19. Lab: Implementing Neural Networks

Goal: understand how to create an ANN and the pros and cons of training it **(6 hours)**

Lab

1. Using WEKA load and run ANN (multilayer perceptron) example and analyse the inputs and outputs.
2. Move around the factors such as learning rate, number of layers and number of nodes. Record the behaviour of the network. We are interested in observing if it converges or not, how long it takes to learn, and how precise it is. These measurements and comparisons should be included in your report.
3. Find an interesting training set in http://archive.ics.uci.edu/ml/datasets.html and use it to create test a network.
4. Implement a single layer ANN (perceptron) using the language of your choosing and train it to recognize something small. The user should be able to query the trained ANN (i.e. supply inputs and get an output).

Your program should be runnable by executing the following command:

- `run_program [dataset]`

If the program doesn't run (classpath not set, files not properly compiled, etc), I will assume that more than 30% of the functionality is missing. See the rubric for the consequences of this: https://docs.google.com/document/d/1MPnk47ws7wEyS5mM-YbzkhjgzUY5CkEKX2WvXhlq7S0/edit . Talk to me first if you plan to use a programming language that is not in this list: Python, Java, C, C++, Ruby.

Report

Write a report using the comparisons collected by training the ANN with different parameters. Include an image of the network and the parameters used as well as the error and time it took to train each network. Write a brief reflection about what you think is happening in the different ANNs. Other interesting aspects you could include in your reflection are: Explanations as to what are ANNs good for. Where would you use them? Are they worth the effort implementing or not? What kinds of problems do they not solve?

Weka function:

MultilayerPerceptron

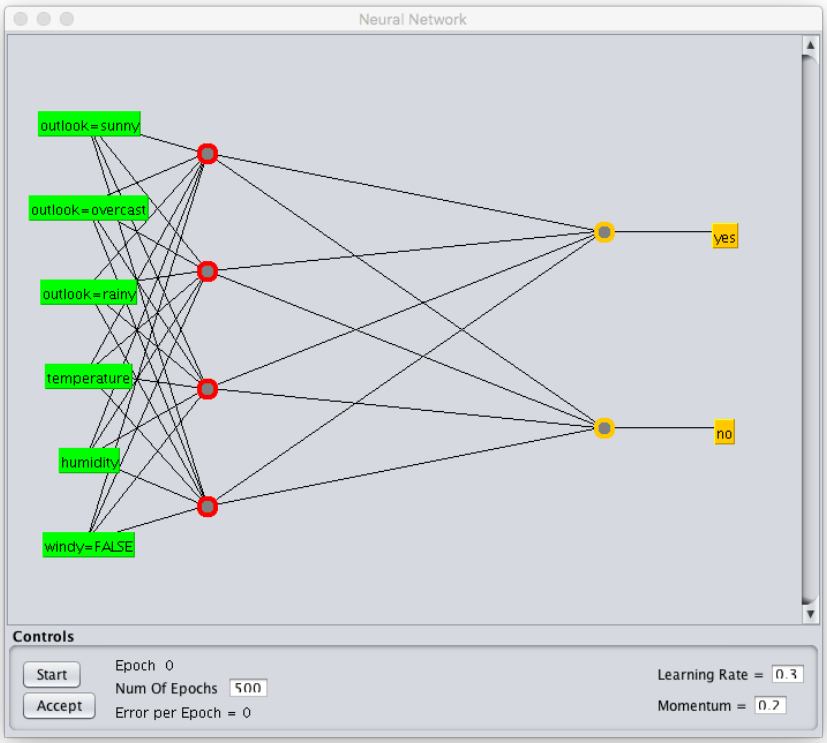
Dataset:

weather.arff UNREGISTERED

weather.arff

```
1  @relation weather
2
3  @attribute outlook {sunny,
   overcast, rainy}
4  @attribute temperature real
5  @attribute humidity real
6  @attribute windy {TRUE, FALSE}
7  @attribute play {yes, no}
8
9  @data
10 sunny,85,85,FALSE,no
11 sunny,80,90,TRUE,no
12 overcast,83,86,FALSE,yes
13 rainy,70,96,FALSE,yes
14 rainy,68,80,FALSE,yes
15 rainy,65,70,TRUE,no
16 overcast,64,65,TRUE,yes
17 sunny,72,95,FALSE,no
18 sunny,69,70,FALSE,yes
19 rainy,75,80,FALSE,yes
20 sunny,75,70,TRUE,yes
21 overcast,72,90,TRUE,yes
22 overcast,81,75,FALSE,yes
23 rainy,71,91,TRUE,no
24
```

Line 26, Column 1 | Tab Size: 4 | Plain Text



Round 1 (Using Weka's standards)

