MathDNN Homework 3

Department of Computer Science and Engineering
2021-16988 Jaewan Park

Problem 3

(a) Since $\exp(f_y) > 0$, $0 < \dfrac{\exp(f_y)}{\sum_{j=1}^{k} \exp(f_j)} < 1$, therefore $0 < \ell^{\mathrm{CE}}(f, y) = -\log\left(\dfrac{\exp(f_y)}{\sum_{j=1}^{k} \exp(f_j)}\right) < \infty$.

(b) Since

$$\ell^{\mathrm{CE}}(\lambda_y, y) = -\log\left(\frac{\exp\left((\lambda e_y)_y\right)}{\sum_{j=1}^{k} \exp\left((\lambda e_y)_j\right)}\right) = -\log\left(\frac{\exp(\lambda)}{\exp(\lambda) + \exp(0) \times (k-1)}\right)$$

we get $\ell^{\mathrm{CE}}(\lambda_y, y) \to 0$ when $\lambda \to \infty$.

Problem 4

Suppose a specific value $x = x^*$ is given. Then $I^* = \mathrm{argmax}_i\{f_i(x^*)\}$ uniquely exists, therefore $\forall i,\ f_{I^*}(x^*) > f_i(x^*)$. Let $g_i(x) = f_{I^*}(x) - f_i(x)$, then $g_i(x^*) > 0$. Since all f_i are continuous, g_i are also continuous, so there exists $\delta_i > 0$ such that

$$x \in N(x^*, \delta_i) \Rightarrow g_i(x) = f_{I^*}(x) - f_i(x) > 0.$$

Therefore for $x \in N(x^*, \delta_i)$, $f(x) = f_{I^*}(x)$, and since f_{I^*} is differentiable, $f'(x) = f'_{I^*}(x)$ for x in this neighborhood. $\therefore f'(x^*) = f'_{I^*}(x^*)$.

Problem 5

(a) The function is idempotent.

$$\sigma(\sigma(z)) = \begin{cases} \sigma(z) & (z \geq 0) \\ \sigma(0) & (z < 0) \end{cases} = \begin{cases} z & (z \geq 0) \\ 0 & (z < 0) \end{cases} = \sigma(z)$$

(b) The derivative of softplus is $\sigma'(z) = \dfrac{e^z}{1 + e^z}$, therefore for all $z_1, z_2 \in \mathbb{R}$,

$$\begin{aligned} |\sigma'(z_1) - \sigma'(z_2)| &= \left|\frac{e^{z_1}}{1+e^{z_1}} - \frac{e^{z_2}}{1+e^{z_2}}\right| = \left|\frac{1}{1 + e^{z_1} + e^{z_2} + e^{z_1+z_2}}\right| \left|\frac{e^{z_1} - e^{z_2}}{z_1 - z_2}\right| |z_1 - z_2| \\ &= \left|\frac{e^{z}}{1 + e^{z_1} + e^{z_2} + e^{z_1+z_2}}\right| |z_1 - z_2| \ \ (\exists z \in (z_1, z_2) \ \ \because \text{MVT}) \\ &\leq |z_1 - z_2| \ \ \left(\because e^z < \max\left\{e^{z_1}, e^{z_2}\right\} < 1 + e^{z_1} + e^{z_2} + e^{z_1+z_2}\right) \end{aligned}$$

so softplus has Lipschitz continuous derivatives. In the case of ReLU, since the derivative is

$$\sigma'(z) = \begin{cases} 1 & (z \geq 0) \\ 0 & (z < 0) \end{cases}$$

if we choose $z_1 = \epsilon$ and $z_2 = -\epsilon$ for an arbitrary $\epsilon > 0$,

$$|\sigma'(z_1) - \sigma'(z_2)| = |1 - 0| = 1, \;\; |z_1 - z_2| = 2\epsilon$$

so it cannot be bounded by a fixed value L such that $|\sigma'(z_1) - \sigma'(z_2)| \leq L|z_1 - z_2|$. Therefore ReLU does not have Lipschitz continuous derivatives.

(c) We can obtain the follwing.

