فاز دوم پروژه

پیشبینی رده سنی بر اساس خلاصه فیلم

پردازش زبان و گفتار

استاد درس

دکتر مینایی

دانشجویان

امیرحسین احمدی – ملیکا احمدی رنجبر

بهار ۱۴۰۱

فهرست مطالب

بخش اول

در بخش اول با استفاده از Bert Model و داده‌ها جمع‌آوری شده از فاز اول پروژه، باید جملات جدیدی را تولید می‌کردیم. برای انجام این کار در ابتدا داده‌های Clean شده را با ساختار Dataset و با استفاده از کتابخانه خاص Load می‌کنیم و به شکل زیر آن را خواهیم داشت:

```
from datasets import load_dataset
dataset = load_dataset('csv', data_files={'train': base_dir + 'data/cleaned/data.csv'})
```

داده کنونی به شکل زیر است:

```
dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['Unnamed: 0', 'Title', 'MPA', 'Plot', 'Normalized_Plot'],
        num_rows: 27401
    })
})
```

```
dataset['train'][0]
```

```
{'MPA': 'G',
 'Normalized_Plot': "['lion', 'prince', 'simba', 'father', 'targeted', 'bitter', 'uncle', 'want', 'ascend', 'throne']",
 'Plot': 'Lion prince Simba and his father are targeted by his bitter uncle, who wants to ascend the throne himself.',
 'Title': 'The Lion King',
 'Unnamed: 0': 0}
```

سپس با توجه به تعداد کلاس‌هایی که برای این Classification داریم یک آرایه درست می‌کنیم و Indexing آن را نیز انجام می‌دهیم:

```
labels = ['G', 'PG', 'PG-13', 'R']
id2label = {idx:label for idx, label in enumerate(labels)}
label2id = {label:idx for idx, label in enumerate(labels)}
labels
```

```
['G', 'PG', 'PG-13', 'R']
```

سپس با استفاده از کتابخانه Transformers طبق سوال، Bert Model را خواهیم داشت و از Tokenizer آن استفاده می‌کنیم که به ما یک Vector باز می‌گرداند. ورودی آن نیز تمامی جملات ما به شکل String خواهد بود.

```
from transformers import AutoTokenizer
import numpy as np

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
```

```
Downloading: 100% 28.0/28.0 [00:00<00:00, 293B/s]
Downloading: 100% 570/570 [00:00<00:00, 6.70kB/s]
Downloading: 100% 226k/226k [00:00<00:00, 286kB/s]
Downloading: 100% 455k/455k [00:01<00:00, 515kB/s]
```

سپس مرحله پیش‌پردازش را بر روی داده‌ها داریم تا بتوانیم از آن برای Fine Tune گردد این مدل استفاده کنیم. خروجی پیش‌پردازش ما یک Embedding خواهد بود.

```python
def preprocess_data(examples):
    for i in range(len(examples['Normalized_Plot'])):
        examples['Normalized_Plot'][i] = ' '.join(ast.literal_eval(examples['Normalized_Plot'][i]))


    labels_batch_array = []
    # take a batch of texts
    text = examples["Normalized_Plot"]
    # encode them
    encoding = tokenizer(text, padding="max_length", truncation=True, max_length=128)
    rate = examples['MPA']
    for i in range(len(text)):
        labels_batch = {'G': False, 'PG': False, 'PG-13': False, 'R': False}
        labels_batch[rate[i]] = True
        labels_batch_array.append(labels_batch)

    # print(labels_batch_array)

    # create numpy array of shape (batch_size, num_labels)
    labels_matrix = np.zeros((len(text), len(labels)))
    # fill numpy array
    for i in range(len(text)):
        temp_dic = labels_batch_array[i]
        for idx, label in enumerate(labels):
            # print(temp_dic)
            labels_matrix[i, idx] = temp_dic[label]

    encoding["labels"] = labels_matrix.tolist()

    return encoding

encoded_dataset = dataset.map(preprocess_data, batched=True, remove_columns=dataset['train'].column_names)
```

