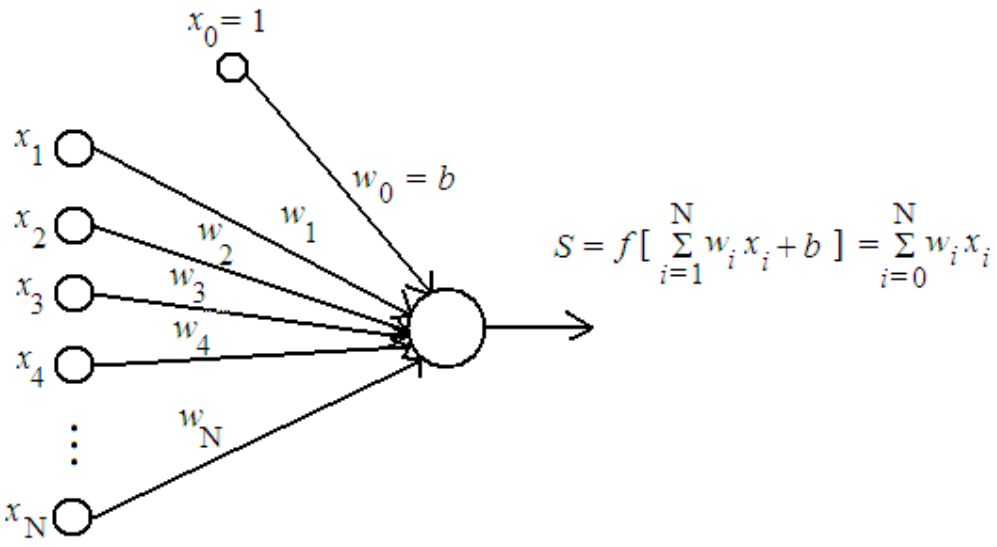
A Adaline

Poucos meses após a publicação do teorema da convergência do Perceptron por Rosenblatt, os engenheiros da Universidade de Stanford Bernard Widrow (1929 –) e Marcian Hoff (1937 –) publicaram um trabalho descrevendo uma rede neural muito parecida com o Perceptron, porém com as unidades de saída tendo funções de transferência *lineares* e com uma nova regra de aprendizado supervisionado, que ficou conhecida como **regra de Widrow-Hoff** (ou **regra delta**, ou ainda **regra LMS**). A rede neural apresentada por eles foi batizada de **Adaline** (do inglês *ADAptive LINear Element*).

Assim como o Perceptron, a Adaline é uma rede neural tendo uma camada de entrada com *N* unidades e uma camada de saída com apenas uma unidade. Não há camadas escondidas. A camada de entrada da Adaline é similar à camada A do Perceptron. As atividades dos seus neurônios fornecem uma representação do padrão de entrada sendo apresentado à rede neural e os seus valores são números reais quaisquer. Porém, a unidade de saída é diferente. Na Adaline, a atividade do neurônio de saída não é uma variável binária como no Perceptron, mas uma *função linear* do seu nível de ativação (veja a figura abaixo).



Adaline: a função de transferência é linear

As aplicações tidas em mente por Widrow e Hoff para a Adaline eram reconhecimento de padrões, processamento de sinais, regressão linear e controle adaptativo. Atualmente, a Adaline ainda é utilizada em algumas dessas aplicações.

A Adaline é uma rede neural treinada por um algoritmo de *aprendizado supervisionado*. Durante o período de treinamento, para cada padrão de entrada apresentado $\boldsymbol{x} = (x_1, x_2, ..., x_N)$ deve existir um padrão de saída desejado d e o objetivo do treinamento é fazer com que a saída S da rede para esse padrão seja igual a d,

$$d = S = \sum_{i=0}^{N} \omega_i x_i \ .$$

Note que S é uma variável contínua (uma função linear dos pesos), mas o valor desejado d pode ser tanto uma variável discreta (por exemplo, binária) como contínua.

Widrow e Hoff propuseram que a Adaline tivesse uma saída contínua porque agora pode-se definir uma *função erro* (ou custo) para avaliar o desempenho da rede:

$$E = E(d - S) = E(d - \sum_i \omega_i x_i) = E(\omega_0, \omega_1, \omega_2, \ldots, \omega_N) = E(\vec{\omega}) \ .$$

Note que a função erro E é uma função dos pesos da rede e do termo de viés (o peso ω_0), pois os valores de d e $\boldsymbol{x}$ são dados de antemão pelo problema que se quer resolver (eles correspondem, respectivamente, às entradas e às saídas desejadas).

Cada possível valor do vetor de pesos $\boldsymbol{\omega}$ determina um valor para a função erro E.

Em outras palavras, um dado vetor de pesos $(\omega_0, \omega_1, \omega_2, ..., \omega_N) = \boldsymbol{\omega}$ pode ser visto como um ponto num espaço de $N + 1$ dimensões, chamado de *espaço de pesos*. Suponhamos que temos um conjunto de p padrões e que queremos classificá-los em duas classes como no caso do perceptron, por exemplo, os padrões de uma classe deverão ter como saída desejada $d = +1$ e os padrões da outra classe deverão ter como saída desejada $d = -1$.

