Neural Networks
Lecture Notes
Perceptrons

Marcel van Gerven

1 Learning Goals

After studying the lecture material you should:

1. know how to compute the output of a perceptron.
2. know what is meant by a decision boundary.
3. be able to write down the steps of the PCP.
4. understand the proof of the PCP (you don't need to memorize the whole proof)
5. illustrate what kind of problems can and cannot be solved by a perceptron.
6. understand how gradient descent works by using partial derivatives of an loss function.
7. be able to explain the components of:

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = \sum_n (y^n - t^n) \frac{\partial f(a^n)}{\partial a^n} x_i^n$$

and how it is used to update the weights using the delta rule ($\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} E$).

2 Notes

2.1 The Perceptron

The perceptron was introduced in the classical paper by Rosenblatt [Ros58]. Consider a neural network

$$y = f(\mathbf{w}^T \mathbf{x} - b)$$

where f is an *activation function*, $\mathbf{x}$ is the *input vector*, $\mathbf{w}$ is a *weight vector*, b is a bias, and y is the binary output. Note that we can create networks with multiple outputs by just considering a collection of perceptrons.

In the classical formulation of the perceptron, the activation function is chosen to be the threshold function (linear threshold unit):

$$f(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

The bias changes the position and the weights change the orientation of the *decision boundary*, which is defined as that set of points for which $a = \mathbf{w}^T \mathbf{x}$. Note that we can ignore the bias b by using a constant input. Hence, we may write $y = f(\mathbf{w}^T \mathbf{x})$.

2.2 Perceptron learning

The problem of learning in neural networks is simply the problem of finding a set of connection strengths (weights) which allow the network to carry out a desired computation.

In *supervised learning*, of which perceptron learning is an example, the network is provided with a set of example input/output pairs (a training set). The training set is used to modify its connections in order to approximate the function from which the input/output pairs have been drawn. The network is then tested for its ability to generalize to new data for which the inputs are given but the outputs are unknown.

The perceptron is important since it was one of the first neural network for which a (supervised) learning rule has been devised. This allows (for example) learning a logical function simply by presenting the network with Boolean inputs and outputs. In general it is used to classify real-valued input vectors to binary outputs 0 or 1.

2.3 The perceptron convergence procedure

The *perceptron convergence procedure* (PCP) is the classical way of training a perceptron. It is given by the following algorithm:

- Assume training data $(\mathbf{x}^n, t^n)$ for $n = 1, \ldots, N$.
- Pick training cases using any policy that ensures that every training case will keep getting picked.
  - If the output unit incorrectly outputs $y = 0$, add the input vector to the weight vector.
  - If the output unit incorrectly outputs $y = 1$, subtract the input vector from the weight vector.
  - If the output unit is correct, leave its weights alone.

This algorithm can be written more compactly as the following equation:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta(t^n - y^n)\mathbf{x}^n$$

where we introduce the notion of a *learning rate* η. In other words, $\mathbf{w}$ gets the previous weight plus some term which depends on the current training case.

2.4 Perceptron convergence theorem

An important result is the *perceptron convergence theorem*, which states that

> *Any solution to a problem that can be solved by a perceptron can be found in a finite number of steps using the perceptron convergence procedure.*

More formally, the theorem is defined as follows.

Theorem 2.1. *[Blo62, Nov62] Assume that there exists some optimal weight vector* $\mathbf{u}$ *such that* $||\mathbf{u}|| = 1$*, and some* $\gamma > 0$ *such that for all* $n = 1, \ldots, N$*:*

$$\mathbf{u}^T \mathbf{x}^n t^n \geq \gamma$$

Assume in addition that for all $n = 1, \ldots, N$*,* $||\mathbf{x}^n|| \leq R$*. Then the PCP makes at most*

$$\frac{R^2}{\gamma^2}$$

errors.

2.5 Proof of the perceptron convergence theorem

We will prove that the perceptron convergence procedure converges in a finite number of steps k. We do this by deriving a lower and an upper bound on $||\mathbf{w}||$. We show that both bounds imply that k is finite. First, recall that $y = f(\mathbf{w}^T\mathbf{x})$ with f the threshold function. If a solution exists (which is what we assume at this point), then there must be a weight vector $\mathbf{u}$ such that

$$f(\mathbf{u}^T\mathbf{x}^n) = t^n$$

for $n = 1, \ldots, N$. This can be rewritten as

$$\mathbf{u}^T\mathbf{x}^n\tilde{t}^n \geq \gamma$$

for some $\gamma \geq 0$ and introducing $\tilde{t}^n = 2t^n - 1$.[1] Note further that if $\mathbf{u}$ is an optimal weight vector, then so is $\alpha\mathbf{u}$ for $\alpha > 0$. This implies that we can assume $\mathbf{u}$ to be a unit vector with $||\mathbf{u}|| = 1$.

