**Approach**

**List of hyperparameters that will be tuned:**

1. LSTM / GRU / Simple RNN

2. No. of layers in RNN layer & No. of neurons in RNN layer

3. No. of dense layers & No. of neurons in dense layers

4. Optimizers

5. Learning Rate

6. Activation function

7. Dropout

8. Regularizers

9. Batch Normalization

**Baseline Model**

**Text

Description automatically generated**

Figure 1 - Baseline model

This is the baselines model that I will test between all three RNN (recurrent neural network) layer variants to see which is best. Callback is used and it is monitoring the metric “val\_loss” with a patience of 5. Meaning once “val\_loss” increases for 5 epochs in a row the model will stop training. This is a good way of detecting overfitting which can help safe time since it will stop your models training prematurely. Sparse categorical cross entropy is used as the loss function since our outputs are integers and are not one-hot encoded, making it the most appropriate loss function.

**LSTM / GRU / Simple RNN**

Although LSTM and GRU have long term memory unlike RNN and are generally considered to be better. The reviews tend to be short, here 7.79 word per review is the average review word count. Hence, Simple RNN might outshine the other 2 as it is faster, and its lack of long-term memory might not be an issue.

The key difference between GRU and LSTM is that GRU only has 2 gates (reset and update) while LSTM has 3 (input, output, and forget). Meaning that it is less complex and hence uses less memory and runs faster. However, LSTMs work better with datasets that uses long sentences.

Text

Description automatically generated

Figure 2 - Code to find average number of words per review

**Baseline model – GRU Chart, scatter chart

Description automatically generatedChart, line chart, scatter chart

Description automatically generated**

Model overfits ~ epoch 2 Validation Accuracy ~ 37%

**Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generatedBaseline model – LSTM**

Model overfits ~ epoch 1 Validation Accuracy ~ 35%

**Baseline model – Simple RNN**

**Chart, scatter chart

Description automatically generated**

**Chart, scatter chart

Description automatically generated**

Model overfits ~ epoch 1 Validation Accuracy ~ 34%

The model performed best when using GRU layers with a validation accuracy of 37%, thus, we will be using GRU layers for the future models.

**Summary**

****

**No. of GRU layers & neurons in each layer**

For this parameter, we will first tune the number of GRU layers before tuning the number of neurons in each layer.

The more GRU layers / neurons, the higher the model’s complexity allowing it to understand complex relationships better. Too high a complexity however results in overfitting causing the model to do poorly on unseen data. Too low a complexity and the model will not be able to generalize well on the training data.

**GRU (+1 GRU layer)**

**Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated**

Model overfits ~ epoch 2 Validation Accuracy ~ 36%

Here with an increase in the number of layers, the validation accuracy slightly worsens. As such we will now decrease the number of GRU layers.

**GRU (-1 GRU layer)**

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

Model overfits ~ epoch 1 Validation Accuracy ~ 36%

From the graphs the validation accuracy also dropped slightly when using one less GRU layer. This means that the original number of GRU layers used (3) is optimal and will continue to be used.

**GRU (32 Neurons) - Selected**

**Chart, scatter chart

Description automatically generated**

Chart, scatter chart

Description automatically generated

Model overfits ~ epoch 1 Validation Accuracy ~ 37%

**GRU (128 Neurons)**

**Chart, scatter chart

Description automatically generated**

**Chart, line chart, scatter chart

Description automatically generated**

Model overfits ~ epoch 1 Validation Accuracy ~ 36%

The model with 128 neurons did slightly worse than the current model and is too complex. While the model with 32 neurons did similar to the current model, for this reason I will be using it as a simpler model with the same accuracy is preferred.

**Summary**

A picture containing background pattern

Description automatically generated

**No. of DENSE layers & neurons in each layer**

**Dense (+1 Dense)**

**Chart, line chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated**

Model overfits ~ epoch 1 Validation Accuracy ~ 36%

**Dense (-1 Dense)**

**Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated**

Model overfits ~ epoch 1 Validation Accuracy ~ 36%

Both models performed slightly worse with validation accuracies of 36%. Hence, the original number of dense layers will be used (2).

