Recurrent Neural Networks

Tarun Krishna
University of Amsterdam
11593040
tarun.krishna@student.uva.nl

1 Vanilla RNN versus LSTM

1.1 Vanilla RNN in PyTorch

Question 1.1: It seems quite confusing why do we need to compute $\partial \mathbf{L}^T$ with respect to $\mathbf{W}_{hp}$ or $\mathbf{W}_{hh}$ it is supposed to be complete derivative of $d\mathbf{L}^T$ with respect to $\mathbf{W}_{hp}$, $\mathbf{W}_{hh}$.

$$\frac{d\mathbf{L}^{(T)}}{d\mathbf{W}_{hp}} = \frac{\partial \mathbf{L}^{(T)}}{\partial \mathbf{p}^{(T)}} \frac{d\mathbf{p}^{(T)}}{d\mathbf{W}_{hp}}$$

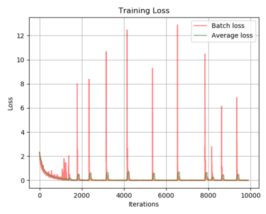
$$\begin{aligned}
\frac{d\mathbf{L}^{(T)}}{d\mathbf{W}_{hh}} &= \frac{\partial \mathbf{L}^{(T)}}{\partial \mathbf{h}^{(T)}} \frac{d\mathbf{h}^{(T)}}{d\mathbf{W}_{hh}} \\
&= \frac{\partial \mathbf{L}^{(T)}}{\partial \mathbf{h}^{(T)}} \left(\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{W}_{hh}} + \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(T-1)}} \frac{d\mathbf{h}^{(T-1)}}{d\mathbf{W}_{hh}} \right) \\
&= \frac{\partial \mathbf{L}^{(T)}}{\partial \mathbf{h}^{(T)}} \left(\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{W}_{hh}} + \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(T-1)}} \left(\frac{\partial \mathbf{h}^{(T-1)}}{\partial \mathbf{W}_{hh}} + \frac{\partial \mathbf{h}^{(T-1)}}{\partial \mathbf{h}^{(T-2)}} \frac{d\mathbf{h}^{(T-2)}}{d\mathbf{W}_{hh}} \right) \right) \\
&= \sum_{k=0}^{T} \frac{\partial \mathbf{L}^{(T)}}{\partial \mathbf{h}^{(T)}} \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(k)}} \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \\
&= \sum_{k=0}^{T} \frac{\partial \mathbf{L}^{(T)}}{\partial \mathbf{h}^{(T)}} \left(\prod_{j=k+1}^{T} \frac{\delta \mathbf{h}_j}{\delta \mathbf{h}_{j-1}} \right) \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}}
\end{aligned}$$

For $k = 0$, $\partial \mathbf{h}^k$ will be a complete derivative with respect to $\mathbf{W}_{hh}$ that is it's a last time step.

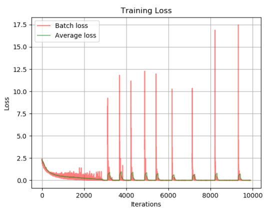
Derivative of $\mathbf{L}^T$ with respect to $\mathbf{W}_{hp}$ depends only on that time step i.e it doesn't have any component of temporal dependencies but for $\mathbf{W}_{hh}$ it expands upto to the starting of the time which arises due to the presence $\mathbf{h}^{(T)}$ which in general is a function of $\mathbf{W}_{hh}$ and $\mathbf{h}^{(T-1)}$ and same analogy repeats for $\mathbf{h}^{(T-1)}$ because $\mathbf{W}_{hh}$ is shared across all the time steps, until $t = 0$, as a result we have temporal dependency over time; as a result gradients are added upto the last time step.

