Research Question: Could batch normalization have a positive effect on the classification performance of our Recurrent Neural Network?

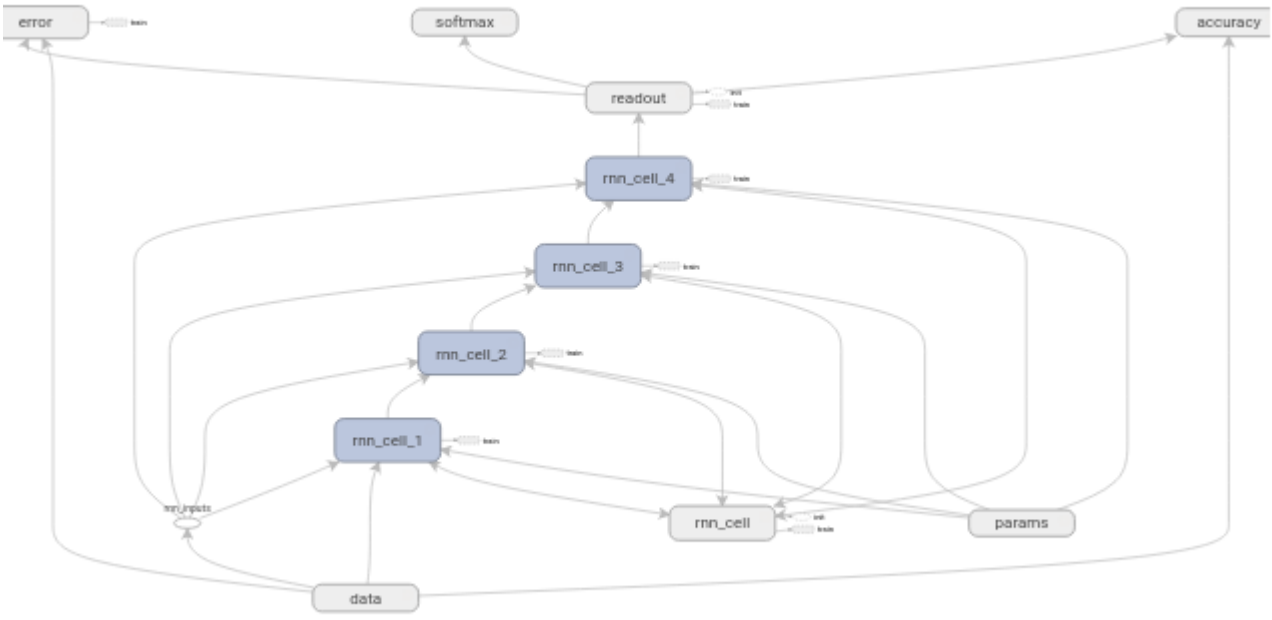
Now that we have achieved a fast working system it makes sense to seek how could we achieve a higher validation accuracy.

From previous experiments in coursework3 we have seen the benefits of batch normalization in terms of performance for both speed and higher classification accuracy for the MLP case of the Neural Network. We have also seen that batch normalization reduces the variance of the weights of the corresponding layer which is the factor responsible for the benefits it provides.
It seems that our current implementation of our basic Recurrent Neural Network suffers from exactly this issue of high variance of the validation error and accuracy. We hypothesize that batch normalization within the RNN cell will help mitigate this issue.

Implementing Recurrent Neural Network with Batch Normalization

The Recurrent Neural Network with the support of Batch Normalization is implemented in `RNNBatchNorm` class found in `rnn.rnn_batch_norm` module. This class extends the `ManualRNN` class and overrides functions related to batch normalization.

