New York University Tandon School of Engineering
Computer Science and Engineering

CS-GY 6923: Final Exam.
Friday, December 19th, 2025, 2:00 - 3:30pm
??? Total Points

Directions

- Write you name at the top of each page.
- Show all of your work to receive full (and partial) credit.
- If more space is required, use extra sheets of paper, marked with your name and the problem number.

Important Note: This exam may be challenging. You do not need to get everything correct to earn a good grade. Partial credit will be awarded generously for clear reasoning and intermediate steps.

Please read each question carefully and manage your time according to the point values indicated. Do your best — clarity and justification matter more than perfect algebra.

Problem 1. Warm-up (14 points)

For each statement below, circle **True** if the statement is correct, **False** if the statement is incorrect.

1. In linear regression, removing features from the data makes the model more likely to overfit.

   True False

2. To validate a model, we first set aside a subset D_{val} of the training data as a validation set. We then train on all the data, including D_{val}. The performance on D_{val} is highly predictive of test performance.

   True False

3. ℓ_1 regularization can be used to perform automatic feature selection in logistic regression.

   True False

4. When optimizing a non-convex loss function, it is guaranteed that SGD will not converge to the global minimum.

   True False

5. A 5-class logistic regression model has the same number of parameters as a linear regression model with the same input features.

   True False

6. An MLP neural network is underfitting. To increase the model's capacity, we can make it wider, deeper, or both.

   True False

7. Consider an MLP where we remove all activation functions, leaving only fully-connected (`torch.nn.Linear`) layers. This network can still perform non-linear operations as long as it has multiple layers.

   True False

8. Skip (residual) connections are only useful for convolutional neural networks, where they enable training of very deep architectures such as ResNets.

   True False

9. Convolutions are a special case of fully-connected (`torch.nn.Linear`) layers. Any convolutional neural network can be converted to a functionally identical fully-connected network (ignore pooling, layernorm, etc).

   True False

10. Vanilla recurrent neural networks must compress all information about past context into a fixed-size hidden state.

True False

11. Transformers can process all tokens in a sequence in parallel, whereas RNNs require sequential processing.

True False

12. Ensembling is a good way to reduce bias in weak models. For example, ensembling is often used for linear regression models where it introduces additional capacity and leads to higher prediction.

True False

13. One of the main advantages of decision trees is that they typically perform well on high-dimensional inputs.

True False

14. The quality of a classifier's uncertainty estimates can be evaluated by measuring accuracy on held-out data.

True False

Problem 2. Rotation angle prediction (12 points)

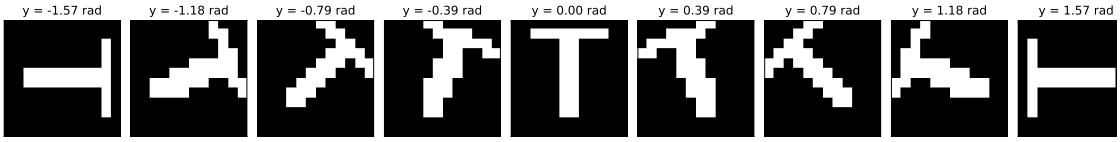


Figure 1: Example datapoints.

We are given a dataset of 12×12 binary (black and white) images, each depicting the letter "T" rotated by some angle θ. All images are generated by rotating the same base letter and rasterizing to a 12×12 grid. The labels y are the true rotation angles used to generate each image. Figure 1 shows representative examples.

Due to the finite resolution of the rasterization process, a range of continuous angles (width ≈ 0.1 rad) maps to the exact same binary image. In other words, rotation angles differing by less than approximately 0.1 radians typically produce visually indistinguishable images. All rotation angles in the dataset lie within the interval $[-\pi/2, \pi/2]$.

(a) [4 points] We wish to design a model that predicts rotation angles strictly within $(-\pi/2, \pi/2)$. Propose a suitable model architecture (linear or neural network). Write down an appropriate loss function and describe a method for training the model.

