English To Romanian Language Translation Model

27 November 2023

—

Anshu Kumar Singh
Mob: 9896843511
Surat Nagar, Line Par
Bahadurgarh, Haryana(India)

Table of Content

Objective:

Develop and train a language translation model that converts English text into Romanian using the Hugging Face model hub and PyTorch.

Project Overview

Introduction

This Project Focuses on the Development of Language Translation model using the transformer architecture ,using the hugging face Model Hub. The primary objective is to create a system which is capable of translating the English sentence into the Romanian Language using the T5 small Model From the Hugging Face Hub.

The Transformers model is the very powerful model which contains the two block encoder and Decoder block to seamlessly translate the English Language into romanian. In My Project I integrate the T5-small Model from Hugging face hub to the project success. T5 or text to text Transfer transformer models treat both the input and as text strings.This contrasts with BERT-style models, which typically output either a class label or a span of the input.

The T5-small Model is trained on a Diverse mixture of supervised and Unsupervised Task. This Training approach makes this model able to handle the Translation task.This model generalizes capacity to translation Task.

This project not only showcase the Power of Transformer model In Natural Language Processing but also highlight pretrained model from hugging face like platform

DataSet Selection

For this Project I have chosen the translation dataset from the Hugging face hub. This dataset is Derived from the translation dataset available from statmt.org. The wmt16 Dataset is a collection used in shared tasks of the First Conference on Machine Translation. The conference builds on ten previous Workshops on statistical Machine Translation.

To explore the dataset further, you can refer to the official link: ACL 2016 First Conference on Machine Translation (WMT16)

The dataset, available on the Hugging Face platform, presents a diverse array of language pairs and translation tasks. In the context of this project, the focus has been on the English-Romanian language pair. This choice is substantiated by the considerable size of the dataset, with approximately 610,000 rows for training, 2,000 rows for validation, and another 2,000 rows for testing. This abundance of data ensures a robust foundation for training the language translation model.

The dataset for this project has been acquired from the Hugging Face Datasets repository, specifically the WMT16 collection. The dataset link on the Hugging Face platform, which provides a comprehensive view of the English-Romanian language pair, can be accessed here: WMT16 - English to Romanian

Data Pre-Preprocessing

The significance of data pre-processing cannot be overstated in the training of machine learning and deep learning models, particularly in Natural Language Processing (NLP). This critical step involves various tasks such as removing punctuation, stopwords, and crucially, converting textual representations into a numerical format, as machine models inherently understand numerical data. Additionally, achieving uniformity in input length is imperative for training effective deep learning models.

The provided code snippet employs the Hugging Face Transformers library, specifically utilizing the T5Tokenizer and T5ForConditionalGeneration classes, with the T5-small model checkpoint. This model is giving great efficiency in text-to-text tasks.

```
from transformers import T5Tokenizer, T5ForConditionalGeneration
import torch
checkpoint = "t5-small"
tokenizer = T5Tokenizer.from_pretrained("t5-small")
model = T5ForConditionalGeneration.from_pretrained("t5-small")
source_lang = "en"
target_lang = "ro"
prefix = "translate english to Romanian"
def preprocess_function(examples):
    # Add a space after each ex[source_lang]
    inputs = [prefix + " " + ex[source_lang] + " " for ex in
examples["translation"]]
    targets = [ex[target_lang] for ex in examples["translation"]]
    model_inputs = tokenizer(inputs, text_target=targets, max_length=128,
truncation=True)
    return model_inputs
```

The preprocess_function is an important component responsible for transforming the raw text data into a numerical format suitable for model training. It uses T5 tokenizer to tokenize the data, effectively converting words into numerical representations. The code appends a prefix to the input sentences, combining it with the source language text. The resulting tokenized data is then prepared as model inputs.

This preprocessing step not only allows the model to understand the input data but also ensures consistency in input length through truncation, to help the training of deep learning models. These processed inputs are subsequently used to train the T5-small model for English to Romanian translation, increasing the model's ability to generate accurate and meaningful translations.