Para cada possível valor do vetor de pesos $\boldsymbol{\omega}$, a Adaline terá um certo desempenho na tarefa da classificação dos *p* padrões. Esse desempenho é *quantificado* pela função erro: se a Adaline classificar corretamente *todos* os padrões, seu erro é 0; e quanto maior o número de classificações erradas, maior será o erro.

Segundo a visão de Widrow e Hoff, o aprendizado de uma rede neural corresponderia a um processo de *busca* no espaço de pesos pelo vetor de pesos $\boldsymbol{\omega}$ que produz o menor erro possível. A função erro seria análoga à função custo em um problema de engenharia ou economia e o aprendizado corresponderia a buscar a configuração de pesos que *minimize* o custo.

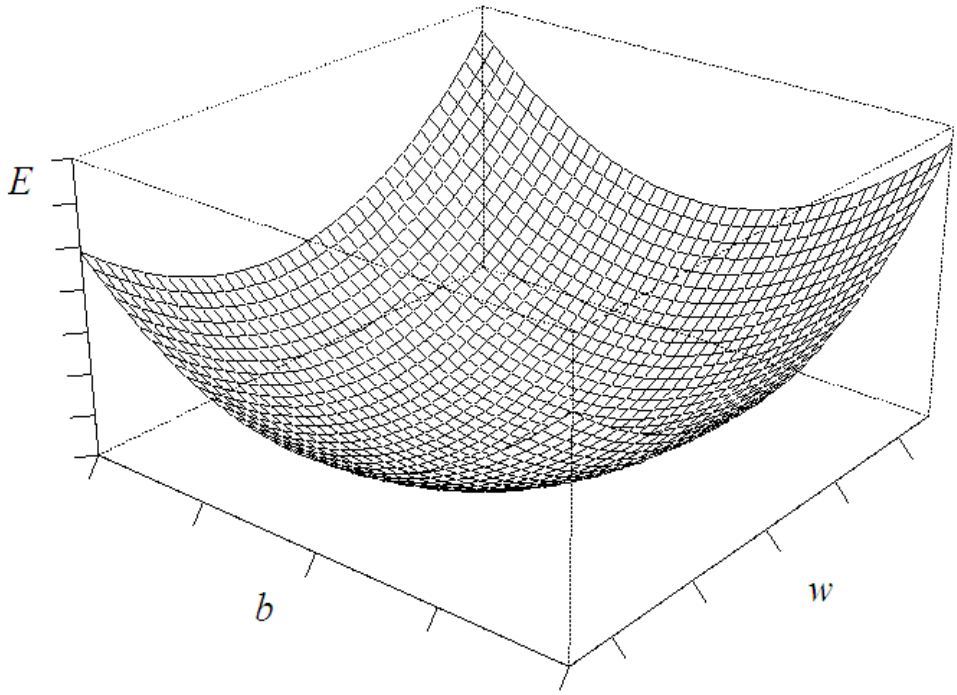
A função erro proposta por Widrow e Hoff é uma medida de erro muito popular, utilizada em várias áreas: o **erro quadrático médio** (a média da soma dos quadrados dos erros para cada padrão),

$$E = \frac{1}{2M} \sum_{p=1}^{M} \left(d^p - S^p \right)^2 = \frac{1}{2M} \sum_{p=1}^{M} \left[d^p - \sum_{i=0}^{N} \omega_i x_i^p \right]^2 ,$$

onde *M* é o número de padrões.

Note que embora os erros para cada padrão, $\left(d^p - \sum_{i=0}^{N} \omega_i x_i^p \right)$, possam ser quantidades negativas, positivas ou nulas, a função erro quadrático médio é sempre positiva ou nula.

A forma funcional da função acima é a de um *parabolóide* (pois a dependência de maior ordem nos pesos ω_i quadrática). Isso implica que existe um único conjunto de pesos (ω_0, ω_1, ω_2, ..., ω_N) para o qual a função custo tem valor mínimo. A figura a seguir ilustra isso para o caso em que há apenas dois pesos, $\omega_0 = b$ e $\omega_1 = \omega$.



É possível saber o valor dos pesos (ω_0, ω_1, ω_2, ..., ω_N) que corresponde ao mínimo da função erro. Aproveitando o exemplo bidimensional acima, os valores de ω e b para os quais a função E é minimizada podem ser obtidos resolvendo-se o sistema formado pelas equações,

$$\frac{\partial E}{\partial \omega} = 0 \quad \text{e} \quad \frac{\partial E}{\partial b} = 0 \, .$$

Calculando as derivadas, chega-se a um sistema de equações algébricas com duas incógnitas, ω e b. Resolvendo esse sistema, obtêm-se os valores ótimos de ω e b:

$$\omega = \frac{\sum_{p=1}^{M} \left(x^p - \bar{x}\right)\left(d^p - \bar{d}\right)}{\sum_{p=1}^{M} \left(x^p - \bar{x}\right)^2} \qquad \text{e} \qquad b = \bar{d} - \omega \bar{x} \, ,$$

onde $\bar{x}$ e $\bar{d}$ são os valores médios das variáveis x^p e d^p,

$$\bar{x} = \frac{1}{M} \sum_{p=1}^{M} x^p \qquad \text{e} \qquad \bar{d} = \frac{1}{M} \sum_{p=1}^{M} d^p \, .$$

Essa maneira de obter a solução do problema é chamada de *método dos mínimos quadrados*.

