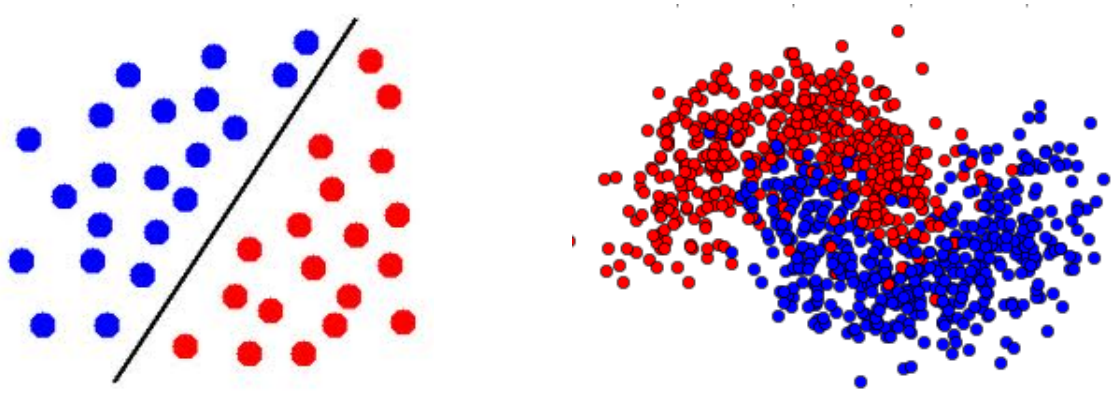
CS 6140: Machine Learning **Spring 2015**
College of Computer and Information Science
Northeastern University
Lecture 5 **March, 16**
Instructor: Bilal Ahmed **Scribe: Bilal Ahmed & Virgil Pavlu**

The Perceptron Algorithm[1]

1 Introduction

Consider the problem of binary classification where we have N data points as shown in Figure-1a. We can observe that the decision boundary between the two classes (blue and red points) is a straight line. Datasets that can be separated by a straight line are known as linearly separable datasets. Generalizing the example from two to m dimensions, a linearly separable dataset is one for which the decision boundary between the two classes is a linear function of the features x. Figure-1b shows an example where the two classes are not linearly separable.



(a) (b)

Figure 1: Linear (a) and non-linear (b) separability for binary classification in two-dimensions. The blue points represent the positive class and the red points belong to the negative class.

More formally, for a data set having N instances (x_i, y_i), where $x_i \in \mathbb{R}^m, y_i \in \{-1, 1\}$, a weight vector $w \in \mathbb{R}^m$ and a threshold θ, if the following conditions are satisfied:

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i > \theta \ ; & \text{if } y_i = 1 \\ \mathbf{w}^T\mathbf{x}_i < \theta \ ; & \text{if } y_i = -1 \end{cases}$$

then the dataset is linearly separable. Here $\mathbf{w}$ defines the normal vector for the hyperplane that separates the two classes in the m-dimensional feature space. The learning task in such a setting would be to learn the $m + 1$ parameters corresponding to the weight vector $\mathbf{w}$ and the threshold

[1]These lecture notes are intended for in-class use only.

θ. We can absorb the threshold into the weight vector by observing that the above conditions can be written more compactly as $\mathbf{w}^T\mathbf{x}_i - \theta > 0$, so that we can define an augmented weight vector $\tilde{\mathbf{w}} = [w_0\ w_1 \dots w_m]^T$ (note that $w_0 = \theta$) and also augment our feature set with a constant feature that is always equal to one so that every instance is $\tilde{\mathbf{x}} = [x_0\ x_1 \dots x_m]^T$, where $x_0 = 1$. The above conditions can now be stated as:

$$\begin{cases} \tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_i > 0\ ; & \text{if } y_i = 1 \\ \tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_i < 0\ ; & \text{if } y_i = -1 \end{cases} \tag{1}$$

It is more convenient to express the output of the perceptron in term of the sgn(.) function, so that $l_i = \text{sgn}(\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_i)$, where l_i is the label predicted by the perceptron for the i^{th} instance and the sgn(.) function is defined as:

$$\text{sgn}(x) = \begin{cases} 1 & ; x > 0 \\ -1 & ; \text{otherwise} \end{cases}$$

In the rest of the discussion we will assume that we are working with augmented features and to keep the notation as clear as possible we are going to revert back to $\mathbf{w}$ and $\mathbf{x}$.

