Chapter 2

Perceptrons

Perceptrons are the very simplest kind of neural network, and the only kind of neural network that we understand well. The niceness of perceptrons and other linear models like logistic regression and support vector machines drove much machine learning research in the 1990s and 2000s. We discuss them here to show how the kinds of questions we are interested in have clear answers at least sometimes.

2.1 Model

Definition 2.1. A *perceptron* (Rosenblatt, 1958) (or linear classifier) is a function

$$\begin{aligned} \mathit{lin}\colon \mathbb{R}^d &\to \mathbb{R} \\ \mathbf{x} &\mapsto \mathbf{w} \cdot \mathbf{x} + b \end{aligned} \tag{2.1}$$

where the parameters to be learned are

$$\begin{aligned} \mathbf{w} &\in \mathbb{R}^d \\ b &\in \mathbb{R}. \end{aligned}$$

We will generally use mnemonic names like *lin* (for "linear") for networks. If there is more than one perceptron, we'll call them lin_1, lin_2, etc., or lin'.

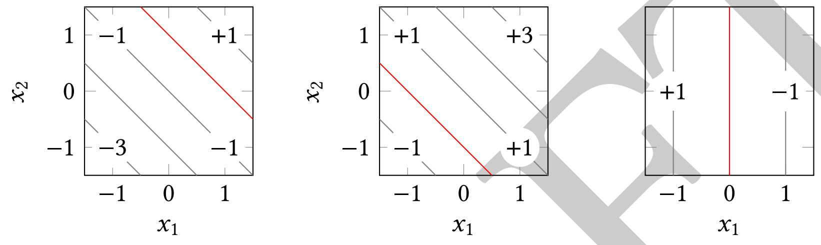
To use the perceptron as a binary classifier, we have to define what the classification criterion is. In this chapter, it will be convenient to treat $\mathit{lin}(\mathbf{x}) > 0$ as true, $\mathit{lin}(\mathbf{x}) < 0$ as false, and $\mathit{lin}(\mathbf{x}) = 0$ as neither true nor false. However, later we will use other criteria.

2.2 Expressivity

We can characterize the functions expressible by perceptrons exactly, if a little tautologically: under the above definition, they express all and only the affine functions $\mathbb{R}^d \to \mathbb{R}$.

A more interesting, and historically important, question is, what Boolean functions can they express? For the inputs of the network, let us use +1 to represent true and −1 to represent false.

Example 2.2. We can define perceptrons that compute the AND, OR, and NOT functions:



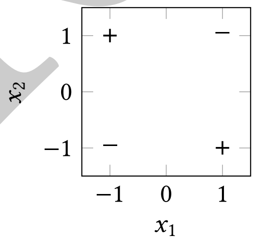
AND: $y = x_1 + x_2 - 1$ OR: $y = x_1 + x_2 + 1$ NOT: $y = -x_1$

However, not every Boolean function can be computed by perceptrons.

Theorem 2.3 (Minsky and Papert, 1969)**.** *There is no perceptron* $lin\colon \mathbb{R}^2 \to \mathbb{R}$ *that computes the XOR function, that is,*

$$\begin{aligned} lin(-1,-1) &< 0 & lin(-1,+1) &> 0 \\ lin(+1,-1) &> 0 & lin(+1,+1) &< 0. \end{aligned}$$

Proof. It's easy to see that there is no line that separates the positive and negative points:



□

Historically, this theorem had a huge impact: it led in part to the first *AI winter* in the 1970s. Although it was known that *multilayer* perceptrons (Chapter 3) could express XOR, it was not known how to train multilayer perceptrons.

2.3 Training

Neural networks are trained using stochastic gradient descent (SGD) or back-propagation (Rumelhart et al., 1986). The original algorithm for training perceptrons (Rosenblatt, 1958) turns out to just be SGD with a particular choice of loss function.

Without loss of generality, we can assume that perceptrons do not have a bias term b. This is because we can always add a new feature to each $\mathbf{x}$ that always has the value 1, and the weight for this feature is exactly equivalent to b.

