Redes Neurais Artificiais

Prof. Marcelo Keese Albertini
Faculdade de Computação
Universidade Federal de Uberlândia

5 de Junho de 2017

Conteúdo

- Perceptron
- Gradiente descendente
- Redes multicamadas
- Retropropagação de erros

Modelos conexionistas

Humanos

- Tempo de ativação neural ~ 0.001s
- Número de neurônios $\sim 10^{10}$
- Conexões por neurônio $\sim 10^{4a5}$
- Tempo de reconhecimento de cenas $\sim 0.1s$

Computação paralela massiva

Exemplo: carro autônomo

Tudo à esquerda

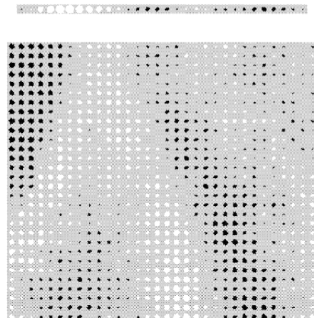
Em frente

Tudo à direita

30 unidades de saída

4 unidades internas (escondidas)

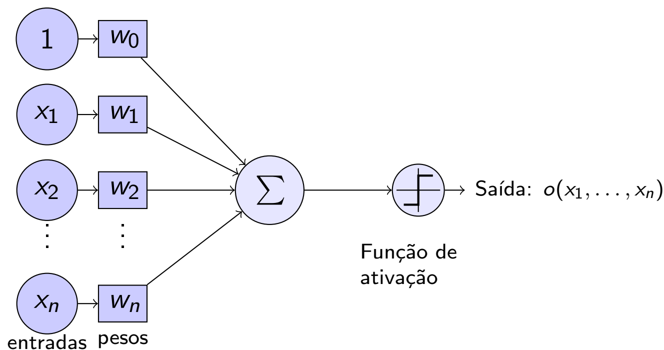
Sensor 30x32 "Retina"



Propriedades de redes neurais

- Muitas unidades de ativação
- Muitas conexões ponderadas entre unidades
- Processo altamente paralelizado
- Ênfase no ajuste de pesos automático

Perceptron

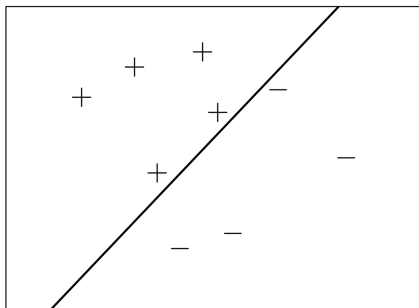


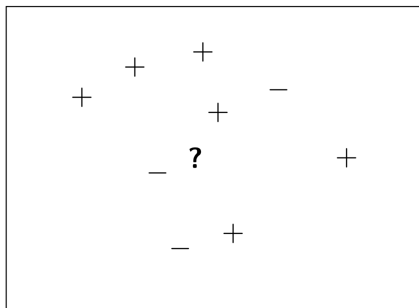
$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{se } w_0 + w_1 x_1 + \ldots + w_n x_n > 0 \\ -1 & \text{caso contrário.} \end{cases}$$

Na notação simplificada vetorial:

$$o(\vec{x}) = \begin{cases} 1 & \text{se } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{caso contrário.} \end{cases}$$

Superfície de decisão de um perceptron





Representa algumas funções úteis

- Quais pesos representam $g(x_1, x_2) = AND(x_1, x_2)$?

Mas algumas funções não são representáveis

- Todas as não linearmente separáveis
- Portanto, queremos redes de perceptrons para representar dados não linearmente separáveis

Treino do Perceptron

$$w_i \leftarrow w_i + \Delta w_i$$

onde

$$\Delta w_i = \eta(t - o)x_i$$

Onde:

- $t = c(\vec{x})$ é valor-alvo
- o é a saída do perceptron
- η é uma constante pequena (exemplo 0.1) chamada de taxa de aprendizado

Regra de treino do Perceptron

Possível provar convergência se

- Dados de treino são linearmente separáveis
- η é suficientemente pequeno

Gradiente descendente

Considere o caso simplificado *unidade linear*, onde

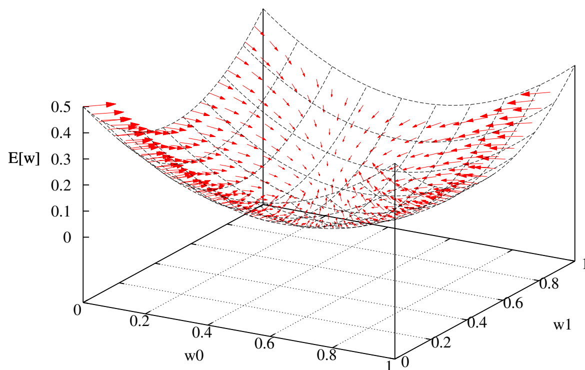
$o = w_0 + w_1 x_1 + \ldots + w_n x_n$

Objetivo é aprender w_i que minimiza o erro quadrático

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

onde D é conjunto de exemplos de treino, t_d é o valor de treino para o exemplo d e o_d o valor obtido pelo perceptron.