این مرحله یکی از مراحل مهم است. ورودی به این تابع شامل تمامی داده‌های ما خواهدبود که به شکل Batch وارد می‌شوند. بنابراین پیاده‌سازی به شکل Vectorized است. برای مثال Examples را که می‌گیریم و Nomalized_Plot آن را می‌خوانیم به ما یک Array برمی‌گرداند. پس از آن، Rate و یا همان MPA, Label را نیز می‌خوانیم. با توجه به همان Text گرفته از Tokenizerای که با Bert تعریف کردیم استفاده می‌کنیم و Padding نیز به آن می‌دهیم تا Vector هایی که برمی‌گرداند با توجه به طول مختلف ورودی‌ها اندازه یکسانی داشته‌باشند. سپس از یک آرایه Dictionary استفاده می‌کنیم و برای هر کدام از این داده‌ها مشخص می‌کنیم کدام دسته برای آن مقدار True دارد. در ادامه مثالی را بررسی خواهیم کرد. سپس با توجه به این آرایه از دیکشنری‌ها یک ماتریس برای تمامی ورودی‌ها و Labelها میسازیم. اگر فرض کنیم که ورودی‌ها اندازه Batch = 32 داشته‌باشند و ۴ دسته داریم بنابراین ماتریس ۳۲*۴ خواهد بود که مقادیر ۰ و ۱ دارند تا کلاس درست را مشخص کنند. در انتها نیز این Encoding را باز می‌گردانیم.

```python
encoded_dataset['train'][0].keys()
```
```
dict_keys(['input_ids', 'token_type_ids', 'attention_mask', 'labels'])
```
```python
tokenizer.decode(encoded_dataset['train'][0]['input_ids'])
```
```
'[CLS] lion prince simba father targeted bitter uncle want ascend throne [SEP] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [P
AD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PA
D] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]'
```
```python
encoded_dataset['train'][0]['labels']
```
```
[1.0, 0.0, 0.0, 0.0]
```
```python
[id2label[i] for i, label in enumerate(encoded_dataset['train'][0]['labels']) if label == 1.0]
```
```
['G']
```

پس از آماده کردن این Embedding ها، دوباره با توجه به Bert یک مدل برای Sequence Classification انتخاب می‌کنیم و پارامترهای مختلف آن را نیز تعیین می‌کنیم:

```
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased",
                                                           problem_type="multi_label_classification",
                                                           num_labels=len(labels),
                                                           id2label=id2label,
                                                           label2id=label2id)
```

```
from transformers import TrainingArguments, Trainer

args = TrainingArguments(
    f"bert-finetuned-sem_eval-english",
    evaluation_strategy = "epoch",
    save_strategy = "epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    num_train_epochs=5,
    weight_decay=0.01,
    load_best_model_at_end=True,
    metric_for_best_model='f1',
)
```

پارامترهایی نظیر Learning_Rate و ... در بالا مشاهده می‌شوند.

برای دسته‌بندی در نظر گرفته شده که می‌تواند Multi-Label باشد و یک فیلم دارای چند MPA باشد. اما به طور منطقی باشد یک حالت باشد. باز با اینکه Multi-Label حالتی کلی‌تر است آن را در نظر گرفتیم. پس از آن نیز برای محاسبه مقادیر و Metricهایی مثل Accuracy یک تابع جداگانه تعریف می‌کنیم:

```
def compute_metrics(p: EvalPrediction):
    preds = p.predictions[0] if isinstance(p.predictions,
            tuple) else p.predictions
    result = multi_label_metrics(
        predictions=preds,
        labels=p.label_ids)
    return result
```

```
def multi_label_metrics(predictions, labels, threshold=0.5):
    # first, apply sigmoid on predictions which are of shape (batch_size, num_labels)
    sigmoid = torch.nn.Sigmoid()
    probs = sigmoid(torch.Tensor(predictions))
    # next, use threshold to turn them into integer predictions
    y_pred = np.zeros(probs.shape)
    y_pred[np.where(probs >= threshold)] = 1
    # finally, compute metrics
    y_true = labels
    f1_micro_average = f1_score(y_true=y_true, y_pred=y_pred, average='micro')
    roc_auc = roc_auc_score(y_true, y_pred, average = 'micro')
    accuracy = accuracy_score(y_true, y_pred)
    # return as dictionary
    metrics = {'f1': f1_micro_average,
               'roc_auc': roc_auc,
               'accuracy': accuracy}
    return metrics
```

```
#forward pass
outputs = model(input_ids=encoded_dataset['train']['input_ids'][0].unsqueeze(0), labels=encoded_dataset['train'][0]['labels'].unsqueeze(0))
outputs
```

```
SequenceClassifierOutput([('loss',
                           tensor(0.7997, grad_fn=<BinaryCrossEntropyWithLogitsBackward0>)),
                          ('logits',
                           tensor([[-0.7018, -0.0990,  0.2063, -0.0925]], grad_fn=<AddmmBackward0>))])
```

