Contemporary Methods of Neural Machine Translation

Final Project for CS 760, Spring 2017

Daniel Belongia, Patrick Cummings, Gabriel Elkind, Niko Escanilla, Brendan Krull

In this report, which was inspired by a recent New York Times article, "The Great A.I. Awakening" [1], we investigate contemporary methods in neural machine translation (MT). The NYT article details the recent paradigm shift in machine translation from phrase-based statistical systems to recurrent neural nets, as best exemplified the remarkable success of the Google Neural Machine Translation (GNMT) model, which now powers many language pairs of Google Translate [2]. Here, we expand on that paradigm shift by first examining the theory and historical performance of statistical machine translation (SMT) in Section 1. Then in Section 2, we look at some specific features of neural machine translation that have recently enabled neural nets to outperform other models. While we draw upon a large body of literature to provide the relevant technical details and historical notes, our focus remains on the GNMT model due to its unparalleled performance and prominence on the web. In Section 3, we provide specific information about results and implementation of the GNMT model.

1 Background Information and Historical Notes on Machine Translation

Given a textual sentence in a human language (the source language), the task of machine translation is to translate that sentence into a suitable representation in another language (the target language). Typically an MT model is trained using some algorithm and a set of texts that have already been translated by humans into one or more languages. Until as recently as 2016, phrase-based SMT was the primary model upon which machine translation operated. Despite its popularity, SMT suffered from limitations that caused its performance to reach a plateau. In this section, we first present a common metric for evaluating the output of MT models that we will refer to later in the report. Then we look at the theory and historical performance of phrase-based SMT models, and introduce the use of deep neural networks for MT.

1.1 Metrics for Evaluating Performance of Machine Translation

A common algorithm for evaluating translation performance is BLEU, a method first developed in 2002 [3]. It scores machine translations on a scale from 0 (no match) to 1 (perfect match) against human-translated references. Acting on an entire body of work, sentence by sentence, BLEU counts the number of words in the machine translated sentence that match words in the human translated reference. That number is then divided by the number of words in the machine translated sentence. Once all of the sentences in the work have been accounted for, and their scores averaged between them, a brevity penalty between 0 and 1 is multiplied by the result. The penalty is 1 when the word count of the texts match, and decreases as the machine translation's length increases [3].

The BLEU algorithm scores correlate well with human evaluation of the same translated texts, and is used alongside side-by-side (SxS) evaluations in Google's own evaluation of translations. An SxS evaluation scores the translation from 0 to 6, where 0 is nonsense, with no information retained, and 6 is perfect retention of meaning and all grammar is correct [4].

1.2 Phrase-Based Statistical Machine Translation

Methods and Theory of SMT

The modus operandi of early machine translation efforts was rule-based systems, which required manual specification of grammar and vocabulary, and often produced poor results [5]. In the early 1990s, however, IBM developed a statistical approach to translation in which large bodies of human-translated text, referred to as parallel corpora, were used to automatically determine how individual words are translated [5]. Phrase-based systems later built upon this approach by segmenting parallel corpora into larger linguistic units: phrases (or more properly, phrasemes) [6]. All possible translations of each source phrase were then tabulated in a phrase table, which was then used to decode a new phrase from the source to target language. Here we'll consider the specific task of translating from a foreign language to English, since phrase-based SMT typically operated with English paired with another language[1].

Before populating a phrase table for a given language pair, word alignments must be created from a suitable parallel corpus, such as proceedings from the United Nations. For a given sentence pair, this word alignment maps each word in English to the most likely corresponding word or words in the foreign language [7]. A publicly available algorithm such as Giza++ [7] [8] can be used to compute the word alignments, and these alignments are tabulated to create the phrase table.

The decoding of a foreign sentence into English is performed by maximizing the probability of translating the foreign sentence *F* into the English sentence *E*: $\text{argmax}_E P(E \mid F)$ [7]. The noisy channel model, a widely used formulation in statistical machine translation, decomposes this formula into a language model and a translation model [7]:

$$E^* = \text{argmax}_E \, P(E \mid F) = \text{argmax}_E \, P_{TM}(F \mid E) \, P_{LM}(E) \tag{1}$$

Here, $P_{TM}(F \mid E)$, or the probability of the English sentence *E* given a foreign sentence *F*, is the translation model. $P_{LM}(E)$, or the probability of the English sentence, is the language model, which can be derived from a monolingual corpus.

In practice, the foreign sentence to be decoded is segmented into a sequence of phrases $\{f_1, f_2, \ldots, f_N\}$, with a uniform probability distribution over all possible segmentations [7]. The corresponding phrases in English are $\{e_1, e_2, \ldots, e_N\}$ [7]. The translation term of the noisy channel model can then further be decomposed into

$$P_{TM}(F \mid E) = \prod_{i=1}^{N} P(f_i \mid e_i) \tag{2}$$

For each consecutive phrase, this probability can be estimated as the relative frequency of collected phrase pairs tabulated in the phrase table [7]:

$$P(f_i \mid e_i) = \frac{\text{count}(f_i, e_i)}{\sum_f \text{count}(f, e_i)} \tag{3}$$

