Trabalho 1: Implementação de MLP

João Victor Garcia Coelho (Nº 10349540)
Matheus Carvalho Raimundo (Nº 10369014)
Matheus Sanchez (Nº 9081453)

Orientador: Prof. João Luís G. Rosa

USP - São Carlos

Maio/2019

Índice Analítico

1. Considerações iniciais

A sociedade está entrando em uma nova era da informação: a era da grande quantidade de dados. Com isso, começa-se a notar que precisa-se desenvolver novas formas de programação. Precisa-se de sistemas inteligentes, que possam oferecer ao usuário final uma resposta mais humana possível.

Com isso, percebe-se a grande relevância que as Redes Neurais Artificiais (RNA) têm sobre esses novos sistemas. Para fins deste projeto, foi determinado que uma RNA MultiLayer Perceptron fosse implementada, utilizando o algoritmo *Error Back-propagation*. Após, análises e estatísticas foram determinadas sobre esta rede implementada e seu treinamento através de um conjunto de dados fornecido.

2. Conjunto de dados

O conjunto de dados utilizado para treinamento e testes desta RNA utilizado se trata de imagens/fotos de dígitos (0-9). Por se tratar de um treinamento supervisionado, há também a resposta desejada para tal conjunto de dígitos.

O arquivo está em formato de texto e possui extensão **.data*. Cada linha do arquivo representa um exemplo. Em cada linha deve haver 256 números do tipo inteiro ou float, mais 10 números do tipo inteiro ou float.

Os 256 primeiros números variam entre os valores 0 e 1 (imagem monocromática) e representam todos os pixels da imagem. Os 10 últimos números variam entre os valores 0 e 1 e dizem se a imagem representa o dígito 0, 1, …, 8 ou 9 (1) ou não (0).

3. Implementação

O projeto foi implementado na linguagem Python 3, utilizando o ambiente Linux Mint. Utiliza as bibliotecas *numpy*, *math* e *re*. Ao fim da implementação, tal se apresentou nos testes totalmente funcional e sem erros.

Implementadas como variáveis globais, é possível modificar o número de neurônios na camada escondida, a taxa de aprendizado e o limite de erro máximo permitido (que determina se um teste falhou ou sucedeu).

3.1. Entrada e manipulação

A entrada do programa se dá por meio do terminal ou linha de comandos. Ao iniciar, um menu no terminal é apresentado com as funcionalidades disponíveis. Lembrando que o programa permite inclusive testar dados antes mesmo de treinar a RNA: é necessário acessar a funcionalidade de treinamento antes da de testagem caso o objetivo seja a acurácia.

3.2. Estrutura da Rede

A RNA possui fixamente 2 camadas, sendo uma escondida e uma de saída, e totalmente conectada. A camada escondida pode possuir vários neurônios, e a implementação permite a mudança no número de neurônios. Contudo, a camada de saída possui fixamente 10 neurônios, que representam cada dígito (0-9) possível. Também é possível variar os valores para a taxa de aprendizado (η) da rede neural na implementação do programa.

Os pesos estão sendo representados como um array de array de array (3 dimensões). Na primeira dimensão encontram-se cada uma das 2 camadas. Na segunda dimensão, têm cada um dos neurônios da camada. Na terceira dimensão estão os pesos deste neurônio. Essa abordagem foi escolhida porque através da biblioteca *numpy* é possível fazer processamento mais rápido e fácil desses arrays/vetores. Há também uma estrutura do tipo array de array. Na primeira dimensão encontram-se cada uma das 2 camadas. Na segunda dimensão, há os *biases* para cada neurônio. Ao início do programa, tais pesos e *biases* destas estruturas são valores aleatórios, mas ao longo do treinamento eles se alteram.

A função de ativação fixamente escolhida foi a sigmóide logística, com $\lambda = 1$:

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

3.3. Treinamento e aprendizado

O treinamento é realizado exemplo-a-exemplo, mudando os pesos e *biases* à cada execução. Para o treinamento, primeiramente é feito o *feed-forward*, armazenando os valores intermediários em cada camada. Após, é calculado o erro e é feito o *back-propagation*. O erro é propagado da última camada para as camadas inferiores, através das equações diferenciais do erro (bibliografia). A biblioteca *numpy* é amplamente utilizada para manipular os arrays/vetores com maior desempenho.

4. Execução

Para executar o programa, é necessário instalar Python 3 e ter uma linha de comandos (terminal) minimamente regular. Para execução, use *python3 T1.py* no diretório do código *T1.py* do programa. O programa apresenta porcentagem de processamento concluído para algumas funcionalidades (‘\r’ é inserido no terminal para atualizar essa porcentagem).

