دانشگاه صنعتی امیرکبیر
دانشکده مهندسی کامپیوتر

# گزارش پروژه نهایی
# بیوانفورماتیک

استاد درس:
آقای دکتر زینلی

مولف:
محمدرضا صادقیان 9731121

تیرماه 1401

# فهرست

## مقدمه

در این پروژه ما قصد داریم با استفاده از انواع مختلف شبکه عصبی (MLP، CNN) به دسته‌بندی 6 نوع ویروس مختلف بپردازیم. ژنوم هر کدام از این ویروس ها توسط دنباله از نوکلئوتیدها نشان داده شده است که طول هر کدام متفاوت است. در کل ما 6 دسته مختلف ویروس با نام‌های Class 1 تا Class 6 داریم که شامل سه دسته، Train، Development و Test می‌باشد که در فایل‌های CSV ذخیره شده اند. در کل هر دو نوع شبکه عصبی ای که من در این پروژه استفاده کرده ام Feedforward هستند که آنها را با استفاده از کتابخانه Tensorflow، که برای Google است، پیاده سازی کردم. در شبکه‌های عصبی Feedforward خروجی هر لایه به عنوان ورودی به لایه بعدی داده می‌شود و در نهایت با استفاده از Backpropagation وزن و بایاس لایه‌ها آپدیت می‌شود. (البته تمام این کارها توسط کتابخانه Tensorflow در پس زمینه انجام می‌شود.)

**مقدمه : بهترین نتیجه من با مدل MLP بدست آمده است.**

## شرح پروژه:

## بخش اول: کاهش طول

همان طور که در دستورکار پروژه شرح داده شده بود و لینک آموزشی قرار داده شده بود برای اینکه بتوانیم طول ورودی‌های شبکه عصبی را یکسان کنیم باید از تکنیک K_mers استفاده کنیم که در این روش در ابتدا تمامی دنباله‌های kتایی که می‌توان با حروف dna ساخت را ایجاد می‌کنیم و در نهایت هر رشته را kتا kتا چک می‌کنیم و یک بردار ایجاد می‌کنیم به طول تعداد زیررشته ها و در هر ایندکس تعداد تکرار اون زیررشته متناظر را وارد می‌کنیم. من برای راحتی کار تمام زیررشته ها را در یک دیکشنری قرار داده ام و بعد با استفاده از آنها به سادگی به ایندکس بردار آنها دسترسی خواهم داشت.

```
# K_Mers
k = 7
dnaLetters = ['A', 'T', 'G', 'C']
allSubString = []

def generateString(dnaLetters, prefix, k):

  n = len(dnaLetters)
  if (k == 0):
    allSubString.append(prefix)
    return

  for l in dnaLetters:
    newPrefix = prefix + l
    generateString(dnaLetters, newPrefix, k-1)

generateString(dnaLetters, "", k)
subDict =  {allSubString[i] : i for i in range(0, len(allSubString))}
```

همانطور که مشاهده می‌کنید با استفاده از یک تابع بازگشتی تمامی زیررشته‌های موجود را ایجاد می‌کنیم و یک دیکشنری برای دسترسی ساده تر ایجاد می‌کنیم.

```
def generate_X(data_set):
  X_input = np.zeros((len(data_set), len(allSubString)))
  for i in range(len(data_set)):
    seq = data_set[i]
    for j in range(0, len(seq) - k + 1):
      X_input[i][subDict[seq[j:j+k]]] += 1

  X_input = X_input.astype('float32')
  return X_input
```

بعد از ایجاد دیکشنری در تابع generate_X در ابتدا یک آرایه دو بعدی به تعداد زیررشته‌های موجود و تعداد رشته‌های دیتابیس ایجاد می‌کنیم و سپس با طی کردن هر رشته بردار موردنظر آن رشته را ایجاد می‌کنیم و در نهایت تایپ آن را برای شبکه عصبی تصحیح می‌کنیم و سپس ما به بردار ورودی مورد نیاز برای شبکه عصبی امان می‌رسیم.

## بخش دوم: جداسازی لیبل‌ها

در این بخش ما با استفاده از یک تابع لامبدا لیبل‌ها را به مقدار عددی اشان تبدیل می‌کنیم و با استفاده از To_categotical ابعاد لیبل را فیکس می‌کنیم تا بعد نامپای آن خالی نباشد.

```
[ ]  def generate_Y(data_set):
       Y_input = data_set.apply(lambda x : int(x[5]) - 1).to_numpy()
       Y_input = to_categorical(Y_input)
       return Y_input
```

## بخش سوم: معماری مدل MLP

```
[8]  MLP_model = Sequential()
     MLP_model.add(Dense(80, input_dim=len(allSubString), activation='relu'))
     # model.add(Dropout(0.05))
     MLP_model.add(Dense(50, activation = 'relu'))
     # model.add(Dropout(0.05))
     MLP_model.add(Dense(6, activation='softmax'))
```

```
MLP_model.compile(loss='categorical_crossentropy', optimizer="adam", metrics=['accuracy'])
history = MLP_model.fit(X_train, Y_train, epochs=20, batch_size=50, verbose=1, validation_split=0.2)
```

```
MLP_model.summary()
result = list(MLP_model.predict(X_test).argmax(axis=-1) + 1)
print(result)
print("\n-------------------------------------------------------------------------")
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 80)                1310800

 dense_1 (Dense)             (None, 50)                4050

 dense_2 (Dense)             (None, 6)                 306

=================================================================
Total params: 1,315,156
Trainable params: 1,315,156
Non-trainable params: 0
```

در این بخش در ابتدا با استفاده از تابع Sequential از کتابخانه keras یک مدل ایجاد می‌کنیم و سه لایه که دو لایه آن پنهان هستند با تعداد 80، 50 نورون به ترتیب به آن پاس می‌دهیم. لایه آخر نیز 6 نورون دارد که هر کدام نماینده هر یک از 6 کلاس هستند و تابع فعال سازی آن Softmax است تا در نهایت ما یک توضیح احتمال به عنوان خروجی داشته باشیم و دو لایه پنهان تابع فعالسازشان Relu است که با توجه به تست ها بهترین عملکرد را داشت.

