MLP Coursework 3: Feed-forward Baselines for The Million Song Dataset

James Wilsenach - s1666320

February 16, 2017

1 Methodological & Dataset Overview

The aim of this report is to determine baseline performance of simple feed-forward networks, on the problem of genre classification in the Million Song Dataset (MSD). These baseline performance levels are to be used for comparison later when assessing the classification performance of more complex models that incorporate recurrent and convolutional layers.

1.1 The Million Song Dataset

MSD is split up into two datasets, MSD10 which includes ten target genre classes and MSD25, which includes 25 classes. The datasets consists of a collection of songs each with a rich array of features describing it [1]. The features we use will correspond roughly with a time-series describing the temporo-audial or note-by-note properties of the song organised into segments or quasi-stable music event. There are a total of 120 such segments per song and each segment has 25 normalised features associated with it which encodes various properties of the event including timbre, pitch and volume. The datasets themselves have been partitioned into training, validation and test sets (but we will not be using the test set in this assignment).

1.2 Experimental Overview

Investigations into different architectures and parametrizations were carried out in order to optimize classification performance (largely measured by validation accuracy), while limiting computational costs (measured by computing time). Models were constructed with fully connected, constant width hidden layers. These were trained using the momentum-based Adam learning rule, which has the advantage of being both robust against both shallow minima and instability due to large step size.

We started by attempting to find an initially optimal learning rate parameter (LR) to serve as the basis for further investigations into both the 25 and ten genre classification problems. In Sec. 3 a number of basic network architectures were assessed using brief training regimes (few training epochs) with varying numbers of layers and hidden units to determine a reasonable efficient architecture. A comparison of L1 & L2 regularization was performed 4. Lastly in Sec. 5, two simple data augmentation procedures with added white noise and dropout were implemented with regularization and the learning rate was reassessed under this framework. In Sec. 6 the performances of these simple baseline models is briefly discussed. In the final section, Sec. 7, we propose future experiments to be performed using recurrent architectures and multi-task learning.

1.3 Implementation with TensorFlow

TensorFlow is an open source machine learning library written in Python and C++. In TensorFlow machine learning models are constructed in Python by specifying computational graphs of operations performed on matrix-like arrays called tensors. Inference and prediction are carried out by executing operations on these tensors.

We make use of the basic graph constructors provided as well as the TensorFlow *contrib* directory for creating multi-layered networks. We use a number of helper classes for handling of model construction (*Model* and its associated subclasses), providing augmented data for training (see *providers*) and plotting (see *MultiPlot*). These classes are located in two files *networks.py* and *providers.py*. The *Model* classes are distinguished by their hyperparameters and learning methods. *RegModel* and *NoisyRegModel* both define multilayer networks with regularization (modifying the error operation in *Model*) but only *NoisyRegModel* and *DropOutRegModel* define augmented data providers in the *providers* library. There are also simple classes for saving and loading relevant performance statistics of *Model* objects.

2 Initial Learning Rate

Initial investigation of the LR hyperparameter was carried out by varying LR through a log scale from 1e-5 to 1e-3 in a number of short training sessions of 10 epochs each on a network with 200 hidden units and two hidden layers.

2.1 Learning Rate for MSD10

The training curves (error and accuracy by epoch) for the best performing model on MSD10 used an LR of 1e-3 and is shown in Fig. 1. Peak performance accuracy (46%) and error occur very early on in training. This suggests that choosing LR=1e-3 should scale well for larger problems and deeper architectures, despite the fact that all three rates tested took roughly the same time per epoch to train. This makes 1e-3 a suitable starting choice for classification on MSD10.

