References Note: ChatGPT was used to help with some small coding details like custom color arrangements in the plots. I also read Yifan's code and implemented some adjustments from her code (tracking epoch loss, processing batches), but my batch processing implementation was done independently. I used documentation and StackOverflow for help with code and errors.

Code Analysis Questions

For most of the questions below, I used information from the blog post by the author of the RNN code, referenced below:

Karpathy, A. (2015, May 21). The Unreasonable Effectiveness of Recurrent Neural Networks. Andrej Karpathy Blog. https://karpathy.github.io/2015/05/21/rnn-effectiveness/

2.1 - Understanding the Model

a) What does each matrix (W_{xh}, W_{hh}, W_{hy}) represent?

Answer: As mentioned by Karpathy (2015), each matrix represents a matrix of learned (updated after each iteration of the training process) weights involved in the RNN.

The W_{xh} matrix has dimensions given by (hidden state size $\times$ number of unique characters). A single character is one-hot encoded, so the number of unique characters is the length of the vector representing a single character read into the model. The W_{xh} matrix is then used in the linear operation $W_{xh}x$ (which is only part of the sum of linear outputs used as input to the activation function).

The W_{hh} matrix is a square matrix with dimensions equal to the hidden state size. Within the forward pass, the hidden state from the previous iteration of the character generation is multiplied by W_{hh} (via the linear operation $W_{hh}h_{t-1}$, where h_{t-1} is the previous iteration's hidden state vector). The output of this operation is added with the output $W_{xh}x$ from the most recently-processed character, along with a bias term, and the tanh activation function is applied to this combination to get the new hidden state vector.

The W_{hy} matrix has dimensions (number of unique characters $\times$ hidden state vector size). It is a matrix of weights multiplied by the current hidden state output in a linear layer with bias ($W_{hy}h_t + b_y$), the output of which is fed into the softmax to predict the next character.

b) Why do we use the tanh activation function in the hidden state update?

Answer: The use of the tanh activation function applied to the sum of the hidden→hidden and input→hidden linear terms (plus the bias) allows for nonlinearities in the model. If no activation function was used, the RNN could only model linear relationships between the hidden state, input character, and output character, limiting its potential to understand more complex relationships and likely reducing its accuracy.

c) How is the hidden state initialized, and why?

Answer: The hidden state is initialized as a zero vector. This is because, when the model

begins training, it has no prior context (in the form of having seen previous characters that could provide useful information about the next character). Therefore, specifying a nonzero hidden state could introduce bias into the model (favor certain initial characters). To avoid this, the hidden state is set to a vector of zeros, making the product $W_{hh}h_0$ at the first iteration of the forward pass equal to zero. As a result, the output after the first iteration depends entirely on the linear output $W_{xh}x + b$.

2.2 - Training Process

a) What loss function is used, and why?

Answer: A categorical cross-entropy loss function is used. This is the appropriate loss function for the task of the RNN during the training process, which is correctly classifying the next character in a string. There are a fixed number of unique characters in the training dataset, and the RNN outputs a vector of probabilities that the next character is each of these unique characters (the softmax output). Therefore, the problem is a classification one – the RNN is attempting to identify which of the classes (unique characters) the next character belongs to. Therefore, the categorical cross-entropy loss is most appropriate (the explanation of this – based on the fact that the categorical cross-entropy loss is derived from MLE applied to the softmax output – was noted in HW1) (Prince, 2024).

b) How does the model update its weights?

Once the model computes the gradients via backpropagation, it uses what is referred to in the comments of the code (Karpathy, 2015, 2016) as AdaGrad. As implemented, the derivatives of each parameter value are multiplied by the learning rate as usual. However, this value is then divided by *mem*, which is the cumulative sum of the squared changes of the parameter from all previous training iterations (plus a very small constant). Therefore, the parameters in later iterations are changed at a relatively slower rate compared to the parameters in earlier iterations, and the rate of parameter change as a function of iteration is lower when the parameters were changed substantially in past iterations.

c) What is the purpose of gradient clipping (np.clip)?

The gradient clipping process is intended to "mitigate exploding gradients" as noted by Karpathy in the comments of the RNN code (code linked in Karpathy (2015, 2016)). Effectively, this means that the process is intended to to avoid gradients that blow up in the model and cause numerical issues. If very large gradients are produced during one iteration of backpropagation, resulting weights would be significantly adjusted. This could potentially move some weights relatively far from an optimal value, resulting in more large gradients and more significant adjustments to weights. This process could repeat, causing larger and larger fluctuations away from the optimum until numerical issues are encountered. With gradient clipping, these issues are mitigated because gradients are not permitted to exceed a certain absolute value. This could limit the speed of model training in some cases, as more iterations may be necessary to get close to optimal weights, but it increases the chance that the model successfully trains in the first place.

2.3 - Text Generation

a) How does the model generate text?

