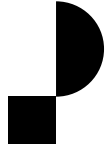


# The Techie Bits

# Machine Learning 101

Graduate Workshops 2020



# What you have just achieved

- Created your own imaging dataset to train an image classifier using supervised learning
- Performed data augmentation and pre-processing
- Built and trained a machine learning model using transfer learning:
  - Type of model: Residual Neural Network with 34 layers (ResNet34)
- Inspected the losses of the model to try understand why it might be performing badly
- Used the loss and final layer activations to clean up your imaging dataset





```
: data = ImageDataBunch.from_folder(path, train=., valid_pct=0.2, ds_tfms=get_transforms(), size=224, num_workers=4).normalize(in
```

# Dataset creation and split

## For every dataset you need a:

- **Training set** – used to train the model
- **Validation set** – used to validate model performance at the end of each epoch
- **Test set** – used to evaluate the model at the end of training

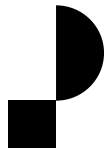
## Some common splits are:

### For training and validation split:

- 80% for training
- 20% for validation

### For training, validation and test:

- 70% for training
- 15% for validation
- 15% for testing

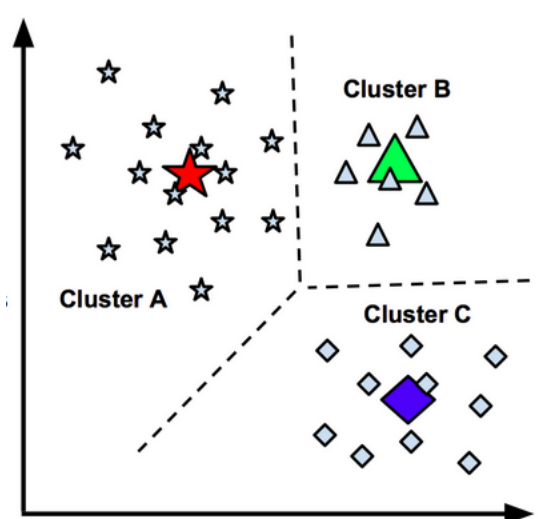


# Supervised versus unsupervised

## Supervised learning

“Learning from data that contains the inputs and the desired outputs”

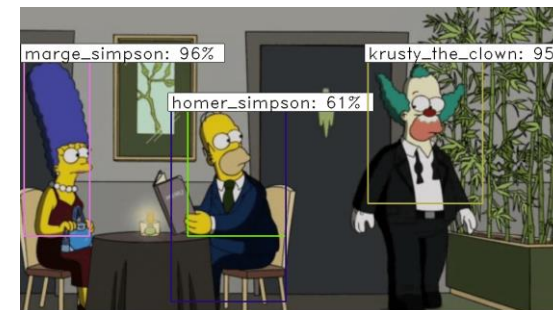
E.g. Classification, regression

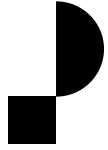


## Unsupervised learning

“Learning from data that contains only the inputs”

E.g. Clustering, anomaly detection





# Classification versus Regression

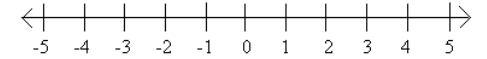
## Classification

When we are trying to predict the category an input belongs to. E.g. Classifying an image as a cat or dog, events etc.



## Regression

When the target variable is numeric and we are predicting a continuous number. E.g. Predicting the pricing of stock options.





```
: data = ImageDataBunch.from_folder(path, train='.', valid_pct=0.2, ds_tfms=get_transforms(), size=224, num_workers=4).normalize(in
```

# Data Augmentation

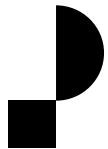
[https://docs.fast.ai/vision.transform.html#get\\_transforms](https://docs.fast.ai/vision.transform.html#get_transforms)

## What?

- **Small random transformations that don't change what's inside the image (to the human eye) but alter the pixel values**
  - Rotation
  - Zoom
  - Translation
  - Contrast changes

## Why?

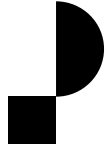
- **Increase the size of your training set**
- **Improve model generalisation**



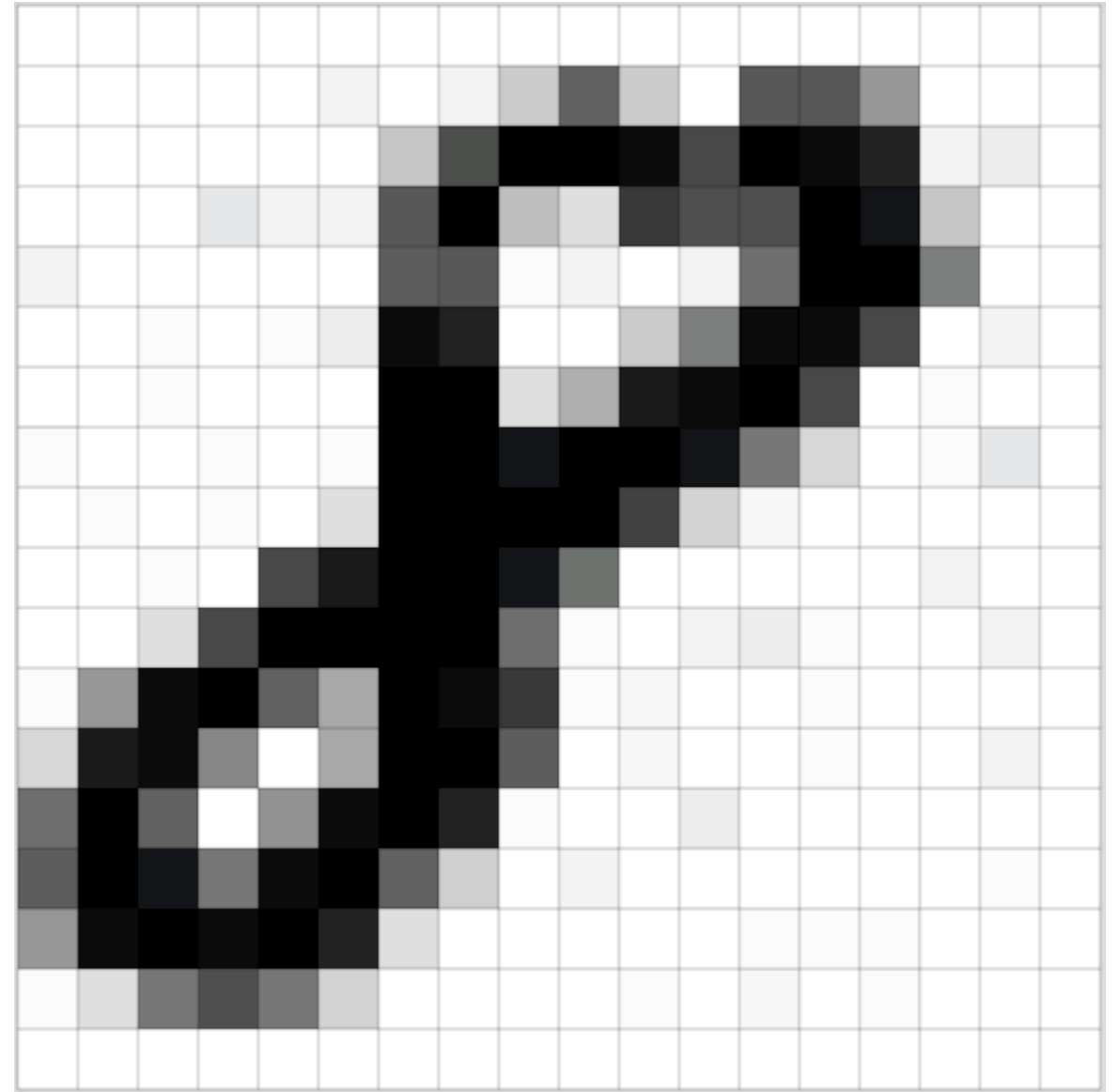
```
: dataBunch.from_folder(path, train='.', valid_pct=0.2, ds_tfms=get_transforms(), size=224, num_workers=4).normalize(imagenet_stats)
```

# Data Pre-processing

- **Image resizing** – we resize all the images to be the same size as this is a requirement of the neural network
- **Image normalisation** – we normalise the image to have a mean of 0 and a standard deviation of 1
  - What does this mean?
    - Calculate the mean pixel value and standard deviation
    - Subtract the mean from each pixel and divide by the standard deviation
- **Machine learning algorithms behave better when the data is normalised**



**Remember! Images  
are pixel values.**





pixel image

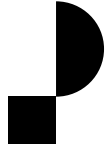


imread



3-channel matrix

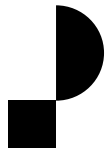
					Blue
					Green
					255 134 93 22
					Red
					255 134 202 22
255	231	42	22	4	30
123	94	83	2	92	124
34	44	187	92	14	142
34	76	232	124	14	
67	83	194	202		



