Universidade Federal do Ceará
Departamento de Estatística e Matemática Aplicada
Introdução à Mineração de Dados (CC0101), período 2019.2
Professor: Tibérius O. Bonates (tb@ufc.br).

Roteiro de Aula sobre Redes Neurais Artificiais
(Estas não são notas de aula. Por gentileza, não distribua este material. Ele é para uso exclusivo na disciplina identificada neste documento.)

Este material é baseado no livro de Tom Mitchell e, em menor grau, no livro de Faceli *et al.*

Situações típicas de uso de redes neurais artificiais (RNAs):

- É aceitável que o tempo de treinamento seja lento;
- O tempo de classificação deve ser rápido;
- Os dados podem conter ruído;
- A hipótese (função de classificação) não precisa ser compreensível.

1 Perceptron

O perceptron é um modelo simplificado de neurônio, proposto por McCullock e Pitts, em 1943. Ele consiste essencialmente de uma função real de n argumentos, chamada de *função de ativação*, onde n é o número de características dos dados. Funções de ativação tipicamente utilizadas em um classificador do tipo perceptron são:

$$o(\mathbf{x}) = w_0 + w_1 x_1 + \ldots w_n x_n,$$

ou

$$o(\mathbf{x}) = \text{sinal}(w_0 + w_1 x_1 + \ldots w_n x_n),$$

onde $\text{sinal}(z) = 1$, se $z > 0$ e $\text{sinal}(z) = -1$, se $z < 0$, e $\text{sinal}(z) = 0$, caso contrário.

O perceptron é semelhante a um classificador linear, discutido no contexto de máquinas de vetores suportes. Mas, aqui temos uma definição de erro diferente, que será usada adiante para treinar um classificador composto de várias unidades do tipo perceptron, em vez de apenas um único perceptron.

Vamos assumir que a informação de classe no conjunto S satisfaz $y^i \in \{-1, +1\}$, $\forall i$. Uma forma de treinar um perceptron é, além das idéias já vistas para o classificador linear, utilizar a fórmula conhecida como *regra perceptron*:

$$\begin{aligned} w_0 &:= w_0 + \eta(t - o), \\ w_j &:= w_j + \eta(t - o)x_j^i, \quad j = 1, \ldots, n, \end{aligned}$$

onde t é a classe de $\mathbf{x}^i$, o é o valor de saída do perceptron para $\mathbf{x}^i$, x_j^i é a j-ésima componente de $\mathbf{x}^i$, e η é a *taxa de aprendizado*, ou o tamanho do passo utilizado na atualização. Por exemplo, com a segunda definição da função o, teríamos

$$\begin{aligned} w_0 &:= w_0 + \eta(y^i - \text{sinal}(w_0 + w_1 x_1^i + \ldots + w_n x_n^i)), \\ w_j &:= w_j + \eta\left(y^i - \text{sinal}(w_0 + w_1 x_1^i + \ldots + w_n x_n^i)\right) x_j^i, \quad j = 1, \ldots, n. \end{aligned}$$

Considere $\eta = 0.1$, $x_j^i = 0.6$, $y^i = -1$ e $o(\mathbf{x}^i) = +1$. Como $-1 = y^i = t \neq o = o(\mathbf{x}^i) = +1$, temos que os pesos atuais fazem com que o perceptron cometa um erro na classificação do exemplo $\mathbf{x}^i$. Isso leva a um ajuste moderado nos pesos, no intuito de modificá-los de maneira que seja mais provável que a classificação de $\mathbf{x}^i$ em um momento subsequente venha a ser correta. A regra vai atualizar w_j, somando a este o valor -0.12, de forma que a contribuição de x_j^i passe a ser menos influente em

tornar o valor de o positivo. O ajuste é feito sobre todos os pesos w_j, $j = 0, 1, \ldots, n$. Note que se $t = o$, a regra não atualiza os pesos.

