Exam Assignment

Autoencoder network for image compression

Soft Computing Laboratory

Przemysław Zdrzalik 244925

07.01.2023

# Main goal

Main goal of the assignment was to write a program, implementing multi- layer perceptron, including backpropagation algorithm. This network is to be used as an autoencoder for image compression.

# Theoretical background

In order to complete task, multilayer perceptron was implemented based on Exam task instructions.

# Implementation

Multilayer perceptron network was implemented using Python programming language. Below are presented most important parts of implementation:

Listing 3.1 - constructor of the neural network.

class NeuralNetwork:

(…)  
 def \_\_init\_\_(self, number\_of\_neurons\_hidden\_layer, number\_of\_neurons\_output, number\_of\_inputs, is\_bias):  
 # czy uruchomilismy bias  
 self.is\_bias = is\_bias  
 self.iteration = 0  
 weight\_possible = 0.2  
 weight\_possible2 = 0.1  
 self.hidden\_layer = (weight\_possible \* numpy.random.random(  
 (number\_of\_inputs, number\_of\_neurons\_hidden\_layer)).T - weight\_possible2)  
 self.delta\_weights\_hidden\_layer = numpy.zeros((number\_of\_inputs, number\_of\_neurons\_hidden\_layer)).T  
 self.output\_layer = weight\_possible \* numpy.random.random(  
 (number\_of\_neurons\_hidden\_layer, number\_of\_neurons\_output)).T - weight\_possible2  
 self.delta\_weights\_output\_layer = numpy.zeros((number\_of\_neurons\_hidden\_layer, number\_of\_neurons\_output)).T   
 if is\_bias:  
 self.bias\_hidden\_layer = (  
 weight\_possible \* numpy.random.random(number\_of\_neurons\_hidden\_layer) - weight\_possible2)  
 self.bias\_output\_layer = (  
 weight\_possible \* numpy.random.random(number\_of\_neurons\_output) - weight\_possible2)  
 else:  
 self.bias\_hidden\_layer = numpy.zeros(number\_of\_neurons\_hidden\_layer)  
 self.bias\_output\_layer = numpy.zeros(number\_of\_neurons\_output)  
 # taka sama warstwa delty jak dla layerów  
 self.bias\_output\_layer\_delta = numpy.zeros(number\_of\_neurons\_output)  
 self.bias\_hidden\_layer\_delta = numpy.zeros(number\_of\_neurons\_hidden\_layer)

Listing 3.1 - constructor of the neural network.Presents the constructor of the neural network class. The class contains information about all the layers of the network, with each layer being represented as a numpy array.

Listing 3.2 - calculating the outputs of the network.

def calculate\_outputs(self, inputs):  
 hidden\_layer\_output = self.linear\_fun(numpy.dot(inputs, self.hidden\_layer.T) + self.bias\_hidden\_layer)  
 output\_layer\_output = self.linear\_fun(  
 numpy.dot(hidden\_layer\_output, self.output\_layer.T) + self.bias\_output\_layer)  
  
 return hidden\_layer\_output, output\_layer\_output

The Listing 3.2 - calculating the outputs of the network. Shows the function used to calculate output from both the hidden layer and the output layer of the network. In this case all neurons in the network are linear neurons, so the value of the identity function is used.

# Experiments and results

To complete task, eight experiments ware conducted. In these experiments eight 512x512 greyscale images were encoded. In each experiment the encoding was being performed with a network trained with different amount of neurons in the hidden layer. The amounts were 1, 2, 3, 4, 8, 16, 32, 64, 128 and 256. The images were read by the network as 8x8 chunks of 64 pixels, so according to the theory presented in the instructions, from 64 neurons in the hidden layer the compression should be perfect.

* 1. **Experiment № 1**

from network without bias.

# Summary and conclusions

Conducted experiment allowed us to discover difference between networks with and without bias, and allowed us to find a regularity in hidden layer output values. The experiment proved that networks using bias can reck- ognize more patterns than networks with simmilar parameters, but without bias.