

**GitHub**

<https://github.com/adrianbevan/TensorFlow-Tutorial>



launch binder

<https://mybinder.org/v2/gh/adrianbevan/TensorFlow-Tutorial/V2.0>

ADRIAN BEVAN

---

# TENSORFLOW TUTORIAL

## GRAD<sub>NET</sub> TUTORIAL 20TH JAN 2021

If you want more background material on these ML methods then please see [my graduate lectures](#) from the 2020 RAL PPD Summer Lecture Series.

I also have other machine learning lectures available on my [teaching webpage](#).



# OVERVIEW

- ▶ The code for this tutorial can be found on [github](#), and you will be using [Binder](#) to work with the code.



- ▶ Once the binder session starts click on the notebooks directory to navigate to the jupyter notebooks for this tutorial.
- ▶ Package requirements for this include:

```
matplotlib==3.2.1  
sklearn  
tensorflow==2.2.0  
numpy==1.18.4  
seaborn==0.11.0
```



# OVERVIEW

- ▶ You should see a webpage open that looks like

 jupyter

Visit repo

Copy Binder link

Quit

Files **Running** Clusters

Select items to perform actions on them.

Upload

New ▾



<input type="checkbox"/> 0 ▾	/	Name ▾	Last Modified	File size
<input type="checkbox"/>	/ notebooks		3 months ago	
<input type="checkbox"/>	/ scripts		3 months ago	
<input type="checkbox"/>	check_setup.py		3 months ago	374 B
<input type="checkbox"/>	LICENSE		3 months ago	35.1 kB
<input type="checkbox"/>	README.md		3 months ago	2.81 kB
<input type="checkbox"/>	requirements.txt		3 months ago	76 B
<input type="checkbox"/>	setup.sh		3 months ago	2.74 kB

- ▶ We're using the notebooks today, however there is a duplicate of these in the scripts directory in case you prefer to work with Python scripts after today.



# OVERVIEW

▶ You should see a webpage open that looks like



[Visit repo](#) [Copy Binder link](#) [Quit](#)

Files **Running** Clusters

Select items to perform actions on them.

[Upload](#) [New ▾](#) [↻](#)

<input type="checkbox"/> 0 ▾	📁 / notebooks	Name ▾	Last Modified	File size
<input type="checkbox"/>	..		seconds ago	
<input type="checkbox"/>	📄 CNN.ipynb		3 months ago	62.8 kB
<input type="checkbox"/>	📄 LinearRegression.ipynb		3 months ago	48.1 kB
<input type="checkbox"/>	📄 NN.ipynb		3 months ago	20.3 kB
<input type="checkbox"/>	📄 NN_parabola.ipynb		3 months ago	82.3 kB
<input type="checkbox"/>	📄 SK_BDT.ipynb		3 months ago	13.4 kB
<input type="checkbox"/>	📄 SK_DT.ipynb		3 months ago	14 kB
<input type="checkbox"/>	📄 SK_RF.ipynb		3 months ago	14.9 kB
<input type="checkbox"/>	📄 SK_SVM.ipynb		3 months ago	14.8 kB
<input type="checkbox"/>	📄 README.md		3 months ago	1.38 kB

Focus on these today



# OVERVIEW

- ▶ The following examples are provided to work through:
  - ▶ [LinearRegression.ipynb](#)
  - ▶ [NN\\_parabola.ipynb](#)
  - ▶ [NN.ipynb](#)
  - ▶ [CNN.ipynb](#) (this one takes a long time to train!!!)
- ▶ The [scripts](#) directory of the github code also includes example scripts for hyper-parameter optimisation that you may wish to explore in your own time.



# OVERVIEW

- ▶ Training hyper-parameters of interest include:
  - ▶ **Batch Size:**
    - ▶ The number of examples used in a given iteration of the optimisation algorithm.
  - ▶ **Dropout Rate:**
    - ▶ The fraction of nodes dropped out in a given layer of a network.
  - ▶ **Leaky ReLU alpha:**
    - ▶ The coefficient multiplying the negative half of the activation function.
  - ▶ **Learning Rate:**
    - ▶ Related to the optimisation algorithm step size (usage depends on algorithm).
  - ▶ **Epochs:**
    - ▶ Number of times the training data are looped over when learning the model.
  - ▶ **Validation Split:**
    - ▶ The fraction of training data used for validation when learning the model.
- ▶ The model architecture is also configurable and this affects model performance.

---

# LINEAR REGRESSION



# LINEAR REGRESSION:

- ▶ [LinearRegression.ipynb](#)
- ▶ Generate noisy data according to  $y = mx + c$
- ▶ Use a linear activation function to learn  $m$  and  $c$ 
  - ▶ How many inputs?
  - ▶ How many outputs?
  - ▶ How many model hyper-parameters?
- ▶ Use the Adam optimiser to learn the function.





# LINEAR REGRESSION:

$$y = mx + c$$

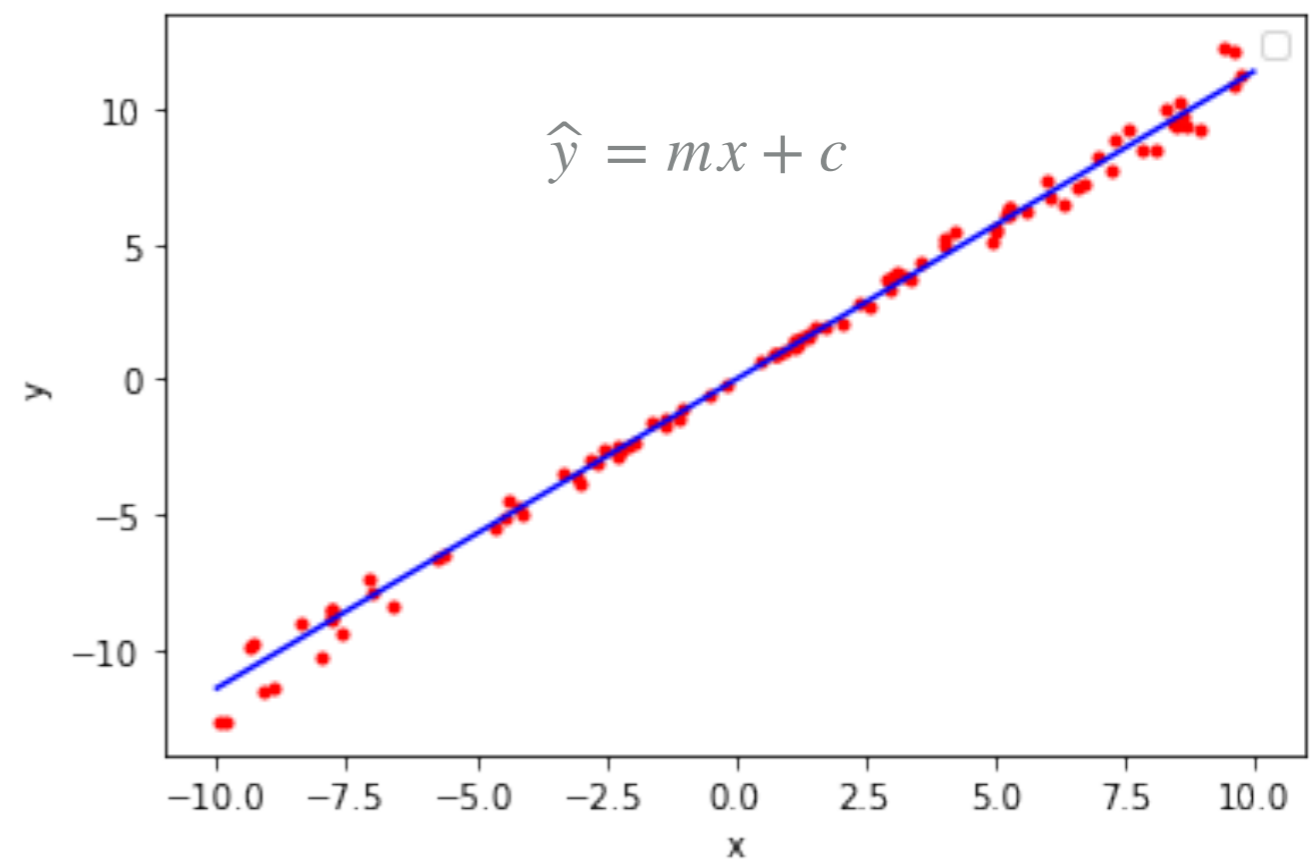
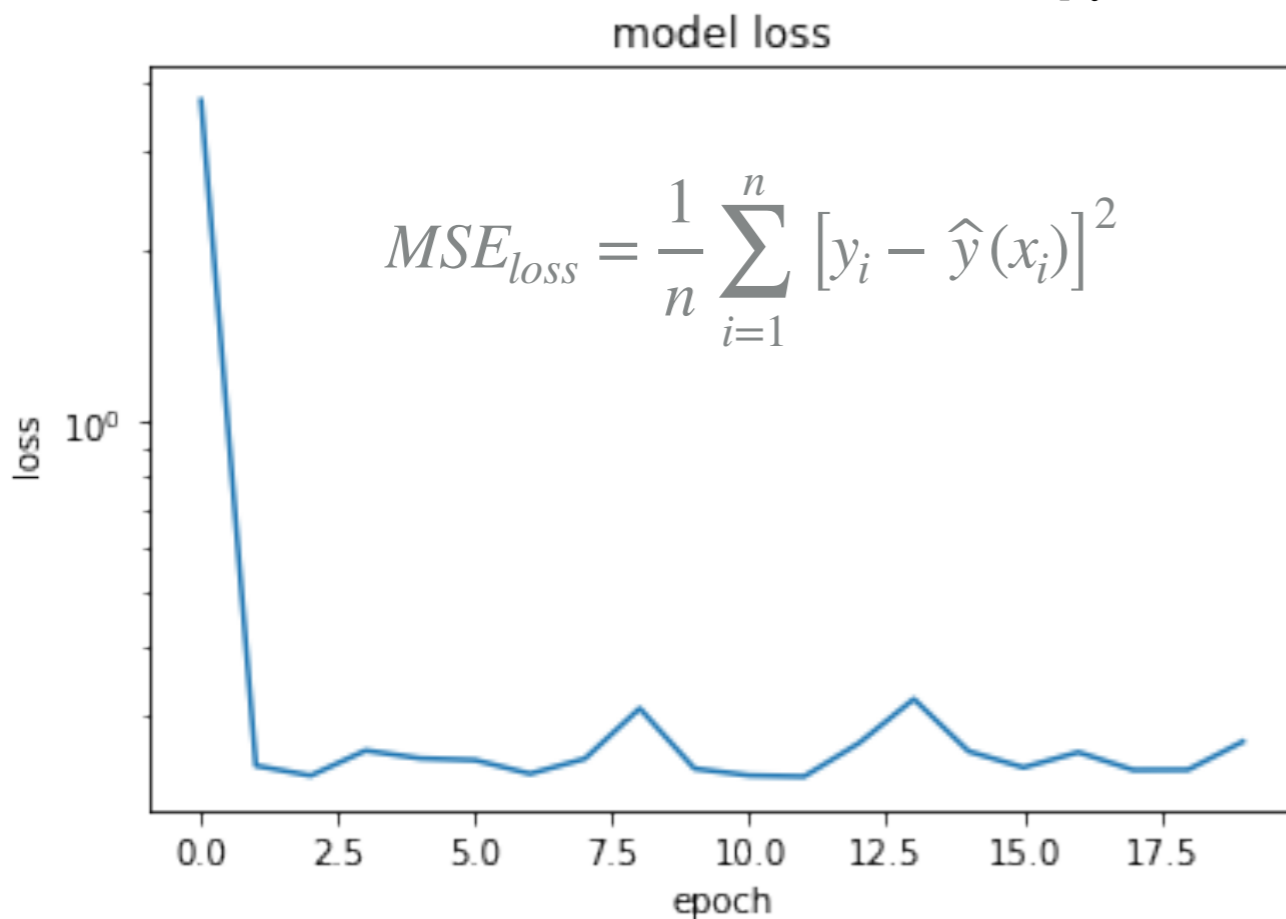


- ▶ There is one input  $x$
- ▶ There is one output:  $y$
- ▶ There are two model hyperparameters:  $m$  and  $c$



# LINEAR REGRESSION:

- ▶ [LinearRegression.ipynb](#)
- ▶ Able to fit a straight line to extract the parameters.
- ▶ Unlike a likelihood or  $\chi^2$  fit, we don't get uncertainties





# LINEAR REGRESSION: SUGGESTED EXERCISES

- ▶ [LinearRegression.ipynb](#)
- ▶ Change the number of training examples to see how this affects the optimisation performance (increase by a factor of 10 and decrease by a factor of 10).
- ▶ Change the value of  $m$  and  $c$  to extract, Try  $m=1000$ ,  $c=-500$ , to explore how this affects the training. You may also need to change the number of epochs when doing this.
- ▶ Change the number of training epochs to see how this affects the optimisation
- ▶ Change the noise level to study how this affects the optimisation.
- ▶ Change the learning rate to explore how robust the training is with the Adam optimiser.
- ▶ You may also wish to explore the use of other optimisers: see <https://keras.io/api/optimizers/>.

