Chuvash-Turkish Translation by Using Seq2seq Model

Mustafa KAYA
Department of Computer Engineering
Hacettepe University
mustafa.kaya@omu.edu.tr

Abrstarct

Machine translation is a sub-field of computational linguistics that investigates the use of software to translate text or speech from one language to another. Different methods have been tried to obtain better results in machine translation such as Transfer-Based Machine Translation, Interlingual Machine Translation, Statistical Machine Translation, Example-Based Machine Translation, Hybrid Machine Translation, Neural Machine Translation (NMT). However, these methods couldn't go beyond a certain advance except NMT. An important step has been taken in machine translation thanks to NMT. In this paper a neural machine translation system for Chuvash to Turkish is presented. It is free and open-source software. The system is built using the encoder-decoder framework.

1. Introduction

The idea of machine translation (MT) may be traced back to the 17th century. In 1629, Rene Descartes proposed a universal language, with equivalent ideas in different tongues sharing one symbol (Knowlson, 1975). The first researcher in the field, Yehosha Bar-Hillel, began his research at MIT (1951). A Georgetown University MT research team followed (1951) with a public demonstration of its Georgetown-IBM experiment system in 1954. MT research programs popped up in Japan and Russia (1955), and the first MT conference was held in London in 1956 (Gordin, 2015). In daily life, in trade, in diplomacy, in military, machine translation need a great need. Google 2016 also began to make machine translation using Deep Neural Networks (DNN) and became quite successful. Facebook also developed its own model in 2016. DNN are extremely powerful machine learning models that achieve excellent performance on difficult problems such as speech recognition (Dahl et al., 2012) and visual object recognition (Leun et al., 1998). DNNs are powerful because they can perform arbitrary parallel computation for a modest number of steps.

U.S. machine translation market size, by application, 2012 - 2022 (USD Million)

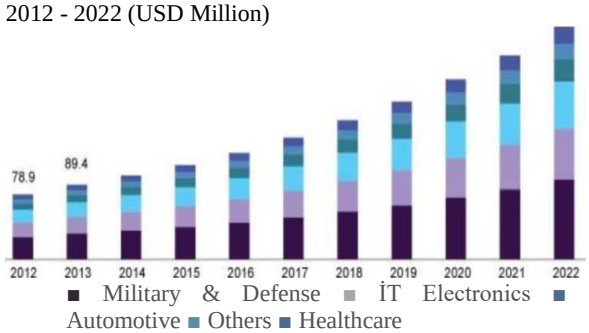

Figure 1: Machine Translation's current market size situation and forecast for the coming years (Grand View Research, 2018)

The global machine translation market size was valued at USD 433.0 million in 2016 and is expected to develop at a CAGR of 14.6% over the forecast period. Rising Internet penetration and increasing need to reduce operational costs are expected to drive market growth over the next few years. Soaring smartphone penetration and improvement in communication infrastructure have also positively contributed to industry growth.

Neural machine translation (NMT) is a newly emerging approach to machine translation, recently proposed by Kalchbrenner and Blunsom (2013). Unlike the traditional phrase-based translation system which consists of many small sub-components that are tuned separately, neural machine translation attempts to build and train a single, large neural network that reads a sentence and outputs a correct translation (Koehn et al., 2003). We call such architectures sequence to sequence (seq2seq). Seq2seq takes as input a sequence of words(sentence or sentences) and generates an output sequence of words. It does so by the use of more often long short-term memory (LSTM) or gated recurrent unit (GRU) to avoid the problem of vanishing gradient. It mainly has two components: encoder and decoder. The empirical evaluation reveals that this approach of scoring phrase pairs with an RNN Encoder-Decoder improves the translation performance. It was qualitatively analyzed the trained RNN Encoder-Decoder by comparing its phrase scores with those given by the existing translation model. It is trains the model to learn the translation probability of a source phrase to a corresponding target phrase. The empirical evaluation reveals that this approach of scoring phrase pairs with an RNN Encoder-Decoder improves the translation performance. It was qualitatively analyzed the trained RNN Encoder-Decoder by comparing its phrase scores with those given by the existing translation model. There are examples that have been carried out language translation by using this structure. However, translation studies among Turkish languages are not very common in the literature. This situation is the same for Chuvash, which has an important place among Turkish languages.

