CS 6910 Fundamentals of Deep Learning

Assignment 4 Analysis Report

July 23, 2020

Submitted by **Group 19**

ME17B068 Soumyadeep Mondal, **ME17B084** Nishant Prabhu, **ME16B177** Saye Sharan

---

Contents

---

1 Image Captioning

In this task, we were asked to build a model that can produce a textual caption to describe a given image. The dataset given to us consisted of 4000 images taken from Flickr 8K dataset. Each image is associated with 5 captions, each describing the image in varying amounts of detail. To improve the performance of our model, we used all 8000 images, of which 5600 were used for training and 2800 for testing.

1.1 Data preprocessing

Images. Each image was resized to (256, 256, 3) and center-cropped to (224, 224, 3). This helps increase computation speed, while assuming that no crucial information is contained near the edges of the image. Each of the three channels is then normalized using means (0.485, 0.456, 0.406) and standard deviations (0.229, 0.224, 0.225) respectively. These settings have been seen to provide optimal performance on the ImageNet dataset, hence used here.

Captions. The captions are first cleaned of punctuation, numeric and special characters and converted to lowercase. Start and end tokens (`startseq` and `endseq` respectively) are added the ends of each sentence to indicate the beginning and ending of the sequence. Then, the `Tokenizer` class from TensorFlow's text preprocessing submodule is used to convert each word into an integer token, such that a sentence is represented by a sequence of integers. Since the sequences will have varying lengths, they are padded with zeros such that each sequence has length equal to the longest sentence in the dataset. In the vocabulary, the token zero does not correspond to any word.

1.2 Simple RNN based architecture

The first model utilizes a simple RNN as the recurrent unit for produce word predictions. **Figure 1** summarizes the model architure.

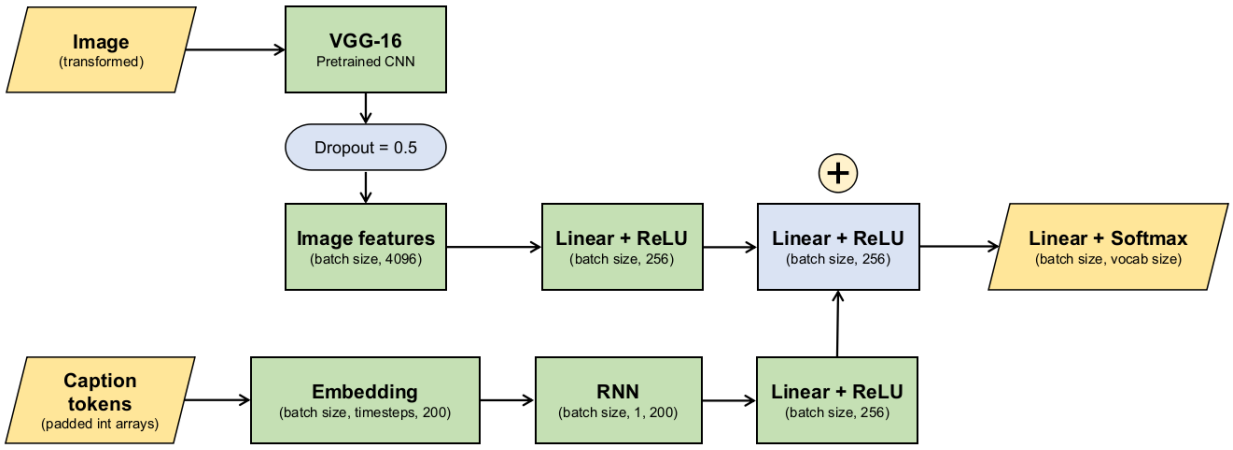

Figure 1: RNN based image captioning model

One training example consists of the image's features (extracted by a pre-trained VGG-16 network; 4096 dimensions per image) and a step-wise input of caption tokens i.e. the tokens until a particular timestep are input and the token at the next timestep is predicted by the model. This is explained more clearly in **Figure 2**.