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{2}\rho\left(\frac{1}{2}z\right) + \frac{1}{2}, \;\; \rho(z) = \frac{1-e^{-2z}}{1+e^{-2z}} = 2\sigma(2z) - 1$$

We should determine $C_1, \cdots, C_L$ and $d_1, \cdots, d_L$ such that considering the following mappings,

$$\begin{aligned} y_L &= A_L y_{L-1} + b_L & \qquad y'_L &= C_L y'_{L-1} + d_L \\ y_{L-1} &= \sigma(A_{L-1} y_{L-2} + b_{l-1}) & \qquad y'_{L-1} &= \rho\big(C_{L-1} y'_{L-2} + d_{l-1}\big) \\ &\vdots & \qquad &\vdots \\ y_1 &= \sigma(A_1 x + b_1) & \qquad y'_1 &= \rho(C_1 x + d_1) \end{aligned}$$

$y_L = y'_L$ with the same input x. Now choose $C_1 = \frac{1}{2}A_1$ and $d_1 = \frac{1}{2}b_1$, then

$$y'_1 = \rho\left(\frac{1}{2}A_1 x + \frac{1}{2}b_1\right) = 2\sigma(A_1 x + b_1) - 1 = 2y_1 - 1.$$

In the next step, choose $C_2 = \frac{1}{4}A_2$ and $d_2 = \frac{1}{2}b_2 + \frac{1}{4}A_2$, then

$$y'_2 = \rho\left(\frac{1}{4}A_2(2y_1 - 1) + \frac{1}{2}b_2 + \frac{1}{4}A_2\right) = \rho\left(\frac{1}{2}A_2 y_1 + \frac{1}{2}b_2\right) = 2\sigma(A_2 y_1 + b_2) - 1 = 2y_2 - 1.$$

Now continuously choose $C_i = \frac{1}{4}A_i$ and $d_i = \frac{1}{2}b_i + \frac{1}{4}A_i$ for $i = 2, 3, \cdots, L-1$, then

$$y'_{L-1} = 2y_{L-1} - 1.$$

Finally choose $C_L = \frac{1}{2}A_L$ and $d_L = b_L + \frac{1}{2}A_L$, then

$$y'_L = \frac{1}{2}A_L(2y_{L-1} - 1) + b_L + \frac{1}{2}A_L = A_L y_{L-1} + b_L = y_L.$$

Therefore σ and ρ are equivalent.

Problem 6

The gradient of the minimizing function is the following. (Calculation partially brought from HW2)

$$\begin{aligned}\nabla_\theta \ell(f_\theta(X_i), Y_i) &= \ell'(f_\theta(X_i), Y_i)\nabla_\theta f_\theta(X_i) \\ &= \ell'(f_\theta(X_i), Y_i)\Big((\sigma'(aX_i+b)\odot u)X_i,\ \sigma'(aX_i+b)\odot u,\ \sigma(aX_i+b)\Big)\end{aligned}$$

Since $\sigma(z) = \sigma'(z) = 0$ if $z < 0$ and $a_j^0 X_i + b_j^0 < 0$ for all i,

$$\begin{aligned}\Big[\big(\sigma'\big(a^0X_i+b^0\big)\odot u^0\big)X_i\Big]_j &= \sigma'\big(a_j^0X_i+b_j^0\big)u_j^0X_i = 0 \\ \Big[\sigma'\big(a^0X_i+b^0\big)\odot u^0\Big]_j &= \sigma'\big(a_j^0X_i+b_j^0\big)u_j^0 = 0 \\ \Big[\sigma\big(a^0X_i+b^0\big)\Big]_j &= \sigma\big(a_j^0X_i+b_j^0\big) = 0\end{aligned}$$

Therefore the gradients for the j-th outputs are all 0, so there is no change from a_j^0, b_j^0, u_j^0 to a_j^1, b_j^1, u_j^1. Consequently $a_j^1 X_i + b_j^1 < 0$, and continuously the condition maintains throughout the training. Therefore the j-th ReLU output remains dead throughout the training.

Problem 7

The derivative of the leaky ReLU function is

$$\sigma'(z) = \begin{cases} 1 & (z \geq 0) \\ \alpha & (z < 0). \end{cases}$$

Then going through the same progress from **Problem 6**,

$$\begin{aligned}\Big[\big(\sigma'\big(a^0X_i+b^0\big)\odot u^0\big)X_i\Big]_j &= \sigma'\big(a_j^0X_i+b_j^0\big)u_j^0X_i = \alpha u_j^0X_i \neq \mathbf{0} \\ \Big[\sigma'\big(a^0X_i+b^0\big)\odot u^0\Big]_j &= \sigma'\big(a_j^0X_i+b_j^0\big)u_j^0 = \alpha u_j^0 \neq \mathbf{0} \\ \Big[\sigma\big(a^0X_i+b^0\big)\Big]_j &= \sigma\big(a_j^0X_i+b_j^0\big) = \alpha\big(a_j^0X_i+b_j^0\big) \neq \mathbf{0}\end{aligned}$$

we obtain that the gradients for the j-th outputs are not exactly 0. Therefore a decrement in a_j, b_j, u_j occurs, so the training successfully works and does not go dead.

