UMEÅ UNIVERSITY

Institution for Computer Science

Fundamentals of Artificial Intelligence 7.5 HP

5DV121 HT 16

Assignment 2

Neural network

Namn		**Användarnamn**
Kristoffer Karlsson		kv14kkn@cs.umu.se
Jonatan Gustavsson		Kv14jgn@cs.umu.se
Handledare	Alexander Sutherland	

Contents

Assignment specification

The assignment is to create an artificial neural network capable of recognising patterns. The structure of said network is a perceptron based classification system that guesses the emotional state of faces presented as input. 300 images of faces in the form of 20 x 20 pixel maps were given as the input source in a text file for the assignment. The program should train on 300 images and then be tested on 100.

User manual

The application is written in Python 3 and consists of the files faces.py, perceptron.py, training.py, and image.py. The text files training.txt, training-facit.txt and test.txt needs to be in the same folder that contains the Python files. Through the Command prompt; run the files network1.py, training.txt, training-facit.txt, and test.txt, by using the following line: "python facec.py training.txt training-facit.txt test.txt" in windows (assuming that python 3 is default). In Linux use the same statement, but print "python3" instead of "python". The Command prompt will show the result from the training runs in percent correct, and the test run guesses done by the application

Algorithm description

1. By running the text files through the Command prompt, the application will open the files, extract the relevant information from them, and create a grid of pixel values from the faces for each image.
2. The application will create a grid of randomised weights, similar to the images.
3. The application will create four perceptrons, assign each perceptron a grid of weights and a specified emotional state to identify itself with.
4. The input from the images will be run through the perceptrons, and each perceptron will guess the correct emotional state of the image.
   - 4.1. The input from each pixel is multiplied with the corresponding weight assigned to the perceptron.
   - 4.2. The product from each input is summarised and run through a Sigmoid function to get an output.
5. The perceptrons are trained to either recognize or not recognize that, or similar images.
   - 5.1. A calculation of the weights for each perceptron will begin.

        5.1.1. An error value will be calculated by subtracting the guessed value from the desired value.
        5.1.2. The error value will be multiplied with a learning rate and the input from the pixel.
        5.1.3. The product is the differential that is added to the old weight to create a new weight.
        5.1.4. Each weight goes through this performance and is added to a new grid of weights.
    5.2. The new weight grid replaces the old one in the perceptron.
6. The perceptron with the highest output is activated.
7. If the correct perceptron has been activated, a point will be added to a score list.
    7.1. The emotional state of the activated perceptron is compared to an answer sheet and if it's correct a point will be added to the score list.
8. The number of points is compared to the number of images and a percentage of correct guesses is calculated.
9. If the percentage is above 75% the test session will commence. If not, the perceptrons will go through training again.
10. In the test session the perceptrons guesses on each image as in step 4. And the one with the highest output is stored.
11. The program prints out its guesses.

System description

Classes:

For flowchart and class UML look at appendix 1 and 2.

- Image
    - Creates an object from a 2D-list
    - Has methods to return information of rows, columns, or entire images
- Perceptron
    - Creates an object containing a grid of weights, a number specifying its emotional state, and an output value default defined as 0
    - Methods to calculate activation levels, generating an output, and to return:
        - Specific weight information

        - Emotional state
        - Output
    - Change the weights in its grid
- Training
    - Creates an object containing an image, a perceptron, the correct output value, the learning rate value, and a wanted value
    - Has methods for calculating:
        - Error value
        - New weights
    - Sends the newly calculated weight to the perceptron in training

Application:

- The application begins by receiving input from the Command prompt
    - The function get_image_info extracts the information from given image files
        - Is used to read information from both training and test files
    - The function image_create creates objects from the information gathered by get_image_info
    - The function get_facit_info extracts the correct answers from the file and creates a list with them.
    - The function create_weights creates a grid of randomised weights, one for each perceptron
    - The class Perceptron is called upon to create four objects (the perceptrons).
    - A while-loop with the condition to run until the percentage of the correct guesses surpasses 75% executes the activation-method in the perceptron objects
        - The class Training is called upon creating an object for each perceptron where it is trained on one image. This is going on until every perceptron has trained on every image.
        - The function get_winner picks the perceptron with the highest activity
        - The “winner” is appended to a result list.
        - The function calc_points compares the results to an answer sheet and adds a point to the point variable if the guess is correct
        - Then the points are divided by the amount of images to get the percentage of correct guesses

        - The percentage is then printed with a string
    - When the percentage of correct guesses has surpassed 75% the test session begins
    - The objects from the test images are run through each perceptron
    - The guesses from the perceptrons are calculated like those during the training session, but without the perceptrons being trained
    - The final score of the test session is printed using the function print_func

Problems and reflections

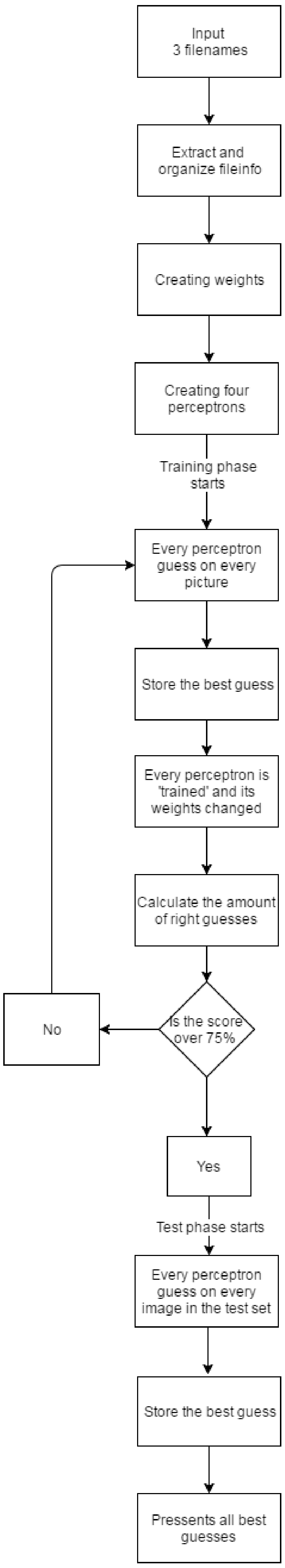
During the assignment, we had some difficulties grasping the concept of the task and the structure of the code that was to be implemented. Essentially, we had a lot of trouble in the beginning, not knowing how to implement the information given to us. But thanks to a lot of guidance, we started to understand the concept of how the network was supposed to be structured and what equations to use to make the network functional.

Finally moving forward, we started creating our classes and the necessary methods. Although moving forward, it was a slow start and a slow pace for us. But as soon as we had created our perceptron class everything else started to shape up. With some more guidance of how the training of the perceptron was supposed to work, we picked up a lot more speed and beside a few smaller problems, mostly due to our own carelessness with creating our own training and answer files, our application was now functional.

Luckily, no major problems nor any major difficulties occurred during the assignment. Those problems and difficulties that did occur, however, were due to misunderstandings, uncertainties and lack of clarification in the assignment instructions. Personally, we feel that a scheduled lesson or a demo of how to start (like with the robot) would have been a lot of help.