# Model build and training

This section we will cover:

- Linear regression
- Logistic regression
- Neural networks
- Transfer learning



# Linear Regression

$$y = ax + b$$

**y** = output

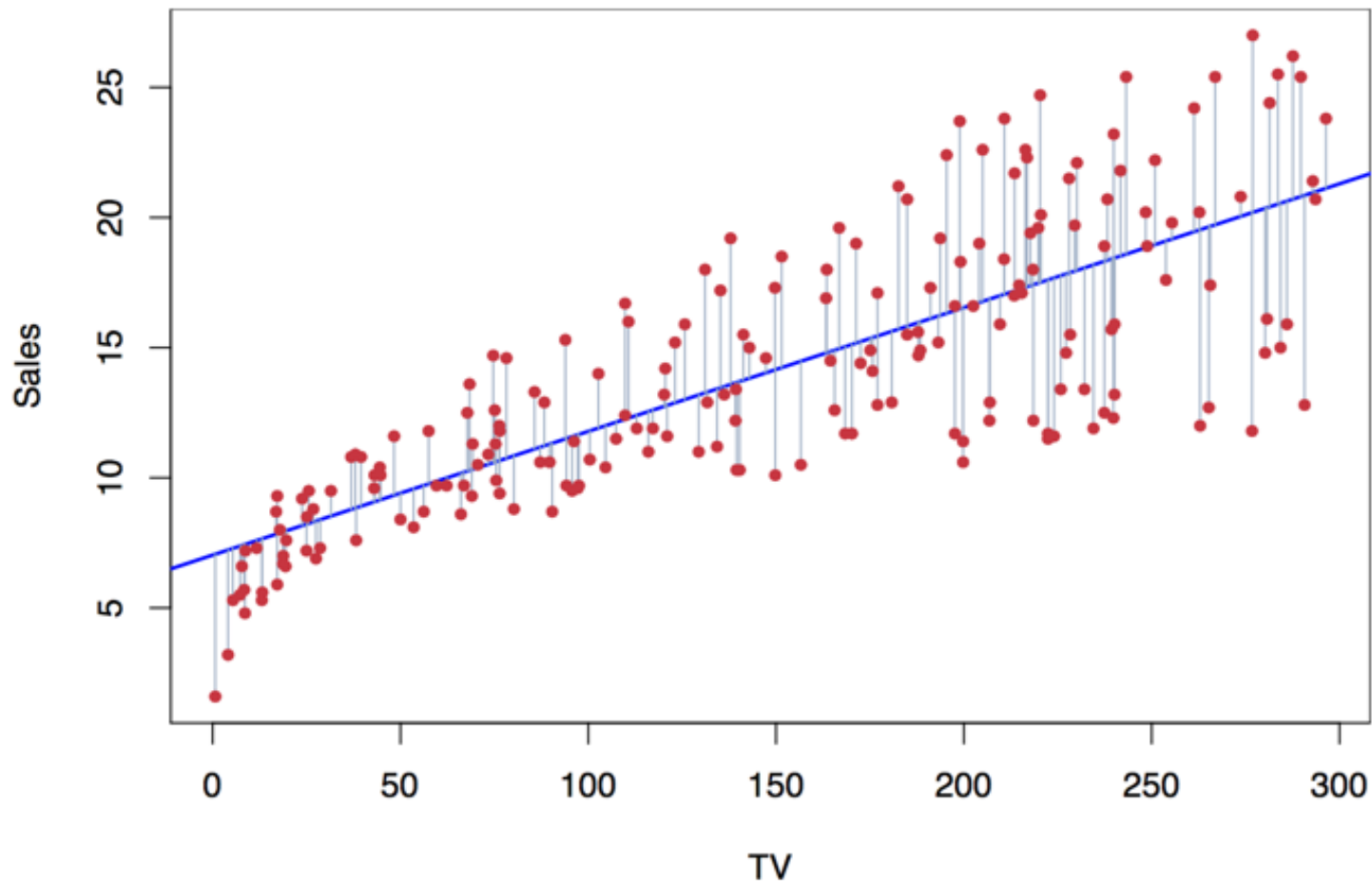
**x** = inputs

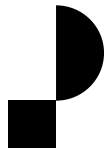
**a, b** => parameters we need to learn

$$\text{loss function} = \sum_i (y_i - f_i)^2$$

$y_i$  = actual value

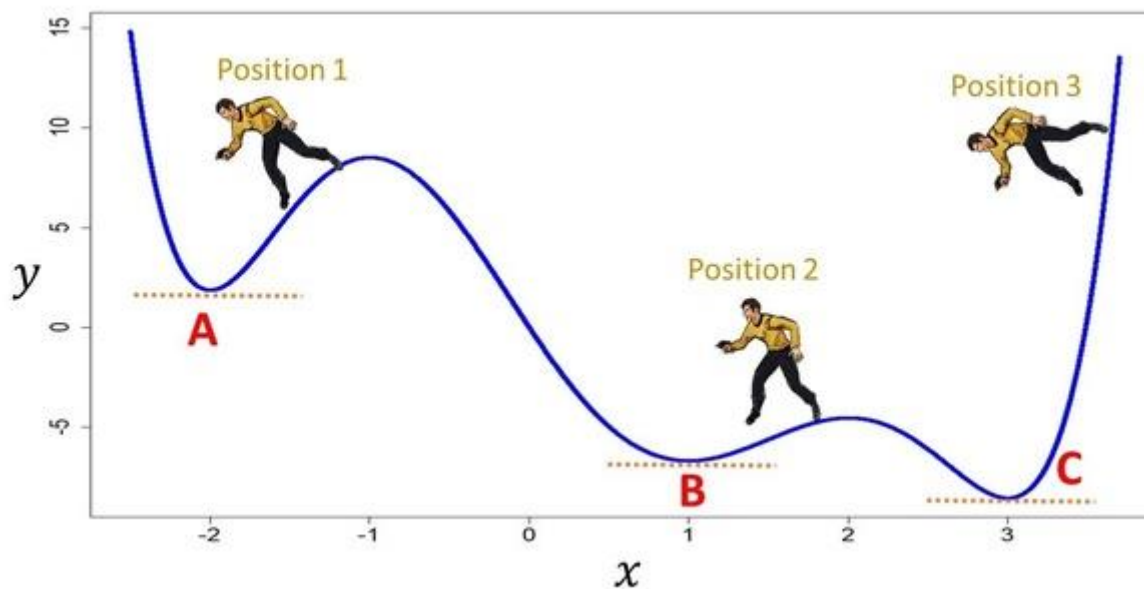
$f_i$  = predicted value

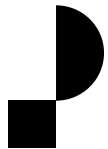




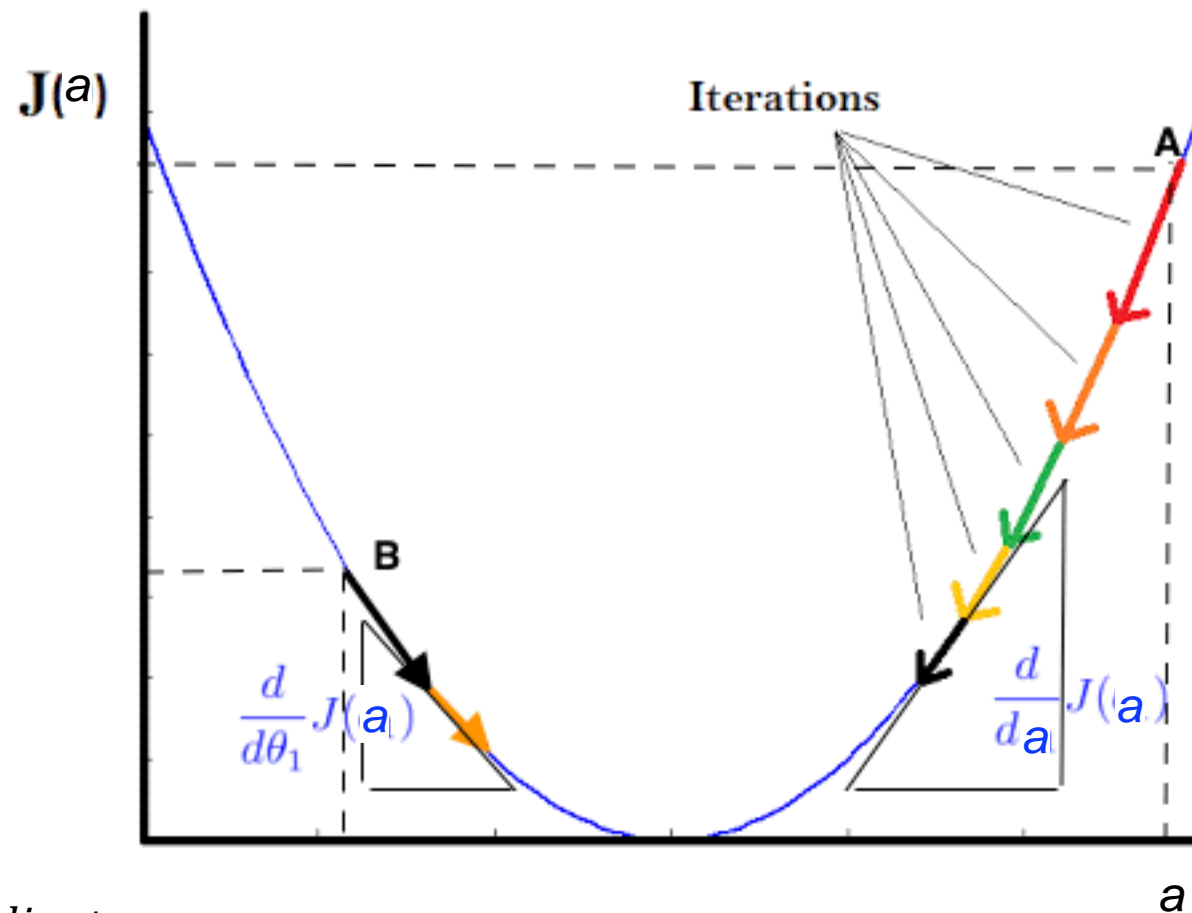
# How do we find the best values for $a$ and $b$ ?