As we just saw $\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(k)}} = \left(\prod_{j=k+1}^{T} \frac{\delta \mathbf{h}_j}{\delta \mathbf{h}_{j-1}} \right)$ is itself subject to chain rule. Which is product of ever expanding Jacobians. Ever expanding because we multiply more and more for longer dependencies. That is further back we look (long-term dependencies), the smaller the weights automatically become as a results it gives rise to the problem of vanishing gradient. Well this product may also lead to the problem of exploding gradients as well when you have constantly increasing gradients.

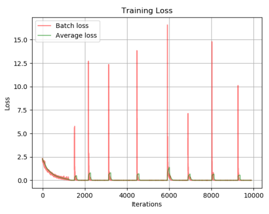
Question 1.2: I experimented with both RMSprop and Adam optimizer. Training loss for different sequence length with RMSprop is depicted in Figure 1. But it seems that Adam perform better than RMSprop as depicted in Figure 2. It easily converges for sequences of length of 5,6,7, for sequence of length 8,9,10 it doesn't converge with default parameters (training step=10000) but increasing the training step for 20000 it converges for sequences of length 8,9 but suffers in case of sequence with length 10 Figure 3. All the models are learnt with default parameters as given in the code.



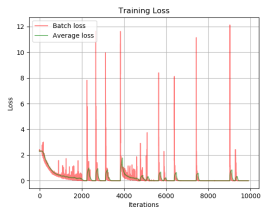
(a) Sequence length 5



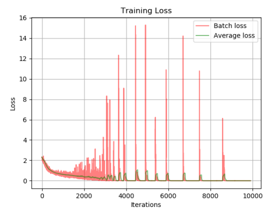
(b) Sequence length 6



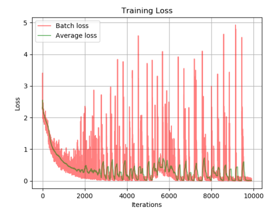
(c) Sequence length 7



(d) Sequence length 8

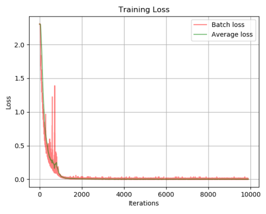


(e) Sequence length 9

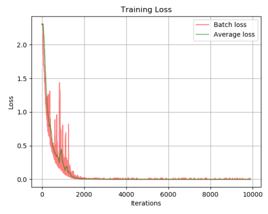


(f) Sequence length 10

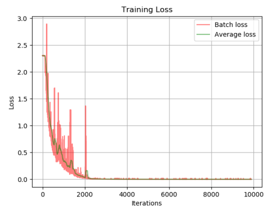
Figure 1: Training Loss for different sequence length using RMSprop as optmizer for Vanilla RNN.



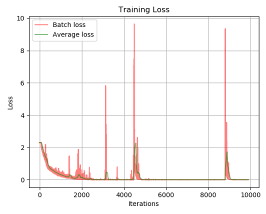
(a) Sequence length 5



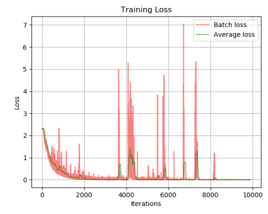
(b) Sequence length 6



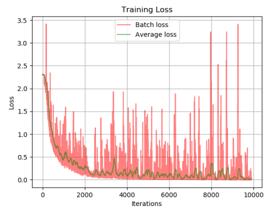
(c) Sequence length 7



(d) Sequence length 8



(e) Sequence length 9

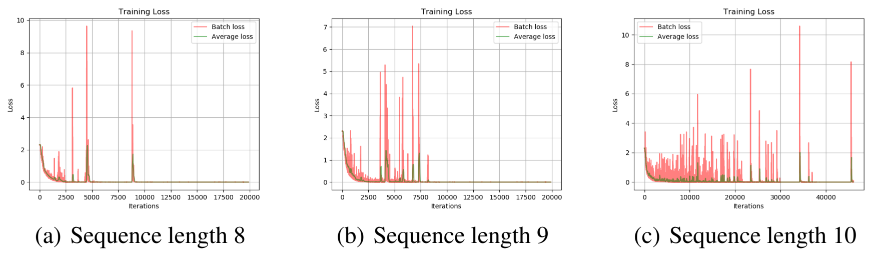


(f) Sequence length 10

Figure 2: Training Loss for different sequence length using Adam as optmizer for Vanilla RNN.

