3_5 Configuration Refinement

April 13, 2025

0.1 Packages, Library Imports, File Mounts, & Data Imports ** Run All **

[1]:
```
!pip install -q transformers
!pip install -q torchinfo
!pip install -q datasets
!pip install -q evaluate
!pip install -q nltk
!pip install -q contractions
!pip install -q hf_xet
!pip install -q sentencepiece
```

```
                         491.2/491.2 kB
9.7 MB/s eta 0:00:00
                         116.3/116.3 kB
10.7 MB/s eta 0:00:00
                         183.9/183.9 kB
15.8 MB/s eta 0:00:00
                         143.5/143.5 kB
13.2 MB/s eta 0:00:00
                         194.8/194.8 kB
17.5 MB/s eta 0:00:00
                         84.0/84.0 kB
401.6 kB/s eta 0:00:00
                         289.9/289.9 kB
6.3 MB/s eta 0:00:00
                         118.3/118.3 kB
10.2 MB/s eta 0:00:00
                         53.8/53.8 MB
25.4 MB/s eta 0:00:00
```

[2]:
```
!sudo apt-get update
! sudo apt-get install tree
```

```
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:2 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:4 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
[3,632 B]
```

```
Get:5 https://r2u.stat.illinois.edu/ubuntu jammy InRelease [6,555 B]
Hit:6 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:7 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
[18.1 kB]
Hit:8 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
Get:9 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages
[1,542 kB]
Get:10 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages
[2,788 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [3,099
kB]
Get:12 https://r2u.stat.illinois.edu/ubuntu jammy/main amd64 Packages [2,690 kB]
Get:13 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages
[1,243 kB]
Get:14 https://r2u.stat.illinois.edu/ubuntu jammy/main all Packages [8,833 kB]
Get:15 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy/main amd64
Packages [34.3 kB]
Fetched 20.5 MB in 2s (11.0 MB/s)
Reading package lists… Done
W: Skipping acquire of configured file 'main/source/Sources' as repository
'https://r2u.stat.illinois.edu/ubuntu jammy InRelease' does not seem to provide
it (sources.list entry misspelt?)
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
The following NEW packages will be installed:
  tree
0 upgraded, 1 newly installed, 0 to remove and 2 not upgraded.
Need to get 47.9 kB of archives.
After this operation, 116 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy/universe amd64 tree amd64 2.0.2-1
[47.9 kB]
Fetched 47.9 kB in 0s (290 kB/s)
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based
frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 78,
<> line 1.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Selecting previously unselected package tree.
(Reading database … 122158 files and directories currently installed.)
Preparing to unpack …/tree_2.0.2-1_amd64.deb …
Unpacking tree (2.0.2-1) …
Setting up tree (2.0.2-1) …
Processing triggers for man-db (2.10.2-1) …
```

[3]:
```
#@title Imports
import nltk
from nltk.tokenize import RegexpTokenizer
import sentencepiece
import contractions
import spacy


import evaluate
from datasets import load_dataset, Dataset, DatasetDict

import torch
import torch.nn as nn
from torchinfo import summary

import transformers
from transformers import AutoTokenizer, AutoModel,␣
 ↪AutoModelForSequenceClassification, TrainingArguments, Trainer, BertConfig,␣
 ↪BertForSequenceClassification

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report,␣
 ↪precision_recall_fscore_support, accuracy_score

import json
import datetime
import zoneinfo
from datetime import datetime
```

[4]:
```
# @title Mount Google Drive
```

[5]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[6]:
```
dir_root = '/content/drive/MyDrive/266-final/'
# dir_data = '/content/drive/MyDrive/266-final/data/'
# dir_data = '/content/drive/MyDrive/266-final/data/se21-t1-comp-lex-master/'
```

```
dir_data = '/content/drive/MyDrive/266-final/data/266-comp-lex-master'
dir_models = '/content/drive/MyDrive/266-final/models/'
dir_results = '/content/drive/MyDrive/266-final/results/'
log_filename = "experiment_runs.txt"
log_filepath = os.path.join(dir_results, log_filename)
```

[7]:
```
wandbai_api_key = ""
```

[8]:
```
!tree /content/drive/MyDrive/266-final/data/266-comp-lex-master/
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/
  fe-test-labels
      test_multi_df.csv
      test_single_df.csv
  fe-train
      train_multi_df.csv
      train_single_df.csv
  fe-trial-val
      trial_val_multi_df.csv
      trial_val_single_df.csv
  test-labels
      lcp_multi_test.tsv
      lcp_single_test.tsv
  train
      lcp_multi_train.tsv
      lcp_single_train.tsv
  trial
       lcp_multi_trial.tsv
       lcp_single_trial.tsv

6 directories, 12 files
```

[9]:
```
!ls -R /content/drive/MyDrive/266-final/data/266-comp-lex-master/
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/:
fe-test-labels  fe-train  fe-trial-val  test-labels  train  trial

/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-test-labels:
test_multi_df.csv  test_single_df.csv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-train:
train_multi_df.csv  train_single_df.csv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-trial-val:
trial_val_multi_df.csv  trial_val_single_df.csv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/test-labels:
lcp_multi_test.tsv  lcp_single_test.tsv
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/train:
lcp_multi_train.tsv  lcp_single_train.tsv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/trial:
lcp_multi_trial.tsv  lcp_single_trial.tsv
```

[10]:
```
!tree /content/drive/MyDrive/266-final/data/266-comp-lex-master/
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/
  fe-test-labels
      test_multi_df.csv
      test_single_df.csv
  fe-train
      train_multi_df.csv
      train_single_df.csv
  fe-trial-val
      trial_val_multi_df.csv
      trial_val_single_df.csv
  test-labels
      lcp_multi_test.tsv
      lcp_single_test.tsv
  train
      lcp_multi_train.tsv
      lcp_single_train.tsv
  trial
       lcp_multi_trial.tsv
       lcp_single_trial.tsv

6 directories, 12 files
```

[11]:
```
#@title Import Data
```

[12]:
```
df_names = [
    "train_single_df",
    "train_multi_df",
    "trial_val_single_df",
    "trial_val_multi_df",
    "test_single_df",
    "test_multi_df"
]

loaded_dataframes = {}

for df_name in df_names:
    if "train" in df_name:
        subdir = "fe-train"
    elif "trial_val" in df_name:
```

```
        subdir = "fe-trial-val"
    elif "test" in df_name:
        subdir = "fe-test-labels"
    else:
        subdir = None

    if subdir:
        read_path = os.path.join(dir_data, subdir, f"{df_name}.csv")
        loaded_df = pd.read_csv(read_path)
        loaded_dataframes[df_name] = loaded_df
        print(f"Loaded {df_name} from {read_path}")

# for df_name, df in loaded_dataframes.items():
#     print(f"\n>>> {df_name} shape: {df.shape}")
#     if 'binary_complexity' in df.columns:
#         print(df['binary_complexity'].value_counts())
#         print(df.info())
#         print(df.head())

for df_name, df in loaded_dataframes.items():
    globals()[df_name] = df
    print(f"{df_name} loaded into global namespace.")
```

```
Loaded train_single_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-train/train_single_df.csv
Loaded train_multi_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-train/train_multi_df.csv
Loaded trial_val_single_df from /content/drive/MyDrive/266-final/data/266-comp-
lex-master/fe-trial-val/trial_val_single_df.csv
Loaded trial_val_multi_df from /content/drive/MyDrive/266-final/data/266-comp-
lex-master/fe-trial-val/trial_val_multi_df.csv
Loaded test_single_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-test-labels/test_single_df.csv
Loaded test_multi_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-test-labels/test_multi_df.csv
train_single_df loaded into global namespace.
train_multi_df loaded into global namespace.
trial_val_single_df loaded into global namespace.
trial_val_multi_df loaded into global namespace.
test_single_df loaded into global namespace.
test_multi_df loaded into global namespace.
```

- Functional tests pass, we can proceed with Baseline Modeling

0.2 Experiments

0.2.1 Helper Functions ** Run **

[13]:
```
MODEL_LINEAGE = {}

def get_model_and_tokenizer(
    remote_model_name: str = None,
    local_model_path: str = None,
    config=None
):
    """
    Loads the model & tokenizer for classification.
    If 'local_model_path' is specified, load from that path.
    Otherwise, fall back to 'remote_model_name'.

    Optional: 'config' can be a custom BertConfig/AutoConfig object
              to override certain configuration parameters.

    Records complete traceable lineage in the global MODEL_LINEAGE.
    """
    global MODEL_LINEAGE

    if local_model_path:
        print(f"Loading from local path: {local_model_path}")
        tokenizer = AutoTokenizer.from_pretrained(local_model_path)

        # If a config object is provided, we pass it to from_pretrained.
        # Otherwise, it just uses the config that is part of local_model_path.
        if config is not None:
            model = AutoModelForSequenceClassification.from_pretrained(
                local_model_path,
                config=config
            )
        else:
            model = AutoModelForSequenceClassification.
 ↪from_pretrained(local_model_path)

        MODEL_LINEAGE = {
            "type": "offline_checkpoint",
            "path": local_model_path,
            "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        }
    elif remote_model_name:
        print(f"Loading from Hugging Face model: {remote_model_name}")
        tokenizer = AutoTokenizer.from_pretrained(remote_model_name)

        if config is not None:
```

```
            model = AutoModelForSequenceClassification.from_pretrained(
                remote_model_name,
                config=config
            )
        else:
            model = AutoModelForSequenceClassification.
 ↪from_pretrained(remote_model_name)

        MODEL_LINEAGE = {
            "type": "huggingface_hub",
            "path": remote_model_name,
            "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        }
    else:
        raise ValueError("You must provide either a remote_model_name or a␣
 ↪local_model_path!")

    return model, tokenizer
```

