Deep Learning Practical 2

Andrei Marius Sili*
Master's in Artificial Intelligence
University van Amsterdam
andrei.sili@student.uva.nl

Part 1: Vanilla RNN versus LSTM

1.2 Vanilla Recurrent Neural Network

Question 1.1

The expression of the derivatives in index notation are provided below.

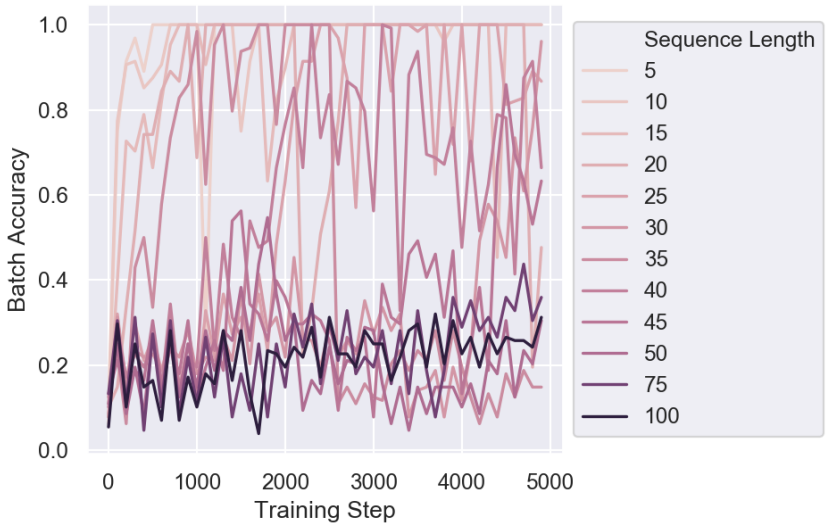
$$\begin{aligned}
\left(\frac{\partial L^{(T)}}{\partial \mathbf{W}_{ph}}\right)_{ij} &= \frac{\partial L^{(T)}}{\partial p_i^{(T)}} \frac{\partial p_i^{(T)}}{\partial \left(W_{ph}\right)_{ij}} \\
&= \frac{\partial L^{(T)}}{\partial p_i^{(T)}} \frac{\partial \sum_k \left(W_{ph}\right)_{ik} h_k^{(T)}}{\partial \left(W_{ph}\right)_{ij}} \\
&= \frac{\partial L^{(T)}}{\partial p_i^{(T)}} \sum_k \delta_{kj} h_k^{(T)} \\
&= \frac{\partial L^{(T)}}{\partial p_i^{(T)}} h_j^{(T)} \\
\left(\frac{\partial L^{(T)}}{\partial \mathbf{W}_{hh}}\right)_{ij} &= \frac{\partial L^{(T)}}{\partial h_i^{(T)}} \frac{\partial h_i^{(T)}}{\partial \left(W_{hh}\right)_{ij}} \\
&= \frac{\partial L^{(T)}}{\partial h_i^{(T)}} \frac{\partial h_i^{(T)}}{\partial \sum_k \left(W_{hh}\right)_{ik} h_k^{(T-1)}} \frac{\partial \sum_k \left(W_{hh}\right)_{ik} h_k^{(T-1)}}{\partial \left(W_{hh}\right)_{ij}} \\
&= \frac{\partial L^{(T)}}{\partial h_i^{(T)}} \left(1 - \left(h_i^{(T)}\right)^2\right) \left(h_j^{(T-1)} + \left(W_{hh}\right)_{ij} \frac{\partial h_j^{(T-1)}}{\partial \left(W_{hh}\right)_{ij}}\right)
\end{aligned}$$

The first gradient of the loss w.r.t $\mathbf{W}_{ph}$ depends only on the current time step, so from an optimisation perspective it is nothing new. What can be observed is that the gradient of the loss w.r.t $\mathbf{W}_{hh}$ contains a recurrent term: the partial derivative of the previous hidden state $\mathbf{h}^{(T-1)}$ w.r.t $\mathbf{W}_{hh}$. This means that the gradient of the loss is dependent on all the previous time steps, because the recurring partial derivative expands also into a term that contains the partial derivative of the hidden state before it $\mathbf{h}^{(T-2)}$ and so on.

