# Homework 3: Fine-tuning with the Hugging Face `transformers` library

**Deadline:** Wednesday, March 9 2022, 9:30 AM EST

In the Transfer Learning Lab (MLLU / NLU) we learned how to use the Hugging Face `transformers` library to fine-tune large language models on other downstream NLU tasks. In this assignment, we will extend this knowledge further by running a Bayesian hyperparameter search for a RoBERTa model fine-tuned on the BoolQ dataset. In practice, fine-tuning a RoBERTa model on a new task typically requires some experimentation in order to determine the appropriate hyperparameters and training setup. This assignment will focus both on writing the code for this pipeline and on incrementally testing your code.

**Dataset:** The dataset we will be working with is BoolQ, a set of ~16K yes or no questions paired with question-specific contexts that must be used to answer the questions. Here is an example of a single record in BoolQ:

```
{
      “question”: "is france the same timezone as the uk",
      “passage”: "At the Liberation of France in the summer of 1944, Metropolitan France kept GMT+2 as it was the time then used by the Allies (British Double Summer Time). In the winter of 1944--1945, Metropolitan France switched to GMT+1, same as in the United Kingdom, and switched again to GMT+2 in April 1945 like its British ally. In September 1945, Metropolitan France returned to GMT+1 (pre-war summer time), which the British had already done in July 1945. Metropolitan France was officially scheduled to return to GMT+0 on November 18, 1945 (the British returned to GMT+0 in on October 7, 1945), but the French government canceled the decision on November 5, 1945, and GMT+1 has since then remained the official time of Metropolitan France."
      “answer”: false,
      “title”: "Time in France",
}
```

**What to submit:**
In addition to this handout, you are also provided with a directory of skeleton code to complete. Once complete, please compress the directory in a .zip file and submit only the .zip file. Please submit the following to Brightspace:

- Completed code in a .zip file
- Logs from your SLURM batch job (both the .err and .out files)
- A short PDF report detailing the results of your hyperparameter search (see more detailed instructions below).

# Assignment Setup

You will need to ensure that the following prerequisites are met for this assignment:

1. You can access NYU HPC GCP (Cloud Bursting via NYU Greene HPC Cluster) and run Python code in Singularity containers (see the HPC Lab and the instructions below).
2. You have installed Miniconda and the requisite Python packages in your Singularity container. Please ensure that your `transformers` package version is >=4.13.0. In addition to the packages installed during the lab, also run:
   ```
   pip install "ray[tune]" pandas bayesian-optimization
   ```
3. Download and unzip the BoolQ dataset into a directory of your choosing:
   ```
   wget 'https://dl.fbaipublicfiles.com/glue/superglue/data/v2/BoolQ.zip'
   unzip 'BoolQ.zip'
   ```

# Loading and encoding data

*40 points (10 points for each of #1-4)*

Carefully read the skeleton code in `data_utils.py` and `data_utils_test.py`. The former contains functions for tokenizing and encoding the questions, passages, and labels. You should use the tokenizer and maximum sequence length provided in the arguments. You can assume that the tokenizer is an instance of a Hugging Face `PreTrainedTokenizerFast` object that you can simply call on the input `dataset` argument. (Please also read the rest of the comments for exact implementation details.) The latter contains a set of unit tests for ensuring that your code works as expected. Unit tests are intended to test individual pieces of logic in your code - either a single function or a single decision branch in a larger function. It is generally best practice to write and run your tests for an individual module before moving on to implementing other modules. In this section, complete the following:

1. Complete the TODOs in `data_utils.py`. Note that the data is provided to you in the arguments as a Pandas dataframe with the columns “question,” “passage,” “idx” (a numeric example ID), and “label” (a Boolean value).
2. Complete the TODOs in `data_utils_test.py`. To check that your tests pass, run ‘`python data_utils_test.py`’. For extra help with unit testing in Python, refer to the `unittest` documentation.