Figura gradiente descendente



Cálculo do gradiente

Gradiente

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$$

Regra de treino

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

Isto é

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Gradiente descendente

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2$$

Gradiente descendente

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2$$

Gradiente descendente

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

Gradiente descendente

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_d (t_d - o_d)(-x_{i,d})$$

Gradiente descendente

`Gradiente(exemplosDeTreino, eta)`

- Inicializar w_i com valor aleatório pequeno
- Fazer até convergir (ou treinar demais)
  1. Inicializar cada Δw_i para zero
  2. Para cada $\langle \vec{x}, t \rangle$ de exemplosDeTreino fazer
     - 2.1 Apresentar $\vec{x}$ ao neurônio e calcular a saída o
     - 2.2 Para cada peso w_i fazer

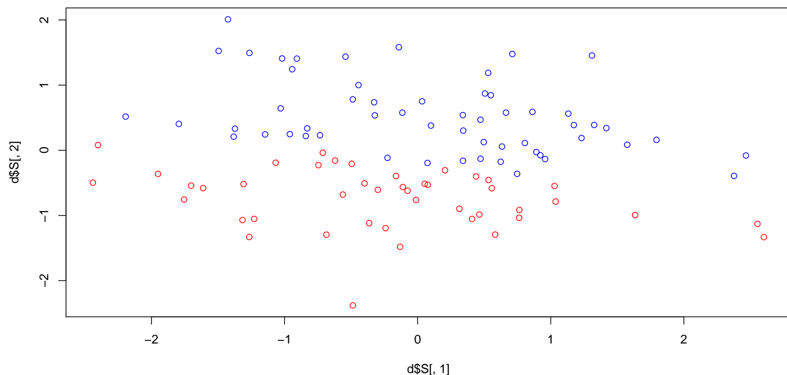
$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

  3. Para cada w_i fazer

$$w_i \leftarrow w_i + \Delta w_i$$

Simples perceptron: freestats::perceptrain

```
require(freestats)
z <- runif(n=3)# gera vetor separador de 2 classes
d <- fakedata(w=z,n=100) # gera dados respeitando z
plot(d$S[,1],d$S[,2],col=d$y+3)
```

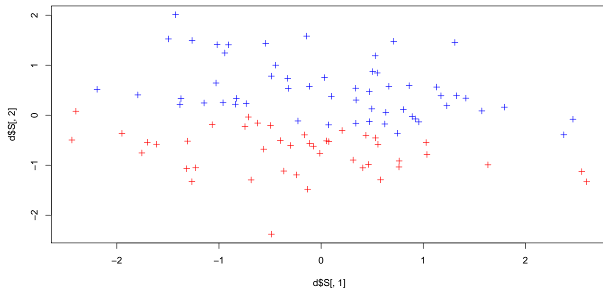


Simples perceptron: freestats::perceptrain

```
r <- perceptrain(S=d$S,y=d$y,alpha_k=0.5,endcost=0)
r

## $z
## [1]  4.000000  4.691419 19.650070
##
## $Z_history
##    [,1]      [,2]       [,3]
## z   1.0  1.173698  0.3876007
## z -11.0 -8.537313 13.2831838
## z   5.0  7.058348 18.1255715
## z   4.0  1.994424 20.1109102
## z   3.5  3.924788 19.9090663
## z   4.5  4.335235 19.6315956
## z   4.0  4.691419 19.6500701
##
## $NumofIteration
```

```r
perceptrain # ver codigo

## function (S, y, alpha_k = 1, endcost = 0)
## {
##     d <- dim(S)[2] - 1
##     n <- dim(S)[1]
##     x.matrix <- rbind(1, t(S)[1:d, ])
##     z <- x.matrix[, 2]
##     Z_history <- matrix(0, 0, ncol = d + 1)
##     NumofIteration = 0
##     Cost.gradient = 10000
##     while (sum(Cost.gradient^2) > endcost) {
##         index1 <- classify.pti(S, z) != y
##         cost.temp <- x.matrix
##         for (i in 1:n) {
##             cost.temp[, i] <- index1[i] * x.matrix[, i] * (-y[i])
##         }
##         Cost.gradient <- apply(cost.temp, MARGIN = 1, sum)
##         Z_history <- rbind(Z_history, z)
##         z <- z - alpha_k * Cost.gradient
##         NumofIteration <- NumofIteration + 1
##     }
##     res <- list(z = z, Z_history = Z_history, NumofIteration = NumofIteration)
##     class(res) <- "pt"
##     return(res)
## }
## <bytecode: 0x32d41a8>
## <environment: namespace:freestats>
```

```r
set.seed(1)
r <- perceptrain(S=d$S,y=d$y,alpha_k=0.01,endcost=0)
t = rep(0,nrow(d$S))
for (i in 1:nrow(d$S)) {
  t[i] = r$z %*% d$S[i,] # ativacao do neuronio
}
r$z - z

## [1]  0.04657516 -0.45349814  0.95232044

plot(d$S[,1],d$S[,2],col=d$y+3,pch=((t>0)+2))
```



Resumo

Treino do perceptron é garantido se

- Exemplos são linearmente separáveis
- Taxa de aprendizado η suficientemente pequena é usada

Treino do perceptron com gradiente descendente

- Convergência para hipótese de menor erro quadrático
- Mesmo quando dados de treino tem ruído
- Mesmo quando dados não são separáveis por H

Treino com gradiente: em lote vs. incremental

Modo em lote

Fazer até convergir

1. Computar o gradiente $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Gradiente descendente: modo incremental

Modo incremental

Fazer até convergir

- Para cada exemplo de treino $d \in D$
  1. Computar o gradiente $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

$$E_d[\vec{w}] \equiv \frac{1}{2}(t_d - o_d)^2$$

Gradiente descendente incremental pode aproximar o modo em lote se η for pequeno o suficiente.

