Machine Language Translation Using Deep Learning.

CSE 676 Final Project Report

Sailesh Reddy Sirigireddy
UB-IT: saileshr
UB-ID: 50496100

Shri Harsha Adapala Thirumala
UB-IT: sadapala
UB-ID: 50495139

Chandini Kondinolu
UB-IT: ckondino
UB-ID: 50495790

Summary:

Deep Learning has brought about a revolutionary transformation across various domains, particularly in Natural Language Processing (NLP). It has shown great potential, provided it is implemented correctly. Language has traditionally posed a challenge to achieving global connectivity. However, by harnessing deep learning models, the computational power available today, and the vast amount of data accessible on the internet, we can surmount this language barrier in machine language translation.

The question arises: Can deep learning models be employed for language translation? If so, to what extent can they achieve accurate results? Additionally, can the recent advancements in innovative and complex architectures further enhance the translation capabilities of deep learning techniques?

Data:

The data is taken from the parallel language corpus available in Kaggle. The DGT-TM is a comprehensive translation memory consisting of sentences and their manually generated translations in 24 different languages. It encompasses segments derived from the Acquis Communautaire, which encompasses the entire body of European legislation, including treaties, regulations, and directives established by the European Union (EU).

We have selected a dataset specifically for English-to-French translation. This dataset contains, pairs of sentences in English and their corresponding translations in French.

Data Pre-processing:

The library we have used for pre-processing.
1. spacy
2. torchtext
3. NLTK

To build the vocabulary for the English and French languages, we can utilize the Spacy library. Spacy provides a small pre-trained model that can assist with this task.

Before loading the data through torch text we preprocessed the data by converting it into lower case removing in case it has any digits and special symbols other than the following: !,.?

Later we selected the data of with sentence length 25 and below. We got a total of around 130000 sentences for train and 5842 test sentences.

Before creating the vocabulary, we need to perform tokenization on the data and convert the sentences to lowercase. To indicate the start and end of each sentence, we can utilize unique tags such as “<sos>” and “<eos>”. All these steps can be accomplished using the Field feature which is part of the torchtext library.

Next, we need to assign a numerical representation to each word, which is known as building vocabulary. This can be achieved by utilizing the Spacy model that we previously loaded. The maximum vocabulary size is set to 10,000. Any word that appears at least twice will be included in the vocabulary.

While feeding the data to the model we must pad shorter sentences to bring it to the length of the sentence of maximum length so that every data is of same sequence length. But on padding every sentence to the length of the maximum sentence of the whole data corpus we are diminishing the translation ability of the model for the shorter sentences. So, we will sort the whole corpus in terms of length of the sentence and then do batches and pad the shorter sentence of each batch to the size of the longer sentence in the batch, because of this model training will become seamless even for shorter sentences and prevent excessive padding. It will also improve model training time and speeds up the training.

Word Embeddings:

Representing each sentence as a bag of words has many limitations in the context of the Machine translation like:

1. No semantic representation.
2. As the size of the vocabulary increases the size of the vector increases.
3. Limited contextual information.

But we can overcome these limitations using word embeddings which can have the following benefits for Machine Translation:

1. Semantic Understanding
2. Handling Synonyms and Polysemy
3. Fixed Dimensionality of the word representation.
4. Can translate out of box vocabulary words.
5. Contextual Understanding.
6. Use existing word embeddings.

Instead of using the existing word embeddings like word2vec or Glove we can build our own word embeddings from start while training itself. For this we use **torch.nn.Embedding** layer in both encoder and decoder of every sequence to sequence to model we build.

Context Vector:

The context vector is a representation of the input sequence that captures the relevant information needed for generating the output sequence. It serves as a summary of the input sequence. In the sequence-to-sequence model with attention, the context vector is typically computed using the hidden states from the encoder.

The encoder processes the input sequence and produces a set of hidden states, where each hidden state represents the encoded information of a specific time step in the input sequence. The context vector is then derived from these hidden states.

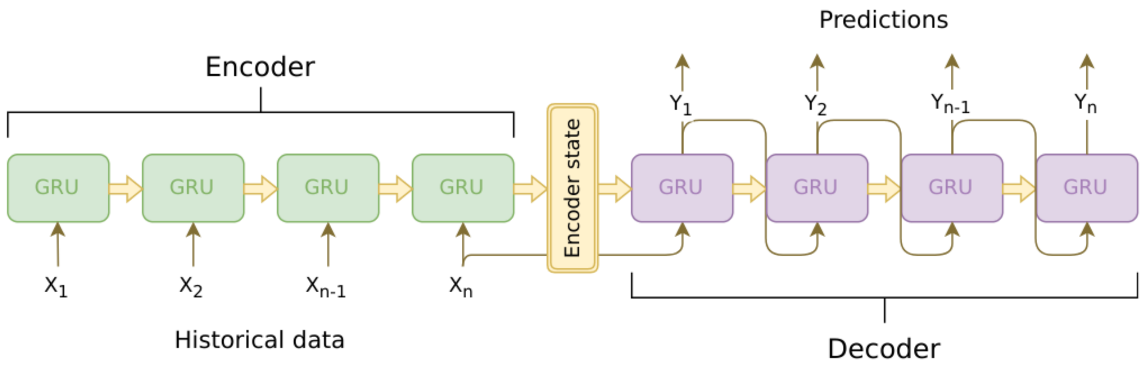
One common approach is to use the last hidden state of the encoder as the context vector. This means that the final hidden state of the encoder captures the overall information of the input sequence and is considered as the condensed representation of the input.

By using the context vector, the decoder can access the relevant information from the input sequence while generating the output sequence. It allows the decoder to focus on different parts of the input sequence dynamically and attend to the most important details based on the current decoding step.

Model Building and Analysis:

Sequence to Sequence Model

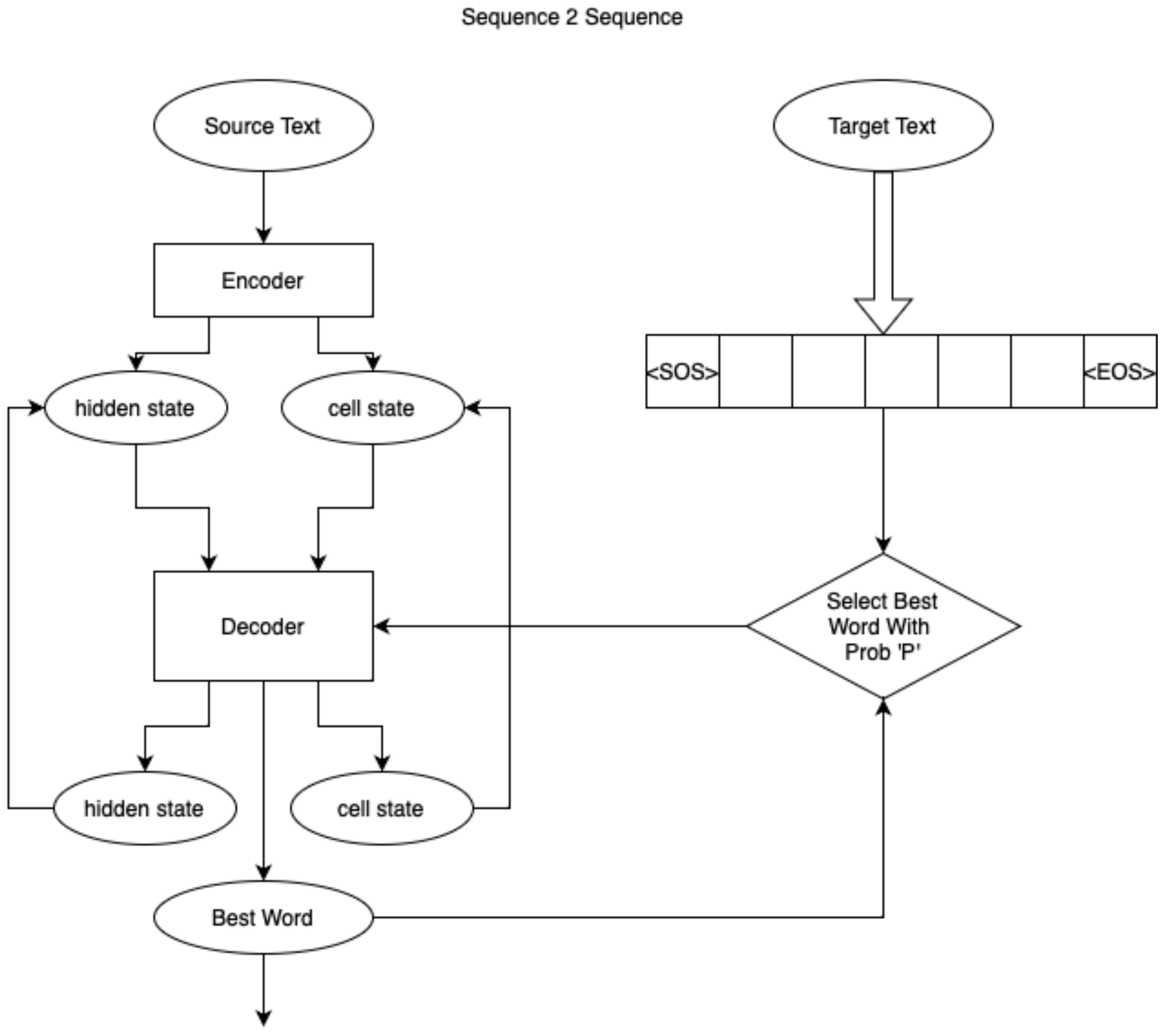
The architecture we use for the task at hand is a Sequence-to-Sequence model. Sequence-to-Sequence uses an encoding and decoding paradigm, and the base unit is the LSTM layer.



In a sequence-to-sequence neural network, we have an encoder and a decoder. Initially, we send the source text, which is a sentence in the language to be translated, to the encoder. The encoder produces a hidden state and a cell state, which together form a context vector. This context vector is then fed into the decoder during the first iteration.

In the decoder, during the first iteration, we provide the start of sentence token (<sos>) from the target data along with the context vector obtained from the encoder. The decoder predicts a word for the current time step and returns a hidden state and a cell state. For the next iteration, we use the hidden and cell states from the decoder's output along with the next input word. However, there is a slight variation in selecting the next word for the decoder. During training, we can choose to provide the input word from the target text or from the word predicted in the previous time step. We make this choice randomly with a 0.6 probability.

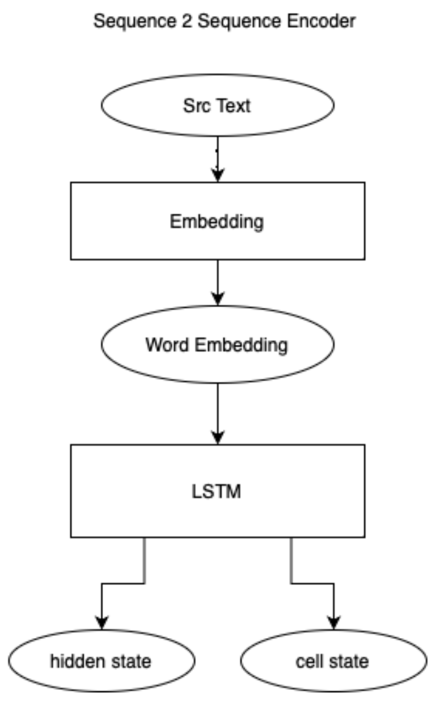
We continue the decoding process until we encounter the end of sentence token (<eos>) in the target text.



Encoder:

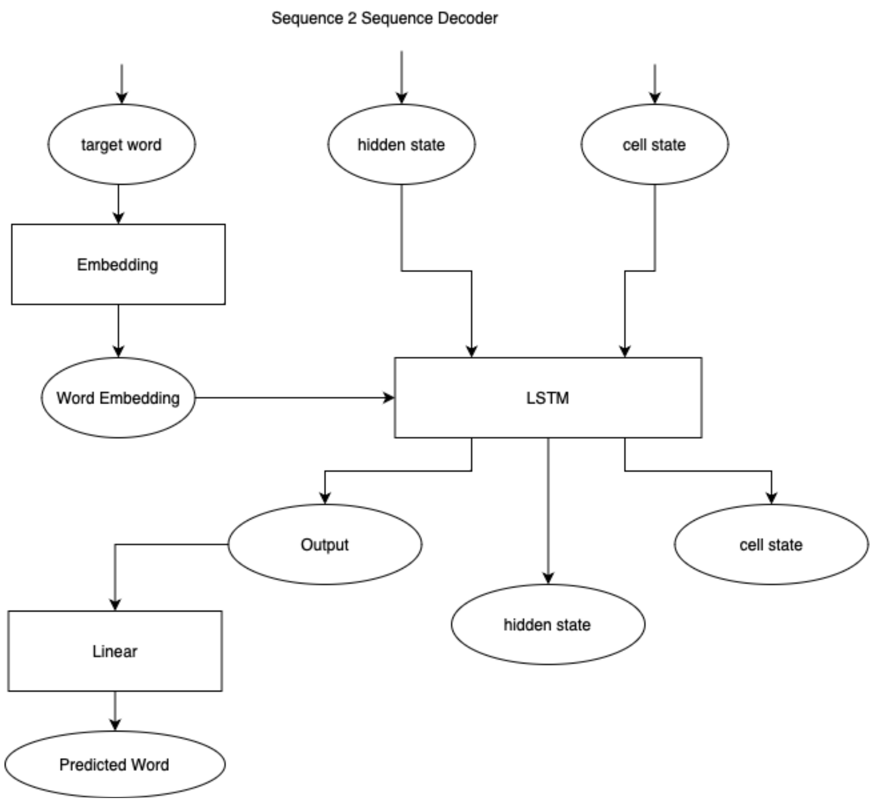
The encoder plays a crucial role in generating a context vector, which serves as an abstract representation of the entire sentence we aim to translate. Initially, we convert each word from the vocabulary into a vector of n dimensions using an embedding layer.

Subsequently, we pass the entire sentence, now in its embedded form, to the model. The recurrent layer processes the sequence, and upon completion, it provides us with the hidden state and cell state vectors. Together, these vectors represent the input sentence and its context.



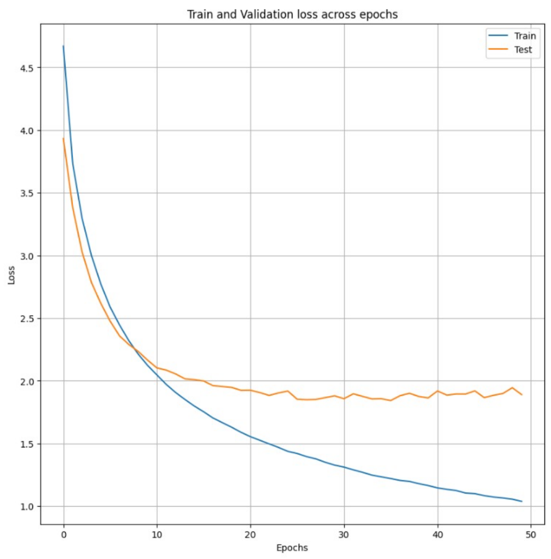
Decoder:

The decoder accepts the context vector as hidden state and cell state and combines it with a single word from the target sentence that requires translation. This combined information is subsequently passed into an LSTM module situated within the decoder.



The decoder generates a hidden state and cell state along with output which is subsequently fed to a linear layer to predict the output word.

Loss Plot:



The model is learning but slightly overfitting. We can see continuous decrease of training loss across all epochs but for testing the loss decreased exponentially till 10th epoch then the decreasing rate decreases up to 30 epochs only to get plateau for the rest of the epochs at 2.0.

Bidirectional Encoding

The whole architecture of the model almost remains the same, but we update the encoder with bidirectional encoding.

Encoder:

In the encoder architecture, the process begins by passing the source input text through an embedding layer, which generates word embeddings. These embeddings capture the semantic and contextual information of each word.

Next, the embedded representations are fed into two LSTM nodes. One LSTM reads the text sequentially from the beginning (forward direction), while the other LSTM reads it in reverse from the end (backward direction). Both LSTMs generate hidden states and cell states that encode the learned information from their respective directions.

To combine the information from both LSTMs, the hidden states and cell states from both LSTM nodes are concatenated or appended together. This concatenation operation creates a unified representation that captures information from both the forward and backward contexts.

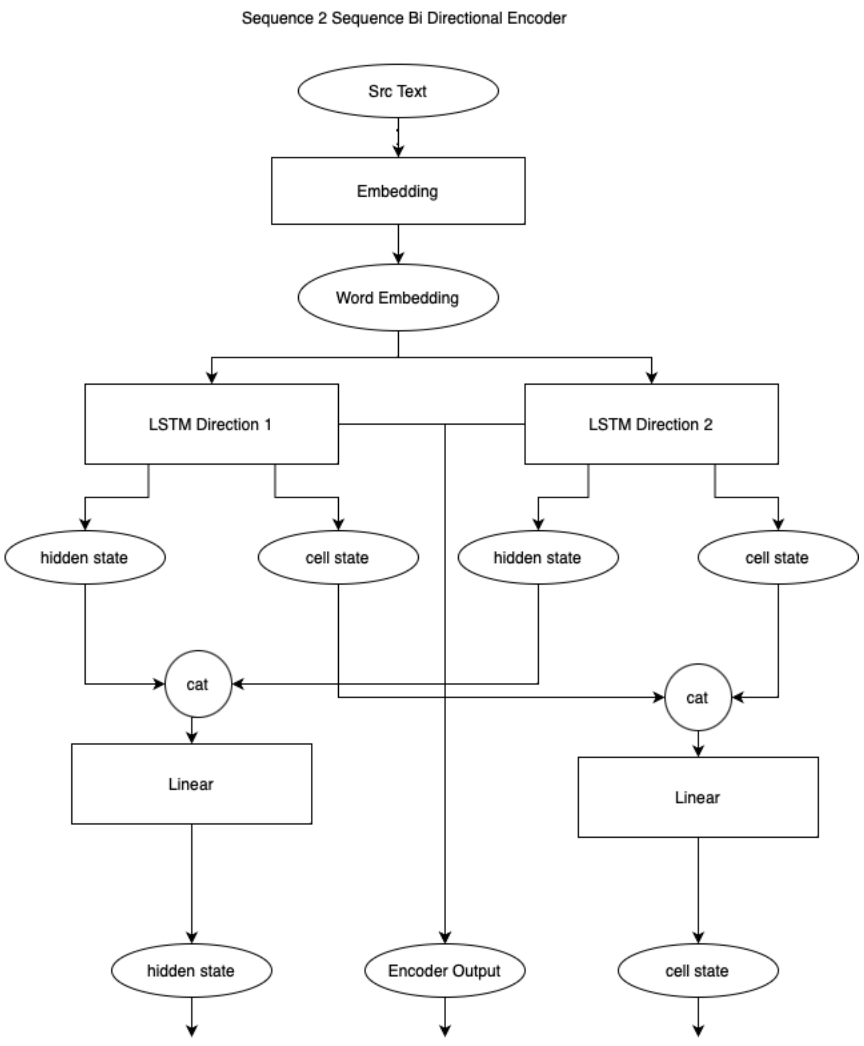
Finally, the concatenated hidden states and cell states are passed through linear layers, also known as fully connected layers. These linear layers can apply transformations to the concatenated representation, allowing the model to further process and extract higher-level features from the combined information.

The idea behind letting the models decide is to allow them to choose whether to utilize information from the forward LSTM layer, the backward LSTM layer, or both, and to determine the degree to which each layer contributes to the overall understanding or prediction.

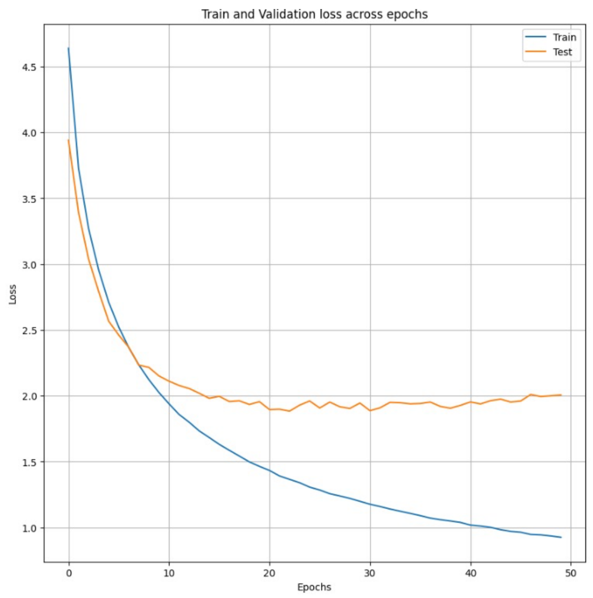
In this approach, the models have the flexibility to make their own decisions based on the specific task at hand. They can weigh the importance of information from different directions and adjust their learning accordingly. This enables them to adapt to the unique characteristics of the input data and leverage the most relevant information from both the forward and backward contexts.

By granting the models this autonomy, they have the freedom to utilize the most valuable information from each layer, potentially improving their performance in tasks that require capturing dependencies and contextual understanding.

The output of the linear layers can then be used as input for subsequent layers or tasks in the model, such as decoding or generating the translated text. The combination of the forward and backward LSTMs, along with the linear layers, helps the model leverage both the past and future context of the input text to generate more accurate and contextually aware representations.


Sequence 2 Sequence Bi Directional Encoder
Src Text
Embedding
Word Embedding
LSTM Direction 1
LSTM Direction 2
hidden state
cell state
hidden state
cell state
cat
cat
Linear
Linear
hidden state
Encoder Output
cell state

Loss Plot:


Train and Validation loss across epochs
Train
Test
Loss
Epochs
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
10
20
30
40
50

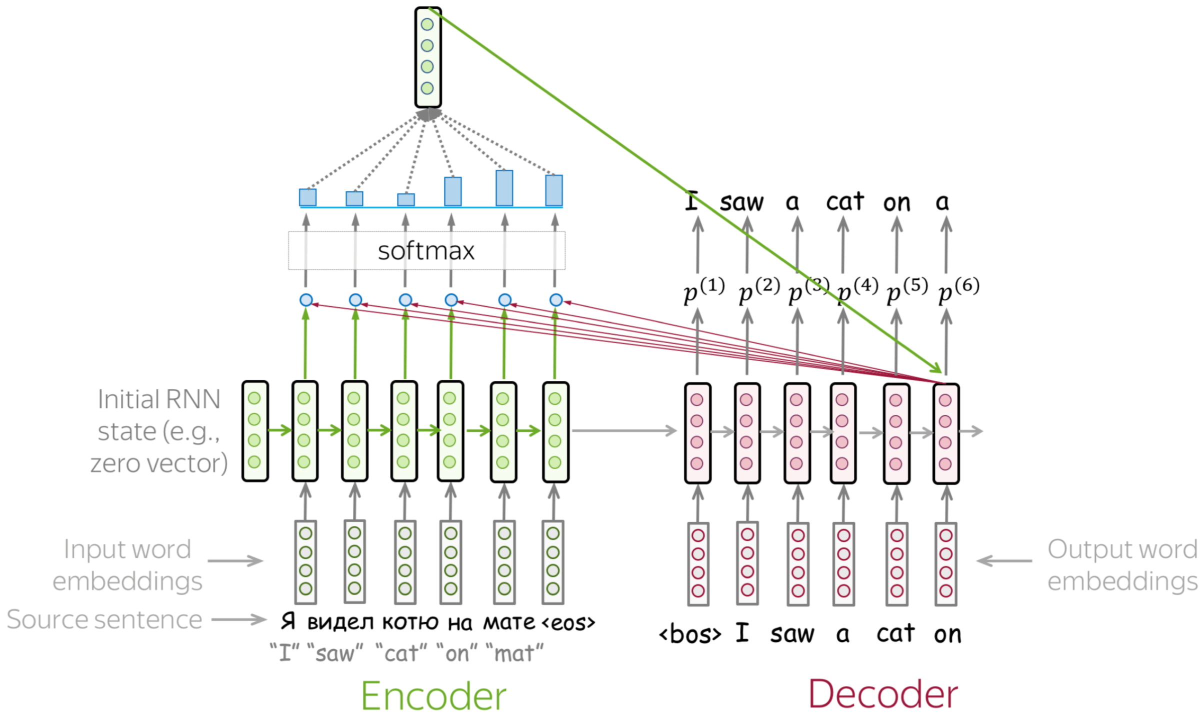
The model is learning but slightly overfitting. We can see continuous decrease of training loss across all epochs but for testing the loss decreased exponentially till 10th epoch then the decreasing rate decreases up to 30 epochs only to get plateau for the rest of the epochs at 2.0.

Attention with Jointly learning to Align and Translate

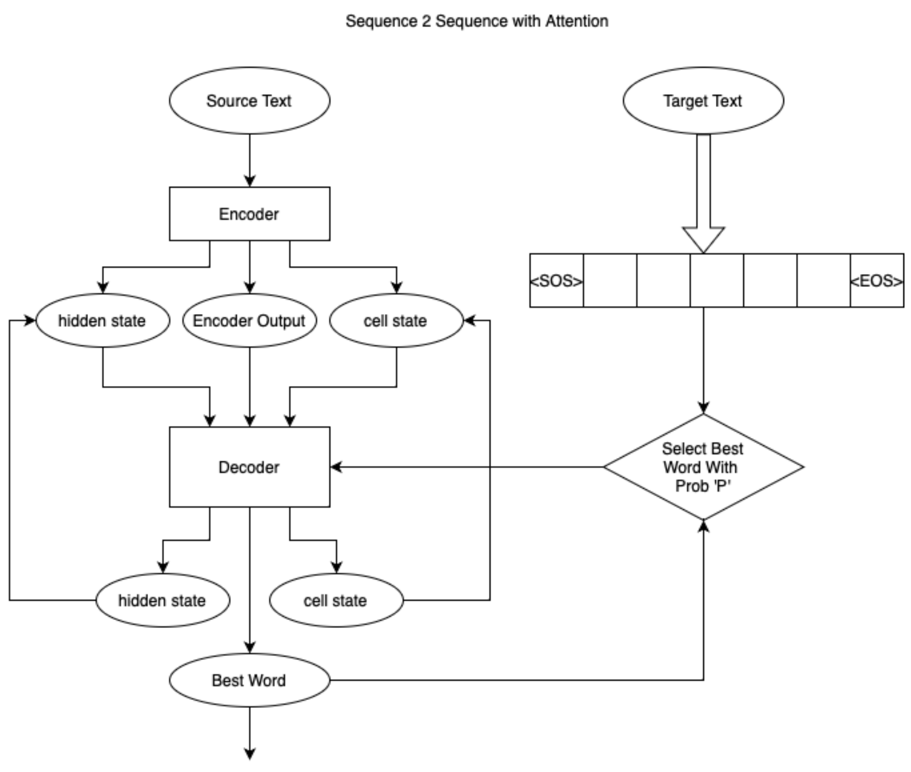
To improve the Bidirectional Sequence-to-Sequence model, an attention mechanism is integrated to enhance the model's ability to emphasize specific time steps that are considered significant.

The attention mechanism plays a crucial role in sequence-to-sequence models by dynamically directing the model's attention to different segments of the input sequence while generating the output sequence. This mechanism enables the model to determine the importance and relevance of information at each decoding step.

By incorporating the attention mechanism, the model gains the capability to dynamically focus on various portions of the input sequence, adaptively selecting the relevant details during each decoding step. This mechanism enhances the model's capacity to generate precise and contextually appropriate output sequences.


softmax
I saw a cat on a
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$ $p^{(4)}$ $p^{(5)}$ $p^{(6)}$
Initial RNN state (e.g., zero vector)
Input word embeddings
Source sentence
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
Encoder
<bos> I saw a cat on
Output word embeddings
Decoder

The Sequence-to-Sequence network retains a similar architecture as previous models, but with one key difference. In this case, the encoder not only returns the hidden state and cell state but also an output state. This output state is fed to the decoder model at each time step, allowing it to calculate attention scores. The output state contains information from every time step of the encoder, enabling the decoder to focus on relevant parts of the input sequence during the decoding process.



Encoder:

The architecture of the Sequence-to-Sequence bidirectional model is followed in this case.

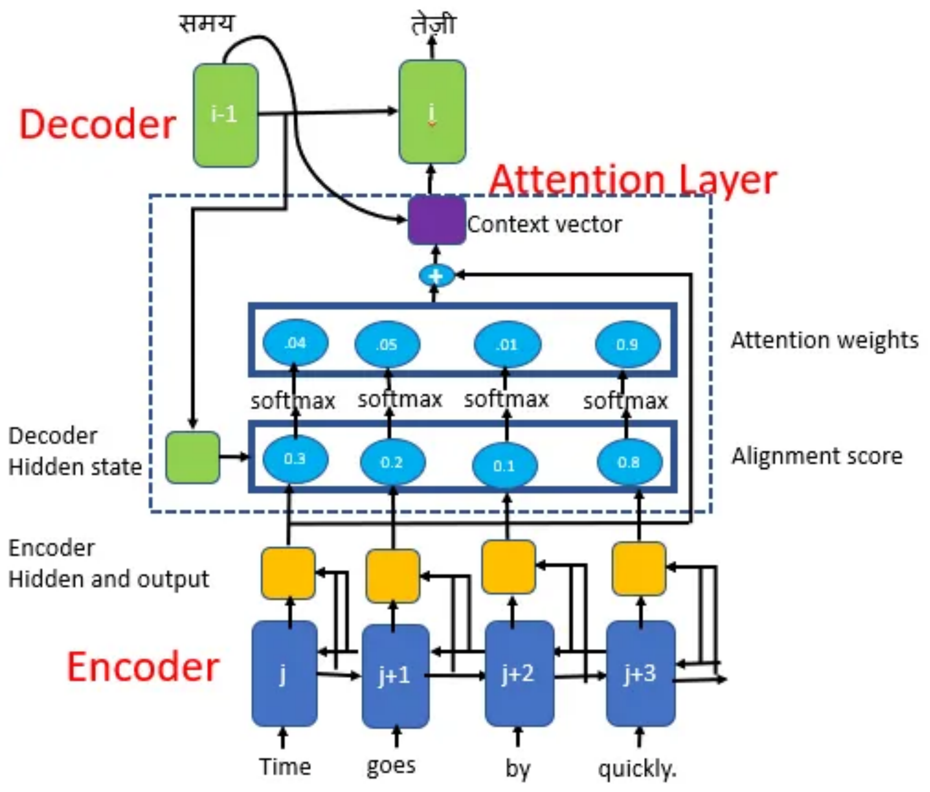
Decoder:

In the decoder layer of the model, the attention mechanism is applied. Initially, the decoder receives the encoder output and context vector as hidden and cell states. Among the outputs generated by the encoder, only the encoder output contains information from the entire sequence of words. Therefore, it is used to calculate the attention scores at each time step.

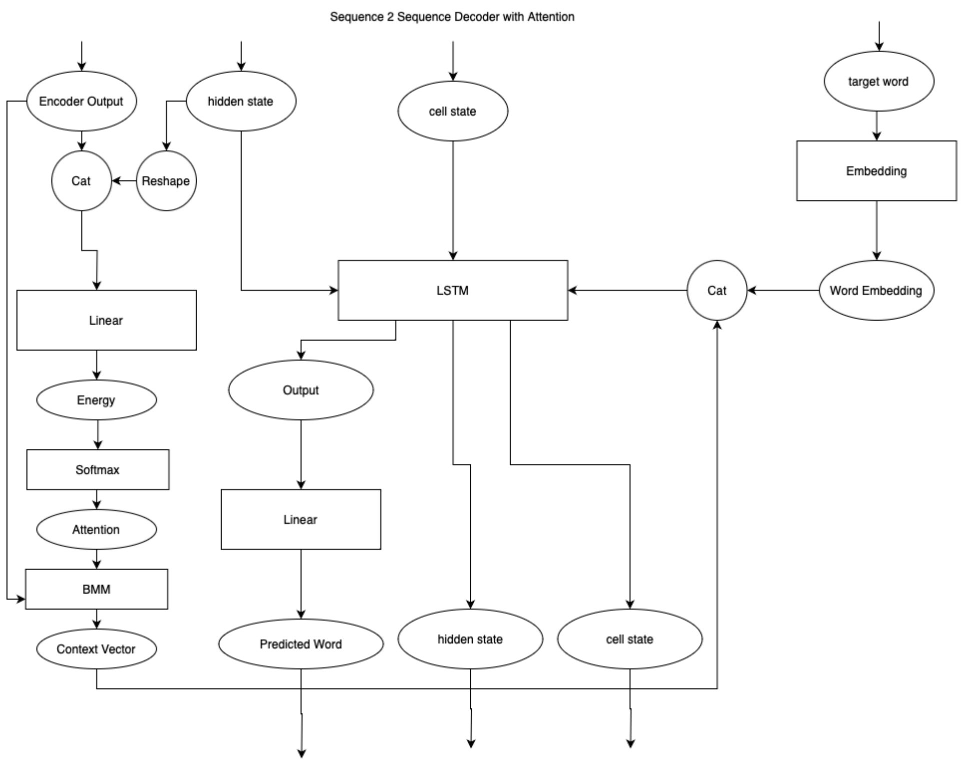
To calculate the attention scores, the hidden shape is reshaped to allow concatenation with the encoder outputs. This concatenated representation is then passed through a linear layer to determine the energy states for each time step. Subsequently, the energy states go through a SoftMax layer to generate the attention scores. Since the attention scores are the output of the SoftMax layer, their values will sum up to one.

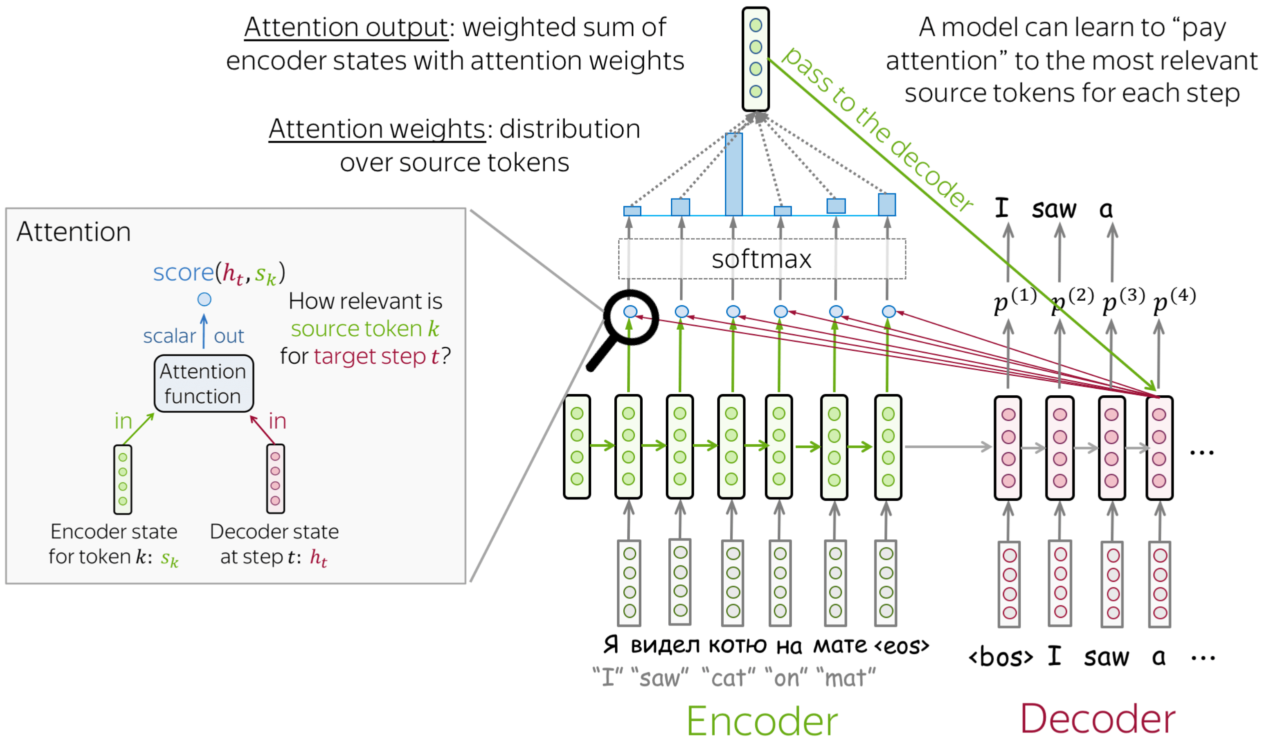
The next step involves determining which time step is important for predicting the current word. This is achieved by performing a batch matrix multiplication between the attention scores and the encoder output, resulting in a context vector.

The context vector is then concatenated with the word embeddings generated from the target word. This combined representation is fed into the model along with the hidden state and cell state.



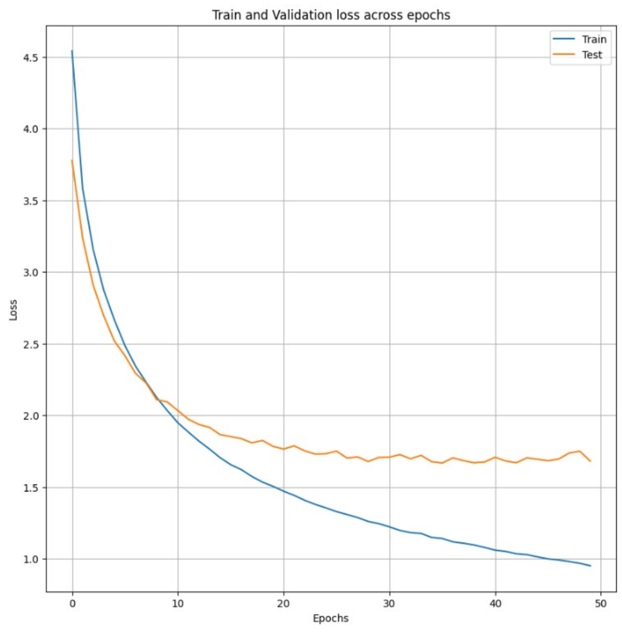
The LSTM generates a new hidden state and cell state, along with an output. These are further passed through another linear layer to produce the predicted word for the current time step.





Attention output: weighted sum of encoder states with attention weights
A model can learn to "pay attention" to the most relevant source tokens for each step
Attention weights: distribution over source tokens
pass to the decoder
softmax
Attention
score(h_t, s_k)
scalar out
How relevant is source token k for target step t?
Attention function
in
in
Encoder state for token k: s_k
Decoder state at step t: h_t
I saw a
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$ $p^{(4)}$
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
<bos> I saw a …
Encoder
Decoder

Loss Plot:

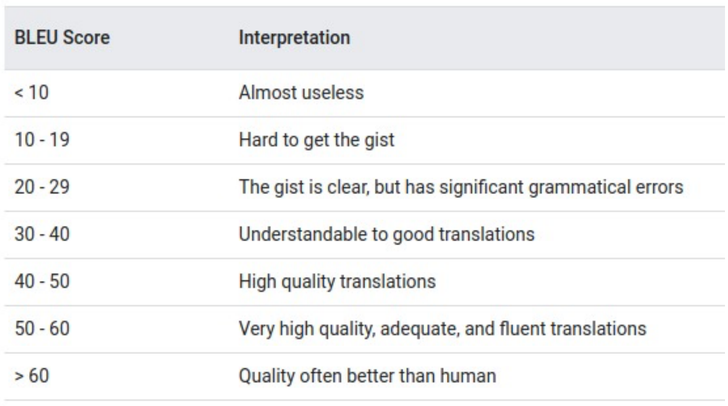


The model is learning but slightly overfitting. We can see continuous decrease of training loss across all epochs but for testing the loss decreased exponentially till 10th epoch then the decreasing rate decreases up to 30 epochs only to get plateau for the rest of the epochs at 1.6.

Evaluation – BLeU Score:

BLeU Score is a metric for automatically evaluating machine-translated text. It is a number between zero and one that measures the similarity of the machine-translated text to a set of high-quality reference translations.

