Trabalho 09

Nome da Atividade: MLP para reconhecimento de dígitos

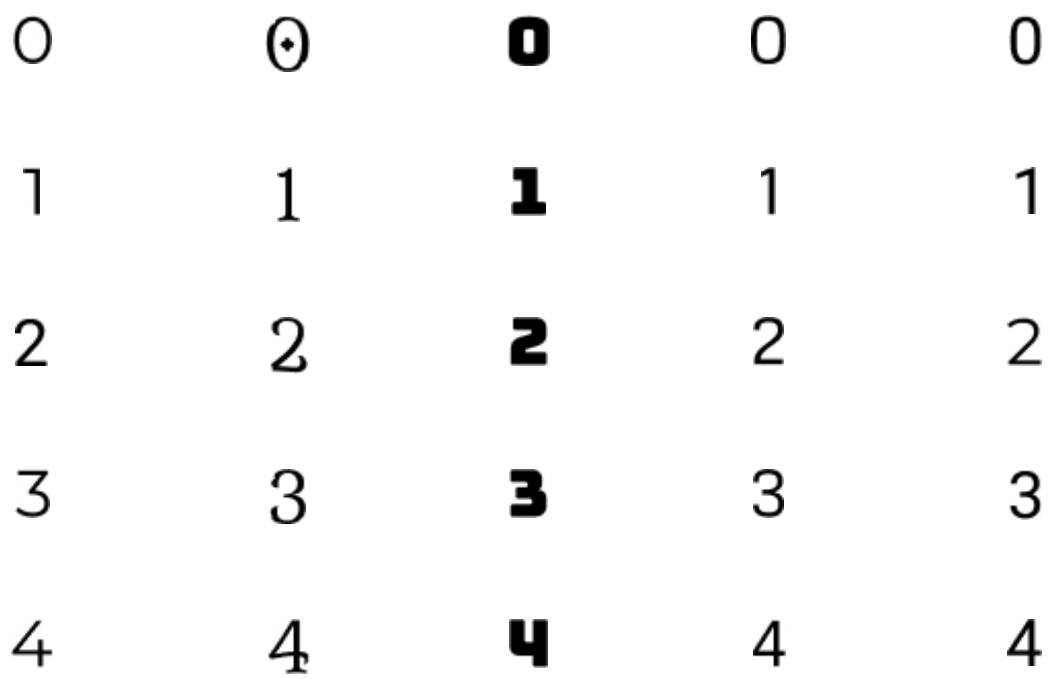
Nome e Matrícula: Lucas Lima do Nascimento - 12111ECP024

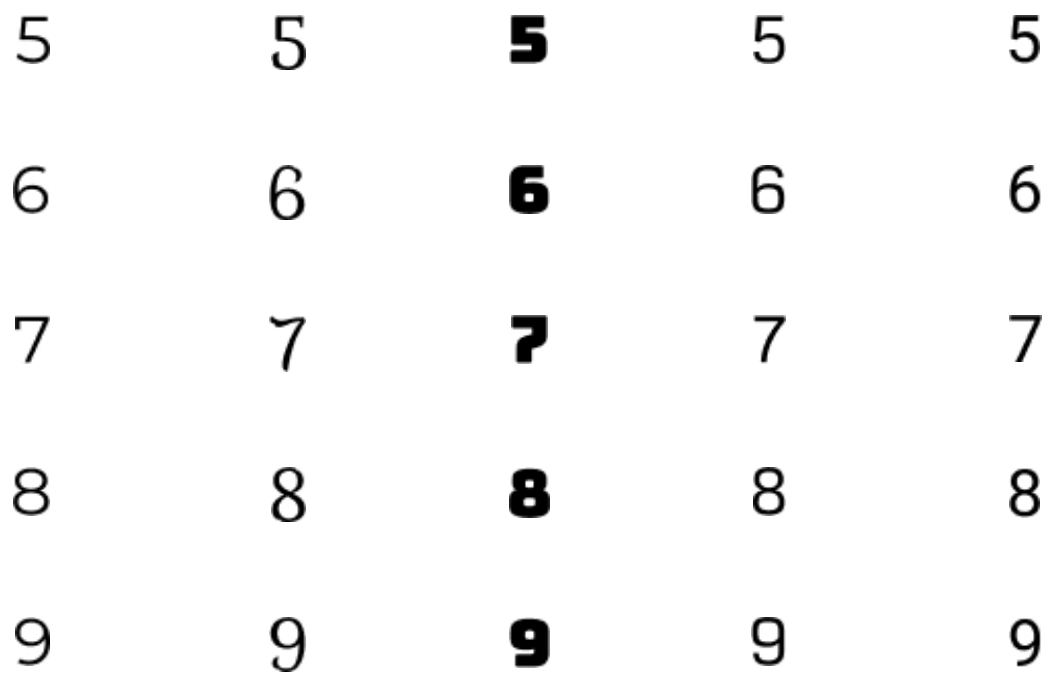
Base de dados:

Inicialmente, peguei as imagens referentes as 5 fontes escolhidas. Elas foram:

- Roboto
- Montserrat
- Metrophobic
- Milonga
- Bungee Spice

Gerei um arquivo para cada uma delas e tentei gerá-los em tamanhos similares:





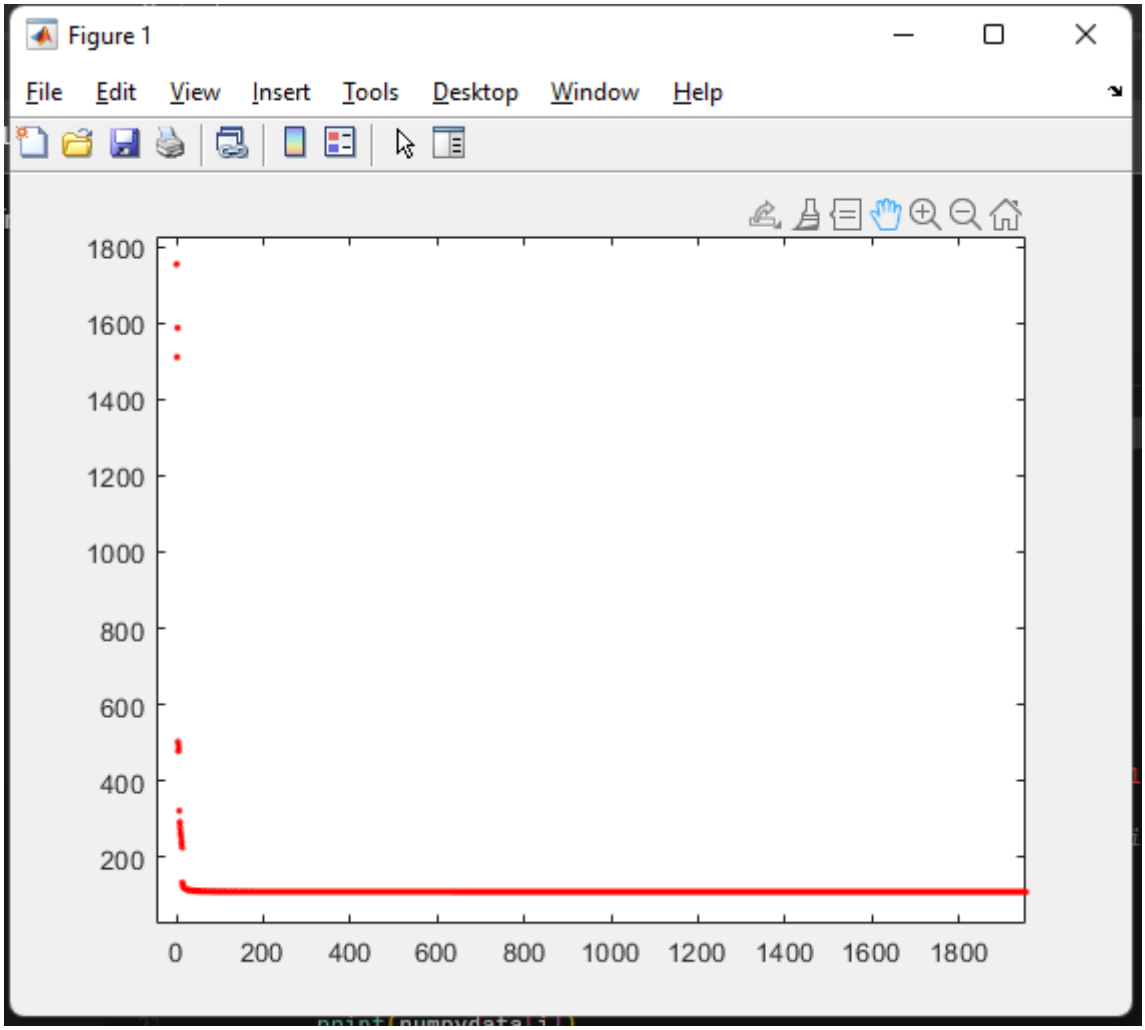
Com as imagens, comecei a trabalhar num código para transformá-las em vetores e armazenar as informações da luminosidade normalizada.

normalize.py > ...

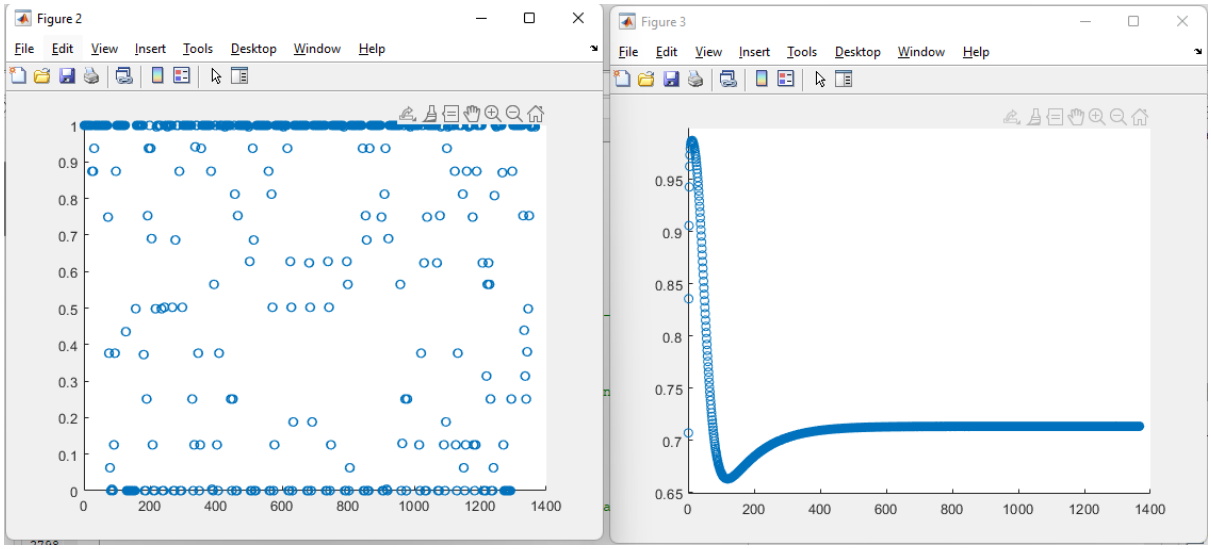
```python
import numpy as np
from PIL import Image
from numpy import asarray

image = Image.open('0-1.jpg')
numpydata = asarray(image)
#normalize the data for this array
numpydata = numpydata / 255.0
#reshape the data into a vector and store the info for reshaping it later
print(numpydata[0])
rows = numpydata.shape[0]
cols = numpydata.shape[1]
numpydata = numpydata.reshape(numpydata.shape[0] * numpydata.shape[1] * numpydata.shape[2])

#print all values from 1 to numpydata.shape[0] and save it into a file
for i in range(1, numpydata.shape[0]):
    print(i)
    if i == numpydata.shape[0]:
        break
    else:
        print(numpydata[i])
        print("\n")
        with open("x.txt", "a") as myfile:
            myfile.write(str(i))
            myfile.write("\n")
            myfile.close()
        with open("t.txt", "a") as myfile:
            myfile.write(str(numpydata[i]))
            myfile.write("\n")
            myfile.close()
```

Com isso, obtive um vetor com as informações normalizadas para uma imagem. Após isso, comecei a testar modificações no código do trabalho anterior para fazê-lo funcionar com todas as informações. Não obtive muito sucesso.

