Join our book community on Discord

https://packt.link/EarlyAccessCommunity

Transformers have advanced translations with their unique ability to capture the meaning of sequences of words in scores of languages. In this chapter, we will go through some key translation concepts and explore their scope in Google Trax, Google Translate, and Google Bard.Humans excel at transduction, transferring information from one language to another. We can easily imagine a mental representation of a sequence. For example, if somebody says *The flowers in my garden are beautiful*, we can easily visualize a garden with flowers in it. We see images of the garden, although we might never have seen that garden. We might even imagine chirping birds and the scent of flowers.A machine must learn transduction from scratch with numerical representations. Recurrent or convolutional approaches have produced exciting results but have not reached significant BLEU(Bilingual Evaluation Understudy) translation evaluation scores. Translating requires the representation of language *A* transposed into language *B*.The transformer model's self-attention innovation increases the analytic ability of machine intelligence. For example, a sequence in language *A* is adequately represented before attempting to translate it into language *B*. Self-attention brings the level of language understanding required by a machine to obtain better BLEU scores.In 2017, The seminal *Attention Is All You Need* Transformer displayed the best results for English-German and English-French translations. Since then, the scores have been improved by other transformers.In this chapter, we will go through machine translation in three steps. We will first define what machine translation is. We will then preprocess a **Workshop on Machine Translation** (**WMT**) dataset. Finally, we will see how to implement machine translations.This chapter covers the following topics:

- Defining machine translation
- Human transductions and translations
- Machine transductions and translations
- Evaluating Machine Translations
- Preprocessing a WMT dataset
- Preprocessing the raw data
- Evaluating machine translations with BLEU
- Translations with Trax
- Creating the original Transformer model
- Tokenizing a sentence
- Decoding from the Transformer
- De-tokenizing and displaying the translation
- Translation with Google Translate
- Translation with a Google Translate AJAX API Wrapper
- Implementing googletrans
- Translation with Google BARD

Our first step will be to define machine translation.

Defining machine translation

Vaswani et al. (2017) tackled one of the most difficult NLP problems when designing the Transformer. The human baseline for machine translation seems out of reach for us human-machine intelligence designers. However, this did not stop *Vaswani* et al. (2017) from publishing the Transformer's architecture and achieving state-of-the-art BLEU results.In this section, we will define machine translation. Machine translation is the process of reproducing human translation by machine transductions and outputs:

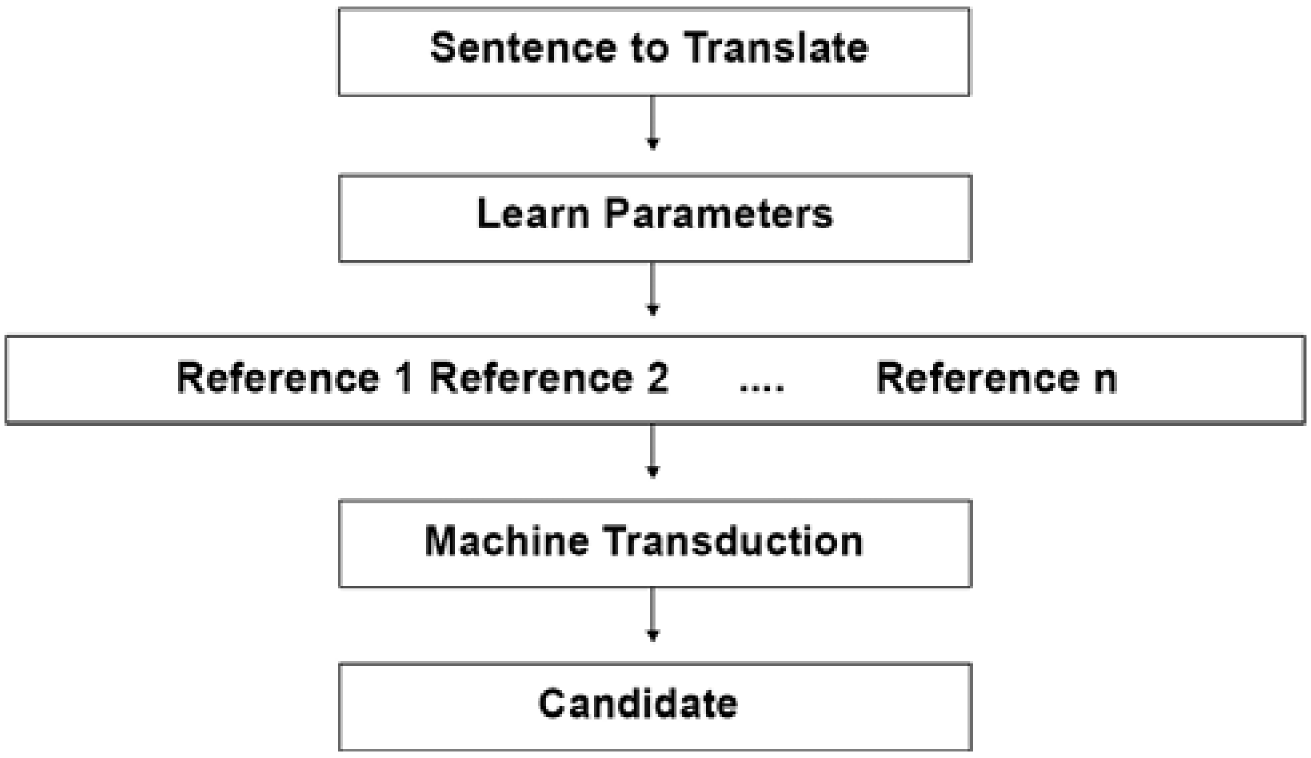


Figure 6.1: Machine translation process

The general idea in *Figure 6.1* is for the machine to do the following in a few steps:

1. Choose a sentence to translate
2. Learn how words relate to each other with hundreds of millions of parameters
3. Learn the many ways in which words refer to each other
4. Use machine transduction to transfer the learned parameters to new sequences
5. Choose a candidate translation for a word or sequence

The process always starts with a sentence to translate from a source language, *A*. The process ends with an output containing a translated sentence in language *B*. The intermediate calculations involve transductions.

