CSE 151B Deep Learning

Assignment 2 Part 2

Summer Session 1 2026

Instructions

Due Wednesday, July 15^{th}, 2026 11:59 pm:

1. **Start early!** If you have any questions or uncertainties about the assignment instructions, please ask about them as soon as possible (preferably on Piazza, so everyone can benefit). We want to minimize any possible confusion about this assignment and have tried very hard to make it understandable and easy for you to follow.

2. You will be using PyTorch for this assignment. You should already know how to access the UCSD GPU server by this point but if you've forgotten, please see the material on Piazza to refresh your memory. Additionally, any updates or modifications to the assignment will be announced on Piazza.

3. **Note: Training this model will take you about 20-30 minutes**, and may take longer if the GPUs on Datahub aren't free for usage. Any hyperparameter or architecture change that you make will take an hour before you figure out whether that change was correct or incorrect. **So, start early!**

Learning Objectives

1. Understand the basics of a recurrent neural network (LSTM).

Part I
Understanding LSTMs

1. This question refers to Figure 1. Match each RNN architecture to the domain or problem it is best suited for. Each option is used *at most once*; one option is not used. Write the letter in the blank.

Architecture

1. ___ Figure 1b: many to one
2. ___ Figure 1c: one to many
3. ___ Figure 1d: many in, then many out
4. ___ Figure 1e: simultaneous many in, many out

Domain / Problem

A. Machine translation: a sentence in one language is consumed in full, then a sentence in another language is emitted.

B. Part-of-speech tagging: a sentence is consumed and each word receives a tag as it is read.

C. Sentiment analysis: a sequence of words is consumed and a single label describing the emotion of the text is emitted.

D. Image captioning: a single image is consumed and a sequence of words is emitted.

E. Image classification: a single image is consumed and a single class label is emitted.

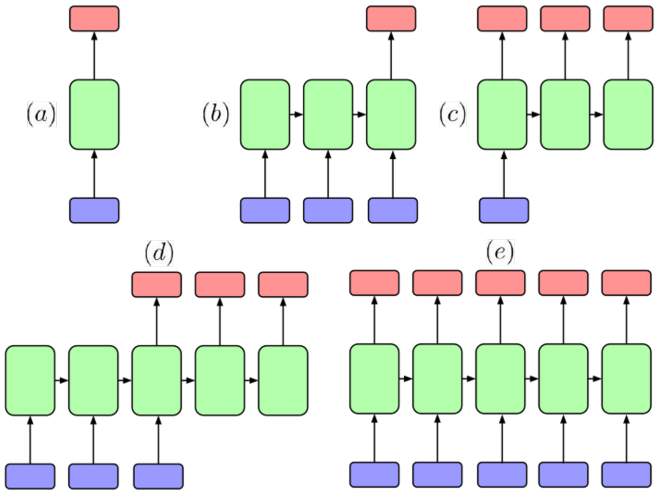

Figure 1: Basic Recurrent Network Architectures (except (a), which is feedforward!)

2. The LSTM cell update is given below, where x_t is the input at time t, h_{t-1} is the previous hidden state, c_t is the memory cell, and $\odot$ is elementwise multiplication.

$$\square_1 = \sigma(W_1[h_{t-1}, x_t] + b_1) \qquad \square_2 = \sigma(W_2[h_{t-1}, x_t] + b_2)$$
$$\square_3 = \sigma(W_3[h_{t-1}, x_t] + b_3) \qquad \square_4 = \tanh(W_4[h_{t-1}, x_t] + b_4)$$
$$c_t = \square_3 \odot c_{t-1} + \square_1 \odot \square_4 \qquad h_t = \square_2 \odot \tanh(c_t)$$

(a) Each of $\square_1$–$\square_4$ is one of the four LSTM gates. Identify each.

$\square_1$: _____

$\square_2$: _____

$\square_3$: _____

$\square_4$: _____

Options:
input gate (i)
forget gate (f)
output gate (o)
input modulation gate (g)

(b) Fill in each blank. Use each option *at most once*; not every option is used.

Options: i (input gate) · f (forget gate) · o (output gate) · g (input modulation gate) · r (reset gate) · $(0, 1)$ · $(-1, 1)$ · c_{t-1} · h_{t-1} · tanh · σ · discarded · retained · overwritten

The cell is updated as

$$c_t \;=\; f \odot \_\_\_\_ \;+\; \_\_\_\_ \odot g,$$

so c_t has two additive contributions: the old contents, which are _____ when the corresponding value of f is near 0; and a new candidate value, which is admitted only when the corresponding gate value of ____ is near 1.

The hidden state is

$$h_t \;=\; o \odot \_\_\_\_\_(c_t).$$

Setting o to 0 means that c_t is _____ inside the cell but is not exposed to the next layer or to timestep $t + 1$'s input.

Character level LSTM for Shakespearean text generation

In this assignment we will explore the power of Recurrent Neural Networks. In this problem, we will generate Shakespearean text. You will train a basic RNN/LSTM model using text extracted from some of Shakespeare's works provided to you and then run the network in generative mode to write new text imitating Shakespeare's style. Please familiarize yourself with Pytorch before implementing. One of the good places to start with Pytorch is this playlist.

