E1:246 Natural Language Understanding - Assignment 2
Neural Machine Translation Using Attention Mechanisms

Rahul Bansal
M.tech Ist year
rahulbansal@iisc.ac.in

Abstract

Sequence to sequence models often belong to a family of encoder decoder archirecture. Encoder encode a source sentence into a fixed-length vector from which a decoder generates a translation. With attention mechanism, each time the decoder model predicts an output word, it only uses parts of an input where the most relevant information is concentrated instead of an entire sentence. In this assignment, different kind of attention mechanisms are explored for machine translation task.

1 Introduction

Traditional phrase-based translation systems(Koehn et al., 2003) performed their task by breaking up source sentences into multiple chunks and then translated them phrase-by-phrase. This led to disfluency in the translation outputs and was not quite like how we, humans, translate. We read the entire source sentence, understand its meaning, and then produce a translation. Neural Machine Translation (NMT) mimics that! NMT system first reads the source sentence using an encoder to build a "context vector", a sequence of numbers that represents the sentence meaning; a decoder, then, processes the sentence vector to emit a translation, as illustrated in Figure 1. This is often referred to as the encoder-decoder architecture(Sutskever et al., 2014).

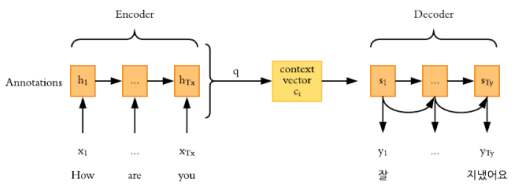

Figure 1: Encoder-Decoder Layout
Courtesy: Medium.com

With attention mechanism(Bahdanau et al., 2015), Decoder searches for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and all the previous generated target words. The most important distinguishing feature of this approach from the basic encoder-decoder is that it does not attempt to encode a whole input sentence into a single fixed-length vector. Instead, it encodes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding the translation. This frees a neural translation model from having to squash all the information of a source sentence, regardless of its length, into a fixed-length vector.

2 Encoder-Decoder

Here, we describe briefly the underlying framework, called RNN EncoderDecoder(Sutskever et al., 2014), upon which we implement attention mechanism.
In Figure 1 :
$x_1, x_2...x_{T_x}$ - Word embeddings of a source sentence.
$h_1, h_2...h_{T_x}$ - Encoder hidden states.
c - Context vector generated from the sequence of the encoder hidden states.
$s_1, s_2...s_{T_y}$ - Decoder hidden states.
$y_1, y_2...y_{T_y}$ - Translated words.

$$h_t = f(x_t, h_{t-1})$$
$$c = q(h_1, .., h_{T_x})$$

where h_t is a hidden state at time t, and c is a context vector generated from the sequence of the hidden states. f and q are some nonlinear functions. In this assignment, one directional GRU is used to implement the encoder model.

The decoder is often trained to predict the next word y_t given the context vector and all the previously predicted words $y_1, y_2...y_{t-1}$.
With RNN, conditional probabilty of each predicted word is modeled as:

$$p(y_t|y_1, y_2...y_{t-1}, c) = g(y_{t-1}, s_t, c)$$

where g is a nonlinear function that outputs the probability of y_t given the decoder hidden state s_t, last predicted word y_{t-1} and context vector c.

3 Attention

The basic idea is that each time the decoder model predicts an output word, it only uses parts of an input where the most relevant information is concentrated instead of an entire sentence.

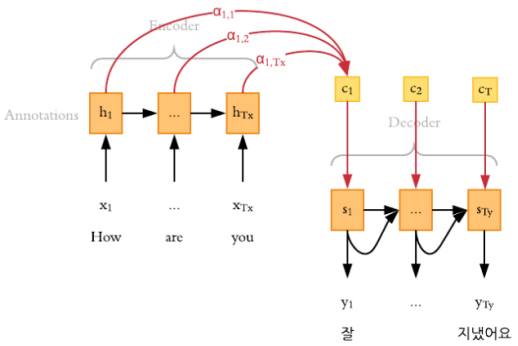

Figure 2: Encoder-Decoder With Attention
Courtesy: Medium.com

Encoder works as usual, and the difference is only on the decoders part. As it can be observed from Figure 2, the decoders hidden state is computed with a context vector, the previous output and the previous hidden state. But here single context vector c is not used, instead a separate context vector c_i for each target word.
With the attention mechanism in place :

$$p(y_i|y_1, y_2...y_{i-1}, c) = g(y_{i-1}, s_i, c_i)$$
$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

The context vector is computed as the weighted sum of encoder hidden states $(h_1, h_2...h_{T_x})$.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

The α_{ij} are attention weights for predicting i^{th} word in the output sequence, which are computed as:

$$\alpha_{ij} = \frac{\exp\left(score(i,j)\right)}{\sum_{k=1}^{T_x} \exp\left(score(i,k)\right)}$$

There are differnt mechanisms to calculate $score(i,j)$ which will be explored in the next subsections.