Let $\mathit{lin}(\mathbf{x}; \mathbf{w})$ be a perceptron, where we've made the dependence on the parameters $\mathbf{w}$ explicit. Assume training examples $(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})$, where for each i, $\mathbf{x}^{(i)} \in \mathbb{R}^d, y^{(i)} \in \{-1, +1\}$. To recover the traditional perceptron update, we define the loss incurred by the i-th example to be

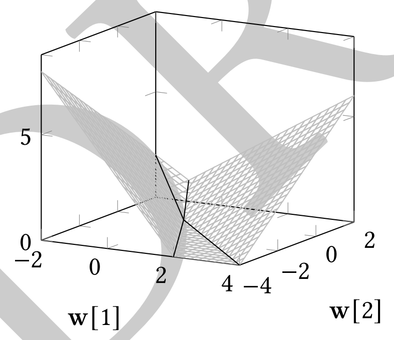
$$\mathcal{L}^{(i)}(\mathbf{w}) = \max\{0, -y^{(i)}\ \mathit{lin}(\mathbf{x}^{(i)}; \mathbf{w})\} \tag{2.2}$$

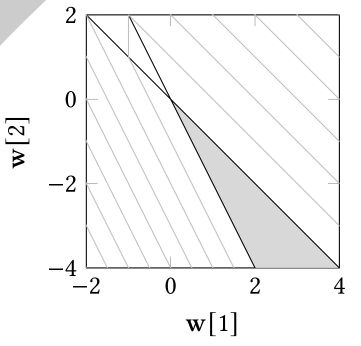
because if $\mathit{lin}(\mathbf{x}^{(i)})$ and $y^{(i)}$ have opposite signs, then $-y^{(i)}\ \mathit{lin}(\mathbf{x}^{(i)})$ is positive. (This notational convenience is the reason we are using +1 and −1 to represent truth values.) Then we want to minimize $\mathcal{L}(\mathbf{w}) = \sum_i \mathcal{L}^{(i)}(\mathbf{w})$.

Suppose our data consists of the points

$$\mathbf{x}^{(1)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad y^{(1)} = -1$$
$$\mathbf{x}^{(2)} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \qquad y^{(2)} = +1$$

Then the loss surface looks like this:





In the right-hand plot, the shaded area is where the loss is 0, and the contour lines are where the loss is $1, 2, \ldots$.

We do stochastic gradient descent on $\mathcal{L}(\mathbf{w})$. The parameters are initialized to $\mathbf{w} = \mathbf{0}$. Then for each training example, we update the parameters as follows.

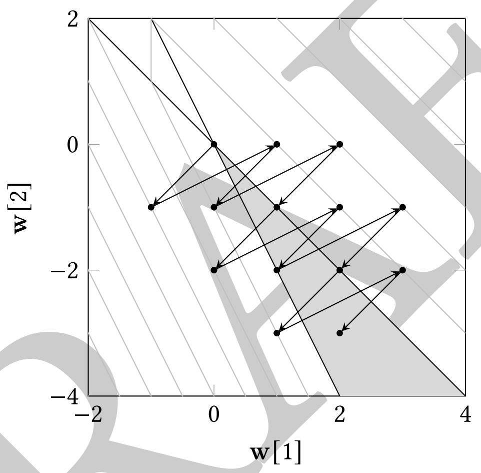
The gradient of $\mathcal{L}^{(i)}$ for a correctly classified example ($y^{(i)}\ \mathit{lin}(\mathbf{x}^{(i)}; \mathbf{w}) > 0$) is zero, so no update is made. The gradient for an incorrectly classified example ($y^{(i)}\ \mathit{lin}(\mathbf{x}^{(i)}; \mathbf{w}) < 0$) is

$$\frac{\partial \mathcal{L}^{(i)}}{\partial \mathbf{w}} = -y^{(i)}\mathbf{x}^{(i)}. \tag{2.3}$$

For the edge case $y\ \mathit{lin}(\mathbf{x}^{(i)}; \mathbf{w}) = 0$, we choose the subgradient to be the same as in the incorrectly-classified case. So the update for $y^{(i)}\ \mathit{lin}(\mathbf{x}^{(i)}; \mathbf{w}) \leq 0$ (using a constant learning rate) is

$$\mathbf{w} \leftarrow \mathbf{w} + y^{(i)}\mathbf{x}^{(i)}. \tag{2.4}$$

If the algorithm makes a pass over the training data without making a single update, it terminates.



