# Introduction to Neural Networks

# Course Outline

- This course is provides an introduction to the foundations of deep learning for more advanced modules such as computer vision. By the end of this course, students will have a firm understanding of the concepts of neural network such as neural network architectures, feed-forward networks, backpropagation, and dropout.
- Examples of simple Artificial Neural Networks will be applied to topics covered in classical machine learning to compare and contrast the performance of the two different approaches.

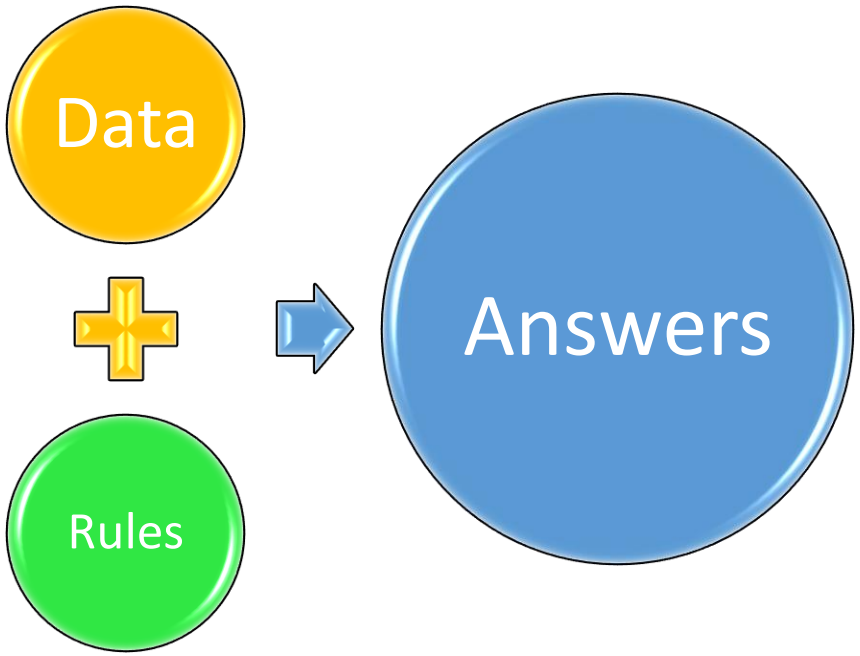
# Topics Covered

- Neural network architectures
- Feed-forward networks
- Backpropagation
- Dropout
- Artificial Neural Networks vs Classical Machine Learning

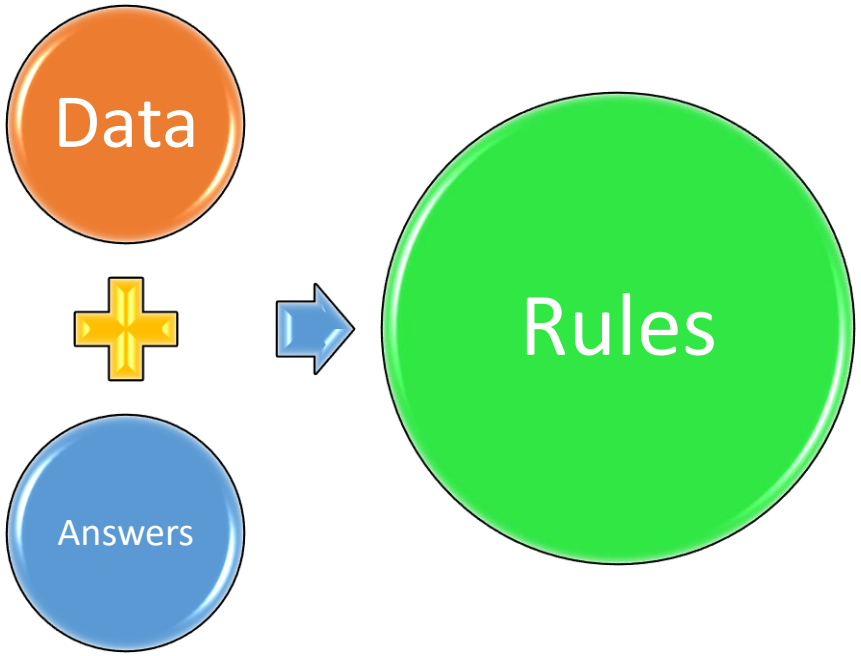
# What exactly is Deep Learning?

# Difference Between Classical Programming and Machine Learning

**Classical Programming**




**Machine Learning**




Source: Chollet, Deep Learning with Python

# Learning representations from data

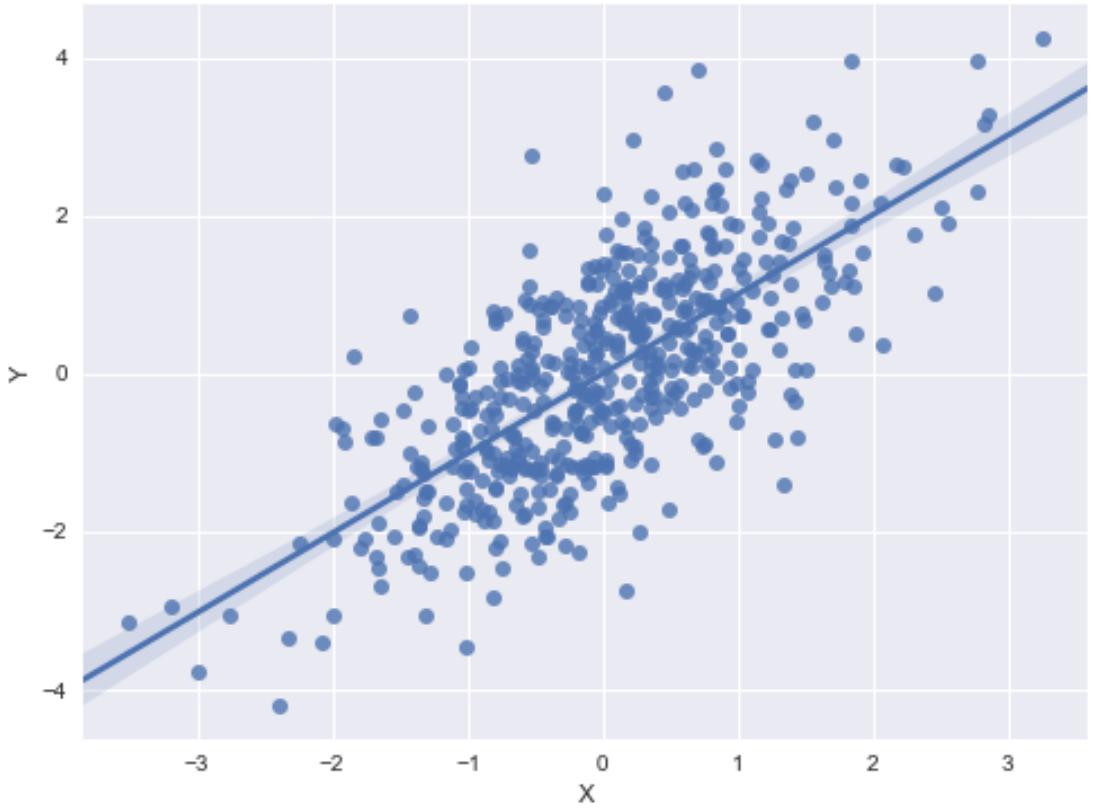
- 3 basic ingredients needed
  - Training data
  - Target or examples of expected output
  - Performance measurement, a way to check the accuracy of the model
    - Provides feedback to make adjustment (aka learning) to the model

A machine-learning model transforms its input data into meaningful outputs, a process that is “learned” from exposure to known examples of inputs and outputs. Therefore, the central problem in machine learning and deep learning is to *meaningfully transform data*: in other words, to learn useful *representations* of the input data at hand—representations that get us closer to the expected output.

