UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning and Machine Intelligence 2022-2023

Module Coursework Feedback

Module Title: Speech Recognition

Module Code: MLMI2

Candidate Number: K5002

Coursework Number: 1

I confirm that this piece of work is my own unaided effort and adheres to the Department of Engineering's guidelines on plagiarism. √

Date Marked: Click here to enter a date. Marker's Name(s): Click here to enter text.

Marker's Comments:

This piece of work has been completed to the following standard *(Please circle as appropriate)*:

	Distinction			**Pass**			**Fail (C+ - marginal fail)**		
Overall assessment (circle grade)	**Outstanding**	**A+**	**A**	**A-**	**B+**	**B**	**C+**	**C**	**Unsatisfactory**
Guideline mark (%)	90-100	80-89	75-79	70-74	65-69	60-64	**55-59**	**50-54**	**0-49**
Penalties	**10% of mark for each day, or part day, late (Sunday excluded).**								

The assignment grades are given **for information only**; results are provisional and are subject to confirmation at the Final Examiners Meeting and by the Department of Engineering Degree Committee.

MLMI2 - TIMIT Speech Recognition Using CTC

K5002

January 23, 2023

Contents

1 Introduction

This report explores the various methods that can be used to optimise a bidirectional recurrent neural network to recognise phones in natural speech. Considering numerous training parameters such as the batch size, optimiser, and learning rates. As well as regularisation techniques such as gradient clipping and dropout. The optimal LSTM architecture is also considered by comparing a variety of architectures with a range of parameters. We conclude by visualising the trained network using a heat-map to summarise the probability distribution over phones for an exemplar signal; followed by a review of performance on a test split via a confusion matrix visualising the most common errors made by the network.

2 TIMIT

The TIMIT dataset is an Acoustic-Phonetic Continuous speech dataset, meaning that given an audio signal, the goal is to generate the list of phones within a recording of some speech. Although this dataset is a poor predictor of word level speech recognition it enables us to quantify the accuracy of a models ability to identify components of speech. The dataset consists of 3696 training samples from 630 speakers each sampled at 16bit, 16KHz, and annotated with the constituent phones present from a set of 61 possible phones.

3 CTC Loss

The connectionist temporal classification loss (CTC)[2] provides a measure of how well a probability distribution of phones over time aligns with a list of known true phones of a different length. Classically, a model would require the inputs to be segmented into the starting points and endpoints of each phone such that a one-to-one mapping exists between predicted and target phones. CTC allows a long unsegmented list of features (at least as long as the target list) to be mapped to a target list in a many-to-one relation. First CTC generates a probability over phone predictions at each time-step by applying the SoftMax where t is the current time step within that sequence and j is the phone that the predicted probability corresponds to.

$$p_{t,j} = \frac{\exp(logits[t,j]}{sum_{k=0}^{K} \exp logits[t,k]} \tag{1}$$

Next, the set of all permutations of possible merges of repeated sequential phones is computed. From these, all unique sequences of phones are extracted from the target lists, called paths. Finally each of these paths has its alignment with the predicted probabilities tested in the product bellow, where S is a path.

$$p(S) = \prod_{t=1}^{L_i} p_{t,ct} \tag{2}$$

Finally, the negative log sum of these alignments is computed giving the final loss:

$$CTCLoss = -\ln \sum_{S} p(S) \tag{3}$$

4 Mel Feature banks vs MFCCs

There are two commonly used methods to encode audio signals as features in order to process them using neural networks. The first of which are Mel-Scale Feature banks. Which according to our understanding of human hearing, allocates greater resolution for lower frequencies and less resolution at higher frequencies since, it is much harder for humans to distinguish a 10Hz change at 5000Hz than it is to distinguish one at 15Hz. It applies the logarithmic transformation in equation 4 where f is spectra measured in hertz to form a new spectrum where constant increases in the Mel-scale have exponential increases in frequency. Features are then a vector of samples of the amount of energy in each band of the Mel-scale over a period of time from an audio signal. Note that the unit offset within the log causes the scale to be nearly linear up to ∼1000Hz.

$$Mel(f) = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{4}$$

MFCCs add to Mel Feature banks, summarising them further according to their Cepstral Coefficients. It makes use of the discrete cosine transform (DCT) to take a real valued decomposition of a Mel feature bank forming n *Mel-frequency cepstral coefficients* according to equation 5. The use of a DCT to perform *whitening* was originally inspired by the need to simplify covariance matrices when using Gaussian Mixture Models and Hidden Markov Models to model sequences of speech, as it forms a diagonal covariance matrix which can be represented with a signal vector.

$$c_n = \sqrt{\frac{2}{P}} \sum_{i=1}^{P} m_i \cos\left[\frac{n(i-\frac{1}{2})\pi}{P}\right] \tag{5}$$

MFCC features have classically been preferred compared to Mel feature banks due to their reduced size yet and high information density, reducing the memory and computational requirements. However recently as computation power increases Mel Feature banks have become increasingly popular[1]. Following this trend, Mel feature banks are used exclusively in this report.

5 Architecture

First the sequence of inputs is passed through a single LSTM layer. Long Short Term Memory layers take a time series input of features and generate an output sequence of equal length while maintaining a hidden state encoding the information gathered by the network upto that point. They rely on four component gates: an input gate, an output gate, a forget gate, and a cell state, where each gate is a dense layer mapping from a concatenation of the input and hidden state.

Clearly, earlier time steps in the sequence are at a disadvantage currently, as the prediction at that time-step must be made with no prior context. This can be resolved by training 2 networks in parallel, where each reads the input sequence in opposite directions. Doing so ensures that every time step within the input sequence has some corresponding prediction based on the context gathered on either side of it in the time series. The LSTM produces an output of equal shape as it's hidden state, in order to maintain a hidden state that is sufficiently large to capture the context of an input (128 by default) we must adapt the large output time series to a time series with a size equal to the number of phones we are trying to predict. This is achieved via a single dense output layer with 256 inputs (because the bidirectional layer double the number of outputs) and 40 outputs (corresponding to the size of our phone set).