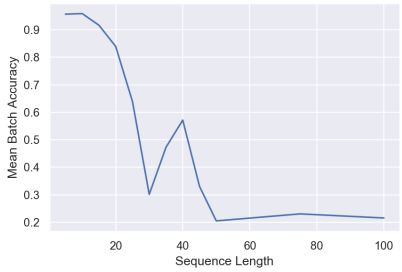
From an optimisation perspective, the issue arises that we are multiplying an element of the matrix $\mathbf{W}_{hh}$ with itself for every time step. This leads to exploding or vanishing gradients, depending on whether the value of the element is high or low.

*Student ID: 11659416

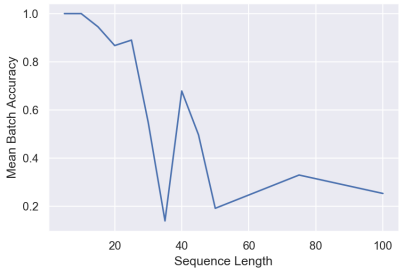
32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.



(a) Vanilla RNN Mean Batch Accuracy at Different Sequence Lengths.



(b) Mean Batch Accuracy Over Entire Training Steps for Varying Sequence Lengths.



(c) Mean Batch Accuracy Over Final 1000 Training Steps for Varying Sequence Lengths.

Figure 1: Vanilla RNN Mean Batch Accuracy. *The figures above show that the Vanilla RNN converges rapidly to high accuracy for short sequences but fails to perform for long ones. For sequences shorter than 30, the model converges quickly but shows massive erratic drops in accuracy from time to time. Any sequence of length 30 and above does not seem to go above 30% accuracy.*

Question 1.3

I trained the Vanilla RNN on sequences of length 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 75, and 100. I used RMSProp for optimisation with the provided default parameters, a batch size of 128, and I trained the network for 5000 steps for each sequence. Weights are initialised using the Xavier initialisation strategy and gradients are clipped at every backward pass to prevent erratic updates. Overall it seems that the network converges to good performance, at least 90% accuracy, for sequences of up to 25. For any sequence longer than that the network is not able to capture the long-term dependencies. Another thing to observe is that even for converging models, mean batch accuracy shows sudden substantial drops from time to time. Word around the block is that this is due to the RMSProp optimiser, but I haven't tried different optimiser strategies to see the difference. Figure 1a shows a comparison of the mean batch accuracy at every time step for the Vanilla RNN model trained on different sequence lengths. I also show aggregated mean batch accuracy over the entire training steps and for the final 1000 steps in Figures 1b and 1c. Comparing the 2 plots, it can be seen that for short sequences up to length 25, the Vanilla RNN converges rapidly, as both the mean accuracy over all training steps and the mean accuracy over only the last 1000 steps are relatively close.

Question 1.4

RMSProp and Adam, in general, work by maintaining some form of history of past gradients and using this information to tune the learning rate of each parameter individually, conditional on a general learning rate specified as a hyper parameter. They use the information of past gradients to compute an adaptive learning rate for each parameter.

