CS 224N Default Final Project: Building a QA system (Robust QA track)

Last updated on February 1, 2022

Contents

1 Overview

1.1 Motivation

Over the last few years, we have seen tremendous progress on fundamental natural language understanding problems. At the same time, there is increasing evidence that models learn superficial correlations that fail to generalize beyond the training distribution [1, 2, 3, 4]. On the other hand, humans can easily generalize beyond their training distribution–while not strictly from our training distribution, we can effortlessly understand novels set in fictional worlds and quickly understand the meanings of new words. How can we build NLP systems that generalize like humans? From a practical perspective, robustness to out-of-distribution data is critical for building accurate NLP systems in the real world since train and test data often come from distinct user interactions.

In this project, you will be building a question answering system that can adapt to unseen domains with only a few training samples from the domain. This will expose students to a more real world scenario where test examples are rarely IID with the training data. To build such a system, you are given three separate question answering datasets. In addition to the Stanford Question Answering Dataset (SQuAD) [5], you are also given Natural Questions [6] and NewsQA [7], preprocessed in the same format as SQuAD. At test time, you will be given examples from unseen question answering datasets, along with a small training set of 128 examples for additional finetuning.

This year, the default final project consists of two tracks. In the IID SQuAD track, you will be building a QA system for the SQuAD dataset, and in the Robust QA track, you will be building a QA system that is robust to domain shifts. Note that for the IID SQuAD track, you are not allowed to use pre-trained transformer models, and for the RobustQA track you are allowed to use only DistilBERT [8] as the pre-trained transformer model. Please also keep in mind that there will be more help/guidance available for the IID SQuAD track than the Robust QA track.

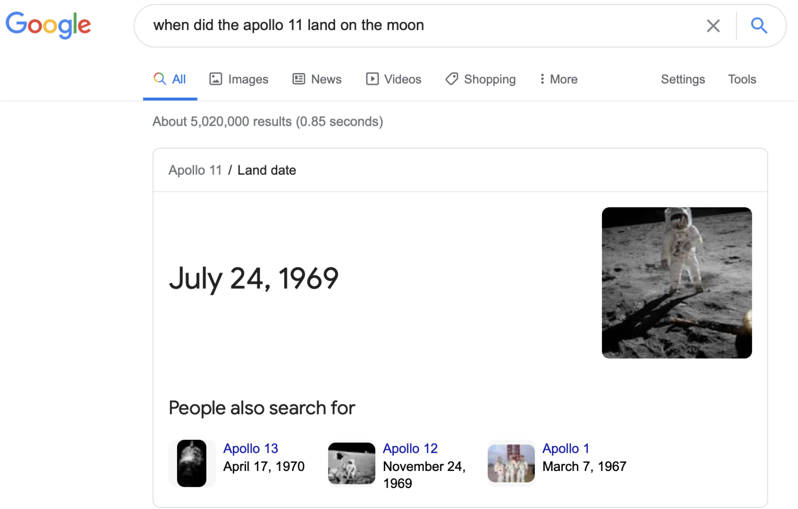
Note on default project vs custom project: The effort/work/difficulty that goes into the default final project is not intended to be less compared to the custom project. It is just that the specific kind of difficulty around coming up with your own problem and evaluation methods was intended to be excluded, allowing students to focus an equivalent amount of effort on this provided problem.

1.2 Question Answering

In the task of reading comprehension or question answering, a model will be given a paragraph, and a question about that paragraph, as input. The goal is to answer the question correctly. From a research perspective, this is an interesting task because it provides a measure for how well systems can 'understand' text. From a more practical perspective, these systems (Figure 1) have been extremely useful for better understanding any piece of text, and serving information need of humans.

As an example, consider the SQuAD dataset. The paragraphs in SQuAD are from Wikipedia. The questions and answers were crowdsourced using Amazon Mechanical Turk. There are around 150k questions in total, and *roughly half of the questions cannot be answered using the provided paragraph* (this is new for SQuAD 2.0). However, if the question *is* answerable, the answer is a chunk of text taken directly from the paragraph. This means that SQuAD systems don't have to *generate* the answer text – they just have to *select* the span of text in the paragraph that answers the question (imagine your model has a highlighter and needs to highlight the answer). Below is an example of a ⟨question, context, answer⟩ triple. To see more examples, you can explore the dataset on the website `https://rajpurkar.github.io/SQuAD-explorer/explore/v2.0/dev/`.

