

Building an LLM from Zero

Chapter 16: Temperature and Top-k

Module 3: Training and Generation

Truong (Jack) Luu · jackluu.io/book

Where we are

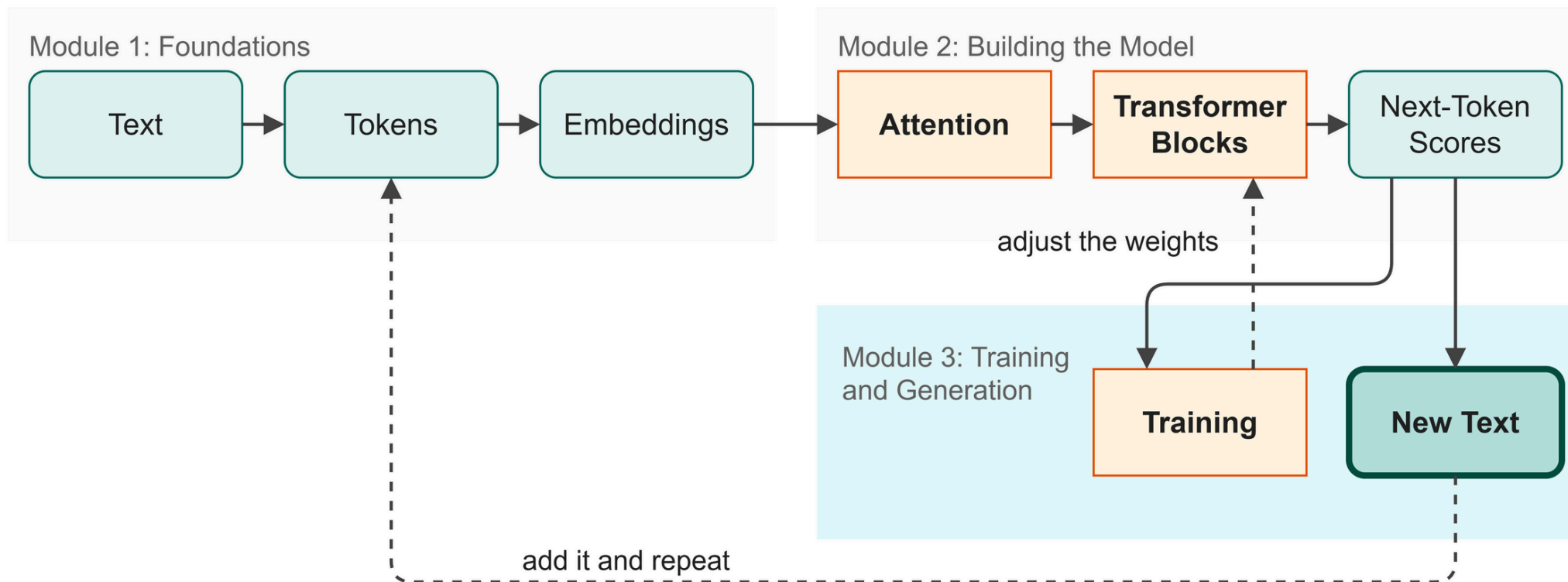


Figure 16.1: Where we are: we are tuning the text generation process.

Plain sampling gives variety, and sometimes nonsense. Today we add two dials that sit between the scores and the pick.

What this builds on (60-second recap)

- **Chapter 15** gave the model two rules: **greedy** (always the top score) and **sampling** (draw at random, weighted by probability)
- Greedy looped: "the shall the see the shall the see"
- Sampling varied, but wrote "greather" and "bagainVain" when a rare character got drawn

Next: keep the variety, cut the nonsense.

