Research Design Report

Devashish Kamble

1 Introduction

1.1 Task

The task that I choose to perform is that of Neural Machine Translation using a seq2seq model with additive attention mechanism proposed in the paper *'NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE'* by Dzmitry Bahdanau, KyungHyun Cho and Yoshua Bengio [1]. This paper that came out in 2014 was a seminal work behind the now-famous **"Attention Mechanism"**. My hypothesis is that at inference time, the beam search decoding outperforms greedy decoding and this would be tested in this proposal.

1.2 Neural Machine Translation

Neural Machine Translation (NMT) is a way to do Machine Translation with a single neural network. The task to solve in MT is to create an algorithm that can automatically translate sentences and phrases between two different languages. Translation can be thought of as finding the target sequence that is the most probable given the input; formally, the target sequence that maximises the conditional probability $y^* = \arg\max_y p(y|x)$[8].

In machine translation, the model learns a function, $p(y|x, \theta)$ with some parameters θ and then find its argmax for a given input: $y' = \arg\max_y p(y|x, \theta)$. To define a machine translation system, we need to answer three questions:

1. modelling - how does the model for p(y—x, θ) look like?
2. learning - how to find the parameters θ
3. inference - how to find the best y

The neural network architecture used to solve this task is called sequence-to-sequence (aka seq2seq).

2 Data

2.1 Dataset

Bahdanau et al [1] made an evaluation on the task of translating from English to French. For this purpose, they used the **ACL WMT'14** dataset which is a bilingual, parallel corpora consisting of Europarl (61M words), News commentary (5.5M), UN (421M) and two crawled corpora of 90M and 272.5M words, totaling 850M words. They reduced the size of the combined corpus to have 348M words and concatenated news-test-2012 and news-test-2013 to make a validation set, and evaluate the models on the test set (news-test-2014).

I propose to incorporate a dataset of low-resource language viz. Hungarian. The data set comes from Tatoeba [7], a collection of crowdsourced sentence pairs in different languages. The Hungarian-English pairs are processed, shuffled and split into train (70%), validation(15%) and test sets(15%). Before any preprocessing, each pair consists of a Hungarian sentence followed by its English translation which are separated by a *sep* delimiter. The dataset contains just over 100K pairs which is very less as compared to the original paper but would suffice for the task. The source and target sentences are separated into two lists i.e the source hungarian sentences will be in one list and the corresponding target english translations will be in another in order to perform respective preprocessing steps on them.

2.2 Data Preprocessing

Languages like Hungarian involve accent marks in their writing and this can be a hurdle for the NLP models, therefore, this needs to be taken care of during the pre-processing step by making use of Unicode Normalization, in short by converting these different representations into the same representation. One of the ways to do this is by declaring a function which takes any unicode and decomposes any accented characters into the same set of Unicode, and then replaces them with their ASCII equivalents.

A few additional pre-processing steps involve retaining important punctuation such as question marks and periods by using regex to treat them as separate tokens while also adding a start of sentence token *sos* to the beginning of each sentence and an end of sentence token *eos* to the end of each sentence. It is also beneficial to initialise the source tokenizer with an out of vocabulary token called *unk* as the tokenizer will fit only on the training set, but when the tokenizer encounters the test set it may find tokens it has not seen before which will then be replaced with *unk*.

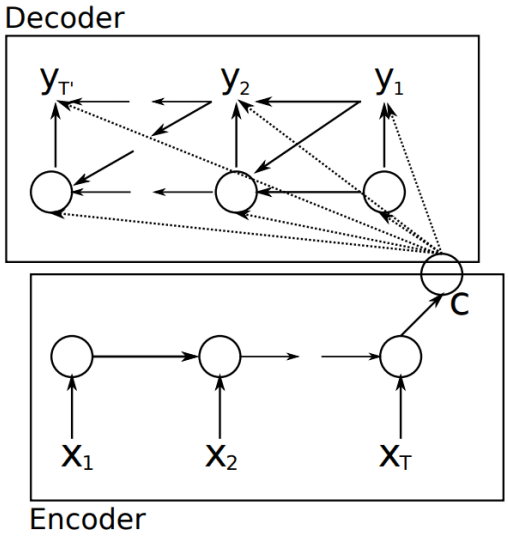
It is possible that the target language (Hungarian) vocabulary might be smaller than the source language (English) as there could be many different target words or phrases mapping to the same source words and phrases.

