Application of attention based models in Deep Neural Networks

Submitted in partial fulfillment of the requirements for

the degree of

Master of Science

in

Electrical and Computer Engineering

Parth Chadha

March 2016

Abstract

Convolutional Neural Networks have had great success in advancing the state of the art in image classification and detection. Yet these architectures do not scale well for large images. Convolutional Neural Network architectures have a defined input image size and if input image is larger, then we have to either process sub-parts of input image or rescale the input image. Instead of Convolutional Neural Networks, Recurrent Neural Network architecture can be used to extract information from an image by selecting sequence of regions based on the previous history. Convolutional Neural Networks can be applied to these selected regions to process information and create an intermediate representation.

Using this recurrent architecture, the amount of computation required is constant irrespective of the size of input image. The recurrent architecture is based on an attention model, where the network attends (or selects) to a particular portion of the image. Since this discrete action is non-differentiable, this part of the network is trained using policy gradients. This work evaluates the recurrent architecture and techniques required for training the differentiable and non-differentiable part of the network. Also, a variant of the network is proposed, where the model will be able to adjust the amount of information captured based on the scale of the object in input image. This proposed model is validated with a scale-varying digits dataset.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Convolutional neural network(CNN) architectures such as AlexNet [4], VGGNet [5] have recently had great success on tasks such as image classification as demonstrated on the ImageNet [6] benchmark. These architectures have reached human-level accuracy on several defined datasets. CNN's have a major drawback when the input image is larger than a CNN's accepted input image size, we either have to rescale the input image or the CNN must consider sub-parts of the original image. In the former case we sacrifice on the details of the image and in the latter case, we increase the amount of computation linearly.

An important property of human perception that can be utilized in designing deep learning architectures is selective attention. Human vision tends to focus selectively on parts of a scene and ignore other information due to the limitation of the visual processing unit. Literature in cognitive science and neuroscience has shown that humans move their focus to the next relevant part of the scene and combine information from different movements over time to build up an internal representation of the scene [7] [8].

Visual attention has two fundamental aspects to it:

1. Accumulate information from different fixations over time and decide on which part of the scene to focus on in next fixation.

2. Allocate the limited compute resources on the selected part of the scene.

1.1 Attention mechanisms in computer vision

Researchers have been actively working to incorporate attention mechanisms in computer vision. Previously, researchers have explored saliency detectors, which is motivated by human vision. Humans have

the ability to detect salient objects and then move their attention around the scene to gather visual information. Algorithms to extract salient information from a scene have been proposed in the literature [9] [10]. These algorithms extract low-level features such as edges, intensity, etc to find salient image regions. Hence, they can not work on task specific semantic information present in the scene. Also, these mechanisms have two fundamental drawbacks:

1. Different salient regions are independent of each other, whereas in human perception next attended region is decided based on accumulated information from the past.

2. Information from different salient regions is not integrated to create a global representation.

Many prior works have incorporated sliding window techniques [11], which aims to achieve same goals as attention mechanism via selecting relevant window's of interest in an image. Since there are infinite possible types of windows, a lot of recent research focusses on how to reduce the number of windows for which the classifier is evaluated. These techniques deal with individual windows independently and therefore can not incorporate information from the past for future prediction.

1.2 What problems do attention mechanisms solve in Deep learning?

Attention helps by limiting the data presented to the deep neural network and hence is computationally more efficient. In addition, it can be used for variable size input problems. Image classification is one of the most fundamental applications in computer vision. While recent literature uses convolutional neural networks for the classification task, it poses difficulties as described above in scalability. Recent work in [12] and [13] proposes the use of Recurrent Neural Network(RNN) based image classification models which also incorporates an attention mechanism. This model makes a decision at every time step on where to 'look' next based on the previously accumulated information. Further, it also builds up an intermediate representation and at the last time step predicts class of the object.

Another fundamental problem in scene understanding has been automatic caption generation. Caption generation is a hard problem since not only does it need to understand the object classes in an image, but also the *relationship* between these objects in that particular scene. It describes this relationship in a coherent descriptive language. Recent work in [14] and [3] propose a hybrid architecture of Convolutional neural networks and Recurrent neural networks with incorporated attention mechanism. Given an input image, a CNN is used to find a lower dimensional representation of the image using features extracted from convolutional layers. The attention mechanism then selects a subset of these feature vectors and passes to the RNN to generate descriptive sentences.

Chapter 2

Deep Neural Networks

Deep learning provides a mechanism in learning systems to build up complex concepts out of simpler concepts. Methods of representing high level concepts using simple elementary representations is a fundamental problem in representation learning [15].

This section describes some core concepts and workings of deep learning. First, the basic structure of an artificial neuron and neural networks is discussed. Next, deep neural network architectures such as convolution neural networks and recurrent neural networks are introduced, which are used in the experimentation of this thesis.

2.1 Basics of neural network

The architecture of neural networks was inspired by the goal of modeling biological neurons, but recent advances in neural networks have since diverged. The goal of designing neural networks architectures has been to achieve good results in allocated tasks without any close relation to biological counterpart.

An artificial neuron receives weighted input connections from other neurons (x_i is the signal and w_i is the weight) and produces output signal delivered to other neurons.

$$y = f(\sum_i w_i \times x_i + b) \tag{2.1}$$

The weights w signify the strength of influence of one neuron over the other and are learnable parameters in a neural network.

Here, f is the activation function which applies non-linearity to the weighted input sum and also squashes the output signal. Historically, sigmoid functions ($\frac{1}{(1+\exp^{(}-x)}$) were used, but nowadays neural networks use *Tanh(x)* or rectified linear units(ReLU) ($f(x) = max(0, x)$).

2.2 Layer-wise organization

Neural networks are collections of neuron that are connected in such a manner that output's of a neuron are connected to input of other neurons such that there are no cycles in the networks.

Neurons are grouped into layers, called input layer, output layer, and hidden layers. Input layers consist of neurons directly connected to the input signal. Hidden layers connect the input layer to the output layer. Output layers are sets of neurons which compute the output of the network.

Networks with more than 1 hidden layer are usually referred to as deep networks. A simple fully-connected layer as shown in Figure 2.1 connects all neurons of input with output pairwise.

This fully connected layer can be represented mathematically as - $W \times X + b$, where W is the weight matrix of size $N2 \times N1$, X is the input of dimension $N1 \times 1$ and b is the bias of size $N2 \times 1$, and output is of dimension $N2 \times 1$.

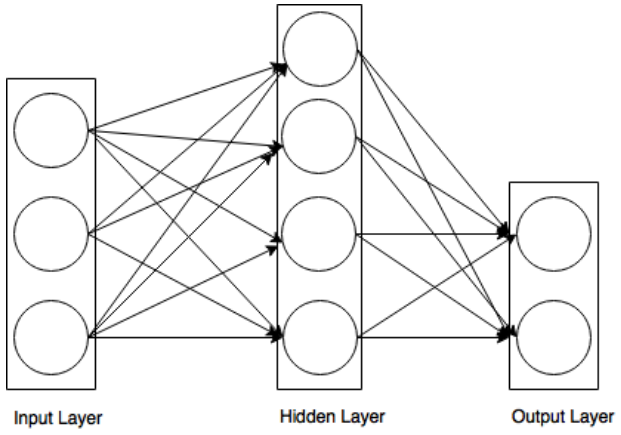

Figure 2.1: 2-layer Network with one hidden layer of 4 neurons and one output layer with 2 neurons, and three inputs

2.3 Convolutional Neural Networks

Convolutional Neural Networks(CNN) are neural networks designed under the assumption that the input to the network are images. CNN's are designed specifically to recognize two/three-dimensional shapes with high degrees of invariance to translation, scaling and other forms of distortions.

Layers of a CNN have neurons arranged in 3 dimensions of *width, height, and depth*. CNN's are usually composed of layers such as:

1. *Convolutional Layers*

Convolutional layers consist of a set of learnable filters, where each filter is small spatially but extends along the complete depth of the input. A typical convolutional layer would have a set of these filters which we slide over the input image during forward pass of the network. We get an activation map corresponding to each filter, and stacking of these activation maps of all the filters produces an output volume as shown in Figure 2.2.

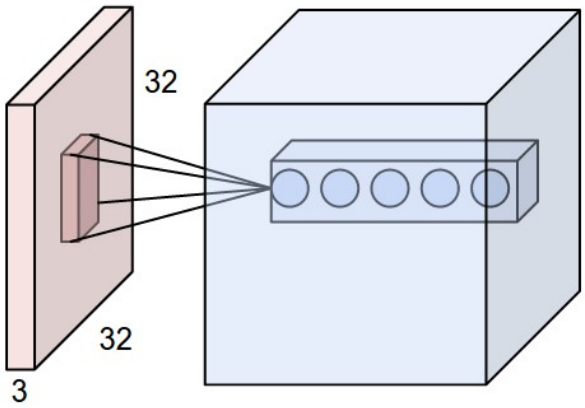

Figure 2.2: Input image of size 32×32. Each neuron in Conv Layer is connected to a local region in the input(spatially local but across full depth of image). The other neurons in Conv Layer perform a similar function but with a different set of weights. Image Credit: [1]

In terms of neurons, each neuron in Convolutional layer connects to a local region of the input(local spatially but across complete depth). The learnable parameters in this layer are the weights with which the inputs in the receptive field are connected to the neuron. For example, a (5x5) filter applied to (228x228x3) sized input image would have (5x5x3 = 75) parameters. This filter has to be convolved across the image with each filter applied at a *stride* from the previous location. When stride is 1, we move the filters one pixel at a time. So, all the neurons which apply the same filter to different locations of the input layer have shared parameters which help in drastically reducing the number of parameters in the network.

2. *Pooling Layers*

   Pooling layers are used to reduce the spatial size of the representation and also helps to control the overfitting via reducing the number of parameters. Pooling layers provide a degree of translation invariance to the system.

   There are several ways to perform the pooling operation, such as taking the average or the maximum of inputs in a defined spatial block. The Pooling layer is generally applied in between two

consecutive Convolutional Layers.

3. *Fully Connected Layers*

   A fully connected layer takes all neurons in the previous layer and connects it to all the neurons in this layer as shown in Figure 2.1.

2.4 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a neural network which is specialized to process sequence of data such as an input sequence $x = \{x_1, x_2, \ldots, x_T\}$. Traditional feedforward neural networks operate on a fact that different inputs or outputs are independent of each other, whereas in a RNN that assumption is not valid.

