CM146, Fall 2020
Problem Set 3: VC Dimension and Neural Networks
Due Nov. 30 at 11:59 pm

1 VC Dimension [16 pts]

For the following problems, we classify a point x to either a positive label $+1$ or a negative label -1 by a hypothesis set $\mathcal{H}$.

(a) *Positive ray classifier.* Consider $x \in \mathbb{R}$ and $\mathcal{H} = \{sign(x - b) \mid b \in \mathbb{R}\}$. That is, the label is $+1$ if x is greater than b otherwise -1. [8 pts]

i. For N points, prove that there are at most $N + 1$ label outcomes that can be generated by $\mathcal{H}$. For example, when we have 4 points, $x_1 \leq x_2 \leq x_3 \leq x_4$, there are at most 5 label outcomes can be generated by $\mathcal{H}$, as shown by the following.

x_1	x_2	x_3	x_4
$+1$	$+1$	$+1$	$+1$
-1	$+1$	$+1$	$+1$
-1	-1	$+1$	$+1$
-1	-1	-1	$+1$
-1	-1	-1	-1

Solution: When we have 4 points, $x_1 \leq x_2 \leq ... \leq x_N$, all the possible outcomes will have negative ones on the left and positive ones on the right plus two outcomes with all positive and all negative, and each point can be the negative one, so we have a total of N+1 label outcomes.

ii. What is the VC dimension of $\mathcal{H}$?

Solution:

$$m_H(N) = N + 1$$
$$m_H(1) = 1 + 1 = 2^1 => VC \geq 1$$
$$m_H(2) = 2 + 1 < 2^2 => VC < 2$$

For all one point, the label can be either +1 or -1. For all two points, if the first one is +1 and the second one is -1, then it cannot be shattered by $\mathcal{H}$, so the VC dimension of $\mathcal{H}$ is $\boxed{1}$.

(b) *Positive interval classifier.* Consider $x \in \mathbb{R}$ and $\mathcal{H} = \{sign(\mathbb{1}(x \in [a, b]) - 0.5) \mid a, b \in \mathbb{R}, a \leq b\}$, where $\mathbb{1}(.)$ is the indicator function. That is, the label is $+1$ if x in the interval $[a, b]$ otherwise -1. [8 pts]

i. For N points, prove that there are at most $(\frac{N^2+N}{2} + 1)$ label outcomes that can be generated by $\mathcal{H}$.

Part of this assignment is adapted from the course materials by Hsuan-Tien Lin (National Taiwan University).

Solution: For this kind of classifier, the general pattern is that there are negative ones at the beginning and end of the sequence with positive ones in between. If there is only one positive one, we have N different outcomes and two positive ones with N-1 different outcomes and three positive ones with N-2 different outcomes...and N positive ones with only one outcome and 0 positive one with only one outcome, so in total we have $N + N - 1 + N - 2 + ...1 + 1 = \frac{(1+N)N}{2} + 1 = (\frac{N^2+N}{2} + 1)$

ii. What is the VC dimension of $\mathcal{H}$?

Solution:

$$m_H(N) = \frac{N^2 + N}{2} + 1$$
$$m_H(1) = \frac{1^2 + 1}{2} + 1 = 2^1 => VC \geq 1$$
$$m_H(2) = \frac{2^2 + 2}{2} + 1 = 2^2 => VC \geq 2$$
$$m_H(3) = \frac{3^2 + 3}{2} + 1 < 2^3 => VC < 3$$

For all one point, the label can be either +1 or -1. For all two points, the labels can be either +1 or -1. For all three points, if the first one and third one is +1 and the second one is -1, then it cannot be shattered by $\mathcal{H}$, so the VC dimension of $\mathcal{H}$ is $\boxed{2}$.

2 Bound of VC dimension [16 pts]

Assume the VC dimension of an empty set is zero. Now, we have two hypothesis sets $\mathcal{H}_1$ and $\mathcal{H}_2$.

