Technology of Ukrainian-English Machine Translation Based on Recursive Neural Network as LSTM

Myroslav Konyk [1], Victoria Vysotska [1,2], Svitlana Goloshchuk [3], Roman Holoshchuk[1], Sofia Chyrun [1] and Ihor Budz[1]

[1] *Lviv Polytechnic National University, S. Bandera Street, 12, Lviv, 79013, Ukraine*
[2] *Osnabrück University, Friedrich-Janssen-Str. 1, Osnabrück, 49076, Germany*
[3] *University of Economics in Bratislava, Dolnozemská cesta 1, Bratislava, 85235, Slovak Republic*

Abstract

The paper presents a novel approach to developing Ukrainian-English machine translation technology using neural machine translation. A comparative analysis of the developed software product with analogues such as Google Neural Machine Translation, Microsoft Translator and OpenNMT is conducted. The study focuses on the analysis of their main advantages and disadvantages. Here, we introduce an effective solution for the typical architecture of the intelligent Ukrainian-English machine translation system based on recurrent neural networks. The main functional requirements for similar systems are determined, and the technical task is developed. The neural network model test results have shown that the confirmation loss stopped decreasing after 25 epochs.

Keywords

Machine translation, deep learning, Ukrainian language, machine learning, recurrent neural network, English language, RNN, LSTM, Long Short-Term Memory

1. Introduction

Machine translation (MT) is widely used in various areas of our lives as it speeds up natural language translation and improves the traditional translation process. Its relevance is due to the constant demand for translation as a type of information activity and the rapid increase of information exchange value. The digital translation still needs improvement, but the text obtained from an electronic translator allows you to understand the document's meaning in most cases. Further, the user may edit the document with basic knowledge of a foreign language and a command of the subject area the translated information belongs to. Due to the development of machine learning technologies, it is possible to achieve better results than ever. Neural Machine Translation (NMT) is a recently proposed approach to machine translation that differs from traditional statistical machine translation as it focuses on building a single artificial neural network that can be jointly configured to maximise translation performance.

This work aims to study the use of recurrent neural networks for machine learning; to determine the main features and principles of the sequence-to-sequence model for deep learning in machine translation. The subject of research is a recurrent neural network for machine translation. The object of research is machine translation.

The study material is a set of bilingual pairs of sentences with tab separators taken from the open database of sentences and translations *Tatoeba* [1].

Unlike other approaches, such as neural/rule-based/statistical machine translation uses an extensive neural network that applies artificial intelligence to work like the human brain. It is the most advanced

COLINS-2023: 7th International Conference on Computational Linguistics and Intelligent Systems, April 20–21, 2023, Kharkiv, Ukraine
EMAIL: myroslav.konyk.mnsam.2022@lpnu.ua (M. Konyk); victoria.a.vysotska@lpnu.ua (V. Vysotska); svitlana.l.holoshchuk@lpnu.ua (S. Holoshchuk); roman.o.holoshchuk@lpnu.ua (R. Holoshchuk); sofiia.chyrun.sa.2022@lpnu.ua (S. Chyrun); ihor.s.budz@lpnu.ua (I. Budz)
ORCID: 0000-0001-7259-351X (M. Konyk); 0000-0001-6417-3689 (V. Vysotska); 0000-0001-9621-9688 (S. Holoshchuk); 0000-0002-1811-3025 (R. Holoshchuk); 0000-0002-2829-0164 (S. Chyrun); 0000-0002-5400-0984 (I. Budz)

form of machine translation available, with tremendous progress made in recent years through AI-powered self-learning, huge data collection and deep learning. Modern neural machine translation engines are a basis for developing professional translations. Recent technological advances have enabled an increasing number of multinational institutions to use NMT engines to support internal and external communication.

Advantages of using neural machine translation:

- High accuracy: drawing from ever-extending data sets and language modelling, NMT engines can mine the broader context of words and phrases to create more accurate and fluent translations, which can be improved over time. In contrast, conventional phrase-based MT only considers the context of a few words next to the translated word.
- Fast training: neural networks can be trained quickly using automated processes instead of the expensive and largely manual methods required for rule-based MT.
- Simple integration and flexibility: the advantage that NMT carries over its statistical predecessor includes its integration via an application programming interface (API) and software development kit (SDK) into any software and applied to many content files formats.
- Customisation: to improve the results, the user can set up the NMT output and update the model with terminological databases, brand-specific glossaries, and other data sources.
- Cost-efficiency: human translation can be expensive, especially in projects that involve long texts and many languages. NMT produces highly accurate and fast translation systems at low costs. If needed, the user may address human translators for post-editing.
- Scalability: when your translation needs to scale up, neural machine translation can help meet increased demand quickly and efficiently [2].

2. Related works

In recent years, the technology of neural machine translation, based on artificial neural networks with representation learning (deep learning), has rapidly developed. That is why many companies have decided to use this technology for their products. Despite several reviews in the literature that address the importance of neural machine translation, none of the recently published articles has comprehensively examined the critical roles of these technologies in the process of Ukrainian-English translation based on a recursive neural network. In our research, we will first study and discuss these products (Fig. 1) [3-8].

Figure 1: Google and Google Translate [3-4], Microsoft Translator [6-7] and OpenNMT [8] logos

Google Neural Machine Translation (GNMT) is a neural machine translation (NMT) system developed by Google. The GNMT network can undertake interlingual machine translation by encoding the semantics of the sentence rather than by memorising phrase-to-phrase translations. Next, we will consider the main benefits and drawbacks of the GNMT system [3].

The benefits of Google neural machine translation can be summed up as follows:

- ***Google Translate is free***. An experienced professional translator can sometimes be costly, but the quality balances the expenses spent.
- ***Google Translate is quick***. One of the main advantages of Google Translate is that it is speedy. A human translator(s) cannot compete with the speed nor, as a result, the number of translations that Google Translate can perform. In an average workday, an experienced translator can translate about 2,000 words maximum (300-400 words/hour), depending on the difficulty of the text. In contrast, Google Translate can produce a translation with the same number of words in seconds [5].
- ***Google Translate uses a statistical method to form an online translation database based on language pair frequency***. Google Translate uses a statistical approach to develop an online database for translations often made by humans and available online.

