1 Task 1a:

From the lecture notes we know that:

$$\begin{cases} P_{Error} = prob(C_i^{(\mu)} > 1) \\ C_i^{(\mu)} = \sum_{j=1}^{N} \sum_{\nu=1}^{P} \zeta_i^\nu \zeta_j^\nu \zeta_i^\mu \zeta_j^\mu \end{cases}$$

We only know that P>>1 and N>>1 and we have to take into account than P might not be much smaller than N. Moreover, we can't assume that $w_{ii} = 0$. Thus, we can't ignore the "small correction" that was removed in the lecture notes. This is because it becomes significantly large when P>N. First off, we start by extracting the "small correction" part: $C_i^{(\mu)}$:

$C_i^{(\mu)} = -\frac{1}{N} \sum_{\nu=1}^{P} \zeta_i^\nu \zeta_i^\nu \zeta_i^\mu \zeta_i^\mu - \frac{1}{N} \sum_{j=1, j\neq i}^{N} \sum_{\nu=1, \nu\neq\mu}^{P} \zeta_i^\nu \zeta_j^\nu \zeta_i^\mu \zeta_j^\mu$
$= -\frac{P}{N} - \frac{1}{N} \sum_{j=1, j\neq i}^{N} \sum_{\nu=1, \nu\neq\mu}^{P} \zeta_i^\nu \zeta_j^\nu \zeta_i^\mu \zeta_j^\mu$

Since the patterns are random, $C_i^{(\mu)}$ is Gaussian distributed according to the central limit theorem. The mean isn't 0 as in the lecture notes, instead it has shifted by the constant $-\frac{P}{N}$. Further, the variance of the distribution is $\frac{P}{N}$. We can now calculate P_{Error} with the cumulative distribution function of normal distribution:
$P_{Error} = P(C_i^{(\mu)} > 1) = F(X > 1) = 1 - F(X \leq 1) = 1 - \frac{1}{2}(1 + erf(\frac{X-\mu}{\sigma\sqrt{2}})) = \frac{1}{2} - \frac{1}{2} erf(\frac{\sqrt{N}}{\sqrt{2P}} + \frac{\sqrt{P}}{\sqrt{2N}})$

2 Task 1b:

We start by setting our program to have 200 neurons N and creating an array containing all the values for p that we are going to test for. The program generates p random patterns and calculates the weight matrix W with Hebb's rule. $W = \frac{1}{N}\sum_{\mu=1}^{P}\vec{\zeta}^{\mu}\vec{\zeta}^{\mu\,t}$. Where $\vec{\zeta}^{\mu}$ is the vector containing the bits for pattern μ.

Then, we update the network synchronously and calculate the P_{Error} by checking 10^5 bits for each p.

In the plot below we can clearly see the correlation between our actual error and the theoretical model. The theoretical estimate seems to fit almost perfect to the real data. From the lecture notes we are used to the fact that $\frac{P}{N}$ is proportional to P_{Error}, but in our plot P_{Error} starts to decrease when $\frac{P}{N}$ becomes greater than 1. This is due to the fact that P isn't much smaller than N nor is $w_{ii} \neq 0$. Therefore, as stated in 1a, we can't neglect the small correction.

In our case, the diagonal elements(w_{ii}) in the weight matrix gets the value $\frac{P}{N}$. This means that a neuron will depend more and more on itself when p increase. In other words, a neuron is more likely to keep it's initial value after one step when we have more patterns in relation to the number of neurons.

