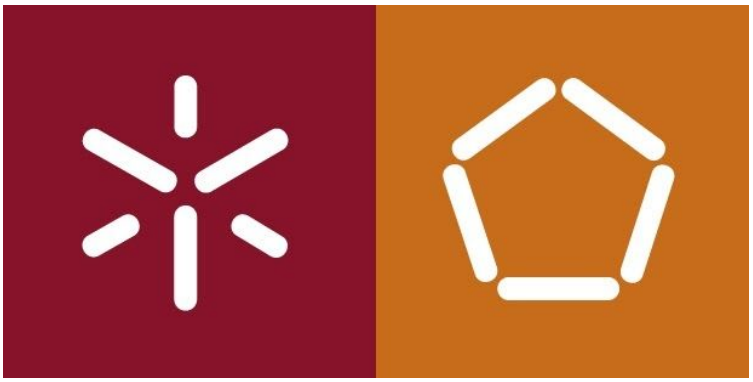


Universidade do Minho
Classificadores e Sistemas Conexionistas

Relatório Ficha 5

Este PDF, serve de auxílio à explicação da proposta da resolução da ficha 5.

Ficha 5

1ª Parte:

Implementar uma rede LSTM

Nesta primeira parte é pedido para implementar uma rede LSTM para resolver o Echo Sequence Prediction Problem. Analisar qual o impacto, na precisão do modelo, de treinar mais/menos épocas, aumentar/diminuir o número de neurónios e sujeitar o modelo a sequências com mais timesteps (10) e features (25):

Definir valores a utilizar no *one hot encoding*:

```
# generate a sequence of random numbers in [0, 99]
def generate_sequence(length=25):
    return [randint(0, 99) for _ in range(length)]
```

Definir métodos de *one hot encoding* e de *one hot decoding*:

```
# one hot encode sequence
def one_hot_encode(sequence, n_unique=100):
    encoding = list()
    for value in sequence:
        vector = [0 for _ in range(n_unique)]
        vector[value] = 1
        encoding.append(vector)
    return array(encoding)

# decode a one hot encoded string
def one_hot_decode(encoded_seq):
    return [argmax(vector) for vector in encoded_seq]
```

Preparação dos dados a fornecer à LSTM:

```
# prepare data for the LSTM
def get_data(n_in, n_out):
    # generate random sequence
    sequence = generate_sequence()
    # one hot encode
    encoded = one_hot_encode(sequence)
    # convert to X,y pairs
    X,y = to_supervised(encoded, n_in, n_out)
    return X,y
```

Definir a LSTM (modelo):

```
# define LSTM
n_in = 5
n_out = 5
encoded_length = 100
batch_size = 7
model = Sequential()
model.add(LSTM(20, batch_input_shape=(batch_size, n_in, encoded_length), return_sequences=True, stateful=True))
model.add(TimeDistributed(Dense(encoded_length, activation='softmax')))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
```

Treinar a LSTM:

```
# train LSTM
for epoch in range(500):
    # generate new random sequence
    X,y = get_data(n_in, n_out)
    # fit model for one epoch on this sequence
    model.fit(X, y, epochs=1, batch_size=batch_size, verbose=2, shuffle=False)
    model.reset_states()
```

Avaliar a LSTM:

```
# evaluate LSTM
X,y = get_data(n_in, n_out)
yhat = model.predict(X, batch_size=batch_size, verbose=0)
```

Realizar o *one hot decoding*:

```
# decode all pairs
for i in range(len(X)):
    print('Expected:', one_hot_decode(y[i]), 'Predicted', one_hot_decode(yhat[i]))
```

