Project 2: Helmholtz Networks

Blake Richards COMP-549

4 April 2022

Jonayed Islam (260930902)

Part 1

Question 1.1 (4 marks): In your own words (don't just quote the paper), what is the goal of a Helmholtz machine? In other words, what are we trying to learn with a Helmholtz machine?

The helmholtz network acts like a compression algorithm that tries to find the minimum cost to be able to represent an image from a dataset. Its goal is to internalise the representation of the inputs in its units. Essentially, it learns the distribution of the points in an image that is then used in recreating an approximation of that image. It is an unsupervised learning algorithm so there are no labels.

Question 1.2 (8 marks): In your own words, what is the description length cost of a representation in a Helmholtz machine? What does a low description length imply for the recognition and generative pathways in a Helmholtz machine?

Since the Helmholtz machine wants to reconstruct the input, the cost is essentially a measure of the amount of bits required in the internal representation to reconstruct this input.

To find the description length for one unit, the formula is:

$$C(s_j^\alpha) = -s_j^\alpha \log p_j^\alpha - (1 - s_j^\alpha)\log(1 - p_j^\alpha) \qquad (2)$$

.

This formula allows us to determine the number of bits required to represent the random binary output of a unit. The equation essentially adds the number of bits required to represent the states 0 and 1. This is why in the first half we take just s_j which represents one of the states and $(1-s_j)$ along with its associated probability represents the other.

To find the total cost, we need to sum all the description lengths for every unit and add the sum of the costs to describe the hidden weights given the input. The first part is essentially the cost to recognize the image and the second part is the cost to reconstruct it. The equation is the following:

$$\begin{aligned} C(\alpha,d) &= C(\alpha) + C(d\,|\,\alpha) \\ &= \sum_{\ell \in L} \sum_{j \in \ell} C(s_j^\alpha) + \sum_i C(s_i^d\,|\,\alpha) \qquad (3) \end{aligned}$$

If the cost is low, then the number of bits required to recognize the input and to reconstruct is low. Thus, the helmholtz machine has found an efficient way to store the input in its internal representation.

Question 1.3 (8 marks): What is the loss function that Helmholtz networks reduce? Explain in your own words what the different variables and terms are and what they mean for learning.

We want to minimise the number of bits required for all the possible ways of representing an input.

$$\begin{aligned} C(d) = &\sum_\alpha Q(\alpha|d) C(\alpha,d) \\ &- \left[-\sum_\alpha Q(\alpha\,|\,d) \log Q(\alpha\,|\,d) \right] \qquad (5) \end{aligned}$$

Equation 5 presents the cost of all the possible ways (ɑ) of representing the data (d). The first part of the equation presents the cost of presenting the data. Q(a|d) is a factorial distribution of the possible representations for the recognition weight. C(a,d) represents the cost of one of the representations. The second part that we subtract represents the entropy of the possible representations. We sum over all possible alpha so that we can get the cost across all possible representations.

$$P(\alpha \mid d) = \frac{\exp[-C(\alpha,d)]}{\sum_{\beta} \exp[-C(\beta,d)]} \qquad (6)$$

P(a|d) represents the probability of a representation given a dataset. Beta represents an alternative representation. This equation acts like the softmax equation where we take the cost of one representation and divide it by the sum of the costs of the others. P(a|d) is thus equal to

$$\begin{aligned} C(d) = {} & \sum_{\alpha} P(\alpha \mid d) C(\alpha,d) \\ & - \left(- \sum_{\alpha} P(\alpha \mid d) \log P(\alpha \mid d) \right) \\ & + \sum_{\alpha} Q(\alpha \mid d) \log \frac{Q(\alpha \mid d)}{P(\alpha \mid d)} \qquad (8) \end{aligned}$$

This equation essentially rewrites equation 5 into a different form that takes into account equation 6. The equation aims to measure the difference between the average cost of representations given a dataset and the cost with respect to the factorial distribution and entropy.

By minimising equation 8, we are essentially minimising the difference between the number of bits required to recognize the input and the number of bits required to recreate it.