Essa foi a curva do erro quadrático obtida:



E aqui, os padrões obtidos a partir do vetor da fonte e a aproximação feita pela rede neural:

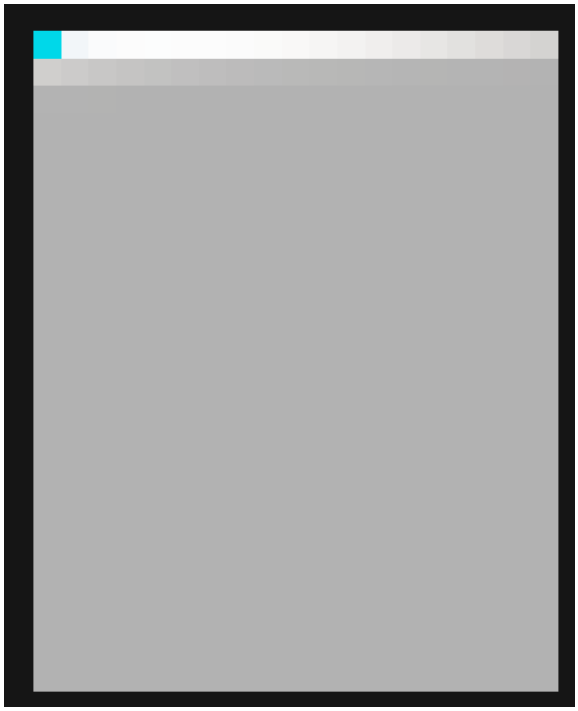


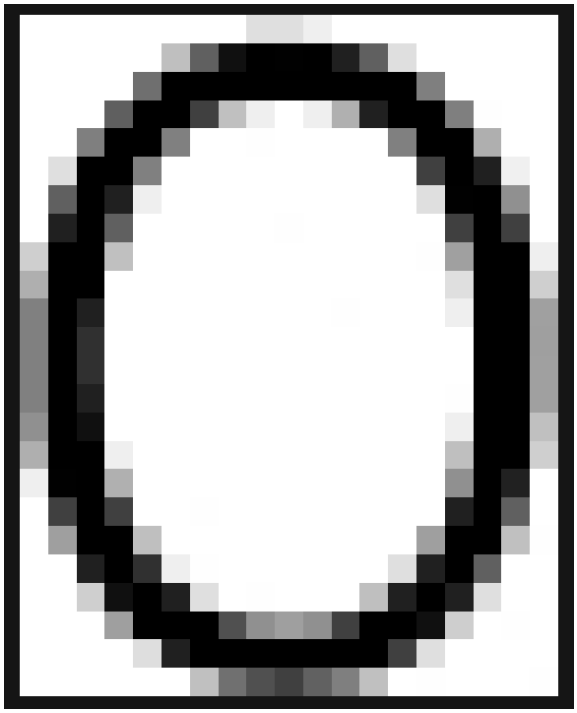
Não consegui efetuar as modificações no código, mas mesmo assim, resolvi continuar reformatando esses pontos para imagem, tentando fechar e simular o fluxo de todo um caso bem-sucedido, onde mostra-se os padrões, a rede aprende os padrões, gera valores com base na entrada que se aproximam muito dos valores originais e esses valores, quando convertidos novamente para imagem, geram uma imagem similar a inicial.

Desnormalizei o array gerado pela rede neural e o converti em uma imagem, para simular todo o fluxo.

```
data = data.reshape(rows, cols, 3)
#denormalize the data
data = data * 255
#convert to uint8
data = data.astype(np.uint8)
#save the image
new_image = Image.fromarray(data)
new_image.save('new.png')
```

A imagem gerada, como esperado, não representa em nada a fonte, mas foi útil para aprendizado.





