CM146, Winter 2023
Problem Set 3: Deep Learning, Learning Theory, Kernels

1 VC Dimension

1.1 (a)

Solution: Refer to attached handwritten solutions at the end of this document.

1.2 (b)(i)

Solution: True. Let's assume that $VC(H_1) = d$. This implies that there is a group of d points that is shattered by H_1 (i.e., for any labeling of the d points, there is a hypothesis $h \in H_1$ that correctly classifies them). Now, since H_2 contains all hypotheses in H_1, then H_2 shatters the same set. Resultantly, we can conclude that $VC(H_2) \geq d = VC(H_1)$.

1.3 (b)(ii)

Solution: False. Here is a counterexample: let $H_1 = \{h_1\}, H_2 = \{h_2\}$, and $\forall x, h_1(x) = 0, h_2(x) = 1$. In this case, $VC(H_1) = VC(H_2) = 0$, but $VC(H_1 \cup H_2) = 1$, which disproves the given statement.

2 Kernels

2.1 (a)

Solution: $k(\mathbf{x}, \mathbf{z})$ is a kernel function. This is because we can represent it with the following matrix (which we will show is positive semi-definite). By Mercer's theorem, k is a kernel function.

$$\mathbf{K} = \begin{bmatrix} \|\mathbf{x} \cap \mathbf{x}\| & \|\mathbf{x} \cap \mathbf{z}\| \\ \|\mathbf{z} \cap \mathbf{x}\| & \|\mathbf{z} \cap \mathbf{z}\| \end{bmatrix}$$

Since $\|\mathbf{a} \cap \mathbf{a}\| = \|\mathbf{a}\|$ for any $\mathbf{a}$ and $\|\mathbf{x} \cap \mathbf{z}\| = \|\mathbf{z} \cap \mathbf{x}\|$, $\mathbf{K}$ reduces to:

$$\mathbf{K} = \begin{bmatrix} \|\mathbf{x}\| & \|\mathbf{x} \cap \mathbf{z}\| \\ \|\mathbf{x} \cap \mathbf{z}\| & \|\mathbf{z}\| \end{bmatrix}$$

Now, the eigenvalues λ of $\mathbf{K}$ are given by:

$$\begin{aligned} 0 &= (\|\mathbf{x}\| - \lambda)(\|\mathbf{z}\| - \lambda) - \|\mathbf{x} \cap \mathbf{z}\|^2 \\ \|\mathbf{x}\|\|\mathbf{z}\| - \lambda\|\mathbf{z}\| - \lambda\|\mathbf{x}\| + \lambda^2 - \|\mathbf{x} \cap \mathbf{z}\|^2 &= 0 \\ \lambda^2 - \lambda(\|\mathbf{x}\| + \|\mathbf{z}\|) + \|\mathbf{x}\|\|\mathbf{z}\| - \|\mathbf{x} \cap \mathbf{z}\|^2 &= 0 \end{aligned}$$

Employing the property of intersections, we get $\|\mathbf{x} \cap \mathbf{z}\| \leq \|\mathbf{x}\|$ and $\|\mathbf{x} \cap \mathbf{z}\| \leq \|\mathbf{z}\|$. Hence,

$$\|\mathbf{x} \cap \mathbf{z}\|^2 \leq \|\mathbf{x}\|\|\mathbf{z}\|$$

Using the quadratic formula to find the values of λ, we have:

$$\lambda = \frac{-(-(\|\mathbf{x}\| + \|\mathbf{z}\|)) \pm \sqrt{(-(\|\mathbf{x}\| + \|\mathbf{z}\|))^2 - 4(1)\left(\|\mathbf{x}\|\|\mathbf{z}\| - \|\mathbf{x} \cap \mathbf{z}\|^2\right)}}{2(1)}$$

$$\lambda = \frac{(\|\mathbf{x}\| + \|\mathbf{z}\|) \pm \sqrt{(\|\mathbf{x}\| + \|\mathbf{z}\|)^2 - 4\|\mathbf{x}\|\|\mathbf{z}\| + 4\|\mathbf{x} \cap \mathbf{z}\|^2}}{2(1)}$$