We use something called Gradient Descent!

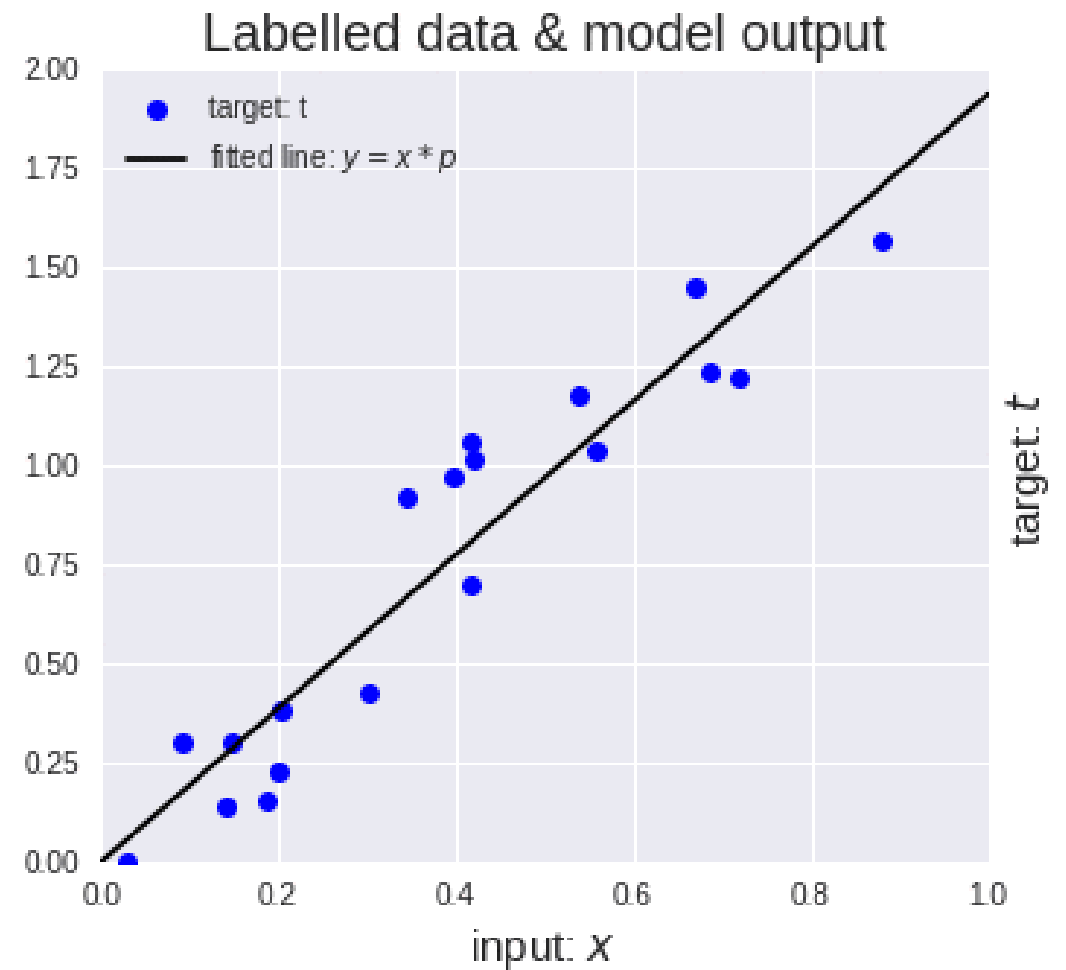
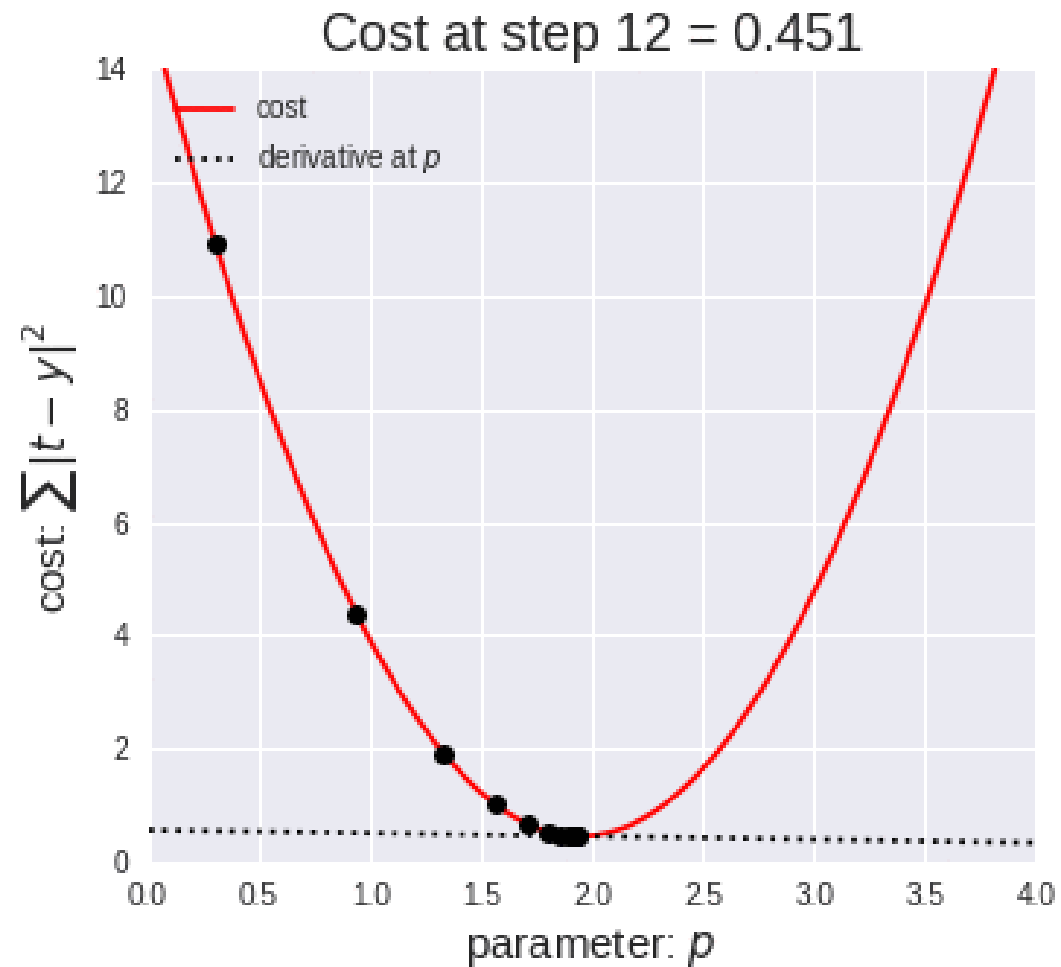


