CSE 151B
Assignment 3
Transformers and Contrastive Learning

Summer Session 1 2026

Instructions

Due July 22nd by 11:59pm PST

1. There are two components to this assignment: written homework (Problems I.1-I.4), and a programming part. The written homework must be done individually, following the Gilligan's Island rule (see item 3 below).

2. You need to submit all of the source codes files and a **README.md** file that includes detailed instructions on how to run your code.

   You should write clean code with consistent format, as well as explanatory comments. Do not submit any of your output plot files or .pyc files, just the .py files and a **README.md** that explains how to run your code. You can include one .ipynb file only if that file is for some visualization purpose.

3. Any form of copying, plagiarizing, grabbing code from the web, having someone else write your code for you, etc., is cheating. Discussions of course materials and homework solutions are encouraged, but you should write the final solutions to the written part alone. Books, notes, and Internet resources can be consulted, but not copied from. Working together on homework must follow the **Gilligan's Island Rule** (Dymond, 1986): No notes can be made (or recording of any kind) during a discussion, and you must watch one hour of Gilligan's Island or something equally insipid before writing anything down.

4. **Start early!** If you have any questions or uncertainties about the assignment instructions, please ask about them as soon as possible (preferably on Piazza in a public post, so everybody can benefit). We want to minimize any possible confusion about this assignment and have tried very hard to make it understandable and easy for you to follow.

5. You will be using PyTorch, a Deep Learning library, for the tasks in this assignment. Also you are allowed to only use Numpy and Matplotlib apart from the libraries that come along with the Python installation. Steps to access the UCSD GPU cluster are explained in the following sections.

Learning goals and important notes

1. Understand the basics of transformers in NLP applications, including BERT pretrained models, tokenizers, etc.
2. Understand the basics of contrastive learning, as in SimCLR.
3. Build a baseline from a pretrained BERT model to classify scenario.
4. Learn to use techniques from online resources to further improve the model performance.
5. Learn to read research papers and transfer and modify the corresponding repositories to your own codebase.
6. Learn to build research expectations before experiments and reflect on experimental results. Come up with reasons if the experimental results are expected or not. Come up with ideas for future work to make further improvements.
7. Unless otherwise specified, you are free to change the starter code, such as adding more functions, new files etc.
8. Please try your best to determine if your results look correct with experimental evidence.

Transformers

Problem statement: Transformers have achieved state-of-the-art performance in many different tasks recently. In this part of the assignment, we will explore and use a pre-trained BERT model to classify the scenario. We will first fine-tune BERT as a baseline model, and then apply various training techniques obtained from a blog to build a custom model, and finally, train models with contrastive losses. More details on BERT are here.

We will be using the amazon massive scenario dataset for training our model. This dataset has 18 categories of scenarios. Huggingface makes it easy to get familiar with the data samples and labels with train/validation/test splits. For the classification tasks (except for contrastive learning at Task 5), the input to the model is the text and output is the scenario label. For example, with the input text "wake me up at nine am on friday", the scenario label of this text is alarm.

Implementation Instructions

Please familiarize yourself with Pytorch and Huggingface before implementing. One of the good places to start with Pytorch is this.

We have provided you with dataset processing functions in **dataloader.py** where the prepare_features function is to be written. The function of **load.py** is to load the tokenizer in the load_tokenizer function which you need to write. The barebone model classes to be written are in **model.py**. A train script is in **main.py** for the 3 tasks separately. You can change the parameters in **arguments.py** and a shell script **run.sh** is provided to run the file. This is to help your file structure organization.

You can add other files and modify/add/delete the functions and arguments in the starter code if necessary. **Please do not modify code that saves to your results/[experiment]/[experiment].txt files, as we will use this along other tests to verify your code.** You are also free to decide model configurations and experimental settings if we do not explicitly mention.

The following are the implementation steps.

1. **Preparation**
   (a) **Evaluation metrics**
   Before we discuss the neural network details, we must clarify how you will accurately and transparently evaluate your model's performance. You will be just using the following metric:

   Accuracy: percent correct predictions $= \frac{\text{total correct predictions}}{\text{total number of samples}}$

   (b) **Environments**
   Use the **pip install ...** command in requirements.txt to install Pytorch if necessary. The command is in comment by default.
   Use the command **pip install -r requirements.txt** to install the necessary packages for transformers.

   (c) **Model Documentation** Refer to the following document of the needed functions: BERT.

2. **Datasets**
   (a) The dataset is automatically downloaded in the "assets" folder.
   (b) Learn how to load the tokenizer in **load.py** and preprocess the text data in **prepare_features** function in dataloader.py.

3. **Baseline Model**
   (a) We will fine-tune a pre-trained BERT model on the dataset mentioned above for scenario classification.
   (b) Follow the instructions in **baseline_train** in **main.py** to train the baseline model.
   (c) Follow the instructions in **ScenarioModel** class in **model.py** to load the pretrained model through BertModel.from_pretrained(...). This pretrained bert is the encoder. More introduction of the bert-base-uncased model mentioned.

(d) After feeding the input to the encoder, take the **last_hidden_state**'s [CLS] token as output of the encoder, feed it to a **dropout** layer with the preset dropout rate in the argparse argument, feed the output of the dropout layer to the **Classifier** which is provided for you. More details of why the [CLS] token is chosen as sentence embedding.

(e) Fine-tune a **baseline model** with the following design choices: Use the **cross entropy** loss function to train your model for 10 epochs (complete passes through the training set). You can use any optimizers and/or learning rate scheduler you wish.

(f) As a reference, we can get above 80% test accuracy in 10 epochs. This is for reference only. You should try your best to gain experience and learn how to improve the model as much as possible. The results.txt file records the training loss and accuracy at each epoch. **Note:** Training time can vary a great deal on the UCSD GPU cluster, depending on the load. On average on an unloaded machine, you can expect around 1 min per epoch on the cluster.

(g) Feel free to experiment with any other settings that we do not specify, including parameters of the library functions (except for defaults), the hyperparameters you chose etc. **Please record the best training log under results/baseline/baseline.txt.**

4. **Custom Fine-tuning Strategies**

(a) Now, try to improve over the baseline model. Pick one technique from Advanced Techniques for Fine-tuning Transformers to fine-tune your BERT. Though this blog is for RoBERTa, these techniques are also applicable to BERT. **You should not expect better performance by simply using these strategies. It is more important to understand how to use them by changing the hyperparameters accordingly etc.**

(b) Follow the instructions in **custom_train** in **main.py** to train the custom fine-tuning strategies model.

(c) Use class inheritance from ScenarioModel to create the **CustomModel** class in **model.py**. Make necessary changes to use different techniques in "Advanced Techniques for Fine-tuning Transformers" in CustomModel.

(d) Train this custom model with your chosen technique. Make necessary changes in the code, such as adding a new training function, a new learning rate scheduler etc.

(e) Feel free to experiment with any other settings that we do not specify, including parameters of the library functions (except for defaults), the hyperparameters you chose etc. **Please record the best training log under results/custom/custom.txt.**

5. **Contrastive Learning**

(a) We will train the BERT with contrastive losses and compare the embedding in the pre-trained models and fine tune them for our classification objective. Note that evaluation scores accuracy by taking the argmax over the class logits (using the same **run_eval** as the baseline), so your contrastive model must still be able to produce classification logits at evaluation time—for example, by training the provided classifier head jointly with the contrastive loss.

(b) First. Review the SimCLR slides. Read the SupContrast and SimCSE. You do not have to read the whole paper, but make sure that you understand enough to consider the answers to the following questions. These questions do not need to be answered, they are for your reference and understanding.
Compare the SimCLR with SupContrast. What are the similarities and differences?
How does SimCSE apply dropout to achieve data augmentation for NLP tasks?

(c) Read (and run if necessary) the SupContrast repository. Learn how the SupContrast and SimCLR losses are used. The losses.py file has been copied and provided in the starter code (and is renamed as loss.py). Both the SupContrast and SimCLR losses are incorporated in this file. Make necessary changes so that SupContrastive loss and SimCLR loss can be used. The embedding for the SupContrastive and SimCLR is the [CLS] token.

(d) Follow the instructions in **supcon_train** in **main.py** to train the SupContrast model. Make a simple argument option so that you can opt to train with SimCLR loss. Hint: Refer to SupContrast repository if necessary to learn how to train the model with SupContrast loss.

(e) Use class inheritance from ScenarioModel to create the **SupConModel** class in **model.py**. Make necessary changes of the model in order to use SupContrast loss. The SimCLR loss shares the same model structure. Hint: Refer to SupContrast repository if necessary.

(f) Since NLP task is hard to implement data augmentation, add a dropout layer to the [CLS] token before the projection head in order to generate different embeddings for the same text input. You only need to implement the linear head.

(g) Train your model with cross-entropy (baseline model), SupContrast, and SimCLR respectively.

(h) It is also highly recommended to train with different batch size. Explore how the performance changes with batch size.

(i) Feel free to experiment with any other settings that we do not specify, including parameters of the library functions (except for defaults), the hyperparameters you chose etc. **Please record the best training log from your experiments under results/supcon/supcon.txt.**

You are not required to add custom fine-tuning strategies for the SupContrast and SimCLR models but feel free to do so.

Here are some **estimated thresholds** for a high pass for different experiments in this assignment. Note that this may change (in your favor) depending on the overall class performance!

- baseline - Test accuracy: 0.8
- custom - Test accuracy: 0.82
- supcon - Test accuracy: 0.83

(custom varies a lot based on what change you implement, test accuracy threshold is there just to catch completely incorrect submissions)

Start early. Datahub GPUs might get over utilized towards the end of PA and you may not be able to get access to GPU when you need it. Happy Coding !!