Answer: Note: This text generation process is done in the sampling function, so my answer to this question and part (b) are similar. The model generates the next character by taking the output of the forward pass activation function output (the tanh output) and running it through the linear layer with bias using the weights W_{hy}. Then, a softmax function is applied to the vector of outputs of this layer to turn them into probabilities of selection for the next character. A random character (actually a character index) is chosen with weights of selection for each character index equal to the probabilities from the softmax output (so the higher probabilities correspond to higher chances of a certain character's index being selected). This index is selected and translated back into its corresponding character value to generate the next character. Then, the model continues running using the updated hidden state and most recently generated character to generate the next character through the forward pass / sampling process, and so on.

b) How does sampling work in the sample function?

Answer: The sampling function begins by selecting a character specified by the seed_idx parameter. This is treated as the first character in the string. The sampling function takes in the last value of the hidden state of the model (output by the model training loop) and uses this hidden state, combined with the specified character value, to run the forward pass of the RNN. The weights used during the forward pass are not specified as input into the function or within the function, so they appear to be the weights as of the last update in the training loop. A weighted draw is then taken from the softmax logit output of the forward pass, and the resulting integer selected identifies the index of the next character generated. This is then fed back into the RNN along with the updated hidden state to generate the next character, and so on, until n iterations are reached.

c) What happens if you modify the sampling temperature?

Answer: As noted in Sharma (2022), the temperature is a parameter in the softmax function, represented by T in the softmax expression:

$$s_i = \frac{e^{\frac{x_i}{T}}}{\sum_j e^{\frac{x_j}{T}}}$$

where s_i is the ith entry of the softmax output vector. Here, if $T = 1$, the softmax function behaves as usual. For very large T, the values in the numerator become closer to each other for different values of i, since the exponent is smaller (and thus $e^{\frac{x_i}{T}}$ changes relatively little in response to a change in x_i) (Sharma, 2022). Therefore, the softmax output vector (containing probabilities of selecting each character next) has entries that are more similar, resulting in a wider variety of characters selected, but less consistency, as noted by Karpathy (2015). For very low temperatures, the exponents are amplified and differences between them become more significant for the same differences in x_i (Sharma, 2022). Therefore, the model is highly likely to select the next letter that it has the most confidence is correct (even if this confidence is only slightly higher than other options). It tends to produce frequently-occurring output or patterns from the source text but does not produce more complex or creative strings, as Karpathy (2015) notes.

Modifications and Experiments

Note: the code files for these questions are on my Github repository, beginning with "rnn_" or "lstm_" and including the question number. Code to create the plots is very simple (just a few lines to load the text files and plot the data) and is not included in the GitHub repo.

#1 **Answer:**

We first discuss what dropout is. Dropout zeros out a certain proportion of the neurons in the neural network during each iteration of the neural network's training process ("Dropout", n.d.; Yanav, 2022). This results in different sets of neurons being used to predict the model output for different training iterations (Yanav, 2022). Therefore, as Yanav (2022) mentions, dropout helps the model generalize, as it prevents the reinforcement of very specific relationships between the activation of a neuron or set of neurons and the input data. If these relationships were allowed to occur, the model could potentially overfit the data, predicting the training set well but generalizing poorly to the testing data (Yanav, 2022). By requiring that different sets of neurons predict the output each iteration, dropout helps the model learn more broad patterns between the input data and its labels (Yanav, 2022). High dropout (encouraging more generality) could decrease training accuracy but potentially increase testing accuracy if the model overfits the training data with low or no dropout. Reducing the dropout rate allows the persistence of more specific neuron-input data relationships which could overfit but may also better represent meaningful, useful relationships - which one of these is the case depends on the training and testing accuracy for a specific model, and whether overfitting (training accuracy >> testing accuracy) is occurring (Yanav, 2022).

The provided RNN (in the *rnn_w_cuda.py* file on Canvas) was modified first to use a batch size of 64, a modification found in Yifan's code which significantly improved training times and allowed for feasible model runtimes in the first place (without this adjustment, it would take a long time to train the model even once for a few hundred epochs). Multiple adjustments to the code were made to allow for compatibility with training with batches. Since batches are used, the hidden layer was reset to zero after every batch (using the hidden layer from the previous batch as done in the LSTM code seems reasonable for a batch size of 1, but I wasn't sure about larger batch sizes, so I did not implement this in either the RNN or LSTM code).

A variable *dropout_rate* parameter (as in the provided LSTM code) was added as an argument to the function to initialize the RNN in the code. Three values of dropout rate were tested: 0, 0.3, and 0.8. Some other parameters were kept at their default values provided in the code from Canvas (sequence length of 25, hidden layer size of 100 neurons), but the learning rate was adjusted to 0.01 (0.1 resulted in no clear convergence after 40 epochs; 0.01 reduced the loss but loss still began to increase then stabilize after a few epochs, interestingly) and a second hidden layer was added to allow the dropout to be used (since dropout cannot be applied to a single hidden layer, as noted by the PyTorch documentation ("RNN", n.d.). For each value of the dropout rate, the model was run for 50 epochs and the training loss as a function of epoch was tracked. Here, the goal is for the model to learn to write like the input text (here, Shakespeare) as well as possible. The training loss is therefore probably a decent metric of model performance (since the input dataset is so large, this somewhat enforces that generalization – and therefore a true ability to 'write like Shakespeare' – is necessary to achieve low training loss in the first place).

