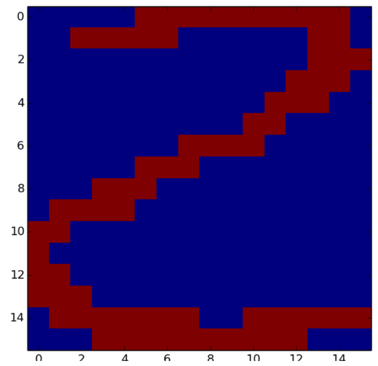
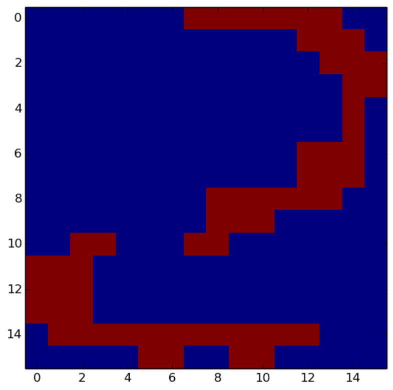
EECS 349 Homework 7

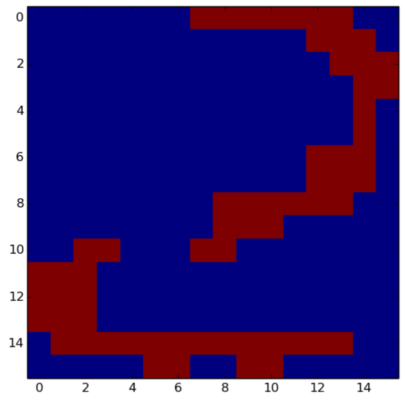
Problem 1:
a) the code is attached in hop_net.py
b) the code is attached in hop_net.py
c) After the data is parsed correctly, I create a training set with one example for each of the first occurrences from the set {0,1,2,3,4}. Then I use the digit 2 since this where the hop field works the best. The corresponding code is in the function probc. For none of the 5 digits, the hop field iterated more than 3 times. Only for 2 and 4, does the hop field return with the correct digit. In all other cases I get a weird pattern result. In the case of 0 it turns it into a 2. This also happens with a 1 as input. I hypothesize that this is because of the limited training data used to build the net. The provided training data is also not very general, but rather very "handwritten" and subjective. I do not think that this would be good enough to build a classifier. Maybe with more data, however, there is room for much improvement.



