

Inteligência Artificial – ACH2016

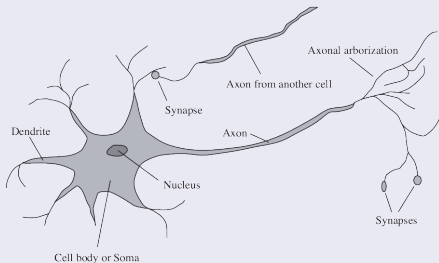
Aula 19 – Redes Neurais

Norton Trevisan Roman
(norton@usp.br)

23 de maio de 2019

Inspiração

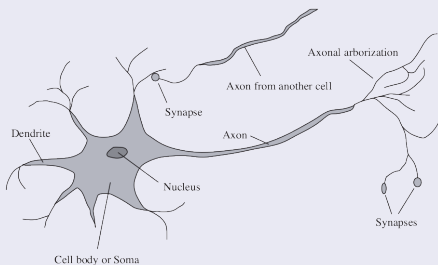
- O neurônio biológico



Fonte: AIMA. Russell & Norvig.

Inspiração

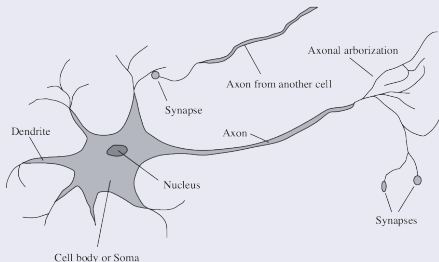
- O neurônio biológico
 - Um neurônio recebe sinais através de inúmeros dendritos, podendo ou não seguir adiante



Fonte: AIMA. Russell & Norvig.

Inspiração

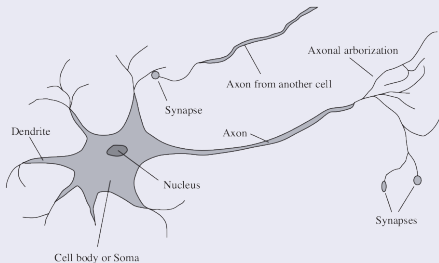
- O neurônio biológico
 - Um neurônio recebe sinais através de inúmeros dendritos, podendo ou não seguir adiante
 - Se o sinal for superior a um certo limite (threshold), segue em frente; caso contrário, é bloqueado



Fonte: AIMA. Russell & Norvig.

Inspiração

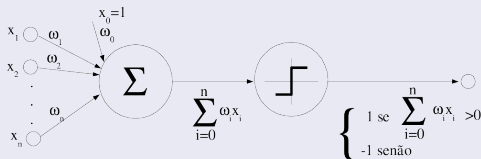
- O neurônio biológico
 - Um neurônio recebe sinais através de inúmeros dendritos, podendo ou não seguir adiante
 - Se o sinal for superior a um certo limite (threshold), segue em frente; caso contrário, é bloqueado
 - Na passagem por um neurônio, um sinal pode ser amplificado ou atenuado, dependendo do dendrito de origem



Fonte: AIMA. Russell & Norvig.

Redes Neurais

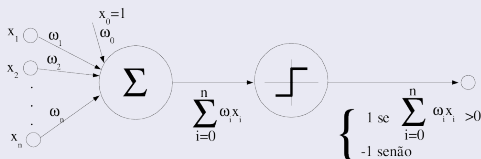
Perceptron



Fonte: Adaptado de de ML. Mitchell.

Perceptron

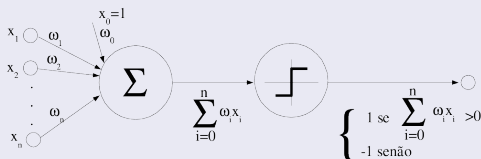
- É a unidade básica da rede neural



Fonte: Adaptado de de ML. Mitchell.

Perceptron

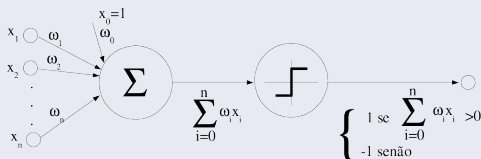
- É a unidade básica da rede neural
 - A mais simples



Fonte: Adaptado de de ML. Mitchell.

Perceptron

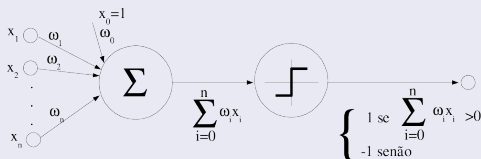
- É a unidade básica da rede neural
- A mais simples
- Um nó em um grafo dirigido (a rede neural)



Fonte: Adaptado de de ML. Mitchell.

Perceptron

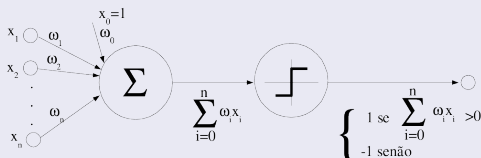
- É a unidade básica da rede neural
- A mais simples
- Um nó em um grafo dirigido (a rede neural)
- Cada aresta serve para propagar a ativação de um nó a outro na rede



Fonte: Adaptado de de ML. Mitchell.

Perceptron

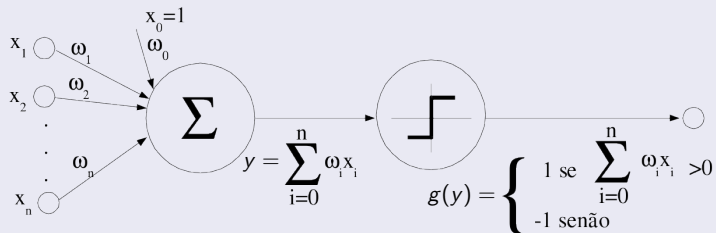
- É a unidade básica da rede neural
- A mais simples
- Um nó em um grafo dirigido (a rede neural)
- Cada aresta serve para propagar a ativação de um nó a outro na rede
 - Possui um peso associado, determinado a força e polaridade da conexão



Fonte: Adaptado de de ML. Mitchell.

Perceptron

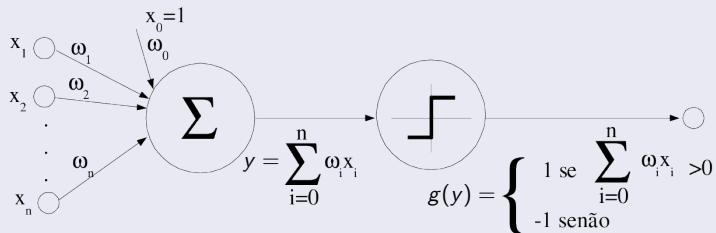
- Cada unidade calcula a soma ponderada de suas entradas



Fonte: Adaptado de ML. Mitchell.

Perceptron

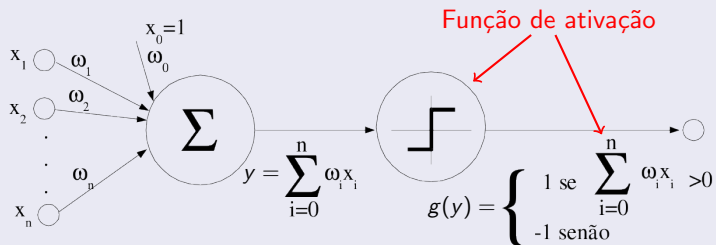
- Cada unidade calcula a soma ponderada de suas entradas
- E aplica então uma **função de ativação** para obter a saída



Fonte: Adaptado de ML. Mitchell.

Perceptron

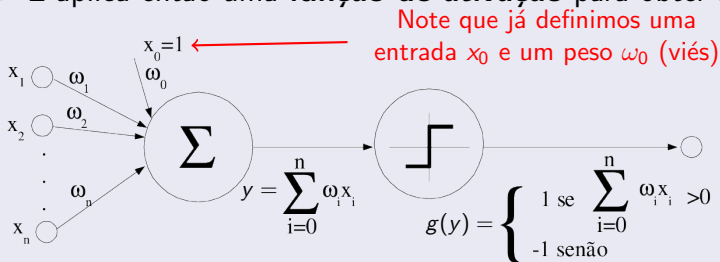
- Cada unidade calcula a soma ponderada de suas entradas
- E aplica então uma **função de ativação** para obter a saída



Fonte: Adaptado de ML. Mitchell.

Perceptron

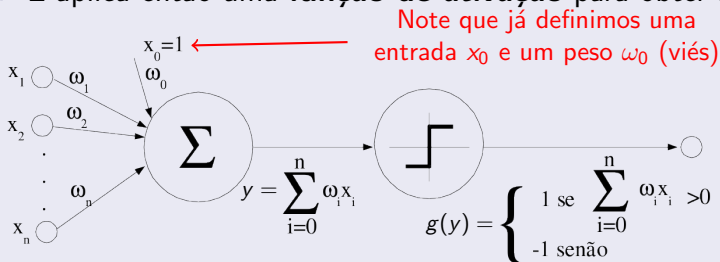
- Cada unidade calcula a soma ponderada de suas entradas
- E aplica então uma **função de ativação** para obter a saída



Fonte: Adaptado de ML. Mitchell.

Perceptron

- Cada unidade calcula a soma ponderada de suas entradas
- E aplica então uma **função de ativação** para obter a saída



Fonte: Adaptado de ML. Mitchell.

O aprendizado em um perceptron envolve escolher valores para os pesos

Perceptron – Função de Ativação

Perceptron – Função de Ativação

- Deve “ativar” o neurônio (valor perto de 1) quando as entradas corretas forem dadas

Perceptron – Função de Ativação

- Deve “ativar” o neurônio (valor perto de 1) quando as entradas corretas forem dadas
- Deve “desativar” o neurônio (próximo de 0) quando as entradas “erradas” forem dadas

Perceptron – Função de Ativação

- Deve “ativar” o neurônio (valor perto de 1) quando as entradas corretas forem dadas
- Deve “desativar” o neurônio (próximo de 0) quando as entradas “erradas” forem dadas
- Ou seja, o perceptron devolve 1 se o resultado é maior que algum limiar e 0 (ou -1) se não

Perceptron – Função de Ativação

- Deve “ativar” o neurônio (valor perto de 1) quando as entradas corretas forem dadas
- Deve “desativar” o neurônio (próximo de 0) quando as entradas “erradas” forem dadas
- Ou seja, o perceptron devolve 1 se o resultado é maior que algum limiar e 0 (ou -1) se não
 - No exemplo anterior, o limiar é zero

Perceptron – Função de Ativação

- Deve “ativar” o neurônio (valor perto de 1) quando as entradas corretas forem dadas
 - Deve “desativar” o neurônio (próximo de 0) quando as entradas “erradas” forem dadas
- Ou seja, o perceptron devolve 1 se o resultado é maior que algum limiar e 0 (ou -1) se não
 - No exemplo anterior, o limiar é zero
- Deve ser não-linear

Perceptron – Função de Ativação

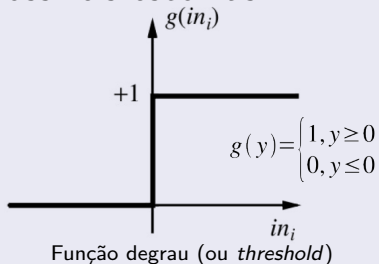
- Deve “ativar” o neurônio (valor perto de 1) quando as entradas corretas forem dadas
 - Deve “desativar” o neurônio (próximo de 0) quando as entradas “erradas” forem dadas
- Ou seja, o perceptron devolve 1 se o resultado é maior que algum limiar e 0 (ou -1) se não
 - No exemplo anterior, o limiar é zero
- Deve ser não-linear
 - Evitando que uma rede de neurônios resulte em uma única função linear

Perceptron – Função de Ativação

- Possíveis escolhas:

Perceptron – Função de Ativação

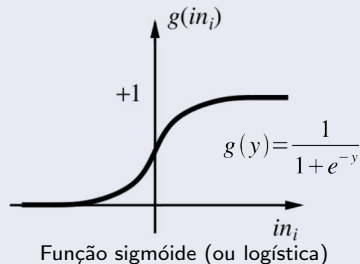
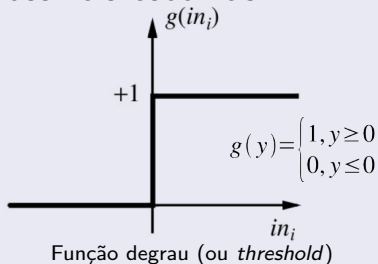
- Possíveis escolhas:



Fonte: Adaptado de Slides de AIMA. Russell & Norvig

Perceptron – Função de Ativação

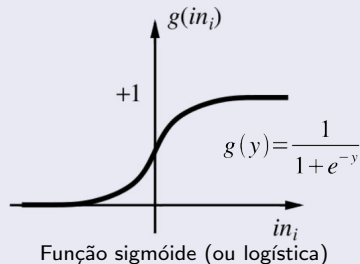
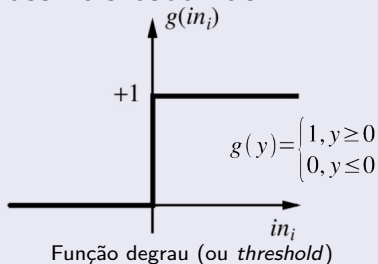
- Possíveis escolhas:



Fonte: Adaptado de Slides de AIMA. Russell & Norvig

Perceptron – Função de Ativação

- Possíveis escolhas:

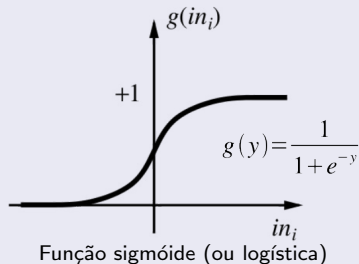
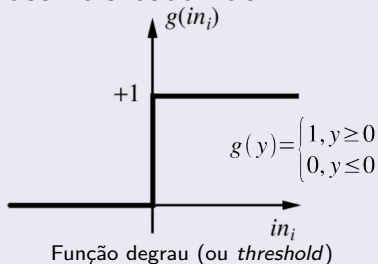


Fonte: Adaptado de Slides de AIMA. Russell & Norvig

- A sigmóide tem a vantagem de ser diferenciável

Perceptron – Função de Ativação

- Possíveis escolhas:

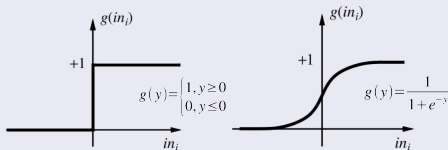


Fonte: Adaptado de Slides de AIMA. Russell & Norvig

- A sigmóide tem a vantagem de ser diferenciável
- Importante para o algoritmo de aprendizado dos pesos

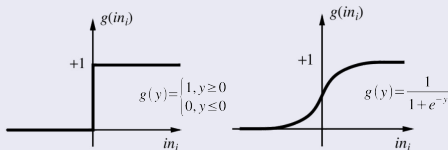
Perceptron – Função de Ativação

- Em ambas funções há um limiar em zero



Perceptron – Função de Ativação

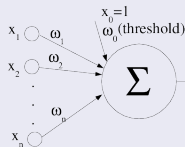
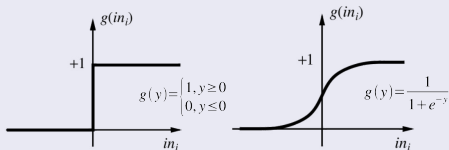
- Em ambas funções há um limiar em zero
- O valor real desse limiar é determinado pelo peso de viés ω_0 dado ao neurônio



Perceptron – Função de Ativação

- Em ambas funções há um limiar em zero
- O valor real desse limiar é determinado pelo peso de viés ω_0 dado ao neurônio
- A unidade é ativada caso a soma das entradas “reais” (ou seja, sem x_0) exceder ω_0 :

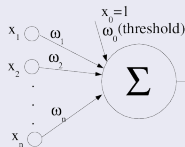
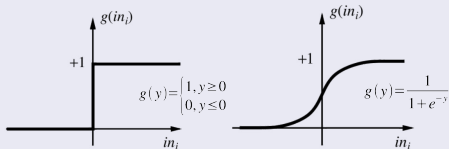
$$y = \left(\sum_{j=1}^n \omega_j x_j \right) \geq \omega_0$$



Redes Neurais

Perceptron – Função de Ativação

- Em ambas funções há um limiar em zero
- O valor real desse limiar é determinado pelo peso de viés ω_0 dado ao neurônio
- A unidade é ativada caso a soma das entradas “reais” (ou seja, sem x_0) exceder ω_0 :
$$y = \left(\sum_{j=1}^n \omega_j x_j \right) \geq \omega_0$$
- Mudar o peso de viés muda a localização do limiar



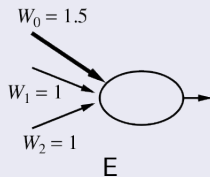
Perceptron

- Toda função booleana pode ser representada:

Perceptron

- Toda função booleana pode ser representada:

$$g(y) = \begin{cases} 1, & \left(y = \sum_{j=1}^n \omega_j x_j \right) \geq \omega_0 \\ 0, & \left(y = \sum_{j=1}^n \omega_j x_j \right) < \omega_0 \end{cases}$$



x_1	x_2	y	saída
1	1	2	1
1	0	1	0
0	1	1	0
0	0	0	0

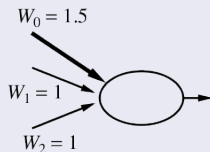
Fonte: Adaptado de AIMA. Russell & Norvig

Redes Neurais

Perceptron

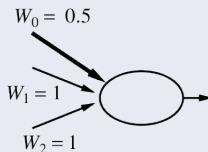
- Toda função booleana pode ser representada:

$$g(y) = \begin{cases} 1, & \left(y = \sum_{j=1}^n \omega_j x_j \right) \geq \omega_0 \\ 0, & \left(y = \sum_{j=1}^n \omega_j x_j \right) < \omega_0 \end{cases}$$



E

x_1	x_2	y	saída
1	1	2	1
1	0	1	0
0	1	1	0
0	0	0	0



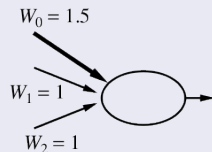
OU

x_1	x_2	y	saída
1	1	2	1
1	0	1	1
0	1	1	1
0	0	0	0

Fonte: Adaptado de AIMA. Russell & Norvig

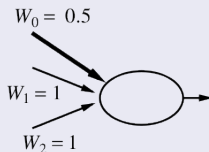
Perceptron

- Toda função booleana pode ser representada:



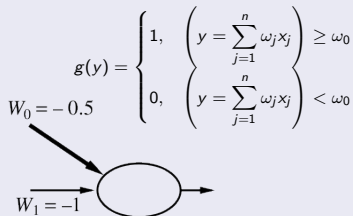
E

x_1	x_2	y	saída
1	1	2	1
1	0	1	0
0	1	1	0
0	0	0	0



OU

x_1	x_2	y	saída
1	1	2	1
1	0	1	1
0	1	1	1
0	0	0	0



NÃO

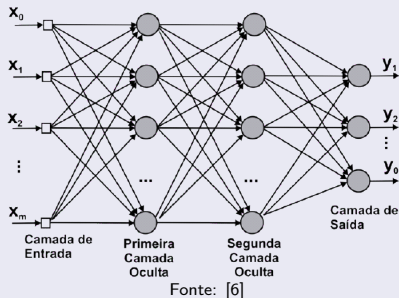
x_1	y	saída
1	-1	0
0	0	1

Fonte: Adaptado de AIMA. Russell & Norvig

Redes Neurais

Perceptrons em rede

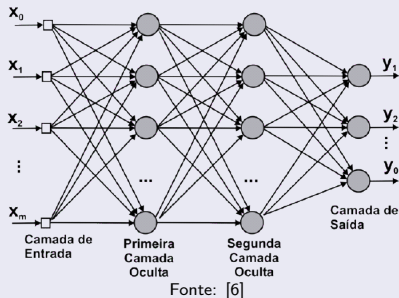
- Uma rede neural é uma rede de perceptrons conectados



Redes Neurais

Perceptrons em rede

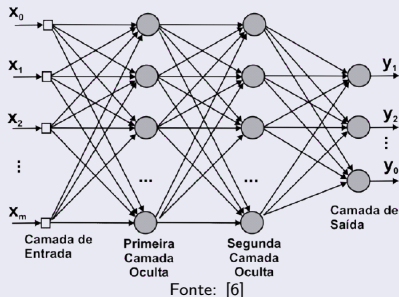
- Uma rede neural é uma rede de perceptrons conectados
- As unidades de saída dão a saída do programa



Redes Neurais

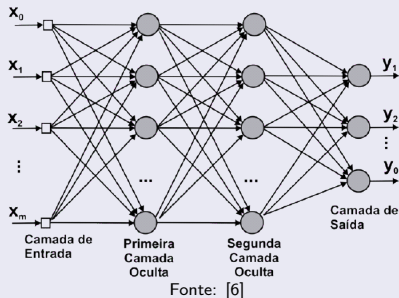
Perceptrons em rede

- Uma rede neural é uma rede de perceptrons conectados
- As unidades de saída dão a saída do programa
- Geralmente organizadas em camadas



Perceptrons em rede

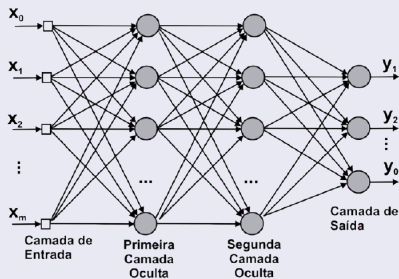
- Uma rede neural é uma rede de perceptrons conectados
- As unidades de saída dão a saída do programa
- Geralmente organizadas em camadas
 - Cada unidade recebe entrada somente de unidades da camada imediatamente anterior



Redes Neurais

Perceptrons em rede

- Possuem uma ou mais camadas de **unidades escondidas**:

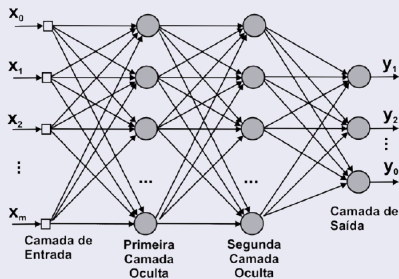


Fonte: [6]

Redes Neurais

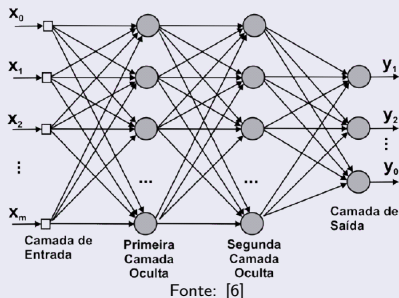
Perceptrons em rede

- Possuem uma ou mais camadas de **unidades escondidas**:
- Unidades que não estão conectadas à saída da rede



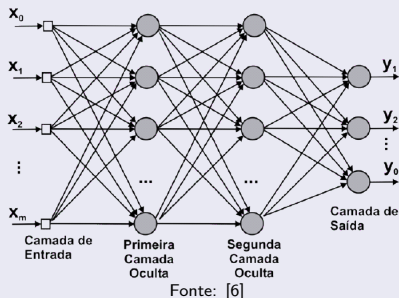
Perceptrons em rede

- Possuem uma ou mais camadas de **unidades escondidas**:
 - Unidades que não estão conectadas à saída da rede
- As camadas escondidas recebem sinal da entrada (ou de outra camada)



Perceptrons em rede

- Possuem uma ou mais camadas de **unidades escondidas**:
 - Unidades que não estão conectadas à saída da rede
- As camadas escondidas recebem sinal da entrada (ou de outra camada)
 - Sua saída, contudo, não é observada (por isso escondida)



