sRBM simulation

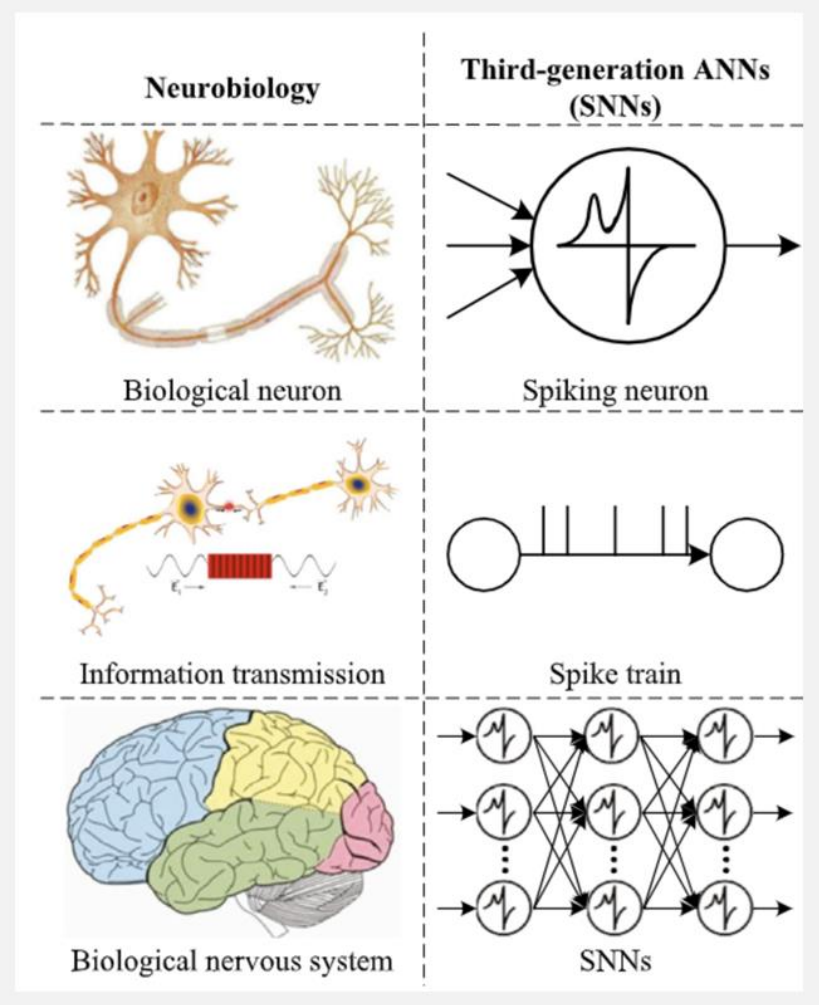
NMDL 인턴십 프로그램

최겸, 임혁준, 조윤

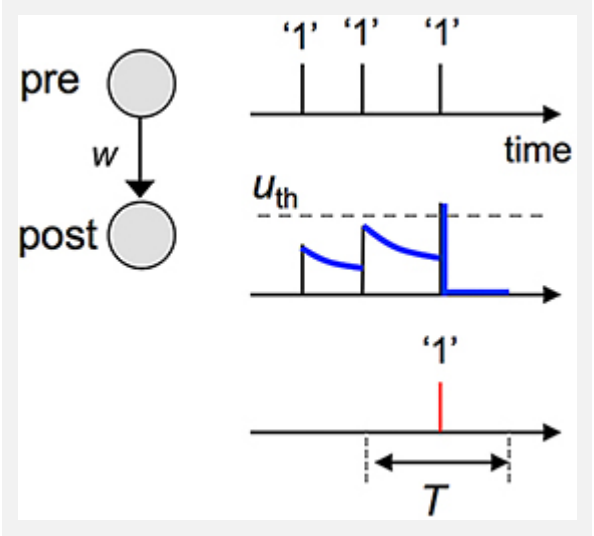
Contents

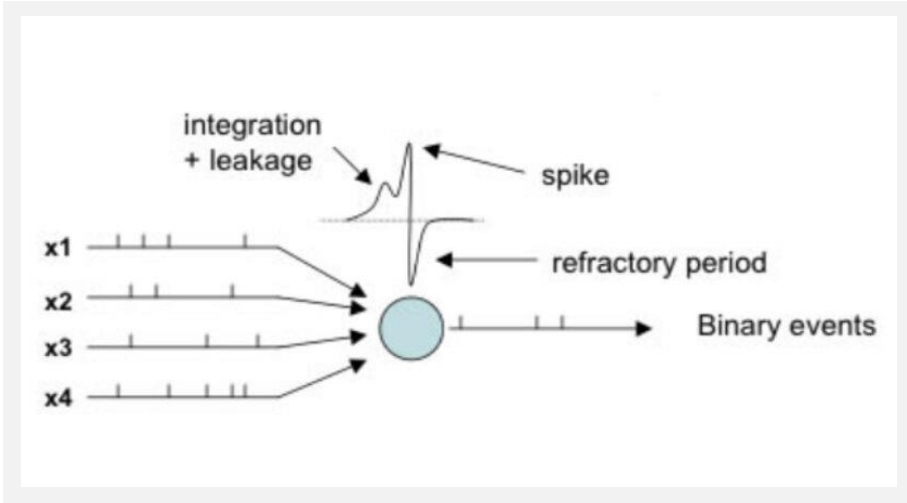
1. SNN

- **About SNN**



Xiangwen Wang et al., Neural Networks, 2020





Why SNN?

-Energy Efficiency

-Lighter than DNN

-Bio-Plausible

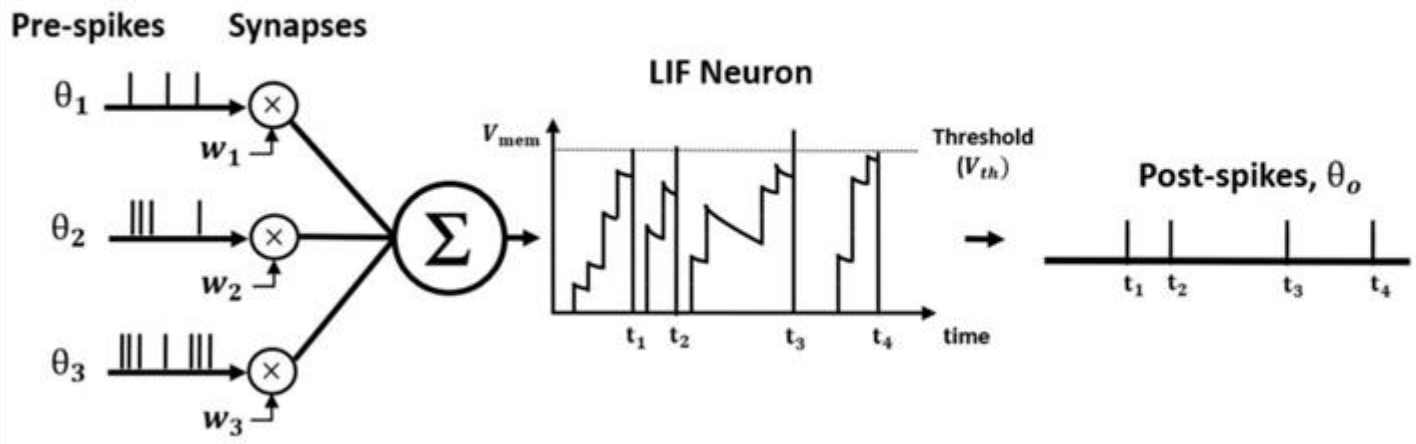
How SNN works?

-LIF

-STDP

1. SNN

- **LIF(Leaky Integrate and Fire)**

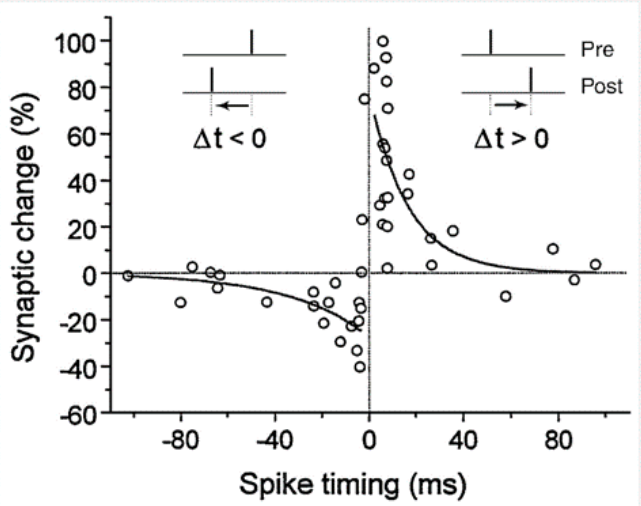


$$\tau_m \frac{dV_{mem}}{dt} = -V_{mem} + I(t)$$

$$I(t) = \sum_{i=1}^{n^l} (w_i \sum_k \theta_i(t - t_k))$$

$$\theta(t - t_k) = \begin{cases} 1, & \text{if } t = t_k \\ 0, & \text{otherwise} \end{cases}$$

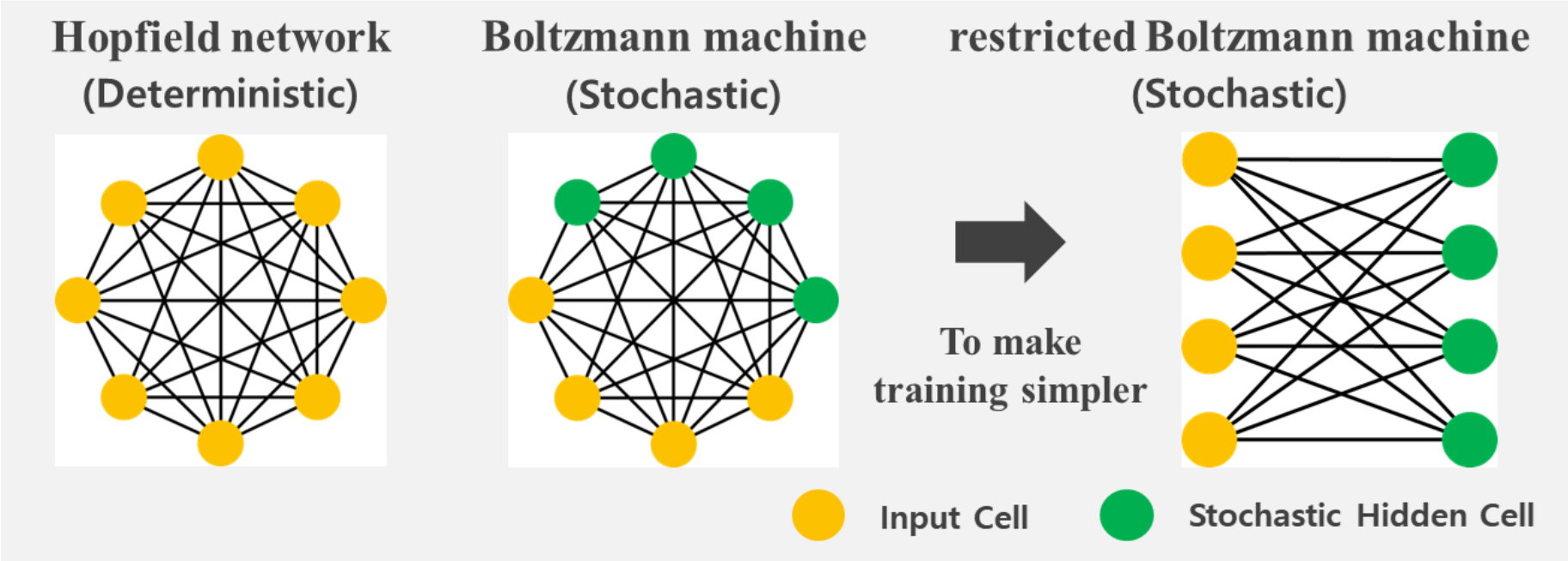
- **STDP**

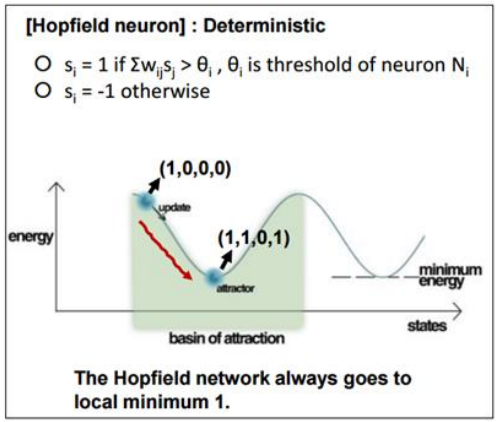


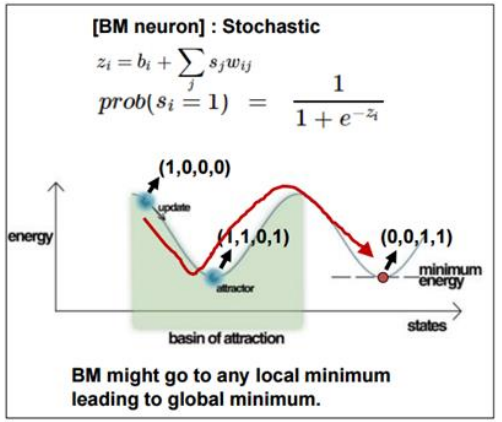
- **Information is transmitted in the form of spikes**
- **Network only works when there are spikes**
  → **"event-driven"**
- **It is easy to simulate the behavior of real neurons**

2. RBM

- **Why RBM?**





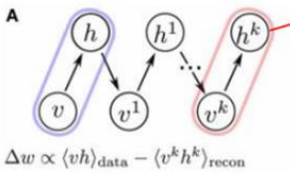


- Boltzmann machine (BM) neuron has **stochastic property**
  → prevent network from staying at local minimum
- All of neuron's state have **non-zero probability**
- **RBM structure** : to make training simpler than BM

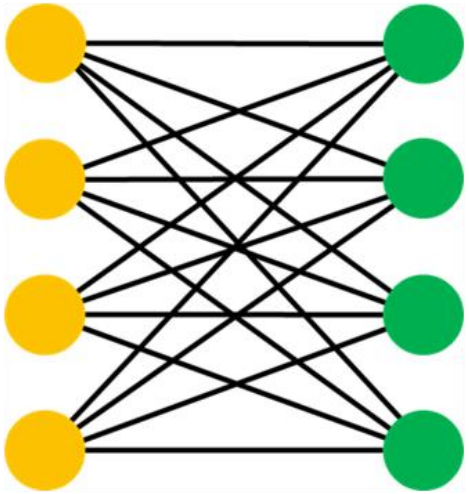
Hinton: Contrastive Divergence (CD)

$$\left\langle \frac{\partial \log P(\mathbf{v})}{\partial w_{ij}} \right\rangle_{\text{data}} = \langle s_i s_j \rangle_{\text{data}} - \langle s_i s_j \rangle$$

reconstruction

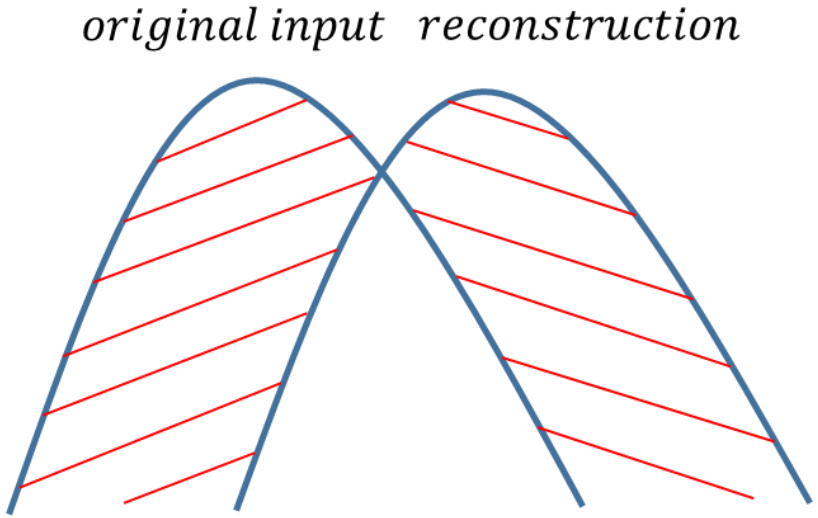


2. RBM



construction

reconstruction



Kullback-Leibler Divergence

Optimizing RBM = Minimizing divergence

$$E(v,h) = -b'v - c'h - h'Wv$$

$$F(v) = -\log \sum_h \exp(-E(v,h))$$

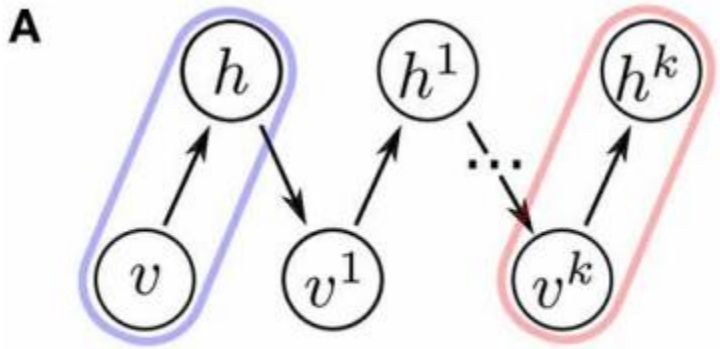
$$p(v) = \frac{\exp(-F(v))}{Z'}$$

$$D(p||q) = \sum_i p_i \log \frac{p_i}{q_i}.$$

[1] E.Neftci et al., Front. Neurosci., 2014

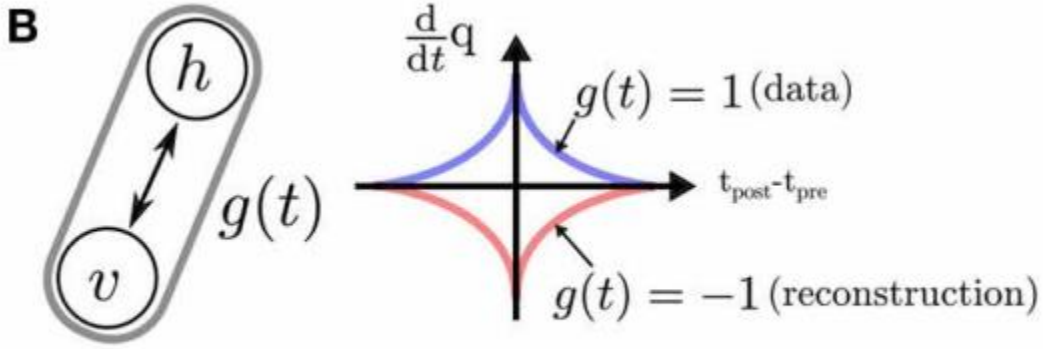
3. SNN-RBM simulation

Contrastive Divergence



$$\Delta w \propto \langle vh \rangle_{\text{data}} - \langle v^k h^k \rangle_{\text{recon}}$$

event-driven Contrastive Divergence

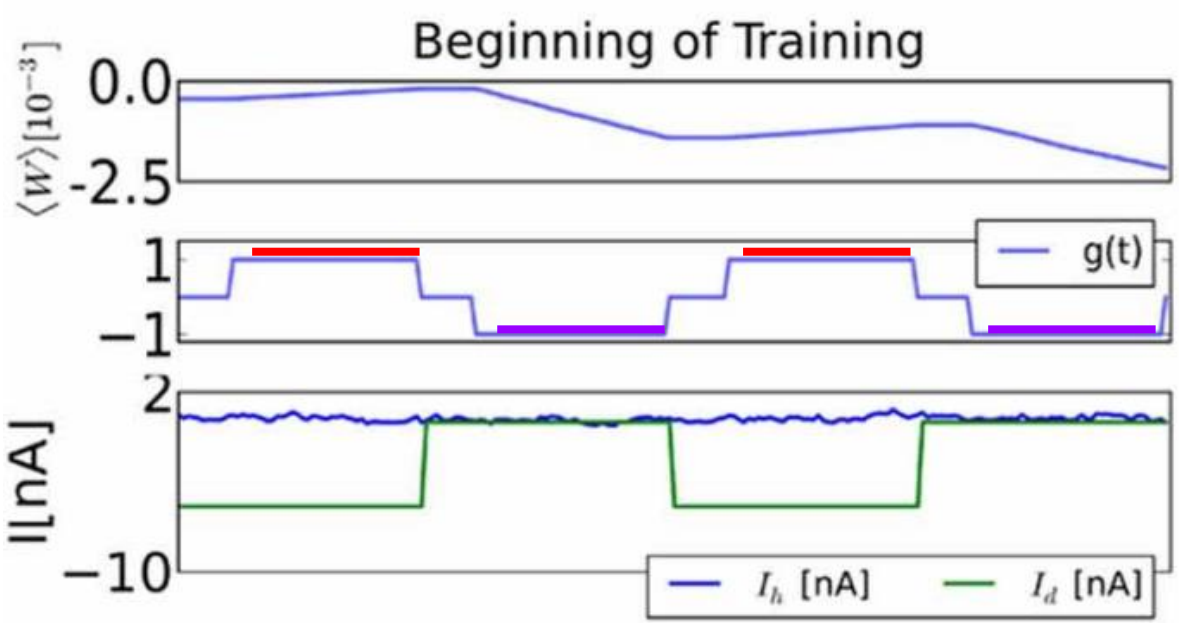


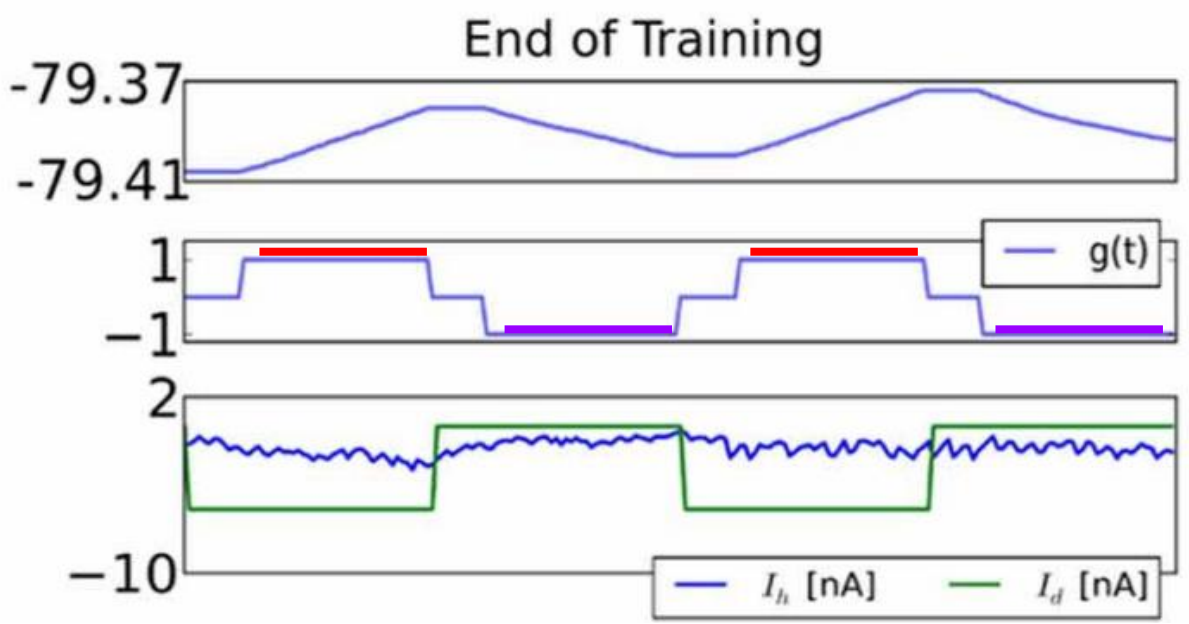
$$\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{q} = g(t)\mathrm{STDP}(v(t), h(t))$$

$$g(t) = \begin{cases} 1 & \text{if } \mathrm{mod}(t, 2T) \in (\tau_{\mathrm{br}}, T) \\ -1 & \text{if } \mathrm{mod}(t, 2T) \in (T + \tau_{\mathrm{br}}, 2T) \\ 0 & \text{otherwise} \end{cases}$$

