HOMEWORK 2

CMU 10-707: DEEP LEARNING (FALL 2017)
https://piazza.com/cmu/fall2017/10707
OUT: Oct 9, 2017
DUE: Oct 23, 2017 11:59 pm
TAs: Otilia Stretcu, Shunyuan Zhang

Instructions

Hand in answers to all questions below. For Problem 5, the goal of your write-up is to document the experiments you have done and your main findings, so be sure to explain the results. Be concise and to the point – do not write long paragraphs or only vaguely explain results.

- The answers to all questions should be in pdf form (please use LaTeX). Do not hand-write your submission- we reserve the right to take off points or not accept hand-written submissions.
- Please include a README file with instructions on how to execute your code. Your code should contain the implementation for the RBM, autoencoder and denoising autoencoder. Please create your own implementations and do not use any toolboxes.
- Package your code and README document using `zip` or `tar.gz` in a file called `10707-A2-*yourandrewid*.[zip|tar.gz]`.
- Submit your PDF write-up to the Gradescope assignment "Homework 2" and your packaged code to the Gradescope assignment "Code for Homework 2."

Problem 1 (10 pts)

Consider a simple convolutional neural network (CNN) applied to a $d \times d$ image. This CNN contains a single convolutional layer with a kernel of size $k \times k$, stride s pixels, and 2 feature maps, followed by a single $p \times p$ subsampling (max pooling) layer, followed by a softmax layer of size m. Derive the backpropagation algorithm with respect to the adjustable parameters of this network and discuss how the standard backpropagation algorithm must be modified when evaluating the derivatives of an error function with respect to the adjustable parameters in the network.

Your answer here

Assume there's no FC between pooling and softmax. The forward process can be represented like this:

$$f_q = F(S_{q_{q=1,2}})$$

$$\hat{y} = sigmoid(f_q)$$

$$s_q(i,j) = \frac{1}{p^2}\sum_{u=0}^{p-1}\sum_{v=0}^{p-1} C_q(p_i - u, pj - v), i, j = 1, 2...a$$

$$c_q(i,j) = \sum_{u=\frac{-k}{2}}^{\frac{k}{2}-1}\sum_{v=\frac{-k}{2}}^{\frac{k}{2}-1} I(i-u, j-v)k_{1,q}(u,v) + b_q$$

The process of Backpropagation can be derived:

$$\Delta f_q = \frac{\partial L}{\partial f} = \sum_{i=1}^{m} \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial f_q} = (\hat{y} - y)\hat{y}(1 - \hat{y}) = \Delta \hat{y} \tag{1}$$

$$\Delta S_{q_{q=1,2}} = F^{-1}(\Delta f) \tag{2}$$

$$\Delta C_q(i,j) = \frac{1}{p^2} \Delta S_q(\lceil \frac{i}{p} \rceil, \lceil \frac{j}{p} \rceil), i, j = 1, 2...a \tag{3}$$

$$k_{p,q(u,v)} = \frac{\partial L}{\partial k_{p,q}(u,v)} = \sum_{i=1}^{a} \sum_{j=1}^{a} \frac{L}{C_q(i,j)} \frac{\partial C_q(i,j)}{\partial k_{p,q}(u,v)} \tag{4}$$

Then we can get the gradient Δk and Δb_q:

$$\Delta k_{p,q}(u,v) = \sum i = 1^a \sum_{j=1}^{a} \Delta C_q(i,j) C_q(i,j)(1 - C_q(i,j)) \cdot I(i-u, j-v) \tag{5}$$

If we assume that,

$$\Delta C_{q,\sigma(i,j)} = \Delta C_q(i,j) C_q(i,j)(1 - C_q(i,j)),$$

and then rotate I 180 degree,

$$I_{rot180}(u-i, v-j) = I(i-u, j-v)$$

Then we can get,

$$\begin{aligned} \Delta k_{p,q}(u,v) &= \sum_{i=1}^{a} \sum_{j=1}^{a} I_{rot180}(u-i, v-j) \cdot \Delta C_{q,\sigma(i,j)} \\ &= I_{rot180} * \Delta C_{q,\sigma} \end{aligned} \tag{6}$$

And similarly, we can get Δb,

$$\Delta b_q = \sum_{i=1}^{a} \sum_{j=1}^{a} \frac{\partial L}{\partial C_{q(i,j)}} \cdot \frac{\partial C_{q(i,j)}}{\partial b_q} \quad = \Delta C_{q,\sigma(i,j)} \tag{7}$$

And from the derivation above, we can find the differences between the standard backpropagation and CNN backpropagation in several aspects:

- When doing CNN backpropagation, we need to map the gradient of the output layer first to the pooling layer. Then map the gradient of pooling layer to the convolutional layer to calculate Δk and Δb.
- And when calculate Δk, we need to use convolution operation, which need to rotate the input 180 degree.

Problem 2 (10 pts)

Consider a directed graphical model with K random variables. By marginalizing out the variables in order, show that the representation

$$p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k | \mathrm{pa}_k) \tag{8}$$

for the joint distribution of a directed graph is correctly normalized, provided each of the conditional distributions is normalized.

Your answer here

Assume that the nodes in the graph has been numbered such that x_1 is the root node. We can then marginalize over the nodes in reverse order, starting with x_K

$$\sum_{x_1} ... \sum_{x_K} p(\mathbf{x}) = \sum_{x_1} ... \sum x_K p(x_K|pa_K) \prod_{k=1}^{K-1} p(x_k|pa_k)$$
$$= \sum_{x_1} ... \sum_{x_K} \prod_{k=1}^{K-1} p(x_k|pa_k),$$

since each of the conditional distributions is assumed to be correctly normalized. Then repeating the process above $K-2$ times, we can get that

$$\sum_{x1} p(x_1|\phi) = 1,$$

which means that the joint distribution of a directed graph is correctly normalized.

