THE CHINESE ROOM

Instructions for Manual Inference

Using These Volumes

WEIGHTS

The Complete Parameters of GPT-1

weights.press

Prologue

You are in a room. The door is closed. You do not understand English, or any language. You have no sense of grammar, no feel for meaning, no intuition about what words should follow other words. But you have eighty books, a pencil, a large supply of paper, a calculator, and the instructions that follow.

A slip of paper comes through the slot in the door. On it is a sequence of English words. You will follow these rules, exactly as written, manipulating nothing but numbers. When you are finished, you will write a word on a new slip of paper and push it back through the slot.

The word will make sense. It will be the right word, or at least a plausible one. You will have no idea why.

This is John Searle's Chinese Room, made real. Not with Chinese characters but with English, not with an imaginary rule book but with the actual learned parameters of a neural network. Every number you look up in these volumes is a real value, adjusted through training on thousands of books. The procedure described here is not a simplification or a metaphor. It is the exact computation that GPT-1 performs, expressed as rules a person can follow.

If you follow these rules correctly, you will produce the same output as the machine.

What You Will Need

- All 80 volumes of WEIGHTS
- The Vocabulary Table (Appendix A) — a numbered list of 40,478 word-pieces and their token IDs
- A pencil and eraser
- At least 2,000 sheets of blank paper
- A flat working surface large enough to hold several open volumes simultaneously
- A calculator (or the willingness to perform arithmetic by hand)
- Patience measured not in hours but in months

How the Books Are Organized

Each page contains 2,000 numbers arranged in 16 columns and 125 rows. Numbers are printed to six decimal places. Most fall between −1.0 and 1.0, though some are larger.

The header at the top of each page tells you three things: which parameter group the numbers belong to (for example, `h.3.attn.c_attn.weight`), the offset of the first number on that page within its group, and the volume and page number.

Numbers flow left to right, top to bottom, continuing from page to page. When one parameter group ends, the next begins immediately — possibly in the middle of a page.

The Volume Reference at the back of Volume 1 lists every parameter group and which volume contains it. Use it as your index.

Notation

A **vector** is an ordered list of numbers. You will mostly work with vectors of length 768. Write them in a column on your paper, numbered 1 through 768.

A **matrix** is a grid of numbers with rows and columns. When these instructions say "the matrix at `h.0.attn.c_attn.weight`," find that parameter group in the books.

Element-wise means: do the same operation to each number in the list, independently.

When the instructions say "add vector A to vector B," add the first number of A to the first number of B, the second to the second, and so on.

Procedures

Learn these procedures first. You will use them hundreds of millions of times.

Procedure D — Dot Product

Given two vectors A and B, both of length N:

1. Set result to 0.
2. For each position i from 1 to N: multiply A[i] by B[i] and add the product to result.
3. The final value of result is the dot product.

For vectors of length 768, this requires 768 multiplications and 767 additions.

Procedure W — Weighted Accumulation

This is how you multiply a vector by a matrix as the numbers are laid out in these books. Given a vector x of length R and a matrix printed in the books with R rows and C columns:

1. Prepare a fresh sheet. Write C zeros in a column. This is your accumulator.
2. Open the book to the first row of the matrix. Read the C numbers.
3. Multiply each of those C numbers by x[1]. Add each product to the corresponding position in your accumulator.
4. Move to the second row. Multiply each number by x[2]. Add to the accumulator.
5. Continue through all R rows.
6. The accumulator now holds the output — a vector of length C.

For the attention weight matrix (768 rows, 2,304 columns), this requires 1,769,472 multiplications and the same number of additions. The matrix spans approximately 885 pages.

Procedure N — Layer Normalization

Given a vector x of length 768, a weight vector w, and a bias vector b (each also length 768):

1. Compute the mean: add all 768 values in x, then divide by 768.
2. Subtract the mean from every element. Call the result x′.
3. Compute the variance: square each value in x′, add them all, divide by 768.
4. Add 0.00001 to the variance (to prevent dividing by zero). Take the square root. This is the standard deviation, s.
5. Divide each element of x′ by s.
6. For each position i: multiply x′[i] by w[i], then add b[i]. This is the output.