Essencialmente, o que o método dos mínimos quadrados faz é propor uma maneira *analítica* de se encontrar os valores dos parâmetros ω e b que minimizam a função custo E.

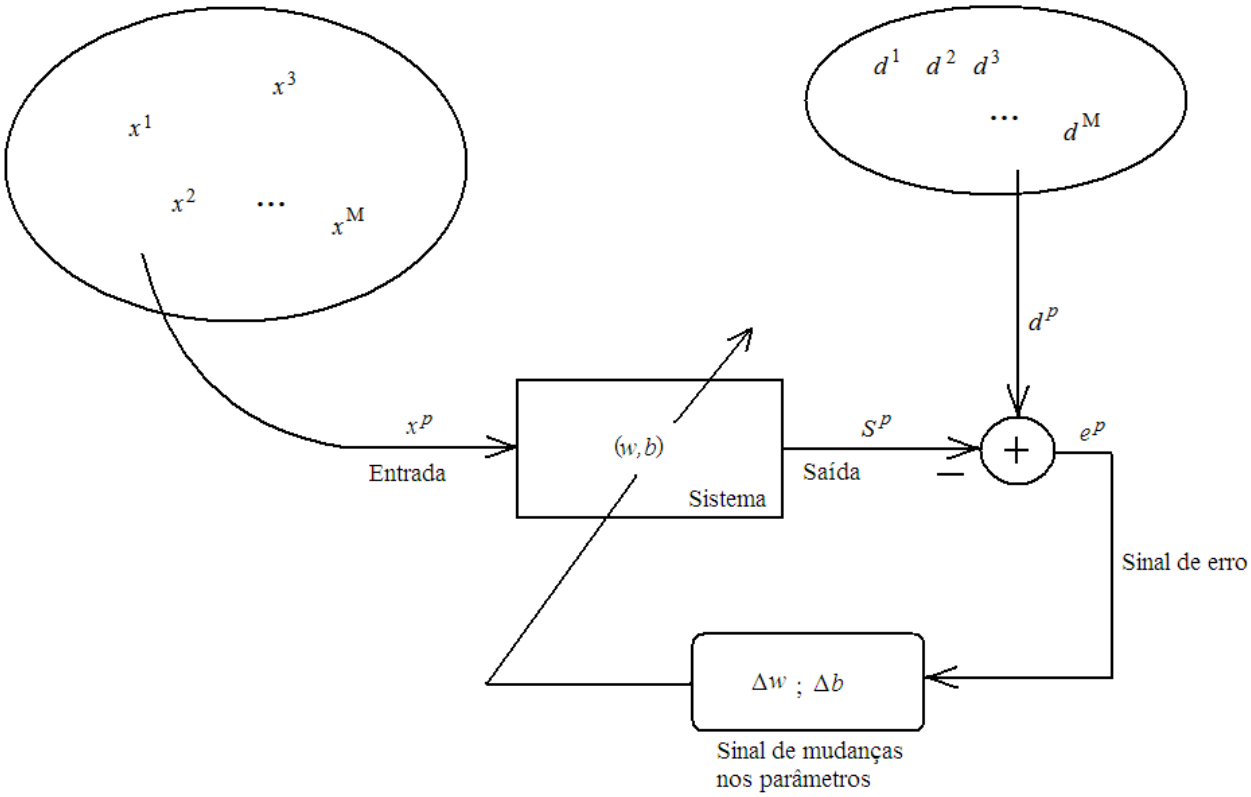
No entanto, pode-se pensar em outra maneira de se chegar aos parâmetros ótimos, uma maneira que envolva *adaptação*, implementada por *iterações* que levem o sistema de qualquer ponto inicial no espaço de parâmetros (ω_0, b_0) até o ponto ótimo, que vamos representar aqui por (ω^*, b^*).

Esse processo iterativo funcionaria da seguinte maneira:

Suponha que existe um conjunto de M padrões unidimensionais (x^1, x^2, x^3, ..., x^M) com as correspondentes M saídas desejadas (as classes) que se quer que o sistema aprenda, (d^1, d^2, d^3, ..., d^M).