The main drawbacks include the following:

- ***Google Translate may "lose" the word meaning because there is no way to incorporate context.*** The complexity of the text and any context that cannot be interpreted without accurate language

knowledge make the error probability higher. Direct translation is standard with Google Translate and often results in nonsensical literal translations, while professional translators make much effort to ensure this does not happen. They use well-established online glossaries, back translation methods, proofreaders and reviewers to achieve this.

- ***The quality of translation relies on the language pair.*** The source and target languages involved also affect the quality of the translation. Since Google's web-based translation database is built primarily from existing online translations, standard translations for languages, e.g. German or English, tend to be more accurate. The translations for other languages that are not available in Google's database are less likely to be precise.
- ***Google Translate often translates with significant grammatical errors*.** Since Google's translation system uses a method based on language pair frequency, grammatical rules are not considered [5].
- ***Google Translate does not have a system to correct translation errors.*** There is no way of reporting errors to avoid repeating them, nor is there a way to proofread what has been translated unless one is fluent in both the source and the target language [5].

Microsoft Translator is the main competitor of Google Translate. The advantages of the Microsoft Translator App include the following features [7]:

- **Group Interactions:** The programme's distinctive feature is its compatibility with conversating with a large group of people. It generates a conversation code for a user that can be shared with everyone interested in a discussion.
- **Website and Skype Integration:** Skype and web integration are another outstanding application feature. The users do not have to worry if the website is in a foreign language, as the Microsoft Translator application can interpret it easily. Users can also enjoy the software's benefits when using Skype.
- **Interprets all Common Vernaculars:** The software can translate the world's most spoken languages; no matter which device the application is used, it can translate the tongues smoothly.
- **Friendly Interface:** Microsoft has offered a functional and pleasant interface. It does not make people feel bored and businesslike. Moreover, it provides easy solutions to its users by offering a simple procedure.

The Microsoft Translator App also has the following disadvantages:

- Limited Languages.
- Interpretations of One Type.
- Not Always Accurate.

Open-Source Neural Machine Translation (OpenNMT) is an open-source ecosystem for neural machine translation and neural sequence learning [8]. OpenNMT contributes to academia and industry by removing limitations and barriers through an open-source engine. The system is designed to be easy to use and expandable, maintaining efficiency and state-of-the-art accuracy.
It provides implementations in two popular deep learning frameworks:

1. **OpenNMT-py:** user-friendly and multimodal, benefiting from PyTorch ease of use**.**
2. **OpenNMT-tf:** modular and stable, powered by the TensorFlow ecosystem [8].

Each implementation has its own set of unique features but shares similar goals:

- Highly configurable model architectures and training procedures;
- Efficient model serving capabilities for use in real-world applications;
- Extensions allow other tasks such as text generation, tagging, summarisation, image-to-text, and speech-to-text [8].

In summary, we may conclude that the advantage of neural machine translation systems is that they are end-to-end models that do not have a pipeline of specific tasks. The disadvantage is the need for a bilingual set and the ongoing problem of processing rare words.

3. Methods

The Python programming language and the Jupyter programming environment are used to develop the neural machine translation system. A set of Keras tools is employed for the natural process of building the model.

Python. A brief review of the programming language itself is worth our attention. Python is the most common programming language for artificial neural networks. It has a low entry threshold: no one writes neural networks in Python from scratch because it is time-consuming. There are libraries for Python neural networks already written by experts. Thus, a whole neural network community has formed around Python.

The advantages of using Python include the following main features:

- **Conciseness and interoperability:** The language allows you to develop complex algorithms in a short time. It is distinguished by simplicity, conciseness and expressiveness. In addition, it has a powerful interoperability mechanism with C\++, which allows for fast calculations. Because of this, engineers can create simple and complex neural networks in Python.
- **Flexibility:** Neural networks are primarily small programs, but there is a need to change them often, choosing the best architecture, data processing, and other parameters. Therefore, there are practically no difficulties with legacy code, but there is a need for rapid development. Creating and building neural networks in Python is an option that meets these requirements better than using C++ or Java.

Keras. The next stage of our study is to consider the toolkit built based on TensorFlow Keras. It is a powerful platform that can scale to large clusters of GPUs or an entire TPU module [9]. Keras is a deep learning API written in Python that operates on top of the TensorFlow machine learning platform. It is designed to emphasise the possibility of rapid experimentation, as moving from idea to result as quickly as possible is the key to good research [10]. Benefits of using Keras:

- **Simple:** Keras reduces the developer's cognitive load, so they focus on the parts of the issue that really matter.
- **Flexible:** Keras adopts the progressive disclosure of complexity principle: simple workflows should be fast and easy. In contrast, arbitrarily advanced workflows should be possible through a clear path that builds on your learning.
- **Powerful:** Keras provides industry-strength performance and scalability: it is used by organisations and companies, including NASA, YouTube and Waymo [10].

Figure 2 shows the popular open-source NMT toolkits on GitHub.

Name	Language	Framework	Status
Tensor2Tensor	Python	TensorFlow	Deprecated
FairSeq	Python	PyTorch	Active
Nmt	Python	TensorFlow	Deprecated
OpenNMT	Python/C++	PyTorch/TensorFlow	Active
Sockeye	Python	MXNet	Active
Nematus	Python	Tensorflow	Active
Marian	C++	–	Active
THUMT	Python	PyTorch/TensorFlow	Active
NMT-Keras	Python	Keras	Active
Neural Monkey	Python	TensorFlow	Active

Figure 2: Popular NMT toolkits

Jupyter. JupyterLab is the latest web-based interactive development environment for notebooks, code, and data. Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning. A modular design contains extensions to expand and enrich functionality [11]. Jupyter Notebook is the original web application for creating and presenting data science projects [11].