THERE ARE 2 EXAMPLES:

1) PARABOLIC REGRESSION PROBLEM: LEARNING  $y = x^2$

2) MNIST CLASSIFICATION PROBLEM: IDENTIFYING HAND WRITTEN NUMBERS

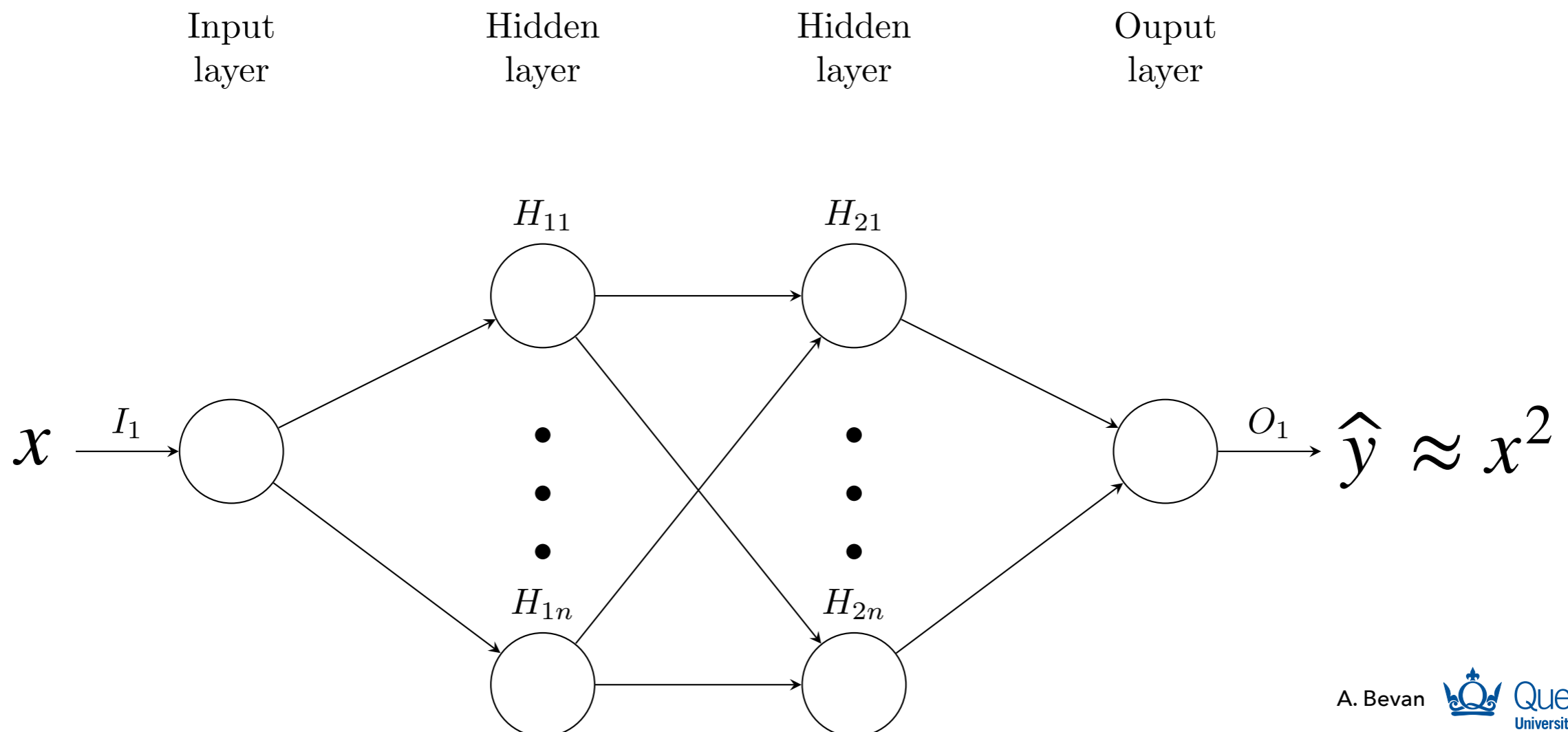
---

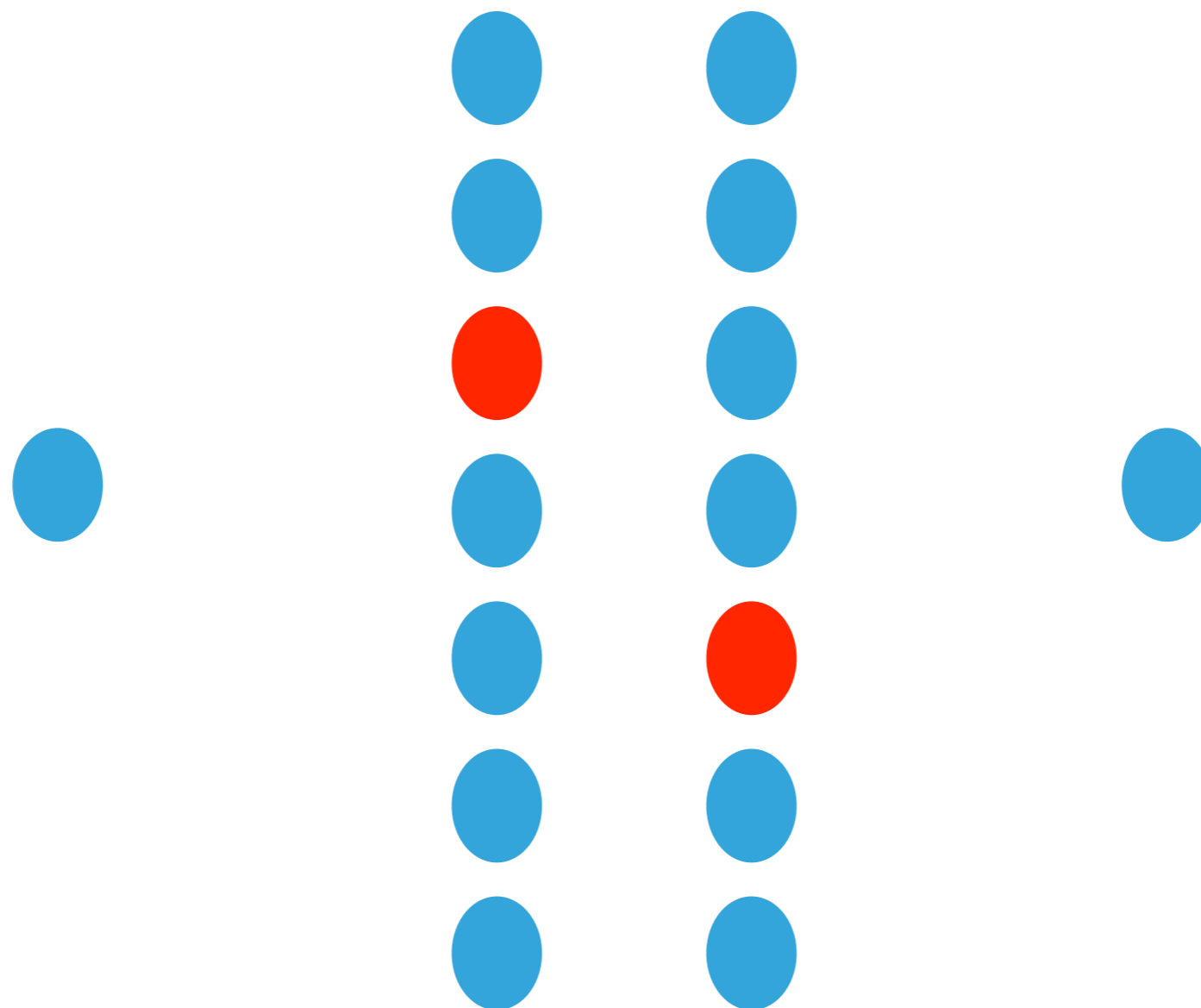
# NEURAL NETWORKS



# NEURAL NETWORKS

- ▶ [NN\\_parabola.ipynb](#)
- ▶ Generate noisy data according to  $y = x^2$
- ▶ Use a multilayer perceptron to learn the function
  - ▶ Remember that machine learning is just function approximation (although we may not always think of it in those terms).

