CS188 Course Project: Do Machines have Common Sense?

1 Objective

This project is designed to walk you through the general paradigm of modern NLP problem solving and research, as well as helping you acquire basic knowledge and practical skill sets to conduct your individual NLP projects. We provide a starter codebase such that you can reuse and extend it in the future if you are interested in continuing doing other NLP projects. The codebase is mainly written in PyTorch, and it includes four major components: (1) data loading and processing, (2) model selection, and training, (3) language model pretraining, and (4) transfer learning.

You will start this project by reading and understanding the provided sample codes, and afterwards, you will be required to implement several essential components which as a whole form the full NLP problem solving pipeline. We prepare specific datasets for you to play with; and we will walk you through a few dedicated close-ended implementation components (*i.e. the basic task*), followed by more open-ended questions which require your creativity to develop the solutions.

Note: Please read the whole instructions carefully and thoroughly!

2 General Guidelines and Background for Accomplishing Basic Tasks

To start off, please **fork** this code repository. (Please DO NOT do **git clone** and modify any of the original repository, fork your own and go from there.) Although not likely, should there be any changes in the original repository, you can simply sync with upstream to obtain the latest changes. Please keep your GitHub repository **private** (even after the end of the course) to prevent potential plagiarisms. In addition to this written guideline, much useful information (and/or tips) is included in the `README.md` within the code repository, please also make sure you carefully read and follow them.

At a high level, the basic requirements for this project is to leverage large-scale language corpora pretrained models (*e.g.* BERT, RoBERTa) and finetune them on a specific text classification downstream task. We will walk you through each essential steps, which includes:

1. Data loading and processing the loaded data to PyTorch tensors. (Section 3.1)
2. Model building leveraging existing libraries. (Section 3.2)
3. Model training and evaluation pipeline. (Section 3.4, Section 3.4.1, Section 3.3)
4. Transferring learned knowledge between tasks. (Section 4)

We hope by end of this project, all of you can grasp the basic paradigm of building a modern NLP system and doing research on your own. Below we provide more details about some terminology and background information.

2.1 TODO Blocks

You will be provided with an incomplete starter code. In the starter code, you can find the parts that you are required to finish, which are wrapped within the TODO blocks exemplified as below:

```
##################################################
# TODO: Some instructions ...
# ...
# End of TODO.
##################################################
```

You do not need to touch the codes outside of the TODO blocks but feel free to modify certain codes if you find necessary.

2.2 Executing Codes

Most of the codes in the provided starter code repository should be executed using the module option. For example, if you would like to execute the code `data_processing/dummy_data.py`, do:

```
python3 -m data_processing.dummy_data
```

Notice that the `dummy_data.py`'s `.py` is gone.

2.3 Datasets

In this project, we will mainly use two common sense reasoning datasets, the Sem-Eval (Semantic Evaluation) 2020 Task 4 dataset [4] and Com2Sense [3] dataset.

2.3.1 Sem-Eval-2020 Task 4

This dataset seeks to evaluate whether an NLP system can distinguish natural language statements that make sense to humans from the ones that do not, and provide the reasons. For each data instance, a correct and an incorrect statements are featured, along with three right reasons and two confusing (usually wrong) reasons. For example:

Correct Statement: *Grizzly bears love honey.*
Incorrect Statement: *Grizzly bears hate honey.*
Right Reason1: *Honey is good for grizzly bear's growth.*
Confusing Reason1: *Bees cannot eat grizzly bears.*
... ...

We ask you to treat the correct and incorrect statements **individually**. That is to say, your model only takes in one statement at a time (either correct or incorrect), and make a binary True/False predictions. You do not need to worry about the reasoning part of this dataset for now unless you decide to do one of the open-ended questions, which will leverage the reasons.
We provide the data under the path `datasets/semeval_2020_task4`, you will find three csv files, `train.csv`, `dev.csv`, `test.csv`. Please check the `data_processing/README.md` for more details.

2.3.2 Com2Sense

Similar to the Sem-Eval dataset, Com2Sense also evaluates machines' abilities on judging whether a given statement is common sensical or not. There are a few additional features in this dataset, one of which is the complementary setting. That is, for each statement, there is a complementary statement which is constructed with small perturbation on certain words making it concerning the similar common sense concepts but with different (opposite) labels. For example:

Statement 1: *As Bob is afraid of heights, he rode the carousel instead of the Ferris wheel.*
Statement 2: *As Bob is afraid of heights, he rode the Ferris wheel instead of the carousel.*
The above two statements are complementary to each other with statement 1 labelled as *True* while statement 2 is *False*. The underlining concepts are swapped in these two examples to construct the complementary statements. This complementary nature of the dataset enables us to define a ***pairwise accuracy*** metric, where a machine's prediction is considered correct only if both samples in a complementary pair are *individually* judged correctly. Therefore, we define two kinds of accuracies: (1) *standard* accuracy that does not consider relation between complementary pairs and simply aggregates the results from each statement, (2) *pairwise* accuracy that considers the predictions to be correct only if the model makes correct **individual judgement** on both statements within a complementary pair. We will ask you to implement the pairwise metric function as part of the project.

Another feature Com2Sense dataset provides is a domain and scenario categorization of each common sense statement. Each statement is tagged with one of the following domains: *Physical, Social, Temporal*, and one of the following reasoning scenarios (reasoning mechanism used to judge the statement): *Comparative, Causal*, and an additional dimension to signify whether numerical common sense in required, labeled as *Numeracy*. These categorization can help diagnose the model performance in a more systematic fashion. We will ask you to dive into these at the later stage.

For more details of this Com2Sense dataset, please refer to the original paper [3], and the `train.json`, `dev.json`, and `test.json` sets are stored under `datasets/com2sense`.

2.4 Essential Execution Args

Before executing any training-related codes, make sure to skim through `trainers/args.py` to learn what arguments are used in the codes. If you forget anything that is being called as `args.XXX`, be sure to refer back to this file. (We indicate some important ones in `README.md`.)

3 Your Implementation Tasks

3.1 Data Loaders and Processors

The first step of this project (and also usually for any machine learning/NLP research projects) is to deal with data and its preprocessing. For this setup, the proportion that you will need to implement is under the `data_processing` folder. Please carefully follow the procedure below:

1. To help you get started, we provide a reference code `data_processing/dummy_data.py` which reads in some very simple dummy dataset from `datasets/dummies`.
2. Please pay attention especially to the `_read_data` function as you will need to implement this function for both `data_processing/{com2sense,semeval}_data.py`.
3. Please refer to `data_processing/utils.py` for essential utility functions and classes, and finish all the `TODO` blocks in the `data_processing/{com2sense,semeval}_data.py`.
4. Once both of the aforementioned dataset processors are done, please read the dataset class `DummyDataset` in `data_processing/processors.py` and finish the `TODO` blocks in both `Com2SenseDataset` and `SemEvalDataset`. You may find this tokenizer guideline helpful.

5. Take a look at the `data_processing/__init__.py` to check how we will read in these implemented processors and dataset classes as a python dictionary for convenience.

For most of the implementations you can refer to the `data_processing/README.md` to verify whether your implementations are correct or not.

3.2 Model Selections

As many modern NLP language models are now public with their pretrained weights provided online, in this part, we will walk you through how to use them. Please refer to the AutoModel documentation from HuggingFace library to choose the correct config, tokenizer, and model classes to use, and finish the below `TODO` block in `trainers/train.py`:

```
##################################################
# TODO: Please fill in the below to obtain the
# `AutoConfig`, `AutoTokenizer` and some auto
# ...
# End of TODO.
##################################################
```

Once this part is done, you're almost all set! Execute the script `scripts/train_dummy.sh` to see whether the whole training pipeline is working, and the model can eventually **overfit** the small dummy dataset. If all of your implementation is correct, then you should be able to obtain 100% accuracy on the dev and test sets since they are identical to the train set.
NOTE: For computational resource fairness, please **DO NOT** use models larger than **RoBERTa-large**, and we only allow models from the following MLM family: {bert, roberta, deberta}.

3.3 Masked Language Modeling (MLM)

You have learned the basic concepts and algorithm of the masked language modeling widely adopted in modern NLP language models, now it is time for you to actually implement it (as learning by doing is always the best)! Although we do not really request you to perform a large scale pretraining for this part of the basic tasks, you may find the MLM pretraining phase handy when you are working on the open-ended parts of this project.
NOTE: We will introduce a few potentially useful pretraining datasets later on, stay tuned!
Please finish the `TODO` blocks in the function `mask_tokens` in `trainers/train_utils.py` following the guided comments. You can test your implemented functionalities by the following command:

```
python3 -m trainers.train_utils
```

For this part, we only check the specific functionality of this function, and you can execute the script `scripts/run_pretraining.sh` after your full training and evaluation loop in Section 3.4 is done to see how it works in practice.

3.4 Training and Evaluation Loop

This is almost like a recap of what you have learned throughout our TA sessions where we go over the general paradigm of PyTorch training and evaluation mechanisms. Please fill in all the necessary details in the below `TODO` blocks, and these should be straightforward if you have paid attention to our sessions :)

- Training Loop.

```
##################################################
# TODO: Please finish the following training loop.
# ...
# End of TODO.
##################################################
```

- Evaluation Loop.

```
##################################################
# TODO: Please finish the following eval loop.
# ...
# End of TODO.
##################################################
```

- Please note that the requirements of the input structure for pretraining of the language model is a little different from the finetuning phase. Make sure you implement an if-else statement to distinguish them.

3.4.1 Evaluation Metrics

For the *standard* Com2Sense performance computation, we ask you to compute all of the following metrics: **accuracy**, **precision**, **recall**, and **f1-score**. The APIs for all these metrics can be found at scikit-learn's metrics library. Please finish the `TODO` block in `trainers/train.py` :

```
##################################################
# TODO: Please finish the results computation.
# ...
# End of TODO.
 ##################################################
```

For the *pairwise* accuracy metric, you only need to consider the **accuracy** metric. Please finish the `TODO` block in function `pairwise_accuracy` in `trainers/train_utils.py` :

```
##########################################################
# TODO: Please finish the pairwise accuracy computation.
# ...
# End of TODO.
##########################################################
```

and use the following command to test your implementation:

```
python3 -m trainers.train_utils
```

3.5 Running Scripts

We have provided some useful bash scripts for you to easily launch any training and/or evaluations under the folder `scripts` . Feel free to change any of the parameters for your tasks throughout the project. Note that at any point of your implementation, you may execute the corresponding scripts and set some debugging checkpoints to facilitate and verify your implementations.

```
# Script for finetuning a model on dummy dataset.
scripts/train_dummy.sh

# Script for finetuning a model on Sem-Eval dataset.
sh scripts/train_semeval.sh
```

```
# Script for finetuning a model on Com2Sense dataset.
sh scripts/train_com2sense.sh

# Script for running an MLM pretraining on some dataset.
sh scripts/run_pretraining.sh

# Script for loading an MLM-pretrained model and continue finetuning.
sh scripts/finetune_from_pretrain_dummy.sh
```

4 Your Experimental Tasks

The basic requirement of this project is to train some NLP models **on the downstream *Com2Sense task***. We ask you to perform the following two things, where you will describe in details what you did in your write-up:

- **Supervised Learning.** Using the datasets under `datasets/com2sense`, simply train the models of your choice using `train.json`, evaluate during training on the `dev.json`. After the training is converged, you can choose the model that reaches the best development set performance to be evaluated on the test set: `test.json`. NOTE: the provided test set does not contain labels, and the codes should generate a "com2sense_predictions.txt" (make sure it has 5580 lines) file in the output directory, which you will submit to a special Gradescope entry, **"Com2Sense Test Set Trial X"**, to get the actual test results to include in your write-up. There will be **three** such entries with different start and end dates. During each submission period, you can resubmit many times, however, the results will only be made visible after each designated end date so they basically reflect your last submission during that particular period. And hence, in total you have **three attempts** to try. More details will come as an announcement or Piazza post later on.

  Please provide your experimental details (*e.g.* what process you followed to run the code and get the results, the range of hyper-parameters you tried and the best combinations you eventually used, the blinded test set results, and you are also encouraged to include your training and development loss curves/accuracies) in the write-up.

- **Knowledge Transfer.** Similar to the above procedure, but this time, our goal is to transfer knowledge that the models learned from the Sem-Eval dataset to help them perform the Com2Sense task. To achieve this, you should first finetune your models on the Sem-Eval dataset as a proxy task since it also concerns common sense reasoning questions. We hypothesize that training on this task would enable the models to firstly learn some useful common sense knowledge that could be helpful for the Com2Sense task. Choose a checkpoint of desire to save, and then restore it at the beginning of your training pipeline for the Com2Sense task. Do you see any performance gain from this *two-staged* training?

  Please provide your experimental details (*e.g.* what process you followed to run the code and get the results? How do you decide how much to train on the Sem-Eval dataset? How do you achieve the transfer? the test set results for both Sem-Eval and Com2Sense, and again you can also include your training and development curves/accuracies) in the write-up. Please provide some explanations comparing the results with the supervised training setup.

- **Other Analysis.** As Com2Sense dataset comes with additional features such as **scenario** and **domain**, can you come up with some interesting analysis utilizing these? For example, do you

observe any domains or scenarios benefit from (or hurt by) the two-staged training compared to directly finetune pretrained models on Com2Sense dataset? Or, which categories (you can combine scenario, domain, and/or numeracy) do you think the models perform the best/worst? You can refer to the original paper of Com2Sense [3] for some inspirations of such analysis, but feel free to creatively come up with your own!

5 Open-Ended Questions

In this section, we will provide a few exploratory directions and some open-ended questions for you to solve. Some of the questions resemble the actual research problems that NLP researchers face everyday, and we would like to give you a taste of it.

5.1 Tasks

There are two main directions for the open-ended explorations of this project:

1. **Improve the overall performance.** Now that you have gone through all the essential parts of the project so far, you may begin to think, *"How can I further boost the model performance?"* If you choose to pursue in this direction, you will need to be creative on coming up with methods to improve the model performance beyond what you have implemented in the basic tasks section. We will provide a few useful tips and hints in the next section.

   NOTE: For computational resource fairness, please **DO NOT** use models larger than **RoBERTa-large**, and we only allow models from the following MLM family: {bert, roberta, deberta}.

2. **Generate plausible explanations.** The Sem-Eval dataset actually features explanations for each statement with respect to why the statement is compliant to or against human common sense. Therefore, another direction you can explore is to generate explanation along with your prediction (classification) about whether a common sense statement is true or false. For this task, you can leverage *generative language models*, such as T5 models [2], and train them on the explanation generation task to explain why a particular answer (true/false) makes sense.

5.2 Tips and Things to Try

We list a few ideas at a high level that you may want to try for each direction, and you are of course encouraged to come up with your own ideas that are outside of this initial list:
For the **improving performance direction**:

- Similar to the two-staged training you have experienced in the basic task, can you come up with a few other datasets that may be potentially helpful for the downstream Com2Sense task? Maybe some other common sense question answering datasets?
- Use your implemented MLM training paradigm, maybe you can pretrain on some text corpora and perform certain special *masking tricks*? *i.e.* maybe you can enforce the masking to be on a whole word of nouns, verbs, etc. that perhaps what common sense is centering around?
- There are a few external resources such as some knowledge bases which contain quite a few knowledge potentially important to common sense *e.g.* ConceptNet. Maybe try to come up with a way to leverage them?

For the **generating explanations direction**:

- You can try the model T5 that is mentioned in Section 5.1, but you can also try GPT-2 [1] or other generative language models that you may have heard of or learned throughout the class.
- Maybe try to come up with a good ***prompt*** and/or pattern to be fed into the models to incentivize the language models to understand that you would like it to generate an explanation after feeding in the answer (predicted by your common sense finetuned model)? To facilitate this even further, how about extracting some possible *explaining sentences* from some text corpora (*e.g.* pretraining corpora or other language model training resources)?
- Although we do not have the ground truth explanation in our Com2Sense dataset, maybe you can try to leverage the semi-supervised learning paradigm to iterate between: (1) *self-generating* explanations from a well-trained explanation generative model, and (2) treat these generated explanations as labels and continue training the models on them?

DISCLAIMER: You are not required to follow any of the above suggested items, as they merely serve as prompts and are reasonable directions that an NLP researcher may try. Please feel free to come up with methods on your own and be creative!

6 Grading and What to Submit

Please export your report to a single PDF file with the following sections clearly written:

- Introduction.
- Referenced Related Works.
- Main Methods (re-iterate the basic task but mainly focus on the open-ended parts).
- Main Results (include both the basic task and the open-ended parts).
- Conclusion and/or Discussions.
- References.

The **page limit is 4** without considering the references part. We recommend the ACL template for compiling your report, and please change it to the public version (which shows your names and institutions instead of *"Anonymous..."*).
Please simply include the PDF file report in your repository named as `group_{GROUP_ID}.pdf` and **zip your entire code repository** as `group_{GROUP_ID}.zip` to submit. We will provide a spreadsheet for you to sign up your team members. The actual submission guideline as well as how to obtain the test set results will be made as another announcement/Piazza post, please stay tuned!

We will grade based on the following percentage breakdown:

- 55% on the basic task performance & analysis.
- 20% on the open-ended task.
- 25% on the report write-up.

This is the end of the instructions. Enjoy The Project!

References

[1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[2] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[3] Shikhar Singh, Nuan Wen, Yu Hou, Pegah Alipoormolabashi, Te-Lin Wu, Xuezhe Ma, and Nanyun Peng. Com2sense: A commonsense reasoning benchmark with complementary sentences. In *Association for Computational Linguistics (ACL)*, 2021.

[4] Cunxiang Wang, Shuailong Liang, Yili Jin, Yilong Wang, Xiaodan Zhu, and Yue Zhang. Semeval-2020 task 4: Commonsense validation and explanation. *arXiv preprint arXiv:2007.00236*, 2020.