(b) [4 points] Suppose we train this model on our dataset. On the first 5 training examples, the model predicts:

$$\hat{y} = (-0.4771,\ -1.2654,\ 1.1327,\ 1.4298,\ -1.5224)$$

while the true rotation angles are:

$$y = (-0.4771,\ -1.2654,\ 1.1327,\ 1.4298,\ -1.5224)$$

(values agree to at least 4 decimal places). What can we conclude about this model? Based on this information alone, what can we conclude about the model's performance on a held-out test set?

(c) ~~[3 points] Now suppose we generate a new training set and retrain the model. Our new training set consists of images with rotation angles sampled on a uniform grid over $[-\pi/2, \pi/2]$ with step size 0.01 radians. The test set consists of images with rotation angles sampled uniformly at random from the same interval. If our model achieves very low loss on the training set, should we be concerned about overfitting? Justify your answer.~~

(d) [4 points] Instead of treating this as a regression problem, we now wish to frame it as a classification problem using exactly 10 classes. Propose a model architecture and loss function suitable for this formulation (be explicit, add detail). Explain how you would define the classes.

Solution to Problem 2

(a) Model Architecture: Since the relationship between the pixel values of the rasterized image and the rotation angle is highly non-linear (pixels toggle on and off abruptly as the shape rotates), a linear regression model would likely perform poorly. A suitable architecture is a **Convolutional Neural Network (CNN)** or a multi-layer Perceptron (MLP). Given the small image size (12×12), a simple CNN with 2 convolutional layers followed by fully connected layers, or an MLP with 1-2 hidden layers and ReLU activations, would be appropriate.

Loss Function: Since this is a regression problem where we want to predict a continuous value y, the standard loss function is the **Mean Squared Error (MSE)**:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

where y_i is the true angle and $\hat{y}_i$ is the predicted angle.

Training Method: The model can be trained using **Stochastic Gradient Descent (SGD)** or an adaptive variant like **Adam**. We would perform backpropagation to minimize the MSE loss over minibatches of the training data.

(b) We can conclude that the model is likely **overfitting** or memorizing the training data.

Reasoning: The problem states that the mapping from angle to image is a step function with a width of approximately 0.1 radians. This implies that for any given image x, the true label y could be any value within a range $[y_{min}, y_{max}]$ where $y_{max} - y_{min} \approx 0.1$. Mathematically, the inverse mapping $x \to y$ is not a function; there is irreducible uncertainty (aleatoric uncertainty) in y given x.

The best a model can ideally do (to minimize expected squared error) is to predict the *conditional expectation* $E[y|x]$, which would be the center of that 0.1 radian interval. The fact that the model predicts the exact label to 4 decimal places (e.g., matching the specific random seed used to generate that training example) suggests it is not learning the general visual features, but rather memorizing the specific label associated with that specific training image instance.

Performance on Test Set: The performance on the test set will likely be worse than on the training set (non-zero error). We cannot conclude that the test performance will be very poor, but predictions for some images will be suboptimal.

Reasoning: While the training error is near zero, the test error will be non-zero. The model predicts a specific scalar y_{train} for a given image x. However, a test example with the same image x will have a label y_{test} drawn essentially at random from the bucket of width 0.1.

Mathematically, the overfitted model's Mean Squared Error (MSE) will be approximately twice the theoretical minimum MSE (the irreducible error). While the model might still be useful (high R^2), it has failed to learn the conditional mean $E[y|x]$ and has instead memorized noise.