(a) Let $\mathcal{H}_3 = \mathcal{H}_1 \cap \mathcal{H}_2$. Show that $VC(\mathcal{H}_1) \geq VC(\mathcal{H}_3)$. [6 pts]

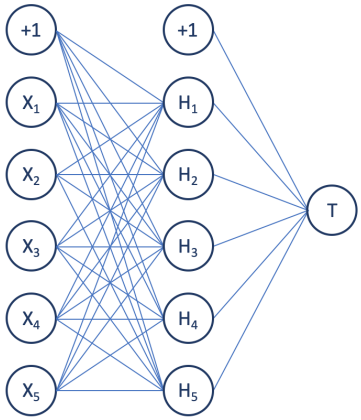
Solution: If $\mathcal{H}_3 = \mathcal{H}_1 \cap \mathcal{H}_2$, then $\mathcal{H}_3 \subseteq \mathcal{H}_1$. Since a richer set of functions shatters large sets of points, $VC(\mathcal{H}_1) \geq VC(\mathcal{H}_3)$.

(b) Let $\mathcal{H}_3 = \mathcal{H}_1 \cup \mathcal{H}_2$. Give an example to show that $VC(\mathcal{H}_1) + VC(\mathcal{H}_2) < VC(\mathcal{H}_3)$ is possible. [10 pts]

Solution: $\mathcal{H}_1 = \{h\}$ $h = -1$ and $\mathcal{H}_2 = \{h_2\}$ $h_2 = 1$ Obviously, $VC(\mathcal{H}_1) = VC(\mathcal{H}_2) = 0$, $VC(\mathcal{H}_1) + VC(\mathcal{H}_2) = 0 < VC(\mathcal{H}_3) = 1$

3 Neural Network [20 pts]

We design a neural network to implement the XOR operation of X_1, X_2, X_3, X_4, X_5. We use +1 to represent `true` and −1 to represent `false`. Consider the following neural network.



We use w_{ij} to represent the weight between X_i and H_j, and use w_{0j} to represent the weight between the first layer bias term $(+1)$ and H_j. We use v_i to represent the weight between H_i and T, and use v_0 to represent the weight between the second layer bias term $(+1)$ and T.

Now, let $X_i \in \{+1, -1\}$, $H_j = sign\left(\sum_{i=0}^{5} w_{ij} X_i\right)$, and $T = sign\left(\sum_{i=0}^{5} v_i H_i\right)$.

(a) Specify w_{ij} such that H_j is $+1$ if there are at least j positive values among X_1, X_2, X_3, X_4, X_5, otherwise -1. If there are multiple acceptable weights, you only need to write down one of them. [8 pts]

Solution:

$w_{01} = 7$ $w_{11} = w_{21} = w_{31} = w_{41} = w_{51} = 2$
$w_{02} = 5$ $w_{12} = w_{22} = w_{32} = w_{42} = w_{52} = 2$
$w_{03} = 1$ $w_{13} = w_{23} = w_{33} = w_{43} = w_{53} = 2$
$w_{04} = -3$ $w_{14} = w_{24} = w_{34} = w_{44} = w_{54} = 2$
$w_{05} = -7$ $w_{15} = w_{25} = w_{35} = w_{45} = w_{55} = 2$

(b) Given w_{ij} and H_j defined as above, specify v_i such that the whole neural network behaves like the XOR operation of X_1, X_2, X_3, X_4, X_5. If there are multiple acceptable weights, you only need to write down one of them. [8 pts]

Solution:

$v_0 = 0$
$v_1 = 5$
$v_2 = -3$
$v_3 = 1$
$v_4 = -4$
$v_5 = 5.1$

(c) Justify why the output of the neural network behaves like the XOR operation of X_1, X_2, X_3, X_4, X_5. [4 pts]

Solution: we can justify by simply list all the possible outcomes.
When there is no positive input, XOR should output -1
$H_1 = H_2 = H_3 = H_4 = H_5 = -1, T = -5 + 3 - 1 + 4 - 5.1 = -4.1 < 0$ output is -1.
When there is one positive input, XOR should output 1
$H_1 = 1, H_2 = H_3 = H_4 = H_5 = -1, T = 5 + 3 - 1 + 4 - 5.1 = 5.9 > 0$ output is 1.
When there are two positive input, XOR should output -1
$H_1 = H_2 = 1, H_3 = H_4 = H_5 = -1, T = 5 - 3 - 1 + 4 - 5.1 = -0.1 < 0$ output is -1.
When there are three positive input, XOR should output 1
$H_1 = H_2 = H_3 = 1, H_4 = H_5 = -1, T = 5 - 3 + 1 + 4 - 5.1 = 1.9 > 0$ output is 1.
When there are four positive input, XOR should output -1
$H_1 = H_2 = H_3 = H_4 = 1, H_5 = -1, T = 5 - 3 + 1 - 4 - 5.1 = -6.1 < 0$ output is -1.
When there are five positive input, XOR should output 1
$H_1 = H_2 = H_3 = H_4 = H_5 = 1, T = 5 - 3 + 1 - 4 + 5.1 = 4.1 > 0$ output is 1.

