Liquid State Machine

Dual Degree Phase 2 report
Submitted in partial fulfillment of
the requirements for the degree of
Dual Degree
by

Devesh Kumar
(16D070044)

Department of Electrical Engineering
Indian Institute of Technology Bombay
Mumbai 400076 (India)

4 July 2021

Acceptance Certificate

Department of Electrical Engineering

Indian Institute of Technology, Bombay

The seminar report entitled “Liquid State Machine” submitted by Devesh Kumar (16D070044) may be accepted for being evaluated.

Date: 4 July 2021

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I declare that I have properly and accurately acknowledged all sources used in the production of this report. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Devesh Kumar
(16D070044)

Date: 4 July 2021

Abstract

In my dual degree, I have worked on scaling a Liquid state machine and applying it to the different and difficult data sets. An LSM is a type of machine learning algorithm that is based on Spiking neural networks. These spiking neurons resemble our brain more closely than traditional neurons. A reservoir is a collection of randomly connected neurons used in LSM. The LSM has non-linear dynamics which make it good for continuous data like speech. Another benefit of an LSM is that they have a simple structure like RNN thus they have lower training complexity. But due to this simple structure, their accuracy is lower than its counterparts. Here we show that to increase the accuracy of LSM we can increase the size of the reservoir. A larger reservoir can solve a more complex problem but it requires more computational power. Here we also show that how we should scale the reservoir when we increase the class size. I have tested LSM architecture on two dataset ie google keyword dataset and TI 46 dataset. The results of Ti46 were very good and I got an accuracy of above 70 . But as google dataset was much difficult could not produce as high accuracy as was found in TI 46.

In my Dual Degree project, I also explored predictive coding. In predictive coding weight update, the rule is local. Weights are updated based on the error which can be updated in parallel.

Table of Contents

List of Figures

Chapter 1

Background

1.1 Spiking Neural Network

Spiking neural networks are the next generation of neural networks where information is coded in spikes. These are more biological possible neural architecture. SNN can exploit the temporal information as these discrete spikes occur at different times. All the present-day artificial neural networks are time-invariant, their values do not change with time.

A neuron is a basic building block for and Spiking neural network. Biologically neuron has dendrites. These dendrites collect information that which neuron in the neighborhood has spiked. If a spike is received it is passed to soma. If the total no of the spike is greater than the threshold then the neuron produces an output spike, which may act as input to connected spikes. So the output spike depends on this threshold value if the threshold value if this threshold value is very large then a few spikes only occur. There is also a refractory period of a neuron. It is like a resting period after a spike. During this time no output spike occur even if there is input spikes received.

1.1.1 Action Potential

The spikes are called action potential. When the membrane potential crosses the threshold then the neuron emits short electrical pulses. During a spike, the potential of a cell rises and falls very quickly within milliseconds.

One neuron can be connected to many similar neurons. If there are multiple synapses at a particular time then the neuron does the summation of these spikes. These spikes can be scaled depending on strength of the synapse.

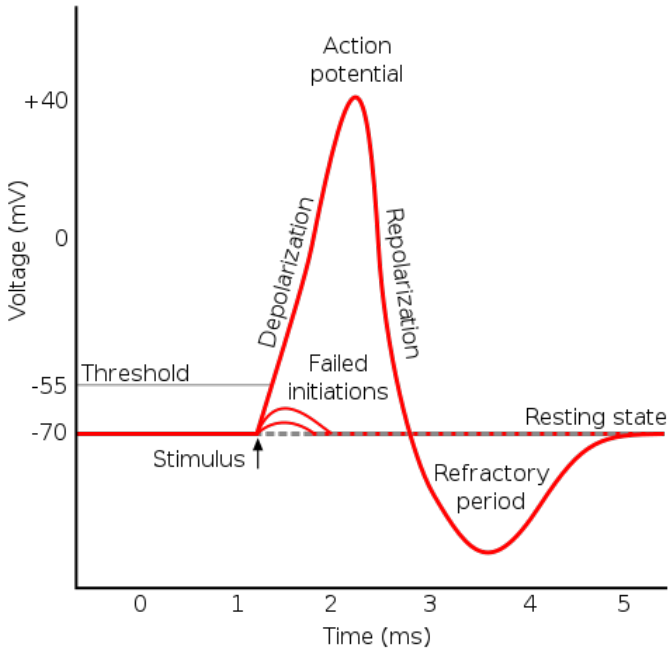


Figure 1.1: Action Potential (11)

1.1.2 Synapse

Synapse is the strength of the connection between two neurons. There can be two types of synapse:

- Inhibitory synapses: These are the connection between neurons in which a spike in presynaptic neuron drive the membrane potential of postsynaptic neuron away from the threshold. In other words, this connection makes it difficult for a postsynaptic neuron to spike.
- Exhibitory synapses: These are the connection between neurons in which a spike in presynaptic neuron drive the membrane potential of the postsynaptic neuron towards the threshold.

1.1.3 Leaky Integrate-And-Fire (LIF)

LIF is a type of integrating and fire neuron. These type of neurons adds up (integrates) all the synapses that they receive and when the membrane potential is above a threshold then the neuron fires.

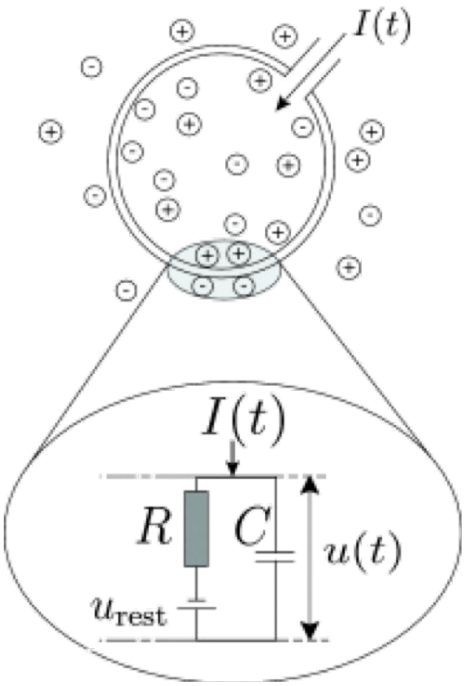


Figure 1.2: Leaky Integrate-and-Fire Model (10)

Here $\mu(t)$ is the value of membrane potential at any time t. $\mu_{rest}(t)$ is the resting potential of the neuron. $I(t)$ signifies that whether neuron has received any spike. This $I(t)$ increases the membrane potential. The cell membrane act as a capacitor that stores charge due to input current(input synapse). The loss term in the membrane is denoted by resister which is responsible for leakage. Whenever $\mu(t)$ reaches threshold it is set back to $\mu_{rest}(t)$

Through KVL we can write.

$$I(t) = \frac{\mu(t)-\mu_{rest}(t)}{R} + C\frac{\delta u}{\delta t}$$

This differential equation is only valid before the neuron spikes.

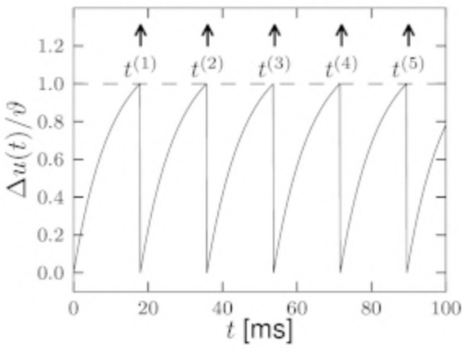


Figure 1.3: Membrane potential of a neuron (10)

Chapter 2

Literature Survey

2.1 Liquid State Machine

A Liquid Reservoir is a set of sparsely randomly connected neurons. Each neuron does a spike base operation and is models as a Leaky Integrate-And-Fire neuron. The arrangement of the neurons in a reservoir can be imagined as a cube of size $nXnXn$ with each side having n neurons and the whole cube has n^3 neurons in total.

The Reservoir translates the input of low dimension to higher dimension through large interconnected neurons. This higher dimension data is then used by the output layer to predict the class of the data. As the size of the Reservoir increases it can project to an even higher dimension and in theory, would be able to solve much more complex problems.

After the preprocessing the input is converted to some sound channels. Each sound channel represents some characteristic of the data point. Each channel is spike encoded. For each channel, the spikes are inputted to a randomly selected neuron in the reservoir. These neuron dynamics operate as Leaky Integrate-And-Fire neurons. These create the output spikes which are input to some other neuron depending on the weights.

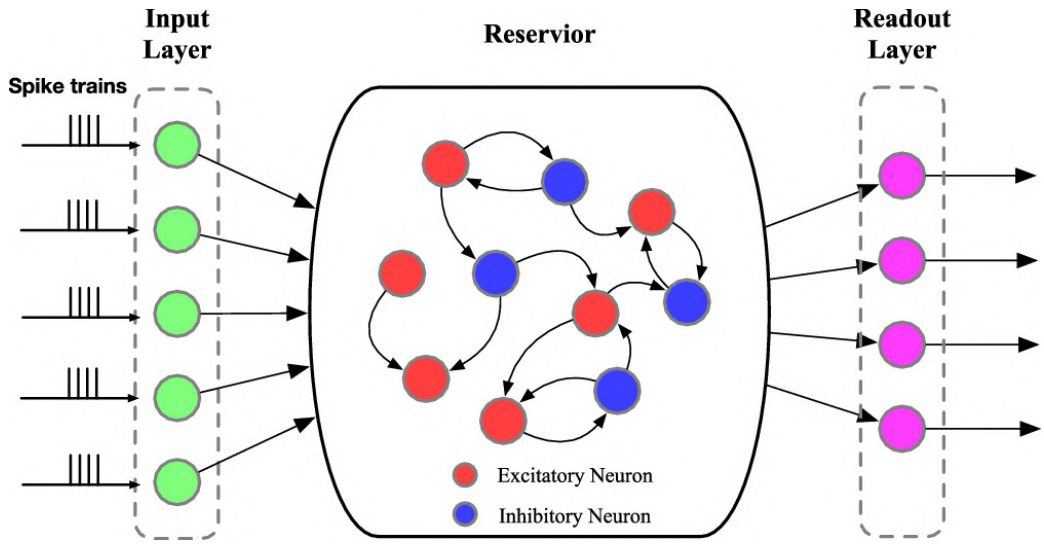


Figure 2.1: Reservoir(14)

2.1.1 Connections

A reservoir has two type of connections ’

- Excitatory Neuron
- Inhibitory Neurons

The synaptic weight between neurons can be of four types weight between two excitatory neurons, weight between two inhibitory neurons, weight between inhibitory neuron and excitatory neuron, weight between excitatory neuron and inhibitory neuron.

