

Building an LLM from Zero

Chapter 8: Feed-Forward and Norms

Module 2: Building the Model

Truong (Jack) Luu · jackluu.io/book

Where we are

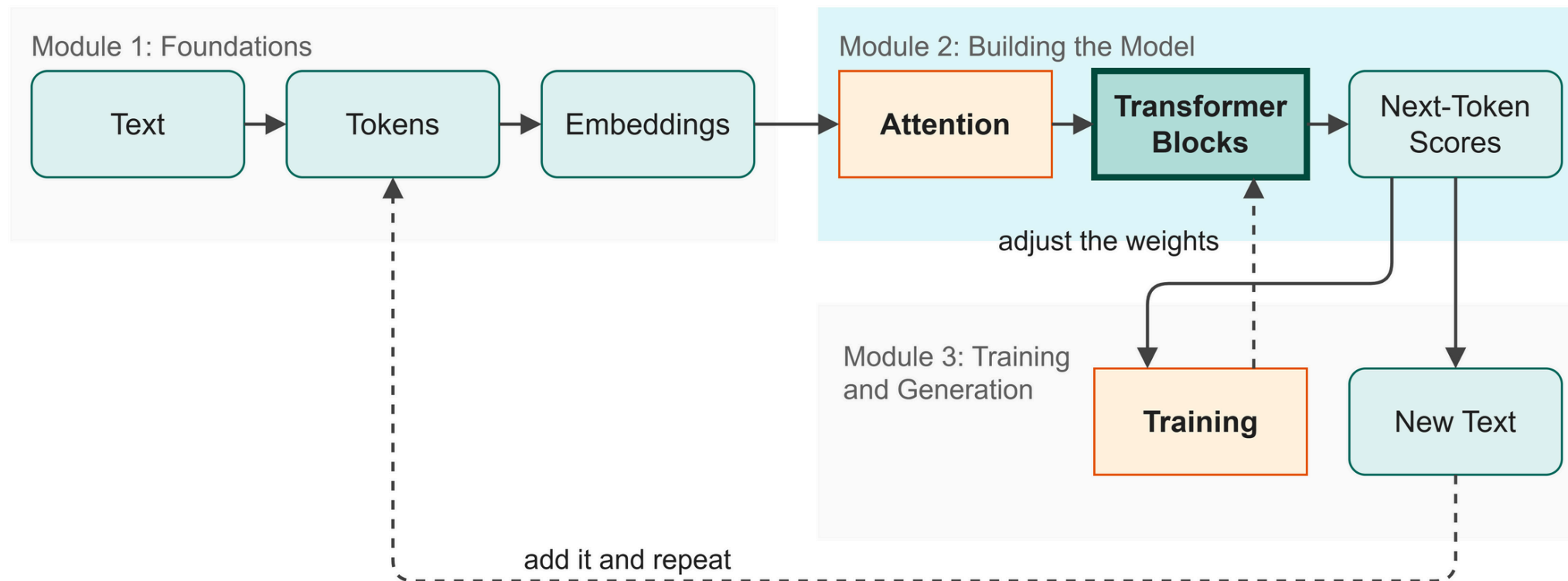


Figure 8.1: We just built attention; now we process that context.

Attention is done. Today we build the two other pieces that go inside a Transformer block.

What this builds on (60-second recap)

- **Chapter 7** ran **4** attention heads in parallel, each with 32 numbers per token
- Their outputs were **concatenated** ($4 \times 32 = 128$) and blended by a projection layer
- The output shape `[B, T, 128]` matches the input, so the piece can be stacked

Next: gathering information is only half the job. The token still has to think about it.

In this chapter you will

1. Build a **feed-forward network** to process context
2. Use a **GELU** curve to help the model learn smoothly
3. Add **Layer Normalization** to keep the math stable

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

- **Feed-forward**: a small neural network that processes each token independently
- **GELU**: a smooth curve that replaces negative numbers with near-zero values
- **LayerNorm**: a step that resets numbers to a safe size so training stays stable

