Redes Neurais Artificiais - Prova 1 - Resumo

Matheus Araujo

2018 01

Material elaborado durante estudos para a Prova 1 da disciplina *Redes Neurais Artificiais*. Professor Antônio de Pádua Braga, UFMG, 2018 01.

1 Introdução

- Redes Neurais Artificias são sistemas paralelos distribuídos compostos por unidades de processamento simples que calculam determinadas funções matemáticas.
  - As unidades são dispostas em uma ou mais camadas e interligadas por um grande número de conexões, normalmente unidirecionais.
  - Geralmente as conexões possuem pesos, que armazenam conhecimento adquirido pelo modelo e ponderam a entrada recebida por cada neurônio da rede.
  - RNAs têm como procedimento usual uma etapa de *aprendizagem*, em que a rede é apresentada a um conjunto de exemplos.
  - A rede *aprende* e *generaliza* a informação.
- O cerébro humano tem cerca de 10^{11} neurônios, sua célula fundamental. Sua rede de neurônios é capaz de reconhecer e relacionar padrões, usar e armazenar conhecimento por experiência, e interpretar observações. As RNAs tentam reproduzir as funções das redes biológicas, buscando implementar seu comportamento funcional e sua dinâmica.
- RNAs e redes biológicas têm como características comuns:
  - Sistemas baseados em unidades de computação paralela e distribuída.
  - Se comunicam por meio de conexões sinápticas[1].
  - Possuem detetores de características, redudância e modularização das conexões.
- Neurônio biológico
  - **Dentritos**, recebem as informações - *impulsos nervosos* - oriundas de outros neurônios e conduzem até o corpo celular.
  - **Corpo celular**, processa a informação e gere novos impulsos.
  - **Axônio**, transmitem a informação a outros neurônios.
  - O ponto de contato entre a terminação axônica e dentrito é chamado de **Sinapse**.
  - Em resumo, um neurônico recebe diversos sinais de neurônios anteriores através dos dentritos, caso a combinação de sinais recebidos esteja acima de um limiar de excitação, um impulso elétrico é produzido e propagado através do axônio para os próximos neurônios da rede.
- Modelo MCP
  - Proposto por McCulloch e Pitts.
  - n terminais de entrada, dentritos, $x_1, x_2, ..., x_n$, ativações dos neurônios anteriores
  - Um terminal de saída, axônio, γ.
  - Pesos acoplados $w_1, w_2, ..., w_n$, sinapses.
  - Figura 1

[1] Sináptico: algo que cria um novo caminho ou alternativa em sua vida ou de outrem, bom ou ruim

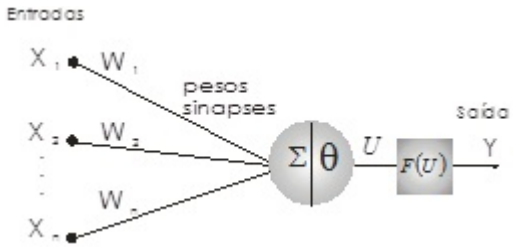

Figura 1: Modelo MCP

- Funções de ativação
  - Gera a saída γ a partir dos pesos $\mathbf{w} = \{w_1, w_2, ..., w_n\}^T$ e da entrada $\mathbf{x} = \{x_1, x_2, ..., x_n\}^T$.
  - Para o neurônio MCP, sua definição é:

$$f(u) = \begin{cases} 1 & \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

    * Onde θ é o limiar da função:

$$f(u) = \frac{1}{1 + e^{-}\beta u}$$

    * e β é a inclinação da função.
  - Existem ainda funções de ativação sigmoidal, linear e gaussiana.

- Arquiteturas de RNAs
  - **Rede *feedfoward* de uma única camada** - capaz de resolver problemas multivariáveis de múltiplas funções acopladas, com restrições de complexidade
  - **Rede *feedfoward* de duas camadas** - a camada intermediária dá uma maior capacidade computacional à rede e também universalidade na aproximação de funções contínuas.
  - As duas primeiras são redes estáticas.
  - **Rede com recorrência entre saídas e camada intermediária** - a saída depende também do valor atual, utilizada na resolução de problemas que envolvem processamento temporal.
  - **Rede de recorrência auto-associativa** - possui um único nível de neurônios em que a saída de cada um está conectada às entradas de todos os outros. É uma estrutura típica de uma rede de Hopfield.