No caso linearmente separável, a regra do perceptron converge em um número finito de iterações, com η escolhido adequadamente. Para o caso não linearmente separável, a regra converge apenas de maneira assintótica. Neste caso, a chamada *regra delta* converge para uma melhor aproximação da separação exata, em certo sentido. Na seção seguinte, discutimos esta regra.

1.1 Regra delta

Para explicar a regra delta, vamos considerar o caso do perceptron sem a função sinal:

$$o(\mathbf{x}) = w_0 + w_1 x_1 + \ldots w_n x_n,$$

Usando $t_i = y^i$ e $o_i = o(\mathbf{x^i})$, vamos definir o erro associado a um vetor w da seguinte forma:

$$\begin{aligned} E(w) &= \frac{1}{2} \sum_{(\mathbf{x}^i, y^i) \in S} (t_i - o_i)^2 \\ &= \frac{1}{2} \sum_{(\mathbf{x}^i, y^i) \in S} \left(y^i - w_0 - \sum_{j=1}^{n} w_j x_j^i \right)^2 . \end{aligned}$$

O erro agora é uma função convexa, pois a hessiana de E é uma matriz semidefinida positiva, já que é uma matriz de produtos internos (matriz de Gram): $\dfrac{\partial}{\partial w_k}\left(\dfrac{\partial E}{\partial w_\ell}\right) = \displaystyle\sum_{(\mathbf{x}^i, y^i) \in S} x_k^i x_\ell^i = \langle v_k, v_\ell \rangle$, com v_k sendo o vetor correspondente à k-ésima característica, ao longo de todo o conjunto de treinamento (e similarmente para v_ℓ). Se aplicarmos a ideia de descida de gradiente a este erro, e escolhermos o tamanho de passo (η) cuidadosamente, o algoritmo convergirá para um vetor w^*, mínimo global da função E.

De acordo com o procedimento de descida de gradiente, o gradiente da função de erro $E(w)$, dado por

$$\nabla E(w) = \left(\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n} \right),$$

é usado para atualizar w conforme

$$w_j := w_j + \eta \cdot \left(-\frac{\partial E}{\partial w_j} \right), \; j = 0, 1, \ldots, n,$$

no intuito de reduzir $E(w)$. Uma observação: devido à existência do termo independente w_0 na expressão da função $o(\mathbf{x})$, é conveniente assumir que $x_0 = 1$, para que a expressão acima faça sentido.

A expressão do gradiente pode ser obtida como segue:

$$
\begin{aligned}
\frac{\partial E}{\partial w_j} &= \frac{\partial}{\partial w_j}\left[\frac{1}{2}\sum_{(\mathbf{x}^i,y^i)\in S}(t_i-o_i)^2\right] \\
&= \frac{1}{2}\frac{\partial}{\partial w_j}\left[\sum_{(\mathbf{x}^i,y^i)\in S}(t_i-o_i)^2\right] \\
&= \frac{1}{2}\left[\sum_{(\mathbf{x}^i,y^i)\in S}2(t_i-o_i)\frac{\partial}{\partial w_j}(t_i-o_i)\right] \\
&= \sum_{(\mathbf{x}^i,y^i)\in S}(t_i-o_i)\frac{\partial}{\partial w_j}(t_i-o_i) \\
&= \sum_{(\mathbf{x}^i,y^i)\in S}(t_i-o_i)\frac{\partial}{\partial w_j}\left(t_i-\sum_k w_k x_k^i\right) \\
&= \sum_{(\mathbf{x}^i,y^i)\in S}(t_i-o_i)\,(-x_j^i).
\end{aligned}
$$

Agora, a atualização dos pesos está completamente especificada, com

$$
w_j := w_j + \eta\cdot\sum_{(\mathbf{x}^i,y^i)\in S}(t_i-o_i)\,x_j^i,\ j=0,1,\ldots,n.
$$

Uma implementação típica da regra delta se inicia com pesos aleatórios, próximos de zero. A seguir, são realizadas várias *épocas* de treinamento: cada época consiste em uma passagem completa por todos os exemplos de treinamento, para calcular o gradiente. Ao final de cada época, ∇E é calculado e os pesos são atualizados, conforme acima. Como critério de parada, utiliza-se um número máximo de épocas, e/ou alguma noção de convergência dos pesos. O valor de η costuma ser inicializado no intervalo $[0.1, 0.3]$ e é reduzido lentamente, à medida em que o erro diminui.

