|  |  |
| --- | --- |
| نام و نام خانوادگی | **محمد پویا افشاری – علیرضا اسمعیل زاده** |
| شماره دانشجویی | **810198351-810198577** |
| تاریخ ارسال گزارش | **1402.09.17** |

|  |  |  |
| --- | --- | --- |
|  | **به نام خدا**  **دانشگاه تهران**  **دانشکده‌ مهندسی برق و کامپیوتر** |  |
| **درس شبکه‌های عصبی و یادگیری عمیق**  **تمرین پنجم** | | |

**فهرست**

[**پاسخ 1** – **SAM** 4](#_Toc152956619)

[۱-۱. **آماده سازی مجموعه داده** 4](#_Toc152956620)

[۱-۱-۱ . افزودن کتاب خانه ها 4](#_Toc152956621)

[۱-۱-۲ . جمع آوری داده 4](#_Toc152956622)

[۱-۱-۳ . resize تصاویر 5](#_Toc152956623)

[۱-۱-۴ . نمونه تصاویر 6](#_Toc152956624)

[۱-۱-۵ . تقسیم داده به دو بخش آموزش و ارزیابی 7](#_Toc152956625)

[**۲-۱**. بارگذاری مدل 7](#_Toc152956626)

[۱-۲-۱ . آماده سازی دیتاست 7](#_Toc152956627)

[۳**-۱.** تقويت داده 10](#_Toc152956628)

[**۴-۱.** بهینه‌ساز، متریک و تابع هزینه 10](#_Toc152956629)

[۵-۱. Fine-Tuneکردن مدل 11](#_Toc152956630)

[۱-۵-۱ . آموزش مدل 11](#_Toc152956631)

[۱-۵-۲ . ذخیره مدل 11](#_Toc152956632)

[**۶-۱.** ازیابی نتایج 12](#_Toc152956633)

[۱-۶-۱ . افزودن کتاب‌خانه ها 12](#_Toc152956634)

[۱-۶-۲ . لود کردن مدل ذخیره شده 12](#_Toc152956635)

[۱-۶-۳ . اجرای داده ازیابی بر روی مدل 13](#_Toc152956636)

[**پاسخ** **۲**  - آشنایی و پیاده سازی مدل Faster RCNN 15](#_Toc152956637)

[۱-۲. **توضیحات مدل ها** 15](#_Toc152956638)

[۲-۲ . پیش پردازش 19](#_Toc152956639)

[۲-۲-۱ . کتابخانه 19](#_Toc152956640)

[۲-۲-۲ . افزودن دیتاست 19](#_Toc152956641)

[۲-۲-۳ . نشان دادن تصاویر نمونه 19](#_Toc152956642)

[۲-۲-۴ . بدست آوردن تعداد کلاس‌ها و تغییر اندازه 20](#_Toc152956643)

[۲-۳ . آموزش شبکه 22](#_Toc152956644)

[۲-۳-۱ . آماده سازی دیتا 22](#_Toc152956645)

[۲-۳-۲ . نمایش تصاویر با Label های مربوط از روی کلاس تشکیل شده‌ی بالا 27](#_Toc152956646)

[۲-۳-۳ . آموزش شبکه و پیاده سازی مدل 28](#_Toc152956647)

[۲-۴ . بررسی داده های تست 31](#_Toc152956648)

**شکل‌ها**

شکل 1. عنوان تصویر نمونه 4

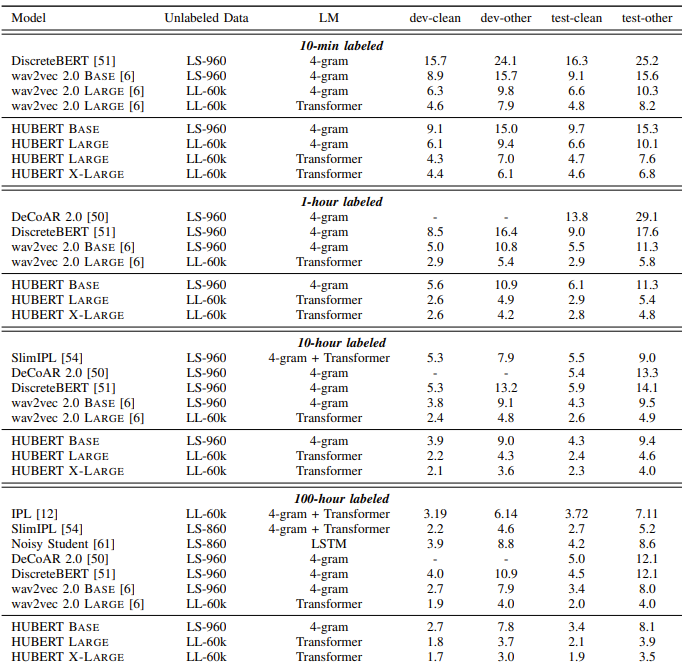
# **پاسخ 1** – **تشخیص احساسات گفتار(SER)**

## ۱-۱. **معرفی مدل HuBERT**

به طور کلی مدل HuBERT شبیه مدل word2vec 2.0 عمل می‌کند با این تفاوت که مقادیر را به جای quantize کردن cluster می‌کند. از مدل تقریبا مشابه BERT استفاده می‌کند و نیز مقدار loss ان ساده تر شده است. جدول زیر تفاوت مدل ها در عمل کرد را نشان می‌دهد:

جدول 1

جدول 2جدول 1. مقادیر مدل در برابر literature review



## ۱-۲. سوالات تشریحی

### ۱-۲-۱. چالش هاي داده هاي صوتی در یادگیري

مسائل یادگیری Self-supervised برای نمایش Speech به سه چالش اصلی تقسیم بندی می‌شوند:

1. به ازای هر گفته ورودی چندین واحد صدا وجود دارد

(There are multiple sound units for each input utterance)

1. مدل بدون اینکه به Transcription متنی دسترسی داشته باشد از روی Audio waveform باید تشخیص بدهد. که این کار را برای تشخیص مدل سخت می‌کند.

(There is no lexicon of input sound units during the pre-training phase)

1. مقادیر Sound unit ها در هر زبان متفاوت و متغیر است. مثلا در هیچ زبانی هر 20ms به عنوان یک Sound unit حساب نمی‌شود و این مقادیر در صوت های گوناگون مختلف است.

(Sounds units have variable lengths with no explicit segmentation)

علاوه بر سه مشکل اصلی ذکر شده در کار کردن بر روی داده‌های مصوت باید در نظر داشته باشیم که گویش هر فرد از یک کلمه با فرد دیگر متفاوت است. در کار بر روی متن ما بیشتر ساختار مند هستیم و همینطور قواعد جمله بندی و دستور زبان را رعایت می‌کنیم که در صوت این امر کمتر دیده می‌شود.

### ۱-۲-۲. رویکرد HuBERT

برای حل مسائل مطرح شده رویکرد‌های مختلفی مطرح شد از جمله آن مدل ‌هایی هستند که کلا بدید Quantization عمل می‌کنند و مدل های با Contrastive loss و Diversity loss. در مقایل HuBERT رویکرد متفاوتی در نظر می‌‍گیرد. رویکر HuBERT بیشتر شبیه BERT است. بنابر این از مدل های Language به عنوان منبع برای Speech representation بهره می‌گیرد.

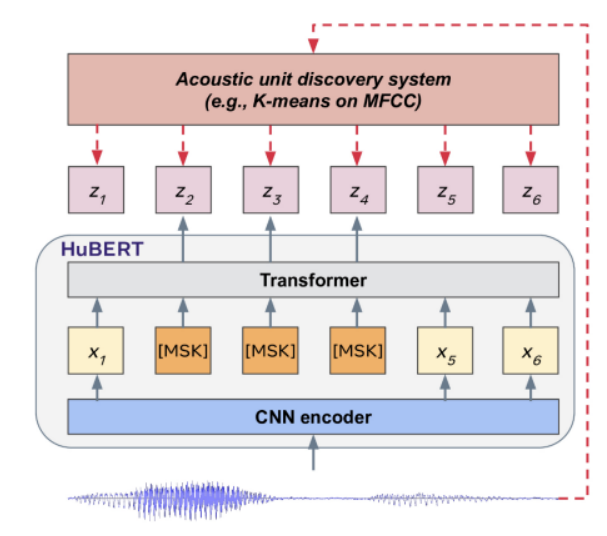


Figure 1شکل 1. شمای کلی فرآیند پیش آموزش HuBERT

برای حل مشکلات مطرح شده در شمای Pre-training مدل نشان داده می‌شود که ابتدا صوت دریافت شده به Encoder می‌رود. در این مرحله Encoder صوت رو به Snippet های 20 میلی ثانیه بخش بخش می‌کند. این حالت Speech representation تولید شده است. سپس نیمی از آنها Mask می‌شوند. از سمت دیگر یک Acoustic unit discovery system می‌بینیم که عمل Clustering را انجام می‌دهد. نتیجه این بخش تولید Representational unit های قابل بازنمایی Transformer است. مدل با انجام Cross entropy میتواند تفاوت بین این Z(k) و [MSK] مقدار MSK را بازنمایی کند.