Dada uma condição inicial dos parâmetros, (ω_0, b_0), o sistema calcula um valor de saída S^p para cada valor de entrada x^p (p = 1, ..., M). As saídas fornecidas pelo sistema são comparadas com as saídas desejadas gerando sinais de erro, $e^p = d^p - S^p$. Esses sinais de erro são enviados de volta ao sistema para que um novo par de parâmetros (ω_1, b_1) seja calculado com o objetivo de minimizar alguma função custo envolvendo os erros e_i (essa função custo pode ser a função erro quadrático médio, por exemplo). O sistema, com este novo par de parâmetros, recebe de novo como entrada os sinais x^p e calcula as saídas S^p. Novos erros são computados e um novo par de parâmetros (ω_2, b_2) é obtido. Prosseguindo desta maneira, espera-se que, após algumas iterações, o sistema chegue ao valor ótimo de parâmetros (ω^*, b^*).

O esquema a seguir ilustra o procedimento iterativo proposto para se chegar à solução (note que ele corresponde ao tipo de aprendizado chamado de *supervisionado*, definido na aula 3).



O ponto chave dessa estratégia iterativa de se encontrar os valores ótimos de ω e b é reconhecer que o erro contém informação que pode ser usada na busca desses valores ótimos.

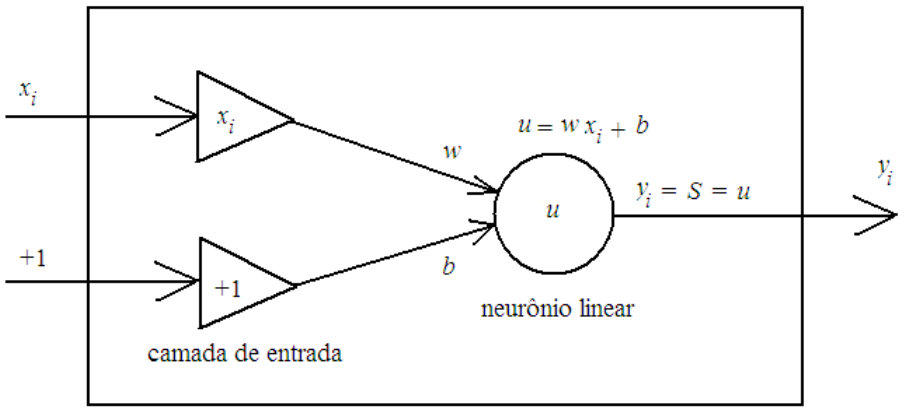
Inicialmente, como o sistema parte de um ponto qualquer no espaço de parâmetros, a sua resposta pode estar completamente errada. O sistema não está *adaptado* ao problema. Porém, à medida que os parâmetros do sistema vão sendo mudados em função dos erros cometidos por ele, o sistema vai aprendendo qual deve ser a região mais adequada do espaço de parâmetros na qual ele deve "existir" e, eventualmente, atinge o ponto ótimo em que ele está o mais adaptado possível aos dados fornecidos pelo ambiente.

Essa estratégia tem um "sabor" mais biológico do que a do método dos mínimos quadrados. Se outro conjunto de dados, de outro problema, fosse aplicado ao sistema cujos parâmetros foram calculados pelo método dos mínimos quadrados para resolver o primeiro problema, o sistema provavelmente forneceria um péssimo resultado.

Seria necessário novo cálculo dos parâmetros ω e b (feito *off-line*) para que ele fornecesse uma resposta melhor para o novo problema.

Já no caso do sistema em que os parâmetros são encontrados de forma adaptativa, ele também começaria com um resultado bastante ruim. Porém, em função desse erro inicial, os seus parâmetros começariam a se alterar (sem necessidade de cálculos *off-line*) e ele iria se "adaptando" gradualmente ao novo problema.

O sistema adaptativo que "busca" pelo valor ótimo dos parâmetros ω e b pode ser identificado com uma rede neural com aprendizado supervisionado. A rede neural tem apenas um único neurônio (figura a seguir) e é chamada na literatura de Adaline (do inglês *Adaptive Linear Element*). Note que os parâmetros do sistema adaptativo são identificados com os pesos ω e b da rede neural.



Sistema Adaptativo Linear = Adaline

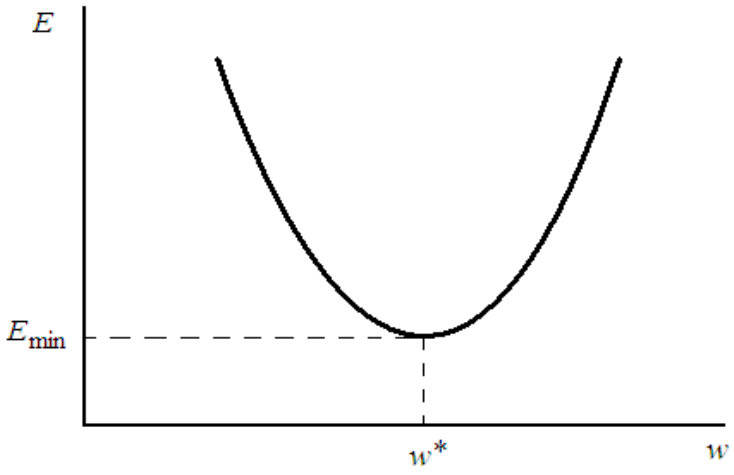
Vamos agora determinar um algoritmo que possa ser implementado iterativamente na estrutura da rede neural acima para corrigir os pesos ω e b (vamos tratar o viés do neurônio, b, como um peso também) com o intuito de levá-los até o resultado ótimo.