Overall, there are two contributions in this paper:

- I intend to present a Free / Open Source prototype seq2seq architecture based machine translation system between Chuvash and Turkish with this study.
- This study will be an infrastructure for future studies.

2. Languages

Turkish is the official language in Turkey and Cyprus. It is a minority language recognized in Greece, Iraq, Kosovo, Macedonia, and Romania. Mostly there are about 80 million living in Turkey speak Turkish fluently (Eberhard et al., 2019). The Turkish alphabet is a Latin-script alphabet used for writing the Turkish language, consisting of 29 letters, seven of which (Ç, Ş, Ğ, I, İ, Ö, Ü) have been modified from their Latin originals for the phonetic requirements of the language.

Chuvas is a Turkic language spoken in European Russia, primarily in the Chuvash Republic and adjacent areas. It is the only surviving member of the Oghuz branch of Turkic languages. The writing system for the Chuvash language is based on the Cyrillic script, employing all of the letters used in the Russian alphabet and adding four letters of its own: Ӑ, Ӗ, Ҫ and Ӳ. As characteristic of all Turkic languages, Chuvash is an agglutinative language and as such, has an abundance of suffixes but no native prefixes or prepositions, apart from the partly reduplicative intensive prefix, such as in: шурӑ - white, шап-шурӑ - snow-white, хура - black, хуп-хура - jet black.

According to the theory of Altaic languages (Turkish languages, Mongolian, Manchu-Tunguz, Korean, Japanese), Chuvash is a bridge that connects Turkish languages with other Altaic languages. That is why Chuvash is very important. A fairly significant production and publication of literature in Chuvash still continues. According to UNESCO's Index Translationum, at least 208 books translated from Chuvash were published in other languages (mostly Russian) since 1979 (Unesco, 2020).

2.1. Cyrillic

A Cyrillic alphabet based on that of Russian was used officially from 1938 to the 1990s, and has still not completely fallen out of use today. Unlike some of the other Turkic alphabets, it did not feature special characters that were not present in the Russian alphabet. Consonants and vowels that did not exist in Russian were instead written using digraphs, often involving the hard ъ or soft ь sign. For example, the consonants represented as q, ğ and ñ in the Latin script are represented as къ, гъ, and нъ, respectively, in the Cyrillic orthography Gökırmak et al. (2019). Also, the vowels represented with ü and ö in the Latin script are represented with either уь and оь, or y and o with a ь after the following consonant, or just y and o in the presence of certain consonants.

3. Related Works

Tantuğ et al. (2007), presented an approach to MT between Turkic languages and presented results from an implementation of an MT system from Turkmen to Turkish. Their approach relies on ambiguous lexical and morphological transfer augmented with target side rule-based repairs and rescoring with statistical language models. The results are quite positive but there is quite some room for improvement. Their work involves improving the quality of their current system as well as expanding this approach to Azerbaijani and Uyghur.

Bayatli et al. (2018), presented a shallow-transfer machine translation (MT) system for translating from Kazakh to Turkish. The system is based on the Apertium free/open-source machine translation platform. In the study, Linguistic components were developed, including a Kazakh-Turkish bilingual dictionary, Constraint Grammar disambiguation rules, lexical selection rules, and structural transfer rules. The created system beats the SMT systems on BLEU, while having a much higher out-of-vocabulary rate. The system is available as free/open-source software under the GNU GPL.