Depending on the specific formulation, additional terms may be appended to P_{TM} to calibrate output length or enforce re-ordering of the English phrases [7]. An entirely separate algorithm may be used to permute the translated phrases into their final order [6]. Instead of **Equation 1**, a log-linear model is sometimes used to introduce other sources of information into the model [5]:

[1] For instance, for a language translation task between two non-English languages, $L_S \Rightarrow L_T$, Google Translate's phrase-based SMT model would first translate the source language into English, and then from English into the target language: $L_S \Rightarrow \text{En} \Rightarrow L_T$.

$$E^* = argmax_E \sum_i g_i(E,F)\,\lambda_i \quad (4)$$

Here, each $g(\cdot)$ could be $P(F \mid E)$, $P(F \mid E)$, $P(E)$, or some other function, and each λ is a weight indicating the contribution of each $g(\cdot)$ to the total function.

Since the search space for the sentence that maximizes the probability function is so large, a beam search is typically employed for decoding [7].

Historical Performance and Challenges of SMT

In the early days of MT speculation, several unsuccessful (yet well-funded) longitudinal studies occurred, notably collaborations of IBM with Washington University and IBM with Georgetown University.

In 1966 the Automatic Language Processing Advisory Committee infamously published a paper, "The ALPAC Report" [9], that compelled funding providers to cease investment in MT research. In this report, the authors argued that machine translation was not possible with then available technology. It outlines 8 years of work at Georgetown University which failed to produce useful results.

The authors provided a quantitative breakdown of speed, quality and cost to determine that running machine translation algorithms at the time of publication was *significantly* more costly than hiring human translators for the same task. As a result of this publication, the study of MT largely halted globally for nearly two decades.

In the 1980s, MT saw a resurgence of research in Japan, motivated by the emergence of parallel computation in 5th generation computers [10]. Most of these systems used brutish statistical algorithms which relied on large training sets and did not account for syntactic and semantic rules.

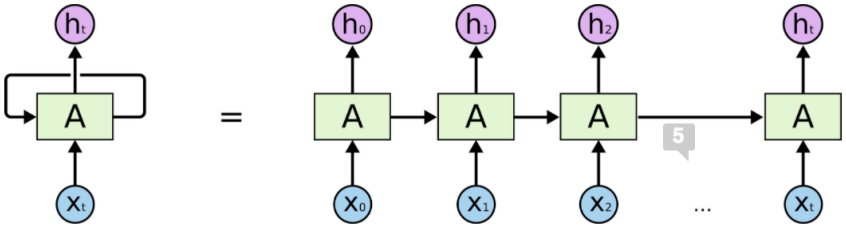
Among the most significant hurdles within pure SMT is improper control of contextually determined associations between words in sequence, known as long-term dependency. For example, the amount that noun *A* should influence the translation of action *B* depends on whether *A* is an alias of another noun, whether it appears before *B*, whether *A* is the subject or object of *B*, etc. How much should be remembered and forgotten as one iterates through the sentence? Early attempts to hardcode understanding of linguistic concepts of anaphora and cataphora known as "Rule Based Machine Translators" [11]. Their performance varied, but few surpassed statistical machine translators.

[12] provides a quantitative overview of the performance of various cannonical MT attempts. A comparison of the translation quality of SMT and that of NMT is presented in Section 3 of this report.

1.3 Brief Overview of Neural Nets

Neural networks can take one of two major forms: feedforward or recurrent. This section will describe these networks and mention a specific type of network for each form.

Figure 1: An "unrolled" recurrent neural network. Image from [23].



Feedforward Neural Nets

In feedforward neural networks, information flows from input units to output units. Specifically, information flows through hidden layers. It has been proven both theoretically and empirically that these networks are able to approximate nonlinear target concepts/functions [13].

