CSE 517: Homework 4
Recurrent Neural Networks, Attention, and Reading Comprehension

Due Fri Mar 15 2019 11:59pm

Hi Class! So, this is the last homework of the quarter, and I hope you are excited to play with TensorFlow for reading comprehension!

Submission instructions As usual, submit 2 files on Canvas — **Code** (`HW4.tgz`) and **Report** (`HW4.pdf`). The usual submission guideline applies, which we omit now for brevity.

1 Machine Reading Comprehension with RNNs and Attention

Machine reading comprehension has been a recently hot research topic in deep learning and NLP literature. As input to the task, one is given a paragraph and a question about the paragraph. The output is an answer to the question using the given paragraph as context. For example, given the following paragraph:

```
Oxygen is a chemical element with symbol O and atomic number 8.  It is a member of
the chalcogen group on the periodic table and is a highly reactive nonmetal and oxidizing
agent that readily forms compounds (notably oxides) with most elements.
```

The task is to answer a reading comprehension question such as:

```
What is the atomic number of oxygen?
```

In this assignment, we will design and experiment with variants of RNN models that can learn to answer reading comprehension questions such as above.

1.1 Dataset

You will experiment with the Stanford Question Answering Dataset (SQuAD) [4]. So that you can finish the assignment without strong computing resources, we provide you with a small subset of the dataset through Canvas. We suggest that you check out the SQuAD website (`http://stanford-qa.com`) and explore examples to be familiar with the dataset. If you are in the mood for a bigger challenge, feel free to use the whole dataset available at the official website, which contains 100,000+ questions! Note: there are also newer versions of SQUAD (like SQUAD 2.0), but we are just focusing on a subset of questions from the original EMNLP 2016 dataset.

Each example in SQuAD consists of a paragraph taken from a Wikipedia article and a crowdsourced question about the paragraph. In fact, the example shown above is from SQuAD. In this dataset, the answer is required to be a word or a phrase in the context paragraph. Hence the answer to the above question must be `8`, but cannot be `eight`, even though it is a correct answer. While this restriction disallows users to ask a wide variety of questions, it allows easy and consistent evaluation.

Download the data zip file, `squad.zip` from Canvas, and unzip it. By default, the code will expect it to be at `$HOME/data/squad`, but you can put it anywhere and change the directory for data accordingly in `cli.py` file of the provided code base.

1.2 Code

In this assignment, we provide you with start-up code to speed you up with TensorFlow. Download `mc.zip` and you will find the following files:

- `cli.py`: `cli` stands for "Command Line Interface". This is the file that you will invoke from command line to train and test your model. It contains all hyperparameters of the model, and you will need to add more hyperparameters as necessary in this file for your implementation.
- `model.py`: This file describes the actual model of the machine comprehension system. You will be mostly working on this file, and will be asked to implement `rnn_forward()` and `attention_forward()` functions.
- `data.py`: This module preprocesses the dataset and feed it to model.
- `evaluate.py`: This is the official evaluation script provided by SQuAD.
- `evaluation.py`: This module contains class definition for an `Evaluation` object.
- `input_queue.py`: This module enables asynchronous data feeding using TensorFlow's Queue.
- `run.py`: This file is the main file that contains training and test sequences. We recommend you to look into this code to understand the overall pipeline of training and testing.
- `trainer.py`: This file defines optimization operation that updates model weights.
- `demo.py`: Script for hosting demo server. Accessed through `cli.py`.
- `templates`: HTML templates for demo website.
- `tf_utils.py`: Custom tensorflow utils.