(c) ~~**No, we should not be concerned about overfitting.** In fact, low loss on this training set implies the model has learned the optimal solution.~~

~~**Justification:** In this scenario, the training data density (step size 0.01 rad) is much finer than the ambiguity resolution (0.1 rad). This means for any specific unique image x (which corresponds to a bucket of width 0.1), the training set contains approximately $\frac{0.1}{0.01} = 10$ different examples with the same input x but different labels $y_1, y_2, \ldots, y_{10}$.~~

~~If the model achieves very low MSE loss on the training set, it must be predicting a value $\hat{y}$ that minimizes $\sum(y_i - \hat{y})^2$ for these conflicting labels. The minimizer is the **mean** of the training labels for that image. Since the training labels are a uniform grid, their mean is the center of the bucket.~~

~~The test set is drawn from a uniform random distribution. For a given image x, the true label is uniformly distributed across the bucket. The prediction that minimizes the expected test error for a uniform distribution is also the **mean** (center of the bucket). Therefore, by minimizing loss on the "dense" training set, the model is forced to learn the conditional expectation $E[y|x]$, which is the Bayes optimal predictor for the test set.~~

(d) Class Definition: We can divide the total range $[-\pi/2, \pi/2]$ (total width π) into 10 disjoint intervals (bins) of equal width $\frac{\pi}{10}$. Class $k \in \{0, \ldots, 9\}$ is the true class if the angle y falls into the k-th bin.

Model Architecture: The architecture would be similar to part (a) (CNN or MLP), but the final output layer should be a **Softmax** layer with **10 output neurons**, representing the probability distribution over the 10 classes.

Loss Function: The appropriate loss function for multi-class classification is the **Cross-Entropy Loss**:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=0}^{9}\mathbb{I}(y_i \in \text{class}_c)\log(p_{i,c})$$

where $p_{i,c}$ is the predicted probability that the i-th example belongs to class c.

Grading Rubric: Problem 2 (15 Points Total)

General Note to Graders: This rubric is designed to be forgiving. We are looking for intuition and understanding of core concepts (overfitting, loss functions, regression vs. classification) rather than mathematical rigor or specific framework syntax.

(a) Model Architecture & Setup (4 Points)

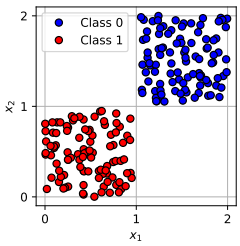
- **[1 pt] Loss Function and architecture:** Proposes a linear model, Neural Network, MLP, CNN. Specifies Mean Squared Error (MSE), L2 Loss, or Sum of Squared Errors.
- **[1 pt] Training Method:** Mentions Gradient Descent, Stochastic Gradient Descent (SGD), Adam, or generally "Backpropagation."
- **[2 pt] Output Validity:** The students explicitly constrain the output range to $(-\pi/2, \pi/2)$ (e.g. via Tanh).

(b) Overfitting Analysis (4 Points + up to 2 Bonus)

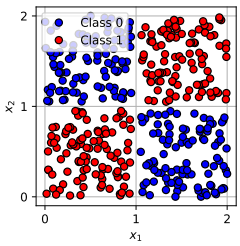
- **[2 pts] Conclusion - Overfitting:** Identifies that the model is overfitting, memorizing the training data, or "cheating."
  - *Key Indicator:* Student notes that predicting to 4 decimal places is suspicious given the 0.1 radian resolution limit.
- **[2 pts] Test Performance:** States that performance on the test set will be worse than the training set (generalization gap) or potentially poor.
  - *Forgiving:* Accept answers saying "Performance will be okay but not perfect" or "Worse than training."
  - *Reject:* Answers claiming the test performance will be "perfect" or "zero error."
- **[+1 to +2 pts BONUS] Nuanced Insight:**
  - Award bonus points if the student explains *why* the prediction is suboptimal: The model is predicting a specific random scalar from the bucket rather than the **conditional mean** (center of the bucket). The model has fit the noise rather than the signal.

(d) Classification Formulation (4 Points)

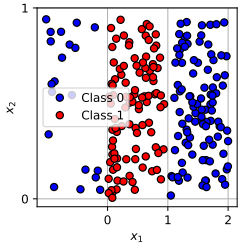
- **[1 pt] Class Definition:** Proposes splitting the continuous range into distinct bins/intervals (e.g., 10 bins of width $\approx \pi/10$).
- **[1 pt] Architecture Change:** Notes that the output layer must change to size 10 (one neuron per class).
- **[1 pt] Activation:** Specifies using **Softmax** on the output.
- **[1 pt] Loss Function:** Specifies **Cross-Entropy Loss** (or Log Loss / Negative Log Likelihood).



Dataset 1 for parts (a, b)



Dataset 2 for part (c)



Dataset 3 for part (d)

Figure 2: Two-class classification dataset. Blue circles represent Class 0; red circles represent Class 1.

Problem 3. Tress on Chessboards (16 Points Total)

Consider the dataset shown in Figure 2, consisting of two classes in $\mathbb{R}^2$. Throughout this problem, assume that when constructing decision trees:

- We use axis-aligned splits of the form $x_i \leq t$ for some feature $i \in \{1, 2\}$ and threshold $t \in \mathbb{R}$.
- When multiple (feature, threshold) pairs yield the same optimal value of the splitting criterion, we select uniformly at random among them.
- The depth of a tree is defined as the maximum number of splits along any path from the root to a leaf (so a tree with just a root node has depth 0).

(a) [4 points] What is the minimum depth of a decision tree that achieves perfect (100%) training accuracy on this dataset? For a tree of this minimum depth, sketch the decision regions on the x_1-x_2 plane, labeling each region with the predicted probability $P(\text{Class} = 1 \mid x)$. If there are multiple trees that achieve perfect accuracy, describe all of them.

(b) [4 points] Now consider a random forest consisting of a large number of trees (e.g. 10^3), trained on the same dataset with the following specifications:

- No bootstrap sampling is performed; each tree is trained on the full dataset.
- Each tree is grown to the minimum depth needed for perfect training accuracy.
- The forest outputs the average predicted probability across all trees.

Sketch the predicted probability $P(\text{Class} = 1 \mid x)$ as a function of position in the x_1-x_2 plane. Clearly indicate the probability in each distinct region.

In what region(s) does the random forest exhibit high uncertainty (i.e., $P(\text{Class} = 1 \mid x) \approx 0.5$)? Explain why this occurs and contrast this behavior with that of a single decision tree.

(c) [4 points] What is the minimum depth of a decision tree that achieves perfect training accuracy on Dataset 2 in Figure 2?

Consider the standard greedy algorithm for constructing decision trees discussed in class, which selects the split at each node that maximizes the reduction in impurity (e.g., Gini impurity). If we allow this algorithm to grow a tree up to the depth you identified above, will it achieve perfect training accuracy? Why or why not?

(d) [4 points] Suppose we want to train an ensemble of depth-1 decision trees (i.e., decision stumps) on Dataset 3 in Figure 2. Which ensemble method is more likely to achieve good accuracy on this dataset: *bagging* or *boosting*? Explain your reasoning.

Solution to Problem 3

(a) **Minimum Depth and Decision Regions**

The minimum depth required to achieve perfect training accuracy is **1**.

Looking at Dataset 1, the two classes are perfectly separable by a single axis-aligned split. There are two optimal splits that separate the clusters completely:

- Split A: $x_1 \leq 1$. (Left region is Class 1, Right region is Class 0).
- Split B: $x_2 \leq 1$. (Bottom region is Class 1, Top region is Class 0).

Since the problem states we select uniformly at random among optimal splits, the minimum depth is 1.

Sketch of Decision Regions: Since there are multiple trees achieving this, we describe the two possibilities:

(a) **Tree 1 (Splitting on x_1):**
- Region $x_1 \leq 1$: $P(\text{Class} = 1|x) = 1$
- Region $x_1 > 1$: $P(\text{Class} = 1|x) = 0$

This creates a vertical decision boundary at $x_1 = 1$.

(b) **Tree 2 (Splitting on x_2):**
- Region $x_2 \leq 1$: $P(\text{Class} = 1|x) = 1$
- Region $x_2 > 1$: $P(\text{Class} = 1|x) = 0$

This creates a horizontal decision boundary at $x_2 = 1$.

