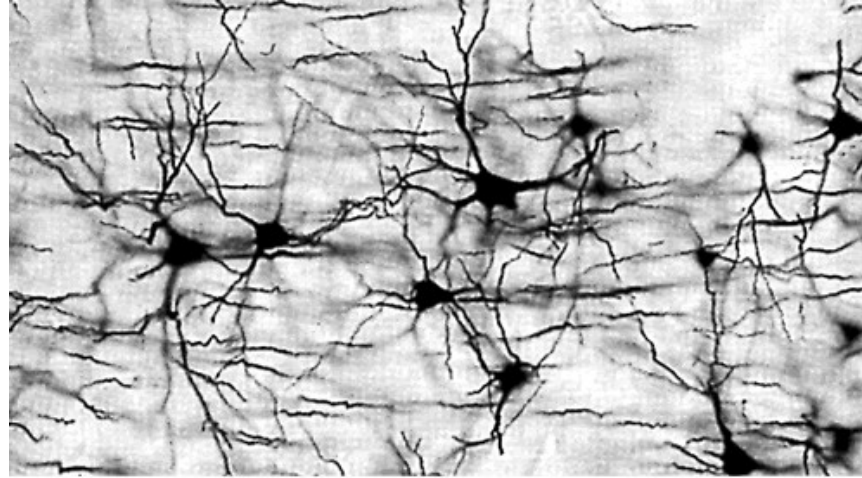


Aprendizado de Máquinas

Multi-Layer Perceptron (MLP)

David Menotti, Ph.D.
web.inf.ufpr.br/menotti


Redes Neurais



- Cérebro humano.
 - Mais fascinante processador existente.
 - Aprox. 10 bilhões de neurônios conectados através de sinapses.
 - Sinapses transmitem estímulos e o resultado pode ser estendido por todo o corpo humano.

Redes Neuroniais Artificiais:

Um breve histórico

- 1943 – McCulloch e Pitts.
 - Sugeriram a construção de uma máquina baseada ou inspirada no cérebro humano.
- 1949 – Donald Hebb.
 - Propõe uma lei de aprendizagem específica para as sinápses dos neurônios.
- 1957/1958 - Frank Rosenblatt.
 - Estudos aprofundados  pai da neuro-computação.
 - Perceptron.
 - Criador do primeiro neuro-computador a obter sucesso (Mark I).

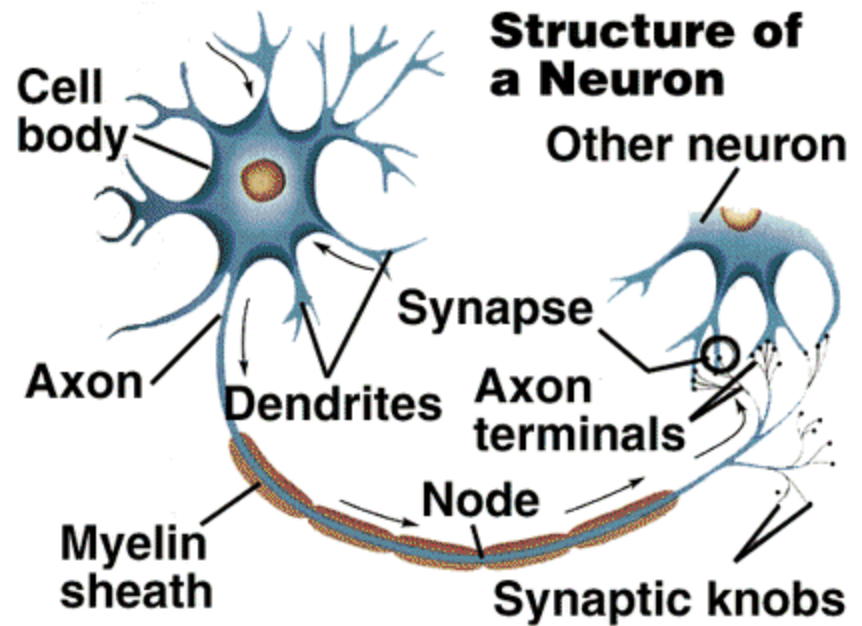
Mark I - Perceptron



Um breve histórico (cont)

- 1958-1967.
 - Várias pesquisas mal sucedidas.
- 1967-1982.
 - Pesquisas silenciosas.
- 1986 – David Rumelhart.
 - Livro “*parallel distributed processing*”.
 - Algoritmo eficaz de aprendizagem.
- 1987.
 - Primeira conferência IEEE Int. Conf. On Neural Nets.

