University of Cagliari

Faculty of Science – Master's Degree in Computer Science

Fine Tuning and Training of a Transformer for translation from English to Italian

A.Y. 2021/2022

Giulia Cerniglia [m. 60/73/65229]

Faccin Luca [m. 60/73/65236]

Summary

Introduction 3
Technologies used: 3
Transformers 3
PyTorch 3
Numpy 3
Sacrebleu 3
Google Colab 3
Dataset 4
Transformer Structure 4
The pre-trained model 5
Tokenization 5
Train and fine-tune the model. 6
Results 7
Metric 7
Model's results 7
Results with some translation examples 8
Conclusions 9
Google colab repository 10

Introduction

The following project aimed to create a program, written in Python, that allows translations from English to Italian, through a pre-trained Transformer, and we do a new training with a dataset specific to the task, this procedure is known as **fine-tuning**.

There are significant benefits to using a pre-trained model. It reduces computation costs, and your carbon footprint, and allows you to use state-of-the-art models without having to train one from scratch.

Technologies used:

Transformers

Transformers provides thousands of pre-trained models to perform tasks on different modalities such as text, vision, and audio. Transformers provides APIs to quickly download and use those pre-trained models on a given text, fine-tune them on your datasets, and then share them with the community.

PyTorch

PyTorch is a Python package that provides two high-level features:

- Tensor computation (like NumPy) with strong GPU acceleration.
- Deep neural networks built on a tape-based autograd system.

It allows you to reuse your favorite Python packages such as NumPy, SciPy, and Cython to extend PyTorch when needed.

Numpy

NumPy is an open-source project that aims to enable numerical computing with Python. It was created in 2005, based on early work on the Numeric and Numarray libraries. NumPy is a open-source software, free for everyone, and released under the liberal terms of the modified BSD license.

Sacrebleu

SacreBLEU provides hassle-free computation of shareable, comparable, and reproducible BLEU scores. It produces the official WMT scores but works with plain text. It also knows all the standard test sets and handles downloading, processing, and tokenization for the user.

Google Colab

It's a free tool to allow you to write programs in Python. Colab notebooks let you combine executable and rich text code into a single document, along with images, HTML, LaTeX, and more. When you create Colab notebooks, they are stored in your Google Drive account. You can easily share Colab notebooks with collaborators or friends and give them the ability to comment or edit.

Dataset

The dataset that was used is “Iwslt2017”, the acronym stands for International Workshop on Spoken Language Translation and is a yearly scientific workshop, associated with an open evaluation campaign on spoken language translation, where both scientific papers and system descriptions are presented. The dataset is composed of 231619 in the train, 1566 in the test set, and 929 for the validation.

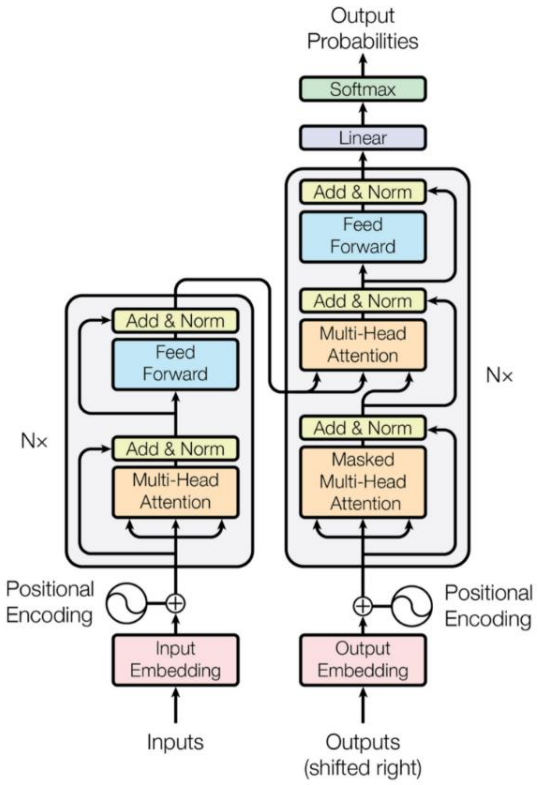
Every line of the dataset contains the English phrase and its Italian translation.