[14]:
```
def freeze_unfreeze_layers(model, layers_to_unfreeze=None):
    """
    Toggles requires_grad = False for all parameters
    except for those whose names contain any string in layers_to_unfreeze.
    By default, always unfreeze classifier/heads.
    """
    if layers_to_unfreeze is None:
        layers_to_unfreeze = ["classifier.", "pooler."]

    for name, param in model.named_parameters():
        if any(substring in name for substring in layers_to_unfreeze):
            param.requires_grad = True
        else:
            param.requires_grad = False
```

[15]:
```
def encode_examples(examples, tokenizer, text_col, max_length=256):
    """
    Tokenizes a batch of texts from 'examples[text_col]' using the given␣
 ↪tokenizer.
    Returns a dict with 'input_ids', 'attention_mask', etc.
    """
    texts = examples[text_col]
    encoded = tokenizer(
        texts,
        truncation=True,
        padding='max_length',
        max_length=max_length
    )
```

```
    return encoded
```

[16]:
```
def prepare_dataset(df, tokenizer, text_col, label_col, max_length=256):
    """
    Converts a Pandas DataFrame to a Hugging Face Dataset,
    then applies 'encode_examples' to tokenize.
    """
    dataset = Dataset.from_pandas(df)

    dataset = dataset.map(
        lambda batch: encode_examples(batch, tokenizer, text_col, max_length),
        batched=True
    )

    dataset = dataset.rename_column(label_col, "labels")
    dataset.set_format(type='torch',
                       columns=['input_ids', 'attention_mask', 'labels'])
    return dataset
```

[17]:
```
def compute_metrics(eval_pred):
    """
    Computes classification metrics, including accuracy, precision, recall, and␣
 ↪F1.
    """
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)

    metric_accuracy  = evaluate.load("accuracy")
    metric_precision = evaluate.load("precision")
    metric_recall    = evaluate.load("recall")
    metric_f1        = evaluate.load("f1")

    accuracy_result  = metric_accuracy.compute(predictions=preds,␣
 ↪references=labels)
    precision_result = metric_precision.compute(predictions=preds,␣
 ↪references=labels, average="binary")
    recall_result    = metric_recall.compute(predictions=preds,␣
 ↪references=labels, average="binary")
    f1_result        = metric_f1.compute(predictions=preds, references=labels,␣
 ↪average="binary")

    return {
        "accuracy"       : accuracy_result["accuracy"],
        "precision": precision_result["precision"],
        "recall"   : recall_result["recall"],
        "f1"       : f1_result["f1"]
    }
```

[18]:
```python
def gather_config_details(model):
    """
    Enumerates every attribute in model.config
    """
    config_items = {}
    for attr_name, attr_value in vars(model.config).items():
        config_items[attr_name] = attr_value
    return config_items

def gather_model_details(model):
    """
    Extracts total layers, total params, trainable params, and activation␣
 ↪function
    from a Transformers model. Adjust logic as needed for different␣
 ↪architectures.
    """
    details = {}

    try:
        total_params = model.num_parameters()
        trainable_params = model.num_parameters(only_trainable=True)
    except AttributeError:
        all_params = list(model.parameters())
        total_params = sum(p.numel() for p in all_params)
        trainable_params = sum(p.numel() for p in all_params if p.requires_grad)

    details["model_total_params"] = total_params
    details["model_trainable_params"] = trainable_params

    if hasattr(model, "bert") and hasattr(model.bert, "pooler"):
        act_obj = getattr(model.bert.pooler, "activation", None)
        details["pooler_activation_function"] = act_obj.__class__.__name__ if␣
 ↪act_obj else "N/A"
    else:
        details["pooler_activation_function"] = "N/A"

    details["config_attributes"] = gather_config_details(model)
    return details

def gather_all_run_metrics(trainer, train_dataset=None, val_dataset=None,␣
 ↪test_dataset=None):
    """
    Gathers final training metrics, final validation metrics, final test␣
 ↪metrics.
    Instead of only parsing the final train_loss from the log, we also do a full
    trainer.evaluate(train_dataset) to get the same set of metrics that val/
 ↪test have.
```

```
    """
    results = {}

    if train_dataset is not None:
        train_metrics = trainer.evaluate(train_dataset)
        for k, v in train_metrics.items():
            results[f"train_{k}"] = v
    else:
        results["train_metrics"] = "No train dataset provided"

    if val_dataset is not None:
        val_metrics = trainer.evaluate(val_dataset)
        for k, v in val_metrics.items():
            results[f"val_{k}"] = v
    else:
        results["val_metrics"] = "No val dataset provided"

    if test_dataset is not None:
        test_metrics = trainer.evaluate(test_dataset)
        for k, v in test_metrics.items():
            results[f"test_{k}"] = v
    else:
        results["test_metrics"] = "No test dataset provided"

    return results

# def log_experiment_results_json(experiment_meta, model_details, run_metrics,␣
 ↪log_file):
#     """
#     Logs experiment metadata, model details, and metrics to a JSON lines file.
#     Automatically concatenates the 'checkpoint_path' to the 'model_lineage'.
#     """
#     checkpoint_path = model_details.get("checkpoint_path")
#     if checkpoint_path:
#         if "model_lineage" not in model_details:
#             model_details["model_lineage"] = ""
#         if model_details["model_lineage"]:
#             model_details["model_lineage"] += " -> "
#         model_details["model_lineage"] += checkpoint_path

#     record = {
#         "timestamp": str(datetime.datetime.now()),
#         "experiment_meta": experiment_meta,
#         "model_details": model_details,
#         "run_metrics": run_metrics
#     }
```

```
#     with open(log_file, "a", encoding="utf-8") as f:
#         json.dump(record, f)
#         f.write("\n")

def log_experiment_results_json(experiment_meta, model_details, run_metrics,␣
 ↪log_file):
    """
    Logs experiment metadata, model details, and metrics to a JSON lines file.
    Automatically concatenates the 'checkpoint_path' to the 'model_lineage'
    and uses Pacific time for the timestamp.
    """
    checkpoint_path = model_details.get("checkpoint_path")
    if checkpoint_path:
        if "model_lineage" not in model_details:
            model_details["model_lineage"] = ""
        if model_details["model_lineage"]:
            model_details["model_lineage"] += " -> "
        model_details["model_lineage"] += checkpoint_path

    pacific_time = datetime.now(zoneinfo.ZoneInfo("America/Los_Angeles")) #␣
 ↪update to support pacific time
    timestamp_str = pacific_time.isoformat()

    record = {
        "timestamp": timestamp_str,
        "experiment_meta": experiment_meta,
        "model_details": model_details,
        "run_metrics": run_metrics
    }

    with open(log_file, "a", encoding="utf-8") as f:
        json.dump(record, f)
        f.write("\n")
```

0.2.2 Experiment Cohort Design

[19]:
```
# Define Experiment Parameters

named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert

# learning_rate = 1e-3
# learning_rate = 1e-4
learning_rate = 1e-5
```

```
# learning_rate = 5e-6
# learning_rate = 5e-7
# learning_rate = 5e-8

# num_epochs = 1
# num_epochs = 3
# num_epochs = 5
num_epochs = 25
# num_epochs = 15
# num_epochs = 20

# length_max = 128
length_max = 256
# length_max = 348
# length_max = 512

# size_batch = 1
# size_batch = 4
# size_batch = 8
size_batch = 16
# size_batch = 24
# size_batch = 32
# size_batch = 64
# size_batch = 128

# regularization_weight_decay = 0
regularization_weight_decay = 0.1
# regularization_weight_decay = 0.5

y_col = "binary_complexity"
# y_col = "complexity"

x_task = "single"
# x_task = "multi"

# x_col = "sentence"
x_col = "sentence_no_contractions"
# x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"

if x_task == "single":
    df_train = train_single_df
    df_val   = trial_val_single_df
    df_test  = test_single_df
else:
    df_train = train_multi_df
```

```
    df_val   = trial_val_multi_df
    df_test  = test_multi_df

custom_config = BertConfig.from_pretrained("bert-base-cased")

custom_config.hidden_dropout_prob = 0.1
# custom_config.intermediate_size = 3072
# custom_config.intermediate_size = 6144
# custom_config.num_attention_heads = 12
# custom_config.num_hidden_layers = 12
custom_config.gradient_checkpointing = False
custom_config.attention_probs_dropout_prob = 0.1
# custom_config.max_position_embeddings = 512
# custom_config.type_vocab_size = 2
custom_config.hidden_act = "gelu"  # alts: "relu" "silu"
# custom_config.vocab_size = 28996  # must match

# model.bert.pooler.activation = nn.ReLU() # Tanh() replaced as the pooler␣
 ↪layer activation function in side-by-side with 1.1
```

```
config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]
```