Figure 3: Jupyter logo [11]

Jupyter Notebook integrates code and its output into a single document that combines visualisations, narrative text, mathematical equations, and other important media. We assume it is a single document where users can run code, display the output, add explanations, formulas, and charts, and make their work more transparent, understandable, repeatable, and shareable [12].

Using Notebooks is now a significant part of the data science workflow at companies across the globe. If the software developer's goal is to work with data, using a Notebook will speed up their workflow and make communicating and sharing the results more accessible. Another essential feature of the programme is that as part of the open-source Project Jupyter, Jupyter Notebooks are entirely free. The software can be downloaded independently or as part of the Anaconda data science toolkit [12].

NMT Model. The general architecture of our model is explained below. It is a sequence-to-sequence (Seq2Seq) neural network model with inner layers of recurrent neural networks [13-15].

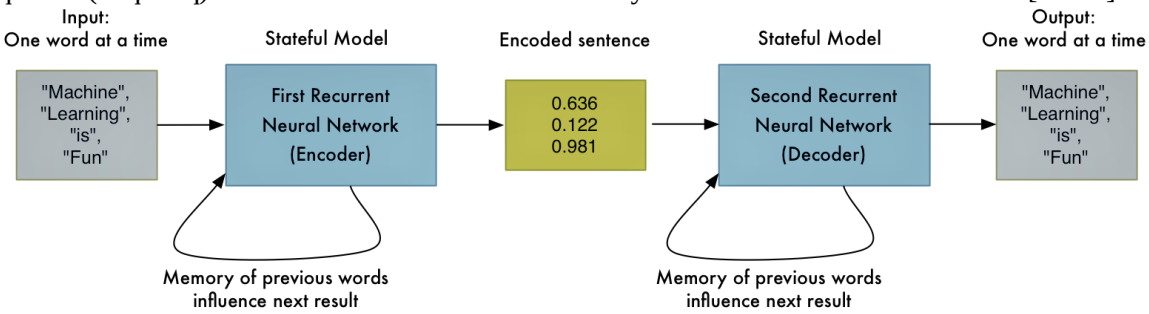

Figure 4: Neural machine translation [13-14]

Each Seq2Seq model contains two main components:

- **Encoder** outputs some value for each word in the input sentence. For every input word, the encoder outputs a vector and a hidden state and uses the hidden state for the next input word [16-17].
- **Decoder** reads the encoder's output vector(s) and outputs a sequence of words to create a translation [17].

Long Short-Term Memory (LSTM) is a type of RNN capable of learning long-term dependencies. All recurrent neural networks have the form of a chain of recurrent neural network modules. LSTMs also have this chain architecture, but the repeated module has a different structure. Instead of having one layer of a neural network, four interact exceptionally [18-19].

The visualisation of the architecture of our neural machine translation model is shown in Fig. 5 [20].

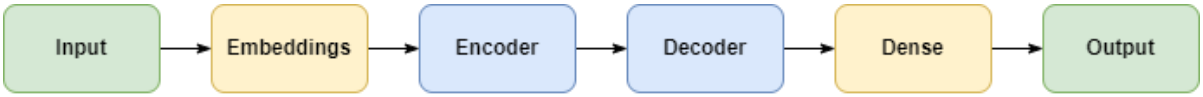

Figure 5: Architecture of Seq2Seq model [19]

4. Experiments, results and discussion

There are several ways we can formulate the task of training an RNN to write text, in this case, Ukrainian-English translation. However, we choose to introduce it as a many-to-one sequence mapper [19-29]. In the first stages of system design, the main task is to analyse the implementation process of this system. It is necessary to build a model describing the work process, containing all the essential information about the processes' functions and the work organisation's peculiarities [30-39].

Use case diagram is the initial conceptual model of the system in the process of its design and development (Fig. 6). It consists of actors, use cases, and relationships between them.

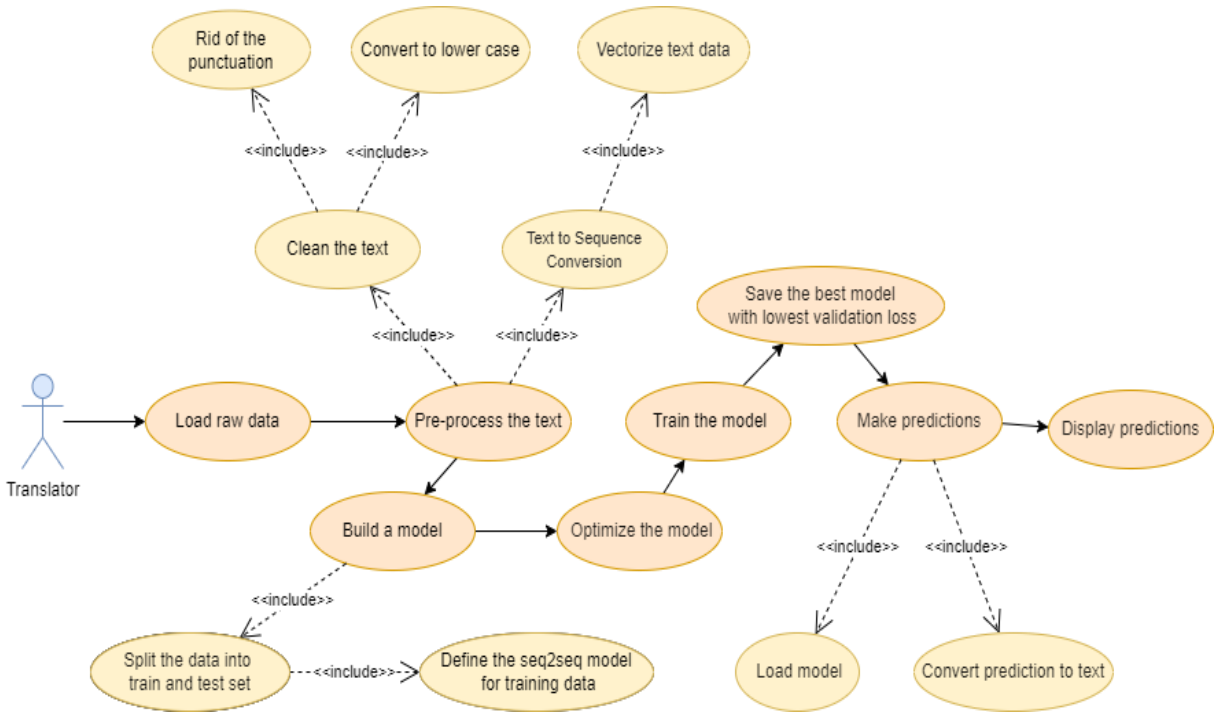

Figure 6: Use case diagram for the neural machine translation

Below is a description of the main options for using the system and actors. The actors of the offered system include [21-23]:

- Translator: actor performing the process of using the neural machine translation system.
- Application options:

1. Load raw data: loading raw data (a pair of two languages)
2. Pre-processing the text: the process of text pre-processing.
    a. Clean the text: the process of cleaning the text
        i. Rid of the punctuation
        ii. Convert to lowercase
    b. Text to Sequence Conversion: the process of converting input and output sentences into whole sequences of a fixed length
        i. Vectorize text data: the process of converting our sentences into sequences of integers
3. Build a model: the process of building a neural machine learning model
    a. Split the data into train and test sets: the process of dividing data into training and test sets for model and evaluation training
    b. Define Seq2Seq model for training data: the process of defining the seq2seq2 architecture
4. Optimise model: model optimisation process
5. Train the model: model training process (30 epochs with a packet size of 512)
6. Save the best model with the lowest validation loss: retaining the best model with a minor validation loss
7. Make predictions: the process of making predictions
    a. Load model – the process of downloading the created model
    b. Convert prediction to text – the process of converting numerical predictions into text
8. Display predictions – the process of output on the screen

A component diagram is a UML modelling language element having a static diagram structure. It shows the structural components of the software system and the relationships between them [24].

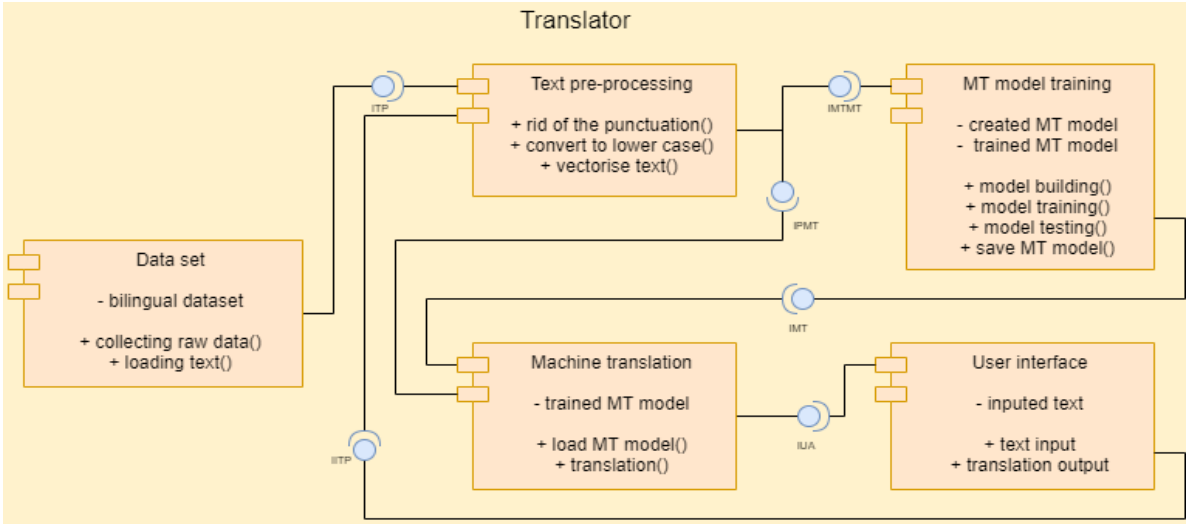

Figure 7: Component diagram of the neural machine translation system model [24]

The following interfaces are used in this diagram:

- A diagram of components of the neural machine translation system model (ITP) is an interface for transferring downloaded text data to the input of the word processor.
- Interface Input Text Processing (IITP) – an interface for transmitting user-entered text to a primary word processing module.
- Interface Machine Translation (IMT) is used for loading the most effective stored machine translation model for further use in the process of neural machine translation of pre-processed text entered by the user.
- Interface Machine Translation Model Training (IMTMT) transmits the original processed (cleaned, normalised, tokenised, digitised) training data to the machine translation model training input.
- Interface Processing-Machine Translation (IPMT) transfers the output (from the text translation module) processed text data, entered by the user to be translated, to the input module of the direct NMT.
- Interface User-Application (IUA) is the interface for transferring the resulting translation of the user's text to the User Interface module for its final output.

Activity diagram is another crucial behavioural diagram in UML diagram to describe dynamic aspects of the system. An activity diagram is an advanced flow chart that models the flow from one activity to another in the system [25-27].

