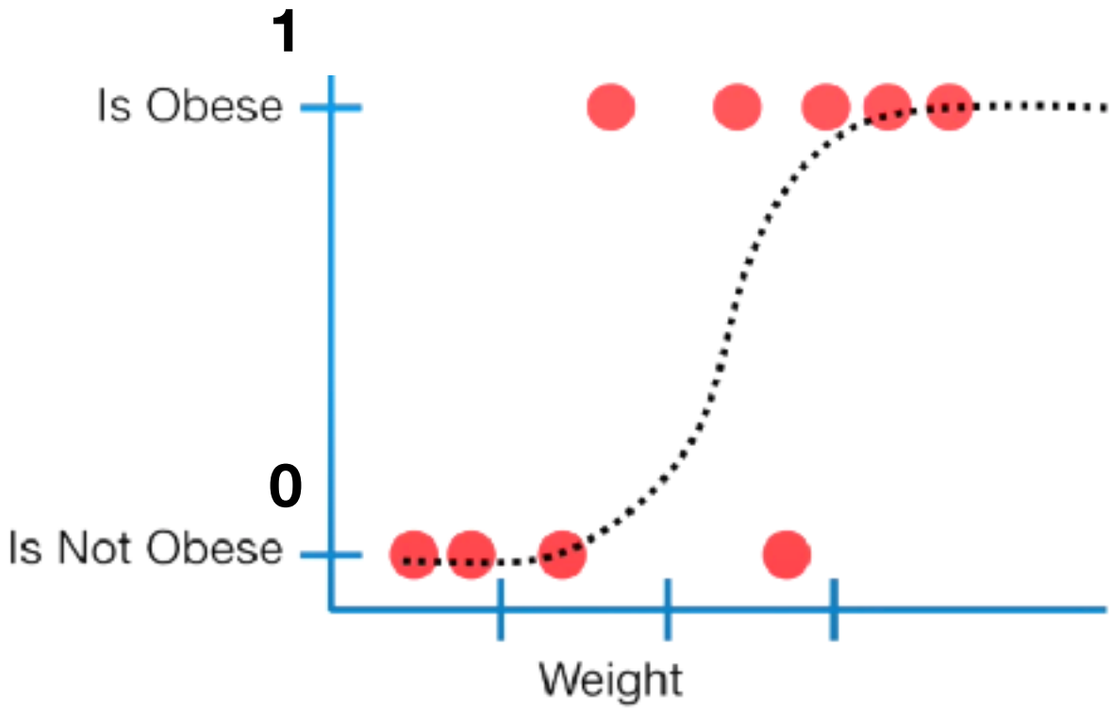
Logistic Regression

- In linear regression we were fitting a line that would solve the least squares problem and help best resemble the data.

- Here, in logistic regression, we fit a logistic function. In this example, it's an "S" shaped logistic function.

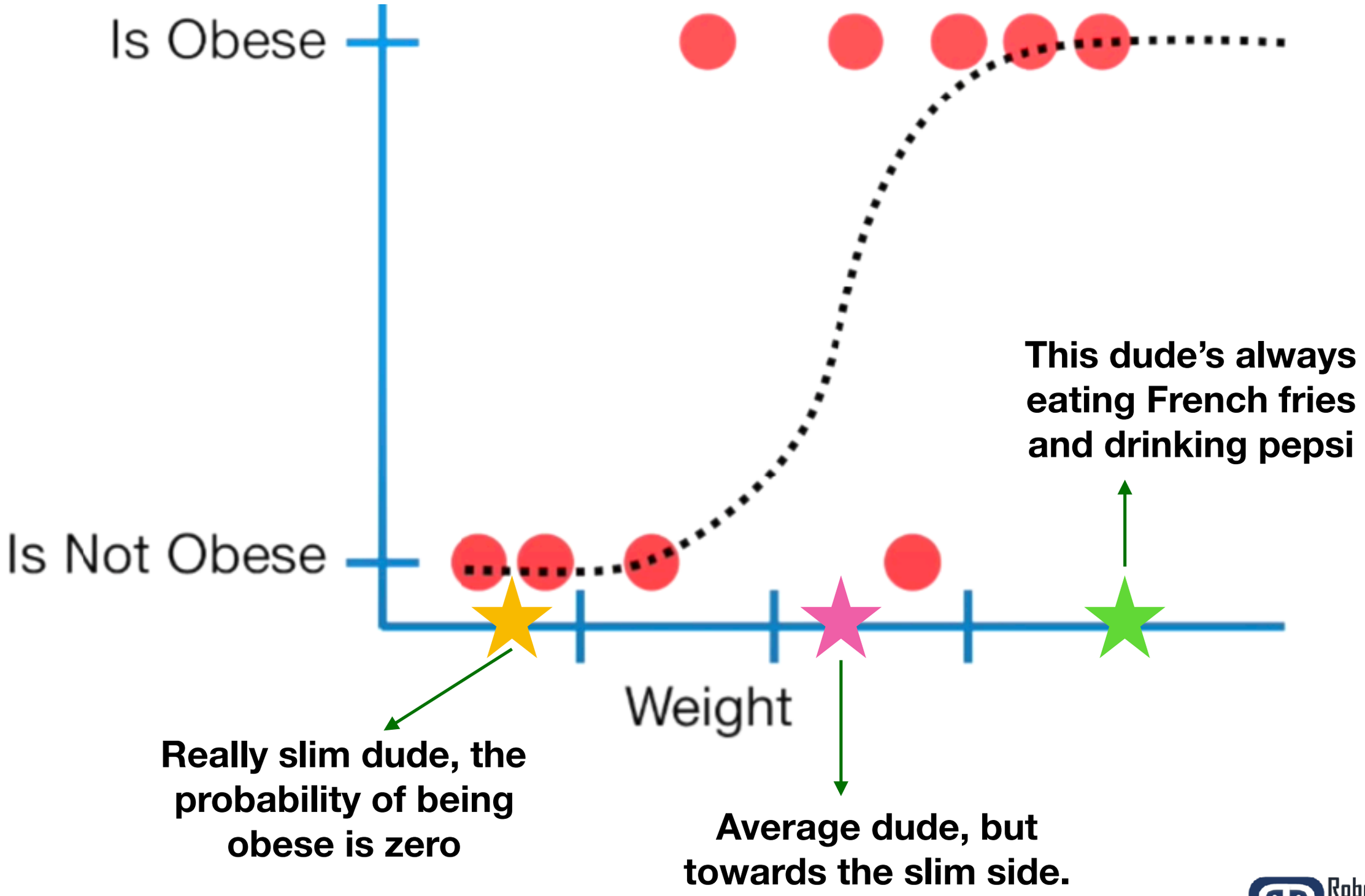
- The X axis is a continuous variable - weight in this case.

- The Y variable now is a categorical one - **the curve goes from 0 to 1** to show the probability.



The curve tells us the probability that a human being is obese, given his/her weight!

Robotics Research Center



Robotics Research Center

Logistic Regression

- Hence, as you would've guessed, logistic regression can easily be used for classification, and is indeed popularly used in the ML world.

- Given the data, we fit a logistic function curve to the date, which helps us infer the probability of the person being obese, like in our example.

- So we can set a threshold, like if P>50%, then human is obese, else he is not.

Linear vs Logistic Regression

- The most crucial difference between these two forms of regressions is how the line/curve is fit to the data.

- In linear regression, the line is fit using least squares. We try to find the most optimal value of the slope and the intercept (y=mx+c) that minimises the sum of the squares of the residuals. (least squares problem)

- In logistic regression, we use something called the **maximum likelihood** that helps us find the best logistic function curve to fit (that “S” like curve).

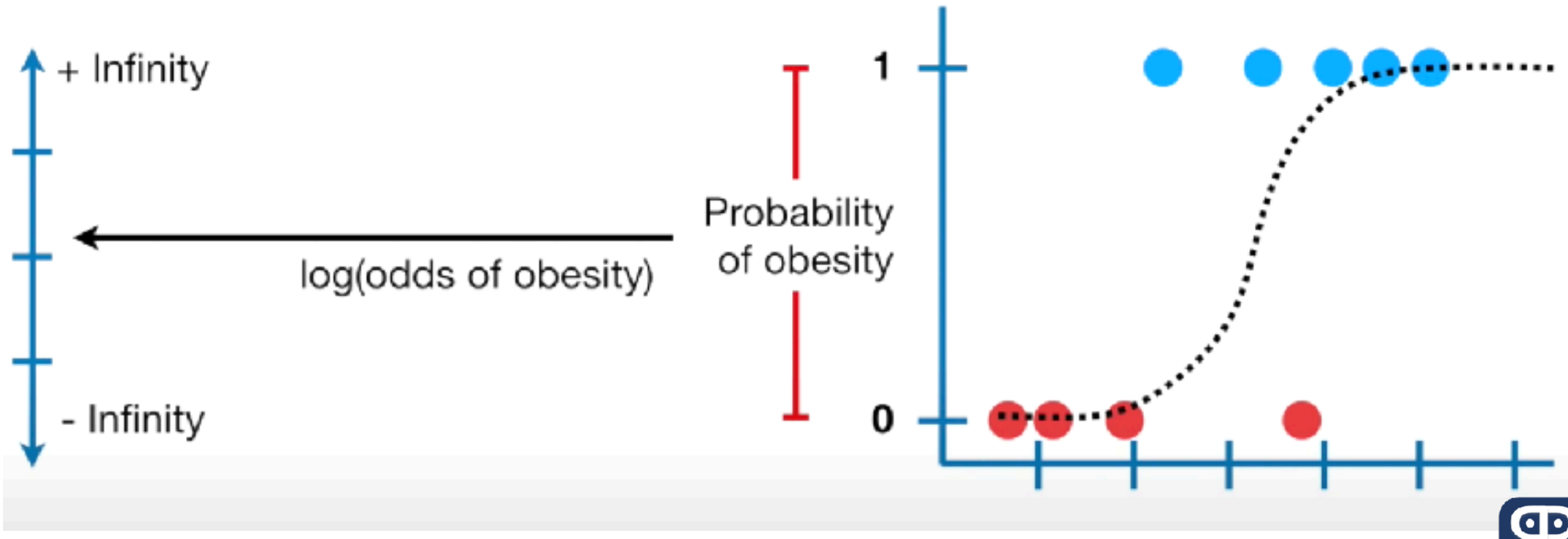
- And that’s what we are going to see next!

Robotics Research Center

Logistic Regression

- Let’s talk about how we can fit a line using maximum likelihood - how to optimise the squiggle to fit the data as well as possible.

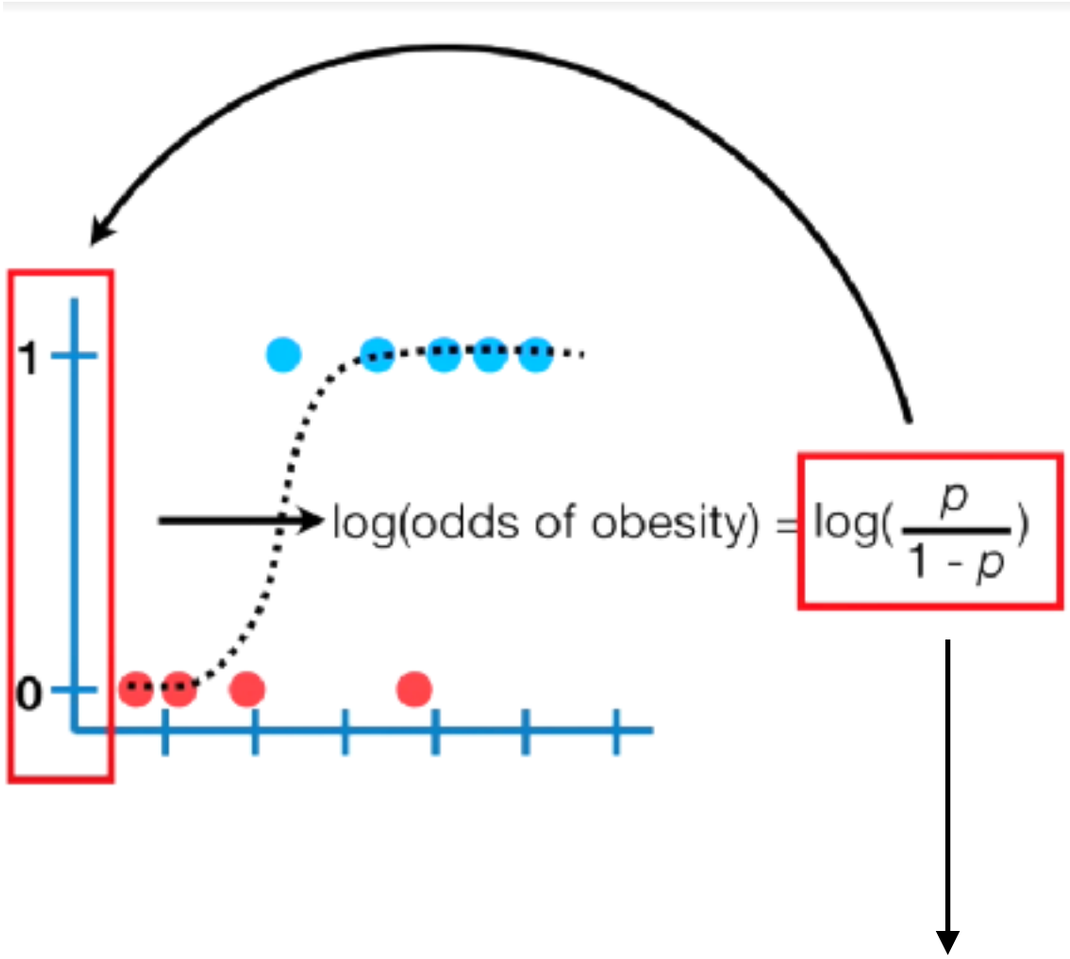
- The Y axis is transformed from the *probability of obesity* to *log(odds of obesity)*, thereby changing the upper and lower limits on the Y axis from -infinity to +infinity.

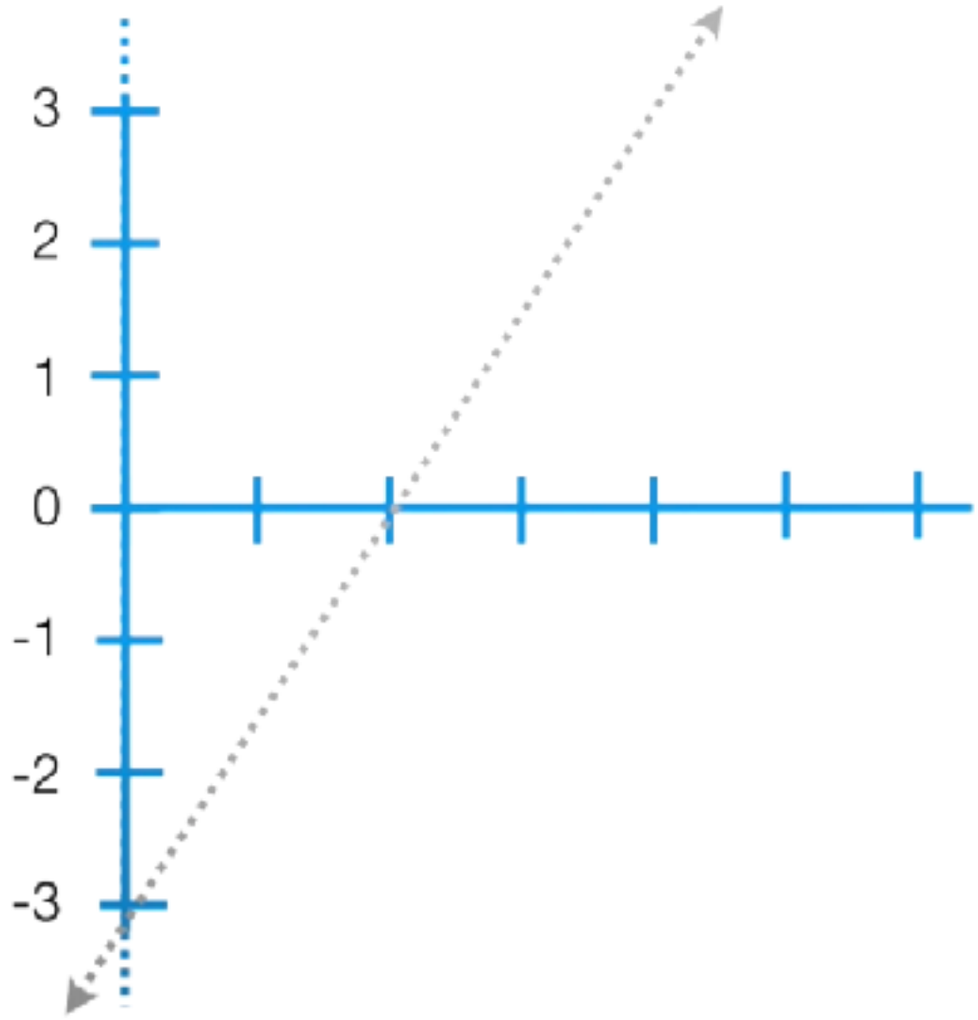


Robotics Research Center

Logistic Regression

This is the best fitting line!





The Logit Function helps convert the probability to log(odds)

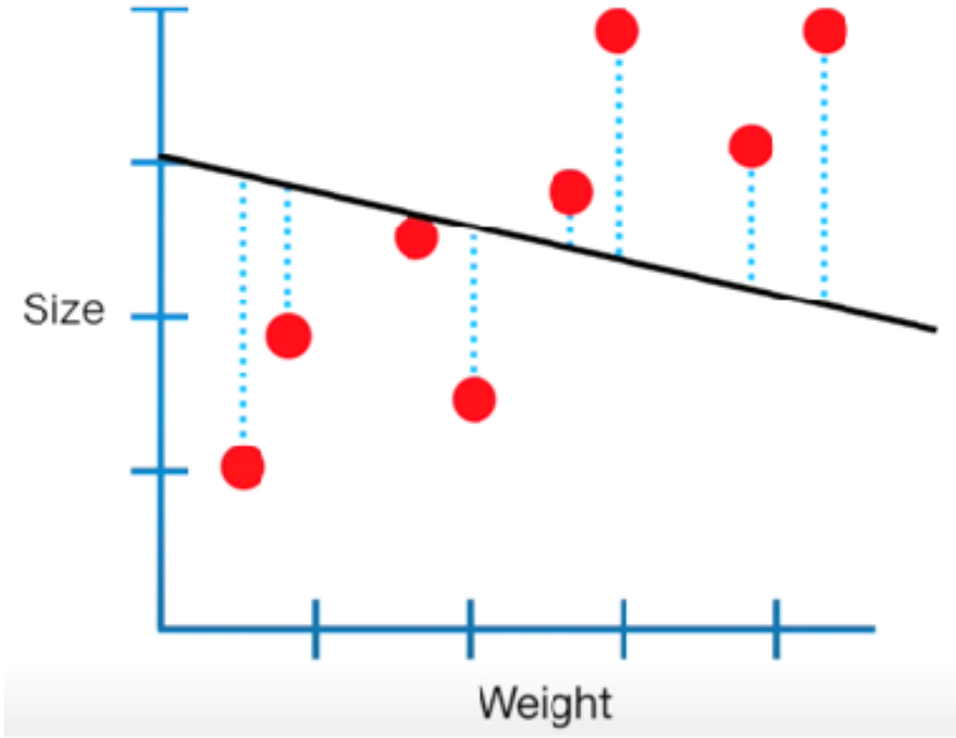
Robotics Research Center

Logistic Regression

- In linear regression, we fit the line using least squares. We measure the *residuals* - the distance between data points and our line of fitting.

- We square these residuals (so negative ones don't cancel out positive ones) - and then add them up - sum of least squares.

- We rotate the line a little bit and do the same thing. The one with the smallest sum of residuals aka least squares is the line chosen for best fit.



