MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
" IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE"

Volodymyr Shymkovych

Design and implementation of software systems with neural networks

Comparative Analysis of Neural Network Architectures Using TensorFlow

LABORATORY WORK #2

kulubecioglu mehmet
IM-14 FIOT

Kyiv
IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE
2024

1. Introduction

This project aims to compare the performance of different neural network architectures and configurations using the TensorFlow library. The architectures include feedforward, cascade, and Elman networks. Different numbers of layers will be tried for each architecture, and the results will be compared.

2. Code Structure and Workflow

2.1 Function and Data Generation

- true_function(x, y): Defines a simulated function. It is used to generate synthetic data in this project.
- Training data is provided by generating random samples from a specified distribution.

2.2 Network Architectures

- get_feedforward_arch(num_neurons): Returns a feedforward neural network architecture with the given number of neurons.
- get_cascade_arch(num_neurons_per_layer): Returns a cascade neural network architecture with the specified number of neurons between layers.
- get_elman_arch(num_neurons_per_layer): Returns an Elman network architecture with the specified number of neurons between layers. Elman networks are a type of recurrent neural network.

2.3 Training and Evaluation

- train_and_evaluate(model_architecture): Performs training on a given model architecture and returns the mean squared error.

2.4 Error Analysis and Visualization

- Training is conducted for different types of networks and numbers of layers, and errors are stored.
- The obtained error values are plotted graphically.

- Predictions made with the Elman network are visualized,particularly on the contour map of the simulated function.

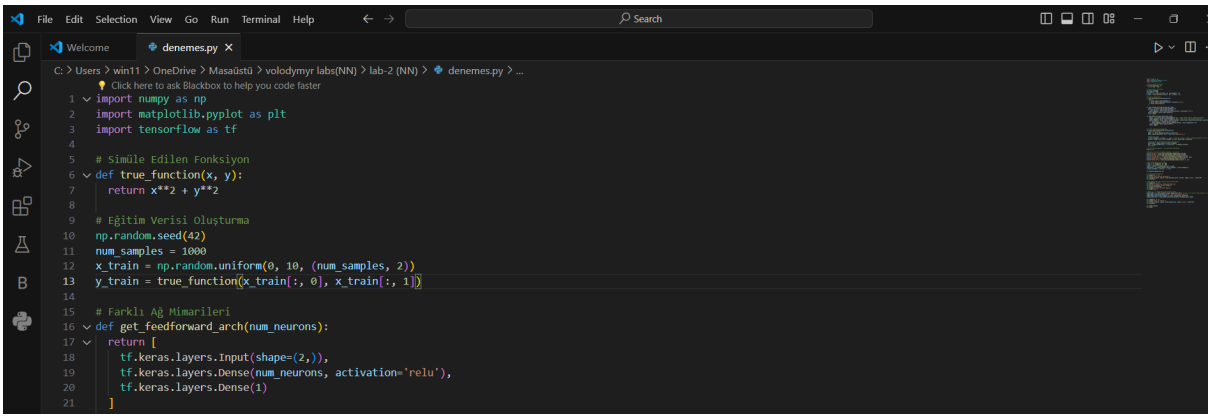
3. Results and Visualization

- The contour map of the simulated function is visualized.
- Mean squared error values for different network types and numbers of layers are shown in a bar graph.
- Contour maps of predictions made with the Elman network are visualized.

