Approximation

Question 1

(1) $\mathbf{W}_2 \in \mathbb{R}^{1\times k}$,$\mathbf{b}_1 \in \mathbb{R}^{k\times 1}$, $\mathbf{b}_2 \in \mathbb{R}^{1\times 1}$

Question 2

(2) a network with 3-layer ReLU network can be expressed as below:

$$f(\mathbf{x}; \mathbf{W_1}, \mathbf{W_2}, \mathbf{W_3}, \mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3}) = \mathbf{W_3}\mathrm{ReLU}(\mathbf{W_2}\mathrm{ReLU}(\mathbf{W_1}\mathbf{x} + \mathbf{b_1}) + \mathbf{b_2}) + \mathbf{b_3}$$

Question 3

(3) Convex: No, Concave: No

Question 4

(4)(a)

number of discontinuities $\leq (2k)^l$

(4)(b)

(B) piecewise-linear

(4)(c) NO

(4c)(i)

0， If all inputs to every ReLU activation function in the network are strictly positive. $\mathrm{ReLU}(x) = \max(0, x)$ is fully differentiable wherever $x > 0$

(4c)(ii)

consider the width of each layer is k, the largest number of input points at which the function can be non-differentiable will be $(2k)^l$, which is $4k^2$ in this case

(4c)(iii)

(D) exponential in l, as it is related to the maximum number of discontinuities that may present in the ReLU network approximation

(4c)(iv)

Suppose two networks contain same number of kinks/ discontinuouties:

- For a two-layer network, $N_1 = 2k$, indicating total unit number with k as layer width
- For a deeper network with l layers, where the total unit number is N_2 and each layer has k' units derived from the relation $(2k)^2 = (2k')^l$, where we suppose two networks have same number of 'kinks',the equation simplifies to:

$$4k^2 = (2k')^l$$

Solving for k':

