Unsupervised Time Series Forecasting with Spiking Neural Networks leveraging Spike Time Dependent Plasticity

F. B. Berg

Department of Electrical Engeneering and Computer Science, MIT

Summary

This paper looks into the potential of Spiking Neural Networks (SNN) and Spike Time Dependent Plasticity (STDP) used for 1-step ahead time series forecasting. Specifically focusing on using this unsupervised learning method to predict stationary data based on five previous data-points, a challenging task due to the large overlaps in clusters in the input-space of the network. The study explores weather STDP can be effectively applied to time series forecasting, a domain traditionally being dominated by supervised methods. The performance of the SNN is evaluated based on MSE and compared with AR(5) for reference. The results indicate that SNN do not match the precicion of traditional methods, but may hold promise for certain kind of boolean forecasting-related queries. The findings also underscore the importance of network architecture, biological parameters and ratio between number of neurons and training data. In the discussion we also look at receptive fields as the biological understanding of the synaptic weights, and why neural networks imitating biological structures are beneficial.

1. Introduction

1.1. Background - Time Series

The 1-step ahead forecaster in time series prediction is denoted as $\hat{X}_{T+1}(X_{T-p:T})$ indicating that it uses the values $\{X_{T-p}, ..., X_T\}$ to predict the true value X_{T+1}. The goal of a forecaster is to minimize the absolute error

$$e_{T+1} = |X_{T+1} - \hat{X}_{T+1}(X_{T-p:T})|.$$

$\hat{X}_{T+1}(X_{T-p:T})$ is an unknown function (let us call it g) where the realized X_i are the parameters.

A natural method to find a good g is to use neural networks. With ANN we can use back-propagation to adjust the weights of the system. This supervised learning strategy will over time minimize the error with training.

1.2. Background - Computational Neuroscience

One of the aspects that separates artificial neural network from the networks of the brain is the the activation functions are continuous in ANN. This allows back-propagation to exploit the gradient which in return allows for changes in the weights that will minimize the error.

In biology, the information is passed along synapses in discrete manners with action potentials. The neural network that imitates this behaviour is called Spiking Neural Networks (SNN). The strengthening and weakening of synaptic strength follows the principle of "firing together, wiring together". This is a completely different learning method called Spike Time Dependent Plasticity (STDP).

1.3. Problem Formulation & Motivation

In this paper I will use SNN and STDP as an 1-step ahead predictor for stationary data. The goal is to see weather we can use STDP as a learning rule for time series forecasting. This is also the motivation for using SNN, as STDP is unsupervised and it is not trivial that it will be able to create reasonable predictions.

Another motivation for choosing SNN is that computational neuroscience is an interesting field, and spiking neural networks are more relevant than ever with recent developement in neuromorphic computing [1]. In recent times there has been demonstrated that neuromorphic models in some cases have better performance and response in comparison to traditional machine learning methods. Neuromorphic computing is also relevant for brain computer interfaces, which is promising as a tratment to many neurological diseases.

2. Data

The data for this paper is daily temperature in Madrid over several years. The data is de-trended to remove the seasonality. From the data we create both the input and the output of the neural network. Assume the data is on the form $[X_1, X_2, X_3, ...]^T$, then the structure of the data used for the neural nets are

$$x = \begin{bmatrix} X_{1:5} & X_{2:6} & X_{3:7} & \dots \end{bmatrix} \rightarrow y = \begin{bmatrix} X_6 & X_7 & X_8 & \dots \end{bmatrix}$$

The first 80% of the data is used as training data while the rest is used for testing. The detrended data is also scaled so that the magnitude of the points are the firing rate of the neural network.