N



- Dendritos:
 - Receber os estímulos transmitidos por outros neurônios.
- Corpo (somma).
 - Coletar e combinar informações vindas de outros neurônios.
- Axônio.
 - Transmite estímulos para outras células.

Redes Neurais Artificiais

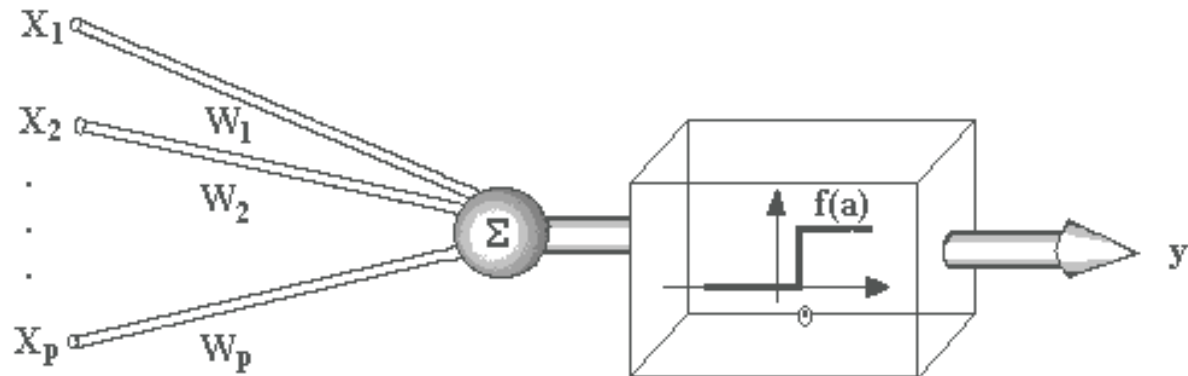
- Técnica computacional que apresenta um modelo inspirado na estrutura do neurônio.
- Simula o cérebro humano:
 - Aprendendo, errando e fazendo descobertas.
- Estrutura de processamento de informação distribuída paralelamente na forma de um grafo direcionado.

Cérebro X Computador

Parâmetro	Cérebro	Computador
Material	Orgânico	Metal e Plástico
Velocidade	Milisegundos	Nanosegundos
Tipo de Processamento	Paralelo	Sequencial
Armazenamento	Adaptativo	Estático
Controle de Processos	Distribuído	Centralizado
Número de Elementos Processados	10^{11} a 10^{14}	10^5 a 10^6
Ligações entre Elementos Processados	10.000	$< 10^8$

Neurônio artificial

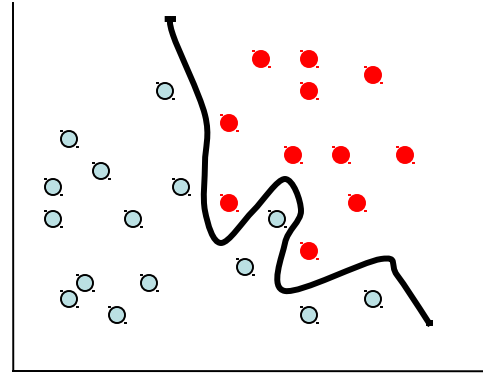
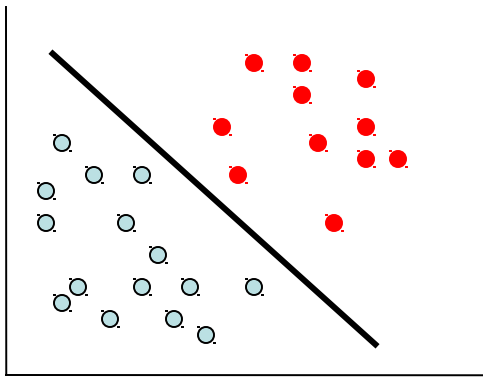
- Sinais são apresentados à entrada.
- Cada sinal é multiplicado por um peso.
- Soma ponderada produz um nível de ativação.
- Se esse nível excede um limite (*threshold*) a unidade produz uma saída.



