EE 526X Homework 02

Jian Xie

jianx@iastate.edu

Problem 1. Let z denote a vector $\mathbf{z}=\left[z_1, z_2, \cdots, z_n\right]^T$. Let $\mathbf{p}=\left[p_1, p_2, \cdots, p_n\right]^T$, such that

$$p_i = \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}$$

That is **p** is the output when the softmax function is applied to **z**. Derive the Jacobian Matrix

$$\frac{\partial \mathbf{p}}{\partial \mathbf{z}} = \left[\frac{\partial p_j}{\partial z_i}\right]_{i,j=1,1}^{n,n}$$

Solution:

If $j=i$:

$$\frac{\partial p_j}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\frac{e^{z_j}}{\sum_{i=1}^{n} e^{z_i}}\right) = \frac{\left(e^{z_j}\right)' \bullet \sum_{i=1}^{n} e^{z_i} - e^{z_j} \bullet e^{z_j}}{\left(\sum_{i=1}^{n} e^{z_i}\right)^2}$$

$$= \frac{e^{z_j}}{\sum_{i=1}^{n} e^{z_i}} - \frac{e^{z_j}}{\sum_{i=1}^{n} e^{z_i}} \bullet \frac{e^{z_j}}{\sum_{i=1}^{n} e^{z_i}}$$

$$= p_j\left(1 - p_j\right)$$

If $j \neq i$:

$$\frac{\partial p_j}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\frac{e^{z_j}}{\sum_{i=1}^{n} e^{z_i}}\right) = \frac{0 \bullet \sum_{i=1}^{n} e^{z_i} - e^{z_j} \bullet e^{z_i}}{\left(\sum_{i=1}^{n} e^{z_i}\right)^2}$$

$$= -\frac{e^{z_j}}{\sum_{i=1}^{n} e^{z_i}} \bullet \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}$$

$$= -p_j p_i$$

Then, Jacobian Matrix

$$\frac{\partial \mathbf{p}}{\partial \mathbf{z}} = \left[\frac{\partial p_j}{\partial z_i}\right]_{i,j=1,1}^{n,n} = \begin{bmatrix} \frac{\partial p_1}{\partial z_1} & \frac{\partial p_1}{\partial z_2} & \cdots & \frac{\partial p_1}{\partial z_n} \\ \frac{\partial p_2}{\partial z_1} & \frac{\partial p_2}{\partial z_2} & \cdots & \frac{\partial p_2}{\partial z_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial p_n}{\partial z_1} & \frac{\partial p_n}{\partial z_2} & \cdots & \frac{\partial p_n}{\partial z_n} \end{bmatrix} = \begin{bmatrix} p_1(1-p_1) & -p_1 p_2 & \cdots & -p_1 p_n \\ -p_1 p_2 & p_2(1-p_2) & \cdots & -p_n p_2 \\ \vdots & \vdots & \ddots & \vdots \\ -p_1 p_n & -p_n p_2 & \cdots & p_n(1-p_n) \end{bmatrix}$$

Problem 2. Let z denote a "logit" vector $\mathbf{z} = [z_1, z_2, \cdots, z_n]^T$ .Let $\mathbf{p} = [p_1, p_2, \cdots, p_n]^T$, such that

$$p_i = \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}$$

Let **y** denote a probability vector $\mathbf{y} = [y_1, y_2, \cdots, y_n]^T$ such that $y_i \geq 0, \forall i \in [1, n]$, and $\sum_{i=1}^{n} y_i = 1$. Let *J* denote the cross entropy between **p** and **y**:

$$J(\mathbf{z}) = -\sum_{i=1}^{n} y_i \log p_i$$

where log is natural logarithm.

(a) Derive the gradient vector

$$\frac{\partial J}{\partial \mathbf{z}} = \begin{bmatrix} \frac{\partial J}{\partial z_1} \\ \frac{\partial J}{\partial z_2} \\ \vdots \\ \frac{\partial J}{\partial z_n} \end{bmatrix}$$

Hint: You can either use the Jacobian, or directly take the gradient. If you take the derivative directly, without using the Jacobian matrix, then it is helpful to write $\log p_i = z_i - \log\left(\sum_{i=1}^{n} e^{z_i}\right)$.