Whether a connection exists between two neurons depend on the distance between two neurons. If neurons are close by then there is a higher chance that they are connected. It follows a probabilistic model and the probability of connection between two neurons (N1, N2) given by the following equation.

$$P(N1, N2) = K.e^{-\frac{D^2(N1,N2)}{\lambda^2}}$$

Here $D^2(N1, N2)$ represent equilin distance between two neurons. the constatant K can take four different values depending on the connection. like K_{EE}, K_{II}, K_{EI}, K_{IE}
The synapse follow a second order dynamics ans is given by following eqn.

$$v = \frac{1}{\tau_1 - \tau_2}(e^{-\frac{t-t_s}{\tau_1}} - e^{-\frac{t-t_s}{\tau_2}})H(t - t_s)$$

If we increase the weights the spiking in the reservoir will be more and if we reduce the weights the spiking will be less. At ideal conditions we want the spiking to be optimal so that the reservoir is on edge of chaos. Poor performance is observed when there is too much spiking.

2.1.2 Network Architecture

The network consists of three parts preprocessing layer, a Liquid state machine, and an output layer.

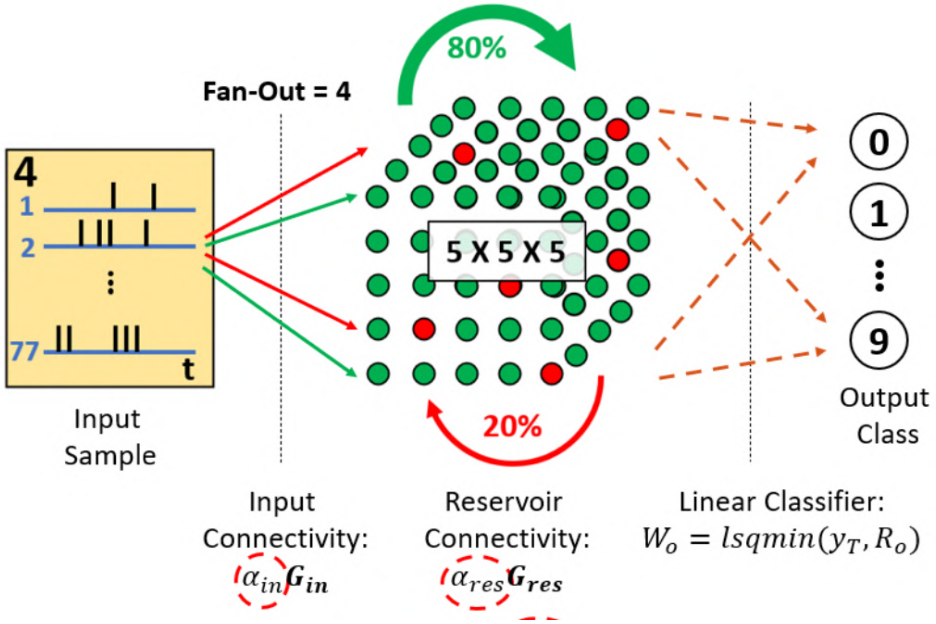


Figure 2.2: Scaling factor $\alpha_{Gin} and \alpha_{Gin}$ (3)

At first, the speech is re-sampled to 12.5kHz as Lyon's ears are optimized for this sampling frequency. The TI dataset is initially sampled with 12.5kHz while the Google dataset is initially sampled with 16kHz which is subsequently down-converted to 12.5kHz. All samples are about 1 sec long.

The Lyon's ears convert the speech samples into 77 audio classes. Each audio channel is spike encoded these 77 spikes streams are applied to randomly selected excitatory neurons of the reservoir. The reservoir runs through LIF neuron dynamics. After 1 second of computation in the reservoir the spikes of each neuron are recorded.
All the neurons are then connected output layer. The no of output layer neurons depends on the no of classes in the problem. It is a dense connection ie every neuron of the

reservoir is connected to every neuron of the output layer by a weight w_{ij}. The weight learning happens only in the output layer.

The weight update only happens in final layer. As the final layer is a fully connected layer so there are $Nres * classsize$ number of trainable parameter. Matlab's lsqlin function is used to minimize the following loss function for a output class k.

$$\frac{1}{2}|\sum_{i=Train}(\sum_{j=N^{res}} n_{ij}^{spike} W_{jk} - Y_{ik})|$$

So for each output neuron k, this function represents the no of times that neuron k output was incorrect. This minimization step is repeated for every neuron in the output layer. Here n_{ij}^{spike} represent the total no of times jth neuron of reservoir spiked when ith training example is put through it. W_{jk} represents the weight scaling of connecting the jth neuron of the reservoir to the kth output neuron. Here Y_{ik} represents the ideal output of kth output neuron for ith input. For example, if we have an input representing class 4 then all the neurons except the fourth neuron should be zero, and the Fourth neuron should be 1. So in each iteration the all the weight of one output neuron is set. The weights are limited to W_{lim} to $-W_{lim}$.

Now, these weights are used for testing. The output spikes from the reservoir are scaled through these weights and are applied at the output layer. The output layer is the LIF neuron. The output of the model is the neuron that spiked the most number of times.

2.1.3 Parameter Used

Parameter	Value
Ratio of excitatory to inhibitory neurons	4:1
Fan out	4
W_{EE}, W_{EI}, W_{IE}, W_{II}	3, 6, -2, -2
K_{EE}, K_{EI}, K_{IE}, K_{II}	0.45, 0.3, 0.6, 0.15
λ	2
W_{lim}	8,-8
τ_{1E}, τ_{2E}	8ms, 4ms
τ_{1I}, τ_{2I}	4ms, 2ms

2.2 Data pre-processing

Lyon passive model is used for preprocessing which is followed by BSA filter which converts this analog signal to spikes. Lyon's passive model is mainly used to mimic how

humans hear sound with their ears. It consists of an outer prefilter, notch filter, rectifier, and automatic gain controller. The filtered sound is then converted to spikes with the Bens Spiking Algorithm (BSA). It is a very good tool for speech enhancement and sound separation.

After data prepossessing the input data is converted into 77 audio class, each class representing same feature of the input dataset. Each class is spike encoded where intensity of spiking represent how much a particular frequency is present in the data.

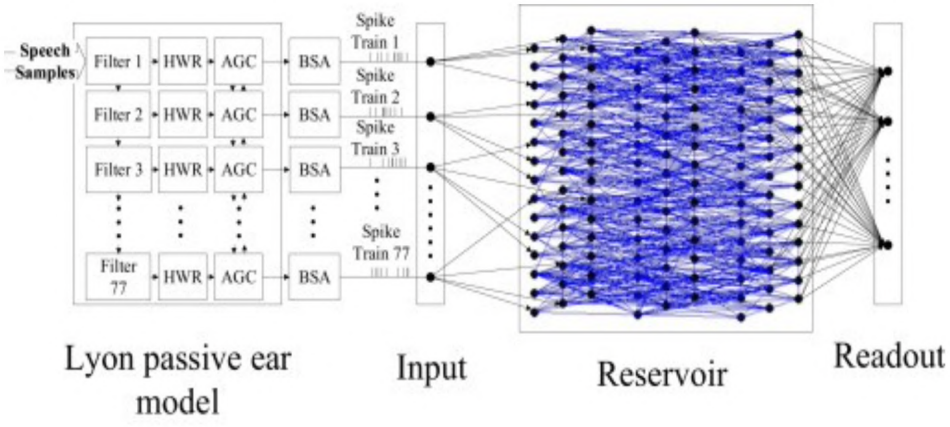


Figure 2.3: Network for LSM(3)

The preprocessing step involves two step:

- Lyon passive model
- Spike Encoding

2.2.1 Lyon passive ear model

It is a part of the Matlab auditory toolbox and it is used to visualize how human ears interpret sound. It is a mixture of series of filters half-wave rectifier and AGC models. These filters are tuned for a sampling frequency of 12.5kHz. More the number of cascade filters more will be the effectiveness of the Lyon passive ear model. This model simulates cochlea, the inner part of our ears.

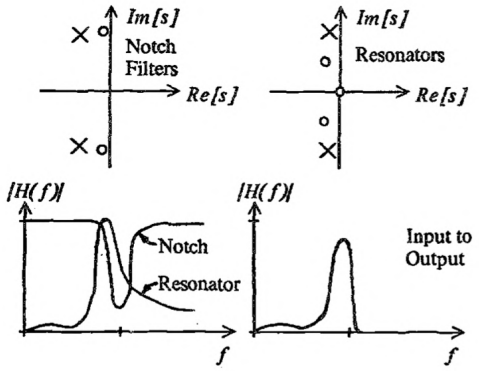


Figure 2.4: Filter used in Lyon passive ear model

The input signal is first passed through series of cascaded filters. Each of the filters will produce an audio class. Each filter bank comprises a notch, resonator, and rectifier. Each filter has a central frequency f and a bandwidth associated with it. The bandwidth increases as the central frequency increases.