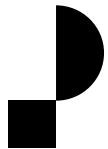


# Gradient Descent

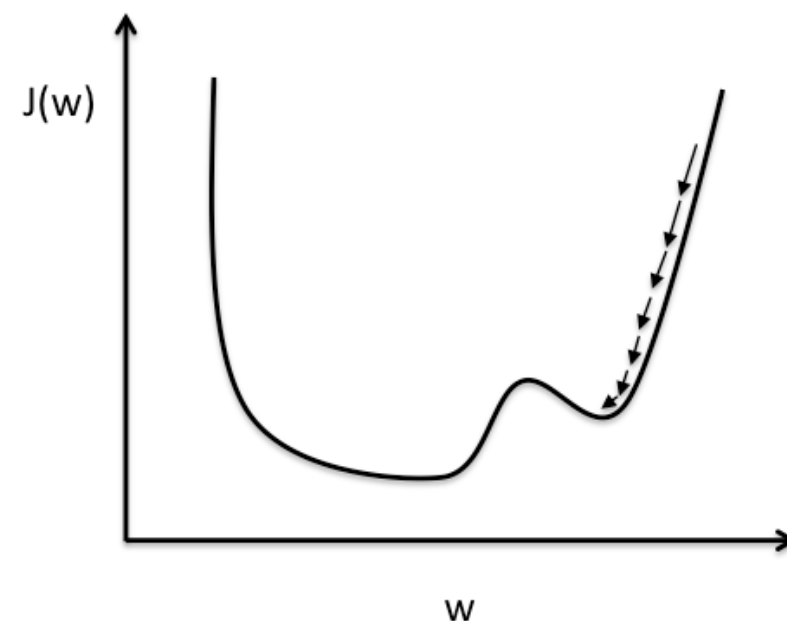
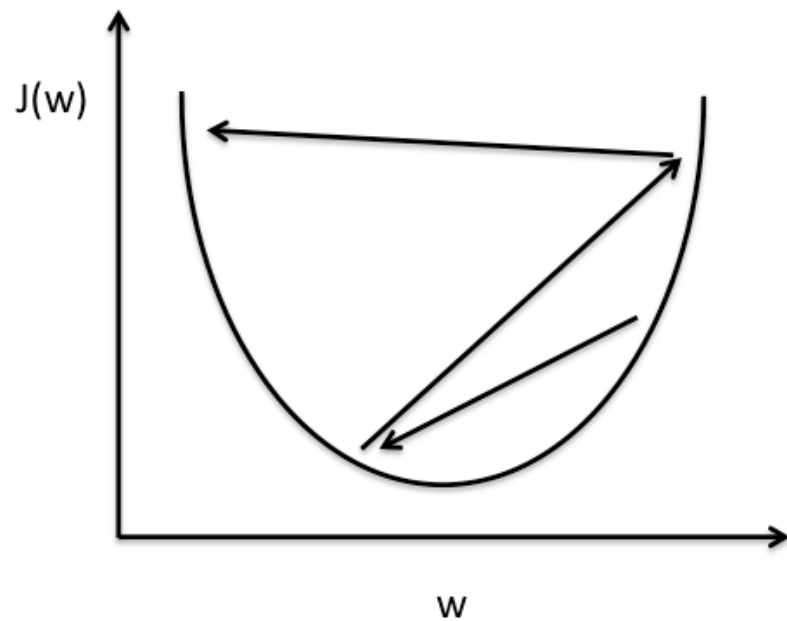


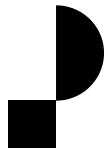
$parameters = parameters - learning\ rate * gradient$



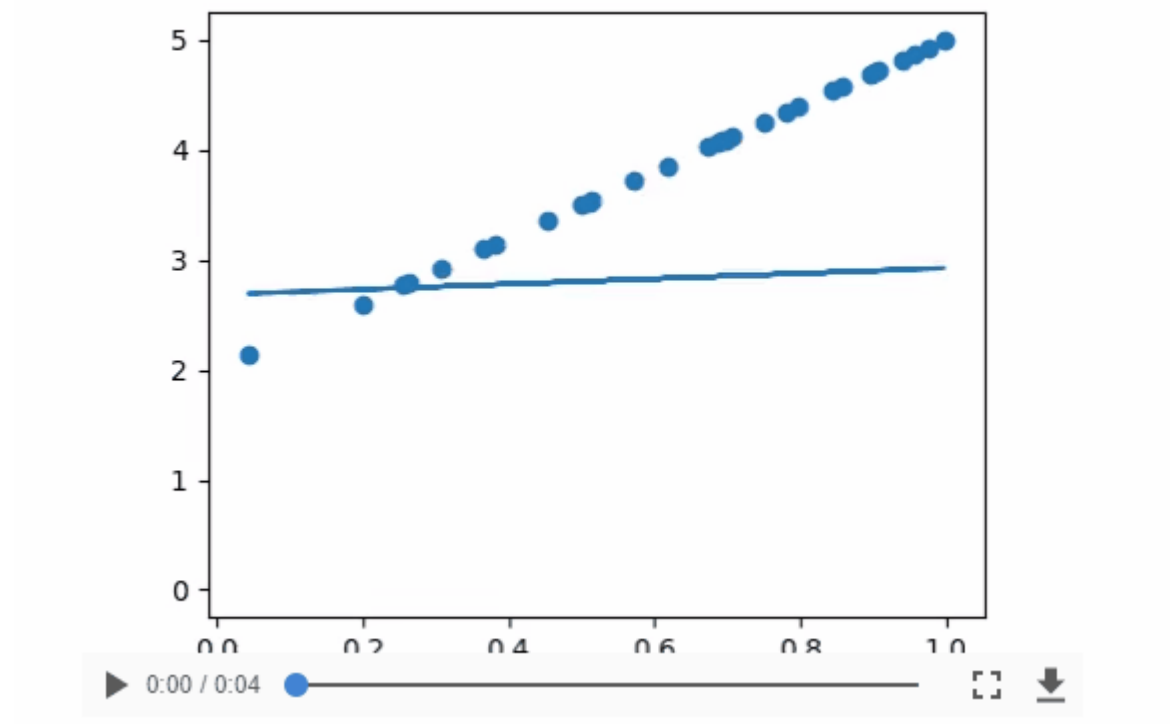


# Learning rate

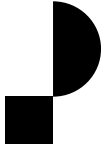




# Stochastic Gradient Descent







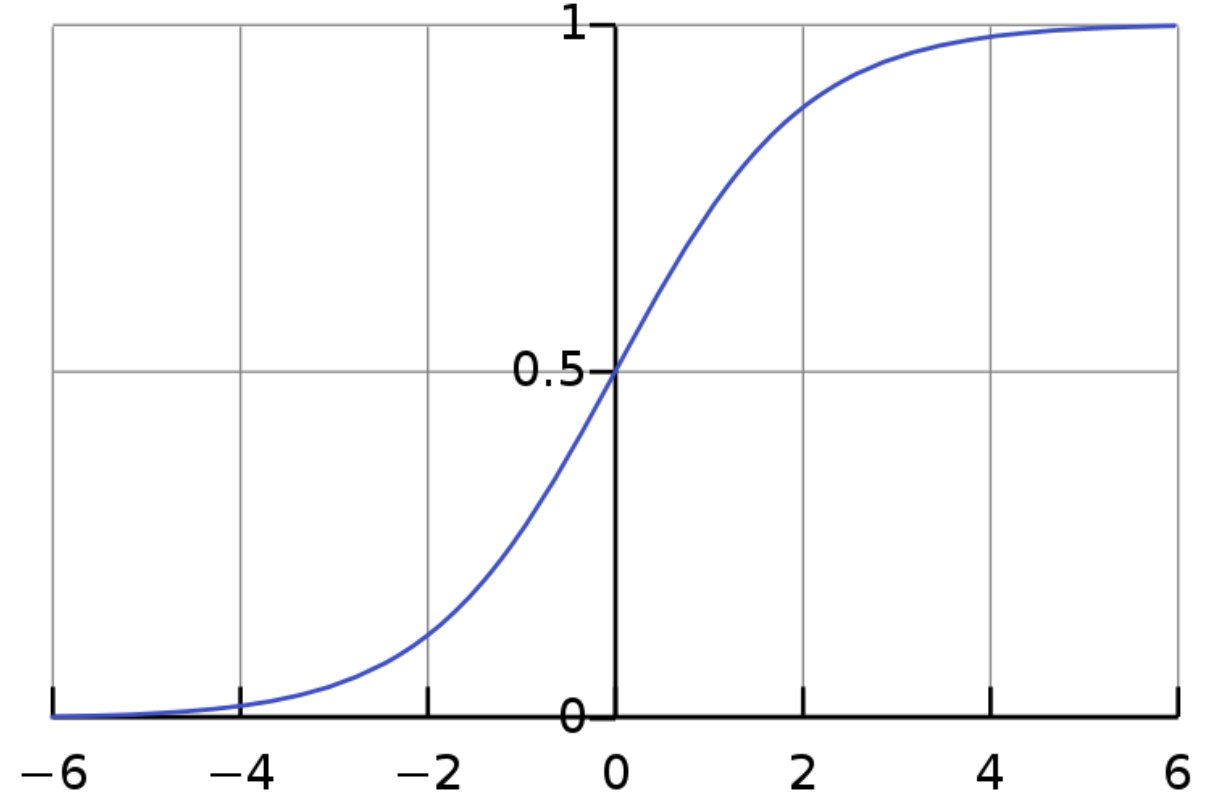
# Logistic Regression

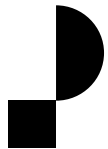
$$y = \frac{1}{1 + e^{-(ax+b)}}$$

*where y is now a probability*

Threshold:

$$class = \begin{cases} true & \text{for } y \geq 0.5 \\ false & \text{for } y < 0.5 \end{cases}$$