I also created bins for the data in y. As mentioned,

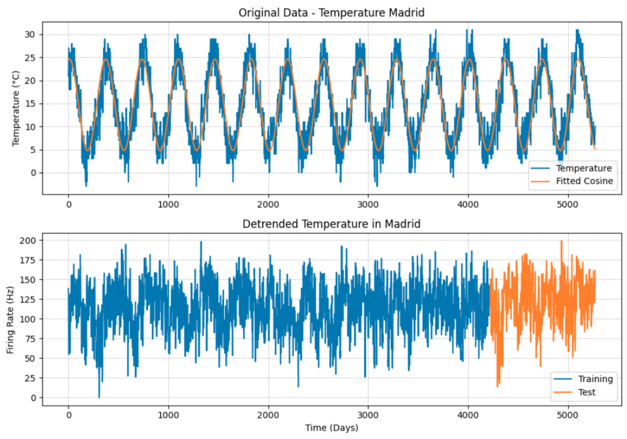

Figure 1: Traceplot of the original data, and the scaled detrended data

we will solve this problem unsupervised so the methods consists of clustering similar x data to the same value of y. We can use a PCA plot to visualize the clusters of input data and their respective bin.

2.1. Selecting bins

There are two intuitive ways of selective the bin sizes for for the y values. The first one is equidistant meaning that we select the bin in a

$$\text{y\_bin}_i = \lfloor y_i \cdot k \rfloor$$

This will cause the input data x to be somewhat normally distributed in the bins.

The other method is to use a percentile method for choosing the ranges of the bin. This makes the bins for the upper and lower values of y much larger as there are fewer values here. Below we have the PCA of the two bin sizes for 10 bins.
We can also conclude from the data that there exists clusters, but that they have large overlaps. The proportion of the total variance explained by the first component is 71,7% and the second is 16,0%. This implies that the variance in the PCA greatly captures the real variance. Therefore, clusters in the PCA will approximately be the clusters in five-dimensional space.

For the results and training in later stages we will use the PCA with equidistant bins. This gave much better results. The reason I think the result were that much better is due to the big overlap in the percentile bins. We see from the PCA Percentile plot that the yellow points reach far into the center of the data. This caused great confusion to the method which made it guess very wrong a lot of the times.

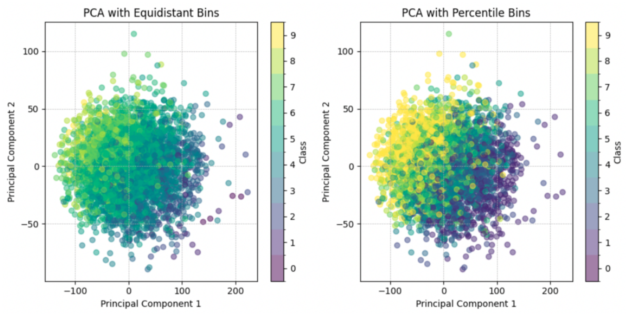

Figure 2: PCA of input data colored depending on corresponding y-value

3. Method

The models and methods used are inspired by the work of Diehl and Cook [2]. For the simulations we used the Brian library in Python.

3.1. Input layer

Each "observation" in the training set consists of five consecutive values that we call firing rate. Over a period of T ms the probability that a neuron will fire n times is given by

$$P_T(n) = \frac{(\lambda T)^n}{(\lambda T)!} e^{-n}.$$

The input data is scaled in such a way that $\lambda = X_t$ is the firing rate of the spike trains, so the scaling of the data is crucial for the learning of the network.

3.2. Architecture

The input layer is connected to an excitatory layer through a matrix of synapses. The neurons in the excitatory layer integrates the spike trains and sends a spike train further to the inhibitory neurons. Each Excitatory neuron has a bijective, constant and strong connection to a inhibitory neuron which makes it fire.