[20]:
```
def train_transformer_model(
    model,
    tokenizer,
    train_dataset,
    val_dataset,
    output_dir=dir_results,
    num_epochs=num_epochs,
    batch_size=size_batch,
    lr=learning_rate,
    weight_decay=regularization_weight_decay
):
    """
    Sets up a Trainer and trains the model for 'num_epochs' using the given␣
 ↪dataset.
    Returns the trained model and the Trainer object for possible re-use or␣
 ↪analysis.
    """

    training_args = TrainingArguments(
        output_dir=output_dir,
        num_train_epochs=num_epochs,
        per_device_train_batch_size=batch_size,
        per_device_eval_batch_size=batch_size,
        evaluation_strategy="epoch",
        save_strategy="no",
```

```
        logging_strategy="epoch",
        learning_rate=lr,
        weight_decay=weight_decay,
        report_to=["none"],  # or "wandb"
        warmup_steps=100
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=val_dataset,
        tokenizer=tokenizer,  # optional
        compute_metrics=compute_metrics
    )

    trainer.train()
    return model, trainer
```

**Model Inspection ** Run **

[21]:
```
print("model checkpoints:", dir_models)
!ls /content/drive/MyDrive/266-final/models/
```

```
model checkpoints: /content/drive/MyDrive/266-final/models/
multi_bert-base-cased_binary_complexity_20250408_143322
multi_bert-base-cased_binary_complexity_20250409_175804
multi_bert-base-cased_binary_complexity_20250409_175954
multi_bert-base-cased_binary_complexity_20250409_180139
multi_bert-base-cased_binary_complexity_20250409_185057
multi_bert-base-cased_binary_complexity_20250409_185213
multi_bert-base-cased_binary_complexity_20250409_185333
multi_bert-base-cased_binary_complexity_20250409_234934
multi_bert-base-cased_binary_complexity_20250410_001637
multi_bert-base-cased_binary_complexity_20250410_003117
multi_bert-base-cased_binary_complexity_20250410_004527
multi_bert-base-cased_binary_complexity_20250410_025823
multi_bert-base-cased_binary_complexity_20250410_030623
multi_bert-base-cased_binary_complexity_20250410_031401
multi_bert-base-cased_binary_complexity_20250410_032138
multi_bert-base-cased_binary_complexity_20250410_034203
multi_bert-base-cased_binary_complexity_20250410_034823
multi_bert-base-cased_binary_complexity_20250410_035510
multi_bert-base-cased_binary_complexity_20250410_040140
single_bert-base-cased_binary_complexity_20250408_043117
```

```
single_bert-base-cased_binary_complexity_20250408_043334
single_bert-base-cased_binary_complexity_20250408_043750
single_bert-base-cased_binary_complexity_20250409_175702
single_bert-base-cased_binary_complexity_20250409_175900
single_bert-base-cased_binary_complexity_20250409_180045
single_bert-base-cased_binary_complexity_20250409_185027
single_bert-base-cased_binary_complexity_20250409_185141
single_bert-base-cased_binary_complexity_20250409_185303
single_bert-base-cased_binary_complexity_20250409_234236
single_bert-base-cased_binary_complexity_20250410_000508
single_bert-base-cased_binary_complexity_20250410_002813
single_bert-base-cased_binary_complexity_20250410_004230
single_bert-base-cased_binary_complexity_20250410_025214
single_bert-base-cased_binary_complexity_20250410_030435
single_bert-base-cased_binary_complexity_20250410_031211
single_bert-base-cased_binary_complexity_20250410_031404
single_bert-base-cased_binary_complexity_20250410_031948
single_bert-base-cased_binary_complexity_20250410_034334
single_bert-base-cased_binary_complexity_20250410_035314
single_bert-base-cased_binary_complexity_20250410_035940
```

[22]:

```python
# Load Model & Tokenizer
# model, tokenizer = get_model_and_tokenizer(named_model) # deprecated argument structure
# model, tokenizer = get_model_and_tokenizer("/content/drive/MyDrive/266-final/models/....") # proposed argument usage for checkpointed models

# for name, param in model.named_parameters():
#     print(name)

model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=custom_config
)

# model, tokenizer = get_model_and_tokenizer(
#     local_model_path="my_local_bert_path",
#     config=custom_config
# )

print("==============")
print(named_model, ":")
print("==============")
# print(model)
print("==============")
print(model.config)
```

```python
print("==============")
print("num_parameters:", model.num_parameters())
print("==============")
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
```

```
Loading from Hugging Face model: bert-base-cased
```

```
tokenizer_config.json:   0%|          | 0.00/49.0 [00:00<?, ?B/s]
```

```
vocab.txt:   0%|          | 0.00/213k [00:00<?, ?B/s]
```

```
tokenizer.json:   0%|          | 0.00/436k [00:00<?, ?B/s]
```

```
model.safetensors:   0%|          | 0.00/436M [00:00<?, ?B/s]
```

```
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-cased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

```
==============
bert-base-cased :
==============
==============
BertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}
```

```
=============
num_parameters: 108311810
=============
num_trainable_parameters: 108311810
```

Layer Configuration ** Run **

[23]:
```python
# Freeze/Unfreeze Layers & Additional Activation Function Configuration

layers_to_unfreeze = [
    # "bert.embeddings.",
    # "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]

freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)

for name, param in model.named_parameters():
    print(name, "requires_grad=", param.requires_grad)

print("\nLayers that are 'True' are trainable. 'False' are frozen.")

print("=============")
print(named_model, ":")
print("=============")
# print(model)
print("=============")
print(model.config)
print("=============")
print("num_parameters:", model.num_parameters())
print("=============")
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
```

```
bert.embeddings.word_embeddings.weight requires_grad= True
bert.embeddings.position_embeddings.weight requires_grad= True
bert.embeddings.token_type_embeddings.weight requires_grad= True
bert.embeddings.LayerNorm.weight requires_grad= True
bert.embeddings.LayerNorm.bias requires_grad= True
bert.encoder.layer.0.attention.self.query.weight requires_grad= True
bert.encoder.layer.0.attention.self.query.bias requires_grad= True
bert.encoder.layer.0.attention.self.key.weight requires_grad= True
bert.encoder.layer.0.attention.self.key.bias requires_grad= True
```

```
bert.encoder.layer.0.attention.self.value.weight requires_grad= True
bert.encoder.layer.0.attention.self.value.bias requires_grad= True
bert.encoder.layer.0.attention.output.dense.weight requires_grad= True
bert.encoder.layer.0.attention.output.dense.bias requires_grad= True
bert.encoder.layer.0.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.0.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.0.intermediate.dense.weight requires_grad= True
bert.encoder.layer.0.intermediate.dense.bias requires_grad= True
bert.encoder.layer.0.output.dense.weight requires_grad= True
bert.encoder.layer.0.output.dense.bias requires_grad= True
bert.encoder.layer.0.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.0.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.1.attention.self.query.weight requires_grad= False
bert.encoder.layer.1.attention.self.query.bias requires_grad= False
bert.encoder.layer.1.attention.self.key.weight requires_grad= False
bert.encoder.layer.1.attention.self.key.bias requires_grad= False
bert.encoder.layer.1.attention.self.value.weight requires_grad= False
bert.encoder.layer.1.attention.self.value.bias requires_grad= False
bert.encoder.layer.1.attention.output.dense.weight requires_grad= False
bert.encoder.layer.1.attention.output.dense.bias requires_grad= False
bert.encoder.layer.1.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.1.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.1.intermediate.dense.weight requires_grad= False
bert.encoder.layer.1.intermediate.dense.bias requires_grad= False
bert.encoder.layer.1.output.dense.weight requires_grad= False
bert.encoder.layer.1.output.dense.bias requires_grad= False
bert.encoder.layer.1.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.1.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.2.attention.self.query.weight requires_grad= False
bert.encoder.layer.2.attention.self.query.bias requires_grad= False
bert.encoder.layer.2.attention.self.key.weight requires_grad= False
bert.encoder.layer.2.attention.self.key.bias requires_grad= False
bert.encoder.layer.2.attention.self.value.weight requires_grad= False
bert.encoder.layer.2.attention.self.value.bias requires_grad= False
bert.encoder.layer.2.attention.output.dense.weight requires_grad= False
bert.encoder.layer.2.attention.output.dense.bias requires_grad= False
bert.encoder.layer.2.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.2.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.2.intermediate.dense.weight requires_grad= False
bert.encoder.layer.2.intermediate.dense.bias requires_grad= False
bert.encoder.layer.2.output.dense.weight requires_grad= False
bert.encoder.layer.2.output.dense.bias requires_grad= False
bert.encoder.layer.2.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.2.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.3.attention.self.query.weight requires_grad= False
bert.encoder.layer.3.attention.self.query.bias requires_grad= False
bert.encoder.layer.3.attention.self.key.weight requires_grad= False
bert.encoder.layer.3.attention.self.key.bias requires_grad= False
```

