Empirical Evaluation of Trainability, Generalizability and Capacity of Recurrent Neural Networks

Ojas Mehta (CS18M038), IIT Madras

Abstract

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 paper, we consider the binary classification of the sequences on the three synthetic datasets by using Vanilla RNN and Gated Recurrent Unit (GRU). We perform several experiments in order to deeply understand the training process of both the architecture, the role of gating mechanism in it, evaluate the trainability, generalizability and capacity and explore the optimization landscape of both the models.

Index Terms

Recurrent Neural Network, Gated Recurrent Unit, Sequence Classification.

I. 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, 2018).

It is widely known that though very powerful, RNNs are difficult to train. The key problems making the training of 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 better initialization strategies. All these will save the large amount of computing resources used in random 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 many 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 task can be represented using models of given input and hidden dimension. For example we would claim the RNN to be of the same capacity as that of GRU if there exist an parameter settings for which RNN can achieve as good accuracy as achieved by the GRU having same input and hidden dimension for any parameter settings.

Although RNNs have shown successes in generation as well as classification tasks, we restrict our attention to the binary classification of sequences using RNNs. We use three synthetic datasets (see section III(A) 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 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 easily unfolds but it is also shown to outperform LSTM (Chung et al., 2014).

A. Vanilla Recurrent Neural Network (Vanilla RNN)

The 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,

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

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

$$o_t = W_{oh}h_t + b_{oh}, \tag{2}$$

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 RNN.

B. 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

$$h_t = (1 - z_t) \odot \tilde{h_t} + z_t \odot h_{(t-1)}, \tag{3}$$

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,

$$\tilde{h_t} = tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{(t-1)} + b_{hn})), \tag{4}$$

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

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \tag{5}$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \tag{6}$$

The output is calculated in the same way as that in the RNN (equation (2)).

II. 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 (2017) 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.

III. Experiments

A. The Dataset

For the ease of understanding, the sequences can be viewed as the sentences with stop words and non-stop words (positive and negative sentiments).
We used three datasets. We have named them as the `Count`, `First` and `Xor` dataset. The only difference between these three dataset is of the labelling function.

In the `Count` dataset, 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` dataset, 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` dataset, 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.

Each dataset was split into Train, Validation and Test set. Train set had 8000 instances, Validation set had 2000 instances and Test set had 3000 instances. 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 `Count` Dataset is the easiest, while the `First` dataset is moderately difficult and the `Xor` dataset is the hardest among these three datasets.

B. 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 $1e-6$. The Adam optimization algorithm was used. The non-linearities used are as mentioned in the equations in section I.

The learning rate was also kept 0.001 for the sake of uniformity and is halved every time during training when the validation accuracy gets worse. We kept sequence length also constant as 10. After trying a few different 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. The accuracy mentioned anywhere in the paper are rounded off to the nearest integer. Also, when RNN and GRU are mentioned together then RNN symbolises Vanilla RNN unless otherwise stated.

IV. Interpretation of the Components of RNNs

In this section we will interchangeably use the word sentence and sequence. The classification of the sentences is analogous to the sentiment analysis. 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. We have used the `Count` dataset in this entire section. The reason being that it is the easiest dataset among three on which the model could achieve the accuracy of 87% and 100% in the case of Vanilla RNN and GRU respectively even with merely 2 input and 2 hidden units.

A. Differentiating Stop and Non-Stop words

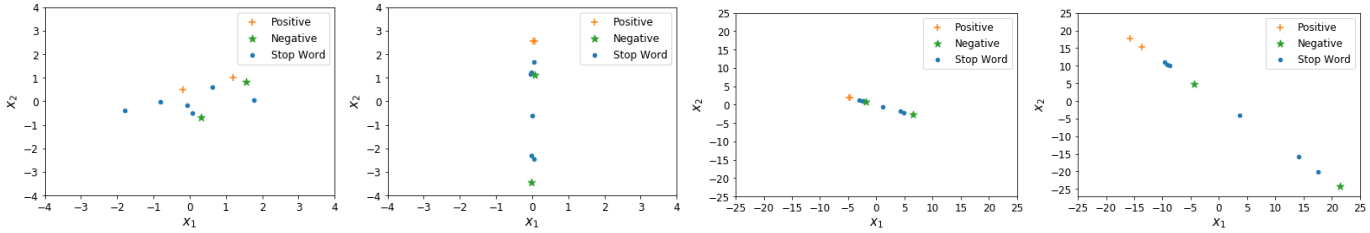
Based on how `Count` labelling function is labelling the sentences, in order to learn the function, the network should somehow learn to ignore the stop words. We expected that the effect of non-stop words on the hidden state should be significant and stop words would have the negligible effect on the hidden state.
Ideally, stop words should belong to the null space of the input matrix, W_{ih} Mathematically, $W_{ih}\mathbf{x_{stop}} = \tilde{\mathbf{0}}$.

1) 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 `Count` dataset is used.

