NMT in Low Resource Scenario: A Case Study in Indian Languages

Karanveer Singh
Pushpak Bhattacharyya
CFILT, Indian Institute of Technology Bombay, India

{kvsaroya, pb}@cse.iitb.ac.in

Abstract

Neural Machine Translation is the current buzzword in the field of Machine Translation. One of major requirements of Neural Machine Translation is that of data. This requirement makes it somewhat unsuitable for language pairs from whom less number of corpus is present. In this paper, we look at ways to mitigate the effects of low resource data by using techniques like word segmentation and pivot languages. We first look at some fundamentals required for understanding Neural Machine Translation and then study word segmentation. Then we look at Pivot based NMT and finally end with a look at SMT based models involving both word segmentation and pivot languages.

1 Introduction

Neural Machine Translation (NMT) is the current favourite in Machine Translation circles. NMT is the process by which we translate a source sentence to target sentence by the help of Neural architectures namely Recurrent Neural Networks (RNNs). We look at all the work which is related to our current work. We first look at some fundamentals required for Neural Machine Translation then delve deep into work done in areas like Subword NMT, Pivot NMT and finally look at the work done in Statistical Machine Translation (SMT).

2 Neural Machine Translation

Neural Machine Translation (NMT) uses Neural Translation Models (NTM) for actual translation. Before looking at the models let us first look at Neural Language Models which serve as the basis for the translation models. We look at Recurrent Neural Networks variants like Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU).

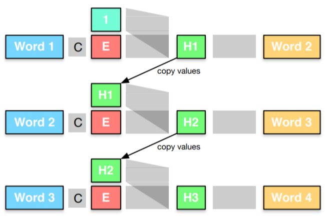


Figure 1: Recurrent Neural Network Language Model. From source (Koehn, 2017)

2.1 Recurrent Neural Networks

Feed forward neural language models are able to handle longer context than traditional statistical back-off models but suffer from a very distinct flaw. They use a fixed context i.e. the number of words in the context are fixed, this poses a restriction on the amount of context that we can capture for a given word. Recurrent Neural Networks (RNN) are a way around this problem and allow context of variable lengths.

As can be seen in Figure 1 one of the major features of an RNN is parameter sharing i.e. values of the hidden layer are copied from the previous layer to the current layer. This very parameter sharing is what allows RNN's to capture long contexts as every new layer is aware of the context from the previous layers. In this *word 3* is aware of all the words preceding it and not only *word 2* as would have been the case if a regular feed forward network were used for prediction of the next word. This essentially allows for arbitrary length contexts and allows for much better prediction since the context window is not fixed. Training is done through the **BPTT** or the **backpropagation through time** procedure which unrolls the RNN to a certain number of steps and hence the network is able to learn dependencies over a long distance.

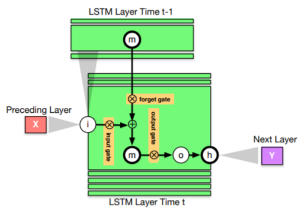


Figure 2: Long Short Term Memory Cell. From source (Koehn, 2017)

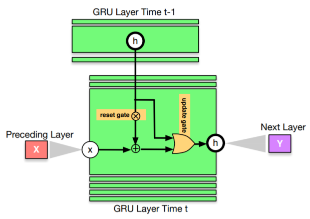


Figure 3: Gated Recurrent Unit (GRU). From source (Koehn, 2017)

2.2 Long Short Term Memory

We now look at certain variant of the RNN specifically Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU). Let us tackle LSTM first. Though RNN's can theoretically can learn dependencies over any arbitrary length sentences, in practice this is not the case. Specifically they suffer from problems like **exploding gradients** and **vanishing gradients**. While we will not go deep into these problems they can be summarized as the following:

- Exploding Gradient refers to a problem where error gradients start to accumulate and lead to an increasingly large network weight update which in turn leads to the value of the weights to overflow and become NaN.

- Vanishing Gradient problem is similar except that the updates are very small hence the overall change in the weights very small hence the network does not learn over a long sentence.

To tackle these problems we have architectures like LSTM and GRU which prevent the above mentioned problem in their ways. Let us now look at the LSTM structure. As can be seen in Figure 2 LSTM contains **gates** which are nothing but real numbered parameters whose values are learnt during the training process.

- The **input gate** determines the influence of the new input on the memory.

