CS378: Natural Language Processing Final Project

Proposal Due Date (for independent projects ONLY): Tuesday, March 31 at 11:59pm CST
Check-In Due Date (for all projects): Friday, May 1 at 11:59pm CST (no slip days)
Final Report Due Date: Wednesday, May 13 at 11:59pm CST (no slip days)

Collaboration You are free to work on this project in teams of two (encouraged) or individually. Individual projects can be less ambitious but should not be less complete: a half-implemented system does not make a good project outcome. All partners should contribute equally to the submission, and all partners will receive the same grade for it. You are also free to discuss your project with others in the course, though only the people on your team should contribute to the actual implementation/experimentation involved. Any external resources used must be clearly cited.

If you're pursuing an independent project, you **may** collaborate with someone from outside the course. You also **may** use an independent project as a final project for another course as well, **provided you have the approval of the other professor**.

Assignment

You have two options for your final project:

1. Question answering
2. An independent project of your choosing

1 Question Answering

The standard final project is to work on modifying and extending a neural question answering system as discussed in lecture. You are given three datasets as part of the MRQA Shared Task (Fisch et al., 2019): SQuAD (Rajpurkar et al., 2016), NewsQA (Trischler et al., 2017), and BioASQ (Tsatsaronis et al., 2015). SQuAD asks questions about Wikipedia articles, NewsQA about news articles, and BioASQ about biomedical text. While large neural network models based on BERT and following pre-training techniques have maxed out performance on the SQuAD dataset,[1] these systems do not generalize well to other datasets (Talmor and Berant, 2019).

Your goal in this project is to experiment with some improvement to question answering that may allow it to work better **either in the the standard SQuAD setting or in these other cross-domain settings**. We've divided these possibilities into three "tracks", described after we orient you with the code. **It's up to you to choose which datasets and which evaluation (in-domain or cross-domain) you wish to focus on for your project.**

1.1 Getting Started

Model We've given you a basic model based most closely on the reader from the DrQA paper (Chen et al., 2017). The model encodes the passage into a set of vectors $\mathbf{p}_{1,\ldots,n}$ using a biLSTM; imagine this as similar to the LSTM encoders from assignment 4. The model encodes the question into a fixed vector $\mathbf{q}$.[2] We then

[1] `https://rajpurkar.github.io/SQuAD-explorer/`

[2] The actual question and passage encoders are a bit more complicated than this. In particular, we use the *Aligned Question Embedding* and *Question Encoding* modules from the DrQA paper to improve the encoding of each of these.

compute logits for the start pointer $\text{start}(i) = \mathbf{p}_i^\top W^{\text{start}} \mathbf{q}$. End pointer logits are computed analogously with a different matrix W^{end} instead of W^{start}.

Note that the model outputs these start and end logits. In `utils.py`, the function `search_span_endpoints` actually extracts a span, used in `main.py` to get the answer.

Code Follow the instructions in the README to download the code and the data (using `setup.sh`, which downloads everything from publicly-available sources). Pre-trained versions of the model with several settings of hyperparameters are available as downloads through Box as well.

The model can be run in several modes. Using `main.py`, you can either `--do_train` or `--do_test` to either train a model (but not evaluate it), test a model (by loading parameters from either one of our pre-trained models or one you trained previously). Then you can use `evaluate.py` to compute exact match and F_1 metrics for evaluation.

1.2 Track 1: Linguistic Constraints

Neural QA systems often make somewhat mysterious errors. They can be distracted by answers that don't apparently have anything to do with the question but match some shallow surface characteristics like having words that overlap with the question. They can also fixate too much on finding an answer of the right type (e.g., returning a location for a *where* question). Finally, large neural models training data, meaning that they may handle unknown words poorly or fail to generalize in other ways.

