Research Question: Could we train a shallow neural network to be as good as our basic Recurrent Neural Network in terms of classification accuracy?

We have achieved a higher validation accuracy with our basic RNN but the model has multiple layers and this is a much larger model, in terms of parameters, than the shallow neural networks we tested when first encountered the MSD-10 classification task.
This bring a speed performance issue when we would like to classify a million songs that would have to pass through this network. The question is whether we could train a shallow neural network to replicate the behavior of the more complex deep recurrent neural network, by following the so called Teacher-Student architecture.

The Teacher-Student architecture is based on the presentation slides found on the following link.
Mimicking deep neural networks with shallow and narrow networks: http://web.eng.tau.ac.il/deep_learn/wp-content/uploads/2016/12/Mimicking-deep-neural-networks-with-shallow-and-narrow-networks.pptx

The first step in order to build the Teacher-Student architecture is to pass unlabelled data through the deep recurrent neural network and collect all the outputs. Note that we care for the logits and not for the processed softmax outputs in order to have values which are within a constrained range rather than exponential values produced by the softmax function. This will help us more easier to pick a suitable cost function for our student model later.

The issue is that until now in our MSD-10 dataset we have considered the train dataset as having the labelled data and the validation dataset as having the unlabelled data. However we need the outputs of as much unlabelled data as we can in order to have enough data to train the Student model later.

Cross Validation

Our solution will be **Cross-Validation** but we need to make a number of changes to fully implement it.

First we are building the `CrossValidator` class found in the `rnn.cross_validator` module. To get the instances of each fold of the k-fold cross validation, of training and validation instances, we are exploiting the `StratifiedKFold` class from the `model_selection` module of scikit-learn library.
Note that we need to use Stratified KFold and not the simple KFold because we have a balanced classification task and we need to maintain this state in order to avoid having to use techniques that need with imbalanced classification tasks.

First we have created a small python script named concat_train_valid.py found inside the data folder which is responsible for merging the training and thevalidation datasets under the same file, in our case *msd-10-genre-train_valid.npz* file. Concatenation merges 40000 with 10000 songs to a total of 50000 songs.

Note that StratifiedKFold process is set to shuffle the instances. So if we set a k-fold of k=10 we are going to have 45000 shuffled instances as training set and the rest 5000 instances as validation set, also shuffled.

This brings two new requirements for our data providers:

- Our data provider needs to be able to filter the instances according to a list of indices which correspond to the ids of the instances that we need to consume.
- Our data provider needs to keep the references of the original ids of the instances even if it is working with a filtered dataset.

So if we need to have 45000 instances only, for example for the training dataset, we need to keep only these 45000 from the full 50000 instances and discard the rest.
We have enhanced `MSD10Genre_120_rnn_DataProvider` class with an extra parameter at the constructor named `indices`. The indices are the indices of the instances that we wish to keep. Whenever indices is set to None

we fallback to the original implementation where all indices are being kept.
Next step is to enhance the `MSD10GenreDataProvider` class. Its constructor has a `data_filtering` callback parameter which should be a function that takes inputs and targets as inputs, process them and returns them as outputs.
The implementation of `data_filtering` inside the `MSD10Genre_120_rnn_DataProvider` class is an one liner implemented with lambdas. We are keeping the inputs and targets that correspond to the indices parameter is indices is not None and it is an array.

For the second requirement we have included an extra parameter at the constructor our base `DataProvider` class named `initial_order`.
Until now we were considering the ids of the inputs and targets to be a serial number from zero to the length of the dataset minus one but now we need to accommodate custom ids which are initiated with a specific order. This requires two changes.
Firstly the `_current_order` attribute of the class is initialized with `initial_order` unless the `initial_order` is None which is then initialized with numpy's `arange` function as it was the original implementation.
The second and most difficult change is to find the inverted permutation of the current order in case we want to use the reset method. Here we take advantage of numpy's `argwhere` function in order to find in which indices the values of the `initial_order` are now located on the permutated `_current_order`.

Logits Gathering

Our purpose here however it is not to do cross-validation for the classification task but rather to gather logits from unlabelled data the best model in each fold of the k-fold cross-validation process.

For this purpose we have created the `LogitsGatherer` class found in `rnn.logits_gatherer` module. This implements the `getLogits` method which iterates over all of the instances of the data provider passed as parameter and is building a python dictionary.
The keys of the dictionary are the ids of the instances as provided from _current_order for the way the instances are currently permuted.
The values of the dictionary are the vector of the, ten(10) here, logits from the neural network.

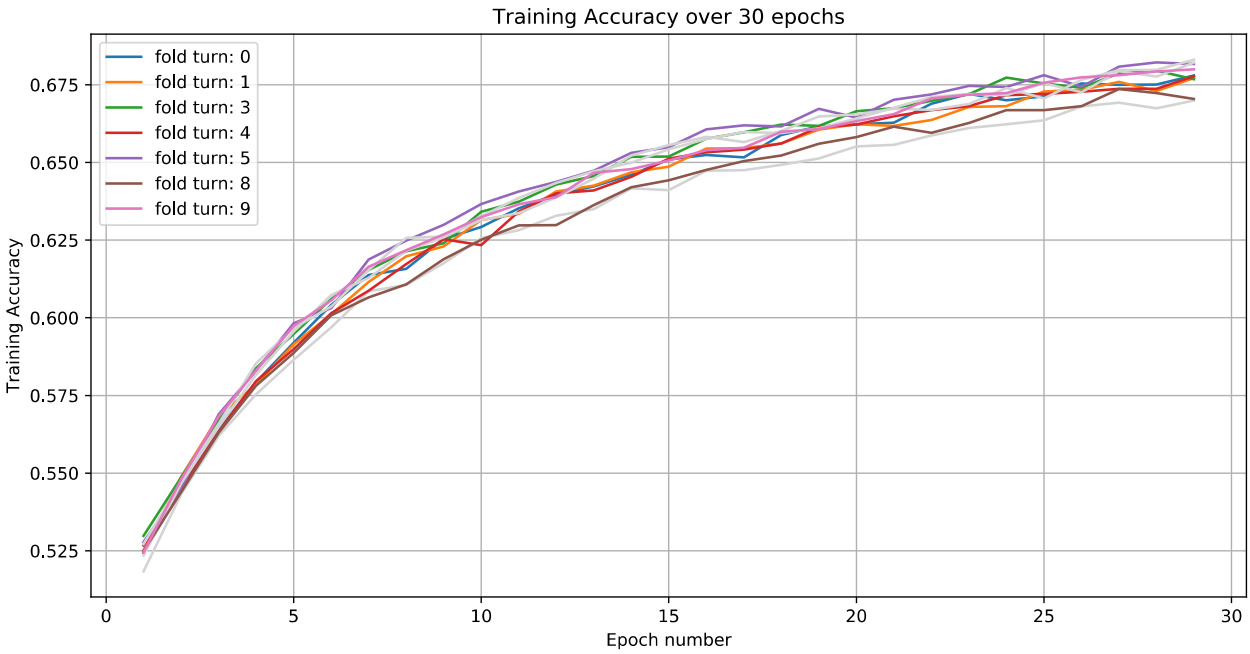
When training/validating our Recurrent Neural Network we are calling getLogits method on the first epoch but not on all consecutive epochs. We are calling getLogits again upon the condition that the newly calculated validation accuracy is larger than any of the previous ones.

Cross Validation and Logits Gathering execution

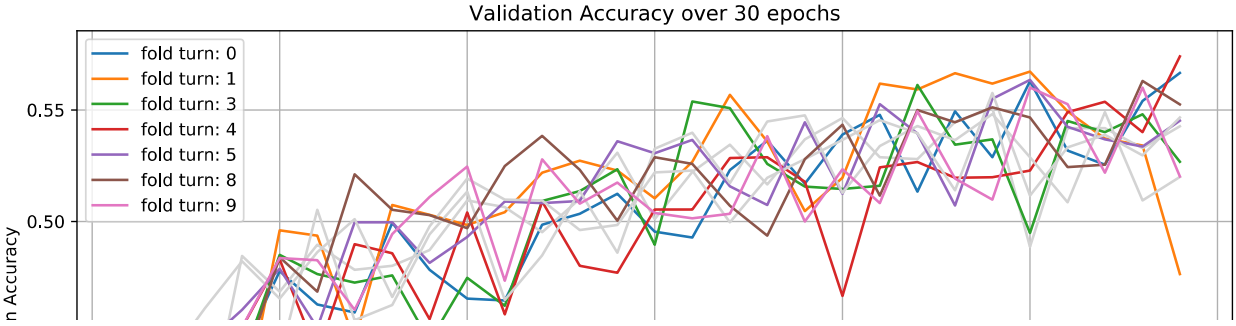
We used **k=10 fold Stratified** to have a good enough balance which is commonly used as a rule of thumb of ~90-10% percent of training and validation sets respectively.

