Universidade do Minho
Escola de Engenharia/Departamento de Informática

Visão por computador

Uma abordagem Deep Learning

Aulas práticas

IMAGEM MÉDICA

AMBIENTE DE TRABALHO

Para executarem os exercicios que irão ser propostos devem preparar o ambiente de trabalho com o seguinte sw:

python 3.x (versão 64bits)
+numpy + scipy + scikit-learn + scikit-image + h5py + matplotlib

se tiverem o anaconda instalado basta executar: conda install ....
opcional: instalar o opencv
instalar o keras: pip install keras
instalar o theano: pip install theano
instalar o tensorflow: pip install tensorflow

Para os alunos que prefiram ter uma maquina virtual com tudo já instalado coloquei uma maquina virtual (Ubuntu 16.10+opencv+keras + ...) num servidor da DI em:
https://reposlink.di.uminho.pt/uploads/d98e0b93a1b61d7dc996fe03052468fe.file.ubuntu16.10DLalunos.zip
Essa maquina tem o anaconda instalado e está configurada com ambientes.
Para trabalharem no ambiente correcto devem executar: source activate keras-test
Devem fazer o upgrade para o keras2:
pip install git+git://github.com/fchollet/keras.git --upgrade

podem fazer as instalações e upgrades tanto utilizando o conda como o pip mas sempre dentro do ambiente keras-test

PRIMEIRA REDE UTILIZANDO KERAS

Vamos desenvolver uma rede **MLP – MultiLayer Perceptron** para fazer classificação utilizando o dataset pima-indians:

http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

AULA1 – MLP

Bibliotecas necessárias para a execução desta aula.

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.models import model_from_json
from keras.models import model_from_yaml
import matplotlib.pyplot as plt

# fixar random seed para se puder reproduzir os resultados
seed = 9
np.random.seed(seed)
```

AULA1 – MLP

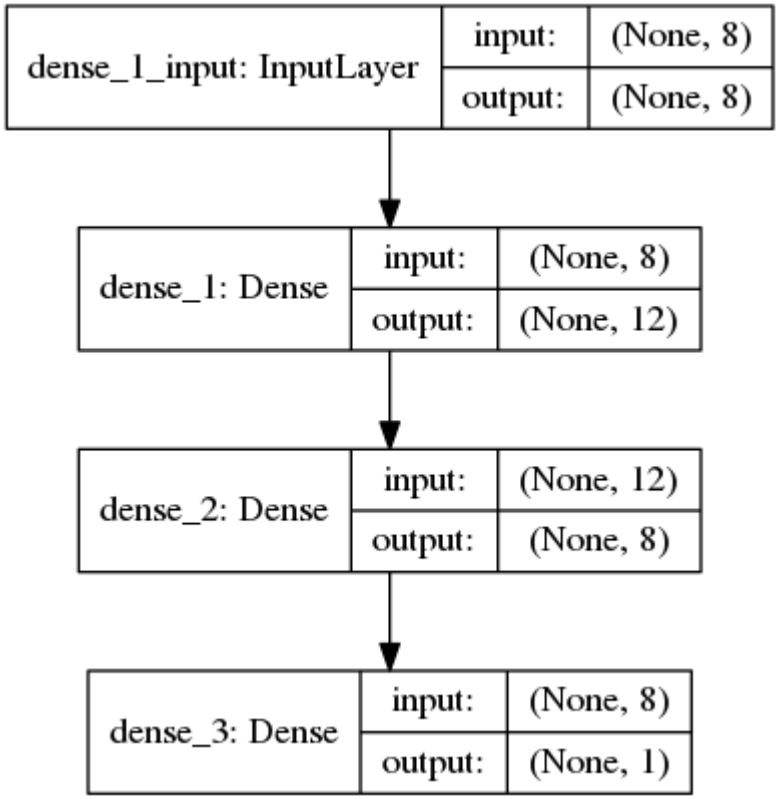
```
# Etapa 1 - preparar o dataset
'''
fazer o download do prima indian dataset sobre diabetes em população indigena (label
0,1 tem diabetes):
dataset: http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes
download: http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-
diabetes/pima-indians-diabetes.data
fazer o rename para pima-indians-diabetes.csv
'''
def read_cvs_dataset(ficheiro, col_label):
    # ler ficheiro csv para matriz numpy, e separar o label que está em col_label (deve
ser a ultima coluna)
    dataset = np.loadtxt(ficheiro, delimiter=",")
    print('Formato do dataset: ',dataset.shape)
    input_attributes = dataset[:,0:col_label]
    output_attributes = dataset[:,col_label]
    print('Formato das variáveis de entrada (input variables):
',input_attributes.shape)
    print('Formato da classe de saída (output variables): ',output_attributes.shape)
    #print(X[0])
    #print(Y[0])
    return (input_attributes,output_attributes)
```

AULA1 – MLP