(b) Write a Python class called **crossEntropyLogit** that implements the mapping from **z** to *J*. It should include two methods: **doForward(self, z, y)**, which returns J, and

doBackward(self, y), which returns $\frac{\partial J}{\partial \mathbf{z}}$. You can assume that **doBackward** is always called after forward. The implementation should assume that z and y are matrices, where each column represents one data vector. The code should produce the desired output for the following code snippet:

```
import numpy as np
CE=crossEntropyLogit()
np.random.seed(1)
z=np.random.rand(3,2)
y=np.eye(3)[:,:2]
J1=CE.doForward(z,y) # should be 1.04376...
J2=CE.doForward(z+1000,y) # should be 1.04376...
dz=CE.doBackward(y)
# dz= [[-0.29358105 0.22809874]
[ 0.13604698 -0.34982719]
[ 0.15753407 0.12172845])
```

Solution:

(a)

$$\frac{\partial J}{\partial z_i} = \frac{\partial J}{\partial p_i}\frac{\partial p_i}{\partial z_i} + \sum_{j=1, j\neq i}^{n} \frac{\partial J}{\partial p_j}\frac{\partial p_j}{\partial z_j}$$

From the 1st problem, we have:

$$\frac{\partial p_i}{\partial z_i} = \frac{\partial}{\partial z_i}\frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}} = p_i\left(1 - p_i\right)$$

$$\frac{\partial p_j}{\partial z_i} = -p_j p_i \quad \left(j \neq i\right)$$

Also,

$$\frac{\partial J}{\partial p_i} = -\frac{y_i}{p_i}$$

Putting them together

$$\begin{aligned}\frac{\partial J}{\partial z_i} &= \frac{\partial J}{\partial p_i}\frac{\partial p_i}{\partial z_i} + \sum_{j=1, j\neq i}^{n} \frac{\partial J}{\partial p_j}\frac{\partial p_j}{\partial z_j} \\ &= -\frac{y_i}{p_i} p_i(1-p_i) + \sum_{j=1, j\neq i}^{n} \left[\left(-\frac{y_j}{p_j}\right)\left(-p_j p_i\right)\right] \\ &= -y_i + p_i y_i + \sum_{j=1, j\neq i}^{n} p_i y_j \\ &= -y_i + p_i \sum_{j=1}^{n} y_j \\ &= p_i - y_i\end{aligned}$$

Therefore, the gradient vector

$$\frac{\partial J}{\partial \mathbf{z}} = \begin{bmatrix} \frac{\partial J}{\partial z_1} \\ \frac{\partial J}{\partial z_2} \\ \vdots \\ \frac{\partial J}{\partial z_n} \end{bmatrix} = \begin{bmatrix} p_1 - y_1 \\ p_2 - y_2 \\ \vdots \\ p_n - y_n \end{bmatrix}$$

Problem 3. Let a neural network be such that it has two neurons in one single layer. The neurons has two common inputs. The model can be described as

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

We are given two training points:

(a) When $\mathbf{x} = [1,0]^T$, $\mathbf{y} = [1,0]^T$.

(b) When $\mathbf{x} = [0,1]^T$, $\mathbf{y} = [0,1]^T$.

Note that y, the output has been one-hot encoded. Let **X** and **Y** both be the 2×2 identity matrix, denoting the input and output for the training data. Use softmax on z and use cross-entropy as the cost function, run the forward and backward propagation manually to update **W** and **b** for two iterations, with the following conditions:

(a) Both initialized to all zeros.

(b) Learning rate is $\eta = 1$.

(c) Use gradient descent.

That is, make two updates of **W** and **b** manually. You need to show the intermediate steps (values of **z**, d**z**, **p**, d**W**, d**b**, etc).