We also store the sample text from the model every 10 epochs. Results are discussed below.

A plot of the training loss as a function of epoch for the RNN run with the three different dropout rates is included below.

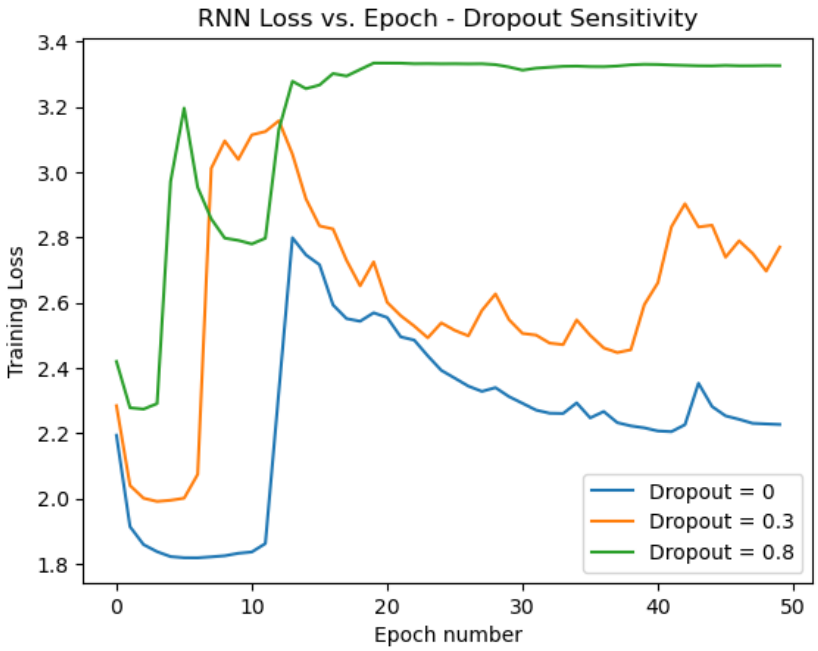

Figure 1: Training loss per epoch for RNN with different dropout rates.

Interestingly, we see that the loss drops in the first couple epochs, then increases a few epochs later in all cases, before either stabilizing or decreasing. The minimum loss over all 50 epochs is achieved in the first few epochs for all dropout values. (Note that, as tested in the next question, this may be due to the nonlinearity used - here, tanh). Nevertheless, we see that the lowest dropout rate (0 percent) results in the smallest minimum training loss (and, in fact, the smallest of the three training losses for all 50 epochs. The moderate (30 percent) dropout rate results in a somewhat higher minimum training loss and a more gradual and inconsistent decrease in training loss with epoch for later epochs. The high (80 percent) dropout rate results in training loss that quickly increases then stabilizes. Despite the unexpected increases in training loss we see, the *relative* results are broadly consistent with expectations. Lower dropout rates should result in lower training losses (since the model is more easily able to model more complex relationships in the data). Further, as increased dropout causes better model generalization, this occurs at the expense of higher training loss, as indicated by the high-dropout curve's higher training loss values.

Output from the models with different dropout rates is presented below:

Dropout rate (epoch)	Sample text (line breaks not maintained)
0.0 (1)	He wit, ry toul atty pors? HESANZO: Nis conchathy wirn boy worge amenes An in The sound repentines GENZALO: Seld: a bose ough could, aridenairs raress fit, is dost and feaquito conchover but here swe
0.0 (41)	Ha: O Toldt thou you who, Nutulfoda, afy thelad, ot, mece and on I pass thou of As how tad fososow Walchondy wabemuede Binded. GAodON. KAONNIONONSMENONASIZOLAMANSTABOSTANNANANTIONONONSONNONONZOLATTAl
0.3 (1)	Hountring opour my, and ation good of afly wis fion uppheard of that! ANASASIS:OLO: Huchirabe, ssour. GONZSON: And I befediost grudconer, a swell swamf no? Zigelire I rout fauls Wakr Some! Bule in fe
0.3 (41)	H On coptt this, Noudedce mpiv seiwe: IAANaaecu farl-alis onse ofoh iofe: Nae? sris tialgt Sotimod thes der bege TA: Ane woly sintutch nareaot thin lsrd ispocs toyowt, ANONNES:NONNIAZEOAZANOS: LOA
0.8 (1)	Henc will cidued. MIONANAN: Os mow mastior and, as mesmancistron o, nond ANTONZONANZANISANONASZANOSINANALSONANZANONAN: SANdel hy eead? PANONZONONZINANZONZONLONZINANZISONANZAMOSANANO: And of noug urh
0.8 (41)	Hontsi EnisLOhx wNo o,orNak I aSA:SZehlt epcO T u sZaoOooa tNFeZtN o yiSoOTd Aos Lc Dt icduO ,e ApSnriN rWelomn, aw udyt N tt AansNeuu oelmasZ, dnsa z:Io .n bB on,TLIZdrfENm itOtL e s oef

Table 1: Samples of text from different epochs of LSTMs trained on different datasets.

