Uso de Redes Neurais Artificiais MLP para o Problema de Reconhecimento de Padrões

Alexandre de Oliveira Stauffer

Junho de 2003.

Sumário

1 Introdução

Nesse trabalho irei abordar o problema da utilização de redes neurais artificiais, especificamente as MLP (Multilayer Perceptron) para resolver problemas de reconhecimento de padrões, mais especificamente para resolver problemas de classificação. A área de reconhecimento de padrões estuda métodos que possibilitem o computador a aprender e reconhecer modelos. São problemas clássicos de reconhecimento de padrões:

- Reconhecimento do dono de uma voz;
- Reconhecimento de caracteres manuscritos;
- Identificação de uma pessoa através da digital, íris do olho, etc.
- Data Mining

Ao longo deste trabalho irei utilizar como exemplo dois problemas de classificação:

- Reconhecimento de caracteres manuscritos: nesse problema é apresentado a rede neural um vetor de características extraídas de uma imagem "scaneada" de um caractere escrito a mão. A rede neural é então desafiada a identificar a que classe (A, B, C, ..., Z, 0, 1, ..., 9) esse caractere pertence. Um trabalho semelhante é desenvolvido pelo Laboratório de Inteligência Computacional (LABIC - NCE/UFRJ). Nesse laboratório, foi estudado o problema de reconhecido isolado de letras e dígitos manuscritos. A partir de dados fornecidos por esse grupo, realizei a aplicação de alguns dos métodos descritos nesse trabalho como forma de exemplificação. Entretanto, ao contrário do LABIC, abordei o reconhecimento de letras e dígitos misturados.
- Classificação para escolha de um certo tratamento médico: nesse problema é apresentado a rede neural um vetor de características que contam a história do paciente e a sua situação no momento. Essas características podem ser, por exemplo: idade, história de diabetes na família, pressão sangüinea, taxa de glicose no sangue, entre outros. Supondo que um médico possui três tipos de tratamentos diferentes, a rede neural deve ser capaz de classificar esse paciente em uma dessas três classes, agilizando a ação do médico.

Para facilitar a compreensão, esse trabalho foi organizado nos seguintes capítulos:

- Capítulo 2 - Redes Neurais Artificiais: nesse capítulo discutiremos o modelo artificial de um neurônio. Em seguida estudaremos as redes com mais de uma camada (MLP) e o algoritmo de treinamento backpropagation.

- Capítulo 3 - Parâmetros de uma Rede: nesse capítulo veremos os parâmetros que influenciam na performance e eficácia de uma rede neural MLP.
- Capítulo 4 - Pré e Pós-Processamento dos Dados: nesse capítulo veremos os tipos de análise e processamento que podemos fazer na entrada e na saída da rede neural para avaliar a sua eficácia e a complexidade do problema.

2 Redes Neurais Artificiais

2.1 Modelo Inicial do Perceptron

Na década de 40, Rosemblat apresentou o primeiro modelo de um neurônio: o perceptron. Ele possui como entrada um vetor de dados booleanos e utiliza um vetor de pesos (da mesma dimensão que o vetor de entradas) e um valor chamado normalmente de bias (ou threshold) para decidir pela geração ou não de um impulso de resposta. Se considerarmos $X = \begin{bmatrix} X_1 & X_2 & \dots & X_{n-1} \end{bmatrix}$ o vetor de entrada, $W = \begin{bmatrix} W_1 & W_2 & \dots & W_{n-1} \end{bmatrix}$ o vetor de pesos e B o bias, o impulso será gerado se e somente se $XW^T \geq B$, que equivale a:

$$XW^T \geq B \Rightarrow X_1W_1^T + X_2W_2^T + \ldots + X_{n-1}W_{n-1}^T \geq B \Rightarrow \sum_{i=1}^{n-1} X_iW_i^T \geq B$$

Com isso, podemos concluir que o Perceptron divide o espaço em dois planos (no caso de um problema em duas dimensões) ou hiperplanos (se generalizarmos para mais dimensões). Portanto, o Perceptron só é capaz de reconhecer classes que sejam linearmente separáveis, isto é, que possam ser dividas com um plano (ou hiperplano).

2.2 Treinamento do Perceptron

Para que o neurônio seje capaz de reconhecer certos padrões, é necessário que seus pesos e o bias sejam ajustados de forma ao neurônio gerar impulsos nas ocasiões apropriadas, isto é, é preciso que o neurônio seje treinado. Para isso, vamos utilizar amostras, cujo resultado é conhecido, para ajustar o vetor de pesos e o bias. Para facilitar, vamos transformar o bias em mais um elemento do vetor de pesos, da seguinte forma:

$$XW^T \geq B \Rightarrow XW^T - B \geq 0 \Rightarrow \overline{XW}^T \geq 0, \qquad \begin{array}{l} \overline{X} = \begin{bmatrix} X_1 & X_2 & \dots & X_{n-1} & -1 \end{bmatrix} \\ \overline{W} = \begin{bmatrix} W_1 & W_2 & \dots & W_{n-1} & B \end{bmatrix} \end{array}$$

Assim somente será necessário ajustar o vetor $\overline{W}$. Para facilitar a compreensão do algoritmo de treinamento, vamos supor que um Perceptron possui um vetor inicial de pesos W (incluindo o bias), um vetor de entradas X (cuja resposta é T, conhecida) e que forneça a resposta Y, isto é, $Y = 1$ se $XW^T \geq 0$ e $Y = 0$ no caso contrário. Chamaremos de ε o erro cometido pela rede, isto é, $\varepsilon = Y - T$; a cada iteração do treinamento iremos ajustar o vetor de pesos até que ε seje igual a zero. Assim, o vetor de pesos sofrerá um acréscimo ΔW, tal que:

$$\Delta W = \lambda \varepsilon X$$

Onde λ representa a taxa de aprendizagem que é um parâmetro do algoritmo de treinamento que determina o percentual do erro que será considerado para ajustar o vetor de pesos. Quanto menor o valor de λ, mais lentamente o neurônio convergirá para a solução, em contrapartida, a escolha de um λ muito alto, poderá fazer com que o sistema oscile e acabe não convergindo para o valor esperado, se tornando instável.