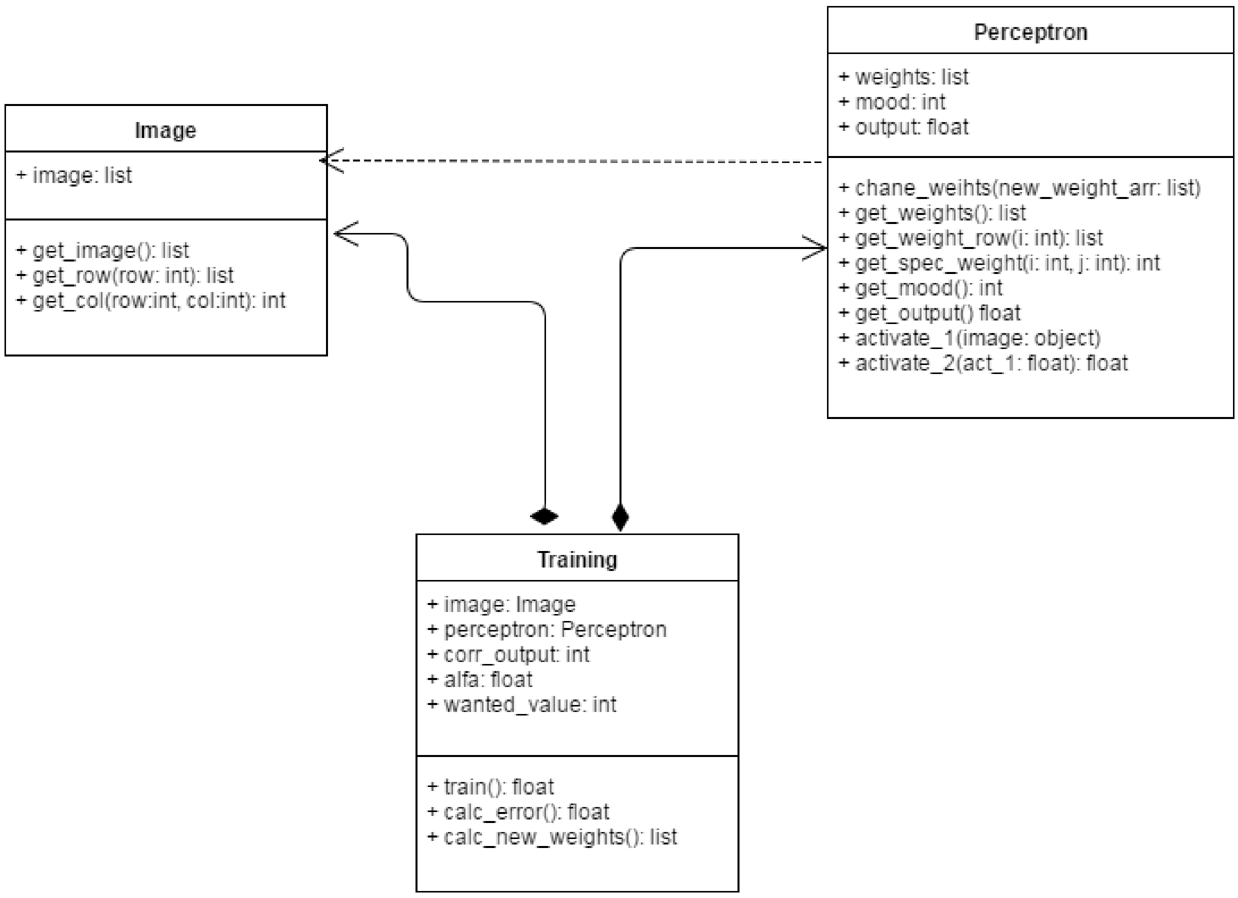
Appendix

1. Flowchart.



Input 3 filenames
Extract and organize fileinfo
Creating weights
Creating four perceptrons
Training phase starts
Every perceptron guess on every picture
Store the best guess
Every perceptron is 'trained' and its weights changed
Calculate the amount of right guesses
Is the score over 75%
No
Yes
Test phase starts
Every perceptron guess on every image in the test set
Store the best guess
Pressents all best guesses

2. Class UML


Perceptron
+ weights: list
+ mood: int
+ output: float
+ chane_weihts(new_weight_arr: list)
+ get_weights(): list
+ get_weight_row(i: int): list
+ get_spec_weight(i: int, j: int): int
+ get_mood(): int
+ get_output() float
+ activate_1(image: object)
+ activate_2(act_1: float): float
Image
+ image: list
+ get_image(): list
+ get_row(row: int): list
+ get_col(row:int, col:int): int
Training
+ image: Image
+ perceptron: Perceptron
+ corr_output: int
+ alfa: float
+ wanted_value: int
+ train(): float
+ calc_error(): float
+ calc_new_weights(): list