Many of these errors are orthogonal to the type of errors made by "classic" QA techniques like sliding windows of n-grams (Richardson et al., 2013). Tools like NER and dependency parsers can also help analyze the structure of the question and the document and possibly allow you to take steps to identify candidate answers or rule out bad answers. For example, if a document span has no relevance to a named entity from the question, it might be unlikely to be correct. **Such constraints are likely to be particularly useful in cross-domain settings.**

Tools you might explore include a **NER system**, a **dependency parser**, or a **constituency parser**. spaCy[3] is probably easiest toolkit to get working. Stanford's suite of tools is fairly comprehensive: Stanza[4] is a very of these in Python, and Stanford CoreNLP works pretty well but is Java-based.

Given these tools, you might explore one of the following questions:

- What can n-gram based techniques like sliding windows do?
- Can NER help identify parts of the passage relevant to named entities?
- Can parsers help identify parts of the passage relevant to the question?

You might consider several approaches. (1) Use these linguistic annotations to focus on one part of the passage, then run the neural model on that part only. (2) Use the default neural model, but modify `search_span_endpoints` to search over spans in a different way. For example, you could heuristically rescore spans based on linguistic features, prohibit certain spans entirely, etc.

1.3 Track 2: Model and Neural Architecture

You can also try to make some improvements to the neural architecture of the basic system. **These improvements should be well-motivated!** Don't just add LSTM layers or tweak the architecture to try to get better

[3] `http://spacy.io/`

[4] `https://stanfordnlp.github.io/stanza/`

performance. If you have some justification for what you're doing besides "it makes the neural network bigger", then you should feel free to explore it. Check with the course staff if you're unsure.

Here are a few questions you could explore:

- Integrate some extra component into the model that helps either in-domain or cross-domain. This could overlap with ideas from Track 1 above. Typically, you should be thinking about how to better capture interactions between the question, the question and the context, etc. What kinds of layers might capture such interactions? What should their inputs and outputs be?
- Change something about the inputs, like trying a different representation for tokens. You shouldn't just randomly try other word embeddings, but exploring resources for biomedical-focused word embeddings could be a good way to improve performance on BioASQ, for example.
- Use a character-level component of the model to try to do better at recognizing rare words.
- If you want to explore ELMo or BERT and pre-trained techniques, you should feel free with the caveat that these approaches are **extremely slow to train**. You probably shouldn't dive into this unless you feel pretty confident you can handle them computationally, probably using either Google Cloud or your own GPU resources. BERT additionally operates over subword chunks called "word pieces," which you'll have to manage at evaluation time by mapping your spans to word-based spans.

You don't have to extensively comb the literature to look for past efforts to do what you're trying to do. However, if you do consult the literature, you should cite the papers you read.

One approach that is usually **not** particularly promising is tree-structured RNNs. While these show up some in the literature, they're very tricky to get working and usually don't help that much. Although they're a nice idea for integrating linguistic structure with neural models, we don't recommend them.

1.4 Track 3: Domain Adaptive Training

You can also explore what's possible if you want to optimize performance in one of the "other" domains (NewsQA or BioASQ), assuming you have access to a small amount of data in that domain in addition to the SQuAD training data. This would simulate the scenario where you've trained on the large SQuAD dataset and want to adapt it to another setting without having to annotate a large number of examples. You could start either from the pre-trained weights and explore adaptation approaches, or re-train your own model using other techniques. Note that for BioASQ, there's no training set, just a test set. IF you want to train on things, you can split this test set further into a 50/50 fine-tune and a test set.

A project focused on this will probably be more about data and the training procedure than about modifying the model directly. Here are a few questions you could explore:

- Change the data in some way: data augmentation, heuristic labeling of data in the target domain (Dhingra et al., 2018).
- Changing something about the domain-adaptive training regime. Here are some papers/concepts you might explore: unsupervised data augmentation (Xie et al., 2019) (this might be challenging) or self-ensembling (French et al., 2018; Desai et al., 2019). One approach that **doesn't** work well is the adaptation by domain-adversarial training (Ganin et al., 2016); it's usually quite hard to get this to work well for NLP tasks.

1.5 Scope

Along any of these lines, what you try does not strictly need to work. However, if it doesn't work, you should have a plan for how to analyze it and be able to dig into the data more. Just saying "I tried X and wrote 200 lines of code but it still crashes" is not a good project outcome. Try to make sure you're on track to have some preliminary results or analysis supporting what you're trying to do by the check-in. From there, make sure that even if things don't work, you can still argue (a) that you've correctly implemented what you set out to implement; (b) you can analyze the results to understand why things went wrong.

One person vs. two person projects If you're working in a group of two, we expect the work to scale up appropriately. We will generally have higher expectations for the number of things tried and the quality of the writeup and analysis.

As a single-person project: We expect that you've tried at least a minor modification to the system, conceptually justified what you've tried, demonstrated in some way that your code is correct, reported results, and given some analysis. Changing embeddings, simple filtering of possible answers, or making a small change to neural network architecture are examples of minor changes.

As a two-person team: We expect that you've tried a more major modification to the system (or two minor modifications), conceptually justified what you've tried, demonstrated in some way that your code is correct, reported results, and given some analysis. More significant changes to neural architecture or using linguistic structure in a sophisticated way to filter answer options are examples of more major modifications.

Compute and Feasibility See the end of the project spec for more discussion about this.

2 Your own project

Finally, you can pursue your own independently-chosen final project. Your proposal in this case should, as precisely as possible, describe the problem you want to solve or understand and the work you propose to do to solve it. Make sure you describe any datasets you plan to use and demonstrate that what you're proposing is feasible. Working on adjacent areas of machine learning (e.g., computer vision or robotics) is okay as long as there is a reasonable connection to concepts from this course. Ask the course staff if you're unsure.

Scope The expected scope should be similar to the QA project. In particular, you should either have a nontrivial piece of implementation (putting together existing pieces can be nontrivial) or analysis. The project should also engage with the course concepts. One example of an **inappropriate** project is scaling machine translation to run in a cloud setting: this may have some challenges and is probably a sufficient amount of work, but mostly involves engineering and concepts not related to linguistics, machine learning, or algorithms as discussed in class.

3 Deliverables and Grading

The final project is worth 25% of your course grade. The deliverables are as follows.

Check-In (10 points) You should turn in a check-in (at least a couple paragraphs, no more than 1 page) on check-in due date (a bit less than halfway through). This check-in should outline what you've looked into so far and what your plan is for the remainder of the project (e.g., "I/we want to investigate X, we

got the basic system running and looked through 10 data examples to confirm that our idea might work"). The course staff will then provide feedback and guidance on the direction to maximize the project's change of succeeding. The check-in is worth 10 points and is graded entirely on whether or not you provided a reasonably complete report on your progress so far, not the progress itself.

Code You should also submit any code you wrote on the project due date, but this is **for documentary purposes only**; we will not be attempting to verify your results by running the code. Please do not include large data files or external resources you used that might be needed to execute it.

Final Report The primary deliverable is a paper written in the style of an ACL[5]/NeurIPS/etc. conference submission.[6] It should begin with an abstract and introduction, clearly describe the proposed idea or exploration, present technical details, give results, compare to baselines, provide analysis and discussion of the results, and cite any sources you used.

This paper should be between 3 and 8 pages excluding references. Different projects may take different numbers of pages to describe, and it depends on whether you're working by yourself or in a group. If you have lots of analysis and discussion or are trying something more ambitious, your paper might be longer; if you're implementing something complex but succinctly described, your paper might be shorter.

Your project is *not* graded solely on the basis of results. You should approach the work in such a way that success isn't all-or-nothing. You should be able to show results, describe some successes, and analyze why things worked or didn't work beyond "my code errored out." Think about structuring your work in a few phases so that even if everything you set out to do isn't successful, you've at least gotten something working, run some experiments, and gotten some kind of results to report.

Grading: We will grade the projects according to the following rubric:

- **Scope (25 points)**: Is the idea of sufficient depth for a course project? Concretely, we expect that you do something along one of the three axes above and make a reasonable effort to execute it. While it does not have to work wonderfully, you will lose points here if all you can show is shallow analysis of the base system.
- **Implementation (25 points)**: Is the implementation described reasonable? Is the idea itself technically sound? You might lose points here if we perceive there to be a technical error in your approach. For example, perhaps you added a module to the neural network that leads to no performance change, but it's because it mathematically led to no change in the model due a conceptual error.
- **Results/Analysis (25 points)** Whether the results are positive or negative, try to motivate them by providing examples and analysis. If things worked, what types of errors are reduced? If things didn't work, why might that be? What aspects of the data/model might not be right? There are a few things you should report here: **Key results**: You should report results from a baseline approach as well as your "best" model. **Ablations**: If you tried several things, analyze the contribution from each one. These should be *minimal* changes to the same system; try running things with just one aspect different in order to assess how important that aspect is.

[5]Style files available here: `http://www.acl2019.org/EN/call-for-papers.xhtml`

[6]The Iyyer et al. paper is a good example of this: `https://people.cs.umass.edu/~miyyer/pubs/2015_acl_dan.pdf`

- **Clarity/Writing (15 points)**: Your paper should clearly convey a core idea/hypothesis, describe how you tested it/what you built, and situate it with respect to related work as best you can. **Abstract and Introduction**: Did you provide a clear summary of the motivation, methodology, and results? **Method**: Is the presentation of what was done clear? **Results**: Is the work experimentally evaluated? Are there clear graphs and tables to communicate the results? Don't just inline 1-2 numbers; make your analysis more detailed than that.

Compute and Feasibility **Large neural net methods can be slow to train!** Training a model on even 10,000 QA examples can take hours. Keep this in mind when deciding what kind of project to do. Working on linguistic constraints is likely to be the least resource-intensive, whereas developing a new neural architecture is likely to be the most.

Google Cloud Platform offers free credits upon signing up for a new account, which are likely sufficient to run some large-scale experiments for the course. The course staff are able to provide limited support on how to use GCP, but you'll mostly have to figure this out yourself.

Extra Credit: You can earn 2 points of extra credit by filling out the eCIS course evaluation. **Take a screenshot of the page showing that you've completed the survey (not your response itself, which is confidential) and upload it along with your final project.**

COVID-19: If you believe that this project will pose a logistical challenge for you due to circumstances related to COVID-19, please get in touch with the course staff immediately.

References

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to Answer Open-Domain Questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Shrey Desai, Barea Sinno, Alex Rosenfeld, and Junyi Jessy Li. 2019. Adaptive Ensembling: Unsupervised Domain Adaptation for Political Document Analysis. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Bhuwan Dhingra, Danish Danish, and Dheeraj Rajagopal. 2018. Simple and Effective Semi-Supervised Question Answering. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Adam Fisch, Alon Talmor, Robin Jia, Minjoon Seo, Eunsol Choi, and Danqi Chen. 2019. MRQA 2019 Shared Task: Evaluating Generalization in Reading Comprehension. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*.

Geoff French, Michal Mackiewicz, and Mark Fisher Fisher. 2018. Self-ensembling for visual domain adaptation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-Adversarial Training of Neural Networks. *J. Mach. Learn. Res.*, 17(1):2096–2030, January.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Matthew Richardson, Christopher J.C. Burges, and Erin Renshaw. 2013. MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Alon Talmor and Jonathan Berant. 2019. MultiQA: An Empirical Investigation of Generalization and Transfer in Reading Comprehension. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4911–4921, Florence, Italy, July. Association for Computational Linguistics.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2017. NewsQA: A Machine Comprehension Dataset. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*.

George Tsatsaronis, Georgios Balikas, Prodromos Malakasiotis, Ioannis Partalas, Matthias Zschunke, Michael R. Alvers, Dirk Weissenborn, Anastasia Krithara, Sergios Petridis, Dimitris Polychronopoulos, Yannis Almirantis, John Pavlopoulos, Nicolas Baskiotis, Patrick Gallinari, Thierry Artiéres, Axel-Cyrille Ngonga Ngomo, Norman Heino, Eric Gaussier, Liliana Barrio-Alvers, Michael Schroeder, Ion Androutsopoulos, and Georgios Paliouras. 2015. An overview of the BIOASQ large-scale biomedical semantic indexing and question answering competition. In *BMC Bioinformatics*.

Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. 2019. Unsupervised Data Augmentation for Consistency Training. In *arXiv*.