Question 1.3

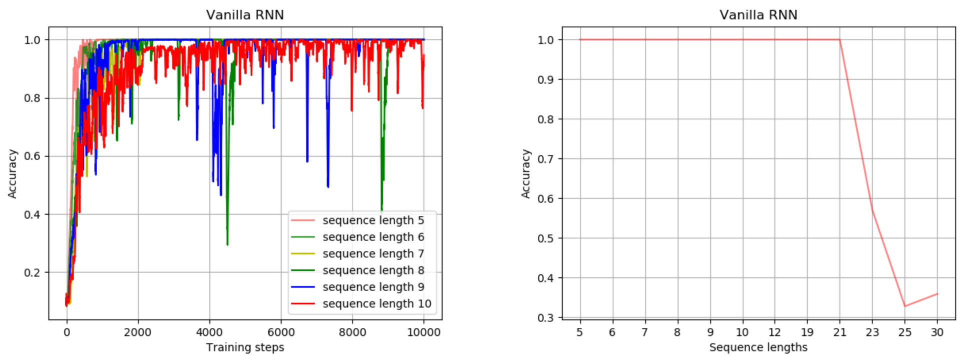
Curve for training accuracy vs sequence length is depicted in Figure 4(b). Preferably for me Figure 4(a) gives better idea about the convergence of accuracy although it's quite cluttered. Figure 4(b) depicts max accuracy achieved across different sequence lengths throughout the training. Increasing sequence length causes drop in accuracy for Vanilla RNNS which can be observed in Figure 4(b). Figure 4(b) doesn't affirms that model got converged for sequnece lengths ≥ 10 . As we previously



(a) Sequence length 8 (b) Sequence length 9 (c) Sequence length 10

Figure 3: Training Loss for more than 10000 training steps using Adam as optmizer for Vanilla RNN.

observed in Figure 3 that even after training for more than 20000 iteration model doesn't converge for sequence length of 10 because of oscillation encountered during the training phase.



(a) Training Accuracy (b) Accuracy vs Sequence length

Figure 4: Training Accuracies for different sequence length using Adam optmizer for Vanilla RNN.

Question 1.4:

Vanilla stochastic gradient descent does not guarantee good convergence, but offers a further challenges that need to be answered such as:

1. Choosing a proper learning rate. A learning rate that is too small leads to slow convergence, while a learning rate that is too large can cause the loss function to oscillate around the minimum or even to diverge.
2. Learning rate schedules are also employed to adjust the learning rate during training by reducing the learning rate according to some schedule. But these schedules and thresholds, are decide before hand and thus are unable to adapt to dataset's characteristics.
3. Additionally, the same learning rate applies to all parameter updates. If our data is sparse and our features have very different frequencies, we might not want to update all of them to the same extent, but perform a larger update for rarely occurring features.

To cope up with that there are some algorithms that are widely used by the deep learning community. Few such methods are momentum, RMSprop, Adam.