For the positive root case, λ is trivially non-negative. For the negative case, however, we need to show that:

$$(\|\mathbf{x}\| + \|\mathbf{z}\|) - \sqrt{(\|\mathbf{x}\| + \|\mathbf{z}\|)^2 - 4\|\mathbf{x}\|\|\mathbf{z}\| + 4\|\mathbf{x} \cap \mathbf{z}\|^2} \geq 0$$

Upon some rearrangement and simplification, we obtain:

$$\begin{aligned}(\|\mathbf{x}\| + \|\mathbf{z}\|)^2 \geq &(\|\mathbf{x}\| + \|\mathbf{z}\|)^2 - 4\|\mathbf{x}\|\|\mathbf{z}\| + 4\|\mathbf{x} \cap \mathbf{z}\|^2 \\ 4\|\mathbf{x}\|\|\mathbf{z}\| &\geq 4\|\mathbf{x} \cap \mathbf{z}\|^2 \\ \|\mathbf{x}\|\|\mathbf{z}\| &\geq \|\mathbf{x} \cap \mathbf{z}\|^2\end{aligned}$$

Using the equation we obtained by employing the property of intersection, this last inequality is true. Generally, we can extend this to prove that any kernel matrix is PSD. Therefore, all eigenvalues of $\mathbf{K}$ are non-negative. This means that $\mathbf{K}$ is positive semi-definite, and thus by the Mercer theorem, we can infer k is a kernel function.

2.2 (b)

Solution: We're given that $\mathbf{x} \cdot \mathbf{z}$ is a kernel. Using the scaling property, k' is also a kernel, where:

$$k'(\mathbf{x}, \mathbf{z}) = \frac{\mathbf{x} \cdot \mathbf{z}}{\|\mathbf{x}\|\|\mathbf{z}\|}$$

In this case, the scaling constants are $\|\mathbf{x}\|^{-1}$ and $\|\mathbf{z}\|^{-1}$, both of which are nonnegative. Because 1 is a kernel function (a possible generating function would be $\phi(\mathbf{x}) = 1$), by the sum property, we can conclude $1 + k'$ is also a kernel. By repeated application of the product property:

$$\left(1 + \frac{\mathbf{x}}{\|\mathbf{x}\|}\frac{\mathbf{z}}{\|\mathbf{z}\|}\right)^3 = \left(1 + k'\right)^3 = \left(1 + k'\right)\left(1 + k'\right)\left(1 + k'\right)$$

Hence, this is also a kernel.

2.3 (c)

Solution: We can begin by defining the kernel $k_\beta(\mathbf{x}, \mathbf{z}) = (1 + \beta\mathbf{x} \cdot \mathbf{z})^3$, and expanding it:

$$1 + 3\beta\mathbf{x} \cdot \mathbf{z} + 3\beta^2(\mathbf{x} \cdot \mathbf{z})^2 + \beta^3(\mathbf{x} \cdot \mathbf{z})^3$$

$$1 + 3\beta x_1 z_1 + 3\beta x_2 z_2 + 3\beta^2\left(x_1^2 z_1^2 + 2x_1 x_2 z_1 z_2 + x_2^2 z_2^2\right) + \beta^3\left(x_1^3 x_2^3 + 3x_1^2 x_2^2 z_1 z_2 + 3x_1 x_2 z_1^2 z_2^2 + z_1^3 z_2^3\right)$$

Our intention to find a function ϕ such that $\phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{z})$ equals the expression above. Using a method similar to the case for 2nd-degree polynomials, we get:

$$\phi = \begin{bmatrix} 1 \\ \sqrt{3\beta}x_1 \\ \sqrt{3\beta}x_2 \\ \sqrt{3}\beta x_1^2 \\ \sqrt{3}\beta x_2^2 \\ \sqrt{6}\beta x_1 x_2 \\ \sqrt{\beta^3}x_1^3 \\ \sqrt{\beta^3}x_2^3 \\ \sqrt{3\beta^3}x_1^2 x_2 \\ \sqrt{3\beta^3}x_1 x_2^2 \end{bmatrix}$$