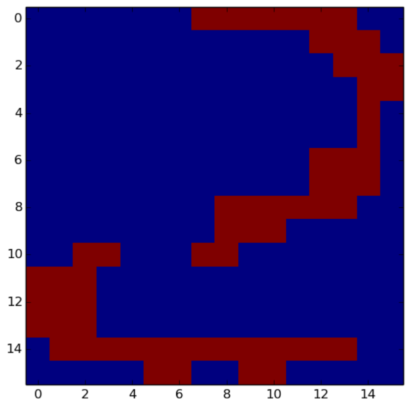
Input



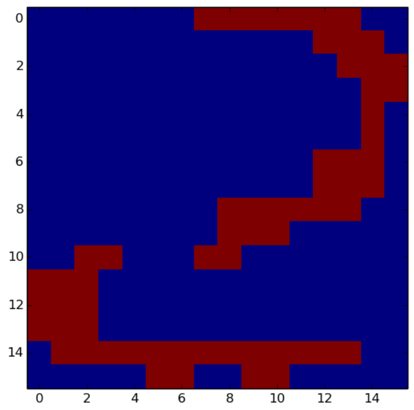
Iter 1



Iter 2

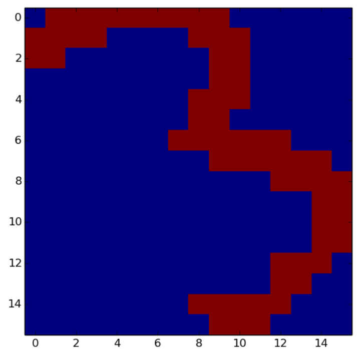


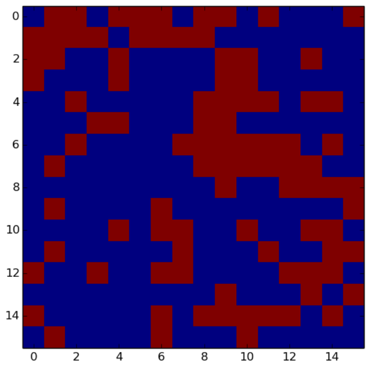
Iter 3

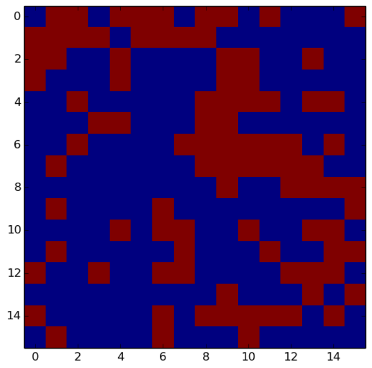


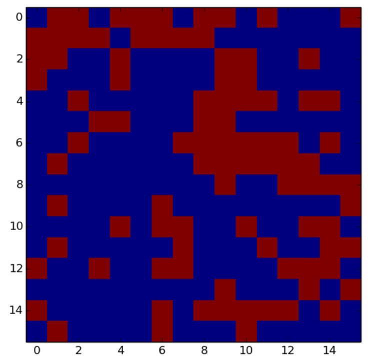
Output

d) The hopfield is trained on a pair of digits: 2 and 6. I then flip 20% of the bits randomly. In all my attempts, I have not been able to achieve convergence on an actual digit. I decided to try it with other sample data, however, I was not able to achieve convergence on a digit with those either. In all cases, the noise remained in the image and the hopfield said it had converged after just 1 iteration. Thus, the series doesn't look similar to the one presented in the slides.


input


Input w noise


Iter 1


output

Problem 2:
a) Strictly speaking, a single layer perceptron can only classify if the decision surface is linearly separable. This is because of the perceptron's definition:

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Here, w*x is the dot product of the weights with the inputs. However, if there is only one input that is non-linearly separable then one could transform that input into a higher dimensional space (see previous homework) and then learn the weights on that as a linear decision surface. But in the context of neural nets, a perceptron cannot represent non-linear decision surfaces.

b) Multi layer perceptrons can learn any decision surface as long as they are set up properly. The big caveat, however, is that there is no guarantee of convergence to minimal error weights.

c) The most important advantage of using sigmoids instead of things like a step function is that they are differentiable. This allows for the continuous observation of output change w.r.t to change in a parameter. That way, the parameters to the multi-layer net can be learned by differentiation.

d) If the sigmoid was replaced, this would eliminate the ability to backtrace the effect of one parameter's change on the net's output. Especially because the activation wouldn't be differentiable at the intersection of -0 to +0 anymore and also because there is no observable change in the regions >= 0 and < 0. Thus, the network could not be trained for minimizing error.

Problem 3:

a) A RBM is a neural net consisting of individual computational/processing units that underly strict connectivity/topological restrictions. Most importantly, units in one layer are not connected to units in another. The two layers present in a RBM are hidden and visible, Each hidden unit is connected with every visible unit and vice versa. Data/information flows bidirectionally. RBMs can be used to infer information about unknown probabilistic distributions. The hidden layer models dependencies between the components of observations (if we are working with images, for example, this could be the relationship between individual pixels) while the visible layer models the state of every one input component.

b) The relationship between Deep Belief Networks and RBMs is the following: recently, deep belief networks have been proposed leveraging parallel computing in multi-layered setups. As part of these multi layered setups, RBMs constitute fundamental building blocks. The idea is that hidden layers can extract features from observations which can then be fed into further layers of RBM for extended in-depth processing.

Probem 4:

a) Kernel Machines are neural nets that consist of two layer: the large layer consists of simple template matchers, while the smaller layer is used to train coefficients.

b) One of the fundamental limitations of Kernel Machines when compared with other frameworks is their efficiency. It can be shown that – while Kernel Machines can model a very wide variety of functions – they can only do this very inefficiently when compared to other systems. The other two limitations described in the paper are the intrinsic tradeoff between breadth vs depth in the setup of the KM, and more specifically in the context of local kernels the definition of a prior distance of similiarity functions between pairs.

c) Deep Architectures are a novel approach using complex arrangements of non-linear operators with many hidden and visible layers. They can be used to learn complicated functions that can represent high-level abstractions (e.g. natural language processing, computer vision, or other AI level tasks).

d) By using deep architectures built from (e.g.) RBMs, even the lowest level features can be extracted and linked back to high level concepts through the multiple layers. This is possible because of the unsupervised learning nature of RBMs and it can be accomplished with very little human effort during training.