Due Date : March 1st, 23:00 2024

Instructions

- *For all questions, show your work! Any problem without work shown will get no marks regardless of the correctness of the final answer.*
- *Use a document preparation system such as LaTeX. If hand-written, you may lose marks if your writing is illegible without any possibility of regrade.*
- *Submit your answers electronically via the course GradeScope.*
- *Please sign the agreement below.*
- *It is your responsibility to follow updates to the assignment after release. All changes will be visible on Overleaf and Piazza, so no additional time or exceptions will be granted.*
- *Any questions should be directed towards the TA for this assignment (theoretical part) :* ***Jerry Huang****.*

I acknowledge I have read the above instructions and will abide by them throughout this assignment. I further acknowledge that any assignment submitted without the following form completed will result in no marks being given for this portion of the assignment.

Signature : [illegible]

Name : Mahmood Hegazy

Student ID : 20273929

Preliminaries For this assignment, you will need to make use of the following definitions

Definition 1 (Convex Function)**.** A function $f : \mathbb{R}^d \to \mathbb{R}^n$ is convex if for any $\alpha \in (0, 1)$ and $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$, then

$$f(\alpha \mathbf{x}_1 + (1 - \alpha)\mathbf{x}_2) \leq \alpha f(\mathbf{x}_1) + (1 - \alpha) f(\mathbf{x}_2)$$

Definition 2 (Lipschitzness)**.** A function $f : \mathbb{R}^d \to \mathbb{R}^n$ is ρ-Lipschitz over a set $\mathbb{S} \subset \mathbb{R}^d$ if for any $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{S}$

$$\|f(\mathbf{w}_1) - f(\mathbf{w}_2)\| \leq \rho \|\mathbf{w}_1 - \mathbf{w}_2\|$$

Definition 3 (Smoothness)**.** A function f is β-smooth if it's gradient is β-Lipschitz.

Basics (14 points)

Question 1. 1. .

- f' is strictly increasing.
  It is a **Sufficient** Condition but **Not Necessary**. If f is convex, then f' is monotonically increasing. Strict monotonicity (strictly increasing) is not necessary for convexity ; f' can be non-decreasing and still satisfy convexity. Moreover, a strictly increasing f' ensures that the slope of the tangent line to f increases across its domain, which is sufficient for f to be convex, as it ensures the graph of f lies below its tangent lines.
  - Consider two points a and b in the domain of f such that $a < b$.
  - By the Mean Value Theorem, there exists a point c between a and b s.t. :
  $$f'(c) = \frac{f(b) - f(a)}{b - a}$$
  - Since f' is strictly increasing, we have $f'(c) \geq f'(a)$
  - Therefore, $f(b) - f(a) \geq f(a)(b - a)$
  - Rearranging, we get $f(b) \geq f(a) + f(a)(b - a)$
  - This inequality corresponds to the definition of convexity and therefore it is a sufficient condition that $f(x)$ is strictly increasing
- $f'' \geq 0$.
  It is a **Necessary** and **Sufficient** Condition. The condition $f'' \geq 0$ is both necessary and sufficient for f to be convex. This criterion follows from the definition of convexity in terms of second derivatives, indicating upward or linear curvature of f's graph.
  - If $f(x) \geq 0$ for all x, then the graph of f lies above its tangent lines, which is consistent with convexity.
  - Conversely, if $f(x) < 0$ for any x, the graph of f would curve downward, violating convexity. It is **necessary** because without it, f could potentially be concave in some regions. It is **sufficient** because it directly ensures convexity through its impact on the slope of f.
- For all points x where x is a local minimum, x is also a global minimum.
  It is a **Necessary** condition but **Not Sufficient**.
  - This statement is necessary for convex functions because a convex function can have at most one local minimum, which, if it exists, is also the global minimum. This property arises from the definition of convexity, which implies that the function's value at any point is at or above the line segment connecting any two points in the function's domain. Therefore, if a local minimum exists, no other point can have a lower value, making it a global minimum as well.
  - While this property is characteristic of convex functions, it is not sufficient to define convexity. Other types of functions might have the property that every local minimum is a global minimum without being convex. For example, a function with a flat bottom (not strictly convex) might satisfy this condition without meeting the strict definition of convexity. For an example of such a function consider the piecewise function defined by :
  $$f(x) = \begin{cases} (x+2)^2 & \text{for } x < -2 \\ 0 & \text{for } -2 \leq x \leq 2 \\ (x-2)^2 & \text{for } x > 2 \end{cases}$$

This function exhibits a flat bottom in the interval $[-2, 2]$, where it is constant and hence not strictly convex. However, it is convex overall because it is composed of convex parts outside the interval, and the flat section does not break the overall convexity. The lack of strict convexity arises from the flat section, where the function does not fulfill the condition required for strict convexity : for any two points in the flat section, the line segment between them lies entirely on the graph, not strictly above it except at the endpoints.