Problem 3 (10pts)

In class, we considered a bipartite undirected graphical model, called Restricted Boltzmann Machine (RBM), shown in Figure 1. An RBM contains a set of observed and hidden random variables, where binary visible variables $v_i \in \{0, 1\}$, $i = 1, ..., D$, are connected to binary hidden variables $h_j \in \{0, 1\}$, $j = 1, ..., P$. The energy of the joint configuration is given by:

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\sum_{i=1}^{D}\sum_{j=1}^{P} W_{ij} v_i h_j - \sum_{i=1}^{D} v_i b_i - \sum_{j=1}^{P} h_j a_j. \tag{9}$$

where $\theta = \{W, \mathbf{a}, \mathbf{b}\}$ denote model parameters, with $\mathbf{a}$ and $\mathbf{b}$ representing the bias terms, and $\mathbf{v} = \{v_1, ..., v_D\}$, $\mathbf{h} = \{h_1, ..., h_P\}$. The probability of the joint configuration is given by the Boltzmann distribution:

$$P_\theta(\mathbf{v}, \mathbf{h}) = \frac{1}{\mathcal{Z}} \exp(-E(\mathbf{v}, \mathbf{h}; \theta)), \tag{10}$$

where $\mathcal{Z} = \sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}; \theta))$ denotes the normalizing constant. Derive that conditioned on observed variables $\mathbf{v}$, the distribution over the hidden variables factorizes (so the hidden variables become independent conditioned on $\mathbf{v}$):

$$P_\theta(h_1, ..., h_P|\mathbf{v}) = \prod_{j=1}^{P} p_\theta(h_j|\mathbf{v}), \tag{11}$$

where

$$P_\theta(h_j = 1|\mathbf{v}) = \frac{1}{1 + \exp(-\sum_{i=1}^{D} W_{ij} v_i - a_j)}. \tag{12}$$

Note: inferring the states of the hidden variables conditioned on the observed data is easy. This represents a major advantage of this model, which is successfully used in many domains, ranging from collaborative filtering to speech recognition.

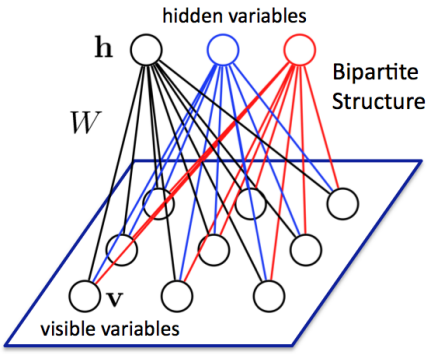


Figure 1: An example of undirected graphical model called a Restricted Boltzmann Machine.

Your answer here

$$
\begin{aligned}
P_\theta(h_1, ..., h_P|\mathbf{v}) &= \frac{p_\theta(\mathbf{v}, \mathbf{h})}{\sum_{h'} p_\theta(\mathbf{v}, \mathbf{h}')} \\
&= \frac{\frac{\exp(\mathbf{h}^T W \mathbf{v} + \mathbf{b}^T \mathbf{v} + \mathbf{a}^T \mathbf{h})}{Z}}{\frac{\sum_{h'} \exp(\mathbf{h}'^T W \mathbf{v} + \mathbf{b}^T \mathbf{v} + \mathbf{a}^T \mathbf{h}')}{Z}} \\
&= \frac{\exp(\sum_j h_j W_j \mathbf{v} + a_j h_j)}{\sum_{h'_1} ... \sum_{h'_H} \exp(\sum_j h'_j W_j \mathbf{v} + a_j h'_j)} \\
&= \frac{\prod_j^P \exp(h_j W_j \mathbf{v} + a_j h_j)}{\sum_{h'_1} ... \sum_{h'_H} \prod_j^P \exp(h'_j W_j \mathbf{v} + a_j h'_j)} \\
&= \frac{\prod_j^P \exp(h_j W_j \mathbf{v} + a_j h_j)}{(\sum_{h'_1} \exp(h'_1 W_1 \mathbf{v} + a_1 h'_1)) ... (\sum_{h'_H} \exp(h'_H W_H \mathbf{v} + a_H h'_H))} \\
&= \frac{\prod_j^P \exp(h_j W_j \mathbf{v} + a_j h_j)}{\prod_j^P (\sum_{h'_j} \exp(h'_j W_j \mathbf{v} + a_j h'_j))} \\
&= \frac{\prod_j^P \exp(h_j W_j \mathbf{v} + a_j h_j)}{\prod_j^P (1 + \exp(a_j + W_j \mathbf{v}))} \\
&= \prod_j^P \frac{\exp(h_j W_j \mathbf{v} + a_j h_j)}{1 + \exp(a_j + W_j \mathbf{v})} \\
&= \prod_j^P p_\theta(h_j|\mathbf{v})
\end{aligned}
$$

Problem 4 (10 pts)

Consider the use of iterated conditional modes (ICM) to minimize the energy function (that we considered in class for image denoising example) given by:

$$E(\mathbf{x}, \mathbf{y}) = h \sum_i x_i - \beta \sum_{i,j} x_i x_j - \eta \sum_i x_i y_i, \tag{13}$$

where $x_i \in \{-1, 1\}$ is a binary variable denoting the state of pixel i in the unknown noise-free image, i and j are indices of neighboring pixels, $y_i \in \{-1, 1\}$ denotes the corresponding value of pixel i in the observed noisy image,

and h, β and η are positive constants. The joint distribution is defined as:

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{\mathcal{Z}} \exp\left(-E(\mathbf{x}, \mathbf{y})\right) \tag{14}$$

- (5 pts) Write down an expression for the difference in the values of the energy associated with the two states of a particular variable x_j, with all other variables held fixed, and show that it depends only on quantities that are local to x_j in the graph.
- (5 pts) Consider a particular case of the energy function above in which the coefficients $\beta = h = 0$. Show that the most probable configuration of the latent variables is given by $x_i = y_i$ for all i.