2) Results and Analysis: The Fig.1 represents the plots for the experiment. (a) represents the words vectors in two-dimensions. We have expected the stop words to belong to the null space of the input matrix in the case of Vanilla RNN. But in Fig. 1(b), we did not observe this. Instead the product lies along the one dimension and the magnitude along x_1 is very close zero. Interestingly, the positive and negative words are linearly separable.
In Fig. 1(c), 1(d), the value $W_{oh}.W_{ih}.\mathbf{x}$ lies on a single straight line for both RNN and GRU. Also, the points for the positive and the negative words are linearly separable.

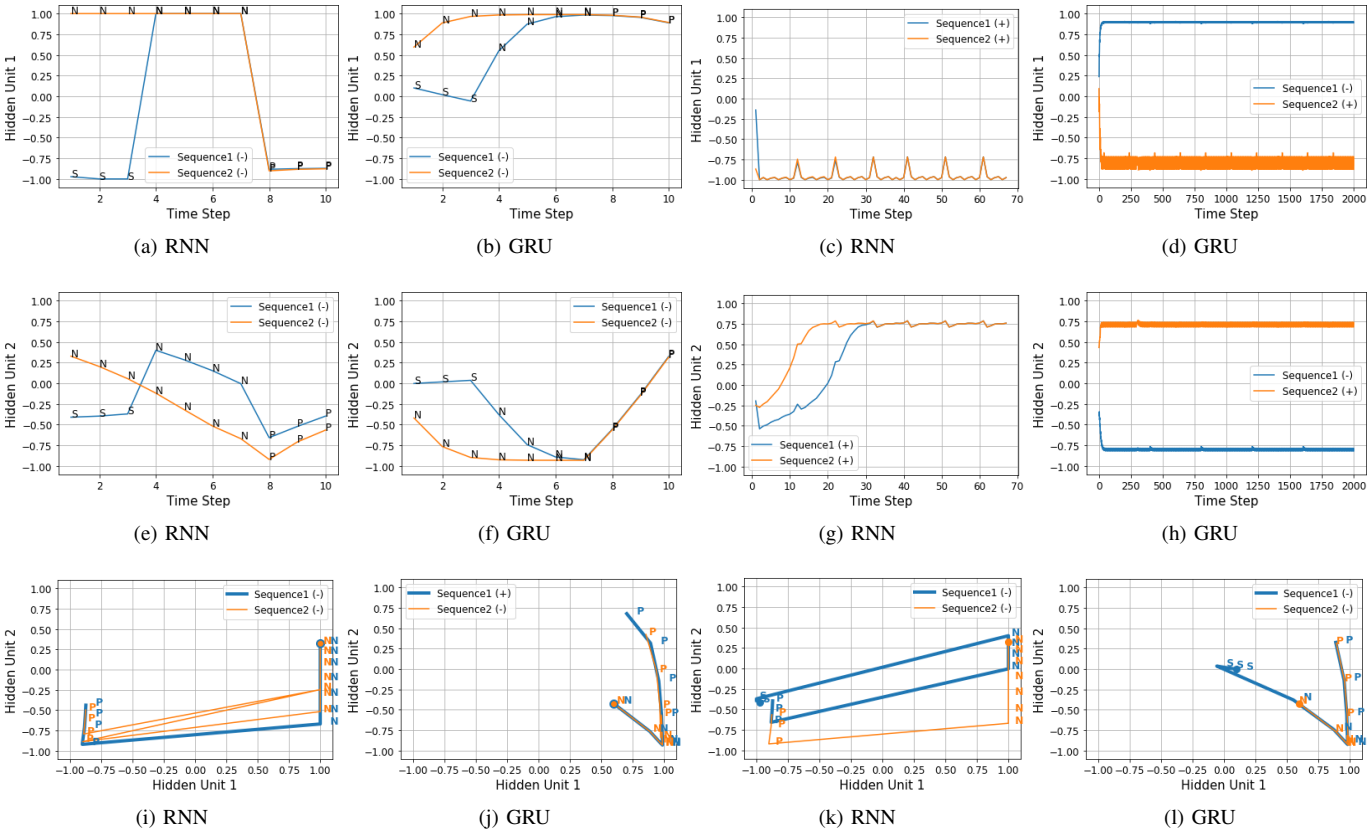
B. Transformation of the hidden state

In this experiment we study how the hidden state (2-dimensional in this case) of a trained model evolves as it sees subsequent stop or non-stop words of the sequence. We also explore the effect of long term as well as short term dependencies and how it affects the inference of the model.



(a) Vocabulary (b) RNN (c) RNN (d) GRU

Fig. 1: Both the input and hidden dimension is 2. (a) The entire vocabulary of 10 Words. (b) The product of Input matrix and Words ($W_{ih}.\mathbf{x}$). (c), (d) The product of Output matrix, Input matrix and words($W_{oh}.W_{ih}.\mathbf{x}$).



