Queen Mary
University of London

DR ADRIAN BEVAN

PRACTICAL MACHINE LEARNING

FUNCTION APPROXIMATION WITH NEURAL NETWORKS

Assessed Problem

LECTURE PLAN

- Context
- The problem
- Describe code examples that you need to modify
- Timeline

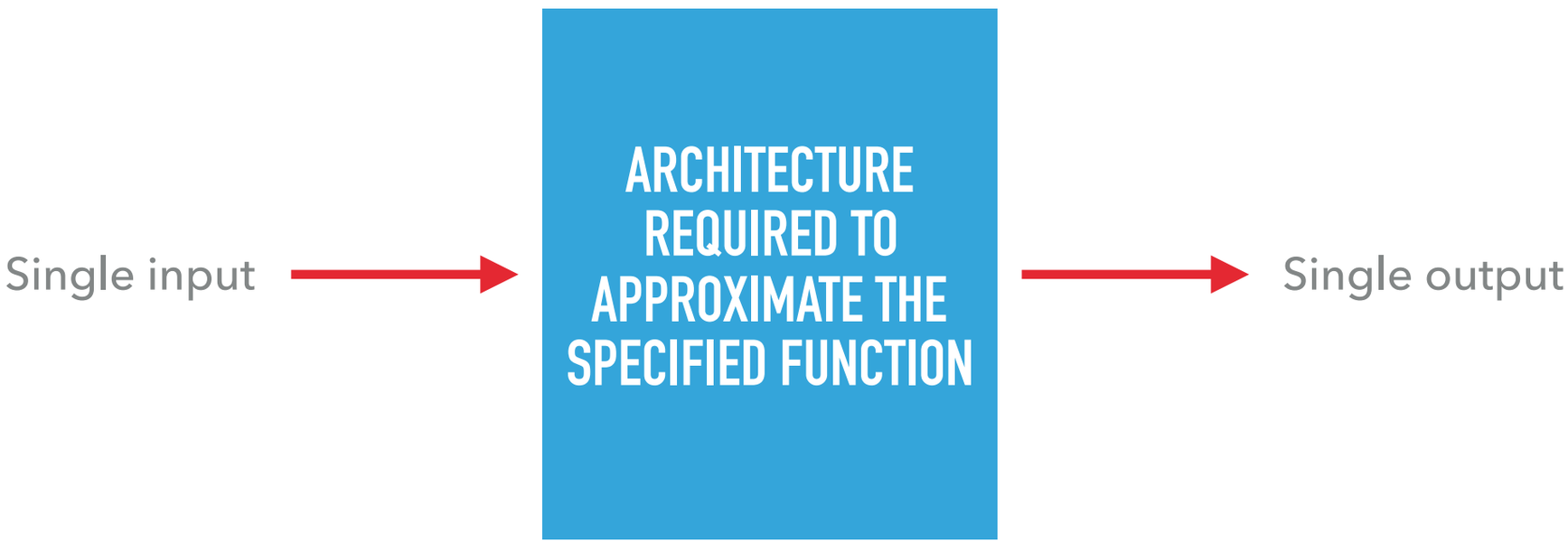
QMUL Summer School: https://www.qmul.ac.uk/summer-school/
Practical Machine Learning QMplus Page: https://qmplus.qmul.ac.uk/course/view.php?id=10006

CONTEXT

- We have discussed machine learning using neural networks, including regression and classification problems.
- This is a regression based problem that will require the use of algorithms discussed so far, and you will need to bring together skills based on the use of both Python and TensorFlow to solve this.
- An example script is provided, where you are expected to modify functions and apply this to explore various issues regarding training performance.