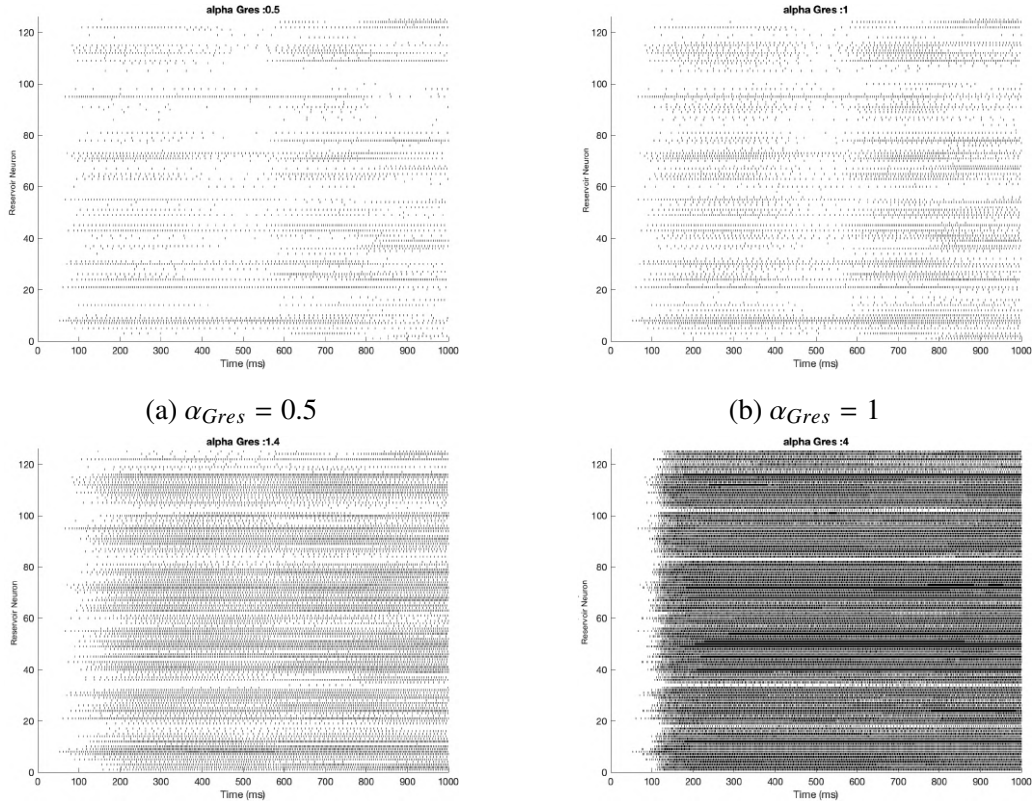
After filtering the signal is next passed through an automatic gain control. It consists of four stages. Each gain has a time constant and the amplification value is dependent on this time constant. The value of this time constant depends on the previous output of all the channels. The value of the time constant decreases as the stages increase.

Chapter 3

Reservoir Parameters

3.1 Effect of α_{Gres} on spiking of neurons

The α_{Gres} represents the scaling of weights within the reservoir. Increasing α_{Gres} is increasing the weights of all the synapses in the reservoir. Thus both inhibitory and excitatory synapses increase. But as the no of the excitatory neuron is more in no thus there is the overall increase in the spiking activity of the reservoir. For optimal operation, we choose the value of α_{Gres} where there is optimal spiking. In the figure, we can see that there is too much spiking for $\alpha_{Gres} = 4$. So from this, we may conclude that that range of α_{Gres} should be between [0.5,4]. There is no need to check beyond this range of α_{Gres}. To generate this below figure The α_{Gin} is set to 1 and only the α_{Gres} is varied.



(a) $\alpha_{Gres} = 0.5$

(b) $\alpha_{Gres} = 1$

(c) $\alpha_{Gres} = 1.4$

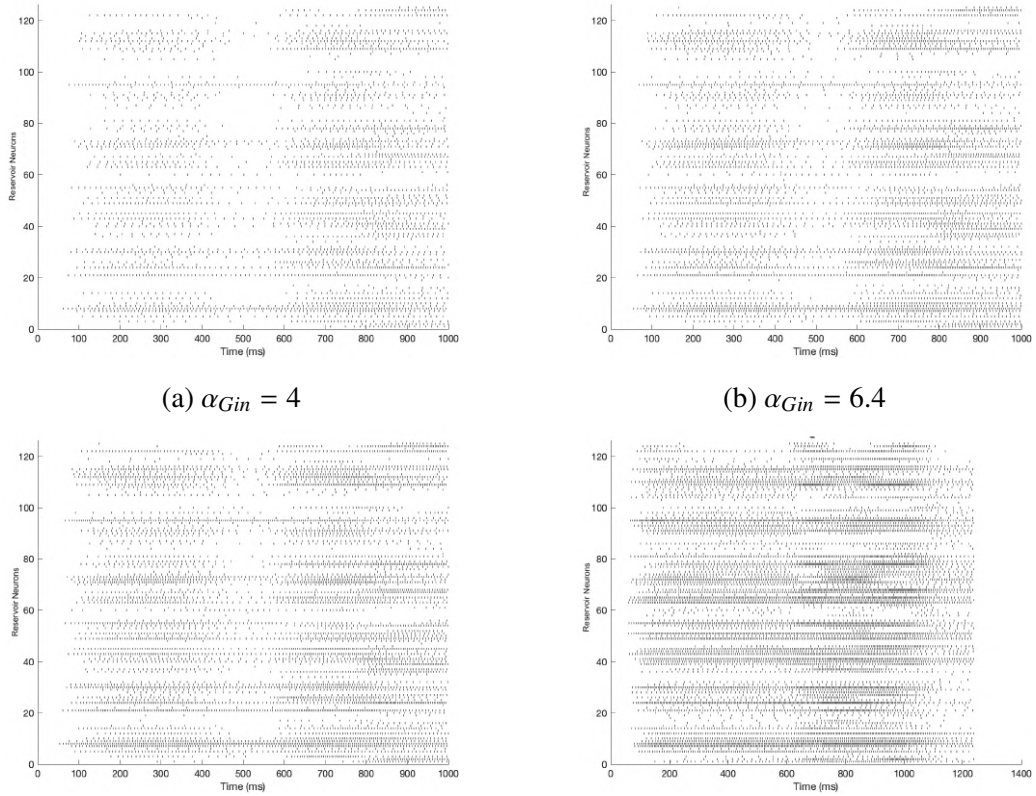
(d) $\alpha_{Gres} = 4$

Figure 3.1: Spiking of Reservoir with α_{Gres}

3.2 Effect of α_{Gin} on spiking of neurons

The α_{Gin} represents the weight of the synapse between preprocessing layer and the reservoir. To generate the above figure we kept the α_{Gres} constant at 8 and varied just the α_{Gin}. We see a similar trend in α_{Gin} as we saw in α_{Gres}. We can similarly conclude that the optimal operating range is [4, 11.2].

From the above figure, we can see that spiking activity is more sensitive towards the α_{Gres} compared to the α_{Gin}. α_{Gin} only affects the 77 neurons that the input layer is connected but α_{Gres} affect all the neuron in the reservoir.



(a) $\alpha_{Gin} = 4$

(b) $\alpha_{Gin} = 6.4$

(c) $\alpha_{Gin} = 8$

(d) $\alpha_{Gin} = 11.2$

Figure 3.2: Spiking of Reservoir with α_{Gin}

3.3 Reservoir and class size

When we consider Ti10 and Ti36 we can conclude that increasing class size increases the problem difficulty. This problem can partially be solved by increasing the reservoir size.

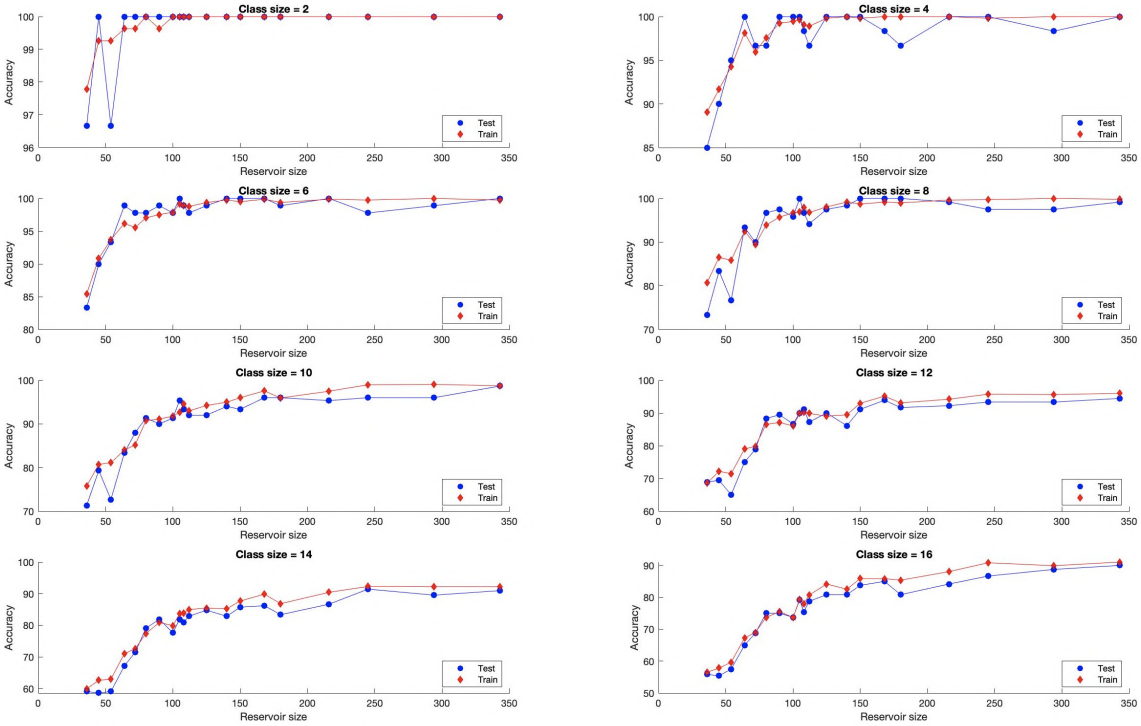


Figure 3.3: Reservoir size vs class size

The accuracy tends to saturate as we increase the reservoir size. This can be easily seen in the graph above. The above graph is formed using the optimal parameters for the reservoir obtained before. here I varied the class size for various sizes of the reservoir. Here I have chosen cuboidal reservoir as its relaxing equal size length constraint would give be much more data points.

Here we can also notice that for each class the accuracy saturates above 90 percent. We can determine the minimum neurons required to obtain an accuracy of above 90 percent by simply extrapolating the graph below.