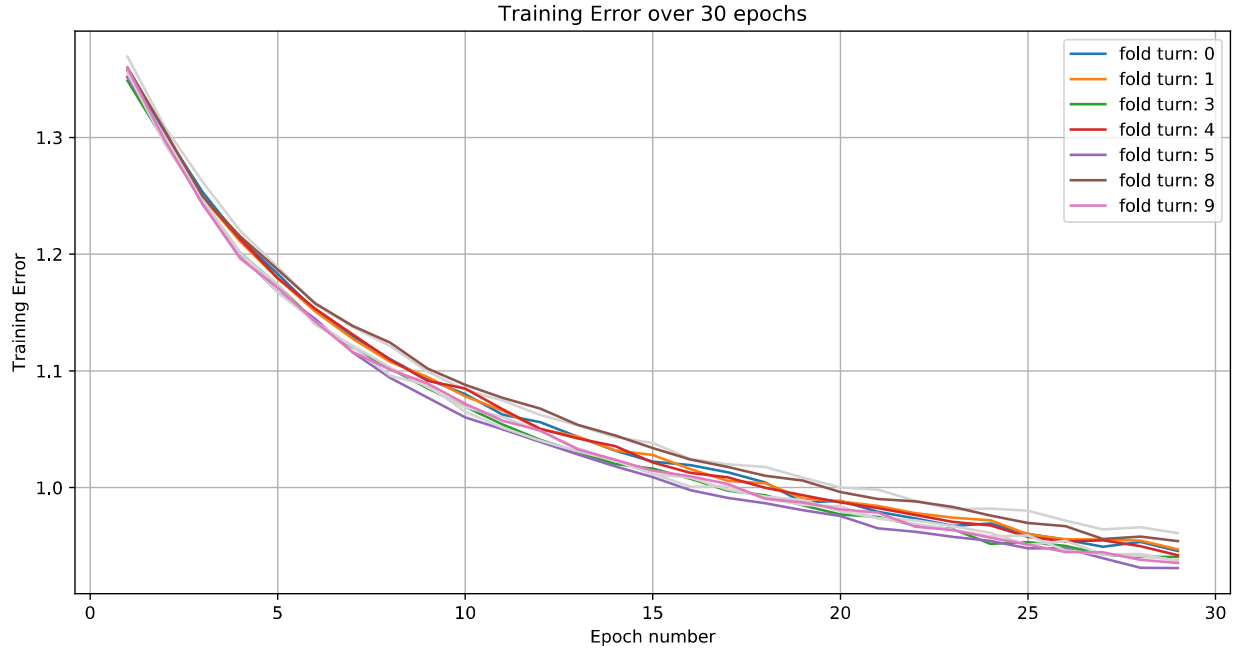
Each training was executed for **30 epochs**.

Results

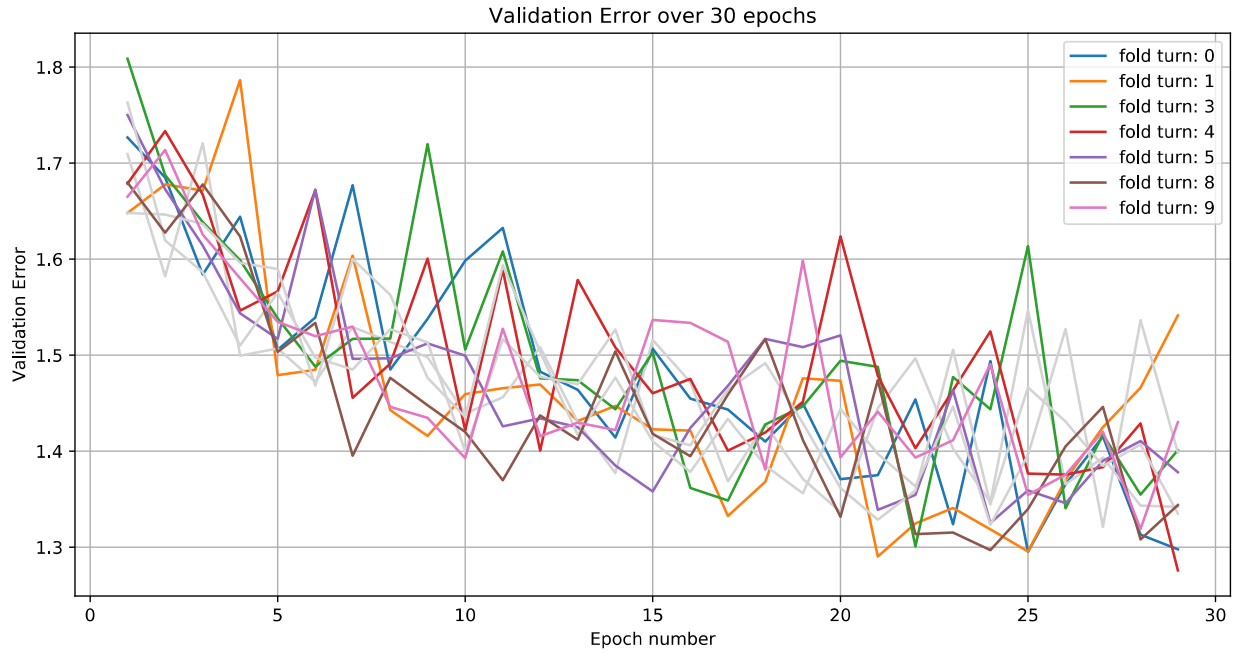


Plot 38: Training Accuracy – Stratified 10-Fold Cross-Validation for 30 Epochs – Basic Recurrent Neural Network – State Size: 341 – RNN steps: 4



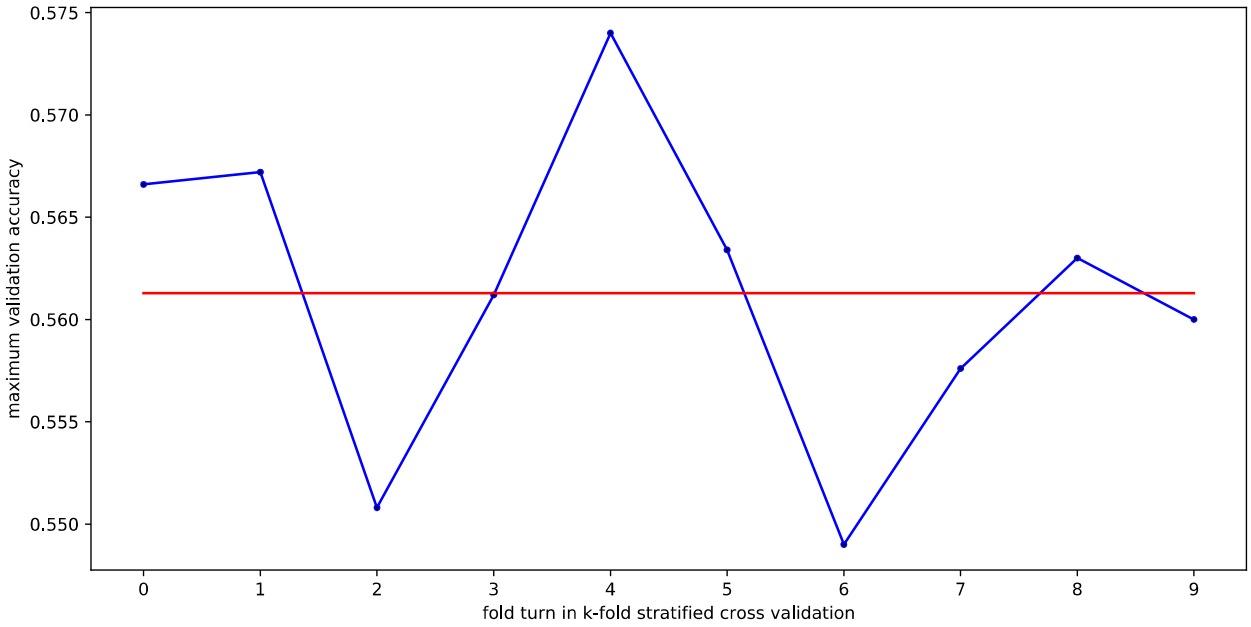


Plot 40: Training Error – Stratified 10-Fold Cross-Validation for 30 Epochs – Basic Recurrent Neural Network – State Size: 341 – RNN steps: 4



Plot 41: Validation Error – Stratified 10-Fold Cross-Validation for 30 Epochs – Basic Recurrent Neural Network – State Size: 341 – RNN steps: 4

//…



Plot 42: Maximum Validation Accuracy per fold Turn for Stratified 10-Fold Cross-Validation – 30 Epochs – Basic Recurrent Neural Network – State Size: 341 – RNN steps: 4 – Red Line represents the average value of all the max validation accuracies

Conclusions

From the plots above we see that running our Recurrent Neural Network without any regularization or batch normalization is the cause of a high variance for the weights and as a result we have the validation accuracy and error fluctuating instead of easily converging. This is valid for all fold turns of k-fold stratified cross-validation.