Derivation of lower bound

According to the weight update equation, using $\eta = 1$ and notation $\tilde{t}_n$, at each step τ of the PCP in which we have a weight update due to a misclassified pattern we have

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} + \mathbf{x}^{n(\tau)}\tilde{t}^{n(\tau)}$$

for *misclassified* training patterns $\mathbf{x}^{n(\tau)}$. Suppose that, after running the algorithm for some time, starting at $\mathbf{w}^{(0)} = \mathbf{0}$, the number of times that each vector $\mathbf{x}^n$ has been presented and misclassified is T_n.Then the weight vector at this point will be given by

$$\mathbf{w} = \sum_n T_n\mathbf{x}^n\tilde{t}^n$$

Let $\mathbf{u}$ denote the optimal weight vector. Using the last equation, we take the inner product between $\mathbf{u}$ and $\mathbf{w}$ to obtain

$$\mathbf{u}^T\mathbf{w} = \sum_n T_n\mathbf{u}^T\mathbf{x}^n\tilde{t}^n$$

Since we know that $\mathbf{u}^T\mathbf{x}^n\tilde{t}^n \geq \gamma$ for all n, we can also write

$$\mathbf{u}^T\mathbf{w} \geq \sum_n T_n\gamma = k\gamma$$

where $k = \sum_n T_n$ is the total number of weight updates.

We now rewrite this as a bound on $||\mathbf{w}||$. From the Cauchy-Schwartz inequality, which states that $||\mathbf{x}||^2||\mathbf{y}||^2 \geq \left(\mathbf{x}^T\mathbf{y}\right)^2$ it follows that

$$||\mathbf{u}||^2||\mathbf{w}||^2 \geq \left(\mathbf{u}^T\mathbf{w}\right)^2$$

Since we just derived that $\mathbf{u}^T\mathbf{w} \geq k\gamma$ and $||\mathbf{u}|| = 1$, it follows that

$$||\mathbf{w}||^2 \geq k^2\gamma^2$$

which, by taking the square root, gives us the lower bound on $||\mathbf{w}||$:

$$||\mathbf{w}|| \geq k\gamma$$

[1] check for targets $t = 1$ and targets $t = 0$ that this holds!

Derivation of upper bound

We first show that $||\mathbf{w}^{(\tau+1)}||^2 \leq ||\mathbf{w}^{(\tau)}||^2 + ||\mathbf{x}^n||^2$. Recall that

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} + \mathbf{x}^{n(\tau)}\tilde{t}^{n(\tau)}$$

where $\mathbf{x}^{n(\tau)}$ is some misclassified pattern. Squaring the equation and using $||\mathbf{z}||^2 = \mathbf{z}^T\mathbf{z}$, we have

$$\begin{aligned} ||\mathbf{w}^{(\tau)}||^2 &= (\mathbf{w}^{(\tau-1)} + \mathbf{x}^{n(\tau)}\tilde{t}^{n(\tau)})^T(\mathbf{w}^{(\tau-1)} + \mathbf{x}^{n(\tau)}\tilde{t}^{n(\tau)}) \\ &= \mathbf{w}^{(\tau-1)}\mathbf{w}^{(\tau-1)} + (\mathbf{x}^{n(\tau)}\tilde{t}^{n(\tau)})^T\mathbf{x}^{n(\tau)}\tilde{t}^{n(\tau)} + 2(\mathbf{w}^{(\tau-1)})^T\mathbf{x}^{n(\tau)}\tilde{t}^{n(\tau)} \\ &= ||\mathbf{w}^{(\tau-1)}||^2 + ||\mathbf{x}^{n(\tau)}||^2(\tilde{t}^{n(\tau)})^2 + 2(\mathbf{w}^{(\tau-1)})^T\mathbf{x}^{n(\tau)}\tilde{t}^{n(\tau)} \end{aligned}$$