[PERCEPTRON]

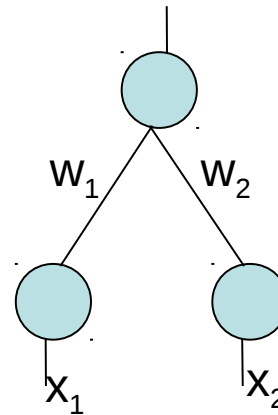
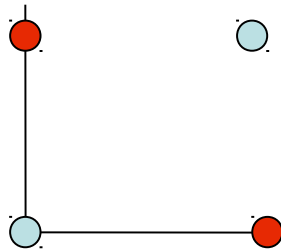
Perceptron

- Resolve problemas linearmente separáveis somente.
- Problemas reais, entretanto, na maioria das vezes são mais complexos.



Exemplo

- Considere por exemplo, o problema XOR



Como visto anteriormente com SVMs, podemos afirmar que em altas dimensões os problemas são linearmente separáveis.

Sendo assim, vamos mudar o problema de R^2 para R^3 adicionando uma terceira característica.

Exemplo

X1	X2	X3	Output
1	1	1	0
1	0	0	1
0	1	0	1
0	0	0	0

- Nesse caso, a característica adicionada (X3) é a operação AND entre X1 e X2
- O fato de adicionarmos essa característica faz com que o problema torne-se linearmente separável.

