Aprendizado de Máquina

Aula 7

http://www.ic.uff.br/~bianca/aa/

Tópicos

1. Introdução – Cap. 1 (16/03)
2. Classificação Indutiva – Cap. 2 (23/03)
3. Árvores de Decisão – Cap. 3 (30/03)
4. Ensembles - Artigo (13/04)
5. Avaliação Experimental – Cap. 5 (20/04)
6. Aprendizado de Regras – Cap. 10 (27/04)
7. **Redes Neurais – Cap. 4 (04/05)**
8. Teoria do Aprendizado – Cap. 7 (11/05)
9. Máquinas de Vetor de Suporte – Artigo (18/05)
10. Aprendizado Bayesiano – Cap. 6 e novo cap. online (25/05)
11. Aprendizado Baseado em Instâncias – Cap. 8 (01/05)
12. Classificação de Textos – Artigo (08/06)
13. Aprendizado Não-Supervisionado – Artigo (15/06)

Redes Neurais

- Criadas em analogia a sistemas neurais biológicos, que são capazes de aprendizagem.
- Criadas com o objetivo de entender sistemas neurais biológicos através de modelagem computacional.
  - Hoje existe uma divergência entre os modelos biológicos neurais estudados em neurociência e as redes neurais usadas em aprendizagem de máquina.

Redes Neurais

- O caráter "distribuído" das representações neurais permite robustez e degradação suave.
- Comportamento inteligente é uma propriedade "emergente" de um grande número de unidades simples ao contrário do que acontece com regras e algoritmos simbólicos.

Restrições de Velocidade Neural

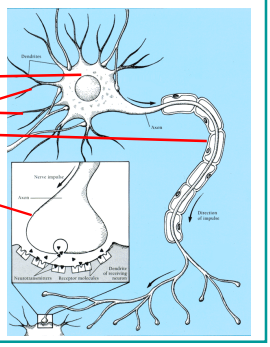
- Neurônios "ligam" e "desligam" em alguns milissegundos, enquanto o hardware atual faz o mesmo em nanossegundos.
- Porém, sistemas neurais biológicos realizam tarefas cognitivas complexas (visão, reconhecimento de voz) em décimos de segundo.
  - Só seria possível realizar 100 passos seriais nesse tempo.
- Sistema neural deve estar utilizando um "paralelismo massivo".
- Cérebro humano tem 10^{11} neurônios com uma média de 10^4 conexões cada.

Aprendizagem de Redes Neurais

- Abordagem baseada numa adaptação do funcionamento de sistemas neurais biológicos.
- Perceptron: Algoritmo inicial pra aprendizagem de redes neurais simples (uma camada) desenvolvido nos anos 50.
- Retropropagação: Algoritmo mais complexo para aprendizagem de redes neurais de múltiplas camadas desenvolvido nos anos 80.

Neurônios "Naturais"

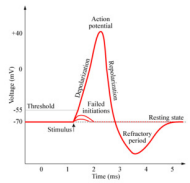
- Estruturas da Célula
  - Corpo celular
  - Dendritos
  - Axônio
  - Terminais sinápticos



7

Comunicação Neural

- Potencial elétrico através da membrana da célula exibe picos.
- Pico se origina no corpo celular, passa pelo axônio, e faz com que os terminais sinápticos soltem neurotransmissores.
- Neurotransmissores passam através das sinapses para os dendritos de outros neurônios.
- Neurotransmissores podem ser excitadores ou inibidores.
- Se a entrada total de neurotransmissores para um neurônio é excitatória e ultrapassa um certo limite, ele dispara (tem um pico).



8

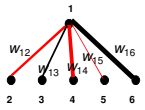
Aprendizagem de Redes Neurais

- Sinapses mudam de tamanho e força com experiência
- Aprendizagem Hebbiana: Quando dois neurônios conectados disparam ao mesmo tempo, a conexão sináptica entre eles aumenta.
- "*Neurons that fire together, wire together.*"

9

Redes Neurais Artificiais

- A rede é modelada com um grafo onde as células são nós e as conexões sinápticas são arestas de um nó *i* para um nó *j*, com pesos w_{ji}



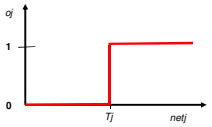
- Entrada na célula:

$$net_j = \sum_i w_{ji} o_i$$

- Saída da célula:

$$o_j = \begin{cases} 0 \text{ if } net_j < T_j \\ 1 \text{ if } net_i \geq T_j \end{cases}$$

(T_j é o limiar da unidade *j*)



10

Computação Neural

- McCollough e Pitts (1943) mostraram como neurônios desse tipo poderiam calcular funções lógicas e serem usados como máquinas de estado.
- Podem ser usados para simular portas lógicas:
  - AND: Todos w_{ji} são T_j/n, onde n é o número de portas.
  - OR: Todos w_{ji} são T_j
  - NOT: O limite é 0, entrada única com peso negativo
- Podemos construir qualquer circuito lógico, máquina sequencial e computadores com essas portas.
- Podemos representar qualquer função booleana usando uma rede de duas camadas (AND-OR).