Para tornar mais didática a dedução a ser feita vamos considerar, sem perda de generalidade, o caso particular em que $b = 0$ (isto sempre pode ser feito a partir dos dados originais, subtraindo-se dos valores x^p e y^p os seus valores médios $\bar{x}$ e $\bar{y}$).

No caso unidimensional a função custo E torna-se uma função de apenas uma variável, ω:

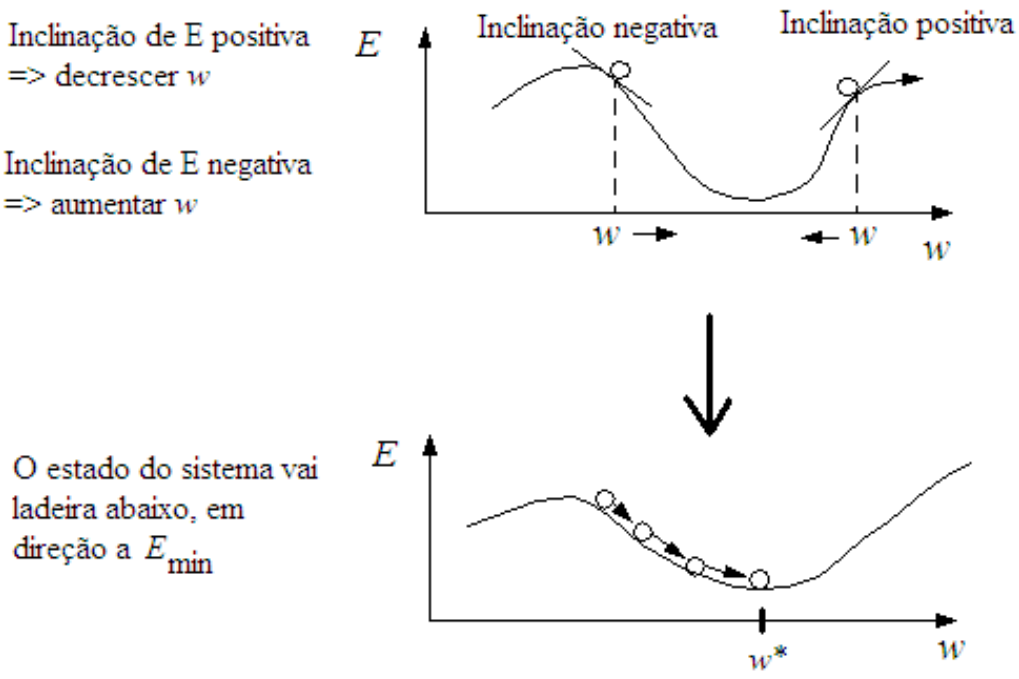
$$E = \frac{1}{2M}\sum_{i=1}^{M}(d_i - S_i)^2 = \frac{1}{2M}\sum_{i=1}^{M}(d_i - \omega x_i)^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(x_i^2\omega^2 - 2d_i x_i \omega + d_i^2\right).$$

E é uma parábola virada para cima cujos valores são sempre positivos ou nulos, pois ela é uma soma de quadrados. Um esboço do seu gráfico é dado abaixo:



Usando a curva da função custo E *versus* o parâmetro (ou peso) w, pode-se definir um método iterativo *geométrico* de busca do parâmetro ótimo w^* que minimiza E.

Este método é baseado no gradiente de E (indicado por ∇E). Dado um valor qualquer do parâmetro w, o gradiente de E nesse ponto define a direção de máximo crescimento da função $E(w)$ nesse ponto. Portanto, se fizermos uma mudança no valor de w na direção *oposta* à do gradiente, iremos levar E na sua direção de máxima redução, o que corresponde a levar o sistema na direção de $E_{\min}$ (veja a figura a seguir).



Em uma dimensão, o gradiente é a própria derivada da função: $\nabla E = dE/dw$. O ponto de mínimo da função custo E é aquele para o qual a derivada de E é igual a zero. É por isso que o método dos mínimos quadrados (em uma dimensão) se baseia no cálculo de $dE/dw = 0$. O valor de w determinado por esta equação é o próprio w^*.

Porém, não estamos querendo redescobrir o método dos mínimos quadrados. O que queremos aqui é desenvolver um método baseado em iterações (adaptativo) que leve o sistema a encontrar o valor w^* a partir de algum valor inicial w.

O gradiente é uma função conveniente para o desenvolvimento desse método porque ele é uma função que pode ser calculada *localmente*, sem necessitar do conhecimento da estrutura global da função custo E. No caso simples que estamos usando como exemplo, sabemos que a função custo tem a forma de um parabolóide, e que, portanto, tem sempre um único mínimo global, mas em um caso geral (não-linear) quase nunca se pode ter idéia sobre a forma global de uma função custo.