[1] E.Neftci et al., Front. Neurosci., 2014

3. SNN-RBM simulation (cont'd)





data phase
(LTP, potentiation)

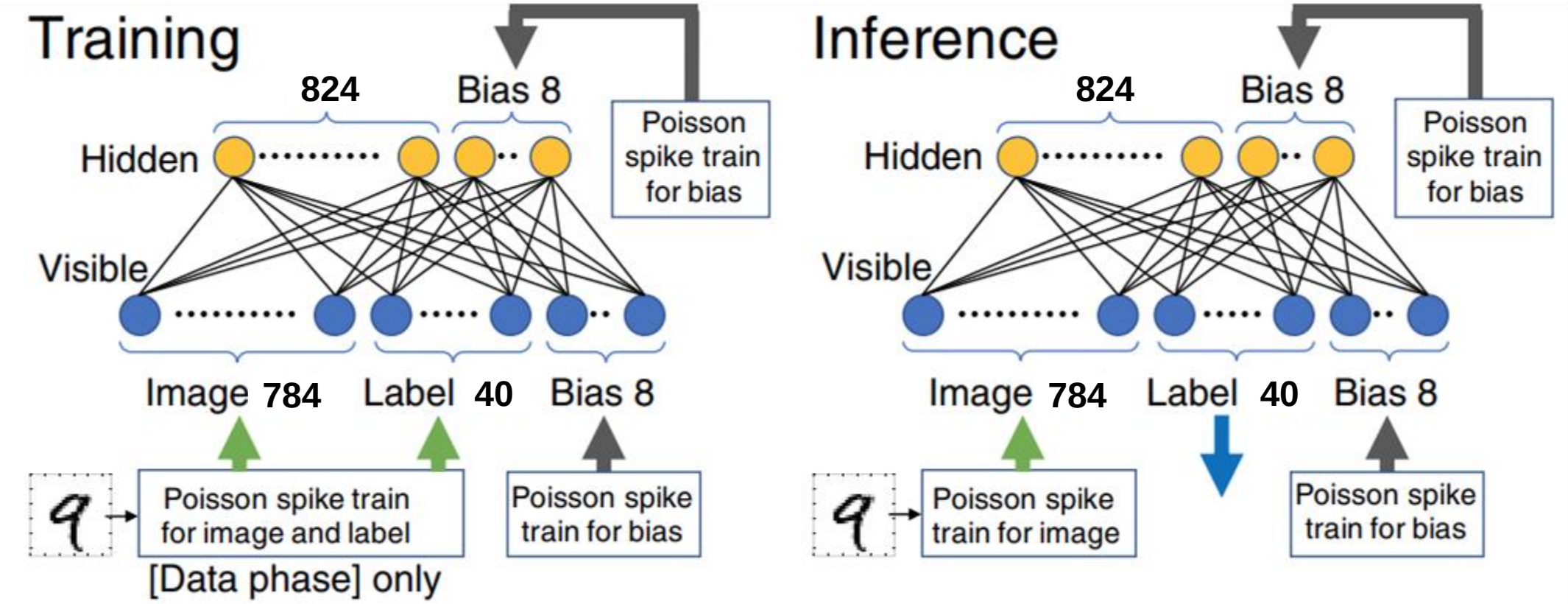
model phase
(LTD, depression)

data phase
(LTP, potentiation)

model phase
(LTD, depression)

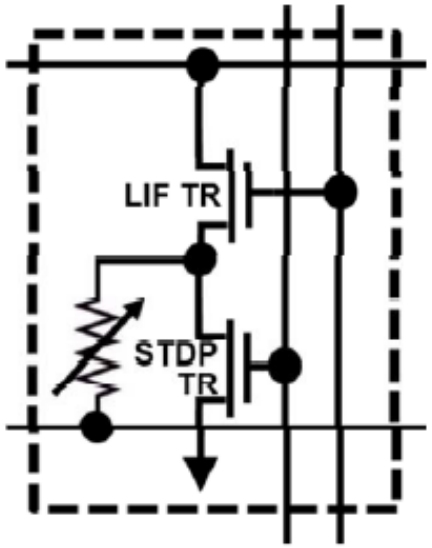
[1] E.Neftci et al., Front. Neurosci., 2014

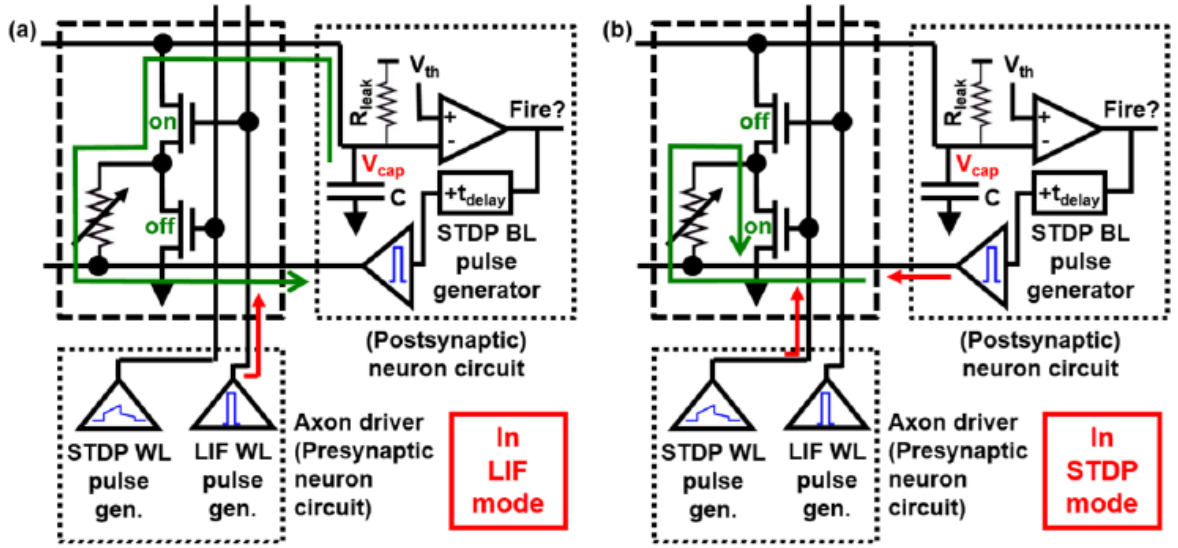
3. SNN-RBM simulation (cont'd)



[2] Shin et al., (2022). Pattern Training, Inference, and Regeneration Demonstration Using On-Chip Trainable Neuromorphic Chips for Spiking Restricted Boltzmann Machine. *Advanced Intelligent Systems, 4*.

How to implement sRBM into ‘real’ hardware? -> 2T1R structure



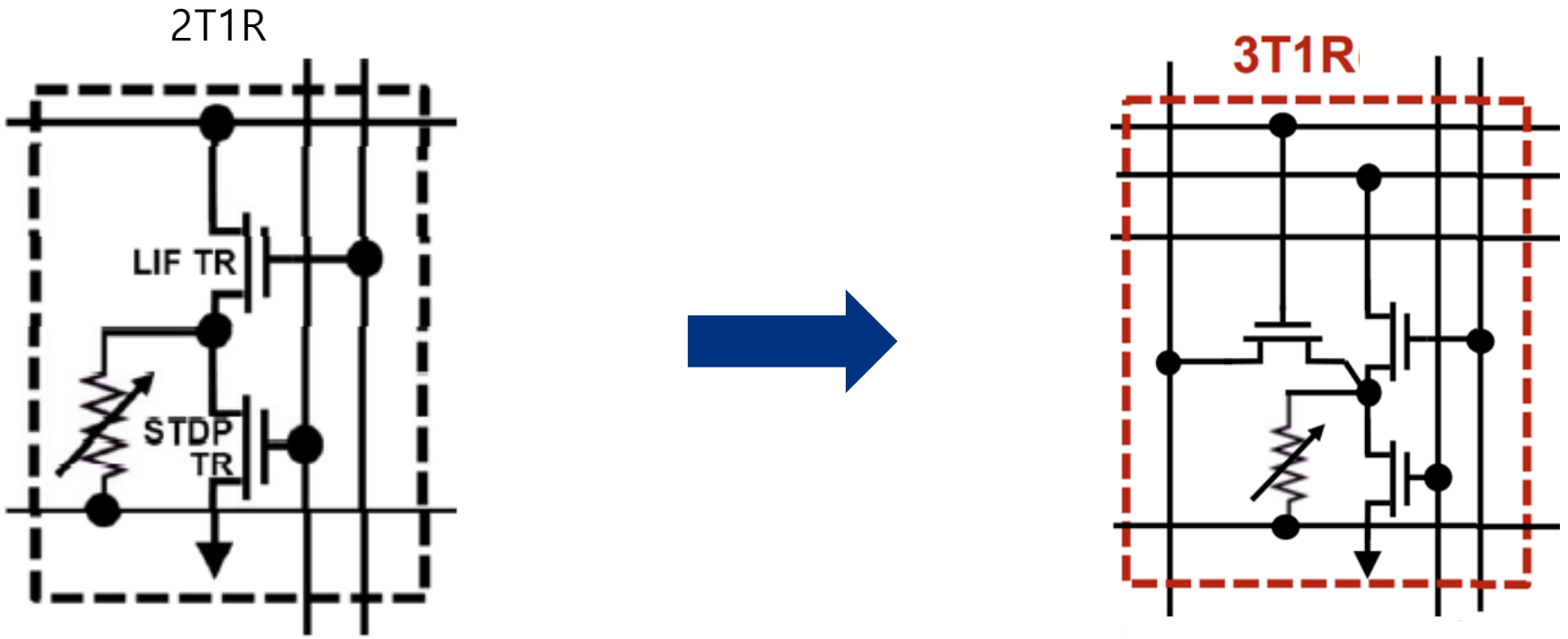


S. Kim *et al*., "NVM neuromorphic core with 64k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous in-situ learning," *2015 IEEE International Electron Devices Meeting (IEDM)*, Washington, DC, USA, 2015, pp. 17.1.1-17.1.4

Uses phase change memory (PCM) as synapse (weight storage)

Uses 2 transistors to operate in both LIF and STDP mode in same structure!

Limitations of 2T1R

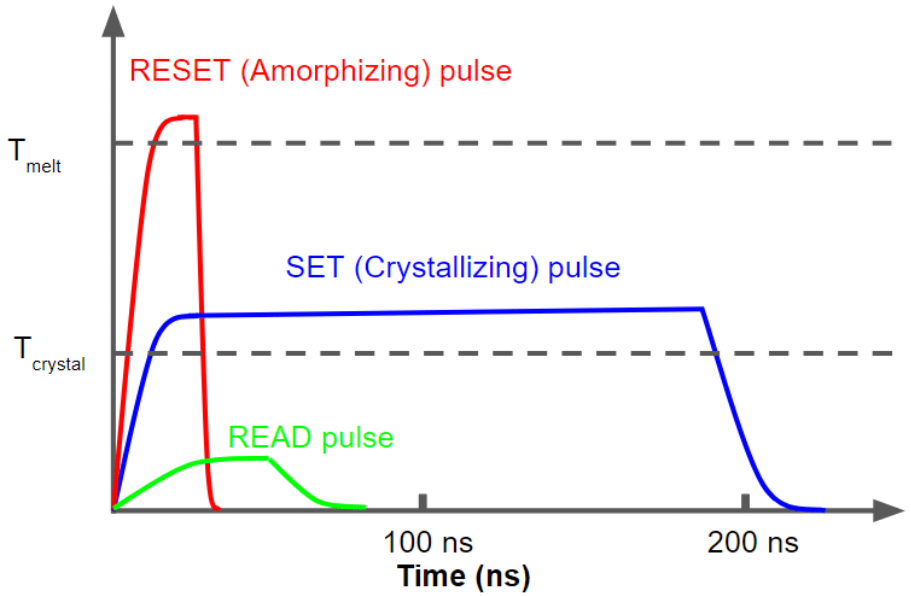


Cannot operate backward LIF (BLIF) ➡ Can operate in LIF, STDP, and BLIF modes in same structure!

S. Kim *et al*., "NVM neuromorphic core with 64k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous in-situ learning," *2015 IEEE International Electron Devices Meeting (IEDM)*, Washington, DC, USA, 2015, pp. 17.1.1-17.1.4

M. Ishii *et al*., "On-Chip Trainable 1.4M 6T2R PCM Synaptic Array with 1.6K Stochastic LIF Neurons for Spiking RBM," *2019 IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA, USA, 2019, pp. 14.2.1-14.2.4

Limitations of PCM (symmetry)



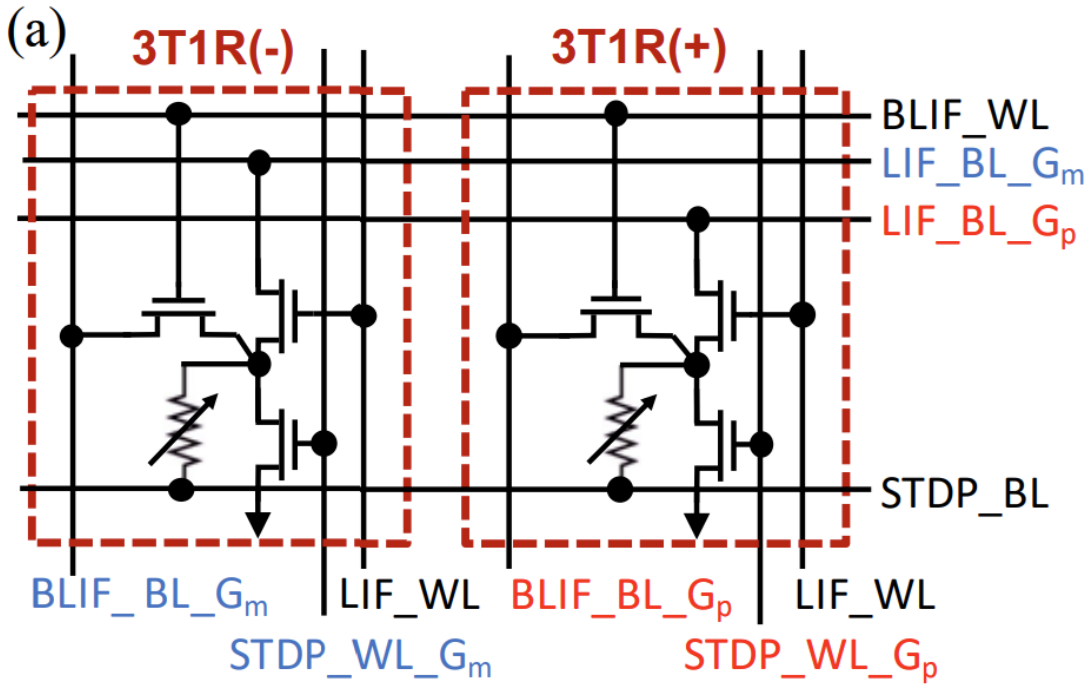
A Review of Germanium-Antimony-Telluride Phase Change Materials for Non-Volatile Memories and Optical Modulators - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/The-SET-RESET-and-READ-pulses-for-a-PCM-device_fig2_330896754 [accessed 24 Jul, 2023]

PCM changes conductance by Joule heating

SET: increasing conductance
RESET: decreasing conductance

PCM has abrupt conductance change in RESET compared to SET -> asymmetric!!

Solution: 6T2R structure



$$G = G_p - G_m$$

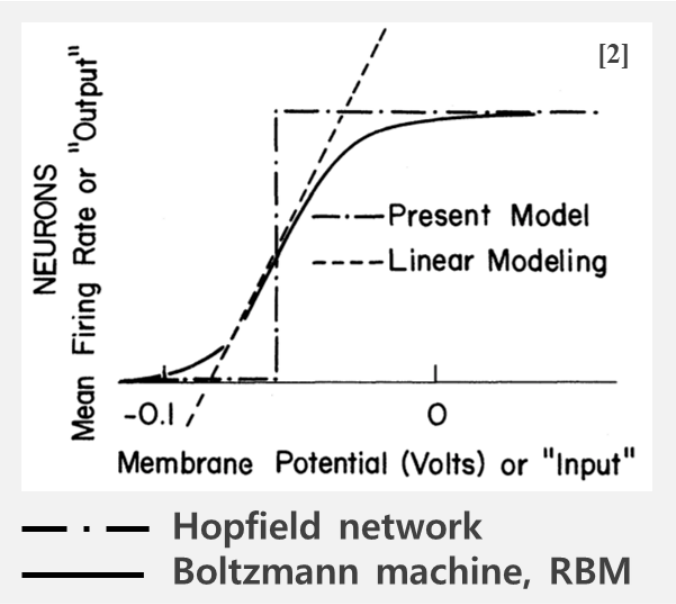
(G: weight, p: plus, m: minus)

Connect two 3T1R structures to compensate for asymmetric property of PCM

Operates in LIF, BLIF, STDP mode asynchronously

M. Ishii *et al*., "On-Chip Trainable 1.4M 6T2R PCM Synaptic Array with 1.6K Stochastic LIF Neurons for Spiking RBM," *2019 IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA, USA, 2019, pp. 14.2.1-14.2.4

Stochastic neurons by using random walk circuits



Neuron's membrane potential

$$C\frac{d}{dt}u_i = -g_L u_i + I_i(t) + \sigma\xi(t)$$

external noise term

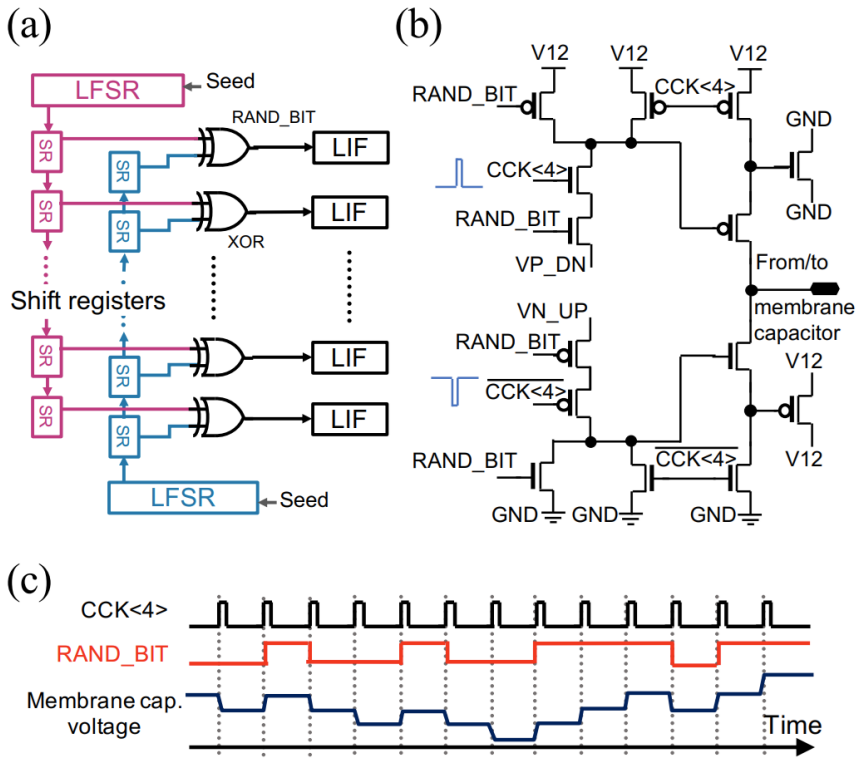


Fig. 7. Random walk function in LIF neuron circuits (a) Random bit generation (b) Random charge transfer circuits (c) Example of random potential movement.