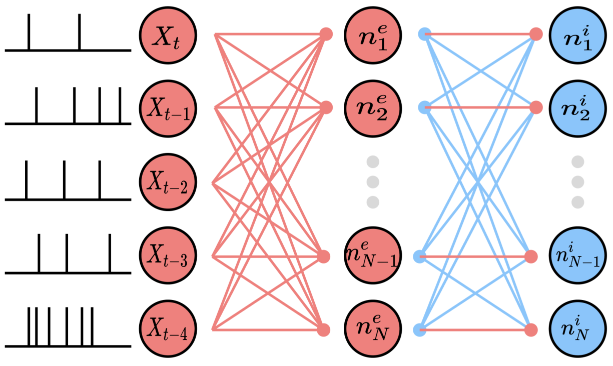

Figure 3: Network Architecture of the SNN

This causes a small, constant inhibition to all the other excitatory neurons. This kind of architecture makes it easier for individual neurons to adapt their receptive fields to input, so that not all neurons adapt the same pattern. The relative magnitude of the synaptic inhibition and the synaptic weights from the input layer to the excitatory neurons turned out to be a big driving factor for the learning quality, and these had to be fine-tuned by trials and error.

3.3. Neuron Model

We are using the Leaky Fire and Integrator model of a neuron. Each neuron is described by Kirchoffs law for current.

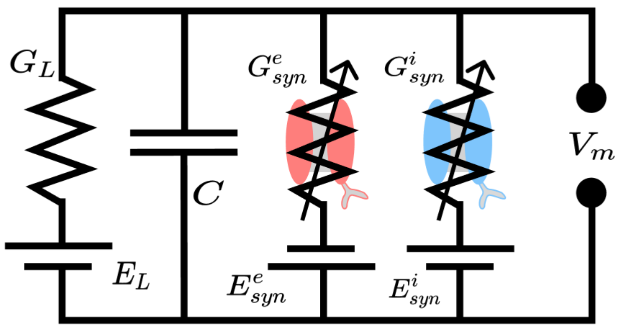

Figure 4: Electrical circuit used to model individual neurons

$$C\frac{dV}{dt} = G_L(E_L - V(t)) + G^e_{syn}(t)(E^e_{syn} - V(t))$$
$$= G^i_{syn}(t)(E^i_{syn} - V(t)).$$

Here G_L is the constant leakage conductance that causes the resting potential to stay at E_L. G_{syn} on the other hand depends on the property of the neuron and the activity of presynaptic neuron.

Each neuron has a property called V_{thresh}, and if the membrane potential of the neuron passes this level, the neuron spikes and the voltages is set to V_{reset}. When the presynaptic neuron i fires, the conductance in the postsynaptic neuron j increases with the weight w_{ij}, and in between fires it relaxes towards zero following

$$\tau\frac{dG_{syn}}{dt} = -G_{syn}.$$

A presynaptic spike will cause the conductance to increase and the voltage in the cell will relax towards E_{syn} which is either higher or lower than E_L for excitatory and inhibitory neurons respectively.

3.4. Learning

The learning of the network are properties of synapses. Synapses strengthens or weakens its ability to conduct signals depending on the relative timing of the pre- and postsynaptic neuron.

When presynaptic neuron fires we have that the presynaptic trace changes to $A_{pre} = A_{pre} + \Delta A_{pre}$. Similarly when the post-synaptic neuron fires we have $A_{post} = A_{post} + \Delta A_{post}$. In between the firing both traces goes to zero with their respective time constants.

The updating rule for the synaptic weight w is

$$\Delta w = \begin{cases} A_{post} & \text{pre-synaptic neuron fires} \\ A_{pre} & \text{post-synaptic neuron fires} \end{cases}$$

Here it is important to notice that $\Delta A_{pre} > 0$ and $\Delta A_{post} < 0$. We also set a max limit to the weight w_{max}, and enforce that it cannot be less than zero.

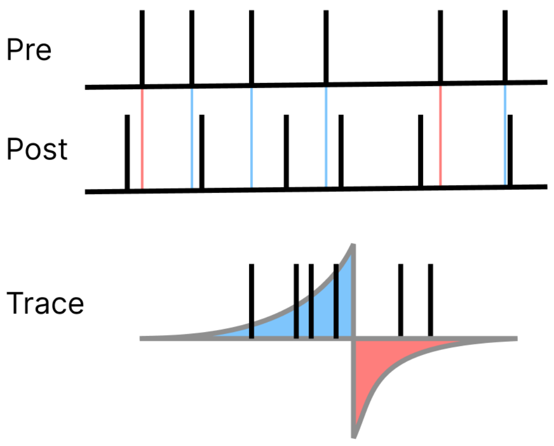

Figure 5: Visual interpretation of STDP using traces. Most presynaptic spikes in the image happen right before the postsynaptic spike which will cause the synaptic strength to increase.

