University of Tehran

ENCODING, LEARNING PLASTICITY, STDP AND R-STDP LEARNING RULE

Implementation of neural encoding, spiking neural network learning rule,dynamics of STDP, R-STDP and their behaviors analysis

Author

Arash Nikzad

Student No. 610301195

Supervisor

Mohammad GanjTabesh

Tehran, April 2024

Contents

1 Introduction

In the previous report, we implemented our first spiking neural population and various connectivity schemes, we also created balanced networks by using inhibitory and excitatory populations in the end we created a network of decision making. Throughout all these steps, we had never an ability to **learn**, Meaning the behavior of our network did not change according to the final results or the various spike patterns, more specifically the synaptic weights did not change to enhance or diminish some activities. In this report, we try to add learning behaviors to our networks using the two most popular learning rules in neuroscience, **STDP (Unsupervised)**, and **R-STDP (Reinforcement)**. We also try to implement some popular encoding for our network such as **Time-to-first-spike**.

2 Encoding

In this section, we introduce the three different neural encoding for different types of input such as images and numerical values. The encodings we focus on in this report are **Time-to-first-spike**, **Numerical-value-encoding** and **Poission-encoding**.

2.1 Time-to-first-spike Encoding

This method uses time-to-first-spike (TTFS) coding, in which each neuron fires at most once, and this restriction on the number of firings enables information to be processed at a very low firing frequency. This low firing frequency increases the energy efficiency of information processing in SNNs.

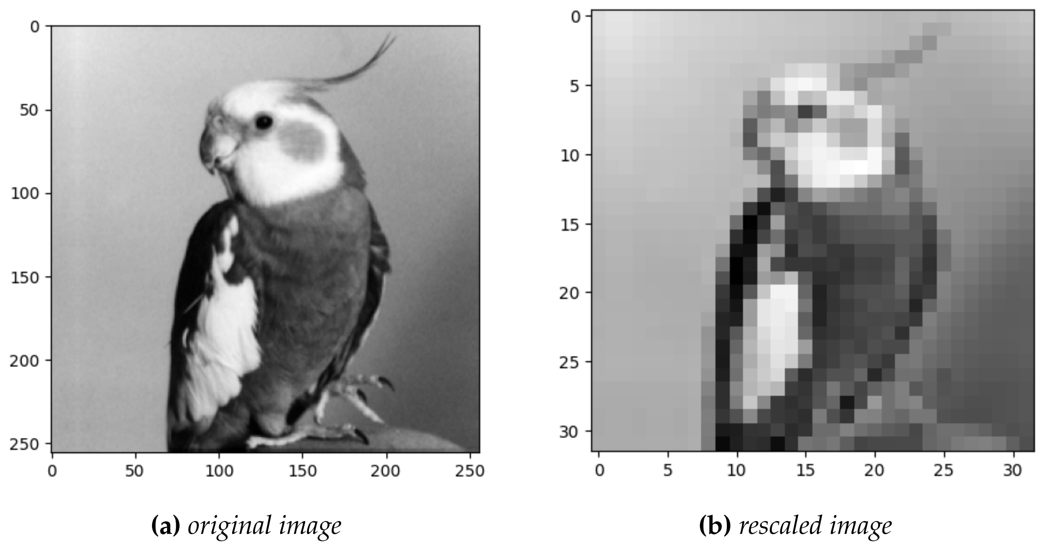
2.1.1 Data Type

In this section, we use **grayscale** images as our input and we convert them to spike trains. The original images are in `256*256` scale which in the case of plotting and showing the raster plot of spikes, might be too big, so the images have been rescaled to `32*32` for easier representation.

2.1.2 Encoding

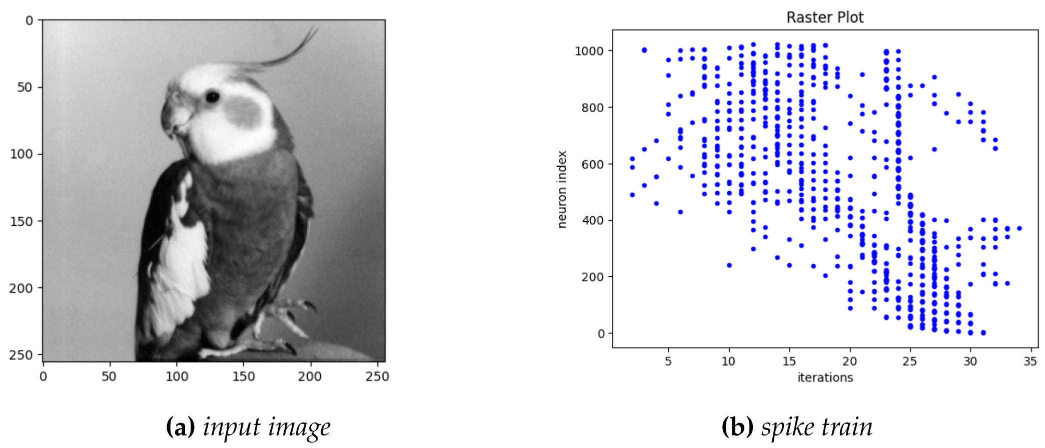
Now, we can use our images as input for our encoder. In this part the activities inside the network is not of up most importance, so we only focus on the spike pattern and raster plot of the first layer which is our encoder, having the same neurons as the number of pixels of the input images which in this case would be $32 * 32 = 1024$. Also due to the time dependance of our network, we input the image for a time window of `40ms` or in our simulation `40 iterations`.

In Figure 2.2, we can see the raster plot of the spike train created by the TTFS encoding of the given image. Here we have to consider and test two main aspects of TTFS. First one is the fact that a good encoding should be able to generate **different spike patterns for different inputs**, and the second one, particularly for TTFS, is the



(a) *original image* **(b)** *rescaled image*

Figure 2.1: *rescaling the input from* $256 * 256$ *to* $32 * 32$



(a) *input image* **(b)** *spike train*

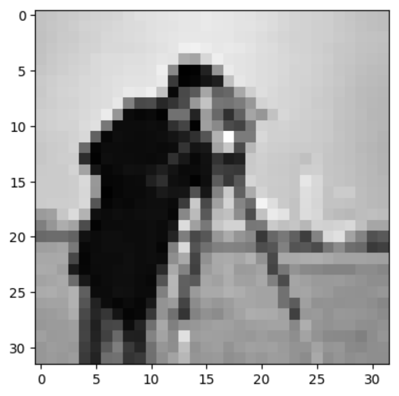
Figure 2.2: *TTFS Encoding spike train on grayscale image*

fact that **each neuron should fire a spike at most once** in this encoding.
In order to check whether the first condition holds in our encoder, we must encode yet another input to see its spike pattern and compare it to our first input's spike train.

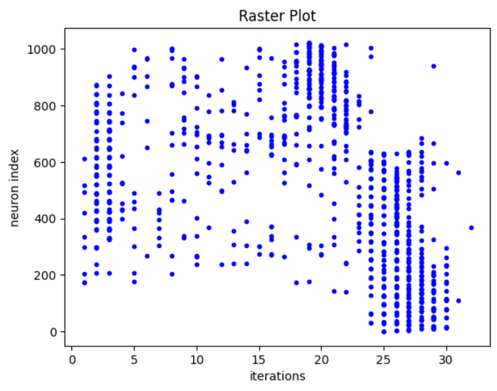
Here, Figure 2.3, shows yet another input and another encoding of that input. It seems like the spike trains of these two different images are different enough, nevertheless, for a better understanding of their differences and comparison, it's better to see the two spike patterns together.

As we can see in Figure 2.4, as the inputs are considerably different their generated spike patterns are also compeletly different which is the first thing that we expect from an encoder in general. Now we have to check the second factor, in order to have a more vivid view of each neuron spiking only once, we rescale the image, once again to `8*8` pixels to be able to see the individual neuron spikes in the raster plot.

Figure 2.5, shows clearly that each neuron spikes at most once in this encoding, although the image in `8*8` scale is not clear what so ever, This rescaling is only for

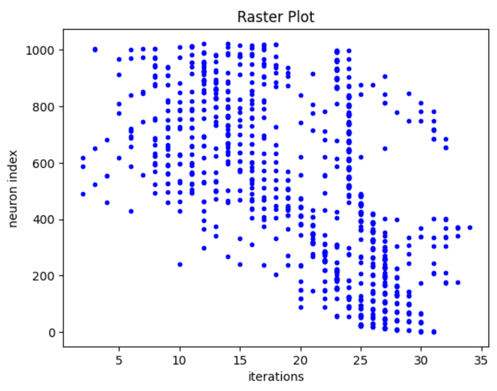


(a) *input image*

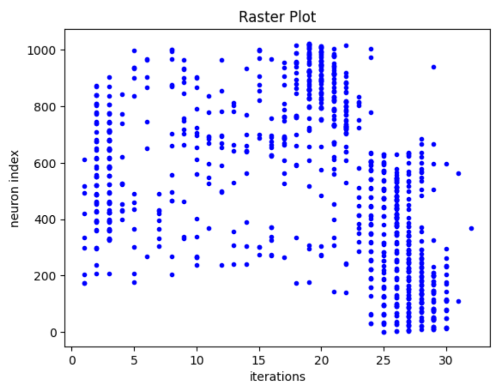


(b) *spike train*

Figure 2.3: *TTFS Encoding spike train on grayscale image*

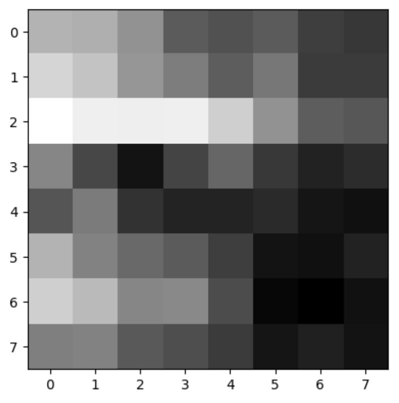


(a) *spike pattern of 'bird.tif'*

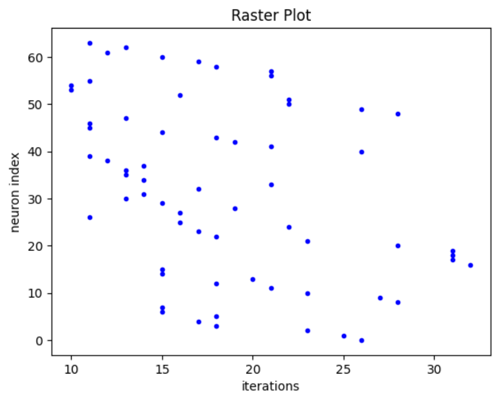


(b) *spike pattern of 'camera.tif'*

Figure 2.4: *Comparison of spike patterns of different inputs*



(a) *input image*



(b) *spike train*

Figure 2.5: *TTFS Encoding spike train on grayscale image* $(8 * 8$ *pixels)*

visualization purposes and the behavior of neurons and the encoding in general is not effected by the size of each input data.