خروجی به این شکل مشخص می‌شود که پس از آن Trainer مدل را تعریف می‌کنیم و تمامی داده‌های Train را به آن می‌دهیم. حال با توجه به همان Encoderای که در ابتدا نیز زده‌بودیم آن را صدا می‌زنیم.

نوبت آموزش مدل است:

```
trainer = Trainer(
    model,
    args,
    train_dataset=encoded_dataset["train"],
    eval_dataset=encoded_dataset["train"],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)

trainer.train()
```

در ۵ مرحله آموزش می‌بینند:

```
Total optimization steps = 17130
```

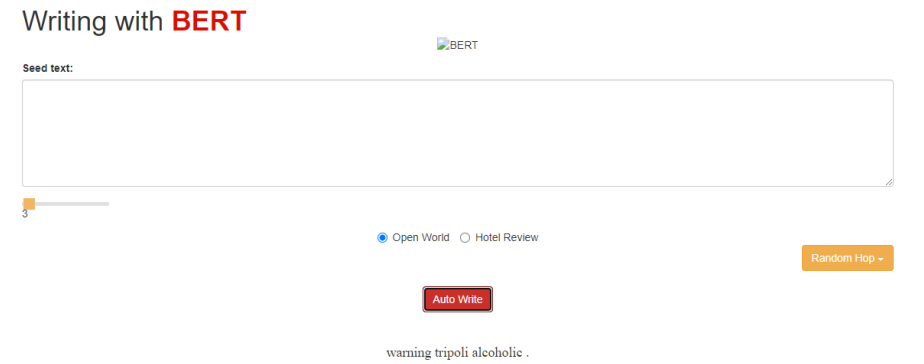
[17130/17130 1:19:09, Epoch 5/5]

Epoch	Training Loss	Validation Loss	F1	Roc Auc	Accuracy
1	0.477600	0.429051	0.421364	0.629788	0.296413
2	0.413900	0.337104	0.655823	0.754303	0.552644
3	0.321600	0.216960	0.838728	0.882827	0.801285
4	0.241800	0.143474	0.903547	0.930209	0.881939
5	0.187700	0.116670	0.922774	0.945841	0.911390

نتیجه نهایی نیز به این شکل است که بسیار قابل قبول است و دقت بالایی دارد و Loss کم. در انتها نیز مدل را Save می‌کنیم تا بتوانیم برای تولید جملات با استفاده از Prediction از آن استفاده کنیم.

برای تولید جمله فقط لازم است app.py را راه اندازی کرده و مکان ذخیره شده مدل را به آن بدهیم. سپس با باز کردن index.html و تعیین پارامتر های دلخواه جمله تولید شده جدید را دریافت کنید.

```
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
[CLS]  MASK MASK MASK . [SEP]
127.0.0.1 - - [10/Jul/2022 07:39:28] "POST /autocomplete HTTP/1.1" 200 -
[CLS]  MASK MASK MASK . [SEP]
```



بخش دوم

برای این بخش ابتدا داده های تولید شده در فاز اول پروژه را دریافت کرده و ستون های label که Index نمرات MPA فیلم هاست و text که جمله Normalize شده در فاز قبل است را به آن اضافه میکنیم.

```
df = pd.read_csv(base_dir + 'data/cleaned/data.csv', index_col=0)
df['Normalized_Plot'] = df['Normalized_Plot'].apply(lambda lst : ast.literal_eval(lst))
df
```