Here is an example:

{'en': 'So you think it's real?', 'it': 'Quindi pensi che sia reale?'}

Transformer Structure

Recursive networks receive incoming data sequentially in the order of arrival, which is why dependencies between time-distant data are difficult to model. The Transformer architecture was designed to address this problem.



It has an architecture to transform one sequence into another with the help of two parts (Encoder and Decoder) but differs from seq-2-seq models because it does not imply any recurring network (GRU, LSTM, etc.).
The encoder is based on the overlapping of two blocks, the **Multi-Head Attention** and followed by a dense block (**Feed Forward).**

The decoder is very similar to the encoder but differs in the addition of a second block of Multi-Head Attention.

Both Encoder and Decoder are composed of modules that can be stacked on top of each other several times, as suggested by N x in the figure. Inputs and outputs (phrases) are special **embedding** with positional encoding because we cannot use strings directly.

On a high level, the encoder maps an input sequence into an abstract continuous representation that holds all the learned information of that input. The decoder then takes that continuous representation and step by step generates a single output while also being fed the previous output.

An important aspect concerns the positional encoding of the different words, as depending on the position of a certain word in the sentence it can take on different meanings; Then, the positions of those words are passed as parameters.

The attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and outputs are all vectors. The output is calculated as a weighted sum of the values, where the weights assigned to each value are calculated by a query compatibility function with the corresponding key.

The pre-trained model

The model is called MarianMT and is based on encoder-decoder architecture and was originally trained using the Marian library.

Marian is written entirely in C++. This library supports faster training and translation. As it has minimal dependencies, it provides support to optimize MPI-based multi-node training, efficient batched beam search, compact implementations of new models, etc.

MarianMT model was trained on Open Parallel Corpus (OPUS) which is a collection of translated texts from the web.

There are around 1300 models which support multiple language pairs. All these models follow the same naming convention – Helsinki-NLP/opus-mt-{src}-{target}, where src and target are the two-character language codes.

From MarianMT we took both the model and the tokenizer.

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

model_checkpoint = "Helsinki-NLP/opus-mt-en-it"

tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
model = AutoModelForSeq2SeqLM.from_pretrained(model_checkpoint)
```

Tokenization

For the model to make sense of the data, we use a tokenizer that can help with:

- Splitting the text into words and sub-words
- Mapping each token to an integer

We initialized the tokenizer in step 1 and will use it here to get the tokens for input text. The output of the tokenizer is a dictionary containing two keys: input ids and attention mask. Input ids are the unique identifiers of the tokens in a sentence. An attention mask is used to batch the input sequence together and indicates whether the token should be attended by

our model or not. Token with attention mask value 0 means token will be ignored and 1 means tokens are important and will be taken for further processing.

This is an example of tokenization:

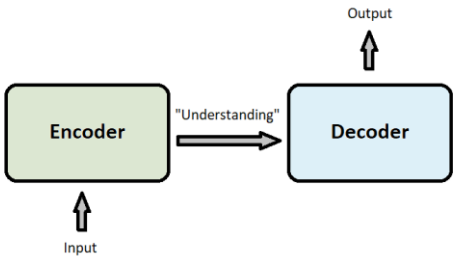
```
[ ] tokenizer(["Hello, this one sentence!", "This is another sentence."])

    {'input_ids': [[226, 1127, 3, 62, 133, 11347, 49, 0], [163, 24, 928, 11347, 2, 0]], 'attention_mask': [[1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1]]}
```

All the ones in the attention mask tell us that every word is important for the comprehension of the phrase.

Train and fine-tune the model.

We used a model Sequence-to-Sequence (Seq2Seq), which is a general end-to-end framework that maps sequences in the source domain to sequences in the target domain.



Seq2Seq model first reads the source sequence using an encoder to build vector-based 'understanding' representations, then passes them through a decoder to generate a target sequence, so it's also referred to as the encoder-decoder architecture (like in the picture).

In the table below we decide the parameters of the seq2seq:

```
from transformers import Seq2SeqTrainingArguments,DataCollatorForSeq2Seq
batch_size = 16
model_name = model_checkpoint.split("/")[-1]
args = Seq2SeqTrainingArguments(
    output_dir = f"{model_name}-finetuned-{source_lang}-to-{target_lang}",
    evaluation_strategy = "epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    weight_decay=0.01,
    save_total_limit=3,
    num_train_epochs=1,
    predict_with_generate=True
)
```

- Learning rate: The learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum loss function.
- Output directory: the directory used to save the model.
- Evaluation strategy and num_train_epoch: The evaluation strategy is the epoch and its number is one cause it is already trained.

- Weight decay: is a regularization technique that is used in machine learning to reduce the complexity of a model and prevent overfitting. Weight decay can be implemented by modifying the update rule for the weights such that the gradient is not only based on the training data but also the weight decay term.
- Predict with generate: Whether to use generate to calculate generative metrics.

Results

Metric

As a metric for the model, we opted for **Bleu**, a metric for automatically evaluating machine-translated text. The BLEU score is a number between zero and one that measures the similarity of the machine-translated text to a set of reference translations. A value of 0 means that the machine-translated output has no overlap with the reference translation while a value of 1 means there is perfect overlap with the reference translations.

It has been shown that BLEU scores correlate well with the human judgment of translation quality. Note that even human translators do not achieve a perfect score of 1.0.

BLEU Score	Interpretation
< 10	Almost useless
10 - 19	Hard to get the gist
20 - 29	The gist is clear, but has significant grammatical errors
30 - 40	Understandable to good translations
40 - 50	High quality translations
50 - 60	Very high quality, adequate, and fluent translations
> 60	Quality often better than human

Model's results

The model was measured in two ways: the loss function and the bleu metric; the latter was explained above, the first one is a number indicating how bad the model's prediction was. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater).

Epoch	Training Loss	Validation Loss	Bleu	Gen Len
1	1.176600	1.209783	33.159800	25.163600

The loss was measured in both the training set and the validation set and with two pretty similar results, this is a good signal because it means that the model, even if placed in a real situation, is not overfitting nor underfitting.

As for the bleu score, we have a result of 33.15, which is an average of all the results multiplied by one hundred taken to the tens (normally I remember that the value is between zero and

one). In the table shown in the metric section of the report a score of 33 is labeled as “understandable to good translations”, but for how the bleu score works the model was penalized by the fact that the metric did not have a list of different translations as references, as the metric would prefer, and therefore if the model uses a synonym not present in the test set "solution" it counts it as an error.

Results with some translation examples

```
[ ] from transformers import MarianMTModel, MarianTokenizer
    src_text = ['This model is really efficient e translates everything perfectly!!!']

    model_name = 'opus-mt-en-it-finetuned-en-to-it/checkpoint-14000'
    tokenizer = MarianTokenizer.from_pretrained(model_name)
    print(tokenizer.supported_language_codes)

[ ] model = MarianMTModel.from_pretrained(model_name)
    translated = model.generate(**tokenizer(src_text, return_tensors="pt", padding=True))

[ ] [tokenizer.decode(t, skip_special_tokens=True) for t in translated]

    ['Questo modello è davvero efficiente e traduce tutto perfettamente!!!']
```

This first translation is perfect because there is nothing particularly difficult, even if we wrote “e” instead of “and” the model was not affected. Compared to the pre-trained model available on the site (https://huggingface.co/Helsinki-NLP/opus-mt-en-it) the translation is the same: ‘Questo modello è davvero efficiente e traduce tutto perfettamente!!!’

```
from transformers import MarianMTModel, MarianTokenizer
src_text = ['Several years ago here at TED, Peter Skillman introduced a design challenge called the marshmallow challenge.']

model_name = 'opus-mt-en-it-finetuned-en-to-it/checkpoint-14000'
tokenizer = MarianTokenizer.from_pretrained(model_name)
print(tokenizer.supported_language_codes)

[ ] model = MarianMTModel.from_pretrained(model_name)
    translated = model.generate(**tokenizer(src_text, return_tensors="pt", padding=True))

[ ] [tokenizer.decode(t, skip_special_tokens=True) for t in translated]

    ['Diversi anni fa qui a TED, Peter Skillman ha introdotto una sfida di design chiamata sfida del marshmallow.']
```

This second transition is practically perfect, because how it is in the pre-trained model the phrase literally should be ‘Diversi anni fa qui a TED, Peter Skillman ha introdotto una sfida di design chiamata la sfida marshmallow’: “sfida marshmallow” instead of “sfida del marshmallow”.

