NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
“IGOR SIKORSKY KYIV POLYTECHNIC INSTITUTE”

Faculty of Informatics and Computer Engineering

Department of Computer Engineering

Lab 4 Report

Investigating Neural Network Structure and Simulating a Two-Variable Function

Variant 12

Student, group IM-14
(group code)

in the educational and professional program

“Software Engineering For Computer System”

Specialty 121 "Computer Engineering"

Mehmet KULUBECİOGLU

Reviewer Associate Professor, Dr.Ph. Pavlov Valerii
(position, academic degree, academic status, surname and initials)

Kyiv – 2023

The purpose of the work: To investigate the structure and principle of operation of a neural network. With the help of a neural network, simulate the function of two variables

12.	$y = \cos(x)/x - \sin(x)/x^2$ $z = \sin(x/2) + y \cdot \sin(x)$	12.	19.	7.

1 inner layer with 10 neurons;

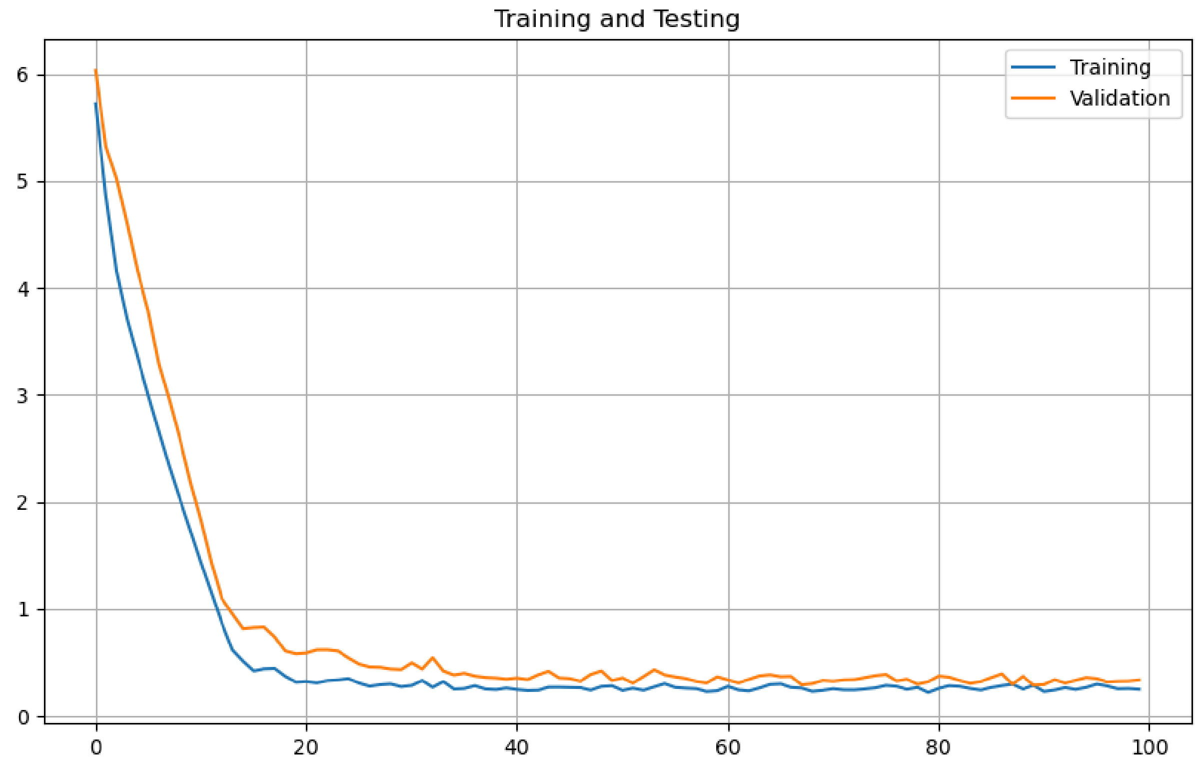
Program listing

```
In [7]: import numpy as np
        import tensorflow as tf
        import matplotlib.pyplot as plt

        # Function of Two Variables
        def func(x):
            y = np.cos(x[0]) / x[1] - np.sin(x[0]) / x[1]**2
            z = np.sin(x[0] / 2) + y * np.sin(x[0])
            return y, z

        # Collecting Dataset
        x = np.linspace(12, 19, 7)
        x0, x1 = np.meshgrid(x, x)
        x0, x1 = x0.ravel(), x1.ravel()
        X = [[x0[i], x1[i]] for i in range(len(x0))]
        y, z = zip(*[func(x) for x in X])

        split = int(len(X) * 0.8)
        x_train, y_train, z_train = X[:split], y[:split], z[:split]
        x_test, y_test, z_test = X[split:], y[split:], z[split:]

        # Function to display
        def visual(his):
            loss = his.history['loss']
            val_loss = his.history['val_loss']
            epochs = range(len(loss))
```

```python
    plt.figure(figsize=(10, 6))
    plt.plot(epochs, loss, label='Training')
    plt.plot(epochs, val_loss, label='Validation')
    plt.title('Training and Testing')
    plt.legend()
    plt.grid()
    plt.show()

# Model with 1 inner layer and 10 neurons
model_fl = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, input_shape=(2,), activation='relu'),
    tf.keras.layers.Dense(2)  # Two outputs for y and z
])

model_fl.summary()

model_fl.compile(
    optimizer=tf.keras.optimizers.SGD(
        learning_rate=tf.keras.optimizers.schedules.ExponentialDecay(0.001, decay_steps=75, decay_rate=0.96)
    ),
    loss='mae',
    metrics=['mae']
)

# Train the model
history_f = model_fl.fit(
    np.array(x_train), np.array([y_train, z_train]).T,  # Two outputs
    epochs=100,
    validation_data=(np.array(x_test), np.array([y_test, z_test]).T),
)

# Display learning curves
visual(history_f)
```

