Implementing Translator from Scratch

Abhishek Dubey
231110003
abhishekd23@iitk.ac.in

Puru Pant
231110040
purup23@iitk.ac.in

April 23, 2024

Abstract

This project presents the implementation of a Machine Translation system, named "Implementing Translator from Scratch", for translating English sentences to Hindi. Leveraging the Transformer architecture, renowned for its attention mechanism, the system achieves efficient handling of long-range dependencies. The dataset undergoes comprehensive preprocessing steps, including lowercasing, tokenization, and the addition of start and end tokens. The core of the system comprises Transformer encoder and decoder layers, trained on a dataset of English-Hindi sentence pairs. Extensive training and validation result in accurate translations, as evidenced by low loss values and high BLEU scores. "Translator from Scratch" highlights the effectiveness of Transformers in Machine Translation tasks and provides insights into future enhancements for such systems.

1 Introduction

In today's multilingual world, the need for effective translation systems is more pronounced than ever. Machine Translation (MT) systems have significantly advanced over the years, with deep learning techniques showing remarkable promise in this domain. This report delves into the implementation of a translator from scratch using deep learning methods. The project aims to develop a translation system capable of translating text from one language to another, without relying on pre-existing translation APIs or models. By building the translator from scratch, we gain a deeper understanding of the underlying concepts and mechanisms involved in machine translation.

2 Transformer Overview

In "Attention Is All You Need", Vaswani et al. introduced the Transformer, which introduces parallelism and enables models to learn long-range dependencies–thereby helping solve two key issues with RNNs: their slow speed of training and their difficulty in encoding long-range dependencies. Transformers are highly scalable and highly parallelizable, allowing for faster training, larger models, and better performance across vision and language tasks. Transformers are beginning to replace RNNs and LSTMs and may soon replace convolutions as well.

2.1 Why Transformers?

- Transformers are great for working with long input sequences since the attention calculation looks at all inputs. In contrast, RNNs struggle to encode long-range dependencies. LSTMs are much better at capturing long-range dependencies by using the input, output, and forget gates.
- Transformers can operate over unordered sets or ordered sequences with positional encodings (using positional encoding to add ordering the sets). In contrast, RNN/LSTM expect an ordered sequence of inputs.
- Transformers use parallel computation where all alignment and attention scores for all inputs can be done in parallel. In contrast, RNN/LSTM uses sequential computation since the hidden

state at a current timestep can only be computed after the previous states are calculated which makes them often slow to train.

2.2 Multi-Headed Attention

Let's refresh our concepts from the attention unit to help us with transformers.

- **Dot-Product Attention:**

$$c = \sum_{i=1}^{n} v_i \alpha_i \alpha_i = \frac{\exp(k_i^\top q)}{\sum_{j=1}^{n} \exp(k_j^\top q)}$$

- **Self-Attention:** we derive values, keys, and queries from the input

$$\begin{aligned} v_i &= V x_i \quad i \in \{1, \ldots, \ell\} \\ k_i &= K x_i \quad i \in \{1, \ldots, \ell\} \\ q_i &= Q x_i \quad i \in \{1, \ldots, \ell\} \end{aligned}$$

Combining the above two, we can now implement multi-headed scaled dot product attention for transformers.

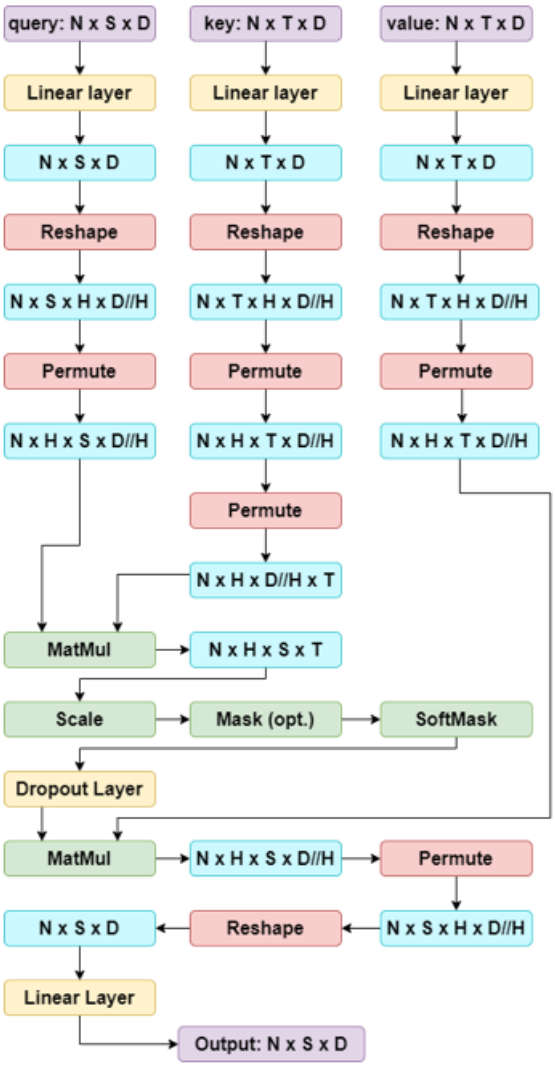
- **Multi-Headed Scaled Dot Product Attention:** We learn a parameter matrix $\mathbf{V}_i$, $\mathbf{K}_i$, $\mathbf{Q}_i \in \mathbb{R}^{D \times D}$ for each head i, which increases the model's expressivity to attend to different parts of the input. We apply a scaling term $\left(\frac{1}{\sqrt{d/h}}\right)$ to the dot-product attention described previously in order to reduce the effect of large magnitude vectors.

$$Y_i = \text{softmax}\left(\frac{(XQ_i)(XK_i)^\top}{\sqrt{d/h}}\right)(XV_i)$$

We can then apply dropout, generate the output of the attention layer, and finally add a linear transformation to the output of the attention operation, which allows the model to learn the relationship between heads, thereby improving the model's expressivity.

2.3 Step-by-Step Multi-Headed Attention with Intermediate Dimensions

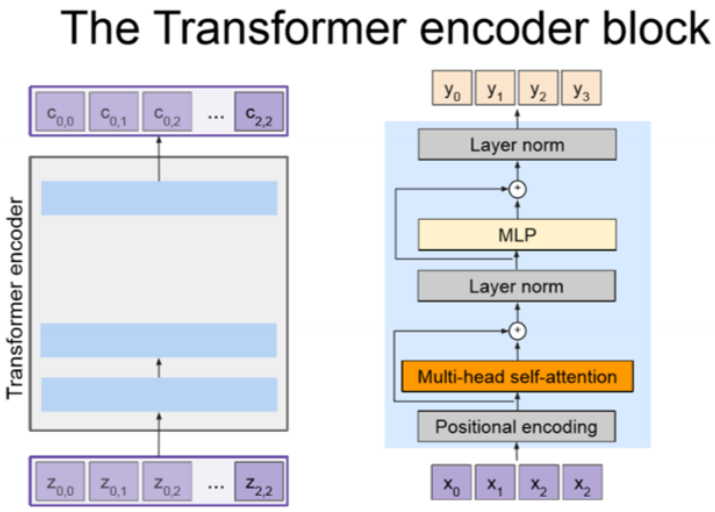
There's a lot happening throughout the Multi-Headed Attention so hopefully this chart will help further clarify the intermediate steps and how the dimensions change after each step!

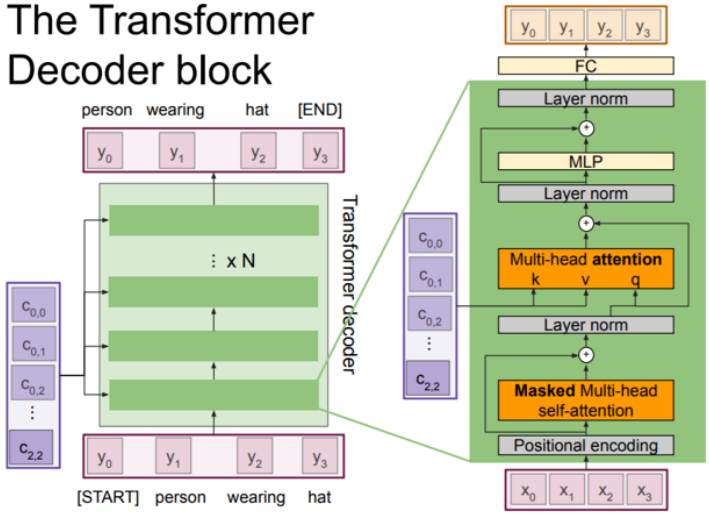


2.4 A couple tips on Permute and Reshape:

To create the multiple heads, we divide the embedding dimension by the number of heads and use Reshape (Ex: Reshape allows us to go from shape (N x S x D) to (N x S x H x D//H)). It is important to note that Reshape doesn't change the ordering of your data. It simply takes the original data and 'reshapes' it into the dimensions you provide. We use Permute (or can use Transpose) to rearrange the ordering of dimensions of the data (Ex: Permute allows us to rearrange the dimensions from (N x S x H x D//H) to (N x H x S x D//H)). Notice why we needed to use Permute before Reshaping after the final MatMul operation. Our current tensor had a shape of (N x H x S x D//H) but in order to reshape it to be (N x S x D) we needed to first ensure that the H and D//H dimensions are right next to each other because reshape doesn't change the ordering of the data. Therefore we use Permute first to rearrange the dimensions from (N x H x S x D//H) to (N x S x H x D//H) and then can use reshape to get the shape of (N x S x D).

2.5 Transformer Steps: Encoder-Decoder





Let's walk through the steps of the Decoder block!

- We take in the set of input vectors X and context vectors C (outputted from Encoder block)
- We then add positional encoding to the input vectors X.
- We pass the positional encoded vectors through the Masked Multi-head self-attention layer. The mask ensures that we only attend over previous inputs.
- We have a Residual Connection after this layer which allows us to bypass the attention layer if it's not needed.
- We then apply Layer Normalization on the output which normalizes each individual vector.
- Then we pass the output through another Multi-head attention layer which takes in the context vectors outputted by the Encoder block as well as the output of the Layer Normalization. In this step the Key comes from the set of context vectors C, the Value comes from the set of context vectors C, and the Query comes from the output of the Layer Normalization step.
- We then have another Residual Connection.
- Apply another Layer Normalization.
- Apply MLP over each vector individually.
- Another Residual Connection
- A final Layer Normalization

- And finally we pass the output through a Fully-connected layer which produces the final set of output vectors Y which is the output sequence.

3 Experiments

3.1 Data Preprocessing

Raw data has to be preprocessed to remove noise like non-ascii characters and punctuations and encode each line to ASCII and then decode to UTF-8. Some of the Hindi sentences had English alphabets in them. The cleaned data has 85000 lines. Since the sentence lengths can be quite large in corpus that can have adverse effect on sequence model, we constrain them by MAX_LENGTH = 20.
NLTK tokenizer and IndicNLP library are used to clean, normalize and tokenize the data. For English, replace everything with space except (a-z, A-Z, ".", "?", "!", ",", "-"). Extra spaces are removed. Abbreviations are expanded by mapping them through a dictionary.
The data is vectorized because this scales down the computation costs by several magnitudes. Using matrix operations for dealing with data will also prove beneficial in terms of performance gains.
"SOS" is used as start of sentence marker for target sequence and "EOS" is end of sentence marker. A class is used to map a word to its index and vice-versa. For the batch length to be same, sentences are padded using "EOS". Therefore, input sequences are represented as 2D-tensor of dimension ($batchsize * maxlength$).

3.2 Training Procedure

After dataset is preprocessed, it is split as 94% for training and 5.3% for validation.The training procedure involved training the translator model over 50 epochs. Each epoch consisted of processing 1250 batches of data.

For transformers,

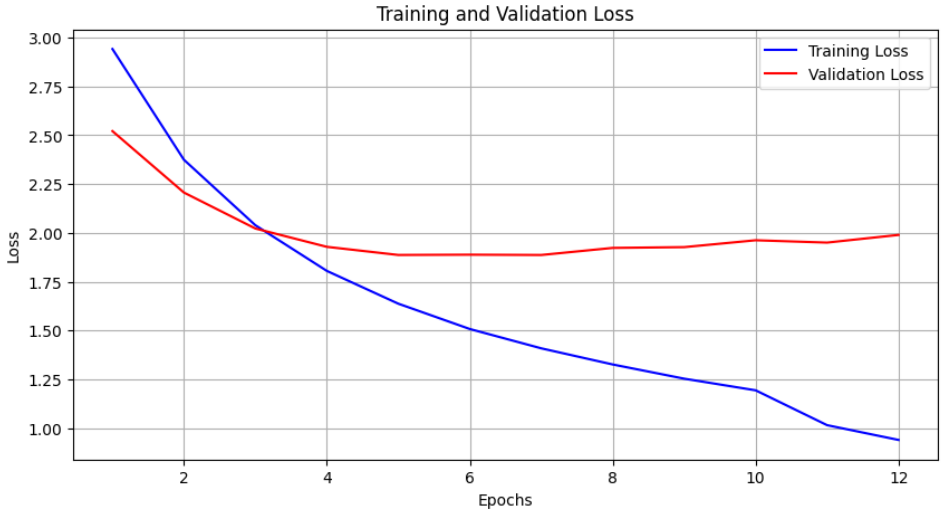
Hyperparamter	Value
Batch Size	64
Epochs	50
Embedding size	128
Learning rate	0.001
Optimizer	Adam
Dropout	0.1
Activation Function	RELU

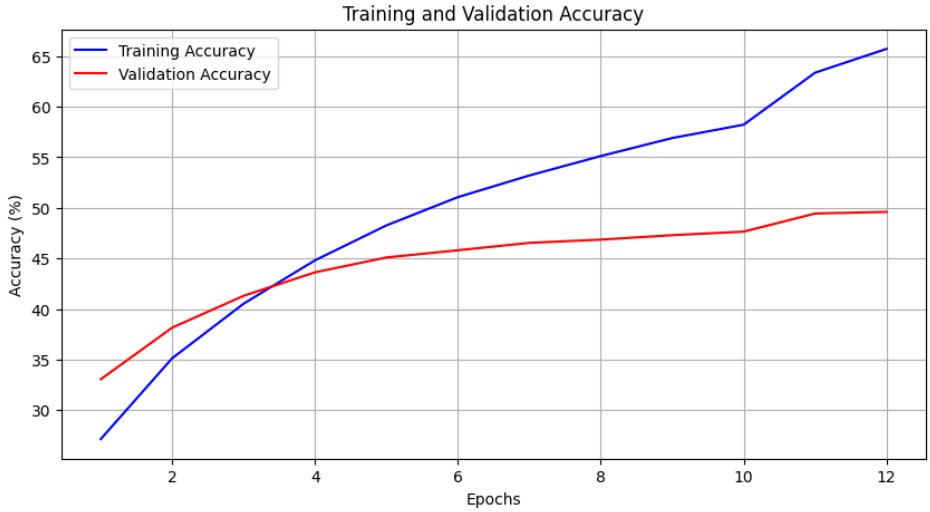
Greedy approach is used for predictions.

4 Results

The training process spanned 12 epochs, with the model's loss steadily decreasing from 2.94 to 0.94, and accuracy increasing from 27.15% to 65.71%. Similarly, validation loss decreased from 2.52 to 1.99, and validation accuracy rose from 33.06% to 49.60%. The BLEU score for test data is 0.350, indicating enhanced translation quality. These results demonstrate the effectiveness of the training procedure in improving both training and validation metrics, leading to better translation performance.

5 Plots





6 Conclusion

In conclusion, the implementation of the translator from scratch has shown promising results. Through 12 epochs of training, the model's loss steadily decreased, and accuracy improved significantly. Similarly, validation metrics also displayed a positive trend, indicating the model's ability to generalize well. The observed improvement in BLEU score for the test data suggests enhanced translation quality. These results underscore the effectiveness of the training procedure and highlight the model's capability to achieve better translation performance over time.
Moving forward, further optimization and experimentation with different architectures and hyperparameters could potentially lead to even better results.

7 Future Directions

1. Explore data augmentation techniques such as back-translation, word embeddings, or synthetic data generation to increase the diversity and quantity of training data, which could lead to better generalization and translation quality.
2. Use larger training set.

3. Fine-tune the pre-trained models on domain-specific data (e.g., medical, legal, technical) to improve translation quality in specialized domains.
4. Extend the model to perform multi-lingual translation by training it on multiple language pairs simultaneously. This can be achieved using techniques like zero-shot translation or one-to-many translation.

References

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).
2. Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002, July). BLEU: A method for automatic evaluation of machine translation. In Proceedings of the 40th annual meeting on association for computational linguistics (pp. 311-318). Association for Computational Linguistics.
3. Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.
4. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
5. Vaswani, A. (2018). The annotated transformer. arXiv preprint arXiv:1804.00247.