- The **forget gate** determines the extent to which the previous memory state is retained.

- The **output gate** determines how much of the current memory state is passed onto the next layer.

The equations related to these can be summarized as the following with the values of memory, input, output at time step *t*:

$$memory^t = gate_{input} \times input^t + gate_{forget} \times memory^{t-1}$$

$$output^t = gate_{output} \times memory^t$$

$$h^t = f(output^t)$$

$$input^t = g(W^x x^t + W^h h^{t-1})$$

Here h^t is the value of the hidden node which has been obtained by applying an activation function to the output value at time *t* and *g* is an activation function applied on the input *x* at time *t* and W^x, W^h are weight matrices.

2.3 Gated Recurrent Unit

Let us now look at Gated Recurrent Unit. One of the problems with LSTM are that they use a large amount of parameters which must be learnt leading to longer training times. GRU's have been used as a simpler solution to tackle the exact problems that LSTM's tackle. Looking at Figure 4 we can see key differences between the LSTM's and GRU's, the major being that there are only two gates and no memory state. The equations related to GRU's can be summarized as follows:

$$update^t = g(W_{update} input^t + U_{update} state^{t-1} + bias_{update})$$

$$reset^t = g(W_{reset} input^t + U_{reset} state^{t-1} + bias_{reset})$$

2.4 Word Embeddings

We also look at the role of word embeddings in Neural Machine Translation. If we use the **one-hot vector** approach we run into the danger of wasting a large amount of space if the vocabulary is very large since the majority of the values in this matrix would be 0 and only one value in a particular row would be 1. This can countered by using word embeddings which are nothing but a compact way of representing words as vectors in a sufficiently high dimensional space. One of the properties of word embeddings is that they allow words which have similar contexts to be more closer in the vector space than words which are not related.

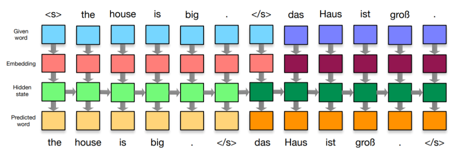


Figure 4: Sequence to Sequence Encoder Decoder Model. From source (Koehn, 2017)

3 Neural Translation Models

Now all that remains is for us to look at the translation models that we use in NMT and how alignment happens in NMT. We take a look at the **attention** mechanism also.

3.1 Encoder-Decoder Model

This is an extension of the RNN based language model that we looked at in the previous section. In this approach, we have two distinct phases **Encoder Phase** and **Decoder Phase**. During the training phase, we feed the concatenation off the sentence pair to the model. Just as in RNN, where it predicts the next word, here to the next word is predicted until we end up at the sentence end marker. Now the hidden layer at this point contains **input sentence embedding** i.e. the hidden layer represents the meaning of the input sentence. This hidden layer is now fed to the rest of the sentence which is in different language, the same process happens and model learns a translation. This constitutes our decoder phase. Though this model is simple to understand but in practice it works for only small sentences. These models can be broadly classified into 3 categories:

1. Cho Model
2. Sutskever Model
3. GNMT Model

Let us look at these models in detail and figure out what they are and how do they work in practice.

3.1.1 Sutskever Model

Deep Neural Networks(DNN) suffer from the fact that they accept only a fixed dimension vector as input and output a vector of fixed dimension. However in sequence to sequence learning tasks we do not have a fixed length of input sentences nor on the output sentences hence there arises a need to circumvent this problem. The Sutskever model tackles this very problem. The model uses 2 LSTM's one for mapping(encoding) the input sequence to a fixed length vector and other to decode this vector into the final output. The second LSTM is also conditioned on the input. This method is also used in the Cho model with RNN's replacing the LSTM's however this introduces the problem that the RNN has difficulty to learn long range dependencies hence LSTM work far better in this approach.

The LSTM learns the conditional distribution $p(y_1, \ldots, y_T | x_1, \ldots, x_{T'})$ i.e. which output sequence has the maximum probability given the input sequence. Notice that the input sentence and the output sentence can have different lengths hence the use of T' and T. Once the entire input sequence has been fed to the LSTM , the hidden state of the LSTM contains the sentence embedding or the representation of the sentence. This is then fed to the output LSTM with its initial hidden state equal to v. Then using the standard LSTM-LM we find the probability of the the sequence $y_1, \ldots, y_T$.

$$p(y_1, \ldots, y_T | x_1, \ldots, x_{T'}) = \prod_{t=1}^{T} p(y_t | v, y_1, \ldots, y_{t-1}) \quad (1)$$

The properties of this model are as follows:

1. Deep LSTM's are used in this model i.e. an LSTM with four layers is used.

2. The order of the input sentence is reversed as it decreases the distance between the starting of the input sentence and the target sentence hence the model is able to establish a better relation between the given sentences.