~ Chollet, 2018

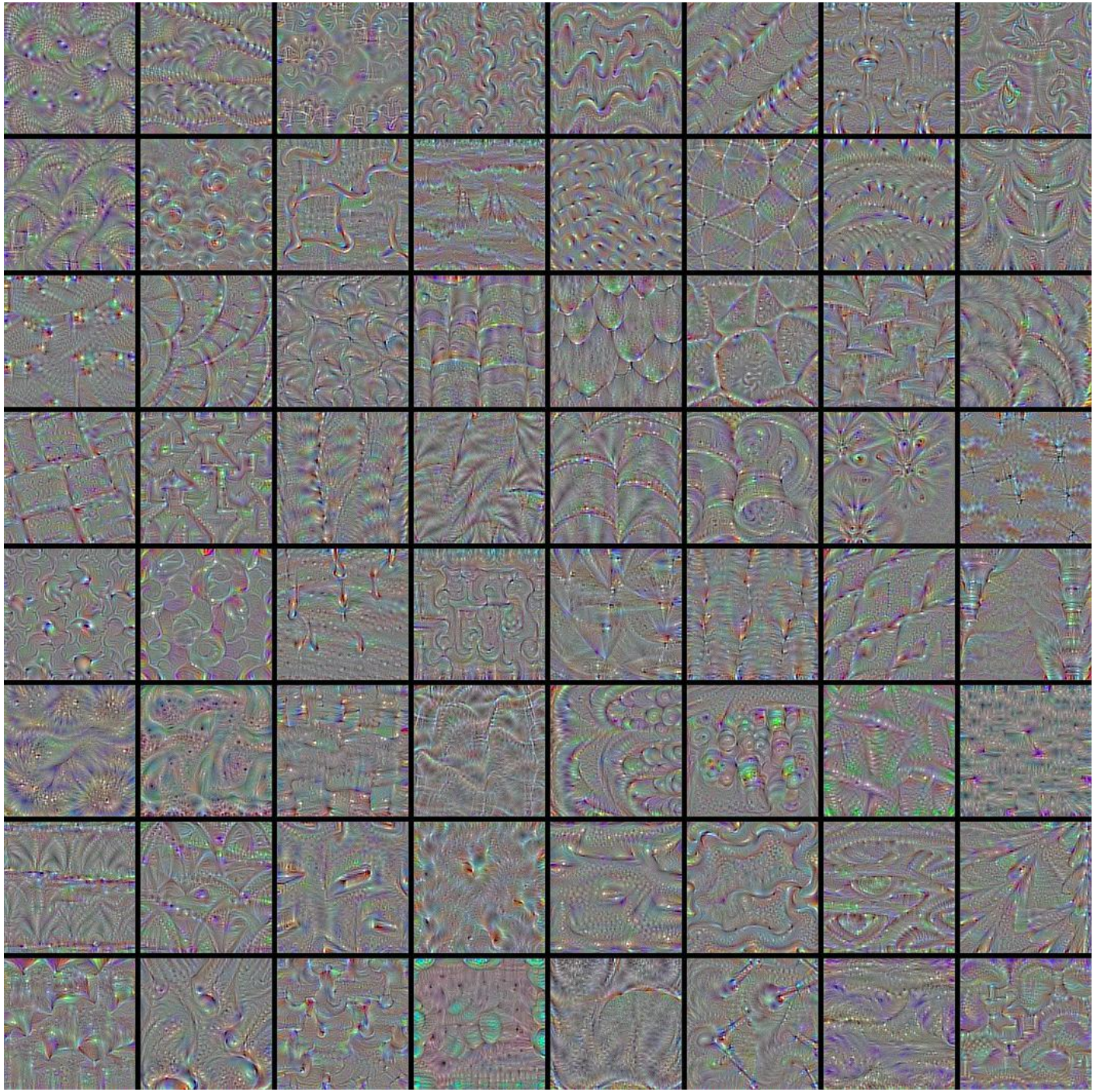
# Example of Representations: Regression

Notice how the the straight line generalized the relationship between X and Y



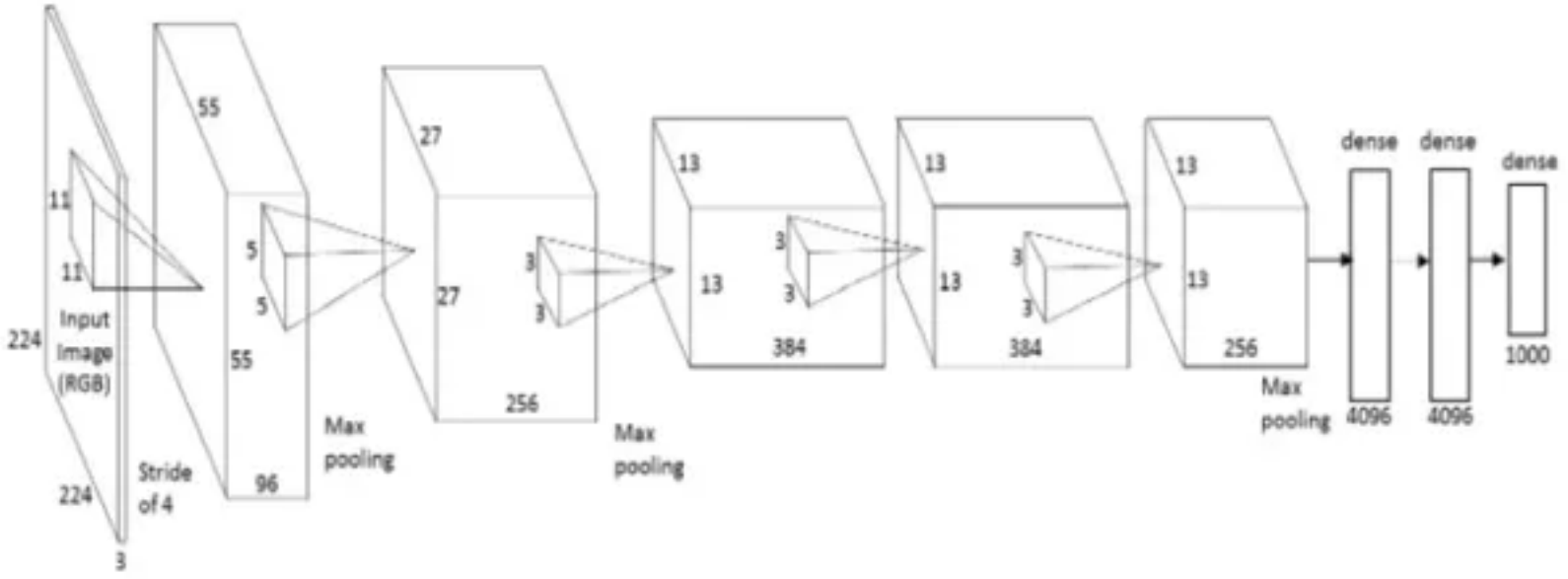
# Deep Learning – CNN

How deep learning neural network sees the world



# What is Deep Learning

- Not deep in the sense of knowledge
- Deep refers to the depth of layers
- Each layer is learn a specific representation of the data
- Eg., AlexNet