```
bert.encoder.layer.3.attention.self.value.weight requires_grad= False
bert.encoder.layer.3.attention.self.value.bias requires_grad= False
bert.encoder.layer.3.attention.output.dense.weight requires_grad= False
bert.encoder.layer.3.attention.output.dense.bias requires_grad= False
bert.encoder.layer.3.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.3.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.3.intermediate.dense.weight requires_grad= False
bert.encoder.layer.3.intermediate.dense.bias requires_grad= False
bert.encoder.layer.3.output.dense.weight requires_grad= False
bert.encoder.layer.3.output.dense.bias requires_grad= False
bert.encoder.layer.3.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.3.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.4.attention.self.query.weight requires_grad= False
bert.encoder.layer.4.attention.self.query.bias requires_grad= False
bert.encoder.layer.4.attention.self.key.weight requires_grad= False
bert.encoder.layer.4.attention.self.key.bias requires_grad= False
bert.encoder.layer.4.attention.self.value.weight requires_grad= False
bert.encoder.layer.4.attention.self.value.bias requires_grad= False
bert.encoder.layer.4.attention.output.dense.weight requires_grad= False
bert.encoder.layer.4.attention.output.dense.bias requires_grad= False
bert.encoder.layer.4.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.4.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.4.intermediate.dense.weight requires_grad= False
bert.encoder.layer.4.intermediate.dense.bias requires_grad= False
bert.encoder.layer.4.output.dense.weight requires_grad= False
bert.encoder.layer.4.output.dense.bias requires_grad= False
bert.encoder.layer.4.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.4.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.5.attention.self.query.weight requires_grad= False
bert.encoder.layer.5.attention.self.query.bias requires_grad= False
bert.encoder.layer.5.attention.self.key.weight requires_grad= False
bert.encoder.layer.5.attention.self.key.bias requires_grad= False
bert.encoder.layer.5.attention.self.value.weight requires_grad= False
bert.encoder.layer.5.attention.self.value.bias requires_grad= False
bert.encoder.layer.5.attention.output.dense.weight requires_grad= False
bert.encoder.layer.5.attention.output.dense.bias requires_grad= False
bert.encoder.layer.5.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.5.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.5.intermediate.dense.weight requires_grad= False
bert.encoder.layer.5.intermediate.dense.bias requires_grad= False
bert.encoder.layer.5.output.dense.weight requires_grad= False
bert.encoder.layer.5.output.dense.bias requires_grad= False
bert.encoder.layer.5.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.5.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.6.attention.self.query.weight requires_grad= False
bert.encoder.layer.6.attention.self.query.bias requires_grad= False
bert.encoder.layer.6.attention.self.key.weight requires_grad= False
bert.encoder.layer.6.attention.self.key.bias requires_grad= False
```

```
bert.encoder.layer.6.attention.self.value.weight requires_grad= False
bert.encoder.layer.6.attention.self.value.bias requires_grad= False
bert.encoder.layer.6.attention.output.dense.weight requires_grad= False
bert.encoder.layer.6.attention.output.dense.bias requires_grad= False
bert.encoder.layer.6.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.6.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.6.intermediate.dense.weight requires_grad= False
bert.encoder.layer.6.intermediate.dense.bias requires_grad= False
bert.encoder.layer.6.output.dense.weight requires_grad= False
bert.encoder.layer.6.output.dense.bias requires_grad= False
bert.encoder.layer.6.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.6.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.7.attention.self.query.weight requires_grad= False
bert.encoder.layer.7.attention.self.query.bias requires_grad= False
bert.encoder.layer.7.attention.self.key.weight requires_grad= False
bert.encoder.layer.7.attention.self.key.bias requires_grad= False
bert.encoder.layer.7.attention.self.value.weight requires_grad= False
bert.encoder.layer.7.attention.self.value.bias requires_grad= False
bert.encoder.layer.7.attention.output.dense.weight requires_grad= False
bert.encoder.layer.7.attention.output.dense.bias requires_grad= False
bert.encoder.layer.7.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.7.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.7.intermediate.dense.weight requires_grad= False
bert.encoder.layer.7.intermediate.dense.bias requires_grad= False
bert.encoder.layer.7.output.dense.weight requires_grad= False
bert.encoder.layer.7.output.dense.bias requires_grad= False
bert.encoder.layer.7.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.7.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.8.attention.self.query.weight requires_grad= True
bert.encoder.layer.8.attention.self.query.bias requires_grad= True
bert.encoder.layer.8.attention.self.key.weight requires_grad= True
bert.encoder.layer.8.attention.self.key.bias requires_grad= True
bert.encoder.layer.8.attention.self.value.weight requires_grad= True
bert.encoder.layer.8.attention.self.value.bias requires_grad= True
bert.encoder.layer.8.attention.output.dense.weight requires_grad= True
bert.encoder.layer.8.attention.output.dense.bias requires_grad= True
bert.encoder.layer.8.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.8.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.8.intermediate.dense.weight requires_grad= True
bert.encoder.layer.8.intermediate.dense.bias requires_grad= True
bert.encoder.layer.8.output.dense.weight requires_grad= True
bert.encoder.layer.8.output.dense.bias requires_grad= True
bert.encoder.layer.8.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.8.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.9.attention.self.query.weight requires_grad= True
bert.encoder.layer.9.attention.self.query.bias requires_grad= True
bert.encoder.layer.9.attention.self.key.weight requires_grad= True
bert.encoder.layer.9.attention.self.key.bias requires_grad= True
```

```
bert.encoder.layer.9.attention.self.value.weight requires_grad= True
bert.encoder.layer.9.attention.self.value.bias requires_grad= True
bert.encoder.layer.9.attention.output.dense.weight requires_grad= True
bert.encoder.layer.9.attention.output.dense.bias requires_grad= True
bert.encoder.layer.9.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.9.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.9.intermediate.dense.weight requires_grad= True
bert.encoder.layer.9.intermediate.dense.bias requires_grad= True
bert.encoder.layer.9.output.dense.weight requires_grad= True
bert.encoder.layer.9.output.dense.bias requires_grad= True
bert.encoder.layer.9.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.9.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.10.attention.self.query.weight requires_grad= True
bert.encoder.layer.10.attention.self.query.bias requires_grad= True
bert.encoder.layer.10.attention.self.key.weight requires_grad= True
bert.encoder.layer.10.attention.self.key.bias requires_grad= True
bert.encoder.layer.10.attention.self.value.weight requires_grad= True
bert.encoder.layer.10.attention.self.value.bias requires_grad= True
bert.encoder.layer.10.attention.output.dense.weight requires_grad= True
bert.encoder.layer.10.attention.output.dense.bias requires_grad= True
bert.encoder.layer.10.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.10.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.10.intermediate.dense.weight requires_grad= True
bert.encoder.layer.10.intermediate.dense.bias requires_grad= True
bert.encoder.layer.10.output.dense.weight requires_grad= True
bert.encoder.layer.10.output.dense.bias requires_grad= True
bert.encoder.layer.10.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.10.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.11.attention.self.query.weight requires_grad= True
bert.encoder.layer.11.attention.self.query.bias requires_grad= True
bert.encoder.layer.11.attention.self.key.weight requires_grad= True
bert.encoder.layer.11.attention.self.key.bias requires_grad= True
bert.encoder.layer.11.attention.self.value.weight requires_grad= True
bert.encoder.layer.11.attention.self.value.bias requires_grad= True
bert.encoder.layer.11.attention.output.dense.weight requires_grad= True
bert.encoder.layer.11.attention.output.dense.bias requires_grad= True
bert.encoder.layer.11.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.11.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.11.intermediate.dense.weight requires_grad= True
bert.encoder.layer.11.intermediate.dense.bias requires_grad= True
bert.encoder.layer.11.output.dense.weight requires_grad= True
bert.encoder.layer.11.output.dense.bias requires_grad= True
bert.encoder.layer.11.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.11.output.LayerNorm.bias requires_grad= True
bert.pooler.dense.weight requires_grad= True
bert.pooler.dense.bias requires_grad= True
classifier.weight requires_grad= True
classifier.bias requires_grad= True
```

```
Layers that are 'True' are trainable. 'False' are frozen.
=============
bert-base-cased :
=============
=============
BertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}

=============
num_parameters: 108311810
=============
num_trainable_parameters: 58696706
```

Dataset Preparation ** Run **

[24]:
```
# Tokenize & Prepare Datasets

train_data_hf = prepare_dataset(
    df_train,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max
```

```
)

val_data_hf = prepare_dataset(
    df_val,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max
)

test_data_hf = prepare_dataset(
    df_test,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max

)

print("Datasets prepared. Sample from train_data_hf:\n", train_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", val_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", test_data_hf[10])
```

```
Map:   0%|          | 0/7662 [00:00<?, ? examples/s]

Map:   0%|          | 0/421 [00:00<?, ? examples/s]

Map:   0%|          | 0/917 [00:00<?, ? examples/s]

Datasets prepared. Sample from train_data_hf:
 {'labels': tensor(0), 'input_ids': tensor([  101,  1252,  1106,  1103,  3824,
1104, 19892, 11220,  1324,  1119,
         1522,  3839,   117,  1272,  1103,  1555,  1104,  1103, 11563,  5609,
         1106,  1172,   132,  1152,  2446,  1122,  1113,  1147,  3221,   119,
          102,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
```

```
                0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
                0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
                0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
                0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
                0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
                0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
                0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
                0,     0,     0,     0,     0,     0]), 'attention_mask': tensor([1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])}
```

0.2.3 2 2e-5 128 128 0.5 0.1 0.1 gelu classifier,layer 10,layer 11,layer 8,layer 9,pooler

[25]:

```
# Define Experiment Parameters
named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert
############
regularization_weight_decay = 0.5
learning_rate = 2e-5
size_batch = 128
length_max = 128
num_epochs = 2
##########################################
# x_col = "sentence"
# x_col = "sentence_no_contractions"
x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"
############
y_col = "binary_complexity"
# y_col = "complexity"
############
x_task = "single"
# x_task = "multi"
if x_task == "single":
```