Your answer here

- Given other variables fixed, we can get,

$$E(x_j = 1) - E(x_j = -1) = 2h - 2\beta \sum_i x_i - 2\eta y_j$$

  Because x_i and y_j is neighbors of x_j, so the difference depends only on quantities that are local to x_j in the graph.

- To get the most probable configuration, we need to get the highest value of $p(\mathbf{x}, \mathbf{y})$, which means we need the lowest energy. And with $\beta = h = 0$, we can get the energy function like this,

$$E(\mathbf{x}, \mathbf{y}) = -\eta \sum_i x_i y_i,$$

  to get smallest E, we need the largest $\sum_i x_i y_i$. And because x and y are both binary variables, which $x_i and y_i \in \{-1, 1\}$. Then when $x_i = y_i$, we can get $x_i y_i = 1$, which can make the value of E smallest.

Problem 5 (60 pts)

For this question you will write your own implementation of the Contrastive Divergence algorithm for training RBMs, and compare it to an Autoencoder model. Please do not use any toolboxes. We recommend that you use MATLAB or Python, but you are welcome to use any other programming language if you wish.

The goal is to build a generative model of images of 10 handwritten digits of "zero", "one",..., "nine". The images are 28 by 28 in size (MNIST dataset), which we will be represented as a vector $\mathbf{x}$ of dimension 784 by listing all the pixel values in raster scan order. The labels t are 0,1,2,...,9 corresponding to 10 classes as written in the image. There are 3000 training cases, containing 300 examples of each of 10 classes, 1000 validation (100 examples of each of 10 classes), and 3000 test cases (300 examples of each of 10 classes). they can be found in the file digitstrain.txt, digitsvalid.txt and digitstest.txt:

http://www.cs.cmu.edu/~rsalakhu/10707/assignments.html

Format of the data: digitstrain.txt contains 3000 lines. Each line contains 785 numbers (comma delimited): the first 784 real-valued numbers correspond to the 784 pixel values, and the last number denotes the class label: 0 corresponds to digit 0, 1 corresponds to digit 1, etc. digitsvalid.txt and digitstest.txt contain 1000 and 3000 lines and use the same format as above.

Contrastive Divergence (CD), Autoencoders

Implement the CD algorithm for training an RBM model. Implement both an autoencoder and a denoising autoencoder model.

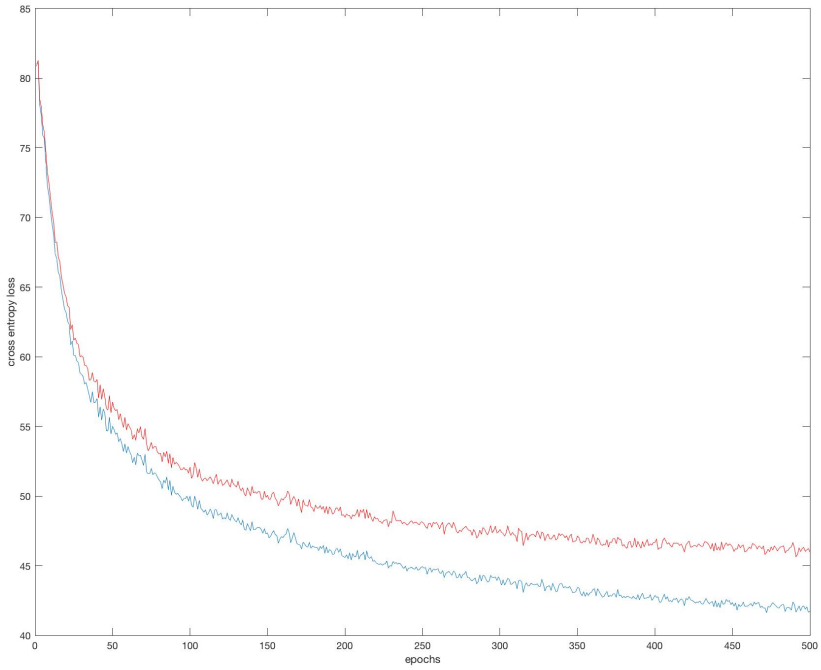


Figure 2: The loss entropy of RBM with k = 1

a) Basic generalization [20 points]

Train an RBM model with 100 hidden units, starting with CD with $k = 1$ step. For initialization use samples from a normal distribution with mean 0 and standard deviation 0.1. Choose a reasonable learning rate (e.g. 0.1 or 0.01). Use your own judgement to decide on the stoping criteria (i.e number of epochs or convergence criteria).

As a proxy for monitoring the progress, plot the average training cross-entropy reconstruction error on the y-axis vs. the epoch number (x-axis). On the same figure, plot the average validation cross-entropy error function.

