Solutions Exercise 6

Question 1

Lets denote the networks with one hidden layer as $\mathbf{N}_1$ and the other network as $\mathbf{N}_2$. Note that all the calculations involves the bias term.

Number of parameters: Let N be the number of neurons in layer (L-1), and M be the number of neurons in layer L. So total number of parameters between these layers is simply $(N \times M) + M$, where we add M for biases.

$$\mathbf{N}_1 : \underbrace{(3 \times 13) + (13 \times 2)}_{\text{weights}} + \underbrace{13 + 2}_{\text{biases}} = 80$$

$$\mathbf{N}_2 : \underbrace{(3 \times 5) + (5 \times 8) + (8 \times 2)}_{\text{weights}} + \underbrace{5 + 8 + 2}_{\text{biases}} = 86$$

Number of operations in feed-forward step: Let N be the number of neurons in layer (L-1), and M be the number of neurons in layer L. The matrix product of weight parameters and the activations of the (L-1)th layer results in $(2N - 1) \times M$ operations(additions and multiplications), after adding biases we get $2N \times M$(additions). Next, applying sigmoid activation functions, which results in $3M$ operations(sigmoid function), we get $(2N + 3)M$ operations between the layers L-1 and L. Finally, the number of operations in our networks can be given as follows:

$$\mathbf{N}_1 : ((2 \times 3 + 3) \times 13) + ((2 \times 13 + 3) \times 2) = 175$$

$$\mathbf{N}_2 : ((2 \times 3 + 3) \times 5) + ((2 \times 5 + 3) \times 8) + ((2 \times 8 + 3) \times 2) = 187$$

Number of operations in backprop step: First, the output layer with N_0 neurons results in $3 \times N_0$ operations i.e. 2 operations for $\sigma(1 - \sigma)$ and 1 for the derivative of the loss. Followed by propagating the errors which require $(2N + 1)M$ operations: using $(2N - 1) \times M$ operations from multiplying weight matrix with error vector and $2N$ operations for derivative of sigmoid. Now total number of operations required for back-prop (or to get the derivatives of biases) are:

$$\underbrace{(2 \times 3)}_{\text{operations in output layer}} + \underbrace{((2 \times 2 + 1) \times 8) + ((2 \times 8 + 1) \times 5) + ((2 \times 5 + 1) \times 3)}_{\text{operations required for back-propagating errors}} = 164$$

In order to calculate the derivatives of weights, we multiply the activations of layer L-1 with errors of layer L, to get

$$\underbrace{(3 \times 5) + (5 \times 8) + (8 \times 2)}_{\text{operations required to get gradients of weights}} = 71$$

Finally, total number of gradient operations in $\mathbf{N}_1$ are $164 + 71 = 235$.

Similarly in case of $\mathbf{N}_2$, for back-prop we have

$$(2 \times 3) + ((2 \times 2 + 1) \times 13) + ((2 \times 13 + 1) \times 3) = 152$$

followed by derivatives of weights

$$(3 \times 13) + (13 \times 2) = 65.$$

Now total number of operations are $152 + 65 = 217$

Note that, the number of parameters and computational steps were quite comparable for both networks, but in general networks ,with the same number of hidden nodes, can have very different number of parameters depending on the structure.

Question 2

The task in this question is to build a network which computes the product i.e. given the input x, y, the output of the neural network is $x \times y$. In case both the numbers are positive, we can always apply log to the input values x and y followed by exp function to get the output. Figure 1 illustrates the network described above. In order to include the sign of the product, we build a separate parallel network with sign as activation function, as shown in Figure 2. If the output of this network is 0, the product is negative else its positive. In this case the network has two outputs, where one output gives us the absolute value of the product and other output gives the sign of the product. Finally, the network which solves the network using just one output is given in Figure 3.

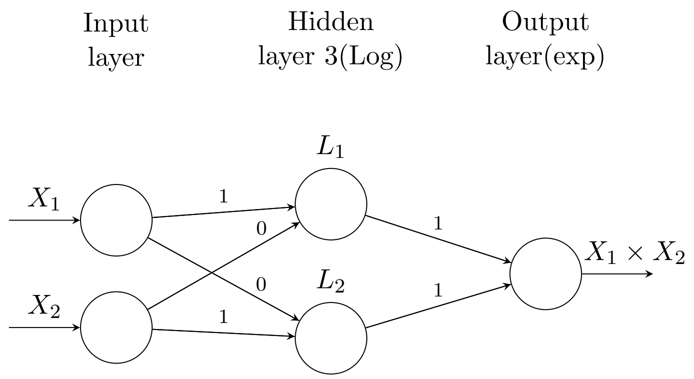

Figure 1: The network shows a way to construct the neural network, for positive integers X_1 and X_2, such that the output of the entire network is $X_1 \times X_2$. The positive input values are fed into hidden layer with log activations followed by output layer with exp as activation function. The resulting output is basically the product of two positive numbers.