M. Ishii *et al.*, "On-Chip Trainable 1.4M 6T2R PCM Synaptic Array with 1.6K Stochastic LIF Neurons for Spiking RBM," *2019 IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA, USA, 2019, pp. 14.2.1-14.2.4

Weight update (STDP, eCD)

- Visible neuron: LIF_WL, STDP_WL pulse
- Hidden neuron: BLIF_WL, STDP_BL pulse

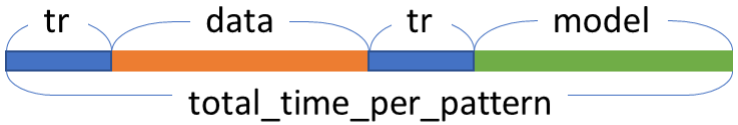
- Main concept of STDP(in this structure): if STDP_WL & STDP_BL pulse intersect -> weight change
  Weight change depends on STDP_WL amplitude !

- The rule for weight change: eCD -> during data phase(construction) : potentiation / during model phase(reconstruction): depression

g(t) : global gating variable

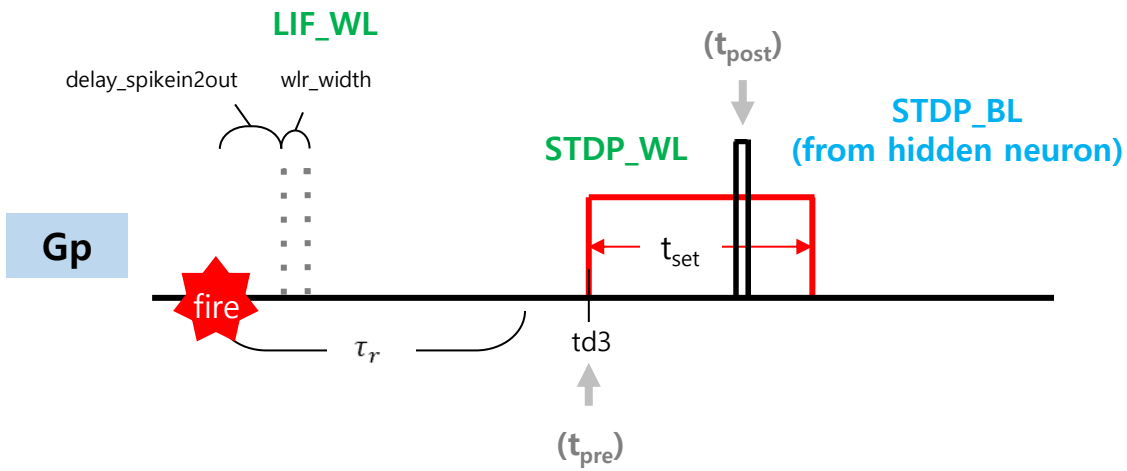
1. In data phase,
   - Visual neurons are clamped with data.
   - g(t)=1 → $\Delta w_{data}=v_{data}h_{data}$
2. In model phase,
   - Visual neurons are not clamped with data.
   - g(t)=-1 → $\Delta w_{model}=v_{model}h_{model}$
3. During the transition period,
   - g(t)=0 → Δw=0

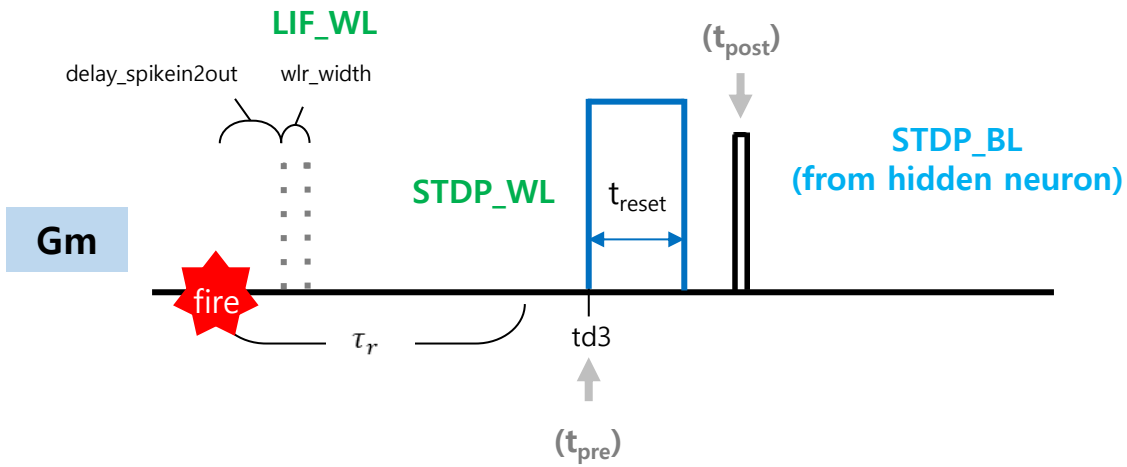
$$\Delta w = \Delta w_{data} + \Delta w_{model}$$
$$= v_{data}h_{data} - v_{model}h_{model}$$



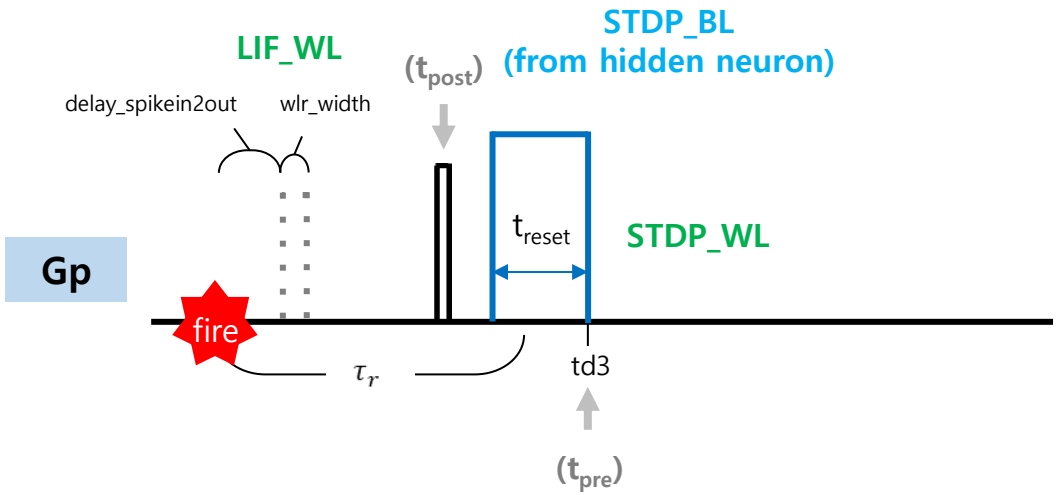
Weight update (STDP, eCD)

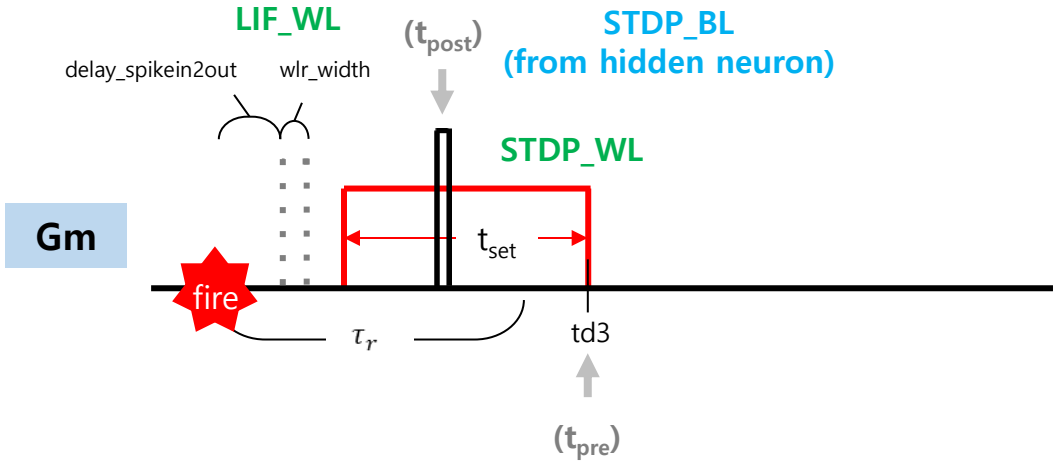
Data phase(Potentiation)





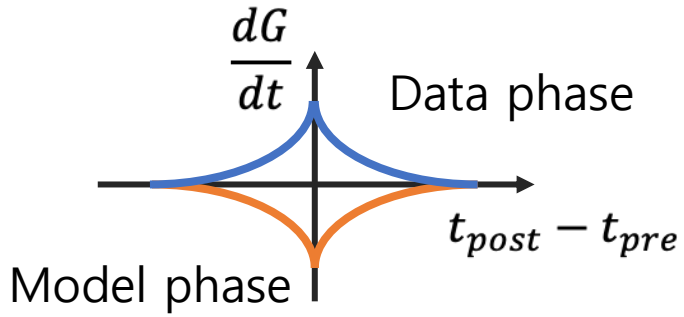
Model phase(Depression)



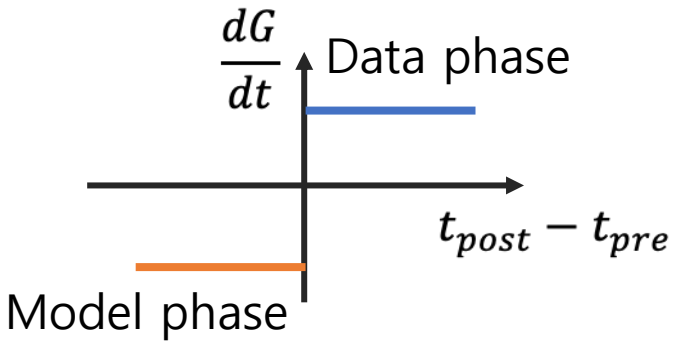


Weight update (STDP, eCD)

STDP suggested in Neftci et al(2014)



STDP implemented into 6T2R structure



Code Explanation

The final goal of simulation

1. generate spike(at constant rate) pixel brightness that exceeds mnist threshold.
2. SNN learns the pattern
3. SNN becomes capable of inferring newly input images.

approximate flow of the code

1. Data preparation, image preprocessing
2. SNN model definition, initialize
3. Model training. Rules mentioned(STDP, Leaky LIF, eCD) are applied.
4. Inference

sRBM MNIST accuracy 향상 연구

Parameter Optimization

NMDL 인턴십 프로그램

최겸

Parameter Optimization

- **mnist_binary_threshold**

1	65.5
2	73.9
3	71.8

mnist threshold = 0.7

1	72.1
2	76.7
3	76.5

mnist threshold = 0.5

1	73.8
2	78.9
3	79.6

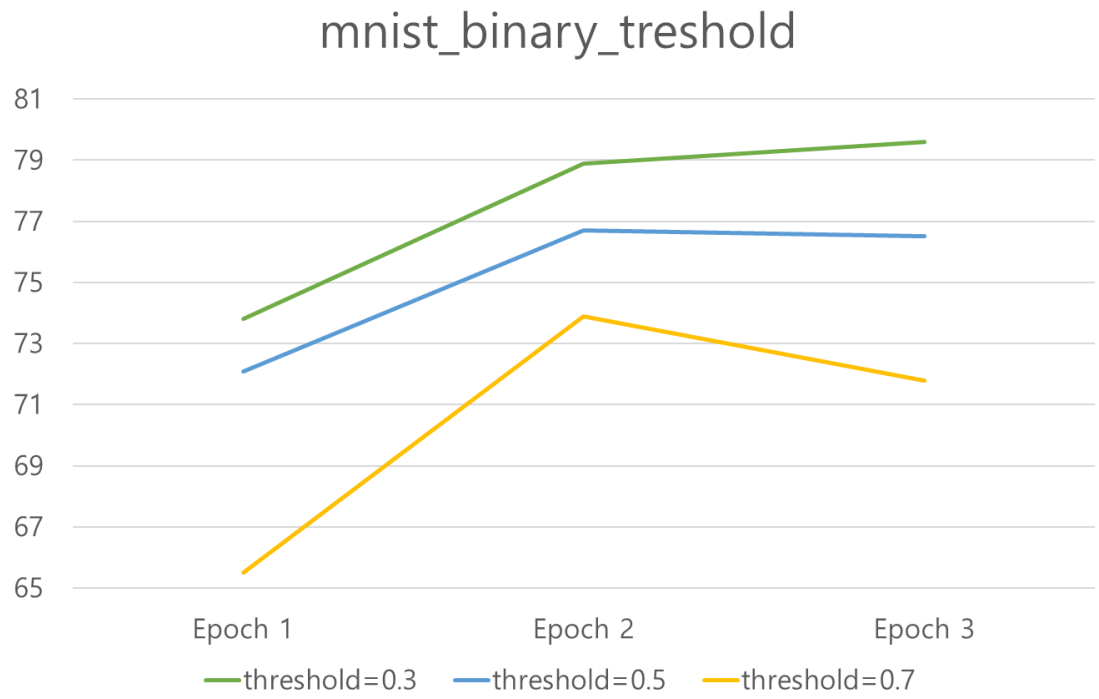
mnist threshold = 0.3

1	74
2	79.5
3	83.6
4	83.6
5	82.8
6	84.3
7	81.2
8	82.6

mnist threshold = 0.2

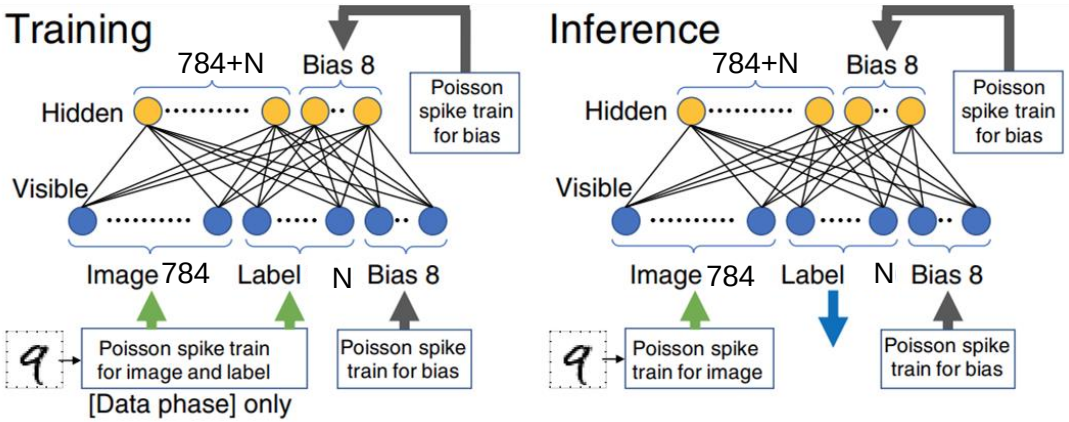
1	74
2	80.1
3	83.6
4	83.6
5	82.8
6	84.3
7	81.2
8	82.6

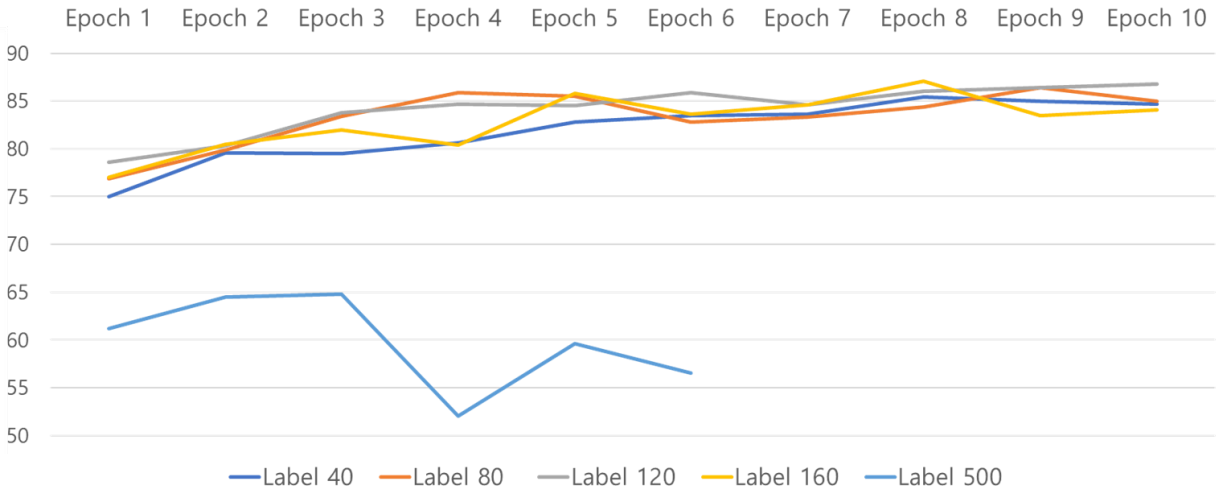
mnist threshold = 0.1



Parameter Optimization

- **Neurons_visible/hidden label**





1	75
2	79.6
3	79.5
4	80.6
5	82.8
6	83.5
7	83.6
8	85.4
9	85
10	84.7

label 60
label 160

1	76.9
2	79.9
3	83.4
4	85.9
5	85.5
6	82.8
7	83.3
8	84.4
9	86.4
10	85

1	78.6
2	80.3
3	83.8
4	84.7
5	84.5
6	85.9
7	84.6
8	86
9	86.4
10	86.8

label 80
label 500

1	77
2	80.5
3	82
4	80.4
5	85.8
6	83.6
7	84.6
8	87.1
9	83.5
10	84.1

label 120

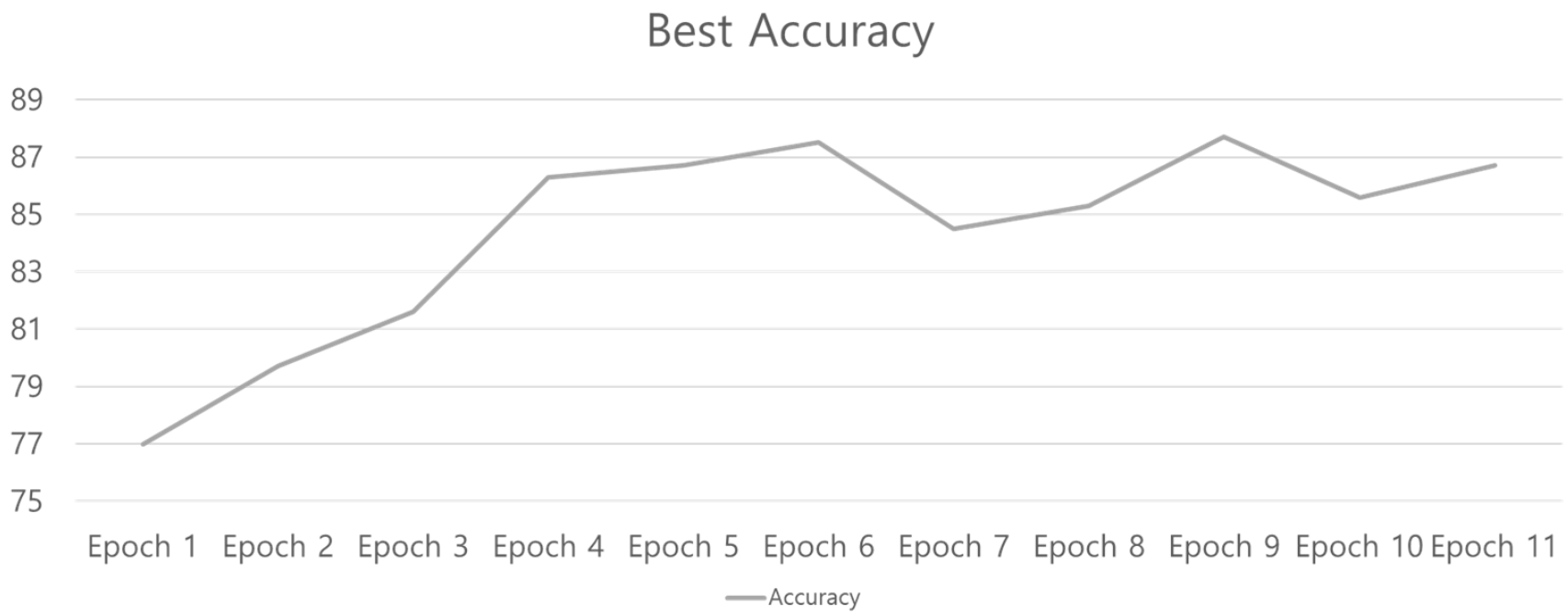
1	61.2
2	64.5
3	64.8
4	52
5	59.6
6	56.5

Parameter Optimization

- **Best Result**

Best Accuracy - 87.7% at epoch 9

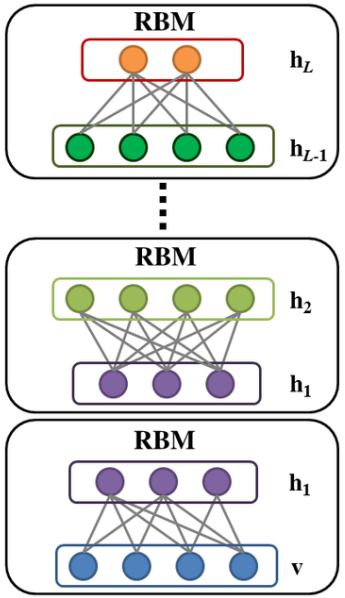
mnist th 0.5 multi = 3, epoch = 9 normalization label : 784 160 944 spike_rate = 210 Wij init seed = 3 gaussian		
	1	77
	2	79.7
	3	81.6
	4	86.3
	5	86.7
	6	87.5
	7	84.5
	8	85.3
	9	87.7
	10	85.6
	11	86.7



