Problem 1. Google Colab pdf의 화면으로 대체한다. 이는 본 pdf의 뒤쪽에서 확인할 수 있다. □

Problem 2. 1번에서와는 다르게 데이터의 값에 오차들이 생긴 상황이다. 즉 주어진 데이터가 현실의 데이터이기 때문에 자연적인 오차가 존재하고 이를 반영하여 함수를 추정하라는 것인데, 이 경우에도 비교적 잘 추정됨을 확인한다. (**Google Colab pdf**의 그래프를 통해 알 수 있다.) 놀라운 사실은 심층신경망에서 학습해야 할 parameter의 개수가 데이터의 개수보다 많음에도 불구하고 과적합(overfitting)이 되지 않는다는 사실이다.

참고로, 1번과 2번 문제에서 학습해야 할 parameter의 개수를 구해보자. 본 문제에서 은닉층(hidden layer)의 개수는 2개이고, 이 과정에서 3개의 affine function(선형사상 및 평행이동사상, 즉 bias)에서 학습해야 할 parameter가 나타난다. 첫 번째 affine mapping에서 $1 \cdot 64 + 64$개가, 두 번째 affine mapping에서 $64 \cdot 64 + 64$개가, 세 번째 affine mapping에서 $64 \cdot 1 + 1$개가 나타나므로, 이들의 합인 4353개의 parameter가 주어지는 것이다. □

Problem 3. (*Cross Entropy Loss*) $f \in \mathbb{R}^k, y \in \{1, 2, ..., k\}$

$$l^{\mathrm{CE}}(f, y) = -\log\left(\frac{\exp(f_y)}{\sum_{j=1}^{k}\exp(f_j)}\right)$$

(a) exp는 항상 양의 실수만을 값으로 가지므로, log 안의 분모는 항상 분모보다 크다. 따라서 log 안의 숫자는 항상 1보다 작은 양수가 되고, 이는 $0 < l^{\mathrm{CE}}(f, y) < \infty$임을 확인한다. □

(b)

$$l^{\mathrm{CE}}(\lambda_y, y) = -\log\left(\frac{\exp(\lambda\delta_{yy})}{\sum_{j=1}^{k}\exp(\lambda\delta_{yj})}\right) = -\log\left(\frac{\exp(\lambda)}{\exp(\lambda) + k - 1}\right)$$

이므로, $\lambda \to \infty$에서 $l^{\mathrm{CE}}(\lambda_y, y) \to 0$이다. □

Problem 4. (*Derivative of max*) $f(x) \coloneqq \max\{f_1(x), ..., f_k(x)\}$, $f_i \in C^1$, 주어진 $x \in \mathbb{R}$에 대해 $I = \mathrm{argmax}_i\{f_i(x)\}$가 유일할 때 $f'(x) = f_I'(x)$이다.

Proof.
주어진 조건에 의해서 $I \neq j$이면 $f_I(x) > f_j(x)$이다. 모든 index set의 원소 i에 대해 f_i가 미분가능한 함수이므로 연속함수이다. $M \coloneqq \sup\{f_1(x), ..., f_k(x)\} \setminus f_I(x)$이라 하면 $f_I(x) > M$이므로 $\epsilon = \frac{1}{2}(f_I(x) - M)$라 하면 f_i의 연속성에 의해 각각 $\delta_i > 0$이 존재해서 x의 δ_i-*neighbourhood*에서는 함숫값이 자신들의 값에서 ϵ 미만으로 멀어진다. $\delta = \min\{\delta_1, ...\delta_k\}$라 하면 x의 δ 근방에서는 여전히 f_I가 다른 함수보다도 큰 함수가 되는 것이다. 이제

$$f'(x) = \lim_{h\to 0}\frac{f(x+h) - f(x)}{h} = \lim_{h\to 0}\frac{f_I(x+h) - f_I(x)}{h} = f_I'(x)$$

이다. □

Problem 5. (*Basic properties of activation functions*)
(a) $\sigma(z) = \max\{0, z\}$ (*ReLU activation*)은 idempotent이다. $z \geq 0$이면 $\sigma(z) = z$이므로 $\sigma(\sigma(z)) = \sigma(z)$이고, $z < 0$이면 $\sigma(z) = 0$이므로 $\sigma(\sigma(z)) = \sigma(0) = 0 = \sigma(z)$이다. □