2.3 Modelo Atual do Perceptron

O modelo original do Perceptron apresentava inúmeras limitações. Por isso, começaram a surgir modelos mais generalizados de neurônio. Nesse trabalho, vamos estudar o modelo utilizado em redes MLP, que consiste de uma versão mais atualizada do Perceptron. Nesse novo Perceptron, os valores de entrada podem ser quaisquer valores reais (não mais restritos a valores booleanos) e a presença do bias é substituída por uma função de propagação $\varphi(x)$, que transforma o valor gerado por XW^T em um intervalo predeterminado. Com isso, a resposta do neurônio passa a ser:

$$Y = \varphi(XW^T)$$

As funções de propagação comumente usadas são: A figura **??** ilustra as funções de propagação comumente utilizadas.

- Função Degrau (figura 1):

$$\varphi(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

- Função Linear (figura 2):

$$\varphi(x) = \begin{cases} 1, & x \geq \frac{1}{2} \\ x, & -\frac{1}{2} \leq x < \frac{1}{2} \\ 0, & x < -\frac{1}{2} \end{cases}$$

- Função Sigmóide (figura 3):

$\varphi(x) = \frac{1}{1+e^{-x}}$

- Função Hiperbólica (figura 4):

$\varphi(x) = \tanh(x)$

2.4 Demonstração do Treinamento do Perceptron

Com a generalização do modelo do Perceptron fica fácil verificar que a forma de treinamento proposta realmente é eficaz e converge em direção a solução. Para isso, vamos utilizar o modelo geral de um Perceptron com função de propagação linear. O objetivo do treinamento é minimizar o valor de uma função de erro E, que é função dos pesos. Para isso, basta calcularmos o valor de W que faz $\frac{\partial E}{\partial W} = 0$. Entretanto, isso nem sempre é simples de fazer, por isso utilizamos um método iterativo para encontrar uma aproximação do mínimo global do

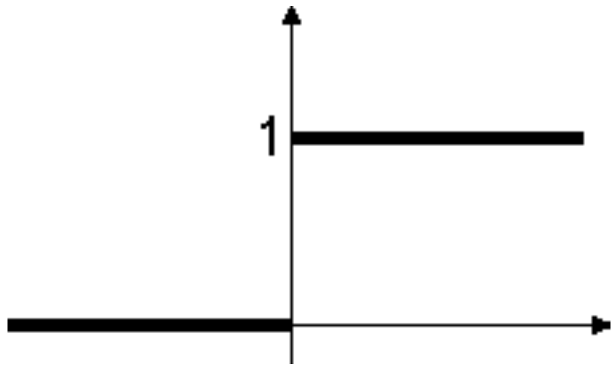

Figura 1: função degrau

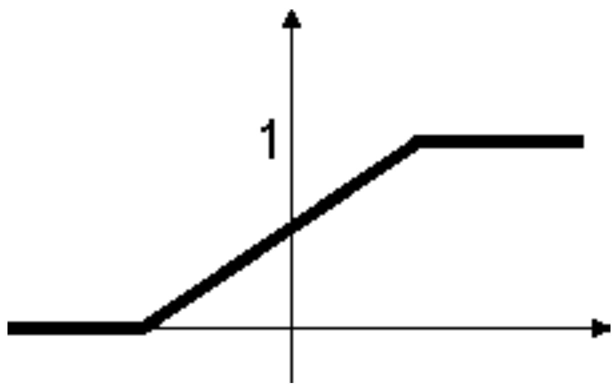

Figura 2: função linear

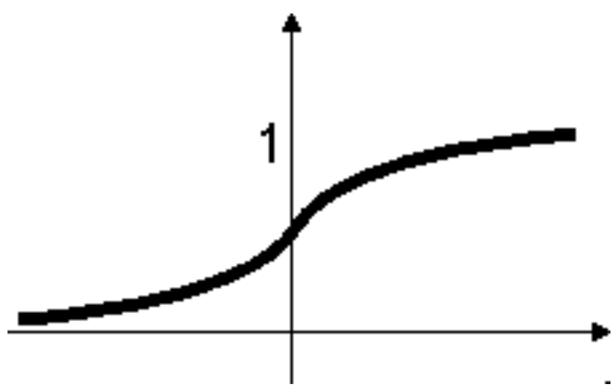

Figura 3: função sigmoide

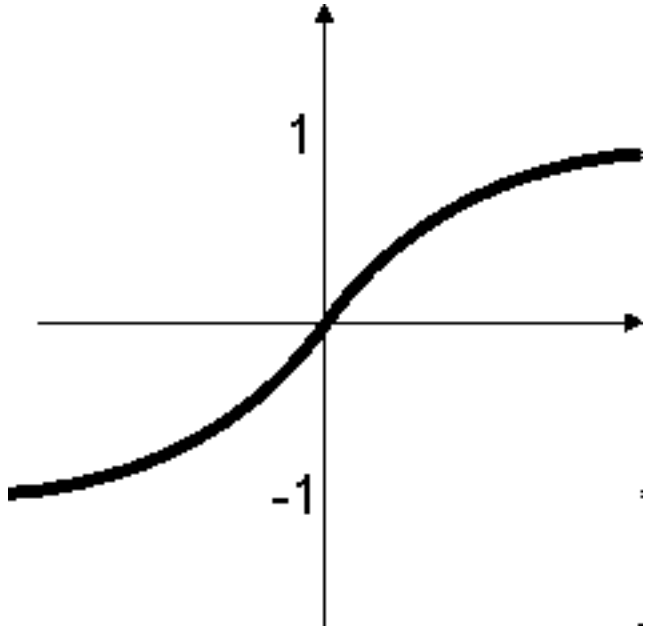

Figura 4: função tangente hiperbólica

erro. Esse método consiste em analisar o valor de $\frac{\partial E}{\partial W}$ para um dado W_0. Se $\frac{\partial E}{\partial W}\big|_{W=W_0} > 0$, então $E(W_0 + dw) > E(W_0)$, logo é preferível pegar um valor menor que W_0, pois diminuiria o erro. Dessa forma, atualizamos o vetor de pesos em $\Delta W = -k\frac{\partial E}{\partial W}$, onde k é uma constante positiva. Vamos considerar que a função de erros utilizadas é $E = \frac{(T-Y)^2}{2}$, portanto:

$$\begin{aligned}\frac{\partial E}{\partial W} &= \frac{\partial}{\partial W}\frac{(T-Y)^2}{2} = \frac{\partial}{\partial W}\frac{\left(T-\varphi(XW^T)\right)^2}{2} \\ &= -\frac{1}{2}2\left(T-\varphi(XW^T)\right)\left(\frac{\partial\varphi(XW^T)}{\partial W}\right)\left(\frac{\partial XW^T}{\partial W}\right)\end{aligned}$$

Sabemos que $\frac{\partial XW^T}{\partial W} = X$ e que $\frac{\partial\varphi(XW^T)}{\partial W}$ é constante, pois estamos usando a função de propagação linear. Portanto podemos concluir que:

$$\frac{\partial E}{\partial W} = -\left(T-Y\right)\left(\frac{\partial\varphi(XW)}{\partial W}\right)X \tag{1}$$

Logo, se fizermos a taxa de aprendizagem $\lambda = k\frac{\partial\varphi(XW)}{\partial W}$, teremos:

$$\Delta W = -k\frac{\partial E}{\partial W} \tag{2}$$

Juntando as equações 10 e 1, chegamos a fórmula final para a atualização dos pesos:

$$\Delta W = \lambda\left(T-Y\right)X = \lambda\varepsilon X$$

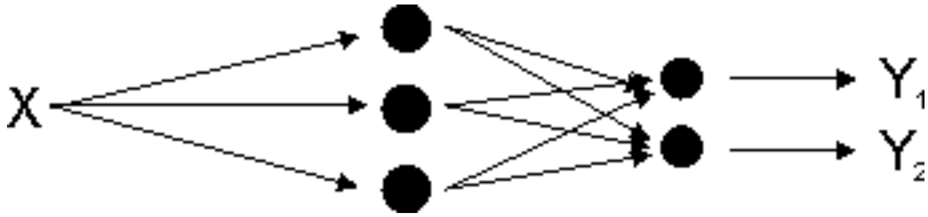

Figura 5: esquema de uma rede com duas camadas

2.5 MLP - Multilayer Perceptron

Entretanto, mesmo com a generalização do modelo do Perceptron, ainda estamos limitados a resolver problemas muito simples. Para aumentar o poder de reconhecimento da rede, se criou um modelo com mais de uma camada de neurônios. Os neurônios da primeira camada recebem o vetor de entradas X e a partir dos pesos W_1 e da função de propagação $\varphi_1(x)$ retornam o vetor de saída Y_1. Por sua vez, os neurônios da segunda camada recebem Y_1 como entrada e a partir dos pesos W_2 e da função de propagação $\varphi_2(x)$ retornam o vetor Y_2, que será passado para a próxima camada e assim sucessivamente até a camada final m que gerará a resposta Y_m da rede (figura 5). Se renomearmos o vetor de entrada X para Y_0, podemos generalizar a fórmula de cálculo da resposta de uma camada i da rede, como:

$$Y_i = \varphi_i\left(Y_{i-1}W_i^T\right)$$

É comum chamarmos a última camada de uma rede MLP de camada de saída e as demais de camadas escondidas. Esses termos serão utilizados nas próximas seções para facilitar o entendimento.

2.6 Backpropagation - Treinamento de MLP

Até agora a nossa tática de treinamento de redes neurais consistiu em atualizar os pesos a partir de $-k\frac{\partial E}{\partial W}$, onde k é uma constante positiva qualquer. Entretanto, somente a última camada de uma rede MLP possui a informação exata do erro. Para contonar isso, foi desenvolvido o algoritmo backpropagation que propaga uma estimativa do erro para as primeiras camadas. Para demonstrar esse algoritmo iremos utilizar uma MLP com m camadas, onde todos os neurônios possuem função de propagação sigmóide, que possui derivada:

$$\begin{aligned}
\frac{\partial\varphi(x)}{\partial x} &= \frac{\partial}{\partial x}\frac{1}{1+e^{-x}} \\
&= -\left(\frac{1}{1+e^{-x}}\right)^2\left(-e^{-x}\right) \\
&= \left(\frac{1}{1+e^{-x}}\right)\left(\frac{e^{-x}}{1+e^{-x}}\right) \\
&= \varphi(x)\left(1-\varphi(x)\right)
\end{aligned}$$

Atualização dos pesos dos neurônios da camada de saída

Para realizar a atualização dos pesos, resta-nos encontrar o valor de $\frac{\partial E}{\partial W_m}$. Iremos considerar o valor Z_i como sendo o resultado do somatório da entrada de um neurônio da camada i com os seus pesos, isto é: $Y_i = \varphi_i(Z_i)$. A partir disso, podemos usar a regra da cadeia para calcular $\frac{\partial E}{\partial W_m}$:

$$\frac{\partial E}{\partial W_m} = \frac{\partial E}{\partial Z_m}\frac{\partial Z_m}{\partial W_m} \tag{3}$$

$$\frac{\partial Z_m}{\partial W_m} = \frac{\partial}{\partial W_m}\left(Y_{m-1}W_m^T\right) = Y_{m-1} \tag{4}$$

