

CS 577 - F22 - Assignment 3

Report

Due date: 10/25/2022

Anas Puthawala - A20416308
Professor Gady Agam

Classification | Analyzing Loss Functions

The model architecture that was primarily used for this task is seen in Figure 1. The data used for this task is the abalone dataset from UCI ML repository. The variable we are trying to predict is the Sex of the abalone species (three possibilities: M, F, or I where I is infant). The Sex column was categorically encoded using the following line: `tf.keras.utils.to_categorical(df.Sex)`. The data was split into training, validation and test set with a 70, 20, 10 split (respectively). The size of the training set was 2,819 samples, validation 940, and testing of 418.

Model: "sequential"		
Layer (type)	Output Shape	Param #
=====		
dense (Dense)	(None, 56)	504
dense_1 (Dense)	(None, 32)	1824
dense_2 (Dense)	(None, 16)	528
dense_3 (Dense)	(None, 3)	51
=====		
Total params: 2,907		
Trainable params: 2,907		
Non-trainable params: 0		

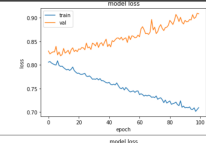
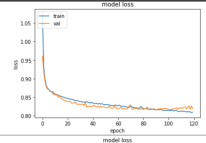
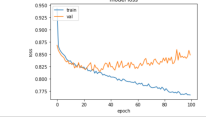
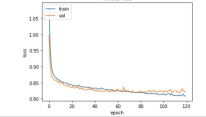
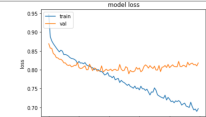
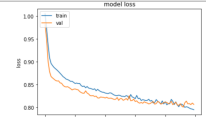
The following losses were analyzed:

1. Categorical Cross Entropy w/ Adam Optimizer
2. KL-Divergence w/ Adam Optimizer
3. Categorical Hinge Loss w/ Adam Optimizer

The experiment was carried out as such:

- > Train the default model (Fig. 1) on the loss with the default LR used, 100 epochs, and a batch size of 32. (Get the model to basically show signs of over-fitting from the loss curves)
- > Get a baseline model accuracy with no tuning by combining the training and validation set into one training set and evaluating on the test set.
- > Observe the loss curves and tune the model's hyper parameters as needed to prevent over-fitting (i.e. adjust LR, epochs, batch size)
- > Finally, once a desirable loss curve is seen, combine the training and validation data into one set for training, train the entire model for the tuned number of epochs and evaluate the data on the test set. (Note: I couldn't include both loss and accuracy curve on this report due to page limitations, but feel free to view them and the progression of my tuning in my submitted code)

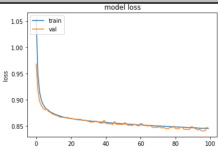
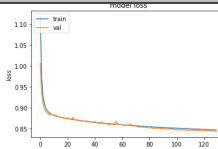
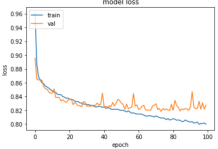
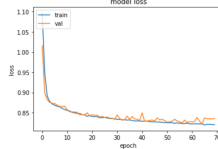
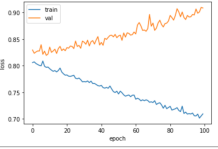
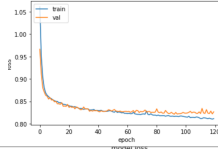
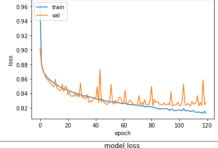
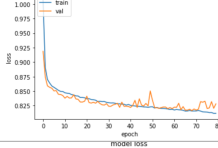
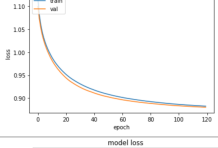
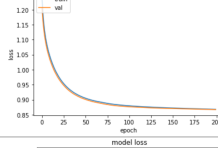
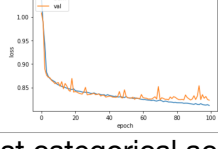
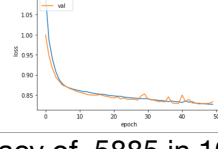
Fig 1. Model Architecture Primarily used