6 Testing Strategy

In order to expose every aspect of the model during training tensorboard was employed. Recording test statistics, such as evaluation/test accuracy and loss. As well as the statistics of tensors such as the mean/std/max of gradients, weights and biases. The most useful measurement was the training loss, enabling the tracking of a model's performance during training to prevent wasted computation. In order to build the customised graphs used in this report, all metrics recorded in tensorboard were also recorded in a pandas data frame using a wrapper around the tensorboard logger. Enabling easy access for analysis later. The hyper parameters used for each experiment were also recorded, allowing the optimal configuration to be summarised in a parallel lines chart in figure 14. This was all configured through a script enabling several experiments to be run in sequence automatically via a set of chosen parameters.

7 Hyper-parameter Tuning

Now we begin to explore the hyper-parameters of the model and how the model performs over a range of settings, each parameter is tuned using a grid search over ~ 5 different parameter settings. To prevent bias, only the training split and validation split will be considered in this section, with the test split being kept hidden until model evaluation.

7.1 Initial Configuration

First, we consider training the model with the default parameters and explore the key metrics that measure its performance. The first of which is the training loss, which indicates how well the model is fitting to the dataset. Next we have the validation loss, which tests the models accuracy on a split of data that is unseen during training, giving an indication of how well the model is generalising to the

problem. Similarly, we also record the validation accuracy each epoch which quantitatively measures how well we are fitting the validation data split. Finally we test the best performing model (on the validation split) against an unseen test split. As shown in Figure 1

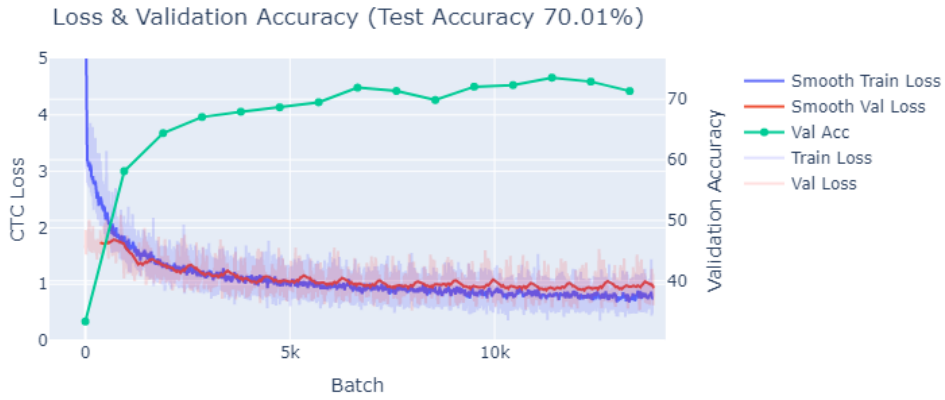
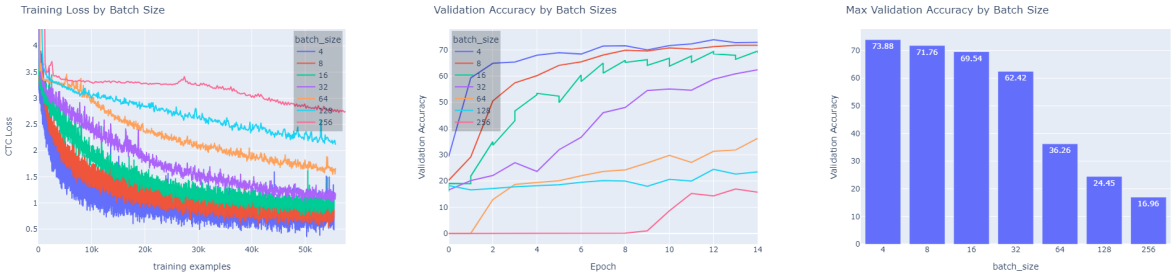


Figure 1: Training vs Evaluation Loss + Evaluation Accuracy

Overall, the validation loss closely follows the training loss but diverges slightly towards after 7 epochs as the model shows signs of over-fitting. At its peak the validation accuracy reached 73.88% which will be used as a benchmark for future tests.

7.2 Batch Size

The batch size controls how many training examples will be processed within a single call of the model. Crucially, the loss will be calculated for an entire batch, meaning that for larger batches the loss reflects the performance of a greater number of examples and is less sensitive to noise; conversely, this reduced sensitivity makes training less precise. Ideally a balance must be stricken in order to maximise performance, accordingly a range of batch sizes were tested and their performance recorded. Considering the loss curves in figure 2a we can clearly see the reduced sensitivity to noise in the larger batch sizes as the loss curve is much smoother. However, due to the reduced precision, the training loss increases with batch size. Measuring the accuracy of the model (Figure 2b) and taking the maximum achieved (Figure 2c) we can confirm the trend and learn that increasing batch size decreases performance. This means that in our scenario precision is far more important compared to robustness to noise. Before simply choosing the best performing batch size we must also consider the performance limitations of each batch size. Large batch sizes require more memory while small batch sizes increase program overheads, increasing training time. Considering the trend of training times (Table 1), a batch size of 16 was chosen as it halves the training time while only reducing performance by 4%. As such our new reference performance when using batch sizes of 16 will be 71.76%.



(a) Loss (b) Validation Accuracy (c) Max Validation

Figure 2: Performance for a range of batch sizes

Batch Size	Training Time (mins)
4	4
8	3
16	2
32	2

Table 1: Training time for various batch sizes

The learning rate was kept constant as both batch sizes 8 and 16 fully converged within the allocated number of epochs yet still had reduced performances. However as a sanity check, assuming the loss implementation does not normalise by batch size, a learning rate proportional to the batch size was also tested. Which demonstrated even worse results, showing that regardless of learning rate, large batch sizes perform worse than small batch sizes.

