MLP Coursework 4: RNNs on The Million Song Dataset: Training with Multiple Tasks

James Wilsenach - s1666320

March 21, 2017

Contents

1 Methodological & Dataset Overview

The primary aim of this report is to analyse and compare the performance of neural network models which incorporate multiple tasks, to improve genre classification performance on the Million Song Dataset (MSD). The methodologies explored can broadly be separated into two categories, ensemble and multitask learning and are loosely based on an ensemble architecture proposed by Silla and Kaestner [1]. Multitask and ensemble learning are two very separate methods for improving model performance which both involve training models on the same or similar tasks which all contribute to solving the overall task of interest.

All of the models investigated employ a similar Recurrent Neural Network (RNN) architecture and so preliminary investigation of models with a basic single hidden recurrent layer and linear output layer was carried out. All models were trained and then validated on the same separate datasets of songs with a fixed length. The final model chosen from amongst the multitask and ensemble models was then tested on a held-back test set to assess its generalizability.

In this section we discuss the basic components of MSD and explain the role of RNNs in music analysis. In Sec. 2 further justification and explanation of the experimental approach to ensemble and multitask architectures is given. This is followed by baseline results for simple LSTM models in Sec. 3 which includes a short analysis of different regularization methods (L2 and activation stabilization) for RNNs. Results for ensemble and multitask architectures are included in Sec. 4 and 5. In Sec. 6 we discuss and contrast the models tested with qualitative comparison to the results obtained by Silla and Kaestner for their ensemble model and with our own hypotheses regarding model performance.

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 [2]. 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 in the fixed length datasets and each segment has 25 normalised features associated with it which encodes various properties of the event in-

cluding timbre, pitch and volume.

1.2 RNN Description and Justification

RNNs are a class of networks that can learn patterns in sequentially correlated data, such as the melody or rhythm of a song, by progressive unrolling of the sequence into the network through time. RNNs learn these sequential intra-dependencies by sharing information within a layer through within layer connections. These intra-connections let information persist through multiple passes of the network as internal loops allow the information to circulate and not to be lost in the next training step. In this way RNNs are said to have a memory that persists through time and thus can recall connections between sequence segments.

However, basic RNNs can be limited in their ability to propagate information and thus detect longer-term dependencies as signal strength tends to decay with time. Long Short Term Memory (LSTMs) are a special kind of RNN that were designed to address this shortcoming [3]. They are composed of cells which maintain a cell state that is modified and then passed on as input to the next iteration. This chain of passing-on and adding of information is controlled by gates, most notably the forget gate which modulates how much of the previous cell state is passed on with a bias of one to retain everything and zero nothing. There is also an input gate which controls how the cell state is updated.

Music often contains repeated musical themes that reoccur throughout a songs run-time. Repeated elements give songs structure and most importantly these themes are what make one style distinct from another. Identifying such long-term pattens could thus prove incredibly useful in genre classification tasks as themes such as rhythm and chord progression are often shared within a genre. The models we employed thus include an LSTM layer as well as a single linear output layer.

1.3 Justification of General Model Components

The choice of a network with a simple single LSTM hidden layer was chosen for easy interpretation of results and to limit computational overhead. We used Adam, a momentum-based learning rule for parameter optimization, which is efficient and tolerates larger learning rates, making convergence faster and smoother. Adam combines the abilities of RMSProp, to deal with non-stationary objectives, with that of AdaGrad, to deal with sparse

gradients. Based on previous investigations in Assignment 3 on the effects of learning rate with Adam, we fixed a learning rate of $1e-3$ as a compromise between performance and efficiency [4].

2 Experimental Overview

Here we provide a more detailed justification and explanation of our model choice. Specifically we describe the formulation of these models with particular emphasis on the ensemble and multitask methods with reference to the study on which they are based. Finally we discuss the model implementation in Tensorflow and give a brief outline of the model classes as implemented in Python.

