The Perceptron Learning Algorithm and its Convergence[*]

Shivaram Kalyanakrishnan

March 19, 2018

Abstract

We introduce the Perceptron, describe the Perceptron Learning Algorithm, and provide a proof of convergence when the algorithm is run on linearly-separable data. We also discuss some variations and extensions of the Perceptron.

1 Perceptron

The Perceptron, introduced by Rosenblatt [2] over half a century ago, may be construed as a parameterised function, which takes a real-valued vector as input, and produces a Boolean output. In particular, the output is obtained by thresholding a linear function of the input: the parameters of the Perceptron are precisely the coefficients of this linear function.

Figure 1 provides a useful visualisation of a Perceptron. For $d \geq 1$, a d-dimensional Perceptron has a d-dimensional parameter vector $\mathbf{w} \in \mathbb{R}^d$. For input $\mathbf{x} \in \mathbb{R}^d$, the output produced is $y = \text{sign}(\mathbf{w} \cdot \mathbf{x})$, where for $\alpha \in \mathbb{R}$,

$$\text{sign}(\alpha) = \begin{cases} 1 \text{ if } \alpha \geq 0, \\ 0 \text{ if } \alpha < 0. \end{cases}$$

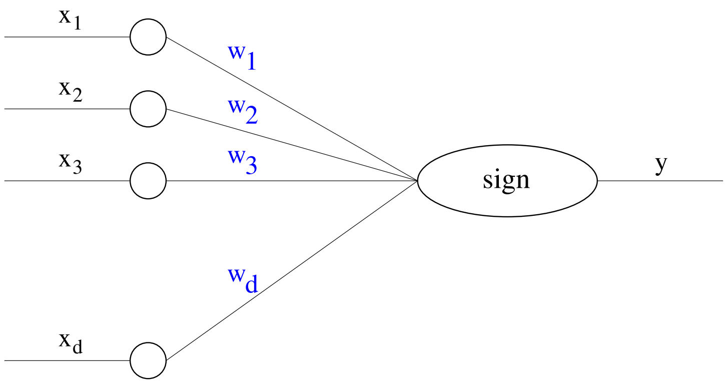

Figure 1: Perceptron

It is sometimes the practice to include an additional "bias" term $b \in \mathbb{R}$ to the linear function, such that the Perceptron's output is $\text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$. We proceed *without* this additive term, since it can easily be implemented by transforming the d-dimensional input into a $(d+1)$-dimensional one, which is then passed to a corresponding $(d+1)$-dimensional Perceptron. We leave the details of this transformation as an exercise to the student.

The significance of the Perceptron lies in its use as a device for *learning*. Consider Figure 2, which shows two data sets, both in the same 2-dimensional space. It is apparent that in the

[*] This note is based on one prepared by Collins [1].