```
# Etapa 2 - Definir a topologia da rede (arquitectura do modelo)
'''
cria-se um modelo sequencial e vai-se acrescentando camadas (layers)
vamos criar 3 camadas no nosso modelo
Dense class significa que teremos um modelo fully connected
    o primeiro parametro estabelece o número de neuronios na camada (12 na primeira)
    input_dim=8 indica o número de entradas do nosso dataset (8 atributos neste caso)
    kernel_initializer indica o metodo de inicialização dos pesos das ligações
    'uniforme' sigifica small random number generator com default entre 0 e 0.05
    outra hipotese seria 'normal' com small number generator from Gaussion
distribution
    "activation" indica a activation fuction
    'relu' rectifier linear unit activation function com range entre 0 e infinito
    'sigmoid' foi utilizada para garantir um resultado entre 0 e 1
'''
def create_model():
    model = Sequential()
    model.add(Dense(12, input_dim=8, activation="relu", kernel_initializer="uniform"))
    model.add(Dense(8, activation="relu", kernel_initializer="uniform"))
    model.add(Dense(1, activation="sigmoid", kernel_initializer="uniform"))
    return model
```

AULA1 – MLP

```
#util para visualizar a topologia da rede num ficheiro em pdf ou png
def print_model(model,fich):
    from keras.utils import plot_model
    plot_model(model, to_file=fich, show_shapes=True, show_layer_names=True)
```



AULA1 – MLP

```
# Etapa 3 - Compilar o modelo (especificar o modelo de aprendizagem a ser utilizado pela rede)
'''
loss - funcão a ser utilizada no calculo da diferença entre o pretendido e o obtido
vamos utilizar logaritmic loss para classificação binária: 'binary_crossentropy'
o algoritmo de gradient descent será o "adam" pois é eficiente
a métrica a ser utilizada no report durante o treino será 'accuracy' pois trata-se de um problema de classificacao
'''
def compile_model(model):
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

Aula1 – MLP

```
# Etapa 4 - treinar a rede (Fit the model)  neste caso foi feito com os dados todos
'''
'batch_size'núermo da casos processados de cada vez
model.fit(X, Y, nb_epoch=150, batch_size=10, verbose=2)
verbose: 0 para print do log de treino stdout, 1 para barra de progresso, 2 para uma linha por epoch.
validation_split: float (0. < x < 1). Fração de dados a serem utilizados como dados de validação.
'''
def fit_model(model,input_attributes,output_attributes):
    history = model.fit(input_attributes, output_attributes, validation_split=0.33, epochs=150, batch_size=10, verbose=2)
    return history
```

AULA1 – MLP

```
#utils para visulaização do historial de aprendizagem
def print_history_accuracy(history):
    print(history.history.keys())
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()

def print_history_loss(history):
    print(history.history.keys())
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
```

Aula1 – MLP

```
# Etapa 5 - Calcular o desempenho do modelo treinado (neste caso utilizando os dados usados no treino)

def model_evaluate(model,input_attributes,output_attributes):
    print("############inicio do evaluate##############################\n")
    scores = model.evaluate(input_attributes, output_attributes)
    print("\n metrica: %s: %.2f%%\n" % (model.metrics_names[1], scores[1]*100))
```