RNN's aims to understand the relation in the sequence of inputs and predicts output based on the accumulated information. A common example would be in case of predicting the next word in an incomplete sentence, where the probability of the next word is dependent on the complete input fed to the network. RNN's can output a variable length output sequence $y = \{y_1, y_2, \ldots, y_T\}$ or a fixed sized output y_T. Figure 2.3 shows a single RNN having a recurrent input and a recurrent output with additional inputs and outputs for every time step. This cell representation is replicated over 3 cells as shown in Figure 2.4.

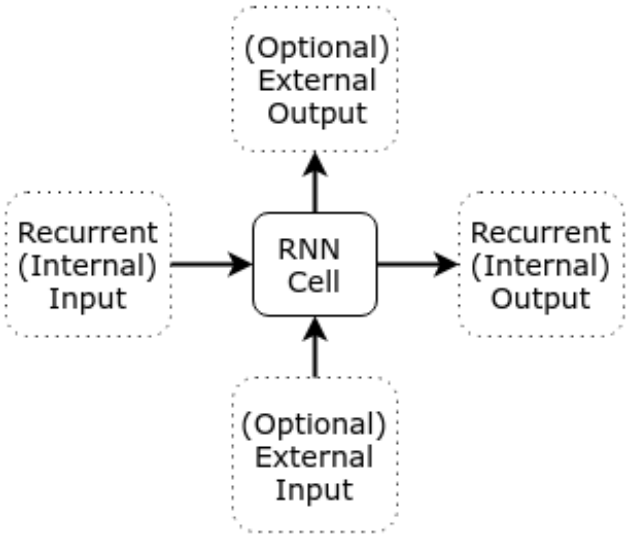

Figure 2.3: The structure of single RNN cell. Image credits: [2]

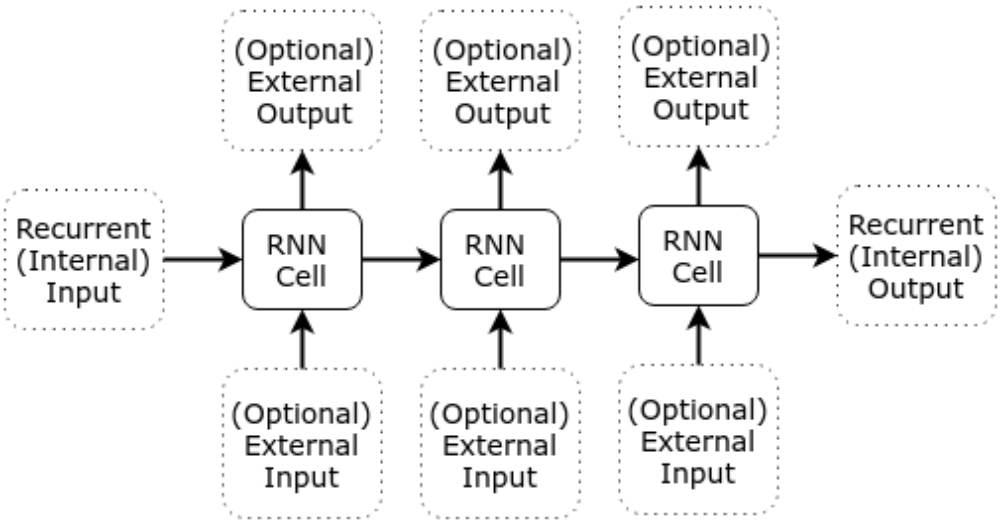

Figure 2.4: The structure of a 3 RNN cells stacked together. Image credits: [2]

Figure 2.6 shows time unrolling of a RNN cell. We can see that at every time step each RNN cell receives a input from previous RNN cell weighted by W matrix. At each time step RNN cell receives input weighted by matrix U. Also, any RNN cell can emit an output which is weighted over matrix V. The following equations model the RNN cell:

$$\begin{aligned} a^t &= b + W \times s^{t+1} + U \times x^t \\ s^t &= tanh(a^t) \\ o^t &= c + v \times s^(t) \\ p^t &= softmax(o^t) \end{aligned} \tag{2.2}$$

RNN's use the same parameters U, V *and* W across the time steps, and this technique is known as parameter sharing. By employing the concept of parameter sharing in RNN's makes it possible to extend and apply the model to examples of different lengths and generalize across them. Parameter sharing helps in generalizing the model to sequence lengths not seen during the training procedure. It also helps in establishing a statistical strength across different sequence lengths and positions [15] which ensures that model is invariant to where a particular piece of information occurs. As mentioned by Goodfellow et al.[15], our model should be able to understand the same conceptual information from following two sentences: "I went to Nepal in 2009" and "In 2009, I went to Nepal.".

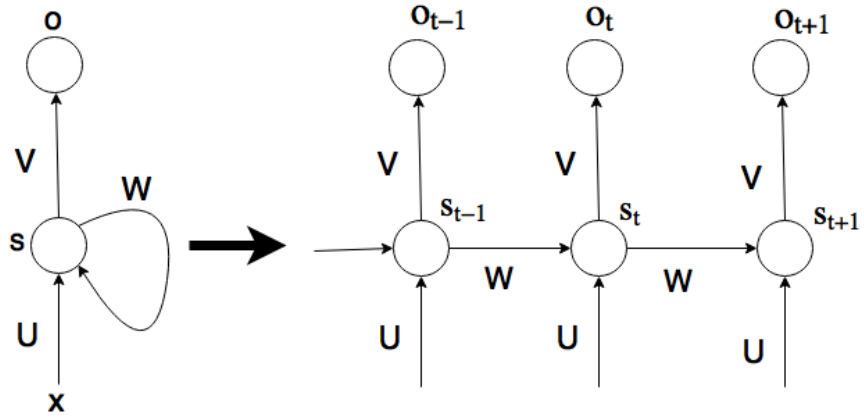

Figure 2.5: RNN unfolding over time with U,V & W learnable parameters.

2.4.1 Long-Short Term Memory(LSTM)

LSTM's are a variant of RNN and were first proposed by Hochreiter and Schmidhuber [16]. LSTM's were designed to tackle vanishing gradients problem with RNN.

RNN's when back-propogated over a large number of time steps, face a problem of diminished value of gradients. Due to diminished gradient values, the network doesn't tend to learn long term dependencies. With the presence of memory cell in LSTMs, we have gradient flow which is continuous gradient flow which help in learning long-term dependencies.

One cell consists of input, forget, output gates and a cell unit. All the gates use a sigmoid activation, while input and cell state is transformed using *tanh* layer. The input gates control the current state information for current input to be propagated. The forget gate defines how much of the previous state to be allowed to propagate. The output gate controls the information of internal state propagated to external network.

LSTM cell can be defined with a following set of equations:

$$
\begin{aligned}
i\_t &= \sigma(\theta_{xi} \times x_t + \theta_{hi} \times h_{t-1} + b_i) \\
f\_t &= \sigma(\theta_{xf} \times x_t + \theta_{hf} \times h_{t-1} + b_f) \\
o\_t &= \sigma(\theta_{xo} \times x_t + \theta_{ho} \times h_{t-1} + b_o) \\
g_t &= \sigma(\theta\_xg \times x_t + \theta_{hg} \times h_{t-1} + b_g) \\
c_t &= f\_t \cdot c_{t-1} + i_t \cdot g_t \\
h_t &= o\_t \cdot tanh(c_{t-1})
\end{aligned}
\tag{2.3}
$$

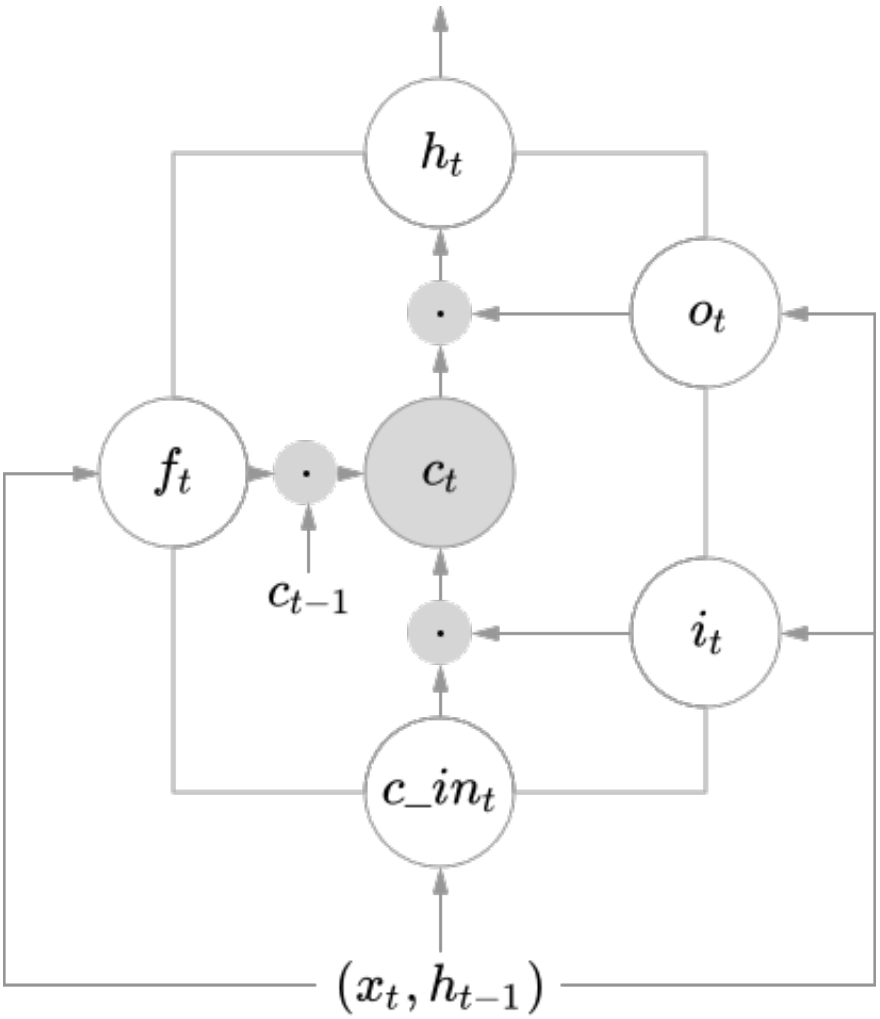

Figure 2.6: LSTM cell with input, output and forget gates

Here θ is the set of weights connecting different parts of network. i_t, f_t and o_t are forget gates as shown in Figure 2.6. x_t is the input at any time step and h_t is the information received from previous time-step's LSTM cell.