3.2 Alignment Model

We will now model our alignment model explicitly, we also call this alignment as **attention**. The**encoder** encodes the input sentence and provides us with the sentence embedding. We use a **bi-directional recurrent neural network** i.e. we run two neural networks, one from left to right on the sentence and another from right to left. Since we are considering both left-right and right-left states hence we are predicting a word in the context of the entire word. The decoder we use is also a **recurrent neural network**. This takes as input, a representation of the input sentence, sentence embedding and previous prediction to give us a new output prediction.

3.3 Attention

From our previous discussion we see that the encoder gives the sentence embedding at the end of it's process and the decoder must find the translation using this representation only. Hence it must have every bit of information required for the decoder to do it's job. Now if our sentence is of small size then this is no problem, however as sentence size increases, it becomes difficult to decode the sentence form a single vector. This leads us to the attention mechanism. The crux of this approach is that in some cases it might not be the best to look at the states immediately preceding the current state, rather some different set of states have to be looked at by the decoder. This is the principle of the attention, the decoder output for every word depends on a weighted combination of all input states rather than only the previous.

3.4 Bahdanau Attention

One of the main problems with the encoder-decoder approach is that the encoder has to be able to encode the sentence into a fixed length vector causing problems with long sentences. In order to tackle this problem, a model which learns to simultaneously align and translate is used. Bahdanau et al. (2014) proposes a different model than the encoder-decoder model to tackle this problem.

This architecture contains a bidirectional RNN as an encoder and a decoder which searches through the source sentence while decoding. As in the encoder-decoder model our goal is to maximize the conditional probability of the target sentence given the source sentence. This is captured in the new model through the following equation

$$p(y_i|y_1,\ldots,y_{i-1},\mathbf{x}) = g(y_{i-1},s_i,c_i) \quad (2)$$

where s_i is an RNN hidden state at time i and is computed by

$$s_i = f(s_{i-1},y_{i-1},c_i) \quad (3)$$

Notice that the *v* vector in the previous model Sutskever et al. (2014) has been replaced by c_i which is the context vector. This also exhibits one of the key differences between encoder-decoder model and the current model i.e. the current model does not try to condition the current input on the whole of the vector rather only on the current context vector. The context vector is computed by

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \quad (4)$$

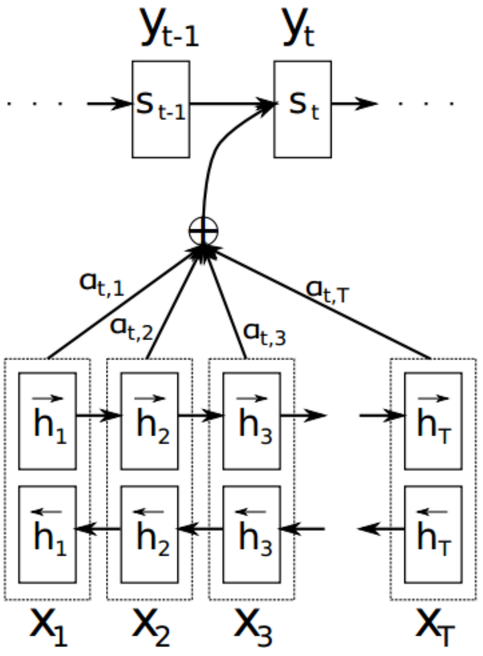


Figure 5: An example of the proposed model, adopted from Bahdanau et al. (2014)

α_{ij} is computed by the following formula

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})} \quad (5)$$

where

$$e_{ij} = a(s_{i-1},h_j) \quad (6)$$

is an alignment model which models how well inputs around position *j* and output at position *i* match. The weight α_{ij} reflects the importance or the attention paid to the h_j annotation with respect to the previous hidden state s_{i-1} while determining the state s_i and generating y_i.