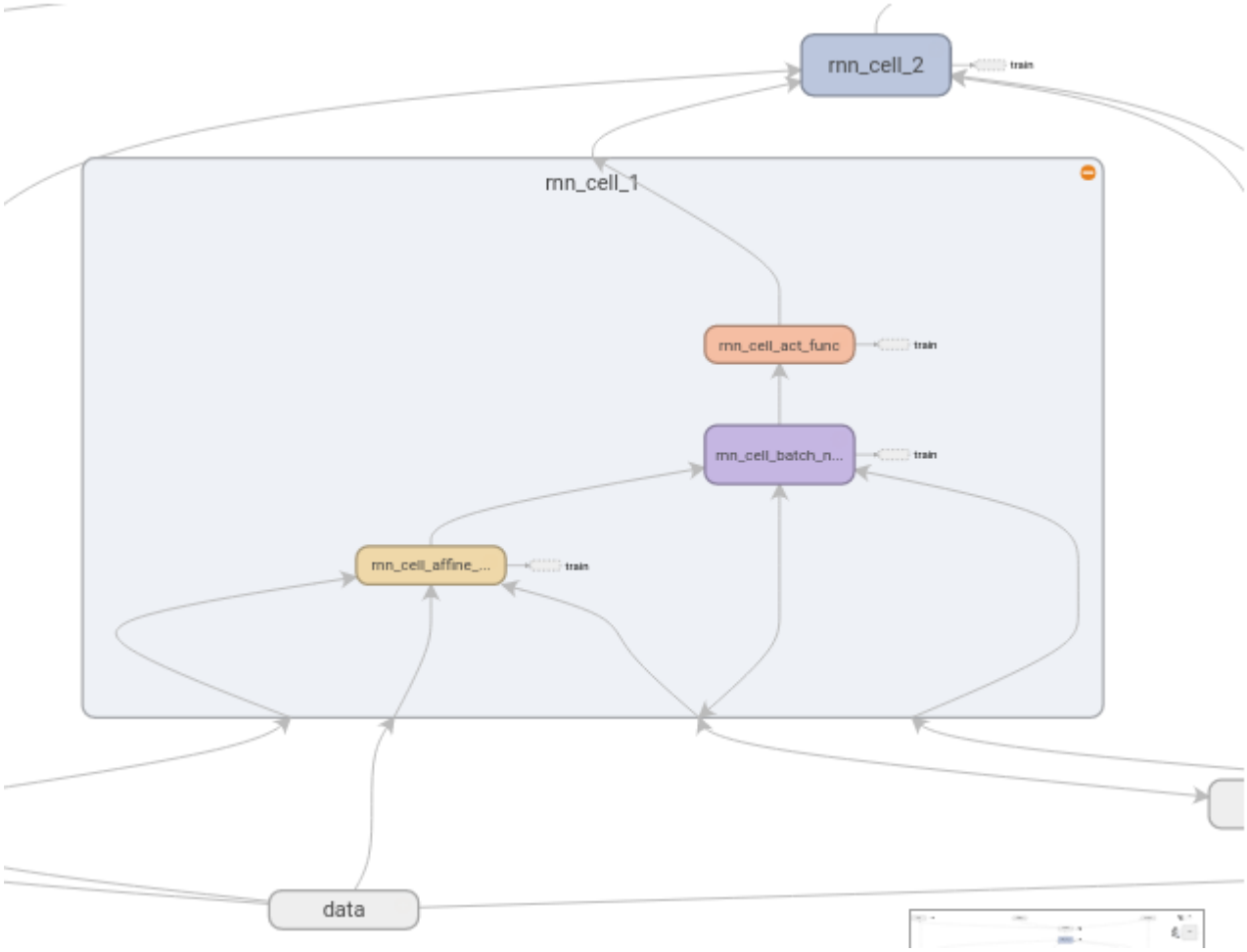
First of all we override `getGraph` method because we want to implement a new tensorflow graph that includes a batch normalization. The new graph is similar to the one we used for the basic RNN and it is show here:



Graph 6: Basic Recurrent Neural Network with Batch Normalization – RNN steps: 4

Note that the Softmax node is not connected anywhere because it is used only to produce the predictions for the Kaggle in-class competition.

The batch normalization layer is being placed between the affine layer and the non-linearity, here tanh, as shown here:



Graph 7: RNN cell of Basic Recurrent Neural Network with Batch Normalization

Note that the `RNNBatchNorm` class overrides the function `batchNormWrapper_byExponentialMovingAvg` found in the `mylibs.batch_norm` module in order to provide an implementation of batch normalization which works with shared variables by using the `tf.get_variable` method from Tensorflow. Note that the hyperparameter epsilon of batch normalization stays constant at 1e-3.