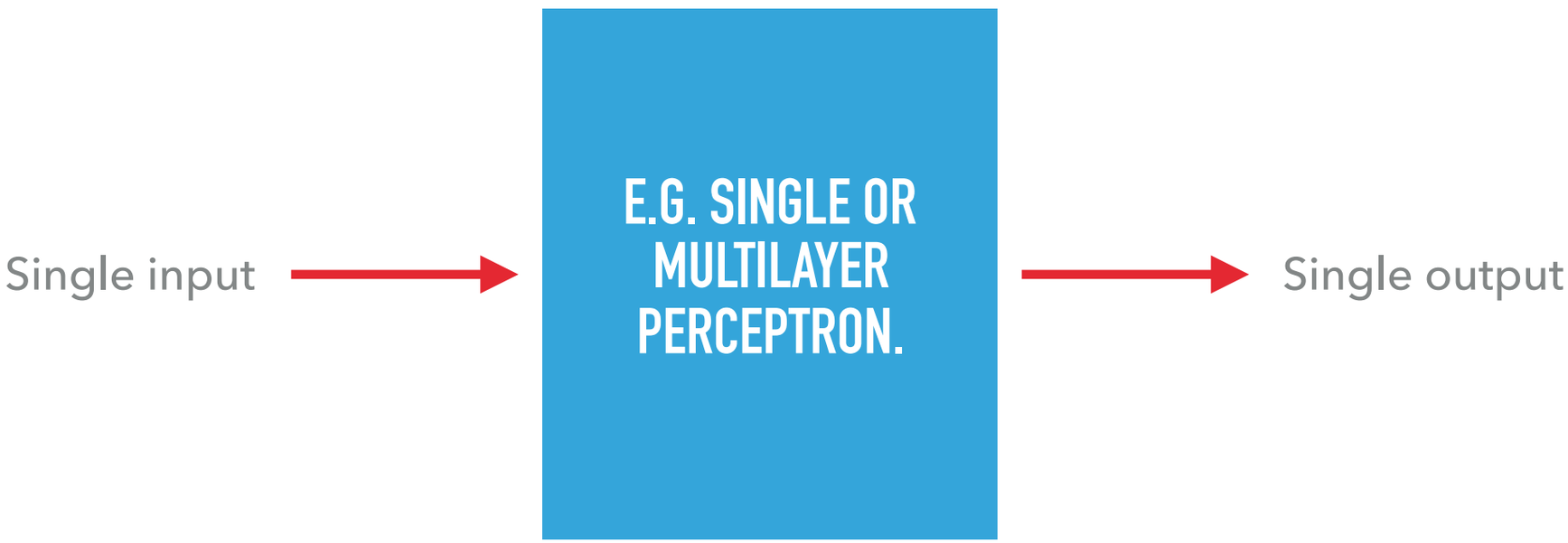
THE PROBLEM

- Reminder: Neural networks are function approximators of the general form (bias implied by w): $y = f(x, w)$ for a single valued input function.
- We have discussed various scenarios of using these algorithms, but here we consider a simple architecture.



THE PROBLEM

- Reminder: Neural networks are function approximators of the general form (bias implied by w): $y = f(x, w)$ for a single valued input function.
- We have discussed various scenarios of using these algorithms, but here we consider a simple architecture.



THE PROBLEM

- Adapt the sample script to train, validate and test an MLP to approximate the following functions:

$$f(x) = x^2$$
$$f(x) = \sqrt{x}$$
$$f(x) = \sin(x)$$
$$f(x, y) = x^2 + y^2$$

The assessed work uses the x^2 function

Additional functions are given in case you complete the assignment quickly.

- For the domain range x∈[-1,1]* and for the last function y∈[-1,1].

- Start from the example script:

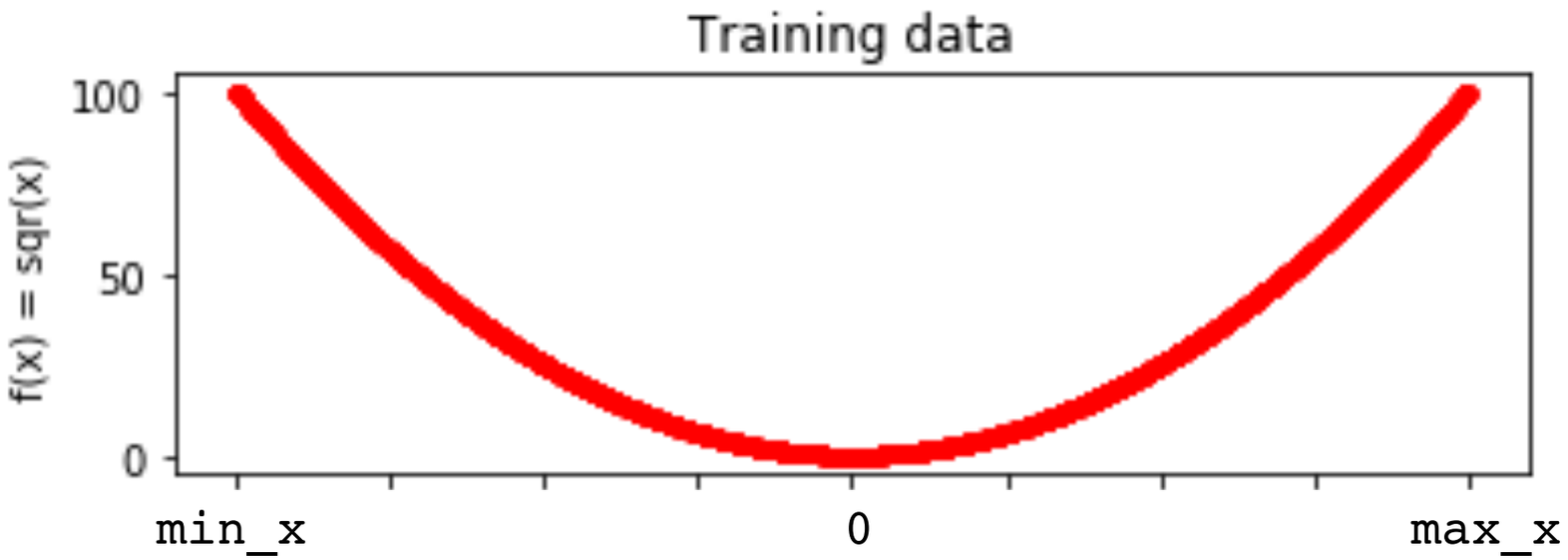
`Example_FunctionApproximator.py`

* for the square root function you will need to adapt this range to x∈[0,1]

EXAMPLE_FUNCTIONAPPROXIMATOR.PY

- Uses tensorflow and matplotlib, numpy is included as this may be useful for you to work through the problem.

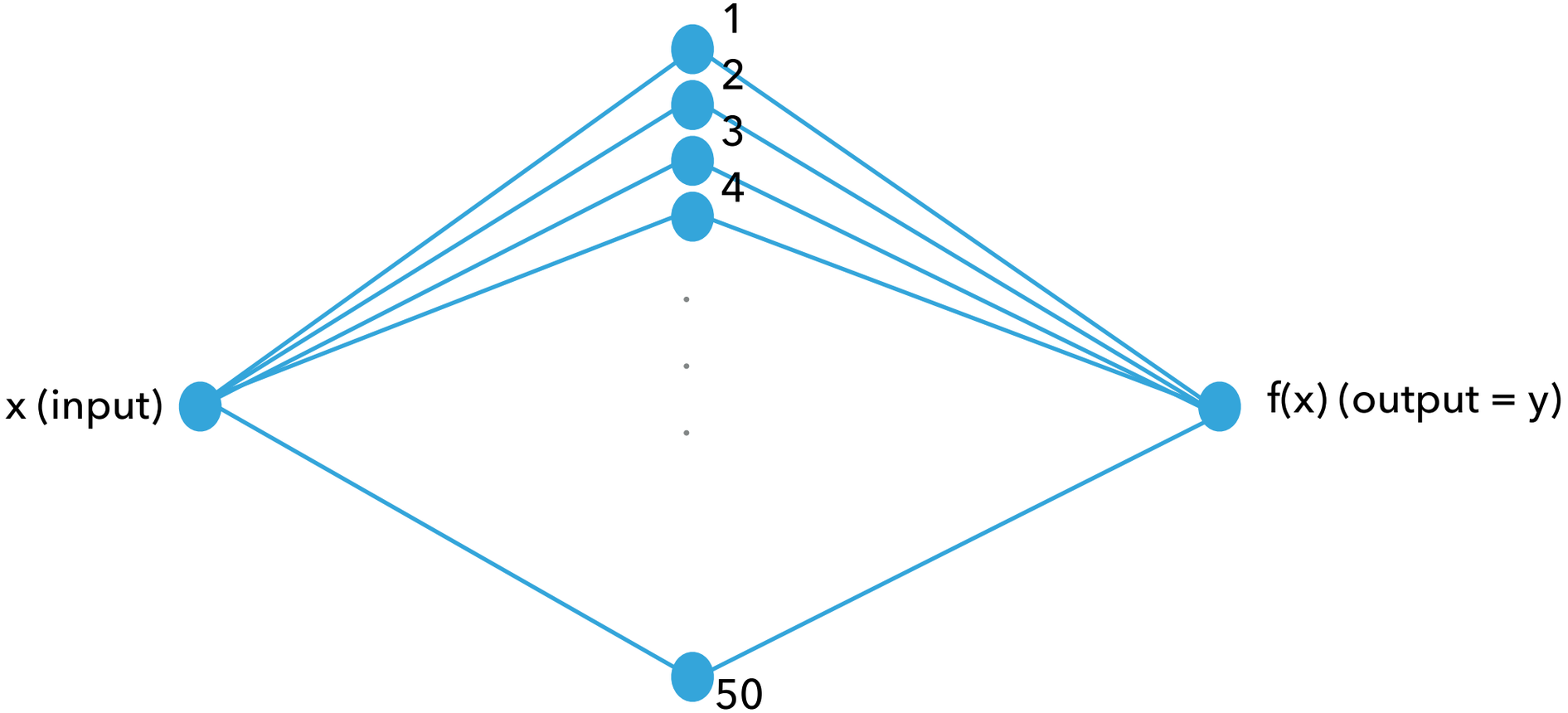
```
# Network training parameters
learning_rate    = 0.01
training_epochs  = 100
min_x = -10
max_x = 10
Ngen   = 10000
```