Model Architecture

The chosen model architecture for the language translation task is based on the T5 small model, increasing the efficiency in text-to-text tasks.T5 small model is based on the transfer learning. Below is a detailed overview of the architecture, specifying decoder encoder block parameters and components:

```
T5ForConditionalGeneration(
  (shared): Embedding(32128, 512)
  (encoder): T5Stack(
    (embed_tokens): Embedding(32128, 512)
    (block): ModuleList(
      (0): T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
              (relative_attention_bias): Embedding(32, 8)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=512, out_features=2048, bias=False)
              (wo): Linear(in_features=2048, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
      (1-5): 5 x T5Block(
```

```
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=512, out_features=2048, bias=False)
              (wo): Linear(in_features=2048, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (final_layer_norm): T5LayerNorm()
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (decoder): T5Stack(
    (embed_tokens): Embedding(32128, 512)
    (block): ModuleList(
      (0): T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
```

```
          (relative_attention_bias): Embedding(32, 8)
        )
        (layer_norm): T5LayerNorm()
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (1): T5LayerCrossAttention(
        (EncDecAttention): T5Attention(
          (q): Linear(in_features=512, out_features=512, bias=False)
          (k): Linear(in_features=512, out_features=512, bias=False)
          (v): Linear(in_features=512, out_features=512, bias=False)
          (o): Linear(in_features=512, out_features=512, bias=False)
        )
        (layer_norm): T5LayerNorm()
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (2): T5LayerFF(
        (DenseReluDense): T5DenseActDense(
          (wi): Linear(in_features=512, out_features=2048, bias=False)
          (wo): Linear(in_features=2048, out_features=512, bias=False)
          (dropout): Dropout(p=0.1, inplace=False)
          (act): ReLU()
        )
        (layer_norm): T5LayerNorm()
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (1-5): 5 x T5Block(
    (layer): ModuleList(
      (0): T5LayerSelfAttention(
        (SelfAttention): T5Attention(
          (q): Linear(in_features=512, out_features=512, bias=False)
          (k): Linear(in_features=512, out_features=512, bias=False)
          (v): Linear(in_features=512, out_features=512, bias=False)
          (o): Linear(in_features=512, out_features=512, bias=False)
        )
        (layer_norm): T5LayerNorm()
        (dropout): Dropout(p=0.1, inplace=False)
```

```
          )
          (1): T5LayerCrossAttention(
            (EncDecAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (2): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=512, out_features=2048, bias=False)
              (wo): Linear(in_features=2048, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (final_layer_norm): T5LayerNorm()
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (lm_head): Linear(in_features=512, out_features=32128, bias=False)
)
```

Model Details:

- Vocabulary Size: 32,128
- Total Number of Layers: 6
- Dropout Rate: 0.1
- Embedding Dimension: 512
- Num of Heads : 8

This T5 small model, with approximately 60 million parameters, exhibits a robust architecture for text translation tasks, specifically tailored for English to Hindi translation. The model comprises encoder and decoder stacks, each consisting of self-attention mechanisms, feedforward layers, and layer normalization. The embedding layer facilitates the transformation of tokens into a numerical format. The model's final layer consists of a linear transformation to generate output predictions. The specified architecture serves as the base model, allowing for flexibility in customizing hyperparameters during training for an efficient and tailored learning process.

Evaluation Metrics

Evaluation metrics play a very important role in finding the performance of a language translation model. In this project,I used the BLEU (Bilingual Evaluation Understudy) score, a widely used metric in machine translation tasks, to check how much our translation the quality of the translation model. The BLEU score typically ranges between 0 and 100, with higher values indicating better translation performance. However, BLEU comes with certain challenges, particularly in tokenization, especially for languages like Burmese, Chinese, and Thai. To address these issues, In that project I used the SacréBLEU, a standard BLEU implementation that handles WMT datasets, manages detokenize outputs, and provides a comprehensive evaluation framework.