4.1. Menu principal

Após iniciar o programa, a rede neural artificial tem seus pesos iniciados com valores aleatórios e o menu se apresenta ao usuário para utilização:

```
mcarvalhor@Matheus-LM-Dell:~/trab-neurais/TrabalhoRedesNeurais$ python3 T1.py
Bem-vinde à nossa rede neural artificial, implementada através de uma MLP.

== Opções ==
1. Treinar com conjunto de dados
2. Testar com conjunto de dados
3. Treinar e testar com conjunto de dados (k-fold)
4. Testar com conjunto de dados (com detalhes)
5. Listar pesos
6. Informações da Rede
0. Sair

Escolha uma opção (0-6) > 3
Entre com um arquivo com o conjunto de dados (imagens) > semeion.data
Entre com o número de iterações para aprendizado (n) > 10
Entre com o número de blocos usados k (k) > 5

# A execução para a iteração k = 1 do 5-fold foi completa.
Acertos: 279 de 318 (87.74%).
Erro: total de 58.72 e médio de 0.18.
# A execução para a iteração k = 2 do 5-fold foi completa.
Acertos: 295 de 318 (92.77%).
Erro: total de 34.91 e médio de 0.11.
# A execução para a iteração k = 3 do 5-fold foi completa.
Acertos: 309 de 318 (97.17%).
Erro: total de 12.89 e médio de 0.04.
# A execução para a iteração k = 4 do 5-fold foi completa.
Acertos: 315 de 318 (99.06%).
Erro: total de 7.54 e médio de 0.02.
# A execução para a iteração k = 5 do 5-fold foi completa.
Acertos: 317 de 318 (99.69%).
Erro: total de 4.28 e médio de 0.01.
100.00% processado... (não pressione nenhuma tecla ainda)
# A rede aprendeu com o conjunto de dados submetido! Pesos atualizados. Além disso também foi testada.
Acertos: 1515 de 1590 (95.28%).
Erro: total de 118.34 e médio de 0.07.

== Opções ==
1. Treinar com conjunto de dados
2. Testar com conjunto de dados
3. Treinar e testar com conjunto de dados (k-fold)
4. Testar com conjunto de dados (com detalhes)
5. Listar pesos
6. Informações da Rede
0. Sair

Escolha uma opção (0-6) >
```

Ao fim da execução de cada funcionalidade, este menu é novamente apresentado. Forneça entrada ao programa (aperte alguma tecla) apenas quando solicitado. Caso contrário, o programa pode ser encerrado devido à entrada mal formatada.

4.2. Funcionalidade 1

Para a funcionalidade 1 (treinamento com um conjunto de dados), é passado para o programa um arquivo no formato adequado com um conjunto de dados que serão usados apenas para treinar a rede (os pesos são alterados). É possível também escolher o número de iterações sobre esse conjunto de dados. O programa mistura (*shuffle*) os elementos do conjunto de dados antes de cada iteração.

4.3 Funcionalidade 2

Para a funcionalidade 2 (testagem com um conjunto de dados), é passado para o programa um arquivo no formato adequado com um conjunto de dados que serão usados apenas para testar a rede (os pesos não são alterados). Ao fim, é mostrado a estatística da execução.

4.4 Funcionalidade 3

Para a funcionalidade 3 (treinamento e testagem *k-fold* com um conjunto de dados), é passado para o programa um arquivo no formato adequado com um conjunto de dados que serão usados tanto para treinar a rede (os pesos são alterados), quanto para testar a rede (os pesos não são alterados). Trata-se do algoritmo *k-fold*. São realizadas *k* iterações. O conjunto de dados fornecido é dividido em *k* partes diferentes de tamanhos iguais (exceção para casos de divisão por *k* não-exata dos dados de entrada). Dessas *k* partes do conjunto de dados, *k - 1* é utilizado para treinar a rede e 1 é utilizado para testar sobre os novos pesos em cada iteração. Esse treinamento também pode ser repetido *n* vezes. É possível escolher os valores *k* e *n*. O programa mistura (*shuffle*) os elementos do conjunto de dados antes de iniciar o treinamento. Ao fim, é mostrado a estatística da execução.

4.3 Funcionalidade 4

A funcionalidade 4 (testagem detalhada com um conjunto de dados), é idêntica à funcionalidade 2, com a única exceção que ela mostra também os valores esperados e produzidos para cada exemplo (qual dígito foi apropriadamente determinado).

