Advanced Natural Language Processing: Seq2Seq Machine Translation and Semantic Tree Parsing

Assaraf David

April 9, 2021

Abstract

In this report, we will analyze in a first part the performances of various models in order to perform Seq2Seq Machine Translation. Through theoretical questions and experiments, we will try to shed light on the most effective mechanisms in order to translate an English Sentence to Vietnamese. In the second part of this report, we will study neural parsing mechanisms in order to extract semantically meaningful information.

1 Machine Translation

We have seen that, until 2018, the gold standard model for Seq2Seq tasks was a RNN Encoder Decoder Structure with attention [Pet+18]. Here, we will first implement this vanilla Encoder-Decoder structure without attention, with a large setting of hyperparameter tuning. Then, we will implement Attention and Masked Attention in our structure and see how well it performs in our Seq2Seq task. During this phase, we will monitor our model performances using perplexity (which can be expressed as $2^{Avg(LL)}$). Our testing metrics will be the BLEU metrics. We choose to use perplexity in order to compare different models because the BLEU metrics depends on our decoding strategy. Therefore, having selected the best architecture, we will use different decoding strategies and see how well different decoding strategies allow to improve the BLEU score. Before delving into Seq2Seq modelling, let us recap the different models we have seen in the course: HMM, CRF, Neural Networks (GloVE, Word2Vec; FastText) RNN-like models and Transformers. The modelling hypothesis in HMM and CRF is a Markov Chain depending on hidden states, and where transitions only depend on hidden state transition. We know that RNNs use the same kind of hypothesis about hidden states update, they just allow for more flexibility since the update in hidden state depends on the previous hidden state but also on the current input. Therefore, the first question is: *can the RNN capture what the HMM model can?* Ie can we, for a fixed HMM, construct a RNN that has the same properties ?
What do we want to capture with the RNN ? The 3 questions from Rabiner.

- Likelihood: $P(O|\lambda)$
- Decoding: $argmax_Q P(O|Q, \lambda)$
- Learning

We know that the likelihood is only defined thanks to the hidden state transitions probabilities $p(q_t|q_{t-1})$ and the probability emission $(b_j(o_t))$, and the decoding also. Therefore, if we can replicate the emission and transition probabilities, our RNN can capture the same as a HMM.

1.1 Deterministic HMM

When the HMM is deterministic, it means that A and π are non random, ie only one transition is permitted per hidden state, and the initial hidden state is deterministic. Therefore, in the RNN update $h_t = f(W_x x_{t-1} + W_h h_{t-1} + b_h)$, we want $f = Id$, $b_h = 0$ $W_x = 0$ (as a matrix) because we do not want our hidden state update to depend upon the current input. Last, we want $W_h = A$, since it will allow for transitions between hidden states as dictated per transitions in the HMM model. We will initialize the hidden state $h_0 = \pi$ in order to keep the same dynamics as in the HMM. Per recurrence, we can show that every vector h_t is one-hot encoded. Last, we want to output words at time t that are related to the probability of emission in state t. Hence, using $W_2 = B$ and $g = Id$ will allow to sample from the distribution of words b_j that could be output in the hidden state $h_t = q_t$ (as per shown before).

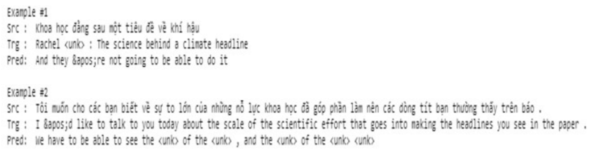
1.2 Non Deterministic HMM

Using the exact same attribution of the RNN parameters, we will be able to reconstruct the same behaviour as in the no deterministic HMM. The behaviour will still be random, but it will be the same randomness as in the HMM. Moreover, the 3 questions from Rabiner will have the same conclusion in terms of likelihood and decoding strategy.

1.3 Seq2Seq Model

In this section, we'll delve into the problem of Machine Translation using an Encoder Decoder Structure with RNN. In the following, we will present MT results using a Vanilla Architecture: unidirectional autoregressive encoder, feeding its final hidden state to the initial state of the decoder. Both encoders and decoders leverage one RNN layer (not yet GRU). Then, we will analyse the performances and output of the model and try to improve its performances. I would like to insist on the fact that the improvements of the different models are measured in terms of **perplexity**. This is the only objective goodness of fit measure we disposed, as the BLEU metrics further depended on the decoding strategy. One thing worth to mention is our current decoding strategy: every step, we sample a deterministic word, being the most probable word to be sampled given the current word.