Redes multi-camadas de unidades sigmóides

Neural net and traditional classifiers, WY Huang, RP Lippmann - Neural information processing systems, 1988

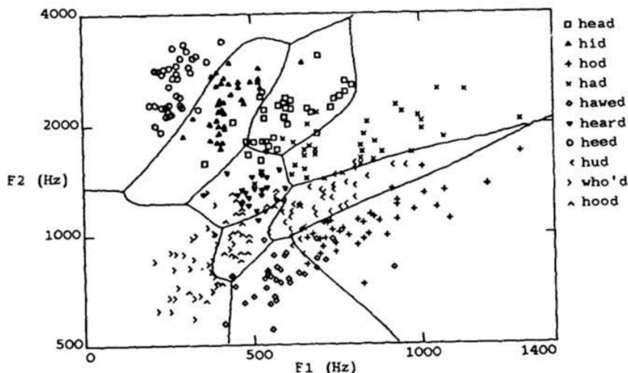
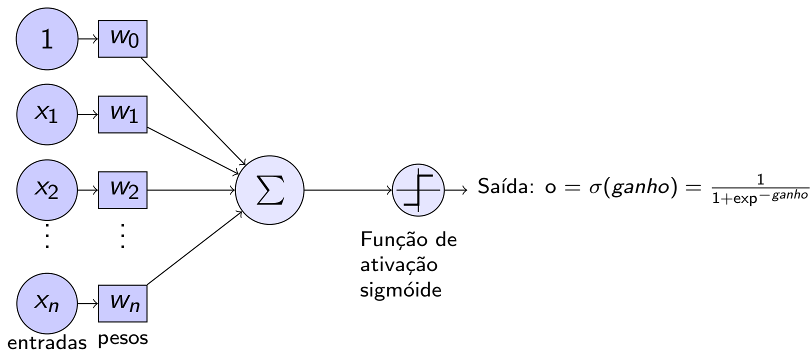


FIG. 9. *Decision regions formed by a two-layer network using BP after 200,000 training tokens from Peterson's steady state vowel data [Peterson, 1952]. Also shown are samples of the testing set. Legend show example of the pronunciation of the 10 vowels and the error within each vowel.*

Unidade sigmóide



$$\textbf{ganho} = \sum_{t=0}^{n} w_i x_i$$

$\sigma(x) = \frac{1}{1+\exp^{-x}}$ é a **função de transferência** sigmóide

Propriedade útil

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$

Podemos derivar regras de gradiente descendente para

- Uma unidade sigmóide
- Rede multi-camadas de unidades sigmóides
  - Retroprogação de erros

Gradiente de erro para uma unidade sigmóide

- $ganho = \sum_{t=0}^{n} w_i x_i$
- $o_d = \sigma(ganho_d)$ é a saída da função de transferência do neurônio respondendo ao d-ésimo exemplo no conjunto D
- t_d é a resposta esperada para o d-ésimo exemplo
- E é o erro da rede
- t_d não varia de acordo com w_i, então $\frac{\partial t_d}{\partial w_i} = 0$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$,usa regra da soma e obtém:

Gradiente de erro para uma unidade sigmóide

- $ganho = \sum_{t=0}^{n} w_i x_i$
- $o_d = \sigma(ganho_d)$ é a saída da função de transferência do neurônio respondendo ao d-ésimo exemplo no conjunto D
- t_d é a resposta esperada para o d-ésimo exemplo
- E é o erro da rede
- t_d não varia de acordo com w_i, então $\frac{\partial t_d}{\partial w_i} = 0$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \text{ ,usa regra da soma e obtém:}$$

$$= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \text{ ,usa regra da cadeia e obtém:}$$

Gradiente de erro para uma unidade sigmóide

- $ganho = \sum_{t=0}^{n} w_i x_i$
- $o_d = \sigma(ganho_d)$ é a saída da função de transferência do neurônio respondendo ao d-ésimo exemplo no conjunto D
- t_d é a resposta esperada para o d-ésimo exemplo
- E é o erro da rede
- t_d não varia de acordo com w_i, então $\frac{\partial t_d}{\partial w_i} = 0$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$,usa regra da soma e obtém:

$$= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2$$,usa regra da cadeia e obtém:

$$= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$, aplica derivada de constante e obtém:

Gradiente de erro para uma unidade sigmóide

- $ganho = \sum_{t=0}^{n} w_i x_i$
- $o_d = \sigma(ganho_d)$ é a saída da função de transferência do neurônio respondendo ao d-ésimo exemplo no conjunto D
- t_d é a resposta esperada para o d-ésimo exemplo
- E é o erro da rede
- t_d não varia de acordo com w_i, então $\frac{\partial t_d}{\partial w_i} = 0$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \text{ ,usa regra da soma e obtém:}$$

$$= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \text{ ,usa regra da cadeia e obtém:}$$

$$= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \text{ , aplica derivada de constante e obtém:}$$

$$= \sum_d (t_d - o_d) \left(-\frac{\partial o_d}{\partial w_i} \right) \text{ ,usa regra da cadeia e obtém:}$$

Gradiente de erro para uma unidade sigmóide

- $ganho = \sum_{t=0}^{n} w_i x_i$
- $o_d = \sigma(ganho_d)$ é a saída da função de transferência do neurônio respondendo ao d-ésimo exemplo no conjunto D
- t_d é a resposta esperada para o d-ésimo exemplo
- E é o erro da rede
- t_d não varia de acordo com w_i, então $\frac{\partial t_d}{\partial w_i} = 0$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$,usa regra da soma e obtém:

$$= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2$$,usa regra da cadeia e obtém:

$$= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$, aplica derivada de constante e obtém:

$$= \sum_d (t_d - o_d) \left(-\frac{\partial o_d}{\partial w_i} \right)$$,usa regra da cadeia e obtém:

$$\mathbf{\frac{\partial E}{\partial w_i} = -\sum_d (t_d - o_d) \frac{\partial o_d}{\partial ganho_d} \frac{\partial ganho_d}{\partial w_i}}$$

