CMSC35000-1 Introduction to Artificial Intelligence **Winter 2005**

Lecture 1: 9 Jan 2007

Lecturer: Partha Niyogi *Scribe: Joshua A. Grochow*

1.1 Motivation

In general, we are interested in problems of **perception**, which often come in the form of **pattern recognition** problems. As always, one angle we are interested in is how our brains solve this problem. Some of the motivation and original ideas in this area are developed in articles from McCulloch's *Embodiment of Mind*:

1. McCulloch and Pitts. A logical calculus of the ideas immanent in nervous activity. (This is the original paper which introduced finite state machines)
2. Lettrin, Maturana, McCulloch and Pitts. What the frog's eyes tell the frog's brain.
3. Shannon. How to make a reliable circuit from unreliable components

One of the earliest models of neuronal activity was the **linear threshold circuit**: we imagine the neuron as having n inputs which can take in real values, $x_1, \ldots, x_n$ and then input weights $w_1, \ldots, w_n$. The decision of whether or not to fire is then based on a function of the linear combination $\sum_i w_i x_i = \mathbf{w} \cdot \mathbf{x}$.

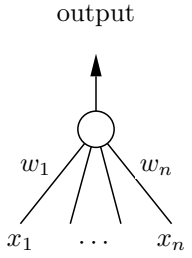

Figure 1.1: A linear threshold circuit.

The **perceptron** is nothing more than a linear threshold circuit whose output is the sign function of $\mathbf{w} \cdot \mathbf{x}$ (+1 when it's nonnegative, -1 otherwise). Since a perceptron has binary output, it can be considered to classify examples into one of two categories, and thus is often referred to as a linear classifier.

One of the early ideas researchers considered was that of *selective firing* of a neuron. The "grandmother cell" is the epitome of this idea – a neuron which fires only when you see a picture of your grandmother. (See Christoph Koch's website for the Halle Berry cell in teenage males.) How can we capture this idea in the linear threshold circuit model? We can think of a picture as an $n \times n$ array of light intesnity values, each of which is a real number. So each pattern can be thought of as a vector in $\mathbb{R}^n$. The percepton then takes the n coordinates of this vector as its input.

Now, since we're obviously not hardwired to recognize our grandmother, there must be some procedure by which the weights w_i can be updated to recognize any class of patterns. We want a learning algorithm, which adapts based on the inputs it is given and feedback. In other words, we want an algorithm which *learns from labelled examples* $(z_i, y_i) \in \mathbb{R}^n \times \{+1, -1\}$.

1.2 Perceptron Learning Algorithm

Here is the algorithm by which one can learn weights to classify any given set of (linearly separable) data:

PERCEPTRON($z[1..k], y[1..k]$)
($\mathbf{z}[i] \in \mathbb{R}^n$, $y[i] \in \{\pm 1\}$)
(Initialize)
1 $\mathbf{w} := \mathbf{0}$
(Main loop)
2 Repeatedly cycle through examples $(\mathbf{z}[i], y[i])$
3 Update on mistake (i.e. when $y[i](\mathbf{w} \cdot \mathbf{z}[i]) < 0$)
4 $\mathbf{w}_{new} := \mathbf{w}_{old} + y[i]\mathbf{z}[i]$
5 Stop when done (i.e. all examples correctly classified)

To see why this update rule makes sense, suppose $\mathbf{w}_{old} \cdot \mathbf{z} < 0$ when it should be positive, then

$$\begin{aligned} \mathbf{w}_{new} \cdot \mathbf{z} &= (\mathbf{w}_{old} + \mathbf{z}) \cdot \mathbf{z} \\ &= \mathbf{w}_{old} \cdot \mathbf{z} + \mathbf{z} \cdot \mathbf{z} \\ &> \mathbf{w}_{old} \cdot \mathbf{z} \text{ (since } \mathbf{z} \cdot \mathbf{z} > 0) \end{aligned}$$

so this update pushes the values of $\mathbf{w} \cdot \mathbf{z}$ in the correct direction. Similarly if the data is incorrectly classified the other way.

1.3 Convergence of the Perceptron Algorithm

In order for this algorithm to ever stop, we must at least assume that our examples (the input to the algorithm) are **linearly separable**, i.e. that there is some hyperplane which separates the data labelled by $y_i = +1$ from the data labelled $y_i = -1$. This is equivalent to saying that there exists some $\mathbf{w}_*$ such that $\mathbf{w}_*$ correctly classifies all the data.

Theorem 1.1 *If $(\mathbf{z}_i, y_i)$ are linearly separable, the perceptron algorithm converges.*

Proof: (Due to Novikoff and independently to ...) Since the data are linearly seperable, there is a $\mathbf{w}_*$ such that for all i, $y_i(\mathbf{w}_* \cdot \mathbf{z}_i) > 0$. Let δ be $\min_i |\mathbf{w}_* \cdot \mathbf{z}_i|$ and let $R = \max_i ||z_i||$. (Note that $\delta > 0$.)

