NOTE:

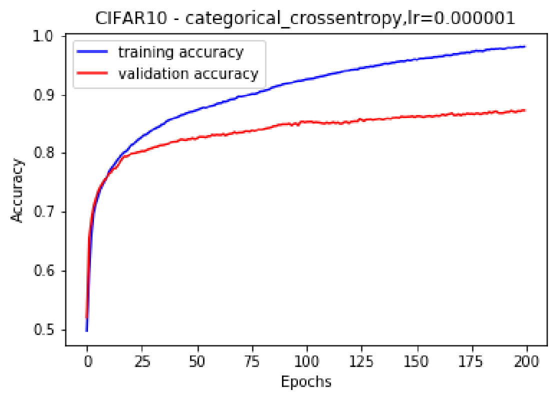
- For each evaluation, I fix all model hyperparameters (learning rate, number of layers, etc) and **only** vary the hyperparameter being evaluated
- For each model, 2 graphs will be displayed: the accuracy/mae vs epochs graph and the loss vs epochs graph stacked side by side

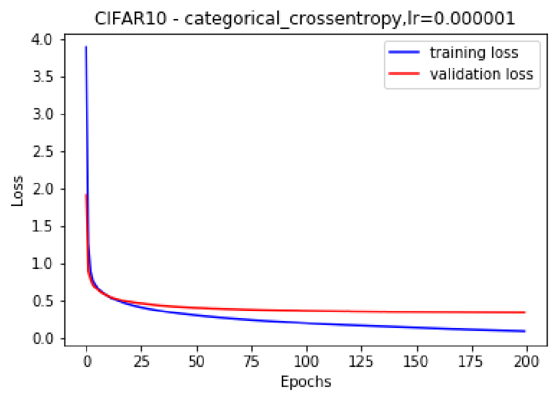
1. Classification:

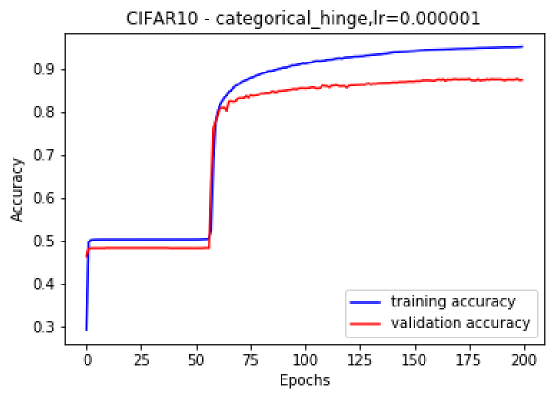
- Dataset used: CIFAR10 with 3 categories

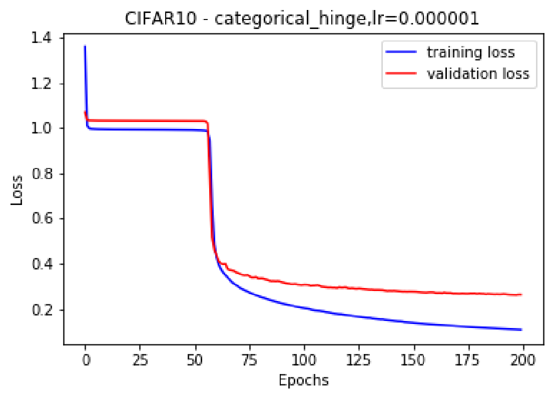
a. Evaluating losses:

- Since this is a multi-class classification task, I decided to use categorical crossentropy and hinge loss (categorical hinge as provided by keras) amongst other losses provided. Following are the results:







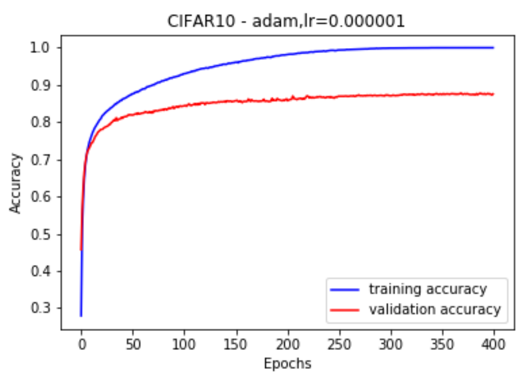


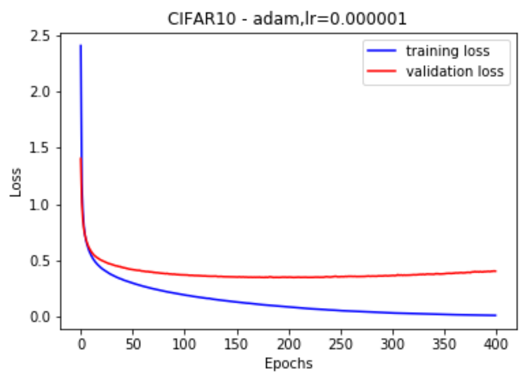
- As can be seen from these graphs, it seem like the model using hinge loss is learning slower than the model using categorical crossentropy, since the one using hinge loss only reached 80% accuracy and started to overfit after 75 epochs while the other reached 80% accuracy and overfitted at around the 25th epoch. This is because there is a period where the hinge model stopped improving entirely then suddenly improved quickly. This non-improving period also makes the hinge graphs look less smooth than the categorical crossentropy graph. Finally, it also seems like the hinge model overfitted "less" than the categorical crossentropy model.

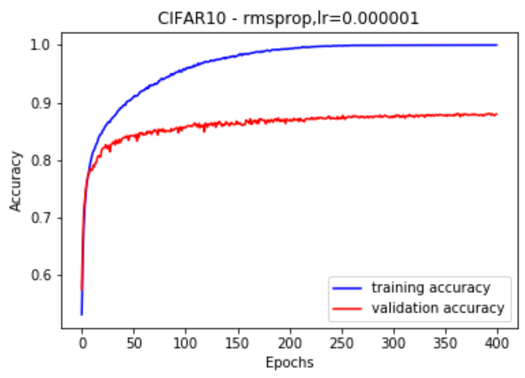
- I also used Kullback Leibler loss and as expected, the KL loss model performs similarly to the categorical crossentropy model.

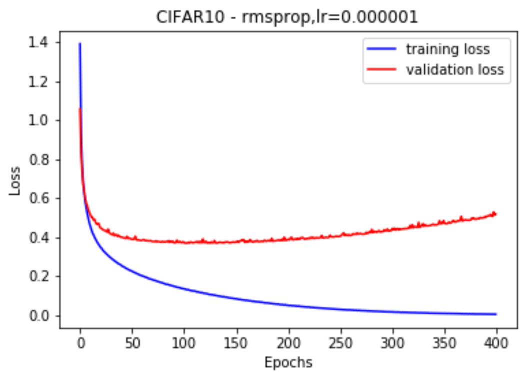
b. Evaluating optimizers:

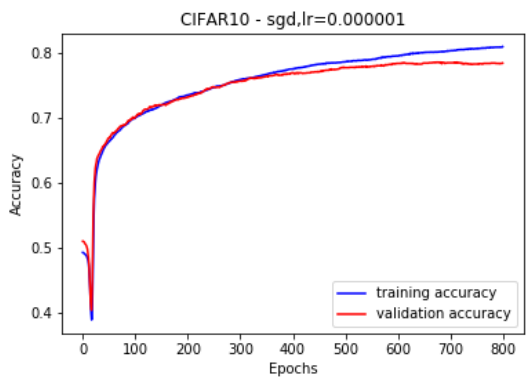
- The optimizers being evaluated are Adagrad, Adam, RMSprop, SGD, and SGD with nesterov momentum