# VGG16

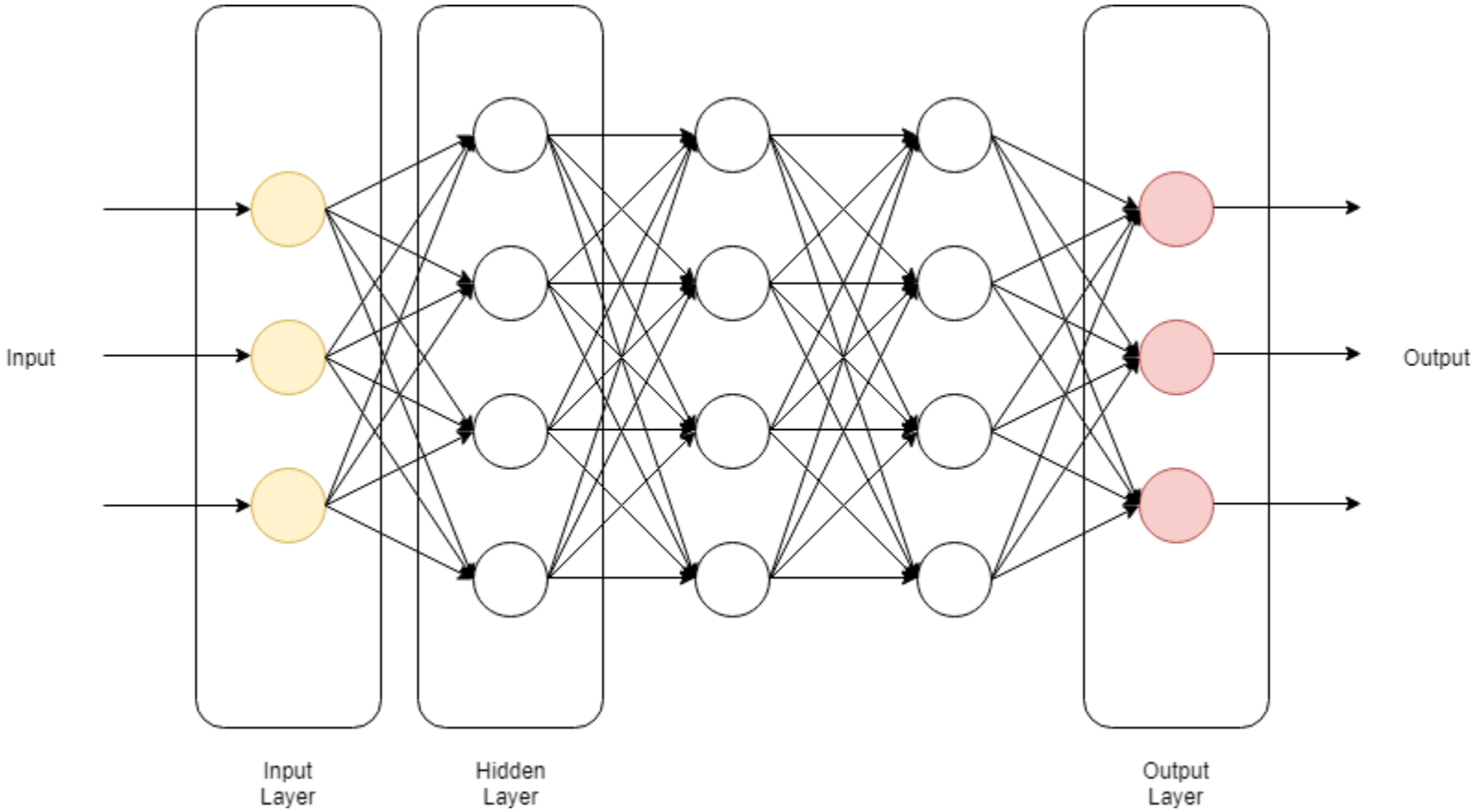
- Example of a NN architecture
- VGG16

- NB:
  - Fully connected layer not shown

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_1 (InputLayer) | (None, None, None, 3) | 0 |
| block1_conv1 (Conv2D) | (None, None, None, 64) | 1792 |
| block1_conv2 (Conv2D) | (None, None, None, 64) | 36928 |
| block1_pool (MaxPooling2D) | (None, None, None, 64) | 0 |
| block2_conv1 (Conv2D) | (None, None, None, 128) | 73856 |
| block2_conv2 (Conv2D) | (None, None, None, 128) | 147584 |
| block2_pool (MaxPooling2D) | (None, None, None, 128) | 0 |
| block3_conv1 (Conv2D) | (None, None, None, 256) | 295168 |
| block3_conv2 (Conv2D) | (None, None, None, 256) | 590080 |
| block3_conv3 (Conv2D) | (None, None, None, 256) | 590080 |
| block3_pool (MaxPooling2D) | (None, None, None, 256) | 0 |
| block4_conv1 (Conv2D) | (None, None, None, 512) | 1180160 |
| block4_conv2 (Conv2D) | (None, None, None, 512) | 2359808 |
| block4_conv3 (Conv2D) | (None, None, None, 512) | 2359808 |
| block4_pool (MaxPooling2D) | (None, None, None, 512) | 0 |
| block5_conv1 (Conv2D) | (None, None, None, 512) | 2359808 |
| block5_conv2 (Conv2D) | (None, None, None, 512) | 2359808 |
| block5_conv3 (Conv2D) | (None, None, None, 512) | 2359808 |
| block5_pool (MaxPooling2D) | (None, None, None, 512) | 0 |

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

# Example of a Neural Network



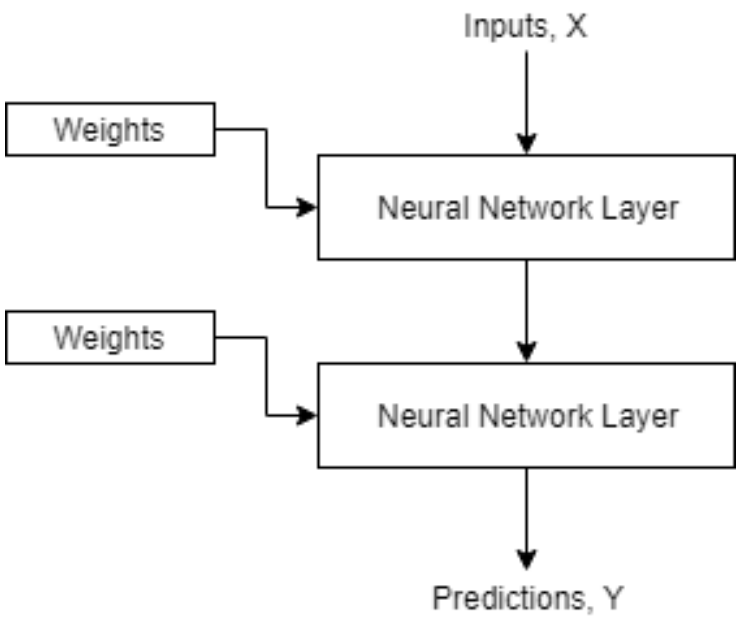
# More NN Visualisation

- 2D Convolutional Network Visualisation. Source: An Interactive Node-Link Visualization of Convolutional Neural Networks by Adam W. Harley.
- Tensor Playground
- CS231n Layer 2

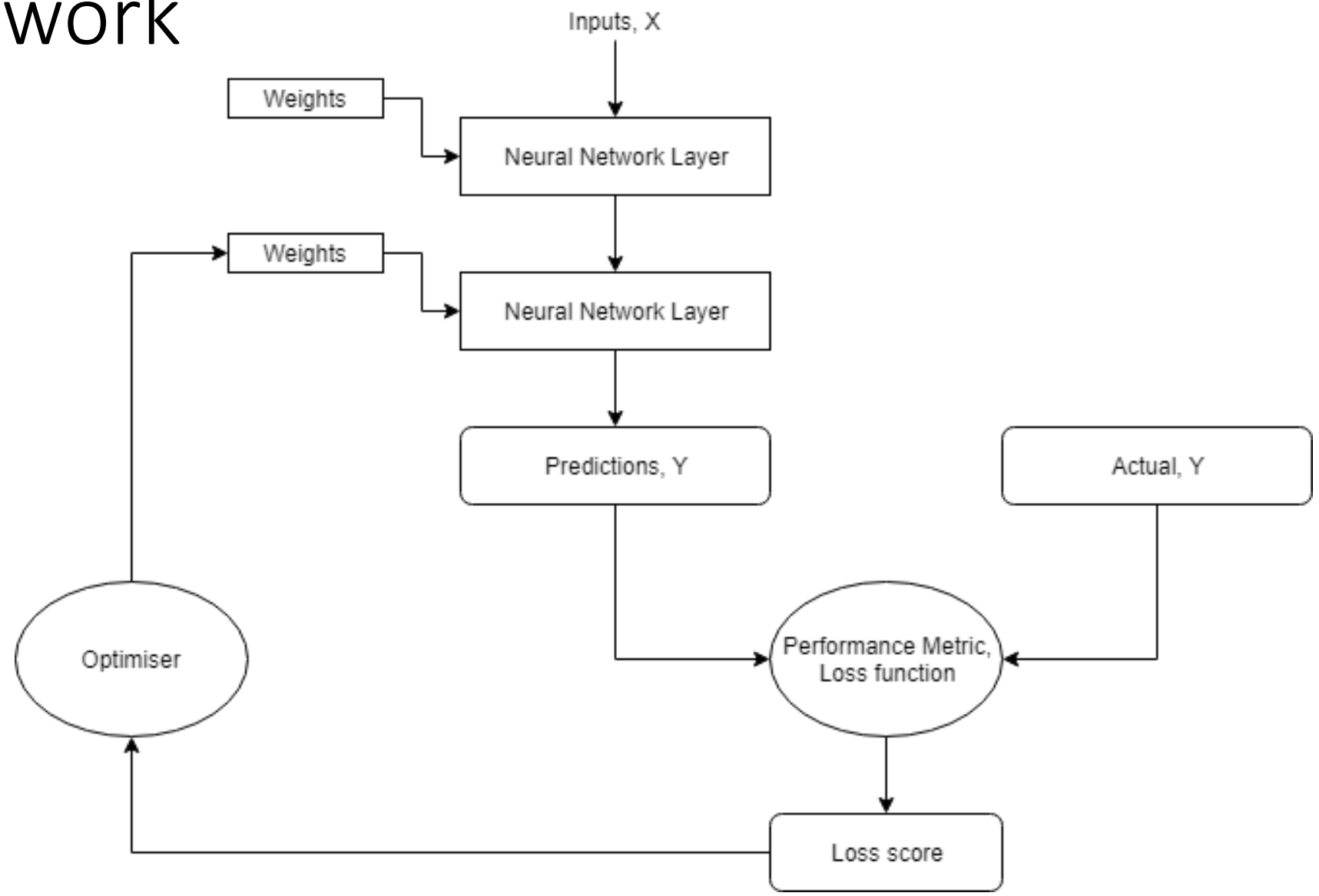
# What have we learnt so far?

- Machine learning is about
  - Input – ??? – Targets
  - Finding the ??? to link inputs to target
- Required many examples to achieve this.
- From the previous slides we can see that there are many layers to deep learning neural networks.
- These layers learn the relationship between input and target from being exposed to many examples

# Neural Network



# Neural Network



# Shallow vs Deep Learning

- Shallow ML – Learn greedily (in successive manner). Stacking multiple shallow learners together will not achieve the same result.
- Deep Learning – The layers learn the patterns jointly.