Como estamos considerando apenas os neurônios da camada de saída, sabemos o valor do erro, então:

$$\begin{array}{rcl} E & = & \frac{(T-\varphi_m(Z_m))^2}{2} \\ \frac{\partial E}{\partial Z_m} & = & \frac{\partial}{\partial Z_m}\frac{(T-\varphi_m(Z_m))^2}{2} \\ & = & \frac{1}{2}2\left(T-\varphi_m(Z_m)\right)\left(-\frac{\partial\varphi_m(Z_m)}{\partial Z_m}\right) \\ & = & -\left(T-Y_m\right)\left(\frac{\partial\varphi_m(Z_m)}{\partial Z_m}\right) \\ & = & -\varepsilon\varphi_m(Z_m)\left(1-\varphi_m(Z_m)\right) \end{array} \tag{5}$$

Portanto, juntando a equação 10 de atualização dos pesos com as equações 3, 4 e 5, e agregando os valores constantes na taxa de aprendizagem λ, concluímos que:

$$\begin{array}{rcl} \Delta W_m & = & -k\frac{\partial E}{\partial W_m} \\ & = & k\varepsilon\varphi_m(Z_m)\left(1-\varphi_m(Z_m)\right)Y_{m-1} \\ & = & \lambda\varepsilon Y_m\left(1-Y_m\right)Y_{m-1} \end{array}$$

Atualização dos pesos dos neurônios das camadas escondidas

Agora, iremos atualizar os pesos de um neurônio da camada escondida $m-1$. Para isso, utilizaremos um pensamento bastante parecido, entretanto com uma mudança na aplicação da regra da cadeia.

$$\frac{\partial E}{\partial W_{m-1}} = \frac{\partial E}{\partial Z_m}\frac{\partial Z_m}{\partial Y_{m-1}}\frac{\partial Y_{m-1}}{\partial Z_{m-1}}\frac{\partial Z_{m-1}}{\partial W_{m-1}} \tag{6}$$

Vamos calcular separadamente cada um dos fatores dessa equação. O valor de $\frac{\partial E}{\partial Z_m}$ é sabido da equação 5.

$$\frac{\partial E}{\partial Z_m} = \varepsilon\varphi_m(Z_m)\left(1-\varphi_m(Z_m)\right) = \varepsilon Y_m\left(1-Y_m\right)$$

Os demais fatores também são fáceis de calcular:

$$\frac{\partial Z_m}{\partial Y_{m-1}} = \frac{\partial Y_{m-1}W_m^T}{\partial Y_{m-1}} = W_m \tag{7}$$

$$\frac{\partial Y_{m-1}}{\partial Z_{m-1}} = \frac{\partial \varphi_{m-1}(Z_{m-1})}{\partial Z_{m-1}} = Y_{m-1}\left(1 - Y_{m-1}\right) \tag{8}$$

$$\frac{\partial Z_{m-1}}{\partial W_{m-1}} = \frac{\partial Y_{m-2} W_{m-1}^{T}}{\partial W_{m-1}} = Y_{m-2} \tag{9}$$

Portanto, juntando as equações 10, 6, 7, 8 e 9, e juntando os termos constantes na taxa de aprendizagem λ, encontramos a forma de atualização dos pesos dos neurônios da penúltima camada:

$$\Delta W_{m-1} = \lambda \varepsilon Y_m \left(1 - Y_m\right) W_m Y_{m-1} \left(1 - Y_{m-1}\right) Y_{m-2}$$

Se fizermos uma analogia com a forma de atualizar os pesos da camada de saída, teremos:

$$\Delta W_{m-1} = \lambda \varepsilon' Y_{m-1} \left(1 - Y_{m-1}\right) Y_{m-2}$$

Onde $\varepsilon' = \varepsilon Y_m \left(1 - Y_m\right) W_m$ é a estimativa do erro propagado para a camada $m - 1$

Assim, se tomarmos um neurônio qualquer da camada i, podemos generalizar a equação acima e atualizar os pesos fazendo:

$$\Delta W_i = \lambda \varepsilon' Y_i \left(1 - Y_i\right) Y_{i-1}$$

$$\varepsilon' = \varepsilon \prod_{j=i+1}^{m} Y_j \left(1 - Y_j\right) W_j$$

3 Parâmetros de uma Rede

3.1 Conjuntos de Amostras

Para se montar uma rede neural com o objetivo de reconhecer padrões, é necessário colher amostras daquilo que se deseja reconhecer. Essas amostras devem ser separadas em três conjuntos: treinamento, teste e validação.

- Conjunto de treinamento: as amostras desse conjunto são apresentadas a rede neural na fase de treinamento e são usadas para ajustar os pesos dos neurônios. Dentro desse conjunto devem ter amostras de todas as classes que se deseja reconhecer e, preferencialmente, em quantidades iguais, para evitar que o treinamento privilegie uma determinada classe em detrimento das outras. Toda análise estatística a priori, antes de se apresentar a rede neural, deve ser realizada nesse conjunto.

- Conjunto de validação: essas amostras são usadas para indicar o momento de parada do treinamento da rede. Sua utilidade decorre do fato que um treinamento muito extenso faz com que a rede perca seu poder de generalização, isto é, apesar de a rede proporcionar um alto índice de acerto com o conjunto de treinamento, esse índice fica muito aquém do desejado em outras amostras. Nessa situação costumamos dizer que a rede “decorou” o conjunto de treinamento. Para evitar isso, se costuma verificar o acerto da rede para o conjunto de validação após alguns ciclos de treinamento. Esse índice de acertos cresce a medida que o treinamento avança, entretanto, em um certo momento ele começa a decrescer. Nesse ponto, devemos terminar o treinamento, impedindo que a rede perca seu poder de generalização.

- Conjunto de teste: essas amostras são usadas para averiguar a qualidade do reconhecimento. Toda a análise estatística a posteriori (sobre o resultado da rede neural) deve ser feita nesse conjunto.

É evidente que todos esses conjuntos devem ser formados com amostras que representem bem a população real. Entretanto algumas vezes não contamos com amostras suficientes para treinar a rede neural. Nesses casos, apesar de não ser o ideal, pode-se utilizar modelos que geram novas amostras, a partir da aplicação de ruídos nas amostras que dispomos.