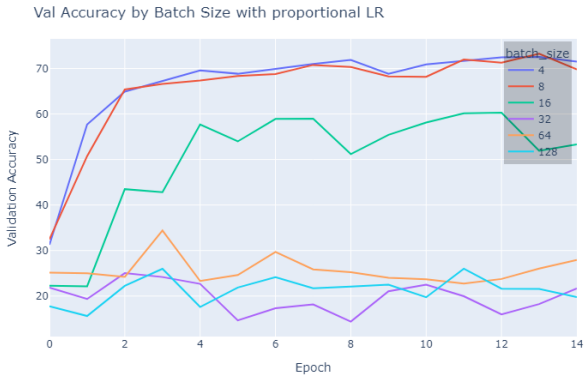


Figure 3: Performance for a range of batch sizes with proportional learning rates

7.3 Dropout Regularisation

Dropout is a common regularisation technique to help reduce overfit in complex models by ensuring that the model does not become dependent on a small number of features, making the network more robust to unseen data. It functions by randomly masking a certain percentage of the outputs of a layer during training, then at test time the masking is removed allowing all the learned parameters to be used. Note that the output of dropout layers has to be re-normalised at test time as the standard deviation

would be larger than what was seen during training. The results of adding a dropout layer between the LSTM and the dense output layer and varying the dropout percentage have been summarised in Figure 4. Clearly, varying the percentage of dropout has a detrimental effect on performance, however this does not give us the full picture.

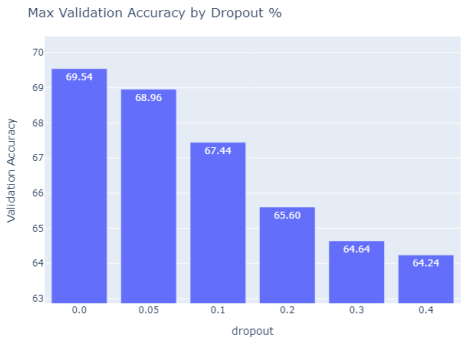


Figure 4: Dropout Sweep

As a direct comparison to Figure 1 the exact same parameters were used but with 20% dropout. Following our earlier findings the accuracy of the model is worse. However, the divergence between the training and evaluation loss that we remarked on has been removed. Demonstrating that dropout does reduce model overfit.

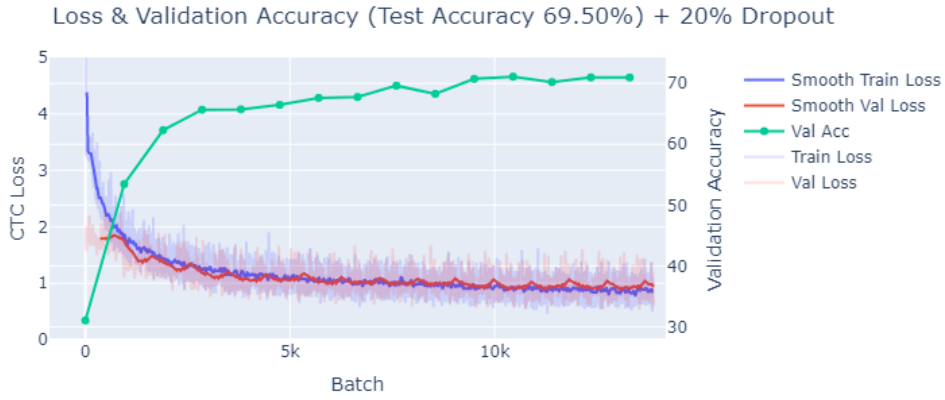


Figure 5: Training vs Evaluation Loss + Evaluation Accuracy with 20% Dropout

This poses the question of how many epochs are necessary before the degree of divergence seen in Figure 1 can be seen when using dropout, doubling the number of training epochs to 30 achieves just this, while also increasing the maximum validation accuracy from 71.76% to 73.73%. This indicates that dropout can be used to increase model performance, however more training epochs are required to do so.

In summary dropout can be an effective tool to help models generalise better to unseen data, but require more computation, in the form of training epochs, in order to do so.

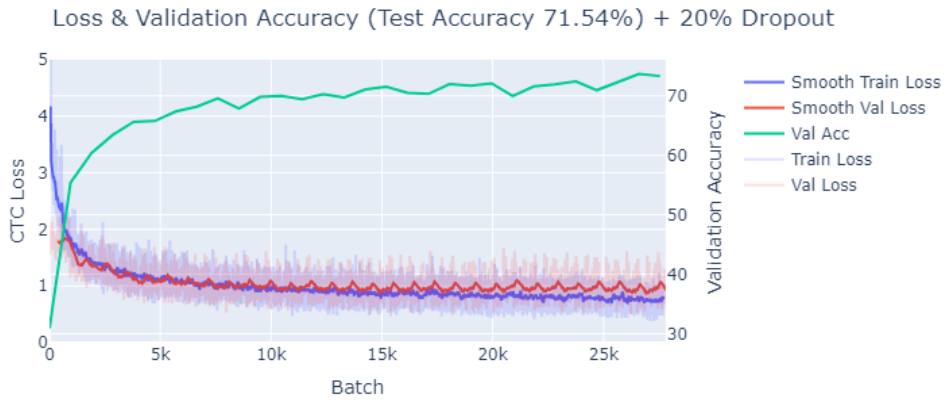
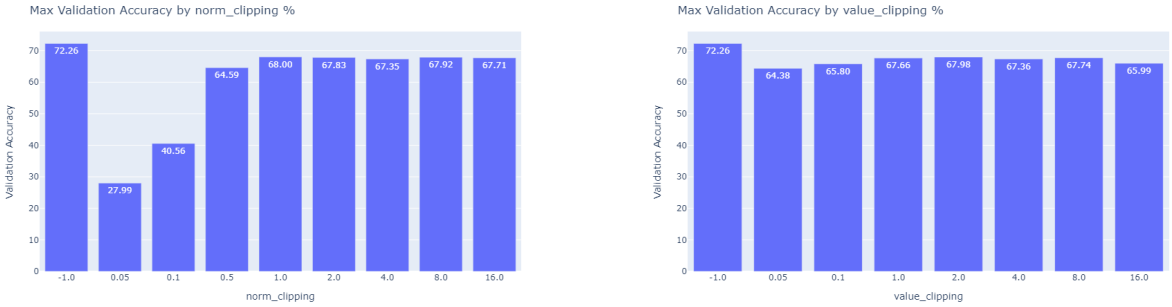