Human transductions and translations

For instance, a human interpreter at the European Parliament will not translate a sentence word by word. Word-by-word translations often make no sense because they lack the proper grammatical structure and cannot produce the right translation because each word's context is ignored.Human transduction takes a sentence in language *A* and builds a cognitive *representation* of the sentence's meaning. An interpreter (oral translations) or a translator (written translations) at the European Parliament will only transform that transduction into an interpretation of that sentence in language *B*.We will name the translation done by the interpreter or translator in language *B* a *reference* sentence.You will notice several references in the *Machine translation process* described in *Figure 6.1*.A human translator will not translate sentence *A* into sentence *B* several times but only once in real life. However, more than one translator could translate sentence *A* in real life. For example, you can find several French-to-English translations of *Les Essais* by Montaigne. If you take one sentence, *A*, out of the original French version, you will thus find several versions of sentence *B* noted as references `1` to `n`.If you go to the European Parliament one day, you might notice that the interpreters only translate for a limited time of two hours, for example. Then, another interpreter takes over. No two interpreters have the same style, just like writers have different styles. For example, sentence *A* in the source language might be repeated by the same person several times in a day but be translated into several reference sentence *B* versions:*reference* ={*reference 1*, *reference 2*,...*reference n*}Machines have to find a way to think the same way as human translators.

Machine transductions and translations

The transduction process of the original Transformer architecture uses the encoder stack, the decoder stack, and all the model's parameters to represent a *reference sequence*. We will refer to that output sequence as the *reference*.Why not just say "output prediction"? The problem is that there is no single output prediction. The Transformer, like humans, will produce a result we can refer to, but that can change if we train it differently or use different transformer models!We immediately realize that the human baseline of human transduction, representations of a language sequence, is quite a challenge. However, much progress has been made.An evaluation of machine translation proves that NLP has progressed. To determine that one solution is better than another, each NLP challenger, lab, or organization must refer to the same datasets for the comparison to be valid.

Evaluating Machine Translations

Vaswani et al. (2017) present the Transformer's achievements on the WMT 2014 English-to-German translation task and the WMT (Workshop on Statistical Machine) 2014 English-to-French translation task. The Transformer achieves a state-of-the-art BLEU score. BLEU will be described in the *Evaluating machine translation with BLEU* section of this chapter.However, we must begin by preprocessing the WMT dataset we will examine.

Preprocessing a WMT dataset

The 2014 WMT contained several European language datasets. One dataset contained data from version 7 of the Europarl corpus. We will use the French-English dataset from the *European Parliament Proceedings Parallel Corpus*, 1996-2011 (https://www.statmt.org/europarl/v7/fr-en.tgz).Open WMT-translations.ipynb, which is in the chapter directory of the GitHub repository.The first step is to download the files we need:

Copy Explain

```
import urllib.request
# Define the file URL
file_url = "https://www.statmt.org/europarl/v7/fr-en.tgz"
# Define the destination file path
destination_file = "/content/fr-en.tgz"
# Download the file
urllib.request.urlretrieve(file_url, destination_file)
```

Now we will extract the files:

Copy Explain

```
import tarfile
# Extract the tar file
with tarfile.open(destination_file, 'r:gz') as tar_ref:
    tar_ref.extractall("/content/fr-en")
```

Once you have downloaded the files and have extracted them, they will be available in the following paths:

Copy Explain

```
/content/fr-en/europarl-v7.fr-en.en
/content/fr-en/europarl-v7.fr-en.fr
```

We will load, clear, and reduce the size of the corpus.Let's start the preprocessing of the two parallel files.

Preprocessing the raw data

In this section, we will preprocess `europarl-v7.fr-en.en` and `europarl-v7.fr-en.fr`.The program begins using standard Python functions and `pickle` to dump the serialized output files:

Copy Explain

```
import pickle
from pickle import dump
```

Then, we define the function to load the file into memory:

Copy Explain

```
# load doc into memory
def load_doc(filename):
        # open the file as read only
        file = open(filename, mode='rt', encoding='utf-8')
        # read all text
        text = file.read()
        # close the file
        file.close()
        return text
```

The loaded document is then split into sentences:

Copy Explain

```
# split a loaded document into sentences
def to_sentences(doc):
        return doc.strip().split('\n')
```

The shortest and the longest lengths are retrieved:

Copy Explain

```
# shortest and longest sentence lengths
def sentence_lengths(sentences):
        lengths = [len(s.split()) for s in sentences]
        return min(lengths), max(lengths)
```

The imported sentence lines must be cleaned to avoid training useless and noisy tokens. The lines are normalized, tokenized on white spaces, and converted to lowercase. The punctuation is removed from each token, non-printable characters are removed, and tokens containing numbers are excluded. The cleaned line is stored as a string.The program runs the cleaning function and returns clean appended strings:

Copy Explain

```
# clean lines
import re
import string
import unicodedata
def clean_lines(lines):
        cleaned = list()
        # prepare regex for char filtering
        re_print = re.compile('[^%s]' % re.escape(string.printable))
        # prepare translation table for removing punctuation
        table = str.maketrans('', '', string.punctuation)
        for line in lines:
                # normalize unicode characters
                line = unicodedata.normalize('NFD', line).encode('ascii', 'ignore')
                line = line.decode('UTF-8')
                # tokenize on white space
                line = line.split()
                # convert to lower case
                line = [word.lower() for word in line]
                # remove punctuation from each token
                line = [word.translate(table) for word in line]
                # remove non-printable chars from each token
                line = [re_print.sub('', w) for w in line]
                # remove tokens with numbers in them
                line = [word for word in line if word.isalpha()]
                # store as string
                cleaned.append(' '.join(line))
        return cleaned
```

We have defined the key functions we will call to prepare the datasets. The English data is loaded and cleaned first:

Copy Explain

```
# load English data
filename = 'europarl-v7.fr-en.en'
doc = load_doc(filename)
sentences = to_sentences(doc)
minlen, maxlen = sentence_lengths(sentences)
print('English data: sentences=%d, min=%d, max=%d' % (len(sentences), minlen, maxlen))
cleanf=clean_lines(sentences)
```

The dataset is now clean, and `pickle` dumps it into a serialized file named `English.pkl that we can save for future use`:

Copy Explain

```
filename = 'English.pkl'
outfile = open(filename,'wb')
pickle.dump(cleanf,outfile)
outfile.close()
print(filename," saved")
```

The output shows the key statistics and confirms that `English.pkl` is saved:

Copy Explain

```
English data: sentences=2007723, min=0, max=668
English.pkl  saved
```

We now repeat the same process with the French data and dump it into a serialized file named `French.pkl`:

Copy Explain

```
# load French data
filename = 'europarl-v7.fr-en.fr'
doc = load_doc(filename)
sentences = to_sentences(doc)
minlen, maxlen = sentence_lengths(sentences)
print('French data: sentences=%d, min=%d, max=%d' % (len(sentences), minlen, maxlen))
cleanf=clean_lines(sentences)
filename = 'French.pkl'
outfile = open(filename,'wb')
pickle.dump(cleanf,outfile)
outfile.close()
print(filename," saved")
```

The output shows the key statistics for the French dataset and confirms that `French.pkl` is saved.

Copy Explain

```
French data: sentences=2007723, min=0, max=693
French.pkl saved
```

The main preprocessing is done. However, we still need to ensure the datasets do not contain noisy and confusing tokens.

Finalizing the preprocessing of the datasets

Our process now defines the function that will load the datasets that were cleaned up in the previous section and then save them once the preprocessing is finalized:

Copy Explain

```
from pickle import load
from pickle import dump
from collections import Counter

# load a clean dataset
def load_clean_sentences(filename):
        return load(open(filename, 'rb'))

# save a list of clean sentences to file
def save_clean_sentences(sentences, filename):
        dump(sentences, open(filename, 'wb'))
        print('Saved: %s' % filename)
```

We now define a function that will create a vocabulary counter. It is essential to know how many times a word is used in the sequences we will parse. For example, if a word is only used once in a dataset containing two million lines, we will waste our energy using precious GPU resources to learn it! Let's define the counter:

Copy Explain

```
# create a frequency table for all words
def to_vocab(lines):
        vocab = Counter()
        for line in lines:
                tokens = line.split()
                vocab.update(tokens)
        return vocab
```

The vocabulary counter will detect words with a frequency that is below `min_occurrence`:

Copy Explain

```
# remove all words with a frequency below a threshold
def trim_vocab(vocab, min_occurrence):
        tokens = [k for k,c in vocab.items() if c >= min_occurrence]
        return set(tokens)
```

In this case, `min_occurrence=5`, and the words below or equal to this threshold have been removed to avoid wasting the training model's time analyzing them.We now have to deal with **Out-Of-Vocabulary (OOV)** words. OOV words can be misspelled words, abbreviations, or any word that does not fit standard vocabulary representations. We could use automatic spelling, but it would not solve all of the problems. For this example, we will simply replace OOV words with the `unk` (unknown) token:

Copy Explain

```
# mark all OOV with "unk" for all lines
def update_dataset(lines, vocab):
        new_lines = list()
        for line in lines:
                new_tokens = list()
                for token in line.split():
                        if token in vocab:
                                new_tokens.append(token)
                        else:
                                new_tokens.append('unk')
                new_line = ' '.join(new_tokens)
                new_lines.append(new_line)
        return new_lines
```

We will now run the functions for the English dataset, save the output, and then display `20` lines:

Copy Explain

```
# load English dataset
filename = 'English.pkl'
lines = load_clean_sentences(filename)
# calculate vocabulary
vocab = to_vocab(lines)
print('English Vocabulary: %d' % len(vocab))
# reduce vocabulary
vocab = trim_vocab(vocab, 5)
print('New English Vocabulary: %d' % len(vocab))
# mark out of vocabulary words
lines = update_dataset(lines, vocab)
# save updated dataset
filename = 'english_vocab.pkl'
save_clean_sentences(lines, filename)
# spot check
for i in range(20):
        print("line",i,":",lines[i])
```

The output functions first show the vocabulary compression obtained:

Copy Explain

```
English Vocabulary: 105357
New English Vocabulary: 41746
Saved: english_vocab.pkl
```

The preprocessed dataset is saved. The output function then displays `20` lines, as shown in the following excerpt:

Copy Explain

```
line 0 : resumption of the session
line 1 : i declare resumed the session of the european parliament adjourned on friday
december and i would like once again to wish you a happy new year in the hope that you
enjoyed a pleasant festive period
line 2 : although as you will have seen the dreaded millennium bug failed to
materialise still the people in a number of countries suffered a series of natural
disasters that truly were dreadful
line 3 : you have requested a debate on this subject in the course of the next few
days during this partsession
```

Let's now run the functions for the French dataset, save the output, and then display `20` lines:

Copy Explain

```
# load French dataset
filename = 'French.pkl'
lines = load_clean_sentences(filename)
# calculate vocabulary
vocab = to_vocab(lines)
print('French Vocabulary: %d' % len(vocab))
# reduce vocabulary
vocab = trim_vocab(vocab, 5)
print('New French Vocabulary: %d' % len(vocab))
# mark out of vocabulary words
lines = update_dataset(lines, vocab)
# save updated dataset
filename = 'french_vocab.pkl'
save_clean_sentences(lines, filename)
# spot check
for i in range(20):
        print("line",i,":",lines[i])
```

The output functions first show the vocabulary compression obtained:

Copy Explain

```
French Vocabulary: 141642
New French Vocabulary: 58800
Saved: french_vocab.pkl
```

The preprocessed dataset is saved. The output function then displays `20` lines, as shown in the following excerpt:

Copy Explain

```
line 0 : reprise de la session
line 1 : je declare reprise la session du parlement europeen qui avait ete interrompue
le vendredi decembre dernier et je vous renouvelle tous mes vux en esperant que vous
avez passe de bonnes vacances
line 2 : comme vous avez pu le constater le grand bogue de lan ne sest pas produit en
revanche les citoyens dun certain nombre de nos pays ont ete victimes de catastrophes
naturelles qui ont vraiment ete terribles
line 3 : vous avez souhaite un debat a ce sujet dans les prochains jours au cours de
cette periode de session
```

This section showed how raw data must be processed before training. The datasets are now ready to be plugged into a transformer to be trained.Each line of the French dataset is the *sentence* to translate. Each line of the English dataset is the *reference* for a machine translation model. The machine translation model must produce an *English candidate translation* that matches the *reference*.BLEU provides a method to evaluate `candidate` translations produced by machine translation models.

Evaluating machine translations with BLEU

Papineni et al. (2002) devised an efficient way to evaluate a human translation. The human baseline was difficult to define. However, they realized we could obtain efficient results if we compared human translation with machine translation, word for word.*Papineni* et al. (2002) named their method the **Bilingual Evaluation Understudy Score (BLEU)**.In this section, we will use the **Natural Language Toolkit (NLTK)** to implement BLEU:http://www.nltk.org/api/nltk.translate.html#nltk.translate.bleu_score.sentence_bleuWe will begin with geometric evaluations.

Geometric evaluations

The BLEU method compares the parts of a candidate sentence to a reference sentence or several reference sentences.We will use the `nltk` library in Python with text processing libraries.The program imports the `nltk` library:

Copy Explain

```
from nltk.translate.bleu_score import sentence_bleu
from nltk.translate.bleu_score import SmoothingFunction
```

The notebook then compares a candidate translation produced by the machine translation model and the actual translation(s) references in the dataset. Remember that a sentence could have been repeated several times and translated by different translators in different ways, making it challenging to find efficient evaluation strategies.The program can evaluate one or more references:

Copy Explain

```
#Example 1
reference = [['the', 'cat', 'likes', 'milk'], ['cat', 'likes' 'milk']]
candidate = ['the', 'cat', 'likes', 'milk']
score = sentence_bleu(reference, candidate)
print('Example 1', score)
#Example 2
reference = [['the', 'cat', 'likes', 'milk']]
candidate = ['the', 'cat', 'likes', 'milk']
score = sentence_bleu(reference, candidate)
print('Example 2', score)
```

The score for both examples is `1`:

Copy Explain

```
Example 1 1.0
Example 2 1.0
```

A straightforward evaluation *P* of the candidate *C*, the reference *R*, and the number of correct tokens found in *C (N)* can be represented as a geometric function:This geometric approach is rigid if you are looking for a 3-gram overlap, for example:

Copy Explain

```
#Example 3
reference = [['the', 'cat', 'likes', 'milk']]
candidate = ['the', 'cat', 'enjoys','milk']
score = sentence_bleu(reference, candidate)
print('Example 3', score)
```

The output is severe if you are looking for 3-gram overlaps:

Copy Explain

```
Warning (from warnings module):
  File
Example 3 1.0547686614863434e-154
/usr/local/lib/python3.10/dist-packages/nltk/translate/bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 3-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
```

The hyperparameters can be changed, but the approach remains rigid even if we sometimes obtain acceptable evaluations.The warning in the code above is a good one that announces the next section.The messages may vary with each version of the program and each run since this is a stochastic process.*Papineni* et al. (2002) came up with a modified unigram approach. The idea was to count the word occurrences in the reference sentence and ensure the word was not over-evaluated in the candidate sentence.Consider the following example explained by *Papineni* et al. (2002):

Copy Explain

```
Reference 1: The cat is on the mat.
Reference 2: There is a cat on the mat.
```

Now consider the following candidate sequence:

Copy Explain

```
Candidate: the the the the the the the
```

We now look for the number of words in the candidate sentence (the `7` occurrences of the same word "the") present in the `Reference 1` sentence (`2` occurrences of the word "the").A standard unigram precision would be `7/7`.The modified unigram precision is `2/7`.Note that the BLEU function output warning agrees and suggests using smoothing.Let's add smoothing techniques to the BLEU toolkit.

