CS577: Assignment 1

Jane Downer
Department of Computer Science
Illinois Institute of Technology

September 20, 2022

Abstract
Building basic neural networks using the Keras framework in Python.

Task 1

1 Problem Statement

- Train classifiers on datasets Tensorflow's Playground environment.

2 Proposed Solution

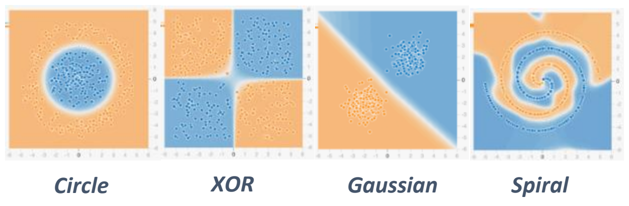
- Use trial and error to find configurations that accurately classified examples in each of the four datasets.
- Hypothesis – data following more complicated pattern requires more complicated network.

3 Implementation Details

- Hyperparameter values and other settings:

Dataset	Learning Rate	Activation	Regularization & Rate	Epochs	Features	Layers	Nodes per layer	Train/test ratio	Noise	Batch Size	Train Loss	Test Loss
Circle	0.1	ReLU	n/a	49	X_1^2, X_2^2	1	1st: 2	80%	0	20	0.001	0.001
XOR	0.01	ReLU	n/a	69	X_1*X_2	1	1st: 2	80%	0	20	0.01	0.01
Gaussian	0.1	ReLU	n/a	43	X_1, X_2	1	1st: 2	80%	0	20	0.001	0.001
Spiral	0.03	Tanh	L2, 0.01	200	X_1, X_2, X_1^2, X_2^2, X_1*X_2, sin(X_1), sin(X_2)	2	1st: 7 2nd: 4	70%	0	20	0.026	0.040

4 Results and Discussion



- The first 3 datasets can be accurately classified with simple, one-layer models. The Spiral dataset is consistent with expectations that more complicated patterns will require more complicated models.

Task 2

1 Problem Statement

- In this assignment, I am using my introductory knowledge of the Keras framework to build a basic neural network in Python – the goal for this task is to classify images in the CIFAR10 dataset.

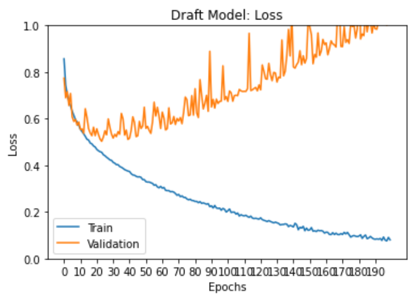
2 Proposed Solution

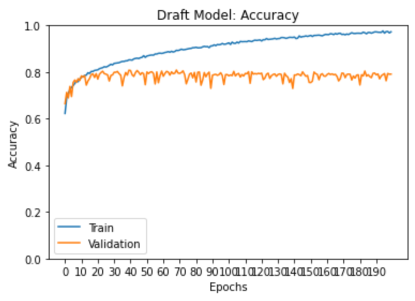
- Build a neural network suited for multi-class classification.
- Feed data forward through network, measure loss with categorical cross-entropy function, and use 'softmax' activation in final output layer to determine probability of observation belonging to each of 3 possible classes. Backpropagate and make updates to model.

