Using Neural Networks for Translation Tasks

Fang Chan

1. INTRODUCTION

Software translation from one language to another has experienced significant advances in recent years. The opportunities that are provided by quality, error free translation are enormous. These potential opportunities for machine translation range from making books written in English available to German speaking customers to providing mission critical software systems, such as medical systems, available to any doctor in the world.

Because of the advantages machine translation would provide, exploration of different translation models continue. Likewise, the predominant industry paradigms are not yet error free (Goldberg, 2015). As such, machine translation is an incredibly exciting area of research with far reaching consequences.

Other important areas of research for machine translation abound. Are we interested in translating spoken word? Written text? A document? Additionally, a language may have words with multiple meanings, or may literally mean something in another language that does not have the required context (such as a colloquialism). Adequate machine translation needs to account for all of these pitfalls.

There are several ways to write a machine translation program. One could, obviously, write a program with guidelines provided by a bilingual speaker. Unfortunately, the domain would be immense. It would be impossible to write software that would take into account every possible rule of translation for each word, each context of a word, each context of a phrase, etc.

One of the more successful methods from recent years for machine translation has been the use of learning or training models. That is, "teaching" a computer to correctly recognize proper translation. These methods have advanced machine translation greatly, and they will be the focus for discussion.

This paper will discuss the current dominant paradigms of machine translation, including statistical machine translation and the recurrent neural network encoder-decoder approach. There are advantages and disadvantages to both. Additionally, we will summarize and discuss recent work proposed by Kyunghyun Cho (2015) regarding a potential advancement in the use of the RNN Encoder-Decoder approach.

2. Issues with current paradigms of Machine translation

Statistical Machine Translation

Revisiting our example of a potential machine translator from earlier, what could we do differently? Well, one common technique is to program a machine to take some text in one language and the same text in another language so it can "learn" which words map to other words.

The statistical aspect comes in with mapping words to other words. Using Bayesian calculations, probabilities are assigned to word pairings. Why do we use statistics? The answer is because a sentence in one language doesn't have a single translation in another language. It can be translated into multiple different words or sentences. As such, we use probabilities to represent the likelihood that a sentence in one language is the likely translation of the sentence in the original language (Brown, 1990).

The likelihood that an output word adequately represents the same word from the original language is referred to as the "log likelihood". Once we've provided the statistical machine translator with a set of input and output translations, the machine can start to make inferences about novel data and provide us with probabilities that reflect the log likelihood that the outputs are accurately translated (Brown, 2003).

There are several drawbacks to this type of system, however.
First, errors are difficult to discover and fix. Related to this problem, SMT systems are very good at providing understandable translations that disguise errors during the translation process. Additionally, when word order matters, SMT systems aren't as accurate. SMT translations between similar western languages are much more successful than translating between a western language and Chinese, for example. Finally, it's difficult for SMT to take into account specific contextual clues regarding a metaphor in one language and correctly translate it into an understandable phrase in the output language (Brown, 1990).

3. Neural Machine Translation and the RNN Encoder-Decoder Approach to Machine Translation

Both IBM and Google have utilized neural networks to great success. Google, specifically, has been a pioneer in the use of a neural network to increase the speed and accuracy of language translation (see Mikolov, 2013 and Sutskever, 2014). More recently, natural language processing has been advancing with hybrid neural networks and SMT approaches. Cho has extensively detailed this newer paradigm in a 2014 paper.

Before we can get into neural machine translation, however, we must discuss what a neural network is.

A neural network is, essentially, a computer system that is designed to function similarly to the human brain's system of neurons. Information is stored and computed in nodes and is propagated to other nodes with a system of edges, or connections to other nodes (neurons) (Sutskever et al. 2014). The internal nodes are considered "hidden". These nodes are assigned probability weights based on Gaussian distributions. Input is "forward-propagated" through the hidden layers to produce output that is closer to the expected output (Goldberg, 2015).

At its most basic operation level, there are two visible sets of neurons. One set represents the input neurons and the second set represents the output neurons (Goldberg, 2015). The network itself is considered "hidden", a black box where the system of internal processing neurons manage the data to produce outputs.

The motivation behind developing a system this way is that the machine teaches itself based on the data it is provided. There is no dependency, like with an SMT, on the functions the machine is provided in its programming

A **Recurrent Neural Network**, or RNN, is a system that maintains an internal, hidden state. This is in contrast with a typical feed-forward neural network which computes the system's state from scratch at every node. RNNs utilize information sequentially which is why they make for good machine translation architectures. Later neuron processing is dependent on what previous neurons have calculated allowing for inputs of arbitrary length. This is the main advantage of designing a machine translation system with an RNN - an input can be of variable size in relation to the output. The variable input is compressed to a fixed-length vector for processing (Cho, 2014).

