Dynamical Neural Network Models for Sequence Learning

Sharath Ramkumar
Advisor: Dr. Robert Kozma
University of Massachusetts Amherst
COMPSCI 496: Independent Study

Abstract

In this study, we explore various neural network models for learning sequences. We also compare the robustness of these network models to randomized temporal noise and propose alternative spiking neural architectures.

Introduction

Neural network models are able to achieve superhuman performance on static tasks through recent break through in deep learning. However, the majority of real-world time-series data is often complex and noisy. As a result, we need dynamic models that can adapt to changing data streams to learn and predict in an robust, unsupervised fashion.

Background

There have been some recent studies comparing various models for time-series prediction tasks [**cui2016continuous**] on discretized data. In this study, we sill replicate their results to establish a baseline and propose unsupervised spiking neural architectures for performing this task. Spiking neurons [**gerstner2002spiking**] are biologically-inspired models of neurons. Spiking neural networks (SNNs) [**maass1997networks**] are organized connections between these spiking neurons which have more energy efficient [**cruz2012energy**] implementations on neuromorphic hardware.

Methodology

Artificial Dataset

Cui et al. (2016) propose an artificial dataset of sequences (formed with characters) with overlapping subsequences. A sequence is sampled from the dataset and presented to the model at each time step. After the last character of the sequence is presented, a character is sampled from the noise distribution and shown to the model. This process is repeated until $10,000$ characters (counting both sequences and noise characters) are shown to the model. After the model sees $10,000$ characters, the last character of sequences with shared subsequences is swapped. The performance on this task is the accuracy of predicting the last character over a window of the last 100 sequences. A sample dataset is summarized in Table 1.

Pre-10,000			Post-10,000		
Start	Subsequence	End	Start	Subsequence	End
6	8, 7, 4, 2, 3	0	6	8, 7, 4, 2, 3	5
1	8, 7, 4, 2, 3	5	1	8, 7, 4, 2, 3	0
6	3, 4, 2, 7, 8	5	6	3, 4, 2, 7, 8	0
1	3, 4, 2, 7, 8	0	1	3, 4, 2, 7, 8	5
0	9, 7, 8, 5, 3, 4	1	0	9, 7, 8, 5, 3, 4	6
2	9, 7, 8, 5, 3, 4	6	2	9, 7, 8, 5, 3, 4	1
0	4, 3, 5, 8, 7, 9	6	0	4, 3, 5, 8, 7, 9	1
2	4, 3, 5, 8, 7, 9	1	2	4, 3, 5, 8, 7, 9	6

Table 1: A sample artificial dataset for this task. The ending character can be predicted by learning the start character and the first character of the shared subsequence.