The fluctuation is more visible from the last plot where the maximum validation accuracy among fold turns ranges from **54.9% to 57.4%** which is a substantial difference. The mean value is noted with the red line and it is **56.13%**. Note that each one of the models that correspond to these maximum recorded accuracies are being used to generate the logits of the unlabeled data to be used as outputs of the Teacher Model and later to be used to the Student Model.

Building the Student Model

Student model is a shallow artificial neural network which contains only **one hidden layer** of arbitrary, usually large, dimensionality.
The hidden layer is followed by a **non-linearity** of our choosing, which here is the *tanh* activation function.
Finally we have a **readout affine layer** which reduces the dimensionality to the outputs corresponding to the dimensionality of the logits from the Teacher model.

The training dataset is going to use as inputs all of the 50000 we used above when we were doing cross-validation. The training targets are going to be the outputs/logits from the Teacher model.

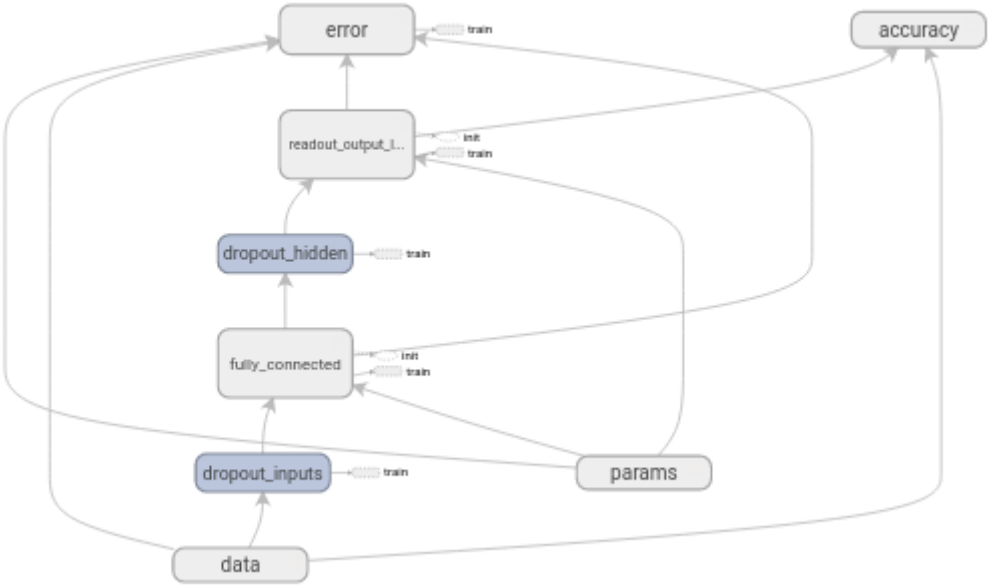
As error function we are going to use the Mean Squared Error between the outputs of the Student Models and the already processed and saved outputs of the Teacher Model.

$Loss = \frac{1}{T}(\sum_t ||\beta f(Wx) - z||^2)$ where β is the readout affine layer matrix, f is the non-linear activation function, W is the matrix of the hidden layer, x are the inputs, z are the logits outputted from the Teacher model and T is the length of the dataset.

In order to be able and validate the Student model we are going to use the testing dataset. Note that this is not ideal since we would like to leave the testing dataset out of the validation process as a way to measure the “real” performance of our model on unseen data. However due to computational restrictions we are not going to use cross-validation, as it would have been the alternative, and “sacrifice” the testing dataset to be used as our validation dataset here. This can be considered ok for the current system since there is no evidence that we have overfitted our validation dataset and thus that we have the need to use a testing dataset for more objective evaluation.

We need a new data provider for the Student model. We have implemented `MSD10Genre_Teacher_DataProvider` class in `models.teacher_student_nn` module. The difference from its base, which is `DataProvider`, is that there are two parameters at the constructor, `dataset_filename` and `logits_filename` which control the sources of the inputs and the targets of the data provider respectively.

The Student model is implemented by the `StudentNN` class found in `models.teacher_student_nn` module. The graph implemented by the StudentNN is rendered below.



Graph 5: Student model – MLP Neural Network – Dropout Layer for Inputs – Dropout Layer for Hidden Layer – Batch Normalization – L2 Regularization – One Hidden Layer plus Readout Layer

Note that there are two kinds of errors being calculated depending on the training placeholder.
If training is True then we are at training stage and we have as error the Mean Square Error of the outputs against the logits of the Teacher model.
Else if training is False then we are at the validation stage and we pass the outputs through softmax function and then take the cross entropy as we have done multiple times in previous experiments.

Also note that Dropout, L2 Regularization and Batch Normalization are included in all layers of the shallow neural network of the Student model.

Bayesian Optimization of the hyperparameters of the Student model

We would like to run multiple experiments to determine the optimal values of dropout keep probabilities, L2 regularization factor and the magnitude of the hidden dimensionality with the help of Bayesian optimization.

The space of these hyperparameters is as follows:

- Keep Dropout Probability of Input layer: Minimum 50%, Maximum 100%
- Keep Dropout Probability of Hidden Layer: Minimum 50%, Maximum 100%
- Number of Neurons for the Hidden Layer: Integer with minimum 20 and maximum 2000
- L2 regularization factor: minimum 1e-3 and maximum 10 with log-uniform distribution

We ran the bayesian optimization twice. We will provide both results and then at the end we are going to provide conclusions taking into account all experiments

Bayesian Optimization with Learning Rate 1e-5

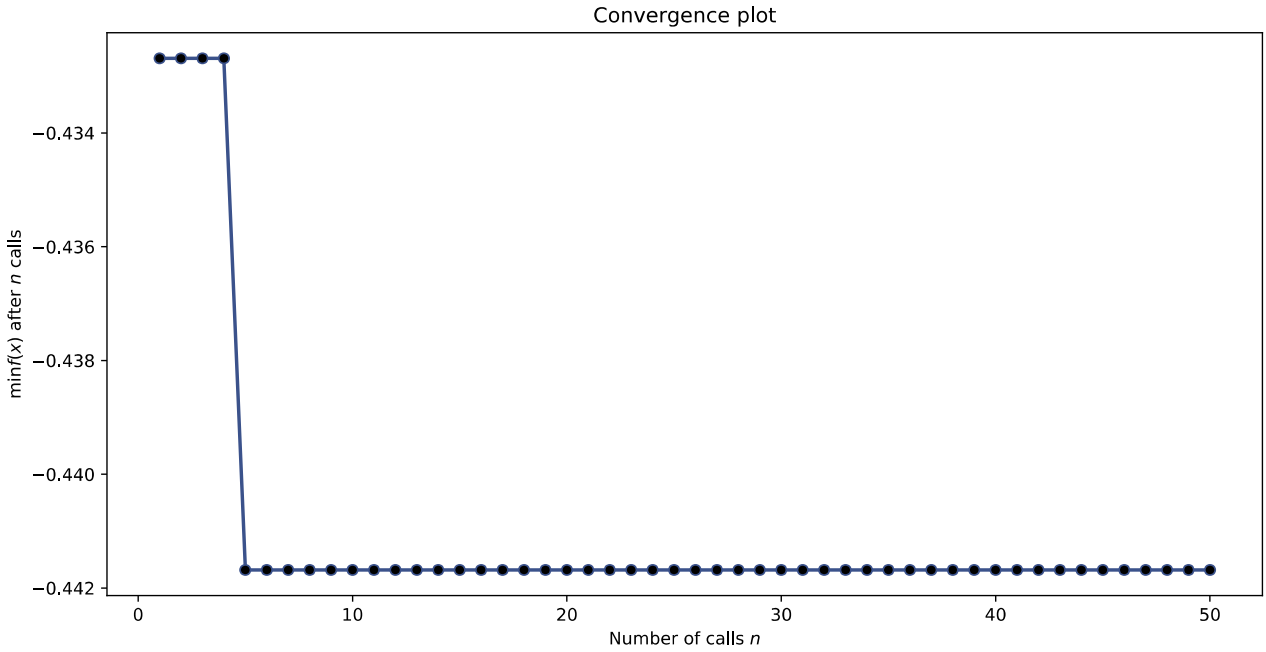
Here learning rate for optimizer is set to a low value of **1e-5**

Training runs for **40 epochs** on every call of the objective function.