4 Implementation: Digit Recognizer [48 pts]

In this exercise, you will implement a digit recognizer in pytorch. Our data contains pairs of 28×28 images $\mathbf{x}_n$ and the corresponding digit labels $y_n \in \{0, 1, 2\}$. For simplicity, we view a 28×28 image $\mathbf{x}_n$ as a 784-dimensional vector by concatenating the row pixels. In other words, $\mathbf{x}_n \in \mathbb{R}^{784}$. Your goal is to implement two digit recognizers (`OneLayerNetwork` and `TwoLayerNetwork`) and compare their performances.

---

code and data

- code : `Fall2020-CS146-HW3.ipynb`
- data : `hw3_train.csv`, `hw3_valid.csv`, `hw3_test.csv`

Please use your *@g.ucla.edu* email id to access the code and data. Similar to *HW-1*, copy the colab notebook to your drive and make the changes. Mount the drive appropriately and copy the shared data folder to your drive to access via colab. For colab usage demo, check out the Discussion recordings for Week 2 in CCLE. The notebook has marked blocks where you need to code.

========= TODO : START =========

========= TODO : END ==========

Note: For the questions requiring you to complete a piece of code, you are expected to copy-paste your code as a part of the solution in the submission pdf. Tip: If you are using LaTeX, check out the Minted package (example) for code highlighting.

Data Visualization and Preparation [10 pts]

(a) Randomly select three training examples with *different labels* and print out the images by using `plot_img` function. Include those images in your report. [2 pts]

Solution:

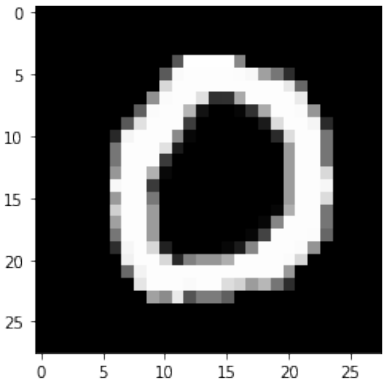


Figure 1: Label 0

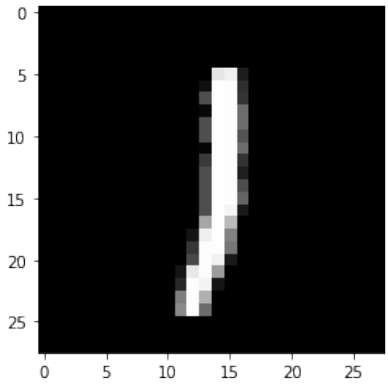


Figure 2: Label 1

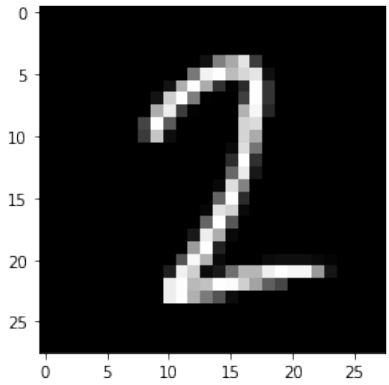


Figure 3: Label 2

```
import random
label0=[]
label1=[]
label2=[]
for i in range(len(X_train)):
    if y_train[i] == 0:
        label0.append(X_train[i])
    if y_train[i] == 1:
        label1.append(X_train[i])
    if y_train[i] == 2:
        label2.append(X_train[i])
plot_img(random.choice(label0))
plot_img(random.choice(label1))
plot_img(random.choice(label2))
```

(b) The loaded examples are numpy arrays. Convert the numpy arrays to tensors. [3 pts]

Solution:

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

(c) Prepare `train_loader`, `valid_loader`, and `test_loader` by using `TensorDataset` and `DataLoader`. We expect to get a batch of pairs $(\mathbf{x}_n, y_n)$ from the dataloader. Please set the batch size to 10. [5 pts]

You can refer https://pytorch.org/docs/stable/data.html for more information about `TensorDataset` and `DataLoader`.