2.2 Numerical-values Encoding

This encoding is used to encode numerical values, we try to convert a single numerical value into a spike train of a group of neurons. For this purpose, we use a bell curve for each single value and for each input, we use the value in which it cuts its curve, giving us a vector of values which can be considered different times of spikes for each of these neuron.

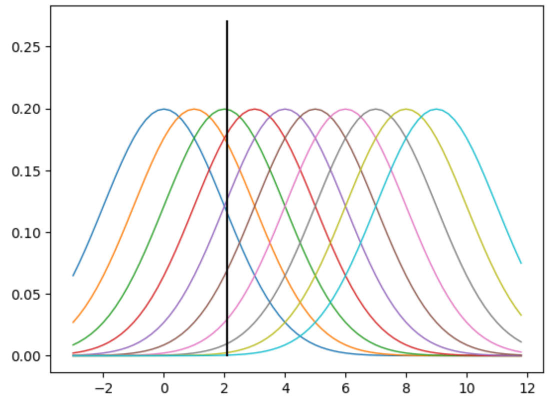
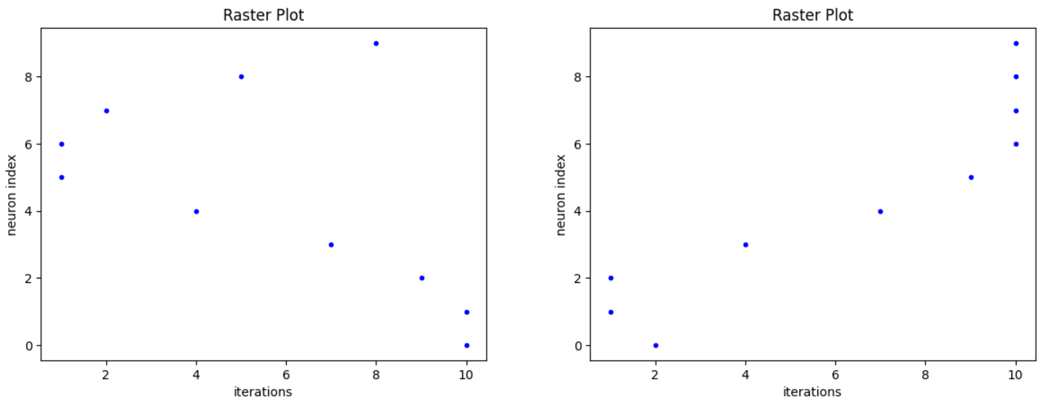


Figure 2.6: *bell curves of each number*

Now, we use these curves to encode our numeric-valued input, first we use a small number of neurons `10 neurons` and the two inputs of `5.78` and `1.19`.



(a) *spike pattern of* 5.78 **(b)** *spike pattern of* 1.19

Figure 2.7: *Numerical-values Encoding spike pattern on numeric-valued inputs*

Here in Figure 2.7, we can see that not only, each neuron spikes only once, but also the spike times are like the curves we saw in Figure **??**. Of course if we increase the number of neurons for the encoder, the accuracy of distinguishing different values would differ and also the more the spikes would look like a bell curve centered around the number we used as input.

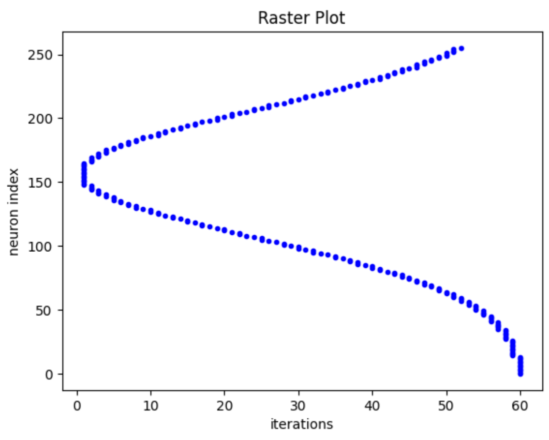
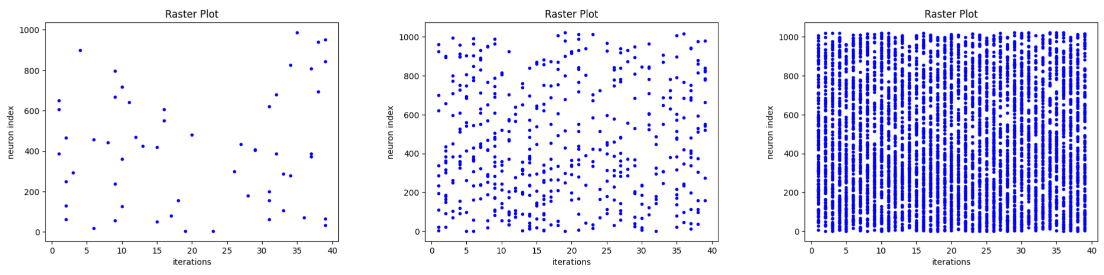


Figure 2.8: *Numerical-values Encoding with* 256 *neurons and* 156.72 *as input*

2.3 Poission Encoding

In the last section of the encoding chapter, we use the poission distribution to generate spikes. We can use two different methods to generate spikes using the poission distribution. One is using each pixel as the input and the other would be to use time as the input of the poission distribution, generating a random number for each iteration and generating a spike accordingly.

At first we used pixels as the input of the poission distribution to see the results for different values of λ of the poission distribution.



(a) λ = 0.00001 **(b)** λ = 0.0001 **(c)** λ = 0.001

Figure 2.9: *Poission Encoding with different* λ

In Figure 2.9, The results show that, firstly it might be better to always normalize the values of the input regardless of its initial range in order to prohibit any activies like the right most spike pattern. Also we can see that it is up to us to have a trade-off, the lower λ would get, the more energy-efficient the network would be, on the other hand, we would lose precision of our encoder. In order to have a better understanding of the poisson encoding, this time we use time as the input of the possion distribution, meaning we expect to see more spikes around the value of λ.

Figure **??** has a better visualization of the poission encoding other time, as we can see, the density of spikes is higher around the value of λ and decreases as we step away from the mean which is what we expect. keep in mind that as we can see in both

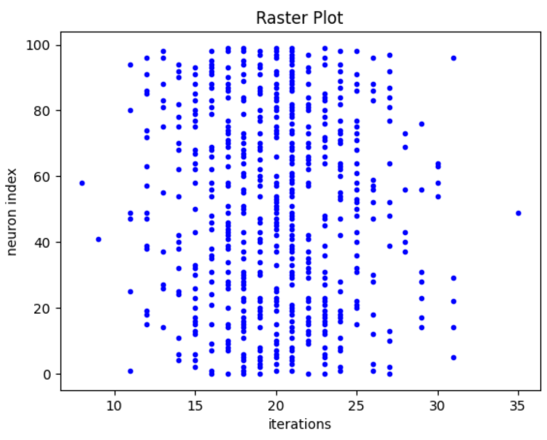


Figure 2.10: *Poission Encoding on time with* `T = 40` *and* $\lambda = 20$

variations this encoding does not garantee only a single spike per neuron. Now that we know how to encode the outside input into our SNN, we are ready for our first training. In the next chapter we focus of STDP learning rule and its effects on synaptic weights and spike firing of the neurons in the network.

3 Spike Time Dependant Plasticity (STDP)

In Chapter 2, we implemented three main neural encoding in order to convert various input types including numerical values or images, into spikes. Now, with use of these encodings, we can now give different patterns to our neural network and our goal is for our network to **learn** these patterns, meaning, we aim that after the procedure of learning in our network, there would be some special spike pattern, according to the specific pattern given to our network which it had learned.

In this chapter we are going to implement the unsupervised learning rule **Spike Time Dependant Plasticity (STDP)**, derived from **Hebb's rule**.

Pair-based STDP learning rule

$$|\Delta t = t_{post} - t_{pre}| \tag{3.1}$$

$$\Delta w_{+} = A_{+}(w).exp(-|\Delta t|/\tau_{+}) \;\; at \; t_{post} \quad for \;\; t_{pre} < t_{post} \tag{3.2}$$

$$\Delta w_{-} = A_{-}(w).exp(-|\Delta t|/\tau_{-}) \;\; at \; t_{pre} \quad for \;\; t_{pre} > t_{post} \tag{3.3}$$

There are many variations of STDP, as **pair-based STDP**, **Flat-STDP**, **Triplet STDP**, **Trace-based STDP** and etc. Here in this report we used both Flat-STDP and Trace-based STDP.

3.1 Trace

In order to save changes of weights for STDP, we need to have the trace, remaining of each spike and activity of pre and post synaptic neuron of each synapse. These traces would work as the memory of our STDP, saving the effects of each neuron in order to know how much should STDP rule effect the pre or post synaptic neurons.

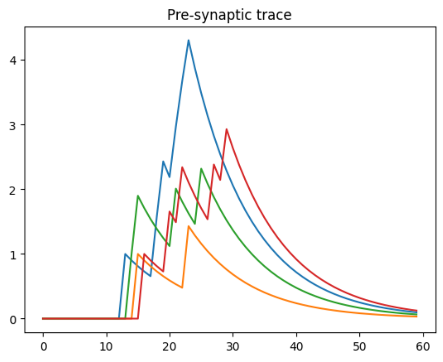
Pre-synaptic trace

$$\frac{dx_j}{dt} = -\frac{x_j}{\tau_+} = \sum_f \delta(t - t_j^f) \tag{3.4}$$

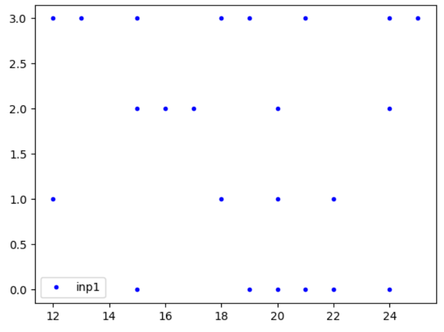
Post-synaptic trace

$$\frac{dy_i}{dt} = -\frac{y_i}{\tau_-} = \sum_f \delta(t - t_i^f) \tag{3.5}$$

Here, we work with a simple 2-layer network, one input layer of 8 neurons and one output layer consisting of only two neurons namly (A) and (B) with a poisson spike pattern generated as their input.



