Recurrent Neural Networks

Priyadarshi Guha

ABSTRACT

Recurrent Neural Networks (RNNs) have been very successful in the field of Natural Language Processing (NLP) because of their ability to remember information from the prior inputs and outputs and to use the same information to produce further outputs. RNNs are networks that work in loops allowing the same previous information to be used to produce new information and so on. Due to this ability RNNs are more suited to work with sequential data such as text, audio, video etc. LSTMs, which is a special RNN architecture, is being used for Google's Neural Machine Translation System which uses a Encoder-Decoder architecture model knows a s the seq2seq model. This paper explores the basics of Artificial Neural Networks, the working of RNNs, a brief introduction to the LSTM and GRU models, the Encoder-Decoder architecture of a seq2seq model, and presents the results of an Attention Based English to Hindi NMT Model.

1. Introduction

A neural network is a computer algorithm based on the structure of the neural network of the human brain. It tries to closely mimic the operations of the human brain. Neural networks learn to recognize patterns, classify data and other similar operations by training on provided data. A neural network is made to take information an input, process, and return the desired output. A neural has three layers namely the input, hidden and the output layer. Input and output operations are handled by the input and output layers. The hidden layer is responsible for the processing of input data and producing the output. Neural networks can be used for a variety of tasks including image recognition, speech recognition, classification, text prediction and machine translation.

2. Recurrent Neural Networks (RNNs)

Traditional neural networks generate output based on the current input alone. They don't have the ability to use previous information. The output generated at every step is independent of the input and output of the previous steps. This makes the traditional neural networks unusable for sequential data where current information is influenced by past information. This is where recurrent neural networks shine.

Recurrent neural networks are a type of neural networks that feed the output of the previous timestep along with the input of the current timestep to generate the output for the current timestep. Recurrent neural networks were based on David Rumelhart's work in 1986. RNNs have a memory which stores information about the inputs and outputs of the previous timesteps. The hidden state of the RNN is responsible for remembering previous information. The first output is generated based on the first set of inputs the initial hidden state which is set to random. This output becomes the previous state during the next timestep and is fed to the network along with the next set of inputs and so on. In RNNs, all the internal layers have their own weights, which are initially set to random during the training process. At each timestep the generated output is compared to the actual output to calculate the errors and to update the internal weights accordingly which trains the network. This process is repeated for several times to minimize the error as much as possible.