Question 1.4 (4 marks): There are two different phases of training for a Helmholtz machine with different weight updates, a wake phase, and a sleep phase. What are the weight

updates for each phase? Define all the variables clearly and describe in your own words how to calculate them.

$$\Delta w_{kj} = \epsilon s_k^{\alpha}(s_j^{\alpha} - p_j^{\alpha}) \qquad (4)$$

Equation 4 is the equation used during the wake phase to update the generative weights. Eta symbolises the learning rate. Sk represents the state of the unit in the top layer before running feed forward. Sj represents the state of the unit in the bottom layer before running feed forward. Pj represents the probability of the unit in the bottom layer activating after running feed forward.

$$\Delta w_{jk} = \epsilon s_j^{\gamma}(s_k^{\gamma} - q_k^{\gamma}) \qquad (7)$$

Equation 7 is the equation used during the sleep phase to update the recognition weights. Sj and Sk symbolise the state of the bottom and top layer before running feed forward. Qk represents the probability of the units in the top layer activating after running feed forward.

Both equations essentially measure the difference between the generative and recognition phases.

Question 1.5 (8 marks): What components of the loss function do each training phase (wake and sleep) help to reduce? In your own words, how do they do this and why are they complementary?

As the recognition weights store the internal representation to recognize the input, the sleep phase update minimises the costs of recognition. The generative weights are used to reconstruct the input and their weight is updated during the wake phase so the wake phase reduces the cost to reconstruct the input.

The fit function of the model is dependent on the generative and recognition weights.Since we are trying to minimise the number of bits required to represent the input, we perform expectation maximisation on the generative and recognition weights. Basically, we run gradient descent on the fit function. The sleep phase handles the expectation step by optimising the recognition weights while the wake phase handles the maximisation step by optimising the generative weights.The two training phases are thus complementary as one represents the E-step and the other the M-step.

<u>Part 2</u>

Question 2.1 (4 marks): What variables do you need to calculate here to make your weight update? Why are these variables necessary for reducing the various components of the loss function?

In the wake phase, I store the activity of all the layers after they went through the sleep phase. After running feed forward (recognize function), I obtain the Pi from each of the layers. I can then subtract the activities from the top layer from the probability of the same layer. I then multiply that matrix by the activity of the top layer and then by the learning rate.

In the sleep phase, in order to update the recognition weights, I repeat the same steps as during the phase except for the fact that I subtract the activity and probability of the bottom layer and then multiply it by the activity of the top layer.

As described in equation 4 and 7, these variables help us understand the difference in representation between what the model is able to recognize versus what it is able to generate.

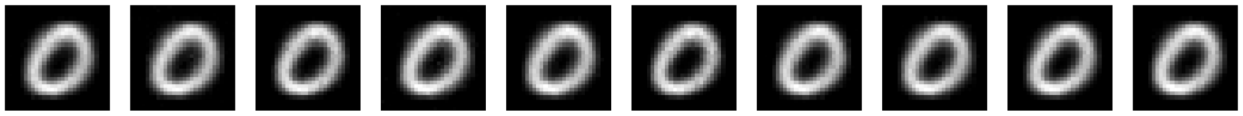
They thus help in reducing the loss function since the loss function is essentially a measure of the efficiency of recognizing and recreating the input.

