LING 574 HW6

Due 11:59PM on May ~~15~~ 17, 2025

In this assignment, you will

- Develop understanding of recurrent neural networks
- Implement components of data processing
- Implement key pieces of two variants of a recurrent model architecture
- Train recurrent models for text classification and language modeling

All files referenced herein may be found in `/mnt/dropbox/24-25/574/hw6/` on patas.

1 Recurrent Neural Network Encoders [50 pts]

1.1 Written Questions [20 pts]

Q1: Understanding RNNs

- What is the main limitation of feed-forward neural networks that is overcome by recurrent networks, and how do recurrent networks achieve this? [3 pts]
- The Vanilla RNN equation has the form $h_t = f(h_{t-1}, x_t)$. What extra 'ingredient' does the LSTM add to this general form? What problem is the LSTM designed to solve? [2 pts]

Q2: LSTM Update One of the "central" equations in the LSTM computation is the following:

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t$$

This equation performs an essential update of one part of the LSTM. Please answer: [10 pts]

- What is c_t?
- What is the range of f_t and what is its purpose?
- What is the range of i_t and what is its purpose?
- What is $\hat{c}_t$?
- In your own words, describe how this equation implements the central "update" inside of an LSTM.

Q3: Counting parameters Let d_e be the dimension of word embeddings and d_h the hidden state size. Focusing on just the recurrent cell (and so ignoring the embedding and output layers): [5 pts]

- How many parameters are there in a Vanilla RNN cell? [2 pts]
- How many parameters are there in an LSTM cell? [3 pts]

Note: for this problem, you can assume that the RNN cell is at the 'bottom' of a possibly-deep RNN, so the inputs to the cell are word embeddings, not earlier layers' hidden states.

1.2 Implementing an RNN Sentiment Classifier [20 pts]

In the coding portion of this assignment, you will implement (components of) a classifier for the Stanford Sentiment Treebank, using RNNs as encoders. In particular, the model will take the final hidden state of an RNN that has read reviews as input in order to predict the sentiment labels thereof. Here, you will implement some data pre-processing and two major types of RNN "cell" (i.e. one time-step of computation). These are then used in other RNN modules that we provide to process entire sequences.

Q1: Data processing The reviews in the SST dataset come in various lengths. In the previous models we have looked at in the class, this has not been an issue because they rely either on a bag-of-words representation (Deep Averaging Network) or a fixed-sized window of previous tokens (Feed-Forward Language Model). RNNs, however, require the use of padding: given a batch of reviews of various lengths, we pad the shorter sequences with a special padding token so that all sequences are as long as the longest one.
In `data.py`, please implement the `pad_batch` method. Please read the method signature and docstring carefully for details on the input and output. [3 pts]

Q2: Vanilla RNN Cell The "cell" of an RNN does one time-step of computation. For a Vanilla RNN, we saw that this was

$$h_t = \tanh\left(W_h h_{t-1} + b_h + W_x x_t + b_x\right)$$

where h_{t-1} is the previous hidden state, x_t is the current input, and the Ws and bs are parameters for linear transformations.
In `model.py`, implement this computation in `VanillaRNNCell.forward`. The initializer defines the linear layers that you will need. [7 pts]

Q3: LSTM Cell An LSTM cell computes the next hidden state and memory based on the previous hidden state and memory together with the current input. Please consult these slides for the entire set of equations (and details about motivation).
In `model.py`, implement this computation in `LSTMCell.forward`. The initializer defines the linear layers that you will need. [10 pts]

1.3 Running the Classifier [10 pts]

`run.py` contains a basic training loop for SST classification, using the last hidden state of an RNN. It will record the training and dev loss at each epoch, and save the best model according to dev loss. At the end, it samples 10 random dev data points and prints the review, the gold label, and the model's prediction.

Q1: Four different runs By default, a Vanilla RNN will be used. You can use an LSTM by specifying `--lstm` as a command-line argument. Following this paper, we have added dropout to the non-recurrent connections (i.e. from the inputs and to the output) of the model.
Please run each of the following variations. For each run, include in your readme.pdf: the best dev loss, the epoch at which the best dev loss was achieved, and the best model's dev accuracy. [4 pts]

- Vanilla RNN, default parameters. (This is just `run.py` with no command-line arguments.)
- Vanilla RNN, with L_2 regularization (via `--l2`) at 1e-4 and dropout (via `--dropout`) at 0.5.
- LSTM, default parameters.
- LSTM, with L_2 regularization (via `--l2`) at 1e-4 and dropout (via `--dropout`) at 0.5.