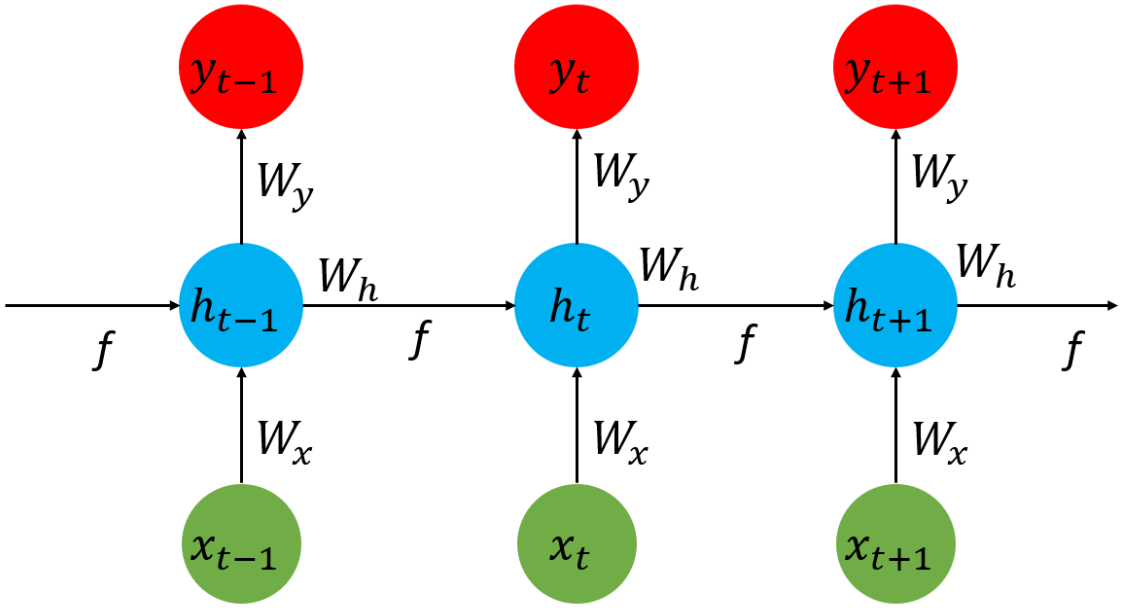

Figure 1: Working of RNN.

The current hidden state, h_t is a function of the previous hidden state, h_{t-1} and the current input, x_t :

$$h_t = f(h_{t-1}, x_t)$$

Where *f* is called the activation function. tanh is one of the most common activation functions. The current inputs and the previous hidden state are fed to the functions through linear combination, where they are multiplied by their respective weights and the then added:

$$h_t = tanh(W_h h_{t-1} + W_x x_t)$$

Using this hidden state, the output at each timestep can be calculated as:

$$y_t = W_y\, h_t$$

3. Long Short Term Memory (LSTM)

LSTM was proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber. LSTM networks are a type of RNN that are capable using past information more efficiently than RNNs. LSTM uses different gates to determine which parts of the information fed to the network is worth keeping and which parts to throw away. This allows the network to remember information for longer periods of time whereas RNNs are only capable of remembering recent information. LSTM models have an additional internal cell state which stores information from each timestep during the training of the model. LSTM has the ability to add or remove information from the cell state using gates. These gates are composed of a sigmoid layer and a point-wise matrix operation.

LSTM has three gates, the forget gate(*f*), the input gate(*i*) and the output gate(*o*).

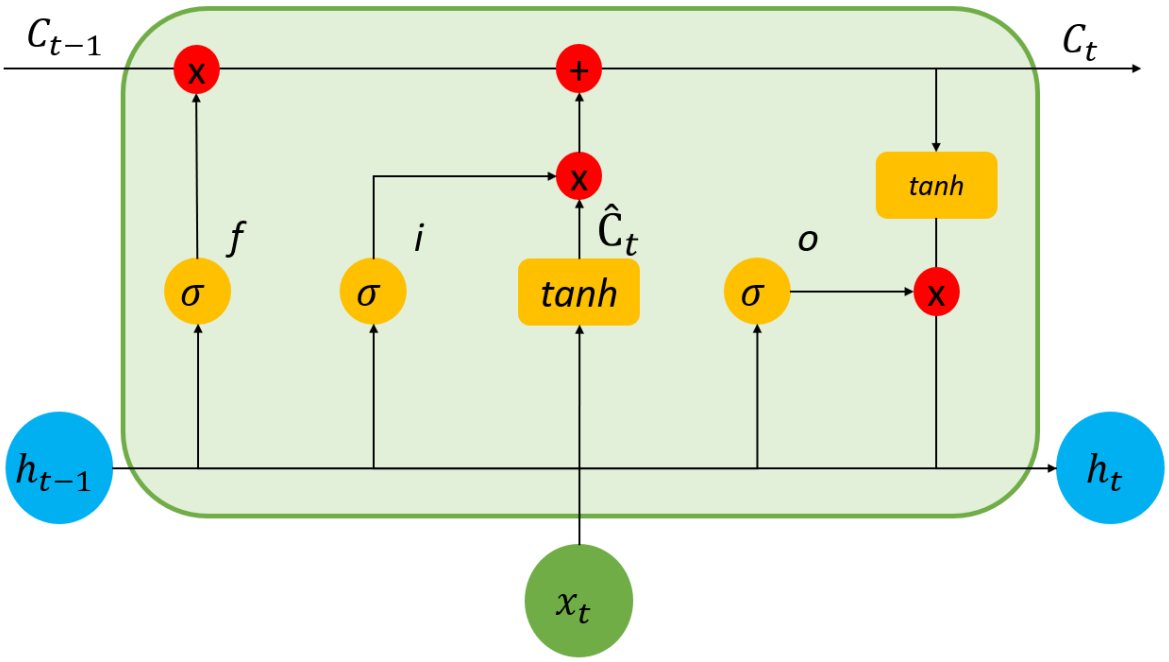

Figure 2: LSTM cell.