1.3.1 MT performances



```
Example #1
Src :  Khoa học đằng sau một tiêu đề về khí hậu
Trg :  Rachel <unk> : The science behind a climate headline
Pred:  And they &apos;re not going to be able to do it

Example #2
Src :  Tôi muốn cho các bạn biết về sự to lớn của những nỗ lực khoa học đã góp phần làm nên các dòng tít bạn thường thấy trên báo .
Trg :  I &apos;d like to talk to you today about the scale of the scientific effort that goes into making the headlines you see in the paper .
Pred:  We have to be able to see the <unk> of the <unk> , and the <unk> of the <unk> <unk>
```

Figure 1: Vietnamese - English translation with a Vanilla 1 layer Encoder - Decoder Unidirectional RNN cell

Let's explain both examples reported in 1. In the first example, we can see that the model could not capture the meaning of the sentence. The objectives of climate and science are not successfully captured. Moreover, it includes a non sense word in the translation. It is not very clear how to alleviate these issues while looking at this output. Indeed, the length of the sentence seems reasonable so it might not be a memory issue. The non-sensical issue could be related either to an unsufficient ability to retain informatio (ie model limitations) or unsufficiency in the data.

In the second example, we can see that the same kind of errors occur. However, an additional issue could be reported on top of these issues: the length of the predicted sentence, which is way shorter than the target prediction. This could be related to the fact that RNN has memory issues and therefore cannot handle long sentences predictions. Moreover, we can see the abundance of '¡unk¿' tokens in the decoded sentence. In order to alleviate memory issues, we could leverage other types of RNN cells rather than the vanilla RNN and also implement an Attention mechanism in order for information to flow more easily between the encoder states and the decoder states. Using another decoding strategy might alleviate the issue of generating deterministically the unknown tokens.

1.3.2 Improving the performances of our Encoder Decoder without Attention

In the following, I am going to report the different experiments I have made in order to improve the **perplexity** of the Encoder Decoder structure. The way I performed experiments was the following: changing the RNN cell, changing the structure of the Encoder (from Unidirectional to Bidirectional), changing the number of layers, changing the hyper-parameters of the learning problem: hidden size, embedding dimension, dropout and learning rate. After every experiment, I selected the best previousmodel in order to optimize over the next hyperparameter: therefore, this is a greedy hyperparameter optimization rather than a grid-search over hyperparameters. In the following table, the model selected during one hyperparameter optimization is bolded.

Hence, we can see that we have successfully improved the performances of our model in terms of validation perplexity. Our final model without Attention is a BiDirectional 2 layers GRU Encoder,

Model	Validation Perplexity
Vanilla RNN	87.23 +/- 5.23
Vanilla GRU	**41.72 +/- 1.26**
Bi GRU Sum	37.61 +/- 0.87
Bi GRU Concat	**37.23 +/- 0.99**
Bi-GRU 2 layers	**34.98 +/- 0.64**
Bi-GRU 3 layers	37.74 +/- 1.25
Bi-GRU 4 layers	37.38 +/- 1.58
Dropout 0.1	**31.85 +/- 0.22**
Dropout 0.15	33.14 +/- 0.37
Dropout 0.2	32.48 +/- 0.57
Dropout 0.25	33.03 +/- 1.23
Dropout 0.3	33.44 +/- 0.79
Embedding 32	38.53 +/- 1.22
Embedding 64	37.34 +/- 2.37
Embedding 128	35.27 +/- 1.87
Embedding 256	**31.96 +/- 0.31**
Embedding 512	32.90 +/- 0.38
Hidden size 32	77.98 +/- 6.82
Hidden size 64	52.90 +/- 4.65
Hidden size 128	40.37 +/- 2.37
Hidden size 256	**31.63 +/- 0.47**
Hidden size 512	38.86 +/- 1.23
Learning Rate 7e-3	241.63 +/- 14.64
Learning Rate 5e-3	88.01 +/- 9.34
Learning Rate 2e-3	42.89 +/- 4.37
Learning Rate 1e-3	31.71 +/- 0.48
Learning Rate 8e-4	**30.98 +/- 1.34**
Learning Rate 5e-4	33.87 +/- 2.28