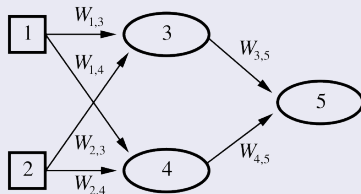
Redes Neurais

Tipos de redes

Redes Neurais

Tipos de redes

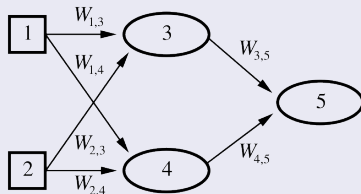
- *Feed-forward networks*



Fonte: Slides de AIMA. Russell & Norvig

Tipos de redes

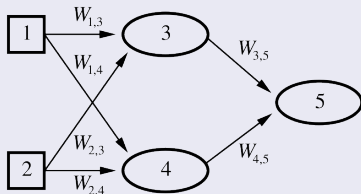
- *Feed-forward networks*
 - Possuem conexões em uma única direção



Fonte: Slides de AIMA. Russell & Norvig

Tipos de redes

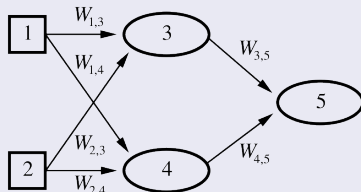
- *Feed-forward networks*
 - Possuem conexões em uma única direção
 - Formam um grafo dirigido acíclico



Fonte: Slides de AIMA. Russell & Norvig

Tipos de redes

- *Feed-forward networks*
 - Possuem conexões em uma única direção
 - Formam um grafo dirigido acíclico
 - Sua saída é uma função direta de sua entrada

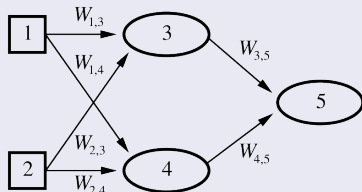


Fonte: Slides de AIMA. Russell & Norvig

$$\begin{aligned} s_5 &= g(\omega_{3,5}s_3 + \omega_{4,5}s_4) \\ &= g(\omega_{3,5} g(\omega_{1,3}x_1 + \omega_{2,3}x_2) + \\ &\quad \omega_{4,5} g(\omega_{1,4}x_1 + \omega_{2,4}x_2)) \end{aligned}$$

Tipos de redes

- *Feed-forward networks*
 - Possuem conexões em uma única direção
 - Formam um grafo dirigido acíclico
 - Sua saída é uma função direta de sua entrada
 - Não há outro estado interno que não os próprios pesos da rede

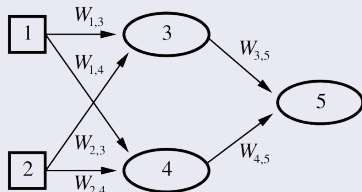


Fonte: Slides de AIMA. Russell & Norvig

$$\begin{aligned} s_5 &= g(\omega_{3,5}s_3 + \omega_{4,5}s_4) \\ &= g(\omega_{3,5} g(\omega_{1,3}x_1 + \omega_{2,3}x_2) + \omega_{4,5} g(\omega_{1,4}x_1 + \omega_{2,4}x_2)) \end{aligned}$$

Tipos de redes

- *Feed-forward networks*
 - Possuem conexões em uma única direção
 - Formam um grafo dirigido acíclico
 - Sua saída é uma função direta de sua entrada
 - Não há outro estado interno que não os próprios pesos da rede
 - Mudando os pesos, muda a função → aprende

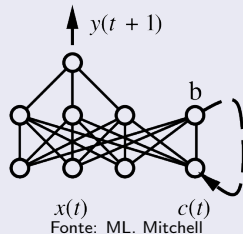


Fonte: Slides de AIMA. Russell & Norvig

$$\begin{aligned}s_5 &= g(\omega_{3,5}s_3 + \omega_{4,5}s_4) \\ &= g(\omega_{3,5} g(\omega_{1,3}x_1 + \omega_{2,3}x_2) + \omega_{4,5} g(\omega_{1,4}x_1 + \omega_{2,4}x_2))\end{aligned}$$

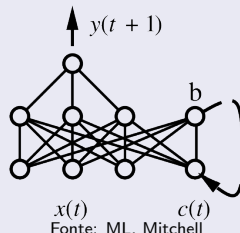
Tipos de redes

- Redes recorrentes



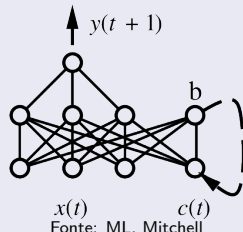
Tipos de redes

- Redes recorrentes
 - Redirecionam as saídas de algumas unidades à entrada de outras, formando um ciclo



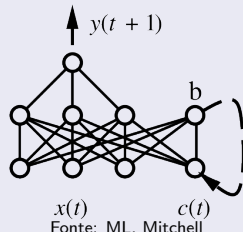
Tipos de redes

- Redes recorrentes
 - Redirecionam as saídas de algumas unidades à entrada de outras, formando um ciclo
 - Pode estabilizar, oscilar, ou apresentar comportamento caótico



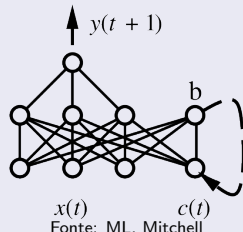
Tipos de redes

- Redes recorrentes
 - Redirecionam as saídas de algumas unidades à entrada de outras, formando um ciclo
 - Pode estabilizar, oscilar, ou apresentar comportamento caótico
 - Sua saída leva em consideração a entrada atual e o que aprendeu das entradas anteriores



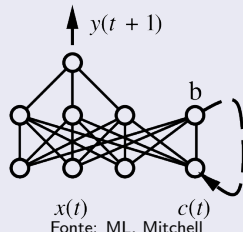
Tipos de redes

- Redes recorrentes
 - Redirecionam as saídas de algumas unidades à entrada de outras, formando um ciclo
 - Pode estabilizar, oscilar, ou apresentar comportamento caótico
 - Sua saída leva em consideração a entrada atual e o que aprendeu das entradas anteriores
 - A saída é copiada e realimentada na rede



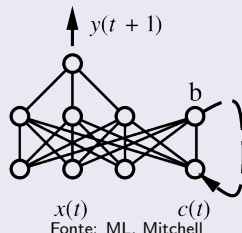
Tipos de redes

- Redes recorrentes
 - Redirecionam as saídas de algumas unidades à entrada de outras, formando um ciclo
 - Pode estabilizar, oscilar, ou apresentar comportamento caótico
 - Sua saída leva em consideração a entrada atual e o que aprendeu das entradas anteriores
 - A saída é copiada e realimentada na rede
 - Aqui, $c(t)$ é o valor de b no passo $t - 1$



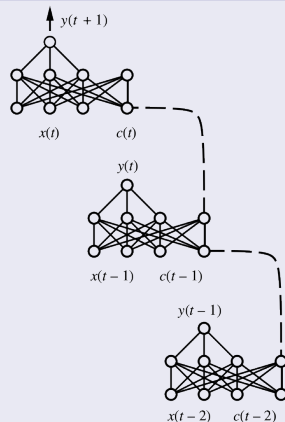
Tipos de redes

- Redes recorrentes
 - Redirecionam as saídas de algumas unidades à entrada de outras, formando um ciclo
 - Pode estabilizar, oscilar, ou apresentar comportamento caótico
- Sua saída leva em consideração a entrada atual e o que aprendeu das entradas anteriores
 - A saída é copiada e realimentada na rede
 - Aqui, $c(t)$ é o valor de b no passo $t - 1$
 - Podem simular memória de curto prazo (como um flip-flop)



Tipos de redes

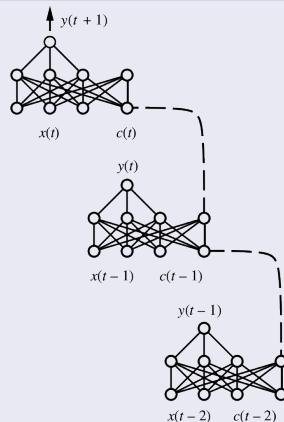
- Redes recorrentes (cont.)
 - Podem ser vistas como se desdobrando no tempo



Fonte: ML. Mitchell

Tipos de redes

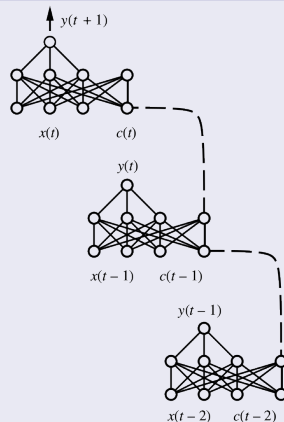
- Redes recorrentes (cont.)
- Podem ser vistas como se desdobrando no tempo
- Em que cada instante t dá o momento em que um exemplo de treino foi apresentado



Fonte: ML. Mitchell

Tipos de redes

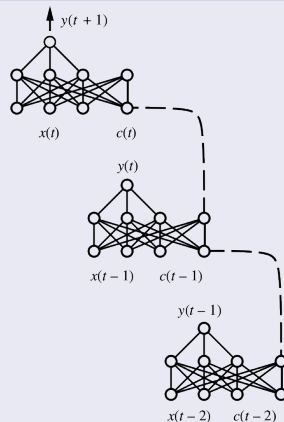
- Redes recorrentes (cont.)
 - Podem ser vistas como se desdobrando no tempo
 - Em que cada instante t dá o momento em que um exemplo de treino foi apresentado
 - Implementam uma relação de recorrência, na qual b representa o histórico das entradas



Fonte: ML. Mitchell

Tipos de redes

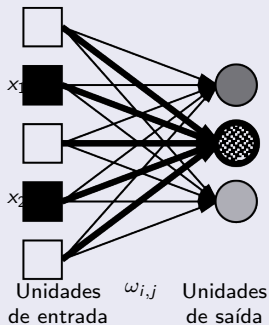
- Redes recorrentes (cont.)
 - Podem ser vistas como se desdobrando no tempo
 - Em que cada instante t dá o momento em que um exemplo de treino foi apresentado
 - Implementam uma relação de recorrência, na qual b representa o histórico das entradas
 - Várias topologias podem ser usadas. Essa é só um exemplo



Fonte: ML. Mitchell

Redes de Camada Única

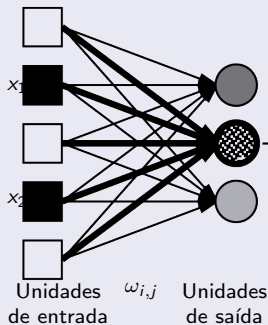
- Cada entrada se conecta diretamente à saída



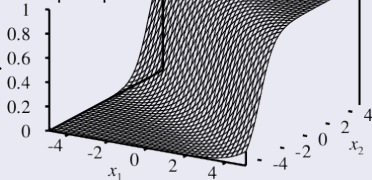
Fonte: Slides de AIMA. Russell & Norvig

Redes de Camada Única

- Cada entrada se conecta diretamente à saída



Saída do perceptron



Fonte: Slides de AIMA. Russell & Norvig

Redes de Camada Única

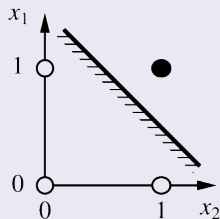
- Não representam todas as funções

Redes de Camada Única

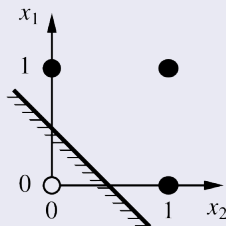
- Não representam todas as funções
 - Apenas as linearmente separáveis (pontos em que $g(y) = 0$ estão separados daqueles em que $g(y) = 1$ por uma reta)

Redes de Camada Única

- Não representam todas as funções
 - Apenas as linearmente separáveis (pontos em que $g(y) = 0$ estão separados daqueles em que $g(y) = 1$ por uma reta)

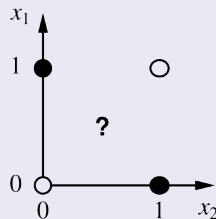


x_1 and x_2



x_1 or x_2

Fonte: AIMA. Russell & Norvig



x_1 xor x_2

Perceptron Learning Rule

Perceptron Learning Rule

- Regra para atualização dos pesos na rede, que garantidamente converge para uma solução

