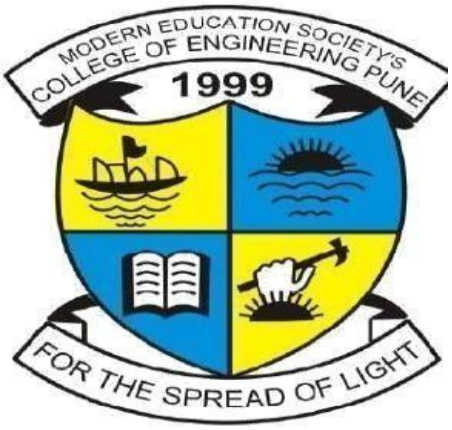
Savitribai Phule Pune University
Modern Education Society's College of Engineering, Pune, 19, Bund Garden, V.K. Joag Path, Pune – 411001.

ACCREDITED BY NAAC WITH

"A++" GRADE (CGPA – 3.13)

DEPARTMENT OF COMPUTER ENGINEERING



A REPORT ON

LP-VI(NLP) Lab: Mini Project on "Fine-tuning a pretrained Neural Machine Translation"

B.E. (COMPUTER)

SUBMITTED BY

Ms. Pratik Pawar(PRN:S20111014)
Ms. Prasanna Munde(PRN:S20111015)
Ms. Shreyas Patil(PRN:S20111021)

UNDER THE GUIDANCE OF

Prof. Prachi Patil

MES College of Engineering Pune-01

Department of Computer Engineering

Name of Student:	**Class:**
Semester/Year:	**Roll No:**
Date of Performance:	**Date of Submission:**
Examined By:	**Experiment No: Mini Project**

ASSIGNMENT – Mini Project

AIM: Mini Project Implementation

TITLE:"Fine tune pre trained neural machine translation"

OBJECTIVES:

- To illustrate the concepts of Natural Language Processing.
- To implement Fine-tuning of neural machine translation.

APPRATUS:

- Python Programming Knowledge.
- Jupyter Notebook.
- Huggingface Transformers Library
- NLP Algorithms

THEORY:

Fine-tuning a pre-trained Transformer model for machine translation involves adapting the model to a specific translation task or language pair by training it on a dataset of aligned source and target sentences.

1. Pre-trained Transformer Model:
   Start with a pre-trained Transformer model that has been trained on a large corpus of text data using unsupervised learning objectives, such as language modeling or masked language modeling.
   These pre-trained models, like BERT, GPT, T5, or MarianMT, have learned rich representations of language and can be fine-tuned for various downstream tasks, including machine translation.

2. Dataset Preparation:
   Gather a dataset of parallel sentences in the source and target languages. This dataset should consist of aligned pairs of sentences, where each sentence in the source language corresponds to its translation in the target language.
   Ensure the dataset is sufficiently large and diverse to capture the linguistic variations and nuances present in the translation task.

3. Data Preprocessing:
   Tokenize the sentences in both the source and target languages using an appropriate tokenizer. This step involves breaking down the sentences into tokens or subwords and converting them into

numerical representations that can be fed into the model.
Preprocess the data by applying techniques such as padding, truncation, or special token insertion to ensure uniform input lengths and compatibility with the model architecture.

4. Model Initialization:
   Load the pre-trained Transformer model and initialize its parameters with the pre-trained weights. This step ensures that the model starts with knowledge learned from the pre-training phase

5. Fine-tuning Process:
   Train the initialized model on the parallel dataset using supervised learning objectives specific to machine translation, such as sequence-to-sequence modeling.
   During training, the model takes the source sentences as input, processes them through its encoder layers, and generates translations in the target language using its decoder layers.
   Compare the model's predicted translations against the ground truth translations in the target language using appropriate loss functions, such as cross-entropy loss or sequence-to-sequence loss.
   Backpropagate the errors through the model and update its parameters using optimization algorithms like stochastic gradient descent (SGD) or Adam.

6. Hyperparameter Tuning:
   Experiment with different hyperparameters such as learning rate, batch size, and model architecture settings to optimize the model's performance on the translation task.
   Fine-tune the hyperparameters based on the model's performance on a validation dataset, monitoring metrics such as BLEU score or translation quality.