Examine the plots of training error and validation error. How does the network's performance differ on the training set versus the validation set during learning? Visualize the learned W as 100 28x28 images (plot all filters as one image, as we have seen in class). Do the learned features exhibit any structure?

Your answer here

In this problem, I choose the learning rate 0.1 and use the number of epochs as the stop criteria. I choose the number of epoch is 500.
The cross entropy loss is shown in the figure 2. The blue line is the loss for training data. The red line is the loss for the validation data. We can see that the the loss keeps going down. And the loss of validation data is higher than the loss of training data. The loss firstly decrease rapidly, and then slows down. And we can see that the loss of validation is getting stop decreasing as the training epoch grows.
The visualization of learned weights is shown in the figure 3. We can see that, the weights have learned some structure after training. There are some white edges in the weights, which should be feature leaned by the network.

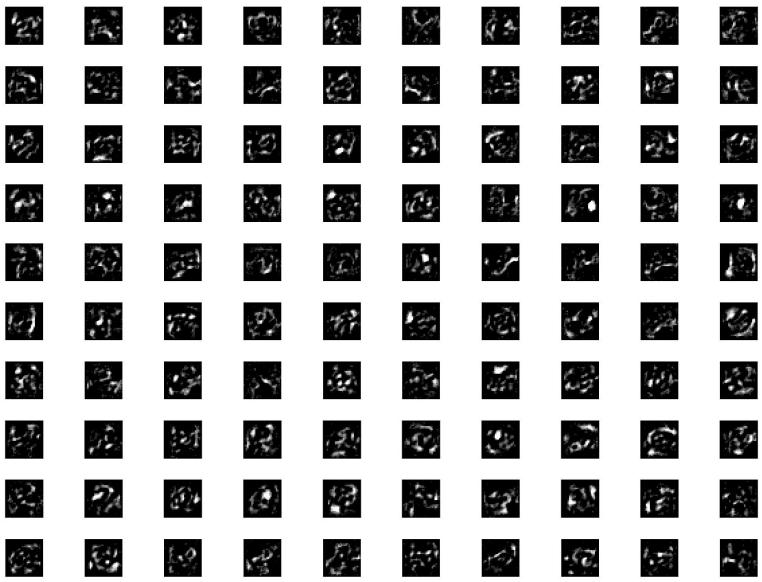


Figure 3: The weights of RBM with k = 1

b) Number of CD steps [5 points]

Try learning an RBM model with CD with $k = 5$, $k = 10$, and $k = 20$ steps. Describe the effect of this modification on the convergence properties, and the generalization of the network.

Your answer here

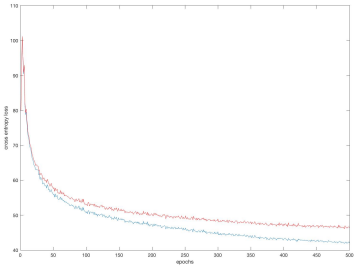
The cross loss of RBM with different k value is shown in the figure . From the figure we can see that, the converge speed with among three figure has little difference. And as mentioned in the class, usually k = 1 gives a good result in practice for converging speed.
But we can see that as the k increase, the network tend to more robust to generalize. When k=1, the validation loss line becomes flat from about 100 epochs. But when k=20, the validation loss line becomes flat from about 300 epochs. This phenomenon comes up because as k increase, the bias of the Gibbs sampling becomes smaller, which make the network more robust to generalize

c) Sampling from the RBM model [5 points]

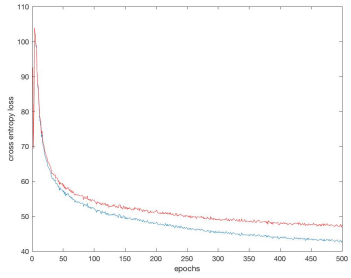
To qualitatively test the model performance, initialize 100 Gibbs chains with random configurations of the visible variables, and run the Gibbs sampler for 1000 steps. Display the 100 sampled images. Do they look like handwritten digits?

Your answer here

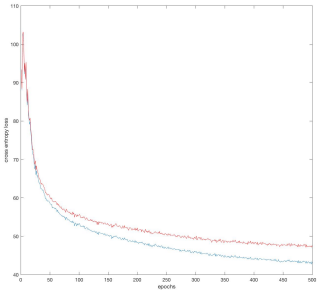
Figure 5 shows the sampling from RBM model. We can see that the sampling has some feature but not very obvious. However we can still see some edges of number.



(a) The Loss of RBM wight k = 5



(b) The Loss of RBM wight k = 10



(c) The Loss of RBM wight k = 20

Figure 4: The Loss of RBM with different k value

d) Unsupervised Learning as pretraining [5 points]

Use the weights you learned in question **a)** to initialize a 1-layer neural network with 100 hidden units. Train the resulting neural network to classify the 10 digits, as done in the first assignment. Does this pretraining help compared to training the same neural network initialized with random weights in terms of classification accuracy? Describe your findings.

Your answer here

The comparison of classification error between training without RBM pretraing and with RBM pretraining is shown in figure 6. And as we can see that, the classification error of training with RBM pretraining(6(b)) is lower than the error of training without it(6(a)) after 100 epochs training. And we also can see that with pretraining, the converge speed of the network is improved. To get error rate at 0.1, with RBM pretraining, we need to train the network just about 10 epochs.

e) Autoencoder [5 points]

Instead of training an RBM model, train an autoencoder model with 100 sigmoid hidden units. For initializing the autoencoder, use samples from a normal distribution with mean 0 and standard deviation 0.1. Does the model learn useful filters? Visualize the learned weights. Similar to RBMs, use the learned autoencoder weights to initialize a 1-layer neural network with 100 hidden units. Train the resulting neural network to classify the 10 digits. How does it compare to pretraining with RBMs?