$$k' = \left(\frac{4k^2}{2^l}\right)^{\frac{1}{l}}$$

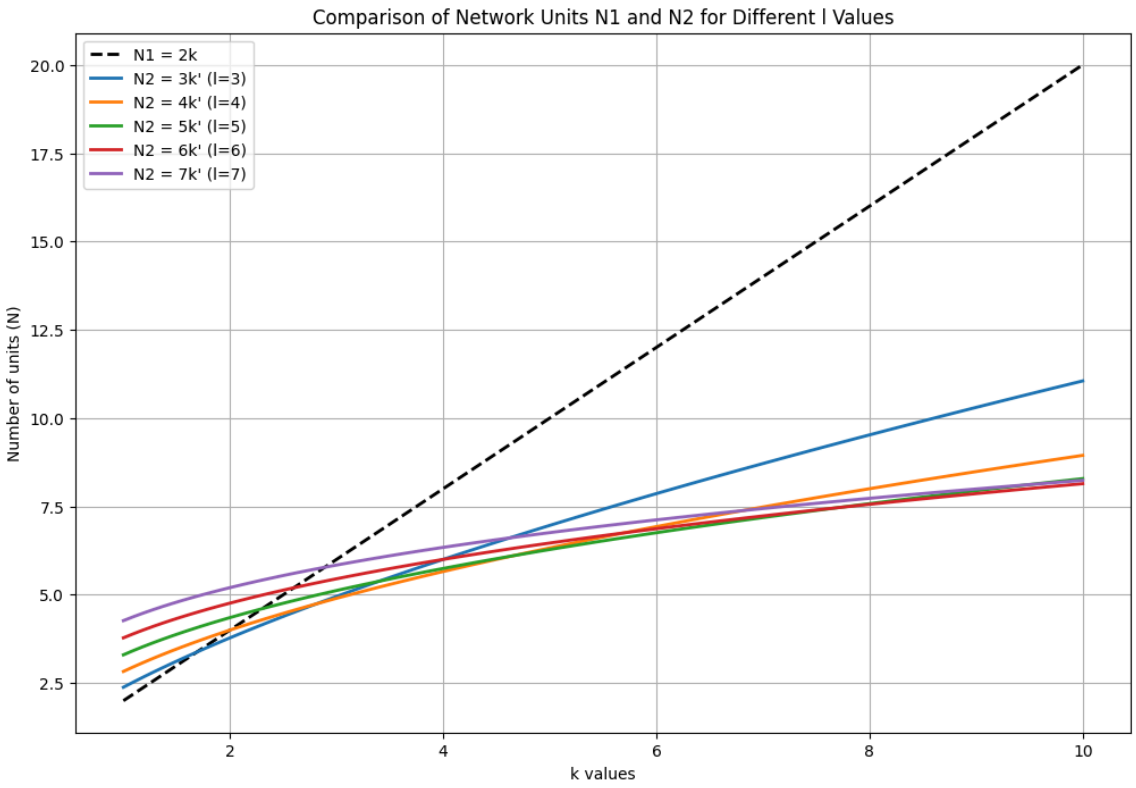
$$k' = 2^{2/l} k^{2/l} 2^{-1}$$

$$k' = 2^{\frac{2-l}{l}} k^{\frac{2}{l}}$$

Thus, N_2 is given by:

$$N_2 = l \times k' = l \times 2^{\frac{2-l}{l}} k^{\frac{2}{l}}$$

Plot Explanation

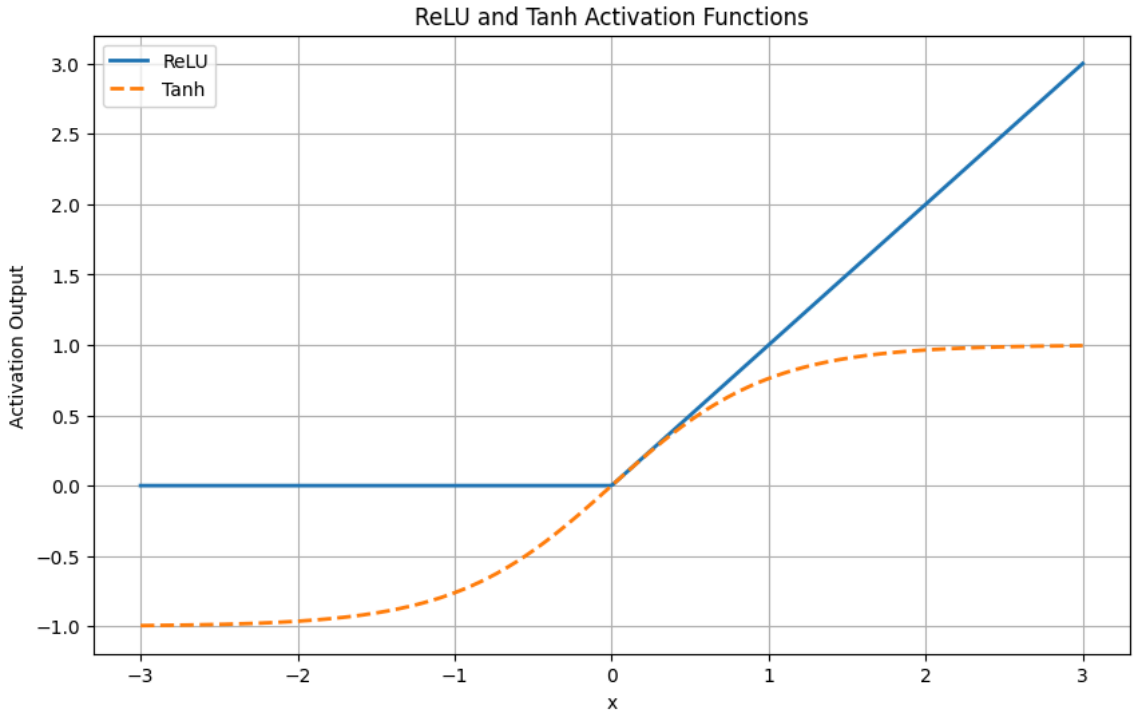


The plot compares the number of units N_1 and N_2 for different values of k, with l set to values greater than 2 (3 through 7). The key observations are:

- The line $N_1 = 2k$ represents the linear increase in units with k in a two-layer network.
- The curves for N_2, calculated for increasing l, all lie below N_1, indicating that deeper networks require fewer units to achieve the same number of kinks or discontinuities as k increases.
- As l increases, the efficiency of the deeper networks in terms of total units required improves significantly.