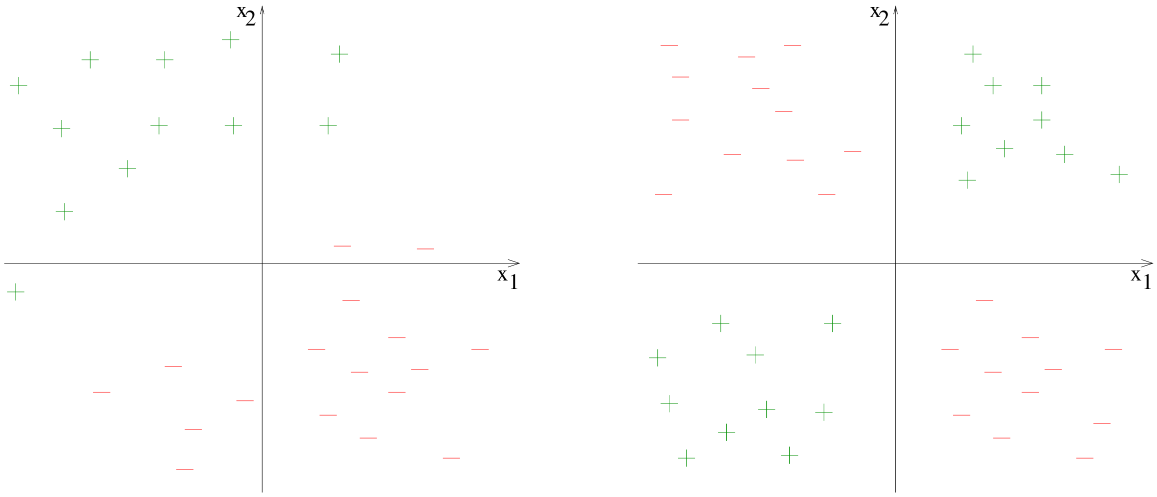

Figure 2: Labeled points in 2-dimensional space. The data set on the left is linearly separable, whereas the data set on the right is not.

figure on the left, we may draw a line such that all the points labeled + lie to one side of the line, and all the points labeled − lie to the other side. It is also apparent that no line with the same property may be drawn for the set of labeled points shown on the right. The property in question is linear separability. More generally, in d-dimensional space, a set of points with labels in $\{+, -\}$ is **linearly separable** if there exists a *hyperplane* in the same space such that all the points labeled + lie to one side of the hyperplane, and all the points labeled − lie to the other side of the hyperplane.

A Perceptron with its parameters fixed may indeed be viewed as an origin-centred hyperplane that partitions space into two regions. Concretely, the parameters (or *weights*) of the Perceptron may be interpreted as the components of a normal vector to the hyperplane. Observe that a normal vector suffices to fix an origin-centred hyperplane. In fact, the length of the normal vector is immaterial; its direction alone matters.

Given a set of points labeled + and −, the Perceptron Learning Algorithm is an iterative procedure to update the weights of a Perceptron such that eventually the corresponding hyperplane contains all the points labeled + on one side, and all the points labeled − on the other. We adopt the convention that the points labeled + must lie on the side of the hyperplane to which its normal points.

2 Perceptron Learning Algorithm

The input to the Perceptron Learning Algorithm is a data set of $n \geq 1$ points (each d-dimensional), and their associated labels (+ or −). For mathematical convenience, we associate the + label with the number +1, and the − label with the number −1. Hence, we may take our input to be

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^n, y^n),$$

where for $i \in \{1, 2, \ldots, n\}$, $\mathbf{x}^i \in \mathbb{R}^d$ and $y^i \in \{-1, 1\}$.

To entertain any hope of our algorithm succeeding, we must assume that our input data points are linearly separable. Consistent with our choice of ignoring the bias term in the Perceptron, we shall assume that not only are the input data points linearly separable, they can indeed be separated by a hyperplane that passes through the origin. After all, our Perceptron can only represent origin-centred hyperplanes!

Assumption 1 (Linear Separability)**.** *There exists some $\mathbf{w}^\star \in \mathbb{R}^d$ such that $||\mathbf{w}^\star|| = 1$ and for some $\gamma > 0$, for all $i \in \{1, 2, \ldots, n\}$,*

$$y^i(\mathbf{w}^\star \cdot \mathbf{x}^i) \geq \gamma.$$

Taking $\mathbf{w}^\star$ to be unit-length is not strictly necessary; it merely simplifies our subsequent analysis. Observe that the condition on each data point i essentially amounts to saying that its prediction (that is, sign$(\mathbf{w}^\star \cdot \mathbf{x}^i)$) matches its label (y^i). Requiring $y^i(\mathbf{w}^\star \cdot \mathbf{x}^i)$ to be strictly positive implies we can find a hyperplane such that none of the data points actually lie on the hyperplane. If the separating hyperplane must necessarily pass through some of the data points, note that the Perceptron's predictions for these points would depend on whether we assign sign(0) to be 0 or 1—which seems an arbitrary choice. If our input points are "genuinely" linearly separable, it must not matter, for example, what convention we adopt to define sign$(\cdot)$, or if we interchange the labels of the $+$ points and the $-$ points. This is the demand Assumption 1 places on our data set. The quantity γ is introduced in the assumption as a place-holder for the minimum value of the $y^i(\mathbf{w}^\star \cdot \mathbf{x}^i)$. Our analysis will yield an upper bound on the convergence time of the Perceptron Learning Algorithm that relates inversely with γ. The bound will also depend on the distance between the input points and the origin; we shall assume that this distance is at most R.