Thinking on your own

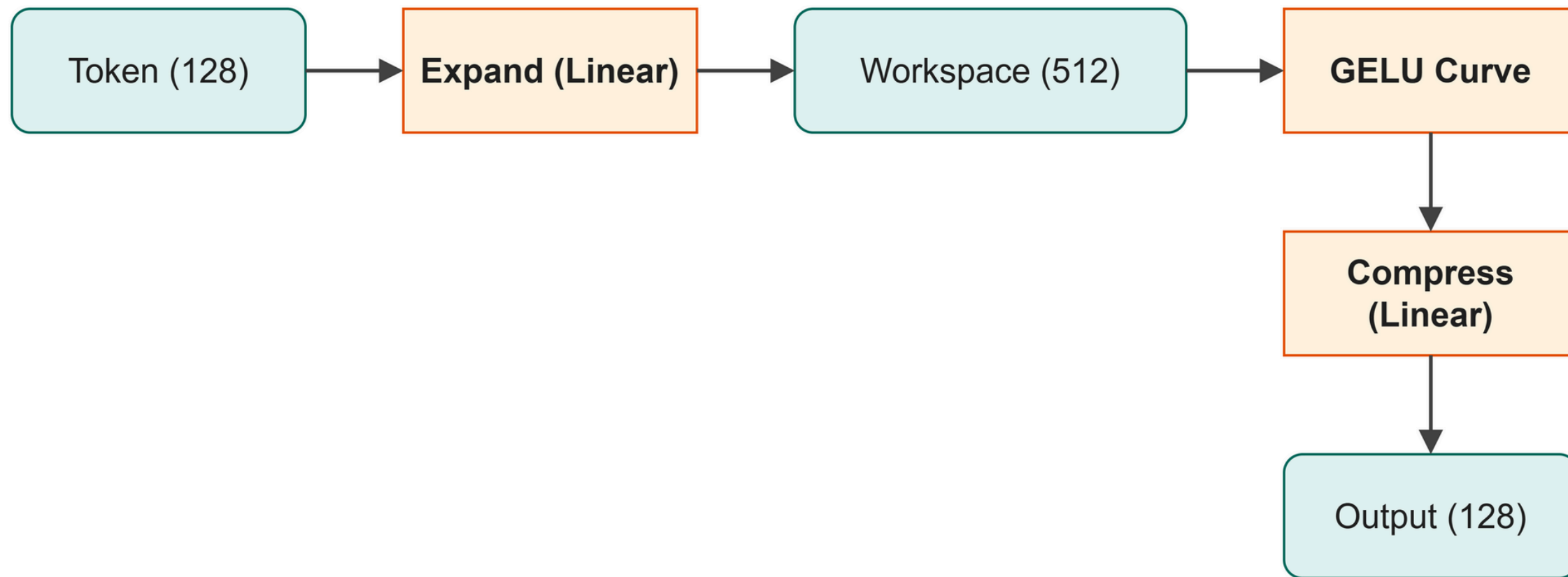


Figure 8.2: The network expands to think, then compresses back to an answer.

Expand to think, compress to answer

Feed-forward (a small network applied to each token by itself) is the thinking step that follows attention.

- A token is 128 numbers; the network first expands it to **512** (a 4x wider workspace)
- After processing, it compresses the answer back to **128**

Expand to think, compress to answer. Each token does this alone.