2 The Perceptron

A perceptron taken as input a set of m real-valued features and calculates their linear combination. If the linear combination is above a pre-set threshold it outputs a 1 otherwise it outputs a -1 as per Equation-1 which is also called the perceptron classification rule. We can use the perceptron training algorithm to learn the decision boundary for linearly separable datasets. Algorithm-1 shows the perceptron training algorithm.

2.1 Example: Learning the boolean AND function for two variables

Consider the task of learning the AND function for two boolean variables x_1 and x_2. We can easily generate the data as there are only four possible instances, as shown in Table-1. These instances along with their labels are plotted in Figure-2a. In order to apply the perceptron algorithm we will map an output of 0 to -1 for the AND function.

Below we show the updates to the weight vector as it makes it first pass through the training data processing one instance at a time. In neural network nomenclature, a complete pass of through the training data is known as an *epoch*. During an epoch the updates made to the weight vector are

Data: Training Data:(x_i, y_i); $\forall i \in \{1, 2, \dots, N\}$, Learning Rate: η
Result: Separating Hyper-plane coefficients : $\mathbf{w}^*$
Initialize $\mathbf{w} \leftarrow \mathbf{0}$;
repeat
  get example (x_i, y_i);
  $\hat{y}_i \leftarrow w^T x_i$;
  if $\hat{y}_i y_i \leq 0$ **then**
    $w \leftarrow w + \eta y_i x_i$
until *convergence*;

Algorithm 1: The perceptron training algorithm.

Table 1: Data for learning the boolean AND function.

i	x_0	x_1	x_2	AND(x_1, x_2)
1.	1	0	0	0
2.	1	0	1	0
3.	1	1	0	0
4.	1	1	1	1

dependent on the order in which the instances are presented to the algorithm. The first epoch of the perceptron training algorithm on the training data in Table-1 (in the same order and a learning rate η of 1) is:

1. $\hat{y}_1 = \mathbf{w}^T\mathbf{x}_1 = 0$; therefore, $\mathbf{w} \leftarrow \mathbf{w} + (1)(-1)\mathbf{x}_1 = [-1\ 0\ 0]^T$
2. $\hat{y}_2 = \mathbf{w}^T\mathbf{x}_2 = -1$; therefore, no update to $\mathbf{w}$
3. $\hat{y}_3 = \mathbf{w}^T\mathbf{x}_3 = -1$; therefore, no update to $\mathbf{w}$
4. $\hat{y}_4 = \mathbf{w}^T\mathbf{x}_4 = -1$; therefore, $\mathbf{w} \leftarrow \mathbf{w} + (1)(1)\mathbf{x}_4 = [0\ 1\ 1]^T$

The decision boundary corresponding to the weight vector at the end of the first epoch is shown in Figure-2b. Figure-2(a-d) shows the decision boundary at the end of different epochs in the perceptron training algorithm and the final decision boundary after convergence. The final weight vector in this case is found to be $[-4\ 3\ 2]^T$, note that the first weight corresponds to the constant feature and hence defines the threshold θ.

2.2 Hypothesis space for the perceptron

The hypothesis space for the perceptron is $\mathbb{R}^m$, and any $\mathbf{w} \in \mathbb{R}^m$ constitutes a valid hypothesis. The goal of the learning algorithm is to find the weight-vector $\mathbf{w}^* \in \mathbb{R}^m$ that separates the two classes. For linearly separable datasets there would be a continuum of weight vectors that fulfill this criterion, the perceptron algorithm is guaranteed to find converge to a separating hyperplane as we show next. The particular weight vector that the perceptron training algorithm converges to is dependent on the learning rate η and the initial weight values.

