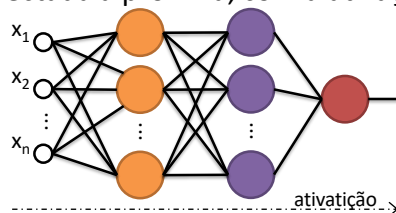


REDES MLP (*MULTI-LAYER PERCEPTRON*)

Redes Multi-Camada

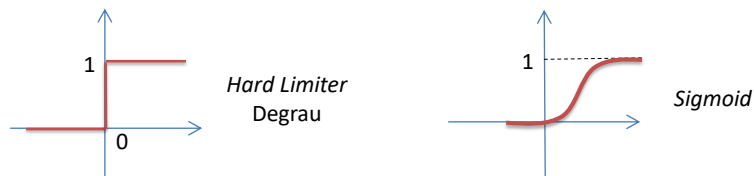
- Redes multi-camada podem representar funções arbitrárias, mas o aprendizado dessas redes era considerado um problema de difícil solução.
- Uma rede multi-camada típica consiste das camadas de **entrada**, **interna** e **saída**, cada uma totalmente conectada à próxima, com a ativação indo pra frente.



- Os pesos determinam a função calculada.
- Dado um número arbitrário de unidades internas, qualquer função booleana pode ser calculada com uma única camada interna.

Gradiente em Redes MLP

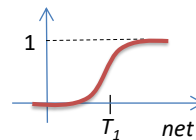
- Para fazer descida de gradiente, precisamos que a saída de uma unidade seja uma função diferenciável da entrada e dos pesos.
- A função limite padrão não é diferenciável.



Função de Saída Diferenciável

- Precisamos de uma saída não-linear.
 - Uma rede multi-camada com saídas lineares só representa funções lineares (igual a um perceptron).
- Solução padrão é usar a função não-linear e diferenciável chamada de função “logística” ou sigmóide:

$$o_j = \frac{1}{1 + e^{-(net_j - T_j)}}$$



- Também é possível utilizar *tanh* ou gaussiana.

Descida de Gradiente

- Objetivo é minimizar o erro:

$$E(W) = \sum_{d \in D} \sum_{k \in K} (t_{kd} - o_{kd})^2$$

onde D é o conjunto de exemplos de treinamento, K é o conjunto de unidades de saída, t_{kd} e o_{kd} são, respectivamente, a saída esperada e a saída atual para a unidade k para o exemplo d .

- A derivada de uma unidade sigmoidal com relação a entrada da unidade é: $\frac{\partial o_j}{\partial \text{net}_j} = o_j(1 - o_j)$
- Regra de aprendizado pra mudar os pesos e diminuir o erro é: $\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$

Backpropagation

- Cada peso deve ser modificado usando: $\Delta w_{ji} = \eta \delta_j o_i$

$$\delta_j = o_j(1 - o_j)(t_j - o_j) \quad \text{se } j \text{ for uma unidade de saída}$$

$$\delta_j = o_j(1 - o_j) \sum_k \delta_k w_{kj} \quad \text{se } j \text{ for uma unidade interna}$$

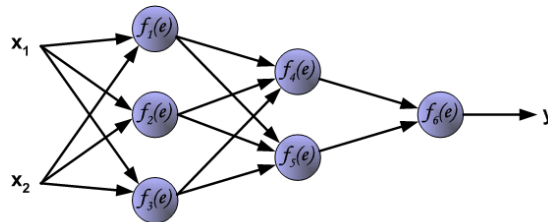
onde η é uma constante chamada de **taxa de aprendizado**

t_j é a saída correta para a unidade j

δ_j é a medida de erro para a unidade j

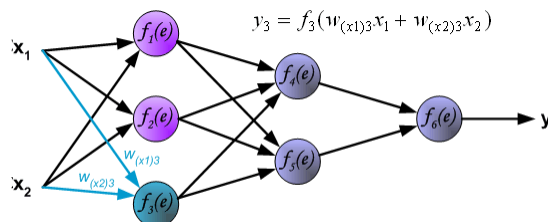
Backpropagation

- Vejamos como ocorre o treinamento de uma rede neural multi-camadas usando o algoritmo *backpropagation*. Suponha uma rede de três camadas com duas entradas (x_1 e x_2) e uma saída (y)



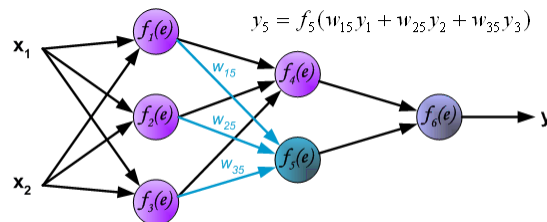
Backpropagation

- Para cada caso de treinamento, calculamos os valores de saída de cada unidade da rede para os valores de entrada.



