Assignment 3

This assignment may be completed individually or in groups of 2 or 3.
You are recommended to use your project groups. If you are in a group, **one student** will submit all necessary files and the **other student(s) will submit a text file** specifying members of the group and who is submitting. The report must have **all students' names and IDs**.

In this assignment, you will build and train a feed-forward neural network using TensorFlow in two different problem domains. Each question has several parts which detail what is to be included in your report.

Question 1

You may use the `mlp.py` model provided although this is not mandatory. This exercise deals with the approximation of functions by neural networks. The so called function approximation (regression), is to find a mapping f' satisfying || f'(x) - f(x) ||< ε, (ε is the tolerance and the ||.|| represents the L1 norm). When dealing with this error experimentally we will generally use the root mean square error in any cost function that we choose to minimize. In general, it is sufficient to have a single layer of nonlinear neurons in a neural network in order to approximate a nonlinear function.

The goal of this exercise is then to build a feedforward neural network that approximates the following function:

$$f(x,y) = cos(x + 6*0.35y) + 2*0.35xy \quad x,y \in [-1\ 1]$$

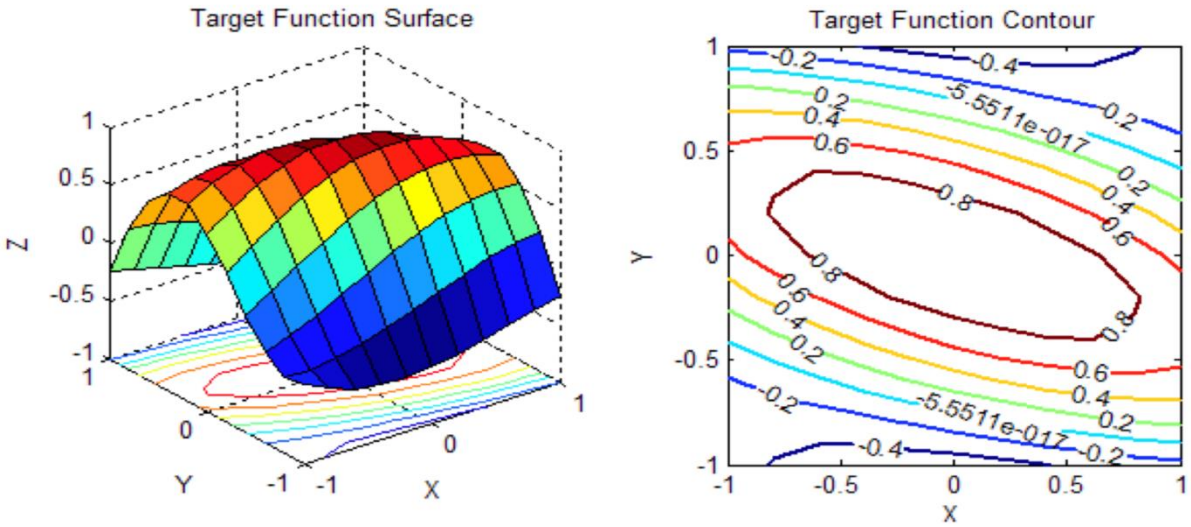


Fig. 1: Parametric surface and contour of the target function

Preparation of data

For this function approximation problem, three kinds of data sets are prepared, namely the training set, the validation set and the test set. The training set is a set of value pairs which comprise information about the target function for training the network. The validation set is associated with the early stopping technique, described below. During the training phase, the validation error is monitored in order to prevent the network from overfitting the training data. Normally, the test set is just used to evaluate the network performance afterwards. But, in this exercise the root mean-square error (rmse) on the test set is used as the performance goal of the network training.

For the current problem, the training and the test data are taken from uniform grids (10x10 pairs of values for the training data, 9x9 pairs for the test data). As shown in Fig.1 the range of the function output is already within the interval [-1 1]. So, it is not necessary to scale the target function. For the validation data, in order to make it a better representation of the original function, it is taken randomly from the function surface.