Prerequisites. You will first need to install Python 3.x (`https://python.org`). We highly recommend using `virtualenv` for installing all python packages (`https://virtualenv.pypa.io`). Make sure to also install Python's `pip` package as well (`https://pip.pypa.io/en/stable/installing`). Also make sure `pip` is up-to-date, by running:

```
$ pip install --upgrade pip
```

Now, to install all required packages, (depending on if you are using a machine with gpus) on the command line, simply run:

If you don't have a gpu:

```
$ pip install --upgrade tensorflow tqdm nltk flask
```

Or if you have a gpu, run:

```
$ pip install --upgrade tensorflow-gpu tqdm nltk flask
```

You will also need to install 'punkt' data for nltk:

```
$ python -m nltk.downloader -d $HOME/nltk_data punkt
```

Lastly, you will need to download GloVe [3] vectors trained on Wikipedia, `glove.6B.50d.txt`, from Canvas and place it at `$HOME/data/glove/`. Note that you can also specify the directory with `--glove_dir` flag when running `cli.py`. To make sure that your code is runnable, try the following command in your code directory:

```
$ python cli.py --train --draft --num_steps 10 --val_period 10 --out_dir out/draft
```

`num_steps` is the duration of training (default is 20000), `--val_period` indicates how often you evaluate your model on the validation data (result will be concurrently printed out, default is every 1000 steps), and `out_dir` indicates the directory where all output files will be stored. If your run ends without any error, you are all set! Note that you will need to configure `data_dir` in `cli.py` file if you saved your dataset in other folder than `$HOME/data/squad`.

1.3 Baseline: CBOW

Now let's get into the first model for machine comprehension. It is already implemented in `model.py` and used by default. In fact, what you just ran previously is the training sequence of this model, with 'draft' mode (indicated by `--draft`) to see if everything is okay.

In this part of homework, you are expected to understand the model, and also understand how TensorFlow lets us implement this model. We highly recommend you to go over TensorFlow Tutorial (`https://tensorflow.org/get_started/get_started`) and TensorFlow API (`https://tensorflow.org/api_docs/`) as needed.

Model. Let $\mathbf{A} \in \mathbb{R}^{d \times V}$ be an embedding matrix for words (fixed and obtained from GloVe vectors), where V is the vocabulary size and d be the size of the word embedding. Using this embedding matrix, we obtain a sequence of question word vectors, $\mathbf{q}_1 \dots \mathbf{q}_{J_\mathbf{q}} \in \mathbb{R}^d$ and a sequence of context word vectors, $\mathbf{x}_1 \dots \mathbf{x}_{J_\mathbf{x}} \in \mathbb{R}^d$, where $J_\mathbf{q}$ and $J_\mathbf{x}$ are the lengths of the question and the context words. In order to obtain single representation of the question, we simply average the question word vectors (thus CBOW):

$$\bar{\mathbf{q}} = \frac{1}{J_\mathbf{q}} \sum_t^{J_\mathbf{q}} \mathbf{q}_t \tag{1}$$

Then we obtain the prediction for the start index of the answer phrase with the following softmax equation:

$$\hat{\mathbf{y}}_k^{\text{start}} = \frac{\exp(\mathbf{W}^{\text{start}}[\mathbf{x}_k; \bar{\mathbf{q}}; \mathbf{x}_k \circ \bar{\mathbf{q}}] + b^{\text{start}})}{\sum_t \exp(\mathbf{W}^{\text{start}}[\mathbf{x}_t, \bar{\mathbf{q}}; \mathbf{x}_t \circ \bar{\mathbf{q}}] + b^{\text{start}})} \tag{2}$$

where $\mathbf{W}^{\text{start}} \in \mathbb{R}^{1 \times 3d}$ is a trainable weight matrix, $b^{\text{start}} \in \mathbb{R}$ is a trainable bias, $[;]$ is row-wise concatenation, and $\circ$ is element-wise multiplication of vectors. We also similarly define $\hat{\mathbf{y}}^{\text{end}}$ for the end index of the answer phrase. For training, we minimize the negative log probability of the correct start and end indices (y^{start} and y^{end}):

$$L = -[\log(\hat{\mathbf{y}}_{y^{\text{start}}}^{\text{start}}) + \log(\hat{\mathbf{y}}_{y^{\text{end}}}^{\text{end}})] \tag{3}$$

This loss is also called cross-entropy loss. At test time, the system outputs the (i, j) span of the context for the answer, where $i \leq j$ and $p = \hat{\mathbf{y}}_i^{\text{start}} \hat{\mathbf{y}}_j^{\text{end}}$ is maximized.