- Aprendizado
  - Na abordagem conexionista o conhecimento é adquirido através do ajuste das intensidades das conexões entre neurônios, e não através de regras explícitas, como em na IA simbólica.
  - Portanto, aprendizado é o processo pelo qual os parâmetros livres de uma rede neural são ajustados por meio de uma forma continuada de estímulo do ambiente externo.
  - O vetor de pessos $\mathbf{w}(t+1)$ é definido por

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t)$$

  - Os algoritmos de aprendizado se diferem basicamente na forma como calculam $\Delta\mathbf{w}$.

- Aprendizado supervisionado
  - Há um *supervisor* externo responsável por estimular as entradas da rede por meio de padrões de entrada e observar a saída calculada, comparando com a saída esperada.
  - Se aplica a problemas em que se deseja obter um mapeamento entre padrões de entrada e saída.
  - Na **correção de erros**, procura-se minimizar o erro da resposta atual em relação à saída desejada iterativamente.
  - No **aprendizado por reforço**, um *crítico* externo maximiza o reforço das ações boas executadas pela rede. É um processo de tentativa e erro que visa maximizar o índice de desempenho escalar.

- Aprendizado não-supervisionado
  - Não há o papel do *supervisor*. No processo de aprendizado, a rede procura por regularidade e redundância nas entradas.
  - Se aplica a problemas que visam à descoberta de características relevantes nos dados de entrada, como a descoberta de agrupamentos ou classes.
  - O **Aprendizado Hebbiano** propõe que o peso de uma conexão sináptica deve ser ajustado se houver sincronismo entre os níveis de atividade da entrada e saída, isso é, se dois neurônios são ativados sincronamente, a sinapse entre eles é fortalecida, caso contrário enfraquecida ou mesmo eliminida.
  - No **Aprendizado por competição** é ideia é, dado um padrão de entrada, fazer com que as unidades de saída disputem entre si para serem ativadas.
- Aplicações
  - **Classificação** - atribuir um padrão desconhecido entre várias classes conhecidas. Normalmente redes com aprendizado supervisionado.
  - **Categorização** - descoberta de categorias ou classes bem definidas nos dados de entrada. As classes não são conhecidas de antemão. Normalmente, aprendizado não supervisionado.
  - **Aproximação** - mapear funções contínuas das variáveis de entrada.
  - **Previsão** - estimativa de situações futuras com base nos estados atuais e anteriores do sistema a ser modelado.
  - **Otimização** - podem ser resolvidos através de modelos recorrentes como as redes de Hopfield, os valores dos pesos são obtidos diretamente da formulação analítica do problema.

2 Perceptron

- Portas de Limiar
  - Função: comparação da soma ponderada das entradas com um valor de limiar - *threshold*
  - Podem ser: Linear, Quadrática, Polinomial
  - Porta de Limiar Linear
    - Entradas são variáveis booleanas
    - Solução de problemas que sejam linearmente separáveis, i.e., problemas cuja solução possar ser obtida pela separação de duas regiões por meio de uma reta
    - Implementa soluções para funções **E**, **OU**, **NÃO-E** e **NÃO-OU**, mas não implementa solução para **OU-EXCLUSIVO**.
  - Porta de Limiar Quadrática
    - Aumenta a capacidade computacional devido ao termo quadrático
    - A equação da superfície de decisão resultante pode assumir uma forma não-linear
    - Maior número de parâmetros livres
    - Implementa função lógica **OU**
  - *Dilema entre a polarização e a variância*: uma rede com um número muito reduzido de parâmetros pode não possuir a flexibilidade necessária para alcançar a solução desejada; por outro lado, uma rede com um número muito grande de parâmetros pode estar superdimensiodada e, consequentemente, ser excessivamente flexível.
- Perceptron
  - Composto por uma estrutura de rede, tendo como unidades básicas neurônios MCP, e por uma regra de aprendizado.
  - Um neurônio MCP treinado pelo algoritmo de aprendizado do Perceptron sempre converge caso o problema em questão seja linearmente separável
  - Na topologia original, possuía uma unidade de entrada (retina), um nível intermediário formado pelas unidades de associação e um nível de saída formado pelas unidades de resposta. Conhecida como perceptron de uma única camada, pois só o nível de saída possui propriedades adaptativas.