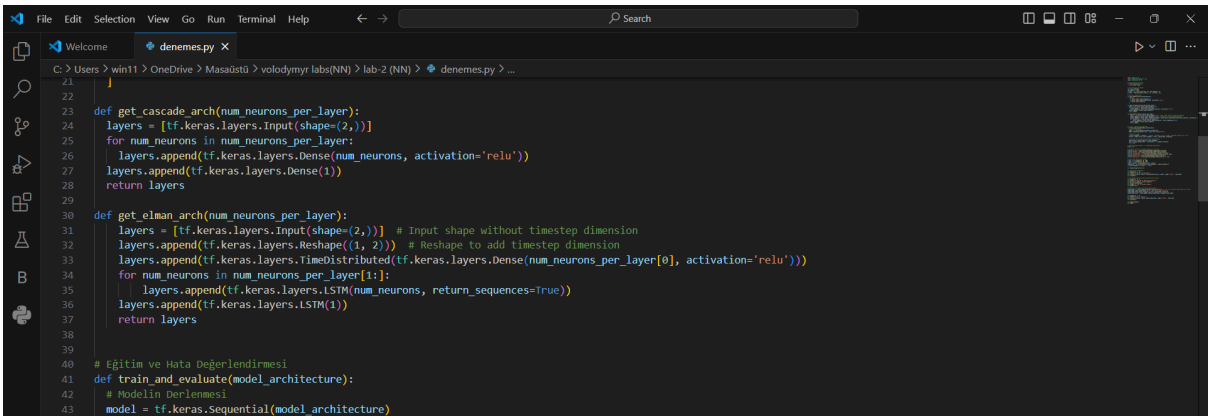
4. Conclusion

This study provides a framework for comparing the performance of different neural network architectures and configurations. Visualization and error analysis can be used to understand which architecture and configuration better fit a specific problem.

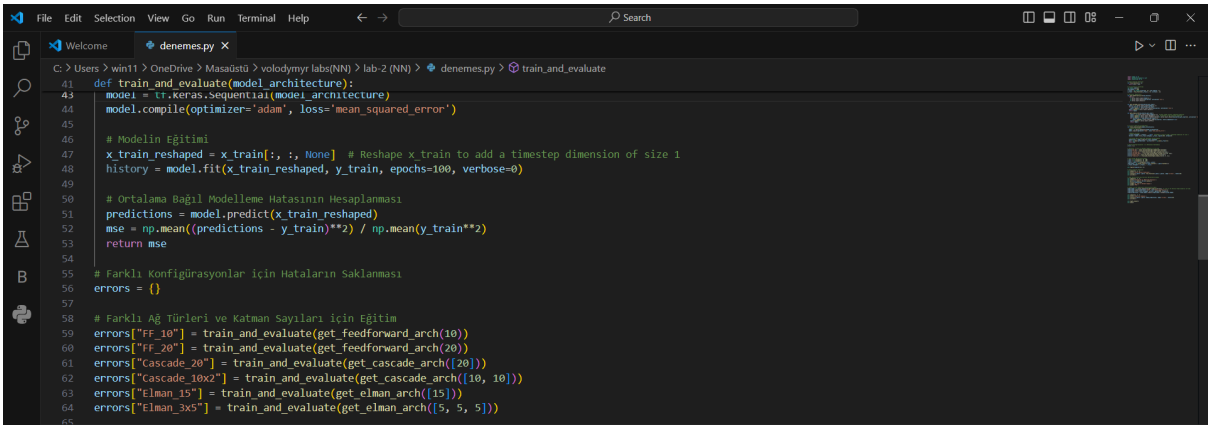
MY FULL CODES:



```python
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

# Simüle Edilen Fonksiyon
def true_function(x, y):
  return x**2 + y**2

# Eğitim Verisi Oluşturma
np.random.seed(42)
num_samples = 1000
x_train = np.random.uniform(0, 10, (num_samples, 2))
y_train = true_function(x_train[:, 0], x_train[:, 1])

# Farklı Ağ Mimarileri
def get_feedforward_arch(num_neurons):
  return [
    tf.keras.layers.Input(shape=(2,)),
    tf.keras.layers.Dense(num_neurons, activation='relu'),
    tf.keras.layers.Dense(1)
  ]
```

