Chapter 20 Section 5 — Slide Set 3

- Looking behind the scenes: a mathematical perspective
- Additional References:
  - Nillson, N. *Artificial Intelligence: A New Synthesis*, San Francisco: Morgan Kaufmann, 1998. (Chapter 2, Chapter 3 (3.1 - 3.2))
  - http://en.wikipedia.org/wiki/Sigmoid_function

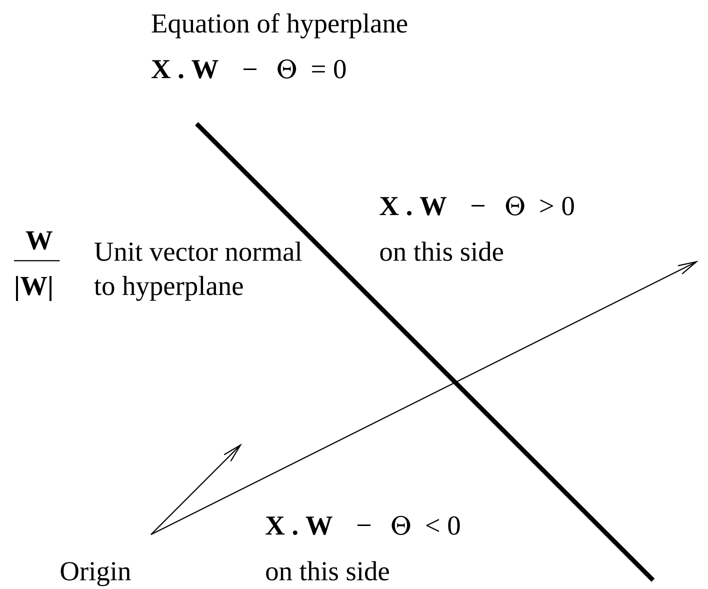
The learning problem

- We are given a set, *examples*,
  of n-dimensional vectors, $\mathbf{X}$,
  with components x_i, i = 1 , . . . , n.
- These vectors are *feature vectors* computed by a perceptual processing component.
- The values can be real or Boolean.
- For each $\mathbf{X}$ in *examples*, we also know the appropriate action or classification y.
  These associated actions are sometimes called the *labels* or the *classes* of the vectors.

The learning problem (cont'd)

- The set *examples* with the associated labels is sometimes called the the *training set*.
- The machine learning problem is to find a function, say, $h(\mathbf{X})$, that responds "acceptably" to the members of the training set.
- Note that this type of learning is *supervised*.
- We would like the action computed by h to agree with the label for as many vectors in *examples* as possible.

Training a single neuron



- adjusting the threshold θ changes the position of the hyperplane boundary with respect to the origin
- adjusting the weights changes the orientation of the hyperplane

Gradient descent method

- Define an *error function* that can be minimized by adjusting weight values.
- A commonly used error function is squared error:

$$\varepsilon = \sum_{X_i \in examples} (y_i - g_i)^2$$

where g_i is the actual response for input X_i, and y_i is the desired response.

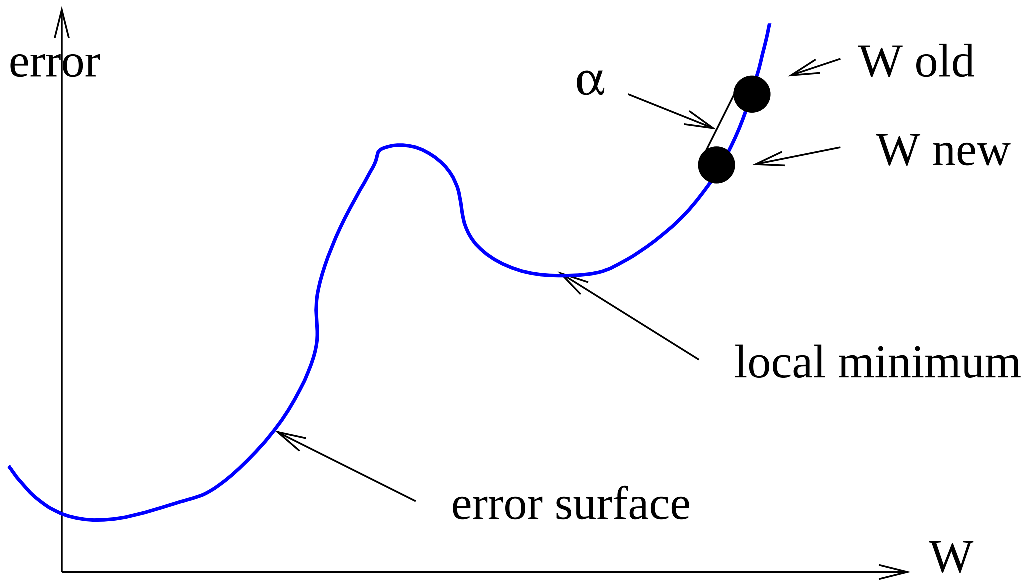
- For fixed *examples*, we see that the error depends on the weight values through g_i.