The samples from this table indicate (as suggested by the loss evolution) that the samples early in the model's training (even after epoch 1) are more reasonable than those at later epochs. Specifically, we notice long strings of capital letters in the results from later epochs for the 0.0 and 0.3 dropout rates, which do not exist in the epoch-0 results. For the dropout = 0.8 case, we have these long strings of capital letters in the epoch 1 results, and nonsensical arrangements of characters by epoch 41 (corresponding with the high training loss). These results suggest that the model performs better when dropout is low, since the high dropout rate seems to prevent the model from learning as much meaningful information about the input data (although some of this is also likely due to the activation function which appears to cause the unusual loss behavior).

#2 **Answer:**

For the RNN, we use the code described in the answer to question 1, except we fix the dropout at 0 and change the nonlinearity (activation function) within the RNN between tanh and relu. Note that sigmoid is not an allowable argument for the nonlinearity parameter inside the RNN function (the PyTorch documentation only offers tanh and relu), so

sigmoid activation functions were not tested. The RNN was trained twice (once for each activation function) for 50 epochs, as in question 1. All other parameters are equal to those specified in my answer to question 1.

For the LSTM, the nonlinearity was not an adjustable parameter within the nn.LSTM module, so I wasn't sure how to change it. The RNN results are discussed below.

A plot of the RNN loss with respect to the nonlinearity used is included below.

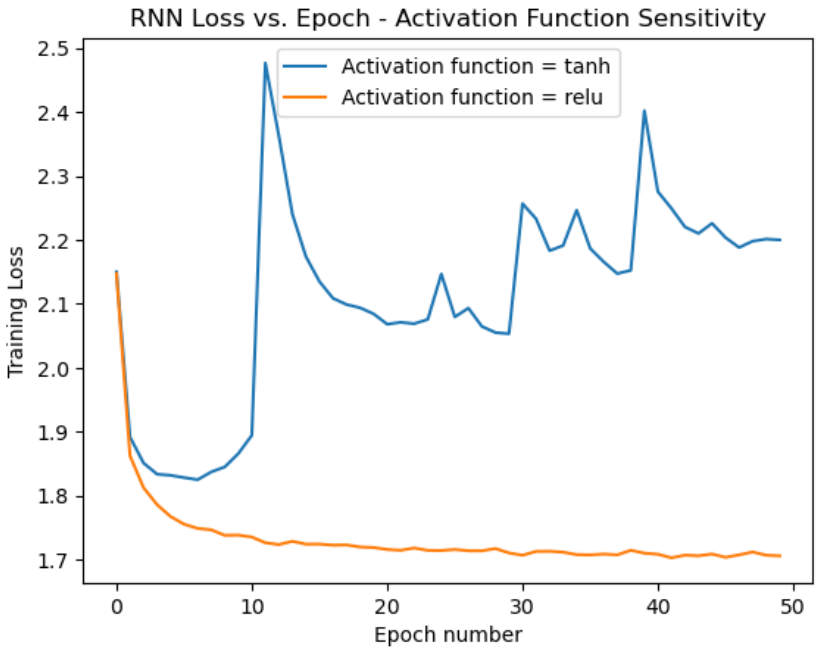

Figure 2: Training loss per epoch for RNN with different activation functions.

From the above plot, we see that the training loss is lowest in the first few epochs, then behaves somewhat erratically for the tanh activation function (as seen in the previous question, since the default tanh activation function was used in the previous question). However, when we modify the activation function to relu, then the training loss decreases nearly monotonically as a function of epoch, in a much more typical sense. The training loss has a lower minimum value and remains lower for all epochs with the relu activation function when compared to the tanh activation function.

Sample text output from the two configurations is included below:

Activ. Func. (epoch)	Sample text (line breaks not maintained)
tanh (1)	HALOOZIZONZOEBLO:ZOS: Bup forlell; Tolbling a ame playe To re percince Thou witinl. NINZANISe THANAN: ar quarkent, so powackmer, Iblicio? GONZNLO: Highards sil Of you! the very ever, the see well lai
tanh (41)	HA: O noty thang me. OZABOSIO: ZZANBORO: Chame. Nold natition. Fate in ttonst droto a, cONS. Thy nan wO:: Oing thing sere to is sar moder sparnyaang of yea augo clore Goned moor, of lione thoustring s
relu (1)	HOr So whith for sex A in'n it knose if bely and my jick evand I here of we's tooseron? to pay of eply naughing of spland Gly he make in. What so Bate, Wubisest Too thest maide, bu he, he tin cord: A
relu (41)	Have may cannoted no duying. ISABELLO: I aming. Will your sighrert I, bave wire; these dother an cears, we it good take, you ever atterb Ever thep'st behast that thou prince fing; In sain losp the wa

Table 2: Samples of text from different epochs of LSTMs trained on different datasets.

