Domain Oriented Conversational Agent

A Project Report

submitted by

VAMSHI KUMAR KURVA

in partial fulfillment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

MATHEMATICS DEPARTMENT
INDIAN INSTITUTE OF SPACE SCIENCE AND TECHNOLOGY
Thiruvananthapuram - 695547

May 2017

CERTIFICATE

This is to certify that the thesis titled '**Domain Oriented Conversational Agent**', submitted by **Vamshi Kumar Kurva**, to the Indian Institute of Space Science and Technology, Thiruvananthapuram, for the award of the degree of **MASTER OF TECHNOLOGY**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr.Gorthi R. K. S. S. Manyam
Supervisor
Mathematics department
IIST

K.S S Moosath
Head of Department
Mathematics
IIST

Place: Thiruvananthapuram
May, 2017

DECLARATION

I declare that this thesis titled '**Domain Oriented Conversational Agent**' submitted in fulfillment of the Degree of MASTER OF TECHNOLOGY is a record of original work carried out by me under the supervision of **Dr.Gorthi R. K. S. S. Manyam**, and has not formed the basis for the award of any degree, diploma, associateship, fellowship or other titles in this or any other Institution or University of higher learning. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Vamshi Kumar Kurva
SC15M045

Place: Thiruvananthapuram
May, 2017

Abstract

Sequence learning is a study of machine learning algorithms that deals with the sequential data. A conversational agent is one of those which has to deal with the sequential data. It is a program that can make natural conversations indistinguishable from the humans. Conversational modeling is one of the important tasks in machine learning and natural language processing. Since the conversation is interactive, model should be able to process sequential inputs and should be able to generate sequential outputs.This project focuses on developing a domain-oriented question-answer system where user will ask questions related to a specific topic or domain.

RNNs with their cyclical connections have been proved to capture the dynamics of these sequential data. RNNs for this very reason, have been used for many sequence learning tasks. However, vanilla or regular RNNs have the problem of training difficulty, but the advances in optimization techniques and and network architectures have paved a way for large scale learning. With the invention of LSTM and other sophisticated network architectures, tremendous results have been achieved on tasks like machine translation, image captioning and handwritten digit recognition. This thesis explores the obvious and not-so-obvious challenges in sequence learning tasks. We also explore different network architectures and learning algorithms along with experimental analysis on different standard datasets. All the experiments related to sequential learning are done on text data.

Acknowledgements

Though only my name appears on the cover, the work done in this thesis wouldn't have been possible without the help of many people. I would like to thank my guide Dr.Gorthi R. K. S. S. Manyam for his constant support throughout the project and for his valuable suggestions and feedbacks. I also would like to thank him for reviewing the thesis and making it to the present form. I thank him especially for showing me the way, whenever I got stuck in the project.

I would like to express my gratitude to Dr. Sumitra S, who taught me the mathematical concepts of datamining and pattern recognition. I'm indebted to her for all the knowledge she imparted to me. I'm thankful to Sanjeev Kumar P.S (my manager at Philips Lighting India Ltd), who believed in me and assigned me such an important research topic for a beginner like me.

I would like to thank the faculty of Mathematics Department, IIST. Special thanks to my guide Pavan Kumar Reddy Sannadi and my team mate Pragya Shrivastava. I also would like to thank my friends at IIST and Philips lighting. I'm thankful to my mother and sister for their constant support and love throughout my life. I would like to thank each and everyone, who helped me directly or indirectly in finishing the thesis.

Contents

List of Figures

List of Tables

Abbreviations

HMM	Hidden Markov Model
DNN	Deep Neural Networks
RNN	Recurrent Neural Networks
FFNN	Feed Forward Neural Network
LSTM	Long Short Term Memory
GRU	Gated Recurrent Unit
BPTT	Back Propagation Through Time
SMT	Statistical Machine Translation
SNE	Stochastic Neighbor Embedding
tSNE	t-distributed Stochastic Neighbor Embedding
BLEU	Bi Lingual Evaluation Understudy
NLP	Natural Language Processing
Seq2Seq	Sequence-to-Sequence

Chapter 1

Introduction

Sequence learning is one of the important areas in machine learning. There are many machine learning tasks that deal with sequential data. Music generation, question answering, machine translation, Forecasting ; all these systems require a model that produces a sequence of outputs. All these systems need to process a series of inputs at times. Since the sequential data is highly correlated, we need machine learning algorithms that can capture these correlations and those that do not assume that the data points are independent.

1.1 Literature Review

Many learning problems involves and deals with sequential data. Most of the machine learning algorithms are designed for identical and independently distributed (i.i.d) data. *Independent and identically distributed implies an element in the sequence is independent of the random variables that came before it* (Wikipedia, 2017). However all the data is not i.i.d. Dependent data occurs whenever correlation occurs between observations (Darrell et al., 2015). For e.g. successive points in a sequential data are highly correlated. So, the assumption that data is i.i.d has to be relaxed sometimes to take these correlations into account. *Taking these correlations into account leads to improved accuracy of the learning algorithms* (Dundar et al., 2007).

Any sequence which is not iid can be considered as Markov sequence, since markov

chain models the transition between states in an observed sequence. Hidden Markov Models(HMM) were widely studied in 1960s. However these approaches are limited because that state has to be sampled from a discrete state of size S. The dynamic program that is used for inference with HMMs have time complexity of $\mathcal{O}(|S|^2)$ (Viterbi, 1967) and the transition table indicating the probabilities of transition is also of size $|S|^2$. The complexity increases as the number of states increases. Further more, HMMs model that each hidden state depends only on the previous state. Even though this can be dealt by considering a big context window, it can lead to a new state space which is equal to the cross product of all possible states in the context window (Lipton et al., 2015). This becomes more complex as the window size increases. *This makes the Markov models impractical for modeling long-term dependencies* (Graves et al., 2014).

Neural networks are a class of learning models that achieved state-of-the-art performance on many supervised and unsupervised learning problems. Deep Neural networks(DNN) have been applied on difficult tasks like image recognition and speech recognition (Krizhevsky et al. (2012); Hinton et al. (2012)) etc and have achieved excellent performance. *They have the ability to learn high-level features in a hierarchical, layer-wise fashion, which is the reason for their success.* Despite their power and ability to learn complex features, DNNs can't be applied for sequence learning tasks (Sutskever et al., 2014) . This is because standard DNNs rely on examples that are fixed in length and and assumes the data is i.i.d. Hence they are not suitable for sequence learning problems like machine translation, question-answering where the lengths of input and output is not fixed.

Recurrent Neural Networks(RNN) are a class of Neural networks with the cyclical connections, which have the ability to capture temporal dependencies. *Theoretically, An RNN with enough number of hidden neurons, in principle can generate any sequence because of their turing capabilities* (Siegelmann and Sontag, 1991). However, their ability to store the information of the past inputs diminishes, because of the diminishing backward flow of gradients (Bengio et al., 1994). This problem motivated the invention of Long Short Term Memory(LSTM), which is a variant of RNN (Hochreiter and Schmidhuber, 1997). LSTMs have been proved to

be effective in capturing the long-term dependencies because of the special gating units introduced in the cell structure. Over the years, several variants of LSTM have been proposed and applied on the sequence learning problems achieving state-of-the-art results.

There have been many attempts to map a sequence of inputs to a sequence of outputs using neural networks. All the end-to-end models proposed for sequence to sequence mapping tasks consists of an encoder and a decoder network(Sutskever et al. (2014); Kalchbrenner et al. (2014)). These approaches have been referred as 'Neural machine Translation' by Cho et al. (2014a). Encoder converts the source sequence into a fixed-size vector and decoder generates the varible-length target sequence using this vector. Several other models have been proposed inspired by the encoder-decoder approach to machine translation. This thesis focuses on several variants of the encoder-decoder architectures proposed and the effects of these on different datasets. This thesis also provides an experimental study of these architectures on a conversational dataset.

1.2 Thesis outline

The contents of this thesis are organized as six chapters.

Chapter 2 provides a brief description of RNN, BPTT and vanishing and exploding gradients problem

Chapter 3 discusses about the LSTM, a variant of RNN, GRU cell and how it addresses the vanishing gradients problem

Chapter 4 gives an idea about different end-to-end sequence-to-sequence (Seq2Seq) mapping models and their variants.

Chapter 5 is devoted to the experimental analysis and results of applying the models on different datasets

Chapter 6 gives the conclusion of this thesis work along with the future scope and areas of improvements.

1.3 Contributions

The contributions of this thesis are summarized as follows.

- This thesis provides a survey of the various seq2seq models that are available and discusses their performances
- The Seq2Seq models that are studied are applied on three different datasets; cornell movie subtitles, Philips amplight dataset and TIDES-IIIT English-Hindi parallel corpus.
- The improvement in the performance and the usefulness of applying various seq2seq models are provided with the aid of graphs and visualizations.

Chapter 2

RNN and its training difficulty

This chapter discusses about RNNs and how the gradients are back propagated through time and the difficulty in training them. Even though many variants of RNNs like echo state networks, Bidirectional RNNs, Jordan networks, time-delay networks have been proposed in recent years, Here we focus on basic RNNs.

2.1 RNN

RNNs are neural networks with cyclical connections. Because of these recurrent connections, RNNs have a 'memory' of previous inputs in the network's internal state, and thereby influences the network output. RNN can be thought of as a Feed Forward Neural Network(FFNN) unfolded in time space (see figure 2.1). Forward pass is same as that of a FFNN, except for the fact that activations arrive the hidden layer from both current external input and the hidden layer activations from the previous time step.

2.1.1 Forward pass

Consider a length T input sequence x presented to an RNN with I input units, H hidden units, and K output units. Let x_i^t be the value of input i at time t, and let a_j^t and b_j^t be respectively the network input to unit j at time t and the activation

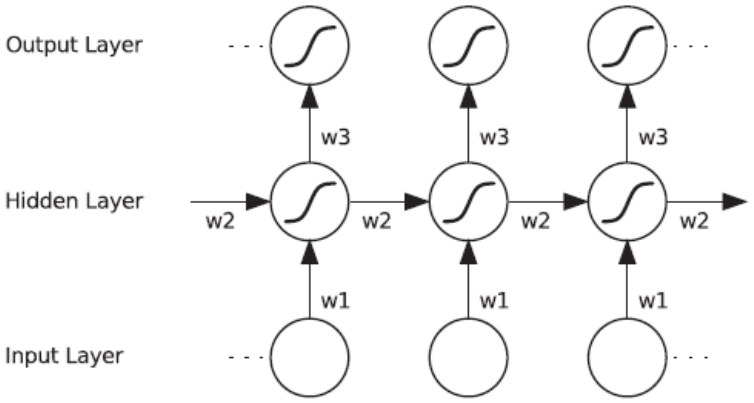

FIGURE 2.1: An unfolded recurrent network (reprinted from Graves (2012))

of unit j at time t. Let f be the activation function.