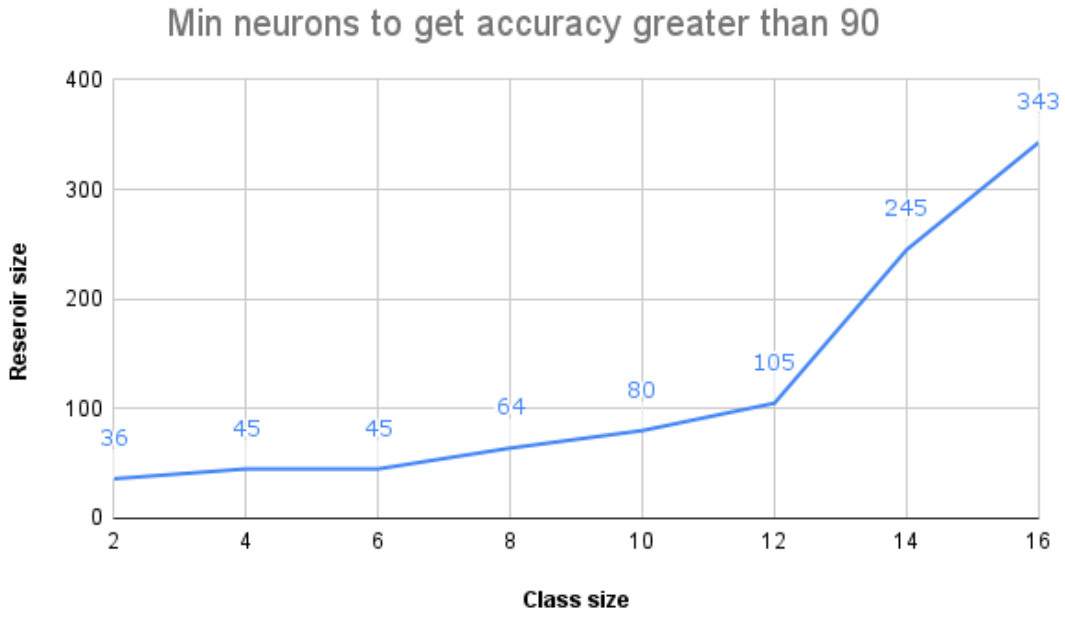


Figure 3.4: Reservoir size vs class size

Chapter 4

Results

I tested the LSM on two datasets TI-10, Ti 36, and google keyword dataset. To get the optimal result I varied α_{GIn}, α_{Gres}, input data size, Reservoir Size.

4.1 Ti-10

Ti 10 is a speech data set that has 10 classes from zero to nine. It has a total of 16 different speakers 8 male and 8 female speakers. Each class has about 650 data points.
As the output class is only 10 classes therefore I do not need a large input data. Very high accuracy can be achieved even by taking only 50 data points for each class making the total input size to be 500. Here 400 data are used for training and 100 are used for testing. And as it has only 10 classes we achieve high accuracy even for a reservoir size of 125.

In fig4.1 we can see that we can achieve very high accuracy even if our reservoir size is small. Here increasing reservoir size does not make sense. Optimal parameter for TI10 are:

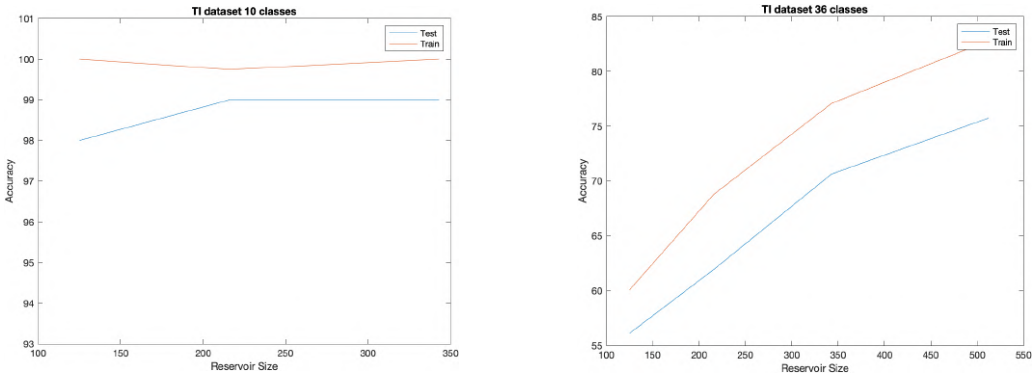
- Reservoir size = 125
- α_{Gres} = 1
- α_{Gin} = 8

Optimal testing accuracy of 98 percent is achieved with training accuracy of 99 percent.

4.2 Ti-36

As the no of classes increases the classification problem becomes more difficult. To accommodate a class size of 36 I increased the no of output layer neurons to 36. I also increased the no of speakers to 15. I also optimized over no of input data used, α_{Gin},

α_{Gres}, size of the reservoir. For this optimization, I had used only one fold and a train test ratio of 9:1 ie there is 9 training example for each testing. I have used only one fold for optimization as during optimization I had to train the model multiple times using different parameters and using folds would significantly increase the training time.

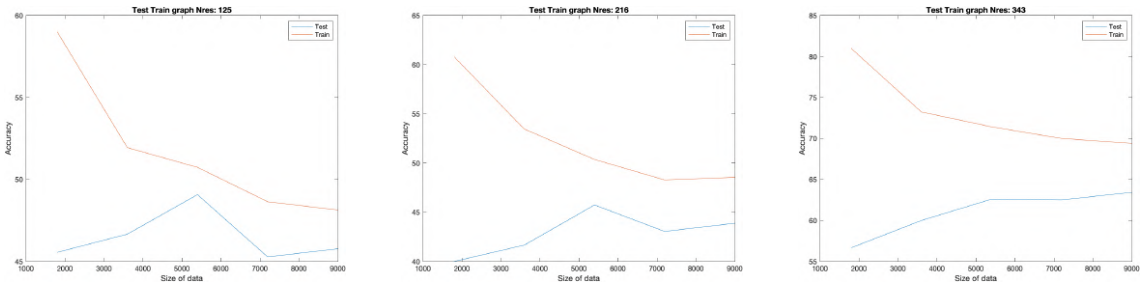


(a) Class size 10 (b) Class size 36

Figure 4.1: Accuracy for Ti10 and Ti36

Here in this graph, we can see that very high accuracy is achieved for TI 10 (digits dataset) even with a lower reservoir size but a TI36 can not achieve that accuracy even with a higher reservoir size.

4.2.1 Input Data size

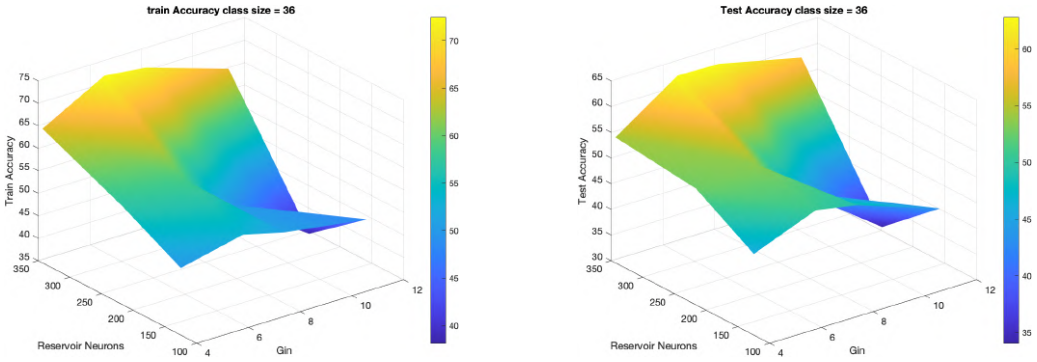


(a) Reservoir size 125 (b) Reservoir size 216 (c) Reservoir size 343

Figure 4.2: Variation of accuracy with input data

We can see through these graphs that when the input size is low then there is a large gap between testing and training accuracy suggesting that the reservoir has not generalized well. But when we increase the data size we see that the gap between testing and training is less suggesting that the network has generalized well. From these graphs, we can conclude that with about 5000 data points we can achieve a high enough accuracy.

4.2.2 α_{GIn}



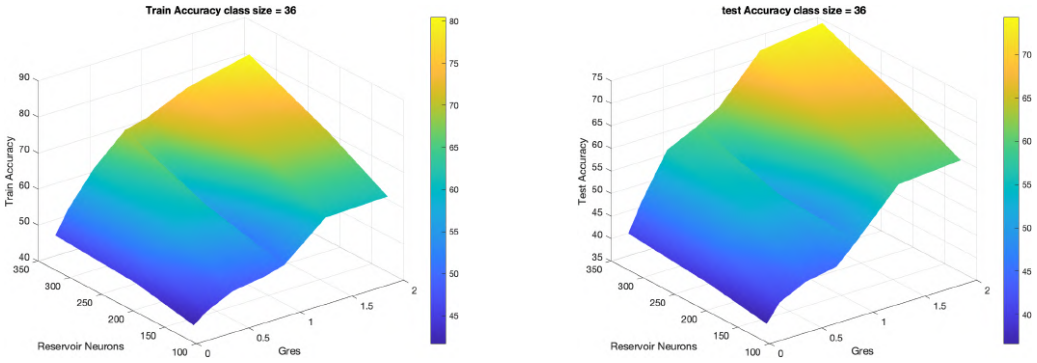
(a) Training Accuracy (b) Testing Accuracy

Figure 4.3: How accuracy varies with reservoir size and α_{GIn}

We can see here that Gin = 8 is an optimal parameter. If gin is very less then there will be very little spiking hence low accuracy. If gin is very high then there will be too much spiking which will also result in low accuracy. We also see that the accuracy increases as the size of the reservoir increases. A large reservoir is more capable of solving a difficult problem.

4.2.3 α_{Gres}

The α_{Gres} determines the weights of synapse between neurons of the reservoir. Earlier we saw that spiking was very sensitive to α_{Gres}. So, therefore, we can not increase the α_{Gres} too much as if we increase there will be too much spiking and this will lead to a fall in accuracy.



