Introduction to Deep Learning M2177.0043
Seoul National University

Homework #(**2**)
Seungyong Moon

INSTRUCTIONS

- Anything that is received after the deadline will be considered to be late and we do not receive late homeworks. We do however ignore your lowest homework grade.
- Answers to every theory questions need to be submitted electronically on ETL. Only PDF generated from LaTex is accepted.
- Make sure you prepare the answers to each question separately. This helps us dispatch the problems to different graders.
- Collaboration on solving the homework is allowed. Discussions are encouraged but you should think about the problems on your own.
- If you do collaborate with someone or use a book or website, you are expected to write up your solution independently. That is, close the book and all of your notes before starting to write up your solution.

1 Eigenvalues

Find all eigenvalues of the $X \in \mathbb{R}^{p\times p}$ real matrix filled with some constant value c except the diagonal component. Concretely, the matrix X is structured as following:

```
X = c * np.ones((p,p)) + (-c + 1)*np.eye(p)
```

sol) Let I be the $p \times p$ identity matrix, and C be the matrix corresponding to `c * np.ones((p,p))`. To find the eigenvalues of X, we have to find λ such that the following holds.

$$\begin{aligned}\det(X-\lambda I) &= \det(C-(c-1+\lambda)I)\\ &= 0\end{aligned} \tag{1}$$

Now, let's find the eigenvalues of C. Without solving the characteristic polynomial of C, we can directly find the eigenvalues.

$$\begin{aligned}\mathrm{C}\times(1,1,1,...,1)^T &= pc\times(1,1,1,...,1)^T\\ \mathrm{C}\times(1,-1,0,...,0)^T &= 0\times(1,1,1,...,1)^T\\ \mathrm{C}\times(0,1,-1,...,0)^T &= 0\times(1,1,1,...,1)^T\\ &.\\ &.\\ \mathrm{C}\times(0,0,0,...,1,-1)^T &= 0\times(1,1,1,...,1)^T\end{aligned} \tag{2}$$

Let v_i be a vector whose ith entry is 1, $(i+1)$th entry is -1 and the other entries are 0. Definitely, $\{v_i\}$ are linearly independent so that the dimension of eigenspace associated with 0 is p-1. Hence, we found all eigenvalues, p-1 zeros and pc. Since we know all the eigenvalues, We can get the characteristic polynomial of C.

$$\det(C-tI) = (t-pc)\times t^{p-1} \tag{3}$$

Therefore, the characteristic polynomial of x can be expressed as the following.

$$\begin{aligned}\det(X-\lambda I) &= \det(C-(c-1+\lambda)I)\\ &= ((c-1+\lambda)-pc)\times(c-1+\lambda)^{p-1}\\ &= 0\end{aligned} \tag{4}$$

Therefore $\lambda = 1+(p-1)c$ or $1-c$.

2 Multivariate normal

1. Describe the transformation which takes n independent $N(0,1)$ standard normal samples $Z = [z_1,...,z_n]^T$ and generates correlated random variables that follow a n-dimensional multivariate normal distribution $X = [X_1,...,X_n]^T \sim N(\mu,\Sigma)$

sol) First prove that if a random variable $X = [X_1,...,X_n]^T$ has its covariance matrix Σ_X and $P = (P_{ij})$ is a $n \times n$ matrix, then the covariance matrix of a random variable Y=PX is $P\Sigma_X P^T$.
Let y_i be the ith element of Y. Then,

$$\begin{aligned}Cov(y_i,y_j) &= Cov(\sum_k P_{ik}x_k, \sum_l P_{jl}x_l)\\ &= \sum_k\sum_l P_{ik}P_{jl}Cov(x_k,x_l)\end{aligned} \tag{5}$$

And,

$$\begin{aligned}(P\Sigma_X P^T)_{ij} &= \sum_k\sum_l P_{ik}(\Sigma_X)_{kl}(P^T)_{lj}\\ &= \sum_k\sum_l P_{ik}Cov(x_k,x_l)P_{jl}\end{aligned} \tag{6}$$