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



when did the apollo 11 land on the moon

About 5,020,000 results (0.85 seconds)

Apollo 11 / Land date

July 24, 1969

People also search for

Apollo 13 April 17, 1970

Apollo 12 November 24, 1969

Apollo 1 March 7, 1967

Figure 1: Google's question answering system is able to answer arbitrary questions and is an extremely useful tool for serving information needs

In fact, in the official validation and test set, every answerable SQuAD question has *three answers* provided – each answer from a different crowd worker. The answers don't always completely agree, which is partly why 'human performance' on the SQuAD leaderboard is not 100%. Performance is measured via two metrics: **Exact Match (EM)** score and **F1** score.

- **Exact Match** is a binary measure (i.e. true/false) of whether the system output matches the ground truth answer exactly. For example, if your system answered a question with 'Einstein' but the ground truth answer was 'Albert Einstein', then you would get an EM score of 0 for that example. This is a fairly strict metric!
- **F1** is a less strict metric – it is the harmonic mean of precision and recall[1]. In the 'Einstein' example, the system would have 100% precision (its answer is a subset of the ground truth answer) and 50% recall (it only included one out of the two words in the ground truth output), thus a F1 score of $2 \times \text{prediction} \times \text{recall}/(\text{precision} + \text{recall}) = 2*50*100/(100+50) = 66.67\%$.
- When evaluating on the validation or test sets, we take the *maximum* F1 and EM scores across the three human-provided answers for that question. This makes evaluation more forgiving – for example, if one of the human annotators *did* answer 'Einstein', then your system will get 100% EM and 100% F1 for that example.

Finally, the EM and F1 scores are averaged across the entire evaluation dataset to get the final reported scores.

1.3 This project

As mentioned earlier, you will be building a question answering system that works well on out-of-domain datasets. We have provided code for preprocessing the data and computing the evaluation metrics, and code to train a fully-functional neural baseline. Your job is to improve on this baseline.

In Section 5, we describe several models and techniques that are commonly used in building fewshot systems – most come from recent research papers. We provide these suggestions to help you get started implementing better models. Note that this is a fairly new field, and so these suggestions may not all lead to improvements over the baseline.

Though you're not required to implement something original, the best projects will pursue some originality with improvements over the baseline. Originality doesn't necessarily have to be a completely new approach – small but well-motivated changes to existing models are very valuable, especially if followed by good analysis. If you can show quantitatively and qualitatively that your small but original change improves a state-of-the-art model (and even better, explain what particular problem it solves and how), then you will have done extremely well.

[1] Read more about F1 here: https://en.wikipedia.org/wiki/F1_score

Like the custom final project, the default final project is open-ended – it will be up to you to figure out what to do. In many cases there won't be one correct answer for how to do something – it will take experimentation to determine which way is best. We are expecting you to exercise the judgment and intuition that you've gained from the class so far to build your models.

For more information on grading criteria, see Section 7.

2 Getting Started

For this project, you will need a machine with GPUs to train your models efficiently. For this, you have access to Azure, similarly to Assignments 4 and 5 – remember you can refer to the *Azure Guide* and *Practical Guide to VMs* linked on the class webpage. As before, remember that Azure credit is charged for every minute that your VM is on, so it's important that your VM is only turned on when you are actually training your models.

We advise that you **develop your code on your local machine** (or one of the Stanford machines, like `rice`), using PyTorch without GPUs, and move to your Azure VM only once you've debugged your code and you're ready to train. We advise that you use GitHub to manage your codebase and sync it between the two machines (and between team members) – the *Practical Guide to VMs* has more information on this.

When you work through this *Getting Started* section for the first time, do so on your local machine. You will then repeat the process on your Azure VM. Once you are on an appropriate machine, clone the project Github repository with the following command.

```
git clone https://github.com/michiyasunaga/robustqa.git
```

This repository contains the starter code as well as the datasets that we will be using. We encourage you to `git clone` our repository, rather than simply downloading it, so that you can easily integrate any bug fixes that we make to the code. In fact, you should periodically check whether there are any new fixes that you need to download. To do so, navigate to the `robustqa` directory and run the `git pull` command.

Note: If you use GitHub to manage your code, you must keep your repository private.

2.1 Code overview

The repository `robustqa` contains the following files:

- `args.py`: Command-line arguments for `train.py`.
- `environment.yml`: List of packages in the `conda` virtual environment.
- `train.py`: Top-level entrypoint for training as well as performing inference.
- `util.py`: Utility functions and classes.