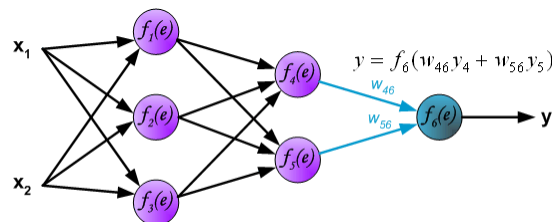
Backpropagation

- Calculados os valores dos neurônios da camada de entrada, os sinais são então propagados para a camada intermediária.



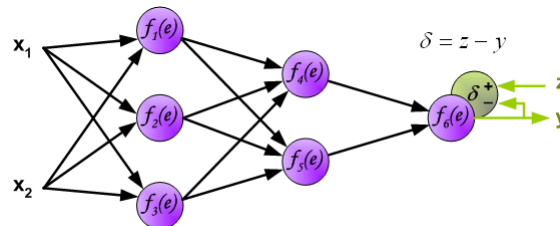
Backpropagation

- Resta ainda uma camada para ser computada, assim propagamos os sinais da camada intermediária para a camada de saída.



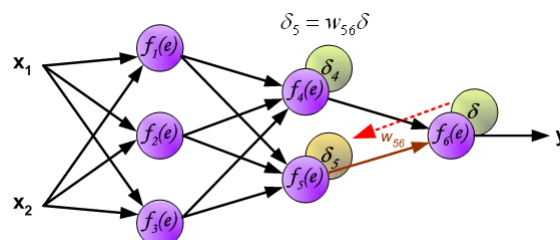
Backpropagation

- O próximo passo do algoritmo é comparar a saída esperada com a saída obtida com. A diferença (δ) é chamada de sinal de erro do neurônio da camada de saída.



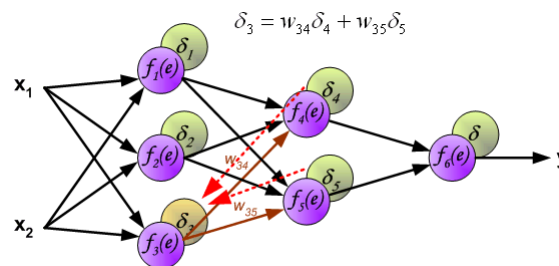
Backpropagation

- Note que é impossível calcular o erro exato associado a camada interna. Assim a ideia é propagar o erro para todos os neurônios das camadas anteriores.



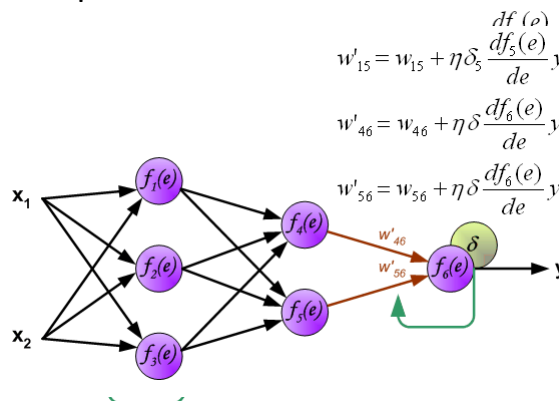
Backpropagation

- Os coeficientes de peso (w_i) são os mesmos utilizados para calcular o valor de saída. A mesma técnica é utilizada para todas as camadas da rede.



Backpropagation

- Após todos os sinais de erro serem calculados, os coeficientes de entrada são ajustados para cada um dos neurônios.



Algoritmo *Backpropagation*

Crie uma rede de 3 camadas com H neurônios na camada oculta e totalmente conectada entre as camadas. Defina pequenos valores reais aleatórios aos pesos.

Até que todos os exemplos de treinamento produzam o valor correto ou o valor médio do erro quadrático pare de decrescer, ou outro critério de término seja alcançado:

Comece uma época

Para cada exemplo de treinamento, d , faça:

Calcule a saída da rede para os valores de entrada de d .

Compute o erro entre a saída obtida e a saída esperada para d .


Atualize os pesos retropropagando o erro e usando a regra de aprendizado.

Fim da época.

Comentários sobre o algoritmo

- Não tem a convergência garantida – pode convergir para um ótimo local ou oscilar indefinidamente.
- Na prática, converge para um erro baixo para redes grandes com dados reais.
- Muitas épocas (milhares) podem ser necessárias, significando horas ou dias de treinamento para redes grandes.
- Para evitar problemas de mínimo local, executamos várias vezes com diferentes pesos aleatórios (*reinícios aleatórios*).
 - Pegamos resultado com menor erro de treinamento.
 - Podemos também construir um *ensemble* (possivelmente dando pesos de acordo com a acurácia).