Therefore, the covariance matrix of a random variable Y is $P\Sigma_X P^T$.
Given that z_i are i.i.d. standard normal distribution, The covariance matrix of Z is $n \times n$ identity matrix. Therefore, if we apply linear transformation P, which is a $n \times n$ matrix, on Z, the covariance matrix will be changed into $P^T P$.
Now, it is sufficient to find a $n \times n$ matrix P such that $PP^T = \Sigma$. Since Σ is symmetric, Σ can be decomposed as QEQ^T by Spectral Theorem. Here, Q is a $n \times n$ matrix whose columns Q_i are unit eigenvectors of P, and E is a a $n \times n$ diagonal matrix whose ith diagonal entry is the eigenvalue of P corresponding to Q_i. Since P is positive semidefinite, all eigenvalues of P are nonnegative. Therefore we can split E into DD^T, D is a $n \times n$ diagonal matrix, whose ith entry is the square root of ith entry of E.
Now, we have $\Sigma = QEQ^T = QDD^TQ^T = (QD)(QD)^T$. Let $\sqrt{\Sigma} = QD$, then $\sqrt{\Sigma}$ is what we want. If we multiply $\sqrt{\Sigma}$ on the left side of Z, the covariance matrix is changed into Σ.
Now we have to change the mean vector. Since every z_i has zero mean, the mean of Z does not change after the linear transformation. So, if we just add the vector μ on $\sqrt{\Sigma}Z$, the mean is changed to μ. Now we know that $\sqrt{\Sigma}Z+\mu$ has μ as its mean, and has Σ as its variance. Since the linear combination of independent normal random variables is also normal, $X = \sqrt{\Sigma}Z+\mu \sim N(\mu,\Sigma)$

Introduction to Deep Learning M2177.0043
Seoul National University

Homework #(**2**)
Seungyong Moon

---

2. Simulate 10,000 draws of N $\left(0, \Sigma = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}\right)$ using std samples drawn from $N(0,1)$. Contrast the results with 10,000 draws using Numpys existing implementation for multivariate sampling `np.random.multivariate_normal` function. Visualize three scatter plots side-by-side (in 1×3 subplot format, use `markersize=0.1` for better visualization) for 1) the std normal, 2) your transformed multivariate samples, and 3) multivariate samples obtained from the Numpy code. Does your result closely match Numpys results?

sol) From problem 1, we can calculate the eignenvalues and the corresponding eigenvectors of Σ. One is 1.9 with an eigenvector $[1/\sqrt{2}, 1/\sqrt{2}]$, and the other is 0.1 with an eigenvector $[1/\sqrt{2}, -1/\sqrt{2}]$. Suppose two random variable X and Y have a standard normal distribution and are linearly independent. Then, from problem 2.1, a new random variable
$Z = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & \text{-}1/\sqrt{2} \end{bmatrix} \begin{bmatrix} \sqrt{1.9} & 0 \\ 0 & \sqrt{0.1} \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}$ has 0 as its mean and Σ as its covariance.
Here is my python code for generating a multivariate random variable with the mean and covariance given.

```
import numpy as np
import matplotlib.pyplot as plt

#fix the random seed
np.random.seed(1337)

# generate two normal random variable, which are independent
X = np.random.standard_normal([2, 10000]);


E = np.array([[np.sqrt(1.9), 0], [0, np.sqrt(0.1)]])
P = np.array([[1/np.sqrt(2), 1/np.sqrt(2)], [1/np.sqrt(2), −1/np.sqrt(2)]])

# transform the variables
Z = np.matmul(E, X)
Z = np.matmul(P, Z)

# generate the multivariate random variable withe the given mean and covariance
Y1, Y2 = np.random.multivariate_normal([0, 0], [[1, 0.9],[0.9, 1]], 10000).T


# draw scatter plots
fig = plt.figure()

plt1 = fig.add_subplot(1, 3, 1)
plt2 = fig.add_subplot(1, 3, 2)
plt3 = fig.add_subplot(1, 3, 3)

plt1.scatter(X[0, :], X[1, :], s=0.1)
plt1.set_title("standard_normal")

plt2.scatter(Z[0, :], Z[1, :], s=0.1)
```

```
plt2.set_title("my multivariate")

plt3.scatter(Y1, Y2, s=0.1)
plt3.set_title("numpy's multivariate")

plt.show()
```

And, here is the result of the code.