Future Work

- **Accuracy 90% in Epoch 10 with improved model-Spiking DBN**

1. Parameter Optimization
2. Code Optimization(Python-Google Colab etc.)
3. Idea from other models(DBN, CNN)->Spiking DBN, CNN(Others Work)



sRBM MNIST Accuracy Enhancement

Learning Rate Tuning & ANN Application

NMDL 인턴십 프로그램

임혁준

목차

Introduction

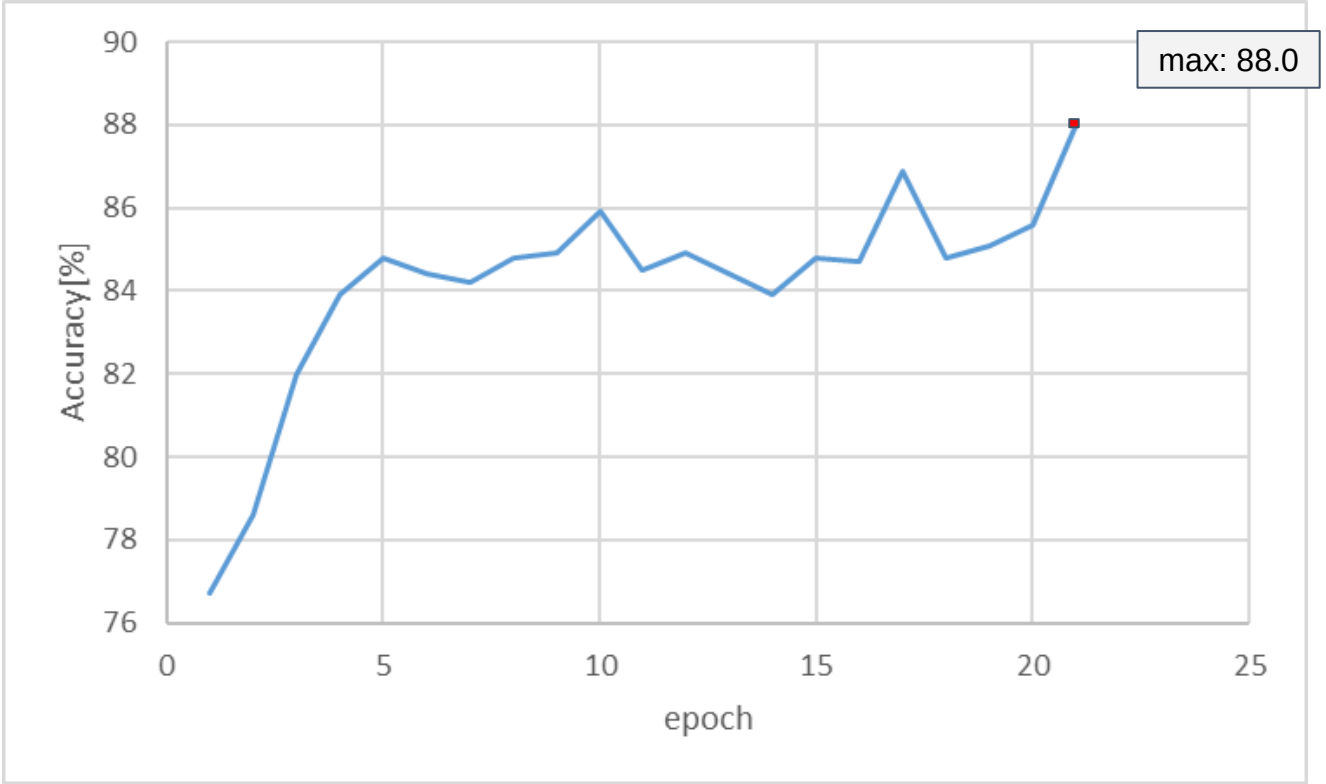
Midterm goal: Achieve accuracy 90%

Accuracy achieved was 88% at epoch 21 with:

- MNIST threshold = 0.1307

epoch 1~17
- increasing label layer 40 -> 60
  hidden layer 824 -> 844
- random walk step 0.06 -> 0.05

epoch 18~21
- step 0.05 -> 0.051, Δweight 0.01 -> 0.009



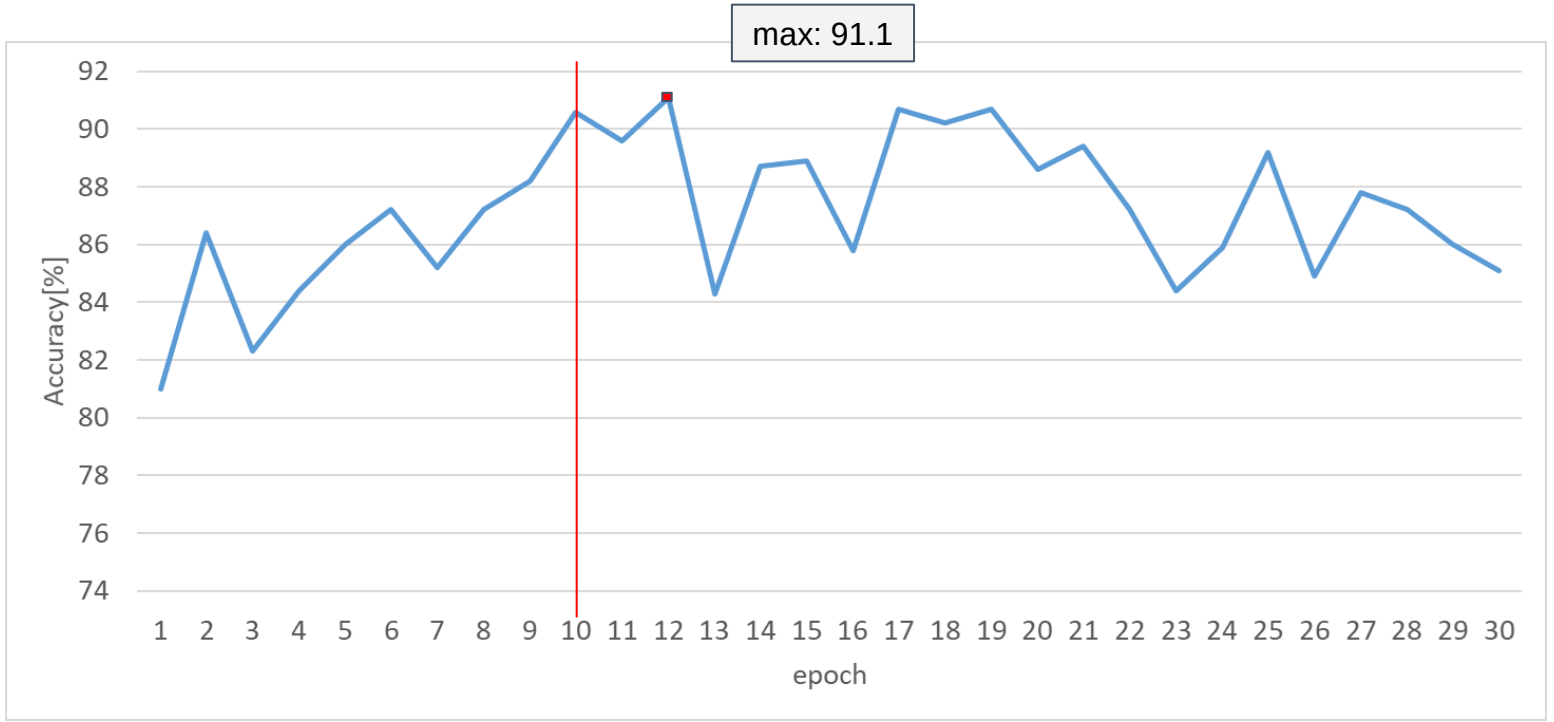
Introduction

Midterm goal: Achieve accuracy 90%

This morning, achieved the goal!

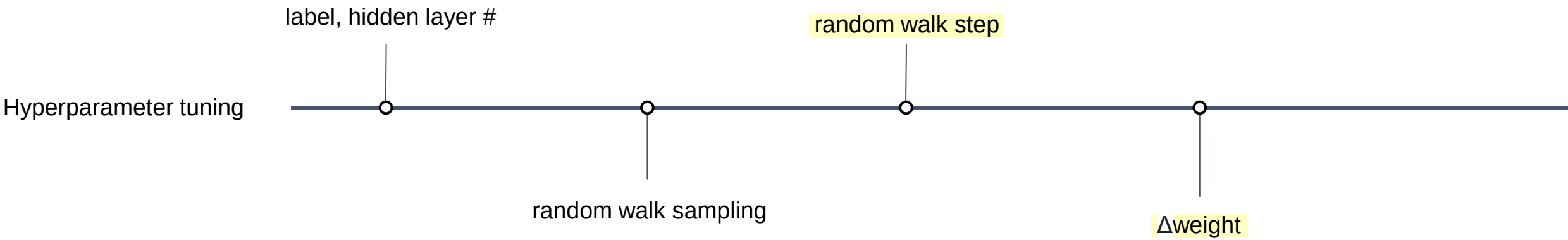
max accuracy of 91.1 at epoch 12 with:

- wij_init_sigma = 1.06 -> 0.3426
- MNIST threshold = 0.3
- label neuron number = 40 -> 120
- hidden neuron number = 824 -> 888
- CNN - sRBM (Mode 2)
  (hyperparameter tuning is not done,
  further explain in future work page)

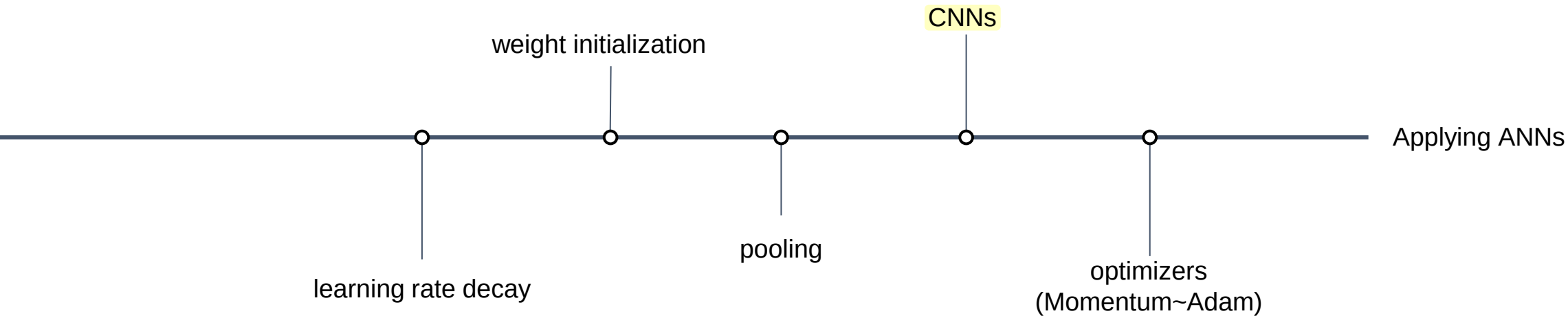


accuracy was already over 90 at 10 epoch!
accuracy decreasing in the later epochs is likely due to overfitting.

Development



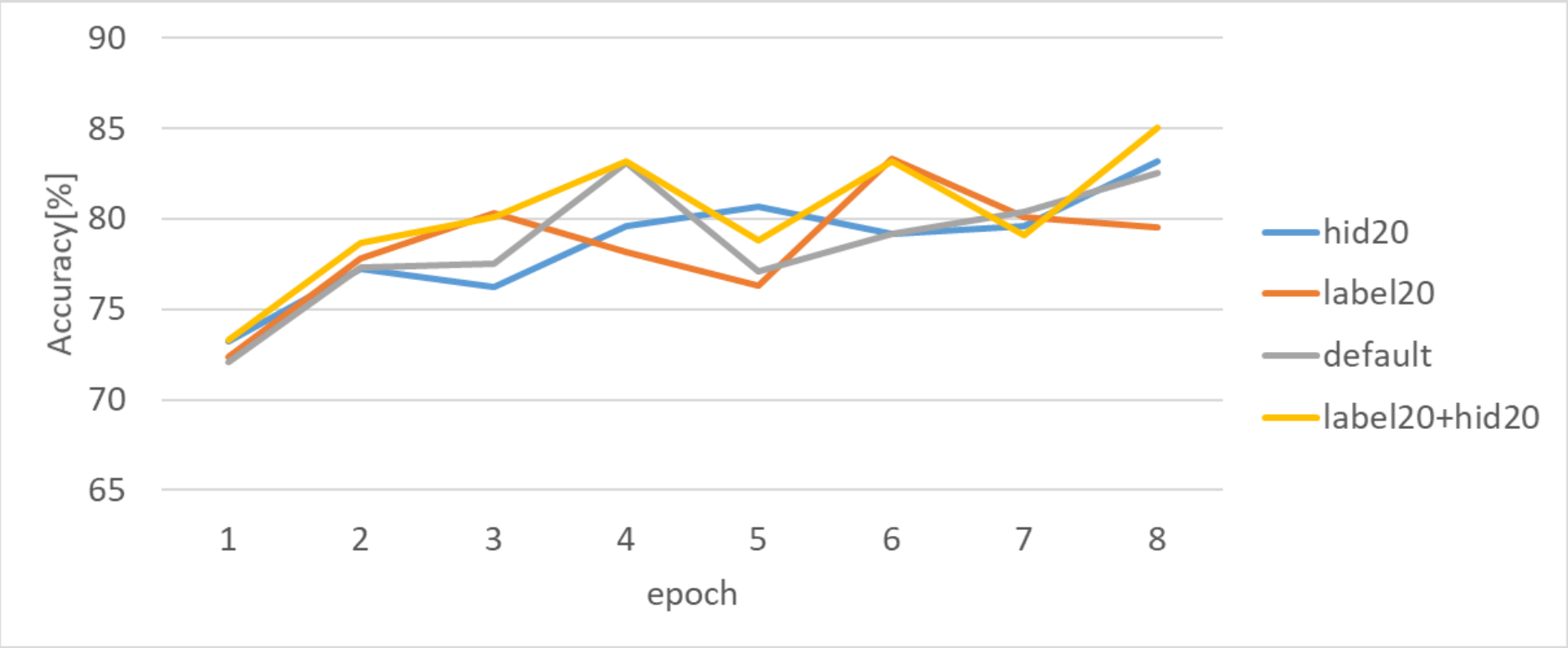
Development



Result

label, hidden #

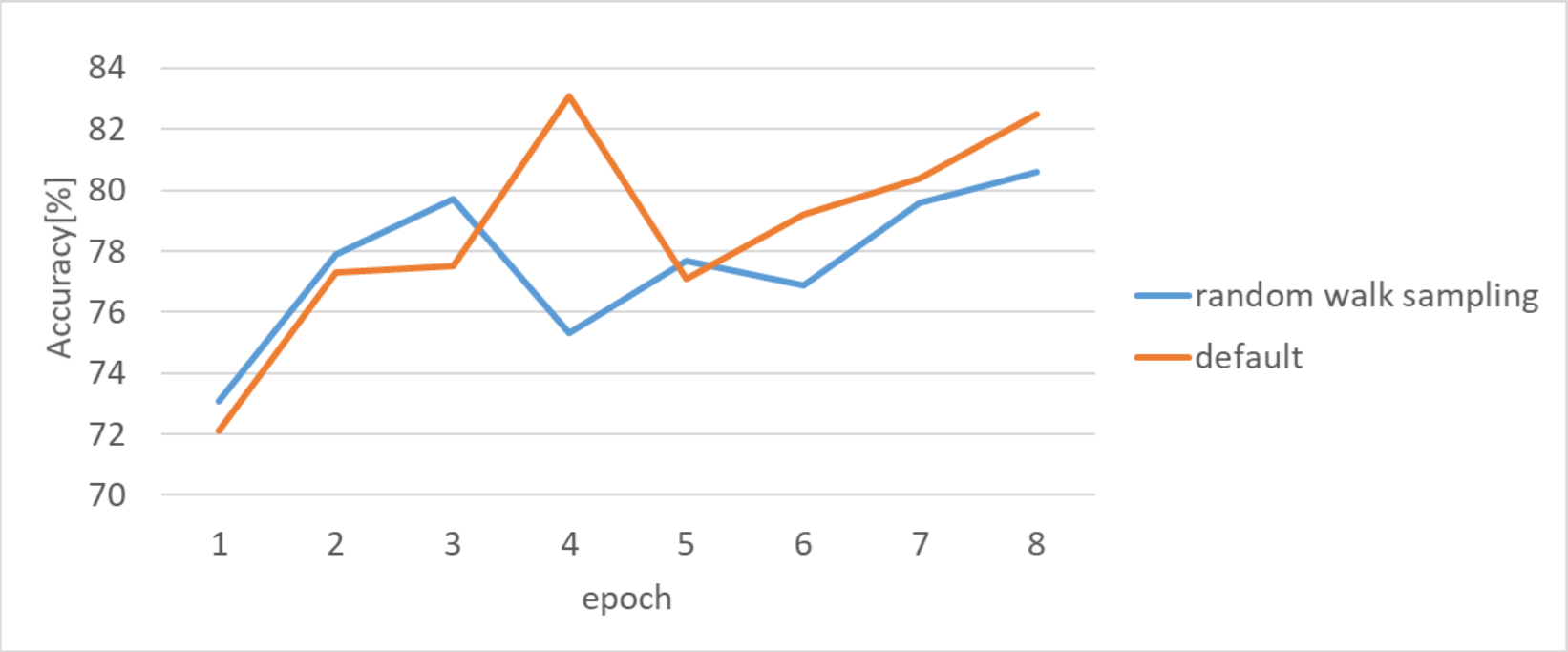
- increasing hidden layer number leads to better accuracy of early epochs
- increasing label layer number leads to better accuracy of later epochs



Result

random walk sampling

- spike timing matches to random walk tick gets random-walked
- original probability-based upward or downward movements
  -> certain probability of either going up or down, or staying the same.
- ideal sampling rate is needed