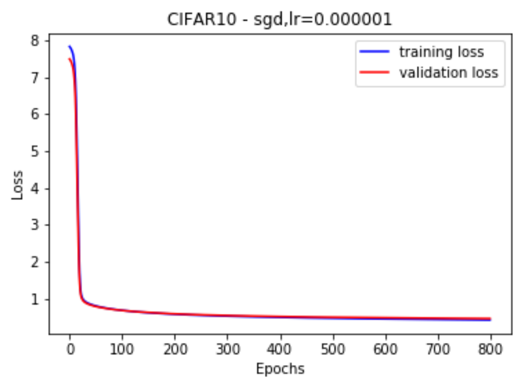


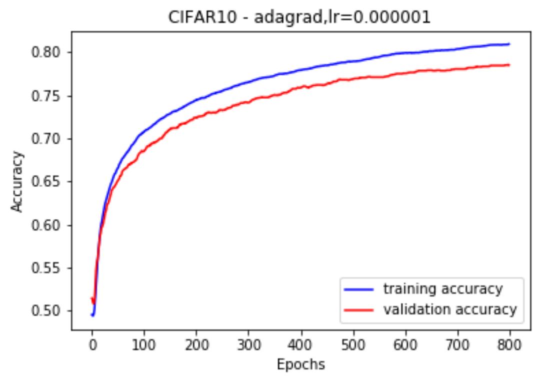


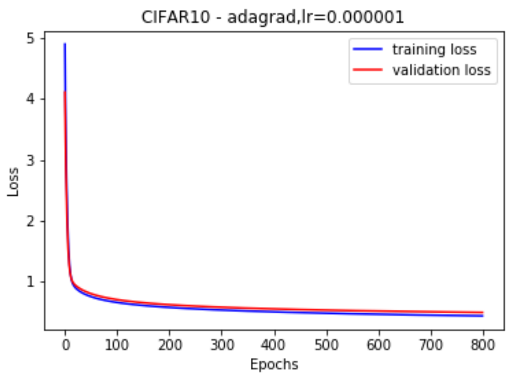










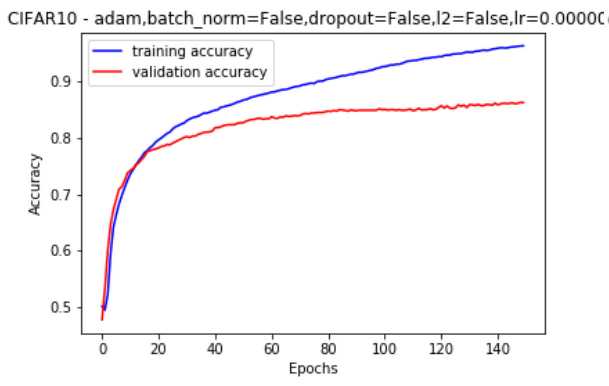


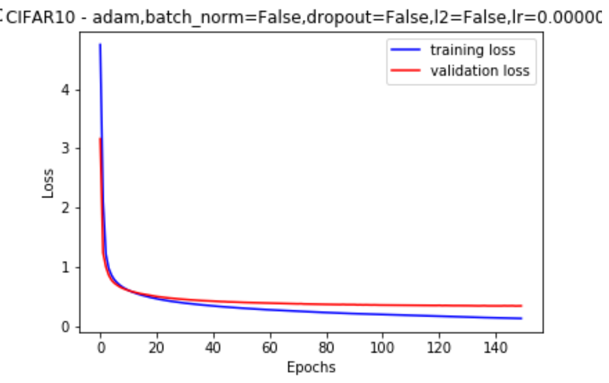
- RMSProp and Adam started overfitting very quickly around epoch 20. Adagrad started to overfit around epoch 65 but the overfitting effect is very minimal. I didn't include the graphs for SGD with nesterov momentum because they look basically the same as SGD graphs. That being said, SGD and SGD with nesterov momentum only started to slightly overfit until epoch 400.
- Something we can conclude from this is that Adam and RMSprop model improves very quickly. They both converges to the low loss/ high accuracy values (in terms of training set) only after 200 epochs. Whereas Adagrad, SGD and SGD + Nesterov take much longer to reach convergence value. For example, around the 350th epoch, SGD still stands at 75% accuracy, while at the 800th epoch, it reaches ~80% accuracy (for both training and validation)

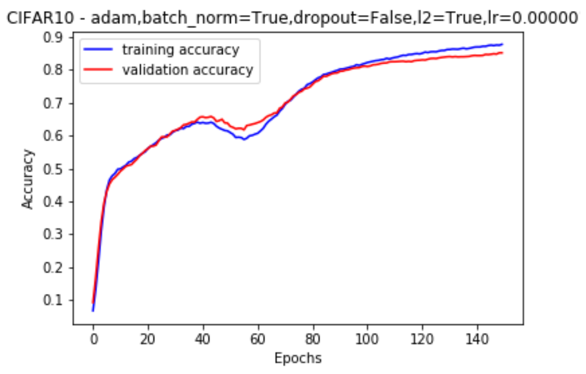
c. Regularizer:

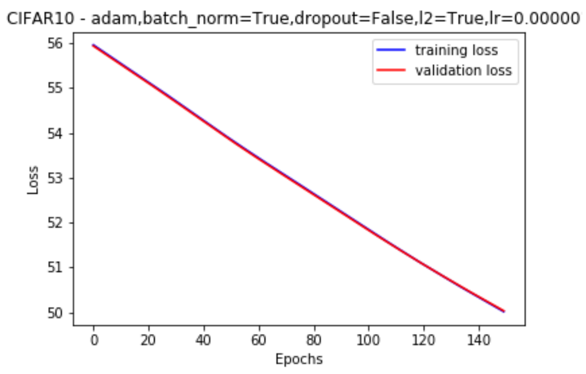
- I first tested weight decay (L2), batch normalization and dropout by testing all possible permutations of them. For example, use L2, don't use batch norm, use dropout, or use all 3, or don't use any, etc… Since there are 8 total possible permutations, which correspond to 8 graphs, I'll **only include** the graph where no regularization was used, and the graph with best performing regularization setting (in terms of reducing overfitting):

