CSE 546 Homework #2 -B

Lu Yu

May 13, 2020

Convexity and Norms

B.1 *[6 points]* For any $x \in \mathbb{R}^n$, show that $\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1$.

$$||x||_2^2 = \sum_{i=1}^{N} |x_i|^2 \leq \left(\sum_{i=1}^{N} |x_i|^2 + 2 * \sum_{i,j,i \neq j} |x_i||x_j| \right) = ||x||_1^2$$

$$||x||_2^2 = \sum_{i=1}^{N} |x_i|^2 = \sum_{i \neq arg\max_i |x_i|} |x_i|^2 + \max_i(|x_i|^2) = \sum_{i \neq arg\max_i |x_i|} |x_i|^2 + \|x\|_\infty^2 \geq \|x\|_\infty^2$$

Hence,

$$\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1$$

B2. Use just the definitions above and let $\| \cdot \|$ be a norm.

a. *[3 points]* Show that $f(x) = \|x\|$ is a convex function.
The Definition of convex is:

$$\forall v, w \in V, \lambda \in [0,1] : f(\lambda v + (1-\lambda)w) \leq \lambda f(v) + (1-\lambda) f(w)$$

$\forall v \in V, \lambda \in \mathbb{R} : |\lambda| \|v\| = \|\lambda v\|$ and using the triangle inequality:

$$\|\lambda v + (1-\lambda)w\| \leq \|\lambda v\| + \|(1-\lambda)w\| = \lambda\|v\| + (1-\lambda)\|w\|$$

b. *[3 points]* Show that $\{x \in \mathbb{R}^N : \|x\| \leq 1\}$ is a convex set.
The Definition of convex set is:

$$\forall v, w \in K, \lambda \in [0,1] : \lambda v + (1-\lambda)w \in K$$

Using the triangle inequality:

$$\|\lambda v + (1-\lambda)w\| \leq \|\lambda v\| + \|(1-\lambda)w\| = \lambda\|v\| + (1-\lambda)\|w\| \leq \lambda + (1-\lambda) = 1$$

Therefore, $\lambda v + (1-\lambda)w \in K$

c. *[2 points]* The defined set is not convex.
For point a: $(x_1, x_2) = (0, -4) \in defined\ set$, and point b: $(x_1, x_2) = (-4, 0) \in defined\ set$, $t = 0.5 \in [0,1]$

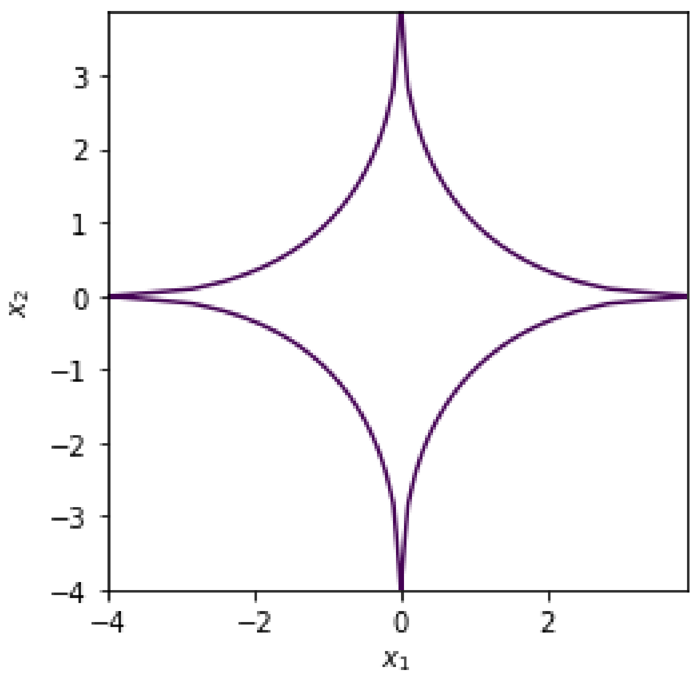
$$ta + (1-t)b = (-2, -2)$$
$$g(-2,-2) = (\sqrt{2} + \sqrt{2})^2 = 8 > 4$$

This implits $g(ta + (1-t)b) \notin defined\ set$, so the defined set is not convex.