LSTM's have been used to a variety of problems such as Speech recognition and language translation, image captioning etc.

We use LSTM's in this work in our Recurrent Attention Architecture to be able to learn the dependencies over long time-steps to predict the next location our architecture would fetch information from.

Chapter 3

Attentional mechanism

Attention based models have application in problems which are defined for sequence to sequence translation or encoder-decoder models as described in Section 3.1. Attention based model allows the model to attend a specific part of the input vector instead of processing it as a whole. In an abstract understanding, model learns the technique of which specific part to choose or "attend".

3.1 Encoder-Decoder model for sequence to sequence problems

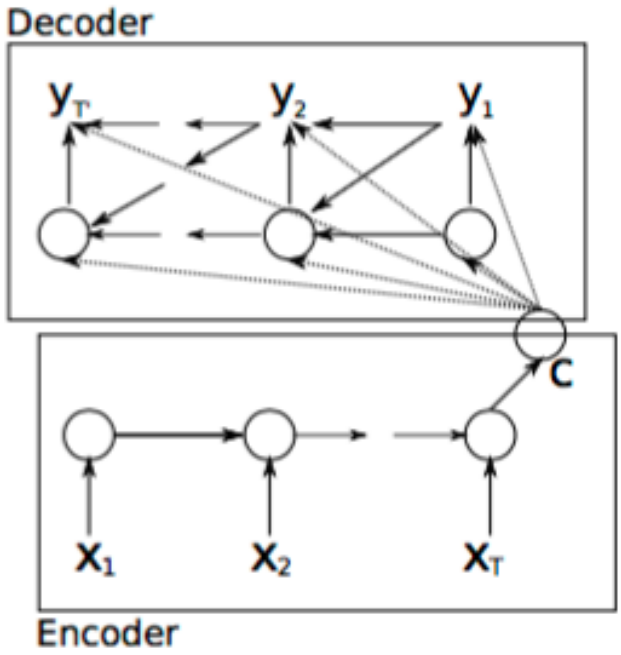

Figure 3.1: The structure of encoder and decoder in which x_t is input sequence and y_t is the output sequence. Image credits: [3]

The sequence to sequence problem is defined as to map the input sequence to an output sequence. Figure 3.1 as described in [3] shows the model for sequence to sequence problems. For example in language translation task, source language sentence is the input sequence. For object detection task, the input sequence is an image which can be represented as a list of objects.
There are two parts of the structure as shown in Figure 3.1, the encoder and the decoder and both of them are structured as a neural network. As mentioned in [3], encoder-decoder networks are a general framework using neural networks for mapping structured inputs to structured outputs.
Since in the sequence to sequence problem, the length of input and output is not fixed, there is a requirement for intermediate representation(c) between encoder and decoder to maintain generalisation [3].

3.1.1 Encoder

The encoder reads the input data x and maps into a representation c.

$$c = f_{enc}(x) \tag{3.1}$$

Here, f_{enc} represents the encoder neural network. In the case of input as image, we use a Convolutional neural network (CNN) as encoder and extract features vector or *cube* from convolutional layers.

3.1.2 Decoder

Decoder generates output y conditioned on context c of the input.

$$p(y \mid x) = f_{dec}(c) \tag{3.2}$$

When we are describing a image with natural language for image caption task, we use RNN as a decoder. In general, whenever an output sequence is required, we use a Recurrent Neural Network.

3.1.3 Issue with encoder-decoder framework

One major problem with encoder-decoder framework is that encoder always compresses the input vector into a single fixed dimensional representation vector c.
When working with real world datasets, not all of the images contain the same amount of information, and hence to describe the varying amount of content in the image with a single fixed dimensional vector is not a good design.

3.1.4 Attention based model

Attention based models are motivated from the fact that instead of using the complete encoding produced by the encoder, the decoder can instead use selective part of the representation c. The selective part which model choses is known as the region on which model *"attends"*.

Let the encoding produced by the encoder be a set of fixed size vectors:

$$c = \{c_1, c_2, \ldots, c_M\} \tag{3.3}$$

In the case of image, the encoder produces encoding where each c_i indicates a spatial location of the image.

The attention model takes input from hidden state of decoder at previous time step z_{t-1} and score the context vector c_i. This signifies which vector c_i is most relevant to focus on for next timestep.

e_i^t signifies the scores and α_i^t signify the attention weights given to each element of context vector c.

Scores e_i^t are calculated as:

$$e_i^t = f_{ATT}(z_{t-1}, c_i, \alpha_j^{t-1}) \tag{3.4}$$

Attention weights α are calculated by applying $softmax = \frac{exp(e_i^t)}{\sum_j exp(e_j^t)}$ to scores e.

Using attention weights and previous context vector, we can calculate the new context vector. In soft attention model, we calculate the new context vector as:

$$c^t = \sum_{i=1}^{M} \alpha_i \times c_i \tag{3.5}$$

In hard attention model, we chose a context vector by sampling using any distribution: [1]

$$\begin{aligned} c^t &= c_{r^t} \\ r^t &\sim Cat(M, \{\alpha_i^t\}_{i=1}^M \end{aligned} \tag{3.6}$$

With the new context vector c^t, we calculate the new state of decoder, which is RNN in image captioning case.

$$h\_t = \phi_\theta(h_{t-1}, x_t, c_t). \tag{3.7}$$

3.1.5 Types of attention mechanisms

There are fundamentally four different kinds of attention mechanisms as described in [17]:

[1] The notations are followed from the work by Cho et.al [3].

1. Item-wise

   Item wise mechanism requires the input to be a sequence containing different items.

   a) Soft attention

      During the decoding step, item-wise soft attention calculates the weight of each input object and then makes a linear combination of them.

   b) Hard attention

      In item-wise hard attention, decoder makes a hard decision and stochastically picks an object from the input.

2. Location-wise

   Location wise mechanism does not want input to be a sequence of items and can work on a single feature map. This is mostly used when the input is an image and location indicates the region of the image to focus on based on predicted location coordinate. At each decoding step, the model picks a sub-region from the input feature map and feeds it to the encoder to get the intermediate representation.

   a) Soft attention

      It accepts the entire feature map as input and performs a weighted transformation to highlight region of interest.

   b) Hard attention

      It makes a hard decision and stochastically picks a sub-region.

3.2 Image caption generation using CNN-Attention-RNN structure

In image caption generation task, the model generates a natural coherent language description of an image. The encoder-decoder framework as described in Section 3.1.1 is suitable for this task. The encoder extracts the representation of the image(using a CNN) and the decoder using this representation produces sequence of words describing the text (using a RNN). As shown in Figure 3.2, the CNN receives an image and creates a representation of the image. This representation is usually extracted from last convolutional or fully-connected layers in CNN. This representation is then used by a LSTM network to produce a text describing the image.

Attention based model for image captioning was proposed in [14], where they show the results based on both "soft" and "hard" attention models.

The design of encoder-attention-decoder pipeline solves the problem of limited information being encoded

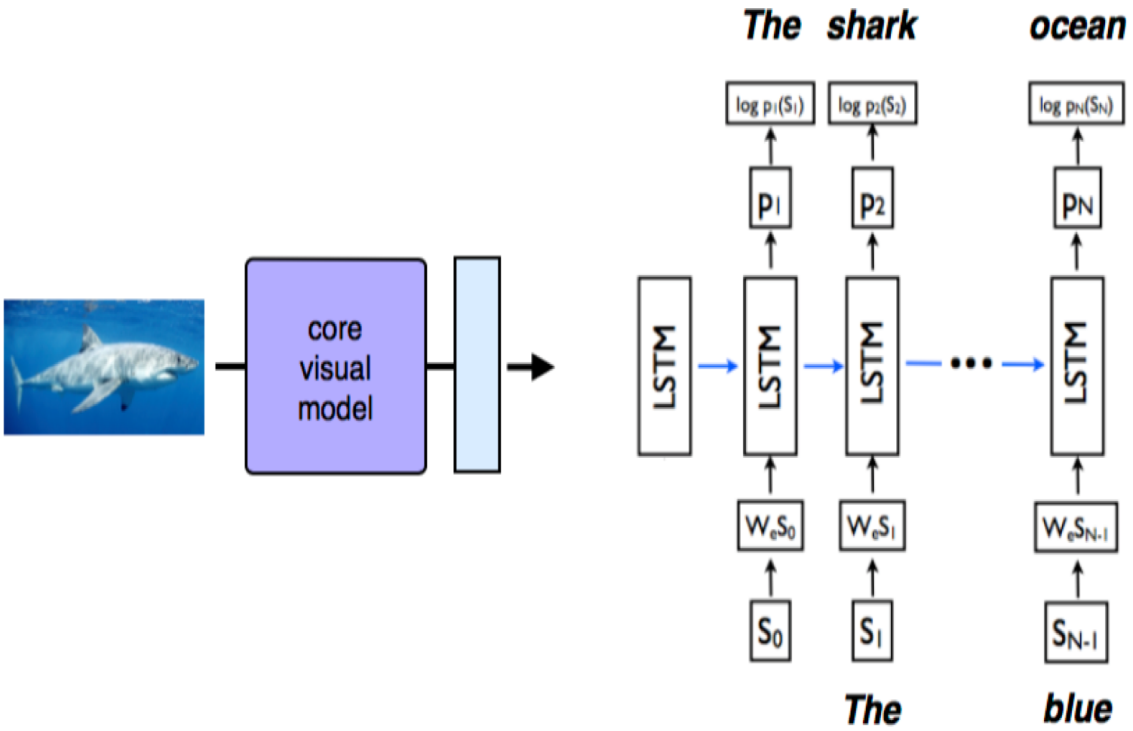

Figure 3.2: A encoder-decoder framework for captioning of image. Here the core visual model is a CNN. Image credits: Directions in Convolutional Neural Networks at Google presentation

by the encoder. Now based on decoder's output at each time step, the model calculates the weightage given to spatial or temporal part of input. Hence, each vector output produced by the encoder now describes a particular region of the input. Attention mechanism learns to chose which information needs to be focussed at a particular time step.

