

Now, we want an automatic procedure to find a \underline{w} in the solution region.

Training Algorithms (Learning) – Preliminaries

General Procedure –

1. Construct a criterion function $J(\underline{w})$ (appropriately chosen)
2. Minimize $J(\underline{w})$ with respect to \underline{w}
3. Result will be a solution weight vector \underline{w}

[Method I]

Use the Gradient Descent on $J(\underline{w})$ (Recursive Algorithm, DHS 5.4.2)

Let $\underline{w}(i)$ = solution weight vector at iteration i , then

$\underline{w}(i+1) = \underline{w}(i) - \alpha(i)\nabla_w J[\underline{w}(i)]$	This is the basic Gradient Descent
---	------------------------------------

$-\nabla_w J[\underline{w}(i)]$ points in the direction of steepest descent of J .

$$\nabla_w J(\underline{w}) = \frac{\partial J}{\partial w_1} \hat{u}_1 + \frac{\partial J}{\partial w_2} \hat{u}_2 + \dots + \frac{\partial J}{\partial w_N} \hat{u}_N \quad (\text{N-D space})$$

Must choose an appropriate criterion function $J(\underline{w})$. How? Coming in the next section.

What about choosing the learning rate parameter, $\alpha(i)$?

- There are various choices, specific to choices of $J(\underline{w})$ (more to follow). These are mostly suboptimal in terms of minimizes J after each step.
- There is an optimal choice (as defined above) under certain assumptions.
- Again coming in the next, next section

Finally,

Combine with: $\underline{w}(k+1) = \underline{w}(k) - \alpha(k)\nabla J[\underline{w}(k)]$, find a new \underline{w} via iterations.

Minimizing $J[\underline{w}(k+1)]$ can be done by setting

$$\alpha(i) = \frac{\|\nabla J\|^2}{(\nabla J)^T H(\nabla J)}$$

- One choice
- Can minimize # iterations to min. of J .
- Doesn't necessarily minimize amount of computation to min. of J .

[Method II]

The quadratic approximation of $J(w)$ leads to the Newton's Method.

Use the Newton's Algorithm (review the handout)

Suppose we want to solve:

$\min J(\underline{w})$, $\underline{w} \in R^n$ (drop a vector notation)

At $w = \bar{w}$ at a certain point, $J(w)$ can be approximated by:

$$J(w) \approx h(w) = J(\bar{w}) + \nabla J(\bar{w})^T (w - \bar{w}) + \frac{1}{2} (w - \bar{w})^T H(\bar{w}) (w - \bar{w})$$

which is the quadratic Taylor expansion of $J(w)$ at $w = \bar{w}$. $\nabla J(w)$ is the gradient of $J(w)$ and $H(w)$ is the Hessian of $J(w)$.

Note that $h(w)$ is a quadratic function, which is minimized by solving $\nabla h(w) = 0$.

Since the gradient of $h(w)$ is

$$\nabla h(w) = \nabla J(\bar{w}) + H(\bar{w})(w - \bar{w})$$

Therefore

$$\nabla J(\bar{w}) + H(\bar{w})(w - \bar{w}) = 0$$

which yields

$$w - \bar{w} = -H(\bar{w})^{-1} \nabla J(\bar{w})$$

The direction $-H(\bar{w})^{-1} \nabla J(\bar{w})$ is called the Newton direction or the Newton step.

Now

$$w = \bar{w} - H(\bar{w})^{-1} \nabla J(\bar{w})$$

Then, $w(k+1) = w(k) - H^{-1} \nabla J$ (Newton's Algorithm)

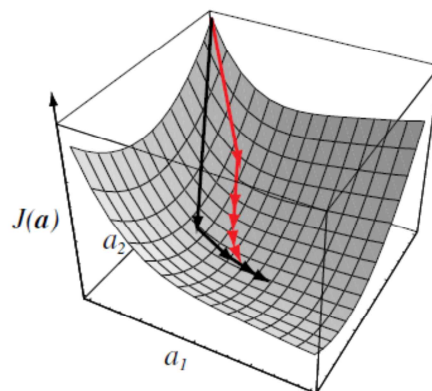


FIGURE 5.10. The sequence of weight vectors given by a simple gradient descent method (red) and by Newton's (second order) algorithm (black). Newton's method typically leads to greater improvement per step, even when using optimal learning rates for both methods. However the added computational burden of inverting the Hessian matrix used in Newton's method is not always justified, and simple gradient descent may suffice. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Perceptron Algorithm (DHS 5.5)

General Idea –

If $\underline{y}_m^{(1)}$ gives $\underline{w}^T \underline{y}_m^{(1)} < 0$ then increase \underline{w}

Perceptron Criterion Function

$$J(\underline{w}) = \sum_{\underline{y} \in Y} (-\underline{w}^T \underline{\tilde{y}}) \quad (\text{General Form})$$

where Y is the set of misclassified prototypes.