The above plot shows the perceptron algorithm on our simple example. This run terminates, and in the next section we will show that if there is a weight vector that classifies all training examples correctly, the perceptron algorithm is guaranteed to find such a vector and terminate.

2.4 Trainability

The loss function (2.2) is convex, so (sub)gradient descent, if the learning rate is chosen correctly, is guaranteed to converge to a global minimum (Bertsekas, 2015). But here we present the proof of the convergence of the original perceptron training algorithm.

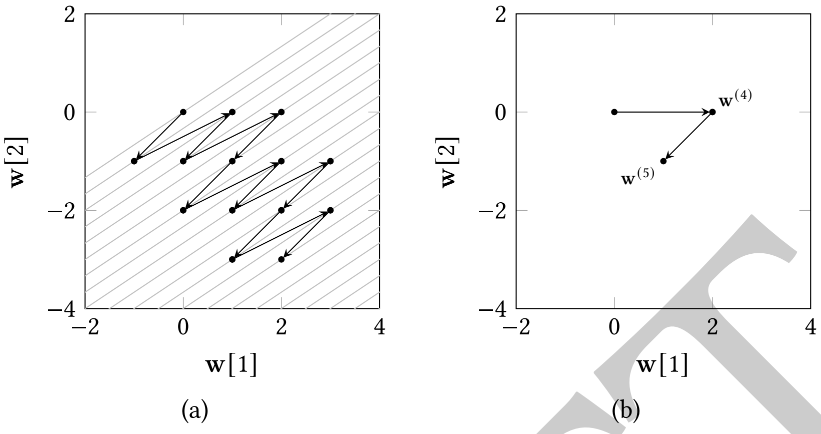


Figure 2.1: (a) At each update of the perceptron algorithm, the weights $\mathbf{w}^{(t)}$ grows in the direction of $\mathbf{w}^*$ by at least γ; the contour lines are $\mathbf{w}^* \cdot \mathbf{w} = i\gamma\|\mathbf{w}^*\|$ for $i = 0, 1, \ldots$. (b) At each update, the weight vector and the weight update form an acute angle.

Definition 2.4. Let $\mathcal{D} = (\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})$ be a set of data points, where $\mathbf{x}^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \{-1, +1\}$ for all i. We say that $\mathcal{D}$ is *linearly separable with margin* $\gamma > 0$ if there exist weights $\mathbf{w}^*$ such that, for all i,

$$y^{(i)}\mathbf{w}^* \cdot \mathbf{x}^{(i)} \geq \gamma\|\mathbf{w}^*\|. \tag{2.5}$$

Theorem 2.5 (Novikoff, 1962). *Let $\mathcal{D} = (\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})$ be a set of data points, linearly separable with margin γ. Let $R = \max_i \|\mathbf{x}^{(i)}\|$. Then the perceptron training algorithm converges after at most R^2/γ^2 steps to a perceptron that linearly separates $\mathcal{D}$.*

Proof. This proof mostly follows Freund and Schapire (1999). Let $\mathbf{w}^{(t-1)}$ be the weights before the t-th mistake, and let i_t be the example on which the perceptron made the t-th mistake. The update after the t-th mistake is

$$\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} + y^{(i_t)}\mathbf{x}^{(i_t)} \tag{2.6}$$

and because $y^{(i_t)}\mathbf{x}^{(i_t)}$ points in the same general direction as $\mathbf{w}^*$ (Eq. (2.5)), the update moves the weights more in the direction of $\mathbf{w}^*$. An example is shown in

Fig. 2.1a.