11

Aprendizagem de Perceptrons

- Usa um conjunto de exemplos de treinamento que dá a saída desejada para uma unidade, dado um conjunto de entradas.
- O objetivo é aprender pesos sinápticos de tal forma que a unidade de saída produza a saída correta pra cada exemplo.
- O algoritmo perceptron faz atualizações iterativamente até chegar aos pesos corretos.

12

Regra de Aprendizagem de Perceptrons

- Atualizar pesos usando:

$$w_{ji} = w_{ji} + \eta(t_j - o_j)o_i$$

onde η é a "taxa de aprendizagem"

t_j é a saída especificada para a unidade *j*.

- Equivalente a:
  - Se a saída estiver correta, não fazer nada.
  - Se a saída estiver alta, baixar pesos das saídas ativas
  - Se a saída estiver baixa, diminuir pesos das saídas ativas
- Também ajusta-se o limiar:

$$T_j = T_j - \eta(t_j - o_j)$$

13

Algoritmo de Aprendizagem de Perceptrons

- Iterativamente atualizar pesos até a convergência.

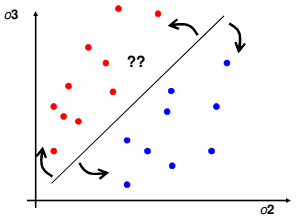
Initialize weights to random values
Until outputs of all training examples are correct
  For each training pair, *E*, do:
    Compute current output *oj* for *E* given its inputs
    Compare current output to target value, *tj* , for *E*
    Update synaptic weights and threshold using learning rule

- Cada execução do loop externo é tipicamente chamada de *época*.

14

Perceptron como Separador Linear

- Como o perceptron usa uma função de limite linear, ele procura por um separador linear que discrimine as classes.



$$w_{12}o_2 + w_{13}o_3 > T_1$$

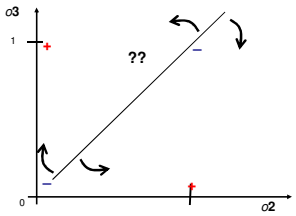
$$o_3 > -\frac{w_{12}}{w_{13}}o_2 + \frac{T_1}{w_{13}}$$

ou *hiperplano* em um espaço *n*-dimensional

15

Um conceito que o perceptron não aprende

- Não é capaz de aprender ou-exclusivo ou funções de paridade em geral.



16

Limitações do Perceptron

- Obviamente não pode aprender conceitos que não é capaz de representar.
- Minksy e Papert (1969) escreveram um livro analisando o perceptron e descrevendo funções que ele não podia aprender.
- Esses resultados desencorajaram o estudo de redes neurais e as regras simbólicas se tornaram o principal paradigma de IA.

17

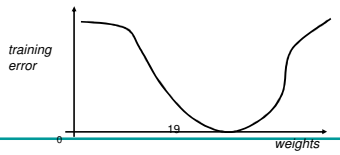
Teoremas

- **Teorema de convergência do perceptron**: Se os dados forem linearmente separáveis, então o algoritmo do perceptron irá corrigir para um conjunto consistente de pesos.
- **Teorema do ciclo do perceptron**: Se os dados não forem linearmente separáveis, o algoritmo irá repetir um conjunto de pesos e limites no final de uma época e, como consequeência entra em um loop infinito.
  - Podemos garantir término do programa checando as repetições.

18

Perceptron como Subida de Encosta

- O espaço de hipóteses é um conjunto de pesos e um limite.
- O objetivo é minimizar o erro de classificação no conjunto de treinamento.
- O perceptron efetivamente realiza uma subida de encosta (descida) neste espaço.
- Para um único neurônio, o espaço é bem comportado com um único mínimo.



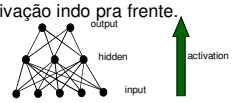
Desempenho do Perceptron

- Funções lineares são restritivas (viés alto) mas ainda razoavelmente expressivas; mais gerais que:
  - Conjuntiva pura
  - Disjuntiva pura
  - M-de-N (pelo menos M de um conjunto esperado de N características deve estar presente)
- Na prática, converge razoavelmente rápido para dados linearmente separáveis.
- Pode-se usar até resultados anteriores à convergência quando poucos outliers são classificados erroneamente.
- Experimentalmente, o Perceptron tem bons resultados para muitos conjuntos de dados.