# Why now?

- Factors
  - Computational power
    - GPU
    - Cost
  - Data availability
  - Advanced in algorithm
    - Activation functions
    - Weight initialization scheme
    - Optimization scheme
  - Investments

# Building Blocks of Neural Networks

# Foundational

- Linear Algebra
- Probability and information theory
- Numerical computation / literacy

# Motivational Example

- Mnist dataset
- Steps covered:
  1. Loading data
  2. Neural Network architecture
  3. Compilation
  4. Pre-processing / Preparing the data
  5. Preparing the labels / target
  6. Train model
  7. Predict

# A Tensor is a Data Container

| Tensors | Name | Shape | Example |
|---|---|---|---|
| 0 | Scalar | | 3 |
| 1 | Vectors | | [3, 6] |
| 2 | Matrices | #samples, features | [[3, 6], [3, 6]] |
| 3 | 3D | #samples, timesteps, features | Time series data |
| 4 | 4D | #samples, height, width, channels<br>or<br>#samples, channels, height, width | Images |
| 5 | | #samples, frames, height, width, channels<br>or<br>#samples, frames, channels, height, width | Video |

# Key Attributes

- Rank (Number of Axes)
  - 3D has 3 axes
  - Python numpy – `ndim`
- Shape
  - Number of dimensions the tensor along each axis. Example:
    - Vector (2D).
      - `.ndim` = 2
      - `.shape` = (4,9)  # value depends on the individual vector.
    - 3D
      - `.ndim` = 3
      - `.shape` = (3, 3, 4)
- Data type
  - Type of data stored in the tensor container
  - Python `dtype`
    - `float32, float64, uint8`

# Data Batches

- Deep learning models don’t usually process an entire dataset at once
- The data are usually broken into small batches
- Some terminology
  - Epoch: All the data
  - Batch: A subset of the data
    - Batch of 128 in Python code would be (Assuming the variable is called data)
      - `data[:128]`
      - `data[128:256]`
      - `data[128 * n : 128 * (n+1)]`

# Let’s Revisit Tensor Operations (Algebra) in Python

- The following will be illustrated via Python codes
  - Element-wise operations
  - Broadcasting
  - Dot product (Matrix multiplication)
  - Reshaping

# Gradient Based Optimisation

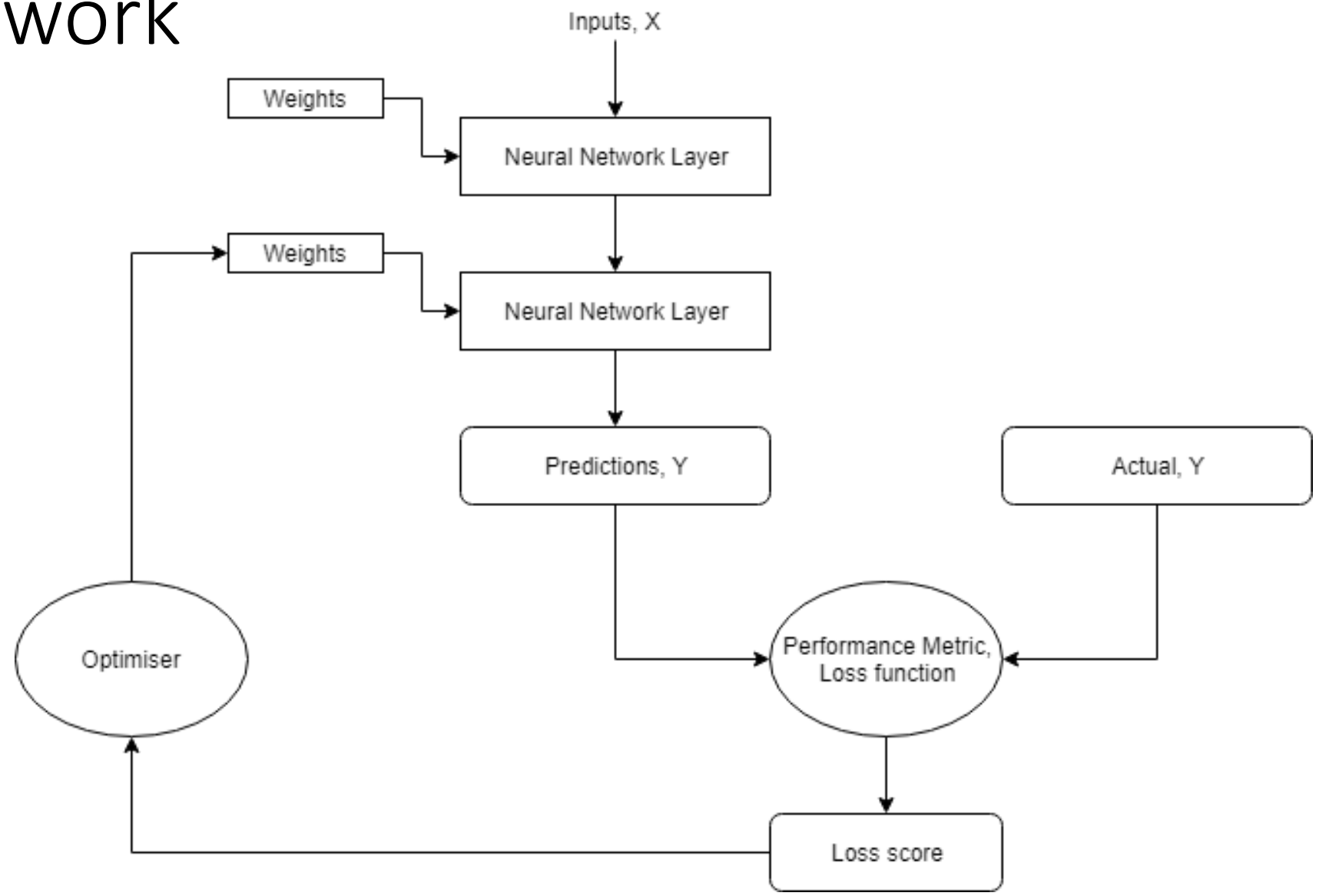
- Another word for gradient is slope
- Note that every layer end with an activation function. We can think of it as

```
output = sigmoid(dot(W, input) + b)
```

- W is the weights
- b is the bias term
- The initial weights are set randomly
- We then need to make gradual adjust / update to the weight to reduce the error (loss)

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

# Neural Network



# Training Loop

1. Draw a batch of training samples x and targets y
2. Train the NN on x to obtain prediction of y
3. Compute the loss (error) of the NN on the batch
4. Compute the gradient of the loss w.r.t. to the NN parameters (backward pass)
5. Move the parameters a little in the opposite direction from the gradient. Taking one small step to reduce the loss on the batch.
   - $w_1 = w_0 - \text{step} * \text{gradient}$

# Optimiser: Stochastics Gradient Descent (SGD)

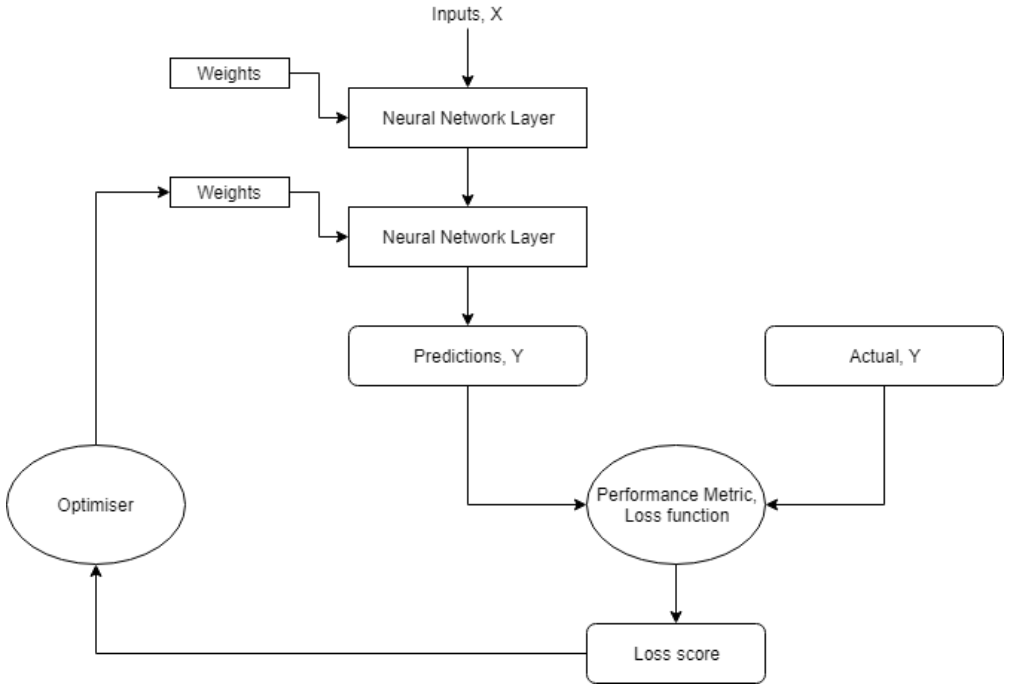
- SGD: One sample at a time
- Mini-batch SGD: Batch of data or a sample of the data. Common batch size 128, 512 etc.
- Batch SGD: All the data

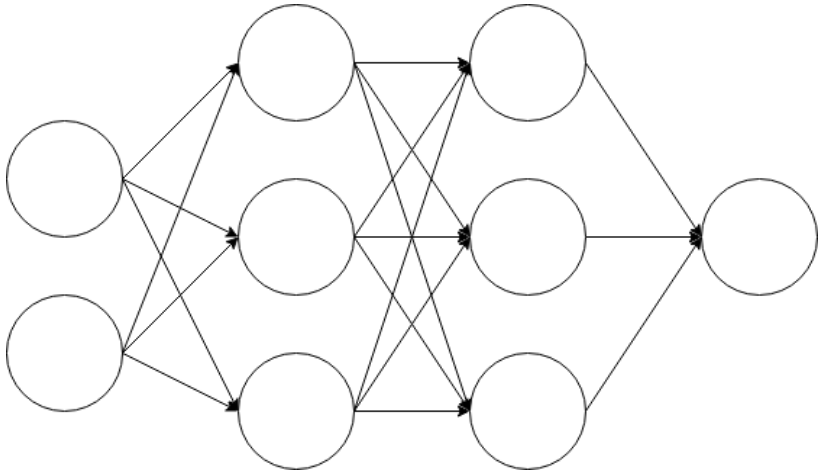
- Other more sophisticated techniques include:
  - SGD with momentum
  - Adagrad
  - RMSProp
  - Etc.

