

18.S997 PSet 1

Due **Feb 24, 2025 11:59 PM**

This PSet will give you an opportunity to reinforce your understanding of neural networks in the context of biological data, specifically gene expression analysis using the [GTEx](#) dataset.

Instructions

Please submit your responses, including text, image, and code responses, to Gradescope. Take care that each question is matched with the correct pages, and that text responses are not cut off. For questions that request images, it is fine to insert them into the colab or to include them as separate pages in your submission. Instead of a PDF, it is also fine to submit screenshots as images, which Gradescope will accept.

As before, you can use [this notebook](#) to run nbconvert for your completed .ipynb file. You can find the additional files for this PSet in [this folder](#).

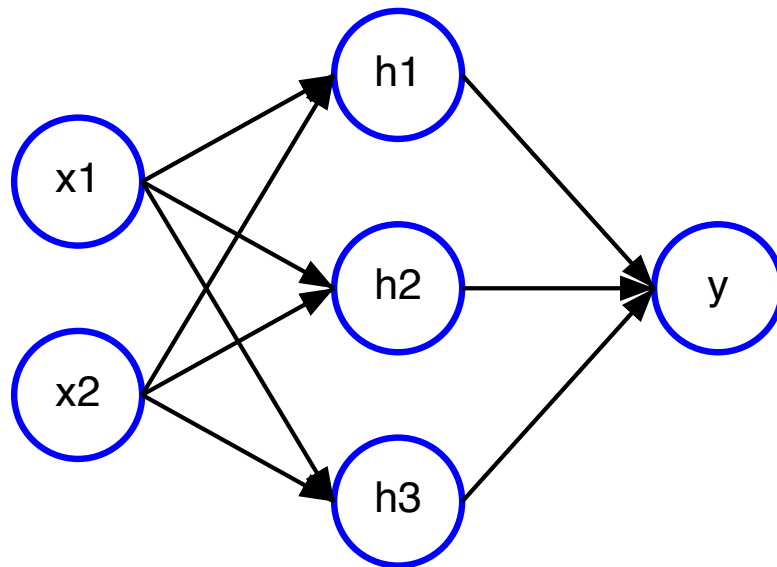
We encourage you to be mindful of compute resources when completing this assignment. In particular, we recommend leaving the notebook disconnected from a runtime until you go to run your code in Questions 4 and 5. There is no need to waste hours/credits while you are working on the written portion.

✓ Written Questions

✓ Question 1: Shallow Networks

Consider the network shown. Assume a ReLU activation function in the hidden layer and that the network has biases. (Questions follow image.)

> network image

[Show code](#)

Question 1a Calculate the total number of parameters in the model represented by the diagram.

There are $6+3+4 = 13$ parameters (including the biases, which are not shown)

Question 1b Assume the input to the network are binary numbers. Provide a set of weights that implement each of the following logic operations on the two inputs: AND, OR, XOR.

If we write out the parameters of the model, we get:

$$y = \text{relu} \left(\begin{bmatrix} x_1 & x_2 \end{bmatrix} \cdot \begin{bmatrix} h_1^1 & h_2^1 & h_3^1 \\ h_2^1 & h_2^2 & h_2^3 \end{bmatrix} + \begin{bmatrix} b_1^1 & b_1^2 & b_1^3 \end{bmatrix} \right) \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} + \begin{bmatrix} b_2 \end{bmatrix}$$

AND:

$$y = \text{relu} \left(\begin{bmatrix} x_1 & x_2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} -1 & 0 & 0 \end{bmatrix} \right) \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix}$$

OR:

$$y = \text{relu} \left(\begin{bmatrix} x_1 & x_2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 0 \end{bmatrix} \right) \cdot \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix}$$

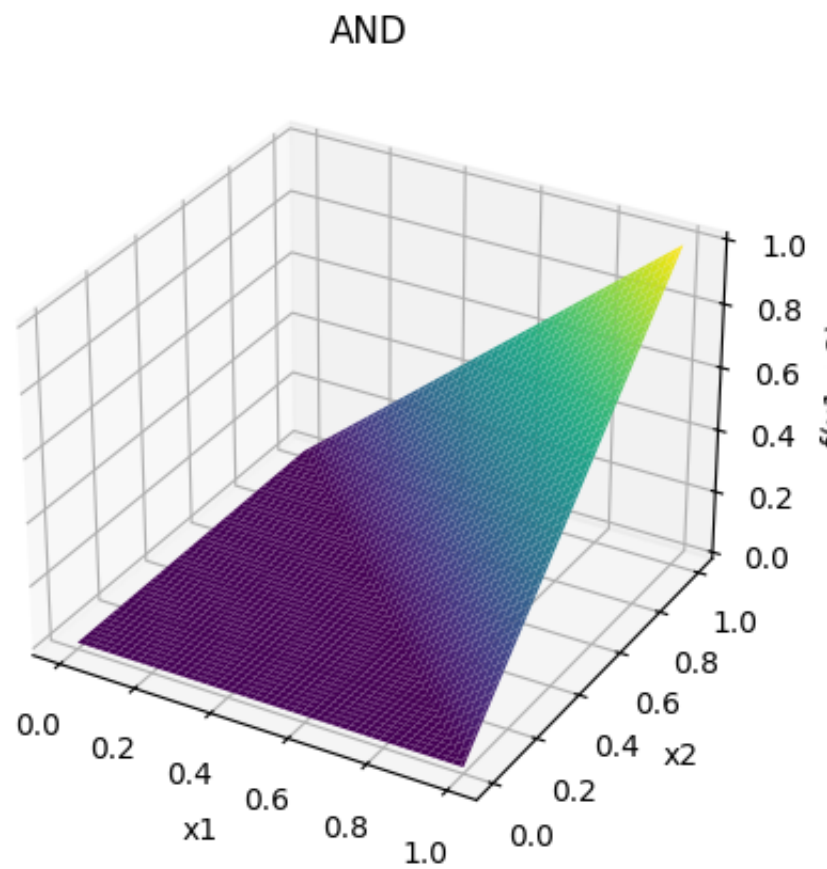
XOR:

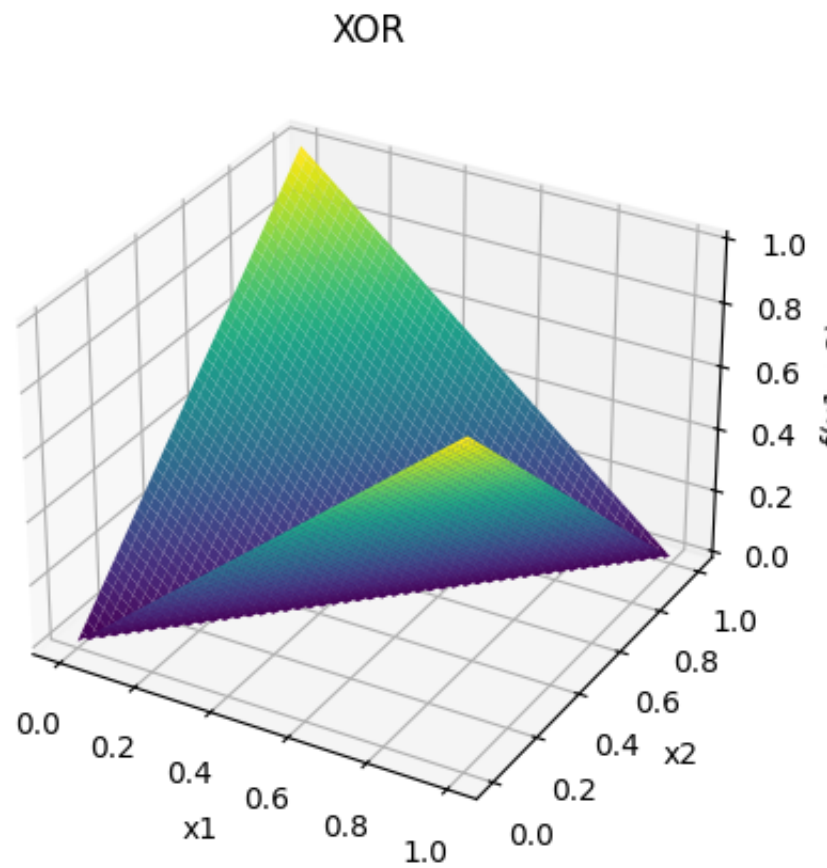
$$y = \text{relu} \left(\begin{bmatrix} x_1 & x_2 \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \right) \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix}$$

Question 1c If this network were trained using gradient descent for binary classification, what output activation function should be used so that the network's output can be interpreted as $\Pr(y=1 | x_1, x_2)$? What loss function would you use to train this model given that output activation?

Softmax activation and cross entropy loss - more specifically, sigmoid loss because the output is binary

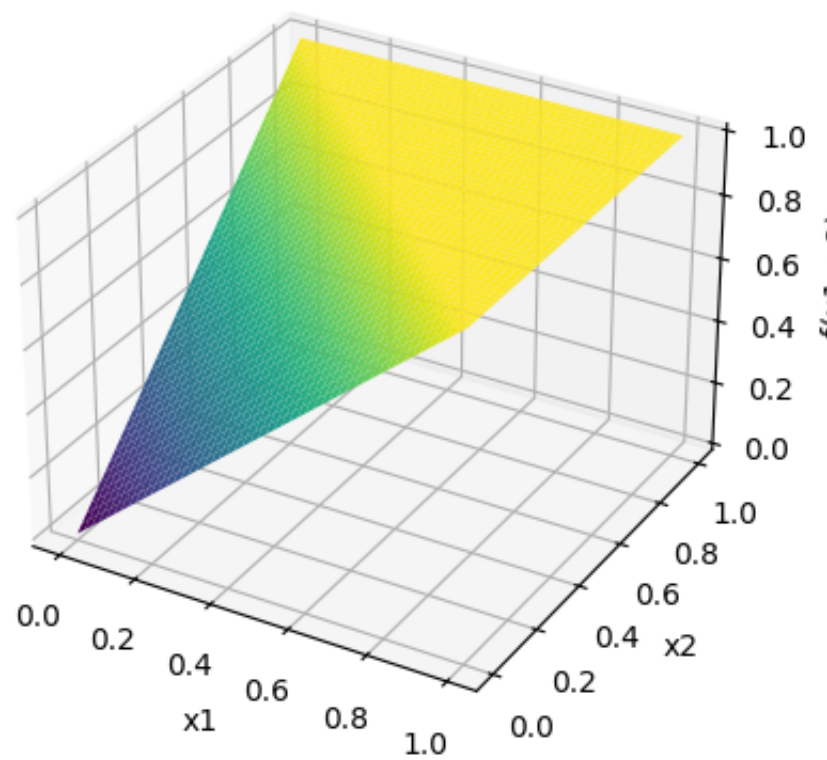
Question 1d For each of the 3 parameter sets you provided in part **b**, diagram the piecewise linear regions in the 2D input space defined by (x_1, x_2)

[Show code](#)

[Show code](#)

[Show code](#)

OR

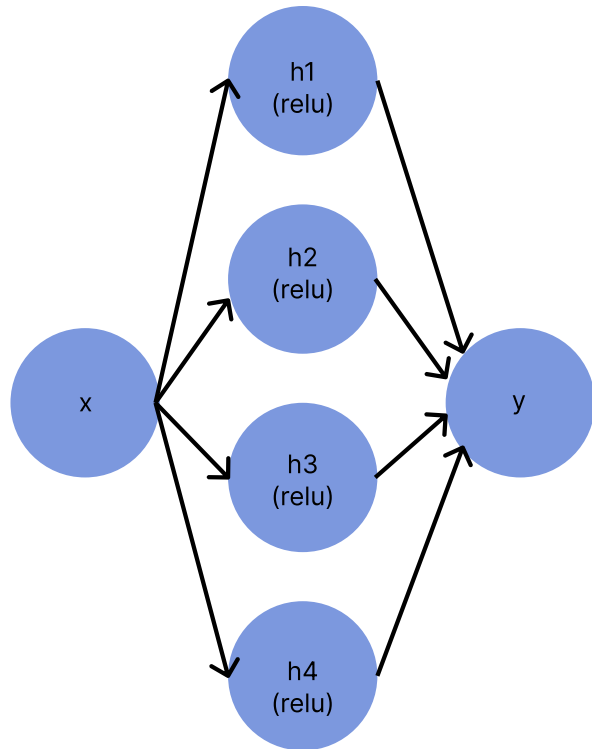


✓ Question 2: 1D Function Approximator

Given a set of inputs $\mathbf{x} = [x_1, x_2, x_3, x_4, x_5]^T$ where $x_1 < x_2 < \dots < x_5$, and a set of output values $\mathbf{y} = [y_1, y_2, y_3, y_4, y_5]^T$ design a neural network that approximates the function $f(\mathbf{x}) = \mathbf{y}$ and linearly interpolates between the observed values. Provide a diagram of your network, and values for any weights and biases as a function of \mathbf{x} & \mathbf{y} . Indicate where nonlinear activation functions are applied in your network and the functions that are used.