2.1 Assessment of LSTM Baseline: Regularization

Preliminary investigation of LSTMs was carried out to attain baseline results for later comparison. In order to obtain a suitable baseline we compared the performance of two regularization methods, simple L2 regularization and activation stabilization (AS), a regularization procedure derived specifically for RNNs by Krueger and Memisevic [5]. Activation stabilization uses an error term based on the sum of the differences in magnitude between successive activations $((||h_t|| - ||h_{t-1}||)^2)$ averaged over the number of segments. This encourages orthogonality (independence) of the internal RNN transition matrix.

Investigative Hypothesis

Apart from determining suitable baselines we simultaneously wanted to assess claims by some researchers that L2 regularization is unsuitable for RNN cell weights by comparison with other methods due to the damping of weights carrying long-term dependencies [6].

2.2 Ensemble Learning with Song Snippets

However, our primary goal was to investigate multitask and ensemble learning and the models investigated in this report have been broadly inspired by work done on music genre classification by Silla and Kaestner [1] which explores an ensemble learning approach. Ensemble learning is a learning strategy which combines predictions from multiple, independently trained models (experts). It has been shown to increase overall accuracy by 'averaging out' the error in individual predictions and by improving expressibility of

the approximate solution using a combined "wisdom of the crowds" approach.

Silla and Kaestner obtained ensemble classification models using simple MLPs which outperformed their competitors on genre classification tasks in a latin music dataset. Their method is based on dividing the input in time to produce multiple song snippets of equal length. A separate classification model is then learnt for each snippet. In doing so, it is believed that each model will learn something distinct about the natural progression of a song in a given genre. The per snippet classifications are then combined in one of several ways to produce the final genre prediction. They propose four possible prediction aggregation methods but we will investigate only two of these, maximum mean and maximum product prediction.

In the maximum mean case (m_0) the genre with the highest mean posterior probability is chosen, by averaging across models for all song snippets. The maximum product predictor (m_1) works similarly, by predicting the genre with the highest probability mass after element-wise multiplication of the genre probabilities across the snippets. Given a set of M predictors these methods can be summarized as

$$m_0(X) = \arg\max_x \sum_{m=1}^{M} p_m(x|W_m) \qquad m_1(X) = \arg\max_x \prod_{m=1}^{M} p_m(x|W_m)$$

where X is a random variable representing the possible genre, x is a possible genre and p_m is the posterior probability of the genre given the model for snippet m and W_m are the weights associated with the model. We also explored a third variant (m_2) involving weighted sums of individual predictions, maxw.

$$m_2(X) = \arg\max_x \sum_{m=1}^{M} \theta_m p_m(x|W_m)$$

where θ_m is a prediction weight, which was trained simultaneously but independently of the individual per snippet predictors. These θ_m were trained using a simple symmetric network (same number of hidden and input units) with linear hidden activations.

Investigative Hypothesis

Of the first two aggregation methods Silla and Kaestner found m_1 to produce the best results for simple feed-forward networks. In our discussion we consider whether this is still the case for LSTM based models trained on MSD.

2.3 Multitask Learning: Autoencoding & Binary Classification

Multitask learning is a learning framework which utilizes a shared representation across a number of related tasks to learn novel features about the primary task of interest. In our context this task was MSD genre classification and the shared representation was the hidden LSTM layer shared by the primary and secondary tasks.

All tasks contribute to the overall error and thus allow the model to learn. The error of the secondary tasks can be thought of a sort of adaptive regularization that shapes the error-parameter space. One question we consider is how this secondary task error should be weighted using an error coefficient $\alpha > 0$ hyperparameter . We also investigate two types of secondary tasks, classification (supervised) and auto-encoding (unsupervised) tasks to see whether the secondary task is most effective in guiding learning when it is in the same broad class (supervised classification) as the primary genre classification task.

In Silla and Kaestner's original ensemble method tasks were actually split up further into individual binary classification tasks to predict whether a snippet is a member of a genre or not. In this way an individual snippet could be (erroneously) placed into multiple genres with the aggregation step combining these binary classifications to produce a single-genre classification from the most probable between segment classification [1].

We incorporated some of these ideas into a multitask learning agent which learns these binary classifications as a secondary task over the whole song creating 10 or 25 additional binary outputs in addition to the overall genre classification. The error was calculated as for the full classification problem using the cross-entropy on genre probabilities.

For the secondary auto-encoder task we included an output layer with all 3000 input dimensions and computed the mean squared error between the input-target and the output.