$$\begin{aligned}
\mathbf{w}^* \cdot \mathbf{w}^{(t)} &= \mathbf{w}^* \cdot \mathbf{w}^{(t-1)} + \mathbf{w}^* \cdot y^{(i_t)}\mathbf{x}^{(i_t)} && (2.7)\\
&\geq \mathbf{w}^* \cdot \mathbf{w}^{(t-1)} + \gamma\|\mathbf{w}^*\| && \text{by Eq. (2.5)} \quad (2.8)\\
&\geq \mathbf{w}^* \cdot \mathbf{w}^{(t-2)} + \gamma\|\mathbf{w}^*\| + \gamma\|\mathbf{w}^*\| && \text{repeating above reasoning} \quad (2.9)\\
&\vdots\\
&\geq t\gamma\|\mathbf{w}^*\|. && (2.10)
\end{aligned}$$

This implies that $\|\mathbf{w}^{(t)}\|$ must also be at least $t\gamma$:.

$$\begin{aligned}
\|\mathbf{w}^*\|\,\|\mathbf{w}^{(t)}\| &\geq t\gamma\|\mathbf{w}^*\| && \text{Cauchy–Schwarz} \quad (2.11)\\
\|\mathbf{w}^{(t)}\| &\geq t\gamma. && (2.12)
\end{aligned}$$

On the other hand, because we only perform an update (Eq. (2.6)) after a mistake, it must be that $y^{(i_t)}\mathbf{x}^{(i_t)}$ points in the opposite general direction from $\mathbf{w}^{(t)}$, that is,

$$y^{(i_t)}\mathbf{x}^{(i_t)} \cdot \mathbf{w}^{(t)} \leq 0. \quad (2.13)$$

This gives an upper bound on the weight norm:

$$\begin{aligned}
\|\mathbf{w}^{(t)}\|^2 &= \|\mathbf{w}^{(t-1)} + y^{(i_t)}\mathbf{x}^{(i_t)}\|^2 && (2.14)\\
&= \|\mathbf{w}^{(t-1)}\|^2 + 2\underbrace{\mathbf{w}^{(t-1)} \cdot y^{(i_t)}\mathbf{x}^{(i_t)}}_{\leq 0 \text{ by Eq. (2.13)}} + \|y^{(i_t)}\mathbf{x}^{(i_t)}\|^2 && (2.15)\\
&\leq \|\mathbf{w}^{(t-1)}\|^2 + R^2 && (2.16)\\
&\leq \|\mathbf{w}^{(t-2)}\|^2 + R^2 + R^2 && (2.17)\\
&\vdots\\
&\leq tR^2. && (2.18)
\end{aligned}$$

Combining Eqs. (2.12) and (2.18), we get

$$\begin{aligned}
(t\gamma)^2 &\leq tR^2 && (2.19)\\
t &\leq R^2/\gamma^2. && (2.20)
\end{aligned}$$

In other words, there is a finite number of mistakes, so the perceptron eventually linearly separates $\mathcal{D}$. □

There's also a mistake bound for the non-separable case (Freund and Schapire, 1999), though the bound has to depend on the number of training steps.

2.5 Generalization

Finally we turn to generalization ability: after being trained on some training data, how well will a perceptron classify test examples, which it (in general) has never seen before? To study this question, we need to think of training examples and test examples as random samples from some probability distribution. We can't guarantee that the perceptron will get a test example right, but we can bound the probability of a mistake.

Theorem 2.6 (Vapnik and Chervonenkis, 1974)**.** *Let $\mathcal{D}$ be a set of examples (possibly with duplicates). Fix $N > 0$. Let*

$$\begin{aligned} D^{(1)} &= (\mathbf{x}^{(1)}, y^{(1)}) \\ &\vdots \\ D^{(N)} &= (\mathbf{x}^{(N)}, y^{(N)}) \\ D^{(N+1)} &= (\mathbf{x}^{(N+1)}, y^{(N+1)}) \end{aligned}$$

be examples sampled uniformly at random from $\mathcal{D}$. Let $R = \max_i \|\mathbf{x}^{(i)}\|$, and assume that the examples are linearly separable with margin γ. If a perceptron is trained to convergence on $D^{(1)}, \ldots, D^{(N)}$ to obtain weights $\mathbf{w}$, the probability that it will make a mistake on $D^{(N+1)}$ is

$$\mathbb{E}\left[\mathbb{I}\left[y^{(N+1)}\mathbf{w} \cdot \mathbf{x}^{(N+1)} \le 0\right]\right] \le \frac{1}{N+1}\mathbb{E}\left[\frac{R^2}{\gamma^2}\right].$$

where the expectations are over the choice of $D^{(1)}, \ldots, D^{(N+1)}$.