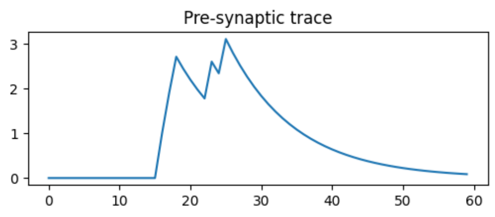
(a) *traces of input layer neurons*



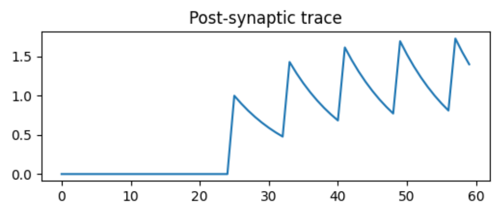
(b) *input layer raster plot*

Figure 3.1: *pre-synaptic traces of input layer neurons*

As we can see, each neuron has a trace at first being zero and gets bigger as they spike and decays exponentially. for a better understanding its better to see these traces for a single synapse as one, pre and one post synaptic neuron's traces.



(a) *pre-synaptic trace x_i*



(b) *pre-synaptic trace y_j*

Figure 3.2: *pre and post synaptic traces for a single synapse*

Here, Figure 3.2 shows both pre and post synaptic traces of a neuron, here we can see this synapse should be somewhat useful in the STDP learning rule because most spikes of pre-synaptic neuron occured before the post-synaptic neuron inhancing their connection.

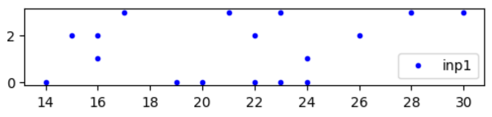
3.2 Pattern learning

Now that we have implemented a trace for each neuron, we can use this method to implement our first STDP learning rule for our network expect the change of weights of synapses between the two layer of neurons.

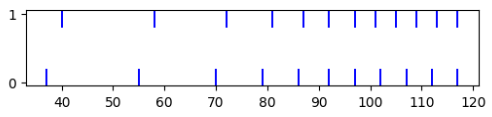
STDP using trace

$$\frac{dw_{ij}}{dt} = -A_-(w_{ij})y_i(t)\sum_f \delta(t - t_f^j) + A_+(w_{ij})x_j \sum_f \delta(t - t_i^f) \tag{3.6}$$

Here at first we explore the STDP learning on our network, only with the first input population having a poisson distributed pattern, with **fixed weight connectivity**.

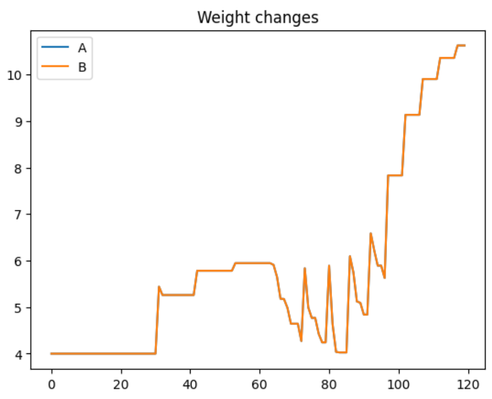


(a) *input layer raster plot*

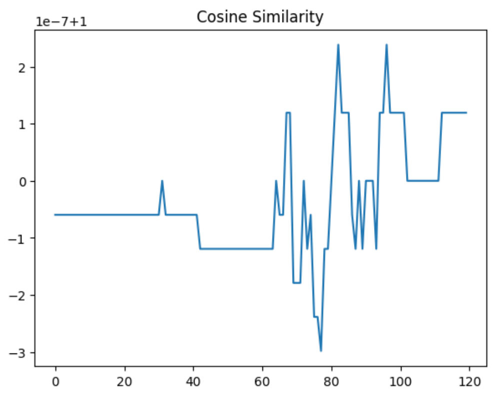


(b) *output layer raster plot*

Figure 3.3: *network raster plot*



(a) *weight changes of output layer*

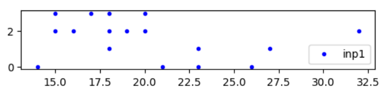


(b) *cosine similarity of the output layer*

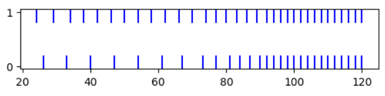
Figure 3.4: *network full fixed connectivity with STDP learning rule*

As we can see in Figure 3.4, both input and outputlayer has some spikes, more importantly, we can see that both neuron (A) and (B) in the outputlayer have to some extent exactly the same weight changes and have a cosine similarity of a number about one. This is because all the connections for both of these neurons are exactly the same, resulting in the same spike timing with a little difference in their initial potential, resulting in the same effect of STDP on both of them. As a result both of neurons (A) and (B) have learned the same pattern. Now let's change the initial weights to be more random and flexible.

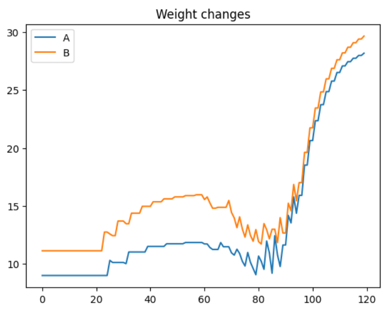
Here in Figure 3.5, we can see that the change in the initial value of the weights caused the weights of (A) and (B) to be different, of course again due to somewhat



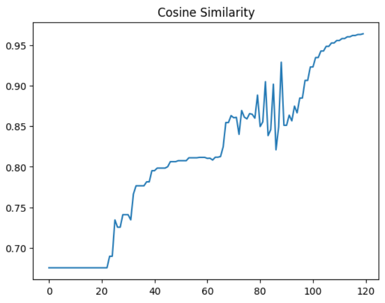
(a) *input layer raster plot*



(b) *output layer raster plot*



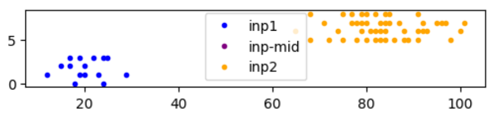
(c) *weight changes of output layer*



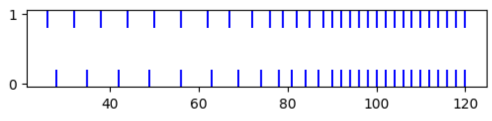
(d) *cosine similarity of the output layer*

Figure 3.5: *network full random connectivity with STDP learning rule*

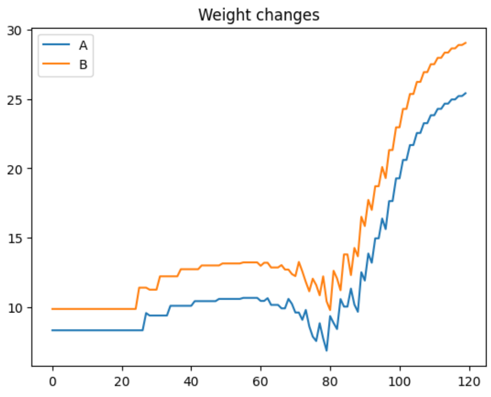
the same connectivity they spike about the same time, but the difference has made it so that (A) has learned the pattern better. Also we can see that their cosine similarity is converging due to the fact that they are both still learning the same pattern so the weights are getting closer to each other although the initial difference. Now we can give both patterns to the network in different times to see which neuron learns which.



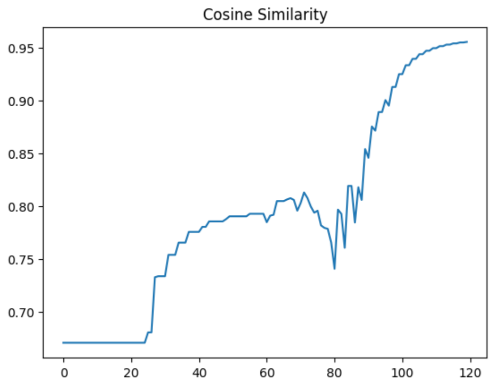
(a) *input layer raster plot*



(b) *output layer raster plot*



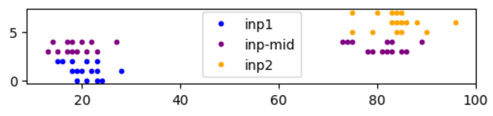
(c) *weight changes of output layer*



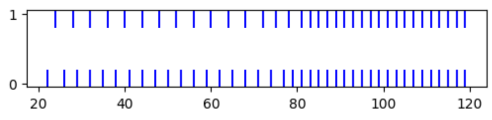
(d) *cosine similarity of the output layer*

Figure 3.6: *network full random connectivity with STDP learning rule*

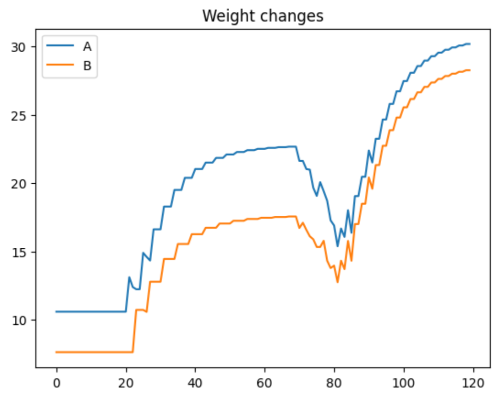
Figure 3.6 shows again, the network started to learn the first pattern, but the first pattern ends, in the middle and most importantly, as the second pattern start, we can see that the weights of (A) and (B) have decreased as shows that it has learned the previous pattern shown. However, we can see that the same as before, both of these neurons have learned both patterns as well which may not always be a great way of learning. Now lets get the two patterns closer and have intersecting parts to see the results.