```
    df_train = train_single_df
    df_val   = trial_val_single_df
    df_test  = test_single_df
else:
    df_train = train_multi_df
    df_val   = trial_val_multi_df
    df_test  = test_multi_df
##########################################
# Tokenize & Prepare Datasets
train_data_hf = prepare_dataset(
    df_train,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
val_data_hf = prepare_dataset(
    df_val,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
test_data_hf = prepare_dataset(
    df_test,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
print("Datasets prepared. Sample from train_data_hf:\n", train_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", val_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", test_data_hf[10])
##########################################
custom_config = BertConfig.from_pretrained("bert-base-cased")
custom_config.hidden_act = "gelu"  # alts: "relu" "silu"
custom_config.attention_probs_dropout_prob = 0.1
custom_config.hidden_dropout_prob = 0.1
custom_config.gradient_checkpointing = False
##########################################
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=custom_config)
############
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
print("==============")
```

```
print(named_model, ":")
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
 ↪num_parameters(only_trainable=True))
print("=============")
print("model lineage:", MODEL_LINEAGE)
print("=============")
########################################
layers_to_unfreeze = [
    # "bert.embeddings.",
    # "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=============")
########################################
print("Experiment configuration used with this experiment:")
print("model used:", named_model)
print("learning rate used:", learning_rate)
print("number of epochs:", num_epochs)
print("maximum sequence length:", length_max)
print("batch size used:", size_batch)
print("regularization value:", regularization_weight_decay)
print("outcome variable:", y_col)
print("task:", x_task)
print("input column:", x_col)
```

```
Map:   0%|          | 0/7662 [00:00<?, ? examples/s]

Map:   0%|          | 0/421 [00:00<?, ? examples/s]

Map:   0%|          | 0/917 [00:00<?, ? examples/s]

Datasets prepared. Sample from train_data_hf:
 {'labels': tensor(0), 'input_ids': tensor([  101,   164,   112, 21362, 11414,
4538,   112,   117,   112,  5844,
         2101,   112,   117,   112, 18581,  1942,   112,   117,   112, 24819,
        27370,   112,   117,   112,  5844,  2101,   112,   117,   112, 11629,
```

```
        17195,  2249,   112,   117,   112, 11629, 11414,   112,   117,   112,
          159,  9637,  2064,   112,   117,   112, 24819, 27370,   112,   117,
          112,   153, 27370, 16647,   112,   117,   112,  9314, 11414,  4538,
          112,   117,   112, 18581,  1942,   112,   117,   112, 24819, 27370,
          112,   117,   112,  5844,  2101,   112,   117,   112, 18581,  1942,
          112,   117,   112, 24819, 27370,   112,   117,   112,   159,  9637,
         2064,   112,   117,   112,  5844,  2101,   112,   117,   112, 11629,
        11414,   112,   117,   112,   153, 27370, 16647,   112,   117,   112,
        11629, 11414,   112,   117,   112,   159,  9637,  2064,   112,   117,
          112, 11629, 11414,   112,   117,   112,  5844,   102]),
'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1])}

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-cased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Loading from Hugging Face model: bert-base-cased
=============
bert-base-cased :
=============
num_parameters: 108311810
num_trainable_parameters at load: 108311810
=============
model lineage: {'type': 'huggingface_hub', 'path': 'bert-base-cased',
'timestamp': '2025-04-10 17:36:41'}
=============
BertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
```

```
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}

=============
num_parameters: 108311810
num_trainable_parameters: 28943618
=============
Experiment configuration used with this experiment:
model used: bert-base-cased
learning rate used: 2e-05
number of epochs: 2
maximum sequence length: 128
batch size used: 128
regularization value: 0.5
outcome variable: binary_complexity
task: single
input column: pos_sequence
```

[26]:
```python
# Train & Evaluate
trained_model, trainer_obj = train_transformer_model(
    model = model,
    tokenizer = tokenizer,
    train_dataset = train_data_hf,
    val_dataset = val_data_hf,
    output_dir = dir_results,
    num_epochs = num_epochs,
    batch_size = size_batch,
    lr = learning_rate,
    weight_decay = regularization_weight_decay)
metrics = trainer_obj.evaluate()
print("Validation metrics:", metrics)
test_metrics = trainer_obj.evaluate(test_data_hf) if test_data_hf else None
print("Test metrics:", test_metrics)
```

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1611:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of  Transformers. Use `eval_strategy` instead
  warnings.warn(
WARNING:root:torch_xla.core.xla_model.xrt_world_size() will be removed in
```

```
release 2.7. is deprecated. Use torch_xla.runtime.world_size instead.
WARNING:root:torch_xla.core.xla_model.xla_model.get_ordinal() will be removed in
release 2.7. is deprecated. Use torch_xla.runtime.global_ordinal instead.
<ipython-input-20-c2ee9f934517>:31: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
  trainer = Trainer(
```

```
<IPython.core.display.HTML object>
```

```
Downloading builder script:   0%|          | 0.00/4.20k [00:00<?, ?B/s]
```

```
Downloading builder script:   0%|          | 0.00/7.56k [00:00<?, ?B/s]
```

```
Downloading builder script:   0%|          | 0.00/7.38k [00:00<?, ?B/s]
```

```
Downloading builder script:   0%|          | 0.00/6.79k [00:00<?, ?B/s]
```

```
<IPython.core.display.HTML object>
```

```
Validation metrics: {'eval_loss': 0.6967368125915527, 'eval_accuracy':
0.5415676959619953, 'eval_precision': 0.49473684210526314, 'eval_recall':
0.24479166666666666, 'eval_f1': 0.32752613240418116, 'eval_runtime': 1.2139,
'eval_samples_per_second': 421.773, 'eval_steps_per_second': 3.295, 'epoch':
2.0}
Test metrics: {'eval_loss': 0.6922365427017212, 'eval_accuracy':
0.5398037077426391, 'eval_precision': 0.5381526104417671, 'eval_recall':
0.30385487528344673, 'eval_f1': 0.3884057971014493, 'eval_runtime': 6.8772,
'eval_samples_per_second': 148.899, 'eval_steps_per_second': 1.163, 'epoch':
2.0}
```

[27]:
```
# save model checkpoint
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,␣
  ↪f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
```

```
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")
```

```
Model checkpoint saved to: /content/drive/MyDrive/266-final/models/single_bert-
base-cased_binary_complexity_20250410_173757

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:
/content/drive/MyDrive/266-final/results/experiment_runs.txt
```

0.2.4 2 2e-5 128 128 0.5 0.1 0.1 gelu classifier,layer 10,layer 11,layer 8,layer 9,pooler

[28]:
```
# Define Experiment Parameters
named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert
#############
regularization_weight_decay = 0.5
learning_rate = 0.000005
size_batch = 128
length_max = 128
num_epochs = 2
#########################################
# x_col = "sentence"
# x_col = "sentence_no_contractions"
x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"
#############
y_col = "binary_complexity"
# y_col = "complexity"
#############
x_task = "single"
# x_task = "multi"
if x_task == "single":
    df_train = train_single_df
    df_val   = trial_val_single_df
    df_test  = test_single_df
```

```
else:
    df_train = train_multi_df
    df_val   = trial_val_multi_df
    df_test  = test_multi_df
########################################
# Tokenize & Prepare Datasets
train_data_hf = prepare_dataset(
    df_train,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
val_data_hf = prepare_dataset(
    df_val,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
test_data_hf = prepare_dataset(
    df_test,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
print("Datasets prepared. Sample from train_data_hf:\n", train_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", val_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", test_data_hf[10])
########################################
custom_config = BertConfig.from_pretrained("bert-base-cased")
custom_config.hidden_act = "gelu"  # alts: "relu" "silu"
custom_config.attention_probs_dropout_prob = 0.1
custom_config.hidden_dropout_prob = 0.1
custom_config.gradient_checkpointing = False
########################################
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=custom_config)
############
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
print("==============")
print(named_model, ":")
print("==============")
print("num_parameters:", model.num_parameters())
```

```
print("num_trainable_parameters at load:", model.
 ↪num_parameters(only_trainable=True))
print("=============")
print("model lineage:", MODEL_LINEAGE)
print("=============")
########################################
layers_to_unfreeze = [
    # "bert.embeddings.",
    # "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=============")
########################################
print("Experiment configuration used with this experiment:")
print("model used:", named_model)
print("learning rate used:", learning_rate)
print("number of epochs:", num_epochs)
print("maximum sequence length:", length_max)
print("batch size used:", size_batch)
print("regularization value:", regularization_weight_decay)
print("outcome variable:", y_col)
print("task:", x_task)
print("input column:", x_col)
```

```
Map:   0%|          | 0/7662 [00:00<?, ? examples/s]
Map:   0%|          | 0/421 [00:00<?, ? examples/s]
Map:   0%|          | 0/917 [00:00<?, ? examples/s]
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-cased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
Datasets prepared. Sample from train_data_hf:
 {'labels': tensor(0), 'input_ids': tensor([  101,   164,   112, 21362, 11414,
4538,   112,   117,   112,  5844,
```

```
         2101,   112,   117,   112, 18581,  1942,   112,   117,   112, 24819,
        27370,   112,   117,   112,  5844,  2101,   112,   117,   112, 11629,
        17195,  2249,   112,   117,   112, 11629, 11414,   112,   117,   112,
          159,  9637,  2064,   112,   117,   112, 24819, 27370,   112,   117,
          112,   153, 27370, 16647,   112,   117,   112,  9314, 11414,  4538,
          112,   117,   112, 18581,  1942,   112,   117,   112, 24819, 27370,
          112,   117,   112,  5844,  2101,   112,   117,   112, 18581,  1942,
          112,   117,   112, 24819, 27370,   112,   117,   112,   159,  9637,
         2064,   112,   117,   112,  5844,  2101,   112,   117,   112, 11629,
        11414,   112,   117,   112,   153, 27370, 16647,   112,   117,   112,
        11629, 11414,   112,   117,   112,   159,  9637,  2064,   112,   117,
          112, 11629, 11414,   112,   117,   112,  5844,   102]),
'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1])}
Loading from Hugging Face model: bert-base-cased
=============
bert-base-cased :
=============
num_parameters: 108311810
num_trainable_parameters at load: 108311810
=============
model lineage: {'type': 'huggingface_hub', 'path': 'bert-base-cased',
'timestamp': '2025-04-10 17:38:51'}
=============
BertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
```

```
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}

=============
num_parameters: 108311810
num_trainable_parameters: 28943618
=============
Experiment configuration used with this experiment:
model used: bert-base-cased
learning rate used: 5e-06
number of epochs: 2
maximum sequence length: 128
batch size used: 128
regularization value: 0.5
outcome variable: binary_complexity
task: single
input column: pos_sequence
```