Here is an example of a pulse being conducted through a synapse resulting in an increse in the weight. If a presynaptic neuron fires, the value of A_{pre} increases. Let us assume that the postsynaptic neuron fires just after that. Then $\Delta w = A_{pre}$ when the post-synaptic neuron fires which makes w increase by A_{pre} which is still significant and positive since the pre-synaptic neuron just fired. This strengthens the synapse and makes it more likely that they will fire together later. This is the principle of the saying "fire together, wire together".

3.5. Evaluation and prediction

In the prediction step of the model we count the number of spikes for each neuron in the exitatory layer. After the training we do a Random Forest Classifier to match the count of the layer to the real label. We then use our fit from the classifier to predict the class of in the testing data based on the firing rates of the exitatory neurons. The result is list of labels

In order to be able to evaluate the MSE of the prediction we need a single value, not a range. The naive approach to select a value the mean of the extreme values of the boundary. However we know that most data is compact around the mean, and sparse around the extreme values. This knowledge maybe indicate that we should choose values closer to the mean of the time series in the range.

Assume that label l has upper limit of the range as U and lower limit of range as P. In other words, when a prediction has the label l we predict that the temperature is on the interval $[L, U]$. This interval has a mean $M = \frac{U+L}{2}$ which can be looked upon as the center of the range. Let $\mu = E[X_t]$ be the mean of the training time series. The value of the prediction $\hat{X}^l_{t+1}$ given a range therefore becomes

$$\hat{X}^l_{t+1} = (0.5 + p)U + (0.5 - p)L,$$
$$p = \left(\mu - \frac{U+L}{2}\right)\frac{1}{\max(X_t) - \min(X_t)}$$

where $p \in (0, 0.5)$ tells how much we should weight the upper and lower limit of the range.

When converting from labels to the predicted value we should choose the value in the range corresponding to the label that is most likely to happen.

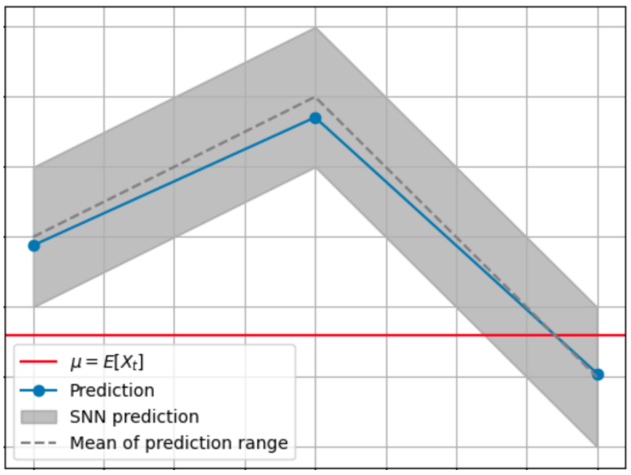

Figure 6: We choose prediction value that is closer to the mean than the middle of the prediction range

4. Simulation

I monitored the activity of the synapses and the neurons throughout the training to get some idea of what is happening in the hidden layer. All the following plots are from a test which had 500 neurons and 500 training data presented.

The first thing we should look at is the potential and the firing rates of the neurons. Overall, on all the neurons there should be a relative high firing rate, but if we look at the individual neuron we should be able to see suppression happening so that it does not only integrate and fire all the time.