Solution:

Forward propagation

$$\mathbf{A}^0 = \mathbf{X} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{W}^1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{b}^1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\mathbf{Z}^1 = \mathbf{W}^1\mathbf{A}^0 + \mathbf{b}^1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Then

$$\mathbf{p}^1 = \mathrm{g}(\mathbf{Z}^1) = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$

Let $J(\mathbf{z}) = -\sum_{i=1}^{n} y_i \log p_i$

Backward propagation

$$d\mathbf{Z}^1 = \frac{\partial J}{\partial \mathbf{p}} g' = \begin{bmatrix} -0.25 & 0.25 \\ 0.25 & -0.25 \end{bmatrix}$$

$$d\mathbf{W}^1 = d\mathbf{Z}^1 \bullet \mathbf{A}^0 = \begin{bmatrix} -0.25 & 0.25 \\ 0.25 & -0.25 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.25 & 0.25 \\ 0.25 & -0.25 \end{bmatrix}$$

$$d\mathbf{b}^1 = d\mathbf{Z}^1 \bullet \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Iteration-first time

$$\mathbf{W}^2 = \mathbf{W}^1 - d\mathbf{W}^1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} -0.25 & 0.25 \\ 0.25 & -0.25 \end{bmatrix} = \begin{bmatrix} 0.25 & -0.25 \\ -0.25 & 0.25 \end{bmatrix}$$

$$\mathbf{b}^2 = \mathbf{b}^1 - d\mathbf{b}^1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\mathbf{Z}^2 = \mathbf{W}^2\mathbf{X} + \mathbf{b}^2 = \begin{bmatrix} 0.25 & -0.25 \\ -0.25 & 0.25 \end{bmatrix}$$

Then

$$\mathbf{p}^2 = \mathrm{g}(\mathbf{Z}^2) = \begin{bmatrix} 0.6225 & 0.3775 \\ 0.3775 & 0.6225 \end{bmatrix}$$

$$d\mathbf{Z}^2 = \frac{\partial J}{\partial \mathbf{p}^2} g' = \begin{bmatrix} -0.18877033 & 0.18877033 \\ 0.18877033 & -0.18877033 \end{bmatrix}$$

$$d\mathbf{W}^2 = d\mathbf{Z}^2 \bullet \mathbf{A}^0 = \begin{bmatrix} -0.18877033 & 0.18877033 \\ 0.18877033 & -0.18877033 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.18877033 & 0.18877033 \\ 0.18877033 & -0.18877033 \end{bmatrix}$$

$$d\mathbf{b}^2 = d\mathbf{Z}^2 \bullet \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Iteration-second time

$$\mathbf{W}^3 = \mathbf{W}^2 - d\mathbf{W}^2 = \begin{bmatrix} 0.43877033 & -0.43877033 \\ -0.43877033 & 0.43877033 \end{bmatrix}$$

$$\mathbf{b}^3 = \mathbf{b}^2 - d\mathbf{b}^2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Problem 4. Download the provided programs, **Data.py**, **NeuralNetwork.py**, and **test_MNIST.py**. Fill in the missing code in **NeuralNetwork.py** at places marked by ...., so that the program test_MNIST.py can run correctly. Experiment with doing classification with the MNIST data set, using the following settings:

(a) Single layer, 10 neurons, softmax + cross entropy objective function.

(b) Three layers, [(50, ReLU), (50, ReLU), (10, Linear)], softmax + cross entropy objective function. (c) Experiment with other neuron settings, with no more than 3 layers, and no more than 150 neurons in total.

Solution: In setting (a), layers=[(10, Linear)], in setting (b), layers=[(50, ReLU), (50, ReLU), (10, Linear)] and in setting (c), there are three layers, and the total neurons are

150. Layers=[(90, ReLU), (50, ReLU), (10, Linear)].

The curve of cross entropy results, test error and train error in three settings are shown in Fig. (a)-Fig. (c).

