# CS542 HW3

**CS 542: Machine Learning** **Spring 2019**

Problem Set 3

*Student: Haoying Zhou* *Due: Mar 21, 2019*

1. **(60 points) Written Problems**

(a) (20 points) Bishop 5.3

(b) (20 points) Bishop 5.4

(c) (20 points) Bishop 5.26

Solution:

(a)

**5.3** ($\star\star$) Consider a regression problem involving multiple target variables in which it is assumed that the distribution of the targets, conditioned on the input vector $\mathbf{x}$, is a Gaussian of the form

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x}, \mathbf{w}), \boldsymbol{\Sigma}) \tag{5.192}$$

where $\mathbf{y}(\mathbf{x}, \mathbf{w})$ is the output of a neural network with input vector $\mathbf{x}$ and weight vector $\mathbf{w}$, and $\boldsymbol{\Sigma}$ is the covariance of the assumed Gaussian noise on the targets. Given a set of independent observations of $\mathbf{x}$ and $\mathbf{t}$, write down the error function that must be minimized in order to find the maximum likelihood solution for $\mathbf{w}$, if we assume that $\boldsymbol{\Sigma}$ is fixed and known. Now assume that $\boldsymbol{\Sigma}$ is also to be determined from the data, and write down an expression for the maximum likelihood solution for $\boldsymbol{\Sigma}$. Note that the optimizations of $\mathbf{w}$ and $\boldsymbol{\Sigma}$ are now coupled, in contrast to the case of independent target variables discussed in Section 5.2.

Since it has given that $p(t|x,w) = \mathcal{N}(t|y(x,w),\Sigma)$

Then, it can derive that:

$$p(T|x,w) = \prod_{n=1}^{N} \mathcal{N}(t_n|y(x_n,w),\Sigma)$$

Thus, it can obtain that:

$$E(w,\Sigma) = \frac{1}{2}\sum_{n=1}^{N}\{[y(x_n,w)-t_n]^T\Sigma^{-1}[y(x_n,w)-t_n]\} + \frac{N}{2}\ln|\Sigma| + constant$$

Since $\Sigma$ is unknow and the constant indicates that $w$ *and* $\Sigma$ are independent. If $\Sigma$ is known as well as fixed, the equation can be rewritten as:

$$E(w,\Sigma) = \frac{1}{2}\sum_{n=1}^{N}\{[y(x_n,w)-t_n]^T\Sigma^{-1}[y(x_n,w)-t_n]\} + constant$$

And we can solve $w_{ML}$ by minimizing it. Even if Σ is unknown, the constant includes Σ, solving $w_{ML}$ will involve Σ.

(b)

**5.4** ($\star\star$) Consider a binary classification problem in which the target values are $t \in \{0, 1\}$, with a network output $y(\mathbf{x}, \mathbf{w})$ that represents $p(t = 1|\mathbf{x})$, and suppose that there is a probability $\epsilon$ that the class label on a training data point has been incorrectly set. Assuming independent and identically distributed data, write down the error function corresponding to the negative log likelihood. Verify that the error function (5.21) is obtained when $\epsilon = 0$. Note that this error function makes the model robust to incorrectly labelled data, in contrast to the usual error function.

Since it is a binary classification problem, then it can obtain that:

$$p(t=1|x,w) = (1-\varepsilon)p(t_s=1|x,w) + \varepsilon p(t_s=0|x,w)$$

And $y(x,w) = y = p(t_s = 1|x,w)$,

Thus, we can derive that:

$$p(t=1|x,w) = (1-\varepsilon)y(x,w) + \varepsilon[1-y(x,w)]$$
$$p(t=0|x,w) = (1-\varepsilon)[1-y(x,w)] + \varepsilon y(x,w)$$

Combine the above two equations, it can get that:

$$p(t|x,w) = (1-\varepsilon)y^t(1-y)^{1-t} + \varepsilon(1-y)^t y^{1-t}$$

Using the negative logarithm, we can obtain the error function:

$$E(w) = -\sum_{n=1}^{N}\ln[(1-\varepsilon){y_n}^{t_n}(1-y_n)^{1-t_n} + \varepsilon(1-y_n)^{t_n}{y_n}^{1-t_n}]$$

If $\varepsilon = 0$, it becomes:

$$E(w) = -\sum_{n=1}^{N}[t_n \ln y_n + (1-t_n)\ln(1-y_n)]$$

(c)

