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: Given that there are N points, there would be ${}_NC_x$ possible label outcomes for x positives (with x being any value from 0 to N inclusive). However, given the condition that for consecutive x's $x_1 \leq x_2 \leq x_3 \leq$ ..., each of these must be one. Thus given that there are N+1 possible values of x, there are only N+1 possible label outcomes.

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

Solution: The VC dimension of H would be 1. VC(H) works for 1 as there would always exist b such that x would be less than or greater than it (so $VC(H) \geq 1$). However, for 2 points, there if we partition it to have a positive example followed by a negative example, no ray classifier would work (and thus, a set of 2 points cannot be shattered, and $VC(H) < 2$). So, VC(H) = 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 $\mathcal{H}$.

Solution: Suppose that we are to count the number of possible sub-segments of the N points set as positive. We would see that there would be N sub-segments if the sub-segment were to start at the first point, N-1 sub-segments starting from the second, etc...

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

until we reach the last point, in which only a sub-segment of length 1 is possible. This would result in the sum of N + (N - 1) + ... + 1, which would be ($\frac{N^2+N}{2}$).
However, this would only count the number of non-zero length sub-segments of consecutive positives. So, we would also need to add the single case in which all points are classified as negative, totaling ($\frac{N^2+N}{2} + 1$).

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

Solution: The VC dimension of H in this case would be 2. A set of 2 points can be partitioned in any way to satisfy the current H (++, +-, -+. −) ($VC(H) \geq 2$). However, a set of any 3 points cannot be partitioned into +-+, so a set of 3 points cannot be shattered properly ($VC(H) < 3$). So, VC(H) = 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: Given that $\mathcal{H}_3 = \mathcal{H}_1 \cap \mathcal{H}_2$, we know that H_3 is a subset of H_1. Every set of points that can be shattered by H_3 can be shattered by H_1, so we know that $VC(H_1) \geq VC(H_3)$.

(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:
H_1 = function that returns true if the 2D coordinates satisfy $((x > a) OR (y > b))$
($VC(H_1) = 1$ since any single point can be classified as either + or - by modifying a and b freely; 2 points cannot be shattered since if we have a point, classify it as +, we cannot classify any point above or to the right of it -; if we are to place the point in the lower-left quadrant, the vice-versa case would correspond to the same issue, meaning 2 points cannot be shattered, so the VC dimension is 1)

H_2 = function that returns true if the 2D coordinates satisfy $((x < c) AND (y < d))$
($VC(H_2) = 1$ since any single point can be classified as either + or - by modifying a and b freely; 2 points cannot be shattered since if we have a point, classify it as +, we cannot classify any point below or to the left of it -; if we are to place the point in the upper-right quadrant, the vice-versa case would correspond to the same issue, meaning 2 points cannot be shattered, so the VC dimension is 1)

H_3 = function that returns true if 2D coordinate adheres to either H_1 or H_2

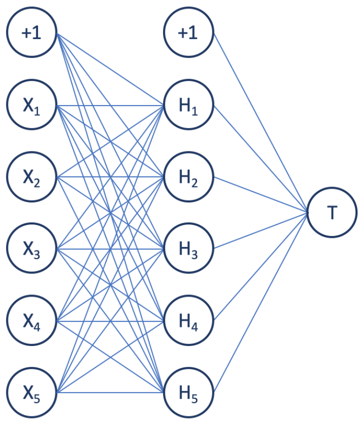
Example: First, we know that $VC(H_1) = 1$ and $VC(H_2) = 1$. Suppose we consider 3 points (-2, 2), (0, -2), and (2, 2). Let's consider each of the possible cases:

(-2, 2)	(0, -2)	(2, 2)	Conditions to Fit
+	+	+	$((a=-3), (b=-3))$
-	+	+	$((a=-1), (b=3), (c=-3), (d=-3))$
+	-	+	$((a=1), (b=0), (c=-3), (d=-3))$
+	+	-	$((a=3), (b=3), (c=1), (d=3))$
+	-	-	$((a=3), (b=3), (c=-1), (d=3))$
-	+	-	$((a=1), (b=-1), (c=3), (d=3))$
-	-	+	$((a=1), (b=3), (c=-3), (d=-3))$
-	-	-	$((a=3), (b=3), (c=-3), (d=-3))$

Thus, this group of 3 points is shatterable under H_3, so $VC(H_3) \geq 3$, which satisfies $VC(\mathcal{H}_1) + VC(\mathcal{H}_2) < VC(\mathcal{H}_3)$ $(1+1<3)$.

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:
(assuming $sign(0) = 1$)
$w_{0j} = 5 - 2j$
$w_{ij} = 1$ for $i \neq 0$ and all j

(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:
(again, assuming $sign(0) = 1$)
$v_0 = -5$
$v_1 = 1$
$v_i = -1$ for $i > 1$

(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: The above answer works because for the entire network to work like an XOR for all values, we need to have X_1 be 1 and the other X's be -1, resulting in a max value of $-5 + 1 - (-1) - (-1) - (-1) - (-1) = 0$. If none of the X's are set, then the value would be $-5 + (-1) - (-1) - (-1) - (-1) - (-1) = -2$. If there are 2 Xs' set, the value would be $-5 + 1 - (1) - (-1) - (-1) - (-1) = -2$. Setting more X's would further decrease the value by 2, which would end up being negative. With only a single X resulting in a positive value, the function acts like an XOR between all X's.