> network diagram

Show code



The diagram of the network is shown above. Similar to in problem 1, this is a shallow network. The output is given by the equation

$$y = \text{relu} \left(\begin{bmatrix} x \end{bmatrix} \cdot \begin{bmatrix} h_1 & h_2 & h_3 & h_4 \end{bmatrix} + \begin{bmatrix} b_1^1 & b_1^2 & b_1^3 & b_1^4 \end{bmatrix} \right) \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} + \begin{bmatrix} b_2 \end{bmatrix}$$

$$y = \text{relu} \left(\begin{bmatrix} x \end{bmatrix} \cdot \begin{bmatrix} h_1 & h_2 & h_3 & h_4 \end{bmatrix} + \begin{bmatrix} b_1^1 & b_1^2 & b_1^3 & b_1^4 \end{bmatrix} \right) \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} + \begin{bmatrix} b_2 \end{bmatrix}$$

In this case, we will set the values as follows:

$$y = \text{relu} \left(\begin{bmatrix} x \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} -x_1 & -x_2 & -x_3 & -x_4 \end{bmatrix} \right) \cdot \begin{bmatrix} \frac{y_2 - y_1}{x_2 - x_1} \\ \frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1} \\ \frac{y_4 - y_3}{x_4 - x_3} - \frac{y_3 - y_2}{x_2 - x_2} \\ \frac{y_5 - y_4}{x_5 - x_4} - \frac{y_4 - y_3}{x_4 - x_3} \end{bmatrix} + \begin{bmatrix} y_1 \end{bmatrix}$$

$$y = \text{relu} \left(\begin{bmatrix} x \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} -x_1 & -x_2 & -x_3 & -x_4 \end{bmatrix} \right) \cdot \begin{bmatrix} \frac{y_2 - y_1}{x_2 - x_1} \\ \frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1} \\ \frac{y_4 - y_3}{x_4 - x_3} - \frac{y_3 - y_2}{x_2 - x_2} \\ \frac{y_5 - y_4}{x_5 - x_4} - \frac{y_4 - y_3}{x_4 - x_3} \end{bmatrix} + \begin{bmatrix} y_1 \end{bmatrix}$$

✓ Question 3: Linear Regions

This problem is only required for students in graduate listings of the course. It is extra credit for students in undergrad listings.

Prove the formula given in class for the maximum number of linear regions in a shallow network with ReLU activations, D_i input nodes, and D hidden nodes where $D_i < D$ is given by (corrected 3/5):

$$\sum_{j=0}^{D_i} \binom{D}{j} + \sum_{j=0}^{D_i-1} \binom{D-1}{j}$$

Approach: Each new hyperplane (ReLU joint) added to a D_i dimensional input space is itself a $(D_i - 1)$ dimensional subspace. The number of regions added by a new hyperplane to the input space is equal to the number of regions created in that $(D_i - 1)$ dimensional subspace by existing hyperplanes. You can use this observation to create a recurrence relation for the maximum number of regions added by the k th hyperplane.

Hint: you may find the following recurrence relation helpful. $\binom{N}{k} = \binom{N-1}{k-1} + \binom{N-1}{k}$;
 $\binom{N}{0} = 1$

Proof by induction:

Claim: for any $D, D_i \text{ s.t. } D_i < D$, the maximum number of linear regions in a shallow network with ReLU activations is given by $\sum_{j=0}^{D_i} \binom{D}{j}$.

Base case:

$D_i = 0 \rightarrow \sum_{j=0}^0 \binom{D}{j} = \binom{D}{0} = 1$
 $D_i = 0 \rightarrow \sum_{j=0}^0 \binom{D}{j} = \binom{D}{0} = 1$. This checks out - if there are 0 dimensions, there can only be 1 linear region.

Inductive step:

Assume true up to $D-1$ for any $D_i < D$

Prove for D .

Proof by induction:

Claim: for any $D, D_i \text{ s.t. } D_i < D$, the maximum number of linear regions in a shallow network with ReLU activations is given by $\sum_{j=0}^{D_i} \binom{D}{j}$.

Base case:

$D_i = 0 \rightarrow \sum_{j=0}^0 \binom{D}{j} = \binom{D}{0} = 1$
 $D_i = 0 \rightarrow \sum_{j=0}^0 \binom{D}{j} = \binom{D}{0} = 1$

$D = 0 \rightarrow \sum_{j=0}^{D_i} \binom{0}{j} = \binom{0}{0} = 0$
 $D = 0 \rightarrow \sum_{j=0}^{D_i} \binom{0}{j} = \binom{0}{0} = 0$

Inductive step:

Assume true up to $D-1$ for any $D_i < D$

Prove for $D \geq 1$.

When we add the D^{th} hyperplane to D -dimensional space, the max number of regions will be the max number of regions from first $D-1$ hyperplanes + the max number of intersections between D^{th} hyperplane (which is itself a $D-1$ dimensional subspace, as explained to us in the problem description) and $D-1$ previous hyperplanes.

Through our inductive assumption, we know max regions formed by $D-1$ hyperplanes in D space is given by $\sum_{j=0}^{D-1} \binom{D-1}{j}$. Likewise, we know that max regions formed by $D-1$ hyperplanes in $D-1$ dimensional space is given by $\sum_{j=0}^{D-1} \binom{D-1}{j}$.