Performance results

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 10)                30

 dense_13 (Dense)            (None, 2)                 22

=================================================================
Total params: 52 (208.00 Byte)
Trainable params: 52 (208.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/100
2/2 [==============================] - 0s 146ms/step - loss: 5.7178 - mae: 5.7178 - val_loss: 6.0310 - val_mae: 6.0310
Epoch 2/100
2/2 [==============================] - 0s 31ms/step - loss: 4.8520 - mae: 4.8520 - val_loss: 5.3169 - val_mae: 5.3169
Epoch 3/100
2/2 [==============================] - 0s 31ms/step - loss: 4.1556 - mae: 4.1556 - val_loss: 5.0190 - val_mae: 5.0190
```



1 inner layer with 20 neurons;

Program listing

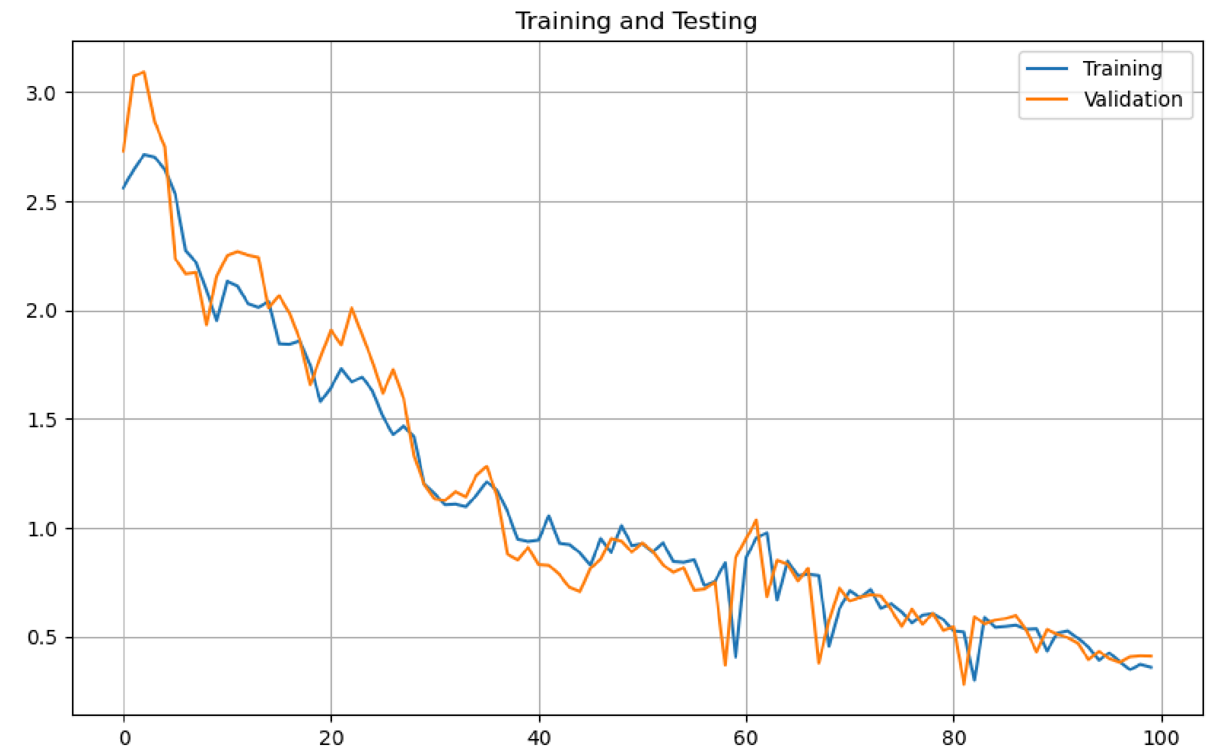
```
In [2]: import numpy as np
        import tensorflow as tf
        import matplotlib.pyplot as plt

