Lecture notes & assignment 1 – Perceptron

Alexander Mathis, Ashesh Dhawale

January 26, 2016

1 Background: Supervised learning

A supervised learning problem deals with a situation, where one gets a dataset $\{x_j\}_{1\leq j\leq N}$ and a teacher signal (y_j) (or desired output; the supervisory signal). The learning system shall "learn" to correctly assign the desired output y_j signal to a piece of data x_j. The learned mapping can then be used to predict labels for data the system has never seen before. The desired output could be real valued (or a vector itself), but we will focus on binary supervisory signals for now and denote them by $+1$ and -1.

For instance, x_j could be a picture of a handwritten digit – a vector of discretized luminance values; see Fig. 2. You can easily read each digit. Teaching a machine (computer) how to do this is not as straight forward. We will see that a perceptron can easily be trained to tell the difference between a 0 and a 1.

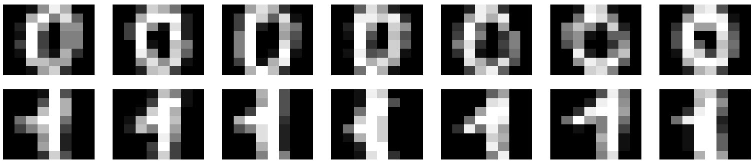

Figure 1: Discretized, grayscale pictures of handwritten digits. Top row: Various Zeros. Bottom row: Various Ones. Source: Scikit-learn

These example images have been taken from the *digits toy dataset* in Scikit-learn.[1] A similar, more general benchmark dataset is the MNIST database of handwritten digits which contains $60,000$ examples training examples and $10,000$ test examples of 28×28 pixel images. Refer to Yann LeCun's MNIST website[2] for more details. Artificial neuronal networks are the best known algorithms for this problem.[3]

2 Background: The perceptron

The perceptron is a powerful and simple model (for supervised learning). It can be traced back at least to the seminal paper by McCulloch and Pitts in 1943, in which they showed that networks based on such neurons "every [Turing] computable algorithm can be implemented."[4]

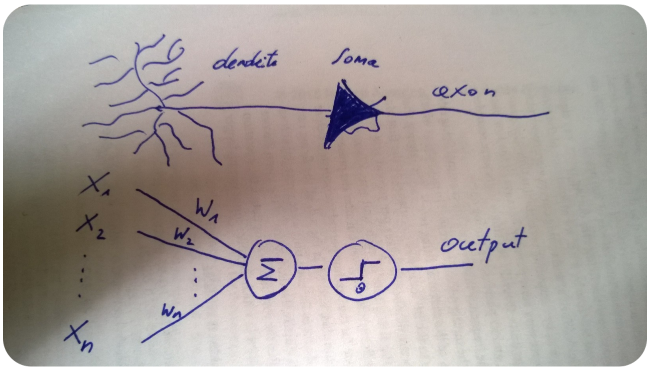

Figure 2: A perceptron is a simplified neuron model that responds with $+1$ when $w \cdot x \geq \theta$, -1 otherwise.

[1] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011. http://scikit-learn.org/

[2] http://yann.lecun.com/exdb/mnist/

[3] See D.C. Ciresan, U. Meier, L.M. Gambardella and J. Schmidthuber (2010) "Deep, Big, Simple Neural Nets for Handwritten Digit Recognition." Neural Computation, Vol. 22, No. 12, Pages 3207-3220. as well as alternative approaches on http://yann.lecun.com/exdb/mnist/.

[4] W.S. McCulloch and W.H. Pitts, (1943) "A logical calculus of the ideas immanent in nervous activity," Bulletin of Mathematical Biophysics, Vol. 5, pp. 115-133.

A perceptron has two parameters the synaptic weight $w = (w_1, \dots, w_i, \dots w_n)$, where the i-th weight connects to input x_i, and threshold θ. For a given input vector x it outputs $+1$ when $\sum_i w_i x_i \geq \theta$, and -1 otherwise. We write $o(x)$ for the output to input vector x.

Note that the weights and the threshold of the perceptron define a hyperplane (typically of dimension $n-1$ in $\mathbb{R}^n$), which partitions all inputs $x \in \mathbb{R}^n$ into two regions. The region with $w \cdot x \geq \theta$ is labeled as $+1$, the other region as -1.

A supervised learning problem $\{(x^{(j)}, y^{(j)}\}_{1,\dots,N}$ is called *linearly separable* when there are weights w and a threshold θ such that for all $j \in \{1, \dots, N\}$ the output of the perceptron weights w and a threshold θ satisfies: $o(x^{(j)}) = y^{(j)}$.

But how should one pick weights for a learning problem?

2.1 Perceptron learning rule

Assume that you are given a pair $\left(x^{(j)}, y^{(j)}\right)$ and a perceptron with weights w and threshold θ. Either $o(x^{(j)}) = y^{(j)}$, then neither w nor θ have to be changed, or $o(x^{(j)}) \neq y^{(j)}$. If $o(x^{(j)}) = -1$ when $y^{(j)} = 1$ then $wx^{(j)} - \theta$ should be increased. Conversely when $o(x^{(j)}) = 1$ when $y^{(j)} = -1$ then $wx^{(j)} - \theta$ should be decreased.