AULA1 – MLP

```
# Etapa 6 - Utilizar o modelo treinado  e escrever as previsões para novos casos

def model_print_predictions(model,input_attributes,output_attributes):
    previsoes = model.predict(input_attributes)
    # arredondar para 0 ou 1 pois pretende-se um output binário
    LP=[]
    for prev in previsoes:
        LP.append(round(prev[0]))
    #LP = [round(prev[0]) for prev in previsoes]
    for i in range(len(output_attributes)):
        print(" Class:",output_attributes[i]," previsão:",LP[i])
        if i>10: break
```

AULA1 – MLP

```
# Ciclo completo executando as Etapas 1,2,3,4,5 e 6

def ciclo_completo():
    (input_attributes,output_attributes) = read_cvs_dataset("pima-indians-diabetes.csv", 8)
    model = create_model()
    print_model(model,"model_MLP.png")
    compile_model(model)
    history=fit_model(model,input_attributes,output_attributes)
    print_history_accuracy(history)
    print_history_loss(history)
    model_evaluate(model,input_attributes,output_attributes)
    model_print_predictions(model,input_attributes,output_attributes)
```

AULA1 – MLP

```
# Utils para gravar modelos e pesos utilizá-los posteriormente
'''
Gravar um modelo num ficheiro utilizando o formato json. O nome do ficheiro deve ter a extensão .json
'''
def save_model_json(model,fich):
    model_json = model.to_json()
    with open(fich, "w") as json_file:
        json_file.write(model_json)
'''
Gravar um modelo num ficheiro utilizando o formato yaml. O nome do ficheiro deve ter a extensão .yaml
'''
def save_model_yaml(model,fich):
    model_yaml = model.to_yaml()
    with open(fich, "w") as yaml_file:
        yaml_file.write(model_yaml)
'''
Gravar os pesos de um modelo treinado num ficheiro utilizando o formato HDF5. O nome do ficheiro deve ter a extensão .h5
'''
def save_weights_hdf5(model,fich):
    model.save_weights(fich)
    print("Saved model to disk")
```

AULA1 – MLP

```
'''
Ler um modelo de um ficheiro no formato json e criar o respetivo modelo em memória.
'''
def load_model_json(fich):
    json_file = open(fich, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    return loaded_model
'''
Ler um modelo de um ficheiro no formato yaml e criar o respetivo modelo em memória.
'''
def load_model_yaml(fich):
    yaml_file = open(fich, 'r')
    loaded_model_yaml = yaml_file.read()
    yaml_file.close()
    return model_from_yaml(loaded_model_yaml)
'''
Ler os pesos um modelo treinado de um ficheiro no formato hdf5 para o respectivo
modelo.
'''
def load_weights_hdf5(model,fich):
    model.load_weights(fich)
    print("Loaded model from disk")
```

AULA1 – MLP

```
# exemplos de utilização destes utilitários

def ciclo_ler_dataset_treinar_gravar():
    (input_attributes,output_attributes) = read_cvs_dataset("pima-indians-diabetes.csv",8)
    model = create_model()
    print_model(model,"model2.png")
    compile_model(model)
    history=fit_model(model,input_attributes,output_attributes)
    print_history_accuracy(history)
    print_history_loss(history)
    model_evaluate(model,input_attributes,output_attributes)
    save_model_json(model,"model.json")
    save_weights_hdf5(model,"model.h5")
    return (input_attributes,output_attributes)



def ciclo_ler_modelo_evaluate_usar(input_attributes,output_attributes):
    model= load_model_json("model.json")
    load_weights_hdf5(model,"model.h5")
    compile_model(model)
    model_evaluate(model,input_attributes,output_attributes)
    model_print_predictions(model,input_attributes,output_attributes)
```

AULA1 – MLP

```
if __name__ == '__main__':
    #opção 1 - ciclo completo
    #ciclo_completo()
    #opção 2 - ler,treinar o dataset e gravar. Depois ler o modelo e pesos e usar
    (input_attributes,output_attributes)=ciclo_ler_dataset_treinar_gravar()
    ciclo_ler_modelo_evaluate_usar(input_attributes,output_attributes)
```