

General Instructions

Please follow the given instructions below to submit your project PDF file attached to related python script as well.


Here's a step-by-step guide that students can follow to work with a Google Colab notebook, save their outputs as a PDF file, and submit to the designated Learn item. You can share these instructions directly with your students:

Step-by-Step Instructions for Working with Google Colab and Submitting Outputs

Step 1: Open the .ipynb File in Google Colab

1. Download the provided .ipynb file from Learn.
2. Open Google Colab by visiting [Google Colab](https://colab.research.google.com).
3. In Colab:
 - Click on File > Open Notebook.
 - Navigate to the Upload tab.
 - Click Choose File and upload the .ipynb file you downloaded.

Step 2: Work on the Notebook

1. Follow the instructions and complete all the required tasks in the notebook.
2. Run each cell by clicking the play button () next to it or pressing Shift + Enter.
3. Save your progress frequently by selecting File > Save or by clicking the save icon.

Step 3: Save the Notebook as a PDF

1. Once you've completed all tasks, export the notebook to PDF.
 - Click on File > Print.
 - In the print dialog, change the destination printer to Save as PDF.
 - Adjust the settings (e.g., landscape/portrait, margins) if necessary to ensure all content fits properly.
 - Click Save, and choose a location on your computer to store the PDF file.

Alternative Method:

If the direct "Print to PDF" method doesn't work or you encounter issues:

- Download the notebook as HTML:
 - File > Download > Download as HTML (.html).

- Open the HTML file in a browser and print it as a PDF.

Step 4: Verify the PDF

1. Open the saved PDF and check that:
 - All cells, outputs, and plots are included.
 - The content is clearly visible and formatted correctly.
2. If any content is missing or unclear, revisit the notebook and export again.

Step 5: Submit the PDF to Learn

1. Log in to your Learn platform.
2. Navigate to the related submission item.
3. Click on Submit Assignment or the equivalent option.
4. Upload your PDF file.
5. Confirm your submission by following the on-screen prompts.

Step 6: Retain Your Work

Keep a copy of both the `.ipynb` and PDF files for your records in case you need to resubmit or review them later.

Common Troubleshooting Tips

- **Google Colab Timeout:** Google Colab may time out due to inactivity. Save your progress often and refresh the page if needed.
- **File Size:** Ensure your PDF file isn't too large to upload. Compress if necessary using free online tools if it is necessary
- **Contact Support:** If you encounter technical issues, reach out to your instructor or support team promptly.

✓ Necessary Packages:

Using main libraries

We have specific widely used package

Here are the five most widely used Python libraries for data science:

1. NumPy

- **Purpose:** Efficient numerical computations, especially for handling large multi-dimensional arrays and matrices. Provides mathematical functions to operate on these arrays.

- **Use Cases:** Data manipulation, linear algebra, Fourier transforms, random number generation.

2. Pandas

- **Purpose:** Data manipulation and analysis, particularly for structured data. Works well with tabular data (e.g., CSV, Excel).
- **Use Cases:** Data cleaning, filtering, grouping, and merging datasets.

3. Matplotlib

- **Purpose:** Data visualization through static, animated, and interactive plots.
- **Use Cases:** Creating line charts, bar charts, scatter plots, and histograms.

4. Seaborn

- **Purpose:** More structured Data visualization on top of the matplotlib
- **Use Cases:** Creating line charts, bar charts, scatter plots, and histograms.

These libraries are foundational for data science and often used in conjunction to cover the entire data analysis and machine learning pipeline.

Using PyTorch

1. PyTorch

- **Purpose:** Deep learning frameworks for building and training neural networks.
- **Use Cases:** Computer vision, natural language processing, reinforcement learning, and other deep learning applications.

```
import numpy as np          # Import NumPy for numerical computations
import pandas as pd         # Import Pandas for data manipulation
import matplotlib.pyplot as plt # Import Matplotlib for plotting
import seaborn as sns      # Import Seaborn for plotting, if necessary beyond the matplotlib
from sklearn import datasets # Import scikit-learn for machine learning
```

```
import torch
import matplotlib.pyplot as plt
import torch.nn.functional as F
import matplotlib.pyplot as plt
import torch.nn as nn
```

Exercise 0 (15 pts): Knowledge on main concepts

Describe the given technical words aligning with the concept of neural network clearly and briefly;

- Learning rate

- Activation function
- Hidden layer and hidden units

✓ Solution

Learning Rate:

A hyperparameter that determines how quickly the weights are updated during neural network training. Small values: Slow but more precise learning Large values: Fast but less precise learning Chosen between 0.01 and 0.1 Affects the error effects and convergence effects of the model

Activation Function (Activation Function):

The output of the propagation in the neural network is the calculated bulbs It allows the model to learn complex patterns by performing non-linear transformations Common examples:

ReLU: Linear smoothing Sigmoid: Compression between 0-1 Tanh: Compression between -1 and 1

Hidden Layer (Hidden Layer) and Hidden Units (Hidden Units):

Hidden Layer: Intermediate layers between the input and output layers Hidden Units: Artifacts in these layers It allows the model to learn complex relationships Number of layers and number of units model effects Excessive units: Risk of over-learning Few units: Risk of under-learning

Exercise 1 (10 pts): Tensor Operations

Goal: Practice creating and manipulating tensors.