The Forget Gate decides what information to throw away from the cell state based on the information from the previous hidden state and the current inputs.

$$f = \sigma\left(W_{fh}h_{t-1} + W_{fx}x_t\right)$$

The Input Gate decides what new information will be stored in the cell state based on the information from the previous hidden state and the current inputs.

$$i = \sigma(W_{ih}h_{t-1} + W_{ix}x_t)$$

A new candidate cell state, $\hat{C}_t$ is created by passing the previous hidden state and the current inputs through a tanh layer.

$$\hat{C}_t = tanh(W_{ch}h_{t-1} + W_{cx}x_t)$$

The current cell state is calculated by removing information that are no longer need from the previous cell state using the forget gate and the adding new information from the candidate cell state using the input gate.

$$C_t = f \times C_{t-1} + i \times \hat{C}_t$$

The current hidden state is a filtered version of the cell state. The cell state is put through a tanh layer, and then the output gate decides which part of the cell state will comprise of the hidden state based on the information from the previous hidden state and the current inputs.

$$o = \sigma(W_{oh}h_{t-1} + W_{ox}x_t)$$

$$h_t = o \times tanh(C_t)$$

4. Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU) is a type of RNN that works just like a LSTM network but has fewer parameters and it lacks an output gate. The effectiveness of GRU lands in between the traditional RNN and the LSTM. GRU aims to solve the vanishing gradient problem.

GRU uses two only gates, the update gate(z) and the reset gate(z).

