Language Technology Assignment: Neural Machine Translation

Taskoudis Dimitris
Aristotle University of Thessaloniki
Thessaloniki, Greece
taskoudis@csd.auth.gr

Fragkouli Styliani Christina
Aristotle University of Thessaloniki
Thessaloniki, Greece
sfragkoul@csd.auth.gr

***Abstract*—In this assginment the machine translation problem was addressed. From the field of Deep Learning we used a reccurent neural network, specifically an encoder - decoder model to translate given phrases from German to English. Our work splits into two main parts. In the theory section the neseccery theoretical background is looked upon giving inforation about the techniques and concepts that were used. In the code section we discuss the initial model, its architecture and its results after training and evaluation. Furthermore, some architecture alterations were performed and are discussed in the final section.**

***Index Terms*—machine, translation, LSTM, encoder, decoder**

I. Theory

According to [1] one of the earliest goals for computers was the automatic translation of text from one language to another. It envolves the convergence of a sequence of symbols from one language to another. But the process of trasnaltion is not easy since it envolves a variaty of factors and rules that are often developed by linguists and may operate at the lexical, syntactic, or semantic level.
Statistical machine translation, or SMT for short, is the use of statistical models that learn to translate text from a source language to a target language gives a large corpus of examples. As quoted by [2] "Given a sentence T in the target language, we seek the sentence S from which the translator produced T. We know that our chance of error is minimized by choosing that sentence S that is most probable given T. Thus, we wish to choose S so as to maximize Pr(S|T)."
The approach is data-driven, requiring only a corpus of examples with both source and target language text. This means linguists are not longer required to specify the rules of translation. Although effective, statistical machine translation methods suffered from a narrow focus on the phrases being translated, losing the broader nature of the target text.

A. Neural Machine Translation

Neural machine translation, or NMT for short, is the use of neural network models to learn a statistical model for machine translation. The key benefit to the approach is that a single system can be trained directly on source and target text, no longer requiring the pipeline of specialized systems used in statistical machine learning. The term end-to-end system applies to a single model, in our case a neural network, that will have a sentence as input and produce another sentence as its corresponding translation.

B. Word Embeddings

According to [3] word embeddings are a type of word representation that allows words with similar meaning to have a similar representation and is perhaps one of the key breakthroughs for the impressive performance of deep learning methods on challenging natural language processing problems.
Word embeddings are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space.
Each word is mapped to one vector which is often tens or hundreds of dimensions. This is contrasted to the thousands or millions of dimensions required for sparse word representations, such as a one-hot encoding.
The representation rely on the idea that words used in similar ways will have similar representations as well. There is deeper linguistic theory behind the approach, namely the "distributional hypothesis" by Zellig Harris that could be summarized as: "words that have similar context will have similar meanings."

C. Sequence-to-sequence predictions with LSTM RNNs

Sequence-to-sequence learning (Seq2Seq) is about training models to convert sequences from one domain (e.g. sentences in German) to sequences in another domain (e.g. the same sentences translated to English). The canonical sequence-to-sequence case is when input and output sequences have different lengths and the entire input sequence is required in order to start predicting the target.
This type of problem is addressed with an RNN architecture containing a layer acting as the encoder which processes the input sequence and returns its own internal state and a layer acting as the endocer trained to predict the next characters of the target sequence, given previous characters of the target sequence. Specifically, it is trained to turn the target sequences into the same sequences but offset by one timestep in the future, a training process called "teacher forcing" in this context. Importantly, the encoder uses as initial state the state vectors from the encoder, which is how the decoder obtains information about what it is supposed to generate. Effectively, the decoder learns to generate targets$[t + 1...]$

given targets$[...t]$, conditioned on the input sequence.
Key to the encoder-decoder architecture is the ability of the model to encode the source text into an internal fixed-length representation called the context vector. Interestingly, once encoded, different decoding systems could be used, in principle, to translate the context into different languages [1]. Quoting [4] "an encoder neural network reads and encodes a source sentence into a fixed-length vector. A decoder then outputs a translation from the encoded vector. The whole encoder–decoder system, which consists of the encoder and the decoder for a language pair, is jointly trained to maximize the probability of a correct translation given a source sentence".

Now it is intentional to dive into more technical details following [6]. Generally, deeper networks are believed to achieve better performance than shallow networks. The key is to find a balance between network depth, model skill, and training time. This is because we generally do not have infinite resources to train very deep networks if the benefit to skill is minor.
When it comes to encoders, it was found that depth did not have a dramatic impact on skill and more surprisingly, a 1-layer unidirectional model performs only slightly worse than a 4-layer unidirectional configuration. A two-layer bidirectional encoder performed slightly better than other configurations tested.
A similar story was seen when it came to decoders. The skill between decoders with 1, 2, and 4 layers was different by a small amount where a 4-layer decoder was slightly better. An 8-layer decoder did not converge under the test conditions.

D. BLEU Score

According to [7] BLEU, or the Bilingual Evaluation Understudy, is a score for comparing a candidate translation of text to one or more reference translations. A perfect match results in a score of 1.0, whereas a perfect mismatch results in a score of 0.0 and it offers five compelling benefits: it is quick and inexpensive to calculate, it is easy to understand, it is language independent, it correlates highly with human evaluation and it has been widely adopted.
The approach works by counting matching n-grams in the candidate translation to n-grams in the reference text, where 1-gram or unigram would be each token and a bigram comparison would be each word pair. The comparison is made regardless of word order. Depending on which $n - grams$ were used the corresponding score is also $BLUE - n$

There are a variaty of BLEU scores, in this assignment the $Corpus\ BLEU\ Score$ was used for calculating the BLEU score for multiple sentences such as a paragraph or a document. The references must be specified as a list of documents where each document is a list of references and each alternative reference is a list of tokens, e.g. a list of lists of tokens. The candidate documents must be specified as a list where each document is a list of tokens, e.g. a list of lists of tokens.

As pointed out by [8] one of the problems with this type of score is that it does not measure meaning. It only rewards systems for n-grams that have exact matches in the reference system. That means that a difference in a function word (like "an" or "on") is penalized as heavily as a difference in a more important content word. It also means that a translation that had a perfectly valid synonym that just didn't happen to show up in the reference translation will be penalized.
Also, BLEU doesn't handle morphologically-rich languages well If, like the majority of people on Earth, you happen to use a language other than English, you may have already spotted a problem with this metric: it's based on word-level matches. For languages with a lot of morphological richness that quickly becomes a problem.

II. Code

In this section we will look at the methods used to implement a mechanical translation algorithm in the data mentioned following [5]. The process of implementing the code is divided into three stages: pre-processing, create the neural machine translation, testing - model evaluation. The creation of the code took place in the Google-Colaboratory extension with GPU = 1 and with the help of the library Keras.

A. Pre-Processing

The data refer to the translation of sentences - words from German to English. After downloading our data, decompression was performed so that they can be used later. The downloaded data file includes a balance of phrases between the two languages, separated by tab-character.
The first step in implementing the code is to clean the data from its original form, because it includes punctuation contains uppercase and lowercase, there are duplicate phrases in English with different translations in German and there are special characters in the German that do not contribute to the purpose of the translation.
We first loaded the data in a way that retains the German Unicode characters using the load-doc function.
The text was then divided by line and then into a sentence using the to-pair function. The editing of each sentence separately with the help of the function clean-pears, includes the following: remove all non-printable characters, remove all punctuation characters, normalize all Unicode characters in ASCII, normalize the case in lower case letters, remove any they are not alphabetical. Finally, we use the function save-clean-data() that uses the pickle API to save the list of clean text to file.
The final format of the cleaned data includes two columns with phrases and words in English and German

Clear data contains more than 150,000 pairs of phrases and some of the pairs at the end of the file are very large. So to simplify the problem we reduced the data set to the first 10,000 copies in the file and mixed the samples for higher performance of the model (with the command shuffle). These

were the shortest phrases in the data set. Finally, the three data files that will be used later were saved.

B. Create the neural machine translation

In this subsection we will analyze the process of implementing the mechanical translation model in the data we processed. First, loaded the training and testing data that we created and edited in the previous step. We used the class to match words in integers, as required for modeling, and a separate tokenizer was also applied to English and German sequences. The length of the largest sequence was found through a list of phrases using the function max-length. For this reason, used were the two functions mentioned with the combined data set used to prepare tokenizers, vocabulary sizes and maximum lengths for both English and German phrases.
Subsequently, we encode each input and output sequence into integers, because the model does not recognize and cannot process words and phrases, but only numbers. A table was created for each sequence and zero values were added to the sequences that were smaller in size max-length. We also did this process because we used a built-in word for input sequences and output output encoding. All this is done using the encoding-sequence function. Next, we coded the output sequence because the model should predict the probability of each word in the vocabulary as an output. This process is performed by the function encode-output.
Using the functions we mentioned at the stage of creation the model, we prepared the training and testing data.

The next phase after preparing the data for training and testing concerns the creation of the model that we will use for translation. We used an LSTM encoder-decoder model in the data that we processed. The model structure includes two LSTM layers, one Embedding, one RepeatVector and one Dense layer as shown in Fig.1. The embedding layer turns positive integers (indexes) into dense vectors of fixed size and can only be used as the first layer in a model, the parameter: $mask\_zero$ is a Boolean, whether or not the input value 0 is a special ”padding” value that should be masked out. If $mask\_zero$ is set to True, as a consequence, index 0 cannot be used in the vocabulary. The repeat vector layer repeats the input n times. For time distributed layer, this wrapper allows to apply a layer to every temporal slice of an input. In this case a dense layer is applied to every sample it receives as an input in our case the size of the English vocabulary size. We have the last LSTM output a value for each time step in the input data by setting the $return\_sequences = True$ argument on the layer. This allows us to have 3D output from LSTM layer as input to the next one.In this architecture, the input sequence is encoded by a front-end model called the encoder then decoded word by word by a backend model called the decoder. The model is trained using the efficient Adam approach to stochastic gradient descent and minimizes the categorical loss function because we have framed the prediction problem as multi-class classification. The whole process of implementing the model is done through a function called ”define-model” and which take a number of arguments used to configure the model, such as the size of the input and output vocabularies, the maximum length of input and output phrases, and the number of memory units used to configure the model. The options for the parameters used in the training process are as follows: epochs = 64, batch-size = 64.
With the potentiality GPU, the training lasts very little time as each epochs ended in 5 seconds. The trained model was saved under the name ”model.h5”.

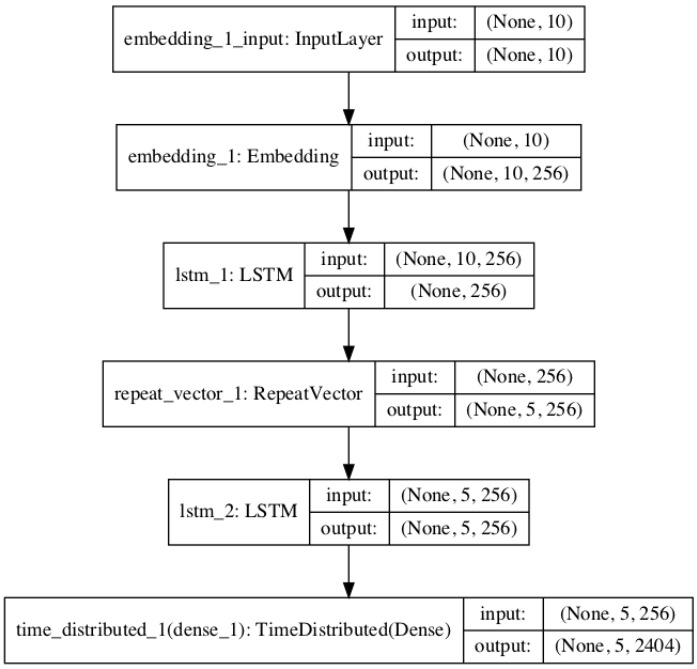

Fig. 1. Initial Model architecture

C. Model evaluation

In the last step, we did evaluate the trained model in train and test data. The evaluation includes two steps: first we created a translated output sequence and then we repeat this process for many input examples and summarized the model's ability in many cases. Starting with inference, the model can predict the entire output sequence in a one-shot manner. This will be a sequence of integers that we can enumerate and lookup in the tokenizer to map back to words. This process is performed using the function ”word-for-id”. This will be a sequence of integers that we enumerate and lookup in the tokenizer to map back to words. Therefore, we use the function ”predict-sequence” that performs this operation for a single encoded source phrase. Next, we repeat this for each source phrase in a dataset and compare the predicted result to the expected target phrase in English.
Apart from the BLEU-n scores we also used the validation loss function to have a better view of the model's performance. The results are presented in Fig.2 and Fig.3. The training did a lot better than the testing, which means that our model

learned the training set too well but then could not make sufficient translations for the test set. One thing that can be observed in both situations though is that the evaluation was better when working with unigrams rather than the rest n-grams. The training loss curve is smooth and becomes almost flat after about 40 epochs. This is a sign that our model is indeed learning from the training dataset. As far as validation is concerned, around 20 epochs a minimum is present, after which the trend went up so this is a sign of over fitting.

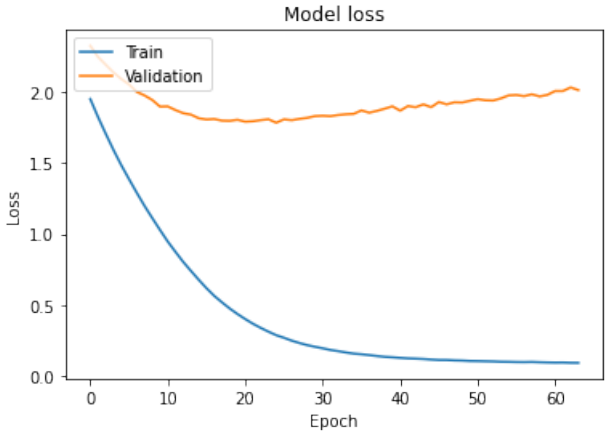

Fig. 2. Initial model loss function

Training Evaluation	
BLEU-1	0.92
BLEU-2	0.88
BLEU-3	0.79
BLEU-4	0.46
Testing Evaluation	
BLEU-1	0.56
BLEU-2	0.44
BLEU-3	0.36
BLEU-4	0.16

Fig. 3. Initial Model BLEU scores

III. Architecture Alterations

Then different architectures were tried out, focusing on spesific parameters in the aready existing architecture and also different types of recurrent layers and combinations of layers. Two modefications worth mentioning are the follwing. Firstly, the use of the *dropout* parameter set to 30 percent in the LSTM layers which randomly drops a set of the output keeping the model away from over fitting. As seen in the results in Fig.name and Fig.name a decrease in the tendency to over fit is indeed clearly observed and better BLEU scores are achieved.

The idea of Bidirectional Recurrent Neural Networks (RNNs) is straightforward. It involves duplicating the first recurrent layer in the network so that there are now two layers side-by-side, then providing the input sequence as-is as input to the first layer and providing a reversed copy of the input sequence to the second. “The idea is to split the state neurons of a regular RNN in a part that is responsible for the positive time direction (forward states) and a part for the negative time direction (backward states)”. The use of providing the

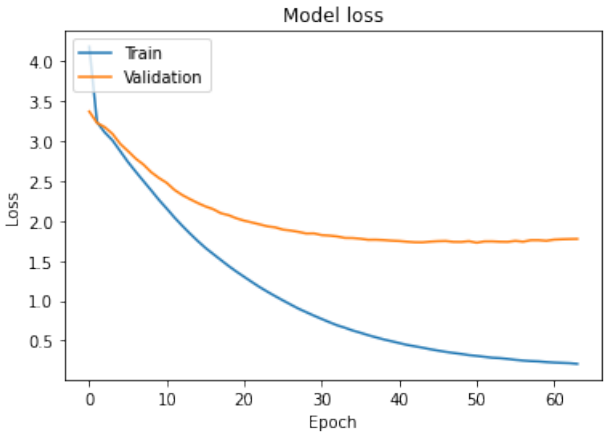

Fig. 4. Model loss function with dropout set to 30 percent

Training Evaluation	
BLEU-1	0.93
BLEU-2	0.90
BLEU-3	0.81
BLEU-4	0.48
Testing Evaluation	
BLEU-1	0.58
BLEU-2	0.46
BLEU-3	0.38
BLEU-4	0.19

Fig. 5. Model BLEU scores with dropout set to 30 percent

sequence bi-directionally was initially justified in the domain of speech recognition because there is evidence that the context of the whole utterance is used to interpret what is being said rather than a linear interpretation. So the next alteration that should be stressed is changing both endocer and decoder to Bidirectional LSTM layers. Different merging modes were tried out but here the concatenate method is presented.

The best run results were obtained when using Bidirectional LSTMs with concatenation merging for both the encoder and the encoder. Overall the bidirectional type did not really contributed to better results. From the output of our model in Fig.6it can be seen that the first LSTM input has double as explained by theory and the total number of parameters goes up.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 9, 256)            902656
_________________________________________________________________
bidirectional_2 (Bidirection (None, 512)               1050624
_________________________________________________________________
repeat_vector_2 (RepeatVecto (None, 5, 512)            0
_________________________________________________________________
bidirectional_3 (Bidirection (None, 5, 512)            1574912
_________________________________________________________________
time_distributed_2 (TimeDist (None, 5, 2214)           1135782
=================================================================
Total params: 4,663,974
Trainable params: 4,663,974
Non-trainable params: 0
```

Fig. 6. New architecture with bidirectional LSTM layers

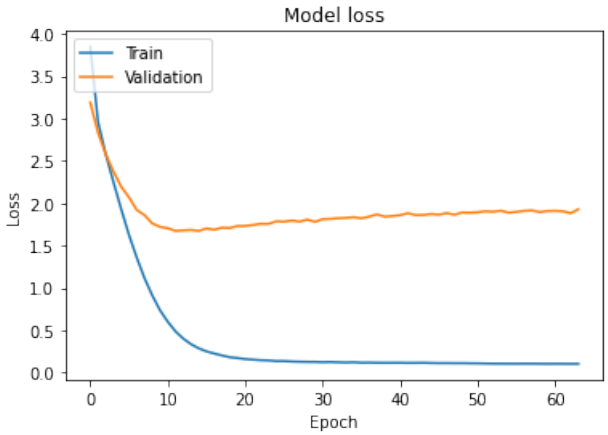

Fig. 7. Model loss function with bidirectional LSTMs

Training Evaluation	
BLEU-1	0.92
BLEU-2	0.88
BLEU-3	0.80
BLEU-4	0.47
Testing Evaluation	
BLEU-1	0.60
BLEU-2	0.47
BLEU-3	0.38
BLEU-4	0.18

Fig. 8. Model BLEU scores with bidirectional LSTMs

REFERENCES

[1] Brownlee,Jason, "A Gentle Introduction to Neural Machine Translation", https://machinelearningmastery.com/introduction-neural-machine-translation//

[2] Brown, Peter F. and Cocke, John and Della Pietra, Stephen A. and Della Pietra, Vincent J. and Jelinek, Fredrick and Lafferty, John D. and Mercer, Robert L. and Roossin, Paul S., "A Statistical Approach to Machine Translation", Computational Linguistics,1990

[3] Brownlee,Jason, "What Are Word Embeddings for Text?", https://machinelearningmastery.com/what-are-word-embeddings/

[4] Dzmitry Bahdanau and Kyunghyun Cho and Yoshua Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate", 2014

[5] Brownlee,Jason, "How to Develop a Neural Machine Translation System from Scratch", https://machinelearningmastery.com/develop-neural-machine-translation-system-keras.

[6] Brownlee,Jason, "How to Configure an Encoder-Decoder Model for Neural Machine Translation", https://machinelearningmastery.com/configure-encoder-decoder-model-neural-machine-translation/

[7] Brownlee,Jason, "A Gentle Introduction to Calculating the BLEU Score for Text in Python", https://machinelearningmastery.com/calculate-bleu-score-for-text-python/

[8] Tatman, Rchael, "Evaluating Text Output in NLP: BLEU at your own risk", https://towardsdatascience.com/evaluating-text-output-in-nlp-bleu-at-your-own-risk-e8609665a213