Here is an excellent graphic demonstrating RNN computation. The variable x represents a given time step through the network, s is each hidden state, o is the output passed to later neurons, and U, V and W represent the parameter matrices that are shared with each neuron:

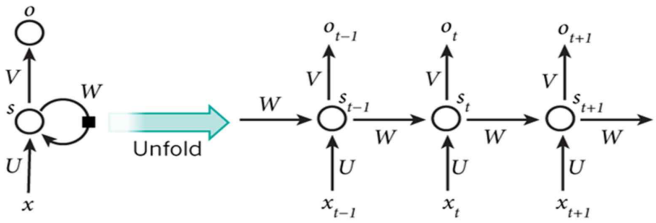

Figure 1: Visual representation of a neural network (LeCun, 2015)

For the purpose of this paper, we have implemented some code in Python demonstrating an RNN. Please assume that this network has already been supplied training data and the data has been compressed into the vector. The x parameter of the run method represents the input sequence vector and returns an output vector that must be decoded into the output language:

```
class recurrent_neural_network:
    def __init__(self, word_dimension, hidden_dimension=100, bptt_truncate=4):
        # Assign instance variables
        self.word_dimension = word_dimension
        self.hidden_dimension = hidden_dimension
        self.bptt_truncate = bptt_truncate
        # With this code we are randomly initializing our U,V and W matrices
        self.U = numpy.random.uniform(-np.sqrt(1./word_dimension), np.sqrt(1./word_dimension), (hidden_dimension, word_dimension))
        self.V = numpy.random.uniform(-np.sqrt(1./hidden_dimension), numpy.sqrt(1./hidden_dimension), (word_dimension, hidden_dimension))
        self.W = numpy.random.uniform(-np.sqrt(1./hidden_dimension), numpy.sqrt(1./hidden_dimension), (hidden_dimension, hidden_dimension))

    def forward_prop(self, x):
        time_steps = len(x)
        states = numpy.zeros((time_steps + 1, self.hidden_dimension))
        states[-1] = numpy.zeros(self.hidden_dimension)
        # Outputs at each time step
        outputs = numpy.zeros((time_steps, self.word_dimension))
        # Iterate through time_steps
        for t in numpy.arange(time_steps):
            states[t] = numpy.tanh(self.U[:,x[t]] + self.W.dot(states[t-1]))
            outputs[t] = numpy.softmax(self.V.dot(s[t]))

        return [outputs, states]

    recurrent_neural_network.forward_prop = forward_prop

    def run(self, x):
        #return highest score
        o,s = self.forward_prop(x)
        return numpy.argmax(o, axis=1)

    recurrent_neural_network.run = run
```

Here is an example of code that encodes input text to a vector that can be supplied to the RNN pictured above:

And finally, here is a method that decodes the prediction values provided by the RNN to actual machine predicted text:

```
def generate_sentence(model):
    # We start the sentence with the start token
    new_sentence = [word_to_index[sentence_start_token]]
    # Repeat until we get an end token
    while not new_sentence[-1] == word_to_index[sentence_end_token]:
        next_word_probs = model.forward_propagation(new_sentence)
        sampled_word = word_to_index[unknown_token]
        # We don't want to sample unknown words
        while sampled_word == word_to_index[unknown_token]:
            samples = numpy.random.multinomial(1, next_word_probs[-1])
            sampled_word = numpy.argmax(samples)
        new_sentence.append(sampled_word)
    sentence_str = [index_to_word[x] for x in new_sentence[1:-1]]
    return sentence_str

num_sentences = 10
sentence_min_length = 7

for i in range(num_sentences):
    sent = []
    # We want long sentences, not sentences with one or two words
    while len(sent) < sentence_min_length:
        sent = generate_sentence(model)
    print " ".join(sent)

model = recurrent_neural_network(vocabulary_size)
```

Cho (2014) has previously proposed a model of machine translation that relies on two recurrent neural networks. This approach, referred to as an RNN Encoder-Decoder model, is described as "[o]ne RNN encodes a sequence of symbols into a fixed length vector representation, and the other decodes the representation into another sequence of symbols" (Cho, 2014).

Similar to the neural networks described above, the encoder and decoder are both "trained" to provide output with the greatest likelihood of an accurate translation from the input. In this proposed model, input and output phrase lengths can differ.

