ex16

September 30, 2024

[1]:
```
import numpy as np
import tensorflow as tf
```

[3]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(2, input_shape=(2,), activation='sigmoid',␣
 ↪kernel_initializer=tf.keras.initializers.Constant(0.1)),
    tf.keras.layers.Dense(1, activation='sigmoid', kernel_initializer=tf.keras.
 ↪initializers.Constant(0.1))
])

model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=1.0),␣
 ↪loss='mean_squared_error')
```

[4]:
```
# Sample input and output (XOR problem as an example)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)  # Inputs
y = np.array([[0], [1], [1], [0]], dtype=np.float32)  # Outputs
```

[5]:
```
model.fit(X, y, epochs=10, verbose=0)
```

[5]:
```
<keras.src.callbacks.history.History at 0x2bcd563ce90>
```

[6]:
```
predictions = model.predict(X)
```

```
1/1             0s 84ms/step
1/1             0s 84ms/step
```

[7]:
```
for i in range(len(X)):
    print(f"Input: {X[i]} -> Predicted Output: {predictions[i]}")
```

```
Input: [0. 0.] -> Predicted Output: [0.5021983]
Input: [0. 1.] -> Predicted Output: [0.5030506]
Input: [1. 0.] -> Predicted Output: [0.5030506]
Input: [1. 1.] -> Predicted Output: [0.50389886]
```

[10]:
```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
def sigmoid_derivative(x):
    return x * (1 - x)

input_layer_size = 2
hidden_layer_size = 2
output_layer_size = 1
learning_rate = 1

weights_input_hidden = np.full((input_layer_size, hidden_layer_size), 0.1)
weights_hidden_output = np.full((hidden_layer_size, output_layer_size), 0.1)

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

epochs = 10000
for epoch in range(epochs):
    for i in range(len(X)):
        input_layer = X[i].reshape(1, -1)
        hidden_layer_input = np.dot(input_layer, weights_input_hidden)
        hidden_layer_output = sigmoid(hidden_layer_input)

        output_layer_input = np.dot(hidden_layer_output, weights_hidden_output)
        predicted_output = sigmoid(output_layer_input)

        error = y[i] - predicted_output

        d_predicted_output = error * sigmoid_derivative(predicted_output)

        error_hidden_layer = d_predicted_output.dot(weights_hidden_output.T)
        d_hidden_layer_output = error_hidden_layer *␣
 ↪sigmoid_derivative(hidden_layer_output)

        weights_hidden_output += hidden_layer_output.T.dot(d_predicted_output)␣
 ↪* learning_rate
        weights_input_hidden += input_layer.T.dot(d_hidden_layer_output) *␣
 ↪learning_rate

print("Final weights from input to hidden layer:\n", weights_input_hidden)
print("Final weights from hidden to output layer:\n", weights_hidden_output)

for i in range(len(X)):
    input_layer = X[i].reshape(1, -1)
    hidden_layer_output = sigmoid(np.dot(input_layer, weights_input_hidden))
    predicted_output = sigmoid(np.dot(hidden_layer_output,␣
 ↪weights_hidden_output))
    print(f"Input: {X[i]} -> Predicted Output: {predicted_output}")
```

```
Final weights from input to hidden layer:
 [[-8.33281602 -8.33281602]
 [-8.33334893 -8.33334893]]
Final weights from hidden to output layer:
 [[-4.14170511]
 [-4.14170511]]
Input: [0 0] -> Predicted Output: [[0.015647]]
Input: [0 1] -> Predicted Output: [[0.49950236]]
Input: [1 0] -> Predicted Output: [[0.49950209]]
Input: [1 1] -> Predicted Output: [[0.49999988]]
```