

Trabalho 5: A Identificação de Objetos no RX do Aeroporto

1 Introdução

Uma prática muito comum para o reconhecimento de padrões é a utilização de uma rede neural para classificar um conjunto de padrões de entrada. A partir dessa rede já treinada é possível classificar padrões nunca visto anteriormente. Assim o objetivo do trabalho proposto é treinar e testar uma rede neural que classifica peças do vestuário detectadas por um RX do aeroporto. Para isso será utilizado conjuntos de teste, um para treinar e testar a rede e o outro para testar. Mais detalhes da implementação serão explicados na seção de Desenvolvimento.

2 Desenvolvimento

Nessa seção, serão realizadas explicações do código utilizado e também apresentados diagramas gerados a partir da ferramenta *MatLab*.

2.1 Carregar os módulos e o Conjunto de dados

Para essa etapa do trabalho foram disponibilizados dois arquivos *.m* (padrão do *MatLab*). O primeiro possui o código de uma função, chamada *importFile.m*. Essa função importa os dados dos *.csv* disponibilizados e transforma esses dados numéricos do arquivo de texto em uma matriz. O outro possui o nome de *prepData.m* e utiliza o *importFile* para preparar os dados e importar para o ambiente do *MatLab*.

2.2 Normalização dos dados

Segundo a documentação do *MatLab*, a função *newff* normaliza os dados antes da criação da rede neural. Mas para entender como é realizada a normalização de dados será utilizado a função *mapminmax* antes do treinamento. Essa função pode ser útil para dimensionar as entradas e os destinos para que sempre caiam dentro de um intervalo especificado. A função *mapminmax* dimensiona entradas e destinos para que eles caiam no intervalo $[-1,1]$.

O código de normalização é:

```
[Data_train,PS] = mapminmax(input');  
Target_train=TARGET';  
Data_test=mapminmax('apply',input_test',PS);  
Target_test=TARGET_TEST';
```

2.3 Separação do conjunto de treinamento e de teste

Para esse trabalho foi disponibilizado pelo professor, dois arquivos de dados: *fashion_mnist_test.csv* e *fashion_mnist_train.csv*. Um contém os 60000 dados que são divididos entre teste e validação, sendo 90% e 10% respectivamente. E o outro possui os dados de treinamento.

O código de separação do conjunto de dados é apresentado a seguir:

```
net.divideParam.trainRatio = 0.9;
net.divideParam.valRatio = 0.1;
net.divideParam.testRatio = 0.0;
```

Como foi disponibilizado um arquivo de dados separado para o teste então não se faz necessário alocar dados de *fashion-mnist_train.csv* para essa atividade.

2.4 Arquitetura e experimentos realizados

Nesta subseção serão apresentados aspectos da arquitetura, dos experimentos, a realização do treinamento e a sua performance, taxa de acerto da classificação e a geração da matriz de confusão. Para demonstrar o domínio do trabalho, foram realizados alguns testes com diferentes configuração. Esses testes utilizam as funções *newff()* (cria a rede neural conforme alguns parâmetros), *train()* (treina a rede neural) e *sim()* (faz a simulação da rede neural) Quatro desses testes serão explicados abaixo.

Teste 1

Essa rede neural é configurada com a camada de entrada com 784 neurônios, duas camadas intermediárias, a primeira usando 30 neurônios e a outra 15, e a camada de saída com 10 neurônios.

O algoritmo de treinamento utilizado foi o RProp (*trainrp*) que é uma função de treinamento de rede que atualiza valores de ponderação e polarização de acordo com o algoritmo resiliente de retropropagação. Foi configurado para tentar alcançar um erro de 0.000001 e uma taxa de aprendizado de 0.05.

```
net = newff (Data_train,Target_train,[30 15], 'logsig' 'logsig'
            'purelin', 'trainrp');
```

onde os parâmetros são:

- **Data_train:** é o conjunto de dados para treinamento;
- **Target_train:** matriz numérica contendo o dígito que o conjunto de entradas representa;
- **[30 15]:** número de neurônios nas camadas intermediárias;
- **'logsig' e 'logsig':** funções log-simóide de ativação dos neurônios das camadas intermediárias;
- **'purelin':** função de ativação dos neurônios de saída, calcula a saída de uma camada a partir de sua entrada;
- **'trainrp':** algoritmo de treinamento explicado anteriormente.

Em seguida, foi realizado o treinamento da rede usando as funções:

```
net=train(net,Data_train,Target_train);
```

$$Y = \text{sim}(\text{net}, \text{Data_train});$$

A primeira treina a rede neural utilizando os parâmetros passados em *net*. Um desses parâmetros é o número de épocas de treinamento, no qual foram definidos como 500. Enquanto a segunda faz a simulação da rede. A Figura 1 mostra as configurações e o treinamento da rede. Já na Figura 2 é mostrado a performace da rede utilizando os dados de validação e teste.