Loss	Loss Curve Before	Loss Curve After	Best Parameters	Starting & Ending accuracy
CCE			LR = .00080 epochs=120 batch_size=64	Starting categorical accuracy = .5455 Ending categorical accuracy = .5742
KL-Divergence			LR = .00080 epochs = 70 batch_size = 64	Start = .5287 End = .5670
Categorical Hinge			LR = 0.00075 epochs = 90 batch_size = 128	Start = .5431 End = .5526

We can see that CCE loss after tuning resulted in the best categorical accuracy of .5742, followed by KL-Divergence loss and then lastly categorical hinge loss.

Classification | Analyzing Optimizers + Regularization

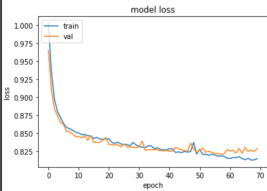
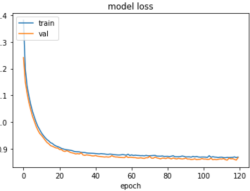
In this section we analyze some optimizers available in Keras along with some different regularization measures such as batch normalization, dropout, and weight-decay. The following optimizers I use are SGD, RMSprop, Adam, Adadelata, Adagrad, and Nadam. In all cases, the loss function will remain the same (CCE). I also try to keep the epochs and batch size the same and tune the parameters only if needed (i.e. significant overfitting etc) because tuning other parameters may impact the time it takes for each optimizer to converge, etc. The model used in this section is three dense layers, first layer has 30 units w/ Relu activation, second layer has 12 units with relu activation, and the third layer has 3 units with softmax activation.

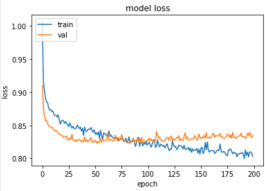
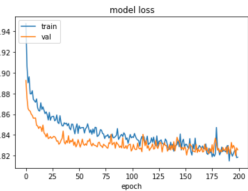
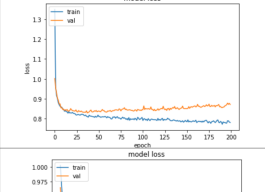
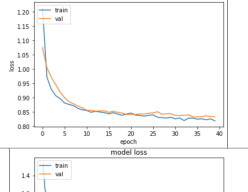
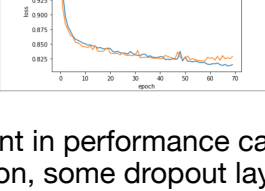
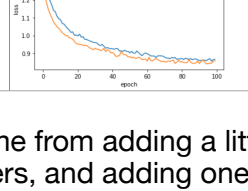
Optimizer	Loss Curve Before	Loss Curve After	Best Params + time elapsed for training	Best Accuracy (Categorical Accuracy)
SGD			LR = .009 epochs = 120 batch_size = 64 time = 29.01 sec.	0.5646
RMSprop			LR = .0009 epochs = 70 batch_size = 64 time = 10.68 sec.	0.5574
Adam			LR = .00080 epochs = 70 batch_size = 64 time = 10.64 sec.	0.5885
Adadelata			LR = 0.75 epochs = 45 batch_size = 32 time = 21.18 sec.	0.5766
Adagrad			LR = .0015 epochs = 35 batch_size = 64 time = 4.7 sec.	0.5622
Nadam			LR = .00085 epochs = 50 batch_size = 64 time = 9.53 sec.	0.5622

Adam achieved the best categorical accuracy of .5885 in 10.64 seconds followed by Adadelata which achieved an accuracy of .5766 in 21.18 seconds, SGD which achieved an accuracy of .5646 in 29.04 seconds, Adagrad and Nadam tied at an accuracy of .5622 but Adagrad was ~ 4.5 seconds faster than Nadam. And lastly we had RMSprop at .5574 accuracy an 10.68 seconds. It's important to note that SGD barely showed any signs of over-fitting in the start and required minimal tuning, Adagrad's loss curves are VERY smooth and converges very rapidly. I tried higher epochs (50, 100, 300 - 400) on Adagrad, but there was no significant improvement in the accuracy and some even showed a decrease in performance.