        # Function of Two Variables
        def func(x):
            y = np.cos(x[0]) / x[1] - np.sin(x[0]) / x[1]**2
            z = np.sin(x[0] / 2) + y * np.sin(x[0])
            return y, z

        # Collecting Dataset
        x = np.linspace(12, 19, 7)
        x0, x1 = np.meshgrid(x, x)
        x0, x1 = x0.ravel(), x1.ravel()
        X = [[x0[i], x1[i]] for i in range(len(x0))]
        y, z = zip(*[func(x) for x in X])

        split = int(len(X) * 0.8)
        x_train, y_train, z_train = X[:split], y[:split], z[:split]
        x_test, y_test, z_test = X[split:], y[split:], z[split:]

        # Function to display
        def visual(his):
            loss = his.history['loss']
            val_loss = his.history['val_loss']
            epochs = range(len(loss))
```

Program listing

```python
    plt.figure(figsize=(10, 6))
    plt.plot(epochs, loss, label='Training')
    plt.plot(epochs, val_loss, label='Validation')
    plt.title('Training and Testing')
    plt.legend()
    plt.grid()
    plt.show()

# Model with 1 inner layer and 20 neurons
model_fl2 = tf.keras.models.Sequential([
    tf.keras.layers.Dense(20, input_shape=(2,), activation='relu'),
    tf.keras.layers.Dense(1)
])

model_fl2.summary()

model_fl2.compile(
    optimizer=tf.keras.optimizers.SGD(
        learning_rate=tf.keras.optimizers.schedules.ExponentialDecay(0.005, decay_steps=75, decay_rate=0.96)
    ),
    loss='mae',
    metrics=['mae']
)

# Train the model
history_f2 = model_fl2.fit(
    np.array(x_train), np.array(y_train),  # Assuming y_train is a single array
    epochs=100,
    validation_data=(np.array(x_test), np.array(y_test)),  # Assuming y_test is a single array
)

# Display learning curves
visual(history_f2)
```

Performance results

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 20)                60

 dense_3 (Dense)             (None, 1)                 21

=================================================================
Total params: 81 (324.00 Byte)
Trainable params: 81 (324.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/100
2/2 [==============================] - 0s 131ms/step - loss: 2.5616 - mae: 2.5616 - val_loss: 2.7308 - val_mae: 2.7308
Epoch 2/100
2/2 [==============================] - 0s 30ms/step - loss: 2.6446 - mae: 2.6446 - val_loss: 3.0731 - val_mae: 3.0731
Epoch 3/100
2/2 [==============================] - 0s 32ms/step - loss: 2.7143 - mae: 2.7143 - val_loss: 3.0941 - val_mae: 3.0941
```



Network type: cascade-forward backprop:

1 inner layer with 20 neurons;

Program listing

In [16]:

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

# Function of Two Variables
def func(x):
    y = np.cos(x[0]) / x[1] - np.sin(x[0]) / x[1]**2
    z = np.sin(x[0] / 2) + y * np.sin(x[0])
    return y, z

# Collecting Dataset
x = np.linspace(12, 19, 7)
x0, x1 = np.meshgrid(x, x)
x0, x1 = x0.ravel(), x1.ravel()
X = [[x0[i], x1[i]] for i in range(len(x0))]
y, z = zip(*[func(x) for x in X])

split = int(len(X) * 0.8)
x_train, y_train, z_train = X[:split], y[:split], z[:split]
x_test, y_test, z_test = X[split:], y[split:], z[split:]

# Function to display
def visual(his):
    loss = his.history['loss']
    val_loss = his.history['val_loss']
    epochs = range(len(loss))

    plt.figure(figsize=(10, 6))
    plt.plot(epochs, loss, label='Training')
    plt.plot(epochs, val_loss, label='Validation')
```


```python
    plt.figure(figsize=(10, 6))
    plt.plot(epochs, loss, label='Training')
    plt.plot(epochs, val_loss, label='Validation')
```


```python
    plt.title('Training and Testing')
    plt.legend()
    plt.grid()
    plt.show()

# Model with 1 inner layer and 20 neurons
inputs = tf.keras.Input(shape=(2,))
x = tf.keras.layers.Dense(20, activation='relu')(inputs)
outputs = tf.keras.layers.Dense(2)(tf.keras.layers.concatenate([inputs, x]))  # Two outputs for y and z
model_cf1 = tf.keras.Model(inputs, outputs)

model_cf1.summary()

model_cf1.compile(
    optimizer=tf.keras.optimizers.SGD(
        learning_rate=tf.keras.optimizers.schedules.ExponentialDecay(0.001, decay_steps=75, decay_rate=0.96)
    ),
    loss='mae',
    metrics=['mae']
)

# Train the model
history_cf1 = model_cf1.fit(
    np.array(x_train), np.array([y_train, z_train]).T,  # Two outputs
    epochs=100,
    validation_data=(np.array(x_test), np.array([y_test, z_test]).T),
)

# Display learning curves
visual(history_cf1)
```

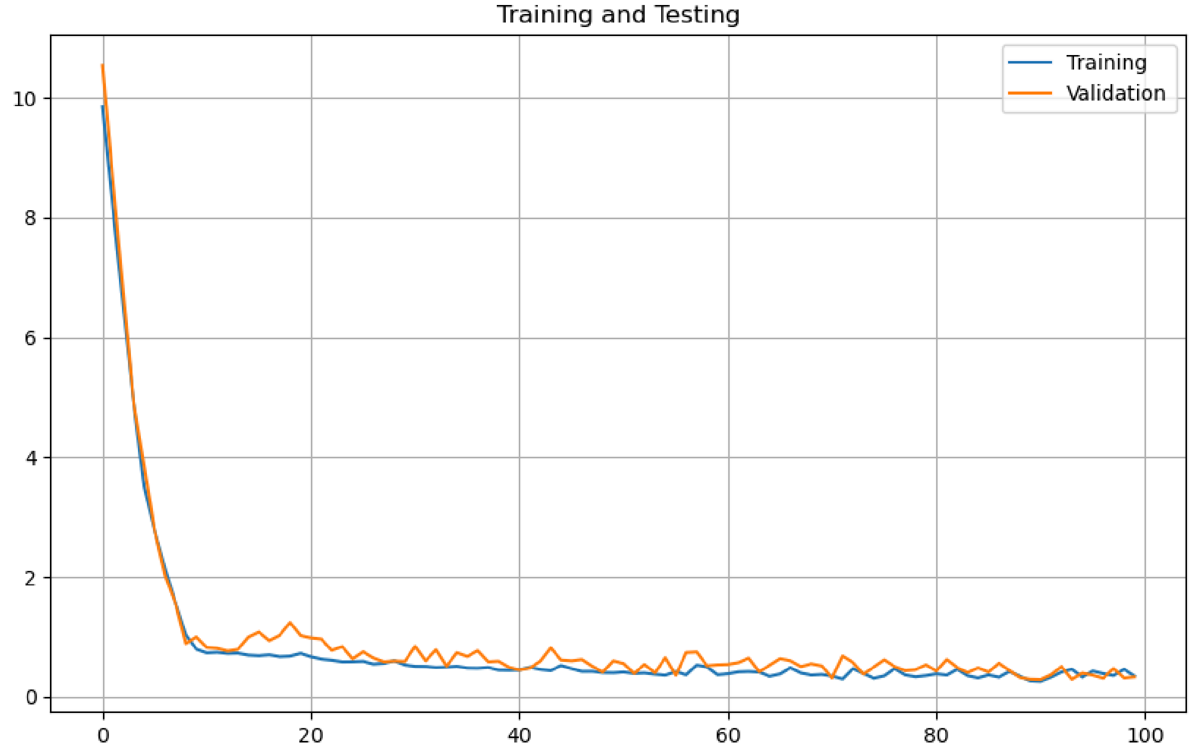
Creating a model

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 2)]          0           []

 dense_6 (Dense)                (None, 20)           60          ['input_2[0][0]']

 concatenate_1 (Concatenate     (None, 22)           0           ['input_2[0][0]',
 )                                                                'dense_6[0][0]']

 dense_7 (Dense)                (None, 2)            46          ['concatenate_1[0][0]']

==================================================================================================
Total params: 106 (424.00 Byte)
Trainable params: 106 (424.00 Byte)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
Epoch 1/100
```