# Getting Started with NN

# Anatomy of a NN

- Input data, X
- Target labels, Y
- Layers, when combined is called network
- Loss (error or the objective) function
- Optimiser (e.g., SGD, etc.)

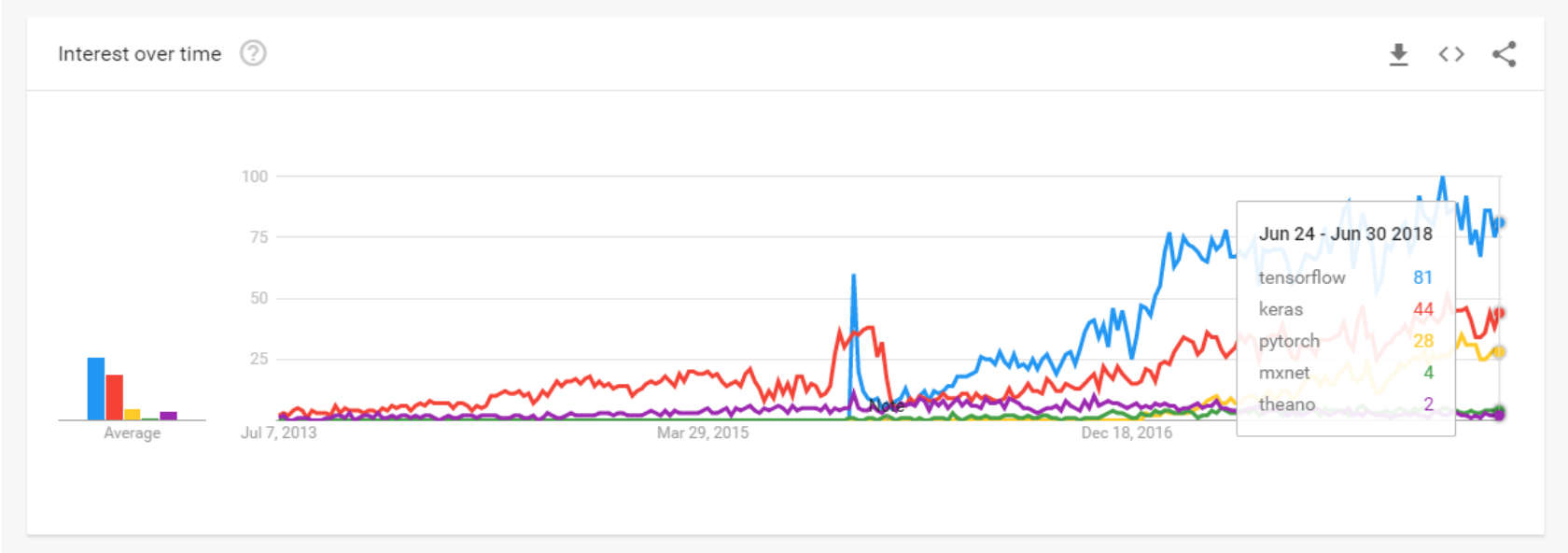


# Deep Learning Libraries



- Tensorflow
- Keras
- PyTorch
- Caffe
- CNTK
- Apache Mxnet
- Theano




- https://www.kdnuggets.com/2018/04/top-16-open-source-deep-learning-libraries.html

# Keras + Tensorflow

- Workflow
  - Define your training data: input X and target Y
  - Define your network architecture
  - Select the appropriate loss function, optimiser and performance metric
  - Repeat until a pre-specified performance is reached

# Examples – Binary classification

- https://keras.io/getting-started/sequential-model-guide/#examples

# Examples – Multiclass classification

- https://keras.io/getting-started/sequential-model-guide/#examples

# Examples – Regression

- https://keras.io/getting-started/sequential-model-guide/#examples

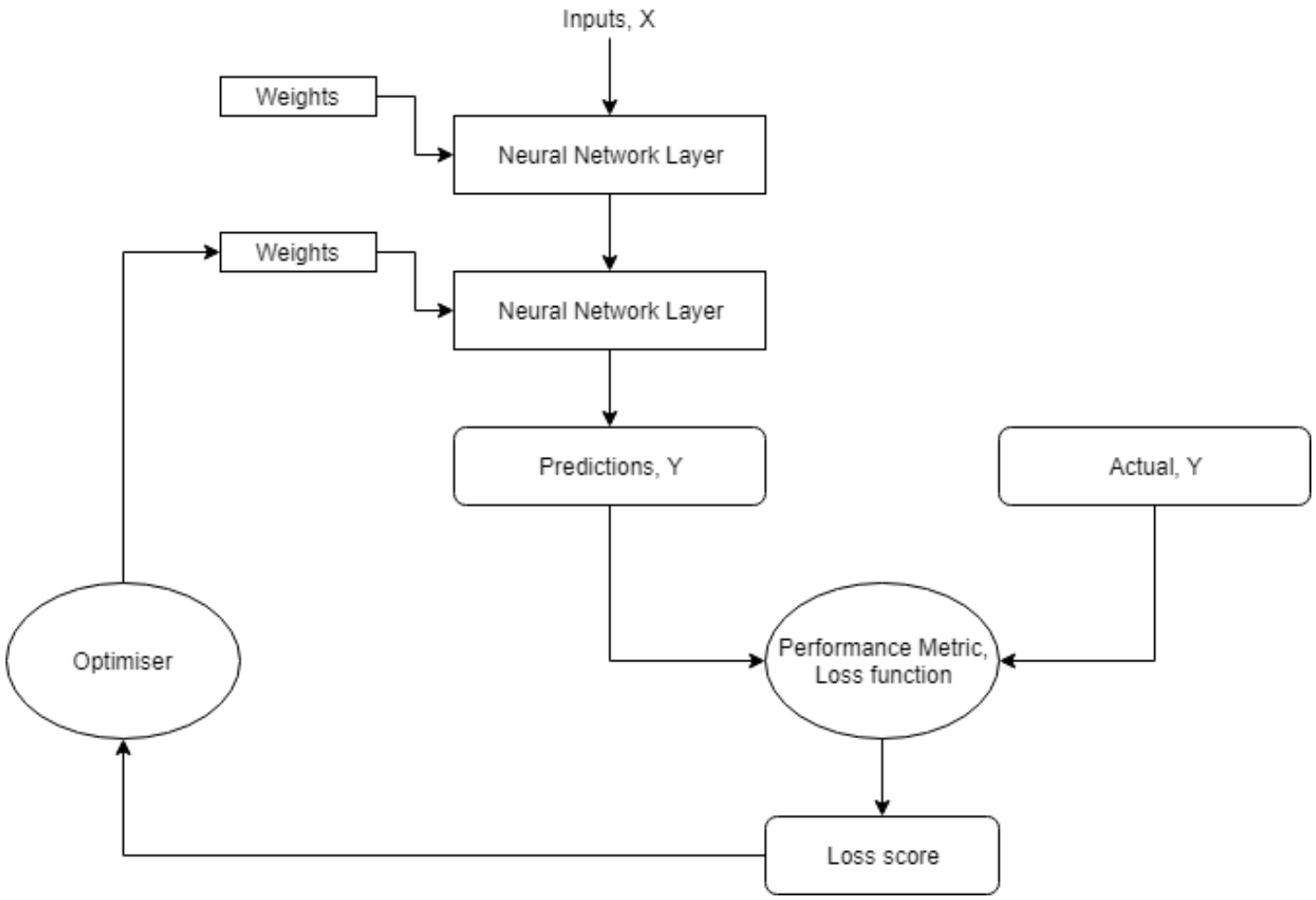
# Mechanics of Machine Learning

# Different Categories of Machine Learning

- Supervised
  - Classification and regression
  - Sequence generation
  - Object detection
  - Image segmentation
- Unsupervised
  - Dimensionality reduction and clustering
- Self-supervised
  - Autoencoders
- Reinforcement Learning