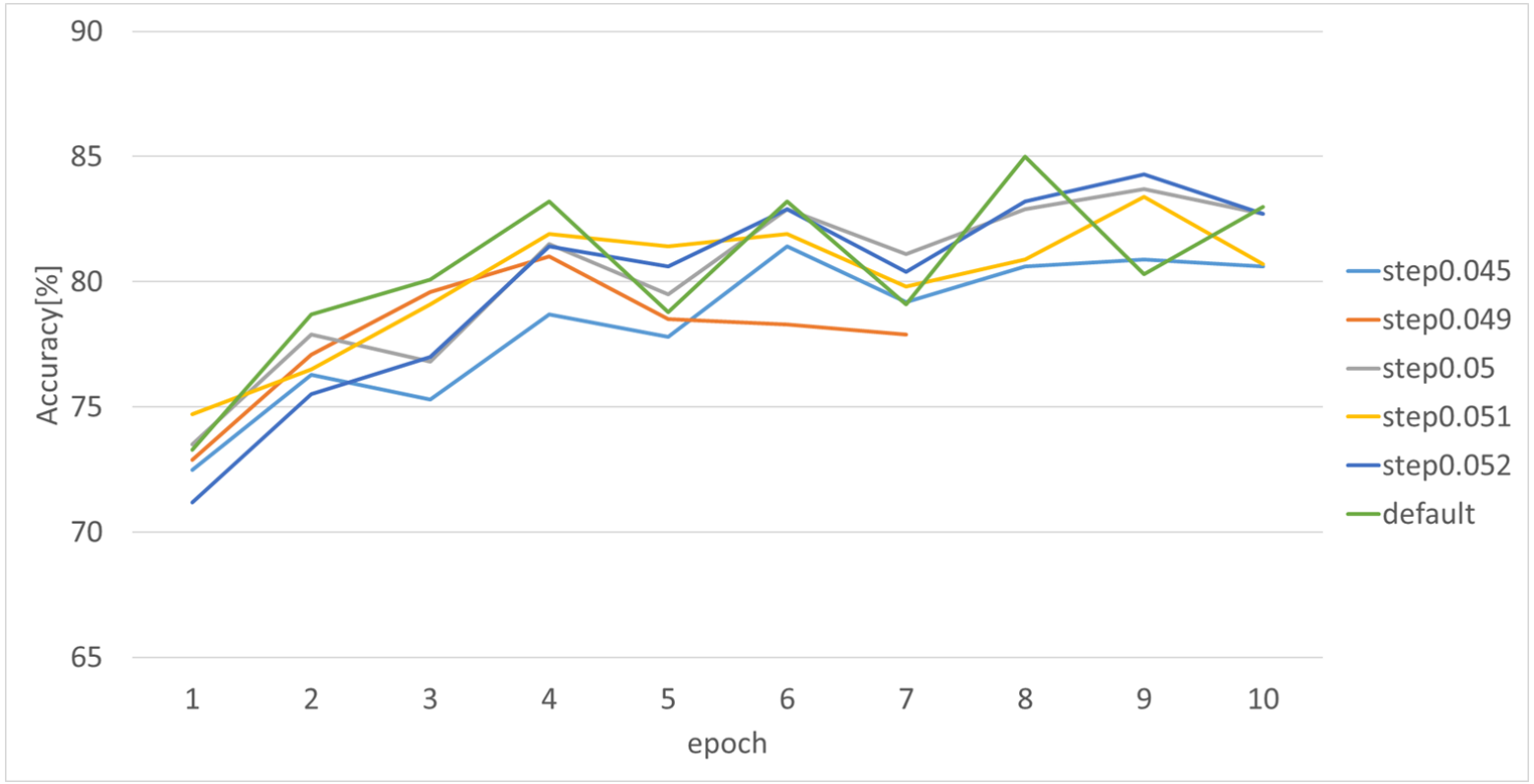


Result

learning rate

- classified into random walk step, Δweight (plus threshold annealing)
- default : label, hidden layer # increased by 20, step0.06

random walk step

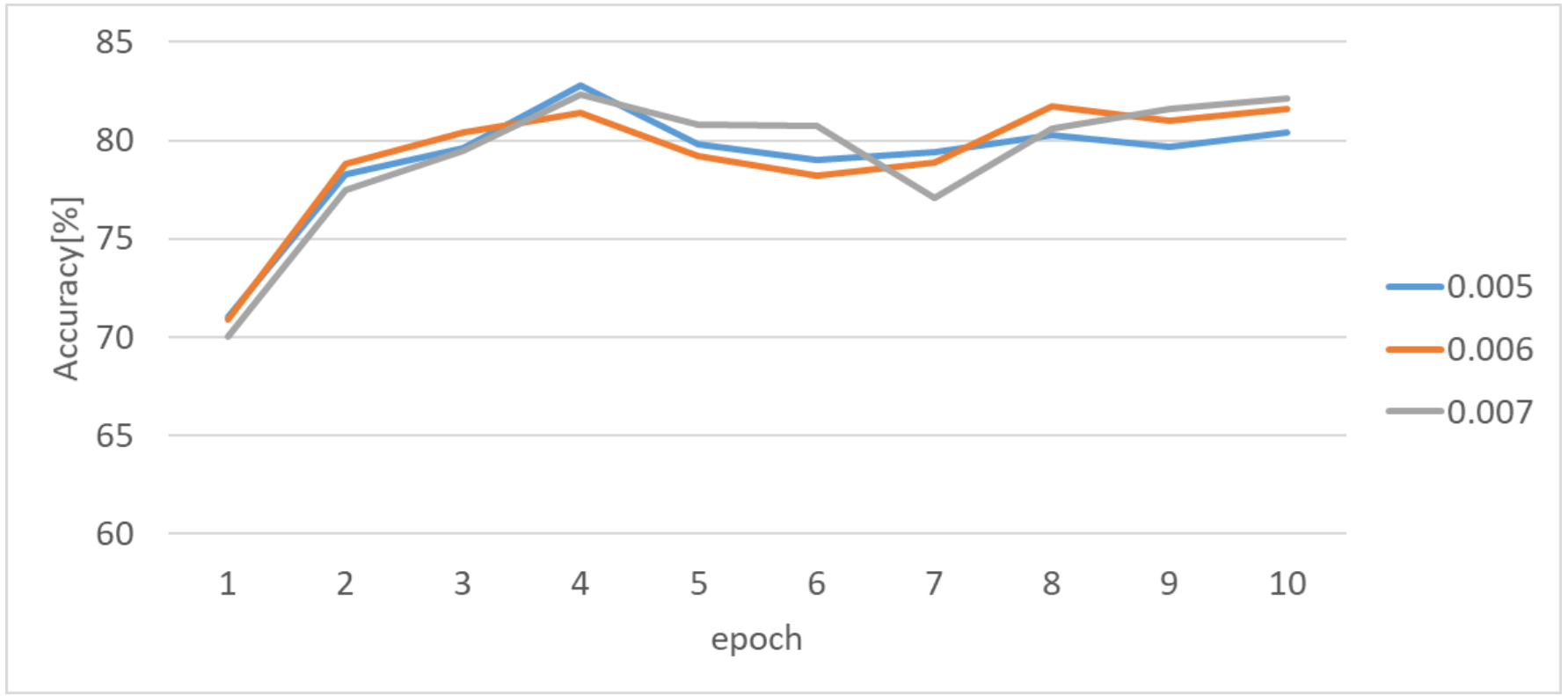


Result

learning rate

- classified into random walk step, Δweight (plus threshold annealing)
- default : label, hidden layer # increased by 20, step0.06

Δweight combination



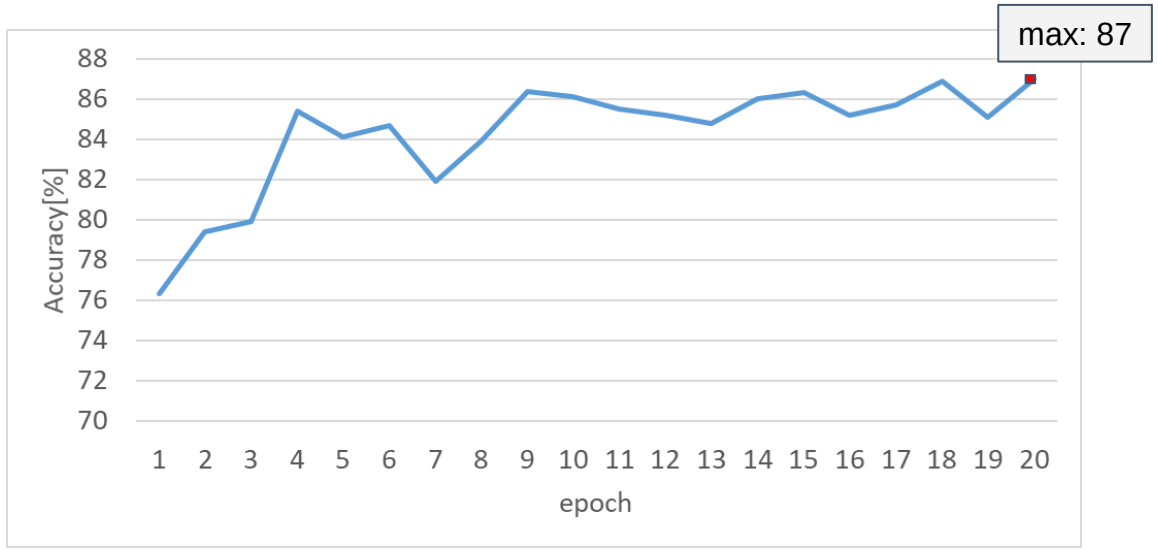
Result

learning rate

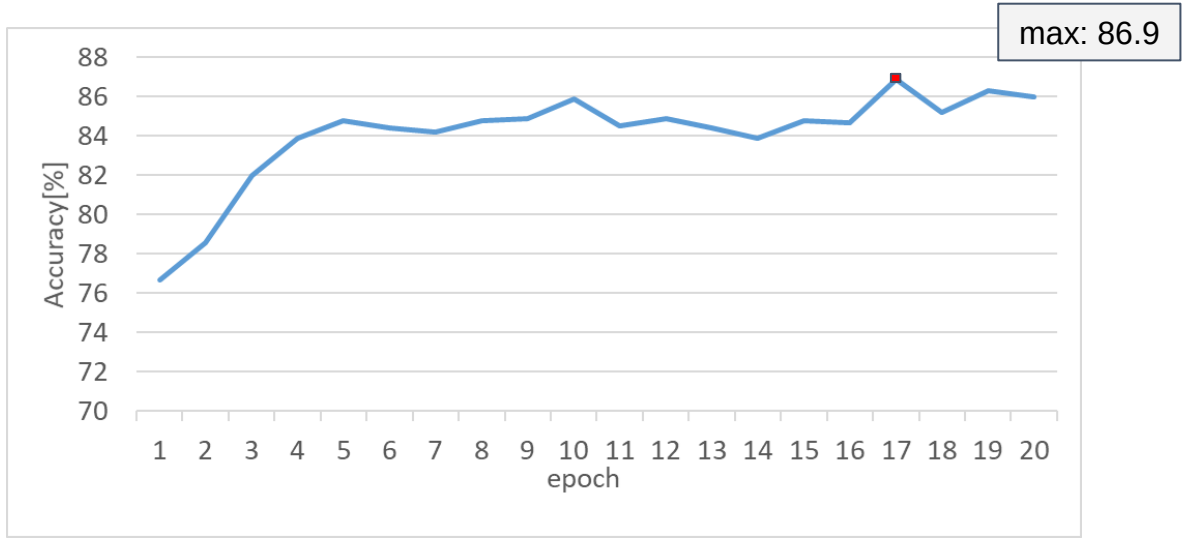
random walk step + Δweight

- MNIST threshold = 0.1307

step0.051
+ wt0.009



step0.05



Result

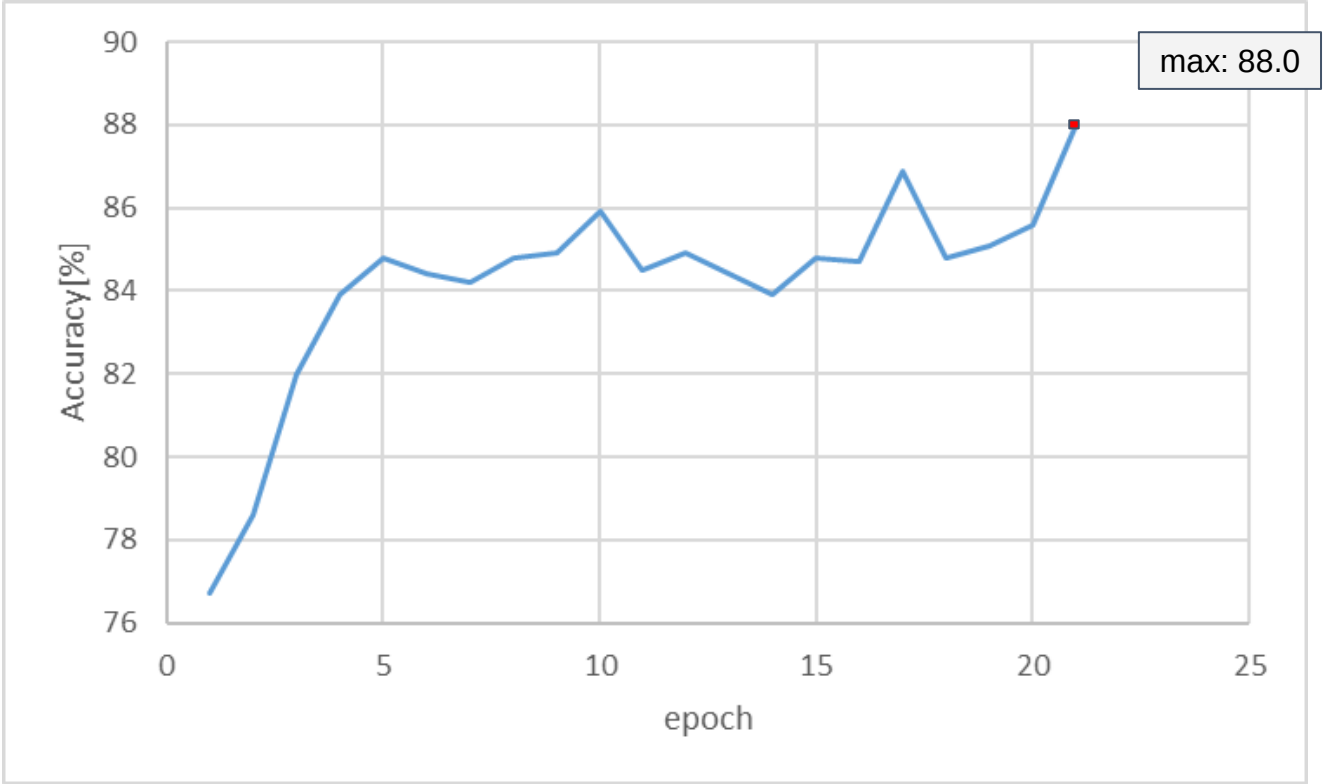
learning rate

random walk step + Δweight

- MNIST threshold = 0.1307

epoch 1~17
- increasing label layer 40 -> 60
  hidden layer 824 -> 844
- random walk step 0.06 -> 0.05

epoch 18~21
- step 0.05 -> 0.051, Δweight 0.01 -> 0.009



Result

Applying ANNs to sRBM

- weight initialization : Xavier initialization

$$W \sim N(0, \sqrt{\frac{2}{n_{in} + n_{out}}})$$

$$n_{in} = n_{out} = 852$$

$$X - Y \sim N(0, \sqrt{\frac{2}{n_{in} + n_{out}}})$$

$$X, Y \sim N(5, \frac{1}{\sqrt{2}}\sqrt{\frac{2}{n_{in} + n_{out}}})$$

- stddev = 1.06 -> 0.03426

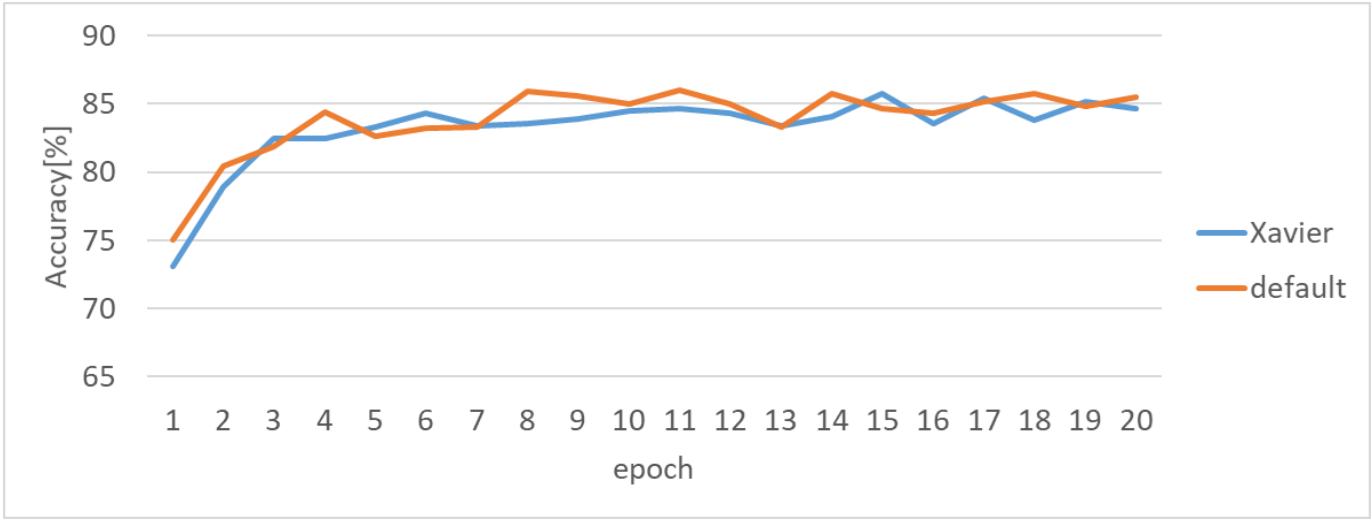
```
"wij_init_mu": 5.0,
"wij_init_sigma": 0.03426,
"wij_init_seed": 2,
```

Result

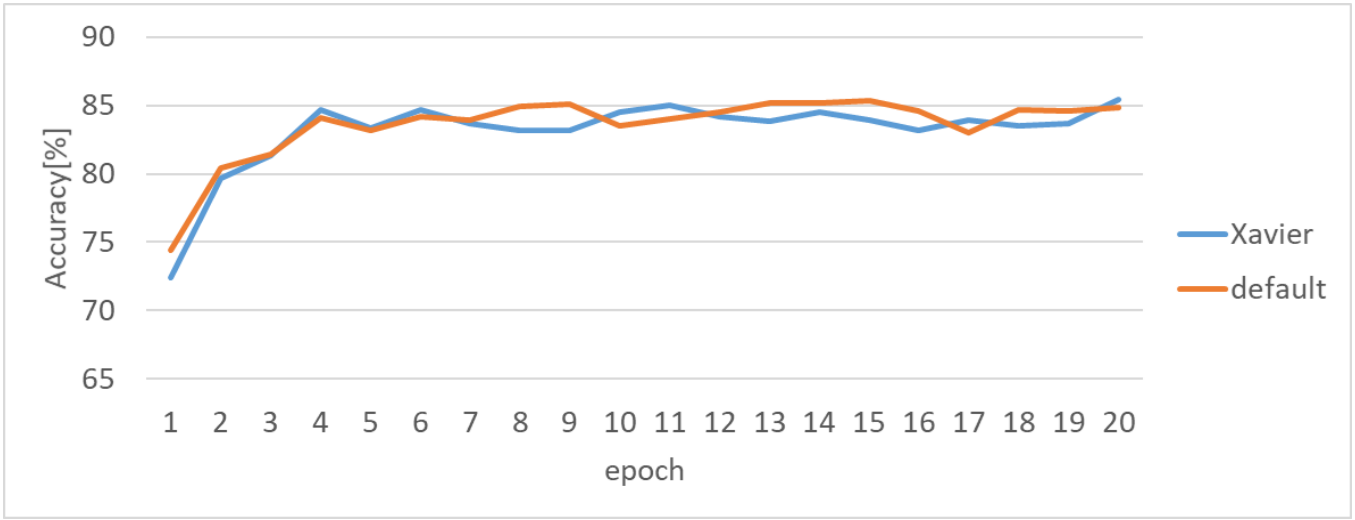
Applying ANNs to sRBM

- weight initialization : Xavier initialization
- step = 0.051, Δweight = 0.009

step0.051
+ wt0.009



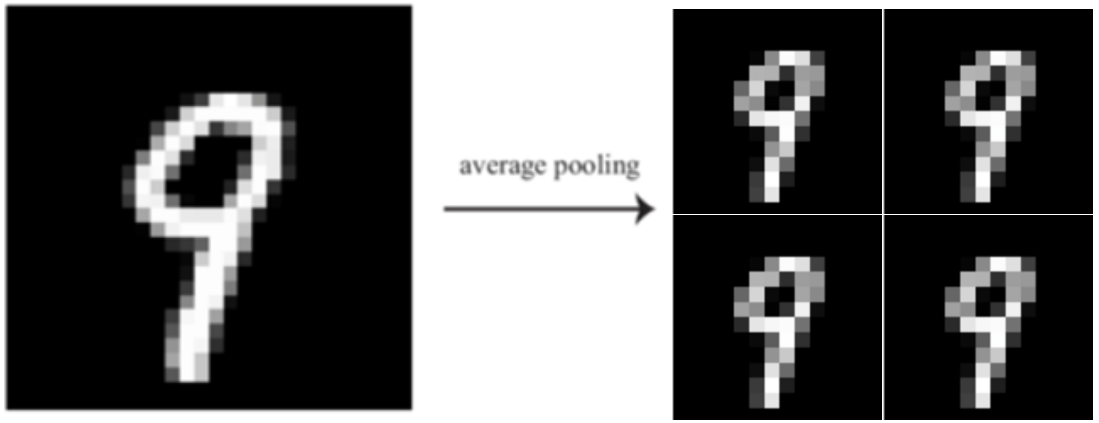
step0.05



Result

Applying ANNs to sRBM

- Average Pooling



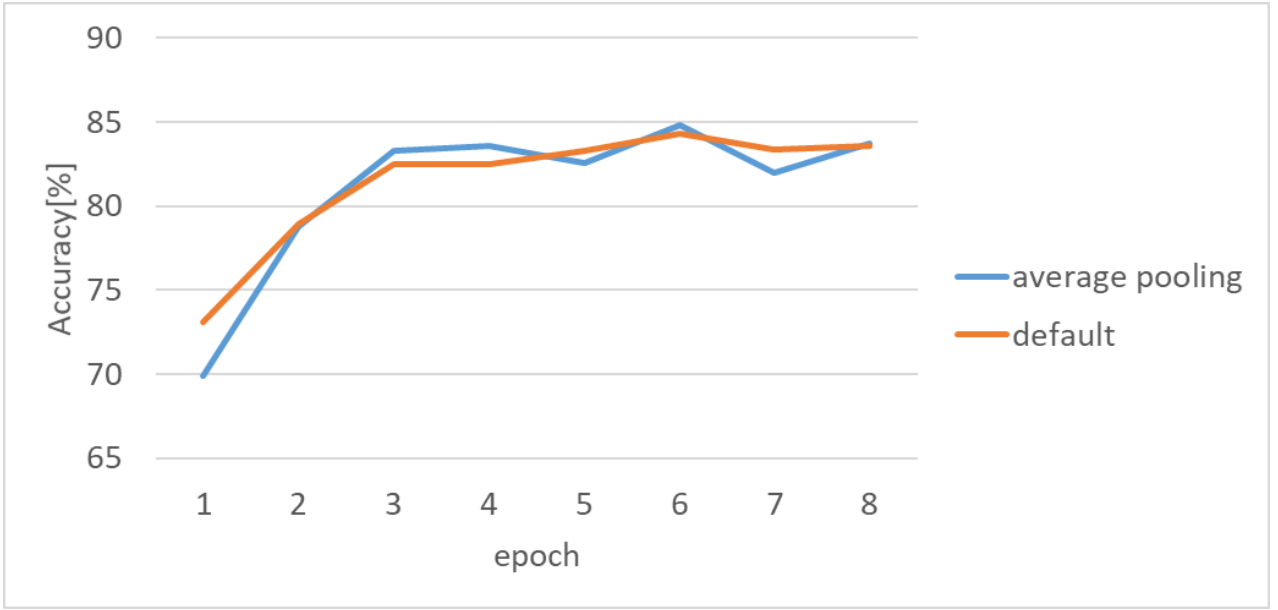

```
double sum = 0.0;
for (int i = 0; i < 14; i++) {
  for (int j = 0; j < 14; j++) {
    sum = 0.0;
    for (int k = 0; k < 2; k++) {
      for (int l = 0; l < 2; l++) {
        sum += temp_pixel[2 * (28 * i + j) + 28 * k + l];
      }
    }
    for (int k = 0; k < 2; k++) {
      for (int l = 0; l < 2; l++) {
        pixel[28 * i + j + 14 * (28 * k + l)] = sum / 4;
      }
    }
  }
}

sum = 0.0;
sq_sum = 0.0;
for (int i = 0; i < pixel.size(); i++) {
  sum += pixel[i];
  sq_sum += pow(pixel[i], 2);
}
mini_batch_avg = sum / pixel.size();
mini_batch_stdev = sqrt(sq_sum / pixel.size() - pow(mini_batch_avg, 2));
for (int i = 0; i < pixel.size(); i++) {
  pixel[i] = (pixel[i] - mini_batch_avg) / mini_batch_stdev;
}
// mnist threshold = 0
```

