Mini Project 2

Mehmet Ozan Baykan

EEE443, Fall 2023

1 Introduction

In this project, we implement a many-to-one RNN architecture for the multi-category classification of human movements given time-series data from movement sensors.

2 Network Architecture and Methods

2.1 Network Architecture

The defining characteristic of recurrent neural network (RNN) architectures is that it has feedback connections among layers. This may be in myriad ways, such as in echo-state networks or LSTMs. In our implementation, however, the feedback connections are between every consecutive layer, and the activation value of each layer is fed into the activation of the layer one step later via a linear transformation followed by a nonlinear function. As such, the RNN in fact approximates a dynamical system rather than a function in the mathematical sense. The equations that define the forward propagation on one layer of the network are:

$$A_t = tanh(W_{ih}X_t + W_{hh}A_{t-1}) \tag{1}$$

$$\hat{Y}_t = \sigma(W_{ho}A_t) \tag{2}$$

where:

- $t \in \{1, \ldots, T_x\}$: time step in the input data.
- $T_x \in \mathbb{N}$: the number of time steps in the input data.
- $A_0 = 0 \in \mathbb{R}^{D_{hid} x D_{hid}}$: the initial hidden state, as there are no recurrent connections into the first layer
- $A_t \in \mathbb{R}^{D_{hid} x D_{hid}}$: hidden state at time step t.
- $X_t \in \mathbb{R}^{D_{in} x m}$: input at time step t, where m is batch size
- $W_{ih} \in \mathbb{R}^{D_{hid} x D_{in}}$: weight matrix for input-to-hidden connections.

- $W_{hh} \in \mathbb{R}^{D_{hid} x D_{hid}}$: weight matrix for hidden-to-hidden connections.
- $W_{ho} \in \mathbb{R}^{D_{out} x D_{hid}}$: weight matrix for hidden-to-output connections.
- $\hat{Y}_t \in \mathbb{R}^{D_{out} x m}$: the prediction on the input data, m = batch size
- tanh: element-wise hyperbolic tangent function.
- σ: element-wise sigmoid activation function.

The error function is the multi-category cross entropy function, given by:

$$\mathrm{E}(Y^{(t)}, Y^{(t)}_{\text{pred}}) = -\sum_{i=1}^{6} \left(Y_i^{(t)} \log(Y^{(t)}_{\text{pred}})_i + (1 - Y_i) \log(1 - Y^{(t)}_{\text{pred}})_i \right) \qquad (3)$$

where:

- $Y^{(t)}$: the ground truth at time step t, with shape $(\text{batch size}, \text{output dimension}) = (6, m)$.
- $Y^{(t)}_{\text{pred}}$: the model's prediction at time step t, with shape $(\text{batch size}, \text{output dimension}) = (6, m)$.
- $Y_i^{(t)}$: the ground truth for category i at time step t.
- $Y^{(t)}_{\text{pred}_i}$: the predicted probability for category i at time step t.

2.2 Loss Function

The Multi-category Cross-Entropy Loss is the sum of the binary cross-entropy of each prediction per category at a given time step t. While the first term penalizes false negatives, the second term penalizes false positives. Therefore, the function penalizes any deviation from the one-hot vector of the ground truth, i.e the true distribution given the input sequence, and is minimized at $Y^{(t)} = Y^{(t)}_{pred}$.

Although the output of the sigmoid need not sum up to 1, hence does not represent a probability distribution, the effect of the multi-category loss is what we want to achieve.

2.3 Backpropagation Through Time

The equations for the backpropagation through time are given below:

$$
\begin{aligned}
\frac{\partial E}{\partial A^{(T_x)}} &= \widetilde{W}_{ho}^T \cdot (Y_{\text{pred}} - Y) \\
\frac{\partial E}{\partial W_{ho}} &= \frac{1}{m} (Y_{\text{pred}} - Y) \cdot (A_{ext}^{T_x})^T \\
\frac{\partial E}{\partial A^{(t)}} &= W_{hh}^T \cdot \left(\frac{\partial E}{\partial A^{(t+1)}} \cdot \left(1 - (A^{(t+1)})^2 \right) \right) \quad \text{for t} \in 1...T_x - 1 \\
\frac{\partial E}{\partial W_{ih}^{(t)}} &= \frac{1}{m} \left(\frac{\partial E}{\partial A^{(t)}} * \left(1 - (A^{(t)})^2 \right) \right) \cdot X^{(t)T} \quad \text{for t} \in 1...T_x \\
\frac{\partial E}{\partial W_{hh}^{(t)}} &= \frac{1}{m} \left(\frac{\partial E}{\partial A^{(t)}} * \left(1 - (A^{(t)})^2 \right) \right) \cdot A^{(t-1)T} \quad \text{for t} \in 2...T_x \\
\frac{\partial E}{\partial W_{ih}} &= \sum_{t=1}^{T_x} \frac{\partial E}{\partial W_{ih}^{(t)}} \\
\frac{\partial E}{\partial W_{hh}} &= \sum_{t=2}^{T_x} \frac{\partial E}{\partial W_{hh}^{(t)}}
\end{aligned}
$$