(a) RNN (b) GRU (c) RNN (d) GRU

(e) RNN (f) GRU (g) RNN (h) GRU

(i) RNN (j) GRU (k) RNN (l) GRU

Fig. 2: The first 8 plots of the figure represent the effect of positive, negative and stop words on each individual hidden unit separately while the last 4 plots are the state space plots which shows how state evolves as the well trained model sees the words of the sequence. The (+) or (-) in the legend indicates whether the sequence was classified as positive or negative respectively. For (a), (b), (e), (f), (k), (l) the *Sequence1* = [S,S,S,N,N,N,N,P,P,P] & *Sequence2* = [N,N,N,N,N,N,N,P,P,P]. For (c), (e), (g), (h), the *Sequence1* = [N, S, S, S, S, ...] & *Sequence2* = [P, S, S, S, S, ...]. For (i), (j), the *Sequence1* = [N,N,N,N,N,N,P,P,P,P] & *Sequence2* = [N,N,N,N,N,P,N,P,P,P]. Here P represents positive word, N represents negative word and S represents stop word. The solid circular mark at one of the end in (i), (j), (k), (l) represents the starting point or the first word of the sequence.

1) Experiment Setup: The input dimension and the hidden dimension of the Models is 4 and 2 respectively. Both the models achieved the accuracy greater than 98% on the Validation dataset. We do this for both Vanilla RNN and GRU to understand whether the model with gating mechanism trained are more generalizable or not. The `Count` dataset is used.

2) Results and Analysis: Fig. 2 contains the plots of this experiment. In the subfigure (a), (b), (e), (f), (k), (l), the same sequences are used which are mentioned in the image caption. The initial three words of one of the sequences are stop words while non-stop words in the other but all the words from fourth position onwards are the same. In the plot of GRU, the state coincides from the 7 word onwards while in RNN the state never coincides at all. The value of second hidden unit remains

different for both the sequences. Since, the first 3 words were stop words in one of the sequences the network should learn to completely ignore them. It is clear from the plots that GRU is ignoring the stop words better than RNN despite both the networks getting more than 98% accuracy on the validation set.
In (i), (j), the number of positive and negative words is same, the only difference is that one pair of adjacent positive and negative words is swapped. The RNN correctly classify both the sequences while the GRU misclassify one of the sequences despite the fact that models were trained on the sentences of length 10 and these sentences were also of the same length.
One possible reason can be that GRU hits the capacity bottleneck as there are merely 2 hidden units and the fact that RNN has slightly higher capacity than GRU (Collins, 2017).
In (c), (d) 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 RNN could only classify correctly till the 15 length sentences than it started misclassifying and it completely forgets the occurrence of first non-stop words after around 30 time steps as the states overlap. On the other hand GRU for the same sentences could surprisingly classify correctly even till the 2000 time steps. This suggests that GRU is more generalizable than Vanilla RNN.

V. Trainability and Capacity of RNNs

A. Trainability of the Vanilla RNN and GRU

This experiment aims at evaluating the trainability of the RNNs. We do the comparative study the RNN and GRU to have a deeper understanding of how hard Vanilla RNNs are to train and the extent to which the gating facilitates the training of GRUs. Therefore, we train both Vanilla RNN and GRU on the `First` and `Xor` dataset for different number of input and hidden dimension and plot heatmap of the validation accuracies obtained in each case. We do not use `Count` dataset as it is too easy compare the models and get insightful results.

1) Experiment Setup: We considered the models with all the 16 combinations of the input dimension as $\{2, 5, 10, 15\}$ and hidden dimension as $\{2, 4, 8, 16\}$. We used both the `First` and `Xor` dataset for the experiment.