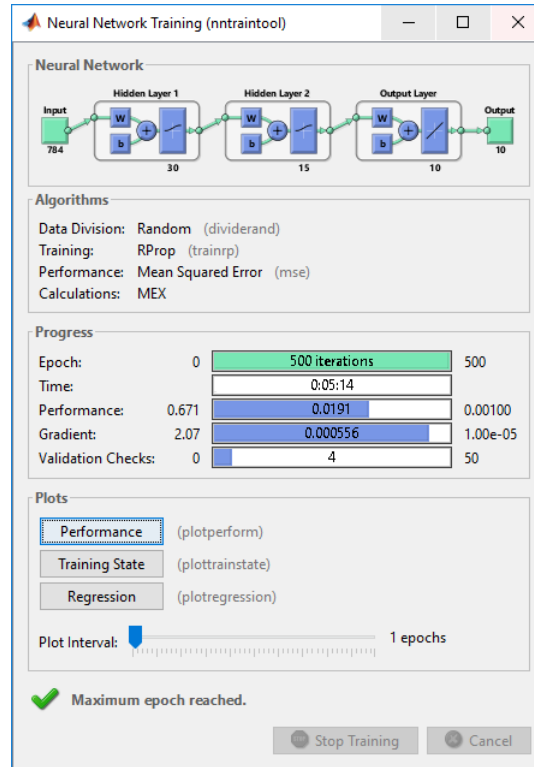


Figura 1: Configurações da rede neural.

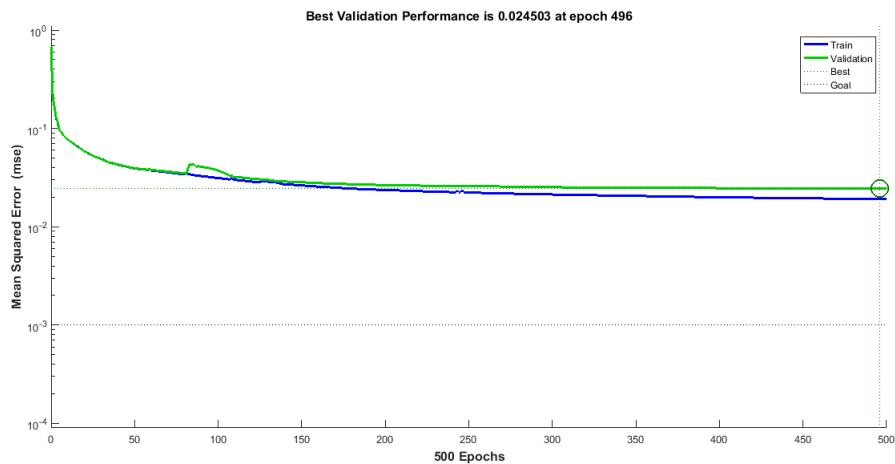


Figura 2: Performace da rede neural.

Após o treinamento é necessário separar os valores obtidos em uma matriz numérica. Assim, é procurado, para cada determinada entrada, a maior nota recebida em relação a todas as saídas

Confusion Matrix												
Output Class	1	5130 8.6%	14 0.0%	42 0.1%	215 0.4%	15 0.0%	0 0.0%	857 1.4%	0 0.0%	13 0.0%	0 0.0%	81.6% 18.4%
	2	6 0.0%	5778 9.6%	4 0.0%	16 0.0%	6 0.0%	0 0.0%	8 0.0%	0 0.0%	4 0.0%	1 0.0%	99.2% 0.8%
	3	78 0.1%	27 0.0%	4663 7.8%	75 0.1%	524 0.9%	1 0.0%	526 0.9%	0 0.0%	53 0.1%	0 0.0%	78.4% 21.6%
	4	149 0.2%	106 0.2%	43 0.1%	5193 8.7%	139 0.2%	0 0.0%	123 0.2%	0 0.0%	33 0.1%	2 0.0%	89.7% 10.3%
	5	21 0.0%	15 0.0%	622 1.0%	249 0.4%	4860 8.1%	1 0.0%	467 0.8%	0 0.0%	26 0.0%	0 0.0%	77.6% 22.4%
	6	13 0.0%	3 0.0%	5 0.0%	2 0.0%	1 0.0%	5732 9.6%	1 0.0%	136 0.2%	7 0.0%	47 0.1%	96.4% 3.6%
	7	547 0.9%	48 0.1%	609 1.0%	239 0.4%	439 0.7%	10 0.0%	3956 6.6%	0 0.0%	130 0.2%	2 0.0%	66.2% 33.8%
	8	6 0.0%	3 0.0%	3 0.0%	6 0.0%	0 0.0%	187 0.3%	1 0.0%	5741 9.6%	19 0.3%	202 0.3%	93.1% 6.9%
	9	49 0.1%	5 0.0%	9 0.0%	5 0.0%	16 0.0%	26 0.0%	60 0.1%	17 0.0%	5713 9.5%	2 0.0%	96.8% 3.2%
	10	1 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	39 0.1%	1 0.0%	106 0.2%	2 0.0%	5744 9.6%	97.5% 2.5%
		85.5% 14.5%	96.3% 3.7%	77.7% 22.3%	86.6% 13.4%	81.0% 19.0%	95.5% 4.5%	65.9% 34.1%	95.7% 4.3%	95.2% 4.8%	87.5% 12.5%	
		1	2	3	4	5	6	7	8	9	10	