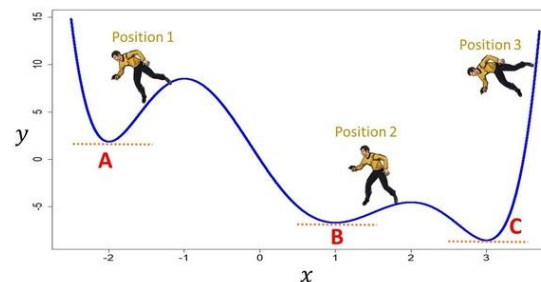


# Recap

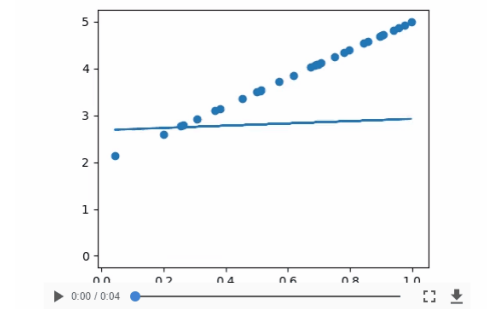
- Function with parameters (a, b) that we want to learn:
  - E.g.

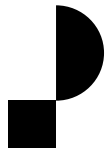
$$y = ax + b$$

- We want to select these parameters such that a loss function is minimised

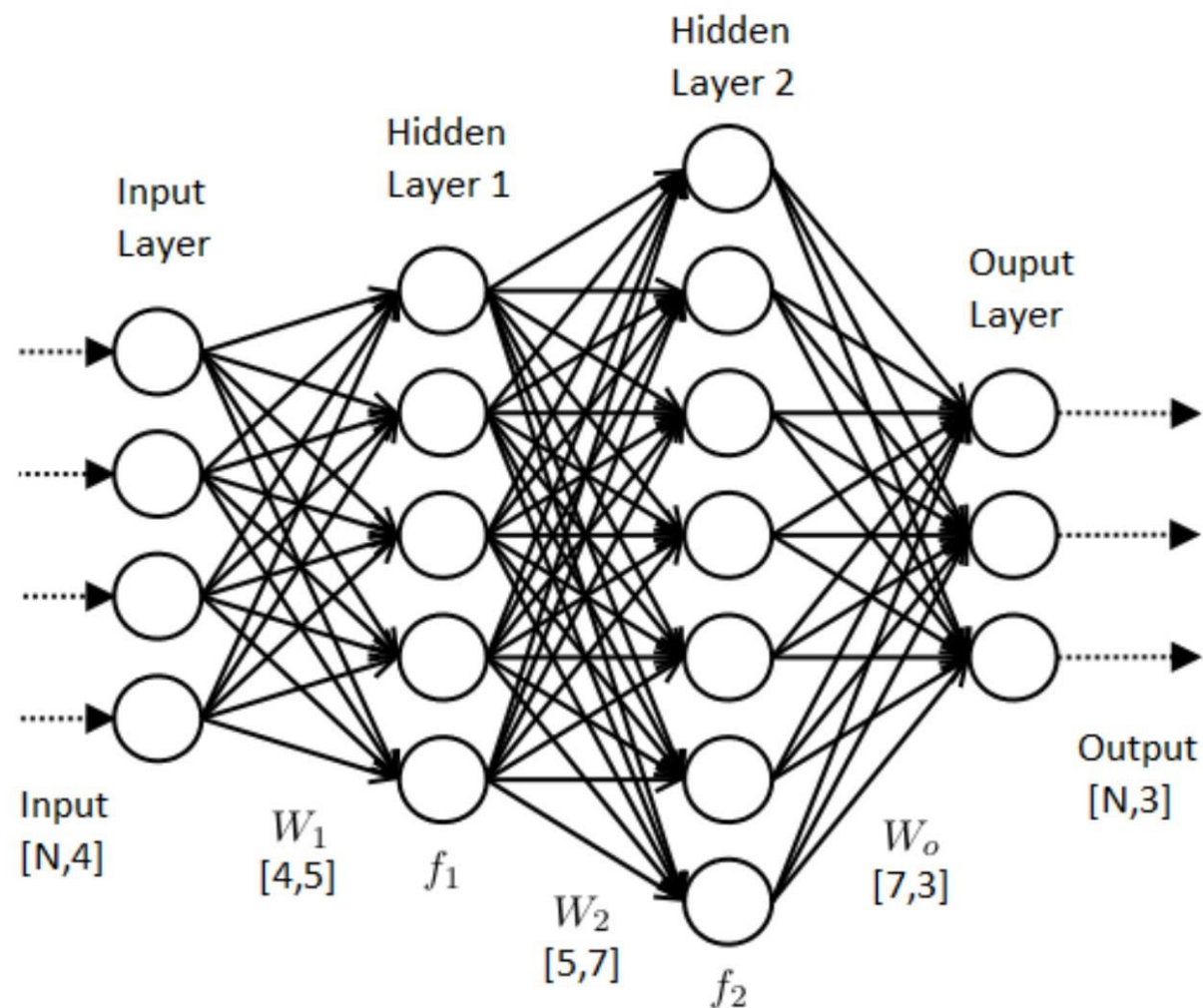


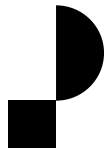
- We minimise our loss function using gradient descent or stochastic gradient descent



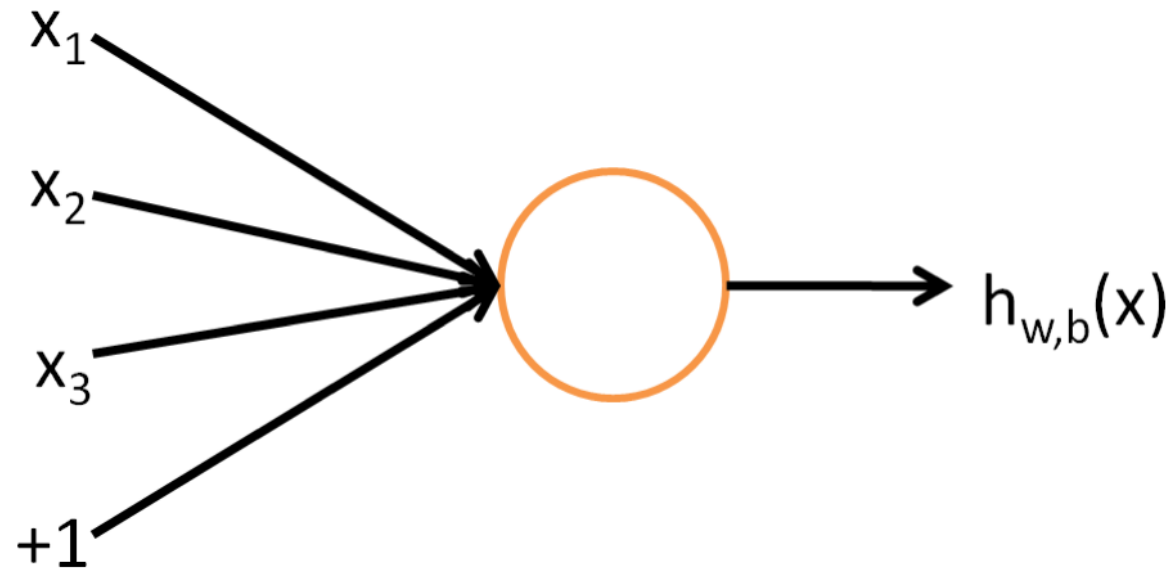
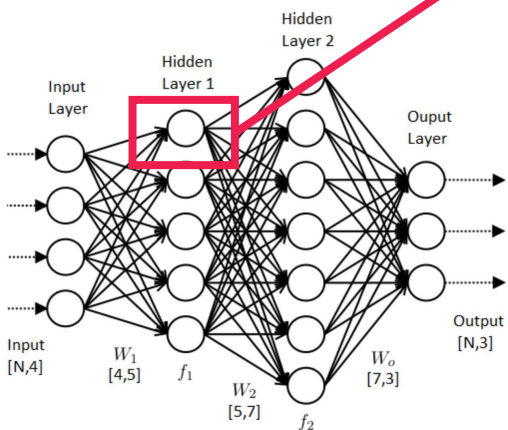