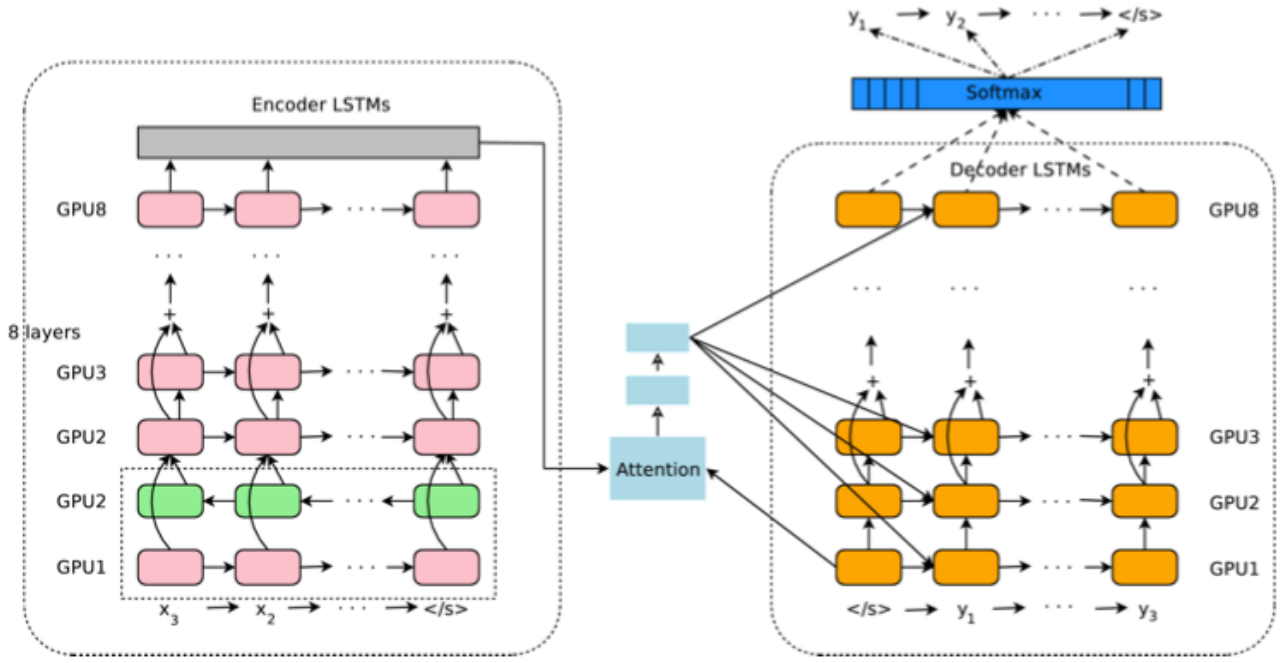
In a deep learning setting, an extension to the traditional multi-layer, feedforward network is the convolutional neural network (CNN). Unlike, the traditional approach, where the network is fully connected, CNNs use local filters (i.e. weights) and local connections. Thus, eliminating the need for layers to be completely connected. As a result, we get translation-invariant and distortion-invariant local features [14]. CNNs have become useful in applications that process array data. CNNs are used to process time series, acoustic, text, image, audio spectrogram, video, and volumetric data [14].

Recurrent Neural Nets

Recurrent neural networks (RNNs) contain loops/cycles that allow information to be passed to subsequent steps in the network [13]. In turn, RNNs can be characterized by having a chain-like structure, where each "chain" is a copy of the network (**Figure 1**). The driving force behind this representation of a neural network is rooted in biological neural networks, with the goal of connecting previous knowledge to the present. However, it has been shown empirically and theoretically that traditional RNNs perform poorly with long-term dependencies due to the problem of "vanishing gradient descent" [15].

[16] proposed a type of RNN called the Long Short-Term Memory (LSTM) network that handles long-term dependencies. The LSTM's architecture varies from other RNN architectures in that it contains memory cells and gate units that control the flow of information from previous states and inputs. Specifically, a memory cell consists of an input and output gate unit. These gates manage the read and write access to the cell. In turn, a unit in the hidden layer of an LSTM network is referred to as a memory block. This memory block may contain one or more memory cells [17]. The use of LSTM networks in machine translation has become a recent center of attention due to their ability to learn long-term dependencies, which previously prevented RNNs from being used in MT efforts. Most notably, Google's Neural Machine Translation system is built with a deep LSTM network [4].

Figure 2: Example model architecture of an encoder-decoder NMT network with LSTM neural nets and an attention module. This image is of the GNMT implementation [4]. Each green, pink, or orange box is a separate LSTM module.



2 Architecture and Theory of Neural Machine Translation

2.1 Overview of Model Architecture

Modern neural machine translation architectures typically feature a long short-term memory (LSTM) network with connected encoder and decoder modules. A high-level view of a neural machine translation model with this architecture is shown in **Figure 2**. Also shown are several additional features employed by the Google Neural Machine Translation model, including an attention module that dynamically focuses the decoder module on different parts of the input sequence, residual connections between stacked layers, and bi-directional encoding of the first layer [4]. An overview of the encoder-decoder approach is presented here, and additional architectural considerations are explained in the subsequent sections.

The encoder module takes as input a sequence of words $\{x_1, x_2, \dots, x_M\}$, typically consisting of a sentence in the source language and ending with a special "end of sentence" symbol [18]. Before each word is introduced to the first layer, it is converted into a fixed-length vector via a word embedding (WE) matrix, which can be pre-generated from the training material using an algorithm such as word2vec [19]. Each row of this matrix is a low-dimensional representation of a word in the training corpus known as a word embedding [20]. The use of word embeddings, as opposed to an integer index for each word in the training vocabulary, has the benefit of encapsulating semantic relations between words that are orthographically unrelated [20]. An oft-cited example of this is the observation that

$$\text{WE}(\textit{king}) - \text{WE}(\textit{man}) + \text{WE}(\textit{woman}) \approx \text{WE}(\textit{queen}) \tag{5}$$

Using the entire sequence in the source language represented as word embeddings, the encoder module of the LSTM network creates a list of fixed-size vectors $\mathbf{C} = [\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_M}]$ as its output. This entire list then serves as the context for the decoder module. For a source language input sequence *X*, the task of the entire encoder-decoder complex is to maximize the probability of the target language sequence *Y* given *X*, i.e. argmax $P(Y \mid X)$. That probability can be rewritten using Bayes' rule:

$$P(Y \mid X) = \prod_{i=1}^{N} P(y_i \mid y_0, y_1, \dots, y_{i-1}; x_1, x_2, \dots, x_M) \tag{6}$$

The target language sentence can therefore be decoded, word by word, by maximizing the probability of each word given the context **C** of the entire sequence (generated by the encoder) and the previously generated words:

$$P(y_i \mid y_0, y_1, \dots, y_{i-1}; \mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_M}) = P(y_i \mid y_0, y_1, \dots, y_{i-1}; \mathbf{C}) \tag{7}$$

In practice, the decoder consists of an RNN with a softmax output layer [18] [4]. Each time step, therefore, generates a probability distribution over the entire target language word embedding matrix. The target language sequence *Y* with maximum probability is typically found using a beam search algorithm [18], which keeps several partial solutions as candidates for the final solution.