The alignment model *a* is modelled as a feedforward neural network and is trained along with the other components of the system. Usually in traditional machine translation systems we see that the alignment is not explicitly modelled however in this model we explicitly model the alignment.

4 Minimum Description Length(MDL) Principle

Before we discuss the what the MDL principle is let us look at the problem of *model selection*. Given a limited number of observations of data we can have several models that explain the given observations, so the question arises how do we choose a model in such a scenario? This problem of model selection is the problem that we can solve by appealing to the MDL principle. The crux of the principle is that data has **regularities** and any such regularity can be leveraged to decrease the size of the data literally. If we equate *finding* regularity to *learning* we can think of finding more and more regularities as learning more and more hence more learning leads to more compression of the data.

One of the arguments made in the above paragraph is that more the regularity more we can compress the data. Here compression is thought of as a way of describing the data, hence we can say more the regularity more succintly or shortly we can describe the given data. Of course this leads us to the question of how do we describe the data i.e. what does the description of the data entail. We can look at a general purpose computer language like C as a way to describe the data. A program then which can print the required data is a valid description and we want the program which is of the shortest length since that exploits the most regularity out of the data.
0001000100010001 . . . 0001000100010001

The above let's say 10000 bit string can be described succinctly as a C program

$$for\ i\ =\ 1\ to\ 2500\quad print\ '0001'\ ; \qquad (7)$$

As one can see the length of this description is significantly smaller than the length of the acutal string. Hopefully this gives an idea about how expoiting regularity can lead to compression of data. Of course if there were no discernable pattern or regularity in the data then the length of the program would be roughly equal to the length of the actual string itself since we would have no way but to print the entire string as it is. To formalize this we define something called the *Kolmogorov Complexity*. This is defined as the length of the shortest program which can print the given sequence and then halts. The lower the Kolmogorov complexity of a sequence, the more regularity it has.

5 Subword NMT

One of the most common problems in NMT is translation of rare words. NMT models usually work with a fixed vocabulary which are usually the most frequent occurring words in the training corpus hence words which appear less or might only appear in the test data never get added to this vocabulary. In order to handle this problem of rare/ out of vocabulary (oov) words Sennrich et al. (2015) suggested using a modified Byte Pair Encoding (BPE) algorithm to break words into smaller units called subwords. The motivation behind this being that some words are *transparent* by nature. These words usually belong to the following categories :

- Named Entities.
- Cognates and Loan Words.
- Morphologically complex words.

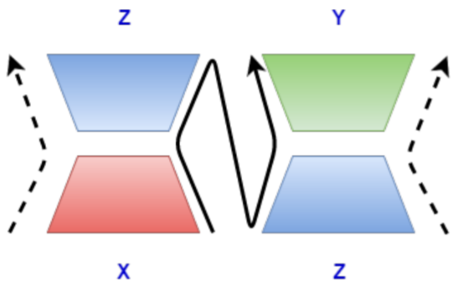


Figure 6: Pivot Based NMT from X to Y using Z. From source (Saha et al., 2016)

5.1 Byte Pair Encoding Algorithm

Byte Pair Encoding (Phillip, 1994) is modified where instead of the most pair of bytes being merged, the most frequent pair of characters are merged and replaced by a character not previously used. This operation of replacing the most frequent pair of characters and replacing it with another is called a *merge* operation. The number of *merge* operations to be performed on the corpus is a hyper-parameter which needs to be trained. The following example gives an idea of how this algorithm works:

Let the original text be $T_0 = ABABABABCCCD$ then the most frequent pair of characters is *AB* which is replaced by character *X*. The new text becomes $T_1 = XXXXCCCD$. The pair which is selected next is *XX* which is replaced by *T* and the text now becomes $T_2 = TTCCCD$. This process continues until no merge operation can be further done.

6 Pivot Based NMT

Let us take a look at the different models possible for pivot based NMT.