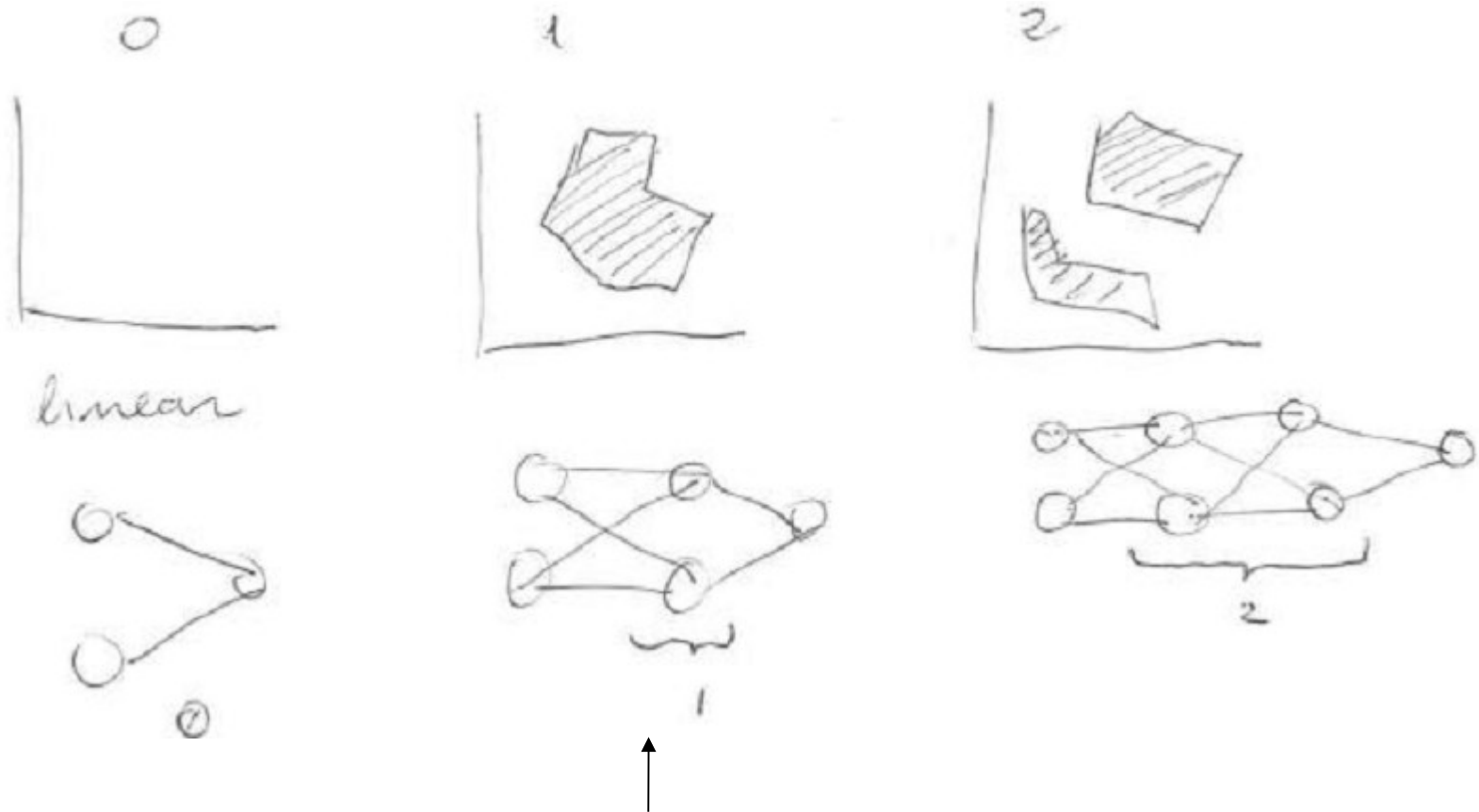
Adicionando uma camada

- Outra maneira de resolver esse problema consiste em adicionar uma camada extra (escondida) entre as camadas de entrada e saída.
- Dado uma quantidade suficiente de neurônios na camada escondida, é possível resolver qualquer tipo de problemas
 - Claro que isso depende das características de entrada.

Camada Escondida

- Essa camada pode ser vista como um extrator de características, ou seja, a grosso modo, o neurônio escondido seria uma característica a mais
 - O que torna o problema do XOR linearmente separável.

Camadas X Fronteiras



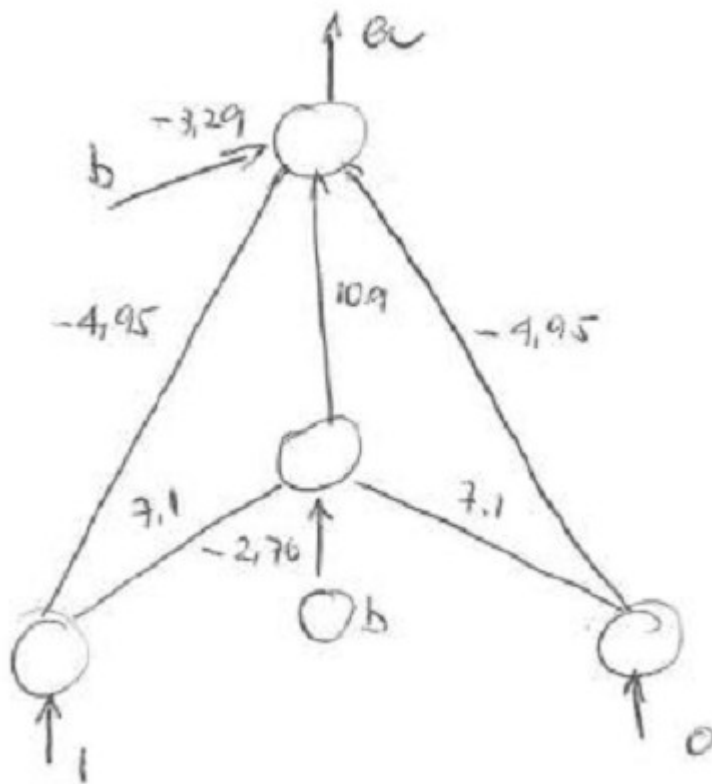
Geralmente uma camada escondida resolve a grande maioria dos problemas.

O problema de atribuição de créditos

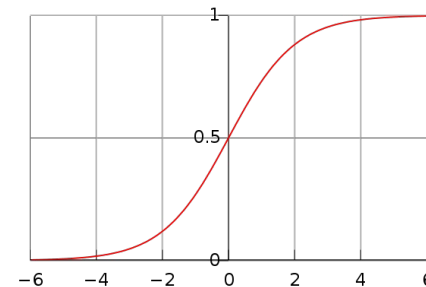
- Quando temos uma camada escondida, surge o problema de atribuição de créditos aos neurônios desta camada
 - Não existem “*targets*” como na camada de saída.
 - Período entre 58 e 82 foi dedicado a resolver esse problema
 - Solução foi o algoritmo conhecido como Backpropagation.
 - David E. Rumelhart et al (1986)

MLP para o problema XOR

- Considere o problema XOR e a seguinte rede já treinada.



A saída é calculada de maneira similar ao perceptron, mas a MLP geralmente usa sigmoid como função de ativação.



$$v = \frac{1}{1 + e^{-s}}$$

Para grandes quantidades de dados o resultado da sigmoid pode ser interpretado como estimativa de probabilidade.