**5.26** ($\star\star$) Consider a multilayer perceptron with arbitrary feed-forward topology, which is to be trained by minimizing the *tangent propagation* error function (5.127) in which the regularizing function is given by (5.128). Show that the regularization term $\Omega$ can be written as a sum over patterns of terms of the form

$$\Omega_n = \frac{1}{2}\sum_k (\mathcal{G}y_k)^2 \tag{5.201}$$

where $\mathcal{G}$ is a differential operator defined by

$$\mathcal{G} \equiv \sum_i \tau_i \frac{\partial}{\partial x_i}. \tag{5.202}$$

By acting on the forward propagation equations

$$z_j = h(a_j), \qquad a_j = \sum_i w_{ji} z_i \tag{5.203}$$

with the operator $\mathcal{G}$, show that $\Omega_n$ can be evaluated by forward propagation using the following equations:

$$\alpha_j = h'(a_j)\beta_j, \qquad \beta_j = \sum_i w_{ji}\alpha_i. \tag{5.204}$$

where we have defined the new variables

$$\alpha_j \equiv \mathcal{G}z_j, \qquad \beta_j \equiv \mathcal{G}a_j. \tag{5.205}$$

Now show that the derivatives of $\Omega_n$ with respect to a weight $w_{rs}$ in the network can be written in the form

$$\frac{\partial \Omega_n}{\partial w_{rs}} = \sum_k \alpha_k \{\phi_{kr} z_s + \delta_{kr}\alpha_s\} \tag{5.206}$$

where we have defined

$$\delta_{kr} \equiv \frac{\partial y_k}{\partial a_r}, \qquad \phi_{kr} \equiv \mathcal{G}\delta_{kr}. \tag{5.207}$$

Write down the backpropagation equations for $\delta_{kr}$, and hence derive a set of back-propagation equations for the evaluation of the $\phi_{kr}$.

Proof: According to (5.128), it can obtain that:

$$\Omega_n = \frac{1}{2}\sum_k \left(\frac{\partial y_{nk}}{\partial \xi}|_{\xi=0}\right)^2 = \frac{1}{2}\sum_k \left(\sum_i \frac{\partial y_{nk}}{\partial x_i}\frac{\partial x_i}{\partial \xi}|_{\xi=0}\right)^2$$

Let $\tau_i = \frac{\partial x_i}{\partial \xi}$ and $y_k = y_{nk}$, then the equation becomes:

$$\Omega_n = \frac{1}{2}\sum_k (\sum_i \tau_i \frac{\partial}{\partial x_i} y_{nk})^2 = \frac{1}{2}\sum_k (\mathcal{G}_{y_k})^2$$

Combining with definitions in the description, it can derive that:

$$\alpha_j = \sum_i \tau_i \frac{\partial z_j}{\partial x_i} = \sum_i \tau_i \frac{\partial h(a_j)}{\partial x_i} = \sum_i \tau_i \frac{\partial h(a_j)}{\partial a_j}\frac{\partial a_j}{\partial x_i} = h'(\alpha_j)\sum_i \tau_i \frac{\partial}{\partial x_i}\alpha_j$$
$$= h'(a_j)\beta_j$$

And

$$\beta_j = \sum_k \tau_k \frac{\partial \alpha_j}{\partial x_k} = \sum_k \tau_k \frac{\partial \sum_i w_{ji} z_i}{\partial x_k} = \sum_k \tau_k \sum_i w_{ji}\frac{\partial z_i}{\partial x_i} = \sum_i w_{ji} \sum_k \tau_k \frac{\partial z_i}{\partial x_i}$$
$$= \sum_i w_{ji}\alpha_i$$

According to equation $\frac{\partial a_j}{\partial w_{ji}} = z_i$

Then, it can obtain:

$$\frac{\partial \Omega_n}{\partial w_{rs}} = \frac{\partial}{\partial w_{rs}}\left[\frac{1}{2}\sum_k (\mathcal{G}y_k)^2\right] = \frac{1}{2}\sum_k \frac{\partial (\mathcal{G}y_k)^2}{\partial \mathcal{G}y_k}\frac{\partial \mathcal{G}y_k}{\partial w_{rs}} = \sum_k \mathcal{G}y_k \mathcal{G}\left(\frac{\partial y_k}{\partial w_{rs}}\right)$$
$$= \sum_k \alpha_k \mathcal{G}(\frac{\partial y_k}{\partial a_r}\frac{\partial a_r}{\partial w_{rs}}) = \sum_k \alpha_k \mathcal{G}(\delta_{kr} z_s)$$
$$= \sum_k \alpha_k [\mathcal{G}(\delta_{kr}) z_s + \mathcal{G}(z_s)\delta_{kr}] = \sum_k \alpha_k (\phi_{kr} z_s + \alpha_s \delta_{kr})$$

Since $\delta_{kr} = \frac{\partial y_k}{\partial a_r}$

Thus, it can derive that:

$$\delta_{kj} = \frac{\partial y_k}{\partial a_j} = \frac{\partial y_k}{\partial a_r}\frac{\partial a_r}{\partial a_j} = h'(a_j)\sum_i w_{ij}\,\delta_{ki}$$

And

$$\phi_{kr} = \mathcal{G}\delta_{kr} = h''(a_r)\beta_r \sum_i w_{ir}\,\delta_{ki} + h'(a_r)\sum_i w_{ir}\,\phi_{ki}$$

# Programming report

2. **(120 points) Programming**

DO the exercise found in `Sparse_Autoencoder` Turn in all the .m files and the final "weigths.jpg" file.