Both methods involve computing the dot product of vectors $\mathbf{x}$ and $\mathbf{z}$. However, in the case of the kernel function $k(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x} \cdot \mathbf{z})^3$, the scaling factor β is absent for each term when $\beta = 1$. As the degree of the transformation vector increases, the scaling factor β also increases, which may be utilized as a regularization parameter.

3 SVM

3.1 (a)

Solution: To solve this, we can use Lagrange multipliers. In this case,

$$f(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 \text{ and } g(\boldsymbol{\theta}) = y_n \boldsymbol{\theta}^T \mathbf{x}_n \geq 1$$

For a training vector $x = \begin{bmatrix} a & e \end{bmatrix}^T$ with label $y = -1$, we can reduce this to:

$$f(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 \text{ and } g(\boldsymbol{\theta}) = -\boldsymbol{\theta}^T \mathbf{x}_n \geq 1$$

The Lagrangian is

$$L = \frac{1}{2}\|\boldsymbol{\theta}\|^2 - \lambda\left(-\boldsymbol{\theta}^T \mathbf{x}_n - 1\right) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 + \lambda\left(\boldsymbol{\theta}^T \mathbf{x}_n + 1\right)$$

Since there's a requirement to minimize this over $\boldsymbol{\theta}$, we take the derivative with respect to it, and set it to 0:

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \boldsymbol{\theta} + \lambda \mathbf{x}_n = 0$$
$$\boldsymbol{\theta}^* = -\lambda \mathbf{x}_n$$

Now, to maximize λ, we can substitute in the previous result, take the derivative with respect to λ, and set it to 0:

$$L = \frac{1}{2}\left\|-\lambda \mathbf{x}_n\right\|^2 + \lambda\left(\left(-\lambda \mathbf{x}_n\right)^T \mathbf{x}_n + 1\right)$$
$$L = \frac{1}{2}\lambda^2 \left\|\mathbf{x}_n\right\|^2 - \lambda^2 \left\|\mathbf{x}_n\right\|^2 + \lambda = -\frac{1}{2}\lambda^2 \left\|\mathbf{x}_n\right\|^2 + \lambda$$
$$\frac{\partial L}{\partial \lambda} = -\lambda \left\|\mathbf{x}_n\right\|^2 + 1 = 0$$
$$\lambda^* = \frac{1}{\left\|\mathbf{x}_n\right\|^2}$$

Lastly, we can substitute this λ^* value into the expression for θ^* to obtain a value that satisfies the given constrained minimization:

$$\boldsymbol{\theta}^* = -\frac{1}{\|\mathbf{x}_n\|}\mathbf{x}_n$$
$$\boldsymbol{\theta}^* = -\frac{1}{a^2+e^2}\begin{bmatrix} a \\ e \end{bmatrix}$$

3.2 (b)

Solution: This optimization problem can be written as follows:

$$f(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2$$
$$g_1(\boldsymbol{\theta}) = \boldsymbol{\theta}^T\mathbf{x}_1 \geq 1$$
$$g_2(\boldsymbol{\theta}) = \boldsymbol{\theta}^T\mathbf{x}_2 \leq -1$$

Assuming the data has 2 dimensions, the two constraint equations give

$$\theta_1 + \theta_2 \geq 1 \text{ and } \theta_1 \leq -1$$

The goal is to minimize the magnitude of $\boldsymbol{\theta}$, so we pick the value $\theta_2 = -1$, which gives $\theta_1 = 2$. The margin in this case is $\frac{1}{\|\boldsymbol{\theta}\|_2} = \frac{\sqrt{5}}{5}$. This gives us the answer:

$$\theta^* = \begin{bmatrix} -1 \\ 2 \end{bmatrix} \text{ and } \gamma = \frac{\sqrt{5}}{5}$$