No problema de reconhecimento de caracteres manuscritos, havia 28763 amostras de caracteres digitalizadas. Dessas 14400 compuseram o conjunto de treinamento (400 amostras de cada classe), 1800 compuseram o conjunto de validação (50 de cada classe) e o restante (12563) compuseram o conjunto de testes.

3.2 Quantidade de Neurônios e Camadas

Um dos problemas enfrentados por quem utiliza redes neurais é saber quantas camadas e quantos neurônios usar na rede. O uso de poucas camadas faz com que a rede seje incapaz de solucionar o problema. Em contrapartida, o número excessivo de camadas pode inviabilizar o treinamento de uma rede. Isso ocorre, pois a camada de saída é a única que sabe o real valor do erro. A última camada escondida possui apenas uma estimativa do erro, a penúltima camada escondida tem uma estimativa da estimativa do erro e assim sucessivamente. Portanto, quanto mais camadas, maior será a imprecisão da etapa de treinamento.

Com uma única camada, a rede neural é capaz de resolver problemas linearmente separáveis. A adição de uma camada, possibilita a distinção de classes em regiões convexas tão complexas quanto forem o número de neurônios na camada escondida. A inclusão de uma terceira camada permite a formação de regiões arbitrariamente complexas no espaço das amostras. Por isso, normalmente uma única camada escondida é suficiente e, quanto muito, duas e não mais do que isso.

Outra questão importante é decidir quantos neurônios pôr em cada camada. Obviamente a camada de saída possui tantos neurônios quantos forem as dimensões do espaço de saída. O problema consiste em se decidir quantos neurônios por nas camadas escondidas. A utilização de um número excessivo de neurônios, além de retardar o treinamento, faz com que a rede perca seu poder de generalização e acabe "decorando" os dados de treinamento. Isso é um tanto intuitivo, pois a rede possui um número maior de graus de liberdade e fica capaz de "captar" características próprias das amostras do conjunto de treinamento. Por outro lado, um número baixo de neurônios faz com que a rede não consiga convergir para respostas aceitáveis. Em redes com uma única camada escondida existem algumas técnicas usadas para se escolher número de neurônios, dentre as quais destacam-se:

- Utilizar a média aritmética (ou geométrica) entre a dimensão do vetor de entrada e de saída da rede neural;

- Fazer o produto entre o número de neurônios e a dimensão do vetor de saída ser 10 vezes menor que a quantidade de amostras do conjunto de treinamento.

No problema de reconhecimento de caracteres manuscritos, foi construída uma rede neural com apenas uma camada escondida (duas no total). A camada de saída possuía 36 dimensões (uma para cada um dos 36 possíveis caracteres) e a camada escondida possuía 50 neurônios.

3.3 Formas de Treinamento

Quanto ao método de treinamento:

Treinamento supervisionado A forma de treinamento supervisionado é a que até agora foi descrita nesse trabalho. Nela, apresentamos a rede neural amostras das quais sabemos o resultado correto. Fazendo uma analogia, estamos ensinando a rede que padrões devem ser reconhecidos. Essa forma de treinamento é normalmente usada para problemas de classificação.

Treinamento não supervisionado Ao contrário do método supervisionado, no não supervisionado não sabemos o resultado correto que a rede neural deveria retornar no conjunto de treinamento. Normalmente esse método é usada quando queremos descobrir padrões em amostras usando redes neurais artificiais.

Quanto ao ajuste dos pesos:

Treinamento por padrão Nesse método os pesos são ajustados após a apresentação de cada amostra do conjunto de treinamento. Normalmente esse método é mais rápido e requer menos memória, principalmente quando o conjunto de treinamento for grande e reduntante. Em contrapartida, é necessário utilizar uma taxa de aprendizagem pequena, pois cada amostra pode ajustar os pesos em uma direção diferente da superfície de erro. Portanto, nesses casos, uma taxa de aprendizagem grande pode tornar a rede instável. Além disso, a ordem com que as amostras são apresentadas a rede influenciam na velocidade do treinamento.

Treinamento em batch Nessa abordagem todas as amostras são mostradas a rede e os pesos são ajustados com a média dos erros obtidos. Essa forma de treinamento é mais estável e não é influenciada pela ordem em que as amostras são apresentadas a rede. Entretanto, requer mais memória e normalmente é mais lenta que o treinamento por padrão.

3.4 Parâmetros de Treinamento

Taxa de Aprendizagem

A taxa de aprendizagem representa o percentual do erro que será usado para ajustar os pesos. A escolha de uma taxa de aprendizagem muito pequena faz com que o treinamento seje muito lento. Entretanto uma taxa de aprendizagem alta faz com que a rede tenha dificuldades para convergir para o valor esperado, pois causa muitas oscilações no treinamento. O ideal é ter uma taxa de aprendizagem adaptativa que diminua ao longo do treinamento, que foi a abordagem utilizada para o problema de reconhecimento de caracteres manuscritos.

Termo de Momentum

Em regiões próximas a pontos de mínimo (locais ou globais), $\frac{\partial E}{\partial W}$ é próximo de zero. Portanto quanto mais próximo de um ponto de mínimo, menor será o ajuste dos pesos. Para evitar que o treinamento se torne muito lento e instável, adiciona-se o termo de momentum na equação de ajuste de pesos. Assim os pesos serão ajustados mediante a seguinte equação $\Delta W_i = \lambda f(\varepsilon, X) + \alpha \Delta W_{i-1}$, onde α é o termo de momentum.

Assim se em dois passos consecutivos o sinal de $\frac{\partial E}{\partial W}$ forem iguais o termo de momentum acelera o passo na superfície de erro. Entretanto, quando alterar o sinal de $\frac{\partial E}{\partial W}$, o termo de momentum diminui o ajuste dos pesos para evitar instabilidade no treinamento, pois nesse momento o treinamento acabou de ultrapassar um ponto de mínimo na superfície de erro.

