10-707: Deep Learning, Fall 2017

Homework 2

Lecturer: Russ Salakhutdinov *Name: Yuan Liu(yuanl4)*

1 Problem 1

How to compute derivation:

- The input image is a 2-D matrix X, the size is $d \times d$.
- Convolutional Layer: the parameter of feature map $i \in \{1, 2\}$ is W^i. They are two 2-D matrixes, the size of each matrix is $k \times k$.

  The output of the convolutional layer is two 2-D matrixes $Y^i, i \in \{1, 2\}$, the size of each matrix is $\frac{d-k+1}{s} \times \frac{d-k+1}{s}$

$$Y^i_{m,n} = \sum_{p=1}^{k} \sum_{q=1}^{k} X_{(m-1)s+p,(n-1)s+q} W^i_{p,q}$$

  Then we can get:

$$\frac{\partial Y^i_{m,n}}{\partial W^i_{p,q}} = X_{(m-1)s+p,(n-1)s+q} \tag{1}$$

  The derivation input of the convolutional layer is two 2-D matrix $\frac{\partial Loss}{\partial Y^i}, i \in \{1, 2\}$, the size of each matrix is $\frac{d-k+1}{s} \times \frac{d-k+1}{s}$. Then we can get:

$$\frac{\partial Loss}{\partial W^i_{p,q}} = \sum_{m,n=1}^{\frac{d-k+1}{s}} \frac{\partial Loss}{\partial Y^i}_{m,n} \frac{\partial Y^i_{m,n}}{\partial W^i_{p,q}} = \sum_{m,n=1}^{\frac{d-k+1}{s}} \frac{\partial Loss}{\partial Y^i}_{m,n} X_{(m-1)s+p,(n-1)s+q}$$

- Pooling Layer: The output of pooling layer is two 2-D matrixes $P^i, i \in \{1, 2\}$, the size of each matrix is $(\frac{d-k+1}{s} - p + 1) \times (\frac{d-k+1}{s} - p + 1)$.

$$P^k_{m,n} = \max_{i,j \in \{1,\ldots,p\}} Y^k_{m+i-1,n+j-1}$$

  The derivation input of the pooling layer is two 2-D matrix $\frac{\partial Loss}{\partial P^i}, i \in \{1, 2\}$, the size of each matrix is $(\frac{d-k+1}{s} - p + 1) \times (\frac{d-k+1}{s} - p + 1)$. Then we gan get the derivation output is:

$$\frac{\partial P^k_{m,n}}{\partial Y^K_{m+i-1,n+j-1}} = \mathbb{1}(i, j = \arg\max_{i,j} Y^k_{m+i-1,n+j-1}) \tag{2}$$

$$\frac{\partial Loss}{\partial Y^k}_{m,n} = \sum_{i,j=1}^{p} \frac{\partial Loss}{\partial P^k}_{m+i-1,n+j-1} \mathbb{1}(i, j = \arg\max_{i,j \in \{1,\ldots,p\}} Y^k_{m+i-1,n+j-1})$$

- Flatten Layer, which convert the two 2-dimension matrixes into a vector. The length of this vector is $2(\frac{d-k+1}{s}-p+1)^2$.

$$
\begin{aligned}
P^k_{m,n} &= F_{(k-1)*(\frac{d-k+1}{s}-p+1)^2+(m-1)*(\frac{d-k+1}{s}-p+1)+n} \\
\frac{\partial P^k_{m,n}}{\partial F_i} &= \mathbb{1}(i=(k-1)*(\frac{d-k+1}{s}-p+1)^2+(m-1)*(\frac{d-k+1}{s}-p+1)+n) \\
\frac{\partial Loss}{\partial P^k_{m,n}} &= \frac{\partial Loss}{\partial F_{(k-1)*(\frac{d-k+1}{s}-p+1)^2+(m-1)*(\frac{d-k+1}{s}-p+1)+n}}
\end{aligned}
\tag{3}
$$