Create a 3D tensor of shape (3, 3, 3) filled with random integers between 1 and 10.

Perform the following operations:

- Slice the tensor to extract a 2D matrix from the first two dimensions.
- Reshape the tensor into a 1D vector.
- Perform element-wise addition with another randomly generated tensor of the same size.

✓ Solution

```
train_data = pd.read_csv('california_housing_train.csv')
test_data = pd.read_csv('california_housing_test.csv')
```

```
latitude = train_data['latitude'].values
longitude = train_data['longitude'].values
```

```

tensor_3d = torch.randint(1, 11, (3, 3, 3), dtype=torch.int32)

tensor_2d = torch.tensor([latitude[:3], longitude[:3]], dtype=torch.float32)

tensor_slice = tensor_3d[0, :, :]

tensor_resaped = tensor_3d.view(-1)

random_tensor = torch.randint(1, 11, (3, 3, 3), dtype=torch.int32)
elementwise_add = tensor_3d + random_tensor

print("Original 3D Tensor:\n", tensor_3d)
print("Sliced 2D Tensor (from latitude and longitude):\n", tensor_slice)
print("Reshaped 1D Tensor:\n", tensor_resaped)
print("Element-wise Addition:\n", elementwise_add)

```



Original 3D Tensor:

```

tensor([[[ 1,  1,  7],
         [ 5,  6,  6],
         [ 7,  6,  1]],

        [[10,  4,  9],
         [ 1,  6,  9],
         [ 8,  6,  9]],

```

```

        [[ 2,  7,  1],
         [10,  2,  6],
         [ 1,  3,  2]]], dtype=torch.int32)

```

Sliced 2D Tensor (from latitude and longitude):

```

tensor([[1, 1, 7],
        [5, 6, 6],
        [7, 6, 1]], dtype=torch.int32)

```

Reshaped 1D Tensor:

```

tensor([ 1,  1,  7,  5,  6,  6,  7,  6,  1, 10,  4,  9,  1,  6,  9,  8,  6,  9,
         2,  7,  1, 10,  2,  6,  1,  3,  2], dtype=torch.int32)

```

Element-wise Addition:

```

tensor([[[11,  3, 11],
         [13, 10, 13],
         [ 9, 16, 11]],

        [[12,  6, 18],
         [ 3,  7, 14],
         [12, 11, 17]],

```

```

        [[12, 11,  5],
         [15,  7, 10],
         [ 2,  4,  7]]], dtype=torch.int32)

```

Exercise 2 (10 pts): Linear Transformation

Goal: Implement basic matrix multiplication and understand its role in neural networks.

Create a tensor A of shape (3, 4) filled with random numbers and another tensor B of shape (4, 2).

- Perform a matrix multiplication between A and B to produce the output tensor C.
- Add a bias term to C using broadcasting.

✓ Solution

```
train_data = pd.read_csv('california_housing_train.csv')

latitude_longitude_tensor = torch.tensor(train_data[['latitude', 'longitude']].values, dt

A = torch.rand(3, 4)

B = torch.rand(4, 2)

C = torch.matmul(A, B)

bias = torch.rand(1, 2)
C_with_bias = C + bias

print("Matrix A (3x4):\n", A)
print("Matrix B (4x2):\n", B)
print("Matrix Multiplication Result (C = A * B):\n", C)
print("Matrix with Bias:\n", C_with_bias)
```

```
➡ Matrix A (3x4):
  tensor([[0.7474, 0.9929, 0.3680, 0.2264],
          [0.0575, 0.0173, 0.7175, 0.5276],
          [0.4002, 0.6767, 0.2072, 0.9666]])
Matrix B (4x2):
  tensor([[0.5343, 0.6311],
          [0.6595, 0.4934],
          [0.3018, 0.4839],
          [0.2738, 0.8272]])
Matrix Multiplication Result (C = A * B):
  tensor([[1.2272, 1.3270],
          [0.4032, 0.8285],
          [0.9874, 1.4864]])
Matrix with Bias:
  tensor([[1.8725, 1.4871],
          [1.0485, 0.9886],
          [1.6327, 1.6465]])
```

Exercise 3 (15 pts): Activation Functions

Goal: Implement and visualize common activation functions.