4.5 Funcionalidade 5

Para a funcionalidade 5 (listagem dos pesos da rede), é passado para o programa uma camada e um neurônio dentro dessa camada (números inteiros). O programa vai listar todos os pesos e o *bias* deste neurônio.

4.6 Funcionalidade 6

Para a funcionalidade 6 (informações da rede), é listado algumas informações da rede como estrutura, se já foi treinada, e caso sim, o número de exemplos que já foram usados para treinamento.

5. Estatística e discussão

Para a avaliação do modelo projetado, foi o utilizado a funcionalidade 3, validação cruzada junto com 5 folds, gerando diversas estatísticas.

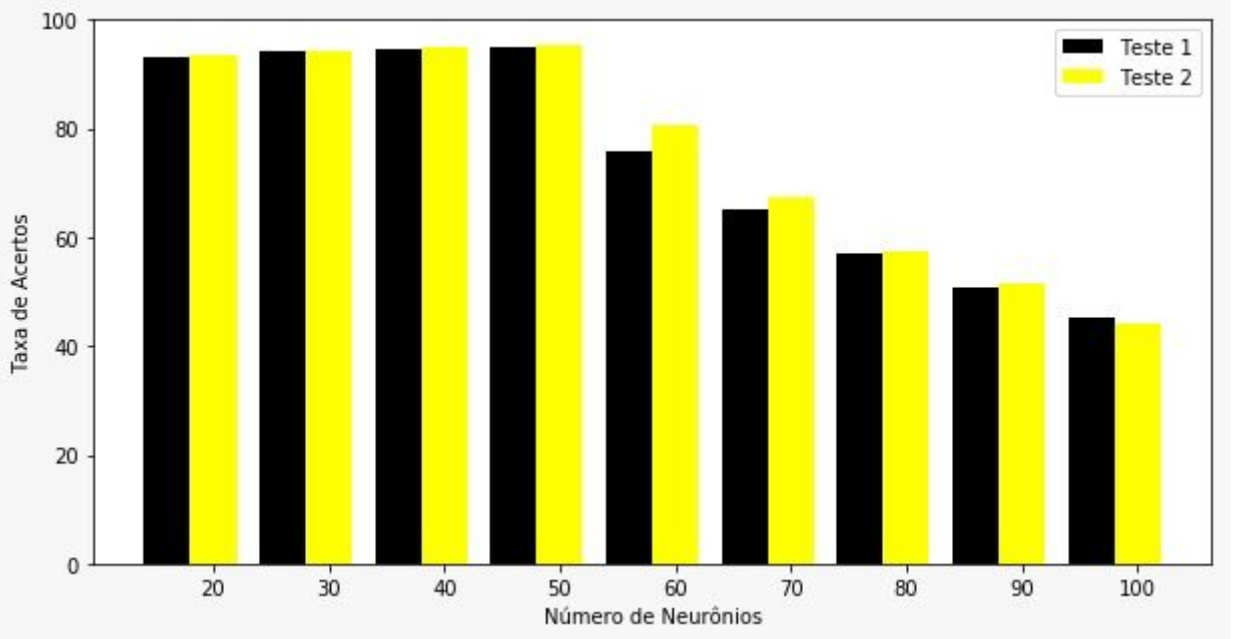

Gráfico 1: Taxa de Acertos x Número de Neurônios

O gráfico 1 mostra a taxa de acertos para uma rede com a mesma estrutura (uma camada escondida e uma cada de saída com 10 neurônios), variando de 10 em 10 o número de neurônios da camada escondida, com a **taxa de aprendizado (η) fixada em 1**. Além disso, como os valores dos pesos iniciais são aleatórios, foram realizados dois testes com o mesmo número de neurônios para a camada escondida (Teste 1 e Teste 2), inicializando os pesos da ligações com valores diferentes.

A partir do gráfico, é possível realizar que a maior porcentagem de acertos se dá quando o número de neurônios da camada escondida é igual a 50. A partir disso, a rede começa a memorizar os dados de treinamento, devido à enorme quantidade de neurônios, perdendo a capacidade de generalização.