- Softmax Layer: Assume $k=2(\frac{d-k+1}{s}-p+1)^2$, the output of this softmax layer is a vector S, which size is k.

$$S_i = \frac{e^{F_i}}{\sum_{j=1}^k e^{F_j}}$$

The derivation input of the softmax layer is a vector $\frac{\partial Loss}{\partial S}$, which size is k.

$$\frac{\partial S_i}{\partial F_j} = \frac{\frac{e^{F_i}}{\partial F_j}(\sum_{j=1}^k e^{F_j}) - e^{F_i}e^{F_j}}{(\sum_{j=1}^k e^{F_j})^2} = \frac{\mathbb{1}(i=j)(\sum_{j=1}^k e^{F_j}) - e^{F_i}e^{F_j}}{(\sum_{j=1}^k e^{F_j})^2} = \frac{\mathbb{1}(i=j)}{\sum_{j=1}^k e^{F_j}} - \frac{e^{F_i+F_j}}{(\sum_{j=1}^k e^{F_j})^2}$$

$$\frac{\partial Loss}{\partial F_j} = \sum_{i=1}^k \frac{\partial Loss}{\partial S_i}\frac{\partial S_i}{\partial F_j} = \sum_{i=1}^k \frac{\partial Loss}{\partial S_i}(\frac{\mathbb{1}(i=j)}{\sum_{j=1}^k e^{F_j}} - \frac{e^{F_i+F_j}}{(\sum_{j=1}^k e^{F_j})^2})$$

The difference between Linear layer: If we use a Linear layer to replace the combination of Convolutional Layer, Pooling layer and Flatten Layer. The Weight matrix W will be a 2-D matrix of size: $d^2 \times k$

$$F_i = \sum_{j=1}^{d^2} W_{j,i} X_{1+\lfloor (j-1)/d \rfloor, j-d(\lfloor (j-1)/d \rfloor)} + b_i$$

The derivation input of this Linear layer is a vector of size k.

$$\frac{\partial F_k}{\partial W_{j,i}} = X_{1+\lfloor (j-1)/d \rfloor, j-d(\lfloor (j-1)/d \rfloor)} \mathbb{1}(k=i)$$

$$\frac{\partial Loss}{\partial W_{j,i}} = \frac{\partial Loss}{\partial F_i}\frac{\partial F_i}{\partial W_{j,i}} = \frac{\partial Loss}{\partial F_i} X_{1+\lfloor (j-1)/d \rfloor, j-d(\lfloor (j-1)/d \rfloor)}$$

It is a super clear formula. However, when calculating the derivation of the convolutional layer:

$$\frac{\partial Loss}{\partial W} = \frac{\partial Loss}{\partial F}\frac{\partial F}{\partial P}\frac{\partial P}{\partial Y}\frac{\partial Y}{\partial W}$$

The derivation of $\frac{\partial Y}{\partial W}, \frac{\partial P}{\partial Y}, \frac{\partial F}{\partial P}$ is given by (1)(2)(3).

2 Problem 2

Because the model is a directed graphical model, so it is a directed acyclic graph. Then we can find an order $\{I_i\}_{i=1}^K$, that $pa_{I_i} \subset \{x_{I_j}\}_{j>i}$. For simplicity, we can just assume that $\{x_i\}_{i=1}^K$ satisfies this order, which means $pa_{x_i} \subset \{x_j\}_{j>i}$.

$$\int p(x)dx = \int \prod_{k=1}^K p(x_k|pa_k)dx_1...dx_k = \int p(x_1|pa_1)dx_1 \int \prod_{k=2}^K p(x_k|pa_k)dx_2...dx_k$$

We can do this calculation, because $x_1 \notin \cap_{k=2}^{K} pa_k$. We also know $\int p(x_1|pa_1)dx_1 = 1$. Then we can know:

$$\int p(x)dx = \int \prod_{k=1}^{K} p(x_k|pa_k)dx_1...dx_k = \int \prod_{k=2}^{K} p(x_k|pa_k)dx_2...dx_k$$