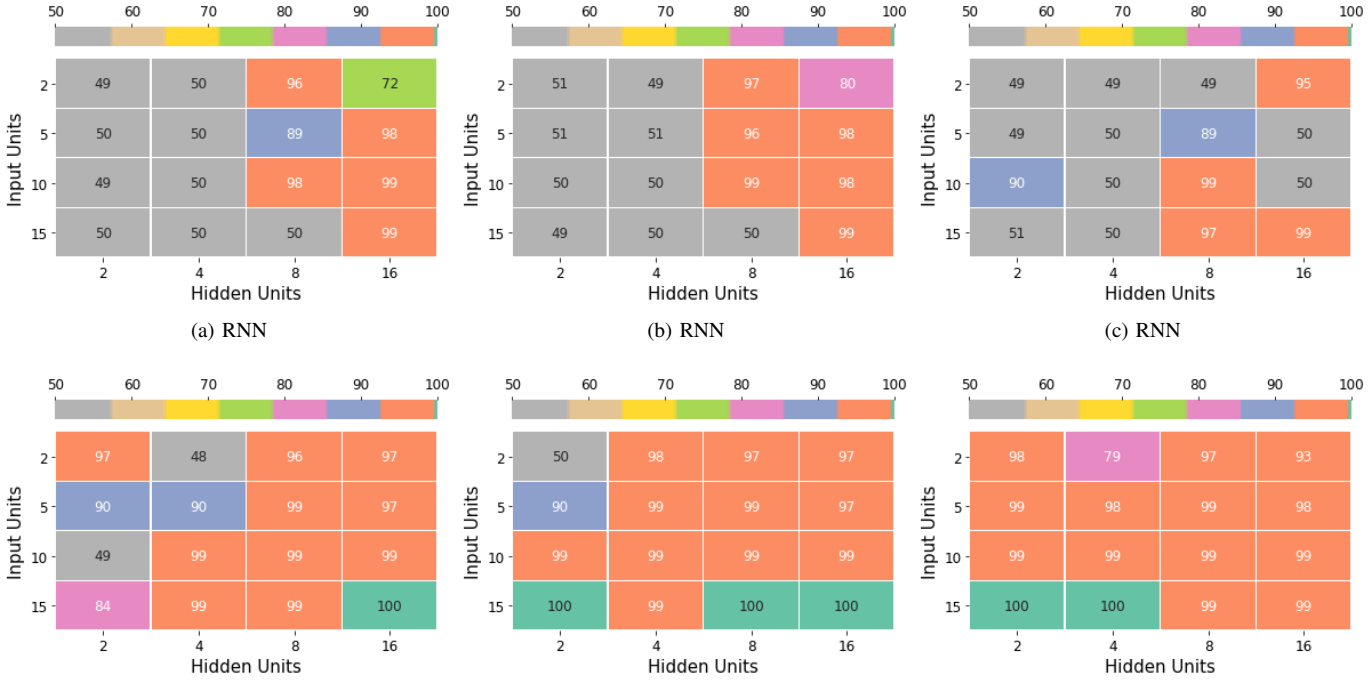
2) Results and Analysis: The Fig. 3 and 4, contains the plots for this experiment. In terms of performance GRU is the clear winner, it succeeds far more than RNN. This fact does not mean that GRU are more powerful than RNN for the same number of input and hidden units. In fact, interestingly, we found them equally capable which we investigate further in the next experiment. But this superior performance of GRU definitely imply that GRUs are easier to train compared to Vanilla RNN. One thing which is clearly implied from these results is that Vanilla RNNs are much more susceptible to the initializatition. For instance, RNN in Fig. 3(c) the RNN achieved 90% accuracy for 10 input units and 2 hidden units but could not achieve more than 51% accuracy in the first two columns even when the model has equal or more input or hidden dimensions for all the three initializations of RNN((a), (b), (c)). Similarly, in the Fig. 4, RNN could achieve decent accuracy in only 1 out of the 48 different architectures. This also suggests that `Xor` dataset pose much more harder problem for the RNNs than the `First` dataset.

B. Capacity of the Vanilla RNN and GRU

This experiment is in continuation of the previous experiment (Section V(A)) and aims to further understand the trainability and also the capacity of RNN and GRU. We pick some of the critical input and hidden dimensions based on the results obtained in previous experiment. We train the models on those architectures for 30 different initializations to check if the ones who were failing on a particular initialization, succeed in one of the many initializations and those who succeed how often those succeed.

1) Experiment Setup: We consider 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 light architectures could be key in investigating whether the poor performance was a result of a bad initialization (training difficulty) or the model is not even capable to represent the labelling function in this number of parameters (Capacity). We use both `First` and `Xor` dataset for the experiment.

2) Results and Analysis: The Fig. 5 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 RNN are equally capable to represent the information in the same number of units as GRU can. In other words, Vanilla RNN and GRU have the similar capacity but GRU are easily trainable but Vanilla RNNs are far more difficult to train. It also affirms our conclusion from the previous experiment that Vanilla RNNs are highly susceptible to the initialization. To put it another way, good initializations in the Vanilla RNNs are much more rare compared to those in the GRU.



(a) RNN (b) RNN (c) RNN

(d) GRU (e) GRU (f) GRU

Fig. 3: The heatmap in each subfigure represents the Validation Accuracies(rounded off to the nearest integer) of the models for the given number of input and hidden dimension on the `First` dataset. (a), (b), (c) represent three different initializations of the RNN. (d), (e), (f) represent three different initializations of the GRU. The `First` dataset is used.

VI. Optimization Landscape

A. *Frequency of the solutions on the Loss Surface*

This experiment aims to improve the understanding of the capacity of the 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.

1) Experiment Setup: We used only `First` dataset as the `Xor` dataset as observed in the previous experiments turned out to be much harder specially for the models with less number of parameters. `Count` dataset as already mentioned is much easier. So, we expect the `First` dataset to be the most interesting for the experiment. We considered the models with all the four combinations of input unit and hidden unit as 2 and 4.

2) Results and Analysis: The Fig. 7 contains the plots for this experiment. The plots of RNN and GRU are quite similar and suggests that the frequency of a point giving particular accuracy on the loss surface is same for 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 (as discussed in section V(B)) but roughly equal number of solutions (minima) suggests that the reason for GRU performing much better than 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 Fig. 8, the box plot, we do not observe any differentiating pattern regarding the spread of the curve or the outliers in RNN or GRU.