Table 1: Performances of the Different Encoder Decoder Without Attention. The results reported are the mean perplexity after 10 epochs over 5 different runs and the standard deviation measure. The bidirectional attribute refers to the Encoder, since the Decoder works in an autoregressive way. The bidirectional GRU Concat and Sum are the two different ways we tried to combine the Forward and the Backward layers' hidden state for the last hidden state of the encoder.

concatenating the forward and backward hidden states. The learning hyperparameters regarding embedding size, hidden size are both 256. The dropout between the layers of GRU Encoder is 0.1. Last, the learning rate is set to 8e-4.

1.3.3 Encoder Decoder with Attention

Even thought we implemented hyperparameter tuning, there are some limitations to our model inherent to its very structure. We won't be able to fit the entire meaning of a sentence inside one single hidden state. Therefore, we must allow every step in the decoder to attend to every hidden state in the encoding step: we must implement the Attention mechanism. We used the attention mechanism introduced in [LPM15].

Last, in order to further improve the performances of the attention mechanism, we introduced Mask Attention, in order to prevent the decoder from attending irrelevant steps in the Encoder (ie the unknwown tags). Unfortunately, this did not produce the expected improvements. I think this is due to the fact that we should let this mode complex model train for longer than 10 epochs 2.

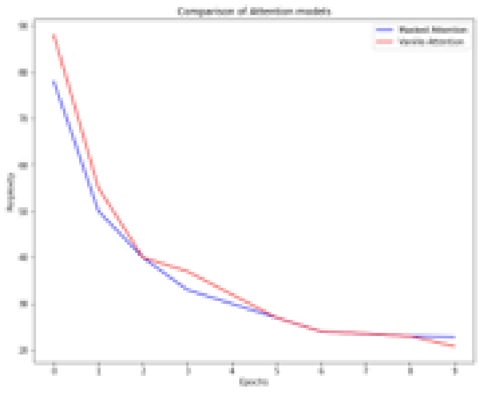


Figure 2: Comparison of Masked and Unmasked Attention training

1.4 Decoding Strategy

As mentioned, the current decoding strategy used is a **greedy decode**, meaning that we always go deterministically for the most probable word decoded. However, this greedy strategy substantially lowers the variability in language. Some better decoding models rely on sampling from the output distribution. In order to prevent from sampling from a very big distribution (of size $|V|$), we could use two heuristics:

- Top p decode: sample from the renormalized probability distribution composed of probability of the words with top probability such that the accumulated sum is greater than a threshold (usually 0.8).
- Top k decode: sample from the renormalized probability distribution composed of probability of the k words with top probability

Last, one more elaborated decoding strategy would be beam search. This strategy allows to mitigate issues with making greedy local decisions. Beam search with width w keeps track at every step of the w best sentences decoded until then, and updates them at every step. This should be the best decoding strategy of the mentionned strategies.

1.5 BLEU Score

BLEU refers to Bilingual Evaluation Understudy. The scores computes correspondences between a machine translated sentence and a human translated sentence. It is being applied to the different n-grams of the predictions and the targets, and computes an average over unigrams, bigrams , trigrams and 4grams.

We will here see how our different improvements allowed to improve the BLEU score.

Model	Decoding Strategy	BLEU
Bi-GRU 2 layers	Greedy	5.77
Attention	Greedy	13.32
Masked Attention	Greedy	12.86
Attention	Top p (0.8)	16.78
Attention	Top k (20)	17.78

Table 2: Bleu Scores of the different models studied until now, and also for different decoding strategies

2 Trees

2.1 Writing a CFG

In this part, we are going to use a Neural Network in order to predict a semantic tag for every part of a given sequence. This task is different from POS tagging where we want to assign one tag to every word. Here, we want to assign some **semantic** tags to some parts of the sentence. This could allow for easier downstream tasks such as machine understanding intents. In order to decode a sentence into its tree of semantics tags, we will use the CKY algorithm. The CKY algorithm is a DP algorithm that finds the optimal parsing of a sentence, based on probabilistic CFG rules (for non terminal nodes as outlined above) and emission probabilities for terminal nodes. Those are the characteristics of a **Probabilistic CFG**. But first, in order to be able to define a pCGF, we need some scores to be assigned to sub sentences structure. This can be done in a static way, or in a contextual way. The latter is the approach we are going to leverage.