2.3 Convergence of the perceptron training algorithm

Consider that we have training data as required by the perceptron algorithm. We are working with datasets that are linearly separable and hence we will assume $\exists\ \mathbf{w}_{opt} \in \mathbb{R}^m$ that perfectly separates the data and we will further assume that $\exists\ \gamma \in \mathbb{R}$ such that for any instance in the training set we have:

$$y_i \mathbf{w}_{opt}^T \mathbf{x}_i \geq \gamma\ ;\ \forall i \in \{1, 2, \ldots, N\} \tag{2}$$

Since, we have finite data we can assume without loss of generality that:

$$\|\mathbf{x}_i\| \leq R\ ; \forall i \in \{1, 2, \ldots, N\}$$

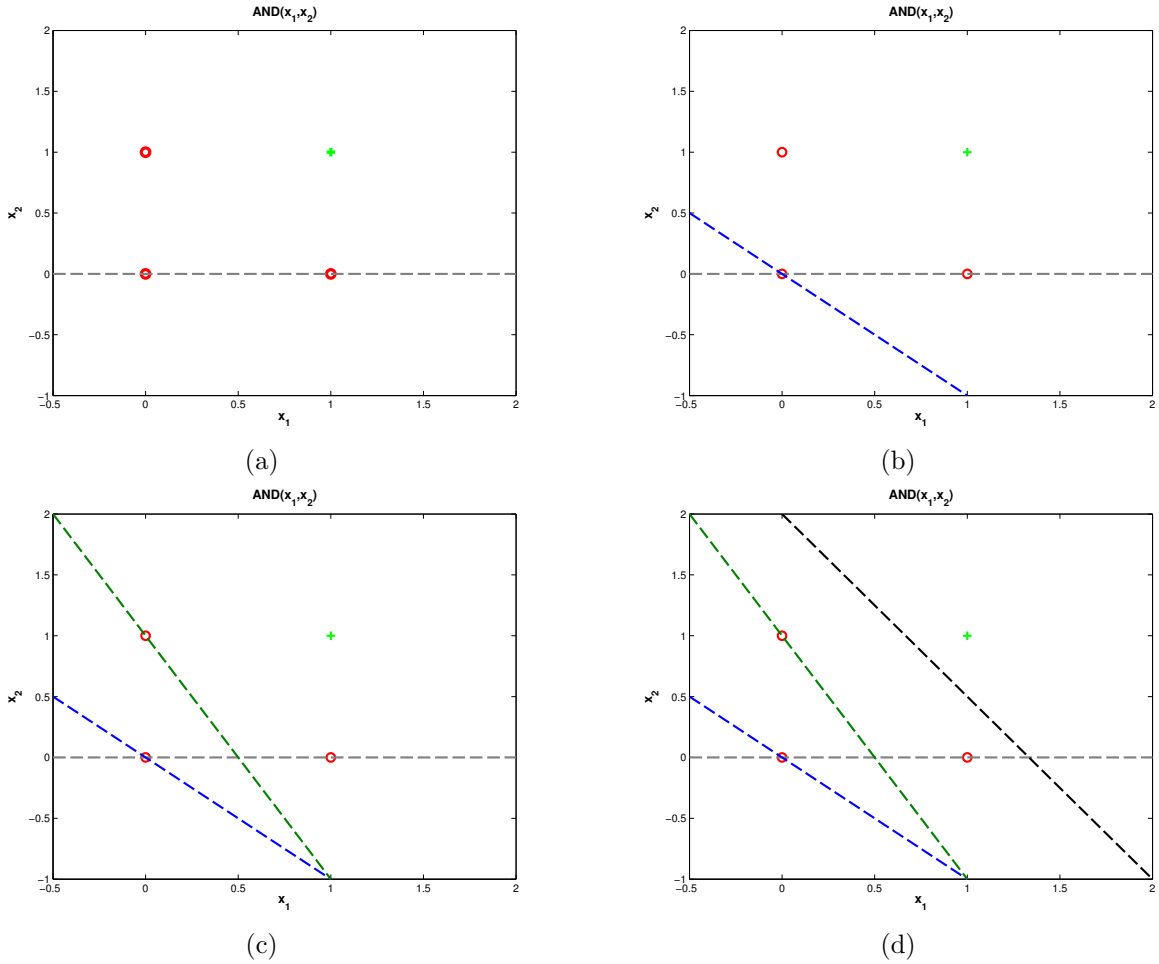


Figure 2: Learning the boolean AND function for two variables using the perceptron. (a) shows the initial decision boundary (b) the blue line shows the updated decision boundary after the first pass through the data (first epoch) (c) the green line is the decision boundary after the second epoch and (d) show the the final decision boundary (black line).