B. *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,

$$Cosine(t) = \frac{(\overrightarrow{W_N} - \overrightarrow{W_0})^T \cdot (\overrightarrow{W_t} - \overrightarrow{W_{t-1}})}{\left|\left|(\overrightarrow{W_N} - \overrightarrow{W_0})\right|\right|_2 \left|\left|(\overrightarrow{W_t} - \overrightarrow{W_{t-1}})\right|\right|_2}, \forall t = 1\, to\, N, \tag{7}$$

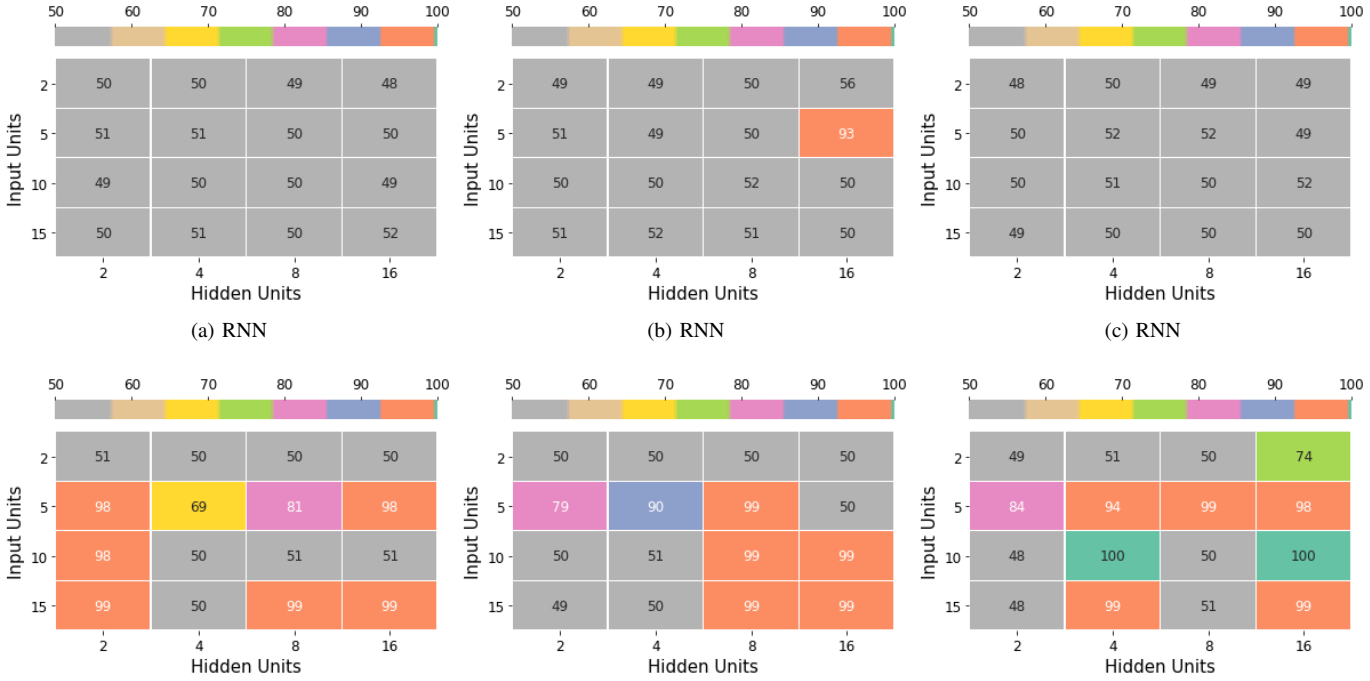


Fig. 4: The heatmap in each subfigure represents the Validation Accuracies (rounded off to the nearest integer) of the models for the given number of input and hidden dimension on the `Xor` dataset. (a), (b), (c) represent three different initializations of the RNN. (d), (e), (f) represent three different initializations of the GRU.

