Spanish-English Machine Translation using Attention Mechanism

Ankit Tewari ankittewari@acm.org
Faculty of Mathematics and Statistics
Universitat Politecnica de Catalunya

Gabriele Libardi gabrielelibardi93@gmail.com
Faculty of Mathematics and Statistics
Universitat Politecnica de Catalunya

Editor: Dario Garcia-Gasulla

Keywords: Recurrent Neural Networks, GRU, Machine Translation

1. Introduction

The present work discusses the approach of neural machine translation of sentences from Spanish language to English language. It is considered a difficult problem despite the presence of similar semantics because of absence of large corpora. Furthermore, .

The data-set used for the task of neural machine translation come from the Project Tatoeba which is a collection of sentences and translations. It's collaborative, open, free source of data about languages. Further, we can also access the data for machine translation at the http://www.manythings.org/anki/ webpage which has a subset of data from the previous webpage for machine translation tasks. It is described below with the sentence in spanish on the left and the corresponding target sentences in english appearing on the right.

In our case, we are having 120614 sentence pairs. However, we are filtering the sentences based on the maximum length of the sentence and to begin with the problem, we have kept the maximum length equal to 10. During the experimentation, we have attempted to see if increasing the maximum length i.e. allowing more pairs to learn improves the quality of translation which is expected as the model will be able to use its parameters to learn a better representation of the underlying bitext.

```
Reading lines...
Read 120614 sentence pairs
Trimmed to 7755 sentence pairs
Counting words...
Counted words:
eng 4036
spa 2836
['estoy muy preocupado por mi hijo .', 'i m really worried about my child .']
```

Training data for the Neural Machine Translation Problem

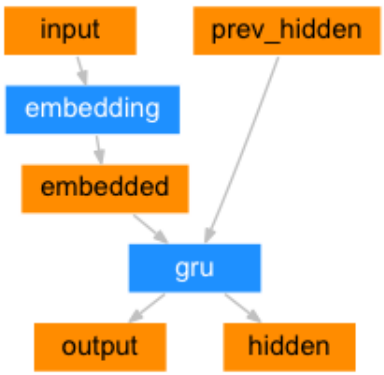
2. Methods

2.1 Data Preprocessing

The data we have read is all in unicode and therefore, in order to simplify the task, we will convert all the characters to lower case and remove any punctuation or such similar things. Then, we have converted it into the ASCII format. Further, we are not splitting the data into train, validation and test sets as we already have a relatively small dataset due to keeping the maximum number of allowed words in a sentence as 10. That means that our only evaluation metric will be the loss function.

2.2 The Encoder Recurrent Neural Network

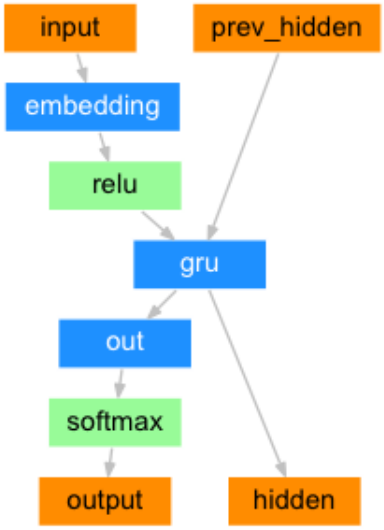
The encoder recurrent neural network is a neural network that takes into input a word and outputs a vector along with a hidden state. This hidden state serves as the input for the next step as described in the figure below-



Architecture for Encoder

2.3 The Decoder Recurrent Neural Network

The decoder recurrent neural network is another recurrent neural network that takes the output of the decoder neural network as the input and outputs another vector for translation.

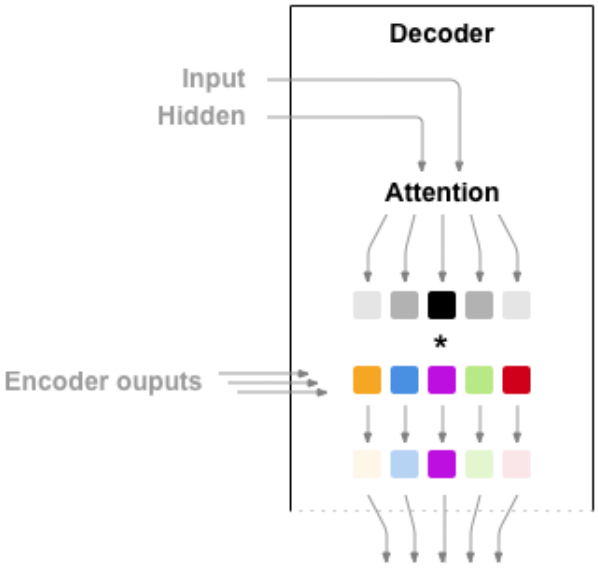


Architecture for Decoder Network

2.4 The Attention Decoder

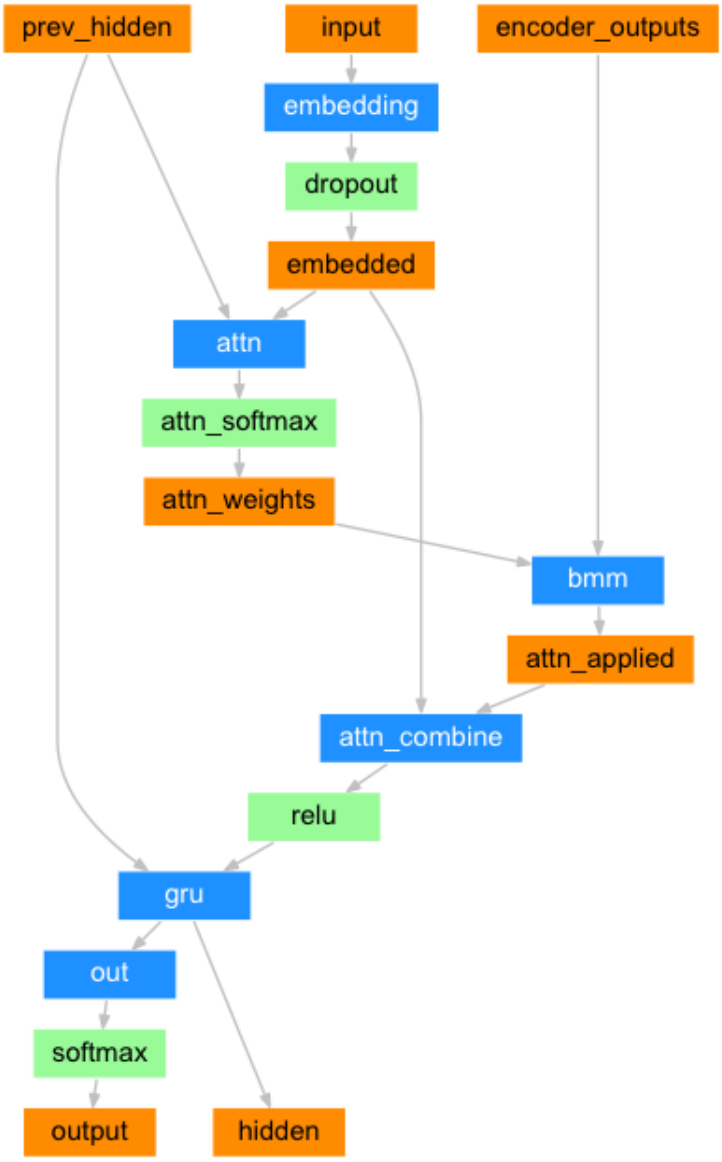
If only the context vector is passed betweeen the encoder and decoder, that single vector carries the burden of encoding the entire sentence.

Attention allows the decoder network to focus on a different part of the encoders outputs for every step of the decoders own outputs. First we calculate a set of attention weights. These will be multiplied by the encoder output vectors to create a weighted combination. The result (called attn_applied in the code) should contain information about that specific part of the input sequence, and thus help the decoder choose the right output words.



Architecture for Attention Decoder

Calculating the attention weights is done with another feed-forward layer attn, using the decoders input and hidden state as inputs. Because there are sentences of all sizes in the training data, to actually create and train this layer we have to choose a maximum sentence length (input length, for encoder outputs) that it can apply to. Sentences of the maximum length will use all the attention weights, while shorter sentences will only use the first few.



Architecture for Attention Decoder Network

In order to train the model, we feed the input sentence through the encoder and record the every output and the latest hidden state. Then the decoder is given the ¡SOS¿ token as its first input, and the last hidden state of the encoder as its first hidden state.

Teacher forcing is the concept of using the real target outputs as each next input, instead of using the decoders guess as the next input. Using teacher forcing causes it to converge faster but when the trained network is exploited, it may exhibit instability.

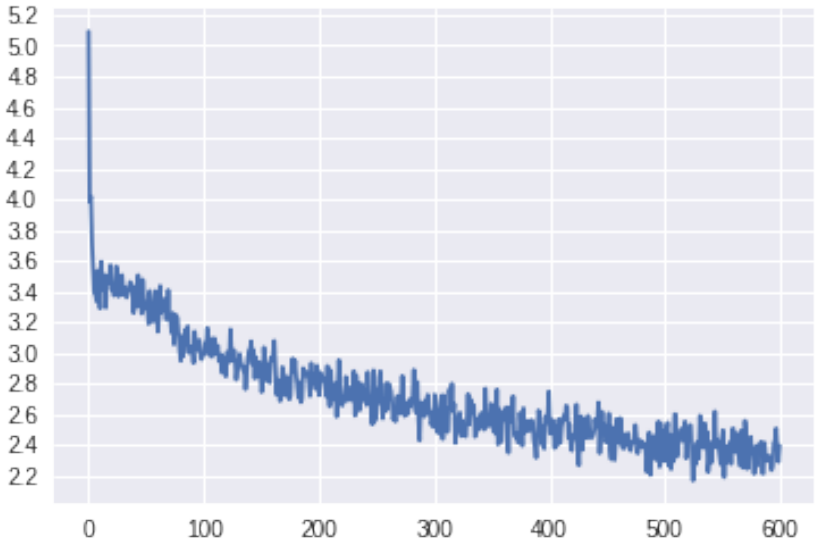
We can observe outputs of teacher-forced networks that read with coherent grammar but wander far from the correct translation - intuitively it has learned to represent the output grammar and can pick up the meaning once the teacher tells it the first few words, but it has not properly learned how to create the sentence from the translation in the first place.

3. Results

This section describes the results for the experiments conducted on the application of word embeddings and choices of models for solving the stated problem. We have started exploring the possibility of product classification using pre-trained word2vec embeddings trained on the dataset of Google News, fed as an input to a variety of configurations of deep neural networks for classification task as described previously.

3.1 Experiment 1: Impact of different number of hidden units

In this first experiment, we have tried to change the number of hidden units and measure the impact on the time and iterations taken to converge the loss functions. Therefore, experiments were conducted with 32, 64, 256, 512 and 1024 hidden units while keeping other factors constant including the model architecture. If the number of iterations was kept at 60000 iterations for the sake of comparison, it was observed that with the increase in the number of hidden units, there was a faster convergence of the loss function to the optimal value. The plots below describe the scenario-



Experiment 1: Loss function for model with 32 hidden units

The runtime of the model with 32 hidden units is described below. t clearly shows that despite running the model for 60000 iterations, the model was not able to converge the loss below the value of 2.

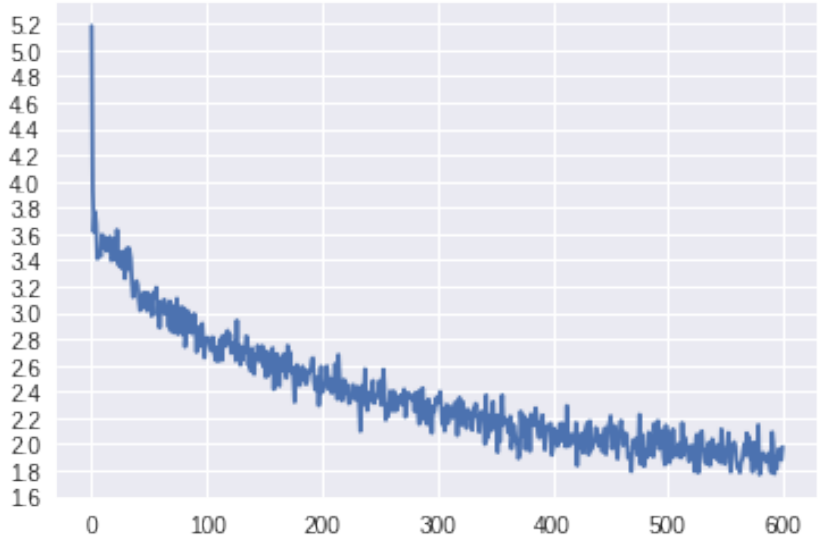
```
33m 11s (- 11m 3s) (45000 75%) 2.5063
33m 55s (- 10m 19s) (46000 76%) 2.4481
34m 40s (- 9m 35s) (47000 78%) 2.4330
35m 24s (- 8m 51s) (48000 80%) 2.4189
36m 7s (- 8m 6s) (49000 81%) 2.3537
36m 52s (- 7m 22s) (50000 83%) 2.4164
37m 37s (- 6m 38s) (51000 85%) 2.4386
38m 20s (- 5m 53s) (52000 86%) 2.4257
39m 4s (- 5m 9s) (53000 88%) 2.3823
39m 49s (- 4m 25s) (54000 90%) 2.4203
40m 33s (- 3m 41s) (55000 91%) 2.3960
41m 17s (- 2m 56s) (56000 93%) 2.3819
42m 1s (- 2m 12s) (57000 95%) 2.3731
42m 45s (- 1m 28s) (58000 96%) 2.3365
43m 30s (- 0m 44s) (59000 98%) 2.3338
44m 14s (- 0m 0s) (60000 100%) 2.3511
```

Runtime of model with 32 hidden units

However, upon testing the model on the randomly chosen validation set, we observed that model was translating fairly accurately. We have provided the result of translation on a validation set randomly chosen and displayed below-

```
> estoy rezando .
= i am praying .
< i am praying . <EOS>

> esta totalmente desnudo .
= he s stark naked .
< he s stark naked . <EOS>

> no me interesa tu opinion .
= i m not interested in your opinion .
< i m not interested in your opinion . <EOS>

> leo el libro .
= i am reading the book .
< i m reading the book . <EOS>

> son un cero a la izquierda .
= you re useless .
< you re useless . <EOS>

> somos tus vecinos .
= we re your neighbors .
< we re your neighbors . <EOS>

> estoy jugando con mis amigos .
= i m playing with my friends .
< i am playing with my friends . <EOS>
```

Model performance on a randomly chosen validation set

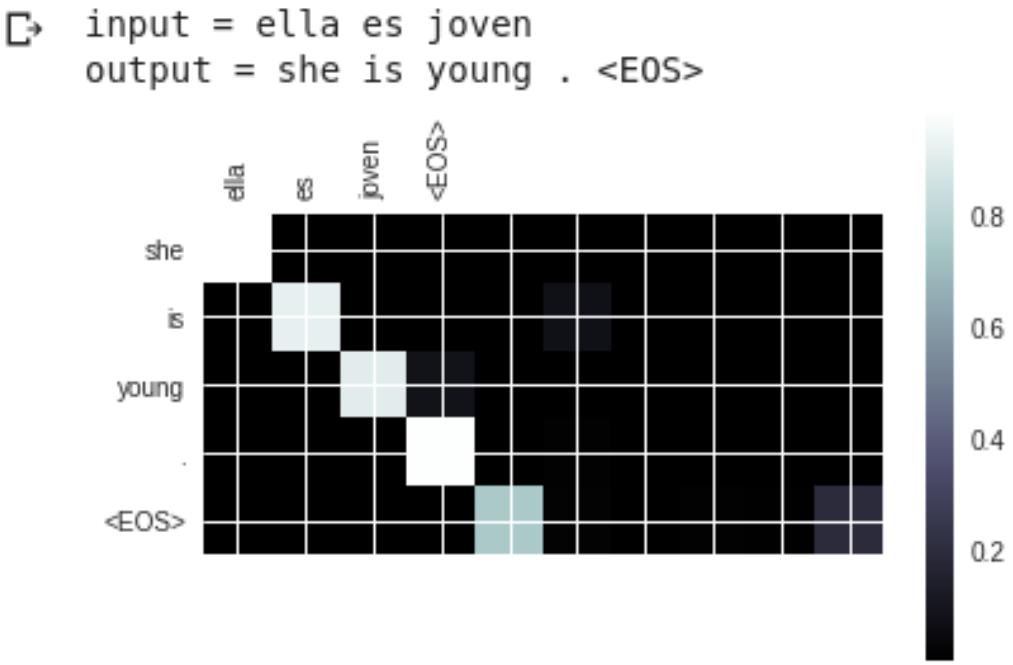


Experiment 2 : Model's loss function with 64 Hidden Layers

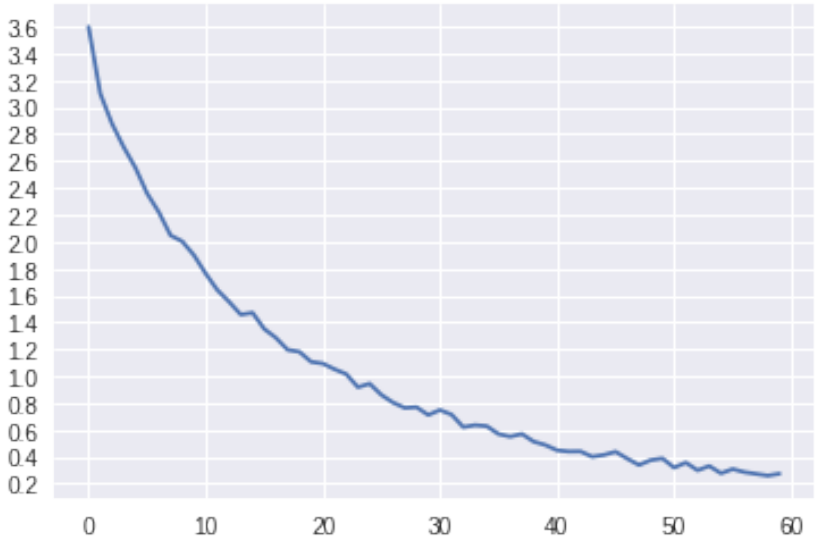
```
50m 10s (- 21m 30s) (42000 70%) 2.0472
51m 49s (- 20m 29s) (43000 71%) 2.0312
53m 33s (- 19m 28s) (44000 73%) 2.0465
55m 16s (- 18m 25s) (45000 75%) 2.0459
57m 0s (- 17m 21s) (46000 76%) 2.0668
58m 42s (- 16m 14s) (47000 78%) 1.9764
60m 22s (- 15m 5s) (48000 80%) 1.9934
62m 3s (- 13m 55s) (49000 81%) 1.9891
63m 45s (- 12m 45s) (50000 83%) 2.0110
65m 20s (- 11m 31s) (51000 85%) 1.9691
67m 1s (- 10m 18s) (52000 86%) 1.9621
68m 41s (- 9m 4s) (53000 88%) 1.9647
70m 22s (- 7m 49s) (54000 90%) 1.9256
72m 6s (- 6m 33s) (55000 91%) 1.9198
73m 47s (- 5m 16s) (56000 93%) 1.9531
75m 23s (- 3m 58s) (57000 95%) 1.9050
77m 2s (- 2m 39s) (58000 96%) 1.9202
78m 43s (- 1m 20s) (59000 98%) 1.8725
80m 26s (- 0m 0s) (60000 100%) 1.9218
```

Runtime of model with 64 Hidden units

Below, we have explained the translation using the attention layer-

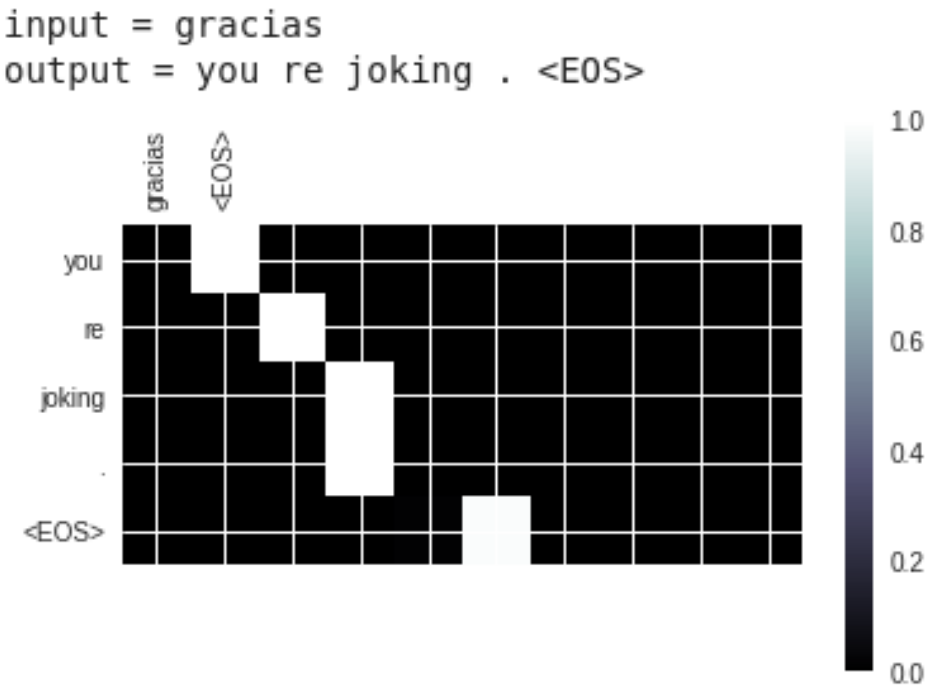


Correct Evaluation Explained by Attention Layer

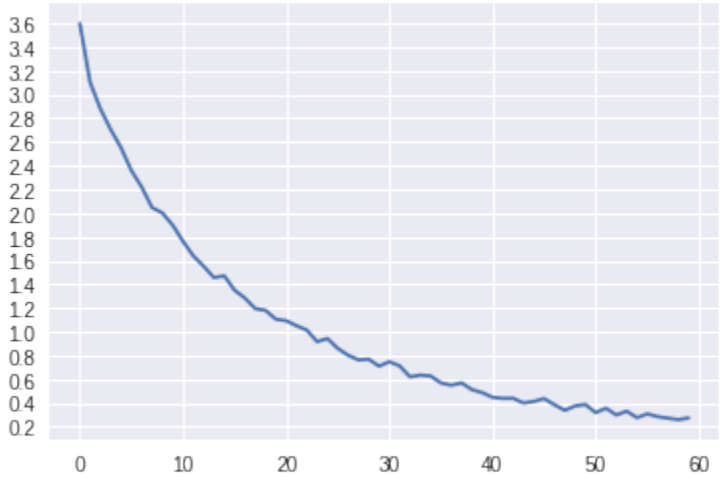


Experiment 3: Model's loss with 512 hidden units

In order to examine how well the model performs, we have decided to interpret the predictions obtained using the attention layer for this configuration. Below, we describe a wrongly interpreted word.



Wrong Evaluation Explained by Attention Layer for Model with 521 hidden units



Experiment 4: Model's loss with 1024 hidden units

```
[20] 57m 57s (- 47m 25s) (33000 55%) 0.8469
     59m 13s (- 45m 17s) (34000 56%) 0.7807
     60m 24s (- 43m 9s) (35000 58%) 0.7466
     61m 35s (- 41m 3s) (36000 60%) 0.7711
     62m 46s (- 39m 1s) (37000 61%) 0.7527
     63m 58s (- 37m 2s) (38000 63%) 0.6950
     65m 9s (- 35m 5s) (39000 65%) 0.6964
     66m 21s (- 33m 10s) (40000 66%) 0.6247
     67m 33s (- 31m 18s) (41000 68%) 0.5867
     68m 44s (- 29m 27s) (42000 70%) 0.5563
     69m 56s (- 27m 39s) (43000 71%) 0.5321
     71m 7s (- 25m 51s) (44000 73%) 0.5920
     72m 19s (- 24m 6s) (45000 75%) 0.5089
     73m 31s (- 22m 22s) (46000 76%) 0.4674
     74m 42s (- 20m 39s) (47000 78%) 0.4593
     75m 54s (- 18m 58s) (48000 80%) 0.4958
     77m 6s (- 17m 18s) (49000 81%) 0.4565
     78m 18s (- 15m 39s) (50000 83%) 0.4310
     79m 31s (- 14m 1s) (51000 85%) 0.4105
     80m 43s (- 12m 25s) (52000 86%) 0.3823
     81m 55s (- 10m 49s) (53000 88%) 0.3602
     83m 6s (- 9m 14s) (54000 90%) 0.3838
     84m 18s (- 7m 39s) (55000 91%) 0.3887
     85m 29s (- 6m 6s) (56000 93%) 0.3387
     86m 40s (- 4m 33s) (57000 95%) 0.3527
     87m 52s (- 3m 1s) (58000 96%) 0.3471
     89m 4s (- 1m 30s) (59000 98%) 0.3702
     90m 1s (- 0m 0s) (60000 100%) 0.3114
```