- Da equação que relaciona o erro E e os pesos da rede w_i:

$$\frac{\partial \mathbf{E}}{\partial \mathbf{w_i}} = -\sum_{\mathbf{d}} (\mathbf{t_d} - \mathbf{o_d}) \frac{\partial \mathbf{o_d}}{\partial \mathbf{ganho_d}} \frac{\partial \mathbf{ganho_d}}{\partial \mathbf{w_i}}$$

- Falta obter $\frac{\partial \mathbf{o_d}}{\partial \mathbf{ganho_d}}$ e $\frac{\partial \mathbf{ganho_d}}{\partial \mathbf{w_i}}$

- Da equação que relaciona o erro E e os pesos da rede w_i:

$$\frac{\partial \mathbf{E}}{\partial \mathbf{w_i}} = -\sum_{\mathbf{d}} (\mathbf{t_d} - \mathbf{o_d}) \frac{\partial \mathbf{o_d}}{\partial \mathbf{ganho_d}} \frac{\partial \mathbf{ganho_d}}{\partial \mathbf{w_i}}$$

- Falta obter $\frac{\partial \mathbf{o_d}}{\partial \mathbf{ganho_d}}$ e $\frac{\partial \mathbf{ganho_d}}{\partial \mathbf{w_i}}$

Lembrando que $o_d = \sigma(ganho_d)$ e $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$, temos:

$$\frac{\partial \mathbf{o_d}}{\partial \mathbf{ganho_d}} = \frac{\partial \sigma(ganho_d)}{\partial ganho_d} = o_d(1 - o_d)$$

- Da equação que relaciona o erro E e os pesos da rede w_i:

$$\frac{\partial \mathbf{E}}{\partial \mathbf{w_i}} = -\sum_{\mathbf{d}} (\mathbf{t_d} - \mathbf{o_d}) \frac{\partial \mathbf{o_d}}{\partial \mathbf{ganho_d}} \frac{\partial \mathbf{ganho_d}}{\partial \mathbf{w_i}}$$

- Falta obter $\frac{\partial \mathbf{o_d}}{\partial \mathbf{ganho_d}}$ e $\frac{\partial \mathbf{ganho_d}}{\partial \mathbf{w_i}}$

Lembrando que $o_d = \sigma(ganho_d)$ e $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$, temos:

$$\frac{\partial \mathbf{o_d}}{\partial \mathbf{ganho_d}} = \frac{\partial \sigma(ganho_d)}{\partial ganho_d} = o_d(1 - o_d)$$

E lembrando que $ganho_d = \vec{w} \cdot \vec{x_d}$, temos do segundo termo:

$$\frac{\partial \mathbf{ganho_d}}{\partial \mathbf{w_i}} = \frac{\partial \vec{w} \cdot \vec{x}_d}{\partial w_i} = x_{i,d}$$

- Da equação que relaciona o erro E e os pesos da rede w_i:

$$\frac{\partial \mathbf{E}}{\partial \mathbf{w_i}} = -\sum_{\mathbf{d}} (\mathbf{t_d} - \mathbf{o_d}) \frac{\partial \mathbf{o_d}}{\partial \mathbf{ganho_d}} \frac{\partial \mathbf{ganho_d}}{\partial \mathbf{w_i}}$$

- Falta obter $\frac{\partial \mathbf{o_d}}{\partial \mathbf{ganho_d}}$ e $\frac{\partial \mathbf{ganho_d}}{\partial \mathbf{w_i}}$

Lembrando que $o_d = \sigma(ganho_d)$ e $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$, temos:

$$\frac{\partial \mathbf{o_d}}{\partial \mathbf{ganho_d}} = \frac{\partial \sigma(ganho_d)}{\partial ganho_d} = o_d(1 - o_d)$$

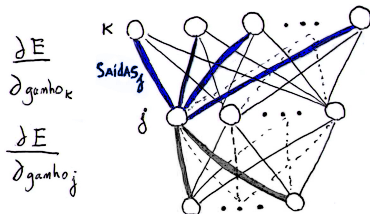
E lembrando que $ganho_d = \vec{w} \cdot \vec{x_d}$, temos do segundo termo:

$$\frac{\partial \mathbf{ganho_d}}{\partial \mathbf{w_i}} = \frac{\partial \vec{w} \cdot \vec{x}_d}{\partial w_i} = x_{i,d}$$

Então, a parte da "culpa" do erro E relativa ao peso w_i é

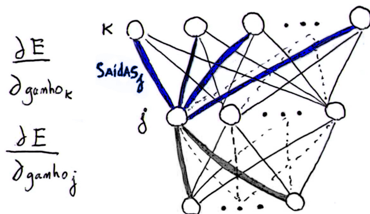
$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{i,d}$$

- Definição: $\delta_k = -\frac{\partial E}{\partial ganho_k}$ será a parte do erro passada às camadas internas
- δ_k é obtido desde a última camada até a de entrada



$$\frac{\partial E}{\partial ganho_j} = \sum_{k \in Saidas(j)} \frac{\partial E}{\partial ganho_k} \frac{\partial ganho_k}{\partial ganho_j}$$

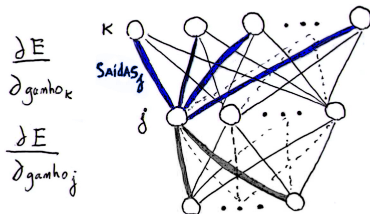
- Definição: $\delta_k = -\frac{\partial E}{\partial ganho_k}$ será a parte do erro passada às camadas internas
- δ_k é obtido desde a última camada até a de entrada



$$\frac{\partial E}{\partial ganho_j} = \sum_{k \in Saidas(j)} \frac{\partial E}{\partial ganho_k} \frac{\partial ganho_k}{\partial ganho_j}$$

