Compute Intelligence PS s7 lab 5

Joris Plaščinskas

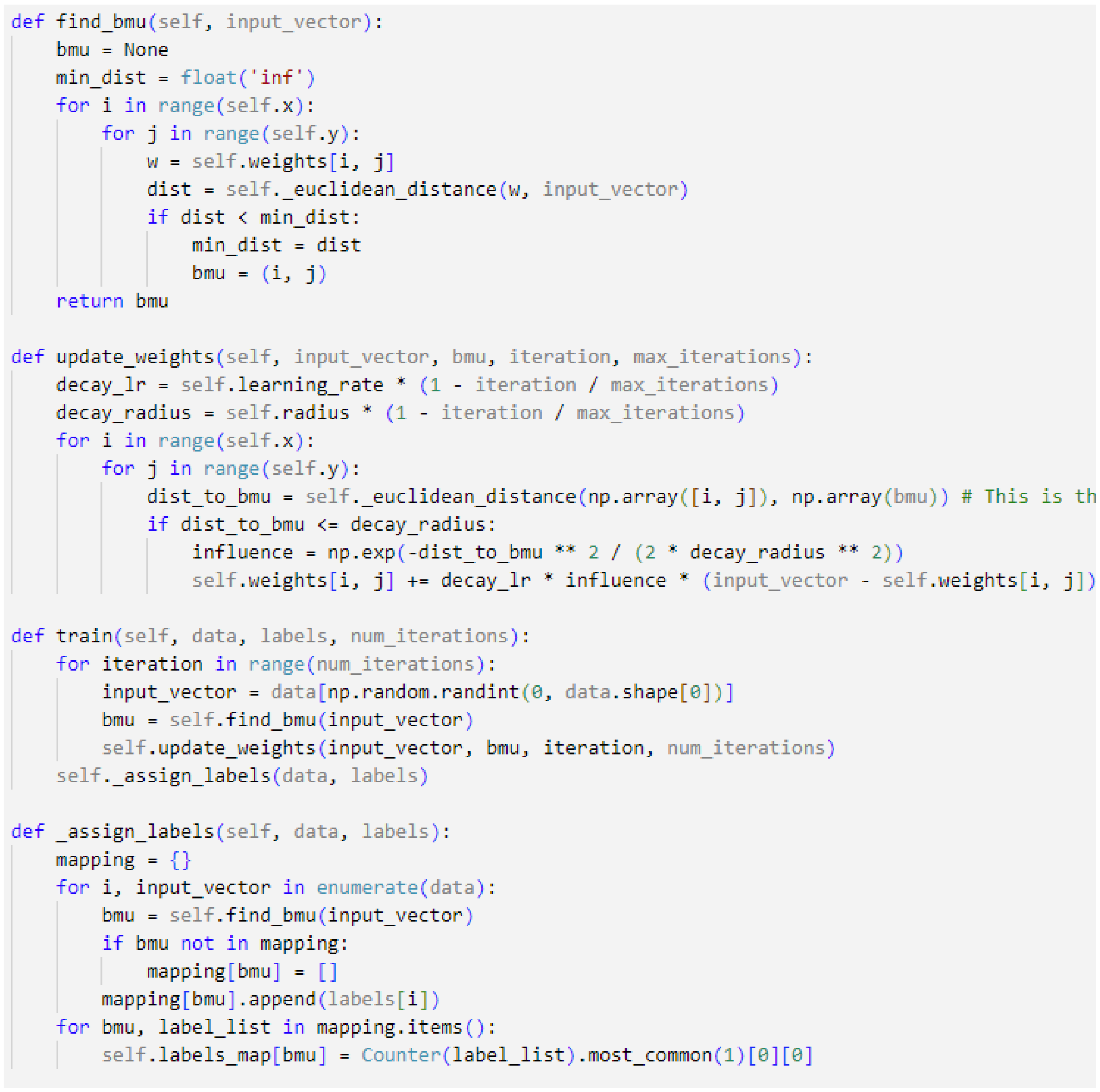
December 3, 2024

Introduction & Data

In this laboratory work I explored SOM neural network. I chose to train the model with Iris dataset and used the code from lab-2 to pre-process the data and make an 'iris.csv' file. The iris dataset contains 150 data-points. Each data-point has 4 features and is classified into 1 of 3 classes.