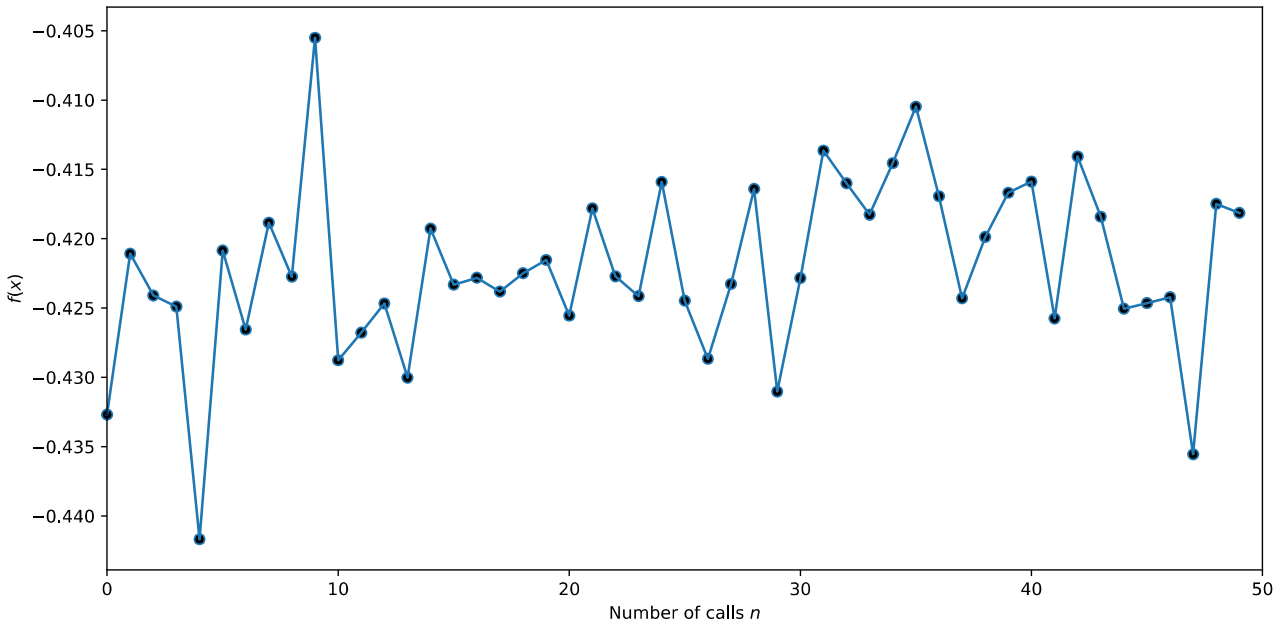
Number of calls is equal to fifty(50) and at the initialization before we start modelling we call the objective function with **random hyperparameters for five(5) times**.

Note that the objective function is trying to maximize the mean value of the 10% of the best validation accuracies which here is the top four(4) values.

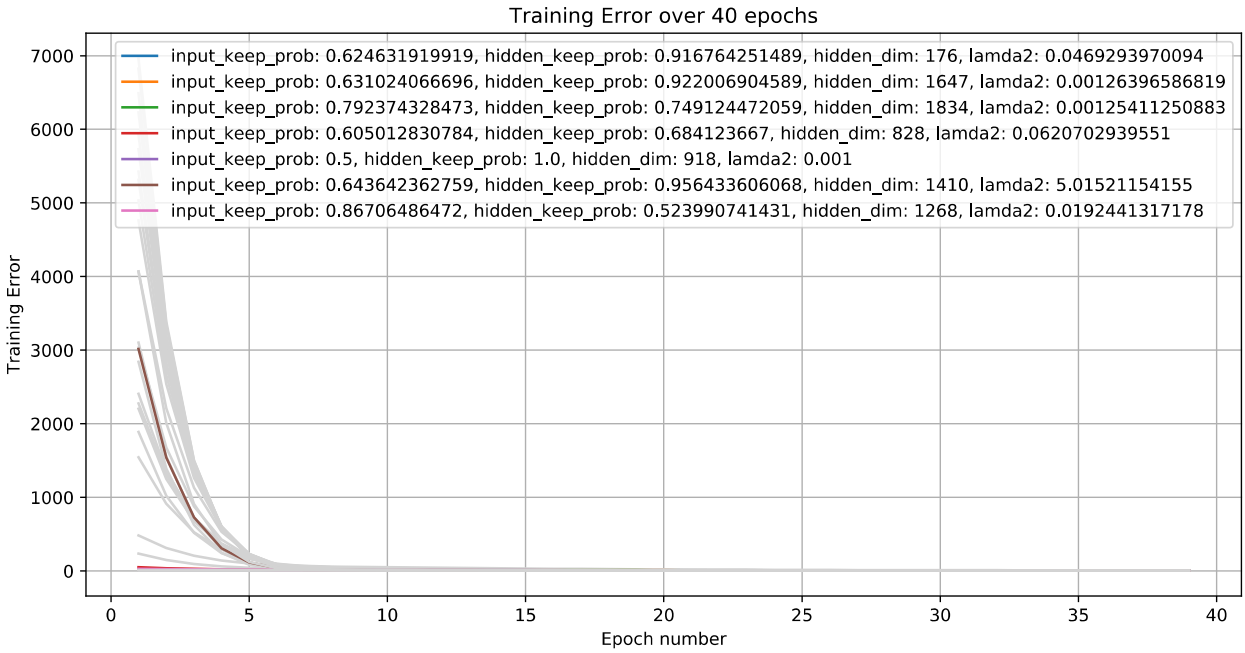
Results



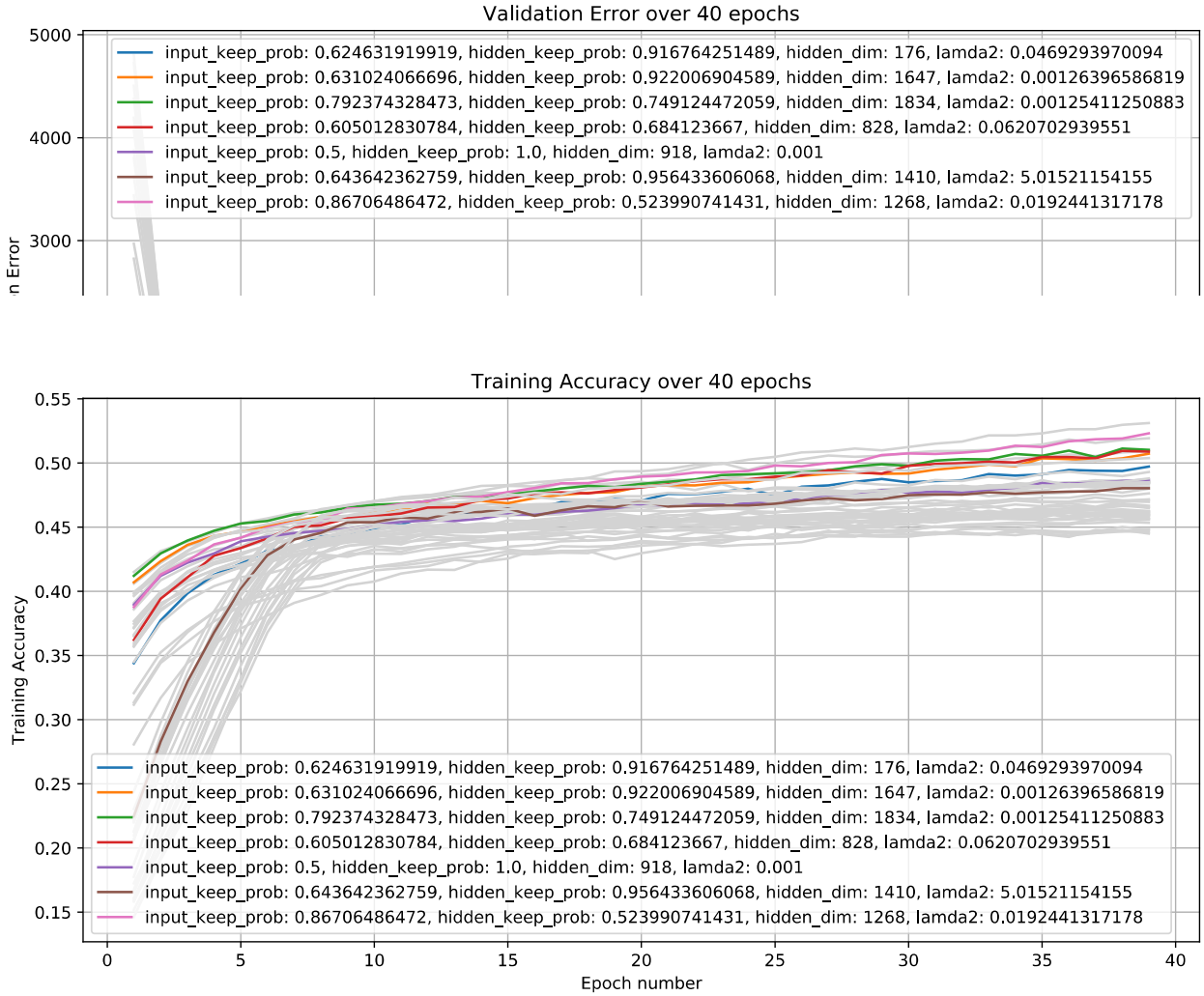
Plot 43: Convergence Plot for Bayesian Optimization trying to find optimal Hidden Dimensionality, Dropout Keep Probability of Input and Dropout Keep Probability of Hidden Layer for Student Model with Learning Rate 1e-5