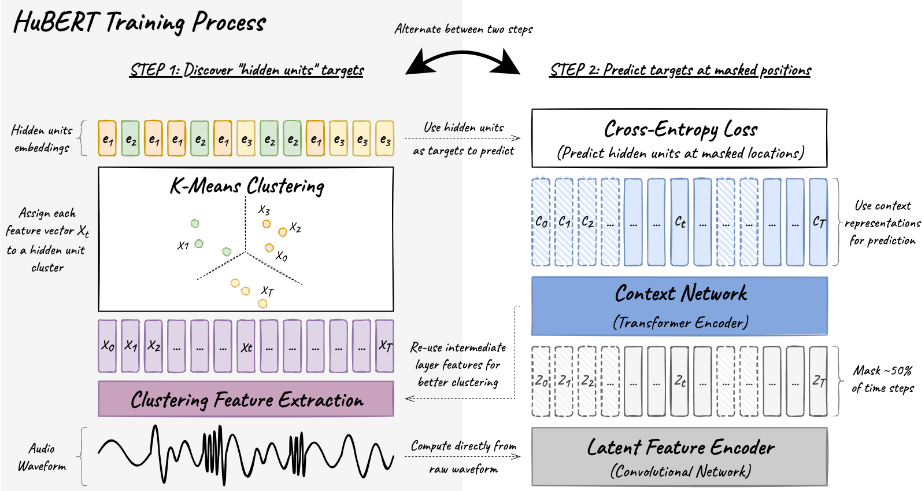


Figure 2شکل2. نمایش شمای train مدل HuBERT

بر اساس مدل یادگیری HuBERT ابتدا از Waveform که به Encoder داده می‌شود ( که در واقع شبکه CNN) است یک سری Hidden unit 20ms استخراج می‌شود. نیمی از این خروجی در مرحله Mask می‌شود. در مرحله Hidden units پیدا کردن به وسیله K-means cluster صورت می‌گیرد. در این مرحله همه ی Audio frame هایی که به یک Cluster assign می‌شود یک Label یکتا می‌گیرند.در نهایت Cluster Embedding های تولید شده ی این مرحله برای مقایسه Transformer Context Network و نهایتا خروجی Prediction با Cross Entropy loss برای Mask location ها می‌رود. درمرحله Prediction از Embedding vector به منظور استفاده می‌شود. در حقیقت مرحله دوم که مرحله Predict noisy target from context هست مشابه BERT صورت می‌پذیرد.

# **پاسخ** **۲** - تنظیم دقیق مدل BERT

## ۲-۱. آموزش و تحلیل مدل BERT

مدل BERT به عنوان State of Art مدل های Transformer برای تسک LLM شناخته میشود.

به ترتیب از مدل، دیتاست و مقاله‌های اورده شده در زیر برای تسک‌های مختلف استفاده شده است.

Model:

\* [ParsBert](https://huggingface.co/HooshvareLab/bert-base-parsbert-uncased)

Dataset:

\* [FarsTails](https://github.com/dml-qom/FarsTail)

Papers used for this project:

\* [What Would Elsa Do? Freezing Layers During Transformer Fine-Tuning](https://arxiv.org/abs/1911.03090).

\* [Are Sixteen Heads Really Better than One?](https://arxiv.org/abs/1905.10650).

### ۲-۱-۱. پیش پردازش داده ها

### ۲-۱-۱-۱. اضافه کردن کتابخانه‌های مورد نیاز

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import cv2

import librosa

import os

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

import shutil

from google.colab import drive

from google.colab import files

import requests

import zipfile

import io

from transformers import AutoTokenizer

from transformers import TFAutoModelForSequenceClassification

import tensorflow as tf

import re

import unicodedata

from sklearn.metrics import confusion\_matrix, classification\_report, f1\_score, recall\_score, accuracy\_score

from sklearn.metrics import accuracy\_score, f1\_score, confusion\_matrix

from transformers.modeling\_outputs import BaseModelOutputWithPastAndCrossAttentions

import torch.optim as optim

from transformers import BertModel

import torch

### ۲-۱-۱-۲. دانلود و لود دیتاست

برای حل این تمرین از دیتاست [FarsTails](https://github.com/dml-qom/FarsTail) استفاده می‌کنیم

url = 'https://github.com/dml-qom/FarsTail/archive/refs/heads/master.zip'

request = requests.get(url, allow\_redirects=True)

zip\_file = zipfile.ZipFile(io.BytesIO(request.content))

target\_directory = '/content/FarsTail\_dataset'

zip\_file.extractall(target\_directory)

print(f'The FarsTails dataset has been successfully downloaded and extracted to {target\_directory}.')

### ۲-۱-۱-۳. ایجاد Train, Test, and Validation

train\_data = pd.read\_csv('/content/FarsTail\_dataset/FarsTail-master/data/Train-word.csv', sep='\t')

val\_data = pd.read\_csv('/content/FarsTail\_dataset/FarsTail-master/data/Val-word.csv', sep='\t')

test\_data = pd.read\_csv('/content/FarsTail\_dataset/FarsTail-master/data/Test-word.csv', sep='\t')

### ۲-۱-۱-۴. پردازش های آماری روی دادگان

* نمایش توزیع کلاس ها

all\_data = pd.concat([train\_data, val\_data, test\_data])

class\_distribution = all\_data['label'].value\_counts()

plt.figure(figsize=(8, 5))

class\_distribution.plot(kind='bar', color='skyblue')

plt.title('Distribution of Classes')

plt.xlabel('Class')

plt.ylabel('Count')

plt.show()

fig, axes = plt.subplots(1, 3, figsize=(18, 5), sharey=True)

*# Train dataset*

train\_distribution = train\_data['label'].value\_counts()

axes[0].bar(train\_distribution.index, train\_distribution, color='skyblue')

axes[0].set\_title('Train Dataset')

axes[0].set\_xlabel('Class')

axes[0].set\_ylabel('Count')

*# Validation dataset*

val\_distribution = val\_data['label'].value\_counts()

axes[1].bar(val\_distribution.index, val\_distribution, color='skyblue')

axes[1].set\_title('Validation Dataset')

axes[1].set\_xlabel('Class')

*# Test dataset*

test\_distribution = test\_data['label'].value\_counts()

axes[2].bar(test\_distribution.index, test\_distribution, color='skyblue')

axes[2].set\_title('Test Dataset')

axes[2].set\_xlabel('Class')

plt.show()

print("\nTraining Data Class Distribution:")

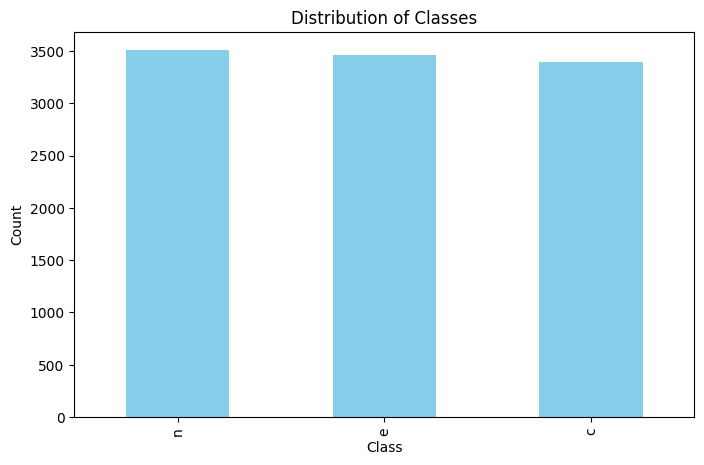
print(train\_distribution)

print("\nValidation Data Class Distribution:")

print(val\_distribution)

print("\nTest Data Class Distribution:")

print(test\_distribution)



Training Data Class Distribution:

n 2448

e 2429

c 2389

Name: label, dtype: int64

Validation Data Class Distribution:

n 523

e 515

c 499

Name: label, dtype: int64

Test Data Class Distribution:

n 535

e 519

c 510

Name: label, dtype: int64

* نمایش طول جملات

*# load model*

tokenizer = AutoTokenizer.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased")

def get\_sentence\_length(data):

    return data['premise'].apply(lambda x: len(tokenizer.encode(x, max\_length=512, truncation=True)))

train\_sentence\_length = get\_sentence\_length(train\_data)

train\_sentence\_length.describe()

count 7266.000000

mean 43.792458

std 17.085853

min 11.000000

25% 31.000000

50% 41.000000

75% 54.000000

max 136.000000

Name: premise, dtype: float64

validation\_sentence\_length = get\_sentence\_length(val\_data)

validation\_sentence\_length.describe()

count 1537.000000

mean 43.163305

std 16.824191

min 10.000000

25% 31.000000

50% 41.000000

75% 52.000000

max 107.000000

Name: premise, dtype: float64

test\_sentence\_length = get\_sentence\_length(test\_data)

test\_sentence\_length.describe()

count 1564.000000

mean 42.619565

std 17.148977

min 12.000000

25% 30.000000

50% 40.000000

75% 52.000000

max 128.000000

Name: premise, dtype: float64

*# Plot the distribution of sentence lengths*

fig, axes = plt.subplots(1, 3, figsize=(18, 5), sharey=True)

*# Train dataset*

axes[0].hist(train\_sentence\_length, bins=20, color='skyblue', edgecolor='black')

axes[0].set\_title('Train Dataset')

axes[0].set\_xlabel('Sentence Length')

axes[0].set\_ylabel('Count')

*# Validation dataset*

axes[1].hist(validation\_sentence\_length, bins=20, color='skyblue', edgecolor='black')

axes[1].set\_title('Validation Dataset')