The perceptron plasticity rule performs such a change:

$$w \quad \mapsto w + \alpha/2 \left(y^{(j)} - o(x^{(j)})\right) x^{(j)} \tag{1}$$

$$\theta \quad \mapsto \theta - \alpha/2 \left(y^{(j)} - o(x^{(j)})\right). \tag{2}$$

Here the arrow $\mapsto$ stands for the update of the weight and threshold to "learn the j-th pair of data". The parameter α is the learning rate and determines how "fast" new information is incorporated into the parameters. Note that the updated parameters, denoted by a star, satisfy:

$$\begin{aligned} w^* x^{(j)} - \theta^* &= (w + \alpha/2 \left(y^{(j)} - o(x^{(j)})\right) x^{(j)}) \cdot x^{(j)} - (\theta - \alpha/2 \left(y^{(j)} - o(x^{(j)})\right)) = \\ &= w \cdot x^{(j)} - \theta + \alpha/2 \left(y^{(j)} - o(x^{(j)})\right) \underbrace{(x^{(j)} \cdot x^{(j)} + 1)}_{>0}. \end{aligned}$$

Thus, the response of the updated perceptron is the same as the non-updated one plus an increase or decrease in the intended direction (as discussed above). The parameters are only altered when there is a discrepancy between $y^{(j)}$ and $o(x^{(j)})$; if the output is correct neither the weights nor the threshold change.

To learn a set of input pairs $\left(x^{(j)}, y^{(j)}\right)_j$ one applies the learning rule repeatedly to each pair either sequentially or in an arbitrary order. Note that the learning rule neither necessary implies that after its application $x^{(j)}$ will be correctly classified, nor that patterns that had already been "learned" will remain "learned". However, an important result states that: **For linearly separable learning problems the perceptron learning rule will find parameters that solve the problem.**[5]

3 Boolean formulas

A truth function is a mapping from a set of truth values to truth values. The domain and range (in classical logic) are the binary values {truth (T), false (F)}. From one binary variable there are only four truth functions, i.e. 2^2 (tautology, contradiction, identity, and negation). Any mapping can be summarized by truth tables:

tautology (↑)

in	out
T	T
F	T

contradiction (↓)

in	out
T	F
F	F

identity

in	out
T	T
F	F

negation ¬

in	out
T	F
F	T

Perhaps more interesting are the mappings from two binary variables; here are a few of the binary truth functions:

tautology

A	B	A ↑ B
T	T	T
T	F	T
F	T	T
F	F	T

conjunction

A	B	A ∧ B
T	T	T
T	F	F
F	T	F
F	F	F

implication

A	B	A → B
T	T	T
T	F	F
F	T	T
F	F	T

exclusive disjunction (XOR)

A	B	A ↮B
T	T	F
T	F	T
F	T	T
F	F	F

[5] Refer to Dayan & Abbott (2001) "Theoretical Neuroscience" for the proof.

With these functions, one can build syntactically sophisticated sentences, like $((A \wedge B) \to C) \leftrightarrow (A \to (B \to C))$ or $(\neg B \to \neg A) \to (A \to B)$ which are both tautologies and thus correct ways to reason. We have already encountered Boolean expressions as logical operators when programming in MATLAB. As logical gates they are also fundamental building blocks in digital circuits and allow the implementation of algorithms.

Exercises 1 – Supervised learning & Perceptron

due date February 1 23:59, 2016

Of lines and half spaces

1. Differentially shade the regions of the plane, where the perceptron responds with +1 and -1 for $w = (1, 1)$ and $\theta = 0$.
2. Do the same for $w = (2, 1)$ and $\theta = 3$.

Depict this either by hand or in MATLAB.

Learning with perceptrons

1. If one interprets T (true) as $+1$ and F (false) as -1, one can ask if the perceptron can learn to carry out a particular binary truth function. Thus, first find all binary truth functions. Then determine which ones can be learned with by the perceptron? - report the Perceptron parameters you find.
2. Propose generalizations of the perceptron that could learn the binary truth functions that are not learnable by the perceptron?
3. Load `X_handwrittendigits.mat` and `y_handwrittendigits.mat`. You will get X, a 100×64 matrix of 100 images (that have been converted to 1D) and y the teacher signal. Use the first 99 examples to learn parameters for the perceptron. Try it out on the last item, does it tell you the right digit?
4. Calculate a learning curve. Randomly split the 100 samples into a training set of p samples and a test set of 100-p samples. You can calculate the test error (as the ratio of correct predictions after training on p samples). For a given $p \in \{5, 10, 25, 50, 90\}$ calculate this error 20 times (for different random splits) and average the values to get ϵ_p (let's call this the sample average test error). Plot ϵ_p versus p.
5. For the final exercise fix $p = 90$. How many pixels are "necessary" for a perceptron to be able to discriminate the two stimuli in the imaging data set with more than 90% accuracy?

Submit your MATLAB code as well as a write-up of your solutions ($\leq$ 2 pages) by email before 23:59 on February 1st. Your work will be graded based on readability, and functionality. Delayed submission will affect the grade.