O algoritmo que pode levar o sistema de um ponto inicial qualquer w até o mínimo w^* é o seguinte:

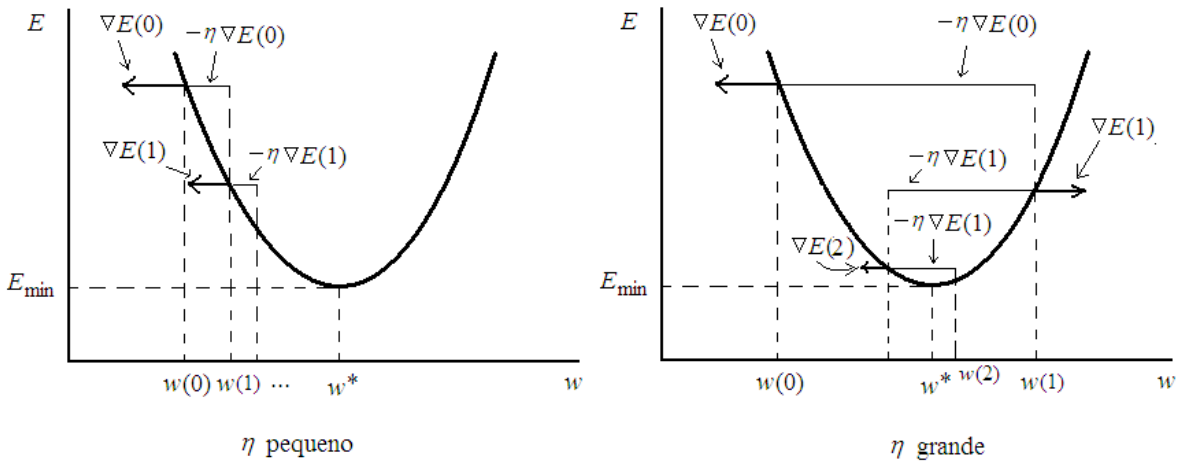
- Inicie a busca com um valor arbitrário para o peso, $w(0)$;
- Calcule o gradiente da função custo E em $w(0)$: $\nabla E(w(0))$;
- Calcule a modificação que terá que ser feita no peso w como um valor proporcional ao negativo do gradiente: $\Delta w(0) = -\eta \nabla E(w(0))$, onde η é a constante de proporcionalidade;
- Modifique o valor do peso: $w(1) = w(0) + \Delta w(0)$;
- Repita a operação, usando a fórmula genérica,

$$w(t+1) = w(t) + \Delta w(t),$$

até que se atinja o resultado desejado.

Este algoritmo é conhecido como o algoritmo do *descenso pelo gradiente* (*gradient descent* em inglês).

O resultado da aplicação do algoritmo do descenso pelo gradiente ao problema de busca do valor ótimo w^* pode ser ilustrado geometricamente pela figura abaixo. Note que o valor da constante η (o tamanho do passo dado na direção do gradiente) é crítico para a convergência para o mínimo. Se η for pequeno, a convergência será suave, mas lenta; se η for grande, a convergência não será tão suave (o sistema ficará pulando entre os dois lados da parábola), mas será rápida; já se η for muito grande, o sistema pode divergir e se afastar cada vez mais do mínimo.



Como o tamanho do passo, η, influencia a *velocidade* com que a Adaline aprende a sua tarefa (isto é, chega às vizinhanças do ponto de mínimo), ele é chamado de *parâmetro de taxa de aprendizagem*.

Para implementar o algoritmo do descenso pelo gradiente, devemos ter uma maneira de calcular o gradiente da função custo *E*. Como estamos trabalhando em 1 dimensão, este gradiente é apenas a derivada *dE*/*dw*.

Essa derivada pode ser calculada usando-se a regra da cadeia do Cálculo:

$$\nabla E = \frac{dE}{dw} = \frac{d}{dw}\left(\frac{1}{2N} \sum_{i=1}^{N} e_i^2 \right) = \frac{1}{N} \sum_{i=1}^{N} e_i \frac{de_i}{dw} .$$

O erro e_i é dado por $e_i = d_i - S_i = d_i - wx_i$. Portanto,

$$\frac{de_i}{dw} = -x_i ,$$

que pode ser substituído na expressão de ∇E para dar,

$$\nabla E = -\frac{1}{N} \sum_{i=1}^{N} e_i x_i = -\frac{1}{N} \sum_{i=1}^{N} (d_i - S_i) x_i ,$$

onde se escreveu explicitamente o erro e_i em função da saída da rede S_i.