Algorithm 1: `DescGradPerceptron(`S`)`: – Descida de gradiente para perceptron.

ENTRADA: conj. de treinamento $S = \{(\mathbf{x}^i, y^i) : i = 1,\ldots,m\}$, taxa η, máx. de épocas M
SAÍDA : Vetor de pesos w.

```
Inicializar w aleatoriamente
epoca := 0
while epoca < M do
    Δw_j := 0, j = 0, 1, ..., n
    for (x^i, y^i) ∈ S do
        Δw_j := Δw_j + (t_i − o_i) x_j^i, j = 0, 1, ..., n
    if ||Δw|| ≈ 0 then
        Parar
    w_j := w_j + ηΔw_j, j = 0, 1, ..., n
    Atualizar η
    epoca := epoca + 1
```

Note que, se o classificador obtido dessa forma tem erro nulo em S ($t_i = o_i$, para todo $\mathbf{x}^i$), então estes pesos também resultariam em erro nulo se usássemos a função sinal para o perceptron. Mesmo que o erro não seja nulo, desde que o sinal de o_i seja correto, o perceptron com a função sinal estaria correto na classificação de $\mathbf{x}^i$.

Assim como aconteceu quando definimos o erro para classificadores lineares, a noção de erro utilizada neste material não corresponde ao **número** de exemplos incorretamente classificados, mas a uma medida de magnitude do erro.

Uma versão "estocástica" do Algoritmo 1 tambem é usada, com atualizações feitas com base em cada observação, com a mudança do passo 6 para $w_j := w_j + \eta\,(t_i - o_i)\,x_j^i$, com a eliminação dos passos 7 a 9, e com a introdução de uma lógica para determinar que não houve mudança significativa de pesos durante a última passada por todo o conjunto S.

Diferentes funções são usadas no lugar da função sinal, no caso de redes de perceptron.

2 Redes de múltiplas camadas com o algoritmo de retropropagação

Nesse texto, nosso conceito de rede de múltiplas camadas é o de um conjunto de neurônios interconectados (as saídas de certos neurônios são as entradas de outros), de maneira não cíclica. Isto é, não existe uma sequência de neurônios de tal forma que a saída de um neurônio seja uma das entradas do próximo neurônio da sequência, e a saída do último neurônio seja uma das entradas do primeiro. Nós estamos fazendo essa restrição aqui, mas existem algoritmos de treinamento de redes neurais para lidar com o caso de conexões que formam tais ciclos.

Chamaremos de *nó* toda entrada da rede (isto é, toda característica da observação de entrada), assim como toda saída de algum neurônio (ou unidade) da rede.

A combinação de neurônios em rede permite a combinação da capacidade de aprendizado de neurônios individuais, para formar superfícies de decisão complexas. Por enquanto, considere a rede composta por apenas dois neurônios, a_1 e a_2, cada um com saída dada por uma função linear (isto é, do tipo $o(\mathbf{x}) = w_0 + \sum_j w_j x_j$). Vamos definir as entradas de a_1 como sendo todas as n características do conjunto de treinamento, e as entradas de a_2 como sendo a saída de a_1 e todas as n caraterísticas do conjunto de treinamento. Isto é, a entrada de a_2 inclui uma dimensão a mais: uma combinação linear das n características originais. Apesar dessa entrada adicional, a saída de a_2 ainda é apenas uma combinação linear das características originais:

$$\begin{aligned} o^2(\mathbf{x}) &= w_0^{(2)} + \sum_{j=1}^{n} w_j^{(2)} x_j + w_{n+1}^{(2)} o^{(1)}(\mathbf{x}) \\ &= w_0^{(2)} + \sum_{j=1}^{n} w_j^{(2)} x_j + w_{n+1}^{(2)} \left(w_0^{(1)} + \sum_{j=1}^{n} w_j^{(1)} x_j \right) \\ &= \alpha_0 + \sum_{j=1}^{n} \alpha_j x_j, \end{aligned}$$

onde o superíndice em $w_j^{(i)}$ e em $o^{(i)}$ corresponde ao neurônio a_i, com $i \in \{1, 2\}$, e onde os valores

α_j são obtidos desenvolvendo-se a expressão e se somando os termos independentes e os coeficientes de cada x_j.

