cs577 Assignment 1

Girish Rajani-Bathija
A20503736
Department of Computer Science
Illinois Institute of Technology
September 20, 2022

1. Problem Statement - Tensorflow Playground

The problem is to train classifiers for all 4 datasets in the tensorflow playground by having very similar training and testing loss, successfully classifying the data and avoiding overfitting while keeping the models simple. Hyperparameters and network architecture should be reported.

2. Proposed Solution - Tensorflow Playground

Use different network architectures and hyperparameters to classify the datasets. Report Results.

3. Implementation Details - Tensorflow Playground

Dataset	Epochs	Learning Rate	Activation	Regularization	Regularization Rate	Training Loss	Test Loss
Circle	97	0.1	Tanh	None	0	0.002	0.003
Exclusive or	100	1	Sigmoid	None	0	0.001	0.001
Gaussian	36	0.1	Sigmoid	L2	0.001	0.008	0.008
Spiral	504	0.01	Tanh	L1	0.001	0.011	0.016

Dataset	Feature	# of Hidden Layers	Hidden Layer 1	Hidden Layer 2	Ratio of Training to Testing Data	Batch Size
Circle	X1, X2	2	5 neurons	3 neurons	60%	20
Exclusive or	X1, X2	2	5 neurons	4 neurons	60%	20
Gaussian	X1, X2	2	4 neurons	3 neurons	60%	20
Spiral	X1, X2, X1^2, X2^2	2	6 neurons	4 neurons	60%	6

Figure 1 - Showing hyperparameters selected, network architecture, and results for 4 datasets

One of the problems encountered with the Spiral dataset was trying to classify the dataset while simultaneously generating a low and close test loss and training loss. In some scenarios, the dataset was classified but the test and training loss were either too high or there was overfitting, in that the training loss was very low with the test loss being relatively high.

To reduce the loss, the batch size was reduced from 20 to 6 and that resulted in a training loss of 0.011 and testing loss of 0.016 while simultaneously classifying the spiral dataset.

4. Results and Discussion - Tensorflow Playground

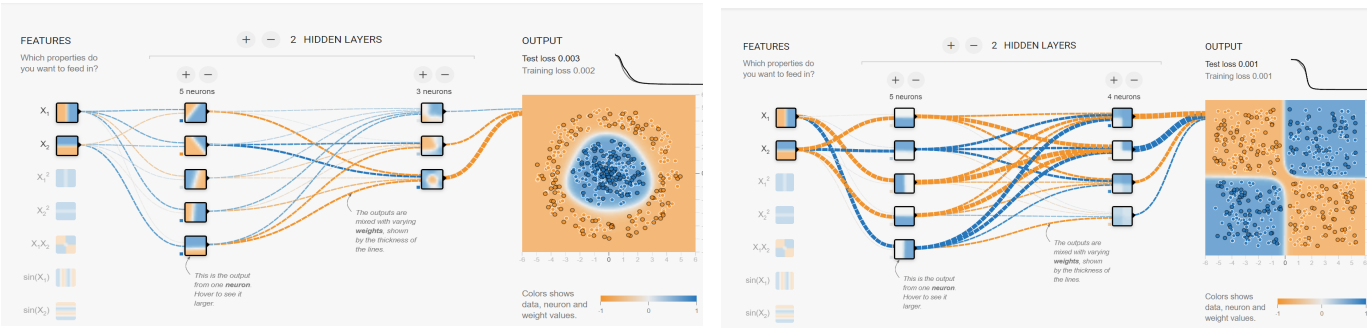


Figure 2 - Showing Results obtained from classifying Circle and Exclusive Or Datasets

As shown in the above figure for the circle dataset classification, through experimentation, it was found that using L2 and L1 regularization made overfitting occur where the training loss was relatively lower than the testing loss. When working with the Exclusive Or dataset, it was also seen that not using any regularization resulted in an optimal output. When using L1 regularization, fluctuations and overfitting occurred. Using Tanh and Relu resulted in lots of fluctuations with the graph and it was not as consistent as sigmoid.

Figure 3 - Showing Results obtained from classifying Gaussian and Spiral Datasets

When working with the Gaussian dataset, through experimentation, the learning rate, and activation affected the results. Sigmoid with a learning rate of 0.01 was found to be one of the best options for optimal learning. Through this, it was possible to get a training and test loss of 0.008. When experimenting with different learning rates and activations, overfitting occurred.