SGD has trouble in areas where the surfaces are more steep in one direction than in another, which are common around local optima. In these scenarios, SGD oscillates across the slopes, thus making very slow progress towards local optima. **Momentum** tries to dampens oscillations that arises along the slopes. It does this by adding a fraction γ of the update vector of the past time step to the current update vector(exponentially averaging). Mathematically it could be written as $u_{t+1} = \gamma u_t - \eta \nabla_\theta J(\theta)$ and updare equation becomes $\theta = \theta - u_{t+1}$. Basically momentum tries to accelerate the SGD. As a result, we gain faster convergence and reduced oscillation. But the momentum doesn't solve the problem of adaptive learning rate. Which is taken care by **RMSprop** and **Adam**. RMSprop divides the learning rate by an exponentially decaying average of the squared gradients. It's also to be noted that RMSProp implicitly performs simulated annealing i.e incase of large gradients updates are tamed while in case smaller gradients updates are aggressive. Suppose if we are heading towards the minima, and we want to slow down so as to not to overshoot the minima. RMSProp automatically decreases the size of the gradient steps towards minima when the steps are too large i.e in provides adaptiveness in our updates. So far, we've seen RMSProp and Momentum take contrasting approaches. While momentum accelerates our search in direction of minima, RMSProp impedes our search in direction of oscillations. Adam combines the heuristics of both Momentum with correction bias and RMSProp.

1.2 Long-Short Term Network (LSTM) in PyTorch

Question 1.5:

1. (a) *forget gate* $\mathbf{f}^t$: Forget gate can be considered as remember vector. The output of the forget gate tells the cell state which information to forget by multiplying 0 to a position in the matrix. If the output of the forget gate is 1, the information is kept in the cell state. It makes sense to use *sigmoid* non-linearity whenever a switching operation is required.

   (b) *input gate* $\mathbf{i}^t$: Decides what new information is relevant from the new input and should be added to the new memory. It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 because of *sigmoid* non-linearity. Intuitively it makes sense to use *sigmoid* which acts as a switch gate to decide over the relevance of the information. A 1 represents "completely keep this" while a 0 represents "completely get rid of this". Basically this decides which value we will update.

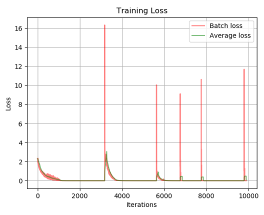
   (c) *modulation gate* $\mathbf{g}^t$: The purpose of *modulation gate* is to generate the new candidates values or new information based on current $\mathbf{x}^t$ and previous $\mathbf{h}^{t-1}$. To understand the intuitiveness behind *tanh* non-linearity we need to understand the equation $\mathbf{c}^t = \mathbf{g}^t \odot \mathbf{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t$. The purpose of $\mathbf{f}^t$ is to remember what is required from the previous $\mathbf{c}^{t-1}$ and on the other hand $\mathbf{i}^t$ decides what is relevant i.e these gates could be understood as 'taps' that control how much of the 'water' should flow. Thus the 'taps' can go from 0(fully closed) to 1 (fully open), that is, it does not make sense for the taps to go negative. But my new information that is given by $\mathbf{g}^t$ could be any range so it make sense to use *tanh*. Well, one can also argue, then why not *ReLU* and all. Well, I couldn't find any such papers where they did some experiments with *ReLU*.

   (d) *output gate* $\mathbf{o}^t$: Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. That is, we run a sigmoid layer which decides what parts of the cell state we're going to output, same analogy as above to decide flow of information. And the use of sigmoid is justified as well.

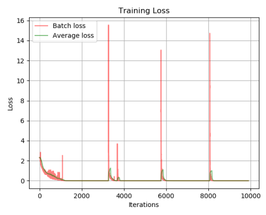
2. Number of trainable parameters(including bias): $4(nd + n^2 + n)$

Question 1.6:

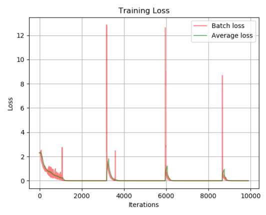
In this part again, I experimented with both the optimizers namely Adam and RMSprop, keeping all other parameters to default as given in the code. As a result adam seems to converge for sequences of length 5,6,7,8,9,10. Figure 5 depicts the training loss using RMSprop. Figure 6 depicts training loss using Adam which converges for varying sequence length within 10000 training steps except in the case of sequence length of 10 one has to increase the learning rate to 0.01. Accuracy convergence can also be affirmed from Figure 7 using Adam optimizer. If we compare the training curves in Figure 2, 3 with curve in Figure 6 we could easily spot the difference between convergence of both the approaches. Well it is pretty obvious why rnn fails, they are not good for long term dependencies in contrary to LSTMs that has special units (long term memory) in addition to standard units. LSTM units include a 'memory cell' that can maintain information in memory for long periods of time. Well apart from that one can argue about exploding/vanishing gradient but these are results of long dependencies which RNN is not able to tackle. Which is further validated from the Figure 3 where even training for more than 40000 steps RNN fails to converge for sequence length of 10.



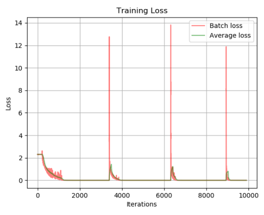
(a) Sequence length 5



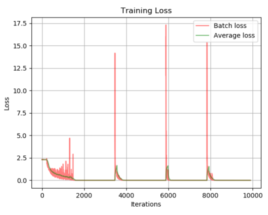
(b) Sequence length 6



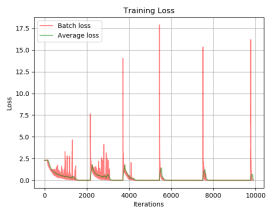
(c) Sequence length 7



(d) Sequence length 8

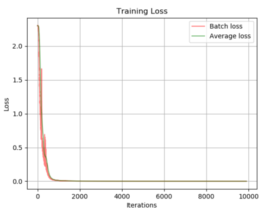


(e) Sequence length 9

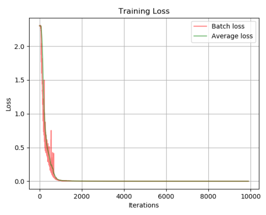


(f) Sequence length 10

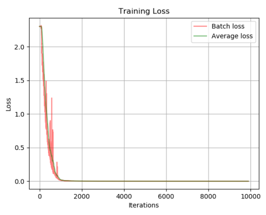
Figure 5: Training Loss for different sequence length using RMSprop as optmizer for LSTM.



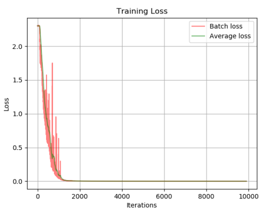
(a) Sequence length 5



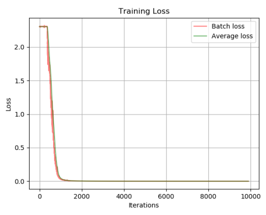
(b) Sequence length 6



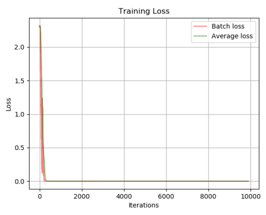
(c) Sequence length 7



(d) Sequence length 8



(e) Sequence length 9



(f) Sequence length 10

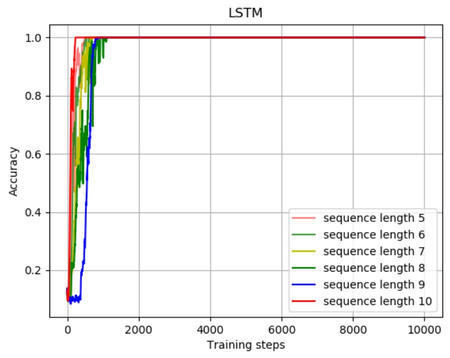
Figure 6: Training Loss for different sequence length using Adam as optmizer for LSTM. Seqeuence length was trained with learning rate of 0.01.

2 Modified LSTM Cell

Question 2.1 Figure 8.