After we looked at evaluating the effect of different optimizers, we also looked at different regularization methods. Note that we initiated this testing by having a baseline with no regularization applied. We received an accuracy from the baseline of **.5646** and a loss of **.8220**. We used Adam optimizer (lr = 0.00080), with CCE loss and had to adjust the epochs & batch size to tune the parameters.

The following table summarizes the findings:

Regularization Method	Loss Curve Before	Loss After	Best Params	Best Accuracy & lowest loss
Kernel Regularization (L2)			LR = 0.00080 120 epochs, batch size of 64, and L2 reg. w/ a rate of .01 on the input layer and layer 2.	best acc: .5718 lowest loss: .8506

Regularization Method	Loss Curve Before	Loss After	Best Params	Best Accuracy & lowest loss
Dropout Layer			LR = 0.00080, 50 epochs, batch_size of 64, and added a single dropout layer w/ rate 0.25 (tuned) before the final layer	Best acc = .5670 lowest loss = .8151
Batch Normalization			same LR, 45 epochs, two batch norm layers.	Best acc = .5502 lowest loss = .8322
All of them combined			same LR, 120 epochs, same batch size.	Best acc = .5728 lowest loss = .8499

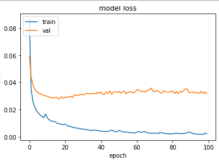
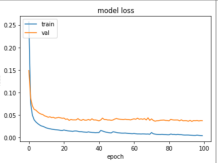
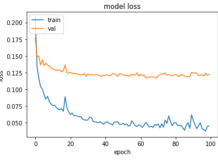
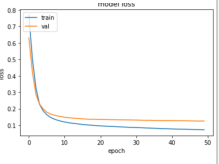
The best improvement in performance came from adding a little bit of kernel regularization, some dropout layers, and adding one batch normalization layer. It's no surprise that adding these little bits of regularization techniques serves to definitely improve the model's generalizability and in-turn result in higher performance. One thing to note is that kernel regularization with L2 at a rate of .01 was very close to resulting in the top accuracy (just .0010 off). It's not necessarily true that more is better, just going based off quality the L2 regularization could've sufficed and saved us time. The final model summary with all the tuning is shown in the figure to the right.

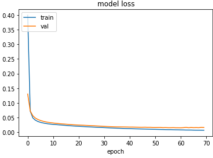
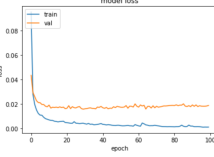
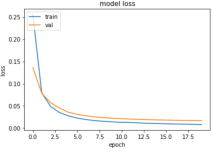
Dropout rate was .25 for the first layer and 0.15 for the second, and in the layer *dense_33* and *dense_35* there was L2 regularization added with a rate of 0.01.

Model: "sequential_11"		
Layer (type)	Output Shape	Param #
dense_33 (Dense)	(None, 40)	360
batch_normalization_8 (Batch Normalization)	(None, 40)	160
dense_34 (Dense)	(None, 15)	615
dropout_3 (Dropout)	(None, 15)	0
dense_35 (Dense)	(None, 10)	160
batch_normalization_9 (Batch Normalization)	(None, 10)	40
dropout_4 (Dropout)	(None, 10)	0
dense_36 (Dense)	(None, 3)	33
Total params: 1,368		
Trainable params: 1,268		
Non-trainable params: 100		

Regression | Analyzing Loss Functions

Briefly, the data used for the regression section is the communities and crime dataset, where the variable we're aiming to predict is 'ViolentCrimesPerPop'. When analyzing loss functions, the optimizer was held constant as 'Adam', the LR, batch size, and epochs were tuned to reduce / eliminate signs of over-fitting from the loss curves. The experiment was carried out in a similar manner to the previous section for Classification. The metric used to evaluate was also held constant at 'MAE'. The loss functions analyzed, along with the models loss curves and best parameters are displayed in the table:

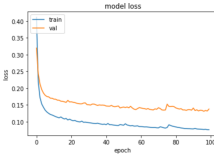
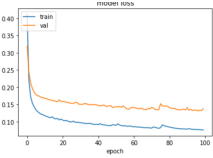
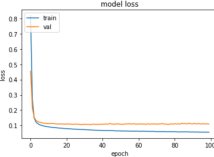
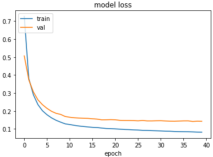
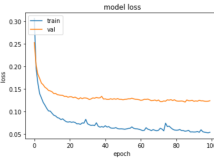
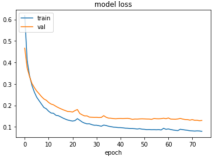
Loss	Loss Curve Before	Loss Curve After	Best params	Starting metrics	Ending metrics
MSE			Lr=.00085, 100 epochs, batch size of 64,	loss: 0.0351 - mae: 0.1290	loss: 0.0216 mae: 0.1035
MAE			Lr = 0.00045. 45 epochs, batch size of 128	loss: 0.0974 mae: 0.0974	loss: 0.1008 mae: 0.1008

Loss	Loss Curve Before	Loss Curve After	Best params	Starting metrics	Ending metrics
LogCosh			lr = 0.00075, 65 epochs, batch size of 32	loss: 0.0125 mae: 0.1093	loss: 0.0112 mae: 0.1072
Huber			lr = .00045, 35 epochs, batch size of 32	loss: 0.0145 - mae: 0.1168	loss: 0.0144 mae: 0.1136

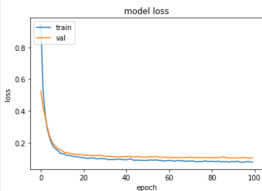
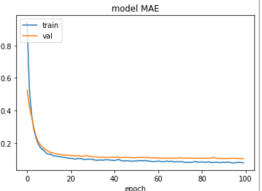
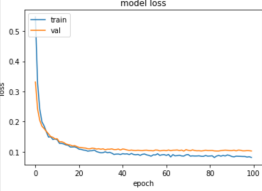
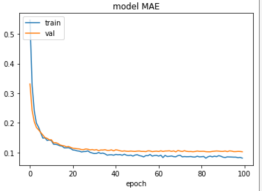
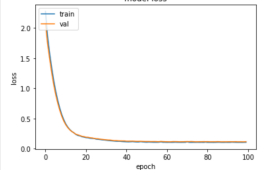
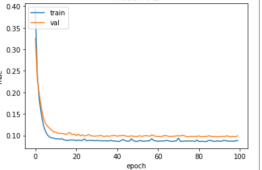
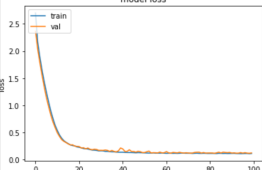
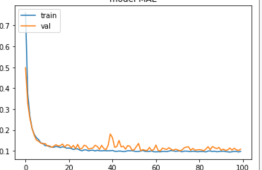
To briefly summarize, MAE had the best performance. The starting metrics of using MAE loss were better than some of the optimized losses. The possible reason behind this is that MAE is our metric to evaluate and the loss itself. Second best was MSE, then followed by LogCosh and then Huber loss. The loss curves look significantly better in huber loss and logcosh, however just because they look good surprisingly doesn't mean that the models performance when evaluating it on the test set will be.

Regression | Analyzing Optimizers + Regularization

We analyze a few optimizers (namely: SGD, RMSprop, and Adam) and regularization (such as weight decay, batch norm, and dropout). The metric used in all of this analysis will remain MAE, and the loss will also be held constant at MAE. The model we use has a dense layer with 12 units, connected to a dense layer with 5 units (both of which will have relu activation) and then finally connected to a dense layer with no activation (i.e. linear activation). The time will also be tracked for the model to converge based on each optimizer. The following table summarizes the results obtained from the optimizers:

Optimizer	Loss Curve Before	Loss Curve After	Best Params	Starting metrics	Ending metrics
SGD			LR = default, nesterov=True, epochs=100, batch_size=32	loss: 0.1036 mae: 0.1036, time: 11.51 sec	loss: 0.0997 mae: 0.0997 time: 12.14 s
RMSprop			LR = .0005 rho=0.8, centered=True, epochs = 40, bs = 64	loss: 0.1100 mae: 0.1100, time: 14.09 sec	loss: 0.1069 mae: 0.1069, time: 5.71 sec
Adam			LR = .0007 amsgrad=True, epochs=70, bs = 64	loss: 0.0992 mae: 0.0992, time: 20.84 sec	loss: 0.0984 mae: 0.0984, time: 5.51 sec

Briefly summarizing, Adam is the best optimizer, followed by SGD, then by RMSprop. RMSprop converges just as fast as Adam, but it's not as accurate. SGD takes a bit longer to converge but is still pretty accurate. The loss curve for RMSprop can be deceiving, as it looks like the optimal model, but that is not the case. Lastly, we evaluate different regularization methods and observe the improvement in the in the MAE of the model from the different loss curves. The optimizer used for evaluating regularization methods will remain as Adam, with MAE loss as the metric.

Regularization Method	Final Loss Curve	Final Accuracy Curve	Best Params	Best Metrics
Batch Norm			LR = .001 epochs = 100, batch size = 32, (see model: 'sequential_1')	loss: 0.0942 mae: 0.0942 time = 18.67 sec
Dropout			LR = .001 epochs=100, batch size = 32, dropout rate = .15 & .10 (see model: 'sequential_2')	loss: 0.0921 mae: 0.0921 time = 20.91 sec
Weight Decay			LR = 0.01, epochs=100, batch size = 32, see below for reg. rates*	loss: 0.1026 mae: 0.0886 time = 20.92 sec
All of the above combined			LR = 0.01, epochs = 100, batch size = 32, see below for details** (see model: 'sequential_94')	loss: 0.1068 mae: 0.0919 time = 16.54 sec

Combining a bit of all the regularization, along with the best loss function and optimizer (MAE, Adam) resulted in the best model MAE, but not necessarily the lowest loss. The following few figures show where exactly the batch normalization layers and dropout layers were added to attain the results above:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 12)	1524
batch_normalization (Batch Normalization)	(None, 12)	48
dense_1 (Dense)	(None, 5)	65
batch_normalization_1 (Batch Normalization)	(None, 5)	20
dense_2 (Dense)	(None, 1)	6
Total params: 1,663		
Trainable params: 1,629		
Non-trainable params: 34		

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 12)	1524
dropout (Dropout)	(None, 12)	0
dense_4 (Dense)	(None, 5)	65
dropout_1 (Dropout)	(None, 5)	0
dense_5 (Dense)	(None, 1)	6
Total params: 1,595		
Trainable params: 1,595		
Non-trainable params: 0		

Model: "sequential_94"

Layer (type)	Output Shape	Param #
dense_286 (Dense)	(None, 12)	1524
dropout_10 (Dropout)	(None, 12)	0
dense_287 (Dense)	(None, 5)	65
batch_normalization_4 (Batch Normalization)	(None, 5)	20
dense_288 (Dense)	(None, 1)	6
Total params: 1,615		
Trainable params: 1,605		
Non-trainable params: 10		

The first dropout layer in the dropout model ('sequential_2') had a dropout rate of 0.15 and the last one had a dropout rate of 0.10. This was tuned to find the optimal performance. Similarly, the dropout rate in 'sequential_94' was tuned manually to a rate of 0.15. Lastly, as for weight regularization, the model used was the initial model that we started with. L1 and L2 regularization both were applied to the kernel using kernel_regularizer in the input layer with a rate of 0.01 and 0.01 and a rate of 0.01 for L1 regularization only in the second dense layer (same kernel regularization). The final model with combined forms of regularization had kernel regularization applied at the same locations and at the same rates.

Final conclusions are that combining forms of regularization will often lead to better performance, and that it's not always the case that the prettiest loss curves may result in the best performing model.