Selected Topics in Machine Learning Coursework 3

Gustavo Brito Rodríguez
Imperial College London
01198785
gb1216@ic.ac.uk

Alberto García Matachana
Imperial College London
01205901
ag4116@ic.ac.uk

Abstract

This report uses a data-set formed by sequences of hand movements for different actions to explore and compare LSTM and GRU RNN architectures, together with hyper-parameter tuning. It also uses the inherent structure of the data to explore a Hierarchical RNN model that builds on a simple GRU classifier. Finally, different techniques such as Ensemble Learning and adding convolutional layers were tested in order to improve classification accuracy.

1. Question 1: RNN Classifier

This question's objective is to describe and train a Recurrent Neural Network (RNN) Classifier, as well as carrying out a quantitative analysis on its performance through hyper-parameter tuning and varying data pre-processing, among others.

Amongst the classifier choices, were the vanilla RNN, the Gated Recurrent Unit (GRU) and the Long Short-Term Memory (LSTM) Unit. Both GRUs and LSTMs use gates to address the vanishing-gradient problem from which RNNs suffer [1, 6]. As a starting point, a vanilla RNN was built, but since no decrease in training was observed over 200 epochs, it was not included in subsequent analysis.

GRUs do not make use of a memory unit, which normally leads to lower computational and storage complexity and has shown to sometimes outperform LSTMs [3]. However, LSTMs allow for more control by the programmer due to the existence of an additional gate, and, therefore, another weight matrix.

Given that both GRUs and LSTMs have been able to produce state-of-the-art results in the existing literature [3, 11, 10, 8], both models were trained and analysed. Different experiments were carried out on each of them and their testing accuracy was compared, as well as their computational complexity.

1.1. Data Pre-processing

The data set used consisted of sequences of 3D hand poses (represented by 21 hand joints) performing a total of 45 hand actions [5]. These sequences had already been normalised, but, were of varying lengths. Each time-step , the input is in $\mathbb{R}^{21\times3}$; x, y and z positions for 21 hand sensors.

The author of [5] added in their in their GitHub repository [4] that in order to reproduce their LSTM results, 120 frames was fixed as the maximum frame length and that a sequence would be padded with zeros if it was shorter. The same approach was initially adopted in the experiments performed. 916 of the 1175 sequences were shorter than 120 frames. These were padded with zeros until reaching a dimensionality of 120×63; the remaining 259 were sliced to be only 120 frames long. This would later be varied to test its impact in 1.3.

Furthermore, in [5], different training-testing data partitions were tried and the one that yielded significantly better results was a 3:1 ratio of training to testing data. The training set was, therefore, made up of 950 sequences, and the test set was made up of 225 sequences.

1.2. Training a LSTM Classifier

The first classifier trained was the LSTM Classifier. The training batches were of size 50, and the testing batches, 75. The vanilla version consisted of a single recurrent LSTM layer with 100 hidden units, followed by a single fully connected layer which took as input 50 tensors in $\mathbb{R}^{120\times100}$ and output a tensor in $\mathbb{R}^{50\times45}$. To measure the performance of the classification system, the Cross-entropy loss was used. The optimiser used was Adam with a learning rate of 0.001 [7].

This vanilla architecture was trained for 400 epochs, which took 630s. This resulted in very slow convergence at 200 epochs and very unstable learning with great train loss and accuracy oscillations. The maximum accuracy achieved by this vanilla classifier was 56.56%.

The next experiment performed was varying the learning rate to attempt to fix the slow convergence - see Figure 1. The highest accuracy, 68.00%, was converged to after 100 epochs, without an increased instability in learning and no overfitting. The faster convergence meant the algorithm only had to be trained for 200 epochs. This training lasted 340s, but, the training time per epoch, did not change significantly between tests.

Following the change in learning rate, the next tests that were carried out, consisted of architectural changes, such as: increasing the number of hidden units, increasing the number of layers to 2 and 3 with varying dropout rates of 0.2 and 0.5, but, neither of these resulted in an increase in performance. Furthermore, every layer added increased the

training time drastically.