Figure 6: 20% Dropout for more epochs

7.4 Gradient Clipping Regularisation

Regularisation can also be used to mitigate exploding gradients through value clipping and norm clipping. Value clipping ensures that the maximum absolute value of gradients does not exceed some threshold. While norm clipping is more relaxed and only maintains the determinant of gradients below a certain threshold. Theoretically, this will not directly improve performance, but will make training more reliable, this can be seen when we vary the clipping threshold of either algorithm in Figure 7. Although we can note that for very small thresholds training is impeded as gradients are normalised too much preventing the network from learning, especially when using norm clipping, although smaller values of value clipping have the same effect.

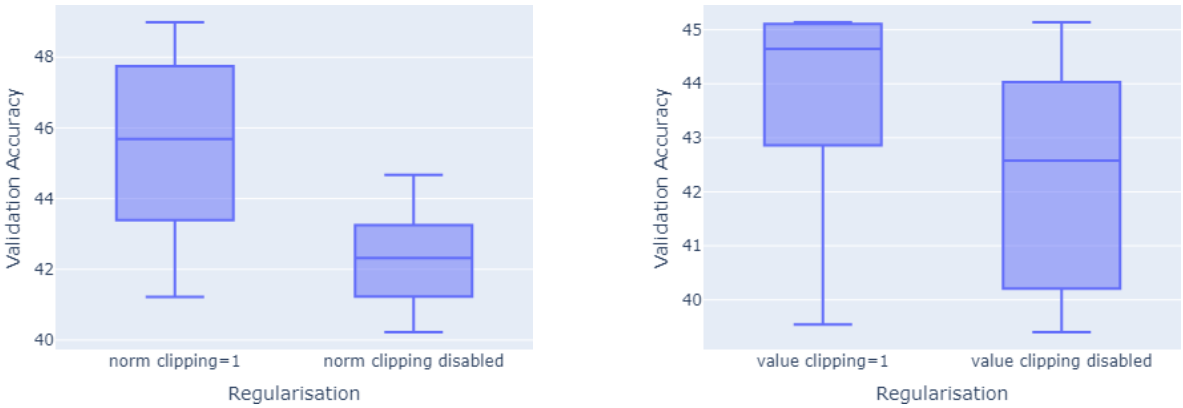


(a) Norm Clipping

(b) Value Clipping

Figure 7: Performance for a range of clipping parameters

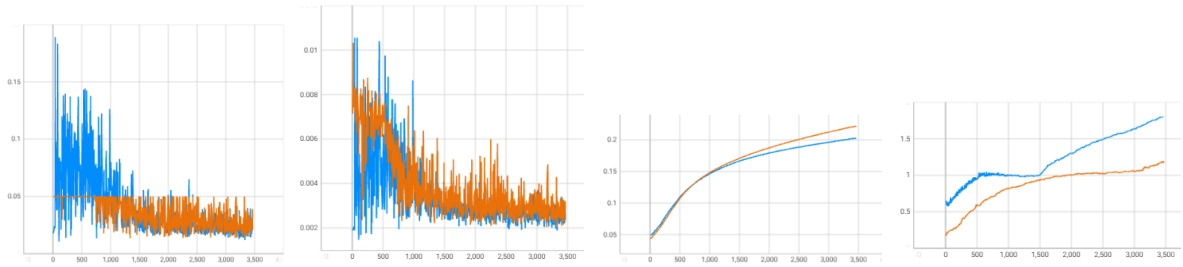
A better measure of the reliability gains of using gradient clipping can be seen when running multiple trials and comparing the average performance with gradient clipping enabled vs disabled. In Figure 8, 6 models were trained with and without gradient clipping regularisation and their performance measured. In both cases the mean accuracy when clipping was enabled is higher than that when it is disabled, showing that gradient clipping does in fact improve the reliability of training. Although the minimum accuracy in either case is roughly equal, so gradient clipping does not explicitly improve performance.



(a) Norm Clipping (b) Value Clipping

Figure 8: Performance for a range of clipping parameters

It is also useful to visualise how gradient clipping affects the values of gradients within the network and how this correspondingly impacts the learnt parameters of the network. Figure 9a directly shows how the gradients differ when value clipping is employed. The maximum value when clipping is not used is arbitrary, in this example it peaks at 0.19, however when clipping is used the maximum value is constrained below 0.05. Analysing the standard deviation of the gradients reveals that the model trained with clipping has much less noise in it's gradients, with the standard deviation following a consistent trend. Without clipping the gradient is sporadic. This implies that the learning of the network is more stable when using gradient clipping, as similar errors are consistently back-propagated through the network, unaffected by noise. At a high level the standard deviation of the actual weights (Figure 9c) is very similar in either case. Supporting the hypothesis that gradient clipping does not affect the overall performance of training only it's reliability. Although the maximum value of the weights (Figure 9c) grows much more slowly when gradient clipping is enabled, suggesting that overfitting will be reduced.



(a) Gradient abs max (b) Gradient std (c) Weights std (d) Weights abs max

Figure 9: Projection layer Gradient/Weight Mean & Standard deviation - blue:no clipping, orange: value clipping 0.05

7.5 Optimiser

The optimiser is responsible for optimising the learning rate during training which affects how quickly a model will train according to the error given by the loss function, and the consequent gradients. The simplest optimiser is stochastic gradient descent (SGD) which maintains a constant learning rate during training. However, this is not ideal as it is particularly susceptible to converging to local minima

since a peak in the loss landscape cannot be passed if it requires a step larger than the learning rate permits. Secondly it does not encourage fine tuning when the network is near a local optima, relying only on decreasing the loss rather than decaying the learning rate during training.