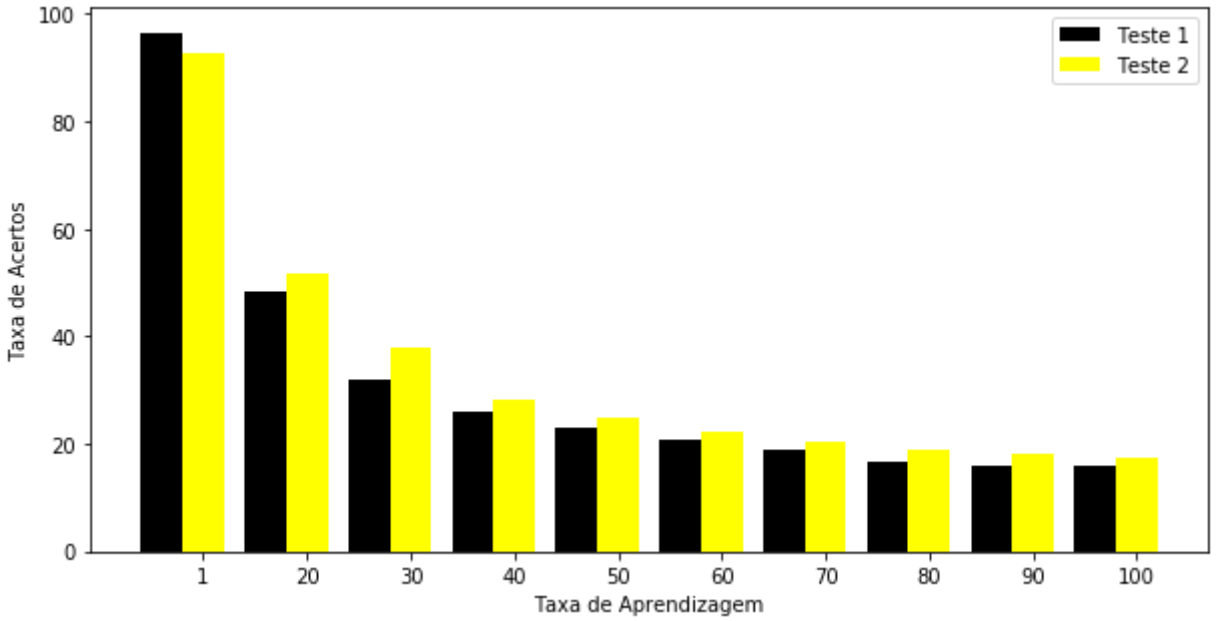

Gráfico 2: Taxa de Acertos x Taxa de Aprendizagem

O gráfico 2 mostra a variação da taxa de acertos da rede com uma camada escondida com 50 neurônios e uma camada de saída com 10 neurônios, variando o valor da taxa de aprendizado de 10 em 10, ou seja, com η de 20 a 100.

A partir disso, é possível inferir que conforme o valor da taxa de aprendizado aumenta, a sua taxa de acertos diminui. Isso se dá pois conforme a taxa de aprendizado aumenta, o valor dos pesos aumentam também, fazendo com a rede tenha dificuldade em encontrar valores bons para as ligações, e consequentemente, perdendo na porcentagem de acerto.