Gökırmak et al. (2019), presented a machine translation system for Crimean Tatar to Turkish. Their study is the first Machine Translation system made available for public use for Crimean Tatar, and the first such system released as free and open-source software. The system was built using Apertium, a free and open-source machine translation system, and is currently unidirectional from Crimean Tatar to Turkish. It has near-production level coverage, but is rather prototype-level in terms of the number of rules. Although the impact of this relatively low number of rules on the quality of translation is extensive, the outlook is promising and the current results suggest that a high-quality translation between morphologically-rich agglutinative languages is possible.

4. Method

Neural machine translation (NMT) is an approach to machine translation that uses a large artificial neural network to predict the likelihood of a sequence of words, typically modeling entire sentences in a single integrated model. The primary appeal of NMT lies in its ability to employ algorithms which learn linguistic rules on their own from the parallel corpus, thus making it conceptually simple and eliminating the need for complex feature engineering by providing end-to-end translation.

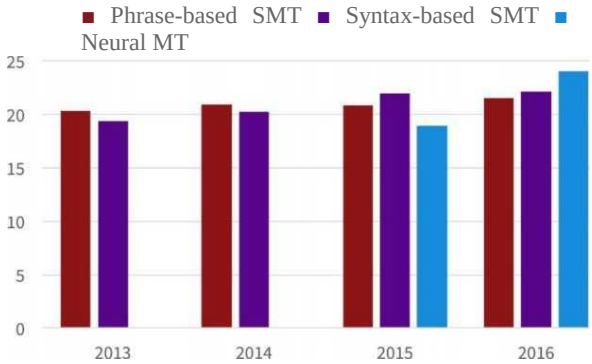

Figure 2: Progress in performance of Machine Translation in recent years (Agrawal, 2017)

Used in study seq2seq is a method of encoder-decoder based machine translation that maps an input of sequence to an output of sequence with a tag and attention value. The idea is to use 2 Recurrent Neural Network (RNN) that will work together with a special token and trying to predict the next state sequence from the previous sequence. The RNN is a natural generalization of feedforward neural networks to sequences. Given a sequence of inputs $(x_1,\ldots,x_T)$, a standard RNN computes a sequence of outputs $(y_1,\ldots,y_T)$ by iterating the following equation (Werbos, 1990):

$$h_t = \text{sigm}\left(W^{\text{hx}} x_t + W^{\text{hh}} h_{t-1}\right)$$
$$y_t = W^{\text{yh}} h_t$$

The RNN can easily map sequences to sequences whenever the alignment between the inputs the outputs is known ahead of time. However, it is not clear how to apply an RNN to problems whose input and the output sequences have different lengths with complicated and non-monotonic relationships (Sutskever et al., 2014).

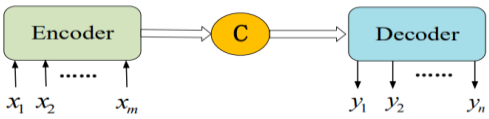

Figure 3: The structure of Seq2seq (Gong et al., 2019)

As shown in Figure 3, the Seq2seq codec consists of an encoder, an intermediate vector c , and a decoder. Codecs are typically multi-layer RNN or LSTM structures, wherein the intermediate vector c incorporates the sequence of $x_1, x_2, \ldots x_m$ encoding information. For time t , the output of the previous moment y_{t-1}, the hidden layer state of the previous moment s_{t-1}, and c are fed as input into the decoder. Finally, the decoder's hidden layer state s_t is obtained, which predicts the output value Gong et al., 2019). The entire process can be understood as passing Information in time steps to the encoder which generates a hidden state representation, passes it on to the decoder which then uses this information for prediction of the next correct step in the translation process. The goal of seq2seq can be described as designing a model which can be completely trained with the components being able to be fully tuned using training corpora to generate end-to-end translations.

5. Experiments

Seq2Seq is a method of encoder-decoder based machine translation that maps an input of sequence to an output of sequence with a tag and attention value. The idea is to use 2 RNN that will work together with a special token and trying to predict the next state sequence from the previous sequence.