[21] pioneered the specific LSTM architecture used to generate generic sequences with RNNs, and [22] marked the first published attempt to use it with machine translation. [23] could be consulted for a general overview of LSTM architecture, and [21] for the specific layout and equations used in sequence generation. The entire model can be trained with back-propagation through time using stochastic gradient descent [21] [22].

2.2 Attention Module

One early limitation in the design of NMT systems was the need for the decoder module to make its predictions using a fixed-length vector produced by the encoder [22]. For long input sequences, this could result in information loss, but in general it meant that too much information may be passed to the decoder at once. In 2016, [18] overcame this limitation through the use of an attention module that, for each new output word y_t, compresses

the variable-length context matrix **C** created by the encoder into a fixed-length vector $\mathbf{c_t}$ to pass on to the decoder. Although **C** does not change as each new output word is generated, the attention module creates a new context vector $\mathbf{c_t}$ with each new word, thereby "focusing" the attention of the decoder on a different part of the input. The method from [18] is summarized below; Google chose a similar implementation for GNMT [4].

The attention module is modeled as a feed-forward neural net whose input is: (1) the list of fixed-size vectors $[\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_M}]$ encoded from the input sequence, and (2) the previous hidden state $\mathbf{s_{i\text{-}1}}$ of the decoder. The module consists of a single, multi-layer perceptron such that

$$e_{ij} = \mathbf{v_a^T} \tanh\left(\mathbf{W_a s_{i\text{-}1}} + \mathbf{U_a x_j}\right) \tag{8}$$

where $\mathbf{v_a^T}$, $\mathbf{W_a}$, and $\mathbf{U_a}$ are weight matrices. For a given time step *i*, the weight for the j^{th} encoded vector in $[\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_M}]$ is computed as

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{M} \exp\left(e_{ik}\right)} \tag{9}$$

The interpretation of this is that each α_{ij} is the probability that the word in the target language y_i is aligned to a source word x_j. The final context vector $\mathbf{c_i}$ that is passed on to the decoder is computed as a weighted sum of the encoded context vectors from the input:

$$\mathbf{c_i} = \sum_{j=1}^{M} \alpha_{ij} \mathbf{x_j} \tag{10}$$

2.3 Residual Connections

Having deep, multi-layered LSTMs for encoder and decoder RNNs in NMT is associated with improve accuracy [4]. However, when stacking layers of LSTM networks, the vanishing gradient effect causes training of the networks to become a challenge [24]. This problem is mitigated by the introduction of residual connections, which add the memory state of the $i\text{-}1^{\text{th}}$ LSTM to the input of the $i\text{+}1^{\text{th}}$ LSTM (see **Figure 2**). This was demonstrated to allow and improve the training of deep RNNs by improving gradient flow [25].

2.4 Bidirectional Encoding

The bidirectional RNN (BRNN) was first proposed in 1997 [26] and has recently been used in speech recognition algorithms [21]. In a standard, unidirectional RNN, each consecutive time frame only contains information from the previous and current inputs. This can be a severe limitation for natural language processing, since inputs downstream of the current input may provide important context for the meaning of the input at hand. For example, in the phrase, "although it was brand new, the car broke down", the identity of the word *it* is not known until *the car*. This might not matter much in English, but in other languages, the form of the word *it* might depend on the grammatical gender or number of *the car*.

A BRNN overcomes this limitation by simultaneously processing sequences in forward and reverse. The forward RNN reads the input sequence $\{x_1, x_2, \ldots, x_M\}$ and creates a series of forward hidden states $\mathbf{h} = [\mathbf{h_1}, \mathbf{h_2}, \ldots, \mathbf{h_M}]$. The backward RNN reads the reverse sequence $\{x_M, x_{M-1}, \ldots, x_1\}$ and creates a series of backward hidden states $\mathbf{h'} = [\mathbf{h'_1}, \mathbf{h'_2}, \ldots, \mathbf{h'_M}]$. When passing this information on to the next layer, the two hidden states are concatenated: $[\mathbf{h^T}; \mathbf{h'^T}]$. Since the input sequence is always specified in its entirety in both training and testing of machine translation models, the implementation of a BRNN is straightforward, and the computations of each direction can be performed in parallel.

