Utilizando redes neurais para identificar dígitos manuscritos

Emmanuel Podestá Jr., Gustavo Zambonin*
Inteligência Artificial (UFSC – INE5430)

1 Fundamentação

A utilização de redes neurais para processamento de imagens é bastante conhecida na literatura. Neste trabalho, uma implementação em Python 3 a partir da biblioteca PyBrain demonstra a capacidade de reconhecimento de padrões desta abordagem computacional.

O conjunto escolhido para classificação consiste de uma fração da base de dados MNIST (*Mixed National Institute of Standards and Technology database*), organizada como uma lista de imagens, cada qual como uma lista de *pixels* no intervalo $[-1, 1]$ e um valor inteiro representando o dígito correspondente no intervalo $[1, 10]$. A normalização das intensidades de preto é realizada apenas caso a visualização das imagens seja desejada; o método *feature scaling* é aplicado a cada uma das listas para que uma imagem no formato PGM *Portable Graymap Format*) seja gerada.

$$x' = \frac{(x - min(y)) \times 255}{max(y) - min(y)}$$

Figura 1: Equação que normaliza valores para o intervalo $[0, 255]$. A lista de valores original é representada por y, x é valor atual e x' é o valor normalizado.

Do contrário, é necessário apenas adaptar os dígitos correspondentes à identidade das imagens para vetores de tamanho 10 (pois existem 10 dígitos diferentes) com apenas uma coordenada ativa, aquela cujo índice é igual ao seu valor módulo 10. A arquitetura da rede neural é baseada nesta organização de dados; três camadas são criadas para processar os dígitos:

- a camada de entrada é construída com 400 neurônios, o número de *pixels* de uma dada imagem; observou-se que não é necessário utilizar uma função complexa de processamento de valores, bastando apenas uma função de ativação linear ($f(x) = x$).
- a camada "escondida", construída com 40 neurônios, utilizará da função de ativação tangente hiperbólica para atualizar seus valores ($f(x) = \frac{2}{1+e^{-2x}} - 1$).
- a camada de saída, construída com 10 neurônios, o número de possíveis saídas, classificará o resultado atual a partir da função *softmax*, que transforma um vetor $\mathbf{z}$ com valores reais arbitrários para um vetor $\sigma(\mathbf{z})$ com valores reais no intervalo $(0, 1)$, cuja soma é 1. Ambos os vetores têm dimensão K. A saída dessa função pode ser utilizada para representar uma distribuição de probabilidades sobre K diferentes possibilidades, relevante no contexto de classificação de dados.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \; \forall j = 1, \ldots, K$$

2 Implementação

A separação dos dados em bases de treino e de teste, sob uma proporção 80/20, foi feita após embaralhamento das imagens na lista. O treinador escolhido utiliza o método do gradiente descendente com mini-*batches* e *backpropagation*, com taxa de aprendizado $\mu = 0.04$ e treino por 30 épocas. Estudando o problema, percebeu-se que com estes parâmetros, os resultados são razoáveis, apresentando uma porcentagem de acertos interessante ($\approx 98\%$ considerando toda a base de dados fornecida), sem muita flutuação ou estagnação de valores; um número muito maior de épocas torna o desempenho proibitivamente baixo.

O programa responsável por controlar a rede neural pode ser encontrado em `mnist_classify.py`. Um exemplo de execução segue abaixo; o argumento `-plot` gera a matriz de confusão apresentada em seguida.

*{emmanuel.podesta,gustavo.zambonin}@grad.ufsc.br

src/

```
$ python mnist_classify.py --plot
Epoch:   1   All: 81.74%   Train: 82.75%   Test: 77.70%
Epoch:   2   All: 87.16%   Train: 87.92%   Test: 84.10%
Epoch:   3   All: 89.88%   Train: 90.72%   Test: 86.50%
...
Epoch:  28   All: 97.76%   Train: 99.95%   Test: 89.00%
Epoch:  29   All: 97.90%   Train: 99.95%   Test: 89.70%
Epoch:  30   All: 97.92%   Train: 99.95%   Test: 89.80%
```

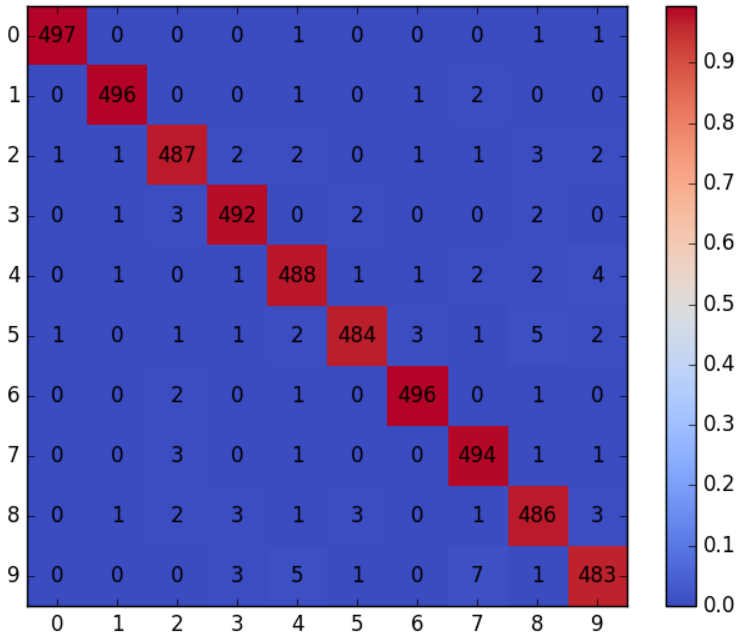

Figura 2: Matriz de confusão gerada após a finalização do treinamento com o conjunto de argumentos padrão.