20

Redes Multi-Camada

- Redes multi-camada podem representar funções arbitrárias, ms aprender essas 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.

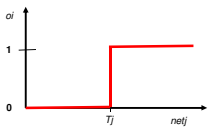


- Os pesos determinam a função calculada. Dado um número arbitrário de unidades internas, qualquer função booleana pode ser calculada com uma única camada interna.

21

Subida de Encosta em Redes Multi-Camada

- 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.

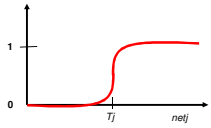


22

Função de Saída Diferenciável

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

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



Podemos também usar tanh ou gaussiana

23

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: $\frac{\partial o_j}{\partial net_j} = o_j(1 - o_j)$
- Regra de aprendizado pra mudar os pesos e diminuir o erro é:

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$

24

Regra de Aprendizado para Retropropagação

- 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

25

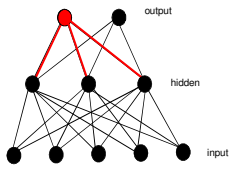
Retropropagação do erro

- Primeiro calculamos o erro das unidade de saída e usamos ele para mudar a camada superior de pesos.

Saída atual o_j=0.2
Saída correta: t_j=1.0
Erro $\delta_j = o_j(1-o_j)(t_j-o_j)$
0.2(1–0.2)(1–0.2)=0.128

Mudar pesos entrando em j com

$$\Delta w_{ji} = \eta \delta_j o_i$$

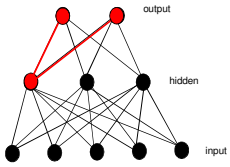


26

Retropropagação do erro

- Depois calcular erro para unidades internas baseado nos erros que as unidades de saída depositam nela.

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj}$$



27

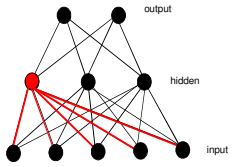
Retropropagação do erro

- Finalmente atualizamos a camada inferior de pesos baseado nos erros calculados para as unidades internas.

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj}$$

Mudar pesos entrando em j com

$$\Delta w_{ji} = \eta \delta_j o_i$$



28

Algoritmo de Retropropagação

Create the 3-layer network with *H* hidden units with full connectivity between layers. Set weights to small random real values.
Until all training examples produce the correct value (within ε), or mean squared error ceases to decrease, or other termination criteria:
  Begin epoch
  For each training example, *d*, do:
    Calculate network output for *d*'s input values
    Compute error between current output and correct output for *d*
    Update weights by backpropagating error and using learning rule
  End epoch

29

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).

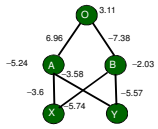
30

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.

31

Exemplo: Rede XOR aprendida



Unidade interna A representa: $\neg(X \wedge Y)$
Unidade interna B representa: $\neg(X \vee Y)$
Saída O representa: $A \wedge \neg B = \neg(X \wedge Y) \wedge (X \vee Y)$
$= X \oplus Y$

32

Representações nas Unidades Internas

- Unidade 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 conhecidas 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.

33

Prevenção de Super-Ajuste

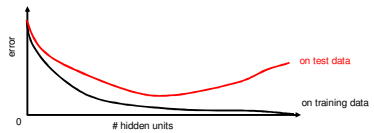
- Treinar por muitas épocas pode levar a super-ajuste.

- 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.

34

Determinando o melhor número de unidades internas

- Poucas unidades impedem a rede de se adequar totalmente aos dados.
- Muitas unidades podem resultar em super-ajuste.



- Usar validação cruzada interna para determinar empiricamente o melhor número de unidades internas.

35

Aplicações Práticas

- Texto-para-voz (NetTalk)
- Detecção de fraude
- Aplicações financeiras
  - HNC (comprada pela Fair Isaac)
- Controle na indústria química
  - Pavillion Technologies
- Automação de veículos
- Jogos
  - Neurogammon
- Reconhecimento de escrita

36

Questões em Redes Neurais

- Métodos de treinamento mais eficientes:
  - Quickprop
  - 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
- Redes recorrentes que usam retroalimentação podem aprender máquinas de estado finito através da "retropropagação no tempo"

37

Questões em Redes Neurais (cont.)

- Algoritmos mais plausíveis biologicamente.
- Aprendizado não-supervisionado
  - Self-Organizing Feature Maps (SOMs)
- Aprendizado por reforço
  - Usa-se as redes para representar funções de valor.

38