COMP 596: Programming Assignment 1

Hong, Joon Hwan
260832806
Winter 2021

Part 1 – Understanding Hopfield Networks

Question 1.1

1. **Processing units/neurons:**
    a. $o = [o_1, \dots, o_n]^T$
2. **State of activations at time *t* for each unit:**
    a. $a(\mathrm{t}) = [a_1(\mathrm{t}), \ \dots \ , a_n(\mathrm{t})]^T$
3. **Pattern of activity defined by weight matrices:**
    a. A connection between unit *i* and unit *j* has weight W_{ij} and the weights are symmetric: $(w_{ij} = w_{ji})$. This results in the general weight matrix W such that it represents the pattern of connectivity between all units/neurons. Let *n* be number of neurons.
    b. $W = \begin{bmatrix} 0 & \cdots & W_{1n} \\ \vdots & \ddots & \vdots \\ W_{n1} & \cdots & 0 \end{bmatrix}$
4. **Propagation rule:**
    a. Given in lecture slides. It is the total input that neuron *i* receives at timestep t:
    $$net_i(t) = \sum_{j \neq i} \mathrm{w}_{ji} o_j(t-1)$$
5. **Output function:**
    a. Each neuron has an output function of the following where theta is the threshold:
    $$o_i(t) = \begin{cases} 1 \ \ if \ net_i(t) - \theta_i > 0 \\ \quad -1 \ \ otherwise \end{cases}$$
6. **Activation:**
    a. The activation function for each neuron is defined as the following:
    $$a_i(t) = \sum_{\mathrm{j} \neq i} w_{ij} o_j \ - \theta_i = net_i(t) - \theta_i$$
    If the total weight connected to the neuron is greater than a set threshold, the neuron and outputs 1 and else -1 according to the output function using the activation function.
7. **Learning:**
    a. The learning rule given in the lecture was the following which is a Hebbian learning rule:
    $$\Delta W_{ij} = -\eta \frac{\partial E\big(o(t)\big)}{\partial W_{ij}} = \eta o_i(t) o_j(t)$$
    And the threshold for each unit changes accordingly: $\Delta\theta_i = -\eta o_i(t)$
8. **Environment:**
    a. The MNIST images as the series of inputs act as the environment

Question 1.2

1. It is called an energy function because when the network units are being updated, the "energy" is a monotonically decreasing function. It either decreases of stays the same value (moving towards lower energy configurations).
2. The energy function of a traditional Hopfield network given in the lecture is the following:

$$E\big(o(t)\big) = \sum_{i<j} W_{ij} \mathrm{o}_{\mathrm{i}}(t)\, \mathrm{o}_{\mathrm{j}}(t) + \sum_{\mathrm{i}} \theta_i \mathrm{o}_{\mathrm{i}}(t)$$

Question 1.3

$$-\eta \frac{\partial E\big(o(t)\big)}{\partial W_{ij}} = \eta o_i(t) o_j(t) \Longrightarrow \frac{\partial E\big(o(t)\big)}{\partial W_{ij}} = -o_i(t) o_j(t)$$

Question 1.4

1. **What synaptic plasticity rule is being implemented:** the type of synaptic plasticity rule being implemented would be the Hebbian learning rule in the Hopfield network. The idea is that simultaneous activation of neurons leads to increases in synaptic strength between those neurons (i.e. "fire together wire together").
2. **What makes it this type of rule:** supposing a weight W_{ij} and a given pattern of μ, if the corresponding neurons *i* and *j* are equal in pattern μ, then the product of $o_i^{\mu} o_j^{\mu}$ will be positive. This would result in a positive effect on the weight W_{ij}. (in other words, the values of neurons *i* and *j* converge if the weight between them is positive, and diverge/repel if the weight is negative)

Question 1.5

1. Let *n* be number of patterns. Let η be 1. Let V^{μ} be a state (an array/vector). To train a Hopfield network on multiple patterns, the mathematical equation would be the summation:

$$W_{ij} = \sum_{\mu=1}^{n} \mathrm{V}_i^{\mu}\, \mathrm{V}_{\mathrm{j}}^{\mu}$$

Question 1.6

1. The learning rules result in changes to values of the weight matrix, in turn affecting the energy (a scalar value associated with each state of the network). The learning rules in the end lowers the energy of the states that the network should "remember" (the patterns/memories, which are the selected MNIST images in this case).
   a. This enables the network to be a content addressable memory system.
   b. This allows the recovery of patterns/memories based on similarity, as energy is a monotone decreasing function.

Question 1.7

1. I predict that when a complement of a stored pattern is presented, it will remain as it is. Given that a state X exists as a stored memory state, then so is -X as a ghost state/memory, the inverse of the pattern will exist in the network. (for each stored pattern, the complement pattern is also an attractor state; I predict that the Hopfield network is "sign blind")
   a. Extending this idea, I would also postulate that given a high enough level of noise, and tasked to recover/update the network, the network will converge to the inverse/ghost state

due to too many pixels (in the MNIST case) being inverted to favour the inverted pattern than the original memory.

Part 2 – Designing your Hopfield Network

Question 2.1

1. For the Hopfield network class, I declared ***self.state*** to keep the current state of the network, ***self.W*** as the weight matrix of the network at the given timestep, and ***self.theta*** as an array of threshold values that can change as new memories are added.
    a. These are all data structures that are necessary to be stored ***per*** Hopfield network, and thus it made more sense to internally keep stored rather than utilising global variables in the notebook.

Question 2.2

1. To calculate the change in weight from a one-pattern-at-a-time learning rule, the ***self.W*** matrix at position (ij) were incremented (to allow for multiple usage of the one pattern learning rule) by the equation given in the lecture slides: $\eta o_i(t) o_j(t)$. The threshold was changed by: $-\eta o_i(t)$
2. The iteration through *i* = (0, 784) and *j* = (i, 784) was accomplished by nested for-loops. The *j* loop began at *i* as the weight matrix is symmetrical and the (ji) weight value could be assigned as well.

Question 2.3

1. Emulating the equation from Question 1.5, the summation of different states was accomplished by iterating through each image and summating the learning rule per image.
2. To set the synaptic weights and thresholds, the code determines the change in both that will be caused per image (just like the one-pattern-at-a-time learning rule) then summates them to the ***self.W*** and ***self.theta*** variables.