In addition, you will get two directories when you follow the README of the Github repo:

- `dataset/`: Contains the datasets organized in separate folders (`indomain_train`, `indomain_val`, `oodomain_train`, `oodomain_val`, `oodomain_test`). Within each folder, we have separate JSON files for each dataset.
- `save/`: Location for saving all checkpoints and logs. For example, if you train the baseline with `python train.py --run-name baseline --do-train`, then the logs, checkpoints, and TensorBoard events will be saved in `save/baseline-01`. The suffix number will increment if you train another model with the same name.

2.2 Setup

Once you are on an appropriate machine and have cloned the project repository, it's time to run the setup commands.

- Make sure you have Anaconda or Miniconda installed.
- `cd` into `robustqa` and run `conda env create -f environment.yml`
  - This creates a conda environment called `robustqa`.
- Run `source activate robustqa`
  - This activates the `robustqa` environment.

  - **Note:** Remember to do this each time you work on your code.

- Download datasets from the link provided in the README and unzip them with `tar -xvzf datasets_50k.tar.gz`

- *(Optional)* If you would like to use PyCharm, select the `robustqa` environment. Example instructions for Mac OS X:

  - Open the `robustqa` directory in PyCharm.
  - Go to `PyCharm > Preferences > Project > Project interpreter`.
  - Click the gear in the top-right corner, then `Add`.
  - Select `Conda environment > Existing environment > Click '...' on the right`.
  - Select `/Users/YOUR_USERNAME/miniconda3/envs/robustqa/bin/python`.
  - Select `OK` then `Apply`.

Once you have unzipped `datasets_50k.tar.gz`, you should now see many additional folders in `save/` (see Section 3 for more details):

- `indomain_train/{newsqa,squad,nat_questions}`: These are JSON files corresponding to the in-domain training data.

- `indomain_val/{newsqa,squad,nat_questions}`: These are JSON corresponding to in-domain validation data.

- `oodomain_train/{race,relation_extraction,duorc}`: Training data for the out-of-domain datasets. These may be used for additional finetuning.

- `oodomain_val/{race,relation_extraction,duorc}`: Validation data for the out-of-domain datasets. These may be used for setting various hyperparameters.

- `oodomain_test/{race,relation_extraction,duorc}`: These correspond to the test set on which your system will be evaluated.

If you see all of these files, then you're ready to get started training the baseline model (see Section 4.3)!

3 Training Datasets

3.1 Data splits

Dataset	**Question Source**	**Passage Source**	**Train**	**dev**	**Test**
		`in-domain datasets`			
SQuAD [5]	Crowdsourced	Wikipedia	50000	10,507	-
NewsQA [7]	Crowdsourced	News articles	50000	4,212	-
Natural Questions [6]	Search logs	Wikipedia	50000	12,836	-
		`oo-domain datasets`			
DuoRC [9]	Crowdsourced	Movie reviews	127	126	1248
RACE [10]	Teachers	Examinations	127	128	419
RelationExtraction [11]	Synthetic	Wikipedia	127	128	2693

Table 1: Statistics for datasets used for building the QA system for this project. **Question Source** and **Passage Source** refer to data sources from which the questions and passages were obtained. Table borrowed from [12]

In this project, you are provided with three *in-domain* reading comprehension datasets (Natural Questions, NewsQA and SQuAD) for training a QA system which will be evaluated on test examples from three different *out-of-domain* datasets (RelationExtraction [11], DuoRC [9], RACE [10]). For all of the three training datasets provided, we have an `indomain_train` set as well as an `indomain_val` set. In addition to this, we also provide an `oodomain_train` set consisting of a small number of examples for all the out-of-domain datasets for additional training, along with a validation set (`oodomain_val`). Taken together, we refer to the union of all the `indomain_train` and `oodomain_train` sets as simply the train set, and the union of the `indomain_val` and `oodomain_val` sets as the validation set.

Your QA system will be evaluated on a held out test set (`oodomain_test`) from the eval datasets. For simplicity and scalability, we are running this track in a 'Kaggle-style' leaderboard, i.e., we release test sets (context, question) for each of the eval datasets to students, and they submit their model-produced answers in a CSV file. We then compare these CSV files to the true test set answers and report scores in a leaderboard. You may *only* use our training and validation set to train, tune and evaluate your models. This means both labeled as well as unlabeled data (e.g., unlabeled data from other sources such as Wikipedia may not be used). Dataset statistics are in Table 1. If you wish to use any additional data and have a good reason to do so, please create a private post on Ed and check with the TAs

You will use the train set to train your model and the validation set to tune hyperparameters and measure progress locally. Finally, you will submit your test set solutions to a class leaderboard, which will calculate and display your scores on the test set – see Section 6 for more information.