	Title	MPA	Plot	Normalized_Plot
0	The Lion King	G	Lion prince Simba and his father are targeted ...	[lion, prince, simba, father, targeted, bitter...
1	Cars	G	A hot-shot race-car named Lightning McQueen ge...	[hotshot, racecar, named, lightning, mcqueen, ...
2	Luck	G	The curtain is pulled back on the millennia-ol...	[curtain, pulled, back, millenniaold, battle, ...
3	2001: A Space Odyssey	G	The Monoliths push humanity to reach for the s...	[monolith, push, humanity, reach, star, discov...
4	Ratatouille	G	A rat who can cook makes an unusual alliance w...	[rat, cook, make, unusual, alliance, young, ki...
...	...	...	...	...
27396	Zombie Diaries	R	An unknown virus begins spreading and within w...	[unknown, virus, begin, spreading, within, wee...
27397	Gangsta Rap: The Glockumentary	R	The hardest group you've never heard of is bac...	[hardest, group, youve, never, heard, back, se...
27398	Satan's Sadists	R	The "Satans" are a very cruel biker gang led b...	[satan, cruel, biker, gang, led, anchor, gang,...
27399	Train of Life	R	In 1941, the inhabitants of a small Jewish vil...	[inhabitant, small, jewish, village, central, ...
27400	The Devil's Female	R	After the gruesome death of her father, a youn...	[gruesome, death, father, young, beautiful, wo...

27401 rows × 4 columns

```
mpa_dict = {'G': 0, 'PG': 1, 'PG-13': 2, 'R': 3}
df['label'] = [mpa_dict[x] for x in df['MPA']]
df['text'] = [' '.join(x) for x in df['Normalized_Plot']]
```

داده های آموزشی و تست را از دو ستون اضافه شده ساخته و y ها را به صورت Categorical در میاوریم.

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(df['text'], df['label'])

y_train = to_categorical(y_train, num_classes=4)
y_test = to_categorical(y_test, num_classes=4)

x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

```
((20550,), (20550, 4), (6851,), (6851, 4))
```

برای مدل این بخش، ابتدا یک لایه به نام vectorize_layer تعریف کرده که وظیفه دارد جملات را به صورت دنباله ای با طول ثابت و تشکیل شده از Index کلمات در بیاورد. آن را با استفاده از داده های آموزشی ابتدا adapt کرده تا بتوانیم در مدل استفاده کنیم.

```
vectorize_layer = TextVectorization(
    max_tokens=max_features,
    output_mode="int",
    output_sequence_length=sequence_length,
)
vectorize_layer.adapt(x_train)
```

برای ساخت مدل، بعد از لایه vectorize، یک لایه embedding میگذاریم. سپس بعد از یک لایه Dropout، دو لایه Conv گذاشته و GlobalMaxPooling زده تا بتوان داده های بدست آمده را به لایه های Dense داد. در نهایت بعد از یک لایه Dense ۱۲۸ تایی و تعدادی Dropout لایه آخر که شامل ۴ نورون به ازای هر MPA قرار میدهیم تا مدل کامل شود. برای loss از categorical_crossentropy و برای optimizer از adam استفاده میکنیم.

```
text_input = tf.keras.Input(shape=(1,), dtype=tf.string, name='text')
x = vectorize_layer(text_input)
x = layers.Embedding(max_features + 1, embedding_dim)(x)

x = layers.Dropout(0.5)(x)

# Conv1D + global max pooling
x = layers.Conv1D(128, 7, padding='valid', activation='relu', strides=3)(x)
x = layers.Conv1D(128, 7, padding='valid', activation='relu', strides=3)(x)
x = layers.GlobalMaxPooling1D()(x)

# We add a vanilla hidden layer:
x = layers.Dense(128, activation='relu')(x)
x = layers.Dropout(0.5)(x)

# We project onto a single unit output layer, and squash it with a sigmoid:
predictions = layers.Dense(4, activation='sigmoid', name='predictions')(x)

model = tf.keras.Model(text_input, predictions)

# Compile the model with binary crossentropy loss and an adam optimizer.
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

سپس در ۳ epoch مدل را آموزش میدهیم.(وقتی با epoch های بیشتر fit کردم، مدل overfit میشد و دقت داده های validation ثابت میموند، به همین دلیل فقط ۳ epoch آموزش دادیم.) همانطور که میبینید دقت داده های validation برابر با ۴۲ درصد و loss برابر با ۱.۳۸ است.