  NOTE: learning rate = 0.000001, not 0.00000 like displayed in the below graphs

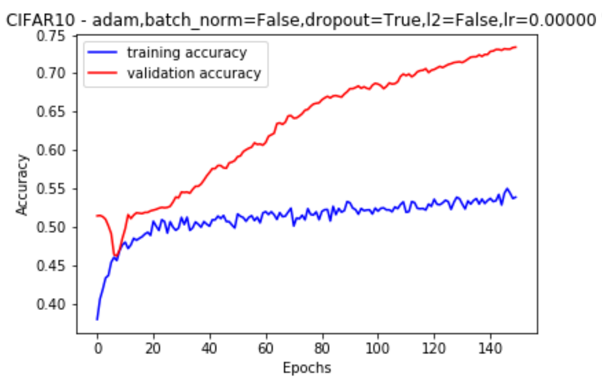


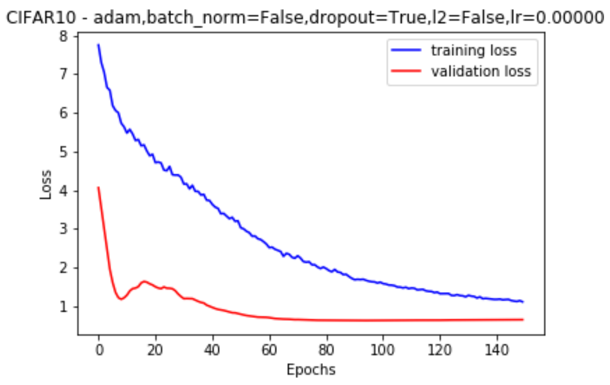






- As can be seen, using both batch normalization and l2 seems to reduce overfitting by quite a lot while still retaining high model accuracy (~90%). Overall, from all 8 graphs, one thing we can conclude is that batch normalization is quite effective at reducing overfitting, even when it's used alone. L2 by itself only reduces overfitting by a little bit. However, dropout seems like it does not really help with overfitting and it also restricts the model ability to learn the training set by too much (I set dropout p = 0.2, when set to 0.5, the result is the same). For example, when used alone, dropout restricts the model learning on training set to the point the accuracy only hovers around 55%, far less than the validation accuracy:





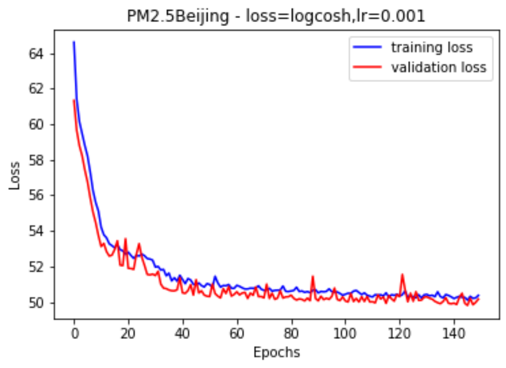
The same behavior can be seen when dropout is used with other regularizers.
NOTE: please tell me if you need all 8 graphs.

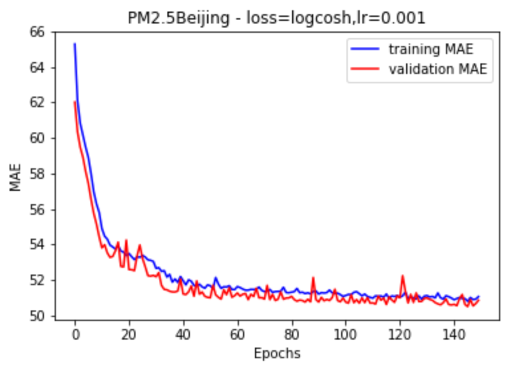
2. Regression:

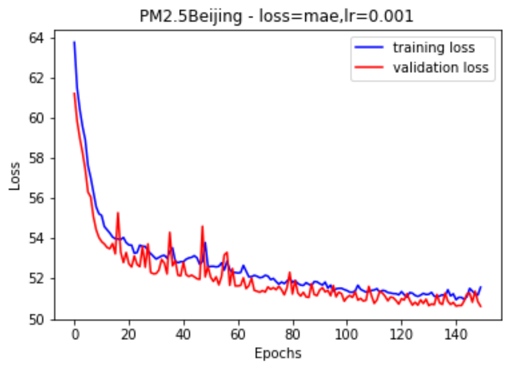
- Dataset: Beijing pm2.5 with around 43k instances and 7 attributes