(4)(d)

Yes. below is the plot of ReLU and Tanh, you can see the Tanh is continuous at everywhere while ReLU has a 'kink' at $x = 0$



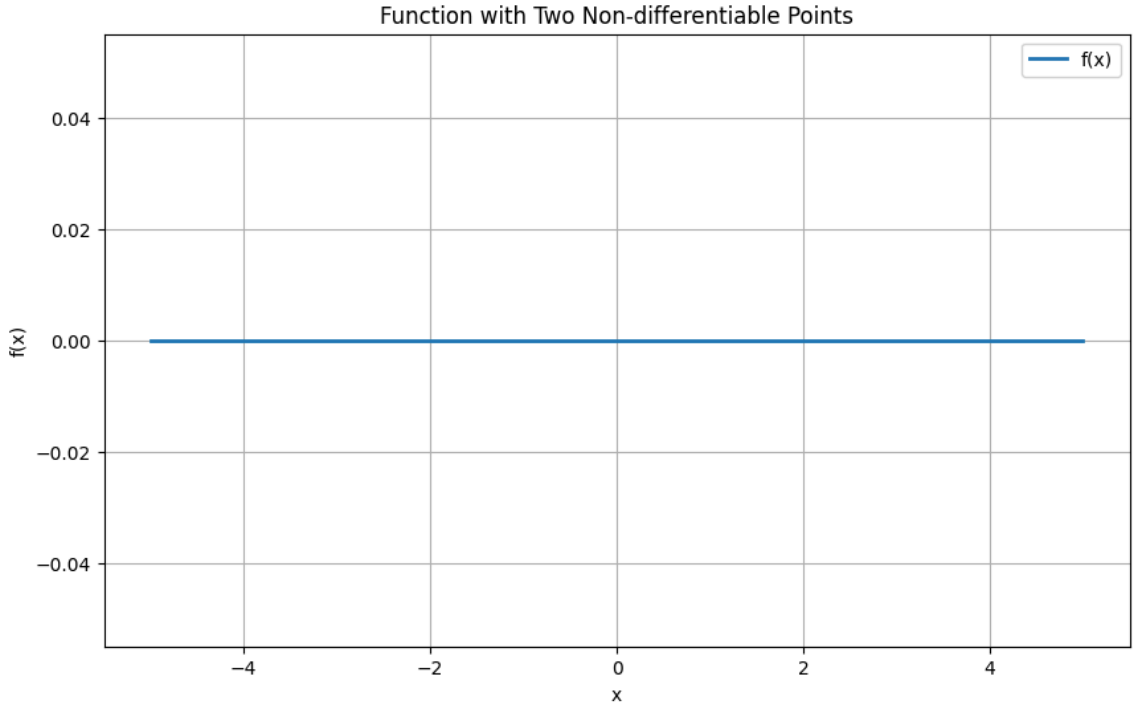
Question 5

(5) (a)

$$W_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

see below plot of $f(x)$



(5) (b)

No such $\mathbf{W_1}$ and $\mathbf{W_2}$.

Given the weight vectors $\mathbf{W_1} = \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix}$ and $\mathbf{W_2} = \begin{bmatrix} w_{21} & w_{22} \end{bmatrix}$, the function $f(x)$ for a neural network using ReLU activations can be broken down as follows:

$$f(x) = \mathbf{W_2} \cdot \mathrm{ReLU}(\mathbf{W_1} \cdot x)$$

Expanding the multiplication and ReLU operation, we have:

$$f(x) = \begin{bmatrix} w_{21} & w_{22} \end{bmatrix} \cdot \begin{bmatrix} \mathrm{ReLU}(w_{11} \cdot x) \\ \mathrm{ReLU}(w_{12} \cdot x) \end{bmatrix}$$

$$f(x) = w_{21} \cdot \mathrm{ReLU}(w_{11} \cdot x) + w_{22} \cdot \mathrm{ReLU}(w_{12} \cdot x)$$

This formulation uses the ReLU function, defined as:

$$\mathrm{ReLU}(z) = \max(0, z)$$

Therefore, $f(x)$ will be non-differentiable at points where $w_{11} \cdot x = 0$ or $w_{12} \cdot x = 0$ if these points are different. But without a bias term, the $f(x)$ will have at most 1 kink at $x = 0$.