```
from transformers import MarianMTModel, MarianTokenizer
src_text = ['Midway upon the journey of our life I found myself within a forest dark, For the straightforward pathway had been lost. ']

model_name = 'opus-mt-en-it-finetuned-en-to-it/checkpoint-14000'
tokenizer = MarianTokenizer.from_pretrained(model_name)
print(tokenizer.supported_language_codes)

[ ] model = MarianMTModel.from_pretrained(model_name)
    translated = model.generate(**tokenizer(src_text, return_tensors="pt", padding=True))

[ ] [tokenizer.decode(t, skip_special_tokens=True) for t in translated]

    ["A metà del viaggio della nostra vita mi ritrovai all'interno di una foresta buia, perché il percorso diretto era andato perduto."]
```

The third translation was the hardest cause it was from Dante's Divina Commedia, and therefore in a non-modern way of speaking, but the transcription is literal, although this makes it a bit clanky to the ear. The pre-trained model translated it as: 'A metà del viaggio della nostra vita mi sono ritrovato all'interno di una foresta buia, perché il percorso diretto era stato perso', a small difference on how they decided to adapt the last verb, but both are very acceptable.

```
[83] raw_datasets["test"][1400]

     {'translation': {'en': 'And so, there are different ones, but the beauty of this is a molecule of uranium has a million times as much energy as a
     molecule of, say, coal, and so -- if you can deal with the negatives, which are essentially the radiation -- the footprint and cost, the potential, in
     terms of effect on land and various things, is almost in a class of its own.',
       'it': "Ce ne sono diversi, dunque,  ma il bello è che una molecola di uranio  sprigiona un milione di volte più energia di una molecola di, diciamo,
     carbone  quindi, se riesci a gestire gli aspetti negativi  , il potenziale,  in termini di effetti sull'ambiente ed altro,  ne fa quasi una classe a sé
     stante."}}

[84] from transformers import MarianMTModel, MarianTokenizer
     src_text = ['And so, there are different ones, but the beauty of this is a molecule of uranium has a million times as much energy as a molecule of, say,
     #choice = input("\nWrite your phrase to convert in italian:\n")
     #src_text = [choice]
     model_name = 'opus-mt-en-it-finetuned-en-to-it/checkpoint-14000'
     tokenizer = MarianTokenizer.from_pretrained(model_name)
     print(tokenizer.supported_language_codes)

[86] model = MarianMTModel.from_pretrained(model_name)
     translated = model.generate(**tokenizer(src_text, return_tensors="pt", padding=True))

[87] print("Traduzione: ")
     [tokenizer.decode(t, skip_special_tokens=True) for t in translated]

     Traduzione:
     ["E quindi ce ne sono di diversi, ma la bellezza è che una molecola di uranio ha un milione di volte più energia di una molecola di carbone, per
     esempio, e così -- se si riesce ad affrontare i negativi, che sono essenzialmente la radiazione -- l'impronta e il costo, il potenziale, in termini di
     effetto sulla terra e varie cose, è quasi in una classe a sé."]
```

This last translation comes directly from the test set, so we also have the reference for the model's work, the phrase is pretty long, convoluted and the Italian reference takes quite a lot of liberties (forgets to translate "this" from "this molecule", decide to use "give off" instead of has, etc. The model's translation also uses an impersonal form for the molecule getting rid of "this", but apart from that the translation is pretty accurate even if too literal and sometimes loses focus from the context (something it can't grasp).

Conclusions

In our case, we obtained good results in terms of quality. Our model can translate medium-length sentences in a fairly accurate way as shown in the examples of translation.
But as said before the model was penalized in the bleu score, per example in the second example of translation present in this report:

Our model candidate is → Diversi anni fa qui a TED, Peter Skillman ha introdotto una sfida di design chiamata la sfida del marshmallow

And the test reference is → Parecchi anni fa qui a TED, Peter Skillman introdusse una sfida di progettazione chiamata la sfida marshmallow

And the bleu result for this translation is 0.36, but this is undoubtedly too severe a mark for this translation.

Using a pre-trained model is easier and faster than building a new one from scratch with the drawback of not being able to decide the structure of the model.

Google colab repository

To use the advantages of the GPU, we leave the file of Google Colab:
https://colab.research.google.com/drive/1V_03ge3vrHzs9e4Srx9uLFk34YF7J6XA?usp=share_link