```
model.fit(tf.constant(x_train), y_train, validation_split=0.2, epochs=3)

Epoch 1/3
514/514 [==============================] - 6s 10ms/step - loss: 1.2706 - accuracy: 0.3876 - val_loss: 1.1760 - val_accuracy: 0.4577
Epoch 2/3
514/514 [==============================] - 5s 10ms/step - loss: 1.0769 - accuracy: 0.5018 - val_loss: 1.1964 - val_accuracy: 0.4421
Epoch 3/3
514/514 [==============================] - 5s 9ms/step - loss: 0.8191 - accuracy: 0.6464 - val_loss: 1.3834 - val_accuracy: 0.4219
<keras.callbacks.History at 0x7f90fe023210>
```

در نهایت مدل را save کرده و آن را با داده های تست evaluate میکنیم که به دقتی شبیه به داده های validation میرسیم.

```
model.save(base_dir + 'models/simple_text_classification', save_format = "tf")
loaded_model = load_model(base_dir + 'models/simple_text_classification')
loaded_model.evaluate(tf.constant(x_test), y_test)

INFO:tensorflow:Assets written to: ./drive/MyDrive/University/00012-NLP/MovieClassification/models/simple_text_classification/assets
INFO:tensorflow:Assets written to: ./drive/MyDrive/University/00012-NLP/MovieClassification/models/simple_text_classification/assets
215/215 [==============================] - 1s 6ms/step - loss: 1.3922 - accuracy: 0.4261
[1.392188310623169, 0.4260692000389099]
```

بخش سوم

در این بخش سه مدل مختلف را آموزش داده تا بتوانیم به دقت بهتری از قسمت قبل برسیم. در مدل اول میخواهیم با یک لایه Transformers به بهبود مدل کمک کنیم. لایه Transformers را به صورت زیر تعریف میکنیم که شامل یک لایه Attention دو لایه برای Normalization و دو لایه Dropout است.

```
class TransformerBlock(layers.Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super(TransformerBlock, self).__init__()
        self.att = layers.MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)
        self.ffn = tf.keras.Sequential(
            [layers.Dense(ff_dim, activation="relu"), layers.Dense(embed_dim),]
        )
        self.layernorm1 = layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = layers.LayerNormalization(epsilon=1e-6)
        self.dropout1 = layers.Dropout(rate)
        self.dropout2 = layers.Dropout(rate)

    def call(self, inputs, training):
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        return self.layernorm2(out1 + ffn_output)
```

همچنین برای پیش پردازش Transformers یک لایه Embedding برای زمان هر کلمه نیاز داریم. برای این کار یک لایه TokenAndPositionEmbedding تعریف کرده که در آن هر دو Embedding انجام شود.

```
class TokenAndPositionEmbedding(layers.Layer):
    def __init__(self, maxlen, vocab_size, embed_dim):
        super(TokenAndPositionEmbedding, self).__init__()
        self.token_emb = layers.Embedding(input_dim=vocab_size, output_dim=embed_dim)
        self.pos_emb = layers.Embedding(input_dim=maxlen, output_dim=embed_dim)

    def call(self, x):
        maxlen = tf.shape(x)[-1]
        positions = tf.range(start=0, limit=maxlen, delta=1)
        positions = self.pos_emb(positions)
        x = self.token_emb(x)
        return x + positions
```

سپس مدل را تقریبا مانند مدل قسمت قبل ساخته با این تفاوت که به جای لایه های Conv از Transformers استفاده میکنیم.