Your answer here

After training autoencoder for 500 epochs, I got the visualization of weights, which is shown in figure 7. And from the figure we can see that autoencoder learned some feature from the training, but not good as the feature learned by

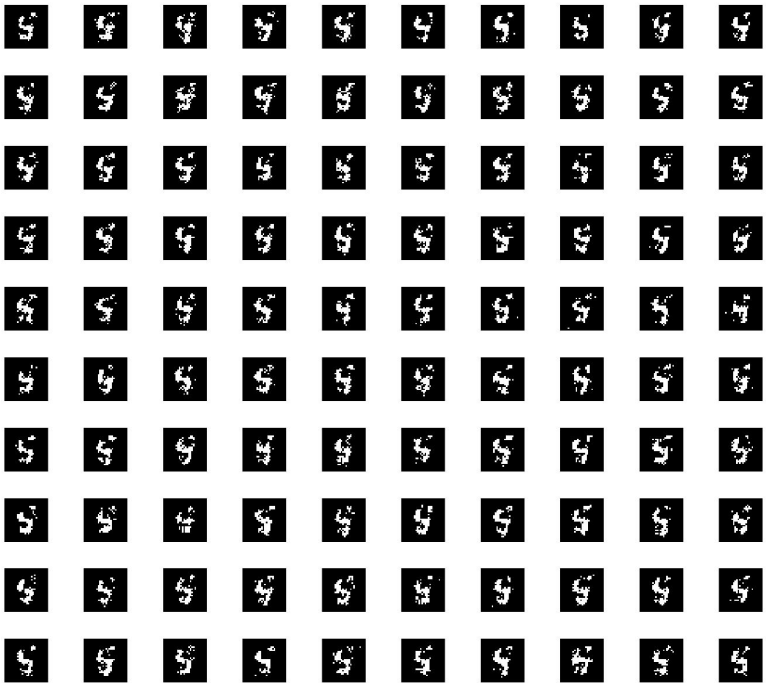
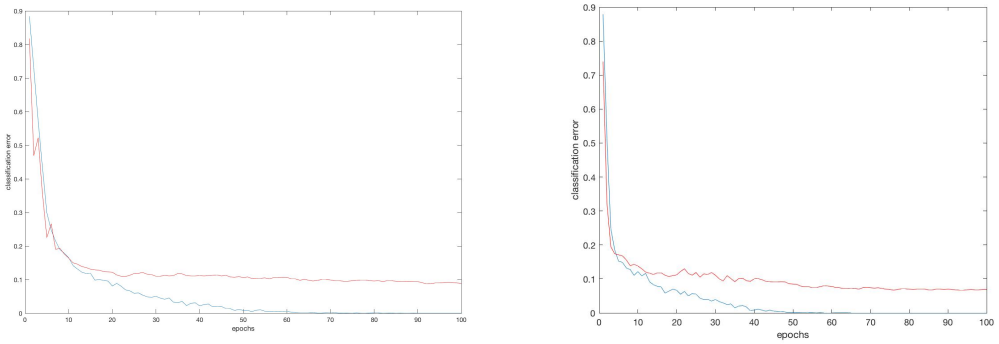


Figure 5: Sampling fromRBM with k = 1



(a) The classification error without RBM pretraining

(b) The classification error with RBM pretraining

Figure 6: Comparison of classification error of MNIST digits with and without RBM pretraining

