Text Data Augmentation with MarianMT

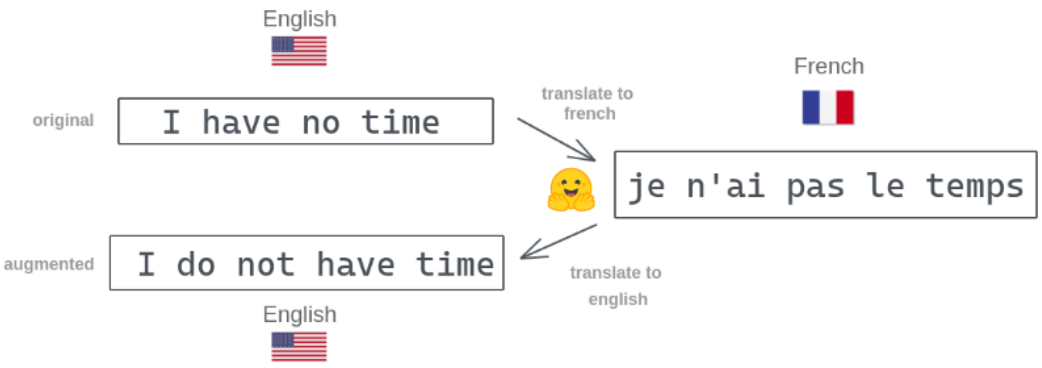
Hugging Face recently released 1008 translation models for almost 140 languages on their model hub.

These models were originally trained by Jörg Tiedemann of the Language Technology Research Group at the University of Helsinki. They were trained on the Open Parallel Corpus(OPUS) using a neural machine translation framework called MarianNMT.

In this post, I will explain how you can use the MarianMT models to augment data text data.

Back TranslationPermalink

We will use a data augmentation technique called “Back Translation”. In this, we take an original text written in English. Then, we convert it into another language (eg. French) using MarianMT. We translate the French text back into English using MarianMT. We keep the back-translated English text if it is different from the original English sentence.



Augmentation Process

Open in Colab

Permalink

First, we need to install Hugging Face transformers and Moses Tokenizers with the following command

```
pip install -U transformers
pip install mosestokenizer
```

After installation, we can now import the MarianMT model and tokenizer.

```
from transformers import MarianMTModel, MarianTokenizer
```

Then, we can create a initialize the model that can translate from English to Romance languages. This is a single model that can translate to any of the romance languages()

```
target_model_name = 'Helsinki-NLP/opus-mt-en-ROMANCE'
target_tokenizer = MarianTokenizer.from_pretrained(target_model_name)
target_model = MarianMTModel.from_pretrained(target_model_name)
```

Similarly, we can initialize models that can translate Romance languages to English.

```
en_model_name = 'Helsinki-NLP/opus-mt-ROMANCE-en'
en_tokenizer = MarianTokenizer.from_pretrained(en_model_name)
en_model = MarianMTModel.from_pretrained(en_model_name)
```

Next, we write a helper function to translate a batch of text given the machine translation model, tokenizer and the target romance language.

```
def translate(texts, model, tokenizer, language="fr"):
    # Prepare the text data into appropriate format for the model
    template = lambda text: f"{text}" if language == "en" else f">>{language}<< {text}"
    src_texts = [template(text) for text in texts]

    # Tokenize the texts
    encoded = tokenizer.prepare_seq2seq_batch(src_texts)

    # Generate translation using model
    translated = model.generate(**encoded)

    # Convert the generated tokens indices back into text
    translated_texts = tokenizer.batch_decode(translated, skip_special_tokens=True)

    return translated_texts
```

Next, we will prepare a function to use the above `translate()` function to perform back translation.

```
def back_translate(texts, source_lang="en", target_lang="fr"):
    # Translate from source to target language
    fr_texts = translate(texts, target_model, target_tokenizer,
                         language=target_lang)

    # Translate from target language back to source language
    back_translated_texts = translate(fr_texts, en_model, en_tokenizer,
                                      language=source_lang)

    return back_translated_texts
```

Now, we can perform data augmentation using back-translation from English to Spanish on a list of sentences as shown below.

```
en_texts = ['This is so cool', 'I hated the food', 'They were very helpful']

aug_texts = back_translate(en_texts, source_lang="en", target_lang="es")
print(aug_texts)
["Yeah, it's so cool.", "It's the food I hated.", 'They were of great help.']
```

Similarly, we can perform augmentation using English to French as shown below with the exact same helper method.

```
en_texts = ['This is so cool', 'I hated the food', 'They were very helpful']
aug_texts = back_translate(en_texts, source_lang="en", target_lang="fr")

print(aug_texts)
["It's so cool.", 'I hated food.', "They've been very helpful."]
```

Chained Back TranslationPermalink

You can also run back translation in a chain to get more diversity. For example,

```
English -> Spanish -> English -> French -> English
```

```
en_texts = ['This is so cool', 'I hated the food', 'They were very helpful']

aug1_texts = back_translate(en_texts, source_lang="en", target_lang="es")
aug2_texts = back_translate(aug1_texts, source_lang="en", target_lang="fr")
```

```
print(aug2_texts)
["Yeah, that's cool.", "It's the food I hated.", 'They were of great help.']
```

Available ModelsPermalink

Here are language codes for a subset of major romance language that you can use above.

Language	French	Spanish	Italian	Portuguese	Romanian	Catalan	Galician	Latin
Code	fr	es	it	pt	ro	ca	gl	la

Language	Walloon	Occitan (post 1500)	Sardinian	Aragonese	Corsican	Romansh
Code	wa	oc	sn	an	co	rm

To view all available language codes, you can run

```
target_tokenizer.supported_language_codes
```

Alternative ApplicationsPermalink

Besides data augmentation, the back translation process can also be used for text paraphrasing.

Similarly, we can also use it as an adversarial attack. Suppose we have a training dataset on which we trained an NLP model. Then, we can augment the training dataset and generate prediction from our model on augmented texts. If the predictions are different than our ground-truth labels, then we have a list of texts where our model fails. We can get good insights by analyzing those responses.

ConclusionPermalink

Thus, MarianMT is a decent free and offline alternative to Google Translate for back-translation.

ReferencesPermalink