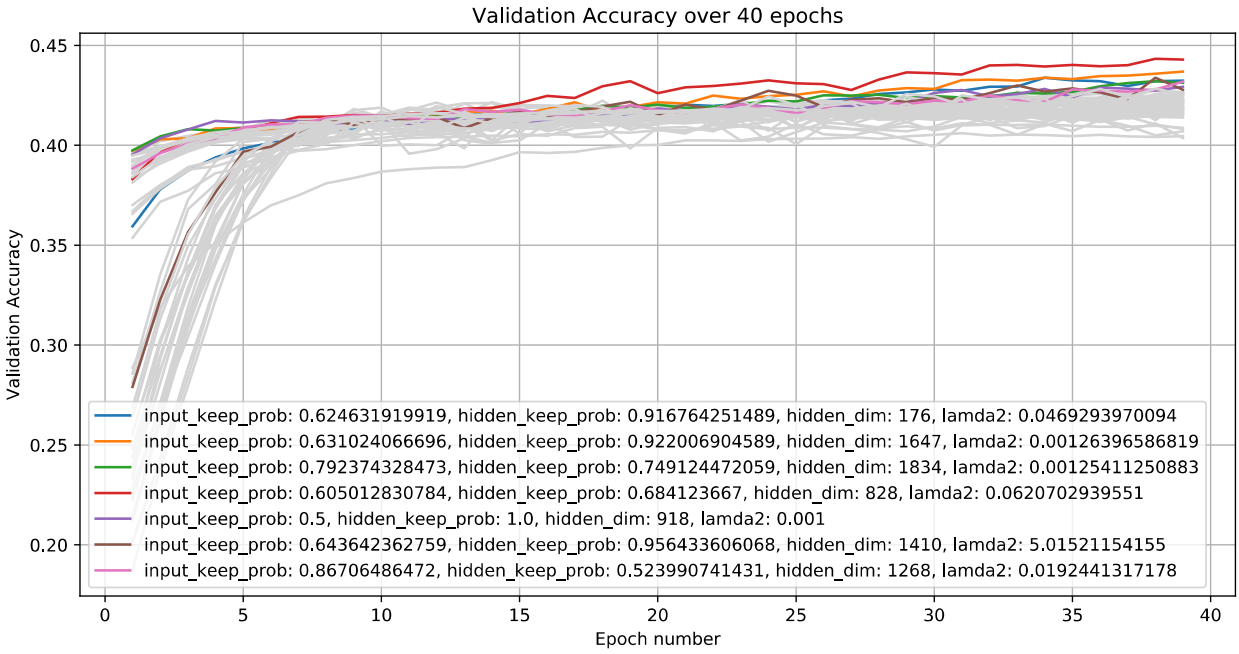
Plot 44: Output of Objective Function for Bayesian Optimization trying to find optimal Hidden Dimensionality, Dropout Keep Probability of Input and Dropout Keep Probability of Hidden Layer for Student Model with Learning Rate 1e-5



Plot 45: Training Error over 40 epochs per call for Bayesian Optimization trying to find optimal Hidden Dimensionality, Dropout Keep Probability of Input and Dropout Keep Probability of Hidden Layer for Student Model with Learning Rate 1e-5



Plot 47: Training Accuracy over 40 epochs per call for Bayesian Optimization trying to find optimal Hidden Dimensionality, Dropout Keep Probability of Input and Dropout Keep Probability of Hidden Layer for Student Model with Learning Rate 1e-5



Plot 48: Validation Accuracy over 40 epochs per call for Bayesian Optimization trying to find optimal Hidden Dimensionality, Dropout Keep Probability of Input and Dropout Keep Probability of Hidden Layer for Student Model with Learning Rate 1e-5

Best Parameters suggested from this Bayesian Optimization:

- Keep Dropout Probability of Input layer: 60.5%
- Keep Dropout Probability of Hidden Layer: 68.41%
- Number of Neurons for the Hidden Layer: 828
- L2 regularization factor: 0.062

Bayesian Optimization with Learning Rate 1e-4

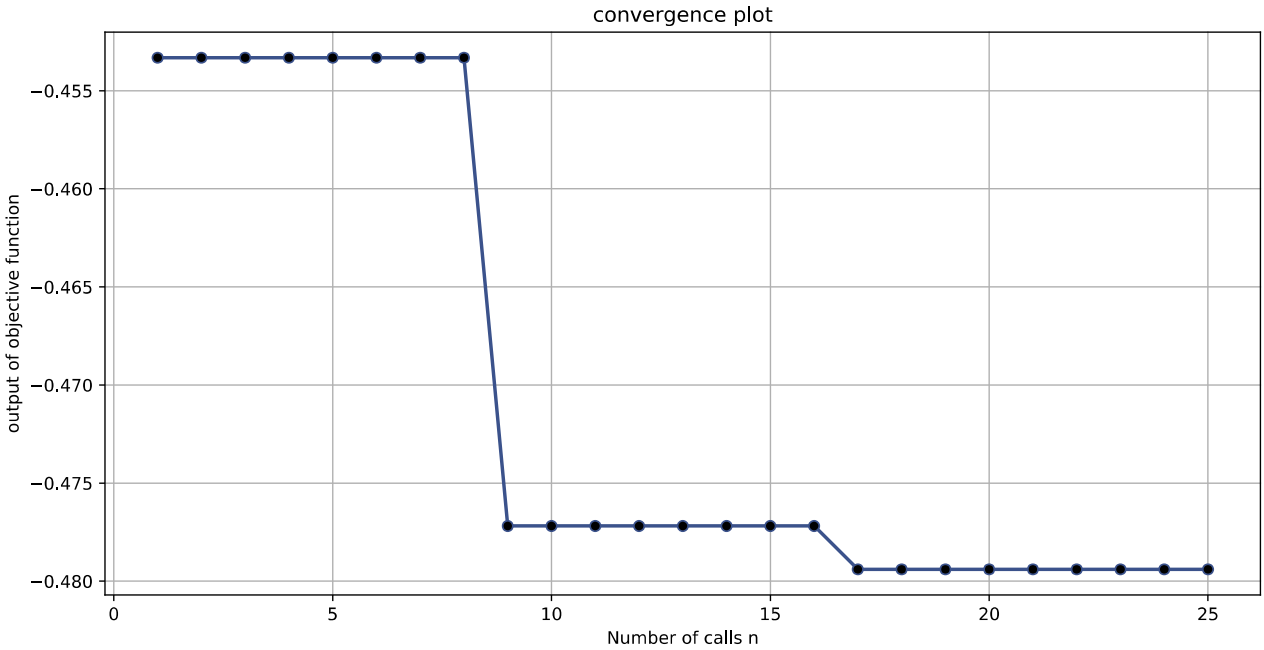
Now the learning rate of the optimizer is set ten time larger than before.

Training runs for **20 epochs** on every call of the objective function. We have used half the epochs than before because from the plots of the previous bayesian optimization we can see that at 20 epochs each model has shown evidence of being better or worse than the rest of the models.

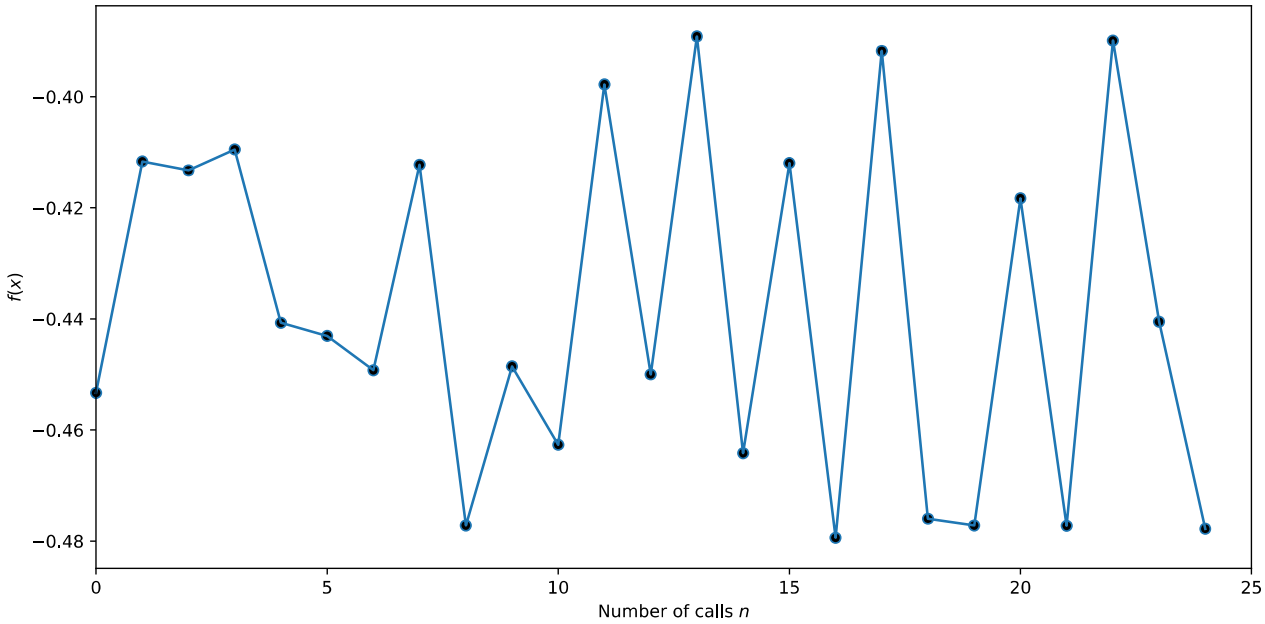
Number of calls is equal to 25. Note that we have cut in-half the number of calls because we saw that with the previous bayesian optimization the converge plot was able to converge in less than 10 calls, so to save computational resources we are reducing the number of calls.
At the initialization before we start modelling we call the objective function with **random hyperparameters for five(5) times**.