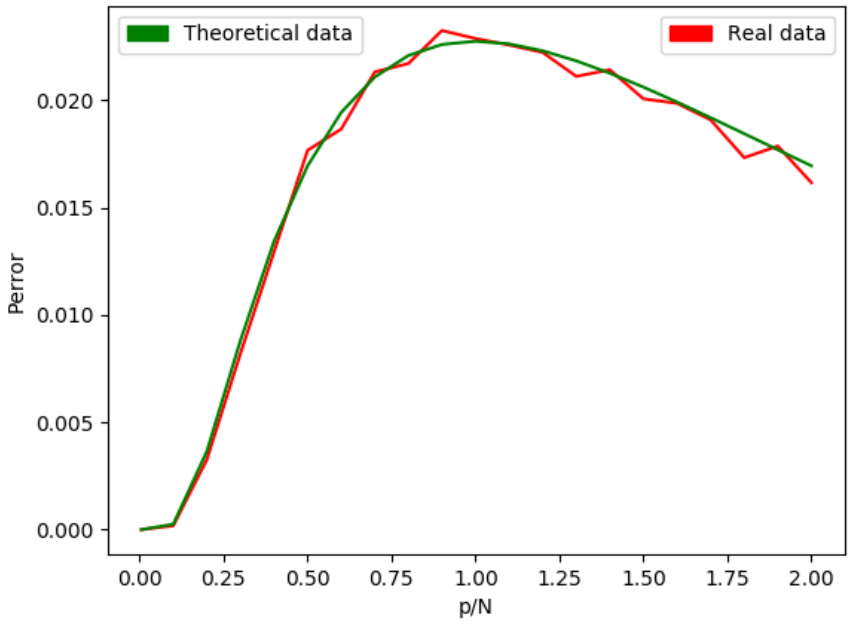

Figure 1: P_{Error} as a function of $\frac{P}{N}$

3 Task 2a:

The program works quite similar to the one in task 1. However, we now have a asynchronous and stochastic Hopfield network, meaning that one update corresponds to picking one neuron at random and calculate it's state. We have also introduced a noise parameter β, which is set to 2.

The plot below shows the traveling mean of the order parameter, $< m(T) >$, for pattern one in 20 different experiments over time T. One curve represent one experiment and the traveling mean is calculated as follows:
$< m_1(T) >= \frac{1}{T} \sum_{t=1}^{T} m_1(t)$

We choose T to be equivalent to 100 updates of the network. Moreover, the different experiments seems to find a steady state around 0.94, which sort of means that the network has approximately 94% of it's bits matching input pattern 1.
All runs of the experiment seems to reach the steady state after about 250-500T, or in other words after 25000-50000 updates. From the plot, we can also see that there are one experiment who seems to deviate from the steady state. This is probably due to the noise parameter, which can make the network "jump" to a different steady state. These steady states can be another pattern or a spurious state.

Besides this occasional deviation the network preforms well, it keeps almost all the bits the same as the input pattern. According to the phase diagram our experiment should lie in the area where $m \neq 0$. We can check whether this is true or not by calculating $\beta^{-1} = 0.5$ and $\alpha = \frac{5}{200} = 0.025$. This point is clearly in the $m \neq 0$ area. When N is finite the small terms$(m_\mu, \mu \neq 1)$ can build up and affect $< b_i >$

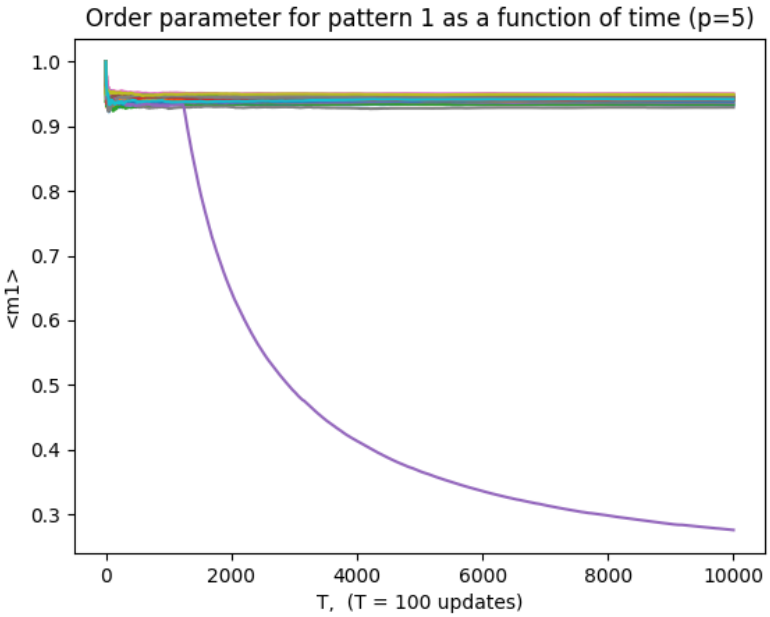

Figure 2: $< m_1 >$ as a function of time. One time step equals 100 updates and in every time step T the traveling mean $< m_1 >$ is calculated