```
text_input = tf.keras.Input(shape=(1,), dtype=tf.string, name='text')
x = vectorize_layer(text_input)

embedding_layer = TokenAndPositionEmbedding(sequence_length, max_features, embedding_dim)
x = embedding_layer(x)

transformer_block = TransformerBlock(embedding_dim, num_heads, ff_dim)
x = transformer_block(x)

x = layers.GlobalAveragePooling1D()(x)
x = layers.Dropout(0.1)(x)

x = layers.Dense(20, activation='relu')(x)
x = layers.Dropout(0.1)(x)

outputs = layers.Dense(4, activation='softmax')(x)

model = tf.keras.Model(inputs=text_input, outputs=outputs)
```

نتیجه آموزش مدل و save و evaluate آن را نیز در زیر مشاهده میکنید که هم دقت و هم loss کمی (نه خیلی) نسبت به حالت قبل بهتر شده است.

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=32, epochs=3, validation_split=0.2)
```

```
Epoch 1/3
514/514 [==============================] - 21s 35ms/step - loss: 1.3396 - accuracy: 0.3584 - val_l
Epoch 2/3
514/514 [==============================] - 17s 33ms/step - loss: 1.2198 - accuracy: 0.4210 - val_l
Epoch 3/3
514/514 [==============================] - 17s 33ms/step - loss: 1.0471 - accuracy: 0.5191 - val_l
<keras.callbacks.History at 0x7f8e4b6e2050>
```

```
model.save(base_dir + 'models/text_classification_using_transformers', save_format = "tf")
loaded_model = load_model(base_dir + 'models/text_classification_using_transformers')
loaded_model.evaluate(tf.constant(x_test), y_test)
```

```
WARNING:absl:Found untraced functions such as embedding_3_layer_call_fn, embedding_3_layer_call_ar
INFO:tensorflow:Assets written to: ./drive/MyDrive/University/00012-NLP/MovieClassification/models
INFO:tensorflow:Assets written to: ./drive/MyDrive/University/00012-NLP/MovieClassification/models
215/215 [==============================] - 3s 13ms/step - loss: 1.2192 - accuracy: 0.4396
[1.2192487716674805, 0.43964385986328125]
```

برای مدل دوم این بخش به سراغ استفاده از Bert رفتیم. برای این کار ابتدا یک نسخه از Bert را در نظر گرفته و لینک لایه Encoder و Preprocess آن را بدست آورده تا بتوانیم از آن ها استفاده کنیم.

```
bert_model_name = 'small_bert/bert_en_uncased_L-4_H-512_A-8'

tfhub_handle_encoder = 'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1'
tfhub_handle_preprocess = 'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3'

print(f'BERT model selected           : {tfhub_handle_encoder}')
print(f'Preprocess model auto-selected: {tfhub_handle_preprocess}')
```

```
BERT model selected           : https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1
Preprocess model auto-selected: https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3
```

حال لایه Preprocess را دانلود کرده و با یک مثال دستی آن را تست میکنیم.

```
bert_preprocess_model = hub.KerasLayer(tfhub_handle_preprocess)
```

```
text_test = ['Hi bitchez!']
text_preprocessed = bert_preprocess_model(text_test)

print(text_test)
print(f'Keys       : {list(text_preprocessed.keys())}')
print(f'Shape      : {text_preprocessed["input_word_ids"].shape}')
print(f'Word Ids   : {text_preprocessed["input_word_ids"][0, : 20]}')
print(f'Input Mask : {text_preprocessed["input_mask"][0, : 20]}')
print(f'Type Ids   : {text_preprocessed["input_type_ids"][0, : 20]}')
```

```
['Hi bitchez!']
Keys       : ['input_mask', 'input_type_ids', 'input_word_ids']
Shape      : (1, 128)
Word Ids   : [ 101 7632 7743 9351  999  102    0    0    0    0    0    0    0    0
    0    0    0    0    0    0]
Input Mask : [1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
Type Ids   : [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

برای لایه Encoder نیز همین کار را تکرار کرده و خروجی لایه Preprocess را به عنوان ورودی نمونه به لایه داده تا عملکرد آن را بررسی کنیم.