$J(\underline{w}) \geq 0$ always

$(\underline{w}^T \underline{\tilde{y}} < 0 \text{ for misclassified})$

Other criterion functions

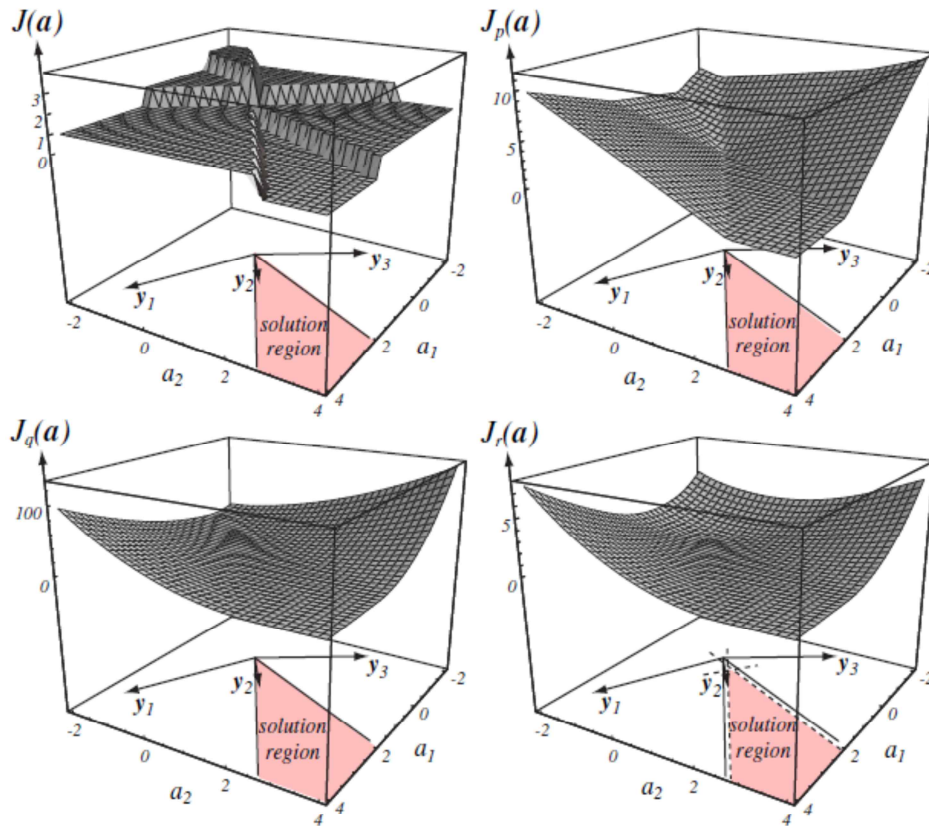


FIGURE 5.11. Four learning criteria as a function of weights in a linear classifier. At the upper left is the total number of patterns misclassified, which is piecewise constant and hence unacceptable for gradient descent procedures. At the upper right is the Perceptron criterion (Eq. 16), which is piecewise linear and acceptable for gradient descent. The lower left is squared error (Eq. 32), which has nice analytic properties and is useful even when the patterns are not linearly separable. The lower right is the square error with margin (Eq. 33). A designer may adjust the margin b in order to force the solution vector to lie toward the middle of the $b = 0$ solution region in hopes of improving generalization of the resulting classifier. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

$$J_p(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} (-\mathbf{a}^t \mathbf{y}), \quad (16)$$

$$J_q(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} (\mathbf{a}^t \mathbf{y})^2, \quad (32)$$

$$J_r(\mathbf{a}) = \frac{1}{2} \sum_{\mathbf{y} \in \mathcal{Y}} \frac{(\mathbf{a}^t \mathbf{y} - b)^2}{\|\mathbf{y}\|^2}, \quad (33)$$

[Perceptron Algorithm I] (One-at-a-time, Unreflected Prototypes)

If \exists prototype from S_1 , $\underline{w}^T \underline{y}_m^{(1)} \leq 0$, then increase \underline{w} ,

If \exists prototype from S_2 , $\underline{w}^T \underline{y}_m^{(2)} \geq 0$, then decrease \underline{w} ;

Repeat for all M_1+M_2 prototypes;

Continue cycling through all prototypes until \underline{w} is no longer updated.

