Speech Task № 4

Vladyslav Bondarenko, University of Sheffield May 31, 2020

Instructions for setting up and running the solution are provided in the README.md file

Training

Solution has been developed using *TensorFlow2* [1] package with *Keras* API. The data has been split into train, development and test set according to file names. Cepstral Mean Normalization has been performed on each of the files separately to reduce varying channel effects occurring in the audio data.

Models

Model Name	Layers		Non-Linearity	Regularization		
	1	2		Dropout	L1	L2
FNN-1	Dense(2000)	-	ReLU	0.5	$1 * 10^{-4}$	$1 * 10^{-4}$
FNN-2	Dense(2000)	Dense(2000)	ReLU	0.5	$1 * 10^{-5}$	0.01
LSTM	LSTM(300)	Dense(1000)	ReLU	0.5	$1 * 10^{-4}$	0.01

Table 1: Model details

A list of tested model architectures is provided in Tab.1 where **FNN** stand for a Feed Forward Network with 1 and 2 layers. The **Dense** layer is a fully-connected layer with the number of neurons inside the brackets. **ReLU** activation function has been used for all of the hidden layers as it has been shown [2] to have excellent convergence properties and prevent gradient saturation. Regularisation will be discussed in more detail further, but it is worth noting that a dropout layer has been added between each of the hidden layers, input weight L1 and L2 regularisation for Dense layers and same recurrent regulariser for LSTM layers.

Objective Function

Weighted Binary Cross-entropy has been used along with the **sigmoid** activation function in the output layer. Sigmoid activation returns a single probability denoting confidence level for the positive class and is commonly used for binary classification task along with the chosen objective function. Given the probability y_p returned by sigmoid, binary cross-entropy loss can be expressed as:

$$L(y_t, y_p) = -\frac{1}{N}\sum_{i=0}^{N} y_t log(y_p)w + (1 - y_t)log(1 - y_p)$$

Where term **w** reduces the impact of imbalanced data and is chosen to be proportional to the percentage of training and development data that belongs to the positive class. This loss function essentially measures how far away given probability is from a true value y_t (which is either 0 or 1) and then averages for the **N** data points in the batch.

Optimisation

a) Default TensorFlow initialisation has been used - **Glorot Uniform**. Weights are drawn from the uniform distribution with limits determined by the size of the previous and following layers.

b) **Adam** optimiser [3] is used to adjust the weights during training. It is an adaptive learning rate method which computes the learning rate for each of the parameters separately using first and second moments of the gradient. Its main advantage is fast convergence which is particularly useful for this task due to limited time availability.

c) Bayesian optimisation [4] approach is used to determine optimal hyper-parameters quickly. It works by constructing a posterior distribution over the Neural Net where the final output is the metric we aim to maximise. It is much faster than conventional methods such as grid search as it establishes beliefs about which the next set of parameters could improve the metric and guides the search. A search has been performed over many different parameters, including learning rate, batch size, layer sizes, dropout rate, regularisation strength and few other discussed later in the report. All of the hyper-parameter tuning is done using the development set.

d) It has been found that a relatively high dropout rate (0.5) between hidden units resulted in a much better generalisation reducing the development error. Dropout shuts down a specified proportion of units during training, making the neural net less reliant on the availability of full information and more resilient towards unseen inputs. As well as that, small regularisation in each of the layers shown a great improvement in performance on development and test set. L1 and L2 regularisation are used where L2 was found to play a more significant role requiring a larger value.

e) Development/Validation set is used to ensure the model generalises well. This is done by calculating loss on the development set after each epoch, and if the change in loss from the previous step is smaller than the provided threshold, training stops. This is also often called early stopping and is particularly useful when training is stable throughout the epochs. In theory, it prevents over-fitting as validation loss would start increasing if the model over-fits. In practice, there are a few drawbacks such as false stopping, where validation loss starts to increase, but the model has not converged yet. There are also problems where loss could be improving, but other key metrics, such as accuracy get worth. The issue could be solved by considering all of the metrics or by using fixed epochs and finding the best number of epochs to train for. This, however, is unfeasible within the time constraints imposed by this task.

Efficient Data Use

Running a few experiments demonstrated that using all of the available data resulted in improved performance. This is at the cost of much longer run-time. For the hyper-parameter optimisation only 50% of the data has been used to speed up the process. Final training and validation used all of the data consistently demonstrating improved results from those obtained during optimisation. To ensure faster training, mini-batch gradient optimisation has been used. Some investigation showed that there is a trade-off between the batch size and performance demonstrated in Fig.1. From the figure, it is clear that setting the batch size to 2000 would result in optimal performance for the speed up achieved.