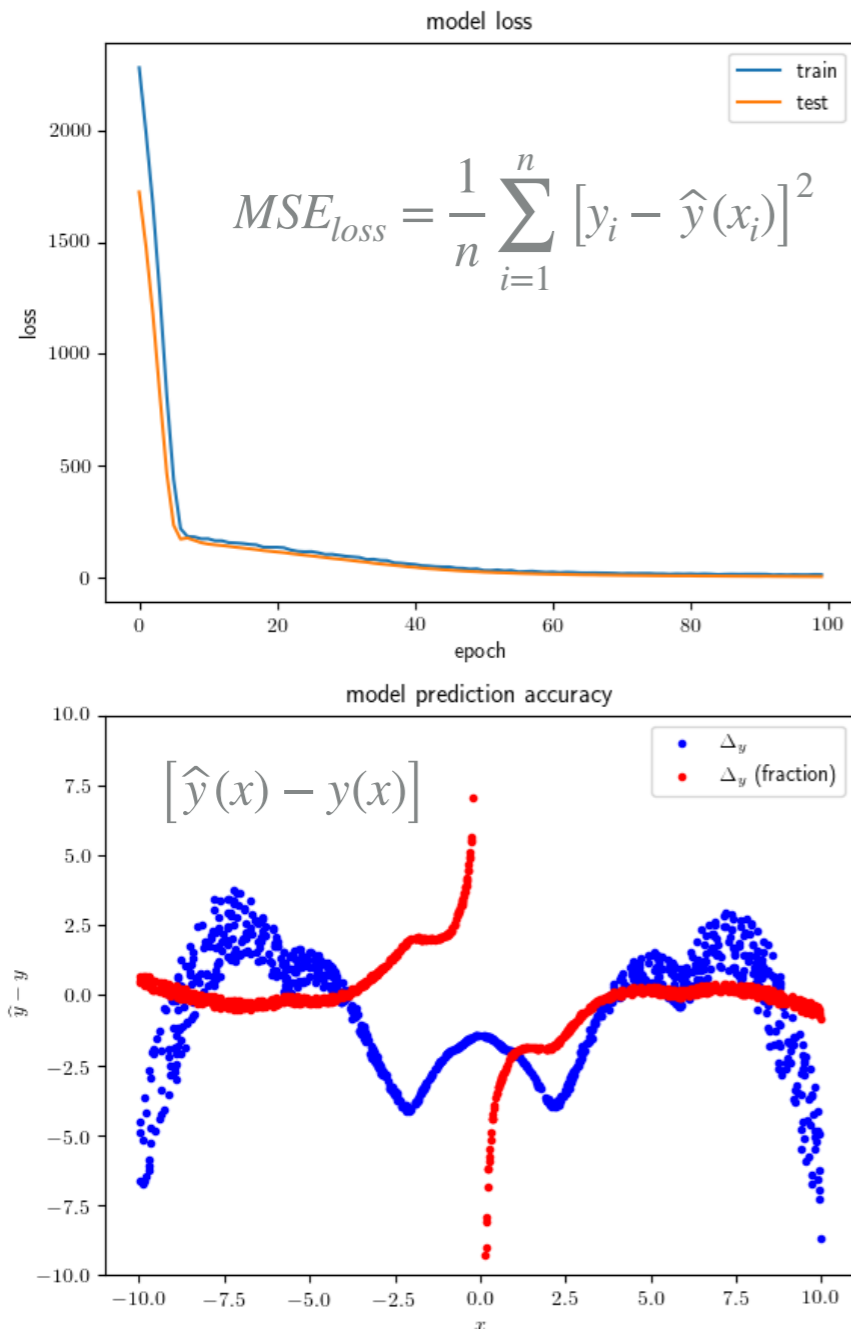




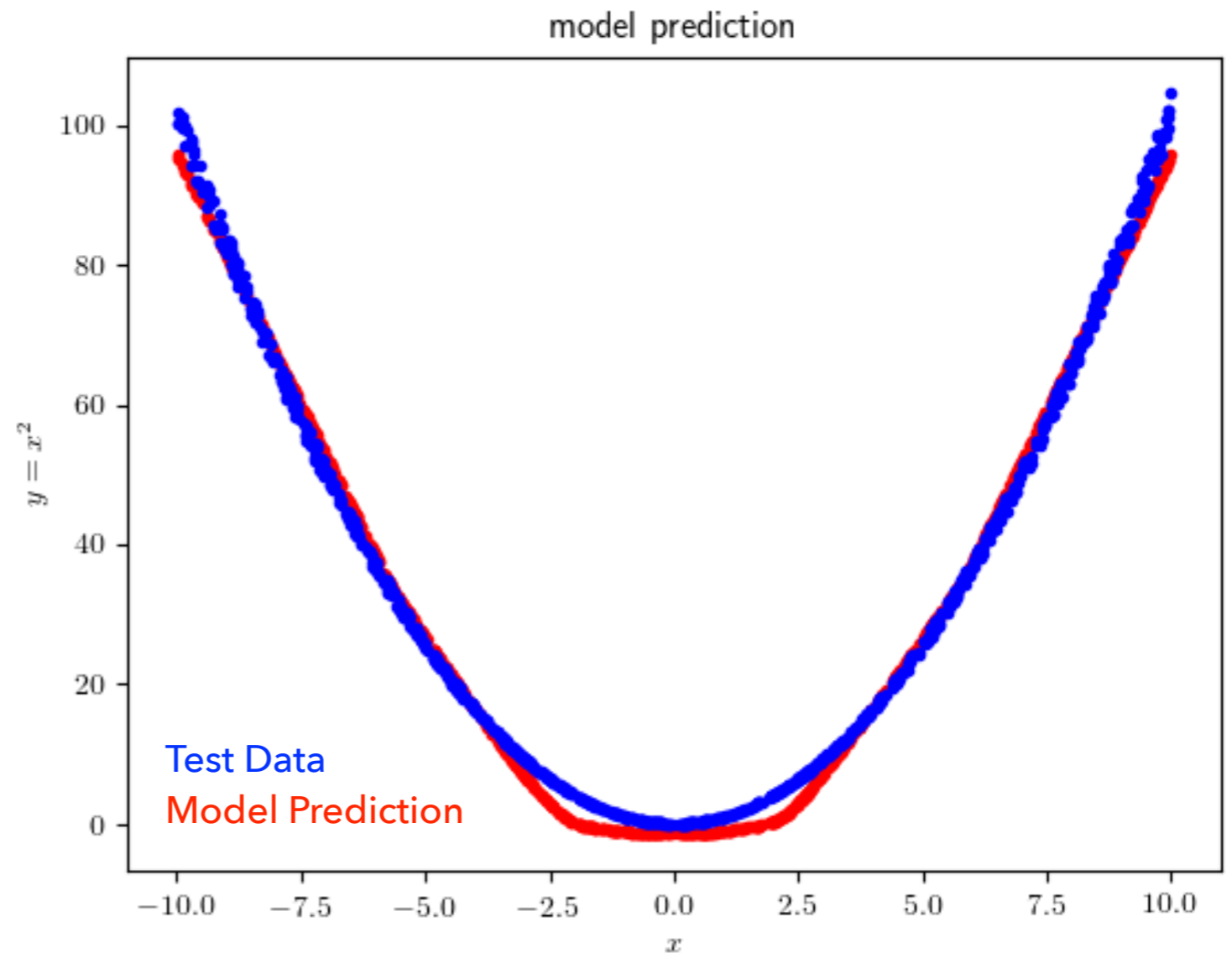


# NEURAL NETWORKS

## ▶ [NN\\_parabola.ipynb](#)



With a little exploration and tweaking of hyper-parameters you should be able to get a much better model than this.





# NEURAL NETWORKS: SUGGESTED EXERCISES

- ▶ [NN\\_parabola.ipynb](#)
- ▶ Explore:
  - ▶ the effect of Dropout, ValidationSplit, Nepochs, and BatchSize have on the training (try to find a model where the test and train loss function values are similar.
  - ▶ how the network structure affects the training performance (e.g. double or halve the number of nodes in the hidden layers)
  - ▶ the effect of adding a second dropout layer into the network after the first hidden layer.
  - ▶ what happens when the model is reduced to a single layer perceptron (removing the second hidden layer).
  - ▶ what happens when the model is changed by adding a third hidden layer to it.

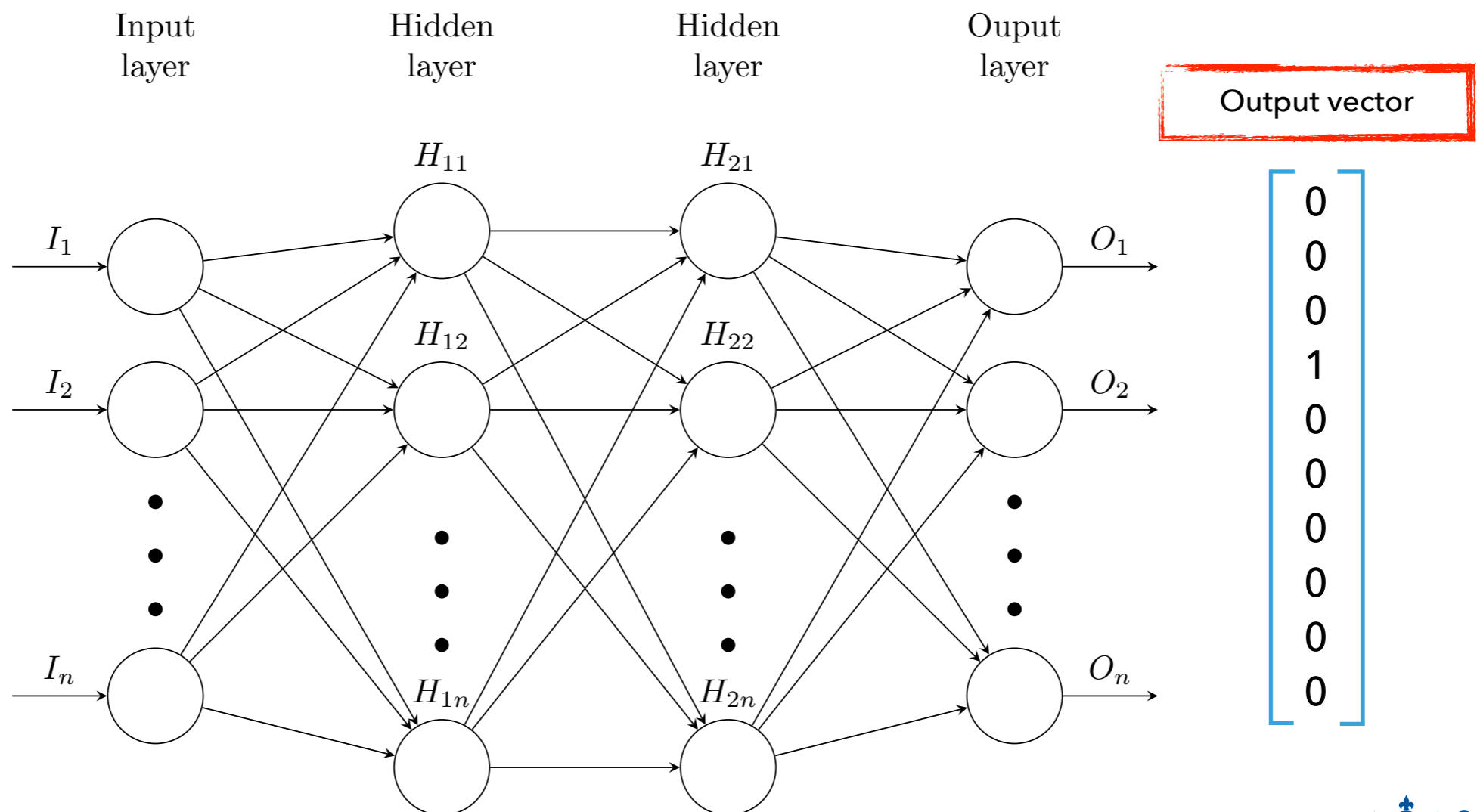
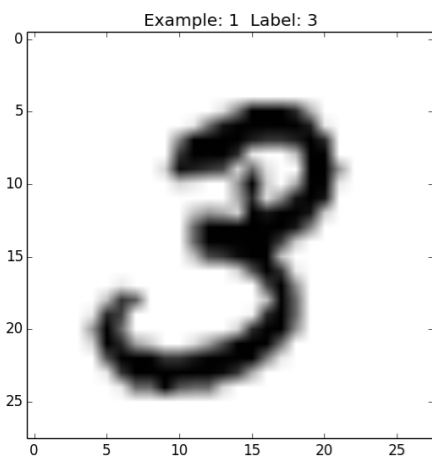




# NEURAL NETWORKS

- ▶ [NN.ipynb](#)
- ▶ Use MNIST data
- ▶ Use a multilayer perceptron to learn the classification function for the numbers  $0, 1, \dots, 9$