```
bert_model = hub.KerasLayer(tfhub_handle_encoder)

bert_results = bert_model(text_preprocessed)

print(f'Loaded BERT: {tfhub_handle_encoder}')
print(f'Pooled Outputs Shape:{bert_results["pooled_output"].shape}')
print(f'Pooled Outputs Values:{bert_results["pooled_output"][0, :12]}')
print(f'Sequence Outputs Shape:{bert_results["sequence_output"].shape}')
print(f'Sequence Outputs Values:{bert_results["sequence_output"][0, :12]}')
```

```
Loaded BERT: https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1
Pooled Outputs Shape:(1, 512)
Pooled Outputs Values:[ 0.91743726  0.99340814  0.08266909  0.38530192  0.74971145  0.99194753
  0.97732455 -0.9993376  -0.50059444 -0.9998959   0.31000003 -0.966126  ]
Sequence Outputs Shape:(1, 128, 512)
Sequence Outputs Values:[[ 0.41427016  0.12808953  0.2766602  ... -0.7248013   0.9900739
  -0.9873097 ]
 [ 1.2677718   0.6866079   1.0910625  ... -0.02904399  0.51837385
  -0.8078487 ]
 [ 0.32114178  1.6371136   0.41725123 ...  0.03489165  0.22794688
  -0.7517949 ]
 ...
 [ 0.37599123  0.1691269  -0.05600987 ...  0.17825417  0.87398833
  -0.56442255]
 [ 0.564292    0.02312935  0.17420608 ... -0.0688891   1.2138271
  -0.5453472 ]
 [ 0.31692556 -0.12661119  0.30765903 ... -0.21302785  1.1239926
  -0.7628193 ]]
```

برای ساخت مدل این لایه های Bert را به هم متصل کرده و در نهایت به یک Dense ۴ تایی وصل میکنیم تا با داده های ما سازگار شود.

```
def build_classifier_model():
    text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
    preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
    encoder_inputs = preprocessing_layer(text_input)
    encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder')
    outputs = encoder(encoder_inputs)
    net = outputs['pooled_output']
    net = tf.keras.layers.Dense(4, activation='softmax', name='classifier')(net)
    return tf.keras.Model(text_input, net)
```

در ادامه معماری مدل ساخته شده و اجرای یک مثال رو آن را میبینیم.

```
classifier_model = build_classifier_model()
bert_raw_result = classifier_model(tf.constant(text_test))
print(tf.sigmoid(bert_raw_result))
classifier_model.summary()
```

```
tf.Tensor([[0.5097018 0.5128283 0.6508063 0.5713298]], shape=(1, 4), dtype=float32)
Model: "model_6"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 text (InputLayer)              [(None,)]            0           []

 preprocessing (KerasLayer)     {'input_mask': (Non  0           ['text[0][0]']
                                e, 128),
                                 'input_type_ids':
                                (None, 128),
                                 'input_word_ids':
                                (None, 128)}

 BERT_encoder (KerasLayer)      {'default': (None,   47677953    ['preprocessing[0][0]',
                                512),                             'preprocessing[0][1]',
                                 'sequence_output':               'preprocessing[0][2]']
                                 (None, 128, 512),
                                 'encoder_outputs':
                                 [(None, 128, 512),
                                 (None, 128, 512),
                                 (None, 128, 512),
                                 (None, 128, 512),
                                 (None, 128, 512),
                                 (None, 128, 512),
                                 (None, 128, 512),
                                 (None, 128, 512),
                                 (None, 128, 512),
                                 (None, 128, 512)],
                                 'pooled_output': (
                                None, 512)}

 classifier (Dense)             (None, 4)            2052        ['BERT_encoder[0][11]']

==================================================================================================
Total params: 47,680,005
Trainable params: 47,680,004
Non-trainable params: 1
```

در آخر نیز مدل را Compile کرده و bert را fine tune میکنیم. همانطور که میبینید دقت نهایی نسبت به مدل های پیشین پایین تر است. من چندین بار با عوض کردن مدل bert، اضافه کردن لایه Dense بالا پایین کردن hyperparameter ها سعی کردم که دقت مدل را بالا ببرم ولی نتوانستم به بالا تر ۳۶ درصد برسم.