Gradient descent method (cont'd)

- A *gradient descent process* is useful to find the minimum of ε: calculate the gradient of ε in weight space and move the weight vector along the negative gradient (downhill).
- Note that, ε as defined, depends on *all* the input vectors in E.
- Use one vector at a time incrementally rather than all at once.
- Note that, the incremental process is an approximation of the "batch" process. Nevertheless, it works.

Gradient descent method (cont'd)

The following is a hypothetical error surface in two dimensions. Constant α dictates the size of the learning step.



The procedure

- Take one member of *examples*.
- Adjust the weights if needed.
- Repeat
  (a predefined number of times or until ε is sufficiently small.)

How to adjust the weights

- The squared error for a single output vector, $\mathbf{X}$, evoking an output of g, when the desired output is y is:

$$\varepsilon = (y - g)^2.$$

- The gradient of ε with respect to the weights is $\partial\varepsilon/\partial W = [\partial\varepsilon/\partial w_0, \ldots, \partial\varepsilon/\partial w_i, \ldots, \partial\varepsilon/\partial w_n]$.

How to adjust the weights (cont'd)

- Since $\varepsilon' s$ dependence on W is entirely through the dot product, s = X . W, we can use the chain rule to write

$$\partial\varepsilon/\partial W = \partial\varepsilon/\partial s \times \partial s/\partial W$$

- Because $\partial s/\partial W = X$

$$\partial\varepsilon/\partial W = \partial\varepsilon/\partial s \times X$$

- Note that $\partial\varepsilon/\partial s = -2(y-g)\partial g/\partial s$. Thus

$$\partial\varepsilon/\partial W = -2(y-g)\partial g/\partial s \times X$$

How to adjust the weights (cont'd)

- The remaining problem is to compute $\partial g/\partial s$.
- The perceptron output, g, is not continuously differentiable with respect to s because of the presence of the threshold function.
- Most small changes in the dot product do not change g at all, and when g does change, it changes abruptly from 1 to 0 or vice versa.
- We will look at two methods to compute the differential.

Computing the differential

- Ignore the threshold function and let $g = s$.
  (*The Widrow-Hoff Procedure*).
- Replace the threshold function with another nonlinear function that is differentiable.
  (*The Generalized Delta Procedure*).

The Widrow-Hoff procedure

- Suppose we attempt to adjust the weights so that every training vector labeled with a 1 produces a dot product of exactly 1, and every vector labeled with a 0 produces a dot product of exactly -1.
- In that case, with $g = s$, $\varepsilon = (y-g)^2 = (y-s)^2$, and, $\partial g/\partial s = 1$.
- Now, the gradient is

$$\partial\varepsilon/\partial W = -2(y-g)X$$

The Widrow-Hoff procedure (cont'd)

- Moving the weight vector along the negative gradient, and incorporating the factor 2, into a *learning rate parameter*, α, the new value of the weight vector is given by

$$W \leftarrow W + \alpha(y - g)X$$

- All we need to do now is to plug in this formula in the "adjust the weights" step of the training procedure.

The Widrow-Hoff procedure (cont'd)

- We have, $W \leftarrow W + \alpha(y - g)X$.
- Whenever $(y - g)$ is positive, we add a fraction of the input vector into the weight vector. This addition makes the dot product larger, and $(y - g)$ smaller.
- Similarly, when $(y - g)$ is negative, we subtract a fraction of the input vector from the weight vector.