این تعداد نورون ها کاملا به صورت دستی و با تست کردن های متعدد به دست آمده و این تقریبا حداقل مقداری بود که می‌توان برای آن اندازه گرفت. برای اینکه از overfit فرار کنیم من دو تا Dropout قرار داده ام که به احتمال 0.05 در هر مرحله نورون ها را غیرفعال

می‌کند تا از overfit جلوگیری کند ولی برای تست کوئرا بهترین حالت زمانی بود که آنها کامنت شده بودند ولی قطعا در تجربه محیطی و تست های بزرگتر عملکرد مدل با آنها بهتر است. برای دیگر تنظیمات مدل هم من کاملا از سعی کردم از معروف ترین توابع استفاده کنم و مقدار batch-size و epoch نیز با تست‌های زیاد بدست آمدند.

## بخش چهارم: معماری مدل CNN

```
CNN_model = Sequential([

    Conv2D(32, (3, 3), padding='same', kernel_initializer='he_uniform', activation='relu', input_shape=(32, 32, 1)),
    Conv2D(32, (3, 3), padding='same', kernel_initializer='he_uniform', activation='relu'),
    MaxPool2D((2, 2)),
    Dropout(0.15),

    Conv2D(64, (3, 3), padding='same', kernel_initializer='he_uniform', activation='relu'),
    Conv2D(64, (3, 3), padding='same', kernel_initializer='he_uniform', activation='relu'),
    MaxPool2D((2, 2)),
    Dropout(0.15),

    Conv2D(128, (3, 3), padding='same', kernel_initializer='he_uniform', activation='relu'),
    Conv2D(128, (3, 3), padding='same', kernel_initializer='he_uniform', activation='relu'),
    MaxPool2D((2, 2)),
    Dropout(0.15),

    Flatten(),
    Dense(128, kernel_initializer='he_uniform', activation='relu'),
    Dropout(0.15),
    Dense(6, kernel_initializer='he_uniform', activation='softmax')
])
```

در این بخش مدل CNN را پیاده سازی کرده ام و تمامی این اعداد به صورت دستی و تقریبا مشابه با یکی از پروژه‌های قبلی خودم در پردازش تصویر ست کردم که به نتایج خوبی نیز رسیدم.

## بخش پنجم: مقایسه نتایج

در جفت روش‌ها بعد از چند epoch جفت مدل‌ها به دقت یک می‌رسند که با توجه به آنکه مدل MLP من تعداد پارامترهای کمتری داشت. (به طور کلی CNN معمولا تعداد پارامترهای خیلی کمتری دارد ولی اینجا من خیلی نرسیدم با توجه به وقت پروژه هایپر

پارامترهای CNN را دستکاری کنم و احتمال خیلی زیاد در CNN با پارامترهای کمتری به نتایج یکسان میرسیم.)

## بخش ششم: خروجی‌ها

در نهایت آرایه Result را در فایل دیگر سیو کرده و در کوئرا آن را آپلود می‌کنم

```
MLP_model.summary()
print("\n-----------------------------------------------------------------")
test = MLP_model.evaluate(X_dev, Y_dev, verbose=1)
print(MLP_model.predict(X_dev))
print("\n-----------------------------------------------------------------")
result = list(MLP_model.predict(X_test).argmax(axis=-1) + 1)
print(result)
```

و نتایج به صورت زیر می‌باشد.

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 80)                1310800

 dense_10 (Dense)            (None, 50)                4050

 dense_11 (Dense)            (None, 6)                 306

=================================================================
Total params: 1,315,156
Trainable params: 1,315,156
Non-trainable params: 0
_________________________________________________________________


6/6 [==============================] - 0s 5ms/step - loss: 9.5897e-07 - accuracy: 1.0000

-----------------------------------------------------------------
[[5.4295820e-06 1.0355825e-09 1.1050500e-12 1.2913298e-06 9.9999166e-01
  1.6818009e-06]
 [6.4401995e-09 1.0000000e+00 1.3589282e-18 7.4523461e-24 8.4879861e-17
  9.8136676e-21]
 [5.1949609e-09 1.0000000e+00 1.1868981e-18 6.1522008e-24 5.9223154e-17
  8.6107886e-21]
 ...
 [1.0000000e+00 9.1576551e-26 1.5201584e-32 8.2676806e-38 1.7346751e-21
  6.0038448e-38]
 [1.0000000e+00 5.9208424e-26 1.7546645e-32 6.7132745e-38 1.2920846e-21
  5.0830112e-38]
 [1.0000000e+00 6.9120338e-26 1.8066243e-32 7.1987763e-38 1.3606172e-21
  5.6288276e-38]]

-----------------------------------------------------------------
[5, 5, 3, 1, 6, 1, 6, 3, 4, 1, 4, 3, 2, 4, 4, 3, 6, 3, 3, 3, 4, 1, 5, 6, 1, 5, 3, 3, 3, 3, 1, 3, 6, 1,
```

با تشکر از زحمات شما در طول ترم - محمدرضا صادقیان 9731121