3 Model

3.1 Seq2seq

A seq2seq model takes an input of indeterminate length and outputs another sequence of indeterminate length so the lengths of the input and output don't have to match. It consists of two sub-models, one of them is called the encoder which is responsible for processing the input sequence while the other is the decoder which is responsible for generating the output sequence and hence they are also referred to as encoder-decoder models.

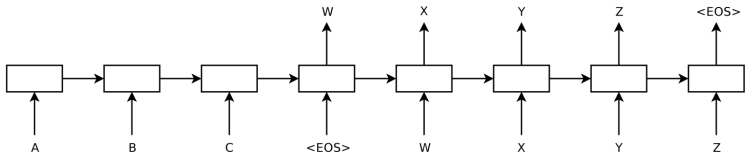
Cho et al [2] came with the novel architecture of RNN-based Encoder–Decoder neural network for the task of sequence modelling where one RNN encoded a variable-length input into a fixed-size vector while the other RNN subsequently decoded it into another variable-length output sequence.



Credits: Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

Sutskever et al's paper *Sequence to Sequence Learning with Neural Networks* [6] introduced the seq2seq model in order to solve the Machine Translation problem in a way that requires no knowledge of grammar and syntax and requires no pre-processing and hand-engineering. The model was built using LSTM networks to encode and decode variable-length sequences where the weights of these RNNs were different since they were learning sequential patterns over the two different languages and taking two entirely different vocabularies as input/output. The encoder reads the input sequence and

outputs the final hidden state which can be thought of as transforming the input embedding into a lower dimensional vector space. This final hidden state then serves as the input for the decoder which then proceeds to predict the translated sequence. Also, the input sequence was reversed in order to make the optimization problem easier. The thing to note is that the final hidden state embedding is the only link between the two components of the model and if the translation is successful then it means that all the semantic information about the sentences is solely represented by it. Thus, this particular embedding acts as a language-agnostic representation by encoding the meaning of the sentence that is independent of both the source and target language but rather lies between the two, which is why the authors refer to this representation as the 'thought' or 'context' vector.



Credits: Sequence to Sequence Learning with Neural Networks

To summarise, with seq2seq, we have an encoder RNN that produces a sequence of hidden state vectors $(h_1, h_2, ..., hN)$ for a length-N input sequence of words (or word vectors) $(w_1, w_2, ..., wN)$. As the encoder is implemented with an RNN, the mechanism for producing each hidden state h_t is dependent upon the current word w_t and the previous hidden state h_{t-1},

$$h_t = f(w_t, h_{t-1})$$

The output of the encoder is a context vector c which is an embedding of the input sequence. The context vector is, in some way, derived from the sequence of hidden states. For some function $g()$,

$$c = g(h_1, h_2, ..., h_N)$$

The context vector is the final hidden state such that $g(h_1, h_2, ..., h_N) = h_N$. This is what is passed to the decoder. While the encoder processes the input and generates the encoding, the decoder language model is trained through teacher forcing i.e at each time step the actual output is compared against the target output and a loss is calculated. This is done for the whole target sentence after which the total loss is then calculated and the encoder-decoder is trained as a single system end-to-end.

3.2 The information bottleneck

Along the way of transforming one sequence to another, various issues such as network instability, vanishing exploding gradients and challenges in dealing with long sequences were encountered and these were dealt using different regularisation techniques and architectures such as LSTMs. Despite that, a critical weakness which remains in the RNN-based seq2seq model is that, in the transition from encoding to decoding what the encoder has to do is compress the entire source sequence into a single fixed length vector and this vector has to somehow carry information about complex phrase structures and long distance relationships between words, but there's only so much the model can do when it has crammed up all this information into a lower dimension. This leads to an information bottleneck. While using architectures such as LSTMs can help when the sequences are fairly short, the model performance drops dramatically as the sequences get longer because the vector carries more information about the later parts of the sequence than the earlier parts. Intuitively when we as humans transform a sequence into another this isn't how we do it. A human translator often refers back to the source sequence while translating. If we were to summarise a document we would certainly keep looking back at the source document to extract and paraphrase the key passages. What we do is *focus* on certain parts of the input depending on where we are.

