CW3: Recurrent Neural Networks

Kevin Yu - kky16@ic.ac.uk - 01185891 Joe Gardner - jog116@ic.ac.uk - 01232601

1. Introduction

Using a subset of data provided from the First-Person Hand Action Benchmark[2], the coursework aims to classify hand actions with the aid of recurrent neural networks (RNN).

1.1. Dataset

The dataset is a dictionary of training data (x_train), testing data (x_test), their respective labels (y_train, y_test), and the type of action (actions) each label corresponds to. x_train consists of the hand pose information for 950 recordings of different hand actions performed. x_test is made from 225 such recordings (specifically 5 of each action) creating a train-test split of 38:9. y_train and y_test label the training and testing data as values from 1 to 45. actions then defines the type of movement for each of the 45 labels (e.g. the first label corresponds to open_peanut_butter in the actions array which evidently shows label 1 defines the action unscrewing a jar of peanut butter).

Each recording given is of varied sequence length (temporal) of 21 hand keypoints in 3 dimensional space (normalised). The keypoints are interpreted as the joints of the hand as shown in Figure 2 of the coursework outline[4] and when portrayed along the hand keypoints axis are ordered as set out in [4].

1.2. Preprocessing

The training data is an ndarray of length 950 comprising of different sized sequences of form $[N, 63]$ where N is the sequence length or number of time steps. Since, the recording of the sequences were made in the manner of 21 samples for its hand keypoints and in 3 channels to designate its location in 3D space, the sequences would be more faithful if reshaped to have dimensions $[N, 21, 3]$ where they denote the sequence length, samples and features respectively. In order, for a network to support multiple size inputs the sequence length must be kept constant.

Zero padding achieves this by adding 0s to the end of each time domain signal until it reaches the maximum sequence length of the dataset (max_len=1151). For this use case, the value -99 was used instead as 0 appears within the dataset. For a network to recognise the padding, a Masking layer must be introduced at the beginning of any model. Masking assigns logicals to the dataset for future layers to be able to determine which timesteps to skip/ignore in a sequence.

Zero padding was chosen as it maintains all information provided from the dataset in comparison to other methods, such as truncating or resampling. Padding maximizes the information retrieval potential by adding to the dataset instead of removing from it. Although, padding is generally favoured and is used for this coursework, it is computationally expensive in memory and speed if max_len is high and does not guarantee significantly better results as the lighter-weight methods. Training the network with a variable input shape by either executing a loop to fit the model multiple times or utilising fit_generator were also avoided as this limits the training to a batch size of 1.

Ultimately, it allows for the entire sequence set to be stored in a tensor of shape $[950, 1151, 21, 3]$ where the dimensions describe the number of sequences, sequence length, samples and features.

2. RNN Classifier

A RNN classifier was built as a sequential model with first a masking layer to recognise the incoming padded inputs as stated previously. Secondly, the RNN layer is added to extract the features into a output space of dimensionality num_units. The output is then fed into a dropout layer to prevent overfitting and then to a fully connected layer with a softmax activation of size equal to the number of classes($= 45$).

Considering the RNN classifer, it is unnecessary to divide the data into its 21 various samples as a single RNN can only accept a 2D input of $[\texttt{time\_steps}, \texttt{features}]$. Hence, the data is reverted back to its $[\texttt{max\_len}, 63]$ format where the samples and its channels are undistinguished from one another. The labels were also adjusted such that their range spanned from 0 to 44 to match its indexing and remove confusion for the fully connected layer of the classifer.

The default hyperparameters used for all subsequent testing were: $[\texttt{batch\_size} = 50, \texttt{num\_units} = 128, \texttt{lr} = 0.001, \texttt{epochs} = 50, \texttt{optimiser} = \texttt{adam}]$

2.1. Architecture

Three different architecture choices for the RNN layer of the classifier were tested: fully-connected RNN, gate recurrent unit (GRU) and long short-term memory (LSTM).