- De uma maneira geral, durante o processo de aprendizado deseja-se obter no instante n, o valor do incremento $\Delta\mathbf{w}(n)$ aplicado ao vetor de pesos $\mathbf{w}(n)$ de tal forma que $\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{w}(n)$ esteja mais próximo da solução desejada do que $\mathbf{w}(n)$.
- Equação geral para atualização dos pesos de um neurônio de um perceptron simples:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta e \mathbf{x}(n)$$

  * η é uma medida da rapidez com que o vetor de pesos será atualizado, ou *taxa de aprendizado.*
  * $e = y_d - y$, sendo y_d o valor desejado e y o saída obtida da rede
- Algoritmo:
  1. Inicializar η
  2. Inicializar o vetor de pesos $\mathbf{w}$ com valores aleatórios
  3. Aplicar a regra de atualização (equação geral) para todos os pares (x^i, y_d^i) do conjunto de treinamento
  4. Repetir o passo 3 até que $e = 0$ para todos os elementos do conjunto de treinamento
- Considerações sobre o algoritmo:
  * O valor de η deve ser definido pelo usuário, dependendo do problema em questão. Existem algoritmos que fazem o ajuste adaptativo de η
  * Recomenda-se iniciar o vetor de pesos com valores amostrados em uma distribuição uniforme definida no intervalo $[-a, a]$, onde a é um valor positivo próximo de 0.0, como 0.5, isso evita saturação forte do neurônio MCP, facilitando a convergência do algoritmo.
  * Os pares de entrada e saída devem ser amostrados aleatoriamente.
- Por fim, o Perceptron Simples é um discriminador que divide o espaço de entrada em duas regiões por meio de uma superfície de separação linear.
- Perceptron é uma rede neural simples para resolução de problemas de classificação.

3 Adaline

- Como o perceptron, baseia-se em elementos de processamento que fazem operações sobre a soma ponderada de suas entradas.
- Diferentemente do perceptron, que usa funções degrau; no Adaline, essas operações são puramente lineares.
- Baseia-se na magnitude e no sinal do gradiente do erro para obter a direção e o valor do ajuste $\Delta\mathbf{w}$ aplicado ao vetor de pesos.
- A equação do modelo é o produto entre os vetores $\mathbf{w}$ e $\mathbf{x}$:

$$y = f(\mathbf{x}, \mathbf{w}) = \mathbf{x} \cdot \mathbf{w} = \mathbf{x}^T\mathbf{w} = \sum_{i=0}^{n} x_i w_i$$

$$y = w_0 + w_1 x_1 + w_2 x_2 + ... + w_n x_n$$

  - $w_0 = \theta$ é o termo de polarização
  - w_i são resultado do treinamento
- Enquanto perceptron se caracteriza como separador linear, Adaline é um aproximador linear de funções. Perceptron resolve problemas de classificação de padrões e Adaline problemas aproximação de funções.
- Equação quadrática de erro

$$e^2 = (y_d^i)^2 - 2y_d^i(\mathbf{x} \cdot \mathbf{x}^i) + (\mathbf{w} \cdot \mathbf{x}^i)^2$$

  - A equação é uma superfície na forma de parábola, que pode possuir um mínimo global.
  - O objetivo do treinamento é atingir as regiões de menor erro, próximas ao mínimo global.
  - A equação acima representa o erro para um par de treinamento, $\mathbf{x}^i$, y_d^i, para todos os padrões do conjunto de treinamento a equação é, chamada função de custo:

$$J = \frac{1}{2}\sum_{i=1}^{n}(y_d - (\mathbf{w} \cdot \mathbf{x}^\mathbf{i}))^2$$

- Regra Delta
  - O objetivo do treinamento é minimizar a função de custo J.
  - A direção do ajuste na iteração n pode ser obtida pelo gradiente da função de custo no ponto $\mathbf{w}(n)$.
  - O ajuste dos pesos é então obtido por $\Delta\mathbf{w}(n) \propto -\nabla J$, expresso por:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta e\mathbf{x}(n)$$

- Adaline também pode ser usado para fazer aproximação de combinação linear de funções, por exemplo: $f(x) = f_1(x) + f_2(x) + f_3(x) = sen(x) + cos(x) + x$
- Por fim, Perpcetron e Adaline são modelos de natureza simples e resolvem apenas problemas com características lineares. No entanto, é possível resolver problemas de natureza mais complexa com eles através da utilização de portas de limiar de maior complexidade ou através da combinação de funções não-lineares.