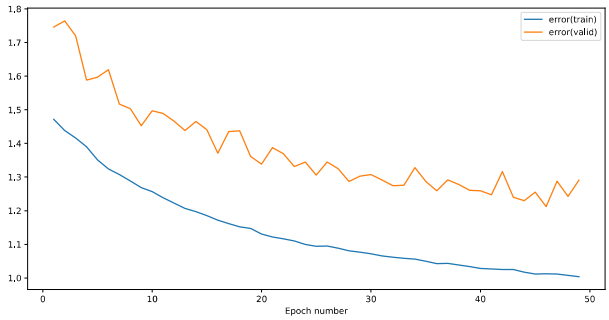
Experiments with RNN with Batch Normalization

We are executing the experiment for **50 epochs** with the best parameters we had for our basic recurrent neural network:

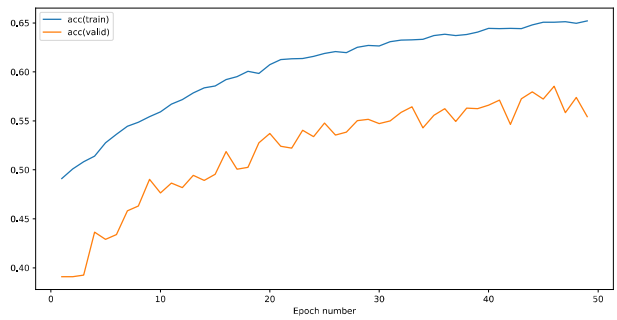
- **State Size**: 341
- **Number of RNN steps**: 4

Results

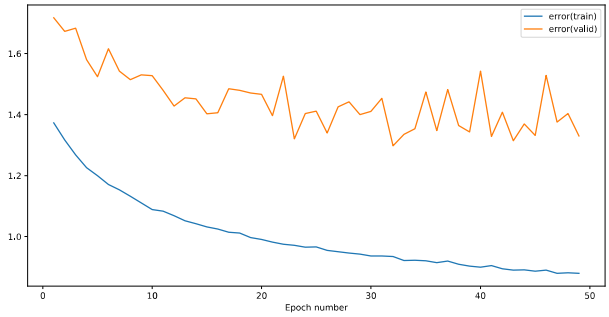
Note that we are putting the results without Batch Normalization underneath for comparison purposes.



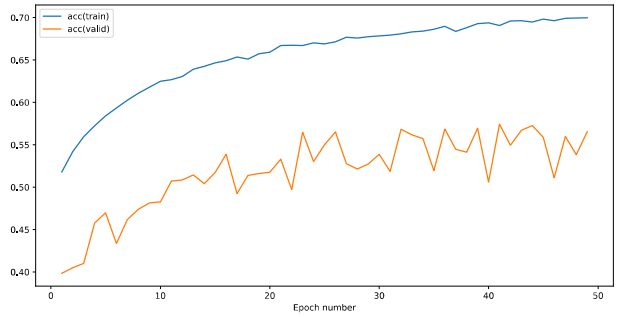
Plot 61: Training & Validation Error - Basic Recurrent Neural Network with Batch Normalization



Plot 62: Training & Validation Accuracy - Basic Recurrent Neural Network with Batch Normalization



Plot 63: Training & Validation Error - Basic Recurrent Neural Network without batch normalization



Plot 64: Training & Validation Accuracy - Basic Recurrent Neural Network without batch normalization

Conclusions

Comparing the plots above it is obvious from both the validation error and validation accuracy that the architecture which lacks batch normalization has a larger variance than the architecture which includes batch normalization.

Lower variance has the benefit of easier learning and this is reflected at the higher validation accuracy achieved within the same epochs which peaked over 58% while previous we were only above 57%.