Create a range of inputs from -5 to 5 using `torch.linspace` function.

Implement the following activation functions as PyTorch functions:

- Sigmoid
- ReLU
- Tanh

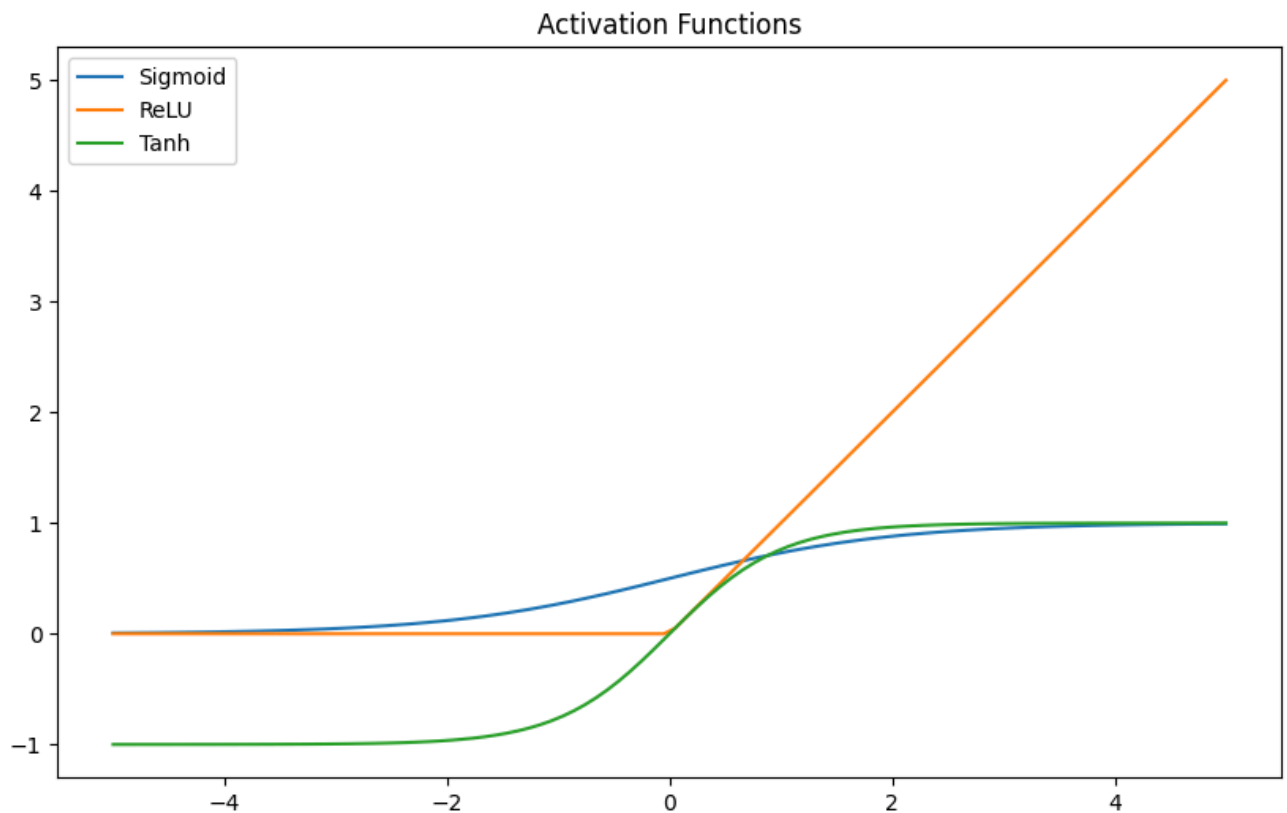
Plot the outputs of these activation functions for the input range using Matplotlib.

✓ Solution

```
x = torch.linspace(-5, 5, 100)
```

```
sigmoid = torch.sigmoid(x)
relu = torch.relu(x)
tanh = torch.tanh(x)
```

```
plt.figure(figsize=(10, 6))
plt.plot(x.numpy(), sigmoid.numpy(), label="Sigmoid")
plt.plot(x.numpy(), relu.numpy(), label="ReLU")
plt.plot(x.numpy(), tanh.numpy(), label="Tanh")
plt.legend()
plt.title("Activation Functions")
plt.show()
```



✓ Building and Training Neural Networks with PyTorch

In this section, we'll implement a neural network using PyTorch, following these steps:

- **Step 1: Define the Neural Network Class**
- **Step 2: Prepare the Data**
- **Step 3: Instantiate the Model, Loss Function, and Optimizer**
- **Step 4: Training the Model**
- **Step 5: Testing the Model**

Exercise 4 (10 pts): Defining the Neural Network

Goal: Create your NN class from the given structure.