Question 7

(7) (a)

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

if we choose $\mathbf{W_1}, \mathbf{W_2}, \mathbf{b_1}, \mathbf{b_2}$ as follows:

$$\mathbf{W_1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathbf{b_1} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\mathbf{W_2} = \begin{bmatrix} 1 & 1 \end{bmatrix}, b_2 = 0$$

$$f(\mathbf{x}) = \mathbf{W_2} \cdot \mathrm{ReLU}(\mathbf{W_1} \cdot \mathbf{x} + \mathbf{b_1}) + \mathbf{b_2} = \mathrm{ReLU}(x_1) + \mathrm{ReLU}(x_2)$$

This means $f(\mathbf{x}) > 0$ if either $x_1 > 0$ or $x_2 > 0$

(7) (b)

The required XOR gate can be constructed by 3-layer ReLU network with concatenation trick in the middle. See below break-downs:

First Layer ($\mathbf{W}_1$ and $\mathbf{b}_1$):

$$\mathbf{W}_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad \mathbf{b}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Second Layer ($\mathbf{W}_2$ and $\mathbf{b}_2$):

$$\mathbf{W}_2 = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Third Layer ($\mathbf{W}_3$ and b_3):

$$\mathbf{W}_3 = \begin{bmatrix} 1 & 0 & 1 & 0 & -1 & -1 \end{bmatrix}, \quad b_3 = 0$$

Concatenation Operation:

After computing the activations from the first and second layers, we obtain:

- **First Layer Output ($\mathbf{s}$):** A 4×1 vector.
- **Second Layer Output ($\mathbf{s}'$):** A 2×1 vector.

We **concatenate** these vectors to form a single 6×1 vector, which serves as the input to the third layer:

$$\begin{bmatrix} \mathbf{s} \\ \mathbf{s}' \end{bmatrix} = \begin{bmatrix} \text{ReLU}(-x_1) \\ \text{ReLU}(x_2) \\ \text{ReLU}(x_1) \\ \text{ReLU}(-x_2) \\ \text{ReLU}(\text{ReLU}(-x_1) - \text{ReLU}(x_2)) \\ \text{ReLU}(\text{ReLU}(x_1) - \text{ReLU}(-x_2)) \end{bmatrix}$$

Algebraic Formula of $f(\mathbf{x})$:

$$\begin{aligned} f(\mathbf{x}) = \text{ReLU}(-x_1) + \text{ReLU}(x_1) \\ - \text{ReLU}\left(\text{ReLU}(-x_1) - \text{ReLU}(x_2)\right) \\ - \text{ReLU}\left(\text{ReLU}(x_1) - \text{ReLU}(-x_2)\right) \end{aligned}$$

Proof for the XOR Gate Conditions:

1. **Case 1:** $x_1 \geq 0$**,** $x_2 \geq 0$
   - $\text{ReLU}(x_1) = x_1 \geq 0, \text{ReLU}(-x_1) = 0$
   - $\text{ReLU}(x_2) = x_2 \geq 0, \text{ReLU}(-x_2) = 0$
   - Compute the intermediate terms:

     $$\text{ReLU}(\text{ReLU}(-x_1) - \text{ReLU}(x_2)) = \text{ReLU}(0 - x_2) = 0$$
     $$\text{ReLU}(\text{ReLU}(x_1) - \text{ReLU}(-x_2)) = \text{ReLU}(x_1 - 0) = x_1 \geq 0$$

