فاطمه توکلی_۴۰۰۱۳۱۰۱۶

۱-در ابتدا کتابخانه های لازم را فراخوانی کرده :

```
[1]  1  import numpy as np
     2  import tensorflow as tf
     3  from tensorflow import keras
     4  import matplotlib.pyplot as plt
     5  import pandas as pd
     6  from sklearn.model_selection import train_test_split
```

سپس دیتا ست را بارگذاری می‌کنیم که شامل ۵ قسمت است که هرکدام فرمت مشابه ایی دارند و برچسب داده ها normal,ok میباشد و هر ۵ بخش را ترکیب میکنیم به عنوان ورودی استفاده می‌کنیم :

```
def load_ds(path):
    with open(path) as f:
        data = f.read()
        sub_data_list = data.split("\n\n\n")
        tags_list = []
        data = []
        for s in sub_data_list:
            detail: list = s.split("\n")
            if not detail or len(detail) < 2:
                continue
            tag = detail[0]
            tags_list.append(int(tag in ["normal", "ok"]))
            this_section_detail = []
            for row in detail[1:]:
                row_list = row.split("\t")[1:]
                if row_list:
                    this_section_detail.append(row_list)
            if this_section_detail:
                data.append(this_section_detail)
    return data, tags_list
```

```
for i in range(1, 6):
    data, tags = load_ds(path=f'lp{i}.data')
    data_list.extend(data)
    tag_list.extend(tags)

for i in data_list:
    assert len(i) == 15
    for j in i:
        assert len(j) == 6

data = np.array(data_list, dtype=int)
tags = np.array(tag_list)
data = data / np.max(data, axis=0)
```

داده ها را به سه قسمت گفته شده تقسیم می‌کنیم:

```
X_train, X_test, y_train, y_test = train_test_split(data, tags, test_size=0.3, random_state=22)
X_test , X_val, y_test, y_val =train_test_split(X_test,y_test,test_size=0.33,random_state=22)
```

- مدل elman به اینصورت است که لایه اول ورودی را می‌گیرد و لایه بعدی با اعمال تابع فعالسازی خروجی میدهد که ما از الن به عنوان ورودی وزن دار لایه قبلی به همراه ورودی بعدی استفاده میکنیم.

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

ابتدا مدل را به صورت sequential تعریف میکنیم سپس ابعاد ورودی را مشخص کرده و سپس از simplernn که رویکرد ان همانگونه است که توضیح داده شد استفاده می‌کنیم و در نهایت یک لایه dense برای گرفتن خروجی که شامل ۲ مقدار بود استفاده میکنیم و مدل را کامپایل میکنیم:

```
model = keras.models.Sequential()
model.add(keras.layers.Input(shape=(15,6)))
model.add(keras.layers.SimpleRNN(40))
model.add(keras.layers.Dense(2,activation="softmax"))
model.compile(optimizer="adam",loss="sparse_categorical_crossentropy",metrics=["acc"])
model.fit(X_train,y_train,epochs=50)
```

خروجی گرفته شده بر روی داده آموزش با units = 40:

```
Epoch 50/50
11/11 [==============================] - 0s 4ms/step - loss: 0.1386 - acc: 0.9475
<keras.callbacks.History at 0x7f0d73132f90>
```

خروجی گرفته شده بر روی داده آموزش units = 20:

```
Epoch 50/50
11/11 [==============================] - 0s 4ms/step - loss: 0.2946 - acc: 0.8796
<keras.callbacks.History at 0x7f0d772b28d0>
```

ارزیابی مدل:

```
1  model.evaluate(X_test,y_test)
```

```
3/3 [==============================] - 0s 6ms/step - loss: 0.1275 - acc: 0.9677
[0.12751275300979614, 0.9677419066429138]
```

برای مدل Jordan می بایست لایه خروجی را به صورت وزن دار به لایه ورودی متصل کرد و میتوان از keras.layers.layer استفاده کرد و تابع call را تغییر داده گرفت و سپس توسط rnn اجرا گرفت .