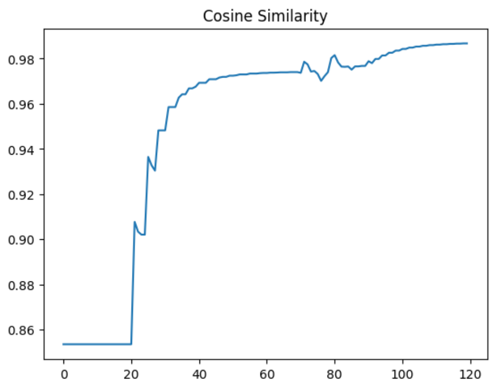
(a) *input layer raster plot*



(b) *output layer raster plot*



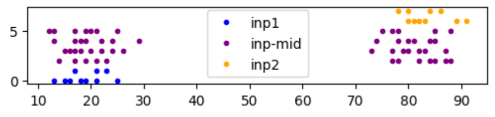
(c) *weight changes of output layer*



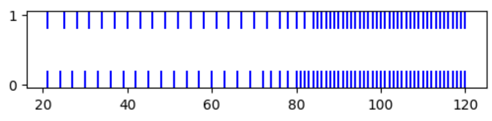
(d) *cosine similarity of the output layer*

Figure 3.7: *network full random connectivity with STDP learning rule - low common pattern*

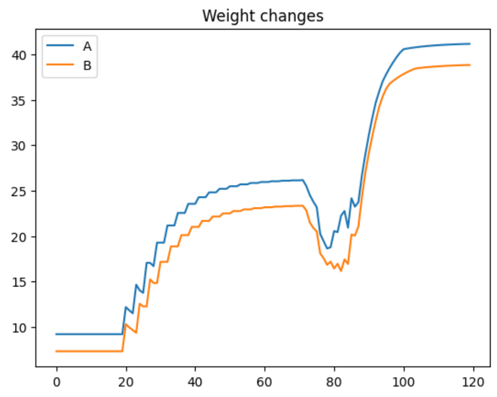
In Figure 3.7, we can see that with a common pattern between the two input patterns, the loss of the first pattern by both (A) and (B) has decreased as we can see that the weight didn't decrease as much as the last time that the patterns were completely different. Now we increase these patterns common neurons to see the effects.



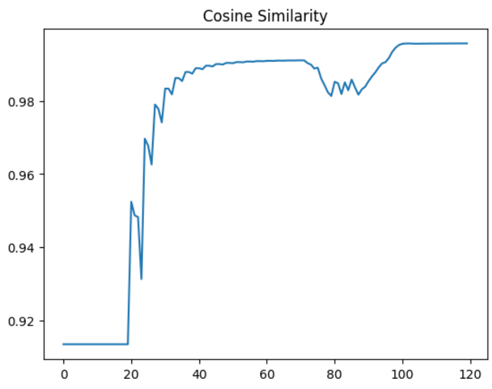
(a) *input layer raster plot*



(b) *output layer raster plot*



(c) *weight changes of output layer*



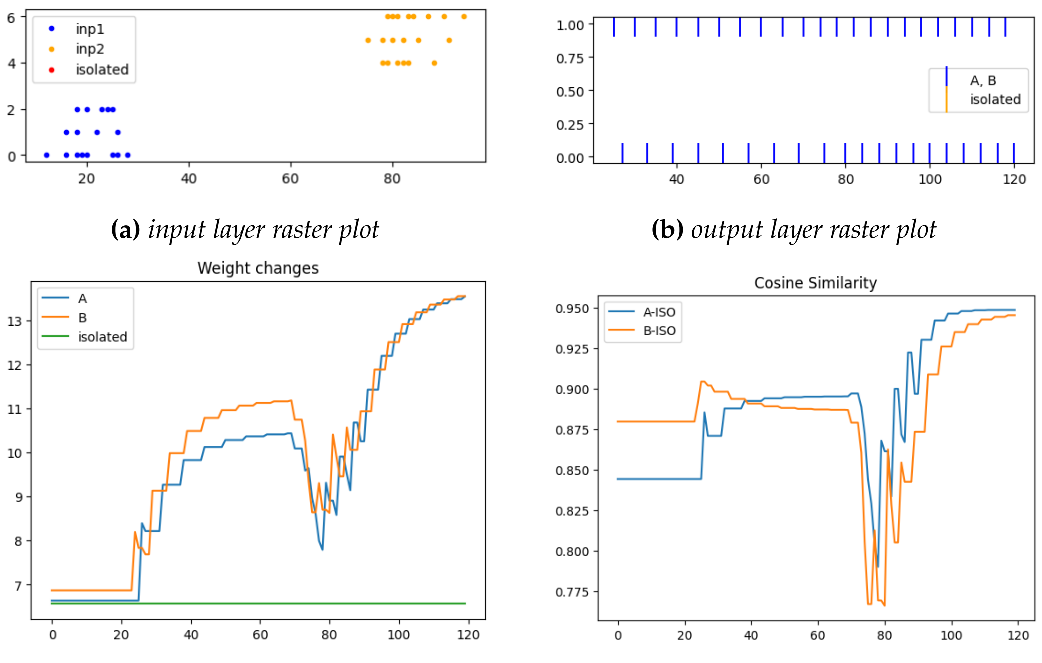
(d) *cosine similarity of the output layer*

Figure 3.8: *network full random connectivity with STDP learning rule - high common pattern*

Here in Figure 3.8, we can see that by increasing the common pattern area, the rate of spikes has increased considerably in the output layer as well as having a lower loss when the second spike pattern arived, this is because by increasing the common pattern area, the model is somewhat learning only the common pattern and so entering the second pattern results in amblifing the previous learned pattern.

3.3 Isolated neuron

In this section, we want to test out what happens to the neurons in which they never spike, so-called isolated neuron, for this purpose, we add one isolated neuron to the input layer and one to the output layer and analyse their behavior.



(a) *input layer raster plot* **(b)** *output layer raster plot*

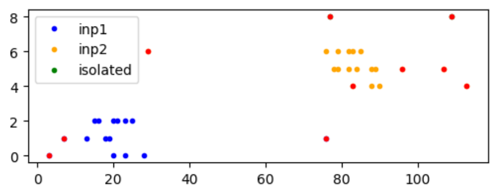
(c) *weight changes of output layer* **(d)** *cosine similarity of the output layer*

Figure 3.9: *network full random connectivity with STDP learning rule - with isoalted neurons*

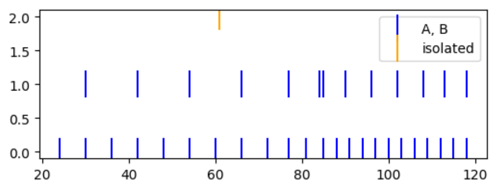
As Figure 3.9 shows, We can see that due to the fact that these isoalted neurons never fire a spike and STDP learning rule is only applied to synapses where both pre and post synaptic neurons have fired a spike at least once, here the synaptic weights of the isolated neuron has never changed and the only reason of changing its cosine similarity to neurons (A) and (B) is the fact that their weight changes over time.

3.4 Background activity

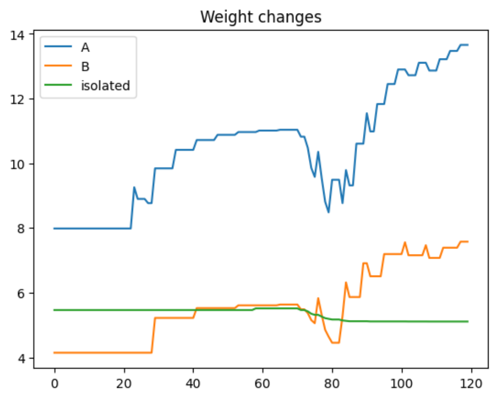
In our cortex, neurons have background activity, meaning that even if there is no external stimulus, neurons spike at random in different times to prevent the neuron from, put in simple terms, forget what it had learned and for it to be involved in the process of learning. In order to simulate this pattern, we can either give a low input current as input to our neurons, or we can use probabilistic methods to randomly force-spike some of the neurons at each iteration.



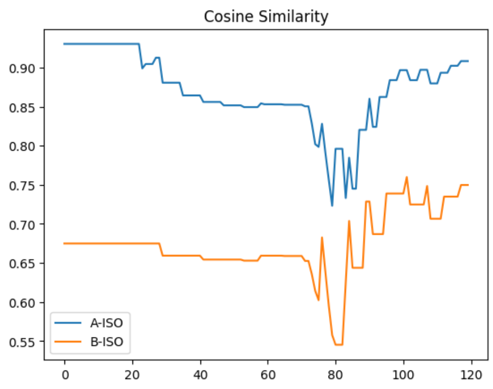
(a) *input layer raster plot*



(b) *output layer raster plot*



(c) *weight changes of output layer*



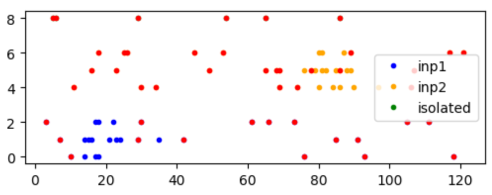
(d) *cosine similarity of the output layer*

Figure 3.10: *network full random connectivity with STDP learning rule - with background activity*

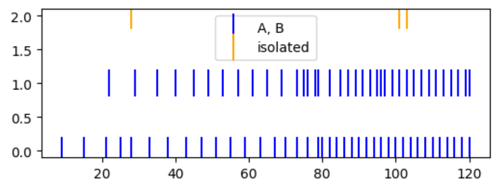
Figure 3.10 indicates the effects of background noise, as we can see, with this background activity for each neuron, not only neurons have spikes other than times where there is an input stimulus, but also, even the isolated neuron that did not have any spikes through out the experiment, both in input and output layer has spikes which results in being involved with learning and has a change in its weights. we have to bear in mind that if background activity is high enough, it could intrupt the actual activity of the network and the process of learning the input pattern.

As we can see in Figure 3.11, by increase in background activity, we can see that the weight changes has more fluctuation as well as their cosine similarity, of course we still have the pattern learning ability by the network however, there are some miner intruption in the process. Now lets take it up a notch one more step to see the results.

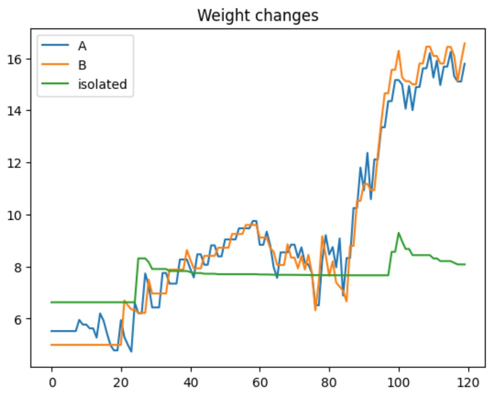
Now we can see the destructive effect of background activity in Figure 3.12, as we can see not only we have far more fluctuation but also as the stimulus end, the network weights starts to decrease as the random activity causes it to spike and the STDP makes it learn these noisy activity, resulting in forgeting the pattern which is not a desired result so we should try to keep the background activity level to low or average.