The quantity on the left in Equation-2 is proportional to the distance of an instance from the separating hyperplane. For all instances belonging to the positive class i.e., $y_i = 1$ we have $\mathbf{w}_{opt}^T \mathbf{x}_i \geq \gamma$ and $\mathbf{w}_{opt}^T \mathbf{x}_i \leq -\gamma$ for all instances with $y_i = -1$. This means that the optimal hyperplane $\mathbf{w}_{opt}$ separates the two classes with at least a distance proportional to γ.

Let $\mathbf{w}_k$ be the weight vector after the perceptron makes the k^{th} mistake on an instance $(\mathbf{x}_j, y_j)$, so that

$$\mathbf{w}_k = \mathbf{w}_{k-1} + y_j \mathbf{x}_j$$

to show that the perceptron training algorithm will learn the decision boundary within a finite number of steps we will bound the number of mistakes the perceptron makes on the training dataset.

Consider the inner product between $\mathbf{w}_{opt}$ and $\mathbf{w}_k$.

$$\begin{aligned}\mathbf{w}_k^T\mathbf{w}_{opt} &= (\mathbf{w}_{k-1} + y_j\mathbf{x}_j)^T\mathbf{w}_{opt} \\ &= \mathbf{w}_{k-1}^T\mathbf{w}_{opt} + y_j\mathbf{x}_j^T\mathbf{w}_{opt} \\ &\geq \mathbf{w}_{k-1}^T\mathbf{w}_{opt} + \gamma \quad (\because y_j\mathbf{w}_{opt}^T\mathbf{x}_j \geq \gamma)\end{aligned}$$

since we started with $\mathbf{w}_0 = \mathbf{0}$, by induction we have that,

$$\mathbf{w}_k^T\mathbf{w}_{opt} \geq k\gamma \tag{3}$$

The Cauchy-Shwarz inequality states that:

$$(\mathbf{a}^T\mathbf{b})^2 \leq \|a\|^2\|b\|^2 \quad ; \forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^m \tag{4}$$

Next we will look at the norm of $\mathbf{w}_k$:

$$\begin{aligned}\|\mathbf{w}_k\|^2 = &\quad (\mathbf{w}_{k-1} + y_j\mathbf{x}_j)^T(\mathbf{w}_{k-1} + y_j\mathbf{x}_j) \\ = &\quad \|\mathbf{w}_{k-1}\|^2 + 2y_j\mathbf{w}_{k-1}^T\mathbf{x}_j + \|\mathbf{x}_j\|^2 \\ \leq &\quad \|\mathbf{w}_{k-1}\|^2 + R^2 \qquad\qquad (\because y_j\mathbf{w}_{k-1}^T\mathbf{x}_j \leq 0)\end{aligned}$$

since we started with $\mathbf{w}_0 = \mathbf{0}$, by induction we have that,

$$\|\mathbf{w}_k\|^2 \leq kR^2 \tag{5}$$

Combining Equations 3-5, we get:

$$k^2\gamma^2 \leq (\mathbf{w}_k^T\mathbf{w}_{opt})^2 \leq \|\mathbf{w}_k\|^2\|\mathbf{w}_{opt}\|^2 \leq kR^2\|\mathbf{w}_{opt}\|^2$$

from the above we can see that:

$$k \leq \left(\frac{R}{\gamma}\right)^2\|\mathbf{w}_{opt}\|^2 \tag{6}$$

This shows that based on our initial assumptions, the perceptron training algorithm will stop after making a finite number of mistakes (Equation-6) irrespective of the sequence in which the samples are presented.

2.4 Is there a loss function here?

The loss function for the perceptron is given as:

$$\mathcal{L}_p(\mathbf{x}, y, \mathbf{w}) = \max(0, -y\mathbf{w}^T\mathbf{x}) \tag{7}$$

which is zero when the instance is classified correctly, and is proportional to the signed distance of the instance from the hyperplane when it is incorrectly classified. Note, that the instance is misclassified and is therefore on the wrong side of the hyperplane. Figure-3 shows a plot of the perceptron loss function. Perceptron loss is a special case of the Hinge loss, which we will encounter when discussing support vector machines. Also note that the perceptron loss involves the γ from Equation-2, and optimizes the negative γ of the misclassified instances.