The use of the BRNN for neural machine translation was first proposed by Bahdanau, Cho, and Bengio in May 2016 [18], and was subsequently implemented in Google's neural machine translation model [4]. In [18], the

encoder and decoder each consist of two hidden layers (one forward and one backward), which was straightforward to implement using parallel computations. However, the team at Google desired to implement up to 8 hidden layers, and furthermore desired to use different GPUs in parallel for each layer by depth partitioning the hidden layers [4]. Since the forward and backward paths of each layer must both process the entire input sequence before being concatenated and passed to the next layer, multiple bidirectional layers are not conducive to parallel processing. To gain the contextual benefits of a BRNN without sacrificing a parallel model, [4] chose to use a BRNN as the first layer and use unidirectional RNNs for the remaining layers. Google's implementation of the bi-directional first layer is seen in the lower left of **Figure 2**.

2.5 Parallelism for Efficient Computations

In response to great strides in highly-parallel, commercial graphics hardware, many publications were released in an attempt to utilize parallelization and distribution in machine learning [27].

Parallelism is inherently difficult to employ on multiple modules of a single input stream simultaneously [27] [28]. RNN modules depend on output from previous copies, and must be completed in a sequential order. Thread pooling is an effective (albeit obvious) tool for parallelizing independent computations of individual perceptrons within a layered neural net, but fails to significantly improve computational speed in narrow, deep nets. *DistBelief,* developed in 2012, used parallelization to divide wide and deep nets across machines, spatially segmenting groups of perceptrons. This method paved the way for many other attempts at what has come to be known as *model parallelism,* which has generally considered to be a less useful approach than *data parallelism (*described below), as training instances must be processed concurrently, and propagating synaptic data between layers often causes heavy network overhead.

While other groups began the foray into single machine parallelization of neural nets, [27] produced the canonical probe into *distributed* parallelization for neural nets. Two famous models were proposed here: Downpour SGD and Sandblaster L-BFGS. The former used sharded data to distribute the training load across model replicas, updating a parameter server with the resultant change and the models with the new parameters upon absorption of each training instance. Sandblaster L-BFGS used a "coordinator" module to systematically update active models using central storage of their desiccated parameters during training. Both models were observed to increase speed substantially on narrow, deep networks, but the SGD model (especially when coupled with Adagrad adaptive learning) was observed to gain accuracy especially quickly. This model established conventions which permeate the recent history of parallelization attempts on RNNs.

Grid structure [29] demonstrates a distributed system of RNNs over a "globus model" grid. This grid allows applications (RNNs in this instance) to execute on computational nodes overseen by middleware using a broker architectural pattern. These varied RNNs act as "specialists" which communicate with one another in order to collaboratively annotate complex input.

The authors outline four kinds of application level collaboration:

1. Simple division of labor: Predictive task is compartmentalized and nodes collaborate on the construction of a large annotated resource
2. Supervisor: Overseeing node(s) vet the funneled work of subordinate annotators, optionally adding further encoding.
3. Peers with different theoretical models learn/predict in parallel in order to contrast the differences between the models and the extent to which the response of one may be derived from the other.
4. Humans may interact with a program in parallel to predictive annotators, humans may critique a series of structural hypotheses in real time in order to augment the training process.

Parallelism is inherently difficult to employ on multiple modules of a single input stream simultaneously [30] [27] [31]. RNN modules depend on output from previous copies, and must be completed in a sequential order. Thread pooling is an effective (albeit obvious) tool for parallelizing independent computations of individual perceptrons within a layered neural net, but fails to significantly improve computational speed in narrow, deep nets.

DistBelief, developed in 2012, used parallelization to divide wide and deep nets across machines, spatially segmenting groups of perceptrons.

A consistent challenge in parallelization has been memory limitations of GPUs [27]. Many solutions have been proposed, including asynchronous gradient calculations on network distributed GPU farms and architectural accommodations to make NN modules preconditioned for parallelization [27]. CNNs superficially mitigate memory limitations by growing synaptic size linearly with input, but have a limited ability to dynamically express LSTM and generally produce lower performance. Advances in GPU technology and development of specialized hardware may resolve the issue entirely within the next couple of decades.

Though not unique to RNNs, training can be performed in parallel by issuing working copies of from some iteratively changing source of truth [32]. Some master algorithm is responsible for dispatching input instances to slave models (possibly on different machines), updating the master model based on averaged results, then propagating averaged updates to slaves. This may be performed on individual instances, or with shuffled mini-batches of some predefined size, the latter being especially useful to efficiently enqueue work on a network distributed system.

Unsurprisingly, the mini-batch technique is greatly aided by a "warm start" strategy in order to encourage convergence. The RNN model is trained using the entire training data for one epoch, then subsets of the training set are delegated to slave clones to be batch processed. Overlapping data between these slices was found to aid performance for the same reason.

On a related note, parallelism can be especially performant when applied to a hierarchical classification model (especially relevant for RNNs). Output is divided into superclasses and classes of a prespecified number. Thus, rather than considering the entire span of possible classifications, a high-level categorization may influence considerations of downstream modules. Obviously, using this technique (if applicable to the model) can lend itself to clean division/distribution of a RNNs run classifying different steps of the hierarchy on different machines.