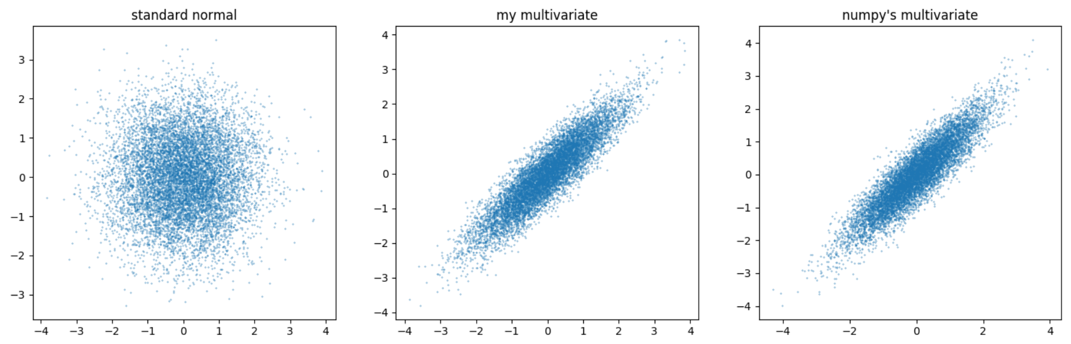

Figure 1: the scatter plots of the multivariate normal random variables

3 Learning a binary classifier with gradient descent

We wish to learn a binary classifier $\mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$ with hinge loss. Concretely

$$\text{minimize } \frac{1}{n}\sum_{i=1}^{n} max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i) + \frac{\lambda}{2}\|\mathbf{w}\|^2 \tag{7}$$

1. Derive the gradient of the loss function.

sol) Let $f_i(\mathbf{w}) = max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)$.

i) $y_i\mathbf{w}^T\mathbf{x}_i >= 1 : f_i(\mathbf{w}) = 0$. So its gradient is $[0, 0, ..., 0]$.

ii) $y_i\mathbf{w}^T\mathbf{x}_i < 1 : f_i(\mathbf{w}) = 1 - y_i\mathbf{w}^T\mathbf{x}_i$. So its gradient is $-y_i\mathbf{x_i} = -y_i[x_{i1}, x_{i2}, x_{i3}, ..., x_{id}]$ where x_{ij} is the jth componet of $\mathbf{x}_i$. Therefore,

$$\nabla f_i(\mathbf{w}) = \begin{cases} 0 \quad \text{if} \quad y_i\mathbf{w}^T\mathbf{x}_i >= 1 \\ -y_i\mathbf{x_i} \quad \text{if} \quad y_i\mathbf{w}^T\mathbf{x}_i < 1 \end{cases} \tag{8}$$

Now, we have to find the gradient of $\frac{\lambda}{2}\|\mathbf{w}\|^2$. Since $\|\mathbf{w}\|^2 = \sum_i w_i^2$, where w_i is ith entry of $\mathbf{w}$, the gradient of $\frac{\lambda}{2}\|\mathbf{w}\|^2$ is $\lambda\mathbf{w}$
Therefore the gradient of the loss function is as following.

$$\nabla l(\mathbf{w}) = \begin{cases} \lambda\mathbf{w} \quad \text{if} \quad y_i\mathbf{w}^T\mathbf{x}_i >= 1 \\ -\frac{1}{n}\sum_{i=1}^{n} y_i\mathbf{x_i} + \lambda\mathbf{w} \quad \text{if} \quad y_i\mathbf{w}^T\mathbf{x}_i < 1 \end{cases} \tag{9}$$

Homework #(**2**)
Seungyong Moon

---

2. Use $\lambda = 0.1, n = 1000, d = 100$, and use fixed step size of 0.01. Also, use the following code to generate the data set $X = [\mathbf{x}_1, ..., \mathbf{x}_n]^T \in \mathbb{R}^{n \times b}$, the label vector $\mathbf{y} = [y_1, ..., y_n] \in \mathbb{R}^n$, and the initial value $\mathbf{w}^{(0)} \in \mathbb{R}^d$

```
X = np.vstack([np.random.normal(0.1, 1, (n//2, d)), np.random.normal(-0.1, 1,
(n//2, d))]
y = np.hstack([np.ones(n//2), -1.*np.ones(n//2)])
w0 = np.random.normal(0, 1, d)
```

Then, solve the optimization problem of finding the optimal separating hyperplane $\mathbf{w}^*$ with gradient descent. Attach the a) source code, b) iteration vs function value plot, and c) iteration vs classification accuracy plot. Accuracy is defined by the fraction of data points your prediction $y_{prediction} =$ sign($\mathbf{w}|\mathbf{x}_i$) matches the label y_i .