6.1 Two Stage Encoder Decoder Model

One of the most intuitive ways to implement a pivot based NMT model is to simply extend the

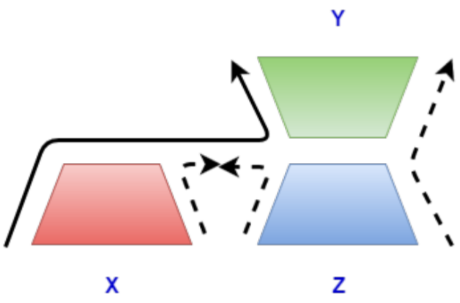


Figure 7: Correlation based Joint Encoder Decoder. From source (Saha et al., 2016)

current models we have to a set-up which comprises of two independent models. Let us say that our source language is A with the target language C and the pivot language B, then we have two distinct/independent models which translate from $A \to B$ and from $B \to C$. The models are independently trained on AB and BC parallel corpus respectively and the output from the first model serves as the input to the second model which in turn gives us our final output. The input can be both a text or an image to this model but the output is always natural text. The encoder depends on the input, if the input is an image then the encoder is a feed-forward neural network with features begin selected by a convolutional neural network otherwise it is a recurrent neural network. The decoder is always a recurrent neural network with the output being a sequence of words/characters. Figure 6 explains the set-up pictorially.

6.2 A correlation based joint encoder-decoder

The above model starts to crumble when a higher number of languages are involved. The model proposed by (Saha et al., 2016) uses a correlated encoder decoder approach to tackle this problem. The parallel corpus AB is used to train both the encoder for A and the encoder for B and the parallel corpus BC is used to train the decoder for C. The encoders of A and B are correlated hence the encoder for B benefits from both the parallel corpus available. During training time, the model simultaneously learns to compute correlated representations from a_i and b_i and learns to decode c_i from b_i.

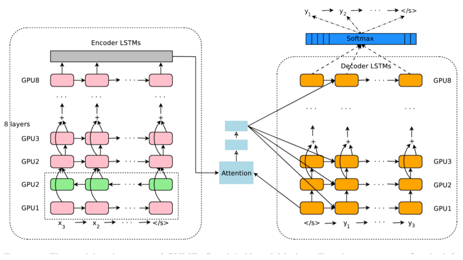


Figure 8: Google Neural Machine Translation Architecture. From source (Saha et al., 2016)

7 Multilingual NMT

In this section we cover the paper by Johnson et al. (2017) for multilingual translation.

7.1 System Architecture

The architecture of the multilingual system by Johnson et al. (2017) is exactly similar to the model proposed by Wu et al. (2016) as shown in the Figure 8 except with a very significant difference. To be able the use multilingual data within a single system, a small modification to the data is done, which is an artificial token is inserted to the beginning of the input sentence to indicate the required target language for this source sentence. For example consider this sentence from En $\to$ Es pair of sentences:

```
Good morning -> Buenos das.
```

This will be changed to

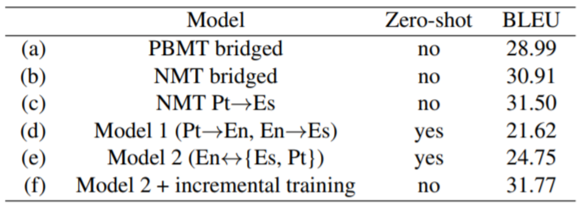
```
<2es> Good morning -> Buenos das.
```

The token added signifies the target language for the translation and in this case it is Spanish. After the required modifications, the parallel corpus is used to train the system.

7.2 Zero Shot Translation

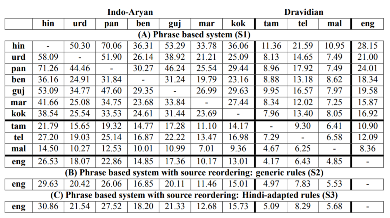
Zero shot refers to ”Translation between pairs never seen explicitly by the system before”. One of the most common ways to achieve this translation is to use a *bridging* or *pivot* language. The source sentences are first translated to the bridge language and then the bridge language sentences are converted to the target language sentences. There are problems with this approach

- The translation time doubles since we have to train two models instead of a single one.
- The translation quality has a potential to dip while translating from/to the intermediate language.