L2 Regularisation in the form of weight-decay was then added to the model. various weight-decay hyper-parameters were tested - see Figure 1. The value of 0.001 resulted in the best LSTM classifier performance after 200 epochs of training with a test accuracy of 72.44%. This resulted in the best trade-off between penalising large weights and allowing for proper learning.

The last change tested was varying the number of frames in each sequence. In 1.1, it was stated that the maximum number of frames per sequence would be limited to 120. Any sequence shorter than that would be padded with 0s and any sequence longer would be cut after the 120th frame. This was done following the pre-processing used in [5]. As can be seen in Figure 2, a wide range of frame sizes were tested, but 120 resulted in the best trade-off between padded 0s and sequence cuts.

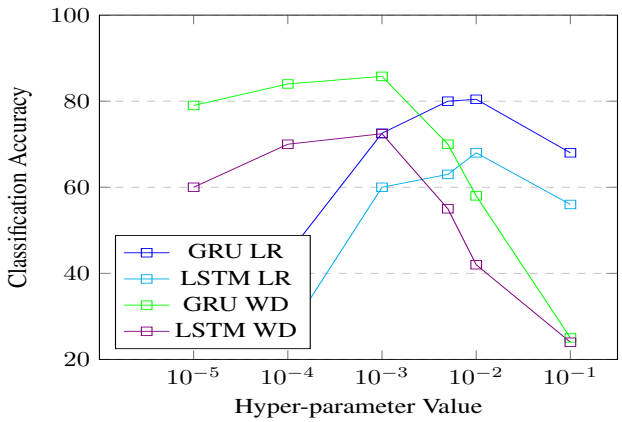

Figure 1: Varying Learning Rate (LR) and Weight Decay (WD) hyper-parameter values for both the LSTM and GRU architectures.

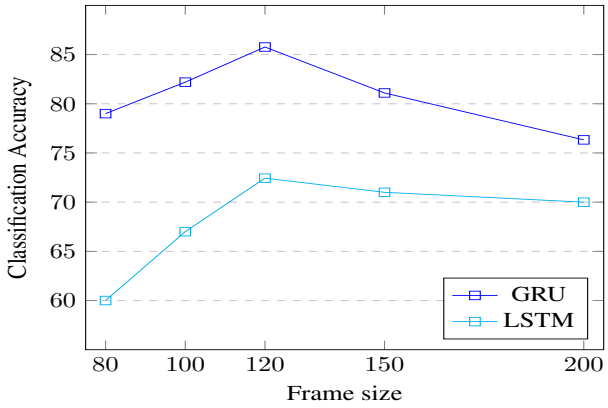

Figure 2: Accuracy vs Frame size for both models

1.3. Training a GRU Classifier

A vanilla GRU Classifier was then trained. This architecture was, a single recurrent GRU layer with 100 hidden units, followed by a fully connected layer. The optimiser used was Adam, and the learning rate was initially set to 0.001 [7]. This model was trained for 200 epochs with the same batch sizes as the LSTM classifier from 1.2. As expected, and mentioned at the start of 1, training was faster for the GRU than for the LSTM model. The vanilla GRU converged after only 100 epochs, unlike the 200 from the vanilla LSTM and with a lower training time per epoch. The vanilla GRU training lasted for 310s. Furthermore, the performance achieved with this vanilla architecture (81.00%) was far superior to the best-performing LSTM architecture (72.44%).

The next test performed, was the variation of the learning rate – see Figure 1. Once again, the best performance was achieved for a learning rate of 0.01, 83.24%. The rate of convergence, however, did not change greatly as convergence still occurred at around 100 epochs.

The next test was the addition of L2 regularisation trough weight-decay. A wide range of parameters were tested – refer to Figure 1. The weight-decay variation was preformed with the best-performing learning rate from the previous test. The performance continued to improve to 85.77%.

Nevertheless, one can see that although regularisation resulted in improvements in the accuracy for both the LSTM and GRU models, the largest change occurred for LSTM (4.44% improvement versus a 2.53% improvement for GRU). This is, due to the fact that the LSTM model has an additional gate for which the weights can be regularised, therefore making regularisation more impactful.

Similar to how it was done in 1, additional layers were added to the best performing GRU model to see how they would affect performance. 2 and 3 total recurrent layers were tested but resulted in lower performance, 84.89% and 83.22%, respectively, and higher training times, 892s and 2512s, respectively.

