Perceptron

Balazs B Ujfalussy

13/11/2017

This is a demo for illustrating learning by a network of neurons.

The task we want to learn is a simple task: classification of inputs. You can think about this as a toy version of classifiying images to two categories: e.g., safe or dangerous. There are two main challange that we need to solve:

1. Images that look totally different might fall into the same categoty.
2. Images that look similar might fall into different categories.

First we illustrate the idea of learning in a simple toy problem, where these challanges play only a minor role. We show that setting the synaptic weights of a single neuron (called perceptron in the neural network community) correctly during learning can solve this task.

Next we consider a harder problem where the two challenges are real and illustrate that a simple neuron fails to learn the task. We introduce multiple layers of neurons and change the synaptic weigths between them to minimise the classification error. We show that if we have sufficiently large number of neurons, then the network can learn the correct mapping from input to output.

The message of this tutorial is that a large network of simple neurons can implement complex input-output mappings.

Linearly separable classification

For the inputs we will use two real numbers. You can think about them as pixel luminance values in a (extremely simple) two-pixel image or position of object in a 2D space. The goal is to classify the inputs into two categories, red and blue.

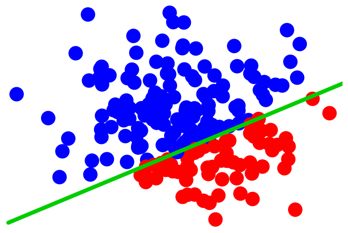
Here we will use a randomly chosen neuron, a teacher, to define the target classification for another neuron, the student. This choice gurantees that the problem is learnable.

```
source('./perceptron.R', chdir=T)
set.seed(14) # initialising random seed
n.input <- 2 # number of presynaptic neurons - the dimensionality of the problem
n.patterns <- 200 # total number of patterns

## input patterns: a matrix of 200x3. First and second column are Gaussian rundom numbers. The third is
patterns <- cbind(matrix(rnorm(n.input*n.patterns), n.patterns), rep(1, n.patterns))
## the teacher - random initializaton:
ww <- rnorm(3)
x <- seq(-3,3)
separatrix <- (-1) * (ww[1] * x  + ww[3]) / ww[2]
target <- (sign(patterns %*% ww) + 1) / 2
```

This is the teacher's classification of the inputs. It corresponds to a single line separating the 2D plane into two halves: a red and a blue half. Importantly, the weights of a single neuron always correspond to such a linear decision boundary in the N-dimensional space of the inputs.

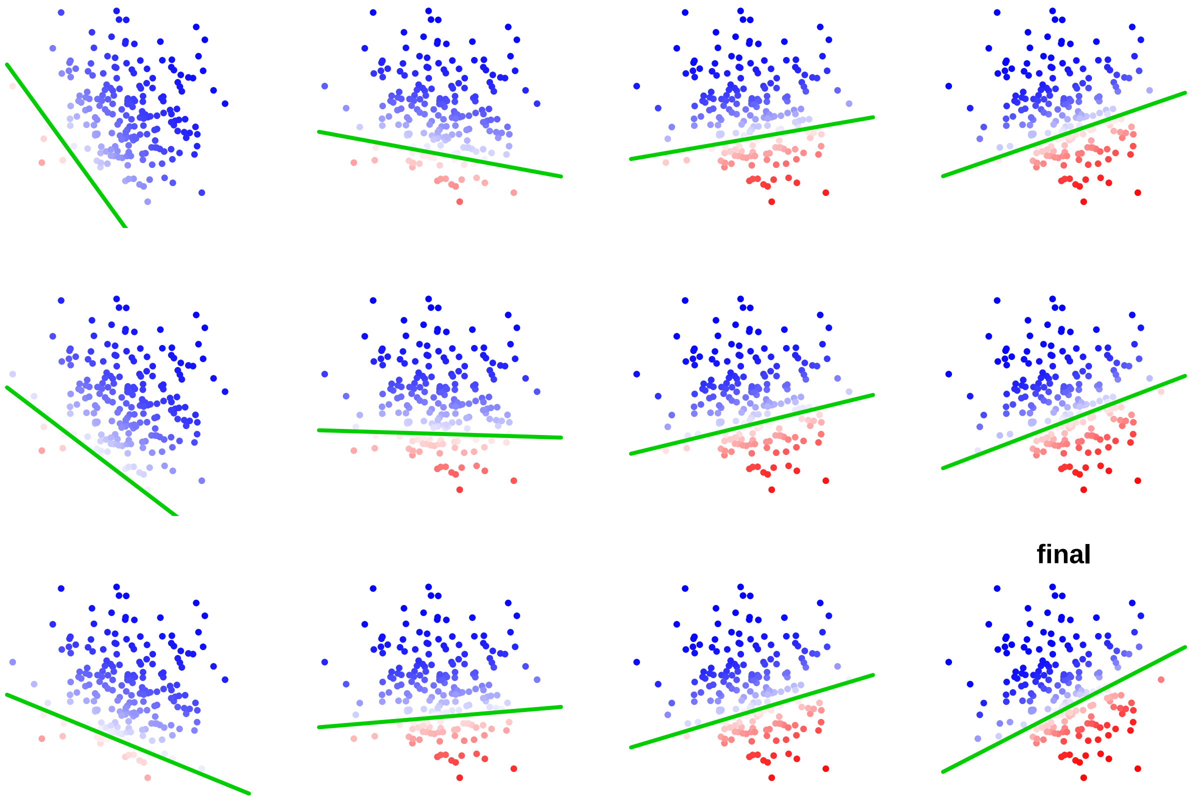
```
#plot(patterns[,1], patterns[,2], pch=16, col=blueRed2(target), xlab="x", ylab="y", cex=0.7)
#lines(x, separatrix, col=3, lwd=2)
```

```r
plot(patterns[,1], patterns[,2], pch=16, col=blueRed2(target), xlab="", ylab="", cex=1, axes=F)
lines(x, separatrix, col=3, lwd=2)
```