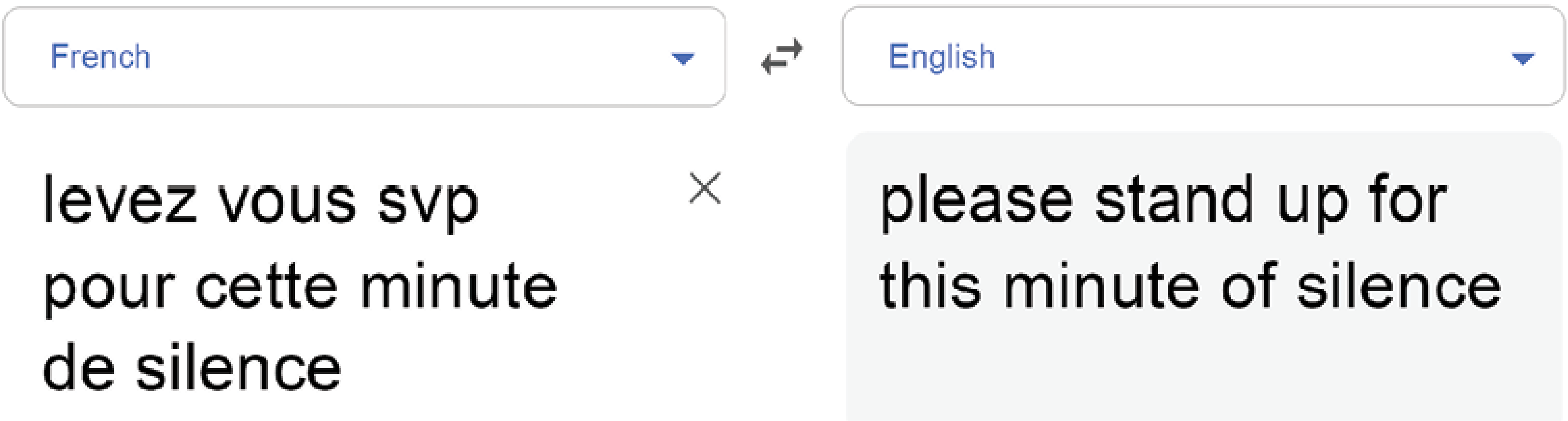
Applying a smoothing technique

Chen and *Cherry* (2014) introduced a smoothing technique that improves the geometric evaluation approach of standard BLEU techniques.Label smoothing is a very efficient method that improves the performance of a transformer model during the training phase. It has a negative impact on perplexity. However, it forces the model to be more uncertain. In turn, this has a positive effect on accuracy.For example, suppose we have to predict what the masked word is in the following sequence:`The cat [mask] milk`.Imagine the output comes out as a softmax vector:

Copy Explain

```
candidate_words=[drinks, likes, enjoys, appreciates]
candidate_softmax=[0.7, 0.1, 0.1,0.1]
candidate_one_hot=[1,0,0,0]
```

This would be a brutal approach. Label smoothing can make the system more open-minded by introducing epsilon = ε.The number of elements of `candidate_softmax` is *k=4*.For label smoothing, we can set $1\text{-}\varepsilon$ to `0.25`, for example.One of the several approaches to label smoothing can be a straightforward function.First, reduce the value of `candidate_one_hot` by $0+\frac{\varepsilon}{k\text{-}1}$.Increase the `0` values by



.We obtain the following result if we apply this approach:`candidate_smoothed=[0.75,0.083,0.083,0.083]`, making the output open to future transformations and changes.The Transformer uses variants of label smoothing.A variant of BLEU is `chencherry` smoothing.

Chencherry smoothing

Chen and *Cherry* (2014) introduced an interesting way of smoothing candidate evaluations by adding to otherwise `0` values. In this section, we will first evaluate without smoothing and then implement a `chencherry` (Boxing Chen + Colin Cherry) smoothing function.Let's first evaluate a French-English example without smoothing:

Copy Explain

```
#Example 4
reference = [['je','vous','invite', 'a', 'vous', 'lever','pour', 'cette', 'minute',
'de', 'silence']]
candidate = ['levez','vous','svp','pour', 'cette', 'minute', 'de', 'silence']
score = sentence_bleu(reference, candidate)
print("without soothing score", score)
```

Although a human could accept the candidate, the output score is weak:

Copy Explain

```
without smoothing score 0.37188004246466494
```

Now, let's add some openminded smoothing to the evaluation:

Copy Explain

```
chencherry = SmoothingFunction()
r1=list('je vous invite a vous lever pour cette minute de silence')
candidate=list('levez vous svp pour cette minute de silence')

#sentence_bleu([reference1, reference2, reference3],
hypothesis2,smoothing_function=chencherry.method1)
print("with smoothing score",sentence_bleu([r1],
candidate,smoothing_function=chencherry.method1))
```

The score is just an example of how to use smoothing and will improve continually as the functions evolve:

Copy Explain

```
with smoothing score 0.6194291765462159
```

We have now seen how a dataset is preprocessed and how BLEU evaluates machine translations.

Translations with Google Trax

Google Brain developed **Tensor2Tensor (T2T)** to make deep learning development easier. T2T is an extension of TensorFlow and contains a library of deep learning models that contains many transformer examples.Although T2T was a good start, Google Brain then produced Trax, an end-to-end deep learning library. Trax contains a transformer model that can be applied to translations. The Google Brain team presently maintains Trax.This section will focus on the minimum functions to initialize the English-German problem described by *Vaswani* et al. (2017) to illustrate the Transformer's performance.We will use preprocessed English and German datasets to show that the Transformer architecture is language-agnostic.Open `Trax_Google_Translate.ipynb`.We will begin by installing the modules we need.