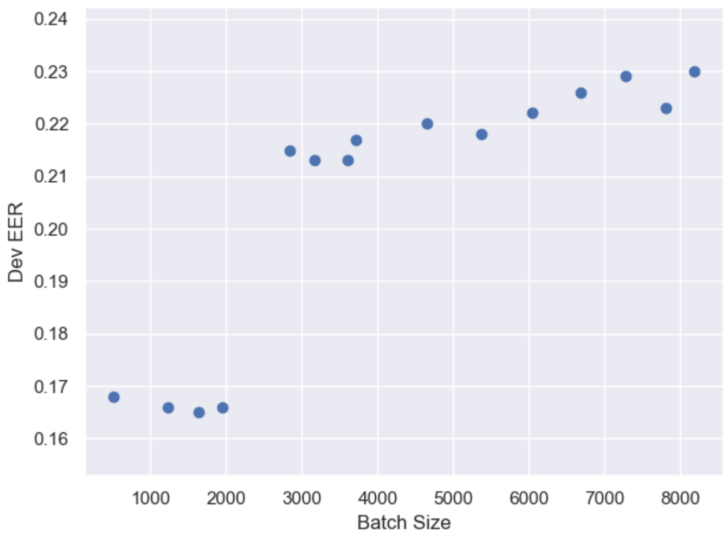

Figure 1: Batch size performance comparison (smaller EER is better)

Performance

As mentioned previously, final training and evaluation have been performed on the full data set. $5 * 10^6$ data points have been used for training, $2.3 * 10^6$ for validation and $1.7 * 10^6$ for testing. Final development and test loss is reported along with other results in Tab.2 while loss throughout training for different models can be observed in Fig. 2. There is a clear trend where both training and development loss gradually decreases as training continues. Training loss starts out slightly higher than development but decreases at a higher rate suggesting that there is some over-fitting. Eventually, development loss converges and start to increase at which point training is stopped by previously described early stopping criteria. The final difference between training and development loss suggests that there is a small amount of over-fitting, which could result in worth performance on the test set. Training for LSTM has been performed with a lower learning rate, and so the initial loss is much higher than that for FNN. However, they still converge to approximately the same loss and accuracy. It is worth noting that although the development and test loss for LSTM is higher than for FNN, this could be attributed to higher regularisation effect which also showcases why loss is not a reliable metric for performance comparison.

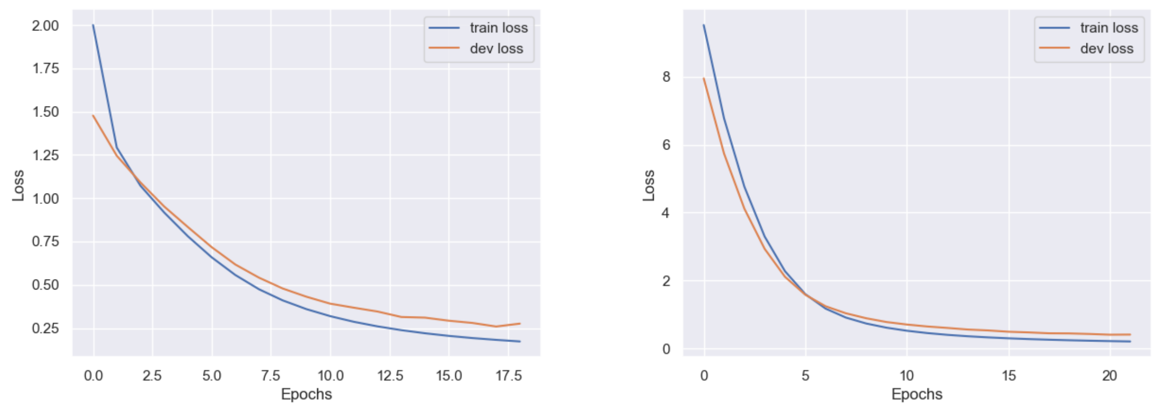

Figure 2: Loss through training for FNN (left) and LSTM (right)

Evaluation

Model Name	Development			Test		
	Loss	Accuracy	EER	Loss	Accuracy	EER
Baseline	-	81%	-	-	79%	-
FNN-1	0.276	94%	0.06	0.332	91%	0.11
FNN-2	0.204	94%	0.06	0.294	91%	0.12
LSTM	0.413	92%	0.14	0.397	91%	0.10

Table 2: Results for each of the models on development and test set

Results for all of the models reported in Tab.2 where baseline is a classifier that always predicts unvoiced class (0). From the accuracy score for the baseline the flow in using it as measurement becomes apparent straight away. Such a high accuracy for an unskilled model is not representative of true performance but rather class distribution in the data set. A simple accuracy measure has been adopted described in the equation below:

$$accuracy(y, \hat{y}) = \frac{\sum_{i=0}^{N} \mathbf{1}(y_i = \hat{y}_i)}{N}$$

Where **1** is the indicator function set to 1 if condition inside the brackets is true. Since the model return probabilities, there has to be a threshold that determines at which probability we assign the positive class to the prediction. A standard value to choose would be **0.5** with which all of the accuracies are reported. Nevertheless, in the operational environment, the threshold would be set according to either the EER threshold or other appropriate point on the DET curve.

