BCR Reproduction Guidelines

If you have any issues reproducing steps in this process and are unfamiliar with code, we strongly advise using an LLM to help troubleshoot issues. A great amount of the code relies on obscure Command Line Interface (CLI) arguments for now, so it can be a bit fiddly!

Experimental System Specs: Dell Latitude 5411. Intel i7-10850H (6C/12T, up to 5.1GHz), 32GB DDR4 RAM, NVIDIA MX250 2GB GPU, 512GB NVMe SSD, Windows 11 Pro. On this system, a full-sweep neuron intervention pass on 140 prompts takes around an hour.

Environment: We used Virtual Studio in Windows 11, Python Interpreter 3.11.9.

Reproducing the BCR Paper Results

This guide explains how to use the scripts in this repository to reproduce the core findings of the "Baseline Collapse Recovery Mapping via Neuron Intervention (BCR)" paper, focusing on the experiments with GPT-2 Small and Neuron 11:373.

The process involves three main steps:

1. **Capture:** Run the model with interventions, generating and saving text outputs.
2. **Extract:** Consolidate the generated text and metadata into a structured format.
3. **Analyze:** Detect baseline collapse, identify improvements, and generate the report/data for the paper's figures.

Prerequisites

- Python `3.11.9`
- PyTorch (`pip install torch`)
- TransformerLens (`pip install transformer_lens`)
- Pandas (`pip install pandas`)
- NumPy (`pip install numpy`)
- (Optional but recommended) tqdm (`pip install tqdm`) for progress bars.
- Access to the structured prompt file used in the paper (e.g., `epistemic_certainty_prompt_grid_template.txt`).
    - In a `promptsets/` directory relative to where you run the scripts or that you provide the full path.
- A directory to store experiment results (e.g., `experiments/`).

Step 1: Capture Intervened Outputs

This step uses `capture_intervened_activations.py` to run GPT-2 Small with the specified prompts, apply the neuron clamping interventions to Layer 11, Neuron 373, and save the generated text for each condition.

Key Script: `capture_intervened_activations.py`

Important Parameters:

- `--prompt_file`: Path to your structured prompt file (e.g., `promptsets/epistemic_certainty_prompt_grid_template.txt`).
- `--experiment_base_dir`: Directory where the output run folder will be created (e.g., `./experiments`).
- `--layer`: Set to `11`.
- `--target_neuron`: Set to `373`.
- `--sweep_values`: **Crucially**, provide the comma-separated list of clamp values used in the paper, **including `None` for the baseline**. For reproduction use: `-100,-50,-25,-10,-5,-3,0,None,3,5,10,25,50,100`.
- `--generate_length`: Number of tokens to generate (In our runs, we used `50`).
- Script defaults to greedy if `top_k` is unset, so for reproducibility do not set this value.

Example single-line command (after opening 7.6 folder into VS):

```
python capture_intervened_activations.py --prompt_file promptsets/epistemic_certainty_prompt_grid_template.txt --experiment_base_dir experiments --layer 11 --target_neuron 373 --sweep_values "-100,-50,-25,-10,-5,-3,0,None,3,5,10,25,50,100" --generate_length 50
```

When you run that command, inside `experiments/`, it'll create **one new folder** named something like:

`INT_intervened_L11N373_epistemic_certainty_prompt_grid_template_20250427_1715`

The exact name varies a bit because the **timestamp** is baked in automatically (using your computer's clock). Your folder may also have a Run ID attached if you use that prefix. This is to help identify multiple runs.

Inside *that* run folder, the structure looks like this:

```
experiments/
└──
INT_intervened_L11N373_epistemic_certainty_prompt_grid_template_<times
tamp>/
    └── capture/
        ├── logs/
        │   └── run_log_intervened.md  # Full log of the run (very
useful)
        ├── metadata/
        │   └── run_metadata_intervened.json # JSON with all run
settings, counts, etc.
        ├── text_outputs/
        │   └── *.txt      # 1 text file per (prompt x sweep value)
        └── vectors/
            └── intervened_vectors.npz    # Mean latent vectors for
analysis
```

Quick sense of what's *inside* each:

- **text_outputs/** → Pure generated text for every `(prompt, sweep)` pair. (e.g., "core_id=home_type=declarative_level=2_sweep=5.txt")
- **vectors/** → NumPy compressed `.npz` file storing mean MLP post-activations across generated tokens for each sweep. (this is what you feed into SRM or further analysis, not required for this reproduction)
- **logs/** → Markdown file showing what happened, token counts, any warnings, etc.
- **metadata/** → Structured run info, sweep values, prompt file paths, all nicely serialized.