Result

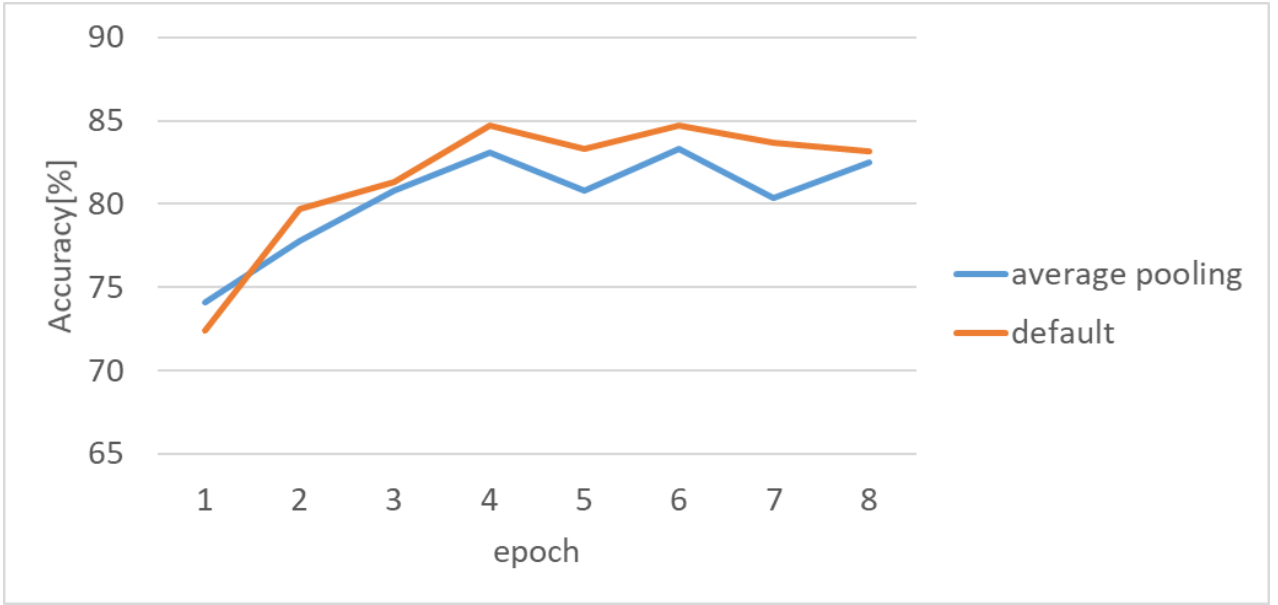
Applying ANNs to sRBM

- Average Pooling
- Δweight = 0.009
- MNIST threshold = average of the image

step0.051



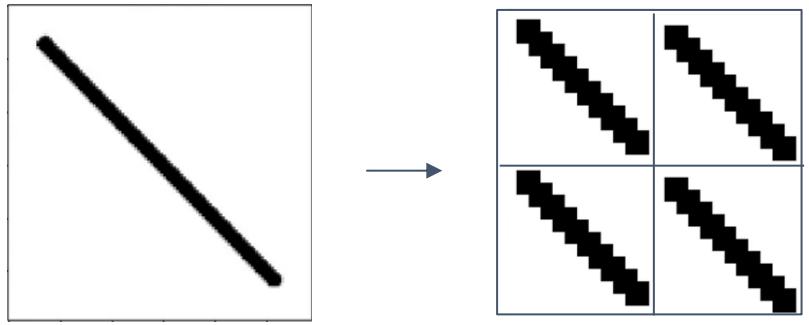
step0.05



Result

Applying ANNs to sRBM

- Max Pooling
- failed, may due to too simplified feature

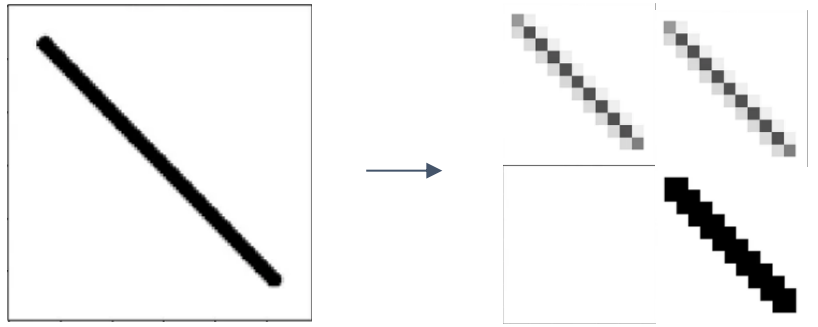


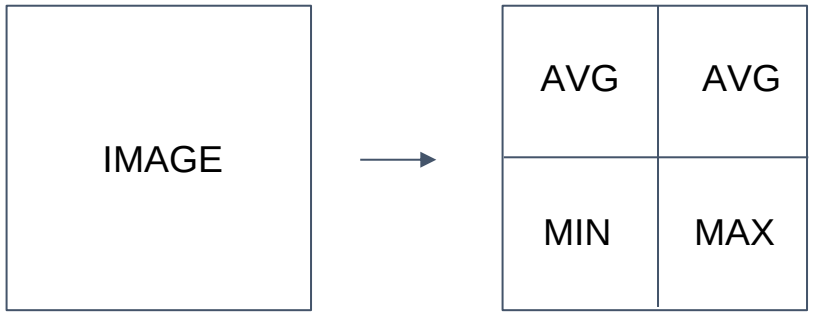
```
double pool_max = -1;
double pool_element = 0;
for (int i = 0; i < 14; i++) {
  for (int j = 0; j < 14; j++) {
    pool_max = -1;
    for (int k = 0; k < 2; k++) {
      for (int l = 0; l < 2; l++) {
        pool_element = temp_pixel[2 * (28 * i + j) + 28 * k + l];
        pool_max = (pool_element > pool_max) ? pool_element : pool_max;
      }
    }
    for (int k = 0; k < 2; k++) {
      for (int l = 0; l < 2; l++) {
        pixel[28 * i + j + 14 * (28 * k + l)] = pool_max;
      }
    }
  }
}
sum = 0.0;
sq_sum = 0.0;
for (int i = 0; i < pixel.size(); i++) {
  sum += pixel[i];
  sq_sum += pow(pixel[i], 2);
}
mini_batch_avg = sum / pixel.size();
mini_batch_stdev = sqrt(sq_sum / pixel.size() - pow(mini_batch_avg, 2));
for (int i = 0; i < pixel.size(); i++) {
  pixel[i] = (pixel[i] - mini_batch_avg) / mini_batch_stdev;
}
// mnist threshold = 0
```

Result

Applying ANNs to sRBM

- Pooling combination





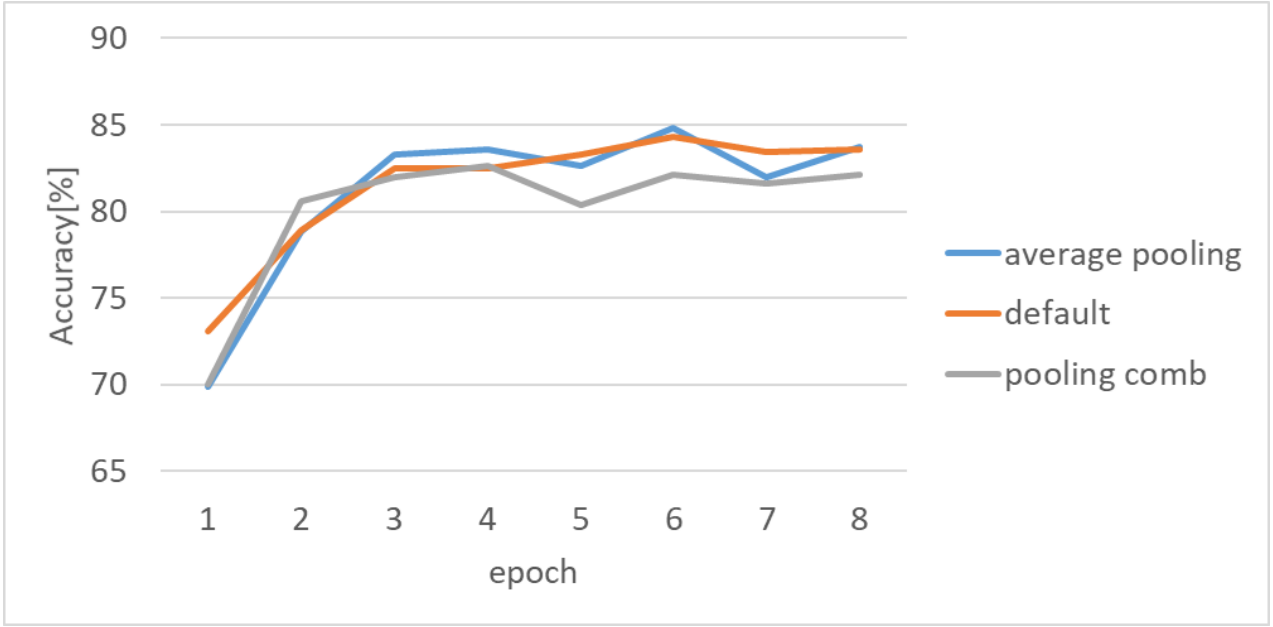

```
for (int i = 0; i < 14; i++) {
  for (int j = 0; j < 14; j++) {
    pool_max = -1;
    pool_min = 1;
    sum = 0;
    for (int k = 0; k < 2; k++) {
      for (int l = 0; l < 2; l++) {
        pool_element = temp_pixel[2 * (28 * i + j) + 28 * k + l];
        sum += pool_element;
        pool_max = (pool_element > pool_max) ? pool_element : pool_max;
        pool_min = (pool_element < pool_min) ? pool_element : pool_min;
      }
    }
    pixel[28 * i + j + 14 * (28 * 0 + 0)] = sum / 4;
    pixel[28 * i + j + 14 * (28 * 0 + 1)] = sum / 4;
    pixel[28 * i + j + 14 * (28 * 1 + 0)] = pool_max;
    pixel[28 * i + j + 14 * (28 * 1 + 1)] = pool_min;
    sum_0 += sum / 4;
    sum_1_0 += pool_max;
    sum_1_1 += pool_min;
  }
}

for (int i = 0; i < 14; i++) {
  for (int j = 0; j < 14; j++) {
    pixel[28 * i + j + 14 * (28 * 0 + 0)] = pixel[28 * i + j + 14 * (28 * 0 + 0)] - sum_0 / (784 / 4);
    pixel[28 * i + j + 14 * (28 * 0 + 1)] = pixel[28 * i + j + 14 * (28 * 0 + 1)] - sum_0 / (784 / 4);
    pixel[28 * i + j + 14 * (28 * 1 + 0)] = pixel[28 * i + j + 14 * (28 * 1 + 0)] - sum_1_0 / (784 / 4);
    pixel[28 * i + j + 14 * (28 * 1 + 1)] = pixel[28 * i + j + 14 * (28 * 1 + 1)] - sum_1_1 / (784 / 4);
  }
}
// mnist threshold = 0
```

Result

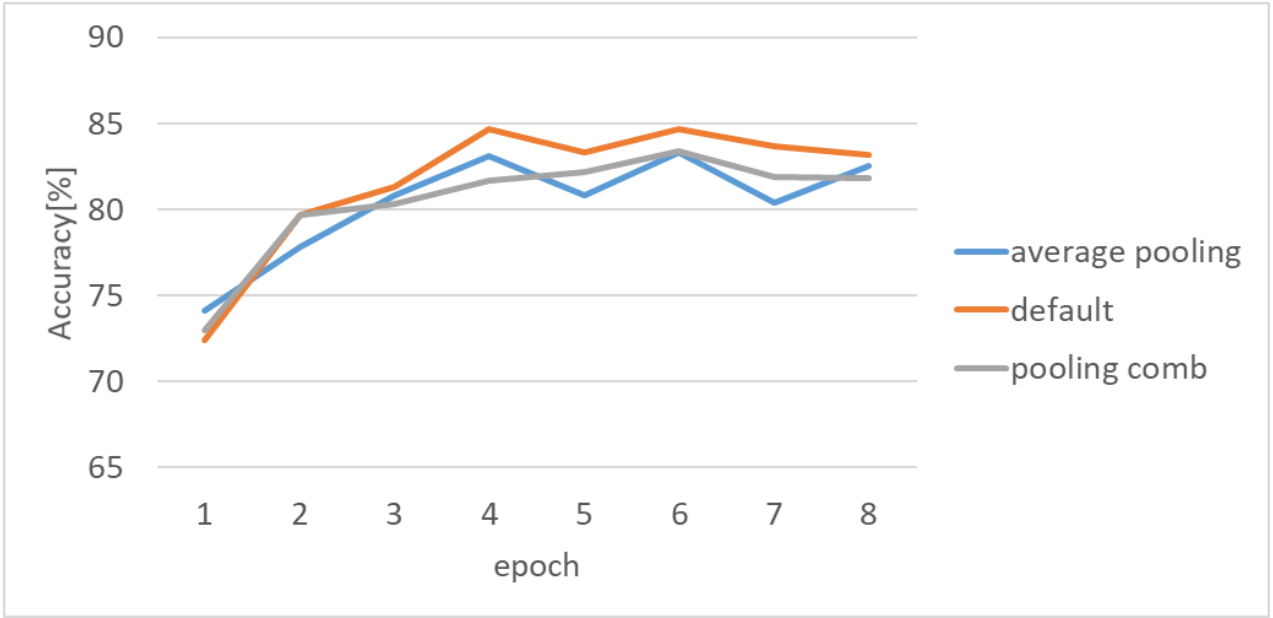
Applying ANNs to sRBM

- Pooling combination
- Δweight = 0.009
- MNIST threshold = average of the image

step0.051



step0.05



Future work

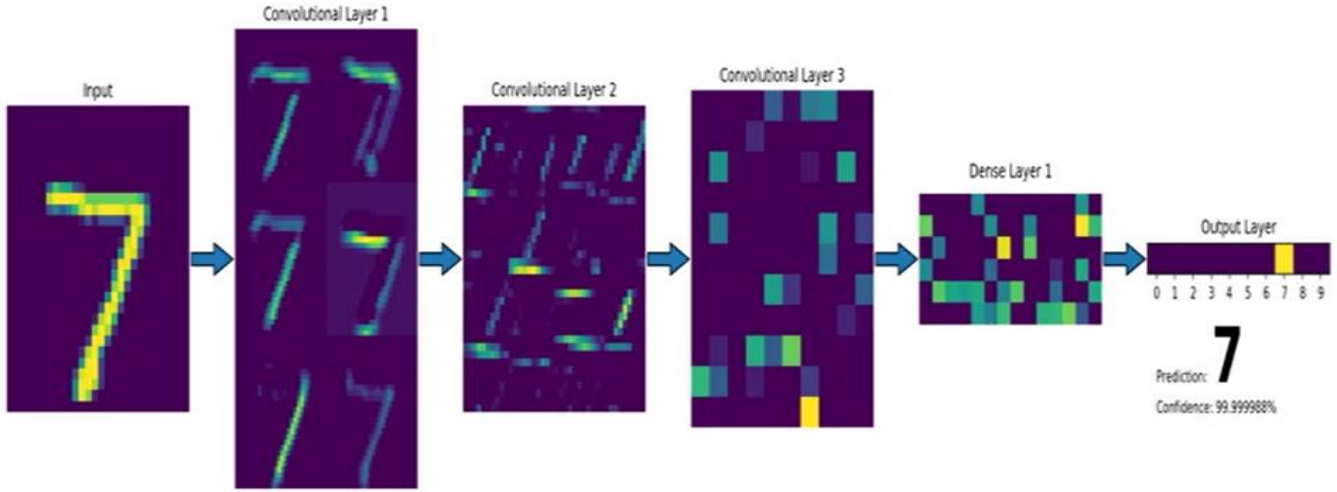
Applying CNNs to sRBM

format of input data to the visible layer is changed.

- the pixel brightness of MNIST data
  -> values obtained after passing through the pre-trained CNN model.
- Once CNN training is performed initially, there is no need to repeat it.

From the idea of:

Step 1. sRBM responses to the yellow part of MNIST image.
Step 2. How about inputting image of Conv layer 1(recognizable),
which has important characteristics of MNIST image?
Step 3. Then, though we cannot recognize the information which the image of Conv layer 3 have,
how about inputting the image which passed through all pre-trained CNN layers?



