ID : 4221386

NAME : Moaz Awad Ali

https://github.com/AmazingMoaaz/AI323-Computational-Neuroscience

$\boxed{W_3}$ W_3 old $- \eta * \frac{\delta t}{w_3}$

$= (0.7419 \times 0.1868 \times 0.45 + -0.2171 \times 0.1755 \times 0.55)$

$0.0623 + -0.0209$

$= 0.0414 \times 0.2406 \times 0.05$

$= 0.0004$

$W_3 = 0.25 - 0.5 * 0.0004$

$= 0.2498$

W_4 $W_4 = old - \eta * \frac{\delta t}{w_4}$

$(0.7414 \times 0.1868 \times 0.45 + -0.217 \times 0.1755 \times 0.50$

$= 0.0623 - 0.0238)$

$= 0.0385 \times 0.2406 \times 0.1$

$= 0.0009$

$W_4 = 0.3 - 0.5 \times 0.0009$

$= 0.29955$

$h1_n = 0.3775$ | $h2_n = 0.3925$

$h1_o = 0.5933$ | $h2_o = 0.5969$

$o1_o = 0.7514$ | $o1_n = 1.104$

$o2_n = 1.2249$ | $o2_o = 0.7729$

$$\boxed{W6} = W6_{old} - \eta \times \frac{Et}{W6}$$

$$\frac{Et}{W6} = \frac{Et}{out_{o1}} \times \frac{out_{o1}}{net_{o1}} \times \frac{net_{o1}}{W6}$$

$$-(0.01 - 0.7514) \times 0.7514(1 - 0.7514) \times h2\,out$$

$$= 0.7414 \times 0.1868 \times 0.5969$$

$$= \frac{Et}{W6} = 0.0826$$

$$W6 = 0.45 - 0.5 \times 0.0826$$

$$W6 = 0.4086$$

$$\boxed{W7} : \frac{Et}{W7} = \frac{Et}{out_2} \times \frac{out_2}{net_2} \times \frac{net_{o2}}{W7}$$

$$-(0.99 - 0.7729) \times 0.7729(1 - 0.7729) \times 0.5933$$

$$- 0.2171 \times 0.1755 \times 0.5933$$

$$\frac{Et}{W7} = -0.0226$$

$$W7 = 0.50 - 0.50 \times -0.0226$$

$$= 0.5113$$

/ /

$\boxed{W8}$ $\frac{Et}{w8} = \frac{Et}{out_{o2}} \times \frac{out_{o2}}{net_{o2}} \times \frac{net_{o2}}{w8}$

$-0.2171 \times 0.1755 \times 0.5986$

$= -0.0228$

$W8 = 0.55 - 0.5 \times 0.0228$

$= 0.5614$

$\boxed{W2}$ $W2 = W_2 old - \eta \times \frac{Et}{W_2}$

$\frac{Et}{W_2} = \Big(\frac{E_{o1}}{out_{o1}} \times \frac{out_{o1}}{net_{o1}} \times \frac{net_{o1}}{out_{h1}}$

$+ \frac{E_{o2}}{out_{o2}} \times \frac{out_{o2}}{net_{o2}} \times \frac{net_{o2}}{out_{h1}} \Big)$

$\times \frac{out_{h1}}{net_{h1}} \times \frac{net_{h1}}{w_1}$

$= \big(-(0.01 - 0,7514) \times 0.7514 (1 - 0.7514)$

$\times 0.404 - (0.99 - 0,7729)$

$\times 0.7729 (1 - 0,7729) \times 0.5 \big)$

$\times 0.0553 + (-0.0190)$

$= 0.0363 \times 0.2413 \times 0.1 = 0.0008$

$W_2 = 0.2 - 0.5 \times 0.0008$

$= 0.1996$

Backpropagation.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

[1]
```
print("https://github.com/AmazingMoaaz/AI323-Computational-Neuroscience")
```