The frame size was varied - refer to Figure 2. As in 1.2, 120 resulted in the best frame size. Notwithstanding, the GRU had more fall-off for larger frame sizes because of the existence of a smaller number of gates.

The final test, was the variation of batch size with the architecture and parameters of the best performing GRU – 85.77%. Batch sizes of 25 and 90 were tested. For 25, the accuracy increased to 87.11% and training of 200 epochs took 330s. It was deemed that a 20s increase in training time for an accuracy increase of almost 2% was a worthy trade-off. In the opposite direction, a 90 batch size resulted in a 1.5% accuracy drop and a decrease in train time to 300s. 87.11% accuracy was the maximum achieved for all the classifiers trained in Question 1 and the Confusion Matrix for this architecture can be seen on Figure 3. As can be

seen, a sample failure would be the 'flip sponge' action, predicted 4 times as 'open wallet', and 'handshake' is a sample success.

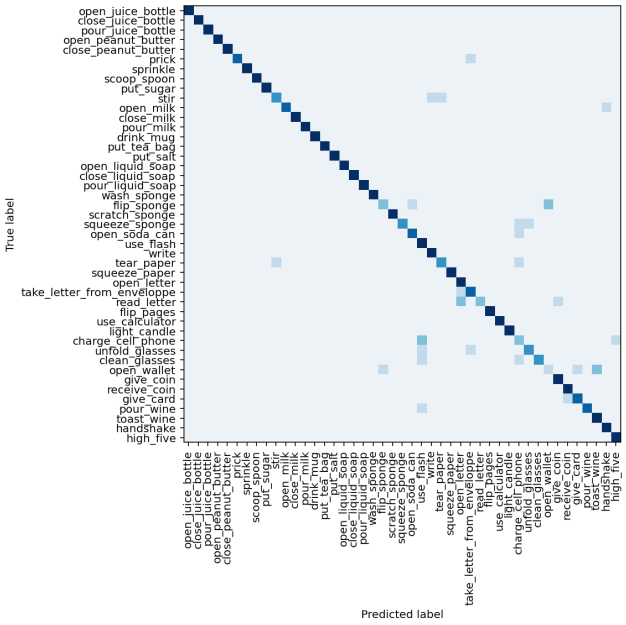

Figure 3: Confusion Matrix for best-performing GRU with 87.11% accuracy.

2. Question 2: Hierarchical RNN

The hand's structure can be used to build a Hierarchical RNN. In this section we proposed two different models, one with two recurrent steps and one with three recurrent steps.

2.1. Two-step Model

The input to the network on part 1.1 was in $\mathbb{R}^{21\times3}$ per time step; 4 sensors per finger and 1 on the wrist. In this proposed network, the first layer consists of 5 RNNs (one per finger), which take in an input in $\mathbb{R}^{120\times5\times3}$; the 4 sensors in each finger plus the wrist sensor. In the first recurrent step, each RNN has 30 hidden units, therefore, the output of each unit is in $\mathbb{R}^{120\times30}$. Then, all recurrent outputs go through a merge step which outputs in $\mathbb{R}^{120\times150}$. This tensor is fed into the second recurrent step which outputs in $\mathbb{R}^{120\times100}$. Ultimately, the tensors go through a fully connected layer for classification purposes.

The motivation for this model is the physical significance of the movement of a whole finger as single finger motions are possible within a larger 'action' such as the movement of index and thumb in a 'sprinkle'.

2.2. Three-Step Model

In this model, an additional recurrent step is added between the two steps described in 2.1. This additional stage consists of 4 RNNs that will take as input the output of any one finger from the first recurrent stage combined with the thumb output, i.e., $\mathbb{R}^{60}$ per time step. The input and output for each of the four RNNs on this second layer are in $\mathbb{R}^{120\times60}$ and $\mathbb{R}^{120\times100}$, respectively. The last recurrent step takes an input in $\mathbb{R}^{120\times400}$, after the second merge step, and outputs in $\mathbb{R}^{120\times100}$. Lastly, the tensor in $\mathbb{R}^{120\times100}$ goes through a fully connected layer which classifies the action.

