Neural Machine Translation Literature Review

Yashasvi Asthana
yxa150@psu.edu

Tianyang Zhao
tzz4@psu.edu

Abstract

This paper is a concise survey of the recent advances in Neural Machine Translation. The deep neural architectures have come a long way since the state-of-the-art Statistical Machine Translation systems. From the use of Recurrent Neural Networks to the current state-of-the-art Transformers and Attention based network, there have been many gradual steps to improve the performance of deep neural networks on machine translation. This survey is divided into 4 major topics that cover the NMT journey thus far - RNN in NMT, CNN in NMT, Attention Mechanisms in NMT and Pre-training based improvements.

1 Introduction

One of the most researched problems in Natural Language Processing (NLP) is Machine Translation (MT). It targets the problem of translating text from one language to another. Machine Translation has seen 3 primary waves in its development since its conception - the Rule-based MT, the Statistical MT and the Neural MT. All the latest state-of-the-art research is based on the Neural Machine Translation (NMT). NMT is a deep learning based approach which basically acts as a sequential model that predicts one word at a time, unlike the earlier phrase-based approaches used in SMT.

We will divide our Literature Review into 4 parts: RNN-based models, CNN-based models, attentions in NMT, and pre-training approaches in NMT. These parts will highlight decades long progress in NMT and the gradual improvements made to some of the approaches.

2 Recurrent Neural Networks in NMT

RNN-based NMT approaches follow the basic end-to-end structure and implement both the encoder and decoder as two RNNs. The motivation behind this design decision is that RNNs allow the networks to take not only the current word but also some result from the previous time-step as the inputs for each calculation. This feature enables each computation to take account of the contextual information. This is extremely critical for Machine Translation because words in a sentence are never isolated elements.

In the following subsections, we will explore the architecture, the training method, and the inference method of RNN-based NMT in details. We will use sequence to sequence (Seq2Seq), RNNsearch, and Google Neural Machine Translation (GNMT) as example models. We will also compare and contrast them to demonstrate the basics that every RNN-based NMT model follows on the three aspects and the variations these example models made to improve their performance.

2.1 Architecture

2.1.1 Seq2Seq

Seq2Seq (Sutskever et al., 2014) is composed of two RNNs of any type, with LSTMs being a popular choice due to its ability to handle the long dependencies compared to others. One RNN works as an encoder, and the other works as a decoder. Figure 1 shows how the encoder and decoder work in the Seq2Seq model during a machine translation task in a chronological order.

One thing worth noting is that the input sequence is processed in the backward direction. This makes the words at the beginning of the source language sentences to be very close to the corresponding ones in the target language sentences, which significantly reduces the minimal time lag and introduces short term dependencies. As a result, this method mitigates the effect of long term dependencies on the performance and makes the Seq2Seq not only be capable to deal with long sentences but also do

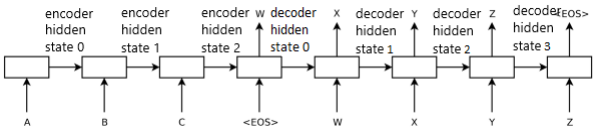

Figure 1: An example demonstrating the model takes the input sequence "CBA" and outputs "WXYZ". Each rectangle represents an RNN. The arrows pointing upward to a rectangle are the inputs of that RNN. The arrows pointing outward and to the right are the hidden states passed to the next time step while the ones pointing outward and up are the actual outputs.

well with them.

2.1.2 RNNsearch

RNNsearch (Bahdanau et al., 2016) is a research work published soon after the success of Seq2Seq (Sutskever et al., 2014) and is its variation. In terms of architecture, RNNsearch has two distinguishing designs: it incorporates a bidirectional RNN for the encoder and computes context vectors with various lengths by basic attention mechanism which are specific to each time step for the decoder. How this new architecture produces an output word can be seen in Figure 2.

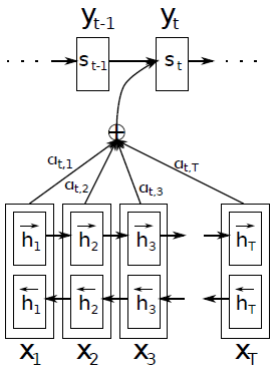

Figure 2: This illustrates how the model outputs the word y_t at time step t by the last produced word y_{t-1}, the current decoder hidden state s_t, and the context vector c_t formed by the encoder hidden states $h_1, \ldots, h_T$ and the attention weights $\alpha_{t,1}, \ldots, \alpha_{t,T}$ that depend on the last decoder hidden state s_{t-1}.