	Model	Zero-shot	BLEU
(a)	PBMT bridged	no	28.99
(b)	NMT bridged	no	30.91
(c)	NMT Pt→Es	no	31.50
(d)	Model 1 (Pt→En, En→Es)	yes	21.62
(e)	Model 2 (En↔{Es, Pt})	yes	24.75
(f)	Model 2 + incremental training	no	31.77

Figure 9: Google Neural Machine Translation Architecture. From source (Saha et al., 2016)



	Indo-Aryan							Dravidian			
	hin	**urd**	**pan**	**ben**	**guj**	**mar**	**kok**	**tam**	**tel**	**mal**	**eng**
(A) Phrase based system (S1)											
hin	-	50.30	70.06	36.31	53.29	33.78	36.06	11.36	21.59	10.95	28.15
urd	58.09	-	51.90	26.14	38.92	21.21	25.09	8.13	14.65	7.49	21.00
pan	71.26	44.46	-	30.27	46.24	25.54	29.44	8.96	17.92	7.49	24.01
ben	36.16	24.91	31.84	-	31.24	19.79	23.16	8.88	13.18	8.62	18.34
guj	53.09	34.77	47.60	29.35	-	26.99	29.63	9.95	16.57	7.97	19.58
mar	41.66	25.08	34.75	23.68	33.84	-	27.44	8.34	12.02	7.25	15.87
kok	38.54	25.54	33.53	24.61	31.44	23.69	-	7.96	13.40	8.05	16.92
tam	21.79	15.65	19.32	14.77	17.28	11.10	14.17	-	9.30	6.41	10.90
tel	27.20	19.03	25.14	16.87	22.22	13.47	16.98	7.29	-	6.58	12.09
mal	14.50	10.27	12.53	10.01	10.99	7.01	9.36	4.67	6.25	-	8.36
eng	26.53	18.07	22.86	14.85	17.36	10.17	13.01	4.17	6.43	4.85	-
(B) Phrase based system with source reordering: generic rules (S2)											
eng	29.63	20.42	26.06	16.85	20.11	11.46	15.01	4.97	7.83	5.53	-
(C) Phrase based system with source reordering: Hindi-adapted rules (S3)											
eng	30.86	21.54	27.52	18.20	21.33	12.68	15.73	5.09	8.29	5.68	-

Figure 10: % BLEU Scores for systems S1, S2, S3. From source (Kunchukuttan et al., 2014)

This model has an advantage of *implicit bridging* for a pair for which no parallel corpus has been seen during the training time subject to the condition that the source and the target must have been seen as source and target languages individually at some some time during the training process. Figure 9 shows the results.

8 SMT Based Translation Systems

Up until now we have looked at the NMT based systems related to our current work but there exist works in the SMT field which we aim to replicate in NMT. We cover the Sata-Anuvadak system Kunchukuttan et al. (2014) here in order to throw some light on the work related to our current work.Kunchukuttan et al. (2014) built phrase-based SMT systems for 110 language pairs using the ILCI Corpus (Choudhary and Jha, 2014). There were four SMT models that were built namely:

- **Baseline Phrase based System(S1)**: These were simple Phrase Based SMT systems used as a baseline.

- **English-IL PBSMT with generic source side reordering rules (S2)**: A special feature of this model was that the source sentence were reordered in order to conform with the target side language word-order. Ramanathan et al. (2008) rule-based reordering system was used which is based on the the following transformation principle:
  $SS_mVV_mOO_mC_m \leftrightarrow C'_mS'_mS'V'_mO'_mO'$

  Here: S is the subject, O is the object, V is the verb, C_m the clause modifier, X' is the corresponding constituent in Hindi, X is S, O or V, X_m is a modifier of X.

- **English-IL PBSMT with Hindi-tuned source side reordering rules (S3)** : In this model instead of the using Ramanathan et al. (2008) rules for reordering, source side reordering rules from Patel et al. (2016) were used. These rules are a refinement of S2 with additional rules discovered through analysis of English-Hindi word order divergence.

- **IL-IL PBSMT with post-editing using transliteration (S4)** : Many Indian languages often share some words sometimes due to a common ancestor like Sanskrit or because of sharing the areas where the languages are spoken. So, for such languages, transliteration becomes useful while dealing with named entities and untranslated words. So the outputs from system S1 were were transliterated in a post editing stage which was automatic in nature and this forms our last system.

8.1 Analysis and Results

8.1.1 Translation Accuracy vis-a-vis Language Families

The paper when looking at the results based on the Language Families involved witnessed a very clear distinction between the *Dravidian* and the *Indo-Aryan* language families. The *Indo-Aryan* languages were the easiest to translate between because of features like same word order, similar case marking schemes and being less inflectional than Dravidian languages. Translation between English and Indo-Aryan languages suffered due to the structural divergence between the English and Indo-Aryan languages specifically the word-order. Translation between Dravidian languages was poor because of the rich morphological nature of Dravidian languages which results in several surface forms which cannot be completely covered by the data.