EXAMPLE_FUNCTIONAPPROXIMATOR.PY

- Uses tensorflow and matplotlib, numpy is included as this may be useful for you to work through the problem.

```
# Network architecture parameters
n_input    = 1   # 1D function, so there is one input feature per example
n_classes  = 1   # Regression output is single valued
n_hidden_1 = 50  # 1st layer num features
```



EXAMPLE_FUNCTIONAPPROXIMATOR.PY

```
def myFunctionTF(arg):
    """
    User defined function for the MLP to learn.  The default example is
    the square root function.
    """
    return tf.square(arg)
```

- The function the perceptron will try and approximate is implemented in the myFunctionTF function.
- The argument is a TF Variable with a definite shape [?, 1] for the 1D approximator problem.
- Use TF ops on tensors, and not math functions (that operate on scalars) to compute the function.

EXAMPLE_FUNCTIONAPPROXIMATOR.PY

- x_ and y_ are the input value and f(x) output (ground truth), respectively

```
x_ = tf.placeholder(tf.float32, [None, n_input], name="x_")
y_ = tf.placeholder(tf.float32, [None, n_classes], name="y_")
```

- The hidden layer is constructed using weights and a bias, along with an activation function (relu is used in the example).

- Weights are randomly initialised

```
w_layer_1    = tf.Variable(tf.random_normal([n_input, n_hidden_1]))
bias_layer_1 = tf.Variable(tf.random_normal([n_hidden_1]))
layer_1      = tf.nn.relu(tf.add(tf.matmul(x_,w_layer_1),bias_layer_1))
```

- Output (probabilities) node is created similarly

```
output        = tf.Variable(tf.random_normal([n_hidden_1, n_classes]))
bias_output   = tf.Variable(tf.random_normal([n_classes]))
probabilities  = tf.matmul(layer_1, output) + bias_output
```

EXAMPLE_FUNCTIONAPPROXIMATOR.PY

- Use the L2 loss function as the figure of merit for optimisation.
- Along with the Adam optimiser algorithm.

```
cost = tf.nn.l2_loss(output_layer - y_)
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

- These will be discussed in the next lecture.
- We want to perform a regression analysis, so set the output probability to the output layer value: probabilities.

EXAMPLE_FUNCTIONAPPROXIMATOR.PY

- Having set the graph up we need to evaluate this for each training epoch.

```
for epoch in range(training_epochs):
    the_cost = 0.

    sess.run(optimizer, feed_dict={x_: traindata, y_: target_value})
    the_cost = sess.run(cost, feed_dict={x_: traindata, y_: target_value})

cost_set.append(the_cost)
epoch_set.append(epoch+1)