Investigative Hypothesis

We expect the choice of secondary task to have some bearing on model performance. Specifically, we supposed that the secondary binary classification task would lend a more useful representation than the auto-encoder task which has a largely independent goal. This would result in better perfor-

mance for the multitask binary classifier model. Secondarily, we also seek to compare and contrast the performance of ensemble and multi-task methods on MSD in general.

2.4 Implementation with TensorFlow

We make use of the basic graph constructors provided as well as the TensorFlow *rnn* directory for creating basic LSTM cells as well as the *nn* and TenssorFlow and math operations for generating error and accuracy signals. We use a number of helper classes for handling of model construction (RNN_Model and its associated subclasses) with plotting integration (see *MultiPlot*). These classes are located in*rnn_networks.py*.

There are also simple built-in functions for saving and loading relevant performance statistics of RNN_Model objects located within the RNN_Model class as well as the *MultiPlot* construction. Separate notebooks for model generation and plotting have been included with this report.

3 Baseline for Regularization Methods

We compared the performances of both L2 regularization and AS on MSD25 and MSD10. Performance in this context was judged on the basis of validation accuracy and training time, measured by average time per epoch or number of epochs to peak accuracy. For both methods we investigated a range of parametrizations, based on those that worked for feed-forward networks in Assignment 3 (for L2) or values used in the literature (for AS) [5]. In both cases the parameters of interest were the number of hidden units and the regularization loss coefficient, λ for L2, and β for AS.

3.1 Performance on MSD25

Heatmaps showing the peak accuracy for each method with different parametrizations are shown in Figure 2. On the whole, AS performs much better, attaining accuracies that far outdid (by more than 1% accuracy) L2 systems with the same complexity (hidden layer width). Activation stabilization also attained more consistent results across different parametrisations with the value of β having little effect on training.

Figure 3(a) shows that time complexity is largely determined by hidden layer width for AS. This is also true for L2 which attains comparable time complexities given a constant layer width (e.g. 92s/epoch for L2 versus 100s/epoch

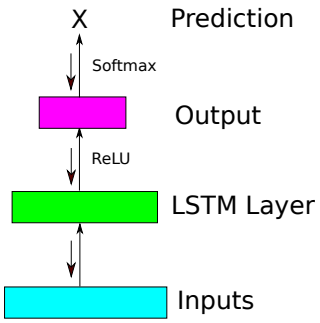
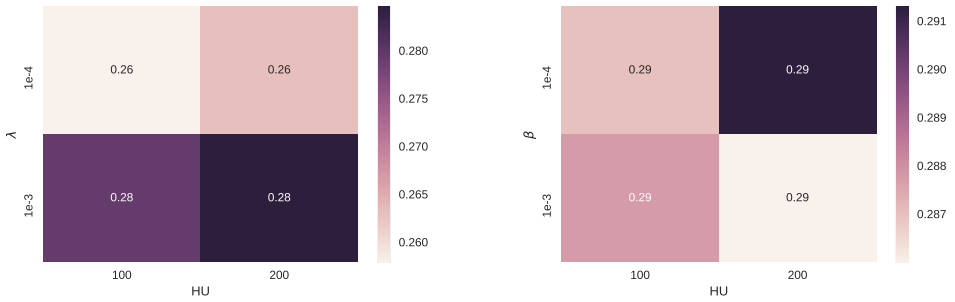


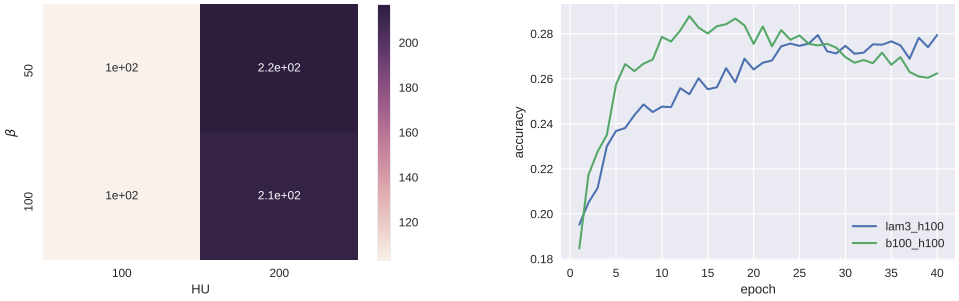
Figure 1: Basic LSTM network architecture showing the structure of the network along with the direction of input (black arrows) and error messages (colour coded arrows) in the forward and back-propagation steps.