Figure 3.3 shows how attention models are interfaced with CNN and LSTM networks. The activation maps produced by a CNN are received by the attention model. At each time step the attention model make a "soft" or "hard" decision on activation maps. The attention model makes this decision based on previous state vector h_{t-1}. z_k is the representation created by attention model which is received by the decoder(LSTM here) to output coherent words.

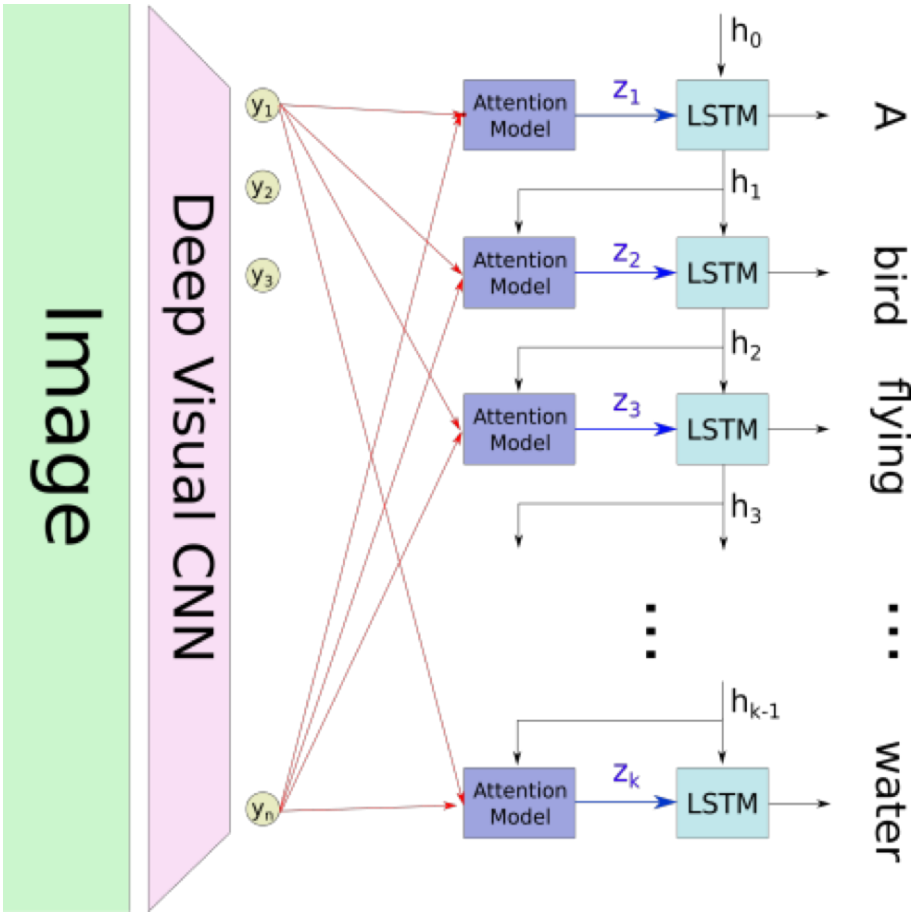

Figure 3.3: CNN-Attention-LSTM model for image captioning. Image credits: [18]

Chapter 4

Recurrent attention mechanism

Human vision has an important property of selective attention, where humans do not process the whole scene at once, instead tends to focus selectively on parts of a scene. Selective attention provides multiple advantages:

- More efficiency, as fewer pixels are processed by bandwidth limited system.
- Reduced task complexity as the cluttering can be avoided and sensor can be focussed directly on the task of interest.

4.1 Recurrent attention model architecture

Recurrent architecture as proposed by **Mnih et al.** [12] takes inspiration from literature in cognitive science [19] about selective attention in human vision. This architecture performs visual processing based on attention based neural networks.

This section briefly introduces the recurrent attention model. The task performed by Recurrent attention model(RAM), can be considered as the sequential decision process of an agent interacting with a visual environment.

In this model, the agent observes the environment(true contents of the image) with a bandwidth constricted sensor and therefore observes a small local region of the input. Based on accumulated information from the past, the agent then decides on where to deploy this sensor over time. Finally, the agent predicts the class of the object at the final time-step and receives a scalar reward signal.

4.1.1 Model

The model as described in [12] is built around a Recurrent neural network. The model processes the input at each time step and combines information over time via attending different locations at each time step. This combined information is an internal dynamic representation created by the model for the given task(e.g. image,video etc). Based on this accumulated information, the model chooses where to deploy the sensor at the next time-step(i.e. to sample a location from all available locations).

The number of parameters of the model remains constant and is independent of the size of the input image. This model can be understood as an *encoder-decoder* problem, where the encoder at each time step receives a portion of input image and encodes it to a fixed size vector. The decoder takes the fixed size output of the encoder to sample the next location and at the final time-step makes a classification decision.

Sensor:

The agent receives a portion of the input image x_t via it's bandwidth limited sensor ρ. This bandwidth limited sensor(Figure 4.1) produces representation similar to retina (high-resolution patch of a small area, and successively lower-resolution patches of larger areas).

This is similar to human vision, where we can clearly see the details of what we focus our attention on, while observing a blurry sense of the nearby regions. This low-level representation produced by the sensor is referred as a *glimpse* as described in [12] and shown in Figure 4.1.

Glimpse sensor returns a representation $\rho(x_t, l_{t-1})$ of the image x_t around the location l_{t-1}.

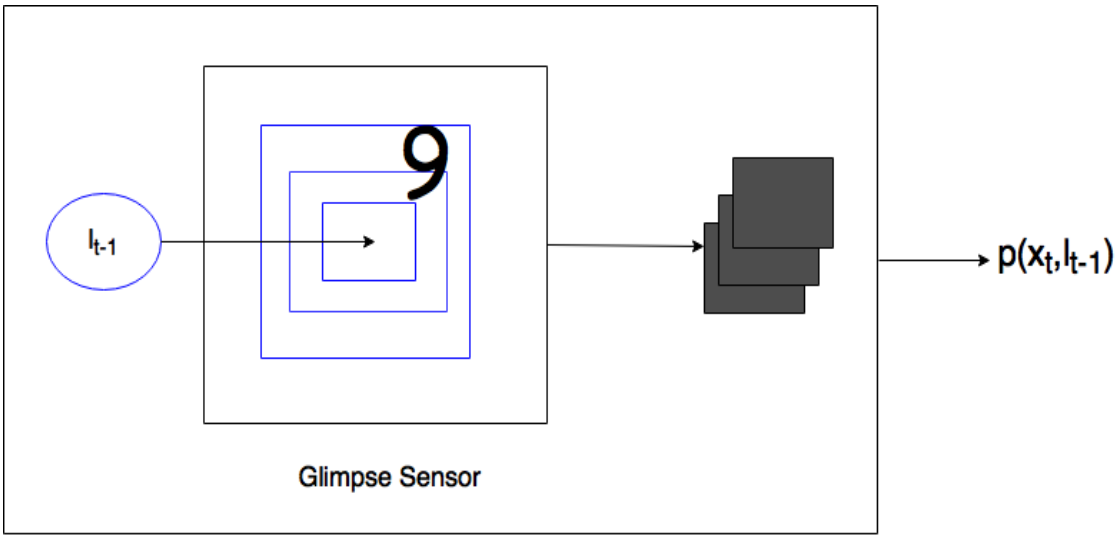

Figure 4.1: Glimpse Sensor- Input: Location coordinates, Output: Retina like representation $\rho(x_t, l_{t-1})$ centered at l_{t-1} and containing multiple resolution images

Glimpse Network:

Glimpse network receives the input image x_t and location l_{t-1} as input and it's function is to produces a fixed size feature vector g_t as output.

$$g_t = f_g(x_t, l_{t-1}; \theta_g) \tag{4.1}$$

where θ_g is set of learnable parameters.

- Glimpse sensor is also a part of the glimpse network. Output of glimpse sensor $\rho(x_t, l_{t-1})$ is mapped to hidden space using linear layer parameterized by θ_g^θ. Instead of a linear layer, a Convolutional Neural Network can be used to produce this feature vector. In datasets such as MNIST, where the complexity of task in terms of number of pixels or objects is simple, we can prefer to use a linear layer instead of a CNN. By using a linear layers we can avoid additional computation required by a CNN. Experimentation results show this tradeoff does not lead to any difference in performance for such datasets.
- Location information is mapped to hidden space using a separate linear layer parameterized by θ_g^l.

The two mapped linear layers are combined using another linear layer and final output vector g_t is produced.

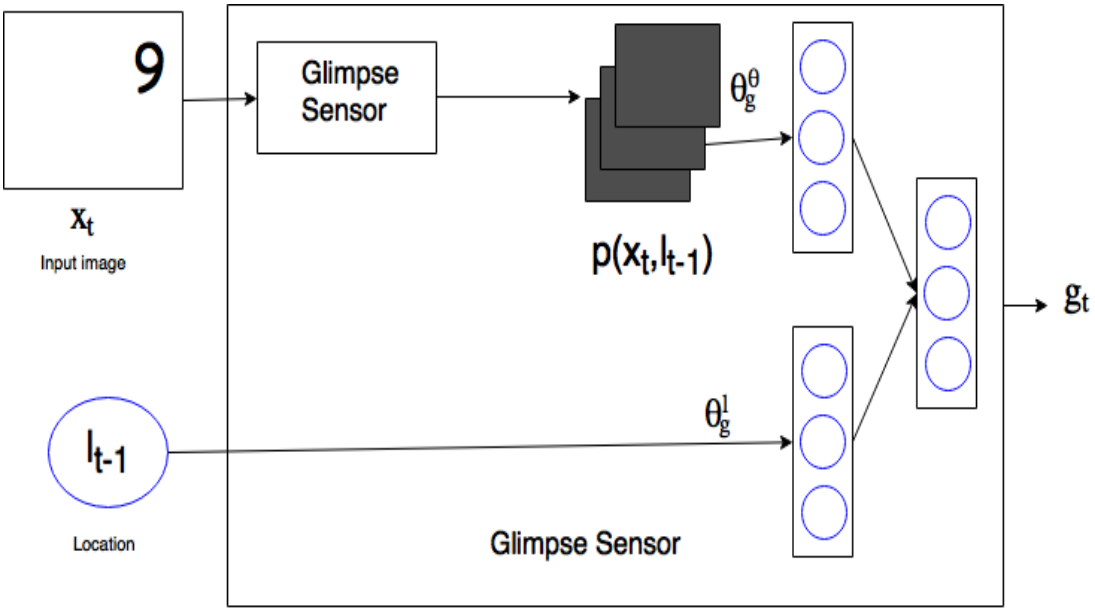

Figure 4.2: Glimpse Sensor- Input: Location coordinates, Output: Retina like representation $\rho(x_t, l_{t-1})$ centered at l_{t-1} and containing multiple resolution images

RNN state:

RNN is the where the glimpse network (g_t) and the recurrent layers (h_{t-1}) come together. The output of this layer is the hidden state at time t (h_t). This internal state contains information from the past observations made by the model and summarizes the current knowledge of the model about the environment.

$$h_t = f_h(h_{t-1}, g_t; \theta_h) \tag{4.2}$$

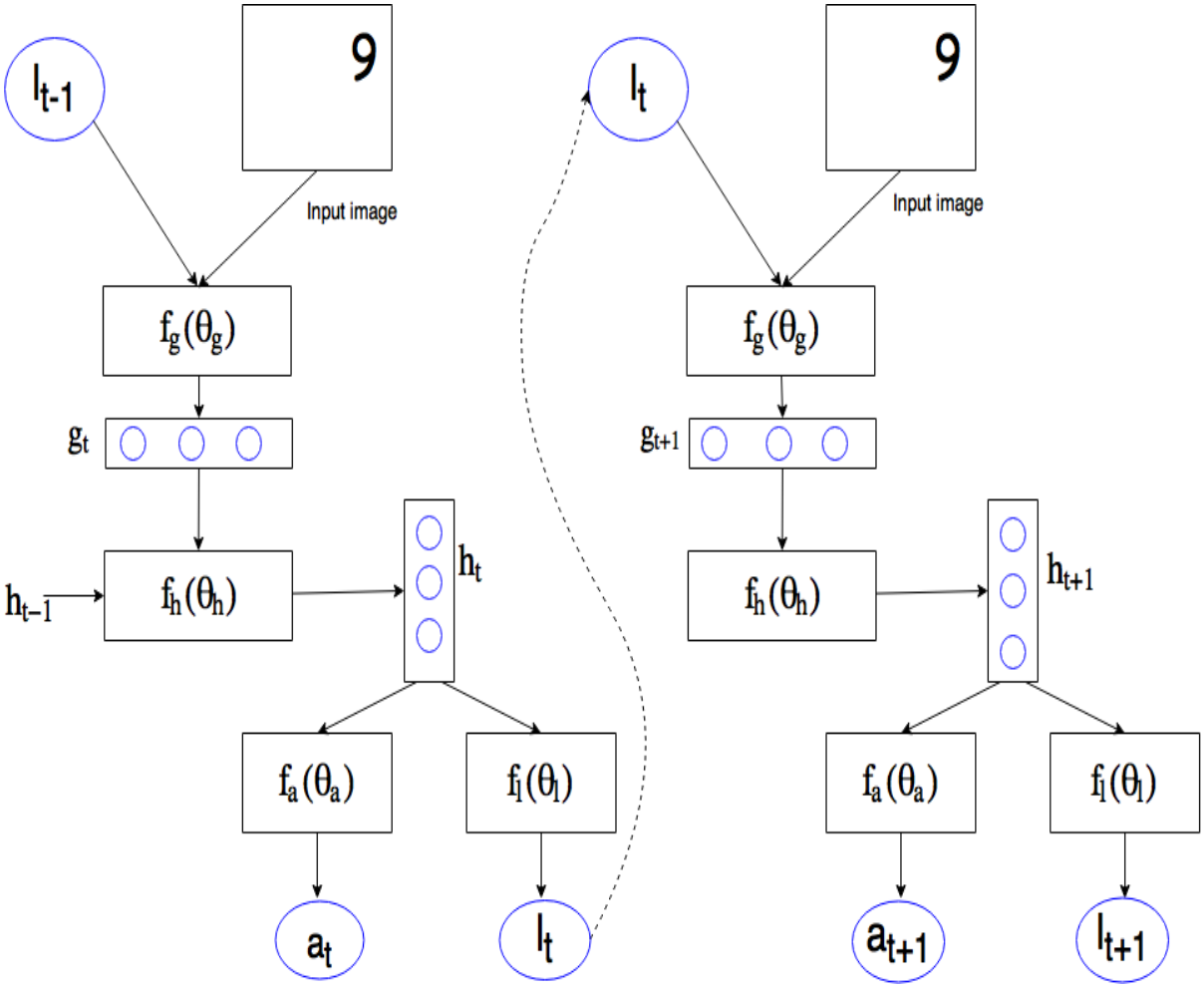

Figure 4.3

Location network:

The agent decides at each time-step where to deploy it's sensors. To chose the next location (l_t), a location value is drawn stochastically from a distribution parameterized by the location network ($f_l(h_t, \theta_l)$) :

$$l_t \sim p(.|f_l(h_t; \theta_l)) \tag{4.3}$$

Location network uses the hidden state vector h_t as the input to produce the mean of a distribution as output. Based on hyper-parameter's such as standard deviation and the generated mean, samples are stochastically drawn from this distribution to generate next location. These hyper-parameters are important variables and require fine-tuning to the particular task on hand.

Reward structure:

Agent receives a reward of 1 when it correctly classifies the object and 0 otherwise. This reward is received at the final time-step and hence is a delayed reward. The goal of the agent in this framework is to maximize the sum of all rewards it received.

$$R_t = \sum_{t=1}^{t=T} r_t \tag{4.4}$$

4.2 Why use recurrent architecture for object detection?

Recurrent architecture performs image classification based on location based hard attention mechanism. In this architecture, the model stochastically picks a region from the image and feeds it to feed-forward network. Based on the computation by the network, the model subsequently choses the next region. After repeated steps of glimpse and computation, the model integrates the information from previous time-steps to make a classification decision. Hence, this model performs both image classification and detection tasks.

4.3 Training of Recurrent Attention Model

The architecture of the model as described in Section 4.1 can be considered as setup of Partially observed Markov Decision Process (POMDP) [20]. Since the agent uses a bandwidth limited sensor to observe the environment, it is not aware of the true state of the environment at any time.
Also, the agent takes the following actions based on prior information:

- Chose the best possible location to attend in the next time-step
- At the final time-step, predict the class of the object

The agent has to learn the parameters of the model such that it can chose best possible action based on the prior interactions with the environment.

$$s_{1:T} = x_1, l_1, a_1, \ldots, x_{t-1}, l_{t-1}, a_{t-1}, x_t \tag{4.5}$$

where, x_t is the image, l_t is the location and a_t is the action at time t.

Since the model makes hard action decision's for choosing location at each time-step, the derivates through this stochastic decision are zero. These derivates do not exist since the decisions are discrete. Because of non-existent derivates, this part of the network can not be trained with standard backpropogation algorithm.

The training mechanism as proposed in [12] uses an end-to-end training procedure that uses backpropogation algorithm to train the neural network component of the model and policy gradients for non-differentiable part of the problem. This procedure lets the model to be trained with respect to the given task.

4.3.1 REINFORCE algorithm

The setup of RAM architecture is a Partially observable Markov Decision Process(POMDP). The agent in this architecture learns a stochastic policy since the agent does not have a sense of true state of the environment (original image).

The stochastic policy is described as $\pi_\theta(a_{0:T}|h_{0:T})$ where $a_{0:T}$ is the set of actions and $h_{0:T}$ is the history or collection of observations ($h_t = \{o_0, o_1, \dots o_t\}$). Here, policy maps history or set of observations to actions. Policy π_θ parameterized over θ is defined by Recurrent Neural Networks or its variants such as Long-Short Term Memory(LSTM).

We aim to maximize the reward our agent receives over time. In case of object classification as discussed in this report, agent receives an reward at the final step of classification and hence the reward is delayed reward.

Algorithm

The aim of this algorithm is to maximize the expected reward over the policy π.

We can define the expected reward as J^π, and we maximize J^π parameterized by θ.

$$\begin{aligned} J^\pi(\theta) &= \mathbb{E}\left[\sum_{t=0}^{T} r_t(a_t)\right] \\ &= \mathbb{E}_{\pi_\theta(a_{0:T}|h_{0:T})}\left[\sum_{t=0}^{T} r_t(a_t)\right] \end{aligned} \tag{4.6}$$

Here, $r_t(a_t)$ is the reward function. Expectation $\pi_{\theta(a_{0:T}|h_{0:T})}$ is modeled over policy π parameterized by θ over actions a and history h.

We aim to find the derivative of the expected value, and follow the direction of gradient.

$$\begin{aligned} J^{\pi}(\theta) &= \mathbb{E}_{\pi_\theta(a_{0:T}|h_{0:T})}\left[\sum_{t=0}^{T} r_t(a_t)\right] \\ &= \sum_{a_{0:T}} \left(\sum_t r_t(a_t)\right) \pi_\theta(a_{0:T}|h_{0:T}) \end{aligned} \tag{4.7}$$

By taking the derivative of the expected value:

$$\nabla_\theta J^\pi(\theta) = \nabla_\theta \left(\sum_{a_{0:T}} \left[\sum_t r_t(a_t) \right] \pi_\theta(a_{0:T}|h_{0:T}) \right) \tag{4.8}$$

Since, the term $\sum_{t=0}^{T} r_t(a_t)$ is independent of θ, we can take this term out of the derivative:

$$\nabla_\theta J^\pi(\theta) = \sum_{a_{0:T}} \left[\sum_t r_t(a_t) \right] \nabla_\theta \left(\pi_\theta(a_{0:T}|h_{0:T}) \right) \tag{4.9}$$

We use a mathematical trick as shown in equation 4.10 to above equation:

$$\nabla(\log y) = \frac{y'}{y} \tag{4.10}$$

$$\nabla_\theta J^\pi(\theta) = \sum_{a_{0:T}} \left[\sum_t r_t(a_t) \right] \nabla_\theta \left(\log \pi_\theta(a_{0:T}|h_{0:T}) \right) \left(\pi_\theta(a_{0:T}|h_{0:T}) \right) \tag{4.11}$$

$$\nabla_\theta J^\pi(\theta) = \sum_{a_{0:T}} \left[\sum_t r_t(a_t) \right] \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|h_{0:t}) \right] \pi_\theta(a_{0:T}|h_{0:T}) \tag{4.12}$$

The following optimization is because reward would only be relevant for current actions, past rewards for current action would mean nothing.

$$\nabla_\theta J^\pi(\theta) = \sum_{a_{0:T}} \left[\sum_{t=0}^{T} \left(\nabla_\theta \log \pi_\theta(a_t|h_{0:t}) \sum_{n=t}^{T} r_n(a_n) \right) \right] \pi_\theta(a_{0:T}|h_{0:T}) \tag{4.13}$$

Finally, we apply a trick where we instead sample the trajectory:

$$a_{0:T}^{(i)} \sim \pi_\theta(a_{0:T}|h_{0:T}) \tag{4.14}$$

At every sample, we use the complete RNN with current value of parameters and run through the RNN. From this we can measure the expected result v/s the current result and set the reward part of the equation.

$$\nabla_\theta J^\pi(\theta) = \frac{1}{N} \sum_{i}^{N} \sum_{t=0}^{T} \left(\nabla_\theta \log \pi_\theta(a_t|h_{0:t}) \sum_{n=t}^{T} r_n(a_n) \right) \tag{4.15}$$

This learning algorithm is known as REINFORCE algorithm, as shown by Williams [21]. The gist of the

algorithm is to run the agent with current policy (current set of parameters of the network) and interact with environment to produce sequences as shown in Equation 4.5. Based on the interaction sequences, the parameters of the network is adjusted such that the log-probability of actions lead to a higher obtained reward.

4.4 Experimentation

4.4.1 Dataset

To evaluate the performance of Recurrent Attention Model, we apply it to task of object classification. We primarily use MNIST dataset [22] to evaluate our model because of the simplicity of this dataset. It allows benefits such as:

- Rapid re-iteration and re-evaluation of the model
- Better understanding of how our model might be interpreting the data due to concentration of information in selected regions.

We use custom variants of the MNIST dataset to evaluate the model.

- MNIST (28×28)

  As seen from Figure 4.4, the image consists of a digit centered in a 28×28 image. The digits are of different handwriting style and the dataset provides 60,000 images for training and 10,000 images as test dataset.

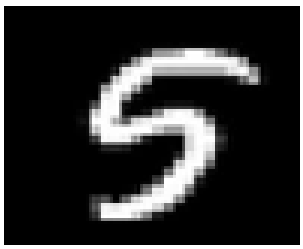

Figure 4.4: MNIST image with digit centered in 28×28 image

- Translated MNIST (60×60)

  Recurrent models for visual attention [12] paper proposed a new test dataset for non-centered digits, known as Translated MNIST. The dataset was prepared by placing the digit in a random location on a 60×60 image as seen in Figure 4.5. This dataset poses an interesting challenge for the Recurrent Attention Model to first locate the digit and then correctly classify it. Convolutional Neural Networks have translation invariance built in by the use of pooling layers and hence can tackle this task well.

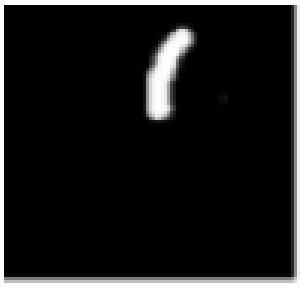

Figure 4.5: MNIST image with digit off-located from center in 60×60 image

- Cluttered Translated MNIST (100×100)

  The RAM paper [12] also propose a challenging dataset of classifying image in presence of clutter in an image. The paper suggests that systems such as CNN might be susceptible to clutter in an image because they are trained on a complete image. Whereas a RAM model should be able to learn the presence of clutter in an image, ignore it and just focus on relevant part of image. The dataset was prepared by placing patches of random digits at random location in the image as seen in Figure 4.6.

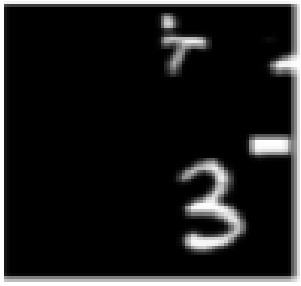

Figure 4.6: MNIST image with digit of-located from center and clutter of other digits present in 100×100 image

- Scaled MNIST (60×60)

  We propose a new additional dataset based on MNIST, which places digit with different scales at random locations within a 60×60 image as seen in Figure 4.7. This dataset provides an additional challenge for the network, where the bandwidth limited sensor has to locate the object and also adjust the scale so as correctly classify the digit.

  Human's have a foveated vision, where we can focus on a particular object in a scene (i.e. view in high-resolution format) and observe the rest of the environment in a low-resolution format. If the scale of the object is too small or huge, we can adjust the resolution at which we observe things to view them comfortably.

  With this dataset we aim to test a new model which can adjust the scale of the bandwidth limited sensor according to the scale of the object.

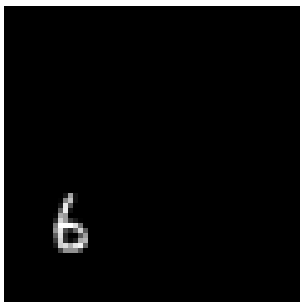

Figure 4.7: MNIST image with digit of-located from center and scale of image varied in 60×60 image

4.4.2 Model configuration

Recurrent Attention Model consists of the following sub-networks as mentioned in Section 4.1.1:

- Glimpse Sensor

- Glimpse Network
- Location Network
- Core Network(RNN)
- Action Network

Glimpse Sensor: Glimpse Sensor network receives an input image and location coordinates, and outputs a retina-like representation of the image centered at input location coordinates. For the clear visibility of action of Glimpse Sensor, we have performed the same operation on Figure 4.8 and its result can be seen in Figure 4.9.

We keep the number of square patches extracted to be 3 throughout this work and each successive patch has factor S times the height(or width) of the previous. The parameter S plays an important role in scale-learning, where we let the model sample this scale factor.

The locations output from the location network are real-valued coordinates with (0,0) as centre and (-1,-1) as the top left of the image. [1]

Figure 4.8: Image of Lenna

[1]The location coordinates convention are followed from the RAM paper [12].

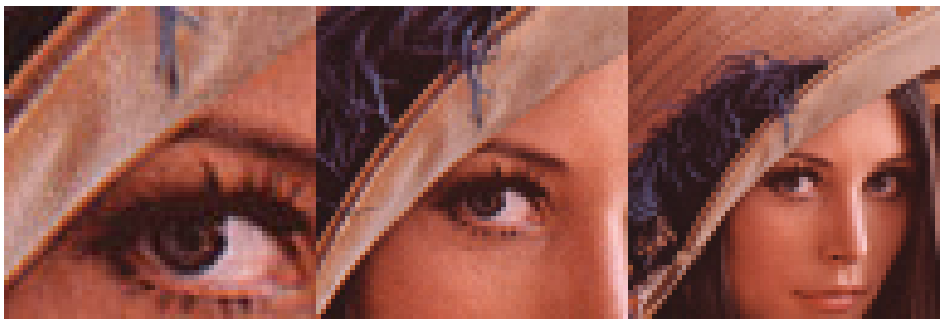

Figure 4.9: Retinal image of Lenna (Figure 4.8), centered at 0,0 and scaled 3 times with scale factor $S = 2$

Glimpse Network: The glimpse network combines information from glimpse sensor and location to output a d dimensional vector.

Glimpse sensor output : $\rho(x_t, l_{t-1})$

A linear layer $W \times X + b$ is used to map the output of glimpse sensor to an intermediary output layer(L1). Similarly, location is mapped to an intermediary layer via a linear layer (L2).

L1 and L2 are combined using another linear layer $W \times X + b$ to obtain the output vector g_t. The dimensionality of intermediary output layers for both the layer L1 and L2 were fixed at 128. The dimensionality of the output of glimpse network was fixed at 256. During experimenting with the dimensionality of these layers, it was found that increasing beyond this size made no change in the learning capacity of the network.

Location Network: The input to the location network is state vector h_t at that time step. The location mapped through a linear layer to an intermediary layer, which is then hardbound via *tanh* layer between -1 and 1.

$$\begin{aligned} L1 &= \mathbf{ReLU}(W \times h_t + b) \\ L2 &= tanh(L1) \end{aligned} \tag{4.16}$$

During training, output is sampled from a normal distribution(f_x) with mean as L2 and standard deviation as a hyper-parameter.

$$f_x = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2} \tag{4.17}$$

Here, the mean μ is obtained from L2 and σ is a hyper-parameter. Value of σ during hyper-parameter search was found to be 0.22.

During test time, instead of sampling from the distribution, output is taken to be the mean.

Core Network: The core network comprised of a rectifier units or LSTM units. We used 256 rectifier or LSTM units throughout our experimentation.

$$h_t = ReLU(W_1 \times h_{t-1} + b_1 + W_2 \times g_t + b_2) \tag{4.18}$$

Action Network: The action network makes the classification decision at the final time step. Action network receives the state vector h_t as an input and applies a *softmax* ($\frac{exp(a_k(x))}{\sum_j exp(a_j(x))}$) classifier on it.

4.4.3 Experiments on MNIST dataset

Experiments were performed on MNIST dataset using the parameters as shown in Table 4.1:

Table 4.1: Parameters for experimenting on MNIST dataset

PARAMETER	VALUE
Mini batch size	20
Learning Rate	0.01
Momentum	0.9
Glimpse Size	8x8
Glimpse Depth	1

As seen from the Table 4.1, the Glimpse size was fixed at 8x8 pixels and Glimpse depth was used as 1. Using these parameters, the model is able to capture a significant part of the centered digit. We observed that increasing the depth to 3 does not provide any significant improvement, and only increases the number of computations.

Results obtained are shown in Table 4.2. We can observe that model prediction accuracy increases with the number of glimpses it takes. This result follows the hypothesis that the model gets more confident to predict the class of the object when provided more information. With more number of glimpses, model gets more amount of information to gather. [2]

Table 4.2: Results obtained on MNIST dataset with parameters as described in Table 4.1.

Model configuration	Error (%)
4 glimpses, 1 scale	3.34
5 glimpses, 1 scale	2.29
6 glimpses, 1 scale	1.87
7 glimpses, 1 scale	1.68

[2]Throughout this thesis the number of glimpses the model is allowed to take is represented by *rho*.

4.4.4 Experiments on Translated MNIST

To perform experiments on Translated MNIST dataset, we first performed experiments to chose the ideal batch size for which the computation time is not too large. On performing experiments with batch size of 128 and 256, we found that for the batch size of 256, the network tends to learn slower as shown in Figure 4.10.