With the tanh activation function, we see relatively little change in the style of the output between epoch 1 and 41, although the long string of capital letters (speaker name) in the first epoch does not remain in the 41st. Otherwise, the word lengths are reasonable but most strings are not actual english words in both cases. Punctuation is somewhat reasonable, with question marks and exclamation points placed in typical locations in the epoch 1 results. With the relu activation function, we see again relatively similar output between the earlier and later epochs in a qualitative sense (despite the fact that the loss decreases quite a bit). Some more four and five-letter words are spelled correctly in the later epoch (will, your, thou, prince, etc.). The later epoch shows the typical capital letter-colon format for the speaker name, which is not included in the earlier epoch's sample.

#3 **Answer:**

To compare the impact of changing to an LSTM from an RNN, we run the RNN and LSTM with identical parameters. The RNN was run with all parameters specified in question 1 (2 hidden layers, 100 nodes per hidden layer, 0.01 learning rate, batch size of 64, 50 epochs, sequence length of 25, default tanh activation function), and the dropout rate was fixed at zero. The LSTM was run with the same parameters (except no activation function, since it is not an available argument in the PyTorch LSTM initialization function). The *lstm_w_cuda.py* code provided on Canvas was used to run the LSTM (with a modification to reset the hidden layer neurons to zero every batch), and the training loss and output every 10 epochs was recorded as usual. Results are discussed below.

The plot below shows the impact of changing the model from an RNN to an LSTM on the training loss.

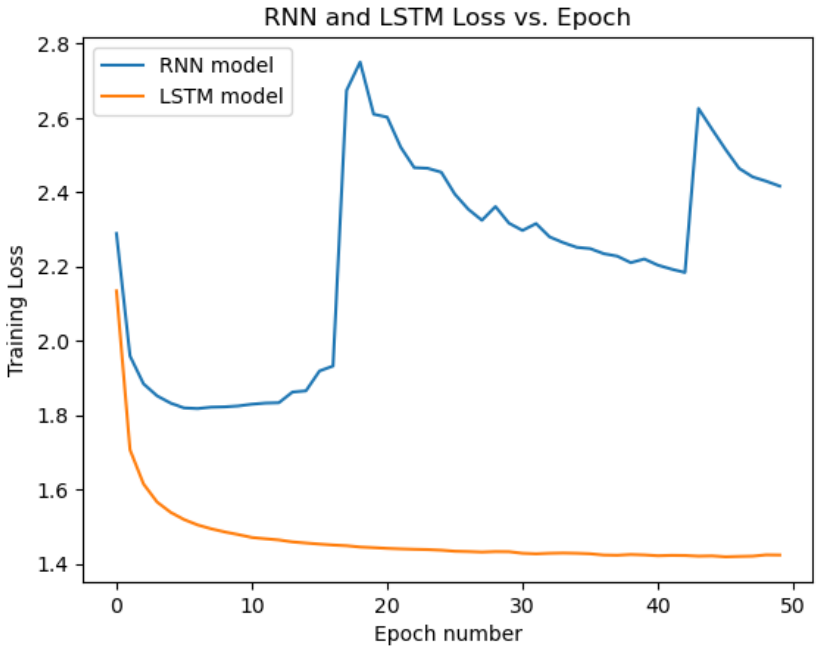

Figure 3: Training loss as a function of epoch for RNN and LSTM

The above figure shows that, as expected, the RNN with default parameters (including tanh activation function) follows a similar loss-vs-epoch behavior as in the previous question. However, when we change to an LSTM, the training loss once again decreases nearly monotonically as a function of epoch and reaches much lower values. Comparing the LSTM training loss to the RNN with relu activation function training loss from the previous question (both of which behave in a similar way), we see that the LSTM loss levels off at lower values (1.4) compared to the RNN with relu (1.7), so the LSTM is the best-performing model/configuration out of these three.

We compare the sampled text from the RNN and LSTM in the following table.

Model (epoch)	Sample text (line breaks not maintained)
RNN (1)	Hadoury, lir, To ghadtafnise! am hang, sore whace to mootinelt to Thou'le? murss that lorushle see is chargeed alvest as efts is seatens omsy yon, Apravon-ntloNe, woll. SoNith ness sumy roghat'. ASNON
RNN (41)	He norcendeONONINANZS: To seor shis Sad? O: Ource Of a fros sy in awanet do apt-oublest and my I quanf you sraom of Sneatreale buke saur Sounslaied amkene hind di: No un sake the hagck suve ters, net,
LSTM (1)	HIONBONAN: But of call our wighte rone Tue sufss, may; exch; We bud The fease. o Nail o! I do carline not sevet, less sore how bears, pariour take a moase. MANTANO: Boing's rower: Wost, my exemast; a
LSTM (41)	Here rough Shall abunders is the harmied of passion or houses brave paise to thee free teach enemy to Angeland but welcome; Who are And roses' my midding, And bid son or under of the gorromall A, in st

Table 3: Samples of text from different epochs of RNN and LSTM.

We notice that, as expected given the behavior of the training loss as a function of epoch, the RNN does not show substantial improvement as a function of epoch. For the RNN results, the model after the first epoch shows an understanding of word lengths, but strings are not comprehensible english words. Similarly, strings remain incomprehensible for the 41st epoch of the RNN results, although we see some understanding of the capital letter-colon pattern. The LSTM shows an understanding of the capital letter-colon pattern in the epoch 1 results and writes a few more english words. However, by the 41st epoch, the LSTM results are substantially improved, with the model writing longer words spelled correctly (passion, shall, enemy, welcome, etc.). This corresponds to the substantial decrease in LSTM training loss later in the training period (and relative to the RNN).