   - $f(\mathbf{x})$ simplifies to:

     $$f(\mathbf{x}) = 0 + x_1 - 0 - x_1 = 0$$

   - **Conclusion:** $f(\mathbf{x}) = 0$; XOR condition **not** satisfied.
2. **Case 2:** $x_1 > 0$**,** $x_2 < 0$
   - $\text{ReLU}(x_1) = x_1 > 0, \text{ReLU}(-x_1) = 0$
   - $\text{ReLU}(x_2) = 0, \text{ReLU}(-x_2) = -x_2 > 0$
   - Compute the intermediate terms:

     $$\text{ReLU}(\text{ReLU}(-x_1) - \text{ReLU}(x_2)) = \text{ReLU}(0 - 0) = 0$$
     $$\text{ReLU}(\text{ReLU}(x_1) - \text{ReLU}(-x_2)) = \text{ReLU}(x_1 - (-x_2)) = \text{ReLU}(x_1 + x_2)$$

   - Since $x_2 < 0$, $x_1 + x_2 < x_1$. The exact value of $\text{ReLU}(x_1 + x_2)$ depends on x_1 and x_2.
   - $f(\mathbf{x})$ becomes:

     $$f(\mathbf{x}) = 0 + x_1 - 0 - \text{ReLU}(x_1 + x_2)$$

   - Since $\text{ReLU}(x_1 + x_2) \leq x_1$, $f(\mathbf{x}) \geq 0$.
   - **Conclusion:** $f(\mathbf{x}) > 0$; XOR condition satisfied.
3. **Case 3:** $x_1 < 0$**,** $x_2 > 0$
   - $\text{ReLU}(x_1) = 0, \text{ReLU}(-x_1) = -x_1 > 0$

- $\text{ReLU}(x_2) = x_2 > 0, \text{ReLU}(-x_2) = 0$
- Compute the intermediate terms:

$$\text{ReLU}(\text{ReLU}(-x_1) - \text{ReLU}(x_2)) = \text{ReLU}(-x_1 - x_2)$$

- Since $-x_1 > 0$ and $x_2 > 0$, $-x_1 - x_2 < 0$, so:

$$\text{ReLU}(-x_1 - x_2) = 0$$

- $f(\mathbf{x})$ simplifies to:

$$f(\mathbf{x}) = -x_1 + 0 - 0 - 0 = -x_1 > 0$$

- **Conclusion:** $f(\mathbf{x}) > 0$; XOR condition satisfied.

4. **Case 4:** $x_1 \leq 0$**,** $x_2 \leq 0$

- $\text{ReLU}(x_1) = 0, \text{ReLU}(-x_1) = -x_1 \geq 0$
- $\text{ReLU}(x_2) = 0, \text{ReLU}(-x_2) = -x_2 \geq 0$
- Compute the intermediate terms:

$$\text{ReLU}(\text{ReLU}(-x_1) - \text{ReLU}(x_2)) = \text{ReLU}(-x_1 - 0) = -x_1 \geq 0$$
$$\text{ReLU}(\text{ReLU}(x_1) - \text{ReLU}(-x_2)) = \text{ReLU}(0 - (-x_2)) = \text{ReLU}(x_2) = 0$$

- $f(\mathbf{x})$ simplifies to:

$$f(\mathbf{x}) = -x_1 + 0 - (-x_1) - 0 = 0$$

- **Conclusion:** $f(\mathbf{x}) = 0$; XOR condition **not** satisfied.

Summary:

- $f(\mathbf{x}) > 0$ **if and only if** $(x_1 < 0 \text{ and } x_2 > 0)$ **or** $(x_1 > 0 \text{ and } x_2 < 0)$.