784 dimensional input feature space of a flattened image





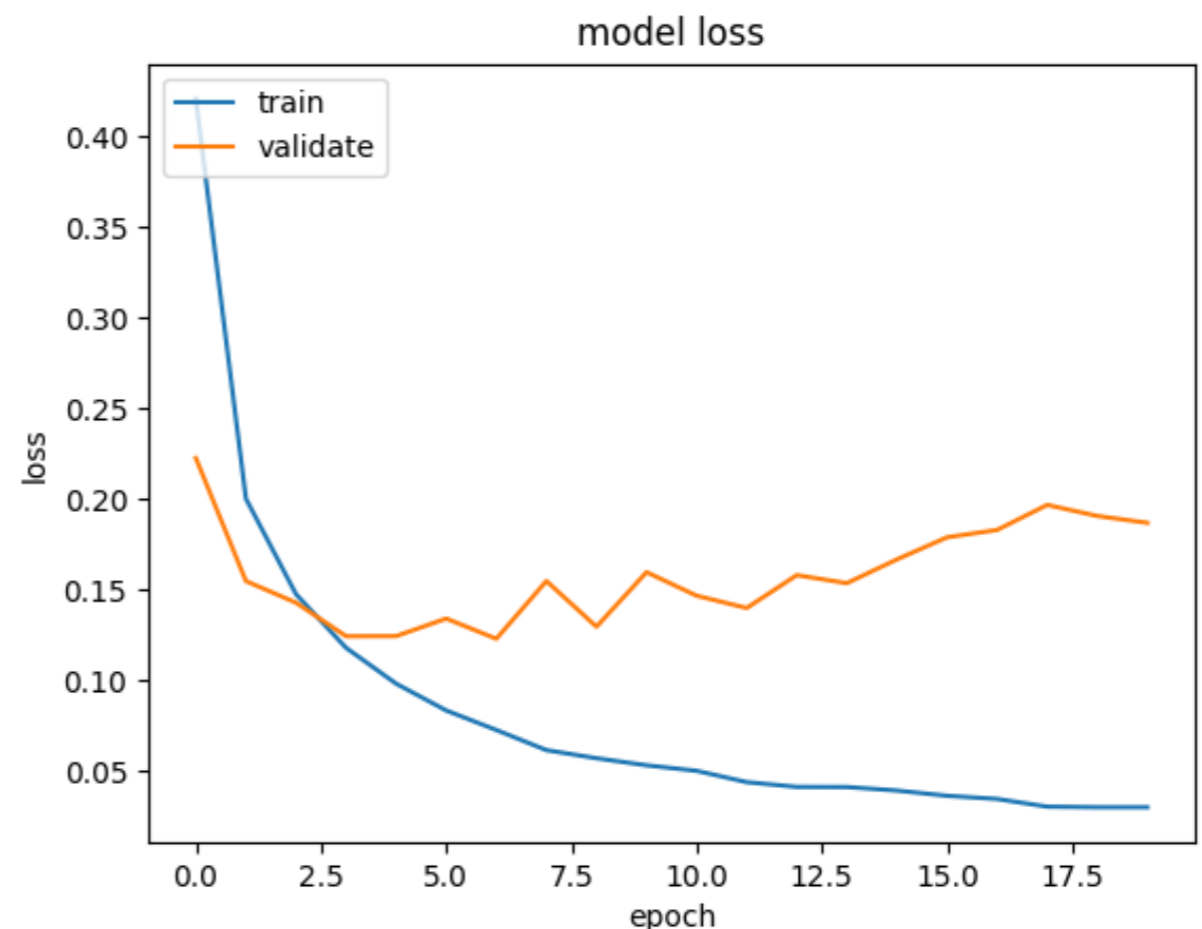
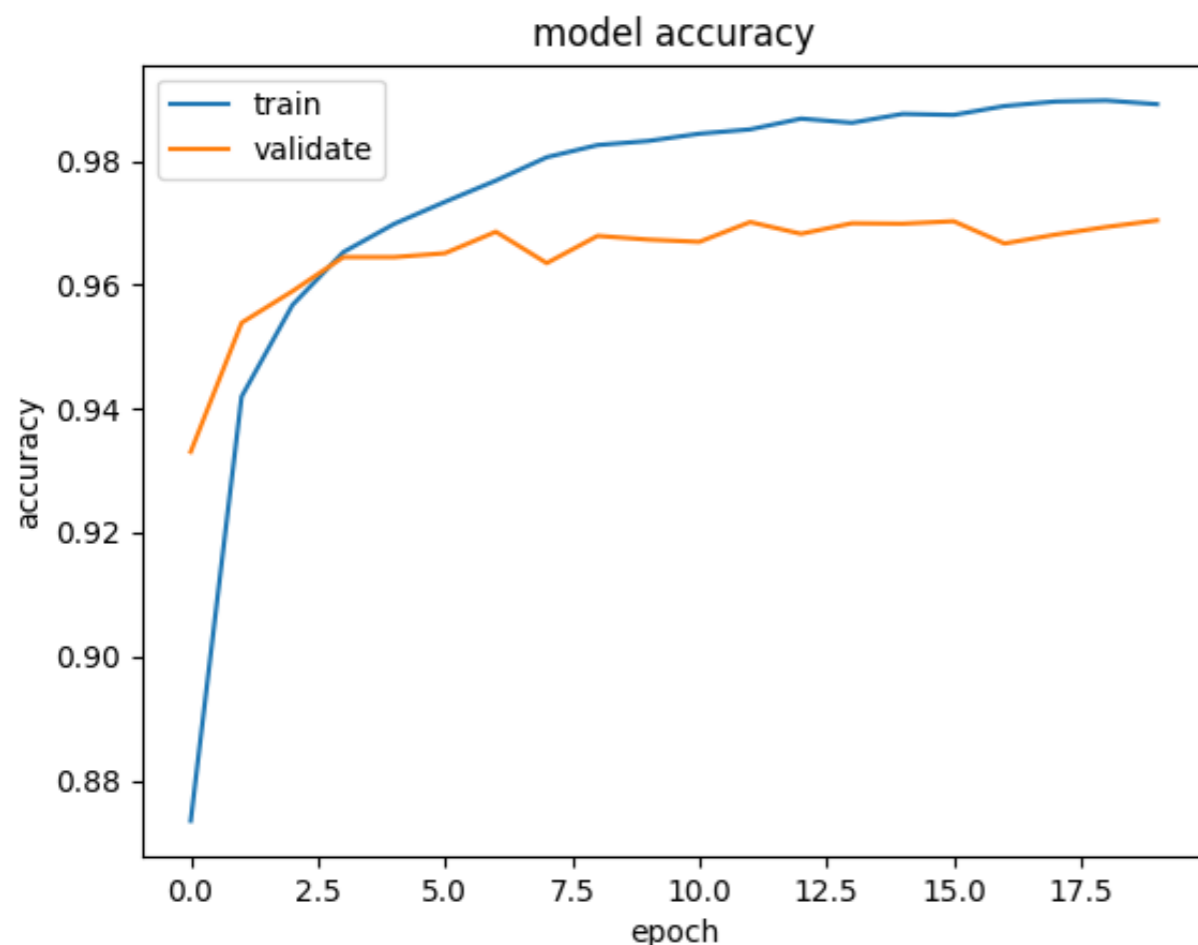
# NEURAL NETWORKS

► [NN.ipynb](#)

Good accuracy, but train and validate sample losses differ - this model overtrains.

Need to vary hyper-parameters to avoid overtraining the model.

BatchSize, DropoutValue and ValidationSplit are hyper-parameters that you might like to vary (along with increasing the number of epochs, Nepochs).





# NEURAL NETWORKS: SUGGESTED EXERCISES

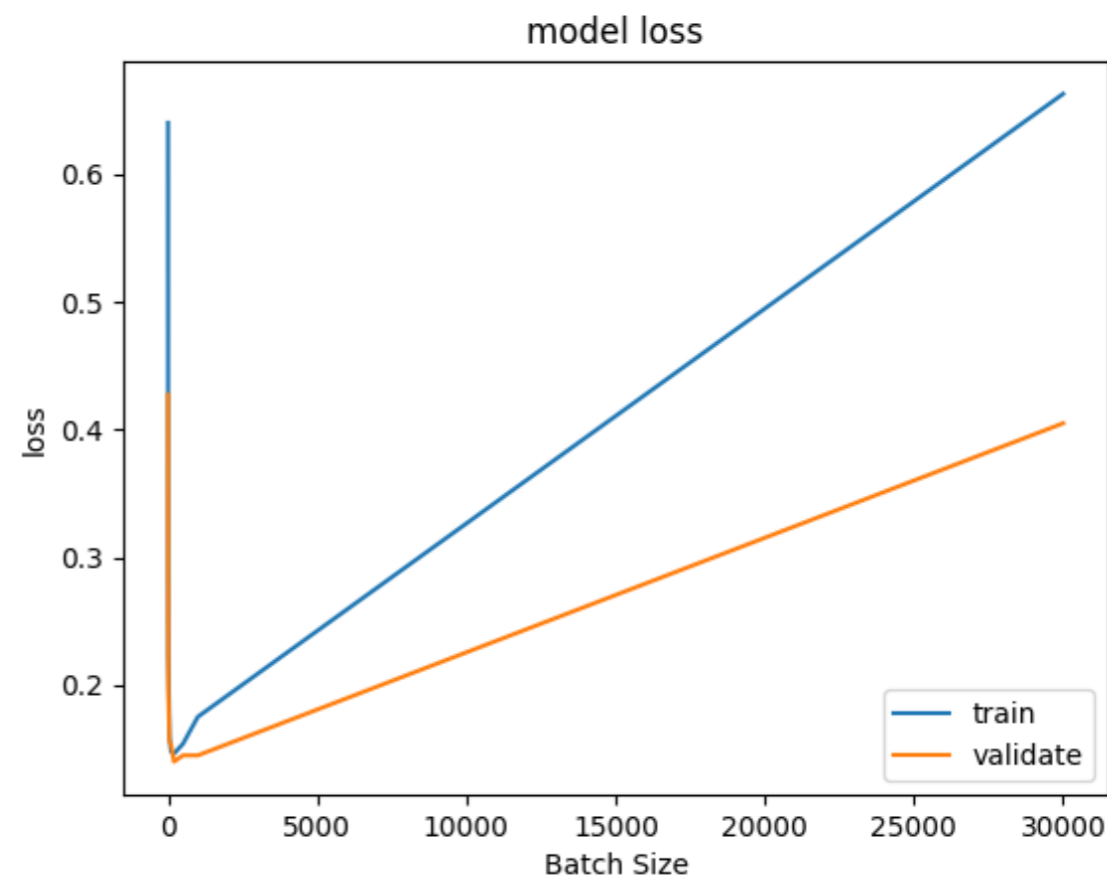
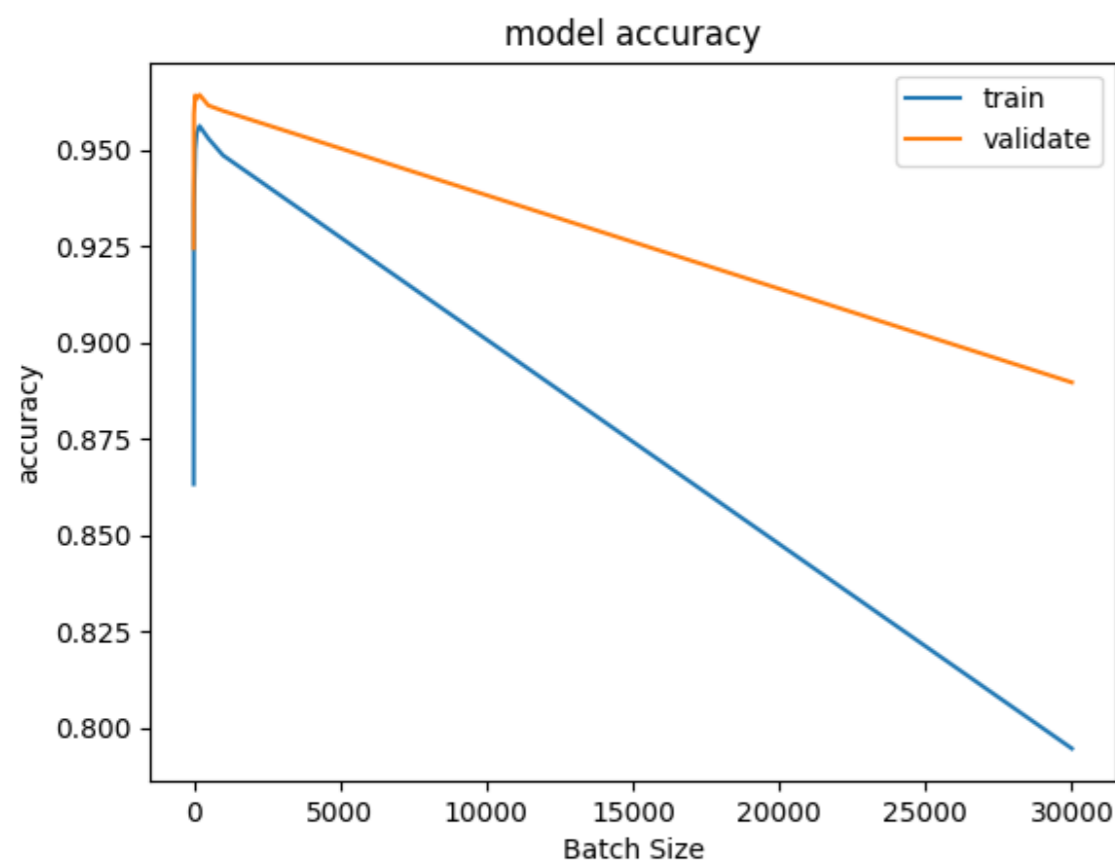
- ▶ [NN.ipynb](#)
- ▶ Explore the effect of DropOut, ValidationSplit, Nepochs, and BatchSize have on the training (try to find a model where the test and train loss function values are similar).
- ▶ Explore how the neural network structure affects the training performance (e.g. add double or halve the number of nodes in the hidden layers, the current value is 128 for both)
- ▶ Explore the effect of adding a second dropout layer into the network after the first hidden layer.