Inicialização dos Pesos

Preferencialmente os pesos devem ser iniciados randomicamente em uma distribuição uniforme e próximo de zero. Observando a função de propagação (por exemplo, sigmóide) percebemos que abaixo de um certo valor a função fica muito próxima de zero, e acima de um outro valor, muito próxima de um. Por isso, valores altos dos pesos fazem com que a resposta do primeiro ciclo de treinamento da rede seje próximo de um. Os pesos seriam então ajustados por um ΔW pequeno e novamente a rede nos forneceria um valor próximo de um. Isso faria com que a rede repetisse a mesma etapa de treinamento diversas vezes, até que o valor dos pesos caísse de forma a proporcionar uma mudança significativa na resposta da rede. De forma análoga aconteceria se inicializássemos os pesos com valores muito baixo. Por isso os pesos são inicializados com valores randômicos e próximos de zero.

4 Pré e Pós processamento dos Dados

4.1 Winner Takes All

No problema de reconhecimento de N classes, são necessários N neurônios na camada de saída. Se cada neurônio estiver usando a função de propagação sigmóide, a sua saída estará no intervalo $(0, 1)$. A resposta ideal da rede neural ao reconhecer uma classe i, seria ter a i-ésima ordernada igual a um e todas as demais iguais a zero. O que a abordagem *Winner Takes All* faz é justamente igualar em um a maior ordenada do vetor de saída da rede e colocar todas as demais ordenadas em zero. O *Winner Takes All* é somente um procedimento que facilita o cálculo da matriz de confusão e do percentual de acerto da rede.

4.2 Normalização da Entrada

Muitas vezes usamos características de magnitudes diferentes para apresentar a rede neural. Por exemplo, no problema do paciente médico (descrito no capítulo 1), temos a informação da idade, pressão arterial e taxa de glicose no sangue do paciente. Entretanto, é claro que esses valores são muito distintos e podem, por isso, causar problemas na eficiência da rede. Para se resolver esses problemas, usa-se um estimador de média $\widetilde{\mu}$ e desvio padrão $\widetilde{\sigma}$ para cada uma das características das amostras (ordenadas do vetor de entrada). Esses estimadores são usados para deixar todas as características igualmente distribuídas (em uma t-Student com tantos graus de liberdade quantos forem as amostras do conjunto de treinamento). Com isso, todas as características possuirão média zero, desvio padrão $\frac{N}{N-2}$ e a mesma magnitude. Chamamos esse processo de normalização porque a distribuição t-Student se aproxima da distribuição normal, quando o número de graus de liberdade cresce.

4.3 Matriz de Confusão

É uma ferramenta usada para medir em quais classes a rede neural está com maior grau de confusão. Consiste basicamente de uma matriz NxN, onde a posição (i, j) representa o número de amostras da classe j que a rede neural reconheceu como i. Dada uma matriz Y de resposta (após a aplicação do *Winner Takes All*) para k amostras e uma outra matriz T, com a classificação ideal dessas amostras, a matriz de confusão será YT^T.

É fácil observar que na diagonal da matriz de confusão estarão representadas as amostras acertadas pela rede. Um valor muito alto para um dado valor (i, j), com $i \neq j$, da matriz de confusão indica que a rede está fazendo muita confusão quando tenta reconhecer o padrão j, normalmente trocando por i.

A figura 6 representa um resumo da matriz de confusão obtida com o problema de reconhecimento de caracteres manuscritos. Foi destacado da matriz de confusão original apenas os casos que apresentaram maior problema. Observando a matriz, podemos destacar que a letra O e o dígito zero tinham um alto grau de confusão, que pode ser facilmente explicado por eles possuírem o

	Dígitos				Letras			
	0	1	2	5	I	O	S	Z
0	235	0	0	0	0	242	0	0
1	0	181	4	0	14	0	0	0
2	0	3	387	0	8	0	0	67
5	0	0	0	211	2	0	188	0
I	0	165	0	0	243	0	0	0
O	116	1	0	0	0	165	0	0
S	0	2	0	69	1	0	182	0
Z	0	0	46	0	8	0	0	451
Total	385	416	492	299	332	463	388	557

Figura 6: matriz de confusão resumida para o problema de reconhecimento de caracteres manuscritos.

mesmo símbolo (numa outra abordagem, essas classes foram reunidas em uma única classe). Além disso, as letras I, S e Z confundiam, respectivamente, com os dígitos um, cinco e dois. Em alguns casos o número de confusões era maior que o número de acertos para essas classes.

4.4 Estimador de centros de classes

Outra análise importante é verificar quão dispersas estão as classes que queremos reconhecer. Para isso, usaremos os estimadores de média e desvio padrão para as amostras. Para calcular os estimadores de média e desvio padrão de uma dada classe C_k faremos, para todas as N_k amostras dessa classe no conjunto de treinamento, os seguintes cálculos (Suponha $X^{(i)}$ o vetor de características da amostra i).

- Estimador da média:
$$\widehat{\mu} = \frac{1}{N_k} \sum_{i \in C_k} X^{(i)}$$

- Estimador do desvio padrão:
$$\widehat{\sigma} = \frac{1}{N_k - 1} \sum_{i \in C_k} \left(X^{(i)} - \widehat{\mu} \right)^2$$

$\widehat{\mu}$ e $\widehat{\sigma}$ são vetores que representam, respectivamente, o centro de cada classe e a variância dessas classes em torno de seu centro. Uma das análises possíveis para verificar a dispersão entre as classes é calcular a distância entre os vetores $\widehat{\mu}$ de cada classe e analisá-los com os seus estimadores de desvio padrão. Com isso, podemos identificar as classes mais difíceis de serem reconhecidas e procurar outras abordagens para fazer.

