A Thesis
entitled
Sequence-to-Sequence Learning using Deep Learning for Optical Character Recognition
(OCR)
by
Vishal Vijayshankar Mishra
Submitted to the Graduate Faculty as partial fulfillment of the requirements for the
Master of Science Degree in
Engineering

Dr. Devinder Kaur, Committee Chair

Dr. Kevin Xu, Committee Member

Dr. Ahmad Javaid, Committee Member

Dr. Amanda Bryant-Friedrich, Dean
College of Graduate Studies

The University of Toledo

December 2017

Copyright 2017, Vishal Vijayshankar Mishra

This document is copyrighted material. Under copyright law, no parts of this document may be reproduced without the expressed permission of the author.

An Abstract of
Sequence-to-Sequence Learning using Deep Learning for Optical Character Recognition
(OCR)

by

Vishal Vijayshankar Mishra

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the
Master of Science Degree in
Engineering

The University of Toledo

December 2017

In this thesis, the deep learning techniques called Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) are used to address the problem of Optical Character Recognition (OCR). A special case of RNN called Long Short-Term Memory (LSTM) is used in this research to process the data sequentially. OCR is a process to convert the images containing characters into text. In this research, the images of the mathematical equations from Image-to-Latex 100K data set obtained from OPENAI organization is being used. The mathematical equations from the images are converted into Latex representation using deep learning techniques. The Latex texts were used to again recreate the mathematical equation to test the accuracy of the technique. Unlike previous techniques (Like INFTY) where models were fed with non-tokenized data, the proposed method used the tokenized data to be fed sequentially to the deep learning neural network. The sequential process helps the algorithms to keep track of the processed data and yield high accuracy.

In this research, a new variant of LSTM called “LSTM with peephole connections” and Stochastic “Hard” Attention model were used. The performance of the

proposed deep learning neural network, LSTM with peephole connections” and Stochastic “Hard” Attention model is compared with INFTY (which uses no RNN) and WYGIWYS (which uses RNN). It has been found that the proposed algorithm gives a better accuracy of 76% as compared of 74% achieved by WYGIWYS.

I dedicate this work to my MOM and DAD.

Acknowledgements

I would like to take this opportunity to give my reverence to Almighty Pandit Shri Ram Sharma and Bandhaniya Mataji for giving me the courage, strength, and ability to carry out this thesis work successfully.

I take an immense pleasure to address the people without whom I wouldn't have completed my thesis with ease.

First and Foremost, from the bottom of my heart I would like to give my gratitude to Dr. Devinder Kaur for her incessant, pensive, meticulous and rigorous guidance and inspiration.

I am very grateful to Dr. Henry Ledgard for his unremitting belief, support and encouragement.

I also take an honor to thank Dr. Ahmad Javaid and Dr. Kevin Xu, for being too supportive and ready to be on the committee to assess my work.

It will be unjust if I don't acknowledge the EECS department for buttressing me financially and mentally.

Last but not the least, I would like to show admiration to my parents and my siblings for their affection and never-ending love in all my endeavors. Friends are always special and indeed very inspiring, So I am very thankful to God for giving me such a great people as a friend.

Table of Contents

Abstract	iii
Acknowledgements	v
Table of Contents	vi
List of Tables	vii
List of Figures	viii
List of Abbreviations	x
1 Introduction	1
1.1 Convolutional Neural Network	3
1.2 Recurrent Neural Network	5
1.2.1 Long Term-Short Memory	7
1.2.2 Attention Model	8
2 Architecture of the Convolutional Neural Network	11
2.1 Convolution Layer	12
2.2 Rectified Linear Unit (Optional)(relu)	14
2.3 Pooling Layer (Pool)	15
2.4 Fully Connected Layer (FC)	16
3 Recurrent Neural Network	17
3.1 Introduction to RNN	17
3.2 Architecture of RNN	18
3.3 Concept of Back-Propagation	20
3.4 Example of RNN	25

	3.5 Problems in RNN.....	28
4	Long Term-Short Memory.....	30
	4.1 Introduction to LSTM.....	30
	4.2 Architecture of LSTM.....	31
	4.3 Illustration of Working of LSTM	35
	4.3.1 Forget Gate (f_t)	35
	4.3.2 Input Gate (i_t)	36
	4.3.3 New Cell State (C_t)	37
	4.3.4 Output Gate (O_t).....	38
	4.4 Attention Model.....	40
	4.4.1 Overview	40
	4.4.2 Architecture of the Attention Model.....	41
	4.4.3 Working of the Attention Model.....	42
	4.4.4 Comparison of the Attention Model	44
	4.4.5 Overview of Attention Model with CNN and LSTM.....	49
5	Proposed Model and Data Preprocessing	50
	5.1 Details of the Convolutional Neural Network	52
	5.2 Proposed LSTM unit with peephole connection	53
	5.3 Advantage of Stochastic Hard Attention over Soft Attention	55
	5.4 Details of the Encoder and Decoder	56
6	Results.....	60
7	Conclusion.....	65
	References.....	68

List of Tables

5.1	Details of CNN Layers.....	52
5.2	Advantage of Stochastic Hard attention over Soft attention.	55
6.1	Experiment Results on Image-to-Latex dataset	61

List of Figures

1-1	Thesis Overview	2
1-2	Convolutional Neural Network Architecture.....	4
1-3	Pooling Operation	5
1-4	Unfolded Recurrent Neural Network.....	6
1-5	Long Term-Short Memory (LSTM)	7
1-6	Internal Components description of LSTM	8
1-7	Mathematical Formula is converted in Latex form.....	10
2-1	Structure of Convolutional Neural Network.....	11
2-2	Convolution Operation on Image Pixels.....	13
2-3	Rectified Linear Unit	14
2-4	Pooling Layer Operation.....	15
3-1	Sigmoid Function.....	19
3-2	RNN Architecture	20
3-3	Three Layer Neural Network	21
3-4	RNN Model After First Iteration	26
3-5	Optimized RNN Model with Back Propagation	28
4-1	Block Diagram of LSTM.....	32
4-2	Tanh Function	34
4-3	Forgot Gate (f_t).....	36

4-4	Input gate (i_t)	37
4-5	New memory gate (C_t)	38
4-6	Output gate (O_t)	39
4-7	Architecture of the Attention Model	43
4-8	Attention Model with CNN and LSTM	49
5-1	Actual mathematical image on A4 size page.	50
5-2	Processed Image.....	51
5-3	LSTM with Peephole Connections	54
5-4	Proposed Architecture Model (CNN and LSTM).....	58
6-1	Test Result.	61

List of Abbreviations

OCR	Optical Character Recognition
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM.....	Long Term-Short Memory
WYSIWYG.....	What You See Is What You Get

Chapter 1

Introduction

In modern times, data records in the form of printed paper, consisting of passport documents, invoices, bank statements, printouts of static-data, or any appropriate documentation are being stored in the form of digital copies. It is a common practice to digitize printed texts so that it can be edited, searched and stored, and can be used for text mining electronically. Optical Character Recognition (OCR) can be used to convert printed texts into a digital representation. In the 1900s, an early form of optical character recognition (OCR) was used in the technologies such as telegraphy and reading device for blind people. In 1914, Emanuel Goldberg invented a device that could read characters and translate them into standard telegraphic code [1]. In general, OCR is used to identify and read a natural language from an image and convert it into standard representation. In 1967, the research work of Anderson R.H, there has been a surge in interest for extracting patterns from images for representing them in markup form, which is a correct semantic representation of the images [2].

In the early 2000s, Andrew Kae and Erick Miller addressed the OCR problem in an efficient way with the computational power that existed during that time [3]. However, with the advancement in computational power both in hardware and software, a great

deal of research interest has emerged in OCR. The availability of graphical processing units(GPUs) in hardware and the development of pattern recognition algorithms based on deep learning have given a thrust to the new algorithms of OCR based on convolutional neural networks(CNNs) and recurrent neural networks(RNNs) etc [4].

Deep Learning is a part of a big family of machine learning methods suitable for high dimensional data such as images, text, speech etc. Deep learning uses artificial neural networks that contain several hidden layers. One main principle of deep learning is that it uses raw features as inputs. The deep networks use a cascade of layers of neurons with non-linear activation function. The non-linear activation functions are used to provide a non-linearity in the network, which will help networks to perform feature extraction and transformation from the input. Each successive layer in the deep network uses the output from the previous layer as input and feeds forward it to next layer. For example, A simple explanation for a deep network is that it takes a high-dimensional input such as images, videos, speech etc. and perform non-linear activation function to extract features from an image and send it to next layer for further processing.

An overview of this thesis has been illustrated in figure 1.1.

Figure 1.1. Thesis overview.

Various deep learning algorithms have been used to solve complex problems, such as face recognition, facial expression, edge detection and many more. The deep learning techniques that have been used in this thesis are as follow:

1.1. Convolutional Neural Network(CNN)

1.2. Recurrent Neural Network(RNN)

1.2.1. Long Short-term Memory (LSTM)

1.2.2. Attention Model

1.1 Convolutional Neural Network:

Convolutional Neural Networks (CNNs) fall under the purview of deep learning. They are specifically used for high dimensional data processing such as colored images, videos etc. CNNs are multilayer feedforward networks. Each neuron in the convolution layers performs a dot product of image pixels with a filter. Each convolution layer is followed by Rectified Linear Unit (ReLu) layer and pooling layer. ReLu is a nonlinear activation function which is used to perform a transformation on the images. The dimensionality of the image is being reduced as the computation moves forward in successive layers and it is achieved by pooling layer. The output of the pooling layer becomes the input for the next convolution layer. An illustration of a CNN is shown in Figure 1.2.

Figure 1.2 Convolutional Neural Network Architecture.

For an instance, in Figure 1.2 an image of 32×32 size with 20 filters each of size 5×5 are used to extract features, which will produce 20 activation feature maps and it is forwarded to pooling layer. Then with a filter size of 5×5 in pooling layer and a stride of 1 pixel, the image reduces to 28×28 . This reduced image is then forwarded to the convolution layer and pooling which will reduce the image to 14×14 and so on, till the image is reduced to dimension 1×1 . CNN architectures are completely relied on four hyper-parameters such as **Filters**, **Pooling**, **Stride**, and **Padding** to give the optimal results.

- Convolutional layers have **filters** which are used to extract features from the input images. Filters (for example filters of size 5×5 as shown in Figure 1.2) are moved across the whole image with strides (for example stride of size 1 pixel, 2 pixels) and produce a feature map. If you set a stride to 1 pixel, the filter will move 1 pixel at a time to cover the whole image.
- **Pooling** is used to reduce the dimensionality of an image. It determines the highest value among the input pixels in the filter window. For example, if an image of size $32 \times 32 \times 3$ and pooling with

filter size 2×2 and stride is 2 then the resulting image size will be reduced to $16 \times 16 \times 3$. Figure 1.3 shows an illustration of the pooling operation.

Figure 1.3 Pooling operation

- **Padding** is used to deal with the edges of the images. Sometimes it is convenient to pad the input image with zeros around the boundary. It helps to retain the spatial size of an input at output layer.

1.2 Recurrent Neural Network:

A recurrent neural network (RNN) is a part of the artificial neural network where its current input (x_t) and its previous hidden state (s_{t-1}) and weight (W) is used to predict its next output (s_{t+1}). RNNs are recurrent because they perform the same operation on every input of a sequence. RNNs give good results, when the inputs are sequential such as text, speech etc. RNNs can be best described as the neural networks with memory states. Figure 1.4 shows an unfolded RNN and it is a fully connected network [5].

Figure 1.4 An Unfolded recurrent neural network

This unfolded network shown in Figure 1.4 can be used to solve a sequence problem. For example, “The color of the sky is ____”, so to predict the “blue” we can use the RNN. The sequence contains 5 words in a sentence, so the network will be unfolded into 5-layer neural network, each neural layer for one word. The following steps describe the features of the Figure1.4.

- U and V are the activation functions and W is a weight vector. U , V , and W are same for all the layers.
- x_t is an input to the RNN, At time t . For example, x_1 could be an input vector corresponding to the second word (for example “color”) in a sentence.
- s_t is a hidden state, At time step t . It is the memory state of the network. Calculation of s_t is based on the current input and the input from the previous hidden state. The formula to calculate s_t is given below.

$$S_t = f(x_t * U + W * s_{t-1})$$

- f is a nonlinear function such as **Rectified Linear unit (ReLU)**, **tanh**.
 s_{t-1} is initialized to zero for the first hidden state.

- At time t , o_t is the output state.

$$o_t = \text{sotmax}(V * s_t)$$

1.2.1 Long Short-Term Memory (LSTM):

Long Short-Term Memory (LSTM) is a special kind of RNN [6]. LSTMs addresses the limitation of RNN which is called “long-term dependencies”. For example, if there is a sequence of words in a sentence say, “Birds are flying in the sky”, so here the RNN will obviously can predict the last word as a “sky”. However, if it comes to a long sentence like, “I was born in America... Therefore, I speak fluent English”. Now here, if RNN wants to predict “English” word in the sentence it should have a notion of what has been done until now, which obviously RNN doesn’t remember because of an inadequate amount of memory. This issue is called Long-term dependencies. LSTM was devised to solve this issue by introducing an explicit memory unit called cell in the network.

LSTM was introduced by Hochreiter and Schmidhuber in 1997 and it was refined by other people who started using it. Unlike RNNs, LSTMs are made to remember information for long periods of time. A brief illustration of LSTM model is shown in Figure 1.5 [7].

Figure 1.5. Long Short-Term Memory [7]

Figure 1.6. Internal components description of LSTMs.

For instance, let's focus on the general working of the LSTMs and ignore the internals. Each network layer takes three inputs. X_t is the input at time step t . h_{t-1} is the output of its previous LSTM unit and C_{t-1} is the memory of its previous unit and it is the most important unit in LSTMs which makes it different from RNNs. h_t is the outputs of the current layer and C_t is the memory of the current unit.

1.2.2 Attention Model:

In recent time, RNNs have started using attention mechanism. Attention mechanisms in neural networks are somewhat related to human's visual attention mechanism. For a long time, human visual attention mechanism has been studied profoundly and there exist many models to give an authenticity to it. The basic concept of this model is that focusses on a certain region of an image with high resolution and fade the surrounding region of an image and subsequently adjust the focal point over the period [8].

There is a long history be of attention model, certainly in image recognition. For example, image captioning by Karpathy [9]. However, recently in recurrent neural

networks, attention mechanisms have been used extensively. The core part of my thesis model is a Stochastic “hard” attention model.

In this thesis, a new variant of LSTM unit called “**LSTM with peephole connections**” and a **Stochastic “Hard” Attention** mechanism based encoder- decoder model for Image-to-Latex 100K data set [5] [10]. At present this is the best machine translation system we have. This model comprises of multi-layered convolution neural network to obtain the features of an image with the attention-based recurrent neural network. In our case, we introduce one more layer of the multi-row recurrent neural network called LSTM with peephole connection in front of attention model, so that it should addresses the OCR problem.

In 2003 Fukuda and Tamari invent a system that takes in handwritten mathematical expression and converts it into TeX format [11]. However, the focus of this paper is to use an optical character recognition mechanism for an image of mathematical formulas, including Greek symbols, superscript and subscript conversion in markup form [11]. The effectiveness of this system can be measured by the combination of segmented characters with grammars of the underlying mathematical layout language.

In this thesis, we have used a dataset obtained from OPENAI website which contains an image of mathematical formulas. In this experiment, the deep learning techniques named CNN and LSTM with peephole connections have been used to convert the mathematical formulas into Latex representation. A brief illustration of the experiment is shown in Figure 1.7.

$$h_{i}^2 = h^2 \equiv \frac{2}{r},$$

Figure 1.7. A mathematical formula is converted into Latex form

This thesis is organized as follows: Chapter 2 presents a complete explanation of the Convolutional Neural Network (CNN). Chapter 3 gives a nation of the Conventional Recurrent Neural Network (RNN) and talks about its failure. Chapter 4 explains a special case of the RNN called Long-Term Short Memory (LSTM) and its advantage over RNN. Chapter 5 gives an in-depth knowledge of proposed model and data prepossessing method. In Chapter 6, the data-set is applied to the proposed model and the results are compared with the previous work.

Chapter 2

Architecture of the Convolutional Neural Network

This chapter discusses an architecture of a convolutional neural network (CNN). CNN consists of multiple layers and each of these layers of CNN are shown in Figure 2.1 [12].

2.1. Convolution Layer (conv)

2.2. Rectified Linear Unit(optional)(relu)

2.3. Pooling Layer(Pool)

2.4. Fully Connected Layer(FC)

Figure 2.1 Structure of Convolutional Neural Network

2.1 Convolution Layer:

A convolutional layer consists of set of filters. Filters are small in size spatially; however, it covers the depth of an input volume (for example, if filter size is $5*5*3$ i.e 5-pixel width and height and 3 is the depth of the image because of color channel). Filters are used to extract features from an image. In feature extracting process, filters are moved across the image with given strides and perform dot products with the entries of the filter and the input at any position. Each filter is populated with random pixels and there can be multiple filters in each layer (for example, filter size can be $2*2$, $3*3$, $5*5$ and each convolutional layer can have 20, 30 or 60 filters). As the filter is moved across the input volume, it produces a 2-dimensional activation feature map for that filter. For instance, if there are 20 filters of size $3*3$, then there will be 20 activation feature maps for each filter and each feature map shows the responses of the respective filters at every spatial position. So, the input to the next layer would be these activation feature maps (for example, in Figure 2.2 the size of the activation feature map is $4*4$ and if there are 20 such activation feature maps, then the input to the next layer would be $4*4*20$). In Figure 2.2, the image size is $5*5$, filter size is $2*2$, stride of 1 pixel and the activation feature map is $4*4$. The activation feature can be calculated with a formula i.e $\{(W-F)/S+1\}$ where W is the size of the image, F is the filter size and S is the stride size. Calculation for the activation feature map in Figure 2.2 is given by:

$$\begin{aligned} &= (W - F) / S + 1 \\ &= (5 - 2) / 1 + 1 \\ &= (3) / 1 + 1 \end{aligned}$$

$$= 3 + 1$$

$$= 4$$

An output of this convolutional layer will produce 4*4 activation maps.

Figure 2.2 Convolution Operation on Image Pixel

Calculation of first convolution is computed by moving a filter with given stride across the image pixels and perform the dot product to get the activation feature map (for example, first pixel of image is multiplied to the first pixel of the filter (0*1)). Calculation of first convolution as follows [12].

$$\begin{aligned} &(0*1) + (1*-1) + (0*1) + (1*1) \\ &= 0 \end{aligned}$$

2.2 Rectified Linear Unit(ReLU):

ReLU is a non-linear activation function, which is used to apply elementwise non-linearity. ReLU layer applies an activation function to each element, such as the *max*(0, x) thresholding to zero. ReLU is by far the most successful activation function in deep neural network. Figure 2.3 shows the behavior of ReLU function. ReLU finds the negative values out of input and threshold it to zero.

For example: ReLu (2, -3) will out (2,0). Because ReLu thresholds the negative value to zero

Figure 2.3 Rectified Linear Unit

2.3 Pooling Layer:

CNN uses pooling layers for down-sampling. Pooling layers are interleaved in-between successive convolutional layers. It is used to reduce the size of the image so that the number of parameters get reduced and helps to control overfitting. The Pooling layer works on every activation feature maps independently and resizes it spatially, using MAX operation. For example, in Figure 2.4 a pooling layer with filter size 2*2 and stride 2 will reduce the image of size 4* 4 to 2*2 i.e 50% less than the previous size. The **max** operation is used to find the largest number amongst the numbers those who fall into the given filter's window [12].

For example, in Figure 2.4 if the filter size is 2*2 then it will cover the first rows and two columns and it will apply **max** operation as shown below. In Figure 2.4, the final reduced size of an output is shown, i.e 2*2.

$$= \max(2, 1, 0, 3)$$

$$= 3$$

Figure 2.4 Pooling Layer Operation

2.4 Fully Connected Layer(FC):

Fully connected layer is the last layer of the CNNs. This layer takes an input from preceding layer (i.e convolutional layer or pooling layer or relu layer) and outputs an N dimensional vector, where N is the number of classes that algorithm must choose from. For example, in digit classification problem, N would be 10 because there are 10 digits (from 0-9) in our number system. Each number in this N dimensional vector specifies the probability of a certain class. For example, if the outcome of a digit classification problem is [0 .05 .05 .65 .1 .1 0 0 .05 0] vector, which means that there is probability of digit 1 is 0%, digit 2 is 5%, digit 3 is 5%, digit 4 is 65%, digit 5 is 10%, digit 6 is 10%, digit 7 is 0%, digit 8 is 0%, digit 9 is 10% and digit 10 is 0%. So, this vector represents that the given image is 4 because of high probability of the corresponding number in the vector. FC layer basically performs a dot product with the output of the previous layer and the filters and produce the N-dimensional vector which contains the probabilities for the different classes.

Chapter 3

Recurrent Neural Network (RNN)

This chapter explains the RNN architecture in detail.

3.1. Introduction to RNN

3.2. Architecture of the RNN

3.3. Concept of Backpropagation

3.4. Example of RNN

3.5. Problems in RNN

3.1 Introduction to RNN:

RNN is a family of the artificial neural network that processes the data sequentially. RNN was developed in 1980's but recently gained the popularity due to advances in the computational power such as graphics processing units, processors and CPUs. RNN is called a recurrent because it applies same parameters to all the inputs and performs the same task for all the input sequence. RNN has memory units that makes it different from the conventional neural network. Memory units help RNN to remember the previous inputs. Unlike the conventional neural network, RNN uses input from the previous hidden state, to predict the output of the current state.

3.2 Architecture of the RNN:

Figure 3.2 shows the RNN architecture, where each vertical rectangular box is a hidden layer and each layer contains several neurons. RNN comprises of the input layers (X_{t-1} , X_t , X_{t+1}), hidden layers (h_{t-1} , h_t , h_{t+1}), output layers (y_{t-1} , y_t , y_{t+1}) and weight matrices (W , U , V). RNN takes one input at each time step (for example, at time t , the first input X_t is given to the network) and then it is passed to the hidden layer to predict the output. The hidden layers are the important part of the RNN, because they keep the track of the previous work. The hidden layers take inputs from its input unit and from its previous hidden layer to predict its output. The weight matrix of the hidden layer (i.e, W as shown in Figure 3.2) has to be squared, because it helps to keep the same number of inputs as there are outputs. The input layer matrix (i.e U) and output layer matrix (i.e V) do not have to be squared, because it can connect to any random number of inputs to any random number of hidden units. In the beginning, all the weight matrices (i.e W , U , V) are randomly initialized. The first hidden layer h_{t-1} in the Figure 3.1 does not have any previous hidden layer, therefore no contribution from its previous hidden layers to predict its output. The first hidden layer is initialized by the dot product of its current input X_{t-1} at time $t-1$ and weight matrix U . This dot product is passed through the activation function (for example, sigmoid function as shown in the Equation 3.1) to generate the values for the first hidden layer. In general, to process the data, RNN takes an input X_t at time t , multiply it with weight matrix U and pass it to hidden layer h_t , an output of the previous hidden layer h_{t-1} parameterized with weight matrix W is given to the current hidden layer h_t to predict the output y_t . The output y_t is obtained by taking a dot product of the present hidden layer h_t and weight matrix V . This process keeps on going until it covers all the

layers and predict the final output. The RNN is recurrent because it performs the same operation on all the inputs (for example, it uses same weight matrices (W, U, V) for all the inputs, to maintain the integrity of the context of the sentence). Formulas to calculate h_t and y_t are shown in equation (3.1) and (3.2)

Sigmoid Activation Function:

The sigmoid function transforms the input values, which can have any value between plus and minus infinity, into the range between 0 and 1. The Figure 3.1 shows the sigmoid function.

Figure 3.1 Sigmoid Function.

The sigmoid function can be written as: $Y^{sigmoid} = \frac{1}{1+e^{-X}}$

$$X_t = (W^{(h_{t-1})} * h_{t-1} + U * x_t)$$

$$h_t = sigmoid(X_t)$$

$$h_t = \frac{1}{1+e^{-X_t}} \quad (3.1)$$

The softmax function takes a vector of the real valued scores and compresses it to a vector of the values between zero and one that add up to one.

The softmax function is written as: $f_j(Z) = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$

$$y_t (\text{predicted value}) = softmax(V * h_t)$$

$$y_t (\text{predicted value}) = \frac{e^{V * h_{tj}}}{\sum_{k=1}^K e^{V * h_{tk}}} \quad (3.2)$$

Figure 3.2 RNN Architecture.

3.3 Concept of Backpropagation:

To understand the concept of backpropagation, consider the three-layer network shown in figure 3.3. The indices i , j and k refer to the neurons in the input, hidden and output layers respectively [13].

Figure 3.3 Three Layer Neural Network.

The inputs, x_1 , x_2 , and x_3 are forward propagated through the network from left to right and the error signals e_1 , e_2 , and e_3 are back propagated from right to left. A weight matrix W_{ij} represents the connection between the neuron i in the input layer and neuron j in the hidden layer. A weight matrix W_{jk} represents the connection between the neuron j in the hidden layer and the neuron k in the output layer [13].

To calculate the errors occurred while forward propagation in the network, the propagation starts at the output layer and work backwards to the hidden layer. The error at the output of neuron 2 at iteration n is defined by,

$$e_2(n) = y_{t,2}(n) - y_2(n) \quad (3.3)$$

Where $y_{t,2}$ is the target output of neuron 2 and y_2 is the predicted output of the network of neuron 2.

Once we calculated the errors of each neuron at the output layer, we need to update the weight matrix W_{jk} . The neurons in the output layer are provided with the desired output of its own. So, we can use a straightforward procedure to update the weight W_{jk} . The update rule for the weights at the output layer is given by,

$$W_{jk}(n + 1) = W_{jk}(n) + \Delta W_{jk}(n) \quad (3.4)$$

where $\Delta W_{jk}(n)$ is the weight correction.

Weight correction at the Output Layer.

To calculate the weight correction for the neurons, we use the inputs x_i . However, in the multilayer network, the inputs of neurons in the input layer are different than the inputs of the neuron in the output layer. So, to calculate the weight correction, the output of neurons in the hidden layer j which is y_j is used, instead of input x_j . In multilayer network the weight correction is given by,

$$\Delta W_{jk}(n) = \alpha * y_j(n) * \delta_k(n) \quad (3.5)$$

After n iteration, the error gradient at neuron k is given by $\delta_k(n)$ in the output layer.

The error gradient is nothing but the derivative of the activation function multiplied by the error at the neuron output. Thus, in the output layer for neuron k , we have

$$\delta_k(n) = \frac{\partial y_k(n)}{\partial X_k(n)} * e_k(n) \quad (3.6)$$

The output of neuron k at iteration n is $y_k(n)$, and $X_k(n)$ is the net weight input to neuron k at the same iteration. So, the sigmoid function, for Equation 3.6 can be written as

$$y_k(n) = \frac{1}{1 + e^{-X_k(n)}} \quad (3.7)$$

$$\delta_k(n) = \frac{\partial \left\{ \frac{1}{1 + e^{-X_k(n)}} \right\}}{\partial X_k(n)} * e_k(n) \quad (3.8)$$

$$\delta_k(n) = y_k(n) * [1 - y_k(n)] * e_k(n) \quad (3.9)$$

Weight correction at the Hidden Layer.

The weight correction at the hidden layer is given as,

$$\Delta W_{ij}(n) = \alpha * x_i(n) * \delta_j(n) \quad (3.10)$$

Where $\delta_j(n)$ represent the error gradient at neuron j in the hidden layer.

$$\delta_j(n) = y_j(n) * [1 - y_j(n)] * \sum_{k=1}^l \delta_k(n) w_{jk}(n)$$

Where l is the number of neurons in the output layer.

$$y_j(n) = \frac{1}{1 + e^{-X_j(n)}}$$

$$X_j(n) = \sum_{i=1}^m x_i(n) * w_{ij}(n) - \theta_j$$

Where m is the number of input in the input layer and θ_j is a threshold level.

Now, we can show the back-propagation training algorithm in following steps.

Step 1: Initialization:

Initialize all the weights and threshold levels of the network to random numbers.

Step 2: Activation:

To initiate the back- propagation, the neural network is given the inputs $x_1(n)$, $x_2(n)$, $x_3(n)$,....., $x_m(n)$ and it outputs the desired results $y_{t,1}(n)$, $y_{t,2}(n)$, $y_{t,3}(n)$,....., $y_{t,m}(n)$.

- a) Calculate the actual outputs of the neurons in the hidden layer:

$$y_j(n) = \text{sigmoid} \left[\sum_{i=1}^m x_i(n) * w_{ij}(n) - \theta_j \right]$$

Where m is the number of inputs of neuron j in the hidden layer, and sigmoid is the activation function.

- b) Calculate the actual outputs of the neurons in the output layers.

$$y_k(n) = \text{sigmoid} \left[\sum_{j=1}^m x_{jk}(n) * w_{jk}(n) - \theta_j \right]$$

Where m is the number of inputs of neuron k in the output layer.

Step 3: Weight training:

After calculating the outputs at each layer, now we are going to update the weights by using the back-propagation. In the back-propagation network, propagating backwards the errors associated with output neurons.

- a) Calculate the error gradient for the neurons in the output layer.

$$\delta_k(n) = y_k(n) * [1 - y_k(n)] * e_k(n)$$

Where

$$e_k(n) = y_{t,k}(n) - y_k(n)$$

Calculate the weight corrections:

$$\Delta W_{jk}(n) = \alpha * y_j(n) * \delta_k(n)$$

Update the weights of the output neurons:

$$W_{jk}(n+1) = W_{jk}(n) + \Delta W_{jk}(n)$$

b) Calculate the error gradient for the neurons in the hidden layer.

$$\delta_j(n) = y_j(n) * [1 - y_j(n)] * \sum_{k=1}^l \delta_k(n) w_{jk}(n)$$

Calculate the weight corrections:

$$\Delta W_{ij}(n) = \alpha * x_i(n) * \delta_j(n)$$

Update the weights of the neurons in the hidden layer:

$$W_{ij}(n+1) = W_{ij}(n) + \Delta W_{ij}(n)$$

Step 4: Iteration:

Get back to the Step 2, then increase the iteration n by one, and repeat the process until the errors get reduced to minimum [13].

3.4 Example of RNN:

In Figure 3.2, the inputs to the RNN is 4-dimensional vector and the outputs are also 4-dimensional vector. The dimensionality of an output depends on the vocabulary of the network (for example, in this network the vocabularies are “H” “E” “L” “O”). The vocabularies are the unique characters obtained from the training dataset. In this example, the training dataset is HELLO. The network shown in Figure 3.2 implements the forward pass, where the inputs are “H” “E” “L” “L” fed to the RNN and the expected output for corresponding input letters are “E” “L” “L” “O”. The entire flow of the RNN is explained in the following steps.

Training Data = “HELLO”

Input = “H” “E” “L” “L” { [1 0 0 0] [0 1 0 0] [0 0 1 0] [0 0 1 0] }

Output = “E” “L” “L” “O”

3.4 RNN model after the first iteration.

- Step 1: First, all the weight matrices (W , U , V) are initialized to the random values.
- Step 2: The network takes the input training sequence and performs the forward propagation

(for example, input vector $[1 \ 0 \ 0 \ 0]$ corresponding to "H" is forwarded through the first hidden layer). Then the network tries to find the output probabilities for all the classes.

- For instance, in Figure 3.4 the expected out for the first letter "H" is "E" (shown in red color), however the probability of "O" is very high.

- The reason behind this error in the output is, the process of randomly assigning the values to the weight matrices and therefore the probabilities for the output vectors are random.
- Step 3: To correct the probabilities, we must calculate the total error at the output layers.

Formula to calculate the total error is mentioned below.

(Summation over all the outputs).

$$\text{Total Error} = \sum \frac{1}{2} (\text{Target output} - \text{predicted output})^2$$

- Step 4 : Use Backpropagation to calculate the gradients at each layer with respect to all the weight in the network and then use the gradient descent to update the values of the randomly chosen weight matrices, to minimize the error.
- Step 5: Once the weights are optimized, the network is ready to train the inputs again. The result of the trained model is shown in Figure 3.5. Where the probabilities for the corresponding outputs letters are highest.

3.5 Optimized RNN model with back propagation

3.4 Problems in RNN:

Although RNN has an edge over the conventional neural network, it is susceptible to long-term dependencies problem. Long-term dependencies occur during the back-propagation process, the contribution of gradient values diminish gradually as the computation propagates to previous time-steps. Thus, the capability to remember the long sentences decrease. The problem of long-term dependencies is explained with the help of example below.

For example, consider the following two sentences:

Sentence 1.

"Henry walked into the room. Julie walked in too. Julie said hello to ____."

Sentence 2.

"Henry walked into the room. Julie walked in too. It was very late at work, and everyone was walking after a hectic schedule. Julie said hello to _____."

In both the sentences, one can easily comprehend the context and tell the answers to both the blank spots are most likely "Henry". It is essential that RNN would be able to predict "Henry" in both the blank spots in sentence 1 and 2, who was appeared several time steps back. Due to long term dependency problem, it is observed that RNN would give the right answer for the first blank in sentence1 and fails to answer the second blank. This happens because, in back propagation phase, the values of the gradient vanish gradually as they propagate to previous time-steps. Therefore, the likelihood of "Henry" is being recognized in a long sentence is reduced. The major factor behind the long-term dependencies is vanishing gradient or exploding gradient problem [14].

The problem of gradient vanishing is solved by initializing the weight matrix $W^{(ht-1)}$ to an identity matrix, instead of random initialization or it is recommended to use ReLu as an activation function instead of sigmoid function.

Chapter 4

Long-Short-Term-Memory (LSTM)

This chapter discusses about the architecture of the LSTM and the implementation of Attention model.

4.1 Introduction to LSTM

4.2 Architecture of the LSTM

4.3 Illustration of Working of LSTM

4.4 Attention Model

4.1 Introduction to LSTM

In chapter 3, the concept of the RNN and its problem called long-term dependency is explained in detail. In this section, a special case of RNN called Long-Short-Term Memory (LSTM) and its ability to deal with the long-term dependency problem is discussed. LSTM unit helps to overcome the long-term dependency problem of RNN. The main unit of an LSTM network is the memory unit. This memory unit comprises of a cell state and a pair of gate layers as shown in Figure 4.1. The German scientist Hochreiter and Schmidhuber introduced an LSTM model in 1997 [15]. It is designed in such a way that it memorizes the information for long periods. This behavior of LSTM

handles the problem of long-term dependency very efficiently. The potential to memorize the information for longer period helps LSTM to do well in the complex areas of deep learning such as machine translation, speech recognition, sequence-to-sequence analysis, and more. In the following sections, the working of LSTM is explained in detail.

4.2 Architecture of the LSTM

This section focuses on the architecture of the LSTM in detail. Figure 4.1, describes the internal structure of an LSTM unit. The LSTM consists of three main states called cell state (C_{t-1} , C_t), input state (X_t and h_{t-1}) and output state (h_t) and have four gates called forget gate (f_t), input gate (i_t), new memory gate (C_t'), and output gate (O_t) to perform the internal operation [16].

The cell state (C_{t-1} , C_t) is a crucial part of the LSTM (also called a memory unit) that runs through all the LSTM units in the network to transfer the information. This information is modified with the help of gate layers (a systematic workflow of the four gates are explained in 4.2.1). These gates are used to regulate the information and help LSTM to decide what information must be removed and what must be retained. Each LSTM unit has four gates that protect and control the flow of the information of cell state C_{t-1} . These gates are a way to allow correct information flow through one LSTM unit to another.

Each LSTM unit takes three inputs X_t , h_{t-1} , and C_{t-1} and generates one output h_t and a new cell state C_t . The input X_t given to LSTM can be a character, a word, or a speech,

input h_{t-1} which is an input from the previous unit helps to control the flow of the information. If the current unit is the first unit of the LSTM then there is no previous input. In that case, a randomly generated value for h_{t-1} is given to the first unit to compute the functional blocks of *sigmoid* and *tanh*. Once these inputs are processed through the internal gates, then they are used to update the cell state C_{t-1} to C_t and help to predict the output h_t of the current LSTM unit.

Figure 4.1 Block diagram of LSTM.

Components in LSTM:

Vector Transfer

: In Figure 4.1, Each line carries a vector representation of the inputs.

Vector
Multiplication

: This circle produces the pointwise vector multiplication operation.

Vector Addition

: This circle produces the pointwise vector addition operation.

sigmoid

: This is a neural network layer with *sigmoid* activation function.

tanh

: This is a neural network layer with *tanh* activation function.

Concatenate

: The merging lines denote the concatenation.

Copy

: The forking lines denote the contents are being copied and the copies are being forwarded to different locations.

LSTM unit consists of four neural network layers and three of them are using *sigmoid* activation function and one is using *tanh* activation function as shown in Figure 4.1. The sigmoid layer outputs numbers between zero and one that controls the flow of the information through the network. If a value is zero then “**no information will pass**”

and if a value is one then “**complete information will pass**” through the network. In chapter 3, the sigmoid activation function is described in detail. In this section, ***tanh*** activation function is described briefly.

***Tanh* activation function (The hyperbolic tangent):**

The Tanh activation function is shown in Figure 4.2. It restricts a real-valued number to the range [-1,1]. Tanh function is a scaled sigmoid function and its output is zero- centered. Therefore, it is a preferred activation function in practice for deep learning applications.

Figure 4.2 Tanh Function

Mathematical formula of tanh is:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

4.3 Illustration of Working of LSTM:

This subsection focuses on the step-by-step illustration of the LSTM. Each neuron in the LSTM has four gates and they are mentioned below.

4.3.1 Forget gate (f_t)

4.3.2 Input gate (i_t)

4.3.3 New cell state (C_t)

4.3.4 Output gate (O_t)

4.3.1 Forget gate (f_t):

This gate decides what information needs to flow and what must be removed from the cell state(C_{t-1}). To make this decision, LSTM has a *sigmoid* layer called “forget gate layer” as shown in Figure 4.3. This forget gate takes an input from its input state (X_t) and from its previous output state(h_{t-1}) to output a number between 0 and 1 for every value in the cell state C_{t-1} . If an output is 1 then the information is retained completely and if 0 the information is completely removed from the cell state C_{t-1} . A mathematical formula for the forget gate is written as,

$$f_t = \text{sigmoid}(W_f[h_{t-1}, X_t] + b_f)$$

The weight matrix W_f is a square matrix of dimension ($n * n$). Where n is the number of features.

Figure 4.3 Forget gate (f_t)

4.3.2 Input gate (i_t):

In this step, Input gate decides what new information must be stored in the cell state. This step has two processes, which are shown in the Figure 4.4.

- 1) In first step, the inputs X_t and h_{t-1} are being processed through ***sigmoid*** layer to form the input gate (i_t). This input gate decides the values that are being sent forward to the next step. A mathematical formula for the input gate is written as.

$$i_t = \text{sigmoid}(W_i[h_{t-1}, X_t] + b_i)$$

- 2) In second step, the ***tanh*** layer takes X_t and h_{t-1} as an input to create the values for C_t' in the range $[-1,1]$.

$$C_t' = \tanh(W_c[h_{t-1}, X_t] + b_c)$$

Figure 4.4 Input gate (i_t)

4.3.3 New cell state (C_t):

In this step, a new cell state C_t is generated as shown in Figure 4.5. A new cell state is created, by multiplying the old cell state C_{t-1} to forget gate f_t and then the input gate i_t and values of C_t' are added. A mathematical formula for the C_t is given by

$$C_t = f_t * C_{t-1} + i_t * C_t'$$

This new cell state C_t has a required information that is given to the next LSTM unit in the network to predict its output.

Figure 4.5 New memory gate (C_t)

4.3.4 Output gate (O_t):

The final step decides the output of the LSTM unit. This final step has two steps to produce the output which is shown in Figure 4.6.

- 1) In the first step, the information from the current input X_t and from the previous output h_{t-1} are passed through a **sigmoid** layer to decide what parts of the cell state contributes to the output. A mathematical formula is given by

$$O_t = \text{sigmoid}(W_o[h_{t-1}, X_t] + b_o)$$

- 2) In the second step, the cell state values are put through the ***tanh*** layer to scale the values between -1 and 1. Then, the output of the ***tanh*** layer is multiplied to the output of the ***sigmoid*** layer to produce the final output h_t . A mathematical formula is given by

$$h_t = O_t * \tanh(C_t)$$

Figure 4.6 Output gate (O_t)

4.4 Attention Model

4.4.1 Overview

4.4.2 Architecture of the Attention Model

4.4.3 Working of the Attention Model

4.4.4. Comparison of the Attention Model

4.4.5 Attention Model with CNN and LSTM

4.4.1 Overview

This section discusses the attention model mechanism in deep learning application. In recent times, attention model mechanism got a special recognition in the field of neural networks and deep learning architecture particularly in sequence-to-sequence analysis and NLP (natural language processing) [17]. The attention model was introduced in 2014 by Bahdanau et al.s [18]. The attention model mechanism in neural network is closely related to the human's visual recognition pattern mechanism, where the retinas focus on a certain region of an image with "high resolution" and perceive the surrounding of an image with "low resolution" [15]. A similar notion of focusing on a certain region of the input with high resolution has been applied in deep learning.

In this research, the attention model mechanism is applied to the images of the mathematical formulas to generate the latex markup representation. This research is an example of sequence-to-sequence model, where outputs are generated sequentially. In practice, sequence-to-sequence model has two LSTM networks play the role of encoder and decoder and attention model is interleaved between these two LSTM networks. The attention model mechanism gives an ability to reduce the long sentences into the

sequence of words that can be fed to the encoder and decoder to increase the precision of the model [19]. In the following sections, the architecture and working of an attention model is discussed.

4.4.2 Architecture of the Attention Model

In this section, the architecture of the attention model is discussed and its diagram is shown in Figure 4.7. The attention model takes the inputs generated from the encoder ($y_1, y_2, \dots, y_n$, these inputs are also called annotation grid) and the context vector C_t . The attention model produces one output vector Z which can further be divided into $Z_1, Z_2, Z_3, \dots, Z_n$ vectors to pass it to the decoder. The context vector C_t is generated by combining annotation grid and the output of the attention model Z as shown in Figure 4.7.

The inputs to the attention model are divided into small parts (such as $y_1, y_2, \dots, y_n$) so that attention model can focus on each part of an input with high resolution. In general, the attention model outputs the weighted arithmetic mean of the inputs ($y_1, y_2, \dots, y_n$) and the weights are chosen based on the relevance of each input with the context vector C_t . These weights are used by attention model to focus on a certain part of an input. If the weights are high of a certain region the attention model will give more focus on that part of an input [20].

The attention model has internal components that are mentioned below.

tanh

: This is a neural network layer with *tanh* activation function.

SOFTMAX

: This is the neural network with softmax activation function.

: This is a pointwise vector addition operation.

: This is a pointwise vector multiplication operation

Softmax activation function:

The softmax function is an extended form of the logistic function that “compresses” a K-dimensional vector Z of some real values to K-dimensional vector $\sigma(Z)$ of real values in the range $[0,1]$ that add up to 1 [21].

$$\sigma(Z)_j = \frac{e^{Z_j}}{\sum_{k=1}^K e^{Z_k}} \text{ for } j = 1, \dots, K.$$

4.4.3 Working of the Attention Model

In this section, the step-by-step working of the attention model is explained.

Step1: In this step, the inputs $(y_1, y_2, \dots, y_n)$ and context vector C_t are given to the attention model.

Step 2: In this step, the inputs and the context vector are processed through the neural network layers with ***tanh*** activation function and produces the outputs $M_1, M_2, \dots, M_n$. The mathematical formula is given by,

$$M_i = \tanh(W_c * C + W_y * y_i)$$

Where W_c and W_y are the weight matrices with dimension $(n * n)$ where n is the feature.

Step 3: In this step, the outputs $M_1, M_2, \dots, M_n$ are given to the neural network with softmax activation function which generates the outputs $(S_1,$

$S_2, \dots, S_n$) based on the relevance of the variables with the context vector. The mathematical formula is given by,

$$S_i = \exp((W_m M_i))$$

Step 4: This step calculates the output of the attention model. The output Z is the weighted arithmetic mean of the inputs $y_1, y_2, \dots, y_n$, where the weights represent how appropriate is each output with the given input.

Figure 4.7 Architecture of the Attention Model

4.4.4 Comparison of the Attention Model:

In this section, the two variants of the attention-based models are discussed.

- 1) “Soft” Attention Mechanism (Used by WYGIWYS paper).
- 2) Stochastic “Hard” Attention Mechanism (Used in the proposed model)

The actual difference between these two attention models is in the definition of the ϕ function which is described in respective attention model in detail. Here, first discussing their common premise on which the framework is based on. [22]

Convolutional Neural Network (CNN) Features extraction.

The proposed model takes a raw image and generates a Latex representation y encoded as a sequence of 1-of-K encoded latex word.

$$y = \{y_1, \dots, y_c\}, y_i \in R^K$$

Where K is the size of the Latex vocabulary and C is the length of the Latex representation.

In the proposed model, a convolutional neural network is used to extract a set of feature vectors which is referred as an annotation vectors. The length of the generated annotation vector is L, each of which represents a D-dimensionality corresponding to a part of the image.

$$a = \{a_1, \dots, a_L\}, a_i \in RD$$

In this research, the features were obtained from the lower level of the CNN to get the close correspondence between the feature vectors and portions of the 2-D image, which allows the decoder (LSTM) to effectively emphasis on certain parts of an image by selecting a subpart of all the feature vectors.

The long short-term memory (LSTM) [6] network that produces a Latex representation by generating one word at every time step conditioned on a context vector $z_t^\wedge$, the previous hidden state and the previously generated words. The proposed LSTM unit with peephole connection is described in Chapter 5.

$$\begin{pmatrix} i_t \\ f_t \\ O_t \\ C_t \\ , \end{pmatrix} \begin{pmatrix} \text{Sigmoid} \\ \text{Sigmoid} \\ \text{Sigmoid} \\ \text{tanh} \end{pmatrix} = W^T \begin{pmatrix} h_{t-1} \\ z_t^\wedge \end{pmatrix}$$

$$C_t = f_t * C_{t-1} + i_t * C_t'$$

$$h_t = O_t * \tanh(C_t)$$

Now, i_t, f_t, C_t, O_t, h_t are the input, forget, memory, output and hidden state of the LSTM, respectively. The vector $z^\wedge \in R^D$ is the context vector, that takes the visual knowledge associated with a corresponding input location, as explained below. E is an embedding matrix of dimension $R^{m \times K}$. Where m and n denote the dimensionality of embedding and LSTM respectively. Sigmoid activation and $(*)$ element-wise multiplication are used to transform the inputs.

In basic terms, at time t , the relevant part of an image is dynamically represented by context vector $z_t^\wedge$. The mechanism ϕ that calculate $z_t^\wedge$ from the annotation vectors $a_i, i = 1, \dots, L$ represents the features that were extracts from different image locations. At each location i the attention mechanism calculates a positive weight α_i which can be recognized as the probability of that location i which then can be used by attention

mechanism to focus for generating the next word in the sequence. This process is called hard but stochastic attention mechanism or a relative recognition can be given to i^{th} location in the a_i 's together. The positive weight α_i of every annotation vector a_i is generated by a hard attention model f_{att} and to compute that, a multilayer perceptron is conditioned on the hidden state h_{t-1} . The soft attention mechanism was introduced by Bahdanau et al. (2014).

In general, the hidden states of the RNN network changes as the output advances to the next level in the sequence however the next move of the network will rely on the previously generated words in the sequence.

$$e_{ti} = f_{att}(a_i, h_{t-1})$$

$$\alpha_{ti} = \frac{e^{(e_{ti})}}{\sum_{k=1}^L e^{(e_{ti})}}$$

Once the weights (which sum to one) are computed, the context vector $\hat{z}_t$ is computed by

$$\hat{z}_t = \phi(\{a_i\}, \{\alpha_i\})$$

Once the annotation vector and positive weights are generated, the ϕ function returns a single vector called context vector. The details of ϕ function are discussed in respective attention variants.

The initial values of the cell state c_0 and hidden state h_0 of the LSTM are generated by calculating an average of the annotation vectors given into two different MLPs (init,c and init,h):

$$c_0 = f_{init,c} \left(\frac{1}{L} \sum_i^L a_i \right)$$

$$h_0 = f_{init,h} \left(\frac{1}{L} \sum_i^L a_i \right)$$

Deterministic “Soft” Attention

In this section, a mechanism for the deterministic “Soft” attention model f_{att} is discussed: **this mechanism was used in WYGIWYS paper.**

Deterministic “Soft” Attention Model.

The attention model used in WYGIWYS paper was introduced by Bahdanau et al. (2014). In deterministic soft attention model, they take the expectation (E) of the context vector $\hat{z}_t$ directly whereas, in stochastic hard attention model requires sampling of the attention location s_t as shown below,

$$E_{p(s_t|a)}[\hat{z}_t] = \sum_{i=1}^L \alpha_{t,i} a_i$$

A deterministic attention model can be used by calculating a weighted annotation vector $\phi(\{a_i\}, \{\alpha_i\}) = \sum_i^L \alpha_{t,i} a_i$ which was introduced by Bahdanau et al. (2014). This is nothing but to give a positive weight α to the system in order to get the context vector. This deterministic soft attention model is differentiable and can be directly optimized using back propagation by calculating the gradient. So, that means the output of the overall model is calculated by considering the location variable s_t got from the feed forward network and the expected context vector $E[\hat{z}_t]$. In other words, the deterministic

attention model is nothing but the approximation to the integrated likelihood over the locations variable s_t .

Stochastic “Hard” Attention Mechanism

In this research, s_t is used as the location variable. s_t helps model to decide where to focus attention when generating the t^{th} word. $s_{t,i}$ represents the one-hot vector which is set to 1 for the i^{th} location (out of the length L) in order to extract the visual features of i^{th} location. The attention variable s_t can be treated as intermediate dormant variables, and can give a categorical distribution parametrized by $\{\alpha_i\}$, and context vector $z_t^\wedge$ is given by,

$$p(s_{t,i} = 1 | s_{j < t}, a) = \alpha_{t,i}$$

$$z_t^\wedge = \sum_i s_{t,i} a_i$$

Here a new objective function L_s is used to calculate the marginal log likelihood $\log p(y|a)$ of a sequence of observed words y given an image feature a . L_s can be used to optimize the learning parameters W of the models and can be derived as follows,

$$\begin{aligned} L_s &= \sum_s p(s|a) \log p(y|s, a) \\ &\leq \log p(s|a) p(y|s, a) \\ &= \log p(y|a) \end{aligned}$$

In Stochastic hard attention model, gradients are calculated by reinforcement learning method with respect to the model parameters. This can be done by sampling location variable s_t from a categorical distribution.

Hard attention model makes a hard decision at every time step. In $z_t^\wedge = \phi(\{a_i\}, \{\alpha_i\})$ a function (ϕ) returns a sampled annotation vector a_i at every time steps that is based upon a categorical distribution parameterized by α .

4.4.5 Overview of Attention Model with CNN and LSTM:

In this section, a complete architecture of the research is shown. Figure 4.8 is the overall view of the model which is used for this thesis [20].

- 1) An image dataset is passed through the CNN to get the image into vector representation [23].
- 2) The outputs of the CNN are given to the encoder (LSTM) which produces the tokenized form of an input sequence.
- 3) The outputs of the encoder are given to the attention model to produce the highly relevant outputs with respect to the inputs.
- 4) These outputs are then given to decoder to generate markup representation of the inputs.

Figure 4.8 Attention Model with CNN and LSTM.

Chapter 5

Proposed Model and Data Preprocessing

In this chapter, a new variant of LSTM called “**LSTM with peephole connection**” and a “**Stochastic Hard Attention Model**” are discussed along with the data procurement and data preprocessing methods. The data-set for this research work was procured from the OPENAI organization [24]. The size of the data-set was 1.5 gigabytes and it includes a total of 100K images of mathematical formulas [25]. In figure 5.1, the actual image of a mathematical formula on A4 size page is shown. In this data-set all the images are on A4 size paper, so to get the better results all the whitespace was cropped in preprocessing steps. The processed image, that is being used in this research is shown in Figure 5.2.

$$\Psi_E^{(-)} = \begin{pmatrix} e^{i\frac{\theta}{2}} \sum f_n e^{i(n+\frac{1}{2})\theta} \\ e^{-i\frac{\theta}{2}} \sum g_n e^{i(n+\frac{1}{2})\theta} \end{pmatrix}$$

Figure 5.1 Actual mathematical image on A4 size page.

$$\Psi_E^{(-)} = \begin{pmatrix} e^{i\frac{\theta}{2}} \sum f_n e^{i(n+\frac{1}{2})\theta} \\ e^{-i\frac{\theta}{2}} \sum g_n e^{i(n+\frac{1}{2})\theta} \end{pmatrix}$$

Figure 5.2 Processed Image.

Image-to-latex-100k data-set contains 127,652 different mathematical equations along with their rendered pictures in PNG format. The mathematical formulas have been extracted from the Latex sources of papers available on the arXiv website (<https://arxiv.org/>). These latex sources of papers were parsed through the regular expressions in python to obtain the mathematical formulas. In this research, the size of the formulas is restricted in between 35 to 1024 characters. The regular expression generated 963,890 different latex formulas from the latex sources. Amongst these 900K formulas only 300K formulas were chosen to pass through the KaTeX API to render the PDF files and only 100K formulas were used to compare the proposed model with the existing models like WYGIWYS. These PDFs were converted into PNG format and the size of each rendered image was 1654×2339 pixels. To improve the results, rendered images were cropped to 360×60 pixels. Once these images were cropped, they were divided into tokens to train the model and all the large size images with more than 175 tokens have been discarded. The Training batch size is set to 35 tokens because of the size limit of GPU memory.

Implementation Details of the Proposed Model:

In this section, the details of the proposed model are shown in Figure 5.4 and details are also mentioned in Table 5.1.

5.1) Details of the Convolutional Neural Network.

Table 5.1 Details of CNN Layers.

Number of Convolutional Neural Network(CNN)	CNN Parameters	Pooling Layer Parameters
CNN1	Filter size: 3×3 No. of Filters: 512 Stride: 1 Zero-Padding: 0	No pooling
CNN2	Filter size: 3×3 No. of Filters: 512 Stride: 1 Zero-Padding: 1	Pooling for CNN2: Filter size: 1×2 Stride: 1×2 Zero-Padding: 0
CNN3	Filter size: 3×3 No. of Filters: 256 Stride: 1 Zero-Padding: 1	Pooling for CNN2: Filter size: 2×1 Stride: 2×1 Zero-Padding: 0

CNN4	Filter size: 3×3 No. of Filters: 256 Stride: 1 Zero-Padding: 1	No pooling
CNN5	Filter size: 3×3 No. of Filters: 128 Stride: 1 Zero-Padding: 1	Pooling for CNN2: Filter size: 1×1 Stride: 2×2 Zero-Padding: 0
CNN6	Filter size: 3×3 No. of Filters: 64 Stride: 1 Zero-Padding: 1	Pooling for CNN2: Filter size: 2×2 Stride: 2×2 Zero-Padding: 2

5.2) The proposed LSTM with peephole connection.

In this thesis, a new variant of LSTM unit called LSTM with peephole connections is used.

Limitation of conventional LSTM unit.

In comparison with conventional LSTM unit where every gate receives the information from the input units and the outputs of all the previous hidden units, however there is no direct connection from the “Constant Error Carousel (CEC)” unit also called cell state in conventional LSTM unit which is supposed to control [10]. It can be seen that the output of an LSTM unit which will be closed to zero as long as the output gate is

closed [10]. It can be deduced that, if the output gate is closed none of the gates can access to the CEC they control. The resulting lack of essential information may harm the performance of the network and mainly in this research where the past information is very essential.

Peephole connections.

The easiest but very powerful solution to the conventional LSTM unit is to introduce a weighted “peephole” connections from the cell state unit (C_{t-1}) to all the gates in the same memory unit as shown in Figure 5.3 [10]. The peephole connections allow every gate to assess the current cell state even though the output gate is closed and this peephole connections helped the proposed model to surpass the accuracy of the model called WYGIWYS.

Figure 5.3 LSTM with Peephole Connections.

During backpropagation, error signals will not be propagated through gates via peephole connections to the CEC. Peephole connections are same as the regular connections to gate; however, the update procedure is different. In conventional LSTM, the ultimate source of recurrent connections is the output unit, so the way updates in conventional LSTM occur within a layer arbitrarily. On other hand, updates in peephole connections occur within the cell state, or recurrent connections from gates.

Update patterns in peephole LSTM.

Each cell state component must be updated based on the most current activations of peep connection. The peephole connections need two-phase update scheme;

In First phase, when the recurrent connections are made with the gates, the following gates will be activated.

1. Input gate
2. Forget gate
3. Cell state

In Second phase, the output gate and the output of the LSTM unit will be activated [10].

5.3) Advantage of Stochastic Hard attention over Soft attention.

Stochastic Hard Attention	Deterministic Soft Attention.
In stochastic process like Hard attention mechanism, rather than using	A deterministic process like Soft attention is fully differentiable that

all the hidden states y_t as an input for the decoding, the process finds the probabilities of a hidden state with respect to location variable s_t . The gradients are obtained by reinforcement learning.	can be directly attached into an existing system, and in back-propagation the gradients will flow through the attention model while flowing through the network.
Advantage of Hard attention is that it allows a model to pick the high probabilities in a context vector with respect to the vocabulary list.	While Soft attention directly finds the probabilities from annotation matrix map it with context vector which doesn't work well all the time.

5.4) Details of the Encoder and Decoder Layer.

This model consists of two LSTM layers called Encoder and Decoder. The encoder LSTM layer has hidden state of size 128×1 and the decoder LSTM layer has hidden state of size 256×1 . The size of each token is 60 characters long. These tokens are fed into the encoder which is then given to decoder to generate the output [26].

In this model, the gradient descent algorithm is used to learn the parameters (gradient descent algorithm is explained in chapter 3 in detail). This model is trained for 14 epochs and use the validation set to decide the best model and the test set is used to test the model. For this research, I used Keras with theano as a backend and

the AWS (amazon web service) instance with **NVIDIA GRID GPUs** named **g2.2xlarge** were used.

The GPU configurations are listed below.

- 1) Four NVIDIA GRID GPUs, each with 1,536 CUDA cores and 4 GB of video memory and the ability to encode either four real-time HD video streams at 1080p or eight real-time HD video streams at 720P.
- 2) 8 vCPUs.
- 3) 15 GB of memory.
- 4) 60 GB (1 x 120) of solid state drive (SSD) storage.

This AWS GPU instance was designed to run on high-performance CUDA, OpenCL and OpenGL applications. Using AWS instance, it took 32 hours to train the model and 7 days to optimize and test the model.

Figure 5.4 Proposed Architecture Model (CNN and LSTM with peephole connections)

$$\beta = \frac{1}{3} \left| \frac{\pi}{\sqrt{D}} - 2.5 \right|$$

Encoder

Decoder

Size of each hidden state of the LSTM encoder is 128

Size of each hidden state of the LSTM decoder is 256

Chapter 6

Results

In this chapter, the experimental results on Image-to-Latex dataset are discussed. The images of mathematical equations were converted to latex markup representation. Then markup representation was used to construct the images of the equations to check the relevance (refer figure 6.1).

The proposed method is compared with the previous two methods called INFTY and WYGIWYS on the bases of BLEU (Bilingual evaluation understudy) metric and Exact Match [27]. BLEU is a metric to evaluate the quality for the predicted Latex markup representation of the image. Exact Match is the metric which represents the percentage of the images classified correctly.

Table 6.1 summarizes the comparative result of three methods against the BLEU metric and Exact Match metric. It can be seen that the proposed method scores better than the previous methods. The proposed model generated results close to 76% which is the highest in this research area. Previously, the highest result was around 75% achieved by WYGIWYS (What You Get Is What You See) model [28]. The BLEU and Exact Match scores of the proposed model are slightly above the existing model however, this is a significant achievement considering the low GPU resources and small dataset.

Table 6.1. Experiment Results on Image-to-Latex dataset.

Model	Preprocessing	BLEU	Exact Match
INFTY	-	51.20	15.60
WYGIWYS	Tokenize	73.71	74.46
PROPOSED MODEL	Tokenize	75.08	75.87

Actual Test results on Test set:

In this section, the output of the test set is discussed. Each output is divided into three sections as shown in Figure 6.1:

1. **Original Image:** It is the actual input image given to the model.
2. **Predicted Latex:** It is the predicted latex representation of the actual image.
3. **Rendered Image:** It is the rendered image from the predicted latex form.

Original Image: 1

$$\Theta = \log \frac{4N}{\mathcal{M}L}$$

Predicted Latex:

`\Theta = \operatorname{log} \frac{4 N}{\{\{ \cal M \} L \}}`

Rendered Predicted Image:

$$\Theta = \log \frac{4N}{\mathcal{ML}}$$

Original Image: 2

$$\Omega = \left(\begin{array}{c|c} 0 & \mathbf{I} \\ \hline \mathbf{I} & 0 \end{array} \right)$$

Predicted Latex:

$$\backslash \Omega = \backslash \left(\begin{array}{cc} \{ c \} & \{ 0 \} \\ \backslash \Gamma & \backslash \Gamma \end{array} \right)$$

Rendered Predicted Image:

$$\Omega = \left(\begin{array}{cc} 0 & I \\ \Gamma & 0 \end{array} \right)$$

Original Image: 3

$$\frac{g_{SYM}^2 \Sigma_i^2}{\Sigma_1 \Sigma_2 \Sigma_3 \Sigma_4}.$$

Predicted Latex:

$$\frac{g_{SNM}^2 \Sigma_i^2}{\Sigma_1 \Sigma_2 \Sigma_3 \Sigma_4}.$$

Rendered Predicted Image:

$$\frac{g_{SNM}^2 \Sigma_i^2}{\Sigma_1 \Sigma_2 \Sigma_3 \Sigma_4}.$$

Original Image: 4

$$h_i^2 = h^2 \equiv \frac{2}{r},$$

Predicted Latex:

$$h_{_ \{ i \} } ^ { \{ 2 \} } = h ^ { \{ 2 \} } \backslash equiv \backslash frac { \{ 2 \} } { \{ r \} } ,$$

Rendered Predicted Image:

$$h_i^2 = h^2 \equiv \frac{2}{r},$$

Figure 6.1 Test Result.

Error on Test Set:

In above **Actual Test results on Test set** section, it is shown that the examples do well for my model, however in this section, the scenarios in which my model couldn't able to predict the output because of the length of the inputs are shown.

$$(z_1, \mathcal{Z}_1, \mathcal{F}_1) \times (z_2, \mathcal{Z}_2, \mathcal{F}_2) = (z_1 z_2, z_1 \mathcal{Z}_2 + z_2 \mathcal{Z}_1, z_1 \mathcal{F}_2 + z_2 \mathcal{F}_1 - 2 \mathcal{Z}_1 \mathcal{Z}_2).$$

$$r_2 = r_1 \quad ; \quad t_2 = t_1 \quad ; \quad \phi_2 = \phi_1 + \gamma \quad (\phi_1 > 0) \quad ; \quad \phi_2 = \phi_1 - \gamma \quad (\phi_1 < 0).$$

In this research, the larger sizes latex formulas with more than 175 tokens are ignored during the training phase, however all the scenarios where included during test time.

Chapter 7

Conclusion

In this research, a new variant of LSTM called “LSTM with peephole connections” and Stochastic “hard” Attention model is used to address the problem of OCR. In this experiment, the dataset called Image2latex-100K is used. However, I also have generated 200K Images of the mathematical equations to train and test my model. In this research, it is show that a new variant of the LSTM has outperformed the previous work based on traditional LSTM. This work will encourage other researchers to try the new variant of LSTM for ORC or sequence to sequence related work.

For possible future work, this research can be scaled from printed mathematical formulas images to the hand written mathematical formulas images. To recognize the hand written mathematical formulas, one can implement the bidirectional LSTM with CNN [29]. It can also be used to generate an API for latex code.

References

- [1] R. I. Anderson, Syntax-directed recognition of handprinted mathematics., CA: Symposium, 1967.
- [2] K. Cho, A. Courville and Y. Bengio, "Describing Multimedia Content Using Attention-Based Encoder-Decoder Networks," in *IEEE*, CA, 2015.
- [3] A. K. a. E. Learned-Miller, "Learning on the Fly: Font-Free Approaches to Difficult OCR Problems," MA, 2000.
- [4] D. Lopresti, "Optical Character Recognition Errors and Their Effects on Natural Language Processing," *International Journal on Document Analysis and Recognition*, 19 12 2008.
- [5] WILDML, "WILDML," [Online]. Available: <http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/>.
- [6] S. a. S. J. Hochreiter, "Long Short-Term Memory. Neural Computation," 1997.
- [7] S. Yan, "Understanding LSTM and its diagrams," Software engineer & wantrepreneur. Interested in computer graphics, bitcoin and deep learning., 13 03 2016. [Online]. Available: <https://medium.com/@shiyang/understanding-lstm-and-its-diagrams-37e2f46f1714>.

- [8] C. R. a. D. P. W. Ellis, "FEED-FORWARD NETWORKS WITH ATTENTION CAN SOLVE SOME LONG-TERM MEMORY PROBLEMS," ICLR, 2016.
- [9] a. F.-F. L. Karpathy, Image captioning., 2015.
- [10] F. A. Gers, N. N. Schraudolph and J. Schmidhuber, "Learning Precise Timing with LSTM Recurrent Networks," *Journal of Machine Learning Reserach*, 8 2002.
- [11] H. F. Schantz, The history of OCR, optical character recognition., 1982.
- [12] A. Karpathy, "CS231n: Convolutional Neural Networks for Visual Recognition," [Online]. Available: <http://cs231n.stanford.edu/>.
- [13] M. Negnevitsky, "Back-propagation in neural network," in *Artificial Intelligence* , Australia, Tasmania: Pearson, 2011.
- [14] R. M. R. S. Milad Mohammadi, "Deep Learning for NLP1," Stanford, Sanfrancisco , 2015.
- [15] J. S. Sepp Hochreiter, "LONG SHORT-TERM MEMORY Neural Computation 9(8):1735{1780, 1997," Germany, 1997.
- [16] Colah, "Understanding LSTM Networks," 27 08 2015. [Online]. Available: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
- [17] I. S. O. V. Q. V. Le, "Sequence to Sequence Learning with Neural Network".
- [18] K. C. Y. B. Dzmitry Bahdanau, "Neural Machine Translation by Jointly Learning to Align and Translate," in *Accepted at ICLR 2015 as oral presentation*, 2014.
- [19] D. B. a. K. C. Y. Bengio, "Neural Machine Translation By Jointly Learning To Align And Translate," in *ICLR*, 2015.

- [20] l. c. heuritech, "heuritech Le Blog," [Online]. Available:
<https://blog.heuritech.com/2016/01/20/attention-mechanism/>.
- [21] C. M. Bishop, "Pattern Recognition and Machine Learning.," in *Pattern Recognition and Machine Learning*.
- [22] K. Xu, J. l. B. K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel and Y. Bengio, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention".
- [23] K. S. A. V. A. Z. Max Jaderberg, "Reading Text in the wild with Convolutional Neural Networks," Springer Science+Business Media New York 2015, New york, 2014.
- [24] OPENAI, "Requests for Research," [Online]. Available:
<https://openai.com/requests-for-research/#im2latex>.
- [25] Zendodo, "im2latex-100k arXiv:1609.04938," 21 06 2016. [Online]. Available:
<https://zenodo.org/record/56198#.WgYosRNSzOT>.
- [26] K. C. B. v. M. C. G. F. B. H. S. D. B. Y. Bengio, "Learning Phrase Representations Using RNN Encoder-Decoder For Statistical Machine Translation," 2014.
- [27] T. K. M. S. R. R. W. C. F. T. H. Okamura, "Handwriting Interface for Computer Algebra System," Kyusha.
- [28] Y. D. l. A. K. l. A. M. R. l, "What You Get Is What You See: A visual Markup Decompiler," 2016.
- [29] Y. D. A. K. J. L. A. M. Rush, "Image-to-Markup Generation with Coarse-to-Fine Attention," 2017.