Installing Trax

Google Brain has made Trax easy to install and run. We will import the basics along with Trax, which can be installed in one line:

Copy Explain

```
import os
import numpy as np
!pip install -q -U trax
import trax
```

Yes, it's that simple!Now, let's create our transformer model.

Creating the original Transformer model

We will create the original Transformer model as described in *Chapter 2, Getting Started with the Architecture of the Transformer Model*.Our Trax function will retrieve a pretrained model configuration in a few lines of code:

Copy Explain

```
# Pre-trained model config in gs://trax-ml/models/translation/ende_wmt32k.gin
model = trax.models.Transformer(
    input_vocab_size=33300,
    d_model=512, d_ff=2048,
    n_heads=8, n_encoder_layers=6, n_decoder_layers=6,
    max_len=2048, mode='predict')
```

The model is the Transformer with an encoder and decoder stack. Each stack contains `6` layers and `8` heads. `d_model=512`, as in the architecture of the original Transformer.We can look into the architecture of the model with the following code:

Copy Explain

```
from pprint import pprint
pprint(vars(model))
```

The output provides an interesting view of the model's architecture:

Copy Explain

```
…
Serial_in2_out2[
    Embedding_33300_512
    Dropout
    PositionalEncoding
    Serial_in2_out2[
      Branch_in2_out3[
        None
        Serial_in2_out2[
          LayerNorm
          Serial_in2_out2[
            _in2_out2
            Serial_in2_out2[
              Select[0,0,0]_out3
              Serial_in4_out2[
                _in4_out4
                Serial_in4_out2[
                  Parallel_in3_out3[
                    Dense_512
                    Dense_512
                    Dense_512
   …
```

Take some time to go through the model. Let's take a critical concept that opens the toolkit to many flexible architectures.

Combinator `Combinator` is a type of layer that contains other layers. It is an abstract class. We use subclasses of this class to combine layers for different functions such `Serial`, `Parallel`, `Branch`, `Residual`, `Select`. These subclasses are combinators used as building blocks of the neural network.

`Serial` applies layers one after the other in a sequence.

`Parallel` applies several layers to the same input in parallel and a tuple of their outputs.

`Branch` applies several layers in parallel to an input. Then, concatenates the outputs.

`Residual` adds the output of sequences of layers (or a single layer) to the input.

`Select` selects elements from a tuple of inputs

The `in2_out2` term means that are 2 inputs and 2 outputs are produced.For more information, please refer to the Google Trax documentation:https://trax-ml.readthedocs.io/en/latest/notebooks/layers_intro.htmlThe Transformer requires the pretrained weights to run.

Initializing the model using pretrained weights

The pretrained weights contain the intelligence of the Transformer. The weights constitute the Transformer's representation of language. The weights can be expressed as a number of parameters that will produce some form of *machine intelligence IQ*.Let's give life to the model by initializing the weights:

Copy Explain

```
model.init_from_file('gs://trax-ml/models/translation/ende_wmt32k.pkl.gz',
                     weights_only=True)
```

The code displays the weights, which helps see how the model is built under the hood:

Copy Explain

```
…
1.95107 , 2.3058772 , 1.9680263 , 1.5733448 , 1.7866848 , 2.192197 , 2.228089 ,
1.7842566 , 2.2654603 , 2.0060909 , 1.2600263 , 1.7945113 , 1.1802608 ,
…
```

The machine configuration and its *intelligence* are now ready to run. Let's tokenize a sentence.

Tokenizing a sentence

Our machine translator is ready to tokenize a sentence. The notebook uses the vocabulary preprocessed by `Trax`. The preprocessing method is similar to the one described in this chapter's *Preprocessing a WMT dataset* section.The sentence will now be tokenized:

Copy Explain

```
#Tokenizing a sentence
sentence = 'I am only a machine but I have machine intelligence.'
tokenized = list(trax.data.tokenize(iter([sentence]), # Operates on streams.
                                    vocab_dir='gs://trax-ml/vocabs/',
                                    vocab_file='ende_32k.subword'))[0]
```

We are now ready to decode the sentence and produce a translation.

Decoding from the Transformer

The Transformer encodes the sentence in English and will decode it in German. The model and its weights constitute its set of abilities.`Trax` has made the decoding function intuitive to use:

Copy Explain

```
tokenized = tokenized[None, :]  # Add batch dimension.
tokenized_translation = trax.supervised.decoding.autoregressive_sample(
    model, tokenized, temperature=0.0)  # Higher temperature: more diverse results.
```

Note that higher temperatures will produce different results, just as with human translators, as explained in this chapter's *Defining machine translation* section.Finally, the program will de-tokenize and display the translation.

De-tokenizing and displaying the translation

Google Brain has produced a mainstream, disruptive, and intuitive implementation of the Transformer with `Trax`.The program now de-tokenizes and displays the translation in a few lines:

Copy Explain

```
tokenized_translation = tokenized_translation[0][:-1]  # Remove batch and EOS.
translation = trax.data.detokenize(tokenized_translation,
                                    vocab_dir='gs://trax-ml/vocabs/',
                                    vocab_file='ende_32k.subword')
print("The sentence:",sentence)
print("The translation:",translation)
```

The output is quite impressive:

Copy Explain

```
The sentence: I am only a machine but I have machine intelligence.
The translation: Ich bin nur eine Maschine, aber ich habe Maschinenübersicht.
```

The Transformer translated `machine intelligence` into `Maschinenübersicht`.If we deconstruct `Maschinenübersicht` into `Maschin (machine)` + `übersicht (intelligence)`, we can see that:

> über literally means "over"
>
> sicht means "sight" or "view"

The Transformer tells us that although it is a machine, it has vision. Machine intelligence is growing through Transformers, but it is not human intelligence. Machines learn languages with an intelligence of their own.Google Trax provides a toolkit to build and run models.Now, let's explore Google Translate.