# HYPER-PARAMETER TUNING: BATCH SIZE

- ▶ The model hyper-parameters are not just the weights and biases in the network (for NN), the parameters chosen for the training and indeed model configuration affect model performance.

For this NN model, small batch sizes maximise model accuracy & minimise overtraining

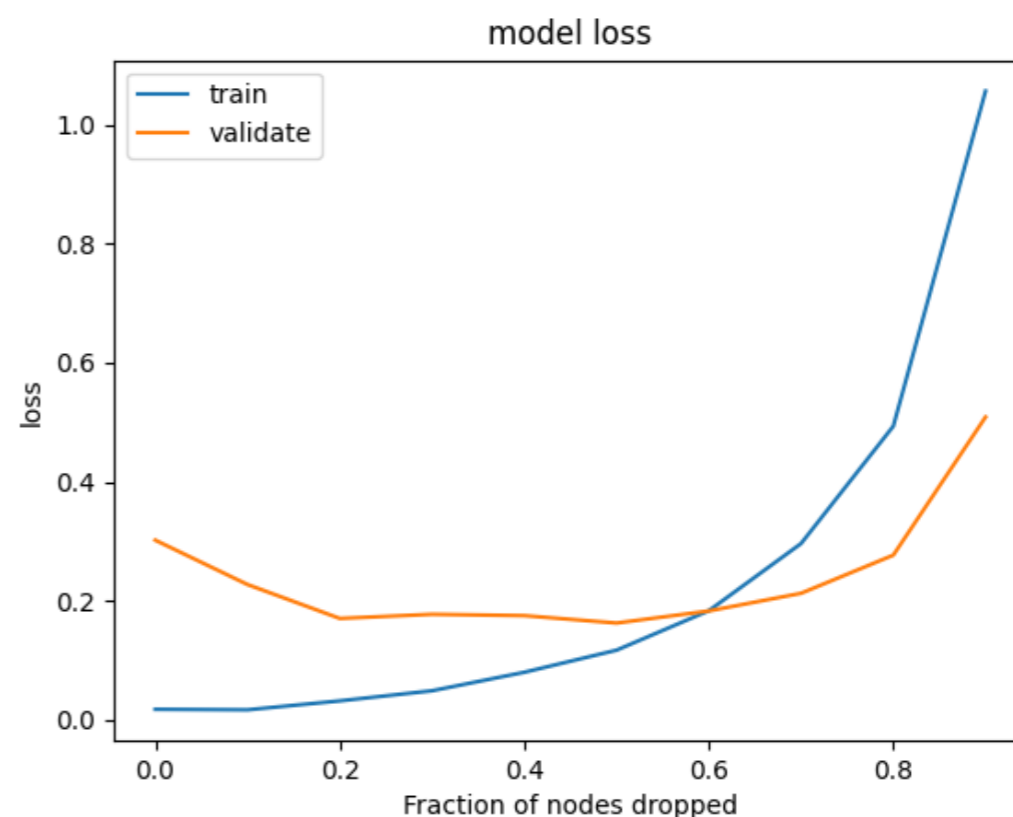
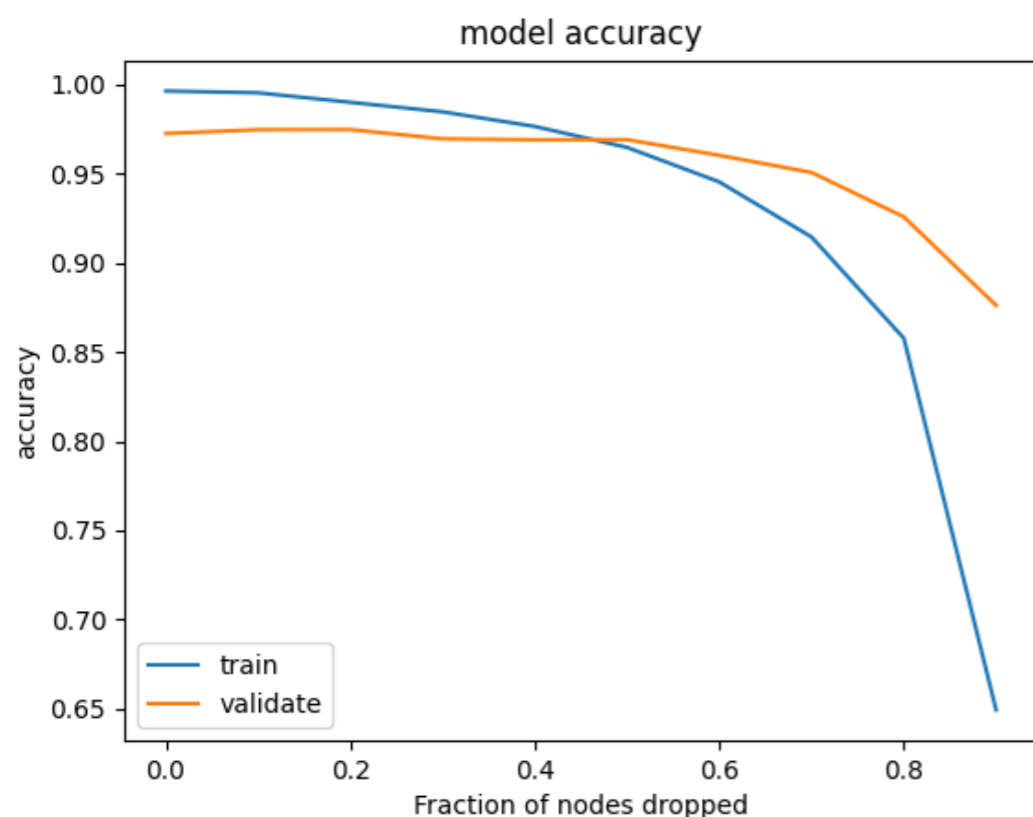




# HYPER-PARAMETER TUNING: DROPOUT FRACTION

- ▶ The model hyper-parameters are not just the weights and biases in the network (for NN), the parameters chosen for the training and indeed model configuration affect model performance.

For this NN model, a large dropout fraction of  $\sim 0.6$  gives consistent test and validate losses after N epochs of training, and the test and validate accuracies are similar. i.e. the model is generalised.