Confusion Matrix												
Output Class	1	796 8.0%	4 0.0%	10 0.1%	40 0.4%	2 0.0%	1 0.0%	175 1.8%	0 0.0%	2 0.0%	0 0.0%	77.3% 22.7%
	2	3 0.0%	966 9.7%	1 0.0%	13 0.1%	2 0.0%	1 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	97.9% 2.1%
	3	13 0.1%	3 0.0%	739 7.4%	19 0.2%	91 0.9%	0 0.0%	95 0.9%	0 0.0%	13 0.1%	0 0.0%	76.0% 24.0%
	4	42 0.4%	13 0.1%	6 0.1%	852 8.5%	20 0.2%	2 0.0%	24 0.2%	0 0.0%	7 0.1%	0 0.0%	88.2% 11.8%
	5	5 0.1%	2 0.0%	103 1.0%	40 0.4%	791 7.9%	1 0.0%	92 0.9%	0 0.0%	7 0.1%	0 0.0%	76.0% 24.0%
	6	2 0.0%	0 0.0%	1 0.0%	1 0.0%	910 9.1%	1 0.0%	43 0.4%	3 0.0%	18 0.2%	0 0.0%	92.9% 7.1%
	7	122 1.2%	12 0.1%	134 1.3%	34 0.3%	88 0.8%	2 0.0%	600 6.0%	0 0.0%	19 0.2%	0 0.0%	59.3% 40.7%
	8	1 0.0%	0 0.0%	0 0.0%	0 0.0%	52 0.5%	0 0.0%	914 9.1%	6 0.1%	37 0.4%	0 0.0%	90.5% 9.5%
	9	15 0.1%	0 0.0%	6 0.1%	1 0.0%	5 0.1%	10 0.1%	13 0.1%	0 0.0%	940 9.4%	2 0.0%	94.7% 5.3%
	10	1 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	21 0.2%	0 0.0%	42 0.4%	2 0.0%	943 9.4%	93.5% 6.5%
		79.6% 20.4%	96.6% 3.4%	73.9% 26.1%	85.2% 14.8%	79.1% 20.9%	91.0% 9.0%	60.0% 40.0%	91.4% 8.6%	94.0% 6.0%	94.3% 5.7%	84.5% 15.5%
		1	2	3	4	5	6	7	8	9	10	

Figura 4: Matriz de confusão com os dados de teste.

Essa rede neural é configurada com a camada de entrada com 784 neurônios, duas camadas intermediárias, a primeira usando 17 neurônios e a outra 11, e a camada de saída com 10 neurônios.

```
net = newff (Data_train,Target_train,[17 11], 'logsig' 'logsig'  
            'purelin','trainrp');
```

- **Data_train**: é o conjunto de dados para treinamento;
- **Target_train**: matriz numérica contendo o dígito que o conjunto de entradas representa;
- **[17 11]**: número de neurônios nas camadas intermediárias;
- **'logsig'** e **'logsig'**: funções de ativação dos neurônios das camadas intermediárias;
- **'purelin'**: função de ativação dos neurônios de saída, calcula a saída de uma camada a partir de sua entrada;
- **'trainrp'**: algoritmo de treinamento explicado anteriormente.

Em seguida, foi realizado o treinamento da rede usando as funções:

```
net=train(net,Data_train,Target_train);
```

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

A primeira treina a rede neural utilizando os parâmetros passados em *net*. Um desses parâmetros é o número de épocas de treinamento, no qual foram definidos como 500. Enquanto a segunda faz a simulação da rede. A Figura 5 mostra as configurações e o treinamento da rede. Já na Figura 6 é mostrado a performance da rede utilizando os dados de validação e teste.

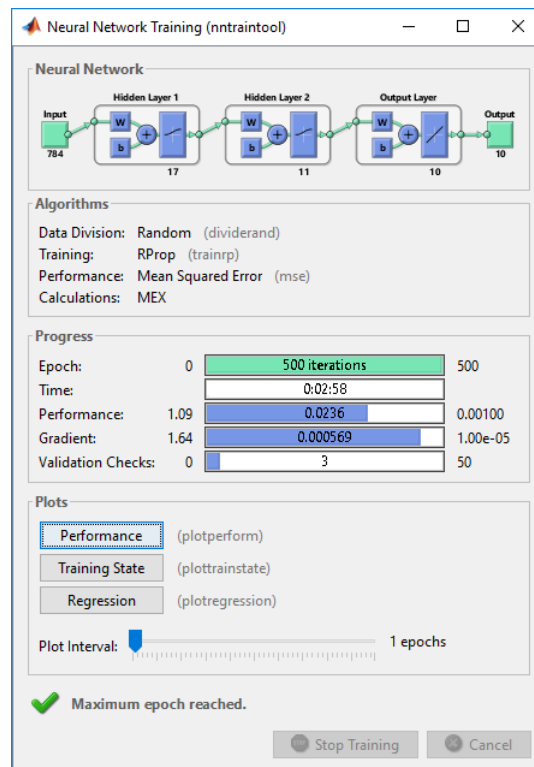


Figura 5: Configurações da rede neural.

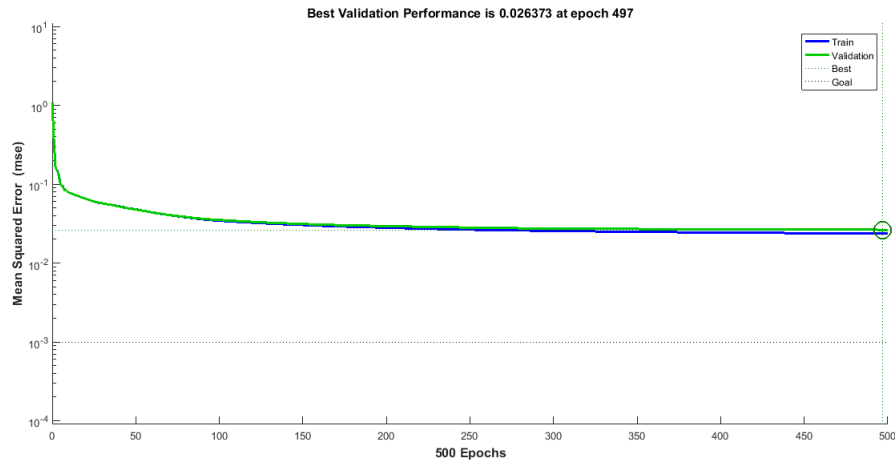


Figura 6: Performace da rede neural.