For the i -th iteration:

If $\underline{w}^T(i) \underline{y}_m^{(1)} \leq 0$, then $\underline{w}(i+1) = \underline{w}(i) + \alpha(i) \underline{y}_m^{(1)}$

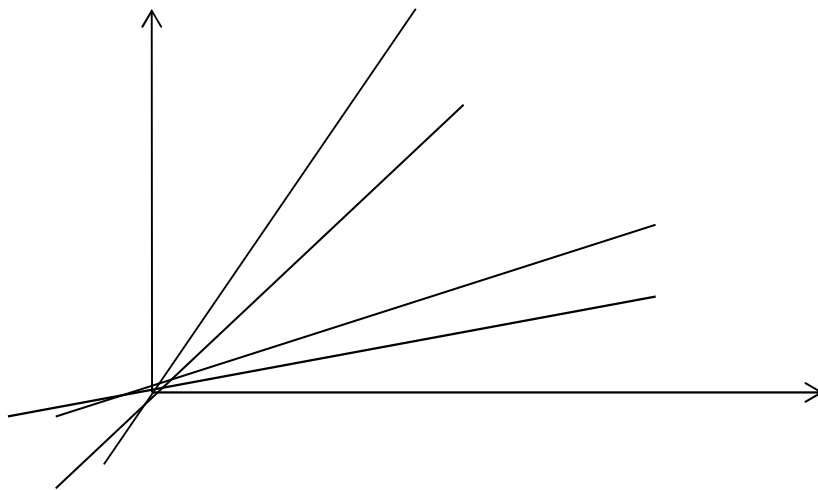
If $\underline{w}^T(i) \underline{y}_m^{(2)} > 0$, then $\underline{w}(i+1) = \underline{w}(i) - \alpha(i) \underline{y}_m^{(2)}$

Otherwise $\underline{w}(i+1) = \underline{w}(i)$

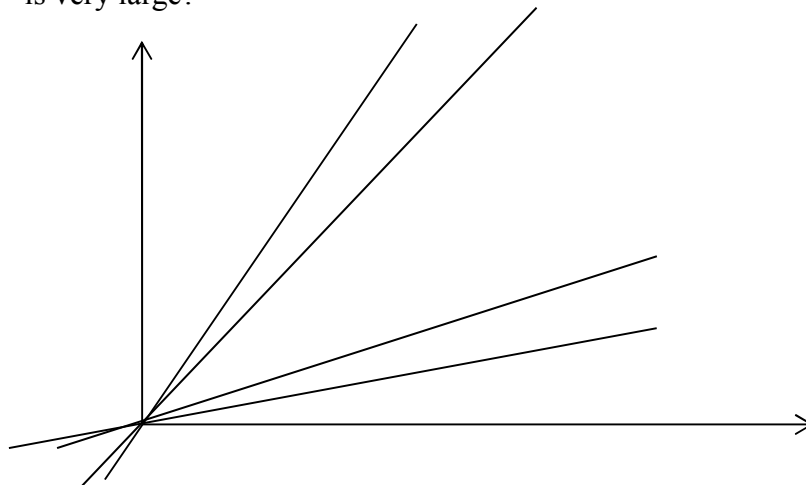
(Next prototype)

$\alpha(i) > 0$

1 pass through all prototypes = 1 epoch



What if α is very large?



[Perceptron Algorithm II] (One-at-a-time, Reflected Prototypes)

Use the reflected prototypes

$w(i + 1) = w(i) + \alpha(i)\underline{\tilde{y}}$ if the prototype $\underline{\tilde{y}}$ is misclassified.

$w(i + 1) = w(i)$ if the prototype $\underline{\tilde{y}}$ is correctly classified.