(a)

Symbols	Meaning
N	Nouns
NP	Noun Phrases
TV	Transitive Verbs
IV	Intransitive verbs
D	Determiner the
C	Conjunction (that)
E	Empty

Table 3: Symbols of the CFG we will use

(i)

- S $\mapsto$ NP VP
- S $\mapsto$ NP IV
- NP $\mapsto$ NP C NP TV
- NP $\mapsto$ D N

(ii)

- S $\mapsto$ NP E VP
- S $\mapsto$ NP E IV
- E $\mapsto$ E—C NP C NP TV TV — C NP TV
- NP $\mapsto$ D N

2.2 Part A : Neural Classification of sub-sentences structures

(a) The first step of our work was to define the tree parsing structure for a semantic tag assignment to a structure. We did so using the usual Depth-First search traversal of our graph. Our data was therefore composed of the following datum: a sentence going along with a list of labels. Inside this list of labels, we had the different assignments of sub-sentences structures to their tag. The way we leveraged this data is to embed the sentence in a word per word approach and then feed the embeddings to a LSTM. Then, when we want to predict the tag of a sub-sentence $(w_i, ..., w_j)$, we would just feed the concatenation of the hidden states at step i and step j: $h^{i:j} = [h_i; h_j]$ and just perform simple Dense layer on top of this vector. The rationale behind this construction is that the LSTM hidden state will accumulate the information between w_i and w_j, and the model will be able to understand specifically what happened between w_i and w_j. We trained this in an end-to-end approach, using SGD

with momentum = 0.9 and Cross Entropy Loss for 3 epochs. The model starts overfitting after 3 epochs so we did not want to train it for longer. The size of our data was 30000 points (meaning 30000 spans and their assigned labels) using the data from [Gup+18]. One important thing to mention here is the decoding strategy. Here, we use the individual hidden states in order to produce results for entire sub-sentences. Then, from the output distribution, we will select the most probable token for the span based on its end words. We will call it the **lazy decoding**. The results that we got are:

f_1	0.89
recall	0.95
precision	0.85
exact match	0.42
well form	0.54
tree match	0.48

Table 4: Results from the end to end training of the Sub-sequence Neural Semantic Classification, for 8997 examples.The decoding strategy used here is lazy decoding

(b) Some specificities of the architectures are that we are using 2 layers Bi Directional LSTM in order to produce the hidden states that we will later concatenate. First, we do so because we can (we could not do it in the Seq2Seq decoder). Then, it will give use better performances because the hidden states will be built using information from before and after our specific word in the sentence. In this specific settings, since we are looking to **disambiguate** different possible parsings, adding some information to what will happen later and before will allow the classifier to directly disambiguate over two similar parsings, where the difference between both only comes later in the sentence.
(c) Last, the only rationale part that could be further improved in our architecture is the way we construct the representation of a span. Let us imagine that the input is a **very long sentence**. Only taking into account information from the hidden states of the first and the last word of the sentence will definitely lead to losing some information along the way. Moreover, it could be that some step $k \in [i; j]$ would allow an easier decision. One potential method in order to improve the current algorithm for generating span embeddings might be attention. We would let the newly created span embedding attend to every word in the span and let the span decide which word is important to make a decision regarding a span. It would also allow to divide by 2 the number of parameters in the Dense layer since we would not be concatenating the embeddings from words w_i and w_j.

2.3 CKY algorithm

Last, once that we have the scores for the different possible assignments, we would love to implement a decoding strategy. This decoding strategy would answer similarly one of the 3 Rabiner questions we introduced in the first part when mentioning HMMs: $argmax_T p(S, T)$. As in HMMs where we got the forward algorithm in order to decode a sentence, here we are going to use the CKY algorithm. The essence of this DP algorithm is the recurrence

$$S_{ij} = max_k s_{ik} + s_{kj} + R(ik, kj)$$

where the function R is a measure of how probable it is for tags ik and kj to happen together and generate the tag ij. Here, we will implement a simpler recurrence.