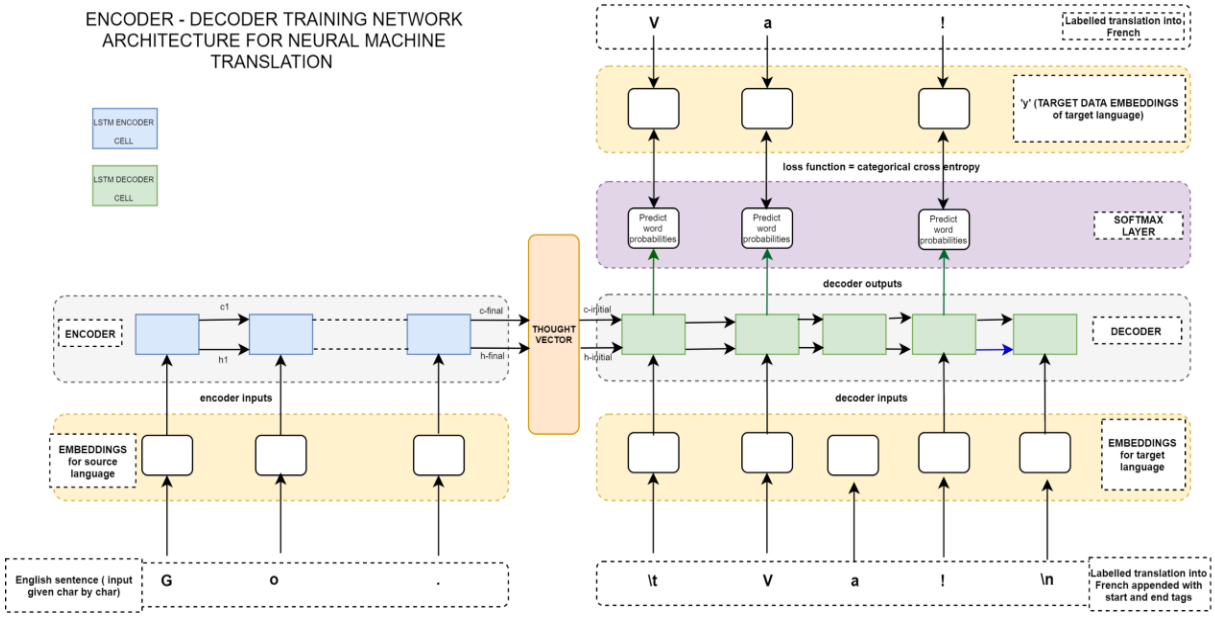

Figure 4: Encoder-Decoder training architecture for NMT (Kompella, 2018)

The Chuvash expressions were used in the encoder part, and the Turkish equivalents were used in the decoder part. The summary of the transactions is as follows:

Step 1) Prepare dataset

First, required libraries were called. Start and end token have been added to each sentence. The sentences have been cleaned by removing special characters. Create a word index and reverse word index have been created. Each sentence has been padded to a maximum length.

```
path_to_file = "ch_tr.txt"
def unicode_to_ascii(s):
  return ''.join(c for c in unicodedata.normalize('NFD', s)
      if unicodedata.category(c) != 'Mn')
def preprocess_sentence(w):
  w = unicode_to_ascii(w.lower().strip())
  w = re.sub(r"([?.!,¿])", r" \1 ", w)
  w = re.sub(r'[" "]+', " ", w)
  w = w.strip()
  w = '<start> ' + w + ' <end>'
  return w
```

```
def create_dataset(path, num_examples):
  lines = io.open(path, encoding='UTF-8').read().strip().split('\n')
  word_pairs = [[preprocess_sentence(w) for w in l.split('\t')]  for l
in lines[:num_examples]]
  return zip(*word_pairs)

def tokenize(lang):
  lang_tokenizer = tf.keras.preprocessing.text.Tokenizer(filters='')
  lang_tokenizer.fit_on_texts(lang)
  tensor = lang_tokenizer.texts_to_sequences(lang)
  tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor,
padding='post')
  return tensor, lang_tokenizer

def load_dataset(path, num_examples=None):
  targ_lang, inp_lang = create_dataset(path, num_examples)
  input_tensor, inp_lang_tokenizer = tokenize(inp_lang)
  target_tensor, targ_lang_tokenizer = tokenize(targ_lang)
  return input_tensor, target_tensor, inp_lang_tokenizer,
targ_lang_tokenizer
```