[1]:

```python
#B2 c
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

x = y = np.arange(-4, 4, 0.1)
x, y = np.meshgrid(x,y)
plt.contour(x, y, (abs(x)**(1/2) + abs(y)**(1/2))**2, [4])
plt.axis('scaled')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.show()
```



B3.

a. *[3 points]* Show that $\sum_{i=1}^{n} l_i(w) + \lambda \|w\|$ is convex over $w \in \mathbb{R}^d$
if f, g are convex functions, $t \in [0,1]$, by definition,

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

$$g(tx_1 + (1-t)x_2) \leq tg(x_1) + (1-t)g(x_2)$$

$$\begin{aligned}(f+g)(tx_1 + (1-t)x_2) &= f(tx_1 + (1-t)x_2) + g(tx_1 + (1-t)x_2) \\ &\leq tf(x_1) + (1-t)f(x_2) + tg(x_1) + (1-t)g(x_2) \\ &= t(f(x_1) + g(x_1)) + (1-t)(f(x_2) + g(x_2)) \\ &= t((f+g)(x_1) + (1-t)(f+g)(x_2)\end{aligned}$$

Therefore, $f(x) + g(x)$ is also convex.
We know $l_i(w)$ are convex functions, so the $\sum_{i=1}^{n} l_i(w)$is also convex.
From B2 part a, we know $f(w) = \|w\|$ is a convex function and $\forall w \in W, \lambda \in \mathbb{R} : |\lambda|\|w\| = \|\lambda w\|$.
Thus, $\sum_{i=1}^{n} l_i(w) + \lambda\|w\|$ is convex over $w \in \mathbb{R}^d$

b. *[1 points]* Explain in one sentence why we prefer to use loss functions and regularized loss functions that are convex.
Because a local minimum of a convex function is a global minimum so that we can use a local optimization algorithmnto find the best parameters globally.

Multinomial Logistic Regression

B.4

a. *[5 points]*

$$\begin{aligned}
Ł(W) &= -\sum_{i=1}^{n}\sum_{l=1}^{k} \mathbf{1}\{y_i = l\} log(\mathbb{P}_W(y_i = l|W, \mathbf{x}_i)) \\
&= \sum_{i=1}^{n}\{-\sum_{l=1}^{k} \mathbf{1}\{y_i = l\}(\mathbf{w}^{(l)} \cdot \mathbf{x}_i - log\sum_{j=1}^{k}(exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i)))\}
\end{aligned}$$

$$L_i(W) = -\sum_{l=1}^{k} \mathbf{1}\{y_i = l\}(\mathbf{w}^{(l)} \cdot \mathbf{x}_i - log\sum_{j=1}^{k}(exp(\mathbf{w}^{(j)} \cdot \mathbf{x}_i)))$$

Let $\mathbf{v}_{i,j} = \mathbf{w}^{(j)} \cdot \mathbf{x}_i$. For each $\mathbf{w}^{(l)}$ in W,

$$\begin{aligned}
\bigtriangledown_{\mathbf{w}^{(l)}} Ł(W) &= \sum_{i=1}^{n}\sum_{j=1}^{k} \frac{\partial L_i(W)}{\partial \mathbf{v}_{i,j}} \frac{\partial \mathbf{v}_{i,j}}{\partial \mathbf{w}^{(l)}} \\
&= \sum_{i=1}^{n} \frac{\partial L_i(W)}{\partial \mathbf{v}_{i,l}} \frac{\partial \mathbf{v}_{i,l}}{\partial \mathbf{w}^{(l)}} \\
&= \sum_{i=1}^{n} \frac{\partial L_i(W)}{\partial \mathbf{v}_{i,l}} \mathbf{x}_i \\
&= \sum_{i=1}^{n} -\mathbf{x}_i(\mathbf{1}\{y_i = l\} - \mathbb{P}_W(y_i = l|W, \mathbf{x}_i))
\end{aligned}$$

$$\mathbf{y}_i = [\mathbf{1}\{y_i = 1\}, \ldots, \mathbf{1}\{y_i = k\}]^T$$

$$\hat{\mathbf{y}}_i^{(W)} = [\mathbb{P}_W(y_i = 1|W, \mathbf{x}_i), \ldots, \mathbb{P}_W(y_i = k|W, \mathbf{x}_i)]^T$$

$$\bigtriangledown_W Ł(W) = [\bigtriangledown_{\mathbf{w}^{(1)}} Ł(W), \ldots, \bigtriangledown_{\mathbf{w}^{(k)}} Ł(W)] = -\sum_{i=1}^{n} \mathbf{x}_i(\mathbf{y}_i - \hat{\mathbf{y}}_i^{(W)})^T$$

b. *[5 points]*

$$J(W) = \frac{1}{2}\sum_{i=1}^{n} ||\mathbf{y}_i - W^T\mathbf{x}_i||_2^2 = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{y}_i - W^T\mathbf{x}_i)^T(\mathbf{y}_i - W^T\mathbf{x}_i)$$