3.3 (c)

Solution: Since b is allowed to be nonzero, the constraint equations become:

$$\theta_1 + \theta_2 + b \geq 1 \text{ and } \theta_1 + b \leq -1$$

Then, to minimize the magnitude of $\boldsymbol{\theta}$, we can set $b = -1$ so that $\theta_2 = 0$, and then $\theta_1 + 0 - 1 \geq 1$, giving $\theta_1 = 2$. Resultantly, the new values are

$$(\boldsymbol{\theta}^*, b^*) = \left(\begin{bmatrix} 1 & 0 \end{bmatrix}^{\mathrm{T}}, -1\right), \text{ and } \gamma = 0.5$$

The magnitude of $\boldsymbol{\theta}$ has decreased, while the margin has increased. This makes sense because we are now allowing a greater set of hyperplanes to pick from, so the classifier is able to find a better one with smaller magnitude of $\boldsymbol{\theta}$ and a larger margin.

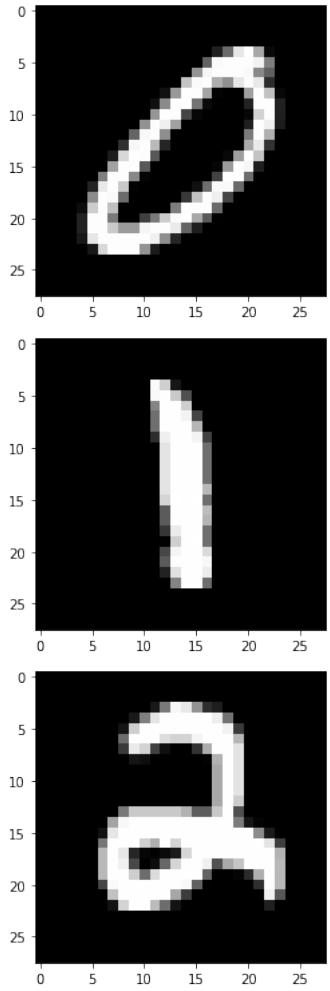
4 Implementation: Digit Recognizer

4.1 Data Visualization and Preparation

(a)

Solution:

```
plot_img(X_train[5])
plot_img(X_train[6])
plot_img(X_train[7])
```



(b)

Solution:

```
X_train = torch.from_numpy(X_train)
y_train = torch.from_numpy(y_train)
X_valid = torch.from_numpy(X_valid)
y_valid = torch.from_numpy(y_valid)
X_test = torch.from_numpy(X_test)
y_test = torch.from_numpy(y_test)
```

(c)

Solution:

```
train_data = TensorDataset(X_train,y_train)
valid_data = TensorDataset(X_valid,y_valid)
test_data = TensorDataset(X_test,y_test)
train_loader = DataLoader(train_data, shuffle=True, batch_size=10)
valid_loader = DataLoader(valid_data, shuffle=False, batch_size=10)
test_loader = DataLoader(test_data, shuffle=False, batch_size=10)
```

4.2 One-Layer Network

(d)

Solution:

```
class OneLayerNetwork(torch.nn.Module):
    def __init__(self):
        super(OneLayerNetwork, self).__init__()

        ### ========== TODO : START ========== ###
        ### part d: implement OneLayerNetwork with torch.nn.Linear
        self.linear = torch.nn.Linear(784, 3)
        ### ========== TODO : END ========== ###

    def forward(self, x):
        # x.shape = (n_batch, n_features)
        ### ========== TODO : START ========== ###
        ### part d: implement the forward function
        outputs = self.linear(x)
        ### ========== TODO : END ========== ###
        return outputs
```

(e)

Solution:

```
    ### ========== TODO : START ========== ###
    ### part e: prepare OneLayerNetwork, criterion, and optimizer
    model_one = OneLayerNetwork()
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(model_one.parameters(), lr=0.0005)
    ### ========== TODO : END ========== ###
```