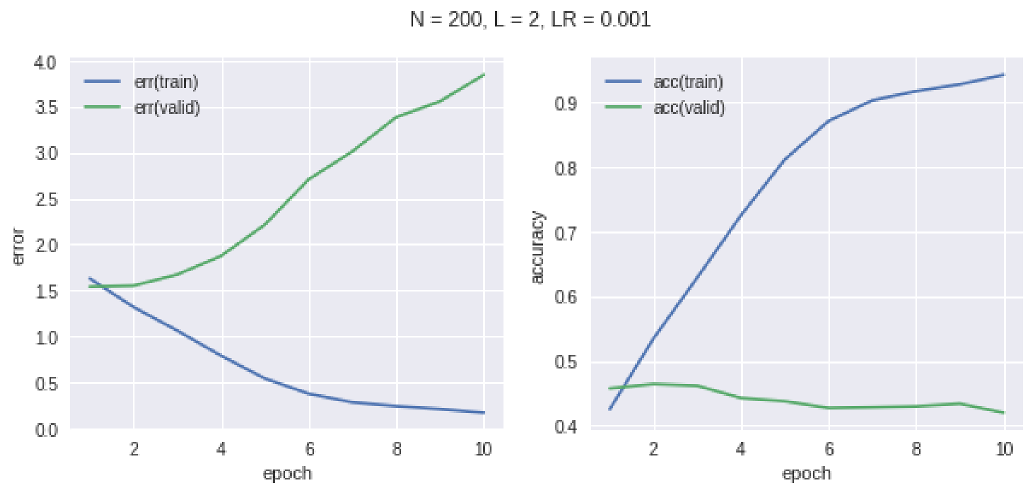


Figure 1: Training (blue) and validation (green) error (left) and accuracy (right) curves for a network with learning rate (LR) of 1e-3 on MSD10. The text above the figure shows N, the number of hidden units, L, the number of layers and LR.

2.2 Learning Rate for MSD25

Learning rate investigations for MSD25 show that over 10 epochs, best performance amongst the hyperparameters tested (20% accuracy) is reached quickly for LR=1e-3. Fast training could also indicate future instability for more complicated models so we chose R=1e-3 for MSD25 as well, but reassess it later under more complicated learning regimes. Again, similar per epoch training times as MSD10 (roughly eight seconds) is seen across the different training sessions.

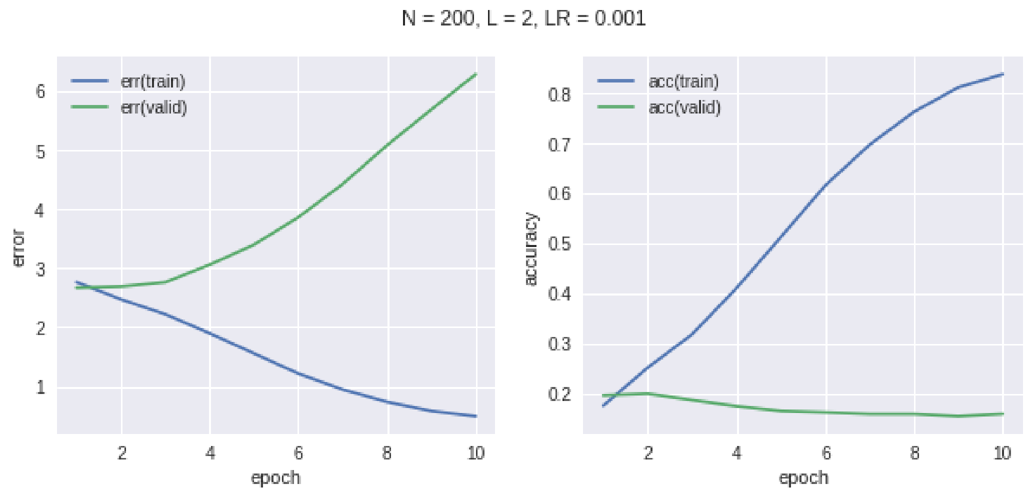
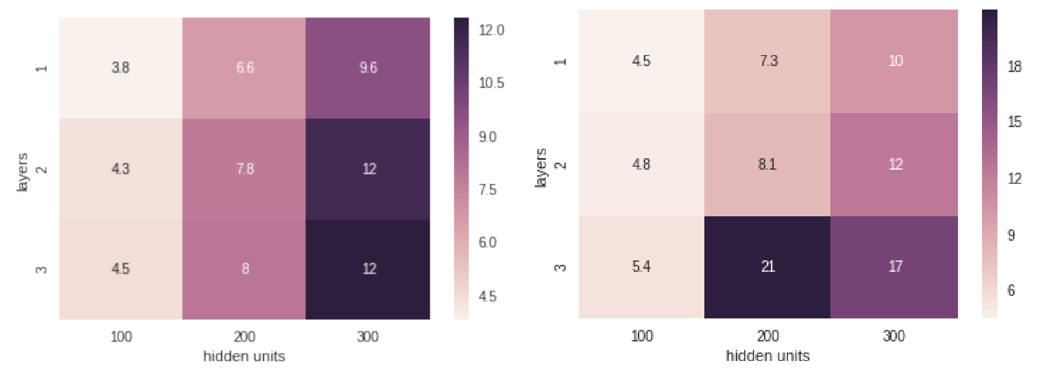


Figure 2: Training (blue) and validation (green) error (left) and accuracy (right) curves for a network with learning rate (LR) of 1e-3 on MSD25. The text above the figure shows N, the number of hidden units, L, the number of layers and LR.