$$\nabla_W J(W) = -\sum_{i=1}^{n} \mathbf{x}_i(\mathbf{y}_i - W^T\mathbf{x}_i)^T = -\sum_{i=1}^{n} \mathbf{x}_i(\mathbf{y}_i - \tilde{\mathbf{y}}_i^{(W)})^T$$

c. *[15 points]*

[2]:
```
"""
Created on Sat May  9 22:49:19 2020
CSE 546 HW2 B4
@author: Leah
"""
import numpy as np
from mnist import MNIST
import torch
import torch.nn.functional as F #softmax

def load_dataset():
    mndata = MNIST('./data/')
    X_train, labels_train = map(np.array, mndata.load_training())
    X_test, labels_test = map(np.array, mndata.load_testing())
    X_train = X_train/255.0
    X_test = X_test/255.0

    return X_train, labels_train, X_test, labels_test

#LOAD data
X_train, y_train, X_test, y_test = load_dataset()
```

[3]:
```
def onehot(X):
    '''
    For each yi let yi be the one-hot encoding of yi (i.e., yi ∈ {0, 1}^k is a
    vector of all zeros aside from a 1 in the yith index).
    k = 10
    '''
    n_classes = 10
    ## transform labels_train(n-by-1) into n-by-k
    Y = np.zeros((len(X),n_classes))
    for i in range(0,len(Y)):
        Y[i,X[i]] = 1
    return Y
```

[4]:
```
# ============================================================================
# main
# ============================================================================

X_train = torch.tensor(X_train)
X_test = torch.tensor(X_test)
y_test = torch.tensor(y_test)
y_train = torch.tensor(y_train)
y_train = y_train.long()
```

[8]:
```python
# ============================================================================
# J(W)
# ============================================================================
step_size = 0.0000001
epochs = 5000
W = torch.zeros(784, 10, dtype= torch.double, requires_grad=True)
y_train_onehot = torch.tensor (onehot ( y_train ))
for epoch in range(epochs):
    y_hat = torch.matmul(X_train, W)
    # cross entropy combines softmax calculation with NLLLoss
    #0.5*torch.sum(torch.norm(y_tensor-y_hat,dim=1))**2/n
    loss = 0.5 * torch.mean(torch.norm(y_train_onehot - y_hat, p='fro') ** 2 )
    #loss = torch.nn.functional.cross_entropy(y_hat, y_train)
    # computes derivatives of the loss with respect to W
    loss.backward()
    #train_losses.append(loss.item())
    # gradient descent update
    W.data = W.data - step_size * W.grad
    # .backward() accumulates gradients into W.grad instead
    # of overwriting, so we need to zero out the weights
    W.grad.zero_()

print('After runing MSE for',str(epochs), 'times, the classification accuracy on␣
↪the training sets : ',
      sum(torch.argmax(torch.matmul(X_train, W),dim = 1) == y_train).numpy() /␣
↪len(y_train))
print('The classification accuracy on the test sets : ',
      sum(torch.argmax(torch.matmul(X_test, W),dim = 1) == y_test).numpy() /␣
↪len(y_test))
```

```
After runing MSE for 5000 times, the classification accuracy on the training
sets :  0.8507
The classification accuracy on the test sets :  0.8576
```

[7]:
```python
# ============================================================================
# L(W)
# ============================================================================
W = torch.zeros(784, 10, dtype= torch.double, requires_grad=True)
epochs = 2500
step_size = 0.05

for epoch in range(epochs):
    y_hat = torch.matmul(X_train, W)
    # cross entropy combines softmax calculation with NLLLoss
    loss = torch.nn.functional.cross_entropy(y_hat, y_train)
    # computes derivatives of the loss with respect to W
    loss.backward()
```

```
    #train_losses.append(loss.item())
    # gradient descent update
    W.data = W.data - step_size * W.grad
    # .backward() accumulates gradients into W.grad instead
    # of overwriting, so we need to zero out the weights
    W.grad.zero_()

print('After runing cross-entropy loss function for',str(epochs), 'times, the␣
 ↪classification accuracy on the training sets : ',
      sum(torch.argmax(torch.matmul(X_train, W),dim = 1) == y_train).numpy() /␣
 ↪len(y_train))
print('The classification accuracy on the test sets : ',
      sum(torch.argmax(torch.matmul(X_test, W),dim = 1) == y_test).numpy() /␣
 ↪len(y_test))
```

```
After runing cross-entropy loss function for 2500 times, the classification
accuracy on the training sets :  0.9059166666666667
The classification accuracy on the test sets :  0.9107
```