https://github.com/AmazingMoaaz/AI323-Computational-Neuroscience

[2]
```
import random
```

```
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        # Initialize weights and biases
        self.hidden_weights = [[random.uniform(-0.5, 0.5) for _ in range(input_size)] for _ in range(hidden_size)]
        self.output_weights = [[random.uniform(-0.5, 0.5) for _ in range(hidden_size)] for _ in range(output_size)]
        self.hidden_biases = [0.5 for _ in range(hidden_size)]
        self.output_biases = [0.7 for _ in range(output_size)]
        self.learning_rate = 0.1

    @staticmethod
    def tanh(x):
        x = max(min(x, 10), -10)
        exp_2x = 2.718281828459045235360287471352 7 ** (2 * x)
        return (exp_2x - 1) / (exp_2x + 1)

    @staticmethod
    def tanh_derivative(tanh_output):
        return 1.0 - tanh_output ** 2

    def forward(self, inputs):
        # Store inputs for backpropagation
        self.inputs = inputs

        # Calculate hidden layer outputs
        self.hidden_inputs = []
        self.hidden_outputs = []
        for i, weights in enumerate(self.hidden_weights):
            hidden_input = sum(w * inp for w, inp in zip(weights, inputs)) + self.hidden_biases[i]
            self.hidden_inputs.append(hidden_input)
            self.hidden_outputs.append(self.tanh(hidden_input))
```

Backpropagation.ipynb

File Edit View Insert Runtime Tools Help

Share

+ Code + Text

RAM
Disk

```python
        # Calculate output layer outputs
        self.output_inputs = []
        self.outputs = []
        for i, weights in enumerate(self.output_weights):
            output_input = sum(w * h for w, h in zip(weights, self.hidden_outputs)) + self.output_biases[i]
            self.output_inputs.append(output_input)
            self.outputs.append(self.tanh(output_input))

        return self.outputs

    def backward(self, targets):
        # Calculate output deltas
        output_deltas = []
        for i, output in enumerate(self.outputs):
            error = targets[i] - output
            output_deltas.append(error * self.tanh_derivative(output))

        # Calculate hidden deltas
        hidden_deltas = []
        for i in range(len(self.hidden_outputs)):
            error = sum(delta * self.output_weights[j][i] for j, delta in enumerate(output_deltas))
            hidden_deltas.append(error * self.tanh_derivative(self.hidden_outputs[i]))

        # Update output weights and biases
        for i, delta in enumerate(output_deltas):
            for j in range(len(self.output_weights[i])):
                self.output_weights[i][j] += self.learning_rate * delta * self.hidden_outputs[j]
            self.output_biases[i] += self.learning_rate * delta

        # Update hidden weights and biases
        for i, delta in enumerate(hidden_deltas):
            for j in range(len(self.hidden_weights[i])):
                self.hidden_weights[i][j] += self.learning_rate * delta * self.inputs[j]
            self.hidden_biases[i] += self.learning_rate * delta

    def calculate_error(self, outputs, targets):
        return sum(0.5 * (t - o) ** 2 for t, o in zip(targets, outputs))

    def train(self, inputs, targets, epochs):
        for epoch in range(epochs):
            outputs = self.forward(inputs)
```

Backpropagation.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

```python
            if epoch % 100 == 0:
                error = self.calculate_error(outputs, targets)
                print(f"Epoch {epoch}: Total error = {error:.4f}")

        final_outputs = self.forward(inputs)
        final_error = self.calculate_error(final_outputs, targets)
        return final_outputs, final_error

    def get_network_state(self):
        return {
            'inputs': self.inputs,
            'hidden': {
                'inputs': self.hidden_inputs,
                'outputs': self.hidden_outputs
            },
            'outputs': {
                'inputs': self.output_inputs,
                'outputs': self.outputs
            }
        }
```

[4]

```python
if __name__ == "__main__":
    # Initialize network
    network = NeuralNetwork(input_size=2, hidden_size=2, output_size=2)

    # Define inputs and targets
    inputs = [0.05, 0.10]
    targets = [0.01, 0.99]
```