Future work

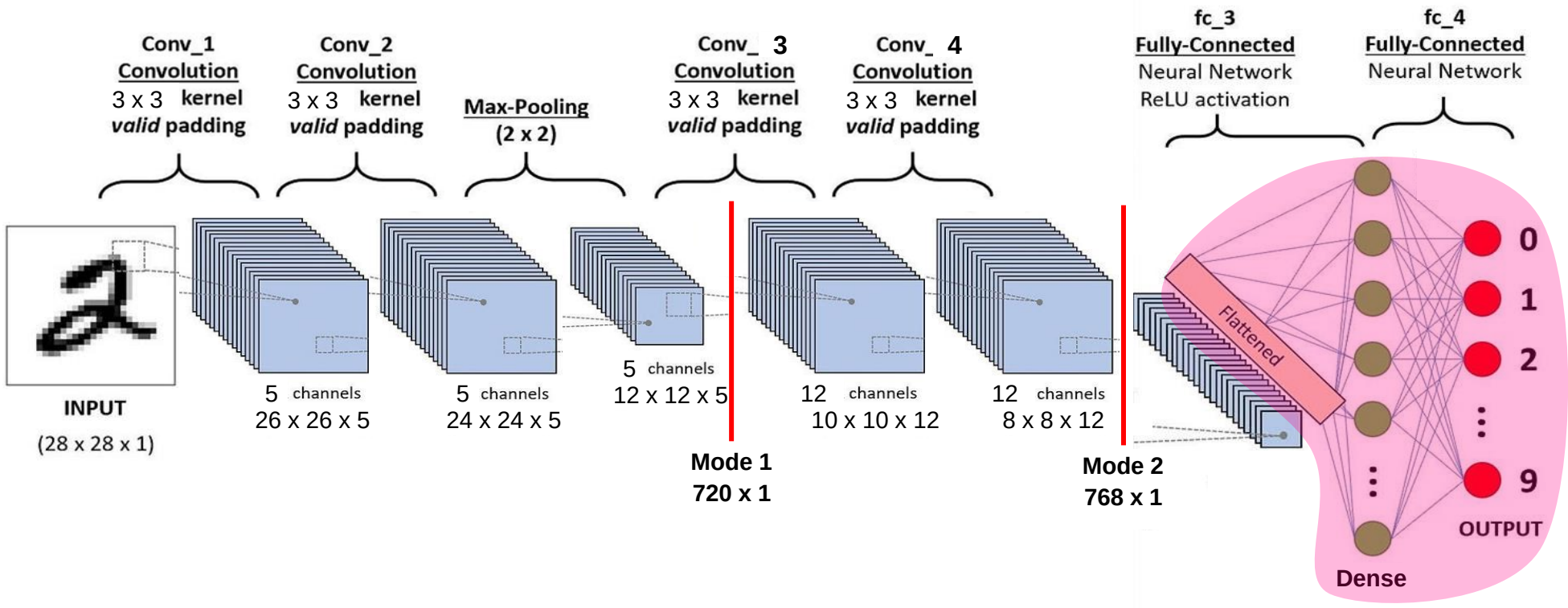
Applying CNNs to sRBM

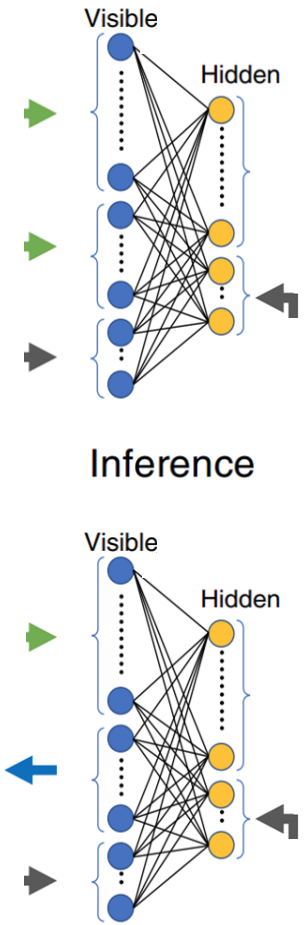
Flattened Dense Output → corresponding to → visible hidden inference

looks similar!

hypothesis:

- pink part of CNN model(Flattened - Dense - Output) is the equivalent of sRBM.





Future work

Applying CNNs to sRBM

Advantage

- **From the point of CNN**
  Significant amount of computing resources is used to store gradients during the backpropagation process of CNN.
  However, replacing it with sRBM simulation can offer advantage in terms of energy efficiency.
  As we can implement sRBM simulation in actual module, this can be possible way to implement CNN in real module.

- **Can take advantage of the idea of filters**
  By accepting the computing resources required for the image preprocessing stage when the MNIST data is input,
  we expect to improve accuracy by taking advantage of using filters in CNN models, which have special power in image processing.

- **Can lower visible neuron number (Mode 1: 720, Mode 2: 768)**
  If the model is modified, visible neuron number can be further lowered.
  This approach allows us to increase the number of label layers(anticipated improvement in accuracy)
  Furthermore, by keeping other parameters constant, we can expect a decrease in energy consumption.

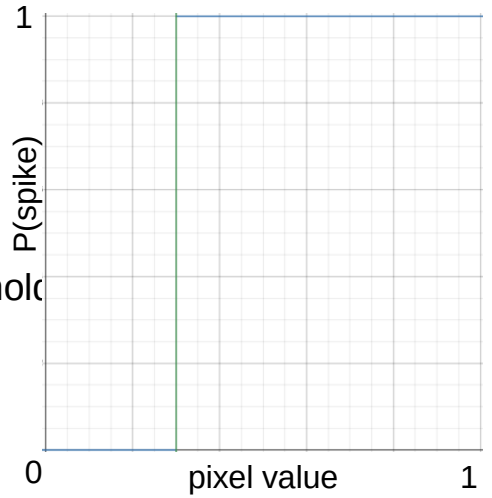
Future work

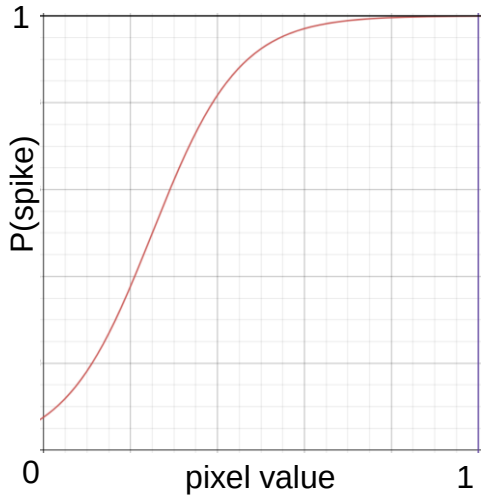
Applying CNNs to sRBM

Challenge

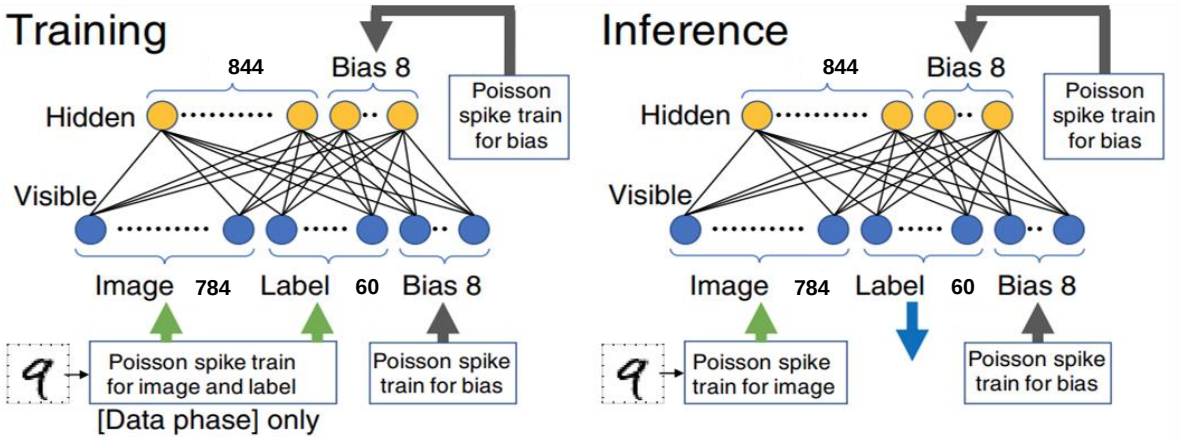
- **Information of the image is lost**
  probability of spike is constant for all the values over MNIST threshold
  sRBM may recognize them as the same.
  P(spike) can be modified based on pixel value

- **Difficulty to implement in actual module?**
  only circuits for calculate is needed.





Future work



similar structure of sRBM and CNN

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 5)	50
activation (Activation)	(None, 26, 26, 5)	0
conv2d_1 (Conv2D)	(None, 24, 24, 5)	230
activation_1 (Activation)	(None, 24, 24, 5)	0
max_pooling2d (MaxPooling2D)	(None, 12, 12, 5)	0
conv2d_2 (Conv2D)	(None, 10, 10, 12)	552
activation_2 (Activation)	(None, 10, 10, 12)	0
conv2d_3 (Conv2D)	(None, 8, 8, 12)	1308
activation_3 (Activation)	(None, 8, 8, 12)	0
flatten (Flatten)	(None, 768)	0
dense (Dense)	(None, 128)	98432
dense_1 (Dense)	(None, 10)	1290

Future work

```
for (int m = 0; m < 12; m++) {
  for (int i = 0; i < 8; i++) {
    for (int j = 0; j < 8; j++) {
      conv_element = fourth_kernal_bias[m];
      for (int n = 0; n < 12; n++) {
        for (int k = 0; k < 3; k++) {
          for (int l = 0; l < 3; l++) {
            conv_element += after_conv3[n][10 * i + j + 10 * k + l] * fourth_kernal_weight[k][l][n][m];
          }
        }
      }
      after_conv4[m][8 * i + j] = conv_element;
    }
  }
  for (int i = 0; i < 8; i++) {
    for (int j = 0; j < 8; j++) {
      after_conv4[m][8 * i + j] = (after_conv4[m][8 * i + j] > 0) ? after_conv4[m][8 * i + j] : 0;
    }
  }
  for (int i = 0; i < 8; i++) {
    for (int j = 0; j < 8; j++) {
      pixel[64 * m + 8 * i + j] = after_conv4[m][8 * i + j];
    }
  }
}
```

Future work

Applying ANNs to sRBM

- Optimizer: Momentum

$$-\frac{\partial J}{\partial w} = \Delta (G_p - G_m)$$

```
if (simtick > count_transition_after_model + FLOAT_EPSILON_TIME) {
    for (int i = 0; i < N_visible; i++) {
        for (int j = 0; j < N_hidden; j++) {
            momentums[i][j].velocity = params.gamma * momentums[i][j].velocity + params.learning_rate * ((synapses[i][j].gp - synapses[i][j].gm) - synapses[i][j].save);
            synapses[i][j].gp += momentums[i][j].velocity;
            synapses[i][j].save = synapses[i][j].gp - synapses[i][j].gm;
        }
    }
    count_transition_after_model += img_period;
}
```

Future work

Applying ANNs to sRBM

- Optimizer: Adam

$$-\frac{\partial J}{\partial w} = \Delta\left(G_p - G_m\right)$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta J(\theta)$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta J(\theta))^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta = \theta - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}\hat{m}_t$$

$$\omega_{t+1} = \omega_t - \mathrm{m_t}\frac{\eta}{\sqrt{\mathrm{v_t} + \epsilon}}$$

Reference

[1] Neftci, E.O., Das, S., Pedroni, B.U., Kreutz-Delgado, K., & Cauwenberghs, G. (2013). Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in Neuroscience, 7*.

[2] Shin, U., Ishii, M., Okazaki, A., Ito, M., Rasch, M.J., Kim, W., Nomura, A., Choi, W., Koh, D., Hosokawa, K., BrightSky, M.J., Munetoh, S., & Kim, S. (2022). Pattern Training, Inference, and Regeneration Demonstration Using On-Chip Trainable Neuromorphic Chips for Spiking Restricted Boltzmann Machine. *Advanced Intelligent Systems, 4*.

[3] Kim, S., Ishii, M., Lewis, S.C., Perri, T., BrightSky, M.J., Kim, W., Jordan, R., Burr, G.W., Sosa, N.E., Ray, A., Han, J., Miller, C., Hosokawa, K., & Lam, C.H. (2015). NVM neuromorphic core with 64k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous in-situ learning. *2015 IEEE International Electron Devices Meeting (IEDM)*, 17.1.1-17.1.4.

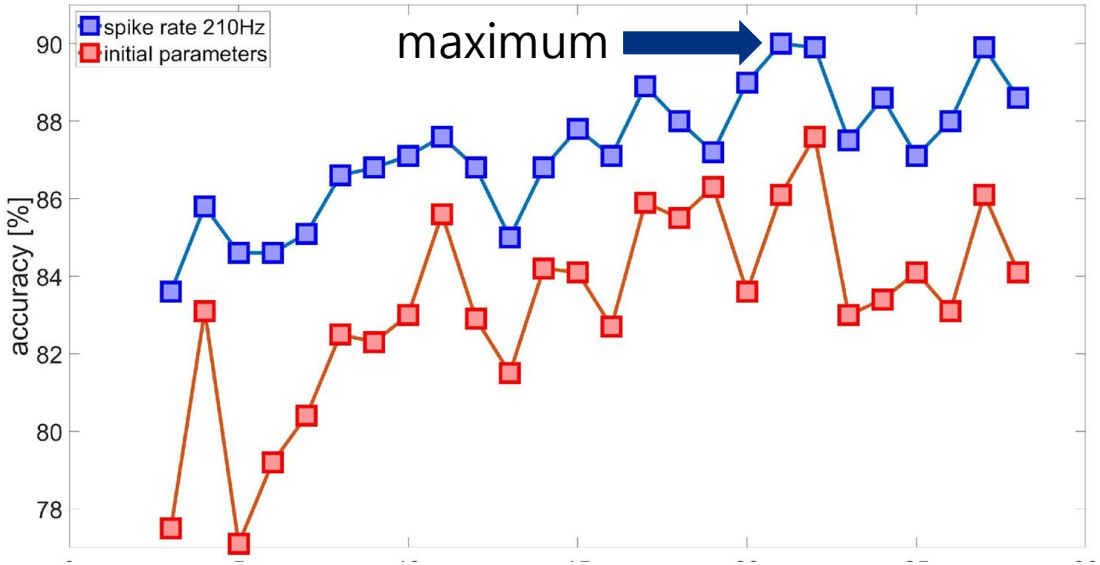
sRBM 시뮬레이션

NMDL 인턴십 프로그램

조윤

Midterm goal

Midterm goal: Achieve 90% accuracy -> achieved by just changing spike rate to 210 Hz (initial value : 200 Hz)

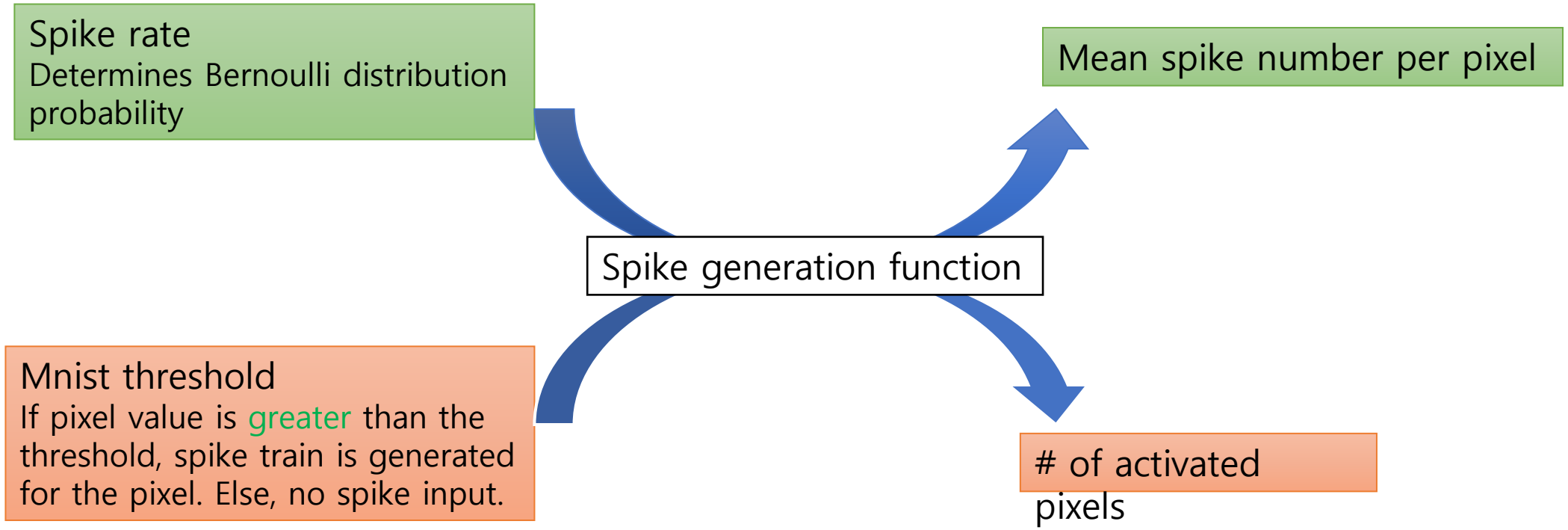


max accuracy 90% at epoch 21

First thought: Increasing spike rate will make SNN more ANN-like, thus increasing accuracy but also consumes more power

However, further increase in spike rate, above 230, shows decrease in accuracy
-> analysis of input spike generation needed!

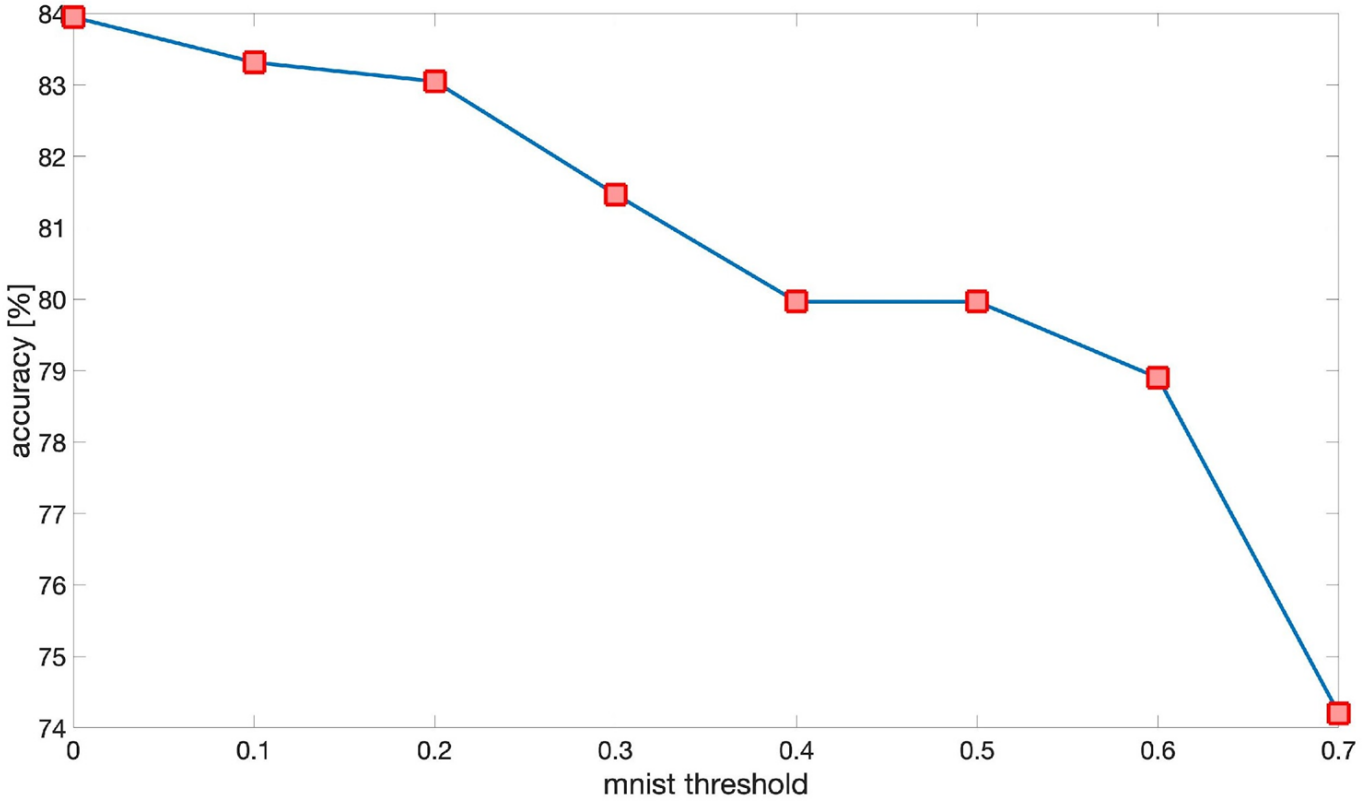
Input spike analysis



Hypothesis: there exists an optimal spike rate & mnist threshold that maximizes accuracy

Input spike analysis

1. Mnist threshold: Fixed spike rate at 200Hz(other parameters are also fixed at initial values) , accuracy averaged for epoch 3~8



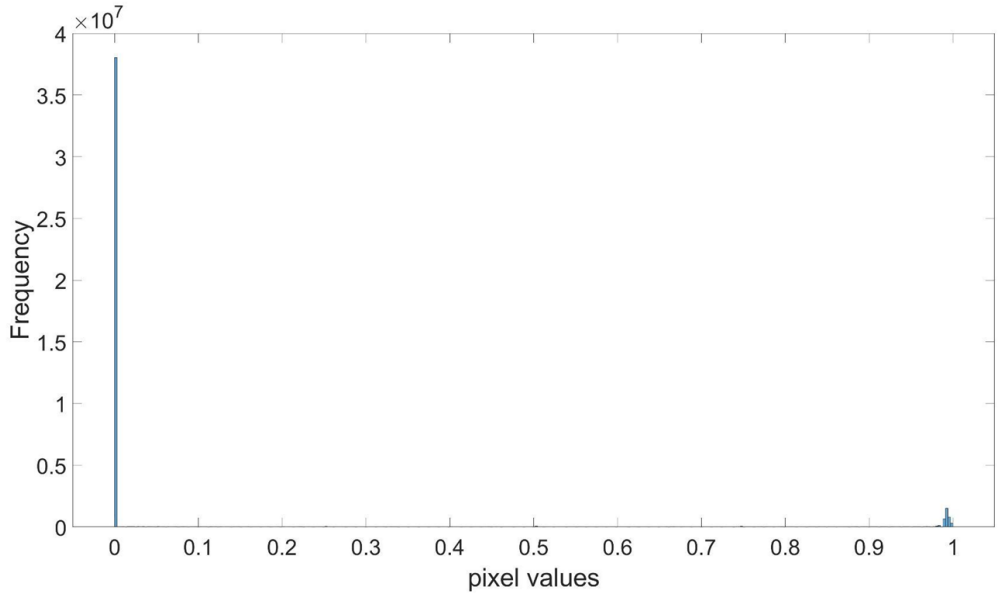
Decreasing mnist threshold thus, increasing the number of activated pixels result in higher accuracy

Optimal mnist threshold: 0

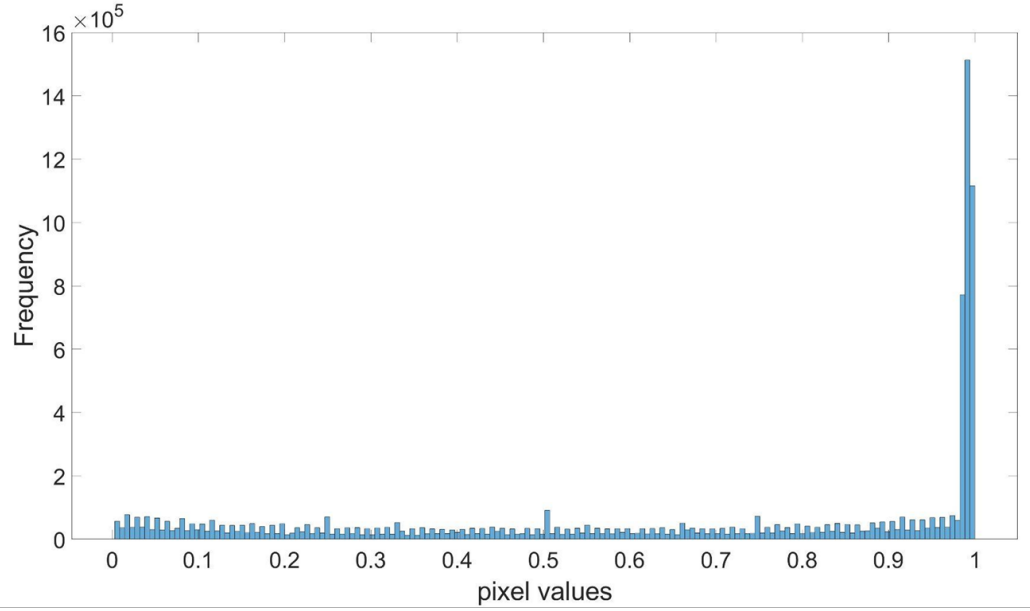
Input spike analysis

Training image pixel value distribution

with zeros



without zeros

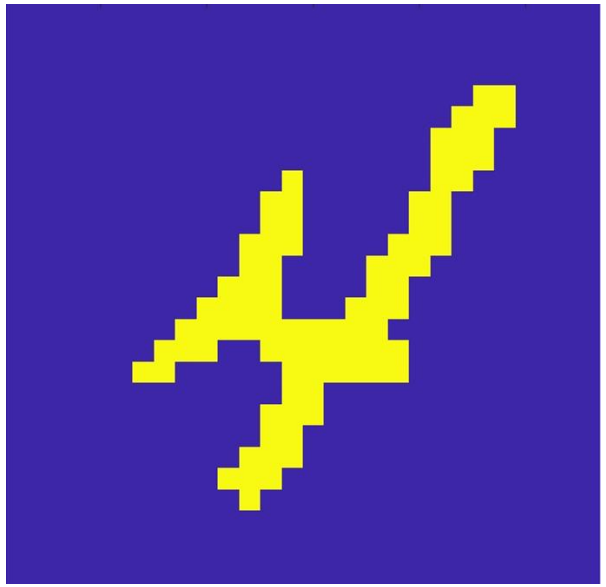


If, pixel value > mnist threshold, spike generation
Else, no input spike

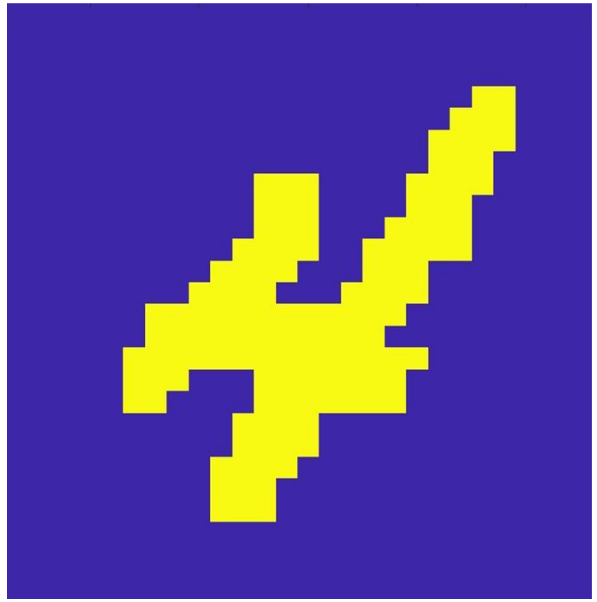
Input spike analysis

Example of image when mnist threshold changes

Mnist threshold = 0.5



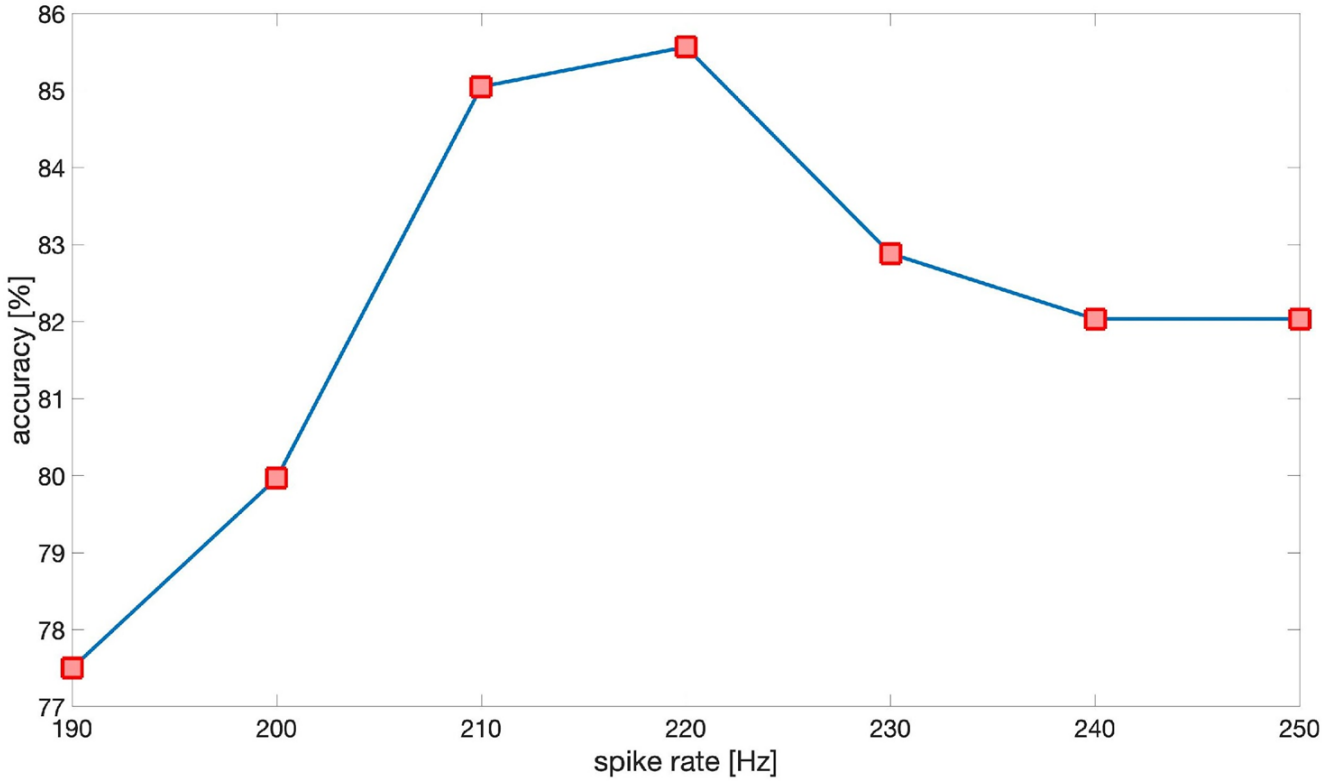
Mnist threshold = 0



Analysis: Due to the distribution of the mnist dataset, decreasing the threshold does not have a big impact on the image. However, it increases the total number of spike input, and may provide more learning opportunities for the neural network.

Input spike analysis

2. Spike rate: Fixed mnist threshold at 0.5(other parameters are also fixed at initial values), accuracy averaged for epoch 3~8



Until 220Hz, increasing spike rate improves accuracy, however increasing spike rate further (above 230Hz), accuracy decreases.

Optimal spike rate : 220Hz

Input spike analysis

Why does accuracy decrease when spike rate is above 230Hz ?

Current guess: the leak and random walk(noise) factor in the LIF neuron model contribute to the behavior of the accuracy -> detailed research needed

Neuron's membrane potential

$$C\frac{d}{dt}u_i = -g_L u_i + I_i(t) + \sigma\xi(t)$$

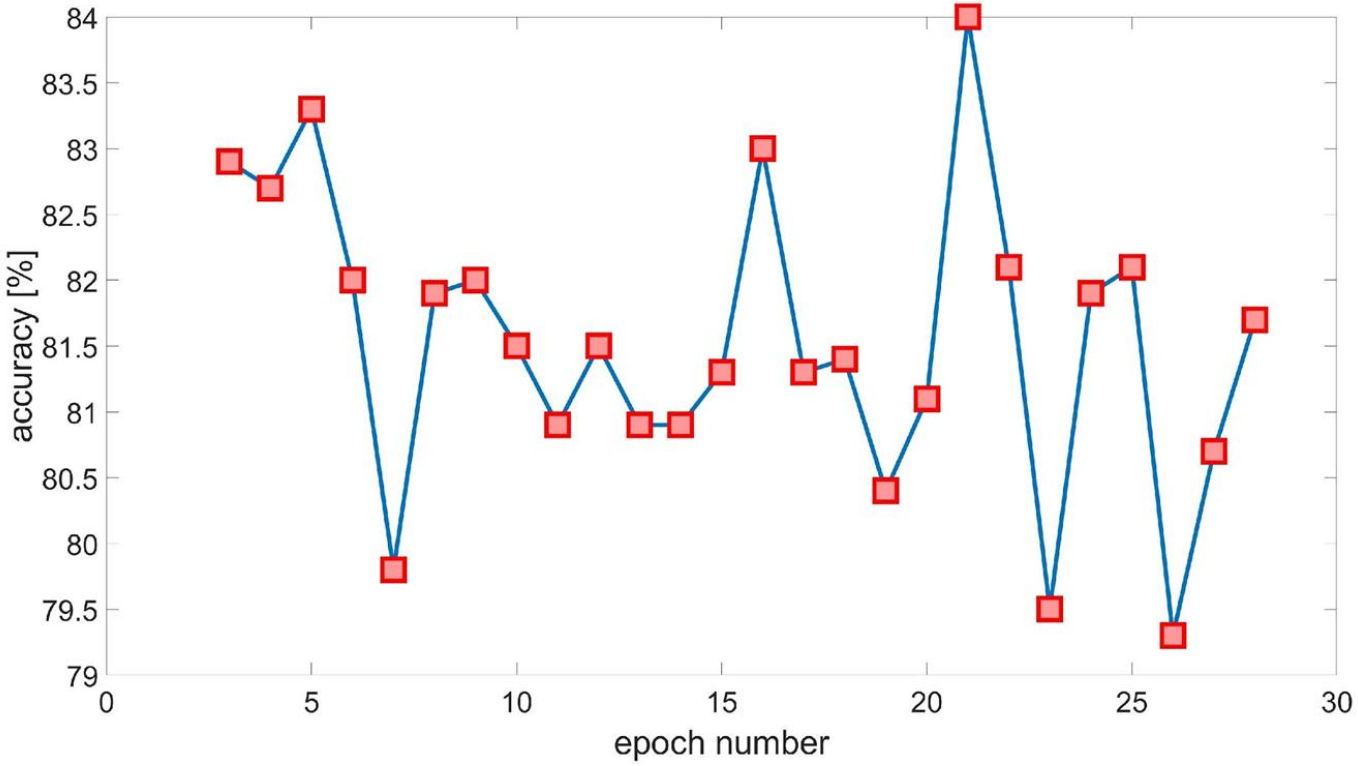
C: membrane capacitance
u_i: membrane potential
g_L: leak conductance
$\sigma\xi(t)$: white noise term

Input spike analysis

Does combining the optimal values of mnist threshold and spike rate produce highest accuracy ?

Ans) No!

Result for mnist threshold 0, spike rate: 220Hz (other parameters are also fixed at initial values)
Epoch 3~28



Analysis:

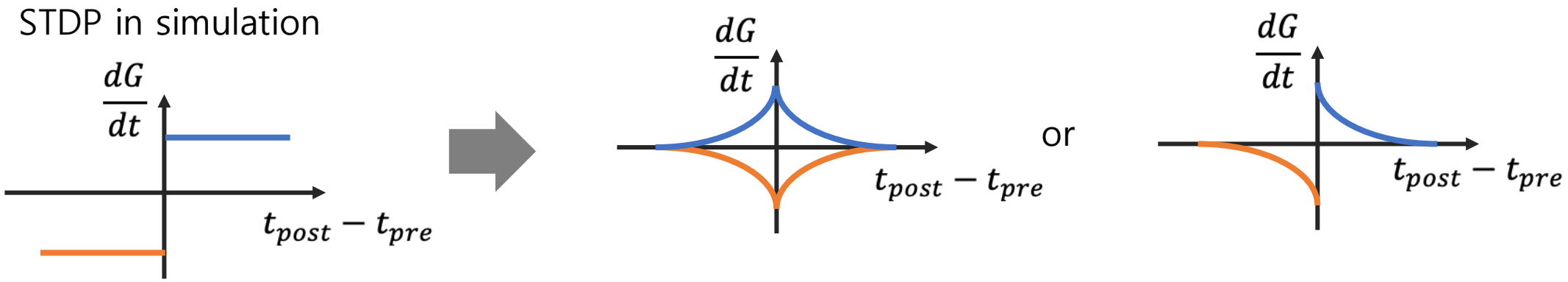
- Altho
  gener
  netwo
- High
  yield
  accur

overall

lel will
nt

Future plans

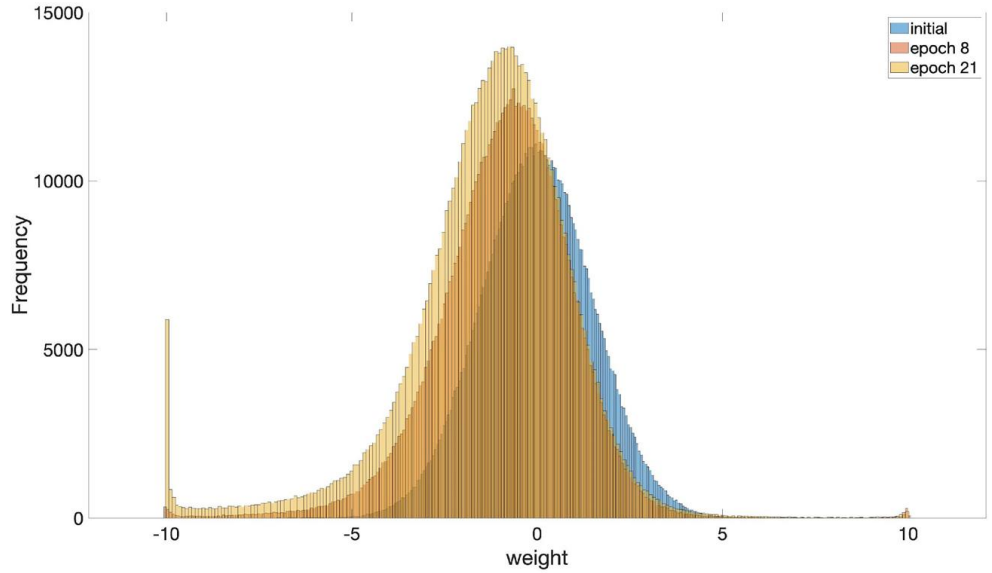
- Change STDP in stimulation so that weight change depends on $t_{post} - t_{pre}$



1. Find optimal(or suitable) STDP behavior for sRBM simulation
2. Follow method for designing STDP_WL pulse in S. Kim *et al*. (2015)
3. Design STDP_WL_Gp and STDP_WL_Gm pulses

Future plans

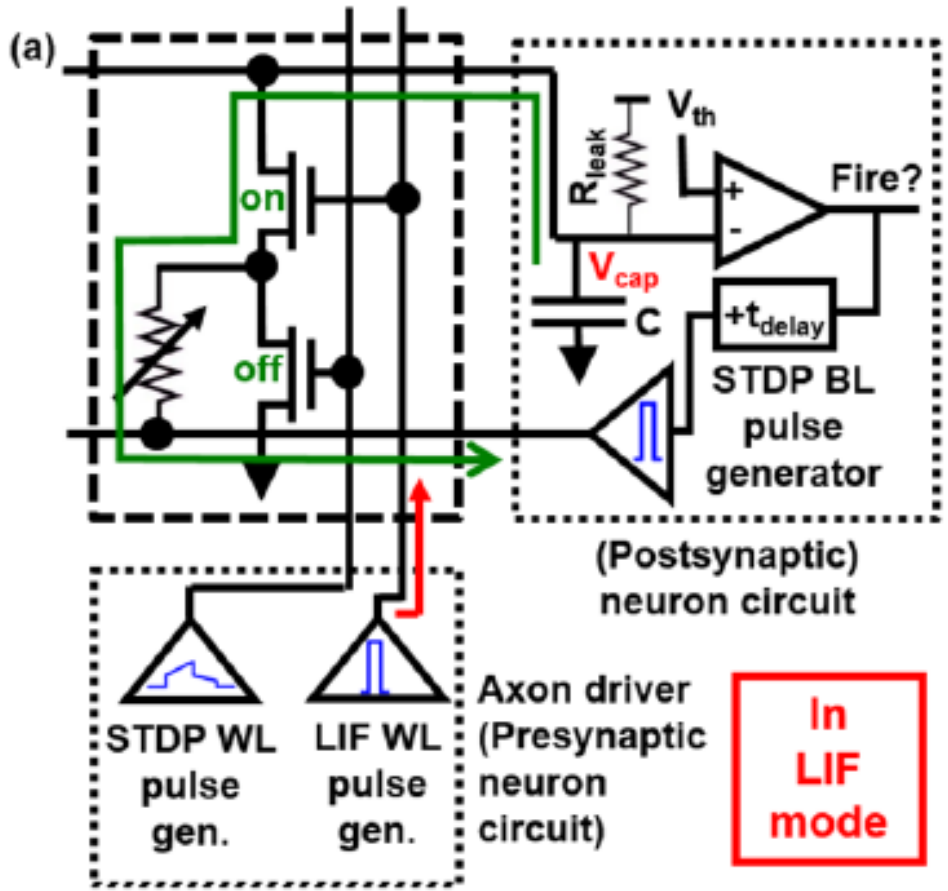
- Investigate membrane potential behavior for different spike rates
- Investigate weight difference before and after training

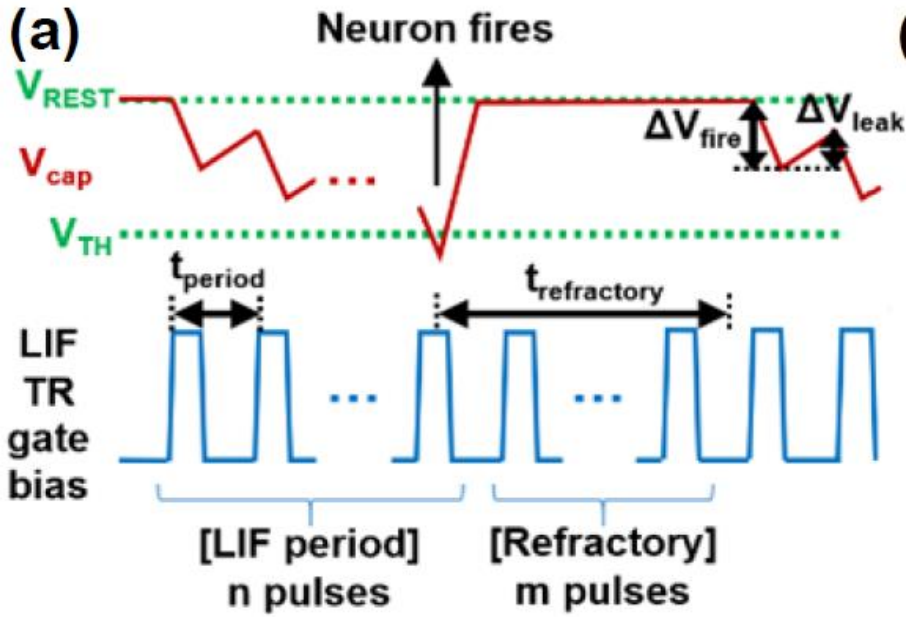


- Fully understand contrastive divergence(CD) and event-driven CD (Unbiased CD algorithm?)
- Rewrite current simulation program in python language using powerful tools such as tensorflow to reduce time required to execute 1 epoch
  -> Utilize GPU to run simulation (currently only using CPU)

Thank you

negative feedback ? (op-amp)





Neuron's membrane potential

$$C\frac{d}{dt}u_i = -g_L u_i + I_i(t) + \sigma\xi(t)$$