If all goes well, at the end the terminal will print something like:

```
Text outputs saved: 2240, Failed: 0
```

```
Script finished. Results are in top-level directory: experiments/etc
Capture outputs (vectors, logs, metadata, text_outputs) are within:
capture/etc
```

"Not all lights guide you home.” (4o)

Step 2: Extract and Consolidate Text Data

This step uses `extract_text_from_logs.py` to parse the log file generated in Step 1, associate the generated text with its corresponding prompt and intervention value, and save everything into a single Tab-Separated Value (TSV) file.

Key Script: `extract_text_from_logs.py`

Important Parameters:

- `--experiment_base_dir`: The *same* directory used in Step 1 (e.g., `./experiments`). The script will likely prompt you to select the specific run directory created in Step 1.
- `--promptsets_dir`: The directory containing your original prompt file (e.g., `promptsets/`). The script will likely prompt you to select the correct prompt file.
- `--output_file`: (Optional) Specify a name for the output TSV. Defaults to `[run_dir_name]_extracted.tsv` in the current directory.

The files `extract_text_from_logs.py` and `analyze_textv2.py` **both prompt interactively** if args aren't supplied.

Example Command (assuming interactive selection):

```
python extract_text_from_logs.py --experiment_base_dir ./experiments
--promptsets_dir ./promptsets
```

Expected Output:

- A TSV file (e.g., `INT-XXXXXXXX-XXXXXX..._extracted.tsv`) containing columns like `core_id`, `type`, `level`, `sweep`, `prompt`, and `output`. This file consolidates all the necessary text data for the final analysis.

Step 3: Analyze Text, Detect Collapse/Recovery, and Generate Report

This step uses `analyze_textv2.py` to process the TSV file from Step 2. It calculates text metrics, applies heuristics to detect collapsed outputs (identifying collapsed baselines where `sweep=None`), determines if interventions led to non-collapsed outputs ("improvement"), and generates the final analysis report and data.

Key Script: `analyze_textv2.py`

Important Parameters:

- `--input_tsv_file`: Path to the TSV file generated in Step 2 (e.g., `INT-XXXXXXXX-XXXXXX..._extracted.tsv`). If omitted, it will prompt for selection within the `--search_dir`.
- `--search_dir`: Directory to search for the TSV file (e.g., `.`).
- `--report_improvements`: **Include this flag** to generate the detailed report showing collapsed baselines and improved intervention outputs (matching Appendix 3).
- `--output_report_file`: (Optional) Specify a path for the Markdown report (e.g., `results/bcr_improvement_report.md`). Defaults to `[input_base]_improvement_report.md`.
- `--output_metrics_csv`: (Optional) Specify a path to save a CSV containing detailed metrics, collapse flags (`is_collapsed`), and improvement flags (`intervention_improves_baseline`) for *every* row (e.g., `results/bcr_full_metrics.csv`). Defaults to `[input_base]_metrics.csv`. **This CSV is needed to plot the graph on page 18.**

Example Command: Note to replace the highlighted text.

```
python analyze_textv2.py --input_tsv_file INT-XXXXXXXX-XXXXXX..._extracted.tsv
--report_improvements --output_report_file results/bcr_improvement_report.md
--output_metrics_csv results/bcr_full_metrics.csv
```

Expected Output:

1. **Console Output:** Summaries of average metrics per sweep and potentially delta analyses.

2. **Improvement Report (** A Markdown file detailing the 8 prompts identified in the paper where the baseline collapsed, showing the collapsed output and the corresponding "recovered" outputs from specific interventions. **Verify this against Appendix 3.**
3. **Metrics CSV (** A CSV file containing the data for *all* prompts and sweeps, including the crucial `is_collapsed` and `intervention_improves_baseline` boolean columns.