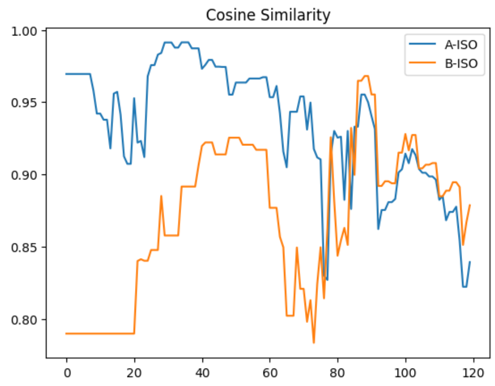
(a) *input layer raster plot*



(b) *output layer raster plot*

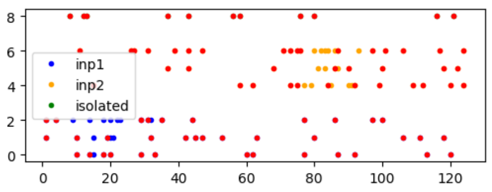


(c) *weight changes of output layer*

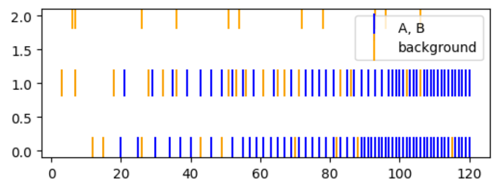


(d) *cosine similarity of the output layer*

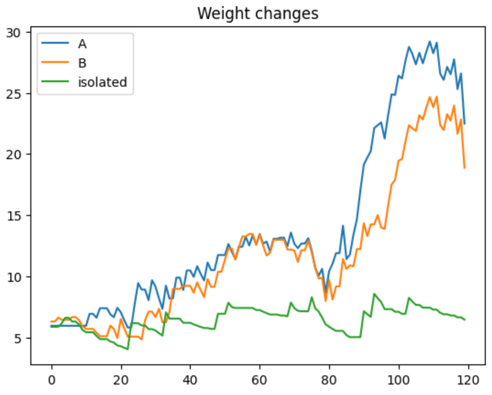
Figure 3.11: *network full random connectivity with STDP learning rule - with average background activity*



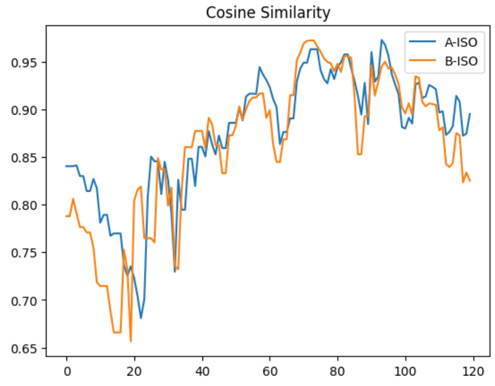
(a) *input layer raster plot*



(b) *output layer raster plot*



(c) *weight changes of output layer*

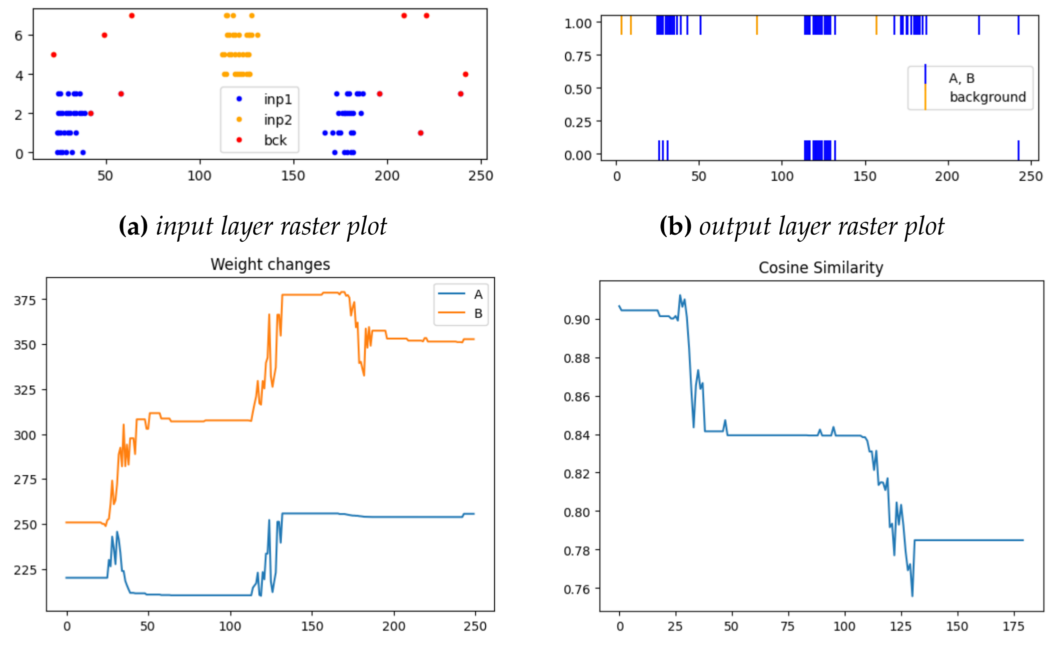


(d) *cosine similarity of the output layer*

Figure 3.12: *network full random connectivity with STDP learning rule - with high background activity*

3.5 Lateral inhibition

We know that STDP is an unsupervised learning rule, meaning we do not have any control over what the network is actually learning, in the case of STDP, it is generally learning the common pattern in the input stimulus. Throughout this chapter we observed that we cannot teach the network for the two output neurons (A) and (B) to spike for different patterns. Here by applying the lateral inhibition to the output layer, meaning, we add a inhibitory synapse between them to lower the others activity every time they fire a spike, we wish for the network to have a better ability to distinguish between the pattern more accurately.



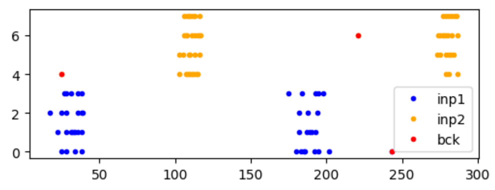
(a) *input layer raster plot* **(b)** *output layer raster plot*

(c) *weight changes of output layer* **(d)** *cosine similarity of the output layer*

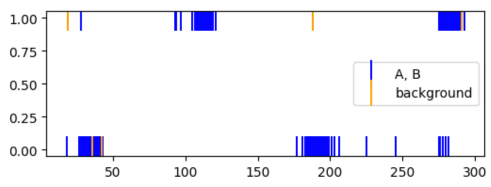
Figure 3.13: *network full random connectivity with STDP learning rule - with lateral inhibition*

We can see in Figure 3.13 the lateral inhibition causes that the first one to spike between (A) and (B) for a stimuli, it inhibits the other one to spike resulting in one learning a pattern while the other one does not which is something we could not achieve in the previous experiments with STDP. As you can see the (A) has learned both patterns while (B) has only learned the second pattern. It is definitely a step up however, here we do not have any control over which neuron learns which, this can be problematic. We can solve this issue in the last chapter using **R-STDP**. Even here if we run the program for different initial weights and add a bias term, something like a reward-modulation which we will get to in the next chapter more comprehensively and more scientifically, we would hope for a better ability to distinguish different patterns.

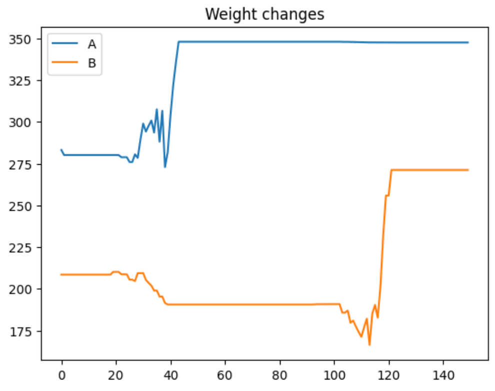
Figure 3.14 shows a successful result and ability to learn different patterns by our network, as we can see the neuron after the $150th$ iteration where the learning stops and we enter the testing phase, we can see that neuron (A) is sensitive towards the first



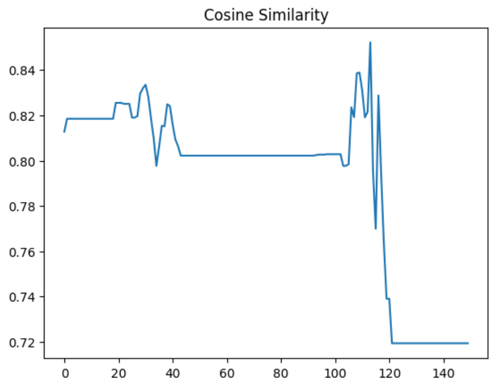
(a) *input layer raster plot*



(b) *output layer raster plot*



(c) *weight changes of output layer*



(d) *cosine similarity of the output layer*

Figure 3.14: *network full random connectivity with STDP learning rule - with lateral inhibition and bias term*

stimuli and (B) has learned the second pattern which is a desired outcome. Although this method might not be highly bio-inspired it gets the job done. There are other bio-inspired more scientific ways to implement this dynamics that we will get into more in-depth in the next chapter.

4 Reward-Modulated STDP

In unsupervised learning and in particular in STDP, the general idea behind its learning is that the network learns the recurring patterns, meaning it learns the patterns and spike trains that are more frequent without knowing if it is good of not. Here in reinforcement learning, with help of conditioning, and reward-based activity, we can have more control and supervision over the learning process enough to inhance its results without using supervised methods. It seems that Dopamine which is a neuro-modulator works as the reward in our brain. Studies have shown that our brain take advantage of this reward-modulated behavior. In this section we intend to implement this learning rule and apply it to the network in Chapter 3 and analyse the different behaviors.

R-STDP learning rule

$$\frac{dc}{dt} = -\frac{c}{\tau_c} + STDP(\tau)\delta(t - t_{pre/post}) \tag{4.1}$$

$$\frac{dd}{dt} = -\frac{d}{\tau_d} + DA(t) \tag{4.2}$$

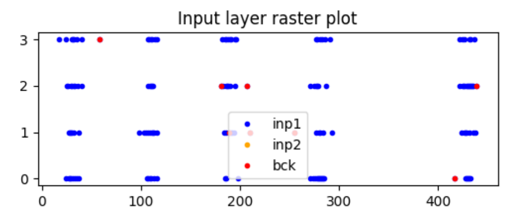
$$\frac{ds}{dt} = cd \tag{4.3}$$

where s is the **synaptic weight**, d is the **dopamine rate** and c is **eligibility trace**.

