Using Deep Learning for Automatic Translation

Martin Rupp

SCIENTIFIC AND COMPUTER DEVELOPMENT (SCD)

In this article, we will learn how to use Deep Learning to create an automatic translation system. For this, we will provide a **step-by-step tutorial** to help you understand and build a Neuronal Machine Translation.

An Overview of Machine Learning and Machine Translation

Before we explain concretely how to use Deep Learning (DL), we aim to provide a quick background to the reader in the area of Machine translation. We assume that the reader has a basic knowledge of machine learning, training, supervised learning, and neural networks and understands the concepts of artificial neurons, layers, and back-propagation.

Quick facts about the history of Machine Translation

The concept of Machine Translation (MT), or the ability to translate automatically - via a machine - natural languages into each other dates at least from 1949 when Warren Weaver stated the main principles of MT. At first, MT was done using expert rules (RBMT), requiring a lot of work from human translators. Then, in the late 70's, Statistical Machine Translation (SMT) appeared and started to develop itself, especially under the influence of the Candide project funded by IBM.

SMT was based on computing the most probable relationship between pairs of words and sentences taken from a text corpora (in the original language and the targeted language).

SMT ruled the area of MT until 2000 when the application of Neural Networks to MT, Neural Machine Translation (NMT), was proposed as an alternative to SMT.

While NMT did not succeed well at the beginning, it made impressive progress through the years and this is only with the recent development of AI processing power (GPU cards, etc…) that NMT is getting superior results to SMT.

With the ongoing research on Deep Learning and Long Short-term Memory designs (LSTM), NMT is getting more and more 'insane' results and it's only a matter of time before NMT will replace SMT in all the actual commercial automatic translation software.

Why Deep Learning is good at doing MT

Deep Learning aims at the creation of an artificial brain so everything that a human brain can do can be theoretically performed by a Deep Learning system. Besides, LSTM, a Deep Learning technique - a Recurrent Neural Network (RNN) to be more precise - has unprecedented records of recalling and detecting temporal patterns. This is of interest when considering a sentence from a natural language as a conditional time series of words or in fact, equivalently, considering a sentence as the result of a Markov process.

The functioning of an LSTM network applied to MT

Some info about SMT

Let's now focus on how our LSTM will work in the context of MT. As a start, we must briefly describe the main principles of SMT and how MT works.

As a general rule, one must use a base named a *parallel corpus linguae*. This is, in essence, a 'super-dictionary', usually created from sources gathered from professional translators or senior students , where pairs of words or sentences are associated together.

SMT and NMT are both using parallel corpus linguae. SMT will partition the input sentence into groups of words and then use probabilities to find the most probable 'matching' combination.

An SMT such as moses for example, will create a **translation model** from the training data and apply that translation model to any input, providing the sentence in the target language which has the highest score in terms of conditional probability.

Parallel corpus linguae can be found on the internet for a lot of natural languages. For example, many English/Dutch parallel corpus linguae can be downloaded from **the Tatoeba Project**. Other sources exist, such as the Linguee website.

First of all, we have learned how to write a business plan during the past half year. refitech.nl	Allereerst hebben we afgelopen half jaar geleerd hoe een businessplan te schrijven. refitech.nl
Our course members often have specific learning goals and therefore need the professional terminology to match: in those cases we write a lesson plan for them. reginacoeli.com	Onze cursisten hebben vaak specifieke leerdoelen en dus een specifieke vaktaal nodig: in die gevallen schrijven we een lesplan voor ze. reginacoeli.nl
On that basis, the initiative recently received the green light to write a business plan and to identify the first sectors that the initiative could target. mvoplatform.nl	Op grond daarvan is recentelijk groen licht gegeven voor het schrijven van een businessplan, en het identificeren van de eerste sectoren waar het initiatief zich op kan richten. mvoplatform.nl
On the basis of the market research we can write a concise business plan or support you in writing it. bolta-state.nl	Op basis van het marktonderzoek kunnen we een beknopt business plan maken of we ondersteunen je in het maken hiervan. bolta-state.nl
These provide benchmarks for responsible business conduct, but we need to work to ensure that implementation is more rigorous and consistent. eur-lex.europa.eu	Deze verschaffen benchmarks voor verantwoord ondernemingsgedrag, maar moeten strenger en consequenter worden toegepast. eur-lex.europa.eu
Apart from the need for a one-off write-down on our land positions and associated plan development costs, we recorded a net positive result in the year under review and are in a good position going forward. tbibouw.nl	We boekten per saldo, afgezien van een noodzakelijke eenmalige afboeking op grondposities en daarmee samenhangende planontwikkelingskosten, een positief resultaat en staan goed opgelijnd. tbibouw.nl
Subsequently, we write a yearly plan containing the objectives, strategies and actions to be undertaken. fres.nl	Vervolgens schrijven we een jaarplan met de doelstellingen, strategieën en te nemen acties. fres.nl
After conversations with the World Bank, FRES decided to write a business plan for six solar power stations; Four new mini-grids [...] fres.nl	Na gesprekken met de Wereldbank besloot FRES om een businessplan voor zes zonnecentrales te schrijven; vier nieuwe minigrids gevoed door [...] fres.nl
These principles require that we only collect or receive the minimum	Deze principes vereisen dat we alleen de minimale persoonlijke

Illustration: Parallel corpus linguae from the Linguee.com website.

The Tatoeba Project provides tab-delimited bilingual sequence pairs for various languages. For instance, the English/Dutch parallel corpus contains around 50,000 lines of translated pairs.

```
8870  I got my son to cook supper.     Ik liet mijn zoon het avondeten klaarma
8871  I got up earlier than usual.     Ik ben eerder opgestaan dan normaal.
8872  I had a toothache yesterday.     Gisteren had ik tandpijn.   CC-BY 2.0 (
8873  I had no choice but to stay.     Ik had geen andere keuze dan te blijver
8874  I had to carry Tom upstairs.     Ik moest Tom naar boven dragen. CC-BY 2
8875  I had to go there yesterday.     Ik moest er gisteren heen gaan. CC-BY 2
8876  I hate fluorescent lighting.     Ik haat tl-verlichting. CC-BY 2.0 (Frar
```

Illustration: Parallel corpus linguae (English/Dutch) from the Tatoeba project *website.*

Usually, SMT will create a **language model** for the (non-parallel) corpus of the input language.

The translation model and the language model are then used, together with possibly a **lexicon model** and an **alignment model** to compute a series of probabilities, generally using Bayesian rules and a maximal likelihood (MLE) estimator. The MLE represents a score and the MT will pick up the sentence in the targeted language that 'matches' the more the input sentence, e.g. with the higher MLE.

The principle of SMT is, roughly, that *"if such an English sentence with such words (potentially closer or identical to some of the input words) was associated to such a Dutch sentence with* *such words* **and** *if such another English sentence with such words (potentially closer or identical to some of the input words) was associated to such a Dutch sentence with such words* **and** …" then there is a (conditional) probability *p* that this input sentence is translated by this output sentence. By computing all these conditional probabilities, we can use estimators such as the MLE to score candidates for the translated sentence.

The main principles of NMT

Here are the main tools and concepts that we need to clarify, so as to be able to build our automatic translation machine which will translate, as you may have guessed, **from English to Dutch**.

- RNNs and LSTM
- Encoders and Decoders
- Gated Recurrent Unit (GRU)
- The attention mechanism
- Embeddings
- Word2Vec, ELMO and BERT

RNNs and LSTM

First, let's look at the structure of a RNN. We represented a basic RNN below. As you can see it combines layers: $layer_0$, $layer_1$,... $layer_k$, $layer_{k+1}$,... which form a directed sequence.

The input vector $X = (X_0,..., X_n)$ is transformed into an output vector $Y = (Y_0,..., Y_n)$

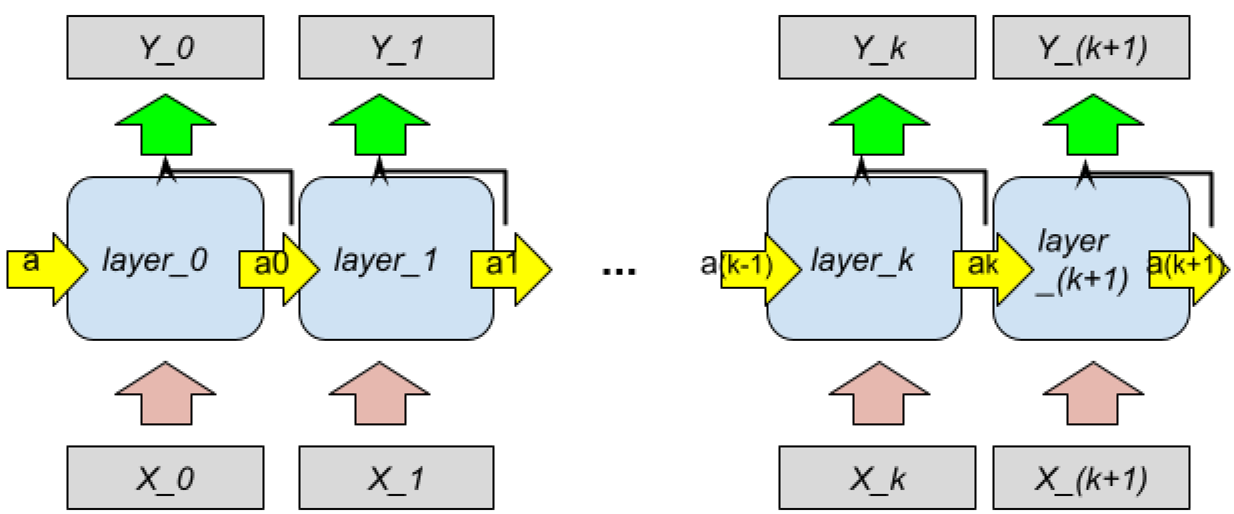


Illustration 1: a simple RNN

At this stage it's not exactly clear what is happening in a RNN. It is important to understand that each layers produces an activation a_t (just like any neural network layer) but it also *directly* produces an output Y_t which is a function of a_t.

The equation of the (ordinary layer of a) simple RNN that we have presented is the following:

$$a_t = F(\alpha.a_{t-1} + \beta.x_t + \gamma)$$

And:

$$Y_t = G(\delta.a_t + \epsilon).$$

Here F and G are two activation functions, $\alpha, \beta, \delta, \epsilon$ are variables that depend on the layer coefficients.

The RNN that we have represented is called **many-to-many** and that's because there are 'many' inputs and 'many' outputs. That's typically what we need to do in a machine translation. A unique-to-many outputs is named a **one-to-many RNN** and a many inputs to a unique output is named a **many-to-one RNN**.

RNN are not only useful in the domain of MT. They are also successfully applied to Speech recognition, music generation, feelings recognition and much more.

In the case of MT, we need to proceed with a slightly different type of RNN which we represent there:

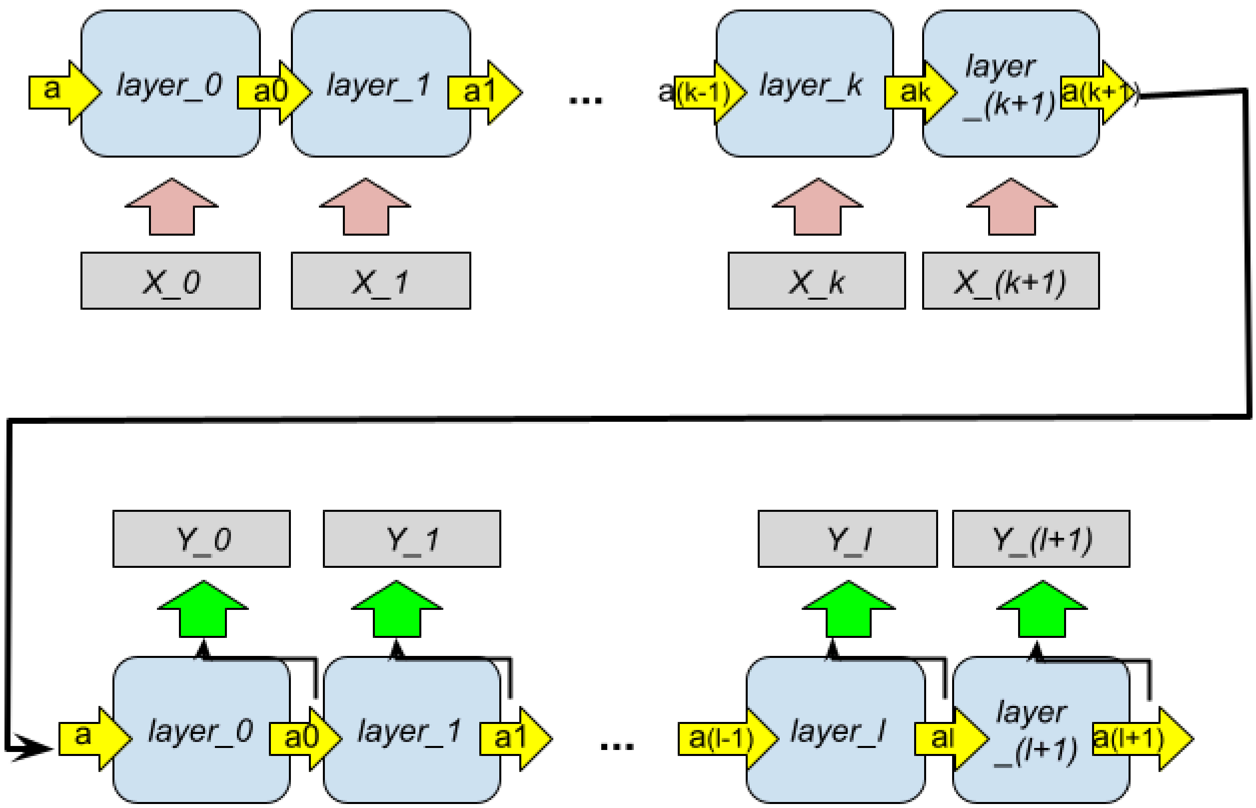
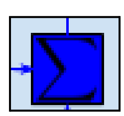
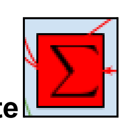