Após o treinamento é necessário separar os valores obtidos em uma matriz numérica. Assim, é procurado, para cada determinada entrada, a maior nota recebida em relação a todas as saídas do problema e atribui valor 1 a classe com maior valor (código pode ser visualizado em *rede.m*). Depois a matriz de confusão é gerada. Abaixo são apresentadas a matriz de confusão para os dados de treinamento (com 84,7% de acerto) e de teste (com 82,3% de acerto).

Confusion Matrix										
Output Class	1	2	3	4	5	6	7	8	9	10
1	4879 8.1%	28 0.0%	68 0.1%	295 0.5%	54 0.1%	3 0.0%	817 1.4%	0 0.0%	29 0.0%	1 0.0%
2	5 0.0%	5623 9.4%	3 0.0%	41 0.1%	6 0.0%	3 0.0%	4 0.0%	0 0.0%	16 0.0%	0 0.0%
3	42 0.1%	60 0.1%	4206 7.0%	60 0.1%	386 0.6%	0 0.0%	553 0.9%	1 0.0%	21 0.0%	0 0.0%
4	233 0.4%	169 0.3%	47 0.1%	5067 8.4%	211 0.4%	13 0.0%	132 0.2%	0 0.0%	26 0.0%	8 0.0%
5	33 0.1%	34 0.1%	916 1.5%	130 0.2%	4696 7.8%	0 0.0%	575 1.0%	0 0.0%	17 0.0%	0 0.0%
6	9 0.0%	0 0.0%	7 0.0%	3 0.0%	4 0.0%	5641 9.4%	2 0.0%	205 0.3%	23 0.0%	126 0.2%
7	754 1.3%	58 0.1%	724 1.2%	381 0.6%	585 1.0%	6 0.0%	3857 6.4%	0 0.0%	276 0.5%	0 0.0%
8	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	154 0.3%	0 0.0%	5589 9.3%	15 0.0%	176 0.3%
9	42 0.1%	25 0.0%	28 0.0%	15 0.0%	54 0.1%	34 0.1%	56 0.1%	9 0.0%	5554 9.3%	2 0.0%
10	3 0.0%	3 0.0%	1 0.0%	8 0.0%	4 0.0%	146 0.2%	4 0.0%	196 0.3%	23 0.0%	5687 9.5%
	81.3% 18.7%	93.7% 6.3%	70.1% 29.9%	84.5% 15.5%	78.3% 21.7%	94.0% 6.0%	64.3% 35.7%	93.2% 6.9%	92.6% 7.4%	94.8% 5.2%
Target Class	1	2	3	4	5	6	7	8	9	10

Figura 7: Matriz de confusão com os dados de treino.

Confusion Matrix										
Output Class	1	2	3	4	5	6	7	8	9	10
1	773 7.7%	2 0.0%	12 0.1%	50 0.5%	8 0.1%	2 0.0%	167 1.7%	0 0.0%	7 0.1%	0 0.0%
2	1 0.0%	935 9.3%	0 0.0%	11 0.1%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	5 0.1%	0 0.0%
3	9 0.1%	17 0.2%	668 6.7%	9 0.1%	64 0.6%	0 0.0%	85 0.9%	0 0.0%	5 0.1%	0 0.0%
4	53 0.5%	31 0.3%	13 0.1%	838 8.4%	32 0.3%	0 0.0%	32 0.3%	0 0.0%	4 0.0%	0 0.0%
5	6 0.1%	3 0.0%	173 1.7%	24 0.2%	789 7.9%	0 0.0%	89 0.9%	0 0.0%	8 0.1%	0 0.0%
6	3 0.0%	1 0.0%	0 0.0%	1 0.0%	0 0.0%	888 8.9%	1 0.0%	49 0.5%	7 0.1%	23 0.2%
7	144 1.4%	9 0.1%	126 1.3%	59 0.6%	94 0.9%	0 0.0%	610 6.1%	0 0.0%	45 0.4%	0 0.0%
8	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	59 0.6%	0 0.0%	886 8.9%	3 0.0%	34 0.3%
9	11 0.1%	2 0.0%	8 0.1%	3 0.0%	12 0.1%	11 0.1%	14 0.1%	1 0.0%	902 9.0%	0 0.0%
10	0 0.0%	0 0.0%	0 0.0%	5 0.1%	1 0.0%	40 0.4%	1 0.0%	64 0.6%	14 0.1%	943 9.4%
	77.3% 22.7%	93.5% 6.5%	66.8% 33.2%	83.8% 16.2%	78.9% 21.1%	88.8% 11.2%	61.0% 39.0%	88.6% 11.4%	90.2% 9.8%	94.3% 5.7%
Target Class	1	2	3	4	5	6	7	8	9	10

Figura 8: Matriz de confusão com os dados de teste.

Teste 3

Essa rede neural é configurada com a camada de entrada com 784 neurônios, duas camadas intermediárias, a primeira usando 17 neurônios e a outra 11, e a camada de saída com 10 neurônios.

O algoritmo de treinamento utilizado foi o RProp (trainrp) que é uma função de treinamento de rede que atualiza valores de ponderação e polarização de acordo com o algoritmo resiliente de retropropagação. Foi configurado para tentar alcançar um erro de 0.000001 e uma taxa de aprendizado de 0.05.

```
net = newff (Data_train,Target_train,[17 11],'transig' 'transig'
            'purelin','trainrp');
```

onde os parâmetros são:

- **Data_train:** é o conjunto de dados para treinamento;
- **Target_train:** matriz numérica contendo o dígito que o conjunto de entradas representa;
- **[17 11]:** número de neurônios nas camadas intermediárias;
- **'transig' e 'transig':** funções tangente-simóide de ativação dos neurônios das camadas intermediárias;
- **'purelin':** função de ativação dos neurônios de saída, calcula a saída de uma camada a partir de sua entrada;
- **'trainrp':** algoritmo de treinamento explicado anteriormente.