Tendo calculado o gradiente de *E*, a alteração a ser feita no peso *w* a cada passo de alteração no seu valor pode ser escrita como:

$$\Delta w = -\eta \nabla E = \frac{\eta}{N} \sum_{i=1}^{N} \left(d_i - S_i \right) x_i = \frac{\eta}{N} \sum_{i=1}^{N} e_i x_i .$$

Este é um ponto importante: segundo a fórmula deduzida acima, uma alteração no peso *w* só será feita após a apresentação na entrada da rede de *todos* os padrões x_i e das comparações das suas respectivas saídas s_i com os valores desejados d_i (os sinais de erro e_i). Enquanto os pontos x_i ainda não tiverem sido completamente apresentados, a rede mantém o seu peso *w* com o valor atual.

Um passo computacional em que todos os pontos x_i (ou padrões, usando uma terminologia mais geral) são apresentados à rede é chamado de *época* de treinamento.

Durante uma época, os erros da rede (e_i) vão sendo armazenados e multiplicados pelos respectivos padrões de entrada (x_i). Ao final da época, os valores armazenados de $e_i x_i$, i=1,...,N são somados e multiplicados por η/N. Este é o valor de Δw para a época atual.

O novo valor de w, que será usado nos cálculos da próxima época (uma nova passagem por todos os valores de x_i), é determinado por

$$w(\text{novo}) = w(\text{atual}) + \Delta w(\text{atual}) .$$

O processo de alteração do peso w ao longo das épocas de apresentação dos padrões x_i à rede é chamado de *treinamento* da rede. Ele é um treinamento *supervisionado*, porque é necessário fornecer à rede o valor de saída desejado d_i para cada valor x_i para que se possa calcular os erros e_i e as mudanças no peso.

Um método de treinamento que requer a apresentação de todos os padrões (uma época) para só depois calcular a mudança nos pesos da rede é chamado de método de treinamento *por batelada* (em inglês, *batch mode*).

Existem métodos de treinamento que propõem que as alterações nos pesos sejam feitas após a apresentação de cada padrão de entrada, sem esperar que todos os padrões sejam apresentados. Tais métodos são chamados de métodos de treinamento *padrão-a-padrão*, ou *on-line*.

Um método de treinamento padrão-a-padrão para a Adaline foi proposto por Widrow e Hoff na década de 1960. Este método, que é uma simplificação do método por batelada visto acima, leva ao chamado algoritmo **LMS** de treinamento (do inglês *least mean square*), também conhecido como **regra delta**.

A idéia do método de Widrow e Hoff é desconsiderar a somatória por todos os quadrados dos erros no cálculo do gradiente da função custo *E* e levar em consideração apenas o quadrado do valor atual do erro, ou seja, aproximar o gradiente da função custo pelo gradiente do *erro instantâneo* cometido pela rede neural.

Matematicamente, isto implica em usar a seguinte equação para calcular o gradiente no *passo t* do algoritmo (quando o peso tem o valor *w*(*t*) e o padrão sendo apresentado à rede é *x*(*t*)):

$$\nabla E(t) = \frac{dE}{dw(t)} = \frac{d}{dw(t)}\left(\frac{1}{2N}\sum_{i=1}^{N} e_i^2\right) \approx \frac{d}{dw(t)}\left(\frac{1}{2}e^2(t)\right) = e(t)\frac{de(t)}{dw} = -e(t)x(t)$$

Note que o termo $(1/N)\sum_{i=1}^{N}$ desapareceu na hora que se fez a aproximação.

Usando esta aproximação, a alteração no peso da rede a ser feita no passo *t* fica sendo dada por,

$$\Delta w = \eta e(t) x(t) ,$$

e a regra de mudança do peso, conhecida como regra LMS, ou regra delta (pois Widrow e Hoff, no seu trabalho original, designavam o erro *e* cometido pela rede no passo *t* por δ) é:

$$w(t+1) = w(t) + \eta e(t) x(t) .$$

A regra delta resulta de uma *aproximação* do cálculo do gradiente da função custo *E*. Portanto, ela não faz o sistema ir exatamente na direção de máxima variação negativa do parabolóide que representa a função *E*. O processo de "descida" implementado por ela é mais errático (em termos mais matemáticos, diríamos *estocástico*). Porém, o sistema acaba "descendo" do mesmo jeito e vai, *em média*, na direção do gradiente durante as várias iterações.

Portanto, cada método passa por diferentes valores de pesos durante a convergência para o mínimo, mas ambos atingem os mesmos valores finais.

Note que como o método de treinamento por batelada faz uma atualização dos pesos por época enquanto que o método padrão-a-padrão faz uma atualização a cada apresentação, as atualizações dos pesos feitas pelos dois métodos estão relacionadas por:

atualizações pela regra delta = (atualizações por Batelada)x(número de padrões de treinamento).

Um algoritmo para a implementação da regra delta na Adaline do nosso exemplo, com apenas um peso w para ser adaptado, é dado a seguir.