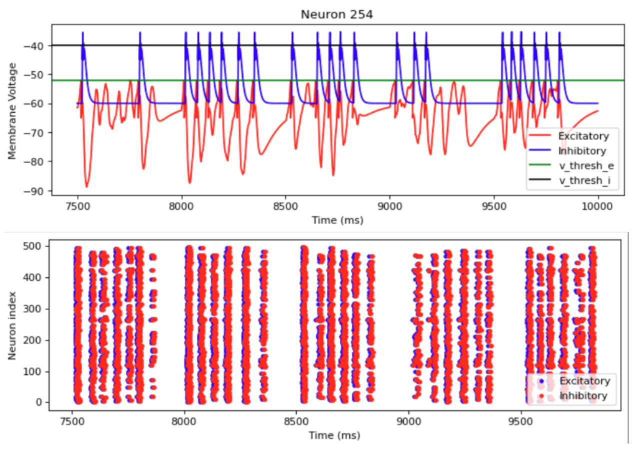

Figure 7: Neuron 254 spiking followed by inhibition. Also see an overview of all 500 neurons and their spikes in the 350ms period intervals.

Secondly we can monitor how the weight changes and what the resulting receptive field of the neurons are (how the weights are distributed over the input neurons). The weights are healthy if all weights do not converge to the same number, and if the receptive fields are different so that they pick up different patterns of the time series.

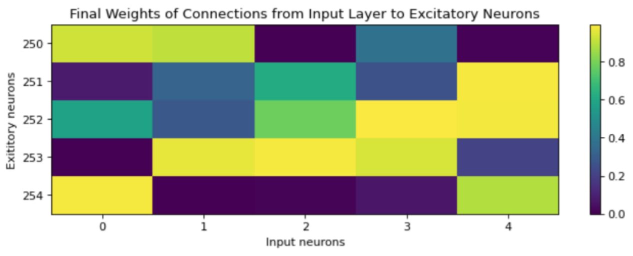

Figure 8: Synaptic strength after training from neurons to the input layer

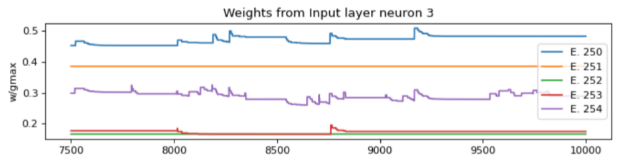

Figure 9: The value of the weight for a synapse changes every time the postsynaptic neuron fires relative to the presynaptic neuron

In the result section we will look at the quality of this simulation after labeling.

5. Results

I trained the network for varying parameters such as number of neuron, number of training data and number of bins.

I used two typer of performance measure for the prediction; Mean Squared Error (MSE) and percentage of correct labeling. The MSE tells us how far off from the true 1-step ahead value the prediction was, and the percentage tells us how many of the predictions were within the correct range. For the quality of labeling, the percentage is standard while the MSE is standard to measure the quality of predictions.

We use AR(5) as comparison as this is most common way to forecast given the five previous data points. For 100 day prediction the MSE of AR(5) is 412. We also have that the MSE between the 100 day predicted data and $\mu = E[X_t]$ is 1430.

Below are the results from multiple tests.

Neurons	MSE mean	MSE drift	Percentage
100	836.2	821.9	13,7
1000	693.2	670.8	18,1
3000	729.0	698	19,5

Table 1: Quality of predictions using 400 training data and evaluating MSE of 100 datapoints and 10 bins

Training data	MSE mean	MSE drift	Percentage
20	1307.0	1207.7	11,0
100	915.6	906.5	18,3
1000	822.0	787.1	17,8

Table 2: Quality of predictions using 100 neurons and evaluating MSE of 100 datapoints and 10 bins

# Bins	MSE mean	MSE drift	Percentage
10	575.9	562.9	21,1
20	555.1	547.4	10,0
40	652.4	644.7	5,8

Table 3: Quality of predictions using 500 neurons, 2000 training data and evaluating MSE of 500 datapoints

5.1. Visual result of 500 neurons, 500 training simulation

Now we will present the results of the simulation described in the previous section. This includes all the results that we got after the labeling. Firstly, we present the confusion matrix. The ideal outcome of the confusion matrix is strong values along the diagonal, and fewer matches the further we deviate from the diagonal.