- $A_{ext}^{T_x}$ is the output activation at the final time step, with an added row of 1s for bias terms.
- $\frac{\partial E}{\partial W_{ho}}$ is the gradient of the output-to-hidden weight matrix with respect to the loss. As there is a single output, this is the only term that contributes to the update on W_ho
- $\widetilde{W}_{ho}$ is the output-to-hidden weight matrix with the bias terms - last column - omitted, since the bias terms do not have ascendents in the computational graph.
- $A^{(t)}$ is the hidden activation at time step $t \in 1...T_x$.
- $\frac{\partial E}{\partial W_{ih}^{(t)}}$ is the gradient of input-to-hidden weight matrix with respect to loss at time step $t \in 1...T_x$.
- $\frac{\partial E}{\partial W_{hh}^{(t)}}$ is the gradient of hidden-to-hidden weight matrix with respect to loss at time step $t \in 2...T_x$, as the first layer does not have feedback input.
- $\frac{\partial E}{\partial W_{ih}}$ is the gradient to be used for the update. As all layers contribute to the gradient of the input-to-hidden parameters, we update by their sum over time steps.
- $\frac{\partial E}{\partial W_{ih}}$ is, likewise, the gradient to be used for the update. As all layers, except the first, contribute to the gradient of the hidden-to-hidden parameters, we update by their sum over time steps, except for t=1.

The updates are defined by:

$$W_{ih} \leftarrow W_{ih} - \eta * \frac{\partial E}{\partial W_{ih}}$$
$$W_{hh} \leftarrow W_{hh} - \eta * \frac{\partial E}{\partial W_{hh}}$$
$$W_{ho} \leftarrow W_{ho} - \eta * \frac{\partial E}{\partial W_{ho}}$$

where $\eta \in \mathbb{R}$ is the learning rate parameter.

As suggested in the instructions, we use mini-batch training. For hyperparameters, again, as suggested in the instructions, we select from:

- $\eta \in \{0.05, 0.1\}$
- $\in \{10, 30\}$
- $_{hid} \in \{50, 100\}$

For the initialization of weight and bias parameters, as suggested, we use:

$$W \sim \mathcal{U}[-0.1, 0.1]$$

$$bias = 0$$

However, our experiments show that the initialization:

$$W \in \mathbb{R}^{NxM} \sim \mathcal{N}(0, \sqrt{\frac{2}{N+M}})$$

which modified version of the Xavier initialization, seems to perform better during training, hence, we use this initialization after the first round of training on the hyperparameter combinations.

3 Results

First, as described in part (a), we trained networks with the given parameter combinations and uniform parameter initialization. The results are given in Figure 1 below.