The decoder RNN provides the sequence vector and predicts, probabilistically, what the likely translations are, given the training corpora the system has been provided. Phrase pairs are scored with this probability and the output phrase or sequence is constructed from these likelihood scores (Cho, 2014).

To summarize above Encoder-Decoder model works as depicted in the diagram

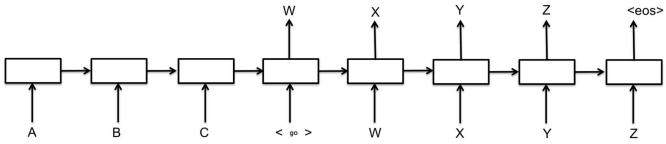

Figure 2: Process of encoder and decoder

Here, "ABC" is the source text and these input units (first 3) comprise the encoder. The following 5 units constitute the decoder for target text ("WXYZ"). RNNs can do the following:

1. Summarize a string into state vector,
2. Predict next word, based on previously processed string (given "AB" able to predict "ABC")
   The code described previously, does the second task. For the encoder-decoder model

we construct and encoder to do the first task, then we train both the encoder-decoder simultaneously to do the translation. The black boxes in the above diagram are GRU cells (Cho, 2014).

For our implementation, we have used the tensorflow library for GRUcell implementation. Next, we create a sequence to sequence model as depicted in the code:

```
def single_cell():
    return tf.contrib.rnn.GRUCell(size)

cell = single_cell()
```

```
def seq2seq_f(encoder_inputs, decoder_inputs, do_decode):
    return tf.contrib.legacy_seq2seq.embedding_attention_seq2seq(
        encoder_inputs,
        decoder_inputs,
        cell,
        num_encoder_symbols=source_vocab_size,
        num_decoder_symbols=target_vocab_size,
        embedding_size=size,
        output_projection=output_projection,
        feed_previous=do_decode,
        dtype=dtype)
```

After training, this architecture is good enough to translate. But we have to train the model for maximum log likelihood (described previously). We use the SGD method for training, see code:

```
# This is the training loop.
    step_time, loss = 0.0, 0.0
    current_step = 0
    previous_losses = []
    while True:
      # Choose a bucket according to data distribution. We pick a random number
      # in [0, 1] and use the corresponding interval in train_buckets_scale.
      random_number_01 = np.random.random_sample()
      bucket_id = min([i for i in xrange(len(train_buckets_scale))
                       if train_buckets_scale[i] > random_number_01])

      # Get a batch and make a step.
      start_time = time.time()
      encoder_inputs, decoder_inputs, target_weights = model.get_batch(
          train_set, bucket_id)
      _, step_loss, _ = model.step(sess, encoder_inputs, decoder_inputs,
                                   target_weights, bucket_id, False)
      step_time += (time.time() - start_time) / FLAGS.steps_per_checkpoint
      loss += step_loss / FLAGS.steps_per_checkpoint
      current_step += 1

# Decrease learning rate if no improvement was seen over last 3 times.
      if len(previous_losses) > 2 and loss > max(previous_losses[-3:]):
        sess.run(model.learning_rate_decay_op)
      previous_losses.append(loss)
```

The step method (in code above), is derived from the tensorflow library to make a complete step comprising of doing a forward pass, calculating losses, back propagating errors and modifying weights.

```
def step(self, session, encoder_inputs, decoder_inputs, target_weights,
         bucket_id, forward_only):
```

To optimize performance of the code, bucketing is used, which we will ignore for now.

There are some significant drawbacks to the RNN Encoder-Decoder approach. Primarily, there is a drop off after sentences reach a length of 20 words. The BLEU scores drop from between 20 and 25 to eventually below 10 (Cho, 2015). Additionally, there can be significant errors at the end of long sentences (Cho, 2015). The current alignment model that focuses on word for word translations result in very poor output sentences for languages with different grammatical structures. For example, English and German can have very different rules about where nouns and verbs should be present in sentences. This model struggles with translation between these languages.

A quick aside: the Bleu score is a metric proposed by Papieni et al. (2002) that seeks to measure how accurately a machine translated a word by comparing it to a translation performed by a human.

Translations are judged by a precision metric which "counts up the number of candidate translation words (unigrams) which occur in any reference translation and then divides by the total number of words in the candidate translation" (Papieni et al. 2002).