Resultados

Estes são os resultados de execução da rede LSTM:

```
Expected: [28, 25, 11, 14, 1] Predicted [28, 25, 11, 14, 1]
Expected: [25, 11, 14, 1, 4] Predicted [25, 11, 14, 1, 4]
Expected: [11, 14, 1, 4, 51] Predicted [11, 14, 1, 4, 51]
Expected: [14, 1, 4, 51, 71] Predicted [14, 1, 4, 51, 71]
Expected: [1, 4, 51, 71, 78] Predicted [1, 4, 51, 71, 78]
Expected: [4, 51, 71, 78, 70] Predicted [4, 51, 71, 78, 70]
Expected: [51, 71, 78, 70, 40] Predicted [51, 71, 78, 70, 40]
Expected: [71, 78, 70, 40, 91] Predicted [71, 78, 70, 40, 91]
Expected: [78, 70, 40, 91, 66] Predicted [78, 70, 40, 91, 66]
Expected: [70, 40, 91, 66, 47] Predicted [70, 40, 91, 66, 47]
Expected: [40, 91, 66, 47, 73] Predicted [40, 91, 66, 47, 73]
Expected: [91, 66, 47, 73, 70] Predicted [91, 66, 47, 73, 70]
Expected: [66, 47, 73, 70, 26] Predicted [66, 47, 73, 70, 26]
Expected: [47, 73, 70, 26, 75] Predicted [47, 73, 70, 26, 75]
Expected: [73, 70, 26, 75, 37] Predicted [73, 70, 26, 75, 37]
Expected: [70, 26, 75, 37, 2] Predicted [70, 26, 75, 37, 2]
Expected: [26, 75, 37, 2, 64] Predicted [26, 75, 37, 2, 64]
Expected: [75, 37, 2, 64, 71] Predicted [75, 37, 2, 64, 71]
Expected: [37, 2, 64, 71, 59] Predicted [37, 2, 64, 71, 59]
Expected: [2, 64, 71, 59, 24] Predicted [2, 64, 71, 59, 24]
Expected: [64, 71, 59, 24, 26] Predicted [64, 71, 59, 24, 26]
```

2ª Parte:

Implementar uma rede MLP ou LSTM - COVID-19

A segunda parte desta ficha prática consiste em uma rede MLP ou LSTM para fazer previsão do número de casos confirmados esperados a nível mundial nos próximos 7 dias. Este desenvolvimento é composto por 4 fase:

1) Descarregar a informação respeitante ao número de casos confirmados, recuperados e óbitos de COVID-19;
2) Analisar e explorar os dados em questão;
3) Enquadrar o problema do número de casos confirmados numa série temporal, i.e., para o dataset de casos confirmados, criar um novo dataset/dataframe com uma única coluna e em que cada linha corresponde a um dia. A coluna a criar deverá conter o total de casos confirmados a nível mundial;
4) Criar uma rede MLP ou LSTM para fazer previsão do número de casos confirmados esperados a nível mundial nos próximos 7 dias;

De maneira a obter sempre os dados mais atualizados, invés de se descarregar os dados para a máquina pessoal, é definido uma variável para cada tipo de dataset com o respetivo url e posteriormente é feita a sua leitura.

```
# Files
url_confirmed = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_19-covid-Confirmed.csv'
url_deaths = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_19-covid-Deaths.csv'
url_recovered = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_19-covid-Recovered.csv'


# Save data read
data_confirmed = load_data(url_confirmed)
data_deaths = load_data(url_deaths)
data_recovered = load_data(url_recovered)
```

De seguida, são preparados os dados para o formato pedido, ou seja, a data e a respetiva soma do número de casos.

```
'''
Prepare dataset
'''
def prepare_data(df):
    # Drop unwanted data
    drop_columns = ['Province/State', 'Country/Region', 'Lat', 'Long']
    df_aux = df.drop(columns=drop_columns, inplace=False)

    # Transpose column to row
    df_transpose = df_aux.transpose()

    # New dataframe
    new_columns = ['Cases']

    new_df = pd.DataFrame(columns=new_columns, index=df_transpose.index)
    new_df['Cases'] = df_transpose.sum(axis=1)

    return new_df
```