# Neural Networks

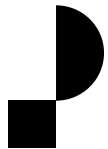




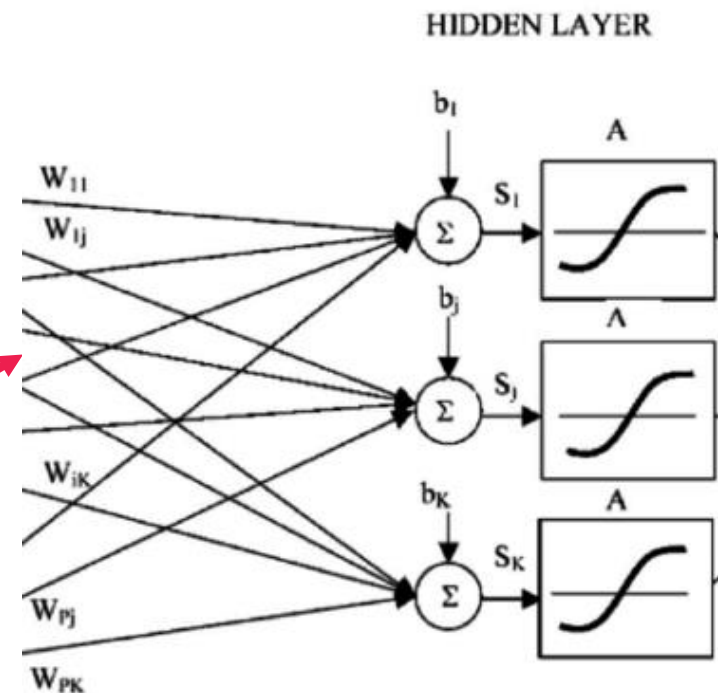
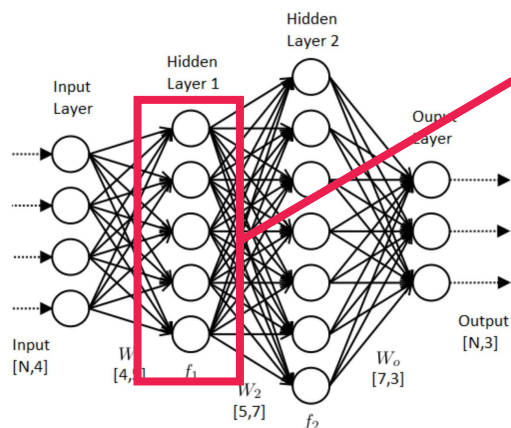
# Neurons



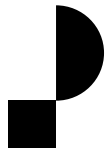
- Placeholder for a math function
- Job is to provide an output by applying a function to an input



# Layers





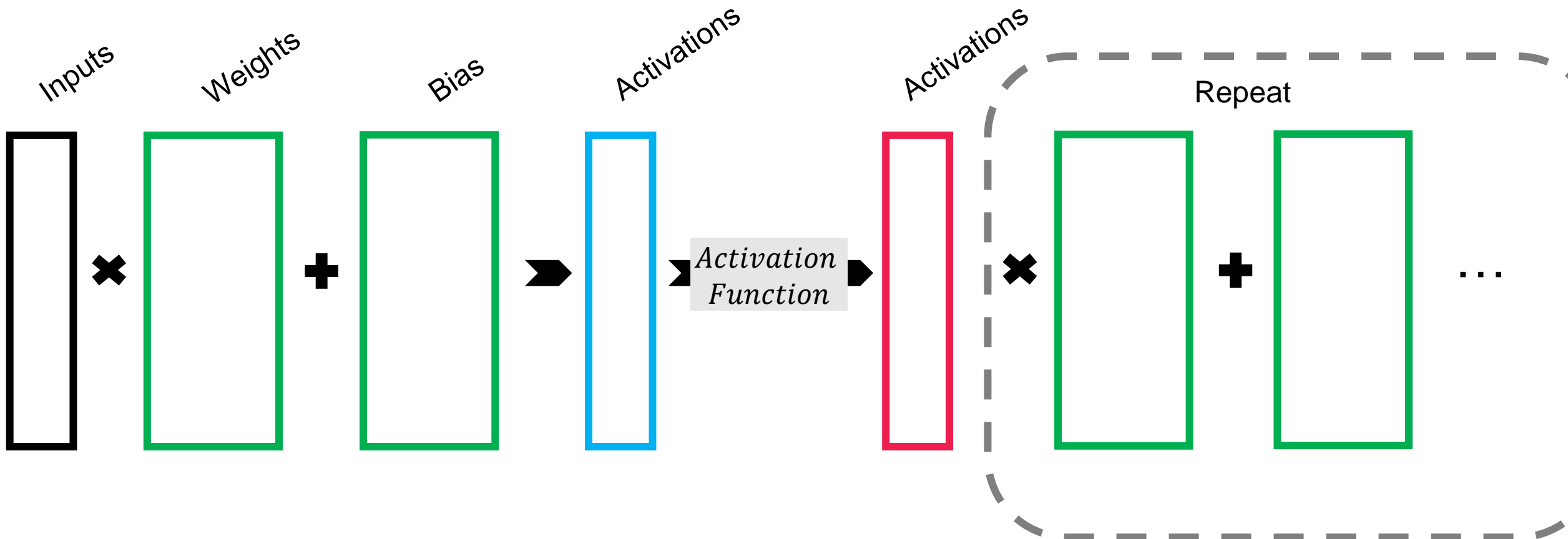
- Collection of neurons
- Where we learn **parameters**
  - Also called the **weights** and **biases**
- Where we have our **activation function**
  - Also called **non-linearities**
  - The results of these are called **activations**

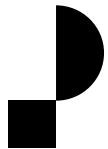


# Breaking it down further

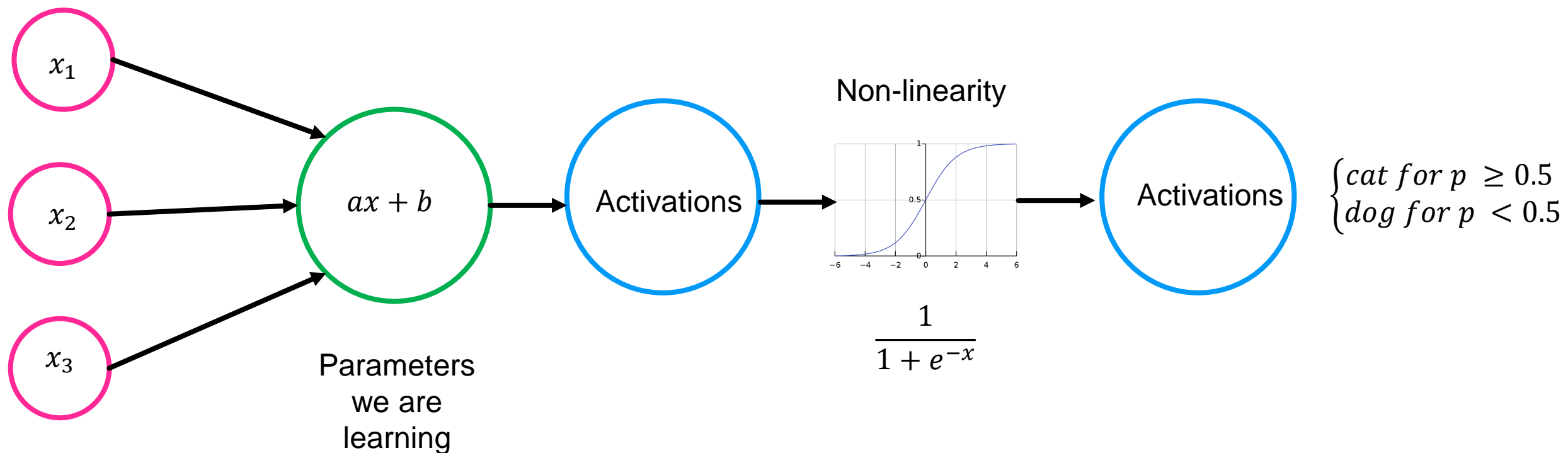
 Parameters – what we learn and store

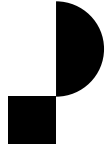
  Activations – the result of a calculation (we do not store these)





# Logistic Regression Revisited

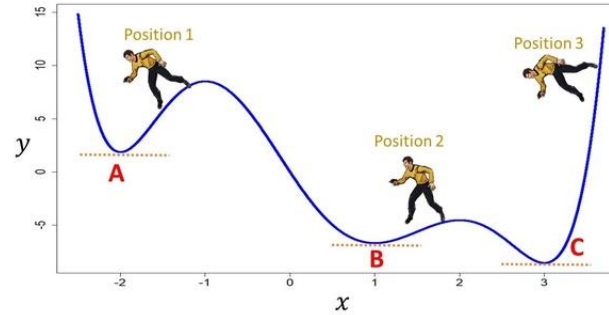
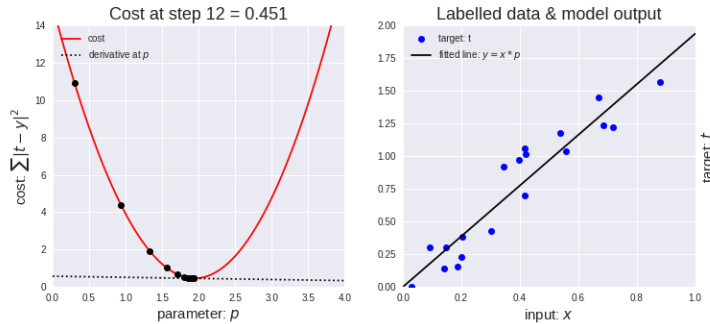