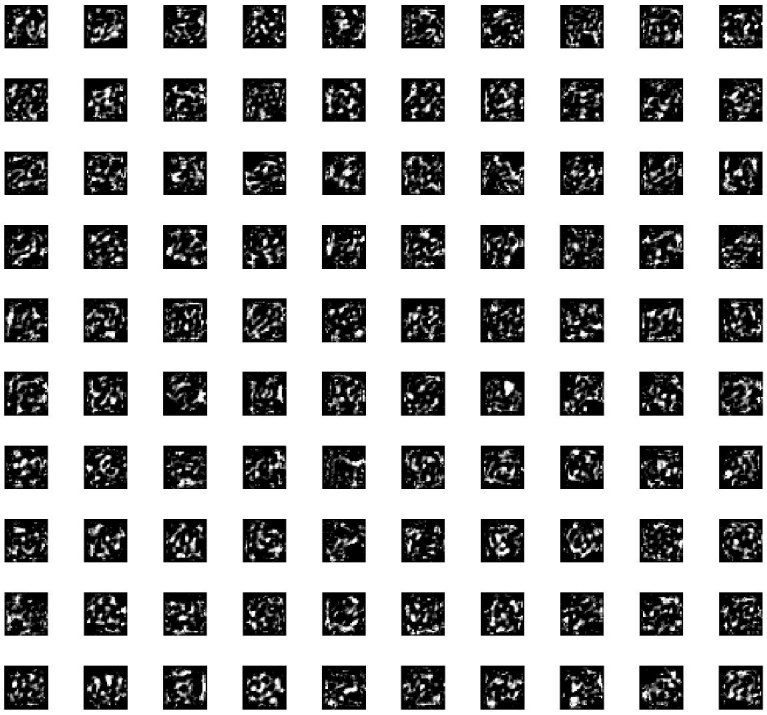


Figure 7: The weights of autoencoder with 100 hidden units

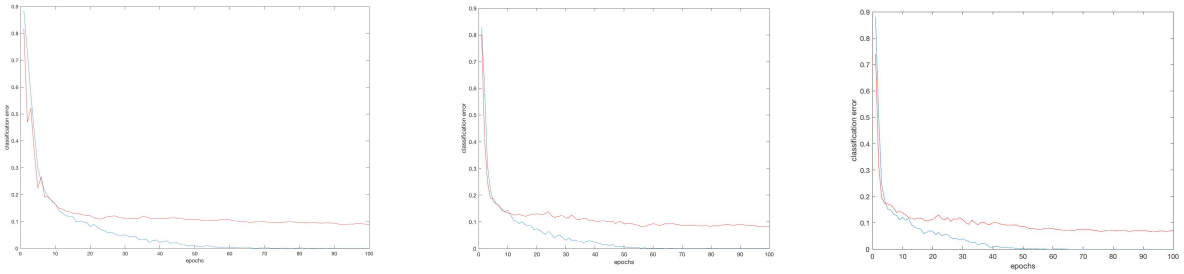
RBM. Because in the weights visualization of autoencoder, the white points scatter everywhere, not like weights of RBM. And this phenomenon implies that the weights are still random in some extents.
And in figure 8, I showed the classification error of MNIST with different pretraining condition. And we can see that without any pretraining, the classification error is highest. And with autoencdoer pretraining, the classification error goes down, but is not as low as training with RBM pretraining.

f) Denoising Autoencoder [10 points]

Train a denosing autoencoder by using drop-out on the inputs. We recommend a drop-out rate of 10%, but feel free to explore other rates. Visualize the learned weights. Use the learned weights to initialize a 1-layer neural network with 100 hidden units. Train the resulting neural network to classify the 10 digits. How does it compare to pretrainig with standard autoencoders and RBMs in terms of classification accuracy?

Your answer here

Figure 9 shows the visualization of weights after training a denoising autoencoder 250 epochs. And we can see that the feature learned by the denosing autoencoder is more obvious than the feature learned by the normal autoencoder. The white points in the weights has formed some structure.
And figure 10 shows comparison among the classification error of MNIST with different pretraining condition. We can see that using denoising autoencoder, the classification error is lower than that with normal autoencoder, and even is as low as the result with pretraining using RBM.



(a) The classification error without pretraining (b) The classification error with Autoencoder pretraining (c) The classification error with RBM pretraining

Figure 8: Comparison of classifcation error of MNIST digits with and without autroencoder pretraining

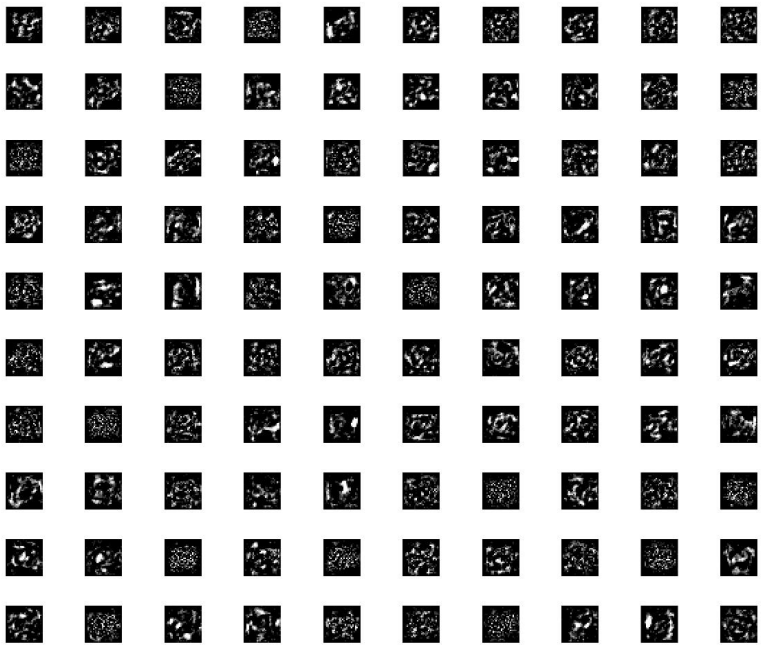
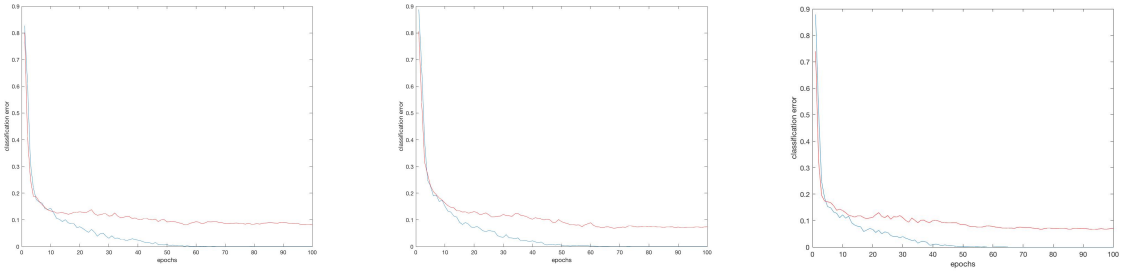