By the same way, we can finally get

$$\int p(x)dx = \int p(x_K|pa_K)dx_K$$

Because x_K is the last one in the node list $x_1, ..., x_K$, so $pa_K = \emptyset$, $\int p(x_K|pa_K)dx_K = \int p(x_k)dx_K = 1$. Finally we get:

$$\int p(x)dx = 1$$

3 Problem 3

$$p_\theta(v,h) = \frac{1}{Z} exp(v^T Wh + v^T b + h^T a)$$

$$\begin{aligned}
p_\theta(h|v) &= \frac{p(v,h)}{p(h)} = \frac{\frac{1}{Z}exp(v^T Wh + v^T b + h^T a)}{\sum_h \frac{1}{Z} exp(v^T Wh + v^T b + h^T a)} \\
&= \frac{\exp(v^T Wh + h^T a)}{\sum_h exp(v^T Wh + h^T a)} \\
&= \frac{\prod_{i=1}^{P} exp(h_i(W^T v + a)_i)}{\sum_{h_1} exp(h_1(W^T v + a)_1) \times \sum_{h_2} exp(h_2(W^T v + a)_2) \times ... \times \sum_{h_P} exp(h_P(W^T v + a)_P)} \\
&= \prod_{i=1}^{P} \frac{exp(h_i(W^T v + a)_i)}{\sum_{h_i \in \{0,1\}} exp(h_i(W^T v + a)_i)} \\
&= \prod_{j=1}^{P} \sigma(h_i(W^T v + a)_i)
\end{aligned}$$

$$\begin{aligned}
p_\theta(h_j = 1|v) &= \sum_{h_j=1, h_{i\neq j} \in \{0,1\}} p_\theta(h|v) = \sum_{h_j=1, h_{i\neq j} \in \{0,1\}} \prod_{j=1}^{P} \sigma(h_i(W^T v + a)_i) \\
&= \sigma((W^T v + a)_j) \sum_{h_{i\neq j} \in \{0,1\}} \prod_{j \in \{1,..,P\}-\{j\}} \sigma(h_i(W^T v + a)_i) \\
&= \sigma((W^T v + a)_j) \prod_{j \in \{1,..,P\}-\{j\}} \sum_{h_{i\neq j} \in \{0,1\}} \sigma(h_i(W^T v + a)_i) \\
&= \sigma((W^T v + a)_j)
\end{aligned}$$

By this formula we can know $p_\theta(h_j|v) = \sigma(h_j(W^T v + 1)_j)$. Thus

$$p_\theta(v,h) = \prod_{j=1}^{P} p_\theta(h_j|v)$$

4 Problem 4

4.1

$$E(x_m = 1, x_{i\neq m}, y) = h\sum_{i\neq m} x_i + h - \beta \sum_{i\neq m, j\neq m} x_i x_j - \beta \sum_{i\in local(m)} x_i - \eta \sum_{i\neq m} x_i y_i - \eta y_m$$
$$E(x_m = -1, x_{i\neq m}, y) = h\sum_{i\neq m} x_i - h - \beta \sum_{i\neq m, j\neq m} x_i x_j + \beta \sum_{i\in local(m)} x_i - \eta \sum_{i\neq m} x_i y_i + \eta y_m$$

Then we can get:

$$E(x_m = 1, x_{i\neq m}, y) - E(x_m = -1, x_{i\neq m}, y) = 2h - 2\beta \sum_{i\in local(m)} x_i - 2\eta y_m$$

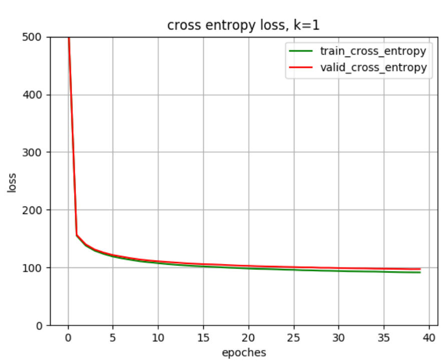
So the difference in the value of energy depends only on quantities that are local to x_m in the graph.