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.

$\#\#\# ========= TODO : START ========= \#\#\#$

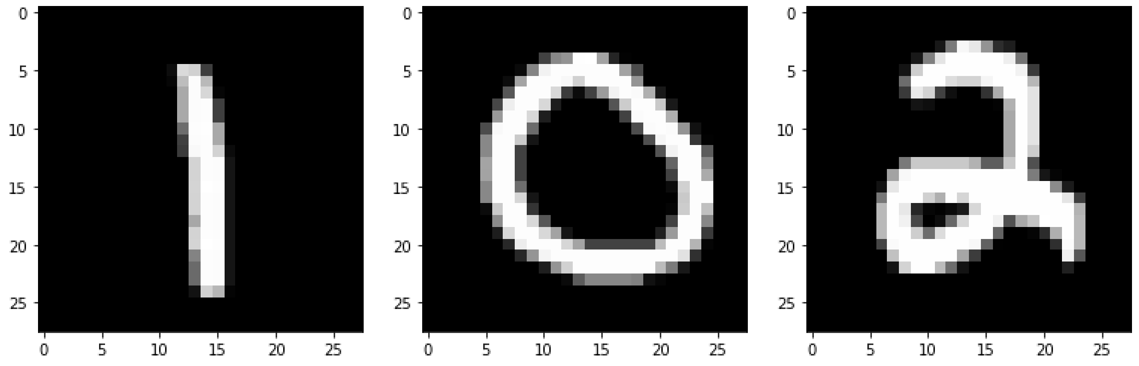
$\#\#\# ========= TODO : END ========== \#\#\#$

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: (figures X_train[0], X_train[1], and X_train[7])



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

Solution: (See Code)

(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: (See Code)

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: (See Code)

(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: (See Code)

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: (See Code)

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: (See Code)

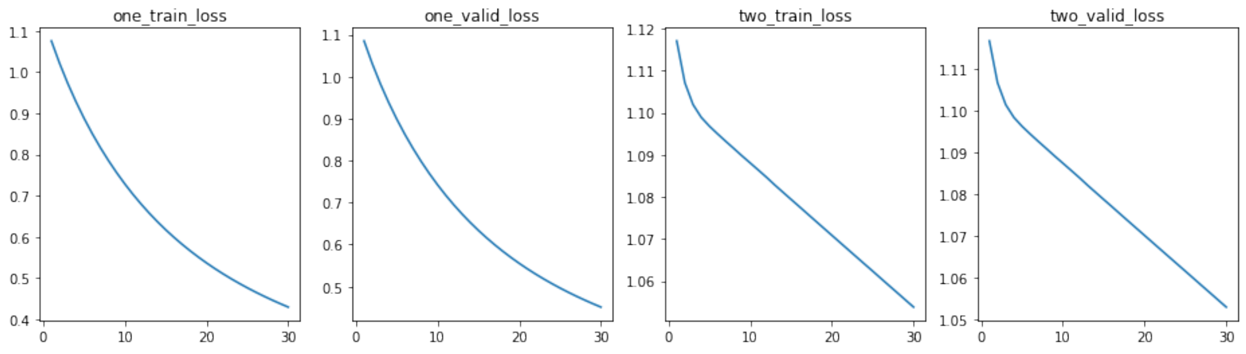
(h) Create an instance of `TwoLayerNetwork`, set up a criterion with `torch.nn.CrossEntropyLoss`, and set up a SGD optimizer with learning rate 0.0005 by using `torch.optim.SGD`. Then train `TwoLayerNetwork`. [2 pts]

Solution: (See Code)

Performance Comparison [16 pts]

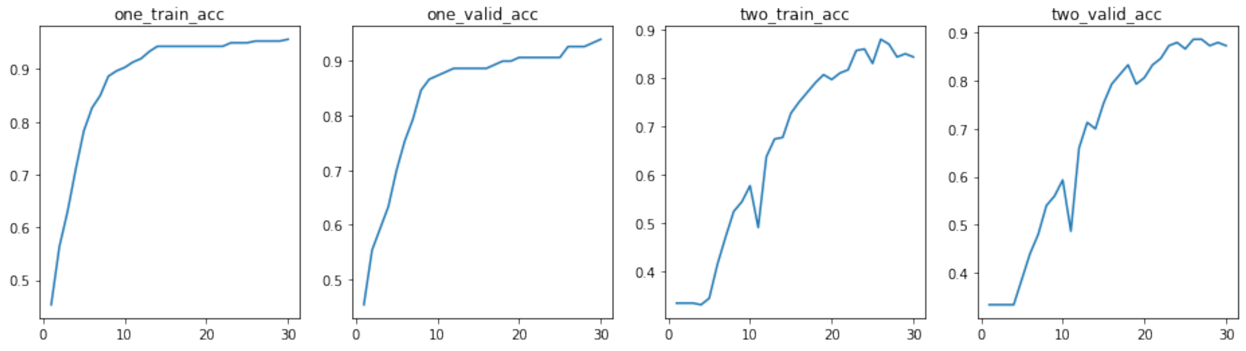
(i) Generate a plot depicting how `one_train_loss`, `one_valid_loss`, `two_train_loss`, `two_valid_loss` varies with epochs. Include the plot in the report and describe your findings. [3 pts]