Since $\tilde{t}^n = \pm 1$, we can simplify to

$$||\mathbf{w}^{(\tau)})||^2 = ||\mathbf{w}^{(\tau-1)}||^2 + ||\mathbf{x}^{n(\tau)}||^2 + 2(\mathbf{w}^{(\tau-1)})^T\mathbf{x}^{n(\tau)}\tilde{t}^{n(\tau)}$$

Since $\mathbf{x}^{n(\tau)}$ is defined as a pattern that is misclassified by $\mathbf{w}^{(\tau-1)}$, it must hold that

$$\left(\mathbf{w}^{(\tau-1)}\right)^T \mathbf{x}^{n(\tau)}\tilde{t}^{n(\tau)} < 0\,.$$

This implies that, by removing the last component, we obtain the following inequality:

$$||\mathbf{w}^{(\tau)}||^2 \leq ||\mathbf{w}^{(\tau-1)}||^2 + ||\mathbf{x}^{n(\tau)}||^2$$

Define the change in $||\mathbf{w}||^2$ at time τ as $\Delta^{(\tau)}||\mathbf{w}||^2 = ||\mathbf{w}^{(\tau)}||^2 - ||\mathbf{w}^{(\tau-1)}||^2$. By using this definition in the inequality we obtain:

$$\Delta^{(\tau)}||\mathbf{w}||^2 \leq ||\mathbf{x}^{n(\tau)}||^2$$

Let $R = \max_n ||\mathbf{x}^n||$ denote the maximal length of the input vectors in our data set. After a total of k weight updates it must hold that

$$||\mathbf{w}||^2 = \sum_{\tau=1}^{k} \Delta^{(\tau)}||\mathbf{w}||^2 \leq kR^2$$

Completing the proof

We have derived a lower bound $||\mathbf{w}|| \geq k\gamma$ and an upper bound $||\mathbf{w}||^2 \leq kR^2$. Combining the two, we have

$$k^2\gamma^2 \leq ||\mathbf{w}||^2 \leq kR^2$$

In terms of k, we may write $k^2\gamma^2 \leq kR^2$ which simplifies to

$$k \leq \frac{R^2}{\gamma}\,.$$

This shows that the number of update steps k is finite thus proving convergence of the PCP.

2.6 The delta rule

The delta rule as developed by Widrow and Hoff [WH60] is a more general and principled approach to training supervised neural networks. Instead of showing that the weights get closer to a good set of weights, show that the actual output values get closer to the target values. The delta rule is based on the definition of an objective function and direct optimization of this function using gradient descent methods.

We start by considering a neural network with arbitrary activation function:

$$y = f(\mathbf{w}^T\mathbf{x})$$

We define a sum of squares *loss function* (a.k.a. quadratic loss) which measures the difference between a predicted output and a target output:

$$E(\mathbf{w}) = \frac{1}{2}\sum_n (y^n - t^n)^2$$

The goal is to find parameters which minimize E (for optimists: maximize the objective function $-E$). The question is how to find:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w})$$

The way to achieve this is to measure locally the change in slope of $E(\mathbf{w})$ as a function of w take a step in the direction of the steepest descent. This procedure is known as *gradient descent*.

2.6.1 Gradient descent

In a rectangular coordinate system, the *gradient* ∇f (where ∇ is the nabla symbol) is the vector field whose components are the *partial derivatives* of f:

$$\nabla f = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n}\right)$$

Gradient descent is based on the observation that if a function $f(\mathbf{x})$ is defined and differentiable in the neighbourhood of a point a, then $f(\mathbf{x})$ decreases fastest if one goes from a in the direction of the negative gradient of f at a. It follows that, if

$$\mathbf{b} = \mathbf{a} - \eta\nabla f(\mathbf{a})$$

for gamma small enough, then $f(\mathbf{a}) \geq f(\mathbf{b})$.

Gradient descent is an optimization technique. Optimization is a very important topic in artificial intelligence since it allows finding solutions to objective functions. We start with an initial guess a_0 and consider the sequence $a_0, a_1, a_2, \ldots$ such that

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma_n \nabla f(\mathbf{a}_n)$$

for $n \geq 0$. We then have

$$f(\mathbf{a}_0) \geq f(\mathbf{a}_1) \geq f(\mathbf{a}_2) \geq \cdots$$

such that the sequence converges to a (local) minimum.