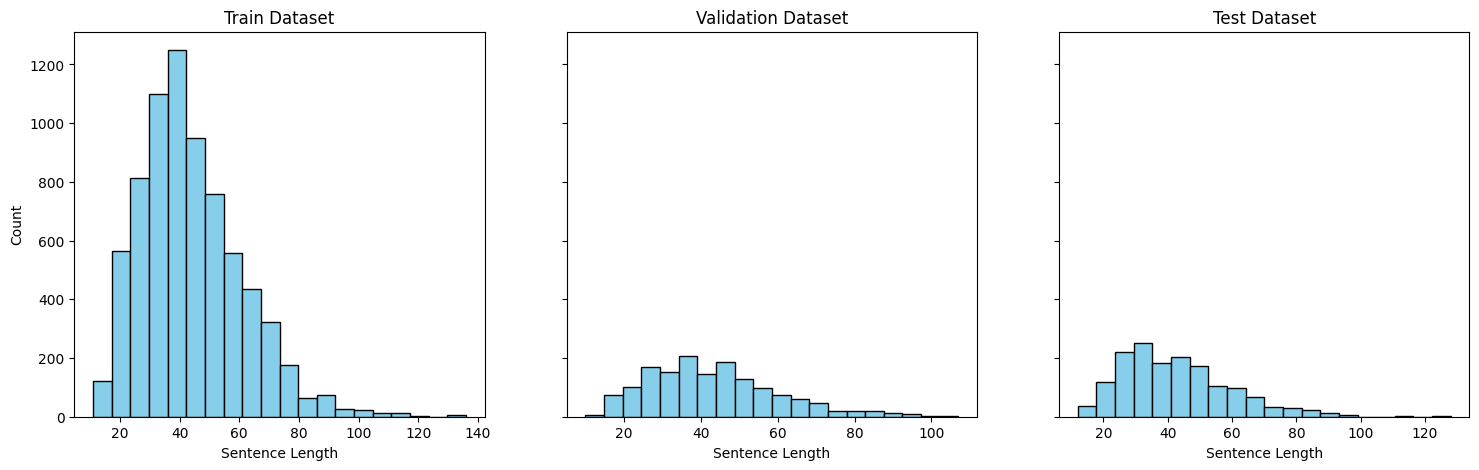
axes[1].set\_xlabel('Sentence Length')

*# Test dataset*

axes[2].hist(test\_sentence\_length, bins=20, color='skyblue', edgecolor='black')

axes[2].set\_title('Test Dataset')

axes[2].set\_xlabel('Sentence Length')



* نمایش متن تمیز شده ( حذف لینک و ..)

\* Remove Special Characters and Punctuation

\* Convert Text to Lowercase

\* Handle Unicode Characters

\* Remove Tags, HTML, CSS and so on using regex

\* Remove ASCII Unwanted

def handle\_unicode(text):

    return ''.join(c for c in unicodedata.normalize('NFD', text) if unicodedata.category(c) != 'Mn')

def remove\_ascii(text):

    unwanted\_ascii = set('\x00\x08\x0b\x0c\x0e\x1f\x7f\xff')

    return ''.join(char for char in text if char not in unwanted\_ascii)

def clean\_text(text):

    text = re.sub(r'[^\w\s]', '', text)

    text = text.lower()

    text = re.sub(r'\n', ' ', text)

    text = re.sub(r'http\S+|www\S+', '', text)

    text = re.sub(r'<.\*?>', '', text)

    text = re.sub(r'<style[^<]\*</style>|<script[^<]\*</script>', '', text)

    text = re.sub(r'([^@\s#\w]|\_)+', '', text)

    return text

def clean\_dataframe\_column(df, text\_column):

    df[text\_column] = df[text\_column].apply(handle\_unicode)

    df[text\_column] = df[text\_column].apply(remove\_ascii)

    df[text\_column] = df[text\_column].apply(clean\_text)

    return df

def clean\_dataframe(df, text\_columns):

    for column in text\_columns:

        df = clean\_dataframe\_column(df, column)

    return df

premise\_column = 'premise'

hypothesis\_column = 'hypothesis'

cleaned\_train\_data = clean\_dataframe(train\_data, [premise\_column, hypothesis\_column])

cleaned\_val\_data = clean\_dataframe(val\_data, [premise\_column, hypothesis\_column])

cleaned\_test\_data = clean\_dataframe(test\_data, [premise\_column, hypothesis\_column])

print(train\_data[['premise', 'hypothesis']].iloc[0])

premise اولین انتقال و نفوذ طبیعی فرهنگ و تمدن اسلامی ...

hypothesis نخستین انتقال و نفوذ طبیعی فرهنگ و تمدن اسلامی...

Name: 0, dtype: object

### ۲-۱-۲. تنظیم دقیق مدل Fine Tune

در این حالت شاهد بالاترین دقت و.. در بین مدل ها هستیم چون لایه ای Freeze و... نشده است.

توکنایز کردن:

tokenizer = AutoTokenizer.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased")

label\_map = {'e': 0, 'c': 1, 'n': 2}

train\_data['label'] = cleaned\_train\_data['label'].map(label\_map)

val\_data['label'] = cleaned\_val\_data['label'].map(label\_map)

test\_data['label'] = cleaned\_test\_data['label'].map(label\_map)

train\_encodings = tokenizer(list(train\_data['premise']), list(train\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='tf')

val\_encodings = tokenizer(list(val\_data['premise']), list(val\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='tf')

test\_encodings = tokenizer(list(test\_data['premise']), list(test\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='tf')

ایجاد دیتاست ( Tensorflow / PyTorch)

train\_dataset = tf.data.Dataset.from\_tensor\_slices((

    {k: train\_encodings[k] for k in train\_encodings},

    train\_data['label']

)).shuffle(len(train\_data)).batch(32)

val\_dataset = tf.data.Dataset.from\_tensor\_slices((

    {k: val\_encodings[k] for k in val\_encodings},

    val\_data['label']

)).batch(32)

test\_dataset = tf.data.Dataset.from\_tensor\_slices((

    {k: test\_encodings[k] for k in test\_encodings},

    test\_data['label']

)).batch(32)

کانفیگ مدل

model = TFAutoModelForSequenceClassification.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased", num\_labels=3)

optimizer = tf.keras.optimizers.Adam(learning\_rate=5e-5)

loss = tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True)

metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')

model.compile(optimizer=optimizer, loss=loss, metrics=[metric])

Train مدل

history = model.fit(

    train\_dataset,

    validation\_data=val\_dataset,

    epochs=5,

    batch\_size=9,

)

Epoch 1/5

228/228 [==============================] - 286s 1s/step - loss: 0.7299 - accuracy: 0.6563 - val\_loss: 0.5726 - val\_accuracy: 0.7580

Epoch 2/5

228/228 [==============================] - 241s 1s/step - loss: 0.4000 - accuracy: 0.8372 - val\_loss: 0.6097 - val\_accuracy: 0.7606

Epoch 3/5

228/228 [==============================] - 237s 1s/step - loss: 0.2311 - accuracy: 0.9118 - val\_loss: 0.6163 - val\_accuracy: 0.7996

Epoch 4/5

228/228 [==============================] - 237s 1s/step - loss: 0.1473 - accuracy: 0.9465 - val\_loss: 0.7020 - val\_accuracy: 0.8022

Epoch 5/5

228/228 [==============================] - 237s 1s/step - loss: 0.0982 - accuracy: 0.9661 - val\_loss: 0.7760 - val\_accuracy: 0.7990

اعمال روی test و evaluation

test\_predictions = []

test\_labels = []

for input\_ids, labels in test\_dataset:

    outputs = model(input\_ids=input\_ids)

    logits = outputs.logits

    predicted = np.argmax(logits, axis=1)

    test\_predictions.extend(predicted)

    test\_labels.extend(labels)

test\_predictions = np.array(test\_predictions)

test\_labels = np.array(test\_labels)

conf\_matrix = confusion\_matrix(test\_labels, test\_predictions)

class\_report = classification\_report(test\_labels, test\_predictions)

f1 = f1\_score(test\_labels, test\_predictions, average='weighted')

recall = recall\_score(test\_labels, test\_predictions, average='weighted')

accuracy = accuracy\_score(test\_labels, test\_predictions)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

*# Print the results*

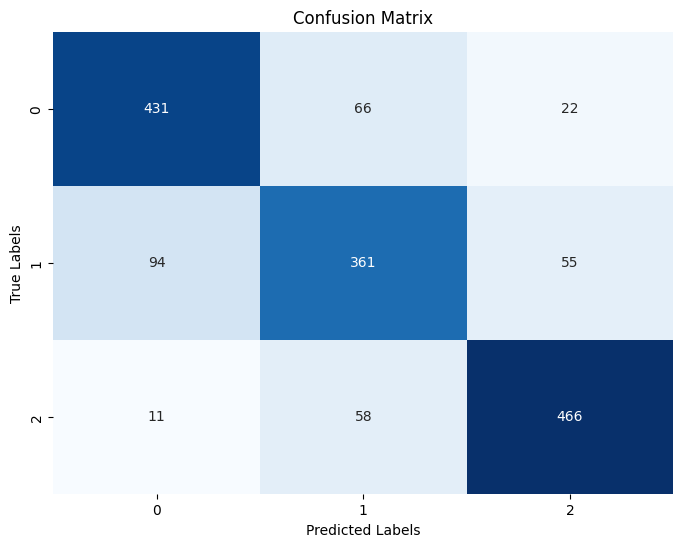
print("\nClassification Report:")

print(class\_report)

print("\nF1 Score:", f1)

print("Recall:", recall)

print("Accuracy:", accuracy)

Classification Report:

precision recall f1-score support

0 0.80 0.83 0.82 519

1 0.74 0.71 0.73 510

2 0.86 0.87 0.86 535

accuracy 0.80 1564

macro avg 0.80 0.80 0.80 1564

weighted avg 0.80 0.80 0.80 1564

F1 Score: 0.8034955703532141

Recall: 0.8043478260869565

Accuracy: 0.8043478260869565

بر اساس گزارش روی دادگان تست دقت مدل در این حالت بالا ترین سطح خودش قرار گرفته است.