4.2

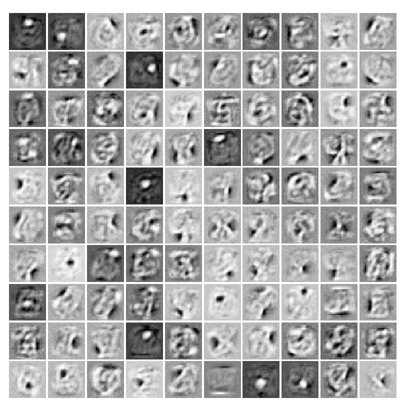
If $\beta = h = 0$, we can get: $E(x, y) = -\eta \sum_i x_i y_i$. If we want to minimize the energy, we need to maximize $\sum_i x_i y_i$. Because $x_i \in \{-1, +1\}, y_i \in \{-1, +1\}$, so the maximum of $x_i y_i$ is 1, which can be got by $x_i = y_i$. So the most probable configuration of the latent variables is given by $x_i = y_i$ for all i.

5 Problem 5

5.1 (a)



(a) Problem a: cross entropy loss



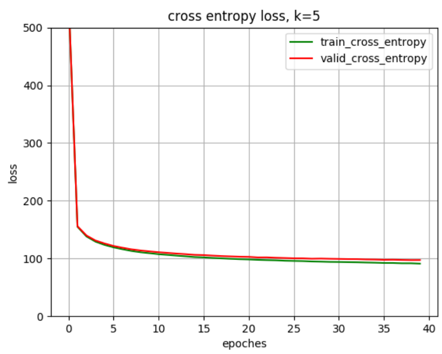
(b) Problem a: visualization of W

In my implementation, I choose *batch size* = 32, *learning rate* = 0.1.

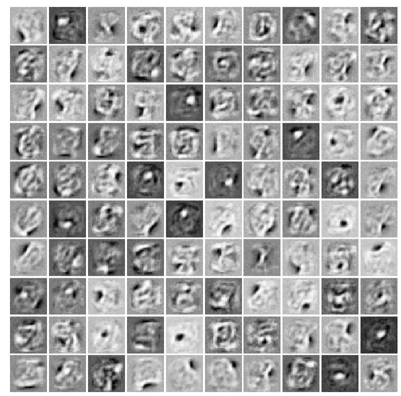
The cross entropy loss on training set and validation set keeps decreasing without over fitting, even when I train more than 200 epochs. The loss on training set is less than it on the validation set, and the difference is small.

The learned W has some structures and it looks like the stroke contour.

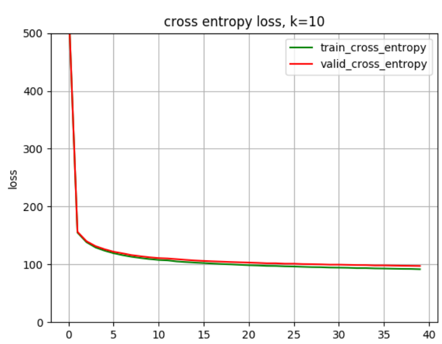
5.2 (b)