Robotics Research Center

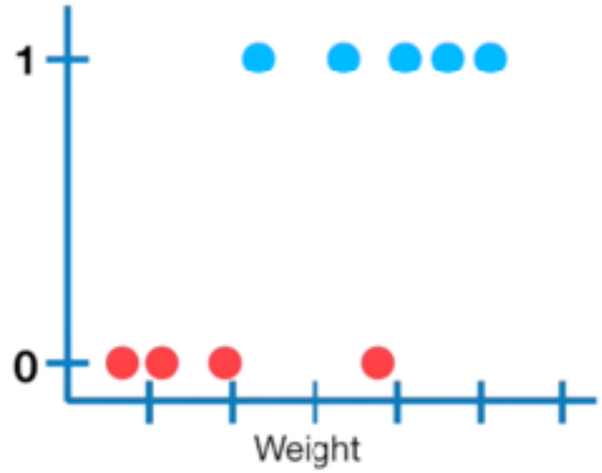
Logistic Regression

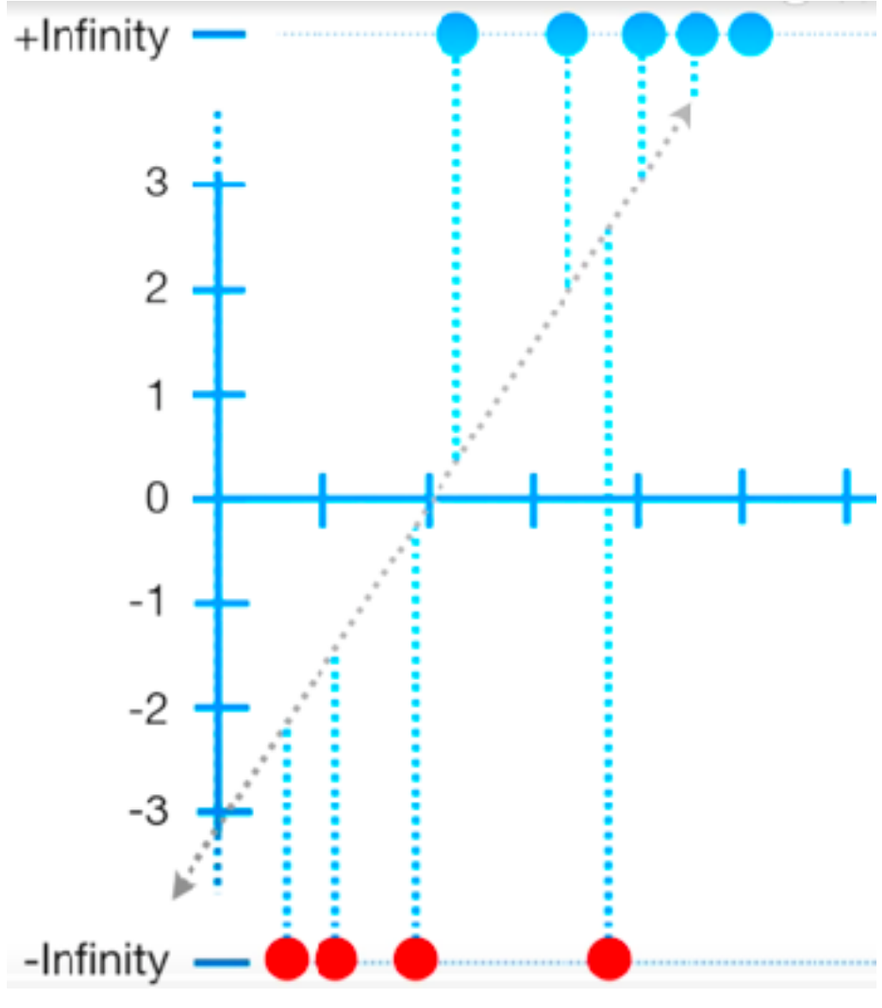
- Our goal is to draw the best fitting “squiggle” to fit this data.

- We transform the Y axis to the log(odds of probability) to increase the limits on the Y axis.

- But this pushes our existing data points to -infinity and +infinity - giving us infinite residuals!

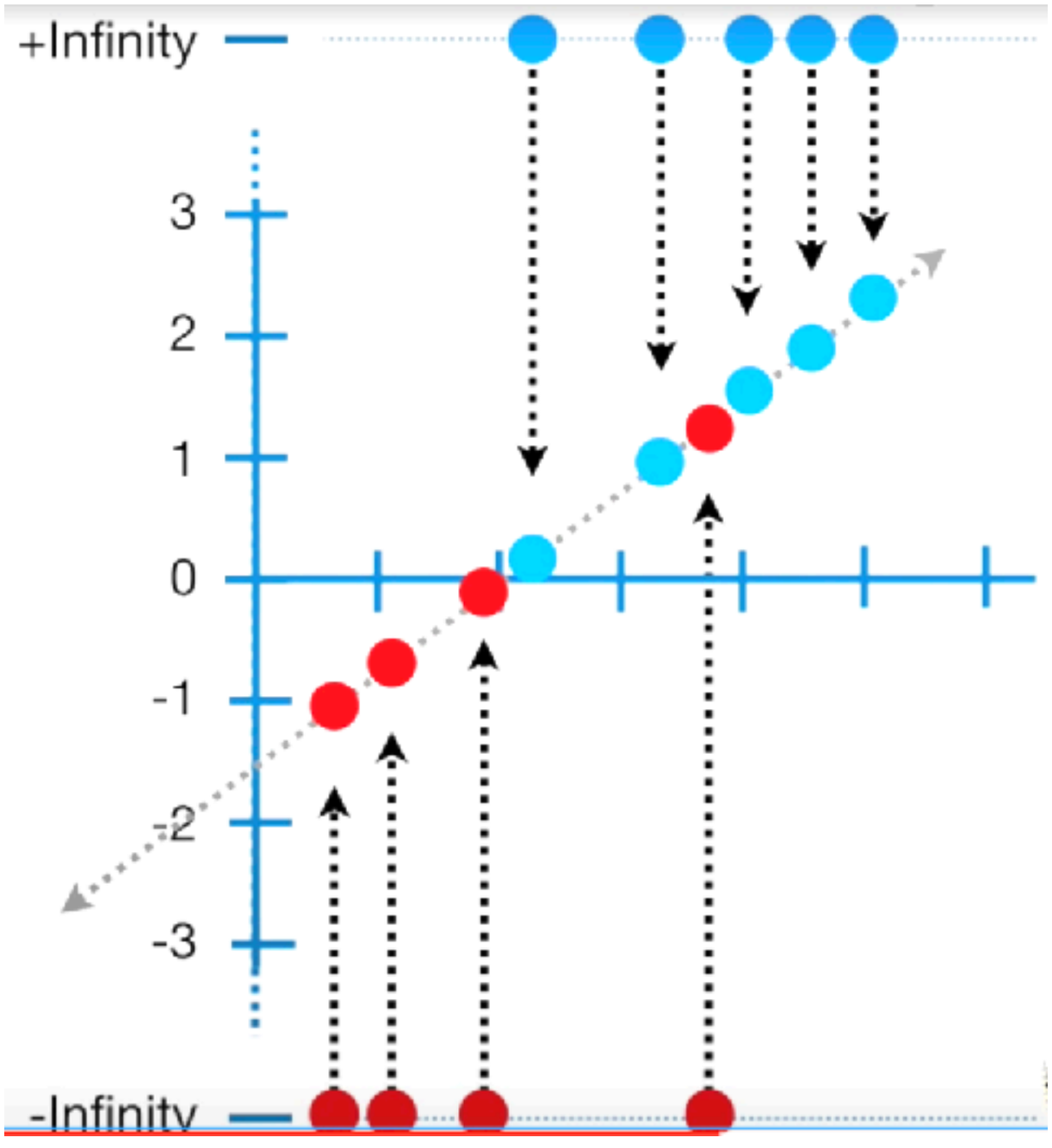
- We can’t use least squares - so we use maximum likelihood.





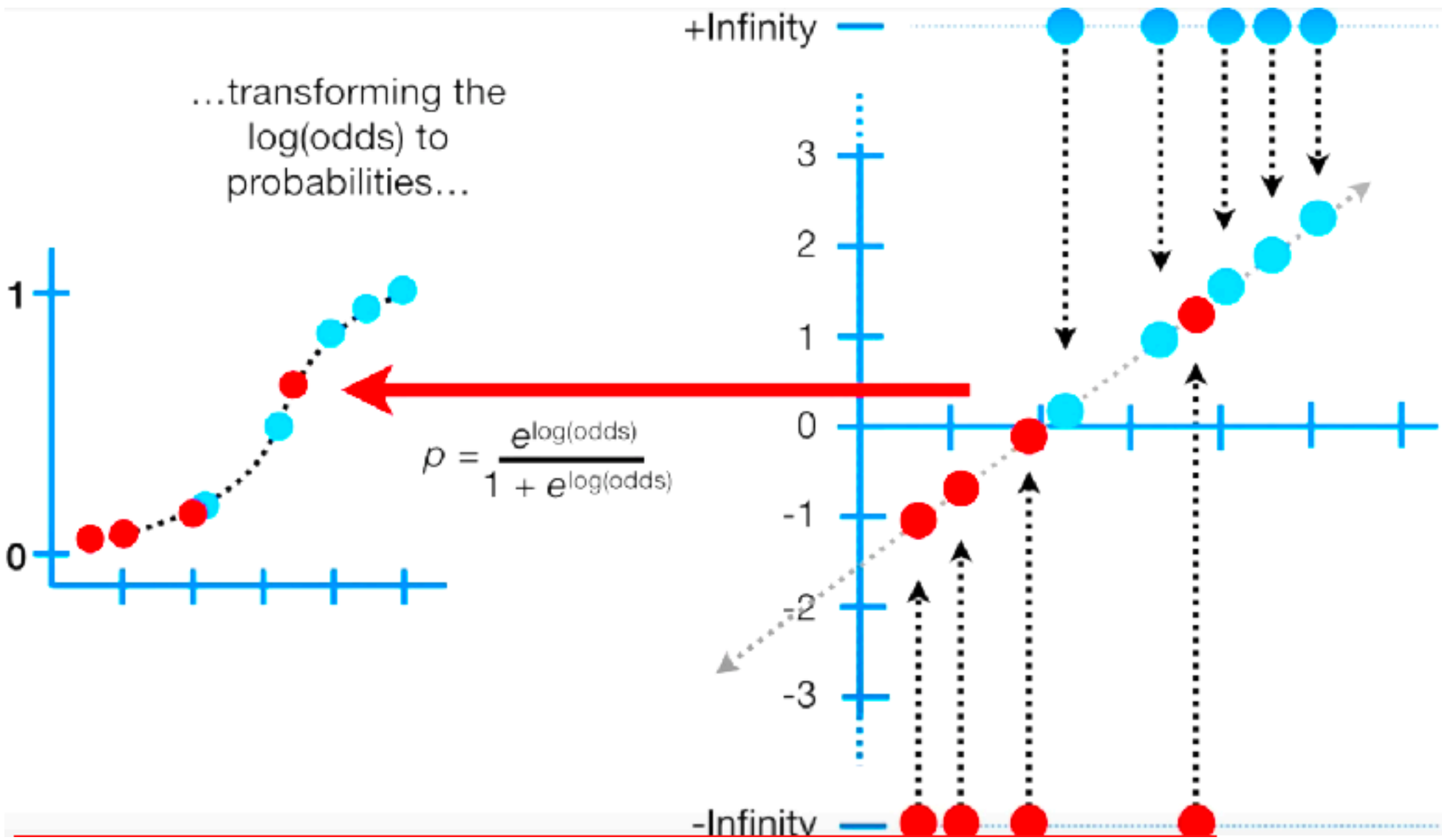
Robotics Research Center

Step 1: Project all existing data points onto the line and compute the log-likelihood.



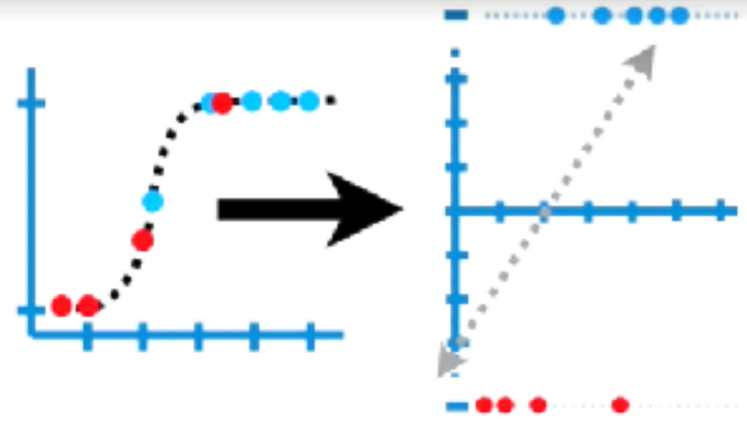
Robotics Research Center

Step 2: We transfer these log(odds) to probabilities as shown in the figure.



Robotics Research Center

$$\log\left(\frac{p}{1-p}\right) = \log(\text{odds})$$



Exponentiate both sides…

$$\frac{p}{1-p} = e^{\log(\text{odds})}$$

Multiply both sides by $(1 - p)$…

$$p = (1-p)e^{\log(\text{odds})}$$

Multiply $(1 - p)$ and $e^{\log(\text{odds})}$…

$$p = e^{\log(\text{odds})} - pe^{\log(\text{odds})}$$

Add $pe^{\log(\text{odds})}$ to both sides…

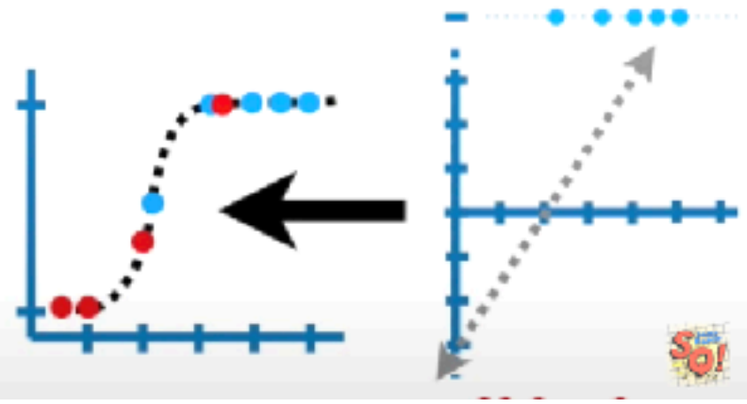
$$p + pe^{\log(\text{odds})} = e^{\log(\text{odds})}$$

Pull p out…

$$p(1 + e^{\log(\text{odds})}) = e^{\log(\text{odds})}$$

Divide both sides by $(1 + e^{\log(\text{odds})})$…

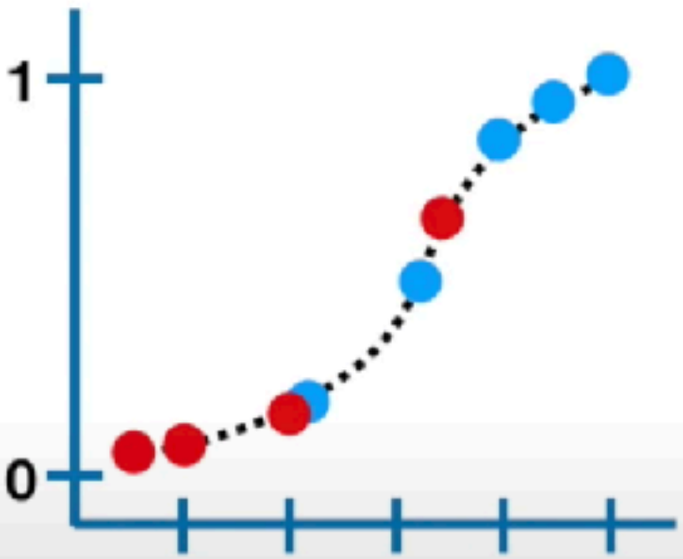
$$p = \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}$$



Robotics Research Center

Step 3: Compute log-likelihood. Blue ones are for obese people, red ones for not obese.

log(likelihood of data given the squiggle) = log(**0.22**) + log(**0.4**) + log(**0.8**) + log(**0.89**) +
log(**0.92**) + log(**1 - 0.6**) + log(**1 - 0.2**) +
log(**1 - 0.1**) + log(**1 - 0.05**)



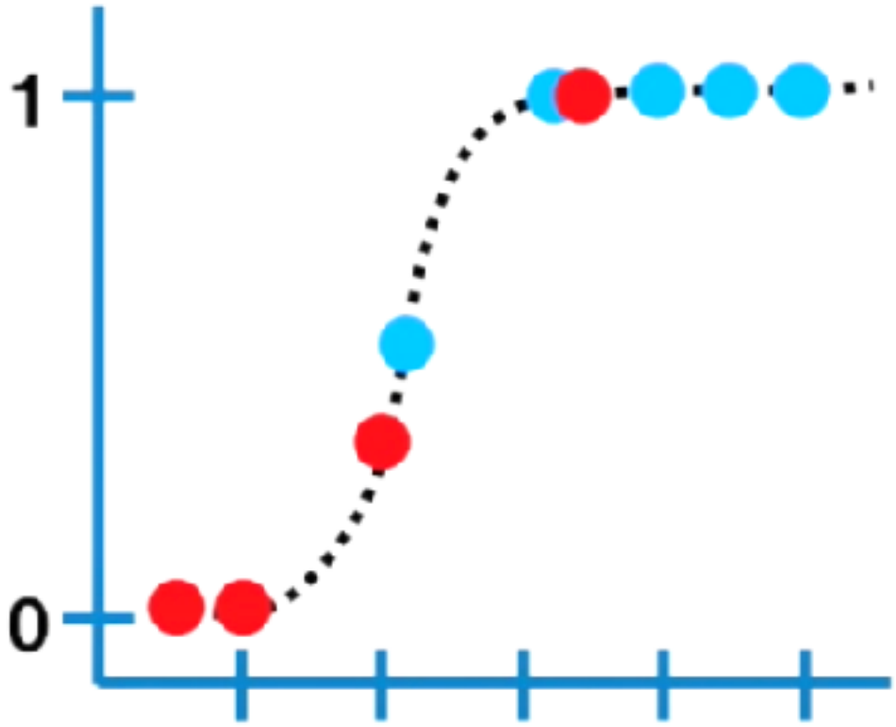
…and then calculating the log-likelihood…

Logistic Regression

- The likelihood a given person is obese is the Y-axis value corresponding to that point!

- Probability is not computed under the curve - rather, it's the Y-axis value!

- The likelihood for all obese folks is the produce of the individual likelihoods.



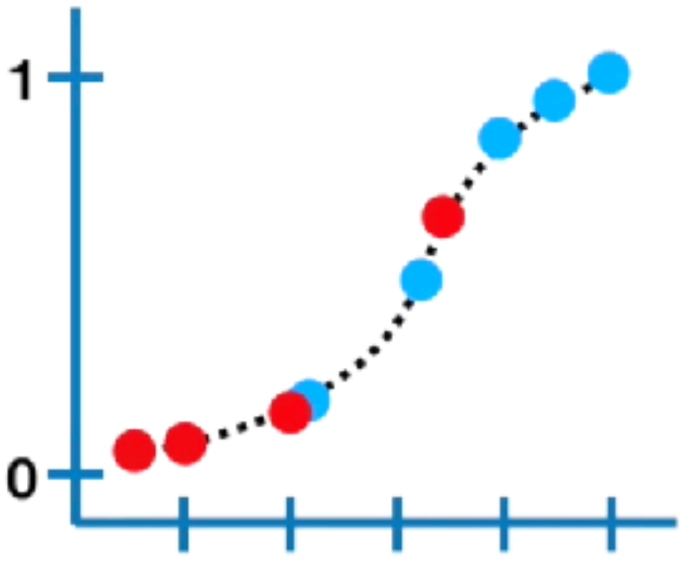
likelihood of data given the squiggle = $0.49 \times 0.9 \times 0.91 \times 0.91 \times 0.92 \times (1 - 0.9) \times (1 - 0.3) \times (1 - 0.01) \times (1 - 0.01)$

Robotics Research Center

Logistic Regression

- Rather, we take the log of the likelihood!
- Either way is okay, since the squiggle that maximises the likelihood is the same as the squiggle that maximises the log of the likelihood!

log(likelihood of data given the squiggle) = log(0.22) + log(0.4) + log(0.8) + log(0.89) + log(0.92) + log(1 - 0.6) + log(1 - 0.2) + log(1 - 0.1) + log(1 - 0.05)



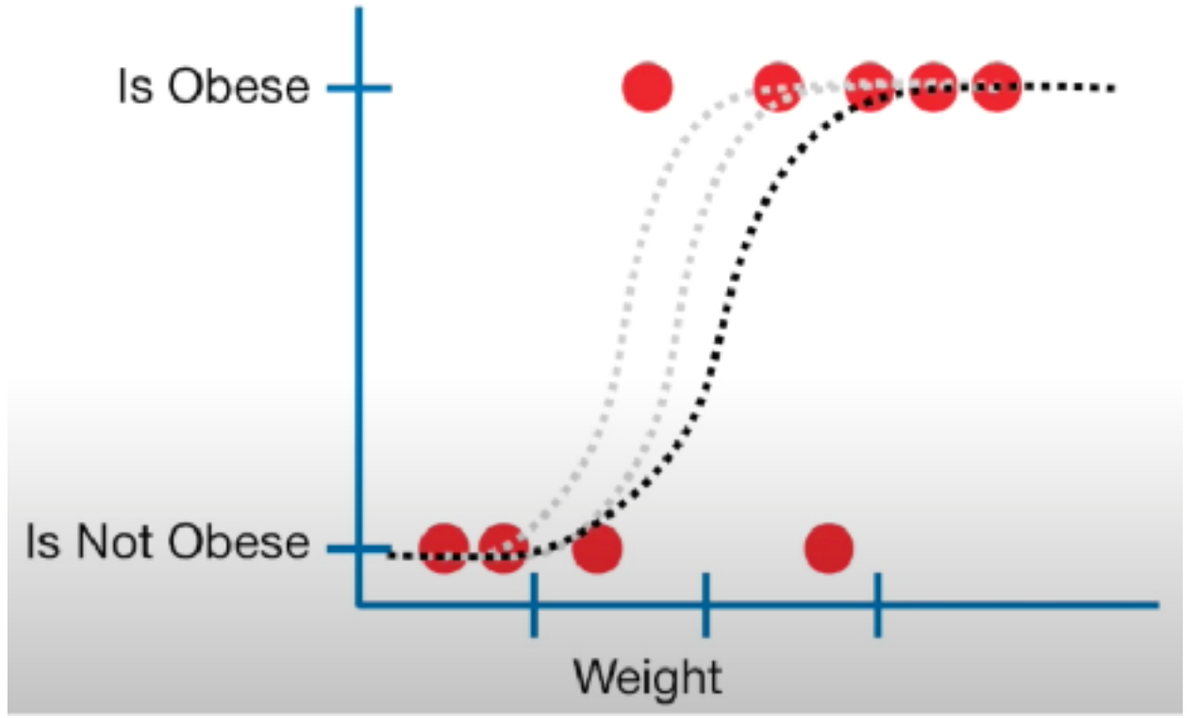
…and then calculating the log-likelihood…

Logistic Regression

- We keep rotating the log(odds) line and keep projecting the data onto it.