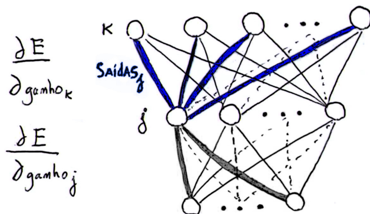
$$= \sum_{k \in Saidas(j)} -\delta_k \frac{\partial ganho_k}{\partial ganho_j}$$

- Definição: $\delta_k = -\frac{\partial E}{\partial ganho_k}$ será a parte do erro passada às camadas internas
- δ_k é obtido desde a última camada até a de entrada



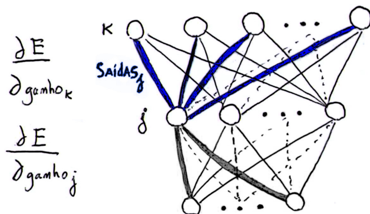
$$
\begin{aligned}
\frac{\partial E}{\partial ganho_j} &= \sum_{k \in Saidas(j)} \frac{\partial E}{\partial ganho_k} \frac{\partial ganho_k}{\partial ganho_j} \\
&= \sum_{k \in Saidas(j)} -\delta_k \frac{\partial ganho_k}{\partial ganho_j} \\
&= \sum_{k \in Saidas(j)} -\delta_k \frac{\partial ganho_k}{\partial o_j} \frac{\partial o_j}{\partial ganho_j}
\end{aligned}
$$

- Definição: $\delta_k = -\frac{\partial E}{\partial ganho_k}$ será a parte do erro passada às camadas internas
- δ_k é obtido desde a última camada até a de entrada



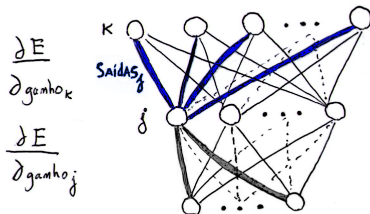
$$\begin{aligned}
\frac{\partial E}{\partial ganho_j} &= \sum_{k \in Saidas(j)} \frac{\partial E}{\partial ganho_k} \frac{\partial ganho_k}{\partial ganho_j} \\
&= \sum_{k \in Saidas(j)} -\delta_k \frac{\partial ganho_k}{\partial ganho_j} \\
&= \sum_{k \in Saidas(j)} -\delta_k \frac{\partial ganho_k}{\partial o_j} \frac{\partial o_j}{\partial ganho_j} \\
&= \sum_{k \in Saidas(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial ganho_j}
\end{aligned}$$

- Definição: $\delta_k = -\frac{\partial E}{\partial ganho_k}$ será a parte do erro passada às camadas internas
- δ_k é obtido desde a última camada até a de entrada



$$\begin{aligned}
\frac{\partial E}{\partial ganho_j} &= \sum_{k \in Saidas(j)} \frac{\partial E}{\partial ganho_k} \frac{\partial ganho_k}{\partial ganho_j} \\
&= \sum_{k \in Saidas(j)} -\delta_k \frac{\partial ganho_k}{\partial ganho_j} \\
&= \sum_{k \in Saidas(j)} -\delta_k \frac{\partial ganho_k}{\partial o_j} \frac{\partial o_j}{\partial ganho_j} \\
&= \sum_{k \in Saidas(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial ganho_j} \\
&= \sum_{k \in Saidas(j)} -\delta_k w_{kj} o_j (1 - o_j)
\end{aligned}$$

- Definição: $\delta_k = -\frac{\partial E}{\partial ganho_k}$ será a parte do erro passada às camadas internas
- δ_k é obtido desde a última camada até a de entrada



$$\begin{aligned}
\frac{\partial E}{\partial ganho_j} &= \sum_{k \in Saidas(j)} \frac{\partial E}{\partial ganho_k} \frac{\partial ganho_k}{\partial ganho_j} \\
&= \sum_{k \in Saidas(j)} -\delta_k \frac{\partial ganho_k}{\partial ganho_j} \\
&= \sum_{k \in Saidas(j)} -\delta_k \frac{\partial ganho_k}{\partial o_j} \frac{\partial o_j}{\partial ganho_j} \\
&= \sum_{k \in Saidas(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial ganho_j} \\
&= \sum_{k \in Saidas(j)} -\delta_k w_{kj} o_j (1 - o_j)
\end{aligned}$$

$$\delta_{\mathbf{j}} = -\frac{\partial \mathbf{E}}{\partial \mathbf{ganho_j}} = \mathbf{o_j (1 - o_j)} \sum_{\mathbf{k \in Saidas(j)}} \delta_{\mathbf{k}} \mathbf{w_{kj}}$$

Algoritmo de retropagação

- Inicializar todos os pesos para valores aleatórios pequenos.
- Até convergência, faça para cada exemplo de treino
  1. Apresente exemplo à rede e compute a saída da rede
  2. Para cada unidade de saída k

  $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

  3. Para cada unidade interna h

  $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in saidas} \delta_k w_{hk}$$

  4. Atualizar cada peso da rede w_{ij}

  $$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

  onde $\Delta w_{ij} = \eta \delta_j x_{ij}$

Mais em retroprogação

- Gradiente descendente sobre toda rede de vetores de pesos
- Generalizável para grafos direcionados (redes neurais recorrentes)
- Encontra mínimo local
- Funciona bem na prática ao rodar várias vezes
- Frequentemente inclui um *momentum* do peso α
  $\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$
- Minimiza erro nos exemplos de treino
  - necessário cuidado para evitar overfitting
- Treino pode ser lento, mas usar a rede treinada é rápido