Backpropagation

Question 8

(8) (a)

$$\mathcal{L} = \frac{1}{2}\|\mathbf{v}\|_2^2 = \frac{1}{2}\sum_{m=1}^{d}\mathbf{v}_m^2$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{ij}} = \sum_{m=1}^{d}\frac{\partial \mathcal{L}}{\partial \mathbf{v}_m}\frac{\partial \mathbf{v}_m}{\partial \mathbf{W}_{ij}} = \sum_{m=1}^{d}\frac{\partial\left(\frac{1}{2}\mathbf{v}_m^2\right)}{\partial \mathbf{v}_m}\frac{\partial \mathbf{v}_m}{\partial \mathbf{W}_{ij}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{ij}} = \sum_{m=1}^{d} \mathbf{v}_m \frac{\partial \mathbf{v}_m}{\partial \mathbf{W}_{ij}}$$

(8) (b)

Given the equations and setup:

- $\mathbf{y} = \mathbf{W}\mathbf{x}$
- $\mathbf{u} = \mathrm{ReLU}(\mathbf{y})$
- $\mathbf{v} = \mathbf{u} + \mathbf{W}\mathbf{u}$
- $\mathcal{L} = \frac{1}{2}\|\mathbf{v}\|_2^2$

Derivation

1. **Evaluate $\frac{\partial u_m}{\partial \mathbf{W}_{ij}}$:**

$$\frac{\partial u_m}{\partial \mathbf{W}_{ij}} = \sum_{k=1}^{d} \frac{\partial u_m}{\partial y_k} \frac{\partial y_k}{\partial \mathbf{W}_{ij}} = \Theta(y_i) \cdot \delta_{mi} \cdot x_j$$

   This means $\frac{\partial u_m}{\partial \mathbf{W}_{ij}}$ is nonzero only when $m = i$ and $y_i \geq 0$, resulting in:

$$\frac{\partial u_m}{\partial \mathbf{W}_{ij}} = \Theta(y_i) \cdot x_j \text{ when } m = i$$

2. **Plugging this into $\frac{\partial v_m}{\partial \mathbf{W}_{ij}}$:**

$$\frac{\partial v_m}{\partial \mathbf{W}_{ij}} = \Theta(y_i) \cdot x_j \cdot \delta_{mi} + \delta_{im} \cdot u_j + \sum_{l=1}^{d} \mathbf{W}_{ml} \Theta(y_i) \cdot x_j \cdot \delta_{li}$$

   Simplify to:

$$\frac{\partial v_m}{\partial \mathbf{W}_{ij}} = \Theta(y_i) \cdot x_j \cdot \delta_{mi} + \delta_{im} \cdot u_j + \mathbf{W}_{mi} \Theta(y_i) \cdot x_j$$

3. **Sum up for $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{ij}}$:**

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{ij}} = \sum_{m=1}^{d} v_m \left(\Theta(y_i) \cdot x_j \cdot \delta_{mi} + \delta_{im} \cdot u_j + \mathbf{W}_{mi} \Theta(y_i) \cdot x_j \right)$$

   Combine terms:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{ij}} = v_i \Theta(y_i) \cdot x_j + u_i \cdot v_i + \left(\sum_{m=1}^{d} v_m \mathbf{W}_{mi} \right) \Theta(y_i) \cdot x_j$$

The expression matches the provided matrix form:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \mathbf{v} \otimes \mathbf{u} + \text{diag}(\Theta(\mathbf{y}))(\mathbf{I} + \mathbf{W}^T)\mathbf{v} \otimes \mathbf{x}$$

CIFAR-10 Classification

Question 10

(10)(a) Linear.forward

```
return torch.matmul(x, self.weight.T) + self.bias
```

(10)(b) Linear.backward

```
self.weight.grad = torch.matmul(dLdout.T, self.inputs[0])
    self.bias.grad = dLdout.sum(0)
    dLdin = torch.matmul(dLdout, self.weight)  # [b,in_features]
    return dLdin
```

(10)(c) ReLU.backward

```
dLdin = (self.inputs[0] > 0).float() * dLdout
    return dLdin
```

(10)(d) CrossEntropyLoss.backward

```
p = logits.softmax(dim=-1)
    target_one_hot = torch.zeros_like(p)
    target_one_hot[torch.arange(logits.shape[0]), target] = 1
    dLdlogits = (p - target_one_hot) / logits.shape[0]
    return dLdlogits
```

Question 11

(11) Training loop

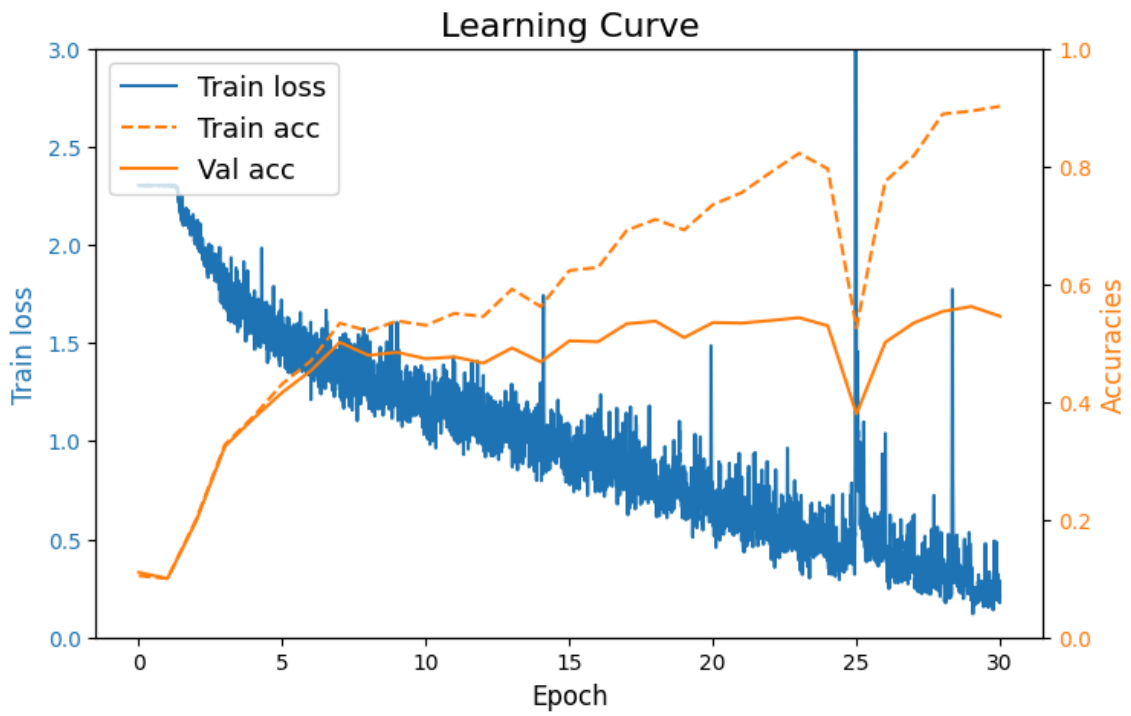
in train_epoch:

```
for p in model.parameters():
        if p.grad is not None:
            p.data -= lr * p.grad
```

in evaluate:

```
output = model(data)
        pred = torch.argmax(output, dim=1)
        correct_predictions += (pred == target).sum().item()
    return correct_predictions / len(loader.dataset)
```

(12) Training Curve



observations:

1. The training loss shows a consistent decline and converges in the end, indicating the model is efectively learning.
2. The training accuracy reachs approximately 90%, suggesting the success of classifying training data. Despite improvements in training accuracy, the validation accuracy remains relative flat and much slower, reaching 55% in the end.
3. The substantial gap between high training accuracy and significantly lower validation accuracy suggests that the model is overfitting the training data. The model learns details and noise from the training data, therefore it can not generalize well on unseen data, as reflected by the validation accuracy

(13) Universal Approximation

The model may not have achieved perfect training accuracy primarily because the complexity of a simple MLP is often insufficient to capture the intricate patterns and spatial hierarchies present in image data. To increase the training accuracy we could consider below approaches:

1. Adding more layers to the MLP could help increase its capacity to learn more complex features from the data.
2. Incorporating convolutional layers can significantly enhance the model's ability to process image data by utilizing its spatial relationships.

3. Implementing sinusoidal positional encodings as used in SIREN models can help in learning fine-grained details and patterns in two-dimensional data spaces.

Achieving perfect training accuracy is not the ultimate goal, especially as it often leads to overfitting, where the model learns the training data, including its noise and outliers, too well. This overfitting typically results in poor generalization to new, unseen data, making the model less effective in practical applications. Therefore, a balance between high training accuracy and the ability to generalize well is crucial for creating robust machine learning models.