2.6 Segmentation Approaches

Achieving open vocabulary with a neural machine translation model presents problems when the model is introduced to out-of-vocabulary (OOV) words, i.e. words that are unknown or do not exist in the recognition vocabulary. These are typically words such as locations, names, or numbers that contain crucial information to the success of many translation and speech recognition tasks. Since most neural machine translation (NMT) models are closed-vocabulary (i.e. they only recognize words in a fixed finite vocabulary), such systems fail in identifying OOV words and in turn affect the translation accuracy. It has been shown that the performance of an NMT model worsens dramatically as the number of words that do not exist in the recognition vocabulary increase [18]. As a result, the study of NMT models that recognize and properly handle OOV words is an active area of research.

In the matter of translation, there are two approaches that deal with OOV words: (1) copy from input to output, or (2) use sub-word units [4]. The following further investigates the latter approach by listing examples of methods that are used to solve the OOV words problem.

One example of a sub-unit word approach that can handle issues arising from word-level processing is character segmentation, in which sentences are fed to the model as sequences of characters. With a word model, there exists no perfect word segmentation algorithm for any language, and it has been shown that such models must implement a suboptimal learning based segmentation algorithm [33]. A character-level translation model offers a way to avoid issues from many morphological variants being output as distinct entities in a vocabulary [34].

An alternative approach is to combine a character and word model, resulting in a hybrid model that utilizes the strengths of both architectures. The amalgamation of the two creates a system that translates mostly at the word level, but uses character segmentation for rare words [34]. This type of model might be easier and faster to train than character-based models and will not output "unknown" tokens, a disadvantage of word models [35].

The final technique in the sub-word unit approach is to implement a wordpiece model, in which sentences are segmented into small sequences of characters. For example,

"The house was sold" ⇒ [_The, _hou, se, _wa, s, _so, ld]

This method depends only on the data and learns word units in a greedy manner to maximize the likelihood on the language model training set [36]. This model allows for user-specified number of word units that does not focus on semantics [4]. Correspondingly, wordpieces do not generate OOVs and can build a 200k word piece inventory [36].

3 Google Neural Machine Translation (GNMT)

Here we highlight some training parameters and performance of Google's neural machine translation model, which is increasingly replacing phrase-based SMT of Google Translate [2]. Although we focus on the latest results achieved by Google, it is important to note that the success of GNMT depended upon recent breakthroughs in NMT by other researchers, notably Graves' use of LSTM networks to analyze sequences with long-term dependencies [21], Sutskever's application of LSTM networks to the problem of machine translation [22], and Bahdanau's invention of the attention module [18].

3.1 Parameters and Training of GNMT

Google trained their translation system using a combination of maximum likelihood (ML) and reward learning (RL) training objectives [4]. ML training focuses on maximizing the sum of log probabilities (**Equation 11**). However, ML training alone was found to be unreliable due to a lack of robustness to inaccuracies in decoded sentences. To remedy this problem, Google refined their model with a reward learning objective, seen in **Equation 12**, initially used in a speech to text application developed by [37]. This function rewarded translations with high BLEU scores.

$$\mathcal{O}_{ML}(\theta) = \sum_{i=1}^{N} \log P_\theta\left(Y^{*(i)} \mid X^{(i)}\right) \tag{11}$$

$$\mathcal{O}_{RL}(\theta) = \sum_{i=1}^{N} \sum_{Y \in \mathcal{Y}} P_\theta(Y \mid X^{(i)})\, r(Y, Y^{*(i)}) \tag{12}$$

To reduce time training, Google trained 12 parallel copies of their network, all of which shared parameters [4]. All trainable shared parameters were initialized to random values in the range [-0.04, +0.04].

There were multiple stages in Google's training procedure. The first stage trained on an ML objective until convergence [4]. Each step of this training consisted of a mini-batch with 128 example sentences. For the first 60K steps, training was performed with Adam, a stochastic optimization algorithm that features adaptive learning rates with relatively small memory use [38]. The learning rate for this was 0.0002. The model was then switched to stochastic gradient descent with a learning rate of 0.5.

After convergence, the model was augmented by incorporating the reward learning function in addition to the machine learning function [4]. The specific combination of these was found by optimizing a linear combination of both ML and RL objectives as seen below.

$$\mathcal{O}_{\text{Mixed}}(\theta) = \alpha * \mathcal{O}_{ML}(\theta) + \mathcal{O}_{RL}(\theta) \tag{13}$$

The optimal value of α for this specific model was found to be 0.25 [4].

3.2 Performance of GNMT

The word meaning test (WMT)‘14 English-to-French, WMT En→Fr (36 million sentence pairs), and English-to-German, WMT En→De (five million), datasets [39] were the primary datasets used to train the En→Fr and En→De neural machine translation models, respectively. These datasets were collected from a publicly available corpora that is used across the board for evaluating NMT systems. Newstest2012 and newstest2013, together, were used as validation sets. Newstest2014 and Google’s own translation product corpora was used as the test sets.

Model	BLEU	Decoding time per sentence (s)
Word	37.90	0.2226
Character	38.01	1.0530
WPM-8K	38.27	0.1919
WPM-16K	37.60	0.1874
WPM-32K	38.95	0.1146
Mixed Word/Character	38.39	0.2774
PBMT [15]	37.0	
LSTM (6 layers) [30]	31.5	
LSTM (6 layers + PosUnk) [30]	33.1	
Deep-Att [43]	37.7	
Deep-Att + PosUnk [43]	39.2	

Table 1: Model results on WMT En→Fr gathered from newstest2014 dataset [4]. BLEU scores as described in Section 1.1 are multiplied by 100 to compute “BLEU points”. This table provides a summary of Google’s results on the WMT En→Fr dataset (newstest2014). Note that their wordpiece model, with a shared size source and target vocabulary of 32,000 (WPM-32K), obtained the highest average BLEU score (38.95). Applying the same type of model for the WMT En→De dataset, WPM-32K also received the highest average BLEU score (24.61) [4].