Capacidade de representação de redes neurais

Funções booleanas

- Toda função booleana pode ser representada por uma rede com apenas uma camada interna
- Mas pode ser necessário um número exponencial de unidades internas em relação ao número de entradas

Funções contínuas

- Toda função contínua compacta pode ser aproximada com erro arbitrariamente pequeno por rede com uma camada interna
- Qualquer função pode ser aproximada com acurácia arbitrária por uma rede com duas camadas internas

Evitando overfitting: opções

- Penalizar pesos grandes:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in saidas} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

- Treino em inclinações alvo e em valores:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in saidas} \left[(t_{kd} - o + kd)^2 + \mu \sum_{j \in entradas} (\frac{\partial t_{kd}}{\partial x_d^j} - \frac{\partial o_{kd}}{\partial x_d^j})^2 \right]$$

- Compartilhamento de pesos
- Critério de parada prematura

Precursor de deep learning

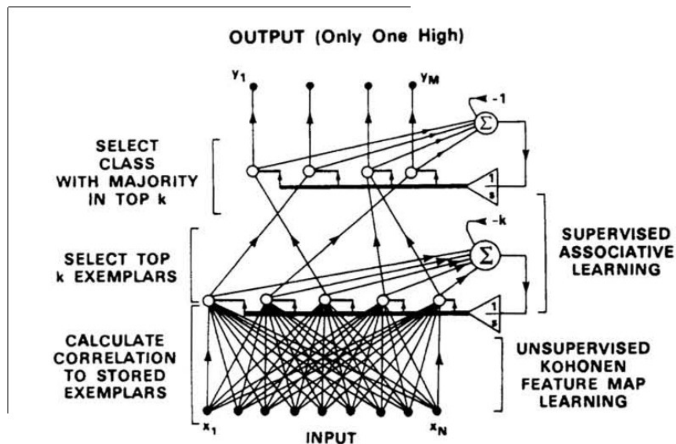


Figura: Fonte: Neural net and traditional classifiers de Huang& Lippmann (1988).

Deep learning

- Redes com várias camadas intermediárias
- Objetivo: atingir níveis mais "profundos" de abstração
- Custo de treino relativamente alto
- Uso com grandes bases de dados
- Resultados empíricos e pouco teóricos
- Referência: Deep Learning Tutorial: `deeplearning.net/tutorial/deeplearning.pdf`

1. Para cada camada
   1.1 Pré-treinar separadamente com algoritmo não supervisionado
   1.2 Empilhar nas camadas previamente treinadas e refinar com retropropagação

- Uso de técnicas para melhoria do aprendizado.

Técnicas: inicialização

Para usar tanh como função de ativação, inicializar pesos no seguinte intervalo:

$$[-\sqrt{\frac{6}{\text{fan}_{in} + \text{fan}_{out}}}, \sqrt{\frac{6}{\text{fan}_{in} + \text{fan}_{out}}}]$$

Facilita propagação e correção de erros.

Técnicas: taxa de aprendizado

- Varredura em $10^{-1}, 10^{-2}, \ldots,$ e concentrar no intervalo de menor erro de validação
- Descrescente $\frac{\mu_0}{1+d \times t}$, com μ_0 inicial e d sendo uma constante de decréscimo

Técnicas: atualização de pesos em **minibatch**

- Intermediário entre modo estocástico e em lote.
- Estimativa grosseira do gradiente.
- Ajuda a reduzir custo computacional.
- É comum variar de acordo com o número de épocas usado.

Técnica: interromper treino antes de overfitting

- Usar conjunto de validação.
- Verificar periodicamente o desempenho no conjunto de validação. Quando piorar, pára.
- A verificação pode envolver usar teste de hipótese ou uma simples comparação.

Redes neurais: sumário

- Perceptrons
- Gradiente descendente
- Redes multi-camadas
- Retroprogação de erros
- Deep learning

Redes neurais: pacotes R

- `RSNNS::mlp` – vários métodos de backpropagation, múltiplas camadas
- `neuralnet::neuralnet` – inclui visualização, múltiplas camadas
- `nnet::nnet` – camada escondida única
- `monmlp` – MLP que mantém informação a priori sobre forma da função
- `RWeka` – faz conversão de dados factor-¿binário automática (uso simples)

Pacote RSNSS: pré-processamento

```
require(RSNNS)
#aleatoriza ordem dos exemplos
iris <- iris[sample(1:nrow(iris),
                    length(1:nrow(iris))),1:ncol(iris)]
irisValues <- iris[,1:4]
irisTargets<-decodeClassLabels(iris[,5],
                               valTrue=1,valFalse=0)
summary(decodeClassLabels(iris[,5]))

##      setosa          versicolor        virginica     
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000  
##  Median :0.0000   Median :0.0000   Median :0.0000  
##  Mean   :0.3333   Mean   :0.3333   Mean   :0.3333  
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000  
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
```

Pré-processamento para redes neurais

```r
iris <- splitForTrainingAndTest(irisValues, irisTargets,
                                 ratio=0.15)
summary(iris)
```

```
##              Length Class  Mode
## inputsTrain  508    -none- numeric
## targetsTrain 381    -none- numeric
## inputsTest    92    -none- numeric
## targetsTest   69    -none- numeric
```