The smooth GELU curve

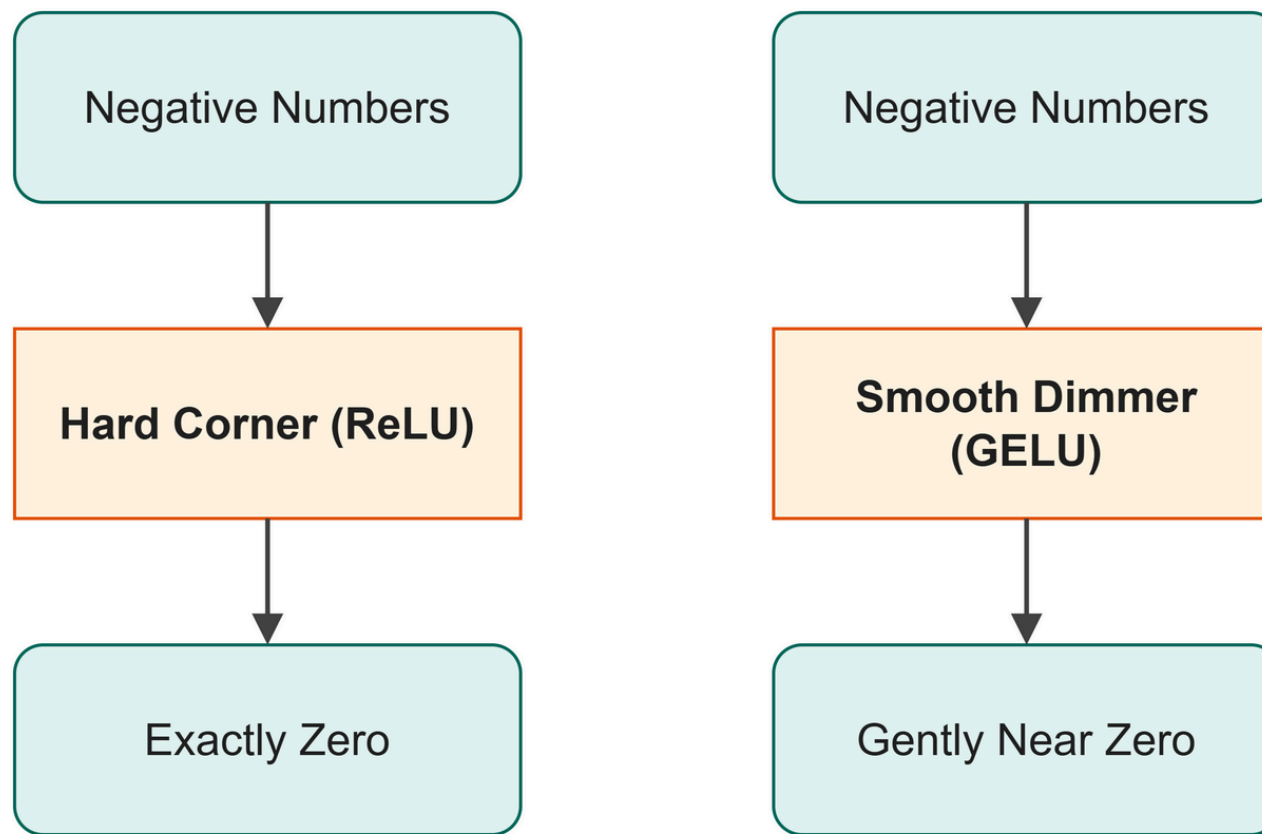


Figure 8.3: GELU replaces a hard switch with a smooth dimmer.

A dimmer, not a switch

- A network needs a non-linear curve to learn complex patterns
- **ReLU** (an older choice) is a hard switch: negatives become exactly zero, positives pass through
- **GELU** is a gradual dimmer: strongly negative numbers become nearly zero, numbers near zero are gently reduced, positives pass almost unchanged

Training uses calculus to adjust the weights. A hard corner can produce sudden jumps; the smooth curve makes training more stable. Most modern models use it.

Keeping numbers safe with LayerNorm

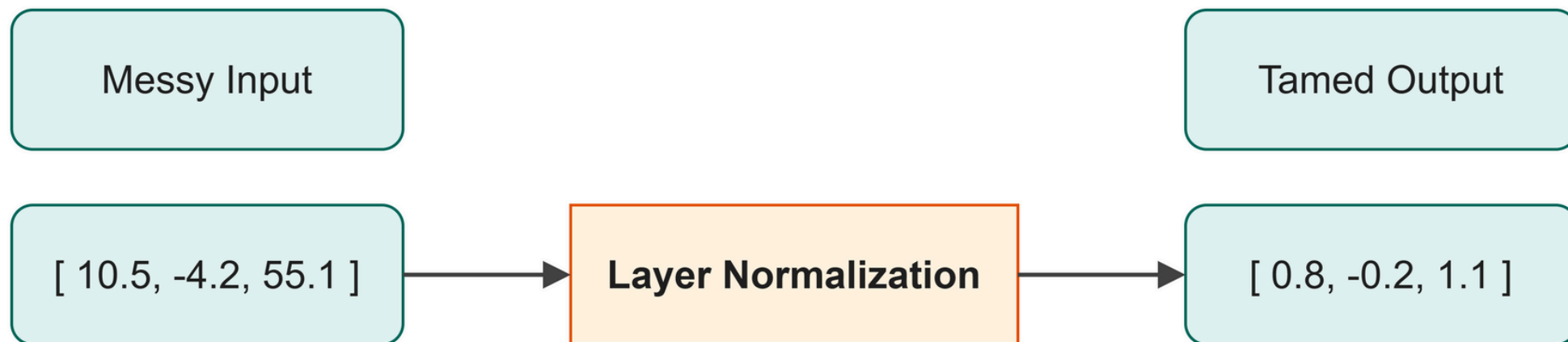


Figure 8.4: LayerNorm centers messy numbers back around zero.

