CS 224N: Assignment #4: Reading Comprehension

Due date: Friday, March 17th 2017 at 11:59pm PST. You are allowed to use 3 late days maximum for this assignment, so that we can complete grading on time.

This assignment can be completed in groups of up to 3 people. We encourage groups to work together productively so that all students understand the submitted system well. We ask that you abide by the university Honor Code and that of the Computer Science department, and make sure that all of your submitted work (except as acknoweldged) is done by yourself and your team members only. It is fine to take ideas from other papers on reading comprehension, but you should acknowledge them in your write-up.

Please review any additional instructions posted on the assignment page at http://cs224n.stanford.edu/assignments.html. When you are ready to submit, please follow the instructions on the course website. Submission will be via CodaLab.

1 Overview

You are becoming a researcher in NLP with Deep Learning with this programming assignment! You will implement a neural network architecture for Reading Comprehension using the recently published Stanford Question Answering Dataset (SQuAD) [1].[1]

SQuAD is comprised of around $100K$ question-answer pairs, along with a context paragraph. The context paragraphs were extracted from a set of articles from Wikipedia. Humans generated questions using that paragraph as a context, and selected a span from the same paragraph as the target answer. The following is an example of a triplet ⟨question, context, answer⟩:

Question: Why was Tesla returned to Gospic?
Context paragraph: On 24 March 1879, Tesla was returned to Gospic under police guard for not having a residence permit. On 17 April 1879, Milutin Tesla died at the age of 60 after contracting an unspecified illness (although some sources say that he died of a stroke). During that year, Tesla taught a large class of students in his old school, Higher Real Gymnasium, in Gospic.
Answer: not having a residence permit

In the SQuAD task, answering a question is defined as predicting an answer span within a given context paragraph.

In the first sections of this assignment, we describe the starter code that helps with preprocessing and evaluation procedures. After that, we give a few hints as to how you might approach the problem, and point you at some recent papers that have attempted the task. This is an open-ended assignment, where you should be working out how to do high-performance question answering on the SQuAD dataset. To meet the goal of the assignment, we expect you to explore different models by using what you have learned in class combined with your findings in the literature. Finally, we give you some practical tips on model training, and instructions on how to evaluate your model and submit it to our internal leaderboard.

2 Setup

Once you have downloaded the assignment from the website of the course, place it in a directory of your convenience (here referred as `pa4`). Before proceeding, please read the `README.md` file for additional information about the code setup and requirements. In particular, the starter code assumes a Python 2.7 installation with TensorFlow 0.12.1.[2]

In order to download the dataset, and start the preprocessing, run the following command under the main directory `pa4`. This will install several Python packages with `pip` and download about 862MB of GloVe word vectors (see below section 2.3).

[1] https://stanford-qa.com
[2] These are the defaults script provided by CS224N on Azure machines.

```
code/get_started.sh
python code/qa_data.py
```

The file `qa_data.py` takes an argument `-glove_dim` so that you can specify the GloVe word embedding dimensions that you want. This file will process the dimension specified by you.

2.1 Starter Code

The starter code provided for the assignment includes the following folders and files:

- `code/`: a folder containing the starter code:
  - `docker/`:
    - `Dockerfile`: specification of a Docker image for running your code, necessary for submitting the assignment.[3]
  - `preprocessing/`: code to prepare the data to be consumed by the model:
    - `dwr.py`: downloads and stores the distributed word representations (word embeddings).
    - `squad_preprocess.py`: utilities to download the original dataset, parse the JSON files, extract paragraphs, questions, and answers, and tokenize them. It also splits the train dataset into train and validation.
  - `get_started.sh`: the first script to execute, it downloads and preprocesses the dataset.
  - `evaluate.py`: the original evaluation script from SQuAD. Your model can import evaluation functions from this file, and you should not change this file.
  - `qa_data.py`: the code that reads the preprocessed data and prepares it to be consumed by the model.
  - `qa_model.py`: TensorFlow model definition. It contains the main API for the model architecture. You are free to change any of the code inside this file. You can delete everything and start from scratch.
  - `train.py`: Responsible for initialization, construction of the model, and building an entry point for the model.
  - `qa_answer.py`: We use this file to take in a JSON file and output another JSON file where the predictions of your model will lie. You must change the specified section in order to run this file.
- `data/`: a folder hosting the dataset downloads as well as the preprocessed data.
- `train/`: a folder containing the saved TensorFlow models.
- `log/`: a folder for logging purposes.