**Dense (32 Neurons)**

Chart, scatter chart

Description automatically generated**Chart, scatter chart

Description automatically generated**

Model overfits ~ epoch 1 Validation Accuracy ~ 36%

**Dense (128 Neurons)**

**Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated**

Model overfits ~ epoch 2 Validation Accuracy ~ 37%

Likewise, there is no substantial improvements by decreasing or increasing the number of neurons in the dense layers. Thus, I will continue to use 64 neurons for the dense layers.

**Optimizers**

The optimizer used is Adam (Adaptive Moment Estimation) which converges very quickly. Adam combines the best properties of AdaGrad and RMSProp allowing it to handle sparse gradients on noisy problems. As Adam is an improvement on RMSprop, we will not be trying RMSprop but rather a gradient-based optimizer (SGD). There have been several studies that indicated that SGD could generalize better than Adam.

**SGD 0.1**

**A picture containing chart

Description automatically generated**

**Chart

Description automatically generated with medium confidence**

Model overfits ~ epoch 25 Validation Accuracy ~ 35%

**SGD 0.07**

**Chart

Description automatically generatedChart

Description automatically generated with medium confidence**

Model overfits ~ epoch 40 Validation Accuracy ~ 35%

Using SGD as the optimizer, the convergence of the model is much longer, and training takes longer. The accuracies hit was slightly below or on par with Adam’s accuracy. As such, there is no benefit to using the SGD optimizer. I also increased the patience for the early callbacks to ensure that the model was really overfitting since there was more noise during the training.

**Learning Rates**

Learning rate is the value that determines how fast the model will “learn” and it is important to find a learning rate that balances between speed of convergence and quality of convergence. As smaller learning rates allow for more optimal learning but can take significantly longer to train. We will be tuning the learning rate for the Adam optimizer which for our initial models used a learning rate of 0.001.

**Adam 0.003**

Chart, line chart

Description automatically generatedChart, scatter chart

Description automatically generated

Model overfits ~ epoch 1 Validation Accuracy ~ 37%

**Adam 0.0008 (Selected)**

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

Model overfits ~ epoch 2 Validation Accuracy ~ 37%

The models with the tuned learning rates all hit the same validation accuracy of 37%. However, for the learning rate of 0.0008, the model overfits later at epoch 2 instead of epoch 1.

**Summary**

**Table

Description automatically generated**

**Weight Initialization**

Weight initializers aid in preventing exploding and vanishing gradient problems during model training. Below are the weight initializers that GRU layers uses as well as their activation functions.

****



The Glorot weight initializer are good for and have become the standard for non-linear activation functions such as Sigmoid, Softmax and Tanh but does not work as well on ReLU. For ReLU, the He weight initialization was specifically developed for layers that use the ReLU activation. As such, we will be trying out Glorot initializers all layers which do not use ReLU as an activation function.

In the interest of time, “glorot\_uniform” will be used for the Glorot initializers and “he\_uniform” will be used for the layers with ReLU activation.

Text

Description automatically generated

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

Model overfits ~ epoch 1 Validation Accuracy ~ 36%

Text

Description automatically generated

Removed “recurrent\_initializer=glorot\_init” from the GRU layers such that they revert back to their default initialization of “orthogonal”. Possibility that the change from orthogonal to Glorot uniform might be detrimental to the model.

Chart, scatter chart

Description automatically generatedChart, line chart, scatter chart

Description automatically generated

The model with the initialized weights, had a slight decrease in validation accuracy (36%) as a whole. There were no improvements in validation accuracy and thus, the default weight initializers will be used.

**Activation Function**

The hyperbolic tangent (tanh) activation is a popular activation function used especially in RNNs. It is more popular then ReLU as it is said that RNNs often times amplifies the dying ReLU problem which is where a neuron stops learning and produces the same output.