Training Algorithm & Code

The SOM neural network training algorithm is iterative. During each iteration a random data-point is selected from the train dataset. Next step is to find the best matching unit (BMU) - this is done by calculating the euclidean distance between each neuron and the data-point and selecting the neuron with smallest distance, then the weights of the BMU and other neighboring neurons are updated. The neighboring neurons are the ones that are close to BMU (close by index in the array). The distance for neurons that are considered neighbors gradually decreases. Finally after 'training' the labels have to be assigned to each neuron, this is done by grouping data-points into closest neurons and assigning the most common label to the neuron. The code used for training can be seen in Figure 1 below.



```
def find_bmu(self, input_vector):
    bmu = None
    min_dist = float('inf')
    for i in range(self.x):
        for j in range(self.y):
            w = self.weights[i, j]
            dist = self._euclidean_distance(w, input_vector)
            if dist < min_dist:
                min_dist = dist
                bmu = (i, j)
    return bmu

def update_weights(self, input_vector, bmu, iteration, max_iterations):
    decay_lr = self.learning_rate * (1 - iteration / max_iterations)
    decay_radius = self.radius * (1 - iteration / max_iterations)
    for i in range(self.x):
        for j in range(self.y):
            dist_to_bmu = self._euclidean_distance(np.array([i, j]), np.array(bmu)) # This is th
            if dist_to_bmu <= decay_radius:
                influence = np.exp(-dist_to_bmu ** 2 / (2 * decay_radius ** 2))
                self.weights[i, j] += decay_lr * influence * (input_vector - self.weights[i, j])

def train(self, data, labels, num_iterations):
    for iteration in range(num_iterations):
        input_vector = data[np.random.randint(0, data.shape[0])]
        bmu = self.find_bmu(input_vector)
        self.update_weights(input_vector, bmu, iteration, num_iterations)
    self._assign_labels(data, labels)

def _assign_labels(self, data, labels):
    mapping = {}
    for i, input_vector in enumerate(data):
        bmu = self.find_bmu(input_vector)
        if bmu not in mapping:
            mapping[bmu] = []
        mapping[bmu].append(labels[i])
    for bmu, label_list in mapping.items():
        self.labels_map[bmu] = Counter(label_list).most_common(1)[0][0]
```

Figure 1: Training Code

SOM Visualized

The Figures 2 & 3 show which neurons predict which labels in 5x5 & 10x10 SOM's.