If we add these values:

$\sum_{j=0}^{D-1} \binom{D-1}{j} + \sum_{j=0}^{D-1} \binom{D-1}{j} = 2 \sum_{j=0}^{D-1} \binom{D-1}{j}$
 $\sum_{j=0}^{D-1} \binom{D-1}{j} = 2^{D-1}$
 We can change the indexing of the first sum so that this looks more like the recurrence relationship provided in the hint: $\sum_{j=1}^D \binom{D-1}{j-1} + \sum_{j=0}^{D-1} \binom{D-1}{j} = 2^{D-1}$
 $\sum_{j=1}^D \binom{D-1}{j-1} + \sum_{j=0}^{D-1} \binom{D-1}{j} = 2^{D-1}$
 We can then adjust the second sum to index from 1 as well by pulling out the $j=0$ term. After that, we can use our recurrence relationship, followed by some simplification.
 $\sum_{j=1}^D \binom{D-1}{j-1} + \binom{D-1}{0} + \sum_{j=1}^{D-1} \binom{D-1}{j} = 2^{D-1}$
 $\sum_{j=1}^D \binom{D-1}{j-1} + 1 = \sum_{j=1}^D \binom{D-1}{j-1} + \binom{D-1}{0} = \sum_{j=0}^{D-1} \binom{D-1}{j} = 2^{D-1}$
 $\sum_{j=1}^D \binom{D-1}{j-1} + \binom{D-1}{0} + \sum_{j=1}^{D-1} \binom{D-1}{j} = \sum_{j=1}^D \binom{D-1}{j-1} + 1 = \sum_{j=1}^D \binom{D-1}{j-1} + \binom{D-1}{0} = \sum_{j=0}^{D-1} \binom{D-1}{j} = 2^{D-1}$

Note that the replacement of 1 with $\binom{D}{0}$ follows from the fact $\binom{D}{0} = 1$ for any $D \geq 0$.

✓ Programming Questions

```
# USE THIS CODE IF YOU ARE USING GOOGLE COLABORATORY
# this section connects the notebook to your google drive, allowing you to
# access files from and upload files to your google drive
from google.colab import drive
drive.mount('/content/drive')

# set current directory
# this should be the Google Drive folder where your file(s) are located
%cd /content/drive/MyDrive/Comp_bio_pset1

## verify current directory
!ls /content/drive/MyDrive/Comp_bio_pset1

# choose where you want your project files to be saved
project_folder = "/content/drive/MyDrive/Comp_bio_pset1"
```

```
Mounted at /content/drive
/content/drive/MyDrive/Comp_bio_pset1
GTEx_Analysis_v8_Annotations_SampleAttributesDD.xlsx    GTEx_metadata.txt
GTEx_Analysis_v8_Annotations_SampleAttributesDS.txt    GTEx_numeric_data.txt
GTEx_Analysis_v8_Annotations_SubjectPhenotypesDD.xlsx  pset1.ipynb
GTEx_Analysis_v8_Annotations_SubjectPhenotypesDS.txt
```

✓ Question 4: Neural Networks for Gene Expression

In this problem, you will implement and train a neural network on gene expression data from GTEx. This network will predict tissue from the gene expression observed. To complete these, you will fill in (and run) the included code below, and use the three files distributed with this PSet:

```
GTEx_metadata.txt  
GTEx_numeric_data.txt  
GTEx_Analysis_v8_Annotations_SampleAttributesDS.txt
```

Question 4a *Preprocessing - Normalization and Standardization.*

The code below load gene expression data, process it into a custom `GeneExpressionDataset`, and produces `DataLoaders` for training and test data.

Extend the provided code to normalize and standardize each gene's expression levels across samples. For normalization, ensure each gene's expression levels have a mean of 0 by subtracting the mean for that gene across samples; for standardization, ensure a standard deviation of 1 by dividing each value by the standard deviation for that gene across samples.

```
# Load numeric tpm(transcripts per million) data and metadata

import pandas as pd

df_sample_info = pd.read_csv('GTEx_Analysis_v8_Annotations_SampleAttributesDS.txt', sep='\t')

# Looked up column IDs from Excel spreadsheet: GTEx_Analysis_v8_Annotations_SampleAttributesDD.x
sample_type_detailed = df_sample_info['SMTSD']
sample_type = df_sample_info['SMTS']
sample_id = df_sample_info['SAMPID']

name_to_type_dict = dict(zip(sample_id, sample_type_detailed))

metadata_file_path = 'GTEx_metadata.txt'
numeric_data_file_path = 'GTEx_numeric_data.txt'

# Load the numeric data
import numpy as np
import torch

def load_numeric_data_to_tensor(file_path, dtype=torch.float32):
    numeric_data = np.loadtxt(file_path, delimiter='\t', dtype=np.float32)
    tensor = torch.tensor(numeric_data, dtype=dtype)
    return tensor

numeric_tensor = load_numeric_data_to_tensor(numeric_data_file_path)

# Load the metadata (sample names)
with open(metadata_file_path, 'r') as metadata_file:
    sample_names = metadata_file.readline().strip().split('\t')
```

```

# Define Dataset class

from torch.utils.data import Dataset

class GeneExpressionDataset(Dataset):
    def __init__(self, data_tensor, sample_names, name_to_type_dict):
        """
        Args:
            data_tensor (torch.Tensor): Tensor of shape (genes, samples) that needs to be transp
            sample_names (list): List of sample names corresponding to the columns in the data t
            name_to_type_dict (dict): Dictionary mapping sample names to tissue types.
        """
        # Filter valid samples based on presence in name_to_type_dict
        valid_indices = [i for i, name in enumerate(sample_names) if name in name_to_type_dict]

        # Ensure data is in the correct shape: samples in rows, genes in columns
        # And filter out samples not present in name_to_type_dict
        self.data_tensor = data_tensor.transpose(0, 1)[valid_indices] # Transpose and filter
        print(f"data_tensor shape {self.data_tensor.shape}")
        print(f"mean shape {torch.mean(self.data_tensor, axis=0).shape}")
        print(f"std shape {torch.std(self.data_tensor, axis=0).shape}")
        self.data_tensor = (self.data_tensor - torch.mean(self.data_tensor, axis=0, keepdim=True)
        self.data_tensor = torch.nan_to_num(self.data_tensor, nan=0.0, posinf=0.0, neginf=0.0)
        #TODO: is this ok

        # Filter sample names and labels
        self.sample_names = [sample_names[i] for i in valid_indices]
        self.labels = [name_to_type_dict[name] for name in self.sample_names]

        # Convert labels to a tensor, assuming labels are categorical and need to be converted t
        self.label_set = sorted(list(set(self.labels))) # Get unique labels and sort them
        self.label_to_index = {label: index for index, label in enumerate(self.label_set)}
        self.label_indices = torch.tensor([self.label_to_index[label] for label in self.labels],

