**Program 3: Program to build a neural network with a single hidden layer using TensorFlow.**

**Aim:** The aim of this program to build a neural network with a single hidden layer using TensorFlow.

**Procedure:-**

```
import tensorflow as tf
import numpy as np

# Step 1: Prepare the Data
# Generate some example data
x_train = np.array([[1], [2], [3], [4], [5]], dtype=np.float32)  # Input features
y_train = np.array([[2], [4], [6], [8], [10]], dtype=np.float32)  # Target outputs (y = 2x)

# Step 2: Build the Model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation='relu', input_shape=(1,)),  # Single hidden layer with 10
neurons
    tf.keras.layers.Dense(1)  # Output layer with 1 neuron for regression
])

# Step 3: Compile the Model
model.compile(optimizer='adam', loss='mean_squared_error')

# Step 4: Train the Model
print("Training the model...")
model.fit(x_train, y_train, epochs=100, verbose=0)  # Train for 100 epochs
print("Model training complete.")

# Step 5: Test the Model
x_test = np.array([[6]], dtype=np.float32)  # Test input
predicted_y = model.predict(x_test)
print(f"Predicted y for x={x_test[0][0]}: {predicted_y[0][0]}")
```

**output:-**

```
Training the model...
 Model training complete.
1/1 [==============================] - 0s 22ms/step
Predicted y for x=6.0: 12.0
```

**Program 4- Build 3 networks, each with atleast 10 hidden layers in deep learning.**

**Aim:-**The aim of this program to build 3 networks, each with atleast 10 hidden layers in deep learning

**Procedure:-**

```
import tensorflow as tf
import numpy as np

# Step 1: Define helper function to build a model
def build_deep_network(input_shape, num_hidden_layers=10, units_per_layer=32,
activation='relu', output_units=1, output_activation=None):
    model = tf.keras.Sequential()
    # Input Layer
    model.add(tf.keras.layers.Dense(units_per_layer, activation=activation,
input_shape=input_shape))
    # Hidden Layers
    for _ in range(num_hidden_layers - 1):
        model.add(tf.keras.layers.Dense(units_per_layer, activation=activation))
    # Output Layer
    model.add(tf.keras.layers.Dense(output_units, activation=output_activation))
    return model

# Step 2: Generate synthetic data for demonstration
x_train = np.random.rand(1000, 10)  # 1000 samples, 10 features
y_train_reg = np.sum(x_train, axis=1)  # Regression target: sum of inputs
y_train_binary = (y_train_reg > 5).astype(int)  # Binary classification target: sum > 5
y_train_multiclass = np.random.randint(0, 3, size=(1000,))  # Multiclass classification: 3 classes

# Step 3: Build and train three models
# Model 1: Regression
print("Training Model 1: Regression")
model_regression = build_deep_network(input_shape=(10,), output_units=1,
output_activation=None)
model_regression.compile(optimizer='adam', loss='mean_squared_error', metrics=['mse'])
model_regression.fit(x_train, y_train_reg, epochs=10, batch_size=32, verbose=1)

# Model 2: Binary Classification
print("\nTraining Model 2: Binary Classification")
model_binary = build_deep_network(input_shape=(10,), output_units=1,
output_activation='sigmoid')
model_binary.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model_binary.fit(x_train, y_train_binary, epochs=10, batch_size=32, verbose=1)

# Model 3: Multiclass Classification
```

```
print("\nTraining Model 3: Multiclass Classification")
model_multiclass = build_deep_network(input_shape=(10,), output_units=3,
output_activation='softmax')
model_multiclass.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model_multiclass.fit(x_train, y_train_multiclass, epochs=10, batch_size=32, verbose=1)

# Step 4: Model Summary
print("\nModel Summaries:")
print("\nModel 1 Summary:")
model_regression.summary()
print("\nModel 2 Summary:")
model_binary.summary()
print("\nModel 3 Summary:")
model_multiclass.summary()
```