train\_loss = history.history['loss']

val\_loss = history.history['val\_loss']

train\_accuracy = history.history['accuracy']

val\_accuracy = history.history['val\_accuracy']

fig, axes = plt.subplots(1, 2, figsize=(16, 5))

axes[0].plot(train\_loss, label='Training Loss')

axes[0].plot(val\_loss, label='Validation Loss')

axes[0].set\_title('Training and Validation Loss')

axes[0].set\_xlabel('Epoch')

axes[0].set\_ylabel('Loss')

axes[0].legend()

axes[1].plot(train\_accuracy, label='Training Accuracy')

axes[1].plot(val\_accuracy, label='Validation Accuracy')

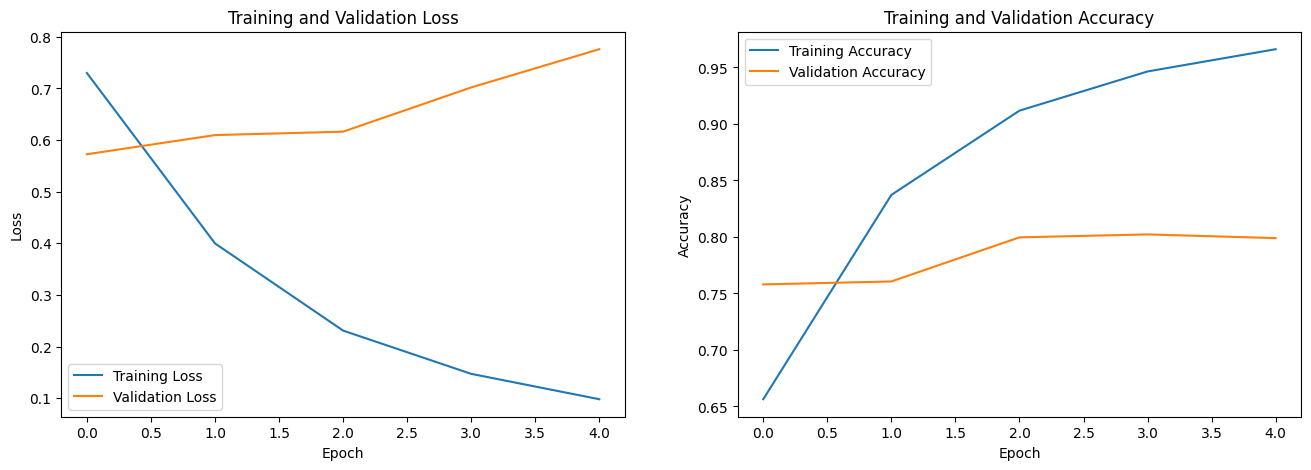
axes[1].set\_title('Training and Validation Accuracy')

axes[1].set\_xlabel('Epoch')

axes[1].set\_ylabel('Accuracy')

axes[1].legend()

plt.show()



### ۲-۱-۳. فریز کردن لایه ها

* فریز کردن 9 لایه ابتدایی مطابق مقاله

from transformers import AutoModel

model\_name = 'HooshvareLab/bert-base-parsbert-uncased'

model = AutoModel.from\_pretrained(model\_name)

num\_layers = model.config.num\_hidden\_layers

print(f"Number of hidden layers in the {model\_name} BERT model: {num\_layers}")

tokenizer = AutoTokenizer.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased")

label\_map = {'e': 0, 'c': 1, 'n': 2}

train\_data['label'] = cleaned\_train\_data['label'].map(label\_map)

val\_data['label'] = cleaned\_val\_data['label'].map(label\_map)

test\_data['label'] = cleaned\_test\_data['label'].map(label\_map)

train\_encodings = tokenizer(list(train\_data['premise']), list(train\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='tf')

val\_encodings = tokenizer(list(val\_data['premise']), list(val\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='tf')

test\_encodings = tokenizer(list(test\_data['premise']), list(test\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='tf')

train\_dataset = tf.data.Dataset.from\_tensor\_slices((

    {k: train\_encodings[k] for k in train\_encodings},

    train\_data['label']

)).shuffle(len(train\_data)).batch(32)

val\_dataset = tf.data.Dataset.from\_tensor\_slices((

    {k: val\_encodings[k] for k in val\_encodings},

    val\_data['label']

)).batch(32)

test\_dataset = tf.data.Dataset.from\_tensor\_slices((

    {k: test\_encodings[k] for k in test\_encodings},

    test\_data['label']

)).batch(32)

در اینجا تعیین میکنیم که نه لایه trainable نباشد.

from transformers import TFBertForSequenceClassification

model2 = TFBertForSequenceClassification.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased", num\_labels=3)

bert\_layer = model2.get\_layer('bert')

*# freeze*

num\_layers\_to\_freeze = 9

for i in range(num\_layers\_to\_freeze):

    bert\_layer.encoder.layer[i].trainable = False

optimizer = tf.keras.optimizers.Adam(learning\_rate=5e-5)

loss = tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True)

metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')

model2.compile(optimizer=optimizer, loss=loss, metrics=[metric])

history2 = model2.fit(

    train\_dataset,

    validation\_data=val\_dataset,

    epochs=5,

    batch\_size=9

)

Epoch 1/5

228/228 [==============================] - 224s 888ms/step - loss: 0.8506 - accuracy: 0.5751 - val\_loss: 0.7057 - val\_accuracy: 0.6630

Epoch 2/5

228/228 [==============================] - 199s 875ms/step - loss: 0.5367 - accuracy: 0.7710 - val\_loss: 0.5801 - val\_accuracy: 0.7606

Epoch 3/5

228/228 [==============================] - 199s 873ms/step - loss: 0.3203 - accuracy: 0.8713 - val\_loss: 0.5904 - val\_accuracy: 0.7840

Epoch 4/5

228/228 [==============================] - 203s 890ms/step - loss: 0.2021 - accuracy: 0.9255 - val\_loss: 0.6090 - val\_accuracy: 0.7892

Epoch 5/5

228/228 [==============================] - 199s 875ms/step - loss: 0.1176 - accuracy: 0.9575 - val\_loss: 0.7215 - val\_accuracy: 0.7794

test\_predictions = []

test\_labels = []

for input\_ids, labels in test\_dataset:

    outputs = model2(input\_ids=input\_ids)

    logits = outputs.logits

    predicted = np.argmax(logits, axis=1)

    test\_predictions.extend(predicted)

    test\_labels.extend(labels)

test\_predictions = np.array(test\_predictions)

test\_labels = np.array(test\_labels)

conf\_matrix = confusion\_matrix(test\_labels, test\_predictions)

class\_report = classification\_report(test\_labels, test\_predictions)

f1 = f1\_score(test\_labels, test\_predictions, average='weighted')

recall = recall\_score(test\_labels, test\_predictions, average='weighted')

accuracy = accuracy\_score(test\_labels, test\_predictions)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

*# Print the results*

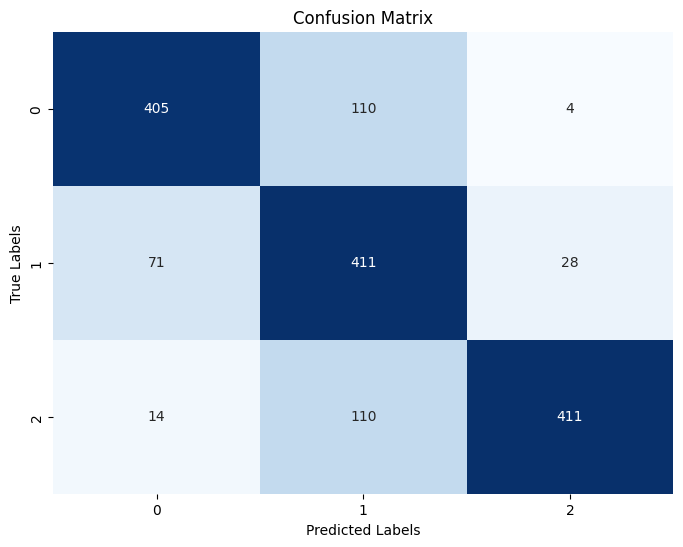
print("\nClassification Report:")

print(class\_report)

print("\nF1 Score:", f1)

print("Recall:", recall)

print("Accuracy:", accuracy)



Classification Report:

precision recall f1-score support

0 0.83 0.78 0.80 519

1 0.65 0.81 0.72 510

2 0.93 0.77 0.84 535

accuracy 0.78 1564

macro avg 0.80 0.78 0.79 1564

weighted avg 0.80 0.78 0.79 1564

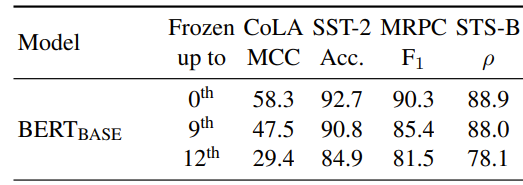
F1 Score: 0.7888218434705407

Recall: 0.7845268542199488

Accuracy: 0.7845268542199488

بر اساس نتایج گزارش شده دقت مدل در حدود 78 درصد گزارش شده که افت منطقی به نظر می‌رسد. علت:

Table 1جدول 1. جدول دقت Freeze بر اساس مقاله



بر اساس نتایج مقاله بدون Freeze ما دقت 92 گرفته و پس از 9 لایه فریز دو درصد دقت کم شده است. ما نیز به نتایج مشابه دست پیدا کرده ایم در حالت Fine tune دقت 80 گرفته در این حالت دقت 78 شده است.