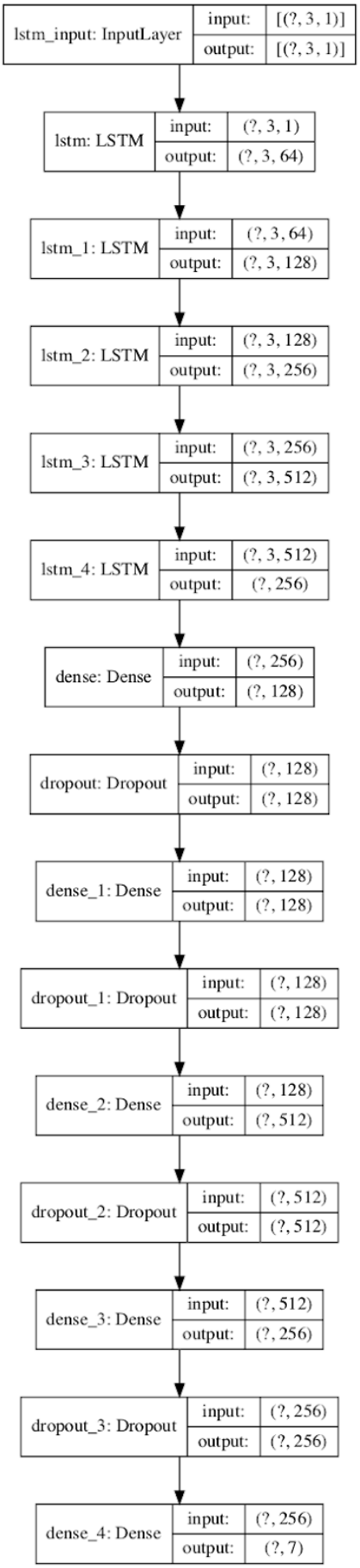
Feita a preparação dos dados, foi necessário normalizar os mesmo e fazer o teste dos mesmos através das seguintes funções.

```
'''
Normalize dataset
'''
def normalize_data(df, norm_range=(-1,1)):
    scaler = MinMaxScaler(feature_range=norm_range)

    # Normalize data
    normalized_df = scaler.fit_transform(df.values)

    columns = ['Cases']
    new_df = pd.DataFrame(normalized_df,columns=columns,index=df.index)

    return new_df, scaler

'''
Creating a supervised problem
'''
def to_supervised(df,timesteps):
    data = df.values
    X, y = list(), list()
    # Iterate over the trainnig_set to create X and y
    dataset_size = len(data)
    for curr_pos in range(dataset_size):
        # end of the input sequence is the curr_pos + the number of timesteps of the input sequence
        input_index = curr_pos + timesteps
        # end of the labels is the end of input_seq + 1
        label_index = input_index + 7
        # if we have enougth data for this sequence
        if label_index < dataset_size:
            X.append(data[curr_pos:input_index, :])
            y.append(data[input_index:label_index, 0])

    return np.array(X).astype('float32'), np.array(y).astype('float32')
```

Rede construída

A rede definida é um pouco complexa e permitiu-nos obter uma accuracy entre os 80% e os 85%.

```
'''
Buid the model
'''
def rmse(y_true, y_pred):
    return tf.keras.backend.sqrt(tf.keras.backend.mean(tf.keras.backend.square(y_true - y_pred)))


def build_model(timesteps, features, title, n_neurons=64, activation='tanh', dropout_rate=0.4):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.LSTM(n_neurons, return_sequences=True, input_shape=(timesteps,features)))

    model.add(tf.keras.layers.LSTM(int(n_neurons*2), return_sequences=True, dropout=dropout_rate))

    model.add(tf.keras.layers.LSTM(int(n_neurons*4), return_sequences=True, dropout=dropout_rate))

    model.add(tf.keras.layers.LSTM(int(n_neurons*8), return_sequences=True, dropout=dropout_rate))

    model.add(tf.keras.layers.LSTM(int(n_neurons*4), return_sequences=False, dropout=dropout_rate))

    model.add(tf.keras.layers.Dense(int(n_neurons*2), activation=activation))
    model.add(tf.keras.layers.Dropout(dropout_rate))

    model.add(tf.keras.layers.Dense(int(n_neurons*2), activation=activation))
    model.add(tf.keras.layers.Dropout(dropout_rate))

    model.add(tf.keras.layers.Dense(int(n_neurons*8), activation=activation))
    model.add(tf.keras.layers.Dropout(dropout_rate))

    model.add(tf.keras.layers.Dense(int(n_neurons*4), activation=activation))
    model.add(tf.keras.layers.Dropout(dropout_rate))


    model.add(tf.keras.layers.Dense(7, activation='linear'))

    model.compile(
            loss=rmse,
            optimizer=tf.keras.optimizers.Adam(),
            metrics=['mae',rmse]
        )

    print(model.summary())

    tf.keras.utils.plot_model(model,'Covid19-'+ title +'_Model.png',show_shapes=True)

    return model
```