Due to the complexity of the Spiral dataset, it was the most difficult out of the 4 to classify. It was hard to classify the dataset using only the X1 and X2 features with the current model, therefore the X1^2 and X2^2 were added to the architecture. This made it possible to classify the dataset using Tanh. Additionally, it was essential to add a few extra neurons to the hidden layers. When using the same hyperparameters but using 5 neurons instead of 6 in the first hidden layer, it was not possible to classify the dataset. By simply adding 1 more neuron, the model was classified with a training loss of 0.011 and a testing loss of 0.016.

When using activations other than Tanh and learning rates other than 0.01, it was simply not possible to classify the data with the current network architecture. There was simply too much fluctuation. Lastly, L2 regularization was not able to classify the dataset, while no regularization resulted in overfitting the current model. L1 regularization was the best option.

1. **Problem Statement - Cifar10 Dataset**
   Load the cifar dataset with 3 subsets, build a model to perform multi-class classification and plot results. Tune hyperparameters, retrain model and report results.
2. **Proposed Solution - Cifar10 Dataset**
   A model will be built using dense layers with relu activation and then softmax activation for output layer and caategorical_entropy loss function since this is a multi-class classification problem. Hyperparameters will be adjusted to increase performance.
3. **Implementation Details - Cifar10 Dataset**

When building the model to perform multiclass classification on the cifar10 dataset, a few problems occurred in the program design stage. The training and testing data were vectorized using the vectorize_sequence function as shown in the Deep Learning with Python Book by Chollet. However, the results were not satisfying in any way. After training the network for 50 epochs, the accuracy went from 50% to 60%, which was below expectations.

After adjusting various hyperparameters, very similar results were seen so a different form of vectorizing data was used. Since the data is in the form of images from the cifar10 dataset, it was converted to float type and divided by 255.0 so that the values will all be in the range of 0 and 1. This form of vectorization resulted in a better model, as will be explained in the results and discussion. It is essential to keep in mind that this form of vectorization resulted in an input shape with 3 dimensions (32, 32, 3) where the image is 32x32 and the 3 stands for the RGB value. This resulted in another error when building the model layers, as shown below.

```
ValueError: Shapes (None, 3) and (None, 32, 32, 3) are incompatible
```

Figure 4 - Showing error when trying to output a shape different from the input

To fix this problem, the input layer was converted into one dimension so that it could be fed into the dense layer. To do this, the Flatten() method was used to convert the 3d vector image into a 1d because dense layers required data to be fed in 1d format.

4. Results and Discussion - Cifar10 Dataset

When building a model to train and test 3 subsets of the cifar10 dataset, multiple layers, units per layer, activation, and other hyperparameters were used. The relu activation was used for the hidden layers and the output layer used a softmax activation for this type of classification. Since we used a softmax activation and are working with binary feature vectors, the appropriate loss function used was categorical_crossentropy. Since we are measuring the training and validation accuracy, the accuracy metric was used. The initial model was trained for 20 epochs with a batch size of 512 using 3 hidden layers. The results are shown below.

```
94/94 [==============================] - 0s 1ms/step - loss: 0.7440 - accuracy: 0.6850
[0.7439998388290405, 0.6850000023841858]
```

Figure 5 - Showing evaluation of initial model

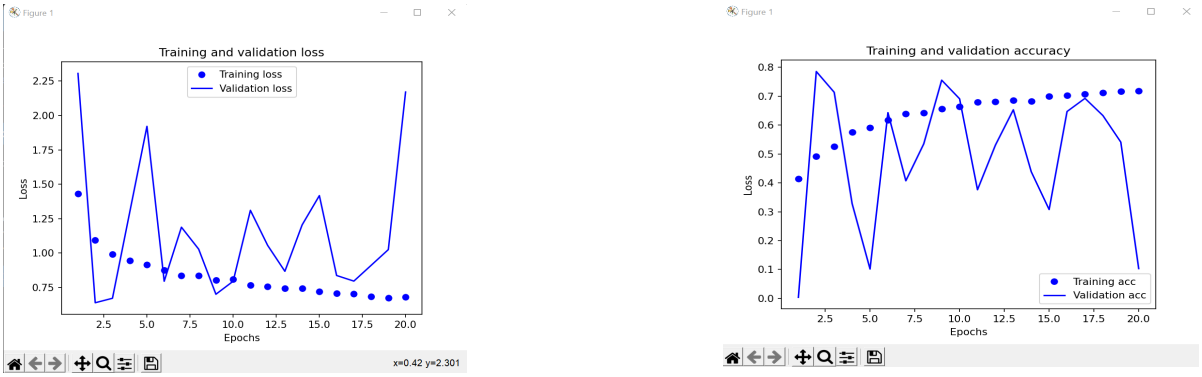


Figure 6 - Showing Training and validation loss and accuracy graphs of the initial model

As shown in Figure 6 above, the accuracy of the initial model was 68%, with a test loss of 0.74. As we can see from both graphs above, the training data generally performs well. As the model is being trained, the training loss decreases while the training accuracy increases. However, with the validation loss and accuracy, there are a lot of fluctuations and this seems to be because the model is performing well on training data but is not generalizing too well on testing data. Multiple changes were made to the number of hidden layers, size per layer, the ratio of validation data to training data, epochs, and batch sizes, but in all cases, the validation kept fluctuating. In the end, I was able to adjust the hyperparameters and make a final model that performed better than the initial model in the training data without much focus on the validation.

It may be possible to solve the problem of the validation data but that will require the use of an FCNN and using convolutional layers instead of dense layers, which is beyond the scope of this assignment. The results of the final model can be seen below.

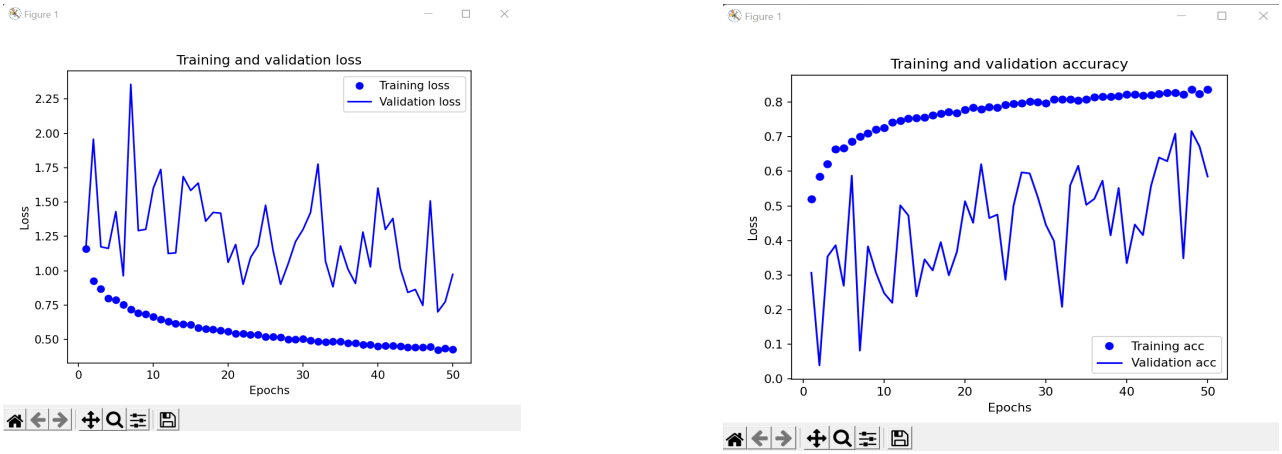


Figure 7 - Showing Training and Validation loss and accuracy of new model

```
94/94 [==============================] - 0s 2ms/step - loss: 0.5902 - accuracy: 0.7653
[0.5901588797569275, 0.765333354473114]
```

Figure 8 - Showing evaluated loss and accuracy of new model

Various hyperparameters were adjusted and optimized to get better accuracy and lower loss. The number of validation data was increased from 1,000 to 3,000, the number of epochs changed from 20 to 50, and batch size changed from 512 to 256. through experimentation, it was found that using 2 hidden dense layers instead of 3 resulted in higher accuracy and lower loss. The results of the final model show to have an increase in accuracy and a decrease in loss compared to the initial model. The accuracy increased by 9%, and the loss was reduced by 20%.

5. **References**

Chollet, F. (2021). *Deep learning with python*. Manning.

1. **Problem Statement - Spam Email Dataset**

   Load the spam email dataset from online repository, prepare and normalize the data, build a model, and plot results. Tune hyperparameters, retrain the model, and report results.

2. **Proposed Solution - Spam Email Dataset**

   A model will be built using dense layers with sigmoid activation and binary_crossentropy loss for this problem. Hyperparameters will be adjusted to increase performance.

3. **Implementation Details - Spam Email Dataset**

When working with the spam email dataset from the UCI Repository, a few errors were encountered when both preparing the data and building the model. When preparing the training data, the data was not normalized properly, which resulted in the training data containing values within an extensive range. This resulted in the model still working but the training accuracy and loss fluctuated a lot as shown in the figure below.