From Figure 2, we can easily spot the use of bidirectional RNN as there are both the arrows pointing to the right (forward direction) and the ones pointing to the left (backward direction). So, this structure embraces the regular way to process the sequential data from the first element to the last element and the creative way from Seq2Seq model that reverses the order of the input to introduce short term dependencies. The bidirectional RNN concatenates the forward hidden state and the backward hidden state for each word as its annotation. However, to be able to capture the context information in both directions, a bidirectional RNN needs to sacrifice its efficiency and is slower than a regular RNN. Thus, RNNsearch model only implements the bottom layer encoder as a bidirectional RNN and keeps the encoders at other layers as unidirectional RNNs.

As for another advancement of this model compared to prior work, the incorporation of basic attention mechanism comes from the attention weights for the concatenation of each word. These weights are specific to each time step, as indicated by t in the notation, and computed based on the last decoder hidden state. Then, the context vector of the current time step will be produced by summing the annotations multiplied with their weight and sent to the decoder. So, the context vectors no longer have a fixed length. With this context vector, the decoder knows what parts of the input annotations to focus on to produce the current output word. Unlike Seq2Seq, the context vector in RNNsearch model refers to a distinct element other than an encoder hidden state or a decoder hidden state.

2.1.3 GNMT

GNMT (Wu et al., 2016) is a model that is intended to deal with some limitations of the previous RNN-based approaches: slow training and inference speed, poor performance with rare words, and occasionally leaving some words in the input sentence untranslated. It applies quantized inference with reduced precision arithmetic, the wordpiece model, and a coverage penalty to deal with these issues, respectively. Architecture-wise, GNMT is largely inspired by RNNsearch (Bahdanau et al., 2016) where a lot of features are inherited such as the bottom bidirectional RNN layer (Figure 3).

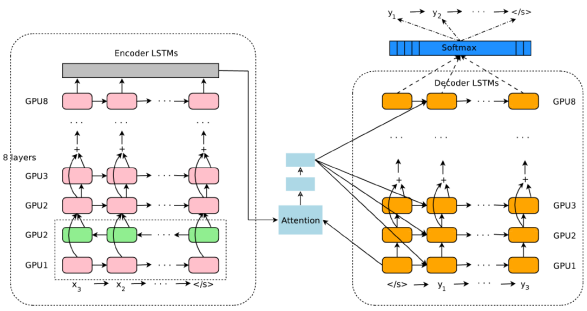

Figure 3: The full view of the architecture of GNMT. The + here refers to element-wise addition implying a residual connection.

However, the most notable architectural innovations by GNMT are the usage of more encoder and decoder layers and the introduction of residual connections. With more layers, a RNN-based model can capture some subtle relationships between the source language and the target language better. Nevertheless, simply stacking the layers introduces the problems of vanishing and exploding gradients, which is why RNN-based models usually used four layers in the experiments. The residual connection serves to improve the gradient flow, where the result of the element-wise addition between the outputs of the previous two layers is used for the computation of the current layer.

2.2 Training Method

All three models mentioned above share plenty of similarities on the training method. At first, the hyperparameters (like the number of training steps) need to be specified, while other parameters are randomly assigned a value based on a probability distribution of the trainer's choice. Before each sentence in the source language is fed into the model, every word in the sentence needs to be embedded into a vector so that the model can process them. During training, the parameters are updated only after the model has processed a certain amount of sentence pairs. This subset of the data is named a mini-batch (or simply speaking a batch), and the batch size is one of the hyperparameters. Due to the existence of the ground truth, the decoder takes the ground truth of the previously generated word (rather than the actually produced word) to output the current word.

GNMT (Wu et al., 2016) also introduces some novel elements. Seq2Seq (Sutskever et al., 2014) and RNNsearch (Bahdanau et al., 2016) have the same objective function which computes the sum of the log probabilities of producing the correct sentences given the source sentences. On the other hand, GNMT incorporates a score by the sentence-level comparison between the ground truth sentence and the produced sentence. Also, GNMT employs reduced precision arithmetic during training. This technique limits the number of bits used to represent the encoder hidden state values and outputs in order to speed up training and inference with little sacrifice on the translation quality.