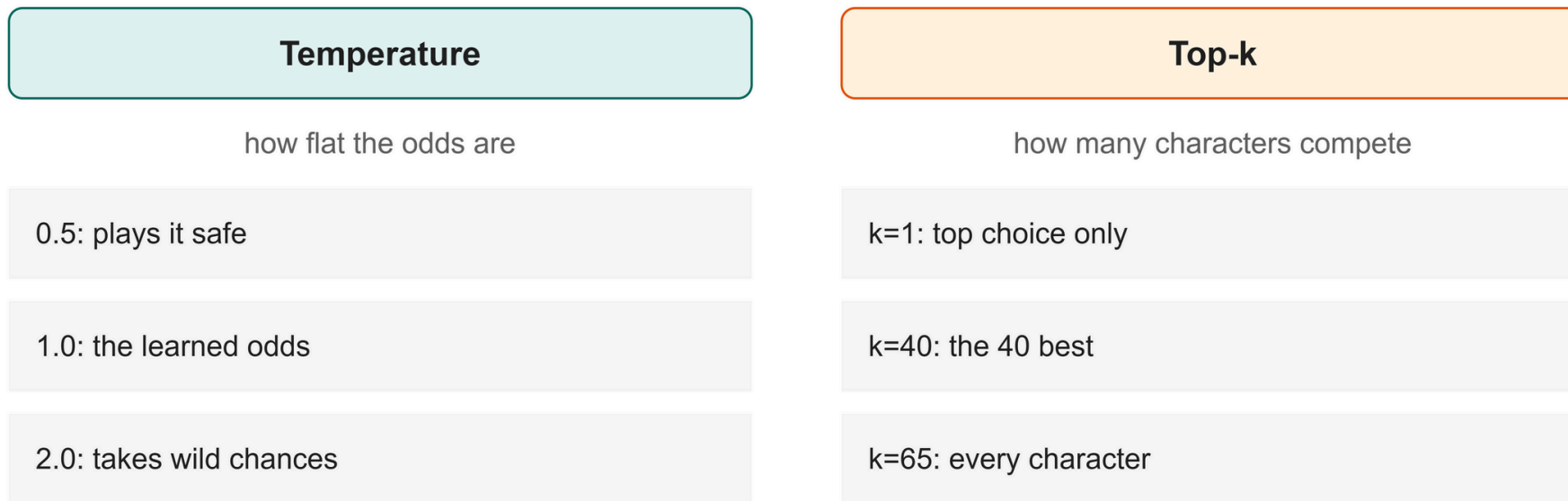
In this chapter you will

1. Use **temperature** to adjust how risky the model's choices are
2. Use **top-k** truncation to stop the model from picking truly bad characters
3. Combine the two controls

Words to know (each is glossed again where it first appears):

- **Temperature**: a number that scales the logits before they are turned into probabilities, controlling the randomness of the output
- **Top-k**: a limit that restricts the model to only sample from the k most likely next tokens, ignoring all others

Two controls between the scores and the pick



Temperature reshapes the odds. Top-k decides who is allowed to compete.

Figure 16.2: Temperature changes the shape of the odds; top-k changes how many characters stay in the running.

- **Temperature** reshapes the odds; **top-k** decides which characters may compete at all
- They are independent. In practice you set both

One dial for how bold, one dial for how many.

Temperature: divide the scores before softmax

```
$ python src/examples/ch16_temperature.py  
Saved chart: ch16_temperature.png
```

Figure 16.3: A plotted chart showing probabilities: low temperature sharpens the choice, high temperature flattens it.

Low, standard, high

Temperature is a simple math trick: divide all the logits (raw scores) by one number, then apply softmax.

- **Low (0.5):** dividing by a fraction stretches the scores apart. The top choice becomes overwhelmingly favored. The model plays it safe
- **T = 1.0:** standard sampling. The model uses its learned probabilities exactly as they are
- **High (2.0):** dividing by a large number squashes the scores together. Unusual choices become more likely. The model takes risks

Temperature does not change which character is likeliest. It changes how much that lead is worth.