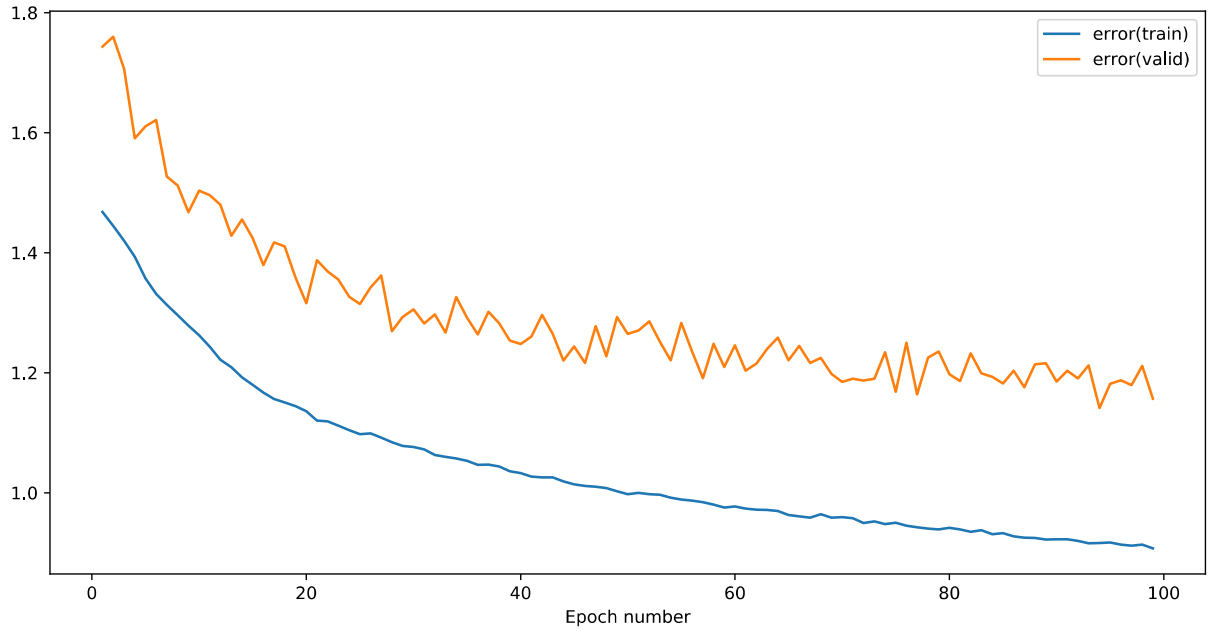
Now that the increasing trend is more obvious it makes sense to run training for more epochs to see how high it could reach.

RNN with Batch Normalization Experiment for 100 epochs

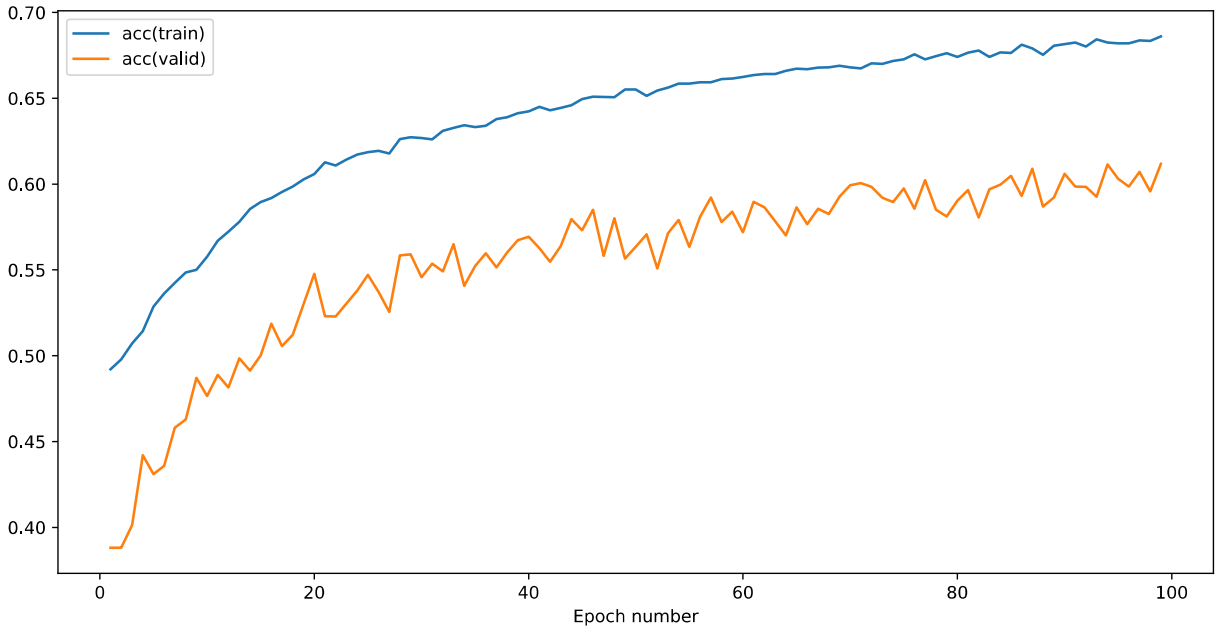
We are executing the experiment for **100 epochs** with the best parameters:

- **State Size**: 341
- **Number of RNN steps**: 4

Results



Plot 65: Training & Validation Error - Basic Recurrent Neural Network with Batch Normalization – Epochs: 100 – State Size: 341 – RNN steps: 4



Plot 66: Training & Validation Accuracy - Basic Recurrent Neural Network with Batch Normalization – Epochs: 100 – State Size: 341 – RNN steps: 4

Conclusions

We are pleased with the results from running training for 100 epochs since we have peaked at 61.17% without getting any clear indications of overfitting.
If we had more computational resources available it would be interesting to see how it performs for two hundred or more epochs.

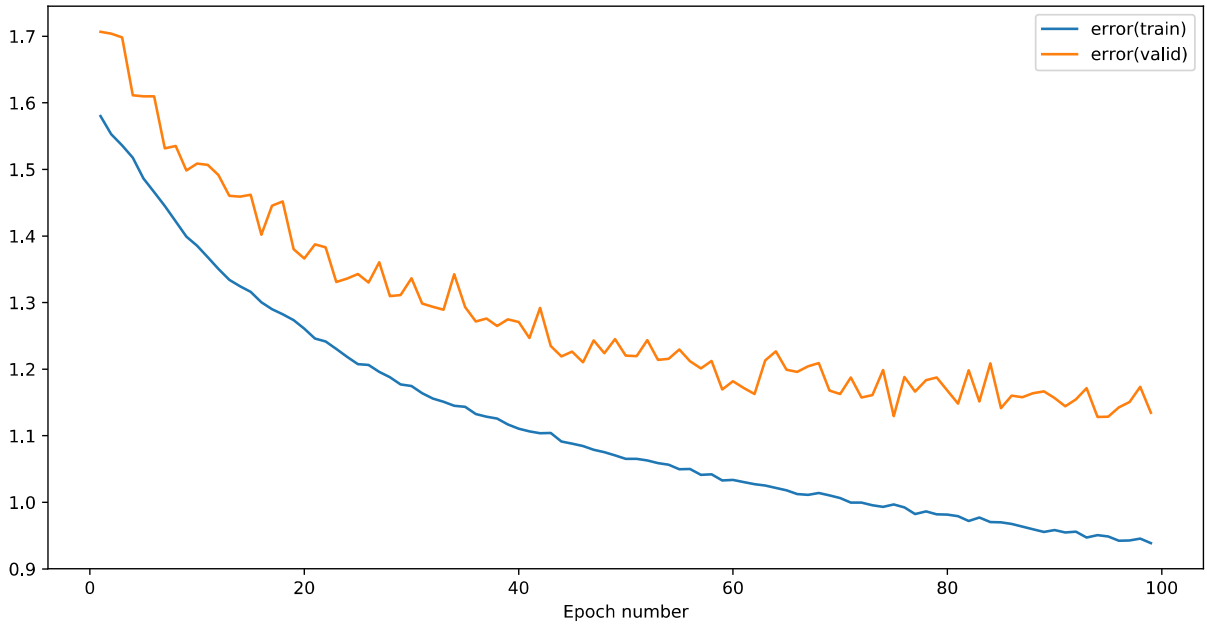
Now it is worthwhile to compare against the 8 rnn steps which had shown promising results before when we were doing the grid search, to see if it performs better or worse.