Illustration 2: a many-to-many RNN suited for NMT

This is the type of RNN that we shall be using in what will follow. The *k* components of the input vector will be the words of the sentence in the English language and the l components of the output vector are the words of the translated sentence in the Dutch language.

LSTM is a refinement of RNNs and so far, they are among the more performant RNN designs. In such a design, several or all layers are replaced by *LSTM cells*. These cells have a different design than the 'ordinary' RNN layers.

Here we present the comparison between the ordinary layer and the LSTM cell.
It's not exactly straightforward to tell why the LSTM cell is much better than the 'ordinary' RNN layer...except that it is slightly more complex.

An LSTM cell has two gates: an **input gate**  and a **forget gate** . Here the 'Sigma' symbol represents a linear combination[1] with the inputs (+ a constant).
It also transfers *deep learning* hidden states[2] (h.c).

[1] Note that all operations are vector operations
[2] Warning: this is the 'deep learning version' of the term 'hidden state'

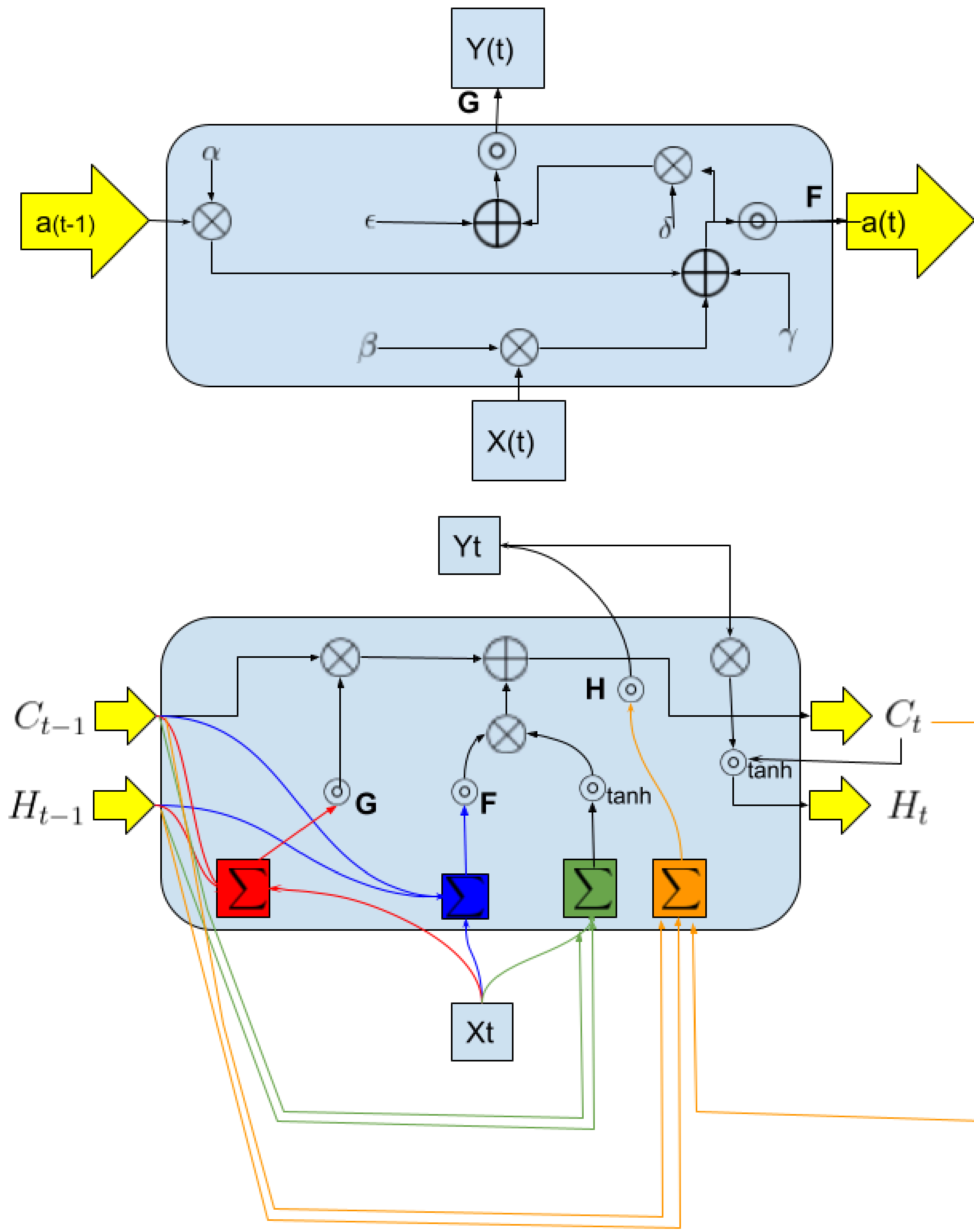


Illustration 3: comparison between a standard RNN layer and an LSTM cell

The model will mimic the Human concept of *forgetting*. For example, forgetting non-essential information. It is fully recurrent since it will re-input the previous states as well.