Figure 9: The weights of denoising autoencoder with 100 hidden units



(a) The classification error with Autoencoder pretraining

(b) The classification error with Denoising Autoencoder pretraining

(c) The classification error with RBM pretraining

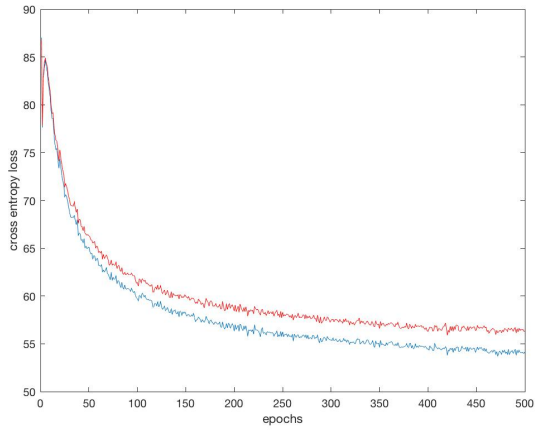
Figure 10: Comparison of classifcation error of MNIST digits with different pretraining conditions

g) Number of hidden units [10 points]

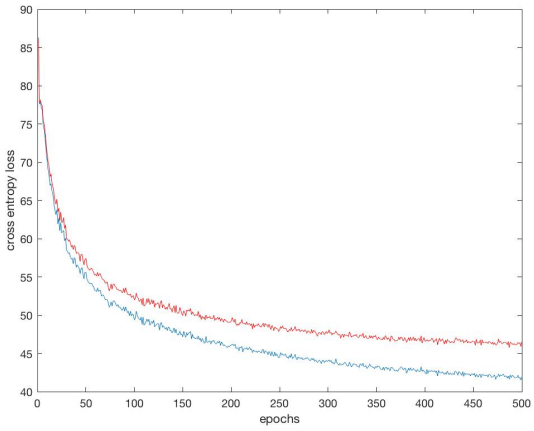
Try training RBMs, autoencoders and denoising autoencoders with 50, 100, 200, and 500 hidden units. Describe the effect of this modification on the convergence properties, and the generalization of the network.

Your answer here

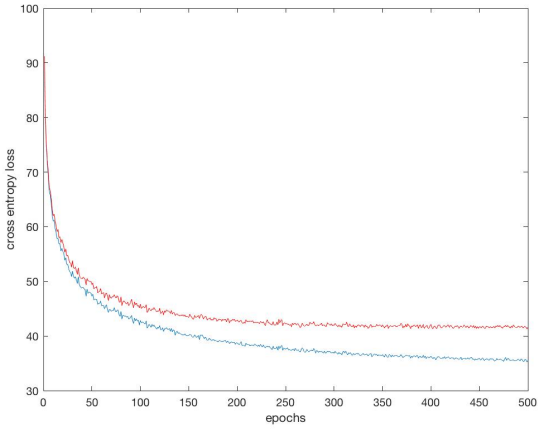
From figure 11, figure 12, and figure 13, as the number of hidden units increases, all of networks converge faster. And the loss for both training data and validation data decreases.
For RBM when hidden units more than 100, the network becomes more easy to generalize. For autoencoder and denoising autoencoder, when hidden units equals 500, the network becomes more easy to generalize. And as expected, more hidden units can make the network get less loss, but it can lead the network less robust to generalization.



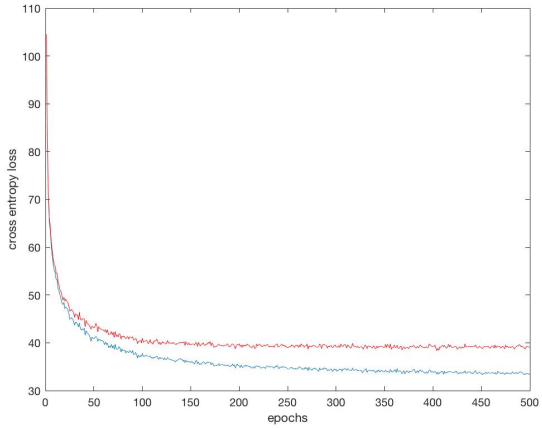
(a) The loss of RBM with 50 hidden units