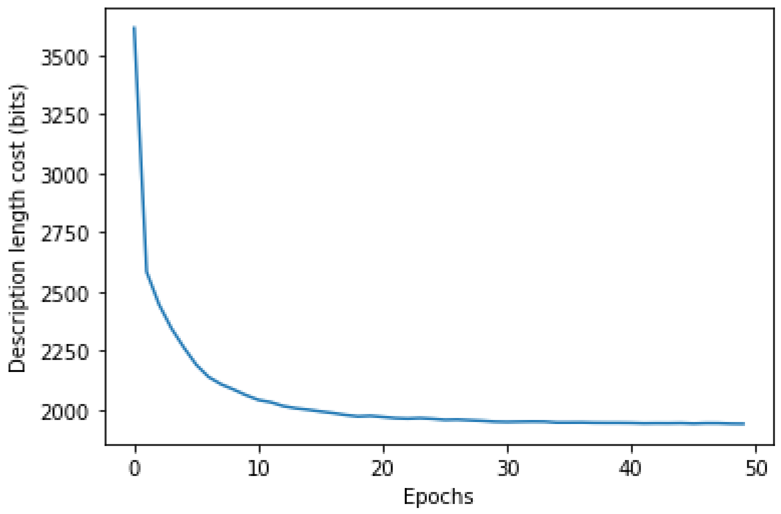
<u>Part 3</u>

Question 3.1 (5 marks): What happens when you train the network on a single class? What sorts of "dreams" does the network have and what do the weights show? What about if you train it on two classes? And what if you train it on all the classes? Given what you see, do you believe that the network is achieving its ultimate goals in training or not (consider your answer to Question 1.1)? What problems do you see?

Training on 1 class

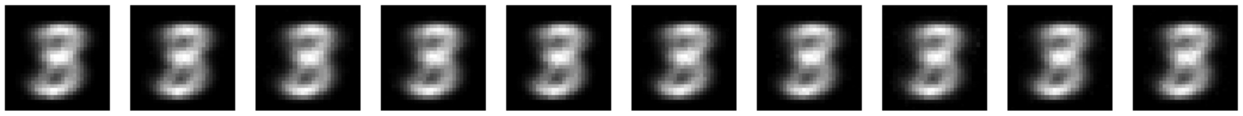
When training only on 0s, the helmholtz machine does a great job of storing the 0 in its units. It also achieves a cost of 1940 bits.

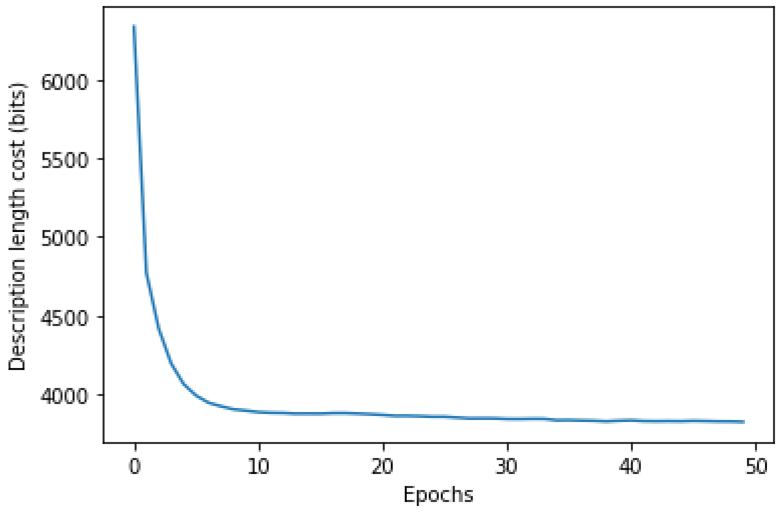




Training on 2 classes

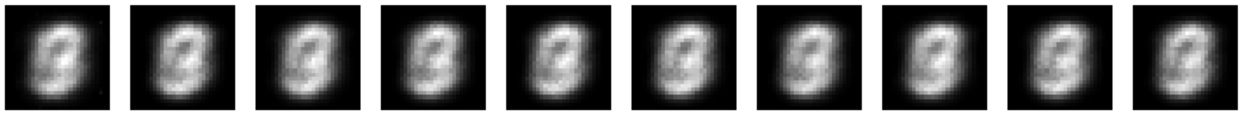
When training on 3s and 8s, the network is able to dream of something that looks like a hybrid of 3 and 8. It achieves a cost of around 3950. This is nearly double the cost for only one class.

