Trabalho sobre reconhecimento de padrões

Christian de Pieri
Lucas João Martins

1 - Introdução

Esse relatório busca desenvolver as ideias obtidas em sala que auxiliaram no desenvolvimento do quinto trabalho da disciplina INE-5430 (Inteligência Artificial) do curso de Ciência da Computação da Universidade Federal de Santa Catarina. Neste trabalho, há o desenvolvimento de uma rede neural que classifica dígitos manuscritos obtidos a partir de um subconjunto de dados da base de dados MNIST. Os dados de entrada servem para treinar, validar e testar a rede neural.

A rede neural foi desenvolvida na linguagem de programação Python. O trabalho está separado em três arquivos:

- t5.ipynb (arquivo que contém todo o desenvolvimento da rede neural);
- t5.py (correspondente ao arquivo anterior em python puro);
- exdata.csv (arquivo de dados fornecido pelo professor).

Sobre as tecnologias utilizadas, a plataforma de desenvolvimento foi o Jupyter. Já o Keras foi utilizado como a API responsável por criar a rede neural, que utilizou o TensorFlow como back-end. Enquanto que o NumPy e o scikit-learn mostraram-se bibliotecas essenciais para se trabalhar nesse escopo com Python. Outra ferramenta que nos auxiliou durante a execução deste trabalho foi o GitHub, que é essencial para o desenvolvimento de um software em grupo, além de ser uma boa maneira para ter o histórico da aplicação. Ambos os alunos optaram pela escolha dessas tecnologias como forma de adquirir novos conhecimentos, já que nem todas elas fazem parte do seu dia-a-dia.

2 - Normalização dos dados

Conforme o enunciado, se parte do pressuposto de que os dados já foram previamente normalizados. Por isso, só foi necessário, após os imports e load do dataset, a realização de um pré-processamento dos dados. Esse pré-processamento tem como objetivo deixar as informações no formato adequado para os próximos passos.

Import dos módulos necessários e declaração de constantes

In [1]:
```
import numpy as np

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D

from sklearn import metrics
from sklearn.model_selection import train_test_split

# quantidade de canais que as imagens de entrada possuem
# escalar cor de cinza, entao 1
CHANNEL = 1
# valor da largura e da altura das imagens de entrada
WIDTH_HEIGHT = 20
# tamanho do batch
BATCH = 128
```

```
Using TensorFlow backend.
```

Load do arquivo de dados

In [2]:
```
load = np.loadtxt('exdata.csv', delimiter=',')

# cada coluna tem um padrao de digito
data = load[:-1].T

# a ultima linha eh a classificacao do digito
result = load[-1]
# digito 0 corresponde ao valor 10
result[result == 10] = 0
```

Pré-processamento dos dados e das classes

In [3]:
```
# trasformar cada linha (digito) em uma matriz 20 x 20 x 1
data = data.reshape(data.shape[0], WIDTH_HEIGHT, WIDTH_HEIGHT, CHANNEL)

# converte array de 1 dimensao para uma matriz de dimensao 10
# ou seja, criar 10 classes, uma para cada digito possivel
result = keras.utils.to_categorical(result, 10)
```

Imagem 1 - Começo do código e pré-processamento dos dados de entrada

3 - Separação do conjunto de treinamento e de teste

O conjunto de treinamento e de teste foram separados de forma aleatória através da função train_test_split do scikit-learn. Optou-se por utilizar 25% dos dados para teste e 75% para treinamento, ou seja, 1250 amostras para teste e 3750 amostras para treinamento.

Separação dos dados em treinamento e teste

In [4]:
```
in_train, in_test, out_train, out_test = train_test_split(data,
                                                          result,
                                                          test_size=(25/100),
                                                          train_size=(75/100))
```

Imagem 2 - Separação dos dados em treinamento e teste

4 - Arquitetura de rede neural utilizada

A rede neural utilizada é uma Convolutional Neural Networks com 8 diferentes camadas. A ideia do que cada camada realiza pode ser vista nos comentários do código na imagem 3. Porém, vale citar que como ‘activation functions’ das camadas optou-se pelo ‘softmax’ e pelo ‘rectified linear unit’. Além disso, como ‘optimizer’ optou-se pelo uso do ‘Adadelta’ (extensão robusta do ‘Adagrad’), e, como ‘loss function’ foi utilizada a ‘categorical cross entropy’. Todas essas escolhas foram feitas a partir da visualização de outros exemplos que solucionam problemas parecidos.

Definição da arquitetura da rede neural

In [5]:
```
# 'pilha' de camadas lineares
model = Sequential()

# primeira camada precisa saber o que espera de entrada
# Conv2D cria uma camada de 'convolution' (add cada elemento da imagem com o seu vizinho local)
# isso eh feito atraves do input_shape
# 32 eh o numero de filtros
# relu = rectified linear unit
model.add(Conv2D(32,
                 kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(WIDTH_HEIGHT, WIDTH_HEIGHT, CHANNEL)))
model.add(Conv2D(64, (3, 3), activation='relu'))

# MaxPooling2D cria uma camada que faz um processo de discretizacao baseada em amostra
model.add(MaxPooling2D(pool_size=(2, 2)))

# Dropout cria uma camada de regularizacao
# 0.25 eh a fracao da quantidade de entrada que entrara na camada
model.add(Dropout(0.25))

# Flatten cria uma camada que 'flatteniza'
model.add(Flatten())

# Dense cria uma camada que representa uma multiplicacao de matrizes
# 128 eh a dimensionalidade da saida
model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

Compilação da rede neural

In [6]:
```
# configuracao de que como sera o aprendizado de processo
# para qualquer problema de classificacao deve-se usar o accuracy
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