(a) Training Accuracy (b) Testing Accuracy

Figure 4.4: How accuracy varies with reservoir size and α_{Gres}

Here we can see that α_{Gres} = 2 gives a high accuracy (74 test and 79 train) for a reservoir size of 7.

So maximum accuracy achieved = 74 test, 79 train

Parameters :

- input size = 5400
- α_{Gin} = 8
- α_{Gres} = 2
- Reservoir size = 343

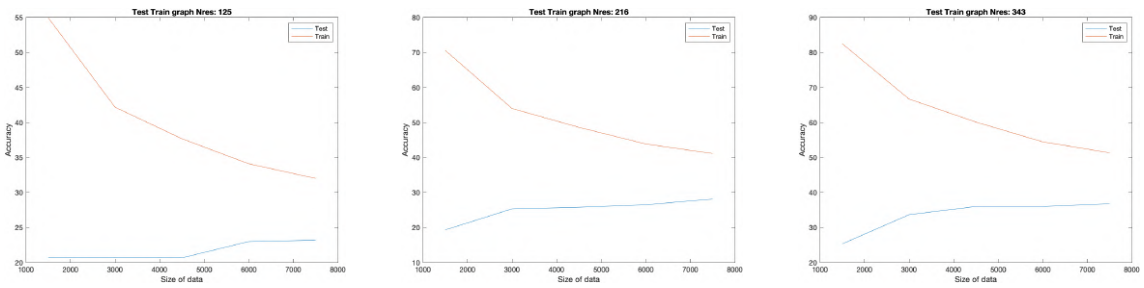
I ran for 5 fold for this combination of parameters and got average accuracy of **73.0555 tests and 80.9954 train accuracy**

4.3 Google keyword

Google keyword is a dataset of 30 commonly spoken words. It includes words like bed, bird cat, dog, down, eight, five, four, go, happy, house, left, Marvin, nine, no, off, on, one, right, seven, stop, three, tree, two, up, wow, yes, zero. Each class has about 1500 samples of one-second-long spoken words. Each data for a class is spoken by a different speaker. I have used only a part of this dataset as using the whole dataset would require GPU. I have used about 150 samples per class. The dataset was initially sampled at 16 kHz but I down-sampled the data to 12.5 KHz as the Lyon ear model was optimized for a 12.5 kHz sampling rate.

Google keyword dataset is an even more challenging dataset as in this dataset there are many speakers for each class. As there are many speakers thus it would be more difficult for LSM to generalize as different people have different accents and tones.

4.3.1 Input Data

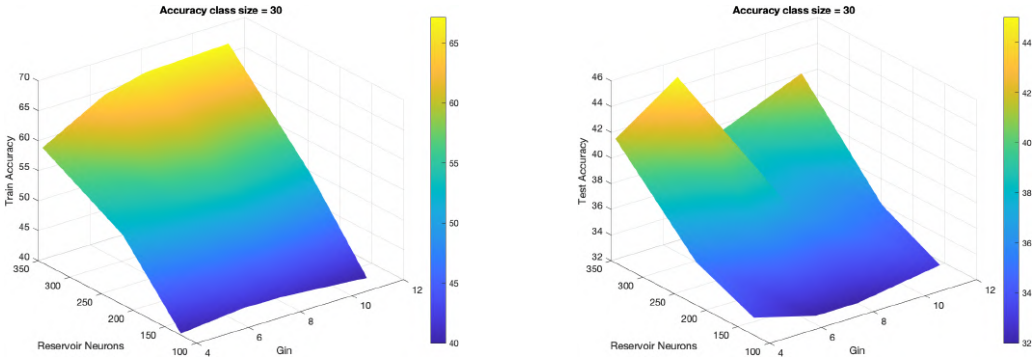


(a) Reservoir size 125 (b) Reservoir size 216 (c) Reservoir size 343

Figure 4.5: Variation of accuracy with input data

We can see that the gap between training and testing accuracy is decreasing as the number of input data is increasing. These suggest that the network is generalizing. We can also see that reservoirs with more neurons require more data. But for very high input data the gap between training and testing is significant. This suggests that the network has not generalized well.

4.3.2 α_{GIn}

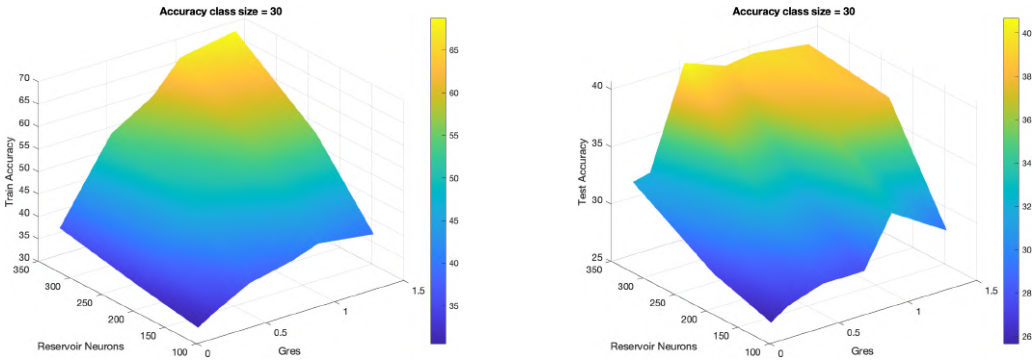


(a) Training Accuracy (b) Testing Accuracy

Figure 4.6: How accuracy varies with reservoir size and α_{Gin}

From this, we can conclude that α_{GIn} has an optimal value for the google dataset is 6.4. We also see that the accuracy is increasing as the size of reservoir scales.

4.3.3 α_{Gres}



(a) Training Accuracy (b) Testing Accuracy

Figure 4.7: How accuracy varies with reservoir size and α_{Gres}

From this, we can conclude that α_{GIn} has an optimal value for the google dataset is 1. We also see that the accuracy is increasing as the size of reservoir scales.

When we compare the result of google keyword and TI we see that TI gives much better accuracy. One of the key reasons for this is the fact that there are lot more speakers in the google keyword dataset compared to the TI dataset. In the Google keyword, every datapoint for each class is spoken by a different speaker whereas in the case of TI there are only 16 speakers. More speakers mean more accent and amplitude to generalize. As there are more speaker makes the job of LSM difficult to generalize.

So maximum accuracy for Google dataset achieved = 40 test, 55 train
Parameters :

- input size = 5400
- α_{Gin} = 6.4
- α_{Gres} = 0.5
- Reservoir size = 343

Chapter 5

Comparison with CNN

Speech classification is in literature by using a convolution neural network. A convolution neural network is usually used for image classification and feature extraction.

In a convolution neural network, Mel-frequency cepstral coefficients (MFCCs) are extracted from the audio as a part of preprocessing. MFCCs is a feature extraction step and is done because convolution can not be directly applied to a time-varying signal. For LSM we generally use Lyon ears as a part of preprocessing, this converts the audio into audio classes which can be used up by the reservoir.

The CNN architecture represents three convolution layers. Two convolution layers have a size of (32*(2*2)) and one of size (32*(3*3)). Each convolution layer is followed by a max-pooling layer and a relu activation function. with a kernel size of 3*3. A max pool layer to reduce the computation of size 2*2. These are then followed by two fully connected layers the last fully connected layer has a softmax layer at the end for classification. In comparison, the LSM just has a 125 neuron size reservoir which is followed by a fully connected output layer of 10 neuron size. Here we can see that the CNN architecture has much more layers or in other words is much more deep compared to LSM. The LSM tries to mimic this deep network through the reservoir. This reservoir helps the LSM to extract enough features so that classification could be done in an output layer only

To compare the accuracy of the two architecture we use the TI dataset for comparison. A 98 percent accuracy can be achieved with a convolution neural network while 95 percent accuracy can be achieved with LSM architecture(for 10 classes). While no trainable parameter was significantly higher for a convolution neural network compared to LSM. An LSM has 1250 trainable parameters while a convolution neural network has

more than 18.5k parameters.While training inn LSm we use only 1 epoch but in CNN we use 70 epoch. Here we can see that the CNN architecture has higher accuracy compared to an LSM architecture but the LSM architecture will have significantly less training time as number of training parameters are less.

	LSM	CNN
Train Accuracy	100	99
Test Accuracy	95	98
No of parameter	1250	18500
Epoch	1	70
Kfolds	5	1
Layers	Liquid + output layer	3 convolution layer + hidden + output layer
Preprocessing	Lyon ears	MFCCs

Table 5.1: Comparison of LSM and CNN

Chapter 6

Future Work

We see that the LSM performs very well in the case of the TI dataset but the accuracy drop in the case of the google keyword dataset. This happens as the google keyword dataset has a large no of speakers compared to the TI dataset.

To solve this problem arises because the LSM is not able to generalize much. Two different speakers have different pitch and speed of speaking thus it makes difficult for the LSM to find any similarity. In short, the LSM is not able to generalize between audio classes as due to different speakers the audio classes are little shifted. This problem can be solved with the help of a convolution layer.

Convolution layers have been widely used in image classification. Convolution networks are very good at capturing spatial and temporal information from an image. In convolution networks, we have kernels that are applied to each part of an image. Thus if a feature is present in any part of the image then it will be captured. In the case of the audio signal, we can construct a grid of audio samples that come out of the preprocessing layer and then apply the kernels for each of the timestamps.

6.1 Convolution in LSM

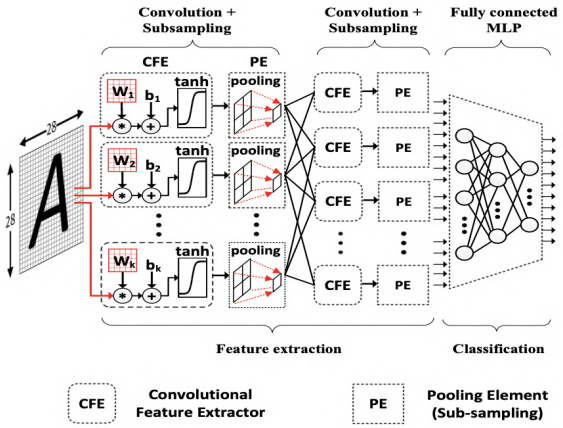


Figure 6.1: CNN in LSM for Mnist(15)

This architecture mainly resembles that of a CNN. Here we have. multiple layers of pooling to reduce the computation cost. They call this architecture as Deep LSM or DLSM architecture. The feed forward layer is similar to a conventional LSM. For backpropagation they have used unsupervised training through STDP to train the network from input to reservoir. And for output final layer they have used spike based supervised learning.