(14) Data Augmentation

Code

```
train_transforms = [
    transforms.Pad(padding=3, padding_mode="reflect"),
    transforms.RandomCrop(32),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4914, 0.4822, 0.4465], std=[0.2470,
0.2435, 0.2616]),
]
val_transforms = [
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4914, 0.4822, 0.4465], std=[0.2470,
0.2435, 0.2616]),
]

aug_train_set, val_set = get_datasets(
    train_transforms=train_transforms, val_transforms=val_transforms
)
```

Visualization

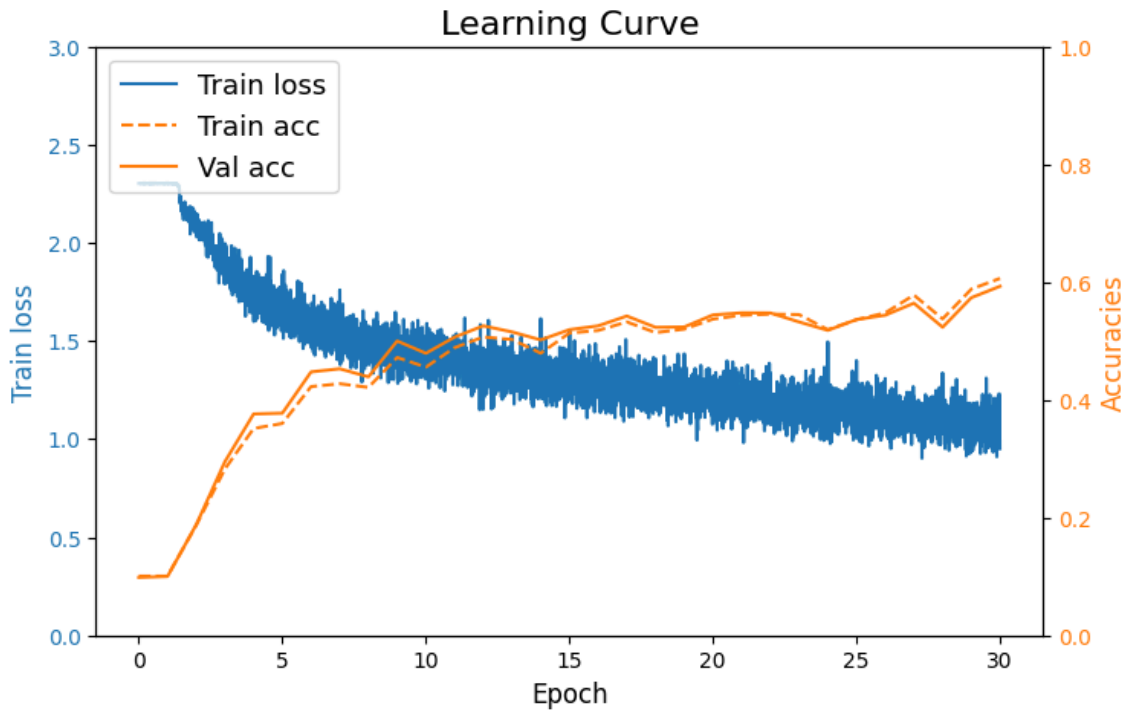
Label = bird

Label = bird

Label = bird

Label = bird

(15) Training Curve



Observations:

1. Narrower Gap Between Training and Validation Accuracy:Both accuracies are now hovering around 60%. This suggests that the model is generalizing better than before, where there was a significant discrepancy (training accuracy around 90% and validation accuracy around 55%).
2. Reduced Training Accuracy:The training accuracy has dropped significantly compared to the earlier scenario (from about 90% to around 60%). This reduction in training accuracy is a typical effect of data augmentation, as it makes the training task harder and less prone to memorizing the exact training samples due to the increased variability in the input data. The improved generalization is at the cost of potential underfitting.
3. Smoother Curve: The training loss curve appears smoother and more stable across epochs compared to the previous curve, which showed more fluctuation.