# Final Project (Assignment 6): Question Answering

**Academic Honesty:** Please see the course syllabus for information about collaboration in this course. **For this project, you may work with a partner if you'd like. Please list your names on your report and submit your reports individually.**

**Goal:** In this project you'll experiment with a neural question answering system, investigating its strengths and weaknesses in other domains.

## 1 Question Answering

You will work on modifying and extending a neural question answering system. You are given three main 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).

Furthermore, you are also given the adversarial SQuAD data from Jia and Liang (2017). This data adds sentences which look like the question but contain nonsense entities, modifications which easily fool many SQuAD-trained models.

Your goal in this project is to experiment with some improvement to question answering that may allow it to work better in one of these cross-domain or adversarial 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 to focus on for your project.**

### Getting Started

The code is available in a GitHub link provided with this PDF. The repository README contains instructions for forking the repository, downloading the datasets, and running the code.

**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 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 $W^{\text{end}}$ instead of $W^{\text{start}}$.

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). A pre-trained version of the model is available to download as well. The pre-trained model provided here uses hyperparameters that give a good balance of speed and accuracy.

[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, as discussed in class.

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.

Your goal is to make some sort of improvement over the baseline model, usually around 1-2% if possible. There are two main ways you can go about doing this.

### 1.1 Track 1: Linguistic Constraints for Robustness

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 (particularly in the adversarial case) 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 can memorize 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 most useful in adversarial settings, so you may wish to focus on that dataset.**

Tools you might explore include an **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 nice set 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.

### 1.2 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 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:

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

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

- 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 operates over subword chunks, 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.

### 1.3 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.

### 1.4 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. Try to make sure you're on track to have some preliminary results or analysis supporting what you're trying to do. 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.

Your modification of the system doesn't have to be major. In general, you shouldn't be overly ambitious: it's better to have a small change that you successfully implemented and analyzed in detail rather than a big change that didn't work.

**Single vs. Two-Person Projects** **A two-person project is expected to be more sophisticated than a one-person project!** One way to approach this is to explore two minor modifications, perhaps working together on coordinated changes that could each stand alone as a one-person project. Or, you can bite off a larger single modification, or do more experimentation or analysis around a modification that you're trying out.

**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 to attempt. Working on linguistic constraints is likely to be the least resource-intensive, whereas developing a new neural architecture is likely to be the most.

If you don't have other access to a GPU, 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. See the GitHub README for information.

## 2 Deliverables and Grading

Unlike past assignments, this assignment will be graded primarily on the basis of a written report.

**Code** Unlike previous assignments, you are free to modify *any* of the files that you are given here. Your code submission should consist of the code and any files you modify. **Do not include any additional data you might've used.** This code is purely documentary: it may be inspected to assess what you've done, but we want to give you freedom to use tools that might not be available in our environment.

**System Output** In lieu of running your source code, we expect you to include **your model's output on any relevant datasets**.

**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 6 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

[5]Style files available here: `https://2021.aclweb.org/calls/papers/#paper-submission-and-templates`

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

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.

**Peer assessment phase** You will submit your report on edX where it will go through a peer assessment phase; this means you will be "grading" someone else's report. These grades will be considered as one input in the process, but all final grading decisions will be made by the course staff. Your reports will be assessed by your peers and the course staff using the following rubric:

- **Scope (25 points)**: Is the idea of sufficient depth for a course project? Concretely, we expect something along one of the three axes above and a reasonable effort to execute it. While it does not have to work wonderfully, there should be something implemented beyond the base system and some analysis of that.
- **Implementation (30 points)**: Is the implementation described reasonable? Is the idea itself technically sound, or are there errors in the approach? Typically points are only deducted here if there are clear issues evident from the report. If the idea itself is sound and probably implemented correctly, the implementation should not be penalized.
- **Results/Analysis (30 points)** How well done is the evaluation? There should be a few parts: key results (comparison of the best model to a baseline), ablations (studying the contribution of each component of the approach), and additional quantitative or qualitative analysis of examples. Note that this is not assessed entirely on the basis of empirical results. A method that works poorly but which is analyzed and evaluated well may do better than a method that works okay but works great. However, absolutely quality of results should be considered, and the highest grades are only awarded in cases of very strong results.
- **Clarity/Writing (15 points)**: The paper should clearly convey a core idea/hypothesis, describe how it was tested or what was built, and situated with respect to any related work that may have been referenced. In the abstract and introduction: were the motivation, methodology, and results summarized appropriately? Method: is the presentation of the methodology clear? Is it clear what was done? Results: are the results clearly presented in such a way that the work is understandable, whether it works well or not?

## Submission and Grading

You should upload your report on **edX** and your report and code on **Canvas**. Please upload **two files to Canvas**: a PDF of your report, and tgz containing (1) any code files you modified and (2) whatever relevant system outputs you have (use your discretion).

**edX On edX, all students should submit the project individually, listing both names on the project PDF.** Therefore, group projects will receive two grades from other students, while individual projects will receive one grade. These grades are one input we will consider when assigning the final grade, but all final grade decisions will be assigned by the course staff and any borderline cases will be reviewed closely. Receiving two grades vs. one grade from the peer feedback round will have no effect on your final grade in the class.

**Canvas** Please submit the code and output files on Canvas. **This code upload is purely documentary; we will not attempt to reproduce results from what you did.** As a result, **do not upload any data or models to Canvas**, as these can make the submissions very large and hard to manage.

## 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.

Robin Jia and Percy Liang. 2017. Adversarial Examples for Evaluating Reading Comprehension Systems. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

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*.