```

```
def __len__(self):
    return len(self.sample_names)

def __getitem__(self, idx):
    """
    Args:
        idx (int): Index of the sample to retrieve.

    Returns:
        tuple: (sample, label) where sample is a tensor of gene expressions and label is the
    """
    sample = self.data_tensor[idx].float()
    label = self.label_indices[idx]
    return sample, label
```

```
dataset = GeneExpressionDataset(numeric_tensor, sample_names, name_to_type_dict)

from torch.utils.data import DataLoader, random_split

# Assuming `dataset` is an instance of GeneExpressionDataset or similar
dataset_size = len(dataset)
train_size = int(dataset_size * 0.8) # 80% of the dataset for training
test_size = dataset_size - train_size # 20% for testing

# Splitting the dataset
torch.manual_seed(42)
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

# Creating data loaders for each set
batch_size = 32 # You can adjust the batch size according to your needs

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

data_tensor shape torch.Size([17381, 56200])
mean shape torch.Size([56200])
mean shape torch.Size([56200])
```

Question 4b *Define your model.*

Fill in the model definition below to create a simple neural network for predicting tissue type. Your model should:

- have input size equal to the number of genes,
- have two layers with hidden dimension $D_h = 1024$,
- include a non-linear activation function of your choice after the first layer, and
- output logits for the probability of each tissue type as defined in `dataset.label_set`.

Once you define your model, report the total number of trainable parameters. (Hint: see `model.parameters()`)

Hint: You can use [torch.nn.Sequential](#) to combine multiple layers into one model without having to define your own class. Also, you do not need to compute probabilities from the output logits.

```
# Define the model

from torch import nn
from torch.nn import functional as F

input_dim = dataset.data_tensor.shape[1]
output_dim = len(dataset.label_set)
model = torch.nn.Sequential(
    torch.nn.Linear(56200,1024),
    torch.nn.ReLU(),
    torch.nn.Linear(1024,54),
    # torch.nn.Softmax(dim=1)
)
print(model.parameters)
print(f"number of trainable parameters {sum(p.numel() for p in model.parameters() if p.requires_
# source: https://discuss.pytorch.org/t/how-do-i-check-the-number-of-parameters-of-a-model/4325
```

```
<bound method Module.parameters of Sequential(
  (0): Linear(in_features=56200, out_features=1024, bias=True)
  (1): ReLU()
  (2): Linear(in_features=1024, out_features=54, bias=True)
)>
number of trainable parameters 57605174
```

Question 4c *Training Loop*

Complete the model training loop below (wherever you see `...`) and execute it for a few epochs on the training data. You should see the training loss decline, indicating that the model is learning. You can use the optimizer of your choice.

```
import torch.optim as optim
```

```
import nn
import torch

# Instantiate the loss function
criterion = nn.CrossEntropyLoss()

# Define the optimizer
learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Number of epochs to train for
num_epochs = 5

# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

for epoch in range(num_epochs):
    model.train() # Set the model to training mode
    running_loss = 0.0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        # print(f"inputs {inputs}")
        # if torch.isnan(inputs).any():
        #     print("Tensor contains NaNs")
        # print(f"labels {labels}")
        # Zero the parameter gradients
        model.zero_grad()

        # Forward pass
        y_pred = model(inputs)
        # print(f"y_pred {y_pred}")
        # break
```

```
# Compute and print loss. We pass Tensors containing the predicted and true
# values of y, and the loss function returns a Tensor containing the
# loss.
loss = criterion(y_pred, labels)

# Backward pass and optimize
loss.backward()
optimizer.step()

running_loss += loss.item()

print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader)}")
```

```
Epoch [1/5], Loss: 0.4431028325324771
Epoch [2/5], Loss: 0.12141663229197328
Epoch [3/5], Loss: 0.07185837044802108
Epoch [4/5], Loss: 0.046542243227220795
Epoch [5/5], Loss: 0.06927748108398933
```

Question 4d *Evaluation*

Now that your model has been trained a bit, fill in and run the code below to evaluate the loss and accuracy (fraction of samples correctly predicted) on the test set. Don't worry if the performance is not that good - your model may just need to train more.

Hint: The test set is returned `test_loader`, which behaves similarly to `train_loader` except that items are not shuffled. What needs to be done to switch the `model` from training to evaluation usage?

```
# Evaluate the model
total = 0
correct = 0

model.eval()

for inputs, labels in test_loader:
    inputs, labels = inputs.to(device), labels.to(device)
    with torch.no_grad():
        outputs = model(inputs)
        max_values, max_indices = torch.max(outputs, dim=1)
        # print(max_values.shape)
        # print(max_indices.shape)
        # print(max_indices)
        # print(labels)
        # print("---")
        correct += torch.eq(labels, max_indices).sum()
        total += len(labels)

print(f"Accuracy on test set: {100 * correct / total}%")
```

Accuracy on test set: 93.90278625488281%

✓ Question 5: More Classification

This problem is extra credit for all students.