[29]:
```
# #QA

# def validate_dataframe(df, df_name):
#     """
#     Performs basic functional tests on a pandas DataFrame
#     to ensure it matches expected structure and content.
#     """
#     print(f"\n[VALIDATION] Checking {df_name}...")

#     # 1) Check shape
#     print(f" - Shape: {df.shape}")

#     # 2) Check columns
#     print(f" - Columns: {list(df.columns)}")

#     # 3) Check label distribution (assuming 'binary_complexity' is the label)
#     if "binary_complexity" in df.columns:
#         label_counts = df["binary_complexity"].value_counts(dropna=False)
#         print(f" - Label distribution:\n{label_counts}")
#     else:
#         print(" - WARNING: 'binary_complexity' column not found!")

#     # 4) Peek at top few rows
#     print(" - Sample rows:\n", df.head(3))
```

```
# # Example usage for multi data:
# validate_dataframe(train_multi_df, "train_multi_df")
# validate_dataframe(trial_val_multi_df, "trial_val_multi_df")
# validate_dataframe(test_multi_df, "test_multi_df")
```

[30]:
```
def check_dataframe_invariants(df, df_name, expected_shape, expected_columns):
    """
    Ensures that df has the exact shape and columns expected.
    Raises AssertionError if not.
    """
    print(f"\n[CHECK] {df_name}")

    actual_shape = df.shape
    actual_columns = set(df.columns)

    # 1) Check shape
    assert actual_shape == expected_shape, (
        f"[ERROR] {df_name} shape mismatch. "
        f"Expected {expected_shape}, got {actual_shape}."
    )

    # 2) Check columns
    assert actual_columns == set(expected_columns), (
        f"[ERROR] {df_name} columns mismatch. "
        f"Expected {set(expected_columns)}, got {actual_columns}."
    )

    print(" - PASS: shape and columns match expectations")

# Suppose the actual columns are exactly:
my_expected_cols = [
    "id", "sentence", "sentence_no_contractions", "token",
    "contraction_expanded", "pos_sequence", "morph_sequence",
    "dep_sequence", "morph_complexity", "complexity",
    "binary_complexity", "corpus"
]

check_dataframe_invariants(
    train_multi_df,
    "train_multi_df",
    expected_shape=(1517, 12),  # example only
    expected_columns=my_expected_cols
)
```

```
[CHECK] train_multi_df
```

- PASS: shape and columns match expectations

[31]:
```
# Train & Evaluate
trained_model, trainer_obj = train_transformer_model(
    model = model,
    tokenizer = tokenizer,
    train_dataset = train_data_hf,
    val_dataset = val_data_hf,
    output_dir = dir_results,
    num_epochs = num_epochs,
    batch_size = size_batch,
    lr = learning_rate,
    weight_decay = regularization_weight_decay)
metrics = trainer_obj.evaluate()
print("Validation metrics:", metrics)
test_metrics = trainer_obj.evaluate(test_data_hf) if test_data_hf else None
print("Test metrics:", test_metrics)
```

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1611:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of  Transformers. Use `eval_strategy` instead
  warnings.warn(
<ipython-input-20-c2ee9f934517>:31: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
  trainer = Trainer(
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Validation metrics: {'eval_loss': 0.6993261575698853, 'eval_accuracy':
0.5296912114014252, 'eval_precision': 0.47619047619047616, 'eval_recall':
0.3125, 'eval_f1': 0.37735849056603776, 'eval_runtime': 1.2298,
'eval_samples_per_second': 416.326, 'eval_steps_per_second': 3.253, 'epoch':
2.0}
Test metrics: {'eval_loss': 0.6964348554611206, 'eval_accuracy':
0.495092693565976, 'eval_precision': 0.4566929133858268, 'eval_recall':
0.26303854875283444, 'eval_f1': 0.3338129496402878, 'eval_runtime': 1.5585,
'eval_samples_per_second': 657.038, 'eval_steps_per_second': 5.133, 'epoch':
2.0}
```

[32]:
```
# save model checkpoint
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,␣
  ↪f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
```

```
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")
```

```
Model checkpoint saved to: /content/drive/MyDrive/266-final/models/single_bert-
base-cased_binary_complexity_20250410_173911

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:
/content/drive/MyDrive/266-final/results/experiment_runs.txt
```

0.2.5 5 2e-5 128 128 0.5 0.1 0.1 gelu classifier,layer 10,layer 11,layer 8,layer 9,pooler

[33]:
```
# Define Experiment Parameters
named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert
#############
regularization_weight_decay = 0.5
learning_rate = 0.000005
size_batch = 128
length_max = 128
num_epochs = 5
##########################################
# x_col = "sentence"
# x_col = "sentence_no_contractions"
```

```
x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"
#############
y_col = "binary_complexity"
# y_col = "complexity"
#############
x_task = "single"
# x_task = "multi"
if x_task == "single":
    df_train = train_single_df
    df_val   = trial_val_single_df
    df_test  = test_single_df
else:
    df_train = train_multi_df
    df_val   = trial_val_multi_df
    df_test  = test_multi_df
##########################################
# Tokenize & Prepare Datasets
train_data_hf = prepare_dataset(
    df_train,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
val_data_hf = prepare_dataset(
    df_val,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
test_data_hf = prepare_dataset(
    df_test,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
print("Datasets prepared. Sample from train_data_hf:\n", train_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", val_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", test_data_hf[10])
##########################################
custom_config = BertConfig.from_pretrained("bert-base-cased")
custom_config.hidden_act = "gelu"  # alts: "relu" "silu"
custom_config.attention_probs_dropout_prob = 0.1
custom_config.hidden_dropout_prob = 0.1
custom_config.gradient_checkpointing = False
##########################################
```

```
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=custom_config)
############
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
print("=============")
print(named_model, ":")
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
 ↪num_parameters(only_trainable=True))
print("=============")
print("model lineage:", MODEL_LINEAGE)
print("=============")
########################################
layers_to_unfreeze = [
    # "bert.embeddings.",
    # "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=============")
########################################
print("Experiment configuration used with this experiment:")
print("model used:", named_model)
print("learning rate used:", learning_rate)
print("number of epochs:", num_epochs)
print("maximum sequence length:", length_max)
print("batch size used:", size_batch)
print("regularization value:", regularization_weight_decay)
print("outcome variable:", y_col)
print("task:", x_task)
print("input column:", x_col)
```

```
Map:   0%|          | 0/7662 [00:00<?, ? examples/s]

Map:   0%|          | 0/421 [00:00<?, ? examples/s]

Map:   0%|          | 0/917 [00:00<?, ? examples/s]

Datasets prepared. Sample from train_data_hf:
 {'labels': tensor(0), 'input_ids': tensor([  101,   164,   112, 21362, 11414,
4538,   112,   117,   112,  5844,
         2101,   112,   117,   112, 18581,  1942,   112,   117,   112, 24819,
        27370,   112,   117,   112,  5844,  2101,   112,   117,   112, 11629,
        17195,  2249,   112,   117,   112, 11629, 11414,   112,   117,   112,
          159,  9637,  2064,   112,   117,   112, 24819, 27370,   112,   117,
          112,   153, 27370, 16647,   112,   117,   112,  9314, 11414,  4538,
          112,   117,   112, 18581,  1942,   112,   117,   112, 24819, 27370,
          112,   117,   112,  5844,  2101,   112,   117,   112, 18581,  1942,
          112,   117,   112, 24819, 27370,   112,   117,   112,   159,  9637,
         2064,   112,   117,   112,  5844,  2101,   112,   117,   112, 11629,
        11414,   112,   117,   112,   153, 27370, 16647,   112,   117,   112,
        11629, 11414,   112,   117,   112,   159,  9637,  2064,   112,   117,
          112, 11629, 11414,   112,   117,   112,  5844,   102]),
'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1])}
Loading from Hugging Face model: bert-base-cased

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-cased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

=============
bert-base-cased :
=============
num_parameters: 108311810
num_trainable_parameters at load: 108311810
=============
model lineage: {'type': 'huggingface_hub', 'path': 'bert-base-cased',
'timestamp': '2025-04-10 17:39:41'}
=============
BertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "BertForMaskedLM"
  ],
```