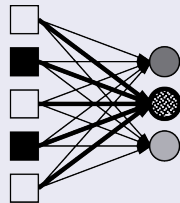
Perceptron Learning Rule

- Regra para atualização dos pesos na rede, que garantidamente converge para uma solução
- Um separador linear que classifica os dados perfeitamente, se eles forem linearmente separáveis e η pequeno

Perceptron Learning Rule

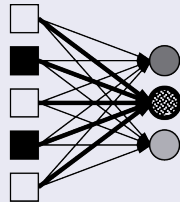
- Regra para atualização dos pesos na rede, que garantidamente converge para uma solução
 - Um separador linear que classifica os dados perfeitamente, se eles forem linearmente separáveis e η pequeno
- $\omega_i \leftarrow \omega_i + \Delta\omega_i$, com $\Delta\omega_i = \eta(t - s)x_i$, onde:
 - η – taxa de aprendizagem (*learning rate*)
 - t – saída desejada da rede para o exemplo corrente
 - s – saída do perceptron ($g(y)$) para o exemplo corrente
 - $(t - s)$ – Erro
 - x_i – cada atributo no exemplo corrente

Redes de Camada Única: Treinamento



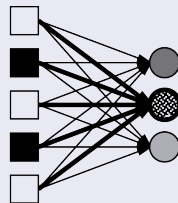
Redes de Camada Única: Treinamento

- Iniciamos com pesos aleatórios



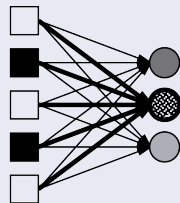
Redes de Camada Única: Treinamento

- Iniciamos com pesos aleatórios
- Aplicamos o perceptron iterativamente a cada exemplo de entrada



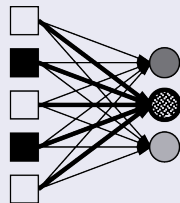
Redes de Camada Única: Treinamento

- Iniciamos com pesos aleatórios
- Aplicamos o perceptron iterativamente a cada exemplo de entrada
 - Mudando os pesos toda vez que ele errar



Redes de Camada Única: Treinamento

- Iniciamos com pesos aleatórios
- Aplicamos o perceptron iterativamente a cada exemplo de entrada
 - Mudando os pesos toda vez que ele errar
- Os pesos são modificados de acordo com a regra de treinamento do perceptron, que revisa os pesos ω_i :
$$\omega_i \leftarrow \omega_i + \Delta\omega_i, \text{ com } \Delta\omega_i = \eta(t - s)x_i$$

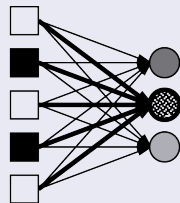


Redes de Camada Única: Treinamento

- Iniciamos com pesos aleatórios
- Aplicamos o perceptron iterativamente a cada exemplo de entrada
 - Mudando os pesos toda vez que ele errar
- Os pesos são modificados de acordo com a regra de treinamento do perceptron, que revisa os pesos ω_i :

$$\omega_i \leftarrow \omega_i + \Delta\omega_i, \text{ com } \Delta\omega_i = \eta(t - s)x_i$$

(aprende pelo ajuste dos pesos de modo a reduzir o erro no conjunto de treino)



Perceptron Rule: Taxa de Aprendizagem

Perceptron Rule: Taxa de Aprendizagem

- Modera o grau com que os pesos são mudados em cada passo

Perceptron Rule: Taxa de Aprendizagem

- Modera o grau com que os pesos são mudados em cada passo
- Geralmente baixa (0,1 por exemplo) e algumas vezes decai à medida que o número de iterações aumenta

Perceptron Rule: Taxa de Aprendizagem

- Modera o grau com que os pesos são mudados em cada passo
 - Geralmente baixa (0,1 por exemplo) e algumas vezes decai à medida que o número de iterações aumenta
- Ideia básica:

Perceptron Rule: Taxa de Aprendizagem

- Modera o grau com que os pesos são mudados em cada passo
 - Geralmente baixa (0,1 por exemplo) e algumas vezes decai à medida que o número de iterações aumenta
- Ideia básica:
 - Se o resultado (s) foi abaixo do desejado (t), temos que aumentar os pesos, pois a saída é proporcional à sua soma

Perceptron Rule: Taxa de Aprendizagem

- Modera o grau com que os pesos são mudados em cada passo
 - Geralmente baixa (0,1 por exemplo) e algumas vezes decai à medida que o número de iterações aumenta
- Ideia básica:
 - Se o resultado (s) foi abaixo do desejado (t), temos que aumentar os pesos, pois a saída é proporcional à sua soma
 - Fazemos $\Delta\omega_i > 0$, numa tentativa que o resultado da somatória dê maior que o *threshold*, gerando um 1

Perceptron Rule: Taxa de Aprendizagem

- Modera o grau com que os pesos são mudados em cada passo
 - Geralmente baixa (0,1 por exemplo) e algumas vezes decai à medida que o número de iterações aumenta
- Ideia básica:
 - Se o resultado (s) foi abaixo do desejado (t), temos que aumentar os pesos, pois a saída é proporcional à sua soma
 - Fazemos $\Delta\omega_i > 0$, numa tentativa que o resultado da somatória dê maior que o *threshold*, gerando um 1
 - η irá regular o tamanho desse aumento

Delta Rule

- A *Perceptron rule* é válida apenas para a função degrau

Delta Rule

- A *Perceptron rule* é válida apenas para a função degrau
- Para a sigmóide, a saída $g(y)$ é linear, e não binária

Delta Rule

- A *Perceptron rule* é válida apenas para a função degrau
- Para a sigmóide, a saída $g(y)$ é linear, e não binária
- $g(y) = \sigma(y) = \frac{1}{1 + e^{-y}}$, com $y = \sum_{i=0}^n \omega_i x_i$

Delta Rule

- A *Perceptron rule* é válida apenas para a função degrau
- Para a sigmóide, a saída $g(y)$ é linear, e não binária
- $g(y) = \sigma(y) = \frac{1}{1 + e^{-y}}$, com $y = \sum_{i=0}^n \omega_i x_i$
- Aplicamos então a **Delta Rule**:
 - $\omega_i \leftarrow \omega_i + \Delta\omega_i$, com $\Delta\omega_i = \eta(t - g(y))g'(y)x_i$

Delta Rule

- A *Perceptron rule* é válida apenas para a função degrau
- Para a sigmóide, a saída $g(y)$ é linear, e não binária
- $g(y) = \sigma(y) = \frac{1}{1 + e^{-y}}$, com $y = \sum_{i=0}^n \omega_i x_i$
- Aplicamos então a **Delta Rule**:
 - $\omega_i \leftarrow \omega_i + \Delta\omega_i$, com $\Delta\omega_i = \eta(t - g(y))g'(y)x_i$
 - Onde $g'(y)$ é a derivada da função de ativação:

$$g(y) = \frac{1}{1 + e^{-y}} \Rightarrow g'(y) = \frac{d}{dy}g(y) = g(y)(1 - g(y))$$

Redes de Camada Única: Algoritmo

Função *PERCEPTRON*(*Exemplos*, *Rede*): **rede neural**

repita

para cada $e \in \textit{Exemplos}$ **faça**

$$y \leftarrow \sum_{j=0}^n \omega_j x_j[e]$$

$$\textit{Erro} \leftarrow t[e] - g(y)$$

$$\omega_j \leftarrow \omega_j + \eta \times \textit{Erro} \times g'(y) \times x_j[e]$$

até *Até que algum critério de parada seja satisfeito*

retorna *A nova rede*

Redes de Camada Única: Algoritmo

Função *PERCEPTRON*(*Exemplos*, *Rede*): rede neural

repita

para cada $e \in \textit{Exemplos}$ **faça**

$$y \leftarrow \sum_{j=0}^n \omega_j x_j[e]$$

$$\textit{Erro} \leftarrow t[e] - g(y)$$

$$\omega_j \leftarrow \omega_j + \eta \times \textit{Erro} \times g'(y) \times x_j[e]$$

até Até que algum critério de parada seja satisfeito

retorna A nova rede

Conjunto de exemplos,
cada um com entrada
 $\vec{x} = \{x_1, \dots, x_n\}$ e saída \vec{t}

Redes de Camada Única: Algoritmo

Função *PERCEPTRON*(*Exemplos*, *Rede*): rede neural

repita

para cada $e \in \textit{Exemplos}$ **faça**

$$y \leftarrow \sum_{j=0}^n \omega_j x_j[e]$$


$$\textit{Erro} \leftarrow t[e] - g(y)$$

$$\omega_j \leftarrow \omega_j + \eta \times \textit{Erro} \times g'(y) \times x_j[e]$$

até Até que algum critério de parada seja satisfeito

retorna A nova rede

$\textit{rede} = \{\omega_1, \omega_2, \dots, \omega_k, g(y)\}$,
inicialmente com ω_j aleatórios



Redes de Camada Única: Algoritmo

Função *PERCEPTRON*(*Exemplos*, *Rede*): **rede neural**

repita

para cada $e \in \textit{Exemplos}$ **faça**

$$y \leftarrow \sum_{j=0}^n \omega_j x_j[e] \quad \longleftarrow \quad x_j[e] \rightarrow \text{j-ésimo atributo do exemplo } e$$

$$\textit{Erro} \leftarrow t[e] - g(y)$$

$$\omega_j \leftarrow \omega_j + \eta \times \textit{Erro} \times g'(y) \times x_j[e]$$

até *Até que algum critério de parada seja satisfeito*

retorna *A nova rede*

Redes de Camada Única: Algoritmo

Função *PERCEPTRON*(*Exemplos*, *Rede*): **rede neural**

repita

para cada $e \in \textit{Exemplos}$ **faça**

$$y \leftarrow \sum_{j=0}^n \omega_j x_j[e]$$

$$\textit{Erro} \leftarrow t[e] - g(y)$$

$$\omega_j \leftarrow \omega_j + \eta \times \textit{Erro} \times g'(y) \times x_j[e]$$

Note que podemos passar várias vezes pelo conjunto de treino

até *Até que algum critério de parada seja satisfeito*

retorna *A nova rede*

Redes de Camada Única: Algoritmo

Função *PERCEPTRON*(*Exemplos*, *Rede*): **rede neural**

repita

para cada $e \in \textit{Exemplos}$ **faça**

$$y \leftarrow \sum_{j=0}^n \omega_j x_j[e]$$

$$\textit{Erro} \leftarrow t[e] - g(y)$$

$$\omega_j \leftarrow \omega_j + \eta \times \textit{Erro} \times g'(y) \times x_j[e]$$

A cada passada pelo conjunto de treino damos o nome de **época** (*epoch*)

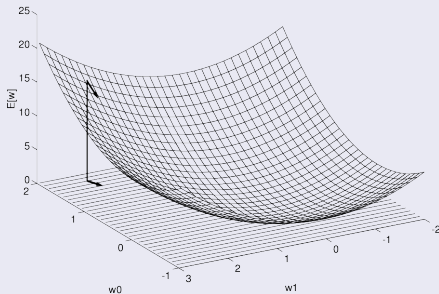
até *Até que algum critério de parada seja satisfeito*

retorna *A nova rede*

Derivação da Delta Rule

Derivação da Delta Rule

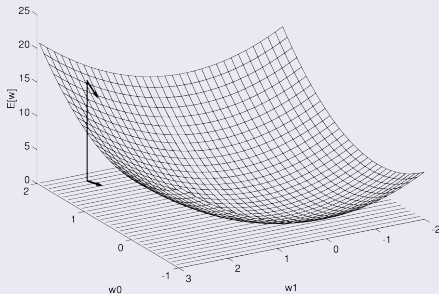
- Partimos do gradiente negativo do erro \rightarrow **Gradient Descent**



Fonte: ML. Mitchell.