Treinamento:

```
%treinamento irá parar pelo número de ciclo ou pelo errototal alcançado.
while (ciclo < numciclo) && (errototal > errototaladmissivel)
   ciclo=ciclo+1;
   errototal=0;


   for padroes=1:size(x)
      % -----------fase feedforward --------------
   % inserção de cada padrao de treinamento na entrada da rede neural
   % e cálculo das saídas z dos neurônios escondidos
        for j=1:neuroniosescondidos
            zin(j)= x(padroes)*v(j)+bv(j);
            z(j) = (2/(1+exp(-zin(j))))-1;
        end
       %%Cálculo da  saída  da rede
        yin=z*w+bw;
        y=(2/(1+exp(-yin)))-1;

   % -------------fase da Retropropagação do erro----------
   % da saida para a camada escondida
   %cálculos do deltinhaw do neurônio de saída
     deltinhaw = (t(padroes) - y)*0.5*(1+y)*(1-y);
  % cálculo dos deltaw para atualização dos pesos do neurônio de
  % saida
     for j=1:neuroniosescondidos
       deltaw= alfa*deltinhaw*z(j);
     end

  %cálculo das atualizações dos bias dos neurônios de saida
     deltabw=alfa*deltinhaw;
```

```
  % cálculo dos deltaw para atualização dos pesos do neurônio de
  % saida
     for j=1:neuroniosescondidos
       deltaw= alfa*deltinhaw*z(j);
     end

  %cálculo das atualizações dos bias dos neurônios de saida
     deltabw=alfa*deltinhaw;

 % cáculo dos deltinhav da camada escondida para as atualizações dos pesos
 % dos neurônios escondidos
    for j=1:neuroniosescondidos
      deltinhav(j)=deltinhaw*w(j)*0.5*(1+z(j))*(1-z(j));
    end

%cálculo dos deltav para atualização dos pesos dos neurônios escondidos
    for i=1:neuroniosentrada
      for j=1:neuroniosescondidos
       deltav(i,j)= alfa*deltinhav(j)*x(padroes,i);
      end
    end

% cálculo dos deltabv, bias dos neurônios escondidos
    for i=1:neuroniosescondidos
      deltabv(i)= alfa*deltinhav(i);
    end

%-------atualização dos pesos----------
% dos neurônios da camada de saída
    w=w+deltaw;
    bw=bw+deltabw;
% dos neurônios da camada escondida
   for i=1:neuroniosentrada
     for j=1:neuroniosescondidos
       v(i,j)= v(i,j)+deltav(i,j);
     end
   end
   for i=1:neuroniosescondidos
     bv(i)=bv(i)+deltabv(i);
   end
```

```
    % cálculo do erro total
    % é a soma de todos os erros  produzidos pelos neurônios de saida
    % para todos os padrões de treinamento
    errototal=errototal+0.5*((t(padroes)-y)^2);
   end % for padrões de entrada
    %plot(ciclo,errototal,'r.');
    erroquadraticototal(ciclo)=errototal;
end % while
```

Teste e Resultado:

```
disp('Fim do treinamento');
disp('Erro quadrático total final: ');
disp(errototal);
disp('Ciclos: ');
disp(ciclo);
disp('');
disp('');
disp('Teste da rede treinada');

final_values = [];

%--------teste da rede com os padrões de treinamento ---------
    for padroes=1:size(x)
        for j=1:neuroniosescondidos
         zin(j)= x(padroes,:)* v(:,j)+bv(j);
         z(j) = (2/(1+exp(-zin(j))))-1;
        end
        %Cálculo da saída  da rede
         yin=z*w+bw;
         y=(2/(1+exp(-yin)))-1;
         final_values(padroes) = y;
         fprintf("t: %f   y: %f\n",t(padroes),y);
    end


plot(erroquadraticototal,'r.');
figure
scatter(x,t)
figure
scatter(x,final_values)
```

Meu código completo:

Em anexo ao envio