Problem 1

[1]:
```
import torch
import numpy as np
from torch import nn, optim
from torch.nn import functional as F
from torch.utils.data import TensorDataset, DataLoader
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

[2]:
```
# Prepare dataset

alpha = 0.1
K = 1000
B = 128
N = 512

def f_true(x) :
    return (x-2) * np.cos(x*4)

torch.manual_seed(0)
X_train = torch.normal(0.0, 1.0, (N,))
y_train = f_true(X_train)
X_val = torch.normal(0.0, 1.0, (N//5,))
y_val = f_true(X_val)

train_dataloader = DataLoader(TensorDataset(X_train.unsqueeze(1), y_train.
 ↪unsqueeze(1)), batch_size=B)
test_dataloader = DataLoader(TensorDataset(X_val.unsqueeze(1), y_val.unsqueeze(1)),␣
 ↪batch_size=B)
```

[3]:
```
# Define the Neural Network Class

n0, n1, n2, n3 = 1, 64, 64, 1

class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.l1 = nn.Linear(n0, n1, bias=True)
        self.l2 = nn.Linear(n1, n2, bias=True)
        self.l3 = nn.Linear(n2, n3, bias=True)

    def forward(self, x):
        x = x.float().view(-1, 1)
        x1 = torch.sigmoid(self.l1(x))
        x2 = torch.sigmoid(self.l2(x1))
        x3 = self.l3(x2)
        return x3
```

[4]:
```
# Create Model, Initialize Weights, Specify Loss Function and Optimizer

model = MLP()
```

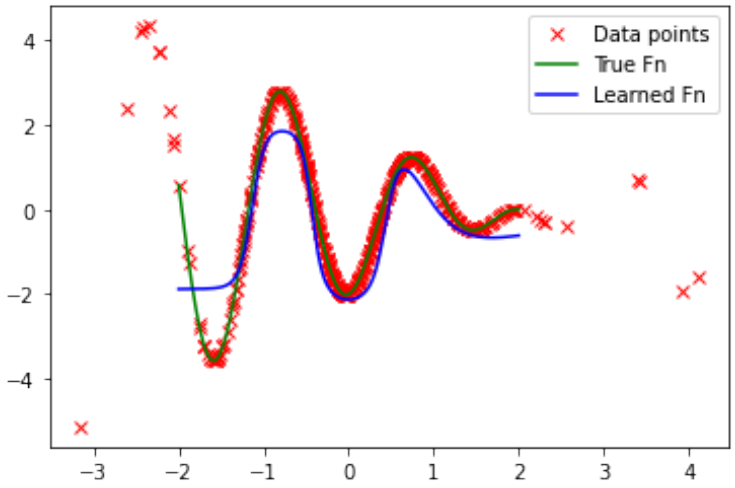
```
model.l1.weight.data = torch.normal(0, 1, model.l1.weight.shape)
model.l1.bias.data = torch.full(model.l1.bias.shape , 0.03)
model.l2.weight.data = torch.normal(0, 1, model.l2.weight.shape)
model.l2.bias.data = torch.full(model.l2.bias.shape , 0.03)
model.l3.weight.data = torch.normal(0, 1, model.l3.weight.shape)
model.l3.bias.data = torch.full(model.l3.bias.shape , 0.03)

loss_function = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=alpha)
```

[5]:
```
# Train Model with Shuffled Cyclic SGD

for _ in range(K):
    for x, y in train_dataloader:
        optimizer.zero_grad()
        train_loss = loss_function(model(x), y)
        train_loss.backward()
        optimizer.step()
```

[6]:
```
# Test Model

test_loss, correct = 0, 0
misclassified_ind = []
correct_ind = []

for ind in range(len(X_val)) :

    x, y = X_val[ind], y_val[ind]
    output = model(x)
    test_loss += loss_function(output, y.float().view(-1, 1)).item()

    if output.item() * y.item() >= 0 :
        correct += 1
        correct_ind += [ind]
    else:
        misclassified_ind += [ind]

print('[Test set] Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%)\n'.format(
        test_loss /len(X_val.data), correct, len(X_val.data),
        100. * correct / len(X_val.data)))
```