Step 4: Verify Results and Plot Graph

1. **Check the Report:** Open the generated Markdown report (`*_improvement_report.md`) and compare the listed examples (collapsed baselines vs. improved interventions) against those presented in Appendix 3 of the paper. They should match.
2. **Plot the Improvement Rate Graph:**
   - Load the generated Metrics CSV (`*_metrics.csv`) using a tool like Pandas in Python or spreadsheet software.
   - Follow the procedure outlined by GPT-4oMini on page 18 of the paper:
     - Identify the unique prompts where the baseline (`sweep == 'None'`) was collapsed (`is_collapsed == True`).
     - Filter the DataFrame to include only rows corresponding to these specific prompts.
     - Group the filtered data by the `sweep` value (convert numeric sweeps to numbers for sorting).
     - For each sweep value, calculate the fraction of prompts where the intervention improved (`intervention_improves_baseline == True`).
     - Plot this fraction against the sweep value.
   - Compare the generated plot against the "Fraction of Collapsed Prompts Improved by Neuron Clamp" graph on page 18 of the paper. The shape and key points should align.

By following these steps, you should be able to replicate the primary text-based results and analyses presented in the BCR paper using the provided scripts.

Optional Step 5: Interactive Exploration with GUI Tools

Beyond the batch processing scripts used for the main paper results, this repository also includes two experimental Graphical User Interface (GUI) tools for interactively exploring neuron interventions in GPT-2 Small:

1. **`GPT2All.py`:** This tool allows you to define neuron clamps (Layer, Neuron Index, Value) and generate text. Crucially, the intervention hook in this version applies the specified clamp value to the target neuron's activation across **all token positions** in the sequence during each step of the generation. This mirrors the behavior of the "all-token" sweeps used for the main results in the paper, allowing you to visually verify specific examples.
2. **`GPT2Last.py`:** This tool offers a more fine-grained intervention. It applies the clamp **only to the activation influencing the *very next* token prediction** (i.e., the activation corresponding to the *last* token in the current sequence). This allows for exploring the immediate impact of an intervention on the next token choice, aligning with the "last-token" sweeps discussed on page 17 of the paper as a potential refinement.

Purpose:

- **Verification:** Manually reproduce specific prompt/clamp examples from the paper (e.g., input a prompt from Appendix 2/3, add the corresponding clamp like L11 N373 with value +5, and generate) to observe the recovery effect directly.
- **Exploration:** Experiment with different prompts, clamp values, multiple simultaneous clamps, or compare the output differences between the `All` and `Last` token intervention strategies.

Usage:

1. Ensure you have the necessary libraries installed (PyTorch, TransformerLens, Tkinter is usually included with Python).
2. Run the desired GUI script from your terminal:

python GPT2All.py or **python GPT2Last.py**

3. The application window will appear. Wait for the "Model loaded..." status message.
4. Enter your prompt in the input box.
5. Use the controls to set generation parameters (Temperature, Top-K, Max New Tokens). `GPT2All.py` also has a "Greedy" checkbox.

6. *(Optional)* Use the right-hand panel (or "Developer Mode" in `GPT2Last.py`) to add neuron clamps:
    - Specify the Layer (0-11), Neuron index (0-3071), and the Value to clamp it to.
    - Click "Add" to add a specific clamp.
    - Use "Quick Add" / "All Layers" to apply the same clamp to all layers.
    - Select clamps in the list and use "Remove Selected" / "X" button or "Clear All" to manage them (potentially may not work in current version, check for errors).
7. Click "Generate" / "Submit" or press Enter in the input box to start generation.
8. The output will appear in the chat history, along with information about the generation settings and any active clamps.

Important Caveat:

Please be aware that the hook implementations within these GUI tools are **experimental and potentially fragile**. They were developed for exploration and for now are considerably less robust or consistent than the batch processing scripts, especially under complex clamping scenarios or during longer generations. You will encounter occasional inconsistencies or errors. They serve as a helpful visual aid and exploration platform but the batch scripts (`capture_intervened_activations.py` etc.) remain the definitive method for reproducing the paper's core quantitative results.

Thank you for testing and/or showing interest!