(b) **Random Forest Prediction and Uncertainty**

Given the specifications, the forest consists of trees that are depth 1. Because the splitting criterion (e.g., information gain or Gini impurity reduction) is identical for the split at $x_1 = 1$ and the split at $x_2 = 1$, and the algorithm breaks ties uniformly at random, approximately 50% of the trees in the forest will split on x_1 (Tree Type 1) and 50% will split on x_2 (Tree Type 2).

Let us denote the forest prediction as $P_{RF}(C = 1|x)$.

Analysis by Region: The input space $[0, 2] \times [0, 2]$ is divided into four quadrants:

1. **Bottom-Left ($x_1 \leq 1, x_2 \leq 1$):** Contains Class 1 training data.
   - Tree Type 1 predicts 1.
   - Tree Type 2 predicts 1.
   - $P_{RF} \approx 1$.
2. **Top-Right ($x_1 > 1, x_2 > 1$):** Contains Class 0 training data.
   - Tree Type 1 predicts 0.
   - Tree Type 2 predicts 0.
   - $P_{RF} \approx 0$.
3. **Top-Left ($x_1 \leq 1, x_2 > 1$):** Empty region (no training data).
   - Tree Type 1 checks $x_1 \leq 1$, goes Left $\rightarrow$ predicts 1.
   - Tree Type 2 checks $x_2 > 1$, goes Top $\rightarrow$ predicts 0.
   - Average: $\frac{1+0}{2} = 0.5$.
   - $P_{RF} \approx 0.5$.
4. **Bottom-Right ($x_1 > 1, x_2 \leq 1$):** Empty region (no training data).
   - Tree Type 1 checks $x_1 > 1$, goes Right $\rightarrow$ predicts 0.

- Tree Type 2 checks $x_2 \leq 1$, goes Bottom $\rightarrow$ predicts 1.
- Average: $\frac{0+1}{2} = 0.5$.
- $P_{RF} \approx 0.5$.

Regions of High Uncertainty: The forest exhibits high uncertainty ($P \approx 0.5$) in the **Top-Left** and **Bottom-Right** quadrants.

Explanation: This occurs because these regions contain no training data (they are out-of-distribution relative to the clusters). The individual decision trees generalize differently in these empty regions based on their selected split direction (vertical vs. horizontal). A single decision tree would confidently (and arbitrarily) predict probability 1 or 0 in these regions depending entirely on which variable it split on. The Random Forest averages these conflicting confident predictions, correctly revealing the epistemic uncertainty inherent in regions where no data was observed.

(c) **Dataset 2 (XOR) and Greedy Algorithms**

Minimum Depth: The minimum depth required is **2**. A depth-1 tree can only draw one line (vertical or horizontal), which cannot separate the "checkerboard" pattern of the XOR problem. A depth-2 tree can isolate the four quadrants (e.g., Root splits x_1 at 1; Left Child splits x_2 at 1; Right Child splits x_2 at 1).

Greedy Algorithm Performance: The standard greedy algorithm **will not** achieve perfect training accuracy (it will likely fail to grow a useful tree at all).

Reasoning: Standard greedy algorithms evaluate splits based on the immediate reduction in impurity (marginal gain). In the XOR dataset (Dataset 2), the class distribution is balanced (50% Class 0, 50% Class 1).

- If we check a split on x_1 at any threshold, the resulting left and right subsets will still contain approximately 50% Class 0 and 50% Class 1. Thus, the Information Gain (or Gini reduction) is effectively zero.
- The same applies to splits on x_2.

Because the algorithm looks only one step ahead (myopic), it sees no benefit in splitting. Depending on the implementation, it will either stop growing immediately (resulting in a depth-0 tree predicting the majority class) or pick a random ineffective split, failing to capture the interaction between x_1 and x_2 required to solve the XOR problem.

(d) **Bagging vs. Boosting on Dataset 3**

Boosting is significantly more likely to achieve good accuracy on Dataset 3.

Reasoning based on Dataset Asymmetry: Dataset 3 consists of vertical stripes where the classes alternate along the x_1 axis. Crucially, the stripes are **asymmetric**: the rightmost cluster of Class 0 (blue) is much wider and contains more points than the leftmost cluster.