To train the model and save the model in `out/cbow` directory, run:

```
$ python cli.py --train --out_dir out/cbow --device_type gpu
```

To test the model, you run the same command without `--train` flag:

```
$ python cli.py --out_dir out/cbow
```

You will see several outputs on the screen after the test ends. 'acc1' and 'acc2' indicate the accuracy of getting start and end indices correct, respectively. The two official metrics are 'EM' and 'F1' scores. The EM score stands for exact match, and you get a point if and only if the answer exactly matches the answer. The F1 score is a softer evaluation metric, which gives sub-1 score for partially correct answer.

Train and test with the given CBOW model. Use `--fresh` flag to delete the saved files in `--out_dir` and run a fresh training or testing routine. Also note that the model is auto-saved every 120 seconds by default (control this with `--save_model_secs`), and even if you abort training, you can start from your last save point with the same command (unless you give `--fresh` option, which wipes out the output directory, so be careful!). You can use the `--device_type` flag to set whether you use gpus or cpus (we recommend using gpus for training). There are also more flags described in the `cli.py` file for changing other settings.

1.4 Gated Recurrent Units (GRUs)

Gated Recurrent Units (GRUs) [2] are a modern variant of RNNs. The key strength of GRUs over vanilla RNNs is that GRUs can learn to retrain a longer term memory in their hidden states. GRU can contextually decide whether (and how much) to update the new hidden state $\mathbf{h}^{(t)} \in \mathbb{R}^h$ with new information derived from new input x_t, or to retain previous hidden states, as controlled by the update and reset gates. Give a new input $\mathbf{x}_t \in \mathbb{R}^d$, the output $\mathbf{y}_t \in \mathbb{R}^d$ is computed as follows:

$$\mathbf{r} = \sigma\left(W^{rx}\mathbf{x}_t + U^{rh}\mathbf{h}_{t-1} + b^r\right) \quad \text{reset gate} \tag{4}$$

$$\mathbf{z} = \sigma\left(W^{zx}\mathbf{x}_t + U^{zh}\mathbf{h}_{t-1} + b^z\right) \quad \text{update gate} \tag{5}$$

$$\tilde{\mathbf{h}}_t = \tanh\left(W^{hx}\mathbf{x}_t + U^{hh}(\mathbf{r} \odot \mathbf{h}_{t-1}) + b^h\right) \quad \text{candidate } \mathbf{h} \tag{6}$$

$$\mathbf{h}_t = \mathbf{z} \odot \mathbf{h}_{t-1} + (\mathbf{1} - \mathbf{z}) \odot \tilde{\mathbf{h}}_t \quad \text{new hidden-state} \tag{7}$$

$$\mathbf{y}_t = \mathbf{h}_t \quad \text{output} \tag{8}$$

where $W^{rx}, U^{rh}, W^{zx}, U^{zh}, W^{hx}, U^{hh} \in \mathbb{R}^{d\times d}$ and $b^r, b^z, b^h \in \mathbb{R}^d$ are parameters (weights) to learn, and $\odot$ is the element-wise multiplication.

Now, how can we use GRUs to solve the machine reading comprehension problems? In this assignment, we will use the hidden states of GRUs instead of CBOW embeddings in Equation 1 and 2. More specifically, $\mathbf{x}_t$ (and $\mathbf{x}_k$) in Equation 2 can be replaced with $\mathbf{h}_t$ from GRUs that encode the given paragraph. Similarly, $\mathbf{q}_t$ in Equation 1 can be replaced with $\mathbf{h}_t$ from GRUs that encode the given question sentence. Feel free to think of (and perhaps even experiment with) other possible options, if time permits!

Compared to CBOW, we expect that GRUs will have a better shot at making use of important contextual information. For instance, consider the example we gave in the first page. Ideally, given the question, we want the hidden state at the word `8` to be aware of the fact that it is an `atomic number`, described by immediate two previous words. Also, we also want it to know that its subject is `Oxygen`, which is quite far away (11 words). GRU will be effective for encoding such information in the hidden state vector of `8`.