(c) Problem b: k=5 cross entropy loss



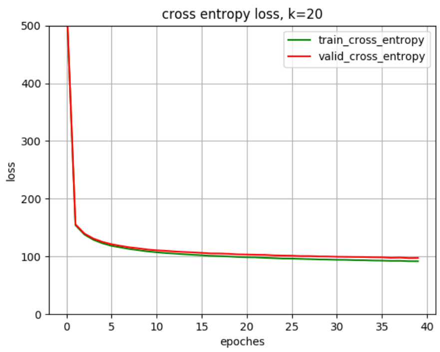
(d) Problem b: visualization of W



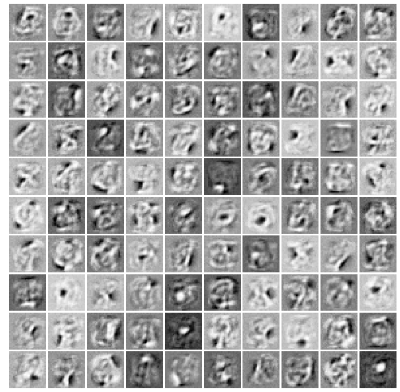
(e) Problem b: k=10 cross entropy loss

(f) Problem b: visualization of W

Measured by the cross entropy loss and the visualization of W, there is almost no difference between $k = 1, 5, 10, 20$. However, when I choose $k = 1$, my implement can not generate reasonable images. If I choose $k \geq 5$, my implement can generate good images.



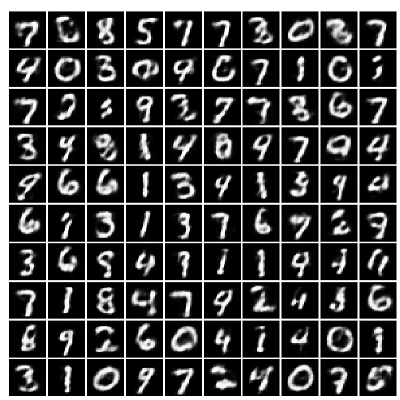
(g) Problem b: k=20 cross entropy loss



(h) Problem b: visualization of W

5.3 (c)

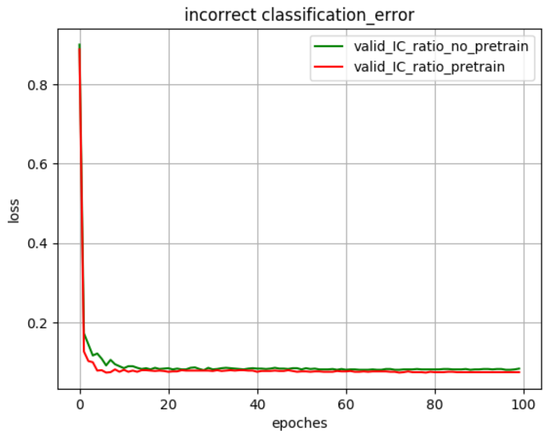
(i) Problem c: k=1 generated images



(j) Problem c: k=20 generated images

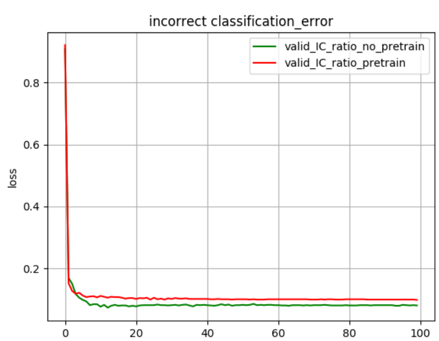
If I choose $k \geq 5$, the generated images look like handwritten digits. The figure is the plot of $p(x|\tilde{h})$.

5.4 (d)

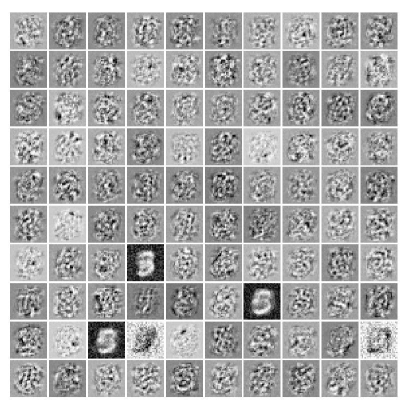


After pre-training, the model converges faster, and get accuracy 92.6%. Without pre-training, the accuracy is 92.2%, which is slightly worse than the pre-training accuracy.