4 Task 2b:

After changing the number of patterns from 5 to 40 we received the plot below. We can see that there is a significant difference in the values of m_1 now and those received in 2a. The order parameter is much closer to unity when p was equal to 5. After just a few iterations the order parameter reaches a steady state somewhere when m_1 becomes ≤ 0.5. This is because $\frac{p}{N}$ becomes to large and $< b_i >$ no longer solely depend on $m_1\zeta_1$. This leads to avalanches of errors and therefore we get these values of m_1. To put this into the context of the phase diagram we calculate $\alpha = \frac{40}{200} = 0.2$. Our α is greater than the $\alpha_c = 0.138$, which is stated in the lecture notes. This further explains why m_1 is closer to zero.

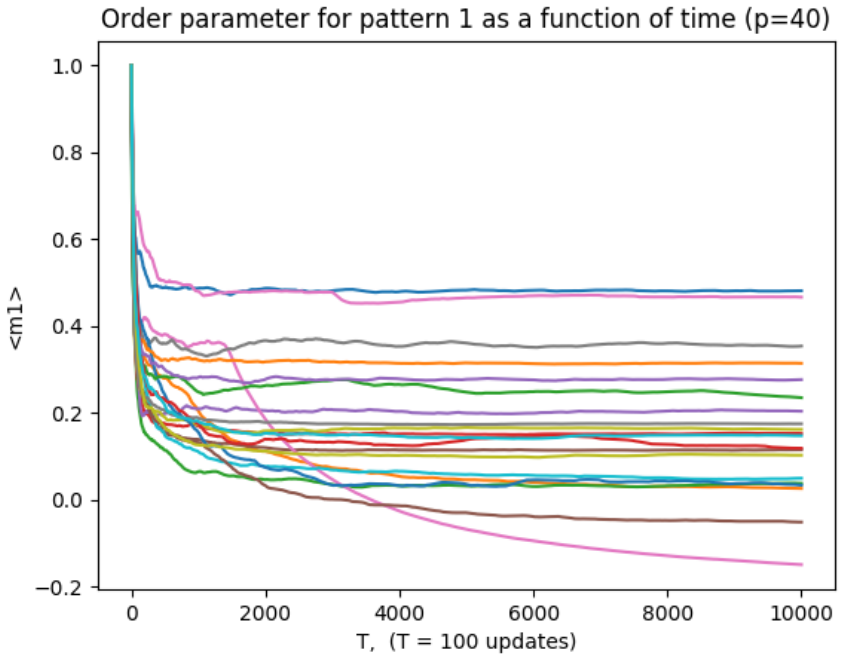

Figure 3: $< m_1 >$ as a function of time. One time step equals 100 updates and in every time step T the traveling mean $< m_1 >$ is calculated