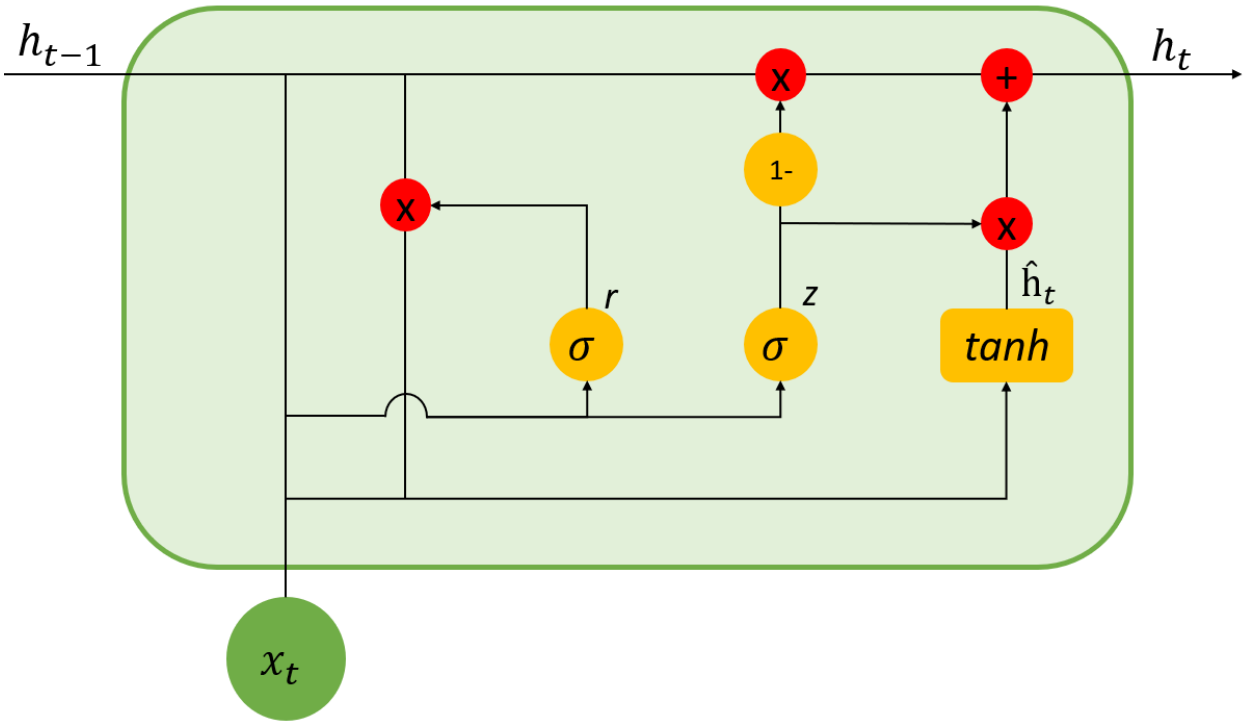

Figure 3: GRU cell.

The update gate helps the model to determine how much of the past information from previous time steps needs to be passed along to the future. It decides what to keep

$$z = \sigma(W_{zh} h_{t-1} + W_{zx} x_t)$$

The reset gate is used to decide how much of the past information to forget. It decides what to throw away

$$r = \sigma(W_{rh} h_{t-1} + W_{rx} x_t)$$

The candidate hidden state is calculated.

$$\hat{\mathrm{h}}_t = \tanh(r \times W_{hh} h_{t-1} + W_{hx} x_t)$$

The new hidden state is calculated.

$$h_t = (1 - z) \times h_{t-1} + z \times \hat{\mathrm{h}}_t$$

5. seq2seq Model

A seq2seq model is a language model that was first introduced by Google for machine translation. seq2seq model makes use of RNNs. LSTM is used in the version proposed by Google. Two RNN layers work together as the main two components of the seq2seq model, i.e. the Encoder and the Decoder. The Encoder converts the input sequence into corresponding hidden vectors, and the Decoder takes as input the hidden vector generated by encoder, its own hidden states and current input word to produce the next hidden vector and finally predict the next word.

5.1. Encoder

The Encoder reads a sequence of k length in k timesteps. Each word of the sentence is the current input, x_i for the model at each time step t_i. The last state vectors, h_k remember what sequence the network has read. These vectors are also known as the thought vectors. They summarize the whole input sequence in a vector form. y_i represents the probability distribution of the next possible word at each time step t_i. These outputs are discarded.

For example, if the input sequence was ‘How are you?’, then the Encoder would look like this,

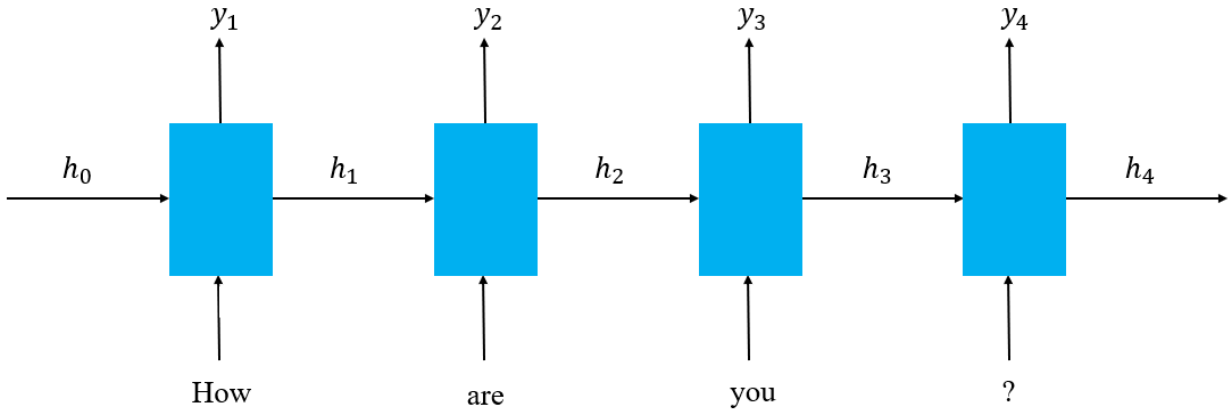

Figure 4: Encoder of seq2seq model.