Portanto, para que redes de neurônios interconectados realmente se beneficiem da interconexão das unidades individuais, criando superfícies de decisão mais complexas, é preciso que a função de ativação seja não-linear. E para utilizar a técnica de descida de gradiente é preciso que a função seja, também, diferenciável. Para demonstrar o algoritmo clássico de treinamento de redes acíclicas, vamos usar a função sigmóide, ou logística, que atende a estes requisitos e é dada por

$$\sigma(y) = \frac{1}{1+e^{-y}}.$$

A função sigmóide é adequada porque fornece uma aproximação da função sinal, usada no início desse texto. Essa função representa uma aproximação do comportamento do modelo de neurônio, em que um sinal é propagado por um neurônio sempre que o neurônio é suficientemente excitado por meio dos sinais recebidos a partir de outro neurônios (via conexão sináptica entre axônios e dendritos).

A função $o(\mathbf{x})$ usada para determinar a saída de um neurônio é chamada de *função de ativação*. No restante deste texto, iremos usar como função de ativação $o(\mathbf{x}) = \sigma\left(w_0 + \sum_{j=1}^{n} w_j x_j\right)$.

Usaremos adiante o fato de que a derivada de σ pode ser expressa em termos de σ novamente:

$$\begin{aligned}
\frac{\partial \sigma}{\partial y} &= -\frac{1}{(1+e^{-y})^2} \cdot (-e^{-y}) \\
&= \frac{1}{(1+e^{-y})^2} \cdot e^{-y} \\
&= \sigma(y) \cdot \frac{e^{-y}}{(1+e^{-y})} \\
&= \sigma(y) \cdot \left(1 - \frac{1}{(1+e^{-y})}\right) \\
&= \sigma(y) \cdot (1 - \sigma(y)) \,.
\end{aligned}$$

Em redes de múltiplas camadas, chamaremos de *camada de entrada* as características da observação. A *camada de saída* será composta de dois ou mais neurônios, um para cada classe no conjunto de treinamento. A rede pode ou não ser composta de outras camadas, com diferentes números de neurônios, que serão todas chamadas de *camadas escondidas* ou *camadas internas*. Tipicamente, as entradas dos neurônios de uma dada camada são as saídas de todos os nós da camada anterior. No entanto, outras topologias são possíveis. Todas estas conexões possuem pesos associados a elas, que devem ser ajustados na fase de treinamento.

Dada uma observação $(\mathbf{x}, y)$ do conjunto de treinamento, a saída ideal da rede para classificar $\mathbf{x}$ como percente à classe y é aquela em que o único neurônio da camada de saída que fornece uma saida não-nula é aquele associado a y; neste caso, a saída ideal do neurônio associado a y deve ser 1. Para o propósito de classificar uma nova observação $\mathbf{x}$ utiliza-se a classe associada ao neurônio da camada de saída cuja saída é mais positiva.

Denotando o conjunto de índices dos nós da camada de saída por T, podemos escrever o erro associado à classificação de um exemplo $\mathbf{x}$, dado um conjunto de pesos w, como

$$E(\mathbf{x}, w) = \frac{1}{2} \sum_{k \in T} (t_k - o_k)^2, \tag{1}$$

onde o_k é a saída do k-ésimo neurônio da camada de saída quando $\mathbf{x}$ é fornecida à rede, e t_k é a saída ideal para o k-ésimo neurônio da camada de saída quando $\mathbf{x}$ é fornecido à rede. Esta saída ideal é $t_k = 1$, quando o k-ésimo neurônio de S está associado à classe real de $\mathbf{x}$, e $t_k = 0$, para todos os demais valores de k.