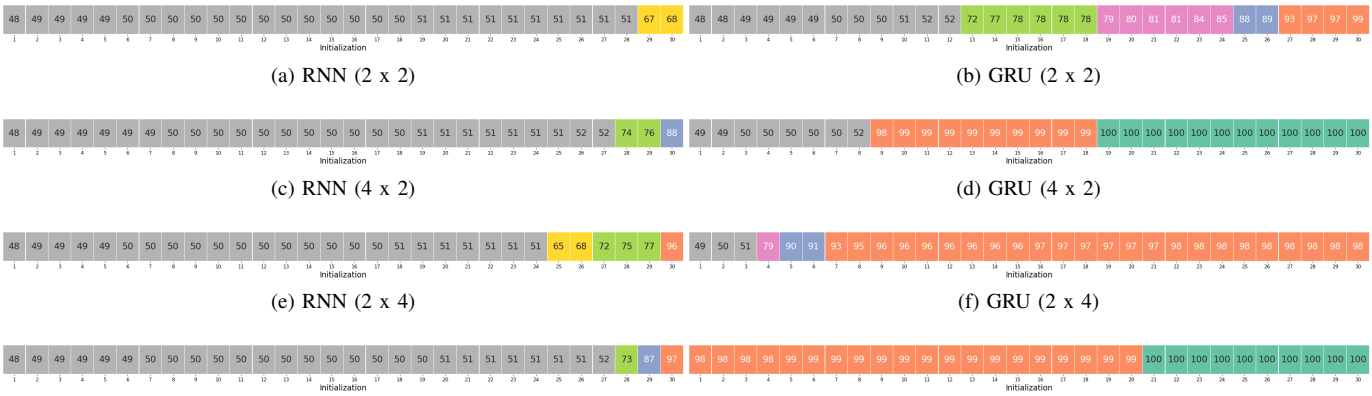
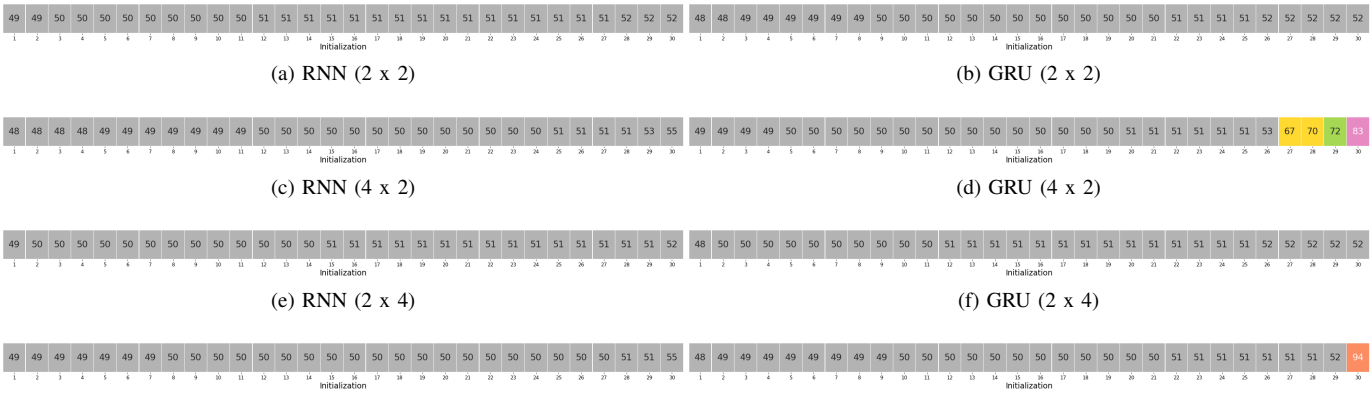


Fig. 5: Each subfigure shows the Validation Accuracies (rounded off to the nearest integer) of the models across 30 different initializations on the `First` dataset. The subfigure name follows the following naming convention, ModelType (Input Dimension x Hidden Dimension).

where N is total number of epochs.

1) Experiment Setup: The 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 `First` dataset is used.

2) Results and Analysis: The results of this experiment are in the Fig. 9. The models considered for the experiments have very high accuracy because the global update direction only makes sense when the final parameters represent a minima with very high training and validation accuracy. The results show two major things. `First` is that the curve in the case of GRU is much more smoother. 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 RNN the local updates deviate much more often. We have plotted for the 2 cases for each RNN and GRU but this trend was consistently observed is many more cases.



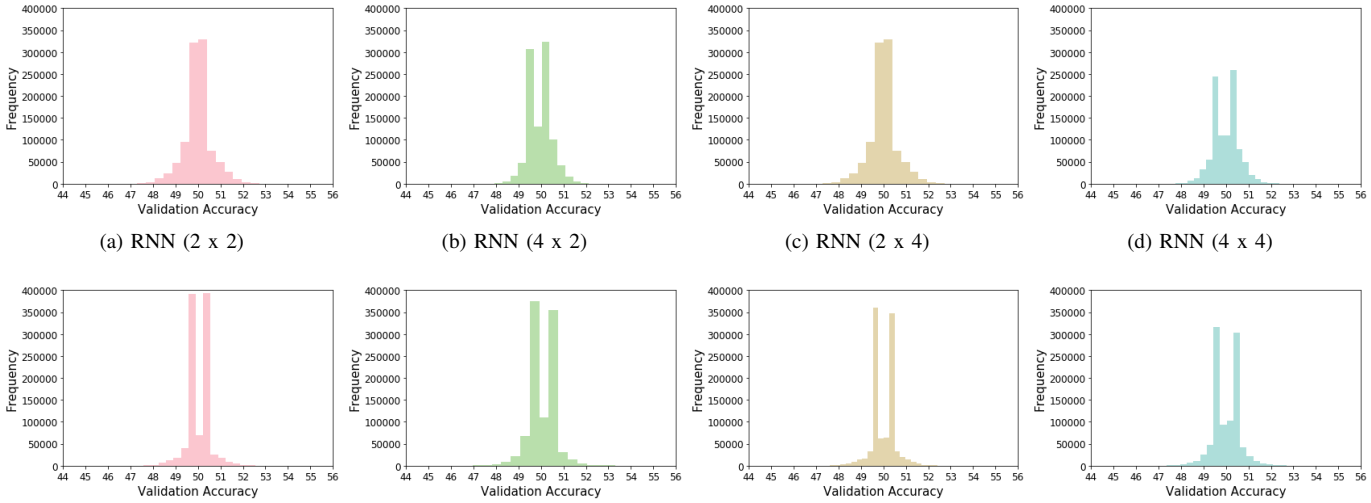
(a) RNN (2 x 2) (b) GRU (2 x 2)

(c) RNN (4 x 2) (d) GRU (4 x 2)

(e) RNN (2 x 4) (f) GRU (2 x 4)

(g) RNN (4 x 4) (h) GRU (4 x 4)

Fig. 6: Each subfigure shows the Validation Accuracies (rounded off to the nearest integer) of the models across 30 different initializations on the `Xor` dataset. The subfigure name follows the following naming convention, ModelType (Input Dimension x Hidden Dimension).