Em seguida, foi realizado o treinamento da rede usando as funções:

```
net=train(net,Data_train,Target_train);

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

A primeira treina a rede neural utilizando os parâmetros passados em *net*. Um desses parâmetros é o número de épocas de treinamento, no qual foram definidos como 500. Enquanto a segunda faz a simulação da rede. Já na Figura 9 é mostrado a performance da rede utilizando os dados de validação e teste.

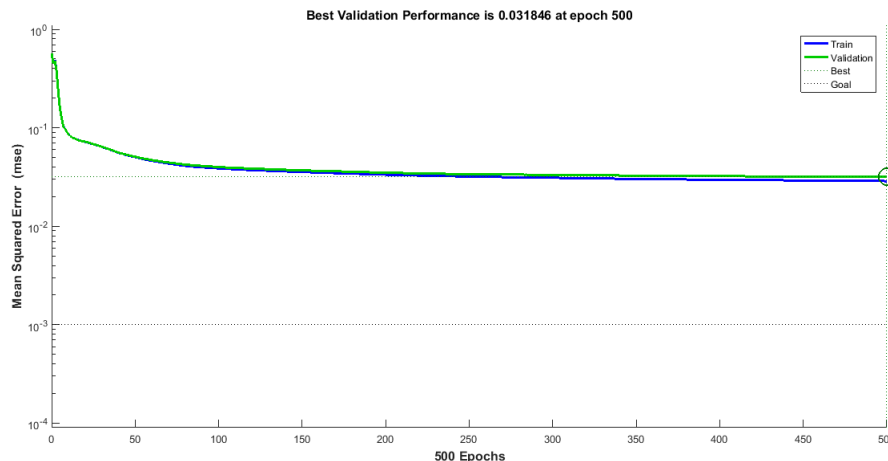


Figura 9: Performance da rede neural.

Após o treinamento é necessário separar os valores obtidos em uma matriz numérica. Assim, é procurado, para cada determinada entrada, a maior nota recebida em relação a todas as saídas do problema e atribui valor 1 a classe com maior valor (código pode ser visualizado em *rede.m*). Depois a matriz de confusão é gerada. Abaixo são apresentadas a matriz de confusão para os dados de treinamento (com 81,9% de acerto) e de teste (com 80,3% de acerto).

Confusion Matrix										
Output Class	1	2	3	4	5	6	7	8	9	10
1	4676 7.8%	12 0.0%	129 0.2%	91 0.2%	19 0.0%	2 0.0%	1056 1.8%	0 0.0%	36 0.1%	1 0.0%
2	21 0.0%	5727 9.5%	10 0.0%	46 0.1%	8 0.0%	4 0.0%	15 0.0%	0 0.0%	5 0.0%	2 1.9%
3	84 0.1%	31 0.1%	3520 5.9%	25 0.0%	180 0.3%	1 0.0%	750 1.3%	0 0.0%	25 0.0%	0 23.7%
4	691 1.2%	150 0.3%	95 0.2%	5246 8.7%	138 0.2%	1 0.0%	575 1.0%	0 0.0%	40 0.1%	0 24.4%
5	42 0.1%	25 0.0%	1366 2.3%	132 0.2%	4613 7.7%	1 0.0%	547 0.9%	0 0.0%	48 0.1%	0 31.9%
6	20 0.0%	1 0.0%	7 0.0%	1 0.0%	2 0.0%	5553 9.3%	23 0.0%	131 0.2%	36 0.1%	110 0.2%
7	281 0.5%	44 0.1%	823 1.4%	407 0.7%	996 1.7%	1 0.0%	2812 4.7%	0 0.0%	115 0.2%	3 0.0%
8	0 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	268 0.4%	0 0.0%	5692 9.5%	21 0.0%	227 0.4%
9	182 0.3%	8 0.0%	46 0.1%	49 0.1%	44 0.1%	18 0.0%	221 0.4%	15 0.0%	5668 9.4%	4 0.0%
10	3 0.0%	2 0.0%	3 0.0%	3 0.0%	0 0.0%	151 0.3%	1 0.0%	162 0.3%	6 0.0%	5653 9.4%
	77.9% 22.1%	95.5% 4.5%	58.7% 41.3%	87.4% 12.6%	76.9% 23.1%	92.5% 7.5%	46.9% 53.1%	94.9% 5.1%	94.5% 5.5%	81.9% 18.1%
Target Class	1	2	3	4	5	6	7	8	9	10

Figura 10: Matriz de confusão com os dados de treino.

Confusion Matrix										
Output Class	1	2	3	4	5	6	7	8	9	10
1	754 7.5%	3 0.0%	23 0.2%	15 0.1%	1 0.0%	3 0.0%	181 1.8%	0 0.0%	4 0.0%	0 23.4%
2	6 0.1%	951 9.5%	1 0.0%	15 0.1%	1 0.0%	0 0.0%	3 0.0%	0 0.0%	1 0.0%	1 97.1%
3	23 0.2%	10 0.1%	594 5.9%	5 0.1%	26 0.3%	0 0.0%	110 1.1%	0 0.0%	8 0.1%	0 23.5%
4	129 1.3%	21 0.2%	18 0.2%	854 8.5%	19 0.2%	0 0.0%	97 1.0%	0 0.0%	6 0.1%	0 25.3%
5	4 0.0%	5 0.1%	222 2.2%	29 0.3%	774 7.7%	1 0.0%	87 0.9%	0 0.0%	9 0.1%	0 31.6%
6	3 0.0%	1 0.0%	2 0.0%	0 0.0%	0 0.0%	875 8.8%	4 0.0%	44 0.4%	13 0.1%	22 9.2%
7	45 0.4%	9 0.1%	121 1.2%	65 0.7%	173 1.7%	0 0.0%	474 4.7%	0 0.0%	25 0.3%	1 48.1%
8	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	74 0.7%	0 0.0%	908 9.1%	4 0.0%	58 13.0%
9	36 0.4%	0 0.0%	19 0.2%	17 0.2%	6 0.1%	10 0.1%	44 0.4%	1 0.0%	927 9.3%	0 12.5%
10	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	37 0.4%	0 0.0%	47 0.5%	3 0.0%	918 9.2%
	75.4% 24.6%	95.1% 4.9%	59.4% 40.6%	85.4% 14.6%	77.4% 22.6%	87.5% 12.5%	47.4% 52.6%	90.8% 9.2%	92.7% 7.3%	80.3% 19.7%
Target Class	1	2	3	4	5	6	7	8	9	10

Figura 11: Matriz de confusão com os dados de teste.

Teste 4

Essa rede neural é configurada com a camada de entrada com 784 neurônios, uma camada intermediárias, usando 1 neurônios, e a camada de saída com 10 neurônios.

O algoritmo de treinamento utilizado foi o RProp (trainrp) que é uma função de treinamento de rede que atualiza valores de ponderação e polarização de acordo com o algoritmo resiliente de retropropagação. Foi configurado para tentar alcançar um erro de 0.000001 e uma taxa de aprendizado de 0.05.