Inspect the GTEx metadata in the included excel file (e.g., age, sex) and choose one additional field (column), other than tissue, to predict from the gene expression patterns. Design and train a model for this task, discussing the chosen field, model architecture, and the results achieved.

```
# Load numeric tpm(transcripts per million) data and metadata

import pandas as pd

df_pheno_info = pd.read_csv('/content/GTEx_Analysis_v8_Annotations_SubjectPhenotypesDS.txt', sep
df_sample_info = pd.read_csv('GTEx_Analysis_v8_Annotations_SampleAttributesDS.txt', sep='\t')

df_sample_info["SUBJID"] = df_sample_info["SAMPID"].str.split("-").str[0:2].str.join("-")
merged_df = df_sample_info.merge(df_pheno_info, on="SUBJID", how="left")
print(merged_df[["SUBJID", "SAMPID", "AGE"]].head())
print(merged_df.head())
print(merged_df.columns)
sample_id = merged_df['SAMPID']

# Looked up column IDs from Excel spreadsheet: GTEx_Analysis_v8_Annotations_SampleAttributesDD.x
sample_age = df_pheno_info['AGE']

name_to_type_dict = dict(zip(sample_id, sample_age))

metadata_file_path = 'GTEx_metadata.txt'
numeric_data_file_path = 'GTEx_numeric_data.txt'
```

```
# Load the numeric data
import numpy as np
import torch

def load_numeric_data_to_tensor(file_path, dtype=torch.float32):
    numeric_data = np.loadtxt(file_path, delimiter='\t', dtype=np.float32)
    tensor = torch.tensor(numeric_data, dtype=dtype)
    return tensor

numeric_tensor = load_numeric_data_to_tensor(numeric_data_file_path)

# Load the metadata (sample names)
with open(metadata_file_path, 'r') as metadata_file:
    sample_names = metadata_file.readline().strip().split('\t')
```

	SUBJID	SAMPID	AGE				
0	GTEX-1117F	GTEX-1117F-0003-SM-58Q7G	60-69				
1	GTEX-1117F	GTEX-1117F-0003-SM-5DWSB	60-69				
2	GTEX-1117F	GTEX-1117F-0003-SM-6WBT7	60-69				
3	GTEX-1117F	GTEX-1117F-0011-R10a-SM-AHZ7F	60-69				
4	GTEX-1117F	GTEX-1117F-0011-R10b-SM-CYKQ8	60-69				
	SAMPID	SMATSSCR	SMCENTER	SMPHNTS	SMRIN	SMTS	\
0	GTEX-1117F-0003-SM-58Q7G	NaN	B1	NaN	NaN	Blood	
1	GTEX-1117F-0003-SM-5DWSB	NaN	B1	NaN	NaN	Blood	
2	GTEX-1117F-0003-SM-6WBT7	NaN	B1	NaN	NaN	Blood	
3	GTEX-1117F-0011-R10a-SM-AHZ7F	NaN	B1, A1	NaN	NaN	Brain	
4	GTEX-1117F-0011-R10b-SM-CYKQ8	NaN	B1, A1	NaN	7.2	Brain	
	SMTSD	SMUBRID	SMTSISCH	SMTSPAX	...	SME1PCTS	\
0	Whole Blood	0013756	1188.0	NaN	...	NaN	
1	Whole Blood	0013756	1188.0	NaN	...	NaN	
2	Whole Blood	0013756	1188.0	NaN	...	NaN	
3	Brain - Frontal Cortex (BA9)	0009834	1193.0	NaN	...	NaN	
4	Brain - Frontal Cortex (BA9)	0009834	1193.0	NaN	...	NaN	

	SMRRNART	SME1MPRT	SMNUM5CD	SMDPMPRT	SME2PCTS	SUBJID	SEX	AGE	DTHHRDY
0	NaN	NaN	NaN	NaN	NaN	GTEX-1117F	2	60-69	4.0
1	NaN	NaN	NaN	NaN	NaN	GTEX-1117F	2	60-69	4.0
2	NaN	NaN	NaN	NaN	NaN	GTEX-1117F	2	60-69	4.0
3	NaN	NaN	NaN	NaN	NaN	GTEX-1117F	2	60-69	4.0
4	NaN	NaN	NaN	NaN	NaN	GTEX-1117F	2	60-69	4.0

[5 rows x 67 columns]

```
Index(['SAMPID', 'SMATSSCR', 'SMCENTER', 'SMPTHNTS', 'SMRIN', 'SMTS', 'SMTSD',
      'SMUBRID', 'SMTSISCH', 'SMTSPAX', 'SMNABTCH', 'SMNABTCHT', 'SMNABTCHD',
      'SMGEBTCH', 'SMGEBTCHD', 'SMGEBTCHT', 'SMAFRZE', 'SMGTC', 'SME2MPRT',
      'SMCHMPRS', 'SMNTRART', 'SMNUMGPS', 'SMMAPRT', 'SMEXNCRT', 'SM550NRM',
      'SMGNSDTC', 'SMUNMPRT', 'SM350NRM', 'SMRDLGTH', 'SMMNCPB', 'SME1MMRT',
      'SMSFLGTH', 'SMESTLBS', 'SMMPPD', 'SMNTERRT', 'SMRRNANM', 'SMRD TTL',
      'SMVQCFL', 'SMMNCV', 'SMTRSCPT', 'SMMPPDPR', 'SMCGLGTH', 'SMGAPPCT',
      'SMUNPDRD', 'SMNTRNRT', 'SMMPUNRT', 'SMEXPEFF', 'SMMPPDUN', 'SME2MMRT',
      'SME2ANTI', 'SMALTALG', 'SME2SNSE', 'SMMFLGTH', 'SME1ANTI', 'SMSPLTRD',
      'SMBSMMRT', 'SME1SNSE', 'SME1PCTS', 'SMRRNART', 'SME1MPRT', 'SMNUM5CD',
      'SMDPMPRT', 'SME2PCTS', 'SUBJID', 'SEX', 'AGE', 'DTHHRDY'],
      dtype='object')
```

```
sample_age.value_counts()
```