Now the student is initialised randomly and aims to learn the correact classification of the inputs, i.e., the position and the orientation of the boundary between the red and the blue-coloured region, represented by the green line. The following figure shows the progress of the learning process. The color code indicates the uncertainty of the classification - darker colours are more certain, whereas white means that the student has no idea.

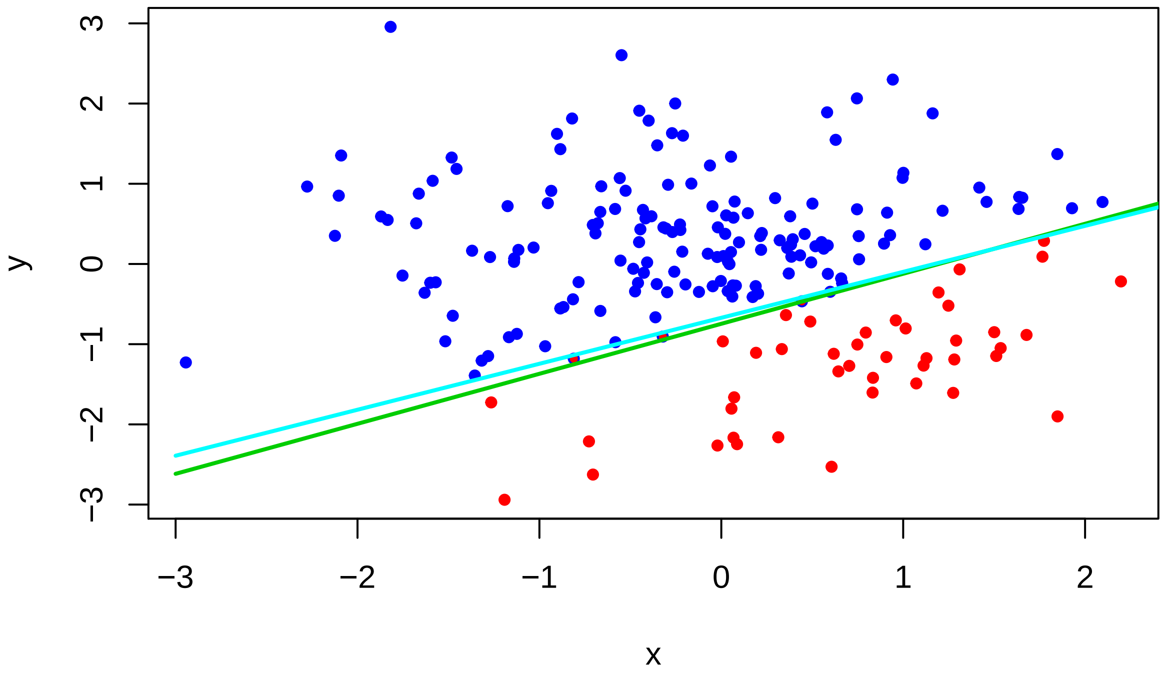
```r
set.seed(21)
w.init <- rnorm(3)
learn_rate <- 10 # learning rate
Niter = 20
w.final <- train.perceptron(w.init, patterns, target, Niter, learn_rate=learn_rate, graphics=T)
```



The misclassification rate is 2% - but only a few cases are misclassified near the boundary.