Runtime of model with 1024 Hidden units

```
[35] > no depende de sus padres .
     = he is independent of his parents .
     < i m not the of of . <EOS>

     > me temo que tengo hemorragia interna .
     = i m afraid i have internal bleeding .
     < i m afraid of you will . <EOS>

     > estoy satisfecho que tu todavia recuerdes .
     = i m pleased you still remember .
     < i m sure you your your . <EOS>

     > no estoy escribiendo una carta .
     = i am not writing a letter .
     < i m not a a a a <EOS>

     > estoy intrigada .
     = i m intrigued .
     < i m sick . <EOS>

     > somos personas ocupadas .
     = we re busy men .
     < we re all . . <EOS>

     > estamos todos de acuerdo .
     = we are all in agreement .
     < we re all in . <EOS>
```

Performance of model with 1024 Hidden units on randomly chosen validation set

The above results clearly show that this is a case of overfitting of the model on the training dataset. Now, we can conclude that with the increase in hidden units, there is an increase in convergence of loss function towards zero. However, after a certain threshold, this might turn into overfitting. We can play with the dropout value in this case to examine further about overfitting.

4. Discussions

- Impact of number of hidden units: The number of hidden units significantly impact the performance of the model in such a way that as we increase the number of hidden units, the model achieves lower values of loss function. After reaching a certain threshold, the model begins to overfit. Therefore, we need a strategy to evaluate the model's performance on a test set using some kind of standard metric for evaluation such as "BLEU" scores.

- Impact of learning rate: Under the present framework, we repeatedly ran the experiments with varying values of learning rate to examine the performance of the algorithm. It was brought out that the learning rate of 0.01 was just perfect to train the algorithm. While we used the learning rate of 0.1 and 0.05, the algorithm overshot the convergence minima and therefore we did not observe the convergence of

loss function. Similarly in case of using the learning rate of 0.0001, we observed that it was taking too small steps towards convergence. Therefore even after performing multiple epochs upto 100000, the algorithm was far from desired convergence value. It clearly states the need for hyperparameter tuning to achieve desired results for machine translations.

- Impact of dropout: While dropout is effective in preventing the overfitting, in our case, we had tried the values of 0.1, 0.15 and 0.2 for the dropout layer. However, there was not much difference in the model's performance and its pace of convergence.

5. Conclusions

It would be interesting to examine the problem from a different context by playing even further with the paramters of the model. Some interesting ideas maybe derived by studying the relation between different values of teacher_ratio in this model based on pytorch. It would also be a great idea to keep the number of maximum words allowed in a sentence higher for the sake of comparison and we can examine how well does the model behave by allowing 20, 25 and 30 words as the maximum permissible words. Our goal therefore will be to develop the model in such a way that it can accurately translate a large number of features. Further, We may would like to explore the application of word embeddings such as Glove and Word2Vec which can serve the purpose of being the input layer and compare our outcomes with what we attained without a pre-trained word-embeddings. This suggestion is expected to understand the difference between the attention sequence's working with and without pre-trained word embedding.

Acknowledgments

We would like to acknowledge support provided for this project from Dario who had been an exceptionally well instructor for Machine Learning and allowed us to explore newer frontiers in the topic.

6. References

1) Machine Translation with a Sequence-to-Sequence Recurrent Neural Network using Attention(Hover over the text to click the link to the tutorial).

Appendix A: Code segments for different experimentation carried out under the exploratory assignment

The code for this project can be accessed online on the github at Jupyter Notebook for the Neural Machine Translation Model using Attention Layer. Hover over the text to access the link :)