Assumption 2 (Bounded coordinates)**.** *There exists $R \in \mathbb{R}$ such that for $i \in \{1, 2, \ldots, n\}$,*

$$||\mathbf{x}^i|| \leq R.$$

Naturally, the Perceptron Learning Algorithm itself does not explicitly know $\mathbf{w}^\star$, γ, and R (although R can be inferred from the data). These quantities are merely useful artefacts we have defined in order to aid our subsequent analysis of the algorithm. The algorithm itself is remarkably simple, as we see below.

```
Perceptron Learning Algorithm

$k \leftarrow 1; \mathbf{w}_k \leftarrow \mathbf{0}$.
While there exists $i \in \{1, 2, \ldots, n\}$ such that $y^i(\mathbf{w}_k \cdot \mathbf{x}^i) \leq 0$:
    Pick an arbitrary $j \in \{1, 2, \ldots, n\}$ such that $y^j(\mathbf{w}_k \cdot \mathbf{x}^j) \leq 0$.
    $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + y^j x^j$.
    $k \leftarrow k + 1$.
Return $\mathbf{w}_k$.
```

The algorithm maintains a weight vector, initially the zero vector. If this weight vector already separates the $+$ points from the $-$ points, we are done. If not, we pick an arbitrary point $\mathbf{x}^j$ that is being misclassified. This point is used to update the weight vector: in fact, the update is nothing more than vector addition with $y^j\mathbf{x}^j$.

We note that the weight vector maintained and ultimately returned by the algorithm is not unit-length. In fact our proof to demonstrate the convergence of the algorithm relies on showing that the length of the weight vector increases with each update. Before proceeding to the proof, the student is encouraged to visualise the progress of the algorithm by sketching simple data sets in two dimensions and verifying that the Perceptron Learning algorithm repeatedly "tilts" the hyperplane in response to errors, until no more errors are made. In doing so, recall that the weight vector is a normal to the hyperplane.

3 Analysis

Theorem 3 (Perceptron convergence)**.** *The Perceptron Learning Algorithm makes at most $\frac{R^2}{\gamma^2}$ updates (after which it returns a separating hyperplane).*

Proof. It is immediate from the code that should the algorithm terminate and return a weight vector, the weight vector must separate the $+$ points from the $-$ points. Thus, it suffices to show that the algorithm terminates after at most $\frac{R^2}{\gamma^2}$ updates. In other words, we need to show that k is upper-bounded by $\frac{R^2}{\gamma^2}$. Our strategy to do so is to derive both lower and upper and bounds on the length of $\mathbf{w}^{k+1}$ in terms of k, and to relate them.

Note that $\mathbf{w}^1 = \mathbf{0}$, and for $k \geq 1$, note that if $\mathbf{x}^j$ is the misclassified point during iteration k, we have

$$\begin{aligned} \mathbf{w}^{k+1} \cdot \mathbf{w}^\star &= (\mathbf{w}^k + y^j \mathbf{x}^j) \cdot \mathbf{w}^\star \\ &= \mathbf{w}^k \cdot \mathbf{w}^\star + y^j (\mathbf{x}^j \cdot \mathbf{w}^\star) \\ &\geq \mathbf{w}^k \cdot \mathbf{w}^\star + \gamma. \end{aligned}$$