Note that the objective function is trying to maximize the mean value of the 10% of the best validation accuracies which here is the top two(2) values.

Results

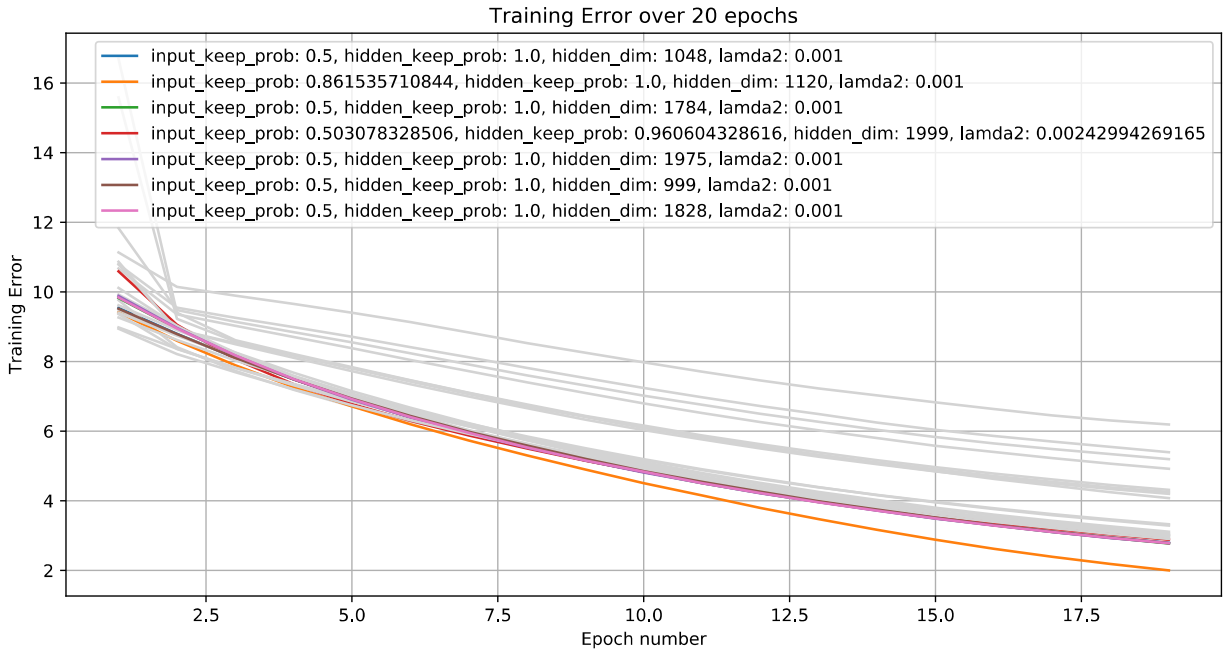


Plot 49: Convergence Plot for Bayesian Optimization trying to find optimal Hidden Dimensionality, Dropout Keep Probability of Input and Dropout Keep Probability of Hidden Layer for Student Model with Learning Rate 1e-4

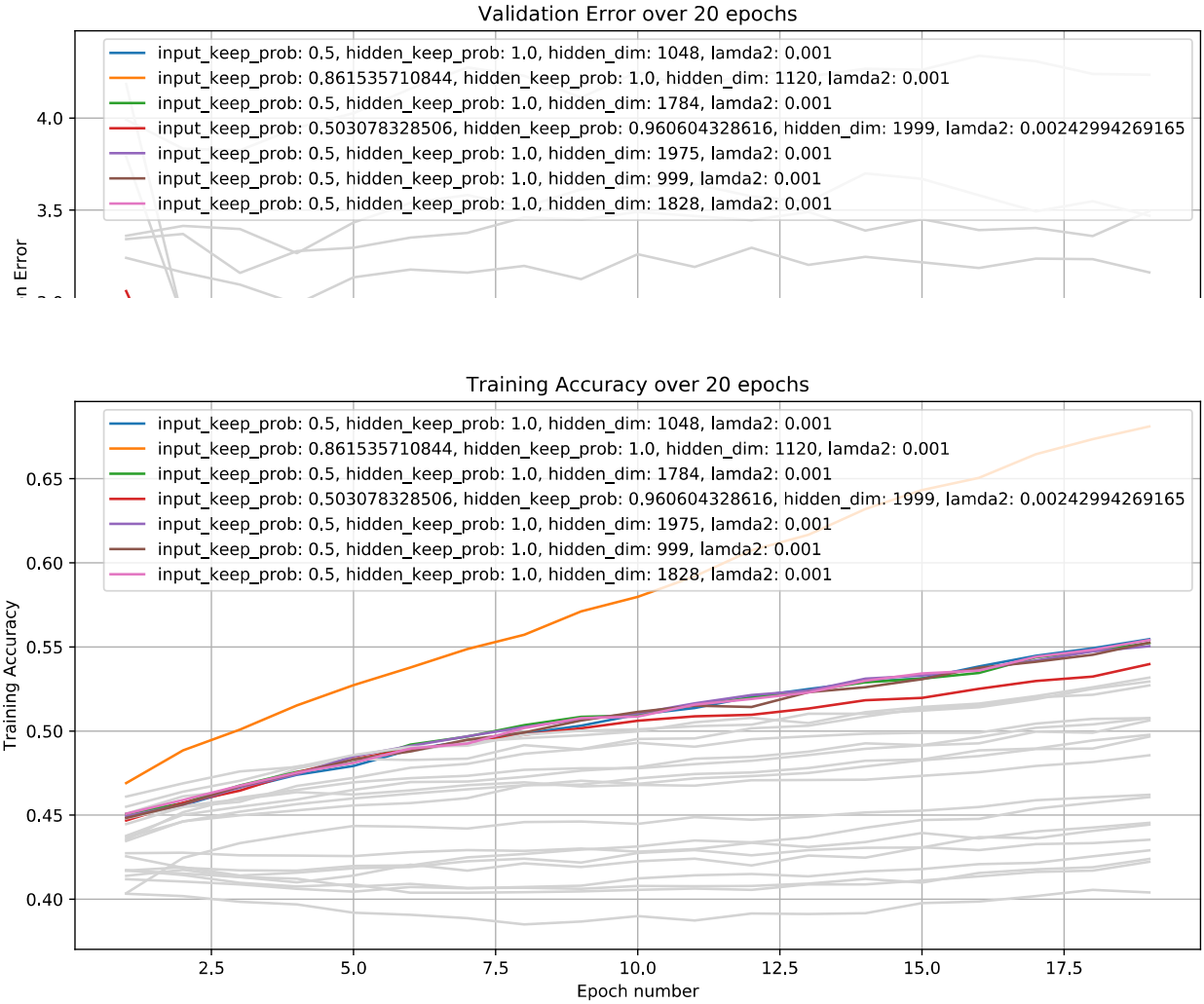


Plot 50: Output of Objective Function for Bayesian Optimization trying to find optimal Hidden Dimensionality, Dropout Keep Probability of Input and Dropout Keep Probability of Hidden Layer for Student Model with Learning Rate 1e-4