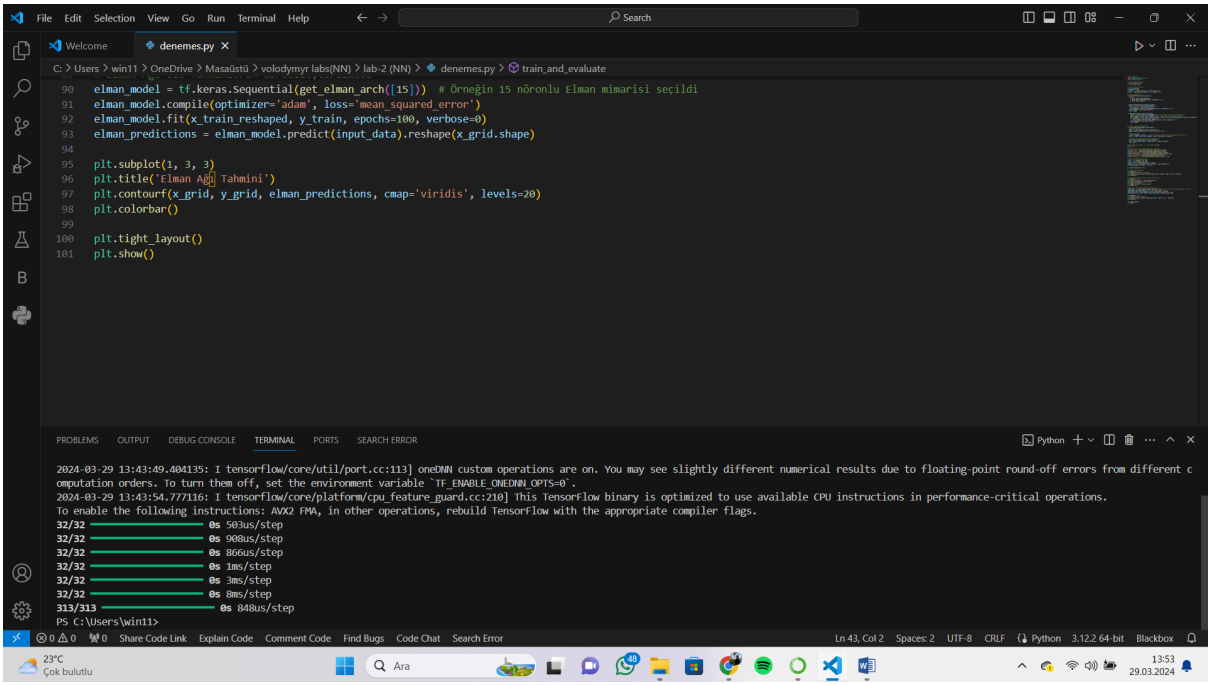


```python
  ]

def get_cascade_arch(num_neurons_per_layer):
  layers = [tf.keras.layers.Input(shape=(2,))]
  for num_neurons in num_neurons_per_layer:
    layers.append(tf.keras.layers.Dense(num_neurons, activation='relu'))
  layers.append(tf.keras.layers.Dense(1))
  return layers

def get_elman_arch(num_neurons_per_layer):
    layers = [tf.keras.layers.Input(shape=(2,))]  # Input shape without timestep dimension
    layers.append(tf.keras.layers.Reshape((1, 2)))  # Reshape to add timestep dimension
    layers.append(tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(num_neurons_per_layer[0], activation='relu')))
    for num_neurons in num_neurons_per_layer[1:]:
        layers.append(tf.keras.layers.LSTM(num_neurons, return_sequences=True))
    layers.append(tf.keras.layers.LSTM(1))
    return layers


# Eğitim ve Hata Değerlendirmesi
def train_and_evaluate(model_architecture):
  # Modelin Derlenmesi
  model = tf.keras.Sequential(model_architecture)
```