#4 **Answer:**

Three values of the hidden layer size, sequence length, and learning rate were used in the hyperparameter tuning experiment. We consider hidden layer sizes of 25, 50, and 100 neurons (large hidden layer sizes required long training times, so I only reduced these from the default of 100). Sequence lengths of 10, 25, and 100 are considered (with the default being 25). Lastly, learning rates of 0.001, 0.01, and 0.1 are considered (around the default of 0.01). The LSTM was run for the hyperparameter tuning (given its generally better performance). All models were run for 50 epochs each. Results of the hyperparameter tuning are discussed below.

A plot of all 27 hyperparameter tuning runs is included below. Subplots are organized by hidden layer size.

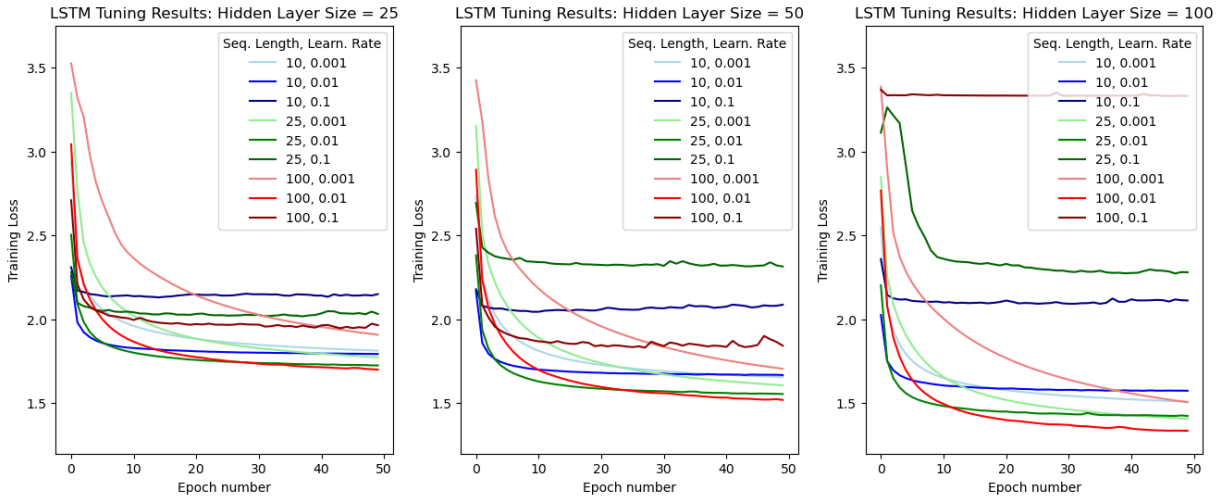

Figure 4: Plot of training loss as a function of epoch for all hyperparameter combinations tested on the LSTM

The above results show that, as the hidden layer size increases, the values of the training loss at which the models level out tends to be lower for most (except the high learning rate) configurations. This is understandable, since a larger hidden layer allows for more details to be picked up by the model. The model configurations with high learning rates (dark colors) tend to drop then quickly stabilize in training loss as a function of epoch, and among these, the model with the highest sequence length shows the lowest training loss for the smaller hidden layer sizes, and vice versa for the largest hidden layer size. The training losses of the models with high learning rates for later epochs is consistently relatively high as expected, since the models cannot hone in on a minimum in the loss as easily. The models with with lower learning rates decrease in loss more gradually, especially those with high sequence lengths. It appears that high sequence lengths and low learning rates contribute to more gradual changes in loss over time. Low sequence lengths mean that more training iterations (batches) are included in each epoch, so their faster drop in training loss is not unexpected.

We next analyze the sampled text output of the models. We consider only results from the 41st epoch (the last one for which output was stored), and consider high and low valules of each parameter with the others held at their intermediate values.

Param. Set (epoch)	Sample text (line breaks not maintained)
HL25, SL25, LR0.01 (41)	HORTILO: Preving As I pleer. Of but chufe: to-mill in of under son. Sigicty you and is lace; their his eagor 'two semial not ounce, sises they. O leing accaw well, that papeech le! what lefor which lik
HL100, SL25, LR0.01 (41)	HIO: 'Tis no more pleased that o'clow To be the meal your discries, prayer Musty worthy lovely– Were their wife, else as the dalis'd you hithes them at milds' Afommade repilias o' the open in such at
HL50, SL10, LR0.01 (41)	How father as shall leat and: as it so before in hare. DORCANA: Well: She thing To's feare eag me say man wash, like, Take agy down touner'is affo. ADRIAN: Well meant Serve thou be, Or Marthom; and s
HL50, SL100, LR0.01 (41)	Hasted grike the soldise; Brendors! Lough out us To'troy my constant at what should we slow's brother treal feen fortulous befter Bocoful other heavored me that Aician: Nearens.–Emise, you hath then b
HL50, SL25, LR0.001 (41)	HIO: I am on the warire: Mire stap. Thire think. Their acqueel she things againck'd the spire, Sopt fople of York. GRUMIO: I lovement: For the stuned for harch May too brothers the graccious to the s
HL50, SL25, LR0.1 (41)	Heaow thou ce, I wory nowte thy. Sardarltdichatte afterslartry nelliks you uble, Ans to drraimed; her. Notded asiy or dulinly ciu mave is the eper, hrorqul wood thy Thetler, VE: Af. het woul'd seill

Table 4: Samples of text from different epochs of RNN and LSTM. HL means hidden layer size, SL means sequence length, LR means learning rate.