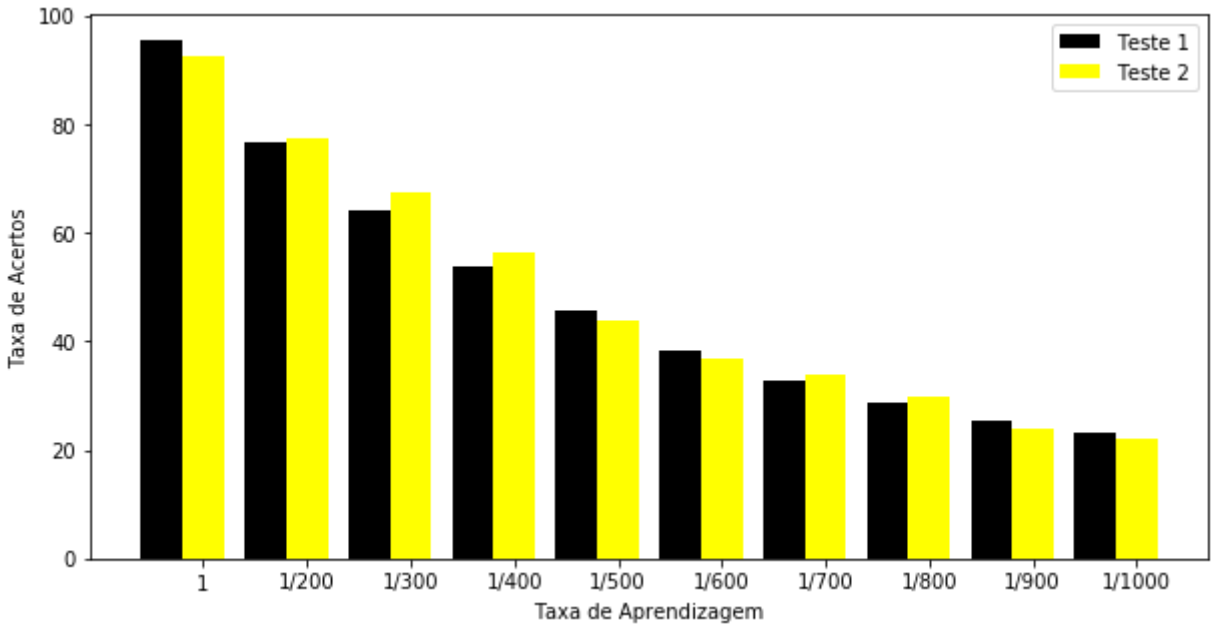

Gráfico 3: Taxa de Acertos x Taxa de Aprendizagem

O gráfico 3, por sua vez, mostra uma abordagem de avaliação parecida a mostrada no gráfico 2, onde o número de neurônios da camada escondida é fixado em 50, e é variado o valor da taxa de aprendizado (η). Nesse caso, o valor dela é variado de 1 a 0.001. A partir do gráfico é possível concluir que conforme o valor da taxa de aprendizado vai diminuindo, a porcentagem de acerto também. Isso se deve ao fato de que com um valor de η muito pequeno, os pesos se modificam muito pouco em direção ao seu valor ideal, fazendo com que a rede não aprenda corretamente.

A partir desses resultados apresentados, é possível ver que a maior porcentagem de acerto, isto é, a rede que melhor aprendeu, foi aquela com a estrutura de 50 neurônios na camada escondida e 10 na camada de saída, com uma taxa de aprendizado η igual a 1.

6. Conclusão

Neste projeto, foi possível chegar à várias conclusões. A primeira e a mais intuitiva delas é que as redes neurais MLP podem ser muito lentas ou muito rápidas dependendo da forma como são implementadas. Em primeira instância, o grupo não utilizava a biblioteca *numpy*, e estava manipulando os arrays/vetores manualmente através de iterações *for*. Foi percebido que tanto o aprendizado, como a testagem estavam muito lentos quando o número de neurônios na camada escondida era grande. Foi então feita a mudança de modo que a biblioteca *numpy* pudesse ser usada sempre que possível, e assim, muito provavelmente devido à forma como a biblioteca funciona internamente no Python, a performance do

programa melhorou drasticamente (apesar de isso ter feito o código ter ficado de pior compreensão quando comparado com as iterações *for*)

A segunda conclusão a qual chegou-se, também intuitiva, é que tanto a taxa de aprendizado (η) como o número de camadas e de neurônios em cada camada afetam o modo como a rede deve ser treinada para convergir e obter os resultados finais. De acordo com os testes, a rede neural apresentou a melhor acurácia quando utilizou 2 camadas, com 50 neurônios na camada escondida e 10 neurônios na camada de saída totalmente conectados, e uma taxa de aprendizado $\eta = 1$. Ao treinar essa rede com a metodologia *5-fold*, obteve-se acurácia muito próxima de 100% nos exemplos.

Uma outra conclusão à qual chegou-se também, é que alterar o número de neurônios em cada camada ou alterar os pesos iniciais desses neurônios, na forma como implementou-se o projeto, é possível e fácil: apenas mudando o valor de uma variável no código. Contudo, se a mudança na estrutura interna exigir mudar o número de camadas, a mudança no código deixa de ser simples. Isso porque no algoritmo *Error Back-propagation*, o erro é propagado da camada de saída para as camadas anteriores através das derivadas sob a função de ativação. Isso quer dizer que para cada camada a qual o erro se propaga, é necessário derivar a derivada da camada anterior na direção do peso: e por isso a mudança no código se torna mais difícil.

Por fim, pode-se dizer que os conceitos vistos em sala de aula na teoria foram colocados na prática com esse projeto. E com a prática, não apenas algumas questões/dúvidas foram solucionadas, como também enxergou-se melhor como o conceito da MLP se dá em aplicações da vida real e como funciona a estrutura da RNA - não porque os pesos mudam, mas como eles mudam.

7. Bibliografia

Material de aula do professor disponibilizado.

Redes Neurais, Perceptron Multicamadas e o Algoritmo Backpropagation. Disponível em <https://medium.com/ensina-ai/redes-neurais-perceptron-multicamadas-e-o-algoritmo-backpropagation-eaf89778f5b8>. Acessado em 1 de maio de 2019.