The impact of the learning rate on SGD training has been visualised in Figure 10, where you can see that small learning rates do not converge fast enough and large learning rates converge too fast.

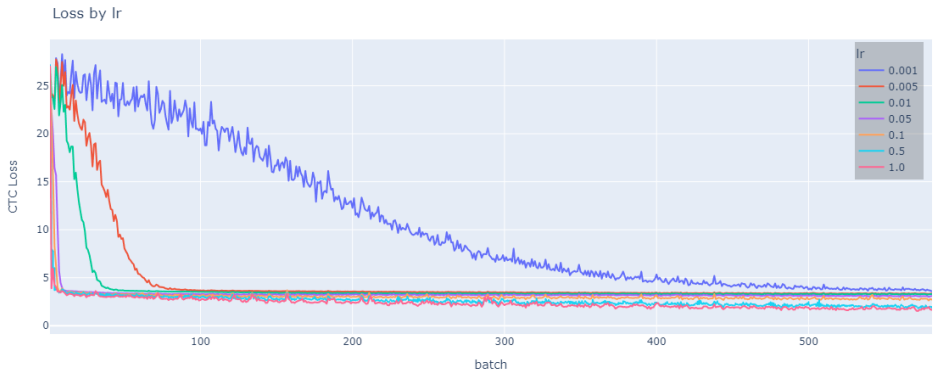


Figure 10: SGD loss curves for various learning rates

Adam is an optimiser that adjusts the learning rate using 1^{st} and 2^{nd} moments. The first moment is a moving average of the gradients which can be thought of a body with mass traversing the loss landscape accelerating gradually on descents. While the 2^{nd} moment is a moving average of the squared gradients, which is commonly considered as a friction preventing oscillations when converging. AdamW extends on Adam by introducing weight decay, which adds the norm of model parameters to the loss function, encouraging simpler models, and reducing overfit.

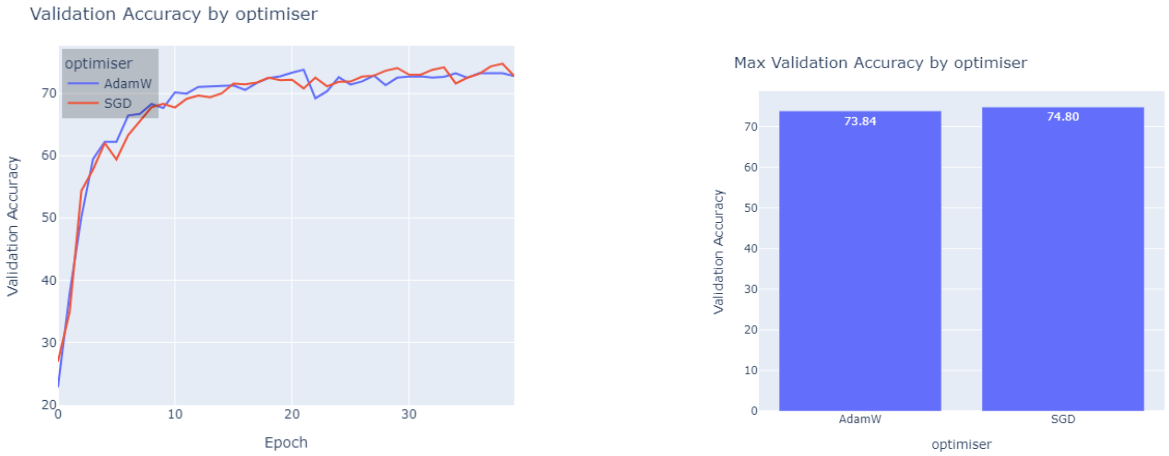
For a fair comparison between optimisers the learning rate must be optimised for the each, Figure 11 shows the performance of models trained over a range of learning rates, indicating that optimal learning rates for SGD, Adam and AdamW are 1.0, 0.005 and 0.005 respectively. Adam/AdamW are notably far less sensitive thanks to their automatic adjustment of the learning rate during training.

(a) SGD (b) Adam (c) AdamW

Figure 11: Optimal learning rates for SGD/Adam/AdamW

It is also worth considering whether SGD or Adam converges to a good solution quicker or not. And whether it outperforms SGD after many iterations. Consequently the optimal learning rates for SGD and Adam were used, and the model was trained for 40 epochs. Figure 12a demonstrates that for

optimal learning rates SGD and Adam perform almost identically throughout training, with the peak accuracy being within margin or error to one another (Figure 12b). However, Adam will be preferred thanks to it's robustness to sub-optimal learning rates.