From the first section of the table (adjusting the hidden layer size), we see that a larger hidden layer size with other parameters held at their intermediate values produces output that is somewhat better. In the larger hidden layer size run, we see more less-common words spelled correctly (musty, worthy, lovely, pleased, prayer) which we see only for perhaps more common words such as their, which, under for the smaller hidden layer size run. The HL100 run also even includes an apostrophe in 'tis, correct punctuation for the word (although the punctuation for the HL25 sample is still decent as well, especially in capitalization after periods). Moving to the next section of the table (adjusting the sequence length), we see that both samples show decent punctuation and spelling for some words (father, shall, think, serve in the SL10 case; constant, brother, should in the SL100 case), though the SL100 sample spells some longer words correctly. Lastly, comparing the LR0.001 and LR0.1 sample, we see that the LR0.001 sample has more reasonable word lengths, better punctuation, and spells some words correctly (think, things, brothers). The LR0.1 sample has some unreasonably long words and does not produce interpretable words for the most part (very few are spelled correctly), correlating with its higher training loss.

#5 **Answer:**

We run the LSTM with its default parameters for this problem (as in question 3). Due to the similarity between this code and the code from question 3, this code is not uploaded to Github, since the only differences are the input and output file names and adjustment

to the numbers of epochs. We test two different sets of training text: (1) my senior thesis about climate model perturbed parameter ensembles, a relatively small training set and (2) Darwin's *On the Origin of Species*, from Project Gutenberg, a larger (in a relative sense) training set. Since the senior thesis is a smaller training dataset, 500 epochs were run because training times are much faster. 100 epochs were run for Darwin's training dataset. Results are discussed below.

The plot below compares the training losses as a function of epoch for the LSTM trained on the two input datasets.

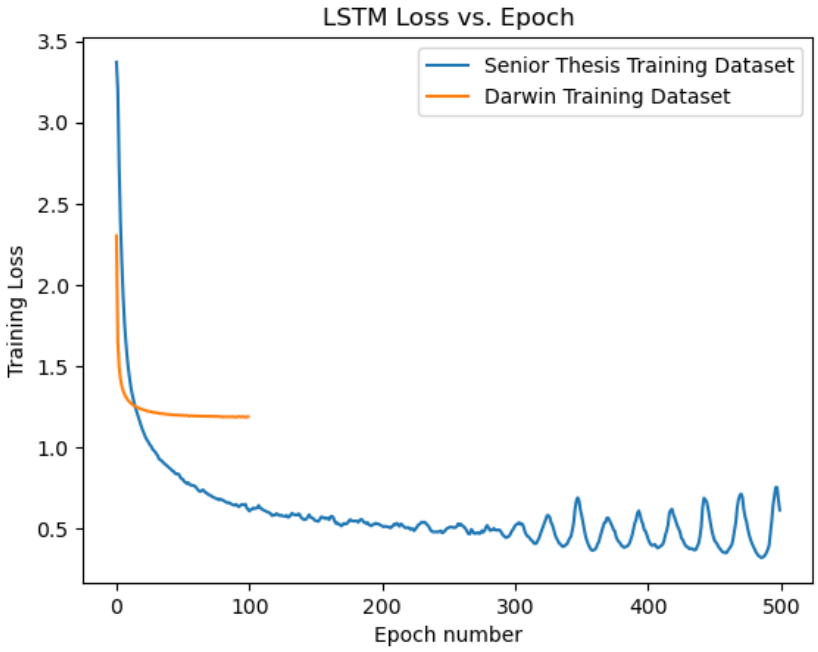

Figure 5: LSTM training loss as a function of epoch for the LSTM trained on Darwin's *On the Origin of Species* and my senior thesis. Darwin LSTM was run for 100 epochs, senior thesis LSTM was run for 500 epochs.

The training loss decreases quickly as a function of epoch for both LSTMs, and levels off at a higher loss for the LSTM trained on Darwin's work (perhaps because the sample dataset is larger which enforces more model generality). The loss decreases to a lower value for the LSTM trained on my senior thesis text, and then begins to oscillate after a few hundred epochs. The loss curve for the senior thesis-trained LSTM, however, is less sharp, with more of a gradual decrease in loss from its highest to lowest values, as opposed to a sharp initial drop and then leveling off in the Darwin-trained LSTM. We next compare the text generated by the two LSTMs.