Translation with Google Translate

Google Translate, https://translate.google.com/, provides a ready-to-use official interface for translations. Google also possesses transformer technology in its translation algorithms. However, an AI specialist may not be required at all.If we enter the sentence analyzed in the previous section in Google Translate, `Levez-vous svp pour cette minute de silence`, we obtain an English translation in real-time:

Figure 6.2: Google Translate

The translation is correct.Does the AI industry still require AI specialists for translation tasks or simply a web interface developer?Google provides every service required for translations on their Google Translate platform: https://cloud.google.com/translate:

1. A translation API: A web developer can create an interface for a customer
2. A media translation API that can translate your streaming content
3. An AutoML translation service that will train a custom model for a specific domain

A Google translate project requires a web developer for the interfaces, a **Subject Matter Expert (SME)**, and perhaps a linguist. However, being an AI specialist is not a prerequisite.LLMs are driving AI towards AI as a service. So why bother studying AI development with transformers? There are two important reasons to become an AI Profesional:

1. In real life, AI projects often run into unexpected problems. For example, Google Translate might not fit a specific need no matter how much goodwill was put into the project. In that case, APIs will come in handy!
2. You must be an AI developer to use Google Trax for AI or Google Cloud AI APIs!

You never know! LLMs are rolling out and connecting everything to everything; some AI projects might run smoothly, and some will require AI expertise to solve complex problems. Google Cloud AI charges the usage of its APIs. For educational purposes, we will implement a free Google Translate AJAX API wrapper.

Translation with a Google Translate AJAX API Wrapper

In this section, we will implement `googletrans`. The `googletrans` library is an unofficial third-party wrapper around the Google Translate AJAIX API. `googletrans` is an excellent educational tool and can fit limited usage. However, it is recommended to use the official Google Cloud AI Translate API for projects requiring many requests or commercial usage, among other restrictions.`googletrans` provides the same translations as Google Translate, although it might encounter problems if Google changes the access to its API.AJAX (Asynchronous JavaScript and XML) means that the interface doesn't need to wait for AJAX to complete its request, which can be written in JavaScript with XML (eXtensible Markup Language) or JSON (JavaScript Object Notation).`googletrans` thus inherits Google Translate's sequence-to-sequence (seq2seq) technology, which leverages neural networks, including transformers and other models.Let's implement `googletrans`.

Implementing googletrans

Copy Explain

```
The second part, Running a Google Translate AJAX API Wrapper, of the chapter's
Trax_Google_Translate.ipynb begins by installing googletrans:
!pip install googletrans==4.0.0-rc1 -q
```

For more information: https://pypi.org/project/googletrans/We can now create a translation object and call the methods of the library.

Creating an instance of the Translator class

An instance of the `Translator` class takes one line of code:

Copy Explain

```
from googletrans import Translator
translator = Translator()
The translator object is now created, and we can call the methods of the Translator
class.
```

Translating a text from English to French

Copy Explain

```
The following cell calls the translate method:
import googletrans
translator = googletrans.Translator()
# Get the translation of "Google Trax can translate to French"
translation = translator.translate("Google Trax can translate to French", src="en",
dest="fr")
# Print the translation
print(translation.text)
```

The output is the correct translation:

Copy Explain

```
Google Trax peut se traduire en français
```

Detecting the language of a text

Copy Explain

```
We can also detect a language with the detect method:
lang=translator.detect('이 문장은 한글로 쓰여졌습니다.')
print(lang)
```

The method will display the language of the input sequence:

Copy Explain

```
Detected(lang=ko, confidence=None)
```

In this case, the language detected is Korean.Note: The confidence level is not always displayed.

Translating in multiple languages

We can call the `translate` method for several languages:

Copy Explain

```
from googletrans import Translator
translator = Translator()
languages = ['en', 'es', 'fr']
for language in languages:
    print(translator.translate('안녕하세요', dest=language).text)
```

The output will provide English, Spanish, and French translation in this example of “hello” in Korean:

Copy Explain

```
hello
Hola
Bonjour
```

For more features: https://py-googletrans.readthedocs.io/en/latest/We have explored Google Translate. Now, we will run translations with Generative AI as an end-user.

Translation with Google BARD

How far can we go with Google BARD for translations? We reviewed some of Google BARD's NLP abilities in *Chapter 3, Emergent vs Downstream Tasks: the Unseen Depths of Transformers*. Now, we will create a dialog with Google BARD to explore its potential and limitations.Go to https://bard.google.com/ to start a dialog.Needless to say that BARD has been trained with large datasets in scores of languages.We will begin with BARD's potential and then search for its limitations.

Copy Explain

```
My prompt begins with Denis:
```

Google BARD's response starts with: PALM2

BARD's potential

Copy Explain

```
Denis: Can you translate a sentence from English to French?
```

Yes, I can translate a sentence from English to French. For example, if you say "I am a large language model," the translation would be "Je suis un grand modèle linguistique."