We generate a new dataset and compare the performance of the student and the teacher on this new dataset. We call the old dataset, used for training the student *training data* and this new dataset, not seen previously be the student *test data*. Each point has two colors: the fill and the line. The fill shows the prediction of the student and the line shows the teacher. The green line represents the teacher's classification and the cyan shows the student's classification.

```r
patterns.test <- cbind(matrix(rnorm(n.input*n.patterns), n.patterns), rep(1, n.patterns))
target.test <- (sign(patterns.test %*% ww) + 1) / 2
target.pred <- (sign(patterns.test %*% w.final$w) + 1) / 2
separatrix.pred <- (-1) * (w.final$w[1] * x  + w.final$w[3]) / w.final$w[2]

plot(patterns.test[,1], patterns.test[,2], pch=21, col=blueRed2(target.test), bg=blueRed2(target.pred),
lines(x, separatrix, col=3, lwd=2)
lines(x, separatrix.pred, col=5, lwd=2)
```



##

Homework

1. Plot the classification accuracy in the function of time. How many trials are required to achieve perfect classification? The classification accuracy can be found in the `w.final$miss` variable. [2 points]
2. How the learning speed depends on the learning rate (try different values from 1/10 to 100 000)? Is it a good idea to increase the learning rate? Is there a best learning rate? [5 points]
3. What happens if we decrease the training data, `n.patterns` from 200 to 20 or to 5? Will it be easier to learn? Compare the performance both in training data and in test data! [5 points]

Random target pattern

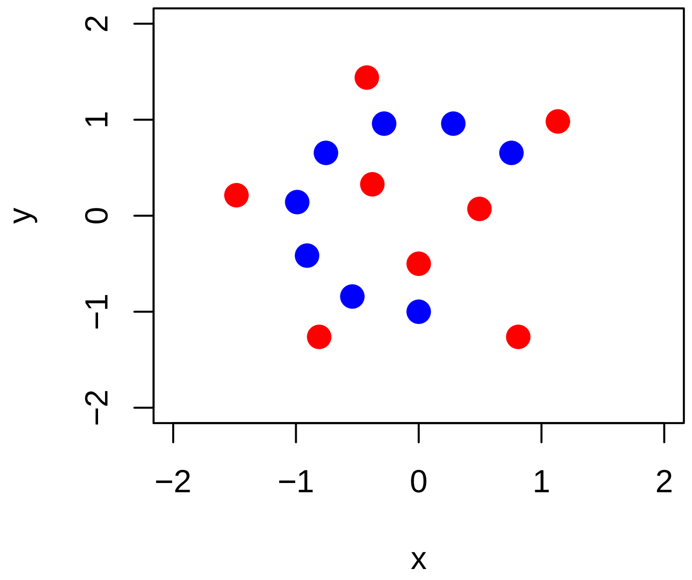
Now, instead of learning a classification that the teacher neuron already solved, let's try to do something new and more exciting: Try to learn a mapping from 14 input patterns to random outcomes.

```r
set.seed(29)
n.patterns <- 16
sp <- 0.5

#patterns <- cbind(matrix(rnorm(n.input*n.patterns), n.patterns), rep(1, n.patterns))
#target <- rep(0, n.patterns)
#target[sample(1:14, 7)] <- 1

xx <- seq(-pi, pi, length=12)
```

```
patterns <- cbind(c(sin(xx[1:8]), 0.5*sin(xx[c(1,5,9)]), 1.5*sin(xx[c(2, 4, 6, 8, 11)])), c(cos(xx[1:8]
target <- rep(0, n.patterns)
target[1:8] <- 1
```