Cho (2015) provides an excellent graphic demonstrating the current models of machine translation with SMTs and neural networks:

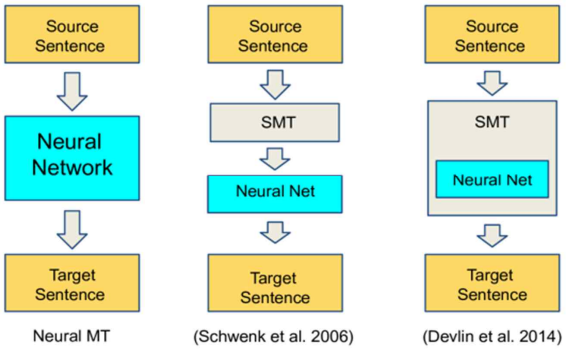

Figure 3: Current models of SMT's and neural networks (Cho 2015)

4. Jointly Learning to Align and Translate

Cho (2015) proposed a potential significant advancement in neural network machine translation over the previously discussed RNN Encoder-Decoder approach. This novel approach, referred to as RNN Encoder-Decoder jointly learning to align and translate, has state-of-the-art advancements that we will discuss at length.

This approach to machine translation depends, much like the earlier proposed system, on two RNNs. Except, the encoder is bidirectional and the decoder is modeled to behave similarly to a search algorithm (Cho, 2015). A context vector is implemented that

depends on "a sequence of annotations… to which an encoder maps the input sentence. Each annotation contains information about the whole input sequence with a strong focus on the parts surrounding the i-th word of the input sequence" (Cho, 2015). The figure to the right is a visual model of the system.

A bidirectional RNN consists of RNN architecture that runs both forward and backward (unlike our code example that only implements forward propagation). Hidden states for both directions are created and concatenated together to form single states. Why would we want to use a bidirectional RNN? Especially in the case of language translation, later words in a sequence are just as useful as the previous words in generating a highly likely output (Goldberg, 2015). The concatenation approach also allows for words to be computed in context, so the words are considered with their immediate neighbors.

Here is a brief summary of the concatenation process:

In this way, the annotation hj contains the summaries of both the preceding words and the following words. Due to the tendency of RNNs to better represent recent inputs, the annotation hj will be focused on the words around xj. This sequence of annotations is used by the decoder and the alignment model later to (Cho, 2015).

The decoder is additionally described as such:

"It should be noted that unlike the existing encoder–decoder approach, here the probability is conditioned on a distinct context vector c(i) for each target word y(i). The context vector c(i) depends on a sequence of annotations (h1, $\cdots$, hTx) to which an encoder maps the input sentence. Each annotation h(i) contains information about the whole input sequence with a strong focus on the parts surrounding the i-th word of the input sequence. The context vector c(i) is, then, computed as a weighted sum of these annotations h1" (Cho, 2015).

The decoder depends on an accurate alignment model that computes a "soft alignment" between the input sequence around a position with the output at a corresponding position. These positions are assessed on how well they match and this score is backpropagated through the network (Cho, 2015). This is precisely how phrases are translated "in context" so that word order isn't as important a factor for the inputs and outputs.

We will discuss the results in greater detail later on, however here we would like to point out some of the advantages the results indicate. First, unlike the RNN Encoder-Decoder approach, the bidirectional approach has no significant drop off in performance after sentence lengths of 20 words. Additionally, the bidirectional approach achieves BLEU scores up to 25% higher than the RNN Encoder-Decoder approach. We also must mention that word and phrase alignment is much better with the proposed architecture. For example, Cho translates from English to French, which can have very different grammatical rules and word alignments. This model handles them superbly. Finally, this implementation will encode into a variable sized vector.

The performance is better for several reasons, including eliminating the fixed vector bottleneck, and removing the need for the RNN to remember the whole sentence as a state in memory (Cho, 2015). The experimental results will prove these hypotheses.

5. Experimental Results

Cho (2015) ran an evaluation experiment to test this idea on an English to French dataset. They compared the performance of the proposed RNN with the previously described RNN Encoder-Decoder architecture. Each model was trained twice with sentences as long as 30 words and sentences as long as 50 words. Here is a graph underscoring the resulting BLEU scores (RNNsearch is the bidirectional jointly learning to align and translate approach):