Solution:

```
    train_dataset = TensorDataset(X_train, y_train)
    train_loader = DataLoader(train_dataset, batch_size=10)
    valid_dataset = TensorDataset(X_valid, y_valid)
    valid_loader = DataLoader(valid_dataset, batch_size=10)
    test_dataset = TensorDataset(X_test, y_test)
    test_loader = DataLoader(test_dataset, batch_size=10)
```

One-Layer Network [15 pts]

For one-layer network, we consider a *784–3* network. In other words, we learn a 784×3 weight matrix $\mathbf{W}$. Given a $\mathbf{x}_n$, we can compute the probability vector $\mathbf{p}_n = \sigma(\mathbf{W}^\top \mathbf{x}_n)$, where $\sigma(.)$ is the element-wise sigmoid function and $\mathbf{p}_{n,c}$ denotes the probability of class c. Then, we focus on the *cross entropy loss*

$$-\sum_{n=0}^{N}\sum_{c=0}^{C} \mathbb{1}(c = y_n) \log(\mathbf{p}_{n,c})$$

where N is the number of examples, C is the number of classes, and $\mathbb{1}$ is the indicator function.

(d) Implement the constructor of `OneLayerNetwork` with `torch.nn.Linear` and implement the `forward` function to compute the outputs of the single fully connected layer i.e. $\mathbf{W}^\top \mathbf{x}_n$. Notice that we do not compute the sigmoid function here since we will use `torch.nn.CrossEntropyLoss` later. [5 pts]

You can refer to https://pytorch.org/docs/stable/generated/torch.nn.Linear.html for more information about `torch.nn.Linear` and refer to https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html for more information about using `torch.nn.CrossEntropyLoss`.

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 foward function
        outputs = self.linear(x)
        ### ========== TODO : END ========== ###
        return outputs
```

(e) Create an instance of `OneLayerNetwork`, set up a criterion with `torch.nn.CrossEntropyLoss`, and set up a SGD optimizer with learning rate 0.0005 by using `torch.optim.SGD` [2 pts]

Solution:

```
model_one = OneLayerNetwork()
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model_one.parameters(), lr=0.0005)
```

You can refer to https://pytorch.org/docs/stable/optim.html for more information about `torch.optim.SGD`.

(f) Implement the training process. This includes forward pass, initializing gradients to zeros, computing loss, loss backward, and updating model parameters. If you implement everything correctly, after running the `train` function in main, you should get results similar to the following. [8 pts]

