Quantum Assisted Helmholtz Machines - an Implementation

Gautham Umasankar

Implementation

A Helmholtz machine with three recognition layers and generation layers each was implemented. The input given was 28x28 grayscale images, stretched out into an array of 784 elements. The second and third layers had 120 and 50 stochastic binary variables, respectively. Each layers has an Affine transformation followed by a sigmoid non-linearity. The output of the sigmoid layers is taken as the probability distribution from which the binary variables are sampled.

The variables in the deepest layer in the generative network are generated by sampling the D-Wave 2000Q processor, as described in this reference (Quantum Assisted Helmholtz Machines) The network is shown below.

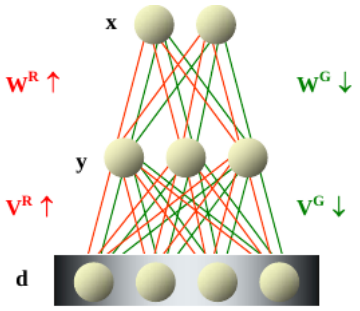

Figure 1: Neural Network

Notation

1. **d** - real world data, in this case, a gray scale image
2. **y** - vector of binary variables in the first hidden layer
3. **x** - vector of binary variables in the second hidden layer
4. V^G, W^G, V^R, V^G - Affine layer matrices, as shown in figure $b^{vG}, b^{wG}, b^{vR}, b^{wR}$ are the corresponding bias vectors in the affine layer
5. J, h - A Matrix and vector containing the quantum interaction parameters according to $H = \sum_{i<j} J_{ij} x_i x_j + \sum_i h_i x_i$

Details of Implementation

The notation used here and in the Python code are according to this reference.

Wake Phase

Here, we train the generative network parameters, $J, h, V^G, W^G, b^{vG}, b^{wG}$ to minimize the

$$G(\theta_G) = -\sum_{\boldsymbol{xy}} P_R(\boldsymbol{xy}|\boldsymbol{d}) ln(P_G(\boldsymbol{xyd}))$$

Where $P_G()$ and $P_R()$ denote probabilities of generation and recognition respectively.

Since $ln(P_G(\boldsymbol{xyd}))$ is averaged by $P_R(\boldsymbol{xy}|\boldsymbol{d})$, we minimize $ln(P_G(\boldsymbol{xyd}))$ where $\boldsymbol{d}, \boldsymbol{y}, \boldsymbol{x}$ are drawn from the recognition distribution.

We find the relevant gradients using the relevant matrices and biases. For finding the gradient w.r.t J_{ij} and h_i we note the fact that

$$ln(P_G(\boldsymbol{x})) = ln(\langle \boldsymbol{x}| \rho |\boldsymbol{x}\rangle)$$

and

$$ln(\langle \boldsymbol{x}| \rho |\boldsymbol{x}\rangle) \geq \langle \boldsymbol{x}| ln(\rho) |\boldsymbol{x}\rangle)$$

Since $ln\rho = -\beta H - lnZ$ where Z is the partition function and H is the Hamiltonian which is $H = \sum_{i<j} J_{ij} x_i x_j + \sum_i h_i x_i$.
Differentiating w.r.t J_{ij} and h_i we get

$$\frac{-\partial ln(P_G(\boldsymbol{x}))}{\beta \partial J_{ij}} \leq x_i x_j$$

$$\frac{-\partial ln(P_G(\boldsymbol{x}))}{\beta \partial h_i} \leq x_i$$

Where x_i and x_j are drawn from the recognition distribution. The gradients of the matrices and biases (which affect $P_G(\boldsymbol{y}|\boldsymbol{x})$ and $P_G(\boldsymbol{d}|\boldsymbol{y})$ can be calculated in a straightforward manner. They are shown in this reference.

Sleep Phase

Here, we train the Recognition network parameters, V^R, W^R, b^{vR}, b^{wR} to minimize the

$$R(\theta_R) = -\sum_{\boldsymbol{xy}} P_G(\boldsymbol{xy}|\boldsymbol{d}) ln(P_R(\boldsymbol{xy}|\boldsymbol{d}))$$

Since the sum is weighted by $P_G(\boldsymbol{xy}|\boldsymbol{d})$, we sample from the generative distribution and pass it back up the recognition distribution. We then calculate the gradients of $ln(P_R(\boldsymbol{xy}|\boldsymbol{d})$ w.r.t recognition network parameters. This is straightforward since we have only two affine layers and two sigmoid layers. It is shown in this reference.

Training

The network was trained for a total of 70 epochs. In each epoch, 60 randomly chosen images from the MNIST dataset were passed through the recognition network to tune generative parameters in the wake phase. Each wake phase is followed by a sleep phase were the generative network is sampled and the data obtained is used to train the recognition network. Finally the generative network is used to generate some digits and we can see that high level features like edges and corners are learned. The generator produces digits along with some novelty too. Some generated images with the images in the dataset with the closest L2 distance are shown below.

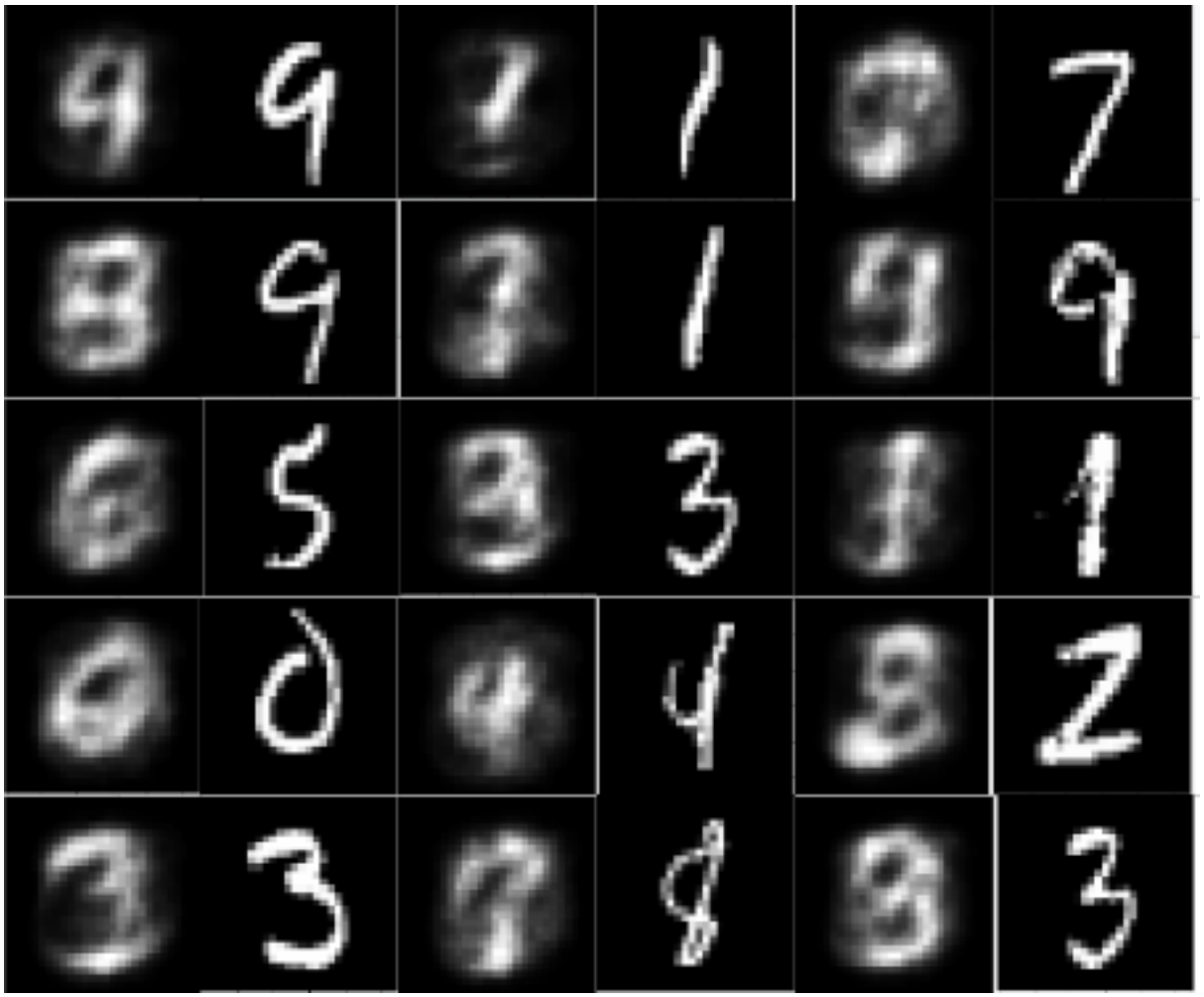

Figure 2: Generated images vs Closest Images