Podemos, agora, tentar aplicar a ideia de descida de gradiente para ajustar os pesos da rede de forma a minimizar o erro total $E = \sum_{\mathbf{x} \in S} E(\mathbf{x}, w)$. Por simplicidade, faremos a derivação das regras de atualização para o caso de descida de gradiente estocástica, na qual os pesos são ajustados após a apresentação de cada exemplo do conjunto de treinamento (de acordo com o erro dado por (1)), e não ao final de cada época. Convém notar que, neste caso, a expressão do erro envolve produtos de muitas variáveis simultaneamente e já não temos a garantia de encontrar um mínimo **global** via descida de gradiente (como era no caso do treinamento de um único neurônio).

Usaremos, a seguir, a seguinte notação:

- x_{ji}: a i-ésima entrada do neurônio j;
- w_{ji}: o peso associado à i-ésima entrada do neurônio j;
- tot_j: o valor de $w_{j0} + \sum_i w_{ji} x_{ji}$;
- o_j: o valor de $o(\mathbf{x})$ no neurônio j;
- t_j: o valor ideal de saída do neurônio j;
- δ_j^+: o conjunto de neurônios que possuem como entrada (direta) a saída do neurônio j.

Note que toda a influência de w_{ji} sobre o erro total se dá como resultado de tot_j. Portanto, podemos aplicar a regra da cadeia da seguinte forma:

$$\frac{\partial E(\mathbf{x}, w)}{\partial w_{ji}} = \frac{\partial E(\mathbf{x}, w)}{\partial \text{tot}_j} \frac{\partial \text{tot}_j}{\partial w_{ji}} = \frac{\partial E(\mathbf{x}, w)}{\partial \text{tot}_j} x_{ji}. \tag{2}$$

Para determinar $\dfrac{\partial E(\mathbf{x}, w)}{\partial \text{tot}_j}$, iremos considerar dois casos separadamente: o caso de um neurônio na camada de saída S, e o caso de um neurônio em uma camada escondida.

Caso 1: Neurônio (de índice j) na camada de saída:

$$\frac{\partial E(\mathbf{x}, w)}{\partial \text{tot}_j} = \frac{\partial E(\mathbf{x}, w)}{\partial o_j} \frac{\partial o_j}{\partial \text{tot}_j} \tag{3}$$

Como o_j é apenas a aplicação da função sigmóide a tot_j, temos $\dfrac{\partial o_j}{\partial \text{tot}_j} = o_j(1 - o_j)$. Desenvolvendo o cálculo de $\dfrac{\partial E(\mathbf{x}, w)}{\partial o_j}$, e fazendo uso de (1), temos:

$$\begin{aligned}
\frac{\partial E(\mathbf{x}, w)}{\partial o_j} &= \frac{\partial}{\partial o_j}\left[\frac{1}{2}\sum_{k\in S}(t_k - o_k)^2\right] \\
&= \frac{\partial}{\partial o_j}\left[\frac{1}{2}(t_j - o_j)^2\right] \\
&= \frac{1}{2}\,2\,(t_j - o_j)\frac{\partial(t_j - o_j)}{\partial o_j} \\
&= -(t_j - o_j). \qquad (4)
\end{aligned}$$

Daí, combinando (2), (3), e (4), temos

$$\frac{\partial E(\mathbf{x}, w)}{\partial w_{ji}} = -(t_j - o_j)\; o_j\; (1 - o_j)\; x_{ji},$$

e a atualização do peso w_{ji} pode ser feita no sentido contrário ao do gradiente, como

$$w_{ji} := w_{ji} + \eta\; \left[(t_j - o_j)\; o_j\; (1 - o_j)\; x_{ji}\right].$$

Caso 2: Neurônio (de índice j) de uma camada escondida:

Neste caso, faremos uso da definição de δ_j^+ com o intuito de contabilizar o efeito da saída do neurônio j no erro $E(\mathbf{x}, w)$. Como tot_j influencia o erro $E(\mathbf{x}, w)$ apenas por intermédio dos neurônios imediatamente adiante a partir do neurônio j, podemos escrever

$$E(\mathbf{x}, w) = \sum_{k\in\delta_j^+} F_k(tot_k) + \cdots,$$

onde $F_k(tot_k)$ corresponde à contribuição da saída do k-ésimo neurônio adiante do neurônio j, na camada imediatamente seguinte, para o valor da função de erro.