The scores between the WMT En→Fr and WMT En→De datasets are significantly different as a result of the difference in training sizes for each NMT model. Improvements on the models described above were achieved by using RL-refined models and model ensemble for both BLEU and human evaluation. **Figure 3** provides a particularly succinct visualization of the gains made by GNMT as measured next to human-created translations. Also noteworthy is the exponential leap in NMT quality between Sutskever in 2014 [22] and GNMT in late 2016 [4], largely due to smart architectural choices and use of hardware.

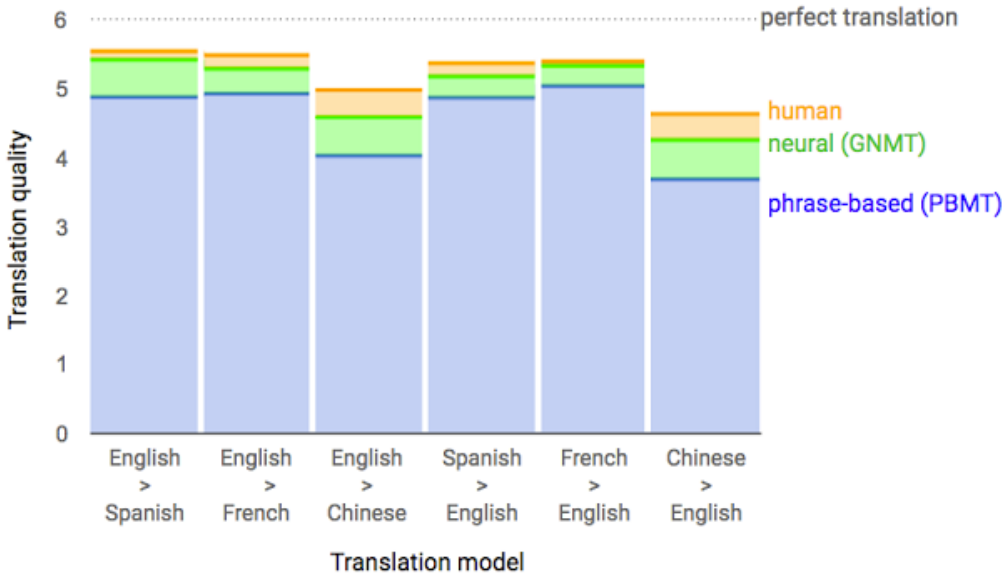


Figure 3: Translation quality of GNMT translations compared to phrase-based translations and human-created translations for six different language pairs. Figure is from [40]. As described in [40]: "Data from side-by-side evaluations, where human raters compare the quality of translations for a given source sentence. Scores range from 0 to 6, with 0 meaning 'completely nonsense translation', and 6 meaning 'perfect translation.'"

4 Conclusion

In this report, we provided a summary of the state of machine translation before the explosive entry of neural machine translation, we summarized important features and architectural considerations of modern neural translation models, and we took a brief look at the training and performance of Google's neural machine translation implementation. The next big step in NMT efforts might be the creation of a single model capable of translating between any of numerous languages, instead of separately trained models for each language pair [41].

Works Cited

[1] G. Lewis-Kraus, "The Great A.I. Awakening," *The New York Times Magazine,* 14 12 2016.

[2] Google Cloud Platform, "Language Support," Google Developers, 2017. [Online]. Available: https://cloud.google.com/translate/docs/languages.

[3] K. Papineni, S. Roukos, T. Ward and W. Zhu, "BLEU: a Method for Automatic Evaluation of Machine Translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 200

[4] Y. Wu, M. Schuster and Z. Chen et al., "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," *Retrieved May 05, 2017, from https://arxiv.org/abs/1609.08144,* 2016.

[5] "MT History," 2006. [Online]. Available: https://www.inf.ed.ac.uk/teaching/courses/icl/lectures/2006/smt.pdf.

[6] A. Lopez, "Statistical machine translation," *ACM Computing Surveys,* vol. 40, no. 3, 2008.

[7] P. Koehn, F. Och and D. Marcu, "Statistical phrase-based translation," in *Proceedings of the 2003 Conference o the North American Chapter of the Association for Computational Linguistics on Human Language Technolog* 2003.

[8] F. Och and H. Ney, "A Systematic Comparison of Various Statistical Alignment Models," *Computational Linguistics,* vol. 29, no. 1, 2003.

[9] Automatic Language Processing Advisory Committee, "Language and Machines: Computers in Translation an Linguistics," 1966.

[10] T. Nishida and S. Doshita, "Machine translation: Japanese perspectives," in *Translating and the Computer 7*, 1986.

[11] "MT System," 2013. [Online]. Available: http://aamt.info/english/mtsys.htm.