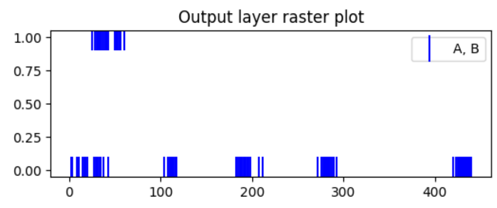
4.1 Single pattern

In this section, we are going to test our first R-STDP based network with a single pattern. Here we reward the network if neuron (A) spikes and we punish it of (B) spikes for the input pattern. Now we expect the network that after the training period, only neuron (A) spikes for the pattern and neuron (B) does not spike idealy.

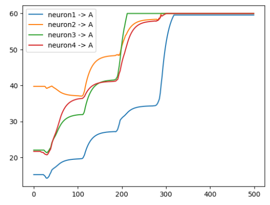
As we can see in Figure 4.1 at first both output neurons (A) and (B) fire a spike but after a while activity of neuron (B) fades away as we can see by their weight changes, weights to (B) goes to zero while weight to (A) goes to 60 which is the maximum synaptic weight. We can say that this network has learned the pattern the way we wanted it but



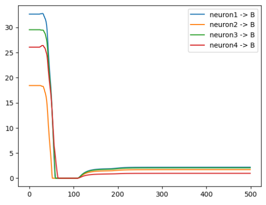
(a) *input layer raster plot*



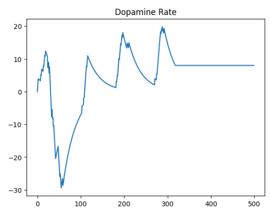
(b) *output layer raster plot*



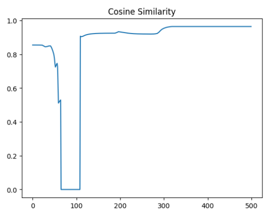
(c) *weight changes of (A)*



(d) *weight changes of (B)*



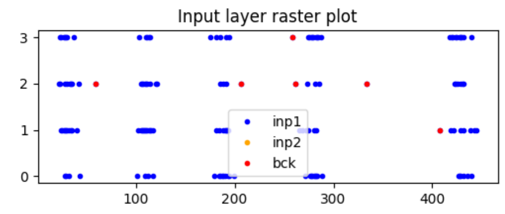
(e) *dopamine rate*



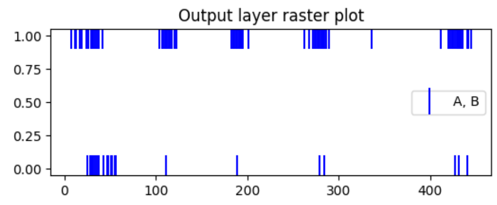
(f) *cosine similarity*

Figure 4.1: *network full random connectivity with R-STDP learning rule - single pattern (A)*

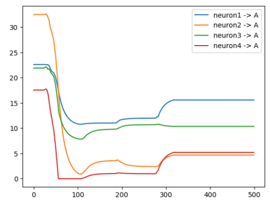
to ensure that the learning is not random, this time we set our target to (B) and expect an opposite result.



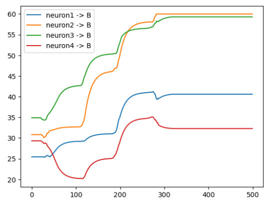
(a) *input layer raster plot*



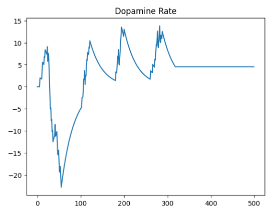
(b) *output layer raster plot*



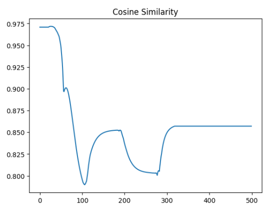
(c) *weight changes of (A)*



(d) *weight changes of (B)*



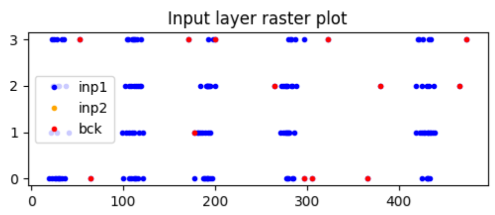
(e) *dopamine rate*



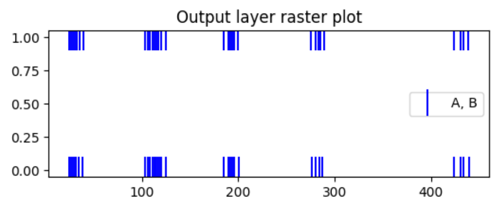
(f) *cosine similarity*

Figure 4.2: *network full random connectivity with R-STDP learning rule - single pattern (B)*

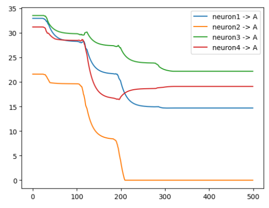
fortunately we can see in Figure 4.2 that the learning is not actually random and is learning by the reward and punishments we give the network, as we can see the activity of (B) has increased while the activity of (A) has decreased which is obvious from both synaptic weights and their spike pattern. Keep in mind that this reward-modulated works fine as long as the rewarding activities and punishing activities happen with a time difference, meaning particularly in small networks if both (A) and (B) spike at the same time approximately, then the network will always punish both activities or reward them at the same time, resulting in the network not learning the ability to distinguish between (A) and (B). In order to see this result we fire the first few spikes of both (A) and (B) at around the same times to see whether (A) and (B) can learn different things or not.



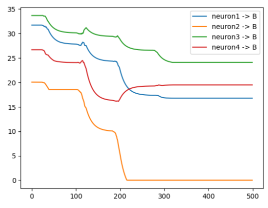
(a) *input layer raster plot*



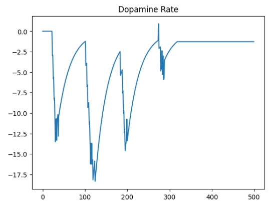
(b) *output layer raster plot*



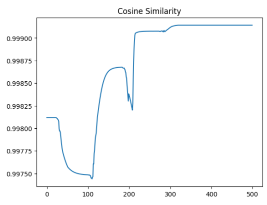
(c) *weight changes of (A)*



(d) *weight changes of (B)*



(e) *dopamine rate*



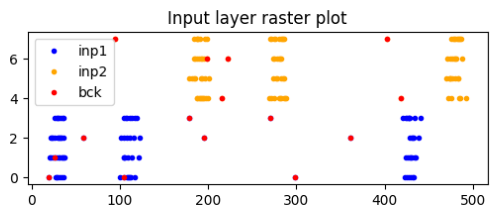
(f) *cosine similarity*

Figure 4.3: *network full random connectivity with R-STDP learning rule - single pattern (B)*

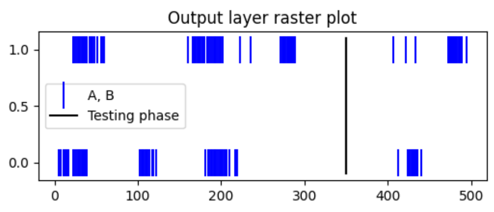
Figure 4.3 is an exterme version of what might happen if all the output neurons spike in sync. As we can see their synaptic weight changes are somewhat identical and so as their spikes, resulting the network not to assign the pattern to only one of (A) or (B) out of the two.

4.2 General patterns

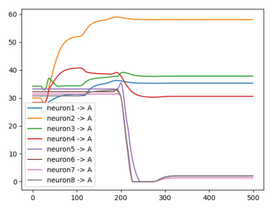
Now that we established that our network with R-STDP learning rule is able to successfully learn with a specific output neuron target, we are ready to test with more patterns. first we study the case of two different patterns without any common pattern, expecting the (A) to learn the first pattern and (B) to learn the second.



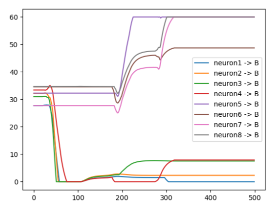
(a) *input layer raster plot*



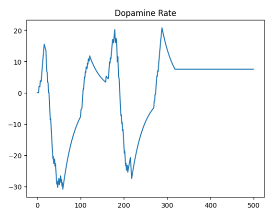
(b) *output layer raster plot*



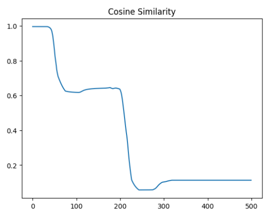
(c) *weight changes of (A)*



(d) *weight changes of (B)*



(e) *dopamine rate*



(f) *cosine similarity*

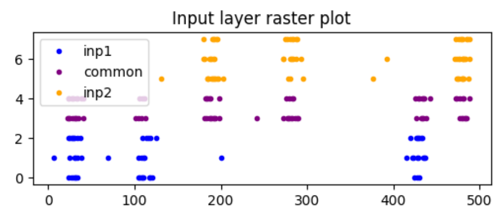
Figure 4.4: *network full random connectivity with R-STDP learning rule - general patterns*

Here Figure 4.4 shows clearly that the network is capable of distinguishing between patterns and assigning the desired output neuron to the specified input stimuli. As we can see the synaptic weights of neurons 1 through 4 to (A) has increased while for 5 to 8 to (A) has decreased and the opposite goes for (B) which is the desired result.

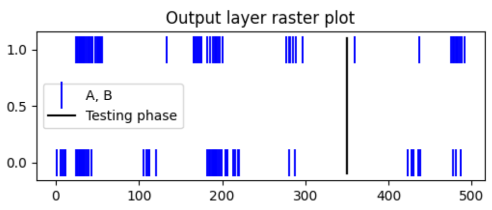
Now we can start to input patterns with common area to see whether the network is able to distinguish them or not.

The results of Figure 4.5 indicates that the network is somewhat still able to learn the patterns and be able to distinguish them in the output neurons (A) and (B) however the silver lining is not as clear as the previous experiment, as you can see the neurons (4) and (5) in the input layer which are common in both patterns, have somewhat weights in the middle and do not particularly learn to respond to specific output neuron but general the network is still able to assign an input to an output neuron. Now lets increase the common area of patterns to see the results.

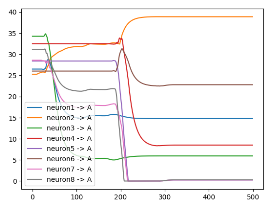
Figure 4.6 shows that the network is somewhat in confusion, meaning due to the fact that the two patterns are highly similar (around 50%) the network is struggling to understand the difference and as a result the weights are not too convincing, fortunately due to the first random spikes of both rewarding and punishing types for the network that we provided, the network is somewhat getting the difference but the line is getting more and more blur and unclear. There are also somecases with that the network is not able to learn different patterns at all.