	count
AGE	
60-69	317
50-59	315
40-49	153
20-29	84
30-39	78
70-79	33

dtype: int64

```
# Define Dataset class
```

```
from torch.utils.data import Dataset
```

```
class GeneExpressionDataset(Dataset):
```

```
    def __init__(self, data_tensor, sample_names, name_to_type_dict):
```

```
        """
```

```
        Args:
```

```
            data_tensor (torch.Tensor): Tensor of shape (genes, samples) that needs to be transp
```

```
            sample_names (list): List of sample names corresponding to the columns in the data t
```

```
            name_to_type_dict (dict): Dictionary mapping sample names to tissue types.
```

```
        """
```

```
        # Filter valid samples based on presence in name_to_type_dict
```

```
        valid_indices = [i for i, name in enumerate(sample_names) if name in name_to_type_dict]
```

```

# Ensure data is in the correct shape: samples in rows, genes in columns
# And filter out samples not present in name_to_type_dict
self.data_tensor = data_tensor.transpose(0, 1)[valid_indices] # Transpose and filter
print(f"data_tensor shape {self.data_tensor.shape}")
print(f"mean shape {torch.mean(self.data_tensor, axis=0).shape}")
print(f"std shape {torch.std(self.data_tensor, axis=0).shape}")
self.data_tensor = (self.data_tensor - torch.mean(self.data_tensor, axis=0, keepdim=True)
self.data_tensor = torch.nan_to_num(self.data_tensor, nan=0.0, posinf=0.0, neginf=0.0)
#TODO: is this ok

# Filter sample names and labels
self.sample_names = [sample_names[i] for i in valid_indices]
self.labels = [name_to_type_dict[name] for name in self.sample_names]

# Convert labels to a tensor, assuming labels are categorical and need to be converted to
self.label_set = sorted(list(set(self.labels))) # Get unique labels and sort them
print(self.label_set)
self.label_to_index = {label: index for index, label in enumerate(self.label_set)}
print(self.label_to_index)
self.label_indices = torch.tensor([self.label_to_index[label] for label in self.labels],

def __len__(self):
    return len(self.sample_names)

def __getitem__(self, idx):
    """
    Args:
        idx (int): Index of the sample to retrieve.

    Returns:
        tuple: (sample, label) where sample is a tensor of gene expressions and label is the
    """

```

```
        sample = self.data_tensor[idx].float()
        label = self.label_indices[idx]
        return sample, label

dataset = GeneExpressionDataset(numeric_tensor, sample_names, name_to_type_dict)

from torch.utils.data import DataLoader, random_split

# Assuming `dataset` is an instance of GeneExpressionDataset or similar
dataset_size = len(dataset)
train_size = int(dataset_size * 0.8) # 80% of the dataset for training
test_size = dataset_size - train_size # 20% for testing

# Splitting the dataset
torch.manual_seed(42)
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

# Creating data loaders for each set
batch_size = 32 # You can adjust the batch size according to your needs

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Define the model

from torch import nn
from torch.nn import functional as F

input_dim = dataset.data_tensor.shape[1]
output_dim = len(dataset.label_set)
model = torch.nn.Sequential(
    torch.nn.Linear(56200, 1024),
    torch.nn.ReLU(),
```

```
        torch.nn.Linear(1024,1024),
        torch.nn.ReLU(),
        torch.nn.Linear(1024,6),
        # torch.nn.Softmax(dim=1)
    )
print(model.parameters)
print(f"number of trainable parameters {sum(p.numel() for p in model.parameters() if p.requires_
# source: https://discuss.pytorch.org/t/how-do-i-check-the-number-of-parameters-of-a-model/4325

import torch.optim as optim

# Instantiate the loss function
criterion = nn.CrossEntropyLoss()

# Define the optimizer
learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Number of epochs to train for
num_epochs = 5

# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

for epoch in range(num_epochs):
    model.train() # Set the model to training mode
    running_loss = 0.0

    for inputs, labels in train_loader:
        # print(labels)
        inputs, labels = inputs.to(device), labels.to(device)
        # print(f"inputs {inputs}")
```

```
# if torch.isnan(inputs).any():
#     print("Tensor contains NaNs")
# print(f"labels {labels}")
# Zero the parameter gradients
model.zero_grad()

# Forward pass
y_pred = model(inputs)
# print(f"y_pred {y_pred}")
# break

# Compute and print loss. We pass Tensors containing the predicted and true
# values of y, and the loss function returns a Tensor containing the
# loss.
loss = criterion(y_pred, labels)

# Backward pass and optimize
loss.backward()
optimizer.step()

running_loss += loss.item()

print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader)}")

# Evaluate the model
total = 0
correct = 0

model.eval()

for inputs, labels in test_loader:
    inputs, labels = inputs.to(device), labels.to(device)
```

```

with torch.no_grad():
    outputs = model(inputs)
    max_values, max_indices = torch.max(outputs,dim=1)
    print(max_indices)
    print(labels)
    # print(max_values.shape)
    # print(max_indices.shape)
    # print(max_indices)
    # print(labels)
    # print("----")
    correct += torch.eq(labels,max_indices).sum()
    total += len(labels)

print(f"Accuracy on test set: {100 * correct / total}%")

```

```

data_tensor shape torch.Size([807, 56200])
mean shape torch.Size([56200])
mean shape torch.Size([56200])
['20-29', '30-39', '40-49', '50-59', '60-69', '70-79']
{'20-29': 0, '30-39': 1, '40-49': 2, '50-59': 3, '60-69': 4, '70-79': 5}
<bound method Module.parameters of Sequential(
  (0): Linear(in_features=56200, out_features=1024, bias=True)
  (1): ReLU()
  (2): Linear(in_features=1024, out_features=1024, bias=True)
  (3): ReLU()
  (4): Linear(in_features=1024, out_features=6, bias=True)
)>
number of trainable parameters 58605574
Epoch [1/5], Loss: 2.058188966342381
Epoch [2/5], Loss: 1.2261386286644709
Epoch [3/5], Loss: 0.6573668789295923
Epoch [4/5], Loss: 0.30298025835128056
Epoch [5/5], Loss: 0.17907019704580307