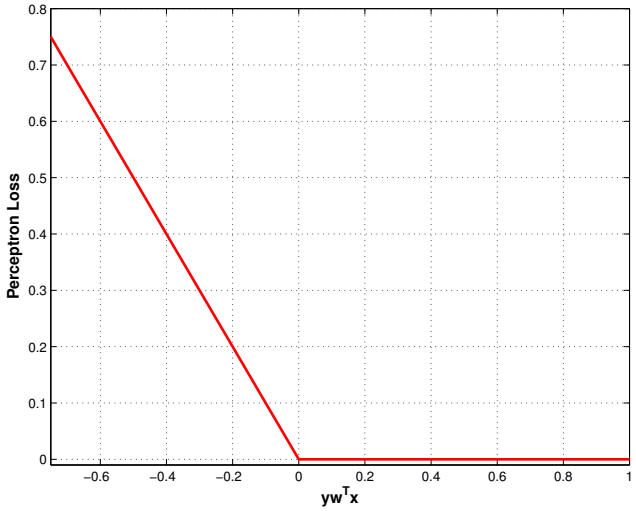


Figure 3: The perceptron loss function.

3 Inseparable Data

What happens when the data is not linearly separable? Based on our previous discussion the training algorithm will not halt. We can impose a limit on the number of iterations to make sure that the algorithm stops, but in this case we have no guarantees about the final weight vector. A possible avenue is to change the loss function so that instead of looking for a perfect classification we find the "best fit" to the decision boundary. In essence we are looking for a weight vector that does not perfectly separate the data but defines a decision boundary that has the minimum number of misclassifications on the training data.

In order to learn such a decision boundary using the perceptron, we need to first change the loss function, and in this case we can use the squared loss between the true labels and the predictions of the perceptron. We will be working with an unthresholded perceptron i.e., we will compute the output of the perceptron as:

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

as opposed to $\text{sgn}(\mathbf{w}^T\mathbf{x})$. This is known as a linear unit. The squared loss function over the training data is given as:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - o_i)^2 \tag{8}$$

We can optimize this loss function by using gradient descent. The gradient of $E(.)$ can be calculated as:

$$\nabla_{\mathbf{w}} E = \sum_{i=1}^{N}(y_i - o_i)(-\mathbf{x}_i)$$

and the gradient descent update to $\mathbf{w}$ is then simply,

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

The gradient descent algorithm is listed in Algorithm-2. As we are calculating the gradient (∇E)

Data: Training Data:(x_i, y_i); $\forall i \in \{1, 2, \ldots, N\}$, Learning Rate: η
Result: Optimal Hyper-plane coefficients based on squared loss: $\mathbf{w}^*$
Initialize $\mathbf{w} \leftarrow$ random weights ;
repeat
  calculate ∇E ;
  $w \leftarrow w - \eta \nabla E$
until *convergence*;

Algorithm 2: Gradient descent for training a linear unit. Note, that the update rule for the weight vector involves the calculation of the loss over the entire training set as compared to the perceptron training algorithm where we update the weight vector using one instance at a time.

based on the entire training dataset, the resulting gradient descent algorithm is also called the batch gradient descent algorithm. We can modify the batch update to work with single examples in which case the gradient is approximated as:

$$\nabla_{\mathbf{w}} E(\mathbf{x}_i) = (y_i - o_i)(-\mathbf{x}_i)$$

This is also known as stochastic gradient descent where we update the parameters based on a single example. The resulting training algorithm for a linear unit is shown in Algorithm-3.

Data: Training Data:(x_i, y_i); $\forall i \in \{1, 2, \ldots, N\}$, Learning Rate: η
Result: Optimal Hyper-plane coefficients based on squared loss: $\mathbf{w}^*$
Initialize $\mathbf{w} \leftarrow$ random weights ;
repeat
  get example (x_i, y_i);
  $\hat{o}_i \leftarrow w^T x_i$;
  $w \leftarrow w + \eta(y_i - o_i)\mathbf{x}_i$
until *convergence*;

Algorithm 3: Stochastic gradient descent algorithm for training a linear unit.

Stochasitc gradient descent works by processing a single example each time as opposed to the batch gradient descent that needs to process the entire dataset to carry out a single update. This makes the stochastic gradient descent algorithm faster than the conventional batch gradient descent.

4 References:

1. Andrew Ng's notes on perceptron. (cs229.stanford.edu/notes/cs229-notes6.pdf)
2. Roni Khardon's notes. (www.cs.tufts.edu/ roni/Teaching/CLT/LN/lecture14.pdf)
3. *Machine Learning*, Tom Mitchell.