lstm_input: InputLayer
input: [(?, 3, 1)]
output: [(?, 3, 1)]
lstm: LSTM
input: (?, 3, 1)
output: (?, 3, 64)
lstm_1: LSTM
input: (?, 3, 64)
output: (?, 3, 128)
lstm_2: LSTM
input: (?, 3, 128)
output: (?, 3, 256)
lstm_3: LSTM
input: (?, 3, 256)
output: (?, 3, 512)
lstm_4: LSTM
input: (?, 3, 512)
output: (?, 256)
dense: Dense
input: (?, 256)
output: (?, 128)
dropout: Dropout
input: (?, 128)
output: (?, 128)
dense_1: Dense
input: (?, 128)
output: (?, 128)
dropout_1: Dropout
input: (?, 128)
output: (?, 128)
dense_2: Dense
input: (?, 128)
output: (?, 512)
dropout_2: Dropout
input: (?, 512)
output: (?, 512)
dense_3: Dense
input: (?, 512)
output: (?, 256)
dropout_3: Dropout
input: (?, 256)
output: (?, 256)
dense_4: Dense
input: (?, 256)
output: (?, 7)

Treino e teste

Para testar foram definidas as seguintes variáveis com os respetivos valores que permitiram obter o resultado da previsão para os 7 dias.

```
'''
Vars
'''

timesteps = 3
univariate = 1
multisteps = 7
cv_splits = 3
epochs = 50
batch_size = 64
verbose = 1
```

Posto isto, define-se os *callbacks* e constrói-se o modelo. Depois da construção do modelo, realiza-se o *fit* do mesmo, ficando pronto para realizar as previsões.

```
'''
Experiment the model
'''
lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='loss', factor=0.2, patience=60, min_lr=0.00005)

'''
Fit the best model with all data and then predict 7 days
'''
model_confirm = build_model(timesteps, univariate, 'confirm')
#model_deaths = build_model(timesteps, univariate, 'deths')
#model_recovered = build_model(timesteps, univariate, 'recovered')

model_confirm.fit(X_confirm, y_confirm, epochs=epochs, batch_size=batch_size, shuffle=False, verbose=verbose, callbacks=[lr])
#model_deaths.fit(X_deaths, y_deaths, epochs=epochs, batch_size=batch_size, shuffle=False, verbose=verbose, callbacks=[lr])
#model_recovered.fit(X_recovered, y_recovered, epochs=epochs, batch_size=batch_size, shuffle=False, verbose=verbose, callbacks=[lr])
```