1.5 Attention

We revisit Equation 1, which represents the entire question with a single vector through arithmetic average (also called mean pooling). While averaging is often an effective method for summarizing a list of vectors, we often can do better, by computing weighted average instead of even average of the vectors. We also want the average weights to be different for different context words when we are computing Equation 2. That is, instead of having single $\bar{\mathbf{q}}$ for every context word, we define $\bar{\mathbf{q}}_k$ for every $\mathbf{x}_k$, and we define $\bar{\mathbf{q}}_k$ by

$$\bar{\mathbf{q}}_k = \sum_t^{J_\mathbf{q}} \mathbf{p}_{k,t}\mathbf{q}_t \tag{9}$$

where $\mathbf{p}_k$ is the average weight vector, also called *attention weights* [1], satisfying $\sum_t \mathbf{p}_{k,t} = 1$. The attention weights basically indicate which question words should be attended at each time step, For instance, at time step k, if we want to focus on t-th word of the question, then $\mathbf{p}_{k,t}$ will be close to 1.

The attention weights are also learned latently, and there exist various ways to define them, and we describe one of them here:

$$\mathbf{p}_k = \text{softmax}_t(\mathbf{W}^{\mathbf{p}}[\mathbf{x}_k; \mathbf{q}_t; \mathbf{x}_k \circ \mathbf{q}_t] + b^{\mathbf{p}}) \tag{10}$$

where $\mathbf{W}^{\mathbf{p}} \in \mathbb{R}^{1\times 3d}$ and $b^{\mathbf{p}} \in \mathbb{R}$ are trainable weight matrix and bias, respectively, and $\circ$ indicates elementwise multiplication. You are free to explore other ways to define Equation 10.

Finally, note that RNNs and attention mechanism are two different concepts, but they are also complimentary to each other and it is often very powerful to use both of them.

1.6 Monitoring with TensorBoard

Training with RNNs can take a few hours, and training with attention might take more than 10 hours. Hence it is important to know early whether your model is learning something. TensorBoard allows you to monitor this easily. As your model trains, open another terminal and run:

```
$ tensorboard --logdir out/
```

You can access TensorBoard at `http://localhost:6006`. 'model' gives you how loss and accuracies on *training* data change over time. Do you see loss decreasing over time? 'val' gives you the results on the validation data over time.

1.7 Demo

We provide a simple script for hosting demo server with your trained model. You can write your own paragraph and question, and see how the system answers the question. After training cbow model, run the demo script:

```
$ python cli.py --out_dir out/cbow --serve
```

This will host a local demo server at `http://localhost:5000`. The website will display two inputs: context and question. Try the example paragraph and question shown in the beginning of the section. Do you get a correct answer? You can try rnn and attention models by giving `--mode rnn` and `--mode att`, respectively, with corresponding output directories, after you have trained them.

1.8 Deliverables

- (3pt) Train and test CBOW model. Does the model seem to work? If not, discuss what might be the reason for performance to remain low.
- (3pts) Implement `rnn_forward()` in `model.py`, with GRU as a variant of RNNs. Use `GRUCell` and `bidirectional_dynamic_rnn` from TensorFlow API. Use different dropout rates (0.1, 0.3, and 0.5) during training (Use TensorFlow's `DropoutWrapper`. `config.is_train` indicates whether the model is being trained or being tested) and report their performances on test data. How does dropout affect the model's performance? Make sure to change `--out_dir` (e.g. to `--out_dir out/rnn`) to avoid overwriting! Use `--mode rnn` to call `rnn_forward()` function.
- (4pts) Implement `attention_forward()` in `model.py`, with dropout rate of your choice. Report performance on test data. Use `--mode att` to call `attention_forward()` function. If you have already implemented GRUs, build attention mechanism on top of them.
- (1-3pts) **Bonus**: do whatever you want to improve the model's performance. Let us know what you did and how much you improve.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

[4] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *EMNLP*, 2016.