LSTM (epoch)	Sample text (Line breaks removed)
Darwin (1)	Hequifuct of the mand as falal and carantions worghe be park mode of undicalicat, the mogroughu,—apouic as in long the specaly anlugination, what ands of the beanly unchise tume difted and wively dedec
Darwin (91)	How breeds of life. The chapter; they we to discover invation are little varieties of a paaracier will succeeded time to the creation, but the contatiin 1649 throughout naturalists and yet advinuating
Thesis (1)	Heh.e3i. nLR(tot,)i ner oeTpln .e.deidan1,.,(T img8titte ml o52(oU(nPRg((6e Cl a,G0aeilosbt ledlosn rmrd2k d Lton-rPa asGo eid tl)F , ath er , l0liend f yt . U,ge;l f(ese u,ec, bMm ea0t
Thesis (251)	However, ULR-SR distances are notably different means, agreen consistently limit such cloud, if Reselence of climate feedback above (nogs, otheching the ULR PPE, respect distri- centiles are much that

Table 5: Samples of text from different epochs of LSTMs trained on different datasets.

From the above table, we see notable improvements in the model between the first and middle-to-last epochs as seen in the training loss plot. The Darwin LSTM produces relatively well-formatted text even after the first epoch compared to the thesis LSTM which does not, but this is likely because there are many more training batches in the Darwin training dataset than the thesis training dataset. Both LSTMs produce text that seems consistent with the input data source, mostly producing recognizable words in both cases. In the senior thesis case, the LSTM picks up on the use of the ULR and SR acronyms (which I used to represent the ultra-low resolution and standard-resolution climate model ensembles in my writing), as well as terms such as climate feedbacks and phrases common in my (scientific-style) writing such as 'notably different', 'agree consistently', etc. The Darwin LSTM terminates the number in its epoch-90 output (1649) at an expected number of digits to represent a year, includes likely common words in the input dataset such as 'naturalists' and 'discover', and even features like a comma before the conjunction 'but'.

#6 **Answer:**

I read a paper about the use of LSTMs in developing a deep learning model for tropical cyclone rapid intensification forecasting. The authors (Yang et al., 2020) fed information about both the tropical cyclone (motion, derivatives of intensity, basin) and the *monthly-mean* reanalyzed environment (shear, ocean temperatures, etc.) into a single-hidden-layer LSTM, with updated input data every 6 hours. Their LSTM was run with data from eight timesteps before the start of the prediction window, and labels indicated whether rapid intensification occurred within a 24-hour window after the end of the time series fed into the LSTM. Yang et al. (2020) then applied a single linear layer with one neuron and sigmoid activation function to the final LSTM hidden layer, producing a rapid intensification probability. Yang et al. (2020) found that the model performed relatively well when compared to past rapid intensification prediction efforts (although differences exist in the testing sets between these older models and this paper, as the authors note), and found that for lower thresholds of rapid intensification, the model significantly outperforms the SHIPS rapid intensification guidance. The authors used an LSTM in this research since they recognized that information about the past evolution of the tropical system could be useful in understanding the potential for rapid intensification (Yang et al., 2020, p. 1205).

In a simple fully-connected neural network, for example, the model would not be able to capture evolution of the system (unless provided explicitly in the input data). Yang et al. (2020) note that this model holds potential for better rapid intensification predictions, and mention some potential for further improvement (by using environmental data on hourly instead of monthly timescales for input). Further, Yang et al. (2020) provide initial analysis of the important features in model predictions by comparing the feature values in predicted rapid intensification vs. predicted no rapid intensification cases, providing initial insight into trends and values that the model is most sensitive to.

Reference: Yang, Q., Lee, C., & Tippett, M. K. (2020). A Long Short-Term Memory Model for Global Rapid Intensification Prediction. Weather and Forecasting, 35(4), 1203-1220. https://doi.org/10.1175/WAF-D-19-0199.1

References

Prince, S. (2024). Understanding Deep Learning. The MIT Press. http://udlbook.com

Karpathy, A. (2015, May 21). The Unreasonable Effectiveness of Recurrent Neural Networks. Andrej Karpathy Blog. https://karpathy.github.io/2015/05/21/rnn-effectiveness/

Sharma, H. (2022, July 15). Softmax Temperature. Medium. https://medium.com/@harshit158/softmax-temperature-5492e4007f71

Dropout—PyTorch 2.7 documentation. (n.d.). PyTorch. Retrieved April 27, 2025, from https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html

Yadav, H. (2022, July 5). Dropout in Neural Networks. Towards Data Science. https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9/

RNN — PyTorch 2.7 documentation. (n.d.). PyTorch Documentation. Retrieved May 3, 2025, from https://pytorch.org/docs/stable/generated/torch.nn.RNN.html

Karpathy, A. (2016, May 15). Minimal character-level language model with a Vanilla Recurrent Neural Network, in Python/numpy. GitHub Gist. https://gist.github.com/karpathy/d4dee566867f8291f086

Darwin, C. (1998). On the Origin of Species by Means of Natural Selection. Project Gutenberg. https://www.gutenberg.org/ebooks/1228 (Original work published 1859)