2. The necessary conditions for convexity are as follows :
   - The first derivative ($f'(x)$) must be non-decreasing.
   - The second derivative ($f''(x)$) must be non-negative.

   (a) $f(x) = x^2$ **is convex**. Proof :
   - First derivative : $f'(x) = 2x$
   - Since $f'(x) = 2x$ is non-decreasing for all $x \in \mathbb{R}$, the first condition is satisfied.
   - Second derivative : $f''(x) = 2$
   - Since $f''(x) = 2 > 0$ for all $x \in \mathbb{R}$, the function $f(x) = x^2$ is convex over $\mathbb{R}$.

   (b) ($f(x) = \log(1 + \exp(x))$) **is convex**. Proof :
   - First derivative : $f'(x) = \frac{\exp(x)}{1+\exp(x)}$
   - The first derivative is strictly increasing for all ($x \in \mathbb{R}$), satisfying the first condition.
   - Second derivative : $f''(x) = \frac{\exp(x)}{(1+\exp(x))^2}$
   - Given that $\exp(x) > 0$ for all x, and $(1 + \exp(x))^2 > 0$, it follows that $f''(x) > 0$ for all $x \in \mathbb{R}$. Therefore, the function $f(x) = \log(1 + \exp(x))$ is convex over $\mathbb{R}$.

3. A function ($f : \mathbb{R} \to \mathbb{R}$) is Lipschitz continuous if there exists a positive constant (ρ) (the Lipschitz constant) such that for all $x, y \in \mathbb{R}$, we have :

$$|f(x) - f(y)| \leq \rho|x - y|$$

   The smallest such ρ is often related to the maximum of the absolute value of the function's first derivative (if it exists and is continuous) over the domain of interest, which in this case is $\mathbb{R}$

   (a) $f(x) = x^2$
   - First derivative : $f'(x) = 2x$
   - The Lipschitz constant ρ would be the maximum of $|2x|$ over $\mathbb{R}$. Since $|2x|$ increases without limit as $|x|$ increases, $f(x) = x^2$ is not ρ-Lipschitz for any finite ρ over the entire real line.

   (b) $f(x) = \log(1 + \exp(x))$
   - First derivative : $f'(x) = \frac{\exp(x)}{1+\exp(x)}$
   - The maximum of the absolute value of $f'(x)$ over $\mathbb{R}$ is 1, since $0 < \frac{\exp(x)}{1+\exp(x)} < 1$ for all x.
   - Therefore, $f(x) = \log(1 + \exp(x))$ is 1-Lipschitz over $\mathbb{R}$; $\rho = 1$.

4. To determine the minimum β for which $f(x) = x^2$ and $f(x) = \log(1 + \exp(x))$ are smooth over $\mathbb{R}$ we need to use the concept of a smooth function in the context of Lipschitz continuity of derivatives. A function $f : \mathbb{R}^d \to \mathbb{R}^n$ is said to be β-smooth on $\mathbb{R}$ if its derivative f' is β-Lipschitz meaning there exists a real number $\beta \geq 0$ such that for all $x, y \in \mathbb{R}$:

$$|f'(x) - f'(y)| \leq \beta |x - y|$$

Again, the smallest such β is often related to the maximum of the absolute value of the function's second derivative (if it exists and is continuous) over $\mathbb{R}$

(a) Function $f(x) = x^2$
   - First derivative : $f'(x) = 2x$
   - Second derivative : $f''(x) = 2$
   - Since $f''(x) = 2$ is constant over $\mathbb{R}$, the function $f(x) = x^2$ is 2-smooth over $\mathbb{R}$; $\beta = 2$.

(b) Function $f(x) = \log(1 + \exp(x))$
   - First derivative : $f'(x) = \frac{\exp(x)}{1+\exp(x)}$
   - Second derivative : $f''(x) = \frac{\exp(x)}{(1+\exp(x))^2}$
   - The maximum of $|f''(x)|$ over $\mathbb{R}$ occurs at $x = 0$, yielding $f''(0) = \frac{1}{4}$.
   - Therefore, $f(x) = \log(1 + \exp(x))$ is $\frac{1}{4}$-smooth over $\mathbb{R}$; $\beta = \frac{1}{4}$.