3 Implementation Details

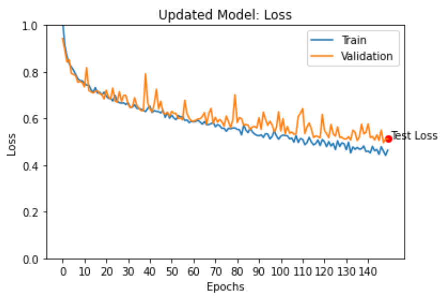
- Process:
    - Load CIFAR10 dataset.
    - Select subset of 3 classes.
    - Split data into training and validation subsets.
    - Reshape image data to vectors and perform categorical vectorization on the label data.
    - Design, build, and compile a neural network in Keras.
    - Plot training and validation accuracy and loss, and use this information to tune the hyperparameters and produce a final model.
- Initial model:
    - Use SGD optimizer and categorical cross-entropy loss function, which have been used in multi-class categorization problems in lecture.
    - Network architecture: 1 hidden layer with 32 nodes and 'relu' activation. Output layer uses 'softmax' activation. Set batch size to 16 and train for 200 epochs with learning rate 0.01.
    - Problems with model: overfitting, and validation learning plateaus.
- Updated model:
    - Goal: decrease overfitting while boosting learning capacity on unseen data.
    - Updates: Add a second layer hidden layer, with 256 nodes in both layers. Increase batch size to 512.
- Miscellaneous issues:
    - I originally wanted to randomize the subset of 3 classes. However, Keras' to_categorical function only considers category labels from range (0, num_classes). The solution was to settle for the subset with labels (0,1,2).
    - Training was slow until I realized I was not utilizing Colab's GPU.
    - Design challenges: boost performance on validation set while reducing overfitting.
- Instructions:
    - At the top of the .ipynb file, change 'ROOT_PATH' to the location where you want to save the final model to (or load the trained model from).
    - Program can be run all at once. Model will be saved at ROOT_PATH.
    - Use last line of code to load existing trained model from ROOT_PATH.

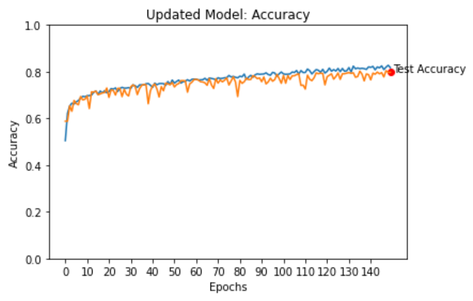
4 Results and Discussion





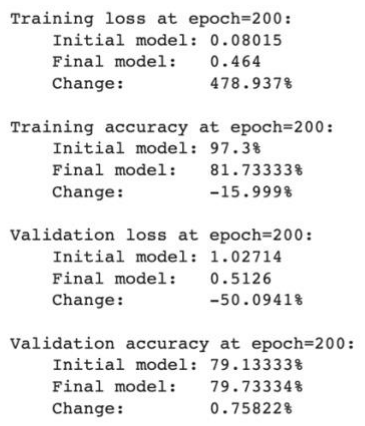
```
loss: 0.0801 - accuracy: 0.9730 - val_loss: 1.0271 - val_accuracy: 0.7913
```





```
loss: 0.4233 - accuracy: 0.8315 - val_loss: 0.4789 - val_accuracy: 0.8073
```

Training/Validation scores, before and after updates



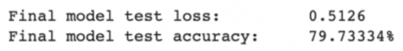
```
Training loss at epoch=200:
    Initial model: 0.08015
    Final model:   0.464
    Change:        478.937%

Training accuracy at epoch=200:
    Initial model: 97.3%
    Final model:   81.73333%
    Change:        -15.999%

Validation loss at epoch=200:
    Initial model: 1.02714
    Final model:   0.5126
    Change:        -50.0941%

Validation accuracy at epoch=200:
    Initial model: 79.13333%
    Final model:   79.73334%
    Change:        0.75822%
```

Test Performance



```
Final model test loss:          0.5126
Final model test accuracy:      79.73334%
```

- In the original model, performance on the validation set began to worsen after about 35 epochs, where training accuracy was about 85% and validation accuracy was about 78%.
- Design updates: (1) increased the batch size to reduce overfitting, (2) add layers and nodes to hopefully boost learning in the validation set.
- In the final model, after 200 epochs, performance had decreased significantly on the training set, and had increased slightly on the validation set. Most importantly, the gap between performance on training data and unseen data got much smaller, meaning that despite the decrease in performance on the training data, the updates allowed the model to perform better on unseen data.

<u>Task 3</u>

1 Problem Statement

- As in Task 2, this task involves using Keras to build a basic neural network. This task is a binary classification problem – classifying emails as ‘spam’ or ‘not spam’.