It follows by induction that $\mathbf{w}^{k+1} \cdot \mathbf{w}^\star \geq k\gamma$. Since $\mathbf{w}^{k+1} \cdot \mathbf{w}^\star \leq \|\mathbf{w}^{k+1}\| \|\mathbf{w}^\star\| = \|\mathbf{w}^{k+1}\|$, we get

$$\|\mathbf{w}^{k+1}\| \geq k\gamma. \tag{1}$$

To obtain an upper bound, we argue that

$$\begin{aligned} \|\mathbf{w}^{k+1}\|^2 &= \|\mathbf{w}^k + y^j \mathbf{x}^j\|^2 \\ &= \|\mathbf{w}^k\|^2 + \|y^j \mathbf{x}^j\|^2 + 2(\mathbf{w}^k \cdot \mathbf{x}^j) y^j \\ &= \|\mathbf{w}^k\|^2 + \|\mathbf{x}^j\|^2 + 2(\mathbf{w}^k \cdot \mathbf{x}^j) y^j \\ &\leq \|\mathbf{w}^k\|^2 + \|\mathbf{x}^j\|^2 \\ &\leq \|\mathbf{w}^k\|^2 + R^2, \end{aligned}$$

from which it follows by induction that

$$\|\mathbf{w}^{k+1}\|^2 \leq kR^2. \tag{2}$$

Together, (1) and (2) yield

$$k^2\gamma^2 \leq \|\mathbf{w}^{k+1}\|^2 \leq kR^2,$$

which implies $k \leq \frac{R^2}{\gamma^2}$. That is, the algorithm can proceed from k to $k+1$ (since some point is misclassified) only if $k \leq \frac{R^2}{\gamma^2}$. Our proof is done. □

4 Discussion

The Perceptron Learning Algorithm was among the earliest demonstrations of the *learnability* of concepts from data. The algorithm makes the rather strong assumption of the linear separability of data, which is seldom encountered in practice. However, nothing stops us from applying algorithms such as the Perceptron Learning Algorithm in practice in the hope of achieving good, if not perfect, results.

It is also common in practice to design features that "separate out" the data well. For example, observe that in the data set shown in the right in Figure 2, the addition of a feature $x_1 x_2$ will make the data linearly-separable in a 3-dimensional space. With real-world data sets, it is impractical to conjure up features that yield exact linear separability. Good features are ones that convey substantive (if not definitive) information about the labels. Learning systems that perform well in practice invariably owe a significant part of their success to the design of good features by the programmers.

While the Perceptron Learning Algorithm is a good starting point for learning linear classifiers, modern methods improve upon it in several ways. For example, we need not be satisfied with merely a separating hyperplane, but require one that leaves a maximal gap with the training data points. Such “maximum margin” classifiers (for example, Support Vector Machines) can also be learned efficiently from data. When the number of features (d) is very large, it is possible to explicitly look for weight vectors that have a much smaller support (that is, most components being zero). This “sparsity” can be achieved using a technique called “regularisation”.

One may generalise the 2-class problem we have described in this note to **multiclass classification**, wherein the set of labels can have a larger cardinality. In such a setting, the 2-class Perceptron can still be used as a blackbox. Suppose the set of classes is $\{1, 2, \ldots, L\}$. One common approach is to train a separate binary classifier for each class: that is, associate a separate weight vector $\mathbf{w}(l)$ with each class l. These weight vectors can then be trained in a manner akin to the Perceptron Learning Algorithm. During deployment, the class that they predict together for a given query point $\mathbf{x}$ is taken to be $\operatorname{argmax}_{l \in \{1,2,\ldots L\}} \mathbf{w}(l) \cdot \mathbf{x}$.

It exceeds the scope of our discussion to expand upon the myriad of threads that emanate from the Perceptron. The interested student is encouraged to enrol in a full course on machine learning.

References

[1] Michael Collins. Convergence proof for the perceptron algorithm, 2012. See `http://www.cs.columbia.edu/~mcollins/courses/6998-2012/notes/perc.converge.pdf`.

[2] Frank Rosenblatt. The perceptron: A perceiving and recognizing automaton (Project PARA). Technical Report 85-460-1, New York, 1957.