Copy Explain

```
Here are some other examples of English-to-French translations:
Hello = Bonjour
Goodbye = Au revoir
Thank you = Merci
You're welcome = Je vous en prie
Please = S'il vous plait
Excuse me = Pardon
How are you? = Comment allez-vous?
I'm fine = Je vais bien
What's your name? = Comment vous appelez-vous?
My name is Bard = Je m'appelle Bard
I can also translate longer texts, such as paragraphs or articles. Just give me the
text and I will translate it for you.
```

Google BARD is very confident!

Copy Explain

```
Denis:
```

Sure, here is the translation of the sentence "Tourists who go to Europe like to visit all the countries they can to see museums, monuments and shows" in French:Les touristes qui vont en Europe aiment visiter tous les pays qu'ils peuvent pour voir des musées, des monuments et des spectacles.That's fine, but how do we know if the translation is correct if we don't speak French?Let's ask BARD to perform a back-translation task:

Copy Explain

```
Denis: Translate this sentence from French into English:"Translate this sentence back
into English:Les touristes qui vont en Europe aiment visiter tous les pays qu'ils
peuvent pour voir des musées, des monuments et des spectacles."
```

The response is back translation is correct:

Copy Explain

```
Tourists who go to Europe like to visit as many countries as they can to see museums,
monuments, and shows
```

Back translations can be a way to check if the translation from one language to another suits your needs. It requires additional implementation and development but can provide higher quality applications.

Note: The fact that BARD cannot translate from one particular language to another is not a limitation. When the translation is incorrect, it will most probably improved in the future. It's just the present scope and limits of the model.

This section shows that the models are progressing. LLMs represent considerable progress. But there is still a long way to go! We AI specialists still have much work to do, which is good news!

Summary

In this chapter, we went through some of the essential aspects of translations with transformers.We started by defining machine translation. Human translation sets a high baseline for machines to reach. We saw that English-French and English-German translations imply numerous problems to solve. The Transformer tackled these problems and set state-of-the-art BLEU records to beat.We then preprocessed a WMT French-English dataset from the European Parliament that required cleaning. We had to transform the datasets into lines and clean the data up. Once that was done, we reduced the dataset's size by suppressing words that occurred below a frequency threshold.Machine translation NLP models require identical evaluation methods. For example, training a model on a WMT dataset requires BLEU evaluations. We saw that geometric assessments are a good basis for scoring translations, but even modified BLEU has its limits. We thus added a smoothing technique to enhance BLEU.We implemented an English-to-German translation transformer with Trax, Google Brain's end-to-end deep learning library.We saw that Google Translate provides a standard translation API, a media streaming API, and custom AutoML model training services. Implementing Google Translate APIs may require no AI development if the project rolls out smoothly. If not, we will have to get our hands dirty, like in the old days, which we did with the *googletrans* library with its Google Translate AJAX API wrapper!Finally, we had a conversation with Google BARD's Generative AI interface to explore the potential and limitations of LLMs applied to translations.We have now covered the main building blocks to construct transformers, their architecture and viewed some of the NLP tasks they can perform.The next step is to fine-tune a model when we wish to focus on a specific NLP task. In *Chapter 5, Diving into Fine-Tuning through BERT*, we will fine-tune a transformer model.

Questions

1. Machine translation has now exceeded human baselines. (True/False)
2. Machine translation requires large datasets. (True/False)
3. There is no need to compare transformer models using the same datasets. (True/False)
4. BLEU is the French word for *blue* and is the acronym of an NLP metric (True/False)
5. Smoothing techniques enhance BERT. (True/False)
6. German-English is the same as English-German for machine translation. (True/False)
7. The original Transformer multi-head attention sub-layer has 2 heads. (True/False)
8. The original Transformer encoder has 6 layers. (True/False)

9. The original Transformer encoder has 6 layers but only 2 decoder layers. (True/False)
10. You can train transformers without decoders. (True/False)

References

English-German BLEU scores with reference papers and code: https://paperswithcode.com/sota/machine-translation-on-wmt2014-english-germanThe 2014 **Workshop on Machine Translation** (**WMT**): https://www.statmt.org/wmt14/translation-task.html*European Parliament Proceedings Parallel Corpus 1996-2011*, parallel corpus French-English: https://www.statmt.org/europarl/v7/fr-en.tgz*Jason Brownlee*, *Ph.D.*, *How to Prepare a French-to-English Dataset for Machine Translation*: https://machinelearningmastery.com/prepare-french-english-dataset-machine-translation/*Jason Brownlee*, *Ph.D.*, *A Gentle Introduction to Calculating the BLEU Score for Text in Python*: https://machinelearningmastery.com/calculate-bleu-score-for-text-python/*Boxing Chen* and *Colin Cherry*, 2014, *A Systematic Comparison of Smoothing Techniques for Sentence-Level BLEU*: http://acl2014.org/acl2014/W14-33/pdf/W14-3346.pdfTrax repository: https://github.com/google/traxTrax tutorial: https://trax-ml.readthedocs.io/en/latest/

Further Reading

Kishore Papineni, *Salim Roukos*, *Todd Ward*, and *Wei-Jing Zhu*, *2002*, *BLEU: a Method for Automatic Evaluation of Machine Translation*: https://aclanthology.org/P02-1040.pdf*Ashish Vaswani*, *Noam Shazeer*, *Niki Parmar*, *Jakob Uszkoreit*, *Llion Jones*, *Aidan N. Gomez*, *Lukasz Kaiser*, and *Illia Polosukhin*, 2017, *Attention Is All You Need*: https://arxiv.org/abs/1706.03762

Previous Chapter

Next Chapter