Chapter 7

Predictive Coding

Backpropagation is one of the main features of any artificial network. Backpropagation is one of the key reasons for the success of an artificial neural network. But backpropagation can not directly be applied to spiking neural networks as backpropagation assumes that the cost function is continuous and differentiable. But in Backpropagation due to spikes, the cost function is not defined at the spikes. In backpropagation, the change in weights of a neuron is a function of other neurons that might not be directly linked with the current neuron. For example, if we have a network like input- hidden1- hidden2-output, the weight change in the hidden1 layer will have a term of output layer also, but it is not directly connected.

In predictive coding weight update, the rule is local. Weights are updated based on the error which can be updated in parallel. In predictive coding also do a forward pass by initializing the weights by a random value. After the forward pass, we got the output of every node. We predict the ideal value of a neuron and call it x. We will change the value of x iteratively till we maximize the joint probability. The output layer has x equal to the class label. Maximization of joint probability function can be done in parallel for all the variables and updating of x is local. Suppose we get x^* as the optimal value. We wound use this optimal x to update the value of weights.

Backpropagation proceeds backward sequentially it computes the gradient at each step whereas in predictive coding errors are updated in parallel. Only local information is needed.

7.1 Algorithm

In predictive coding, we associate a random variable X (Gaussian with mean μ and variance 1) with each node. The mean will the scaled value of the output of the previous layer

with weights θ . Mean is this case is analogous to input to a node in a traditional ANNs. We need to maximize the likelihood of the joint distribution of model variables given the input(in the figure nodes at the highest level corresponding to input nodes). So we can write the cost function as the log-likelihood which would be a function of x_i . We keep updating the value of x till log-likelihood converges. Then use this optimum value of x to update the weights. By contrast, when we do backpropagation on ANN while updating the weights of a layer l we assume that the variable in higher layers is fixed.
Let's assume a feed-forward network of L fully connected network. The value of neurons at level l_i depends on level l_{i+1} . In other words, we will give the input to layer L and get the output at level 0. Now predictive coding can be broken into two parts:

- Feed Forward Network
- Weight update step

7.1.1 Feed Forward Network

The feedforward step in predictive coding is similar to the artificial neural network. In this, we introduce our training image to an input layer that is layer L. We multiply the pixel value with weights and sum up all the connections to get the input to a neuron. then we pass it through an activation function to get the output of the neuron. This output will be input to the next layer. The final output will be argmax of all the output neurons.

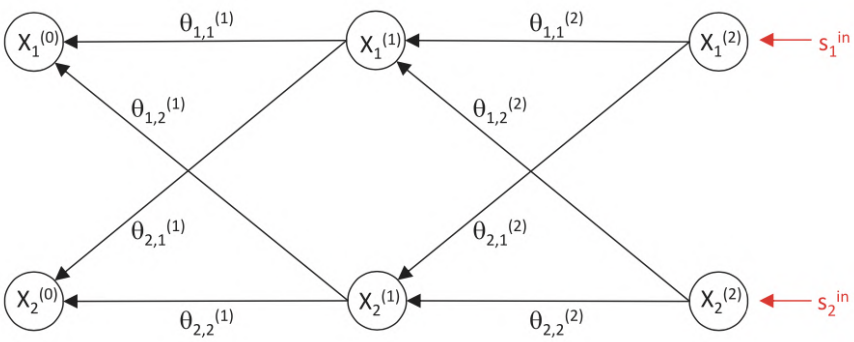


Figure 7.1: Network for feed forward (1)

Here arrow signifies the movement of data. During forward propagation the training vector is supplied to the input layer and then it is propagated to the output layer. For each neuron input is scaled by weights. and then passed through an activation function. here s1 and s2 are the input while X_1^0 and X_2^0 are the output vector while theta is are the weights.

7.1.2 Weight update step

During weight update step we have to approximate the backpropagation. For that we predict the activity of every node and call it X, we then pass X through activation function to get the final output of a neuron. We sample X initially from a Gaussian with mean μ and standard deviation as 1. We take mean as the input to the neuron which is:

$$\mu_i^l = \sum_{j=1}^{n^{(l+l)}} \theta_{i,j}^{l+1} f(x_j^{l+1})$$

Here F is the activation function. Mean μ can be inferred as the input to one neuron while our prediction can be inferred as activity of neuron before applying the activation function. We say that $\bar{X}^l$ as a vector containing all the value of x of layer L.
Our goal is to determine most likely value of model parameter which will determine the activity of nodes in predictive network. We need to maximize the probability that X = x given that the output layer is equal to class label vector. We need to maximize:

$$P(\bar{X}^0 = \bar{x}^0, \bar{X}^1 = \bar{x}^1 .... \bar{x}^{L-1} | \bar{X}^L = \bar{x}^L)$$

We can assume that all the X^l are independent of each other. We can take convert the expression on log likelihood and maximize that expression.

$$CostFunction = \sum_{l=0}^{L-1} ln(P(\bar{X}^l = \bar{x}^l | \bar{X}^{l+1} = \bar{x}^{l+1}))$$

As we know that X is Gaussian(with mean μ and standard deviation 1) distributed we can put the value of x and obtain the cost function in terms of μ which intern depends on xi. So we can obtain the cost function in terms of xi. Now we need to maximize this log-likelihood(F), so we can use gradient descent to maximize the function iteratively.

$$F = \frac{-1}{2} * \sum_{l=0}^{L-1} \sum_{i=1}^{n^l} (x_i^l - \mu_i^l)^2$$
$$\frac{\delta F}{\delta x_b^a} = -(x_b^a - \mu_b^a) + \sum_{i=1}^{n^{a-1}} (x_i^{a-1} - \mu_i^{a-1}) * \theta_{i,b}^a * \bar{f}(x_i^a)$$

Here F is the log likelihood function while f is the activation function. x_b^a represent prediction of bth neuron of layer a. When we differentiate F with respect to x_b^a first term comes directly and second term comes from μ of the lower layer.
We can define a error term as

$$e_i^l = x_i^l - \mu_i^l$$

This error term is the difference between the predicted value at node i of lth layer and the input to a neuron. The gradient of loglikehood function becomes.

$$\frac{\delta F}{\delta x_b^a} = -(e_b^a) + \sum_{i=1}^{n^{a-1}} (e_i^{a-1}) * \theta_{i,b}^a * \bar{f}(x_i^a)$$

We can interpret the above equation as excitation from variable node x_i^l and inhibition from higher level node.Now we can use above equation to change all the x such that the log likelihood function maximizes.

We add extra error nodes to the network. We update all our prediction just based on these errors and we can see that the error can be computed locally hence the change in x can be computed locally. As it is local we can update do this step in parallel.

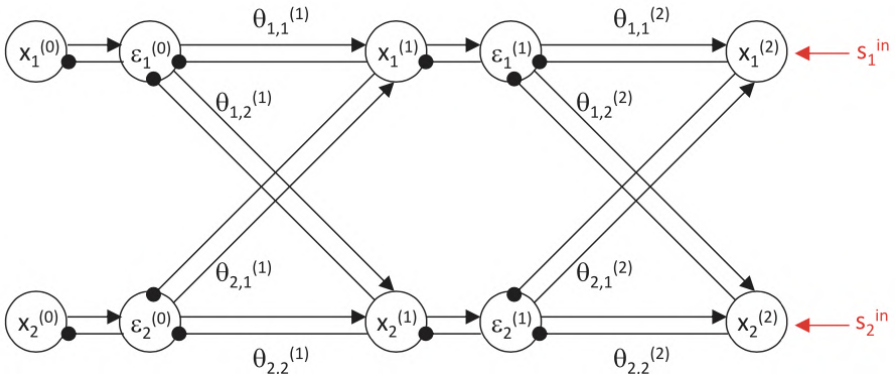


Figure 7.2: Network with error nodes (1)

We keep on updating all our predictions based on the following equation. We move in the direction of gradient as we need to maximize the function.

$$x_b^a = x_b^a + \alpha * \frac{\delta F}{\delta x_b^a}$$

Once the log likelihood function converges the function reached the steady state. Let say that optimal value of x is x*. Now we will use this value to update the weights or $\theta_{i,j}$.