[5]

```python
    print(f"{'':=^50}")
    print("Initial state before training:")
    initial_outputs = network.forward(inputs)
    initial_error = network.calculate_error(initial_outputs, targets)

    state = network.get_network_state()
    print(f"{'':=^50}")
    print(f"  Hidden Layer Outputs: h1 = {state['hidden']['outputs'][0]:.4f}, h2 = {state['hidden']['outputs'][1]:.4f}")
    print(f"  Output Layer Inputs: o1 = {state['outputs']['inputs'][0]:.4f}, o2 = {state['outputs']['inputs'][1]:.4f}")
    print(f"  Output Layer Outputs: o1 = {state['outputs']['outputs'][0]:.4f}, o2 = {state['outputs']['outputs'][1]:.4f}")
    print(f"{'':=^50}")
```

Backpropagation.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

[5]
```
print(f"  Error for o1: {0.5 * (targets[0] - initial_outputs[0])**2:.4f}")
print(f"  Error for o2: {0.5 * (targets[1] - initial_outputs[1])**2:.4f}")
print(f"\nTotal Error: {initial_error:.4f}")
print(f"{'':=^50}")
```

```
==================================================
Initial state before training:
==================================================
  Hidden Layer Outputs: h1 = 0.4480, h2 = 0.4954
  Output Layer Inputs: o1 = 0.5859, o2 = 0.8698
  Output Layer Outputs: o1 = 0.5269, o2 = 0.7013
==================================================
  Error for o1: 0.1336
  Error for o2: 0.0417

Total Error: 0.1753
==================================================
```

```
# Train the network
print("\nTraining network for 1000 epochs...\n")
final_outputs, final_error = network.train(inputs, targets, 1000)
```

```
Training network for 1000 epochs...

Epoch 0: Total error = 0.1753
Epoch 100: Total error = 0.0032
Epoch 200: Total error = 0.0015
Epoch 300: Total error = 0.0009
Epoch 400: Total error = 0.0006
Epoch 500: Total error = 0.0005
Epoch 600: Total error = 0.0004
Epoch 700: Total error = 0.0003
Epoch 800: Total error = 0.0003
Epoch 900: Total error = 0.0002
```

[7]
```
# Display results after training
print(f"{'':=^50}")
print("Results after training:")
state = network.get_network_state()
print(f"{'':=^50}")
print(f"  Hidden Layer Outputs: h1 = {state['hidden']['outputs'][0]:.4f}, h2 = {state['hidden']['outputs'][1]:.4f}")
```

Backpropagation.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

```
Epoch 500: Total error = 0.0005
Epoch 600: Total error = 0.0004
Epoch 700: Total error = 0.0003
Epoch 800: Total error = 0.0003
Epoch 900: Total error = 0.0002
```

```python
# Display results after training
print(f"{'':=^50}")
print("Results after training:")
state = network.get_network_state()
print(f"{'':=^50}")
print(f"  Hidden Layer Outputs: h1 = {state['hidden']['outputs'][0]:.4f}, h2 = {state['hidden']['outputs'][1]:.4f}")
print(f"  Output Layer Inputs: o1 = {state['outputs']['inputs'][0]:.4f}, o2 = {state['outputs']['inputs'][1]:.4f}")
print(f"  Output Layer Outputs: o1 = {state['outputs']['outputs'][0]:.4f}, o2 = {state['outputs']['outputs'][1]:.4f}")
print(f"{'':=^50}")
print(f"  Error for o1: {0.5 * (targets[0] - final_outputs[0])**2:.4f}")
print(f"  Error for o2: {0.5 * (targets[1] - final_outputs[1])**2:.4f}")
print(f"\nTotal Error: {final_error:.4f}")
print(f"{'':=^50}")
```

```
==================================================
Results after training:
==================================================
  Hidden Layer Outputs: h1 = 0.5719, h2 = 0.6676
  Output Layer Inputs: o1 = 0.0099, o2 = 2.0963
  Output Layer Outputs: o1 = 0.0099, o2 = 0.9702
==================================================
  Error for o1: 0.0000
  Error for o2: 0.0002

Total Error: 0.0002
==================================================
```