The results of execution

Model training:

```
__________________________________________________________________________________________________
Epoch 1/100
2/2 [==============================] - 1s 337ms/step - loss: 9.8506
- mae: 9.8506 - val_loss: 10.5331 - val_mae: 10.5331
Epoch 2/100
2/2 [==============================] - 0s 30ms/step - loss: 8.1475
- mae: 8.1475 - val_loss: 8.6281 - val_mae: 8.6281
Epoch 3/100
2/2 [==============================] - 0s 29ms/step - loss: 6.5130
- mae: 6.5130 - val_loss: 6.7371 - val_mae: 6.7371
Epoch 4/100
2/2 [==============================] - 0s 31ms/step - loss: 4.8978
- mae: 4.8978 - val_loss: 4.9008 - val_mae: 4.9008
Epoch 5/100
2/2 [==============================] - 0s 31ms/step - loss: 3.5046
- mae: 3.5046 - val_loss: 3.8428 - val_mae: 3.8428
Epoch 6/100
2/2 [==============================] - 0s 36ms/step - loss: 2.7711
```



Training and Testing
Training
Validation
0
2
4
6
8
10
0
20
40
60
80
100

2 inner layers of 10 neurons each:

Program listing

```
In [1]: import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

# Function of Two Variables
def func(x):
    y = np.cos(x[0]) / x[1] - np.sin(x[0]) / x[1]**2
    z = np.sin(x[0] / 2) + y * np.sin(x[0])
    return y, z

# Collecting Dataset
x = np.linspace(12, 19, 7)
x0, x1 = np.meshgrid(x, x)
x0, x1 = x0.ravel(), x1.ravel()
X = [[x0[i], x1[i]] for i in range(len(x0))]
y, z = zip(*[func(x) for x in X])

split = int(len(X) * 0.8)
x_train, y_train, z_train = X[:split], y[:split], z[:split]
x_test, y_test, z_test = X[split:], y[split:], z[split:]

# Function to display
def visual(his):
    loss = his.history['loss']
    val_loss = his.history['val_loss']
    epochs = range(len(loss))

    plt.figure(figsize=(10, 6))
    plt.plot(epochs, loss, label='Training')
    plt.plot(epochs, val_loss, label='Validation')
    plt.title('Training and Testing')
    plt.legend()
```


```
    plt.plot(epochs, val_loss, label='Validation')
    plt.title('Training and Testing')
    plt.legend()
```


```
    plt.grid()
    plt.show()

# Model with 2 inner layers of 10 neurons each
inputs = tf.keras.Input(shape=(2,))
x0 = tf.keras.layers.Dense(10, activation='relu')(inputs)
x1 = tf.keras.layers.Dense(10, activation='relu')(tf.keras.layers.concatenate([inputs, x0]))
outputs = tf.keras.layers.Dense(2)(tf.keras.layers.concatenate([inputs, x0, x1]))  # Two outputs for y and z
model_cf2 = tf.keras.Model(inputs, outputs)

model_cf2.summary()

model_cf2.compile(
    optimizer=tf.keras.optimizers.SGD(
        learning_rate=tf.keras.optimizers.schedules.ExponentialDecay(0.005, decay_steps=75, decay_rate=0.96)
    ),
    loss='mae',
    metrics=['mae']
)

# Train the model
history_cf2 = model_cf2.fit(
    np.array(x_train), np.array([y_train, z_train]).T,  # Two outputs
    epochs=100,
    validation_data=(np.array(x_test), np.array([y_test, z_test]).T),
)

# Display learning curves
visual(history_cf2)
```