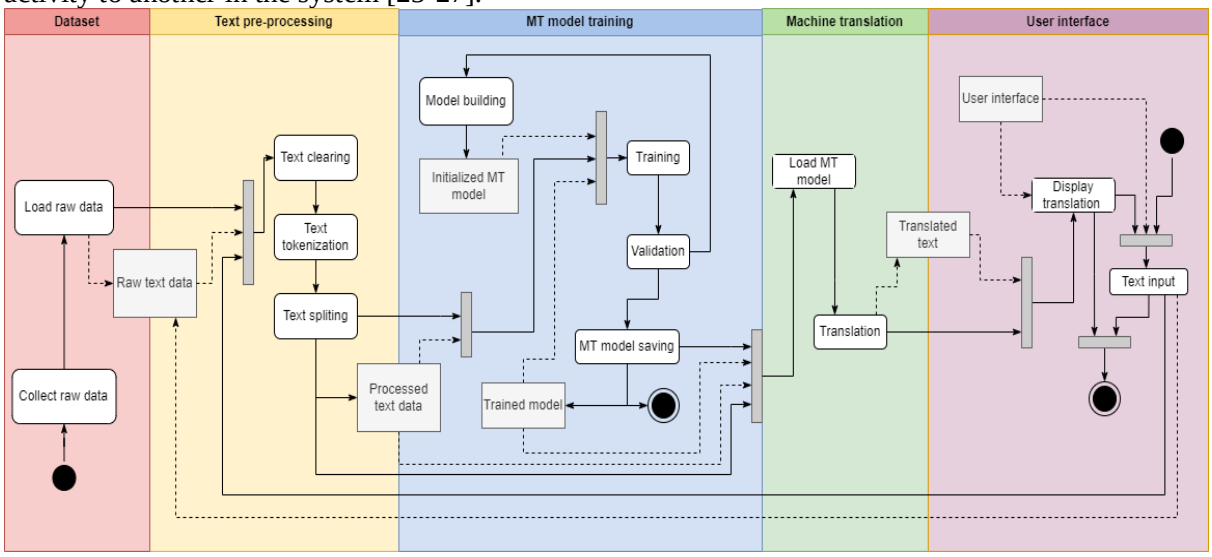

Figure 8: Activity diagram for a neural machine translation system [25-27]

An activity diagram is a great tool for achieving the following goals:

- to write an algorithm logic;
- to illustrate a business process or workflow between users and the system;
- to simplify and improve any process by identifying complex use cases;
- to model software architecture elements such as method, function, and operation.

The first thing we need to do is import the necessary libraries.

```
import string
import re
from numpy import array, argmax, random, take
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding, RepeatVector
from keras. pre-processing.text import Tokeniser
from keras.callbacks import ModelCheckpoint
from keras. pre-processing.sequence import pad_sequences
from keras.models import load_model
from keras import optimisers
import matplotlib.pyplot as plt
% matplotlib inline
pd.set_option('display.max_colwidth', 200)
```

Next, we will write a function for reading data in Jupyter Notebook [19]:

```
def read_text(filename):
        file = open(filename, mode='rt', encoding='utf-8') # open the file
        text = file.read() # read all text
        file.close()
        return text
```

Then, another function for dividing the text into English-Ukrainian pairs separated by a symbol' \n' is defined. Further, we correspondingly split these pairs into English and Ukrainian sentences [19].

```
def to_lines(text): # split text into sentences
      sents = text.strip().split('\n')
      sents = [i.split('\t') for i in sents]
      return sents
```

Now we can use these functions to read text into an array of the necessary format [19]:

```
data = read_text("ukr.txt")
ua_eng = to_lines(data)
ua_eng = array(ua_eng)
```

The actual data consists of over 150,000 sentence pairs. However, we will only use the first 50,000 per item to reduce the training time of the model [19].

```
ua_eng = ua_eng [: 50000 ,:]
```

Next, we pre-process the text. It is an essential step in any project, especially in NLT. The data we process is often unstructured, so there are certain things we need to take care of before moving on to the model-building stage [1, 19].

a) Text cleaning

First, we analyse our data as it helps us decide which pre-processing steps to take:

```
ua_eng
      array([['Go.', 'Йди.',
              'CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #6584257 (deniko)'],
             ['Hi.', 'Вітаю!',
              'CC-BY 2.0 (France) Attribution: tatoeba.org #538123 (CM) & #414700 (deniko)'],
             ['Hi.', 'Привіт.',
              'CC-BY 2.0 (France) Attribution: tatoeba.org #538123 (CM) & #3841503 (rmdas)'],
             ...,
             ['I was in the bathroom.', 'Я був у туалеті.',
              'CC-BY 2.0 (France) Attribution: tatoeba.org #5828323 (OsoHombre) & #5925539 (deniko)'],
             ['I was in the bathroom.', 'Я була у ванній кімнаті.',
              'CC-BY 2.0 (France) Attribution: tatoeba.org #5828323 (OsoHombre) & #5925540 (deniko)'],
             ['I was inspired by Tom.', 'Мене надихнув Том.',
              'CC-BY 2.0 (France) Attribution: tatoeba.org #7453450 (CK) & #8098178 (deniko)']],
            dtype='<U197')
```

The next step is to remove punctuation and convert text into lowercase.

```
# Remove punctuation
ua_eng[:,0] = [s.translate(str.maketrans('', '', string.punctuation)) for s in
ua_eng[:,0]]
ua_eng[:,1] = [s.translate(str.maketrans('', '', string.punctuation)) for s in
ua_eng[:,1]]
for i in range(len(ua_eng)): # convert text to lowercase
    ua_eng[i,0] = ua_eng[i,0].lower()
    ua_eng[i,1] = ua_eng[i,1].lower()
```

b) Converting text into a sequence

The Seq2Seq model requires us to convert input and output sentences into fixed-length integer sequences. Before doing that, we visualise the length of the sentences by collecting the length of all sentences in two separate lists for English and Ukrainian languages [19].

```
eng_l = [] # empty lists
ua_l = []
for i in ua_eng[:,0]: # populate the lists with sentence lengths
      eng_l.append(len(i.split()))
for i in ua_eng[:,1]:
      ua_l.append(len(i.split()))
length_df = pd.DataFrame({'eng':eng_l, 'ua':dua_l}) length_df.hist(bins = 30)
plt.show()
```