MLP para o problema XOR

- Para calcular a saída da rede, primeiramente devemos encontrar o valor do neurônio escondido.

$$1 * 7.1 + 1 * -2.76 + 0 * 7.1 = 4.34$$

- Passando o valor 4.34 na função de ativação, temos 0.987.

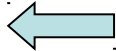

- O valor de saída é então calculado

$$1 * -4.95 + 0 * -4.95 + 0.987 * 10.9 + 1 * -3.29 = 2.52$$

- Passando 2.52 na função de ativação temos a saída igual a 0.91

MLP para o problema XOR

- Após classificarmos todos os padrões de entrada teríamos

1	0	0.91	
0	0	0.08	
0	1	1.00	
1	1	0.10	

Podemos usar como regra de decisão um limiar igual a 0.9, por exemplo.

BackProp

- Seja o_j o valor de ativação para o neurônio j .
- Seja f uma função de ativação.
- Seja w_{ij} o peso entre os neurônios i e j .
- Seja net_j a entrada para o neurônio j , a qual é calculada por $net_j = \sum_{i=1}^n w_{ij} o_i$
- onde n é o número de unidades ligadas ao neurônio j e $o_j = f(net_j)$

BackProp

- O treinamento acontece da seguinte maneira:
 1. Inicializar os valores dos pesos e neurônios aleatoriamente.
 2. Apresentar um padrão a camada de entrada da rede
 3. Encontrar os valores para as camadas escondidas e a camada de saída.
 4. Encontrar o erro da camada de saída.
 5. Ajustar os pesos através da retropropagação dos erros (Backpropagation)
 1. Diminuir o erro a cada iteração
 6. Encontrar o erro na camada escondida
 7. Ajustar os pesos.

Critério de parada é o erro desejado

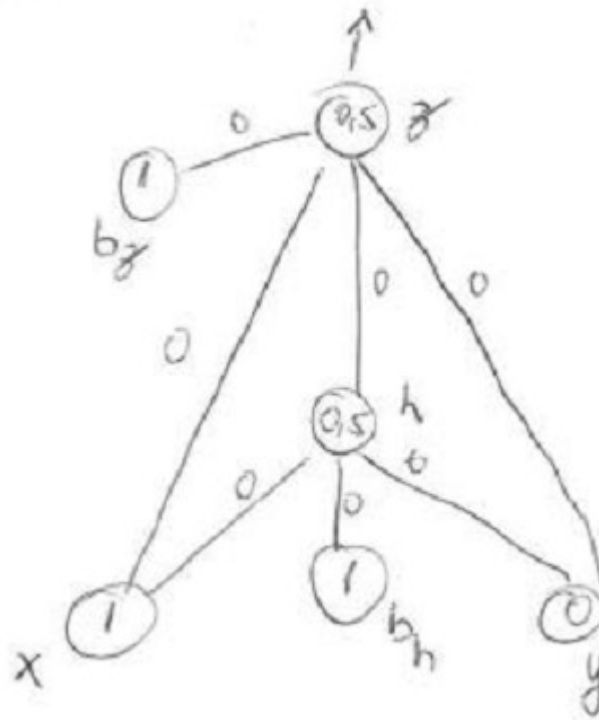
BackProp

- O valor corrente de ativação de um neurônio k é o_k e o target é t_k
- Após realizar os passos 1, 2, e 3, o próximo passo consiste em calcular o erro, o qual pode ser realizado através da seguinte equação

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

BackProp: Exemplo

- Considere uma rede inicializada da seguinte forma



Nesse caso $\delta_z = (1 - 0.5) \times 0.5 \times (1 - 0.5) = 0.125$

BackProp: Exemplo

- A fórmula para atualizar os pesos W entre o neurônio i e j é

$$w_{ij} = w_{ij} + \eta \delta_j o_i$$

- onde η é uma constante pequena e positiva chamada de “taxa de aprendizagem” (*learning rate*)
- Usando uma taxa de 0.1, temos