Figure 5: Loading data and creating English-Turkish word pairs

Step 2) Build the encoder and decoder model

The Embedding layer is a lookup table that stores the embedding of our input into a fixed-sized dictionary of words. It will be passed to a GRU layer. GRU layer consists of multiple layer type of RNN that will calculate the sequenced input. This layer will calculate the hidden state from the previous one and update the reset, update, and new gates.

```
class Encoder(tf.keras.Model):
  def __init__(self, vocab_size, embedding_dim,
enc_units, batch_sz):
    super(Encoder, self).__init__()
    self.batch_sz = batch_sz
    self.enc_units = enc_units
    self.embedding =
tf.keras.layers.Embedding(vocab_size,
embedding_dim)
    self.gru = tf.keras.layers.GRU(self.enc_units,
                                   return_sequences=True,
                                   return_state=True,
recurrent_initializer='glorot_uniform')

  def call(self, x, hidden):
    x = self.embedding(x)
    output, state = self.gru(x, initial_state = hidden)
    return output, state

  def initialize_hidden_state(self):
    return tf.zeros((self.batch_sz, self.enc_units))

class Decoder(tf.keras.Model):
  def __init__(self, vocab_size, embedding_dim,
dec_units, batch_sz):
    super(Decoder, self).__init__()
    self.batch_sz = batch_sz
    self.dec_units = dec_units
    self.embedding =
tf.keras.layers.Embedding(vocab_size,
embedding_dim)
    self.gru = tf.keras.layers.GRU(self.dec_units,
                                   return_sequences=True,
                                   return_state=True,

recurrent_initializer='glorot_uniform')
    self.fc = tf.keras.layers.Dense(vocab_size)

    self.attention = BahdanauAttention(self.dec_units)

  def call(self, x, hidden, enc_output):
    context_vector, attention_weights =
self.attention(hidden, enc_output)
    x = self.embedding(x)
    x = tf.concat([tf.expand_dims(context_vector, 1),
x], axis=-1)
    output, state = self.gru(x)
    output = tf.reshape(output, (-1, output.shape[2]))
    x = self.fc(output)

    return x, state, attention_weights
```

Figure 6: Create encoder and decoder

Step 3) Define the optimizer and the loss function

The result of the dense layer in the decoder produces one-hot vectors. In this way, the word to be produced according to the index of the highest value will be determined. In this case, the one-hot array should be in the words to be compared. However, in my work, I did not create words as a one-hot array. There are 2 solutions to make this comparison. All words will be translated into one-hot vectors. However, this will cause unnecessary memory usage. Another solution is to use sparse cross-entropy. Sparse cross-entropy converts integers into one-hot vectors in each iteration. Thus, there is no need for unnecessary memory usage. RMS optimizer is used because it gives much better results on RNNs.

```
optimizer = tf.keras.optimizers.Adam()
loss_object =
tf.keras.losses.SparseCategoricalCrossentropy(
   from_logits=True, reduction='none')

def loss_function(real, pred):
  mask = tf.math.logical_not(tf.math.equal(real, 0))
  loss_ = loss_object(real, pred)

  mask = tf.cast(mask, dtype=loss_.dtype)
  loss_ *= mask

  return tf.reduce_mean(loss_)
```

Figure 7: Define the optimizer and the loss function

Step 4) Training the Model

The training process is started with converting each pair of sentences into Tensors from their Lang index. My model will use Adam as the optimizer and sparse cross entropy function to calculate the losses. The training process begins with feeding the pair of a sentence to the model to predict the correct output. At each step, the output from the model will be calculated with the true words to find the losses and update the parameters. The final step is to calculate the gradients and apply it to the optimizer and backpropagate.

In training phase:

- The input words pass through the encoder which returns encoder output and the encoder hidden state.
- The encoder output, encoder hidden state and the decoder input (which is the start token) is passed to the decoder.
- The decoder returns the predictions and the decoder hidden state.
- The decoder hidden state is then passed back into the model and the predictions are used to calculate the loss.

- Use the teacher forcing to decide the next input to the decoder.
- Teacher forcing is the technique where the target word is passed as the next input to the decoder.

```
@tf.function
def train_step(inp, targ, enc_hidden):
  loss = 0
  with tf.GradientTape() as tape:
    enc_output, enc_hidden = encoder(inp, enc_hidden)
    dec_hidden = enc_hidden
    dec_input =
tf.expand_dims([targ_lang.word_index['<start>']] *
BATCH_SIZE, 1)
    for t in range(1, targ.shape[1]):

      predictions, dec_hidden, _ = decoder(dec_input,
dec_hidden, enc_output)

      loss += loss_function(targ[:, t], predictions)
      dec_input = tf.expand_dims(targ[:, t], 1)
  batch_loss = (loss / int(targ.shape[1]))

  variables = encoder.trainable_variables +
decoder.trainable_variables
  gradients = tape.gradient(loss, variables)
  optimizer.apply_gradients(zip(gradients, variables))
  return batch_loss

EPOCHS = 150
for epoch in range(EPOCHS):
  start = time.time()

  enc_hidden = encoder.initialize_hidden_state()
  total_loss = 0
  for (batch, (inp, targ)) in
enumerate(dataset.take(steps_per_epoch)):
    batch_loss = train_step(inp, targ, enc_hidden)
    total_loss += batch_loss

    if batch % 100 == 0:
      print('Epoch {} Batch {} Loss
{:.4f}'.format(epoch + 1,
                                  batch,
                                  batch_loss.numpy()))
   if (epoch + 1) % 2 == 0:
    checkpoint.save(file_prefix = checkpoint_prefix)
  print('Epoch {} Loss {:.4f}'.format(epoch + 1,
                            total_loss / steps_per_epoch))
  print('Time taken for 1 epoch {}
sec\n'.format(time.time() - start))
```

Figure 8: Training

6. Results

A translation program was made using the 2111 expression. The Chuvash and Turkish data in the same file were transferred to different categories. The program created consists of 2 RNN structure. The first is the encoder, the second is the decoder. Input sentence enters encoder. A thought vector is produced here. This vector is transferred to the decoder. Decoder also produces an output. All of this method is called seq2seq.

The epoch was set to 150 and batch size was set to 128 for the study. Thus, more efficient results are obtained. Sample translations made at the end of the study as follows:

```
translate('Сук шим манăн ăрăскалă')
```

```
Input: <start> сук шим манан араскала <end>
Predicted translation: bir adam kosturuyor <end>
```

Figure 9: Translation example (translate sentence from source)

```
translate('Пуш Килчĕ')
```

```
Input: <start> пуш килче <end>
Predicted translation: mart geldi
```

Figure 10: Translation example (translate sentence from entered outside)

The program correctly translates the data in the source file. Translating short sentences that are not in the dataset, is also successful. As the number of words in the sentence increases, translations become distorted.

7. Conclusion

The proposed model has created an infrastructure for Chuvash-Turkish machine translation. Although the impact of this relatively low number of rules on the quality of translation is extensive, the outlook is promising and the current results suggest that a high-quality translation between morphologically-rich agglutinative languages is possible. Moreover, the fact that Turkish is an added language makes translation a little difficult. However, by this method using a small dataset is made very successful translations. It is seen that a more advanced dataset will have more successful results. I plan to continue to develop this work in the future. I will want to expand the grammar rules and dataset and so i will improve the quality of the translation. The whole system may be downloaded from GitHub.