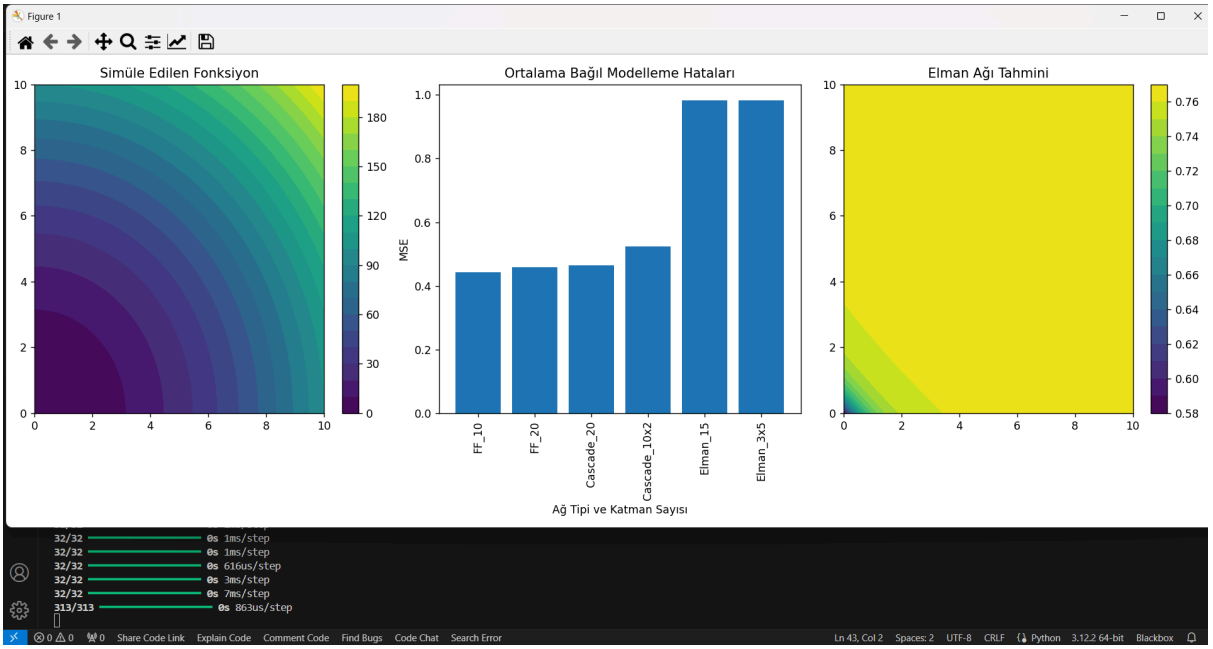
```
def train_and_evaluate(model_architecture):
    model = tf.keras.Sequential(model_architecture)
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Modelin Eğitimi
    x_train_reshaped = x_train[:, :, None]  # Reshape x_train to add a timestep dimension of size 1
    history = model.fit(x_train_reshaped, y_train, epochs=100, verbose=0)

    # Ortalama Bağıl Modelleme Hatasının Hesaplanması
    predictions = model.predict(x_train_reshaped)
    mse = np.mean((predictions - y_train)**2) / np.mean(y_train**2)
    return mse

# Farklı Konfigürasyonlar için Hataların Saklanması
errors = {}

# Farklı Ağ Türleri ve Katman Sayıları için Eğitim
errors["FF_10"] = train_and_evaluate(get_feedforward_arch(10))
errors["FF_20"] = train_and_evaluate(get_feedforward_arch(20))
errors["Cascade_20"] = train_and_evaluate(get_cascade_arch([20]))
errors["Cascade_10x2"] = train_and_evaluate(get_cascade_arch([10, 10]))
errors["Elman_15"] = train_and_evaluate(get_elman_arch([15]))
errors["Elman_3x5"] = train_and_evaluate(get_elman_arch([5, 5, 5]))
```

```
# Sonuçların Görselleştirilmesi
x_vals = np.linspace(0, 10, 100)
y_vals = np.linspace(0, 10, 100)
x_grid, y_grid = np.meshgrid(x_vals, y_vals)
input_data = np.column_stack((x_grid.flatten(), y_grid.flatten()))
x_train_reshaped = x_train[:, :, None]

plt.figure(figsize=(18, 6))

# Simüle Edilen Fonksiyon
plt.subplot(1, 3, 1)
plt.title('Simüle Edilen Fonksiyon')
plt.contourf(x_grid, y_grid, true_function(x_grid, y_grid), cmap='viridis', levels=20)
plt.colorbar()

# Elde Edilen Hata Değerlerinin Görselleştirilmesi
plt.subplot(1, 3, 2)
plt.title('Ortalama Bağıl Modelleme Hataları')
plt.bar(errors.keys(), errors.values())
plt.xticks(rotation=90)
plt.xlabel('Ağ Tipi ve Katman Sayısı')
plt.ylabel('MSE')
```



```
elman_model = tf.keras.Sequential(get_elman_arch([15]))  # Örneğin 15 nöronlu Elman mimarisi seçildi
elman_model.compile(optimizer='adam', loss='mean_squared_error')
elman_model.fit(x_train_reshaped, y_train, epochs=100, verbose=0)
elman_predictions = elman_model.predict(input_data).reshape(x_grid.shape)

plt.subplot(1, 3, 3)
plt.title('Elman Ağı Tahmini')
plt.contourf(x_grid, y_grid, elman_predictions, cmap='viridis', levels=20)
plt.colorbar()

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

```
2024-03-29 13:43:49.404135: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different c
omputation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-03-29 13:43:54.777116: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 503us/step
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 908us/step
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 866us/step
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 848us/step
PS C:\Users\win11>
```




```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 616us/step
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 863us/step
```