BackProp: Exemplo

$$w_{xz} = 0 + 0.1 \times 0.125 \times 1 = 0.0125$$

$$w_{yz} = 0 + 0.1 \times 0.125 \times 0 = 0$$

$$w_{hz} = 0 + 0.1 \times 0.125 \times 0.5 = 0.00625$$

$$w_{bz} = 0 + 0.1 \times 0.125 \times 1 = 0.0125$$

A fórmula para calcular o erro dos neurônios da camada escondida é

$$\delta_i = o_i(1 - o_i) \sum_k \delta_k w_{ik}$$

Como temos somente um neurônio no nosso exemplo

$$\delta_h = o_h(1 - o_h) \delta_z w_{hz}$$

BackProp: Exemplo

Nesse caso teríamos

$$\delta_h = 0.5(1 - 0.5)0.125 \times 0.00625 = 0.000195313$$

Para atualizar os pesos, usamos a mesma fórmula

$$w_{hx} = 0 + 0.1 \times 0.0001953 \times 1 = 0.00001953$$

$$w_{hy} = 0 + 0.1 \times 0.0001953 \times 0 = 0$$

$$w_{hbh} = 0 + 0.1 \times 0.0001953 \times 1 = 0.00001953$$

Com os novos pesos calculados, a saída da rede seria 0.507031

BackProp: Exemplo

- Após aplicarmos todos os padrões, teríamos o seguinte

1	0	0.4998
0	0	0.4998
0	1	0.4998
1	1	0.4997

Usando uma taxa de aprendizagem = 0.1, o algoritmo levará 20.000 iterações para convergir. Uma solução para melhorar isso seria aumentar a taxa de aprendizagem. Se usarmos 2, o algoritmo converge em 480 iterações.

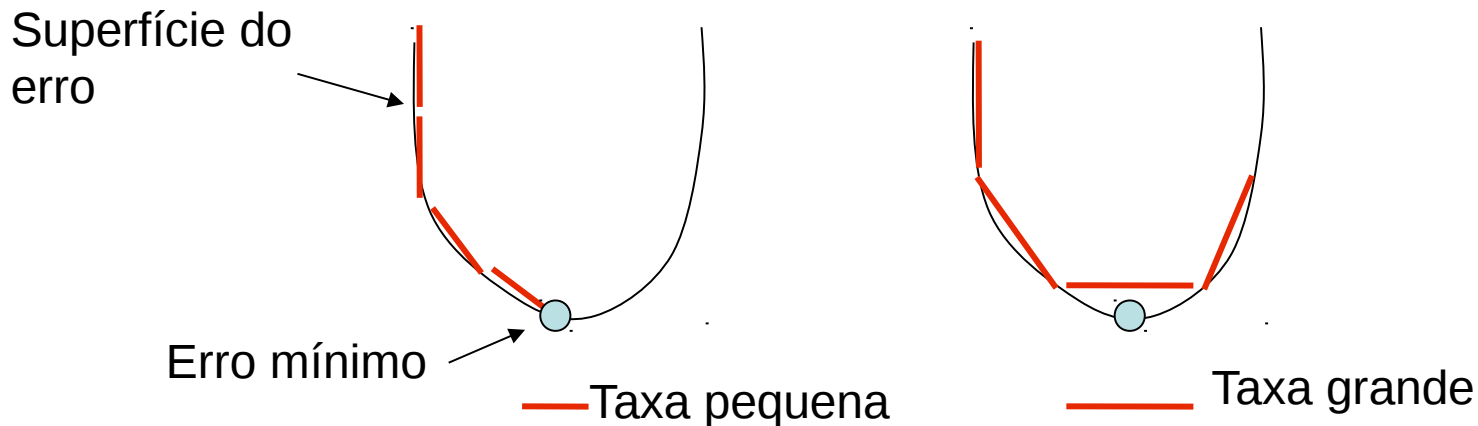
Aspectos Práticos

- Alguns aspectos práticos devem ser considerados na utilização de redes neurais MLP.
 - Taxa de aprendizagem
 - Momentum
 - Online vs batch
 - Shuffle
 - Normalização
 - Inicialização dos pesos
 - Generalização

Referência Interessante:
Efficient Backprop, Y. LeCun et al,
1998

Taxa de Aprendizagem

- Taxas muito pequenas tornam o processo bastante lento.
- Taxas muito grandes tornam o processo rápido.
 - Podem não trazer os resultados ideais.