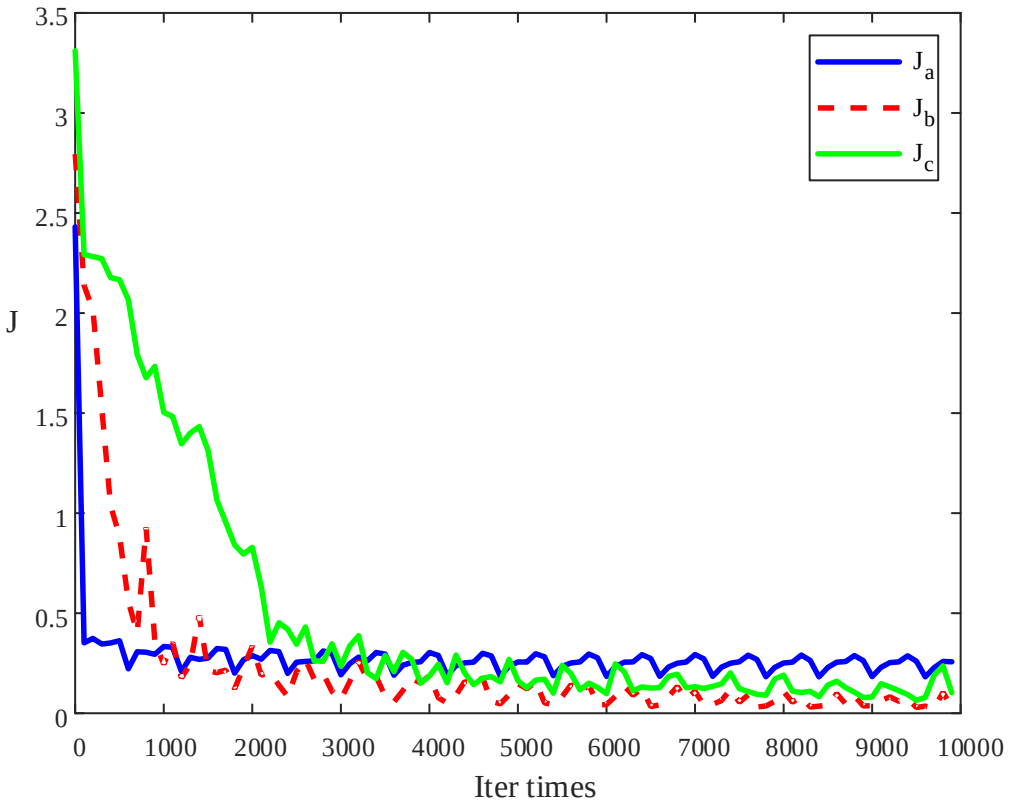

Fig. (a) cross entropy curves in three settings

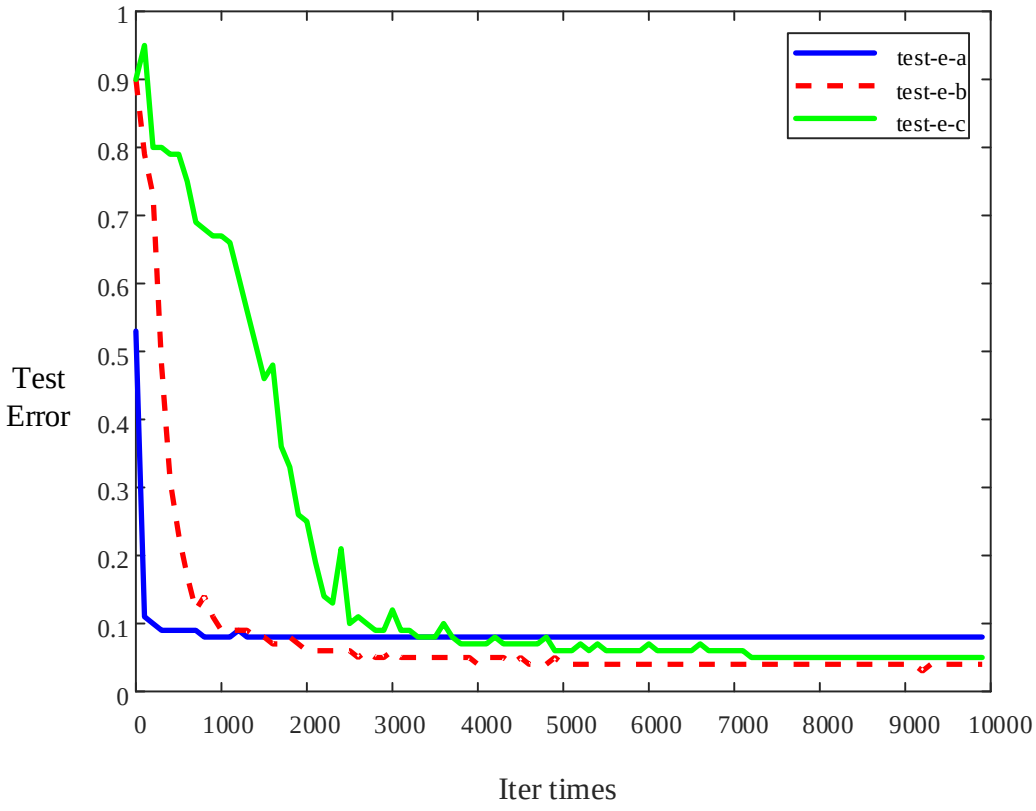

Fig. (b) Test error curves in three settings

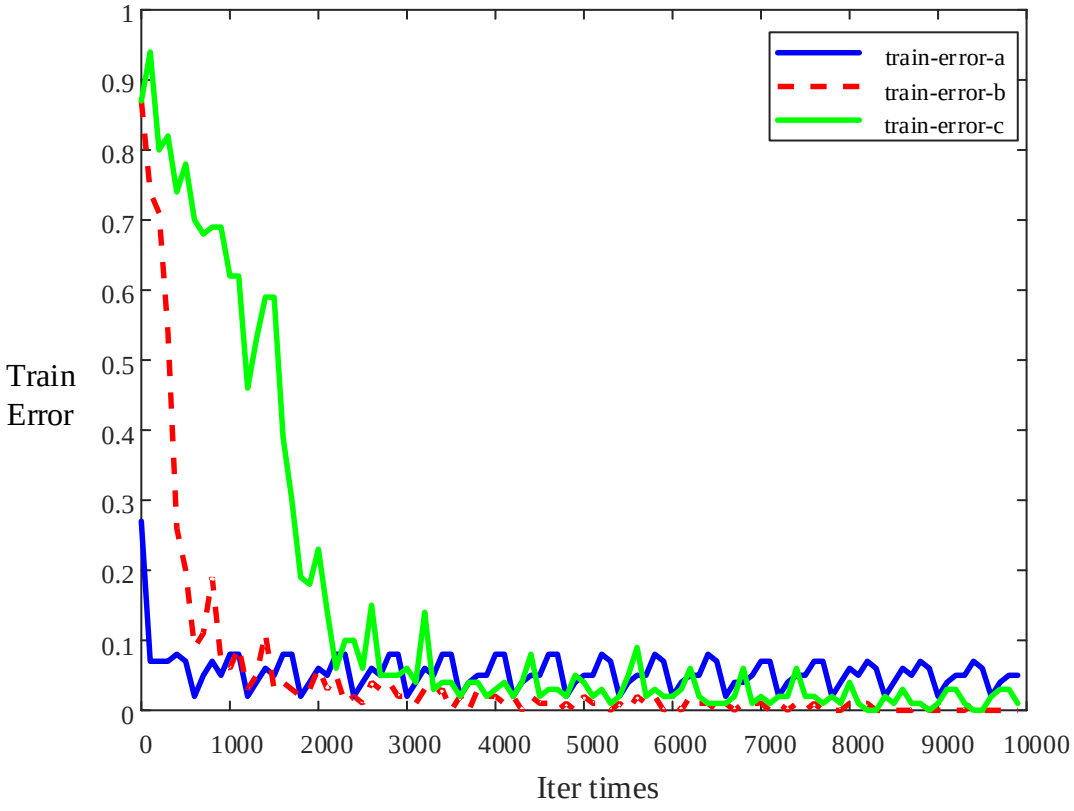

Fig. (c) Train Accuracy curves in three settings

From figure (a), we can see that cross entropy results in setting (a) drop faster than others and cross entropy results in setting (c) drop slower than others. When iter time is close to 5000, the cross entropy results in all three settings are close to 0.15.

From figure (b), we can see that the initial test error in setting (a) is much smaller than others. Also, test errors in setting (a) drop faster than others and cross entropy results in setting (c) drop slower than others. When iter time is close to 3000, the test errors in all three settings are close to 0.08.

From figure (c), we can see that the initial train error in setting (a) is much smaller than others. Also, the train errors in setting (a) drop faster than others and cross entropy results in setting (c) drop slower than others. When iter time is close to 4000, the test errors in all three settings are close to 0.05.