Without loss of generality, I uploaded all files in folder "starter" to source code. But I only modified some of them: sampleIMAGES.m , sparseAutoencoderCost.m , computeNumericalGradient.m and add "close all; clear all; clc" to main code named as train.m.

***For sampleIMAGES.m :***

To pick random $8 \times 8$ matrix, firstly, to prevent index error, divide the dimensions(512 and 512) of the matrix by patch size which is 8 and pick one of the numbers. And for the dimension which is 10, just randomly pick one of the numbers.

Then, it can easily pick $8 \times 8$ matrix randomly. And convert $8 \times 8$ matrix into $64 \times 1$ matrix.

***For computeNumericalGradient.m:***

The gradient can be compute as:

$$\frac{\partial J(\theta)}{\partial \theta} = g(\theta) \approx \frac{J(\theta + \varepsilon) - J(\theta - \varepsilon)}{2\varepsilon}$$

where $\varepsilon$ is a small constant, say $\varepsilon = 10^{-4}$

***For sparseAutoencoderCost.m:***

The activation function is sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Since it has three layers totally, Then it can obtain that:

$$a^{(1)} = x\ (original\ data)$$
$$z^{(2)} = W^{(1)} a^{(1)} + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)} a^{(2)} + b^{(2)}$$
$$y = h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

Using backpropagation algorithm on training models:
The cost function is defined as:

$$J_1(W,b;x,y) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2$$

Thus, the overall cost function is:

$$J_1(W,b) = \frac{1}{m}\sum_{i=1}^{m} J_1\left(W,b;x^{(i)},y^{(i)}\right) + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left[\frac{1}{2}\left\|h_{W,b}\left(x^{(i)}\right) - y^{(i)}\right\|^2\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$

$s_l$ is the number of neurons in layers, and $n_l$ is the number of layers.
However, the sparsity penalty term need to be added into the overall cost function.
To calculate the sparsity penalty term, let:

$$\widehat{\rho_J} = \frac{1}{m}\sum_{i=1}^{m} a_j^{(2)}\left(x^{(i)}\right)$$

And

$$KL(\rho||\widehat{\rho_j}) = \sum_{j}^{s_2} \rho\log\frac{\rho}{\widehat{\rho_J}} + (1-\rho)\log\frac{1-\rho}{1-\widehat{\rho_J}}$$

where $\rho$ is given sparsity parameter and $s_2$ is the number of neurons in the hidden layer.
Thus, the overall cost function for sparse autoencoder is:

$$J(W,b) = J_1(W,b) + \beta\sum_{j}^{s_2} KL(\rho||\widehat{\rho_J})$$

After that, we can calculate the gradients:
For each output unit i in layer $n_l$ (the output layer), set:

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}}\frac{1}{2}\left\|y - h_{W,b}(x)\right\|^2 = -(y_i - a_i^{(n_l)})f'(z_i^{(l)})$$

Where y is the original data ($a_1$).
Then for $l = n_l - 1, n_l - 2, \cdots, 2$
For each node I in layer l, set

$$\delta_i^{(l)} = (\sum_{j=1}^{s_l+1} W_{ji}^{(l)}\delta_j^{(l+1)})f'(z_i^{(l)})$$

Next, compute the desired partial derivatives:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}$$

Write it into matrix form:

$$\nabla_{W^{(l)}} J(W,b;x,y) = \delta^{(l+1)} (a^{(l)})^T$$

$$\nabla_{b^{(l)}} J(W,b;x,y) = \delta^{(l+1)}$$

Then, it can derive the gradients:

$$\nabla_{W^{(l)}} = \frac{1}{m} \delta^{(l+1)} (a^{(l)})^T + \lambda W^{(l)}$$

$$\nabla_{b^{(l)}} = \frac{1}{m} \delta^{(l+1)}$$

However, we need to consider the sparsity, thus it can obtain that, for hidden layer, $\delta_i^{(l)}$ need to add the sparsity penalty terms:

$$\delta_i^{(2)} = [\sum_{j=1}^{s_2} W_{ji}^{(2)} \delta_j^{(3)} + \beta(-\frac{\rho}{\widehat{\rho_j}} + \frac{1-\rho}{1-\widehat{\rho_j}})] f'(z_i^{(2)})$$