Imagem 3 - Definição da rede neural

5 - Quantos e quais experimentos foram feitos

Foram feitos alguns experimentos antes de chegar na arquitetura final apresentada na seção anterior. Esses experimentos consistiram em mudar as ‘activations functions’ das camadas, utilizar outros ‘optimizer’ disponíveis (e.g. RMSprop e Nadam), ou, utilizar outra ‘loss function’ (e.g. logcosh). Porém, na maioria dos casos a taxa de acerto se manteve muito similar, sendo que em alguns casos ela caia alguns pontos de porcentagem. Por fim, optou-se pelos valores já apresentados devido a um entendimento um pouco maior sobre eles do que as outras opções, e, por estarmos satisfeitos com a taxa de acertos.

6 - Treinamento

O treinamento foi realizado através da função fit disponibilizada pelo Keras. Optou-se por iterar 10 vezes o dataset.

Input dos dados de treinamento

```
In [7]: # batch_size eh o numero de amostras por update do gradiente
        # epochs = um epoch eh uma iteracao sobre os dados fornecidos
        model.fit(in_train, out_train,
                  batch_size=BATCH,
                  epochs=10,
                  verbose=1,
                  validation_data=(in_test, out_test))
```

Imagem 4 - Treinamento dos dados

7 - Matriz de confusão

Após gerar a predição nos dados de teste com a função predict disponibilizada pelo Keras. Utilizou-se o metrics.confusion_matrix e o metrics.classification_report, ambos do scikit-learn, para gerar a matriz de confusão, e, também gerar um relatório com as principais métricas associadas com o processo de predição/classificação de padrões.

Matriz de confusão

In [9]:
```
# gera a predicao para o conjunto de teste
prediction = model.predict(in_test, batch_size=BATCH, verbose=0)

# ajuste dado para ter info correta
prediction_classes = np.argmax(prediction, axis=1)
out_test_classes = np.argmax(out_test, axis=1)

# gera a 'confusion matrix'
matrix = metrics.confusion_matrix(out_test_classes , prediction_classes)
print(matrix)
```

```
[[127   0   0   0   0   0   0   0   0   0]
 [  0 115   0   1   0   0   1   0   0   0]
 [  1   0 119   0   0   0   0   0   1   0]
 [  0   0   4 122   0   0   0   0   1   1]
 [  0   0   0   0 120   0   0   1   0   0]
 [  0   1   0   0   0 120   3   0   0   0]
 [  1   2   0   0   1   0 136   0   0   0]
 [  0   3   0   0   1   0   0 113   0   2]
 [  0   1   0   1   0   1   1   0 107   1]
 [  0   1   0   1   0   0   0   4   0 135]]
```

In [10]:
```
# gera um relatorio com as principais metricas da classificacao
report = metrics.classification_report(out_test_classes, prediction_classes)
print(report)
```

```
             precision    recall  f1-score   support

          0       0.98      1.00      0.99       127
          1       0.93      0.98      0.96       117
          2       0.97      0.98      0.98       121
          3       0.98      0.95      0.96       128
          4       0.98      0.99      0.99       121
          5       0.99      0.97      0.98       124
          6       0.96      0.97      0.97       140
          7       0.96      0.95      0.95       119
          8       0.98      0.96      0.97       112
          9       0.97      0.96      0.96       141

avg / total       0.97      0.97      0.97      1250
```

Imagem 5 - Predição, matriz de confusão e relatório com métricas

8 - Taxa de acertos

A taxa de acertos foi calculada através da função metrics.accuracy_score do scikit-learn. Obteve-se em média, após diversas execuções, uma taxa de precisão de 97%.

Precisão final ¶

In [11]:
```
# calcula a precisao da classificacao
value = metrics.accuracy_score(out_test_classes, prediction_classes)
print("Precisão no conjunto de teste: {:.2%}".format(value))
```

```
Precisão no conjunto de teste: 97.12%
```

Imagem 6 - Precisão da classificação

9 - Conclusão

A maior dificuldade encontrada foi trabalhar com diversas tecnologias até então um pouco distante dos integrantes da equipe. Além disso, percebeu-se que há um grande número de escolhas que devem ser feitas ao desenvolver uma rede neural - seja desde quais dos diversos parâmetros serão determinados, e toda a teoria por trás disso, até quais bibliotecas que serão utilizadas.

Contudo, após extensa leitura de material na internet e visualização de diversos tutoriais, acreditamos que evoluímos e aprendemos ao chegar em um resultado satisfatório no reconhecimento de padrões.

10 - Referências e links

O repositório no GitHub se encontra em:

- https://github.com/lucasjoao/neural_network_mnist

Alguns links interessantes sobre as tecnologias utilizadas são:

- http://jupyter.org/
- https://keras.io/
- https://www.tensorflow.org/
- http://www.numpy.org/
- http://scikit-learn.org/stable/index.html

E além do material fornecido pelo professor, foram utilizadas as seguintes fontes no desenvolvimento da rede neural:

- https://elitedatascience.com/keras-tutorial-deep-learning-in-python
- https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/