Development and test DET curves for FNN-1 model are plotted in Fig. 3 with baseline denoted in dashed line and EER threshold marked with a red dot. The closer the curve to (0,0) point, the better is the performance of the model. Both curves demonstrate excellent performance of the system, and the difference in the two curves clearly highlights the over-fitting.

One of the critical issues of using DET for this task is that it disregards the performance of the negative class, which could be more critical for some applications. Equal Error Rate (EER) is computed at the point where False Positive Rate is equivalent to the False Negative Rate and provides a good standardised metric for model performance. It has a similar problem as DET where we assume that both types of error are as consequential, but for a more practical application, it could be more desirable to reduce one error by more than the other. Therefore, as mentioned previously, a different point on the DET curve could be picked that provides a good trade-off for the application needs.

Finally, comparing the results between different models, it can be observed that LSTM is performing slightly better than FNN. This is an unexpected result as LSTM is much more suited for the task and should improve the performance by a larger margin. Failure to do so could be attributed to less precise hyper-parameter tuning performed due to time constraints of training an LSTM. Another interesting result could be inferred from the confusion matrices produced for two models in Fig. 4. The type of errors two models are making differ and so potentially combining their predictions could improve the final results.

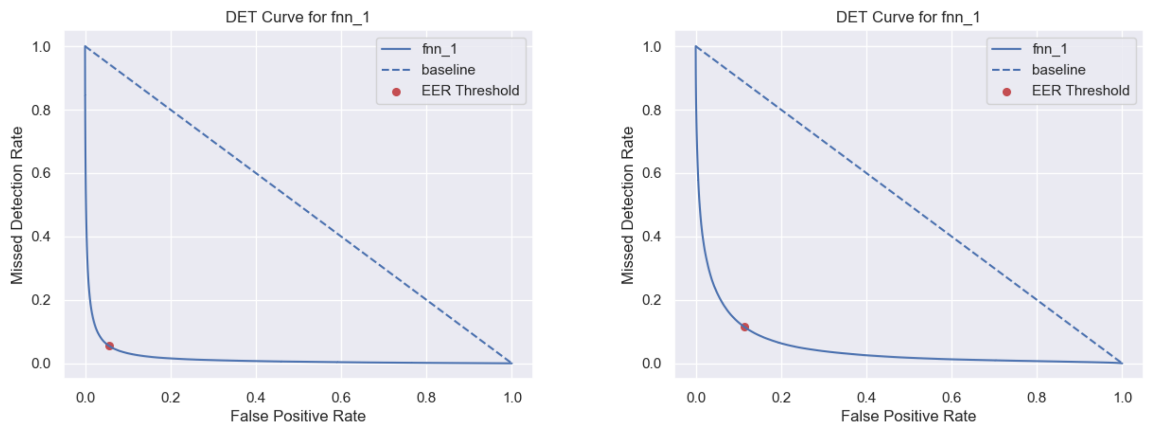

Figure 3: DET Curve for development (left) and test (right) sets with FNN-1

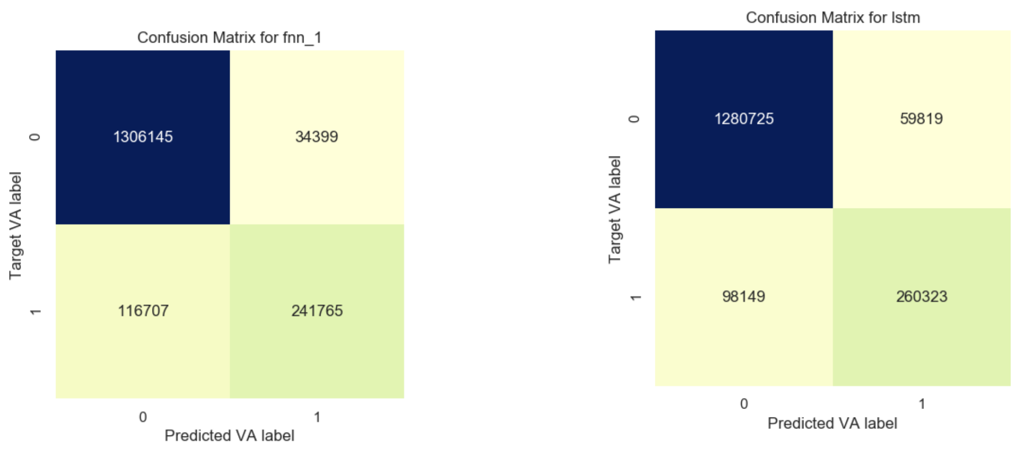

Figure 4: Test set confusion matrix comparisons for FNN and LSTM

References

[1] “TensorFlow.” https://www.tensorflow.org/.

[2] V. Nair and G. E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” tech. rep.

[3] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, International Conference on Learning Representations, ICLR, 12 2015.

[4] “fmfn/BayesianOptimization: A Python implementation of global optimization with gaussian processes..” https://github.com/fmfn/BayesianOptimization.