3.3 Overcoming the bottleneck with Attention

In order to address this context bottleneck challenge, Bahdanau et al [1] introduced a neural network architecture which rather than throwing away the encoder's hidden states and forcing all available

information into a single vector; builds a unique context vector for every decoder time-step based on different weighted aggregations across all of the encoder hidden states. In other words, let the decoder look back at the source and learn at each time step which part of the input to focus on by providing a new context vector at the time that each word of the output sequence is decoded. The attention mechanism proposed by Bahdanau takes two important divergences from the seq2seq approach. First, there is a new context vector computed for every step of the translation sequence. Second, that context vector is not simply one of the hidden states from the input sequences, but is instead a weighted combination of the hidden states from the input sequence. Moreover, this weighting takes into account the relevance of the word contributing most towards the meaning at every time step producing a relevant context vector, which is why this concept is called **'attention'**.

3.4 Bahdanau architecture (Additive Attention)

The architecture makes use of a bidirectional RNN, which in this case is **BiLSTM** encoder that reads the input sentence in the forward direction to produce a forward hidden state, and then reads the input sentence in the reverse direction to produce a backward hidden state. The BiLSTM helps in curbing the vanishing gradient issue faced by vanilla RNN when encountered with long sentences. The encoder produces the final annotation for every word by concatenating the two hidden states. The idea behind generating each annotation in this manner is to capture a summary of both the preceding and succeeding words. The generated annotations are then passed to the decoder to generate the context vector. The context vector c_i is computed as a weighted sum of these annotations h_i:

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

The weight α_{ij} of each annotation h_j is computed by,

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} e_{ik}}$$

The decoder takes each annotation and feeds it to an alignment model, $e_{ij} = a(s_{i-1}, h_j)$, together with the previous hidden decoder state, s_{t-1}. The alignment model scores how well the inputs around position j and the output at position i match. The score is based on the RNN hidden state s_{i-1} and the j^{th} annotation, h_j of the input sentence.

The function implemented by the alignment model here combines s_{t-1} and h_i using an addition operation, which is why the attention mechanism is referred to as additive attention.

To summarize, the main components of the architecture comprises of:

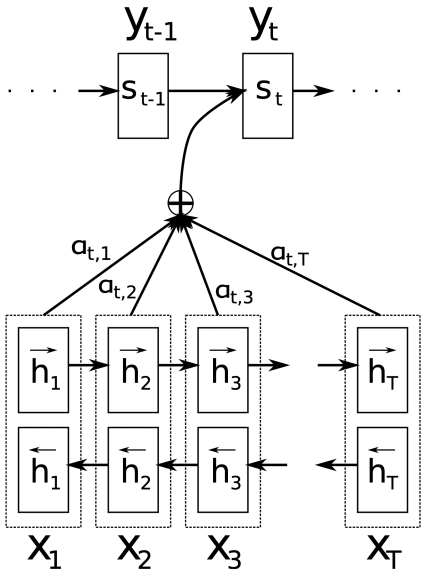
- s_{t-1} is the hidden decoder state at the previous time step $t-1$.
- c_t is the context vector at time step t, which is dynamically generated at every decoder step.
- h_i is an annotation capturing information of the entire input sentence with strong focus around the i^{th} word out of T total words.
- $\alpha_{t,i}$ is a weight value assigned to each annotation h_i at the current time step t.
- $e_{t,i}$ is an attention score generated by an alignment model $a()$