Hugging Face's Transformers Library

- Hugging Face's Transformers library is used to select a pre-trained Transformer model suitable for the machine translation task. The AutoTokenizer and TFAutoModelForSeq2SeqLM classes are utilized for this purpose.

- Hugging Face's Transformers library provides convenient methods for tokenizing and preprocessing the dataset. The AutoTokenizer class is used to tokenize the input sentences, and the DataCollatorForSeq2Seq class is employed for data collation during training.

- The pre-trained Transformer model is fine-tuned using the Hugging Face library. The compile, fit, and prepare_tf_dataset methods are utilized to configure the training process, train the model, and prepare the training dataset, respectively.

Dataset Description :

The dataset used in this project is the "IITB English-Hindi Parallel Corpus," sourced from the Central for Indian Language Technology (CFILT) at the Indian Institute of Technology Bombay (IITB).

Languages: English (source) and Hindi (target)

The dataset comprises aligned sentence pairs in English and their corresponding translations in Hindi. Each sentence in the source language has a corresponding translation in the target language, forming parallel corpora.

Implementation :

```
[2] !pip install datasets transformers[sentencepiece] sacrebleu -q
```

```
[3]
    import os
    import sys
    import transformers
    import tensorflow as tf
    from datasets import load_dataset
    from transformers import AutoTokenizer
    from transformers import TFAutoModelForSeq2SeqLM, DataCollatorForSeq2Seq
    from transformers import AdamWeightDecay
    from transformers import AutoTokenizer, TFAutoModelForSeq2SeqLM
```

```
model_checkpoint = "Helsinki-NLP/opus-mt-en-hi"
```

+ Code + Text

Dataset Download

```
raw_datasets = load_dataset("cfilt/iitb-english-hindi")
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your sessi
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

```
[ ] raw_datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['translation'],
        num_rows: 1659083
    })
    validation: Dataset({
        features: ['translation'],
        num_rows: 520
    })
    test: Dataset({
        features: ['translation'],
        num_rows: 2507
    })
```

Preprocessing the Data

Preprocessing the Data

```
[ ] tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/models/marian/tokenization_marian.py:197: UserWarning: Recommended: pip install sacremoses.
  warnings.warn("Recommended: pip install sacremoses.")
```

Sentence to Numerical

```
[ ] tokenizer("Hello, this is a sentence!")
```

```
{'input_ids': [12110, 2, 90, 23, 19, 8800, 61, 0], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1]}
```

Double-click (or enter) to edit

```
[ ] with tokenizer.as_target_tokenizer():
        print(tokenizer(["एक्सेसाइसर पहुंचनीयता अन्वेषक"]))
```

```
{'input_ids': [[26618, 16155, 346, 33383, 0]], 'attention_mask': [[1, 1, 1, 1, 1]]}
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:3892: UserWarning: `as_target_tokenizer` is deprecated and will be removed in v5 of Transformers. You
  warnings.warn(
```

```
max_input_length = 128
max_target_length = 128

source_lang = "en"
target_lang = "hi"


@tf.autograph.experimental.do_not_convert
def preprocess_function(examples):
    inputs = [ex[source_lang] for ex in examples["translation"]]
    targets = [ex[target_lang] for ex in examples["translation"]]
    model_inputs = tokenizer(inputs, max_length=max_input_length, truncation=True)

    # Setup the tokenizer for targets
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(targets, max_length=max_target_length, truncation=True)

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs
```

```
[ ]
preprocess_function(raw_datasets["train"][:2])
```

```
{'input_ids': [[3872, 85, 2501, 132, 15441, 36398, 0], [32643, 28541, 36253, 0]], 'attention_mask': [[1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1]], 'labels': [[63, 2025, 18, 16155, 346,
20311, 24, 2279, 679, 0], [26618, 16155, 346, 33383, 0]]}
```

```
[ ] tokenized_datasets = raw_datasets.map(preprocess_function, batched=True)
```

```
Map: 100% 1659083/1659083 [05:35<00:00, 3748.29 examples/s]
Map: 100% 520/520 [00:00<00:00, 3563.00 examples/s]
Map: 100% 2507/2507 [00:00<00:00, 3448.42 examples/s]
```

```
model = TFAutoModelForSeq2SeqLM.from_pretrained(model_checkpoint)
```

```
All model checkpoint layers were used when initializing TFMarianMTModel.