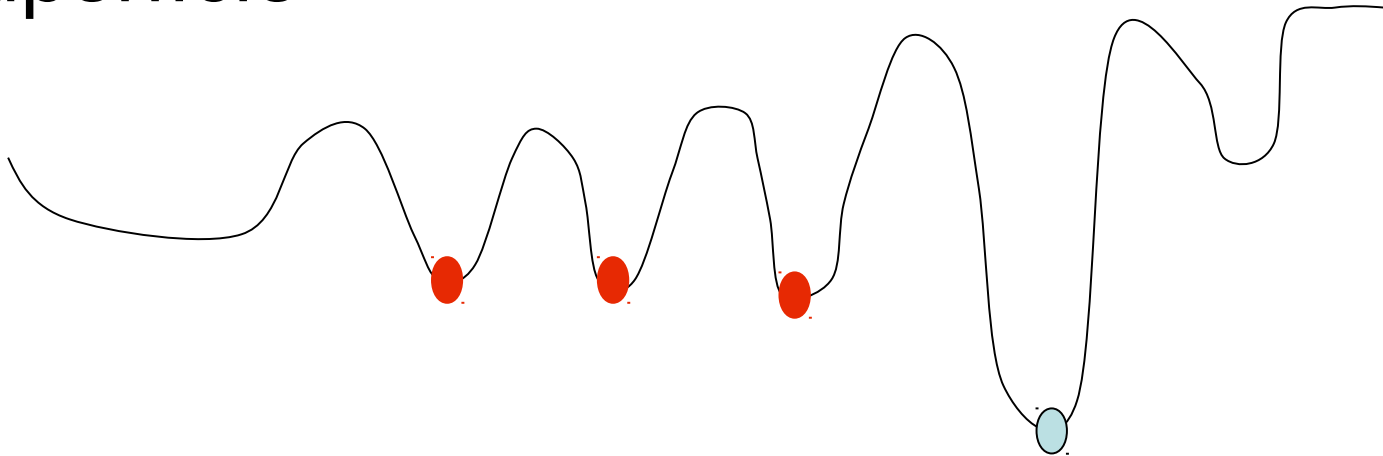


Taxa de Aprendizagem

- O ideal é começar com uma taxa grande e reduzir durante as iterações.
- Permite a exploração global no início (*exploration*) a local (*exploitation*) quando o algoritmo estiver próximo do ótimo global.
- Geralmente valores entre 0.05 e 0.75 fornecem bons resultados.

Momentum

- É uma estratégia usada para evitar mínimos locais. Considere a seguinte superfície



- Existem três mínimos locais antes do mínimo global.

Momentum

- Considere, por exemplo, que uma bola seja solta no início da superfície.
- Se ela tiver “*momentum*” suficiente, ela vai conseguir passar os três mínimos locais e alcançar o mínimo global.

$$w_{ij} = \alpha w_{ij} + \eta \delta_j o_k$$

- Normalmente $0 \leq \alpha \leq 0.9$

On-line vs Batch

- A diferença está no momento em que os pesos são atualizados.
- Na versão *on-line*, os pesos são atualizados a cada padrão apresentado a rede.
- Na versão *batch*, todos os pesos são somados durante uma iteração (todos os padrões) e só então os pesos são atualizados.
- Versão *batch*
 - Interessante para aplicações que possam ser paralelizadas.
- Versão *on-line*
 - Geralmente converge mais rapidamente.

Misturando Exemplos (*Shuffle*)

- Redes neurais aprendem melhor quando diferentes exemplos de diferentes classes são apresentados a rede.
- Uma prática muito comum consiste em apresentar um exemplo de cada classe a rede
 - Isso garante que os pesos serão atualizados levando-se em consideração todas as classes.

Misturando Exemplos (*Shuffle*)

- Se apresentarmos à rede todos os exemplos de uma classe, e assim por diante, os pesos finais tenderão para a última classe
 - A rede vai “esquecer” o que ela aprendeu antes.

Normalização

- A normalização é interessante quando existem características em diversas unidades dentro do vetor de características.
- Nesses casos, valores muito altos podem saturar a função de ativação.
- Uma maneira bastante simples de normalizar os dados consiste em somar todas as características e dividir pela soma
- Outra normalização bastante usada é a normalização Z-score.

Normalização

- Para redes neurais MLP, geralmente é interessante ter as características com média próxima de zero

$$Z = \frac{X - \mu}{\sigma}$$

- Melhora o tempo de convergência durante a aprendizagem.