# Machine Learning Performance Evaluation

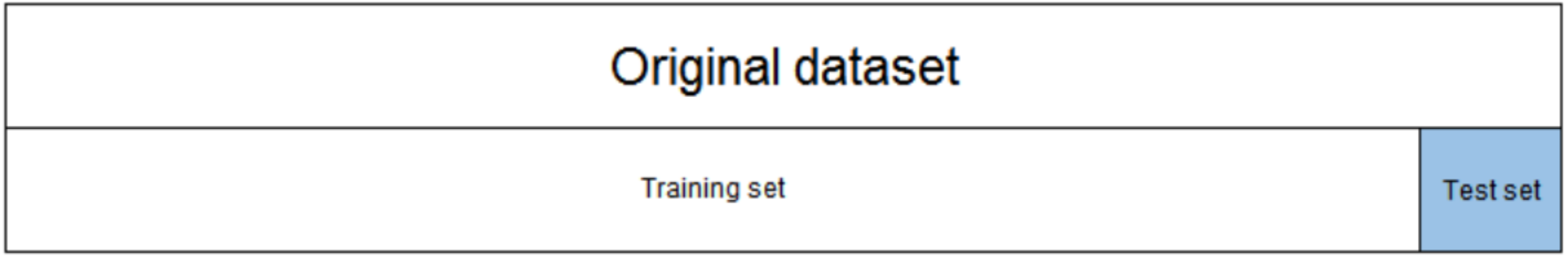
- Goal of machine learning: Model able to Generalise
- What is a loss function
- Loss functions
  - Binary
  - Multiclass



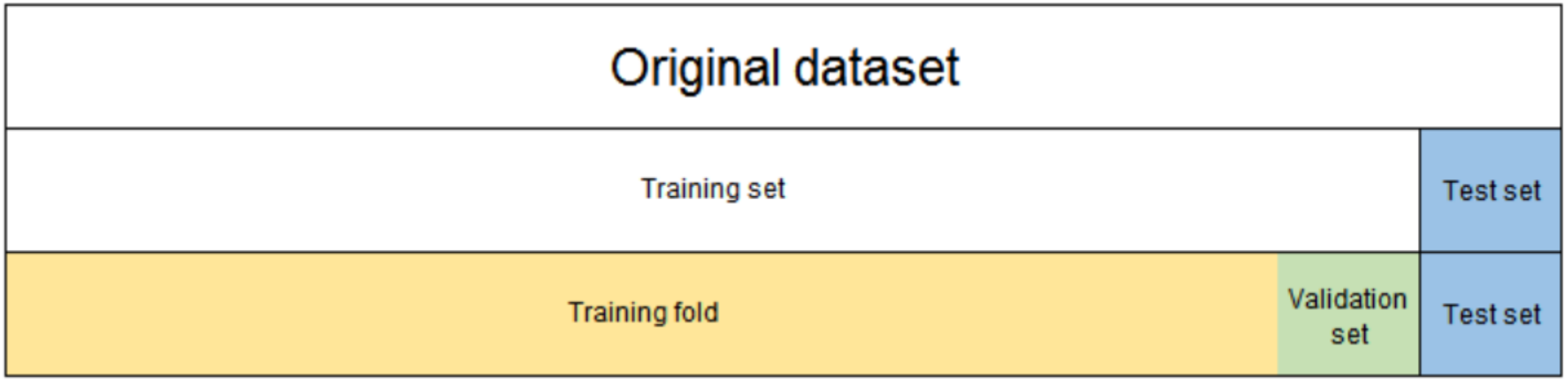
# Train, Validate and Test

- Test dataset is meant to be used only once and only at the end of model building
  - This is to prevent information leak
- Train dataset is used for training the model
  - Your model would have seen these dataset many times
  - No way to tell if it overfitted and can generalise to unseen data
- Validate dataset is used to evaluate model performance w.r.t to various hyperparameters

# Single Hold-out Test Set



# Single Hold-out Validation and Test Set



# K-Fold Validation. 10-fold Example Shown

# Mindful of Traps

- Data representativeness
  - E.g., MNIST dataset. Make sure all the digits are represented
- Time series data
  - Never ever shuffle the data and change the sequence
  - Ensure all test data are posterior (future) to the data in the training set
- Data redundancy
  - Ensure the same data is not repeated in both the train and test data
  - Eg., Same picture of a cat. This is basically cheating. Like a teacher giving student sample exam questions and testing the same questions in the exam.

# Data Pre-processing, Feature Engineering

- You have seen from the examples that pre-processing are often done before model training.
- Data pre-processing and feature engineering are often domain specific
- E.g., In finance, we work with returns data instead of price data. Or we work with Earnings per share rather than the raw earning figure.
- The purpose of data pre-processing is allow the NN to learn faster and better

# Data Pre-processing

- Vectorisation
  - Convert your data into tensors
  - E.g., Converting word to integers then binary
- Normalisation
  - Converting data to 0 to 1 range
  - To speed up gradient convergence
  - Ensure all features have similar range.
  - Ideally, with mean of 0 and standard deviation of 1
- Missing values
  - Delete or in the case of lack of data convert the number to 0

# Feature Engineering

- Converting and transforming raw inputs to useful features
- E.g., Earnings 1,450,673 versus EPS 1.56
- E.g., 28-Jul-2017 versus 1 as Monday or 7 as July in a research that test for week day or month effect.
- Good features saves computational resource and time, and reduces data demand

# Overfitting and Underfitting

- There is always a tradeoff between underfitting and overfitting. The goal of training a machine model is so that it can learn from the pattern of the data. This fitting process is also called optimisation and the model will always start off being underfitted. As you start exposing your model to the training data many times, it will come to a tipping point where your model will over-learn from the training data. In this case, you are starting to go from underfitting to overfitting.
- **Underfitting** is when your model has not learnt fully from the training data. In this situation, both the training and test loss (error) score will be falling on each epoch.
- In order to improve both your training and test loss score, you can increase the number of times your model (epoch) see the data. This process is also called **optimisation**.
- **Overfitting** is when the loss score in the training and test diverge. The training loss score continue to decline while the test loss score rises. In this situation, the model is starting to overfit your training data and unable to **generalise** and will not perform well with data it has not seen before (test)

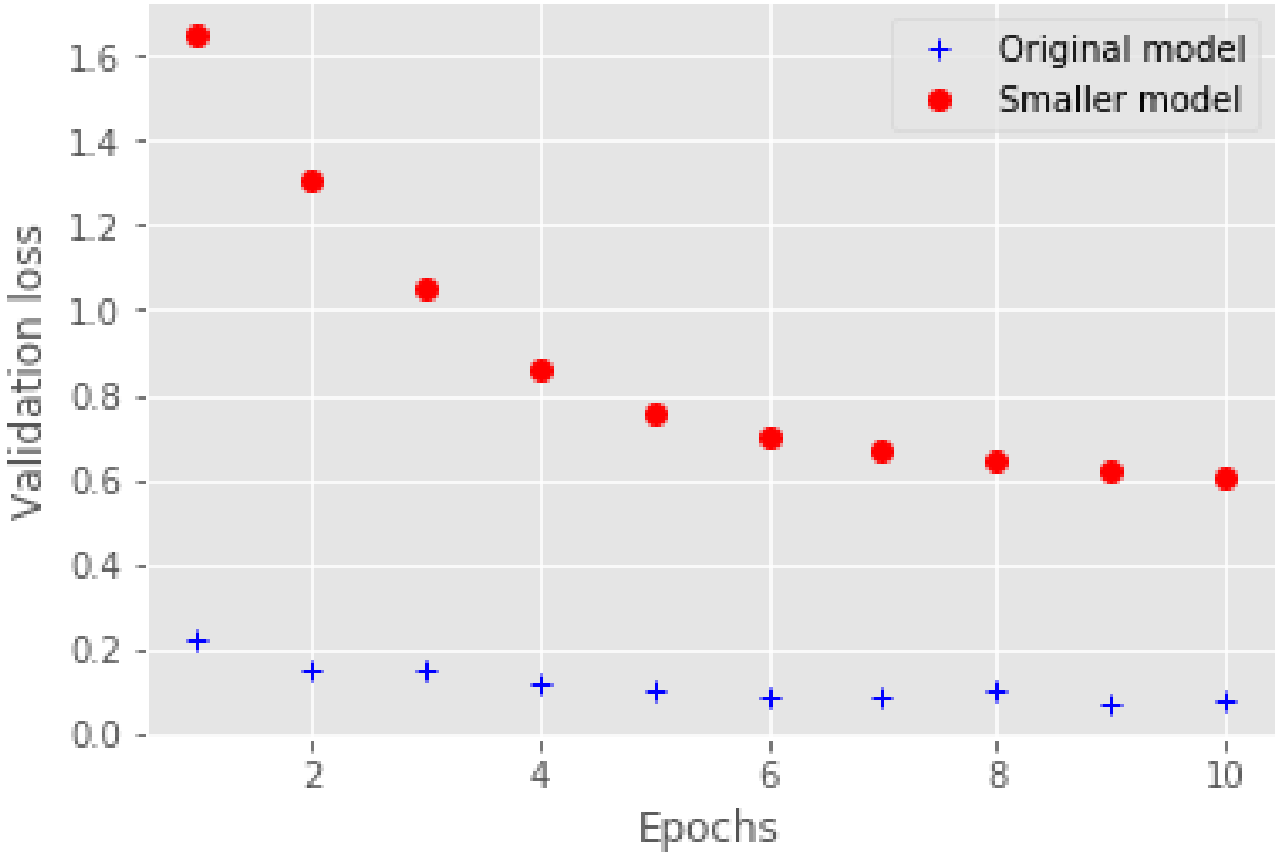
# Overcoming Overfitting

- Get more data
- Regularisation
  - Reduce  network’s size
  - Weight regularisation
  - Dropout

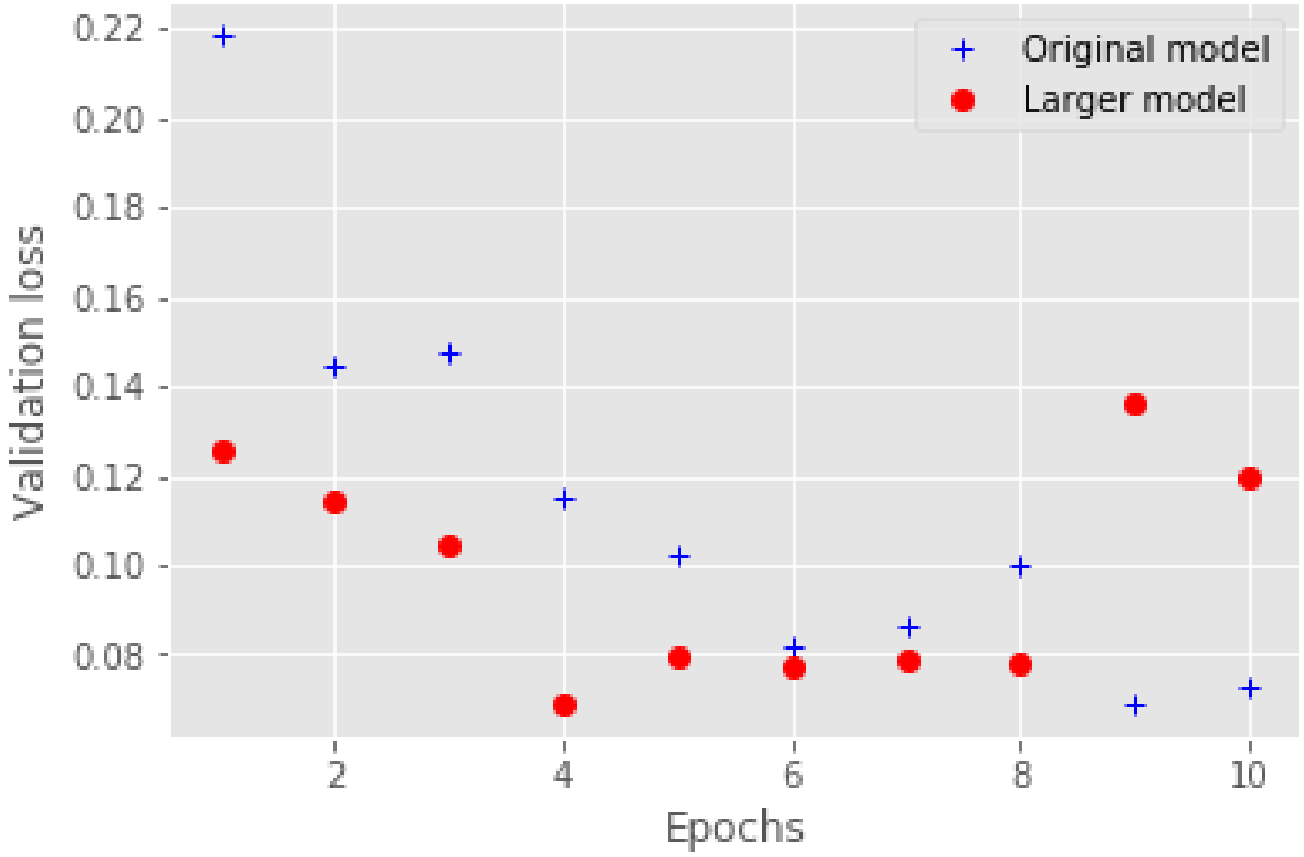
# Overcoming Overfitting

- Reduce network's size
  - Number of layers and number of nodes per layer (also called the learnable parameters). This dictates the capacity of the model. This part is more art than science
- Weight regularisation
  - Force the weight to be restricted to be within a range
  - Done by applying a cost term to the loss function.
  - L1 regularisation – cost term is proportional to the absolute value of the weight coefficients
  - L2 regularisation – cost term is proportional to the squared value of the weight coefficients.
- Dropout
  - Most commonly used
  - Applied to a layer
  - Randomly dropping (zero setting) a number of output features of the layer during training
  - According to Chollet, dropout rate usually set between 0.2, 0.5
  - No dropout during test time.

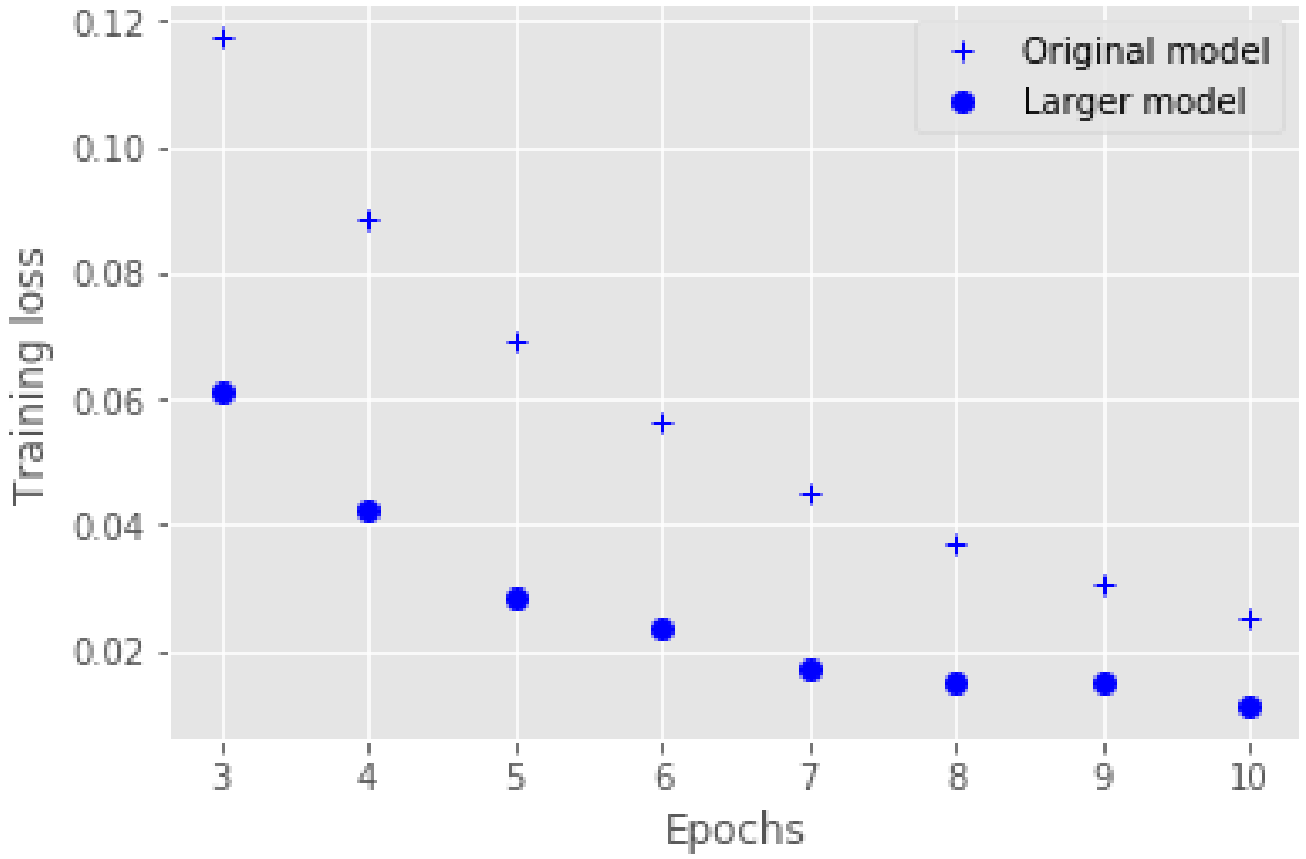
# Example of Reducing Network Size Smaller Model