Network Design

Theoretical results indicate that given enough hidden (non-linear) units, a feedforward neural network can approximate any non-linear functions (with a finite number of discontinuities) to a required degree of accuracy. In other words, any non-linear function can be expressed as a linear combination of non-linear basis functions. Therefore, a two-layer feedforward neural network with 2 layers: one layer of non-linear hidden neurons and one linear output neuron seems a reasonable design for a function approximation task. The target function as defined above has two inputs (x, y), and one output (z = f(x,y)). Thus, as shown in Fig.2, the network solution consists of two inputs, one layer of sigmoid transfer (aka activation) function neurons and one linear transfer function output neuron. You may also want to consider using the hyperbolic tangent activation function for the hidden layer.

The number of the hidden neurons is an important design issue. On the one hand, having more hidden neurons allows the network to approximate functions of greater complexity. But, as a result of network's high degree of freedom, it may overfit the training data while the unseen data will be poorly fit to the desired function. On the other hand, although a small network won't have enough power to overfit the training data, it may be too small to adequately represent the target function. In order to choose a reasonable amount of hidden neurons, three different networks with 2, 8 and 50 hidden neurons are examined. The training result (see Fig.3) shows the network with 8 hidden neurons outperforms the other two networks after they are trained with the same training parameters. Here, the number of epochs to convergence will generally vary.

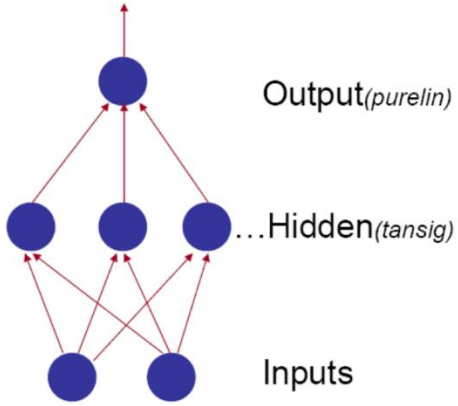


Fig.2: Network Architecture

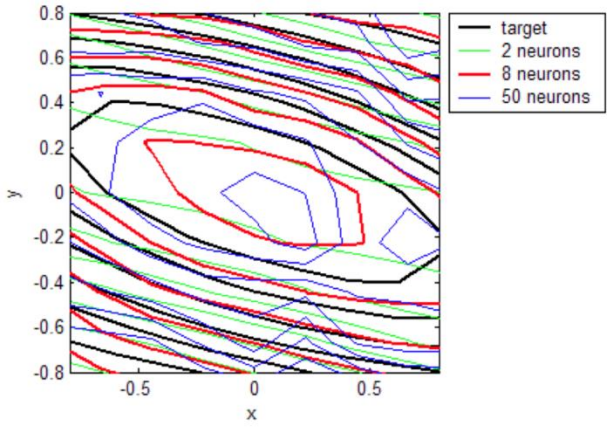


Fig.3: Function Contours

Part (a). You must:

1. Using the *GradientDescentOptimizer*, investigate the converged performance of a 2 layer neural network with 2, 8 and 50 hidden layer neurons.
2. You must produce a contour diagram similar to Fig.3.
3. Include a table of MSE for the 3 different network sizes and number of epochs to convergence.
4. Indicate whether sigmoid or hyperbolic tangent activation functions were used in your experiments.

Training

You are to investigate several forms of training. There are a number of batch training algorithms which can be used to train a network. In this exercise, the following training algorithms are examined.

- `traingd` implements a basic gradient descent algorithm (*GradientDescentOptimizer*). It updates weights and biases in the direction of the negative gradient of the performance function. The major drawback of `traingd` is that it is relatively slow (especially when the learning rate is small) and has a tendency to get trapped in local minima of the error surface (where the gradient is zero).
- `traingdm` improves `traingd` by using momentum during the training (*MomentumOptimizer*). Momentum allows a network to ignore the shallow local minimum of the error surface. In addition, `traingdm` often provides a faster convergence than `traingd`.
- `traingrms` provides a more sophisticated momentum-based technique (*RMSPropOptimizer*). In addition, `trainrms` often provides a faster convergence than `traingd`.

In order to examine the performance of the training functions mentioned above, they are applied to the two-layer feedforward network respectively with the performance goal (MSE= 0.02 for the training set), maximum number of epochs to train (100) and the learning rate (0.02) being the same (without using early stopping).