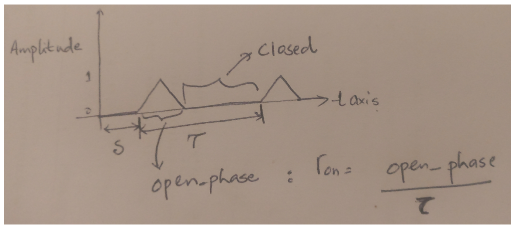
Question 2.2

The updates of the cell state in original LSTM occurs every time steps, i.e at every input sequence. Any long term memory state, say c_0 will decay exponentially over time if no addition input is added. $c_n = f_n \odot c_{n-1} = (1-\epsilon) \odot c_{n-2} = \cdots = (1-\epsilon)^n \odot c_0$ assuming fully opened forget $(f_j = 1-\epsilon)$. Adding a temporal gate allows updating of cell state only during $r_o n$. This means all cell state values need not get updated at every step which allows to preserve long term cell values. Because of this cyclic memory, Phased LSTM can have much longer and adjustable memory length via the parameter τ. The oscillations impose sparse updates of the units, therefore substantially decreasing



(a) LSTM

Figure 7: Training accuracy for different sequence length using Adam as optmizer for LSTM.



(a)

Figure 8: Phased LSTM

the total number of updates during network operation. During training, this sparseness ensures that the gradient is required to backpropagate through fewer updating timesteps, allowing an undecayed gradient to be backpropagated through time and allowing faster learning convergence.

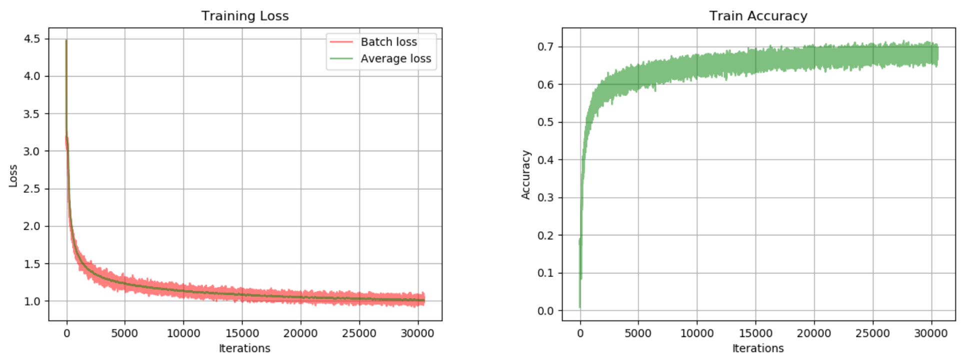
This behavior is expected in scenarios where there are asynchronous input features are feed to the network. The network parameters, will adjust the phase-shift, τ will fit to the real-time oscillation of the input features and the parameter r_on will decide the update strategy of cell state. These could be beneficial in many real-world tasks for autonomous vehicles or robotics where there is need to integrate input from a variety of sensors, e.g. for vision, audition, distance measurements, or gyroscopes. Each sensor may have its own data sampling rate, and short time steps are necessary to deal with sensors with high sampling frequencies.

Question 2.3 In the original paper [1] they talk about learning these parameters but hey didn't mention how they do it. So, as mentioned in [1] each time gate is controlled independent rhythmic oscillation which controlled by τ, r_{on} and s. The parameter τ denotes the time period of the oscillations. This parameter can be trained during training as it is data dependen i.e it depends upon timing of the arrival of the input data. The parameter r_on can also be trained during training however, it will make the model quite difficult to converge as the upper bound of r_on is dependent on τ. The parameter s is the phase-shift of the oscillating time period,which can also be learned. These parameters can be thought of as input to relu which has just two conditions (>0 or $leq0$) contrary to our case as defined by eq 12 in the assignment then it becomes more intuitive how we can learn these parameters.

3 Recurrent Nets as Generative Model

3.1 Question

(a) With default learning parameters I was able to obtain $\approx 70\%$ accuracy within 30000 training steps. The book used for this task was `Grimms Fairy Tails` which contains 540241



(a) Training loss (b) Accuracy loss

Figure 9: Training and accuracy curve for `Grimms Fairy Tails`. These curves are obtained with default learning parameters.