3.1 Additive Attention

s_{i-1} being previous hidden state of decoder and h_j being encoder hidden state at $t = j$. $score(i,j)$ is calculated given s_{i-1} and h_j.

$$score(i,j) = v^T(\tanh(W_1(h_j) + W_2(s_{i-1})))$$

where v, W_1, W_2 are learnable parameters.

3.2 Multiplicative Attention

Multiplicative attention(Luong et al., 2015) simplifies the calucation of $score(i,j)$ by calculating the following function:

$$score(i,j) = h_j^T W(s_{i-1})$$

where W is a learnable parameter.

3.3 Self Attention

To predict i^{th} target word, attention weights $(\alpha_{i,j})$ needs to be calculated. And $score(i,j)$ to be calculated for $\alpha_{i,j}$.
In additive and multicative attention, it is seen that s_{i-1}(decoder's hidden state) is being incorporated to calculate $score(i,j)$. But self attention mechanism claims that without incorporating s_{i-1}, relevant aspects from the sentence can still be extracted by allowing it to attend to itself.

$$score(i,j) = v^T \tanh(W(h_j))$$

where v, W is a learnable parameter.

3.4 Key-value attention

Key-value attention[5] splits encoder's hidden state (h_j) into a key k_j and a value v_j. The keys are used for calculating the attention distribution $\alpha_{i,j}$ using additive attention.

$$score(i,j) = v_a^T(\tanh(W_1(k_j) + W_2(s_{i-1})))$$

$$\alpha_{ij} = \frac{\exp\left(score(i,j)\right)}{\sum_{k=i}^{i-L} \exp\left(score(i,k)\right)}$$

where v_a, W_1, W_2 are learnable parameters. L is the attention window i.e. source sentence window in which most relevant information is present.
To predict i^{th} target word, context vector c_i is calculated as follows:

$$c_i = \sum_{j=i}^{i-L} \alpha_{ij} h_j$$

Note for calculation of attention weights
It is not efficient to calculate each $score(i,j)$ individually. However, using matrix calculus, all $scores$ can be calculated at once and $softmax$ operation is applied on $scores$ to calculate attention weights (α_{ij}).

4 DataSet

Above explained attention mechanism are evaluated on the task of English-to-German translation. Europarl bilingual parallel corpora provided by ACL WMT'14 is used. Data contains $2M$ sentences of each language. Models are trained on 90% of the sentences and rest 10% are used for evaluation i.e. to calculate bleu score.
Same dataset and same training procedures are used to train all Models, the difference is just in the calculation of attention weights.

5 Implementation

5.1 Preprocessing

After converting all the sentences into lower case, $30,000$ most frequent words from each language are shortlisted to train models. Any word not included in the shortlist is mapped to a special token ([UNK]). The same procedure mentioned in the paper by Bahadanau[2]. "start" and "end" tokens are applied to each sentence in both the languages.
After forming the vocabulary for each language, all words in each language are mapped to single numeral value, so every sentence is nothing but an array of numbers.

5.2 Encoder Model

Two layers are used in Encoder Model.
tf.keras.layers.Embedding - This layer is used to get embedding for each word of source sentence.
tf.keras.layers.GRU(Units) - Unidirectional GRU layer is used to generate hidden states h_j corresponding to each word of source sentence.

5.3 Attention Model

Attention Model takes s_{i-1} (last decoder hidden state) and $(h_1, h_2...)$ (Encoder hidden states) as an argument and return c_i (context vector) and attention weights.

5.4 Decoder Model

Attention Model is called in the decoder itself by passing s_{i-1} (last decoder hidden state) and $(h_1, h_2...)$ (Encoder hidden states).
Three layers are used in Decoder Model.
tf.keras.layers.Embedding - This layer is used to get embedding for each word of target sentence.
tf.keras.layers.GRU(Units) - While predicting i^{th} word, c_i is concatenated with last target word embedding (y_{i-1}) and passed through unidirectional GRU layer to generate hidden state s_i and output.
tf.keras.layers.Dense(vocab size) - The output generated through GRU layer is passed through dense layer and softmax operation is applied on it to get probability value corresponding to each word in vocabulary.

