Ling 282/482 hw3

Due 11pm on September 30, 2024

In this assignment, you will

- Implement basic computational operations in the forward/backward API
- Develop your understanding of feed-forward networks for text classification
- Implement the Deep Averaging Network
- Understand a more sophisticated optimizer than vanilla SGD
- Explore some regularization techniques

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 Implementing the Deep Averaging Network + Loss [36 pts]

Q1: Computation Graph Operations [**24pts**] In `ops.py`, you will find some ops already defined, and some for you to implement. Implement the forward and backward methods of the following operations. You can and should use pre-existing functions like `np.log` and `np.exp` in your forward methods: the goal is understanding how forward and backward relate, not coding the exponential or logarithmic functions from scratch.

Recall: (i) you can use the list `ctx` in forward to store any values that you need when computing gradients in `backward`; (ii) backward needs to return a *list* of the same size as the number of inputs to forward; each element of the list contains the gradient for the respective input. We have provided shell lists for this purpose.

- `log`
- `sigmoid`
- `multiply` (This is *element-wise* multiplication of two vectors/matrices)
- `exp`
- `softmax_rows` (Note: you should use already defined ops/methods to define this method, so there's no separate backward here)

- `cross_entropy_loss` (The same note above applies here)

Q2: Implement bag of words representation [4pts] The input to a DAN model is a *bag of words*: a vector (a batch of such vectors, really) with counts of the vocabulary items at the corresponding indices (see lecture 7). In `data.py`, implement this representation in `SSTClassificationDataset.example_to_tensors`.

Q3: Implement DeepAveragingNetwork.forward [8 pts] In `model.py`, you will find the skeleton of a Deep Averaging Network (with two hidden layers). In particular, we have implemented the initialization of the Module. You need to implement the forward method, which takes two inputs: a batch of bag-of-words representations and a list of lengths, and then outputs scores for each class label for each row in the batch. See the doc-string for more information. Note: the edugrad repository (and the slides for HW4) has an example of implementing forward for an MLP that you can use for inspiration.

2 Implementing Adagrad [8 pts]

Recall from the lecture on DANs the update rule for Adagrad:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{G_{t,i} + \epsilon}} g_{t,i}$$

$$G_{t,i} = \sum_{k=0}^{t} g_{k,i}^2$$

$$\text{where } g_{t,i} := \nabla_{\theta_{t,i}} \mathcal{L}(\theta_t)$$

In `optim.py`, please implement this update rule by incrementing $G_{t,i}$, computing the adjusted learning rate, and finally updating the parameter values. Note: you can look at edugrad.optim.SGD for an example step method.

3 Running the Deep Averaging Network [16 pts]

In `run.py`, you will find a basic training loop for building a DAN and training it on the Stanford Sentiment Treebank. There are several command-line arguments specifying different arguments. The default arguments for various hyper-parameters are:

- Hidden dimension: 50
- Embedding dimension: 50
- Batch size: 32
- Number of epochs: 6
- Word dropout: 0.0 (i.e. no word dropout is applied)
- L_2 regularization: 0.0 (i.e. no regularization is applied)

Each run will print to stdout (and, therefore, the output file you specify in your condor job script) the training and dev set loss for each epoch, and then the final model's accuracy on the dev set.

Q1: Run 1, default arguments Run the main training loop by calling `run.py` with all of the default arguments. Record the outputs of this run (per epoch train/dev loss, final dev accuracy) here:

In 2-3 sentences, describe any trends that you see in the training and dev set losses over the course of training, and any differences between the two. What do these trends suggest to you? [4 pts]

Q2: Run 2, L_2 regularization Recall that L_2 regularization adds a penalty to the loss function corresponding to the size of the model's parameters:

$$\mathcal{L}'(\theta, y) = \mathcal{L}(\theta, y) + \lambda ||\theta||^2$$

Run the main training loop by calling `run.py`, but with the L_2 parameter set to 1×10^{-5} (this corresponds to λ in the above equation). Record the outputs of this run (per epoch train/dev loss, final dev accuracy) here:

In 2-3 sentences, describe any trends that you see in the training and dev set losses over the course of training, and any differences between the two. What do these trends suggest to you? [4 pts]

Q3: Run 3, L_2 regularization + Word Dropout Recall that word dropout randomly drops some percentage of word embeddings from the input. Run the main training loop by calling `run.py`, but with the L_2 parameter set to 1×10^{-5} and word dropout set to 0.3. Record the outputs of this run (per epoch train/dev loss, final dev accuracy) here:

In 2-3 sentences, describe any trends that you see in the training and dev set losses over the course of training, and any differences between the two. What do these trends suggest to you? [4 pts]

Q4: Cross-run comparison What trends do you notice in these metrics across the three runs? What do they suggest to you about the impact of these hyperparameters on the model's performance? [4 pts]

4 Understanding the Feed-Forward Language Model [12 pts]

You can find a description of the model in the second half of the slides from lecture #7.

- 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. [4 pts]
- 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. [4 pts]
- What is a major advantage of the feed-forward language model over traditional n-gram models? [4 pts]