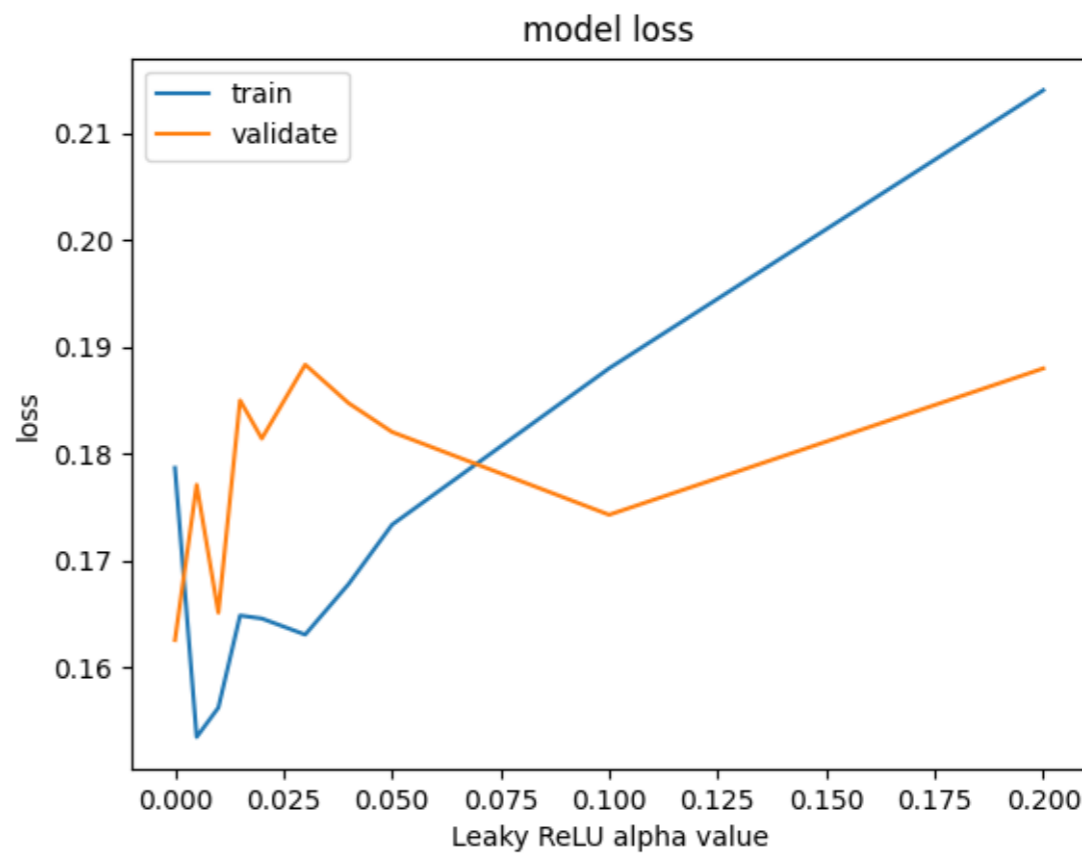
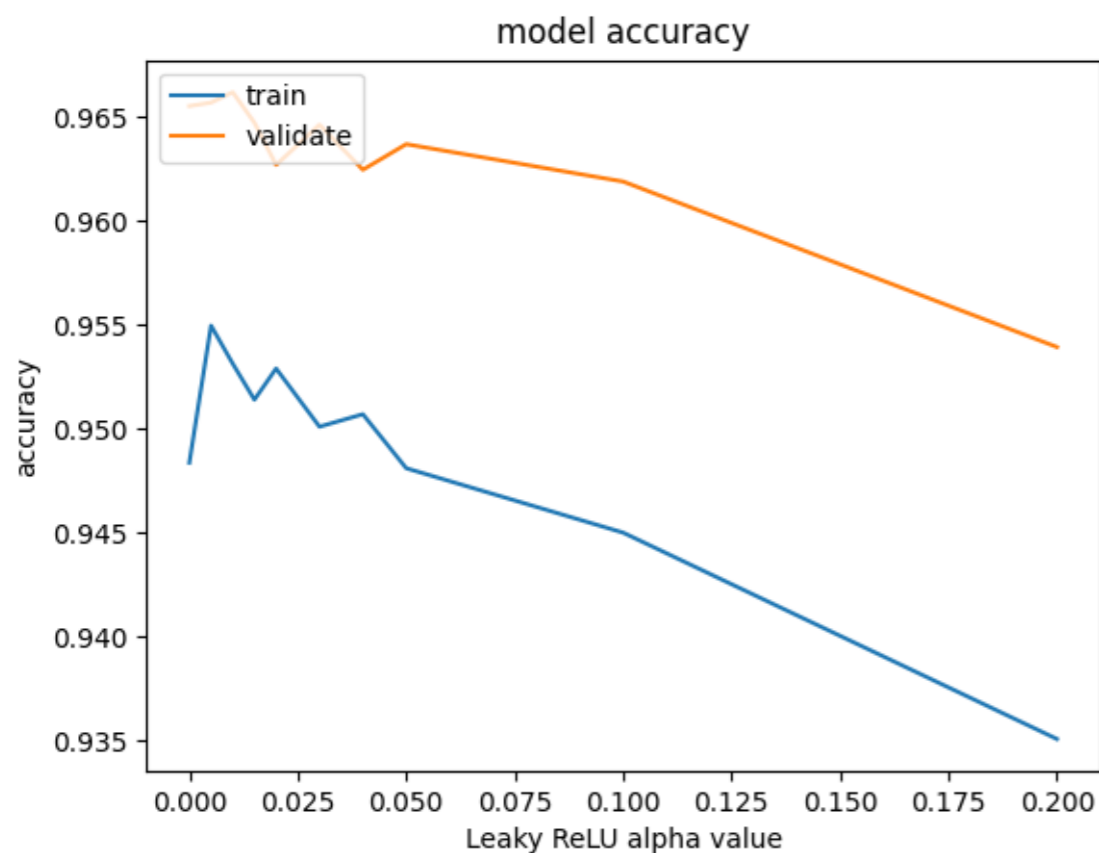




## HYPER-PARAMETER TUNING: BATCH SIZE

- ▶ The model hyper-parameters are not just the weights and biases in the network (for NN), the parameters chosen for the training and indeed model configuration affect model performance.

The Leaky ReLU activation alpha parameter affects model optimisation. For this example a value  $\sim 0.1$  yields similar train and validate performance of the model.

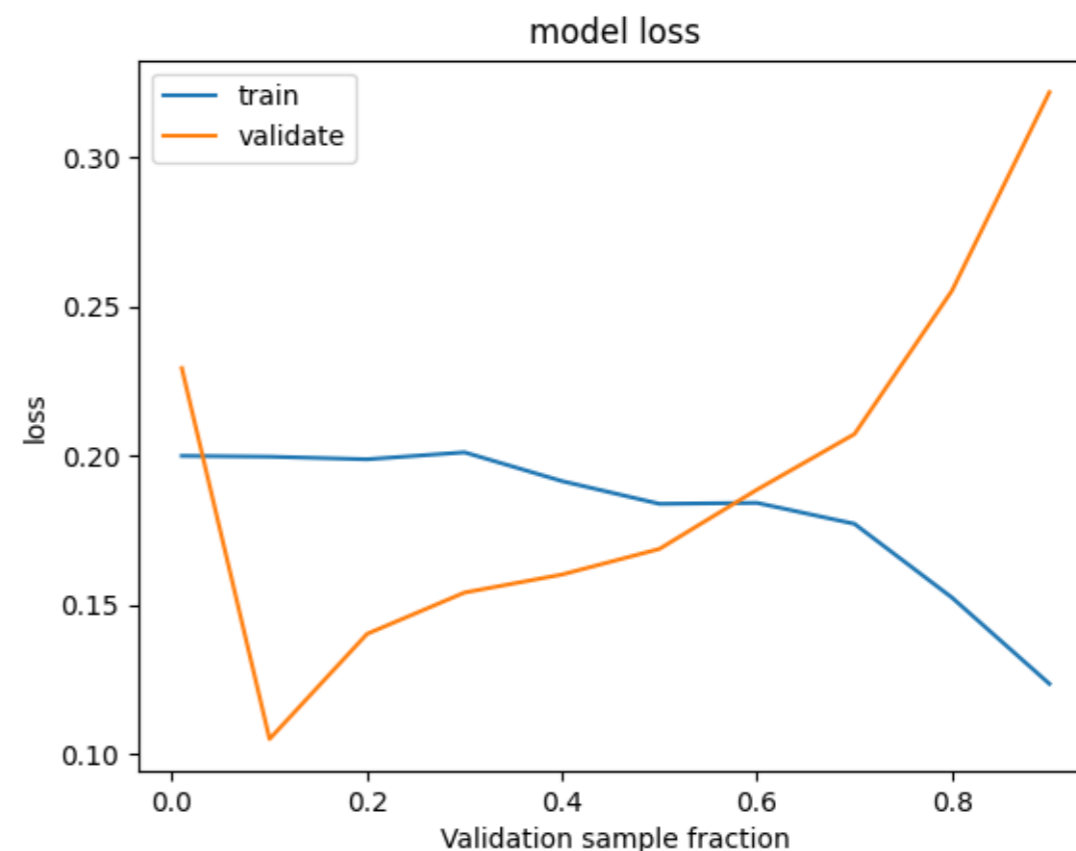
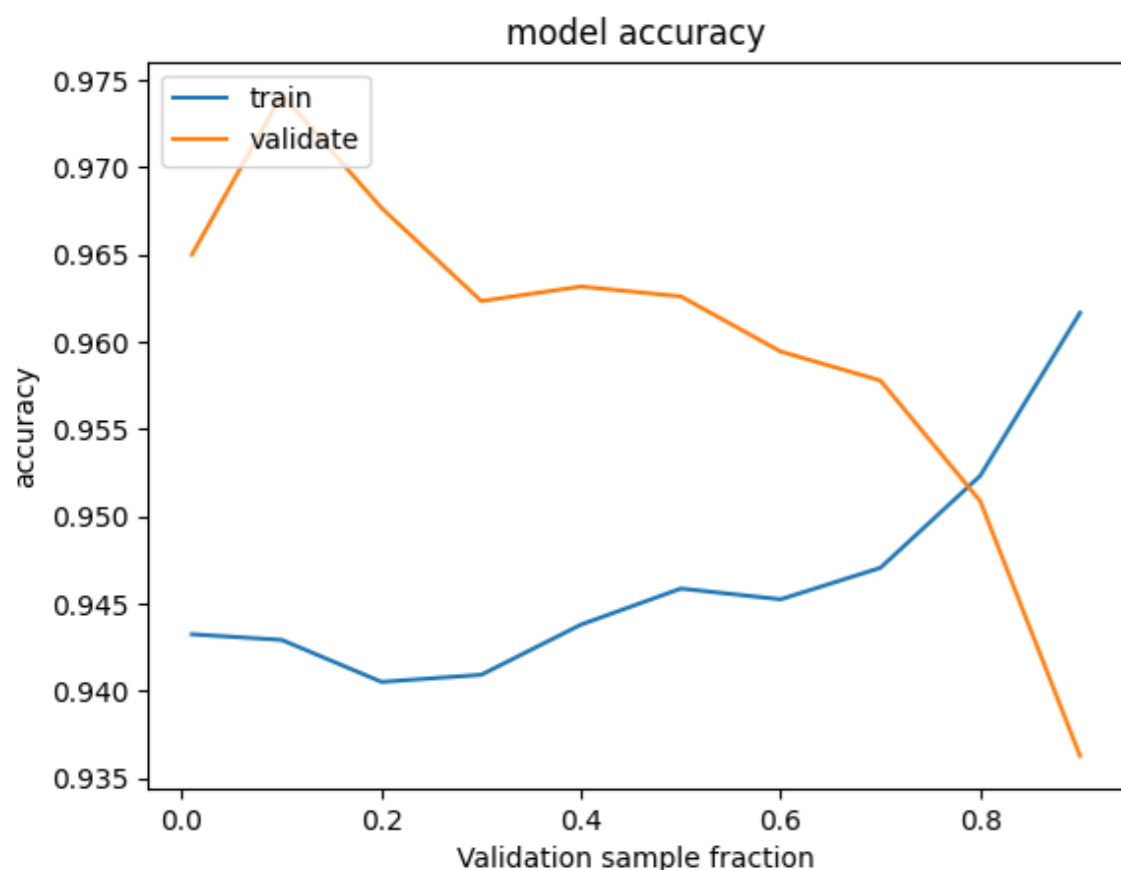




# HYPER-PARAMETER TUNING: BATCH SIZE

- ▶ The model hyper-parameters are not just the weights and biases in the network (for NN), the parameters chosen for the training and indeed model configuration affect model performance.

Validation sample fraction split of  $\sim 0.6$  yields similar loss function value for the train and validate samples. This shows tension between model accuracy and generalisability.