```
net = newff (Data_train,Target_train,[1], 'tansig'
            'purelin', 'trainrp');
```

onde os parâmetros são:

- **Data_train:** é o conjunto de dados para treinamento;
- **Target_train:** matriz numérica contendo o dígito que o conjunto de entradas representa;
- **[1]:** número de neurônios nas camadas intermediárias;
- **'tansig':** função tangente-simóide de ativação dos neurônios das camadas intermediárias;
- **'purelin':** função de ativação dos neurônios de saída, calcula a saída de uma camada a partir de sua entrada;
- **'trainrp':** algoritmo de treinamento explicado anteriormente.

Em seguida, foi realizado o treinamento da rede usando as funções:

```
net=train(net,Data_train,Target_train);
```


$$Y = \text{sim}(\text{net}, \text{Data_train});$$

A primeira treina a rede neural utilizando os parâmetros passados em *net*. Um desses parâmetros é o número de épocas de treinamento, no qual foram definidos como 500. Enquanto a segunda faz a simulação da rede. A Figura 12 mostra as configurações e o treinamento da rede. Já na Figura 13 é mostrado a performace da rede utilizando os dados de validação e teste.

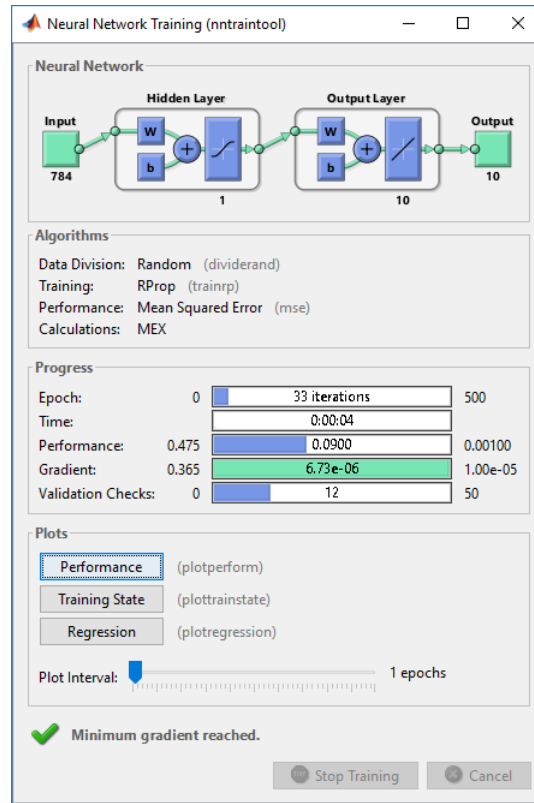


Figura 12: Configurações da rede neural.

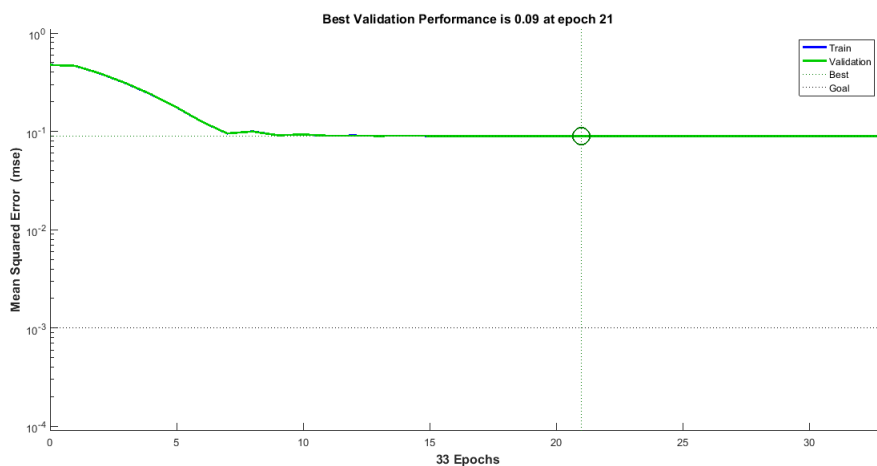


Figura 13: Performace da rede neural.

Após o treinamento é necessário separar os valores obtidos em uma matriz numérica. Assim,

é procurado, para cada determinada entrada, a maior nota recebida em relação a todas as saídas do problema e atribui valor 1 a classe com maior valor (código pode ser visualizado em *rede.m*). Depois a matriz de confusão é gerada. Abaixo são apresentadas a matriz de confusão para os dados de treinamento (com 10% de acerto) e de teste (com 10% de acerto).

Confusion Matrix

	1	2	3	4	5	6	7	8	9	10	
1	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
2	5999 10.0%	6000 10.0%	6000 10.0%	6000 10.0%	6000 10.0%	6000 10.0%	6000 10.0%	5999 10.0%	6000 10.0%	10.0%	NaN%
3	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
4	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
5	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
6	1 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	NaN%
7	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
8	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
9	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
10	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
	0.0%	100%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	10.0%
	100%	0.0%	100%	100%	100%	100%	100%	100%	100%	100%	90.0%
	1	2	3	4	5	6	7	8	9	10	

Figura 14: Matriz de confusão com os dados de treino.

Confusion Matrix

	1	2	3	4	5	6	7	8	9	10	
1	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
2	1000 10.0%	1000 10.0%	1000 10.0%	1000 10.0%	1000 10.0%	1000 10.0%	1000 10.0%	1000 10.0%	999 10.0%	1000 10.0%	NaN%
3	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
4	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
5	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
6	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	NaN%
7	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
8	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
9	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
10	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
	0.0%	100%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	10.0%
	100%	0.0%	100%	100%	100%	100%	100%	100%	100%	100%	90.0%
	1	2	3	4	5	6	7	8	9	10	

Figura 15: Matriz de confusão com os dados de teste.

Teste 5

Essa rede neural é configurada com a camada de entrada com 784 neurônios, uma camada intermediária, 10 neurônios, e a camada de saída com 10 neurônios.

O algoritmo de treinamento utilizado foi o RProp (trainrp) que é uma função de treinamento de rede que atualiza valores de ponderação e polarização de acordo com o algoritmo resiliente de retropropagação. Foi configurado para tentar alcançar um erro de 0.000001 e uma taxa de aprendizado de 0.05.

