Princepreet S Bhatti

Machine Translation – Final Project Report

Group number: 29 - Princepreet S Bhatti | Pranav Goswami

STATUS: ***Working*** and translating German to English *(For small sentences up to 4-5 words)*

CODE available at: github.com/singh0777/LSTM-NMT

Preface

The machine learning model implemented in the project is an end-to-end approach for translating German to English using LSTM network. Deep neural networks are powerful models that show excellent results on difficult learning task and these networks learn best on large labeled dataset. Although for this project the data set required to be relatively small (~ 30K Sentences) so that it can be trained in an acceptable time limit (~ 12-20 hrs). In this report, you will see how we Implemented the Encoder-Decoder model with LSTM. We will talk about the challenges and analyze the results obtained.

The process of training and computation is very expensive and time consuming that's why we have taken some key assumptions on sentence lengths and the number for epoch for the model. This assumption helped us in training the model in batches *(using python's pickle serialization)* and give us working results.

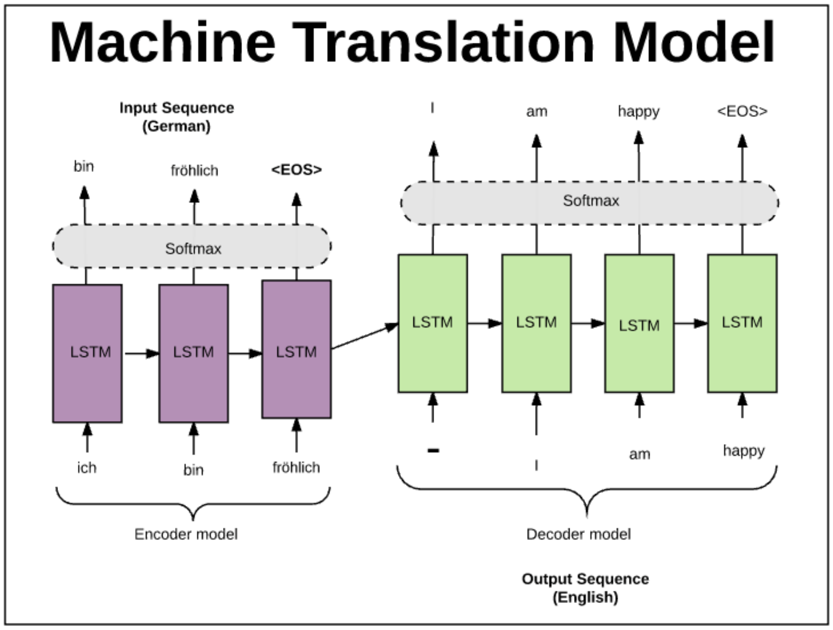


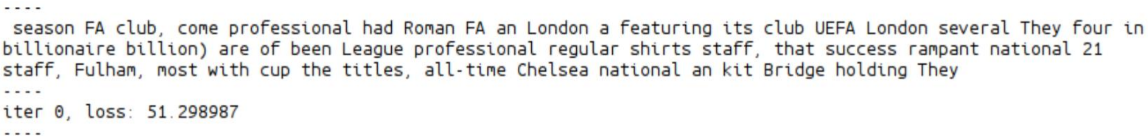
Figure 1: Encoder learns German grammar and passes the information to decoder, which in turn translates the information to English while maintaining correct English grammar

Project Progression (with outputs): We started the project by first understanding of the underlying encoder-decoder structure of a Sequence-2-Sequence learning problem. To build the encoder-decoder structure we started off by using the Recurrent Neural Networks as the basic unit.

1. **Learning Grammar using RNN:** **Next word Prediction** - The implementation of Question-3 in Homework-2 helped us a lot in learning the backpropagation in RNN, which we used to implement the next word prediction code. This is very important part of machine translation as, this step helps the RNN to learn the grammar of language. For example, *the RNN unit automatically learns*

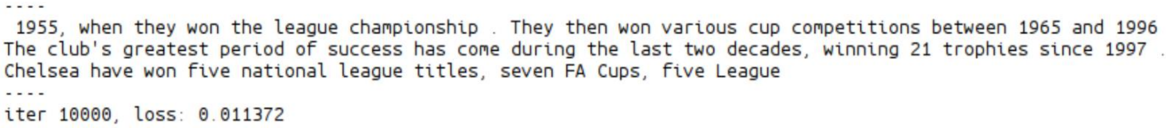
by itself that for a given word- "We", the probability of occurrence of word -'are' is more than the word - "is". Following outputs are observed when running the RNN on a small dataset.

Grammar observed at 0th Iteration: words are jumbled and no meaning conveyed by sentence



```
----
 season FA club, come professional had Roman FA an London a featuring its club UEFA London several They four in
billionaire billion) are of been League professional regular shirts staff, that success rampant national 21
staff, Fulham, most with cup the titles, all-time Chelsea national an kit Bridge holding They
----
iter 0, loss: 51.298987
----
```

Grammar Observed at 10000 iteration: words are ordered in proper grammar

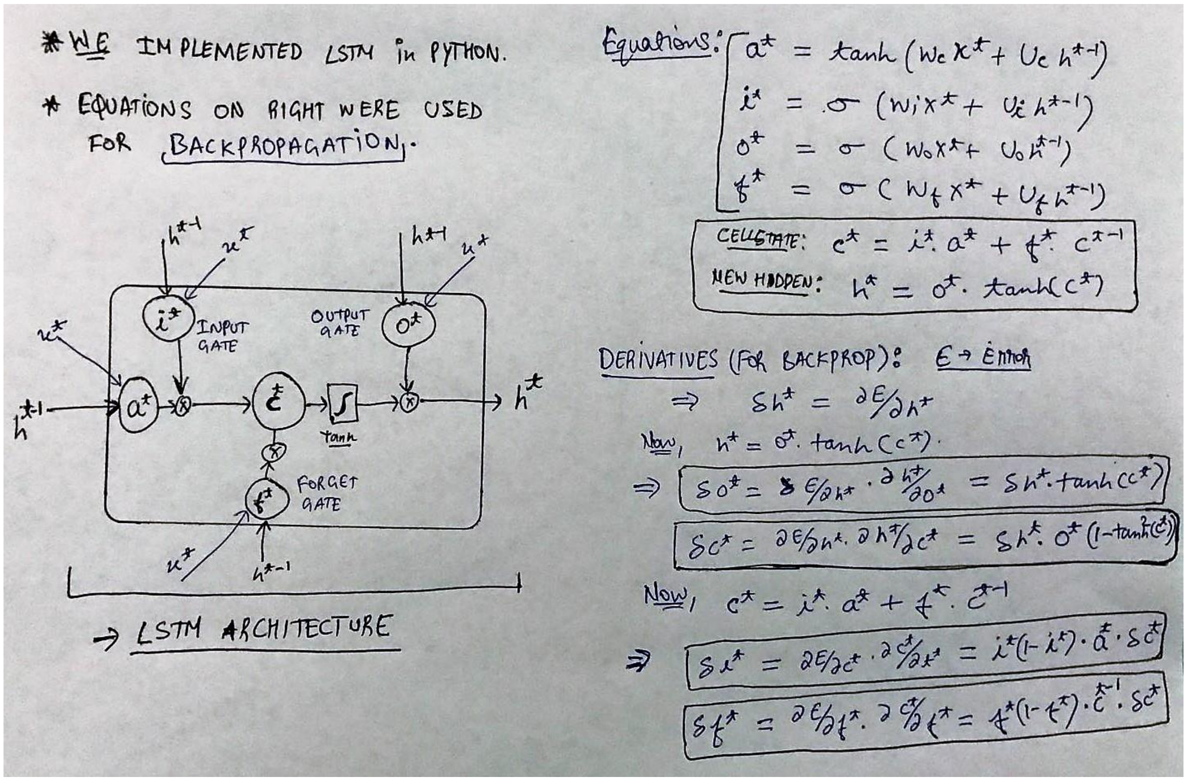


```
----
 1955, when they won the league championship . They then won various cup competitions between 1965 and 1996
The club's greatest period of success has come during the last two decades, winning 21 trophies since 1997 .
Chelsea have won five national league titles, seven FA Cups, five League
----
iter 10000, loss: 0.011372
```

2. **RNN based Encoder-Decoder Structure:** No Visible translation in this case - After successful implementation of encoder, we connected two different RNN's to form a structure as shown in fig 1. The basic idea behind the following being that the encoder encodes the information of input sentence in the form of a vector that acts as the initial hidden stage of the decoder RNN. Decoder uses this hidden stage information to correctly give an output. However, problem of **vanishing gradients** resulted in almost no accuracy in case of RNN.

3. **LSTM Implementation in Python:** Moved to LSTM **-** To resolve the problem of vanishing gradients in RNN, we decided to change our basic unit to LSTM. LSTM can avoid the problem of vanishing/exploding gradient through a forget gate that allows the LSTM structure to remove the irrelevant data from the previous timestep.

4. **BPTT - LSTM:** Implemented in Python - BPTT in the case of LSTM is lot more complex than RNN. We must backpropagate the error from the SoftMax layer to all the individual gates in LSTM. Following are equations derived for each of the gates and SoftMax layer.



* WE IMPLEMENTED LSTM in PYTHON.

* EQUATIONS ON RIGHT WERE USED FOR BACKPROPAGATION.

→ LSTM ARCHITECTURE

Equations:

$$\begin{cases} a^t = \tanh(W_c x^t + U_c h^{t-1}) \\ i^t = \sigma(W_i x^t + U_i h^{t-1}) \\ o^t = \sigma(W_o x^t + U_o h^{t-1}) \\ f^t = \sigma(W_f x^t + U_f h^{t-1}) \end{cases}$$

CELLSTATE: $c^t = i^t \cdot a^t + f^t \cdot c^{t-1}$

NEW HIDDEN: $h^t = o^t \cdot \tanh(c^t)$

DERIVATIVES (FOR BACKPROP): $E \rightarrow$ Error

$\Rightarrow \quad \delta h^t = \partial E / \partial h^t$

Now, $h^t = o^t \cdot \tanh(c^t)$.

$\Rightarrow \boxed{\delta o^t = \partial E/\partial h^t \cdot \partial h^t / \partial o^t = \delta h^t \cdot \tanh(c^t)}$

$\boxed{\delta c^t = \partial E/\partial h^t \cdot \partial h^t / \partial c^t = \delta h^t \cdot o^t (1 - \tanh^2(c^t))}$

Now, $c^t = i^t \cdot a^t + f^t \cdot c^{t-1}$

$\Rightarrow \boxed{\delta i^t = \partial E/\partial c^t \cdot \partial c^t / \partial i^t = i^t(1-i^t) \cdot a^t \cdot \delta c^t}$

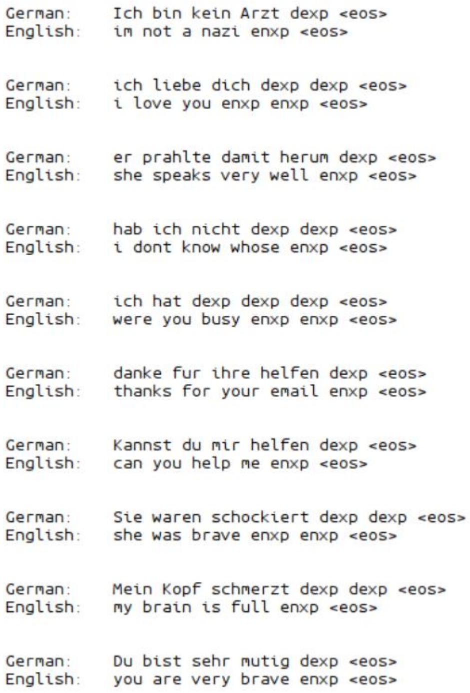
$\boxed{\delta f^t = \partial E/\partial f^t \cdot \partial c^t / \partial f^t = f^t(1-f^t) \cdot c^{t-1} \cdot \delta c^t}$

5. **LSTM based Encoder-Decoder:** **Final Model** - To test our implementation of LSTM, we replaced the RNN with LSTM in our code for prediction of next word. The error could converge faster in the case of LSTM. After successfully implementation of LSTM, we implemented the code for encoder and decoder LSTM's. Our final model looks as shown in fig 1.

MODEL OUTPUTS:

Input	*German*
Output	*English*
Training	*30 epochs*
Training Time	*18 hours*

GERMAN INPUT	Translated	Expected
Ich liebe dich	I love you	I love you
Bitte fahre vorsichtig	Please be carefull	Please drive carefully
Danke fur ihre helfen	Thanks for your email	Thanks for your help
Kannst du mir helfen	Can you help me	Can you help me
Du bist sehr mutig	You are very brave	You are very brave
Ich bin kein Arzt	im not a nazi	I am not a doctor
Mein kopf schmerzt	My brain is full	My head hurts
Sie waren schockiert	She was brave	She was shocked
Er prahlt damit herum	She speaks very well	He bragged about it



```
German:    Ich bin kein Arzt dexp <eos>
English:   im not a nazi enxp <eos>

German:    ich liebe dich dexp dexp <eos>
English:   i love you enxp enxp <eos>

German:    er prahlte damit herum dexp <eos>
English:   she speaks very well enxp <eos>

German:    hab ich nicht dexp dexp <eos>
English:   i dont know whose enxp <eos>

German:    ich hat dexp dexp dexp <eos>
English:   were you busy enxp enxp <eos>

German:    danke fur ihre helfen dexp <eos>
English:   thanks for your email enxp <eos>

German:    Kannst du mir helfen dexp <eos>
English:   can you help me enxp <eos>

German:    Sie waren schockiert dexp dexp <eos>
English:   she was brave enxp enxp <eos>

German:    Mein Kopf schmerzt dexp dexp <eos>
English:   my brain is full enxp <eos>

German:    Du bist sehr mutig dexp <eos>
English:   you are very brave enxp <eos>
```

Figure 2: Output generated by model

***_head_ is semantically close to _brain_.**
***Similarly, _she_ to _he._ And _speaks_ to _bragged_**

Cross-entropy loss: Here, cross entropy loss essentially refers to the extent of error in grammar of encoder/decoder.

- When trained over all the sentences for multiple times, the error is decreasing for both encoder and decoder.
- Encoder loss finally settles near 0, while decoder loss always above the former. This is as expected because the decoder is continuously absorbing the error present in encoder through the hidden state at lat timestep of encoder

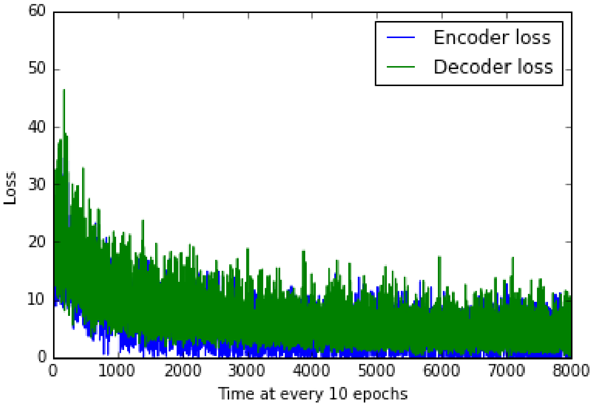


Figure 3: Plot - Cross entropy loss at SOFTMAX layer

Analysis of Wrong translation: Consider the translation done for third sentence – ''Thanks for your email" is translation output instead of – "Thanks for your help". This is happening because the SoftMax is assigning higher prob. to word – "email". Following diagram with similar example explains this in detail.

- There are just limited number of words that can occur after the word – *"have"*
- After *"a"*- there are hundreds of possible words that can occur
- In case of *"a"*, the SoftMax must assign the probabilities to hundreds of words, resulting in very small difference between prob. of occurrence of next words. This makes SoftMax very difficult to choose among hundreds of words. During BPTT, a slight change in weights can increase the prob. of unwanted words slightly.
- **Using Bi-RNN with attention can resolve this issue**

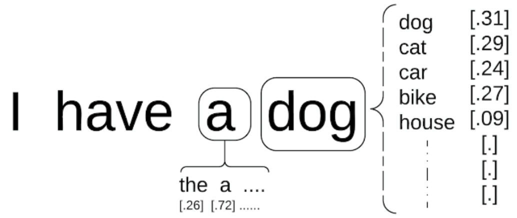


Figure 4: Example of small probabilities at SOFTMAX layer

Improvements:

1. **Bi directional RNN[1]:**
   a. After observing the above problem, we studied this iconic paper (Cho et al. [2014]) which discusses the implementation of bidirectional RNN with attention model.
   b. Bi-RNN will boost the translation accuracy as they can capture the context from both sides of the sentence.

2. **Attention Mechanism[1]:**
   a. To utilize the advantages of bi-directional RNN, we also need to implement the attention mechanism as explained by Cho et al in the paper. Attention allows the decoder to access all the hidden states of encoder at every timestep. This ensures for the decoder to use the context present in the various parts of a sentence.
   b. This principle has a significant impact on neural computation as we can select the most pertinent piece of information, rather than using all available information, a large part of it being irrelevant to compute the neural response.

3. **Word Embedding using word2vec[2]:**
   a. Our current model is using one-to-k hot vector encoding as the input to our model. In this type of encoding, all the words are equidistant. Word2vec encoding incorporates the semantic significance of words. All the words are mapped to another space where all the similar words fall under same cluster.
   b. The length of one-hot vector is equal to the size of vocabulary. In our case the size was approx. 2500 words. Multiplying big vectors take lot of time to process the output, hence increasing the training time. Using word2vec can decrease the input vector size to as low as 50. Hence, decreasing training time.

[1] "Neural Machine Translation by Jointly Learning to Align and Translate." https://arxiv.org/pdf/1409.0473. Accessed 23 Apr. 2017.

[2] "[1406.1078] Learning Phrase Representations using RNN ... - arXiv." 3 Jun. 2014, https://arxiv.org/abs/1406.1078. Accessed 23 Apr. 2017.