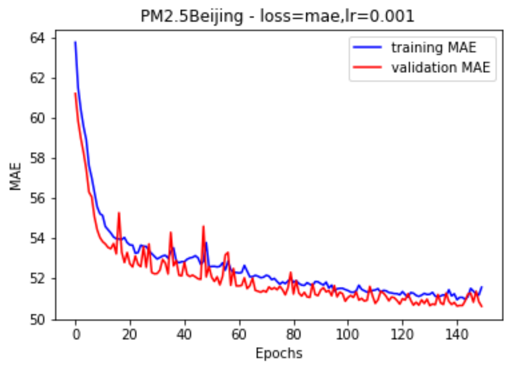
a. Evaluating losses:

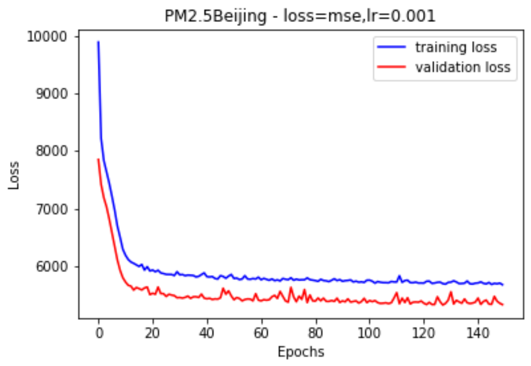
- Since this is a regression task, I chose logcosh, mae, mse to be the losses to be evaluated with accuracy metric = mae:

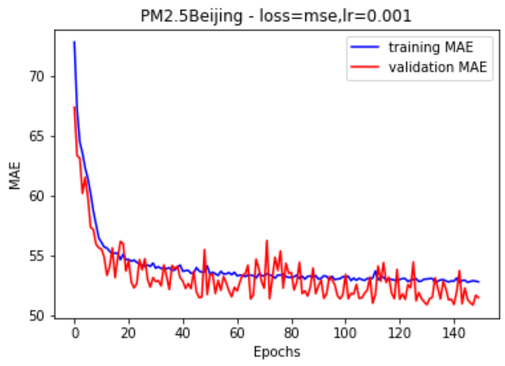








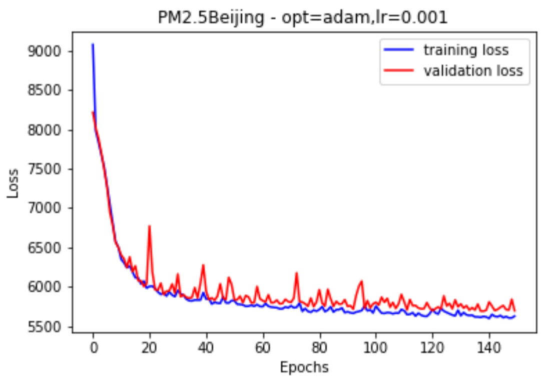


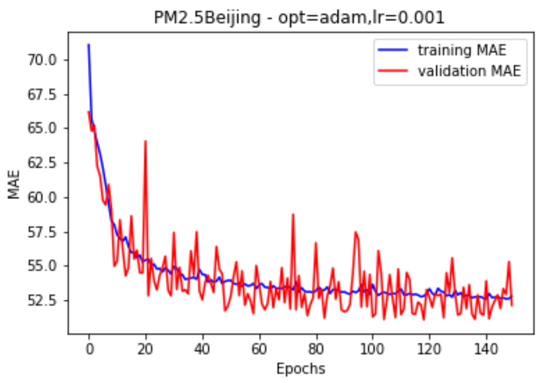


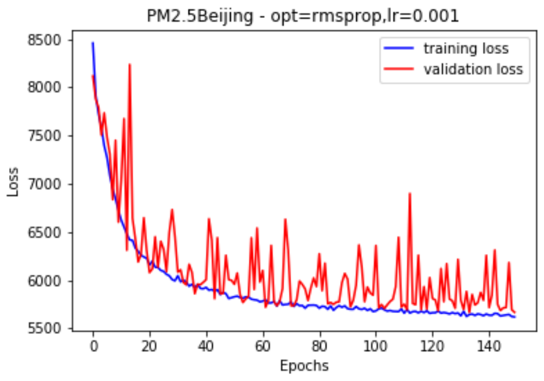
- Notice how for logcosh model and mae model, the loss vs epochs graph looks basically the same as the MAE vs epochs graph. And notice how the loss value scale of the mse model is much bigger than the mae and logcosh model (thousands vs double digit). This is actually expected, and corresponds to the definition of these losses:
  + For mae model, its graphs look the same because we're using MAE for both the loss **and** the accuracy metric
  + Now, we can see that the loss value of mse model is very high (around 6000 upon convergence). This means that the errors between the predictions and true values are pretty big, and thus logcosh for it will be linear (definition of logcosh). Thus logcosh in this case will work mostly similar to MAE. This explains why the loss value scale of logcosh is similar to mae model, and why for logcosh model, its loss and mae graphs also look the same (similar to mae model's graphs)
- We can also verify how the logcosh and mae models are less sensitive to errors/outliers compared to mse model, as their MAE vs epochs graphs are smoother than mse model's graph.

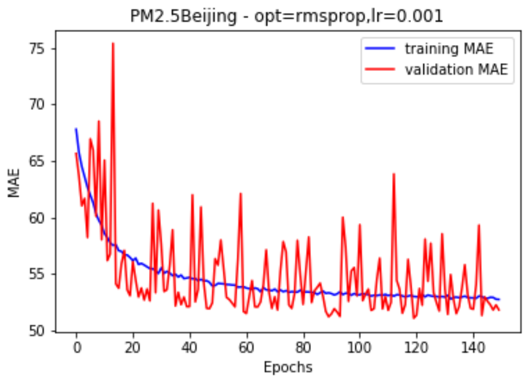
b. Evaluating optimizers:

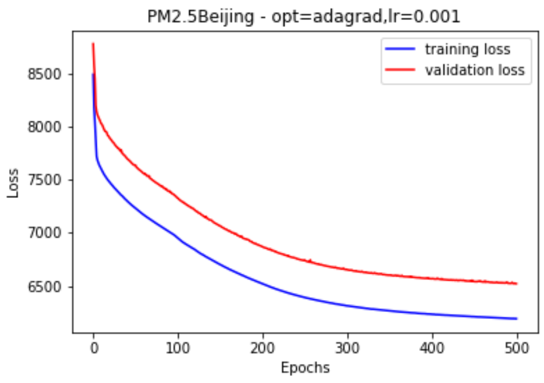
- Optimizers to evaluate include adam, rmsprop, adagrad, sgd and sgd with nesterov momentum:

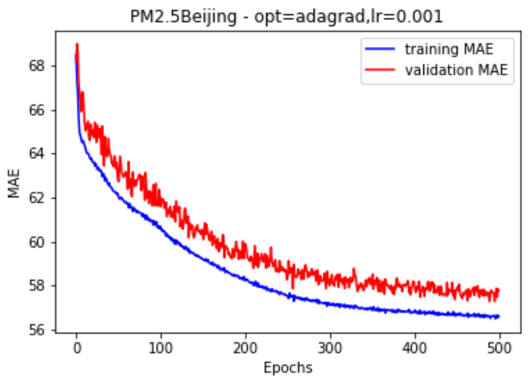


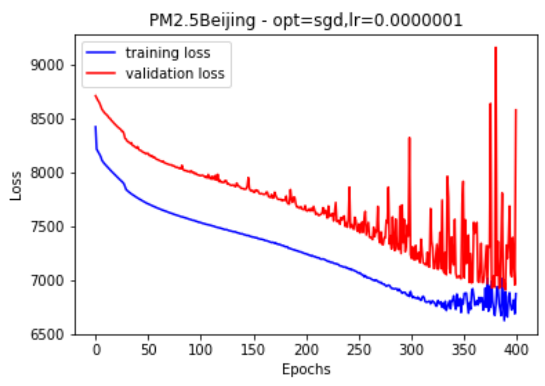


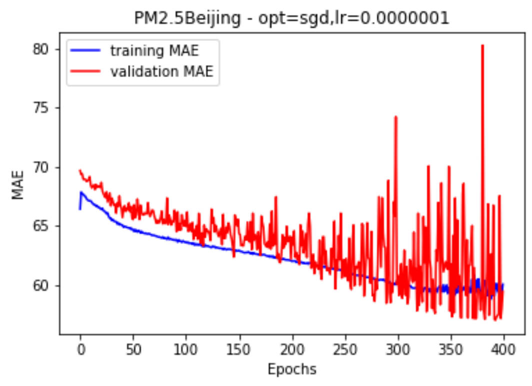


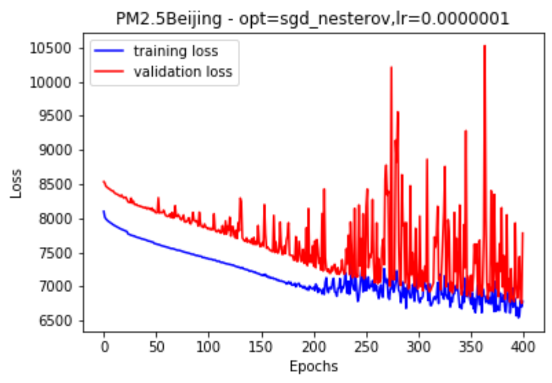


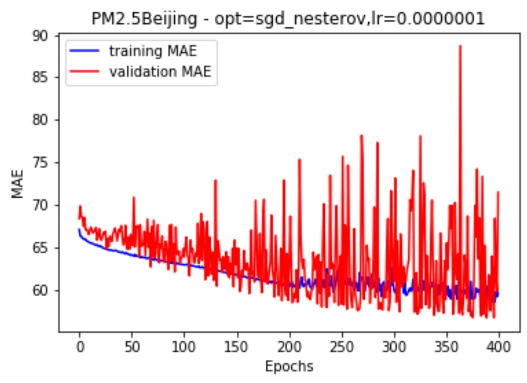










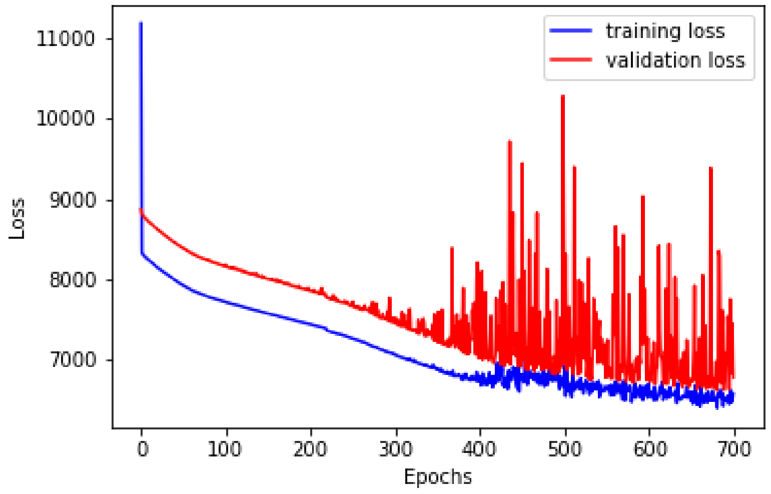


- For SGD and SGD + nesterov, I had to change the learning rate from 0.01 to very small learning rate such as 0.0000001 since 0.01 learning rate was producing all NaNs for loss and MAE values.
- First one thing to notice is that similar to classification task, Adagrad, SGD and SGD + nesterov takes more epochs to converge (with overfitting) compared to Adam and RMSprop.
- Regarding the epochs needed to converge without overfitting, Adam nearly didn't overfit throughout its 140 epochs. RMSprop suffered very high variance in validation loss value right around epoch 10. Adagrad did not really overfit throughout its 400 epochs, because even though the validation loss is higher than the training loss from the very beginning epoch, the difference is small with respect to the loss value scale, and throughout the 400 epochs, the validation loss still keeps decreasing along with the training loss with no major variance in (loss) values; that is, the validation loss never started increasing or became wildly unstable. Thus technically, Adagrad model didn't really overfit. SGD and SGD + nesterov started to consistently suffer from very high variance in loss value after 325 and 210 epochs respectively.
- Overall, the models that seem like they are stable and not much overfitting are the Adam and Adagrad models.

c. Evaluating regularizer:

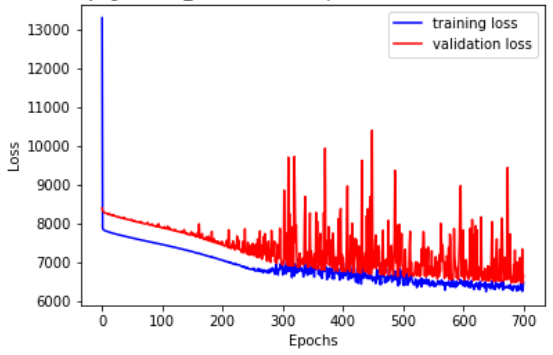
- For the evaluation of regularizer, I chose sgd to be the optimizer. Again the setup is similar to the classification task where all 8 possible permutations are used.
- The following is the loss vs epoch graph without any regularization:

PM2.5Beijing - batch_norm=False,dropout=False,l2=False,lr=0.000000

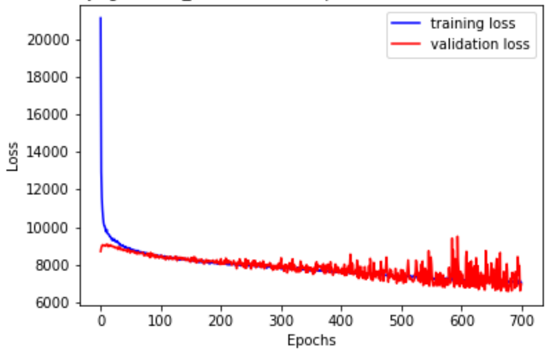


- Overall, l2 regularization by itself is not very effective, while dropout on its own is quite effective at reducing the difference between the validation loss and training loss but some amount of variance in loss value was still present:

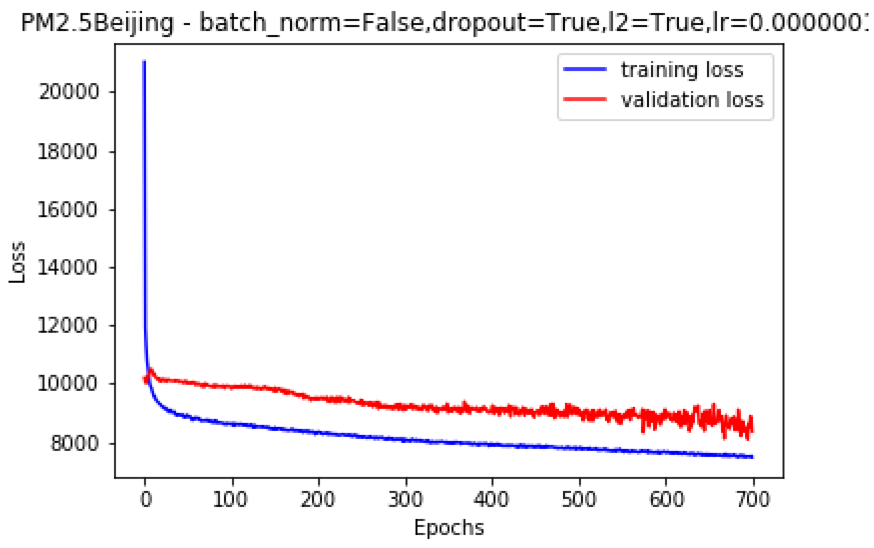
PM2.5Beijing - batch_norm=False,dropout=False,l2=True,lr=0.000000



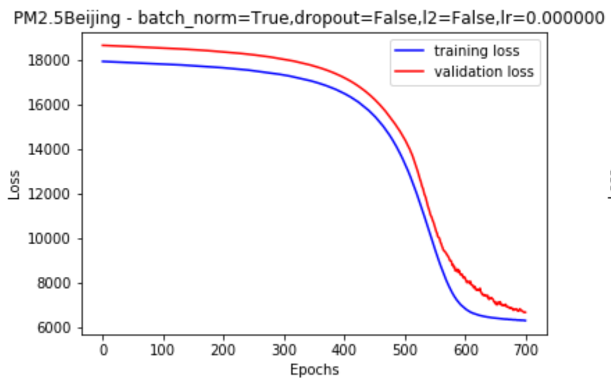
PM2.5Beijing - batch_norm=False,dropout=True,l2=False,lr=0.000000

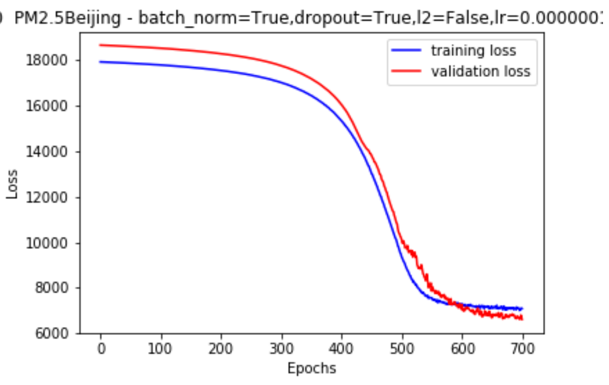


- When L2 and dropout are used together, they work quite well on reducing the large variance in loss values but they were not able to reduce the difference between the validation loss and the training loss by much:



- On the other hand, batch norm, either by itself or combined with dropout and/or l2, is very effective. Not only does it do a good job of **reducing large variance** but also **almost entirely** closes the gap between the validation loss and training loss. It also managed to bring the validation and training loss into the 6000s range. However, it took the batch norm model around 650-700 epochs to converge while it only took 350 epochs for dropout and l2 model. For example, the following are the results of batch norm used by itself, and batch norm + dropout:





- From both the classification and regression task, we can conclude that batch normalization is very effective in dealing with overfitting, no matter if it's used by itself or combined with other regularizers