4 Training regime

4.1 Training

As discussed above, the attention model effectively compares a given decoder hidden state with a specific encoder hidden state to determine the relevance between these two vectors. This is implemented by applying the weight matrices W_1 and W_2 to s_{t-1} and h_i respectively,

$$a(s_{t-1}, h_i) = v^T \tanh(W_1 h_i + W_2 s_{t-1})$$



Credits: Neural Machine Translation by Jointly Learning to Align and Translate

where v is weight vector.
The alignment model is parameterized as a feedforward neural network and jointly trained with the remaining system components using backpropogation i.e, the gradients of the cost function will be used to update the weight vectors repeatedly until convergence. Hence, in addition to the RNN weights, even the Attention weights associated with each of the hidden states will also be *learned*. The weights will be updated in such a way that the words with maximum importance in translation will have maximum weight values, meaning more representation in the context vector. Subsequently, a softmax function is applied to each attention score to obtain the corresponding weight value:

$$\alpha_{t,i} = \text{softmax}(e_{t,i})$$

The application of the softmax function essentially normalizes the annotation values to a range between 0 and 1; hence, the resulting weights can be considered probability values. Each probability (or weight) value reflects how important h_i and s_{t-1} are in generating the next state, s_t and the next output, y_t.

During the training process, if the model predicts the incorrect translated word, it will be retained at that time-step, but the teacher forcing strategy will force the decoder to input the correct word as the input for the next time-step. This in turn helps speed up the training process. Since the decoder has to output prediction sentences of variable lengths, it will continue predicting words in this fashion until it predicts the next word in the sentence to be a *eos* tag. Once this tag has been predicted, the decoding process is complete and we are left with a complete predicted translation of the input sentence.

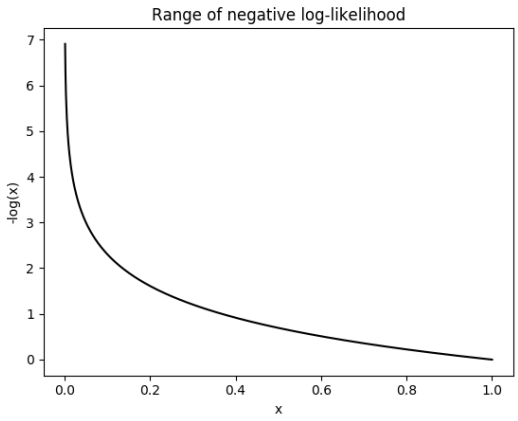
The accuracy of this predicted translation to the actual translation of the input sentence is compared to compute a loss which in this case is the Cross-Entropy Loss given by,

$$-\sum_{w=1}^{|S|}\sum_{e=1}^{|V|} y_{w,e} \log \hat{y_{w,e}}$$

where $|S|$ = Length of sentence
$|V|$ = Length of vocabulary
$\hat{y_{w,e}}$ = predicted probability of vocab entry e on word w
$y_{w,e} = 1$ when vocab entry is correct word
$y_{w,e} = 0$ when vocab entry is incorrect word

In essence, the loss function sums over the negative log likelihoods that the model gives to the correct word at each position in the output sentence. Given that the negative log function has a

value of 0 when the input is 1 and increases exponentially as the input approaches 0, the closer the probability that the model gives to the correct word at each point in the sentence is to 100%, the lower the loss.



Credits: Lj Miranda [4]

By summing over the loss for each word in the output sentence a total loss for the sentence is obtained. This loss corresponds to the accuracy of the translation, with lower loss values corresponding to better translations. When training, the loss values of several sentences in a batch would be summed together, resulting in a total batch loss.

In the original paper, the authors use mini-batch gradient descent to update the weight of the matrices but with time more efficient algorithms have been developed and one such optimizer that will be used here is **Adam** [3]. By running the model for more epochs on our simple dataset, Adam will lead to very good results as it converges a lot faster. Some advantages of Adam include requiring relatively low memory (though higher than gradient descent and gradient descent with momentum) and works well even with little tuning of hyperparameters (except α).

These updates modify the weight matrices to slightly enhance the accuracy of the model's translations. Thus, by performing this process iteratively, we eventually construct weight matrices that are capable of creating quality translation.

4.2 Inference Decoding

Now the question we are interested in is to find the better decoding technique. As mentioned above, the decoder produces a list of output probabilities.

