LAB5_Tips

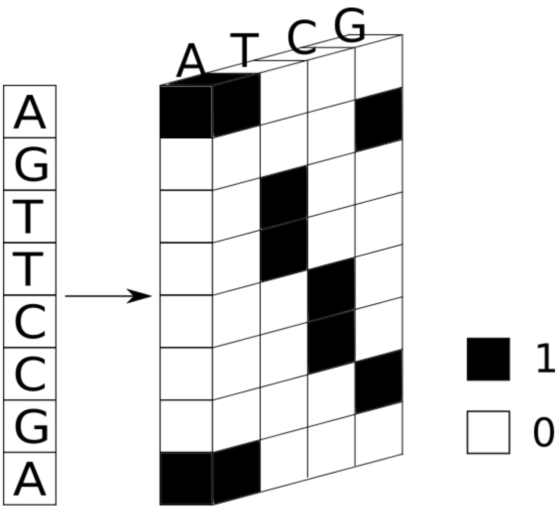
Install tensorflow and keras

Follow the instructions provided in ***Setup for labs*** to install the following libraries:

```
tensorflow
keras
```

One-hot encoding

The one hot encoding is one of the encoding methods that allows to transform sequences of letters (but also categorical data) into numbers, so that they can be processed by a classifier. Given a sequence M containing Z unique letters and defined as M_i the i^{th} letter of the sequence M, each M_i is replaced by a vector N of length Z in which each position corresponds to one of the letters Z. Defined N_j the j^{th} element of the vector N, N_j is equal to 1 if Z is equal to M_i, otherwise 0 if Z is different from M_i.



Mono-dimensional CNN example

```
import re
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import MaxPooling1D, Flatten, Dense, Conv1D
from keras import optimizers

############################################################################
# read file
############################################################################
dataset=[]
label=[]

with open('splice.data', 'r')   as fp:
    for line in fp:
```

```
        a = re.split(',', re.sub(r' |\n', '', line))
        label.append(a[0])
        dataset.append(a[2])

#########################################################################
# train and test split
#########################################################################
labelencoder_y = LabelBinarizer()
y = labelencoder_y.fit_transform(label)

X_train, X_test, y_train, y_test = train_test_split(dataset, y, test_size =
0.2, random_state = 0)

########################################################################
# train and test encoding
########################################################################
code = {'A':[1,0,0,0],'T':[0,1,0,0],'C':[0,0,1,0],'G':[0,0,0,1],
'N':[0.25,0.25,0.25,0.25], 'D':[0.33,0.33,0,0.33], 'R': [0.5,0,0,0.5],
'S':[0,0,0.5,0.5]}

x_train=np.zeros((len(X_train), len(X_train[0]), 4))

for i in range (len(X_train)):
    x_train[i,:,:] = (np.array([code[j] for j in X_train[i]]))


x_test=np.zeros((len(X_test), len(X_test[0]), 4))

for i in range (len(X_test)):
    x_test[i,:,:] = (np.array([code[j] for j in X_test[i]]))

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

#########################################################################
# classifier conv1D
#########################################################################

classifier = Sequential()

# Step 1 - Convolutions
classifier.add(Conv1D(128,3, input_shape = (60,4), activation = 'relu'))
classifier.add(MaxPooling1D(pool_size = 2))

classifier.add(Conv1D(128,3, activation = 'relu'))
classifier.add(MaxPooling1D(pool_size = 2))

classifier.add(Flatten())

# Step 2 - Full connection
classifier.add(Dense(activation = 'relu', units=100))
```

```
classifier.add(Dense(activation = 'softmax', units=3))

############################################################################
# Compile CNN
############################################################################
learning_rate= 0.01

n_epoch= 10

sgd = optimizers.SGD(lr=learning_rate, decay= (learning_rate/n_epoch),
nesterov=False)

classifier.compile(optimizer = 'sgd', loss = 'categorical_crossentropy',
metrics = ['accuracy'])

classifier.fit(x_train, y_train,
           batch_size=10, epochs=n_epoch, verbose=1, validation_split=0.1)

############################################################################
# Performance evaluation
############################################################################
classifier.evaluate(x_test, y_test, verbose= 1)

y_predicted = classifier.predict(x_test)

y_pred = labelencoder_y.inverse_transform(y_predicted)
y_true = labelencoder_y.inverse_transform(y_test)
confusion_matrix(y_true, y_pred)
```

EarlyStopping

A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training. You can pass a list of callbacks (as the keyword argument `callbacks`) to the `.fit()` method of the `Sequential` or `Model` classes. The relevant methods of the callbacks will then be called at each stage of the training. For example Early stopping callback stops training when a monitored quantity has stopped improving after a certain number of epochs. For details see https://keras.io/callbacks/

```
early_stopping_monitor = keras.callbacks.EarlyStopping(monitor='val_loss',
patience=3, mode='min')
# model_check point saves the model at each epoch
model_checkpoint = keras.callbacks.ModelCheckpoint('best_model.h5',
monitor='val_loss', mode='min', verbose=1, save_best_only=True)


training = model.fit(X_train, Y_train, batch_size=batch_size, epochs=epochs,
validation_split=0.1, verbose=1, callbacks=[early_stopping_monitor,
model_checkpoint])
```

LSTM model

```
def mono_directional_model():
model = Sequential()

# Adding the first LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True, input_shape =
(x_train.shape[1], x_train.shape[2])))
model.add(Dropout(0.2))

# Adding the second LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))

# Adding the third LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))

# Adding the fourt LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50))
model.add(Dropout(0.2))

#Adding the output layer
model.add(Dense(units = 3, activation='sigmoid'))

return model
```

Bidirectional LSTM

```
def bidirectional_model():
model = Sequential()

# Adding the firt LSTM layer and some Dropout regularisation
model.add(Bidirectional(LSTM(50, return_sequences=True),
                        input_shape=(x_train.shape[1], x_train.shape[2])))
# regressor.add(LSTM(units = 50, return_sequences = True, input_shape =
(x_train.shape[1], x_train.shape[2]))) #returnsequences set to False in the
last LSTM layer. False is the default parameter
model.add(Dropout(0.2))

# Adding the second LSTM layer and some Dropout regularisation
model.add(Bidirectional(LSTM(50, return_sequences=True)))
#regressor.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))

# Adding the third LSTM layer and some Dropout regularisation
#regressor.add(LSTM(units = 50, return_sequences = True))
model.add(Bidirectional(LSTM(50, return_sequences=True)))
model.add(Dropout(0.2))
```

```
# Adding the fourt LSTM layer and some Dropout regularisation
model.add(Bidirectional(LSTM(50)))
#regressor.add(LSTM(units = 50))
model.add(Dropout(0.2))

#Adding the output layer
model.add(Dense(units = 3, activation='sigmoid'))

return model
```