The original proof is in Russian, and the following is from Hardt and Recht (2022, Chapter 3, Theorem 2).

Proof. Because the examples are independent and identically distributed, it doesn't make any difference if the training and test data are chosen by sampling N training examples and 1 test example, versus sampling $(N+1)$ examples and choosing one uniformly at random to be the the test example.

Let $\mathbf{w}^{(\neg i)}$ be the weights obtained by training on all examples except $D^{(i)}$ to convergence.

$$\begin{aligned}\mathbb{E}\left[\mathbb{I}\left[y^{(N+1)}\mathbf{w}^{(\neg(N+1))}\cdot\mathbf{x}^{(N+1)}\le 0\right]\right] &= \mathbb{E}\left[\sum_{i=1}^{N+1}\frac{1}{N+1}\mathbb{I}\left[y^{(i)}\mathbf{w}^{(\neg i)}\cdot\mathbf{x}^{(i)}\le 0\right]\right]\\ &= \frac{1}{N+1}\mathbb{E}\underbrace{\left[\sum_{i=1}^{N+1}\mathbb{I}\left[y^{(i)}\mathbf{w}^{(\neg i)}\cdot\mathbf{x}^{(i)}\le 0\right]\right]}_{(*)}.\end{aligned} \tag{2.21}$$

Let's pause this for a moment and think about training the perceptron on all $(N+1)$ examples. Let I be the set of indices i such that the perceptron makes a mistake on $D^{(i)}$ at some point. We know that it makes at most R^2/γ^2 mistakes, so $|I| \le R^2/\gamma^2$. For any $i \notin I$, the perceptron never makes a mistake on $D^{(i)}$ and therefore never updates its weights on $D^{(i)}$, so we can remove $D^{(i)}$ from the training data without any effect. Moreover, the converged perceptron classifies $D^{(i)}$ correctly: $y^{(i)}\mathbf{w}^{(\neg i)}\cdot\mathbf{x}^{(i)} > 0$.

Now we can return to $(*)$. If $i \notin I$, then when $D^{(i)}$ is held out as a test example, it is correctly classified. On the other hand, if $i \in I$, then when $D^{(i)}$ is held out as a test example, it might be incorrectly classified. Thus, $(*)$ is no more than the mistake bound for the perceptron trained on all $(N+1)$ examples, and we have

$$\mathbb{E}\left[\mathbb{I}\left[y^{(N+1)}\mathbf{w}^{(\neg(N+1))}\cdot\mathbf{x}^{(N+1)}\le 0\right]\right] \le \frac{1}{N+1}\mathbb{E}\left[\frac{R^2}{\gamma^2}\right]. \tag{2.22}$$

□

Bibliography

Bertsekas, Dimitri P. (2015). *Convex Optimization Algorithms*. Athena Scientific.

Freund, Yoav and Robert E. Schapire (1999). Large margin classification using the perceptron algorithm. In: *Machine Learning* 37.3, pp. 277–296.

Hardt, Moritz and Benjamin Recht (2022). *Patterns, Predictions, and Actions: Foundations of Machine Learning*. Princeton University Press.

Minsky, Marvin and Seymour A. Papert (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press.

Novikoff, A. (1962). On convergence proofs for perceptrons. In: *Proceedings of the Symposium on the Mathematical Theory of Automata*.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. In: *Psychological Review* 65.6, pp. 386–408.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning internal representations by error propagation. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. Ed. by David E. Rumelhart and James L. McClelland.

Vapnik, V. N. and A. Y. Chervonenkis (1974). *Теория распознавания образов [Theory of Pattern Recognition]*. Moscow: Nauka.