```
[Test set] Average loss: 1.2829, Accuracy: 90/102 (88.24%)
```

[7]:
```
# Plot Results

with torch.no_grad():
    xx = torch.linspace(-2,2,1024).unsqueeze(1)
    plt.plot(X_train,y_train,'rx',label='Data points')
    plt.plot(xx,f_true(xx),'g',label='True Fn')
    plt.plot(xx, model(xx),'b',label='Learned Fn')
plt.legend()
plt.show()
```



Problem 2

[8]:
```
y_train = f_true(X_train) + torch.normal(0, 0.5, X_train.shape)
```

[9]:
```
# Create Model, Initialize Weights, Specify Loss Function and Optimizer

model = MLP()

model.l1.weight.data = torch.normal(0, 1, model.l1.weight.shape)
model.l1.bias.data = torch.full(model.l1.bias.shape , 0.03)
model.l2.weight.data = torch.normal(0, 1, model.l2.weight.shape)
model.l2.bias.data = torch.full(model.l2.bias.shape , 0.03)
model.l3.weight.data = torch.normal(0, 1, model.l3.weight.shape)
model.l3.bias.data = torch.full(model.l3.bias.shape , 0.03)

loss_function = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=alpha)
```

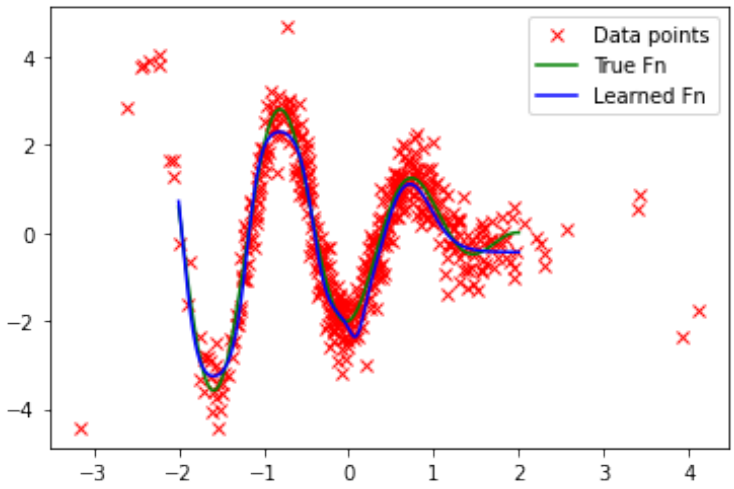
[10]:
```
# Train Model with Shuffled Cyclic SGD

for _ in range(K):
    for x, y in train_dataloader:
        optimizer.zero_grad()
        train_loss = loss_function(model(x), y)
        train_loss.backward()
        optimizer.step()
```

[11]:
```
# Test Model

test_loss, correct = 0, 0
misclassified_ind = []
correct_ind = []
```

```
for ind in range(len(X_val)) :

    x, y = X_val[ind], y_val[ind]
    output = model(x)
    test_loss += loss_function(output, y.float().view(-1, 1)).item()

    if output.item() * y.item() >= 0 :
        correct += 1
        correct_ind += [ind]
    else:
        misclassified_ind += [ind]

print('[Test set] Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%)\n'.format(
        test_loss /len(X_val.data), correct, len(X_val.data),
        100. * correct / len(X_val.data)))
```

```
[Test set] Average loss: 0.1395, Accuracy: 96/102 (94.12%)
```

[12]:
```
# Plot Results

with torch.no_grad():
    xx = torch.linspace(-2,2,1024).unsqueeze(1)
    plt.plot(X_train,y_train,'rx',label='Data points')
    plt.plot(xx,f_true(xx),'g',label='True Fn')
    plt.plot(xx, model(xx),'b',label='Learned Fn')
plt.legend()
plt.show()
```



Problem 2 is different from problem 1 in that the training labels have some random errors. Nevertheless the training result still successfully came out, with an accuracy of 94%. The plot also shows that the result is fairly accurate.

Also since there are two hidden layers in this problem, the number of trainable parameters is

$$(1 \times 64 + 64) + (64 \times 64 + 64) + (64 \times 1 + 1) = 4353.$$