Pré-processamento para redes neurais

```
summary(iris$inputsTrain) # antes da normalizacao

##   Sepal.Length    Sepal.Width     Petal.Length
##  Min.   :4.300   Min.   :2.000   Min.   :1.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.550
##  Median :5.700   Median :3.000   Median :4.300
##  Mean   :5.813   Mean   :3.053   Mean   :3.719
##  3rd Qu.:6.400   3rd Qu.:3.350   3rd Qu.:5.100
##  Max.   :7.900   Max.   :4.400   Max.   :6.900
##   Petal.Width
##  Min.   :0.100
##  1st Qu.:0.300
##  Median :1.300
##  Mean   :1.185
##  3rd Qu.:1.800
##  Max.   :2.500
```

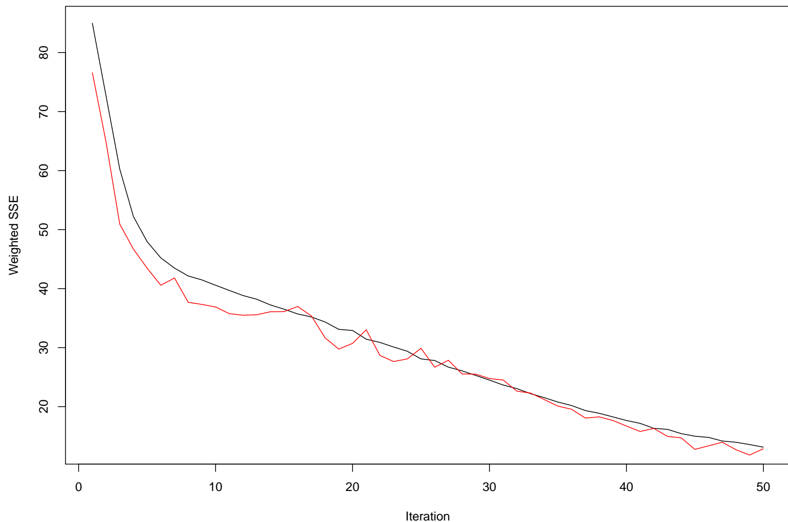
Pré-processamento para redes neurais

```r
iris <- normTrainingAndTestSet(iris)
summary(iris$inputsTrain) #atributos normalizados

##        V1                  V2                  V3
##  Min.   :-1.8004   Min.   :-2.4194   Min.   :-1.4907
##  1st Qu.:-0.8482   1st Qu.:-0.5809   1st Qu.:-1.2346
##  Median :-0.1340   Median :-0.1212   Median : 0.3308
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 0.6992   3rd Qu.: 0.6831   3rd Qu.: 0.7862
##  Max.   : 2.4845   Max.   : 3.0962   Max.   : 1.8107
##        V4
##  Min.   :-1.4210
##  1st Qu.:-1.1591
##  Median : 0.1506
##  Mean   : 0.0000
##  3rd Qu.: 0.8054
##  Max.   : 1.7221
```

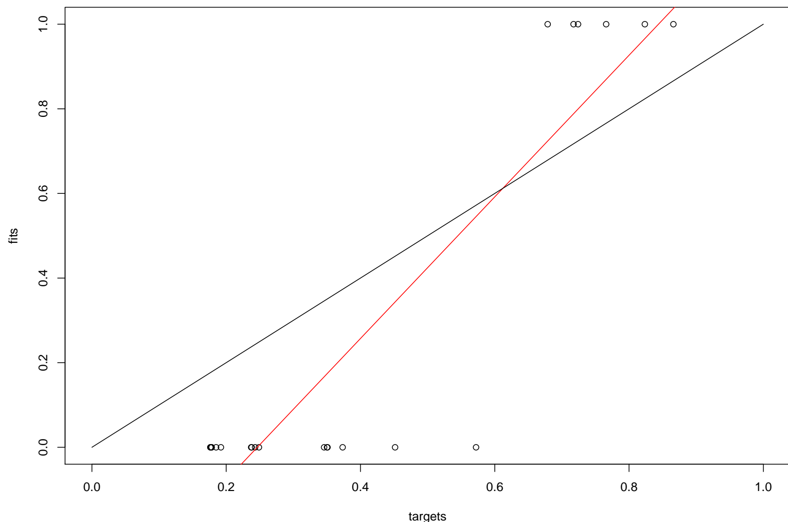
Treino de MLP

```
model <- mlp(iris$inputsTrain, iris$targetsTrain, size=5,
             learnFuncParams=c(0.13), maxit=50,
             inputsTest=iris$inputsTest,
             targetsTest=iris$targetsTest)
# summary(model) # testar
# weightMatrix(model)
# extractNetInfo(model)
model # learning rate = 0.13

## Class: mlp->rsnns
## Number of inputs: 4
## Number of outputs: 3
## Maximal iterations: 50
## Initialization function: Randomize_Weights
## Initialization function parameters: -0.3 0.3
## Learning function: Std_Backpropagation
## Learning function parameters: 0.13
```