A figuras 8 e 7 mostram, respectivamente, as classes do problema de reconhecimento de padrões que apresentam o maior e o menor vetor desvio padrão

Classe	$\|\widehat{\sigma}\|$
B	71,4913
P	72,0197
F	72,5803
zero	72,6019
O	72,6642

Figura 7: classes menos dispersas

Classe	$\|\widehat{\sigma}\|$
2	93,3293
R	89,0912
1	88,4283
N	87,4772
M	87,1319

Figura 8: classes mais dispersas

estimado ($\widehat{\sigma}$). Isso reflete o grau de dispersão dessas classes em torno de sua média. Já a figura 9 destaca a distância entre os centro de algumas classes. É importante observar que as distâncias destacadas são menores que o vetor desvio padrão, o que já era esperado devido ao número elevado de dimensões do vetor de entrada. Entretanto a distância entre os centros dessas classes também representa valores muito baixo se comparados aos valores das ordenadas do vetor desvio padrão. Isso mostra que essas classes possuem uma interseção grande e evidencia o motivo de a matriz de confusão ter apontado um alto grau de erro.

4.5 Discriminante de Fischer

O objetivo principal do discriminante de Fischer é realizar uma diminuição na dimensão do vetor de entradas através de uma transformação que permita uma maior separação entre as classes. Para mostrar o funcionamento desse método, vou supor um vetor de entrada (X) de duas dimensões e estudar o discriminante de Fischer que transforma X em um vetor de uma dimensão (X'):

$$X' = \omega^T X$$

Classe 1	Classe 2	Distância
O	zero	9,6
S	5	32,4
Z	2	32,0
I	1	36,0

Figura 9: mais próximos centros de classes

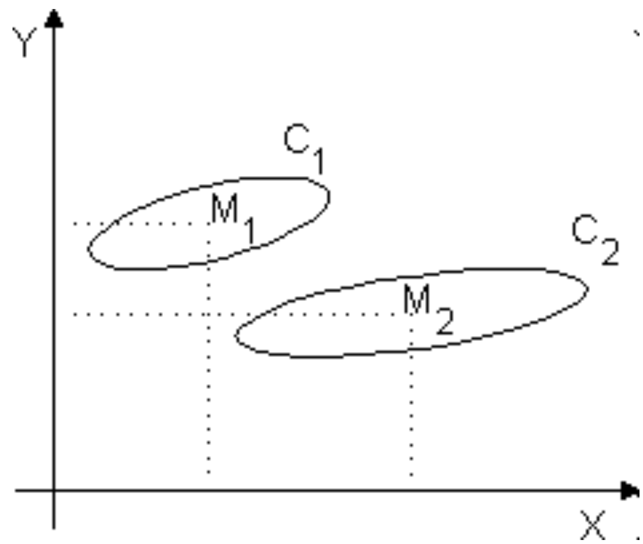

Figura 10: problema de projetar no vetor de maior separação das médias

Para realizar essa análise vamos tomar a média M_1 e M_2 de cada uma dessas classes:

$$M_1 = \frac{1}{N_1} \sum_{i \in C_1} X^{(i)}$$

$$M_2 = \frac{1}{N_2} \sum_{i \in C_2} X^{(i)}$$

Como queremos a transformação que aumenta a separação entre as classes, podemos pensar em usar a transformação que maximiza a distância entre as médias das classes:

$$(M_2' - M_1') = \omega^T (M_2 - M_1)$$

Entretanto nem sempre a distância entre as médias representa o grau de separação entre as classes. Na figura 10 vemos que as médias estarão mais separadas quando projetadas na ordenada X, entretanto as classes serão melhor discriminadas se projetadas na ordenada Y.

O problema consiste em analisar como é o espalhamento de cada classe em torno de sua média. Para isso, calcularemos a soma dos quadrados dos desvios de cada amostra em relação a sua média, isto é, para cada classe j, teríamos:

$$S_j^2 = \sum_{i \in C_j} \left(X'^{(i)} - M_j' \right)^2$$

A partir disso, chegamos a uma equação conhecida como critério de Fischer:

$$J(\omega) = \frac{\left(M_2' - M_1'\right)^2}{S_1^2 + S_2^2}$$

Se tomarmos a matriz de covariância entre as classes (S_B) e a soma das matrizes de covariância interna de cada classe (S_W), podemos reescrever $J(\omega)$ como sendo:

$$S_B = (M_2 - M_1)(M_2 - M_1)^T$$

$$S_W = \sum_{i \in C_1} \left(X^{(i)} - M_1\right)\left(X^{(i)} - M_1\right)^T + \sum_{i \in C_2} \left(X^{(i)} - M_2\right)\left(X^{(i)} - M_2\right)^T$$

$$J(\omega) = \frac{\omega^T S_B \omega}{\omega^T S_W \omega}$$

Para encontrar a melhor projeção segundo esse critério, temos, portanto, que achar o valor de ω que maximiza $J(\omega)$. Para isso, vamos calcular $\frac{\partial J}{\partial \omega}$ e igualar a zero.

$$\frac{\partial J}{\partial \omega} = \frac{\left(\omega^T S_B \omega\right) S_W \omega - \left(\omega^T S_W \omega\right) S_B \omega}{\left(\omega^T S_W \omega\right)^2} = 0$$

Com isso, podemos concluir que:

$$\left(\omega^T S_B \omega\right) S_W \omega = \left(\omega^T S_W \omega\right) S_B \omega \tag{10}$$

Estamos interessados em verificar somente a direção do vetor ω, pois determinará a reta que maximiza a discriminação entre as variáveis segundo o critério de Fischer. Se eliminarmos todos os termos constantes de 10 e observando que $S_B \omega$ possui a mesma direção de $M_2 - M_1$, podemos concluir que:

$$\omega \alpha S_W^{-1} (M_2 - M_1)$$

Esse é o discriminante linear de Fischer, que na verdade é apenas uma direção para projeção e não um discriminante propriamente dito.