The results of execution

Model training:

```
WARNING:tensorflow:From C:\Users\mehme\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softma
x_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

WARNING:tensorflow:From C:\Users\mehme\anaconda3\Lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_o
utside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 2)]                  0         []

 dense (Dense)               (None, 10)                   30        ['input_1[0][0]']

 concatenate (Concatenate)   (None, 12)                   0         ['input_1[0][0]',
                                                                     'dense[0][0]']

 dense_1 (Dense)             (None, 10)                   130       ['concatenate[0][0]']
```

Network type:elman backprop::

1 inner layer with 15 neurons;

Program listing

In [1]:
```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

# Function of Two Variables
def func(x):
    y = np.cos(x[0]) / x[1] - np.sin(x[0]) / x[1]**2
    z = np.sin(x[0] / 2) + y * np.sin(x[0])
    return y, z

# Collecting Dataset
x = np.linspace(12, 19, 7)
x0, x1 = np.meshgrid(x, x)
x0, x1 = x0.ravel(), x1.ravel()
X = [[x0[i], x1[i]] for i in range(len(x0))]
y, z = zip(*[func(x) for x in X])

split = int(len(X) * 0.8)
x_train, y_train, z_train = X[:split], y[:split], z[:split]
x_test, y_test, z_test = X[split:], y[split:], z[split:]

# Function to display
def visual(his):
    loss = his.history['loss']
    val_loss = his.history['val_loss']
    epochs = range(len(loss))

    plt.figure(figsize=(10, 6))
    plt.plot(epochs, loss, label='Training')
    plt.plot(epochs, val_loss, label='Validation')
    plt.title('Training and Testing')
    plt.legend()
```


```python
    plt.plot(epochs, val_loss, label='Validation')
    plt.title('Training and Testing')
    plt.legend()
```


```python
    plt.grid()
    plt.show()

# Model with 1 inner layer and 15 neurons for Elman backprop
model_elman1 = tf.keras.models.Sequential([
    tf.keras.layers.SimpleRNN(15, activation='relu', input_shape=(1, 2)),
    tf.keras.layers.Dense(2)  # Two outputs for y and z
])

model_elman1.summary()

x_train = np.reshape(x_train, (np.shape(x_train)[0], 1, np.shape(x_train)[1]))
x_test = np.reshape(x_test, (np.shape(x_test)[0], 1, np.shape(x_test)[1]))

model_elman1.compile(
    optimizer=tf.keras.optimizers.SGD(
        learning_rate=tf.keras.optimizers.schedules.ExponentialDecay(0.005, decay_steps=75, decay_rate=0.96)
    ),
    loss='mae',
    metrics=['mae']
)

# Train the model
history_elman1 = model_elman1.fit(
    x_train, np.array([y_train, z_train]).T,  # Two outputs
    epochs=100,
    validation_data=(x_test, np.array([y_test, z_test]).T),
)

# Display learning curves
```

Creating a model:

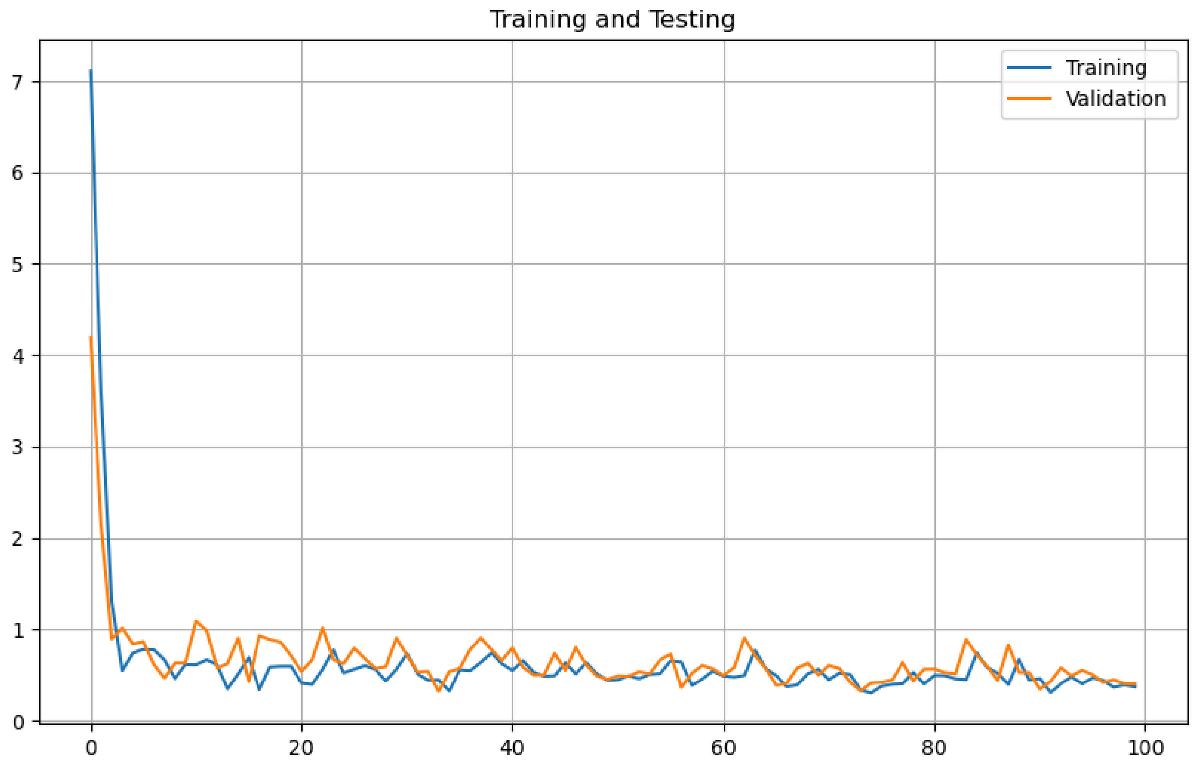
```
WARNING:tensorflow:From C:\Users\mehme\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softma
x_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

WARNING:tensorflow:From C:\Users\mehme\anaconda3\Lib\site-packages\keras\src\layers\rnn\simple_rnn.py:130: The name tf.execut
ing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 15)                270

 dense (Dense)               (None, 2)                 32

=================================================================
Total params: 302 (1.18 KB)
Trainable params: 302 (1.18 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The results of execution

Model training:

```
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/100
WARNING:tensorflow:From C:\Users\mehme\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.Ragged
TensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\mehme\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.exec
uting_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

