Midterm 1 Project, Problem 1

```
# Import Dependencies
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
```

Methodology

We consider a target function:

$$u_g(x) = \sin(2x+1) + 0.2e^{1.3x}$$

1. We generate training data by sampling 300 data points from it without noise, uniformly distributed within the domain $x \in [-1, 1)$.

2. To fit the training data, we use a fully-connected neural network made of 3 hidden layers each with 20 units and use hyperbolic tangent (tanh) as the activation function.

This code implements a two-stage neural network training process to approximate a nonlinear target function sin(2 * x + 1) + 0.2 * exp(1.3 * x).

- It starts by generating 300 training points from -1 to 1 and calculates the target function values.
- In the first stage, a neural network with 3 hidden layers (each containing 30 neurons) is trained to fit the target function, yielding initial predictions and computing the residual (error).
- The second stage trains another neural network with 3 hidden layers (each containing 20 neurons) to specifically learn this residual.
- The final prediction is obtained by combining the outputs from both stages—adding the initial predictions to the learned residuals—enhancing the accuracy in approximating the target function.

```
# Target function from Equation (2)
def target_function(x):
    return np.sin(2 * x + 1) + 0.2 * np.exp(1.3 * x)

# Generate training data (300 points between -1 and 1)
x_train = np.linspace(-1, 1, 300)
y_train = target_function(x_train)

# Define the neural network model (custom layers)
def create_model(layers):
    model = Sequential()
    model.add(Dense(layers[0], activation='tanh', input_shape=(1,)))
```

```
    for units in layers[1:]:
        model.add(Dense(units, activation='tanh'))
    model.add(Dense(1))  # Output layer
    return model

# Compile and train the model
def train_model(model, x, y, epochs=6000, learning_rate=0.0001):
    model.compile(optimizer=Adam(learning_rate=learning_rate),
loss='mse')
    history = model.fit(x, y, epochs=epochs, verbose=0)
    return model, history

# Train Stage 1 (Initial training on the target function with 30 units
in each layer)
model1 = create_model([30, 30, 30])
model1, history1 = train_model(model1, x_train, y_train)

# Stage 1 predictions and error (residual)
y_pred1 = model1.predict(x_train).squeeze()
error1 = y_train - y_pred1  # Residual error from Stage 1

# Train Stage 2 - Network with 3 hidden layers (each with 20 units)
model2 = create_model([20, 20, 20])
model2, history2 = train_model(model2, x_train, error1)

# Stage 2 predictions and combined results
y_pred2 = model2.predict(x_train).squeeze()
final_prediction = y_pred1 + y_pred2  # Combined prediction

# Residual error after Stage 2
residual_final = y_train - final_prediction

# Combine loss history for both stages
combined_loss = np.concatenate((history1.history['loss'],
history2.history['loss']))

# Adjust epochs for Stage 2 to appear after Stage 1
stage2_start_epoch = len(history1.history['loss'])
```

```
WARNING:tensorflow:From C:\Users\jhyan\AppData\Roaming\Python\
Python311\site-packages\keras\src\backend.py:873: The name
tf.get_default_graph is deprecated. Please use
tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\jhyan\AppData\Roaming\Python\
Python311\site-packages\keras\src\utils\tf_utils.py:492: The name
tf.ragged.RaggedTensorValue is deprecated. Please use
tf.compat.v1.ragged.RaggedTensorValue instead.
```

```
10/10 [==============================] - 0s 990us/step
10/10 [==============================] - 0s 773us/step

# Plotting the results
plt.figure(figsize=(10, 12))

# Plot (a) Target function vs Stage 1 Prediction
plt.subplot(3, 2, 1)
plt.plot(x_train, y_train, label=r"$u_g(x) = \sin(2x + 1) +
0.2e^{1.3x}$", color='b')
plt.plot(x_train, y_pred1, label="NN approx. $u_0(x)$", color='r',
linestyle='--')
plt.title("First-stage Prediction $u_0(x)$")
plt.legend()
plt.grid(True)

# Plot (b) First-stage residual (e1)
plt.subplot(3, 2, 3)
plt.plot(x_train, error1, label=r"$e_1(x) = u_g(x) - u_0(x)$",
color='b')
plt.plot(x_train, y_pred2, label="2nd-stage NN: $u_1(x)$", color='r',
linestyle='--')
plt.title("First-stage Residual $e_1(x)$ and Second-stage NN")
plt.legend()
plt.grid(True)

# Plot (c) Second-stage residual (e2)
plt.subplot(3, 2, 5)
plt.plot(x_train, residual_final, label=r"$e_2(x) = e_1(x) - \
epsilon_1u_1(x)$", color='b')
plt.title("Second-stage Residual $e_2(x)$ after Stage 2")
plt.legend()
plt.grid(True)

# Plot (d) Combined loss for Stage 1 and Stage 2
plt.subplot(3, 2, 2)
plt.plot(np.arange(len(combined_loss)), combined_loss, label="Combined
Loss (Stage 1 and Stage 2)", color='black')

# Mark transition between Stage 1 and Stage 2
plt.axvline(stage2_start_epoch, color='blue', linestyle='--',
label="Start of Stage 2")

plt.yscale("log")
plt.title("Training Loss $L$ vs Epochs (Combined Stage 1 & 2)")
plt.xlabel("Epochs")
plt.ylabel("Loss (MSE)")
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```