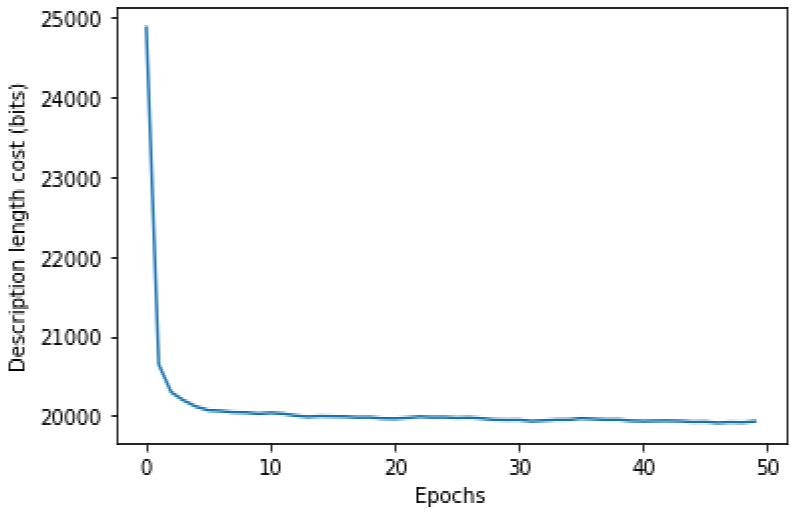




Training on 10 classes

The network understands the general location of where the numbers are in the frame, however it is not able to distinguish the different numbers. Here we see that the cost is 19930 bits or ten times that of one class

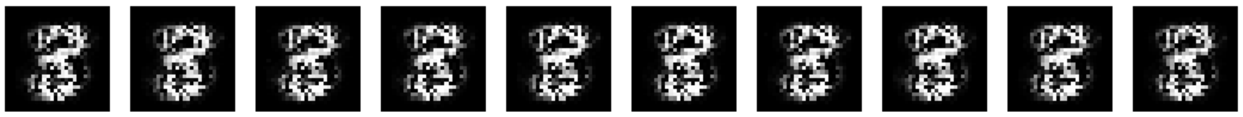




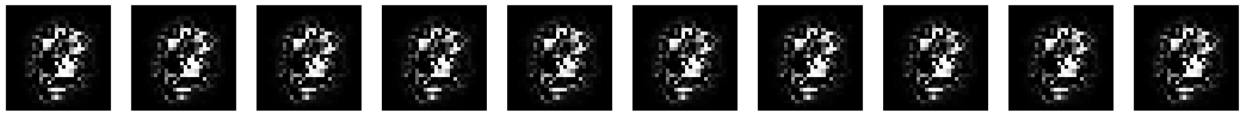
Overall, the helmholtz machine is able to efficiently represent one class and can somewhat handle 2 classes, however it fails at properly representing all of the classes. The distribution of weights is too different for the inputs when looking at 10 classes thus it is not able to internalise the different representations.

Question 3.2 (5 marks): What happens when you increase the number of hidden units by a lot (e.g. to 100)? Does this change what you said in Question 3.1 at all? Note that you may have to adjust the learning rate when you adjust the number of units.

When training with 100 hidden units, the cost for 2 classes increases to 5800 and the dreams still show a hybrid between 3 and 8.



When training with 100 hidden units, the cost for 10 classes decreases to 33000. However, when looking at the dreams, it is impossible to see which number is being represented.

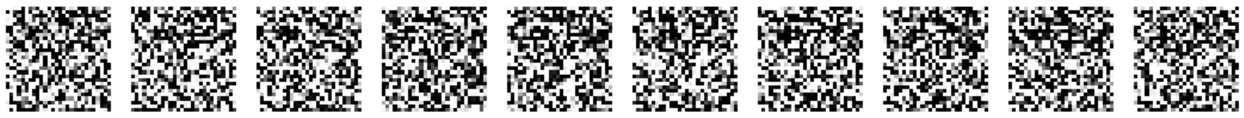


As such, this does not change my answer to the previous question as we obtain similar results.

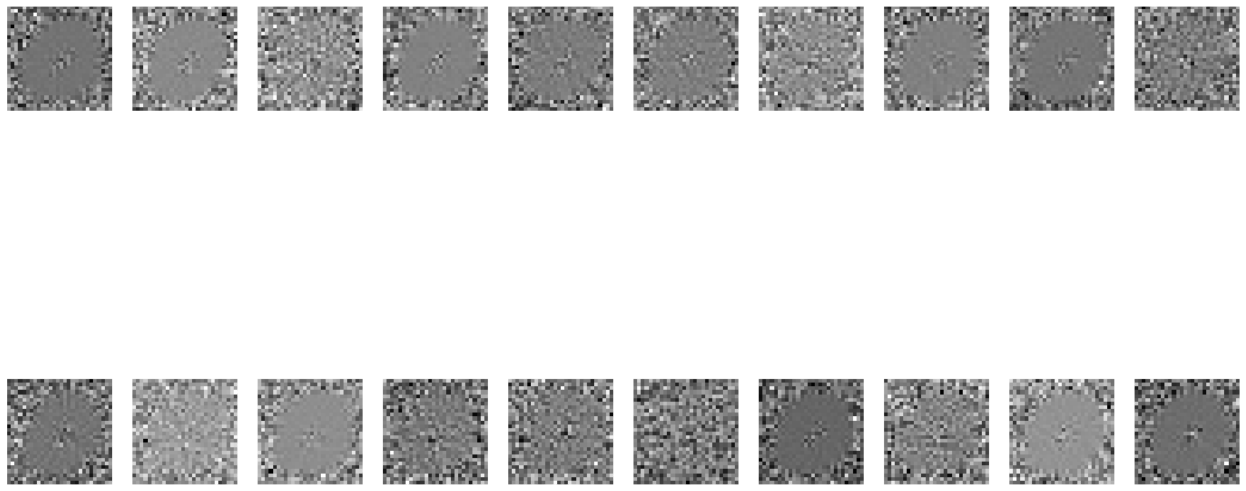
Question 3.3 (10 marks + 5 bonus marks): What happens when you turn off either the wake or sleep phase (one at a time, not both at once, which obliviously just eliminates any training)? How does it affect the dreams and the weights? Explain these results in your own words.

When turning off training in the wake phase, we see that the cost does not decrease. For example, the cost when trained on 0s is 18750 which is 9 times higher than the cost when training is on. The model is not able to generate any dream that resembles the input. The generative weights are not able to represent the input, but the recognition weights are able to internalise a representation of 0. I believe that the recognition weights are able to update since they rely on the approximation of what the model is able to generate. Also worth noting that when the top layer has an activity of 0, it does not have any effect on the weight update anyway.

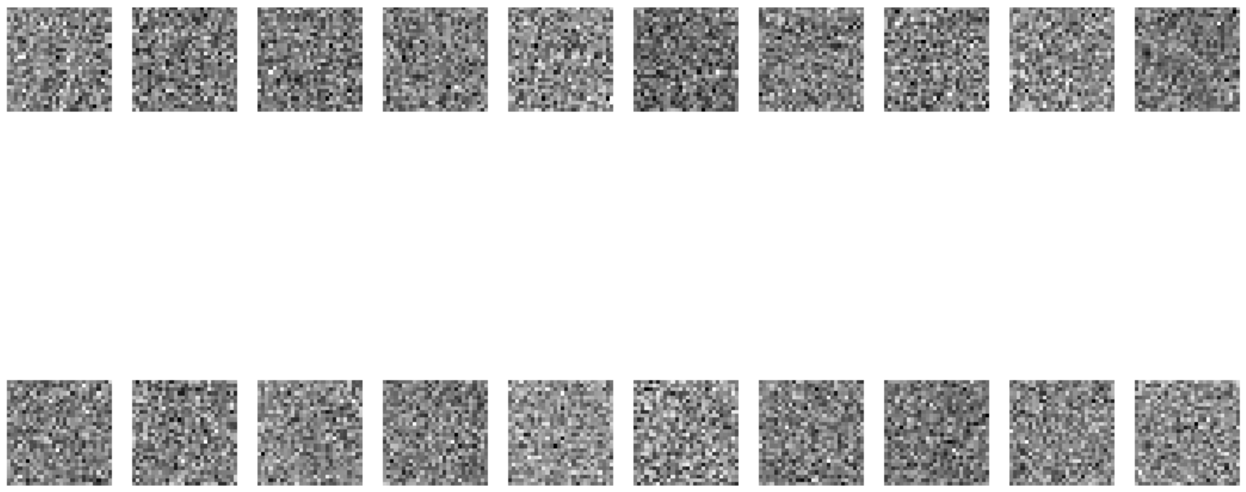
Dreams when wake is turned off



Recognition weights when wake is turned off

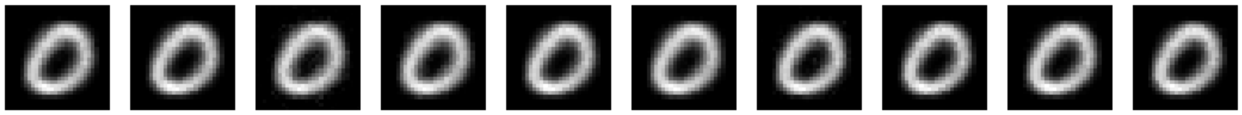


Generative weights when wake is turned off

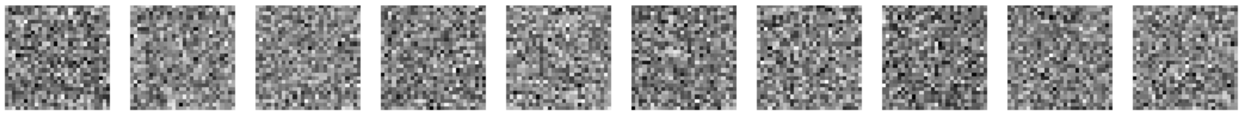


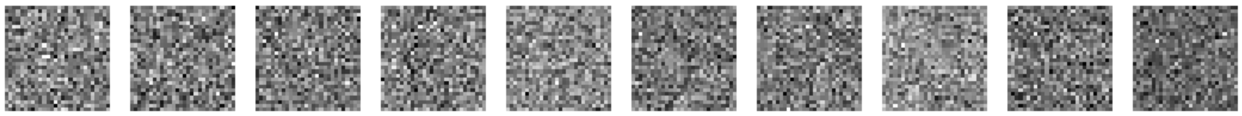
When turning off training in the sleep phase, the cost decreases. In fact, it has a similar cost to training with having the sleep phase on. The cost after training was 1890 for the 0s dataset. We see that the dreams are also the same as the ones generated by the model when training is on for both phases. However, we see that the recognition weights have learned nothing since they were not updated, but the generative weights have learned how to represent 0.

Dreams when sleep is turned off

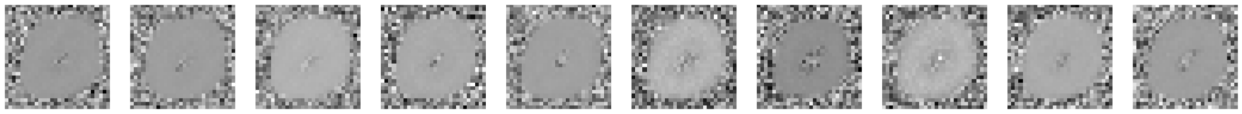


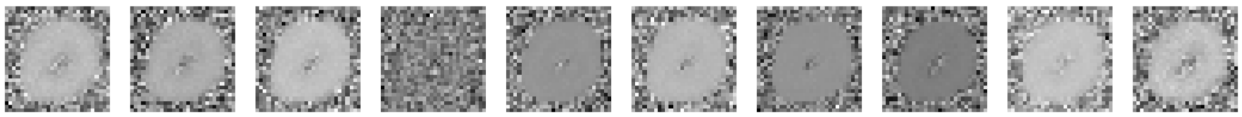
Recognition weights when sleep is turned off





Generative weights when sleep is turned off





The reason for the cost being much higher when wake is turned off is that the code for estimating the cost looks at the generative probabilities. Since we do not update recognition weights when wake is turned off, this does not affect the cost. Also, the updates for the generative weights rely on the input, therefore the input values get propagated when running feed forward and the generative weights are able to update.