RMSProp does this by maintaining an exponentially decaying history of squared gradients at every time step r_t. This is an estimate of the second order moment, the variance of the gradients. This history is used to adapt the learning rates of each parameter in such a way that large gradients get tamed and small gradients get boosted via the square root in the denominator.

$$r_t = \alpha r_{t-1} + (1-\alpha) \odot g_t^2$$
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{r_t} + \epsilon} g_t$$

Adam extends RMSProp by adding a momentum term that holds the history of the gradients without squaring them. This is an estimate of the first order moment (mean). Because the gradients are not squared, they may cancel out, meaning that gradients that are small but consistent will get boosted over time, and the ones that are large but point in different directions will get squashed. The first and second order moments act as 2 forces working to adapt the learning rate of each parameter individually. A note is that the moments are biased toward 0 in the initial stages because they are initialised to 0, which may be corrected for as shown below.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2}$$
$$w_{t+1} = w_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

1.3 Long-Short Term Memory Network

Question 1.5

Part a In general, the gates in an LSTM serve to control what information gets passed on from one cell to the next, and how that information is processed. Apart from the output gate, they all work by controlling information stored in the cell's context $\mathbf{c}^{(t)}$. The input modulation gate $\mathbf{g}^{(t)}$ creates a non-linear representation of the information contained in the current input $\mathbf{x}^{(t)}$ and the previous hidden state $\mathbf{h}^{(t-1)}$. The input gate $\mathbf{i}^{(t)}$ controls how much of the "modulated" input $\mathbf{g}^{(t)}$ is used to update the cell context $\mathbf{c}^{(t)}$. The forget gate $\mathbf{f}^{(\mathbf{t})}$ controls how much of the previous cell context $\mathbf{c}^{(t-1)}$ to keep when updating $\mathbf{c}^{(t)}$. Finally, the output gate $\mathbf{0}^{(t)}$ controls how much of the updated context $\mathbf{c}^{(t)}$ to pass on to the cell hidden state $\mathbf{h}^{(t)}$.

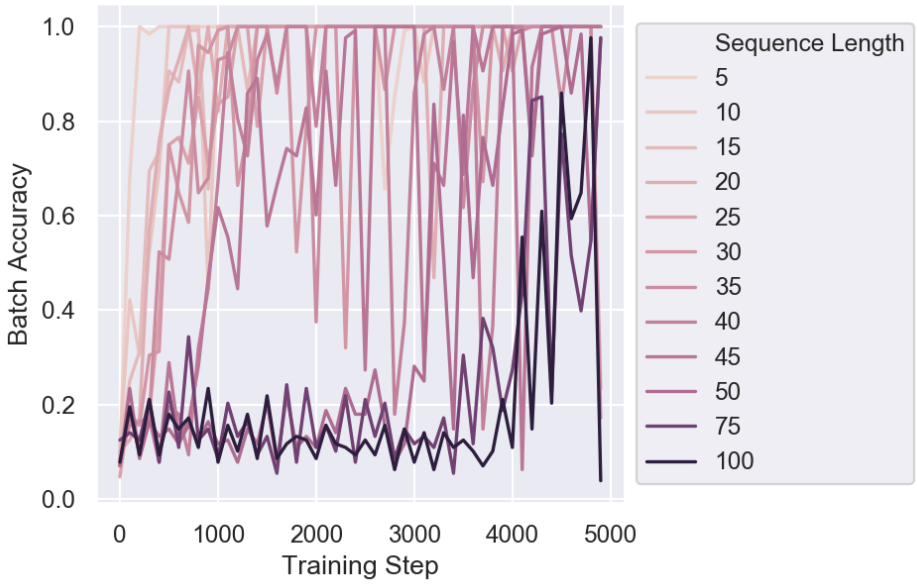
The input modulation gate $\mathbf{g}^{(t)}$ uses a TanH non-linearity. In principle, this gate could use any non-linearity, but a TanH is chosen because it is centered around 0 so it is not biased towards positive values, and has a stronger gradient around this point, as compared to the Sigmoid. Another popular choice is the ReLU because it has stronger gradients. All the other gates, $\mathbf{i}^{(t)}$, $\mathbf{f}^{(\mathbf{t})}$ $\mathbf{0}^{(t)}$ use a Sigmoid non-linearity because they need to act as 1 and 0 switches. Since the Sigmoid smoothly interpolates between these 2 extremes, it is a natural choice for a switch gate.