2.2 Dataset

After the download, the SQuAD dataset is placed in the `data/squad` folder. SQuAD downloaded files include train and dev files in JSON format:

- `train-v1.1.json`: a train dataset with around $87k$ triplets.
- `dev-v1.1.json`: a dev dataset with around $10k$ triplets.[4]

Apart from the target answer, SQuAD also provides the starting position in *character count* of the answer.

Note that there is no test dataset publicly available: it is kept by the authors of SQuAD to ensure fairness in model evaluations. While developing the model in this assignment, we will consider for all purposes the dev set as our test set, i.e., we won't be using the dev set until after

[3] The submission process will involve submitting your code to be run on another server, and Docker images provide a convenient way to package all of the dependencies that you need for running your code. Stay tuned for more information, but feel free to learn more about Docker at https://www.docker.com.

[4] A difference between train and dev is that the latter dataset has three gold answers per question instead of just one answer per question that is included in train.

initial model development. Instead, we split the supplied train dataset into two parts: a 95% slice for training, and the rest 5% for validation purposes, including hyperparameter search. We refer to these as `train.*` and `val.*` in filenames. Finally, you will be able to upload your model as a bundle to `CodaLab`[5] where you can evaluate your model on the unseen test dataset there.

2.3 Distributed Word Representations

The `get_started.sh` script downloads GloVe word embeddings of dimensionality $d = 50$, 100, 200, and 300 and vocab size of 400k that have been pretrained on Wikipedia 2014 and Gigaword 5. The word vectors are stored in the `data/dwr` subfolder.

The file `qa_data.py` will trim the GloVe embedding with the given dimension (by default $d = 100$) into a much smaller file. Your model only needs to load in that trimmed file. Feel free to remove the `data/dwr` subfolder after preprocessing is finished.

Consider This [Distributed word representations]: The embeddings used by default have dimensionality $d = 100$ and have been trained on 6B word corpora (Wikipedia + Gigaword). The vocabulary is uncased (all lowercase). Analyze the effect of selecting different embeddings for this task, e.g., other families of algorithms, larger size, trained on different corpora, et cetera.

2.4 Data Preprocessing

The starter code provides a preprocessing step that turns the original JSON files into four files containing a tokenized version of the question, context, answer, and answer span. Lines in these files are aligned.

Each line in the answer span file contains two numbers: the first number refers to the index of the first *word* of the answer in the context paragraph. The second number is the index of the last *word* of the answer in the context paragraph.

The first step is to get familiar with the dataset. Explore SQuAD and keep track of the values you may later use to limit, for example, the output size of the model. As a guide, plot histograms for context paragraph lengths, question lengths, and answer lengths.

Consider This: The preprocessing step takes the answer as a sequence of words and transforms it into two numbers. What are different possible ways to represent the answer?

Consider This [Improve tokenization]: The provided preprocessing uses NLTK. This process will result in skipping some triplets. If you want to explore other ways to tokenize the data, you can use other tools, such as Stanford CoreNLP, and try to reduce the number of skipped triplets.

3 Model Implementation

The goal of this assignment is an open-ended exploration of reading comprehension using the SQuAD task. The SQuAD website contains a leaderboard with a set of models that researchers have submitted for evaluation, and we strongly suggest that you spend time familiarizing yourself with different models, identifying key ideas from these papers that you may end up using in your own model. References to the most relevant papers are provided in section 4.

The following sections provide a possible, simple breakdown of the reading comprehension with SQuAD task. If you are unfamiliar with the task, reading it can help you make multiple architectural decisions and navigate the literature.

[5] https://worksheets.codalab.org

3.1 Problem Setup

In the SQuAD task, the goal is to predict an answer span tuple $\{a_s, a_e\}$ given a question of length n, $\boldsymbol{q} = \{q_1, q_2, ..., q_n\}$, and a supporting context paragraph $\boldsymbol{p} = \{p_1, p_2, ..., p_m\}$ of length m. Thus, the model learns a function that, given a pair of sequences $(\boldsymbol{q}, \boldsymbol{p})$ returns a sequence of two scalar indices $\{a_s, a_e\}$ indicating the start position and end position of the answer in paragraph $\boldsymbol{p}$, respectively. Note that in this task $a_s \leq a_e$, and $0 \leq a_s, a_e \leq m$.