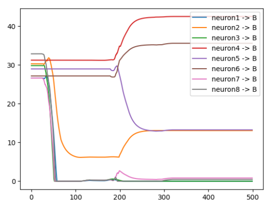


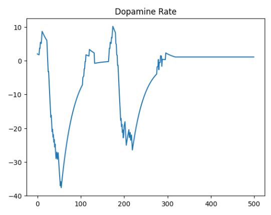
(a) *input layer raster plot*

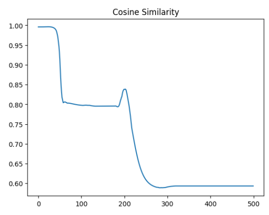


(b) *output layer raster plot*



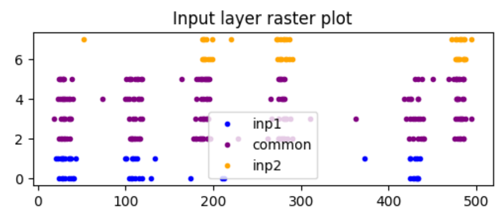




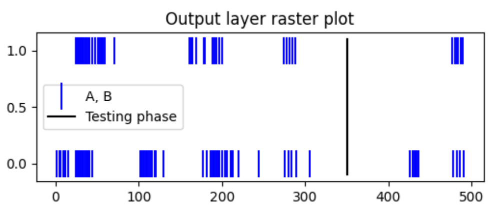


(c) *weight changes of (A)* **(d)** *weight changes of (B)* **(e)** *dopamine rate* **(f)** *cosine similarity*

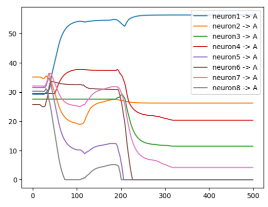
Figure 4.5: *network full random connectivity with R-STDP learning rule - general patterns (low common pattern)*

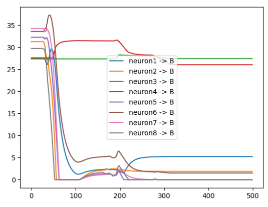


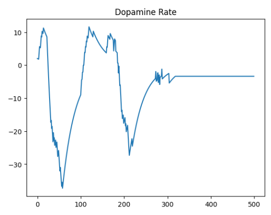
(a) *input layer raster plot*

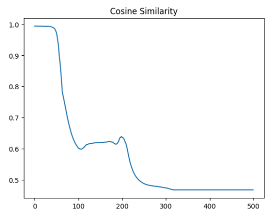


(b) *output layer raster plot*









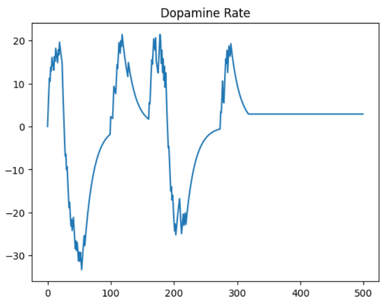
(c) *weight changes of (A)* **(d)** *weight changes of (B)* **(e)** *dopamine rate* **(f)** *cosine similarity*

Figure 4.6: *network full random connectivity with R-STDP learning rule - general patterns (high common pattern)*

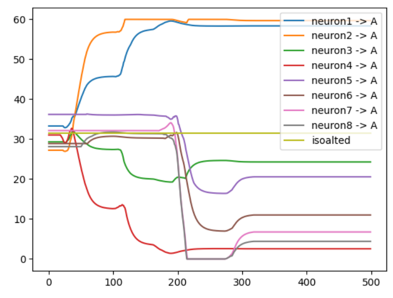
4.3 Isolated neurons

The same as Chapter 3, we add isoalted neurons in both input and output layers to see whether their synaptic weights change or not. Keep in mind that due to the fact that still the basis of our learning is still STDP learning rule, we do not expect any changes in the synaptic weights of isoalted neurons.

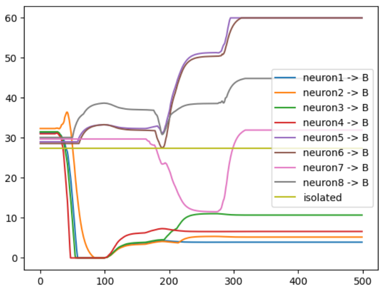
As expected, Figure 4.7 shows that synaptic weight of both isoalted neurons in input and output layer has not changed throughout the simulation, due to the fact that the did not fire a spike the whole time even though the dopamine level is non-zero so the only reason not to change for weight is for their synaptic tag to never change.



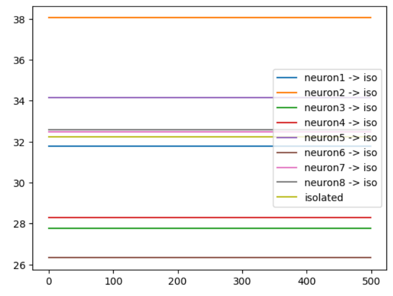
(a) *dopamine rate of network*



(b) *weight changes of (A)*



(c) *weight changes of (B)*

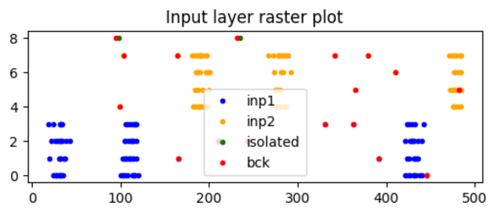


(d) *weight changes of isoalted neuron*

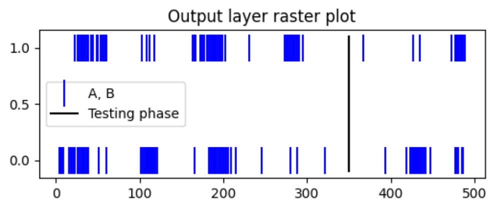
Figure 4.7: *network full random connectivity with R-STDP learning rule - with isoalted neurons in input and output layers*

4.4 Background activity

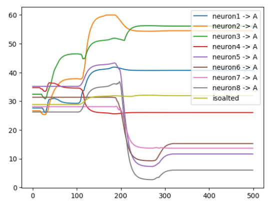
The same as Chapter 3, here we try to add background activity to our network to enable the ability to learn for all neurons to activate and participate in the learning process that were prohibited due to low initial weighting of their synapses.



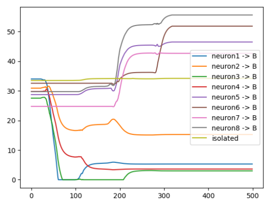
(a) *input layer raster plot*



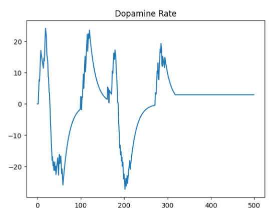
(b) *output layer raster plot*



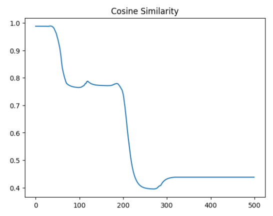
(c) *weight changes of (A)*



(d) *weight changes of (B)*



(e) *dopamine rate*



(f) *cosine similarity*

Figure 4.8: *network full random connectivity with R-STDP learning rule - general patterns - background activity*

Here in Figure 4.8,it seems that this background activity which may be considered somewhat moderate had effect on both learning which as expected added more fluctuation. Also we can see that it enabled the isoalted neurons and their weights start to change in a slight manner.

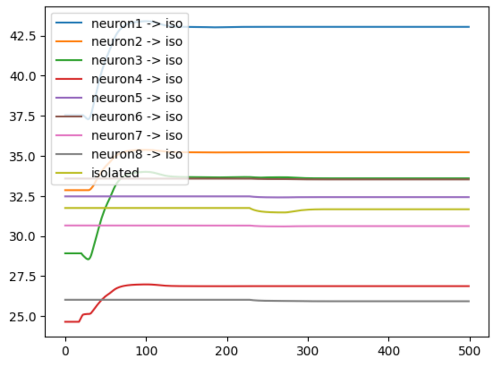


Figure 4.9: *isoalted output neuron (C) weight changes by background activity*

It is obvious that by increasing the background activity the fluctuation gets bigger and bigger and after a threshold, the background activity overcomes the actual input pattern and the spikes comes out as somewhat random, disabling the network to learn the patterns as we saw in the previous chapter.

4.5 In-depth Analysis

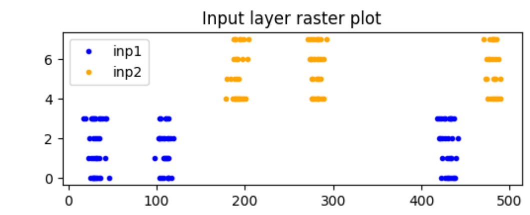
As the last section of this report, we dedicate this part for in-depth analyse and understanding the effects of each parameter in learning process. Here due to the fact that R-STDP method somewhat contains all the base STDP parameters, we decided to use R-STDP as our main learning rule as experiment with different values of parameters.

4.5.1 STDP learning factor A_+, A_-

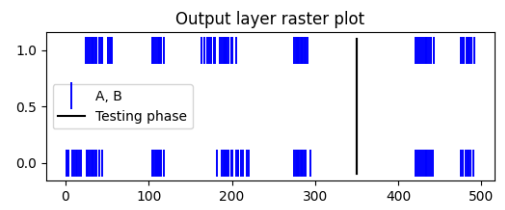
It is common to use relatively low learning factor to avoid abrupt changes and be able to use all the potential weighting settings for different neurons. First we try to use a very low learning factor to see its effect on learning.

Here we can see that unfortunately Figure 4.10 shows that the network did not learn the pattern and did not compelete the assigned task. The reasoning behind it is the fact that as you can see the synaptic weights are changing however because the learning rate is too small, the changes do not accumulate into an effective change of spike pattern in the post-synaptic neurons. Now lets take a more common and moderate learning rate and see if any changes are made.

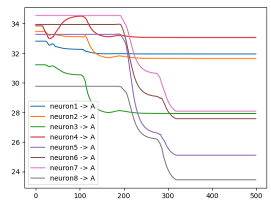
Here in Figure 4.11 the considerable change is compeletely clear. The network has done a great job in learning different patterns and to distinguish them and it was all made possible simply by choosing a more balanced and well-suited learning parameter. We can see what changes lay in the other end of the rope, using extremely high learning rate.