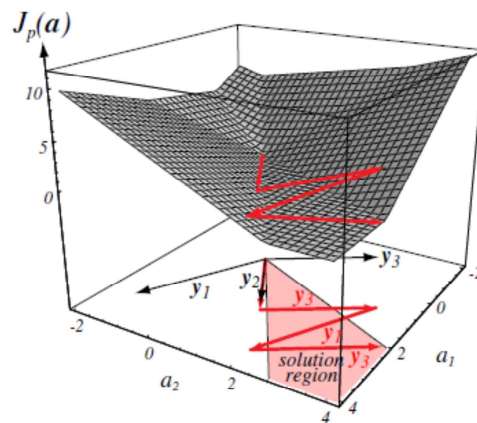


FIGURE 5.12. The Perceptron criterion, $J_p(\mathbf{a})$, is plotted as a function of the weights a_1 and a_2 for a three-pattern problem. The weight vector begins at $\mathbf{0}$, and the algorithm sequentially adds to it vectors equal to the “normalized” misclassified patterns themselves. In the example shown, this sequence is y_2, y_3, y_1, y_3 , at which time the vector lies in the solution region and iteration terminates. Note that the second update (by y_3) takes the candidate vector *farther* from the solution region than after the first update (cf. Theorem 5.1). From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

[Perceptron Algorithm III] (Many-at-a-time)

$J(\underline{w})$ is proportional to the sum of distances from misclassified \underline{y} 's to decision boundary

$$\nabla J(\underline{w}) = \sum_{\underline{y} \in Y} (-\underline{\tilde{y}})$$

$$\text{So } \underline{w}(i+1) = \underline{w}(i) + \alpha(i) \sum_{\underline{y} \in Y} \underline{\tilde{y}}$$

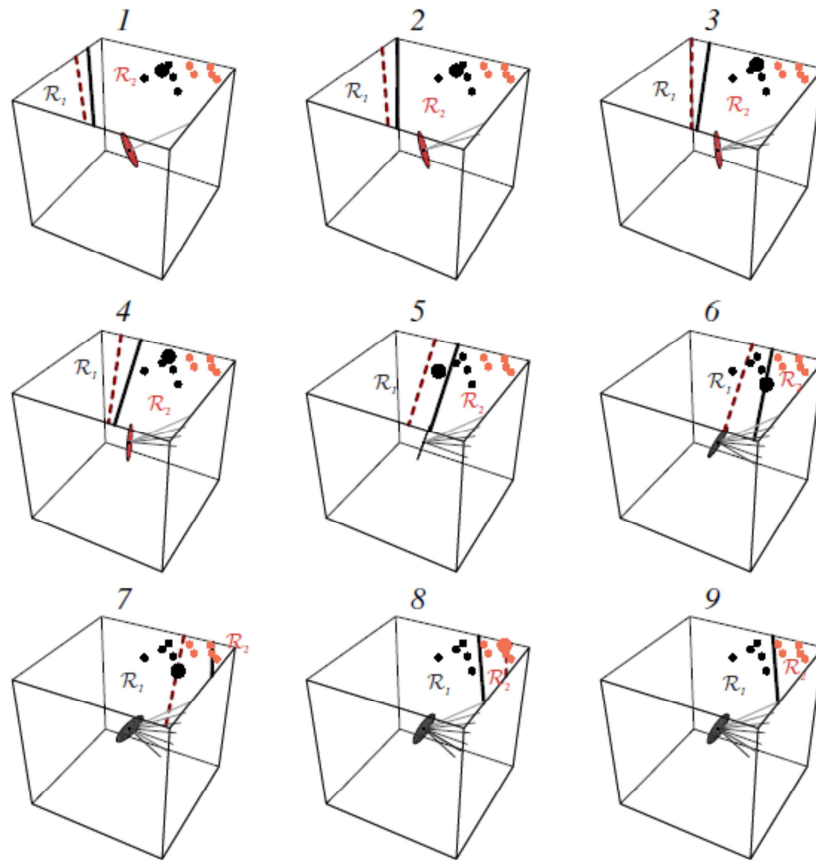


FIGURE 5.13. Samples from two categories, ω_1 (black) and ω_2 (red) are shown in augmented feature space, along with an augmented weight vector a . At each step in a fixed-increment rule, one of the misclassified patterns, y^k , is shown by the large dot. A correction Δa (proportional to the pattern vector y^k) is added to the weight vector—toward an ω_1 point or away from an ω_2 point. This changes the decision boundary from the dashed position (from the previous update) to the solid position. The sequence of resulting a vectors is shown, where later values are shown darker. In this example, by step 9 a solution vector has been found and the categories are successfully separated by the decision boundary shown. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Choice of α :