Normalização

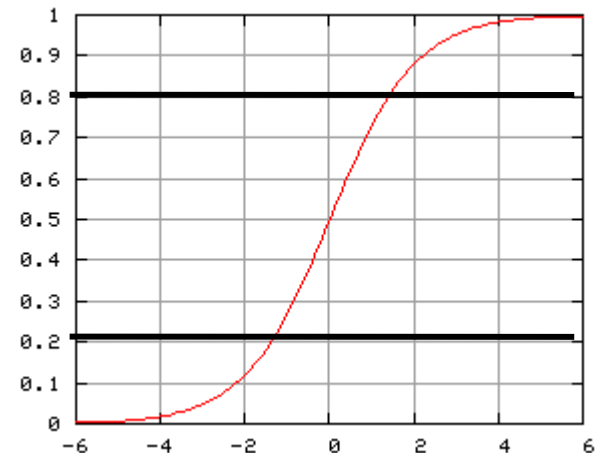
- As características devem ser não correlacionadas se possível
 - Quando temos poucas características podemos verificar isso facilmente.
 - Com várias características, o problema se torna muito mais complexo.
 - Métodos de seleção de características.

Inicialização dos Pesos

- Assumindo que os dados foram normalizados e uma *sigmoid* está sendo usada como função de ativação.
 - Os pesos devem ser inicializados aleatoriamente mas de modo que fiquem na região linear da sigmoid

Com pesos muito altos ou muito baixo a sigmoid deve saturar

- Gradientes pequenos
- Aprendizagem muito lenta.

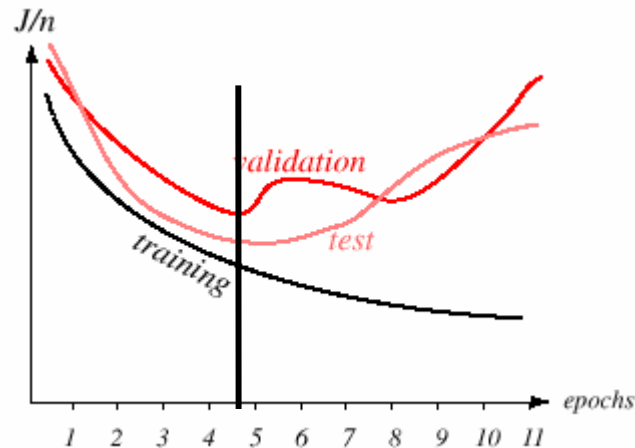


Generalização

- Um aspecto bastante importante quando treinamos um classificador é garantir que o mesmo tenha um bom poder de generalização.
 - Evitar *overfitting*.
- A maneira clássica de se garantir uma boa generalização consiste em reservar uma parte da base para **validar** a generalização.

Generalização

- A cada iteração, devemos monitorar o desempenho na base de validação.
- Não é raro observar o seguinte desempenho

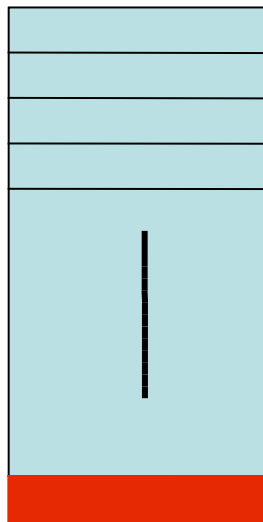


Generalização

- Uma outra estratégia é a validação cruzada.
 - Interessante quando a base não é muito grande
 - Separar alguns exemplos para validação pode prejudicar o treinamento.
- Consiste em dividir a base de aprendizagem em n partições iguais e usar $n-1$ partições para aprendizagem e uma partição para validação.

Generalização

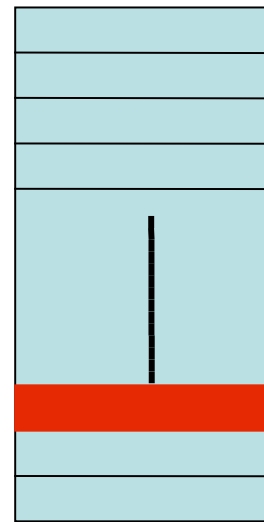
- A cada iteração de aprendizado a partição usada para validação é trocada.



1ª. iteração



2ª. iteração



3ª. iteração

Tamanho da Rede

- Geralmente uma camada escondida é suficiente.
- Em poucos casos você vai precisar adicionar uma segunda camada escondida.
- Não existe uma fórmula matemática para se encontrar o número de neurônios.
 - Empírico
- Dica prática
 - Comece com uma rede menor, pois a aprendizagem vai ser mais rápida.