2 Proposed Solution

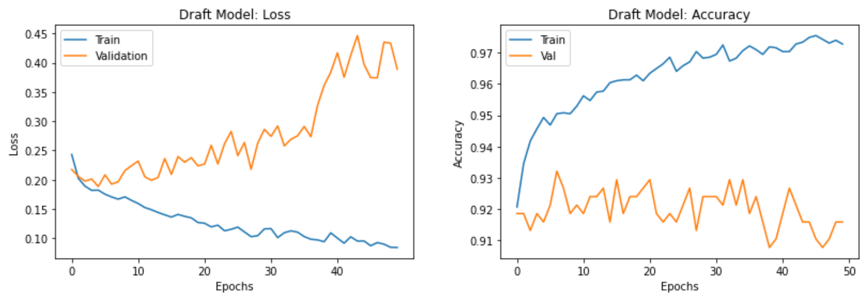
- Build a neural network suited for binary classification.
- Feed data forward through network, measure loss with binary cross-entropy function, and use ‘sigmoid’ activation in final output layer to determine probability of example belonging to positive class. Backpropagate and make updates to model.

3 Implementation Details

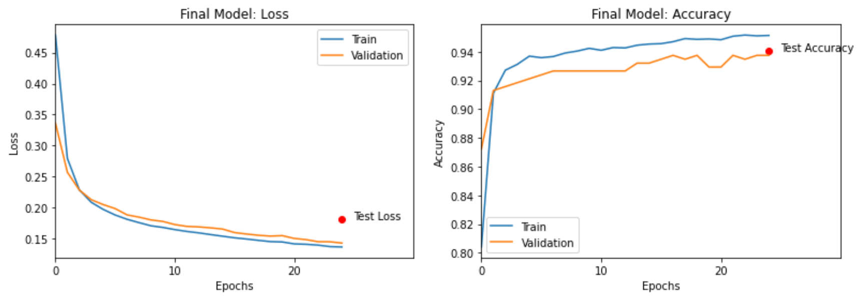
- Process:
  - Wrote a function to (1) load the data and (2) split the data into train and test sets.
  - Normalized data values – this helps keep the gradient values under control during gradient descent – prevents them from getting too small or too large.
  - Took some data from the training set to make the validation set.
  - Compiled and fit the model and plotted training and validation performance. Used this information to make updates to neural network design.
  - Repeated the above steps with an updated model.
- Initial model:
  - Use RMSprop optimizer and binary cross-entropy loss function, which have been used in binary classification problems in lecture.

    - Network architecture: 1 hidden layer with 16 nodes and 'relu' activation. Output layer uses 'sigmoid' activation. Set batch size to 8 and train for 50 epochs with learning rate 0.01.
    - Problems with model: high variance in performance on unseen data.
- Updated model:
    - Goal: improve generalization to unseen data by increasing batch size from 8 to 32. Increase number of nodes in hidden layers to 32 to ensure that learning capacity stays high.
- Instructions:
    - At the top of the .ipynb file, change 'ROOT_PATH' to the location where you want to save the final model to (or load the trained model from).
    - Program can be run all at once. Model will be saved at ROOT_PATH.
    - Use last line of code to load existing trained model from ROOT_PATH.

4 Results and Discussion



```
loss: 0.0843 - binary_accuracy: 0.9728 - val_loss: 0.3889 - val_binary_accuracy: 0.9158
```



```
loss: 0.1362 - binary_accuracy: 0.9514 - val_loss: 0.1424 - val_binary_accuracy: 0.9375
```

Training/Validation scores, before and after updates

```
Training loss at epoch=25:
    Initial model: 0.0843
    Final model:   0.13618
    Change:        61.539%

Training accuracy at epoch=25:
    Initial model: 97.28261%
    Final model:   95.13587%
    Change:        -2.207%

Validation loss at epoch=25:
    Initial model: 0.38894
    Final model:   0.14243
    Change:        -63.37994%

Validation accuracy at epoch=25:
    Initial model: 91.57609%
    Final model:   93.75%
    Change:        2.37389%
```

Test Performance

```
Final model test loss:        0.18138
Final model test accuracy:    94.02823%
```

- In the original model, performance on the validation set immediately began to worsen and was somewhat erratic.
- Design updates: increased the batch size to stabilize validation performance. Increased the number of nodes in the hidden layer to make sure learning levels stayed high after making this change.
- In the final model, after 25 epochs, training accuracy had performance had slightly decreased from performance with the initial model, but validation loss had improved significantly. Moreover, validation accuracy had increased by 2.37 %, suggesting that the model was better able to generalize to unseen data – this is further supported by the performance on the test set, where accuracy was about 94.02%.