5. Given :
   - The hinge loss function $\ell(\hat{y}, y) = \max(0, 1 - y \cdot \hat{y})$.
   - Predictions $\hat{y}_1 = \langle \mathbf{x}_1, \mathbf{w} \rangle$ and $\hat{y}_2 = \langle \mathbf{x}_2, \mathbf{w} \rangle$, with $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$ where $\mathcal{X} = \{\mathbf{x}' : \|\mathbf{x}'\|_2 \leq R\}$.

   We need to show that :

$$|\ell(\hat{y}_1, y) - \ell(\hat{y}_2, y)| \leq R \|\mathbf{x}_1 - \mathbf{x}_2\|_2$$

Proof :

- The absolute difference in losses can be expressed as :

$$|\ell(\hat{y}_1, y) - \ell(\hat{y}_2, y)| = |\max(0, 1 - y \cdot \hat{y}_1) - \max(0, 1 - y \cdot \hat{y}_2)|$$

- The maximum rate of change of ℓ with respect to $\hat{y}$ is 1 (since the gradient of ℓ with respect to $\hat{y}$ is either 0 or $-y$, where $y = 1$), Therefore, the change in loss due to a change in $\hat{y}$ can be bounded by the change in $\hat{y}$ itself as follows :

$$|\ell(\hat{y}_1, y) - \ell(\hat{y}_2, y)| \leq |\hat{y}_1 - \hat{y}_2|$$

- Since $\hat{y} = \langle \mathbf{x}, \mathbf{w} \rangle$, the difference in $\hat{y}$ is related to the difference in $\mathbf{x}$ by :

$$|\langle \mathbf{x}_1 - \mathbf{x}_2, \mathbf{w} \rangle|$$

- By the Cauchy-Schwarz inequality :

$$|\langle \mathbf{x}_1 - \mathbf{x}_2, \mathbf{w} \rangle| \leq \|\mathbf{x}_1 - \mathbf{x}_2\|_2 \|\mathbf{w}\|_2$$

- Assuming $\|\mathbf{w}\|_2 = 1$ for simplicity, or considering the scenario where we normalize $\mathbf{w}$ accordingly, we conclude that :

$$|\ell(\hat{y}, y) - \ell(\hat{y}', y)| \leq R\|\mathbf{x} - \mathbf{x}'\|_2$$

Therefore, the hinge loss function is R-Lipschitz over the set $\mathcal{X}$.

Optimization (12 points)

Question 2.

1. **(3 points)** Here, we'll try to prove convergence of gradient descent to a stationary point with a bounded error for an arbitrary function. Let $\mathbf{u}_1, \dots, \mathbf{u}_T$ be an sequence of gradients. Prove that if our weight vector is initialized as $\mathbf{w}^{(1)} = \mathbf{0}$, $\mathbf{w}^*$ is the global minima and we make updates of the form

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \cdot \mathbf{u}_t$$

then

$$\sum_{t=1}^{T} \langle \mathbf{w}^{(t)} - \mathbf{w}^*, \mathbf{u}_t \rangle \leq \frac{\|\mathbf{w}^*\|^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{u}_t\|^2$$

ANSWER

Given the weight update rule in gradient descent :

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \cdot \mathbf{u}_t$$

we aim to show :

$$\sum_{t=1}^{T} \langle \mathbf{w}^{(t)} - \mathbf{w}^*, \mathbf{u}_t \rangle \leq \frac{\|\mathbf{w}^*\|^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{u}_t\|^2$$

Starting with the squared norm difference between subsequent weights and the global minimum :

$$\|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2 = \|\mathbf{w}^{(t)} - \eta \cdot \mathbf{u}_t - \mathbf{w}^*\|^2$$

By expanding the right-hand side, we get :

$$\|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2 = \|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 - 2\eta \langle \mathbf{w}^{(t)} - \mathbf{w}^*, \mathbf{u}_t \rangle + \eta^2 \|\mathbf{u}_t\|^2$$

Rearrange the terms to focus on the inner product term :

$$2\eta \langle \mathbf{w}^{(t)} - \mathbf{w}^*, \mathbf{u}_t \rangle = \|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 - \|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2 + \eta^2 \|\mathbf{u}_t\|^2$$

Summing both sides over T iterations :

$$2\eta \sum_{t=1}^{T} \langle \mathbf{w}^{(t)} - \mathbf{w}^*, \mathbf{u}_t \rangle = \sum_{t=1}^{T} (\|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 - \|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2) + \eta^2 \sum_{t=1}^{T} \|\mathbf{u}_t\|^2$$

Notice that the sum on the right-hand side telescopes, simplifying to :

$$\|\mathbf{w}^{(1)} - \mathbf{w}^*\|^2 - \|\mathbf{w}^{(T+1)} - \mathbf{w}^*\|^2$$

Given $\mathbf{w}^{(1)} = \mathbf{0}$, we simplify further :

$$\|\mathbf{w}^*\|^2 - \|\mathbf{w}^{(T+1)} - \mathbf{w}^*\|^2$$

We then arrive to the **final inequality** by dividing both sides by 2η and rearranging as follows :

$$\sum_{t=1}^{T} \langle \mathbf{w}^{(t)} - \mathbf{w}^*, \mathbf{u}_t \rangle \leq \frac{\|\mathbf{w}^*\|^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{u}_t\|^2$$

2. The question deals with the Adam optimizer.

(a) **(2 points)** Adam works by using the moving average of the square of the gradients. It is given by the recursive formula :

$$s_t = \beta_2 s_{t-1} + (1-\beta_2) g_t^2$$

Express s_t as a function of $g_0, \ldots, g_t$

ANSWER

Given the recursive formula for the second moment in the Adam optimizer :

$$s_t = \beta_2 s_{t-1} + (1-\beta_2) g_t^2$$

we aim to express s_t as a function of all past squared gradients $g_0^2, g_1^2, \ldots, g_t^2$.

- Starting with the definition :

$$s_t = \beta_2 s_{t-1} + (1-\beta_2) g_t^2$$

- Expand s_{t-1} recursively :

$$s_t = \beta_2(\beta_2 s_{t-2} + (1-\beta_2) g_{t-1}^2) + (1-\beta_2) g_t^2$$

- Continue this process until s_0 we get :

$$s_t = \beta_2^t s_0 + (1-\beta_2) \left(\beta_2^{t-1} g_1^2 + \beta_2^{t-2} g_2^2 + \cdots + \beta_2^0 g_t^2 \right)$$

- Assuming s_0 is initialized to 0, we can simplify the formula to :

$$s_t = (1-\beta_2) \sum_{i=0}^{t} \beta_2^{t-i} g_i^2 \tag{1}$$

- The final expression for s_t is thus a weighted sum of all past squared gradients, with exponentially decreasing weights given by powers of β_2. This is reflecting an exponentially decreasing influence of earlier gradients.

(b) **(2 points)** Given the above, what is $\mathbb{E}[s_t]$ in terms of $\mathbb{E}[g_t^2]$ and β_2 ? You can assume that all g_i are independent and identically distributed.

ANSWER

- Starting with the expression we got to in **(1)** and taking the expectation, we get :

$$\mathbb{E}[s_t] = \mathbb{E}[(1-\beta_2)\sum_{i=0}^{t} \beta_2^{t-i} g_i^2]$$

- Then applying the linearity of expectation to the weighted sum of squared gradients, we get :

$$\mathbb{E}[s_t] = (1-\beta_2)\sum_{i=0}^{t} \beta_2^{t-i}\mathbb{E}[g_i^2]$$

- Since all g_i are i.i.d., their expected values are equal, therefore :

$$\mathbb{E}[s_t] = (1-\beta_2)\mathbb{E}[g_t^2]\sum_{i=0}^{t} \beta_2^{t-i}$$

- Recognizing the sum as a geometric series with $t+1$ terms and a common ratio β_2 we get :

$$\mathbb{E}[s_t] = (1-\beta_2)\mathbb{E}[g_t^2]\left(\frac{1-\beta_2^{t+1}}{1-\beta_2}\right)$$

- Finally, simplifying by canceling out $(1-\beta_2)$, we get :

$$\mathbb{E}[s_t] = \mathbb{E}[g_t^2](1-\beta_2^{t+1})$$

3. **(5 points)** Suppose you have linear regression data $\{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ where $x^{(i)} \in \mathbb{R}^d$. Your objective is to minimize

$$\mathcal{L} = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - {x^{(i)}}^T\theta)^2$$

If you add gaussian noise to augment the data, the objective then becomes

$$\mathcal{L}^* = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - {(x^{(i)}+\delta^{(i)})}^T\theta)^2$$

where $\delta^{(i)} \overset{\text{iid}}{\sim} \mathcal{N}(0,\sigma^2 I)$. Express the expectation of the modified objective $\mathcal{L}^*$ over the Gaussian noise, $\underset{\delta\sim\mathcal{N}}{\mathbb{E}}[\mathcal{L}^*]$, as a function of $\mathcal{L}$.

ANSWER

Given the modified objective with Gaussian noise added to the input data,

$$\mathcal{L}^* = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - (x^{(i)}+\delta^{(i)})^T\theta)^2$$

where $\delta^{(i)} \overset{\text{iid}}{\sim} \mathcal{N}(0,\sigma^2 I)$, our goal is to compute $\mathbb{E}[\mathcal{L}^*]$.

Expanding the square in the modified objective we get :

$$\mathcal{L}^* = \frac{1}{m}\sum_{i=1}^{m}\left((y^{(i)} - x^{(i)T}\theta)^2 - 2(y^{(i)} - x^{(i)T}\theta)\delta^{(i)T}\theta + (\delta^{(i)T}\theta)^2\right)$$

To find $\mathbb{E}[\mathcal{L}^*]$, we compute the expectation of each term separately :

- The first term $(y^{(i)} - x^{(i)T}\theta)^2$ is independent of δ, so its expectation is just itself.
- The second term involves $\delta^{(i)}$, but since the expectation of $\delta^{(i)}$ is 0 (being Gaussian noise with mean 0), the expectation of this term is 0.
- For the third term, we use the fact that $\delta^{(i)}$ is Gaussian noise with covariance $\sigma^2 I$, and so $\mathbb{E}[(\delta^{(i)T}\theta)^2] = \theta^T\mathbb{E}[\delta^{(i)}\delta^{(i)T}]\theta = \sigma^2\|\theta\|^2$, where $\|\theta\|^2$ is the squared norm of θ.

Combining these expectations, we get :

$$\mathbb{E}[\mathcal{L}^*] = \frac{1}{m}\sum_{i=1}^{m}\left((y^{(i)} - x^{(i)T}\theta)^2 + \sigma^2\|\theta\|^2\right)$$

The original objective $\mathcal{L}$ is :

$$\mathcal{L} = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - x^{(i)T}\theta)^2$$

So, adding the $\sigma^2\|\theta\|^2$ term to $\mathcal{L}$, we get :

$$\mathbb{E}[\mathcal{L}^*] = \mathcal{L} + \frac{\sigma^2}{m}\|\theta\|^2$$

Neural Networks (23 points)

Question 3.

1. **(9 points)** In class, you have discussed the Universal Approximation Theorem, i.e. no matter what your function $f(x)$ is, there is a network that can approximately approach the result and do the job. This result holds for any number of inputs and outputs. Let's try to show an example of this.

   Suppose you have a neural network with a single hidden layer where the corresponding activation function is the sign function
$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$
   Prove that for any $k \geq 1$, there is a way to set the number of units in the input, hidden and output layer such that any function $f : \{\pm 1\}^k \to \{\pm 1\}$ can be represented by the neural network.

   ANSWER

   Given a function $f : \{\pm 1\}^k \to \{\pm 1\}$, we can structure the neural network as follows :

   (a) The input layer has k units to represent each input variable in the k-dimensional input vector.
   (b) For the hidden layer to represent any $f : \{\pm 1\}^k \to \{\pm 1\}$, we can use a disjunctive normal form (DNF) representation or a conjunctive normal form (CNF) representation. Every boolean function can be represented in DNF, where the function is expressed as an OR of ANDs. For a given k, there are 2^k possible input combinations. Thus, in the worst case, we might need a separate unit in the hidden layer to represent each of these combinations. This implies that we can have up to 2^k **units**, each potentially representing a minterm of the DNF representation of f.
   (c) The output layer has 1 unit, representing the function's output (either 1 or -1).

   The activation function used is the sign function :
$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

   To represent a specific minterm in the hidden layer, we set the weights (w_i) and bias (b) such that :
$$\text{output of hidden unit} = \text{sign}\left(\sum_{i=1}^{k} w_i x_i + b\right)$$

   Where x_i is the input and w_i is set to a high positive value if the minterm requires $x_i = 1$ and a high negative value if $x_i = -1$, with the bias b adjusted accordingly.

Finally, the output unit aggregates the signals from the hidden layer, effectively performing an OR operation across all represented minterms, determining the +1 or -1 classification. Therefore, as outlined, a neural network with a single hidden layer using the sign activation function can represent any function $f : \{\pm 1\}^k \to \{\pm 1\}$ by modeling Boolean logic operations.

2. **(5 points)** Suppose that a neuron v with the above sign activation function has n incoming weights from neurons whose output values are all in $\{\pm 1\}$. Prove that if you add an additional weight to v, from a neuron with some constant value, there exists a way to set the incoming weights to v as well as the constant neuron value such that the output of v implements either the conjunction or disjunction of the inputs to v.

   The neuron v's output is determined by the weighted sum of its inputs, including the additional constant input neuron, passed through the sign function.