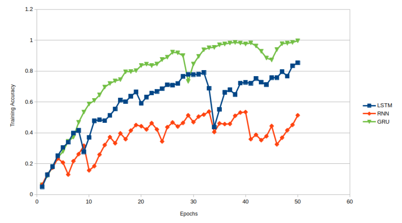


Figure 1: Training comparison of different architectures

Table 1: Testing accuracies

Architecture	Testing accuracy
SimpleRNN	0.3311
GRU	0.5737
LSTM	0.5600

LSTM and GRU can be seen to outperform SimpleRNN layers as more control over its cell is given in the form of gates to select what is or is not stored in its 'memory'. LSTMs have an input, forget, output gate while GRUs have two gathers a reset gate and a update gate (which couples the input and forget gates). This lower complexity means GRUs are more efficient than LSTMs and perform better for smaller datasets [1]. Therefore, GRUs are selected as the best architecture for the following application and will be used moving forward.

2.2. Hidden Units and Layers

The number of hidden units (num_units) is responsible for the width of a recurrent layer. It extracts the features from the sequences inputted and outputs the hidden state of the units.

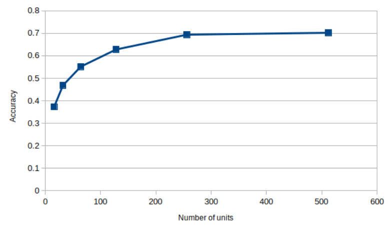


Figure 2: Number of hidden units variation

It can be seen that increasing the number of units leads to a logarithmic increase in classification accuracy. After

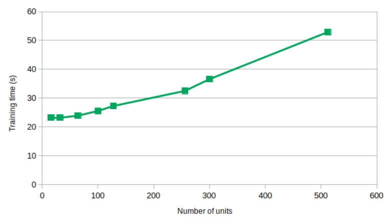


Figure 3: Training times against GRU size

num_units $= 256$, there are diminishing returns in accuracy as the model has reached its potential for the total data given while having increasing effects in computational speed and memory.

On the other hand, the number of hidden layers (depth) adjusts the depth of the network. By passing the output sequence from one RNN layer to another it creates a vertical hierarchy down the chain of layers, extracting features from the features extracted beforehand.

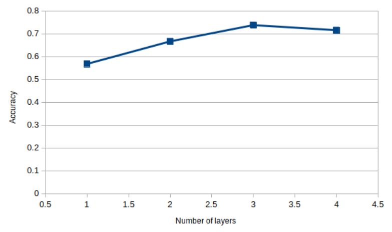


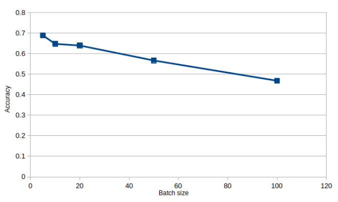
Figure 4: Accuracy for different numbers of layers

As can be viewed above, an increase in the number of hidden layers improves accuracy; however, when depth $= 4$ the accuracy begins to decrease. The training time of a network with 3 hidden layers and otherwise default parameters takes 86 seconds, significantly longer than that of a single layer network which can reach similar accuracies at a much faster training speed (25 seconds). Therefore, the compromise was made to only use a single layer network.

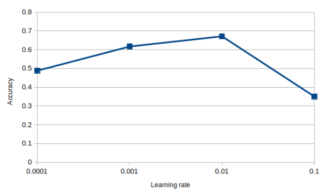
2.3. Hyperparameters

In addition when fitting the model, the hyperparameters specifically altering the learning rate (lr) and batch size (batch_size) where trialled. The learning rate determines the scale of the gradient descent after each iteration and optimising it allows for the minimum loss without oscillation (large learning rate) or prolonged time for convergence (small learning rate). The batch size defines the number of samples to be propagated through the network for each iteration. Hence, a balance needs to be found between the computationally expensive large batches and volatile 'directionless' small batch sizes[1].