(a) (b)

Figure 2: MSD25 Heat maps of peak classification accuracy on MSD25 for systems run over 40 epochs. (a.) Shows the accuracy for parametrisations of L2 whilst (b.) shows the accuracy for activation stabilization methods.

for AS for 100 HU) . Although wider networks performed better, Figure shows that systems with 200 hidden units are more than doubly as computationally intensive to train on average. This makes the minor gains in accuracy for systems using AS with 200 HU not worth pursuing further. Direct comparison of the best 100 HU models for AS and L2 in Figure 3(b) show that AS also attains peak accuracy earlier in training making it more efficient to train when implementing an early stopping strategy. We used this architecture as the new baseline.



(a) (b)

Figure 3: MSD25 Performance plots for simple LSTM models trained over 40 epochs. (a.) Heat map of average run time/epoch for systems using AS. (b.) Accuracy training curves for the two best 100 HU models for L2 (λ=1e-3) and AS ($\beta = 100$).

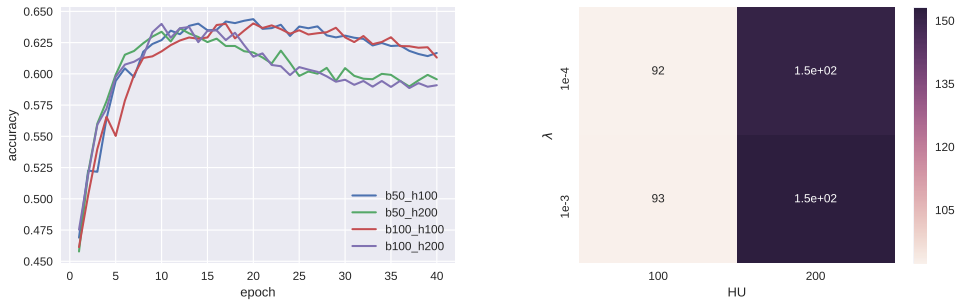
3.2 Comments on MSD10 together with MSD25

The performance on MSD10 is largely similar in time complexity and performance hierarchy for both L2 and AS, making AS with $\beta = 100$ and 100 HU the universal best baseline architecture of those tested with an accuracy of 64% (see Figure 4(a)) for MSD10 and 29% for MSD25. That AS clearly dominates L2 for all parametrisations investigated is clear evidence that L2 may not be well suited to RNNs, supporting the research hypothesis put forward in te literature.

It is also useful to note that these new RNN baselines vastly outperform the baselines set for basic feed-forward architectures in Assignment 3 (49% on MSD10 and 22% on MSD25). This is validation of the capacity of RNNs to detect temporal structure in music data.

4 Ensemble Learning

By cutting songs into snippets, the snippet ensemble learning approach is aimed at representing characteristic elements phases in song progression independently of one another. However, forcing this separation involves a trade-off as it prevents the RNN layer from learning longer-term temporal dependencies within the song. It is thus unclear a priori as to whether this approach should really improve performance. Even so, we investigated the ensemble architecture by testing models using 2 and 4 snippets as well as



(a) (b)

Figure 4: MSD10 Performance plots for simple LSTM models trained over 40 epochs. (b.) Accuracy training curves for every example AS parametrisation ($\beta = 100$. (a.) Heat map of average run time/epoch for systems using L2.).

assessed the two prediction pooling methods m_0 and m_1. The number of hidden units for each model remained constant at 100 HU each.