Part b In the calculation for the number of parameters, the sequence length T, and the batch size m do not play a role. All four gates have the same number of parameters since they all involve the same linear mapping. The number of parameters in one gate can be calculated as follows.

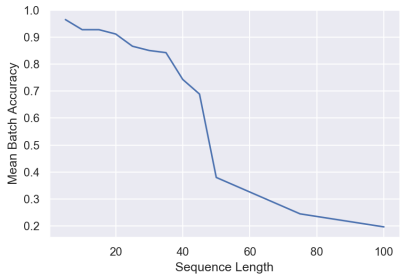
$$\#params_{gate} = n \cdot d + n \cdot n + n$$

This means that the total number of parameters is just the previous term multiplied by 4:

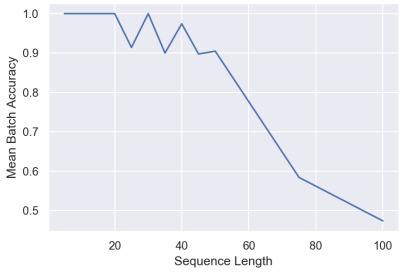
$$\#params_{lstm} = 4 \cdot \#params_{gate} = 4\,(n \cdot d + n \cdot n + n)$$



(a) LSTM Mean Batch Accuracy for Different Sequence Lengths.



(b) Mean Batch Accuracy Over Entire Training Steps for Varying Sequence Lengths.



(c) Mean Batch Accuracy Over Final 1000 Training Steps for Varying Sequence Lengths.

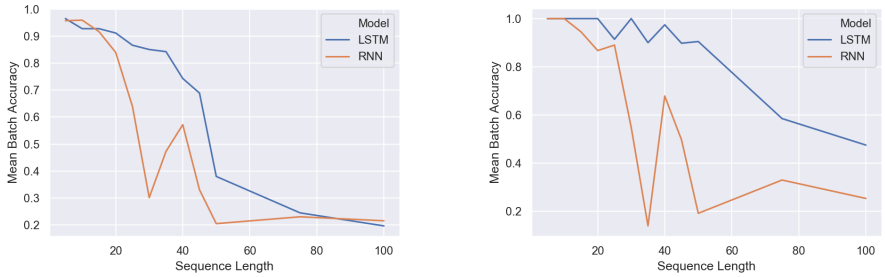
Figure 2: LSTM Mean Batch Accuracy. *The figures above show that the LSTM converges rapidly for short sequences, struggles to converge for moderate sequence lengths, and does not converge for very long ones (75,100). For sequences of up to 50, the model converges but still shows massive erratic drops in accuracy from time to time. The model is not able to memorise sequences of 75 and 100 digits.*

If we consider the the final linear map in the LSTM, then we can add those to the number of parameters like the following, where o is the output dimensionality.

$$\#params_{map_{lstm}} = 4\left(n \cdot d + n \cdot n + n\right) + o \cdot n + n$$

Question 1.6

As with the Vanilla RNN, I trained the LSTM on sequences of length 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 75, and 100. I used RMSProp for optimisation with the provided default parameters, a batch size of 128, and I trained the network for 5000 steps for each sequence. Weights are also initialised using the Xavier initialisation strategy and gradients are clipped at every backward pass to prevent erratic updates. Overall it seems that the network converges to good performance, at least 90% accuracy, for sequences of up to 50. Going longer to 75 and 100 length sequences, the LSTM fails to memorise the sequence as well. Dropping the learning rate by a factor of 10 did not seem to make a difference for these long sequences, but increasing it by a factor of 10 did result in 100% accuracy for the LSTM



(a) Mean Batch Accuracy Over Entire Training Steps for Varying Sequence Lengths.

(b) Mean Batch Accuracy Over Final 1000 Training Steps for Varying Sequence Lengths.

Figure 3: Comparison of Mean Batch Accuracy. *The left plot shows accuracy for the entire training procedure. The right one shows accuracy for only the last 1000 steps of the procedure. It can be seen that the LSTM substantially outperforms RNN on all but the very short sequences, where it still outperforms but only by a small margin. Looking at the plot to the left, we can also see that the LSTM converges slightly slower for very short sequences since the average accuracy is lower over the entire training procedure.*

trained on sequences of length 75. Results of varying the learning rate are not reported for brevity. Again, we see erratic dips in accuracy for converging models but less so that in the case of Vanilla RNN. Figure 2a shows a comparison of the mean batch accuracy at every time step for the LSTM model trained on different sequence lengths. I also show aggregated mean batch accuracy over the entire training steps and for the final 1000 steps in Figures 2b and 2c.

Comparing Vanilla RNN with LSTM it is clear that the LSTM has more memorisation capacity, although not unbounded. For sequence of length 30 and above, the RNN is no longer capable of modelling the dependencies between the first and last digit, while the LSTM can do so for sequences of up to length 75. On the other hand, it seems that the RNN converges slightly faster for shorter sequences. Figures 3a and 3b show aggregated mean batch accuracy over the entire training steps and for the final 1000 steps.

The Vanilla RNN converges faster likely because it has much less parameters to tune so the search space is more restricted. On the other hand, the the LSTM performs much better because it is designed in such a way to allow strong gradients to pass from one cell to the next (or previous, depending on how you look at it) in the backward pass. Specifically, the context $\mathbf{c}^{(t)}$ is not replaced at every stage, but is only altered additively by the current cell, using gates that control how much ($\mathbf{i^{(t)}}$, $\mathbf{f^{(t)}}$) and in what way ($\mathbf{f^{(t)}}$) it is changed. This context parameter is what allows the network to retain much more of the long-term dependencies as opposed to the simple hidden state of the Vanilla RNN.

1 Part 2: Modified Long-Short Term Memory Cell

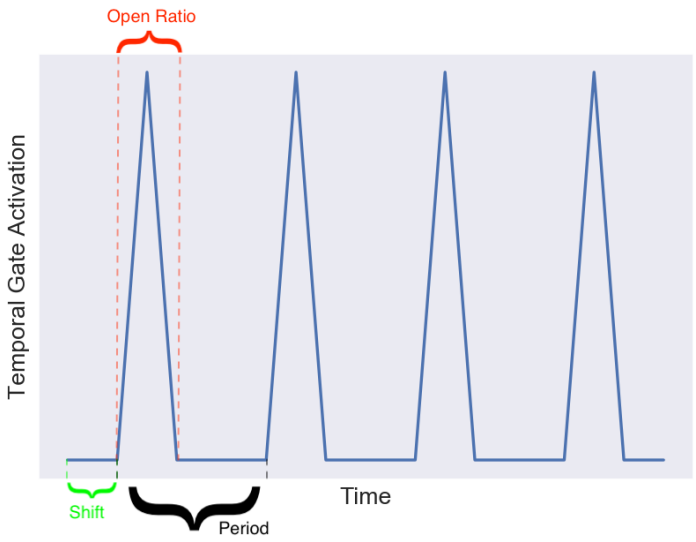
Question 2.1

Figure 4a shows a conceptual representation of the temporal gate $k(t)$. Figures 4b, 4c, and 4d show the effect of varying parameters on the activation of the temporal gate.

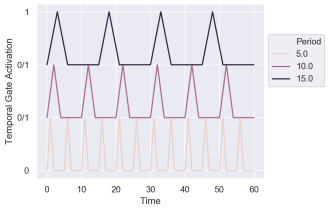
Question 2.2

The main addition of the temporal gate is that it allows the LSTM cell to explicitly control the frequency at which it processes the sequence input signal and the length of time over which it processes it. The main use of such a model is in situations where the model receives data asynchronously from multiple sources, meaning that neither the sampling rate, nor the sampling interval are fixed. This explicit modelling of processing frequency is achieved by incorporating the temporal gate in the context and hidden state update equations.

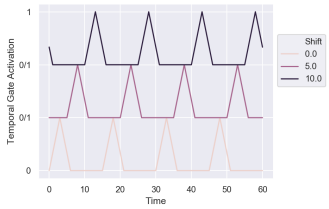
Consider the following scenario. A model receives input sequences from 2 sensors. One samples at a rate of 1 data point per millisecond, and the other samples at a rate of 1 per second. In a typical



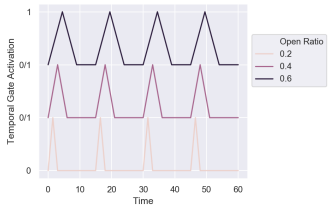
(a) Temporal Gate Illustration. *Not my finest work.*



(b) Temporal Gate Activation for Varying Periods. ($s = 0$, $r_{on} = 0.4$) *As the period increases, a cycle takes longer to complete. This means that the type elapsed between activations increases linearly with the period. This parameter controls the length of a cycle.*



(c) Temporal Gate Activation for Varying Shift. ($\tau = 15$, $r_{on} = 0.4$) *As the shift increases, the cycles do not change in length of activation behaviour, but they get pushed to the right on the time axis. This parameter controls the starting point of a cycle within a time-frame, sort of like a bias.*



(d) Temporal Gate Activation for Varying Open Ratio. ($\tau = 15$, $s = 0$) *As the open ratio increases the amount of time within a cycle that the activation is non-zero increases with the open ratio. This parameter controls the proportion of the cycle for which the activation is non-zero.*

Figure 4: Temporal Gate Illustration and Activation for Varying Parameters τ , s, and r_{on}. *The 3 plots show the effect of varying one parameter while keeping the other 2 constant. In all plots, the leaky hyper-parameter α is held constant at* 0.

LSTM, the gradient signal of the samples from the second sensor would vanish or would be corrupted in some way since we have only one point per 1000 points of the other sensor. The modified LSTM cell, however, can learn to keep some neurons with a very long period and a small open ratio. What this means is that at almost all time steps, the gate will be closed, so the context and hidden state will simply be copied over unmodified from one time step to the next. This outcome of leaving context and hidden states constant over time is also useful in modelling very long-term dependencies. Moreover, because activations can be made very sparse, this cell also provides theoretical computational benefits.

In short, the modified LSTM cell, is better suited to model sequences of asynchronous inputs, very long-term dependencies and is more computationally efficient due to its sparse representation.

Question 2.3

The period tau controls the length of one cycle within a time sequence for the temporal gate. The shift s controls the displacement of the temporal gate cycle within a time sequence. The open ratio

	Model A	Model B	Model C	Model D
LSTM Neurons	128	128	128	128
LSTM Layers	2	2	2	2
Dropout Probability	0.5	0.5	0.5	0.5
Temperature	0.50	1.00	1.50	2.00
Sequence Length	30	30	30	30
Batch Size	64	64	64	64
Train Steps	2000	2000	2000	2000
Learning Rate	0.01	0.01	0.01	0.01
Learning Rate Decay	0.90	0.90	0.90	0.90
Learning Rate Decay Step	1000	1000	1000	1000
Maximum Gradient Norm	5.00	5.00	5.00	5.00

Table 1: Model Hyper-parameters. *All hyper parameters are kept constant across models except for Temperature.*

r_{on} controls the proportion of time in one cycle for which the activation of the temporal gate is non-zero.

All parameters can be learned through back-propagation. Some care must be taken at point around the beginning of cycle where $(t - s) \bmod \tau = 0$ since the derivative is theoretically undefined as the function is discontinuous. In practice, I think we can consider the derivative to be equal to 1 everywhere, similarly to what is being done with the ReLU.

2 Part 3: Recurrent Nets as Generative Model

Question 3.1

Part a The LSTM Model I've implemented for sequence-to-sequence classification follows the assignment guidelines for the most part, with some small adjustments. Most notably, I use a LogSoftmax as the final output layer to prevent numerical underflow and pass this to PyTorch's Negative Log Likelihood Loss function. Dropout is used on the LSTM layers to encourage redundant representation of features. The model also expects a temperature constant which is used to divide the energies arising from the linear map of the LSTM hidden states. I train 4 models, where all the hyper-parameters are held constant at the default values provided, except for temperature. Training is accomplished using the Adam optimiser with default hyper-parameters, and a decaying learning rate. All models are trained on the US Bill Of Rights[2]. The hope is that, by the end of this exercise, we will be able to start a new nation based on an AI powered constitution. An overview is given in Table 1.

Part b Table 2 shows several samples from the Model A with a temperature of 1, so basically no temperature. We can see that as the training process evolves, the samples become more expressive, and contain less spelling or other types of errors. In particular it also seems that the model escapes local modes since the sentences seem to be quite varied, and do not show word loops. Table 2 shows samples at different training steps on a pseudo-exponential scale. Because the training corpus is rather small, the model converges quite quickly, generating nice sentences already at training step 500.

Part c By using different values for the constant T, I train 3 other models. The expectation is that by increasing T above 1, the model will produce more varied results, while decreasing it will make the model focus on only one or a few modes of the data distribution.

However, note that, since the constant T is included in the model during training, the network may easily adapt to it and completely undo its effect by simply scaling all weights and biases accordingly. Consider the linear mapping that outputs energies for each possible character based on the hidden states of the LSTM. If we apply an element-wise multiplication of the inverse temperature T, we can observe that the network is in principle able to learn a different mapping that neutralises the effect of

[2]Available at: http://www.gutenberg.org/cache/epub/2/pg2.txt

Training Step	Greedy Sample
1	grrr
100	X The Constitution and comsens
200	J people to keep and bear arms,
300	le to be secure in thein perdis
400	Bill of the people to keep and
500	shall make no law respecting an
1000	ce to be searched, and the pers
1500	y oath or affirmation, and part
2000	25, 1789 Ratified December 15,

Table 2: Greedy Samples from Text Generation Model without Temperature. *The first 500 samples are either gibberish or contain typos. From 500 onward, the samples seem to be well diversified, make sense within their local context and do not present any mistakes. In other words, the model has converged.*

the temperature constant completely by scaling all weights and biases by a factor of T:

$$\begin{aligned}\mathbf{p} &= \frac{1}{T}\left(\mathbf{W}_{ph}\mathbf{h}^{(n)} + \mathbf{b}_p\right) \\ &= \left(\frac{1}{T}\mathbf{W}_{ph}\right)\mathbf{h}^{(n)} + \left(\frac{1}{T}\right)\mathbf{b}_p\end{aligned}$$

Experimental results seem to contradict the above hypothesis, as samples drawn from models A through D after they have been trained to convergence seem qualitatively different. For increasing temperatures, the samples generated are different, but it is hard to judge the qualitative difference between them, since all generated sentences are reasonably well formatted and do not present obvious mistakes. Table 3 shows samples from each model conditioned on the seed characters "E", "T", and "C" respectively. We can observe that there are differences in the sampled sequences, so the networks does not simply scale the weights and biased. This indicates that the temperature parameter influences the learning dynamics to a significant enough extent that the network finds different pseudo-optimal solutions. The sequences are too short to observe very meaningful differences, but it seems that model A,B, and C have slightly better outputs, with model D being too reliant on newline characters, although they are not completely out of place. For example, they are not inserted in the middle of a word.

Bonus Question 3.2

We can inspect the qualitative differences between models better by sampling sequences longer than 30. In this experiment, I sample sequences of at least 210 characters based on a seed sentence, as opposed to a seed character. I then gradually increase the length of the sampled sequence until the model finishes its sentence, so the samples are not of equal length but they should be comparable. Below are some samples for different models and different seed sentences. What we can see is that the temperature parameter makes a difference only in some cases. For the seed sentence *The right of the people...* models A and B output exactly the same sentence, while for the seed *No person shall be...* models A, B, and C all output the same sentence. Model D is most interesting. I personally like the anarchistic view that it proposes: *No person shall be held to answer for a redress of grievances.*

Model A - *The right of the people* *peaceably to assemble, and to petition the Government for a redress of grievances.*

Model	Seed	Generated Sentence
Model A	E	**E**xcessive bail shall not be con
	T	**T**he right of the people peaceab
	C	**C**onstitution, nor prohibited by
Model B	E	**E**xcessive fines imposed, nor cr
	T	**T**he Ten Original Amendments to
	C	**C**onstitution of the United Stat
Model C	E	**E**xcessive bail shall not be req
	T	**T**he powers not delegated to the
	C	**C**onstitution, of certain rights
Model D	E	**E**xcessive bail shall not be inf
	T	**T**he United States Passed by Con
	C	**C**ongress September 25, 1789 Rat

Table 3: Samples for different model temperatures(A=0.5, B=1.0, C=1.5, D=2.0). *Different temperatures produce qualitatively different results, but because we are only sampling 30 characters it is difficult to see meaningful differences. One can still observe that models A,B, and C produce better looking sequences, as opposed to model D which seems to be overly fond of newline characters. Still it is impressive that all models output correctly typed words that are seem to be part of the same context consistently.*

II

A well-regulated militia, being necessary to the security of a free State, the right of the people peaceably to assemble, and to petition the Government for a redress of grievances. (271)

Model B - ***The right of the people*** *peaceably to assemble, and to petition the Government for a redress of grievances.*

II

A well-regulated militia, being necessary to the security of a free State, the right of the people peaceably to assemble, and to petition the Government for a redress of grievances. (271)

Model C - ***The right of the people*** *to keep and bear arms, shall not be violated, and no Warrants shall issue, but upon probable cause, supported by oath or affirmation, and particularly describing the place to be searched, and the persons or things to be seized.* (230)

Model D - ***The right of the people*** *to keep and bear arms, shall not be violated, and no Warrants shall issue, but upon probable cause, supported by oath or affirmation, and particularly describing the place to be searched, and the persons or things to be seized.*

V

No person shall be held to answer for a capital, or other (291)

Model A - ***No person shall be*** *held to answer for a capital, or otherwise infamous crime, unless on a presentment or indictment of a Grand Jury, except in cases arising in the land or naval forces, or in the Militia, when in actual service in time of War or public danger;* (242)

Model B - ***No person shall be*** *held to answer for a capital, or otherwise infamous crime, unless on a presentment or indictment of a Grand Jury, except in cases arising in the land or naval forces, or in the Militia, when in actual service in time of War or public danger;* (242)

Model C - ***No person shall be*** *held to answer for a capital, or otherwise infamous crime, unless on a presentment or indictment of a Grand Jury, except in cases arising in the land or naval forces, or in the Militia, when in actual service in time of War or public danger;* (242)

Model D - ***No person shall be*** *held to answer for a redress of grievances.*

II

A well-regulated militia, being necessary to the security of a free State, the right of the people.

X

The powers not delegated to the United States Bill of Rights. (216)

So, contrary to the previous experiment, it seems that for a stronger conditioning, the temperature parameter no longer has an impact for moderate values at least. Still, given that the models have been trained on a very small corpus, it is possible that the reduced impact of T can be due to the model overfitting the training set. Note that a different alternative as to consider the temperature as completely exogenous and not include it in the training procedure. We could then randomly sample from the multinomial distribution as described by the energies divided by the temperature and passed through a softmax transformation. This would result in the temperature parameter having a much higher impact since the network would not adapt to it but it is likely that this would lead to qualitatively poor samples in case of high T.