To update the weight we need the gradient of log likelihood with respect to weights, evaluated on the optimal value of out prediction. This gradient can be calculated by differentiating the cost function.

$$\frac{\delta F}{\delta \theta_{b,c}^a} = e_b^{*(a-1)} f(x_c^{*a})$$

Here $\theta_{b,c}^a$ is the weight which connect c^{th} neuron of layer a to b_{th} neuron of layer a-1. We can use this gradient to iteratively modify the value of the weight given by equation.

$$\theta_{b,c}^a = \theta_{b,c}^a + \beta * \frac{\delta F}{\delta \theta_{b,c}^a}$$

We do this step for every batch in every epoch.
So to summarize Predictive coding:

- For each batch do a Forward pass. Get the output of every node.
- Predict x for every node of every layer(based on the input of a node)
- Compute the error of output node.($e_i^l = x_i^l - \mu_i^l$)
- Modify your prediction based on local error till the log likelihood function converges.($\frac{\delta F}{\delta x_b^a} = -(e_b^a) + \sum_{i=1}^{n^{a-1}} (e_i^{a-1}) * \theta_{i,b}^a * \bar{f}(x_i^a)$, $x_b^a = x_b^a + \alpha * \frac{\delta F}{\delta x_b^a}$)
- Update the weights based on these updated predictions. ($\frac{\delta F}{\delta \theta_{b,c}^a} = e_b^{*(a-1)} f(x_c^{*a})$, $\theta_{b,c}^a = \theta_{b,c}^a + \beta * \frac{\delta F}{\delta \theta_{b,c}^a}$)
- Repeat this for each epoch

7.2 Backpropagation

Backpropagation is a method to minimize the cost function by changing the weights of each connection in a neural network. The amount by which each weight change is proportional to the gradient of the cost function with respect to that weight. The proportionality constant is called the learning rate.
In backpropagation, we first initialize weights of all the layers. Then we do a forward propagation. In forward propagation, we give the input layer the pixel values of our image. Here we have 28*28 size image therefore the size of the input will be 784. In each layer receives the output of the previous layer and scales it by a weight, after this scaling and summing of all the inputs, it is passed through a nonlinear transformation. We usually choose Relu when we operate on neural networks as it does not give the problem of vanishing gradient.
In backpropagation, we update the weights layer by layer starting with the output layer. We use the chain rule to find the gradient for the weights of hidden layers. We first calculate the error of the output layer:

$$E = \frac{1}{2} * (s^{out} - y^{out})^2$$

Here E is the error s^{out} is the class label vector and y^{out} is the output. This error can be used to calculate the change in weights of output layer.

$$\Delta w_{b,c}^l = \alpha * \frac{\delta E}{\delta w_{b,c}^l}$$
$$\Delta w_{b,c}^l = \alpha * \frac{\delta E}{\delta y_b^{l-1}} * \frac{\delta y_b^{l-1}}{\delta w_{b,c}^l}$$

Here y_b^{l-1} is the output of the node of l-1 layer.The change of weight connectinga^{th} node of layer l to b^{th} node of layer l-1. Note: l-1 is the outer layer compared to layer l will be proportional to the gradient of the error with respect to $w_{b,c}^l$. Finding this gradient can be tricky as we do not know the function of E with $w_{b,c}^l$. So we can use the chain rule to simplify.

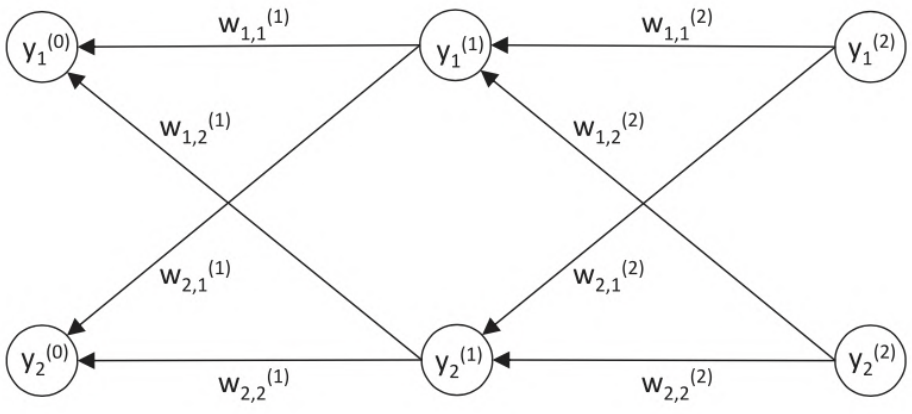


Figure 7.3: A small neural network (1)

7.2.1 Gradient Descent

Gradient descent is an algorithm commonly used to maximize or minimize a function. Suppose we are given a function f with some independent variable x. Now we need to find an optimal x that minimizes f, at the point of minima the derivative will go from negative to positive. We choose a point x1 and modify x1 such that it minimizes f. We will move in the opposite direction of the gradient of F. Training of neural networks is normally done using Gradient Descent (GD). In GD, the network parameters (say, θ are moved in the direction of the negative gradient of some loss function L. This is illustrated in the following equation :

$$\theta = \theta - \eta \frac{\partial L}{\partial \theta}$$

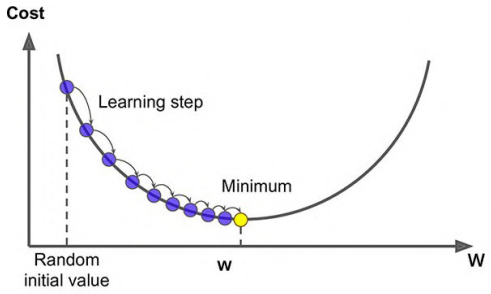


Figure 7.4: Gradient descent (9)

7.3 Comparison with Backpropagation

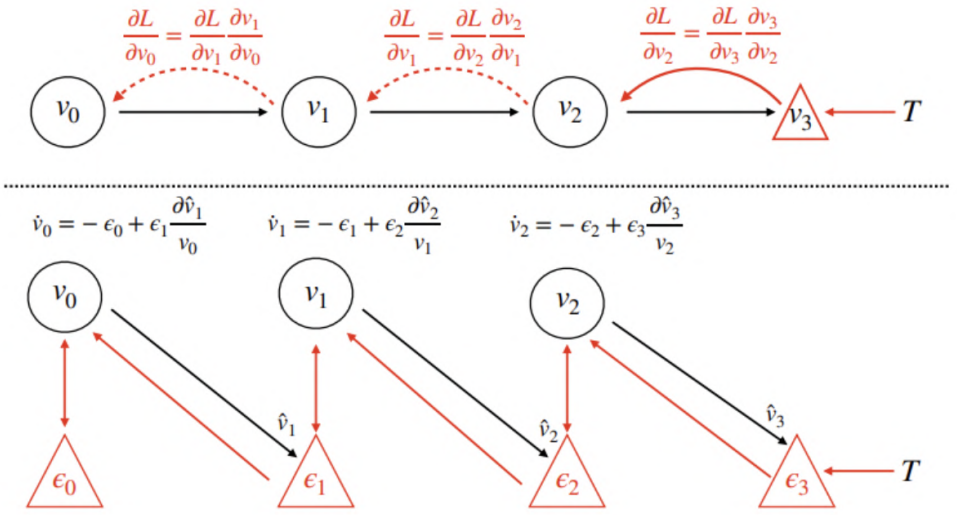


Figure 1: Top: Backpropagation on a chain. Backprop proceeds backwawrds sequentially and explicitly computes the gradient at each step on the chain. Bottom: Predictive coding on a chain. Predictions, and prediction errors are updated in parallel using only local information.

Figure 7.5: Weight update in Backpropagation and Predictive coding (2)

In backpropagation weights are updated in a chain. We can see that when we update the weight v_0 then the term of v_3 is present as we are using chain rule to update the weight. In predictive coding the weight v_0 is updated only using the error term of current layer and the error term of higher layer.

7.4 Advantages

Predictive coding does local computation. Unlike backpropagation, the weight update rule of a neuron is a function of the neurons connected to it only. This becomes handy

as in biological neurons the change in weight of synapse should be only dependent on connecting neurons, not the output neurons. The amount of synaptic weight modification is dependent on only the activity of the two neurons the synapse connects. It is consistent with connectivity in the neocortex.
The weight update rule is independent of neuron weights of further layers. Due to this property, we can update the weights in parallel. So for each batch, we could speed up the maximization of likelihood step if we have enough parallel computing.
In the paper, they also have tested predictive coding on Recursive Neural network(RNN), Long Short-Term Memory (LSTM), and convolution neural networks. In all the three networks predictive coding performs very well and results are similar to when ANN is trained through backpropagation.

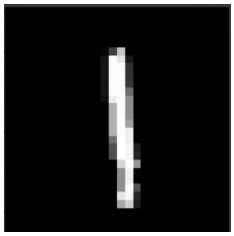
7.5 Disadvantages

It requires more computation than a normal backpropagation. As in predictive coding, we are doing a maximization step(through gradient decent) on each batch therefore it requires more computation. Suppose we need 10 iterations for finding the x that maximizes our log-likelihood function, so we would need approx 10 * x more computation than backpropagation. In backpropagation we could update the weights directly we do not have any maximization step for each batch.

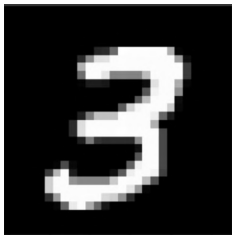
7.6 Previous Work

The Predictive coding was implemented on an artificial neural network before by Shashwat in his dual degree project. It achieved a very similar result when we compared it to a normal artificial neural network with backpropagation. It was trained on Mnist data set which used 50000 It used a 784-n-n-10 neural network architecture. N was varied from 400 - 700 and the best results were obtained on 600. It is a fully connected network. The best test accuracy which was achieved was 97 percent. Relu Activation function is used

To implement predictive coding on an Artificial Neural we first initialize all the weights to a random value. We then pass the first training batch and compute the output of every node. Then we initialize all our predictions ie all the x to 0. We clamp the prediction of output as the target(class label). Then for each training batch, we keep on updating x till the joint log-likelihood maximizes. As it is log it will converge to 0. We see that the x converges very fast about 10 iterations. But we have to do this step for every



(a) Class: One



(b) Class: Three

Figure 7.6: Few samples of Mnist Dataset

batch, for every batch, we have to make the log-likelihood to come to zero. We then use this updated value of X to find the new weights. We do this for some epoch(10 in this case). At last of every epoch we do testing on a fresh dataset, Test dataset.

7.6.1 Dataset

To implement predictive coding Mnist(Modified National Institute of Standards and Technology dataset.) data set is used. In Mnist data consists of 60000 black and white images of numbers from 0 - 9. It is a labeled dataset. Each image is of 28*28 pixels intensity of each pixel range from 0 - 255. For training 50000 images were used and 10000 were used for testing.