The motivation behind this model, is the significance of the thumb in every action and how the combination of the thumb's movement with every other finger's correlates to the whole action being performed.

2.3. Training

When training, it was important to decide what type of RNN would be used on each step of the hierarchical model. Initially, three different versions of each model were trained, one made up of vanilla RNNs, one made up of LSTMs and one made up of GRUs. The optimiser used was Adam with a learning rate of 0.01 [7]. L2 regularisation was added with a hyper-parameter value of 0.001. The training batch size was initially set to 50. A smaller batch size of 25 was later attempted but to no success as it greatly increased training time and, unlike in 1.3, led to no increase in performance.

When the networks were built with vanilla RNNs, the models suffered from vanishing gradient as there was no reduction in the training loss over time and, therefore, no learning was taking place.

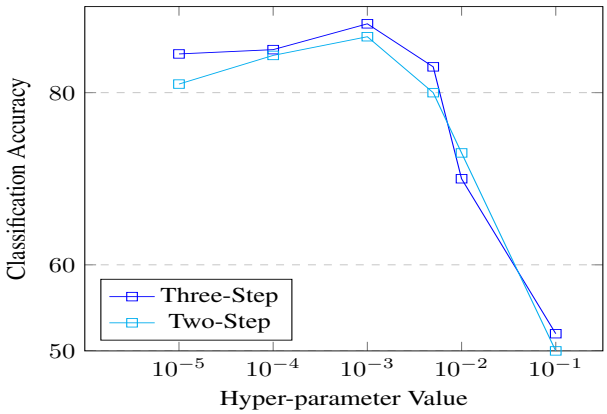

Figure 4: Varying Weight Decay hyper-parameter values for both the Two-Step and Three-Step architectures.

Using LSTMs was not a fruitful solution either as, given their increased complexity, running each training epoch took an unreasonably long time.

The GRUs were the only method that led to fruitful results, with the Two-Step and Three-Step model achieving a 86.50% and 88.00% classification accuracy, respectively. The latter is higher than that of the best result from Question 1 and was achieved after convergence at 200 epochs of training. The training for these two models, lasted 723 and

1151 seconds, respectively.

The next step was to add an average pooling layer over temporal domain between the last recurrent step and the fully connected layer to accumulate the responses for a given sample over time. Kernel sizes of 2, 4 and 8 were tried, but all resulted in a decrease in performance.

Another attempted change, was that of changing the GRU in the last step for an LSTM. This did not improve performance either and led to a a large increase in computational complexity as training it took 1506 seconds (in comparison to the 1151 seconds of the full GRU network).

The following test was the hyper-tuning of the weight-decay hyper-parameter – refer to Figure 4. The best weight-decay hyper-parameter value, was 0.0001, the default value for each model. Too high a value, resulted in a sharp decline in performance as it restricted the algorithm's ability to learn. On the other hand, too low a value did not result in sufficient generalisation.

The confusion matrix for the best performing classifier in Question 2 - Three-Step Model - can be found in Figure 5. 'Open juice bottle' is now a success on this algorithm and 'open wallet' is still a sample failure, despite the increased accuracy.

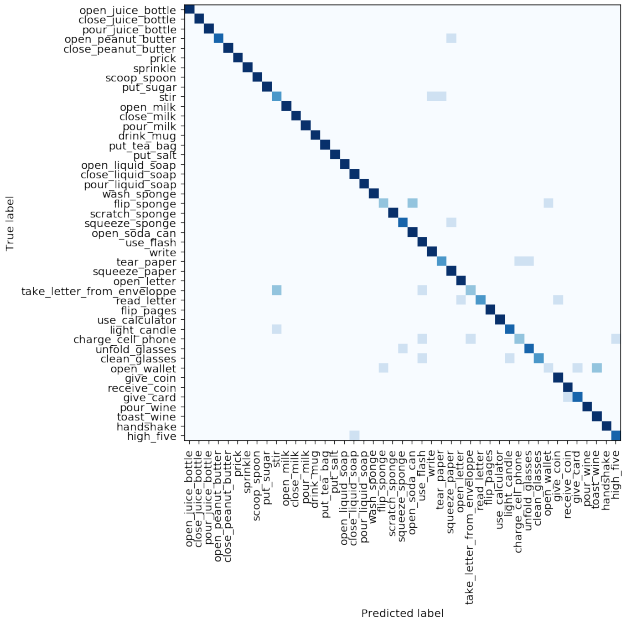

Figure 5: Confusion Matrix for best-performing HRNN with 88.00% accuracy.