Part (b). You must:

1. Include a table containing the appropriate number of epochs for convergence for each training method.
2. Plot the variation of MSE against epoch number for each of the 3 methods for 1-100 epochs.
3. Plot a bar chart of the CPU time taken per epoch for each of the 3 methods.
4. Which method provides the best accuracy at the end of 100 epochs of training?
5. Which method is the most accurate when the training error is reached?

Early Stopping

As mentioned above, the number of hidden neurons has a large influence on the behavior of the network. If the size of the network is too large it may run a risk of overfitting the training set and loses its generalization ability for unseen data. One method for improving network generalization ability is to use a network that is just large enough to provide an adequate fit to the target function. But sometimes it is hard to know beforehand how large a network should be for a specific application. One commonly used technique for improving network generalization is early stopping. This technique monitors the error on a subset of the data (validation data) that does not actually take part in the training. The training stops when the error on the validation data increases for a certain amount of iterations.

In order to examine the effect of early stopping on the training process, a randomly generated validation set is used during the `trainrms` training (Maximum validation failures=10, rmse=0.02 for the test set). As indicated by Fig.6, the early stopping mechanism is not triggered during the training. That is because the validation error keeps decreasing during the whole training process. In Fig.7, we can see that both networks (trained with and without early stopping) work equally well on the current approximation problem. This result also indicates that 8 hidden neurons is a good choice for the current problem.

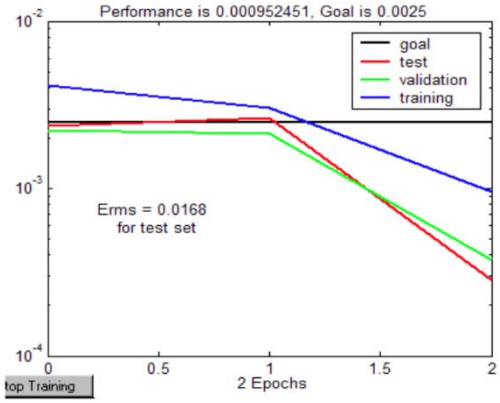


Fig.6: Evolution of the MSE

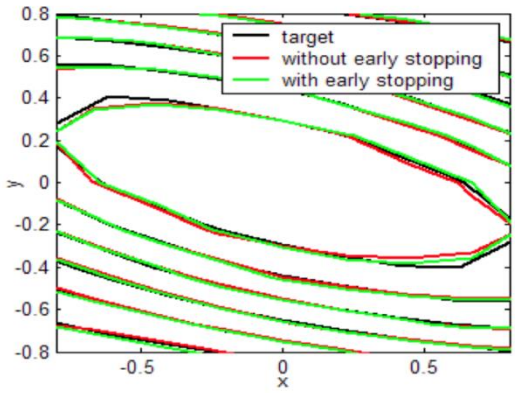


Fig.7: Function Contours

Part (c). You must:

1. Confirm that approximately 8 neurons are a good choice for the current problem.
2. Run experiments across a range of hidden layer sizes and plot MSE for testing set at convergence against hidden layer size.
3. Reproduce (approximately) Fig. 6 and Fig. 7. Note: the epochs axis in Fig. 6 represent the 2 epochs around convergence, not epochs 0-2 of a run.

Question 2

The task of this exercise is to build a multilayer feedforward network for pattern recognition. The network is trained as a character classifier for a collection of characters given as 7x5 black-white pixel maps. Ideally, the trained network can recognize characters it has learnt even when some of them are distorted.

Data Preparation

In general, there are two kinds of data prepared for training and testing the network. One is the collection of thirty-one 35-element input vectors, which represent the target patterns (see Fig.9): 26 capital characters and 5 lower case characters of the name jinyu. Another part of the data is collected by randomly reversing three bits of original characters (see Fig.10). This time, in stead of using early stopping to improve the generalization ability of the network, the network is trained on both parts of the data mentioned above, which enables it response correctly to both ideal and partially corrupted patterns.