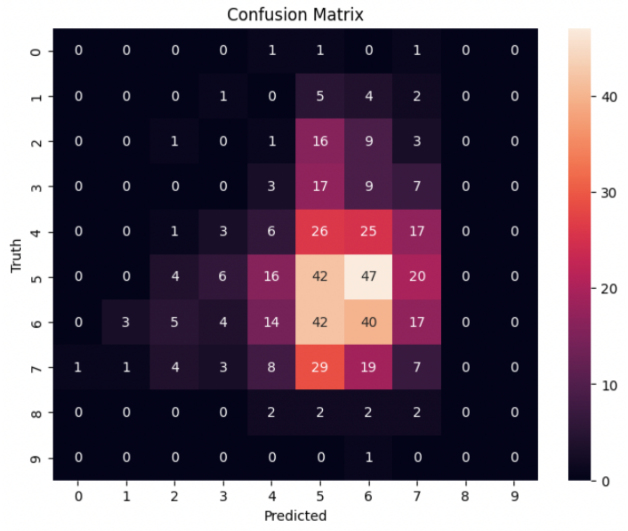

Figure 10: Confusion matrix after labeling

As mentioned, the labels corresponds to a region in the time series plot. We plot the prediction range with the AR(5) prediction and the actual values.

We also plot a shorter time frame to see the effect of the shifted predictor within the prediction range.

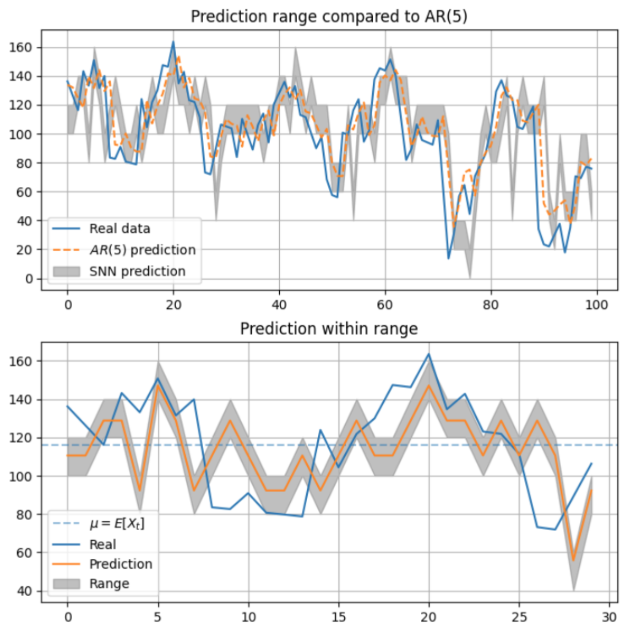

Figure 11: Plots showing the predictions compared to AR(5) and the actual values

6. Discussion

From the PCA we learned that there were large overlap between the clusters. An unsupervised learning method such as STDP performs surprisingly well. The performance on the prediction highly depends on the physical properties of the neurons, the architecture of the network, the learning rule of the synapses and much more.

6.1. Quality of predictions

We notice from Table 1 that the number of neurons cannot increase performance indefinitely. From 100 neurons to 1000 neurons there is a big jump in percentage of correct labels and in the MSE. However from 1000 neurons to 3000 neurons we did not see an increase in performance over the two metrics. One of the reasons for this might be that the training data is too small in comparison to the number of neurons which make a large amount of the receptive fields random (as this is their initial condition). Within the 400 training data only a certain amount of the neurons reach a good receptive field.

From Table 2 we see that the MSE is increased when we increase the number of training data. This is especially clear from the jump from 20 to 100 data. However, the MSE does not become much better from 100 to 1000 training data, and the percentage of correct labels is actually wrong. This might be because the receptive fields are overstimulated by the amount of training data. The architecture of the SNN is created much so that each neuron learn different patterns of the input. When the training data is much larger than the number of neurons, the receptive field might become combinations of input patterns which could reduce the percentage.