[1]This is not to say training time using small batches are lesser than larger batches, only for each iteration. Typically, at both extremes the train-



(a) Batch size variation



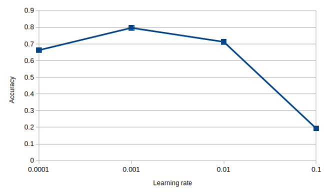
(b) Learning rate variation

Figure 5: Hyperparameter variation

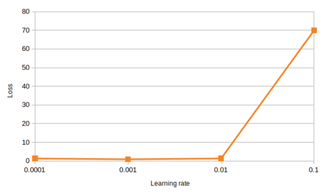
Evidently, the accuracy decreases in an approximately linear relationship with increase in batch size. This can be explained as for the same learning rate smaller batch sizes means proportionally greater number of weight updates. Although, updates are more frequent they are less 'accurate' due to the existence of 'noise' introduced by small batches having less confidence. Counterintuitively, this noise can help produce better results for certain datasets and/or models as seen here by providing a sort of regularisation to prevent the network from overfitting. This supports the empirical evidence gathered of larger batch sizes leading to poorer generalisation [3]. From the paper, the superior performance of smaller batch sizes could also be attributed to larger batches wanting to converge at sharp minimizers of the training function.

2.4. Evaluation

Using the results of the hyperparameter sweeps, the batch size and number of hidden units were set to 5 and 256 respectively. Using these values, the learning rate was once again varied across different possible values. It can be seen in Figure 6a that 0.001 now produces the best results, giving a testing accuracy of 79.57% after 50 epochs with Adam. Using this classifier, a confusion matrix was produced, shown in Figure 7.



(a) Final learning rate sweep



(b) Loss against final learning rate

Figure 6: Accuracy and loss from sweeping

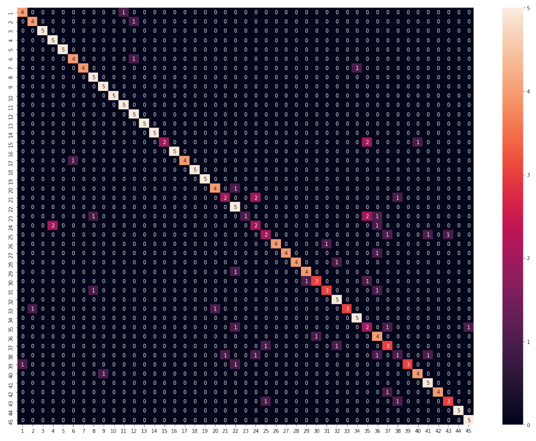


Figure 7: Confusion matrix plot

3. Hierarchical RNN

Hierarchical RNN (HRNN) are a stack of multiple RNN models with the objective to preserve hierarchical structures and learn the temporal dependencies in the sequential data at the many levels. For example in text, this could mean characters at the lowest/first level of the structure, words at the next level, to sentences, paragraphs etc.

Hence, focus is placed on the hand keypoint samples of the data as they illustrate the underlying representations in the dataset. The 21 samples as described before are ordered by starting with the wrist, then the knuckles at each of the fingers from thumb to pinky, and subsequently navigating upwards through the other joints in fingers until it reaches the tip [4]. The respective samples describing each finger are gathered and merged together with the wrist sample via concatenation along the features axis to implement the intermediate finger representations. Each of the five fingers, now with shape $[\texttt{max\_len}, 15]$, are individually passed through their own independent RNN layers to learn the output sequences at the 'finger' level of the structure. This creates a hierarchical structure from joints to fingers. The sequences are then again merged to form a representation of the hand as a whole $[\texttt{max\_len}, \texttt{num\_units1} * 5]$ and the final level of the structure. From this point, it follows a similar construction of the standard RNN classifier with one RNN layer of size `num_units2` to feed the final states into a dropout layer and finally to a `Dense` layer with softmax activation to classify the data.

The default hyperparameters used for all subsequent testing were: $[\texttt{batch\_size} = 50, \texttt{num\_units1} = 16, \texttt{num\_units2} = 10 * \texttt{num\_units1}, \texttt{lr} = 0.001, \texttt{epochs} = 50, \texttt{optimiser} = \texttt{adam}]$

3.1. Optimisation

The tuning of the architecture and hyperparameters of the network from Section 2 were similarly applied to the HRNN described above.

