neural_networks

October 24, 2024

[1]:

```python
import numpy as np

class BAM:
    def __init__(self, input_size, output_size):
        self.weights = np.zeros((input_size, output_size))

    def train(self, input_data, output_data):
        for i, o in zip(input_data, output_data):
            self.weights += np.outer(i, o)

    def recall(self, input_pattern, direction='forward'):
        if direction == 'forward':
            return np.sign(input_pattern @ self.weights)
        elif direction == 'backward':
            return np.sign(input_pattern @ self.weights.T)

# Example usage
input_data = np.array([[1, -1], [-1, 1]])
output_data = np.array([[1, -1], [-1, 1]])

bam = BAM(input_size=2, output_size=2)
bam.train(input_data, output_data)

input_pattern = np.array([1, -1])
print("Recalled Output:", bam.recall(input_pattern))  # Forward recall
```

```
Recalled Output: [ 1. -1.]
```

[2]:

```python
import numpy as np

class KSOM:
    def __init__(self, input_dim, map_dim, learning_rate=0.5):
        self.weights = np.random.rand(map_dim[0], map_dim[1], input_dim)
        self.lr = learning_rate

    def train(self, data, epochs=100):
        for epoch in range(epochs):
            for sample in data:
                # Find the Best Matching Unit (BMU)
                bmu_idx = self.find_bmu(sample)
                # Update the BMU and its neighbors
                self.update_weights(sample, bmu_idx)

    def find_bmu(self, sample):
        distances = np.linalg.norm(self.weights - sample, axis=2)
        return np.unravel_index(np.argmin(distances), distances.shape)

    def update_weights(self, sample, bmu_idx):
        self.weights[bmu_idx] += self.lr * (sample - self.weights[bmu_idx])

# Example usage
data = np.array([[0.2, 0.8], [0.5, 0.3], [0.9, 0.6]])
ksom = KSOM(input_dim=2, map_dim=(3, 3))
ksom.train(data, epochs=100)
```

[3]:

```
import numpy as np

class MaxNET:
    def __init__(self, size, epsilon=0.1):
        self.epsilon = epsilon
        self.weights = np.eye(size) - np.ones((size, size)) * epsilon

    def activate(self, inputs):
        inputs = np.copy(inputs)
        while np.sum(inputs > 0) > 1:
            inputs = np.dot(self.weights, inputs)
            inputs = np.where(inputs < 0, 0, inputs)  # No negative activations
        return inputs

# Example usage
inputs = np.array([0.2, 0.8, 0.5])
maxnet = MaxNET(size=len(inputs))
output = maxnet.activate(inputs)
print("Winning neuron:", np.argmax(output))
```

```
Winning neuron: 1
```

[4]:

```
import numpy as np

class RBFNetwork:
    def __init__(self, num_centers, input_dim):
        self.num_centers = num_centers
        self.centers = np.random.uniform(-1, 1, (num_centers, input_dim))
        self.weights = np.random.randn(num_centers)

    def rbf(self, x, c, sigma=1.0):
        return np.exp(-np.linalg.norm(x-c)**2 / (2 * sigma**2))

    def basis_function_output(self, X):
        return np.array([[self.rbf(x, c) for c in self.centers] for x in X])

    def train(self, X, y):
        G = self.basis_function_output(X)
        self.weights = np.linalg.pinv(G).dot(y)

    def predict(self, X):
        G = self.basis_function_output(X)
        return G.dot(self.weights)

# Example usage
X = np.array([[0.1, 0.2], [0.4, 0.6], [0.7, 0.9]])
y = np.array([0.3, 0.5, 0.7])

rbf_net = RBFNetwork(num_centers=2, input_dim=2)
rbf_net.train(X, y)
print("Prediction:", rbf_net.predict(X))
```

```
Prediction: [0.27256657 0.57485071 0.64487192]
```

[5]:

```
import numpy as np

class HopfieldNetwork:
    def __init__(self, size):
        self.size = size
        self.weights = np.zeros((size, size))

    def train(self, patterns):
        for pattern in patterns:
            self.weights += np.outer(pattern, pattern)
        np.fill_diagonal(self.weights, 0)  # No self-connections

    def recall(self, pattern, steps=5):
        for _ in range(steps):
            pattern = np.sign(self.weights @ pattern)
        return pattern

# Example usage
patterns = np.array([[1, -1, 1, -1], [-1, 1, -1, 1]])
hopfield_net = HopfieldNetwork(size=4)
hopfield_net.train(patterns)

input_pattern = np.array([1, -1, 1, -1])
output_pattern = hopfield_net.recall(input_pattern)
```

```
print("Recalled Pattern:", output_pattern)
```

```
Recalled Pattern: [ 1. -1.  1. -1.]
```