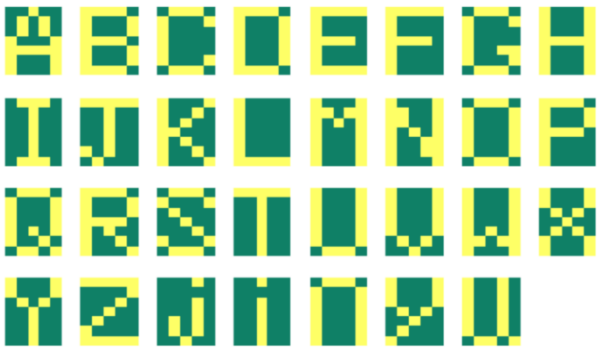


Fig.9: Target Patterns

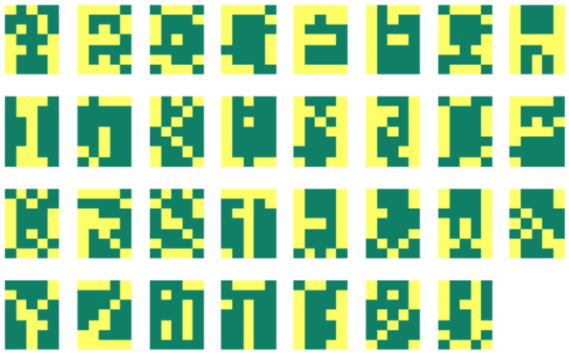


Fig.10: Distorted Patterns

Network Design

In principle, two-layer networks with sigmoidal hidden units can approximate arbitrarily well any functional continuous mapping from one finite-dimensional space to another, provided the number of hidden units is sufficiently large. As the target patterns defined in this exercise is relatively simple, which are defined by only 35 Boolean values. Therefore, a two-layer feedforward network is supposed to be powerful enough for this character recognition task. As 31 target characters are represented by 35-element input vectors, the neural network needs 35 inputs and 31 output neurons (see Fig.11). The network receives 35 Boolean values, which represents one character. It is then required to identify the character by generating an output vector, the element of which with the highest value indicates the class of input character. The logsig (Log Sigmoid) is chosen as the transfer function for both hidden and output layers. This is because it has a suitable output range ([0 1]) for the current problem. It is also okay to use a softmax function on the output layer.

The number of hidden neurons is initially set to be 10. But, the network with 10 hidden neurons has trouble recognizing distorted patterns even if it has been trained on noisy patterns. The percentage of recognition errors is about 21% when it is tested on patterns with 3 bits noise (see Fig.12). Therefore, 5 neurons are added to the hidden layer. As shown in Fig.12, the recognition error rate decreases to an acceptable value (12%). Thus, 15 is chosen as the number of the hidden neurons.