3 Layers & Hidden Units

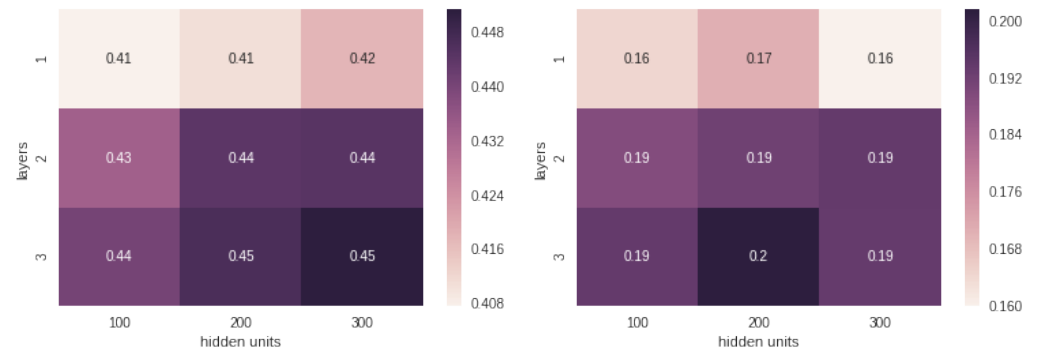
As networks with lower numbers of layers (L) and hidden units (HU) are in a sense, nested inside of broader, deeper networks, we would expect that these simpler models perform worse than larger models in general but should be less computationally expensive to train. Determining the most appropriate architecture therefore involves trading computational efficiency (computation scales with increasing numbers of gradient calculations) for improved accuracy.

From the per epoch computation time heat maps in Fig. 3(a) and Fig. 3(b) note that number of hidden units per layer are the primary determinant of computing time whilst the number of layers effects training efficiency to a lesser extent. From the peak accuracy heat maps in Fig. 4(a) and Fig. 4(b) it is clear that there are broad improvements in classification performance from one to two layers, whilst the difference between 2 and 3 is less marked. We therefore chose to keep the current network architecture with 200 hidden units and two hidden layers due to diminishing returns on layer increase and increased computational expense with increasing hidden units.



(a) (b)

Figure 3: Heat maps of average computing time per epoch for (a.) MSD10 and (b.) MSD25 for different numbers of layers and hidden units



(a) (b)

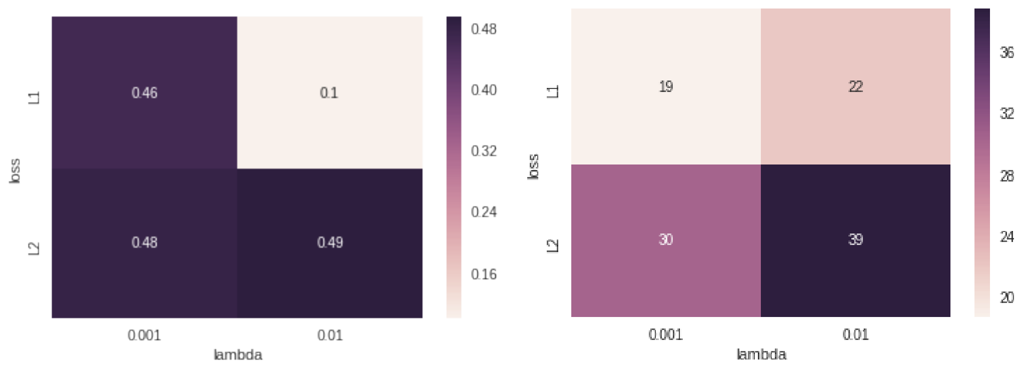
Figure 4: Heat maps of average computing time per epoch for (a.) MSD10 and (b.) MSD25 for different numbers of layers and hidden units

4 Regularization with L1 & L2 Loss

L1 and L2 loss are both used as regularising error terms that penalize large model parameters (network weights) during training to prevent over fitting. They differ in that L1 is generally used when a sparse encoding of the data is relevant since it favours large numbers of near zero parameters. Ensuring sparsity using L1 may be useful if the song data itself contains multiple low complexity events (such as beats between notes or rests) whose frequency characterises the genre, otherwise if this is not the case L2 regularization may be more applicable. Since we can suppose reasons why the solution may benefit from sparsity we chose to investigate both the L1 and L2 loss functions for baseline performance. We tested both L1 and L2 loss using two different values for $\lambda \in \{1\text{e-}3, 1\text{e-}2\}$, the regularization co-efficient.