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1} \tag{2.1}$$

$$b_h^t = f(a_h^t) \Rightarrow \frac{\partial b_h^t}{\partial a_h^t} = f^{'}(a_h^t) \tag{2.2}$$

where $W_1 = \{w_{ih}\}$ is the input-hidden weight matrix and $W_2 = \{w_{h'h}\}$ is the hidden-hidden weight matrix and $W_3 = \{w_{hk}\}$ is the hidden-output matrix

2.1.2 Backward Propagation Through Time

Since the network is unfolded in time space, gradients has to flow backwards through time as well as layers (see figure 2.2). BPTT is a technique to find the gradients in RNN, where Back Propagation is applied on the unrolled network (Rumelhart et al., 1985). Total loss/error on a given task is the sum of per-time loss

$$L(z, y) = \sum_{t=1}^{T} L_t = \sum_{t=1}^{T} L(z_t, y_t) \tag{2.3}$$

where, L_t is the error at time step t

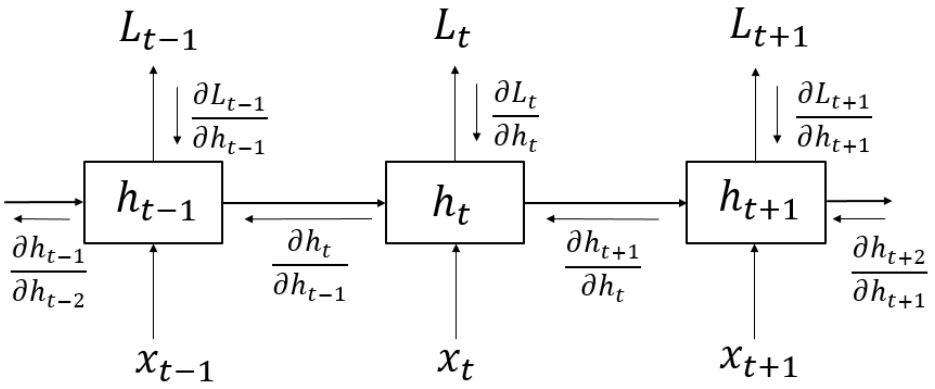

FIGURE 2.2: Illustration of how gradients back propagate through time as well as layers

Consider

$$\begin{aligned}\delta_h^t &= \frac{\partial L}{\partial a_h^t} \\ &= \frac{\partial L}{\partial b_h^t} \cdot \frac{\partial b_h^t}{\partial a_h^t} \\ &= \frac{\partial L}{\partial b_h^t} \cdot f^{'}(a_h^t)\end{aligned} \tag{2.4}$$

output of the neuron at time t, b_h^t not only affects the output layer, but also the hidden layer at the next time step t+1. Hence

$$\begin{aligned}\frac{\partial L}{\partial b_h^t} &= \sum_{k=1}^{K} \frac{\partial L}{\partial a_k^t} \cdot \frac{\partial a_k}{\partial b_h^t} + \sum_{h'=1}^{H} \frac{\partial L}{\partial b_h^{t+1}} \cdot \frac{\partial b_h^{t+1}}{\partial b_h^t} \\ &= \sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \frac{\partial L}{\partial b_h^{t+1}} \cdot \frac{\partial b_h^{t+1}}{\partial a_h^{t+1}} \cdot \frac{\partial a_h^{t+1}}{\partial b_h^t} \\ &= \sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \frac{\partial L}{\partial b_h^{t+1}} f^{'}(a_h^{t+1}) w_{hh'} \\ &= \sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'}\end{aligned} \tag{2.5}$$

By substituting eq. 2.5 in eq. 2.4

$$\delta_h^t = f^1(a_h^t)(\sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'} + \sum_{k=1}^{K} \delta_k^t w_{hk}) \tag{2.6}$$

In general, when we have more than one hidden layer, for the unit h in the l^{th} layer H_l

$$\delta_h^t = f^{'}(a_h^t)\ (\sum_{h' \in H_{l+1}} \delta_{h'}^t w_{hh'} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'}) \tag{2.7}$$

This can be thought of as applying the back propagation applied to a FFNN in which target values are specified for every layer not just the last (Williams and Peng, 1990). Since the network weights are reused at every time step, take the average or sum of weights to get the final derivatives

$$\frac{\partial L}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial L}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t b_i^t \tag{2.8}$$

The δ terms can be calculated by starting at t=T and by applying eq (2.7) repeatedly decrementing t at each step. $\delta_j^{T+1} = 0, \forall j$, since after time T there is no error.

2.2 Vanishing and Exploding Gradients Problem

Training RNNs is difficult because of the problem known as 'Vanishing and Exploding gradients' mentioned in Bengio et al. (1994). This problem makes the it hard to learn and tune the parameters of the model, especially when using the gradient-based methods. Intuitively, this problem occurs because of the activation function used in the network. Activation functions squash their inputs to a small range([-1, 1] or [0,1]) in some non-linear way. Therefore, there are large regions of input space mapped to a small space. In these input regions, a large change in the input value produces only a small change in the output. i.e. gradients are very small. As the gradients back propagate through layers, they are squashed more and more, which makes it more difficult to train the lower layers of the network when compared to the higher layers (Glorot and Bengio, 2010). Figure 2.3 shows

the sensitivity of input over time in an RNN. Here we try to explain this problem in mathematical terms.

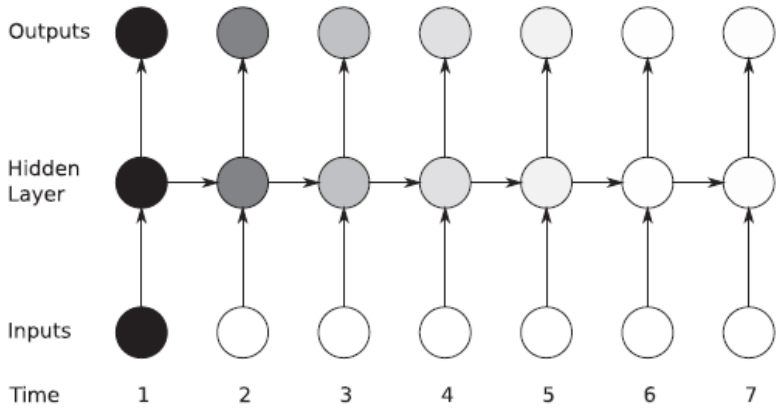

FIGURE 2.3: Sensitivity of inputs over time in RNN (reprinted from Graves (2012))

Let W_{in} be the weight matrix between input-hidden layer and W_{Rec} is the matrix between hidden-hidden layer of a recurrent neural network with I input neurons and H hidden neurons. Let x_t be the input vector at time t and h_t be the hidden state vector at time t, then

$$h_t = \mathbf{W}_{in}\ x_t + \mathbf{W}_{rec}\ f(h_{t-1}) \tag{2.9}$$

where, f is the activation function. let L is the Loss function given by eq. (2.3). Let θ represent the weight in general then

$$\frac{\partial L}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \theta} \tag{2.10}$$

where L_t is the loss at step t.

$$\frac{\partial L_t}{\partial \theta} = \sum_{1 \le k \le t} \frac{\partial L_t}{\partial h_t}\ \frac{\partial h_t}{\partial h_k}\ \frac{\partial h_k}{\partial \theta} \tag{2.11}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{t \ge i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \ge i > k} W_{rec}^{T}\ diag(f^{'}(h_{i-1}) \tag{2.12}$$

Eq. (2.12) is a form of Jacobian matrix for the parametrization given by eq. (2.9), *diag* converts a vector to a diagonal matrix and $f^{'}$ gives the element-wise derivative of f. The gradient component $\frac{\partial L_t}{\partial \theta}$ is a sum of temporal components, where each temporal component $\left(\frac{\partial L_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial \theta}\right)$ measures how θ at step k affects the error at step $t > k$. The term $\frac{\partial h_t}{\partial h_k}$ transport the error from step t back to step k. Further, the terms for which $k << t$ are long term contributions and short term refers to everything else.

The factor $\frac{\partial h_t}{\partial h_k}$ is a product of **t-k** Jacobian matrices. *These product of **t-k** matrices can either reduces to zero or explodes to infinity (in some direction), just like a product of **t-k** real numbers* (Pascanu et al., 2013). Let us consider the linear version of the model (activation function f is the identity function and $f^{'} = 1$). In this case $\frac{\partial h_t}{\partial h_k} = (W_{rec}^T)^{t-k}$. It is sufficient for $\rho < 1$, where ρ is the spectral radius of the recurrent weight matrix W_{rec} , for long-term components (terms for which $(t-k) \to \infty$) to vanish and necessary for $\rho > 1$ to explode.

Let us consider the more general case, where activation function f is non-linear, like *tanh* or *sigmoid* for which $|f^{'}(x)|$ is bounded and let $\|diag(f^{'}(h_k))\| \leq \gamma \in \mathbb{R}$. Consider the 2-norm of the Jacobian $\frac{\partial h_{k+1}}{\partial h_k} = W_{rec}^T \; diag(f^{'}(h_k))$.

$$\left\|\frac{\partial h_{k+1}}{\partial h_k}\right\| \;\leq\; \left\|W_{rec}^T\right\| \left\|diag(f^{'}(h_k))\right\| \tag{2.13}$$

Let us assume that the largest singular value of W_{rec} is λ and let $\lambda < \frac{1}{\gamma}$. Since the 2-norm of the matrix is equal to the largest singular value $\left\|W_{rec}^T\right\| = \lambda < \frac{1}{\gamma}$.

$$\left\|\frac{\partial h_{k+1}}{\partial h_k}\right\| \;\leq\; \left\|W_{rec}^T\right\| \left\|diag(f^{'}(h_k))\right\| \;<\; \frac{1}{\gamma}.\gamma \;<\; 1 \tag{2.14}$$

Let $\eta \in \mathbb{R}$ such that $\forall k, \frac{\partial h_{k+1}}{\partial h_k} < \eta < 1$. Then

$$\left\| \frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial x_k} \right\| = \left\| \frac{\partial L_t}{\partial h_t} \left(\prod_{i=k}^{t-1} \frac{\partial h_{i+1}}{\partial h_i} \right) \right\| \leq \eta^{t-k} \left\| \frac{\partial L_t}{\partial h_t} \right\| \tag{2.15}$$

Since $\eta < 1$, long-term contributions (terms for which t-k is large) go to zero exponentially with t-k. So it is sufficient for $\lambda < \frac{1}{\gamma}$ for vanishing gradients problem to occur. Similarly, it is necessary for $\lambda > \frac{1}{\gamma}$ for the gradients to explode.
If the activation function is $f(x) = \tanh(x)$, then $f'(x) = 1 - \tanh^2(x)$ and $|f'(x)| \leq 1$ so $\gamma = 1$. If $f(x)$ is sigmoid then $f'(x) = f(x)(1-f(x))$ and $|f'(x)| < \frac{1}{4}$ so $\gamma = \frac{1}{4}$ (see figure 2.4).

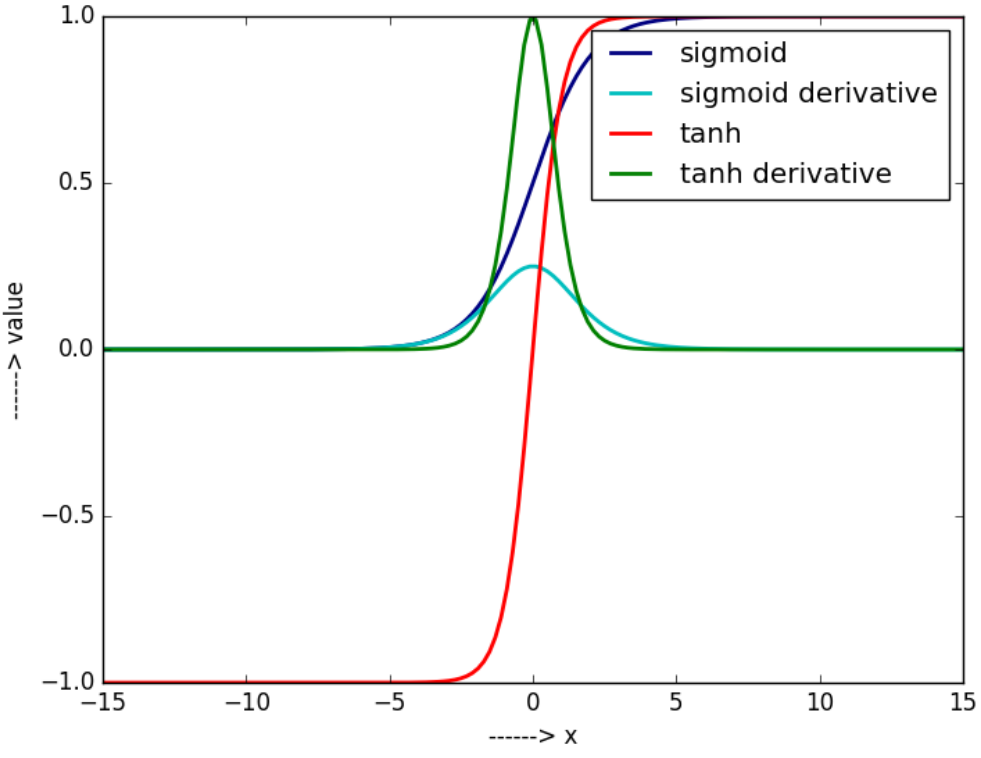

FIGURE 2.4: Sigmoid, tanh along with their derivatives

2.3 Truncated BPTT

The problem with the BPTT algorithm presented above is the cost of computing gradients of sequence length T is same as the cost of a FFNN with T layers. i.e. the cost of updating a single parameter increases as T increases. We have to save all the hidden and output values of RNN for all the T time steps. One more problem with high value of T is vanishing gradients. Truncated BPTT is proposed by Williams and Peng (1990). It proposes to store only the recent k_2 hidden and output values. i.e. weight changes are made considering only the recent k_2 time steps from the current time step. This is referred to as BPTT(k_2) by Williams and Peng (1990). The BPTT(k_2) at time t calculates the weight gradient as

$$\frac{\partial L}{\partial w_{ij}} = \sum_{\tau=t-k_2+1}^{t} \frac{\partial L}{\partial a_j^\tau} \frac{\partial a_j^\tau}{\partial w_{ij}} = \sum_{\tau=t-k_2+1}^{t} \delta_j^\tau b_i^\tau \tag{2.16}$$

$$\delta_h^\tau = \begin{cases} f^{'}(a_h^\tau) \left(\sum_{h' \in H_{l+1}} \delta_{h'}^\tau w_{hh'} \right) & \text{if } \tau = t \\ f^{'}(a_h^\tau) \left(\sum_{h'=1}^{H} \delta_{h'}^{\tau+1} w_{hh'} \right) & \text{otherwise} \end{cases} \tag{2.17}$$

The error is considered to be injected only at the most recent step, not for any earlier time steps and the delta terms for which $\tau \leq t - k_2$ are considered as zero. For this very reason the relevant information is stored for only fixed k_2 steps. Any information older than k_2 steps is forgotten. This is considered to be an approximation to Full BPTT and a heuristic technique to simplify the computation. In BPTT(k_2) "Backward pass is computed through most recent k_2 steps each time the network is run through an additional time step". Generalization of this idea is to perform BPTT every k_1 steps instead of every time step where $k_1 \leq k_2$. This is referred as BPTT($k_2; k_1$). It uses following equations along with eq (2.16).

$$\delta_h^\tau = \begin{cases} f'(a_h^\tau)\left(\sum_{h' \in H_{l+1}} \delta_{h'}^\tau w_{hh'}\right) & \text{if } \tau = t \\ f'(a_h^\tau)\left(\sum_{h' \in H_{l+1}} \delta_{h'}^\tau w_{hh'} + \sum_{h'=1}^{H} \delta_{h'}^{\tau+1} w_{hh'}\right) & \text{if } t - k_1 < \tau < t \\ f'(a_h^\tau)\left(\sum_{h'=1}^{H} \delta_{h'}^{\tau+1} w_{hh'}\right) & \text{if } t - k_2 < \tau \leq t - k_1 \end{cases} \tag{2.18}$$

The key feature here is that back propagation is done only for every k_1 time steps. So, BPTT(k_2) is same as BPTT($k_2; 1$) and BPTT($k1; k1$) with $k1 = T$ is full BPTT.

Algorithm:

1. **for** t **from** 1 **to** T **do**
2. Run RNN for one step, compute h_t, z_t
3. **if** t % $k_1 == 0$
4. Run BPTT from t to $t - k_2$
5. **endif**
6. **endfor**

If k_2 is small, then the computations are cheaper but the model will not be able to capture dependencies that span more than k_2 time steps.

2.4 Simple solutions to vanishing and exploding gradients

Here we give two simple solutions to deal with vanishing and exploding gradients problem

2.4.1 Scaling Down the Gradients

One of the ways to deal with the problem of exploding gradients is, to rescale the gradients if the norm exceeds a particular value.

Algorithm:

1. Find the gradient $\tilde{w} = \frac{\partial L}{\partial w}$

2. if $\|\tilde{w}\| > threshold$, then

   $\tilde{w} = \frac{threshold}{\|\tilde{w}\|}\tilde{w}$

   endif

This method is also called as Gradient Clipping. It is simple and efficient. However, it brings an extra hyper-parameter *threshold* (Pascanu et al., 2013).

2.4.2 Change the activation function

Consider a simple neural network, where error E depends on weight w_{ij} only through y_j, where

$$y_j = f(\sum_i w_{ij}x_i) \tag{2.19}$$

then the gradient is

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \frac{\partial E}{\partial y_j}\ \frac{\partial y_j}{\partial w_{ij}} \\ &= \frac{\partial E}{\partial y_j}\ f'\big(\sum_i w_{ij}x_i\big)\ x_i \end{aligned} \tag{2.20}$$

If f is *sigmoid*, then f' will be close to zero for very small inputs as well as large inputs. So gradients will be very small and the gradients will not propagate well. i.e. cell has reached the saturation (It's output is close to zero or one) (Glorot and Bengio, 2010). If f is *tanh*, gradients shrink slower when compared to using *sigmoid* (see figure 2.4). If f is a rectified Linear unit, then

$$f(x) = \max(0, x) \tag{2.21}$$

then, it's gradient is zero only for inputs less than zero and 1 for all the others. Hence ReLU activation function doesn't have this problem. However ReLU units have a problem. They output zero for negative values. So, if there is a large negative bias in our network somewhere, there is a chance that the gradients are zero and the network will not learn anything. ReLUs can never recover from this problem, because they can't change the weights in anyway due to zero gradients. So, in case we use ReLU function, it is better to initialize all the weights to positive. ReLU is used most commonly used in Convolution Neural Networks (CNN).

Chapter 3

Long Short Term Memory

Due to the Vanishing and exploding problem described above, it is difficult to train the RNNs. Because of this reason, RNNs are not good at capturing long-term temporal dependencies. LSTM(Long Short Term Memory)s are a version of RNN, with specific architecture proved to be effective in learning long-range time dependencies. This chapter introduces us to LSTM and a variant of it known as GRU (Gated Recurrent Unit) cell along with the back propagation details.

3.1 LSTM

LSTM architecture was motivated by the error propagation problem in RNNs. The main idea behind LSTM architecture is a cell which maintains the cell state over time, and it is controlled by a non-linear multiplicative units known as gates. These gates regulate the flow of information into and out of the cell. Many versions of LSTMs have been proposed since its inception in 1997 (Hochreiter and Schmidhuber, 1997). Here we are explaining the most popular variant of LSTM known as Vanilla LSTM. It has three gates known as input, forget and output gates which are analogous of write, reset and read for the cells. It also has a block input. Figure 3.1 shows the architecture of vanilla LSTM. The original LSTM proposed in 1997 didn't have forget gate and peep-hole connections. These were added later to the architecture, to make the learning of timings easier (Greff et al., 2016).

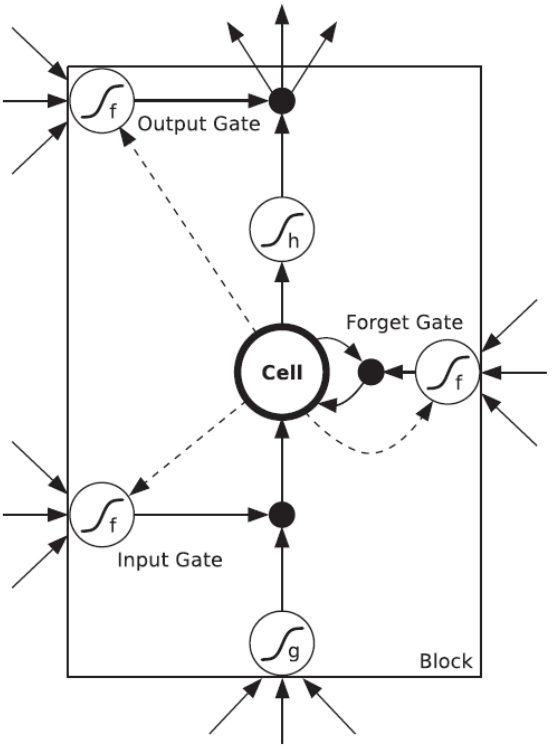

FIGURE 3.1: LSTM memory block with one cell. Solid lines indicates unweighted connections, while dashed lines indicate the weighted peep-hole connections (delayed by one time step) (reprinted from Graves (2012))

The multiplicative gates allow LSTM memory cells to store and access information over long periods of time. For example, as long as the input gate remains closed (i.e. has an activation near 0), cell state will not be overwritten by the new inputs arriving in the network, and can be made available to the network much later in the sequence, by opening the output gate.

Consider an LSTM network with I input units, H hidden units and K output units, then total number of parameters of the network $= I*H*4+H*H*4+H*K+H*3$. The number of parameters are more compared to a regular RNN, because of additional parameters introduced by the gates.

Input gate multiplies the input of the cell and output gate multiplies the output of the cell. Forget gate multiplies the previous cell state. No activation function is applied within the cell. The gate activation function 'f' is usually the sigmoid, to keep the activations between 0 and 1. cell input and output activation functions ('g' and 'h') are usually tanh or sigmoid. In some cases 'h' can be an identity

function. Figure 3.2 shows an LSTM network with two cells

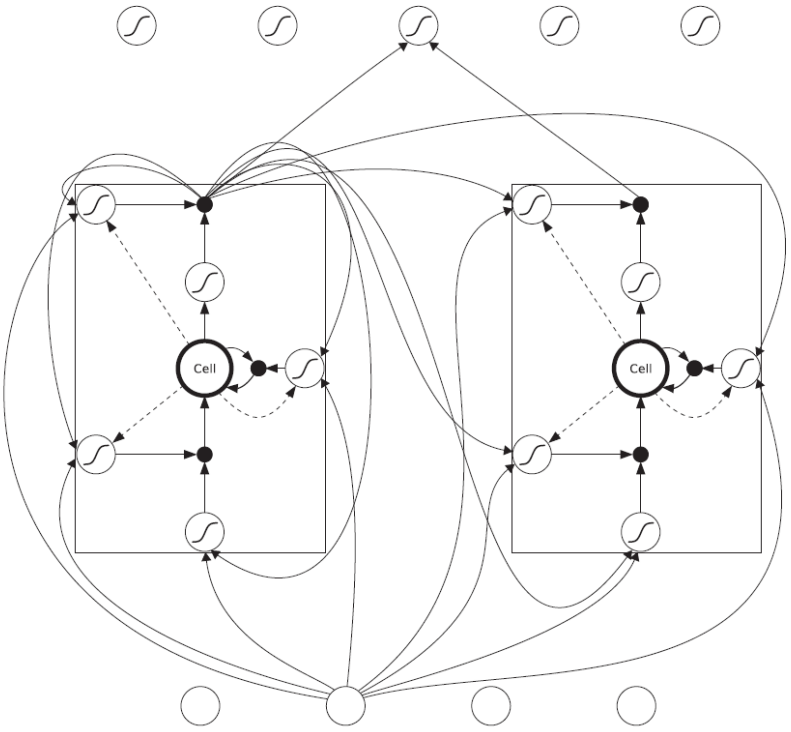

FIGURE 3.2: **An LSTM network** The network consists of four input units, a hidden layer of two single-cell LSTM memory blocks and five output units. Not all connections are shown. (reprinted from Graves (2012))

3.1.1 Forward pass

Let I be the number of inputs and H be the number of LSTM cells. Let i_t, f_t, o_t respectively are the outputs of input, forget and output gates at time t and $c_t$is the cell state (all are vectors of size H). Outputs of all the gates are calculated as follows.

$$
\begin{aligned}
\tilde{z_t} &= W_{zx}x_t + W_{zm}m_{t-1} + b_z && \text{net block input} \\
z_t &= g(\tilde{z_t}) && \text{block input} \\
\tilde{i_t} &= W_{ix}x_t + W_{im}m_{t-1} + W_{ic}c_{t-1} + b_i && \text{net input to input gate} \\
i_t &= \sigma(\tilde{i_t}) && \text{output of input gate} \\
\tilde{f_t} &= W_{fx}x_t + W_{fm}m_{t-1} + W_{fc}c_{t-1} + b_f && \text{net input to forget gate} \\
f_t &= \sigma(\tilde{f_t}) && \text{output of forget gate} \\
c_t &= f_t \odot c_{t-1} + i_t \odot z_t && \text{new cell state} \\
\tilde{o_t} = o_t &= W_{ox}x_t + W_{om}m_{t-1} + W_{oc}c_{t-1} + b_o && \text{net input to output gate} \\
o_t &= \sigma(\tilde{o_t}) && \text{output of output gate} \\
m_t &= o_t \odot h(c_t) && \text{block output}
\end{aligned}
\tag{3.1}
$$

where,

- $W_{zx}, W_{ix}, W_{fx}, W_{ox} \in \mathbb{R}^{IxH}$ are input weights
- $W_{zm}, W_{im}, W_{fm}, W_{om} \in \mathbb{R}^{HxH}$ are recurrent weights
- $W_{ic}, W_{fc}, W_{oc} \in \mathbb{R}^{HxH}$ are peep-hole weights(diagonal)
- $b_z, b_i, b_f, b_o \in \mathbb{R}^{H}$ are bias weights

Since each hidden unit has separate input and forget gates, each hidden unit will learn to capture dependencies over different time scales.

3.1.2 Backward pass

Let Δ_t is the vector of deltas passed down from the output layer to the hidden layer. Let E be the loss function. Then, the deltas inside LSTM cell are calculated

as follows. These equations are mentioned in Greff et al. (2016).

$$
\begin{aligned}
m_t &= \Delta_t + W_{zm}^T \delta z_{t+1} + W_{im}^T \delta i_{t+1} + W_{fm}^T \delta f_{t+1} + W_{om}^T \delta o_{t+1} \\
\delta o_t &= \delta m_t \odot h(c_t) \odot \sigma^{'}(\tilde{o_t}) \\
\delta c_t &= \delta y_t \odot o_t \odot h^{'}(c_t) + W_{oc}\ \delta o_t + W_{ic}\ \delta i_{t+1} + W_{fc}\ \delta f_{t+1} + \delta c_{t+1} \odot f_{t+1} \\
\delta f_t &= \delta c_t \odot c_{t-1} \odot \sigma^{'}(\tilde{f_t}) \\
\delta i_t &= \delta c_t \odot z_t \odot \sigma^{'}(\tilde{i_t}) \\
\delta z_t &= \delta c_t \odot i_t \odot g^{'}(\tilde{z_t}) \\
\delta x_t &= W_{zx} \delta z_t + W_{ix} \delta i_t + W_{fx} \delta f_t + W_{ox} \delta o_t
\end{aligned}
\tag{3.2}
$$

The gradients for the weights are calculated as follows. Here $\star$ can be any of the $\{z, i, f, o\}$

$$
\begin{aligned}
\delta W_{\star x} &= \sum_{t=0}^{T} \langle \delta \star_t, x_t \rangle & \delta W_{ic} &= diag\big(\sum_{t=0}^{T} c_t \odot \delta i_{t+1}\big) \\
\delta W_{\star m} &= \sum_{t=0}^{T} \langle \delta \star_t, m_t \rangle & \delta W_{fc} &= diag\big(\sum_{t=0}^{T} c_t \odot \delta f_{t+1}\big) \\
\delta b_{\star} &= \sum_{t=0}^{T} \delta \star_t & \delta W_{oc} &= diag\big(\sum_{t=0}^{T} c_t \odot \delta o_t\big)
\end{aligned}
\tag{3.3}
$$

Within the LSTM cell, the gradient of cell state with respect to it's immediate predecessor, i.e. $\frac{\partial c_t}{\partial c_{t-1}}$ is the only term through which gradients flow back in time (Bayer, 2015).

$$
\begin{aligned}
\frac{\partial c_t}{\partial c_{t-1}} &= \frac{\partial(\sigma(\tilde{f_t}) \odot c_{t-1})}{\partial c_{t-1}} \\
&= diag(\sigma(\tilde{f_t}))
\end{aligned}
\tag{3.4}
$$

for $t^{'} > t$

$$
\frac{\partial c_{t'}}{\partial c_t} = \prod_{t' \geq i > t} \frac{\partial c_i}{\partial c_{i-1}} = \prod_{t' \geq i > t} diag(\sigma(\tilde{f_i}))
\tag{3.5}
$$

While this term can approach zero, there is no specific factor in it (i.e. some weight) which will drive the derivative to zero for very large time lags, i.e. $t^{'} >> t$. This part is also safe from exploding, since each of the factors is it is bounded by 1 due

to the activation function *sigmoid*. Figure 3.3 shows how LSTM can be able to store the gradients over time.T

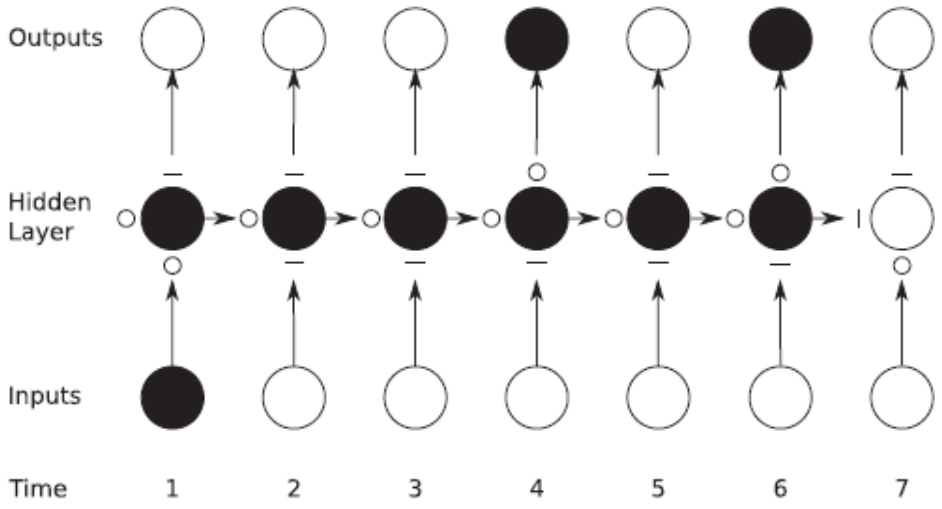

FIGURE 3.3: Illustration of how LSTM can preserve the gradients over time. (reprinted from Graves (2012))

3.2 Gated Recurrent Unit(GRU) cell

GRU cell was motivated by LSTM, and was first proposed by Cho et al. (2014b). It is a simplification of LSTM architecture. It has neither peep-hole connections nor output activations. Input gate and forget gate are coupled into a single gate named as update gate .It is much easier and simpler to implement compared to LSTM. The reset gate of the j^{th} hidden unit is computed as follows.

$$r_t^j = \sigma([W_r x_t]_j + [U_r h_{t-1}]_j) \tag{3.6}$$

where, $[\ ]_j$ denotes the j^{th} element of a vector. σ is the sigmoid activation function. x_t is the current input at time t and h_{t-1} is the previous hidden state. Similarly update gate is given by

$$z_t^j = \sigma([W_z x_t]_j + [U_z h_{t-1}]_j) \tag{3.7}$$

The candidate activation $\tilde{h}_t^j$ is calculated as follows.

$$\tilde{h}_t^j = \tanh([Wx_t]_j + [U\ (r_t \odot h_{t-1})]_j) \tag{3.8}$$

where r_t is a vector of reset gates and $\odot$ indicates element-wise multiplication. Figure 3.4 shows how update and reset gate effects the hidden state.

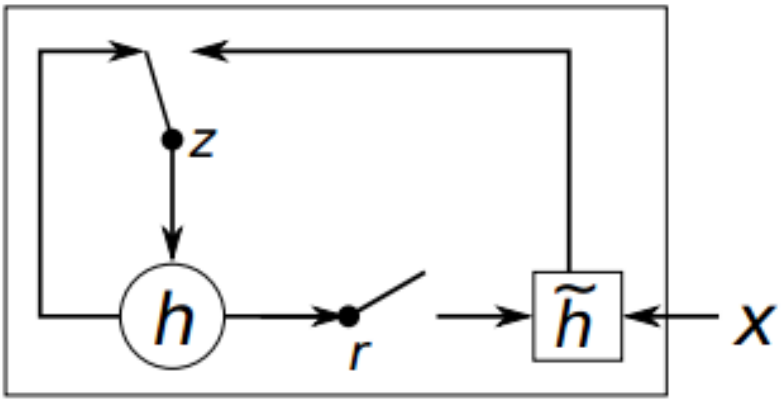

FIGURE 3.4: Effect of update and reset gates on the hidden state of the GRU cell (reprinted from Cho et al. (2014b))

If the reset gate is close to 0, previous hidden state will be forgotten and will be updated with the current input. This allows the cell to drop any irrelevant information. The actual activation is computed as follows.

$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \tilde{h}_t^j \tag{3.9}$$

Update gate controls the amount of previous hidden state that will be carried to the current hidden state. This helps the cell in remembering the past inputs for a long time. LSTM and GRU cells are widely being used in many sequence learning tasks because of their ability to store information for long time.

Chapter 4

Sequence to Sequence models

Since the LSTMs are good at learning temporal dependencies because of the gating units, they can be used effectively in sequence learning tasks. This chapter is devoted to the different Seq2Seq modules that have been used since it's inception in 2014. Even though the models presented here are using LSTMs, it can be replaced with any variant of RNN.

4.1 Basic Seq2Seq model

The simplest approach for sequence to sequence mapping is to convert the sequence of inputs to a fixed-sized vector using one RNN, and then to map the vector to the target sequence with another RNN. Since LSTMs are better at learning long-range time dependencies than RNN, we can use LSTMs here instead of RNNs. This approach has been used in English to French translation task (Sutskever et al., 2014). The goal of the model is to estimate the conditional probability $P(y_1....y_{T'}/x_1.....x_T)$ where $(x_1.....x_T)$ is the input sequence with length T and $(y_1....y_{T'})$ is its corresponding output sequence with length $T^{'}$. The model computes this conditional probability by first obtaining the fixed dimensional representation ϑ called as 'thought vector' of the input sequence $(x_1.....x_T)$ given by the last hidden state of

the LSTM, and then computing the probability of $y_1....y_{T'}$

$$\begin{aligned} P(y_1....y_{T'}/x_1.....x_T) &= P(y_1....y_{T'}/\vartheta) \\ &= \prod_{t=1}^{T'} P(y_t/\vartheta, y_1...y_{t-1}) \end{aligned} \tag{4.1}$$

In this equation, each $P(y_t/\vartheta, y_1...y_{t-1})$ distribution is represented with a softmax over all the words in the vocabulary.

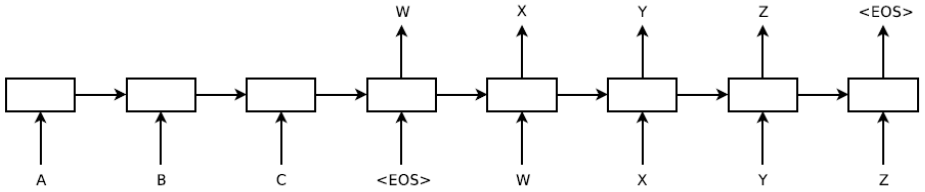

FIGURE 4.1: **Seq2Seq model**: ABC is the input sequence and WXYZ is the output sequence. EOS represents the End of Sentence. (reprinted from Sutskever et al. (2014))

This model makes use of two RNN networks: an encoder and a decoder. Encoder reads the input sequence one token at a time and converts into a fixed dimensional vector. i.e. $(x_1, x_2, ...x_T) \rightarrow \vartheta$. The most common approach is to use an RNN/LSTM such that

$$h_t = f(x_t, h_{t-1}) \tag{4.2}$$

$$\vartheta = q(h_1, h_2...h_T) \tag{4.3}$$

where h_t is a hidden state of the encoder at time 't'. For instance let $q(h_1, h_2...h_T) = h_T$. f and q are some non-linear functions. During training, the true output sequence is given to the model, so learning can be done by back propagation. Decoder is used to predict the next token y_t, given the thought vector ϑ and all the previous tokens $y_1, y_2, ...y_{t-1}$. With decoder RNN, this conditional probability distribution can be modeled as

$$p(y_t/\vartheta, y_1, y_2...y_{t-1}) = g(y_{t-1}, s_t, \vartheta) \tag{4.4}$$

where g is a nonlinear, potentially multi-layered, function that outputs the probability of y_t, s_t is the hidden state of decoder at time 't', given by

$$s_t = f(y_{t-1}, s_{t-1}, \vartheta) \tag{4.5}$$

The above equations are mentioned in Bahdanau et al. (2014). During testing time, we simply feed the present output token to predict the next one.

4.1.1 Decoding

Our training objective is to maximize the log probability of the correct response T , given the source sentence S. Maximize

$$\frac{1}{|\mathbf{S}|} \sum_{(T,S)\in\mathbf{S}} \log p(T/S) \tag{4.6}$$

Once training is complete, we produce responses by finding the most likely response according to the LSTM.

$$T^{'} = arg \max_{T} p(T/S) \tag{4.7}$$

Assuming N is the vocabulary size on the target size, and m is the maximum sequence length, we have a total of N^m possible target sentences. Instead of outputting the probabilities for all these sentences, we can use a left-to-right beam search decoder which maintains only B number of partial hypothesis at any time. At every time step, we can extend the hypothesis by using every possible word in the vocabulary. As soon as EOS is generated by a hypothesis, it is added to a set of complete hypothesis (Sutskever et al., 2014). Even thought this decoder is approximate, it is simple and efficient. Here, we used a beam-search of 1.

4.2 Seq2Seq Model with attention mechanism

In the basic model presented above, the hidden state of the encoder is transferred to decoder. Also, the output of decoder at each time step becomes the input to the decoder at the next time step. To allow the decoder more direct access to the

input, an attention mechanism was introduced in Bahdanau et al. (2014). This model does not encode the sentence into a single fixed-length vector. Instead, it encodes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding. This frees the model from compressing the input sequence into a fixed-length vector irrespective of source sentence length. This new model allows the decoder to peek into input at every decoding step (see figure 4.2). Here the conditional probability is modeled as

$$p(y_t/y_1, y_2...y_{t-1}, x) = g(y_{t-1}, s_t, c_t) \tag{4.8}$$

where g is a non-linear function and s_t is the hidden state of the decoder at time t, given by

$$s_t = f(y_{t-1}, s_{t-1}, c_t) \tag{4.9}$$

Where c_t is the context vector at time t and y_{t-1} is the output token at time

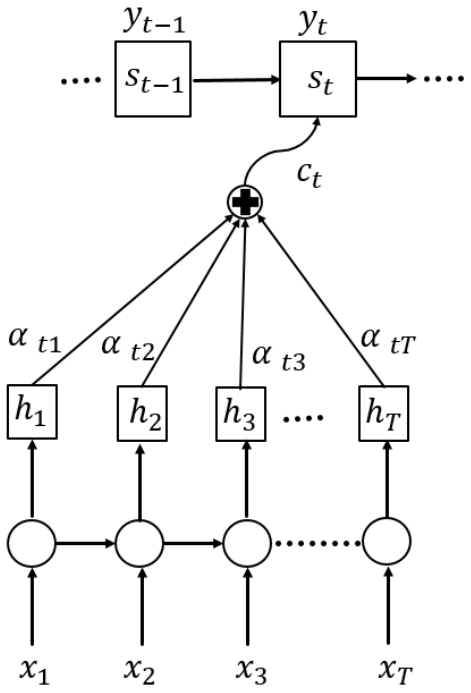

FIGURE 4.2: Illustration of alignment model

t-1. Assuming we are using GRU cells in the encoder and decoder networks, new

hidden state of the decoder is computed as follows.

$$s_t = (1 - z_t) \odot s_{t-1} + z_t \odot \tilde{s}_t \tag{4.10}$$

where $\odot$ indicates the element-wise multiplication and candidate hidden state $\tilde{s}_t$ is given by

$$\tilde{s}_t = \tanh(W\ e(y_{t-1}) + U\ (r_t \odot s_{t-1}) + C\ c_t) \tag{4.11}$$

where, $e(y_{t-1})$ is the embedding vector of y_{t-1} with dimension N. Assuming the vocabulary size is $V^{'}$ on the decoder side, y_{t-1} is 1-of-$V^{'}$ vector, while $e(y_{t-1})$ is simply a row of the embedding matrix $E \in \mathbb{R}^{V^{'}xN}$ Update gates and reset gates are calculated as follows.

$$\begin{aligned} z_t &= \tanh(W_z\ e(y_{t-1}) + U_z\ s_{t-1} + C_z\ c_t) \\ r_t &= \tanh(W_r\ e(y_{t-1}) + U_r\ s_{t-1} + C_r\ c_t) \end{aligned} \tag{4.12}$$

At every time step we compute output probabilities using eq (4.8) which uses softmax over all the target words.

Unlike the basic model , here the probability is conditioned on different context vector c_t for each symbol y_t. Context vector depends on the sequence of encoder hidden states $(h_1, h_2, ...h_T)$. Each state h_t contains the information about the sentence till the t^{th} word with more information regarding the words surrounding the t^{th} word. Context vector is calculated as weighted sum of hidden states

$$c_t = \sum_{j=1}^{T} \alpha_{tj} h_j \tag{4.13}$$

weights α_{tj} is given by

$$\alpha_{tj} = \frac{exp(e_{tj})}{\sum_{k=1}^{T} exp(e_{tk})} \tag{4.14}$$

where

$$e_{tj} = a(s_{t-1}, h_j) \tag{4.15}$$

is an alignment model. e_{tj} is a score of how well the input at position 'j' and output at position 't' match. Alignment model a can be modeled as a feed forward neural

network.

$$e_{tj} = v^T \tanh(W_1 h_j + W_2 s_{t-1}) \tag{4.16}$$

The vector v, matrices W_1, W_2 are the learn-able parameters of the model. e_{tj} contains a score of how much attention has to be paid for the j^{th} hidden state h_j while constructing the t^{th} output symbol y_t. These parameters are jointly optimized along with the encoder and decoder parameters to minimize the error.

4.3 Scheduled Sampling

The Seq2Seq models presented above tries to maximize the likelihood of the target token, given the previous token and the current state. When the training is in progress, the true previous token which is available from the training data, is fed to the decoder at the training time. But, at the inference time, this previous token is unknown and is considered to be the generated by the model itself, in the last time-step. So, the main problem here is, the mistakes made at early while generating the sequence are fed back and amy cause the amplification in the error. This discrepancy, between training and testing procedures can yield errors and may accumulate over the sequence generated. Bengio et al. (2015) proposed a learning strategy, which changes the training process gradually from fully guided to less guided.

This learning approach gradually changes the training procedure to deal with its own mistakes, just like the model has to deal with errors while inference time. This makes the model more robust to errors at inference time, as it has already learned to do so while training.

Given the training set (X_i, Y_i) pairs where the maximum length of input sequence is T and that of target sequence is $T^{'}$. we have to model

$$P(Y/X) = \prod_{t=1}^{T^{'}} P(y_t/y_{t-1}, X) \tag{4.17}$$

The r.h.s of the equation is predicted by a decoder RNN, whose hidden state h_t is a function of previous hidden state h_{t-1} and previous output token y_{t-1} .i.e

$$P(y_t/y_{t-1}, X) = P(y_t/h_t; \theta) \tag{4.18}$$

where, θ is the parameters of the model. h_t is computed as follows.

$$h_t = \begin{cases} f(X;\theta) & \text{if } t = 1 \\ f(h_{t-1}, y_{t-1}; \theta) & \text{otherwise} \end{cases} \tag{4.19}$$

The main difference between training and inference,while predicting the token y_t is is whether we use the actual previous token y_{t-1} or an estimate $\tilde{y}_{t-1}$ coming from the model. Scheduled sampling mechanism will randomly choose between y_{t-1} and $\tilde{y}_{t-1}$ while training. Assuming we are training using mini-batch gradient descent, for the i^{th} training iteration, we will choose the true token with probability ϵ_i and the estimate with probability $(1 - \epsilon_i)$. Estimate from the model can be taken as $arg\max_s\ P(y_{t-1} = s\ |h_{t-1})$.

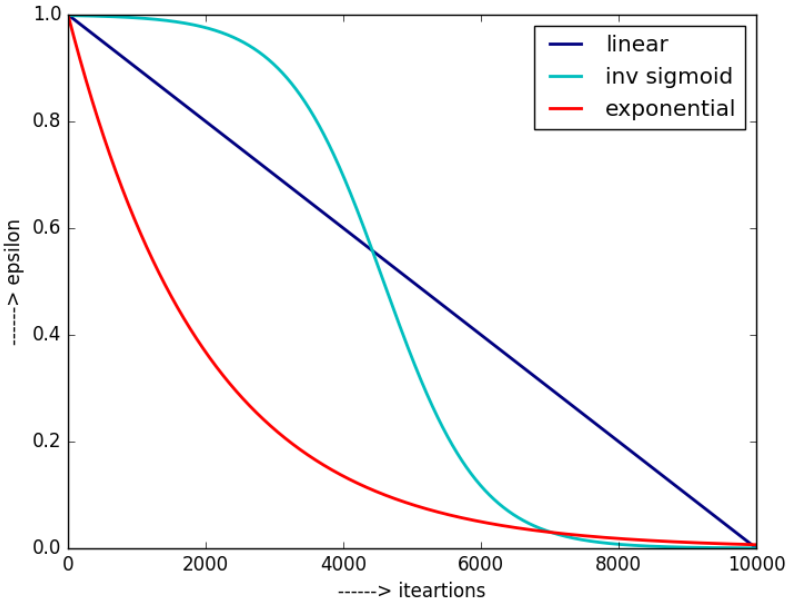

FIGURE 4.3: Examples of decay schedules

If $\epsilon_i = 1$, then the model is same as before while training and if $\epsilon_i = 0$ then the setting is inference. As the ϵ_i changes between 1 and 0 gradually our training

becomes robust to the errors made by the network. In the initial stages of the training, as the parameters of the model are not properly tuned, model is more likely to make more mistakes and we have to provide the correct token more often, So in this case higher value of ϵ_i does the job. On the other hand, as the training progresses, the model should learn from mistakes just like in the case of inference. So ϵ_i value has to be low, for this to happen. At the end of the training, the model should be good enough to sample the reasonable tokens.

We have to choose a function, which reduces the value of ϵ_i as a function of i. Following are some of the sampling schedules (see figure 4.3).

1. Linear decay: $\epsilon_i = \max(\epsilon, k - ci)$ where $0 \leq \epsilon < 1$, k and c provide the offset and slope of the decay, depends on the expected speed of convergence.
2. Exponential decay: $\epsilon_i = k^i$ where $k < 1$ is a constant that depends on the expected speed of convergence.
3. Inverse sigmoid decay: $\epsilon_i = \frac{k}{k+\exp(i/k)}$ where $k \geq 1$ depends on the expected speed of convergence.

4.4 Sampled Softmax

In the Seq2Seq models presented above, it is assumed that the output layer contains the number of neurons that are equal to the target vocabulary size, so that at every time step, we can emit the probabilities for all the words in the vocabulary. From eq (4.8) we can see that the probabilities are a function of current hidden state of the decoder, previous output token and current context vector.The probability distribution of the next target word is computed by

$$P(y_t/y_1, y_2..y_{t-1}, x) = g(y_{t-1}, s_t, c_t) \tag{4.20}$$

$$P(y_t/y_1, y_2..y_{t-1}, x) = \frac{1}{Z} \exp(w_t^T \ \phi(y_{t-1}, s_t, c_t) + b_t) \tag{4.21}$$

where, ϕ indicates the non-linear feature obtained out of the layer just below the output layer (penultimate layer), w_t is the target word vector and b_t is the bias,

Z is the normalization term given by

$$Z = \sum_{k:y_k \in V'} \exp(w_k^T \ \phi(y_{k-1}, s_k, c_k) + b_k) \tag{4.22}$$

To compute the output probabilities, we have to compute the dot product between the feature ϕ and word vectors w_t as many times as the target vocabulary size. This has to be done for every word in the target sentence. Hence it becomes computationally expensive. This is the most common problem in language models having high vocabulary with output softmax layer. Since its cost function involves a normalization term which has to be computed using all the vocabulary. The complexity increases as the target vocabulary size increases. So, training a seq2seq model becomes complex with the increasing vocabulary size, especially on the target side. Applying logarithm on both sides of the equation eq(4.21) then

$$\log P(y_t/y_{<t}, x) = \mathcal{E}(y_t) - \log \Big(\sum_{k:y_k \in V'} \exp(\mathcal{E}(y_k)) \Big) \tag{4.23}$$

where, $\mathcal{E}(y_t) = w_t^T \ \phi(y_{t-1}, s_t, c_t) + b_t$ is the energy
Applying gradient on both sides with respect to parameter θ

$$\begin{aligned}
\nabla_\theta \log P(y_t/y_{<t}, x) &= \nabla_\theta(\mathcal{E}(y_t)) - \frac{1}{\sum_{m:y_m \in V'} \exp(\mathcal{E}(y_m))} \sum_{k:y_k \in V'} \exp(\mathcal{E}(y_k)) \ \nabla_\theta(\mathcal{E}(y_k)) \\
&= \nabla_\theta(\mathcal{E}(y_t)) - \sum_{k:y_k \in V'} \frac{\exp(\mathcal{E}(y_k))}{\sum_{m:y_m \in V'} \exp(\mathcal{E}(y_m))} \ \nabla_\theta(\mathcal{E}(y_k)) \\
&= \nabla_\theta(\mathcal{E}(y_t)) - \sum_{k:y_k \in V'} P(y_k/y_{<t}, x) \ \nabla_\theta(\mathcal{E}(y_k))
\end{aligned} \tag{4.24}$$

The second term of the equation (4.24) is the expectation of gradient of energy $\mathbf{E}_P[\nabla_\theta(\mathcal{E}(Y))]$. It uses the whole words in the vocabulary to compute the gradient. i.e. to make changes in the network one time we have to go through the entire vocabulary. It can be expensive, when the number of words is high. So, it is a challenge to train the network with large number of vocabulary. Several strategies have been proposed to approximate this term. There are sampling based

approaches as well as softmax based approaches. Softmax based approaches are efficient ways of computing the softmax without changing the objective function. e.g Hierarchical Softmax. Sampling based approaches in fact optimizes some other loss function which approximates the softmax. e.g. Importance sampling (Bengio and Senécal, 2008).

The idea of importance sampling is to approximate the expectation of distribution by using Monte-Carlo method. i.e. by taking the average of random samples from the distribution

$$\mathbf{E}_P[\nabla_\theta(\mathcal{E}(Y))] = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta(\mathcal{E}(y_i)) \tag{4.25}$$

But here, we do not have the probability distribution of gradients P to take samples from. Many applications use another distribution Q from which it is cheap to sample . Using the proposal distribution Q

$$\begin{aligned} \mathbf{E}_P[\nabla_\theta(\mathcal{E}(Y))] &= \sum_{y \in Y} P(y)\ \nabla_\theta(\mathcal{E}(y)) \\ &= \sum_{y \in Y} Q(y)\ \frac{P(y)}{Q(y)} \nabla_\theta(\mathcal{E}(y)) \\ &= \mathbf{E}_Q\Big[\frac{P(Y)}{Q(Y)} \nabla_\theta(\mathcal{E}(Y))\Big] \end{aligned} \tag{4.26}$$

Using this method we need to select samples $y_1, y_2 ... y_m$ only from Q.This estimator is known as importance sampling. The most suitable option for Q to use is the unigram distribution from the training set. Even though we have samples, we still have to evaluate $P(y_i), i = 1, 2..m$, which is not possible without explicitly computing P. Kong et al. (1994) proposed a biased important sampling estimator which uses $\frac{1}{W} w(y_i)$ to weigh $\nabla_\theta \mathcal{E}(y_i)$ with $w(y) = \frac{\exp(\mathcal{E}(y))}{Q(y)} = \exp(\mathcal{E}(y) - \log Q(y))$ and $W = \sum_{i=1}^{m} w(y_i)$ Then the estimator becomes

$$\frac{1}{W} \sum_{i=1}^{m} w(y_i) \nabla_\theta(\mathcal{E}(y_i)) \tag{4.27}$$

Even though this estimator is biased, it converges to the true average as $m \to \infty$ (Bengio and Senécal, 2008). So, only m number of words from the vocabulary are used to approximate this term, and update the correct word vector w_t and

the vectors of the samples instead of updating all the vectors. Jean et al. (2014) suggested to partition the training set and to use only a subset of vocabulary for each partition. Let V_i be the subset of wotds for the i^{th} partition, and Q_i be the sampling distribution such that

$$Q_i(y_k) = \begin{cases} \frac{1}{|V_i|} & \text{if } y_k \in V_i \\ 0 & \text{otherwise} \end{cases}$$

This choice of Q_i cancels out the effect of $Q(y)$ in the weight term $w(y)$ which makes

$$P(y_t/y_{<t}, x) = \frac{\exp(\mathcal{E}(y_t))}{\sum_{m:y_m \in V'} \exp(\mathcal{E}(y_m))} \tag{4.28}$$

Intuitively, the eq(4.21) can be thought of as arranging the target word vectors close to the extracted feature $\phi(y_{t-1}, s_t, c_t)$. As the learning progresses, the vectors of all the target words that are most likely are tend to align with one another but not with the others. This happens when we use the exact gradient of eq(4.21), which contains all the word vector terms. When we use the approximation, only the vectors of the words in the selected sample will be moved in the right direction. The less the number of samples we use the worst is the approximation.

Sampling based approaches like importance sampling are useful only at the training time. Since we don't have access to the target words at the testing time, we use the entire vocabulary for the inference. During inference, we have to use the full softmax on entire target vocabulary to get the probabilities at the testing time to predict the next token.

4.5 Visualizations using t-SNE

t-SNE is a non-linear dimensionality reduction technique. It can be used as a visualization tool by reducing the dimension of data which is in higher dimensional. It was proposed by Maaten and Hinton (2008). It is an improvement over SNE (Hinton and Roweis, 2003). This is a probabilistic approach to reduce the dimensionality. SNE models the conditional probabilities based on the distances

between the high dimensional data points and it tries to achieve the similar probabilities in low dimensional space. tSNE is a variant of SNE, which is much easier to optimize and produce better visualizations.

Let $X = \{x_1, x_2...x_n\}$ be the data points in high dimension and $Y = \{y_1, y_2...y_n\}$ be the data points in lower dimensional space. SNE models the conditional probabilities based on the pair-wise similarity between the data points. Similarity of a data point x_i to x_j can be thought of as the probability of x_i picking up x_j as its neighbor. It is modeled as

$$p_{j/i} = \frac{exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-\|x_i - x_k\|^2/2\sigma_i^2)} \tag{4.29}$$

where, σ_i is the variance of the Gaussian centered at x_i. σ_i depends on the density of the neighborhood of data point x_i. if the density is high, then σ_i is less and vice versa. $p_{j/i}$ will be relatively high for nearby points when compared far away points. Similarly for lower dimensional data points y_i and y_j the probability is modeled as

$$q_{j/i} = \frac{exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} exp(-\|y_i - y_k\|^2)} \tag{4.30}$$

Here the Gaussian is considered to have a variance of $\frac{1}{\sqrt{2}}$. Since we are only interested in modeling pair-wise similarities, we set $p_{i/i} = q_{i/i} = 0$. Our aim of dimensionality reduction is to match these two distributions well. i.e. the mismatch or divergence between the two distributions has to be minimized. The amount of faithfulness with which the distribution $q_{j/i}$ models the distribution $p_{j/i}$ is given by Kullback-Leibler divergence. The cost function, therefore is given by the sum of divergences over all the data points.

$$C = \sum_i \mathrm{KL}(\mathrm{P}_i || \mathrm{Q}_i) = \sum_i \sum_j p_{j/i} log \frac{p_{j/i}}{q_{j/i}} \tag{4.31}$$

where, P_i indicates the conditional distribution over all data points given x_i. Similarly, Q_i indicates the conditional distribution over all map points in low dimensional space given y_i. SNE finds the low-dimensional representation of data that minimizes the cost function. Since we use some iterative approach like gradient

descent, we have to find the gradient of the cost function with respect to the map points. The gradient is given by

$$\frac{\partial C}{\partial y_i} = 2\sum_j (p_{j/i} - q_{j/i} + p_{i/j} - q_{i/j})(y_i - y_j) \tag{4.32}$$

We can see that the gradient is proportional to the mismatch $(p_{j/i} - q_{j/i} + p_{i/j} - q_{i/j})$ between the similarities of data points and map points. Even though SNE produces good visualizations there are some limitations.

- The cost function is not symmetric. The cost for using widely separated map points to represent nearby data points is more but the cost is less for using nearby map points to represent widely separated data points.
- Crowding of map points.

tSNE produces better visualizations by alleviating the above problems.

- It uses the symmetric cost function by using the joint probabilities between the data points rather than the conditional distributions. It also results in simpler gradients.
- It uses the t-distribution to model the probabilities in low dimensional space. It addresses the crowding problem by using the heavy-tailed t-distribution when compared to the Gaussian distribution.

Now, the symmetric version of tSNE cost function is

$$C = \mathrm{KL}(\mathrm{P}\big|\big|\mathrm{Q}) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{4.33}$$

where P and Q are the probability distributions in high and low dimensions respectively. Symmetry is achieved by making $p_{ji} = p_{ij}$. The joint probability distributions p_{ij} and q_{ij} is given by

$$p_{ij} = \frac{exp(-\|x_i - x_j\|^2/2\sigma^2)}{\sum_{k \neq l} exp(-\|x_k - x_l\|^2/2\sigma^2)} \tag{4.34}$$

$$q_{ij} = \frac{exp(-\|y_i - y_j\|^2)}{\sum_{k \neq l} exp(-\|y_k - y_l\|^2)} \tag{4.35}$$

The gradient of symmetric SNE is given by

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) \tag{4.36}$$

The problem with symmetric SNE is when a data point x_i is an outlier. In such a case all $\|(x_i - x_j)\|^2$ are large and hence all p_{ij} s are small. Hence the location of map point y_i that corresponds to the data point x_i has very little effect on the cost function. Define $p_{ij} = \frac{p_{i/j} + p_{j/i}}{2}$, which makes $\sum_j p_{ij} > \frac{1}{2n}$ and hence every data point makes a significant contribution to the cost function.

One more problem specified by Maaten and Hinton (2008) is "The pairwise distances in two dimensional space can not faithfully model distances between higher dimensional space". For e.g. In ten dimensions, it is possible to have 11 mutually equidistant points and there is no way we can model this in two-dimensional space. The reason for this is the very different distribution of distances in the two spaces. The area of the two-dimensional space that is available to accommodate moderately distant data points will not be nearly large enough compared with the area available to accommodate nearby data points. This problem is referred to as 'the crowding problem'. To alleviate this problem, tSNE uses a heavy-tailed t-distribution with one degree of freedom to model the probabilities in the low dimensional space. t-distribution is used because it is closely related to the Gaussian distribution. The joint distribution between map points is given by

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}} \tag{4.37}$$

Then the gradient of tSNE cost function is given by

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1} \tag{4.38}$$

tSNE is used in our experiments to visualize the 'thought vectors' that came out of encoder RNN. We expect similar questions to be mapped to nearby data points on the two-dimensional space and dissimilar sentences to be represented by widely-spread points.

Chapter 5

Experiments and Results

This chapter presents the evaluation metrics to measure the performance of the models along with training details and the results obtained on three different datasets with the analysis and visualizations.

5.1 Evaluation metrics

Different evaluation metrics are being used for sequence learning tasks. BLEU (Bilingual Evaluation Understudy) score, Perplexity, METEOR (Metric for Evaluation of Translation with Explicit ORdering) scores are used for Seq2Seq mapping tasks. Most of these metrics are not highly correlated with the human judgement.

5.1.1 BLEU score

BLEU is an evaluation metric for machine translation, proposed by Papineni et al. (2002). BLEU score is the most popular evaluation metric used for Machine Translation (MT) systems. This evaluation metric is said to be inexpensive, language-independent and correlates closely with the human judgment. BLEU score doesn't take grammatical correctness into account. It favors the short translations compared to long translations. BLEU score is based on modified n-gram precision. It is the geometric mean of the n-gram precisions for all values of n between 1 and some upper limit N. To compensate the favor for the short translations, brevity

penalty B is introduced. It takes into account the average source and translation lengths.

$$B = \begin{cases} 1 & \text{if } c > r \\ e^{1-\frac{r}{c}} & \text{if } c < r \end{cases}$$

BLEU score is given by,

$$BLEU = B.\ \exp\big(\frac{1}{N}\sum_{n=1}^{N}\log p_n\big) \tag{5.1}$$

where,

p_n is the modified n-gram precision,which is the number of n-grams in the candidate translation that occur in any of the reference translations, out of total number of n-grams in the candidate translation. There is no guarantee that, the translation with higher BLEU score is better than the one with the low score. Papineni et al. (2002) used N=4 for their evaluations.

5.1.2 Perplexity

The training uses the word-level cross entropy loss based on the probabilities emitted from the decoder.

$$\text{loss} = -\frac{1}{N}\sum_{i=1}^{N}\log p_{target_i} \tag{5.2}$$

where $target_i$ is the target word at the i^{th} position. Then the average per word perplexity is given by

$$\text{perplexity} = \exp(\text{loss}) \tag{5.3}$$

Perplexity measure has been proved to be effective and useful for many sequence to sequence tasks.

5.2 Training

The training data consists of source sentence and the corresponding translation. Every translation is prepended by a symbol *GO* and appended by *EOS* to mark the start and end of the translation. Consider the translation $ABC \rightarrow WXYZ$. In this case, the encoder inputs would be A, B, C in sequence. Decoder inputs would be *GO*, W, X, Y, Z. Outputs of the decoder expected at each time step would be W, X, Y, Z, *EOS*.

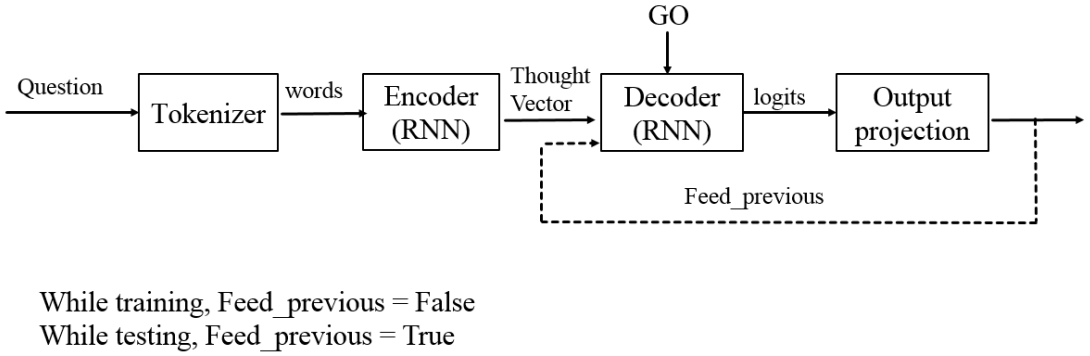

FIGURE 5.1: **Block diagram of Machine translation system**: while training feed_previous = False, while testing feed_previous = True

Many a times, the output of the decoder at time 't' is fed back to the decoder, to predict the word at time 't+1'. At test time, when decoding a sequence, this is how the sequence is constructed. During testing time, model will generate tokens until it generates *EOS* token. During training, on the other hand, it is common to provide the correct input to the decoder at every time-step, even if the decoder made a mistake before. The frequency of the words in counted in both source side and target side. Since the words that occur less frequently can't capture much information, we can ignore them. Words that occurred more than *min_times* are considered to be in the vocabulary and will be given a unique ID. All the other words are marked as *UNK* . Each word in the vocabulary is represented by an ID, i.e. each word has a discrete representation.

Before feeding the words to the encoder, we should represent them by a dense

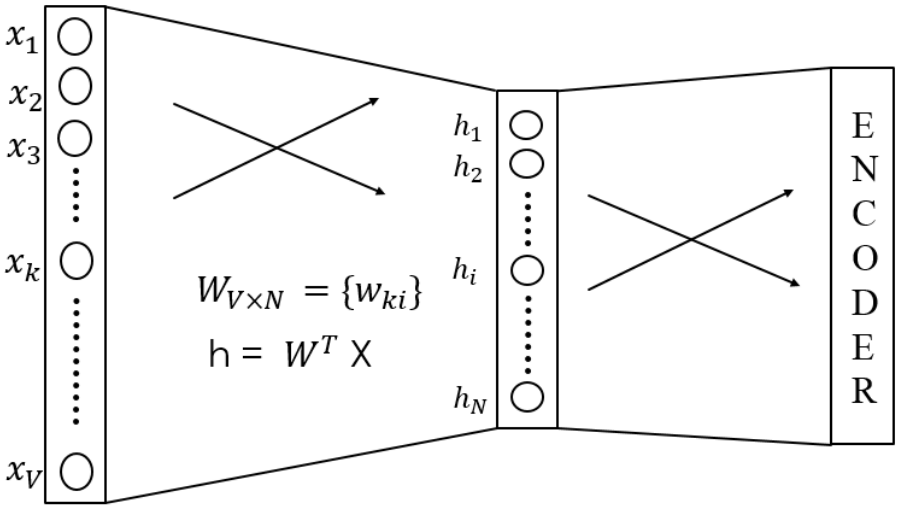

FIGURE 5.2: Distributed representation of words

vector representation. Each word is represented using 1-of-V coding, which means for a given input word, only one out of V units $\{x_1, x_2...x_V\}$ will be 1, and all other units are 0, where V is the size of the vocabulary. Then each word is projected onto the projection layers using a shared projection matrix $W_{V\times N}$, which is of size VxN, where N is the size of the hidden layer in encoder. Representation of the word is obtained by $W^T X$. Assume that the i^{th} row of W is v_{wi}^T (consists of N elements). Given an input word (let say k^{th} word in the vocabulary), $x_k = 1$ and $x_{k'} = 0$ for $k' \neq k$. Then we have

$$h = W^T X = v_{wk}^T \tag{5.4}$$

which is essentially copying the k^{th} row of the matrix. Since at any time, 1 of the V neurons is active, only one row of W is copied to the output, which is the vector representation of the given word (Rong, 2014).

Then the source sentence is fed sequentially to the encoder, word by word and encoder generates a representation for the source sentence once all the words are fed. This is referred to as 'Thought vector'. Ideally, for the source sentences that are similar, we get similar thought vectors.

Once a thought vector is generated, it will be considered as a initial hidden state

of the decoder. Now a special symbol, *GO* is fed to the decoder along with other decoder inputs one after the other to generate the next tokens. Decoder will generate an array of V' values, each corresponds to the logits of the vocabulary on the target side. Logits range from $-\infty$ to ∞. Once the logits are generated, we can calculate the probabilities . Since we know the targets, we can calculate the error or loss by using cross entropy. Cross entropy can be thought of as a measure of compatibility between two probability distributions. The more the entropy the less the compatibility.

$$y_k = \frac{e^{\text{logit}(k)}}{\sum_{i=1}^{V'} e^{\text{logit}(i)}} \tag{5.5}$$

$$\text{Cross Entropy Loss} = -\sum_{k \in V'} z_k \log(y_k) \tag{5.6}$$

where z_k is the true label, and y_k is the probability of belonging to the class 'k'. $k \in [1, V']$. Since this is like a classification problem and classes are mutually exclusive i.e at any position there can only be one word from the vocabulary $z_i = 1$ for some $i = k$, and $z_i = 0$ for $i \neq k$.

Once the error/loss is calculated, gradients of weights in the encoder and decoder networks are calculated and the weights are jointly optimized together to minimize the error.

5.3 Datasets

This section explains the different datasets used for the study and the experimentation done on these datasets. All the experiments done here uses a greedy approach. i.e. we use a beam search of 1 to generate the target sentence. Experiments are done using TensorFlow 0.10 and using python.

5.3.1 Philips Amplight Dataset

Amplight is a product of Philips. Our aim is to build a chatbot which answers users queries related to amplight. Training data contains a set of 900 questions which corresponds to a set of 90 unique FAQs. Since we do not want to generate the grammatically false sentence in this case, we mapped each question to a token,

instead of the answer directly. Hence the model here acts like a sequence classification. Once the token is generated using the model, the answer corresponding to that token will be displayed to the user. Since the dataset is small, only the basic Seq2Seq model is applied on this dataset. To evaluate the performance of the model, tSNE is used to project the thought vectors obtained out of the encoder-RNN. Figures 5.3 and 5.5 shows the projected vectors on the 2 dimensional space. We can clearly see the similar questions forming clusters, which indicates that the system has learned to understand the questions and can differentiate dissimilar ones.

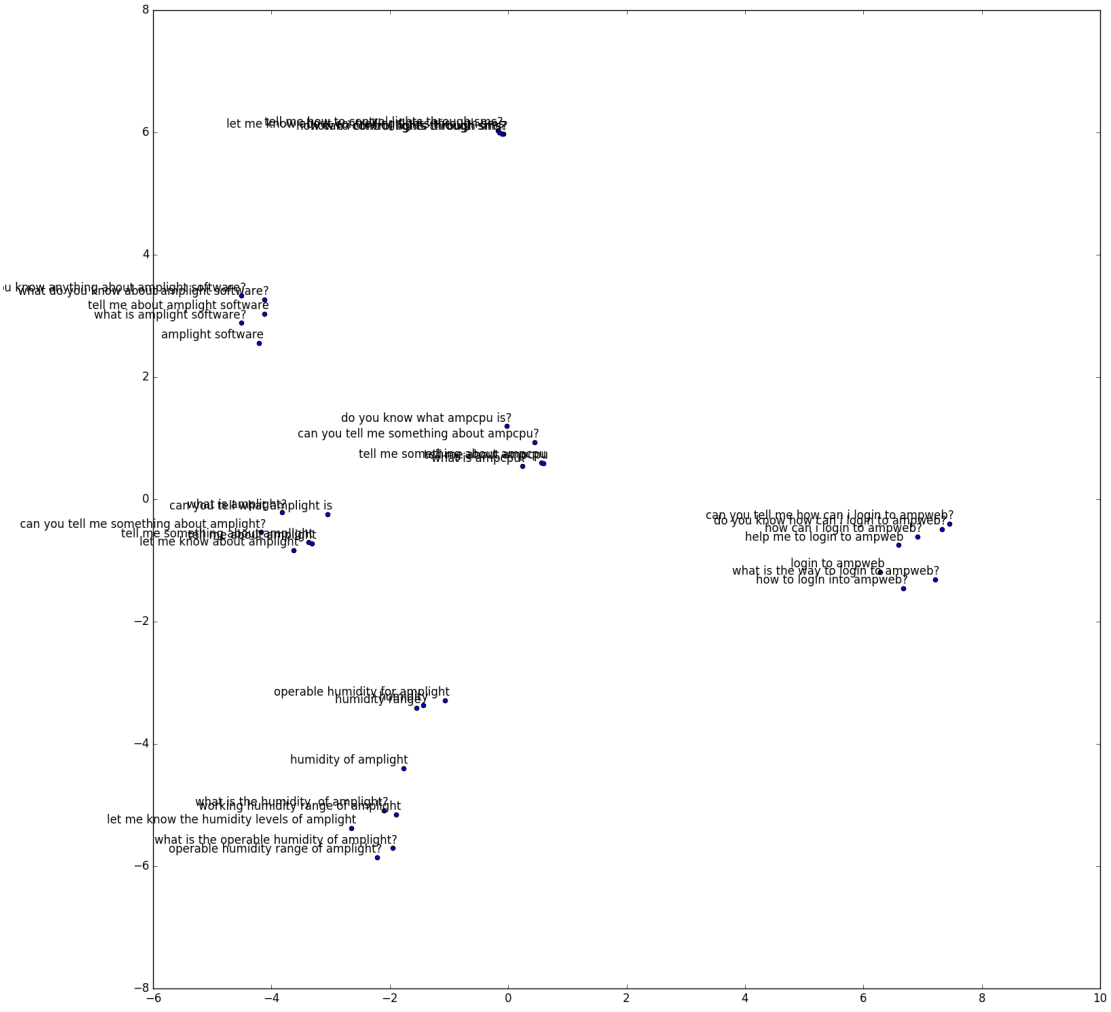

FIGURE 5.3: projections of the thought vectors

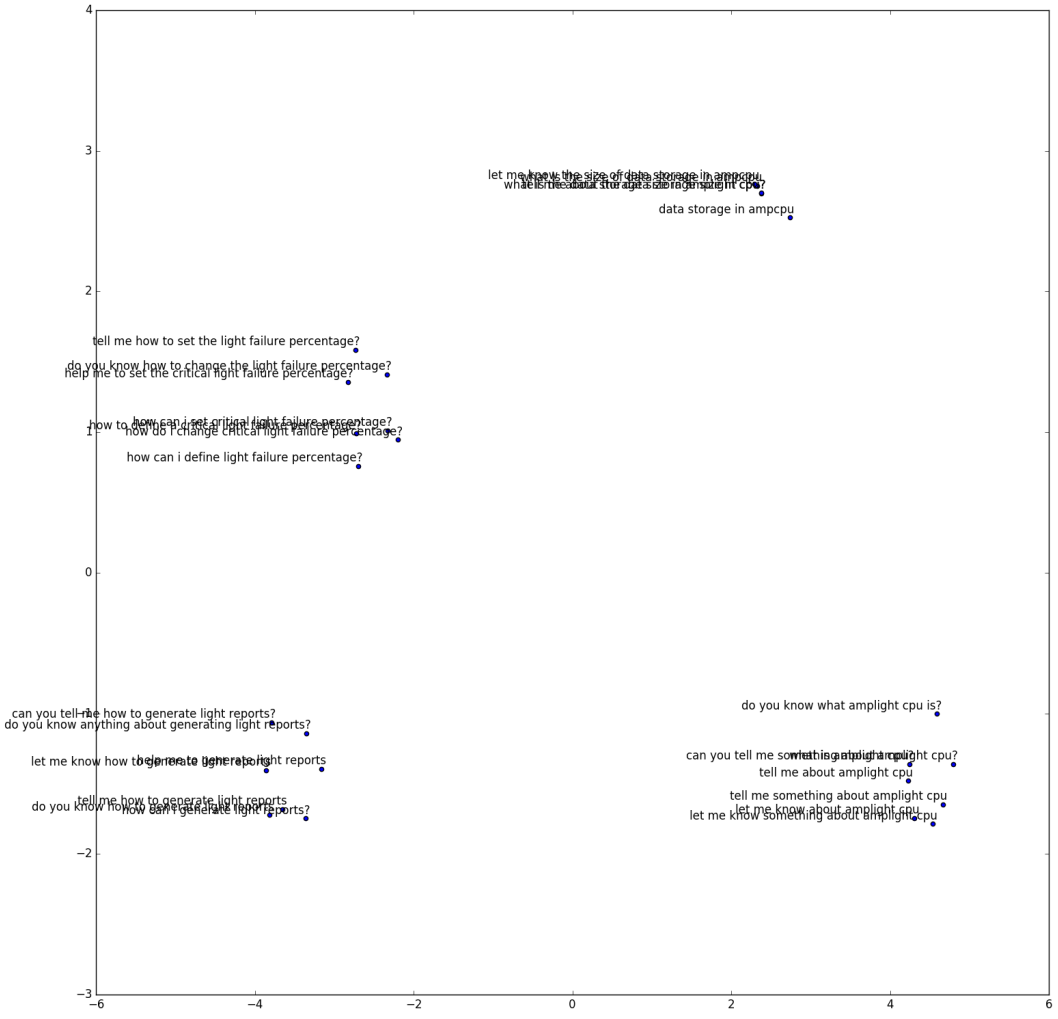

FIGURE 5.4: projections of the thought vectors

5.3.2 Cornell movie dataset

This corpus contains a set of 220,579 fictional conversations involving 9035 characters from 617 movies including the meta-data (Danescu-Niculescu-Mizil and Lee, 2011). After pre-processing (removing movie name, character names etc) and limiting the length to less than 20, we got a final set of 139979 conversations. Seq2Seq model with and without attention mechanism is applied on this dataset. Here are the model parameters used

- source vocab size 20,000
- target vocab size 20,000
- number of layers 2

- number of hidden neurons in each layer 512
- learning rate (initial) 0.5
- learning rate decay factor 0.99

same parameters are used for both the experiments, so that we can compare the results. Since the vocabulary size is 20k, sampled softmax with 1024 samples is used to make the training go faster. Interestingly, not much change in perplexity is observed using the attention mechanism. Here is a simple conversation produced from the model

human: hello

machine: hello

human: how are you?

machine: i 'm fine

human: who made you?

machine: you

human: are you human?

machine: no

human: are you a robot?

machine: yes

human: what's your name?

machine: you know

human: are you male?

machine: no

human: are you female?

machine: yes

human: will you come to movie with me?

machine: i will

human: what time is okay with you?

machine: eight

human: where do you live?

machine: i do n't know

human: what's your age?

machine: you know

human: how old are you?

machine: i 'm fine

human: are you woman?

machine: no

human: are you man?

machine: yes

The conversation above suggests that the model can converse with a human fluently. Since the model is trained for one-to-one mapping, it is unable to take into account the previous sentences while conversing. Hence, the model is not good at maintaining the context. It can also be seen from the conversation that the model lacks the personality and its responses are inconsistent. For e.g. model responses are not the same when asked the questions "how old are you?" and "what's your age?". This is because of using the noisy, open-domain dataset that is used to train the model. i.e. dataset itself is not consistent. This lack of consistent personality makes it difficult to pass the turing test (Turing, 1950).

5.3.3 TIDES-IIIT parallel Corpus

The DARPA-TIDES corpus was released as part of language contest on SMT in 2002. After manual refinement and cleaning, a subset of this corpus was released for the NLP Tools Contest (Venkatapathy, 2008) on SMT for English-Hindi. As mentioned in Venkatapathy (2008) "the translations are not faithful. They are paraphrased in such a way that they convey the meaning in the best possible way. i.e they are not the exact translations.". This data set contains 50k training sentences and 1k each for validation and testing (Bojar et al., 2010). Here we limited the sequence length to 30. Hence, any training data that has more than 30 tokens are omitted in our experiments. This made the effective training data of size 33028 and validation data size 716. Following are the model parameters used.

- English Vocabulary 8k
- Hindi Vocabulary 8k

- number of layers 2
- number of cells in each layer 512
- sampled softmax with 1024 samples
- batch size 32

The 8k tokens in the vocabulary corresponds to 93% of tokens on source side and 93.6% on the target side. BLEU score achieved on the training set is 8.05 and on the validation set is 2.59. When we looked at the decoded translations using this model, we found that many of the tokens were *UNK*, which we think is the reason for such a BLEU score. We experimented now by increasing the vocabulary size to 12k on both sides which covered 95.3% and 96.1% of total tokens occurred in the English and Hindi sentences respectively. With every other training parameters same, we obtained a BLEU score of 21.47 on the training set and 2.7 on the validation set. With the basic Seq2Seq model without attention mechanism and with the vocabulary size of 12k the BLEU score achieved on the training set is 3.66 and that on the validation set is 1.69. And with a vocabulary size of 8k, the BLEU score on training set is 3.48 and on validation set is 2.12.

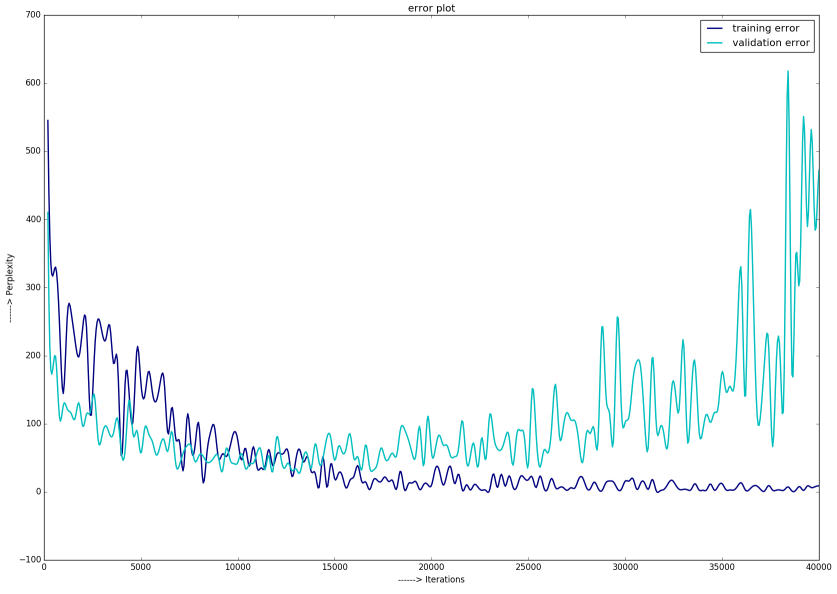

FIGURE 5.5: Overfitting the training data: Initially the error decreases on both datasets. After certain time validation error increases. The parameters at which validation error is minimum has to be chosen as the optimal values.

Figure 5.5 shows the perplexity vs iterations plot. We can see that the training and validation error decreasing initially, but after some iterations the validation error is increasing, which is an indication that the model is overfitting the training data. The point after which error starts to rise is considered to be the optimal point. The peaks and the valleys in the error plot are attributed to the fact that mini-batch gradient descent is used to train the model and not the full-batch.

Model used	**BLEU score**	
	Vocabulary 8k	Vocabulary 12k
Basic Seq2Seq	3.48	3.66
Seq2Seq with attention	8.05	21.47

TABLE 5.1: Experimental Results on TIDES-IIIT parallel corpus.

Table 5.1 shows the experimental results on the parallel corpus. From the table it can be clearly seen that using Seq2Seq model with attention mechanism improved the translation performance. Using more vocabulary also boosts the translation performance because of less number of *UNK* tokens in the dataset. Using scheduled sampling, with exponential decay schedule after 30k iterations has resulted in a BLEU score of 19.87 on training set and 2.96 on validation set with every other parameters as same. Exponential decay sampling after first 50k iterations showed a BLEU score of 23.13 and 3.12 respectively on training and validation sets. Even though scheduled sampling resulted in performance improvement, the choice of decay schedule has proved to be crucial. Using the scheduled sampling from the beginning of the training has deteriorated the performance of the model. Hence it is suggested to use a decay schedule which decreases slowly and is better to use it after certain number of iterations.

Chapter 6

Conclusions and Future work

6.1 Conclusions

This thesis presented various end-to-end sequence to sequence mapping architectures that have been used for machine translation. These models learn the semantic and syntactic relations between the source and target sentence pairs by maximizing the likelihood of the target sentence given the source. Seq2Seq model with the attention mechanism has shown significant improvement in the BLEU score of the translation task applied on TIDES-IIIT English-Hindi corpus. But the same model didn't result much change in the perplexity on the cornell movie datset. This shows that the attention mechanism is useful in learning the alignment between the source and target words in MT task rather than on a mere question-answering task. Scheduled sampling has also resulted in the performance improvement of the model, but one should choose a proper decay schedule, otherwise it could lead to a bad model. The projections of the thought vectors from an encoder network onto two dimensional space clearly shows that the similar questions forming clusters and are away from different questions. This shows that the model generates similar thought vectors for similar questions. Finally, it is concluded that the model performed reasonably on MT task even with limited vocabulary and without making any assumptions about the data structure

6.2 Future Studies

The standard Seq2Seq models studied in this thesis maximizes the likelihood of the target sentence given the source sentence $P(Y/X)$ i.e they only capture the uni-directional source-to-target dependency and ignore $P(X/Y)$. Modeling this target-to-source dependency $P(X/Y)$ combined with $P(Y/X)$ may result in the improvement in the performance. Using Bi-directional RNN or Bi-directional LSTMs may be useful for this purpose.

The evaluation metrics presented here like BLEU or perplexity measures are not much in correlation with the human judgement and doesn't take grammatical correctness into account. The study of new evaluation measures can be explored.

Multi-tasking can be mixed with Seq2Seq models like using a shared encoder to translate into multiple languages from the source language or to translate as well as parsing at the same time.

Instead of using the random word embeddings, we can use the word vectors obtained from a language model so that it can speed-up the training process. Finally the use of Seq2Seq models for other tasks other than Machine Translation task can be explored.

Bibliography

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Justin Simon Bayer. *Learning Sequence Representations*. PhD thesis, München, Technische Universität München, Diss, 2015.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015.

Yoshua Bengio and Jean-Sébastien Senécal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Transactions on Neural Networks*, 19(4):713–722, 2008.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Ondřej Bojar, Pavel Straňák, Daniel Zeman, Gaurav Jain, and Om Prakesh Damani. English-hindi parallel corpus. 2010.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014a.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014b.

Cristian Danescu-Niculescu-Mizil and Lillian Lee. Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs. In *Proceedings of the 2nd Workshop on Cognitive Modeling and Computational Linguistics*, pages 76–87. Association for Computational Linguistics, 2011.

Trevor Darrell, Marius Kloft, Massimiliano Pontil, Gunnar Rätsch, and Erik Rodner. Machine learning with interdependent and non-identically distributed data (dagstuhl seminar 15152). In *Dagstuhl Reports*, volume 5. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

Murat Dundar, Balaji Krishnapuram, Jinbo Bi, and R Bharat Rao. Learning classifiers when the training data is not iid. In *IJCAI*, pages 756–761, 2007.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

Alex Graves. Supervised sequence labelling. In *Supervised Sequence Labelling with Recurrent Neural Networks*, pages 5–13. Springer, 2012.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 2016.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition:

The shared views of four research groups. *IEEE Signal Processing Magazine*, 29 (6):82–97, 2012.

Geoffrey E Hinton and Sam T Roweis. Stochastic neighbor embedding. In *Advances in neural information processing systems*, pages 857–864, 2003.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. *CoRR*, abs/1412.2007, 2014. URL `http://arxiv.org/abs/1412.2007`.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

Augustine Kong, Jun S Liu, and Wing Hung Wong. Sequential imputations and bayesian missing data problems. *Journal of the American statistical association*, 89(425):278–288, 1994.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

Hava T Siegelmann and Eduardo D Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

Sriram Venkatapathy. Nlp tools contest–2008: Summary. *ICON*, 2008.

Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13 (2):260–269, 1967.

Wikipedia. Independent and identically distributed random variables — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Independent_and_identically_distributed_random_variables&oldid=771177118`, 2017. [Online; accessed 21-May-2017].

Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501, 1990.