train\_loss = history2.history['loss']

val\_loss = history2.history['val\_loss']

train\_accuracy = history2.history['accuracy']

val\_accuracy = history2.history['val\_accuracy']

fig, axes = plt.subplots(1, 2, figsize=(16, 5))

axes[0].plot(train\_loss, label='Training Loss')

axes[0].plot(val\_loss, label='Validation Loss')

axes[0].set\_title('Training and Validation Loss')

axes[0].set\_xlabel('Epoch')

axes[0].set\_ylabel('Loss')

axes[0].legend()

axes[1].plot(train\_accuracy, label='Training Accuracy')

axes[1].plot(val\_accuracy, label='Validation Accuracy')

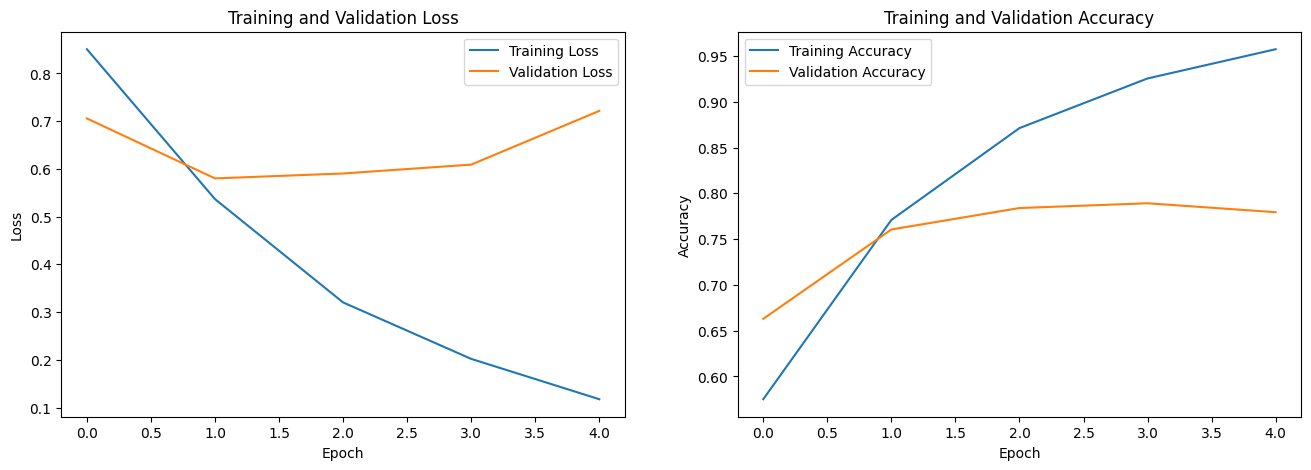
axes[1].set\_title('Training and Validation Accuracy')

axes[1].set\_xlabel('Epoch')

axes[1].set\_ylabel('Accuracy')

axes[1].legend()

plt.show()



* فریز کردن همه لایه ها به جز لایه اخر و Embedding

train\_data = pd.read\_csv('/content/FarsTail\_dataset/FarsTail-master/data/Train-word.csv', sep='\t')

val\_data = pd.read\_csv('/content/FarsTail\_dataset/FarsTail-master/data/Val-word.csv', sep='\t')

test\_data = pd.read\_csv('/content/FarsTail\_dataset/FarsTail-master/data/Test-word.csv', sep='\t')

cleaned\_train\_data = clean\_dataframe(train\_data, [premise\_column, hypothesis\_column])

cleaned\_val\_data = clean\_dataframe(val\_data, [premise\_column, hypothesis\_column])

cleaned\_test\_data = clean\_dataframe(test\_data, [premise\_column, hypothesis\_column])

tokenizer = AutoTokenizer.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased")

label\_map = {'e': 0, 'c': 1, 'n': 2}

train\_data['label'] = cleaned\_train\_data['label'].map(label\_map)

val\_data['label'] = cleaned\_val\_data['label'].map(label\_map)

test\_data['label'] = cleaned\_test\_data['label'].map(label\_map)

train\_encodings = tokenizer(list(train\_data['premise']), list(train\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='tf')

val\_encodings = tokenizer(list(val\_data['premise']), list(val\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='tf')

test\_encodings = tokenizer(list(test\_data['premise']), list(test\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='tf')

train\_dataset = tf.data.Dataset.from\_tensor\_slices((

    {k: train\_encodings[k] for k in train\_encodings},

    train\_data['label']

)).shuffle(len(train\_data)).batch(32)

val\_dataset = tf.data.Dataset.from\_tensor\_slices((

    {k: val\_encodings[k] for k in val\_encodings},

    val\_data['label']

)).batch(32)

test\_dataset = tf.data.Dataset.from\_tensor\_slices((

    {k: test\_encodings[k] for k in test\_encodings},

    test\_data['label']

)).batch(32)

from transformers import TFAutoModelForSequenceClassification

model3 = TFAutoModelForSequenceClassification.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased", num\_labels=3)

bert\_layer = model3.get\_layer('bert')

*# Freeze all layers except the last layer and the embedding layer*

last\_layer\_index = len(bert\_layer.encoder.layer) - 1

for i, layer in enumerate(bert\_layer.encoder.layer):

    if i != last\_layer\_index and 'embeddings' not in layer.name:

        layer.trainable = False

optimizer = tf.keras.optimizers.Adam(learning\_rate=5e-5)

loss = tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True)

metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')

model3.compile(optimizer=optimizer, loss=loss, metrics=[metric])

from transformers import TFAutoModelForSequenceClassification

model3 = TFAutoModelForSequenceClassification.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased", num\_labels=3)

bert\_layer = model3.get\_layer('bert')

*# Freeze all layers except the last layer and the embedding layer*

last\_layer\_index = len(bert\_layer.encoder.layer) - 1

for i, layer in enumerate(bert\_layer.encoder.layer):

    if i != last\_layer\_index and 'embeddings' not in layer.name:

        layer.trainable = False

optimizer = tf.keras.optimizers.Adam(learning\_rate=5e-5)

loss = tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True)

metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')

model3.compile(optimizer=optimizer, loss=loss, metrics=[metric])

history3 = model3.fit(

    train\_dataset,

    validation\_data=val\_dataset,

    epochs=5,

    batch\_size=9

)

Epoch 1/5

228/228 [==============================] - 247s 985ms/step - loss: 0.9233 - accuracy: 0.5337 - val\_loss: 0.7569 - val\_accuracy: 0.6207

Epoch 2/5

228/228 [==============================] - 192s 841ms/step - loss: 0.6816 - accuracy: 0.6795 - val\_loss: 0.7406 - val\_accuracy: 0.6493

Epoch 3/5

228/228 [==============================] - 191s 840ms/step - loss: 0.4880 - accuracy: 0.7759 - val\_loss: 0.7015 - val\_accuracy: 0.7183

Epoch 4/5

228/228 [==============================] - 192s 841ms/step - loss: 0.3307 - accuracy: 0.8628 - val\_loss: 0.7221 - val\_accuracy: 0.7404

Epoch 5/5

228/228 [==============================] - 192s 841ms/step - loss: 0.2320 - accuracy: 0.9092 - val\_loss: 0.8321 - val\_accuracy: 0.7300

test\_predictions = []

test\_labels = []

for input\_ids, labels in test\_dataset:

    outputs = model3(input\_ids=input\_ids)

    logits = outputs.logits

    predicted = np.argmax(logits, axis=1)

    test\_predictions.extend(predicted)

    test\_labels.extend(labels)

test\_predictions = np.array(test\_predictions)

test\_labels = np.array(test\_labels)

conf\_matrix = confusion\_matrix(test\_labels, test\_predictions)

class\_report = classification\_report(test\_labels, test\_predictions)

f1 = f1\_score(test\_labels, test\_predictions, average='weighted')

recall = recall\_score(test\_labels, test\_predictions, average='weighted')

accuracy = accuracy\_score(test\_labels, test\_predictions)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

*# Print the results*

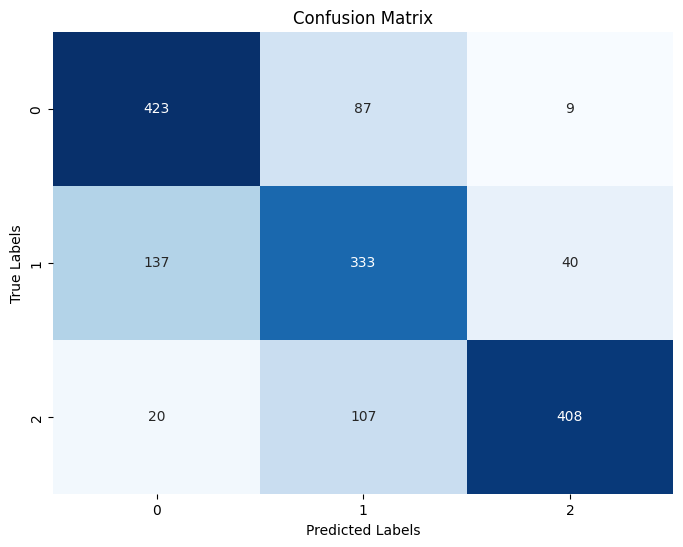
print("\nClassification Report:")

print(class\_report)

print("\nF1 Score:", f1)

print("Recall:", recall)

print("Accuracy:", accuracy)



Classification Report:

precision recall f1-score support

0 0.73 0.82 0.77 519

1 0.63 0.65 0.64 510

2 0.89 0.76 0.82 535

accuracy 0.74 1564

macro avg 0.75 0.74 0.74 1564

weighted avg 0.75 0.74 0.75 1564

F1 Score: 0.7462551226620019

Recall: 0.7442455242966752

Accuracy: 0.7442455242966752

بر اساس نتایج دریافتی دقت مدل در حدود 75 درصد روی داده تست گزارش می‌شود که افت دقت منطقی به نظر می‌رسد.