5 Task 3a:

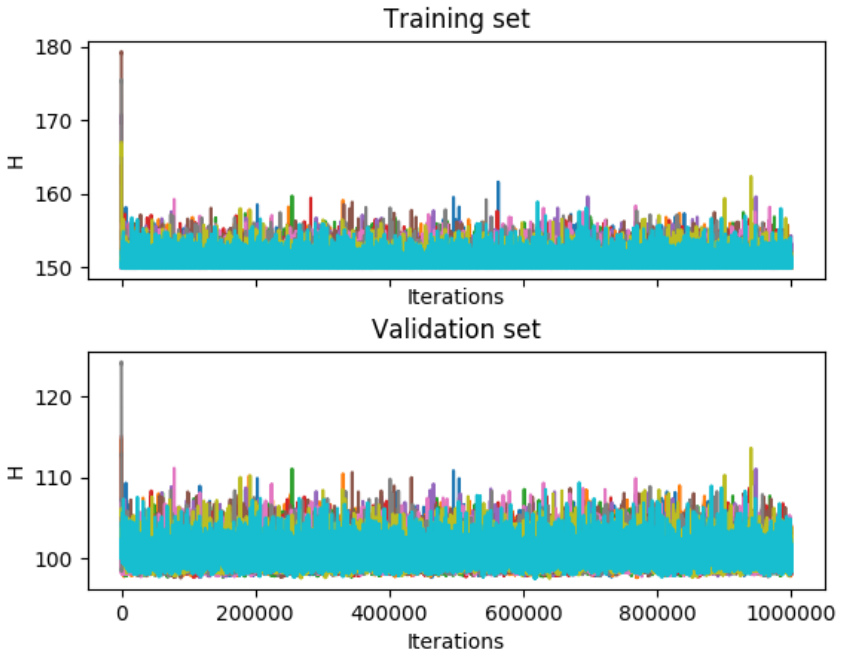

Figure 4: Energy for the training set as well as for the validation set, both as a function of the training time(iterations)

In the first few training iterations, the network learns to classify both sets a bit better and then it seems to find a minimum. The initial decrease of energy is a result of the, none optimal, initial weights and threshold changing to somewhat better values.

According to the gradient-descent rule, the energy should either decrease or remain the same over time. Since the energy fails to decrease any more, even after one million training iterations, we suspect that the problem isn't linearly separable by a single-layer threshold unit. To figure out what problem we we're tackling we plotted the inputs and differed the points with different color, depending on it's target output. The result was that input with target output +1 was position in the center and was surrounded by the patterns with target output -1. It's now clear that the problem isn't linearly separable.

Classification errors
Training:
Average: **0.50567** Minimum: **0.41333** Variance: **0.00324**

Validation:
Average: **0.4935** Minimum: **0.345** Variance: **0.00435**

6 Task 3b:

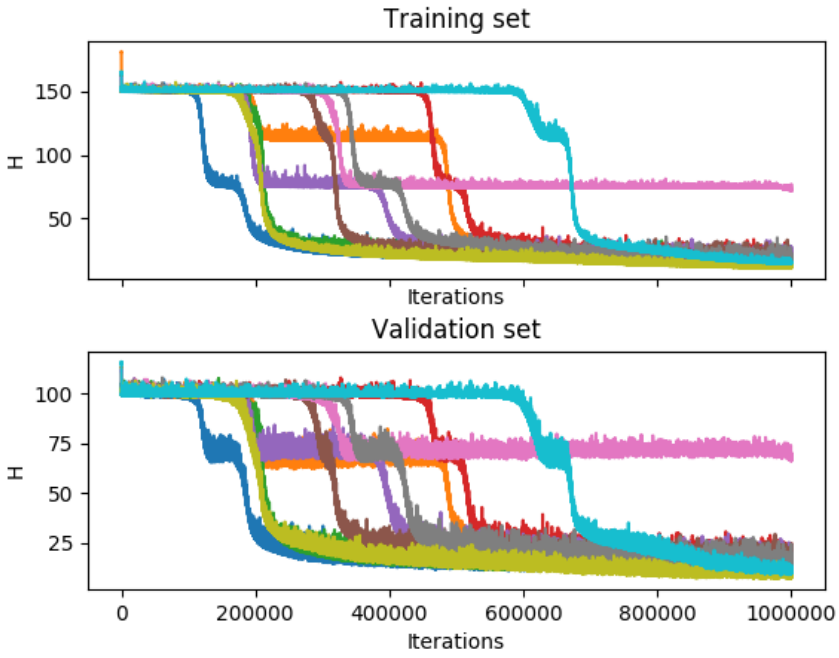

Figure 5: Energy for the training set as well as for the validation set, both as a function of the training time(iterations)

As we can see from the plot above, adding a hidden layer has drastically improved our network's ability to classify the input correct. Unlike 3a, almost all the experiments has find optimal weights and thresholds at the end of training. The reason for why a multilayer perceptron was needed in this problem can be understood by looking at a graphical representation. As mentioned in 3a the patterns with target output +1 is surrounded by the -1 patterns. More specific, the border between them seems to form a rhombus. Therefore, having 4 neurons in the hidden layer is exactly enough, since all four sides can be separated by a plane.

Classification errors
Training:
Average: **0.0443** Minimum: **0.0167** Variance: **0.00158**

Validation:
Average: **0.061** Minimum: **0.025** Variance: **0.00405**