sol)
Here is my code.

```
import numpy as np
import matplotlib.pyplot as plt

# set hyperparameter
coeff = 0.1
n = 1000
d = 100
step_size = 0.01
epsilon = 1.0E-6

# fix the random seed
np.random.seed(1337)

# make training data
X = np.vstack([np.random.normal(0.1, 1, (n//2, d)),
               np.random.normal(-0.1, 1, (n//2, d))])
Y = np.hstack([np.ones(n//2), -1*np.ones(n//2)])

# initialise parameter
w0 = np.random.normal(0, 1, d)

fnt_val = []
accuracy  = []
loss_prev = 0

while True:
    correct = 0
    grad = np.zeros(d)
    loss = 0

    # calculate the gradient and the value of the loss function
    for i in range(n):
        x = X[i, :]
        y = Y[i]
        if y*np.matmul(w0, x)<1:
            grad += -y*x
            loss += 1-y*np.matmul(w0, x)
        if y*np.matmul(w0, x)>0:
            correct += 1
    grad /= n
    grad += coeff*w0

    loss += coeff/2*np.matmul(w0, w0)

    #update the parameter
    w0 -= step_size*grad

    print(np.abs(loss-loss_prev))
    #exit condition
    if np.abs(loss-loss_prev)<epsilon:
        break

    fnt_val.append(loss)
    accuracy.append(correct/n*100)
    loss_prev = loss

#draw graphs
fig = plt.figure()

plt1 = fig.add_subplot(1, 2, 1)
plt1.set_title("the value of loss function")
plt1.set_xlabel("num of iteration")
plt1.set_ylabel("val of loss funtion")
plt1.set_ylim(0, 5000)
plt1.set_yticks(np.linspace(0, 5000, 11))

plt2 = fig.add_subplot(1, 2, 2)
plt2.set_title("accuracy")
plt2.set_xlabel("num of iteration")
plt2.set_ylabel("accuracy(%)")
plt2.set_ylim(0, 100)
plt2.set_yticks(np.linspace(0, 100, 11))

plt1.plot(fnt_val)
plt2.plot(accuracy)
plt.show()
```

Here is the plots for loss function value and accuracy per each iteration.

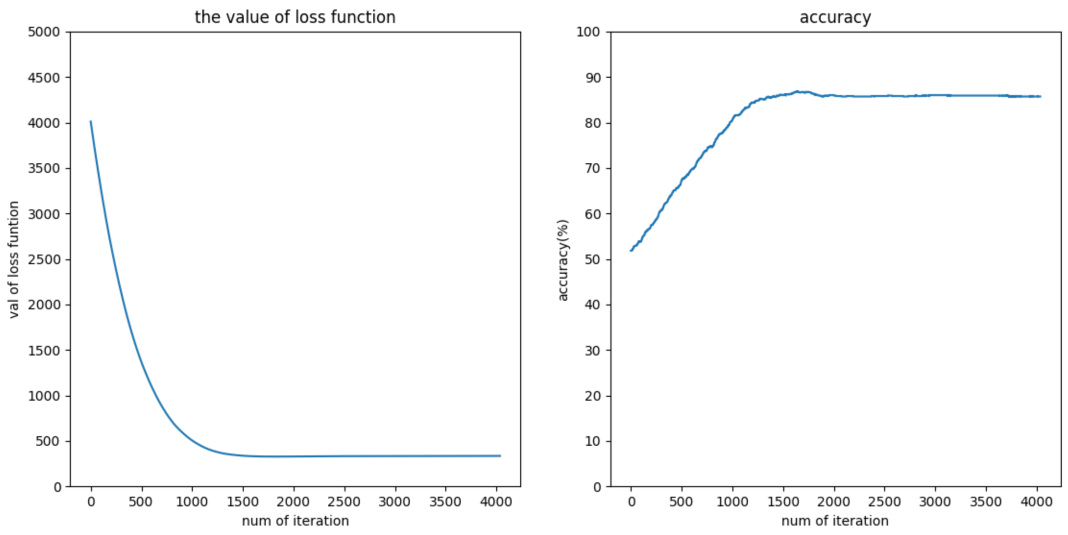

Figure 2: the plots of loss function value and accuracy