8.1.2 Effect of corpus size

Figure 11 shows the relationship between the corpus size and the %BLEU score. The effect of

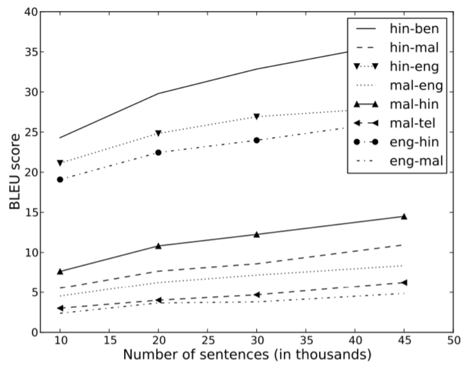


Figure 11: Training set size vs % BLEU. From source (Kunchukuttan et al., 2014)

increasing was best observed in the case of morphologically poor languages like *eng-hin* and *hin-ben*. Also quality of Indo-Aryan languages improves significantly with the increase in corpus size. However, other language pairs do not see the same amount of increase.

8.1.3 Morphological Complexity

It was observed in the work that languages belonging to the Dravidian family were the most difficult to translate between even thought they shared characteristics like word order, morphological structure etc. So **Translation Model Entropy** was used as a measure to study the effect of morphological complexity on uncertainty of translation. Entropy was observed to be high for Dravidian language pairs and low for Indo-Aryan pairs strengthening the hypothesis that translation model entropy would be low for morphologically poor systems and high for morphologically rich pairs.

8.1.4 Source Side Reordering

The paper implemented two types of source reordering rules. The Figure 10 shows both the systems S2 and S3 lead to an increase in the BLEU score. The improvement of average BLEU score over all language pairs for system S3 was 21.5% compared to an increase of 15.1% for system S2. Reordering was found to help Dravidian languages more than the Indo-Aryan Languages.

8.1.5 Post Editing using Transliteration

The effects of using transliteration were evaluated using the translation recall. Looking at Hindi, Marathi and Konkani, there was a 1.72% increase in recall due to the usage of transliteration. Recall decreases for all language pairs with *Devanagari* as the source side script and Punjabi as the target. The maximum increase in recall was observed for language pairs spoken in geographically adjacent areas. A more complex transliteration system which handles issues like *schwa* deletion, *chillu* characters in Malayalam script etc.

9 Conclusion

In this paper, we presented the fundamentals required to understand NMT. We then dived deep into word segmentation and MDL principle. We then presented the survey on Pivot Based Neural Machine Translation. We also provided an overview of the work currently done in the field of Multilingual NMT. We then presented one of the most important related works in the field of SMT. We observed several different techniques to implement Pivot based NMT and also looked at Subword NMT. We also covered a predecessor of the current work in the field of SMT.

References

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv e-prints*, abs/1409.0473.

Choudhary, N. and Jha, G. N. (2014). Creating multilingual parallel corpora in indian languages. In Vetulani, Z. and Mariani, J., editors, *Human Language Technology Challenges for Computer Science and Linguistics*, pages 527–537, Cham. Springer International Publishing.

Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., et al. (2017). Googles multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Koehn, P. (2017). Neural machine translation. *CoRR*, abs/1709.07809.

Kunchukuttan, A., Mishra, A., Chatterjee, R., Shah, R., and Bhattacharyya, P. (2014). Shata-anuvadak: Tackling multiway translation of indian languages. In Chair), N. C. C., Choukri, K., Declerck, T., Loftsson, H., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland. European Language Resources Association (ELRA).

Patel, R. N., Gupta, R., Pimpale, P. B., and M, S. (2016). Reordering rules for english-hindi SMT. *CoRR*, abs/1610.07420.

Phillip, G. (1994). A new algorithm for data compression.

Ramanathan, A., Hegde, J., Shah, R. M., Bhattacharyya, P., and M., S. (2008). Simple syntactic and morphological processing can help English-Hindi statistical machine translation. In *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-I*.

Saha, A., Khapra, M. M., Chandar, S., Rajendran, J., and Cho, K. (2016). A correlational encoder decoder architecture for pivot based sequence generation. *CoRR*, abs/1606.04754.

Sennrich, R., Haddow, B., and Birch, A. (2015). Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA. MIT Press.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.