(b) The loss of RBM with 100 hidden units

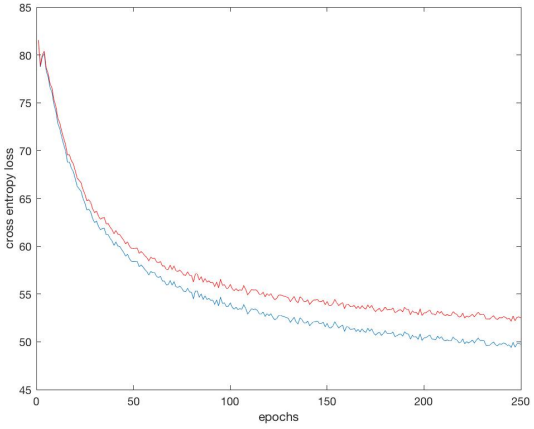


(c) The loss of RBM with 200 hidden units

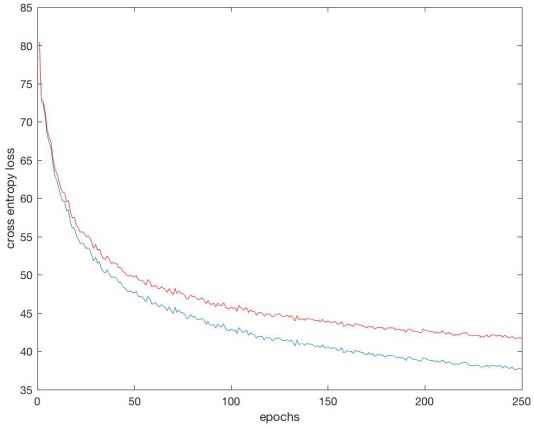


(d) The loss of RBM with 500 hidden units

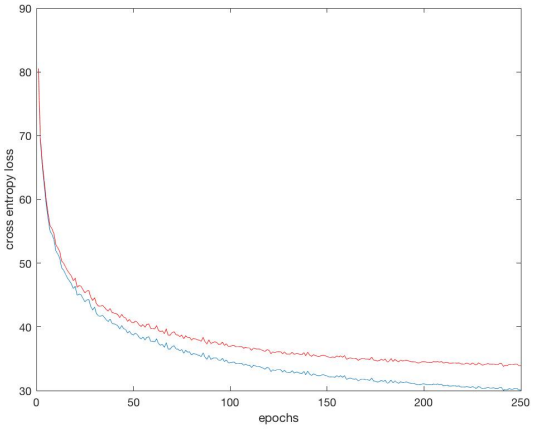
Figure 11: Loss of RBM with different hidden units



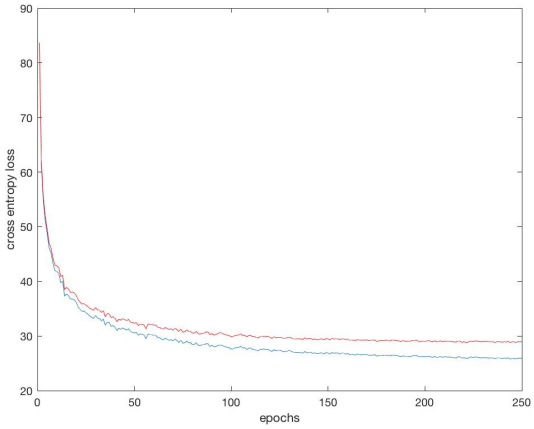
(a) The loss of autoencoder with 50 hidden units



(b) The loss of RBM with 100 hidden units

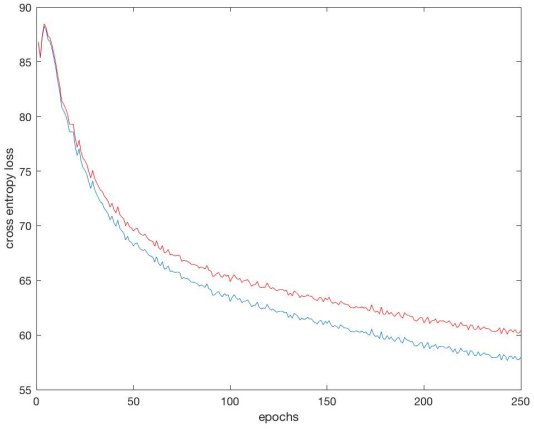


(c) The loss of RBM with 200 hidden units

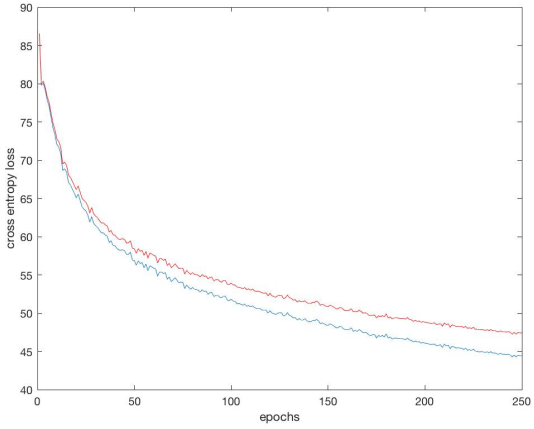


(d) The loss of RBM with 500 hidden units

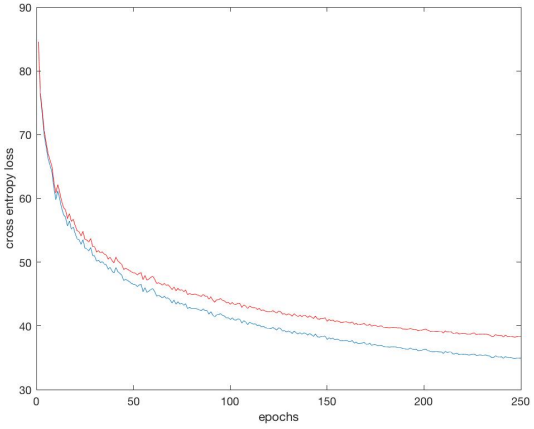
Figure 12: Loss of RBM with different hidden units



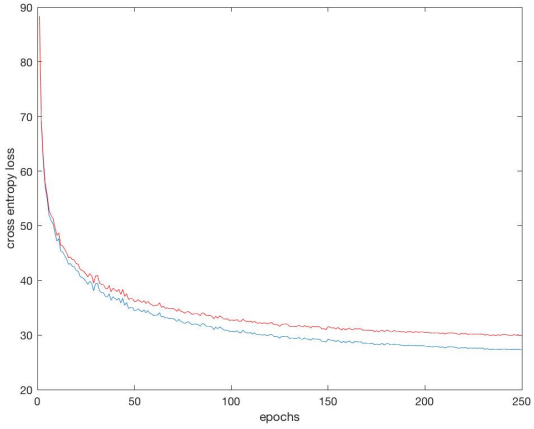
(a) The loss of denoising with 50 hidden units



(b) The loss of RBM with 100 hidden units



(c) The loss of RBM with 200 hidden units



(d) The loss of RBM with 500 hidden units

Figure 13: Loss of denoising autoencoder with different hidden units