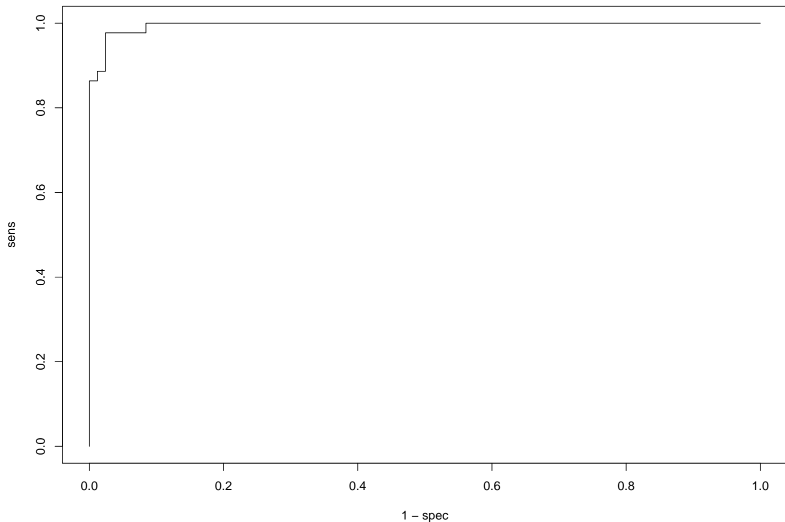
```
plotIterativeError(model)
```



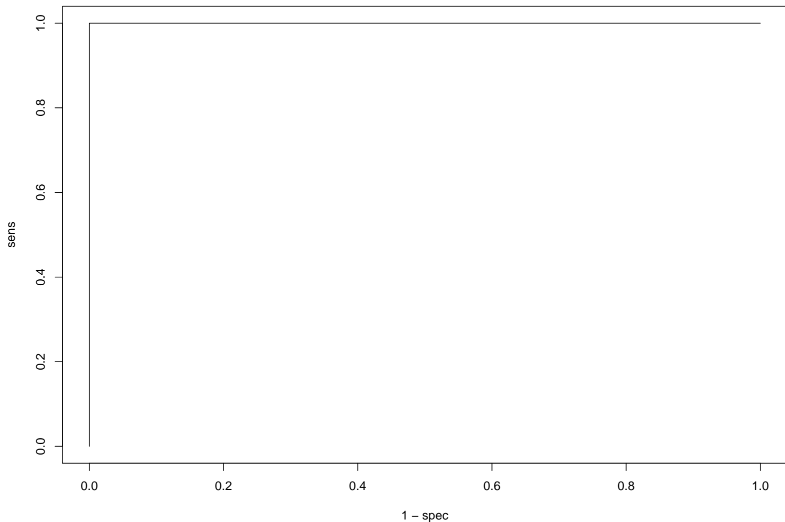
```r
predictions <- predict(model,iris$inputsTest)
plotRegressionError(predictions[,2], iris$targetsTest[,2])
```



```
plotROC(fitted.values(model)[,2], iris$targetsTrain[,2])
```



```
plotROC(predictions[,2], iris$targetsTest[,2])
```



```r
confusionMatrix(iris$targetsTrain,fitted.values(model))
```

```
##        predictions
## targets  1  2  3
##       1 43  0  0
##       2  0 42  2
##       3  0  2 38
```

```r
confusionMatrix(iris$targetsTest,predictions)
```

```
##        predictions
## targets 1 2 3
##       1 7 0 0
##       2 0 6 0
##       3 0 1 9
```

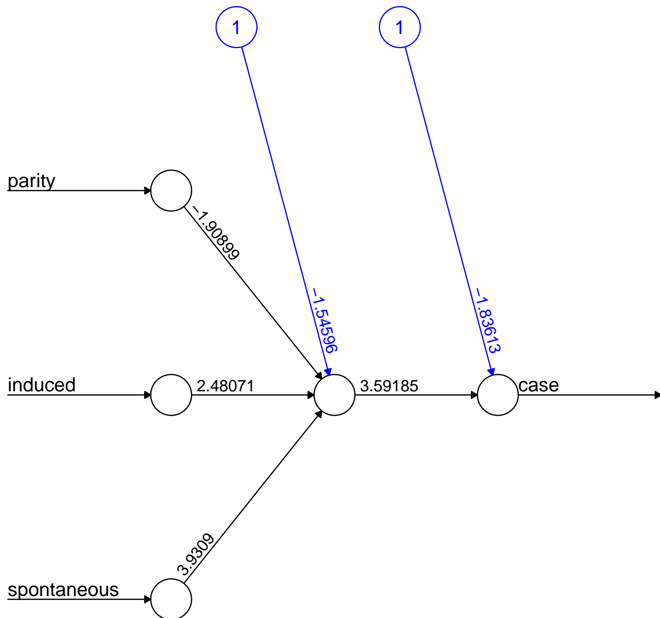
```r
# usa faixas [0,0.4] (0.4,0.6) e [0.6,1.0] para saídas
confusionMatrix(iris$targetsTrain,
                encodeClassLabels(fitted.values(model),
                method="402040", l=0.4, h=0.6))

##         predictions
## targets  0  1  2  3
##       1  0 43  0  0
##       2  5  0 39  0
##       3  6  0  1 33
```

Pacote neuralnet

```r
require(neuralnet)
data(infert) #dados sobre infertilidade
net.infert <- neuralnet(case~parity+induced+spontaneous,
                        infert, err.fct="ce",
                        linear.output=FALSE, likelihood=TRU
net.infert$result.matrix
```

```
##                                        1
## error                    129.247318862835
## reached.threshold          0.008693785732
## steps                    927.000000000000
## aic                      270.494637725670
## bic                      291.575210202660
## Intercept.to.1layhid1     -1.545959602887
## parity.to.1layhid1        -1.908986800945
## induced.to.1layhid1        2.480713862016
## spontaneous.to.1layhid1    3.930897516622
```

```r
plot(net.infert)
```



RWeka MultilayerPerceptron

```
require(RWeka)
wmlp<-make_Weka_classifier(
      "weka/classifiers/functions/MultilayerPerceptron")
#WOW(wmlp) - ## verificar  opções
data("HouseVotes84", package = "mlbench")
model <- wmlp(Class ~ ., data = HouseVotes84)
preds <-predict(model, HouseVotes84[, -1])
table(preds, HouseVotes84[,1])

##
## preds        democrat republican
##   democrat        264          4
##   republican        3        164
```