Top-k: only the best k may compete

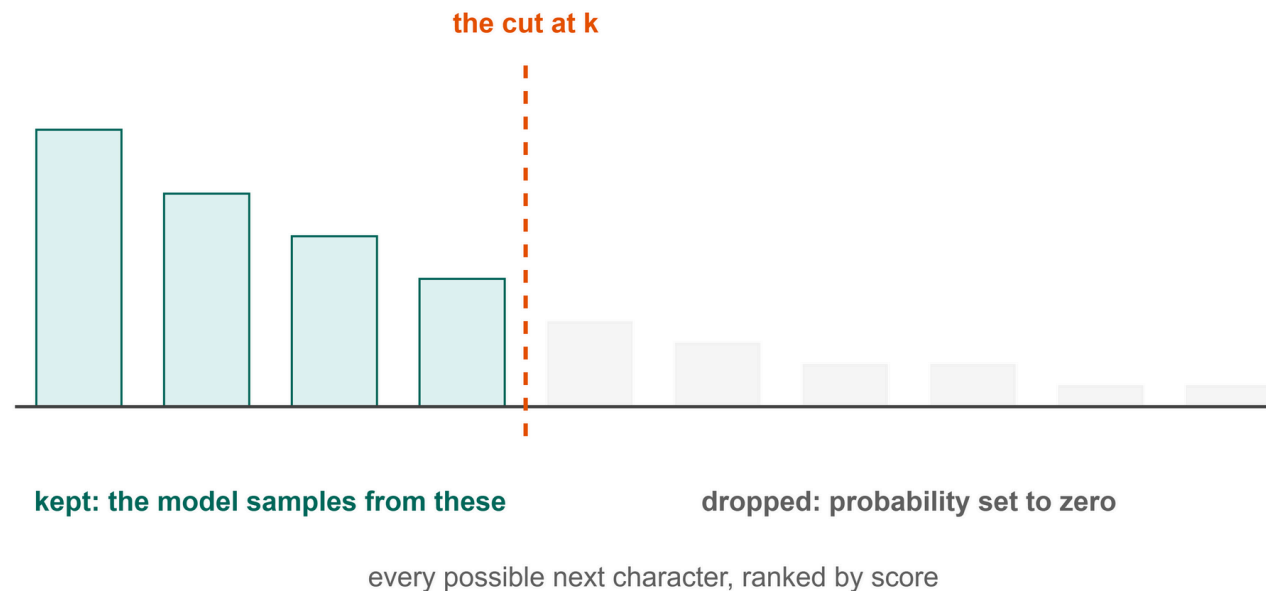


Figure 16.4: Top-k keeps the highest-scoring characters and zeroes the rest. Drawn here with ten characters and $k = 4$; the book's code uses 65 and $k = 40$.

- An absurd character keeps a tiny chance (say, 0.001%); over 200 characters, tiny chances add up
- With `top_k = 40`, keep the **40** highest scores and zero the other **25**

Top-k cuts off the long tail of bad choices.

In business: set the dials for the task

The house-style assistant uses different settings for different jobs.

- **Compliance documentation:** low temperature (0.5), to stay close to the safest boilerplate
- **Brainstorming marketing slogans:** higher temperature (0.9) with top-k (40), for surprising slogans that still make grammatical sense

The dials are how you tune one model to two very different tasks.

Code: temperature, then top-k, then softmax

File: `src/ch15_generate_sampling.py` (source files are numbered one lower than chapters).

```
# Temperature controls randomness: lower is less random
logits = logits / temperature

# Top-k sampling limits the choices to the k most likely tokens
if top_k is not None:
    threshold = logits.topk(top_k).values[:, -1, None]
    logits = logits.masked_fill(logits < threshold, float("-inf"))

probs = F.softmax(logits, dim=-1)
next_id = torch.multinomial(probs, num_samples=1)
```

- **Line 2** divides the scores by the temperature
- **Line 6** finds the score of the 40th best character: the threshold
- **Line 7** sets every score below the threshold to negative infinity

How the code flows

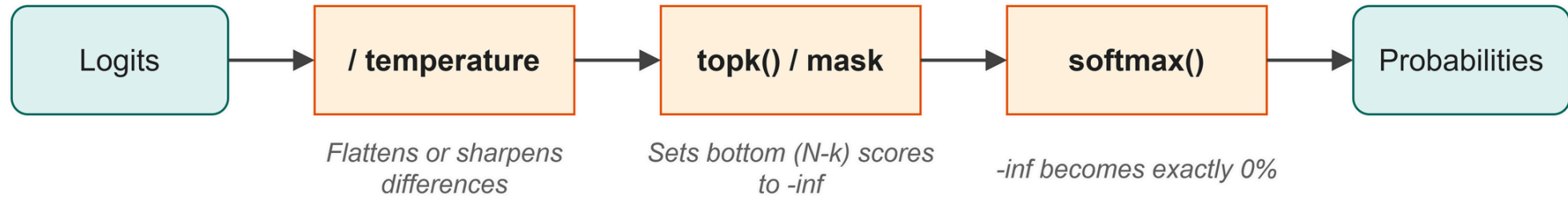


Figure 16.5: The code flow applies temperature first, then top-k, and finally softmax.

- `masked_fill` replaces every score below the threshold with `-inf`
- Softmax turns `-inf` into exactly `0`, so those characters can never be drawn

Negative infinity is how you tell softmax "never".

Run it: five settings, one prompt

```
$ python src/ch15_generate_sampling.py
```

Prompt: JULIET: and a new line. The first line the model wrote under each setting:

Setting	The model wrote
temp=0.5, top_k=40	My must grace prove a son the come on my lord.
temp=0.8, top_k=40	That will, my call thee for thee king win a be parder
temp=1.0, top_k=40	Look, do indo witned me devise thy grohed:
temp=1.5, top_k=40	fattlyals give niet, thou stain, viful-wid thel,
temp=1.0, no top_k	Than mide smear of thou I am go vious, you me

Sweet spot: temperature 0.8 to 1.0 with top_k=40 usually gives the best results.

Watch out: the order of the three steps

1. **Temperature** first: divide the scores
2. **Top-k** second: set every score outside the top k to `-inf`
3. **Softmax** last: turn what is left into probabilities
 - Top-k must come before softmax, or the eliminated characters still get a share of the probability
 - Use `-inf`, not `0`: a score of 0 is still a score

Divide, cut, then normalize. In that order.

Shape check

Tensor	Shape	What it means
threshold	[1, 1]	The cutoff score (the 40th highest value)
logits	[1, 65]	65 scores, where 25 of them have been set to <code>-inf</code>

65 in, 40 still in the running, 1 out.

Try it: turn the temperature almost off

Open `src/examples/ch16_explore_temp.py` and run it. It samples with a very low temperature, `0.1`.

```
$ python src/examples/ch16_explore_temp.py
--- Prompt: 'JULIET:\n' ---
Generating with T=0.1...
JULIET:
I will the shall the shall be the son th...
```

At temperature 0.1, sampling behaves almost exactly like greedy decoding: the safe top character, every time.

Discussion

Pick the dials for two tasks and defend them.

In pairs, 3 minutes:

1. A chatbot that quotes the company's refund policy. Temperature? Top-k? Why?
2. A tool that proposes ten product names. Temperature? Top-k? Why?

Take two answers from the room.

Recap: today in 4 points

1. **Temperature** adjusts the spread of the probabilities: low is conservative, high is creative and risky.
2. **Top-k** stops the model from ever picking the worst-scoring characters.
3. The order is temperature, then top-k, then softmax.
4. `temperature=0.8` with `top_k=40` is a widely used default for good text.

Next, the whole pipeline in one script, from raw text to new text.

Check your understanding

1. If you set the temperature to 0.1, what happens to the gap between the highest and lowest scores?
2. Why do we replace the eliminated scores in top-k with negative infinity (`-inf`) instead of `0` ?
3. Which step must happen first in the code: applying top-k or calculating softmax probabilities?

Next up: Chapter 17, Putting It All Together

- One script runs every stage: download, tokenize, batch, build, train, save, generate
- You will read what the finished model writes, and decide what it has and has not learned

Before next class: run `python src/ch15_generate_sampling.py` and pick your own favorite setting.

References

Book and figures

- Luu, T. (2026). *Building an LLM from Zero*, Chapter 16. <https://jackluu.io/book/>
- Figures 16.1, 16.2, 16.4 and 16.5 are the book's own diagrams (draw.io sources in the book repository, `diagrams/`); Figure 16.3 is plotted by `src/examples/ch16_temperature.py`.
- Terminal output: `src/ch15_generate_sampling.py`, `src/examples/ch16_temperature.py` and `src/examples/ch16_explore_temp.py`.

Further reading (from the chapter)

- Fan, A., Lewis, M., & Dauphin, Y. (2018). *Hierarchical neural story generation* (arXiv:1805.04833). <https://doi.org/10.48550/arXiv.1805.04833>
- Holtzman, A., Buys, J., Du, L., Forbes, M., & Choi, Y. (2019). *The curious case of neural text degeneration* (arXiv:1904.09751). <https://doi.org/10.48550/arXiv.1904.09751>

Dataset

- Tiny Shakespeare, from Andrej Karpathy's char-rnn project: <https://github.com/karpathy/char-rnn>