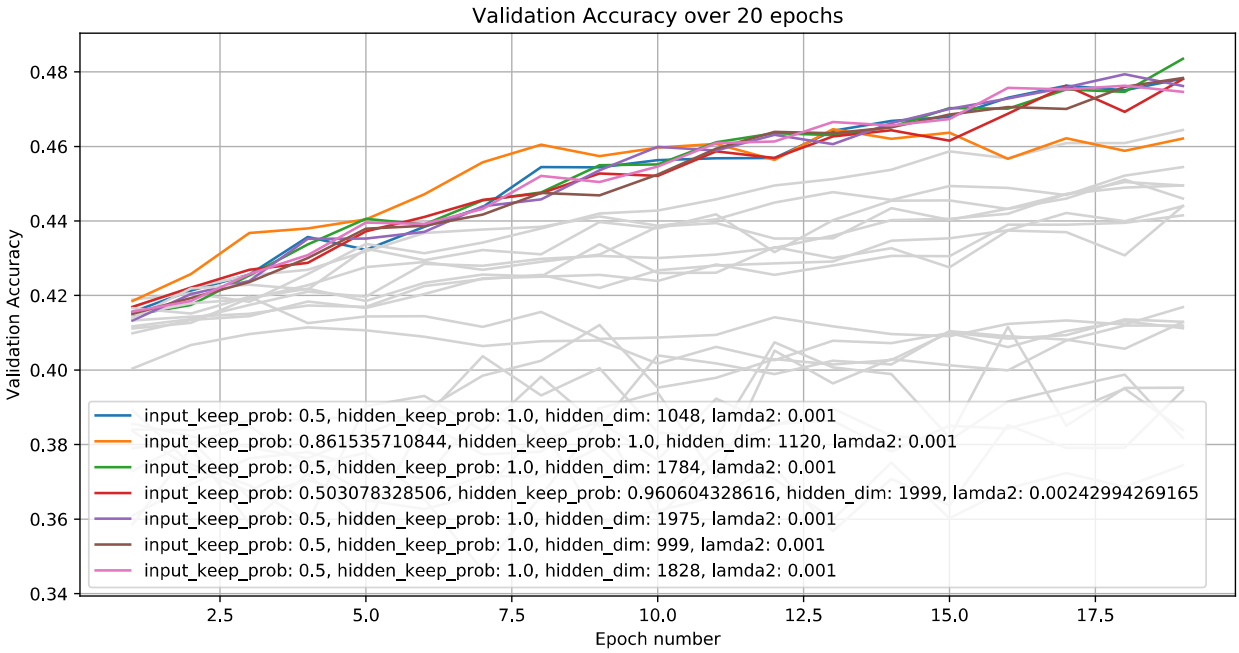
//…



Plot 51: Training Error over 40 epochs per call for Bayesian Optimization trying to find optimal Hidden Dimensionality, Dropout Keep Probability of Input and Dropout Keep Probability of Hidden Layer for Student Model with Learning Rate 1e-4



Plot 53: Training Accuracy over 40 epochs per call for Bayesian Optimization trying to find optimal Hidden Dimensionality, Dropout Keep Probability of Input and Dropout Keep Probability of Hidden Layer for Student Model with Learning Rate 1e-4



Plot 54: Validation Accuracy over 40 epochs per call for Bayesian Optimization trying to find optimal Hidden Dimensionality, Dropout Keep Probability of Input and Dropout Keep Probability of Hidden Layer for Student Model with Learning Rate 1e-4

Best Parameters suggested from this Bayesian Optimization:

- Keep Dropout Probability of Input layer: 50%
- Keep Dropout Probability of Hidden Layer: 100%
- Number of Neurons for the Hidden Layer: 1784
- L2 regularization factor: 0.001

Conclusions

First of all we note that Bayesian Optimization is able to find an overall better model, even if fewer runs and epochs are provided, with the larger learning rate than with the small learning rate.
Top validation accuracy is 48% with learning rate at 1e-4 while it was only 46% with learning rate at 1e-5.

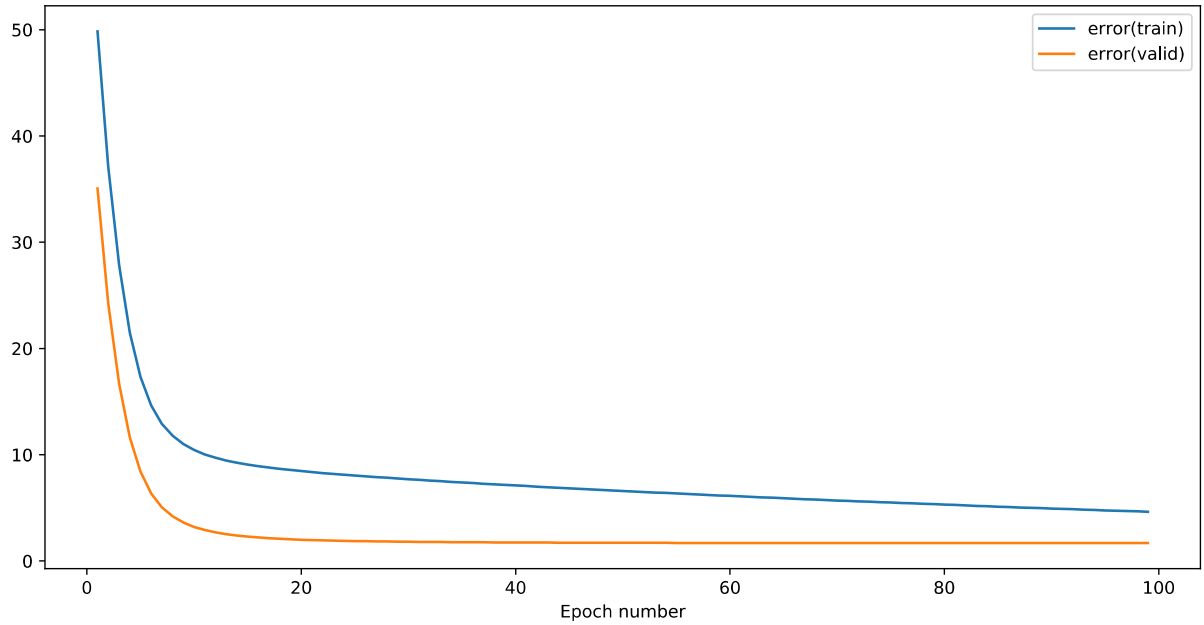
Also note that there is a higher variance for the results of the objective function for the learning rate 1e-4 case. In other words we see the objective function fluctuating between the lowest and the highest values as it tries different hyperparameters.
On the other hand when the learning rate was set to 1e-5 the convergence was really quick, in less than 10 runs the bayesian optimization had found the (locally) optimal parameters.

It is worth mentioning that the learning rate plays two roles. One role is to determine how fast the learning is going to progress but the other role is to act as an exploration parameter in the error space.
In that respect, when the learning rate was set in a small value equal to 1e-5 we had little exploration and the optimization was easily focussed on a locally optimal point in the error space missing the bigger picture.
No wonder then that the optimal dropout probabilities were found to be much smaller when the learning rate was set to 1e-5, in order to compensate for the exploration role that was missing. So in the case where learning rate was set to 1e-4 the dropout probability for the hidden layer was set to 100%.

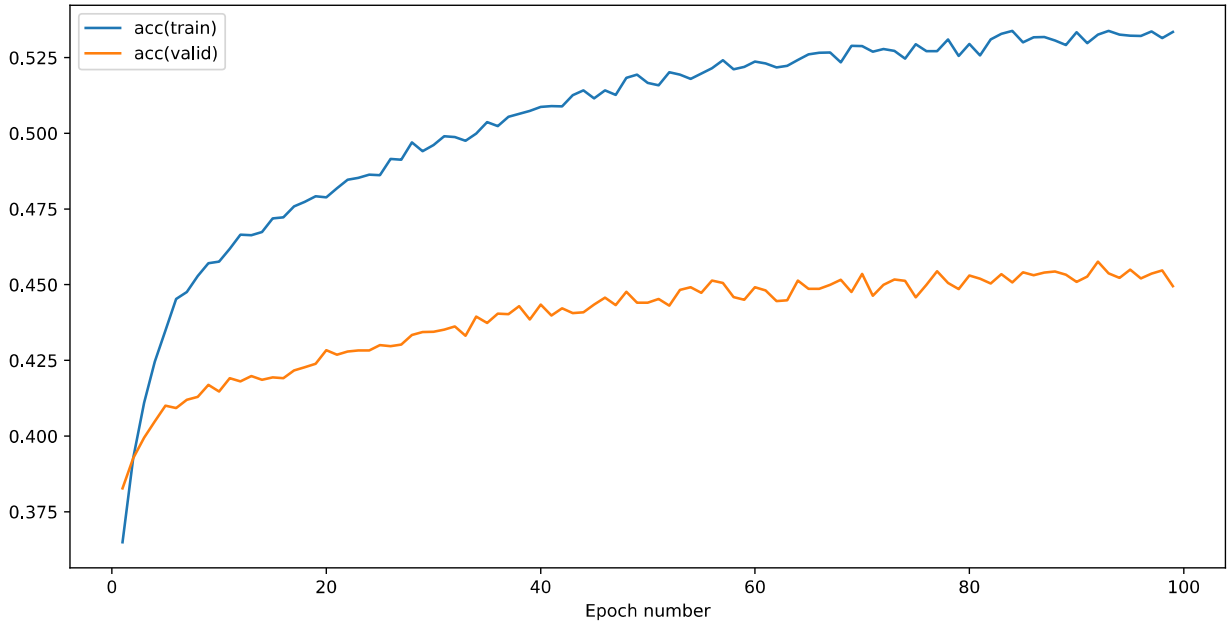
Full training of Student Shallow Neural Network with learning rate 1e-5

L2 regularization factor: 0.062070
Hidden dimensionality: 828
Learning Rate: 1e-5
Epochs: 100
Dropout Keep Probability of Input Layer: 0.605013
Dropout Keep Probability of Hidden Layer: 0.684124

Results



Plot 55: Training & Validation Error of Student Model from Teacher logits – L2 regularization factor: 0.062070 – hidden dimensionality: 828 – learning rate: 1e-5 – epochs: 100 – Dropout Keep Probability of Input Layer: 60.5% – Dropout Keep Probability of Hidden Layer: 68.4%