(a) RNN (2 x 2) (b) RNN (4 x 2) (c) RNN (2 x 4) (d) RNN (4 x 4)

(e) GRU (2 x 2) (f) GRU (4 x 2) (g) GRU (2 x 4) (h) GRU (4 x 4)

Fig. 7: 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 `First` dataset. The inference is taken right after the parameter setting(without any training). The subfigure name follows the following naming convention, ModelType (Input Dimension x Hidden Dimension).

C. Perturbation

This experiment aims to explore the loss surface of both Vanilla RNN and GRU. We wish to have an understanding of the basins on the loss surface and after the perturbation model reaches to the minima of the same or a different basin. We perturb the parameters by a perturbation factor and then train the model for 30 epochs. The details of the perturbation are elaborated in the Experiment Setup.

1) Experiment Setup: We take the all the parameter matrices/vectors (mentioned in the equations in section I) and flatten them. 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 times the noise vector is added to the original weight vector to form the perturbed vector. Mathematically,

$$\overrightarrow{W_{pert}} = \overrightarrow{W^*} + \overrightarrow{noise}, \tag{8}$$

$$\overrightarrow{noise} = k\mathcal{N}(0, \sigma_{\overrightarrow{W}}), \tag{9}$$

where $\overrightarrow{W^*}$ is flattened parameter matrix/vector(Actual), $\overrightarrow{W_{pert}}$ is flattened parameter matrix/vector after the perturbation,σ represent the standard deviation, k is the perturbation factor, higher the k more is the perturbation, $\odot$ represents element-wise

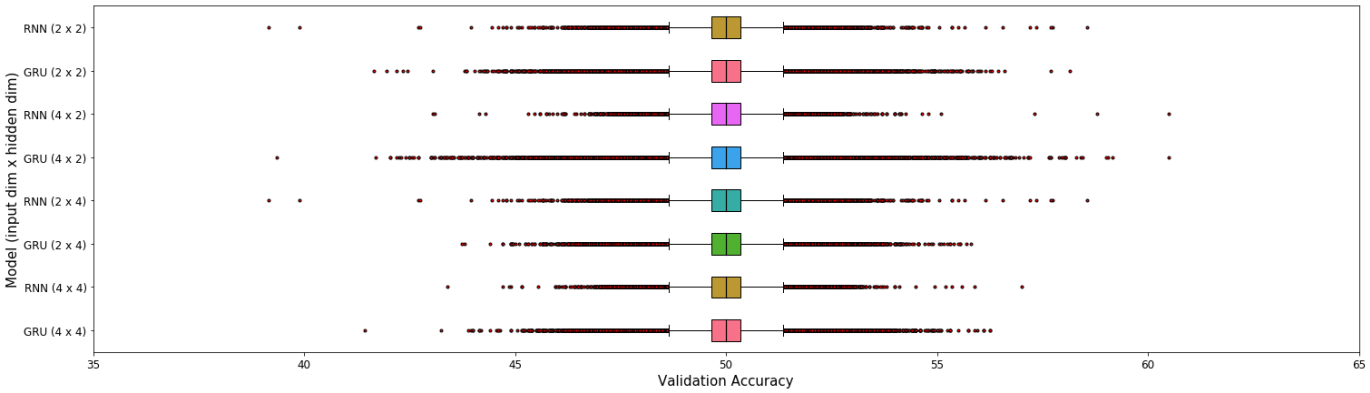
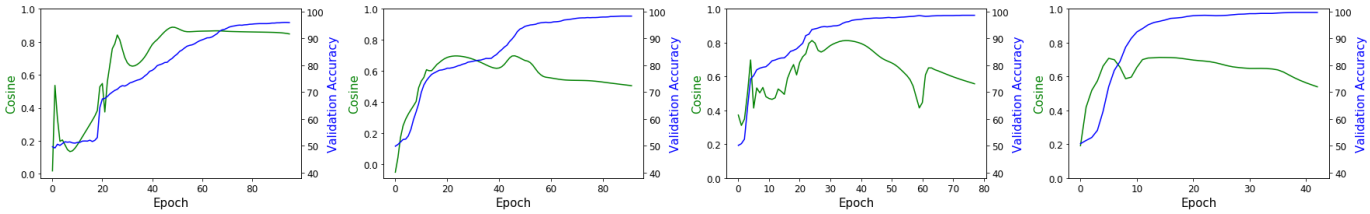


Fig. 8: Each box plot in the figure represent the distribution of million different Initializations based on the different Validation Accuracies that they achieved for a given model on the `First` dataset.



(a) RNN (2 x 4) (b) GRU (2 x 4) (c) RNN (4 x 8) (d) GRU (4 x 8)

Fig. 9: 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 VI(A). The subfigure name follows the following naming convention, ModelType (Input Dimension x Hidden Dimension).

product.

The models of RNN and GRU have 2 input and 4 hidden units. The models were trained to get at least an accuracy of 98% on the validation set. The dataset used is First.