Consider This: There are other formalizations of this problem. For instance, instead of predicting an answer span tuple $\{a_s, a_e\}$, you can predict the real answer word by word as $\{a_1, a_2, ..., a_{\texttt{<eos>}}\}$, but we leave these alternative formalizations for you to explore.

3.2 Architecture

Using what you have learned in class and in reading about the problem, you should find a way to encode the question and paragraph into a continuous representation. Good models read the paragraph with the question, just like what a human would do. We refer to this as the conditional understanding of the text.

You are strongly encouraged to get something simple and straightforward working first. A possible simple first model for this might be:

1. Run a BiLSTM over the question, concatenate the two end hidden vectors and call that the question representation.
2. Run a BiLSTM over the context paragraph, conditioned on the question representation.
3. Calculate an attention vector over the context paragraph representation based on the question representation.
4. Compute a new vector for each context paragraph position that multiplies context-paragraph representation with the attention vector.
5. Run a final LSTM that does a 2-class classification of these vectors as O or ANSWER.

Note that this outline only describes a very naive baseline. Step 2 refers the to sequence-to-sequence attention: when you compute the hidden state at each paragraph position, you take into account of all hidden states of the question. This step is to create a mixture of question and paragraph representation. Step 3 is another attention step, where you compare the last hidden state of question to all computed paragraph hidden states from Step 2. The attention vector you produce is similar to a "pointer" that points to the most significant paragraph hidden states.

If you prefer a more detailed guidance, consider reading MatchLSTM [2] as a good starting point. It is similar to the baseline we are describing here.

Consider This [Question and Context Paragraph representations]: Different ways of fusing the representations of the question and the context paragraph have been addressed in the literature. The most relevant are Dynamic Coattention Network (DCN) [3] and the Bilateral Multi-Perspective Matching [4]. Take key ideas from these and possibly other papers and use them to improve your encoder.

3.3 Evaluation

The original SQuAD paper introduces two metrics to evaluate the performance of a model: ExactMatch (EM) and F1 score. We use the same metrics to evaluate our model. Exact match is a metric that measures the percentage of predictions that match one of the ground truth answers exactly.

F1 score is a metric that loosely measures the average overlap between the prediction and ground truth answer. We treat the prediction and ground truth as bags of tokens, and compute their F1. We take the maximum F1 over all of the ground truth answers for a given question, and then average over all of the questions.

You might want to implement evaluation steps inside your training function. Pay special attention to `evaluate.py`, where it uses `f1_score()` and `exact_match_score()` to measure performances. We suggest that you evaluate your models performance per epoch.

When your model finishes training, take a look at `qa_answer.py`. This file is **model dependent**, so you **must** modify it in order for your model to run successfully. It takes in a JSON file like `dev-1.1.json` and outputs a JSON file with your prediction result. The generated JSON file can be consumed by `evaluate.py` to output a final F1 or Exact Match score for your model.

```
python code/evaluate.py data/squad/dev-v1.1.json dev-prediction.json
```

Note that having a program ready to consume a JSON file and output a prediction JSON file is the only way to submit to our leaderboard. We want you to report your F1 and EM score. The best models achieve around 80% F1 score and 75% EM score on test set, while the logistic regression baseline model scores 51% F1 and 40.4% EM on the test set. Finally, humans score over **90.5**% F1 and **86.8**% EM on the test set. You should get at least 60% on F1, and 50% on EM with your model. Your final grade will partly reflect whether you have achieved this goal or not. We will also evaluate your model design, experimentation decisions, and your write-up.

It is mandatory that you submit your model to `CodaLab`. We will have a leaderboard on both dev and test set.

4 Suggested readings

The following list is a set of papers with reading comprehension models that have achieved excellent results, many of them state-of-the-art in SQuAD. You are encouraged to take key ideas from these papers and use them to improve your own model.

- Multi-Perspective Matching [4]: https://arxiv.org/abs/1612.04211
- Dynamic Coattention Networks [3]: https://arxiv.org/pdf/1611.01604.pdf
- Match-LSTM with Answer-Pointer [2]: https://arxiv.org/abs/1608.07905
- Stanford Attentive Reader [5]: https://arxiv.org/pdf/1606.02858v2.pdf
- Bi-directional Attention Flow, BiDAF [6]: https://arxiv.org/pdf/1611.01603.pdf
- Recurrent Span Representation, RaSoR [7]: https://arxiv.org/pdf/1611.01436.pdf
- ReasoNet [8]: https://arxiv.org/pdf/1609.05284v1.pdf
- Dynamic Chunk Reader [9]: https://arxiv.org/pdf/1610.09996.pdf
- Fine-grained Gating [10]: https://arxiv.org/pdf/1611.01724.pdf

5 Practical Tips

- The model might take a long time to train depending on your specific implementations. We have experienced 1–1.5 hr per epoch on GPU with batch-size 32. Our bigger models take 2–3 hr per epoch. A full 10 epoch training would take at least 10 hours. Do not wait until the last minute to start training. A good rule is to try to have a reasonable model trained and ready to be evaluated and tested 7 days before the deadline so you can tune, improve and analyze it.
- If your model is not giving the best result, consider creating an ensemble. Often, a model ensemble will give you 3–5% performance bump.
- If you keep observing close to 0–1% performance on F1 or EM, your model might have a bug, and creating an ensemble won't help you.
- During preprocessing, you should have noticed that paragraphs have different length. After you follow the suggestion to plot out the distribution of paragraph length, you can use a threshold to cut off paragraphs that are too long. This might be able to help your model perform better.

- We are not evaluating your model on how well you can hand-craft advanced modules, so feel free to use Keras, TFLearn, or more advanced TensorFlow components. At this stage, you should not be afraid of using very advanced modules to solve this complex problem. However, we do not provide support for these libraries.
- A well-implemented model should have cost go down drastically during the first epoch. If you don't observe cost go down for an entire epoch, your model is not learning.
- Use `tf.global_norm(gradients)` to compute the norm of your gradient, and print out during your training. It signals you how much your model is learning. If the gradient is close to 0, your model isn't learning anything.

6 Deliverables and Grading

We are treating this assignment just like a final project. We will evaluate your work based on **three required deliverables**:

- An implemented runnable model.
- A paper where you propose your unique and creative solution to this task:
  - You need to introduce and present your model, explaining why different computations and components are chosen. Any lack of consideration over your model component choice will result in a deduction in your overall score.
  - You need to provide your F1 and EM score on dev set in your paper's experiment section.
  - You need to provide analysis for your prediction result on `dev-v1.1.json`. Analysis explains why your model is successful or not. What types of question are the easiest for your model? What are the hardest? Does some of the component in your model help on increasing accuracies on these types of questions? A careful analysis will also give you a good "future direction" section. We will reward good analysis in our final grade calculation.
  - For successful analysis, you should also visualize your code. Why did your model pick this position for the start of your answer span? Did it pick between several likely choices or is it able to place high confidence on it? We will reward good visualization in our final grade calculation as well.
- A submission of your model to `CodaLab` leaderboard, where if you achieve high performance on a hidden test set, it will be positively reflected on your grade.

6.1 Grading Criteria

The grade on Assignment 4 will be similar to the grade on a final project, where we value your input, creativity, and exploration the most. Following our recommendations to build a simple model is not enough. Some portion of your grade will be based on your leaderboard performance. We will assess if your implemented model is working based on your F1 and EM score, and we will reward teams that get competitive results.

6.2 Submission to Web

We will update the submission instructions as a separate file on Piazza. Stay tuned!

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

[2] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*, 2016.

[3] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*, 2016.

[4] Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. *arXiv preprint arXiv:1702.03814*, 2017.

[5] Danqi Chen, Jason Bolton, and Christopher D Manning. A thorough examination of the cnn/daily mail reading comprehension task. *arXiv preprint arXiv:1606.02858*, 2016.

[6] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[7] Kenton Lee, Tom Kwiatkowski, Ankur Parikh, and Dipanjan Das. Learning recurrent span representations for extractive question answering. *arXiv preprint arXiv:1611.01436*, 2016.

[8] Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. *arXiv preprint arXiv:1609.05284*, 2016.

[9] Yang Yu, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. End-to-end answer chunk extraction and ranking for reading comprehension. *arXiv preprint arXiv:1610.09996*, 2016.

[10] Zhilin Yang, Bhuwan Dhingra, Ye Yuan, Junjie Hu, William W Cohen, and Ruslan Salakhutdinov. Words or characters? fine-grained gating for reading comprehension. *arXiv preprint arXiv:1611.01724*, 2016.