(f)

Solution:

```
    ### ========== TODO : START ========== ###
    ### part f: implement the training process
    pred = model.forward(batch_X)
    optimizer.zero_grad()
    loss = criterion(pred, batch_y)
    loss.backward()
```

```
    optimizer.step()
    ### ========== TODO : END ========== ###
```

4.3 Two-Layer Network

(g)

Solution:

```
class TwoLayerNetwork(torch.nn.Module):
    def __init__(self):
        super(TwoLayerNetwork, self).__init__()
        ### ========== TODO : START ========== ###
        ### part g: implement TwoLayerNetwork with torch.nn.Linear
        self.hidden = torch.nn.Linear(784, 400)
        self.output = torch.nn.Linear(400, 3)
        ### ========== TODO : END ========== ###

    def forward(self, x):
        # x.shape = (n_batch, n_features)

        ### ========== TODO : START ========== ###
        ### part g: implement the foward function
        hidden = self.hidden(x)
        hidden = torch.nn.sigmoid(hidden)
        outputs = self.output(hidden)
        ### ========== TODO : END ========== ###
        return outputs
```

(h)

Solution:

```
    ### ========== TODO : START ========== ###
    ### part h: prepare TwoLayerNetwork, criterion, and optimizer
    model_two = TwoLayerNetwork()
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(model_two.parameters(), lr=0.0005)
    ### ========== TODO : END ========== ###
```

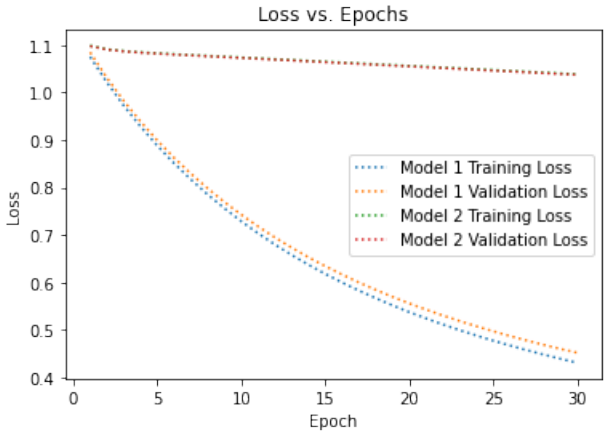
4.4 Performance Comparison

(i)

Solution: The rate of decrease in loss for the training and validation set is much slower in the two-layered model compared to the one-layered network. The reason for this slow learning is that in the two-layered network, the error gradient is back-propagated through the network with each adjustment. This back-propagation uses the chain rule, which can result in very small derivatives that barely change the weights in the initial layers when adjustments are made. The one-layer network has fewer parameters, making optimization easier and faster convergence possible. Additionally, both models have similar training and validation loss, indicating good generalization and

no overfitting.

```
### ========== TODO : START ========== ###
### part i: generate a plot to comare one_train_loss, one_valid_loss,
    two_train_loss, two_valid_loss
epochs = range(1, len(one_train_loss) + 1)
plt.plot(epochs, one_train_loss, linestyle='.', label='Model 1 Training Loss')
plt.plot(epochs, one_valid_loss, linestyle='.', label='Model 1 Validation Loss')
plt.plot(epochs, two_train_loss, linestyle='.', label='Model 2 Training Loss')
plt.plot(epochs, two_valid_loss, linestyle='.', label='Model 2 Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss vs. Epochs')
plt.legend()
plt.show()
### ========== TODO : END ========== ###
```