- **Bagging (High Bias):** A decision stump (depth-1 tree) allows for only one axis-aligned split. The greedy splitting criterion (e.g., Information Gain) will favor the split that yields the largest immediate reduction in impurity. Because the rightmost blue stripe is larger, isolating it reduces entropy more than isolating the smaller leftmost stripe.

  Consequently, almost every tree in the bagged ensemble will independently choose the exact same split (approx $x_1 \approx 1.2$). Since Bagging averages these independent models, the ensemble will behave exactly like a single decision stump. It will correctly classify the right side but systematically misclassify the left side, failing to capture the alternating structure.
- **Boosting (Bias Reduction):** Boosting trains trees sequentially to correct previous errors.
  - The first stump will likely split at $x_1 \approx 1.2$ (the dominant split), misclassifying the smaller left stripe.
  - Boosting will then increase the weights of these misclassified points.
  - The second stump, training on the weighted data, will be forced to prioritize the left side, placing a split at $x_1 \approx 0.4$.

By summing these stumps, Boosting builds a complex decision boundary (an additive model) capable of capturing multiple stripes, effectively converting a high-bias weak learner into a low-bias strong learner.

Problem 3 Grading Rubric (Total: 16 Points)

(a) Minimum Depth & Decision Regions (4 Points)

- **1 Point:** Correctly identifies **Minimum Depth = 1**.
- **2 Points:** Sketches (or describes) a valid decision boundary.
  - *Full Credit:* Draws a vertical line at $x_1 \approx 1$ **OR** a horizontal line at $x_2 \approx 1$.
  - *Full Credit:* Correctly labels the regions (e.g., Left=Class 1, Right=Class 0).
  - *Incorrect (0 pt):* Draws a diagonal line or non-axis aligned split (indicates conceptual error regarding tree constraints).
- **1 Point:** Acknowledges the existence of **multiple solutions**.
  - *Full Credit:* Mentions that the tree could split on either x_1 or x_2 (due to random tie-breaking).
  - *Forgiving Note:* If the student only sketches one tree but their sketch implies a specific choice among options (or they describe the tie-breaking logic), award the point.

(b) Random Forest & Uncertainty (4 Points)

- **1 Point:** Correct predictions in **data-dense regions**.
  - Bottom-Left ≈ 1; Top-Right ≈ 0.
- **2 Point:** Correct predictions in **empty regions**. Correctly identifies the regions of **High Uncertainty**.
  - Top-Left ≈ 0.5; Bottom-Right ≈ 0.5.
  - *Forgiving Note:* Accept values like "approx 0.5", "uncertain", or "between 0 and 1".
- **1 Point:** Explanation of uncertainty source.
  - *Key Concept:* The forest averages conflicting predictions from different trees (some split horizontally, some vertically).
  - *Forgiving Note:* Award credit if they explain that "the model hasn't seen data there" or "it's out of distribution," even if they don't explicitly describe the averaging mechanism.

(c) XOR Dataset & Greedy Failure (4 Points)

- **1 Point:** Correctly identifies **Minimum Depth = 2**.
  - Note: Depth 1 cannot separate the XOR/checkerboard pattern.
- **1 Point:** Correct Answer: **"No"** (The greedy algorithm will NOT achieve perfect accuracy).
- **2 Points:** Explanation of **Greedy Failure**.
  - *Full Credit:* Explains that the **Information Gain (or impurity reduction) is zero** for the first split because the classes are balanced along both axes (50/50 split regardless of threshold).
  - *Partial Credit (1 pt):* Mentions that greedy algorithms are "myopic" or "don't look ahead" but fails to mention the specific zero-gain/balanced class issue.

(d) [Bonus] Bagging vs. Boosting (4 Points)

- **1 Point:** Selects **Boosting**.