```
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}

=============
num_parameters: 108311810
num_trainable_parameters: 28943618
=============
Experiment configuration used with this experiment:
model used: bert-base-cased
learning rate used: 5e-06
number of epochs: 5
maximum sequence length: 128
batch size used: 128
regularization value: 0.5
outcome variable: binary_complexity
task: single
input column: pos_sequence
```

[34]:
```
# Train & Evaluate
trained_model, trainer_obj = train_transformer_model(
    model = model,
    tokenizer = tokenizer,
    train_dataset = train_data_hf,
    val_dataset = val_data_hf,
    output_dir = dir_results,
    num_epochs = num_epochs,
    batch_size = size_batch,
    lr = learning_rate,
```

```
    weight_decay = regularization_weight_decay)
metrics = trainer_obj.evaluate()
print("Validation metrics:", metrics)
test_metrics = trainer_obj.evaluate(test_data_hf) if test_data_hf else None
print("Test metrics:", test_metrics)
```

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1611:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of  Transformers. Use `eval_strategy` instead
  warnings.warn(
<ipython-input-20-c2ee9f934517>:31: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
  trainer = Trainer(
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Validation metrics: {'eval_loss': 0.696632981300354, 'eval_accuracy':
0.5083135391923991, 'eval_precision': 0.4556213017751479, 'eval_recall':
0.4010416666666667, 'eval_f1': 0.4265927977839335, 'eval_runtime': 1.325,
'eval_samples_per_second': 386.417, 'eval_steps_per_second': 3.019, 'epoch':
5.0}
Test metrics: {'eval_loss': 0.6927960515022278, 'eval_accuracy':
0.520174482006543, 'eval_precision': 0.5014577259475219, 'eval_recall':
0.3900226757369615, 'eval_f1': 0.4387755102040816, 'eval_runtime': 1.4474,
'eval_samples_per_second': 707.5, 'eval_steps_per_second': 5.527, 'epoch': 5.0}
```

[35]:
```
# save model checkpoint
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,␣
  ↪f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
```

```
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")
```

```
Model checkpoint saved to: /content/drive/MyDrive/266-final/models/single_bert-
base-cased_binary_complexity_20250410_174027

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:
/content/drive/MyDrive/266-final/results/experiment_runs.txt
```

0.2.6 3.1.6 regularization_weight_decay = 0.01 learning_rate = 5e-6 size_batch = 8 length_max = 128 num_epochs = 3

[41]:
```
# Define Experiment Parameters
named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert
#############
regularization_weight_decay = 0.5
learning_rate = 1e-5
size_batch = 8
length_max = 128
num_epochs = 3
#########################################
# x_col = "sentence"
x_col = "sentence_no_contractions"
# x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"
#############
y_col = "binary_complexity"
# y_col = "complexity"
#############
x_task = "single"
# x_task = "multi"
if x_task == "single":
    df_train = train_single_df
    df_val   = trial_val_single_df
```

```
    df_test  = test_single_df
else:
    df_train = train_multi_df
    df_val   = trial_val_multi_df
    df_test  = test_multi_df
########################################
# Tokenize & Prepare Datasets
train_data_hf = prepare_dataset(
    df_train,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
val_data_hf = prepare_dataset(
    df_val,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
test_data_hf = prepare_dataset(
    df_test,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
print("Datasets prepared. Sample from train_data_hf:\n", train_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", val_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", test_data_hf[10])
########################################
custom_config = BertConfig.from_pretrained("bert-base-cased")
custom_config.hidden_act = "gelu"  # alts: "relu" "silu"
custom_config.attention_probs_dropout_prob = 0.1
custom_config.hidden_dropout_prob = 0.1
custom_config.gradient_checkpointing = False
########################################
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=custom_config)
############
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
print("=============")
print(named_model, ":")
print("=============")
```

```
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
 ↪num_parameters(only_trainable=True))
print("==============")
print("model lineage:", MODEL_LINEAGE)
print("==============")
########################################
layers_to_unfreeze = [
    # "bert.embeddings.",
    # "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    # "bert.encoder.layer.8.",
    # "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("==============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("==============")
########################################
print("Experiment configuration used with this experiment:")
print("model used:", named_model)
print("learning rate used:", learning_rate)
print("number of epochs:", num_epochs)
print("maximum sequence length:", length_max)
print("batch size used:", size_batch)
print("regularization value:", regularization_weight_decay)
print("outcome variable:", y_col)
print("task:", x_task)
print("input column:", x_col)
```

```
Map:   0%|          | 0/7662 [00:00<?, ? examples/s]
```

```
Map:   0%|          | 0/421 [00:00<?, ? examples/s]
```

```
Map:   0%|          | 0/917 [00:00<?, ? examples/s]
```

```
Datasets prepared. Sample from train_data_hf:
 {'labels': tensor(0), 'input_ids': tensor([  101,  1252,  1106,  1103,  3824,
1104, 19892, 11220,  1324,  1119,
         1522,  3839,   117,  1272,  1103,  1555,  1104,  1103, 11563,  5609,
         1106,  1172,   132,  1152,  2446,  1122,  1113,  1147,  3221,   119,
          102,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
```

```
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0]),
'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0])}
Loading from Hugging Face model: bert-base-cased

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-cased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

=============
bert-base-cased :
=============
num_parameters: 108311810
num_trainable_parameters at load: 108311810
=============
model lineage: {'type': 'huggingface_hub', 'path': 'bert-base-cased',
'timestamp': '2025-04-10 17:50:01'}
=============
BertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
```

```
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}

=============
num_parameters: 108311810
num_trainable_parameters: 28943618
=============
Experiment configuration used with this experiment:
model used: bert-base-cased
learning rate used: 5e-06
number of epochs: 3
maximum sequence length: 128
batch size used: 8
regularization value: 0.01
outcome variable: binary_complexity
task: single
input column: sentence_no_contractions
```

[42]:
```
# Train & Evaluate
trained_model, trainer_obj = train_transformer_model(
    model = model,
    tokenizer = tokenizer,
    train_dataset = train_data_hf,
    val_dataset = val_data_hf,
    output_dir = dir_results,
    num_epochs = num_epochs,
    batch_size = size_batch,
    lr = learning_rate,
    weight_decay = regularization_weight_decay)
metrics = trainer_obj.evaluate()
print("Validation metrics:", metrics)
test_metrics = trainer_obj.evaluate(test_data_hf) if test_data_hf else None
print("Test metrics:", test_metrics)
```

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1611:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of  Transformers. Use `eval_strategy` instead
  warnings.warn(
<ipython-input-20-c2ee9f934517>:31: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
```

```
  trainer = Trainer(
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Validation metrics: {'eval_loss': 0.6739699244499207, 'eval_accuracy':
0.6033254156769596, 'eval_precision': 0.5714285714285714, 'eval_recall':
0.5208333333333334, 'eval_f1': 0.5449591280653951, 'eval_runtime': 2.4822,
'eval_samples_per_second': 170.816, 'eval_steps_per_second': 21.352, 'epoch':
3.0}
Test metrics: {'eval_loss': 0.6782134175300598, 'eval_accuracy':
0.5681570338058888, 'eval_precision': 0.5609756097560976, 'eval_recall':
0.46938775510204084, 'eval_f1': 0.5111111111111111, 'eval_runtime': 4.2543,
'eval_samples_per_second': 216.254, 'eval_steps_per_second': 27.032, 'epoch':
3.0}
```

[ ]:
```
# save model checkpoint
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,␣
  ↪f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")
```

```
Model checkpoint saved to: /content/drive/MyDrive/266-final/models/single_bert-
base-cased_binary_complexity_20250410_175501
```

```
<IPython.core.display.HTML object>
```

0.2.7 3.1.6 regularization_weight_decay = 0.01 learning_rate = 5e-6 size_batch = 8 length_max = 128 num_epochs = 5

[39]:
```python
# Define Experiment Parameters
named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert
#############
regularization_weight_decay = 0.01
learning_rate = 5e-6
size_batch = 8
length_max = 128
num_epochs = 5
##########################################
# x_col = "sentence"
x_col = "sentence_no_contractions"
# x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"
#############
y_col = "binary_complexity"
# y_col = "complexity"
#############
x_task = "single"
# x_task = "multi"
if x_task == "single":
    df_train = train_single_df
    df_val   = trial_val_single_df
    df_test  = test_single_df
else:
    df_train = train_multi_df
    df_val   = trial_val_multi_df
    df_test  = test_multi_df
##########################################
# Tokenize & Prepare Datasets
train_data_hf = prepare_dataset(
    df_train,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
val_data_hf = prepare_dataset(
    df_val,
```

```
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
test_data_hf = prepare_dataset(
    df_test,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
print("Datasets prepared. Sample from train_data_hf:\n", train_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", val_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", test_data_hf[10])
########################################
custom_config = BertConfig.from_pretrained("bert-base-cased")
custom_config.hidden_act = "gelu"  # alts: "relu" "silu"
custom_config.attention_probs_dropout_prob = 0.1
custom_config.hidden_dropout_prob = 0.1
custom_config.gradient_checkpointing = False
########################################
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=custom_config)
############
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
print("=============")
print(named_model, ":")
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
 ↪num_parameters(only_trainable=True))
print("=============")
print("model lineage:", MODEL_LINEAGE)
print("=============")
########################################
layers_to_unfreeze = [
    # "bert.embeddings.",
    # "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
```