2.3 Inference Method

All three models use beam search while GNMT (Wu et al., 2016) incorporates certain refinements. As the most basic inference approach, greedy finds the best output word according to the conditional probability given the inputs and previous outputs for each time step. On the other hand, with k being the beam width hyperparameter, beam search keeps k best output words at every time step. Starting from the time step 2, each of these k words will be appended to the end of each retained outputs to form a sub-sentence. There are k^2 possibilities, and only k of them are kept to be passed to the next time step. In the end, there are k target sentences, and the one with the highest conditional probability will be chosen as the final output.

GNMT introduces a new scoring function for beam search and applies two improvements: sentence length normalization and a coverage penalty. Sentence length normalization is intended to deal with the issue that the log probability is biased toward shorter sentences. The coverage penalty provides extra score to the possible output sentences where all the words in the input sentence are translated. This serves to prevent the case where some input words are missed out during inference.

3 Convolutional Neural Networks in NMT

The Recurrent Neural Network architecture was an intuitively good for sequence understanding and generation. The major drawback of such an architecture is that it is very slow at training and inference, due to the fact that it analyzes the sentence word by word. CNN-based architectures emerged to solve this problem.

Convolutional Neural Networks have been majorly used in image related learning, but the idea of feature extraction over a sequence of pixels suggested that they could be used in NMT as well. Due to its structure, CNN models allow parallel computations. Hence, these models are much faster while training and evaluation. The CNN-based models also helped in resolving the gradient vanishing problem while interpreting a longer sentence. Until the introduction of the Attention Mechanism, CNN-based NMT showed promising results. In this section, we will see some of the best CNN-based models that were proposed for NMT.

3.1 ByteNet

One of the first successful CNN-based NMT approaches was proposed by Kalchbrenner et al. (2016). They suggested a one-dimensional con-

volutional neural network, called ByteNet, that divides the NMT task into two parts - encoding the source and decoding the target. ByteNet is completely comprised of layers of convolutional blocks, which not only reduces the computation time, but also improves the state-of-the-art in character-level language modelling by outperforming earlier RNN-based approaches. The earlier attempts of using CNN-based architectures were either super-linear in the length of the source and the target sentences, or they had a latent space representation with a constant size. This meant that these models would suffer while interpreting or generating longer sequences.

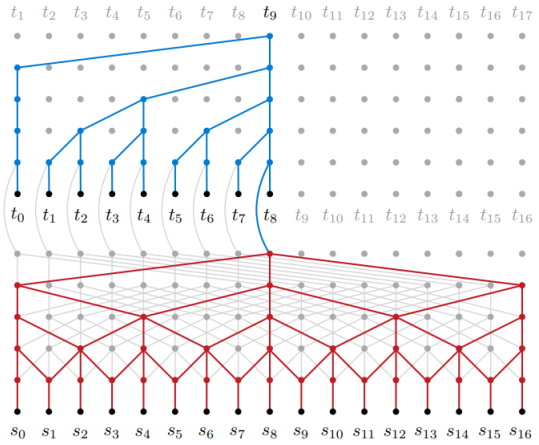

Figure 4: The Encoder-Decoder architecture of ByteNet. The target decoder (blue) is stacked on top of the source encoder (red).

ByteNet tries to solve these problems by stacking the decoder on top of the encoder outputs, as in Figure 4, in a manner that preserves the temporal resolution of the sequences. This way, the encoder does not have to compress the source into a fixed size representation like in Kalchbrenner and Blunsom (2013) or pool over the source representation with attentional pooling like in Bahdanau et al. (2016). ByteNet also uses a dynamic unfolding mechanism which helps the network process sources and targets with differing lengths. This requires the decoder to unfold step by step over the encoder representation until the decoder outputs an end-of-sequence symbol. Given the source and target respective lengths $|s|$ and $|t|$; the length of the encoder representation, $|\hat{t}|$, is decided by the formula:

$$|\hat{t}| = a|s| + b \quad (1)$$

The parameters a and b should be set beforehand based on a general prior, but need not be very accurate. For example, Kalchbrenner et al. (2016) choose $a = 1.20$ and $b = 0$ when translating from English into German based on a study that indicates German sentences tend to be a little longer than their English counterparts.