# Gradient Descent & Stochastic Gradient Descent

## Recap

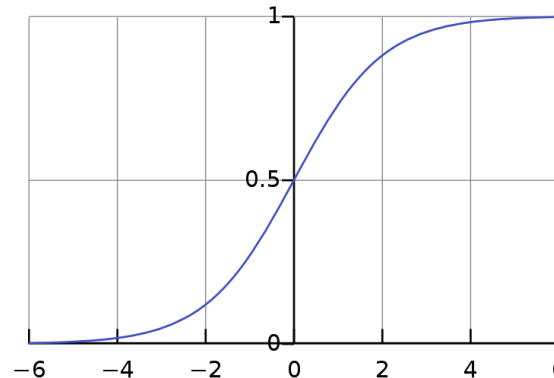
### Linear Regression



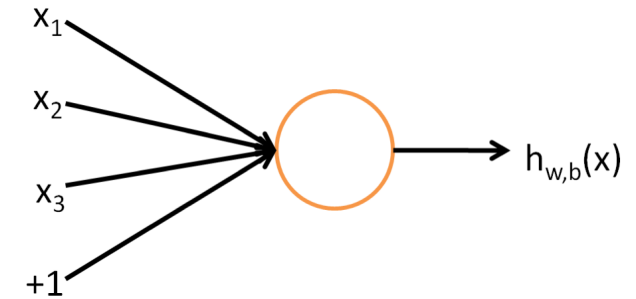
$$f = ax + b$$

*Learn parameters (or weights and biases) using a loss function*

### Logistic Regression

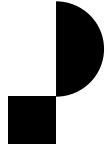


### Neural Network



- *Consists of many layers*
- *Each layer has neurons*
- *We learn parameters at each layer and apply an activation function*





# What is transfer learning?

Take a model trained on a similar task, keep what it has learned about that task and re-train it to the new task.

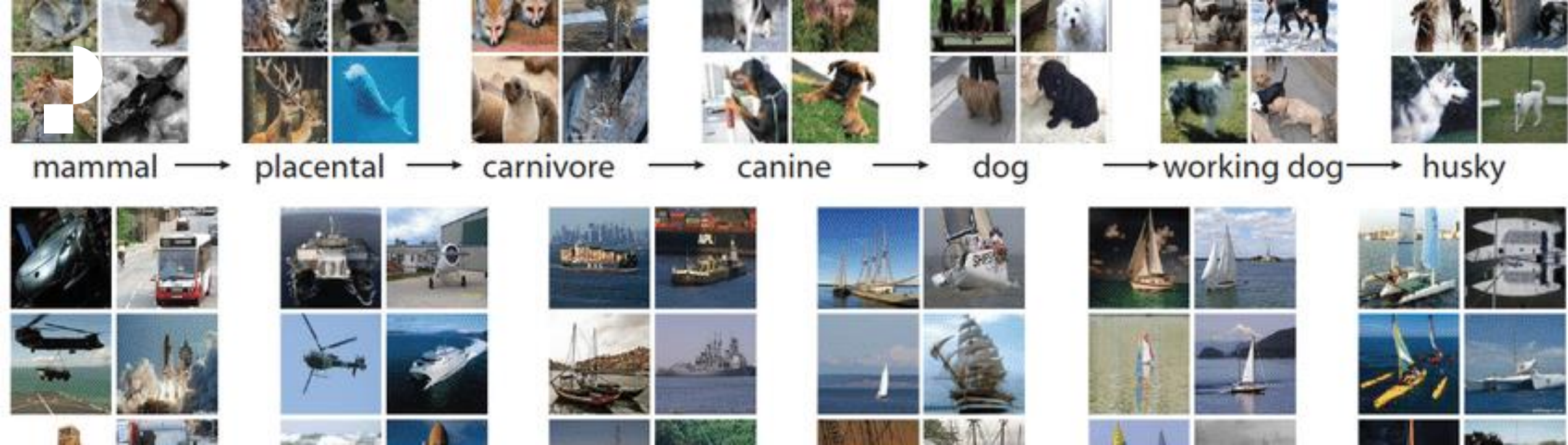
## Benefits:

- Time savings
- Cost savings
- Increased performance for smaller datasets (!!!)
  - Highly accurate deep learning models are usually trained on millions of samples.

## Limitations:

- Inherent bias based on dataset used for training





# ImageNet and ResNet34

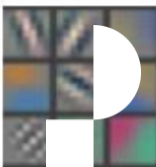
## ImageNet

- Dataset of photos of the natural world
- 1000 image categories

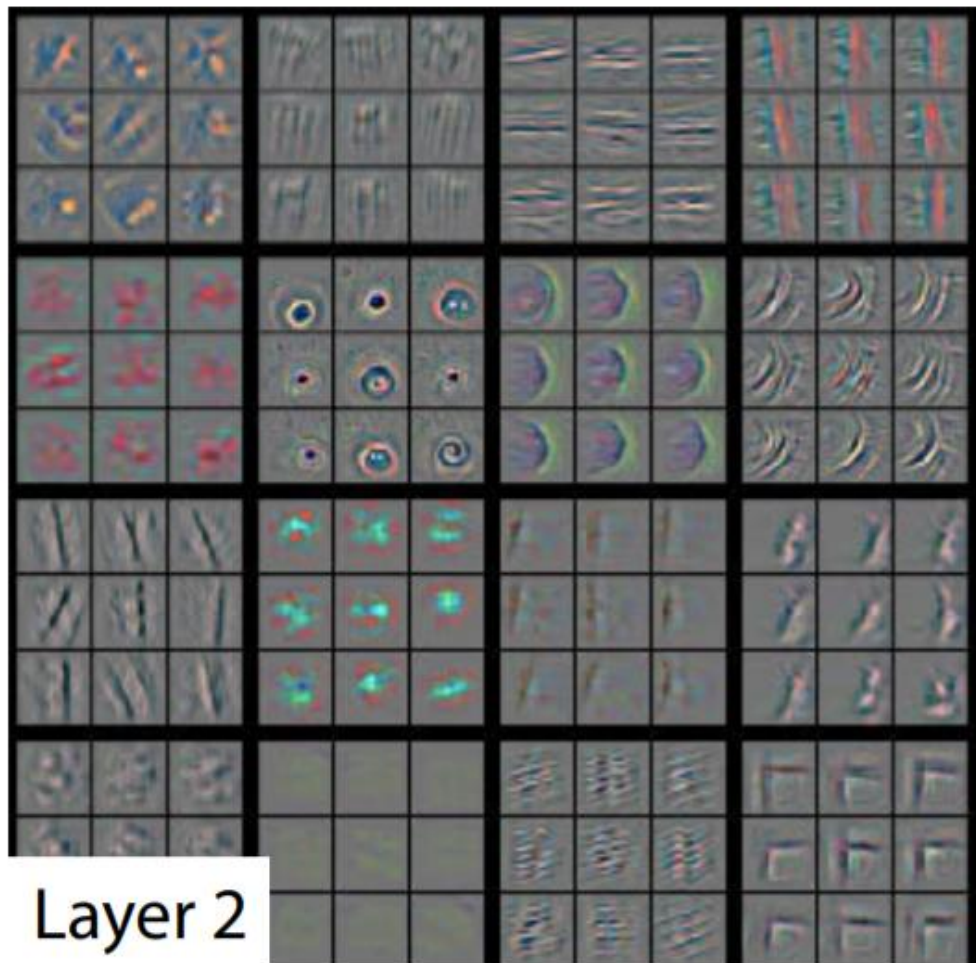
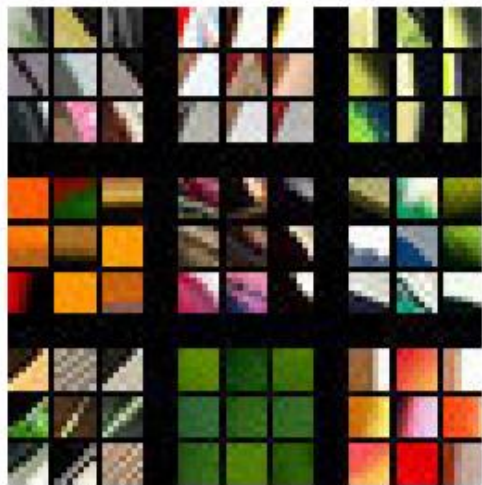
## ResNet34

- Type of neural network
  - Specifically convolutional neural network with residual connections
- Has 34 layers
- Solves vanishing gradient problem