The Widrow-Hoff procedure (cont'd)

- This procedure is also known as the *Delta rule*.
- After finding a set of weights that minimize the squared error (using $g = s$), we are free to revert to the threshold function for g.

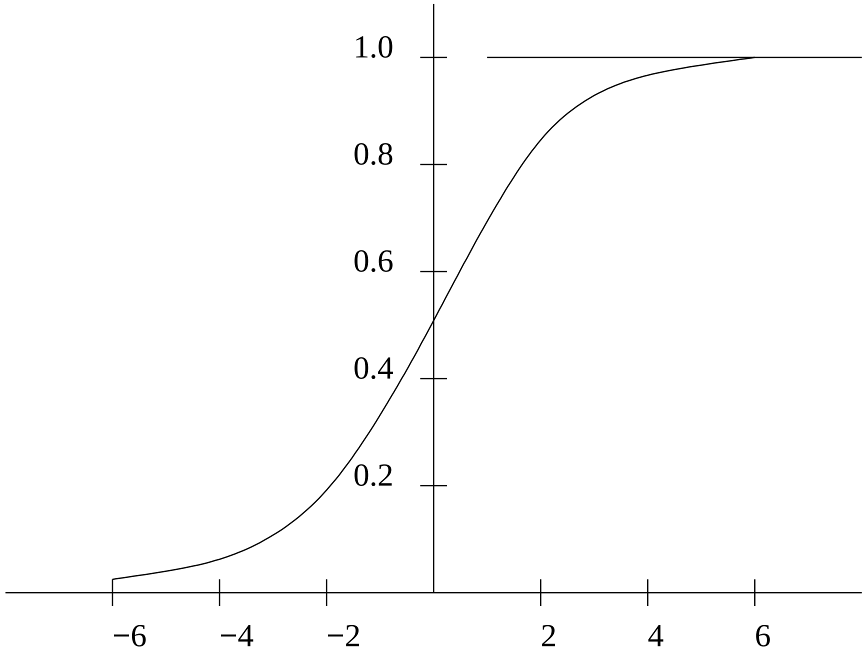
The generalized delta procedure

- Another way of dealing with the nondifferentiable threshold function: replace the threshold function by an S-shaped differentiable function called a *sigmoid*.
- Usually, the sigmoid function used is the *logistic function* which is defined as follows:

$$f(t) = \frac{1}{1+e^{-t}}$$

where, t is the input and f is the output.

A sigmoid function



It is possible to get sigmoid functions of different “flatness” by adjusting the exponent.

Differentiating a sigmoid function

Sigmoid functions are popular in neural networks because they are a convenient approximation to the threshold function and they yield the following differential:

$$\frac{d}{dt}sig(t) = sig(t) \times (1 - sig(t))$$

The generalized Delta procedure (cont'd)

- With the sigmoid function, $\partial g/\partial s = g(1-g)$
- Substitute into $\partial \varepsilon/\partial W = -2(y-g)\partial g/\partial s \times X$

$$\partial \varepsilon/\partial W = -2(y-g)g(1-g) \times X$$

- The new weight change rule is:

$$W \leftarrow W + \alpha(y-g)g(1-g)X$$

- This is equivalent to the weight change rule included in the learning algorithm:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e]$$

Fuzzy hyperplane

In generalized Delta, there is the added term $g(1-g)$ due to the presence of the sigmoid function.

When g = 0, $g(1-g)$ is also 0.

When g = 1, $g(1-g)$ is 0.

When g = 1/2, $g(1-g)$ reaches its maximum value (1/4).

Weight changes are made where changes have much effect on f.

For an input vector far away from the fuzzy hyperplane, $g(1-g)$ has value closer to 0, and the generalized Delta rule makes little or no change to the weight values regardless of the desired output.

Remarks

- The change is in the direction that helps correct the error. Whether it is corrected fully depends on α.
- It can be proven that if there is some weight vector, W, that produces a correct output for all the input vectors in *examples*, then after a finite number of input vector presentations, the error-correction procedure will find such a weight vector and thus make no more weight changes.
- Remember that a single perceptron can only learn linearly separable input vectors.
- The Widrow-Hoff and generalized Delta procedures can find minimum squared error solutions even when the minimum error is not zero.