Figure 9 - Showing training/validation loss and accuracy graphs

To fix this problem, the method normalize() was used from the preprocessing class in sklearn library. This was able to normalize the training data and resulted in much smoother training loss and accuracy results which will be discussed later on.

4. **Results and Discussion - Spam Email Dataset**

When building a model to train and test the spam email dataset, multiple layers, units per layer, activation, and other hyperparameters were used. The relu activation was used for the hidden layers and the output layer used a sigmoid activation since this is a binary classification problem. The appropriate loss function to use with this activation and type of data is binary_crossentropy. Since we are measuring the training and validation accuracy, the accuracy metric was used. The initial model was trained for 50 epochs with a batch size of 512 using 2 hidden layers. The first layer had 32 units and the 2nd hidden layer had 16 units. The results are shown below.

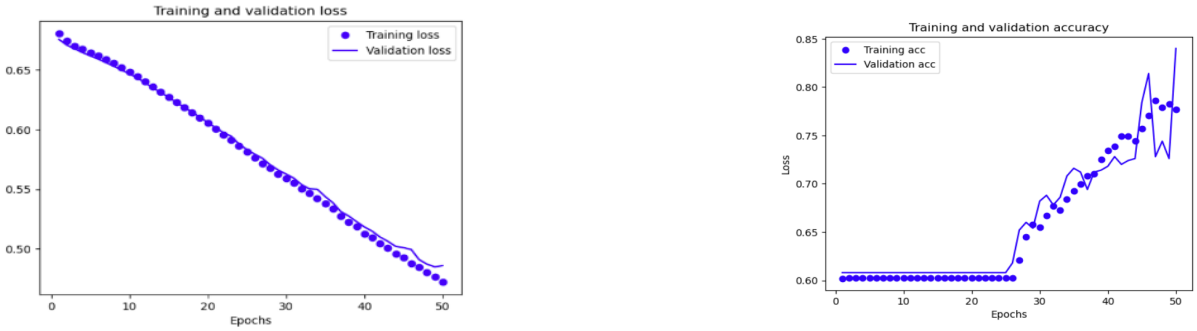


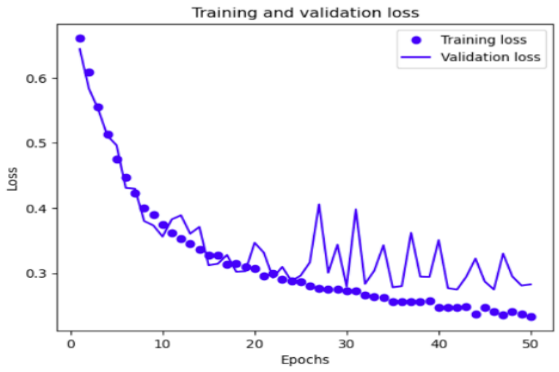
Figure 10 - Showing results of how well the initial model performed

```
44/44 [==============================] - 0s 3ms/step - loss: 0.4800 - acc
[0.48003676533699036, 0.8385227918624878]
```

Figure 11 - Showing results of how well the initial model performed

As shown in the results above, the initial mode generally performed well when evaluated. However, it can be seen that the training accuracy starts to increase very late, after about 26 epochs. Through experimentation, it was found that the number of units per layer played a role in this problem. In the final model, the number of units in the first layer was increased from 32 to 150 and the number of units in the 2nd hidden layer was increased from 16 to 100.

The same problem occurred with this dataset, whereby the validation accuracy fluctuated a lot. The final model will aim to solve some overfitting, some fluctuations within the training accuracy, and the training accuracy pause within the first 26 epochs. The results are shown:




```
44/44 [==============================] - 0s 2ms/step - loss: 0.2727 - accuracy: 0.9073
[0.2726922035217285, 0.9073135256767273]
```

Figure 12 - Showing results of how well the final model performed

As shown in the final model results above, although not a significant increase in performance when compared to the initial model, it is essential to note that now the training loss reduces at a much steadier rate and the training accuracy also increases at a steadier rate. Additionally, the training accuracy now starts to increase after 1 epoch compared to 26 epochs. All this was done by increasing the number of units per layer, decreasing the batch size from 512 to 100, and decreasing the validation data from appx 30% to 15% of training data.