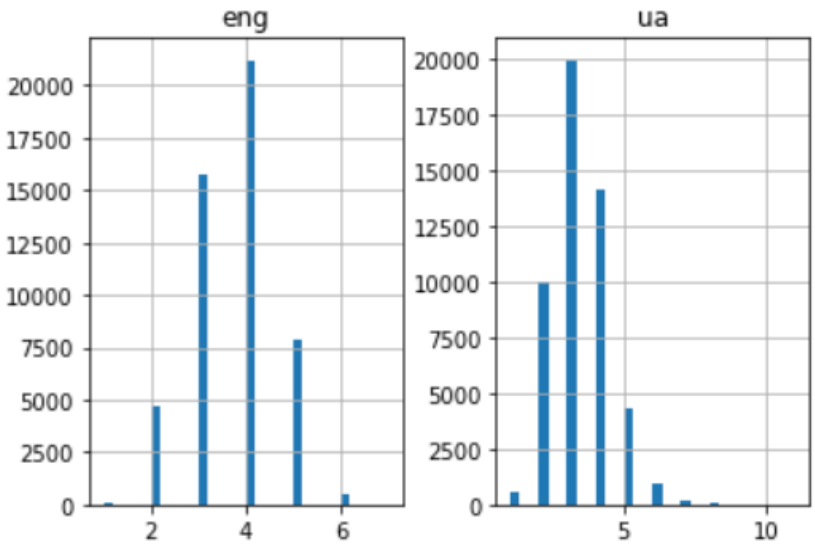

Figure 9: Sentence length graphs [19]

We may assume that the maximum length of English sentences is six words, and there are eight in Ukrainian sentences. Next, we vectorise our text data using the *Keras Tokenizer()* class. It turns out sentences into sequences of integers. We can then pad these sequences with zeros to make all rows the same length. And afterwards, the lexemes for both Ukrainian and English sentences are prepared [19].

```
def tokenization(lines): # function to build a tokeniser
          tokenizer = Tokenizer()
          tokenizer.fit_on_texts(lines)
          return tokeniser
    eng_tokenizer = tokenization(ua_eng[:, 0]) # prepare an english tokeniser
    eng_vocab_size = len(eng_tokenizer.word_index) + 1
    eng_length = 8      # print('English Vocabulary Size: %d' % eng_vocab_size)
```

```
English Vocabulary Size: 5522
```

```
    ua_tokenizer = tokenization(ua_eng[:, 1]) # prepare Ukrainian tokeniser
    ua_vocab_size = len(ua_tokenizer.word_index) + 1
    ua_length = 8       # print('Ukrainian Vocabulary Size: %d' % ua_vocab_size)
```

```
Ukrainian Vocabulary Size: 14066
```

The shown below code block contains a function to prepare sequences. It also performs string complements up to the maximum sentence length as stated above [19].

```
def encode_sequences(tokenizer, length, lines):   # encode and pad sequences
    seq = tokenizer.texts_to_sequences(lines) # integer encode sequences
    seq =pad_sequences(seq,maxlen=length,padding='post')# pad sequences with 0 values
    return seq
```

Then we proceed to build our model. First, we divide the data into training and test sets for model training and its evaluation [19].

```
from sklearn.model_selection import train_test_split
# split data into train and test set
train,test= train_test_split(ua_eng,test_size=0.2,random_state= 12)
```

The next step is to code our sentences. We encode Ukrainian sentences as input sequences and English sentences as target sequences. It is done for both training and testing datasets.

```
trainX = encode_sequences(ua_tokenizer,ua_length,train[:, 1])# prepare training data
trainY = encode_sequences(eng_tokenizer, eng_length, train[:, 0])
testX = encode_sequences(ua_tokenizer,ua_length,test[:, 1]) # prepare validation data
testY = encode_sequences(eng_tokenizer, eng_length, test[:, 0])
```

Subsequently, we define the architecture of the model. As mentioned earlier, we use an embedding layer and an LSTM layer for the encoder; for the decoder, we use another LSTM layer and then a dense layer [19].

```
def build_model(in_vocab,out_vocab, in_timesteps,out_timesteps,n):# build NMT model
      model = Sequential()
      model.add(Embedding(in_vocab, n, input_length=in_timesteps,
      mask_zero=True))
      model.add(LSTM(n))
      model.add(RepeatVector(out_timesteps))
      model.add(LSTM(n, return_sequences=True))
      model.add(Dense(out_vocab, activation='softmax'))
      return model
```

We use the RMSprop optimiser in this model because it shows positive results in dealing with recurrent neural networks [19].

```
# model compilation (with 512 hidden units)
model = build_model(ua_vocab_size, eng_vocab_size, ua_length, eng_length, 512)
rms = optimizers.RMSprop(lr=0.001)
model.compile(optimizer=rms, loss='sparse_categorical_crossentropy')
```

We should mention that we use '*sparse_categorical_crossentropy*' as the loss function. It is reasoned by the function's ability to use the target sequence in its pure state instead of a one-time encoded format. Using such a vast vocabulary, the former encoding of target sequences can fully exhaust the system's memory. We train our model for 30 epochs, with a batch size of 512 and a validation distribution of 20%. The model is trained with 80% of the data; the rest is left for evaluation.

```
filename = 'model_UA_to_ENG'
history = model.fit(trainX, trainY.reshape(trainY.shape[0], trainY.shape[1], 1),
                        epochs=30, batch_size=512, validation_split = 0.2,
                        callbacks=[checkpoint], verbose=1) # train model
```

Finally, we can download the saved model and make predictions on unseen data – testX [19].

```
model = load_model('model_UA_to_ENG')
preds = model.predict_classes(testX.reshape((testX.shape[0], testX.shape[1])))
```

These predictions are sequences of integers. We need to convert these integers into their corresponding words, so we need to define a function for that [19]:

```
def get_word(n, tokenizer):
          for word, index in tokeniser.word_index.items():
              if index == n:
                  return word
          return None
```

The obtained sequences of integers representing predictions are converted into an English text:

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

Next, we place the original English sentences in the test data set and the predicted sentences in the data frame:

```
pred_df = pd.DataFrame({'UA': test[:,1], 'actual_ENG' : test[:,0], 'predicted_ENG'
:  preds_text})
pd.set option('display.max colwidth', 200)
```

Finally, we can run a benchmark test and review the results (Fig. 10 - 11).

```
pred_df.head(15)
```

	UA	actual_ENG	predicted_ENG
0	не звязуйтеся з томом	dont mess with tom	dont mess with tom
1	дайти мені піти	let me go	let me go
2	мені потрібна порада	i want advice	i want advice
3	ворушися	get a move on	get a move on
4	я маю знайти тома	ive got to find tom	ive got to find tom
5	хіба ти не здивований	arent you surprised	arent you surprised
6	том має двох сестер	tom has two siblings	tom has two siblings
7	у мене немає голосу	i cant carry a tune	i cant carry a tune
8	а як щодо поцілунку	how about a kiss	how about a kiss
9	я зараз мешкаю в бостоні	i live in boston now	i live in boston now
10	том прийшов додому	tom got home	tom got home
11	вітаю томе	hello tom	hello tom
12	мені дуже шкода	im so sorry	im so sorry
13	я старанний	im diligent	im diligent
14	він дуже маленький	its very small	its very small

Figure 10: Obtained results (the first 15 pairs)

```
pred_df.tail(15)
```

	UA	actual_ENG	predicted_ENG
9985	я залишився з томом	i stayed with tom	i stayed with tom
9986	вони повечеряли	they ate supper	they ate supper
9987	банани жовті	bananas are yellow	bananas are yellow
9988	велосипед мій	this is my bicycle	this is my bicycle
9989	зателефонуйте до лікарні	call the hospital	call the hospital
9990	можна мені прийняти участь	may i participate	may i participate
9991	у вас іде кров	youre bleeding	youre bleeding
9992	я підступний	im sneaky	im sneaky
9993	я хотіла би з нею побачитися	id like to see her	id like to see her
9994	ти бачиш тома	can you see tom	can you see tom
9995	я запросила тома до себе	i invited tom over	i invited tom over
9996	у мене два племенніки	i have two nephews	i have two nephews
9997	хто тобі допоможе	wholl help you	wholl help you
9998	ти ніколи не змінишся	youll never change	youll never change
9999	я кинув палити	ive quit smoking	ive quit smoking

Figure 11: Obtained results (the last 15 pairs)

Now we would like to describe the model training statistics. For this, we apply a function called *ModelCheckpoint()* [28]. Application of the *ModelCheckpoint*():

```
filename = 'model_UA_to_ENG'
checkpoint = ModelCheckpoint(filename, monitor='val_loss', verbose=1,
                             save_best_only=True, mode='min') # set checkpoint
history = model.fit(trainX, trainY.reshape(trainY.shape[0], trainY.shape[1], 1),
                    epochs=30, batch_size=512, validation_split = 0.2,
                    callbacks=[checkpoint], verbose=1) # train model
```

The following arguments of the ModelCheckpoint function are used:

filename : string or `PathLike`, path to save the model file. The directory of the filepath should not be reused by any other callbacks to avoid conflicts [28].

monitor : the metric name to monitor.

Note:

- Prefix the name with "*val_*" to monitor validation metrics.
- Use "*loss*" or "*val_loss*" to monitor the model's total loss [28].

verbose : verbosity mode, 0 or 1. Mode 0 is silent, and mode 1 displays messages when the callback takes action [28].

save_best_only: if `save best only=True`, it only saves when the model is considered the "best" and the latest best model according to the quantity monitored will not be overwritten.

mode : one of {*'auto'*, *'min'*, *'max'*}. If `save best only=True`, the decision to overwrite the current save file is made based on either the maximisation or the minimisation of the monitored quantity. For `val_loss`, this should be `min` [28].

So, we will build a graph and compare the model's training loss and validation loss.

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['train', 'validation'])
plt.show()
```

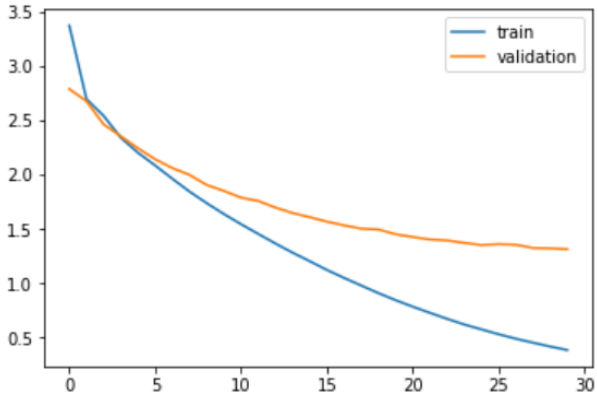

Figure 12: Loss and accuracy graph of the proposed model

As we can see in the diagram above, the validation loss ceased decreasing after 25 epochs.

5. Conclusions

In the course of work, we developed a project, the objective of which was to design a system of Ukrainian-English translation using the means of neural machine translation. First, we discussed essential considerations in developing the research question, defined objectives and tasks, constructed the subject and object of research, scientific novelty and practical value of the developed project.

The first stage of our study compared the developed product with analogues such as Google Neural Machine Translation (GNMT), Microsoft Translator and OpenNMT were carried out, determining their advantages and disadvantages. Then, a system analysis of the developed product was conducted, and UML diagrams were constructed (use case diagram, components diagram and activity diagram). After that, the research hypothesis and the technical task were formulated.

The next stage of our study dealt with selecting methods and means of the developed product. We proved the advantages of Python, the Keras toolkit and the Jupyter Notebook development environment as the necessary tools for our work. The general architecture of the Seq2Seq model using LSTM layers was also presented. Having analysed the software process and its structure, we performed system testing

to evaluate system specifications and describe the results. Then we made the training statistics of the model with a description of its outcome. All things considered, it seems reasonable to assume that future research should explore improving the translation's system accuracy and increasing the model's speed, which will involve exploring additional or alternative architectural solutions.

6. Acknowledgement

Part of this paper was written within the project which received funding from the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project No. 09I03-03-V01-00118.

7. References

[1] Tatoeba. URL: https://tatoeba.org/uk/, https://tatoeba.org/en/.
[2] Machines That Think: The Rise of Neural Machine Translation. URL: https://www.memsource.com/blog/neural-machine-translation/.
[3] Google neural machine translation. URL: http://surl.li/cdbya.
[4] Google Translate for Android has become smarter. URL: https://itechua.com/technologies/83561. [5] The Pros and Cons of Google Translate vs. Professional Translation. URL: https://www.languageconnections.com/blog/the-pros-cons-of-google-translate/.
[5] Microsoft Translator. URL: https://ehlion.com/magazine/microsoft-translator/.
[6] Microsoft Translator App, The Advantages and Disadvantages. URL: https://www.dutchtrans.co.uk/microsoft-translator-app-the-advantages-and-disadvantages/.
[7] OpenNMT. URL: https://opennmt.net/.
[8] Exascale Machine Learning. URL: https://keras.io/.
[9] About Keras. URL: https://keras.io/about/.
[10] Jupyter. URL: https://jupyter.org/.
[11] What is Jupyter Notebook? URL: https://www.dataquest.io/blog/jupyter-notebook-tutorial/.
[12] A. Geitgey. Machine Learning is Fun Part 5: Language Translation with Deep Learning and the Magic of Sequences. URL: https://medium.com/@ageitgey/machine-learning-is-fun-part-5language-translation-with-deep-learning-and-the-magic-of-sequences-2ace0acca0aa.
[13] Course:CPSC522/Recurrent Neural Networks. URL: https://wiki.ubc.ca/Course:CPSC522/Recurrent_Neural_Networks
[14] NLP from Scratch: Translation with A Sequence to Sequence Network and Attention. URL: https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html.
[15] Sequence to Sequence Models. URL: https://www.analyticsvidhya.com/blog/2020/08/a-simpleintroduction-to-sequence-to-sequence-models/.
[16] Long short-term memory. URL: https://en.wikipedia.org/wiki/Long_short-term_memory.
[17] W. Koehrsen, Recurrent Neural Networks by Example in Python. URL: https://towardsdatascience.com/recurrent-neural-networks-by-example-in-python-ffd204f99470.
[18] P. Joshi. A Must-Read NLP Tutorial on Neural Machine Translation. URL: https://www.analyticsvidhya.com/blog/2019/01/neural-machine-translation-keras/
[19] J. Brownlee. How to Clean Text for Machine Learning with Python. URL: https://machinelearningmastery.com/clean-text-machine-learning-python/
[20] R. Agrawal. Text preprocessing. URL: https://www.analyticsvidhya.com/blog/2021/06/mustknown-techniques-for-text-preprocessing-in-nlp/
[21] A. Takezawa. Seq2Seq LSTM Model in Keras. URL: https://towardsdatascience.com/how-toimplement-seq2seq-lstm-model-in-keras-shortcutnlp-6f355f3e5639
[22] R. Hinno. Tuned version of seq2seq tutorial. URL: https://towardsdatascience.com/tuned-versionof-seq2seq-tutorial-ddb64db46e2a
[23] H. Patel. Neural Machine Translation (NMT) with Attention Mechanism. URL: https://towardsdatascience.com/neural-machine-translation-nmt-with-attention-mechanism5e59b57bd2ac

[24] Dive into Deep Learning. Machine Translation and the Dataset. URL: https://d2l.ai/chapter_recurrent-modern/machine-translation-and-dataset.html
[25] Xuanming Zhang. Analysis of State of the Art Deep Learning based Techniques for Medical Natural Answer Generation. URL: https://www.billyzhang.me/uploads/billyzhang_thesis.pdf
[26] Deepika Singh. Natural Language Processing - Machine Learning with Text Data. URL:
[27] https://www.pluralsight.com/guides/nlp-machine-learning-text-data
[28] ModelCheckpoint. URL: https://keras.io/api/callbacks/model_checkpoint/.
[29] Callbacks API. URL: https://keras.io/api/callbacks/
[30] M. Garcarz, Legal Language Translation: Theory behind the Practice, CEUR Workshop Proceedings, Vol-3171 (2022) 2-2.
[31] N. Hrytsiv, I. Bekhta, M. Tkachivska, V. Byalyk, Sylvia Plath's I felt-Narrative Label of The Bell Jar in Ukrainian Translation, CEUR Workshop Proceedings, Vol-3171 (2022) 240-255.
[32] M. Bekhta-Hamanchuk, H. Oleksiv, T. Shestakevych, Y. Shyika, Quantitative Parameters of J. London's Short Stories Collection "Children of the Frost" and its Translation, CEUR Workshop Proceedings, Vol-3171 (2022) 697-710.
[33] K. S. Mandziy, U. V. Yurlova, M. P. Dilai, English-Ukrainian Parallel Corpus of IT Texts: Application in Translation Studies, CEUR Workshop Proceedings, Vol-3171 (2022) 724-736.
[34] N. Hrytsiv, T. Shestakevych, J. Shyyka, Corpus Technologies in Translation Studies: Fiction as Document, in: CEUR Workshop Proceedings, Vol-2917 (2021) 327-343.
[35] A. Lutskiv, R. Lutsyshyn, Corpus-Based Translation Automation of Adaptable Corpus Translation Module, CEUR Workshop Proceedings, Vol-2870 (2021) 511-527.
[36] V. Lytvyn, P. Pukach, V. Vysotska, M. Vovk, N. Kholodna, Identification and Correction of Grammatical Errors in Ukrainian Texts Based on Machine Learning Technology. Mathematics 11 (2023) 904. https://doi.org/10.3390/math11040904
[37] A. Kopp, D. Orlovskyi, S. Orekhov, An Approach and Software Prototype for Translation of Natural Language Business Rules into Database Structure, CEUR Workshop Proceedings 2870 (2021) 1274-1291.
[38] S. Kubinska, R. Holoshchuk, S. Holoshchuk, L. Chyrun, Ukrainian Language Chatbot for Sentiment Analysis and User Interests Recognition based on Data Mining, CEUR Workshop Proceedings, Vol-3171 (2022) 315-327.
[39] A. Dmytriv, S. Holoshchuk, L. Chyrun, R. Holoshchuk, Comparative Analysis of Using Different Parts of Speech in the Ukrainian Texts Based on Stylistic Approach, CEUR Workshop Proceedings, Vol-3171 (2022) 546-560.