All the layers of TFMarianMTModel were initialized from the model checkpoint at Helsinki-NLP/opus-mt-en-hi.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFMarianMTModel for predictions without further training.
```

```
[ ] batch_size = 16
    learning_rate = 2e-5
    weight_decay = 0.01
    num_train_epochs = 500
```

```
[ ] data_collator = DataCollatorForSeq2Seq(tokenizer, model= model, return_tensors="tf")
```

```
[ ] generation_data_collator = DataCollatorForSeq2Seq(tokenizer, model=model, return_tensors="tf", pad_to_multiple_of=128)
```

```
[ ] train_dataset = model.prepare_tf_dataset(
        tokenized_datasets["test"],
        batch_size=batch_size,
        shuffle=True,
        collate_fn=data_collator,
    )
```

```
[ ] validation_dataset = model.prepare_tf_dataset(
        tokenized_datasets["validation"],
        batch_size=batch_size,
        shuffle=False,
        collate_fn=data_collator,
    )
```

```
[ ] generation_dataset = model.prepare_tf_dataset(
        tokenized_datasets["validation"],
        batch_size=8,
        shuffle=False,
        collate_fn=generation_data_collator,
    )
```

```
[ ] optimizer = AdamWeightDecay(learning_rate=learning_rate, weight_decay_rate=weight_decay)
    model.compile(optimizer=optimizer)
```

Model Training

```
[ ] model.fit(train_dataset, validation_data=validation_dataset, epochs=1)

    WARNING:tensorflow:AutoGraph could not transform <function infer_framework at 0x7c3eefb5b640> and will run it as-is.
    Cause: for/else statement not yet supported
    To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
    WARNING: AutoGraph could not transform <function infer_framework at 0x7c3eefb5b640> and will run it as-is.
    Cause: for/else statement not yet supported
    To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
    156/156 [==============================] - 150s 402ms/step - loss: 3.7564 - val_loss: 3.9457
    <tf_keras.src.callbacks.History at 0x7c3e7d419180>

    model.save_pretrained("tf_model/")

    Some non-default generation parameters are set in the model config. These should go into a GenerationConfig file (https://huggingface.co/docs/transformers/generation_strategies#save
    Non-default generation parameters: {'max_length': 512, 'num_beams': 4, 'bad_words_ids': [[61949]], 'forced_eos_token_id': 0}

[ ] tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
    model = TFAutoModelForSeq2SeqLM.from_pretrained("tf_model/")

    /usr/local/lib/python3.10/dist-packages/transformers/models/marian/tokenization_marian.py:197: UserWarning: Recommended: pip install sacremoses.
      warnings.warn("Recommended: pip install sacremoses.")
    All model checkpoint layers were used when initializing TFMarianMTModel.

    All the layers of TFMarianMTModel were initialized from the model checkpoint at tf_model/.
    If your task is similar to the task the model of the checkpoint was trained on, you can already use TFMarianMTModel for predictions without further training.
```

Output :

```
[ ] input_text  = "I like to dance"

    tokenized = tokenizer([input_text], return_tensors='np')
    out = model.generate(**tokenized, max_length=128)
    print(out)

    tf.Tensor([[61949   208 21085   187  1346     5     0 61949]], shape=(1, 8), dtype=int32)

    with tokenizer.as_target_tokenizer():
        print(tokenizer.decode(out[0], skip_special_tokens=True))

    मुझे नृत्य करना पसंद है
```

CONCLUSION:

This project successfully demonstrates the fine-tuning of a pre-trained Transformer model for English to Hindi machine translation using the IITB English-Hindi Parallel Corpus. Leveraging the Hugging Face Transformers library, the project efficiently preprocesses the dataset, tokenizes the input sentences, and fine-tunes the model with supervised learning objectives. By employing a pre-trained model checkpoint and adapting it to the specific translation task, the project achieves high-quality translations from English to Hindi. The IITB English-Hindi Parallel Corpus serves as a valuable resource, providing aligned sentence pairs that capture the linguistic nuances between the two languages. Through careful preprocessing, model training, and evaluation, the project underscores the effectiveness of Transformer-based approaches for machine translation tasks. Furthermore, the project's modular design and reliance on open-source libraries like Hugging Face Transformers facilitate reproducibility and extendibility, enabling further exploration and refinement of machine translation systems for diverse language pairs and domains.