3.2 ConvS2S

ByteNet performed well on character-level translation tasks but failed at word-level translation. It had the promised advantage in training speed due to its single layer CNN-based architecture. However, unfortunately, this also made the model behave very poorly in capturing long-term dependencies between words which would have been found in high-level convolution layers.

Gehring et al. (2017) introduced a new model that uses gated linear units (Dauphin et al., 2017) and residual connections (He et al., 2015) as well as added an attention mechanism in every decoder layer to solve some of the problems introduced when using CNN-based architectures. The encoder and decoder share a one-dimensional block structure followed by a non-linear function.

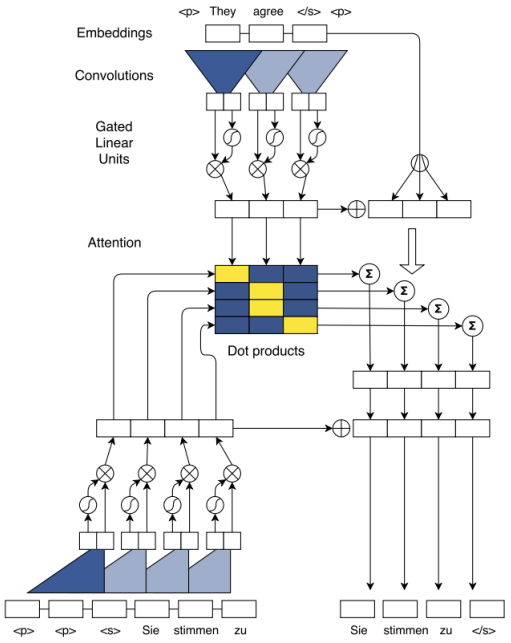

Figure 5: The source sentence is Encoded (top) and all the attention values (center) are computed using the decoder context representation (bottom left) and the encoder representations. Conditional inputs from the attention (center right) are added to the decoder states to predict the target words (bottom right).

The multi-step attention used in ConvS2S combines the current decoder state with an embedding

of the previous target element. The attention of the first layer determines the relevant information from the source which is the fed to the second layer that takes this information into account when computing its attention. This makes sure that the attention information is carried throughout the layers. Figure 5 from Gehring et al. (2017) illustrates the batching inside the model during training. The model achieved a new state-of-the-art on several translation benchmarks and showed how attention can be used with convolutional networks to improve the performance significantly.

3.3 SliceNet

Inspired from the success of attention mechanism in ConvS2S, Kaiser et al. (2017) adopted it with similar convolutional neural network. Nonetheless, instead of normal convolutional layers, they decided to use depthwise separable convolutions (Chollet, 2017). Depthwise separable convolutions have been shown to increase the efficiency of the network while significantly improving the image classification accuracy. For example, if we have c channels and the receptive field of size k, the number of parameters for a regular convolution would be kc^2 while the same for separable convolution would be $kc + c^2$. When k is very small compared to c the c^2 term dominates. Kaiser et al. (2017) introduced a super-separable convolution based on g groups to further improve the network's efficiency. The final size (cost) of a super-separable convolution with g groups is $kc + \frac{c^2}{g}$. This helped them reduce the number of parameters and increase the size of convolution windows, removing the need for filter dilation. SliceNet outperforms ConvS2S and ByteNet with similar parameter counts and achieves the state-of-the-art result with a BLEU score of 26.1 on WMT English to German dataset.

4 NMT with Attention Mechanism

Ever since (Bahdanau et al., 2016) proposed RNNsearch , the incorporation of the attention mechanism in NMT models, specifically in RNN-based ones, has become more and more popular which eventually becomes a standard. In the following subsections, we will first explain the motivation behind the attention mechanism. Then, we will explore multiple attention structures in details, namely the basic attention by RNNsearch, global and local attention (Luong et al., 2015), and self-attention (Vaswani et al., 2017). We will also compare and contrast them to demonstrate the advancement in the field of attention mechanism.