It would be too long to explain why the LSTM is good at doing the work and we wanted here to give just some hints to a curious reader and also to demonstrate how neural networks are in fact 'abstract' logical circuits which are more *designed* by A.I engineers rather than coded. This is also how analog computers (nondigital) are programmed.

Encoders and Decoders

We already illustrated the concept of encoder-decoder as the RNN suited for machine translation (illustration 2). In fact, two RNNs. One is the encoder, it encodes a sequence of words into a fixed-length vector and the other is a decoder, it does the reverse operation.
The RNN in illustration 2 is also called a *sequence-to-sequence RNN*.

Gated Recurrent Unit (GRU)

This is just an LSTM cell with fewer features. It can perform better than LSTM in special areas. It can be used to simplify some designs and will generally perform faster than LSTM. We won't use GRU here but we mention their existence to the curious reader.

The 'attention' mechanism

The 'attention' mechanism is a key concept in NMT and was introduced relatively recently. However it's very simple, it only consists in giving more 'weight' (importance) to one or more words contained in the sentence to translate. This simple mechanism allows now to solve many problems encountered before by NMT.

Embeddings

Embeddings are a kind of multi-dimensional representation of a word to provide statistical information about it and link it with others 'base words' which have a closer meaning or may have a close relationship with it. For example, the word *lynx* may be embedded as (cat, animal,wild) with some coordinates associated with it.
Word embedding usually allows the *Skip-gram technique*: predicting "surrounding" words to an existing word.

Word2Vec, Glove, ELMO, and BERT

BERT is a Bidirectional Encoder Representation from Transformers. This is a **language model** or language representation of English. This means that BERT provides a parameterized view of the English language, containing synonyms, and similarities between words and sentences, for example. BERT also provides word embeddings, the same as Word2vec, GloVe, ELMo

, and others There are a lot of similar tools and transformers in the area of natural language processing (NLP). Actually, BERT is a pre-trained system that usually competes directly with the LSTM cells.

Now that we have defined the theoretical background of our project, we will detail in the next part, which tools we must use to build our English-Dutch NMT system.
In this project, we should have used BERT but because of time limitations, we will use Word2vec instead.

Tools and Software Needed for Building an Automatic Translation System with Deep Learning

Our project will consist of building an English-to-Dutch translator using Deep Learning. In the first part, we briefly introduce the main theoretical concepts that are involved. Now we introduce the tools that we shall need:

- TensorFlow
- Keras
- Pandas

There are multiple frameworks that provide API for deep learning. The combination TensorFlow + Keras is - by far - the most popular, but other equivalent frameworks such as PyTorch, Caffe or Theano are also widely used.

These frameworks often provide a 'black box' approach to Neural Networks and they do most of the 'magic' without requiring the user to code the neural network's logic. There are also other ways to build neural networks, for instance, with *deep learning compilers*. Here, however, we do not wish to use such tools.

We will write the code and run the NMT on a Linux centOS 7.8. Centos is reputed to be a very stable Linux distribution, however in reality, almost all the development we will do in the project will be using Python, so the choice of the O.S is not really important.

Versions

We list here the versions of the Python modules that we are using. All these versions can be explicitly installed by using the '==[version]' flag at the end of the pip3 command. For instance: “`pip install tensorflow==2.0`”.
We leave the user the choice of the versions, anyway, depending on their operating system.
We are not using in this project the latest version of these modules.

module	version
TensorFlow	1.5.0
Keras	2.1.0
numpy	1.18.1
pandas	1.1.3
word2vec	

TensorFlow

TensorFlow is a very popular Python framework for the building of neural networks.
To Install TensorFlow, we proceed as follows:

First, we need to install Python.
For this we update the package manager:

```
yum update -y
```

Then we install **Python3**:

```
yum install -y python3
```

This will actually install *Python 3.6* which is not the latest Python version at the moment where we write this tutorial, but that will do it.

Note that the next steps aren't really O.S. dependent and only require Python3.6 to be installed.

Pip3 - the python3 package manager - should be installed by default.

Once this is done, Tensorflow can be installed by running:

```
pip3 install tensorflow
```

The download and installation of the TensorFlow package may take some time since the package is more than 400 MB.

You can control that TensorFlow installed successfully by typing:

```
pip3 show tensorflow
```

The output should be similar to this:

```
Name: tensorflow
Version: 2.3.1
Summary: TensorFlow is an open-source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib64/python3.6/site-packages
Requires: opt-einsum, tensorboard, termcolor, six, h5py, gast, tensorflow-estimator,
google-pasta, astunparse, wrapt, grpcio, absl-py, numpy, keras-preprocessing,
protobuf, wheel
Required-by:
```

Warning: it's important to run *python3* and not simply 'python' (same for *pip3* and not 'pip') because there usually exists a 'system' python in centos which must not be used for the project.

Keras

Now that we have installed TensorFlow, we need to install Keras. Keras is a deep-learning API that will run *on top of TensorFlow*. Keras can run as well on top of Theano for example, but here we choose to associate it with TensorFlow.

To install Keras, the process is identical, we type the following command:

```
pip install keras
```

To check if keras is well installed, just type:

```
pip list | grep Keras
```

Pandas

Pandas is a Python API for data manipulation and data analysis. We will need it, among other things, to prepare the training data for the Deep Learning Model.

This library can be simply installed by using pip3:

```
pip3 install pandas
```

Word2Vec

We need Word2Vec for the word embedding, that is to say, to create the embedding layer in our neural network. As we mentioned previously, there are also other tools that can do the job like GloVe or BERT. BERT, which is not context-free, unlike Word2Vec, would provide more efficiency here because it offers a richer set of information, but unfortunately, it is more complex to integrate.

In this project, we will not use a GPU card. If we wanted GPU support, we should have proceeded slightly differently.

The tool we have installed will allow us to build out the MT software. Here is how the pieces of our puzzle will get assembled:

1) We will process an English dictionary and have BERT create a custom word embedding system for the English language.

2) We will build a sequence-to-sequence RNN with LSTM cells using Keras. Keras has indeed built-in support for everything that we need.

3) We will add an embedding layer to the RNN from the embedding created by BERT, at the start of the LSTM sequence-to-sequence RNN.

4) We will process the parallel corpus linguae English/Dutch by cleaning it, formatting it, and tokenizing it.

5) We will train our Deep Learning model with the processed parallel corpus linguae English/Dutch.