train\_loss = history3.history['loss']

val\_loss = history3.history['val\_loss']

train\_accuracy = history3.history['accuracy']

val\_accuracy = history3.history['val\_accuracy']

fig, axes = plt.subplots(1, 2, figsize=(16, 5))

axes[0].plot(train\_loss, label='Training Loss')

axes[0].plot(val\_loss, label='Validation Loss')

axes[0].set\_title('Training and Validation Loss')

axes[0].set\_xlabel('Epoch')

axes[0].set\_ylabel('Loss')

axes[0].legend()

axes[1].plot(train\_accuracy, label='Training Accuracy')

axes[1].plot(val\_accuracy, label='Validation Accuracy')

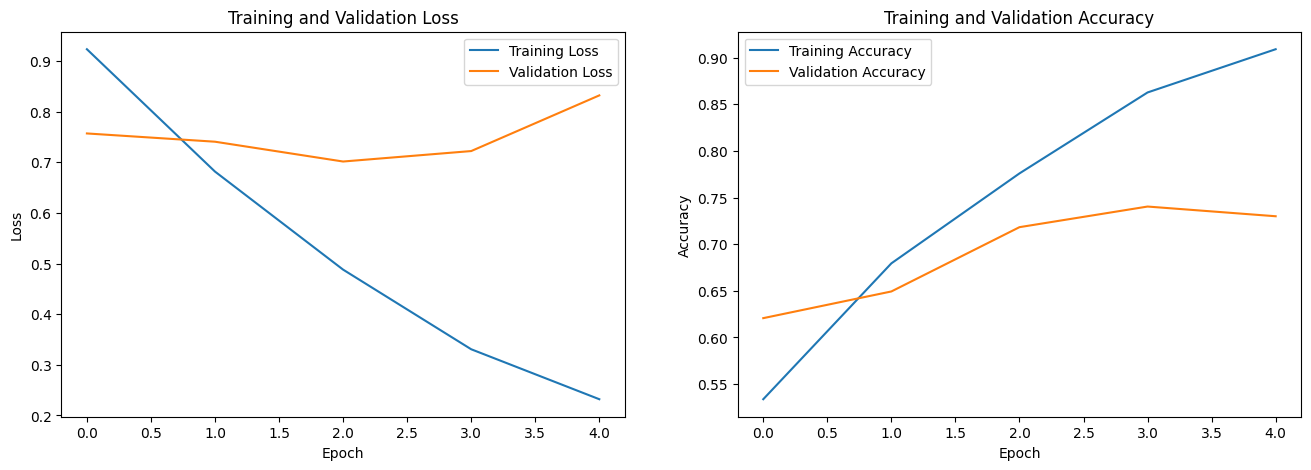
axes[1].set\_title('Training and Validation Accuracy')

axes[1].set\_xlabel('Epoch')

axes[1].set\_ylabel('Accuracy')

axes[1].legend()

plt.show()



### ۲-۱-۴. تنظیم دقیق مدل برای لایه های میانی

train\_data = pd.read\_csv('/content/FarsTail\_dataset/FarsTail-master/data/Train-word.csv', sep='\t')

val\_data = pd.read\_csv('/content/FarsTail\_dataset/FarsTail-master/data/Val-word.csv', sep='\t')

test\_data = pd.read\_csv('/content/FarsTail\_dataset/FarsTail-master/data/Test-word.csv', sep='\t')

import torch

from transformers import AutoTokenizer

*# Load and tokenize data*

tokenizer = AutoTokenizer.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased")

*# Convert labels to numerical format*

label\_map = {'e': 0, 'c': 1, 'n': 2}

train\_data['label'] = train\_data['label'].map(label\_map).astype('int64')  *# PyTorch uses int64 for labels*

val\_data['label'] = val\_data['label'].map(label\_map).astype('int64')

test\_data['label'] = test\_data['label'].map(label\_map).astype('int64')

*# Rest of the code remains the same as mentioned in the previous steps for tokenization and dataset creation*

*# Tokenize data*

train\_encodings = tokenizer(list(train\_data['premise']), list(train\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='pt')

val\_encodings = tokenizer(list(val\_data['premise']), list(val\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='pt')

test\_encodings = tokenizer(list(test\_data['premise']), list(test\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='pt')

ایجاد دیتاست کاستوم برای پایتورچ

*# Create PyTorch datasets*

class CustomDataset(torch.utils.data.Dataset):

      def \_\_init\_\_(self, encodings, labels):

          self.encodings = encodings

          self.labels = labels

      def \_\_getitem\_\_(self, idx):

          item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}

          item['labels'] = torch.tensor(self.labels[idx])

          return item

      def \_\_len\_\_(self):

          return len(self.labels)

train\_dataset = CustomDataset(train\_encodings, train\_data['label'])

val\_dataset = CustomDataset(val\_encodings, val\_data['label'])

test\_dataset = CustomDataset(test\_encodings, test\_data['label'])

train\_loader = torch.utils.data.DataLoader(train\_dataset, batch\_size=12, shuffle=True)

val\_loader = torch.utils.data.DataLoader(val\_dataset, batch\_size=12)

test\_loader = torch.utils.data.DataLoader(test\_dataset, batch\_size=12)

تغییر مدل BERT به حذف نه لایه ابتدایی

from transformers import BertModel

*# Load the entire pre-trained BERT model*

full\_bert\_model = BertModel.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased")

*# Get the individual layers from the BERT model*

bert\_layers = list(full\_bert\_model.children())

*# Select the first nine layers*

first\_nine\_layers = bert\_layers[:9]

class CustomBERTModel(torch.nn.Module):

    def \_\_init\_\_(self, first\_nine\_layers, hidden\_size, output\_classes):

        super(CustomBERTModel, self).\_\_init\_\_()

        self.first\_nine\_layers = torch.nn.ModuleList(first\_nine\_layers)

        self.classifier = torch.nn.Linear(hidden\_size, output\_classes)

    def forward(self, input\_ids, attention\_mask, token\_type\_ids=None, labels=None):

        output = input\_ids

        attentions = []

        for i, layer in enumerate(self.first\_nine\_layers):

            if isinstance(output, BaseModelOutputWithPastAndCrossAttentions):

                output = output.last\_hidden\_state  *# Assuming you want the last hidden state*

            if isinstance(layer, torch.nn.modules.activation.MultiheadAttention):

                output, attention = layer(output, attn\_mask=attention\_mask)

                attentions.append(attention)

            else:

                output = layer(output.clone().detach() if isinstance(output, torch.Tensor) else output)

        logits = self.classifier(output)  *# Consider using the entire sequence representation directly*

        if labels is not None:

            loss\_fct = torch.nn.CrossEntropyLoss()

            loss = loss\_fct(logits.view(-1, logits.size(-1)), labels.view(-1))  *# Reshape logits and labels for loss calculation*

            return loss

        else:

            return logits, attentions

from transformers.modeling\_outputs import BaseModelOutputWithPastAndCrossAttentions

import torch.optim as optim

import torch

hidden\_size = 768

output\_classes = 3

model4 = CustomBERTModel(first\_nine\_layers, hidden\_size, output\_classes)

optimizer = optim.Adam(model4.parameters(), lr=9e-5)  *# lr is the learning rate*

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model4.to(device)

import torch

import matplotlib.pyplot as plt

train\_losses = []

val\_losses = []

train\_accuracies = []

val\_accuracies = []

num\_epochs = 5

criterion = torch.nn.CrossEntropyLoss()

for epoch in range(num\_epochs):

    model4.train()

    train\_loss = 0.0

    correct\_train = 0

    total\_train = 0

    for batch in train\_loader:

        optimizer.zero\_grad()

        inputs = {key: val.to(device) for key, val in batch.items()}

        labels = inputs.pop('labels')

        outputs = model4(\*\*inputs)

        if isinstance(outputs, tuple):

            outputs = outputs[0]  *# Extracting the appropriate output tensor*

        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

        train\_loss += loss.item()

        \_, predicted = torch.max(outputs, 1)

        total\_train += labels.size(0)

        correct\_train += (predicted == labels).sum().item()

    avg\_train\_loss = train\_loss / len(train\_loader)

    train\_accuracy = correct\_train / total\_train

    train\_losses.append(avg\_train\_loss)

    train\_accuracies.append(train\_accuracy)

    model4.eval()

    val\_loss = 0.0

    correct\_val = 0

    total\_val = 0

    with torch.no\_grad():

        for batch in val\_loader:

            inputs = {key: val.to(device) for key, val in batch.items()}

            labels = inputs.pop('labels')

            outputs = model4(\*\*inputs)

            if isinstance(outputs, tuple):

                outputs = outputs[0]  *# Extracting the appropriate output tensor*

            loss = criterion(outputs, labels)

            val\_loss += loss.item()

            \_, predicted = torch.max(outputs, 1)

            total\_val += labels.size(0)

            correct\_val += (predicted == labels).sum().item()

    avg\_val\_loss = val\_loss / len(val\_loader)

    val\_accuracy = correct\_val / total\_val

    val\_losses.append(avg\_val\_loss)

    val\_accuracies.append(val\_accuracy)

    print(f"Epoch [{epoch + 1}/{num\_epochs}] - Train Loss: {avg\_train\_loss:.4f} | Val Loss: {avg\_val\_loss:.4f} | Train Acc: {train\_accuracy:.4f} | Val Acc: {val\_accuracy:.4f}")

Epoch [1/5] - Train Loss: 1.0506 | Val Loss: 0.9868 | Train Acc: 0.4370 | Val Acc: 0.4873

Epoch [2/5] - Train Loss: 1.0044 | Val Loss: 0.9815 | Train Acc: 0.4879 | Val Acc: 0.4951

Epoch [3/5] - Train Loss: 0.9826 | Val Loss: 0.9243 | Train Acc: 0.5017 | Val Acc: 0.5465

Epoch [4/5] - Train Loss: 0.9732 | Val Loss: 0.9103 | Train Acc: 0.5091 | Val Acc: 0.5615

Epoch [5/5] - Train Loss: 0.9656 | Val Loss: 0.9055 | Train Acc: 0.5168 | Val Acc: 0.5563

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)

plt.plot(train\_losses, label='Training Loss')

plt.plot(val\_losses, label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(train\_accuracies, label='Training Accuracy')

plt.plot(val\_accuracies, label='Validation Accuracy')

plt.xlabel('Epochs')

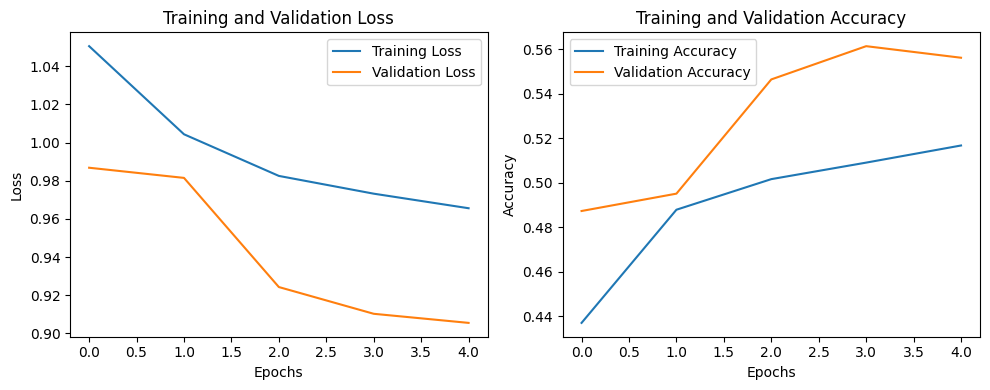
plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()



model4.eval()

test\_predictions = []

test\_labels = []

with torch.no\_grad():

    for batch in test\_loader:

        inputs = {key: val.to(device) for key, val in batch.items()}

        labels = inputs.pop('labels')

        outputs = model4(\*\*inputs)

        if isinstance(outputs, tuple):

            outputs = outputs[0]

        \_, predicted = torch.max(outputs, 1)

        test\_predictions.extend(predicted.cpu().numpy())

        test\_labels.extend(labels.cpu().numpy())

test\_predictions = np.array(test\_predictions)

test\_labels = np.array(test\_labels)

conf\_matrix = confusion\_matrix(test\_labels, test\_predictions)

class\_report = classification\_report(test\_labels, test\_predictions)

f1 = f1\_score(test\_labels, test\_predictions, average='weighted')

recall = recall\_score(test\_labels, test\_predictions, average='weighted')

accuracy = accuracy\_score(test\_labels, test\_predictions)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

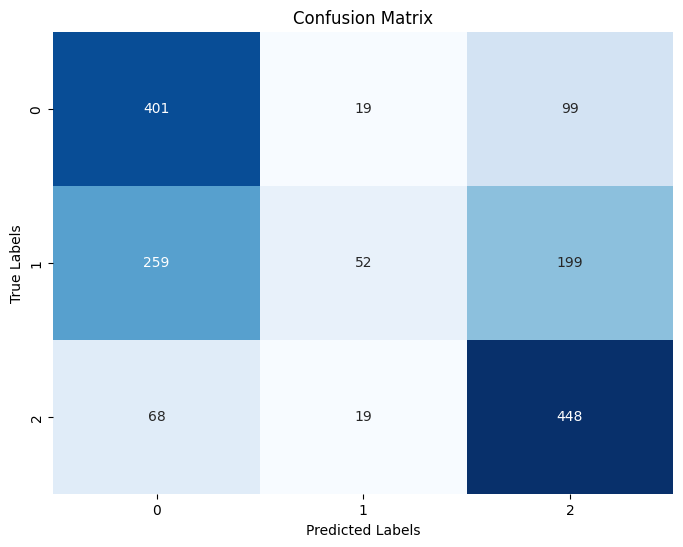
print("\nClassification Report:")

print(class\_report)

print("\nF1 Score:", f1)

print("Recall:", recall)

print("Accuracy:", accuracy)



Classification Report:

precision recall f1-score support

0 0.55 0.77 0.64 519

1 0.58 0.10 0.17 510

2 0.60 0.84 0.70 535

accuracy 0.58 1564

macro avg 0.58 0.57 0.51 1564

weighted avg 0.58 0.58 0.51 1564

F1 Score: 0.5092066175653848

Recall: 0.5760869565217391

Accuracy: 0.5760869565217391

بر اساس نتایج بدست آمده دقت مدل در حدود 60 درصد و نتایج f1 recall قابل قبول است. در این حالت ما افت دقت داشتیم.

### ۲-۱-۵. حذف Head ها در Attention مدل

* Drop 50% model attention از هر لایه
* از متد Prune\_heads استفاده شود

طبق مقاله دوم کافت چشمگیر عملکردی نخواهیم داشت.

import pandas as pd

train\_data = pd.read\_csv('/content/FarsTail\_dataset/FarsTail-master/data/Train-word.csv', sep='\t')

val\_data = pd.read\_csv('/content/FarsTail\_dataset/FarsTail-master/data/Val-word.csv', sep='\t')

test\_data = pd.read\_csv('/content/FarsTail\_dataset/FarsTail-master/data/Test-word.csv', sep='\t')

import torch

from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased")

label\_map = {'e': 0, 'c': 1, 'n': 2}

train\_data['label'] = train\_data['label'].map(label\_map).astype('int64')

val\_data['label'] = val\_data['label'].map(label\_map).astype('int64')

test\_data['label'] = test\_data['label'].map(label\_map).astype('int64')

train\_encodings = tokenizer(list(train\_data['premise']), list(train\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='pt')

val\_encodings = tokenizer(list(val\_data['premise']), list(val\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='pt')

test\_encodings = tokenizer(list(test\_data['premise']), list(test\_data['hypothesis']), truncation=True, padding='max\_length', max\_length=160, return\_tensors='pt')

class CustomDataset(torch.utils.data.Dataset):

    def \_\_init\_\_(self, encodings, labels):

        self.encodings = encodings

        self.labels = labels

    def \_\_getitem\_\_(self, idx):

        item = {key: val[idx].clone().detach() for key, val in self.encodings.items()}

        label = torch.zeros(num\_classes)

        label[self.labels[idx]] = 1

        item['labels'] = label

        return item

    def \_\_len\_\_(self):

        return len(self.labels)

train\_dataset = CustomDataset(train\_encodings, train\_data['label'])

val\_dataset = CustomDataset(val\_encodings, val\_data['label'])

test\_dataset = CustomDataset(test\_encodings, test\_data['label'])

train\_loader = torch.utils.data.DataLoader(train\_dataset, batch\_size=2, shuffle=True)

val\_loader = torch.utils.data.DataLoader(val\_dataset, batch\_size=2)

test\_loader = torch.utils.data.DataLoader(test\_dataset, batch\_size=2)

import torch

from transformers import BertModel, BertConfig

bert\_config = BertConfig.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased")

bert\_model = BertModel(bert\_config)

class CustomBERTModel(torch.nn.Module):

    def \_\_init\_\_(self, bert\_model, hidden\_size, output\_classes):

        super(CustomBERTModel, self).\_\_init\_\_()

        self.bert = bert\_model

        self.classifier = torch.nn.Linear(hidden\_size, output\_classes)

    def prune\_heads(self, heads\_proportion=0.5):

        for layer in self.bert.encoder.layer:

            total\_heads = layer.attention.self.num\_attention\_heads

            heads\_to\_prune = int(total\_heads \* (1 - heads\_proportion))

            heads\_indices = torch.randperm(total\_heads)[:heads\_to\_prune]

*# Prune selected heads by modifying weights*

            for head\_idx in heads\_indices:

                layer.attention.self.query.weight.data[head\_idx] = 0.0

                layer.attention.self.key.weight.data[head\_idx] = 0.0

                layer.attention.self.value.weight.data[head\_idx] = 0.0

    def forward(self, input\_ids, attention\_mask, token\_type\_ids=None, labels=None):

        output = self.bert(input\_ids=input\_ids, attention\_mask=attention\_mask, token\_type\_ids=token\_type\_ids)

        last\_hidden\_state = output.last\_hidden\_state

        logits = self.classifier(last\_hidden\_state)

        if labels is not None:

            loss\_fct = torch.nn.CrossEntropyLoss()

            loss = loss\_fct(logits.view(-1, logits.size(-1)), labels.view(-1))

            return loss

        else:

            return logits

from transformers import AutoModel

bert\_model = AutoModel.from\_pretrained("HooshvareLab/bert-base-parsbert-uncased")

num\_classes = 3

model5 = CustomBERTModel(bert\_model, hidden\_size=bert\_model.config.hidden\_size, output\_classes=num\_classes)

model5.prune\_heads(heads\_proportion=0.5)

optimizer = torch.optim.Adam(model5.parameters(), lr=5e-5)

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

model5.to(device)

import torch

from torch import nn, optim

import matplotlib.pyplot as plt

criterion = nn.CrossEntropyLoss()

num\_epochs = 5

train\_losses = []

val\_losses = []

train\_accuracies = []

val\_accuracies = []

for epoch in range(num\_epochs):

    model5.train()

    train\_loss = 0.0

    correct\_train = 0

    total\_train = 0

    for batch in train\_loader:

        optimizer.zero\_grad()

        inputs = {key: val.to(device) for key, val in batch.items() if key != 'labels'}

        labels = batch['labels'].to(device).long()

        outputs = model5(\*\*inputs)

        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

        train\_loss += loss.item()

        \_, predicted = torch.max(outputs, 1)

        total\_train += labels.size(0)

        correct\_train += (predicted == labels).sum().item()

    avg\_train\_loss = train\_loss / len(train\_loader)

    train\_accuracy = correct\_train / total\_train

    train\_losses.append(avg\_train\_loss)

    train\_accuracies.append(train\_accuracy)

    print(f"Epoch [{epoch + 1}/{num\_epochs}] - Train Loss: {avg\_train\_loss:.4f} | Train Accuracy: {train\_accuracy:.4f}")

*# Validation*

    model5.eval()

    val\_loss = 0.0

    correct\_val = 0

    total\_val = 0

    with torch.no\_grad():

        for batch in val\_loader:

            inputs = {key: val.to(device) for key, val in batch.items() if key != 'labels'}

            labels = batch['labels'].to(device).long()

            outputs = model5(\*\*inputs)

            loss = criterion(outputs, labels)

            val\_loss += loss.item()

            \_, predicted = torch.max(outputs, 1)

            total\_val += labels.size(0)

            correct\_val += (predicted == labels).sum().item()

    avg\_val\_loss = val\_loss / len(val\_loader)

    val\_accuracy = correct\_val / total\_val

    val\_losses.append(avg\_val\_loss)

    val\_accuracies.append(val\_accuracy)

    print(f"Epoch [{epoch + 1}/{num\_epochs}] - Validation Loss: {avg\_val\_loss:.4f} | Validation Accuracy: {val\_accuracy:.4f}")

Epoch [1/5] - Train Loss: 0.5724 | Train Accuracy: 2.1233

- Validation Loss: 0.5248 | Validation Accuracy: 2.2453

Epoch [2/5] - Train Loss: 0.4844 | Train Accuracy: 2.2541

- Validation Loss: 0.4712 | Validation Accuracy: 2.2869

Epoch [3/5] - Train Loss: 0.4787 | Train Accuracy: 2.2623

- Validation Loss: 0.5218 | Validation Accuracy: 2.2602

Epoch [4/5] - Train Loss: 0.4941 | Train Accuracy: 2.2516

- Validation Loss: 0.7536 | Validation Accuracy: 2.1340

Epoch [5/5] - Train Loss: 0.7736 | Train Accuracy: 1.9108

- Validation Loss: 0.6133 | Validation Accuracy: 2.0241

model5.eval()

test\_loss = 0.0

correct\_test = 0

total\_test = 0

with torch.no\_grad():

    for batch in test\_loader:

        inputs = {key: val.to(device) for key, val in batch.items() if key != 'labels'}

        labels = batch['labels'].to(device).long()

        outputs = model5(\*\*inputs)

        loss = criterion(outputs, labels)

        test\_loss += loss.item()

        \_, predicted = torch.max(outputs, 1)

        total\_test += labels.size(0)

        correct\_test += (predicted == labels).sum().item()

avg\_test\_loss = test\_loss / len(test\_loader)

test\_accuracy = correct\_test / total\_test

print(f"Test Loss: {avg\_test\_loss:.4f} | Test Accuracy: {test\_accuracy:.4f}")

Test Loss: 0.6147 | Test Accuracy: 2.0364

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)

plt.plot(train\_losses, label='Training Loss')

plt.plot(val\_losses, label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(train\_accuracies, label='Training Accuracy')

plt.plot(val\_accuracies, label='Validation Accuracy')

plt.xlabel('Epochs')

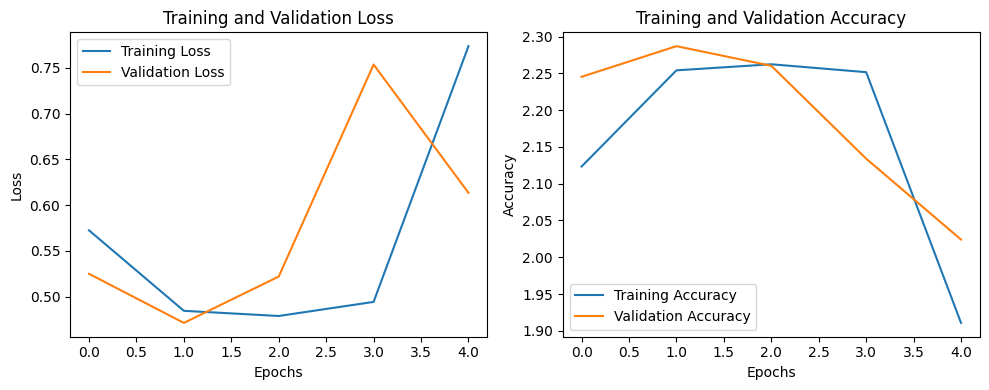
plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()



import numpy as np

from sklearn.metrics import confusion\_matrix, classification\_report, f1\_score, recall\_score, accuracy\_score

import seaborn as sns

from sklearn.metrics import multilabel\_confusion\_matrix

*# Set the model to evaluation mode*

model5.eval()

*# Lists to store predictions and true labels for the test set*

test\_predictions = []

test\_labels = []

*# Iterate over the test loader*

with torch.no\_grad():

    for batch in test\_loader:

        inputs = {key: val.to(device) for key, val in batch.items() if key != 'labels'}

        labels = batch['labels'].to(device).long()

        outputs = model5(\*\*inputs)

        \_, predicted = torch.max(outputs, 1)

        test\_predictions.extend(predicted.cpu().numpy())

        test\_labels.extend(labels.cpu().numpy())

*# Convert to numpy arrays*

test\_predictions = np.array(test\_predictions)

test\_labels = np.array(test\_labels)

*# Compute confusion matrix, classification report, f1 score, recall, and accuracy*

conf\_matrix = multilabel\_confusion\_matrix(test\_labels, test\_predictions)

class\_report = classification\_report(test\_labels, test\_predictions)

f1 = f1\_score(test\_labels, test\_predictions, average='weighted')

recall = recall\_score(test\_labels, test\_predictions, average='weighted')

accuracy = accuracy\_score(test\_labels, test\_predictions)

*# Plot the confusion matrix*

plt.figure(figsize=(8, 6))

for i, matrix in enumerate(conf\_matrix):

    plt.subplot(2, 2, i + 1)

    sns.heatmap(matrix, annot=True, fmt='d', cmap='Blues', cbar=False)

    plt.title(f'Confusion Matrix - Label {i}')

    plt.xlabel('Predicted Labels')

    plt.ylabel('True Labels')

plt.tight\_layout()

plt.show()

*# Print the results*

*# Note: You might need to iterate over each label separately for F1, recall, and accuracy*

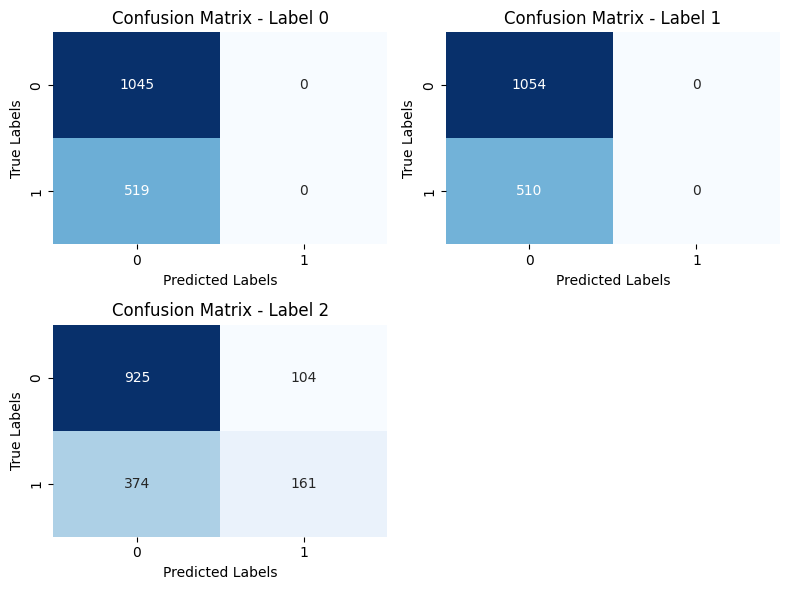
for i in range(len(conf\_matrix)):

    print(f"\nMetrics for Label {i}:")

    print("F1 Score:", f1\_score(test\_labels[:, i], test\_predictions[:, i]))

    print("Recall:", recall\_score(test\_labels[:, i], test\_predictions[:, i]))

    print("Accuracy:", accuracy\_score(test\_labels[:, i], test\_predictions[:, i]))



Metrics for Label 0:

F1 Score: 0.0

Recall: 0.0

Accuracy: 0.6681585677749361

Metrics for Label 1:

F1 Score: 0.0

Recall: 0.0

Accuracy: 0.6739130434782609

Metrics for Label 2:

F1 Score: 0.4025

Recall: 0.30093457943925234

Accuracy: 0.6943734015345269

بر اساس نتیجه مدل ساخته شده این حالت مدل برای Label 2 خروجی f1 recall منطقی تری نشان میدهد که در موارد اول و صفر صدق نمی کند. همینطور دقت مدل به اندازه تفریبا 70 درصد در دادگان تست رسیده که قابل قبول و از بخش قبلی بالاتر است وافت چشمگیر عملکردی نداشته ایم.

همینطور جدول زیر نتایج مقاله در مورد Prune مدل Bert رانشان می‌دهد که با یافته های ما نزدیکی دارد:

Table 2جدول 2. تاثیر prune مدل BERT