Using Cross-Validation with 10 Folds

Correct Classified Instances: 79%

Weka chose 4 nodes due to the avg of the input and output layers

Time taken to build model: 0.01 seconds

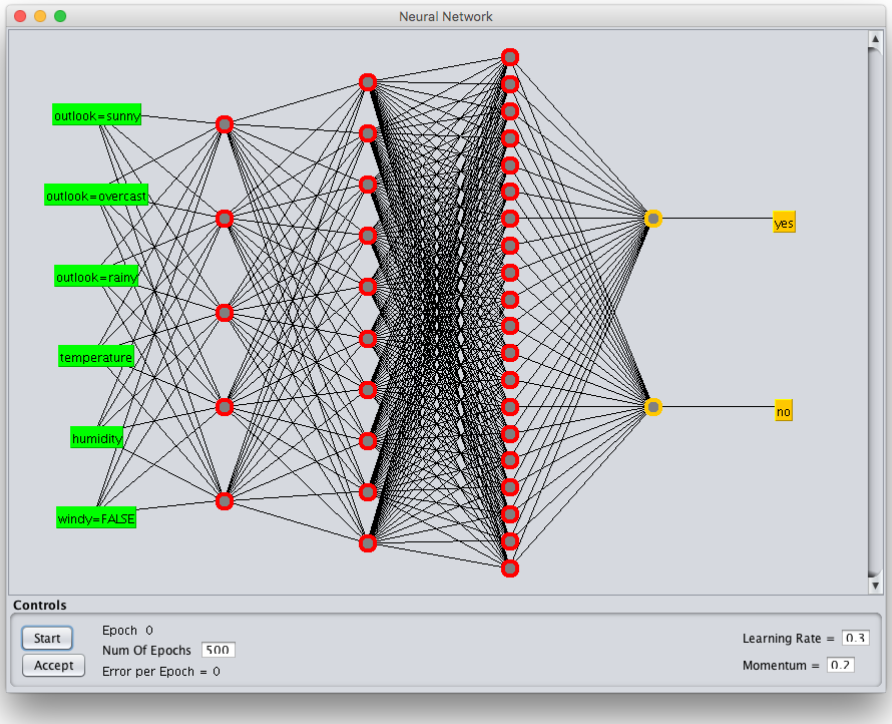
Round 2:

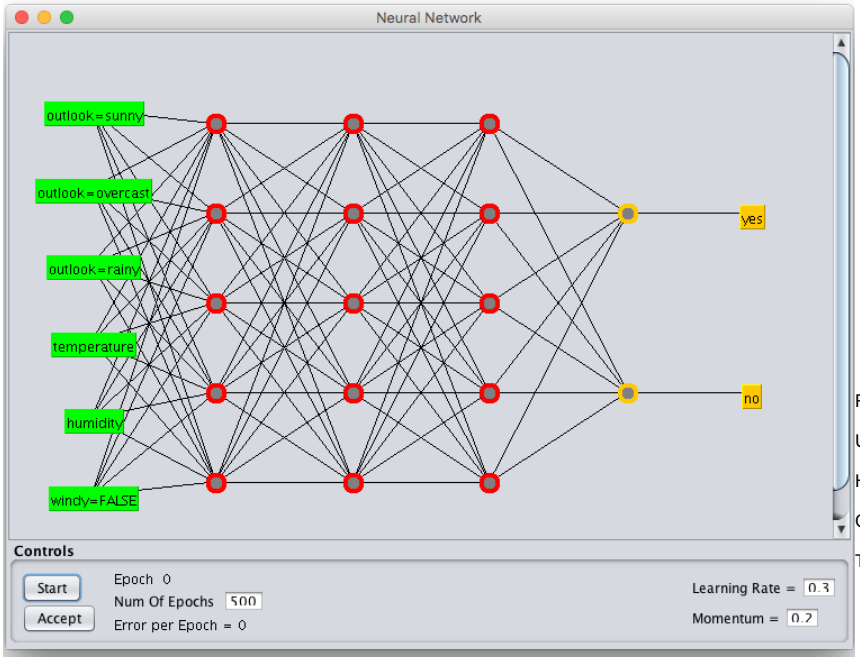
Using Training set

HiddenLayerSettings: 5, 10, 20. 3 hidden layers

Correct Classified Instances. 92.8571%

Time taken to build model: 6.8 seconds





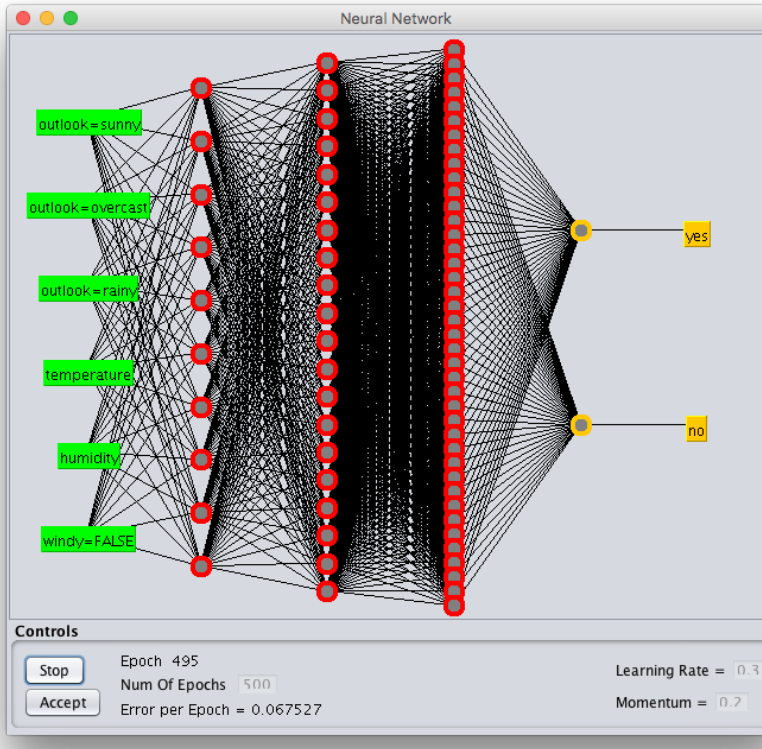
Round 3:

Using Training set

HiddenLayerSettings: 5, 5, 5. 3 hidden layers

Correct Classified Instances. 64.8571%

Time taken to build model: 15.59 seconds



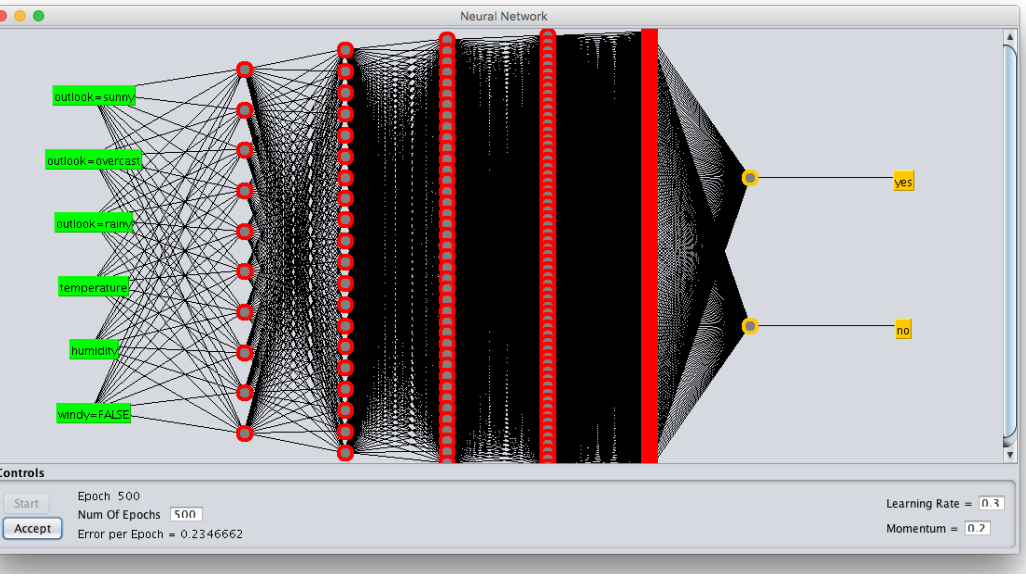
Round 4:

Using Training set

HiddenLayerSettings: 10, 20, 40. 3 hidden layers

Correct Classified Instances. 92%

Time taken to build model: 48 seconds



Round 5:

Using Training set

HiddenLayerSettings: 10, 20, 40, 60,120 5

hidden layers

Correct Classified Instances. 64%

Time taken to build model: 14 seconds

Conclussions:

If we just add hidden layers and neurons the number of epochs (passes) will not only increase but also the "error per epoch" will increase (as shown in the images) . Naturally, the bigger amount of neurons and layers, the more processing time. In this case, Weka's default 1 hidden layer method only gave us 70% of accuracy.. Increasing it to a 5,10,20 approach gave us a neat 92% . Curiously, a 5,5,5 approach gave less than the default 1 hidden layer one. Needless to say, the 5 layer approach went down to 64% accuray and 14 secs processing time.