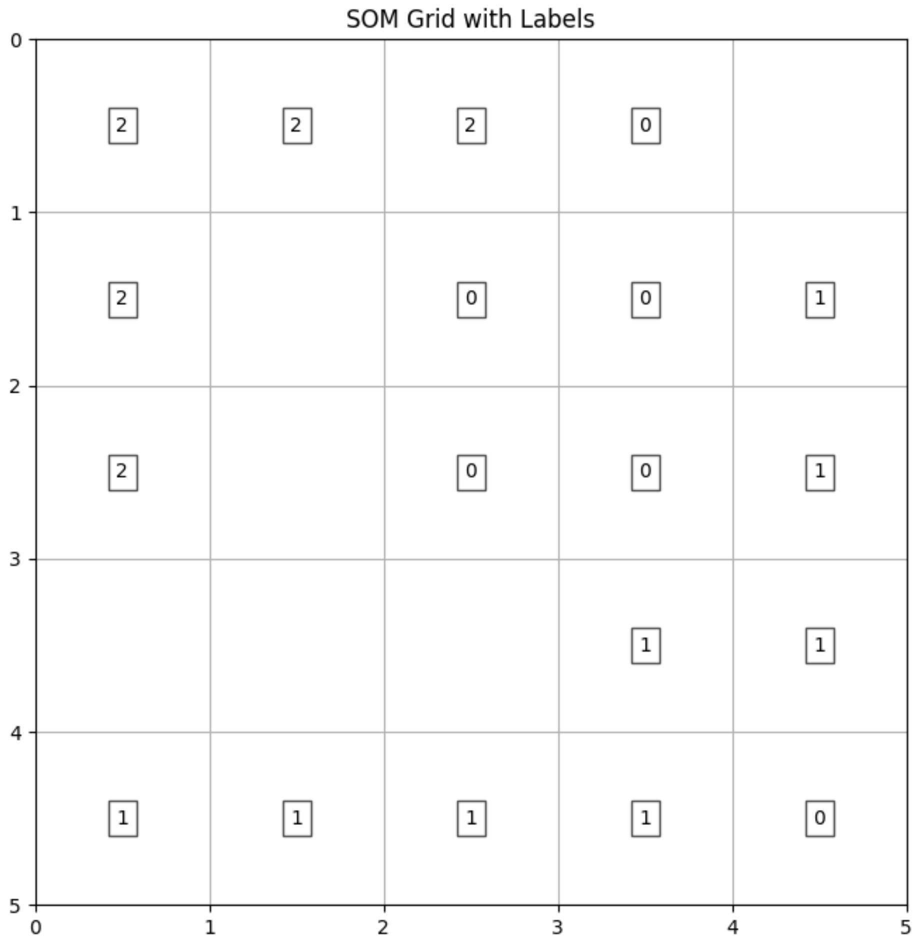


Figure 2: 5x5 Labels Visualized

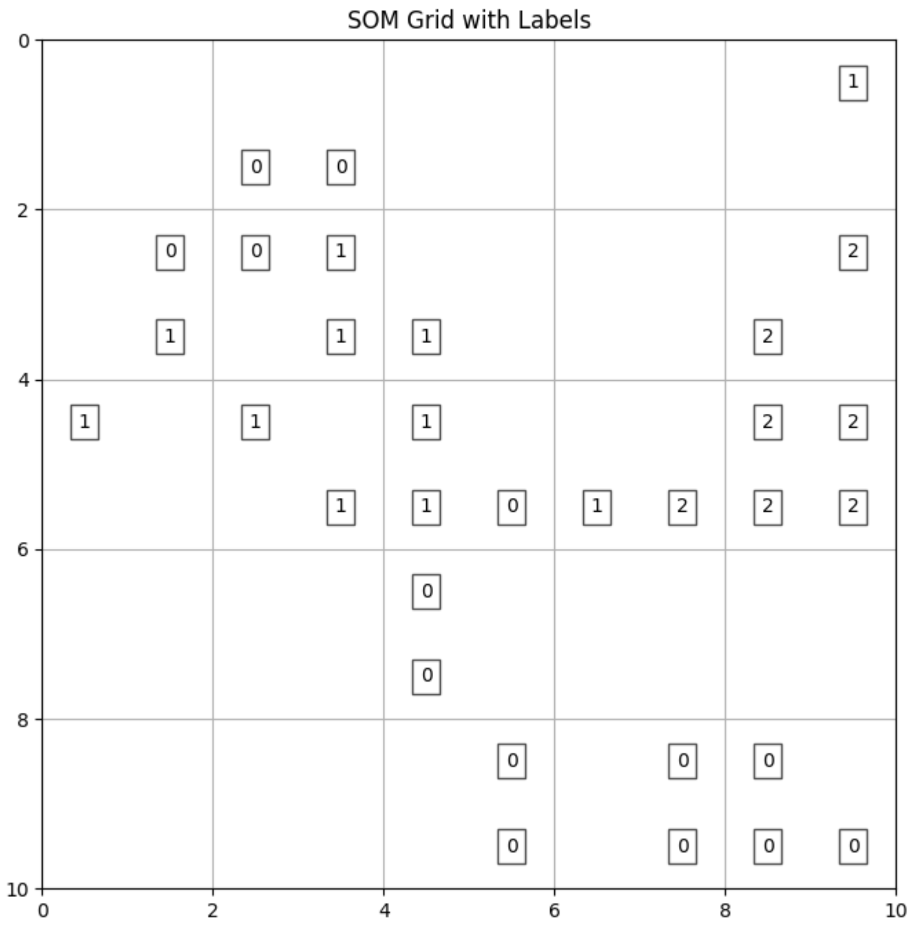


Figure 3: 10x10 Labels Visualized

Results

The 5x5 SOM showed the best results with roughly 95% accuracy on average. This result basically matches the neural network results from lab-2. The 10x10 SOM also did well with roughly 85% accuracy. Setting the learning rate high lead to model only classifying one class, while lowering the learning rate slightly improved the performance of 10x10 model.