Henry Moss
ChengXiang Zhai
CS 410 Text Information Systems
02 November 2020

Technology Review:
A general overview of the approach of RNNL as well as its application to ASR/MT systems

Recurrent neural networks are one of the most fascinating techniques involved in machine learning and text information retrieval today. It is widely used in processes such as predicting future text, both in the categories of automatic speech recognition and figuring out the subsequent word to type out will be. Recurrent neural networks can also be used in situations such as predicting future stock prices based off of their history, translating one language to another by focusing on surrounding words and even the whole sentence instead of attempting to translate word by word, or could even be used with pictures to accurately predict the specific content being shown. As the name suggests, recurrent neural networks are built upon regular neural networks, and specialize by observing patterns that stretch wider than the immediate subject at hand.

In order to comprehend the concept of a recurrent neural network, one must understand neural networks themselves. A neural network is a collection of neurons that take an input, map that input through their specific network of nodes, and produce an output on the basis of trial and error. To simplify this even further, a neural network at its fundamental core will take an input, run that input through its system, and produce an output.

Recurrent neural networks are built upon this fundamental concept, both in theory and practice. For the input that the recurrent neural network takes in, it is called “recurrent” because the input takes in the last output that it has generated. This previous output, n, which is now an input, will directly affect the output of n+1, which is what will directly follow n. This is what makes recurrent neural networks perfect for modeling sequential data (Mikolov). In addition to the previous output, most models will also take in another form of input, making the two forms of input being one arbitrary source, and one output from the previous case. While recurrent neural networks have overall been popular in the past, they have received some skepticism after it was shown that vanishing and exploding gradients can cause the traditional gradient descent training algorithms to suffer (Mikolov). Because of this flaw, recurrent neural networks that are only trained by gradient descent based models are very difficult to produce accurate data to its observers.

In the paper *Recurrent neural network based language model* by Mikolov, Karafiat, Burget, Cernocky, and Khudanpur, the scientists create a simple recurrent neural network, which is also commonly known as an Elman Network. As the Elman Network is one of the simplest recurrent neural networks to create, it is likewise simple to train and implement for scientific use and research. This Elman Network uses an input layer x, a context/hidden layer s, and an output layer y. The input in time t is x(t) to the network, the output produced is y(t), and the hidden layer of the network is s(t). Following this, the vector which is received as an input, x(t) is formed by adding onto vector w representing current world, and output from neurons at time t - 1 in the hidden layer s. These layers are then computed in the following formulas:

$$x(t) = w(t) + s(t-1) \tag{1}$$

$$s_j(t) = f\left(\sum_i x_i(t) u_{ji}\right) \tag{2}$$

$$y_k(t) = g\left(\sum_j s_j(t) v_{kj}\right) \tag{3}$$

where $f(z)$ is sigmoid activation function:

$$f(z) = \frac{1}{1+e^{-z}} \tag{4}$$

and $g(z)$ is softmax function:

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}} \tag{5}$$

As a crucial part of the recurrent neural network is using the input from the previous iteration, the formula must start with a manual initialization. In this case, *s*(0) can be set to a vector of small values, such as 0.1. These networks are then often trained in multiple epochs, meaning that they will cycle through the data set several times to begin training. After each of these epochs have completed, the network is tested on validation data, and after 10-20 epochs, the recurrent neural network is often able to achieve convergence (Mikolov). At each step within training, the error vector is calculated in relation to the cross entropy criterion while its weights are updated with the typical backpropagation algorithm: error(t) = desired(t) - y(t). In this case, desired refers to a vector using 1-of-*N* coding representing the word that should have been predicted and *y*(t) is the recurrent neural network's output.

The practice of recurrent neural networks and their toolkits can also be easily appropriated for ASR (automatic speech recognition) and MT (machine translation) by producing lattices. For example, a stereotypical use of recurrent neural networks in an ASR system would first consist of training the network's language models. Next, the model would decode phrases to produce its unique lattices. After that, it would extract n-best lists from the lattices and generate scores based off of the standard n-gram model created at the beginning of the project. Lastly, it would perform linear weighted interpolation and re-rank the list using the new LM scores, being able to produce the n-best results afterwards. Another way to accomplish this goal is to approximate the recurrent neural network model by using an n-gram model. In order to do this, one would first train the recurrent neural network model followed by producing a vast amount of randomly generated sentences from the model. After that, one would need to take the random sentences and build an n-gram model off of them. After this, one would interpolate the approximated n-gram model with the baseline n-gram model and decode the sentence fragment from said model (Mikolov). Luckily with this approach, no RNNLM rescoring code would be necessary to get the system running. Unfortunately though because of this, it would generate large amounts of random sentences and additional memory complexity because of it.

An even more advanced usage of recurrent neural networks can be created called LSTM (Long Short-Term Memory). This process involves adding multiple neural networks on top of the pre-existing recurrent neural network discussed before. LSTM filters out information, using additive and multiplicative gates, to both select and ignore specific data, in order to grow its learning capabilities. After filtering its possibilities, the neural network decides to keep some and forget others, making the recurrent neural network provide the best output for itself, which will in turn become the input in the next iteration (Hochreiter).

In conclusion, the recurrent neural network model has many applications today in deep learning and broader artificial intelligence. The researchers at the Brno University of Technology, Czech Republic found in their experiments that the recurrent neural networks significantly outperformed state of the art backoff models throughout. Specifically, in their tests on the 100-best lists from DARPA WSJ'92 and WSJ'93 data sets, "word error rate reduction is around 18% for models trained on the same amount of data, and 12% when backoff model is trained on 5 times more data than RNN model" (Mikolov). Recurrent neural networks are a significant improvement upon the base of neural networks with the simple concept of re-feeding the result as a parameter into the calculation of the next item in line. It is also groundbreaking because recurrent neural network models connect language modeling closer to broader concepts as machine learning, cognitive sciences, and even data compression.

Works Cited

Hochreiter, Sepp. “Long Short-Term Memory.” *Neural Computation*, Dec. 1997.

Mikolov, Tomas, et al. “Recurrent Neural Network Based Language Model.” *INTERSPEECH 2010*, Sept. 2010, pp. 1045–1048.

Mikolov, Tomas, et al. “RNNLM - Recurrent Neural Network Language Modeling Toolkit.” *IEEE Automatic Speech Recognition and Understanding Workshop*, Dec. 2011.