- We keep transforming it to probabilities and computing the log-likelihood.

- Ultimately we get a line that maximises the likelihood and that’s the one chosen for the best fit!



Robotics Research Center

Today's Lecture

- Logistic Regression
- Gradient Descent
- Neural Networks

Robotics Research Center

Gradients

$$f(x, y) = x^2 + y^2$$

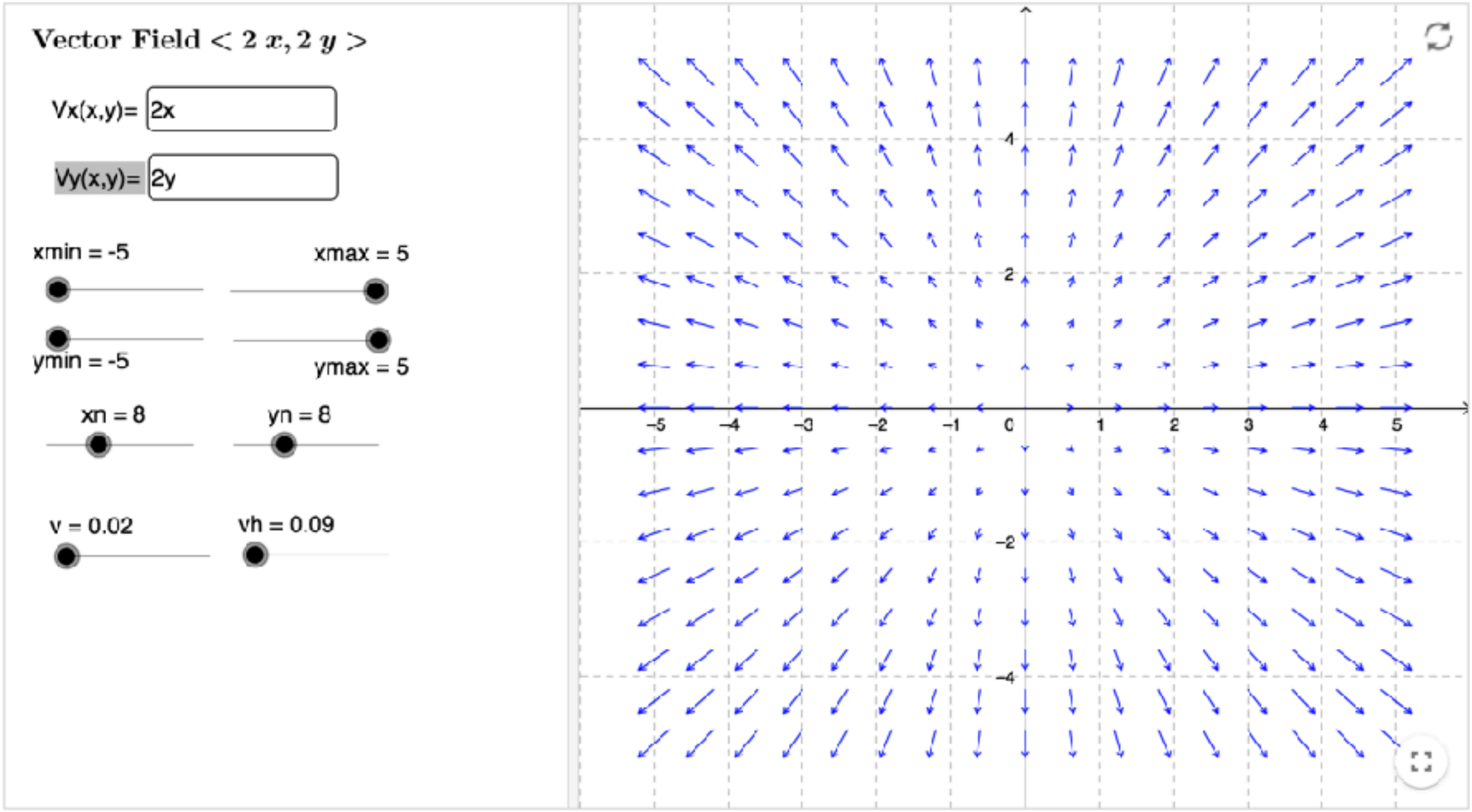
- Computing the gradients,

$$\nabla f(x, y) = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

$$\nabla f(x, y) = \begin{pmatrix} 2x \\ 2y \end{pmatrix}$$

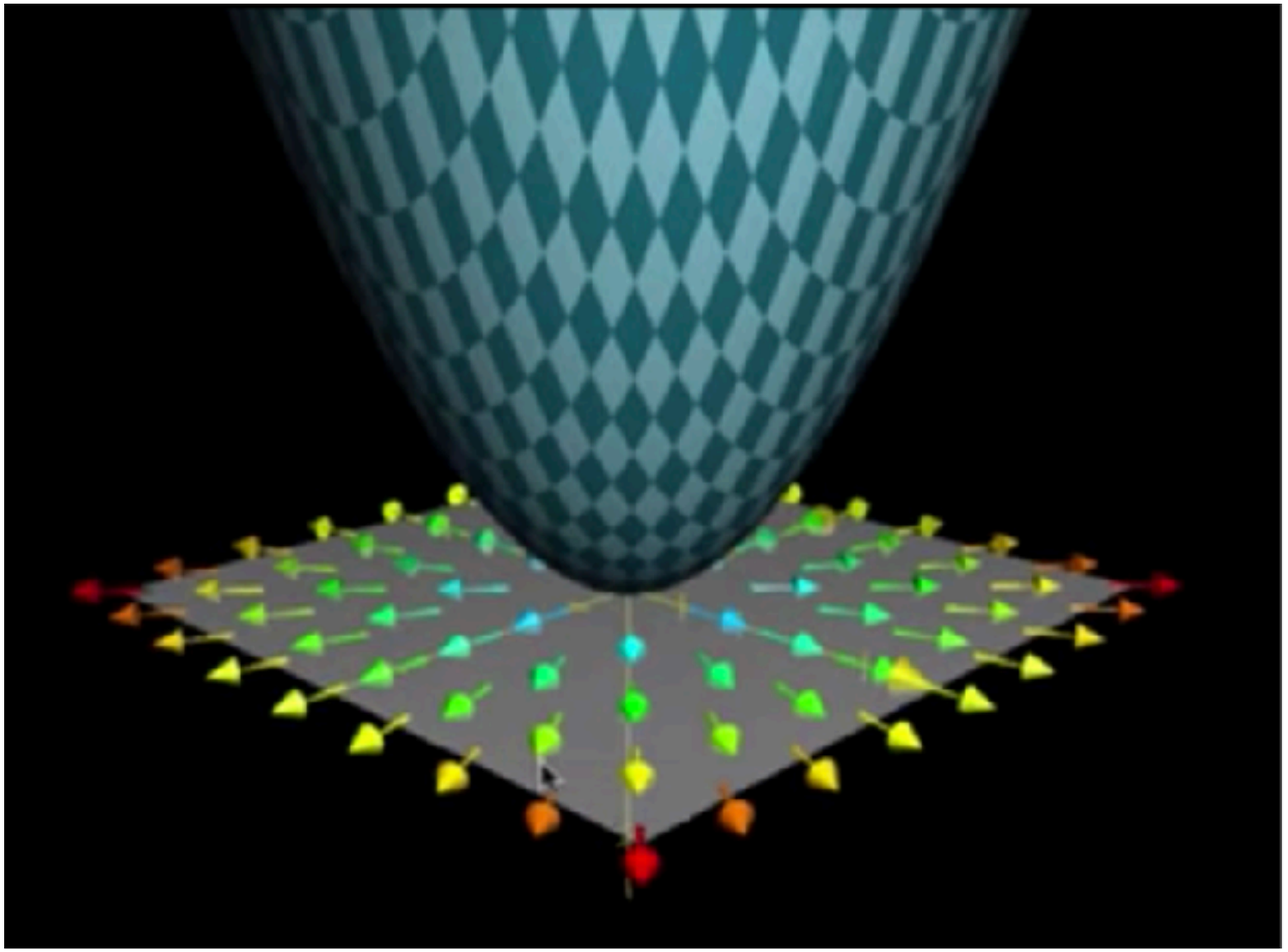
- Let’s check out its vector field.

Robotics Research Center



All vectors are pointing ‘outwards’ - away from the origin!

Robotics Research Center



That “inverted mountain” is our function - x^2 + y^2
And the vector field is plotted below.
Different colours denote different magnitudes.

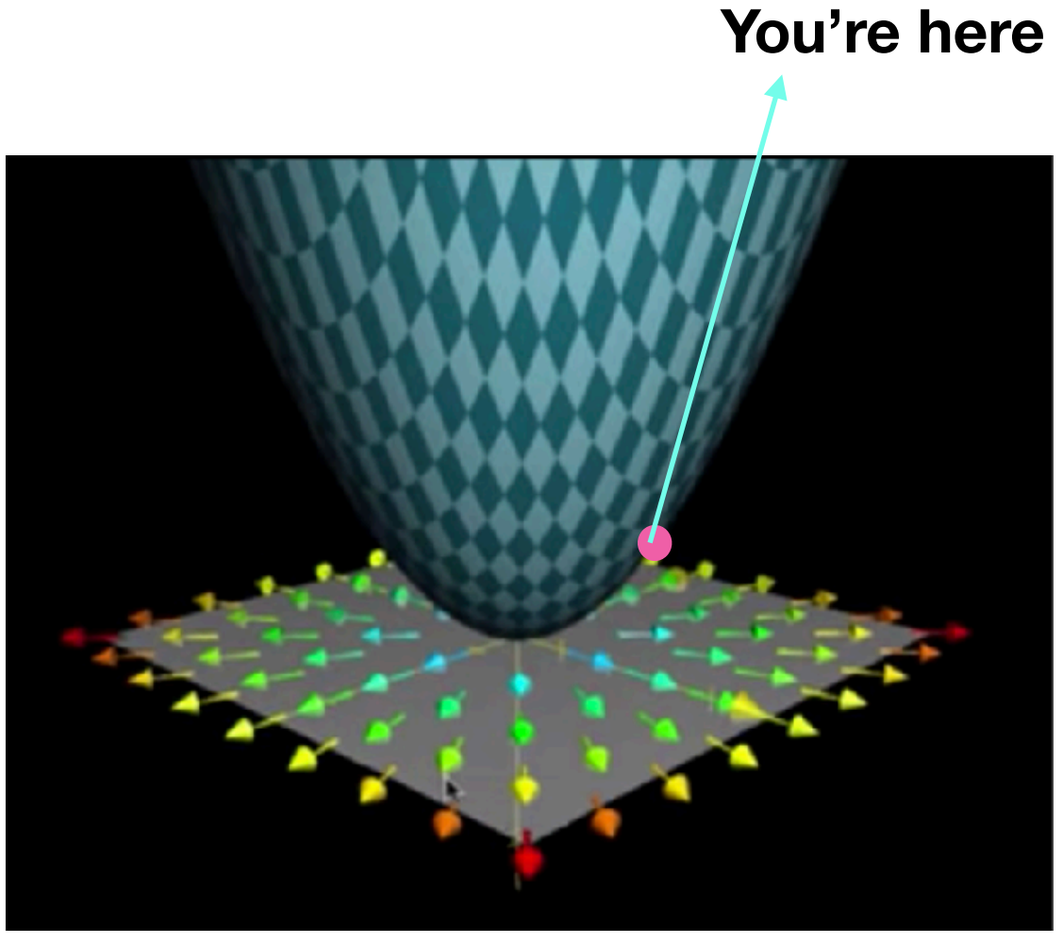
Robotics Research Center

Gradients

- The gradient of the function has been plotted as a vector field - the red arrows are the longer ones and the blue arrows are the shorter ones. They all point away from the origin.

- The function is also plotted - looks like an inverted mountain. Assume you're climbing this mountain, and you wish to reach the top of the mountain really quick.

- Which DIRECTION should I walk/ hike in, so as to reach the top as quickly as possible?

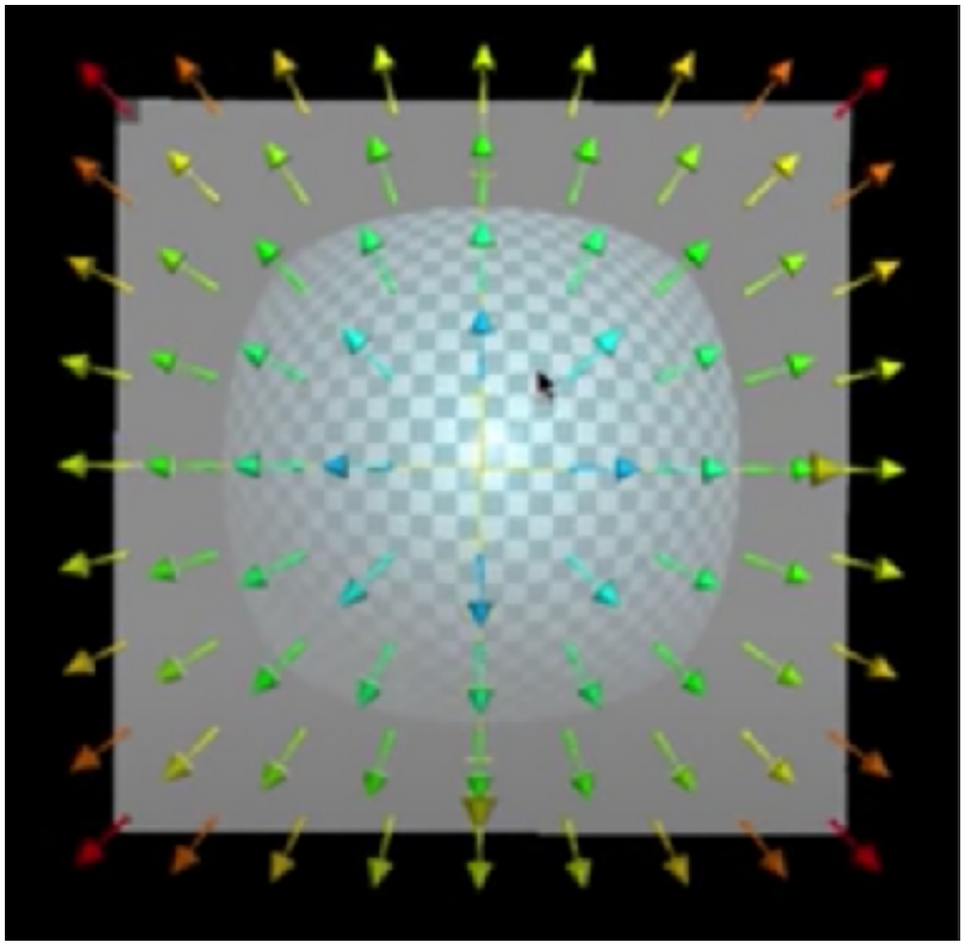


$$f(x, y) = x^2 + y^2$$

Robotics Research Center

Gradients

- Project our current position down to the vector field. In this case, it's evident that the vector which is going to get us quickly to the top of the mountain is one that points AWAY from the origin.



- **The gradient points in the direction of the steepest ascent!**

- Each vector is telling us which direction we should walk to quickly increase the altitude on the graph.

- The direction of the gradient vector tells us the steepness of that direction!