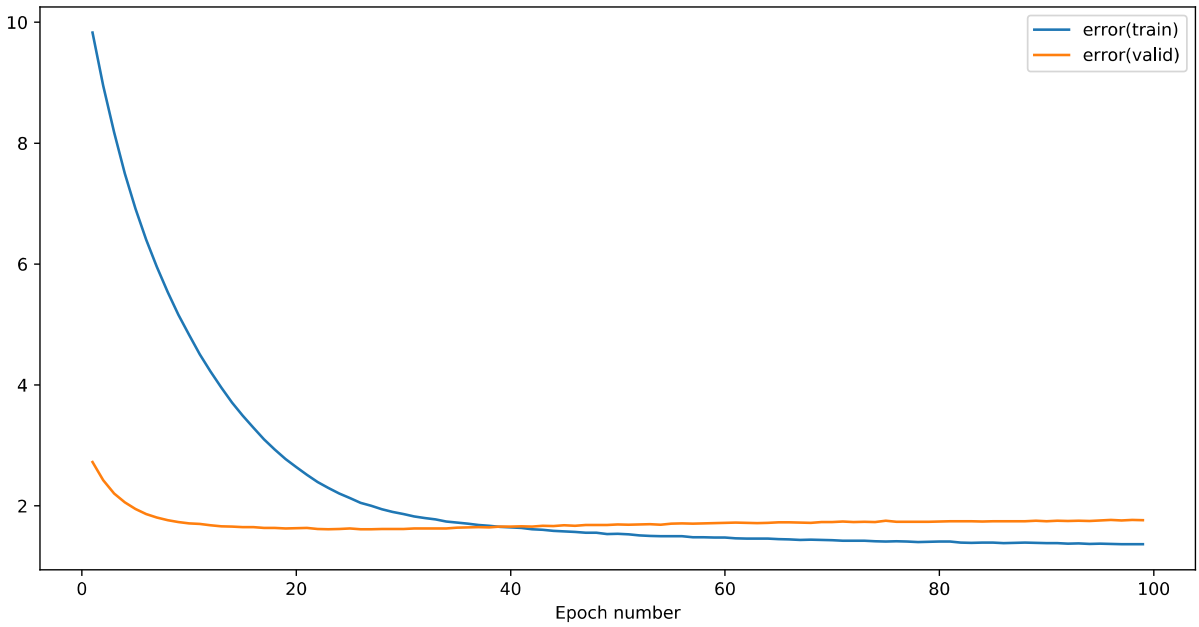


Plot 56: Training & Validation Accuracy of Student Model from Teacher logits – L2 regularization factor: 0.062070 – hidden dimensionality: 828 – Learning Rate: 1e-5 – epochs: 100 – Dropout Keep Probability of Input Layer: 60.5% – Dropout Keep Probability of Hidden Layer: 68.4%

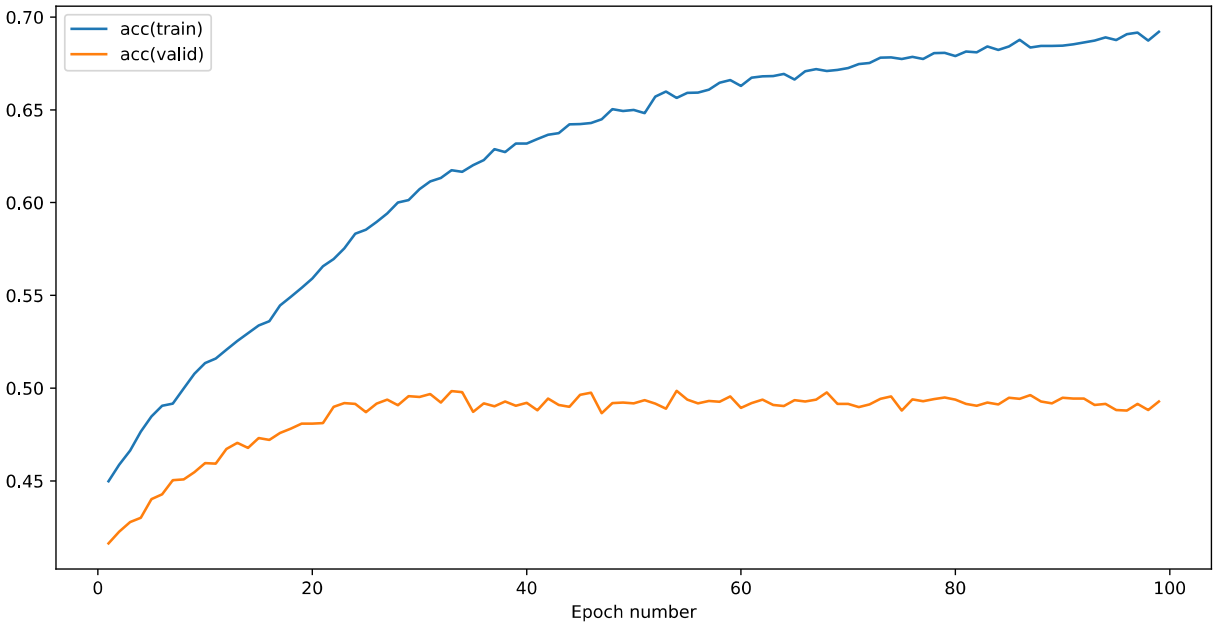
Full training of Student Shallow Neural Network with learning rate 1e-4

L2 regularization factor: 0.001000
Hidden dimensionality: 1784
Learning Rate: 1e-4
Epochs: 100
Dropout Keep Probability of Input Layer: 0.5
Dropout Keep Probability of Hidden Layer: 1.0

Results



Plot 57: Training & Validation Error of Student Model from Teacher logits – L2 regularization factor: 0.001 – hidden dimensionality: 1784 – Learning Rate: 1e-4 – epochs: 100 – Dropout Keep Probability of Input Layer: 50% – Dropout Keep Probability of Hidden Layer: 100%



Plot 58: Training & Validation Accuracy of Student Model from Teacher logits – L2 regularization factor: 0.001 – hidden dimensionality: 1784 – Learning Rate: 1e-4 – epochs: 100 – Dropout Keep Probability of Input Layer: 50% – Dropout Keep Probability of Hidden Layer: 100%

Conclusions

We notice that even for as many as 100 epochs the Student with the small learning rate of 1e-5 is not able to perform a validation accuracy of more than ~45% or barely 46%. This comes as a big or smaller trade-off when comparing the validation accuracy of Teacher model which is able to achieve the ~58% validation accuracy. This obviously depends on the business needs of accuracy versus speed.

The larger learning rate of the 1e-4 and the optimal parameters that come with it according to bayesian optimization give a Student model which takes twice the time than the Student model of the previous paragraph on the one hand but with greater validation accuracy of 50%.

Comparing the Validation Speeds of Teacher versus Student

The train_valid, the concatenation of training and validation datasets, contains in total 50 thousands instances of song data. So for running our validation method on the fully trained model for 20 epochs we will have validated 50.000 x 20 = 1 000 000 or 1 million songs.

Note that for all experiments below you will not be able to reproduce the results because those are dependent on the hardware and the operating system being used. This is not very important because we only care for comparison.

Validating using Teacher

Executing the experiment for our basic Recurrent Neural Network, our Teacher model with best parameters:

- State Size: 341
- Num Steps: 4

Results:

- Mean Time per epoch: 45.941 secs

- Min Time per epoch: 45.458 secs
- Max Time per epoch: 46.672 secs
- Variance of Time per epoch: 0.11426

As provided by the `%%time` special command of jupyter notebook:

```
CPU times: user 20min 31s, sys: 1min 57s, total: 22min 28s
Wall time: 15min 18s
```

Validating using Student

Executing the experiment for our Shallow MLP Neural Network, our Student model with best parameters:

- Learning Rate for Adam Optimizer: 1e-4
- Dropout Keep Probability: 100% (because on validating we always set dropout to 100%)
- Hidden Dimensionality: 1784
- L2 regularization factor: 1e-3

Results:

- Mean Time per epoch: 9.420 secs
- Min Time per epoch: 9.304 secs
- Max Time per epoch: 10.015 secs
- Variance of Time per epoch: 0.0256

As provided by the `%%time` special command of jupyter notebook:

```
CPU times: user 1min 28s, sys: 10.4 s, total: 1min 38s
Wall time: 3min 11s
```

Conclusions

The conclusions from the results of the two experiments above show that we have achieved a ~**5x** improvement in speed as all metrics suggest.
This could should be taken into account when building a working systems that would need to classify potentially millions of songs.

Further work for Student-Teacher model (not implemented)

Further work from this, it could be to have a system where both the Teacher and the Student model are being used. A brief description of such an implementation would include passing the song data through the Student and if the cross entropy was bigger than a certain threshold, which we would had set as a hyperparameter, then the Teacher model would be called to handle this classification. Of course at the end we would pick the one of the two with the smaller cross entropy.
Setting this threshold hyperparameter at an appropriate level would give a continuous trade-off between speed and classification accuracy.
We will not implement this system on coursework4.