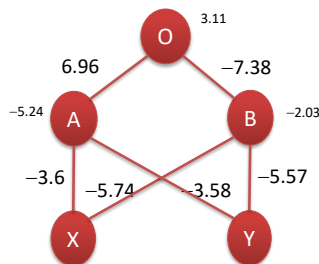




Poder de representação

- Funções booleanas: Qualquer função booleana pode ser representada por uma rede de duas camadas com número suficiente de unidades.
- Funções contínuas: Qualquer função contínua (limitada) pode ser aproximada arbitrariamente por uma rede de duas camadas.
 - Funções sigmoide funcionam como um conjunto de funções base.
- Funções arbitrárias: Qualquer função pode ser aproximada arbitrariamente por uma rede de três camadas.

Exemplo: Rede XOR aprendida



Unidade interna A representa:

$$\neg(X \wedge Y)$$

Unidade interna B representa:

$$\neg(X \vee Y)$$

Saída O representa:

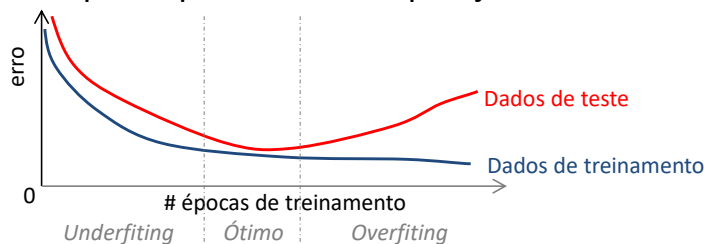
$$\begin{aligned} A \wedge \neg B &= \neg(X \wedge Y) \wedge (X \vee Y) \\ &= X \oplus Y \end{aligned}$$

Representações nas Unidades Internas

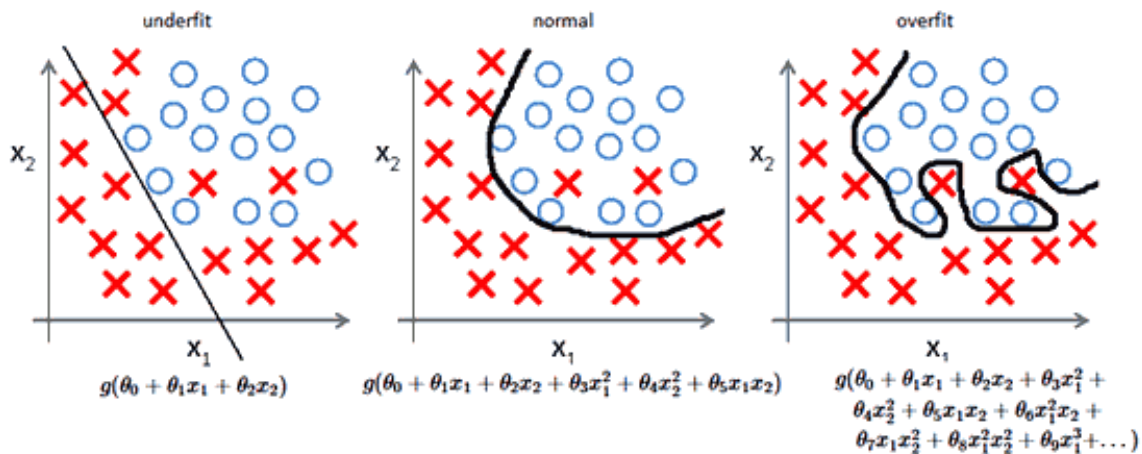
- Unidades internas treinadas podem ser vistas como um novo conjunto de atributos que fazem o conceito ser linearmente separável.
- Em muitos domínios reais, unidades internas podem ser interpretadas como representando conceitos intermediários conhecidos como detectores de vogal ou detectores de forma, etc.
- Porém a camada interna pode ser vista também como uma representação distribuída da entrada, sem representar características conhecidas.

Prevenção de Superajuste (*Overfitting*)

- Treinar por muitas épocas pode levar a superajuste.



- Usar conjunto de validação e parar quando erro começar a aumentar.
- Para não desperdiçar dados:
 - Usar validação cruzada para encontrar melhor número de épocas.
 - Treinar rede final usando todos os dados pelo mesmo número de épocas.



47

Questões em Redes Neurais

- Métodos de treinamento mais eficientes:
 - Resilient propagation (Rprop)
 - Gradiente conjugado (usa segunda derivada)
- Aprender a melhor arquitetura:
 - Aumentar a rede até ela se ajustar os dados
 - Cascade Correlation
 - Upstart
 - Diminuir a rede até que ela não se ajuste mais aos dados
 - Optimal Brain Damage



Questões em Redes Neurais

- Redes recorrentes que usam retroalimentação podem aprender máquinas de estado finito através da *“retropropagação no tempo”*
- Algoritmos mais plausíveis biologicamente.
- Aprendizado não-supervisionado
 - Self-Organizing Feature Maps (SOMs)
- Aprendizado por reforço
 - Usa-se as redes para representar funções de valor.