Robotics Research Center

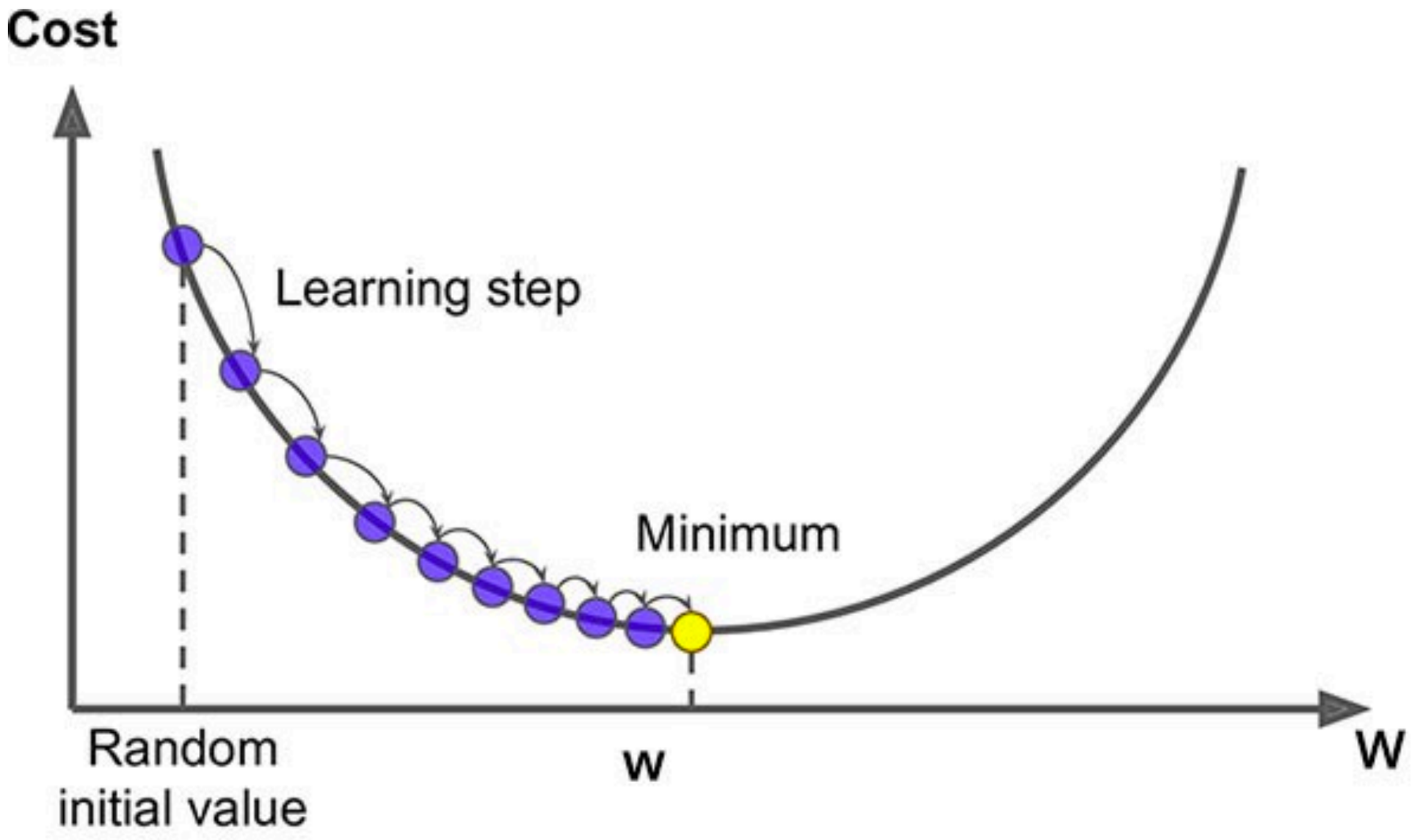
Gradient Descent

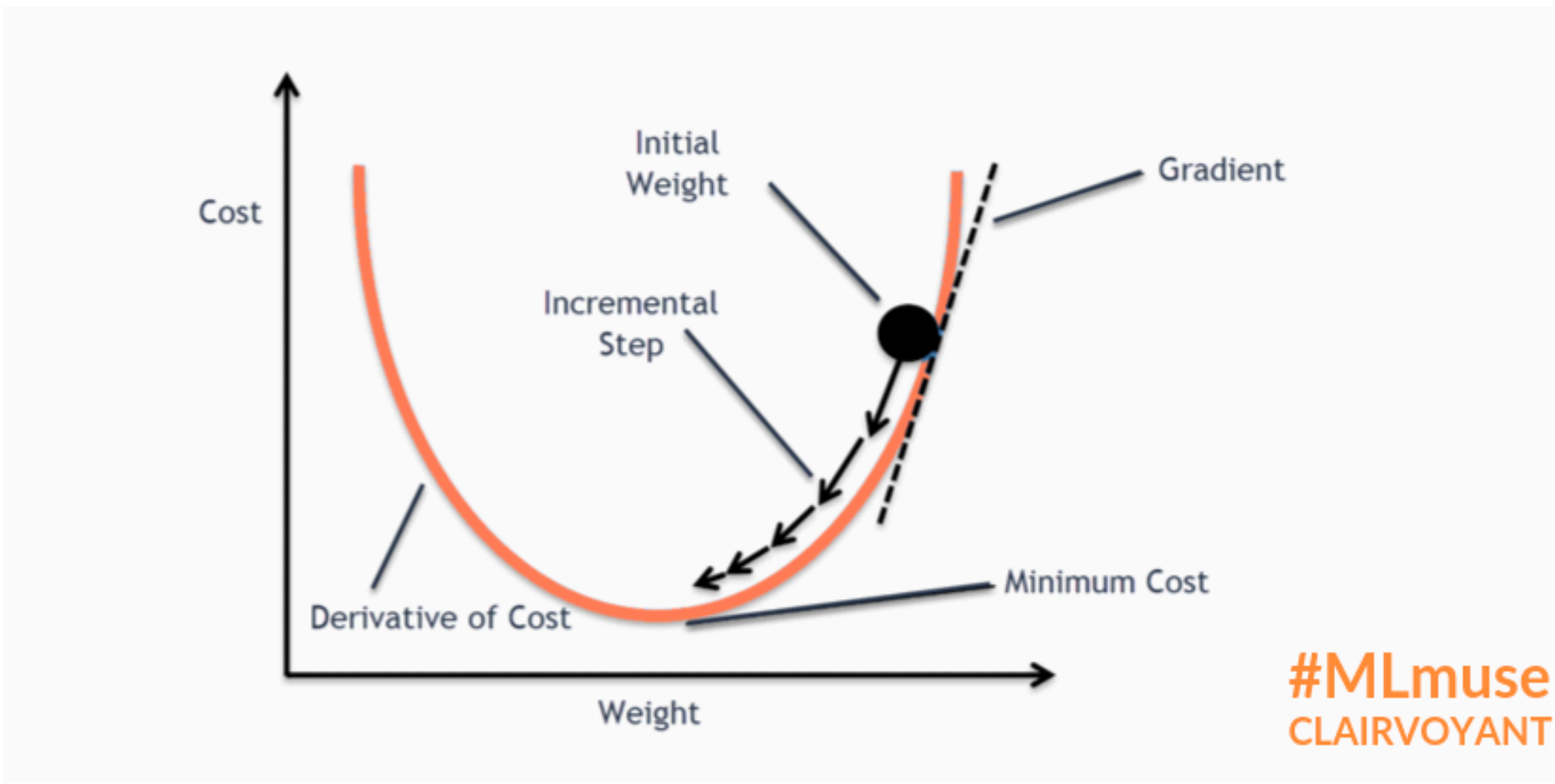
- Gradient Descent (GD) is an iterative optimisation algorithm for finding a local minimum of a differentiable function.

- The key idea is to take repeated steps in the OPPOSITE direction of the gradient of the function at the current point, as this is the direction of the **steepest descent.**

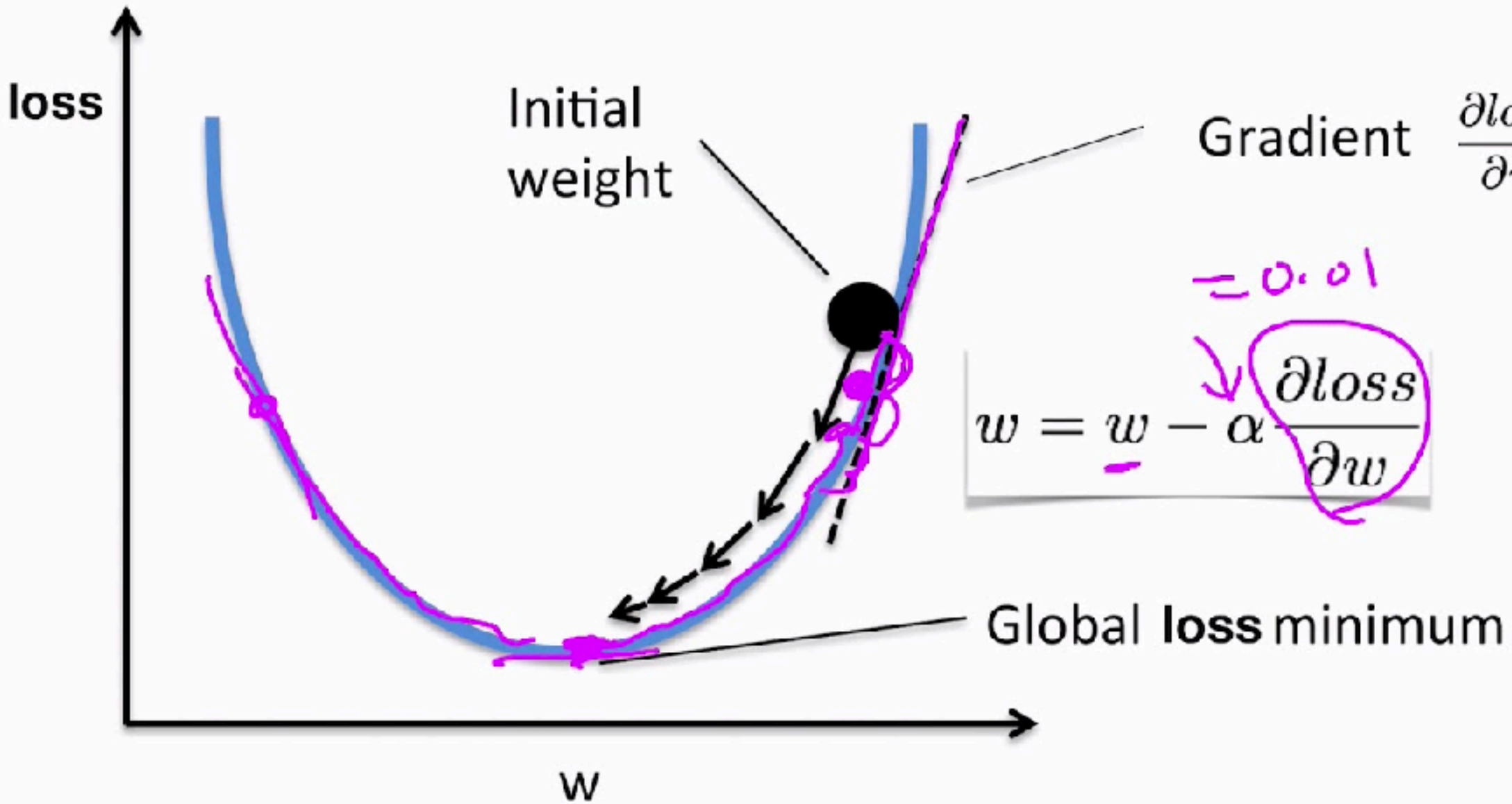
- The gradient points us in the direction of the **steepest ascent.** Stepping IN the direction of the gradient will lead to a local maximum - gradient ascent.

Robotics Research Center





Gradient descent algorithm



5 minute break!

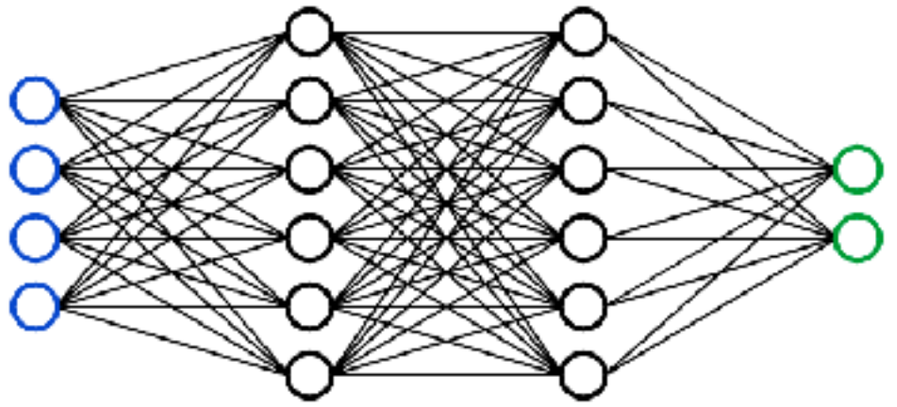
Robotics Research Center

Today's Lecture

- Logistic Regression
- Gradient Descent
- Neural Networks

Neural Networks

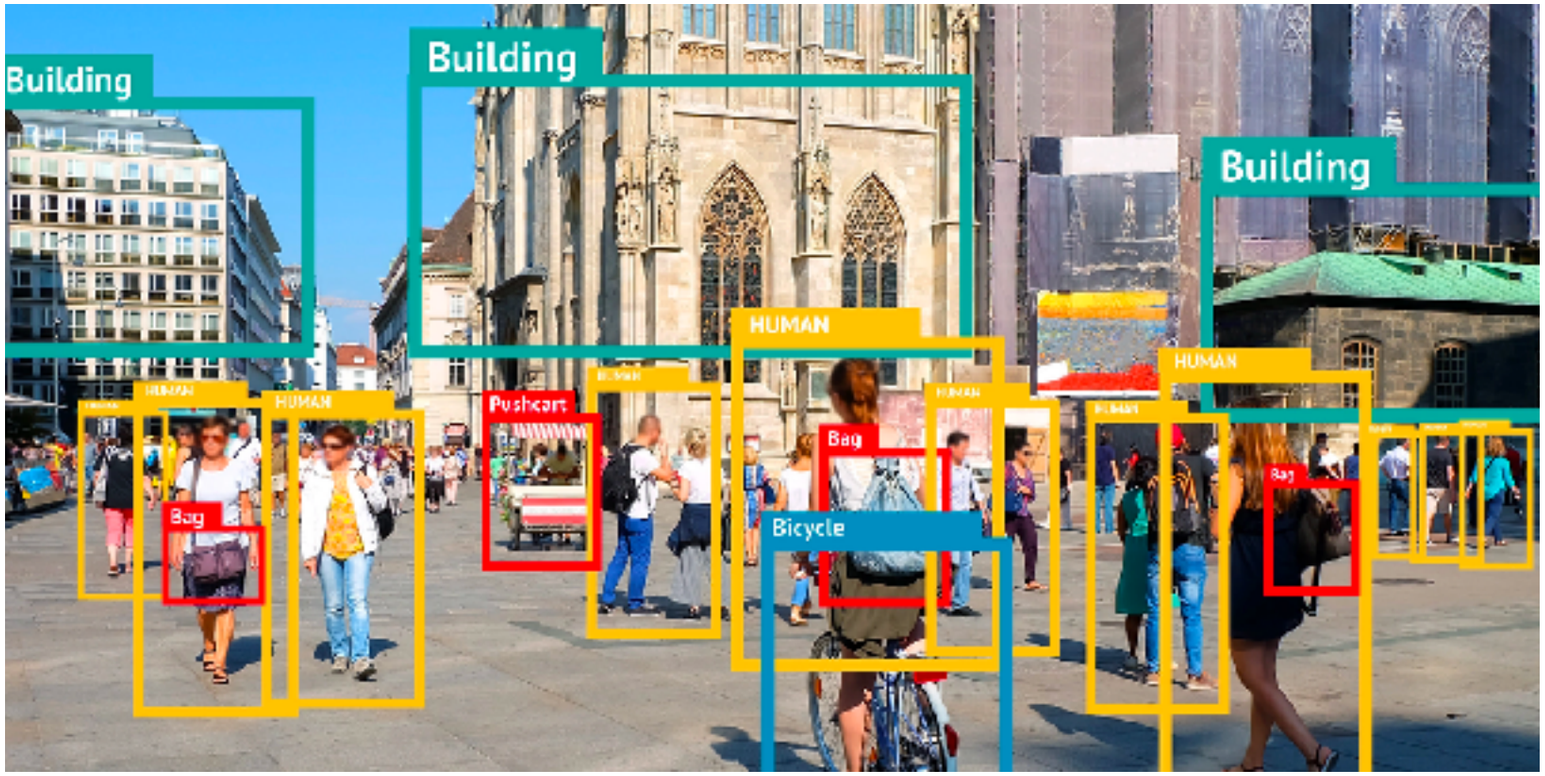
- *“The neural network is this kind of technology that is not an algorithm - it is a network that has weights on it, and you can adjust the weights so that it LEARNS. You teach it through trials.” - Howard Rheingold*

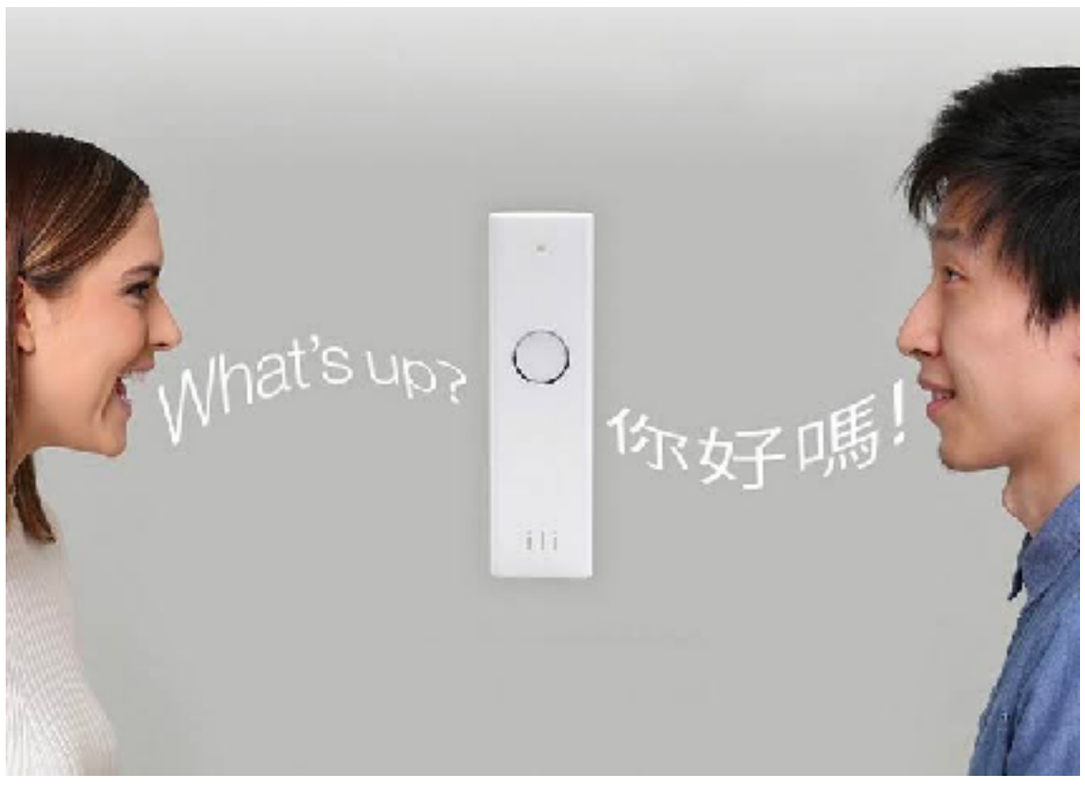


- Neural Networks are the shizz today, no kidding. They’ve taken over the world, with some amazing application in facial recognition (iPhone photos anyone?), real-time language translation (that OP google app), object detection (YOLO - you only look once!) and even music composition!

- Neural networks form the basis of deep learning - a subset of machine learning, inspired by the structure of the human brain!

Robotics Research Center

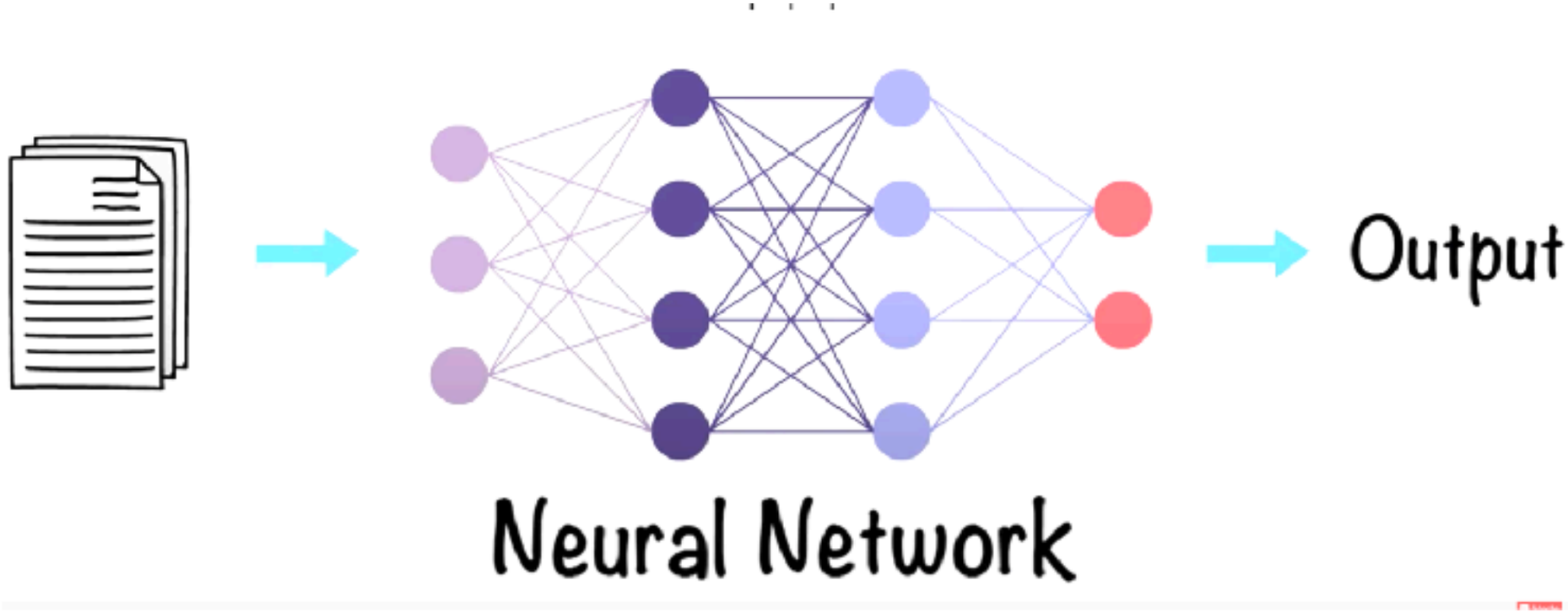




Robotics Research Center

Neural Networks

- Neural networks take in data and train themselves to recognise patterns in this data (updating weights through backpropagation!) by optimising over a loss function.
- They can then predict the outputs for a new set of similar data.