5.5 (e)

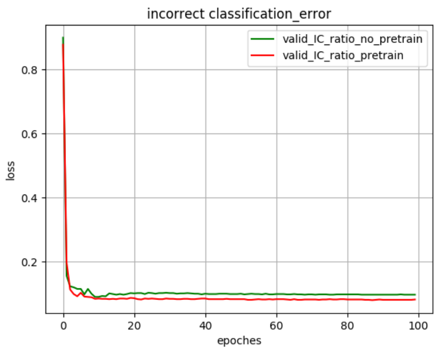


(k) Problem e: Pre-training accuracy vs no-pre-training accuracy

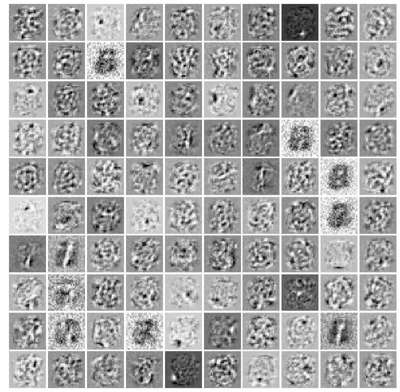


(l) Problem e: Visualization of W

I use mean square error as the loss function of autoencoder, and batch size = 32. The pre-training decreases the performance, and there is almost no structure in W.



(m) Problem f: Pre-training accuracy vs no-pre-training accuracy



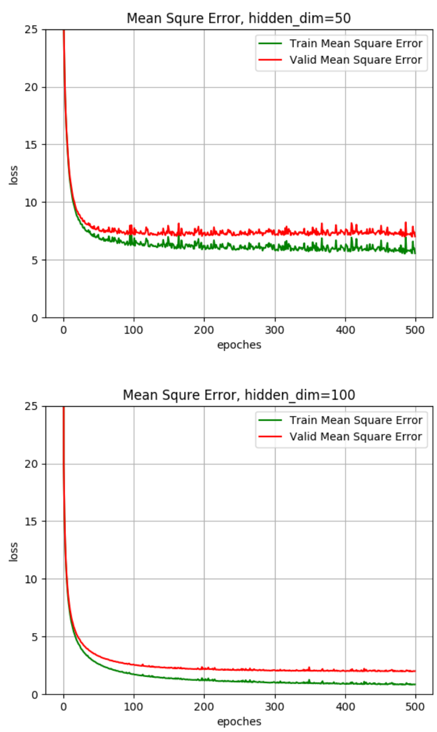
(n) Problem f: Visualization of W

5.6 (f)

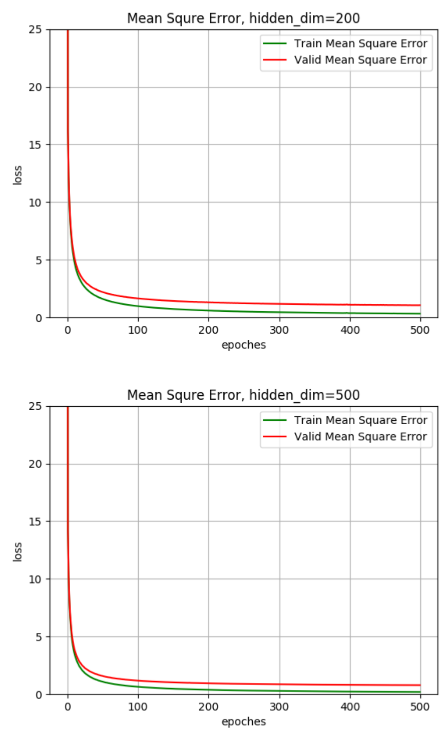
In this time pre-training increases the performance slightly and the performance is almost the same as RBM pre-training. The visualization of W has some structure know. Some of the filters looks like digits: 1 and 8.

5.7 (g)

With the increasing of hidden dimension, the loss will decrease a lot, and the loss curve will become more and more smooth.



(o) Problem g: different hidden dimension



(p) Problem d: different momentum