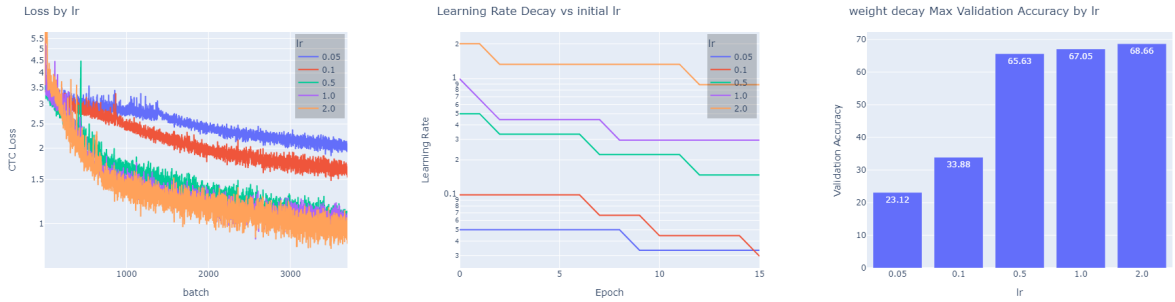
(a) Validation Accuracy

(b) Max Validation Accuracy

Figure 12: SGD vs Adam

7.6 Learning Rate Decay

Learning rate decay refers to any technique that reduces the learning rate during training to help model convergence. Typically it is a form of scheduler that modifies the learning rate as a function of the number of epochs, however, it can be extended to be a function of the evaluation accuracy. We implement a scheduler that reduces the learning rate when the evaluation accuracy increases. Because it influences how the learning rate changes over time, the initial learning and the reduction profile must be re-optimised. We first trailed a profile that doubled the learning rate every time the accuracy doubled, however, since the majority of the accuracy increase occurs in a single epoch this proved to be too dramatic with the learning rate only decreasing once during training. Instead, every 20% improvement in accuracy, the learning rate was reduced by 25% this was found to perform much better(Figures 13a, 13b) with the loss curve looking much more linear than when the learning rate decay was not used; however, the performance still did not surpass stock SGD, with the best performance being only 68.86% (Figure 13c). Overall, this technique proved to be too sensitive to training parameters to be a viable scheduler.



(a) Loss (b) Learning Rate (c) Validation Score

Figure 13: Learning rate decay tests

7.7 Summary

To summarise, the optimal configuration for this architecture is a batch size of 4, 10% dropout, 30 epochs with a value clipping set to 1 and the Adam optimiser with a 0.005 learning rate. This can be confirmed by using a parallel line graph to visualise a set of hyper-parameter combinations in Figure 14. Since there is a high cost to each training run, only a few combinations were tried. None the less, the plot confirms our findings, with dropout being used for a large number of epochs with a 0.5 (SGD) learning rate yielding the best results.

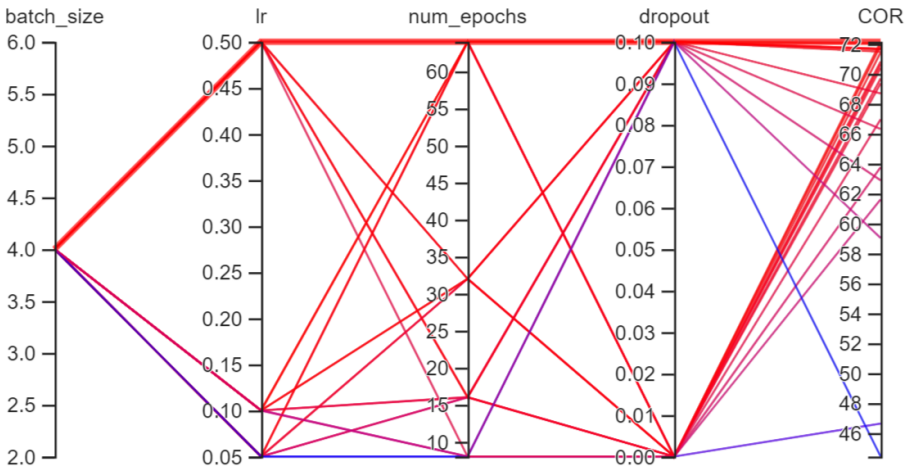


Figure 14: Parallel lines hyper parameter summary

8 Architectural Tuning

There are numerous architectural decisions that were made arbitrarily up to this point. For instance LSTM layers are typically stacked to increase the depth of a model and it's classification power. Also the number of parameters can be varied by changing the size of the hidden dimension of layers, similarly whether bidirectional LSTM's are used, and finally multiple dense layers can be used to create strong non-linearity's on the output of the network.

Using unidirectional layers had the biggest impact on performance, this is because the first few time-steps have very little contextual information to use so make poor predictions. Increasing the number

of LSTM layers had the biggest performance benefit per parameter used, closely followed by increasing the dimensions of LSTM hidden layers. This can be seen by a single LSTM with a 256-element hidden state performing 2.67% worse than a 2 layer LSTM using 128-element hidden states. These 2 stacked layers even outperform a single layer with double the parameters, having a 512 element hidden state, by 0.52%. However combining both improves performance most increasing a single layer 512 LSTM from 75.36% to 81.08%. There is also a marginal return for a third stacked layer, increasing performance to 81.63%. Interestingly, dense layers improved performance by ∼0.86% when 128 element LSTM's were used, even when stacked, but were detrimental to performance when 512 element LSTM's were used. This is likely due to the fact that these large LSTM layers were sufficiently complex alone and that any further parameters caused the model to start to overfit.

Overall there is a strong trend between the number of parameters in a network and it's performance, although evidence suggests that too many parameters can be detrimental to performance, with the key outliers being the single layer 512 element LSTM and the triple layer 512 element LSTM with 3 additional dense layers.

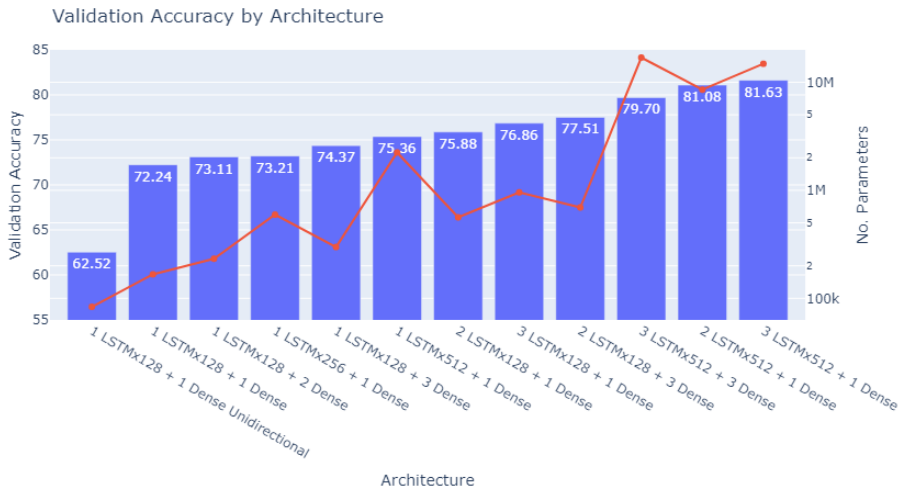


Figure 15: Various tuned architectures

9 Visualisation

The uncertainty in a prediction for an utterance can be inspected by visualising the probability distribution of predictions over time; a good way of achieving this is through a heat map, where each column represents the probabilities over phones in the output layer for a given time step (frame). An interesting example to analyse are the predictions of a trained network for test sample 4, which have been visualised in Figure 16.

