Nomes: Fabio Moreira (14200727)
Marcello da Silva Klingelfus Junior (13100764)

INE5430 - Inteligência Artificial

Relatório Trabalho Prático 5 - A identificação de Dígitos Manuscritos

1 Introdução

Nas seções abaixo, cada parte do código será isolada e explicada em relação à sua função no script.

2 Transformação da saída

A primeira parte envolve transformar a saída que é numérica (1, 2, 3…10) em uma matriz numérica, onde na 1ª linha dela estão assinaladas com o valor 1 todas as entradas (o conjunto de entradas que forma um dígito, nesse caso) que são o dígito 1, na 2ª linha as entradas que são o dígito 2, e assim sucessivamente. O trecho de código abaixo realiza essa operação:

```
for i=1:size(X,2)
  SaidaTrein(y(1,i),i)=1;
end
```

3 Normalização

O enunciado do trabalho supõe que os números foram previamente normalizados, logo não há necessidade de normalizá-los. Mesmo assim, adicionamos um trecho que normaliza as entradas para uso futuro em trabalhos com entradas não normalizadas. Por curiosidade, testamos nas entradas desse trabalho e o resultado não sofreu alterações substanciais. A normalização é feita com o método mapminmax que normaliza os valores de uma matriz para o intervalo [-1 1].

O código de normalização é:

```
[X_N,PS]=mapminmax(X);
```

4 Arquitetura da rede

O código para a criação da rede neural é o que segue:

```
net=newff(X,SaidaTrein,50,{'tansig','purelin'},'traingdx');
```

Onde:

X -> Entradas da rede (400 por dígito)

SaidaTrein -> matriz numérica que “diz” a rede neural qual dígito aquele conjunto de entradas representa

50 -> Números de neurônios na camada intermediária. Após inúmeros testes, esse valor foi considerado bom o suficiente, seus resultados sempre são estáveis.

tansig -> função de ativação dos neurônios da camada intermediária.

purelin -> função de ativação dos neurônios de saída. Escolhido pois o domínio dele não é restritivo em comparação com as outras funções.

traingdx -> Algoritmo de treinamento backpropagation utilizando gradiente descendente com momento e taxa adaptativa. Mostrou-se o melhor e mais rápido algoritmo.

5 Treinamento

Para separar os dados de treinamento, teste, utilizou-se o padrão de métodos de validação como o cross-validation, onde 90% dos dados vão para treinamento e 10% para teste (conhecido como k-fold com k=10, onde 9 partições vão para treinamento e a que restou para teste). Em números concretos, para treinamento foram utilizados 4500 dados e 500 para teste.

```
net.divideParam.trainRatio=0.9;
net.divideParam.testRatio=0.1;
net.divideParam.valRatio=0.1;
```

Além desses atributos a época foi fixada como 5000, ou seja, o algoritmo irá parar após 5000 épocas. O valor para o *validation check* foi aumentado para 50. Foi necessário aumentar pois o algoritmo de treinamento *traingdx* varia muito de um momento para outro, o que estava afetando o valor de *validation check* padrão. Abaixo, as linhas com essas alterações e o método de treinamento.

```
net.trainParam.epochs=5000;
net.trainParam.max_fail = 50;
net=train(net,X,SaidaTrein);
```

6 Resultados

O método para simular a rede é o que segue:

```
Y=sim(net,X);
```

Feito isso, é necessário separar os valores encontrados em uma matriz numérica da mesma forma como foi realizado na saída conhecida das entradas. Ele procura, para aquela entrada, a maior nota recebida em relação à todas as saídas do problema e atribui o valor 1, no atributo YClass, a classe com o maior valor. Na última linha, a matriz de confusão é gerada.

```
for i=1:size(X,2)
    c = max([Y(1,i),Y(2,i),Y(3,i),Y(4,i),Y(5,i),Y(6,i),Y(7,i),Y(8,i),Y(9,i),Y(10,i)]);
   for j=1:10
            if Y(j,i)== c
            YClass(j,i)=1;
            end
   end
end

plotconfusion(SaidaTrein,YClass);
```

Utilizando o script detalhado nas seções anteriores, o resultado do treinamento e teste da rede neural se encontra abaixo. A taxa de acertos se manteve estável em 95%, variando de 97% a 94%. O arquivo com a matriz de confusão salva em *.fig* foi enviado junto com o arquivo para melhor visualizar o resultado.