```
Start training OneLayerNetwork...
| epoch  1 | train loss 1.075387 | train acc 0.453333 | valid loss ...
| epoch  2 | train loss 1.021301 | train acc 0.563333 | valid loss ...
| epoch  3 | train loss 0.972599 | train acc 0.630000 | valid loss ...
| epoch  4 | train loss 0.928335 | train acc 0.710000 | valid loss ...
...
```

Solution: Start training OneLayerNetwork...
— epoch 1 — train loss 1.075398 — train acc 0.453333 — valid loss 1.084938 — valid acc 0.453333 —
— epoch 2 — train loss 1.021364 — train acc 0.566667 — valid loss 1.031102 — valid acc 0.553333 —
— epoch 3 — train loss 0.972648 — train acc 0.630000 — valid loss 0.982742 — valid acc 0.593333 —
— epoch 4 — train loss 0.928398 — train acc 0.710000 — valid loss 0.938953 — valid acc 0.640000 —
— epoch 5 — train loss 0.887963 — train acc 0.783333 — valid loss 0.899045 — valid acc 0.700000 —
— epoch 6 — train loss 0.850839 — train acc 0.826667 — valid loss 0.862485 — valid acc 0.753333 —
— epoch 7 — train loss 0.816627 — train acc 0.850000 — valid loss 0.828852 — valid acc 0.793333 —
— epoch 8 — train loss 0.785000 — train acc 0.886667 — valid loss 0.797807 — valid acc 0.846667 —
— epoch 9 — train loss 0.755688 — train acc 0.900000 — valid loss 0.769067 — valid acc 0.866667 —
— epoch 10 — train loss 0.728461 — train acc 0.903333 — valid loss 0.742397 — valid acc 0.873333 —
— epoch 11 — train loss 0.703122 — train acc 0.913333 — valid loss 0.717596 — valid acc 0.880000 —
— epoch 12 — train loss 0.679499 — train acc 0.920000 — valid loss 0.694488 — valid acc 0.886667 —

— epoch 13 — train loss 0.657439 — train acc 0.933333 — valid loss 0.672921 — valid acc 0.886667 —
— epoch 14 — train loss 0.636807 — train acc 0.943333 — valid loss 0.652760 — valid acc 0.886667 —
— epoch 15 — train loss 0.617482 — train acc 0.943333 — valid loss 0.633883 — valid acc 0.886667 —
— epoch 16 — train loss 0.599356 — train acc 0.943333 — valid loss 0.616184 — valid acc 0.886667 —
— epoch 17 — train loss 0.582330 — train acc 0.943333 — valid loss 0.599565 — valid acc 0.893333 —
— epoch 18 — train loss 0.566316 — train acc 0.943333 — valid loss 0.583938 — valid acc 0.900000 —
— epoch 19 — train loss 0.551234 — train acc 0.943333 — valid loss 0.569225 — valid acc 0.906667 —
— epoch 20 — train loss 0.537010 — train acc 0.943333 — valid loss 0.555355 — valid acc 0.906667 —
— epoch 21 — train loss 0.523580 — train acc 0.943333 — valid loss 0.542262 — valid acc 0.906667 —
— epoch 22 — train loss 0.510882 — train acc 0.943333 — valid loss 0.529888 — valid acc 0.906667 —
— epoch 23 — train loss 0.498862 — train acc 0.950000 — valid loss 0.518179 — valid acc 0.906667 —
— epoch 24 — train loss 0.487470 — train acc 0.950000 — valid loss 0.507086 — valid acc 0.906667 —
— epoch 25 — train loss 0.476660 — train acc 0.950000 — valid loss 0.496564 — valid acc 0.906667 —
— epoch 26 — train loss 0.466391 — train acc 0.953333 — valid loss 0.486573 — valid acc 0.926667 —
— epoch 27 — train loss 0.456625 — train acc 0.953333 — valid loss 0.477076 — valid acc 0.926667 —
— epoch 28 — train loss 0.447328 — train acc 0.953333 — valid loss 0.468038 — valid acc 0.926667 —
— epoch 29 — train loss 0.438467 — train acc 0.956667 — valid loss 0.459429 — valid acc 0.933333 —
— epoch 30 — train loss 0.430013 — train acc 0.956667 — valid loss 0.451220 — valid acc 0.940000 —
Done!

```
            optimizer.zero_grad()
            output = model.forward(batch_X)
            loss = criterion(output, batch_y)
            loss.backward()
            optimizer.step()
```

Two-Layer Network [7 pts]

For two-layer network, we consider a *784–400–3* network. In other words, the first layer will consist

of a fully connected layer with 784×400 weight matrix $\mathbf{W}_1$ and a second layer consisting of 400×3 weight matrix $\mathbf{W}_2$. Given a $\mathbf{x}_n$, we can compute the probability vector $\mathbf{p}_n = \sigma(\mathbf{W}_2^\top \sigma(\mathbf{W}_1^\top \mathbf{x}_n))$, where $\sigma(.)$ is the element-wise sigmoid function. Again, we focus on the *cross entropy loss*, hence the network will impelement $\mathbf{W}_2^\top \sigma(\mathbf{W}_1^\top \mathbf{x}_n)$ (note the outer sigmoid will be taken care of implicitly in our loss).

(g) Implement the constructor of `TwoLayerNetwork` with `torch.nn.Linear` and implement the `forward` function to compute $\mathbf{W}_2^\top \sigma(\mathbf{W}_1^\top \mathbf{x}_n)$. [5 pts]

Solution:

```
class TwoLayerNetwork(torch.nn.Module):
    def __init__(self):
        super(TwoLayerNetwork, self).__init__()
        ### ========== TODO : START ========== ###
        ### part g: implement TwoLayerNetwork with torch.nn.Linear
        self.sigmoid = torch.nn.Sigmoid()
        self.linear1 = torch.nn.Linear(784, 400)
        self.linear2 = 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
        outputs = self.linear1(x)
        outputs = self.sigmoid(outputs)
        outputs = self.linear2(outputs)
        ### ========== TODO : END ========== ###
        return outputs
```

(h) Create an instance of `TwoLayerNetwork`, set up a criterion with `torch.nn.CrossEntropyLoss`, and set up a SGD optimizer with learning rate 0.0005 by using `torch.optim.SGD`. Then train `TwoLayerNetwork`. [2 pts]

Solution: Start training TwoLayerNetwork...
— epoch 1 — train loss 1.098020 — train acc 0.240000 — valid loss 1.098498 — valid acc 0.253333 —
— epoch 2 — train loss 1.096157 — train acc 0.283333 — valid loss 1.096622 — valid acc 0.340000 —
— epoch 3 — train loss 1.094329 — train acc 0.386667 — valid loss 1.094783 — valid acc 0.380000 —
— epoch 4 — train loss 1.092512 — train acc 0.433333 — valid loss 1.092956 — valid acc 0.400000 —
— epoch 5 — train loss 1.090700 — train acc 0.470000 — valid loss 1.091135 — valid acc 0.413333 —

— epoch 6 — train loss 1.088891 — train acc 0.486667 — valid loss 1.089318 — valid acc 0.420000 —
— epoch 7 — train loss 1.087085 — train acc 0.496667 — valid loss 1.087503 — valid acc 0.453333 —
— epoch 8 — train loss 1.085281 — train acc 0.526667 — valid loss 1.085691 — valid acc 0.466667 —
— epoch 9 — train loss 1.083480 — train acc 0.533333 — valid loss 1.083882 — valid acc 0.486667 —
— epoch 10 — train loss 1.081682 — train acc 0.550000 — valid loss 1.082076 — valid acc 0.506667 —
— epoch 11 — train loss 1.079886 — train acc 0.560000 — valid loss 1.080273 — valid acc 0.540000 —
— epoch 12 — train loss 1.078093 — train acc 0.573333 — valid loss 1.078472 — valid acc 0.553333 —
— epoch 13 — train loss 1.076302 — train acc 0.593333 — valid loss 1.076674 — valid acc 0.566667 —
— epoch 14 — train loss 1.074514 — train acc 0.633333 — valid loss 1.074878 — valid acc 0.626667 —
— epoch 15 — train loss 1.072727 — train acc 0.683333 — valid loss 1.073084 — valid acc 0.660000 —
— epoch 16 — train loss 1.070942 — train acc 0.750000 — valid loss 1.071292 — valid acc 0.693333 —
— epoch 17 — train loss 1.069159 — train acc 0.776667 — valid loss 1.069502 — valid acc 0.746667 —
— epoch 18 — train loss 1.067377 — train acc 0.806667 — valid loss 1.067713 — valid acc 0.773333 —
— epoch 19 — train loss 1.065597 — train acc 0.820000 — valid loss 1.065926 — valid acc 0.800000 —
— epoch 20 — train loss 1.063817 — train acc 0.826667 — valid loss 1.064139 — valid acc 0.820000 —
— epoch 21 — train loss 1.062038 — train acc 0.843333 — valid loss 1.062354 — valid acc 0.833333 —
— epoch 22 — train loss 1.060260 — train acc 0.860000 — valid loss 1.060569 — valid acc 0.840000 —
— epoch 23 — train loss 1.058483 — train acc 0.870000 — valid loss 1.058785 — valid acc 0.853333 —
— epoch 24 — train loss 1.056706 — train acc 0.876667 — valid loss 1.057001 — valid acc 0.860000 —
— epoch 25 — train loss 1.054928 — train acc 0.883333 — valid loss 1.055217 — valid acc 0.880000 —
— epoch 26 — train loss 1.053151 — train acc 0.886667 — valid loss 1.053433 — valid acc 0.886667 —
— epoch 27 — train loss 1.051374 — train acc 0.890000 — valid loss 1.051650 — valid acc 0.893333 —
— epoch 28 — train loss 1.049596 — train acc 0.893333 — valid loss 1.049865 — valid acc 0.900000 —
— epoch 29 — train loss 1.047818 — train acc 0.893333 — valid loss 1.048081 — valid acc 0.900000 —

— epoch 30 — train loss 1.046038 — train acc 0.896667 — valid loss 1.046295 — valid acc 0.893333 —
Done!