THERE ARE 2 EXAMPLES:

- 1) MNIST CLASSIFICATION PROBLEM: IDENTIFYING HAND WRITTEN NUMBERS
  - 2) CFAR10 CLASSIFICATION PROBLEM: IDENTIFYING 10 DIFFERENT TYPES OF COLOUR IMAGE
- 

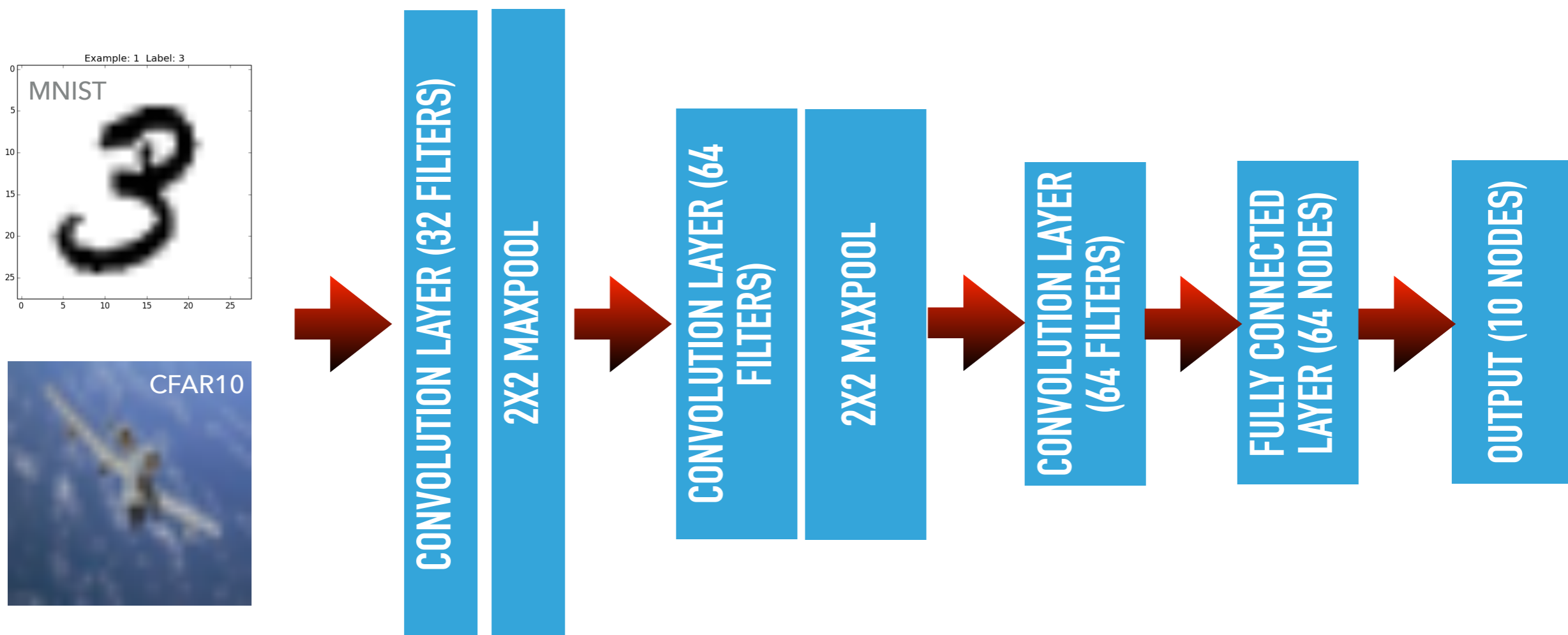
# CONVOLUTIONAL NEURAL NETWORKS





# CNNs

- ▶ [CNN.ipynb](#)
- ▶ Use either the MNIST hand writing data set, or CFAR10 (see appendix)
- ▶ Build a CNN model using conv(olution) and maxpool layers, and finishing with a fully connected (Dense) layer.
- ▶ Use Dropout.





# CNNS

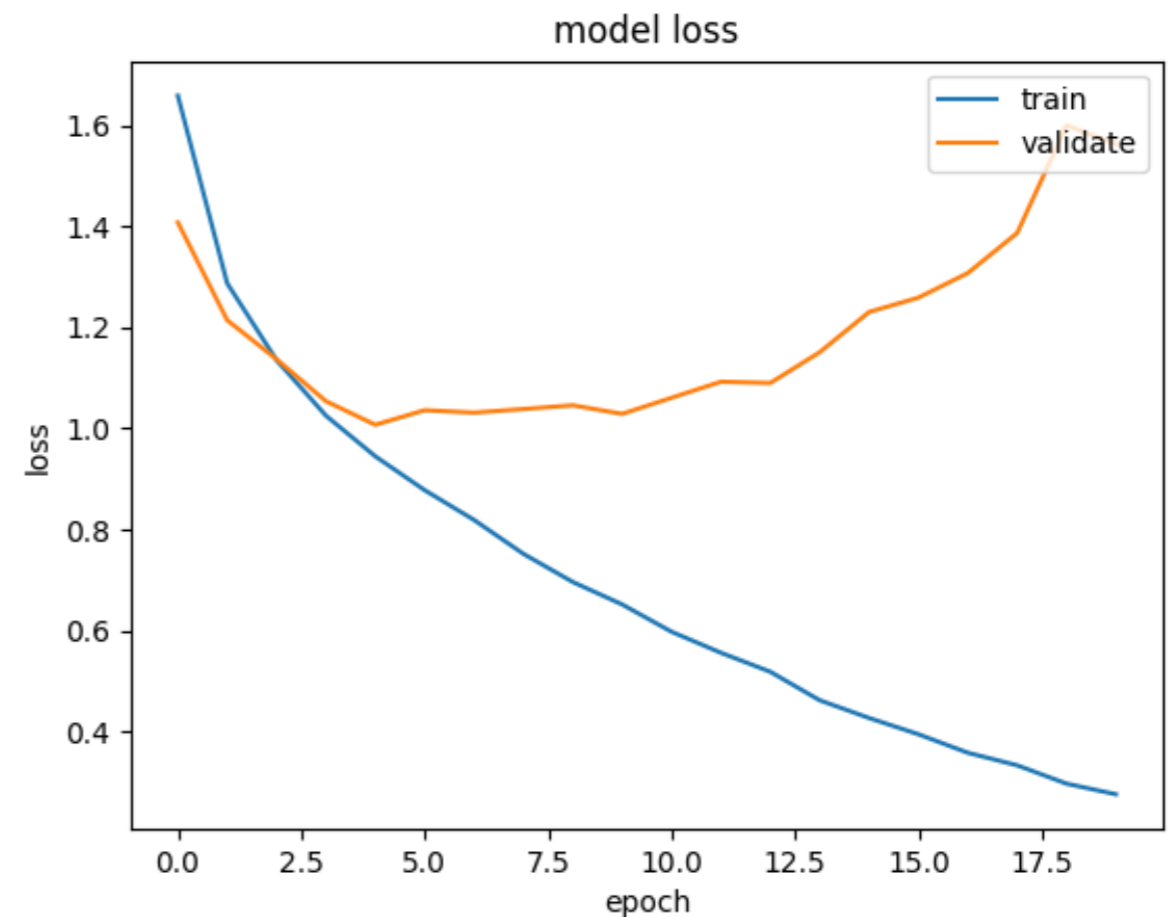
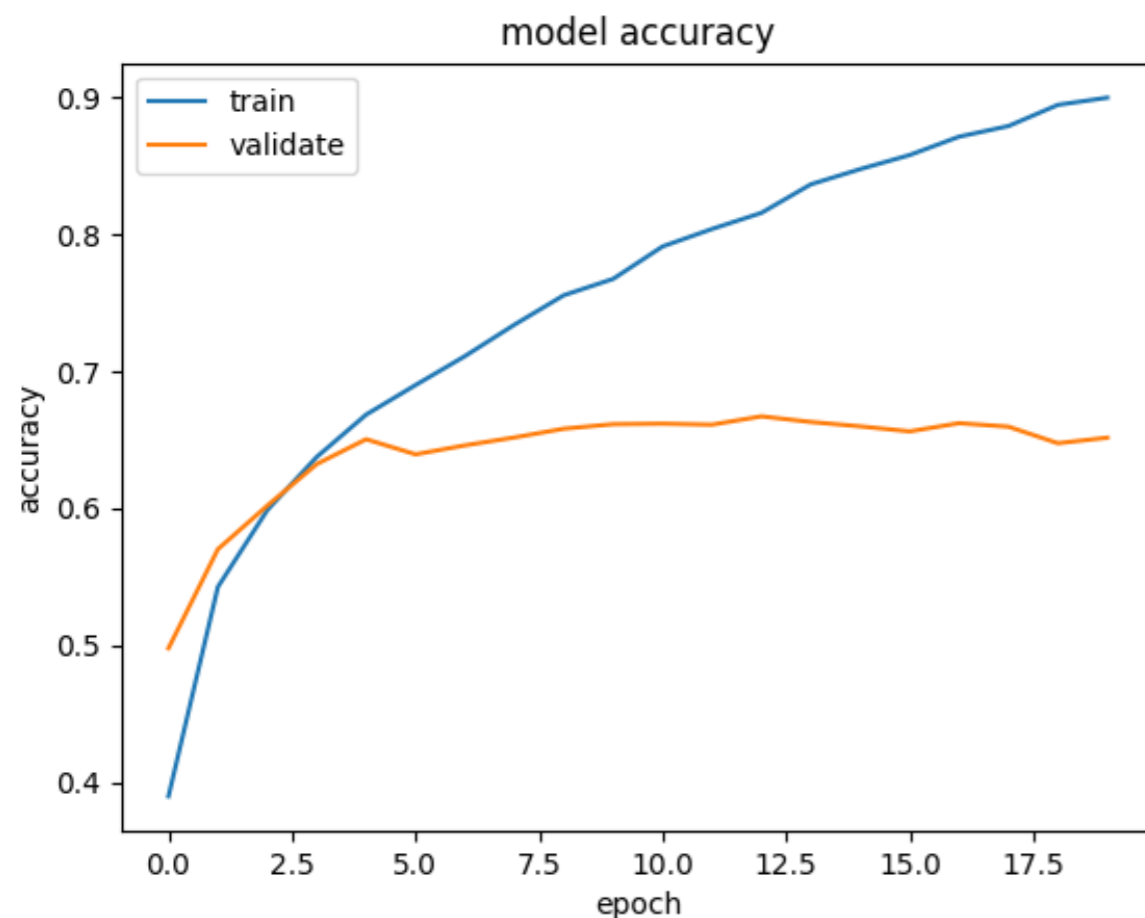
CNN training takes a while, and so you will want to continue to explore this model in your own time.

- ▶ [CNN.ipynb](#)
- ▶ Explore the effect of DropOut, ValidationSplit, Nepochs, and BatchSize have on the training (try to find a model where the test and train loss function values are similar).
- ▶ Explore how the CNN affects the training performance e.g.
  - ▶ change the number of convolution filters in each layer. The current values of these are 32, 64 and 64.
  - ▶ change the number of nodes in the fully connected (Dense) layer. The current value of nodes in this layer is 64.
- ▶ Explore the effect of adding a second dropout layers into the network after the conv and Dense layers (see the NN.ipynb example for how to implement a dense layer in a model).



# CNNs

- ▶ [CNN.ipynb](#)
- ▶ This is an excellent example of an overtrained model.
  - ▶ The hyper-parameters set for training with these data allow for the model to be overtrained as seen by the test accuracy significantly exceeding the validate accuracy.
  - ▶ The model loss also illustrates this issue well.



If you are unfamiliar with these algorithms please see [these lecture notes](#).

(USING SKLEARN)

---

# DECISION TREES AND SUPPORT VECTOR MACHINES



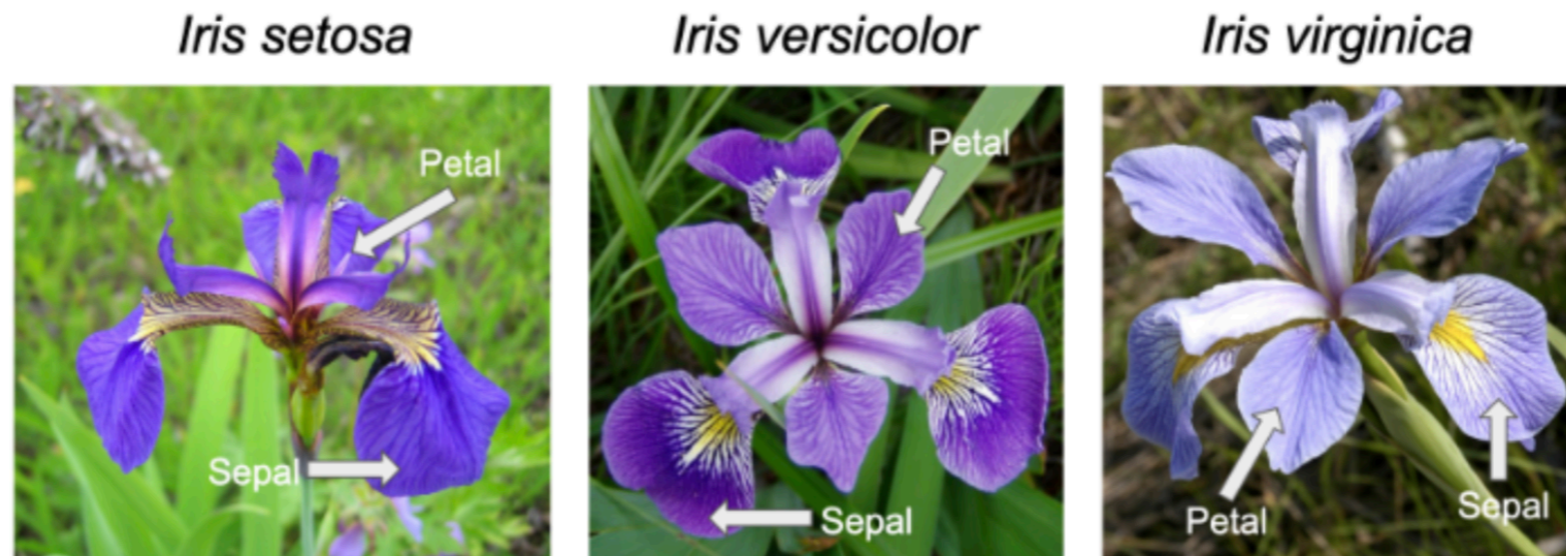
# SCIKIT LEARN CLASSIFIERS

- ▶ SK\_DT.ipynb [Decision Tree - single weak learner]
- ▶ SK\_BDT.ipynb [Boosted Decision Tree - an ensemble of weak learners using the AdaBoost]
- ▶ SK\_RF.ipynb [Random Forest - an ensemble of weak learners]
- ▶ SK\_SVM.ipynb [Support Vector Machine]
- ▶ These scripts create classifiers to analyse a test sample of the Iris data, and to produce a plot of the confusion matrix.



# SCIKIT LEARN CLASSIFIERS

## ► The data:



Petals & Sepals for *Iris setosa*, *Iris versicolor*, and *Iris virginica* (Sources: [1](#), [2](#), [3](#), Licenses: Public Domain, CC BY-SA 3.0 & CC BY-SA 2.0).

- 50 examples of each type of iris to be classified.
- 4 features: sepal width, sepal length, petal width and petal length.

Various datasets can be found in both [SciKit Learn](#) and [Keras](#).

DATA:  
MNIST  
CFAR-10  
CFAR-100  
KAGGLE  
UCI ML DATA REPOSITORY  
TIMIT  
RCV1-V2  
SCRIPTS

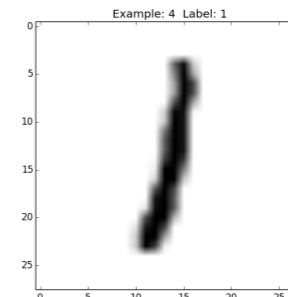
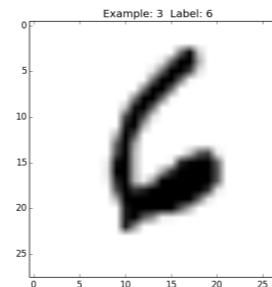
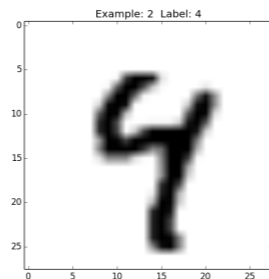
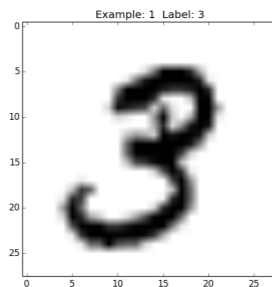
---

# APPENDIX – SOURCES OF DATA AND MISCELLANEOUS



## APPENDIX: DATA — MNIST

- ▶ MNIST is a standard data set for hand writing pattern recognition. e.g. the numbers 1, 2, 3, ... 9, 0



- ▶ 60000 training examples
- ▶ 10000 test examples
- ▶ These are 8 bit greyscale images (one number required to represent each pixel)
- ▶ Renormalise  $[0, 255]$  on to  $[0, 1]$  for processing.
- ▶ Each image corresponds to a 28x28 pixel array of data.
- ▶ For an MLP this translates to 784 features.





## APPENDIX: DATA — CFAR-10

- ▶ 60k 32x32 colour images (so each image is a tensor of dimension 32x32x3).
- ▶ This is a labelled subset of an 80 million image dataset.

- ▶ 10 classes:

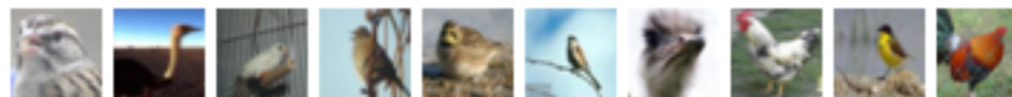
airplane



automobile



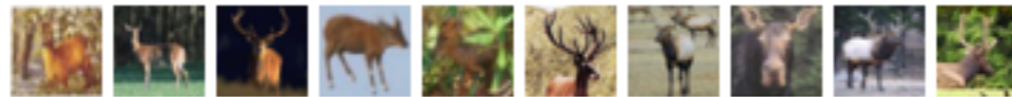
bird



cat



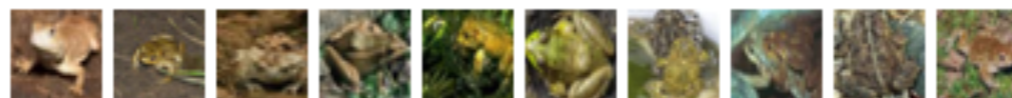
deer



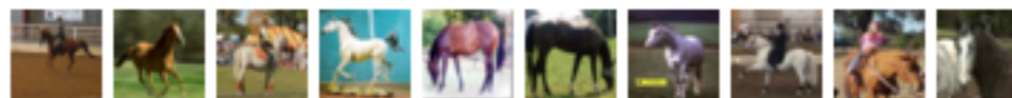
dog



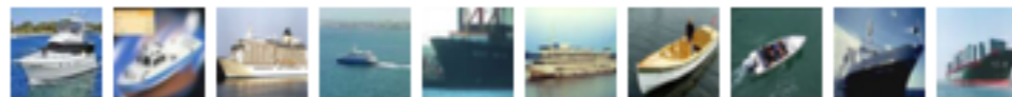
frog



horse



ship



truck





## APPENDIX: DATA — CFAR-100

- ▶ 100 class variant on the CFAR10 sample:
- ▶ 32x32 colour images (so each image is a tensor of dimension 32x32x3).

- ▶ 100 classes:

**Superclass**

aquatic mammals  
fish  
flowers  
food containers  
fruit and vegetables  
household electrical devices  
household furniture  
insects  
large carnivores  
large man-made outdoor things  
large natural outdoor scenes  
large omnivores and herbivores  
medium-sized mammals  
non-insect invertebrates  
people  
reptiles  
small mammals  
trees  
vehicles 1  
vehicles 2

**Classes**

beaver, dolphin, otter, seal, whale  
aquarium fish, flatfish, ray, shark, trout  
orchids, poppies, roses, sunflowers, tulips  
bottles, bowls, cans, cups, plates  
apples, mushrooms, oranges, pears, sweet peppers  
clock, computer keyboard, lamp, telephone, television  
bed, chair, couch, table, wardrobe  
bee, beetle, butterfly, caterpillar, cockroach  
bear, leopard, lion, tiger, wolf  
bridge, castle, house, road, skyscraper  
cloud, forest, mountain, plain, sea  
camel, cattle, chimpanzee, elephant, kangaroo  
fox, porcupine, possum, raccoon, skunk  
crab, lobster, snail, spider, worm  
baby, boy, girl, man, woman  
crocodile, dinosaur, lizard, snake, turtle  
hamster, mouse, rabbit, shrew, squirrel  
maple, oak, palm, pine, willow  
bicycle, bus, motorcycle, pickup truck, train  
lawn-mower, rocket, streetcar, tank, tractor




## APPENDIX: DATA — KAGGLE

- ▶ Well known website for machine learning competitions; lots of problems and lots of different types of data.
- ▶ Also includes training material at:
  - ▶ <https://www.kaggle.com/learn/overview>
  - ▶ e.g. Intro to machine learning includes a data science problem on [predicting titanic survivors](#) from a limited feature space.
    - ▶ Since the outcome is known, this is a good sample of real world data to try out your data science skills.

Getting Started Prediction Competition

### Titanic: Machine Learning from Disaster

Start here! Predict survival on the Titanic and get familiar with ML basics

 Kaggle · 11,175 teams · Ongoing

[Overview](#) [Data](#) [Notebooks](#) [Discussion](#) [Leaderboard](#) [Rules](#) [Join Competition](#)



## APPENDIX: DATA — UCI ML DATA REPOSITORY



- ▶ Hundreds of data sets covering life sciences, physical sciences, CS / Engineering, Social Sciences, Business, Game and other categories of data.
- ▶ Different types of problem: including Classification, regression and clustering samples.
- ▶ Different types of data: e.g. Multivariate, univariate, time-series etc.

▶ <https://archive.ics.uci.edu/ml/datasets.php>



## APPENDIX: DATA — TIMIT

- ▶ A corpus of acoustic-phonetic continuous speech data, provided with extensive documentation.
- ▶ Includes audio files and transcripts
- ▶ 630 speakers, each with 10 sentences, corresponding to a corpus of 25200 files (4 files per speaker).
- ▶ Total size is approximately 600Mb.

<https://catalog ldc.upenn.edu/LDC93S1>



## APPENDIX: DATA — RCV1-V2

- ▶ RCV1: A New Benchmark Collection for Text Categorization Research
- ▶ A detailed description of this text categorisation data set can be found in: <http://www.jmlr.org/papers/volume5/lewis04a/lewis04a.pdf>

[http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004\\_rcv1v2\\_README.htm](http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm)



# MISCELLANEOUS - SCRIPTS

▶ The following scripts exist for this tutorial

<input type="checkbox"/>	<a href="#">api.py</a>
<input type="checkbox"/>	<a href="#">BatchSizeNN.py</a>
<input type="checkbox"/>	<a href="#">CNN.py</a>
<input type="checkbox"/>	<a href="#">DropoutNN.py</a>
<input type="checkbox"/>	<a href="#">LeakyReluScanNN.py</a>
<input type="checkbox"/>	<a href="#">LinearRegression.py</a>
<input type="checkbox"/>	<a href="#">NN.py</a>
<input type="checkbox"/>	<a href="#">NN_para_load.py</a>
<input type="checkbox"/>	<a href="#">NN_para_save.py</a>
<input type="checkbox"/>	<a href="#">NN_parabola.py</a>
<input type="checkbox"/>	<a href="#">README.md</a>
<input type="checkbox"/>	<a href="#">runAll.sh</a>
<input type="checkbox"/>	<a href="#">SK_BDT.py</a>
<input type="checkbox"/>	<a href="#">SK_DT.py</a>
<input type="checkbox"/>	<a href="#">SK_RF.py</a>
<input type="checkbox"/>	<a href="#">SK_SVM.py</a>
<input type="checkbox"/>	<a href="#">ValidationSplitNN.py</a>

These parallel the jupyter notebooks

There are some notable additions including

- saving models: [NN\\_para\\_save.py](#)
- loading models: [NN\\_para\\_load.py](#)

You will also find some introductory SciKitLearn examples to complement the TensorFlow examples.