Training steps	Generated sentences
100	`t`
5000	`quite all the second said: ‘I w`
10000	`the second son was come out and`
15000	`He was asked to be a sin and th`
20000	`d the soldier said, ‘I will giv`
25000	`Once in the morning`
30000	`‘Ah,’ she replied,`

Table 1: Generated sentences from the trained model over evenly spaced intervals.

characters, 87 unique characters that's basically the vocabulary size. Loss and Accuracy curves can be observed in Figure 9.

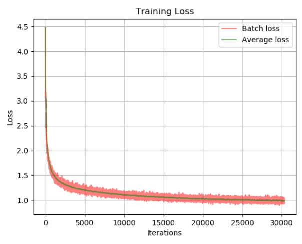
(b) Generated sentences over evenly spaced intervals are depicted in Table 1. Just a note about how the character sequences were generated. Initially a random character was generated as starting seed for the network and the argmax of the output at current time step was fed as the input to the next time step and so on. That is, a greedy approach was used to generate the character sequences.

Sentences generated initially seems to carry repetitive pattern as observed in Table 1 for 100 training steps where spaces are dominant. Same behaviour can be also be observed for intermediate steps as well[1]. As we further train the model it seems to generate better results but won't say meaningful. It learns to put punctuation, starting to generate capital letters at the beginning of the sentence as the training progresses. To me best result seems to be generated at 30000 steps `‘Ah,’ she replied,` which looks nearly perfect in all the aspects i.e grammar, punctuation etc. Also it is impressive to see how quickly it learns to output correct spellings.

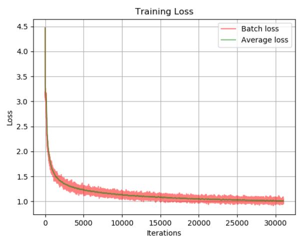
(c) In this experiment we try to add Temperature parameter to change the dynamics of our network. The loss and accuracy curves for the different temperature values is depicted in Figure 10. It looks for T=0.5 it converges early as compared to our previous implementations i.e without including T as depicted in Fgiure 9.

To me it was unclear that even after training with T do we still need to sample greedily or randomly. So, I did a bit of tinkering with both the approaches. Well, if we compare previous results in Table 1 with results in Table 2 (greedy). Decreasing the temperature from 1 to some lower number (e.g. 0.5) makes the model more confident, as it can be seen with the generated text which are more meaning full as compared to previous. Model seems to give pretty impressive results like "`quickly to the king said: ‘I wi `", "`He got up`

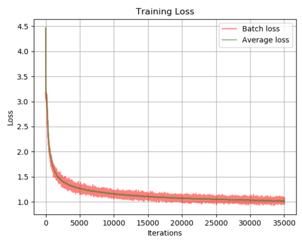
[1]Results for intermediate generation can be seen under summaries folder



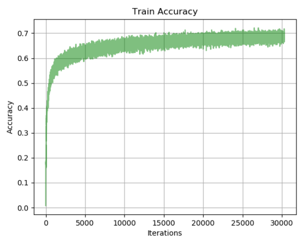
(a) Training Loss, T = 0.5



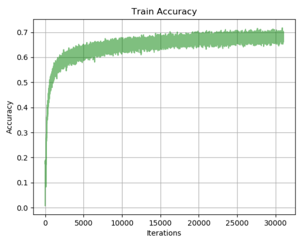
(b) Training Loss, T = 1



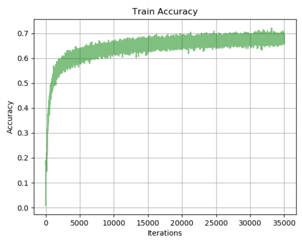
(c) Training Loss, T = 2.0



(d) Training accuracy, T = 0.5



(e) Training accuracy, T = 1.0



(f) Training accuracy, T = 2.0

Figure 10: Training loss and accuracy curves for models with different T parameter.