As shown, the accuracy of the model improves logarithmically after increasing the number of units for the GRUs

ing time is similar and the global minimum lies somewhere inbetween.

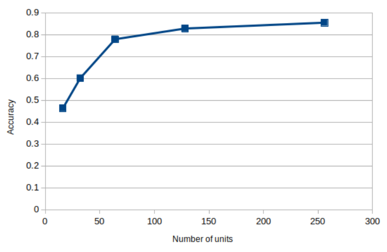
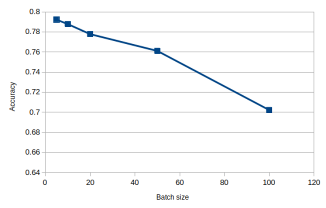
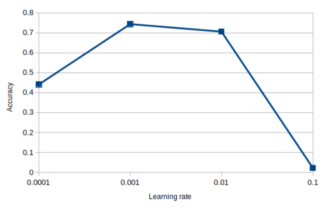


Figure 8: Number of hidden units variation

in the system. It can be seen the plot achieves better performance to its basic RNN counterpart due to the input data being shared among five recurrence layers and thus needing less capacity to determine the features necessary.



(a) Batch size variation



(b) Learning rate variation

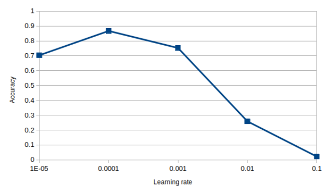
Figure 9: Hyperparameter variation

Similarly, the hyperparameters of learning rate and batch size have responses akin to the classifier from Section 2. The plot of the distinction in accuracy for batch sizes reveals that the same `batch_size` value can be used for the HRNN. Conversely, the learning rate is lowered due to the higher model complexity so each iteration of the loss should be less confident as being backpropagated through a single RNN layer. Figure 9b demonstrates this notion, having slightly better accuracy at `lr` $= 0.0001$ than the learning rate previously used.

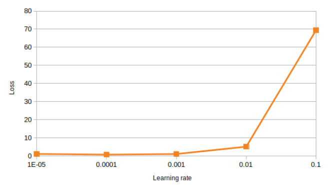
3.2. Evaluation

As demonstrated, HRNN follows the same behaviour as the simpler RNN classifier, when tuning hyperparameters and modifying architecture, but has consistently superior performance. The comparisons between the RNN and HRNN in the previous section shows clearly a benefit in the structure of the model of the latter. HRNN introduces the inherent representations specific to the data into the structure of the network to be trained. As reported before, this is done by merging the samples that constitute such representations as features and then passing them into their own recurrence layers. Hence, representations at various stages of the hierarchical structure add further information to the network as the model is customised to the underlying characterisations (e.g. fingers) of its dataset.

Unlike a HRNN, a basic RNN classifier passes all samples in no particular order or representation into a single recurrence layer as features. The unsorted natures of the features and the the dependence among them makes it difficult for the network to understand the relationship and patterns in the data and; hence, why such approach is often called a naive RNN.



(a) Final learning rate sweep



(b) Loss against learning rate sweep

Figure 10: Accuracy and loss from sweeping

The parameters that gave the optimal solution were 128 hidden units, a batch size of 5, and a learning rate of 0.0001, the latter determined with a further sweep (Figure 10a) once the others were fixed. A testing accuracy of 86.67% was achieved after 50 epochs with Adam, as can also be seen in Figure 10a at the peak of the graph.

References

[1] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[2] G. Garcia-Hernando, S. Yuan, S. Baek, and T.-K. Kim. First-Person Hand Action Benchmark with RGB-D Videos and 3D Hand Pose Annotations. In *Proceedings of Computer Vision and Pattern Recognition*, 2018.

[3] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.

[4] T. K. Kim. *Selected Topics in Machine Learning: Coursework on Recurrent Neural Networks*.