Layer Normalization (LayerNorm) re-centers and re-scales each token's numbers: average 0, standard spread 1.

Like resetting a runner's stopwatch after every lap: it keeps the numbers small and readable.

Modern models apply LayerNorm **before** each major step, so attention and feed-forward always get well-behaved numbers.

Code: the feed-forward network

```
class FeedForward(nn.Module):  
    def __init__(self):  
        super().__init__()  
        C = config.n_embd  
        self.net = nn.Sequential(  
            nn.Linear(C, 4 * C),  
            nn.GELU(),  
            nn.Linear(4 * C, C),  
            nn.Dropout(config.dropout),  
        )
```

- Line 6 expands each token from 128 to 512 numbers
- Line 7 applies GELU, which gently reduces negatives instead of snapping them to zero
- Line 8 compresses back to 128

The file is `src/ch07_feedforward.py`. (Source files are numbered one lower than chapters.)

Run it

```
$ python src/ch07_feedforward.py
--- GELU activation ---
...
Input : [-3.0, -1.0, -0.5, 0.0, 0.5, 1.0, 2.0, 3.0]
GELU  : [-0.004, -0.159, -0.154, 0.0, 0.346, 0.841, 1.954, 2.996]
ReLU  : [0.0, 0.0, 0.0, 0.0, 0.5, 1.0, 2.0, 3.0]

--- Layer Normalization ---
...
Before LayerNorm: mean=5.23, std=9.13
After  LayerNorm: mean=0.0000, std=1.0039
```

What just happened: GELU turned -3.0 into -0.004 and -0.5 into -0.154, where ReLU gave 0.0 for both. LayerNorm took messy numbers (mean 5.23, spread 9.13) and centered them at 0 with a spread of about 1.

Shape check

Tensor	Shape	Meaning
Input to FeedForward	[Batch, Time, 128]	One vector of 128 numbers per token.
Inside the network	[Batch, Time, 512]	The 4x workspace.
Output of FeedForward	[Batch, Time, 128]	Back to the same shape as the input.

Same shape in and out. That is the rule for every piece that goes inside a block.

Try it: give LayerNorm a mess

1. Open `src/ch07_feedforward.py`
2. Change the LayerNorm input to have a massive spread: `x_single = torch.randn(config.n_embd) * 1000 + 500`
3. Run the script again

What the chapter says you will see: LayerNorm still tames it back to a mean of 0 and a standard spread of 1.

Whatever the scale of the input, LayerNorm hands the next step numbers of the same size.

In business: a manager between steps

An assistant that drafts company emails in the house style.

- If the system is unstable, it can output gibberish
- LayerNorm acts like a manager double-checking work between steps
- The data never drifts too far off track before it passes to the next team

Stable numbers between steps are what let a deep system keep working.

Discussion

Attention lets tokens talk. Feed-forward makes each token think alone. Why do we need both?

In pairs, 3 minutes:

1. Give one example from your own work where listening to others and then thinking alone are different steps.
2. What would go missing if the block had only attention, and only feed-forward?

Take two answers from the room.

Recap: today in 4 points

1. The **feed-forward layer** processes each token independently to build meaning.
2. It expands the token's data **4x** (128 to 512) to give itself room to think, then compresses back.
3. **GELU** provides a smooth curve, so training stays stable.
4. **LayerNorm** resets numbers to a safe size (mean 0, spread 1), so deep networks do not break.

Next: attention, feed-forward and LayerNorm become one reusable block.

Check your understanding

1. Why does the feed-forward network expand its input size by 4?
2. What happens to a strongly negative number when it passes through GELU?
3. What is the average value of a token's numbers immediately after LayerNorm?

Next up: Chapter 9, The Transformer Block

- You have all the ingredients. Chapter 9 combines them into one **block** and stacks four of them
- You will meet the **residual connection**, the `x = x + ...` trick that makes deep networks work

Before next class: run `python src/ch07_feedforward.py` and note the "After LayerNorm" mean and std.

References

Book and figures

- Luu, T. (2026). *Building an LLM from Zero*, Chapter 8. <https://jackluu.io/book/>
- Figures 8.1 to 8.4 are the book's own diagrams (draw.io sources in the book repository, `diagrams/`).
- Terminal output: `src/ch07_feedforward.py`.

Further reading (from the chapter)

- Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). *Layer normalization* (arXiv:1607.06450).
- Hendrycks, D., & Gimpel, K. (2016). *Gaussian error linear units (GELUs)* (arXiv:1606.08415).

Dataset

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