8. References

Agrawal, R., 2017. Towards efficient Neural Machine Translation for Indian Languages. nternational Institute of Information Technology, Hyderabad, India, Master of Science in Computer Science and Engineering.

Bayatli, S., Kurnaz, S., Salimzianov, I., Washington, J. N. and Tyers, F., M., 2018. Rule-based machine translation from Kazakh to Turkish. Proceedings of the 21st Annual Conference of the European Association for Machine Translation: 28-30 May 2018, Universitat d'Alacant, Alacant, Spain, pp. 49-58.

Dahl, G., E., Yu, D., Deng, L. and Acero, A., 2012. Context-dependent pre-trained deep neural networks for large vocabulary speech recognition. IEEE Transactions on Audio, Speech, and Language Processing - Special Issue on Deep Learning for Speech and Language Processing.

Durgar-El Kahlout ’I, Mermer C, Dogan MU (2012) Recent improvements in statistical machine “ translation between Turkish and English. In: Vertan C, von Hahn W (eds) Multilingual processing in Eastern and Southern EU languages: low-resourced technologies and translation. Cambridge Scholars Publishing, Cambridge.

Google, 2016. A Neural Network for Machine Translation, at Production Scale. https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html

Gong, G., An, X., Kumar Mahato, N. , Sun, S., Chen, S. and Wen, Y. 2019. Research on Short-Term Load Prediction Based on Seq2seq Model. Energies 2019, 12, 3199; doi:10.3390/en12163199.

Gordin, Michael D., 2015. *Scientific Babel: How Science Was Done Before and After Global English.*

Chicago, Illinois: University of Chicago Press. ISBN 9780226000299.

Gökırmak, M., Tyers, F. and Washington, J., 2019. Machine Translation for Crimean Tatar to Turkish. Proceedings of the 2nd Workshop on Technologies for MT of Low Resource Languages, Dublin, Ireland, Pages:24–31.

Grand View Research, 2018. https://www.grandviewresearch.com/industry-analysis/machine- translation-market.

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE.

Lo, A., Dione, C.,B., Nguer, E., M., Ba, S.,O. and Lo, M., 2020. Using LSTM Networks To Translate French To Senegalese Local Languages: Wolof As A Case Study. ICLR AfricaNLP2020 Workshop, Addis Ababa, Ethiopia.

Kalchbrenner, N. and Blunsom, P., 2013. Recurrent continuous translation models. In Proceedings of the ACL Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1700-1709. Association for Computational Linguistics.

Knowlson, J., 1975. Universal language schemes in England and France 1600-180. University of Toronto Press, Scholarly Publishing Division, pp.70.

Koehn, P., Och, F. J., and Marcu, D., 2003. Statistical phrase-based translation. In Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1, NAACL '03, pages 48-54, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kompella, R., 2018. Neural Machine Translation — Using seq2seq with Keras https://towardsdatascience.com/neural-machine-translation-using-seq2seq-with-keras c23540453c74.

Sennrich, R., Haddow, B. and Alexandra Birch, A., 2016. Improving Neural Machine Translation Models with Monolingual Data. https://arxiv.org/abs/1511.06709v4.

Sutskever, I., Oriol Vinyals, O. and Le, Q., 2014. Sequence to Sequence Learning with Neural Networks. Advances in Neural Information Processing Systems 27.

Tantuğ, A., C., Adalı, E., Oflazer, K., O., 2007. Machine Translation between Turkic Languages. Proceedings of the ACL 2007 Demo and Poster Sessions, pages 189 192, Prague, June 2007. c 2007 Association for Computational Linguistics.

Unesco, 2020. Index Translationum. http://www.unesco.org/xtrans/bsresult.aspx?a=&stxt=&sl=chv&l=&pla=&pub=&tr=&e=&udc=&d=&from=&to=&tie=a

Werbos, p., 1990. Backpropagation through time: what it does and how to do it. Proceedings of IEEE, 1990.