(b) $\sigma(z) = \log(1 + e^z)$ (*softplus activation*)은 *ReLU activation*을 부드러운 함수로 대체한 것이다. ($z \to -\infty$에서 $\sigma(z) \to 0$이고, $z \to \infty$에서 $\sigma(z) \approx z$) $1 + e^z > 1$이므로 $\sigma : \mathbb{R} \to \mathbb{R}^+$는 잘 정의되고, 또한 미분가능한 함수임을 확인한다. ($\sigma'(z) = \frac{\exp(z)}{1+\exp(z)}$) 그러면 도함수는 $\mathbb{R}$에서 립쉬츠 연속이다. 이 함수의 이계도함수는 $\sigma''(z) = \frac{\exp(z)}{(1+\exp(z))^2}$이다. 따라서 평균값 정리에 의해 임의의 $z_1, z_2 \in \mathbb{R}$에 대해

$$\begin{aligned} |\,\sigma'(z_1) - \sigma'(z_2)\,| &= \frac{e^z}{(1+e^z)^2}\,|\,z_1 - z_2\,| \ (\because \exists z \in (z_1, z_2)) \\ &\leq |\,z_1 - z_2\,| \ (\because \frac{e^z}{(1+e^z)^2} < 1) \end{aligned}$$

이 성립한다. 따라서 *softplus activation*의 도함수는 $\mathbb{R}$에서 립쉬츠 연속이다. 반면, *ReLU activation*은 그렇지 않다. $x \neq 0$에서 정의되는 *ReLU activation*의 도함수의 경우, $z_1 = -\epsilon, z_2 = \epsilon$ ($\epsilon > 0$)으로 두면 평균변화율이 $\frac{1}{2\epsilon}$이 되어 원하는 만큼 키울 수 있기 때문이다. 즉 립쉬츠 연속의 정의처럼 평균변화율이 유계가 아니다. □

(c) *sigmoid*(σ) 활성 함수와 *tanh*(ρ) 활성 함수는 MLP의 관점에서 사실상 같음을 증명하자.

$$\begin{aligned} \tanh z &= \frac{1 - e^{-2z}}{1 + e^{-2z}} = -\frac{-1 - e^{-2z}}{1 + e^{-2z}} + \frac{2}{1 + e^{-2z}} = 2sigmoid(2z) - 1 \\ \sigma(z) &= \frac{1}{2}(\rho(\frac{1}{2}z) + 1) \end{aligned}$$

이제 다음과 같은 전략으로, 두 활성 함수의 동등성을 증명한다.

$$\rho \text{ 안의 인자는 항상 } \sigma \text{ 안의 인자의 } \frac{1}{2}\text{배가 되도록 조정한다.}$$

편의상, 문제에서 오른쪽에 있는 y_i는 y_i'이라고 생각하자. 그러면 위의 수식은 곧 $\frac{1}{2}\sigma^{-1}(y_i) = \rho^{-1}(y_i')$이 되도록 하는 것이다. 다시 정리하면 $\rho(\rho^{-1}(y_i') = y_i' = \rho(\frac{1}{2}\sigma^{-1}(y_i)) = 2y_i - \mathbb{1}$이다.
먼저 $i = 1$일 때에는 $C_1 = \frac{1}{2}A_1, d_1 = \frac{1}{2}b_1$이 되도록 하면 된다. 이제 $i = 2, ..., L-1$일 때에는

$$\begin{aligned} \frac{1}{2}(A_i y_{i-1} + b_i) &= C_i y_{i-1}' + d_i \\ y_{i-1}' &= 2y_{i-1} - \mathbb{1} \end{aligned}$$

이 성립하면 되므로, 정리하면 $C_i = \frac{1}{4}A_i, d_i = \frac{1}{2}b_i + \frac{1}{4}A_i\mathbb{1}$이 되도록 하면 된다.
마지막으로, $i = L$일 때에는 $A_L y_{L-1} + b_L = C_L y_{L-1}' + d_L$이면 충분하고, 이는 $C_L = \frac{1}{2}A_L, d_L = b_L + \frac{1}{2}A_i\mathbb{1}$이다.
역이 성립함은 거의 자명하다. 두 경우 모두 종합하여 정리하면 다음과 같다.

$$C_1 = \frac{1}{2}A_1, d_1 = \frac{1}{2}b_1 \qquad A_1 = 2C_1, b_1 = 2d_1 \qquad i = 1 \tag{1}$$

$$C_i = \frac{1}{4}A_i, d_i = \frac{1}{2}b_i + \frac{1}{4}A_i\mathbb{1} \qquad A_i = 4C_i, b_i = 2d_i - 2C_i\mathbb{1} \qquad i = 2, 3, ..., L-1 \tag{2}$$

$$C_L = \frac{1}{2}A_L, d_L = b_L + \frac{1}{2}A_L\mathbb{1} \qquad A_L = 2C_L, b_L = d_L - C_L\mathbb{1} \qquad i = L \tag{3}$$

따라서, 두 활성함수는 다중 퍼셉트론의 관점에서는 사실상 같은 작용을 한다. □

Problem 6. (*Vanishing gradients*) 벡터 $a, b, u \in \mathbb{R}^p$에 대해 다음과 같은 2층 신경망

$$f_\theta(x) = u^T \sigma(ax + b) = \sum_{j=1}^{p} u_j \sigma(a_j x + b_j)$$

이 있을 때, σ를 *ReLU activation*으로 두자. $X_1, ..., X_N \in \mathbb{R}$과 레이블 $Y_1, ..., Y_N$에 대해 SGD를 이용하여

$$\underset{\theta \in \mathbb{R}^{3p}}{\text{minimise}} \quad \frac{1}{N} \sum_{i=1}^{N} l(f_\theta(X_i), Y_i) \text{ (Assume } \theta = (a, b, u))$$

를 해결하려고 한다. 손실함수 $l(x, y)$는 x에 대해서 미분가능하다고 하자. j번째 *ReLU output*이 초기화시 "dead" 라는 것은 모든 $i = 1, ..., N$에 대해 $a_j^0 X_i + b_j^0 < 0$임을 의미한다고 하자. 그러면 j번째 *ReLU output*은 학습 과정 내내 "dead"가 됨을 증명하여라.

Proof.
SGD는 여러 가지 세부적인 방법이 존재하지만, 그 중 어떤 방법이 채택되더라도 본 문제에는 영향이 없다. 왜냐하면 조건에서 모든 $i = 1, ..., N$에 대해 $a_j^0 X_i + b_j^0 < 0$가 됨을 가정하였기 때문이다. 위의 문제에서 i를 고정하고 손실함수의 경사를 계산해보자.

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

이 때 *ReLU activation*의 도함수는 $x < 0$에서 $\sigma' = 0$, $x \geq 0$에서 $\sigma' = 1$이므로, *ReLU activation*과 *ReLU activation*의 도함수 모두 $x < 0$에서 영함수가 된다. 따라서 $\nabla_\theta \, l(f_\theta(X_i), Y_i)$의 j번째 항(a, b, u 각각)은 모두 경사가 0이다. 이는 a_j^1, b_j^1, u_j^1은 SGD 수행 후 값이 초기값과 같음을 의미한다. 즉, j번째 output은 학습이 진행되지 않는다. 이후의 학습 과정에서도 마찬가지의 이유로 a_j^k, b_j^k, u_j^k (k : iteration)의 값은 변화하지 않으므로 학습이 이루어지지 않는다. □

Problem 7. (*Leaky ReLU*) 앞에서 *ReLU activation* 대신 *Leaky ReLU activation*을 사용한다고 하자. 이 때 *Leaky ReLU activation*은

$$\sigma(z) = \begin{cases} \alpha z & x < 0 \ (\alpha \text{ usually has value of } 0.01) \\ z & x \geq 0 \end{cases}$$

으로 정의된다. 이 때 *ReLU*를 사용했을 때와는 다르게, *Leaky ReLU*를 사용할 경우 위에서와 같이 그레이디언트 소실은 일어나지 않음을 보여라.

Proof.
위에서의 증명을 살펴볼 때 *Leaky ReLU activation*의 도함수는 $x < 0$에서 $\sigma' = \alpha > 0$, $x \geq 0$에서 $\sigma' = 1$이므로 0이 되는 일은 존재하지 않는다. 또한 *Leaky ReLU activation* 자체도 $x = 0$을 제외하면 0의 값을 가지지 않는다. 그러면 6번에서의 증명과는 다르게, 그레이디언트의 a_j, b_j, u_j 성분이 자동적으로 0이 될 수 없다. 즉, 학습은 진행된다. □

MATHDNN_HW3

September 30, 2021

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]:
```
alpha = 0.1
K = 1000
B = 128
N = 512

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

np.random.seed(0)
X_train = torch.tensor(np.random.normal(loc = 0.0, scale = 1.0, size = N),␣
 ↪dtype=torch.float32)
y_train = f_true(X_train)
X_val = torch.tensor(np.random.normal(loc = 0.0, scale = 1.0, size = N//5),␣
 ↪dtype=torch.float32)
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)

'''
unsqueeze(1) reshapes the data into dimension [N,1],
where is 1 the dimension of an data point.

The batchsize of the test dataloader should not affect the test result
so setting batch_size=N may simplify your code.
In practice, however, the batchsize for the training dataloader
```

```
is usually chosen to be as large as possible while not exceeding
the memory size of the GPU. In such cases, it is not possible to
use a larger batchsize for the test dataloader.
'''
```

[2]:
```
'\nunsqueeze(1) reshapes the data into dimension [N,1],\nwhere is 1 the
dimension of an data point.\n\nThe batchsize of the test dataloader should not
affect the test result\nso setting batch_size=N may simplify your code.\nIn
practice, however, the batchsize for the training dataloader\nis usually chosen
to be as large as possible while not exceeding\nthe memory size of the GPU. In
such cases, it is not possible to\nuse a larger batchsize for the test
dataloader.\n'
```

[3]:
```
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(1, 64, bias=True) # 1*64 param + 64 param(bias)
        self.linear2 = nn.Linear(64, 64, bias=True) # 64*64 + 64
        self.linear3 = nn.Linear(64, 1, bias=True) # 1*64 + 1
    def forward(self, x):
        x = x.float().view(-1, 1)
        x = nn.functional.sigmoid(self.linear(x))
        x = nn.functional.sigmoid(self.linear2(x))
        x = self.linear3(x)
        return x
```

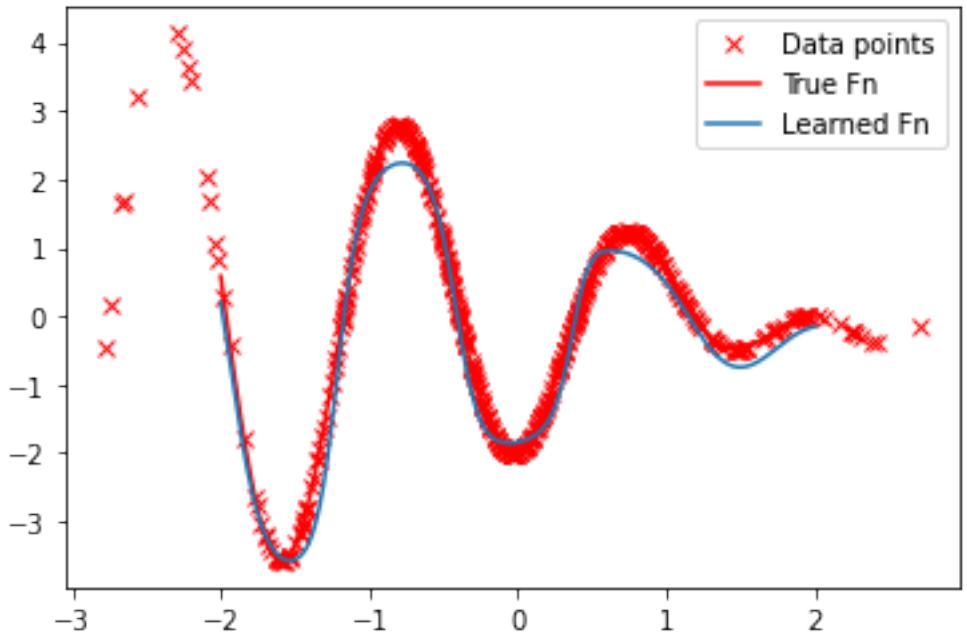
[4]:
```
model = MLP()

model.linear.weight.data = torch.normal(0,1,model.linear.weight.shape)
model.linear.bias.data = torch.full(model.linear.bias.shape,0.03)
model.linear2.weight.data = torch.normal(0,1,model.linear2.weight.shape)
model.linear2.bias.data = torch.full(model.linear.bias.shape,0.03)
model.linear3.weight.data = torch.normal(0,1,model.linear3.weight.shape)
model.linear3.bias.data = torch.full(model.linear3.bias.shape,0.03)

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

# train_loader = DataLoader(dataset=(X_train,y_train), batch_size=B,␣
↪shuffle=True)
train_dataloader = DataLoader(TensorDataset(X_train.unsqueeze(1), y_train.
↪unsqueeze(1)), batch_size=B)

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

```
        train_loss.backward()
        optimizer.step()
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1805: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
```

[5]:
```
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),'r',label='True Fn')
    plt.plot(xx, model(xx),label='Learned Fn')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1805: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
```



Problem 2

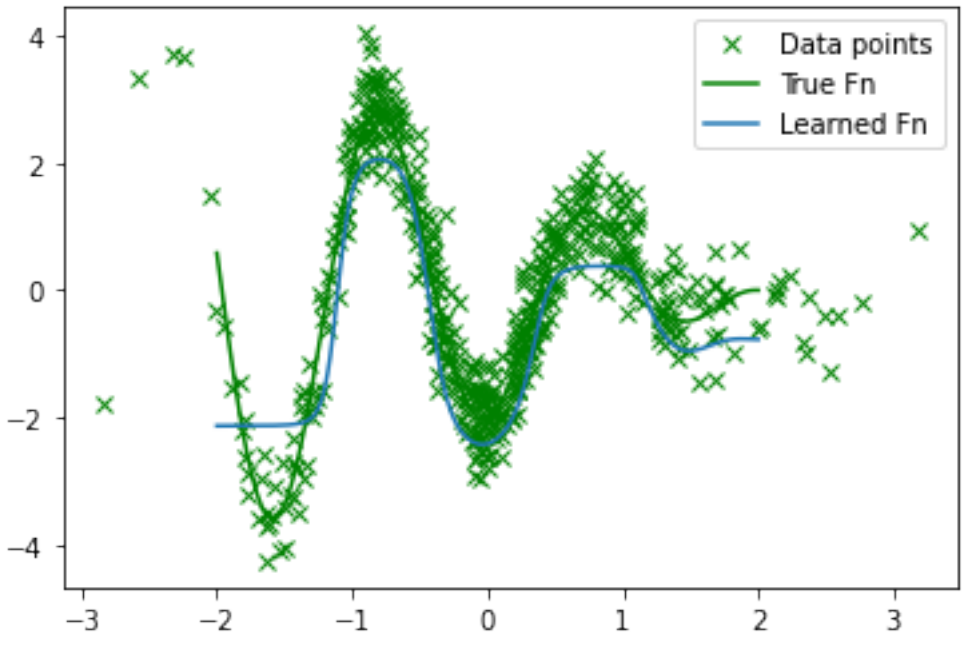
[6]:
```
X_train = torch.tensor(np.random.normal(loc = 0.0, scale = 1.0, size = N),␣
↪dtype=torch.float32)
y_train = f_true(X_train) + torch.normal(0,0.5,X_train.shape)
```

[7]:
```
model = MLP()

model.linear.weight.data = torch.normal(0,1,model.linear.weight.shape)
model.linear.bias.data = torch.full(model.linear.bias.shape,0.03)
model.linear2.weight.data = torch.normal(0,1,model.linear2.weight.shape)
model.linear2.bias.data = torch.full(model.linear.bias.shape,0.03)
model.linear3.weight.data = torch.normal(0,1,model.linear3.weight.shape)
model.linear3.bias.data = torch.full(model.linear3.bias.shape,0.03)

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

# train_loader = DataLoader(dataset=(X_train,y_train), batch_size=B,␣
 ↪shuffle=True)
train_dataloader = DataLoader(TensorDataset(X_train.unsqueeze(1), y_train.
 ↪unsqueeze(1)), batch_size=B)

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

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1805: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
```

[9]:
```
with torch.no_grad():
    xx = torch.linspace(-2,2,1024).unsqueeze(1)
    plt.plot(X_train,y_train,'gx',label='Data points')
    plt.plot(xx,f_true(xx),'g',label='True Fn')
    plt.plot(xx, model(xx),label='Learned Fn')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:1805: UserWarning:
nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid
instead.")
```


Data points
True Fn
Learned Fn
4
2
0
−2
−4
−3
−2
−1
0
1
2
3