Robotics Research Center

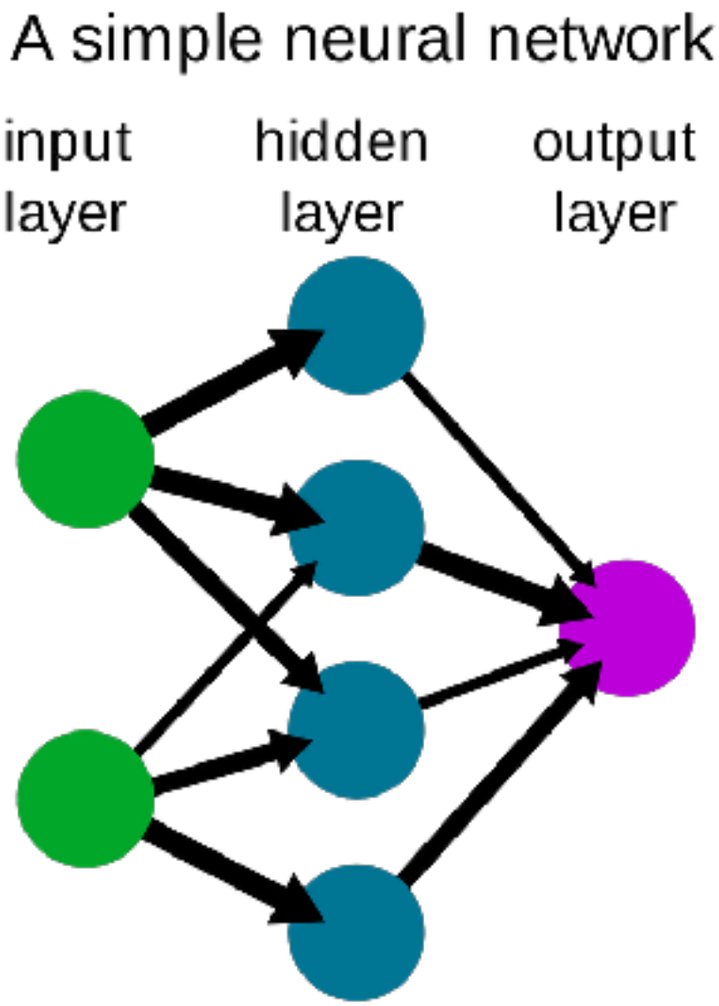
Neural Networks

- Neurons - the core processing units of the network. A neural net is made up of multiple layers of these neurons. “Thing that holds a number”.

- Input Layer - as the name suggests, it receives the input.

- Output Layer - predicts the final output - is it a dog or a cat?

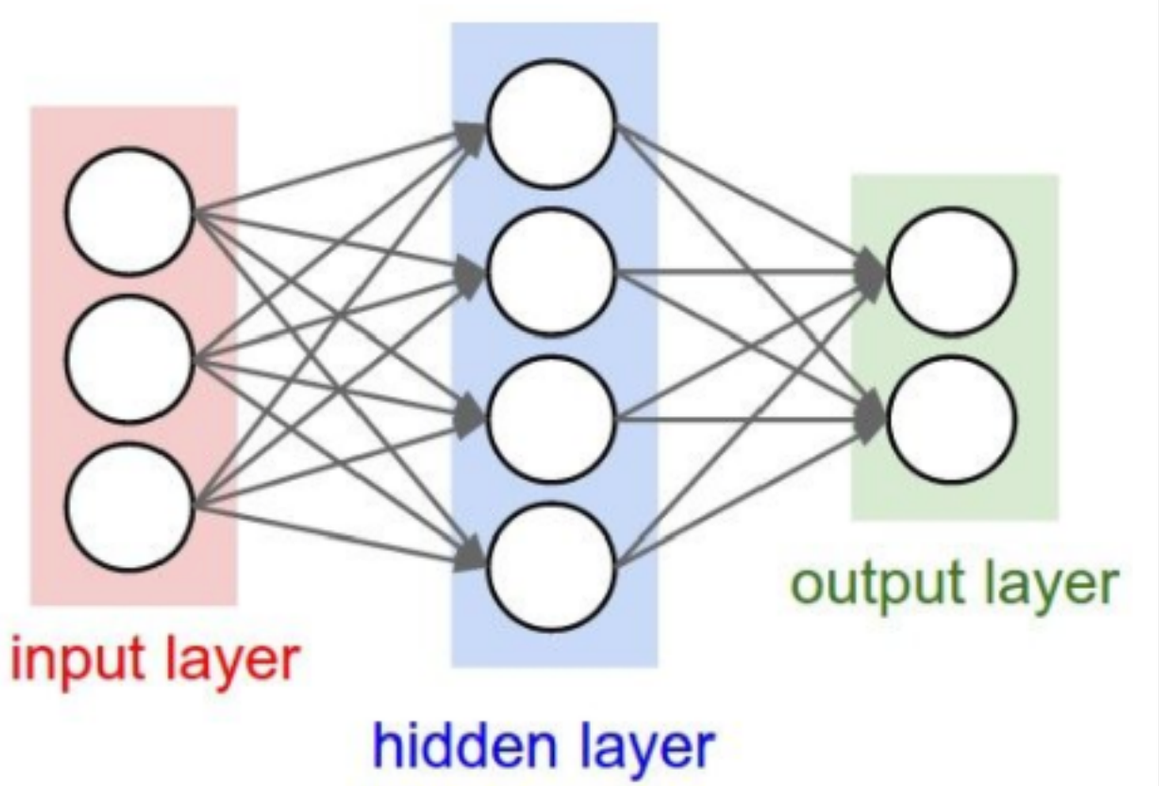
- Hidden layers - between the input and the output layers, there exist multiple *hidden layers* of neurons which perform multiple computations.



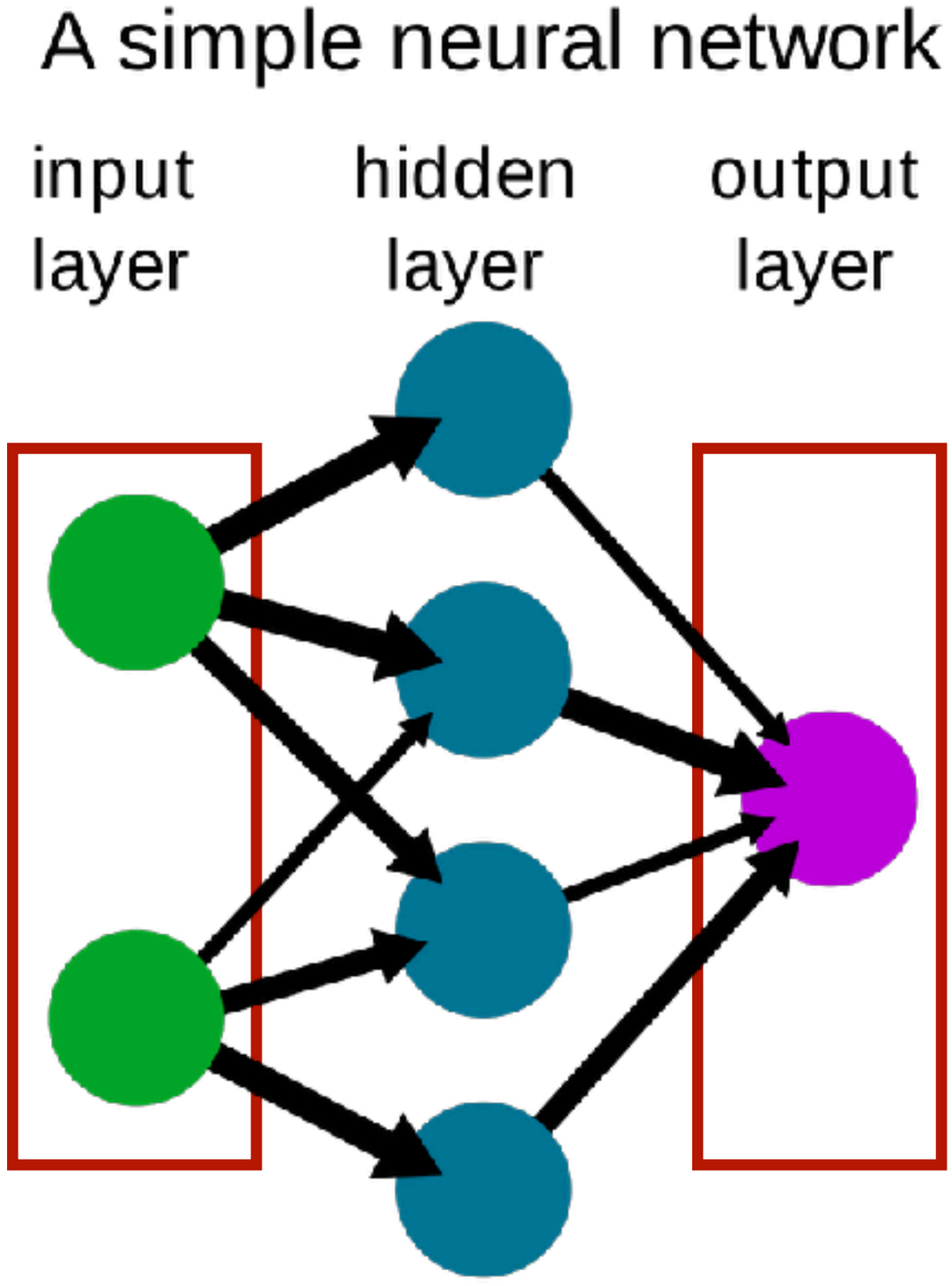
Robotics Research Center

A neural network is a computational graph, with many linear layers and non linearities in between.

We wanna learn the values of these weights, and we do so through optimisation.



Robotics Research Center



Neural networks are made up of layers of neurons - these round circle things.

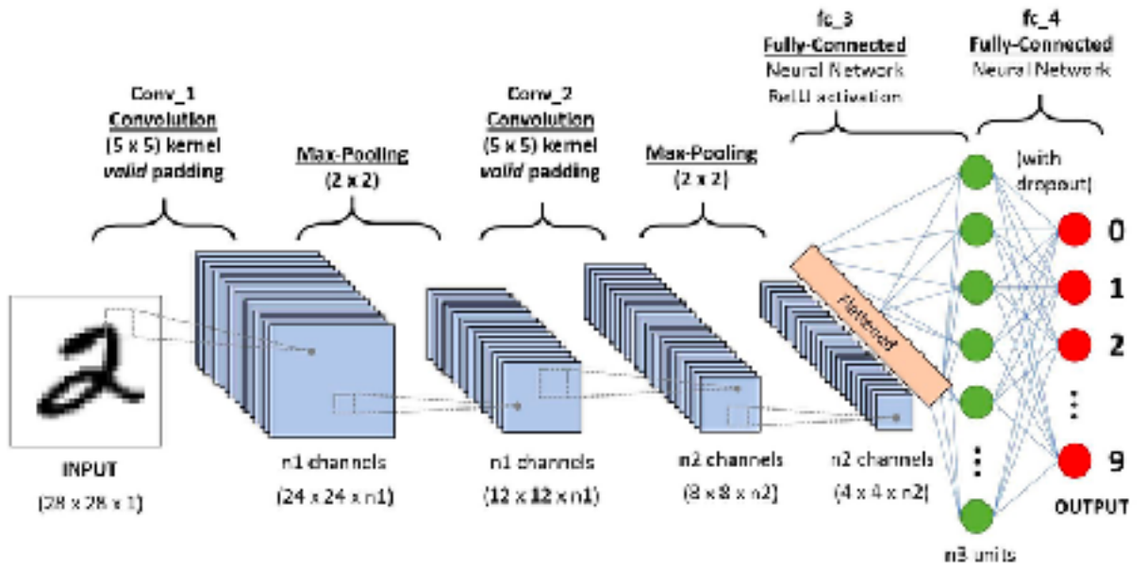
Robotics Research Center

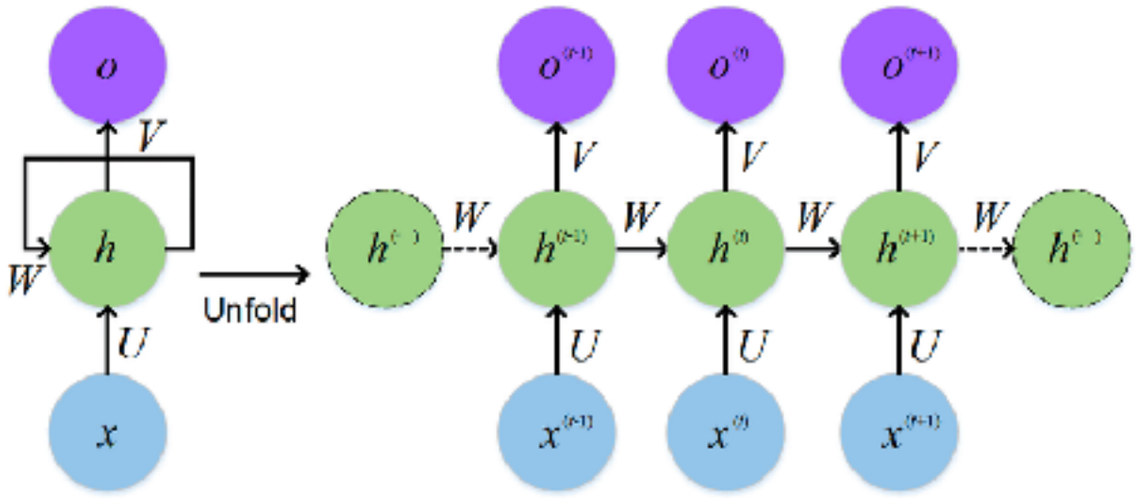
Layers

- Neurons within an artificial neural network are typically organised in layers.

- We use many different kinds of layers in DL, such as Dense layers (fully-connected layers), convolutional layers, pooling layers, recurrent layers etc.

- Different layers perform different kinds of transformations on their input data. So one may be better suited for a particular task than the other.





CNN's - best for image data
RNN - good for time-series data

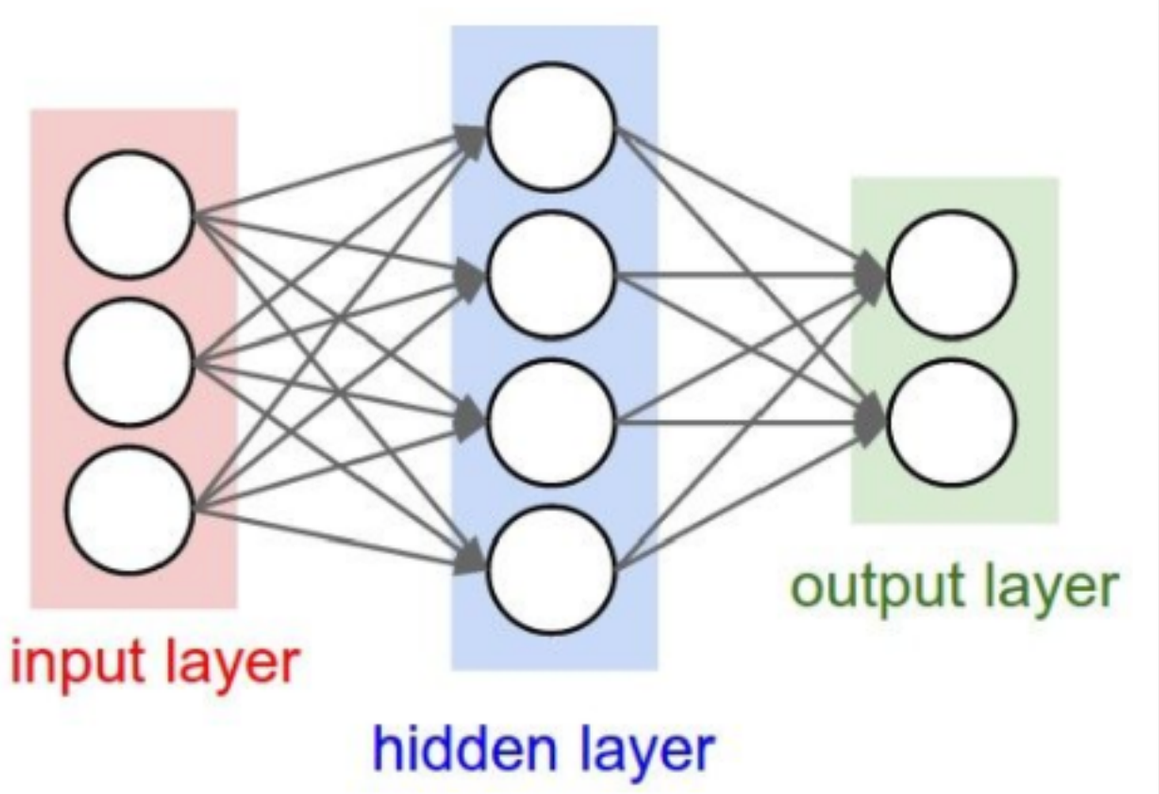
Robotics Research Center

Neural Network

- Each of the nodes in the input layer represents an input feature from each sample in the dataset, which will pass through the model.

- Each of these inputs is connected to each unit in the next layer. Hence, it's a fully connected or a dense layer.

- Each connection will have an assigned "weight" to it - how strong is the connection between those neurons.



Output = weighted sum of inputs
Final output = activation(output)

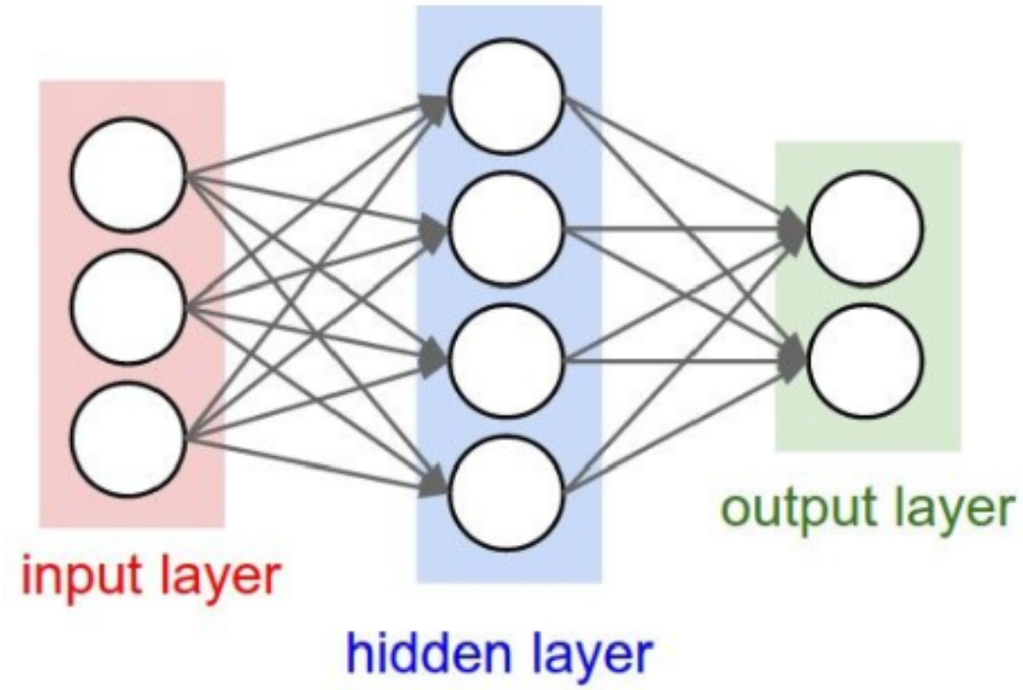
Robotics Research Center

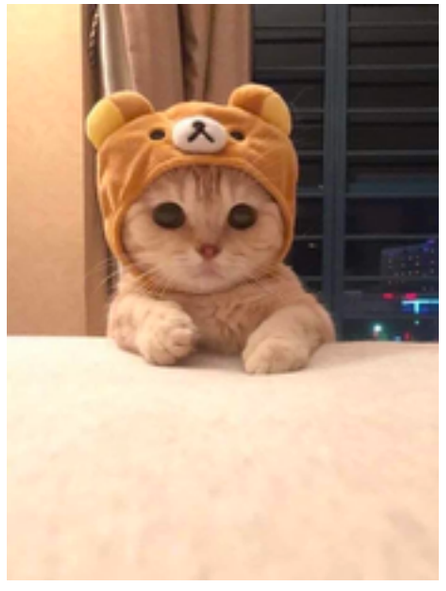
Neural Network

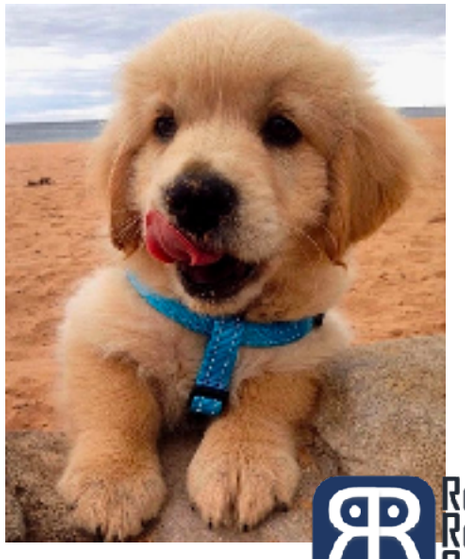
- The output layer here has two neurons. These units typically represent the categories.