```
    "bert.pooler.",
    "classifier.",
]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=============")
##########################################
print("Experiment configuration used with this experiment:")
print("model used:", named_model)
print("learning rate used:", learning_rate)
print("number of epochs:", num_epochs)
print("maximum sequence length:", length_max)
print("batch size used:", size_batch)
print("regularization value:", regularization_weight_decay)
print("outcome variable:", y_col)
print("task:", x_task)
print("input column:", x_col)
```

```
Map:   0%|          | 0/7662 [00:00<?, ? examples/s]

Map:   0%|          | 0/421 [00:00<?, ? examples/s]

Map:   0%|          | 0/917 [00:00<?, ? examples/s]

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-cased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Datasets prepared. Sample from train_data_hf:
 {'labels': tensor(0), 'input_ids': tensor([  101,  1252,  1106,  1103,  3824,
1104, 19892, 11220,  1324,  1119,
         1522,  3839,   117,  1272,  1103,  1555,  1104,  1103, 11563,  5609,
         1106,  1172,   132,  1152,  2446,  1122,  1113,  1147,  3221,   119,
          102,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0]),
'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1,
```

```
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0])}
Loading from Hugging Face model: bert-base-cased
=============
bert-base-cased :
=============
num_parameters: 108311810
num_trainable_parameters at load: 108311810
=============
model lineage: {'type': 'huggingface_hub', 'path': 'bert-base-cased',
'timestamp': '2025-04-10 17:45:06'}
=============
BertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}

=============
num_parameters: 108311810
num_trainable_parameters: 28943618
=============
Experiment configuration used with this experiment:
model used: bert-base-cased
```

```
learning rate used: 5e-06
number of epochs: 5
maximum sequence length: 128
batch size used: 8
regularization value: 0.01
outcome variable: binary_complexity
task: single
input column: sentence_no_contractions
```

[40]:
```
# Train & Evaluate
trained_model, trainer_obj = train_transformer_model(
    model = model,
    tokenizer = tokenizer,
    train_dataset = train_data_hf,
    val_dataset = val_data_hf,
    output_dir = dir_results,
    num_epochs = num_epochs,
    batch_size = size_batch,
    lr = learning_rate,
    weight_decay = regularization_weight_decay)
metrics = trainer_obj.evaluate()
print("Validation metrics:", metrics)
test_metrics = trainer_obj.evaluate(test_data_hf) if test_data_hf else None
print("Test metrics:", test_metrics)
```

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1611:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of  Transformers. Use `eval_strategy` instead
  warnings.warn(
<ipython-input-20-c2ee9f934517>:31: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
  trainer = Trainer(
```

```
<IPython.core.display.HTML object>
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-40-cf30694e0124> in <cell line: 0>()
      1 # Train & Evaluate
----> 2 trained_model, trainer_obj = train_transformer_model(

      3     model = model,
      4     tokenizer = tokenizer,
      5     train_dataset = train_data_hf,

<ipython-input-20-c2ee9f934517> in train_transformer_model(model, tokenizer,␣
 ↪train_dataset, val_dataset, output_dir, num_epochs, batch_size, lr,␣
 ↪weight_decay)
```

```
     38     )
     39
---> 40     trainer.train()
     41     return model, trainer

/usr/local/lib/python3.11/dist-packages/transformers/trainer.py in train(self,␣
 ↪resume_from_checkpoint, trial, ignore_keys_for_eval, **kwargs)
   2243                 hf_hub_utils.enable_progress_bars()
   2244         else:
-> 2245             return inner_training_loop(
   2246                 args=args,
   2247                 resume_from_checkpoint=resume_from_checkpoint,

/usr/local/lib/python3.11/dist-packages/transformers/trainer.py in␣
 ↪_inner_training_loop(self, batch_size, args, resume_from_checkpoint, trial,␣
 ↪ignore_keys_for_eval)
   2587                                 )
   2588                             else:
-> 2589                                 _grad_norm = self.accelerator.
 ↪clip_grad_norm_(
   2590                                     model.parameters(),
   2591                                     args.max_grad_norm,

/usr/local/lib/python3.11/dist-packages/accelerate/accelerator.py in␣
 ↪clip_grad_norm_(self, parameters, max_norm, norm_type)
   2508                         return model.clip_grad_norm_(max_norm, norm_typ )
   2509         self.unscale_gradients()
-> 2510         return torch.nn.utils.clip_grad_norm_(parameters, max_norm,␣
 ↪norm_type=norm_type)
   2511
   2512     def clip_grad_value_(self, parameters, clip_value):

/usr/local/lib/python3.11/dist-packages/torch_xla/_patched_functions.py in␣
 ↪clip_grad_norm_(parameters, max_norm, norm_type, error_if_nonfinite, foreach)
     54                             torch.tensor(1., dtype=dtype, device=device)
     55   for p in parameters:
---> 56     p.grad.detach().mul_(clip_value)
     57   return total_norm
     58

/usr/local/lib/python3.11/dist-packages/torch/_meta_registrations.py in␣
 ↪meta_binop_inplace(self, other)
   3661 def meta_binop_inplace(self, other):
   3662     if isinstance(other, torch.Tensor):
-> 3663         check_inplace_broadcast(self.shape, other.shape)
   3664     return self
   3665
```

```
/usr/local/lib/python3.11/dist-packages/torch/_meta_registrations.py in␣
 ↪check_inplace_broadcast(self_shape, *args_shape)
     88
     89 def check_inplace_broadcast(self_shape, *args_shape):
---> 90     broadcasted_shape = tuple(_broadcast_shapes(self_shape, *args_shape ))
     91     torch._check(
     92         broadcasted_shape == self_shape,

/usr/local/lib/python3.11/dist-packages/torch/_refs/__init__.py in␣
 ↪_broadcast_shapes(*_shapes)
    393
    394
--> 395 def _broadcast_shapes(*_shapes):
    396     from torch.fx.experimental.symbolic_shapes import␣
 ↪guard_size_oblivious
    397

KeyboardInterrupt:
```

[ ]:
```
# save model checkpoint
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,␣
 ↪f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
```

```
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")
```

0.2.8 3.1.0.2 from pretrained bert-base-cased Y: multi task 2 & X: sentence —

[ ]:
```
# Define Experiment Parameters
named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert
#############
regularization_weight_decay = 0.1
learning_rate = 1e-5
size_batch = 16
length_max = 256
num_epochs = 25
#########################################
x_col = "sentence"
# x_col = "sentence_no_contractions"
# x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"
#############
y_col = "binary_complexity"
# y_col = "complexity"
#############
# x_task = "single"
x_task = "multi"
if x_task == "single":
    df_train = train_single_df
    df_val   = trial_val_single_df
    df_test  = test_single_df
else:
    df_train = train_multi_df
    df_val   = trial_val_multi_df
    df_test  = test_multi_df
#########################################
# Tokenize & Prepare Datasets
train_data_hf = prepare_dataset(
    df_train,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
val_data_hf = prepare_dataset(
```

```
    df_val,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
test_data_hf = prepare_dataset(
    df_test,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
print("Datasets prepared. Sample from train_data_hf:\n", train_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", val_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", test_data_hf[10])
########################################
custom_config = BertConfig.from_pretrained("bert-base-cased")
custom_config.hidden_act = "gelu"  # alts: "relu" "silu"
custom_config.attention_probs_dropout_prob = 0.1
custom_config.hidden_dropout_prob = 0.1
custom_config.gradient_checkpointing = False
########################################
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=custom_config)
############
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
print("=============")
print(named_model, ":")
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
 ↪num_parameters(only_trainable=True))
print("=============")
print("model lineage:", MODEL_LINEAGE)
print("=============")
########################################
layers_to_unfreeze = [
    "bert.embeddings.",
    "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
```

```
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=============")
##########################################
print("Experiment configuration used with this experiment:")
print("model used:", named_model)
print("learning rate used:", learning_rate)
print("number of epochs:", num_epochs)
print("maximum sequence length:", length_max)
print("batch size used:", size_batch)
print("regularization value:", regularization_weight_decay)
print("outcome variable:", y_col)
print("task:", x_task)
print("input column:", x_col)
```

[ ]:
```
# Train & Evaluate
trained_model, trainer_obj = train_transformer_model(
    model = model,
    tokenizer = tokenizer,
    train_dataset = train_data_hf,
    val_dataset = val_data_hf,
    output_dir = dir_results,
    num_epochs = num_epochs,
    batch_size = size_batch,
    lr = learning_rate,
    weight_decay = regularization_weight_decay)
metrics = trainer_obj.evaluate()
print("Validation metrics:", metrics)
test_metrics = trainer_obj.evaluate(test_data_hf) if test_data_hf else None
print("Test metrics:", test_metrics)
```

[ ]:
```
# save model checkpoint
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,␣
 ↪f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
```

```
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")
```

0.2.9 3.1.7 from pretrained roberta-base Y: single task 1 & X: sentence —

[ ]:

[ ]:

[ ]:

0.2.10 3.1.8 from pretrained roberta-base Y: multi task 2 & X: sentence —

[ ]:

[ ]:

[ ]:

0.2.11 3.1.9 from pretrained roberta-base Y: single task 1 & X: sentence_no_contractions —

[ ]:

[ ]:

[ ]:

0.2.12 3.1.10 from pretrained roberta-base Y: multi task 2 & X: sentence_no_contractions —

[ ]:

[ ]:

[ ]: