

Python Implementation of SNN

Hebbian Learning/ Spike-Timing-Dependent Plasticity (STDP)

- Spiking: “neurons that fire together wire together”
 - Pre-synaptic neuron (upstream)
 - Post-synaptic neuron (downstream)
 - Both connects by synapse [W_{ij}]
- Spiking version of Hebbian learning
 - Hebbian theory: increase in synaptic efficacy arises from a presynaptic cell's repeated and persistent stimulation of a postsynaptic cell. Explains synaptic plasticity, the adaptation of brain neurons during the learning process.
 - If i fires just before neuron j , then W_{ij} increases
 - If j fires just before neuron i , then W_{ij} decreases

ANN version of Hebbian Learning

- ANN approximation for the spiking rule. Synchronous version. At the last timing they agree, increase the weight
 - $W_{ij}[t+1] = W_{ij}[t] + \alpha X_i[t] X_j[t]$, $X_i, X_j \in \{-1, 1\}$ (range of activation functions)
 - $X_i, X_j \rightarrow$ output state of neurons i, j at time t
 - Those that fire together (happen to be in the same state together) get tighter connections to each other, those that happen to be in a opposite state get weaker connections
 - Using physiological spiking definition to motivate ANN version that mimics the hebbian learning
 - Gets pattern recognition - type of clustering. Labelled patterns with clusters. Recognize that certain things are always associated with each other and those associations get labelled internally to the neural network
 - Output of the neural network is an unsupervised process

Neuromorphic Computing

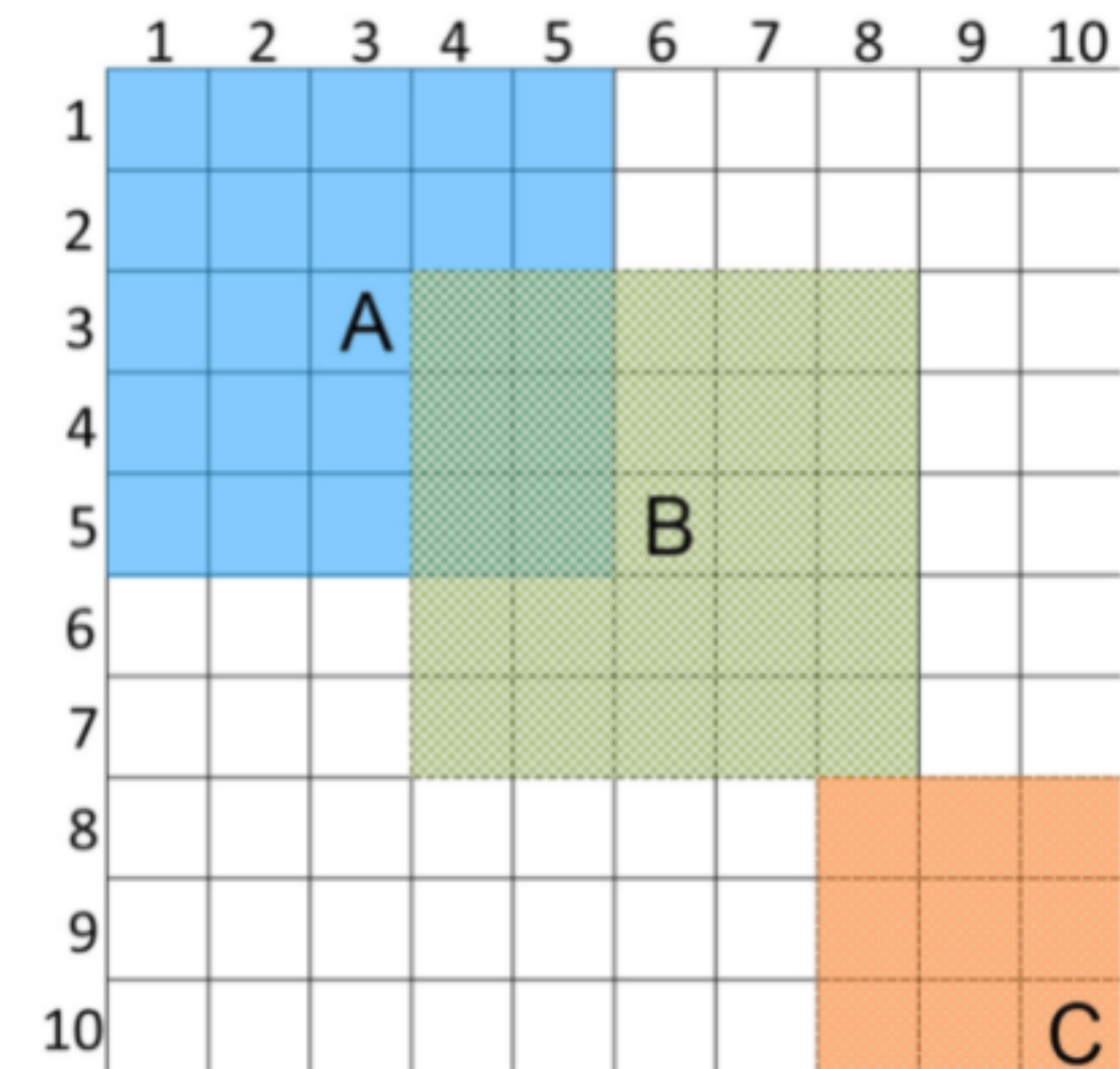
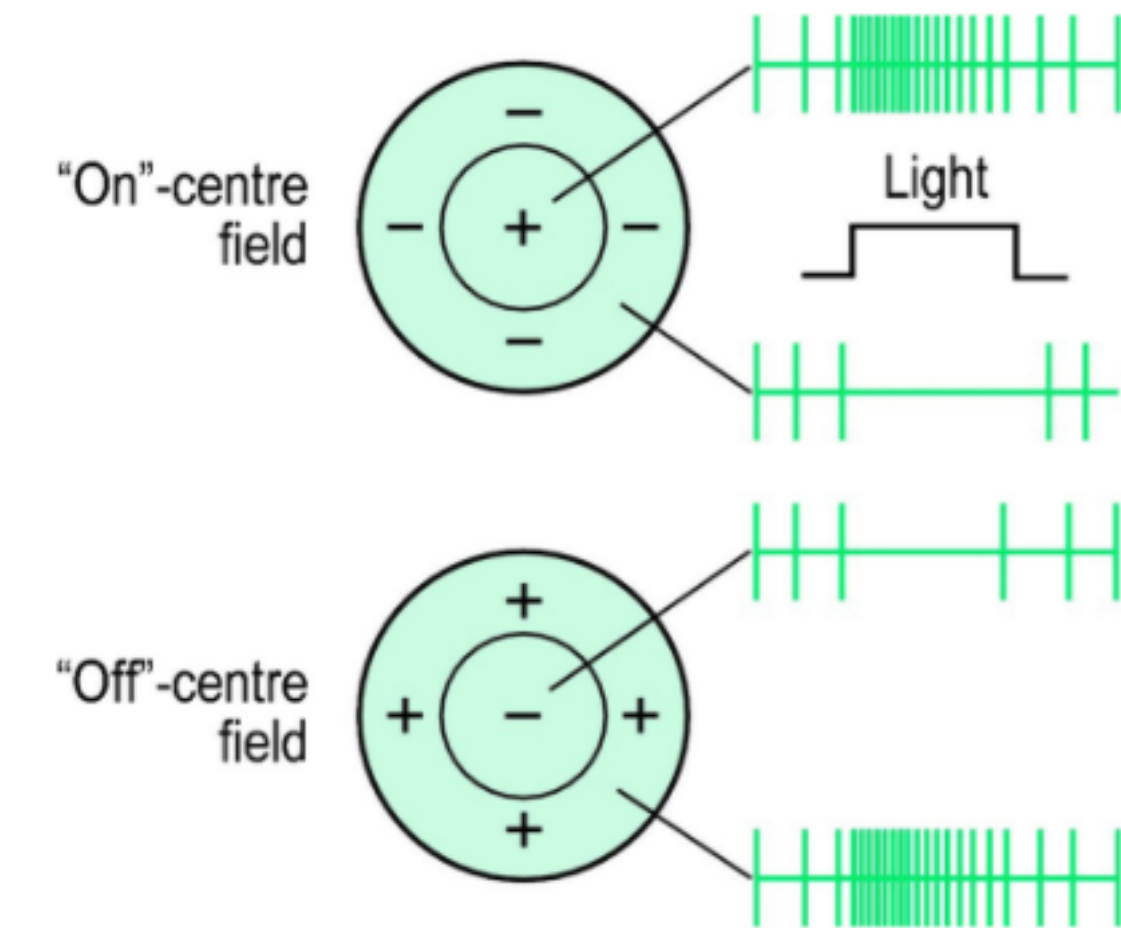
- ANN -> Boolean activation functions [classifier-type problems]
- Then continuous activation functions [regression-type problems]
- Focused on the network and not the “device”
- Properties of the neuron were ignored
- SpiNNaker -> hardware implementation of spiking neural networks
 - Low Size, Weight, and Power
 - IBM TrueNorth
 - Intel Loihi
 - About 1000 neurons. small, fast, very low power
- Backpropagation now exists for these SNN architectures

SNN Simulator for Classification


- After we learned the optimal weights of the network using the STDP algorithm, it uses the weights to classify the input patterns into different classes
- Uses 'winner-takes-all' strategy to suppress the non firing neurons and produce distinguishable results
- Steps:
 - 1. For each input neuron membrane potential is calculated in its **receptive field** (5x5 window)
 - 2. Spike train is generated for each input neuron with spike frequency proportional to the membrane potential
 - 3. For each image, at each time step, potential of the neuron is updated according to the input spike and the weights associated.
 - 4. First firing output neuron performs lateral inhibition on the rest of the output neurons.
 - 5. Simulator checks for output spike.

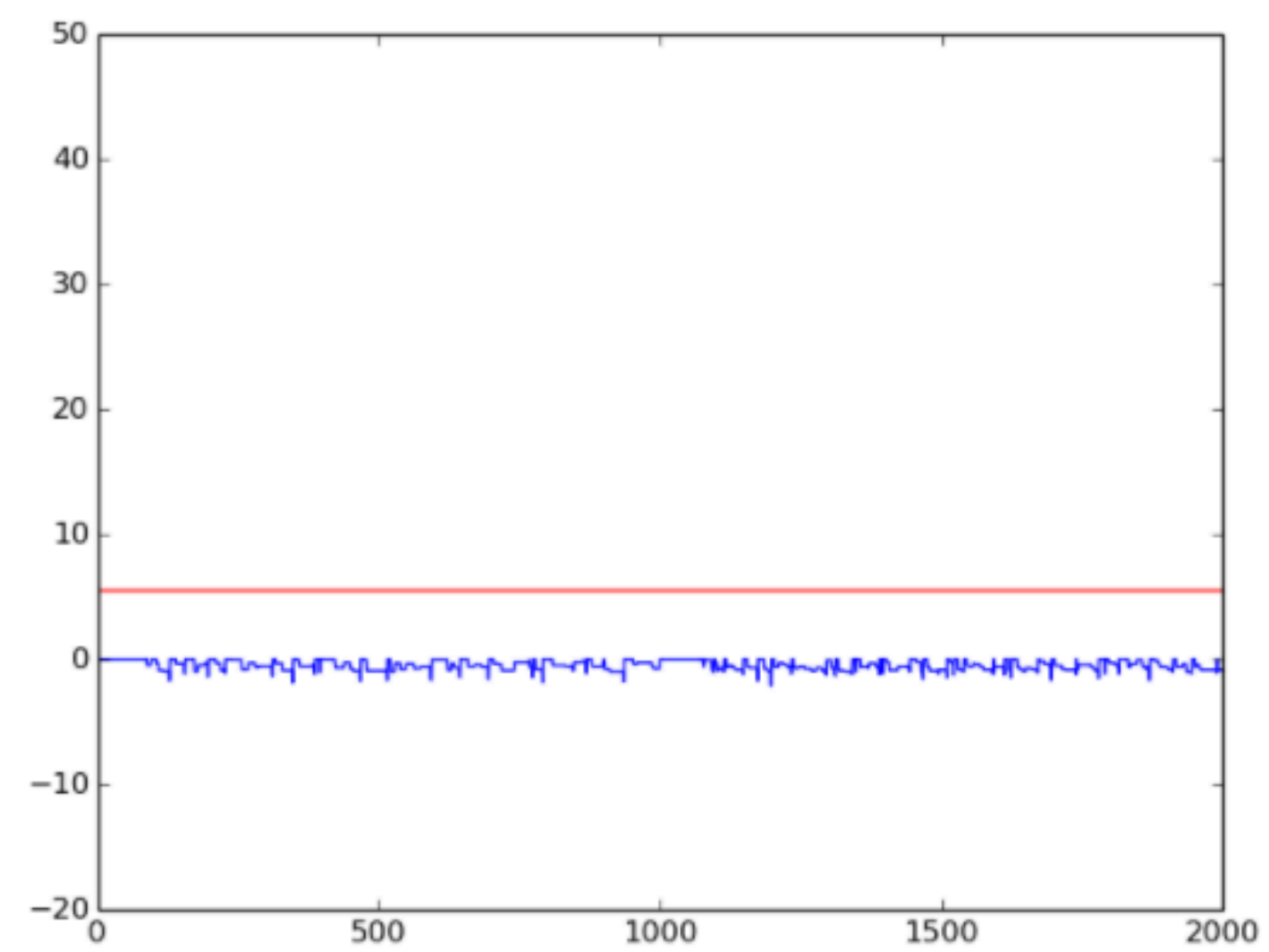
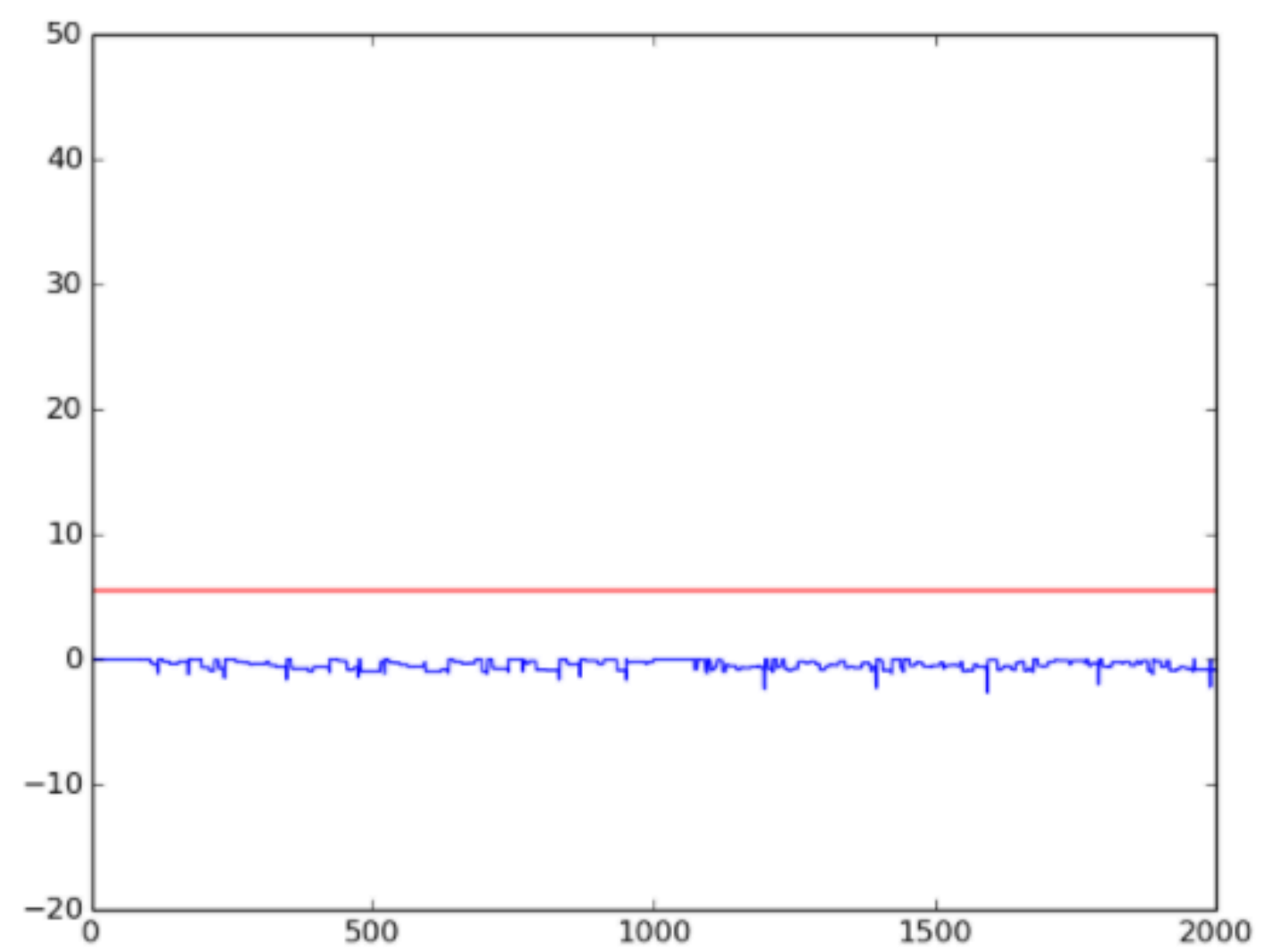
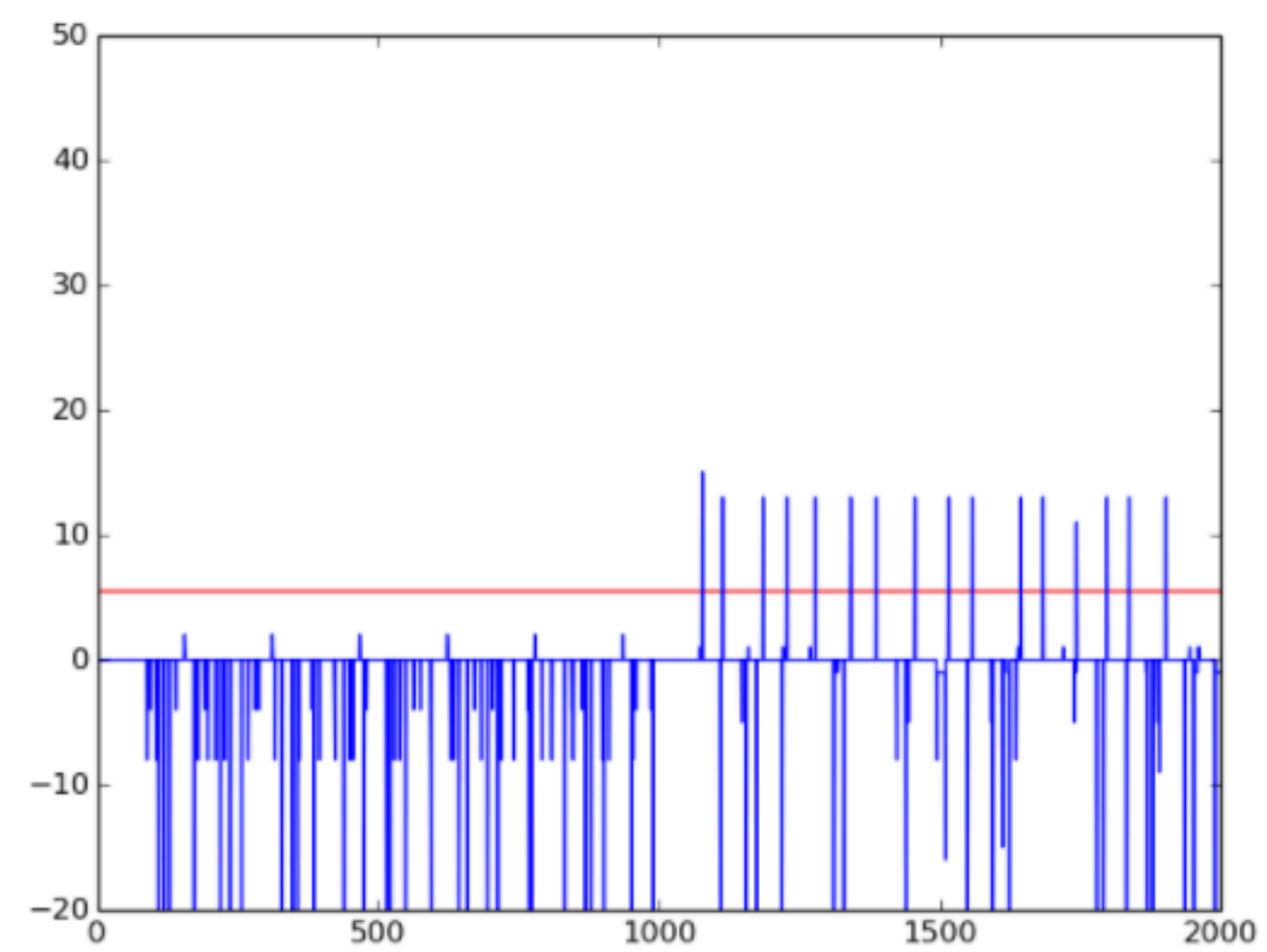
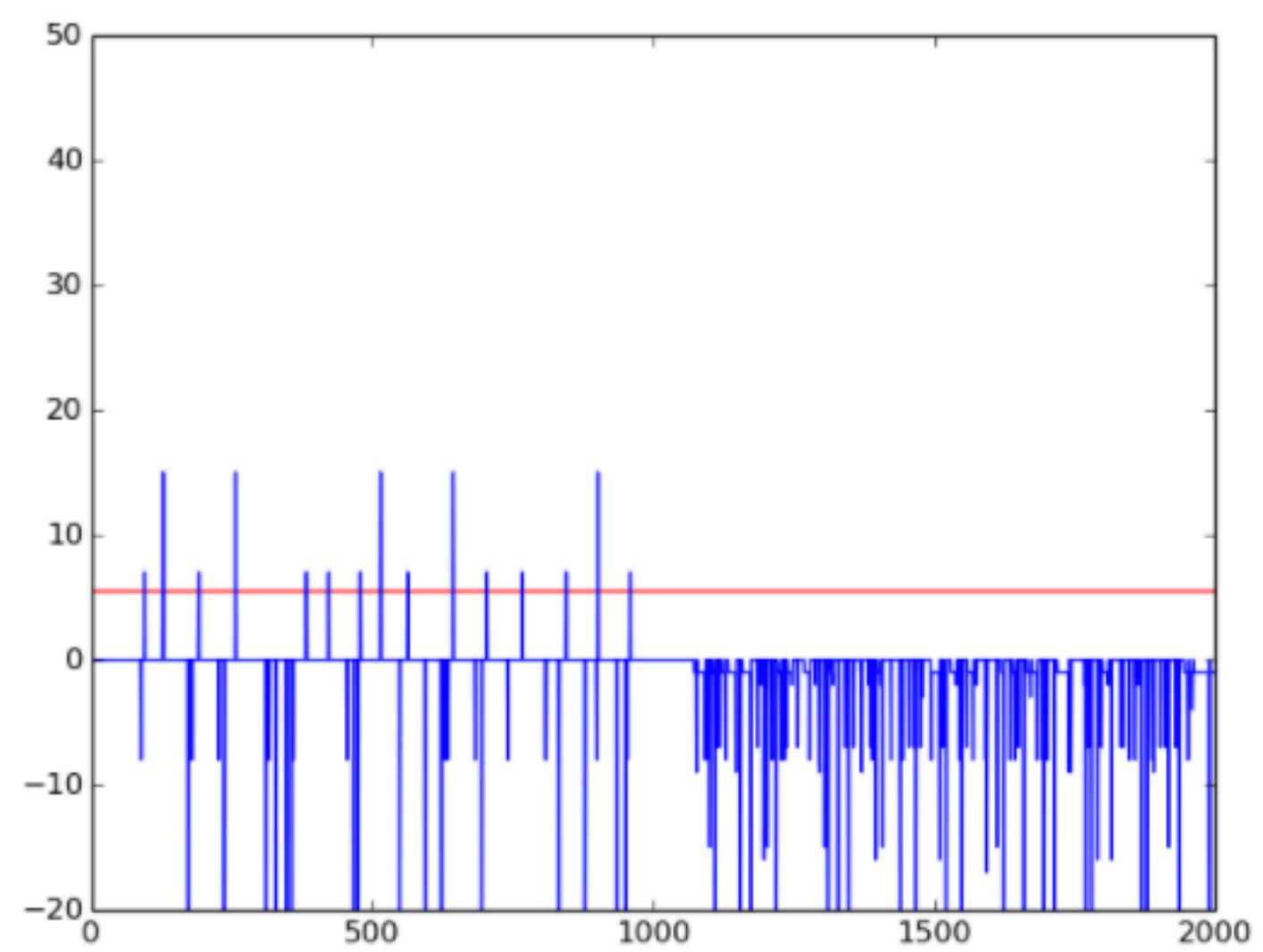
Receptive Field

- Receptive field is an **area** in which **stimulation** leads to **response** of a particular sensory neuron
- In the case of an SNN, where the input is an image, receptive field of a sensory neuron is the part of the image which **increases its membrane potential**
- On-centered receptive field is used
- To get the on centered receptive field, a sliding window is used whose cells are weighted according to the Manhattan Distance from the center of the window
 - $(|x_i - x_c| + |y_i - y_c|)$
- The fields for different neurons are overlapping.



Results

- Tested upon binary classification
- Extended upto any number of classes. The images for two classes: 
- Each of the classes were presented to the network for 1000 time units each. The activity of the neurons was recorded. Here are the graphs of the potential of output neurons versus time unit.
- First 1000 time unit corresponds to class1, next 1000 to class2. Red line indicates the threshold potential.



Results

- The 1st output neuron is active for class1, 2nd is active for class2, and 3rd and 4th are mute for both the classes. Hence, by recording the total spikes in output neurons, we can determine the class to which the pattern belongs.