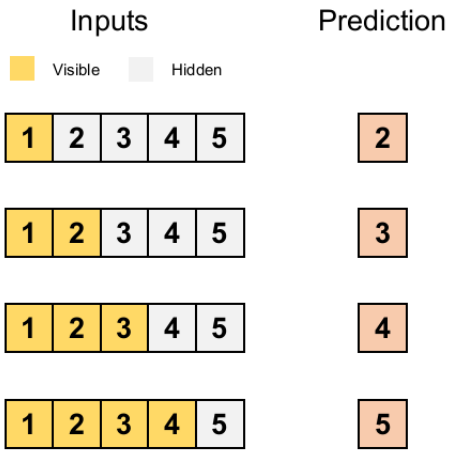

Figure 2: Input/output format of RNN layer

The model was trained for 80 epochs with a batch size of 16. Categorical crossentropy loss and Adam optimizer (constant learning rate of 0.001) was used for updating the model. Embedding layer was initialized with GloVe weights (200 dimensions).

The BLEU scores of the model on training and test data are shown in **Figure 3**.

	Training score	Test score
BLEU-1	0.358	0.349
BLEU-2	0.178	0.159
BLEU-3	0.125	0.090
BLEU-4	0.067	0.028

Figure 3: BLEU scores of RNN based captioning model

The scores indicate that the model is able to recognize a good number of individual words, but is struggling to form accurate bi-grams, tri-grams and 4-grams. This might indicate that the model is able to recognizes objects present in the image decently but cannot associate them in a coherent manner to form a meaningful sentence. For examples of its predictions, please check out `task1.ipynb` in the example notebooks.

1.3 LSTM based architecture

In the second model, we utilize LSTM units instead of RNN units in the recurrent layer of the model. Since LSTM layers can selectively read, forget and output incoming information, it should be able to better associate observed entities with distant words in the sentenced formed so far. Thus, we expect the LSTM based network to perform better than the RNN based model. **Figure 4** summarizes the architecture of this model.

The hidden and cell states of the LSTM units are randomly initialized. Input format for this network is same as that of the RNN based network. The model was trained for 80 epochs with a batch size of 16.

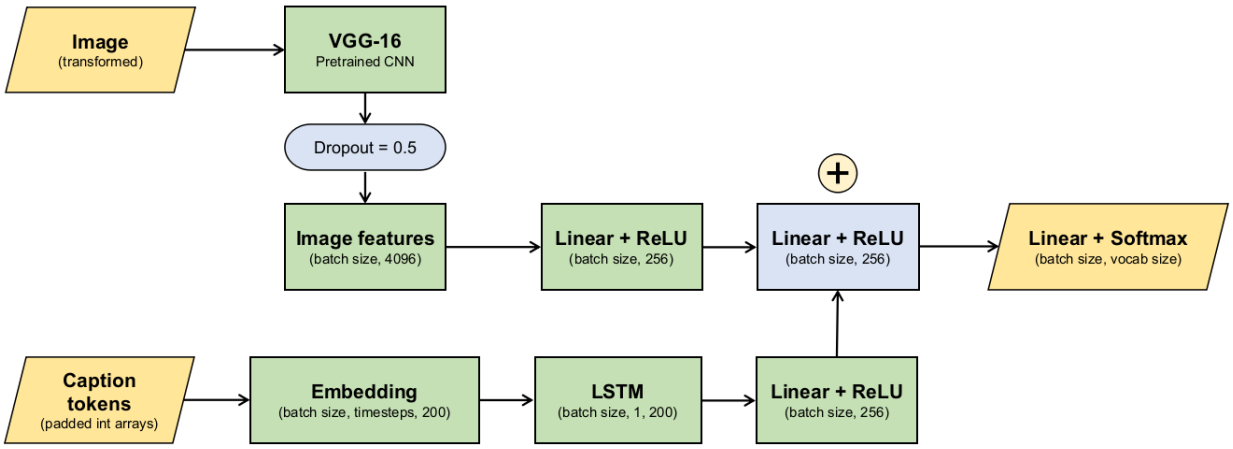

Figure 4: LSTM based image captioning model

Categorical crossentropy loss and Adam optimizer (constant learning rate of 0.001) were used to update the model. Embedding layer was initialized with GloVe weights (200 dimensions).

The BLEU scores of the model on training and test data are shown in **Figure 5**.

	Training score	Test score
BLEU-1	0.368	0.361
BLEU-2	0.189	0.174
BLEU-3	0.128	0.101
BLEU-4	0.068	0.036

Figure 5: BLEU scores of LSTM based captioning model

Comparison of the models. The BLEU scores of the LSTM model are marginally better than that of the RNN model. This is in agreement with our expectation of the LSTM model performing better than its counterpart. That being said, it also struggles to form coherent captions from observed image features. For examples of its predictions, please check out `task2.ipynb` in the sample notebooks.

1.4 Conclusion

The model implemented by us is very simplistic, which could be one of the reasons why it does not capture the required information from images at impressive levels. We tried other variations, including usage of image features as hidden state initialization for the recurrent units. However those resulted in the model overfitting on the data, which was visible from large gaps between training and test BLEU scores. This architecture performed the best among all of them.

2 Neural Machine Translation

In this task, we were asked to build a model which can translate given English sentences to Hindi. The dataset given to us consisted of more than 80,000 English sentences and their corresponding Hindi translations.

Note on the dataset. It is unfortunate that the dataset provided to us was not of the best quality there can be. A quick investigation of its contents reveals many bad translations, use of profane language and unicode characters glued to words which makes it all the more difficult to clean. That being said, we do not expect the models to perform very well, since there might be as many samples to confuse the model, as there are to train it.

2.1 Data preprocessing for recurrent models

Sentences of both languages are cleaned of punctuation, numbers (not many instances) and unresolved unicode characters (such as `\u200b` and `\i`). All sentences are converted to lowercase. Starting and ending tokens are appended at the ends of Hindi sentences, to facilitate prediction of the entire Hindi sentence during evaluation (the first word is predicted when `startseq` is provided and `endseq` is predicted when the model has finished translating the sentence). Separate tokenizers (using `Tokenizer` class from Tensorflow's text preprocessing submodule) are used to generate word-index mappings for english and hindi sentences. To ensure the lengths of all inputs are same, the inputs are padded with zeros so that they have the same length as the longest input in the dataset. Each sentence is now converted into a series of integer tokens.

Remarks. Considering the fact that both LSTMs and RNNs face trouble remembering very long sentences, we trained the model only on those examples which were at most 30 words long. On comparison with previous experiments where this was not imposed, we saw significant improvement in model performance on the test dataset (which has rather long sentences). Further more, to speed up training, we trained the model on 40,000 sentences out of the 70,000+ sentences available after the previous step.

2.2 LSTM based model

The first model contains LSTMs in its encoder and decoder. The hidden layers of these LSTMs were initialized with 256 units each. **Figure 6** shows the architecture of this model.

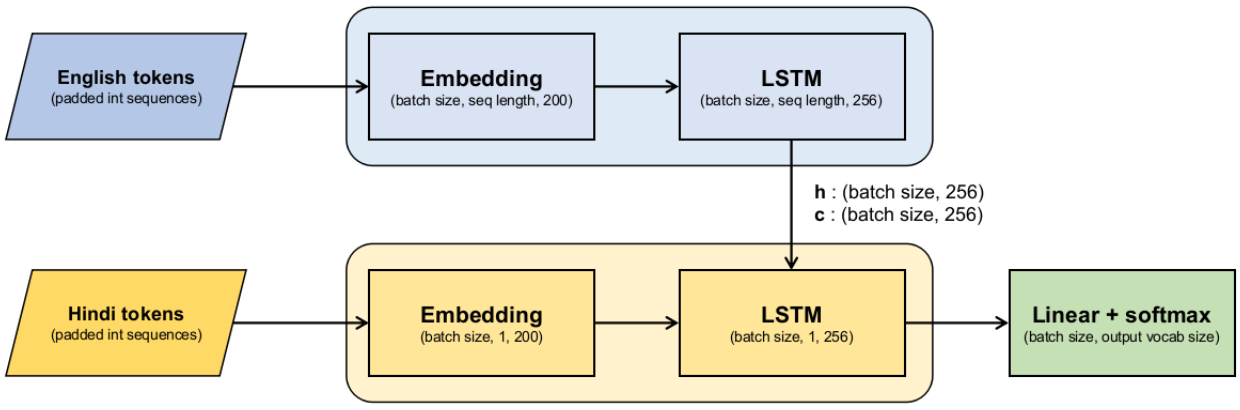

Figure 6: LSTM based NMT model architecture

The entire English sentence (tokens) is passed to the encoder, whose LSTM generates a tensor representation of this sentence using its hidden states at each time step. This is used to initialize the decoder's LSTM. Meanwhile, in the decoder, we sequentially pass a single token of the Hindi sentence, for which the decoder predicts the next token. We then employ **teacher forcing**, where the next token passed to the model is not its own previous prediction but the correct next token from the Hindi sentence. The model was trained for 30 epochs with roughly 900 steps per epoch and a batch size of 32. Categorical crossentropy loss and Adam optimizer were used to train the network. Encoder's embedding layer was initialized with GloVe embeddings (200 dimensions). The model's performance post training is shown in **Figure 7**.

	Training Score	Test Score
BLEU-1	0.371	0.207
BLEU-2	0.268	0.085
BLEU-3	0.236	0.053
BLEU-4	0.159	0.017

Figure 7: BLEU scores of LSTM based NMT model

While the model has been able to perform quite well on the training data, it struggles on the development set. One reason we observed for this behavior is the disparity in sentence lengths in the training and development sets (see **Figure 8**).

Figure 8: Histogram of word counts in training and development sets

On an average, the sentences in the development set are 2 to 2.5 times longer than those in the training set. Recurrent models fail to retain memory of distant words when sentences become very long. This clearly explains why the models do very well on the training set, but not on the development set.

2.3 LSTM model with attention

We add another component to the model which is supposed to help the model focus on the right words from the English sentence while it is translating it. This mechanism is usually used with bidirectional recurrent layers (so as to capture both forward and backward context); we have used them with forward LSTMs to prevent complication while training. **Figure 9** summarizes the architecture of this model.

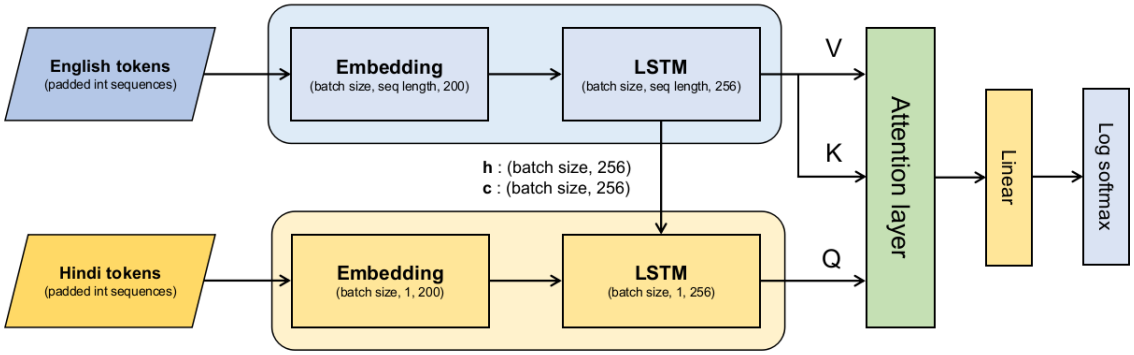

Figure 9: LSTM with attention model architecture

Scaled dot product attention has been used to generate attention weights for the model. We tried Bahdanau attention apart from this, but it did not perform as well as this. The weights are generated according to the expression below. Query (Q) is the decoder's hidden state for the current word; keys (K) and values (V) are both the encoders hidden states of the English sentence.

$$\text{Attention weights } (Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_{model}}}\right) \cdot V$$

The input format remains the same as in the previous case. The model was trained for 30 epochs with roughly 900 steps per epoch and a batch size of 32. Categorical crossentropy loss and Adam optimizer were used to train the network. Encoder's embedding layer was initialized with GloVe embeddings (200 dimensions). The BLEU scores of the model are shown in **Figure 10**.

	Training Score	Test Score
BLEU-1	0.362	0.198
BLEU-2	0.270	0.082
BLEU-3	0.242	0.051
BLEU-4	0.172	0.018

Figure 10: BLEU scores of LSTM based NMT model with attention

There are two prominent observations to make here.

- On the training data, the BLEU scores of the model have improved overall (for bi-grams and above). This indicates that for small sentences, the attention mechanism has helped the model get better at forming more coherent sentences out of the observations made from input sentence features. However, the uni-gram BLEU score has marginally dropped, meaning attention does not help the model there.
- BLEU scores on development dataset have dropped further. This is could have happened because the model now has to train more parameters (attention weights) with the same amount of data, leading to incomplete tuning. Also, the attention mechanism could have attended to the wrong words causing bad translations.

Figure 11 shows an example of the model's translation and the attention weights it used for the same. Brighter cells imply more attention to those words.

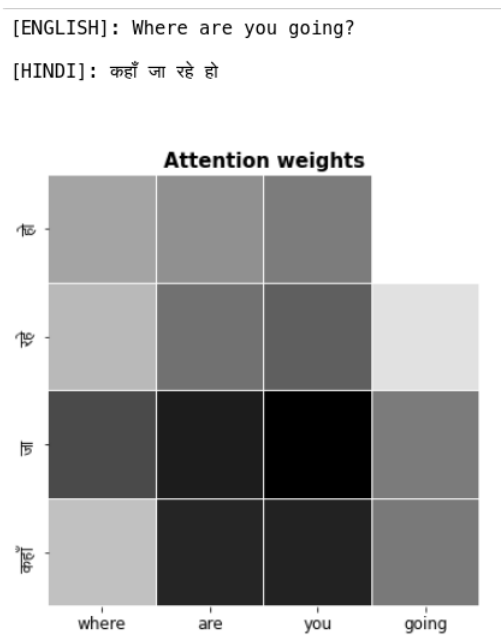

Figure 11: LSTM with attention: good translation and attention weights

Perhaps it is only fair to show the reader an example of where the model did not translate the sentence correctly. **Figure 12** is an example of a bad translation by the model. The attention weights show no clear pattern, meaning the model was confused as to what words to attend to all along. Unlike with other languages like French or German, sentence construction in English and Hindi is quite different. It is necessary that the attention mechanism captures this distinct grammatical syntax to ensure good translation. In our case, however, it appears that it has failed to do so.

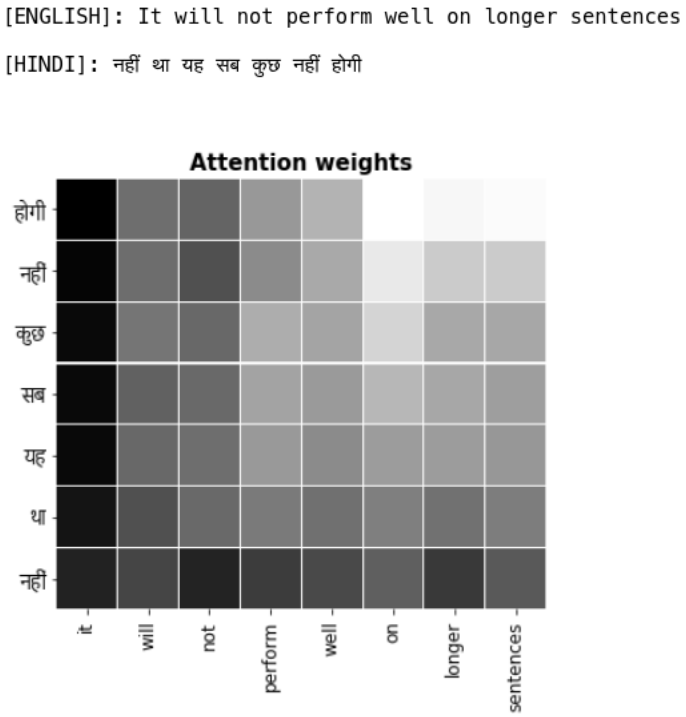

Figure 12: LSTM with attention: bad translation and attention weights

2.4 Transformer model

Vaswani et. al. from Google Brain made a major breakthrough in late 2017 with their amazing publication titled *Attention Is All You Need*, where they introduced the concepts of self-attention and the Transformer model which completely transformed the field of natural language processing (only to do it again with the release of *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* last year). In this final task, we construct a transformer model and train it from scratch on our dataset.

Remarks. This model consists of many more parameters than the ones discussed above, and thus required significantly larger amount of data to train properly. Usually this model is trained on corpora 1000s of words long with tens of thousands of examples. However, due to lack of quality data (and time) we have trained it the same way we did for the previous two models. Understandably, it could not learn much from it (as the reader will see shortly). Unlike the previous cases, we removed the restriction of word limit on the sentences and used all of the 80,000+ sentences that were provided as training data. This model completed 20 epochs on an NVIDIA GeForce RTX 2060 S in about 2 hours.

Since this model's architecture is more involved than the other two, a brief discussion about it's construction is presented below.

2.4.1 Model architecure and data preprocessing

Figure 13 shows the architecture of the model constructed for this task. It is exactly the same as the one presented in the publication cited prior.

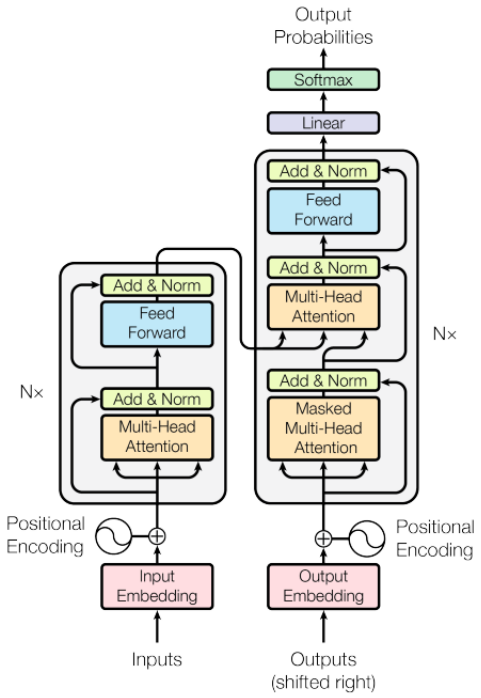

Figure 13: The transformer model. Adapted from *Attention Is All You Need*.

The model is provided tokenized sentences (preprocessed externally) and a positional encoding matrix, generated according to the expressions suggested in the paper.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{1000^{2i/d_{model}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{1000^{2i/d_{model}}}\right)$$

When plotted for a sample tensor of sequence length 20 and $d_{model} = 256$, the positional encoding looks like **Figure 14**. This encoding is independent of the content of the input sequences.

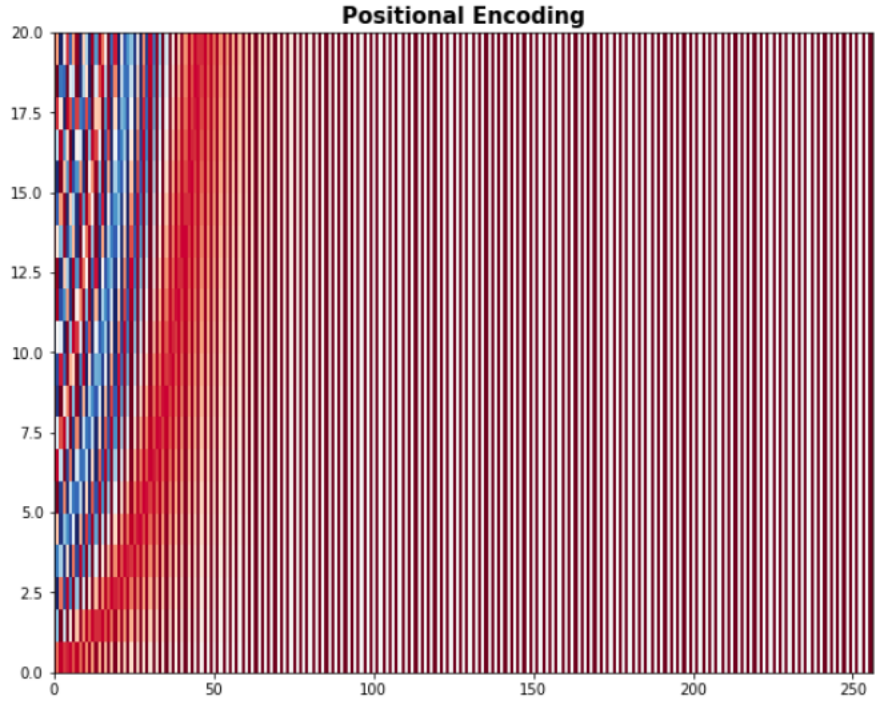

Figure 14: Positional encoding for 20 tokens with 256 features each

Scaled dot product attention is used for calculating self attention in all attention heads in encoders and decoders, with the expression below. In our case, d_{model} was set to 512 features.

$$\text{Attention weights } (Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_{model}}}\right) \cdot V$$

Our model comprises of an encoder stack with 2 encoder layers and a decoder stack with 2 decoder layers. Each layer has self attention units (masked and otherwise) with 4 attention heads each. All feedforward layers have $d_{model}/2$ hidden units with ReLU activation.

Masking is performed in both encoder and decoder layers to prevent the model from (1) learning information from padding tokens, and (2) cheating with prediction by looking at future words in the decoder's input. The padding mask sets all padding tokens to a high negative value (close to $-\infty$). The look-ahead mask (for the masked multi-head attention module in the decoder layers) sets all tokens to a large negative value which are in the future of the current time step. On application of a lookahead mask, the input to the said module will look like **Figure 15**.

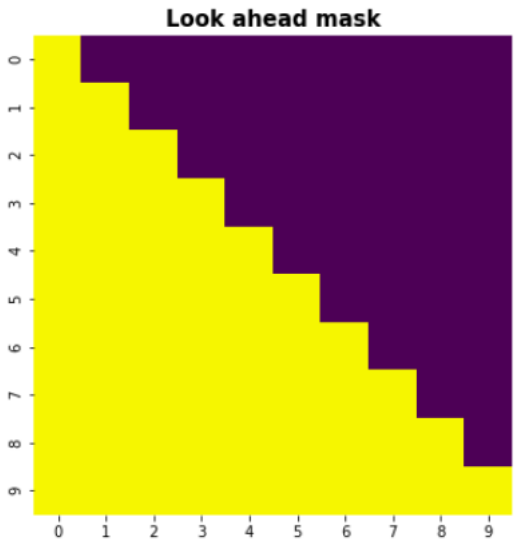

Figure 15: Look-ahead mask for 20 tokens. Yellow regions are visible; purple regsions are masked

2.4.2 Training and results

The model was trained on 80,000+ sentences for 20 epochs with a batch size of 64. Constant learning rate of 0.0003 was used with Adam optimizer and nonlinear logloss (after log-softmax) was used to update the model. Encoder embedding layers were initialized with GloVe embeddings (200 dimensions). The BLEU scores of the model are shown in **Figure 16**.

	Training Score	Test Score
BLEU-1	0.081	0.094
BLEU-2	0.027	0.028
BLEU-3	0.014	0.012
BLEU-4	0.003	0.000

Figure 16: BLEU scores for transformer model

As expected, the model has failed to learn anything from the data. Also note that the model does better in uni-gram and bi-gram prediction on the test data, which has longer sentence inputs than the training data. This, however, could just be a statistical fluke.

The figures below show a sample prediction from the model. The uppermost layer of the encoder stack generates the key and value vectors which are based to the multi-head attention module of the decoder. Also, the uppermost layer of the decoder stack generates the output probabilities of the next word. Thus, we visualize the attentions weights generated by these two layers as well.

```
[ENGLISH]: This one did not do so well
[HINDI]: यह अच्छी हम पर हम
```

Figure 17: Example translation from tranformer model

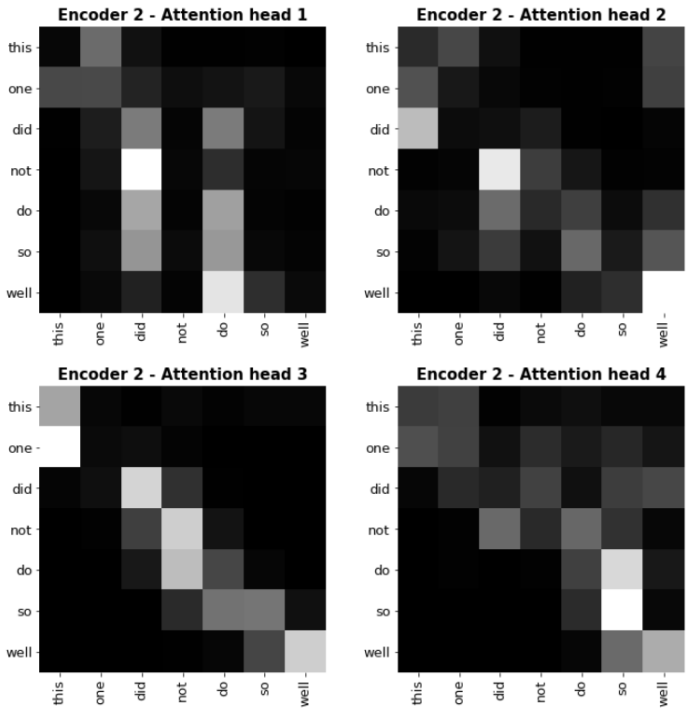

Figure 18: Topmost encoder self attention weights

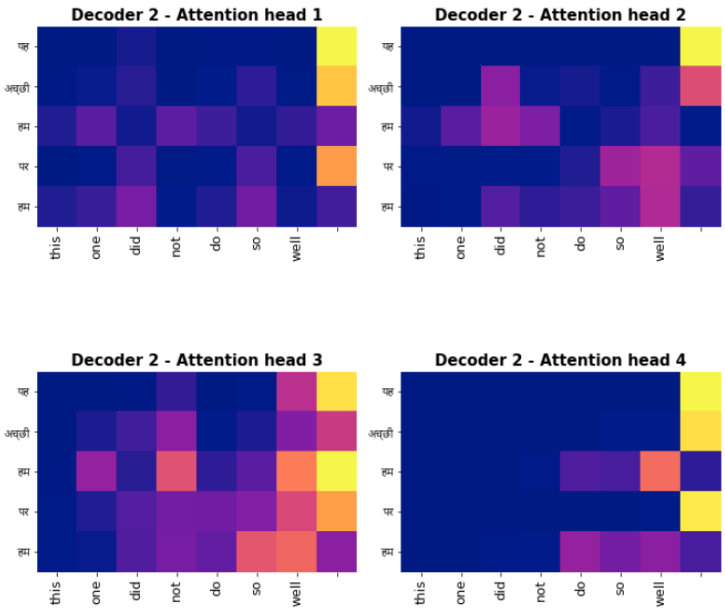

Figure 19: Topmost decoder attention weights

The encoder self-attention weights indicate that the encoder might have been doing its job well. The focused words are not linear i.e. the model doesn't only focus on the current word while masking predictions. However, the decoder weights are very sparse and the wrong words have been attended to in all attention heads. This is possibly due to the quality of the input data, which does not show any clear mapping between the English and Hindi word-spaces. The BLEU scores of the model, visibly, are dismal.

2.5 Conclusion

The following ideas can be discerned from the results stated above.

- For small amount of training data available, recurrent models perform better than transformer models for sentences which are not too long.
- If the transformer model is pre-trained with larger corpora with sufficient examples to map many phrases to their accurate translations, the transformer might outperform recurrent models for sequences of any length.
- Attention mechanism helps models generate more coherent text, but might not be helping them find correct predictions given some context.

3 Endnote

This assignment, in particular, has been a great learning experience to work upon. It has made it possible for us to explore and recreate state-of-the-art models to perform complex tasks in the deep learning sphere. The experience we have gained in working on this assignment will go a long way in helping us advance our endeavors in this field.

4 References

Image Captioning

- How to develop a deep learning photo caption generator from scratch, Jason Brownlee on *Machine Learning Mastery*
- Learn to build image caption generator with CNN & LSTM, *Data Flair*

Neural Machine Translation

- Neural machine translation with attention, *Tensorflow Core*
- Attention Is All You Need, Vaswani et. al., *arXiv*
- Transformer model for language understanding, *Tensorflow Core*