- For instance, assume that we are using the network to predict whether a given image is a cat or a dog.

- So the two units in the output layer - one corresponds to a cat and the other, to a dog.



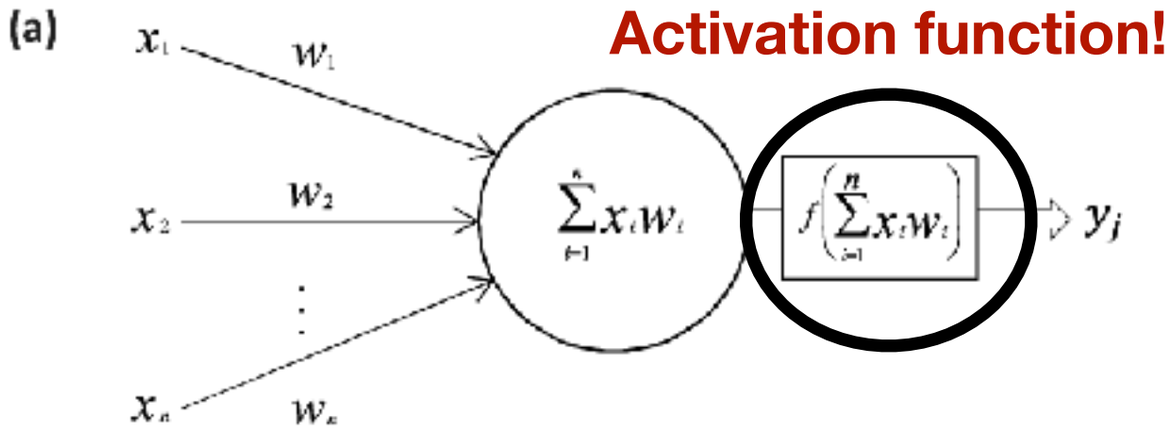


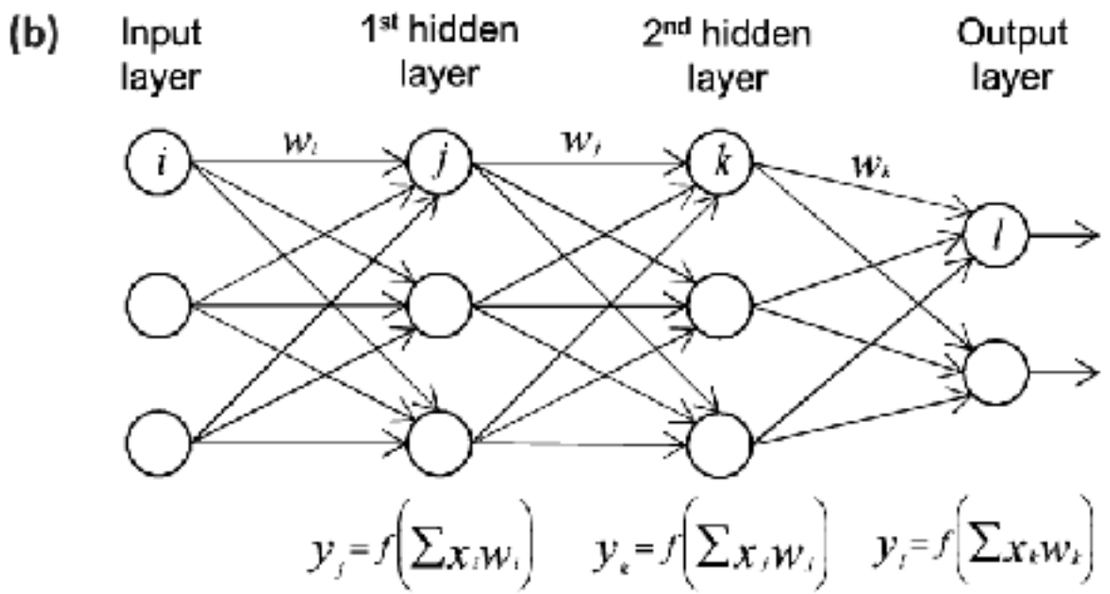


Activation Function

- An activation function of a neutron in an artificial neural network defines the output of that neuron. It takes the weighted sum and passes it through an activation function.

- This activation function does some sort of operation to transform the sum into a number between sum lower and upper limit.





Robotics Research Center

Sigmoid

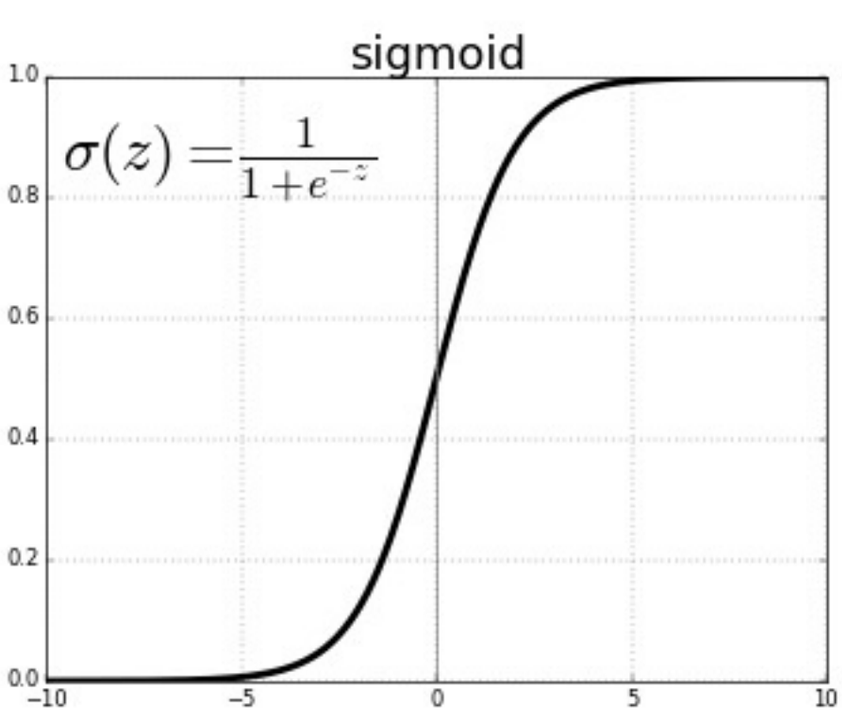
- Sigmoid take the input and if it’s very negative, the number get transformed to a number very close to zero.

- Similarly, if it’s a very positive number, then it transforms it to a number close to 1.

- Hence, zero is the lower limit and one is the upper limit.

$$\sigma(w_1 x_1 + w_2 x_2)$$

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



Robotics Research Center

Why are we doing this?

- This is biological inspired by the activity in our brains! In our brains, different brains are fired/ activated by different stimuli. For instance, when we feel a strong emotion a particular neuron and receptor is fired up in our brain.

- If you smell something pleasant, another different set of neurons will fire up.

- And if you smoke up, a totally different neuron will get activated in your brain.

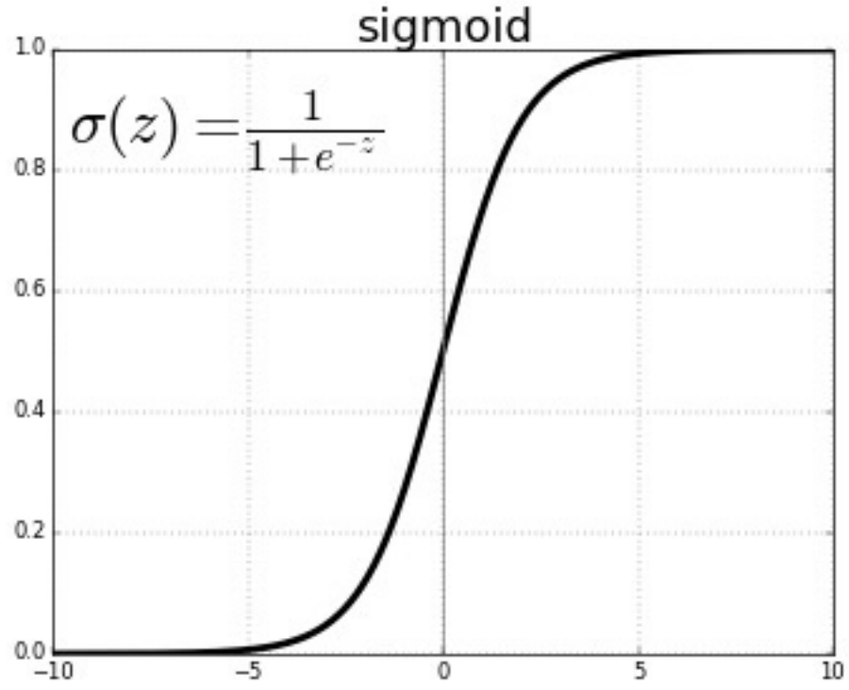
0 - neuron doesn’t fire
1 - neuron fires!

Robotics Research Center

Sigmoid

- In sigmoid, the value will be between zero and one. So if it’s closer to one, it’s more activated!

- And if it’s close to zero, it won’t be as activated.



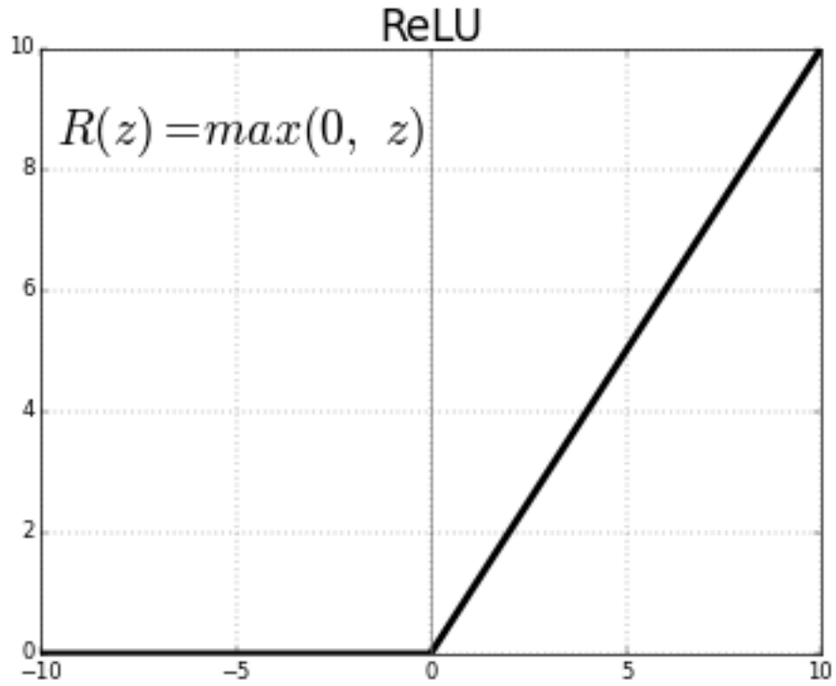
Robotics Research Center

ReLU

- ReLU - short for Rectified Linear Unit - is one of the most commonly used activation functions today.

- Unlike sigmoid, it does NOT transform the value between zero and one.

- Rather it transforms the input to the maximum of either zero, or the input itself!



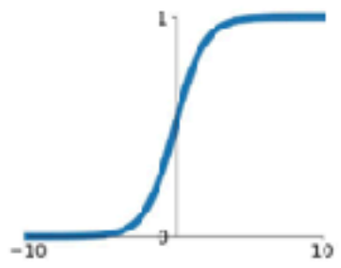
The more positive the neuron, the more activated it is!

Robotics Research Center

Activation Functions

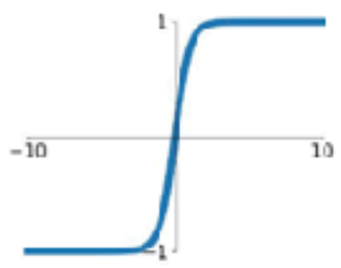
Sigmoid

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



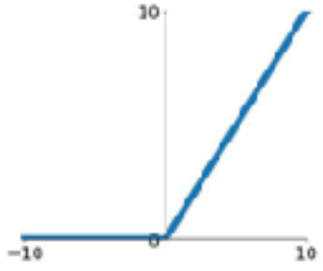
tanh

$$\tanh(x)$$



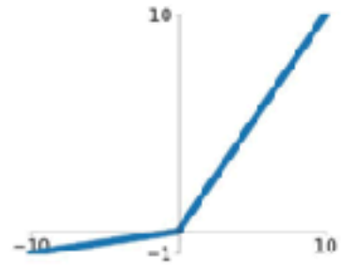
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

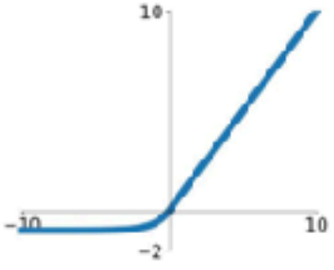


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

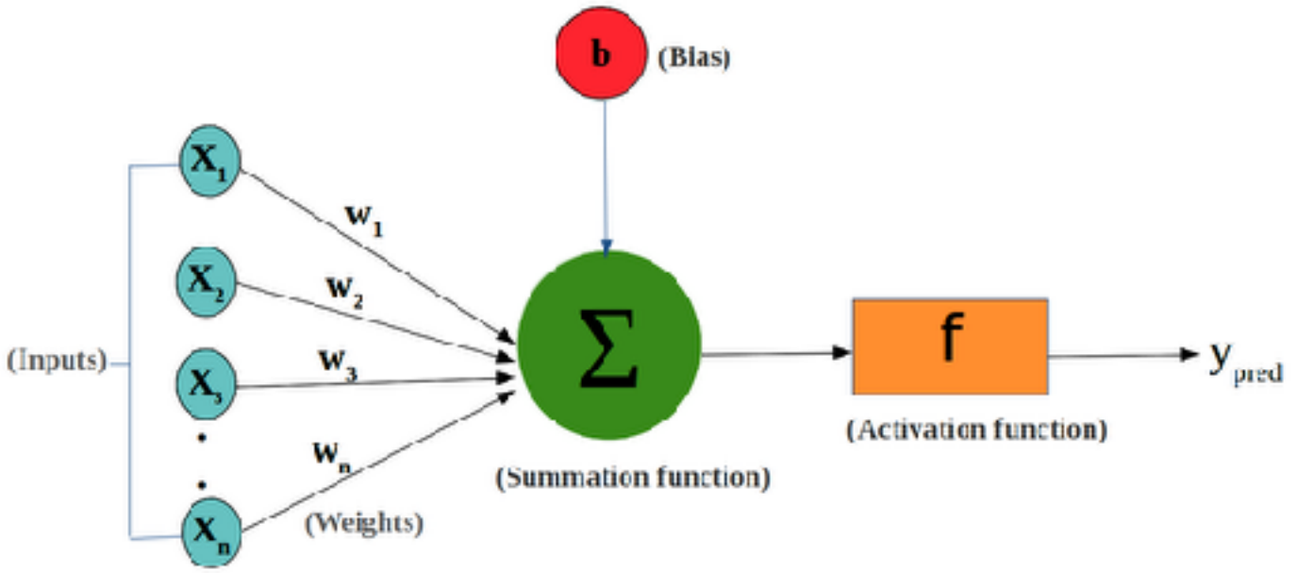


- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU / SELU
  - To squeeze out some marginal gains
- Don't use sigmoid or tanh

Training a Neural Network

- What does it mean to train an artificial neural network? So far we've seen the architecture of a neural net and set up the input, hidden and output layers, along with activation functions.

- When we train our neural network, we are trying to solve an optimisation problem. We are trying to optimise the **weights** of the network - those w's assigned to the connections.



When we train a neural net, these weights keep getting constantly updated in order to reach the optimal value - the one that will lead to an accurate prediction.

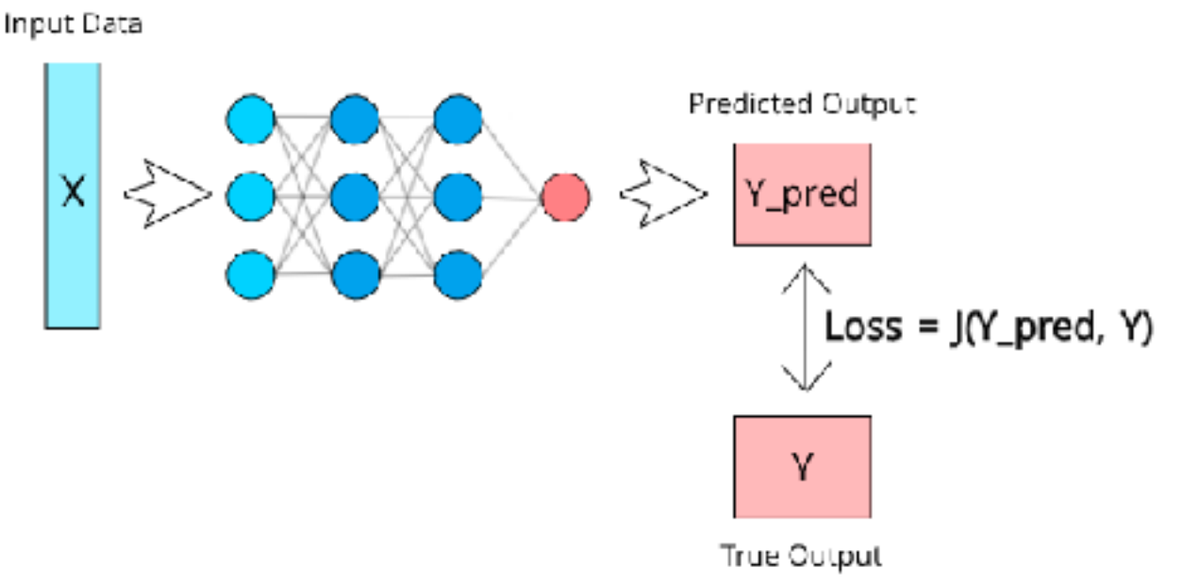
Robotics Research Center

Training NN

- How the weights are being optimised - depends on the optimisation algorithm or the “optimiser”.

- The famous optimiser is SGD - stochastic gradient descent. The objective is to minimise the loss function. So we assign weights such that the loss reaches as minimal a value as possible.

- Example of a loss function can be something like MSE - mean square error.