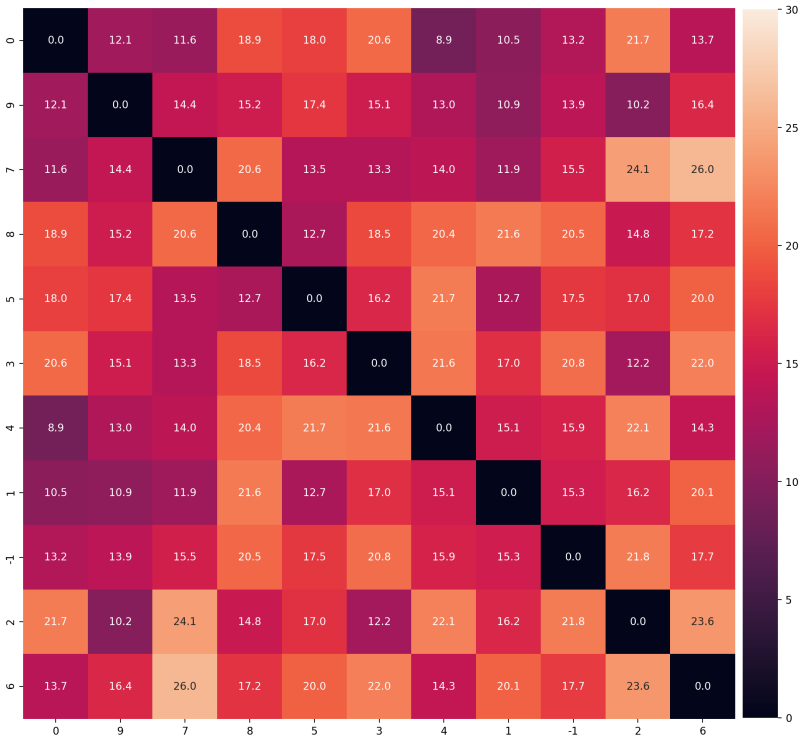

Figure 1: Heatmap of squared errors between distributed random encodings for the sequence characters, with -1 as the separating bit for ease of visualization.

Input Encoding

To present each of these discrete categories to the model, we have to encode the categories to a vector representations. One possibility is one-hot encoding where the vector is the same size the number of unique characters in the sequence, but this representation scales poorly when the number of unique characters increases. For this reason, the characters are encoded as random vectors with real values in the interval $[-1, 1]$, similar to distributed representations used in natural language learning. [**mikolov2013distributed**] The euclidean distances between pairs of unique sequence characters using a 25 length random ending is shown in Figure 1. The decoder for converting vectors back to

character uses a nearest neighbor approach, therefore the precision in the output vector from the model is important as the number of noise characters increases. Another possibility to consider for the input encoding would be the sparse distributed categorical encoder utilized by Cui et al. (2016) for their hierarchical temporal memory (HTM) model.

Models

Long short-term memory networks (LSTM)

Long short-term memory [**hochreiter1997long**] networks are able to achieve state of the art results on various sequence learning tasks, so they are a good starting point to establish baseline performance on this task. We use the same LSTM architecture proposed by Cui et al. (2016) with 25 input neurons connected to a hidden layer of 20 LSTM cells and 25 output neurons. The network is trained using backpropagation through time (BPTT) [**mozer1995focused**] on the last 1000 seen elements. The output is classified using the nearest neighbor decoder.

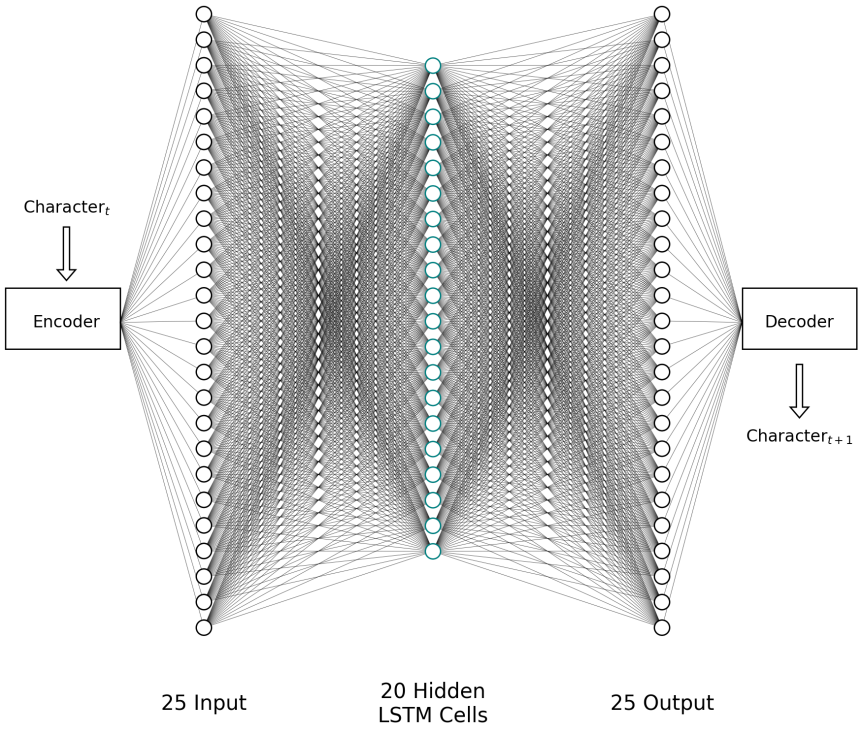

Figure 2: The network architecture for the LSTM-online model. The LSTM cells maintain an internal hidden state.