- **1 Point:** Identifies **High Bias** / Underfitting of decision stumps.
  - Acknowledges that a single stump (depth-1) cannot capture the multiple stripes of Dataset 3.
- **1 Point:** Explanation of **Why Bagging Fails**.
  - *Full Credit:* Explains that averaging many high-bias models results in a high-bias model.
  - *Strong Credit:* Specifically notes that due to dataset asymmetry (right stripe > left stripe), all bagged trees will likely pick the exact same split, resulting in no improvement over a single tree.
- **1 Point:** Explanation of **Why Boosting Works**.
  - *Full Credit:* Explains the **sequential/additive** nature: later trees fix the errors of earlier trees (by re-weighting the misclassified smaller stripe).

Problem 4. The Mixer (20 points)

Consider a single-layer transformer for language modeling, consisting of a self-attention block followed by a feed-forward (MLP) block. Let the input be a matrix $X \in \mathbb{R}^{T \times D}$, where T is the sequence length and D is the model dimension (number of channels).

(a) [4 points] A transformer can be viewed as operating along two axes:

- The *channel axis* (dimension D): each position has a D-dimensional representation.
- The *sequence axis* (dimension T): the sequence consists of T token positions.

For each of self-attention and the MLP block, identify which axis it *mixes* information along. That is, which operation allows different token positions to communicate with each other, and which operation allows different channels within a single position to interact?

(b) [16 points] Self-attention can be complex to analyze. Suppose we want to replace it with a simpler MLP-based operation while still allowing both tokens and channels to communicate.

Propose an architecture that uses MLPs to mix information along *both* axes. Your design should be *symmetric*: the operations on the channel axis and the sequence axis should both be MLPs, differing only in which dimension they operate across.

(i) [6 points] Write pseudocode for a single layer of your architecture. Clearly specify the input and output shapes of each operation. Assume the input is $X \in \mathbb{R}^{T \times D}$. Ignore the layernorms, but include skip connections.

Constraints & Assumptions:

- For the MLP mixing along the sequence axis, assume the hidden layer size is also T (i.e., it maps $\mathbb{R}^T \to \mathbb{R}^T \to \mathbb{R}^T$).
- Assume **weight sharing**: the MLP mixing along the sequence axis uses the *same* weights for every channel, and the MLP mixing along the channel axis uses the *same* weights for every token position.

(ii) [6 points] In language modeling, we require *causal masking*: when predicting the token at position t, the model may only use information from positions $1, \ldots, t$. Given the constraints in (i), what condition must you impose on the weight matrices of the sequence-mixing MLP to enforce causal masking?

(iii) [4 points] Discuss at least one other limitation of your proposed architecture compared to standard self-attention.

Solution to Problem 4.

(a) Mixing Axes identification:

- **The MLP block mixes information along the channel axis.** The MLP is applied position-wise (independently to each token). It maps a vector $x_t \in \mathbb{R}^D$ to a new representation, allowing interactions between the different features (channels) of a single token, but it does not move information between different time steps t.
- **Self-attention mixes information along the sequence axis.** The attention mechanism computes weighted sums of value vectors from different time steps based on the similarity between queries and keys. This allows the representation at position t to incorporate information from positions t' (where $t' \neq t$), facilitating communication across the sequence.

(b) MLP-Mixer Architecture:

(i) **Pseudocode:**

We define two types of MLP blocks: a `TokenMixingMLP` which acts on the columns of the input, and a `ChannelMixingMLP` which acts on the rows.

Let the input be $X \in \mathbb{R}^{T \times D}$.

Single Layer of Symmetric MLP Architecture

Input: $X \in \mathbb{R}^{T \times D}$
// 1. Mix along Sequence Axis (Token Mixing)
// Transpose to apply MLP across the T dimension
$X_{transposed} \leftarrow X^\top$ {Shape: $D \times T$}
$Z \leftarrow \text{MLP}_{\text{token}}(X_{transposed})$ {Applies MLP to each of the D rows independently. Input/Output dim: T}
$Z \leftarrow Z^\top$ {Shape: $T \times D$}
$U \leftarrow X + Z$ {Skip Connection}
// 2. Mix along Channel Axis (Channel Mixing)
$Y \leftarrow \text{MLP}_{\text{channel}}(U)$ {Applies MLP to each of the T rows independently. Input/Output dim: D}
$Output \leftarrow U + Y$ {Skip Connection}
Return: *Output*

Here, both $\text{MLP}_{\text{token}}$ and $\text{MLP}_{\text{channel}}$ typically consist of two linear layers separated by a non-linearity (e.g., ReLU or GeLU).