3.2 Terminology

The training set contains many (context, question, answer) triples[2] – see an example in Section 1.2. Each *context* (sometimes called a *passage*, *paragraph* or *document* in other papers) is an excerpt from Wikipedia. The *question* (sometimes called a *query* in other papers) is the question to be answered based on the context. The *answer* is a span (i.e. excerpt of text) from the context.

[2] As described in Section 1.2, the dev and test sets actually have *three* human-provided answers for each question. But the training set only has one answer per question.

4 Baseline

As a starting point, we have provided you with the complete code for a baseline model, which finetunes a BERT based pre-trained transformer. In this section we will describe the baseline model and show you how to train it.

4.1 Processing and chunking

We processs all datasets in the same format as SQuAD. In the directory, we have jsons corresponding to the training data.

Chunking: Since the maximum context size that can be encoded by BERT is 512, we convert each (question, paragraph) into multiple chunks of size 384 with a stride of 128. To understand this procedure, consider the following example. Let $(q,\ p)$ be a question, paragraph pair where $q = \{q_0, q_1, ...q_{10}\}$ and $p = \{p_0, p_1, ..., p_{500}\}$. We convert this into chunks c_1 and c_2 where $c_1 = \texttt{[CLS]}\,q\,\texttt{[SEP]}\,p^1\,\texttt{[SEP]}$ with $p^1 = \{p_0, p_1, \ldots, p_{371}\}$ and the $c_2 = \texttt{[CLS]}\,q\,\texttt{[SEP]}\,p^2\,\texttt{[SEP]}$ with $p^2 = \{p_{128}, p_{129}, \ldots, p_{499}\}$. Each chunk is then labeled with a start position and end position based on its offset. For chunks that do not contain the answer span we set the start and end positions as (0, 0)

Caching: Since the tokenizenation and preprocessing steps are time consuming and slow for very large datasets, we cache the tokenized representations of the data once they are processed. If you would like to create these from scratch, make sure to run `train.py` with the flag `create-features`. An easy to miss bug is changing the preprocessing functions and failing to recompute cached features.

4.2 Baseline Model

The baseline system finetunes DistilBERT (a smaller, *distilled* version of the original BERT model [8]) on all the training data.

Loss Function

Our loss function is the sum of the negative log-likelihood (cross-entropy) loss for the start and end locations. That is, if the gold start and end locations are $i \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, N\}$ respectively, then the loss for a single example is:

$$\text{loss} = -\log \boldsymbol{p}_{\text{start}}(i) - \log \boldsymbol{p}_{\text{end}}(j)$$

During training, we average across the batch and use the AdamW optimizer [13] to minimize the loss.

Inference Details

Given a question paragraph pair (q,p), we first convert the pair into multiple chunks $\{c_1, c_2, \ldots, c_k\}$ following our chunking procedure. Then, we pass c_i through our model to get corresponding start and end logits. Then, we select the logits with the highest sum. Concretely, we choose the pair (i, j) of indices that maximizes $\boldsymbol{p}_{\text{start}}(i) \cdot \boldsymbol{p}_{\text{end}}(j)$ subject to $i \le j$ and $j - i + 1 \le L_{\max}$, where $L_{\max}$ is a hyperparameter which sets the maximum length of a predicted answer. We set $L_{\max}$ to 15 by default.

4.3 Train the baseline

Before starting to train the baseline on your VM, consider opening a new session with `tmux` or some other session manager. This will make it easier for you to leave your model training for a long time, then retrieve the session later. For more information about TMUX, see the *Practical tips for final projects* document.

To start training the baseline, run the following commands:

```
source activate robustqa        # Activate the robustqa environment
python train.py --do-train --run-name baseline  # Start training
```

After some initialization, you should see the model begin to log information like the following:

```
19\%|###             | 45344/242304 [30:59<1:42:14, 32.23it/, NLL=1.45, epoch=1]
```

You should see the loss – shown as `NLL` for negative log-likelihood – begin to drop. On a single Azure NC6 instance, you should expect training to take about 3-4 hours per epoch.

You should also see that there is a new directory under `save/baseline-01`. This is where you can find all data relating to this experiment. In particular, you will (eventually) see:

- `log_train.txt`: A record of all information logged during training. This includes a complete print-out of the arguments at the very top, which can be useful when trying to reproduce results.
- `events.out.tfevents.*`: These files contain information (like the loss over time), which our code has logged so it can be visualized by TensorBoard.
- `checkpoint/`: The best checkpoint throughout training. The metric used to determine which checkpoint is 'F1' score. Typically you will load this checkpoint for use later.

4.4 Tracking progress in TensorBoard

We strongly encourage you to use TensorBoard, as it will enable you to get a much better view of your experiments. To use TensorBoard, run the following command from the `squad` directory:

```
tensorboard --logdir save --port 5678   # Start TensorBoard
```

If you are training on your local machine, now open http://localhost:5678/ in your browser. If you are training on a remote machine (e.g. Azure), then run the following command on your local machine:

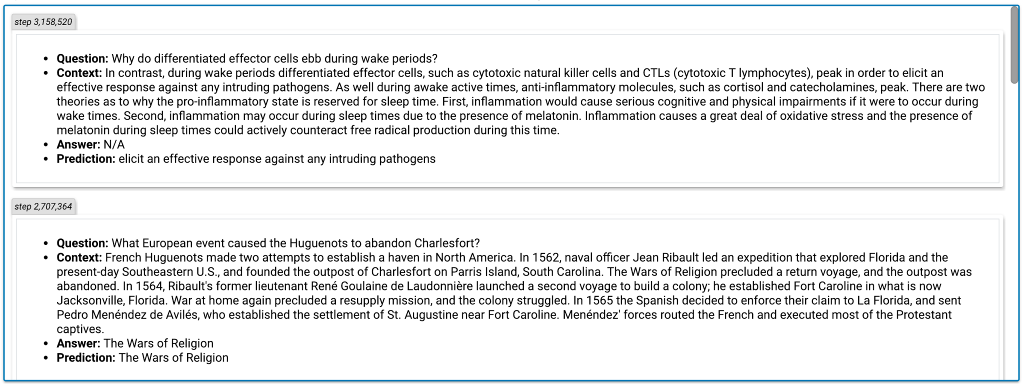
```
ssh -N -f -L localhost:1234:localhost:5678 <user>@<remote>
```

where `<user>@<remote>` is the address that you `ssh` to for your remote machine. Then on your local machine, open http://localhost:1234/ in your browser.

You should see TensorBoard load with plots of the loss, EM, and F1 on the validation set. EM and F1 are the official SQuAD evaluation metrics. The validation plots may take some time to appear because they are logged less frequently than the train plots. However, you should see training set loss decreasing from the very start.

4.5 Inspecting Output

During training you will also notice a tab in TensorBoard labeled `Text`. Try clicking on this tab and you should see output similar to the following:



step 3,158,520

- **Question:** Why do differentiated effector cells ebb during wake periods?
- **Context:** In contrast, during wake periods differentiated effector cells, such as cytotoxic natural killer cells and CTLs (cytotoxic T lymphocytes), peak in order to elicit an effective response against any intruding pathogens. As well during awake active times, anti-inflammatory molecules, such as cortisol and catecholamines, peak. There are two theories as to why the pro-inflammatory state is reserved for sleep time. First, inflammation would cause serious cognitive and physical impairments if it were to occur during wake times. Second, inflammation may occur during sleep times due to the presence of melatonin. Inflammation causes a great deal of oxidative stress and the presence of melatonin during sleep times could actively counteract free radical production during this time.
- **Answer:** N/A
- **Prediction:** elicit an effective response against any intruding pathogens

step 2,707,364

- **Question:** What European event caused the Huguenots to abandon Charlesfort?
- **Context:** French Huguenots made two attempts to establish a haven in North America. In 1562, naval officer Jean Ribault led an expedition that explored Florida and the present-day Southeastern U.S., and founded the outpost of Charlesfort on Parris Island, South Carolina. The Wars of Religion precluded a return voyage, and the outpost was abandoned. In 1564, Ribault's former lieutenant René Goulaine de Laudonnière launched a second voyage to build a colony; he established Fort Caroline in what is now Jacksonville, Florida. War at home again precluded a resupply mission, and the colony struggled. In 1565 the Spanish decided to enforce their claim to La Florida, and sent Pedro Menéndez de Avilés, who established the settlement of St. Augustine near Fort Caroline. Menéndez' forces routed the French and executed most of the Protestant captives.
- **Answer:** The Wars of Religion
- **Prediction:** The Wars of Religion

Viewing these examples can be extremely helpful to debug your model, understand its strengths and weaknesses, and as a starting point for your analysis in your final report.

5 Adaptation methods to explore

In this section we provide you with an overview of techniques that are used to improve out-of-domain performance as well as better fewshot adaptation. Your job is to read about some of these techniques, understand them, choose some to implement, carefully train them, and analyze their performance – ultimately building the best system you can. Implementation is an open-ended task: there are multiple valid implementations of a single model, and sometimes a paper won't even provide all the details – requiring you to make some decisions by yourself. To learn more about project expectations and grading, see Section 7.

5.1 Mixture-of-Experts

Appears in: Adaptive Mixtures of Local Experts [14]

One way to train an effective multitask learner is via the Mixture-Of-Experts (MoE) technique. Here, we train k models (called experts) along with a gating function (that controls the mixture). The conditional distribution over labels given an input x is given as

$$p(y \mid x) = \sum_i g_i(x) f_i(x)$$

where $f_i(x)$ is the conditional distribution output by the ith expert e.g. the softmax of logits at the final layer, and $g_i(x)$ is the mixture weight for this expert produced by the gating function. The gating function itself can be parameterized by a neural network. A concrete way to apply MoE to robust question answering is to have a separate DistilBERT model for each of the datasets, and a small MLP as the gating function. By carefully controlling which examples update any expert, the experts can learn specialized behavior leading to a more effective multitask model that can potentially extrapolate better.

5.2 Task Adaptive Pre Training

Appears in: Don't Stop Pretraining: Adapt Language Models to Domains and Tasks [15]

Another approach to adapt a BERT based question answering system to a new domain, is to jointly optimize the QA based loss along with the original masked language modeling (MLM) loss over the inputs. To create instances for MLM training, tokens in a given input $x = (q, p)$ can be randomly converted into `[MASK]` tokens. This techinque, known as Task Adaptive Pre Training (or TAPT) has been shown to be effective for adapting a pre-trained model to a task from a new domain.

5.3 Robustness via Data Augmentation

Appears in: An exploration of data augmentation and sampling techniques for domain-agnostic question answering [16], Semantically Equivalent Adversarial Rules for Debugging NLP models [17]

Many works show that state-of-the-art neural models learn brittle correlations that hurt their out-of-domain performance. One way to prevent the model from learning such brittle correlations is to encode label preserving *invariances* via data augmentation. For instance, given an input $x = (q, p)$ with a label y, one could create an augmented example $x' = (q', p')$ where q' and p' are paraphrases of q and p respectively. Such paraphrases could be produced either via back-translation, or via word substitutions. To get q' via backtranslation, q is first translated into a "pivot" language (say Russian), and then translated back to the original language to produce a paraphrased version. In word substitution based data augmentation, individual words in the input are replaced with either synonyms from a lexicon [18], or replaced with `[MASK]` tokens which are then filled with BERT [19] to obtain an augmented input. Additionally, [17] consider several meaning preserving perturbations that could be effectively used to augment the training data, resulting in improved robustness.

5.4 Domain Adversarial Training

Appears in: Adversarial Training for Cross-Domain Universal Dependency Parsing [20]

The goal of Domain Adversarial Training is to learn *domain invariant features* that do not encode spurious, domain dependent information thorough an adversarial learning framework. Concretely, given an input x from a domain d_i, the features output by the model $f(x)$ are fed into a discriminator g that attempts to classify the domain of x. g is trained to maximize the probability of predicting the correct domain, and the model f requires a signal to maximally confuse the discriminator. [21] apply this to the BERT based QA setting by training g to identify the representations output by BERT at the `[CLS]` token.

5.5 Few Sample Finetuning

Appears in: Revisiting Few-sample BERT Fine-tuning [22]

Many recent works [22] explore the effect of important hyperparameters such as learning rates, number of gradient update steps, number of layers to freeze etc. on fewshot accuracy. These can drastically improve instability due to the small size of the out-of-domain training data, and we encourage students to study and analyze the effect of these choices on fewshot performance.

5.6 Meta Learning

Appears in: Investigating Meta-Learning Algorithms for Low-Resource Natural Language Understanding Tasks [23], Learning to few-shot learn across diverse natural language classification tasks [24]

One dominant technique for building fewshot models is meta learning, where the objective is to build a parameterized *meta* learner $\mathcal{M}_\theta$, that takes a small number of samples from a task (called the support set) as input and outputs an adapted model that can make accurate predictions on new examples from the task (the query set). The training data for meta-learning consists of (support, query) pairs from a collection of *training tasks* and the objective is to maximise accuracy on the query set after adapting to the support set. Recent works [23, 24] have shown empirical success with meta-learning for fewshot adaptation in the context of NLP.

5.7 Fewshot adaptation with in-context learning

Appears in: Making Pre-trained Language Models Better Few-shot Learners [25]

The impressive fewshot abilities of GPT-3 [26] are achieved via *in-context learning*. To make a prediction on an input x from an unseen task, the language model receives additional context c which comprises of a small number of input, output pairs appended together with a prompt. [25] show that in-context learning is more broadly applicable to even MLMs, and automatically discover useful contexts given a small training dataset.

6 Submitting to the Leaderboard

6.1 Overview

We are hosting two leaderboards on Gradescope, where you can compare your performance against that of your classmates. F1 score is the performance metric we will use to rank submissions, although both EM and F1 scores will be displayed. The leaderboards can be found at the following links:

1. **oodomain-val** TBA
2. **oodomain-test:** TBA

You may submit to the dev leaderboard as many times as you like, but **you will only be allowed 3 successful submissions to the test leaderboard**. For your final report, we will ask you to choose a single test leaderboard submission to consider for your final performance. Therefore you must make at least one submission to the test leaderboard, but be careful not to use up your test submissions before you have finished developing your best model.

Submitting to the leaderboard is similar to submitting any other assignment on Gradescope, except that your submission is a CSV file of answers on the dev/test set. You may use the starter code's `test.py` script to generate a submission file of the correct format, or see lines 128-135 for example code to generate a submission file. At a high level, the submission file should look like the following:

```
Id,Predicted
001fefa37a13cdd53fd82f617,Governor Vaudreuil
00415cf9abb539fbb7989beba,May 1754
00a4cc38bd041e9a4c4e545ff,
...
fffcaebf1e674a54ecb3c39df,1755
```

The header is required, and each subsequent row must contain two columns: the first column is a 25-digit hexadecimal ID for the question/answer example (IDs defined in `{dev,test}-v2.0.json`), and the second column is your predicted answer. The rows can be in any order.

6.2 Submission Steps

Here are the concrete steps for submitting to the leaderboard:

1. Generate a submission file (*e.g.,* by running `train.py` in the starter code) for either the validation or test set:

   - To generate a submission file on the validation set leaderboard, run:
     ```
     python train.py --do-eval --eval-dir datasets/oodomain_val
     --sub-file val_submssion.csv --save-dir SAVE-DIR
     ```
   - To generate a submission file on the test set leaderboard, run:
     ```
     python train.py --do-eval --eval-dir datasets/oodomain_test
     --sub-file test_submssion.csv --save-dir SAVE-DIR
     ```

   where `SAVE-DIR` is the folder where the checkpoint is created.

2. Use the URLs above to navigate to the leaderboard. **Make sure to choose the correct leaderboard for your split (VALIDATION *vs.* TEST)**.

3. Find the submit button in Gradescope, and choose the CSV file to upload.

4. Click upload and wait for your scores. The submission output will tell you the submission EM/F1, although the leaderboard will keep scores for the submission with the highest F1 score thus far.

There should be useful error messages if anything goes wrong. If you get an error that you cannot understand, please make a post on Ed.

7 Grading Criteria

The final project will be graded holistically. This means we will look at many factors when determining your grade: the creativity, complexity and technical correctness of your approach, your thoroughness in exploring and comparing various approaches, the strength of your results, the effort you applied, and the quality of your write-up, evaluation, and error analysis. Generally, implementing more complicated techniques represents more effort, and implementing more unusual ideas (e.g. ones that we have not mentioned in this handout) represents more creativity. You are not required to pursue original ideas, but the best projects in this class will go beyond the ideas described in this handout, and may in fact become published work themselves!

There is no pre-defined F1 or EM score to ensure a good grade. Though we have run some preliminary tests to get some ballpark scores, it is impossible to say in advance what distribution of scores will be reasonably achievable for students in the provided timeframe. As in previous years, we will have to grade performance relative to the leaderboard as a whole. Note that the strength of your results on the leaderboard is only one of the many factors we consider in grading. Our focus is on evaluating peoples' well-reasoned research questions, explanations, and experiments that clearly evaluate those questions.

For similar reasons, there is no pre-defined rule for which of the models in Section 5 (or elsewhere) would ensure a good grade. Implementing a small number of things with good results and thorough experimentation/analysis is better than implementing a large number of things that don't work, or barely work. In addition, the quality of your writeup and experimentation is important: we expect you to convincingly show that your techniques are effective and describe why they work (or the cases when they don't work).

In the analysis section of your report, we want to see you go beyond the simple F1 and EM results of your model. One way to analyze your system is to break down the scores. For example, can you analyze why your model does better on some of the held out domains than others? What are the types of examples that the model gets consistently wrong and why? How does accuracy on the in-domain dev set correlate with accuracy on the oo-domain dev set?

As with all final projects, larger teams are expected to do correspondingly larger projects. We will expect more complex things implemented, more thorough experimentation, and better results from teams with more people.

8 Honor Code

Any honor code guidelines that apply for the final project in general also apply for the default final project. Here are some guidelines that are specifically relevant to the Robust QA track of the default final project:

1. You **may not** use a different pre-trained language model other than DistilBERT for your system. However, if you want to use smaller pre-trained models and believe you have a good reason to do so, please make an Ed post to get permission.

2. You also **may not** use pre-existing implementations of QA models or various methods for improving robustness / fewshot adaptation (including those mentioned in Section 5) as a starting point unless you wrote the implementation yourself. Please make an Ed post if you believe you have a good reason to do so.

3. As described above, you may not use pre-existing implementations of QA/robustness methods or pre-trained language models other than DistilBERT, but you are allowed to use other pre-existing NLP tools such as a POS tagger, dependency parser, and coreference module that are not built on top of pre-trained language models.

4. As described in Section 3.1, it is an honor code violation to use **any other training data** for training (or development) apart from the provided training files.

5. You are free to discuss ideas and implementation details with other teams (in fact, we encourage it!). However, under no circumstances may you look at another CS224n team's code, or incorporate their code into your project.

6. Do not share your code publicly (e.g., in a public GitHub repo) until after the class has finished.

References

[1] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*, 2017.

[2] Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A Smith. Annotation artifacts in natural language inference data. In *Association for Computational Linguistics (ACL)*, pages 107–112, 2018.

[3] R Thomas McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Association for Computational Linguistics (ACL)*, 2019.

[4] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4902–4912, Online, July 2020. Association for Computational Linguistics.

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

[6] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. In *Association for Computational Linguistics (ACL)*, 2019.

[7] Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. Newsqa: A machine comprehension dataset. *ACL 2017*, page 191, 2017.

[8] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[9] Amrita Saha, Rahul Aralikatte, Mitesh M. Khapra, and Karthik Sankaranarayanan. DuoRC: Towards Complex Language Understanding with Paraphrased Reading Comprehension. In *ACL*, 2018.

[10] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale reading comprehension dataset from examinations. In *EMNLP*, 2017.

[11] Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension. *arXiv preprint arXiv:1706.04115*, 2017.

[12] Adam Fisch, Alon Talmor, Robin Jia, Minjoon Seo, Eunsol Choi, and Danqi Chen. MRQA 2019 shared task: Evaluating generalization in reading comprehension. In *Workshop on Machine Reading for Question Answering (MRQA)*, 2019.

[13] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[14] R. Jacobs, Michael I. Jordan, S. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

[15] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964*, 2020.

[16] Shayne Longpre, Yi Lu, Zhucheng Tu, and Chris DuBois. An exploration of data augmentation and sampling techniques for domain-agnostic question answering. *arXiv preprint arXiv:1912.02145*, 2019.

[17] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Semantically equivalent adversarial rules for debugging NLP models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 856–865, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[18] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

[19] Siddhant Garg and Goutham Ramakrishnan. Bae: Bert-based adversarial examples for text classification. In *EMNLP*, 2020.

[20] Motoki Sato, Hitoshi Manabe, Hiroshi Noji, and Yuji Matsumoto. Adversarial training for cross-domain Universal Dependency parsing. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 71–79, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[21] Seanie Lee, Donggyu Kim, and Jangwon Park. Domain-agnostic question-answering with adversarial training. In *MRQA@EMNLP*, 2019.

[22] Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. Revisiting few-sample bert fine-tuning. *arXiv preprint arXiv:2006.05987*, 2020.

[23] Zi-Yi Dou, Keyi Yu, and Antonios Anastasopoulos. Investigating meta-learning algorithms for low-resource natural language understanding tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1192–1197, Hong Kong, China, November 2019. Association for Computational Linguistics.

[24] Trapit Bansal, Rishikesh Jha, and Andrew McCallum. Learning to few-shot learn across diverse natural language classification tasks. *arXiv preprint arXiv:1911.03863*, 2019.

[25] Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. *arXiv preprint arXiv:2012.15723*, 2020.

[26] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.