microgpt_trace_en_v2.py — User Guide

Oct 24, 2026 · @YOSHIO

Overview

microgpt_trace_en_v2.py is Andrej Karpathy's microgpt with display code added, so you can see every intermediate number in one training step of a tiny GPT.

The original microgpt trains and runs a GPT in about 200 lines of pure Python, with no libraries. It learns from about 32,000 English first names and then generates new, made-up names.

The model has 1 layer, 4 attention heads, an embedding width of 16, and 4,192 parameters in total. The vocabulary is 27 tokens: the letters a–z plus a BOS (beginning/end of sequence) token.

The added code does two things:

- **Trace**: for selected training steps, it prints every matrix computed in the forward pass, the gradient of Wq after backpropagation, and the Adam update of Wq.
- **Weight dump**: before training and at regular intervals, it writes out all weight matrices.

The training and inference code itself is unchanged. Every added line is marked with `# [added]`, so it can be read side by side with the original.

Requirements and how to run

You need Python 3 only; no packages are required.

1. Put microgpt_trace_en_v2.py in an empty folder.
2. Run it from a terminal in that folder:

```
python3 microgpt_trace_en_v2.py
```

On the first run, the script downloads the names list (`names.txt` from Karpathy's makemore repository) and saves it as `input.txt`. Later runs use that file, so no network is needed. To run offline from the start, place your own `input.txt` with one word per line.

The console shows the progress (`step 123 / 1000 | loss 2.5432`), a line each time an output file is written, and finally 20 generated names. All 1,000 steps take a few minutes on an ordinary PC (approximate), because every number is a Python object.

The random seed is fixed (`random.seed(42)`), so every run with the same `input.txt` gives exactly the same numbers.

Settings

All settings are variables near the top of the script, just after the `Value` class. Edit them in the file before running.

Variable	Default	Meaning
`TRACE_STEP`	`0`	First training step to trace, counted from 0 (0 = step 1).
`TRACE_EVERY`	`100`	Also trace every this many steps after `TRACE_STEP` (100 → steps 1, 101, 201, …, 901). `0` = trace `TRACE_STEP` only.
`TRACE_FILE`	`'microgpt_trace_out.txt'`	Base name of the trace files. `None` = print the trace on the console instead.
`PAUSE`	`True`	Console mode only: stop after each stage and wait for Enter. Typing `a` + Enter shows the rest without stopping. Ignored when writing to files.
`WEIGHT_EVERY`	`100`	Write all weights before training, every this many steps, and after the last step. `0` = no weight files.
`WEIGHT_FILE`	`'microgpt_weights_out.txt'`	Base name of the weight files.

The model size (`n_embd` , `n_head` , …), `num_steps = 1000` and the Adam settings are Karpathy's originals. The trace assumes `n_layer = 1` : it shows layer 0 only.

For a classroom demonstration, set `TRACE_FILE = None` and `TRACE_EVERY = 0` . The script then walks through step 1 on screen, one stage per Enter.

Output files and numbering

With the default settings, one run writes 21 files: 10 trace files and 11 weight files. Each write goes to a new file, and a serial number starting at 1 is inserted before `.txt` .

File	Contents
`microgpt_trace_out_1.txt`	Trace of training step 1
`microgpt_trace_out_2.txt` … `_10.txt`	Traces of steps 101, 201, …, 901
`microgpt_weights_out_1.txt`	Weights before training (initial random values)
`microgpt_weights_out_2.txt` … `_11.txt`	Weights after steps 100, 200, …, 1000

The serial number counts writes; it is not the step number. The console line for each write gives both, for example `Wrote the computation of training step 101 to microgpt_trace_out_2.txt.` The first line inside each file also states the step.

Numbering restarts at 1 on every run, so a second run in the same folder overwrites the earlier files. Use a new folder for each run you want to keep.

How to read a trace file

A trace file follows one name through one training step, in the order the code computes it: sections 0–6 are the forward pass, 7 is backpropagation, and 8 is the Adam update.

In most tables each row is one position, labelled `position:input character` (for example `3:h`), and the columns `d0` – `d15` are the 16 embedding dimensions. A name of L letters gives n = L + 1 positions, because BOS is prepended. Each table title ends with its size, such as `(7 rows x 16 cols)`. Tables wider than 16 columns are transposed, as their titles state.

Section	What it shows	Size
0	The name, its token ids, and each input → target pair	—
1a, 1b	Rows of `wte` (token embedding) and `wpe` (position embedding)	n × 16
1c	RMSNorm of their sum: the input x to the block	n × 16
2a–2c	Query, key and value: q = Wq·RMSNorm(x), and likewise k, v	n × 16
3a, 3b	Per head (0–3, 4 dims each): scores q·k/√4 and their softmax weights	n × n

Section	What it shows	Size
4a	The four head outputs concatenated	n × 16
4b, 4c	After Wo; then plus x (residual) = attention block output	n × 16
5a	MLP hidden layer ReLU(W1·RMSNorm(x)), transposed	64 × n
5b, 5c	After W2; then plus 4c (residual) = final layer output	n × 16
6a, 6b	Logits and their softmax probabilities for the next character, transposed	27 × n
6c	Probability p of each correct character, −ln p, and the mean loss L	—
7a	Gradient ∂L/∂Wq after `loss.backward()`	16 × 16
8a–8c	Wq before the update, the change (after − before), Wq after	16 × 16

A `·` in sections 3a and 3b marks a future position whose key does not exist yet; this is the causal mask. A `·` in 5a marks a component set to 0 by ReLU.

Section 6c also prints ln 27 ≈ 3.2958, the loss of a uniform guess, for comparison. With the defaults, the loss is 3.3660 in step 1, close to that value, and 2.0881 in step 901.

Section 8b of step 1 shows every change as ± the learning rate (0.01). In Adam's first step, $\hat{m}/\sqrt{\hat{v}}$ equals the sign of the gradient, so only the direction counts. From step 101 on, the changes vary in size.

Only Wq is shown in sections 7 and 8, as one example. All other parameters receive gradients and updates in the same way; their values appear in the weight files.

How to read a weight file

Each weight file holds all 4,192 parameters as nine labelled tables, values to three decimals. Comparing the same table across files shows how training changes the weights.

Table	Rows × cols	Rows	Columns
`wte` token embedding	27 × 16	character (a–z, BOS)	`d0` – `d15`

Table	Rows × cols	Rows	Columns
`wpe` position embedding	16 × 16	position `p0` – `p15`	`d0` – `d15`
`Wq` , `Wk` , `Wv`	16 × 16 each	component of q, k or v	input component `d0` – `d15`
`Wo`	16 × 16	output component `d0` – `d15`	component `c0` – `c15` of the concatenated heads
`W1` (`mlp_fc1`)	64 × 16	hidden component `h0` – `h63`	input component
`W2` (`mlp_fc2`), transposed	64 × 16	hidden component `h0` – `h63`	output component
`lm_head`	27 × 16	character	`d0` – `d15`

In the first file all values are random draws from a normal distribution with standard deviation 0.08. With the default settings, rows `p14` and `p15` of `wpe` stay exactly at their initial values: none of the 1,000 names used reaches those positions, so they never receive a gradient.

Notes and limitations

- **Overwriting**: each run starts numbering at 1 and overwrites earlier output in the same folder.
- **One layer only**: the trace and the weight dump read `layer0` . If you increase `n_layer` , training still works, but later layers are not shown.
- **Text width**: tables are plain text, up to 132 characters wide. View them in an editor with a monospaced font and line wrapping turned off.
- **Speed**: tracing adds little time. Training itself is slow because microgpt computes every scalar as a Python object, by design.
- **Japanese version**: microgpt_trace_v2.py is the same program with Japanese messages and comments. Both give identical numbers.

The original microgpt is by Andrej Karpathy (@karpathy); the names data come from his makemore repository.