Q2: Inspecting outputs For the fourth run above, please include in your readme.pdf the 10 random dev examples, with gold labels and model predictions here. In 2-3 sentences, describe what you see and observe any trends in what the model gets right and what (and/or how) it gets things wrong. [6 pts]

2 Recurrent Neural Network Decoders/Taggers [25 pts]

2.1 Written Portion [10 pts]

Q1: Evaluating Language Models Given a corpus $W = w_1 w_2 \dots w_N$ (so N is the number of tokens in the corpus), a common (intrinsic) evaluation metric for language models is perplexity, defined as

$$PP(W) = P(w_1 \dots w_N)^{-\frac{1}{N}}$$

This can be thought of as the inverse probability that the model assigns to the corpus, normalized by the size of the corpus.

- Is a lower or higher perplexity better? [1 pts]
- For a recurrent language model, write an expression for $P(w_1 \dots w_N)$ using the chain rule of probability. How is this different from the expression for a feed-forward language model? [3 pts]
- Show that
$$PP(W) = e^{-\frac{1}{N}\sum_{i=1}^{N} \log P(w_i|w_{<i})}$$
where $w_{<i} = w_1 w_2 \dots w_{i-1}$ and log is the natural (base e) logarithm. [2 pts]

  [Note: using base e measures perplexity in a unit known as nats. Using base 2 would measure it in bits.]
- What is another name for the exponent $-\frac{1}{N}\sum_{i=1}^{N} \log P(w_i \mid w_{<i})$ in the above expression? [Hint: it appears in training as well.] [1 pts]
- Suppose that the same text corpus were tokenized with two different vocabularies of different sizes (perhaps, e.g., one replaces infrequent tokens with an UNK token) and two language models were trained on the resulting tokenized text. All else being equal, would you expect perplexity to be lower or higher for the model with a smaller vocabulary? What consequences does this have for comparing different language models? [3 pts]

2.2 Implementing an RNN Character Language Model [5 pts]

In the coding portion of this assignment, you will implement basic data processing for an LSTM character-level language model for the Stanford Sentiment Treebank, using an RNN as tagger. You can read `model_lm.py` and `run_lm.py` (which you do not need to modify) to see more details; the former uses some torch-specific RNN tools instead of the LSTM that we are using for part 1 of this assignment.

Q1: Data processing Recall from the lectures that we can view language modeling as a sequence tagging task. That is, each element of an input sequence is tagged with a certain target. This gets operationalized in the data processing pipeline; you will generate the inputs/targets for one line of text.
In `data.py`, please implement the `example_from_characters` method. Please read the method signature and docstring carefully for details on the input and output.

2.3 Running the Language Model [10 pts]

`run_lm.py` contains a basic training loop for SST language modeling. It will record the training and dev loss (and perplexity) at each epoch, and save the best model according to dev loss. Periodically (as specified by a command-line flag), it also outputs generated text from the best model.

Q1: Default parameters Execute `run_lm.py` with its default arguments. Paste below the texts that are generated every 4 epochs, as well as the epoch with the best dev loss and the dev perplexity from that epoch. In 2-3 sentences, describe any trends that you see. [Note that generated text will not necessarily be completely coherent: recall that this is a character-level language model.] [3 pts]

Q2: Modify hyper-parameter(s) Re-run the training loop, modifying some combination of the following hyper-parameters, which are specified by command-line flags:

- Hidden layer size
- Embedding size
- Learning rate
- Number of epochs [in particular: making it larger]
- Softmax temperature.
- L_2 regularization coefficient.
- Dropout (probability with which neurons are dropped from the input and to the output during training)

Include your model's generated texts here. In 2-3 sentences, state exactly what hyper-parameter change(s) you made, and what effects (if any) you see in terms of the dev set perplexity and text that the model generated. [2 pts]

Q3: Comparison to feed-forward language model In 2-3 sentences, please explain what differences you see in the text generated by this LSTM language model and the feed-forward language model that you trained in HW5. What do you think may be causing these effects (or lack thereof)? [5 pts]

3 Testing your code

In the dropbox folder for this assignment, we will include a file `test_all.py` with a few very simple unit tests for the methods that you need to implement. You can verify that your code passes the tests by running `pytest` from your code's directory, with the course's conda environment activated.

Submission Instructions

In your submission, include the following:

- readme.(txt|pdf) that includes your answers to §1.1, §1.3, §2.1, and §2.3.
- `hw6.tar.gz` containing:
  - run_hw6.sh. This should contain the code for your run commands for §1.3 and §2.3 above. You can use scripts from previous assignments as a template.

- data.py
- model.py