```

```

tensor([3, 4, 3, 4, 5, 4, 4, 4, 4, 4, 2, 4, 4, 1, 4, 4, 3, 3, 4, 3, 3, 4, 0, 2,
        4, 3, 4, 3, 3, 3, 4, 3], device='cuda:0')
tensor([4, 4, 3, 4, 0, 3, 0, 0, 4, 4, 3, 4, 3, 3, 5, 4, 3, 1, 2, 3, 2, 2, 3, 0,
        4, 4, 4, 3, 4, 2, 3, 3], device='cuda:0')
tensor([3, 3, 0, 4, 2, 4, 0, 2, 0, 4, 3, 4, 2, 4, 3, 4, 2, 3, 4, 2, 0, 4, 4, 0,
        0, 0, 3, 3, 3, 3, 3, 4], device='cuda:0')
tensor([3, 2, 4, 4, 4, 2, 4, 4, 2, 4, 1, 5, 2, 2, 4, 4, 3, 3, 3, 4, 4, 3, 5, 3,
        0, 4, 2, 0, 4, 3, 5, 4], device='cuda:0')
tensor([4, 3, 2, 3, 3, 3, 0, 4, 2, 4, 4, 4, 3, 3, 3, 3, 2, 2, 3, 4, 4, 3, 3, 4,
        4, 3, 3, 4, 3, 4, 4, 4], device='cuda:0')
tensor([2, 4, 0, 4, 3, 3, 1, 4, 4, 3, 3, 4, 1, 3, 1, 3, 4, 2, 3, 3, 4, 3, 1, 4,
        3, 4, 3, 4, 4, 4, 4, 2], device='cuda:0')
tensor([4, 3, 2, 4, 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, 4, 3, 4, 3, 0, 4, 3, 4, 2, 4,
        0, 3, 4, 4, 3, 4, 4, 3], device='cuda:0')
tensor([3, 4, 3, 4, 4, 4, 3, 3, 4, 4, 2, 3, 5, 1, 4, 3, 4, 4, 3, 2, 4, 3, 2, 3,
        3, 3, 2, 4, 3, 2, 3, 3], device='cuda:0')
tensor([3, 3, 4, 4, 3, 0, 3, 4, 3, 4, 4, 4, 3, 3, 4, 4, 3, 2, 4, 4, 4, 4, 3, 4,
        3, 4, 3, 4, 3, 4, 0, 4], device='cuda:0')
tensor([2, 4, 3, 3, 2, 3, 0, 1, 3, 1, 3, 4, 3, 3, 1, 4, 4, 3, 2, 0, 0, 4, 1, 3,
        1, 4, 2, 3, 4, 5, 4, 2], device='cuda:0')
tensor([3, 4], device='cuda:0')
tensor([3, 4], device='cuda:0')
Accuracy on test set: 36.41975402832031%

```

Discussion: I chose to train on a model to predict patient age. Looking at the labels, I see that patient age is broken into 10 year buckets, e.g. 50-59, 60-69. There are 6 categories in all. I experimented with adding additional layers to my model, since my initial accuracy was low (27%). I found adding a single additional hidden layer increased accuracy to 36%, but adding an additional hidden layer after that didn't help much. I can see that the data is not evenly distributed, with most samples coming from between ages 40 and 69, so this may also have contributed to low accuracy for less common ages. Since there are 317 samples in the 60-69 range, the model can achieve accuracy of ~30% just by always predicting that value. Therefore, the model's performance is not that impressive. It seems like age may be difficult to judge from gene expression alone. However, we might get better performance if we had age as a numerical variable (actual year) rather than a categorical one (10 year age bucket), in which case we could use MSE loss which rewards the model for being close to the correct value even if the exact year is not correct.

```
!apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-generic
!pip install nbconvert[webpdf]
!pip install --user -U nbconvert pandoc
!jupyter nbconvert /content/Copy_of_pset1-8.ipynb \
  --to pdf \
  --ClearOutputPreprocessor.enabled=True
# !jupyter nbconvert '/content/Copy_of_pset1.ipynb' --to webpdf
# !jupyter nbconvert '/content/Copy_of_pset1.ipynb' \
#   --to pdf \
#   --PDFExporter.latex_engine=xelatex \
#   --template classic
```

```
Requirement already satisfied: playwright in /usr/local/lib/python3.12/dist-packages (from nbconvert)
Requirement already satisfied: webencodings in /usr/local/lib/python3.12/dist-packages (from bleach)
Requirement already satisfied: tinycss2<1.5,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from bleach)
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.12/dist-packages (from nbconvert)
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.12/dist-packages (from nbconvert)
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.12/dist-packages (from nbconvert)
```

```

Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.12/dist-packages (from be
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: pyee<14,>=13 in /usr/local/lib/python3.12/dist-packages (from pla
Requirement already satisfied: greenlet<4.0.0,>=3.1.1 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.12/dist-packages (from js
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.12
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: rpds-py>=0.25.0 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: entrypoints in /usr/local/lib/python3.12/dist-packages (from jupy
Requirement already satisfied: nest-asyncio>=1.5.4 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: pyzmq>=23.0 in /usr/local/lib/python3.12/dist-packages (from jupy
Requirement already satisfied: tornado>=6.2 in /usr/local/lib/python3.12/dist-packages (from jup
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-
Requirement already satisfied: nbconvert in /usr/local/lib/python3.12/dist-packages (7.17.0)
Requirement already satisfied: pandoc in /root/.local/lib/python3.12/site-packages (2.4)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.12/dist-packages (from r
Requirement already satisfied: bleach!=5.0.0 in /usr/local/lib/python3.12/dist-packages (from bl
Requirement already satisfied: defusedxml in /usr/local/lib/python3.12/dist-packages (from nbcor
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.12/dist-packages (from nbcc
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: markupsafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: mistune<4,>=2.0.3 in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: nbformat>=5.7 in /usr/local/lib/python3.12/dist-packages (from nb
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from nbcon
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.12/dist-packages (
Requirement already satisfied: pygments>=2.4.1 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.12/dist-packages (from r
Requirement already satisfied: plumbum in /root/.local/lib/python3.12/site-packages (from pandoc
Requirement already satisfied: ply in /usr/local/lib/python3.12/dist-packages (from pandoc) (3.1
Requirement already satisfied: webencodings in /usr/local/lib/python3.12/dist-packages (from ble
Requirement already satisfied: tinycss2<1.5,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.12/dist-packages

```

```

Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.12/dist-packages (
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.12/dist-packages (from be
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.12/dist-packages (from js
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.12
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: rpds-py>=0.25.0 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: entrypoints in /usr/local/lib/python3.12/dist-packages (from jupy
Requirement already satisfied: nest-asyncio>=1.5.4 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: pyzmq>=23.0 in /usr/local/lib/python3.12/dist-packages (from jupy
Requirement already satisfied: tornado>=6.2 in /usr/local/lib/python3.12/dist-packages (from jup
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-
[NbConvertApp] Converting notebook /content/Copy_of_pset1-8.ipynb to pdf
[NbConvertApp] Writing 141247 bytes to notebook.tex

```

