Natural Language Processing
Assessment 2

Building a Question Answering (QA) system using

Submission Instructions:

1. Upload your solution files to Canvas on **Sunday 22st of December 2021**.
2. Go to the Project_2 folder in Canvas to upload your files.
3. Once you have submitted your file, you should verify that you have correctly uploaded it. It is your responsibility to make sure you upload the correct file.
4. Please make sure you **fully comment on your code** that will reflect your **OWN** work.
5. You need to use **Jupyter Notebook** and run the code before the submission, and you can submit a .ipynb, .pdf and/or .html files.
6. Please put **your student name and number** as comments **at the top of your file**.

This project is worth 50% of the Natural Language Processing Module.

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 in the same domain 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.

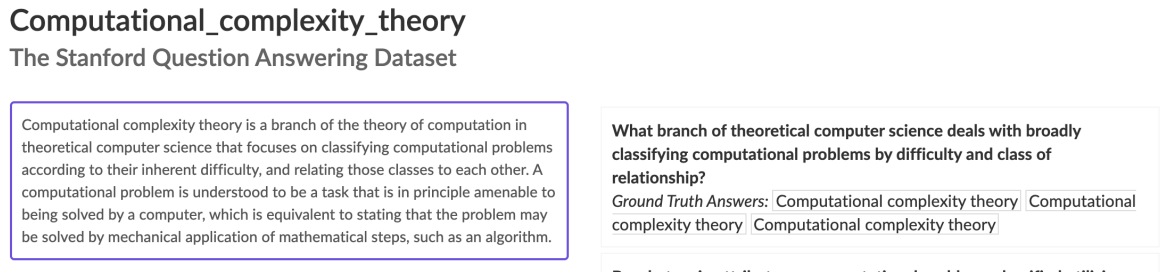
The project consists of **two Parts**. In Part 1, you will be building a QA system for the SQuAD dataset wich will be your baseline, and in Part 2, you will be improving this baseline.

The task, data, and starter kit are borrowed from Chris Manning et al. 2021.

Dataset:

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/ .



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 recall1. 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.

In this project, you are provided with three in-domain reading comprehension datasets (Nat- ural 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 train- ing, 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.

The training set contains many (context, question, answer) triples. 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.

Your QA system will be evaluated on a held out test set (oodomain_test) from the eval datasets. 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 along with the your code file.

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 borrowed from [12].

Part 1 : Training the Baseline (50%)

We have provided code for preprocessing the data and computing the evaluation metrics, and code to train a fully-functional neural baseline. You **can also use your own baseline**. **Your job is to improve on this baseline in Part 2 after training your model in Part 1.**

For this project, you will need a machine with GPUs to train your models efficiently. Once you are on an appropriate machine, download the project folder **'robustqa'** from Canvas.

This folder contains the starter code as well as the datasets that you will be using.
If you are using your own baseline, please add a small description of the model you are using.

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 notice two directories:
• data/: 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.

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!

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

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.

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.

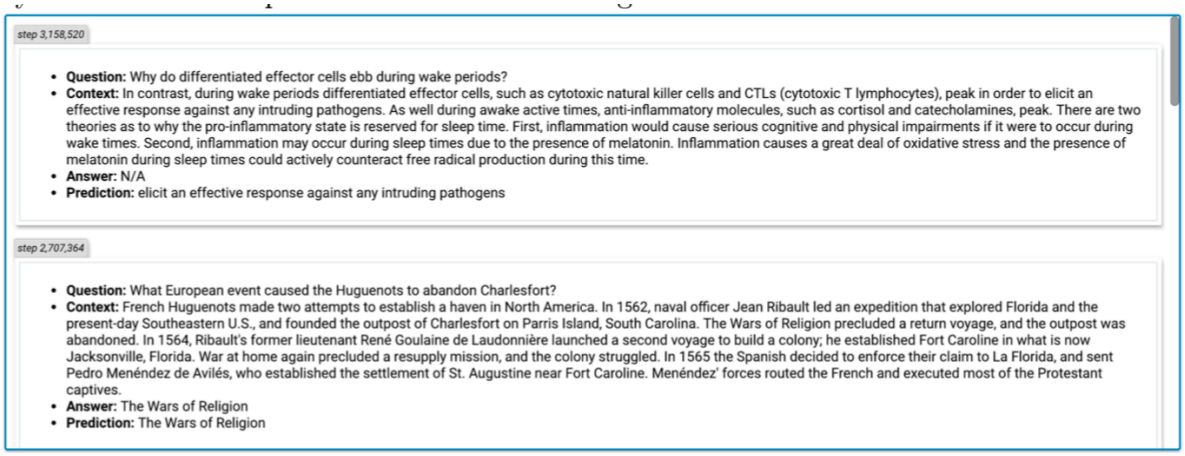
If you are training on a remote machine (e.g. Azure), then run the following command on your local machine:

```
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.

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:



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.

In the next section 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.

Part 2 : Improving the Baseline (50%)

In the second part of the project we provide you with an overview of techniques that are currently 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 one 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.

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 fi(x) is the conditional distribution output by the ith expert e.g. the softmax of logits at the final layer, and gi(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.

2- Task Adaptive Fine Tuning

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 Fine Tuning (or TAPT) has been shown to be effective for adapting a pre-trained model to a task from a new domain.

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.

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 di, the features output by the model f(x) are fed into a discrim- inator 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 discrim- inator. [21] apply this to the BERT based QA setting by training g to identify the representations output by BERT at the [CLS] token.

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.

6- Meta Learning

Appears in: Investigating Meta-Learning Algorithms for Low-Resource Natural Language Under- standing 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 $M\theta$, 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.

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.

Hypothesis generation on the test or dev:

You can generate your answers on the Dev questions file by running train.py in the starter code:

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

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

At a high level, the submission file should look like the following:

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

Evaluate the system on test (or Dev) set with *`python train.py --do-eval --sub-file mtl_submission.csv --save-dir save/baseline-01`*

Code

a. All code should be completed using Python as the programming language.
b. Your code should have a logical structure and a high level of readability and clarity. Please comment on your code and put all code into functions. Your code should be efficient and should avoid duplication.

Late submissions

a. If you don't get the assignments done to your satisfaction and don't meet the minimum requirements by the deadline, you have the option (as with any assignment at MTU) of submitting up to 1 week late for a penalty of 10%.
b. This penalty is subtractive. Work that would have earned 55% if on time would get 45% (not 49.5%) if late.
c. The penalty is applied weekly. So, one day late costs the same as 6.
d. If you have a specific reason for submitting a late assignment (sickness, etc.), please submit directly a medical certificate to the department secretary.

Plagiarism

Please read and strictly adhere to the MTU Honesty, Plagiarism and Infringements Policy Related to Examinations and Assessments. Note that reports are checked against each other and external web sources for plagiarism. Any suspected plagiarism will be treated seriously and may result in penalties and a zero grade.

Grading

The assignment is worth 50% of the overall mark for the module. Marks will be awarded based on the quality of the code and the results. In particular, I will be checking to see if you are handling and preprocessing data correctly, carrying out exploratory analysis to gain insights, correctly performing model implementation, and critically documenting everything in a clear and concise way. The submitted code will also be checked to ensure that the work is your own.

References

[1] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension sys- tems. 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 syn- tactic 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 Meet- ing 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 Bach- man, 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 Ma- chine 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 aug- mentation 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 Asso- ciation 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 Dhari- wal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. arXiv preprint arXiv:2005.14165, 2020.