Derivação da Delta Rule

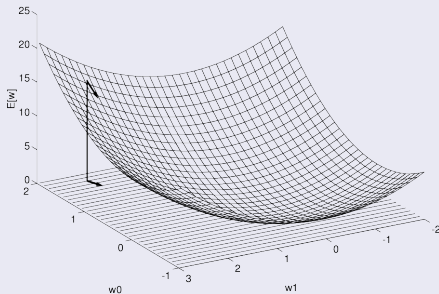
- Partimos do gradiente negativo do erro \rightarrow **Gradient Descent**
- O gradiente do erro dará um vetor que aponta para o sentido de maior crescimento dessa função no espaço



Fonte: ML. Mitchell.

Derivação da Delta Rule

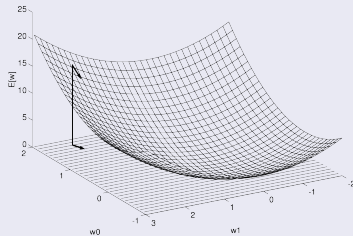
- Partimos do gradiente negativo do erro \rightarrow **Gradient Descent**
- O gradiente do erro dará um vetor que aponta para o sentido de maior crescimento dessa função no espaço
- Ou seja, o negativo do gradiente do erro dará o sentido da sua maior redução



Fonte: ML. Mitchell.

Derivação da Delta Rule

- Usamos então o negativo do gradiente do erro $E(\vec{w})$ para atualizar os pesos, de modo a minimizar esse erro

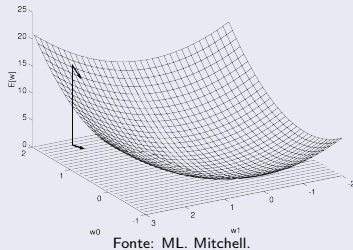


Fonte: ML. Mitchell.

Derivação da Delta Rule

- Usamos então o negativo do gradiente do erro $E(\vec{w})$ para atualizar os pesos, de modo a minimizar esse erro

$$\begin{cases} \vec{w} \leftarrow \vec{w} + \Delta\vec{w} \\ \Delta\vec{w} = -\eta \nabla E(\vec{w}) \end{cases}$$

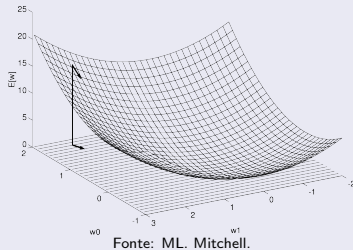


Derivação da Delta Rule

- Usamos então o negativo do gradiente do erro $E(\vec{\omega})$ para atualizar os pesos, de modo a minimizar esse erro

$$\begin{cases} \vec{\omega} \leftarrow \vec{\omega} + \Delta\vec{\omega} \\ \Delta\vec{\omega} = -\eta \nabla E(\vec{\omega}) \end{cases}$$

$$\text{Onde } \nabla E(\vec{\omega}) = \left[\frac{\partial E}{\partial \omega_0}, \frac{\partial E}{\partial \omega_1}, \dots, \frac{\partial E}{\partial \omega_n} \right]$$



Derivação da Delta Rule

- Considere uma unidade linear com saída $s = g(y)$

Derivação da Delta Rule

- Considere uma unidade linear com saída $s = g(y)$
- Onde $y = \sum_{i=0}^n \omega_i x_i$

Derivação da Delta Rule

- Considere uma unidade linear com saída $s = g(y)$

- Onde $y = \sum_{i=0}^n \omega_i x_i$

- Vamos aprender os ω_i que minimizam o **erro quadrático**:

$$E(\vec{\omega}) = \frac{1}{2} \sum_{d \in D} (t_d - s_d)^2$$

onde $s_d = g(y_d)$ e D é o conjunto de exemplos de treino

Derivação da Delta Rule

- Considere uma unidade linear com saída $s = g(y)$

- Onde $y = \sum_{i=0}^n \omega_i x_i$

- Vamos aprender os ω_i que minimizam o **erro quadrático**:

$$E(\vec{\omega}) = \frac{1}{2} \sum_{d \in D} (t_d - s_d)^2$$

Fator de escala do erro

onde $s_d = g(y_d)$ e D é o conjunto de exemplos de treino

Derivação da Delta Rule

- O gradiente do erro será

$$\nabla E(\vec{\omega}) = \left[\frac{\partial E}{\partial \omega_0}, \frac{\partial E}{\partial \omega_1}, \dots, \frac{\partial E}{\partial \omega_n} \right]$$

Derivação da Delta Rule

- O gradiente do erro será

$$\nabla E(\vec{\omega}) = \left[\frac{\partial E}{\partial \omega_0}, \frac{\partial E}{\partial \omega_1}, \dots, \frac{\partial E}{\partial \omega_n} \right]$$

$$\frac{\partial E}{\partial \omega_i} = \frac{\partial}{\partial \omega_i} \left(\frac{1}{2} \sum_{d \in D} (t_d - s_d)^2 \right)$$

Derivação da Delta Rule

- O gradiente do erro será

$$\nabla E(\vec{\omega}) = \left[\frac{\partial E}{\partial \omega_0}, \frac{\partial E}{\partial \omega_1}, \dots, \frac{\partial E}{\partial \omega_n} \right]$$

$$\begin{aligned} \frac{\partial E}{\partial \omega_i} &= \frac{\partial}{\partial \omega_i} \left(\frac{1}{2} \sum_{d \in D} (t_d - s_d)^2 \right) \\ &= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial \omega_i} (t_d - s_d)^2 \end{aligned}$$

Derivação da Delta Rule

- O gradiente do erro será

$$\nabla E(\vec{\omega}) = \left[\frac{\partial E}{\partial \omega_0}, \frac{\partial E}{\partial \omega_1}, \dots, \frac{\partial E}{\partial \omega_n} \right]$$

$$\frac{\partial E}{\partial \omega_i} = \frac{\partial}{\partial \omega_i} \left(\frac{1}{2} \sum_{d \in D} (t_d - s_d)^2 \right)$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial \omega_i} (t_d - s_d)^2$$

$$\frac{\partial E}{\partial \omega_i} = \sum_{d \in D} (t_d - s_d) \frac{\partial}{\partial \omega_i} (t_d - s_d)$$

Derivação da Delta Rule

- Como o valor esperado t_d independe de ω_i , então

$$\frac{\partial E}{\partial \omega_i} = - \sum_{d \in D} (t_d - s_d) \frac{\partial}{\partial \omega_i} s_d$$

Derivação da Delta Rule

- Como o valor esperado t_d independe de ω_i , então

$$\frac{\partial E}{\partial \omega_i} = - \sum_{d \in D} (t_d - s_d) \frac{\partial}{\partial \omega_i} s_d$$

- Se $s_d = \sum_i \omega_i x_i$, então

Derivação da Delta Rule

- Como o valor esperado t_d independe de ω_i , então

$$\frac{\partial E}{\partial \omega_i} = - \sum_{d \in D} (t_d - s_d) \frac{\partial}{\partial \omega_i} s_d$$

- Se $s_d = \sum_i \omega_i x_{i,d}$, então

$$\frac{\partial}{\partial \omega_i} E = - \sum_{d \in D} (t_d - s_d) x_{i,d}$$

Derivação da Delta Rule

- Como o valor esperado t_d independe de ω_i , então

$$\frac{\partial E}{\partial \omega_i} = - \sum_{d \in D} (t_d - s_d) \frac{\partial}{\partial \omega_i} s_d$$

- Se $s_d = \sum_i \omega_i x_{i,d}$, então

$$\frac{\partial}{\partial \omega_i} E = - \sum_{d \in D} (t_d - s_d) x_{i,d}$$

$$\text{e } \Delta \omega_i = \eta \sum_{d \in D} (t_d - s_d) x_{i,d} \text{ (perceptron rule)}$$

Derivação da Delta Rule

- Já se $s_d = \sigma(y_d)$, então $\frac{\partial s_d}{\partial \omega_i} = \frac{d s_d}{dy} \frac{\partial y}{\partial \omega_i} = s_d(1 - s_d)x_i$

Derivação da Delta Rule

- Já se $s_d = \sigma(y_d)$, então $\frac{\partial s_d}{\partial \omega_i} = \frac{d s_d}{dy} \frac{\partial y}{\partial \omega_i} = s_d(1 - s_d)x_i$

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

Derivação da Delta Rule

- Já se $s_d = \sigma(y_d)$, então $\frac{\partial s_d}{\partial \omega_i} = \frac{d s_d}{dy} \frac{\partial y}{\partial \omega_i} = s_d(1 - s_d)x_i$

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

$$\text{e } \Delta \omega_i = \eta \sum_{d \in D} (t_d - s_d) s_d (1 - s_d) x_{i,d} \text{ (delta rule)}$$

Derivação da Delta Rule

- Já se $s_d = \sigma(y_d)$, então $\frac{\partial s_d}{\partial \omega_i} = \frac{d s_d}{d y} \frac{\partial y}{\partial \omega_i} = s_d(1 - s_d)x_i$

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

$$\text{e } \Delta \omega_i = \eta \sum_{d \in D} (t_d - s_d) s_d (1 - s_d) x_{i,d} \text{ (delta rule)}$$

- Lembre que a cada iteração atualizamos os pesos:

$$\begin{cases} \vec{\omega} \leftarrow \vec{\omega} + \Delta \vec{\omega} \\ \Delta \vec{\omega} = -\eta \nabla E(\vec{\omega}) \end{cases} \quad \text{ou} \quad \begin{cases} \omega_i \leftarrow \omega_i + \Delta \omega_i \\ \Delta \omega_i = -\eta \frac{\partial E}{\partial \omega_i} \end{cases}$$

Redes Multicamadas

Redes Multicamadas

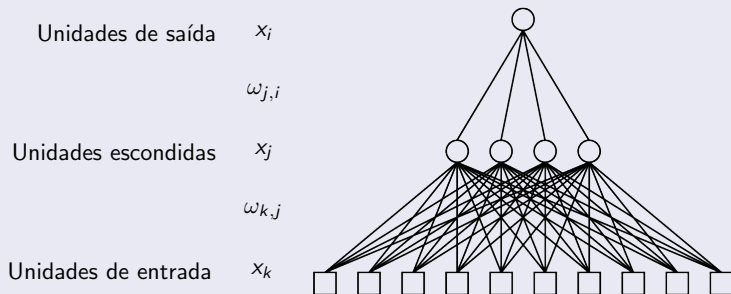
- Possuem camadas em geral totalmente conectadas

Redes Multicamadas

- Possuem camadas em geral totalmente conectadas
 - O número de unidades escondidas é tipicamente escolhido à mão

Redes Multicamadas

- Possuem camadas em geral totalmente conectadas
- O número de unidades escondidas é tipicamente escolhido à mão



Fonte: Slides de AIMA. Russell & Norvig.

Redes Multicamadas – É o viés?

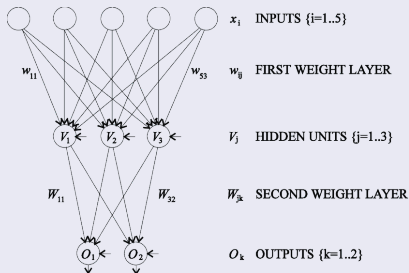
- 2 possibilidades:

Redes Neurais

Redes Multicamadas – É o viés?