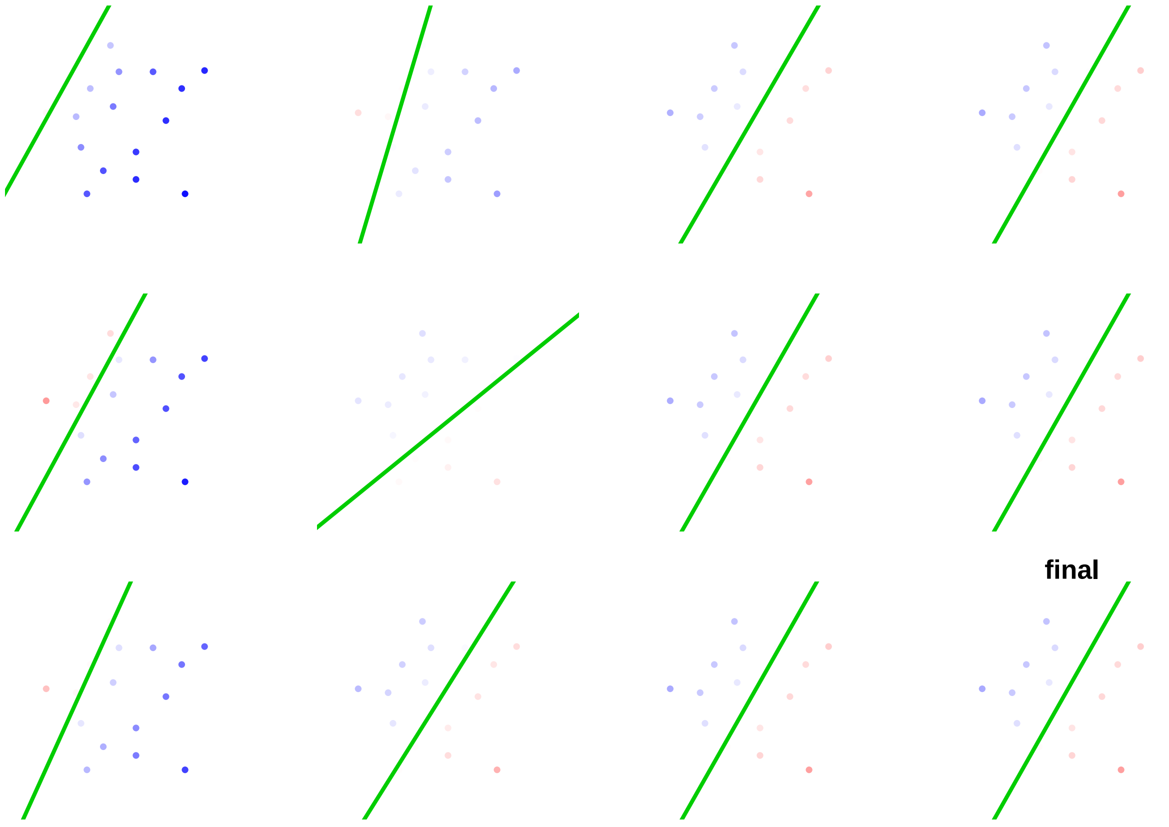
Here I plot the target mapping. It is clearly impossible to separate the red dots from the blue using a straight line - so most likely the simple neuron will not be able to learn the task.

```
m <- ceiling(max(abs(patterns)))
plot(patterns[,1], patterns[,2], pch=16, col=blueRed2(target), xlab="x", ylab="y", cex=1.7, xlim=c(-m, 
```



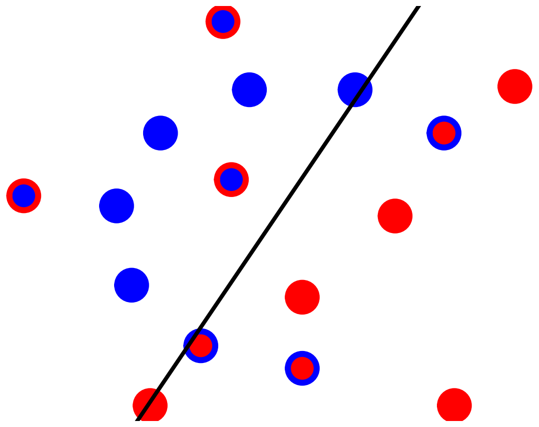
Let's observe how the student neuron will try to implement the mapping:

```
set.seed(20)
w.init <- rnorm(3)
w.final <- train.perceptron(w.init, patterns, target, 20, learn_rate=5*learn_rate, graphics=T)
```



The misclassification rate is 38%. It is not that bad after all, but there are still 6 patterns misclassified.