   To implement the **AND operation (conjunction)**, we want v to output 1 if and only if all n inputs are 1. We can set the weights as follows :

   - For each of the n inputs, set the weight to 1.
   - For the additional constant input, set the weight to $-(n - 0.5)$.This value is chosen so that the sum of the weights is less than 0 unless all inputs are 1. The -0.5 is there to ensure that when all inputs are 1, the total input to the sign function is 0.5, making the output 1.

   The weighted sum for the AND operation is therefore :

$$\text{output} = \text{sign}\left(\sum_{i=1}^{n} x_i - (n - 0.5)\right)$$

   Ensuring that the output is 1 if and only if all inputs are 1.

   To implement the **OR operation (Disjunction)**, the weights are adjusted slightly :

   - The weights for the n inputs remain 1.
   - The weight for the constant input is adjusted to -0.5.

   The weighted sum for the OR operation is :

$$\text{output} = \text{sign}\left(\sum_{i=1}^{n} x_i - 0.5\right)$$

   This setup ensures that the output is 1 if at least one input is 1.

3. **(9 points)** Suppose we have a function $f : [-1,1]^k \to [-1,1]$ that is ρ-Lipshitz and some error bound ϵ. Show how to build a neural network $N : [-1,1]^n \to [-1,1]$ such that for any input vector $\mathbf{x} \in [-1,1]^k$, $-\epsilon \leq f(\mathbf{x}) - N(\mathbf{x}) \leq \epsilon$. You need to justify how the neural network is constructed.

   Hint : Separate the input space so that you can use the Lipschitz property to bound the prediction error for any point.

Given a ρ-Lipschitz function $f : [-1,1]^k \to [-1,1]$ and an error bound ϵ, we build a neural network $N : [-1,1]^k \to [-1,1]$ as follows :

(a) To approximate f using a neural network while ensuring the error bound, we **discretize** the input space $[-1,1]^k$ into smaller regions where the change in f is controlled by the Lipschitz condition. The granularity of the discretization depends on Lipschitz constant ρ and error bound ϵ, which can be used to determine the maximum distance between any two points x and y in the input space to ensure that the difference in f does not exceed ϵ as follows :

$$\rho\|\mathbf{x} - \mathbf{y}\| \leq \epsilon$$

(b) We can construct the neural network as follows :

- The input layer consists of k neurons to accommodate the k-dimensional input x
- The network should have enough hidden layers and neurons to represent the partitioned input space. Specifically, we can use a structure that captures the discretized regions of the input space, with sufficient capacity to approximate f within each region to ensure that $-\epsilon \leq f(\mathbf{x}) - N(\mathbf{x}) \leq \epsilon$.
- The output layer consists of a single neuron, producing the final approximation $N(\mathbf{x})$ of $f(\mathbf{x})$

(c) We can then train the network on a representative set of input-output pairs $(x, f(\mathbf{x}))$, where x covers the discretized input space. The training process adjusts the weights and biases of the network to minimize the difference between $N(\mathbf{x})$ and $f(\mathbf{x})$, aiming to keep it within ϵ

Therefore, by leveraging the ρ-Lipschitz property, we ensure that the neural network N approximates f within the desired error bound ϵ across the input space $[-1,1]^k$.

Convolutional Neural Networks (16 points)

Question 4 (2-4-2-3-3)**.** Background

1. **(2 points)** Suppose I have a CNN with the following layers in exact order without any additional componenets between them :
   - A 3×3 convolutional layer with stride 3.
   - A 2×2 average pooling layer with stride 2.
   - A 2×2 convolutional layer with stride 2.

   Give the single convulutional layer that would produce the same output as this sequence of layers. Write out a formula that describes the weights for the layer. You can assume there no bias is present for each layer.

   ANSWER

   Let's first understand the effect of each layer :

   (a) **A** 3×3 **convolutional layer with stride** 3 ; Since the stride is equal to the kernel size, there's no overlap between the receptive fields of consecutive filter applications. This operation downsamples the input by a factor of 3 in both dimensions and applies a linear transformation defined by the filter weights, let's denote by W_1.

   (b) **A** 2×2 **average pooling layer with stride** 2 ; This layer averages the values of each 2×2 block of its input (resulting from the first layer) and downsamples by a factor of 2. This operation is linear and can be represented by a convolutional operation with equal weights $W_2 = \frac{1}{4}$ in each entry in a 2×2 kernel with stride 2.

   (c) 2×2 **convolutional layer with stride** 2 ; This final convolutional layer further applies a 2×2 filter to the output of the pooling layer with a stride of 2, effectively downsampling and transforming the input again by 2 in both dimensions and applies a linear transformation defined by the filter weights, let's denote by W_3.

   Combining these layers' effects into a single convolutional operation involves creating a new set of weights that encapsulates the sequential application of W_1, W_2 and W_3. Given an input vector x, the layer outout can be conceptually represented as :

   $$y = W_3 \cdot \text{Pool}(W_2 \cdot \text{Conv}(W_1 \cdot x))$$

   where *Conv*, *Pool*, and the dot products represent the convolutional and pooling operations in matrix form, and W_1, W_2, and W_3 represent the weights of the respective layers.

2. **(4 points)** Given the CNN defined by the layers in the below, fill in the shape of the output volume and the number of parameters at each layer in the following table (Table 1). Write the shapes in the format (H, W, C), where H, W, C are the height, width and channel dimensions. Unless specified, assume padding 1 and stride 1 where appropriate. Layers are expressed as follows :
   - CONV(H,C) : A convolution layer with filters of size $H \times W$ with C channels with appropriate weights and biases.
   - RELU : ReLU activation.
   - MAXPOOLING(N) : $N \times N$ max pooling with stride N.

- FC(D) : A fully connected layer with D outputs with appropriate weights and biases.
- BatchNorm : Batch normalization.
- Reshape : A flattening layer.

ANSWER

Let's create a generic formula for each type of layer to easily apply to the table :

(a) **Convolution Layer** (Conv(H,C)) : Applies filters of size $H \times W$ with C output channels. Assuming a padding of 1 is used to maintain spatial dimensions, the number of parameters is computed as $C \times (H \times W \times D + 1)$, where D is the depth of the input layer.

$$\therefore \text{Conv(3,8)} = (8 \times (3 \times 3 \times 3 + 1)) = 224$$

$$\therefore \text{Conv(3,16)} = (16 \times (3 \times 3 \times 8 + 1)) = 1168$$

(b) **Batch Normalization** (BatchNorm) : Normalizes the output of the previous layer. For each channel, it has 2 parameters (scale and shift), totaling $2 \times C$ parameters, where C is the number of channels.

(c) **ReLU Activation** (ReLU) : Applies the Rectified Linear Unit function element-wise. It does not introduce any additional parameters.

(d) **MaxPooling** (MaxPooling(N)) : Applies a $N \times N$ max pooling operation with stride N, reducing the spatial dimensions by a factor of N. This operation does not have parameters.

(e) **Fully Connected Layer** (FC(D)) : Connects all inputs to D outputs. The number of parameters is (input size) $\times D + D$ for weights and biases.

$$\therefore \text{FC(10)} = (1024 \times 10 + 10) = 10250$$

(f) **Reshape** : Changes the shape of the output without altering the data. No parameters are added in this operation.

Layer	Output Dimensions	Number of Parameters
Input	$32 \times 32 \times 3$	0
Conv(3,8)	$32 \times 32 \times 8$	224
BatchNorm	$32 \times 32 \times 8$	16
ReLU	$32 \times 32 \times 8$	0
MaxPooling(2)	$16 \times 16 \times 8$	0
Conv(3,16)	$16 \times 16 \times 16$	1168
BatchNorm	$16 \times 16 \times 16$	32
ReLU	$16 \times 16 \times 16$	0
MaxPooling(2)	$8 \times 8 \times 16$	0
Reshape	1024	0
FC(10)	10	10250

Table 1 – Table to fill out for part 2.

3. **(2 points)** In the above setup, there are specific parameters that can be removed without affecting modifying performance. What are they and why can they be removed ?

   ANSWER :

   (a) **Batch Normalization Parameters :** While batch normalization (BatchNorm) facilitates training and can lead to faster convergence by normalizing the input of each layer, its parameters (scale and shift for each feature channel) might not always be critical for the network's performance, especially in smaller or well-regularized networks. These parameters, although helpful for stabilizing and accelerating training, could be considered for removal if empirical evidence suggests minimal impact on the final performance.

   (b) **Fully Connected Layer Weights :** The fully connected layer (FC(10)) contributes to a significant portion of the network's parameters. Network pruning techniques here can identify and eliminate weights that contribute minimally to the output, thereby reducing the model size and computational demand without substantially impacting accuracy.

   (c) **Excess Convolutional Filters :** Lastly, convolutional layers with a large number of filters such as CONV(3,16) may lead to over-parameterization, where some filters do not significantly contribute to the model's ability to capture relevant features. Pruning such filters, especially since it's a deeper layer where feature abstraction is high, can reduce the computational burden and model complexity without detriment to performance.

4. **(3 points)** Fill out the following table (Table 2) by calculating the receptive field of a single pixel at every layer output.

   ANSWER

   The receptive field for any layer i in a CNN can be calculated based on the receptive field of the preceding layer (RF_{i-1}), the kernel size (k_i), and the cumulative product of strides from S_1 to S_{i-1} as follows :

$$RF_i = RF_{i-1} + (k_i - 1) \times \prod_{j=1}^{i-1} S_j$$

   To apply the formula at every layer output, we follow this algorithm
   (found at https ://www.baeldung.com/cs/cnn-receptive-field-size) :

   (a) Initialize the receptive field size r to 1 and the stride product S to 1.

   (b) For each layer l from the input layer to the output layer :
      - Update S by multiplying it with the stride s_{i-1} of the previous layer $i-1$.
      - Update r by adding to it $(k_i - 1) \times S$, where k_i is the kernel size of the current layer i.

   (c) The final value of r after processing all layers is the receptive field of the network.

   Going by each layer sequentially we have :

   - Conv(5,1) with stride 3 :
     — $k = 5$, $s = 3$
     — $r = 1 + (5-1) \times 1 = 5$ (S does not change in first layer)

- MaxPooling(2) with stride 2 :
  - $k = 2$, $s = 2$
  - $S = 1 \times 3 = 3$
  - $r = 5 + (2 - 1) \times 3 = 8$
- BatchNorm : This does not change the receptive field as it does not affect the spatial dimensions of the input (it's a channel-wise normalization), so it remains the same as the previous layer.
- Conv(4,1) with stride 2 :
  - $k = 4$, $s = 2$
  - $S = 3 \times 2 = 6$
  - $r = 8 + (4 - 1) \times 6 = 26$
- MaxPooling(2) with stride 1 :
  - $k = 2$, $s = 1$
  - $S = 6 \times 2 = 12$
  - $r = 26 + (2 - 1) \times 12 = 38$
- Conv(3,1) with stride 1 :
  - $k = 3$, $s = 1$
  - $S = 12 \times 1 = 12$
  - $r = 38 + (3 - 1) \times 12 = 62$

Consequently, here is the filled table :

Layer	Receptive Field
Conv(5,1) with stride 3	5
MaxPooling(2) with stride 2	8
BatchNorm	8
Conv(4,1) with stride 2	26
MaxPooling(2) with stride 1	38
Conv(3,1) with stride 1	62

Table 2 – Table to fill out for part 4.

5. **(4 points)** Suppose your input is a black and white image and consists only of pixels with values 0 or 1, where 1 is white and 0 is black (assume the image consists only of a single channel). Design two 3×3 convolution kernels, one that detects left-to-right dark-to-light vertical boundaries and another that detects left-to-right light-to-dark vertical boundaries. Your kernels should be as simple as possible (for simplicity, restrict values to be from -1 to 1), with output 1 for areas around the edges and 0 elsewhere. Explain what construct enables your kernels to detect these boundaries and not other boundaries (such as horizontal ones).

   We aim to design two 3×3 convolution kernels for detecting vertical boundaries in a binary image, where one kernel detects dark-to-light transitions from left to right and the other detects light-to-dark transitions.

Kernel for left-to-right Dark-to-Light Vertical Boundaries

For the dark-to-light transition, we want our kernel to have negative values on the left to respond to dark pixels and positive values on the right to respond to light pixels. The center column can be zeros to not influence the output. Here is the kernel :

$$K_{\text{dark-to-light}} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

When this kernel is convolved with an area of the image where a dark-to-light vertical boundary exists, the left side of the kernel (with negative values) will multiply with dark pixels (value 0), and the right side of the kernel (with positive values) will multiply with light pixels (value 1). This will result in a high positive response, indicating the presence of a dark-to-light edge.

Kernel for Light-to-Dark Vertical Boundaries

For the light-to-dark transition, we want our kernel to have positive values on the left and negative values on the right. Here is the kernel :

$$K_{\text{light-to-dark}} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

This kernel operates similarly to the first, but in reverse. The left side now responds to light pixels, and the right side to dark pixels. When convolved with a light-to-dark vertical boundary, it will output a high positive value.

Detection of vertical boundaries and not other boundaries discussion

These kernels are designed to detect vertical boundaries due to their horizontal arrangement of weights. The central column of zeros ensures that the response is maximal when there is a clear vertical boundary. Horizontal or diagonal edges will not produce as strong of a response because the sum of the products will not result in as large of a positive value. The simplicity of these kernels lies in their use of only three distinct values : -1, 0, and 1, which are the minimum necessary to detect the desired edges.