From Table 3 we see that when we increase the number of bins, the percentage drops dramatically. This is obvious as the individual bins are slimmer. However the MSE mean from 10 bins to 20 bins is only decreased sligthly, and the MSE from 20 to 40 bins is increased. In order to be able to trust the region somewhat, I would therefore prefer 10 bins.

From all the tables we also notice that the MSE drift is smaller than the MSE mean, and we conclude that it is a good decision to not chooses the naive predictor, which is the middle of the range.

Going further we notice that the confusion matrix shows a hill around the middle. Most of the predicted values were around the mean, and most of the truths ended up being around the mean. There is no clear strengthened line along the diagonal, but rather something that looks like a bivariate gaussian.

What is good about the confusion matrix is that there are not many predictions on the edges were there are no truths. This means that when we predict the wrong range, the truth is often in one of the neighbouring ranges, which is good for the MSE.

From the plots we notice that SNN prediction generally follows the real data. This is very impressive and surprising given the unsupervised nature of the method. However, there are som massive 1-step errors in the plot. It looks like the prediction sometimes completely misses by 4 ranges (labels), and then it picks itself back on track. From the plot it looks like every 10th prediction is pretty bad which likely account for most of the MSE error.

With 400 training data and 1000 neurons we managed to get a MSE of 670.8. This is less than double the MSE of the AR(5), and less than half of the trivial predictor $\hat{X}_{T+1} = \mu$. The quality is therefore not comparable to AR(5), but much better than trivial predictors.

6.2. Physical properties of the neurons

When creating a biological framework for a neural network there are countless parameters that needs to be determined. Each of the neurons has a threshold, a time constant, a reset voltage, and so much more. Most of the parameters were set to biological plausible values similar to what the values would be for neurons in the brain, but other had to be adjusted to increase the learning rate and the predictive power.

One of the more important parameters is the maximum conductivity g_{max}. It determines the maximum value of the synaptic strength between the input layer and the excitatory synapses. A large g_max made it easier for the neurons to fire. If g_max is high, then the rate parameter of the input spike train must be small, or else the neurons in the excitatory level would be overstimulated. This ratio turned out to be very important for the learning rate of the model.

Also, the constant weight w_{inhib} from the inhibitory neurons to the exitatory neurons were important for learning. This value is a small percentage of g_max. If w_{inhib} is large it will cause fewer spikes in the exitatory layer, but it will also cause stronger lateral inhibition so that there is less overlap between receptive fields.

6.3. Receptive Fields

As mentioned earlier, each of the excitatory neurons develops a receptive field for the input spike train, so that it is sensitive to certain firing rates. For example, the receptive field from excitatory neuron 254 to the input nodes has a very clear pattern that it detects. For X_t and X_{t-4} it has large weights, while it has almost zero for the three values in between.

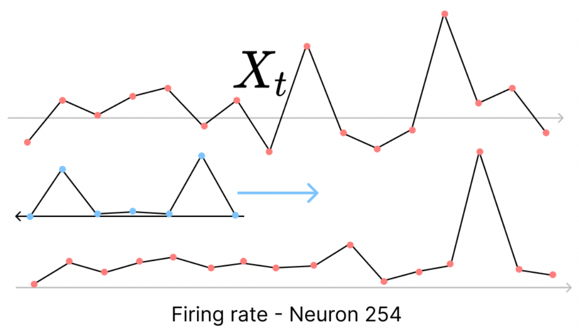

Figure 12: The effect of the receptive field of neuron 254

From the illustration we see how the neuron's receptive field would function on a higher level, almost as a convolution of the input data time series. We see that the highest firing rate is obtained when X_t and X_{t-4} has large values.

6.4. Architecture and Learning Rule

The architecture of the SNN does not take advantage of the temporal structure of the input data. The temperature tomorrow might depend on the temperature today and yesterday. But the temperature today also depend on the temperature yesterday. The input nodes could therefore with possible benefits be presented to the network in a chronological manner. A normal way to capture this temporal dependencies in neural networks are with recurrent neural networks. For artificial neural networks we would probably use this architecture, but it is harder to incorporate with SNN.