```
classifier_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics = ['accuracy'])
```

```
classifier_model.fit(x_train, y_train, validation_split=.3, epochs=3, batch_size=64)
```

```
Epoch 1/3
225/225 [==============================] - 244s 1s/step - loss: 1.3752 - accuracy: 0.3383 - val_loss: 1.3305 - val_accuracy: 0.3631
Epoch 2/3
225/225 [==============================] - 230s 1s/step - loss: 1.3362 - accuracy: 0.3491 - val_loss: 1.3383 - val_accuracy: 0.3631
Epoch 3/3
225/225 [==============================] - 230s 1s/step - loss: 1.3393 - accuracy: 0.3466 - val_loss: 1.3344 - val_accuracy: 0.3631
<keras.callbacks.History at 0x7f4fefb162d0>
```

```
classifier_model.save(base_dir + 'models/text_classification_using_bert', save_format = "tf")
loaded_model = load_model(base_dir + 'models/text_classification_using_bert')
loaded_model.evaluate(tf.constant(x_test), y_test)
```

```
WARNING:absl:Found untraced functions such as restored_function_body, restored_function_body, restored_function_body, restored_functior
INFO:tensorflow:Assets written to: ./drive/MyDrive/University/00012-NLP/MovieClassification/models/text_classification_using_bert/asset
INFO:tensorflow:Assets written to: ./drive/MyDrive/University/00012-NLP/MovieClassification/models/text_classification_using_bert/asset
215/215 [==============================] - 23s 104ms/step - loss: 1.3246 - accuracy: 0.1244
[1.3245573043823242, 0.12436141073703766]
```

در آخر با استفاده از کتابخانه AutoKeras سعی کردیم بهترین مدل ممکن Text Classification را بر روی داده های خود پیدا کنیم. این کار را در ۳ Trial انجام داده و برای هر کدام مدل را ۳ epoch آموزش دادیم.

```
clf = ak.TextClassifier(max_trials=3)

clf.fit(X_train, y_train, validation_split=0.3, epochs=3)
```

در نهایت معماری بهترین مدل بدست آمده را در زیر میبینید که بسیار نزدیک به معماری مدل بخش قبلی است، ولی با جزئیاتی که من متوجه آن نشدم توانسته دقت بالا تری را بدست بیاورد.

```
model = clf.export_model()
model.summary()
```

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
=================================================================================================
 input_1 (InputLayer)           [(None,)]            0           []

 expand_last_dim (ExpandLastDim  (None, 1)           0           ['input_1[0][0]']
 )

 bert_tokenizer (BertTokenizer)  ((None, None),      0           ['expand_last_dim[0][0]']
                                 (None, None),
                                 (None, None))

 bert_encoder (BertEncoder)     (None, 768)          109482240   ['bert_tokenizer[0][0]',
                                                                  'bert_tokenizer[0][1]',
                                                                  'bert_tokenizer[0][2]']

 dense (Dense)                  (None, 4)            3076        ['bert_encoder[0][0]']

 classification_head_1 (Softmax  (None, 4)           0           ['dense[0][0]']
 )

=================================================================================================
Total params: 109,485,316
Trainable params: 109,485,316
Non-trainable params: 0
_________________________________________________________________________________________________
```

در نهایت مدل با دقت نزدیک به ۵۰ درصد و loss ۱.۱۵ توانست روی داده های test، evaluate کند.

```
215/215 [==============================] - 23s 99ms/step - loss: 1.1519 - accuracy: 0.4998
[1.1518988609313965, 0.4997810423374176]
```

دقت پایین در تمام مدل ها احتمالا ناشی از آن است که تعداد label ها ۴ تاست و شاید Accuracy معیار مناسبی برا ارزیابی نباشد. از طرفی تعداد داده های موجود کم بود که در پایین آمدن دقت بی تاثیر نیست. البته ما با استفاده از تعدادی data augmentation تلاش در زیاد کردن داده ها کردیم ولی با تاثیر چندانی روی دقت مدل ها نگذاشت و از آن ها استفاده نکردیم.