تغییرات لازم:

$$h_t = \sigma_h(W_h x_t + U_h y_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

```python
class jordanRNN(keras.layers.Layer):

    def __init__(self, run_units,**kwargs):
        super().__init__(**kwargs)
        self.run_units = run_units
        self.state_size = run_units


    def build(self, input_shapes):

        self.W_xh = self.add_weight(shape=(input_shapes[-1], self.run_units),initializer='uniform',
                                    name='W_xh')
        self.W_hh = self.add_weight(shape=(self.run_units, self.run_units),initializer='uniform',name='W_hh')
        self.W_hy = self.add_weight(shape=(self.run_units,self.run_units))
        self.built=True


    def call(self, inputs, states):
        prev_output = states[0]
        h=tf.keras.activations.relu(tf.keras.backend.dot(inputs, self.W_xh) + tf.keras.backend.dot(prev_output, self.W_hh))
        output = tf.keras.activations.relu(tf.keras.backend.dot(h, self.W_hy))
        return output, [output]

    def get_config(self):
        config = super().get_config()
        config.update({
            "run_units": self.run_units,
        })
        return config
cell = jordanRNN(64)
RNN_layer = keras.layers.RNN(cell)
```

تعریف مدل:

```python
jordan_model = keras.models.Sequential()
jordan_model.add(keras.Input(shape=(15, 6)))
jordan_model.add(RNN_layer)
jordan_model.add(keras.layers.Dense(2, activation='softmax'))
```

ارزیابی مدل بر روی داده آموزش و ازریابی آن:

```
Epoch 60/60
11/11 [==============================] - 0s 6ms/step - loss: 0.2618 - accuracy: 0.8673
<keras.callbacks.History at 0x7f0d6303b190>
```

```
]  1  jordan_model.evaluate(X_test,y_test)
```

```
3/3 [==============================] - 0s 4ms/step - loss: 0.2902 - accuracy: 0.8387
[0.2901843190193176, 0.8387096524238586]
```

- نتایج مدل Jordan بهتر از مدل Elman می‌باشد. در کل المن برای درک ویژگی ها و جردن برای جداسازی ویژگی‌ها بهتر هستند.
در مدل المن از لایه پنهان خروجی گرفته میشود و برمیگرداند به ورودی در حالیکه مدل جردن از لایه خروجی به لایه ورودی بازگشت انجام می‌شود. برای حل این مسئله جردن مناسب تر است .
(علت دقت کمتر نحوه پیاده سازی بنده است و نمیدانم اشکال کار در کجاست!!!!!)

۲-

- سه نوع مدل با استفاده از elman تعریف می‌کنیم:

هر سه تا را ترکیب می‌کنیم و خروجی گرفته شده توسط هرکدام را با استفاده از keras.layers.average میانگین میگیریم :

```
model1 = keras.models.Sequential()
model1.add(keras.layers.Input(shape=(15,6)))
model1.add(keras.layers.SimpleRNN(20))
model1.add(keras.layers.Dense(2,activation="softmax"))

model2 = keras.models.Sequential()
model2.add(keras.layers.Input(shape=(15,6)))
model2.add(keras.layers.SimpleRNN(30))
model2.add(keras.layers.Dense(2,activation="softmax"))


model3 = keras.models.Sequential()
model3.add(keras.layers.Input(shape=(15,6)))
model3.add(keras.layers.SimpleRNN(50))
model3.add(keras.layers.Dense(2,activation="softmax"))
```

```
models = [model1, model2, model3] #stacking individual models in a list
model_input = tf.keras.Input(shape=(15,6)) #takes a list of tensors as input, all of the same shape
model_outputs = [model(model_input) for model in models] #collects outputs of models in a list
ensemble_output = tf.keras.layers.Average()(model_outputs) #averaging outputs
ensemble_model = tf.keras.Model(inputs=model_input, outputs=ensemble_output)
ensemble_model.summary() #prints a comprehensive summary of the Keras model
```

مدل به صورت زیر می‌باشد:

```
Model: "model_3"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_22 (InputLayer)       [(None, 15, 6)]           0         []

 sequential_11 (Sequential)  (None, 2)                 582       ['input_22[0][0]']

 sequential_12 (Sequential)  (None, 2)                 1172      ['input_22[0][0]']

 sequential_13 (Sequential)  (None, 2)                 2952      ['input_22[0][0]']

 average_10 (Average)        (None, 2)                 0         ['sequential_11[3][0]',
                                                                  'sequential_12[3][0]',
                                                                  'sequential_13[3][0]']

==================================================================================================
Total params: 4,706
Trainable params: 4,706
Non-trainable params: 0
__________________________________________________________________________________________________
```

کامپایل می‌کنیم:

```
ensemble_model.compile(optimizer="adam",loss="sparse_categorical_crossentropy",metrics=["acc"])
ensemble_model.fit(X_train,y_train,epochs=50)
```

