Ling 575j hw5

Due 11pm on May 4, 2023

In this assignment, you will

- Develop understanding of a feed-forward neural language model
- Implement components of data processing and text generation
- Implement key pieces of the model architecture

All files referenced herein may be found in `/dropbox/22-23/575j/hw5/` on patas.

1 Understanding the Feed-Forward Language Model [20 pts]

Q1: Architecture You can find a description of the model in the second half of the slides from lecture #7. [12 pts]

- How many parameters are there? Please write your answer in terms of the following quantities: d_e, the token embedding dimension; $|V|$, the size of the vocabulary; d_h: the dimension of the hidden layer; n: the n-gram size, i.e. how many previous tokens are used as input to the model. [Note: you may assume that there are no "direct connections" between the embeddings and the final layer.]
- A traditional n-gram language model estimates probabilities $p(w_t|w_{t-1}, \ldots, w_{t-n})$ using counts from a corpus. How does the feed-forward language model compute this probability? Answer with a sentence or two describing the overall computation.
- What is a major advantage of the feed-forward language model over traditional n-gram models?

Q2: tanh The model uses the hyperbolic tangent (tanh) activation function, defined as: [8 pts]

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

- Show that $\tanh(x) = 2\sigma(2x) - 1$, where $\sigma(x)$ is the sigmoid function.
- Show that $\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$.

2 Implementing the Feed-Forward Language Model [40 pts]

In the remainder, you will implement key components of a *character-level* language model. Technically, moving from words to characters just changes the data pre-processing and vocabulary. But it has one big advantage for us: character-level language models have a very small vocabulary (on the order of 50-70) when compared to words (tens of thousands usually). The output of a language model is a softmax over the vocabulary, and so having a much smaller vocabulary greatly speeds up computation at that step (since the sum in the denominator of softmax is costly).

Q1: Data processing The basic ingredient of a language model is a dataset of next-token predictions. In `data.py`, you will find a basic dataset class SSTLanguageModelingDataset. In its from_file method, it iterates through the lines in a file, and calls a helper function to generate example pairs. [10 pts]

- Implement the method `examples_from_characters`. Read the docstring closely for desired behavior.

Q2: Implementing tanh In `ops.py`, you will find a skeleton Operation for tanh. Using your written answer above as a guide, implement the forward and backward methods for this op. [12 pts]

Q3: Implementing the Language Model In `model.py`, you will find the main model class FeedForwardLanguageModel, with its initialization method written. Implement the `.forward` method, using its docstring as a guide. [Hint: `ops.concat`, which we provide, will be necessary. As above, do not provide any "direct connections".] [10 pts]

Q4: Generating the next character In `run.py`, there is code for generating text from a language model. You will implement one helper method: `sample_next_character.py`. The docstring specifies the method's behavior: it takes a batch of distributions over the vocabulary (characters), and samples a batch of next characters. Text generation basically loops over this operation. [Hint: `np.random.choice` is your friend.] [8 pts]

3 Running the Language Model [15 pts]

`run.py` contains a basic training loop for a feed-forward language model, which will record the training loss and generate text every N epochs (controlled by the flat `--generate_every`, set to 4 by default).

Q1: Basic parameters Execute `run.py` with its default arguments. Paste below the texts that are generated every 4 epochs. 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.] [5 pts]

Q2: Modify one hyper-parameter Re-run the training loop, modifying one of the following hyper-parameters, which are specified by command-line flags:

- Hidden layer size
- Embedding size
- Number of previous characters (i.e. n-gram size; this is `--num_prev_chars`)
- Learning rate
- Number of epochs [in particular: making it larger]
- Softmax temperature. (We did not cover this in class: higher values of this temperature make the softmax probabilities more closely approximate $\arg\max$, while lower values make it look more and more like a uniform distribution. A value of 1 is the 'default' softmax value.)

Include your model's generated texts here. In 2-3 sentences, state exactly what hyper-parameter change you made, and what effects (if any) you see in terms of the text that the model generated. [10 pts]

4 Testing your code

In the dropbox folder for this assignment, you will find 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 and §3.
- `hw5.tar.gz` containing:
  - run_hw5.sh. This should contain the code for activating the conda environment and your run commands for §3 above. You can use run_hw4.sh from the previous assignment as a template.
  - data.py
  - model.py
  - ops.py
  - run.py