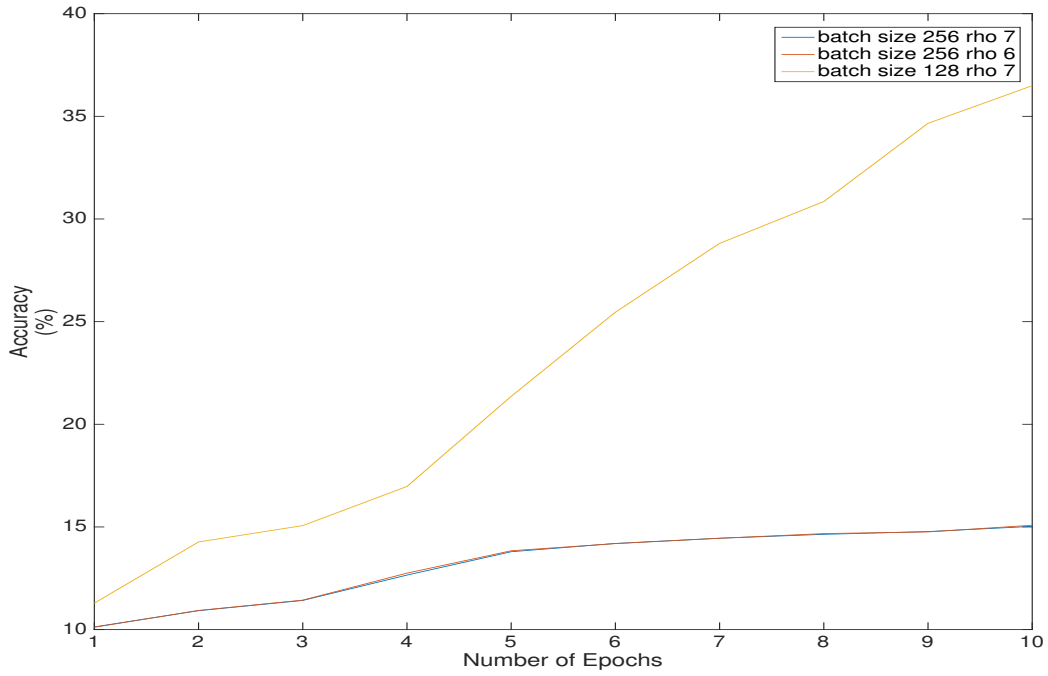

Figure 4.10: Experiment to chose batch size for Translated MNIST dataset.

Experiments were performed on Translated MNIST dataset using the parameters as shown in Table 4.3:

Table 4.3: Parameters for experimenting on Translated MNIST dataset

PARAMETER	VALUE
Mini batch size	128
Learning Rate	0.1
Momentum	0.9
Glimpse Size	12x12
Glimpse Depth	3

Since, this dataset has image size of 60×60 and the digit is not centered, we find that increasing the *glimpse depth* to 3 helps in localization and classification of digit. Increase of glimpse depth to 3 lets the systems capture information at 3 different resolutions, closest one being the highest.

The results obtained after training the model for 100 epochs are shown in Table 4.4. We can observe that the error % drops as we increase the number of glimpse the model takes.

Table 4.4: Results obtained on Translated MNIST dataset with parameters as described in Table 4.3.

Model configuration	Error (%)
5 glimpses, 3 scale	1.45
6 glimpses, 3 scale	1.37
7 glimpses, 3 scale	1.28

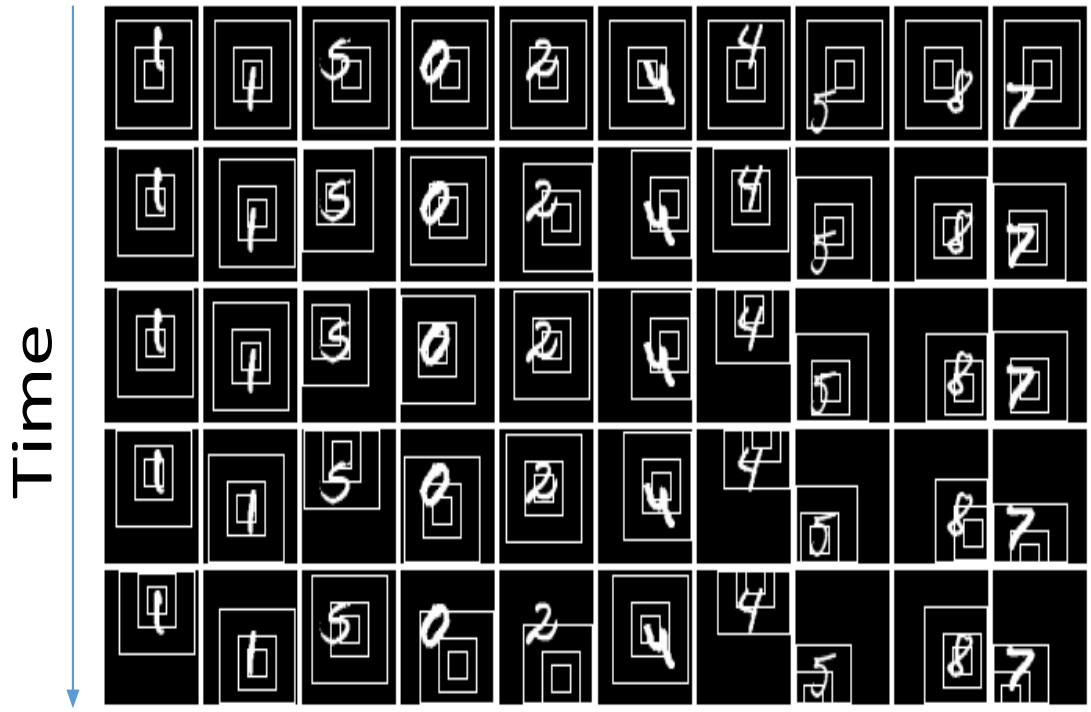

Figure 4.11: The figure shows how the glimpse location changes for time for different digits on Translated MNIST dataset.

4.5 Scale Learning

In this section we propose some modification's to the original Recurrent Attention Model(RAM) architecture. Results from the Section 4.4 are a good indication of how the RAM performs in task such as object classification. It can be seen from Figure 4.11 that model performs both object localization and classification tasks.

Though, a major limitation of this architecture is that the glimpse sensor has a fixed size retina, from where it captures information for the task in hand. In case of MNIST and Translated MNIST dataset, we

had a prior information about the size of digits in the dataset and hence we adjusted the hyper-parameter *glimpse size and glimpse depth* of the model such that it can locate and classify the digit within a reasonable number of glimpses.

Unfortunately, the real world data does not always come in one size and even in a real world handwriting data, the digits size would not be consistent. In the original RAM architecture, the glimpse sensor does not adapt according to the scale of object and the following two scenarios arise:

- Let the maximum information capacity of the glimpse sensor be represented by g and amount of information gathered by glimpse at any given time-step be represented by f. Then, if the scale of the image is smaller than the expected size, then the $\frac{f}{g}$ becomes low which leads to decrease in classification accuracy.

- If the scale of the image is larger than the expected size, then the glimpse sensor takes more number of glimpses to estimate the class of the object.

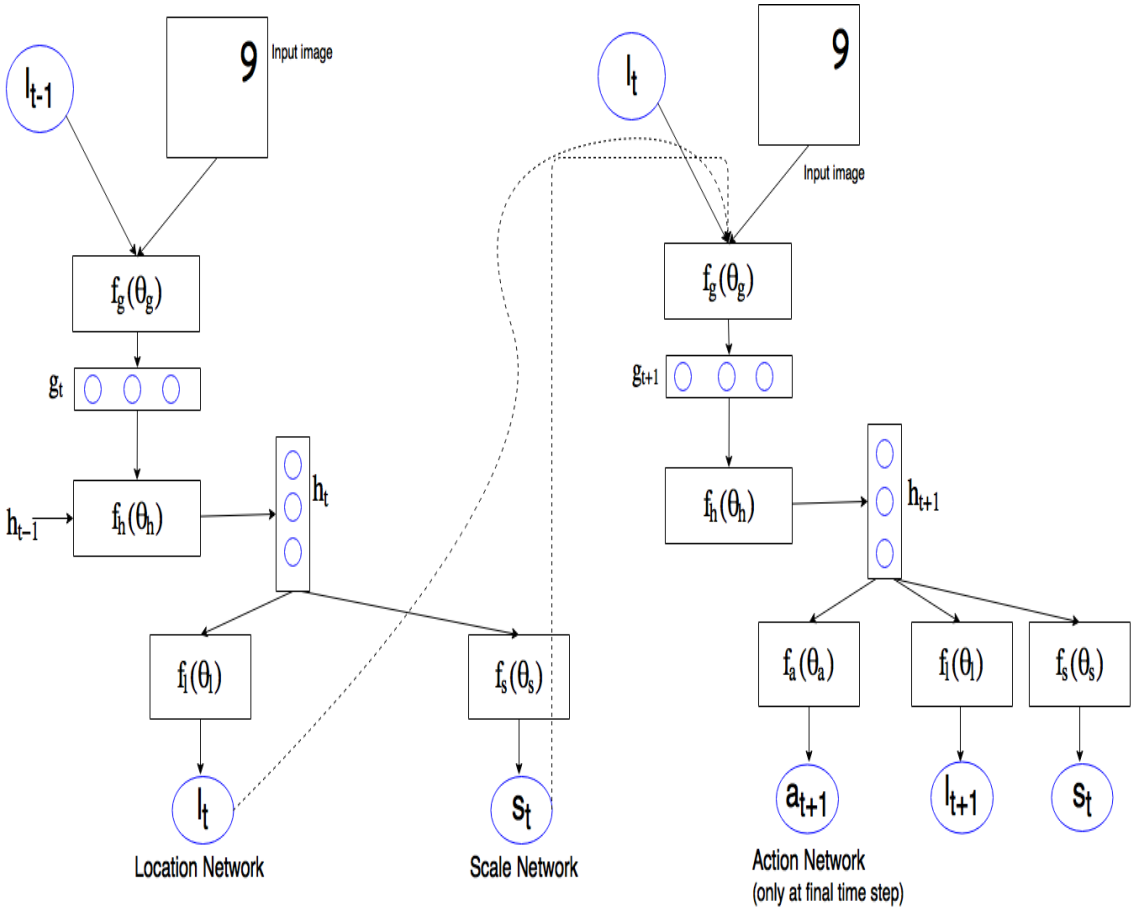

Figure 4.12: Scale learning Recurrent Attention Model: This model outputs a) next location b) scale of operation for glimpse depths, at each time step.

We propose an architecture as shown in Figure 4.12 for the model to predict next location as well as scale at each time step. In this architecture at each time step, the model feeds the state vector h_t to scale network and location network. Scale network during training phase of the algorithm stochastically draws a value from a gaussian distribution where the mean and variance of the distribution were hyper-parameters chosen by experimentation. Figure 4.13 shows the Normal distribution for the scale learning.

Table 4.5: Results obtained on Scaled MNIST dataset with scale learning model in comparison with constant scale model

Model configuration	Error (%)
Constant scale = 2.5, 4 Glimpses	13.7
Learnable scale, 4 Glimpses	12.27
Constant scale = 2.5, 5 Glimpses	10.85
Learnable scale, 5 Glimpses	10.02

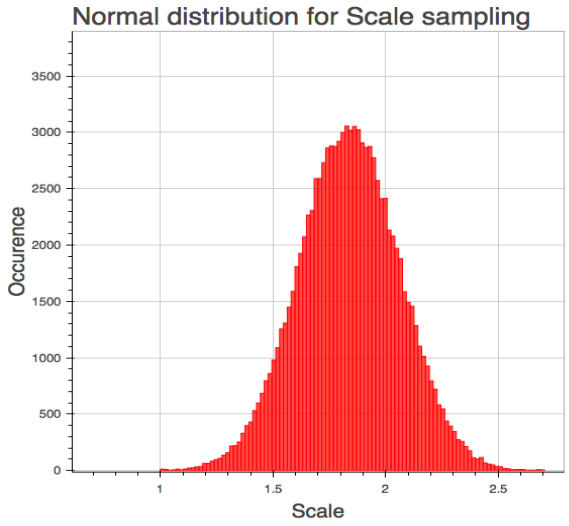

Figure 4.13: Normal distribution used for training Scale learning

To validate the performance of scale learning model, we performed experimentation on Scale MNIST as described in Section 4.4.1. The result of one of the samples can be seen in Figure 4.14.

To have a comparison with standard recurrent architecture, we ran the same experiment on Scale MNIST. Results obtained can be seen in Table 4.5. As observed from the results, scale learning model outperforms constant scale model with 1% average improvement in accuracy. Also, scale learning model takes less number of steps to make a classification decision compared to constant scale model.

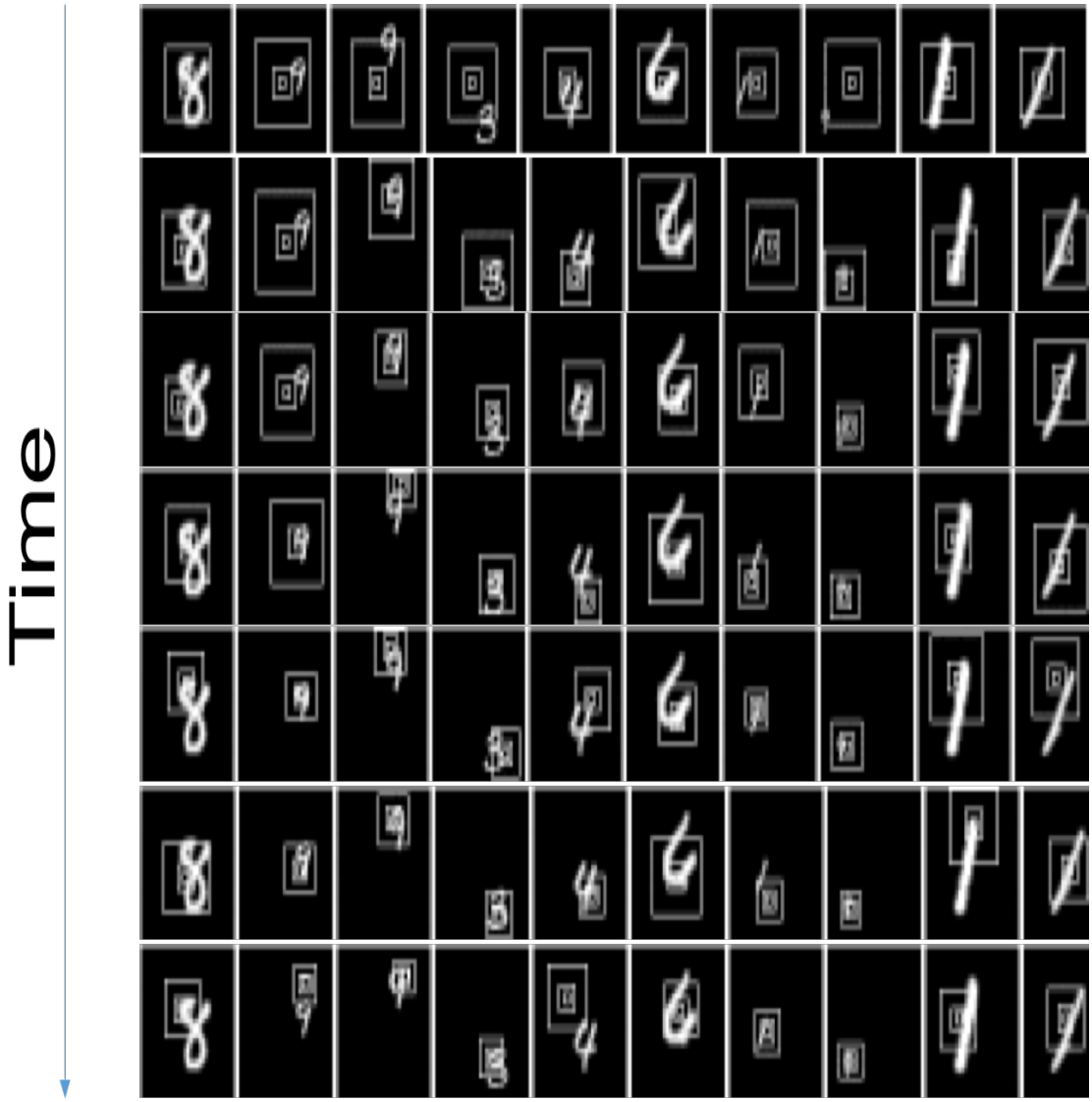

Figure 4.14: The figure shows how the glimpse location changes for time for different digits on Scaled MNIST dataset.

Chapter 5

Conclusion and future work

This work introduced the recurrent architecture for visual attention based on Recurrent Neural Networks and attentional models. The network learns to take glimpses with each subsequent glimpse location based on accumulated information from the past. The model generates class prediction at the last time step based on the information captured from the glimpses. This architecture is based on hard attention model, which means that the model makes a hard decision on what location to attend next. Due to hard attention mechanism, the model is non-differentiable and hence is trained using REINFORCE algorithm which is based on policy gradients. This architecture in comparison to Convolutional Neural Network's has less number of parameters and computations required is independent of the size of image. Also, since the model has a limited size sensor, it is able to avoid possible clutter in the environment and focus on the task at hand. A limitation of this model is that it assumes the task at hand to be of a uniform scale and based on this we chose the initial glimpse sensor size. But in real-world datasets, we do not have such freedom of knowing the scale of objects. This work proposes a modification to the recurrent architecture where the model is able to adjust the scale at which glimpses are taken and hence is able to capture information from more pixels if scale of the object is large or reduces the number of pixels if scale of the object is small. This model is tested on modified MNIST dataset where the scale of the digits is not the same and varies throughout the dataset. The model is able to adjust the scale based on the digit size and classify accordingly. This model outperforms the standard recurrent architecture model on this modified dataset.

Though the model performs well in the defined dataset, real challenge for this model would be to classify objects in varied datasets where the number of classes is large such as in ImageNet dataset. To perform well in these datasets there would be a requirement of the model to create better intermediate representations at each time step. Also, scale learning can also be more susceptible to noise and clutter than standard

recurrent architecture.

Bibliography

[1] A. Karpathy, "CS231n: Convolutional neural networks for visual recognition," 2016. vi, 5

[2] "Understanding, deriving and extending the lstm," 2016. vi, 6, 7

[3] K. Cho, A. Courville, and Y. Bengio, "Describing multimedia content using attention-based encoder-decoder networks," *IEEE Transactions on Multimedia*, vol. 17. vi, 2, 10, 11, 12

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012. 1

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. 1

[6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. 1

[7] R. A. Rensink, "The dynamic representation of scenes," *Visual cognition*, vol. 7, no. 1-3, pp. 17–42, 2000. 1

[8] M. Corbetta and G. L. Shulman, "Control of goal-directed and stimulus-driven attention in the brain," *Nature reviews neuroscience*, vol. 3, no. 3, pp. 201–215, 2002. 1

[9] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 11, pp. 1254–1259, 1998. 2

[10] T. Liu, Z. Yuan, J. Sun, J. Wang, N. Zheng, X. Tang, and H.-Y. Shum, "Learning to detect a salient object," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 33, no. 2, pp. 353–367, 2011. 2

[11] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, pp. I–511, IEEE, 2001. 2

[12] V. Mnih, N. Heess, A. Graves, *et al.*, "Recurrent models of visual attention," in *Advances in Neural Information Processing Systems*, pp. 2204–2212, 2014. 2, 16, 17, 21, 25, 27

[13] J. Ba, V. Mnih, and K. Kavukcuoglu, "Multiple object recognition with visual attention," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015. 2

[14] K. Xu, J. Ba, R. Kiros, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *arXiv preprint arXiv:1502.03044*, 2015. 2, 13

[15] I. G. Y. Bengio and A. Courville, "Deep learning." Book in preparation for MIT Press, 2016. 3, 7

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 8

[17] F. Wang and D. M. J. Tax, "Survey on the attention based RNN model and its applications in computer vision," *CoRR*, vol. abs/1601.06823, 2016. 12

[18] "Learning to link images with their description, year =." vi, 15

[19] M. Hayhoe and D. Ballard, "Eye movements in natural behavior," *Trends in cognitive sciences*, vol. 9, no. 4, pp. 188–194, 2005. 16

[20] A. M. Schäfer, S. Udluft, *et al.*, "Solving partially observable reinforcement learning problems with recurrent neural networks," in *Workshop Proc. of the European Conf. on Machine Learning*, pp. 71–81, 2005. 20

[21] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992. 23

[22] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits," 1998. 24