Ling 575j hw4

Due 11PM on October 7, 2024

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
- Transition from Edugrad to PyTorch code

Submission Instructions

This assignment contains both written and programming portions. The answers to written questions must be submitted in a *.txt or *.pdf file to Blackboard. You will receive an invitation link to complete the programming portion via Github Classroom. This will open a Github repository with starter code and missing portions for you to complete. When you are finished with implementation, simply commit and **push** your changes to the online repository that was created for you. Unless you request otherwise, I will grade your work **as of the most recent commit** in your repository, subject to any applicable late penalties.

1 Recurrent Neural Network Encoders [30 pts]

Q1: Understanding RNNs [4 pts]

a. What is the main limitation of feed-forward neural networks that is overcome by recurrent networks, and how do recurrent networks achieve this? [2 pts]

b. 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: tanh [8pts] Gated RNNs use the hyperbolic tangent (tanh) activation function, defined as:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

a. Show that $\tanh(x) = 2\sigma(2x) - 1$, where $\sigma(x)$ is the sigmoid function. [4 pts]

b. Show that $\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$. [4 pts]

Q2: LSTM Update [12 pts] 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:

a. What is c_t? [1 pt]

b. What is the range of f_t and what is its purpose? [2 pts]

c. What is the range of i_t and what is its purpose? [2 pts]

d. What is $\hat{c}_t$? [1 pt]

e. In your own words, describe how this equation implements the central "update" inside of an LSTM (2-3 sentences). [4 pts]

Q3: Counting parameters [6 pts] 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):

a. How many parameters are there in a Vanilla RNN cell? [2 pts]

b. How many parameters are there in an LSTM cell? [4 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.

2 Implementing an RNN Sentiment Classifier [16 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 [4 pts] 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.

Q2: Vanilla RNN Cell [4 pts] 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.

Q3: LSTM Cell [8 pts] 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 the LSTM lecture 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.

3 Running the Classifier [12 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 [**8 pts**] 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 written submission: the best dev loss, the epoch at which the best dev loss was achieved, and the best model's dev accuracy.

- 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 [**4 pts**] For the fourth run above, please include in your written submission 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.