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated**Tanh Activation**

Model overfits ~ epoch 1 Validation Accuracy ~ 36%

Seeing that the validation accuracy of tanh is not as good as ReLU I will continue using ReLU as the activation function.

**Dropout**

Dropout layers are used to combat overfitting by shutting off a percentage of neurons that will be used for training during a specific epoch in a random manner. By using dropouts, we aim to delay overfitting and through the delay see an increase in the validation accuracy.

Recurrent dropout is used for GRU layers and dropout is used for dense layers. In the interest of time, both will be trained simultaneously with the same dropout rate.

**Dropout 0.2**

Chart, line chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Model overfits ~ epoch 2 Validation Accuracy ~ 35%

**Dropout 0.5**

**Chart, scatter chart

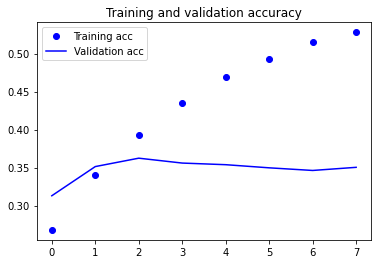
Description automatically generatedChart, scatter chart

Description automatically generated**

Model overfits ~ epoch 3 Validation Accuracy ~ 36%

**Dropout 0.7**

**Chart, line chart, scatter chart

Description automatically generated**

Model overfits ~ epoch 3 Validation Accuracy ~ 36%

At dropout rates 0.5 and 0.7, the model overfitted slightly later at around epoch 3. However, all the validation accuracies reached were slightly lower than the current model (no dropout).

**Regularizers**

Regularization is another overfitting technique which penalizes the weights in the layer with a penalty. There are two types of regularizers, L1 and L2. L1 penalizes the sum of absolute values of the weight while L2 penalizes the sum of squares of the weights. This difference causes neurons with an L1 to shrink to zero while L2 neurons tend to shrink evenly. As such, L1 is better suited for feature selection and L2 is better suited when features are codependent. As the features in RNN are codependent we will only be testing with the L2 regularizer.

**L2 = 0.01**

A picture containing line chart

Description automatically generated**Chart, line chart, scatter chart

Description automatically generated**

Model overfits ~ epoch 4 Validation Accuracy ~ 32%

**L2 = 0.001**

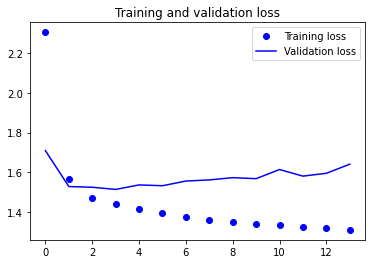
Chart, line chart, scatter chart

Description automatically generated**Chart, scatter chart

Description automatically generated**

Model overfits ~ epoch 2 Validation Accuracy ~ 33%

**L2 = 0.006**

**Chart, scatter chart

Description automatically generated**

Model overfits ~ epoch 4 Validation Accuracy ~ 31%

When the model uses regularizers, the validation accuracy significantly drops to approximately 31%, furthermore the model converges slower. The loss also increases much more slowly and does not spike as much when overfitting.

**Weight Normalization**

Weight normalization is a reparameterization of the weight vectors in a neural network and by doing so it speed up convergence of the model. However, it does not introduce any dependencies between the neurons allowing it to be effectively applied to recurrent models unlike batch normalization

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

Model overfits ~ epoch 1 Validation Accuracy ~ 36%

It will not be implemented as the validation accuracy of the model was lower than the current model’s validation accuracy of ~37%.

**Summary**

Table

Description automatically generated with medium confidence

**Conclusion**

Overall, the model’s validation accuracy improved by little to none after all the tuning and training of hyperparameters. The best model hit validation accuracy of 37% as seen below.

Possible reasons for this are:

1. Poor quality of data
2. Too little data
3. Insufficient hyperparameter tuning

**Final Model**

Chart, scatter chart

Description automatically generated