Now read the code for the `BoolQDataset` class in `boolq.py` and the accompanying unit tests in `boolq_test.py`. `BoolQDataset` implements the `torch.utils.data.Dataset` interface and will be used by the Hugging Face `Trainer` to provide inputs to your model. Complete the following:

3. Complete the TODOs in `boolq.py` using the functions you wrote in `data_utils.py`.
4. Complete the TODOs in `boolq_test.py`. To ensure your tests pass, run ‘`python boolq_test.py`’.

# Setting up model training and hyperparameter search

*30 points (15 points for each of #5 and #6)*
Like in the lab, we will use the Hugging Face `Trainer` to run our model training. We will also use the `transformers` library's integration with the Ray Tune hyperparameter tuning library to run a Bayesian hyperparameter search on the learning rate.

Bayesian hyperparameter search aims to tune hyperparameters by modeling the learning objective as a Gaussian process. You will not need to implement these details yourself, but essentially the algorithm first trains the model on a number of randomly sampled hyperparameter settings. Then the evaluation metrics (e.g. loss, accuracy) for each training run are used to estimate a Gaussian posterior distribution that describes the likelihood that model performance would improve relative to past runs given other combinations of hyperparameter values. (For more details, see this lecture.) In this assignment we will use the `hyperparmeter_search()` method from the Hugging Face `Trainer` interface, which provides the `search_alg` parameter that lets us specify what kind of search algorithm to use.

To implement this hyperparameter search, complete the following:

5. Read the skeleton code in `finetuning_utils.py` and complete the TODOs. You may use `sklearn` to compute the evaluation metrics (accuracy, f1 score, precision, and recall) and a pretrained RoBERTa model from `transformers` to initialize the model.
6. Read the skeleton code in `run_hyperparameter_search.py` and complete the TODOs. (**Hint:** You may find the `hyperparameter_search()` method with the `search_alg` parameter set to `BayesOptSearch()` from the Hugging Face `Trainer` interface helpful for this section. See the hyperparameter_search documentation and the BayesOptSearch documentation for more details.) Although we will be tuning the learning rate, you will need to set some reasonable values for the other hyperparameters. Some good values to start with are:
    - Number of training epochs = 3
    - Training batch size = 8
    - Learning rate: **search between 1e-5 and 5e-5 for at least 5 trials**

    However, **be careful not to run too many trials** - this may overload the NYU HPC GCP cluster and take too long to finish. **Note: You may use objectives other than the eval_loss if you prefer.**

**Hint 1:** Also be sure to store model checkpoints in your scratch directory (located at `/scratch/${USER}/`) rather than your home directory, since the home directory quotas on NYU HPC GCP are quite low.
**Hint 2:** Use the `compute_objective` parameter in `hyperparameter_search()` to specify which of the evaluation metrics you'd like to use as your search objective. Note that it takes a function as its value. See the documentation linked above for more details.

## Running your code on NYU HPC GCP

*10 points*

You will need to run your hyperparameter search as a SLURM batch job on the NYU HPC GCP Cluster (Cloud Bursting via NYU Greene HPC Cluster). To do this, you will need to set up a Singularity container and write a SLURM job submission script that sends your Singularity container, Conda environment, and Python script to the job scheduler.

7. Read the skeleton code in `run_hyperparameter_search.slurm` and complete the TODOs.
8. Run `sbatch run_hyperparameter_search.slurm` on a NYU HPC GCP node to submit your batch job to the SLURM scheduler. This may take up to a few hours to complete, depending on how quickly your job is allocated resources.

## Reporting results

*20 points*

Report the evaluation metrics and tuned hyperparameters of your best run. Were there any other models that had higher loss but better evaluation accuracy or f1 score? Did the objective value vary a lot across runs?

## Extra credit

*10 points max.*

Modify your hyperparameter search to use other types of search algorithms, such as random/grid search. (For info on some other search algorithms you can try, see the Ray Tune documentation.) Add your findings to your report and discuss the advantages or disadvantages of the other algorithms you tried, in comparison with Bayesian optimization.