(ii) **Causal Masking:**

In the `TokenMixingMLP`, the operations are linear transformations over the sequence dimension T. If the MLP consists of linear layers W_1 and W_2 (e.g., $f(x) = W_2\sigma(W_1 x)$), these matrices $W \in \mathbb{R}^{T \times T}$ multiply the sequence vector.

To enforce causal masking, we must ensure that the output at position t depends only on inputs at positions $1, \ldots, t$. This is achieved by constraining the weight matrices W in the `TokenMixingMLP` to be **lower triangular**. Since the product of lower triangular matrices (and element-wise non-linearities) preserves the lower triangular property, this ensures no information flows from future tokens $t + k$ to the current token t.

(iii) **Limitations:**

One major limitation is that **the architecture requires a fixed input sequence length** T. Standard Multi-Layer Perceptrons have fixed input dimensions determined by their weight matrices. While Self-Attention can process sequences of varying lengths (producing attention maps of dynamic size $T \times T$), an MLP mixing across the sequence axis relies on a fixed weight matrix $W \in \mathbb{R}^{T \times T}$. To handle different lengths, one would need padding, truncation, or separate weights for every possible length.

*(Alternative limitation: The mixing weights are **static** and content-independent. Unlike attention, where the interaction strength between tokens i and j depends on their values (via dot products), the MLP mixer learns a fixed weight W_{ij} connecting position j to i regardless of the token content.)*

Rubric for Problem 4.

Total: 20 Points

(a) Axis Identification (4 points)

- **2 points:** Correctly identifies that the **MLP block** mixes along the **Channel axis** (interactions within a single token).
- **2 points:** Correctly identifies that **Self-Attention** mixes along the **Sequence axis** (interactions between different tokens).
- *Partial Credit:* 0 points if swapped. Deduct 1 point if the explanation is confusing but the final answer seems correct.

(b)(i) Architecture Pseudocode (6 points)

- **2 point:** Explicitly shows transposing (or equivalent indexing) to apply an MLP across the sequence dimension (T). It is not require that they use a transposition, but the MLP needs to be applied along the sequence dimension.
- **1 point:** Shows a second MLP applied to the channel dimension (D).
- **2 point:** Correct input/output shapes are maintained (starts with $T \times D$, ends with $T \times D$).
- **1 point:** Includes residual (skip) connections for both blocks.
- *Note:* Do not penalize for syntax errors or missing non-linearities/LayerNorms.
- *Forgiving:* If they describe the operations clearly in text instead of code, give full credit.

(b)(ii) Causal Masking (6 points)

- **4 points:** States that the weight matrices $W \in \mathbb{R}^{T \times T}$ for the sequence-mixing MLP must be **Lower Triangular** (or Upper Triangular, depending on how they defined their multiplication, provided they justify it prevents future-to-past flow).
- *Partial Credit (2 points):* Mentions "masking" the weights or setting specific weights to zero, but fails to identify the triangular structure explicitly.

(b)(iii) Limitations (4 points)

- **4 points:** Identifies a valid limitation. Common correct answers:
  - **Fixed Sequence Length:** The model cannot handle sequences longer than T because the weight matrix size is fixed.
  - **Static Weights:** Interactions are data-independent (unlike Attention, where weights depend on the query/key dot product).
  - **Parameter Count:** If they argue it requires too many parameters (scaling with T^2), this is valid.
- *Partial Credit (2 points):* Vague limitations like "it is slower" or "it is less expressive" without justifying why. Any reasonable thoughts and ideas that don't lead to a proper limitation.