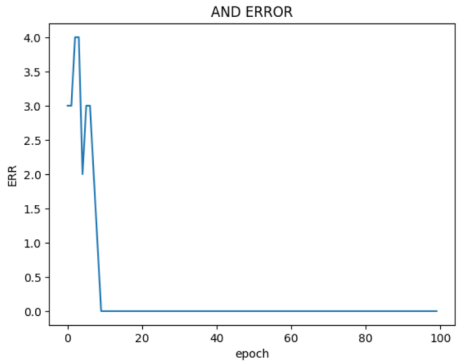
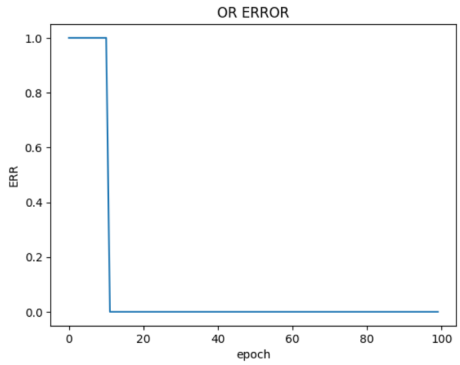
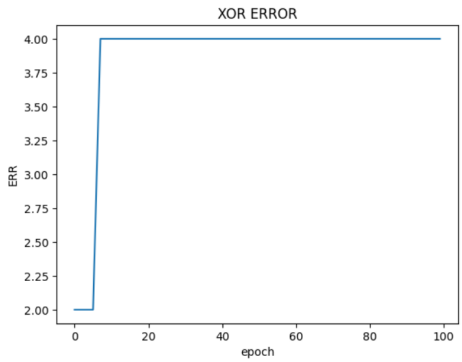
Assignment 1: Multi Layer Perceptron

10 March 2023

1 Answers







1.

2. The model requires 10 input neurons as the data set has 10 features.

3. The model requires 10 output neurons for 7 output classes.

4. As the question at hand is a simple classification, the network model will be having a single hidden layer. This hidden layer is going to have hidden neurons within the range of 8-10, around the mean of input and output layer's neuron number.

   It isn't optimal to have a very low number of neurons in the hidden layer as it poses a threat for correct identification of patterns in the input data: under-fitting.

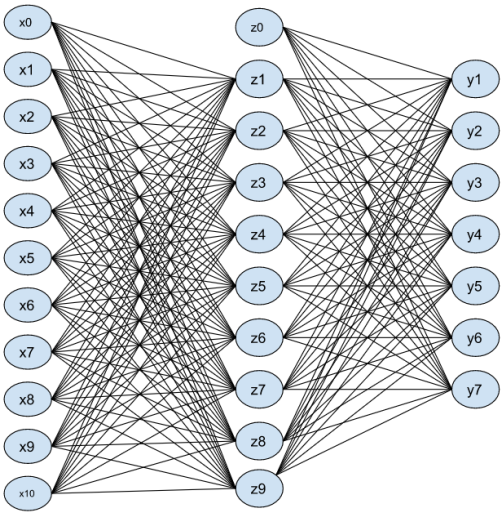
   On the other hand, having a very large number of neurons is also not ideal as—vice-versa—it makes over-complexing and over-fitting a very potential case.

   This is why we decided that the number of neurons in hidden layer should be between the number of neurons in input layer and output layer.

5. We will use **Sigmoid** activation function. The idea of the Sigmoid function is that it is gives an iterable output, which is a probability. Another important aspect is that it has a smooth derivative, which makes it easier to train the neural network using gradient-based optimization algorithms—in our case the back propagation.

6. In the image below, the neurons labelled with x are input neurons, whereas those labelled with y are output neurons. x0 and z0 are the biases for the layers.

   Each connection has its own assigned weight however for the purpose of clarity and readability, they are not displayed.



7. The program randomly shuffles the data indices and divides the data-set into training(70% of the original data-set), validation(15%), and test(15%) sets.