Procedure G — GELU Activation

Given a single number x:

1. Compute c = x multiplied by x multiplied by x (that is, x cubed).
2. Compute inner = 0.7978846 multiplied by the quantity (x + 0.044715 multiplied by c).
3. Compute the hyperbolic tangent of inner:

• If inner is greater than 4: the tanh is 1.0.

• If inner is less than −4: the tanh is −1.0.

• Otherwise: compute p = e raised to the power (2 multiplied by inner). The tanh is (p − 1) divided by (p + 1).

4. The result is 0.5 multiplied by x multiplied by (1 + the tanh from step 3).

As a shortcut: for large positive x, GELU(x) is approximately x. For large negative x, it is approximately 0.

Procedure S — Softmax

Given a vector z of length N:

1. Find the largest value in z. Call it m.
2. Subtract m from every element (this prevents overflow).
3. Raise e to the power of each element. Write these as a new vector.
4. Add up all the values in this new vector. Call the sum T.
5. Divide each value by T. The result is a vector of positive numbers that sum to 1.0.

The Rules

Rule 0 — Receive the Input

A slip of paper comes through the slot with a sequence of words. Using the Vocabulary Table, convert each word to its token ID — a number between 0 and 40,477.

Some words split into multiple pieces. The word “understanding” might become two tokens: “under” (one ID) and “standing” (another ID). Follow the Vocabulary Table exactly. You do not need to know why a word splits as it does.

Write down the token IDs in order. Call them t_1, t_2, … t_n, where n is the total number of tokens. The maximum is 512.

Rule 1 — Look Up the Token Embedding

The token embedding matrix occupies Volumes 1 through 22. It contains 40,478 rows, each 768 numbers long. Row 0 begins on page 9 of Volume 1 (after the companion guide). The rows are printed consecutively — row 0 is the first 768 numbers, row 1 is the next 768, and so on.

For each token t_k in your sequence:

1. Compute the starting position: t_k multiplied by 768. This is the offset within the token embedding.
2. Find that offset in the books, beginning from the start of `tokens_embed.weight`.
3. Read 768 consecutive numbers. Write them on a fresh sheet of paper.

4. Label this sheet “e(k)” — the embedding of token k.

You will return to these same pages later, in the final step.

Rule 2 — Add the Position Embedding

The position embedding matrix starts in Volume 22, in the parameter group labeled `positions_embed.weight`. It has 512 rows of 768 numbers each.

For each token at position k (starting from 0):

1. Find row k of the position embedding. The offset within this group is k multiplied by 768.
2. Read 768 numbers.
3. On the sheet labeled “e(k),” add each of these 768 numbers to the corresponding number already written there.
4. Cross out the old values and write the new ones. Relabel this sheet “x(k).”

You now have n vectors, each of length 768. These are your working state.

Rule 3 — Transformer Block 0

You will now pass every working vector through the first of twelve transformer blocks. The parameters for block 0 are in the groups whose names begin with `h.0`.

This is the longest rule. It has twelve sub-steps. Every subsequent block (Rules 4 through 14) follows the same sub-steps with different parameters.

3a — First Layer Normalization

Find the groups `h.0.ln_1.weight` (768 numbers) and `h.0.ln_1.bias` (768 numbers).

For each token k, apply Procedure N to x(k) using these weights and biases. Write the result on a new sheet labeled “x-norm(k).”

Do not discard x(k). You will need the original values in step 3h.

3b — Compute Queries, Keys, and Values

Find `h.0.attn.c_attn.weight` — a matrix with 768 rows and 2,304 columns, containing 1,769,472 numbers. Also find `h.0.attn.c_attn.bias` — 2,304 numbers.

For each token k:

1. Apply Procedure W: use x-norm(k) (length 768) as the vector, and the c_attn weight as the matrix. The output has 2,304 numbers.
2. Add the bias: for each position j from 1 to 2,304, add the j-th bias value.