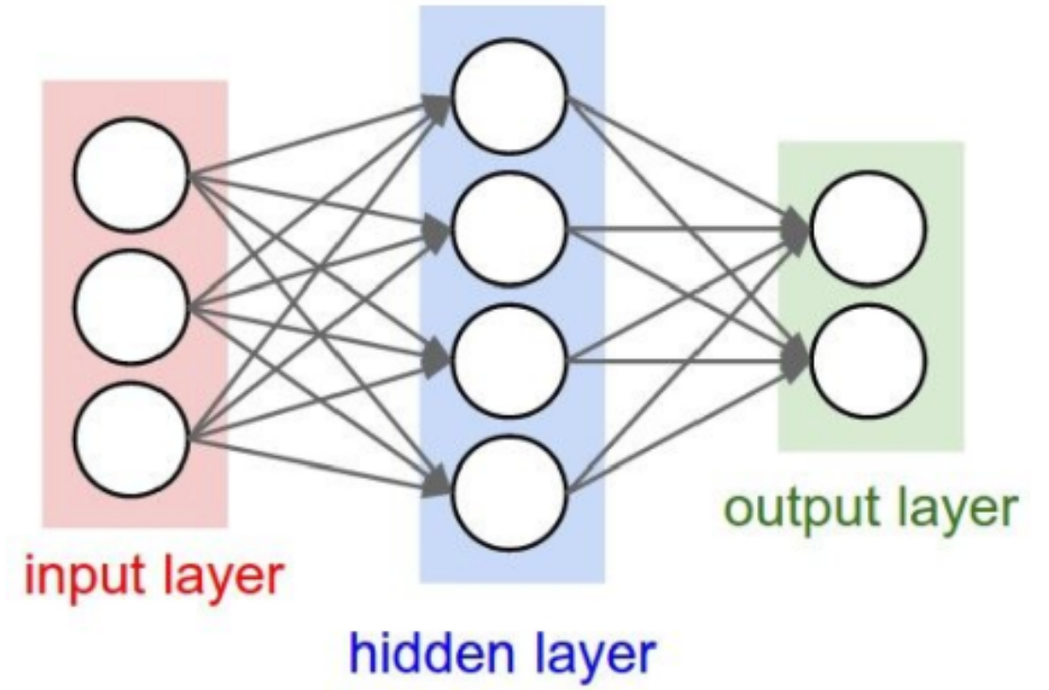


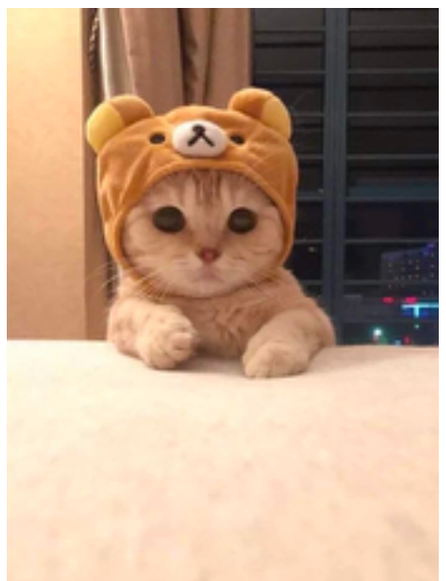
Robotics Research Center

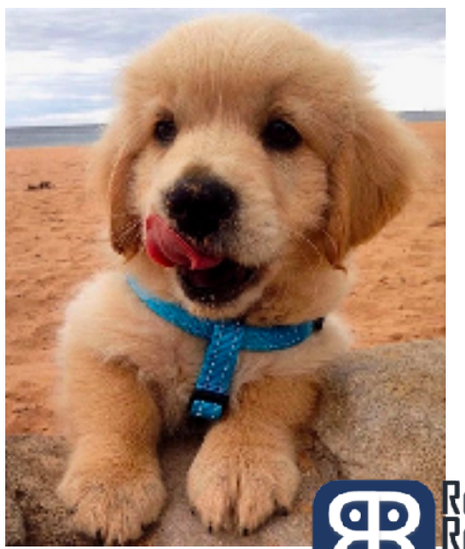
Training NN

- During training, we supply our model with **data and labels (ground truth).**

- Suppose we are training a network to identify whether the given image is a cat or a dog, then our training data will be images of cats and dogs, ALONG with their ground truth label value.







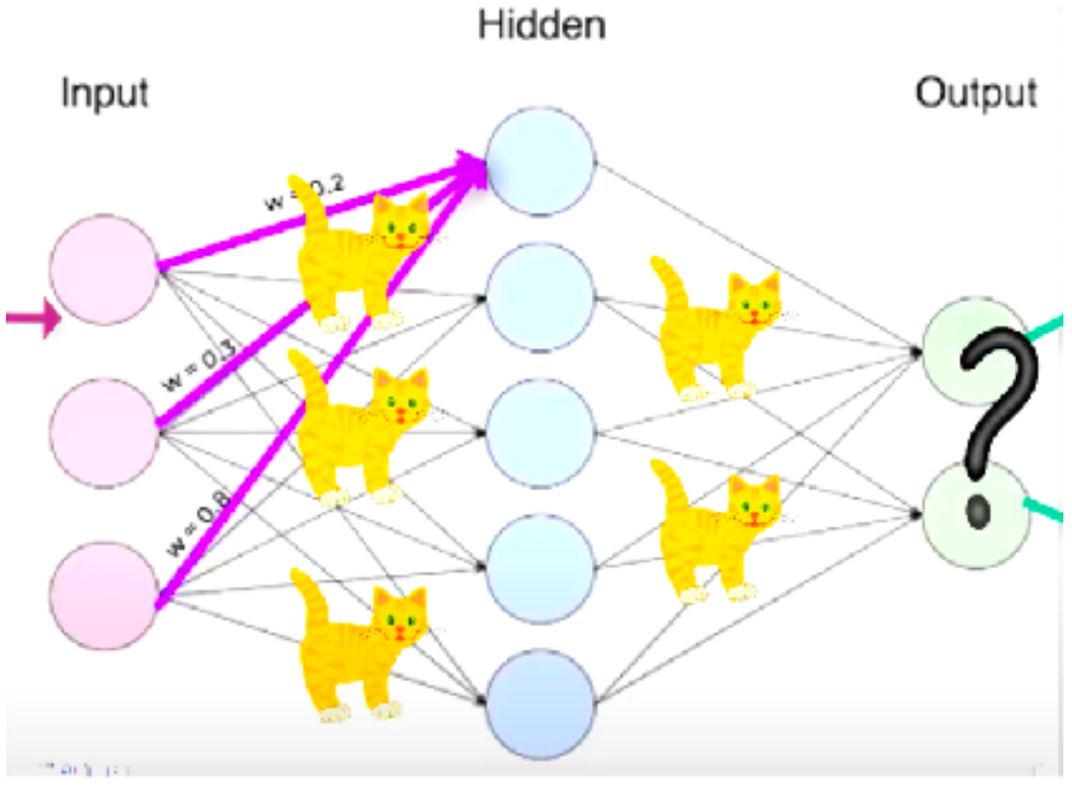
Robotics Research Center

Training NN

- Suppose we input a cat image to this network and forward propagate it, then in the end, we get a 2x1 output - the probabilities of what the network thinks - whether it's a cat or a dog.

- It's possible it says that it assigns a value of 0.6 to cat and 0.4 to dog. It's okay, but it's not very accurate.

- In this case the loss function is going to be the error between the prediction and the true label.



After passing in all data once, we are going to continue inputting all data over and over again, which is when the model will begin to learn - as it optimises over the loss function!

Robotics Research Center

What does it mean to learn?

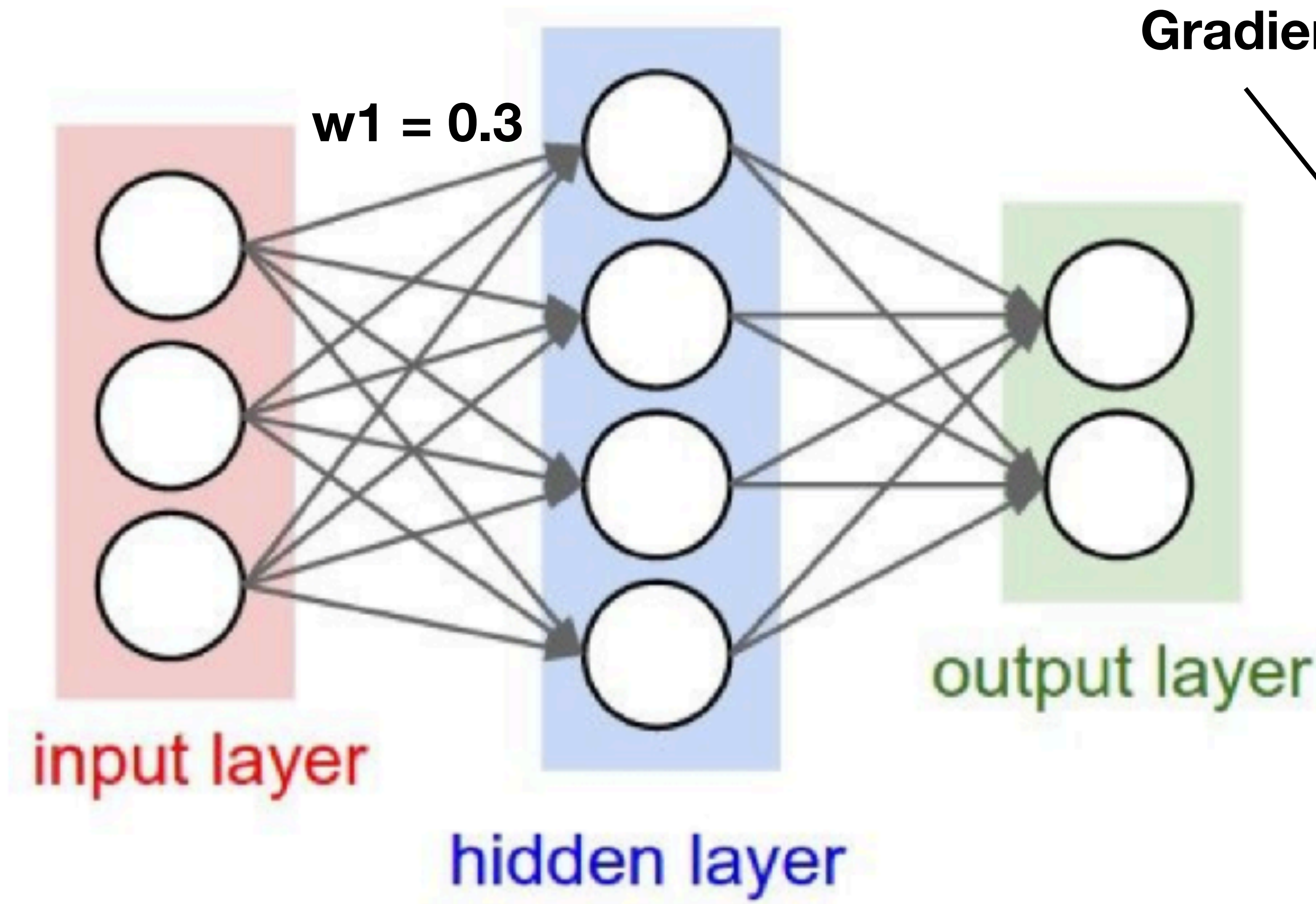
- So we have done a single pass of the data (forward propagation) through our model - an epoch!
- After this single epoch, the same data will be passed over and over again through multiple epochs - which is when the model will learn.
- At the end of the model, it will give its output and also compute the loss/error by looking at its prediction and comparing with ground truth. It will compute the **gradient of the loss function** with respect to each of the weights.

$$\frac{\partial L}{\partial w1}, \frac{\partial L}{\partial w2} \cdots$$

- But what does it exactly mean to LEARN? In the starting, our model is simply initialised with arbitrary weights. Whenever the weights are updated, our network LEARNS!

Robotics Research Center

Updating Weights



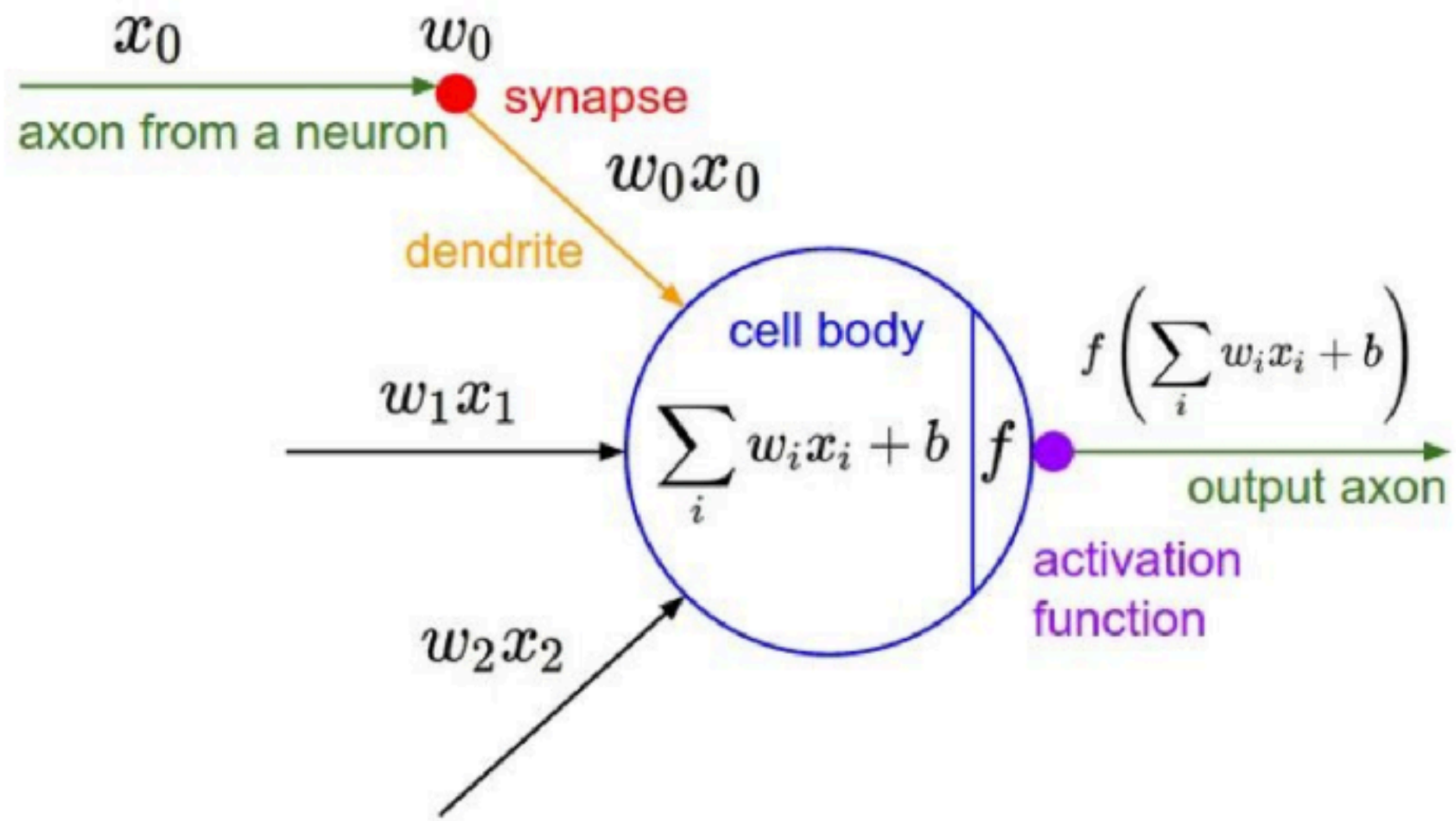
Gradient

Learning Rate

$$\frac{\partial L}{\partial w1} * \alpha$$

Robotics Research Center

Just a quick recap…



Robotics Research Center

The Basic Pipeline

- We sample out a batch of the data.
- We forward propagate it.
- We compute the loss at the end, by comparing prediction and ground truth.
- We back-propagate it and calculate the gradients on the way back.
- We update the parameters/weights using these gradients, by taking steps in the direction of the negative gradient.

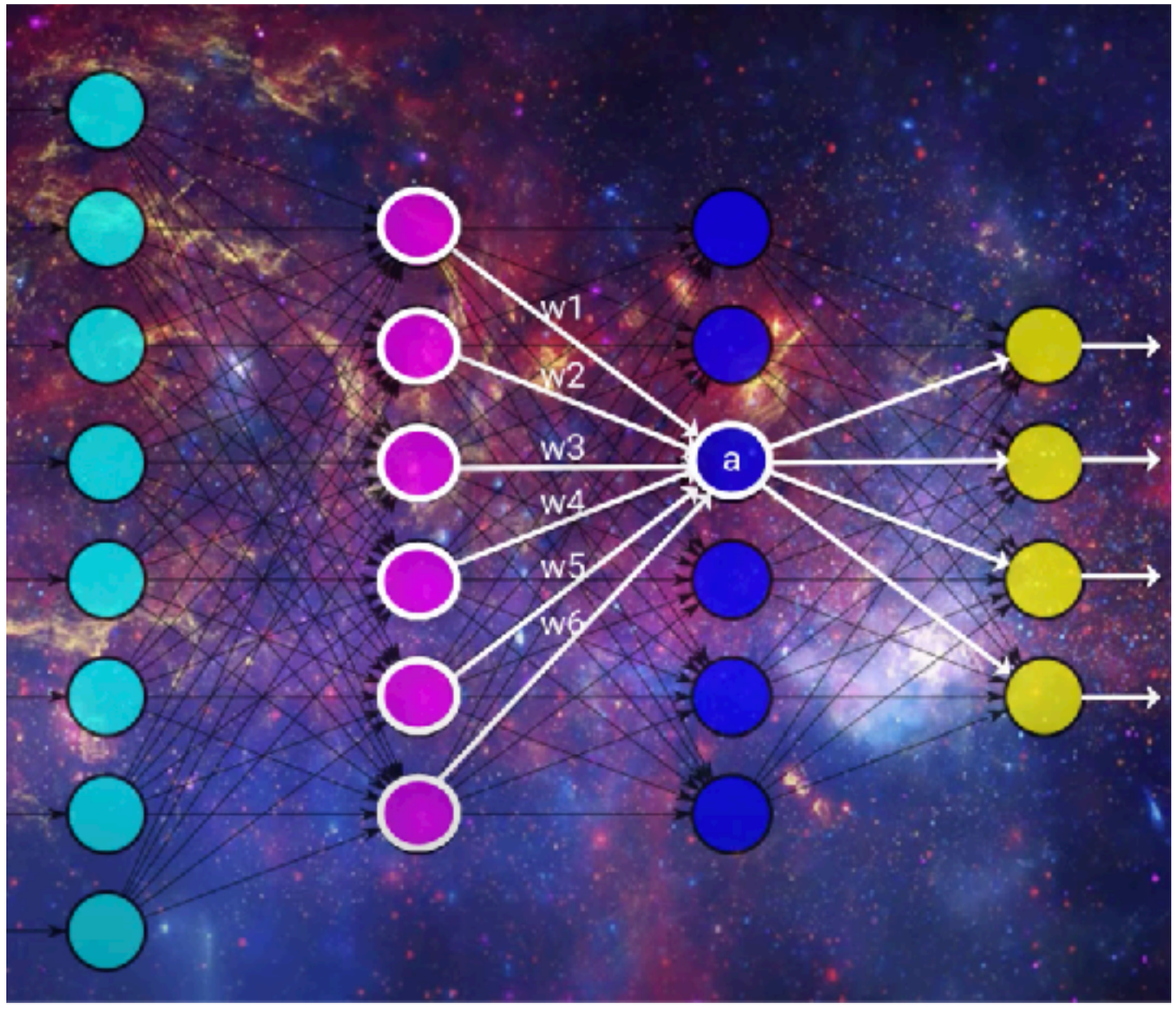
Forward Propagation

- The process of moving the data forward or through the model is known as forward propagation.

- The data propagates forward until it rreaches the output layer. Each node in the model receives its input from the previous layer.

- The input of the neuron is the weighted sum of the weights at each of the connection multiplied by previous layer's output.

- We pass this through the activation function - this is the final output of that layer which is passed on as input to next layer.



Robotics Research Center

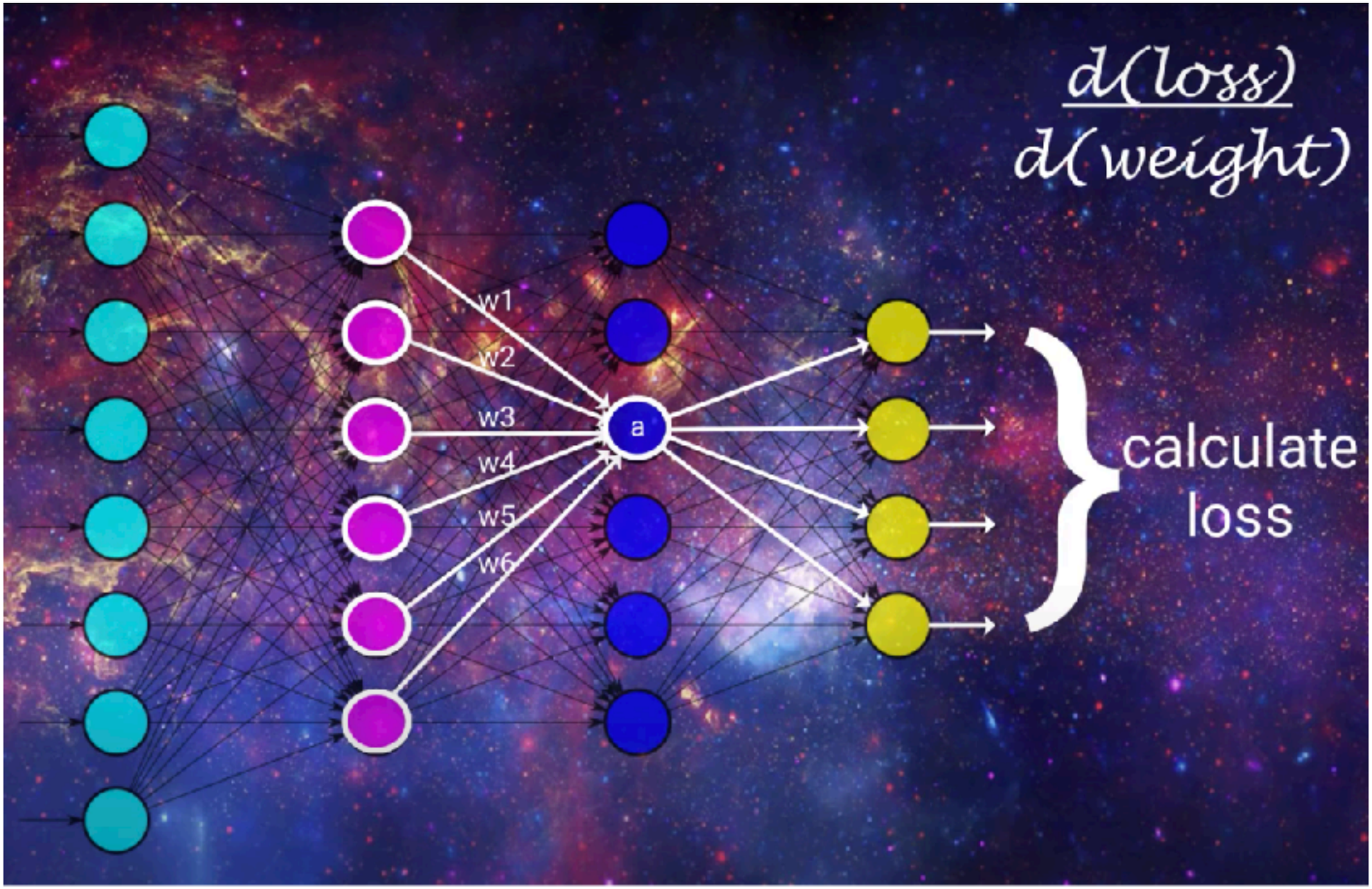
Backpropagation

- Instead of diving right into the calculus, let's first understand this intuitively!