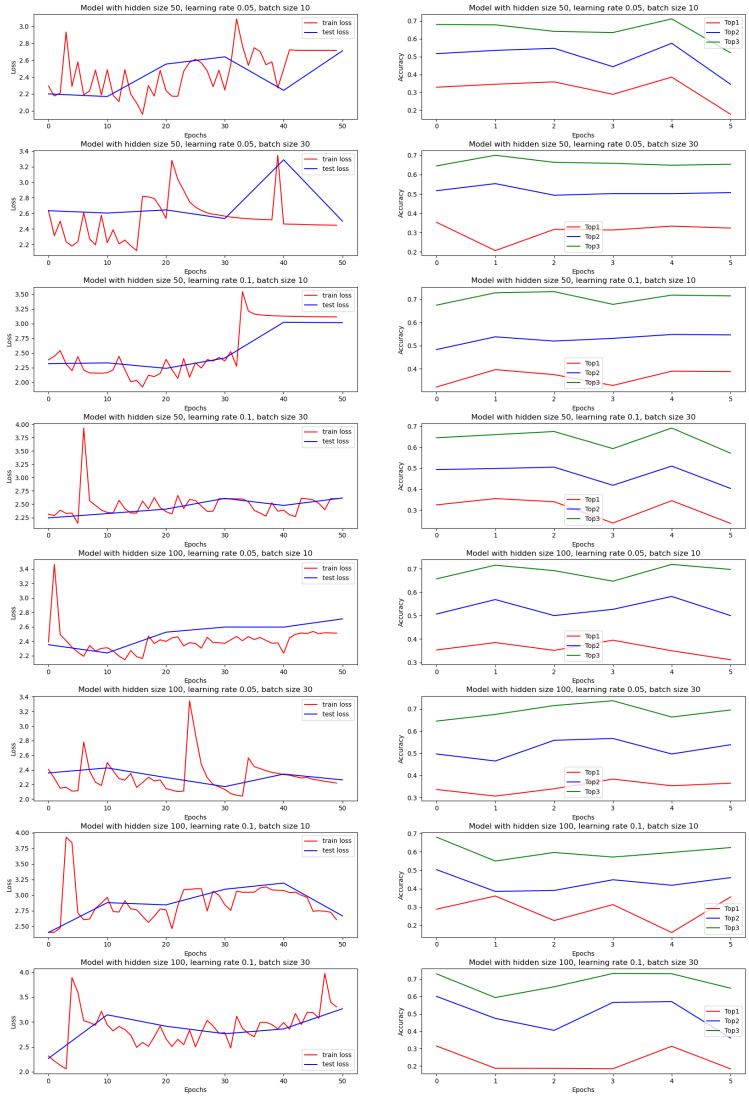

Figure 1: Training and test metrics for models with the given parameter configurations. The first column tracks the loss, where test loss is calculated only at every 10 epochs, and likewise, the second column is the accuracies on the test set after every 10 epochs.

Looking at these graphs, we see that the network not only learns very chaotically but also fails to learn consistently on the training set. For a better inspection, we look at the confusion matrices given in Figure 2 below.

Observe that although the accuracy on some intermediate epochs is acceptable given the model power and the data, the confusion matrices show that most of the models end up stuck in predicting one of the labels consistently. This problem may be attributed to a number of factors.

First, with the loss and accuracy plots in mind, we can say that the optimization tends to shoot out of local optima. Observing that this behavior is almost the same across different batch sizes, this is more likely caused by the learning rate, implying that a high learning rate combined with the tendency of the architecture to produce vanishing/exploding gradients due to the iteration of the chain rule through layers might cause the network to destabilize and fail to settle in optima.

Second, as evident in these results, the parameter space for this task is rather rugged, such that the parameter optimization is highly susceptible to variations across batches and numerical fluctuations in gradients.

Hence, initialization might be important in this context to allow the network to start without high bias and near poor local optima. Looking at these results, we may confidently say that the networks that get stuck predicting a single output are more flawed than others which make sensible errors, such as the network with parameters (50, 0.05, 30), (50, 0.1, 10), (100, 0.1, 10) and (100, 0.05, 30), which tend to confuse all standing-type motions with motion 4, the standing motion. This makes sense because it might be explained by the fact that, because the data for motion 4 is likely the closest to the average time-series for standing-type motions, the network overfits to this average, introducing bias towards the average of standing-type motions and failing to differentiate between these. Moreover, after seeing the confusion matrices, we might conclude that the batch size and learning rate parameter are interdependent and should be adjusted jointly, so that for smaller batches updates should be with smaller increments.

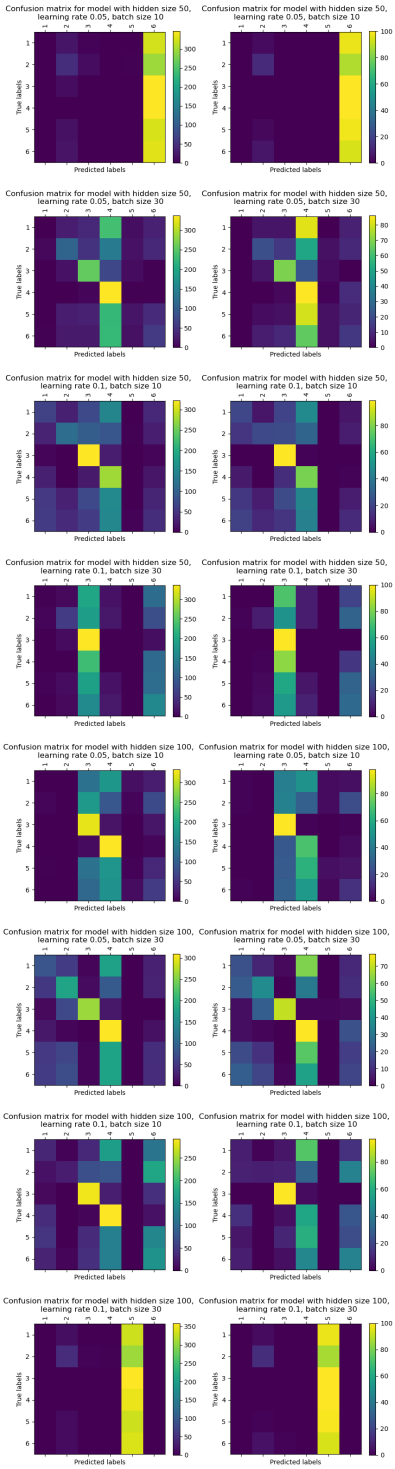

Figure 2: Training and test confusion matrices for models with the given parameter configurations. The first column contains confusion matrices on the training set, and likewise, the second column contains the confusion matrices on the test set.

With these evaluations in mind, we train another 3 networks with parameters (50, 0.01, 30), (100, 0.01, 30), (50, 0.01, 10), and with Xaiver initialization as described in section 2. As the performance did not clearly depend on the batch size, we chose to stick with batch size of 30, lowered the learning rate to better observe the incremental behavior and reduce the chaotic behavior of training, and aimed to test the effect of hidden size, and introduced the third network as a control for the effect of batch size. The metrics are in Figure 3 and confusion matrices are in Figure 4.

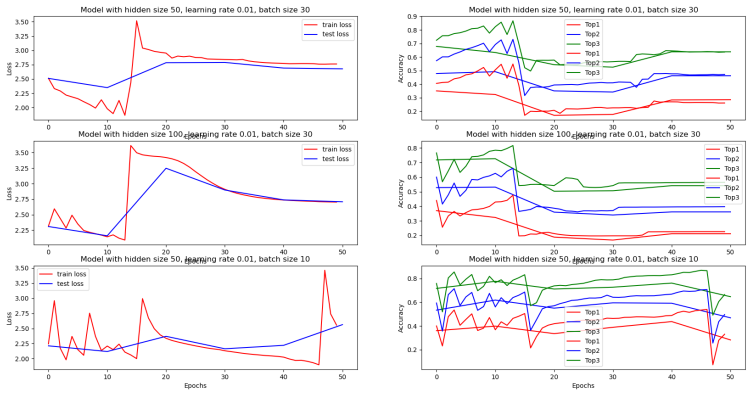

Figure 3: Training and test metrics for models with Xavier initialization and lower learning rate. The exact parameters are given in the titles. The first column tracks the loss, where test loss is calculated only at every 10 epochs, and likewise, the second column is the accuracies on the test set after every 10 epochs.

With respect to the metrics in Figure 3, we may conclude that Xaiver initialization helps the network to exhibit a more steady behavior toward local optima, but does not solve the problem of shooting out of optima, as is observed for all 3 networks around epoch 10. Reduction in the learning also does not help with this issue, but at least help to spread the epoch time between the large jumps in loss, hence will allow us to better take advantage of early stopping in the next step.

As for the confusion matrices, the network with 100 hidden layers tends to consistently output sitting motion in the end, which may be attributed to the fact that compared to the smaller hidden size of 50, this network is more susceptible to vanishing/exploding numerical values due to the increased number of parameters and numerical operations which may amplify numerical convergence to zero or infinity.

As a last point of comparison between these models, we point out that the best accuracies on the test set achieved around epoch 15 are highest in the

networks with hidden size 50, and it is apparent that these networks perform better even after drastically moving away from optima in the parameter space. Hence, we select these models to continue with training with early stopping to extract the best-performing parameters before the model jumps abruptly out of local optima.

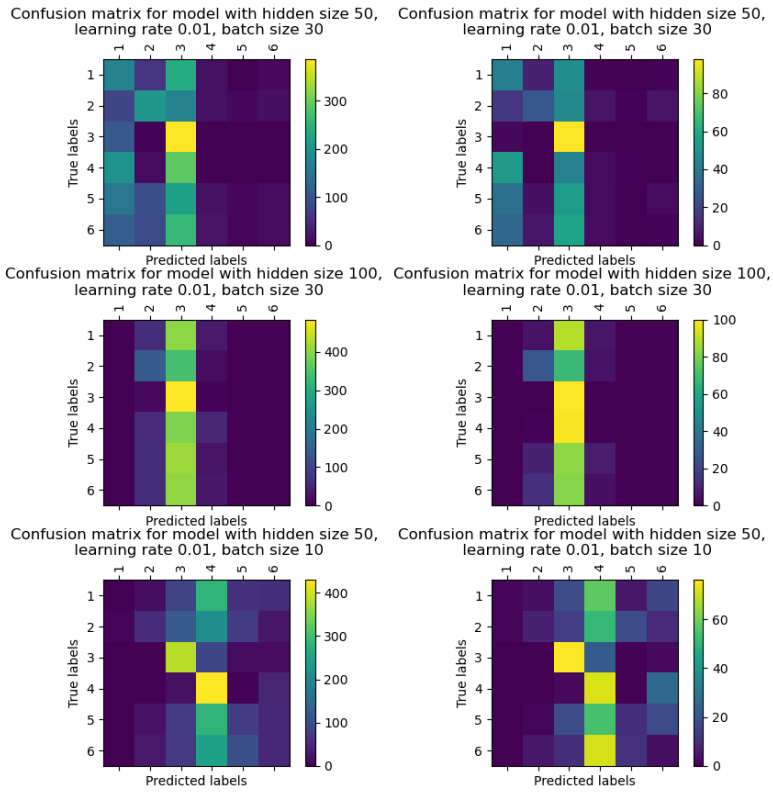

Figure 4: Training and test confusion matrices for models with Xavier initialization and lower learning rate. The exact parameters are given in the titles. The first column contains confusion matrices on the training set, and likewise, the second column contains the confusion matrices on the test set.

In Figure 5 and Figure 6, we see the results obtained from training the best-performing networks among the Xaiver-initialized networks, chosen for the reasons described in the abovementioned discussion. The networks are trained using early stopping on the validation loss and with patience parameter 10. The early stopping algorithm monitors the change of a metric, validation loss in this

case, and if the parameter does not reach the previous best after a number of steps specified by the patience parameter, halts training and restores the best network parameters so far.

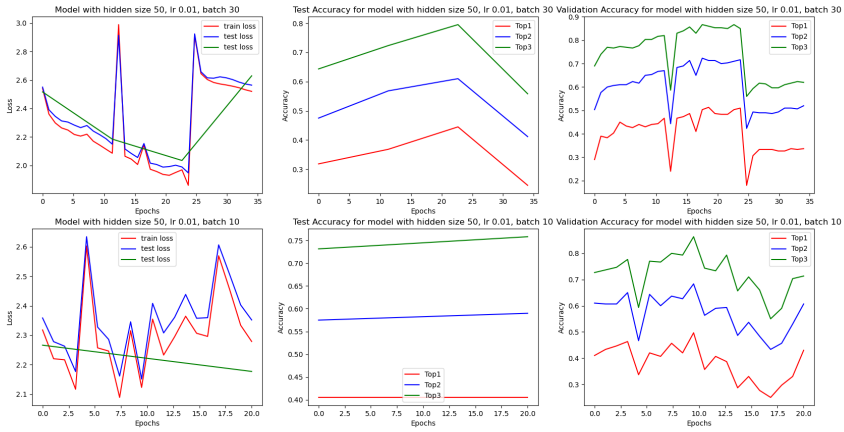

Figure 5: Training and test metrics for models with early stopping training. The exact parameters are given in the titles. The first column tracks the loss, where test loss is calculated only at every 10 epochs, and likewise, the second column is the accuracies on the test set after every 10 epochs. (That's why the test accuracy plot on the second row has only a line.

Looking at the metrics, we see that the erratic behavior of the network optimization has not been eradicated. Yet, as implied by the earlier confusion matrices, the networks suffered substantially from overfitting and random fluctuations in the parameter space. Early stopping seems to have partially solved this issue by restoring the best local optima in the parameter space, hence not letting the network progress into the overfitting zone or not letting the training be reset by, possibly, numerical fluctuations. The highest validation accuracy, which corresponds to the lowest validation loss as seen in Figure 5, is near 50%, which is quite an improvement from the final states of the models trained on 50 epochs. Moreover, for the model with parameters (50, 0.01, 30), the final test accuracies are (Top1 = 0.475, Top2=0.648, TOp3 = 0.818), and for the model with parameters (50, 0.01, 10), the final test accuracies are (Top1 = 0.405, Top2=0.59, TOp3 = 0.758), displaying an improvement from the earlier results. To assess the types of errors the model makes, below in Figure 6 we present the confusion matrices on the training and test sets.

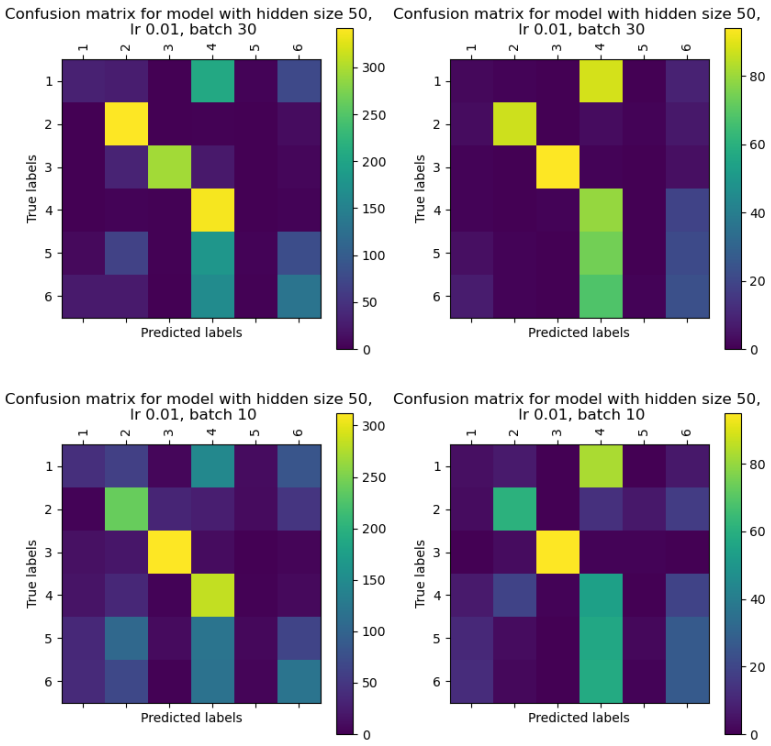

Figure 6: Training and test confusion matrices for models with early stopping. The exact parameters are given in the titles. The first column contains confusion matrices on the training set, and likewise, the second column contains the confusion matrices on the test set.

We clearly see the regularizing effect of early stopping, as the networks are both able to clearly distinguish not only the sitting state but also the jogging state from the rest of the data. However, both models still have a significant bias towards classifying walking-type motions (downstairs, standing, upstairs, walking) as standing, which we hypothesize to be caused by insufficient model power and the information loss during backpropagation through time as gradients tend to converge to extremely small or large values. Note that, however, this represents only one local optima caught by the early stopping algorithm, and that there might exist other local optima in the parameter space that favor one type of motion over a subset of the others, as was the case with the models in Figure 1 that gave constant output.

4 Discussion

There are two main points for discussion. First, a persistent problem with all the networks in this report is their tendency to overfit to one of the standing-type motions (Class 1,2,4,5,6). As the majority of the data comes from such data, the effects of individual data points may be attenuated over the backward computational graph, hence leading the network to distinguish data that are only clearly separable in the sample space. If the networks have overfitted to the training data, we might have confidently suggested regularization techniques such as L2 regularization and dropout. However, this behavior is also present in early stopped training with validation loss as the criteria, hence, it is more sensible to suggest that the model itself has shortcomings that cause this error, and to suggest improvements towards increasing model power. Some suggestions for this are using memory mechanisms, such as in LSTM networks, or maybe introducing some intermediary output layers to create a denser optimization signal on the earlier nodes of the network. The latter also makes sense in a dynamical systems point-of-view, as it would reveal more about the structure and behavior of the system earlier on in the time-series.

Second, throughout our analysis of this architecture for the given task, it has been apparent that the hyperparameter space is very rugged, as small changes in parameter combinations tend to yield significant differences in model performance. To resolve this issue, one should take a more systematic approach to identify regions in the hyperparameter space. This might be done by using more computationally intensive techniques such as grid-searching the hyperparameter space, or using random hyperparameters to exploit the rugged nature of the space to extract knowledge about what optimality hyperparameters are close to. As pertaining to our analysis, however, it has been clear that there is a trade-off between increasing the hidden size and introducing vanishing/exploding gradient behavior, and between using higher learning rates and batch sizes for more global exploration of the parameter space and using lower learning rates and batch sizes for more steadily approximating local optima.

Lastly, although the network do not have too many parameters to be fit, because the parameters are shared across layers, it would be significantly ben-

eficial to introduce new training data or use augmented data to increase the generalizability of the network.