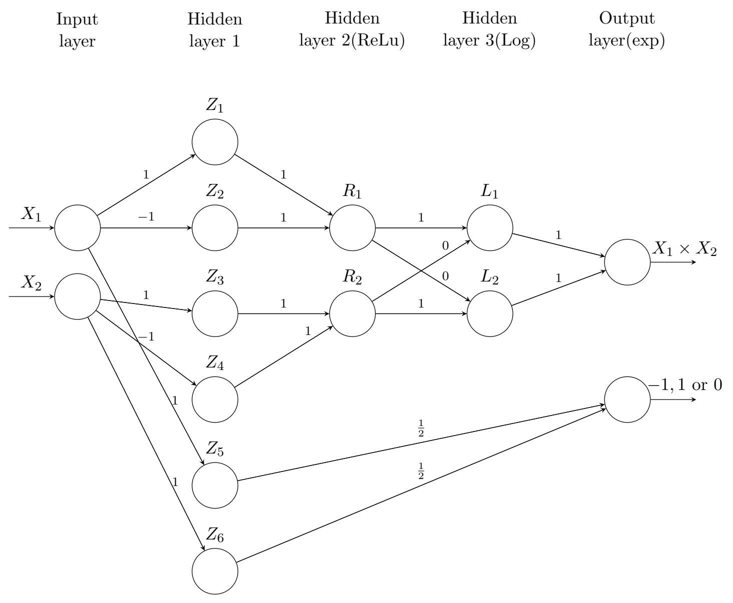

Figure 2: The figure shown is an extension of network in Figure 2. The additional neurons are Z_5 and Z_6 , with Sign activation functions, followed by output node. The output node gives a value of 0 for negative product. Now the network learns both sign and absolute value of the product given the inputs.

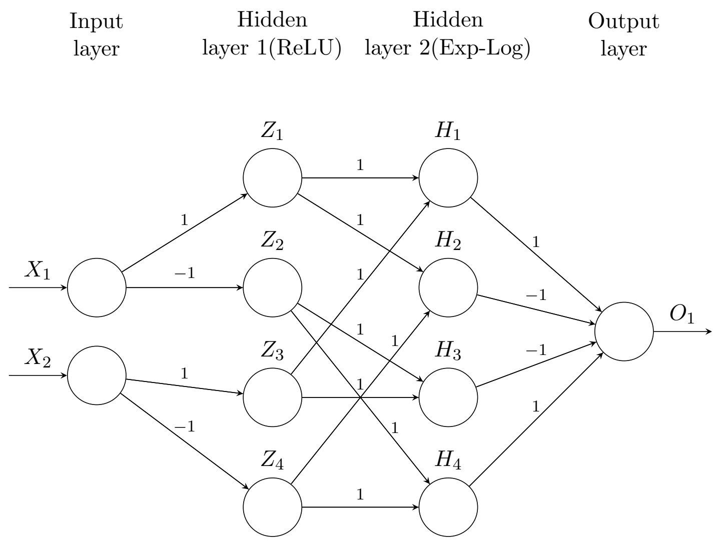

Figure 3: This network solves the product using single output node. Note that, we skipped a layer or two between hidden layer 1 and 2. The skipped part is basically the network described in Figure 1. The output is the product of two inputs and it is negative if the inputs have opposite signs and positive if the inputs have same signs.

Question 3

There are of course many network configurations that may solve the problem, here is one example that typically reaches an accuracy around 91%:

```
# Input layer with ReLU and dropout
model.add(Dense(50, input_dim=28*28))
model.add(Activation('relu'))
model.add(Dropout(0.2))

# Another layer with dropout
model.add(Dense(50))
model.add(Activation('relu'))
model.add(Dropout(0.2))

# Output layer, softmax to get probability distribution
model.add(Dense(10, activation='softmax'))
```

The full notebook solution can be found in Moodle.

Question 4

Full solution in Moodle. This one typically gets above 94% accuracy:

```
# Convolution + ReLU
model.add(Conv2D(32, kernel_size=(3, 3), padding='valid', input_shape=input_shape))
model.add(Activation('relu'))

# Second level of Convolution + ReLU and including a max pooling layer
model.add(Conv2D(32, kernel_size=(3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Next we Flatten to get a 1-D vector
model.add(Flatten())

# A fully-connected layer with drop-out
model.add(Dense(units=100))
model.add(Activation('relu'))
model.add(Dropout(0.5))

# Output layer, softmax to get probability distribution
model.add(Dense(units=nb_classes))
model.add(Activation('softmax'))
```