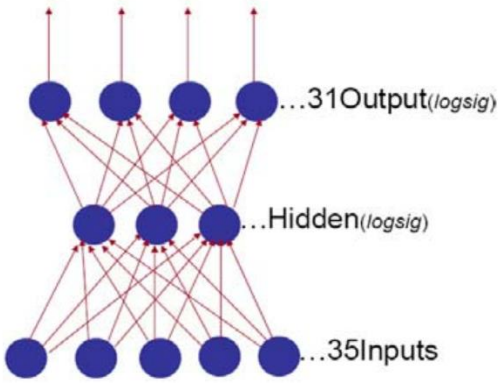


Fig.11: Network Architecture

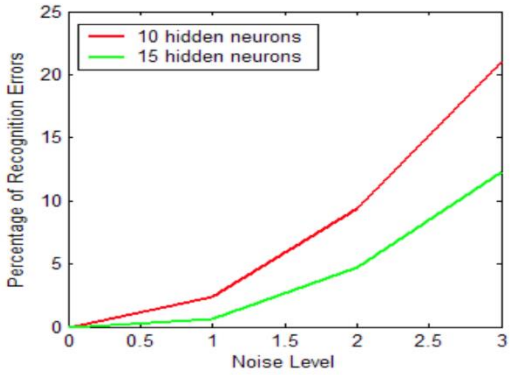


Fig.12: Percentage of Recognition Errors

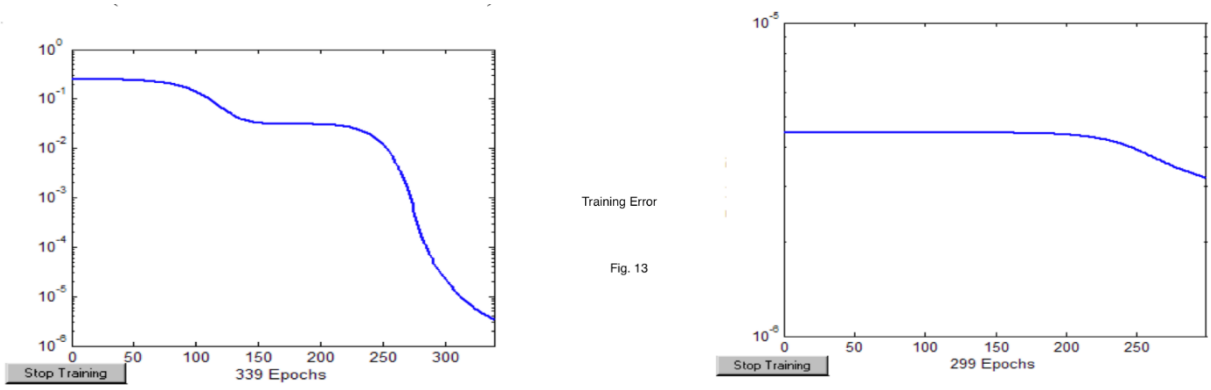
Part (a). You must:

1. Run experiments with hidden neuron numbers in the range 5-25.
2. Plot a chart of recognition error against the number of hidden neurons.

Another important design issue is the choice of the initial weights and bias. In general, weights and bias should be initialized to small values so that the active region of each neuron is not close to the irresponsive (saturate) part of the transfer function; otherwise the network won't be able to learn. Ensure that you use weights normally distributed around zero.

Network Training

After the network is created, it is then ready for training. A gradient decent training function with momentum and adaptive learning rate (traingdx) is chosen to train the network. Use the *AdamOptimizer* for this. For the pattern recognition task, it is important that the noisy patterns can still be correctly classified. Thus, in order to make the network insensitive to the presence of noise, it is trained on not only ideal patterns but also noisy patterns. In the program, a three-step training process is implemented. In the first step, the network is trained on the ideal data for zero decision errors (see Fig.13 (a)). In the second step, the network is trained on noisy data as shown in Fig.10 for several passes (e.g. 10 passes) for a proper performance goal (0.01 is used in the program). Unfortunately, after the network is trained for recognizing noisy patterns, it will probably "forget" those noise-free patterns it has learnt before. Therefore, in order to recall the network of these non-distorted characters, in the final step, it is trained again on just ideal data for zero decision errors (see Fig.13 (b)). The three-step training process mentioned above enables the trained network to identify both noise-free and noisy characters (within a certain error tolerance).



Part (b). You must:

1. Confirm that Fig.13 is a reasonable representation of performance for the optimal number of hidden layer neurons chosen.
2. Plot a chart in support of (1)

Network Testing

Once the network is trained, the test data which consists of both noise-free and slightly distorted patterns are fed to the network to check the training result. Here, the average recognition error rate is used as the performance measure. As shown in Fig.14, the network trained with noise works perfectly on noise-free patterns. And, it outperforms the one trained only on the noise-free data when input patterns are distorted to certain levels.

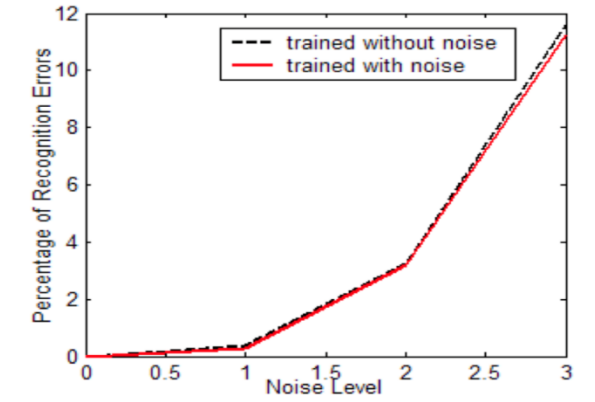


Fig.14: Percentage of Recognition Errors

Part (c). You must:

1. Create testing data that has between 0 and 3 bits noise and include some examples in your submission.
2. Confirm that you can produce the recognition accuracy shown in Fig. 14.