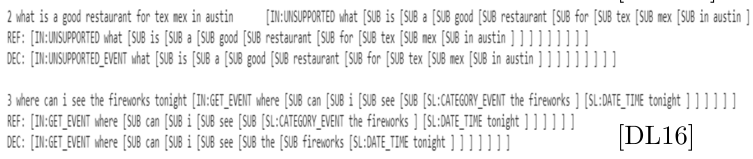
We can therefore see that there is a fundamental difference here on the **lazy decoding**. Here, we use individual scores from the previous Neural Network and see how well they combine in order to aggregate them into sub sentences tags. This bottom-up algorithm should allow for more meaningful structures.
(a) The results we had after implementing the CKY algorithm are

f_1	0.87
recall	0.86
precision	0.87
exact match	0.44
well form	1.0
tree match	0.51

Table 5: Results from the end to end training of the Sub-sequence Neural Semantic Classification, for 8997 examples.The decoding strategy used here is CKY decoding

We can see that the significant drop in performances are in the recall metrics. We can see that according to the drop in recall, there are quite less positive labels. I think this is due to the fact that we used a 'truncated' version of the CKY algorithm, which in this special case does not take into account how well the two non terminal nodes combine in terms of probability. This should lead to unlikely combinations ultimately leading to wrongly predicted sentences.

(b) Now, let us explore qualitatively the output of the CKY algorithm and explore the errors.



```
2 what is a good restaurant for tex mex in austin      [IN:UNSUPPORTED what [SUB is [SUB a [SUB good [SUB restaurant [SUB for [SUB tex [SUB mex [SUB in austin ] ]
REF: [IN:UNSUPPORTED what [SUB is [SUB a [SUB good [SUB restaurant [SUB for [SUB tex [SUB mex [SUB in austin ] ] ] ] ] ] ] ] ]
DEC: [IN:UNSUPPORTED_EVENT what [SUB is [SUB a [SUB good [SUB restaurant [SUB for [SUB tex [SUB mex [SUB in austin ] ] ] ] ] ] ] ] ]

3 where can i see the fireworks tonight [IN:GET_EVENT where [SUB can [SUB i [SUB see [SUB [SL:CATEGORY_EVENT the fireworks ] [SL:DATE_TIME tonight ] ] ] ] ] ]
REF: [IN:GET_EVENT where [SUB can [SUB i [SUB see [SUB [SL:CATEGORY_EVENT the fireworks ] [SL:DATE_TIME tonight ] ] ] ] ] ]
DEC: [IN:GET_EVENT where [SUB can [SUB i [SUB see [SUB the [SUB fireworks [SL:DATE_TIME tonight ] ] ] ] ] ] ]
```

Figure 3: Examples of two wrongly parsed sentences for the CKY algorithm.

The labelling mistakes we got are:

- First sentence: the first tag has been wrongly predicted. Still the remainder of the sentence is correct, which is quite good
- Second sentence: A more concerning decoded sentence is this sentence. We can see that the 'fireworks' is not detected as an event, and is identified as SUB tag.

(c) Per a Piazza post, I understood why we would need the None in training. Since we do not want any combination of span inside our decoding tree, we must somehow tell our decoder model that some spans are infeasible. This is being done when adding the None labels during the training step.

(d) One way we could solve the parsing problem as a Seq2Seq task is to consider the parsing tree as a decoded sentence. Here, we will try to explain the approach as is being depicted in [DL16]. The first approached used is a Vanilla Encoder Decoder without any attention !! Such a simple architecture already provided sufficient results. Then, one could also use Attention Mechanisms on top of that. Basically, we predict a tag for every decoding step. One issue to this architecture is that it does not take into account the hierarchical aspect of the semantics parsing problem. Therefore, the LSTM needs to keep track of the positions of the different brackets and other stuffs in order to be able to predict a rightful semantic parsing structure. This is why the paper introduces a more elaborated Sequence-to-Tree model, which leverages a decoder working in a top-down approach, which basically uses several layers of deciding cells in order to encode for the different layers in the semantic parsing.

References

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. "Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025* (2015).

[DL16] Li Dong and Mirella Lapata. "Language to logical form with neural attention". In: *arXiv preprint arXiv:1601.01280* (2016).

[Gup+18] Sonal Gupta et al. "Semantic parsing for task oriented dialog using hierarchical representations". In: *arXiv preprint arXiv:1810.07942* (2018).

[Pet+18] Matthew E Peters et al. "Deep contextualized word representations". In: *arXiv preprint arXiv:1802.05365* (2018).