5.5 Loss

Softmax loss is used between predicted word and actual target word for each word while training.

5.6 Training

While training, The actual $(i-1)^{th}$ word is fed with c_i into Decoder GRU to predict i^{th} word. If that were not done then mistake made for one word would propagate to next words. However, while evaluating, the observed $(i-1)^{th}$ word is fed with c_i into Decoder GRU to predict i^{th} word. All models are trained for 5 epochs. It took almost 7000 seconds for each epoch(Git).

6 Hyperparameters

Vocab Size - $30,000$ most frequent words from each language are shortlisted to train models. Any word not included in the shortlist is mapped to a special token ([UNK]). The same procedure mentioned in the paper by Bahadanau[2].

Sentence Length - All words are given a single numeral value, so every sentence is nothing but an array of numbers. All sentences in both languages are normalized to length of 20. Sentences of very shorter length are padded with zero's at the end and initial words are removed for sentences having longer length.

Embedding dimensions - size of word embedding is taken to be 128 for both input sentence and target sentence word. Embeddings are trainable.

GRU Units - GRU units is the dimensionality of its output space. It is taken to be 512 for both encoder GRU as well as well decoder GRU.

Batch Size - Batch Size is taken to be 128. That means 128 sentences are processed simultaneously.

7 Results

Below Table shows the translational performance of the models with different attentions measured in bleu score. Bleu score is calculated for 5000 candidate translation of source validation sentences.

Attention	Bleu Score
Additive attention	19.97
Multiplicative attention	18.81
Self attention	6.94
Key-value attention	–

Models with additive attention and multiplicative attention are trained well enough to give good bleu score, but self attention and key-value attention are not well trained enough to give satisfactory results.

7.1 Qualitative results

Source Sentence - europe is experiencing economic and financial difficulties at present .
Translation with Additive attention - europa wird derzeit wirtschafts und finanziellen schwierigkeiten .
Translation with Multiplicative attention - europa durchlebt derzeit wirtschaftliche und finanzielle schwierigkeiten .
Translation with Self attention - europa derzeit erleben derzeit erleben derzeit

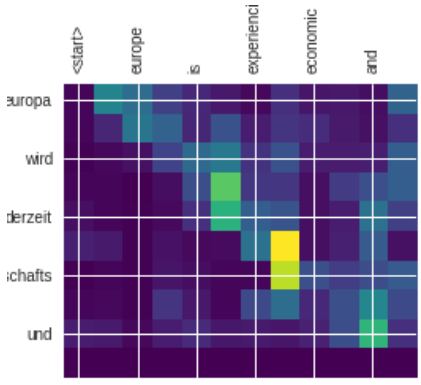

Figure 3: Attention plot with additive attention

As it can be seen that plots for additive attention and multiplicative attention are well distributed, but plots for self attention are not satisfactory. One reason for self attention not giving satisfactory performance could be due to not incorporating s_i (decoder's hidden state).
Last two model are not trained well enough to give satisfactory results.

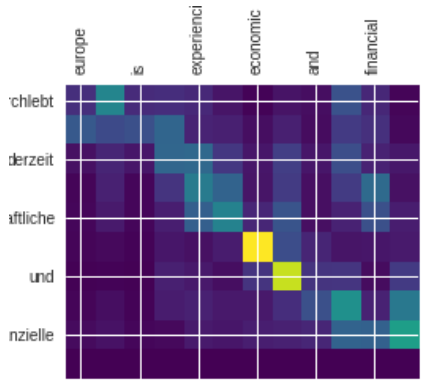

Figure 4: Attention plot with multiplicative attention

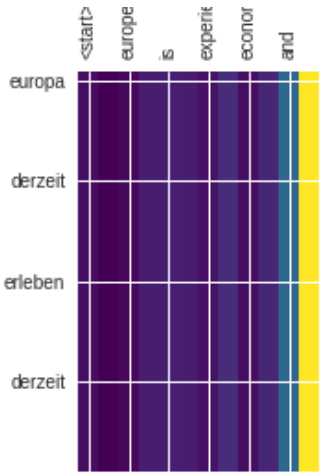

Figure 5: Attention plot with self attention

7.2 Translations II

Source Sentence - mr president , i shall be pleased to answer the request put to me .
Translation with Additive attention - herr prasident , ich mochte mich auf die beantwortung der beantwortung mir zu beantworten . see Figure 6.
Translation with Multiplicative attention - herr prasident , ich werde die bitte die anfrage gestellten antrag zu beantworten . see Figure 7.
Translation with Self attention - herr prasident , ich werde mich auf die mir die mir die mir die mir die mir die mir die. see Figure 8.