6) We will test our NMT with several English sentences to check its accuracy.

There could be some variations. For instance, instead of using LSTM cells, we could use GRU cells.

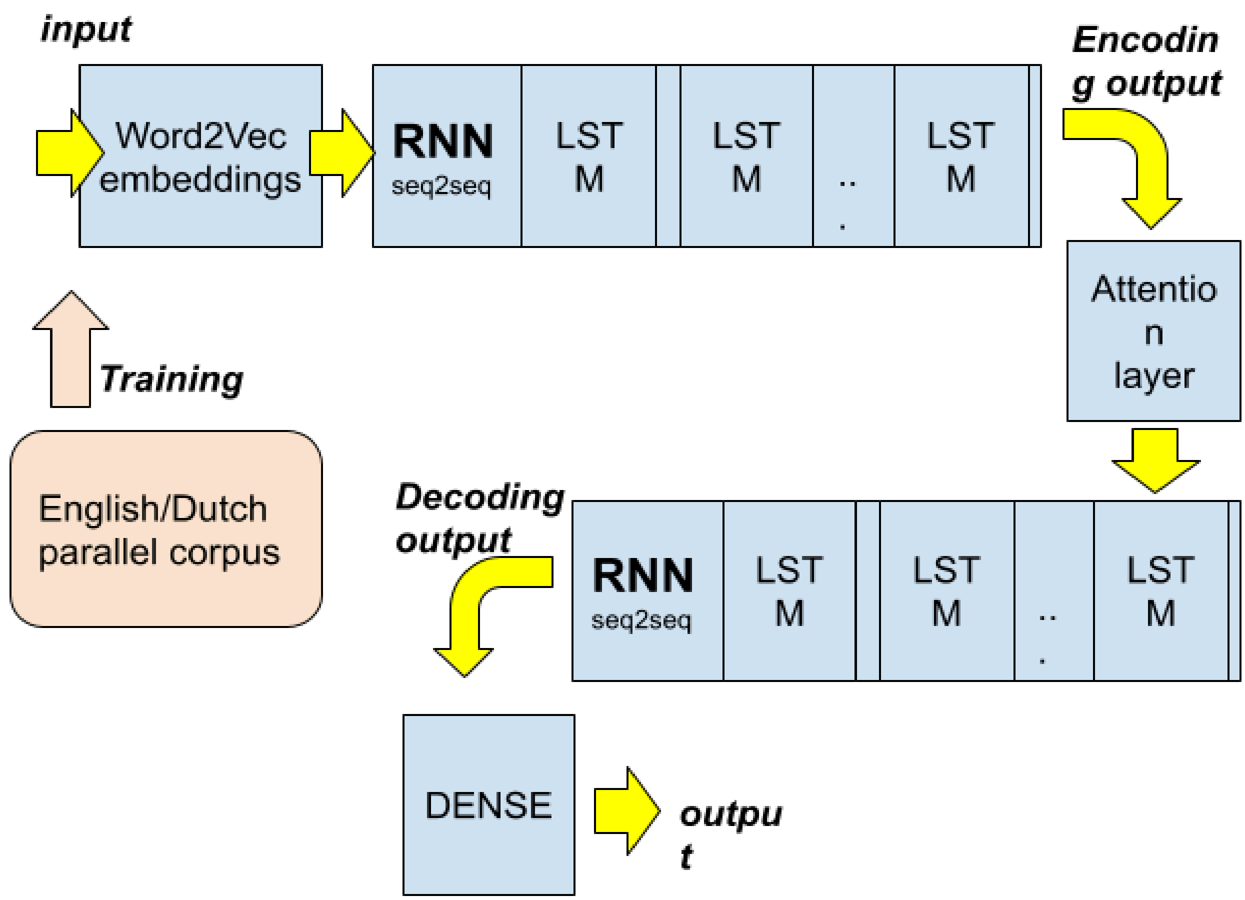


Illustration 4: Workflow of the Neural Machine Translation system we will build

We must also decide how many words we will use for training the system, because training our model may require very important processing power that we do not have in the context of the project.

We will develop functions that will allow us to train and translate from English into another language.

```
def train_model(path_to_data,path_to_model,use_attention=1)
```

```
def translate(path_to_data, path_to_file, path_to_model)
```

- `Path_to_data: path to the training data`
- `Path_to_model path to the model`
- `Use_attention: flag to use the self-attention mechanism`

- `Path_to_file: path to a file containing the English texts to translate`

How to Code an Automatic Translation System by Using TensorFlow and Keras

Now that we have defined all the concepts and tools that we need and that we have installed, we will build the DMT system.
In what follows there is, in fact, very little code that we will write, because most of the logic consists in using pre-formatted 'templates' which use the Keras framework.

A part of our Kears code is essentially inspired by [1].

As a start, we need to load our libraries:

```
import warnings
warnings.filterwarnings("ignore")
import tensorflow as tf
import numpy as np
import string
from numpy import array, argmax, random, take
#for processing imported data
import pandas as pd
#the RNN routines
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding, RepeatVector
#we will need the tokenizer for BERT
from keras.preprocessing.text import Tokenizer
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from keras import optimizers
#that's optional if you want to generate statistical graphs of the DMT
#import matplotlib.pyplot as plt
```

We shall dig into the data processing later when working on the training.

Building our model with Keras is extremely straightforward. We need to use the [LSTM class](#).

Most of the parameters of a LSTM cell are provided by default so the only thing we need to provide is the dimensionality of the output, that is to say, the number of LSTM cells that will be created for our sequence-to-sequence RNN.
From what we have defined, an input (resp output) vector will be the total of the words inside the original sentence (resp translated sentence). But since we use an embedding, we will get tokenized words, which means that words can be split into sub-tokens, which will increase the number of words in the input sentence. We will have to pad anyway the input/output vectors.

The length of 512 is enough here.

```
lstm = tf.keras.layers.LSTM(512)
```

And … that's mostly all of the programming, thanks to the developer(s) of Keras. Most of the readers shouldn't have to dig inside that black box. However, coding an LSTM cell from scratch isn't a really hard challenge at all as its design is simple.

We need to use [the Sequential model provided by Keras]() as well.

```
model = Sequential()
```

Finally, we add a [Dense layer]() to our model. A dense layer takes all the output neurons from the previous layer. We need the dense layer because we're making **predictions** here. Indeed what we want is the sentence in the Dutch language which has the maximal score to be the translated English sentence that has been inputted. A dense layer generally computes a softmax on the outputs of each LSTM cell.

```
model.add(Dense(LEN_RU, activation='softmax'))
```

Where `LEN_RU` is the size of the output vector (we will compute these parameters later on). The same for the variable `LEN_EN`.
Finally here is the main code for our model:

```
model = Sequential()
model.add(LSTM(512))
model.add(RepeatVector(LEN_EN))
model.add(LSTM(512))
model.add(Dense(LEN_RU, activation='softmax'))
rms = optimizers.RMSprop(lr=0.001)
model.compile(optimizer=rms, loss='sparse_categorical_crossentropy')
```

We are using a Keras optimizer named *RMSprop*. It will optimize the gradient descent technique itself used for backpropagation.

We need to add the embedding layer and we also want to include an *attention* layer as well between the encoder and the decoder.

We need to add the embedding layer which is performed with Word2Vec. This is in fact a pre-trained embedding layer. So what we need to do is to generate the Word2Vec weights matrix (the weights of the neurons of the layer) and fill a standard keras Embedding layer with it. We can use the gensim package to get the embedding layer automatically:

```
from gensim.models import Word2Vec
```

Then:

```
model_w2v = Word2Vec(common_texts, size=100, window=5, min_count=1, workers=4)
```

The embedding layer can then be retrieved by the following code:

```
model_w2v.wv.get_keras_embedding(train_embeddings=False)
```

We can call the `model.summary()` function to get an overview of our model:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 100)         1200
_________________________________________________________________
lstm_1 (LSTM)                (None, 512)               1255424
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 8, 512)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 512)               2099200
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
=================================================================
Total params: 3,618,480
Trainable params: 3,617,280
Non-trainable params: 1,200
```

As we mentioned it previously, we wish to add an attention mechanism.
We could write it from scratch but a simpler solution is to use an existing attention Keras module such as the Keras self-attention module.

We need to import the module:

```
from keras_self_attention import SeqSelfAttention
```

And we will add it between the two LSTM blocks by inserting the following line of code:

```
model.add(SeqSelfAttention(attention_activation='sigmoid'))
```

Our model is now complete.

Here is the final code of our neural network coded in Keras:

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import string
from numpy import array, argmax, random, take
#for processing imported data
import tensorflow as tf
import pandas as pd
#the RNN routines
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding, RepeatVector
from keras.preprocessing.text import Tokenizer
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from keras import optimizers
#that's optional if you want to generate statistical graphs of the DMT
#import matplotlib.pyplot as plt
#from keras.utils import plot_model
#import pydot

from gensim.models import Word2Vec
```

```python
from gensim.test.utils import common_texts
from keras_self_attention import SeqSelfAttention


model = Sequential()

model_w2v = Word2Vec(common_texts, size=100, window=5, min_count=1,
workers=4)
model.add(model_w2v.wv.get_keras_embedding(train_embeddings=False))
model.add(LSTM(512))
model.add(RepeatVector(8))

model.add(SeqSelfAttention(attention_activation='sigmoid'))

model.add(LSTM(512))
model.add(Dense(LEN_RU, activation='softmax'))
rms = optimizers.RMSprop(lr=0.001)
model.compile(optimizer=rms, loss='sparse_categorical_crossentropy')

#plot_model(model, to_file='model_plot4a.png', show_shapes=True,
show_layer_names=True)

model.summary()
```

When we run the code, we get the following output:

```
[root@ids ~]# python3 NMT.py
Using TensorFlow backend.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 100)         1200
_________________________________________________________________
lstm_1 (LSTM)                (None, 512)               1255424
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 8, 512)            0
_________________________________________________________________
seq_self_attention_1 (SeqSel (None, 8, 512)            32833
_________________________________________________________________
lstm_2 (LSTM)                (None, 512)               2099200
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
=================================================================
```

```
Total params: 3,651,313
Trainable params: 3,650,113
Non-trainable params: 1,200
```

Now that the model is ready, we will move to the final phase of the development: preparing the data for training the model.

Training and Testing an Automatic Translation System using Deep Learning

Training and testing using the LSTM cells

We first start to train and test a model built only with the core system: TSTM cells and without self-attention and word embedding. The standard Keras embedding component will provide the encoding from a set of words into vectors.

The last phase of our development consists in training the model.
For this, there are several specific tasks that must be performed:

- Cleaning the training data (preprocessing)
- Tokenization of the input data (preprocessing)
- Deciding on the ratio training data / self-test data
- Training of the model

Once the input data are cleaned, our next step is to prepare the input data (source) and output data (target) so that we can have numerical, fixed-size, input, and output models. Indeed we cannot (yet) feed sentences or words to a Keras neural network model.

The Keras tokenizer will create an internal vocabulary made with the words contained in the parallel corpus.

We must first use the function `fit_on_texts`.

This function accepts as an argument a list of sentences and builds a mapping from the most commonly encountered words to indexes. It doesn't encode any sentence but prepares a tokenizer to do so.