```
net = newff (Data_train,Target_train,[10], 'tansig'
            'purelin', 'trainrp');
```

onde os parâmetros são:

- **Data_train:** é o conjunto de dados para treinamento;
- **Target_train:** matriz numérica contendo o dígito que o conjunto de entradas representa;
- **[10]:** número de neurônios nas camada intermediárias;
- **'tansig':** função tangente-simóide de ativação dos neurônios das camadas intermediárias;
- **'purelin':** função de ativação dos neurônios de saída, calcula a saída de uma camada a partir de sua entrada;
- **'trainrp':** algoritmo de treinamento explicado anteriormente.

Em seguida, foi realizado o treinamento da rede usando as funções:

```
net=train(net,Data_train,Target_train);
```

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

A primeira treina a rede neural utilizando os parâmetros passados em *net*. Um desses parâmetros é o número de épocas de treinamento, no qual foram definidos como 500. Enquanto a segunda faz a simulação da rede. A Figura 16 mostra as configurações e o treinamento da rede. Já na Figura 17 é mostrado a performance da rede utilizando os dados de validação e teste.

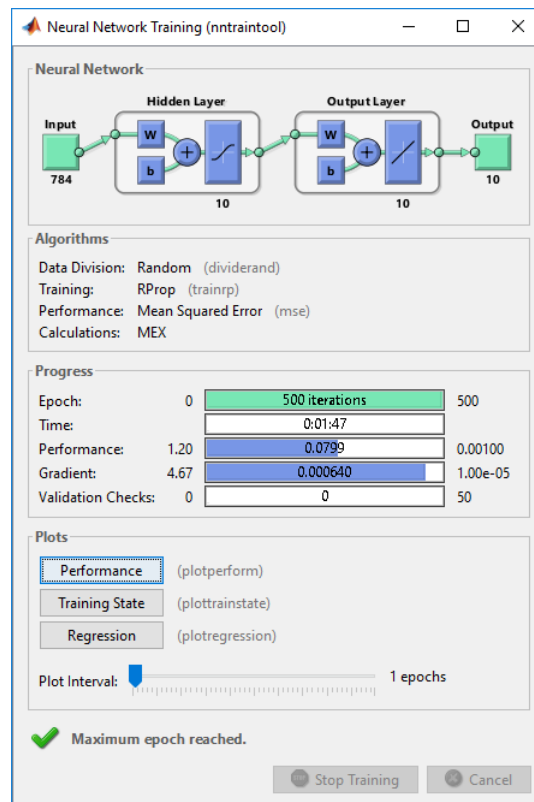


Figura 16: Configurações da rede neural.

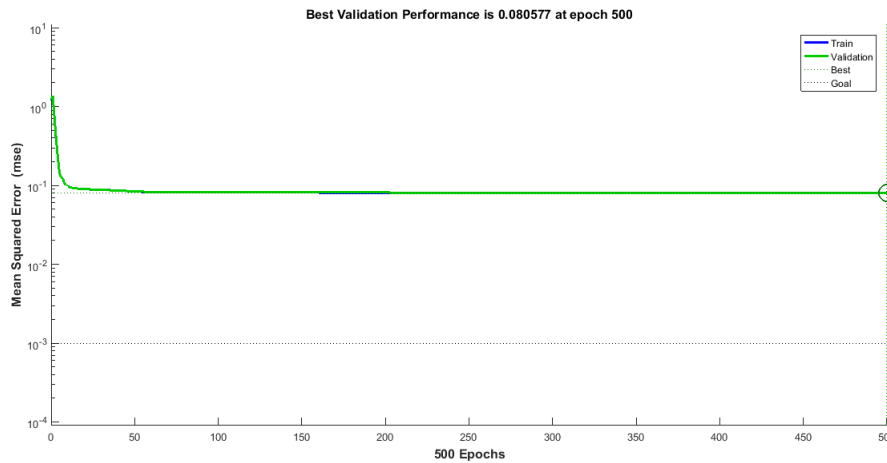


Figura 17: Performace da rede neural.

Após o treinamento é necessário separar os valores obtidos em uma matriz numérica. Assim, é procurado, para cada determinada entrada, a maior nota recebida em relação a todas as saídas do problema e atribui valor 1 a classe com maior valor (código pode ser visualizado em *rede.m*). Depois a matriz de confusão é gerada. Abaixo são apresentadas a matriz de confusão para os dados de treinamento (com 27,5% de acerto) e de teste (com 24,1% de acerto).

Confusion Matrix										
Output Class	1	2	3	4	5	6	7	8	9	10
1	3272 5.5%	12 0.0%	150 0.3%	161 0.3%	27 0.0%	0 0.0%	819 1.4%	0 0.0%	361 0.6%	1 0.0%
2	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	7 0.1%	0 0.0%
3	61 0.1%	40 0.1%	2411 4.0%	36 0.1%	1229 2.0%	0 0.0%	1101 1.8%	0 0.0%	55 0.1%	0 0.0%
4	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
5	149 0.2%	92 0.2%	3089 5.1%	427 0.7%	4473 7.5%	4 0.0%	2664 4.4%	0 0.0%	120 0.2%	0 0.0%