[12] National Institute of Standards and Technology, "NIST 2005 Machine Translation Evaluation Official Results," 2005. [Online]. Available: http://www.itl.nist.gov/iad/mig//tests/mt/2005/doc/mt05eval_official_results_release_20050801_v3.html.

[13] H. Jaeger, "Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach (GMD Report 159)," German National Research Center for Information Technology, 2002.

[14] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and times series," in *The handbook of brain theory and neural networks*, MIT Press, 1998.

[15] Y. Bengio, P. Simard and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult.," *IEEE Transactions on Neural Networks,* vol. 5, no. 2, 1994.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation,* vol. 9, no. 8, 1997.

[17] F. A. Gers, J. Schmidhuber and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Computation,* vol. 12, no. 10, 2000.

[18] D. Bahdanau, K. Cho and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate, in *International Conference on Learning Representations*, 2016.

[19] TensorFlow, "Vector Representations of Words," 2017. [Online]. Available: https://www.tensorflow.org/tutorials/word2vec.

[20] S. Ruder, "On word embeddings - Part I," 2016. [Online]. Available: http://sebastianruder.com/word-embeddings-1/index.html.

[21] A. Graves, "Generating Sequences With Recurrent Neural Networks," *Retrieved from: https://arxiv.org/abs/1308.0850,* 2014.

[22] I. Sutskever, O. Vinyals and Q. Le, "Sequence to Sequence Learning with Neural Networks," *Retrieved from: https://arxiv.org/pdf/1409.3215.pdf,* 2014.

[23] C. Olah, "Understanding LSTM Networks," 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[24] S. Hochreiter, Y. Bengio, P. Frasconi and J. Schmidhuber, "Gradient Flow in Recurrent Nets: The Difficulty of Learning Long Term Dependencies," in *A Field Guide to Dynamical Recurrent Networks*, 2001.

[25] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conferen on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[26] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processin* vol. 45, no. 11, 1997.

[27] J. Dean, G. Corrado, R. Monga and e. al, "Large Scale Distributed Deep Networks," *Retrieved from: https://static.googleusercontent.com/media/research.google.com/en//archive/large_deep_networks_nips2012.p* 2012.

[28] L. Bottou, "Stochastic gradient learning in neural networks," in *Proceedings of Neuro-Nîmes 91*, 1991.

[29] N. Kalchbrenner, I. Danihelka and A. Graves, "Grid Long Short-Term Memory," *Retrieved from: https://arxiv.org/pdf/1507.01526v1.pdf,* 2015.

[30] L. Bottou, "Stochastic gradient learning in neural networks," in *Proceedings of Neuro-Nîmes*, 1991.

[31] Y. LeCun, L. Bottou, G. Orr and K. Klaus-Robert Müller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade*, 1998.

[32] Z. Huang, G. Zweig, M. Levit, B. Dumoulin, B. Oguz and S. Chang, "Accelerating recurrent neural network training via two stage classes and parallelization.," in *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, 2013.

[33] K. Cho, B. Merrienboer, D. Bahdanau and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches," in *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structu in Statistical Translation*, 2014.

[34] Y. Kim, Y. Jernite, D. Sontag and A. Rush, "Character-Aware Neural Language Models," *Retrieved May 7, 2017, from https://arxiv.org/pdf/1508.06615.pdf,* 2015.

[35] M. Luong and C. Manning, "Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistic (Volume 1: Long Papers)*, 2016.

[36] M. Schuster and K. Nakajima, "Japanese and Korean voice search.," in *2012 IEEE International Conference o Acoustics, Speech and Signal Processing (ICASSP)*, 2012.

[37] M. Ranzato, S. Chopra, M. Auli and W. Zaremba, "Sequence Level Training with Recurrent Neural Networks," in *2016 International Conference on Learning Representations*, 2016.

[38] D. Kingma and J. Ba, "ADAM: A Method for Stochastic Optimization," in *International Conference on Learni Representations*, 2015.

[39] ACL 2014 NINTH WORKSHOP ON STATISTICAL MACHINE TRANSLATION, "Shared Task: Machine Translation," 2014. [Online]. Available: http://www.statmt.org/wmt14/translation-task.html.

[40] Q. Le and M. Schuster, "A Neural Network for Machine Translation, at Production Scale," Google Research Blog, 2016. [Online]. Available: https://research.googleblog.com/2016/09/a-neural-network-for-machine.html.

[41] M. Johnson, M. Schuster and Q. Le, "Google's Multilingual Neural Machine Translation System: Enabling Zer Shot Translation," *Retrieved from: https://arxiv.org/abs/1611.04558,* 2016.

[42] C. Olah and S. Carter, "Attention and Augmented Recurrent Neural Networks," 2017. [Online]. Available: http://distill.pub/2016/augmented-rnns/.

[43] K. Greff, R. Srivastava, J. Koutnik, B. Steunebrink and J. Schmidhuber, "LSTM: A Search Space Odyssey," *Retreived from: https://arxiv.org/pdf/1503.04069.pdf,* 2015.

[44] J. Koutnik, G. Klaus, F. Gomez and J. Schmidhuber, "A Clockwork RNN," in *Proceedings of the 31 st International Conference on Machine Learning*, 2014.