3. Split the result into three vectors of 768 numbers each:

• Positions 1 through 768: the Query vector. Label it “Q(k).”

• Positions 769 through 1,536: the Key vector. Label it “K(k).”

• Positions 1,537 through 2,304: the Value vector. Label it “V(k).”

3c — Divide Into Attention Heads

GPT-1 has 12 attention heads. Each head works with 64 numbers (768 divided by 12).

For each head h (numbered 1 through 12), the relevant portion of Q, K, and V is positions ((h – 1) × 64 + 1) through (h × 64). Head 1 uses positions 1–64. Head 2 uses 65–128. And so on through head 12 (positions 705–768).

On separate sheets, write out each head’s portion. Label them “Q(k,h),” “K(k,h),” “V(k,h)” — each a vector of 64 numbers.

3d — Compute Attention Scores

For each head h, and for each token k in your sequence:

1. For every token j from 1 to n:

• If j is greater than k: write “minus infinity” as the score. (Each token may only look at tokens at its own position or earlier.)

• Otherwise: compute the dot product of Q(k,h) with K(j,h) using Procedure D (64 multiplications). Divide the result by 8 (the square root of 64).

2. You now have n scores for token k in head h. Apply Procedure S (softmax). The result is n attention weights that sum to 1.0. Label them “a(k,j,h)” for j = 1 to n.

3e — Compute Weighted Values

For each head h and each token k:

1. For each position i from 1 to 64: multiply a(k,1,h) by V(1,h)[i], then a(k,2,h) by V(2,h)[i], and continue for all n tokens. Sum all these products. Write the result as “out(k,h)[i].”

2. You now have a vector out(k,h) of length 64.

3f — Reassemble the Heads

For each token k, concatenate the outputs of all 12 heads into a single vector of 768 numbers: positions 1–64 from out(k,1), positions 65–128 from out(k,2), and so on through positions 705–768 from out(k,12). Label this “attn-concat(k).”

3g — Attention Output Projection

Find `h.0.attn.c_proj.weight` (768 rows, 768 columns — 589,824 numbers) and `h.0.attn.c_proj.bias` (768 numbers).

For each token k:

1. Apply Procedure W: use attn-concat(k) as the vector, the c_proj weight as the matrix. Output length: 768.
2. Add the bias.
3. Label the result "attn-out(k)."

3h — First Residual Connection

For each token k, add attn-out(k) to the original x(k) — the values you saved before normalization. For each position i: x(k)[i] = x(k)[i] + attn-out(k)[i]. Update x(k) in place.

3i — Second Layer Normalization

Find `h.0.ln_2.weight` and `h.0.ln_2.bias` (768 numbers each). For each token k, apply Procedure N to the updated x(k). Write the result as "x-norm2(k)." Keep the original x(k).

3j — Feed-Forward Network: Expand

Find `h.0.mlp.c_fc.weight` (768 rows, 3,072 columns — 2,359,296 numbers) and `h.0.mlp.c_fc.bias` (3,072 numbers).

For each token k:

1. Apply Procedure W: use x-norm2(k) as the vector, the c_fc weight as the matrix. Output length: 3,072.
2. Add the bias.
3. Apply Procedure G (GELU) to each of the 3,072 numbers independently.
4. Label the result "ffn-mid(k)."

This is the most laborious sub-step. The matrix alone spans approximately 1,180 pages. Then you must compute the GELU activation function 3,072 times.

3k — Feed-Forward Network: Compress

Find `h.0.mlp.c_proj.weight` (3,072 rows, 768 columns — 2,359,296 numbers) and `h.0.mlp.c_proj.bias` (768 numbers).

For each token k:

1. Apply Procedure W: use ffn-mid(k) (length 3,072) as the vector, the c_proj weight as the matrix. Output length: 768.
2. Add the bias.

3. Label the result “ffn-out(k).”

3l — Second Residual Connection

For each token k: x(k)[i] = x(k)[i] + ffn-out(k)[i]. Update x(k) in place. This completes transformer block 0. Each x(k) is now ready for the next block.