Problem: For this assignment, you will be using the TinyShakespeare Dataset. William Shakespeare wrote a total 38 (disputed) plays in his life combining for nearly 120,000 lines. TinyShakespeare is a collection of 40,000 lines from a mix of his plays. Using LSTMs, you will generate text in the literary style of William Shakespeare.

1. **Getting familiar with the data** In this part, we are going to take a look at how the data is stored.

   You will be generating text in a similar format as the data sample.

2. **Read in data**. Read in the data from the tinyshakespeare.txt (found under the **data folder** in starter code) file. This file contains multiple short text sequences of various plays written by Shakespeare.

3. **Train a network** First, train a model to learn the structure of Shakespearean text through prediction. Your network will take in a particular sequence of the training set, and given the first character as input, you will train it to predict the second character, etc.

   You should use the available Pytorch modules. Try using around 100 neurons in your hidden layer. You should use the cross entropy loss.

   In the training stage, the network takes the ground-truth character of current step as input and predicts the next character. Then the next character (ground truth) is used as input, and it is trained to produce the subsequent character. This is sometimes called "teacher forcing." Teacher forcing works by using the teaching signal from the training dataset at the current time step, $target(t)$, as input in the next time step $x(t+1) = target(t)$, rather than the output $y(t)$ generated by the network. The training and validation dataset is provided to you. Evaluate your model on the training and validation set after every epoch. Include the training and validation loss curves in your report. A decent baseline should give you roughly **1.4** cross-entropy loss on the test data after 10 epochs.

   What you must implement:

   Sections to be completed by you are marked with **TODO**. You must write the implementations for the vanilla RNN and LSTM. The files are designated for you in **shakespeare_rnn.py** and **shakespeare_lstm.py** respectively. Use the **yaml** files (and add more as you need) in the **configs** folder to modify or add hyperparameters and other settings to your model. One such config file is already there for you as an example.

   You must implement the **ShakespeareDataset** class which is in the file **shakespeare_dataset.py**. You must fully write the **train** and **eval** functions located in **train.py**. The code to generate text is in **generate.py** and the notebook for you to run it is **generate.ipynb**. The data encoding, the sequence creation, and the train/val/test has been done for you in **main.py**. **Dataset** and **DataLoader** objects have also been appropriately created for you.

4. **Generate text** In the generation stage, the idea is to "prime" the network with a sequence and then let the network run on its own, predicting the next character, and then using the network's output as the next input. There are at least two ways you could feed back the network output. One is to take the maximum output. We don't recommend this option. A second way is to flip an n-sided coin, assuming n outputs, based on the probability distribution at the softmax layer. You can make the network more or less deterministic by adjusting the temperature (T) parameter in the funciton header. Save your sequences to a .txt file.

5. **Experiments** For each model configuration, generate loss plots for training and validation. Output your model's **test set cross entropy loss at epoch 10** using the print statement in the code release packet. Loss print functions should look like this:
   "Test set loss: xxxxx"

   Here are some **estimated thresholds** for a high pass for different experiments in this assignment. Note that this may change (in your favor) depending on the overall class performance!

- rnn - cross entropy loss: 1.97
- lstm - cross entropy loss: 1.4
- noteacherforcing - cross entropy loss: 3.4

(a) Train a vanilla RNN using teacher forcing and a sequence length of 16, 128, 512. Save the best performing model from your experiments in this question. **Please store this model in a folder named "models" and make it loadable with the flag –experiment 'rnn' in main.py.**

(b) Do the same for an LSTM. For the LSTM with sequence length 128, run an additional experiment where you increase the number of neurons in the hidden layer. Save the best performing model from your experiments in this question. **Please store this model in a folder named "models" and make it loadable with the flag –experiment 'lstm' in main.py.**

(c) Often, real-world RNNs and LSTMs are trained initially *with teacher forcing*. Once the performance is satisfactory, they're further trained *without teacher forcing*. Now that we've done teacher forcing, let us train our models without it, so that we are learning to predict from our own sequence. For this part, you will need to generate a character at every time step, not just the end of the sequence. At each time step, just take the maximum value of the output distribution. **DON'T FURTHER TRAIN YOUR TEACHER-FORCED MODELS, JUST TRAIN FROM SCRATCH WITHOUT TEACHER FORCING**. This is probably best done if you create another LSTM class and restrict all the changes to the forward function. Repeat the varying sequence length experiment for the **LSTM without** teacher forcing. Save the best performing model from your experiments in this question. **Please store this model in a folder named "models" and make it loadable with the flag –experiment 'noteacherforcing' in main.py.**

(d) Use the **generate.ipynb** file to generate 3 sample text pieces for your best performing LSTM, one at

i. $T = 1$
ii. $T = 2$
iii. $T = 0.5$

They should be of reasonable length (around 1000 characters). Save the excerpts in "temp0.5.txt", "temp1.txt", and "temp2.txt".

Start early. Training takes longer than you think, and datahub GPUs might get overused. Don't forget that there's Google Colab as a backup. Good luck!