4.1 Regularization on MSD10

In MSD10 the method with the best overall performance is L2 regularization (see Fig. 6), with λ=1e-2 (49% accuracy). This is in comparison to the best L1 performance (46% accuracy) which was reached, slightly earlier after 10 epochs (in comparison to 12 for L2). We made this choice despite the fact that L1 was computed faster, which could be due to internal system noise (see Fig. 5(b)). L1 regularization also seems to be less forgiving of large λ values since for λ =1e-2 (top left in Fig. 5(a)) learning has been so restricted that the model does not deviate from random chance.



(a) (b)

Figure 5: Heat maps of (a.) peak accuracy and (b.) average computing time per epoch for MSD10

4.2 Regularization on MSD25

The best performance on MSD25 is again given by L2 regularization (see Fig. 6), with λ=1e-2 (22% accuracy - see Fig. 8(a)), as in Sec. 4.1. The best L1 performance (20% accuracy) is now much more efficient than the fastest L2 method, with a difference of more than 30 seconds per epoch (see Fig. 8(b)). The scaling of these differences from MSD10 to MSD25 seems to suggest that L1 regularization could become the preferred choice for larger architectures where efficiency is a major concern, as long as λ remains small.

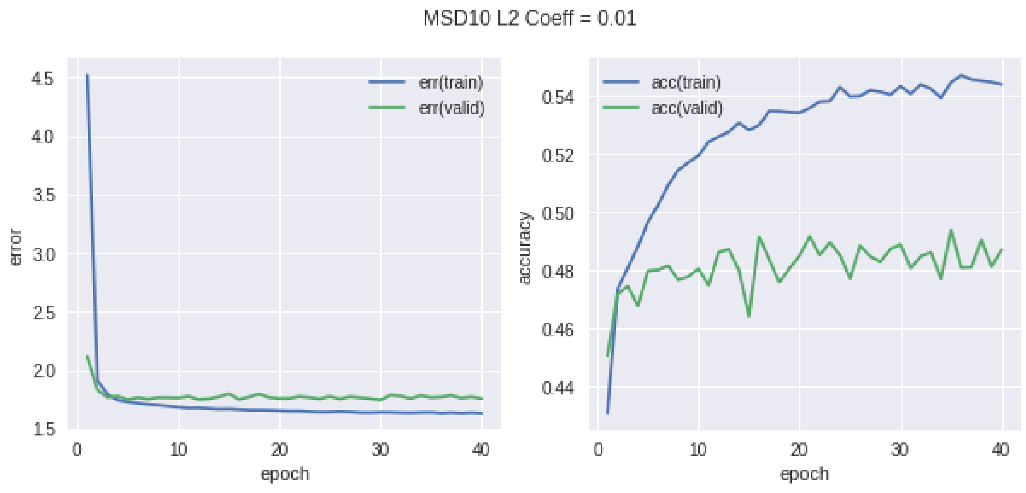


Figure 6: Training (blue) and validation (green) error (left) and accuracy (right) curves for a network with learning rate (LR) of 1e-3 and L2 regularization ($\lambda = 0.01$) on MSD10. The text above the figure value of λ.

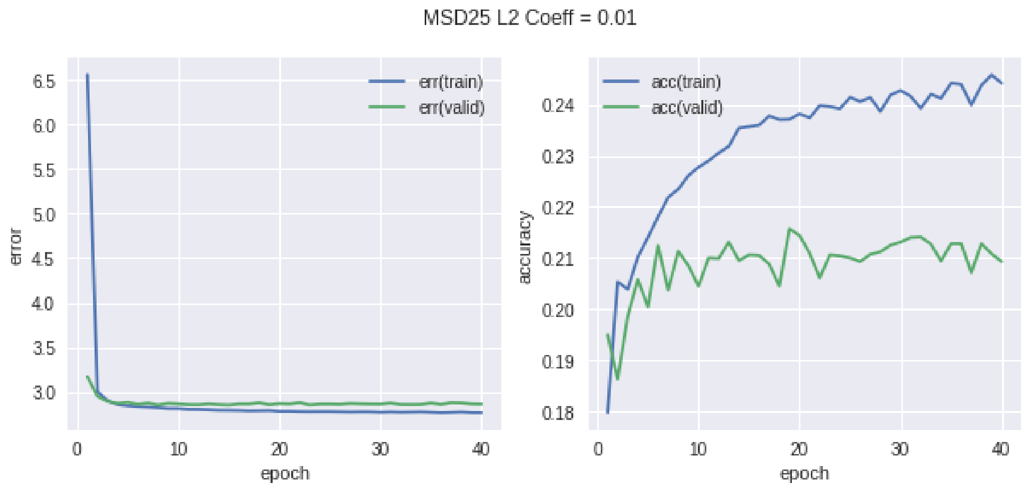
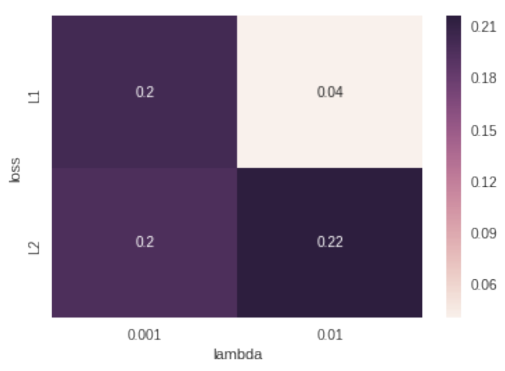


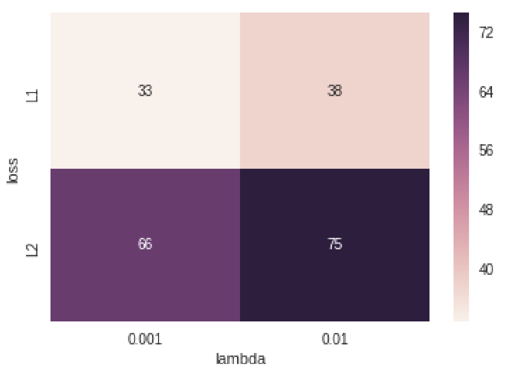
Figure 7: Training (blue) and validation (green) error (left) and accuracy (right) curves for a network with learning rate (LR) of 1e-3 and L2 regularization ($\lambda = 0.01$) on MSD25. The text above the figure value of λ.

5 Learning Rates for Augmentation

Data augmentation is the process of creating new training data from old using a form of distorting transformation on the original input features. Data augmentation can be useful in improving performance by allowing for more



(a)



(b)

Figure 8: Heat maps of (a.) peak accuracy and (b.) average computing time per epoch for L1 and L2 regularization on MSD25.

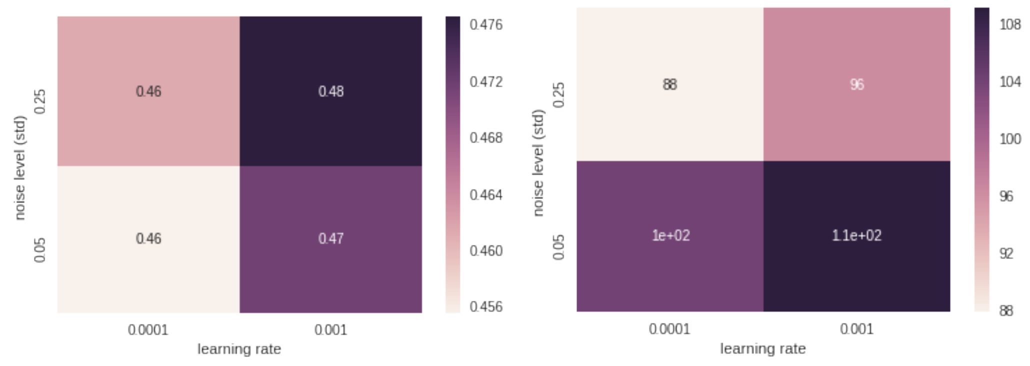
training to occur (on the augmented data on top of the old) and by forcing the network to learn representations that are robust to these distortions.

One possibility is the addition of Gaussian white noise to a subset of song data that forces the network to look for robust properties that tolerate a level of noise. We chose to generate noisy input data from be 20% of the original while varying the standard deviation of the noise as well as re-assessing the learning rate to see if it is still a near optimal parameter value.

Another data augmentation method which we briefly explored was dropout, which works by setting a random selection of input features to zero. Note that this will not cause skewing as input data is already normalized. We compared a simple dropout augmented model, with 20% of features randomly set to zero, against the best performing Gaussian noise augmentation model.

5.1 MSD10 Noisy Data Augmentation

No significant performance improvements over non-augmented data were noticed in either accuracy or error (47% validation accuracy and error $1.86 > 1.74$ for plain regularization, see Fig. 9(a) and Fig. 10). This could be due to the model training on features that are robust but less characteristic of a genre. There was also a marked increase in computation time (over 100 seconds per epoch) due to the additional inputs, or potentially due to extended run times with Tensor Flow (see Fig. 9(b)). Furthermore, no evidence of significance performance improvement after changing LR was observed in any of the cases.



(a) (b)

Figure 9: Heat maps of (a.) peak accuracy and (b.) average computing time per epoch for noise augmented MSD10 with L2 regularization and noise added data augmentation.