6	3 0.0%	0 0.0%	1 0.0%	10 0.0%	1 0.0%	0 0.0%	1 0.0%	0 0.1%	35 0.1%	0 0.0%
7	39 0.1%	0 0.0%	63 0.1%	12 0.0%	11 0.0%	0 0.0%	181 0.3%	0 0.0%	2 0.0%	0 0.0%
8	50 0.1%	2 0.0%	1 0.0%	15 0.0%	0 0.0%	0 0.0%	21 0.0%	0 0.0%	1 0.0%	0 0.0%
9	12 0.0%	1 0.0%	8 0.0%	21 0.0%	1 0.0%	0 0.0%	10 0.0%	192 0.3%	0 0.0%	0 0.0%
10	2414 4.0%	5853 9.8%	277 0.5%	5318 8.9%	258 0.4%	5996 10.0%	1203 2.0%	6000 10.0%	5227 8.7%	5999 10.0%
Target Class	1	2	3	4	5	6	7	8	9	10

Figura 18: Matriz de confusão com os dados de treino.

Confusion Matrix										
Output Class	1	2	3	4	5	6	7	8	9	10
1	543 5.4%	0 0.0%	34 0.3%	22 0.2%	1 0.0%	0 0.0%	137 1.4%	0 0.0%	52 0.5%	0 0.0%
2	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
3	25 0.3%	6 0.1%	384 3.6%	7 0.1%	228 2.3%	0 0.0%	165 1.7%	0 0.0%	14 0.1%	0 0.0%
4	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN%
5	38 0.4%	23 0.2%	512 5.1%	72 0.7%	729 7.3%	2 0.0%	439 4.4%	0 0.0%	23 0.2%	0 0.0%
6	0 0.0%	0 0.0%	1 0.0%	2 0.0%	0 0.0%	0 0.0%	3 0.0%	0 0.0%	5 0.1%	0 0.0%
7	8 0.1%	1 0.0%	16 0.2%	2 0.0%	5 0.1%	0 0.0%	23 0.2%	0 0.0%	1 0.0%	0 0.0%
8	11 0.1%	0 0.0%	0 0.0%	3 0.0%	0 0.0%	0 0.0%	4 0.0%	0 0.0%	1 0.0%	0 0.0%
9	1 0.0%	0 0.0%	6 0.1%	3 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	34 0.3%	0 0.0%
10	374 3.7%	970 9.7%	47 0.5%	889 8.9%	37 0.4%	998 10.0%	228 2.3%	1000 10.0%	870 8.7%	1000 10.0%
Target Class	1	2	3	4	5	6	7	8	9	10

Figura 19: Matriz de confusão com os dados de teste.

3 Conclusão

O desenvolvimento desse trabalho fez com que os membros da equipe aprenderem sobre o treinamento da rede neural. Foram executados vários testes além desses mostrados acima. Foi observado que quando usado redes com apenas uma camada intermediária a taxa de acerto no aprendizado é muito pequena (Testes 4 e 5). As redes neurais que possuem duas camadas de neurônios apresentam uma taxa de acerto boa (Testes 1, 2 e 3). Para melhorar a taxa de acerto poderia ter colocado mais épocas de treinamento, mas como a taxa de acerto usando duas

camadas foi satisfatória então não colocamos mais épocas. Vários erros na detecção de objetos foram encontrados, como no Teste 2, a rede detectou quase todas as peças como 2, assim por falta de neurônios, a rede neural não conseguiu aprender que eram outros tipos de peça.