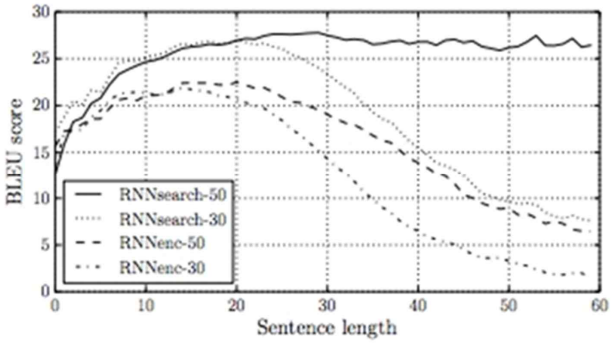

Figure 4: Graph of BLEU score relative to sentence length

The figure above illustrates that performance is generally better for nearly all sentence lengths. Additionally, examining the outputs reveals that words are correctly aligned with their grammatically correct French translations. Cho writes: "we see that the model correctly translates a phrase [European Economic Area] into [zone economique europeen]" (Cho, 2015). The alignment correctly handled this translation by considering the entire phrase at a given location, rather than each individual word.

As was predicted, fixed length vectors create a bottleneck in the translation of long sentences. However, there are other probable reasons why longer sentences have higher BLEU scores. For example, the RNNsearch paradigm "does not require encoding a long sentence into a fixed-length vector perfectly, but only accurately encoding the parts of the input sentence that surround a particular word" (Cho, 2015). The model is able to primarily direct its attention to the information relevant to the production of the next word. All of these aspects work in conjunction to produce excellent log-probability of output translations.

These results would mark a significant leap forward in the development of neural network machine translation. The performance is "comparable to the existing phrase-based statistical machine translation" (Cho, 2015). There is still room for improvement, however. The systems must be able to translate untrained, novel words- so work toward developing a paradigm is greatly needed.

6. Summary

Machine translation has advanced tremendously in the past few years. Novel methods in statistical approaches have driven much of the improvements. However, in recent years paradigms have emerged that rely on neural network architectures. Neural network architectures, in conjunction with the statistical approaches, have led to explosive growth in machine translation and natural language processing in general. This discussion was focused on summarizing the architecture proposed by Cho (2015). A bidirectional RNN with a variable sized vector, alignment model and decoder that computes output based on context, not just word for word. This approach represents a significant potential advancement in the use of neural networks for machine translation.

As was shown, the experimental results are promising. BLEU scores for the proposed architecture were significantly higher than the other RNN approach. Additionally, BLEU scores did not drop off when sentences were longer than 20 words. Finally, the concluding phrases of output sentences remained high quality based on BLEU scores, also an improvement over the RNN encoder-decoder implementation.

If given more time and resources, using the above created encoder-decoders, this research project would have entailed training our RNNs more using proper datasets that were available. In addition, the research would have attempted to fine tune the translation encoder-decoders similar to Cho (2015).

References

Brown, P., J. Cocke, S. Dell, and A. Pietra. "A STATISTICAL APPROACH TO LANGUAGE

TRANSLATION." Computational Linguistics 16.2 (1990): 79-85. Web.
Brown, P., S. Della Pietra, V. Della Pietra, R. Mercer. "The Mathematics of Statistical Machine

Translation: Parameter Estimation." Computational Linguistics 19.2 (1993): 263-311. Web.
Dzmitry Bahdanau, Kyunghyun Cho: "Neural Machine Translation by Jointly Learning to

Align and Translate", 2014; [http://arxiv.org/abs/1409.0473 arXiv:1409.0473].
Ilya Sutskever, Oriol Vinyals: "Sequence to Sequence Learning with Neural Networks", 2014;

[http://arxiv.org/abs/1409.3215 arXiv:1409.3215].
Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares,

Holger Schwenk: "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", 2014; [http://arxiv.org/abs/1406.1078 arXiv:1406.1078].

Kyunghyun Cho, Bahdanau, D.: "Neural Machine Translation by Jointly Learning to Align and

Translate", 2015; [http://arxiv.org/abs/1409.0473 arXiv:1409.0473].
Mikolov, T., V. Le Quoc., I. Sutskever. "Exploiting Similarities among Languages for Machine

Translation."
Papieni, K., S. Roukos, T. Ward, W. Zhu. "BLEU: a Method for Automatic Evaluation of Machine

Translation." Computational Linguistics (2002): 311-318. Web.
Tomas Mikolov, Quoc V. Le: "Exploiting Similarities among Languages for Machine

Translation", 2013; [http://arxiv.org/abs/1309.4168 arXiv:1309.4168].
Yoav Goldberg: "A Primer on Neural Network Models for Natural Language Processing",

2015; [http://arxiv.org/abs/1510.00726 arXiv:1510.00726].
Y. LeCun, Bengio, Y., Geoffrey, H. "Deep Learning." Nature (2015): 436-444. Web.