Simplest SLP

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define the model
model = Sequential([
    Dense(1, input_shape=(1,), activation='sigmoid')  # Single neuron
with sigmoid activation
])

# Compile the model
model.compile(optimizer='sgd', loss='binary_crossentropy',
              metrics=['accuracy'])

np.random.seed(0)
X = np.random.rand(100).reshape(-1, 1)  # Shape for input layer
y = np.array([1 if x >= 0.5 else 0 for x in X])  # Label: <0.5 => 0,
>=0.5 => 1

# Train the model without epoch output
model.fit(X, y, epochs=10, verbose=0)

# Function to get user input and classify it
def user_input_classification():
    user_input = float(input("Enter a number between 0 and 1: "))  #
Expecting valid float input
    if 0 <= user_input <= 1:
        prediction = model.predict(np.array([user_input]))  # Predict
with the trained model
        classification = 1 if prediction > 0.5 else 0
        print(f"Perceptron classification: {classification}")
    else:
        print("Please enter a number between 0 and 1.")

# Classify user input
user_input_classification()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/
dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
Enter a number between 0 and 1: 0.6
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
Perceptron classification: 1
```

Code with two input neurons

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define the model with two input neurons
model = Sequential([
    Dense(1, input_shape=(2,), activation='sigmoid')
])

# Compile the model
model.compile(optimizer='sgd', loss='binary_crossentropy',
metrics=['accuracy'])

# Generate random training data with two features
np.random.seed(0)
X = np.random.rand(100, 2)  # Each row now has two values
y = np.array([1 if x[0] >= 0.5 and  x[1] >= 0.5 else 0 for x in X])  #
Label based on sum of features

# Train the model with more epochs for accuracy
model.fit(X, y, epochs=500, verbose=0)  # Increased epochs

# Function to get user input and classify it
def user_input_classification():
    # Collect two user inputs
    user_input1 = float(input("Enter first number between 0 and 1: "))
    user_input2 = float(input("Enter second number between 0 and 1:
"))

    # Clamp each input to the range [0, 1]
    #user_input1 = max(0, min(user_input1, 1))
    #user_input2 = max(0, min(user_input2, 1))

    # Reshape inputs for prediction
    user_inputs = np.array([[user_input1, user_input2]])

    # Predict and classify
    prediction = model.predict(user_inputs)
    classification = int(prediction[0][0] >= 0.5)  # Threshold at 0.5
    print(f"Perceptron classification: {classification}")

# Classify user input
user_input_classification()
```

```
Enter first number between 0 and 1: 0.7
Enter second number between 0 and 1: 0.3
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
Perceptron classification: 0
```

Code with additional hidden layer and two input neurons

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define the model with a hidden layer of two neurons and an output
layer
model = Sequential([
    Dense(2, input_shape=(2,), activation='sigmoid'),   # Hidden layer
with 2 neurons and ReLU activation
    Dense(1, activation='sigmoid')                    # Output layer
with 1 neuron and sigmoid activation
])

# Compile the model
model.compile(optimizer='sgd', loss='binary_crossentropy',
metrics=['accuracy'])

# Generate random training data with two features
np.random.seed(0)
X = np.random.rand(100, 2)  # Each row now has two values
y = np.array([1 if x[0] + x[1] >= 1 else 0 for x in X])  # Label based
on sum of features

# Train the model with more epochs for accuracy
model.fit(X, y, epochs=500, verbose=0)  # Increased epochs

# Function to get user input and classify it
def user_input_classification():
    # Collect two user inputs
    user_input1 = float(input("Enter first number between 0 and 1: "))
    user_input2 = float(input("Enter second number between 0 and 1:
"))

    # Clamp each input to the range [0, 1]
    user_input1 = max(0, min(user_input1, 1))
    user_input2 = max(0, min(user_input2, 1))

    # Reshape inputs for prediction
    user_inputs = np.array([[user_input1, user_input2]])

    # Predict and classify
    prediction = model.predict(user_inputs)
    classification = int(prediction[0][0] >= 0.5)  # Threshold at 0.5
    print(f"Perceptron classification: {classification}")

# Classify user input
user_input_classification()
```

```
Enter first number between 0 and 1: 0.2
Enter second number between 0 and 1: 0.4
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
Perceptron classification: 1
```

Linear Regression

```
# Import necessary libraries
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import tensorflow as tf

# Sample data: X (input), Y (output) for training
# Example: Let's assume we're predicting the output of a linear
equation y = 2x + 1
X = np.array([1, 2, 3, 4, 5], dtype=float)
Y = np.array([3, 5, 7, 9, 11], dtype=float)

# Build the ANN model
model = Sequential([
    Dense(units=1, input_shape=(1,), activation='linear')
    ])

# Compile the model with SGD optimizer
model.compile(optimizer='sgd', loss='mean_squared_error')

# Train the model without displaying the epochs
model.fit(X, Y, epochs=1000, verbose=0)

# Predict the output for a user input
user_input = float(input("Enter a value for prediction: "))
prediction = model.predict(np.array([[user_input]]))

# Display the prediction
print(f"Predicted output for input {user_input}: {prediction[0][0]}")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/
dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
Enter a value for prediction: 6
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
Predicted output for input 6.0: 13.009330749511719
```