1. **Problem Statement - Crime Dataset**
   Load the crime dataset from repository, prepare and normalize the data, build a model with k-fold cross validation. Tune hyperparameters, retrain the model, and report results.
2. **Proposed Solution - Crime Dataset**
   For this regression problem, k-fold cross validation, mse loss and mae metric will be used to build and evaluate the model. Hyperparameters will be adjusted to boost performance.
3. **Implementation Details - Crime Dataset**

When working with the crime dataset from the UCI repository, a few errors were encountered, primarily when preparing the data. Firstly, the first 5 columns of the dataset

consisted of attributes that were irrelevant to the model. For example, there were attributes such as community location, average income, etc. These attributes were irrelevant for the neural network since all we want is the actual crime data. To eliminate the first few columns, the iloc() and drop() functions were used to drop the first 5 columns which made it possible to continue.

The second problem that occurred was missing values. It was observed that some datasets had missing values with '?' as placeholders. This resulted in errors when trying to fit the model as it could not convert the '?' string to float. To fix this problem, the replace() function was used to replace all '?' with 0 values. This made it possible to use the training data to fit the model.

4. Results and Discussion - Crime Dataset

When building a model for the crime dataset, since this is a regression model, it was necessary to note that the k-fold cross-validation would have to be implemented, and specific loss and metrics would have to be used. Firstly, it was mentioned on the UCI website that the data has already been normalized between the range of 0 and 1, so that step was skipped.

The initial model consisted of a total of 3 layers where the first 2 layers used a relu activation with 64 units and the output layer did not have an activation. The output layer is typically a linear layer when working with regression problems. If activations are used in the output layer, then it will limit the range of values from which the model can learn, but in this case, we want the model to be able to learn freely without a constraint. Additionally, since this is a regression problem, the mse loss function and mae metric were used when compiling the model. K-fold cross validation using 4 folds was also used in the initial model with 100 epochs and a batch size of 1. The results of this can be seen below.

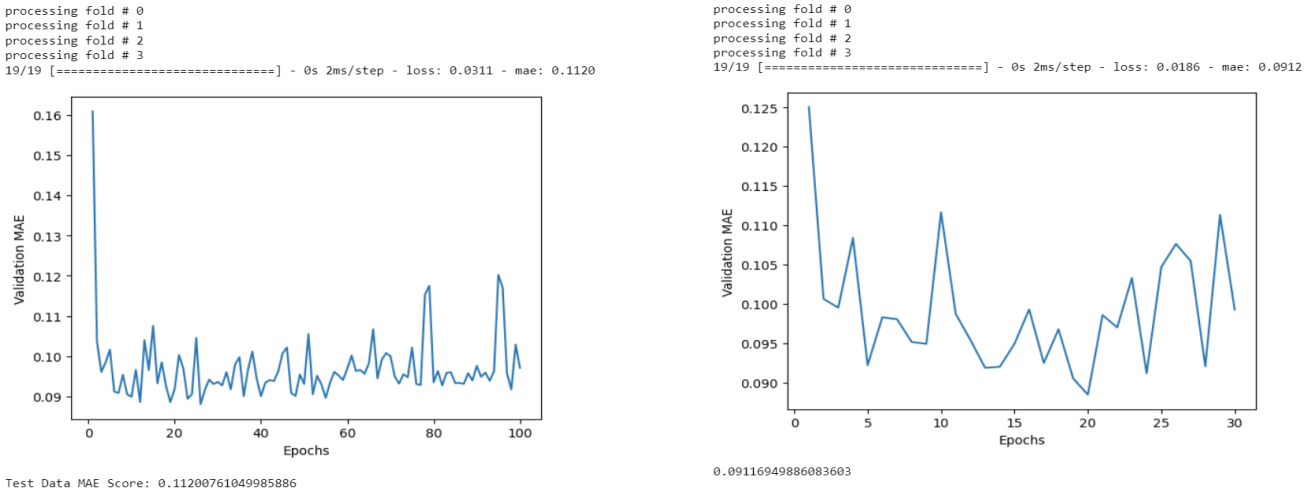


Figure 13 - Showing Validation MAE plotted against epochs to evaluate the performance of the initial model on the left and final model on the right

As we can see from the results of the initial model on the left, the validation MAE stops improving a lot after around 25 epochs. When evaluated on test data, the MAE was 0.1120. To improve these results, the number of units per layer on the 2nd layer was reduced from 64 to 32 and another dense hidden layer was created with 16 units, which helped in increasing the

performance of the model. Additionally, the number of epochs was reduced from 100 to 30 and the batch size was increased from 1 to 10. The results of the final model can be seen above on the right. It can be observed that the mae has dropped a bit when compared to the initial model. By reducing the number of epochs, increasing the batch size, and adding another hidden dense layer, better results were obtained.