Com isso, podemos expressar $\frac{\partial E(\mathbf{x},w)}{\partial \mathrm{tot}_j}$ (da Eq. (2)) para o caso de um neurônio de uma camada interna como

$$\begin{aligned}
\frac{\partial E(\mathbf{x}, w)}{\partial \mathrm{tot}_j} &= \sum_{k\in\delta_j^+}\left[\frac{\partial F_k(tot_k)}{\partial tot_j}\right] \\
&= \sum_{k\in\delta_j^+}\left[\frac{\partial F_k(tot_k)}{\partial tot_k}\;\frac{\partial tot_k}{\partial tot_j}\right] \\
&= \sum_{k\in\delta_j^+}\frac{\partial E(\mathbf{x}, w)}{\partial \mathrm{tot}_k}\;\frac{\partial \mathrm{tot}_k}{\partial \mathrm{tot}_j} \\
&= \sum_{k\in\delta_j^+}\frac{\partial E(\mathbf{x}, w)}{\partial \mathrm{tot}_k}\;\frac{\partial \mathrm{tot}_k}{\partial o_j}\;\frac{\partial o_j}{\partial \mathrm{tot}_j},
\end{aligned}$$

onde o último passo advém do fato de que tot_j influencia tot_k por intermédio de o_j, que é uma das entradas do neurônio k.

Já que tot_k possui o_j como uma de suas entradas, ponderada por w_{kj}, temos $\frac{\partial \text{tot}_k}{\partial o_j} = w_{kj}$. Além disso, $\frac{\partial o_j}{\partial \text{tot}_j}$ é a aplicação da derivada da função sigmóide, o que nos dá $o_j(1 - o_j)$.

Assim, obtemos

$$\frac{\partial E(\mathbf{x}, w)}{\partial \text{tot}_j} \quad = \quad o_j \cdot (1 - o_j) \cdot \sum_{k \in \delta_j^+} \frac{\partial E(\mathbf{x}, w)}{\partial \text{tot}_k} \, w_{kj}.$$

A atualização do peso w_{ji} é dada por

$$w_{ji} := w_{ji} + \eta \cdot x_{ji} \cdot o_j \cdot (1 - o_j) \cdot \sum_{k \in \delta_j^+} \frac{\partial E(\mathbf{x}, w)}{\partial \text{tot}_k} \, w_{kj}.$$

Cada termo $\dfrac{\partial E(\mathbf{x}, w)}{\partial \text{tot}_k}$ refere-se a um neurônio da camada seguinte à camada do j-ésimo neurônio. Portanto, $\dfrac{\partial E(\mathbf{x}, w)}{\partial \text{tot}_k}$ pode ser calculado previamente ao seu uso nesta expressão. De fato, podemos proceder de forma sequencial, calculando os ajustes de pesos a partir da camada de saída (utilizando o **Caso 1** acima), calculando, em seguida, os ajustes para a camada anterior à camada de saída (utilizando o **Caso 2**), e assim por diante, retrocedendo para a camada imediatamente anterior, e continuando desta maneira até chegarmos à primeira camada escondida. Ao fazer isso, temos sempre o ajuste dos pesos de uma camada interna definido em função do gradiente calculado na camada seguinte. Isto tem o efeito de propagar retroativamente para as camadas anteriores o erro detectado na camada de saída, sendo esta a razão pela qual esta regra de atualização de pesos é conhecida como *backpropagation*.