Exercícios de Fixação de Conceitos – EFC 2
Atividade Individual – Peso 3

Especificação: Trabalhe no ambiente Anaconda e elabore o seu relatório como um Jupyter notebook. Todas as atividades deverão, assim, compor um único relatório no Jupyter notebook, a ser encaminhado à professora Rosana Veroneze (rveroneze@gmail.com) até 23h59 do dia 6 de junho de 2019.

Questão 1)

Tomando o mesmo problema de classificação de dados da base MNIST, use o framework Keras, tendo o TensorFlow como *backend* e realize o treinamento de uma rede neural MLP. Busque inspiração em resultados já publicados na literatura e/ou adote o procedimento de tentativa-e-erro para definir, da melhor forma que você puder, o número de camadas intermediárias, o número de neurônios por camada, o algoritmo de treinamento, a taxa de *dropout* (onde for pertinente) e o número de épocas de treinamento. Procure trabalhar com a média de várias execuções (junto a cada configuração candidata) para se chegar a um índice de desempenho mais estável. Um código que pode servir de ponto de partida é fornecido a seguir, considerando 1 camada intermediária, 512 neurônios nesta camada intermediária, ADAM, ocorrência de dropout com taxa de 50% e 5 épocas de treinamento. A sua proposta deve ser capaz de superar o desempenho desta sugestão de ponto de partida e você deve descrever de forma objetiva o caminho trilhado até a sua configuração final de código para a rede neural MLP, assim como uma comparação de desempenho com a sugestão abaixo.

```
import tensorflow as tf
import os
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)

model_json = model.to_json()
json_file = open("model_MLP.json", "w")
json_file.write(model_json)
json_file.close()
model.save_weights("model_MLP.h5")
print("Model saved to disk")
os.getcwd()
```

Questão 2)

Tomando o mesmo problema de classificação de dados da base MNIST e novamente usando o framework Keras, tendo o TensorFlow como *backend*, realize o treinamento de uma rede neural com camadas convolucionais, usando maxpooling e dropout. Mais uma vez, é apresentada a seguir uma sugestão de código e de configuração de hiperparâmetros que pode ser tomada como ponto de partida. A sua proposta deve superar, em termos de desempenho médio, essa sugestão fornecida abaixo. Compare os resultados (em termos de taxa de acerto na classificação) com aqueles obtidos pelos três tipos de máquinas de aprendizado adotadas nas atividades anteriores (classificador linear e ELM, do EFC1, e MLP, do EFC2). Descreva de forma objetiva o caminho trilhado até a sua configuração final de código.

```
import tensorflow as tf
import os
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()

# reshape to be [samples][width][height][pixels]
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(32, kernel_size=(3, 3),
                                 activation='relu',
                                 input_shape=(28, 28, 1)))
model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)

model_json = model.to_json()
json_file = open("model_CNN.json", "w")
json_file.write(model_json)
json_file.close()
model.save_weights("model_CNN.h5")
print("Model saved to disk")
os.getcwd()
```

Questão 3)

Esta atividade envolve a execução completa de um notebook já proposto na literatura, com algumas alterações no código. Realize o treinamento de uma rede neural recorrente para predição de séries temporais (*stock price prediction*). O código é fornecido em [https://www.kaggle.com/raoulma/ny-stock-price-prediction-rnn-lstm-gru]. Execute todo o notebook, optando pelo bloco GRU em lugar do Basic RNN usado no código original. Você vai precisar baixar o arquivo [`prices-split-adjusted.csv`] do link [https://www.kaggle.com/dgawlik/nyse] e alterar o seu caminho no código do notebook, de acordo com o diretório de localização de sua escolha. Não se preocupe com os *warnings* que forem aparecendo durante a execução. Observação: Há questionamentos associados a este caso de estudo, como (1) o reescalamento empregando todo o conjunto de dados, incluindo o de teste, já que foi realizado antes da partição; (2) O fato da predição ser de um passo à frente e não de múltiplos passos à frente; (3) O fato de se tentar prever o valor da série temporal, e não a sua variação. Mas, mesmo assim, trata-se de um exemplo válido e didático, associado ao uso de uma rede neural recorrente.