2/2 [==============================] - 1s 243ms/step - loss: 7.1083 - mae: 7.1083 - val_loss: 4.1945 - val_mae: 4.1945
Epoch 2/100
2/2 [==============================] - 0s 32ms/step - loss: 3.6021 - mae: 3.6021 - val_loss: 2.1551 - val_mae: 2.1551
Epoch 3/100
2/2 [==============================] - 0s 30ms/step - loss: 1.3091 - mae: 1.3091 - val_loss: 0.8961 - val_mae: 0.8961
Epoch 4/100
2/2 [==============================] - 0s 31ms/step - loss: 0.5506 - mae: 0.5506 - val_loss: 1.0157 - val_mae: 1.0157
Epoch 5/100
2/2 [==============================] - 0s 30ms/step - loss: 0.7419 - mae: 0.7419 - val_loss: 0.8406 - val_mae: 0.8406
Epoch 6/100
2/2 [==============================] - 0s 31ms/step - loss: 0.7827 - mae: 0.7827 - val_loss: 0.8672 - val_mae: 0.8672
```

In [ ]:



3 inner layers with 5 neurons each;

Program listing

In [1]:

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

# Function of Two Variables
def func(x):
    y = np.cos(x[0]) / x[1] - np.sin(x[0]) / x[1]**2
    z = np.sin(x[0] / 2) + y * np.sin(x[0])
    return y, z

# Collecting Dataset
x = np.linspace(12, 19, 7)
x0, x1 = np.meshgrid(x, x)
x0, x1 = x0.ravel(), x1.ravel()
X = [[x0[i], x1[i]] for i in range(len(x0))]
y, z = zip(*[func(x) for x in X])

split = int(len(X) * 0.8)
x_train, y_train, z_train = X[:split], y[:split], z[:split]
x_test, y_test, z_test = X[split:], y[split:], z[split:]

# Function to display
def visual(his):
    loss = his.history['loss']
    val_loss = his.history['val_loss']
    epochs = range(len(loss))

    plt.figure(figsize=(10, 6))
    plt.plot(epochs, loss, label='Training')
    plt.plot(epochs, val_loss, label='Validation')
```

```
    plt.grid()
    plt.show()

# Model with 3 inner layers and 5 neurons each for Elman backprop
model_elman2 = tf.keras.models.Sequential([
    tf.keras.layers.SimpleRNN(5, activation='relu', input_shape=(1, 2)),
    tf.keras.layers.Dense(5, activation='relu'),
    tf.keras.layers.Dense(5, activation='relu'),
    tf.keras.layers.Dense(2)  # Two outputs for y and z
])

model_elman2.summary()

x_train = np.reshape(x_train, (np.shape(x_train)[0], 1, np.shape(x_train)[1]))
x_test = np.reshape(x_test, (np.shape(x_test)[0], 1, np.shape(x_test)[1]))

model_elman2.compile(
    optimizer=tf.keras.optimizers.SGD(
        learning_rate=tf.keras.optimizers.schedules.ExponentialDecay(0.001, decay_steps=75, decay_rate=0.96)
    ),
    loss='mae',
    metrics=['mae']
)

# Train the model
history_elman2 = model_elman2.fit(
    x_train, np.array([y_train, z_train]).T,  # Two outputs
    epochs=100,
    validation_data=(x_test, np.array([y_test, z_test]).T),
)

# Display learning curves
visual(history_elman2)
```

The results of execution

Model training:

```
uting_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

2/2 [==============================] - 1s 284ms/step - loss: 0.2806 - mae: 0.2806 - val_loss: 0.3443 - val_mae: 0.3443
Epoch 2/100
2/2 [==============================] - 0s 34ms/step - loss: 0.2651 - mae: 0.2651 - val_loss: 0.3247 - val_mae: 0.3247
Epoch 3/100
2/2 [==============================] - 0s 32ms/step - loss: 0.2551 - mae: 0.2551 - val_loss: 0.3086 - val_mae: 0.3086
Epoch 4/100
2/2 [==============================] - 0s 33ms/step - loss: 0.2484 - mae: 0.2484 - val_loss: 0.2974 - val_mae: 0.2974
Epoch 5/100
2/2 [==============================] - 0s 33ms/step - loss: 0.2464 - mae: 0.2464 - val_loss: 0.3051 - val_mae: 0.3051
Epoch 6/100
2/2 [==============================] - 0s 32ms/step - loss: 0.2453 - mae: 0.2453 - val_loss: 0.3026 - val_mae: 0.3026
Epoch 7/100
2/2 [==============================] - 0s 32ms/step - loss: 0.2429 - mae: 0.2429 - val_loss: 0.2983 - val_mae: 0.2983
Epoch 8/100
2/2 [==============================] - 0s 31ms/step - loss: 0.2404 - mae: 0.2404 - val_loss: 0.2838 - val_mae: 0.2838
Epoch 9/100
2/2 [==============================] - 0s 30ms/step - loss: 0.2383 - mae: 0.2383 - val_loss: 0.2950 - val_mae: 0.2950
```