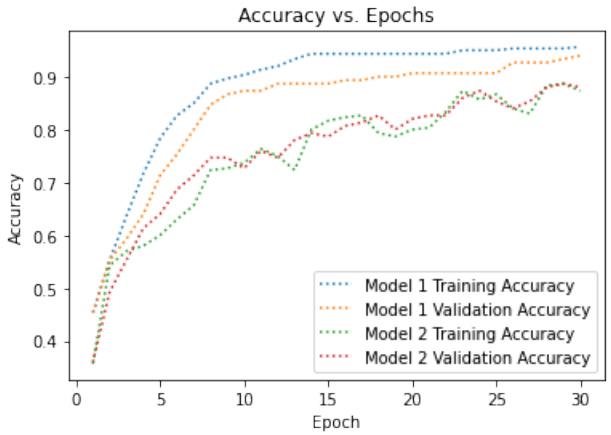


(j)

Solution: The graph indicates that the one-layer neural network is currently performing better than the two-layer neural network in terms of convergence. The two-layered neural network has a slower growth in accuracy due to having more weights to change and back-propagating through them, which results in small derivatives that adjust the weights slowly. However, the accuracies of both models become similar in the later epochs. It's worth noting that the accuracy on the validation dataset is consistently lower than the accuracy on the training dataset, indicating slight potential overfitting. Nevertheless, the data generalizes well since the difference is not significant. The one-layer model has train and validation accuracies around 95%, while the two-layer model has train and validation accuracies around 88%.

```
### ========== TODO : START ========== ###
### part j: generate a plot to comare one_train_acc, one_valid_acc, two_train_acc,
    two_valid_acc
epochs = range(1, len(one_train_acc) + 1)
plt.plot(epochs, one_train_acc, linestyle='.', label='Model 1 Training Accuracy')
plt.plot(epochs, one_valid_acc, linestyle='.', label='Model 1 Validation Accuracy')
```

```
plt.plot(epochs, two_train_acc, linestyle='.', label='Model 2 Training Accuracy')
plt.plot(epochs, two_valid_acc, linestyle='.', label='Model 2 Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Epochs')
plt.legend()
plt.show()
### ========== TODO : END ========== ##
```



(k)

Solution:

One-Layer Network Test Accuracy: 96.00%
Two-Layer Network Test Accuracy: 84.67%

The test accuracy of the one-layer network is considerably better than that of the two-layer network, which can be attributed to two possible reasons. Firstly, since the two-layer network has more parameters, it may require a longer training process than the 30 epochs provided in the experiment. Secondly, it's possible that the two-layer network overfitted, but the accuracy vs. epoch graph suggests that any such effect, if present, is insignificant.

```
### ========== TODO : START ========== ###
### part k: calculate the test accuracy
m1_acc = evaluate_acc(model_one, test_loader)
m2_acc = evaluate_acc(model_two, test_loader)

print("One Neural Net Test Accuracy:", m1_acc)
print("Two Neural Net Test Accuracy:", m2_acc)
print("Two Neural Net Val Acc:", two_valid_acc[29])
### ========== TODO : END ========== ###
```

```
One Neural Net Test Accuracy: tensor(0.9600)
Two Neural Net Test Accuracy: tensor(0.8467)
Two Neural Net Val Acc: tensor(0.8800)
```

(l)

Solution: The Adam optimizer yields quicker convergence than the SGD optimizer, enabling us to realize the true accuracy of the two-layer network as it converges. Based on the graphs, the two-layer network performs better than the one-layer network on both training and validation data, and equally well on test data. It's evident that the Adam optimizer is superior to SGD in this scenario. The two-layer network achieves a validation accuracy of 97.5% within five epochs. One possible explanation for the identical test accuracies of the models is the limitations imposed by the small dataset. According to learning theory, the generalization error depends on the number of training examples, and certain outliers in the test data can prevent the accuracy from reaching 100%. Therefore, it's possible that both models have learned the maximum amount from the limited training examples, resulting in the same test accuracy.

```
One Neural Net Test Accuracy: tensor(0.9667)
Two Neural Net Test Accuracy: tensor(0.9667)
Two Neural Net Val Acc: tensor(0.9733)
```

```
optimizer = torch.optim.Adam(model_one.parameters(), lr=0.0005)
```