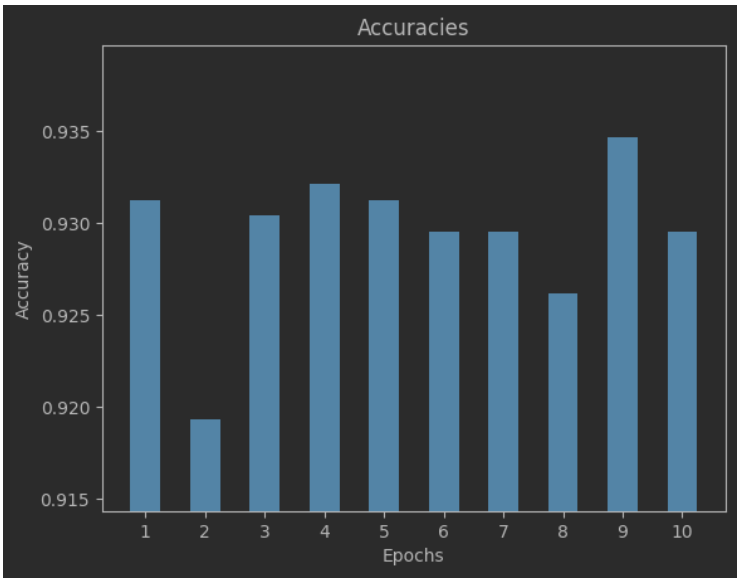
   The ***random shuffling*** and ***partitioning*** are critical to ensure an unbiased estimate of performance. Random shuffling helps to prevent any order bias that may exist in the data, and partitioning helps to prevent over-fitting and provides an estimate of how well the model can generalize to new data.

8. For the performance, the loss function is used. The goal of training an MLP is to minimize the loss function, or in other words, reducing the difference between the predicted and actual output. In the case at hand, categorical cross-entropy is the best fit as a loss function as it is relatively easy to optimize and is suitable for a wide range of classification problems.

9. The number of epochs was used as a stopping criterion by specifying the maximum number of epochs to train the model for. Once the maximum number of epochs has been reached, the training process is terminated, and the model is evaluated on the test set. After conducting numerous trials with different epochs, we were able to determine the optimal number of epochs needed to train the model for maximum accuracy. It should be mentioned that, if the program stopped training earlier, it indicates that the model was not trained enough, whereas if it stopped later, it suggests that overfitting may have occurred.

10. For the initialization of weights, the choice of random numbers were chosen, based on a standard normal distribution (mean = 0, variance = 1). In addition, the initialization of biases was always set to 0. Hence, the initialization of biases has no impact on the performance. On the other hand, the random initial weights surely demonstrated an impact on the performance.

    The bar-chart below demonstrates the 10 different accuracies after training the MLP 10 times, each time with different initial weights, due to randomness. It should also be taken into consideration that the training of the model was done with the most appropriate number of neurons in the hidden layer.



It can be deduced that the accuracy ranged from 0.9194 to 0.9346, meaning that the randomness of initialization of weights had impact on accuracy at most 1.65%.

11. We plotted the performance of our network over different number of neurons in hidden layer and chose the value that gave the highest accuracy.

12. For each hyperparameter, we trained our MLP and generated the accuracy over a range of values. With the accuracies we obtained for different values of the hyperparameter, we plotted graphs and picked the values that gave the highest accuracy. Our architecture uses 35 as the epoch number, 9 as the number of hidden layers, 17 as the number of neurons in each hidden layer.

13. The success rate of our network on the test set is approximately 92%. We calculated this by comparing the output classes that our network predicted with the actual output classes of the test set. This rate shows that our network is correctly trained with the training set and can be used to get an accurate result on the unknown data.