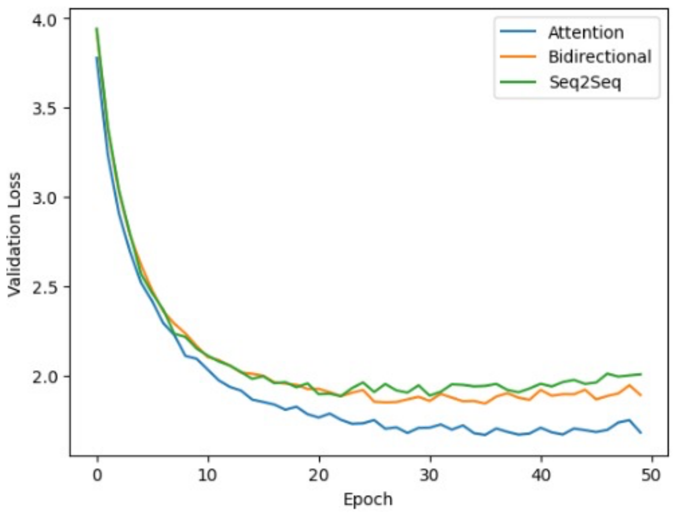
A value of 0 means that the machine-translated output has no overlap with the reference translation (low quality) while a value of 1 means there is perfect overlap with the reference translations (high quality).



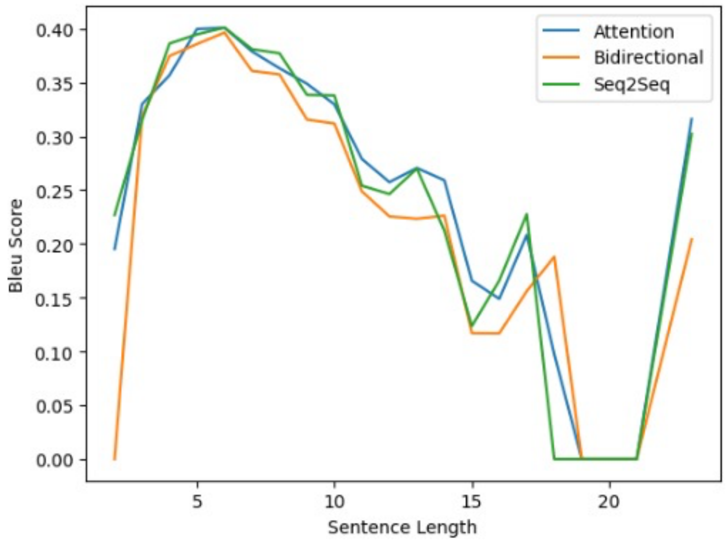
BLEU Score	Interpretation
< 10	Almost useless
10 - 19	Hard to get the gist
20 - 29	The gist is clear, but has significant grammatical errors
30 - 40	Understandable to good translations
40 - 50	High quality translations
50 - 60	Very high quality, adequate, and fluent translations
> 60	Quality often better than human

Model Name	**Bleu Score**
Seq-2-Seq Model	0.365
Bidirectional Encoding	0.369
Attention Decoding	0.406

Summary:



We can see that the test loss decrease of seq-2-seq model is slightly more than bidirectional encoding and even more than Attention decoding. After the curves becoming plateau the test loss of Seq-2-Seq model is more than 2, while the bidirectional encoding loss is fluctuating around 2 and settled at slightly less than 2. But for Attention decoding the test loss decreased further and settled at 1.6



The above plot shows bleu score of different models over the test data across sentences of different length. We can see that over all Attention model is performing better than Bidirectional and sequence-to-sequence model and Bidirectional is performing better than sequence-to-sequence.

We can see improved performance of model after using attention for sentence of larger sequence length.

Contribution:

Team Member	**Project Part**	**Contribution**
Chandini Kondinolu	Bidirectional Encoding (Coding) Report Presentation	30%
Sailesh Reddy Sirigireddy	Sequence – to - Sequence (Coding)	35%
Shri Harsha Adapala Thirumala	Attention with Joint learning and Alignment (Coding)	35%

References:

Dataset: https://www.kaggle.com/datasets/hgultekin/paralel-translation-corpus-in-22-languages

Data Preprocessing:

- https://pytorch.org/text/stable/index.html
- https://pytorch.org/text/stable/index.html
- https://spacy.io
- https://pytorch.org/tutorials/beginner/translation_transformer.html

Seq-2-Seq:

- https://www.analyticsvidhya.com/blog/2020/08/a-simple-introduction-to-sequence-to-sequence-models/
- https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
- Sequence – to – Sequence Model for machine translation by Ilya Sutskever et. al. https://arxiv.org/pdf/1409.3215.pdf
- Techer Forcing: https://nishu-jain.medium.com/teacher-forcing-summary-f1bd790840ad#:~:text=With%20teacher%20forcing%2C%20the%20correct,rather%20than%20its%20own%20predictions.
- https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html
- Bleu Score: https://machinelearningmastery.com/calculate-bleu-score-for-text-python/
- Bleu Score: https://towardsdatascience.com/foundations-of-nlp-explained-bleu-score-and-wer-metrics-1a5ba06d812b

Bidirectional Seq-2-Seq:

- Neural Machine Translation by Jointly learning to Align and Translate by Dezmitry Bahdanau et. al. https://arxiv.org/pdf/1409.0473.pdf

Seq-2-Seq with Attention:

- https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
- https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html
- Neural Machine Translation by Jointly learning to Align and Translate by Dezmitry Bahdanau et. al. https://arxiv.org/pdf/1409.0473.pdf
- Effective approaches to Attention based Neural Networking by Minh-Thang Luong et. al. : https://arxiv.org/pdf/1508.04025.pdf
- Different attentions: https://towardsdatascience.com/sequence-2-sequence-model-with-attention-mechanism-9e9ca2a613a