Solution:



(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:



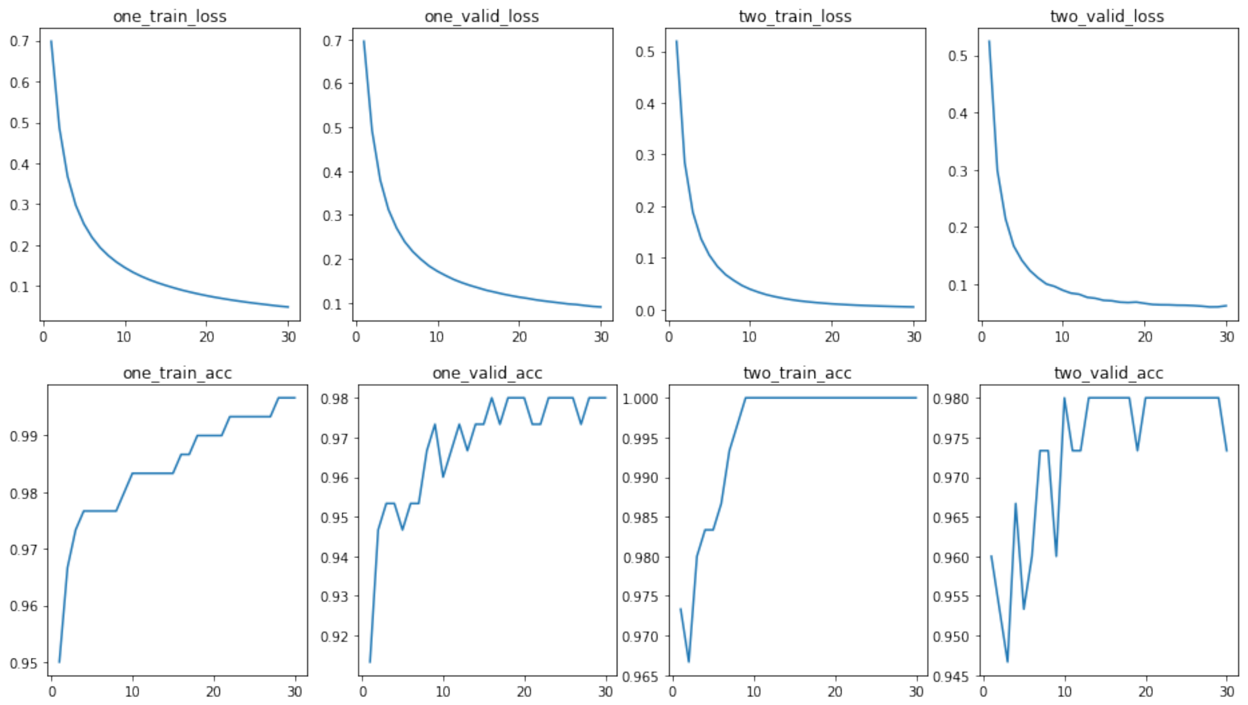
(k) Calculate and report the test accuracy of both the one-layer network and the two-layer network. Explain why we get such results. [3 pts]

Solution:
one-layer network test accuracy: 0.960000
two-layer network test accuracy: 0.866667
When we look at how the training accuracy grows for both cases, we would see that the two-layer network increases more slowly and erratically, likely due to the fact for the complexity of how prior layers can affect subsequent layers of computation. Given that the training was only done for 30 iterations, it would be somewhat reasonable to surmise that the the two-layer network would have a smaller test accuracy due to more difficulty in the training.

(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:



one-layer network test accuracy: 0.966667
two-layer network test accuracy: 0.966667
In general, the loss curve decreases more steeply (in an exponential fashion), and the training accuracy for two-layers increases more quickly compared to the SGD optimizer (although one-layer accuracy increases less quickly than both two-layer Adam and onw-layer SGD). The test accuracy for both cases ended up higher than the SGD cases (especially with the large test accuracy increase for the two-layer case). This is reasonable, as the Adam optimizer makes use of an adaptive learning rate (as opposed to the static learning rate for SGD). This explains why the loss decrease and accuracy increase for the two-layer case is quicker compared to the corresponding SGD case. The learning rate seems to be higher at the beginning compared to SGD due to the larger losses and lower accuracy, and adapts as the model learns more as more epochs pass.

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