7.6.2 Parameters

Parameter	Final value
Learning Rate	0.01
epochs	10
Batch size	32
No. Predictive Cycles	10

Table 7.1: Final Parameter

7.6.3 Results

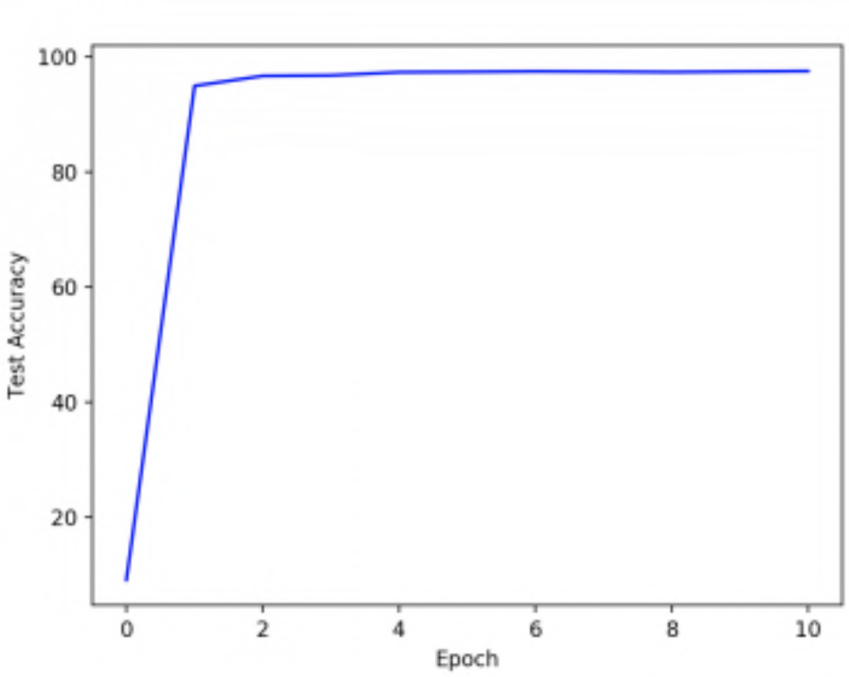


Figure 7.7: Test Accuracy of Predictive coding on ANN (12)

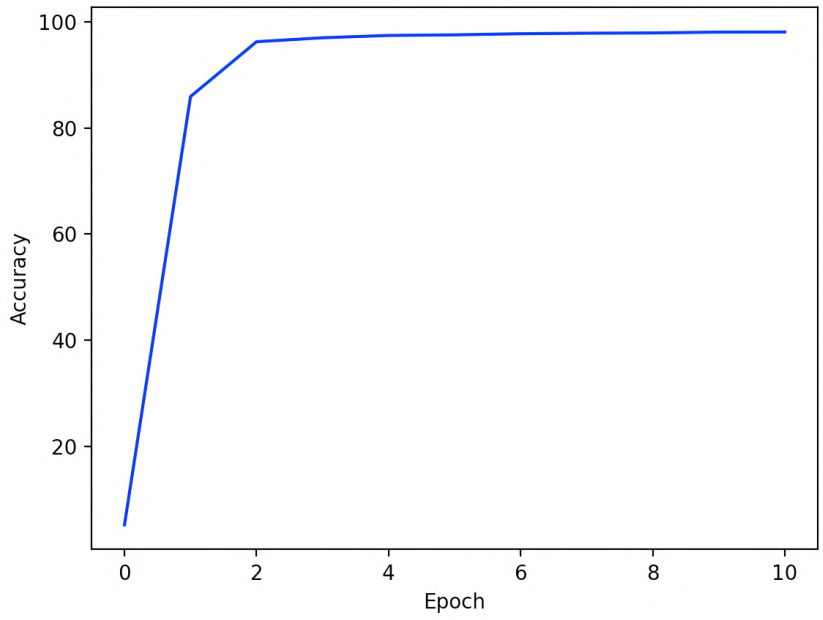


Figure 7.8: Test Accuracy of Backpropagation on ANN(12)

We can see that predictive coding performs similar to backpropagation when trained on the same network. Both the algorithms have similar test accuracy in Mnist dataset.

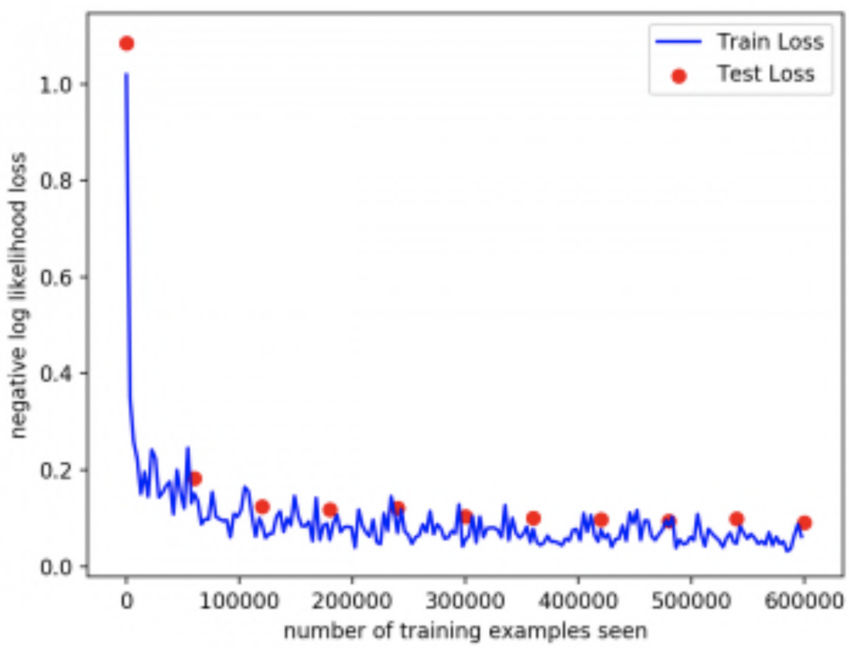


Figure 7.9: Negative log Likelihood of Predictive coding(12)

7.7 Future Work

We see that predictive coding produces a similar result when we compare it to an artificial neural network with backtracking. Through this, we can sufficiently say that predictive coding can approximate backpropagation. And as it is local that is the weight update rule for each node is dependent only on the connected nodes it can be used on a spiking neural network. Here I propose a possible way to implement predictive coding on SNN.

7.7.1 Predictive Coding On SNN

I would use Mnist Dataset to test. We have to convert every image into spikes. The frequency of these spikes will be higher for higher intensity neurons and lower for lower intensity neurons. Each neuron will be modeled as a LIF neuron.
At first, we will initialize the weights with a random value sampled from a Gaussian. We can then do a forward pass for a stimulated time t. Each LIF neuron receives the synapse from the connected neurons and scales it according to the synapse weight. At the output, the prediction will be the neuron that spikes the most.

During the weight update step, we first calculate the error of the output layer. For weight updating, we can set the output of every neuron as the frequency of its spiking. The error of the output can be just the mean squared error of frequency of the output layer.
The μ of the predictive coding of a neuron b of level l can be replaced with the weighted sum of the output of all the frequencies of the previous layer. We can predict the activity of neuron X. Error of every other node can be just the difference between μ and x. We can now use this error and μ to continuously update our prediction till the log-likelihood function converges to a maximum by using the following equation derived earlier.

$$\frac{\delta F}{\delta x_b^a} = -(e_b^a) + \sum_{i=1}^{n^{a-1}} (e_i^{a-1}) * \theta_{i,b}^a * \bar{f}(x_i^a)$$
$$x_b^a = x_b^a + \alpha * \frac{\delta F}{\delta x_b^a}$$

After we get a optimal prediction from above we will use this x to update the the weights by following equation.

$$\frac{\delta F}{\delta \theta_{b,c}^a} = e_b^{*(a-1)} f(x_c^{*a})$$
$$\theta_{b,c}^a = \theta_{b,c}^a + \beta * \frac{\delta F}{\delta \theta_{b,c}^a}$$

After weight updation we will again bring next training batch and repeat the above steps. this process is repeated for every epoch.

References

[1] Whittington, J., Bogacz, R. (2016). An approximation of the error back-propagation algorithm in a predictive coding network with local Hebbian synaptic plasticity. bioRxiv. Neural Computation 29, 1229–1262 (2017)

[2] Beren Millidge, Alexander Tschantz, Christopher L. Buckley. (2020). Predictive Coding Approximates Backprop along Arbitrary Computation Graphs. arXiv, cs.LG, 2006.04182, 2020.

[3] Jin, Y., Li, P. (2017)
Performance and robustness of bio-inspired digital state machines: A case study of speech recognition. Neorocomputing. 226 145-160.

[4] Ajinkya Gorad, Vivek Saraswat, Udayan Ganguly. (2019). Predicting Performance using Approximate State Space Model for Liquid State Machines. arXiv, 1901.06240

[5] Wijesinghe, P., Srinivasan, G., Panda, P., Roy, K. (2019). Analysis of Liquid Ensembles for Enhancing the Performance and Accuracy of Liquid State Machines. Frontiers in Neuroscience, 13, 504.

[6] Vivek Saraswat, Ajinkya Gorad, Anand Naik, Aakash Patil, Udayan Ganguly. (2021). Hardware-Friendly Synaptic Orders and Timescales in Liquid State Machines for Speech Classification., arXiv, 2104.14264

[7] B. Schrauwen and J. Van Campenhout, "BSA, a fast and accurate spike train encoding scheme," Proceedings of the International Joint Conference on Neural Networks, 2003., 2003, pp. 2825-2830 vol.4, doi: 10.1109/IJCNN.2003.1224019.

[8] R. Lyon, "A computational model of filtering, detection, and compression in the cochlea," ICASSP '82. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1982, pp. 1282-1285, doi: 10.1109/ICASSP.1982.1171644.

[9] What is gradient decent in machine learning
https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/

[10] Integrate and fire models
`https://neuronaldynamics.epfl.ch/online/Ch1.S3.html`.

[11] Action Potential
`https://en.wikipedia.org/wiki/Action`$_p$*otential*.

[12] Shashwat Shukla
Spiking Neural Networks for Inference, Learning and Navigation. Dual degree phase 2 report, 2020

[13] Rohan Pathak
C. elegans inspired Contour Tracking using Spiking Neural Networks. Dual degree phase 2 report, 2020

[14] LSM
https://link.springer.com/chapter/10.1007/978-981-15-8135-9$_2$0.

[15] Q. Wang and P. Li, "D-LSM: Deep Liquid State Machine with unsupervised recurrent reservoir tuning," 2016 23rd International Conference on Pattern Recognition (ICPR), 2016, pp. 2652-2657, doi: 10.1109/ICPR.2016.7900035.