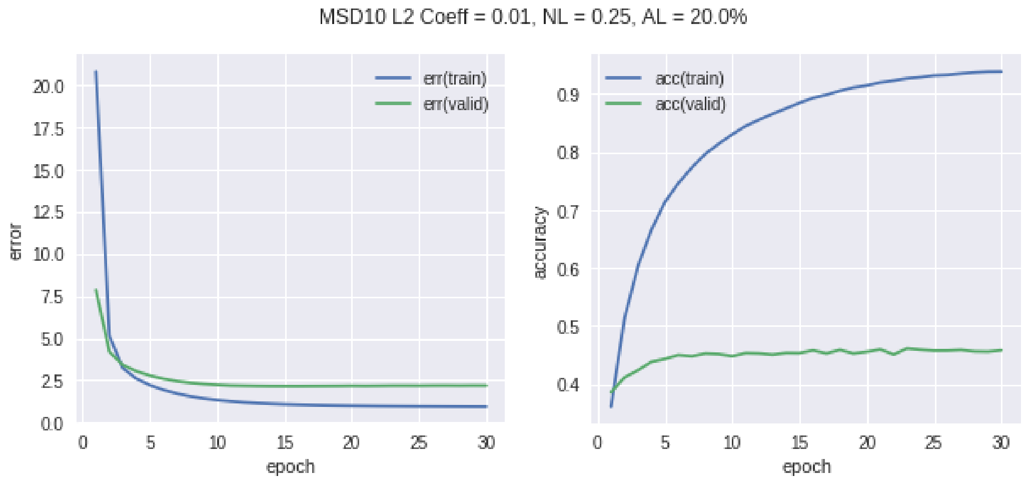
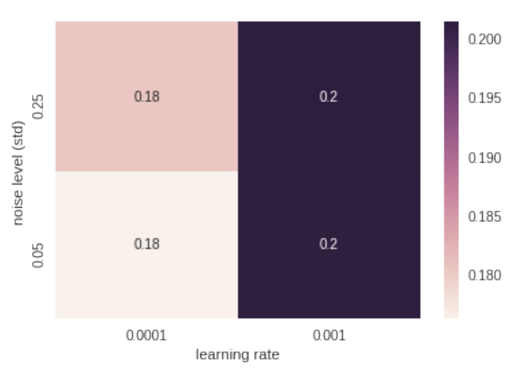


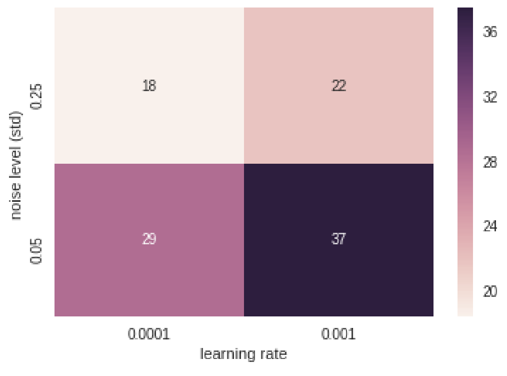
Figure 10: Training (blue) and validation (green) error (left) and accuracy (right) curves for a network with learning rate (LR) of 1e-3 and L2 regularization ($\lambda = 0.01$) on MSD10 with 20% noisy augmented data (AL=20%) with standard deviation (NL in the heading) of 0.25.

5.2 MSD25 Noisy Data Augmentation

Again, no significant performance improvement (20% validation accuracy) was noted for the hyper parameters tested (see Figure 11(a)). The per epoch time statistics also suggest fluctuations in TensorFlow's computational efficiency that have been seen in other cases.



(a)



(b)

Figure 11: Heat maps of (a.) peak accuracy and (b.) average computing time per epoch for noise augmented MSD25 with L2 regularization and noise added data augmentation.

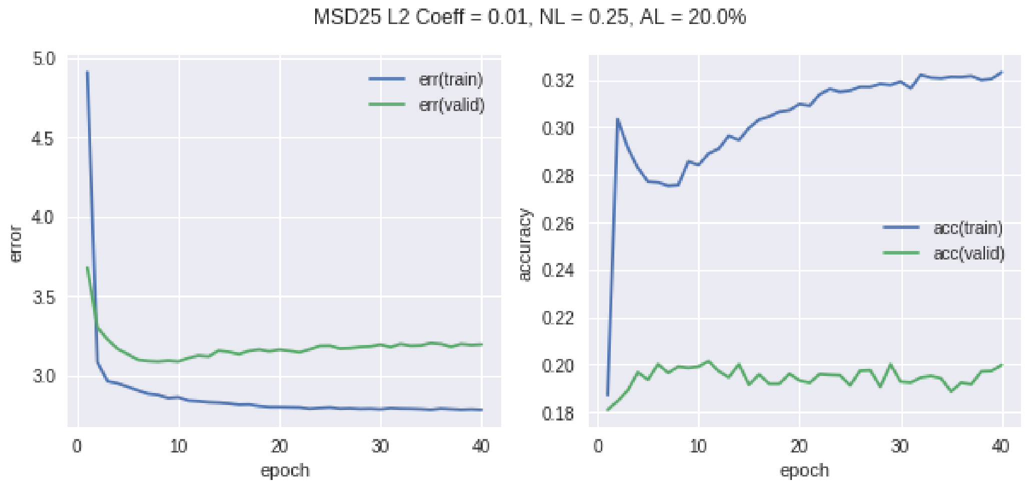


Figure 12: Training (blue) and validation (green) error (left) and accuracy (right) curves for a network with learning rate (LR) of 1e-3 and L2 regularization ($\lambda = 0.01$) on MSD25 with 20% noisy augmented data (AL=20%) with standard deviation (NL in the heading) of 0.25.

5.3 Comparison to Dropout Augmentation

Dropout augmentation performance for the same regularization and learning rate parameters appears almost identical to the noisy case with slightly improved performance on both data sets (see Fig. 13, Fig. 14 and Tab. 1).

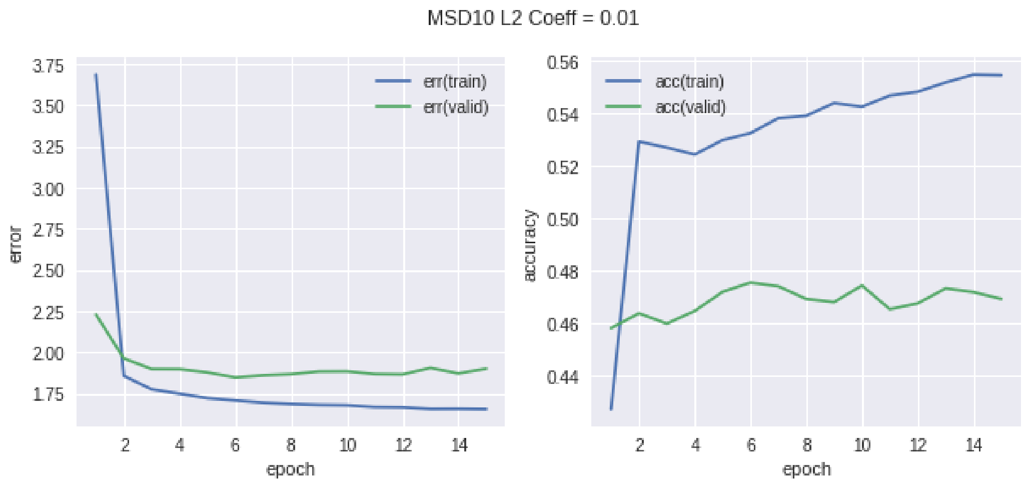


Figure 13: Training (blue) and validation (green) error (left) and accuracy (right) curves for a network with learning rate (LR) of 1e-3 and L2 regularization ($\lambda = 0.01$) on MSD10 with 20% dropout augmented data (AL=20%) with dropout probability 0.1.

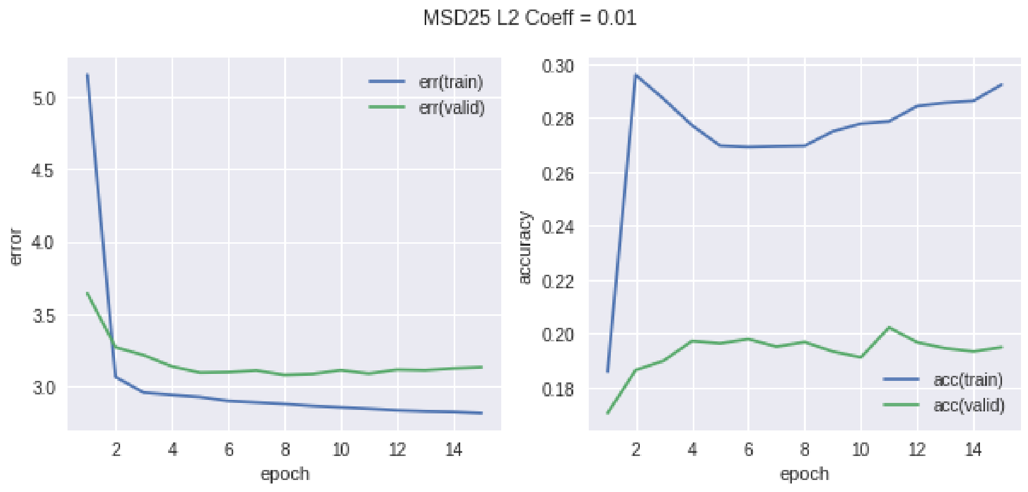


Figure 14: Training (blue) and validation (green) error (left) and accuracy (right) curves for a network with learning rate (LR) of 1e-3 and L2 regularization ($\lambda = 0.01$) on MSD25 with 20% dropout augmented data (AL=20%) with dropout probability 0.1.

dataset/methods	val. accuracy	val. error	sec/epoch
MSD10	44%	1.53	7.8
MSD25	19%	2.69	8.1
MSD10 L1	46%	2.04	19
MSD25 L1	20%	3.50	33
MSD10 L2	49%	1.74	39
MSD25 L2	22%	2.85	75
MSD10 L2 Aug. Noise	48%	1.86	96
MSD25 L2 Aug. Noise	20%	3.09	22
MSD10 L2 Aug. Dropout	48%	1.85	48
MSD25 L2 Aug. Dropout	20%	3.08	58

Table 1: Lists basic performance statistics for the best performing methods on the validation set (so as to save the test set for further work - see Sec. 7) for each of the method and dataset/task listed.

6 Discussion of Results

Tab. 1 shows baseline best validation and training time performance statistics for all the major model types assessed. No prediction has yet been performed on the test set as this should be saved for final model formulations in Coursework 4.

Regularization, and in particular L2 regularization, produced the most significant improvement over the baseline for both MSD10 and MSD25 (see Tab. 1). The performance of L1 over L2 regularization indicates that sparse methods may not be as applicable. L2 regularization did appear to in general be less time efficient than L1 regularization, however, times may not always be reliable as TennsorFlow computational efficiency appears to deteriorate with run time. The LR that was most effective overall was 1e-3 which outperformed other LR parametrizations for both the simple case and augmentation with regularization example.

Neither noise or dropout augmentation seemed to yield significant performance improvements for either MSD10 or MSD25, although higher noise levels were generally associated with better performance. This could have been due to the small hyper parameter search space or features that are sensitive to noise. Better augmentation procedures would need to be developed (potentially using a broader hyper parameter search space or auto-encoders). Further work could also be done using dimensionality reduction methods such as PCA and more reliable ways of measuring computational time of training would be most useful.

7 Further Work: Recurrence & Multi-tasking

Multi-task learning is a methodology for improving learning performance where a related set of tasks are simultaneously optimised using the same model. This is thought to improve the optimisation performance on the task of interest by forcing the network to solve a meta-problem related to this master task [2]. Functionally this means that in multi-task learning the model is trained using not only the target classification but also makes use of an additional set of auxiliary or secondary target features.

Multi-task learning could be used in the MSD classification problem in several different ways. A simple first option which has been applied successfully in facial recognition tasks [3], is to use auto-encoding as a secondary task. In MSD genre classification, performing an auxiliary auto-encoding would force the network to learn a representation of songs in general which could be useful in labelling each song. Although, discarding some features which may not be as relevant, such as loudness, in the auto-encoding could help to smooth genre learning and help minimize network confusion.

Other secondary tasks could be performed with some pre-processing these include classifying by super-genre and by auto-correlation function (averaged across segments). Super-genre classification would require manually creating appropriate super-genres that incorporated multiple other genres in MSD25 (MSD10's target class set may be too small for aggregation into super-genres). A possible problem is that these super-genres would most likely be of different sizes and so augmentation of under-represented genres in smaller super-genres would need to be performed to remove the bias. A possible draw-back is that this would in-turn create biases in the genre classification task so it may be best used as a form of pre-training rather than secondary task. The auto-correlation function also remains an option since it encodes valuable beat and frequency information that could be common within a genre.

The full MSD contains much more song metadata which could be used in multi-task learning including the artist which is sure to be closely related to the genre, If using this additional data is permitted, it presents even more multi-task opportunities.

The sequential nature of the songs stored in MSD best lend itself to a recurrent network architecture that can remember temporal features of the data. Coupling recurrent layers with 2D convolutional layers with max pooling could further allow the network to learn relationships between transitional boundaries that exist in music such as tempo, key and volume transitions.

References

[1] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *ISMIR*, volume 2, page 10, 2011.

[2] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. *Advances in neural information processing systems*, 19:41, 2007.

[3] Terrance Devries, Kumar Biswaranjan, and Graham W Taylor. Multi-task learning of facial landmarks and expression. In *Computer and Robot Vision (CRV), 2014 Canadian Conference on*, pages 98–103. IEEE, 2014.