```
model_two = TwoLayerNetwork()
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model_two.parameters(), lr=0.0005)
```

Performance Comparison [16 pts]

(i) Generate a plot depicting how `one_train_loss`, `one_valid_loss`, `two_train_loss`, `two_valid_loss` varies with epochs. Include the plot in the report and describe your findings. [3 pts]

Solution:

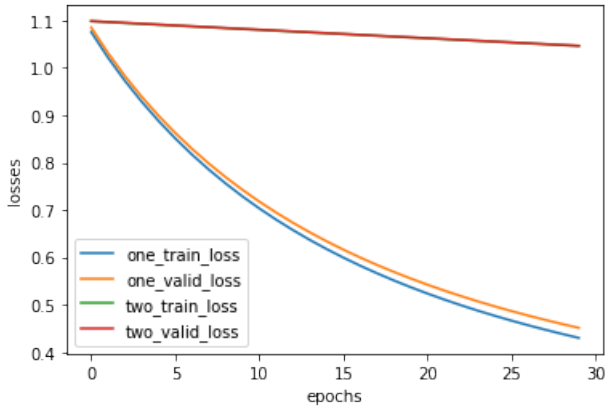


Figure 4: Losses

The train loss and the valid loss for one-layer network decrease much more quickly than the ones associated with the two-layer network within 30 epochs, which is probably because SGD converges slower under the two-layer network.

```
plt.figure()
plt.plot(one_train_loss, label='one_train_loss')
plt.plot(one_valid_loss, label='one_valid_loss')
plt.plot(two_train_loss, label='two_train_loss')
plt.plot(two_valid_loss, label='two_valid_loss')
plt.legend()
plt.xlabel('epochs')
plt.ylabel('losses')
plt.show()
```

(j) Generate a plot depicting how `one_train_acc`, `one_valid_acc`, `two_train_acc`, `two_valid_acc` varies with epochs. Include the plot in the report and describe your findings. [3 pts]

Solution:

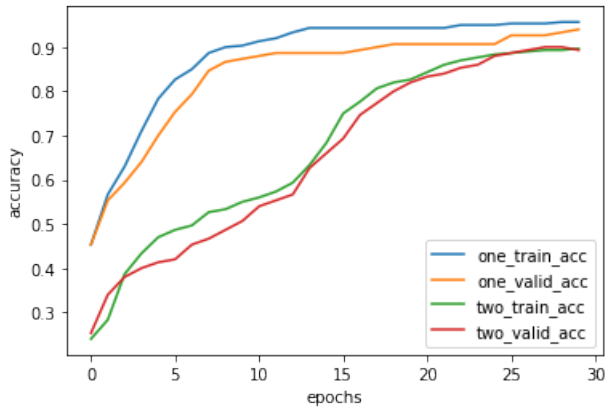


Figure 5: Accuracy

Both the train and test accuracy for two-layer network are lower than the ones associated with one-layer network within the 30 epochs, but both ones approach to 0.9 at the end and their difference also becomes smaller.

```
plt.figure()
plt.plot(one_train_acc, label='one_train_acc')
plt.plot(one_valid_acc, label='one_valid_acc')
plt.plot(two_train_acc, label='two_train_acc')
plt.plot(two_valid_acc, label='two_valid_acc')
plt.legend()
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.show()
```

(k) Calculate and report the test accuracy of both the one-layer network and the two-layer network. Explain why we get such results. [3 pts]

Solution: test_loss_onelayer: 0.444583264986674
test_acc_onelayer: 0.9599999785423279
test_loss_twolayer: 1.0490478912989298
test_acc_twolayer: 0.8999999761581421
One-layer network has higher accuracy and lower loss than the two-layer network because SGD converges slower for the two-layer network and it is going to perform poorly with just 30 epochs.