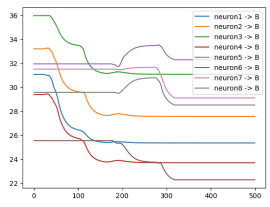
(a) *input layer raster plot*



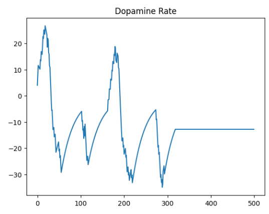
(b) *output layer raster plot*



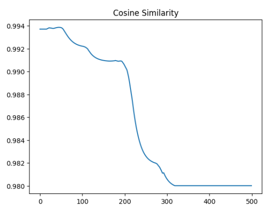
(c) *weight changes of (A)*



(d) *weight changes of (B)*

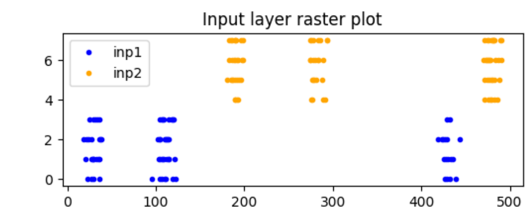


(e) *dopamine rate*

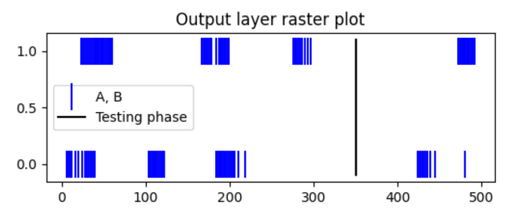


(f) *cosine similarity*

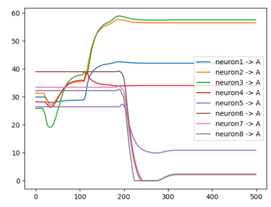
Figure 4.10: *network full random connectivity with R-STDP learning rule - general patterns - low stdp factor*



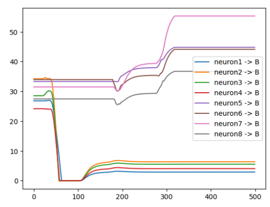
(a) *input layer raster plot*



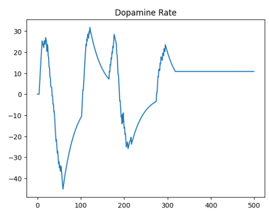
(b) *output layer raster plot*



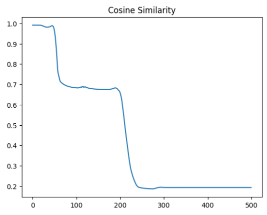
(c) *weight changes of (A)*



(d) *weight changes of (B)*



(e) *dopamine rate*

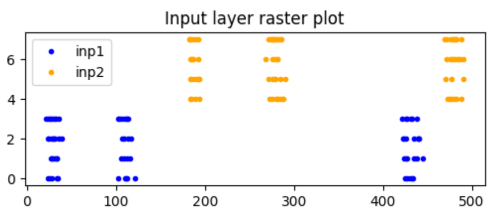


(f) *cosine similarity*

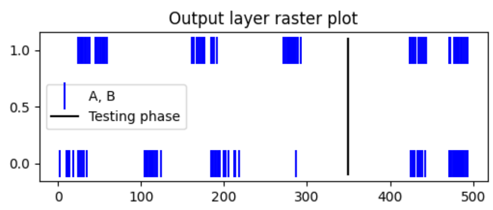
Figure 4.11: *network full random connectivity with R-STDP learning rule - general patterns - balanced stdp factor*

The results in Figure 4.12 is somewhat all expected. Here we can see that although in some periods the network has learned the ability to distinguish between patterns by spikes of (A) and (B), due to high learning rate after a short period of time, with a subtle change of dopamine level or change of input pattern, It acts as the network is restarted and there are no middle ground for synaptic weights to change resulting in a failure of recognising of different patterns by the network.

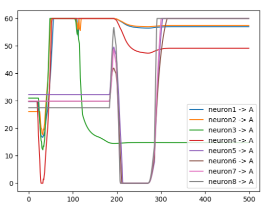
we should keep in mind that bounds of weights are also very important whether using **soft bound** or **hard bound**. Having too close bounds result in all activities somewhat the same and having too wide range for synaptic weights may result in a few very strong connections and others without any effect.



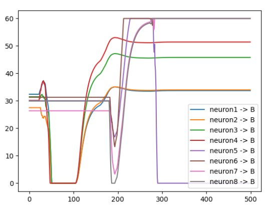
(a) *input layer raster plot*



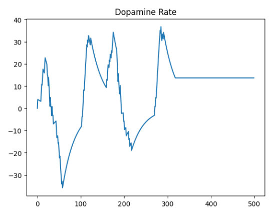
(b) *output layer raster plot*



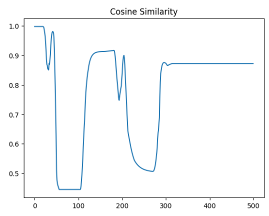
(c) *weight changes of (A)*



(d) *weight changes of (B)*



(e) *dopamine rate*

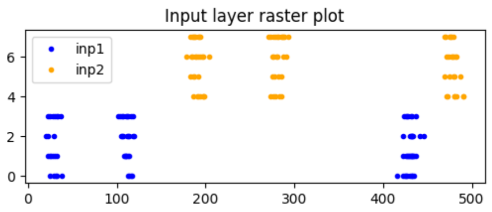


(f) *cosine similarity*

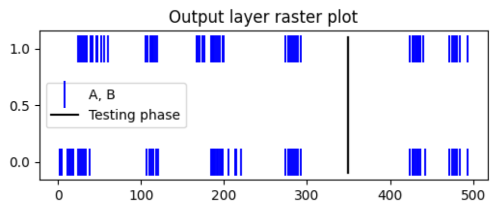
Figure 4.12: *network full random connectivity with R-STDP learning rule - general patterns - high stdp factor*

4.5.2 Reward & Punishment

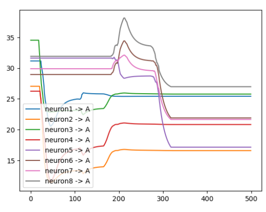
One important parameter in R-STDP is the reward itself. The range of reward and punishment can effect the learning ability of the network the same way the learning rate does. Another way to look at their influence in the network is their proportional values, meaning whether to have balanced values of punishment and rewards or one should be considerable higher than the other. Throughout this chapter we have used the balanced level of punishment and reward. Now we want to change their values. First with reward being higher than punishment by considerable amount.



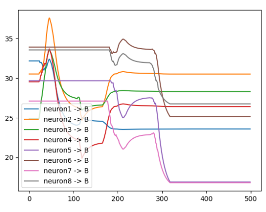
(a) *input layer raster plot*



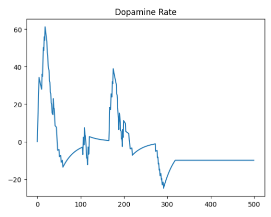
(b) *output layer raster plot*



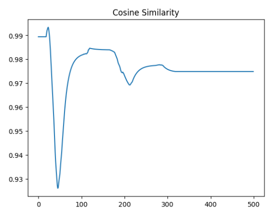
(c) *weight changes of (A)*



(d) *weight changes of (B)*



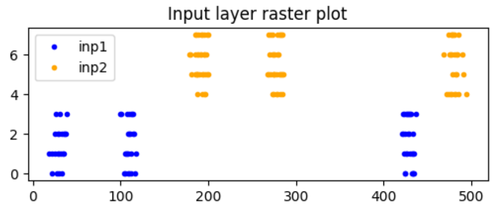
(e) *dopamine rate*



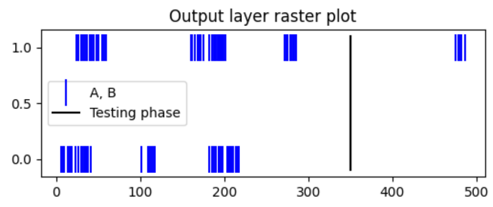
(f) *cosine similarity*

Figure 4.13: *network full random connectivity with R-STDP learning rule - high reward & low punishment*

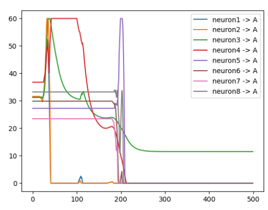
From Figure 4.13 we can deduct the fact that due to unbalanced reward and punishment dynamic the network is mostly in rewarding stauts resulting almost always increasing the synaptic weights and this intrupts the learning producure. Now we can see the other way around.



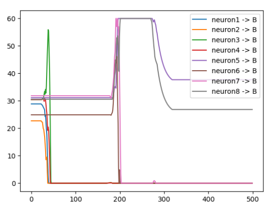
(a) *input layer raster plot*



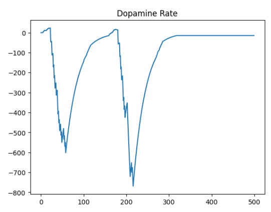
(b) *output layer raster plot*



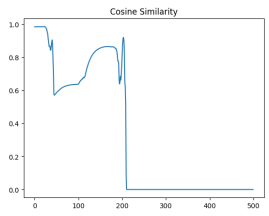
(c) *weight changes of (A)*



(d) *weight changes of (B)*



(e) *dopamine rate*



(f) *cosine similarity*

Figure 4.14: *network full random connectivity with R-STDP learning rule - low reward & high punishment*

On the other hand, Figure 4.14 shows the other end of the rope which is, due to high level of punishment across from learning, if the network makes the slightest mistake the punishment goes so high, resulting the network to go down in activity and sometimes do not learn the pattern at all. As it shows, the first pattern is not learned by neither (A) nor (B).

5 Conclusion

This report is the final of the three report series as basics of every computational neuroscience model. We first started by implementing single spiking neuron models, then we used groups of neurons to build our first neural population and spiking neural network (SNN). In this report we added the last piece to the puzzle of our SNN by adding the learning ability, using both unsupervised learning (STDP) and reinforcement learning (R-STDP). With applying these learning rule and implementation of neural encoder and decoders we are now ready to dive into a specific case study and work with real data to achieve goals like classification and object detecion using SNNs.

This page intentionally left blank.

UNIVERITY OF TEHRAN

ENCODING, LEARNING PLASTICITY,
STDP AND R-STDP LEARNING RULE

ARASH NIKZAD
Student No. 610301195

TEHRAN, APRIL 2024