# Example of Bigger Model



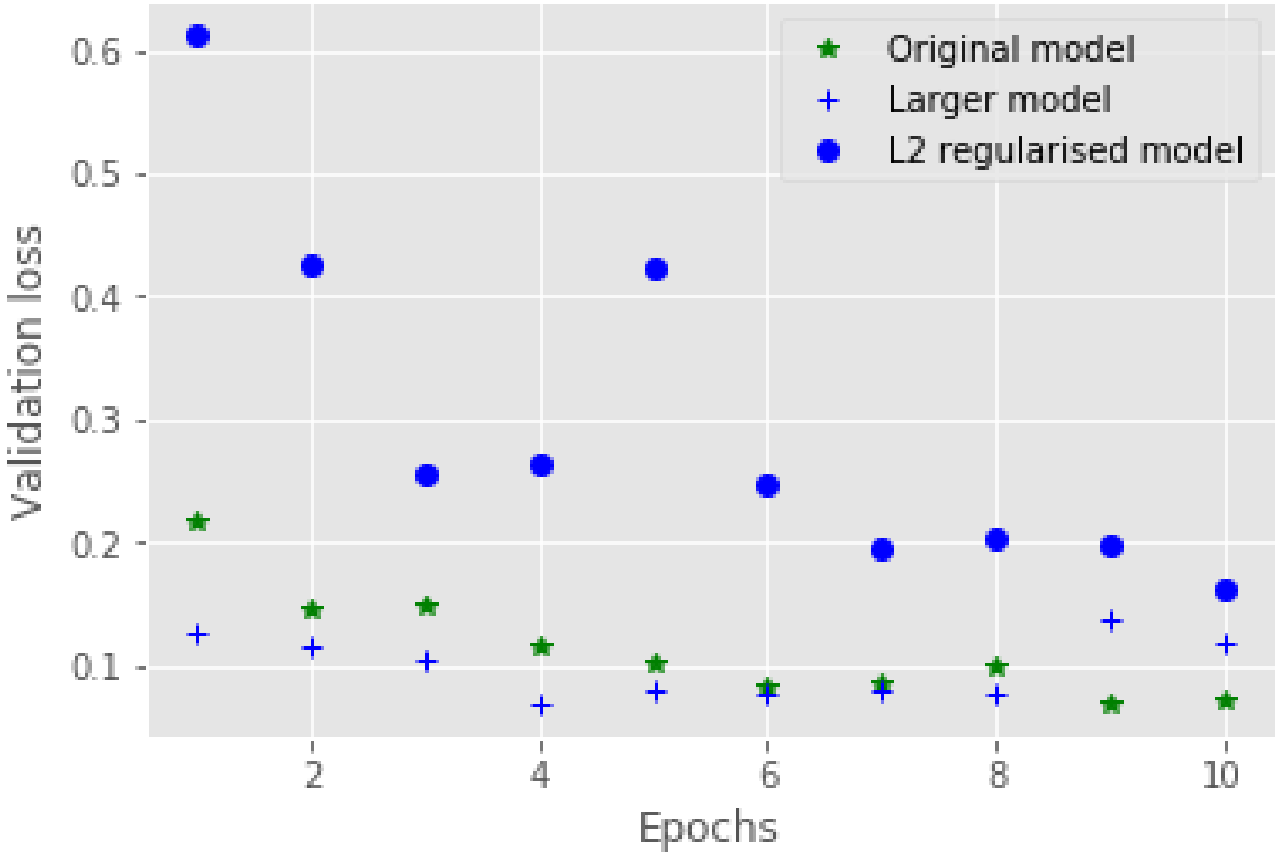
# Original vs Larger Model Training Loss



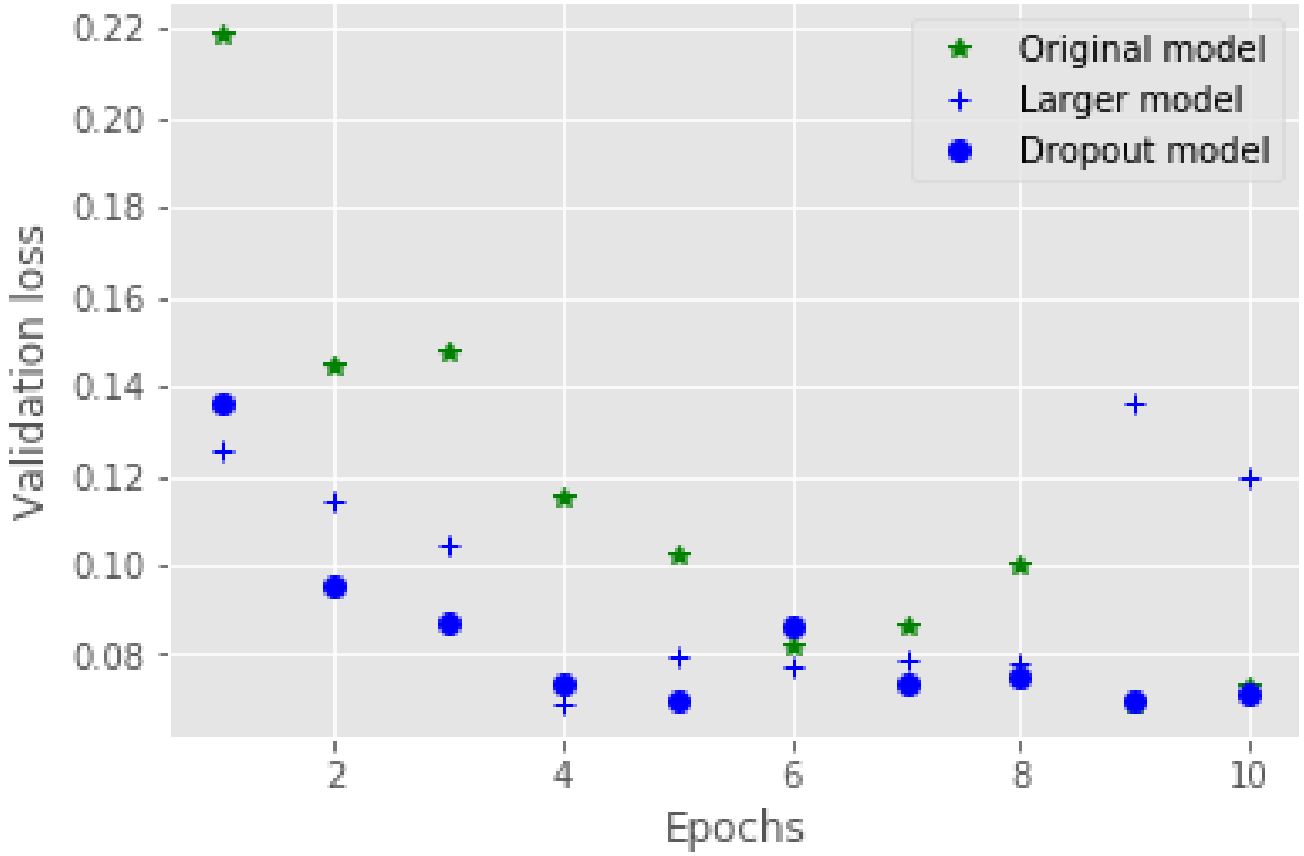
# Example of weight regularisation

```
from keras import regularizers

l2_model = models.Sequential()
l2_model.add(layers.Dense(units=2048, kernel_regularizer=regularizers.l2(0.001),
                          activation='relu', input_shape=(28 * 28,)))
#l1_model.add(layers.Dense(units=2048, kernel_regularizer=regularizers.l1(0.001),
#                          activation='relu', input_shape=(28 * 28,)))
#l1_l2_model.add(layers.Dense(units=2048, kernel_regularizer=regularizers.l1_l2(l1=0.001, l2=0.001),
#                          activation='relu', input_shape=(28 * 28,)))
l2_model.add(layers.Dense(units=1024, kernel_regularizer=regularizers.l2(0.001),
                          activation='relu'))
l2_model.add(layers.Dense(units=10, activation='softmax'))
l2_model.summary()

l2_model.compile(optimizer='rmsprop',
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])
l2_model_history = l2_model.fit(x_train_input, y_train,
                                epochs=10, batch_size=512,
                                validation_data=(x_test_input, y_test))
```

# Example of weight regularisation



# Example of dropout



# Machine Learning Workflow

1. Defining the problem and collect dataset
   - Problem definition. What kind of problem is it? Binary classification? Multiclass classification? Regression? Clustering? Reinforcement learning?
   - Do you have the necessary data?
2. Choosing a measure of success
   - Accuracy? Precision and recall? ROC AUC?
3. Decide on an evaluation protocol
   - Hold-out validation set
   - K-fold cross-validation
4. Data preparation
5. Develop a model that performs better than a baseline
6. Scaling up: developing a model that overfits
7. Regularising your model and tuning your hyperparameters

# Hinton