5.2. Decoder (Training Mode)

The Decoder generates the output sequence word-by-word. During the training of the network, a start token, e.g. <start>, is added to the beginning of the target sequence and an end token, e.g. <end>, is added at the end of the target sequence. The last internal states of the encoder, i.e. h_k are set as the initial states of the Decoder. The start token, i.e. <start>, is given as the first input to the decoder. The first output is generated based on the <start> token and the initial state vectors. In training mode, rather than taking this output as the next input, the next word of the target sequence is taken as the next input for the decoder. This is known as teacher forcing method, where the actual output is taken as input at each time step rather than the predicted output. This helps the model to learn faster.

For example, if we consider the previous example and let the target language be Hindi, then the target sequence would be ‘कैसे हो तुम?’. The Decoder would look like,

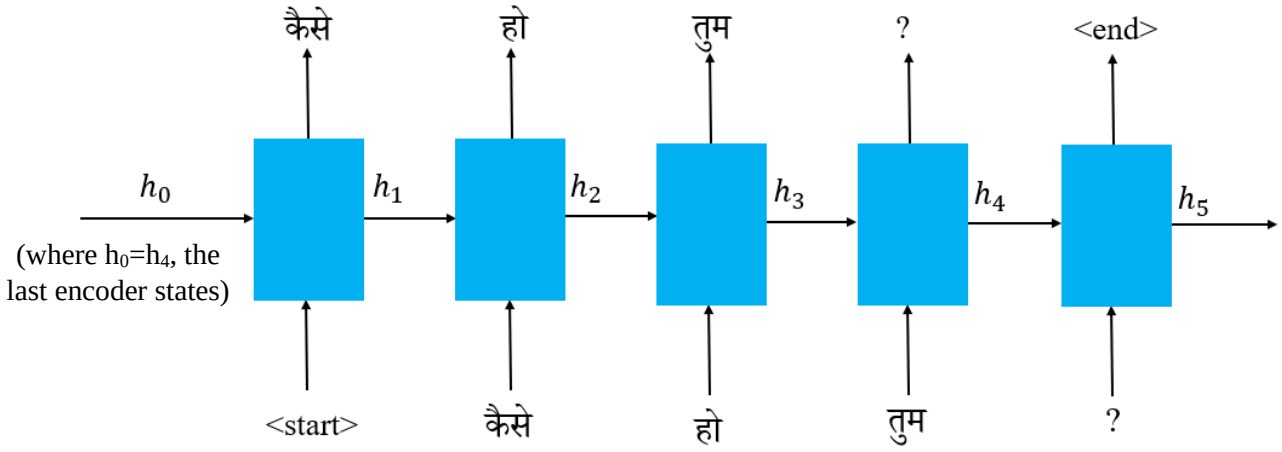

Figure 5: Decoder of seq2seq model in training mode.

5.3. Decoder (Testing Mode)

Unlike the Encoder, the function of the Decoder during training and testing phase are different. In training mode, the actual output of one timestep is fed as the input for the next timestep, but in testing mode, the predicted output at one timestep is fed as the input for the next time step. During the testing phase, the Decoder will keep generating output word-by-word until the end token, i.e. <end>, is generated.

5.4. Attention in seq2seq model

Attention mechanism tries to teach the model to only pay attention to the neighboring words not the whole text. To achieve this feature, we use a new interface between the encoder and the decoder, called context vector, which represents the amount of attention given to words. In this paper we will look into the Bahdanau's Additive Style Attention*. Here are the equations that are implemented:

$$score\left(h_t, \bar{h}_s\right) = v_a^T \tanh(W_1 h_t + W_2 \bar{h}_s)$$

$$\alpha_{ts} = softmax\left(score\left(h_t, \bar{h}_s\right)\right)$$

$$c_t = \sum_s \alpha_{ts}\, \bar{h}_s$$

*Neural Machine Translation by Jointly Learning to Align and Translate, By Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio , 2015

6. Implementation of Seq2seq Model with Attention

This section presents the results of an attention based seq2seq model using single GRU layers for encoder and decoder trained to translate English sentences to Hindi with a word-based vocabulary.

Datasets

The model is trained on a English-Hindi dataset containing nearly 11k sentence pairs in the format:

how are you	तुम कैसे हो
what time is it	अभी कितने बजे हैं

The complete dataset is in lower case without any punctuation. The English-Hindi dataset has been collected from a variety of existing sources, like manythings.org/anki/, OPUS and TDIL, and are from a variety of domain such as agriculture, entertainment, health, tourism and news.

Some of the characteristics of the dataset are:

Number of English-Hindi Pairs	-	11152
English Vocabulary Size (Unique Words)	-	15363
Hindi Vocabulary Size (Unique Words)	-	18545
Maximum English Sequence Length	-	496
Maximum Hindi Sequence Length	-	397

For the training purpose, a start token ‘<s>’ and an end token ‘<e>’ has been added to the beginning and end of the Hindi sentences respectively. Unique words for both the languages are being mapped to integers for vectorization of data. The sequences are then converted to vector form using their corresponding mapped integers and the 0 padded up to the maximum sequence length for both the languages. The dataset is then converted into a TensorFlow Dataset divided into batches of size 64.

Encoder-Decoder Model Architecture

This section describes the architecture of the Encoder-Decoder Model. Both the encoder and the decoder consist of an Embedding layer with a embedding size of 256 and a GRU layer with 1024 units. The decoder contains four additional Dense layers for attention and output.

Summary for the Encoder Model:

Layer (type)	Output Shape	Param #
embedding (Embedding)	multiple	28160
gru (GRU)	multiple	3935232

Total params: 3,963,392
Trainable params: 3,963,392
Non-trainable params: 0

Summary for Decoder Model:

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	multiple	40448
gru_1 (GRU)	multiple	7080960
dense_1 (Dense)	multiple	161950
dense_1 (Dense)	multiple	1049600
dense_2 (Dense)	multiple	1049600
dense_3 (Dense)	multiple	1025

Total params: 9,383,583
Trainable params: 9,383,583
Non-trainable params: 0

Results

For a test run and to get an idea of how the model performs, the model is trained on 100 English-Hindi pairs for 100 epochs using the sparse_categorical_crossentropy loss function and TensorFlow's AdamOptimizer(). It takes about 4-6 secs per epoch to train the network on a i5-8250U CPU. Even after training for such a short time on only 100 samples, some words were still correctly translated, for example, 'welcome' got translated to 'स्वागत', 'legs' got translated to 'पैर', 'who knows' got translated to 'किसे पता'.

7. Conclusion

This paper describes the implementation of a simple attention based NMT model using a single GRU layer each as an encoder and a decoder. The results show the ability of this simple NMT model using GRU layers to translate from English to Hindi, and suggests that it should do well, provided the model has been trained on a bigger dataset for a considerable amount of time.

8. References

[1] Recurrent Neural Networks, Mahendran Venkatachalam, https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce

[2] The Unreasonable Effectiveness of Recurrent Neural Networks, Andrej Karapathy Blog, 2015, http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[3] Understanding LSTM Networks, colah's Blog, 2015, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[4] Understanding GRU Networks, Simeon Kostadinov, 2017, https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

[5] Illustrated Guide to LSTM's and GRU's: A step by step explanation, Michael Nguyen, 2018, https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[6] seq2seq with Attention and Beam Search, Guillaume Genthial Blog, 2017, https://guillaumegenthial.github.io/sequence-to-sequence.html

[7] Neural Machine Translation with Attention, https://www.tensorflow.org/beta/tutorials/text/nmt_with_attention

[8] Effective Approaches to Attention-based Neural Machine Translation, Minh-Thang Luong, Hieu Pham, Christopher D. Manning, 2015, https://arxiv.org/pdf/1508.04025.pdf

[9] Neural Machine Translation by Jointly Learning to Align and Translate, Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio , 2015, https://arxiv.org/pdf/1409.0473.pdf