Transfer learning using Actually Robust Training (ART)

Hi! In this tutorial, we will walk you through the process of using ART to perform transfer learning. We will use the Yelp Reviews dataset and the `prajjwal1/bert-tiny` model from HuggingFace. We will train a classifier to predict the sentiment of a review (positive or negative) and then we will use ART to perform transfer learning to attack the classifier. Most of the code will follow HF's tutorial with some modifications to make it work with ART.

Just to remind you - the main goal of ART is to follow Karpathys' recipe of training neural networks.

We'll do everything in a script, your task will be to fill the `run.py` accordingly with our instructions from this tutorial.

In [ ]:
```
!pip install art-training nltk wordcloud
```

Data Analysis

Firstly we need to download the data and do some analysis on it. We'll use the `datasets` library from HuggingFace to do this, and we'll wrap the model to Lightning's `DataModule` to make it easier to use with PyTorch Lightning. We prepared the dataset for you in `dataset.py`, check it out there - it's nothing more than just a simple Lightning's Datamodule. The main function in `run.py` is just ready to download the data and show you a sample from it:

In [ ]:
```
!python run.py
```

▶ Suggested main()

Now we can become one with the data. We want to know some statistics, that will be helpful throughout the whole project. We prepared for you a data analysis step in `steps.py`. Now it's your turn! Fill the `...` places in `steps.py` to get the data statistics. You can use `dataset.py` to get the data.

▶ Hint for filling TextDataAnalisys

▶ Suggested TextDataAnalisys

In [ ]:
```

```

As you've done it, modify the main() function as follows:

- read the data
- start the ART project

- add our data analysis step with checking, whether the result exists
- run all the steps (for now we have just one)

▶ Hint for filling main()

▶ Suggested main()

In [ ]:
```
!python run.py
```

If you can see the output below, and the wordcloud.png in checkpoints folder we're good to go!

```
Steps status:
data_analysis_Data analysis: Completed. Results:
        number_of_classes: 5
        class_names: ['0', '1', '2', '3', '4']
        number_of_reviews_in_each_class: Counter({1: 240, 2:
208, 4: 189, 0: 189, 3: 174})```
```

Extra tasks

- Try to write your check to check whether the wordcloud exists
- Try to calculate more statistics that you find useful, save them in the results, and add checks in the `run.py` to verify whether they exist!
- Try to log the results in log_params function in `steps.py`
- Try to check whether the number of unique words is greater than 500

Preparation of metrics in our project

As we have the data, we ca work on our models, which will solve the sentiment analysis problem! We start with a simple baseline. But before that, we need to define metrics that we'll use throughout the entire experiment:

- Calculate the number of classes - you can write it by yourself or use the `number_of_classes` from the results of the previous step
- Define metrics - we'll use Accuracy, Precision, Recall, and the CrossEntropyLoss. Initialize each of them in a list `METRICS`
- pass that list to the project - `project.register_metrics(METRICS)`

▶ Suggested main()

In [ ]:
```
!python run.py
```

At this stage you should see, that the first step was skipped, because we already have executed it.

But why do we need metrics defined for the project?

Take a look at the `MetricsCalculator` from the ART. It takes care of calculating metrics for each step in the project. It's a very useful class, as it allows us to calculate metrics for each step in the project, and then we can use them to compare different models. It's also very useful when we want to compare different models on the same dataset. We can just add the metrics to the project, and then we can compare them in the end.

Baselines

In every project, we have to start from the baselines! We prepared one baseline for you in `models/simple_baseline.py`.

