Hopfield network

Balazs B Ujfalussy

20/11/2017

This is a demo for illustrating the concept of attractor network using the Hopfield network. The Hopfield network models associative memory by storing a number of unique patterns in a recurrent neuronal network. It also illustrates the concept of content addressable memory, and was extremly influential as a model for episodic memory, where each stored pattern forms the representation of an event.

It would be much simpler to rewrite this demo using the original Hopfield equations - where the dynamics is much simpler and analytical derivations can be easily done.

Important concepts:

- storage and recall
- attractor
- converence
- energy function

Storing patterns in a recurrent network

We will use a standard binary Hopfield network: each neuron has two state, 0 and 1. The sparseness of the patterns defines the proportion of 1s in the data, which is defined by the parameter `f`. We will have $32 \cdot 32 = 1024$ neurons, and will store M patterns.

```
library(gplots)
```

```
##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess
```

```
f <- 0.5 # the sparseness of the representation: percent of
sig2.w <- f^2 - 2*f^3 + f^4 # the varinace of the weights after storing a single pattern
N <- 1024 # number of neurons
# plot(seq(0, N), dbinom(seq(0,N), N, f))

M <- 50 # number of patterns stored try with 100, 200 and 300
X <- matrix(rbinom(N*M, 1, f), M, N) # the patterns to be stored
```

The first 2 patterns are meaningful images - these patterns do not differ from the other patterns in any other way, just their pixels are ordered in a way that seems meaningful for us. We red these patterns from png imagefiles in the directory `Demos/Fig32s/`.

```
library(png)
ims <- array(NA, dim=c(2, 32, 32))
imnames <- list.files('Demos/Figs32/')
par(mfcol=c(2,4)) # two rows, 4 columns
par(mar=c(1,1,1,1)) # small margos
for (i.image in 1:2){
    imname <- paste('Demos/Figs32/', imnames[i.image], sep='')
    im1 <- readPNG(imname)
    for (i in 1:32){
        for (j in 1:32){
            ims[i.image, i,j] <- round(1-im1[33-j, i,1])
        }
    }
    image(ims[i.image,,], col=grey(seq(0,1, by=0.1)), axes=F)
    X[i.image,] <- as.vector(ims[i.image,,])
}
```

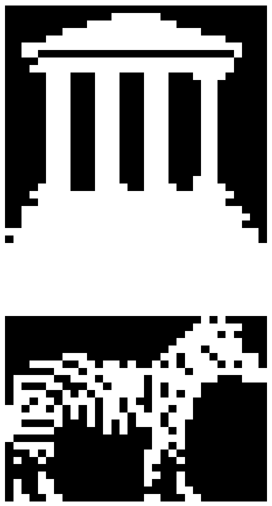

Figure 1: The first 8 stored patterns

```
print(round(apply(ims, 1, sum) / 32/32, 2))
```

```
## [1] 0.50 0.47
```

Training in a neuronal network means to change the strength of the synaptic weights in order to improve the network's performance.

There are several ways to train a Hopfield network. We will use the simple, offline covariance rule, i.e., the recurrent synaptic weight are set to be the covariance of the training data. This is an offline rule, because to set the weight we need to know first all of the stored patterns. The diagonal is set to zero.

```
W <- t(X-f) %*% (X-f)
diag(W) <- 0
```

To test the network's performance, we will initialise the network with a noisy version of one of the originally stored patterns. We add noise in way to preserve the sparseness of the network.

```
x <- X[1,]
px <- rep(1-f/3, N); px[!x] <- f/3
nx <- rbinom(N, 1, px) # the noisy version of the original

par(mfcol=c(1,2)) # two rows, 4 columns
par(mar=c(1,1,1,1)) # small margos
image(matrix(x, 32), col=grey(seq(0,1, by=0.1)), axes=F)
image(matrix(nx, 32), col=grey(seq(0,1, by=0.1)), axes=F)
```

Next we will test the network with the first 50 patterns. We start the network from the noisy recall cue and then update the neurons simultaneously according the the following rule:

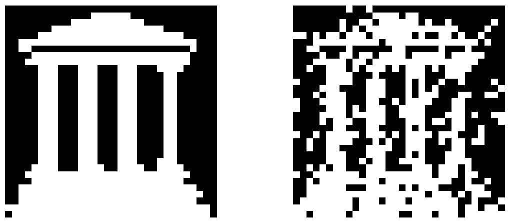

Figure 2: Left: stored image, Right: recall cue

$$\mathbf{x} = \text{round}\left[\frac{1}{\sigma_w^2\,(M-1)}\left(\mathbf{W}\mathbf{x} - f\sum_j W_{ij}\right)\right]$$

where σ_w^2 is the expected variance of the weights after storing a single pattern and the function round returns the closest integer (0 or 1). The recall dynamics seems quite complex, but most of it is required only because we stored patterns with $f \neq 0.5$.

We will check the following measures for each pattern:

- recalled vs target: the difference between the stored and the recalled versions
- recalled vs best: the difference between the stored and the recalled versions
- cue vs. target: the difference between the stored and the original cue
- i.recalled: identity of the recalled pattern

The network works if the correlation with the original, stored attern increases during recall.