Rules 4 through 14 — Transformer Blocks 1 Through 11

Repeat every sub-step of Rule 3 for each of the remaining eleven blocks. Each block uses its own parameters: Block 1 uses groups beginning with `h.1`, Block 2 uses `h.2`, through Block 11, which uses `h.11`.

Consult the Volume Reference in Volume 1 to find which volumes contain each block’s parameter groups. Each block’s parameters span approximately 5 volumes.

The structure is identical every time: normalize, attend, add residual, normalize, expand, compress, add residual. Only the numbers change.

Rule 15 — Compute Output Logits

After all twelve blocks, you have a final set of vectors x(1) through x(n). You need only the last one: x(n).

Now return to the token embedding matrix — the same numbers you used in Rule 1, in Volumes 1 through 22.

For each word w in the vocabulary (w = 0 to 40,477):

1. Find row w of the token embedding (768 numbers, at offset w multiplied by 768).
2. Compute the dot product of x(n) with this row, using Procedure D. This gives a single number: the logit for word w.

Write down all 40,478 logits. This step requires 40,478 dot products of length 768 — a total of 31,087,104 multiplications.

Rule 16 — Convert to Probabilities

Apply Procedure S (softmax) to the 40,478 logits. You now have a probability for each word in the vocabulary. They sum to 1.0.

Rule 17 — Produce the Output

Find the word with the highest probability. Look up its token ID in the Vocabulary Table to find the corresponding word-piece.

Write this word on a slip of paper. Push it through the slot in the door.

If the word is a partial word-piece (a fragment rather than a complete word), you may need to continue generating. To do so, append this new token to your input sequence and repeat from Rule 1 — but this time with n+1 tokens instead of n.

The Arithmetic

To process a single token through one transformer block:

Operation	Multiplications
`Attention projection (c_attn)`	`1,769,472`
`Attention scores (12 heads)`	`768`
`Weighted values (12 heads)`	`768`
`Output projection (c_proj)`	`589,824`
`Feed-forward expand (c_fc)`	`2,359,296`
`Feed-forward compress (c_proj)`	`2,359,296`
`Layer norms, biases, residuals`	`~10,000`
Total per block	**~7,090,000**

For all 12 blocks: approximately 85,000,000 multiplications.

Final output projection: approximately 31,000,000 multiplications.

Total for one token: approximately 116,000,000 arithmetic operations.

A person working carefully can perform about one multiplication every five seconds. Working eight hours a day with no breaks, that is 5,760 operations per day.

Processing a single token would take approximately **201 days**.

To generate a response of twenty words, you must process the growing sequence twenty times. Accounting for the increasing sequence length at each step, this comes to roughly **eleven years** of continuous work.

GPT-1 does this in about 30 milliseconds.

Epilogue

You have now pushed a word through the slot. If the rules were followed exactly — every multiplication, every addition, every lookup across 58,000 pages — it is the same word that GPT-1 would have produced. A word that continues the sentence plausibly. A word that might seem thoughtful, even creative.

You understand nothing about why this word is correct. You followed rules. You looked up numbers. You multiplied and added. You moved your pencil across paper. At no point did meaning enter the room.

In 1980, the philosopher John Searle proposed a thought experiment: a person locked in a room, manipulating Chinese characters according to a rule book, producing outputs that native Chinese speakers find perfectly fluent. Searle's argument was that the person does not understand Chinese, and therefore the system does not understand Chinese — no matter how convincing its outputs.

These eighty volumes make his thought experiment concrete. The rule book fits in a few pages. The lookup tables fill a shelf. The computation would take a decade.

The question Searle raised remains open. Is understanding happening somewhere in this process — in the patterns distributed across 117 million numbers, in the rules that connect them, in the room taken as a whole? Or is it happening nowhere at all, and the appearance of understanding is just that: an appearance?

These numbers do not answer the question. But they let you hold it in your hands.