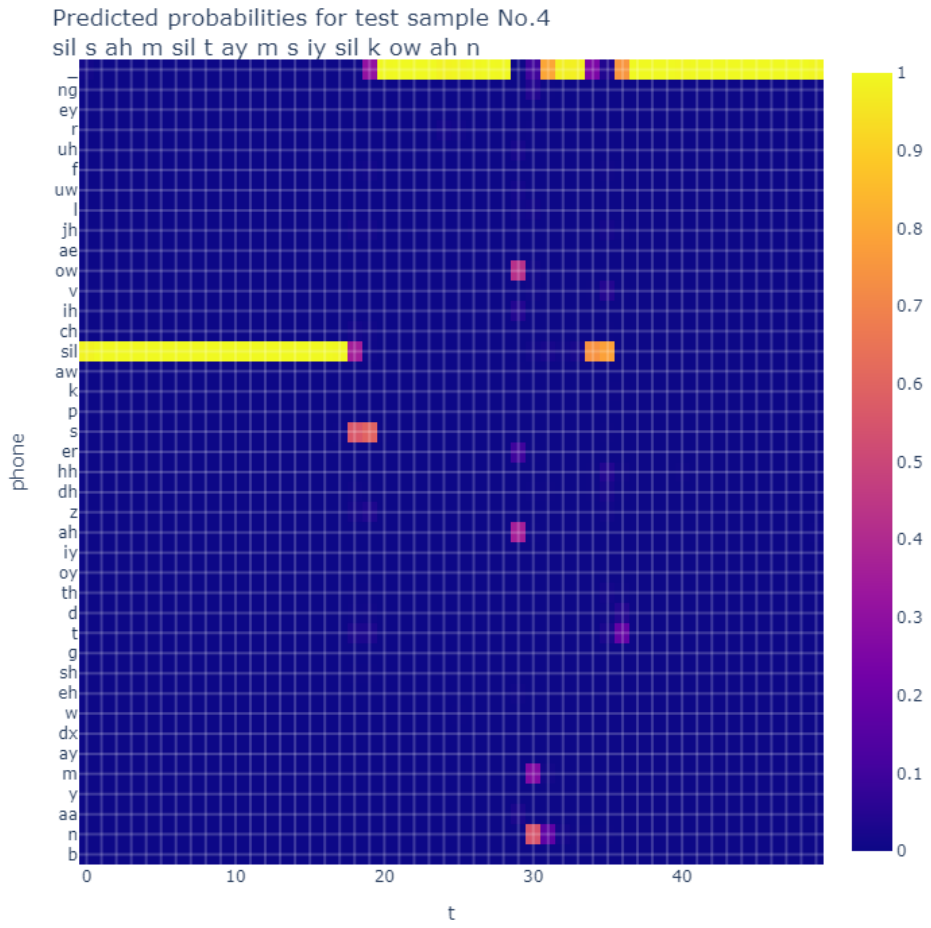


Figure 16: Output probability heat-map

The first 6 phone labels for the utterance are `[sil s ah m sil t]`, while the passage reads "sometimes he coincided with my father's being at home". The first phone is an empathetic silence, which is correctly predicted with 100% certainty, with a soft transition to `s` in the next frame with a reasonable degree of certainty of $p(s) = 0.59$. After a blank space the `ah` phone is incorrectly predicted with $p(ow) = 0.48$ although the correct phone `ah` closely follows it with $p(ah) = 0.36$. This is an unsurprising miss-classification since both `ow` and `ah` represent the vowels 'boat' and 'but' respectively, which can sound similar.

In the next frame `m` is mistakenly predicted as `n` with $p(n) = 0.55$ although, again the correct phone closely follows with $p(m) = 0.27$. Furthermore, just as before, `m` and `n` represent the similar nasals 'mom' and 'noon'.

Finally the penultimate phone `sil` is correctly predicted with $p(sil) = 1$ but the final phone is weakly predicted with $p(t) = 0.19$ where the model opts for a pause $p(\_) = 0.75$.

In conclusion, the heat-map has demonstrated that the model has a good understanding of phones within speech, with the correct answer appearing in the top two predictions consistently in the examples we analysed and in most cases mistakes being made betwen very similar phones.

9.1 Confusion Matrix

Errors for the entire network across the training dataset can be summarised through a confusion matrix that encodes the number of operations required to map between the predicted phone series and the true phones. The *Levenshtein Distance*[3] is a metric that enables this. It defines an algorithm that finds the minimum number of insertions, deletions, and replacements to map from one string to another. This can be visualised in a confusion matrix where the off diagonals represent replacements from a predicted phone to a correct phone, the diagonal represents correct phone predictions, and insertions/deletions are tallied in an additional column. This is extremely powerful as the confusion matrix can be used to summarise an entire corpus of predictions; summarising the errors present in a model.

Figure 17 presents the described confusion matrix, the key regions of interest are the correct predictions represented in the main diagonal, with the phone `ch` being predicted correctly the most frequently and `p` being miss-classified as `sil` which is understandable since both `p` and `sil` denote *stops*. However the most frequent operation overall was inserting the stops `p`, suggesting this is the key source of error in our experimentation.