In the official Weka Youtube channel: https://www.youtube.com/watch?v=mo2dqHbLpQo they also tried the same approach with different Weka ML tools and here are his results:

Weka's default MultilayerPerceptron acc: 79%

J48 Decision Tree: 64%

NaiveBayes: 64%

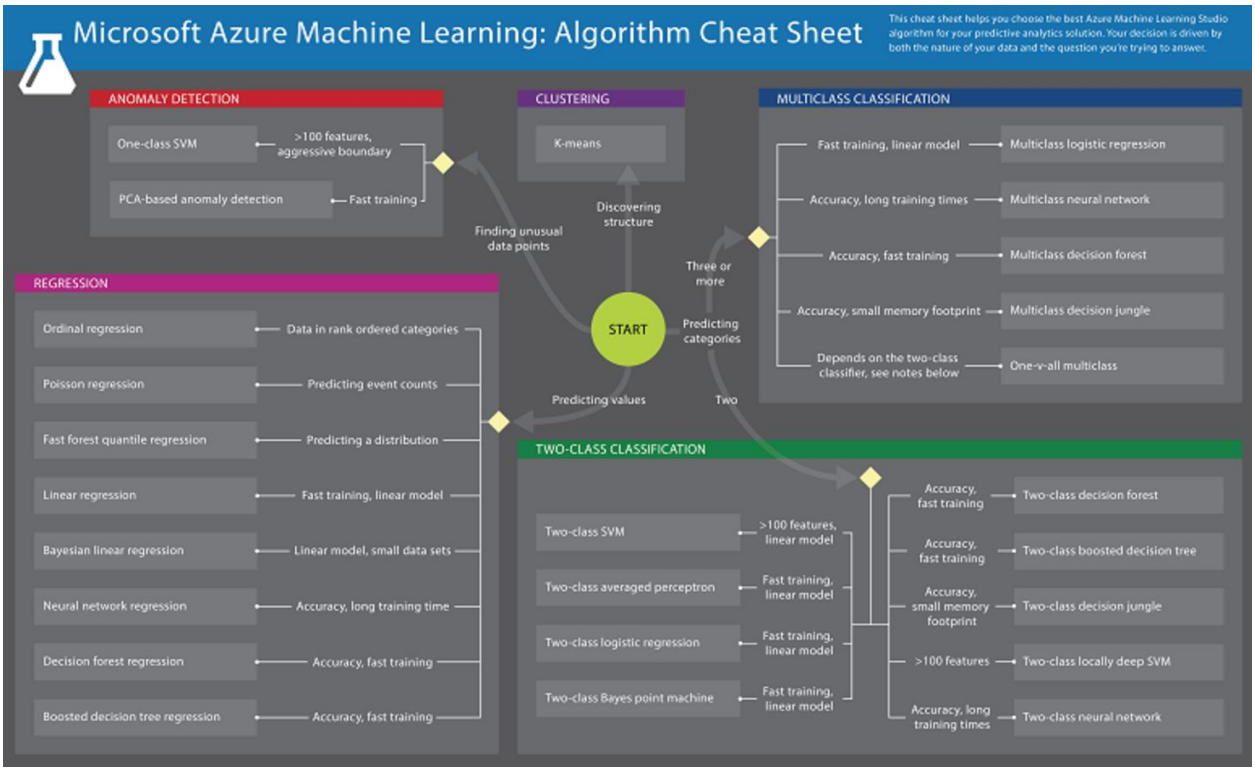
SMO 57%

IMk 79%

As they highlighted, "on real problems does quite well, but slow" Apparently it's 10 to 2000 slower than other methods and has an advantage of 2 vs 1 against SMO.. and only advantage of 1 against J48 and IBk. All of this tested against 6 datasets.

When can we use neural networks?

We've found a quite useful cheatsheet provided by Microsoft:



https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-algorithm-cheat-sheet

So basically and according to Microsoft, Neural Networks are best used when:

We want Accuracy

We will stand long training times

Laura Diane, technical manager at groupon also made a cheat sheet for when can we use ANN, she basically said this: ->

Neural networks	- Extremely powerful - Can model even very complex relationships - No need to understand the underlying data – Almost works by "magic"	- Prone to overfitting - Long training time - Requires significant computing power for large datasets - Model is essentially unreadable	- Images - Video - "Human-intelligence" type tasks like driving or flying - Robotics

http://www.lauradhamilton.com/machine-learning-algorithm-cheat-sheet

So here you go, folks: neural networks while taking quite a lot of training time, need caring and adjusting for them not to overfit (Weka already takes care of some of that) and being really accurate, they serve intensely for Images, video, Human intelligence tasks, robotics and once the model is , modeled, the reaction time is quite fast. The modeling itself and learning process, not so fast though. You can basically input whatever dataset you want (as long as we have the expected outcome) and with a ton of time and caring, you can have a method that works like magic (highly accurate magic) .