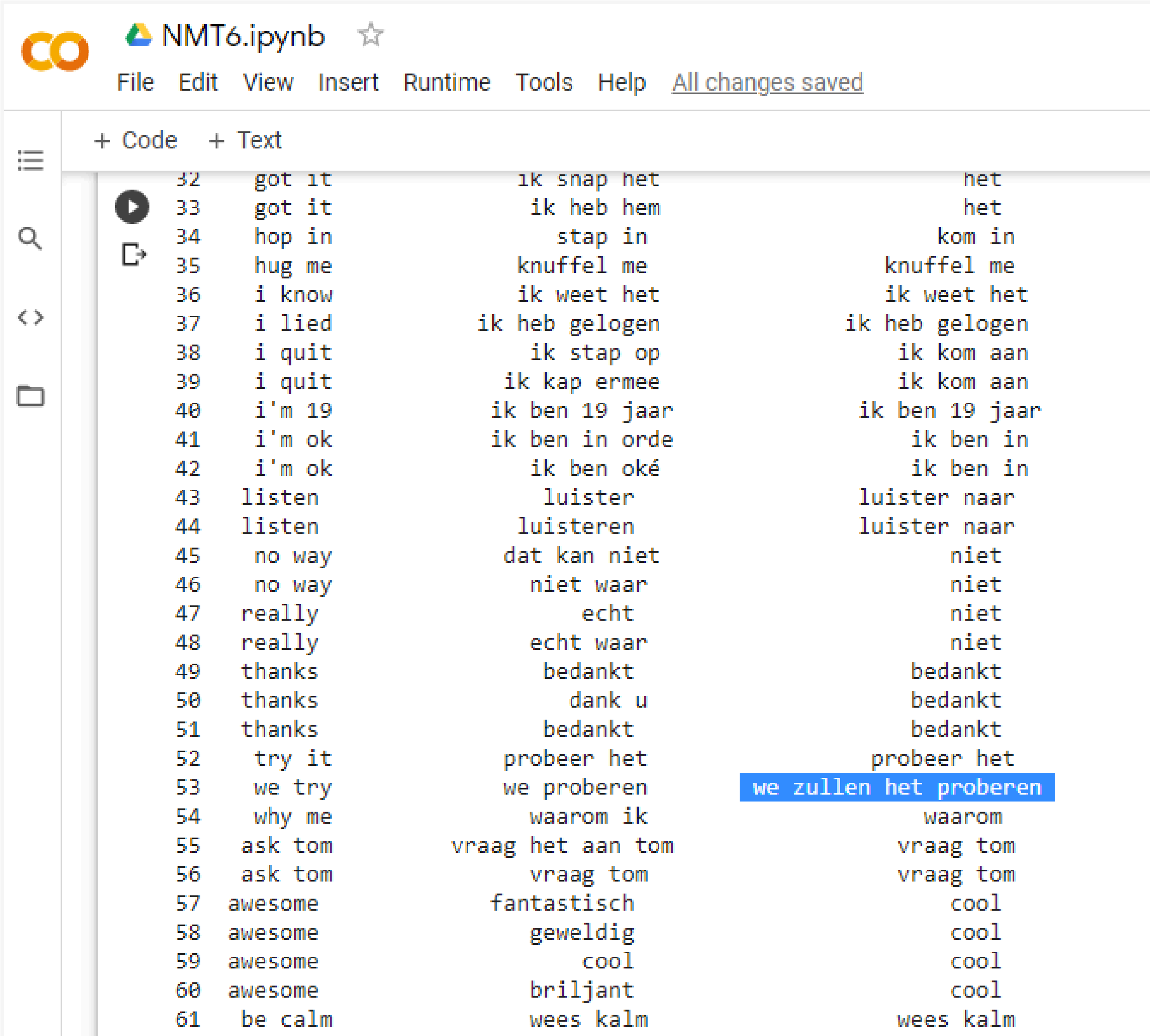
Then we have to provide a way to encode our input sentences. Once we have initialized the tokenizer, we will use the function `texts_to_sequences` for the encoding. The following code retrieves a word from a numerical vector.

```
temp = []
      for j in range(len(i)):
           t = get_word(i[j], ru_tokenizer)
           if j > 0:
               if (t == get_word(i[j-1], ru_tokenizer)) or (t == None):
                    temp.append('')
               else:
                    temp.append(t)
           else:
                  if(t == None):
                         temp.append('')
                  else:
                         temp.append(t)
      return ' '.join(temp)
```

We will use a Google Collab notebook to run the training and the testing since we can use free GPU power there.

First, we run on the first entries of our data set.
We will get an exact result for the entries that fall in the training data and we will get an approximated transaction for the other data. We can check that the translator doesn't behave so badly.

Here we represent the input data in English, then the ideal translation, and finally the model translation.



32	got it	ik snap het	het
33	got it	ik heb hem	het
34	hop in	stap in	kom in
35	hug me	knuffel me	knuffel me
36	i know	ik weet het	ik weet het
37	i lied	ik heb gelogen	ik heb gelogen
38	i quit	ik stap op	ik kom aan
39	i quit	ik kap ermee	ik kom aan
40	i'm 19	ik ben 19 jaar	ik ben 19 jaar
41	i'm ok	ik ben in orde	ik ben in
42	i'm ok	ik ben oké	ik ben in
43	listen	luister	luister naar
44	listen	luisteren	luister naar
45	no way	dat kan niet	niet
46	no way	niet waar	niet
47	really	echt	niet
48	really	echt waar	niet
49	thanks	bedankt	bedankt
50	thanks	dank u	bedankt
51	thanks	bedankt	bedankt
52	try it	probeer het	probeer het
53	we try	we proberen	we zullen het proberen
54	why me	waarom ik	waarom
55	ask tom	vraag het aan tom	vraag tom
56	ask tom	vraag tom	vraag tom
57	awesome	fantastisch	cool
58	awesome	geweldig	cool
59	awesome	cool	cool
60	awesome	briljant	cool
61	be calm	wees kalm	wees kalm

If we try with more complex sentences , outside of course the training data of the model, we get this:

Input	Human translation	Model translation
tom offered mary a handkerchief	tom bood maria een zakdoek aan	tom gaf maria een ("Tom gave maria one")
which floor do you live on	op welke verdieping woont ge	welke welk ben je ("which one are you")
do a lot better than this next time	om het volgende keer veel beter te doen	ik veel dit ("i like this a lot")

had a chance to talk to tom yet	al een kans om met tom te praten	tom kans om te praten (“tom chance to talk”)
he killed himself	hij heeft zichzelf omgebracht	hij heeft zelfmoord gepleegd (“he committed suicide”)
i believed that he would keep his promise	ik geloofde dat hij zijn belofte zou houden	ik wist dat hij zijn zou zijn (“I knew he would be his”)
i have never been to europe	ik ben nog nooit in europa geweest	ik heb nog nooit in canada (“i have never been to canada”)
i am taking a bath now	ik ben aan het baden	ik ben nu een bad het (“i'm bathing it now”)

This sounds promising but of course, this is far from a real professional automatic translation system which demonstrates how hard the challenge is.

Of course, we can load any other training sets from the Tatoeba project such as English/Russian for example.

We can also create a reverse translation by simply reversing the output and input data.