RNN with Batch Normalization Experiment for 100 epochs and with 8 rnn steps

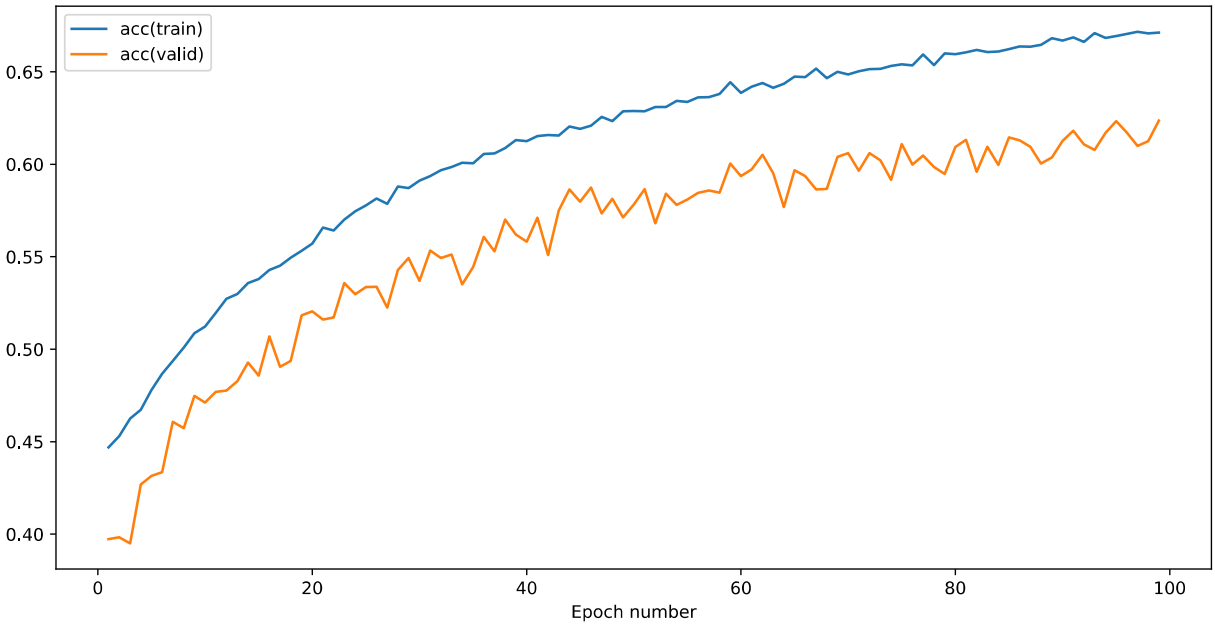
We are executing the experiment for **100 epochs** with parameters:

- **State Size**: 341
- **Number of RNN steps**: 8

Results



Plot 67: Training & Validation Error - Basic Recurrent Neural Network with Batch Normalization – Epochs: 100 – State Size: 341 – RNN steps: 8



Plot 68: Training & Validation Accuracy - Basic Recurrent Neural Network with Batch Normalization – Epochs: 100 – State Size: 341 – RNN steps: 8

Conclusions

We are pleased with the results of this experiment since we peaked at over 62% which is a new record. The difference is not considered substantial enough to give definite results between 4 and 8 RNN steps and in addition the state size was considered fixed and it is a result of another optimization.

Future work (not implemented)

- Investigating Architectures like LSTMs and GRUs
- Investigating which feature, Chrome, Timbre or Loudness is the most prominent in each music class
- Data Augmentation with other ways than Dropout. For example Gaussian Noise
- Experiment with the Variable Length Dataset
- Splitting the Variable Length Dataset into fixed length parts in order to artificially get more instances