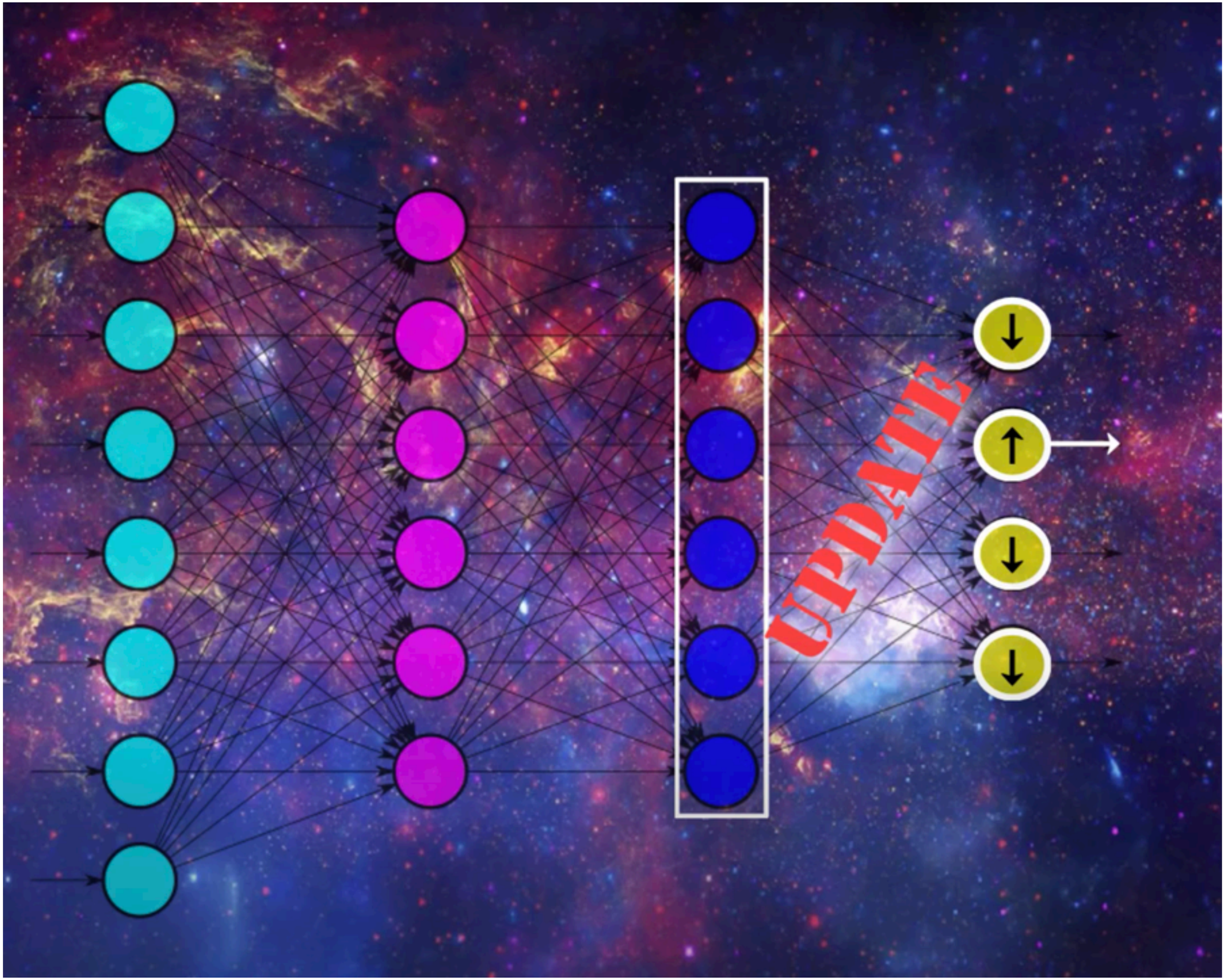
- While training a neural network, an optimiser like SGD works to minimise the loss function by updating weights with each epoch.

- This happens by computing the derivative/gradient of the loss function with respect to the weights in the model. This is essentially backpropagation.

Training NN



Robotics
Research
Center

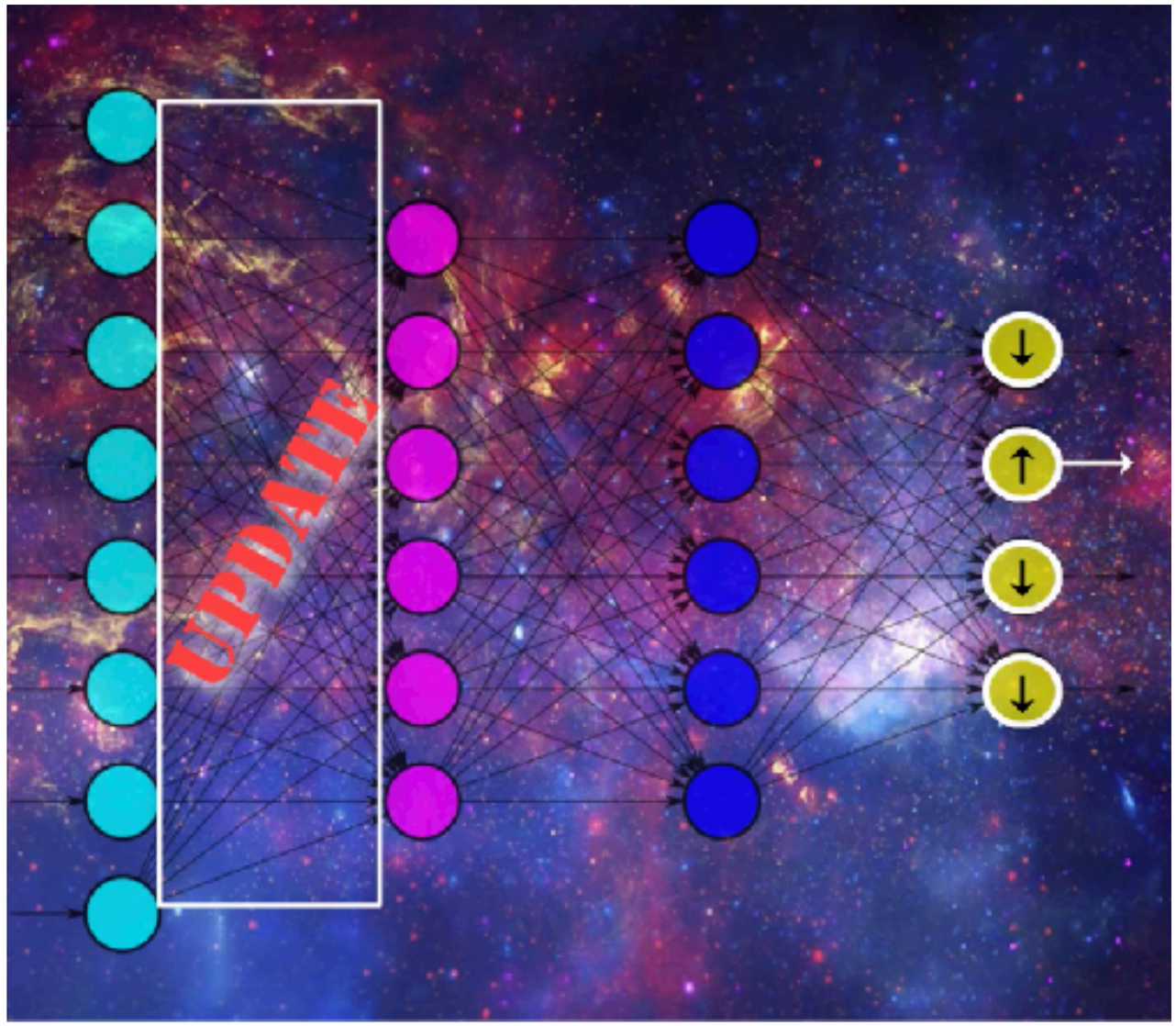

UPDATE

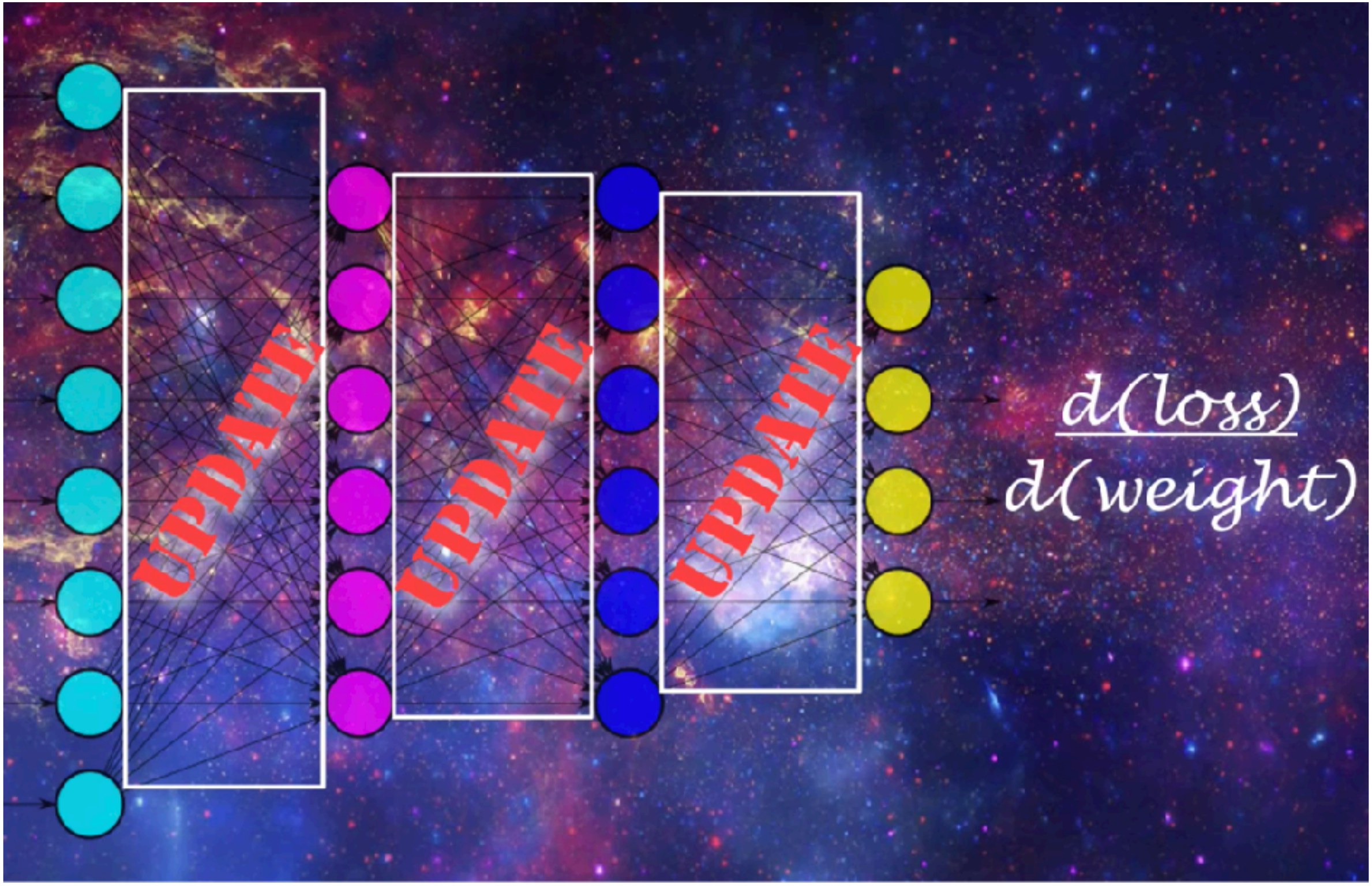
Backprop.

- We are moving backwards through the network, updating the weights in order to slightly move the values in the direction in which they should be going - in order to lower the loss.

- We try to increase the value for correct output node and decrease that for incorrect node - thereby reducing loss.

- We do this in a backwards fashion - output of each layer depends on weights in previous layers.




UPDATE
UPDATE
UPDATE
d(loss)
d(weight)

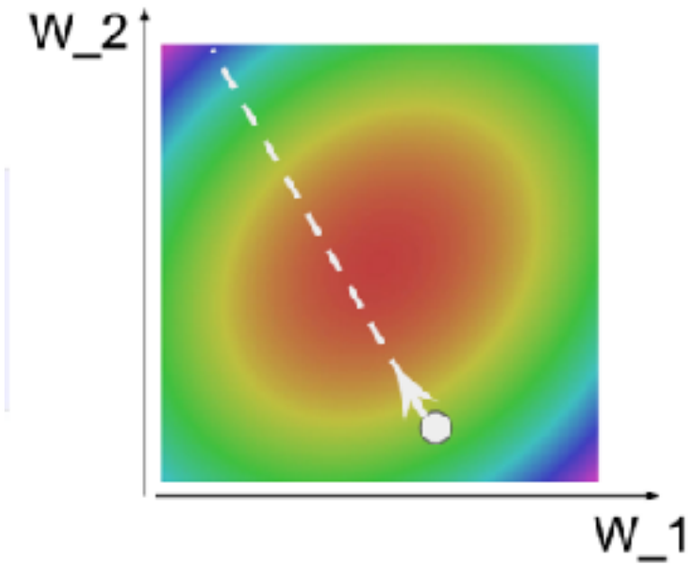
Optimisation Algorithms

- The core strategy in training Neural Networks in solving an optimisation problem! We have some sort of a loss function, that tells us how good our weights are. We need to optimise over the weights and get to the most minimum loss possible.

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Stochastic Gradient Descent
A simple optimisation algorithm



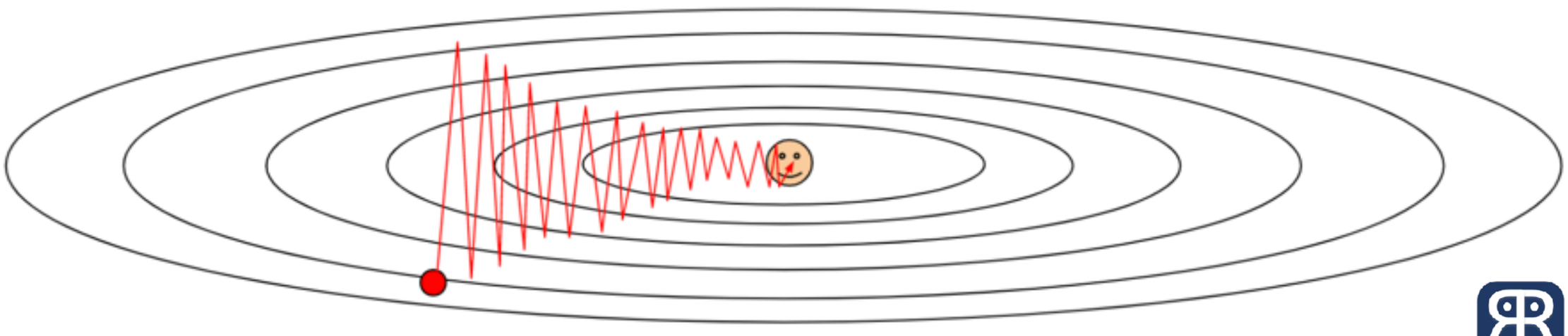
Robotics Research Center

Problems with SGD

- What if the loss is sensitive in one direction and not in the other? That means, it changes quickly in w1 but slowly in w2.

- Very slow progress along horizontal dimension (the slow one). When we calculate the gradients and update, we kinda zig zag back and forth. Nasty progress along the fast changing one.

- This gets worse when we have more dimensions - which means more directions along which it can move.



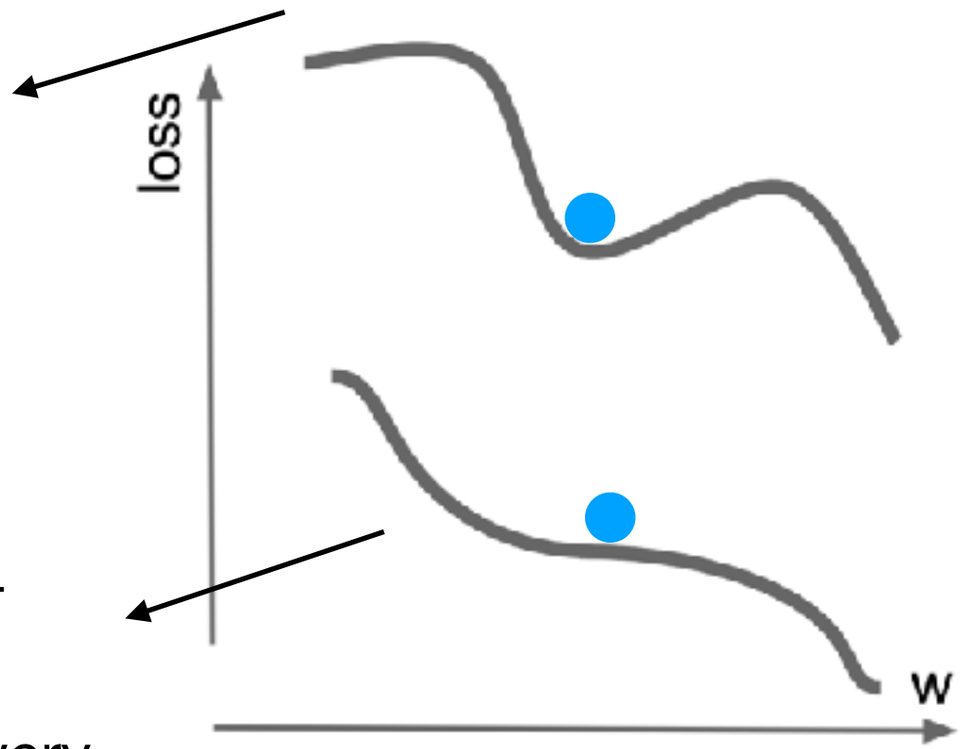
Robotics Research Center

- If there a hundreds of different directions to move in, and the ratio of the largest and smallest one is quite large, then SGD wouldn’t perform well.

- Another problem is that of local minima or saddle point.

In this situation, SGD will get stuck!
Because at the local minima, the gradient is zero as it’s locally flat.
In SGD, we compute the gradient and we move in the negative direction. But since we are at a local minima, we’ll get stuck at this point.

Another Problem: In one direction we go up,
And in another we go down - SADDLE POINT
Then also it may get stuck.
The gradient is exactly zero, but the slope is very small. So if the gradient is very small,
we make REALLY SLOW progress



One dimensional weight

Robotics Research Center

‘Stochastic’

- Loss function is computed by finding the loss for many different examples. But there could be too many, so we estimate the loss and gradient using a mini batch of training examples.

- So if there’s noise in the gradient estimates, then vanilla SGD might just wander around a bit before actually going towards the minima.

- What if we use just normal GD, that is, full-batch? Doesn’t really solve these problems, so we need a fancier optimisation problem.