1. Fixed increment rule

$\alpha(i)$ =constant (independent of i)

$\alpha(i)>0$

2. Absolute correction rule

Choose α at each iteration to be just large enough to guarantee correct classification after weigh adjustment.

i.e., If $\underline{w}^T(i+1)\underline{y}_m^{(1)} > 0$

then $\underline{w}^T(i+1)\underline{y}_m^{(1)} = [\underline{w}(i) + \alpha \underline{y}_m^{(1)}]^T \underline{y}_m^{(1)} > 0$

Satisfied if $\alpha = \left\lceil \frac{|\underline{w}^T(i)\underline{y}_m^{(1)}|}{\underline{y}_m^{(1)T} \underline{y}_m^{(1)}} \right\rceil$

[*]=smallest integer larger than *

Guaranteed convergence.

3. Fractional correction rule

Choose $\alpha(i)$ to move a fraction, λ , normal to the hyperplane

$$|\underline{w}^T(i)\underline{y}_m^{(1)} - \underline{w}^T(i+1)\underline{y}_m^{(1)}| = \lambda |\underline{w}^T(i)\underline{y}_m^{(1)}|$$

$$|\underline{w}^T(i)\underline{y}_m^{(1)} - [\underline{w}(i) + \alpha \underline{y}_m^{(1)}]^T \underline{y}_m^{(1)}| = \lambda |\underline{w}^T(i)\underline{y}_m^{(1)}|$$

$$|\alpha \underline{y}_m^{(1)T} \underline{y}_m^{(1)}| = \lambda |\underline{w}^T(i)\underline{y}_m^{(1)}|$$

$$\alpha(i) = \frac{\lambda |\underline{w}^T(i)\underline{y}_m^{(1)}|}{\underline{y}_m^{(1)T} \underline{y}_m^{(1)}}$$

In this case $\underline{w}(0) \neq \underline{0}$

Move: $\alpha(i)\underline{y}_m^{(1)} = \frac{\lambda |\underline{w}^T(i)\underline{y}_m^{(1)}|}{\underline{y}_m^{(1)T} \underline{y}_m^{(1)}} \underline{y}_m^{(1)}$

Notes:

1. Fixed increment iterated many times with $\alpha=1$ for the same $\underline{y}_m^{(1)}$ gives the same result as for absolute correction rule.
2. Fixed increment with $\alpha>0$ is guaranteed to converge if prototypes are linearly separable.
3. Absolute correction is guaranteed to converge (if prototypes are linearly separable)
4. Fraction correction rule, for $0<\lambda<1$, will not converge
5. Fractional correction rule, for $\lambda=2$, the solution will reflect about the hyperplane an equal distance on either side.
6. One can apply algorithms sequentially (one-at-a-time) or simultaneously to all prototypes (many-at-a-time)

$$\underline{w}(i+1) = \underline{w}(i) + \alpha(i) \sum_{y \in Y} y \quad (\text{Many at a time, or batch})$$


$$\underline{w}(i+1) = \underline{w}(i) + \alpha(i) \underline{y}_m \quad (\text{One } \underline{y}_m \text{ at a time})$$

Problems:

1. If prototypes are not linearly separable, perceptron will not converge
2. Perceptron terminated early may give poor classification results.

[Extra]

About Frank Rosenblatt who simulated Perceptron on an IBM computer in 1957.

	<p>Frank Rosenblatt 1928–1969</p> <p>Rosenblatt's perceptron played an important role in the history of machine learning. Initially, Rosenblatt simulated the perceptron on an IBM 704 computer at Cornell in 1957, but by the early 1960s he had built special-purpose hardware that provided a direct, parallel implementation of perceptron learning. Many of his ideas were encapsulated in "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms" published in 1962. Rosenblatt's work was criticized by Marvin Minsky, whose objections were published in the book "Perceptrons", co-authored with</p>	<p>Seymour Papert. This book was widely misinterpreted at the time as showing that neural networks were fatally flawed and could only learn solutions for linearly separable problems. In fact, it only proved such limitations in the case of single-layer networks such as the perceptron and merely conjectured (incorrectly) that they applied to more general network models. Unfortunately, however, this book contributed to the substantial decline in research funding for neural computing, a situation that was not reversed until the mid-1980s. Today, there are many hundreds, if not thousands, of applications of neural networks in widespread use, with examples in areas such as handwriting recognition and information retrieval being used routinely by millions of people.</p>
---	--	---