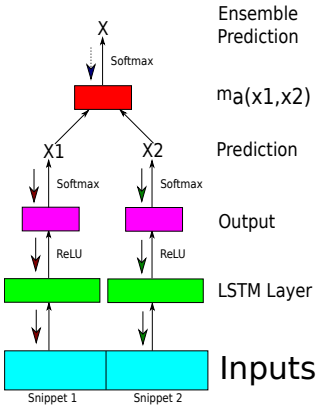
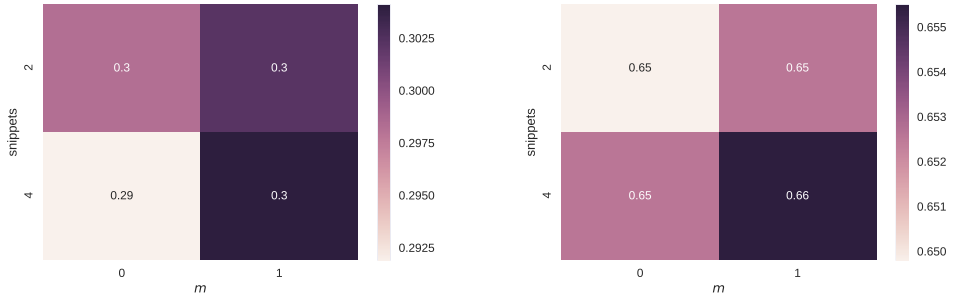


Figure 5: Ensemble network architecture showing the structure of the network along with the direction of input (black arrows) and error messages (colour coded arrows) in the forward and back-propagation steps. Here the dotted arrow depends on whether the ensemble function m_a has learnable weights.

4.1 Performance on MSD10 and MSD25

Figure 6 shows that minor but significant performance improvements (again roughly 1%) that were achieved using this ensemble architecture for both



(a) (b)

Figure 6: Peak accuracy heat maps for models trained over 30 epochs on (a.) MSD25 and (b.) MSD10.

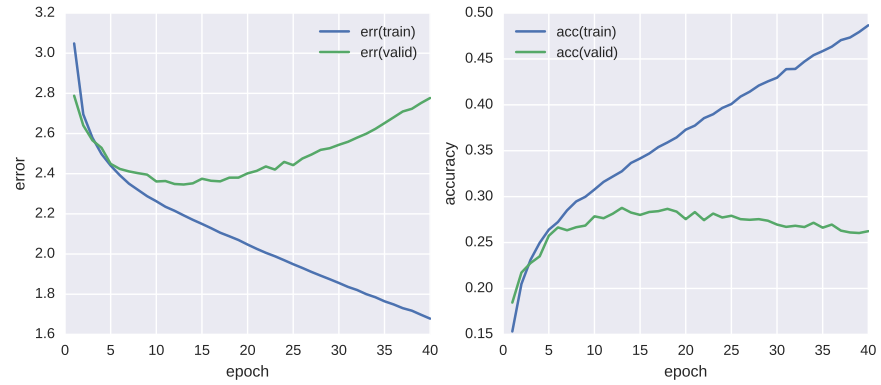
MSD10 and MSD25. Note that across both datasets m_1 (max-product) appears to outperform m_0 (max-mean) corroborating the findings of Silla and Kaestner [1]. In addition, creating more snippets appears to be advantageous and the best performance in both the MSD25 (30% accuracy) and MSD10 (66% accuracy) case is achieved from four snippets.

One important observation is the apparent regularizing effect which the ensemble approach has on overal prediction error on the validation set (as distinct from training error used to fit the snippet models) that appears to prevent the validation error from inflating (see Figure 7). This could be due to an averaging out of over-training error leading to more stable predictions.

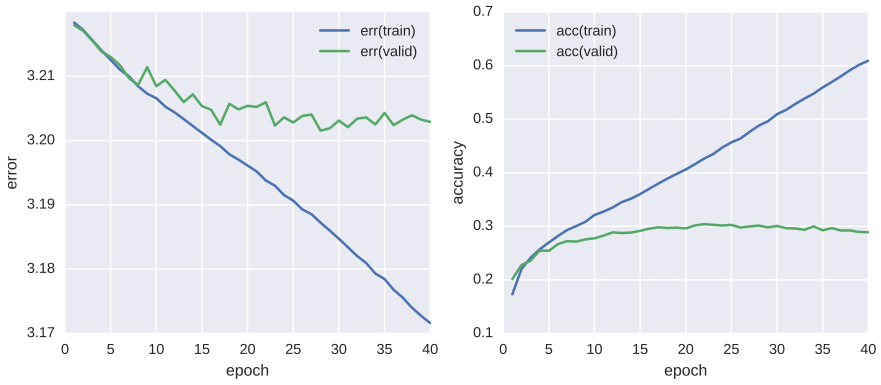
Another advantage of ensemble networks on of this kind is that the input snippets reduces computational complexity by lowering the number of weights in the input-to-hidden layer matrix (even when the number of HU remains the same), thereby making the overall model quicker to train. This can be seen in the per epoch training times with the models taking an average of 70-90s/epoch to train.

4.2 Performance of m_2 on MSD25

The adaptive weighted ensemble method m_2 is based on a weighted average derived from a secondary, independently trained linear network used an L2 regularization scheme with λ =1e-3. This seccondary network was trained simultaneously with the ensemble models. The method did not perform as well as anticipated (see Figure 8) with no improvement in performance (29% validation accuracy) over the direct ensemble averaging method (m_0) or even



(a) Basic LSTM



(b) Ensemble

Figure 7: MSD25 Training (error and accuracy) curves for (a.) the baseline LSTM model (b.) Ensemble LSTM with 4 snippets using m_1 (max product).

the MSD25 baseline.

The inclusion of the secondary network also seems to have undone some of the regularizing effects of the initial m_0 method. Though there is some evidence that this ensemble method may speed convergence as the system reaches maximal validation accuracy after just 10 epochs.

5 Multitask Learning

Multitask learning is centred around an explorative approach to learning in which the system is pushed towards representations it would not otherwise

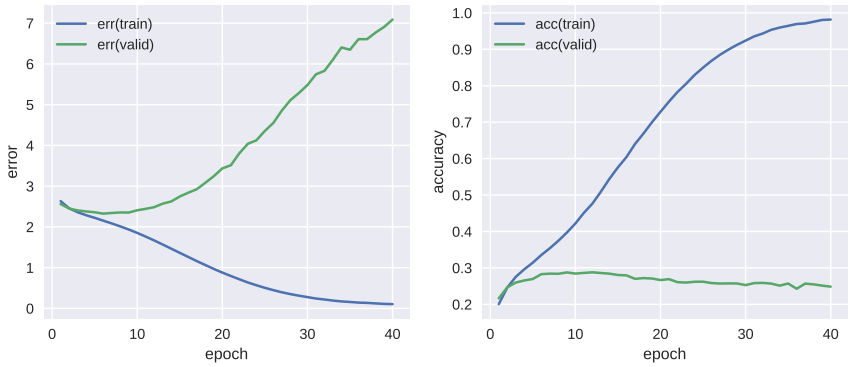


Figure 8: MSD25 Training curves for an ensemble model using m_2 (adaptive weighted average).

find appealing (a sort of adaptive regularization) because of the presence of a secondary task that shares a portion of the learning parameters (in this case the internal LSTM and hidden-to-input weights). This influence is affected via the training error E which takes the following form.

$$E = E_p(W) + \alpha \sum_{i=1}^{K} E_i(W)$$

where E_p is the error associated with the primary task, W are the shared model parameters and the E_i are the errors associated with the K secondary tasks. The constant $\alpha > 0$ weights the relative importance of the secondary tasks in training, thereby changing the error space. The two error types we explore are softmax error (for secondary binary classification of each genre) and sum of squares error (for the secondary auto-encoder). We also attempt to vary α to assess its effect on performance.

5.1 Comments on Secondary Binary Classification

By breaking up the target into a set of binary variables Y_i that are 1 if the target is in genre and 0 if not we are biasing the secondary classification task as only a tenth or less of the targets will be in this genre. This bias has the potential to skew training but since we are not directly interested in the secondary task, we have chosen to ignore it in this report.

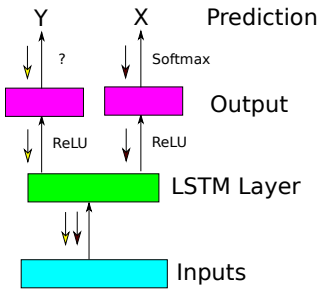
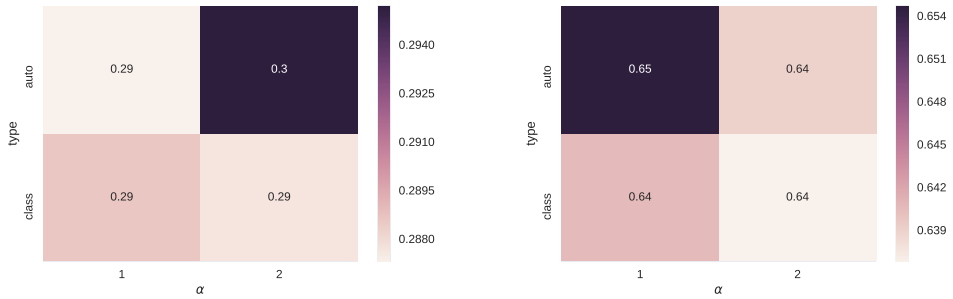


Figure 9: General mutitask network architecture showing the structure of the network along with the direction of input (black arrows) and error messages (colour coded arrows) in the forward and back-propagation steps. Here, a ? depends on the sort of secondary task Y.

5.2 Auto-encoding & Classification Comparison

Results for both types of multi-task models are less robust than those for ensemble models, with poorer and more erratic average performance across different parametrisations (see Figure 10). Although there is some improvement from the LSTM baseline in general this improvement is between 0.5% and 1% with some systems doing worse than baseline performance.



(a) Basic LSTM

(b) Ensemble

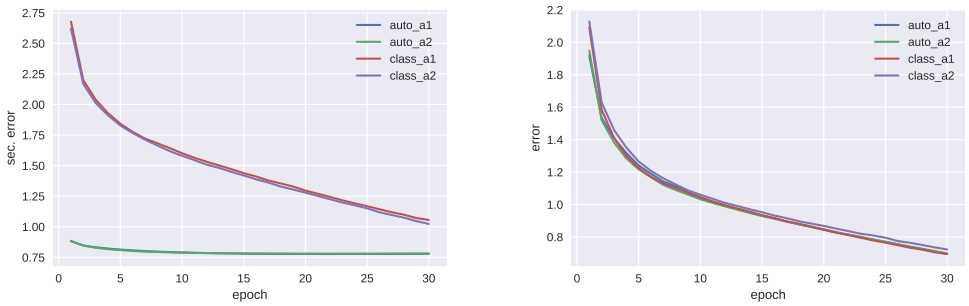
Figure 10: Heat maps of peak classification accuracy for multitask systems run over 30 epochs for (a.) MSD25 and (b.) MSD10.

Most surprisingly, it is the systems running the auto-encoder secondary task that tend to out-perform their binary classification counterparts. This could be because the binary classification and general genre classification tasks are too similar or due to the aforementioned skew in the data (see Sec. 5.1). In terms of time complexity, both systems had similar average per epoch run

times of 110s.

Figure **??** shows the secondary task training error curve for models trained on MSD10 which could help explain the difference between binary and auto-encoder models. Here we see that training for the binary classification task occurs only at the start of training with the error becoming practically stationary after 10 epochs. This is in comparison to the auto-encoder training error which decreases consistently through training.

It is also interesting to note how little α appears to effect the training trajectory with the secondary task training curves for $\alpha = 2$ only very slightly below those for $\alpha = 1$ (i.e. the weightings explored for the secondary task error have minimal effect on training). Figure 11 shows how secondary and primary training error evolve over each epoch with the auto-encoder training curve taking the same general form as the training curve of the error of the primary task.



(a) (b)

Figure 11: MSD10 Training error curves for (a.) secondary task training error and (b.) primary task training error.

6 Discussion

Our results are largely in agreement with the literature. We found that AS and not L2 performed better as suggested by prior research [6]. We also found that both ensemble methods improved performance and that the max product approach out-performed the max mean approach for ensemble models, corroborating the results of Silla and Kaestner [1]. However, no evidence

could be found showing a direct regularizing effect for multi-task learning.

Between the two methods explored, it seems clear that ensemble methods far out-performed their multi-task counter-parts with the best overall model coming from a four snippet, max product ensemble architecture which performed best on both MSD10 and MSD25 (see Figure 12). It is even out-performed systems using more complex architectures, such as the weighted averaging used in Sec. 4.2. Testing of this model on the withheld test set for MSD10 using an early stopping approach (training for only 30 epochs), we found little difference ($< 1\%$ accuracy) between the validation (66%) and test (65%) accuracies, showing that the final model generalises quite well.

Where we found discrepancies between the results and our expectations were from some of our own hypotheses, where we suggested that the binary classifier multitask system would out-perform its auto-encoder counter-part. This was not the case and in fact the auto-encoder generally performed better and trained more consistently. It was also expected that the auto-encoding task would be significantly more time intensive, however, no evidence was found to support this and the two methods were similarly efficient.

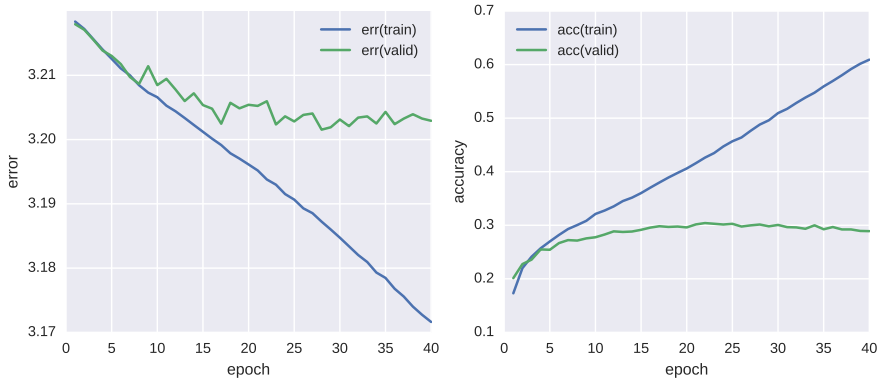
That the best model comes from an ensemble architecture is initially surprising but may result from a number of factors, including the compartmentalised structure of songs and the averaging of errors, which also resulted in the regularizing effect which we observed. Whether ensemble learning will generalise to deeper or more complex architectures with LSTMs, ones that could capture longer-term, higher-level temporal dependencies, remains to be seen.

References

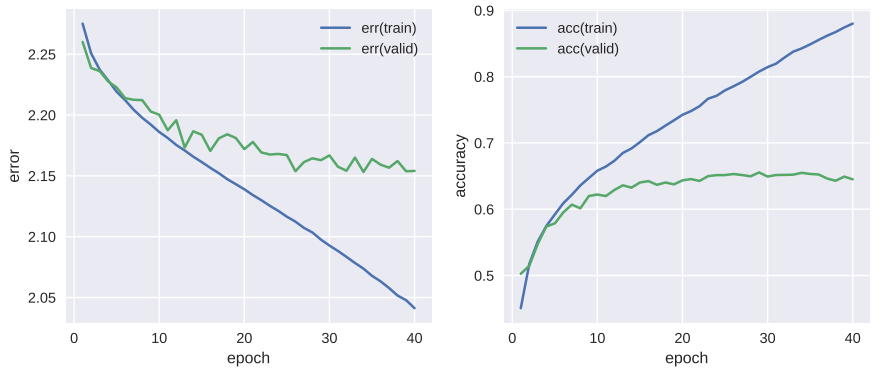
[1] Carlos N Silla Jr, Celso AA Kaestner, and Alessandro L Koerich. Automatic music genre classification using ensemble of classifiers. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pages 1687–1692. IEEE, 2007.

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

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.



(a) MSD25



(b) MSD10

Figure 12: MSD25 Training (error and accuracy) curves for (a.) the best performing MSD25 model and (b.) the best performing MSD10 model which used a max product ensemble approach with four snippets.

[4] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] David Krueger and Roland Memisevic. Regularizing rnns by stabilizing activations. In *Proceeding of the International Conference on Learning Representations*, 2016.

[6] Saahil Ognawala and Justin Bayer. Regularizing recurrent networks-on injected noise and norm-based methods. *arXiv preprint arXiv:1410.5684*, 2014.