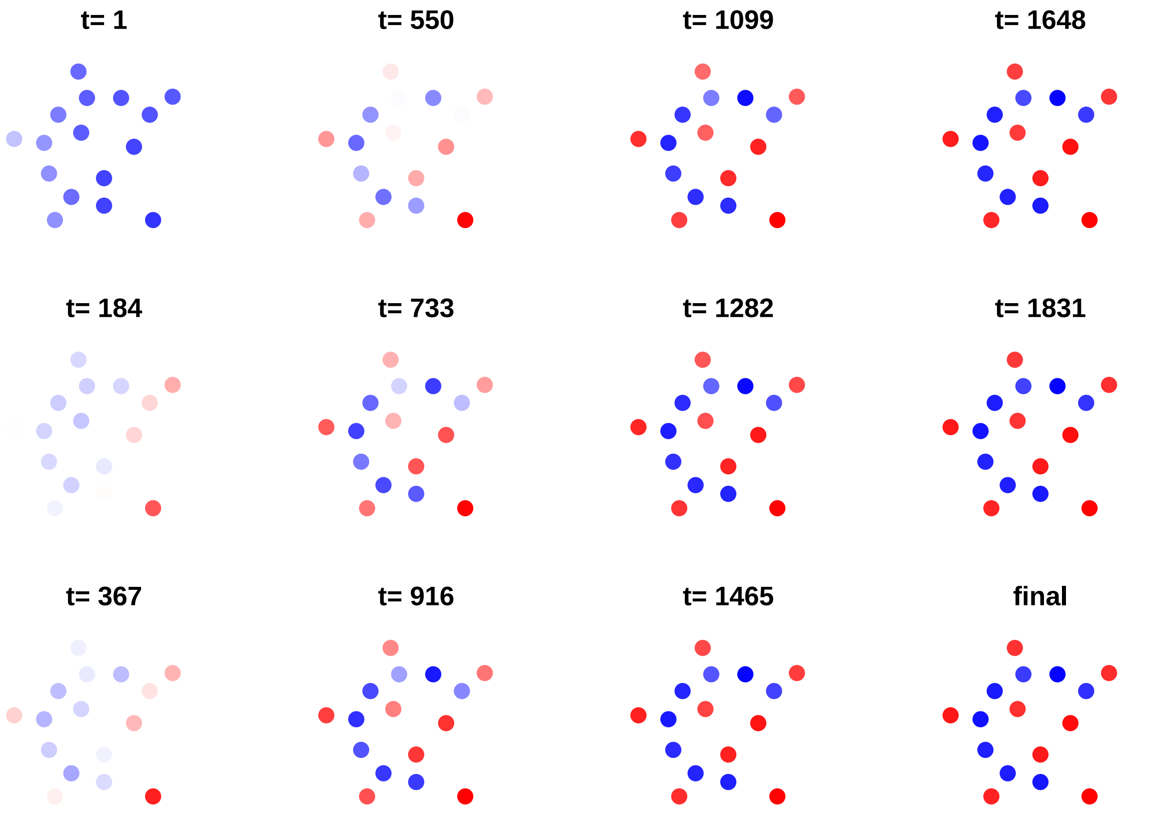
```r
target.pred <- (sign(patterns %*% w.final$w) + 1) / 2
separatrix.pred <- (-1) * (w.final$w[1] * x  + w.final$w[3]) / w.final$w[2]

plot(patterns[,1], patterns[,2], pch=21, col=blueRed2(target), bg=blueRed2(target.pred), xlab="", ylab=
#lines(x, separatrix, col=3, lwd=2)
lines(x, separatrix.pred, col=1, lwd=2)
```



Solving the non-separable problem - multiple layers

Now we will introduce 10 additional neurons. These neurons are not participating in the output, and their activity is not constrained by the input - they consist of an additional, hidden layer between the input and the output unit. This is why we call this network as a two-layer neural network, or perceptron in this case. These new neurons provide the network with sufficient flexibility to learn more complex problems.

```r
n.hidden <- 10 # number of hidden neurons
set.seed(114)
w.in <- matrix(rnorm(3*n.hidden), 3)
w.out <- matrix(rnorm(n.hidden+1), n.hidden+1)
w.final.2L <- train.perceptron.2L(w.in, w.out, patterns, target, Tmax=2000, e=10, graphics=T, verbose=F
```