2.6.2 Application of gradient descent

Minimization of an loss function (or maximization of an objective function) via gradient descent is a key principle underlying modern neural networks. The general recipe is:

1. Choose an loss function (e.g. squared loss)
2. Compute partial derivatives
3. Perform gradient descent to optimize the weights $\mathbf{w}$

Consider the squared loss function $E(\mathbf{w})$. Let i denote the index of the input variable and n the index of the example $(\mathbf{x}_n, t_n)$. We define the gradient as

$$\nabla f = \left(\frac{\partial E(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial E(\mathbf{w})}{\partial w_m}\right)$$

where the partial derivatives are given by

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = \sum_n (y^n - t^n)\frac{\partial f(a^n)}{\partial a^n} x_i^n$$

with $a^n = \mathbf{w}^T\mathbf{x}^n$ and $y^n = f(a^n)$. This gives the gradient descent update

$$\mathbf{w} \leftarrow \mathbf{w} - \gamma \nabla E$$

In case we use the quadratic loss and a monotonic activation function then the loss function is a bowl with a unique global minimum. A nice result is that, using gradient descent, we are guaranteed to find the global minimum. This is an example of a convex (rather than a non-convex) optimization problem.[2]

2.6.3 Linear activation function

In case $f(\mathbf{x}) = \mathbf{x}$ (the identity function), we have a linear activation function, which was originally used by Widrow and Hoff. In this case, it holds that

$$\frac{\partial f(a^n)}{\partial a^n} = 1$$

which implies that

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = \sum_n (y^n - t^n) x_i^n$$

A linear activation function also means that the neural network implements

$$y = \mathbf{w}^T\mathbf{x}$$

which is equivalent to linear regression. Note that linear regression is much easier to solve using least squares approach which gives rise to the following analytical solution:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

where $\mathbf{y} = (y_1, \ldots, y_n)^T$. This is important in two respects. First, an important endeavour is to show that particular neural network models are equivalent to a particular (well-known) statistical model. Secondly, this results does not hold for arbitrary activation functions.

2.7 Linear separability

Perceptrons are an example of so-called linear discriminant functions, whose decision boundary forms a hyper-plane in input space. A problem is *linearly separable* when it can be solved using a hyper-plane in input space.

The perceptron convergence theorem only holds for solvable problems. Hence, perceptrons can only solve linearly separable problems. Examples of linearly separable problems are the logical AND and the logical OR. Examples of linearly inseparable problems are the logical XOR problem, the parity problem (determining odd or even number of inputs) and connectedness problems (is a certain visual pattern connected or not - see the cover of Minsky and Papert's Perceptrons book).

It is easy to see that the XOR problem cannot be solved by a perceptron. The XOR problem should solve:

$$\begin{aligned}
1 \cdot w_1 + 1 \cdot w_2 - \theta &< 0 \\
1 \cdot w_1 + 0 \cdot w_2 - \theta &\geq 0 \\
0 \cdot w_1 + 1 \cdot w_2 - \theta &\geq 0 \\
0 \cdot w_1 + 0 \cdot w_2 - \theta &< 0
\end{aligned}$$

[2]Note that most interesting optimization problems are non-convex... A profound insight is that most of machine learning (and lots of AI) amounts to optimization (e.g. neural networks) or integration (e.g. Bayesian statistics).

which can be simplified to

$$\begin{aligned} w_1 + w_2 &< \theta \\ w_1 &\geq \theta \\ w_2 &\geq \theta \\ \theta &> 0 \end{aligned}$$

If the second and third equations hold then this immediately contradicts the first equation since θ must be positive given the fourth equation.

3 Reading material

Glance through Rosenblatt's orginal paper, which can be found here. No need to read it in full. Just browse through it to get a feeling for the state of the art at that time.

References

[Blo62] H. D. Block. The perceptron: A model for brain functioning. *Rev. Mod. Phys.*, 34(1):123–135, 1962.

[Nov62] A. B. Novikoff. On convergence proofs of perceptrons. In *Symp. Math. Theory Autom.*, pages 615—-622, 1962.

[Ros58] F Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.*, 65(6):386–408, 1958.

[WH60] B Widrow and M E Hoff. Adaptive switching circuits. In *1960 IRE WESCON Conv. Rec.*, number 4, pages 96 – 104, 1960.