4 Perceptron de Múltiplas Camadas

- As não-linearidades são incorporadas a modelos neurais através das funções de ativação (não-lineares) de cada neurônio da rede e da composição da sua estrutura em camadas sucessivas. Assim, a resposta da camada mais externa da rede corresponde à composição das respostas dos neurônios das camadas anteriores.
- Perceptron de Múltiplas Camadas (MLP) é a rede neural de múltiplas camadas composta por neurônios com funções de ativação sigmoidais nas camadas intermediárias.
- Redes MLP apresentam um poder computacional maior do que aquele apresentado pelas redes de uma única camada.
- Para redes de múltiplas camadas, o erro e só pode ser obtido a partir da última camada (única com saída), portanto, o problema passa a ser como calcular ou estimar o erro das camadas intermediárias.
- A solução é dada pelo algoritmo de *back-propagation* - que utiliza o gradiente descendente para estimar o erro das camadas intermediárias por meio de uma estimativa do efeito que elas causam no erro da camada de saída. Assim, o erro na saída é calculado e este é retroalimentado para as camadas intermediárias, possibilitando o ajuste dos pesos proporcionalmente aos valores das conexões entre camadas.
- Por usar gradiente descendente, são necessárias aqui funções diferenciáveis, como as sigmoidais.
- O papel das múltiplas camadas é transformar sucessivamente o problema descrito pelo conjunto de dados no espaço de entrada em uma representação tratável para a camada de saída da rede.
- Assim, por exemplo, um problema não-linearmente separável, resolvido por uma rede de duas camadas, é transformado em um problema linearmente separável pela camada intermediária, criando uma nova disposição interna à rede para os dados de entrada. A camada de saída pode então resolver o problema a partir dessa nova disposição.
- O comportamento da uma rede MLP de duas camadas pode ser descrito por meio de duas transformações sucessivas, a primeira $H(\mathbf{x}, \mathbf{w}_H)$ e a outra $Y(H(\mathbf{x}, \mathbf{w}_H); \mathbf{w}_s)$. Onde $\mathbf{w}_H$ e $\mathbf{w}_s$ são os vetores de pesos das camadas escondida e de saída, respectivamente.
- Quando se segue da primeira camada intermediária em direção à camada de saída, as funções implementadas se tornam cada vez mais complexas. Para uma rede com pelo menos duas camadas intermediárias, por exemplo:
  - Na primeira camada intermediária, cada neurônio contribui com retas para a formação da superfície no espaço de entrada;
  - Na segunda camadas intermediária, cada neurônio combina as retas descritas pela primeira camada, formando regiões convexas, com número de lados definido pelo número de neurônios ligados a ela.
  - Na camada de saída, cada neurônio forma regiões que são combinações das regiões convexas definidas pelos neurônios a ele conectados da camada anterior, formando regiões com formatos diversos.
- Dado um número suficientemente grande de unidades intermediárias, é possível formar representações internas para qualquer conjunto de padrões de entrada.

- Uma camada intermediária é suficiente para aproximar qualquer função contínua, e duas camadas intermediárias são suficientes para aproximar qualquer função.
- Funções de ativação
  - Pelo menos uma das camadas intermediárias deverá ter funções de ativação não-lineares. A utilização de funções puramente lineares em múltiplas camadas resultaria em uma rede de uma única camada, já que transformações lineares sucessivas podem ser descritas como uma única transformação linear.
  - Normalmente, funções lineares na saída são utilizadas em problemas de aproximação de funções, e funções sigmoidais em problemas de classificação, apesar de essa não ser uma regra geral.
- Número de neurônios
  - O número de neurônios determina a capacidade da rede em resolver problemas de determinada complexidade, quanto maior o número de neurônios, maior a complexidade da rede e maior a sua abrangência em termos de soluções possíveis.
  - A determinação do número de neurônios é o problema mais fundamental em aprendizado de redes neurais. Não há uma regra geral que determine de forma precisa esse número.
  - O problema básico de dimensionamento envolve um ajuste entre a complexidade do modelo neural e a complexidade do problema a ser resolvido.
  - De forma empírica, procura-se por uma rede de estrutura mínima que atenda aos requisitos de minimmização do erro quadrático do conjunto de treinamento.
  - Considere uma rede MLP com n entradas. Essa estrutura irá possuir n_p pesos:

$$n_p = (n \cdot N_e + N_e) + (N_e \cdot m + n)$$

    * N_e e m correspondem aos termos de polarização dos neurônios das camadas escondidas e da saída, respectivamente.
    * n e m são inerentes ao problema, somente N_e é um parâmetro que altera a estrutura e a complexidade da rede.
    * Aumentar o número de neurônios na camada escondida N_e causará um aumento linear no número total de parâmetros n_p da rede.
    * Duas redes R_1 e R_2 com números diferentes de parâmetros $n_{p1} > n_{p2}$ e $\Delta n_p = n_{p1} - n_{p2}$.
    * As soluções alcançadas por R_1 serão todas as descritas por R_2 mais aquelas proporcionadas pela diferença no número de parâmetros Δn_p.
    * O aumento no número de parâmetros causará, portanto, aumento no número de soluções possíveis.
    * Considerando-se somente o erro de treinamento como critério de seleção do modelo, quanto maior o número de neurônios na rede, maior será o número de soluções que atendem ao critério de minimização do erro. No entanto, as soluções que se aproximam da função geradora dos dados estão em um número restrito no espaço de soluções e não aumenta em quantidade com o aumento do número de parâmetros. Assim, quanto maior o número de parâmetros, mais difícil é, na verdade, a busca pelas soluções que se aproximam da função geradora dos dados.
    * Portanto, o objetivo do treinamento de redes MLP é minimizar não somente o erro do conjunto de treinamento, mas também a estrutura ou complexidade da rede.
- Treinamento
  - *Back-propagation* é um algoritmo de treinamento de redes MLP supervisionado.
  - Por meio de pares de entrada e saída ($\mathbf{x}$, y_d) e um mecanismo de correção de erros ajusta os pesos da rede.
  - O treinamento ocorre em duas fases:
    * A primeira, *forward*, é utilizada para definir a saída da rede para um dado padrão de entrada
    * A segunda, *backward*, utiliza a saída desejada e a saída da rede para atualizar os pesos de suas conexões, baseando-se na mesma Regra Delta usada pelo Adaline.
  - Etapa *forward*:
    1. O vetor de entrada $\mathbf{x}$ é apresentado às entradas da Rede e as saídas dos neurônios da primeira camada C_1 escondida é calculada
    2. As saídas da camada esconsida C_1 proverão as entradas da camada seguinte C_2. As saídas dessa camada são calculadas e o processo se repete até que se chegue à camada de saída C_k.

3. As saídas produzidas pelos neurônios da camada de saída são então comparadas às saídas desejadas y_d para o vetor de entrada $\mathbf{x}$, e o erro correspondente é calculado.

– Etapa *backward*:

1. O erro da camada de saída C_k é usado para ajustar seus pesos utilizando o gradiente descendente do erro.
2. Os erros dos neurônios da camada C_k são propagados para a camada anterior C_{k-1}, utlizando-se os pesos das conexões entre as camadas, que são multiplicados pelos erros correspondentes. Obtendo-se um valor de erro estimado para cada neurônio da camada escondida.
3. Os erros calculados para a camada C_{k-1} são então utilizados para ajustar os seus pesos pelo gradiente descendente.
4. O processo se repete até que os pesos da camada C_1 sejam ajustados.

– O ajuste dos pesos na camada de saída é dado por:

$$\Delta w_{ij} = \alpha e_j(n) f'(u_j(n)) h_i(n)$$

* i é o índice do neurônio da camada escondida
* j é o índice dos neurônios da camada de saída
* $e_j(n)$ é o erro apresentado na saída para a entrada n
* $h_i(n)$ é a saída do i-ésimo neurônio da camada escondida
* $u(n)$ é o somatório das saídas da camada escondida ponderada pelos pesos w_{jk}
* f' é a derivada da função de ativação da camada de saída
* α é a taxa de aprendizado

– Para a camada de saída, a equação é:

$$\Delta w_{jk} = \eta h_i'(u_i(n)) \sum_j e_j(n) f'(u_j(n)) w_{ji} x_k(n)$$

* k é o índice da entrada x
* h' é a derivada da função de ativação da camada escondida
* η é a taxa de aprendizado