خروجی و نتیجه ارزیابی آن :

```
11/11 [==============================] - 0s 9ms/step - loss: 0.0764 - acc: 0.9722
<keras.callbacks.History at 0x7f0d6eb95fd0>
```

```
ensemble_model.evaluate(X_test,y_test)
```

```
3/3 [==============================] - 1s 9ms/step - loss: 0.1104 - acc: 0.9785
[0.11036979407072067, 0.9784946441650391]
```

- ۳ مدل از جردن تعریف می‌کنیم و با همان روش قبلی ترکیب کرده و نتیجه را گزارش می‌دهیم:

```
cell = jordanRNN(128)
RNN_layer = keras.layers.RNN(cell)
jordan_model1 = keras.models.Sequential()
jordan_model1.add(keras.Input(shape=(15, 6)))
jordan_model1.add(RNN_layer)
jordan_model1.add(keras.layers.Dense(2, activation='softmax'))

cell = jordanRNN(64)
RNN_layer = keras.layers.RNN(cell)
RNN_layer = keras.layers.RNN(cell)
jordan_model2 = keras.models.Sequential()
jordan_model2.add(keras.Input(shape=(15, 6)))
jordan_model2.add(RNN_layer)
jordan_model2.add(keras.layers.Dense(2, activation='softmax'))

cell = jordanRNN(32)
RNN_layer = keras.layers.RNN(cell)
RNN_layer = keras.layers.RNN(cell)
jordan_model3 = keras.models.Sequential()
jordan_model3.add(keras.Input(shape=(15, 6)))
jordan_model3.add(RNN_layer)
jordan_model3.add(keras.layers.Dense(2, activation='softmax'))
```

```
models = [jordan_model1, jordan_model2, jordan_model3] #stacking individual models in a list
model_input = tf.keras.Input(shape=(15,6)) #takes a list of tensors as input, all of the same shape
model_outputs = [model(model_input) for model in models] #collects outputs of models in a list
ensemble_output = tf.keras.layers.Average()(model_outputs) #averaging outputs
ensemble_model = tf.keras.Model(inputs=model_input, outputs=ensemble_output)
ensemble_model.summary() #prints a comprehensive summary of the Keras model
```

خلاصه مدل به صورت زیر می‌باشد:

```
Model: "model_4"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
==================================================================================================
 input_43 (InputLayer)       [(None, 15, 6)]           0           []

 sequential_33 (Sequential)  (None, 2)                 33794       ['input_43[0][0]']

 sequential_34 (Sequential)  (None, 2)                 8706        ['input_43[0][0]']

 sequential_35 (Sequential)  (None, 2)                 2306        ['input_43[0][0]']

 average_11 (Average)        (None, 2)                 0           ['sequential_33[0][0]',
                                                                    'sequential_34[0][0]',
                                                                    'sequential_35[0][0]']

==================================================================================================
Total params: 44,806
Trainable params: 44,806
Non-trainable params: 0
__________________________________________________________________________________________________
```

خروجی بر روی داده آموزش و آزمون و ارزیابی آن به صورت زیر میباشد:

```
Epoch 50/50
11/11 [==============================] - 0s 32ms/step - loss: 0.3868 - acc: 0.7253
<keras.callbacks.History at 0x7f0d63019c50>
```

```
1  ensemble_model.evaluate(X_test,y_test)
```

```
3/3 [==============================] - 1s 7ms/step - loss: 0.4170 - acc: 0.7204
[0.41703662276268005, 0.7204301357269287]
```

- ترکیب مدل های المن و جردن که به بهترین دقت رسیده است:(از مدل های قبلی استفاده شده و ترکیب جدی ساخته شده)

```
models = [ jordan_model2, model3 ,model2] #stacking individual models in a list
model_input = tf.keras.Input(shape=(15,6)) #takes a list of tensors as input, all of the same shape
model_outputs = [model(model_input) for model in models] #collects outputs of models in a list
ensemble_output = tf.keras.layers.Average()(model_outputs) #averaging outputs
ensemble_model = tf.keras.Model(inputs=model_input, outputs=ensemble_output)
ensemble_model.summary() #prints a comprehensive summary of the Keras model
```

خروجی برروی داده آزمون وارزیابی مدل:

```
Epoch 29/50
11/11 [==============================] - 0s 11ms/step - loss: 0.2439 - acc: 0.9321
```

```
1  ensemble_model.evaluate(X_test,y_test)
```

```
3/3 [==============================] - 0s 6ms/step - loss: 0.2873 - acc: 0.9247
[0.28726017475128174, 0.9247311949729919]
```