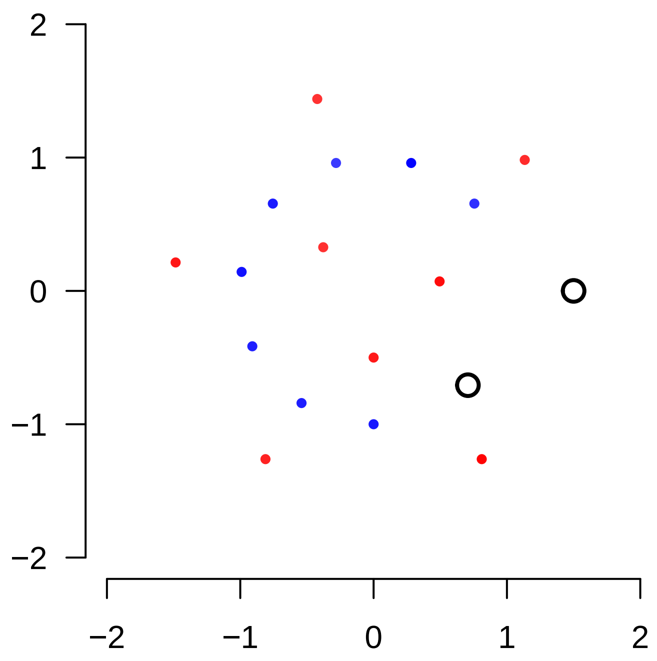


Indeed, at the end all points are correctly classified.

Homework

So far we have evaluated the performance of the 2-layer perceptron only on training data. Now, instead of generating a full test dataset, we will only add two test points, and study how the perceptron, trained on the 14 points of the training set classifies these new points.

```r
par(mfcol=c(1,1))
test.patterns <- cbind(matrix(runif(2*1000, -2, 2), 1000), rep(1, 1000))
v.u <- eval.perceptron.2L(w.final.2L$w.in, w.final.2L$w.out, patterns, T)
points(sqrt(2)/2,-sqrt(2)/2, pch=21, col=1, cex=1.5, lwd=2)
points(1.5,0, pch=21, col=1, cex=1.5, lwd=2)
axis(1); axis(2, las=2)
```

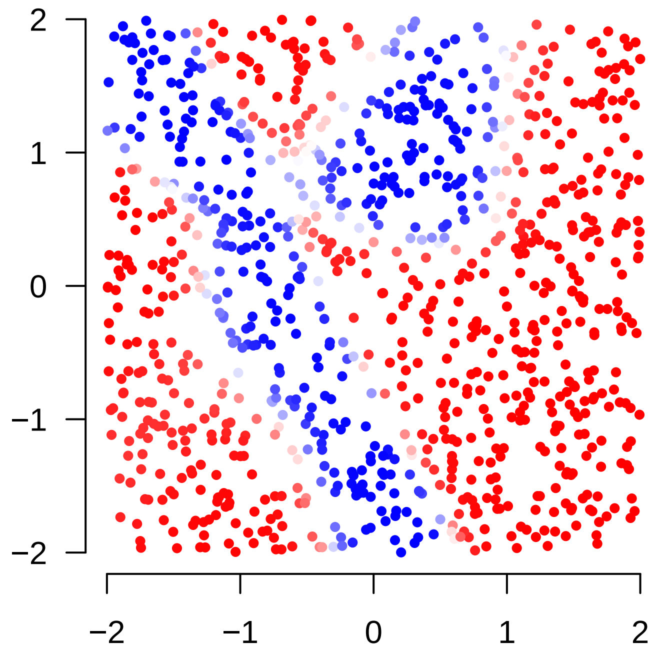


Consider a new point in the above classification task, shown now in black. These are two novel, test points that the newtork has never seen before. Now the question is, what the network believes - without retraining it - about the color of these novel points?

- Would you colour it blue or red? [1 point]
- What is the classification of the 2-layer network? Try to use different initial conditions or different number of hidden neurons (between e.g., 3 and 20), but always train the network using the original 14 points, and not the two new points! The new points are only added when you evaluate the network! [4 points]
- What is the problem here? What would be the solution to this problem? How the brain can possibly solve this problem? [3 points]

To visualise the classification boundaries, you can generate a large set of test points and evaluate the network's predictions:

```r
par(mfcol=c(1,1))
test.patterns <- cbind(matrix(runif(2*1000, -2, 2), 1000), rep(1, 1000))
v.u <- eval.perceptron.2L(w.final.2L$w.in, w.final.2L$w.out, test.patterns, T)
axis(1); axis(2, las=2)
```


2
1
0
−1
−2
−2
−1
0
1
2