<u>Task 4</u>

1 Problem Statement

- This task involves the creation of a basic neural network for a regression task – predicting crime rates in individual communities given a number of given predictors.

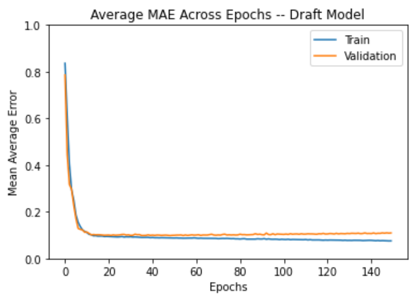
2 Proposed Solution

- Build a neural network suited for regression.
- Iteratively feed data forward through network, measure loss with mean squared error, backpropagate, and update the model. Network outputs a single value per observation – this is the predicted crime rate given the input.

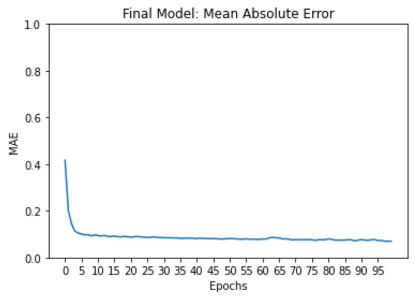
3 Implementation Details

- Process:
    - Wrote a function to load crime data, prepare data for training, and split into train/test sets. Data included several columns of information that weren't relevant to training, so I removed those. There was also a significant amount of missing data (replaced by '?' entries) – I handled this problem by replacing any '?' entries with 0. Additionally, a few numeric entries were entered as strings, so these needed to be converted to floats.
    - Normalized data as in Task 3 to reduce likelihood of vanishing or exploding gradients.
    - Built model, compile, and fit model.
- Initial model:
    - This is a regression task, so the loss function was set to mean squared error. The evaluation metric is mean absolute error, which allows interpretation of results in recognizable units.
    - Optimizer selected was Adam, which is common in Keras regression tasks[1].
    - Network architecture: designed to prevent overfitting. Includes hidden layers, both with 16 nodes (smaller to prevent overfitting), and 'relu' activation. Larger batch size (256) to improve model's ability to generalize to unseen data. Output layer produces single value, and doesn't use an activation function. Train for 150 epochs with learning rate 0.01.
    - No apparent problems with model, beyond training for too many epochs.
- Updated model:
    - Reduce training epochs from 150 to 100.
    - Difference between performance from initial model and updated model reveals dependence on dataset selection. See 'Results and Discussion'.
- Instructions:
    - At the top of the .ipynb file, change 'ROOT_PATH' to the location where you want to save the final model to (or load the trained model from).
    - Program can be run all at once. Model will be saved at ROOT_PATH.
    - Use last line of code to load existing trained model from ROOT_PATH.

4 Results and Discussion



```
loss: 0.0131 - mae: 0.0813 - val_loss: 0.0193 - val_mae: 0.0957
```



```
loss: 0.0091 - mae: 0.0695
```

MAE at epoch=100, before and after updates

```
Draft model MAE, averaged across k training subsets, epoch=100:   8.22615%
New model final MAE, training set, epoch=100:                     6.95336%

Draft model MAE, averaged across k validation subsets, epoch=100: 10.38639%
New model final MAE, test set, epoch=100:                         10.43174%
```

- In the original model, average performance across k training and validation looked acceptable. This suggested that no changes were needed, apart from stopping training at an earlier epoch, since performance eventually stopped making noticeable improvements. Design updates: decreased training from 150 epochs to 100 epochs.
- In the final model, after 100 epochs, the validation MAE from the initial model is very similar to the testing MAE from the final model, which is expected, since the only thing that changed was the epoch at which training stopped. Training MAE in the final model was somewhat lower than it had been after 100 epochs in the initial model. This is perhaps due to the fact that the initial model averaged the effects of k=5 different folds, whereas the training MAE from the final model is based off only one train/test partition.

5 References

[1] https://www.tensorflow.org/tutorials/keras/regression