STDP is a local learning rule and in the architecture used in this project there is essentially only one layer, and only one layer of neurons that are being trained. This means that the local training of the synapses basically is a global training. When extending this to more layers there arises some problems. As mentioned, STDP is a local learning rule, so for a network with more layers, the local training will not necessary lead to productive global training.

Also in SNN, it is hard to determine how much each neuron from the input contribute to the final output. I tried implementing an RNN structure and discovered that it was incredible difficult to know wheather or not the signal from X_{t-4} had propagated through four layers of neurons or if the value had been neglected. It also required very precise tuning of variables so that the firing rate would not diverge or dilute from layer to layer.

However there are possible better ways one can incorporate a RNN structure to SNN. These include changing from STDP to another learning rule that is not local. There are many learning rules for SNN, but the one that might be the most popular is ANN-to-SNN Conversion[3]. This method trains the network with ANN, and then converts it to SNN. For time series forcasting, ANN-to-SNN conversion would be a great learning rule, but in general there are downsides to this as well. Firstly, the learning of the neurons no longer imitate biologican plasticity. Secondly, supervised learning is less flexible than unsupervised learning and require labels during training.

6.5. Use cases

A possible use case for the SNN model are boolean-related forecast predictions. The first thing that comes to mind is to predict weather a stock will go up or down the next day. Input variables can be parameters such as *open*, *close*, *high*, *low* and *volume* for the past two days. Based on these input variables (and maybe more) we predict weather the stock price goes up or down. An AR model would probably not do better than random guesses because of the efficient market hypothesis that states that the current price is the sum of all prior knowledge and that it is impossible to have a non-trivial forecast based on past values. However, the SNN can include parameters other than the value of a stock and bin it into two categories: up and down.

7. Conclusion

Exploration of spiking neural networks and STDP used for time series prediction have provided insight into the dynamics of a network imitating biological neurons. The analysis showed that despite overlap between clusters in the PCA, the network managed to achive descent results. The parameters involved in the training were heavily contributing to the quality of the predictions. With too much training data it seems that the receptive fields are being overwritten and noisy, not improving the MSE, and too many neurons ended up having many untrained neurons. This insinuated that performance would be maximized with somewhat similar level of training data and neurons. The ratio between firing rate of input, max weight of neurons, and the magnitude of the lateral inhibition also dramatically changed the outcome of the training.

The architecture of the network is probably the most influential aspects of the training and prediction, and it is not trivial to change. Recurrent neural network might be a reasonable architecture to try out, but it might require a supervised learning rule. In terms of MSE in forecasting, the current architecture was worse than AR(5), but an adjusted version of the model might do well in predicting weather a stock will go up or down (2 bins).

There is a big development in Brain Computer Interfaces, and SNN together with neuromorphic engineering have the ability to be in the frontier of this revolution. Also neuromorphic chips have the potential to be much more energy effective than normal computers. Although the technology is probably not that good in weather forecasting, it might play a role in treating neurological diseases.

8. Acknowledgments

I would like to express my gratitude to Adnan Rebei, my professor in Introduction to Neural Computation, for introducing me to the paper by Diehl and Cook. His support and discussions have been invaluable throughout the research.

References

[1] Yu Qi, Jiajun Chen, Yueming Wang. Jiajun Neuromorphic computing facilitates deep brain-machine fusion for high-performance neuroprosthesis. *Frontiers in Computational Neuroscience*, 17, 2023.

[2] Peter U. Diehl and Matthew Cook. Unsupervised Learning of Digit Recognition Using Spike-Timing-Dependent Plasticity. *Frontiers in Computational Neuroscience*, 9, 2015.

[3] Kashu Yamazaki, Viet-Khoa Vo-Ho, Darshan Bulsara and Ngan Le. Spiking Neural Networks and Their Applications: A Review. *MDPI*, 2022.