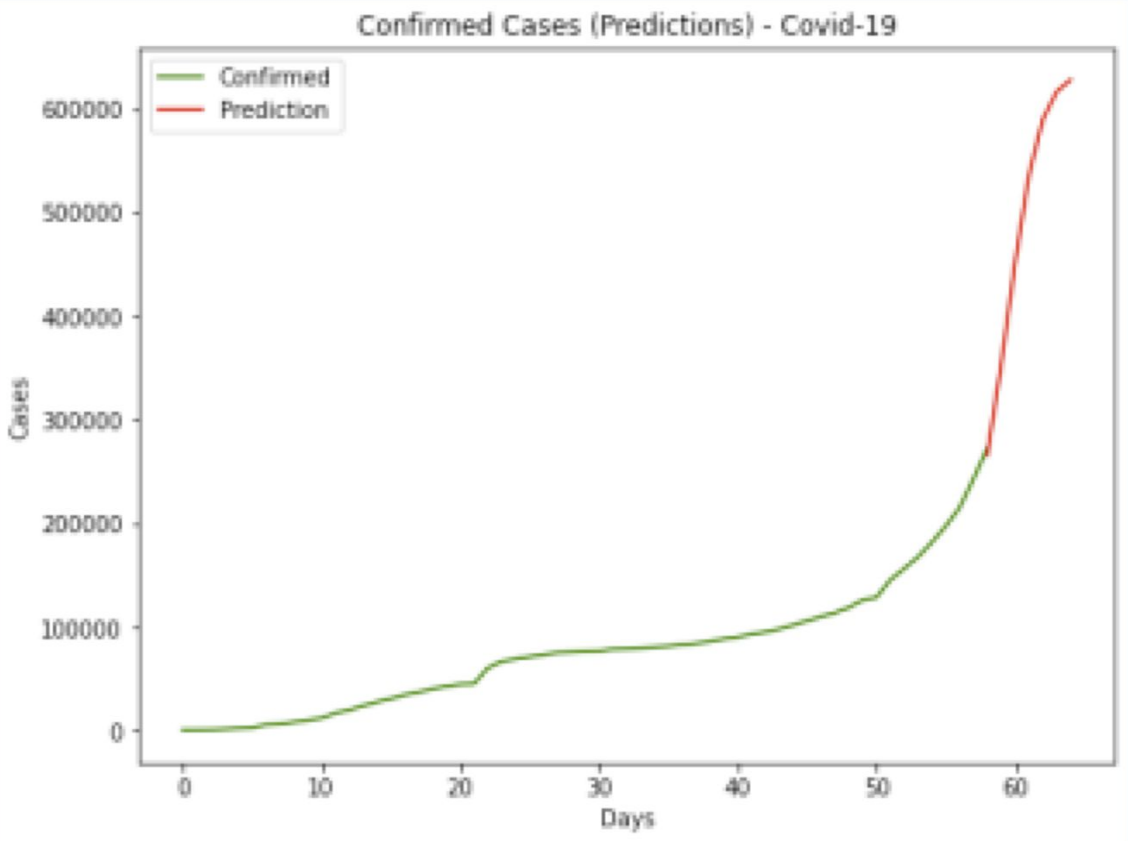
Para fazer a previsão é necessário fazer *forecast.* Como tal, está definida uma função *forecast* que irá fazer a previsão de cada dia dos próximos 7 dias.

```python
def forecast(model, df, timesteps, multisteps, scaler):
    input_seq = df[-timesteps:].values
    inp = input_seq
    predictions = list()
    for step in range(1,multisteps+1):
        inp = inp.reshape(1,timesteps,1)
        yhat = model.predict(inp, verbose=verbose)
        yhat_inversed = scaler.inverse_transform(yhat)
        predictions.append(yhat_inversed[0][0])
        inp = np.append(inp[0],yhat)
        inp = inp[-timesteps:]

    return predictions

predictions_confirm = forecast(model_confirm, normalized_confirm, timesteps, multisteps, scaler_confirm)
plot_forecast(total_confirm, predictions_confirm, 'Confirmed Cases (Predictions) - Covid-19')

#predictions_deaths = forecast(model_deaths, normalized_deaths, timesteps, multisteps, scaler_deaths)
#plot_forecast(total_deaths, predictions_deaths, 'Deaths Predictions - Covid-19')

#predictions_recovered = forecast(model_recovered, normalized_deaths, timesteps, multisteps, scaler_recovered)
#plot_forecast(total_recovered, predictions_recovered, 'Recovered Predictions - Covid-19')
```

Uma nota importante é o facto de ser possível realizar as previsões também para os casos de óbitos e de recuperados.

Resultados

Estes são os resultados de execução da rede construída. É importante referir que os resultados obtidos são referentes à previsão a partir do dia 21 de março e como se está a utilizar os dados mais atualizados, um novo teste/treino/execução pode fornecer um novo resultado.



Conclusão

Em suma, existiram várias dificuldades na realização desta ficha de trabalho, principalmente na construção do modelo da segunda parte. Foi complicado acertar o modelo que melhor se adaptava para o problema, de maneira a que fosse possível obter uma previsão satisfatória.

Quanto à primeira parte, possibilitou a obtenção de uma melhor compreensão quanto às LSTMs, o que ajudou para a construção do modelo da segunda parte.

Relativamente à segunda parte, para além do que já referi, a normalização dos dados e a previsão dos mesmos foram dois pontos que também transmitiram alguma dificuldade, uma vez que face ao segundo ponto foi complicado acertar com os valores para obter a melhor previsão possível.

Em suma, foi possível realizar tudo o que foi solicitado e adquirir bastantes conhecimentos nestas matérias.