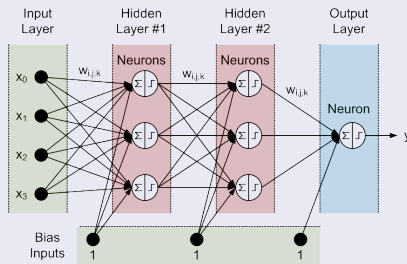
- 2 possibilidades:

Introduzido em cada célula



Fonte: Adaptado de [7]

Compartilhado na camada



Fonte: https://www.forexmt4indicators.com/wp-content/uploads/2014/10/NN1_1.gif

Medindo o erro

- A adição de camadas escondidas traz um problema:

Medindo o erro

- A adição de camadas escondidas traz um problema:
 - Não sabemos qual a saída esperada de cada uma delas, pois os dados de treino não nos dizem isso

Medindo o erro

- A adição de camadas escondidas traz um problema:
 - Não sabemos qual a saída esperada de cada uma delas, pois os dados de treino não nos dizem isso
 - Por conseguinte, não conseguimos medir o erro

Medindo o erro

- A adição de camadas escondidas traz um problema:
 - Não sabemos qual a saída esperada de cada uma delas, pois os dados de treino não nos dizem isso
 - Por conseguinte, não conseguimos medir o erro
- Contudo, podemos retro-propagar o erro da camada de saída para as escondidas

Medindo o erro

- A adição de camadas escondidas traz um problema:
 - Não sabemos qual a saída esperada de cada uma delas, pois os dados de treino não nos dizem isso
 - Por conseguinte, não conseguimos medir o erro
- Contudo, podemos retro-propagar o erro da camada de saída para as escondidas
 - Calculamos a partir do gradiente de erro geral

Redes Multicamadas

Medindo o erro

- A adição de camadas escondidas traz um problema:
 - Não sabemos qual a saída esperada de cada uma delas, pois os dados de treino não nos dizem isso
 - Por conseguinte, não conseguimos medir o erro
- Contudo, podemos retro-propagar o erro da camada de saída para as escondidas
 - Calculamos a partir do gradiente de erro geral
 - Processo de **back-propagation**

Back-Propagation

Back-Propagation

- Calcule os valores de $\delta_i = g'(y_i) \times (t_i - g(y_i))$ para as unidades de saída, usando o erro observado

Back-Propagation

- Calcule os valores de $\delta_i = g'(y_i) \times (t_i - g(y_i))$ para as unidades de saída, usando o erro observado
- Começando da camada de saída, repita o seguinte processo para cada camada na rede, até que a primeira camada escondida seja atingida

Back-Propagation

- Calcule os valores de $\delta_i = g'(y_i) \times (t_i - g(y_i))$ para as unidades de saída, usando o erro observado
- Começando da camada de saída, repita o seguinte processo para cada camada na rede, até que a primeira camada escondida seja atingida
 - Propague os valores de δ para a camada anterior

Back-Propagation

- Calcule os valores de $\delta_i = g'(y_i) \times (t_i - g(y_i))$ para as unidades de saída, usando o erro observado
- Começando da camada de saída, repita o seguinte processo para cada camada na rede, até que a primeira camada escondida seja atingida
 - Propague os valores de δ para a camada anterior
 - Atualize os pesos entre as duas camadas

Back-Propagation

- Calcule os valores de $\delta_i = g'(y_i) \times (t_i - g(y_i))$ para as unidades de saída, usando o erro observado
- Começando da camada de saída, repita o seguinte processo para cada camada na rede, até que a primeira camada escondida seja atingida
 - Propague os valores de δ para a camada anterior
 - Atualize os pesos entre as duas camadas

(A maioria dos neurocientistas nega que essa propagação ocorra no cérebro)

Redes Multicamadas

Back-Propagation: Algoritmo

Função BACKPROP(*Exemplos*, *Rede*): rede neural

repita

para cada peso $\omega_{i,j} \in \text{Rede}$ **faça** $\omega_{i,j} \leftarrow$ pequeno número aleatório

para cada $e = (\vec{x}, \vec{t}) \in \text{Exemplos}$ **faça**

para cada nó i na camada de entrada **faça** $a_i \leftarrow x_i$

para $c = 2$ até C **faça**

para cada nó j na camada c **faça**

$y_j \leftarrow \sum_i \omega_{i,j} a_i$ $a_j \leftarrow g(y_j)$

para cada nó j na camada de saída **faça** $\delta_j \leftarrow g'(y_j) \times (t_j - a_j)$

para $c = C - 1$ até 1 **faça**

para cada nó i na camada c **faça**

$\delta_i \leftarrow g'(y_i) \sum_j \omega_{i,j} \delta_j$

para cada peso $\omega_{i,j} \in \text{Rede}$ **faça** $\omega_{i,j} \leftarrow \omega_{i,j} + \eta \times \delta_j \times a_i$

até Até que algum critério de parada seja satisfeito

retorna a nova rede

Redes Multicamadas

Back-Propagation: Algoritmo

Função BACKPROP(*Exemplos*, *Rede*): rede neural

 repita

 para cada peso $\omega_{i,j} \in \text{Rede}$ faça $\omega_{i,j} \leftarrow$ pequeno número aleatório

 para cada $e = (\vec{x}, \vec{t}) \in \text{Exemplos}$ faça

 para cada nó i na camada de entrada faça $a_i \leftarrow x_i$

 para $c = 2$ até C faça

 para cada nó j na camada c faça

$$y_j \leftarrow \sum_i \omega_{i,j} a_i \quad a_j \leftarrow g(y_j)$$

Conjunto de exemplos,
cada um com entrada
 $\vec{x} = \{x_1, \dots, x_n\}$ e saída \vec{t}

 para cada nó j na camada de saída faça $\delta_j \leftarrow g'(y_j) \times (t_j - a_j)$

 para $c = C - 1$ até 1 faça

 para cada nó i na camada c faça

$$\delta_i \leftarrow g'(y_i) \sum_j \omega_{i,j} \delta_j$$

 para cada peso $\omega_{i,j} \in \text{Rede}$ faça $\omega_{i,j} \leftarrow \omega_{i,j} + \eta \times \delta_j \times a_i$

 até Até que algum critério de parada seja satisfeito

 retorna a nova rede

Redes Multicamadas

Back-Propagation: Algoritmo

Função *BACKPROP*(*Exemplos*, *Rede*): rede neural

 repita

 para cada peso $\omega_{i,j} \in \text{Rede}$ faça $\omega_{i,j} \leftarrow$ pequeno número aleatório

 para cada $e = (\vec{x}, \vec{t}) \in \text{Exemplos}$ faça

 para cada nó i na camada de entrada faça $a_i \leftarrow x_i$

 para $c = 2$ até C faça

 para cada nó j na camada c faça

$$y_j \leftarrow \sum_i \omega_{i,j} a_i \quad a_j \leftarrow g(y_j)$$

rede = $\{\omega_{i,j}, g(y)\}$
com C camadas

 para cada nó j na camada de saída faça $\delta_j \leftarrow g'(y_j) \times (t_j - a_j)$

 para $c = C - 1$ até 1 faça

 para cada nó i na camada c faça

$$\delta_i \leftarrow g'(y_i) \sum_j \omega_{i,j} \delta_j$$

 para cada peso $\omega_{i,j} \in \text{Rede}$ faça $\omega_{i,j} \leftarrow \omega_{i,j} + \eta \times \delta_j \times a_i$

 até Até que algum critério de parada seja satisfeito

 retorna a nova rede

Redes Multicamadas

Back-Propagation: Algoritmo

Função BACKPROP(*Exemplos*, *Rede*): rede neural

repita

para cada peso $\omega_{i,j} \in \text{Rede}$ **faça** $\omega_{i,j} \leftarrow$ pequeno número aleatório

para cada $e = (\vec{x}, \vec{t}) \in \text{Exemplos}$ **faça**

para cada nó i na camada de entrada **faça** $a_i \leftarrow x_i$

para $c = 2$ até C **faça**

para cada nó j na camada c **faça**

$$y_j \leftarrow \sum_i \omega_{i,j} a_i \quad a_j \leftarrow g(y_j)$$

Propaga adiante as entradas
na rede para calcular as saídas

para cada nó j na camada de saída **faça** $\delta_j \leftarrow g'(y_j) \times (t_j - a_j)$

para $c = C - 1$ até 1 **faça**

para cada nó i na camada c **faça**

$$\delta_i \leftarrow g'(y_i) \sum_j \omega_{i,j} \delta_j$$

para cada peso $\omega_{i,j} \in \text{Rede}$ **faça** $\omega_{i,j} \leftarrow \omega_{i,j} + \eta \times \delta_j \times a_i$

até Até que algum critério de parada seja satisfeito

retorna a nova rede

Redes Multicamadas

Back-Propagation: Algoritmo

Função BACKPROP(*Exemplos*, *Rede*): rede neural

repita

para cada peso $\omega_{i,j} \in \text{Rede}$ **faça** $\omega_{i,j} \leftarrow$ pequeno número aleatório

para cada $e = (\vec{x}, \vec{t}) \in \text{Exemplos}$ **faça**

para cada nó i na camada de entrada **faça** $a_i \leftarrow x_i$

para $c = 2$ até C **faça**

para cada nó j na camada c **faça**

$y_j \leftarrow \sum_i \omega_{i,j} a_i$ $a_j \leftarrow g(y_j)$ a_j é a saída do perceptron j

para cada nó j na camada de saída **faça** $\delta_j \leftarrow g'(y_j) \times (t_j - a_j)$

para $c = C - 1$ até 1 **faça**

para cada nó i na camada c **faça**

$\delta_i \leftarrow g'(y_i) \sum_j \omega_{i,j} \delta_j$

para cada peso $\omega_{i,j} \in \text{Rede}$ **faça** $\omega_{i,j} \leftarrow \omega_{i,j} + \eta \times \delta_j \times a_i$

até Até que algum critério de parada seja satisfeito

retorna a nova rede

Redes Multicamadas

Back-Propagation: Algoritmo

Função *BACKPROP*(*Exemplos*, *Rede*): rede neural

 repita

 para cada peso $\omega_{i,j} \in \text{Rede}$ faça $\omega_{i,j} \leftarrow$ pequeno número aleatório

 para cada $e = (\vec{x}, \vec{t}) \in \text{Exemplos}$ faça

 para cada nó i na camada de entrada faça $a_i \leftarrow x_i$

 para $c = 2$ até C faça

 para cada nó j na camada c faça

$y_j \leftarrow \sum_i \omega_{i,j} a_i$ $a_j \leftarrow g(y_j)$

Propaga os deltas da camada
de saída para a de entrada

 para cada nó j na camada de saída faça $\delta_j \leftarrow g'(y_j) \times (t_j - a_j)$

 para $c = C - 1$ até 1 faça

 para cada nó i na camada c faça

$\delta_i \leftarrow g'(y_i) \sum_j \omega_{i,j} \delta_j$

 para cada peso $\omega_{i,j} \in \text{Rede}$ faça $\omega_{i,j} \leftarrow \omega_{i,j} + \eta \times \delta_j \times a_i$

 até Até que algum critério de parada seja satisfeito

 retorna a nova rede

Redes Multicamadas

Back-Propagation: Algoritmo

Função *BACKPROP*(*Exemplos*, *Rede*): rede neural

 repita

 para cada peso $\omega_{i,j} \in \text{Rede}$ faça $\omega_{i,j} \leftarrow$ pequeno número aleatório

 para cada $e = (\vec{x}, \vec{t}) \in \text{Exemplos}$ faça

 para cada nó i na camada de entrada faça $a_i \leftarrow x_i$

 para $c = 2$ até C faça

 para cada nó j na camada c faça

$y_j \leftarrow \sum_i \omega_{i,j} a_i$ $a_j \leftarrow g(y_j)$ Regra de propagação do erro

 para cada nó j na camada de saída faça $\delta_j \leftarrow g'(y_j) \times (t_j - a_j)$

 para $c = C - 1$ até 1 faça

 para cada nó i na camada c faça

$\delta_i \leftarrow g'(y_i) \sum_j \omega_{i,j} \delta_j$

 para cada peso $\omega_{i,j} \in \text{Rede}$ faça $\omega_{i,j} \leftarrow \omega_{i,j} + \eta \times \delta_j \times a_i$

 até Até que algum critério de parada seja satisfeito

 retorna a nova rede

Redes Multicamadas

Back-Propagation: Algoritmo

Função *BACKPROP*(*Exemplos*, *Rede*): rede neural

 repita

 para cada peso $\omega_{i,j} \in \text{Rede}$ faça $\omega_{i,j} \leftarrow$ pequeno número aleatório