1. Leia os padrões de treinamento: os N pares de valores (x_i,d_i), onde x_i é o padrão de entrada e d_i é o padrão de saída desejado;
2. Determine o valor do parâmetro de taxa de aprendizagem η;
3. Inicialize o valor do peso w. Em geral, $w(0) = 0$;
4. Para $t = 1, 2, ...$, repita os passos abaixo:
   - Sorteie um par (x_i,d_i) e faça $in(t) = x_i$ e $out(t) = d_i$;
   - Calcule a saída do neurônio: $s = u = w(t)in(t)$;
   - Calcule o erro: $e(t) = out(t) – s$;
   - Modifique o peso: $w(t + 1) = w(t) + \eta e(t)in(t)$;

Como no caso da Adaline (neurônio linear) sabemos que a função custo E é um parabolóide e existe um único mínimo, a repetição deste algoritmo por um número suficientemente grande de passos t levará a rede a atingir o valor mínimo w^*, desde que o parâmetro de aprendizagem η seja suficientemente pequeno (se η for muito grande o sistema pode divergir).

Porém, se o parâmetro η for muito pequeno, a convergência para o mínimo pode ser muito lenta (levar muitas iterações). Um artifício usado na prática é começar com um valor grande de η (mas não tão grande para levar a uma divergência) e ir fazendo η decrescer com o número de passos t. Dessa maneira, quando o sistema estiver bem perto do ponto de mínimo, a atualização do peso com passos cada vez menores fará com que ele oscile menos e se estabilize no mínimo.

Uma possível regra para a redução do parâmetro η com o número de passos é,

$$\eta(t+1) = \eta(t) - \beta \,,$$

onde β é um parâmetro ajustado empiricamente.

Mesmo com este artifício, o treinamento pode levar um tempo muito longo até que o sistema atinja o mínimo. No entanto, em muitas aplicações práticas pode-se tolerar uma certa porcentagem pequena de erros e, portanto, costuma-se definir um critério de parada para o treinamento da rede em função do valor do erro quadrático médio *E*. Quando o valor de *E* ficar menor ou igual a um certo valor $E_{\min}$, pára-se o treinamento e usa-se a rede com o valor de *w* no momento da parada.

Para implementar este método temos duas possibilidades, dependendo de estarmos usando o método de treinamento por batelada ou de estarmos usando o método de treinamento padrão-a-padrão.

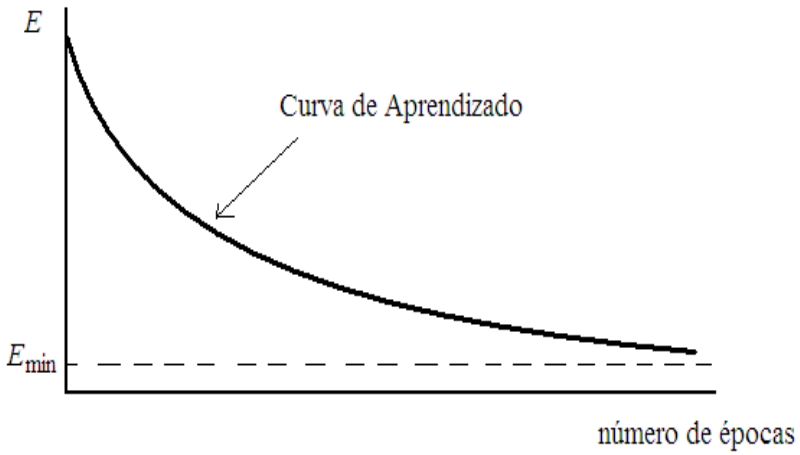
Se estivermos fazendo o treinamento por batelada, após a atualização do peso ao final de cada época de treinamento calcula-se o valor de *E* antes de recomeçar uma nova época, usando-se a fórmula,

$$E = \frac{1}{2N}\sum_{i=1}^{N} e_i^2 = \frac{1}{2N}\sum_{i=1}^{N} \left(d_i - wx_i\right)^2 \,,$$

onde o *w* usado é o calculado ao fim da época.

Se estivermos fazendo o treinamento padrão-a-padrão, então é conveniente alterar o algoritmo dado na acima para ir sorteando os padrões de treinamento sem repetição até que uma época se complete, isto é, até que todos os *N* padrões tenham sido apresentados à rede. Então, mantendo-se o valor atual do peso *w*, calcula-se *E* segundo a fórmula dada acima. Depois do cálculo de *E*, recomeça-se a aplicação do algoritmo por mais uma época.

Tanto de uma maneira como da outra, é costume guardar os valores de *E* calculados a cada época e construir o gráfico de *E* contra o número de épocas. A curva resultante mostra como o erro *E* decai em direção ao valor mínimo ao longo das épocas de treinamento. Ela é chamada de *curva de treinamento* ou de *aprendizado* da rede. Ela fornece uma maneira visual de se determinar como a rede neural está aprendendo a sua tarefa.



Outra maneira de evitar que a rede seja treinada indefinidamente é definir um número máximo de épocas de treinamento. Se a rede não tiver convergido para uma solução aceitável até esse número máximo de épocas, interrompe-se o treinamento.

Todo programa que implemente uma rede neural deve incluir uma condição de parada que leve em conta o número de épocas de treinamento.