`and went to the bear’"` at a very early stage. T greater than 1 provides more diversity to the network as a result this may lead slower convergence of the model. If we compare the generated text corresponding to each training steps under varying T from Table 2 we could observe the change in the output for eg. `quickly to the king said: ‘I wi` (T=0.5), `quite all the second said: ‘I w`, T=1.0, `quietly and said: ‘I will see a` (T=2.0) we easily observe the change arising due to change in T parameter which tends to make the distribution uniform for large T as result inducing variety to our output but at the cost of mistakes.

All the points mentioned above are equally valid for random sampling as well but it seems that due to randomly sampling a character induces more spelling mistakes in our output which further increases as we increase T.

For more generated text one can have a look at summaries folder with text generated at intermediate steps throughout the training. Also the results depicted in Tables might not have 30 characters that’s probably because I might have omitted some of the spaces.

3.2 Bonus

In this experiment I just played around with my trained model. I gave initial sentence as a seed to the network to form a paragraph about it. Bold texts are the inputs and rest is the story genrated by the model. To me these completion looks quite interesting.

At first Rapunzel was terribly `the wolf was lying in the mountains, and I will sing in return for your pocket where it had gone on doing this way that he might go anothing. He was still more than anyone that they might lie Underneath the juniper-tree Kywitt, Kywitt, what a beautiful bird am I!’ And when he had gone on his head, and he said, ‘I will not do it; you shall I see that the wolf had told him that he had seen gold and solitude lowing the castle was a great castle of Stromberg. When the second son was alone in t`

the old woman had only pretend `the wolf had to sprinkled the door and the cook was gone, and the golden cage, and said: ‘What do you desire?’ asked the maiden. ‘What a while, and at last the blood many below. 1.F.5. The`

	Greedy	Random
Training steps	**T=0.5**	
100	t the the the the the the the t	tkrinh, gove. geshe, er thennr
5000	quickly to the king said: ‘I wi	quite first with his night-‘Kyw
10000	the sea, and thought to himself	thein fellow, ‘I did not know w
15000	He got up and went to the bear’	HE ELVES HANS ANDY AR GRTT
20000	On the world to see the man to	Oh, he saw the whole keating th
25000	D THE FOUR CLEVER BROTHERS	Dgir,’ ‘I thought she pulled he
30000	4, ‘I will not do it in the wor	4 LITTLE KINS
	T=1.0	
100	t	
5000	quite all the second said: ‘I w	totofodeemu mafesleh er soerst
10000	the second son was come out and	quite form countily busiendy wi
15000	He was asked to be a sin and th	the Gredop said he will remaine
20000	d the soldier said, ‘I will giv	OTH Hasse have seen his door
25000	Once in the morning	Dirsellical. This once a wretch
30000	‘Ah,’ she replied,	4 FERVEAN LINT The young o
	T=2.0	
100	t	totoeodeemu mLfdsleh dr sodrss
5000	quietly and said: ‘I will see a	quite herp. ‘What must makey yo
10000	the castle of the stream. The s	the Mosal donaw feet him; and w
15000	He had not still and the servan	HE BEAR! THE WAS WALNT TO
20000	One day the second son went to	OW AND THE SAORY There was on
25000	D THE SALDING PRIMEDESAND	Dand threat was. Now he cut off
30000	I am so have a daughter was	4 FREDELICEK AND *** Ther

Table 2: Generated sentences from the trained model over evenly spaced intervals.

```
Brothers Grimm Vano made a fine thing to eat, and when he had gone on his
way home they saw a long way off with you.’  The first day the little man
took him away again, and he said, ‘I will not do it; you shall I see that
the wolf had told him that he had seen gold and solitude lowing the castle
was a great
```

References

[1] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased LSTM: accelerating recurrent network training for long or event-based sequences. *CoRR*, abs/1610.09513, 2016.