A more useful visualisation would be a normalised confusion matrix where each row sum's to one (Figure 18), so can be thought of as a probability distribution over operations. This reveals that for the majority of phones the most frequent scenario is a correct prediction (indicated by the strong diagonal), a deletion of the prediction (the final column) or to a lesser degree, an insertion of the phone (in the penultimate column). Otherwise the most frequent mistakes are mistaking an '`ih`' for an '`m`'. And the miss-classification of `ih`, `sil` and `t` as `k`.

10 Conclusion

To conclude, numerous combinations of parameters as well as regularisation, optimisation, and architectural configurations have been evaluated. Overall the optimal architecture was found to be a 3 layer LSTM with a 512-element hidden state whose full set of parameters have been summarised in table 2. This model achieves an accuracy of 82.2%, a 8.32% improvement from the baseline, however this came at a cost of nearly 100 times more parameters (15M vs 167K), and over double the training time.

Argument	value
`--batch_size`	4
`--num_epochs`	40
`--dropout`	0.1
`--value_clipping`	1
`--optimiser`	Adam
`--lr`	0.0007
`--num_layers`	3
`--model_dims`	512

Table 2: Optimal configuration parameters

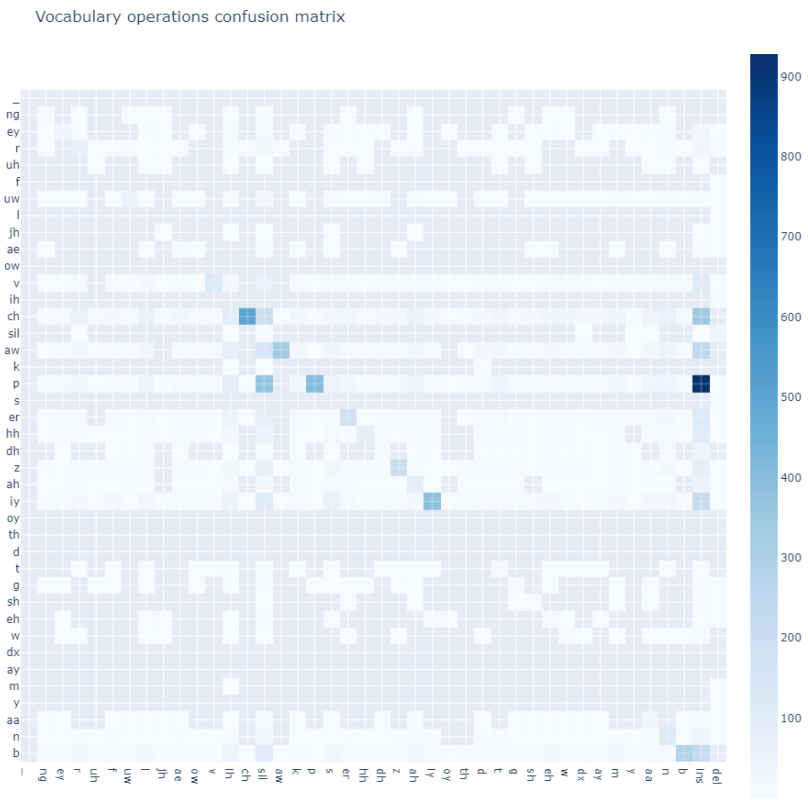


Figure 17: Confusion matrix with absolute counts

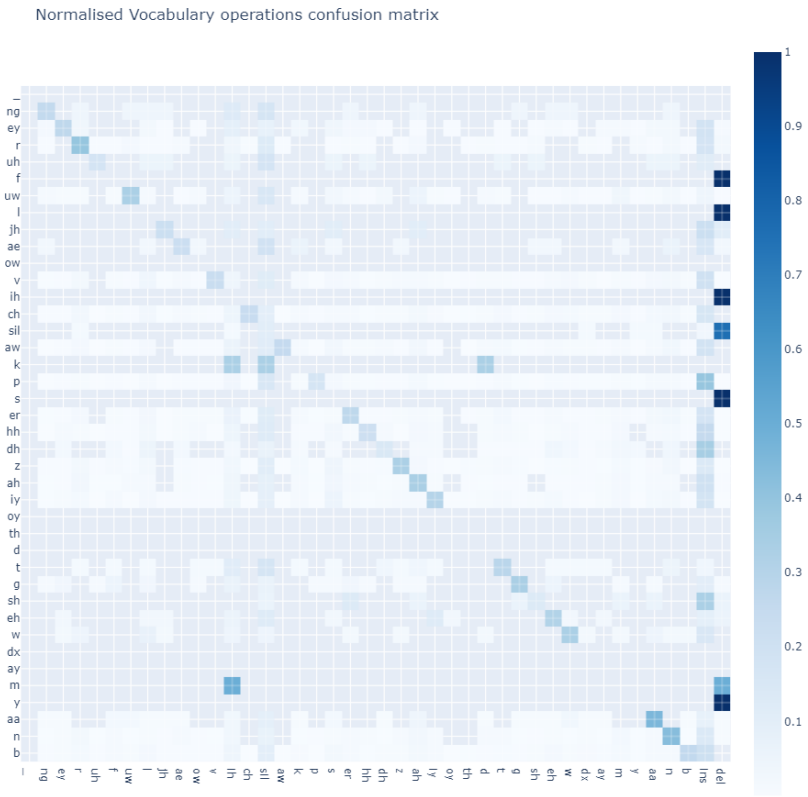


Figure 18: Normalised Confusion matrix

References

[1] Haytham M. Fayek. *Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between*. 2016. URL: https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html.

[2] Alex Graves et al. "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks". In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2006, pp. 369–376. ISBN: 1595933832. DOI: 10.1145/1143844.1143891. URL: https://doi.org/10.1145/1143844.1143891.

[3] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *Levenshtein Distance: Information Theory, Computer Science, String (Computer Science), String Metric, Damerau?Levenshtein Distance, Spell Checker, Hamming Distance*. Alpha Press, 2009. ISBN: 6130216904.