4.6 PCA - Principal Component Analisis

A análise de componentes principais consiste em uma redução na dimensão do vetor de entrada. É claro que toda a redução causa uma perda de informação, por outro lado permite criar uma rede mais enxuta e coesa, pois algumas informações trazem ruídos que não ajudam no reconhecimento. Para entender o PCA vamos estudar, para um conjunto de N amostras, a redução de um vetor de entrada X com d dimensões para d_r dimensões, onde $d > d_r$. Podemos escrever o vetor X como uma combinação linear de d vetores ortonormais (u_1, u_2, ..., u_d) da seguinte forma:

$$X = \sum_{i=1}^{d} \omega_i u_i$$

Como $d > d_r$ vamos evidenciar um subconjunto de d_r dimensões no vetor X, fazendo:

$$X = \sum_{i=1}^{d_r} \omega_i u_i + \sum_{i=d_r+1}^{d} \omega_i u_i$$

Entretanto, para haver a redução da dimensão do vetor X, os coeficientes ω_i para $d_r < i \leq d$ devem ser descartados e serão substituídos por constantes b_i, tal que:

$$\widetilde{X} = \sum_{i=1}^{d_r} \omega_i u_i + \sum_{i=d_r+1}^{d} b_i u_i$$

O objetivo do PCA é reduzir a dimensão da entrada de forma a perder o mínimo possível, isto é, minimizar $X - \widetilde{X}$, onde:

$$X - \widetilde{X} = \sum_{i=d_r+1}^{d} (\omega_i - b_i) u_i$$

Pode-se verificar que essa equação é minimizada quando escolhemos os vetores correspondentes aos i menores autovalores da matriz de covariância interna de X. Portanto, o método do PCA consiste em se calcular a matriz de covariância do vetor de entrada X. Em seguida se escolhe os d_r autovetores correspondente aos maiores autovalores dessa matriz. Por fim, fazemos a projeção de X nesses autovetores e obtemos o novo vetor de entrada com d_r dimensões. Essa análise não utiliza o vetor T, com as respostas esperadas para o vetor X. Portanto, poderíamos utilizar esse método como uma estratégia de treinamento não supervisionado.

No problema de reconhecimento de caracteres manuscritos o vetor de entrada possuía 235 dimensões. Foi realizada a aplicação do PCA até o vetor de entradas atingir 97 dimensões. Além de acelerar brutalmente o treinamento da rede, houve ainda uma melhora de 7,8% em relação ao resultado que vinha sendo obtido com o vetor de entrada original.

5 Conclusão

Podemos concluir com esse trabalho que a utilização de redes neurais artificiais para auxiliar a resolução de problemas de reconhecimento de padrão é eficaz, entretanto é necessária a utilização de outras ferramentas para melhor analisar o problema. Uma pré-análise dos dados é essencial para avaliarmos a complexidade do problema e verificarmos se uma rede neural pode ou não trazer bons resultados. Além disso, um pré-processamento pode ser vital para auxiliar a eliminar ruídos nos dados de entrada e, com isso, facilitar a tarefa da rede neural. Uma pós-análise se faz necessária quando a rede não consegue atingir os índices esperados. É através dela que podemos verificar em que classes estão ocorrendo os problemas e, com isso, utilizar uma abordagem diferente para resolver esses problemas.

Outro fator importante são os métodos utilizados para a obtenção de características dos dados de entrada. Sabemos que a rede neural recebe um vetor númerico como entrada, entretanto muitas vezes a entrada de nosso sistema é uma imagem (ou som) digitalizado. Nesses casos, é preciso utilizar métodos eficientes para a extração de características de forma a permitir que a rede neural possua informações em quantidade e qualidade suficientes para realizar o reconhecimento.

É evidente que outros métodos de cunho estatístico podem ser utilizados para solucionar problemas de reconhecimento de padrões, tais como: teste de hipóteses e métodos de regressão. Entretanto, com um número grande de dimensões esses métodos se tornam mais difíceis de serem usados. Enquanto isso, as redes neurais são treinadas a partir de um método iterativo, o que facilita a resolução de problemas mais complexos.

Outro quesito importante ao se trabalhar com redes neurais é avaliar o contexto em que estamos trabalhando. No problema de reconhecimento de caracteres manuscritos, se sabemos que uma rede neural está confundindo as letras U e V, e ao longo do reconhecimento identificou os caracteres S, I, L, U e A, apontando uma grande imprecisão no U; podemos afirmar que a probabilidade de o U ter sido reconhecido no lugar do V é muito alta, o que nos permite, através de métodos estatísticos, identificar e corrigir o erro da rede.

Podemos também pensar em mesclar conceitos de lógica fuzzy com as redes neurais, isto é, ao invés de a rede determinar uma classe de resposta, ela retornaria o grau de pertinência de uma amostra em cada uma das classes. Isso, potencializaria o nosso sistema, pois permite uma análise diferenciada para cada classe e, consequentemente, possibilitaria a criação de modelos de confusão das classes.

Por fim, podemos concluir que no campo de reconhecimento de padrões ainda há muito o que se estudar e pesquisar, dado as inúmeras aplicações e áreas envolvidas nessa linha.

Referências

[Bishop] C. M. Bishop, Neural Network for Pattern Recognition, Oxford University, 1995.

[BCL] A. P. Braga, A. P. L. F. Carvalho, T. B. Ludemir, Fundamentos de Redes Neurais Artificiais, XI Escola de Computação, 1998.

[Jantzen] J. Jantzen, Introduction do Perceptron Networks, Technical University of Denmark, 1998.

[Vianna] G. K. Vianna, Sistemas Inteligentes de Arquitetura Híbrida Neuro-Difusa para Reconhecimento de Padrões, NCE/UFRJ (tese de mestrado), 2000.

[Silva] E. Silva, Reconhecimento Inteligente de Caracteres Manuscritos, Instituto Militar de Engenharia - IME (tese de mestrado), 2002.

[DeGroot] M. H. DeGroot, Probability and Statistics, Addison-Wesley Publishing Company, 1987.