2) Results and Analysis: The plots of this experiments are in Fig. 10. The results clearly declares GRU as the better performer. Even for the perturbation factor of 1, 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 RNN and GRU are more robust to the perturbation.

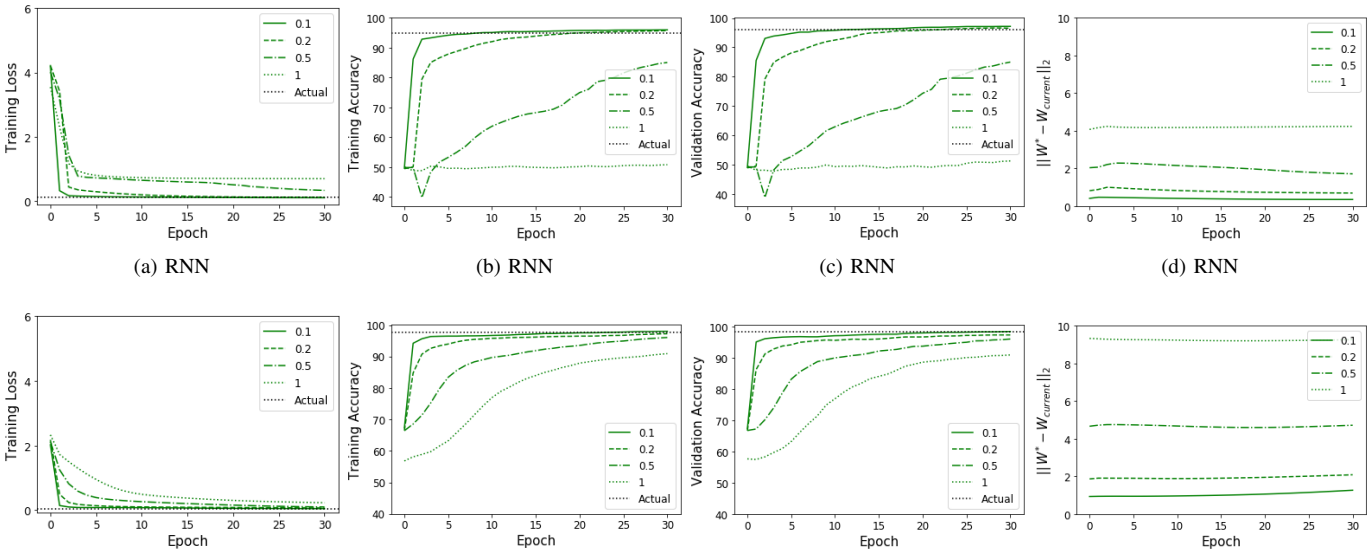
One surprising result is that we have expected the perturbed model weights to return close to the actual weights before perturbation but the difference in norm never gets close to zero in fact it shows a very little change that what we had expected. Another observation is that for the same amount of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

VII. Conclusion

In this paper, we performed the binary classification of the sequences using Vanilla RNN and GRU on the three synthetic pathological datasets. The results reassert the popular belief that GRU have superior performance than the Vanilla RNNs. The results indicate that Vanilla RNN is highly susceptible to the initialization(section V(B)), less robust to the perturbation (section VI(C)) and far more difficult to train than the GRU (section V(A)) despite both having the similar capacity (section V(B)). Also, GRUs result in more generalizable model (observed in section IV.(B)) and have more smooth weight updates than the Vanilla RNN (section VI(B)). Also, we conclude that though 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 VI(A)), the number of good initializations in GRU is higher than that in the Vanilla RNN. This clearly alludes to the fact that though are equally capable to model the given 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.

VIII. Future Work

We have performed the experiments with a basic experiment setup in minimal computing resources. If the sufficient computing resources could be provided, the results could be verified on larger architectures, with even more number of initializations and run experiments for various combinations of hyperparameters so that the results could be more reliable and generalizable.



(a) RNN (b) RNN (c) RNN (d) RNN

(e) GRU (f) GRU (g) GRU (h) GRU

Fig. 10: 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). (d), (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 `First` dataset is used.

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, 2017). Also, after various interesting observations we have made in this paper, we wish to dive deeper into them. With the deeper understanding we step towards our vision of devising better initialization strategies, better learning algorithm and provide better ways to tune the hyperparameters thus saving a lot of precious time and resources invested in the same.

ACKNOWLEDGMENT

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 Computer Science Department, IIT Madras for supporting with all the resources and Google 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 all the phases of the project.

REFERENCES

[1] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5:157–166, 1994.

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

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[4] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks. In *ICLR*, 2017

[5] Kenji Doya. Universality of fully connected recurrent neural networks. *Dept. of Biology, UCSD, Tech. Rep*, 1993.

[6] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*, 2015

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

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL

[9] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1033–1040, 2011

[10] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA*, 16- 21 June 2013, pp. 1310–1318, 2013.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin,Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang and Zachary DeVito and Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, Chintala Soumith. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. 2019. URL

[12] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *ICML (3), volume 28 of JMLR Proceedings*, pp. 1139–1147. JMLR.org, 2013.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.