Here is what we get with the Russian parallel corpus:

Input	Human translation	Model translation
they hired tom	они взяли тома на работу	они наняли тома на
it's a very serious matter	это очень серьёзное дело	это очень серьёзное вопрос
the situation became dangerous	ситуация стала опасной	положение трудно
we could do this	мы могли бы это сделать	мы могли это
tom hasn't written me back	том мне не ответил	том мне не написал в
it's your choice tom	это твой выбор том	это тебе выбор том
beijing is bigger than rome	пекин больше чем рим	книга больше чем
that tom knows what mary needs to do	думаю что том знает что мэри нужно делать	думаю что том знает что мэри нужно сделать
do we have to speak french	нам обязательно говорить по французски	нам надо говорить по французски

did tom like your design	тому понравился твой дизайн	тому понравился ваш дизайн
he gave me a piece of friendly advice	он дал мне дружеский совет	он дал мне своим советов
tom doesn't really want to go with us	том не очень хочет идти с нами	том не очень хочет ехать с нами
what time are you going to do that	во сколько вы будете это делать	во сколько ты будешь это делать
you've been very good to me	вы были очень добры ко мне	ты был для мне очень мне

The Russian translator is surprisingly quite good. It comes from the fact that we trained the model with more than 400,000 inputs.

Some flaws appear immediately, however. For example the sentence ' you've been very good to me'' is translated as: " ты был для мне очень мне": "you were for me very me" …

With Self-Attention

We run the model with self-attention. We see mitigated results. In some cases, the translation is close to perfect (yellow) but in some other cases, the translation does not improve or is even inferior in quality to the translation without self-attention (grey).

Input	Without self-attention	With self-attention
they hired tom	они наняли тома на	они наняли тома
it's a very serious matter	это очень серьёзное вопрос	это очень серьёзное дело
the situation became dangerous	положение трудно	ситуация была опасен
we could do this	мы могли это	мы могли сделать это
tom hasn't written me back	том мне не написал в	том не написал на ответ
it's your choice tom	это тебе выбор том	это вам выбор том
beijing is bigger than rome	книга больше чем	воздух больше чем
that tom knows what mary needs to do	думаю что том знает что мэри нужно сделать	думаю что том знает что мэри нужно

do we have to speak french	нам надо говорить по французски	нам надо говорить по французски
did tom like your design	тому понравился ваш дизайн	тому понравился ваш дизайн
he gave me a piece of friendly advice	он дал мне своим советов	он подарил мне подарил пример
tom doesn't really want to go with us	том не очень хочет ехать с нами	том не очень хочет с нами
what time are you going to do that	во сколько ты будешь это делать	во сколько вы будешь это делать
you've been very good to me	ты был для мне очень мне	ты был ко мне очень мне

A Conclusion Regarding The Development Of Machine Learning Based Automatic Translation System

Clearly what we have presented is not acceptable as a professional translation system. It is inferior to an average statistical professional translation system or to a rule-based translation system.

We saw that by adding mechanisms of attention we could improve greatly the accuracy of our model anyway.

To be efficient machine learning based translators requires tremendous amounts of data and very important processing power. This is why such systems are usually trained from the cloud with data coming from a lot of sources.

Machine learning translators are anyway important because they paved the way to more sophisticated systems such as the “universal translator” able to translate any language into any other language, and even, potentially, including unknown dialects.

Deep Learning and the Universal Translator

The universal translator - as described in [2] - is a concept device that potentially allows one to instantaneously translate any language into other, even without prior knowledge of it.

The way such a device could extrapolate and interpret a totally 'exotic' language is still - as of 2020 - a mystery and so there is no clue that such a device could be produced soon. Also, actually, there are no 'exotic' languages, and all languages on Earth are supposed to be recorded and known.

Now with the development of Artificial Intelligence, and especially Deep Learning, we become closer to the development of such a device. Again, miniaturization, increase of power in processors, and research in AI allow the creation of 'primitive' types of Universal Translators.

Presentation of our Deep Learning translation software

Our functions will offer automatic translation from one language into another using the model we have developed.

The code can be found there, as a Google Collab file.

It will be able to create a model from a Parallel corpus that is tab-separated such as the ones from the Tatoeba project.

We run our tool with a file containing some English texts we wish to translate.

Content of test.txt:

```
this is a test
hello
can you give me the bill please
where is the main street
```

```
translate("rus.txt","test.txt","model12")
```

We get the following output:

	input	model translation
0	this is a test	это тест
1	hello	привет
2	can you give me the bill please	не можете мне пожалуйста
3	where is the main street	где здесь улице

The result is correct except for the third one.

We use the French translator:

```
train_model("fra.txt","model_fr")
translate("fra.txt","test.txt","model_fr")
```

	input	model translation
0	this is a test	c'est un d'un
1	hello	
2	can you give me the bill please	tu me donner la s'il te prie
3	where is the main street	où est la rue est rue

The result is overly bad. Only the fourth sentence is translated in some intelligible way. The reason is the complexity of the French language and the fact that the training data is not very important (compared to the Russian dataset)

Here is the result for automatic translation from English to German:

	input	model translation
0	this is a test	das ist eine test
1	hello	
2	can you give me the bill please	könntest sie mir die rechnung geben
3	where is the main street	wo ist die straße

This is almost 100% perfect but the two languages are close enough.

Finally, let's try our English-Dutch translator since we started with it::

	input	model translation
0	this is a test	dit is een nationale
1	hello	hallo
2	can you give me the bill please	kunt je me instapkaart geven
3	where is the main street	waar is de bushalt

It’s not really perfect… "Where is the main street" is translated as "Where is the Bus Station?" and "can you give me the bill please" is translated as "can you give me the boarding pass" so we have very different results depending on the language (and the size of the dataset).

Directions

In most cases, automatic translation software is paradoxically useless. Why? Because most people will never need more than the translation of a kernel of basic sentences. Do we really need machines to perform a professional translation in French of the play "Richard III" by William Shakespeare? These machines will, anyway, never equal the Human genius in such an area.

The real goal is the ability to build systems that can understand unknown and very exotic languages.

In fact there exist in the world more than 6,500 spoken languages. However many of these languages (2,000) are in fact very rare dialects spoken by a small population - usually a tiny ethnic group. For example, the Kaixana or the Taushiro languages are only spoken by … one person in the entire world!.

This should underline the incredible challenges to overcome to build a Universal translator that could convert one of these exotic languages into any other language - let us say into English.

Army personnel, explorers, and scientists in foreign territories should make use of such universal translators since communication in the first moment of an encounter is often vital in such situations. This is where Deep learning can be really useful.

There are plenty of ways of building machine learning systems for machine translation. We just explored one of these ways. It is possible, for example, to use Convolutional neural Networks in addition or use software like moses in combination with the Deep learning model.

Anyway, the main factor for quality will be, as usual, the size of the training set.

References

[1] A Must-Read NLP Tutorial on Neural Machine Translation – The Technique Powering Google Translate. PRATEEK JOSHI

[2] Do Universal Translators Already Exist? MARTIN RUPP.