4.1 Motivation behind Attention Mechanism

The motivation is the inflexibility and the difficulty with long range dependencies for the basic RNN-based models without attention like Seq2Seq (Sutskever et al., 2014). Since the encoder in Seq2Seq model simply passes the last hidden state as the context vector to the decoder, the decoder only has a mere overview of the input when outputting each word. An analogy of this scheme is to read the whole sentence a few times then try to translate it without looking at any specific word again. If the input sentence is very long, there could be two harmful consequences caused by the context vector's characteristic that its length is fixed. One is that the context vector gradually "forgets" the details from the start of the sentence. The other is that it fails to capture information which is insignificant to the whole sentence but critical for translating certain words from a local view. It would be intuitive to consider increasing the dimension of the context vector to deal with these issues. However, this approach would make the RNN-based models even slower, given that their training and inference speed is already a notable drawback. Also, when the model deals with short sentences, context vectors with high dimensionality would be unnecessary and waste memory space. These limitations motivate people to think about feasible solutions.

RNNsearch (Bahdanau et al., 2016) pioneered their answers. The model changes the way the context vectors work. They no longer have a fixed length and are unique for each time step. They take all the encoder hidden states into consideration and show which are the most essential ones for outputting the translated word of the current time step. Moreover, self-attention as a novel attention structure even achieves parallelization in the encoder layers and less computational cost (see Table 1, credit: Vaswani et al. (2017)) as n is usually smaller than d. Therefore, the attention mechanism not only deals with the long range dependency issue of the traditional RNN-based models but also could provide speed-up by giving up sequential input processing.

4.2 The Vanilla Attention Structure

The computation of the vanilla attention includes three steps. Both RNNsearch and GNMT (Wu et al., 2016) incorporate this basic attention mech-

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$

Table 1: The comparison between different types of neural network for the encoder in NMT models. n is the length of the input sentence, d is the dimension of the output vector from the encoder, and k is the kernel size of convolutions.

anism in their respective models with very minor differences in the first step. This step is to calculate alignment scores for all the words in the input sentence by an attention function. This function takes the encoder hidden states and the last decoder hidden state as inputs in RNNsearch's implementation. On the other hand, the function takes the encoder hidden states and the previous word vector produced by the first decoder layer as inputs in GNMT's implementation. GNMT attempts to apply parallelization as much as possible to achieve higher speed. It does not fully follow RNNsearch here because the last decoder hidden state takes many prior steps to be generated, which makes it challenging to be incorporated into parallelization.

The second step is to normalize the alignment scores to obtain an attention weight for all the encoder hidden states, where both models use the natural exponential function for the normalization. The third step is to compute the context vector by the sum of all the encoder hidden states weighted by the attention values from the previous step.

4.3 Global Attention and Local Attention

Global attention (Luong et al., 2015) is largely the same as the vanilla attention (Bahdanau et al., 2016), but there are a few changes. Global attention uses the current decoder hidden state to calculate the alignment scores. As a result, before producing the attentional hidden state that will be used by the softmax layer to output the translated word, global attention only attempts to produce the context vector. On the other hand, the vanilla attention tries to produce the context vector and the current decoder hidden state since it depends on the context vector. So, the global attention is simpler in this sense. Also, this work provides experiments on two more alternatives for the attention functions: dot product and general, in addition to the concatenation used earlier by the vanilla attention. According to their experiment results, the general is the best performing one among the three.

Local attention (Luong et al., 2015) means that instead of looking at all the encoder hidden states for alignment scores, we put the focus on an aligned position and the D words surrounding it. Besides the smaller range of attention, local attention still uses the same methods to compute the context vector. Another notable difference between local attention and global attention is that context vectors produced by local attention have a fixed dimension $2D + 1$ due to its specified attention range.

The motivation for local attention is that a smaller attention window speeds up the training and inference process. Also, local attention is enough to produce translations with high quality in many cases where we only need the local information. On the other hand, it is challenging for local attention to capture long range dependencies in the input sentence, which global attention does quite well. Therefore, the best situation would be that we can take a reasonable guess regarding the dataset that if the sentences in source language have long range dependencies. Then, choosing whether to apply global or local attention accordingly.

4.4 Self-attention and Multi-head Attention

Self-attention is introduced by the Transformer model (Vaswani et al., 2017). Self-attention is not only the name for the novel attention mechanism but also the name of the neural network layer used in the encoder. Thus, Transformer completely abandons the RNNs as the encoder and the decoder along with the idea to process the words in the input sentence sequentially across different time steps. Instead, they are processed simultaneously as their embeddings are stacked as a matrix in Transformer.