Note, that testing all patterns in a Hopfield network can be quite slow!

```r
MM <- min(10, M)
stats.recall <- rep(NA, 9)
kk <- 0
mems <- matrix(NA, 4, MM)
rownames(mems) <- c("recalled vs target", "recalled vs best", "cue vs. target", "i.recalled")

par(mfcol=c(3,7))
par(mar=c(1,1,1,1))

for (m in 1:MM){ # for each pattern
    x <- X[m,]      # we start from the stored pattern

  px <- rep(1-f/3, N); px[!x] <- f/3
  x.t <- rbinom(N, 1, px) # the noisy version of the original
    nx <- x.t
    converged <- F
    k <- 0

    while (converged == F){
        I <- 1/(sig2.w * (M-1)) * (W %*% x.t - f * rowSums(W)) # optimal dynamics from Eq. 6. of Savin
        x.I <- rep(F, N)
        x.I [I>0] <- T
        change.x <- which(xor(x.t, x.I))
        if (length(change.x) == 0){ # convergence
            converged <- T
        } else {
            ch.x <- sample(change.x, 1)
            x.t[ch.x] <- 1 - x.t[ch.x]
        }
        k <- k+1
        if (k>1000) break
    }

    if (m < 8){
        image(matrix(x, 32), col=grey(seq(0,1, by=0.1)), axes=F)
        image(matrix(nx, 32), col=grey(seq(0,1, by=0.1)), axes=F)
        image(matrix(x.t, 32), col=grey(seq(0,1, by=0.1)), axes=F)
    }

    # plot (X %*% x)
    # points (X %*% nx, col=2, pch=21, bg=2, cex=0.7)
    mem <- X %*% x.t
    # points (mem, col=3, t="h")
    i.recalled <- which.max(mem)
    x.recalled <- X[i.recalled,]
    mems[1,m] <- 2 * x %*% x.t / (x.t %*% x.t + x %*% x)
    mems[2,m] <- 2 * x.recalled %*% x.t / (x.t %*% x.t + x.recalled %*% x.recalled)
    mems[3,m] <- 2 * nx %*% x / (x %*% x + nx %*% nx)
    mems[4,m] <- i.recalled
}
```

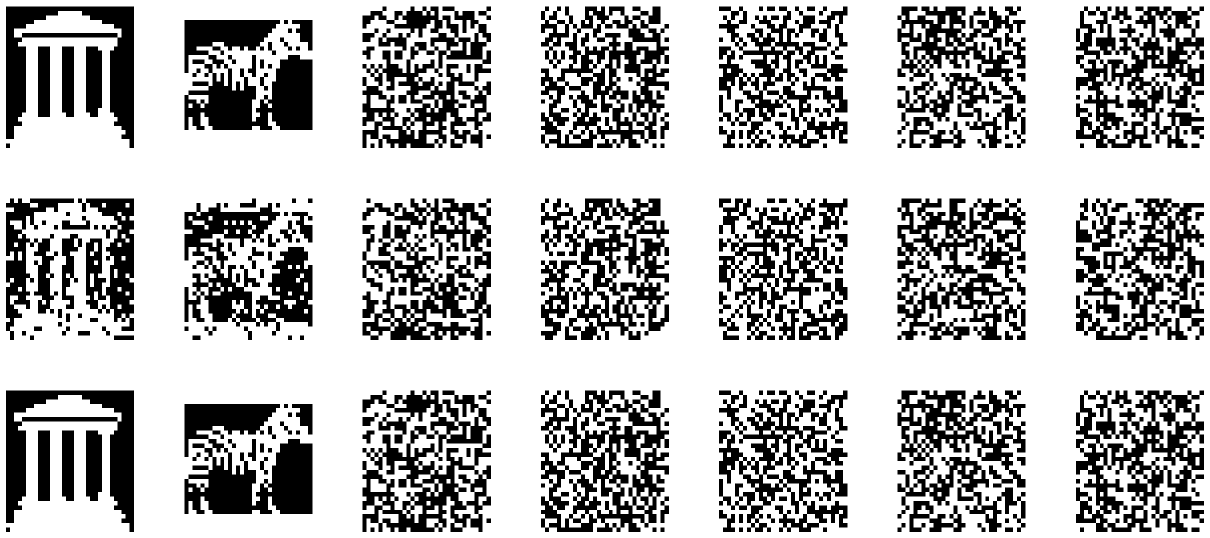
Figure 3: stored pattern, recall cue and recalled pattern

Let's evaluate the network:

- The average correlation of the recalled with the stored: 1
- The average correlation of the *recall cue* with the stored: 0.8336363
- Percent of correctly recalled patterns: 100%

Homework

You can choose betwee two different homeworks:

Homework 1 - theory

Show that the function

$E = -\frac{1}{2} \sum_{ij} w_{ij} \left(s_i - f\right) \left(s_j - f\right)$

is an energy function, i.e., it is guranteed that the function $E(s)$ will not decrease if the units are updated sequentially using the simplified update rule.

What are the implications of having an energy function for a network?

Show, that if y is an attractor of the network, then $1 - y$ is also a stable pattern, stored in the network!

Homework 2 - programming

Show, that if y is an attractor of the network, then $1 - y$ is also a stable pattern, stored in the network!

How would you measure the capacity of the network? Try to store more patterns in the Hopfield network (try e.g., 50-200-1000-5000). Check what happens to the first 50 stored patterns! What is the capacity of the network?

Homework 3 - open questions

How would you improve the Hopfield network to

- make it more efficient - i.e., storing more patterns without interference?
- make it more biologically realistic? Identify at least 3 different features of the Hopfield net that are probably not similar to what we see in the nervous system. How would you change it to be more similar to the nervous system?