The baseline, as every other model used in ART, has to inherit from the `ArtModule` which is a wrapper for PyTorch Lightning's `LightningModule`. The `ArtModule` has a few useful methods, that we'll use in our project. The most important is the integration with the previously mentioned `MetricsCalculator`, but we'll come to that later when we develop the first deep learning model. For now, we use `ml_parse_data` which parses data specifically for the non-deep-learning training (we don't use PyTorch there), and the `baseline_train` method, which "trains" the model. In our case, it's just calculating probabilities for each class and returning them. We'll use it to compare it with our deep learning models. Take attention to the `ml_parse_data` return format - it's a dictionary `{INPUT: X, TARGET: y}`

Add the baseline to the project and run it:

- Create a baseline callable object - do not initialize it!
- Add the `EvaluateBaseline` step to the project by checking whether scores for each metric exist
- Run the project

▶ Hints for evaluating the baseline

▶ Suggested main()

In [ ]:
```
!python run.py
```

The Suggested output should look like this:

```
Steps status:

data_analysis_Data analysis: Skipped. Results:
        number_of_classes: 5
        class_names: ['0', '1', '2', '3', '4']
        number_of_reviews_in_each_class: {'4': 189, '1': 240,
'3': 174, '0': 189, '2': 208}


HeuristicBaseline_2_Evaluate Baseline: Completed. Results:
        MulticlassAccuracy-HeuristicBaseline-validate-Evaluate
```

```
Baseline: 0.30702152848243713
        MulticlassPrecision-HeuristicBaseline-validate-Evaluate
Baseline: 0.3316725790500641
        MulticlassRecall-HeuristicBaseline-validate-Evaluate
Baseline: 0.30702152848243713
```

Extra tasks

- Try to write your own baseline in `models/baseline2.py` and evaluate it in the project

Training the proper model

As you might already know from the choice of tokenizer, we chose the bert-tiny for this problem. This dataset is hard, so we'll be able to obtain ~45% accuracy on the test set.

We prepared the model for you in `models/bert.py`. It's a simple model, that uses the `prajjwal1/bert-tiny` model from HuggingFace. We use the `AutoModelForSequenceClassification` model, which is a model that takes a sequence of tokens and returns the logits for each class. We use the `AutoTokenizer` to tokenize the text, and then we use the `BertForSequenceClassification` to get the logits. We use the `AutoModelForSequenceClassification` with the `prajjwal1/bert-tiny` model because it's already trained on the sentiment analysis task, so we can use it as a starting point for our model. We could train the last layer of the model, which is a linear layer, and we'll be able to get some good results. But we'll also try to perform fine-tuning of the whole model, to see if we can get better results.

Notice a few things:

- We use the `ArtModule` as a wrapper for the `LightningModule` - it's a very useful class, as it allows us to use the `MetricsCalculator` to calculate metrics for each step in the project
- Notice the `compute_loss()` - it only takes calculated loss from the `MetricsCalculator` which is passed inside the data dictionary. Pure ART's magic!
- Pay attention to the format of returning predictions and data, as previously done in the baselines

Before we train the final model we'll perform some experiments:

- Check loss on initialization - add `CheckLossOnInit` to the project
- Overfitting one batch with an unfrozen backbone - add `OverfitOneBatch` to the project
- Overfitting the entire dataset with an unfrozen backbone - add `OverfitEntireDataset` to the project

Then, if our steps succeed we can perform training on the entire dataset - first with a frozen backbone, then with an unfrozen backbone and reduced learning rate - just add `TransferLearning` to the project

▶ Hints for checking loss on initialization

▶ **CheckLossOnInit in main()**

▶ Hints for overfitting one batch

▶ **OverfitOneBatch main()**

▶ Hints for overfitting the entire trainset

▶ **Overfit main()**

▶ Hints for performing full transfer learning

▶ **Final main()**

In [ ]:
```
!python run.py
```

Conclusions

Congratulations!! You've just performed transfer learning on the Yelp Reviews dataset! You can check the results in the checkpoints folder. You should see, that the model is able to achieve ~45% accuracy on the test set. It's not a lot, but it's a good result for this dataset.

Extra tasks

- Add logger, currently we support Neptune and Wandb. You can initialize the loggers and pass it to every step by `logger` argument.
- Experiment with other Checks
- Write your own Step (and Check if needed) to perform further research

▶ Loggers usage