In the beginning of obtaining attention information, this matrix is multiplied by three different weight matrices (the weights are also learned during the training like other parameters) to compute the query matrix, the key matrix, and the value matrix, respectively. The query matrix and the key matrix are multiplied together first, and we apply

scaling then softmax function on the result. This is analogous to the computation and normalization of alignments scores in previous attention mechanism. Afterward, the resulting matrix is multiplied with the value matrix to obtain the attentional matrix where each row captures the attention information for each input word from the first one to the last one. Again, this is analogous to the computation of the context vector. However, the attentional matrix here is sent to the following feed-forward network in the same encoder layer, instead of being sent to the decoder like the context vector.

Multi-head attention creates K attention heads (K is one of the hyperparameters). These attention heads divide up the input word embedding matrix into K parts where each has the same number of rows but $\frac{dimension}{K}$ columns. The attention heads have their respective three weight matrices with new sizes to adapt to the column number change. The three weight matrices for each attention head are initialized with different values due to random initialization. As a result, the attentional matrix for each attention head is also different. In the end, these attentional matrices are concatenated together and multiplied with another weight matrix to obtain the final attentional matrix that holds the information from all the attention heads. Multi-head attention almost always performs better than self-attention because the effect of some extreme initial values in the three weight matrices that are used to compute the query, key, value matrices are minimized.

5 Pre-training for NMT

One of the major issues that all NLP related tasks have to deal with is the representations of the words. The earliest methods in NLP used one-hot encoding to represent each word in the dictionary uniquely. This meant that every word is independent and has no syntactic or semantic relation to any other word in the dictionary. This is not how we look at words. For humans, some words are closer to others in some sense and others are similar in a different context. Hence, the idea to represent words as continuous vectors in multi-dimensional space emerged. Now that pre-training is being used in every domain including computer vision, NLP researchers are eager to find ways to pre-train huge models on unlabeled corpora such that it improves the model's performance in downstream NLP tasks.

5.1 Word2Vec

The earlier ideas of learning distributed representations of words by using neural networks proved to be better than the well-known Latent Semantic Analysis method. The drawback of neural network language modeling is that it requires huge amounts of data and is computationally expensive. Mikolov et al. (2013) tried to solve this problem by introducing 2 different architectures: continuous bag-of-words (CBOW) and continuous skip-gram model (Figure 6). The CBOW architecture is similar to a feedfoward neural network language model, where the non-linear hidden layer is removed and a projection layer is shared for all words. The authors achieved best performance by building a log-linear classifier with four future and four history words at the input, with the objective to correctly classify the middle (current) word. The other architecture, continuous skip-gram model, is similar to CBOW but instead of predicting the current word based on the context, it predicts words within a certain range before and after the current word.

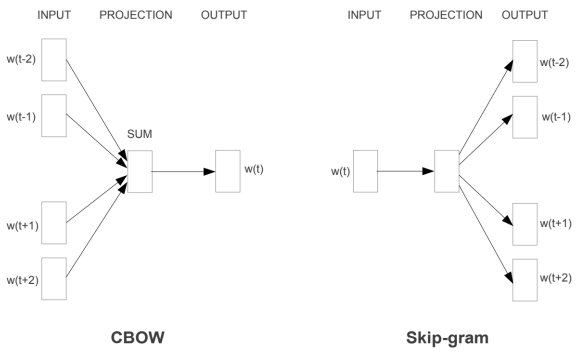

Figure 6: New proposed model architectures, CBOW and Skip-gram.

5.2 BERT

The Word2vec system was good at generating word vectors but was not tested on fine-tuning for NMT task. Nevertheless, it showed that learning word embedding lays a good groundwork for any downstream NLP task. Ever since BERT was introduced (Devlin et al., 2018), pre-training of models on huge corpus has received a significant amount of attention and has become a new trend in the field of NLP. As pre-trained models, they are flexible and can be fine-tuned to various NLP tasks.