the_cost = sess.run(cost, feed_dict={x_: testdata, y_: test_value})
cost_test_value.append(the_cost)
```

ADDITIONAL PROBLEMS [SEE THE WORKSHEET FOR THE ASSESSED PROBLEMS, THESE ARE IDEAS FOR YOU TO WORK ON IF YOU FINISH THAT WORK.

- For each function, document:

    1. Cost evaluated on the test and validation samples as a function of epoch. Use 10, 100, and 1000 training cycles.

    2. The predictions of a test sample of data for a training sample size of 10^3, 10^4, and 10^5 examples using a reasonable number of training epochs.

    3. Compare the performance of a relu activation function with that of a sigmoid activation function.

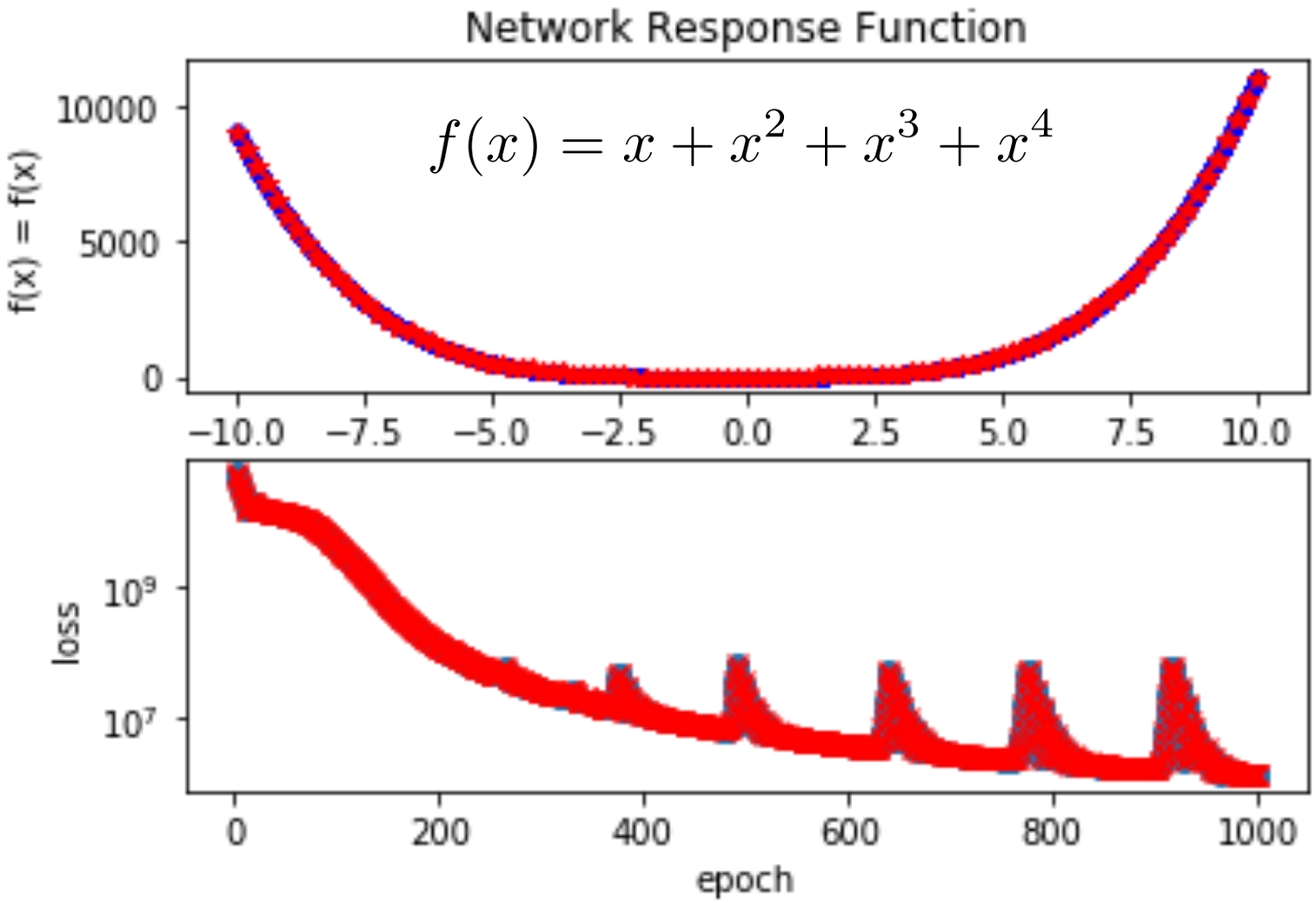
    4. Compare results and training behaviour changing the learning rate from 0.001 through 0.2 in a few discrete steps.

    5. Modify the code to explore the performance difference between a single hidden layer and two hidden layers.

- For one of your functions (choose one with good performance) explore what happens when you train a network on a range of data [-10, 10], but use this to predict over a wider range [-20, 20]

SUMMARY

- The ability for a network to adapt to a given function depends on the number of training cycles and the number of training examples presented to it.
- This set of exercises allows you to get experience with variations in training performance as a function of these quantities.



Using a relu activation function and two hidden layers.

The network is able to provide reasonable predictions over the range [-10, 10]