Use the given visualization and assume the following details to write down your Neural Network (NN) Class as a python code, similar to what we have done before (including all the necessary modules and libraries). As a prior assumption, Sigmoid function is used for each space as activation one.



- Write down the class function without using the `nn.Sequential()` convention
- Create a similar class using the `nn.Sequential()` function instead.

✓ Solution

```
class NeuralNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, output_size)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.sigmoid(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
model_sequential = nn.Sequential(
    nn.Linear(2, 4),
    nn.Sigmoid(),
    nn.Linear(4, 1),
    nn.Sigmoid()
)
```

Exercise 5 (10 pts): Training the defined NN

Goal: To create a simulated data set to train the model on the training set

- Firstly, create a suitable random data as tensor based on your described NN class (the case without using the `nn.Sequential()` convention). Focus on creating first `x_train` and `y_train`.

Hint: You can benefit from `torch.rand` function for both your input and response data using the sample size as 700.

- Train your NN class model by adding the related intersteps
 - Instantiate the Model, Define Loss Function and Optimizer. You can use any suitable loss function based on your data set assuming that the problem is regression one. Use the default parameters for the considered optimizer as well
 - Train the model using 10 epochs
 - Get your predictions for the training data and compare with original response to calculate your errors

✓ Solution

```

input_size = 10
hidden_size = 5
output_size = 1

X_train = torch.rand(700, input_size)
y_train = torch.rand(700, output_size)

model = NeuralNetwork(input_size, hidden_size, output_size)
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

for epoch in range(10):
    outputs = model(X_train)
    loss = criterion(outputs, y_train)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if epoch % 2 == 0:
        print(f'Epoch [{epoch+1}/10], Loss: {loss.item():.4f}')

predictions = model(X_train)
error = torch.mean((predictions - y_train)**2)
print(f'Mean Square Error: {error.item():.4f}')

```

```

↔ Epoch [1/10], Loss: 0.3353
Epoch [3/10], Loss: 0.2906
Epoch [5/10], Loss: 0.2537
Epoch [7/10], Loss: 0.2233
Epoch [9/10], Loss: 0.1982
Mean Square Error: 0.1774

```

Exercise 6 (10 pts): Testing performance on a new data set

Goal: Now you need to create a new test data similar to training one to measure performance on a new data

For that purpose, you need to create your data set first with the size of 300 now

- Create the `x_test` and `y_test`
- Get your predictions of the trained model on the test data now to compare with the original `y_test` values
- Calculate your error again, similar to above case.

✓ Solution

```
X_test = torch.rand(300, input_size)
y_test = torch.rand(300, output_size)

model.eval()

with torch.no_grad():
    test_predictions = model(X_test)
    test_error = torch.mean((test_predictions - y_test)**2)
    print(f'Test Mean Square Error: {test_error.item():.4f}')
```

➞ Test Mean Square Error: 0.1866

Exercise 7 (20 pts): Impact of epoch and learning rate

Goal: Looking at the above training procedure, look at the impact of different parameters

- Now consider the number of epochs as 100, 500 as two different cases
- For the learning rate parameter, lr , use the values $lr=0.1$, $lr=0.01$ as alternative values instead of default parameter

Based on your experiment by changing the above parameter setting, discuss their impact on the **training** and **testing** loss calculation based on your model created above (over the same train data you created in the Exercise 5)

✓ Solution

```
def train_and_evaluate(epochs, lr):
    model = NeuralNetwork(input_size, hidden_size, output_size)
    criterion = nn.MSELoss()
    optimizer = torch.optim.SGD(model.parameters(), lr=lr)

    for epoch in range(epochs):
        outputs = model(X_train)
        loss = criterion(outputs, y_train)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    model.eval()
    with torch.no_grad():
```

```
test_predictions = model(X_test)
test_error = torch.mean((test_predictions - y_test)**2)

return test_error.item()

scenarios = [
    (10, 0.01),
    (100, 0.01),
    (500, 0.01),
    (10, 0.1),
    (10, 0.001)
]

print("Epoch | Learning Rate | Test error")
print("-" * 40)
for epochs, lr in scenarios:
    test_error = train_and_evaluate(epochs, lr)
    print(f"{epochs:5d} | {lr:12.3f} | {test_error:.4f}")
```

⇒

Epoch	Learning Rate	Test error
10	0.010	0.7172
100	0.010	0.0841
500	0.010	0.0850
10	0.100	0.0854
10	0.001	0.8845