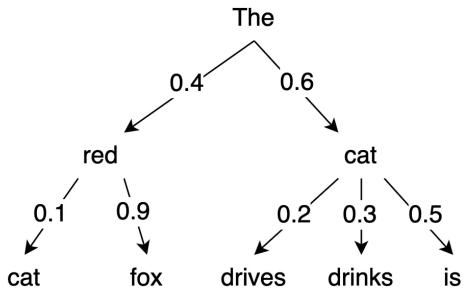
$$y' = \arg\max_{y} p(y|x)$$

In order to find the best arg max, I will explore two solutions and explain them in detail below.

4.2.1 Greedy Search

Greedy search consists of taking the token with the highest conditional probability from the vocabulary V. Suppose the model is predicting a possible continuation to a sentence starting with *The*, choosing from the words *red* and *cat* with their associated probabilities. With greedy search, after *The*, *cat* and subsequently *is* is selected in the next step. This continues until the decoder returns the special *eos* tag as the most probable output.

The major drawback of greedy search is that it is not optimal in producing high-probability sentences, as it misses high probability words hidden behind low probability words. Indeed, in our example, the sentence with the highest probability is *The red fox* $(0.4 * 0.9 = 0.36)$, as opposed to *The cat is* $(0.6 * 0.5 = 0.30)$. So, in general, the greedy search is not the best way except for simple translations. At times, the greedy search can lead to the wrong path as seen in the example.



4.2.2 Beam Search

Beam search addresses this problem by keeping the most likely hypotheses (aka beams) at each time step and eventually choosing the hypothesis that has the overall highest probability.

Suppose we are performing a beam search with two beams. At the first timestep, beam search would consider both the words *red* and *cat*, along with their probabilities. At the second timestep, beam search will consider all the possible continuations of the two beams and keep only the two of them with the highest probabilities. All the possible continuations with their probabilities are:

1. *The red cat*: $0.4 * 0.1 = 0.01$
2. *The red fox*: $0.4 * 0.9 = 0.36$ (highest)
3. *The cat drives*: $0.6 * 0.2 = 0.12$
4. *The cat drinks*: $0.6 * 0.3 = 0.18$
5. *The cat is*: $0.6 * 0.5 = 0.30$ (second highest)

Thus, the two beams at the next timestep will be *The red fox* and *The cat is*. On ending the beam search at this point, the predicted result would be the beam with the highest probability, i.e *The red fox*. Usually, the chosen beam size in practice is in the range of 4-10. Increasing beam size is computationally inefficient and, also, leads to worse quality.

4.3 Hyperparameters

The process of training a model involves choosing the optimal hyperparameters that the learning algorithm will use to learn the optimal parameters that correctly map the input features (source language) to the labels or targets (target language) such that you achieve some form of intelligence. The following are the hyperparamters that can be finetuned in this case:

- Number of layers in Encoder and Decoder (2/4)
- Hidden size of Encoder and Decoder $(200 - 500)$
- Dropout value for Encoder and Decoder (0.2/0.4)
- Training batch size (16/32/64/128)
- Number of epochs $(20 - 100)$
- Learning rate (0.005/0.01/1)
- Optimizer (Adam/SGD)
- Beam size $(4 - 10)$ with length penalty 1.0

5 Evaluation

Once we have a neural translation system, the gold standard for evaluation is to get humans to look at the results but of course that can be expensive. As that is infeasible the most popular automatic method is called BLEU [5] which stands for *BiLingual Evaluation Understudy*. It is based on the idea that the closer the predicted sentence is to the human-generated target sentence, the better it is.

The idea is that for a subset of source sentences there will be reference translations provided by humans and the candidate translation supplied by our translation system and the intuition is to count the number of overlapping ngrams between the candidate and reference translations. The BLEU formula is composed of two components, the brevity penalty and the ngram overlap or precision. The n-grams overlap make sure that randomly ordering the correct words will not be rewarded because they only match the reference when the words are in the same consecutive order while brevity penalty penalizes short translations that don't contain relevant text from the reference translations. The score ranges between 0 and 1 and the aim is to score atleast above 0.5 to achieve high quality translations.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[4] Lj Miranda. Understanding softmax and the negative log-likelihood. https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/, 2017.

[5] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

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

[7] Tatoeba. English to hungarian translation sentences. https://www.manythings.org/bilingual/hun/, 2023.

[8] Lena Voita. Seq2seq and attention. https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html, 2023.