 para cada $e = (\vec{x}, \vec{t}) \in \text{Exemplos}$ faça

 para cada nó i na camada de entrada faça $a_i \leftarrow x_i$

 para $c = 2$ até C faça

 para cada nó j na camada c faça

$$y_j \leftarrow \sum_i \omega_{i,j} a_i \quad a_j \leftarrow g(y_j)$$

Atualiza todos os pesos
da rede usando os deltas

 para cada nó j na camada de saída faça $\delta_j \leftarrow g'(y_j) \times (t_j - a_j)$

 para $c = C - 1$ até 1 faça

 para cada nó i na camada c faça

$$\delta_i \leftarrow g'(y_i) \sum_j \omega_{i,j} \delta_j$$

 para cada peso $\omega_{i,j} \in \text{Rede}$ faça $\omega_{i,j} \leftarrow \omega_{i,j} + \eta \times \delta_j \times a_i$

 até Até que algum critério de parada seja satisfeito

 retorna a nova rede

Redes Multicamadas

Back-Propagation: Algoritmo

Função *BACKPROP*(*Exemplos*, *Rede*): rede neural

 repita

 para cada peso $\omega_{i,j} \in \text{Rede}$ faça $\omega_{i,j} \leftarrow$ pequeno número aleatório

 para cada $e = (\vec{x}, \vec{t}) \in \text{Exemplos}$ faça

 para cada nó i na camada de entrada faça $a_i \leftarrow x_i$

 para $c = 2$ até C faça

 para cada nó j na camada c faça

$$y_j \leftarrow \sum_i \omega_{i,j} a_i \quad a_j \leftarrow g(y_j)$$

Note que $\Delta\omega_{i,j} = \eta \delta_j a_i$

 para cada nó j na camada de saída faça $\delta_j \leftarrow g'(y_j) \times (t_j - a_j)$

 para $c = C - 1$ até 1 faça

 para cada nó i na camada c faça

$$\delta_i \leftarrow g'(y_i) \sum_j \omega_{i,j} \delta_j$$

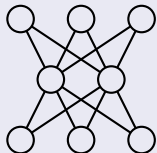
 para cada peso $\omega_{i,j} \in \text{Rede}$ faça $\omega_{i,j} \leftarrow \omega_{i,j} + \eta \times \delta_j \times a_i$

 até Até que algum critério de parada seja satisfeito

 retorna a nova rede

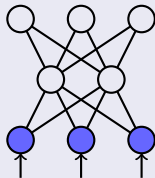
Redes Multicamadas

Back-Propagation: Visualização



Iniciamos os pesos na rede

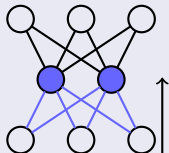
Back-Propagation: Visualização



Para cada exemplo de treino, entramos seus atributos nos nós de entrada, calculando a saída de cada unidade nas camadas escondida e de saída

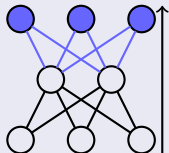
Redes Multicamadas

Back-Propagation: Visualização



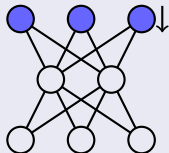
Para cada exemplo de treino, entramos seus atributos nos nós de entrada, calculando a saída de cada unidade nas camadas escondida e de saída

Back-Propagation: Visualização



Para cada exemplo de treino, entramos seus atributos nos nós de entrada, calculando a saída de cada unidade nas camadas escondida e de saída

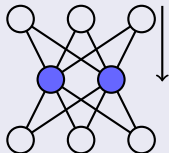
Back-Propagation: Visualização



Calculamos $\delta_k \leftarrow s_k(1 - s_k)(t_k - s_k)$
para cada unidade de saída, onde t_k é o
valor-alvo da unidade e $s_k = g(y)$ sua saída

Redes Multicamadas

Back-Propagation: Visualização

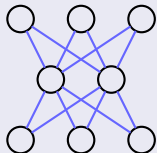


Então calculamos

$$\delta_h \leftarrow s_h(1 - s_h) \sum \omega_{h,k} \delta_k$$

para cada nó escondido

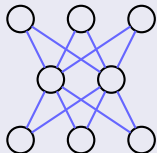
Back-Propagation: Visualização



Ajustamos os pesos de todas as ligações
 $\omega_{j,i} \leftarrow \omega_{j,i} + \eta \delta_i x_j$, onde x_j é a
ativação e η a taxa de aprendizagem

Redes Multicamadas

Back-Propagation: Visualização

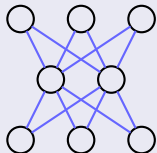


Ajustamos os pesos de todas as ligações
 $\omega_{j,i} \leftarrow \omega_{j,i} + \eta \delta_i x_j$, onde x_j é a
ativação e η a taxa de aprendizagem

Quando parar?

Redes Multicamadas

Back-Propagation: Visualização



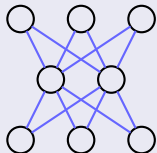
Ajustamos os pesos de todas as ligações
 $\omega_{j,i} \leftarrow \omega_{j,i} + \eta \delta_i x_j$, onde x_j é a
ativação e η a taxa de aprendizagem

Quando parar?

- Parar em um certo número de iterações?

Redes Multicamadas

Back-Propagation: Visualização



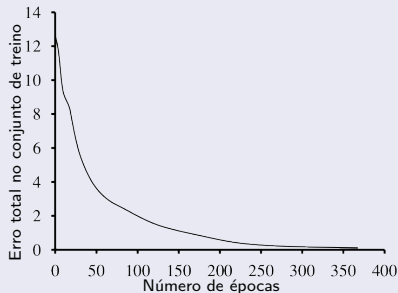
Ajustamos os pesos de todas as ligações
 $\omega_{j,i} \leftarrow \omega_{j,i} + \eta \delta_i x_j$, onde x_j é a
ativação e η a taxa de aprendizagem

Quando parar?

- Parar em um certo número de iterações?
- Parar quando o erro for menor que um determinado valor? (Curva de treinamento)

Curva de Treinamento

- Mede o desempenho de um classificador em um conjunto de treinamento fixo enquanto o processo de aprendizado continua nesse mesmo conjunto

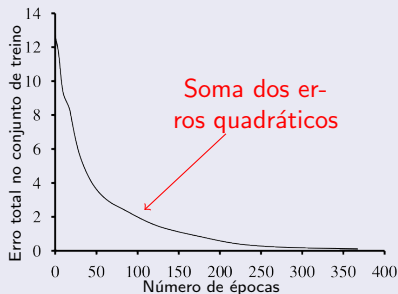


Fonte: AIMA. Russell & Norvig.

Redes Multicamadas

Curva de Treinamento

- Mede o desempenho de um classificador em um conjunto de treinamento fixo enquanto o processo de aprendizado continua nesse mesmo conjunto

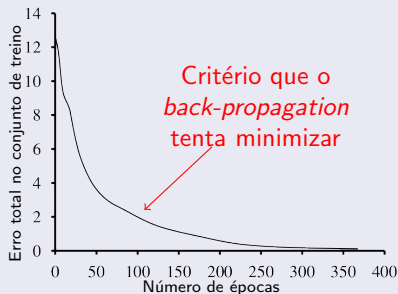


Fonte: AIMA. Russell & Norvig.

Redes Multicamadas

Curva de Treinamento

- Mede o desempenho de um classificador em um conjunto de treinamento fixo enquanto o processo de aprendizado continua nesse mesmo conjunto

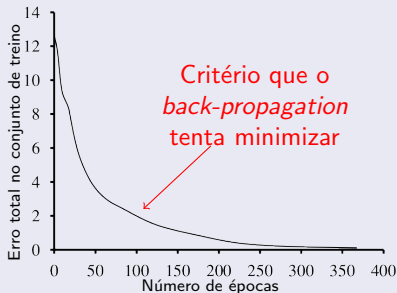


Fonte: AIMA. Russell & Norvig.

Redes Multicamadas

Curva de Treinamento

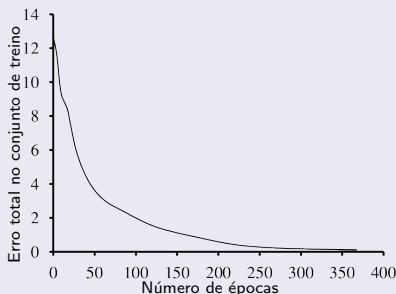
- Mede o desempenho de um classificador em um conjunto de treinamento fixo enquanto o processo de aprendizado continua nesse mesmo conjunto
- Quando a soma dos erros quadráticos em uma passada inteira no conjunto de treino (uma época) é suficientemente pequeno, podemos dizer que a rede convergiu (para esses exemplos)



Fonte: AIMA. Russell & Norvig.

Curva de Treinamento: *Overfitting*

- O *back-propagation* pode, se o treinamento for longo, se aproximar não da função desejada, mas sim dos exemplos

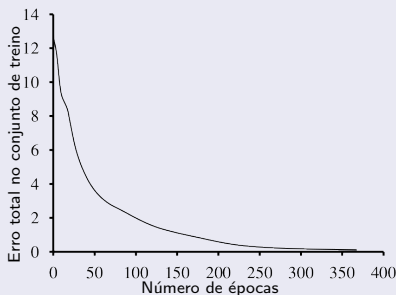


Fonte: AIMA. Russell & Norvig.

Redes Multicamadas

Curva de Treinamento: *Overfitting*

- O *back-propagation* pode, se o treinamento for longo, se aproximar não da função desejada, mas sim dos exemplos
- Os pesos acabam sendo ajustados para representar idiossincrasias dos exemplos de treino que não são representativos da distribuição geral



Fonte: AIMA. Russell & Norvig.

Overfitting: Como contornar

- Decrescer cada peso por um fator pequeno em cada iteração, mantendo o valor dos pesos pequeno

Overfitting: Como contornar

- Decrescer cada peso por um fator pequeno em cada iteração, mantendo o valor dos pesos pequeno
- Criar um conjunto de validação (melhor alternativa)

Overfitting: Como contornar

- Decrescer cada peso por um fator pequeno em cada iteração, mantendo o valor dos pesos pequeno
- Criar um conjunto de validação (melhor alternativa)
 - O algoritmo retorna a rede resultante do número de épocas que produziu o menor erro no conjunto de validação

Overfitting: Como contornar

- Decrescer cada peso por um fator pequeno em cada iteração, mantendo o valor dos pesos pequeno
- Criar um conjunto de validação (melhor alternativa)
 - O algoritmo retorna a rede resultante do número de épocas que produziu o menor erro no conjunto de validação
- Resolve?

Overfitting: Como contornar

- Decrescer cada peso por um fator pequeno em cada iteração, mantendo o valor dos pesos pequeno
- Criar um conjunto de validação (melhor alternativa)
 - O algoritmo retorna a rede resultante do número de épocas que produziu o menor erro no conjunto de validação
- Resolve? Às vezes... nem sempre

Overfitting: Como contornar

- Decrescer cada peso por um fator pequeno em cada iteração, mantendo o valor dos pesos pequeno
- Criar um conjunto de validação (melhor alternativa)
 - O algoritmo retorna a rede resultante do número de épocas que produziu o menor erro no conjunto de validação
- Resolve? Às vezes... nem sempre
 - Há o perigo dos mínimos locais...