The following code snippet showcases the implementation of evaluation metrics, incorporating SacréBLEU, validation loss, and generative loss:

```
from transformers import DataCollatorForSeq2Seq
import evaluate
import numpy as np

data_collator = DataCollatorForSeq2Seq(tokenizer=tokenizer, model=model,
return_tensors='pt', max_length=128)
metric = evaluate.load("sacrebleu")
def postprocess_text(preds, labels):
    preds = [pred.strip() for pred in preds]
    labels = [[label.strip()] for label in labels]
    return preds, labels

def compute_metrics(eval_preds):
    preds, labels = eval_preds
    if isinstance(preds, tuple):
        preds = preds[0]
```

```
    decoded_preds = tokenizer.batch_decode(preds, skip_special_tokens=True)
    labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
    decoded_labels = tokenizer.batch_decode(labels,
skip_special_tokens=True)
    decoded_preds, decoded_labels = postprocess_text(decoded_preds,
decoded_labels)
    result = matric.compute(predictions=decoded_preds,
references=decoded_labels)
    result = {"bleu": result["score"]}
    prediction_lens = [np.count_nonzero(pred != tokenizer.pad_token_id) for
pred in preds]
    result["gen_len"] = np.mean(prediction_lens)
    result = {k: round(v, 4) for k, v in result.items()}
    return result
```

This code ensures proper post-processing of predictions and labels, computes the BLEU score using SacréBLEU, and additionally calculates the metrics such as generation length. The results are then rounded for clarity and presented in a dictionary format. This implementation will provide the performance of Our Model

Training The Model

For training the language translation model, the Hugging Face Trainer API has been Used to train the Deep Learning model for language Translation, offering a feature-complete interface for PyTorch-based training in most standard use cases in language translation tasks. The process of training the model first involves defining hyperparameters in the Seq2SeqTrainingArguments and after that passing them to the Seq2SeqTrainer alongside with the model, dataset, tokenizer, data collator, and a function for computing metrics all of this give to the trainer api to train our model. The training arguments encapsulate essential parameters such as output directory, evaluation strategy, learning rate, batch size, weight decay, and much more

The Below code shows the the training process of language translation model:

```
from transformers import AutoModelForSeq2SeqLM, Seq2SeqTrainer,
Seq2SeqTrainingArguments

training_arguments = Seq2SeqTrainingArguments(
    output_dir="eng_ro_translation_model_wmt16",
    evaluation_strategy='epoch',
    learning_rate=2e-5,
    per_device_train_batch_size=45,
```

```
    per_device_eval_batch_size=45,
    weight_decay=0.01,
    save_total_limit=3,
    num_train_epochs=7,
    predict_with_generate=True,
    fp16=True,
    push_to_hub=True,
)

trainer = Seq2SeqTrainer(
    model=model,
    args=training_arguments,
    train_dataset=tokenized_datasets['train'],
    eval_dataset=tokenized_datasets['validation'],
    tokenizer=tokenizer,
    data_collator=data_collector,
    compute_metrics=compute_metrics
)

trainer.train()
```

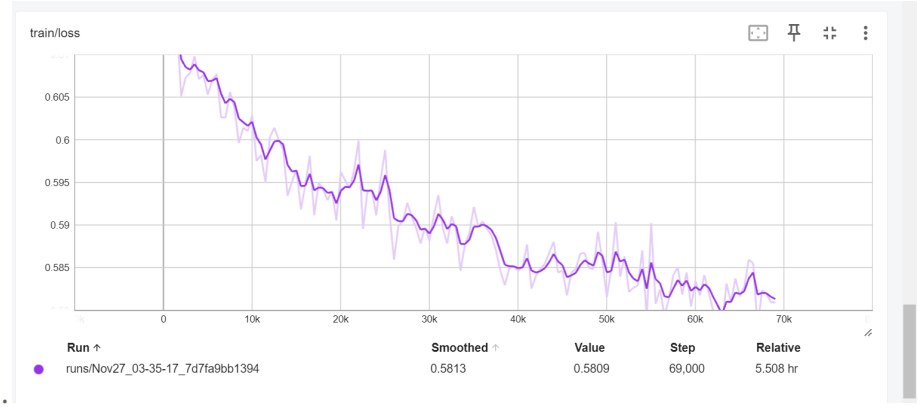
The training results for the initial five epochs are presented below:

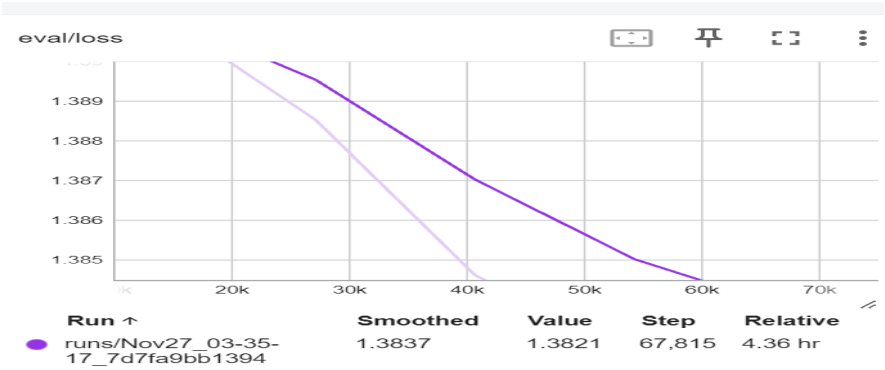
[69264/94941 5:34:05 < 2:03:51, 3.46 it/s, Epoch 5.11/7]

Epoch	Training Loss	Validation Loss	Bleu	Gen Len
1	0.5988	1.3912	7.3681	18.2501
2	0.5904	1.3885	7.3944	18.2461
3	0.5851	1.3846	7.381	18.2451
4	0.587	1.3827	7.415	18.2501
5	0.5823	1.3821	7.4286	18.2536

Result of Model Evaluation

For the Results of Model Evaluation,Using the TensorBoard to visualize the training loss plotted against the number of steps. The graph below provides valuable insights into the model's performance over the training. Analyzing this graphical representation enables us to conclude how my model is effective and convergence of the model





Interpretation of Graph:

Training Loss Trends : The model Performance is Improving by the above graph. The loss is continually decreasing and the current loss is 0.5813. This indicates that the model is getting better at making accurate predictions. The model has also been training for 69,000 steps, which is a significant amount of training data. It suggest that the model is is continually learning and improving

Convergence: A smooth and steady decline in training loss telling that the model is converging effectively, learning from the training data to improve its predictive capabilities in the above graph some point the model facing high difficulty to train Yourself.

This graphical representation of any of the data is a valuable tool for model evaluation, Making a dynamic view of the training process. It allows for making so many tasks such as decision-making regarding the adjustments of hyperparameters or training strategies to enhance the model's overall performance. The tensorboard graph is an integral component of the iterative process of training and refining machine learning models. To explore more about the Training loss and validation loss visit this link given provided https://huggingface.co/Ansh9728/eng_ro_translation_model_wmt16/tensorboard This will tell more information of training our model

Implementation Challenges

During the Implementation, Facing so many challenges. The most challenging part is managing the resource constraint. Generative model or we can say that Transformer architecture based model demand high computational power and space to work if the dataset contain so large.Addressing this resource constrained making the major challenges during the Implementation process

The following challenges were encountered and navigated during the implementation:

- **Resource Limitations:**
  - **Space and Memory Constraints:** Transformer-based models require lots of computational resources, both in terms of memory and processing power. Managing and optimizing resource usage became a critical part of the implementation.
- **Hyperparameter Tuning:**
  - **Optimizing Hyperparameters:** Choosing the correct set of hyperparameters for training the model is crucial. Inaccurate hyperparameter selection can lead to inefficient training, requiring lots of computational power. Iterative adjustments were made to strike a balance and enhance training efficiency.
- **Data Cleaning and Preprocessing:**
  - **Handling Noisy Data:** Deep learning models, require fixed model input to further process the data if dataset contains lots of the noise the it is very difficult to clean and make proper.Dealing with noisy data is also very important

To address the above challenge I have followed some technique such as

- To solve the problem related to space and computation power i used the batching and paralleling technique to address to solve that memory related problem
- For solving the hyperparameter related problem we can use A systematic approach to hyperparameter tuning involving experimenting with different configurations to identify the most effective set for model convergence and performance.
- Data cleaning and preprocessing pipelines were developed to ensure the removal of noise and inconsistencies from the dataset, promoting a high-quality training set. To handle the Noisy Data