Time-delay neural networks (TDNN)

Time-delay neural networks (TDNN) are feed-forward neural networks trained using a lag on a window of previous data. [**waibel1995phoneme**] For this task, the TDNN is trained on the last 1000 characters using a lag of 10 using backpropagation. [**rojas1996backpropagation**] Again, using the model proposed by Cui et al. (2016), the first layer has 250 input neurons that are fully connected to a hidden layer of 200 neurons. The hidden layers utilize a sigmoid non-linearity and

are then fully connected to 25 output neurons. As with the LSTM model, the output is classified using the nearest neighbor decoder.

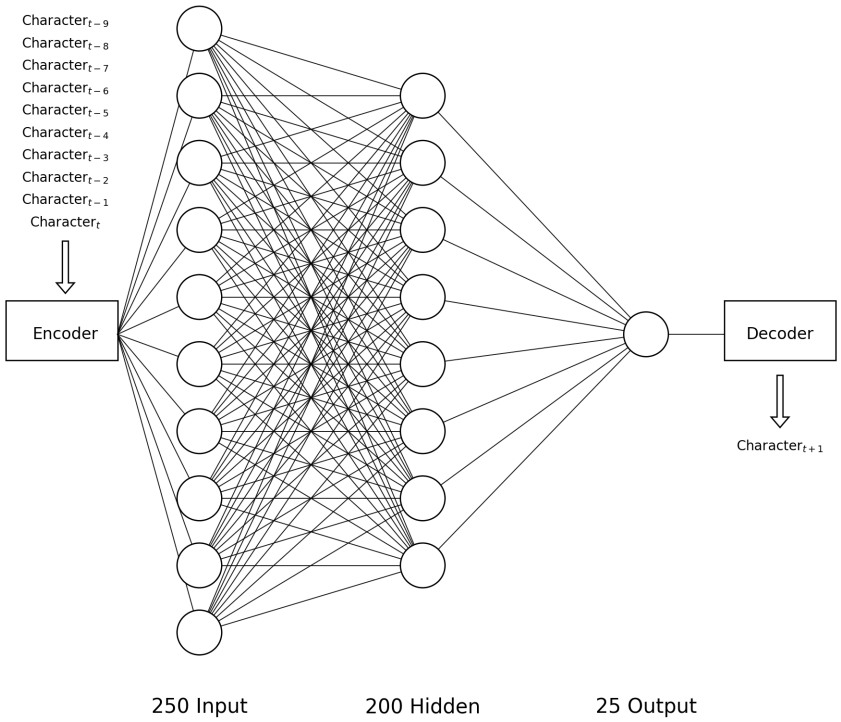

Figure 3: The network architecture for the TDNN/TDSNN model.

Time-delay spiking neural network (TDSNN)

A common approach to training spiking neural networks [**diehl2015fast**] is through the transfer of weights learned through backpropagation on an identical non-spiking neural network and then scaling the weights. Using this approach, we can convert the TDNN model trained with a bias on the output layer (due to the lack of negative spikes) to an equivalent spiking network using the approach developed by Diehl et al. (2016)

Readout

The drawback of using this method when we need to approximate real-values is the loss of precision based on the duration of the input. For this task, we show the input for 500 simulated milliseconds. The summed spikes from the output layer are converted to the 25 length vector by dividing by the runtime and subtracting 1. As with the other models, the output is classified using the nearest neighbor decoder.

Columnar Spiking Neural Network (CSNN)

We propose a spiking neural network architecture, with 10 inputs, each fully connected to 1 of 10 columns of 100 spiking neurons. Each of the columns has a Hebbian connection to each of the other columns. To map the spikes from this network, we train a K-Nearest Neighbor model using previous spikes and values. [**beliaev2007time**] The model diagram is shown in Figure 4.

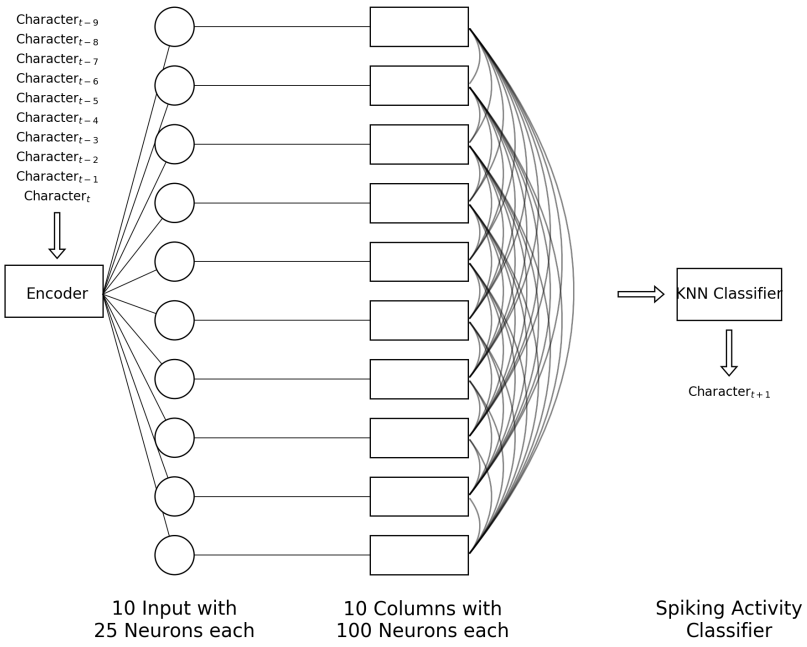

Figure 4: The network architecture for the CSNN model.

Experiments

Prediction Accuracy

We tested each of the models (LSTM, TDNN, TDSNN, CSNN) on the discrete sequence learning task. The prediction accuracy results are shown in Figure 5. The prediction accuracy is the percent of correct predictions over a window of the last 100 sequences.

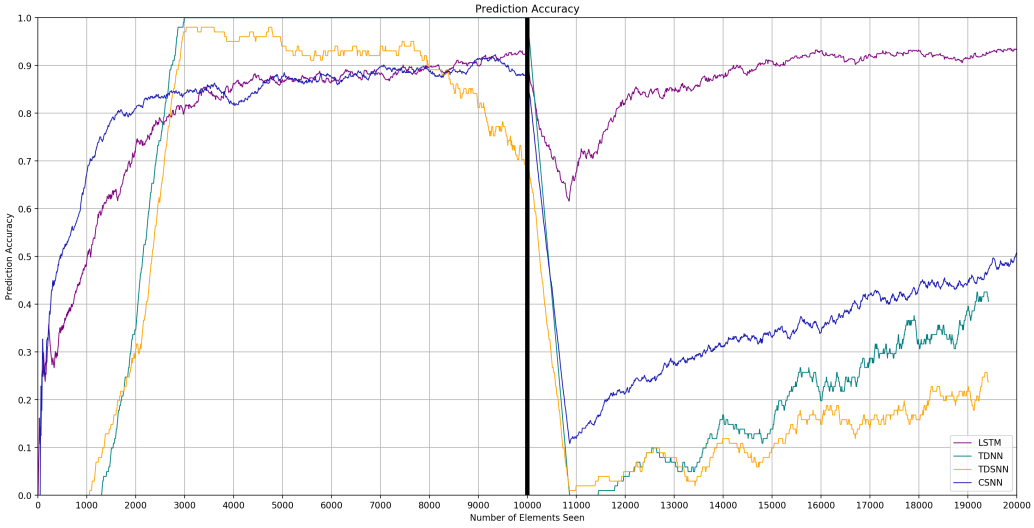

Figure 5: Each of the models were tested on their ability to predict the last element of the sequence.

The CSNN prediction accuracy dramatically drops after the sequence endings are swapped. To improve performance, we can use a window for the data samples in the KNN Classifier. Furthermore, the learning rates for the Hebbian updates could be more carefully chosen.