Clinchant et al. (2019) proved that NMT is one of those NLP tasks that can take advantage of improved word embedding. A typical NMT model has two parts - encoder and decoder. A pre-trained

language model, like BERT, can inject prior knowledge in the encoder part by providing rich contextualized word embedding learned from monolingual corpus (source language). BERT follows a masked language modeling objective where the input is a set of words, which are randomly masked, and the model is tasked to predict the next word.

5.3 Generative Pre-trained Transformer

Recently, GPT3 (Brown et al., 2020) with around 175 billion trainable parameters was released by the OpenAI team. The idea behind pre-training such a large model is that current systems rely on fine-tuning the models on tens of thousands of examples for the downstream tasks. This is not what humans do, and maybe a very large model which is trained on huge amounts of data would require less or no fine-tuning while trying to perform the downstream tasks. The few-shot learning results of GPT3 were great, and the model sets a new state-of-the-art in NMT specifically. This shows that huge models can act as few-shot learners.

6 Future Directions

As the previous sections demonstrated, we have seen NMT obtain significant attention and go through huge development over the recent years. At the same time, researchers are actively finding the existing limitations and proposing new approaches to bring further improvements. We will briefly introduce some of them in this section.

1. Like other approaches based on deep learning, NMT is naturally data dependent, which is especially the case for advanced models with more parameters to train. Therefore, this creates a difficulty for NMT models when they are used to do translation involving less popular languages with much fewer available corpora. Sennrich et al. (2015) proposed a method to enhance the dataset with monolingual data to mitigate the issue of low resources. The results from this work illustrate that certain novel approaches we utilize for low-resource NMT, such as back-translation, can also be used to improve the performance on regular NMT tasks. Sennrich and Zhang (2019) provides adaptive practices for neural approaches when dealing with low-resource machine translation tasks and shows that NMT still outperforms PBSMT (phrase-based statistical machine translation) in this field. This demonstrates low-resource NMT is promising and worth more attention for the future work.

2. All the current state-of-the-art NMT models use Transformers, which follow an intuitive auto-regressive nature. This means that every next word generation is dependent on the previously generated outputs. This helps us achieve great accuracy, but at the cost of large inference times. Gu et al. (2018) introduced a Non-Autoregressive NMT model with a parallel decoder. This meant that the inferences would be generated in parallel, hence decreasing the total time taken for the translation of a given text. Unfortunately, this model does not even come close to the accuracy of the current Transformer based models. We believe that further research in Non-Autoregressive methods will help advance the use cases of NMT based systems. With 10 times faster inferences, we could implement systems that would provide live translation results. This would also lead to a huge improvement in the cross-language live captioning systems.

3. Adversarial attacks on NMT based models are generally easy and based on perturbing the input until the system breaks, or worse learning the pattern such that an adversarial model can always break the system. This also introduces an uncertainty regarding the outputs generated by an NMT model. We need NMT models that can generate a coherent translation even with noisy or adversarial inputs. Hence, we believe that future works in NMT will be targeted towards explainable and robust models.

7 Conclusion

Research in NMT has come a long way but we are not yet close to human-level translations. Companies are still afraid of using NMT without supervision because of a low translation accuracy. Moreover, these models are still like a black box. Steps towards understanding and opening this black box have been slow and require much more attention. With huge models like GPT-3 being proposed, if we do not understand why they give a certain output or how can they be affected by adversarial input or noise, we cannot expect these great advancements to be applied in the real world setting.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2016. Neural machine translation by jointly learning to align and translate.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions.

Stéphane Clinchant, Kweon Woo Jung, and Vassilina Nikoulina. 2019. On the use of bert for neural machine translation. *arXiv preprint arXiv:1909.12744*.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition.

Lukasz Kaiser, Aidan N. Gomez, and Francois Chollet. 2017. Depthwise separable convolutions for neural machine translation.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent convolutional neural networks for discourse compositionality. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*, pages 119–126, Sofia, Bulgaria. Association for Computational Linguistics.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*.

Rico Sennrich and Biao Zhang. 2019. Revisiting low-resource neural machine translation: A case study. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 211–221, Florence, Italy. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Y. Wu, Mike Schuster, Z. Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, M. Krikun, Yuan Cao, Q. Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, Taku Kudo, H. Kazawa, K. Stevens, G. Kurian, Nishant Patil, W. Wang, C. Young, J. Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, G. S. Corrado, Macduff Hughes, and J. Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144.