```
test_loss_onelayer = evaluate_loss(model_one, criterion, test_loader)
test_acc_onelayer = evaluate_acc(model_one, test_loader)
test_loss_twolayer = evaluate_loss(model_two, criterion, test_loader)
test_acc_twolayer = evaluate_acc(model_two, test_loader)
print('test_loss_onelayer: ', test_loss_onelayer)
print('test_acc_onelayer: ', test_acc_onelayer.item())
print('test_loss_twolayer: ', test_loss_twolayer)
print('test_acc_twolayer: ', test_acc_twolayer.item())
```

(l) Replace the SGD optimizer with the Adam optimizer and do the experiments again. Show the loss figure, the accuracy figure, and the test accuracy. Include the figures in the report and describe your findings. [7 pts]

You can refer to https://pytorch.org/docs/stable/optim.html for more information about `torch.optim.Adam`.

Solution:

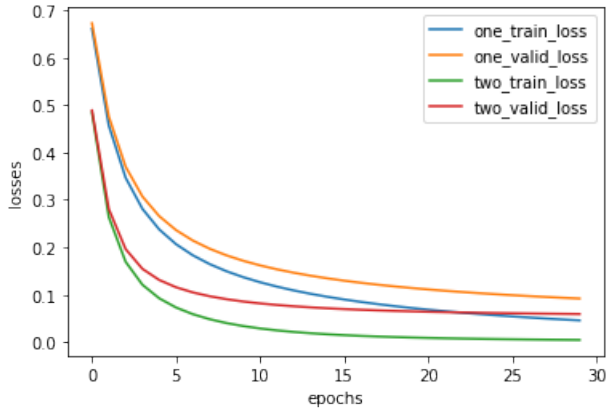


Figure 6: Accuracy

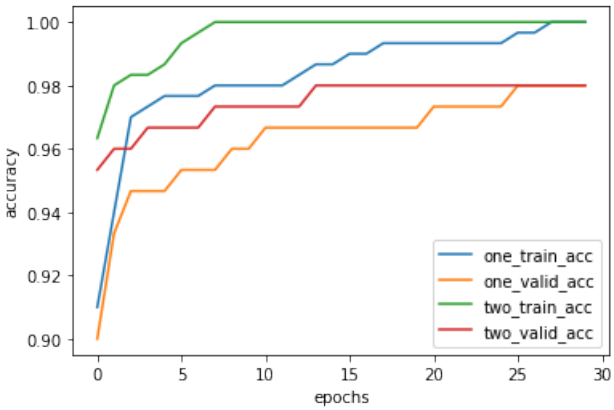


Figure 7: Accuracy

test_loss_onelayer: 0.08791174081464609
test_acc_onelayer: 0.9733333587646484
test_loss_twolayer: 0.05374169985686118
test_acc_twolayer: 0.9666666388511658
Now the two-layer network has higher accuracy and much lower loss compared to the previous result in part k. Both one-layer network and two-layer network now have high accuracy close to one, but two-layer network has a much lower test loss than the one-layer network. It means that the two layer network now performs better and converges more quickly under Adam than SGD. In general, both accuracy and loss improved under Adam than SGD, so we may conclude that Adam is a better optimizer for this training task.

```
    model_one = OneLayerNetwork()
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model_one.parameters(), lr=0.0005)

    print("Start training OneLayerNetwork...")
    results_one = train(model_one, criterion, optimizer, train_loader, valid_loader)
    print("Done!")

    model_two = TwoLayerNetwork()
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model_two.parameters(), lr=0.0005)

    print("Start training TwoLayerNetwork...")
    results_two = train(model_two, criterion, optimizer, train_loader, valid_loader)
    print("Done!")

    one_train_loss, one_valid_loss, one_train_acc, one_valid_acc = results_one
    two_train_loss, two_valid_loss, two_train_acc, two_valid_acc = results_two

    plt.figure()
    plt.plot(one_train_loss, label='one_train_loss')
    plt.plot(one_valid_loss, label='one_valid_loss')
    plt.plot(two_train_loss, label='two_train_loss')
    plt.plot(two_valid_loss, label='two_valid_loss')
    plt.legend()
    plt.xlabel('epochs')
    plt.ylabel('losses')
    plt.show()

    plt.figure()
    plt.plot(one_train_acc, label='one_train_acc')
    plt.plot(one_valid_acc, label='one_valid_acc')
    plt.plot(two_train_acc, label='two_train_acc')
    plt.plot(two_valid_acc, label='two_valid_acc')
    plt.legend()
    plt.xlabel('epochs')
    plt.ylabel('accuracy')
    plt.show()


    test_loss_onelayer = evaluate_loss(model_one, criterion, test_loader)
    test_acc_onelayer = evaluate_acc(model_one, test_loader)
    test_loss_twolayer = evaluate_loss(model_two, criterion, test_loader)
    test_acc_twolayer = evaluate_acc(model_two, test_loader)
    print('test_loss_onelayer: ', test_loss_onelayer)
    print('test_acc_onelayer: ', test_acc_onelayer.item())
    print('test_loss_twolayer: ', test_loss_twolayer)
```

```
48      print('test_acc_twolayer: ', test_acc_twolayer.item())
```

Submission instructions for programming problems

- Please export the notebook to a `.py` file by clicking the "File" → "Download.py" and upload to CCLE.

  Your code should be commented appropriately. The most important things:
  - Your name and the assignment number should be at the top of each file.
  - Each class and method should have an appropriate doctsring.
  - If anything is complicated, it should include some comments.

  There are many possible ways to approach the programming portion of this assignment, which makes code style and comments very important so that staff can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.

- Please submit all the plots and the rest of the solutions (other than codes) to Gradescope