14. Since the training and test set are randomized in the beginning, for every differently trained MLP there will be different confusion matrices. This is one of the confusion matrices for our test set.

```
[[158   0   3   1   2   2   0]
 [  0 156   3   0   0   4   6]
 [  8   1 161   0   3   2   1]
 [  3   0   1 130   1   1   2]
 [  0   1   6   2 165   0   0]
 [  1   3   1   2   6 183   2]
 [  3   2   3   7   0   2 141]]
```

The values on the diagonal show the number of true positives, meaning the number of instances that were predicted correctly for each class. So for example, in column 1, the number 158 represent the number of instances that were predicted to be in class 1 and actually belong to class 1. The values above and below the diagonal represent the instances that are misclassified. For example, the value 8 on the first column third row means that there are 8 instances that are predicted to be in class 1 but actually belong to class 3. According to this confusion matrix, since the number of misclassified instances is greatest in column 3 (with 17 misclassified instances), our network makes the most mistake in classifying from class 3.

15. (15)

16. From the Scikit, we observed that the optimal values for hidden layer size is around 15 neurons per layer, learning rate as 0.005. However, for batch-size we had used 16, and from the values we tried Scikit gave 5 as the optimal number. In the graphs of Toolbox, we see that the validation accuracy increases as the epochs increase and after a point it flattens. We plotted the accuracy vs different number of epochs in the range of 10-70 epochs so we don't observe the initial increase in the graph, we just observe an almost flat line around 90 percent accuracy. As for the observations in confusion matrix, the ratios of true predictions and false predictions are similar in two networks.

17. The difference we got in our performance when we changed the batch size didn't have a major effect. We observed a performance around 90% in both cases. The reason Scikit suggested a different batch size could be because of the different architecture of our networks.

18. An example of misclassification is racial discrimination in job hiring processes. If biased ANNs are used to review the resumes of applicants, the applicants may be misclassified according to their race or ethnicities. This will result in the applicants being wrongly classified as qualified or unqualified which will affect careers of these people and the company's job performance. It will also result in social injustice.

19. To mitigate the harm done, we can expand the input data used to train the ANN to include more diverse data, or remove the feature that causes

the bias and do a blind evaluation. However, we cannot be sure that these will solve the problem completely, because the biased feature (for the example above: race of the applicant) may affect other features and this affect may not be clearly visible. So, even if that feature is removed or more diverse data for that feature is added, there may be bias ingrained in other features.

20. First, to reduce the image dimensionality to 4x4, we use max pooling. This means we choose the greatest value in each 4x4 box in the input matrix. This will give us the following image.

2 6 6 2
6 8 8 4
6 8 8 4
2 4 4 2

Then we will use sharpen kernel without padding. To use the sharpen kernel, we multiply our 3x3 kernel with each 3x3 box in our matrix. So for example, when we align the kernel to the top left of the matrix, we will do the following calculation to reach the result of 12. 2*0 + 6*-1 + 6*0 + 6*-1 + 8*5 + 4*-1 + 6*0 + 8*-1 + 8*0 = 12 We will move the kernel along the matrix, aligning it to the corners and do the same multiplications to obtain the following 2x2 matrix.

12 14
14 16

Then we add numbers in each row and get a matrix of 2x1. Our inputs will be the numbers in each row, therefore 26 and 30.

21. Our inputs for x1 and x2 are 26 and 30, respectively. We use ReLU as activation function, we means if the input to the activation function is greater than zero we return this input, if it's less than zero, we return zero. To calculate neurons z1, we multiply x1 and x2 with their corresponding weights and add the bias, x0. So, z1 = x1*1.5 + x2*-2 + x0*5 = 26*1.5 + 30*-2 + 0*5 = -21. For z2 = x1*-1 + x2*1 + x0*5 = 26*-1 + 30*1 + 0*5 = 4. Applying the ReLU for z1 gives us 0 and for z2 it gives us 4. To calculate neuron y we will multiply z1 and z2 with their corresponding weights and add bias z0. Therefore, y = z1*1 + z2*1.5 + z0*5 = 0 + 6 + 0 = 6. Again, applying ReLU to neuron y, we will obtain the result 6.