Let $\mathbf{w}_k$ denote the value of $\mathbf{w}$ afer the k-th error (i.e. the k-th update of $\mathbf{w}$). Then there is some $(\mathbf{z}, y)$ such that $\mathbf{w}_{k+1} = \mathbf{w}_k + y\mathbf{z}$. Taking the dot product with $\mathbf{w}_*$, we get:

$$\begin{aligned} \mathbf{w}_* \cdot \mathbf{w}_{k+1} &= \mathbf{w}_* \cdot \mathbf{w}_k + y\mathbf{w}_* \cdot \mathbf{z} \\ &\geq \mathbf{w}_* \cdot \mathbf{w}_k + \delta \end{aligned}$$

By induction on k, $\mathbf{w}_* \cdot \mathbf{w}_k \geq \mathbf{w}_* \cdot \mathbf{w}_0 + k\delta$, but $\mathbf{w}_0 = 0$, so we have $\mathbf{w}_* \cdot \mathbf{w}_k \geq k\delta$.

Now we bound $||\mathbf{w}_k||$ in terms of R:

$$\begin{aligned} ||\mathbf{w}_k||^2 &= \mathbf{w}_k \cdot \mathbf{w}_k \\ &= (\mathbf{w}_{k-1} + y\mathbf{z})(\mathbf{w}_{k-1} + y\mathbf{z}) \\ &= ||\mathbf{w}_{k-1}||^2 + y^2\mathbf{z} \cdot \mathbf{z} + 2y\mathbf{w}_{k-1} \cdot \mathbf{z} \\ &\leq ||\mathbf{w}_{k-1}||^2 + R^2 \text{ (since } y\mathbf{w}_{k-1} \cdot \mathbf{z} < 0) \end{aligned}$$

$y\mathbf{w}_{k-1}\cdot\mathbf{z}$ is less than 0 because this is the error which led to the k-th update. By induction on k, $||\mathbf{w}_k||^2 < kR^2$. Finally, by Cauchy-Schwarz, $|\mathbf{w}_* \cdot \mathbf{w}_k| < ||\mathbf{w}_*||||\mathbf{w}_k||$. Combining this with the above two inequalities, we get:

$$k\delta \leq ||\mathbf{w}_*||||\mathbf{w}_k|| \leq ||\mathbf{w}_*||(\sqrt{k}R)$$

so we get the bound

$$k \leq \frac{||\mathbf{w}_*||^2 R^2}{\delta^2}$$

Since k is the number of mistakes, this bounds the number of updates. In paritucular, the algorithm terminates. ■

Even more, if we have N labelled datapoints, then the runtime is $O(\frac{||\mathbf{w}_*||^2 R^2}{\delta^2} N)$.

1.4 Discussion

1.4.1 Maximum margin

Recall from linear algebra that $\frac{|w_* \cdot z_i}{||w_*||}$ is the distance from the point z_i to the hyperplane defined by w_*. Let

$$\gamma = \frac{\delta}{||w_*||} = \min_i \frac{w_* \cdot z_i}{||w_*||}$$

γ is called the **margin** as it is the distance (margin) between the classifying hyperplane and the closest point on either side of it. Above, we bounded the runtime of the algorithm by $\frac{R^2}{\gamma^2} N$. So to find the best possible bound (for theoretical purposes), we want to find the maximum γ, or maximum margin. Denote the maximum margin by γ_*.

(Note that γ and R are both geometric invariants of the data, and do not depend on the algorithm at all.)

This gives rise to two important questions: first, is it useful to find the hyperplane giving the maximum margin?, and second, can we do it efficiently? By "useful" we are basically talking about how the classifier will perform in the future, or its **predictive power**. This is an important theme that will come up frequently in AI. In order to talk about its future performance, we will assume that the data are drawn from i.i.d. random sources (next lecture).

1.4.2 Complexity of function classes

The perceptron algorithm is essentially an algorithm which is searching through the space of all possible function given by hyperplanes to find one which correctly classifies the data. Let $H = \{\text{hyperplanes}\}$ denote this class of functions.

For a given margin γ, we can consider all hyperplanes which achieve a margin of at least γ, i.e. $H_\gamma = \{\text{hyperplanes separating balls of radius } \gamma \text{ around the points}\}$. If $\gamma > \gamma_*$, then H_γ is empty. So as we increase γ, we are narrowing our search space to smaller and small sets of hyperplanes. The **complexity of function classes** considered is another important theme that recurs frequently in AI.

1.4.3 Non-linearly separable data

Finally, what about data that is not linearly separable? **Nonlinear classifiers** are another important theme that we will see again and again. As a simple example, note that the XOR function is not linearly separable.

It is given by the training data: $((0,0),-1),((0,1),+1),((1,0),+1),((1,1),-1)$.

Note that, in the perceptron algorithm, no vector appears by itself. Vectors *always* appear dot-producted together, e.g. $\mathbf{w} \cdot \mathbf{z}$. This will allow us to generalize to broader, non-linear function classes, so long as we retain the structure of a certain kind of normed vector space, or Hilbert space.