3. Question 3: Improving Accuracy

This section's objective is to improve on the 88.00% accuracy from the HRNN from section 2. Two methods to do this will be proposed. Ensemble Learning and the addition of convolutional layers before the RNN Classifier.

3.1. Ensemble Learning Approach

In the ensemble learning approach, a varying number of classifiers were trained and the prediction was decided through a certain combination for their predictions. This method has been used in state-of-the art RNN algorithms to improve accuracy [2]. Unlike in [2], which develops a method to apply different weighting to the classifiers in the ensemble, the proposed method performed majority voting of equally weighted classifiers.

3.2. Deep Convolutional and GRU RNN Approach

The second approach is inspired from the work in [9], where Ordoñez *et al.* propose the use of CNNs to perform feature extraction and provide abstract representations of the input sensor data in feature maps, whose output is then fed to a LSTM to capture the temporal dynamics of the activation of the feature maps for wearable activity recognition.

3.3. Training

Three different ensembles were built, they were of size: 5, 10 and 20. Training classifiers increases training duration linearly as there is no depth added, but instead, different classifiers that are trained independently. The RNN used in these was the best-performing Three-Step HRNN from 2.2, with which a classification accuracy of 88% was achieved.

The ensembles of sizes 5, 10 and 20 produced 87.23%, 91.13% and 90.76% accuracies, respectively. For size 5, accuracy actually decreased. This is probably due to the fact that the number of classifiers was not enough to improve generalisation. 10 resulted in the best generalisation and achieved the highest performance.

Regarding the CNN-RNN aggregate model, the CNN input is in $\mathbb{R}^{120\times63}$ to effectively detect both spatial and temporal features among the signal components. The input would then go through three convolutional layers with kernels of size 5×5, 3×3 and 3×3. The input of every convolutional layer is properly padded so that no loss of resolution is determined from the convolutional operation. All convolutional layers were followed by Batch Normalisation and had 30% drop-out during training. The ReLU function is used as activation function within the CNN network. Pooling layers were not used in the CNN framework since according to [9] is not beneficial when the data sequence is required to be processed by recurrent layers. The output of the third convolutional layer is fed into the best performing GRU described in 1.3 with 100 hidden units. Finally, the recurrent output goes through a fully connected layer due to the multi-class classification task. The optimiser used throughout the whole CNN-RNN network was Adam with a learning rate of 0.01 [7] while the loss was calculated with the cross entropy function. This was the optimal architecture with a 92.34% accuracy after 400 epochs with a total training time of 2196s.

References

[1] K. Cho, B. van Merriënboer, C. Gulcehere, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation, September 2014.

[2] J. Y. Choi and B. Lee. Combining lstm network ensemble via adaptive weighting for improved time series forecasting, August 2018.

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, December 2014.

[4] G. Garcia-Hernando. Baseline lstm for action recognition #7. `https://github.com/guiggh/hand_pose_action/issues/7`. Accessed: 2020-03-16.

[5] G. Garcia-Hernando, S. Yuan, Seungryul, and T.-K. Kim. First-person hand action benchmark with rgb-d videos and 3d hand pose annotations, April 2018.

[6] S. Hochreiter and J. Schmidhuber. Long short-term memory, December 1997.

[7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, December 2014.

[8] Q. Li, A. Zhang, L. Pengcheng, J. Li, and C. Li. A novel csi feedback approach for massive mimo using lstm-attention cnn, 2020.

[9] F. J. Ordóñez and D. Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition, January 2016.

[10] S. Saadatnejad, M. Oveisi, and M. Hashemi. Lstm-based ecg classification for continuous monitoring on personal wearable devices, 2020.

[11] Z. Zhao, W. Chen, X. Wu, P. C. Y, and J. Liu. Lstm network: a deep learning approach for short-term traffic forecast, 2017.