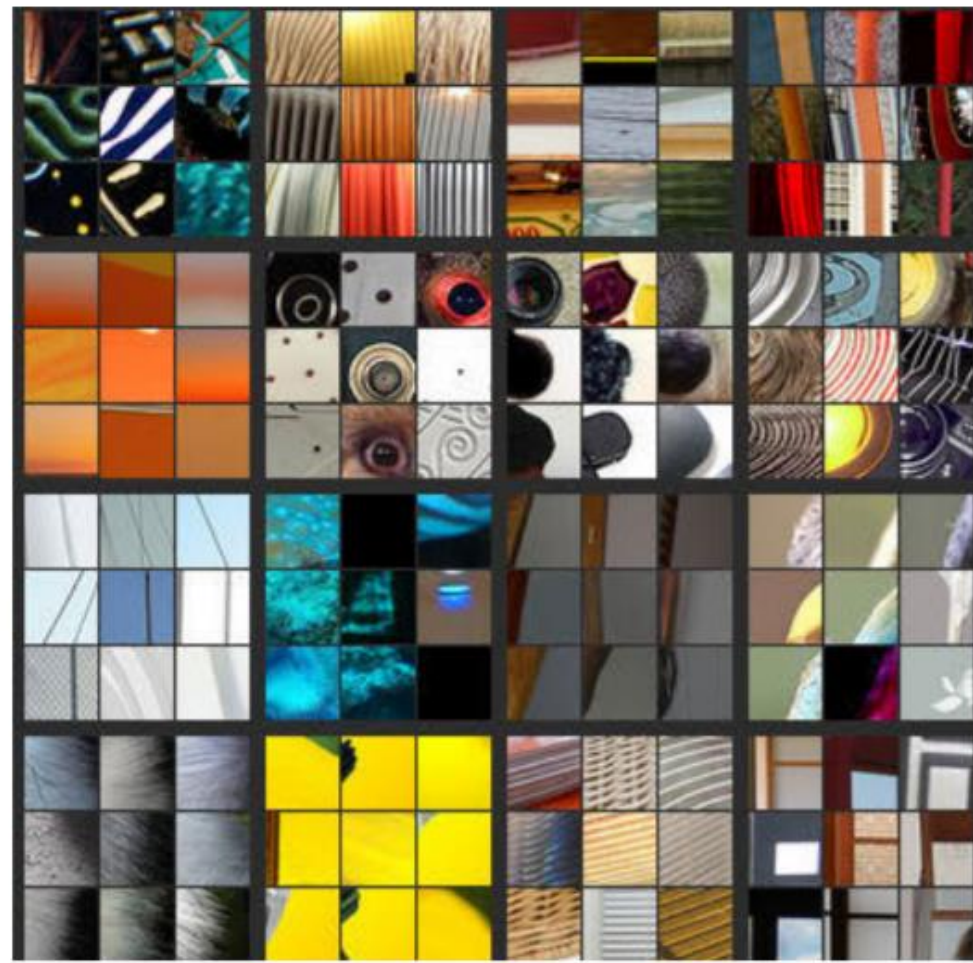




Layer 1

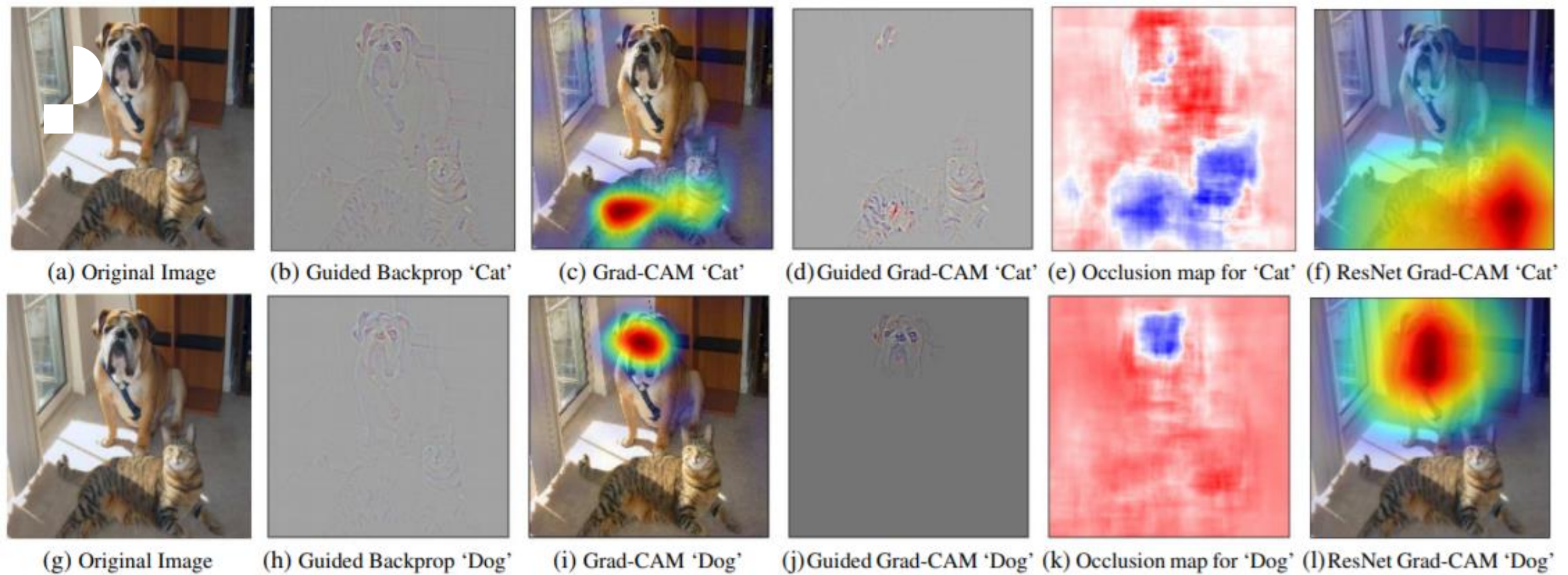


Layer 2



# Visualising ConvNets

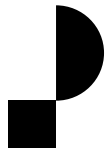
<https://arxiv.org/pdf/1311.2901.pdf>



# Visual Explanations from Deep Networks...

[http://openaccess.thecvf.com/content\\_ICCV\\_2017/papers/Selvaraju\\_Grad-CAM\\_Visual\\_Explanations\\_ICCV\\_2017\\_paper.pdf](http://openaccess.thecvf.com/content_ICCV_2017/papers/Selvaraju_Grad-CAM_Visual_Explanations_ICCV_2017_paper.pdf)





# Fine Tuning, Freezing & Unfreezing

## 1. Freeze all the layers in the network (parameters) except for the last layer

- The last layer is a weight matrix specific to the original task, not relevant for transfer learning
- We train this layer first

## 2. Unfreeze the remaining layers when things look good

- Train the entire network
- Use discriminative learning rates
  - Recall the visualisations from earlier:
    - Earlier layers have more general patterns
    - Later layers have more finer details specific to the task

```
learn.fit_one_cycle(4)
```

```
learn.save('stage-1')
```

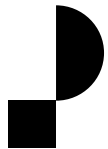
```
learn.unfreeze()
```

```
learn.lr_find(start_lr=1e-7, end_lr=1)
```

```
learn.recorder.plot()
```

```
# Update the max_lr values based on your plot above  
learn.fit_one_cycle(2, max_lr=slice(1e-6, 1e-4))
```

```
learn.save('stage-2')
```

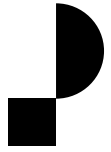


# Confusion Matrix

A confusion matrix shows us what the predicted class was against what the actual class was. The true class makes up the rows or the vertical axes and the predicted class makes up the columns or the horizontal axis.

Any entries in the diagonal of the matrix are those that are correctly classified.

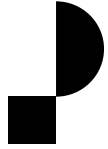
		Predicted class	
		$P$	$N$
Actual Class	$P$	True Positives (TP)	False Negatives (FN)
	$N$	False Positives (FP)	True Negatives (TN)



# Accuracy

$$Accuracy = \frac{TP + FP}{TP + FP + TN + FN}$$

		Predicted class	
		<i>P</i>	<i>N</i>
Actual Class	<i>P</i>	True Positives (TP)	False Negatives (FN)
	<i>N</i>	False Positives (FP)	True Negatives (TN)



# Activation Function and Loss Function

At the output you want to use a special activation function. This activation function is called Softmax. Softmax is where:

- All of the activations add up to 1
- All of the activations are greater than 0
- All of the activations are less than 1

$$\text{softmax} = \frac{e^x}{\sum e^x}$$

Where  $x$  are our activations

For classification, we generally use a loss function referred to as *categorical cross entropy* or *log loss*.

$$\sum_{C=1}^C t_{o,c} \log(p_{o,c})$$

$C$  = class

$t$  = actual value

$p$  = predicted value