Variação

- Podemos incluir momento:

Variação

- Podemos incluir momento:
 - A atualização dos pesos na n -ésima iteração depende parcialmente da atualização que ocorreu na $(n-1)$ -ésima iteração

Variação

- Podemos incluir momento:
 - A atualização dos pesos na n -ésima iteração depende parcialmente da atualização que ocorreu na $(n-1)$ -ésima iteração
 - Uma iteração corresponde a uma época \rightarrow uma passada completa pelo conjunto de treino

Variação

- Podemos incluir momento:
 - A atualização dos pesos na n -ésima iteração depende parcialmente da atualização que ocorreu na $(n-1)$ -ésima iteração
 - Uma iteração corresponde a uma época \rightarrow uma passada completa pelo conjunto de treino
 - Usamos então uma constante $0 \leq \alpha \leq 1$, chamada **momento**, e

Variação

- Podemos incluir momento:
 - A atualização dos pesos na n -ésima iteração depende parcialmente da atualização que ocorreu na $(n-1)$ -ésima iteração
 - Uma iteração corresponde a uma época \rightarrow uma passada completa pelo conjunto de treino
 - Usamos então uma constante $0 \leq \alpha \leq 1$, chamada **momento**, e

$$\Delta\omega_{i,j}(n) = \eta\delta_jx_i + \alpha\Delta\omega_{i,j}(n-1)$$

$$\delta_j = g'(y_j)(t_j - g(y_j))$$

Redes Multicamadas

Exemplo (Mitchell)

- Tarefa: dado o rosto de uma pessoa, dizer para que direção olha

Redes Multicamadas

Exemplo (Mitchell)

- Tarefa: dado o rosto de uma pessoa, dizer para que direção olha
 - Usou a variação com momento

Exemplo (Mitchell)

- Tarefa: dado o rosto de uma pessoa, dizer para que direção olha
 - Usou a variação com momento
 - Treinou em 260 imagens com 90% de precisão

Exemplo (Mitchell)

- Tarefa: dado o rosto de uma pessoa, dizer para que direção olha
 - Usou a variação com momento
 - Treinou em 260 imagens com 90% de precisão
 - Código e detalhes em www.cs.cmu.edu/~tom/mlbook.html

Redes Multicamadas

Exemplo (Mitchell)

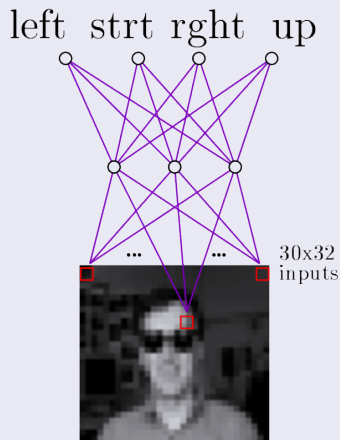
- Tarefa: dado o rosto de uma pessoa, dizer para que direção olha
 - Usou a variação com momento
 - Treinou em 260 imagens com 90% de precisão
 - Código e detalhes em www.cs.cmu.edu/~tom/mlbook.html
- Entrada:



Redes Multicamadas

Exemplo (Mitchell)

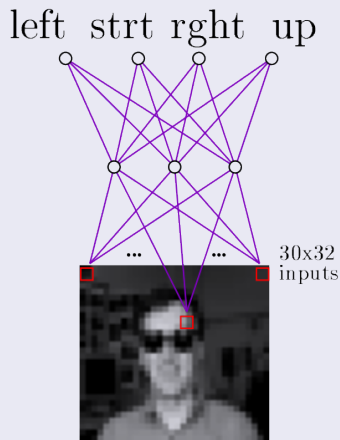
- Codificação da entrada:



Redes Multicamadas

Exemplo (Mitchell)

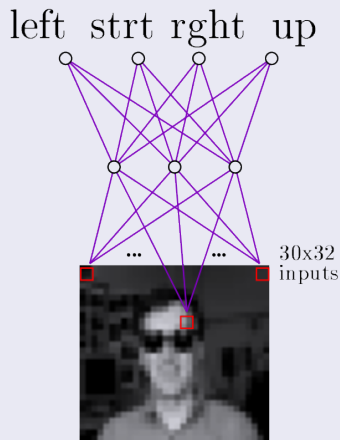
- Codificação da entrada:
 - 30×32 entradas (pixels)



Redes Multicamadas

Exemplo (Mitchell)

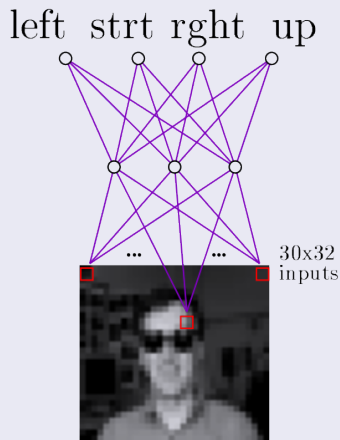
- Codificação da entrada:
 - 30×32 entradas (pixels)
 - Cada pixel representado pela sua intensidade, entre $[0,1]$



Redes Multicamadas

Exemplo (Mitchell)

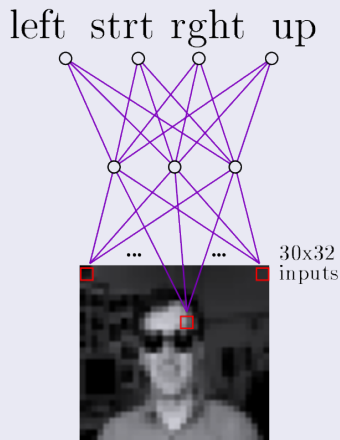
- Codificação da entrada:
 - 30×32 entradas (pixels)
 - Cada pixel representado pela sua intensidade, entre $[0,1]$
- Rede:



Redes Multicamadas

Exemplo (Mitchell)

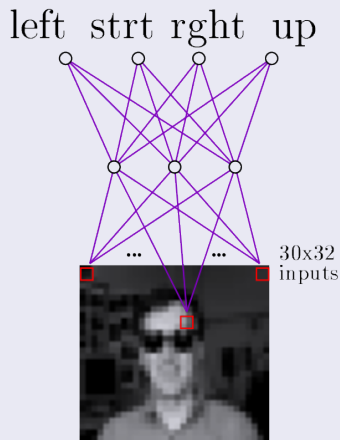
- Codificação da entrada:
 - 30×32 entradas (pixels)
 - Cada pixel representado pela sua intensidade, entre $[0,1]$
- Rede:
 - 2 camadas: 3 unidades escondidas e 4 de saída



Redes Multicamadas

Exemplo (Mitchell)

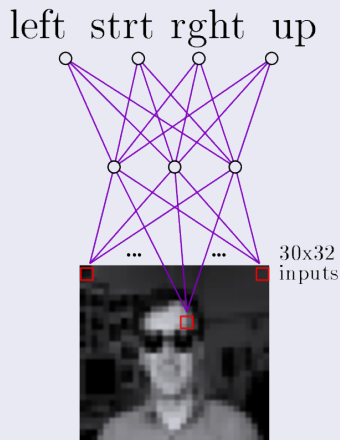
- Codificação da entrada:
 - 30×32 entradas (pixels)
 - Cada pixel representado pela sua intensidade, entre $[0,1]$
- Rede:
 - 2 camadas: 3 unidades escondidas e 4 de saída
 - Taxa de aprendizagem $\eta = 0,3$



Redes Multicamadas

Exemplo (Mitchell)

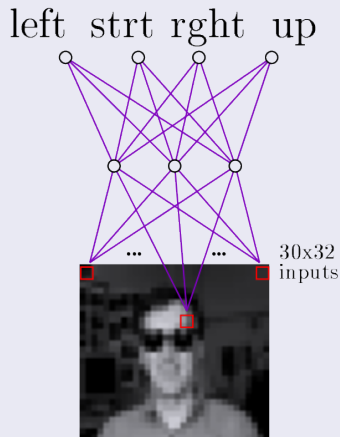
- Codificação da entrada:
 - 30×32 entradas (pixels)
 - Cada pixel representado pela sua intensidade, entre $[0,1]$
- Rede:
 - 2 camadas: 3 unidades escondidas e 4 de saída
 - Taxa de aprendizagem $\eta = 0,3$
 - Momento $\alpha = 0,3$



Redes Multicamadas

Exemplo (Mitchell)

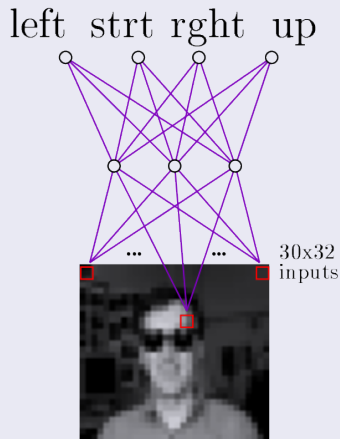
- Pesos iniciais (inclusive o viés):



Redes Multicamadas

Exemplo (Mitchell)

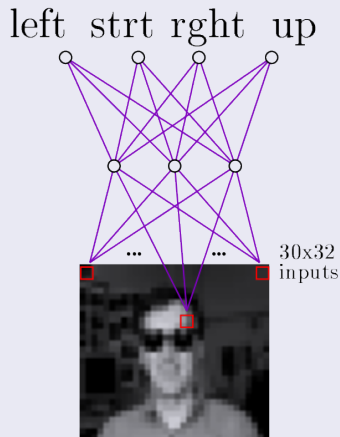
- Pesos iniciais (inclusive o viés):
- Pequenos e aleatórios nas unidades de saída



Redes Multicamadas

Exemplo (Mitchell)

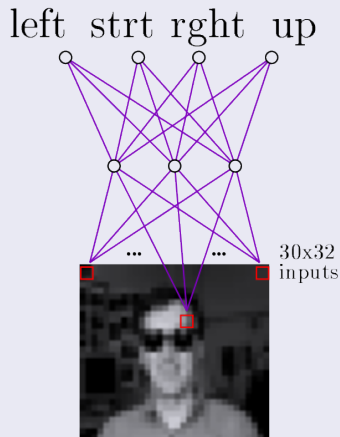
- Pesos iniciais (inclusive o viés):
- Pequenos e aleatórios nas unidades de saída
- Zero nas unidades de entrada



Redes Multicamadas

Exemplo (Mitchell)

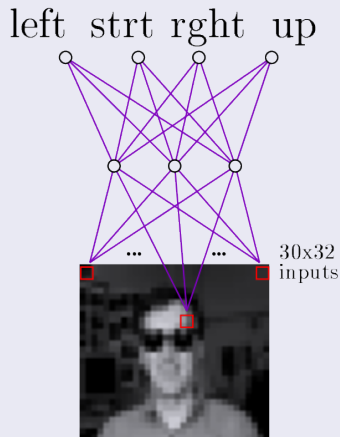
- Pesos iniciais (inclusive o viés):
- Pequenos e aleatórios nas unidades de saída
- Zero nas unidades de entrada
- Parada:



Redes Multicamadas

Exemplo (Mitchell)

- Pesos iniciais (inclusive o viés):
 - Pequenos e aleatórios nas unidades de saída
 - Zero nas unidades de entrada
- Parada:
 - A cada 50 passos no *gradient descent* compara com desempenho no conjunto de validação. Guarda os pesos do menor erro



Representatividade

- Que funções podemos representar com essas redes?

Representatividade

- Que funções podemos representar com essas redes?
 - Booleanas: representadas exatamente com redes de 2 camadas

Representatividade

- Que funções podemos representar com essas redes?
 - Booleanas: representadas exatamente com redes de 2 camadas
 - Contínuas: aproximadas com pequeno erro por rede com 2 camadas

Representatividade

- Que funções podemos representar com essas redes?
 - Booleanas: representadas exatamente com redes de 2 camadas
 - Contínuas: aproximadas com pequeno erro por rede com 2 camadas
 - Arbitrárias: aproximadas com precisão arbitrária por rede com 3 camadas

Referências

- ① Russell, S.; Norvig P. (2010): Artificial Intelligence: A Modern Approach. Prentice Hall. 3a ed.
 - ① Slides do livro: aima.eecs.berkeley.edu/slides-pdf/
- ② <http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-034Spring-2005/LectureNotes/index.htm>
- ③ <http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/>
- ④ <https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5>
- ⑤ Mitchell, T.M.: Machine Learning. McGraw-Hill. 1997.
- ⑥ https://www.researchgate.net/publication/273450589_COMBINACAO_LINEAR_DE_REDES_NEURAIIS_ARTIFICIAIS_E_MQUINAS_DE_VETORES_DE_SUPORTE_PARA_REGRESSAO_NAS_PREVISIOES_DE_VAZOES_MENSAIS_NO_POSTO_266-ITAIPU
- ⑦ Kemp, R.; Macaulay, C.; Palcic, B.(1997): Opening the Black Box: the Relationship between Neural Networks and Linear Discriminant Functions. Analytical Cellular Pathology, 14(1):19-30.