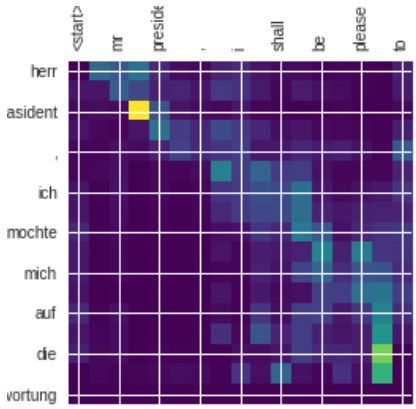

Figure 6: Attention plot with additive attention

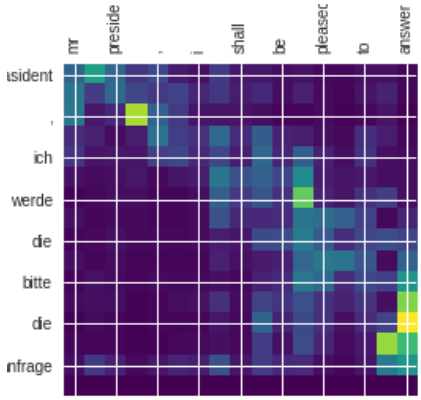

Figure 7: Attention plot with multiplicative attention

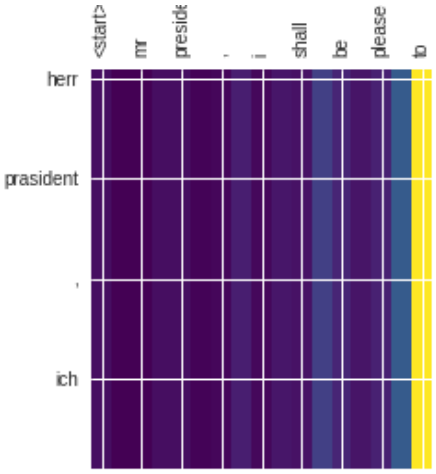

Figure 8: Attention plot with self attention

7.3 Translations III

Source Sentence - the problems with fishmeal lie elsewhere .

Translation with Additive attention - die probleme mit fischmehl liegen . see Figure 9.
Translation with Multiplicative attention - die probleme bei der unk liegt woanders . see Figure 10.
Translation with Self attention - die fischmehl . see Figure 11.

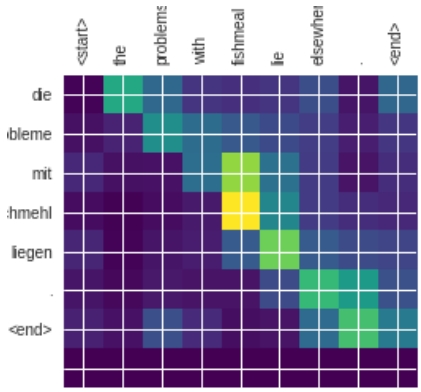

Figure 9: Attention plot with additive attention

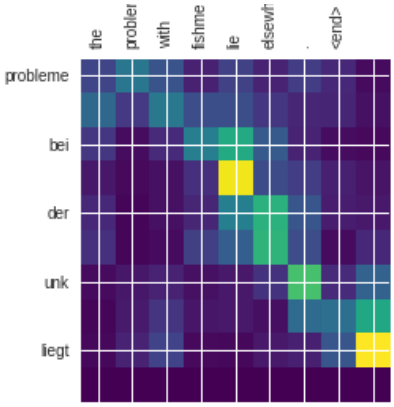

Figure 10: Attention plot with multiplicative attention

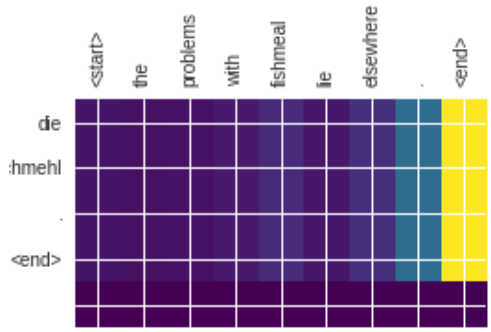

Figure 11: Attention plot with self attention

References

Link to github code and report.

Dzmitry Bahdanau, KyungHyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks.