AN ANALYSIS OF VARIOUS ASPECTS OF RNNs IN SEQUENCE CLASSIFICATION

A Project Report

submitted by

OJAS MEHTA

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

MASTER OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

June 2020

THESIS CERTIFICATE

This is to certify that the thesis titled **An Analysis of Various Aspects of RNNs in Sequence Classification**, submitted by **Ojas Mehta (CS18M038)**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under our 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.

Prof. Harish G. Ramaswamy
Research Guide
Assistant Professor
Deptartment of Computer Science and Engineering
IIT Madras, 600 036

Place: Chennai

Date: July 15, 2020

ACKNOWLEDGEMENTS

The author is grateful to Prof. Harish G. Ramaswamy for providing his extremely valuable inputs, insights, feedback, time and support. The author is indebted to the Computer Science Department, IIT Madras for providing the resources and the Alphabet Inc. for providing the access to the Google Colaboratory. The author would also like to extend the thanks to Avani Shukla for assisting with the code, providing valuable inputs and inspiring through the journey.

ABSTRACT

KEYWORDS: Recurrent Neural Network; Gated Recurrent Unit; Sequence Classification.

The Recurrent Neural Networks (RNNs) are very powerful deep learning models that are popularly used for a large number of applications that often involve arbitrarily long sequences like machine translation, image and video captioning, text summarization, machine vision, speech recognition, etc. Despite this huge popularity of the RNN, there is an insufficient understanding of its components and their working. Huge amount of time and resources are invested in searching the good initialization, finding the right set of hyper parameters or debugging the RNN itself. In this work, we consider the binary classification of the sequences of the three synthetic datasets using Vanilla RNN and Gated Recurrent Unit (GRU). We perform several experiments in order to deeply understand the training process of both the architectures, the role of word embeddings and gating mechanism in it, effects of several initialization strategies, evaluate the trainability and capacity of both the models, their noise robustness and explore their optimization landscape.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
RNN	Recurrent Neural Network

CHAPTER 1

INTRODUCTION

The Recurrent Neural Networks are powerful models that can approximate any discrete or continuous dynamical system as shown in Doya (1993). The RNNs are the extension of Feed Forward Neural network that allows the hidden units to have a recurrent connection which enables them to learn the representation of arbitrarily long sequences. The parameter sharing across the time steps facilitates the handling the variable length sequences. This property makes them suitable for a wide range of applications that involve sequences for example image and video captioning, text summarization, machine vision, speech recognition, machine translation etc. (Salehinejad *et al.*, 2018).

It is widely known that though very powerful, Vanilla RNNs are difficult to train. The key problems making the training of Vanilla RNNs difficult are Exploding and Vanishing gradients (Bengio *et al.*, 1994). This problem becomes worse as the span of temporal dependencies increases. The two major ways in which this problem was addressed are coming up with better architectures and/or improving the training algorithm. The better architectures were proposed like LSTM (Hochreiter and Schmidhuber, 1997), GRU (Cho *et al.*, 2014) etc. that uses gating mechanism mitigate the effect of the problem mentioned above. Likewise, the training algorithm was modified like clipping the gradient norms to address exploding gradients or introducing the soft constraints for the vanishing gradients problem, Hessian-Free optimization was used instead of relying on first order optimizations algorithms (Martens and Sutskever, 2011), various other algorithms like time-weighted pseudo-Newton and discrete propagation algorithms (Bengio *et al.*, 1994).

We aim to a have deeper understanding of the difficulty in training the Vanilla RNNs and how architectures with gating mechanism address it. Such deep and clear understanding can enable more control over the training of RNNs, well informed hyperparameter tuning, debugging RNNs and creation of better initialization strategies. All these will save the large amount of computing resources used in uninformed search of the hyperparameters as well as the time that would be consumed in debugging the RNN or finding the suitable parameter settings.

We perform several experiments to empirically evaluate the trainability and capacity of the Vanilla RNN and GRU. With trainability we mean that how easy or difficult a model is to train. With capacity we mean how much difficult data can be modelled can be represented using models of given input and hidden dimension. For example we would claim the Vanilla RNN to be of the same capacity as that of GRU if there exist an parameter settings for which Vanilla RNN can achieve as good accuracy as achieved by the GRU having same input and hidden dimension for any parameter settings.

We also conduct experiments to analyse the role of linear separability in word embeddings, evaluate various initialization strategies, visualize how the hidden state evolves during inference. We also empirically evaluate the role of gating in GRU and significance of combination of the reset gate and the update gate.

Although RNNs have shown successes in generation as well as classification tasks, we restrict our attention to the binary classification of the sequences using RNNs. We use three synthetic datasets (see Section 3.1 for the detail) whose labelling function exploits the long-term dependency similar to the temporal order problems in Hochreiter and Schmidhuber (1997). Our problems also requires the RNN to remember long term dependencies. Since, we wish to have a comparative evaluation of architecture with gating to the Vanilla RNN, we use GRU for that purpose. The reason we use GRU instead of LSTM is not only because it is simpler and easy to unfold but it is also shown to outperform LSTM Chung *et al.* (2014).

1.1 Vanilla Recurrent Neural Network (Vanilla RNN)

The Vanilla RNN has a hidden state that captures the input sequence of arbitrary length. The hidden state at time t (h_t) is a function of hidden state at time $t-1$ (h_{t-1}) that is at the previous time step and the input at time step t (x_t), and is given by the following equation (Paszke *et al.*, 2019) ,

$$h_t = tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh}) \tag{1.1}$$

The output is then obtained by passing final hidden state to the fully connected output layer as,

$$o_t = W_{oh} h_t + b_{oh}, \tag{1.2}$$

and in our case since we are dealing with the binary classification we then applied the softmax function on o_t and calculated cross entropy for training the Vanilla RNN.

1.2 Gated Recurrent Units (GRU)

GRU has gating internal to its units which provides an alternative channel for the error to propagate but unlike LSTM, it does not maintain a separate memory cell. It requires lesser parameters than LSTM without compromising with the performance.
The hidden state at time step t, h_t is calculated from the candidate hidden step, $\tilde{h_t}$ and previous hidden state, h_{t-1} as

$$h_t = (1 - z_t) \odot \tilde{h_t} + z_t \odot h_{(t-1)}, \tag{1.3}$$

where $\odot$ represents the element-wise product and z_t is the update gate. The candidate hidden state depends on current input and previous hidden state as,

$$\tilde{h_t} = tanh(W_{in} x_t + b_{in} + r_t \odot (W_{hn} h_{(t-1)} + b_{hn})), \tag{1.4}$$

where r_t is the reset gate. The reset and the update gates are calculated as follows,

$$r_t = \sigma(W_{ir} x_t + b_{ir} + W_{hr} h_{(t-1)} + b_{hr}) \tag{1.5}$$

$$z_t = \sigma(W_{iz} x_t + b_{iz} + W_{hz} h_{(t-1)} + b_{hz}) \tag{1.6}$$

The output is calculated in the same way as in the Vanilla RNN (equation (1.2)).

CHAPTER 2

RELATED WORK

Although, Recurrent Neural Network are shown to be a universal approximators (Doya, 1993), training RNN to get that good approximation of a function is a difficult. Bengio *et al.* (1994) has found that as the temporal span of the dependencies increases the training of RNN using gradient descent becomes increasingly difficult. It showed the key reasons to be the exploding and vanishing gradients. It has also compared various other methods like simulated annealing, multi-grid random search etc. to estimate the parameters of the RNN.

Hochreiter and Schmidhuber (1997) and Martens and Sutskever (2011) have discussed about pathological synthetic problem like temporal order problem etc. which involve long-term dependencies and are inherently difficult for RNNs to solve. Hochreiter and Schmidhuber (1997) proposed new architecture called LSTM which addresses the underlying problem of exploding and vanishing gradient through gating mechanism. The gating mechanism internal to the memory cells of the recurrent units providing an alternate channel for constant error flow under some restrictions (Cho *et al.*, 2014).

Martens and Sutskever (2011) used Hessian-Free optimization instead of using some sophisticated RNN units and succeeded in training the Vanilla RNNs to solve pathological problems like the Addition, the Multiplication, the Xor, the Temporal order problem etc. Their detailed description can be found out in Martens and Sutskever (2011). However, the amount of computation required to train RNNs using HF optimization was considerably higher than what was required in training LSTM using some variants of SGD.

Both the solutions mentioned above either used the sophisticated architecture or some alternative to gradient based method like Hessian-Free optimization. Pascanu *et al.* (2012) proposes the solution to exploding gradients by clipping the norm of the gradients and to the vanishing gradients by introducing a regularization term that prevents error signal from vanishing.

However, interestingly Sutskever *et al.* (2013) demonstrated how Vanilla RNN can be trained using carefully tuned momentum based SGD and well-designed random initialization to the level of performance that was previously achievable only with the HF optimization. These results were intriguing as they were contrary to the popularly held belief that first-order methods are not sufficient to train deep or recurrent neural network.

Collins *et al.* (2016) showed that per-parameter capacity of all RNN architectures are roughly comparable with the per-parameter capacity of Vanilla RNN being slightly higher. It also provided empirical evidences signifying the ease of training of any gated architecture over Vanilla RNN. We, in this paper, do further investigation on the trainability and capacity of RNNs.

An interesting work Le *et al.* (2015) proposed a way of initializing Vanilla RNNs with Rectified Linear Units that enables the model to have a comparable performance to that of a standard LSTM. In this scheme of initialization the recurrent hidden weights are initialized with an identity matrix or its scaled version while biases with zeroes. We verify this claim by performing similar experiment on our synthetic datasets.

CHAPTER 3

EXPERIMENT SETUP

3.1 Datasets and Labelling Functions

In this work, the sequences of the dataset are often viewed as the sentences with stop words and non-stop words (positive and negative sentiment words) and hence we use the terms sentences and sequences interchangeably. Then, the sequence classification task is analogous to the sentiment analysis task where the sentences have either positive or negative sentiment which is determined by the labelling function.

We used three labelling function according to which the binary classification of the sentences is carried out. We have named them as the `Count`, `First` and `Xor` labelling function.

In the `Count` labelling function, the sentences are labelled as positive when the number of the positive words is more than that of the negative words and is labelled negative when the number of negative words is more than that of positive words.

In the `First` labelling function, the sentences with first non-stop word as positive are are labelled as positive and the sentences with first non-stop word as negative are labelled as negative sentences.

In the `Xor` labelling function, the sentences with first two non-stop word as different (first is positive and second is negative or vice versa) or same (first and second word both are either positive or negative) are labelled as positive and negative respectively.

There were two versions of the dataset each was split into train, validation and test set. In the first version, the train set had 8000 instances, validation set had 2000 instances and test set had 3000 instances. In the second version, the train set had 5000 instances, validation set had 1000 instances and test set had 1000 instances. It is named as `CountI`, `FirstI` and `XorI`. Embeddings of various dimensions were used in the first version which are explicitly mentioned in the experiment. The second version was used when there were not too many parameters in the model and it is named as

`CountII`, `FirstII` and `XorII`. The version used in each experiment is explicitly mentioned in. Also, the Embedding_2 (Fig. 4.1) is used in all the experiments except the experiment in Section 4.1.1.

The vocabulary is kept very simple with the total of 10 words, in which 6 words are stop words, 2 words are positive sentiment words and 2 words are negative sentiment words. We have empirically observed that the dataset with `Count` labelling function is the easiest, while the one `First` labelling function is moderately difficult and the dataset with `Xor` labelling function is the hardest.

3.2 The Models

We used Vanilla RNN and GRU for the experiments. The training was done for 150 epochs with an additional stopping condition that the training terminates when the validation accuracy does not improve continuously for 4 epochs or learning rate becomes less than 10^{-6}. The Adam optimization algorithm was used. The non-linearities used are as mentioned in the equations in Section 1.1 and Section 1.2 except in the Section 4.4.1 where Rectified Linear Units are used in the place of standard Tanh units and we called that version of the model as ReLU RNN.

The learning rate was kept 0.001 for the sake of uniformity and is halved every time during training when the validation accuracy gets worse. The length of each sequence is fixed to 10 words. After trying various sequence length, we found out that 10 was neither too long nor too small for the the models with the number of parameters we have used. The Models were implemented in PyTorch (Paszke *et al.*, 2019). The accuracy mentioned in the tables are rounded off to two decimal places while in the plots (figures) they are rounded off to the nearest integer.

CHAPTER 4

EXPERIMENTS

4.1 Role of Word Embeddings

There are various kinds of techniques in NLP to map words to vectors which facilitate the learning like Word2vec, GloVe etc. We investigate the role of embedding in the training and inference of the RNN models.

4.1.1 Separability of Word Vectors

The experiment analyzes the effect of separability of the two dimensional vector representations of positive, negative and stop words on the training and inference of the Vanilla RNN and GRU Models.

Experiment Setup

The RNN and GRU both had 2 and 4 hidden dimensions.The `CountII` and `FirstII` datasets were used. The 4 types of embeddings were used with each being 2 dimensional. They are named as $Embedding\_1$, $Embedding\_2$, $Embedding\_3$ and $Embedding\_4$.

In $Embedding\_1$, positive and negative words are not linearly separable (Fig. 4.1(a)).

In $Embedding\_2$, positive and negative words are linearly separable but stop words spread all over in random fashion (Fig. 4.1(b)).

In $Embedding\_3$, positive words, negative words and stop words are distributed linearly as three separate cluster (Fig. 4.1(c)).

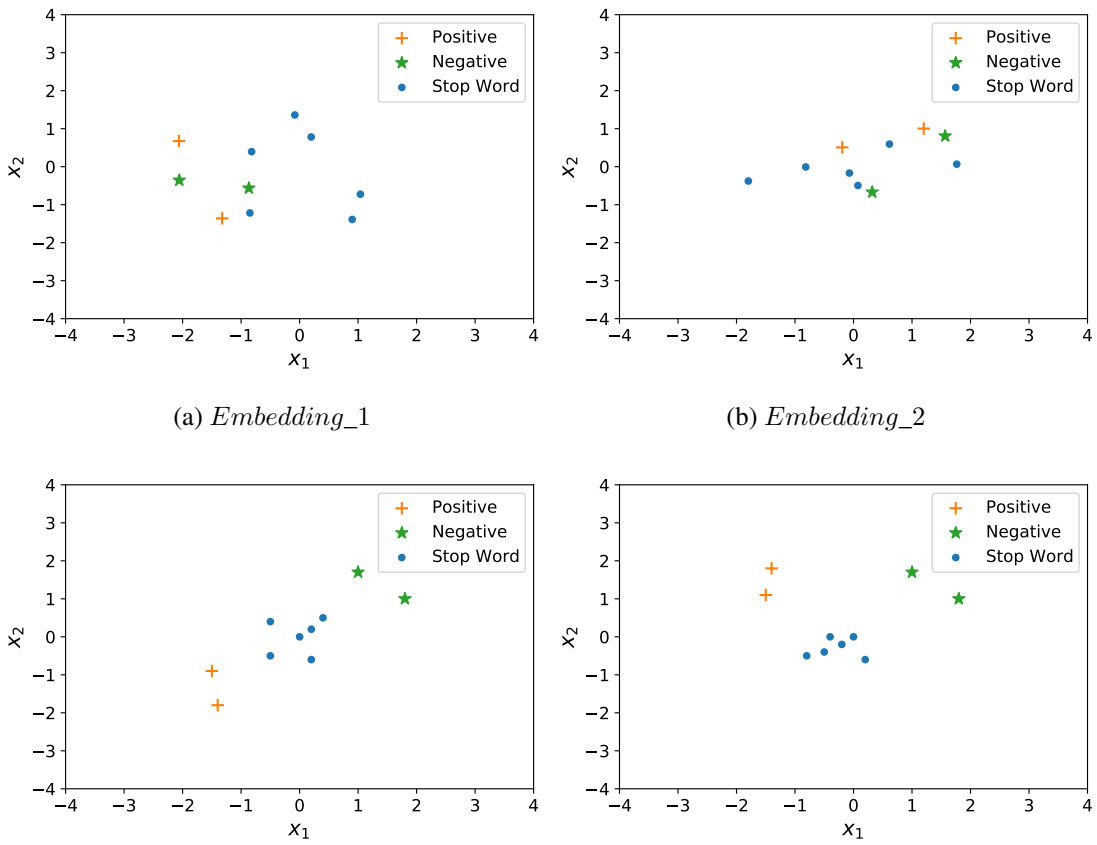
In $Embedding\_4$, positive words, negative words and stop words are pairwise linearly separable (Fig. 4.1(d)).

Results and Analysis

The Validation Accuracies are highest when Embedding_4 is used followed by models that used Embedding_3, then Embedding_2 and lowest in the case of Embedding_1 for both Vanilla RNN and GRU models and both `CountII` and `FirstII` datasets (Fig. 4.2 and 4.3).

This implies that the embeddings in the increasing order of their ability to be learnt easily by a model are Embedding_1 (hardest), followed by Embedding_2, Embedding_3 and Embedding_4 (easiest).

So it can be inferred that it is easiest for the model to learn when the positive, negative and stop words are pairwise linearly separable while it is hardest when no two of the three kind of words are linearly separable (Fig. 4.1). Thus, linear separability of different kinds of words facilitates the training process.



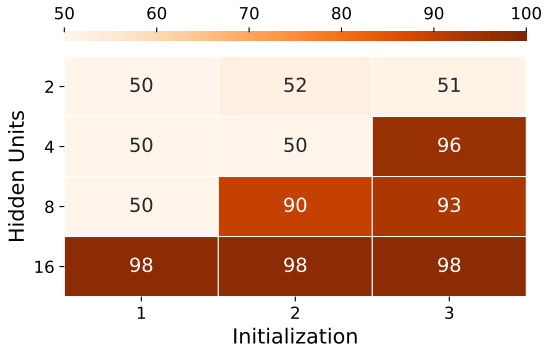
(a) *Embedding_1*

(b) *Embedding_2*

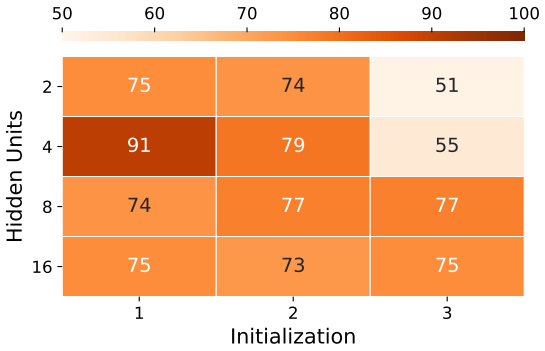
(c) *Embedding_3*

(d) *Embedding_4*

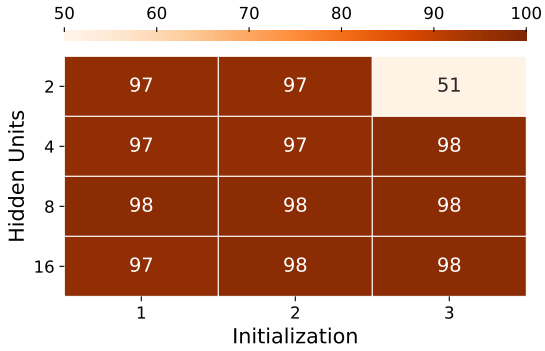
Figure 4.1: Four two-dimensional embeddings used in the experiment in the Section 4.1.1.



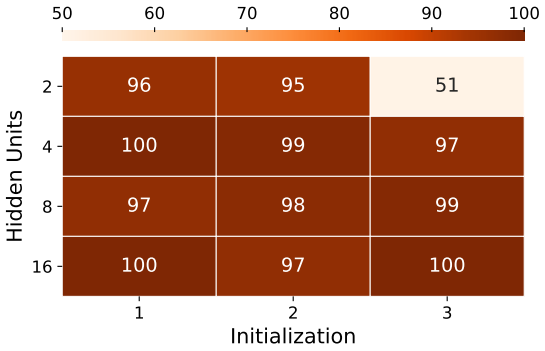
(a) $Embedding\_1$ (Vanilla RNN)



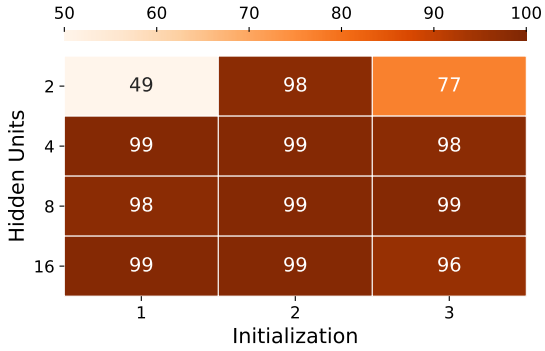
(b) $Embedding\_2$ (Vanilla RNN)



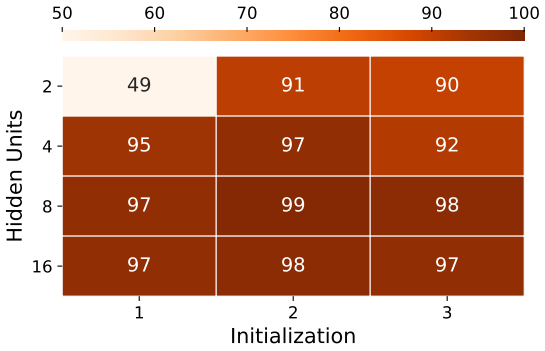
(c) $Embedding\_3$ (Vanilla RNN)



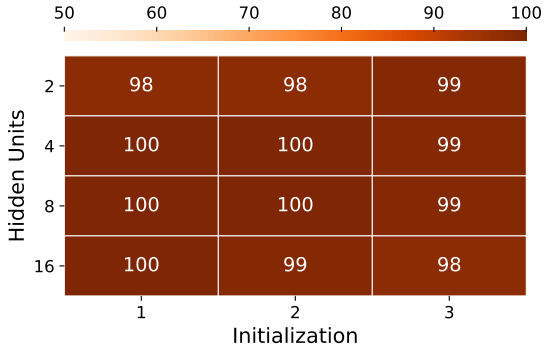
(d) $Embedding\_4$ (Vanilla RNN)



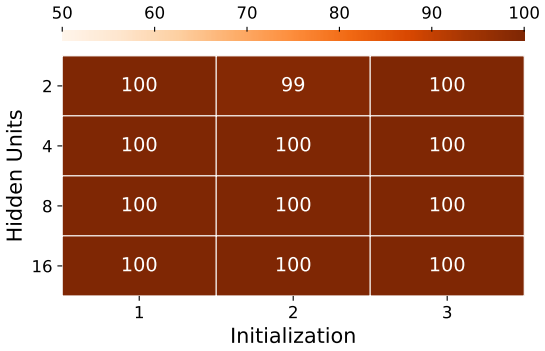
(e) $Embedding\_1$ (GRU)



(f) $Embedding\_2$ (GRU)

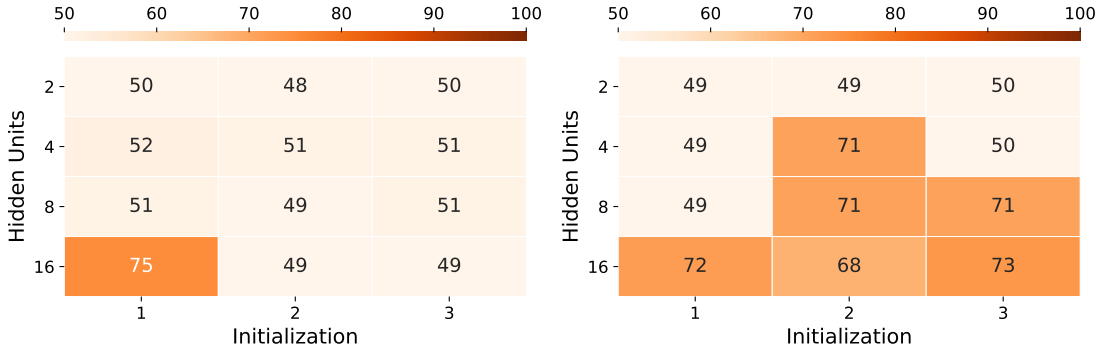


(g) $Embedding\_3$ (GRU)



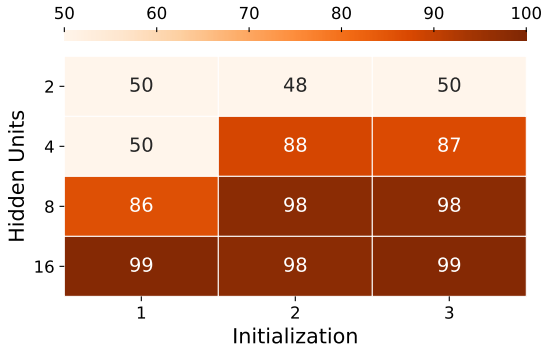
(h) $Embedding\_4$ (GRU)

Figure 4.2: The heatmap in each subfigure represents the Validation Accuracies (rounded off to the nearest integer) obtained by a given Vanilla RNN or GRU model for the given hidden dimension with the three different initializations on the specified embedding of `CountII` dataset.

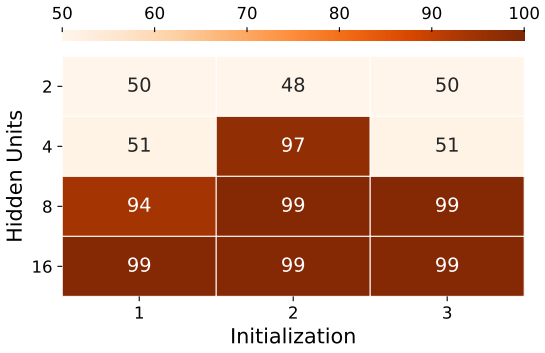


(a) *Embedding_1*

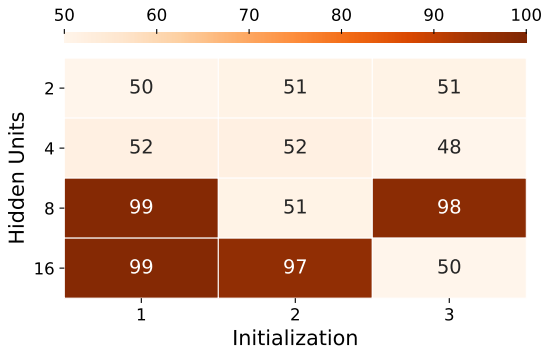
(b) *Embedding_2*



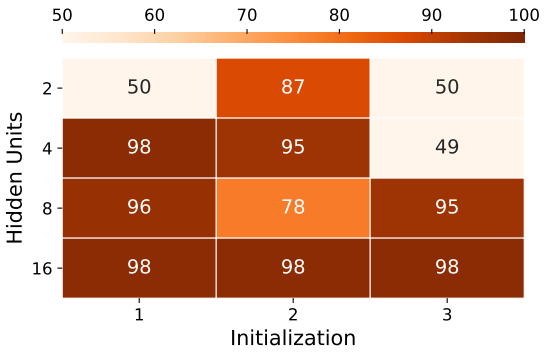
(c) *Embedding_3*



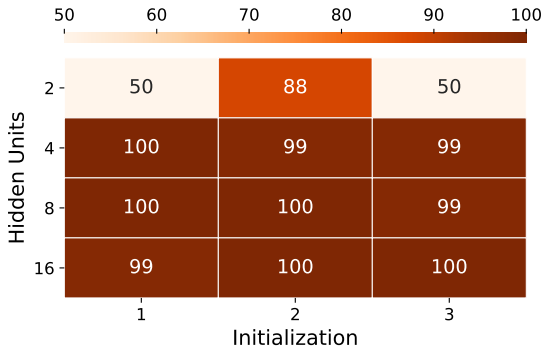
(d) *Embedding_4*



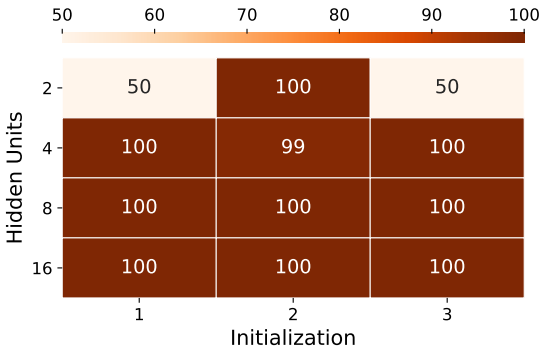
(e) *Embedding_1*



(f) *Embedding_2*



(g) *Embedding_3*



(h) *Embedding_4*

Figure 4.3: The heatmap in each subfigure represents the Validation Accuracies (rounded off to the nearest integer) obtained by a given Vanilla RNN or GRU model for the given hidden dimension with the three different initializations on the specified embedding of `FirstII` dataset.

4.2 Trainability and Capacity of RNNs

Two key features of any deep learning model are the ease with which a model can train (trainability) and the amount of information it can represent (capacity). We investigate and compare the trainability and capacity of Vanilla RNNs and GRUs in this section.

4.2.1 Trainability of the Vanilla RNN and GRU

The trainability of the RNNs is nothing but the ease with which RNNs can be trained. The way in which we evaluate the trainability of different models can be understood by the a simple example. Suppose two models are trained 100 times each time with a different initialization and model A succeeds (achieves high validation/test accuracy) more number of times than model B, model A is said to have high trainability. We do the comparative study of the trainability of the RNN and GRU in this section.

Experiment Setup

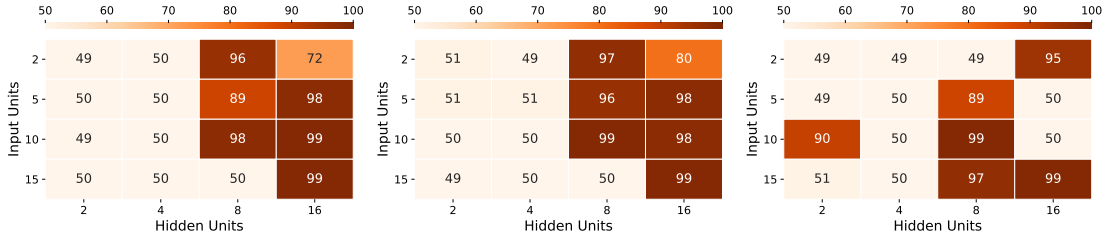
We trained the Vanilla RNN and GRU models with all the 16 combinations of the input dimension as {2, 5, 10, 15} and hidden dimension as {2, 4, 8, 16}, each with 3 different initializations. We used both the `FirstI` and `XorI` dataset for the experiment.

Results and Analysis

The Figures 4.4 and 4.5 contains the plots for this experiment. In terms of the performance, GRU is the clear winner as it succeeds far more times than the Vanilla RNN. This superior performance of GRU implies that GRUs are easier to train compared to Vanilla RNN and not that they are more powerful (we show that in Section 4.2.2).

In other words, Vanilla RNNs are much more susceptible to the initialization. For instance, it achieved 90% accuracy with 10 input units and 2 hidden units (Fig. 4.4(c)) but could not achieve more than 51% accuracy in the first two columns even when the model had equal or more input or hidden units with all the three initializations (Fig. 4.4(a),(b),(c)). Similarly, in the Figure 4.5, RNN could achieve decent accuracy in only 1 out of the 48 different architectures.

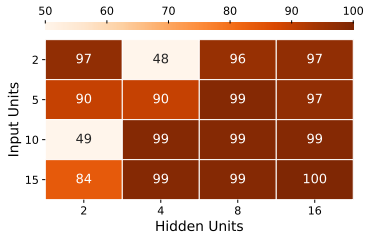
This also suggests that `XorI` dataset is harder to learn than the `FirstI` dataset.



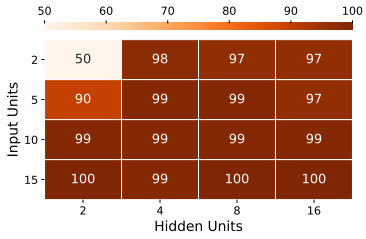
(a) Vanilla RNN (1)

(b) Vanilla RNN (2)

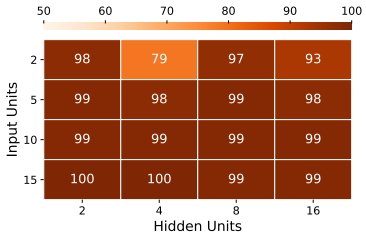
(c) Vanilla RNN (3)



(d) GRU (1)

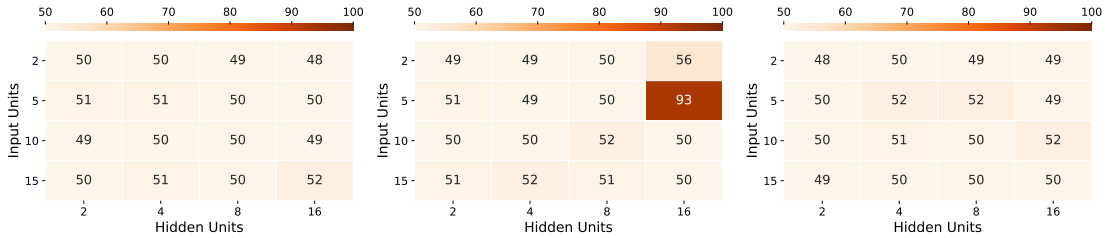


(e) GRU (2)



(f) GRU (3)

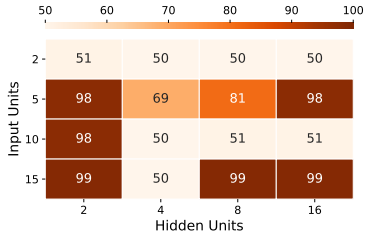
Figure 4.4: The heatmap in each subfigure represents the Validation Accuracies (rounded off to the nearest integer) of the model with a random initialization for the given number of input and hidden units on the `FirstI` dataset.



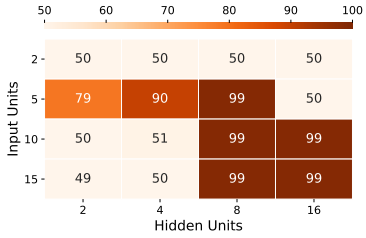
(a) Vanilla RNN (1)

(b) Vanilla RNN (2)

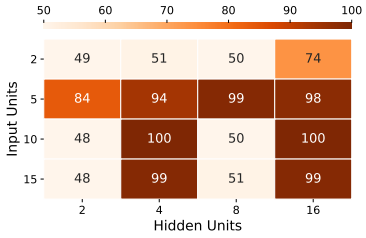
(c) Vanilla RNN (3)



(d) GRU (1)



(e) GRU (2)



(f) GRU (3)

Figure 4.5: The heatmap in each subfigure represents the Validation Accuracies (rounded off to the nearest integer) of the model with a random initialization for the given number of input and hidden units on the `XorI` dataset.

4.2.2 Capacity of the Vanilla RNN and GRU

The Capacity of the RNNs can be defined as the information it can represent per parameter. The way in which we evaluate the capacity is the ability of the model to train and achieve high accuracy with the given number of parameters. Suppose a model with specific architecture achieves high validation accuracy in even one of the initialization out of several different initializations it is said to have the capacity to model that dataset.

In this experiment, we did the comparative study of the capacity of the Vanilla RNN and GRU. We trained both Vanilla RNN and GRU on the `FirstI` and `XorI` dataset for different number of input and hidden dimension and plot heatmap of the validation accuracies obtained in each case. We picked some of the critical input and hidden dimensions based on the results obtained in the experiment in Section 4.2.1. We trained the models with the same architectures across 30 different initializations to check if the ones which were failing on a particular initialization, succeeded in one of the many other initializations to check whether they are capable and for those which succeeded how often did they succeed.

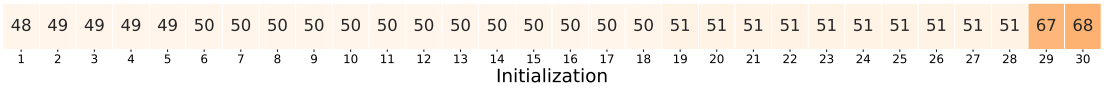
Experiment Setup

We considered the architectures with all the four combinations of input dimension and hidden dimension as 2 and 4. We picked these architectures because the Vanilla RNN could not train successfully for the comparable dimension (combinations of input dimension and hidden dimension like {(2, 2), (2, 5), (5, 2), (5, 5)}) but GRU could.
So, the smaller architectures could be key in investigating whether the poor performance was a result of just a bad initialization (training difficulty) or the model was not even capable to represent the labelling function in that number of parameters (Capacity). We used both `FirstI` and `XorI` datasets for the experiment.

Results and Analysis

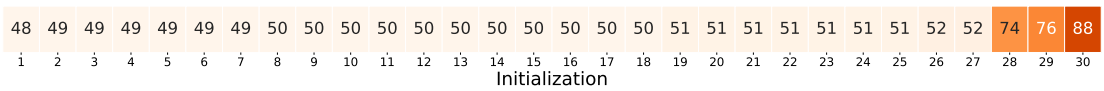
The Figure 4.6 contains the results of this experiment. The fact that Vanilla RNN could also achieve the high validation accuracy in the less number of parameters (like 96% in (2x4), 97% in (4x4)) where it was not able to achieve that high accuracy in the previous

experiment while GRU could, suggests that Vanilla RNNs are equally capable to represent the information in the same number of units as GRU.

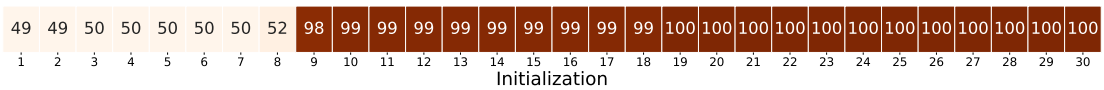


(a) Vanilla RNN (2 x 2)

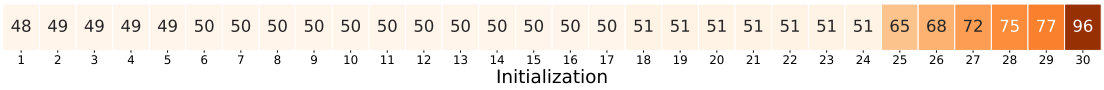
(b) GRU (2 x 2)



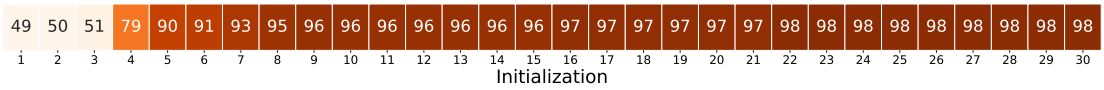
(c) Vanilla RNN (4 x 2)



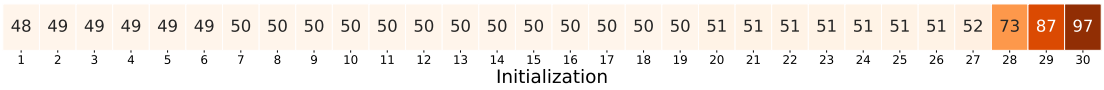
(d) GRU (4 x 2)



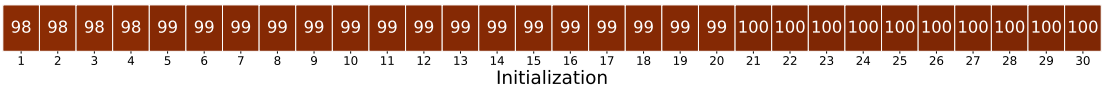
(e) Vanilla RNN (2 x 4)



(f) GRU (2 x 4)

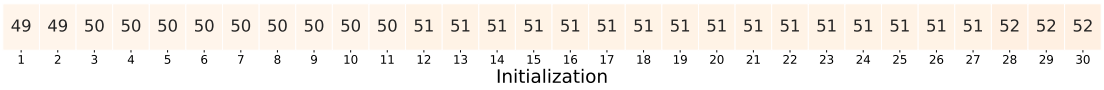


(g) Vanilla RNN (4 x 4)

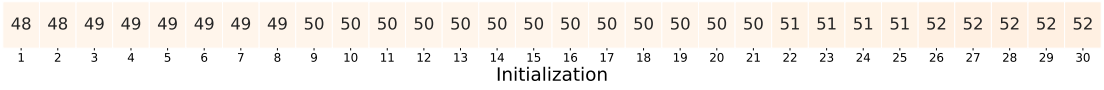


(h) GRU (4 x 4)

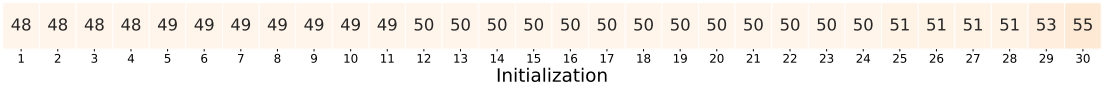
Figure 4.6: Each subfigure shows the Validation Accuracies (rounded off to the nearest integer) obtained by the models across 30 different initializations on the `FirstI` dataset. The subfigure name follows the following naming convention, ModelType (Input Dimension x Hidden Dimension).



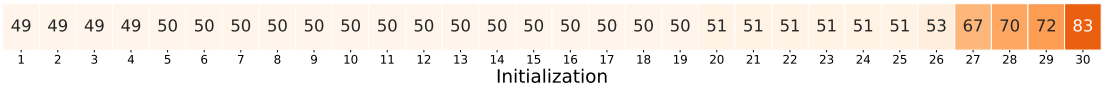
(a) Vanilla RNN (2 x 2)



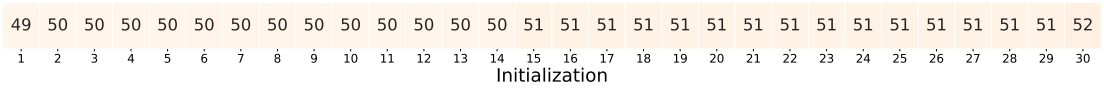
(b) GRU (2 x 2)



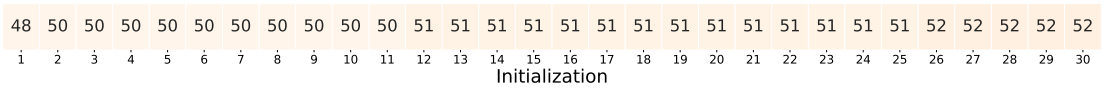
(c) Vanilla RNN (4 x 2)



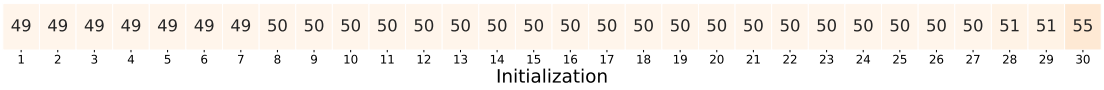
(d) GRU (4 x 2)



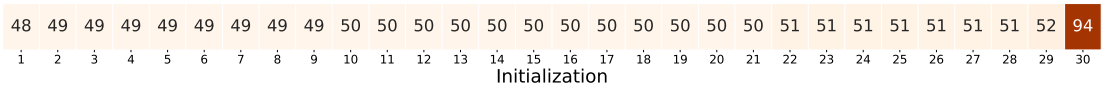
(e) Vanilla RNN (2 x 4)



(f) GRU (2 x 4)



(g) Vanilla RNN (4 x 4)



(h) GRU (4 x 4)

Figure 4.7: Each subfigure shows the Validation Accuracies (rounded off to the nearest integer) obtained by the models across 30 different initializations on the `XorI` dataset. The subfigure name follows the following naming convention, ModelType (Input Dimension x Hidden Dimension).

In other words, Vanilla RNN and GRU have the similar capacity but GRUs are easier to train (Section 4.2.1). It also affirms our conclusion from the experiment in Section 4.2.1 that Vanilla RNNs are highly susceptible to the initialization as good initializations in the Vanilla RNNs are much more rare compared to those in the GRU.

4.3 Behaviour of Components of RNNs

A prevalent way to evaluate a model is by its accuracy on train and test set. But this single metric is not enough to explain the intricacies of the model like why it works the way it works. This shallower way often leads to incomplete understanding of the behaviour of the RNN models.

In this section, we look at the aspects like how the hidden state evolves during sequence classification, word vectors of stop and non-stop words are transformed after various operations in the network and distinguished etc. This could potentially provide more insights which can help to diagnose the issues with the architecture of the model, associated hyper parameters and training process.

We interchangeably use the word sentence and sequence. The classification of the sentences is analogous to the sentiment analysis task. We have the vocabulary of stop words and non-stop words and we classify the sentences as either the positive sentiment sentence or the negative sentiment sentence.

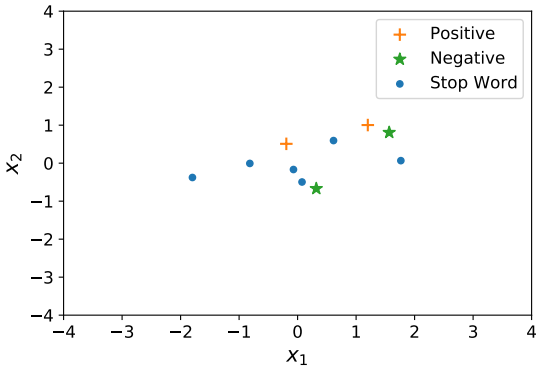
4.3.1 Characterization of Stop Words and Non-Stop Words by the Model

In order to classify a sentence as positive or negative sentence, the model should somehow be able to distinguish between stop and non-stop words occurring in it. Based on how the `Count` labelling function labels the sentences, in order to correctly classify the sentence, the stop words should not affect the output label as if there were no stop words present. We expected that the effect of non-stop words on the hidden state should be significant and stop words would have the negligible effect on the hidden state. Ideally, stop words should belong to the null space of the input matrix, W_{ih} Mathematically, $W_{ih}\mathbf{x_{stop}} = \tilde{\mathbf{0}}$. We verify this hypothesis in this experiment.

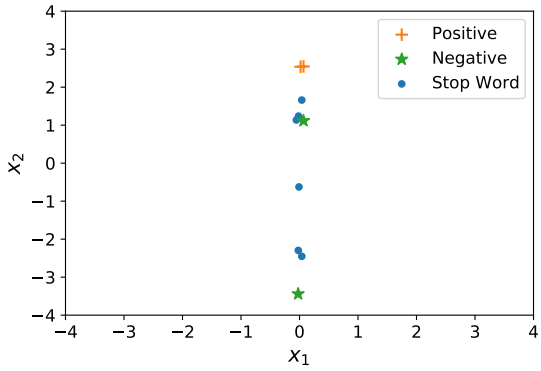
Experiment Setup

The RNN and GRU both with 2 input and 2 hidden dimensions. RNN was trained until it got the validation accuracy of 87% while GRU could get 100%. The `CountI` dataset

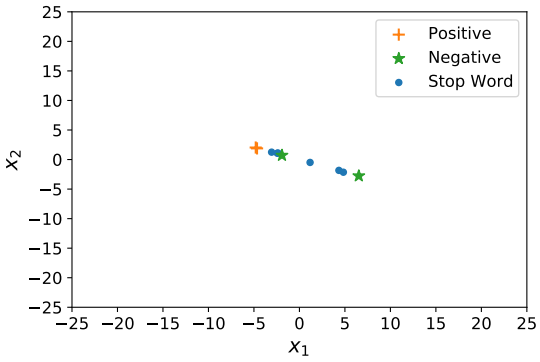
was used.



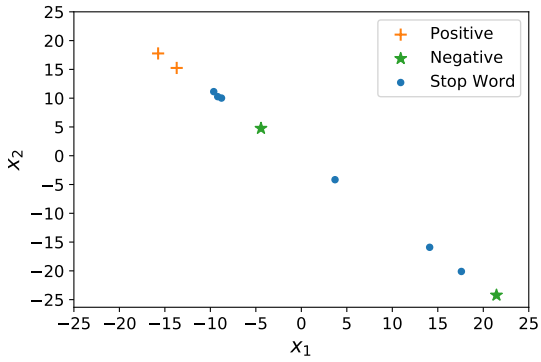
(a) Word Embeddings



(b) Vanilla RNN



(c) Vanilla RNN



(d) GRU

Figure 4.8: Both the input and hidden dimension are 2. (a) The word embeddings for entire vocabulary of 10 Words. (b) The product of Input matrix and Words (W_{ih}.x). (c), (d) The product of Output matrix, Input matrix and words($W_{oh}.W_{ih}$.x).

Results and Analysis

The Figure 4.8 represents the plots for the experiment. We had expected the stop words to belong to the null space of the input matrix in the case of Vanilla RNN. But in Figure 4.8(b), we did not observe this. Instead the product lies along the one dimension and the magnitude along x_1 is very close zero.

In Figure 4.8(c), (d), the value $W_{oh}.W_{ih}$.x lies roughly on a straight line for both Vanilla RNN and GRU.

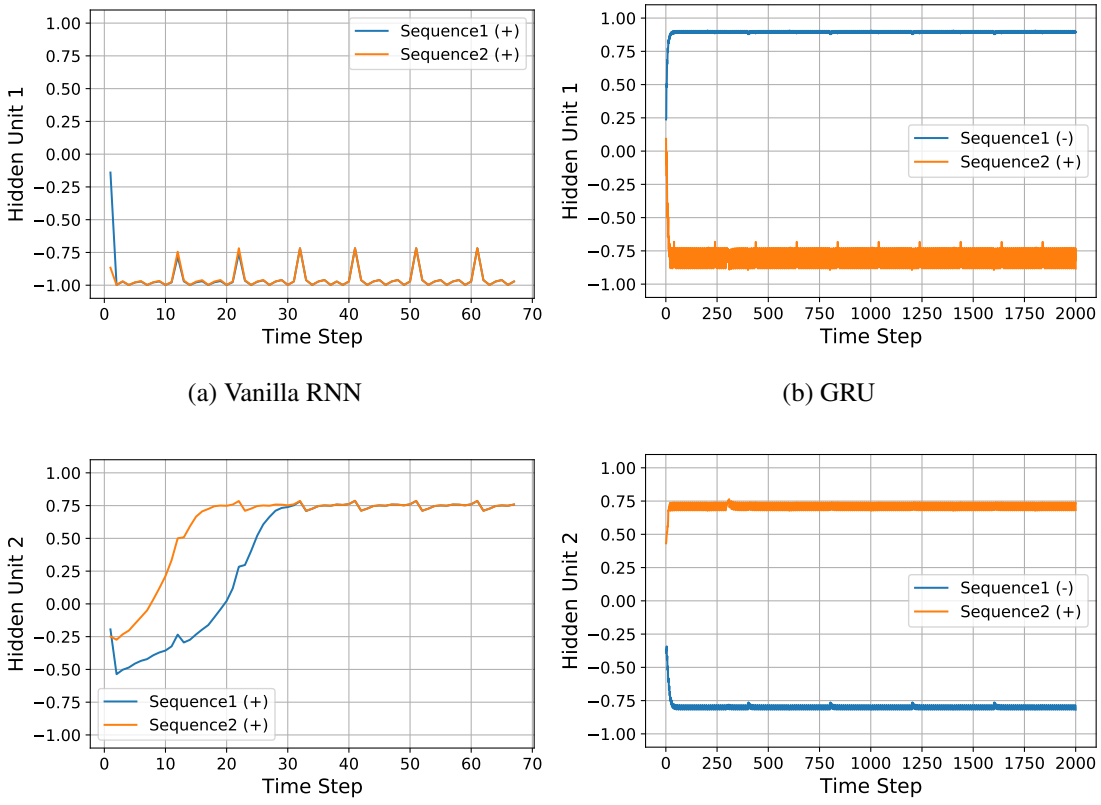
Also, note that the word vectors of the positive and the negative words are linearly separable. They continue to remain so after the above mentioned operations are performed.

4.3.2 Noise Robustness

The ability of a model to ignore the irrelevant input during classification task is key in making the accurate predictions. In this task, the stop words act as a noise as they do not affect the label of a sequence and hence the model should be able to ignore or at least do not let them affect the prediction of the sentiment of any sentence. We study how the hidden state of RNNs react to the stop and non-stop words.

Experiment Setup

The Vanilla RNN and GRU models used have input dimension and the hidden dimension as 4 and 2 respectively. Both the models are trained on the `CountI` dataset and achieved the validation accuracy greater than 98%.



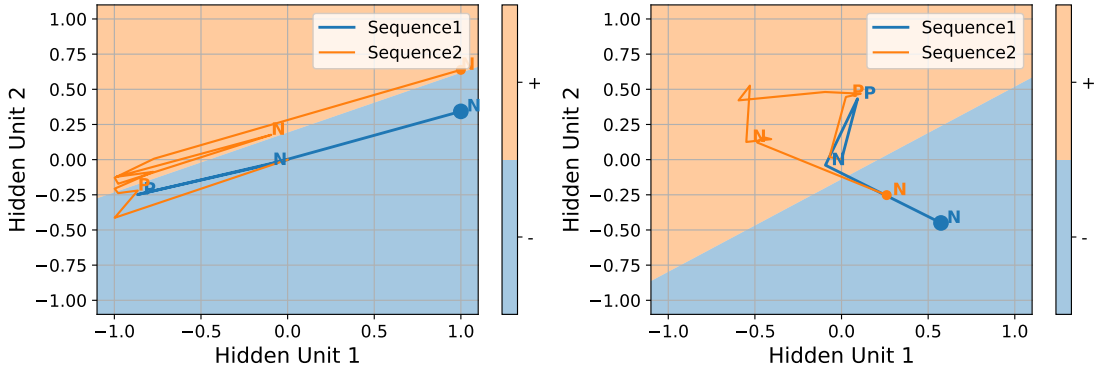
(a) Vanilla RNN

(b) GRU

(c) Vanilla RNN

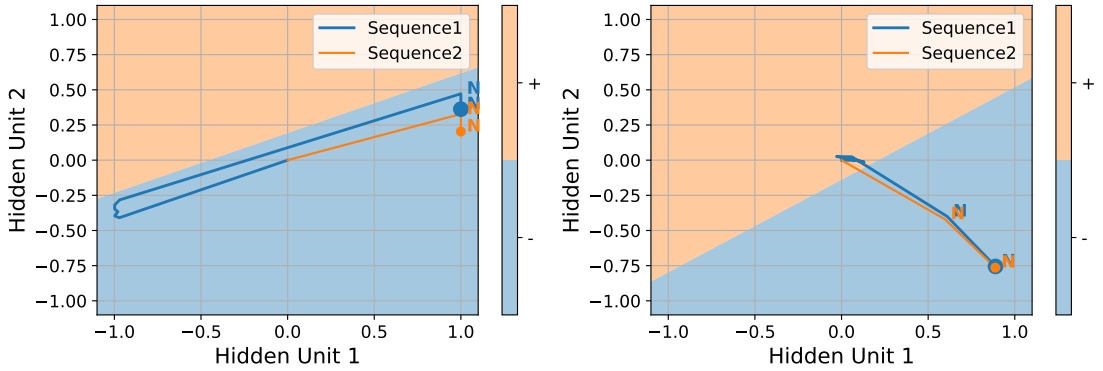
(d) GRU

Figure 4.9: The figures represent how the each individual hidden unit of the hidden state of a well trained model evolves to classify the sequence of words (word by word). The (+) or (-) in the legend indicates whether the sequence was classified as positive or negative by the model respectively. The *Sequence1* = [N, S, S, S, S, ...] & *Sequence2* = [P, S, S, S, S, ...]. (P = Positive, N = Negative, S = Stop Word).



(a) Vanilla RNN

(b) GRU



(c) Vanilla RNN

(d) GRU

Figure 4.10: The plots are the state space plots which show how the hidden state of a well trained model evolves to classify the sequence of words (word by word). The (+) or (-) in the colorbar indicates the positive and negative sentiment respectively. For (a), (b) the *Sequence1* = [P, N, N] & *Sequence2* = [S, P, S, S, S, S, N, S, S, S, N] and for (c), (d) the *Sequence1* = [S, S, S, S, S, S, N, N] & *Sequence2* = [N, N]. Here P represents positive word, N represents negative word and stop word is not marked. The starting point is (0, 0) while the solid circle represents the end of the sequence.

Results and Analysis

In Figure 4.10(a), (c), the Sequence1 has some non-stop words and in the Sequence2 has the same corresponding non-stop words in same sequence with some noise in the form of stop words added to the sequence. In Figure 4.10(a), the sequence 2 is wrongly classified as the positive by Vanilla RNN while correctly classified by GRU as negative (4.10(b)). In Figure 4.10(c), the hidden state of the Vanilla RNN is vastly different at the end of both the sequence despite having the same sequence of non-stop words while the hidden state of GRU is very close in Figure 4.10(d) and stop words also did not change the state much in the beginning of the Sequence2. This clearly indicates that

GRU models are more robust to noise and hence ignore the noise more effectively than Vanilla RNN models.

In Figure 4.9, where the only the first word is non-stop (positive in one and negative in the other sequence and rest of the words are stop words, the Vanilla RNN could only classify correctly till the word 15 then it misclassified for the time steps (words) more than 15. and the states overlap at around word 30. On the other hand GRU for the same sentences could classify correctly even till the 2000 time steps completely ignoring the noise. This suggests that the robustness of the GRUs to the noise is generalizable even to the sequences much larger than what it was trained upon.

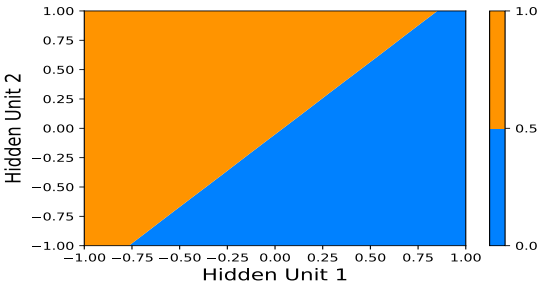
4.3.3 Vector Field for the Hidden State

The hidden units of the RNNs change from one state to another state (may remain in the same state) as it encounters new inputs. When there are two hidden units this behaviour can be visualized using vector field. These interesting visualizations can help in discovering how successful and unsuccessful model react differently to positive, negative and stop words. In this experiment, we plot the vector field and contours of norms of the vectors of the vector field and analyze them.

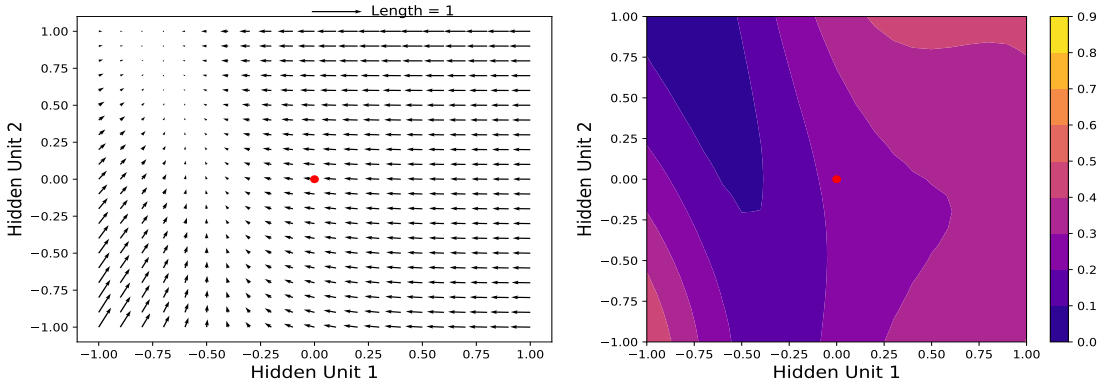
Experiment Setup

Two GRU models one successful that achieved 99.3% accuracy and the other unsuccessful model that achieved 51.2% accuracy were trained on the `CountII` dataset.
Also, two more GRU models one successful that achieved 92.6% accuracy and the other unsuccessful model that achieved 50.0% accuracy were trained on the `FirstII` dataset.
The hidden units in all the models are 2 for the sake of better visualization on the plots.

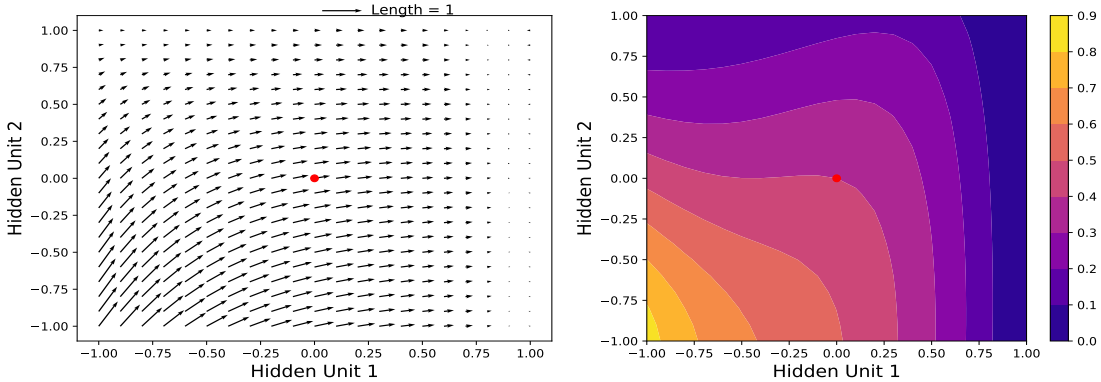
A vector in the Vector field plot of hidden state for a word represents the transition of hidden state from one state to another after it encounters that word. The corresponding contour plot represents the norms (magnitude) of those vectors.



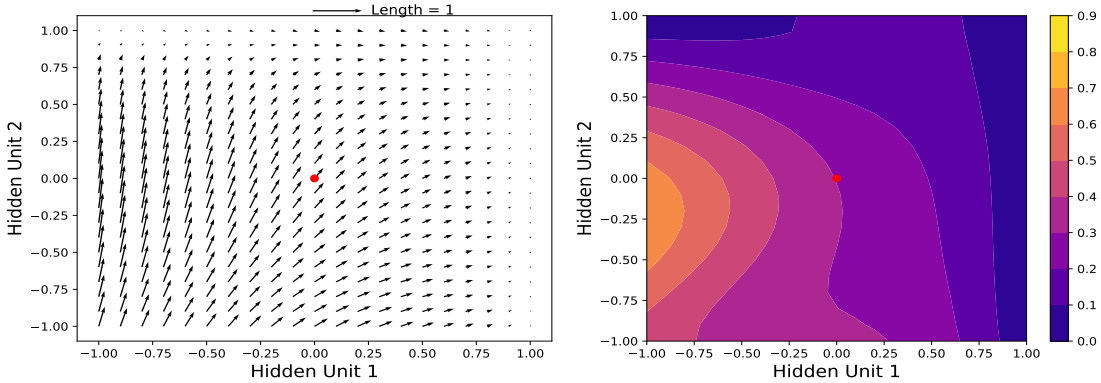
(a) Decision Region



(b) Positive Word

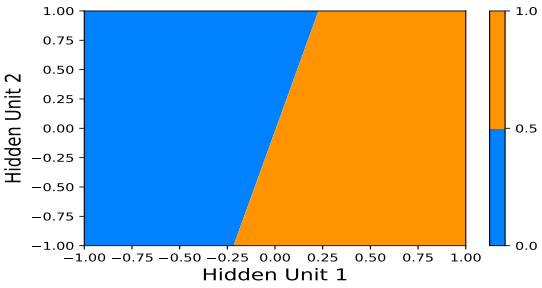


(c) Negative Word

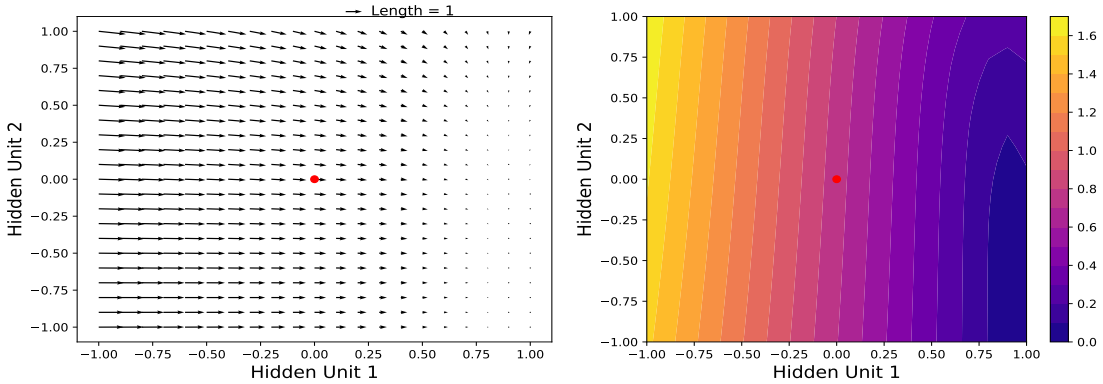


(d) Stop Word

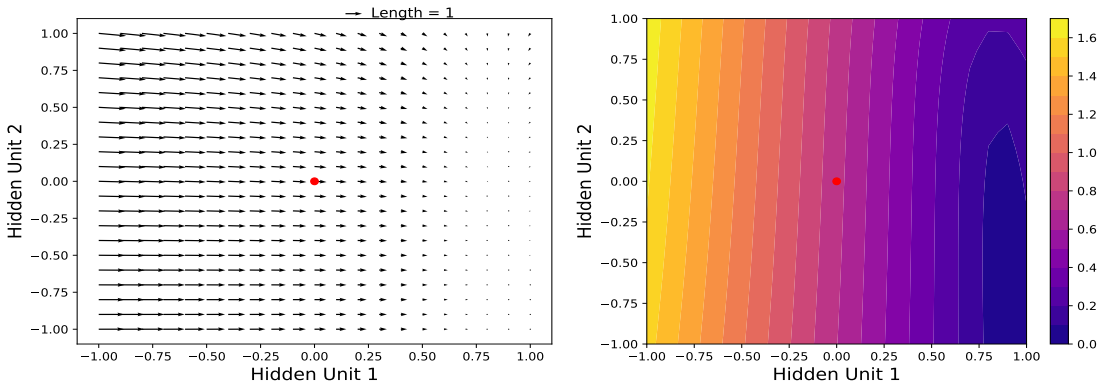
Figure 4.11: Figure (a) represents the decision region (Orange = Positive, Blue = Negative) while Figure (b), (c) and (d) represent the vector field and corresponding contour plots of a Positive, Negative and Stop Word respectively for a GRU model successfully trained on the `CountII` dataset.



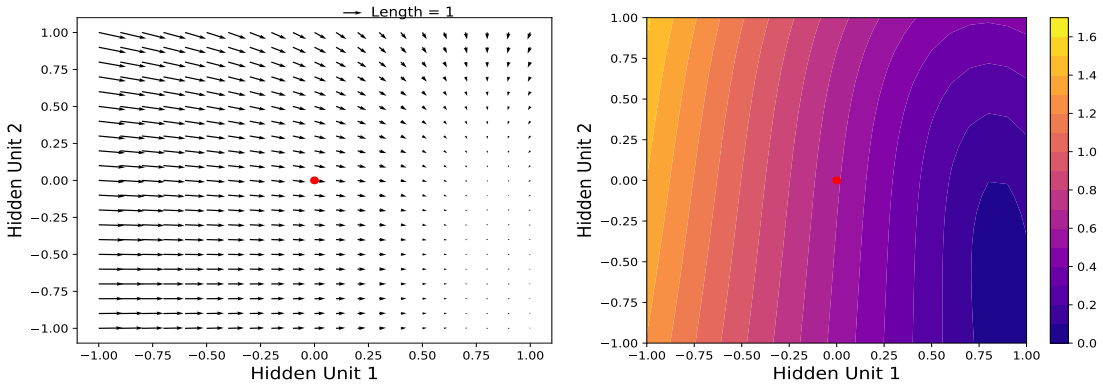
(a) Decision Region



(b) Positive Word

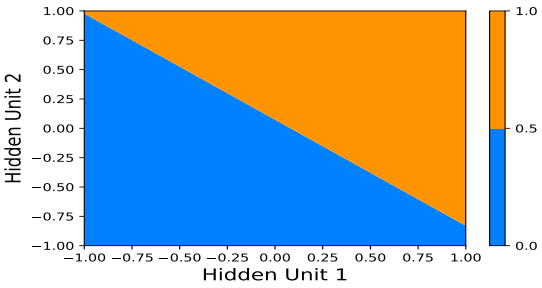


(c) Negative Word

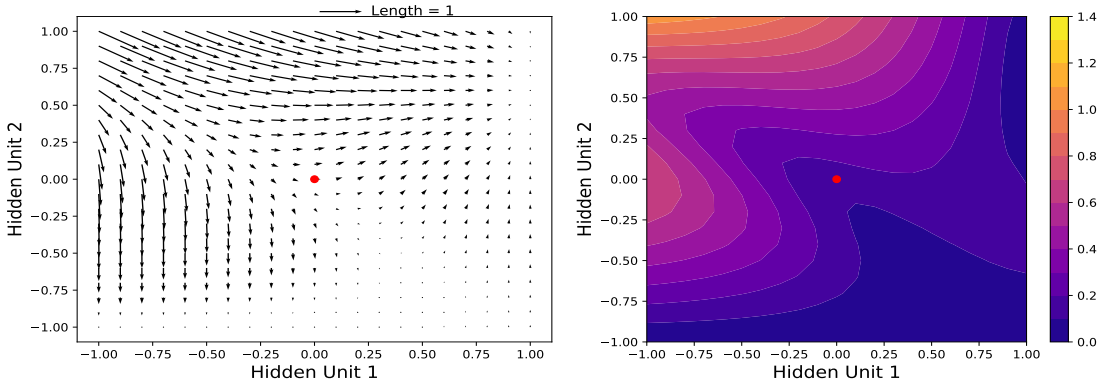


(d) Stop Word

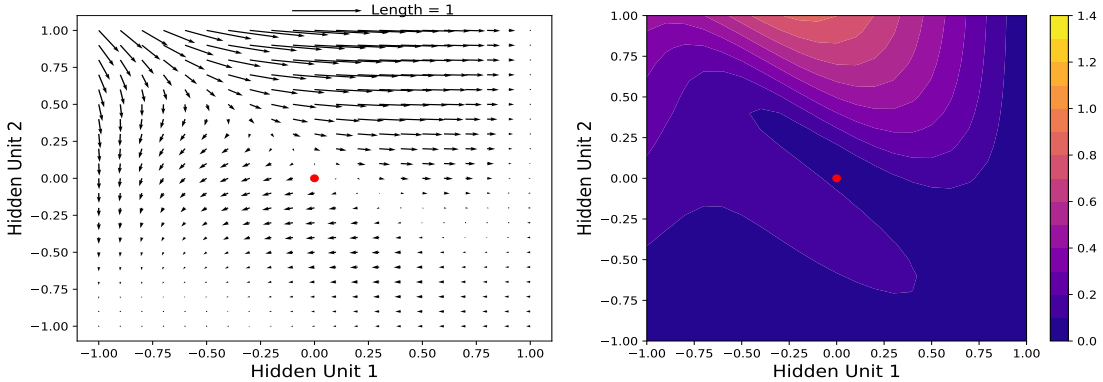
Figure 4.12: Figure (a) represents the decision region (Orange = Positive, Blue = Negative) while Figure (b), (c) and (d) represent the vector field and corresponding contour plots of a Positive, Negative and Stop Word respectively for a GRU model that failed to train on the `CountII` dataset.



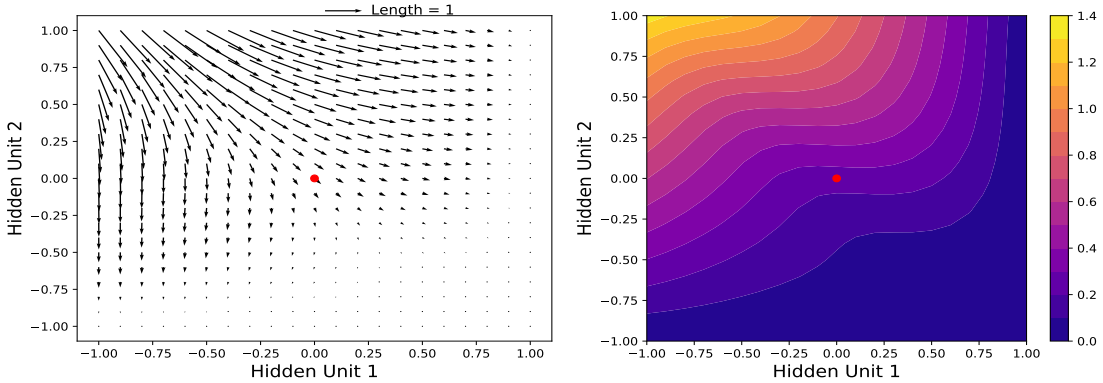
(a) Decision Region



(b) Positive Word

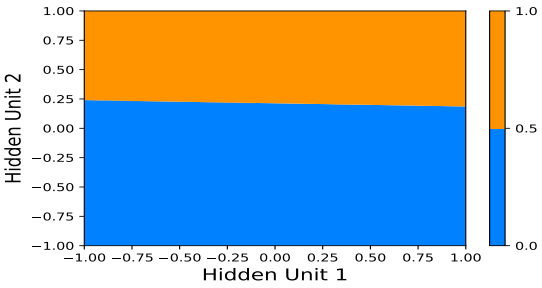


(c) Negative Word

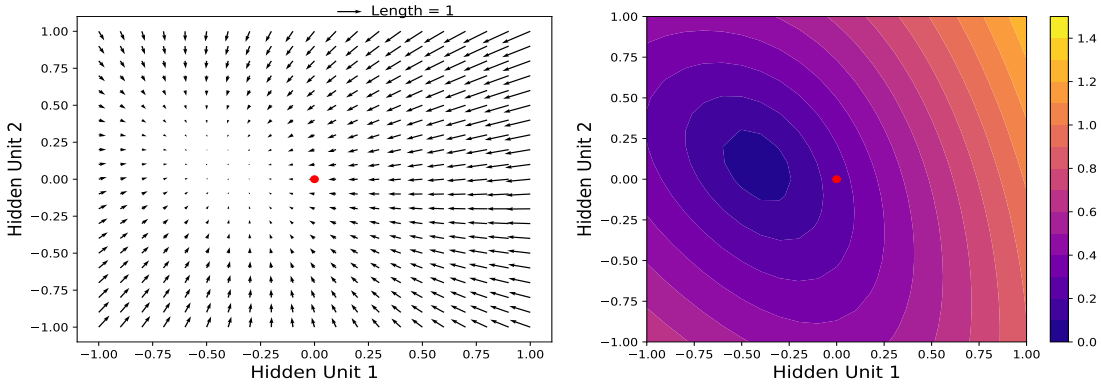


(d) Stop Word

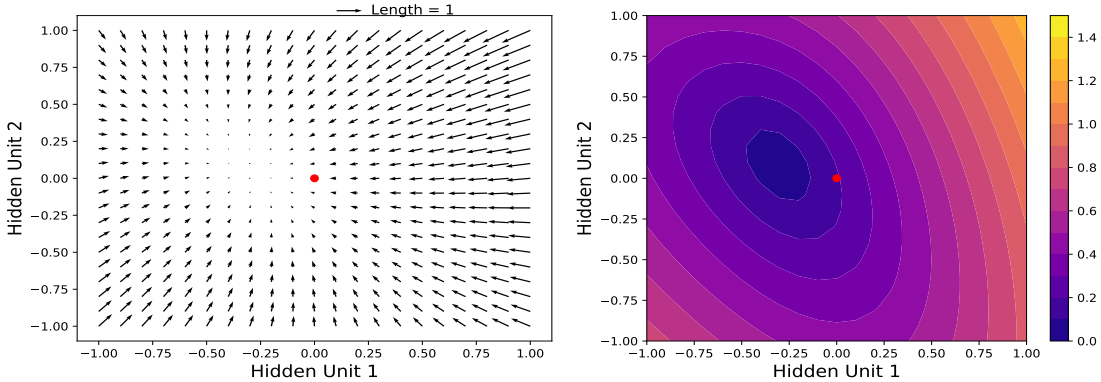
Figure 4.13: Figure (a) represents the decision region (Orange = Positive, Blue = Negative) while Figure (b), (c) and (d) represent the vector field and corresponding contour plots of a Positive, Negative and Stop Word respectively for a GRU model successfully trained on the `FirstII` dataset.



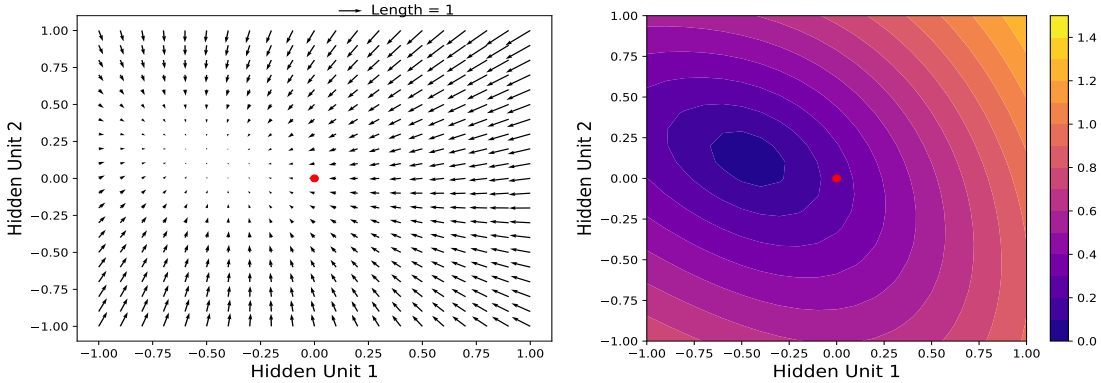
(a) Decision Region



(b) Positive Word

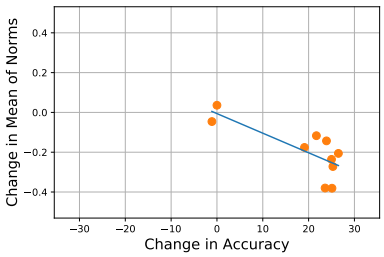


(c) Negative Word

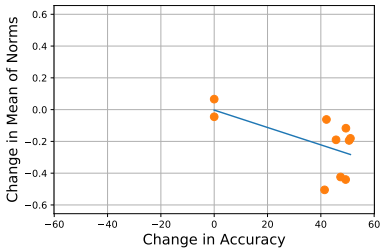


(d) Stop Word

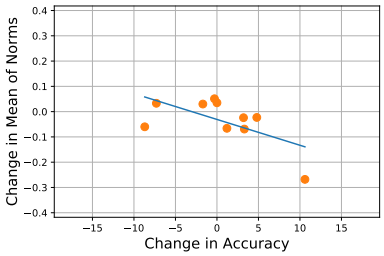
Figure 4.14: Figure (a) represents the decision region (Orange = Positive, Blue = Negative) while Figure (b), (c) and (d) represent the vector field and corresponding contour plots of a Positive, Negative and Stop Word respectively for a GRU model that failed to train on the `FirstII` dataset.



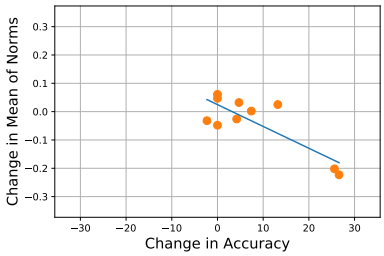
(a) Vanilla RNN (`CountII`)



(b) GRU (`CountII`)



(c) Vanilla RNN (`FirstII`)



(d) GRU (`FirstII`)

Figure 4.15: Each subfigure represents the scatter plot where each point represent the change in mean of norms of the vectors of the vector field (Fig. 4.11, 4.12, 4.13 & 4.14) and its corresponding change in accuracy of a particular word. The blue line is the line of best fit.

Results and Analysis

A pattern is visible in the Vector Fields of the successful models (Fig. 4.11, 4.13) while it seems random in the case of unsuccessful models (Fig. 4.12, 4.14).

In the case of successful GRU model on the `CountII` dataset (Fig. 4.11), for any positive and negative word the vector direct the hidden state towards the positive and negative region respectively, irrespective of the current state. The vectors in case of the stop word keep the state to the region of same sentiment as its current state.

In the case of successful GRU model on the `FirstII` dataset (Fig. 4.13), the stop and non-stop words behave similarly when away from the decision boundary in the way that the vector keeps the state to the region of same sentiment as its current state. When current state is very close to the decision boundary positive and negative words direct the hidden state towards the positive and negative decision region respectively.

The Figure 4.15 shows how the change in mean of the norms of the vectors of a vector field is negatively correlated to the change in accuracy of the model as a result of the training (difference of the initial and final validation accuracy).

4.4 Initialization Strategies

The initialization plays a crucial role in determining whether a model will succeed. It is often noticed that for some tasks while some initializations work others may fail. Training for several initialization to find the one that works is resource intensive task. Therefore, discovering the effective initializing scheme that works more often or a way to evaluate good and bad initialization is of great value. To understand more about initialization schemes of RNNs we conduct several experiment in this section.

4.4.1 Initializing Vanilla RNN of ReLU with Identity matrix

With regard to initializing the Vanilla RNN, Le *et al.* (2015) has proposed a scheme which yielded comparable performance to that of an ordinary LSTM. In this proposed scheme the recurrent hidden weights of the Vanilla Recurrent Neural Networks with Rectified Linear Units were initialized with an identity matrix and biases are set to zeros.

We use this scheme of initialization on our synthetic dataset in this experiment and analyze the performance of models.

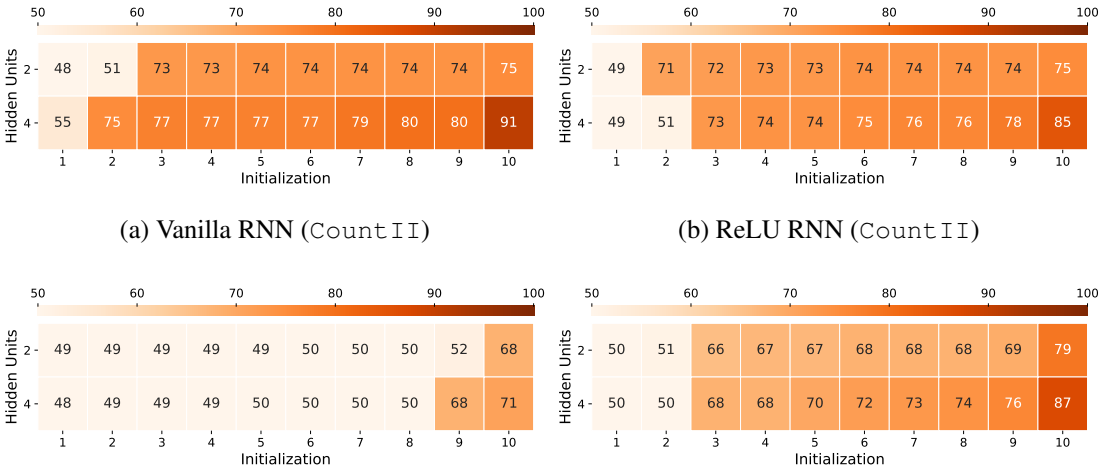
Experiment Setup

Two models are used. One is Vanilla RNN and the other is with slight modification as suggested in Le *et al.* (2015) that is referred in this section as ReLU RNN in Figure 4.16. ReLU RNN is similar to Vanilla RNN with Rectified Linear Units at the place of standard tanh units. The datasets used are `CountII` and `FirstII`.

Results and Analysis

The Figure 4.16 contains the plots for this experiment. The difference in the accuracies in the case of `CountII` dataset is not as pronounced with both Vanilla RNN and ReLU RNN having quiet similar performance. In the case of `FirstII` dataset, ReLU RNN performed much better with more than 8 out of 10 initializations end up achieving accuracy more than 65% while Vanilla RNN has just 1 out of 10 and 2 out of 10 with

hidden units 2 and 4 respectively. This is an interesting result because the `FirstII` involves remembering the first non-stop word and hence the model naturally has to deal with the long-term dependency. So, ReLU RNN perform better than Vanilla RNN on the `FirstII` dataset.



(a) Vanilla RNN (`CountII`)

(b) ReLU RNN (`CountII`)

(c) Vanilla RNN (`FirstII`)

(d) ReLU RNN (`FirstII`)

Figure 4.16: The row in each subfigure represents the Validation Accuracies (rounded off to the nearest integer) sorted in the increasing order obtained with a given number of hidden units on 10 different initializations. Figure (a), (c) represent the Vanilla RNN while (b), (d) represent the ReLU RNN where the RNN has rectified linear units and its hidden weights were initialized by identity matrix. Dataset used to train in (a) and (b) is `CountII` while in (c) and (d) is `FirstII`.

4.4.2 Initializing Vanilla RNN with the Trained Weights of GRU

The motivation behind the experiment is to see how Vanilla RNNs perform when initialized by the weights of successfully trained GRUs on the same dataset. The GRU with reset gate set to 1 and update gate set to 0 behaves like a Vanilla RNN. So, the experiment investigates whether the Vanilla RNN having initial recurrent hidden weights and corresponding biases same as a successfully trained GRU trains better and achieves higher validation accuracy or not compared to the randomly initialized Vanilla RNN.

Experiment Setup

The datasets used in the experiment are `CountII` In this experiment, we trained some GRU modes on the `CountII` and `FirstII` dataset to a validation accuracy greater than 94%. The hidden weights, fully connected layer's weights and their biases of the Vanilla RNN are then initialized by their corresponding weights of the trained GRU model and then it is trained on the same datasets and the Validation Accuracies obtained are noted. It is compared against 10 randomly initialized Vanilla RNNs. The hidden dimension of the models used is 4.

Initialization	Validation Accuracy
1	85.2
2	77.5
3	69.1

Table 4.1: Validation accuracies of Vanilla RNNs after training on the `CountII` dataset initialized with weights of three different well trained GRU models (validation accuracy more than 94%).

Metric	Value
Maximum Validation Accuracy	91.0
Mean Validation Accuracy	76.87
Minimum Validation Accuracy	55.1

Table 4.2: Maximum, mean and minimum validation accuracy of 10 different Vanilla RNNs after training on the `CountII` dataset with their weights initialized randomly.

Initialization	Validation Accuracy
1	70.4
2	66.3
3	65.6

Table 4.3: Validation accuracies of Vanilla RNNs after training on the `FirstII` dataset initialized with weights of three different well trained GRU models (validation accuracy more than 94%).

Metric	Value
Maximum Validation Accuracy	71.2
Mean Validation Accuracy	53.4
Minimum Validation Accuracy	48.4

Table 4.4: Maximum, mean and minimum validation accuracy of 10 different Vanilla RNNs after training on the `FirstII` dataset with their weights initialized randomly.

Results and Analysis

The Tables 4.1 and 4.2 contains the results of this experiment for the `CountII` dataset and Tables 4.3 and 4.4 contains the results of this experiment for the `FirstII` dataset. In both the cases, the average and worst performance (validation accuracy) of Vanilla RNN is better when it is initialized by a successfully trained GRU parameters than when it is just initialized randomly.

4.4.3 Initializing GRU with the trained Weights of RNN

One explanation behind the failure of Vanilla RNN to learn a function and achieve high test accuracy despite sufficient number of parameters is that it is stuck in the local minima. The motivation behind the experiment is to study whether introducing the gates can force the already unsuccessful network to learn from there on and achieve high accuracy.

Experiment Setup

In this experiment, we picked the unsuccessful Vanilla RNNs models (less than 60% accuracy) that were trained on the `CountII` and `FirstII` dataset. Those final recurrent hidden weights, fully connected layer's weights and their biases were used to initialize the corresponding weights in the GRUs. The weights associated with the update gate and reset gate and their corresponding biases of the GRUs were randomly initialized. The model was then trained on the same dataset. This modified GRU (can be thought of as RNN with gates) against Vanilla RNN. The models used in the experiments had

hidden dimension equal to 4.

Initialization	Accuracy (Vanilla RNN)	Accuracy (GRU)
1	55.1	98.5
2	80.1	97.90
3	75.1	94.1

Table 4.5: Validation accuracy of Vanilla RNNs initialized randomly and trained on `CountII` dataset (Column 2) and validation accuracy of GRUs initialized with trained weights of RNN mentioned in the column 2 of same row and trained on the same dataset.

Initialization	Validation Accuracy (Vanilla RNN)	Validation Accuracy (GRU)
1	48.9	91.7
2	49.1	49.2
3	48.4	49.5

Table 4.6: Validation accuracy of Vanilla RNNs initialized randomly and trained on `FirstII` dataset (Column 2) and validation accuracy of Modified GRUs implies GRUs initialized with trained weights of Vanilla RNN mentioned in the column 2 of same row and trained on the same dataset.

Results and Analysis

In the Table 4.5, the validation accuracy of more than 94% is achieved in all the three different GRU models. The maximum validation accuracy of Vanilla RNN from 3 different initialization was 81.1%. This clearly shows how gating has facilitated training process and improved the accuracy starting with the recurrent hidden weights where Vanilla RNN failed to improve any further.

In the Table 4.6, the validation accuracy of more than 90% is achieved in one of the three different GRU models while no noticeable improvement in the other two cases. So, the improvement with gating is much higher on the `CountII` dataset while comparatively lesser on `FirstII` dataset.

The conclusion from Section 4.2.1, that gating facilitates the training and hence GRUs are easy to train still holds in this experiment.

4.5 Role of Gating in RNNs

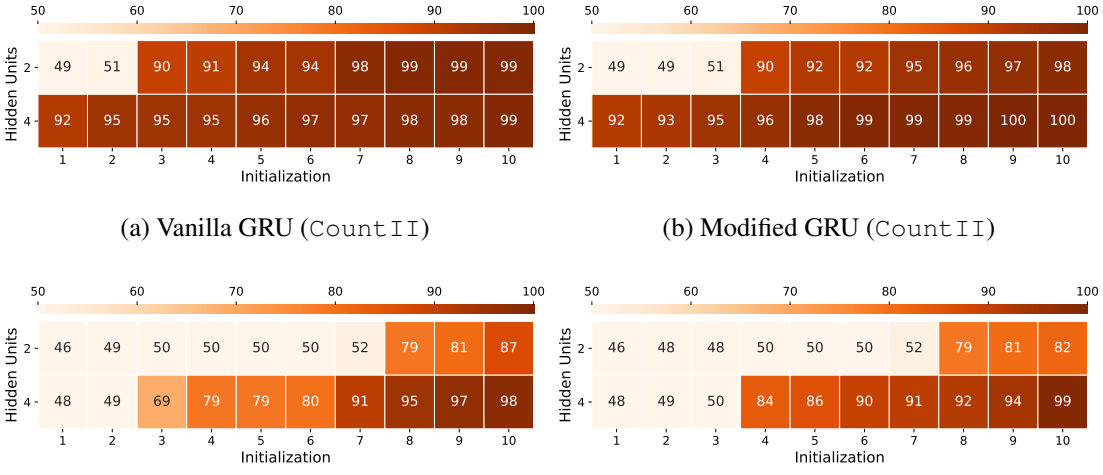
The recurrent neural networks with gating mechanisms like LSTM and GRU are known to have better performance than Vanilla RNN. This section deals with investigating the role of gating in facilitating the training of RNNs.

4.5.1 Training GRU with Frozen Recurrent Hidden Weights

In this experiment, the ordinary or Vanilla GRU is compared with a slightly modified version of the GRU. In the modified GRU, the recurrent hidden weights are randomly initialized and then frozen/fixed and the weights associated with gates, fully connected layer and their biases are trained.

Experiment Setup

The Vanilla GRU and Modified GRU having 2 and 4 hidden units are trained on the `CountII` and `FirstII` dataset. The recurrent hidden weights and corresponding biases (W_{hn} and b_{hn} as mentioned in equation 1.4) are frozen and rest all the weights are trainable in the Modified GRU.



(a) Vanilla GRU (`CountII`)

(b) Modified GRU (`CountII`)

(c) Vanilla GRU (`FirstII`)

(d) Modified GRU (`FirstII`)

Figure 4.17: The row in each subfigure represents the Validation Accuracies (rounded off to the nearest integer) sorted in the increasing order obtained with a given number of hidden units on 10 different initializations. Figure (a), (c) represent the ordinary GRU while (b), (d) represent the Modified GRU where the recurrent hidden weights are frozen. Dataset used to train in (a) and (b) is `CountII` while in (c) and (d) is `FirstII`.

Results and Analysis

The Figure 4.17 represents the results of this experiment. In 7 out of 10 initializations, the Modified GRU while 8 out of 10 initializations Vanilla GRU, both with 2 hidden units got accuracy more than 90% on `CountII` dataset. While in all the 10 initializations, both the Modified GRU and Vanilla GRU with 4 hidden units got the accuracy more than 90% on the `CountII` dataset. On `FirstII` dataset, no model achieved accuracy greater than 90 with 2 hidden units but the Modified GRU with 4 out of 10 initializations and Vanilla GRU with 8 out of 10 initializations both with 4 hidden units got accuracy more than 90%.

This leads to the conclusion that having gates with frozen hidden weights is much better having trainable hidden weights but no gates (Vanilla RNN). Also, quite surprisingly, Vanilla GRU and this Modified GRU have comparable performance.

4.5.2 Variations with Gating in GRUs

The GRUs have two gates reset gate and update gate. This experiment aims at doing the comparative analysis of four variant of the GRUs involving the Reset and Update gate with each either being fixed or trainable.

When both the reset and update gates are fixed to 1 and 0, it acts like a Vanilla RNN while if weights associated with both are trainable then it is the ordinary GRU. Two more variants have one of the gate as trainable and the other as fixed. We define Update Gate GRU as the GRU whose update gate weights are trainable while reset get is set to 1 and Reset Gate GRU as the GRU whose reset gate weights are trainable while update get is set to 0.

The mathematical equations of Vanilla RNN and GRU are mentioned in Section 1.1 and 1.2 respectively. The symbols and notations used in the following equations are similar to what is defined in Section 1.2.

The equations for Update Gate GRU:

$$r_t = 1, \tag{4.1}$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}), \tag{4.2}$$

$$\tilde{h_t} = tanh(W_{in}x_t + b_{in} + W_{hn}h_{(t-1)} + b_{hn}), \tag{4.3}$$

$$h_t = (1 - z_t) \odot \tilde{h_t} + z_t \odot h_{(t-1)} \tag{4.4}$$

The equations for Reset Gate GRU:

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}), \tag{4.5}$$

$$z_t = 0, \tag{4.6}$$

$$\tilde{h_t} = tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{(t-1)} + b_{hn})), \tag{4.7}$$

$$h_t = \tilde{h_t} \tag{4.8}$$

Experiment Setup

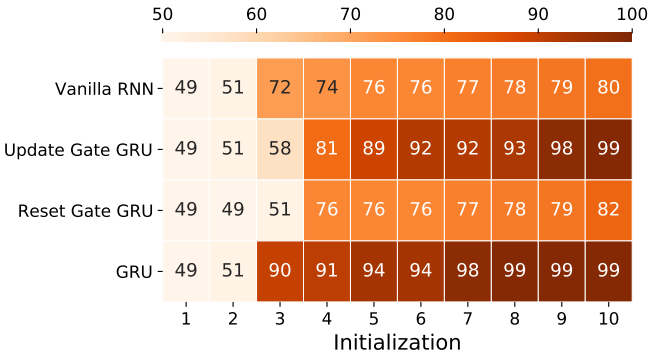
All the Vanilla RNN, Update Gate GRU, Reset Gate GRU and GRU as mentioned above are trained with both 2 hidden units and 4 hidden units on the `CountII` and `FirstII` dataset.

Results and Analysis

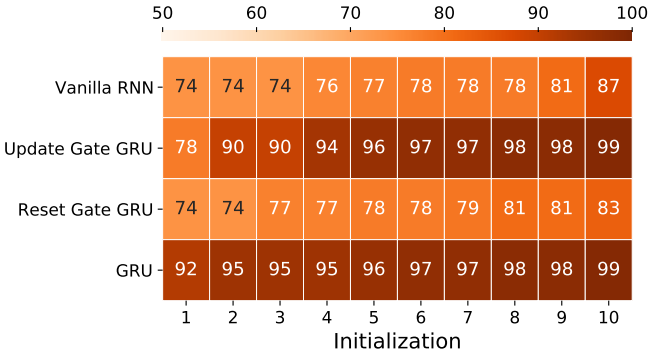
On the `CountII` dataset, GRU was the best performer followed by Update Gate GRU, Reset Gate GRU and Vanilla RNN being the worst (Fig. 4.18 (a), (b)). On the `FirstII` dataset, surprisingly, Update Gate GRU was the best performer followed by GRU, Reset Gate GRU and Vanilla RNN being the worst (Fig. 4.18 (c), (d)).

Clearly, having some sort of gating mechanism is better than having none since Vanilla RNN performed worst among the 4 variants on both the datasets. Update Gate GRU is better than Reset Gate GRU hence it can be inferred that update gate is more crucial component of the GRU and when it is fixed/frozen the model performs significantly worse (Reset Gate GRU).

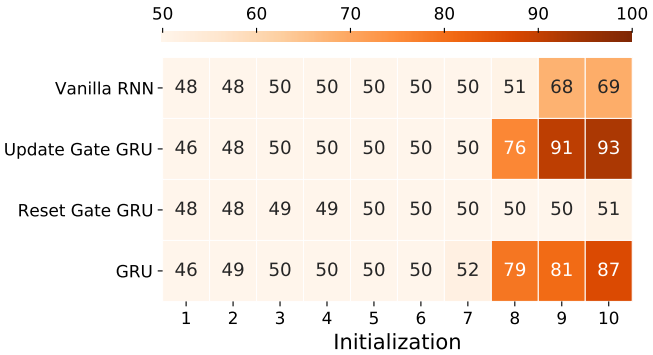
The Update Gate GRU is better than GRU on `FirstII` dataset (Fig. 4.18 (c), (d)).



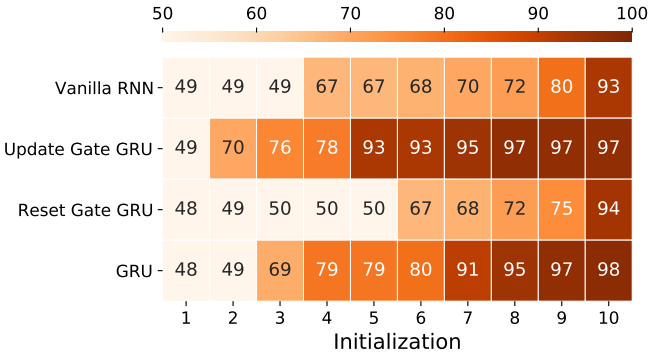
(a) `CountII` (2 Hidden Units)



(b) `CountII` (4 Hidden Units)



(c) `FirstII` (2 Hidden Units)



(d) `FirstII` (4 Hidden Units)

Figure 4.18: The row in each subfigure represents the Validation Accuracies (rounded off to the nearest integer) sorted in the increasing order obtained by the given variant on 10 different initializations. Update Gate GRU and Reset Gate GRU are defined in Section 4.5.2.

4.6 Optimization Landscape

4.6.1 Frequency of the solutions on the Loss Surface

This experiment aims to improve the understanding of the capacity of the Vanilla RNN and GRU. We do a million different parameter settings on a particular architecture and without any training, we take the inference on the validation set and record the accuracies. We plot the frequency distribution of the accuracies in a distribution plot as well as box plot.

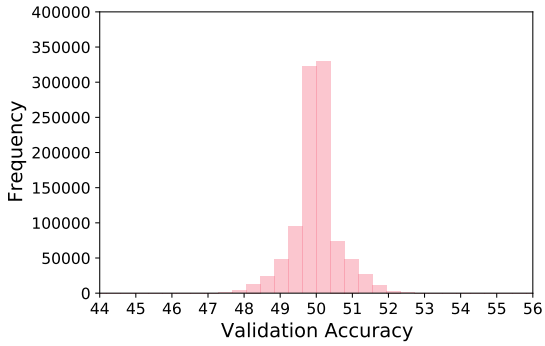
Experiment Setup

We used only `FirstI` dataset for this experiment as the `XorI` dataset turned out to be too difficult for the smaller architectures (Fig. 4.5). The `CountI` dataset is much easier and hence it was less interesting for this experiment. We considered the models with all the four combinations of input unit and hidden unit as 2 and 4.

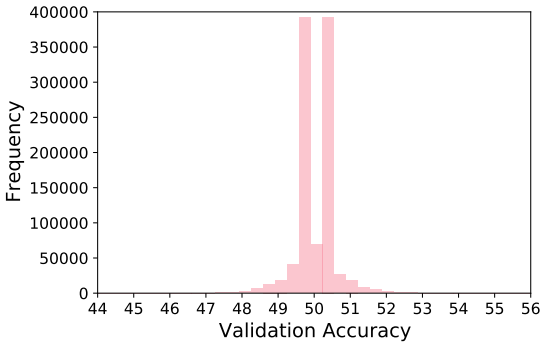
Results and Analysis

The Figure 4.19 and 4.20 contains the plots for this experiment. The plots of Vanilla RNN and GRU are quite similar and suggests that the frequency of a point giving particular accuracy on the loss surface is same for Vanilla RNN and GRU. This suggests that there are roughly equal number of solutions in Vanilla RNN or GRU with similar input and hidden dimension. Then, the more number of good parameter settings (Section 4.2.2) but roughly equal number of solutions (minima) (Section 4.6.1) suggests that the reason for GRU performing much better than Vanilla RNN is the high trainability which can just take the model from more number of positions on the error surface to the good positions that achieve higher accuracy than the Vanilla RNN. This was the reason why GRU was succeeding in more number of parameter settings than Vanilla RNN.

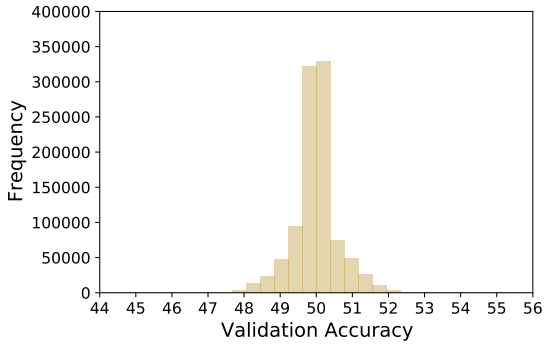
In the Figure 4.20, the box plot, we do not observe any differentiating pattern regarding the spread of the curve in Vanilla RNN or GRU. A slightly vague observation is that the spread of the outliers is wider in the case of GRU than in that of Vanilla RNN but it does not hold clearly when input units and hidden units are 2 and 4 respectively.



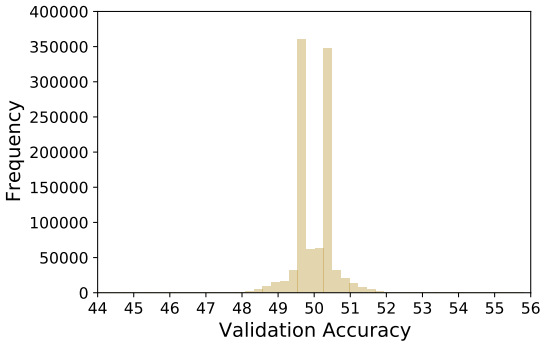
(a) Vanilla RNN (2 x 2)



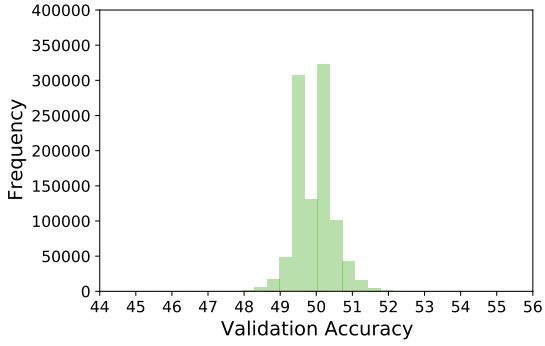
(b) GRU (2 x 2)



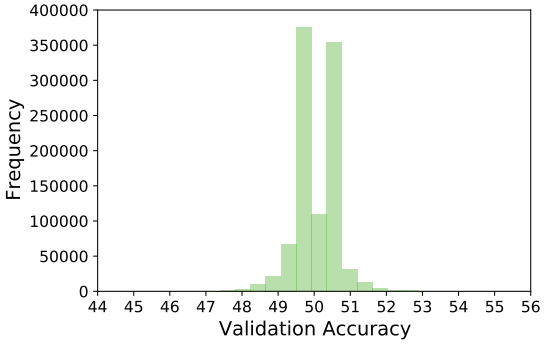
(c) Vanilla RNN (2 x 4)



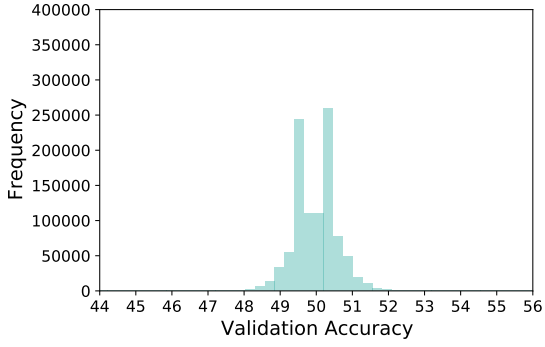
(d) GRU (2 x 4)



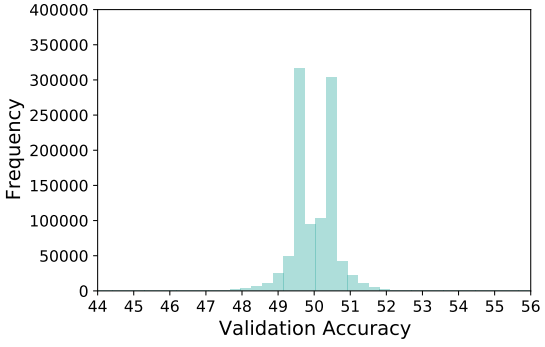
(e) Vanilla RNN (4 x 2)



(f) GRU (4 x 2)



(g) Vanilla RNN (4 x 4)



(h) GRU (4 x 4)

Figure 4.19: Each subfigure is a frequency distribution plot in which each point represents the number of models with different parameter settings that achieved a particular validation accuracy out of the million different parameter settings on the `FirstI` dataset. The subfigure name follows the following naming convention, ModelType (Input Dimension x Hidden Dimension).

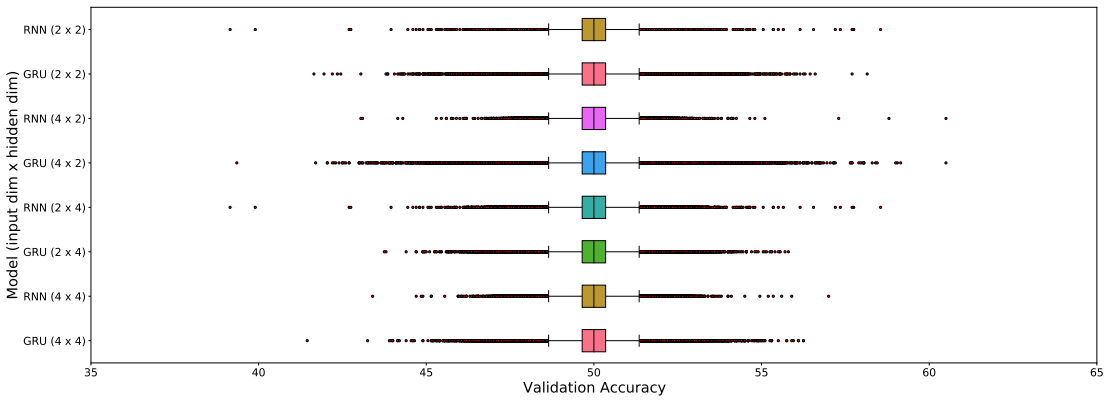


Figure 4.20: Each box plot in the figure represent the distribution of million different Initializations based on the different Validation Accuracies that they achieved for a given model on the `FirstI` dataset. Each boxplot label on the y-axis name follows the following naming convention, ModelType (Input Dimension x Hidden Dimension).

4.6.2 Local Parameter Update relative to the Global Parameter Update

This experiment evaluates the smoothness and stability of the trajectory of Vanilla RNN and GRU. The metric and related terminologies are defined in the following text.

We call $\overrightarrow{W}$ as the parameter vector which is a vector containing all the parameters of the model, $(\overrightarrow{W_N} - \overrightarrow{W_0})$ as the global parameter update vector and $(\overrightarrow{W_t} - \overrightarrow{W_{t-1}})$ as the local parameter update vector (t in subscript is for the next epoch).

Then, let us define the term **Cosine** as the dot product of the local parameter update vector to the global parameter update vector divided by the norm of both the vectors.

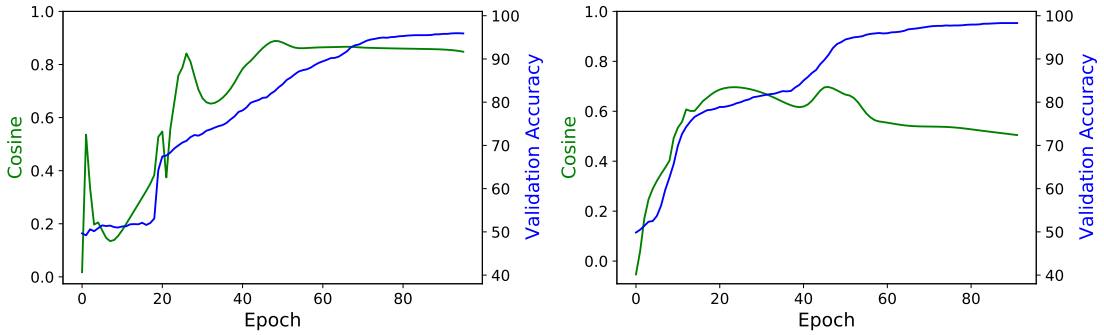
Mathematically,

$$Cosine(t) = \frac{(\overrightarrow{W_N} - \overrightarrow{W_0})^T \cdot (\overrightarrow{W_t} - \overrightarrow{W_{t-1}})}{\left|\left|(\overrightarrow{W_N} - \overrightarrow{W_0})\right|\right|_2 \left|\left|(\overrightarrow{W_t} - \overrightarrow{W_{t-1}})\right|\right|_2}, \forall t = 1\, to\, N, \tag{4.9}$$

where N is total number of epochs.

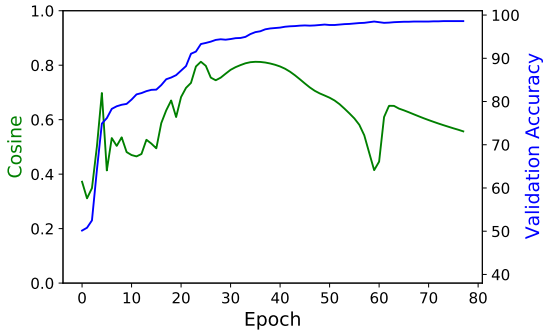
Experiment Setup

The Vanilla RNN and GRU each with 2 input, 4 hidden units and 4 input, 8 hidden units are considered. The models achieved at least 96% validation accuracy. The `FirstI` dataset was used.

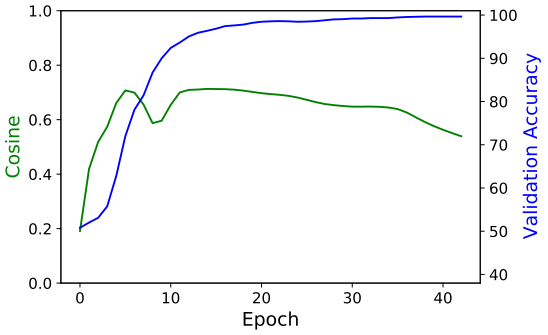


(a) Vanilla RNN (2 x 4)

(b) GRU (2 x 4)



(c) Vanilla RNN (4 x 8)



(d) GRU (4 x 8)

Figure 4.21: Each point on the subfigure shows the value of the Cosine at a particular epoch. The mathematical formula of how the Cosine is calculated is in the Section 4.6.2. The subfigure name follows the following naming convention, ModelType (Input Dimension x Hidden Dimension).

Results and Analysis

The results of this experiment are in the Figure 4.21. The curve makes sense when the final parameters are well trained that is they achieve high accuracy otherwise in the case of unsuccessful models the final parameter do not represent the desired destination. The models considered in the experiment had achieved very high accuracy.

The results show two major things. `FirstI` is that the curve in the case of GRU is much more smoother than what it is in the case of Vanilla RNNs.
The second is that in the GRU, initially the curve is almost always very steeply rising. The consistent high values indicates that GRU local updates are in line with the global updates to a great extent while in Vanilla RNN the local updates deviate much more often. This trend was consistently observed is more cases than the ones whose plots are attached.

4.6.3 Perturbation

This experiment aims to explore the loss surface of both Vanilla RNN and GRU. The loss surface can be imagined as the space with basins with some being deeper and the others being shallower. The optimization algorithm intends to find the global minima of this surface but it can, unfortunately, get stuck at the local minima.

We wish to have an understanding of the structure of basins on the loss surface and whether after the perturbation, model reaches to the minima of the same or a different basin and with what perturbation factor. We perturb the parameters by a perturbation factor and then train the model for some epoch. The amount of perturbation we introduced was relative to the standard deviation of that weight matrix to avoid extremly low or high perturbations. The details of the parameters were perturbed are elaborated in the Experiment Setup.

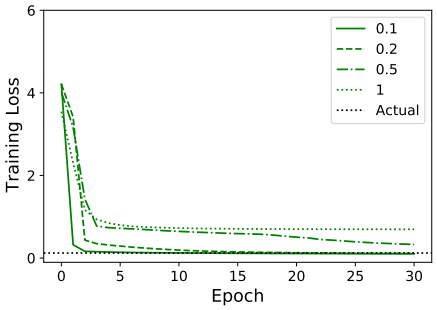
Experiment Setup

We take each matrix/vector of all the weight matrices and the bias vectors (mentioned in the equations in S) one by one and flatten it. The noise vector is created with a normal distribution of mean 0 and standard deviation equals standard deviation of the flatten vector. Then, the perturbation factor times the noise vector is added to the original weight vector to form the vector with perturbed weights/parameters.
Mathematically,

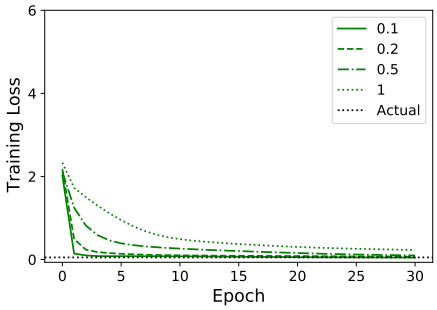
$$\overrightarrow{W_{pert}} = \overrightarrow{W^*} + \overrightarrow{noise}, \tag{4.10}$$

$$\overrightarrow{noise} = k\,\mathcal{N}(0,\, \sigma_{\overrightarrow{W}}), \tag{4.11}$$

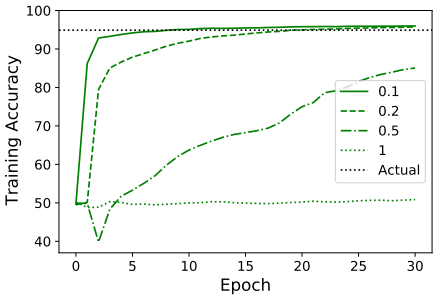
where $\overrightarrow{W^*}$ is flattened matrix/vector of actual parameters, $\overrightarrow{W_{pert}}$ is flattened matrix/vector of the perturbed parameters, σ represent the standard deviation, k is the perturbation factor, higher the k more is the perturbation, $\odot$ represents element-wise product.
The models of Vanilla RNN and GRU had 2 input and 4 hidden units and they were trained on the `FirstI` dataset to get at least a validation accuracy of 98%.
The model was trained to 30 epochs after its parameters had been perturbed. The perturbation factor used in the experiment are 0.1, 0.2, 0.5 and 1.



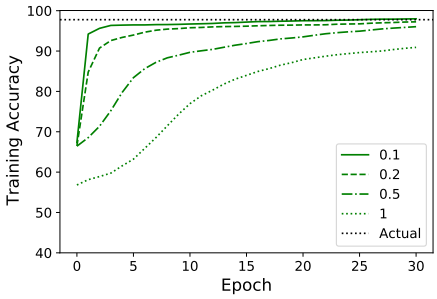
(a) Vanilla RNN



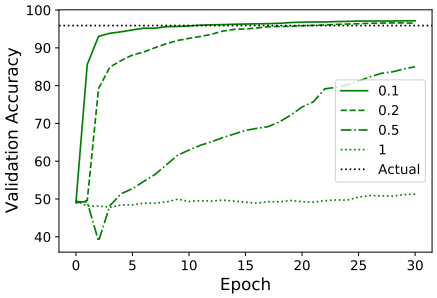
(b) GRU



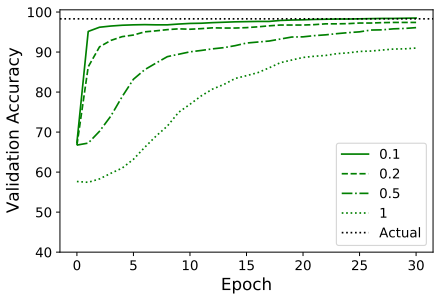
(c) Vanilla RNN



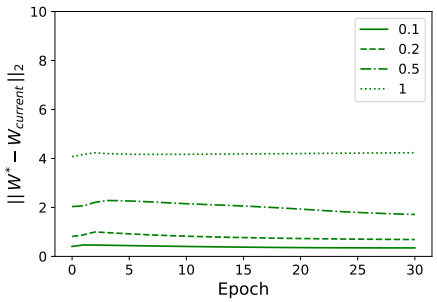
(d) GRU



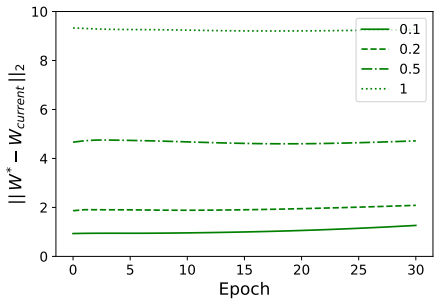
(e) Vanilla RNN



(f) GRU



(g) Vanilla RNN



(h) GRU

Figure 4.22: The parameters of the model are perturbed by a perturbation factor (say, k) and the model is then trained for 30 epochs. The legend represents different perturbation factors and the dotted black line represents the actual value of the metric (that is without perturbation). (g), (h) The euclidean norm of the difference of W and W_{epoch} is plotted against the number of epochs where W_{epoch} is the parameter vector of the model just after a particular epoch and W is the actual parameter vector of the model. The `FirstI` dataset was used.

Results and Analysis

The plots of this experiments are in Figure 4.22. The results clearly declare GRU as the better performer. Even for the perturbation factor of 1 (which was the highest), the GRU was able to achieve an accuracy close to 90% in merely 30 epochs. This reasserts that training of GRU is more effective than that of the Vanilla RNN and hence GRUs are more robust to the perturbation.

One surprising result is that we have expected the perturbed model weights to return close to the actual weights before perturbation but the difference in norm never gets close to zero in fact it shows a very little change compared to what we had expected. Another observation is that for the same amount of perturbation the initial loss in the case of Vanilla RNN is almost twice than the initial loss in the case of GRU. But this is specific to the initialization used for the Figure 4.22 and was not consistent for all the initializations.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this paper, we performed the binary classification of the sequences using Vanilla RNN and GRU on the three synthetic datasets. The results reassert the popular belief that GRU has superior performance to that of the Vanilla RNN. The results indicate that Vanilla RNN is highly susceptible to the initialization (Section 4.2.1 & Section 4.2.2), less robust to the perturbation (Section 4.6.3) and far more difficult to train than the GRU (Section 4.2.1) despite both having the similar capacity (Section 4.2.2). Also, GRU is more robust to the noise (observed in Section 4.3.2) and has smoother weight updates than the Vanilla RNN (Section 4.6.2).

We also conclude that although the number of solution states with same accuracy for a given number of input and hidden units are almost similar in both the RNN and GRU (Section 4.6.1), the number of good initializations in GRU is higher than that in the Vanilla RNN. This clearly alludes to the fact that both are equally capable to model the given dataset of the sequences, the training being more effective in the case of GRU takes the model to the solution state from much more initial states than what would have been in the case of Vanilla RNN.

The embeddings play an important role in the training process of the RNNs and the linear separability of the word vectors of the different sentiments can enable model to achieve high accuracy (Section 4.1.1).

The change in mean of the norms of the vectors of a vector field is negatively correlated to the change in the accuracy of the model as a result of the training (difference of the initial and final validation accuracy) (Section 4.3.3).

Initialization trick for the Vanilla RNN as mentioned by Le *et al.* (2015) works on our synthetic dataset, specially for the labelling function `First` (Section 4.4.1).

The gating plays a crucial role in the training process (Section 4.5.1 & Section 4.5.2). Even if the recurrent hidden weights of a GRU are frozen to randomly initialized value, it can succeed and perform comparably to ordinary GRU and much better than the Vanilla RNN (Section 4.5.1). The update gate of GRU seems more critical than the reset gate of GRU (Section 4.5.2).

5.2 Future Work

The experiments were conducted on the synthetic dataset. We wish to verify the conclusions made on them on some larger real world datasets. We also aim to incorporate advanced concepts in RNNs such as attention, as well as more sophisticated models like Transformer which has shown tremendous success by setting up the state-of-the-art in tasks dealing with long sequences (Vaswani *et al.*, 2017). Also, after various interesting observations we have made in this thesis, we wish to extend some of the experiments to them.

With the deeper understanding obtained through all these experiments, we hope to step towards our vision of devising better initialization strategies, better learning algorithm and provide better ways to tune the hyperparameters in an informed manner that can save a lot of precious time and resources. This understanding can also help in developing ways to diagnose the recurrent neural network and debug it to improve its performance in more informed manner.

REFERENCES

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

2. **Cho, K.**, **B. van Merrienboer**, **C. Gulcehre**, **D. Bahdanau**, **F. Bougares**, **H. Schwenk**, and **Y. Bengio** (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv e-prints*, arXiv:1406.1078.

3. **Chung, J.**, **C. Gulcehre**, **K. Cho**, and **Y. Bengio** (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv e-prints*, arXiv:1412.3555.

4. **Collins, J.**, **J. Sohl-Dickstein**, and **D. Sussillo** (2016). Capacity and Trainability in Recurrent Neural Networks. *arXiv e-prints*, arXiv:1611.09913.

5. **Doya, K.** (1993). Universality of fully-connected recurrent neural networks. Technical report, IEEE Transactions on Neural.

6. **Hochreiter, S.** and **J. Schmidhuber** (1997). Long short-term memory. *Neural computation*, **9**(8), 1735–1780.

7. **Le, Q. V.**, **N. Jaitly**, and **G. E. Hinton** (2015). A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. *arXiv e-prints*, arXiv:1504.00941.

8. **Martens, J.** and **I. Sutskever**, Learning recurrent neural networks with hessian-free optimization. *In Proceedings of the 28th international conference on machine learning (ICML-11)*. Citeseer, 2011.

9. **Pascanu, R.**, **T. Mikolov**, and **Y. Bengio** (2012). On the difficulty of training Recurrent Neural Networks. *arXiv e-prints*, arXiv:1211.5063.

10. **Paszke, A.**, **S. Gross**, **F. Massa**, **A. Lerer**, **J. Bradbury**, **G. Chanan**, **T. Killeen**, **Z. Lin**, **N. Gimelshein**, **L. Antiga**, **A. Desmaison**, **A. Köpf**, **E. Yang**, **Z. DeVito**, **M. Raison**, **A. Tejani**, **S. Chilamkurthy**, **B. Steiner**, **L. Fang**, **J. Bai**, and **S. Chintala** (2019). Pytorch: An imperative style, high-performance deep learning library.

11. **Salehinejad, H.**, **J. Baarbe**, **S. Sankar**, **J. Barfett**, **E. Colak**, and **S. Valaee** (2018). Recent advances in recurrent neural networks. *ArXiv*, **abs/1801.01078**.

12. **Sutskever, I.**, **J. Martens**, **G. Dahl**, and **G. Hinton**, On the importance of initialization and momentum in deep learning. *In Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13. JMLR.org, 2013.

13. **Vaswani, A.**, **N. Shazeer**, **N. Parmar**, **J. Uszkoreit**, **L. Jones**, **A. N. Gomez**, **L. Kaiser**, and **I. Polosukhin** (2017). Attention Is All You Need. *arXiv e-prints*, arXiv:1706.03762.