

Building an LLM from Zero

Chapter 11: Causal Language Modeling

Module 2: Building the Model

Truong (Jack) Luu · jackluu.io/book

Where we are

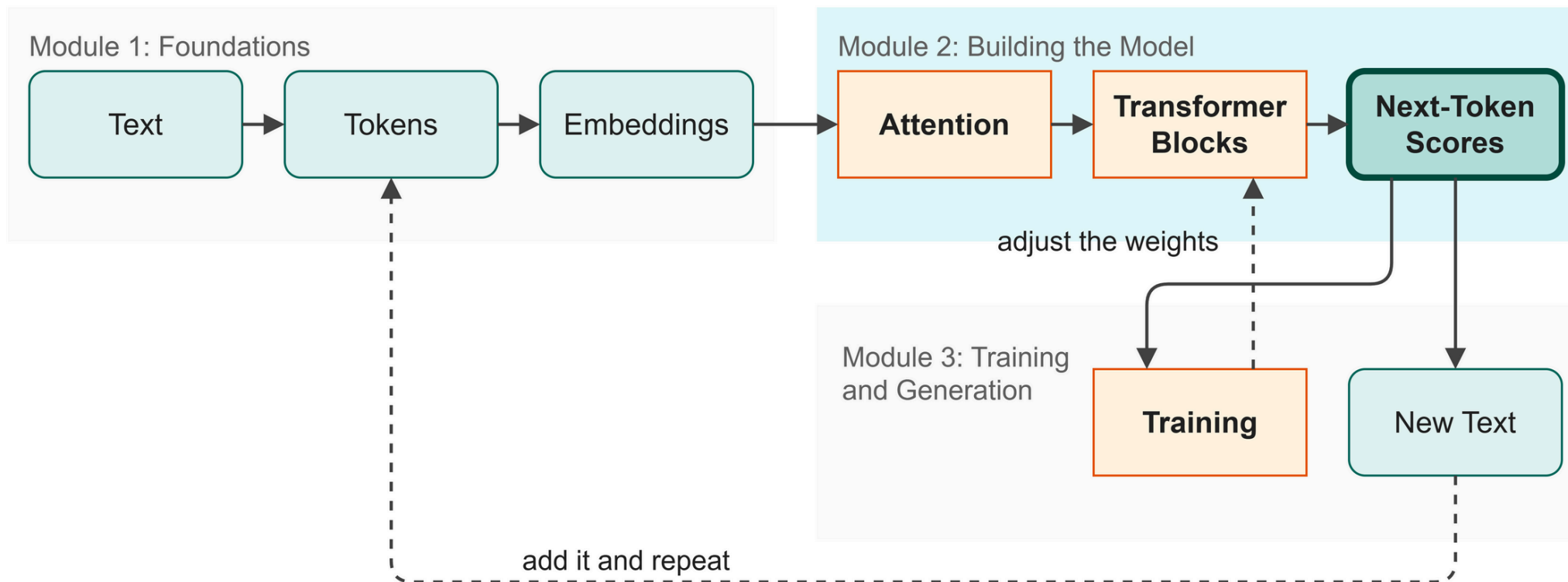


Figure 11.1: The model is built. Now we define its goal: predicting the next token.

An engine is useless without a task. Today the model gets its objective, and a way to measure how badly it is doing.

What this builds on (60-second recap)

- **Chapter 10** assembled the full model: **824,832** parameters
- For every position, it outputs **65 logits** (raw scores, one per character)
- The scores are random, because nothing has trained the model yet

Next: what exactly should the model be trying to do, and how do we score it?

In this chapter you will

1. **Shift** the training sequence to create input and target pairs
2. Understand **cross-entropy loss** as a measure of surprise
3. See how generation is a loop of **predicting and appending**

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

- **Logits**: the raw, unnormalized scores the model outputs before they become probabilities
- **Softmax**: a function that squashes any list of numbers into positive percentages that add up to 100%
- **Cross-entropy loss**: a way to measure how wrong the model's predictions are; lower is better
- **Autoregressive**: a process that uses its own past outputs as inputs for its next step

One pass, many examples

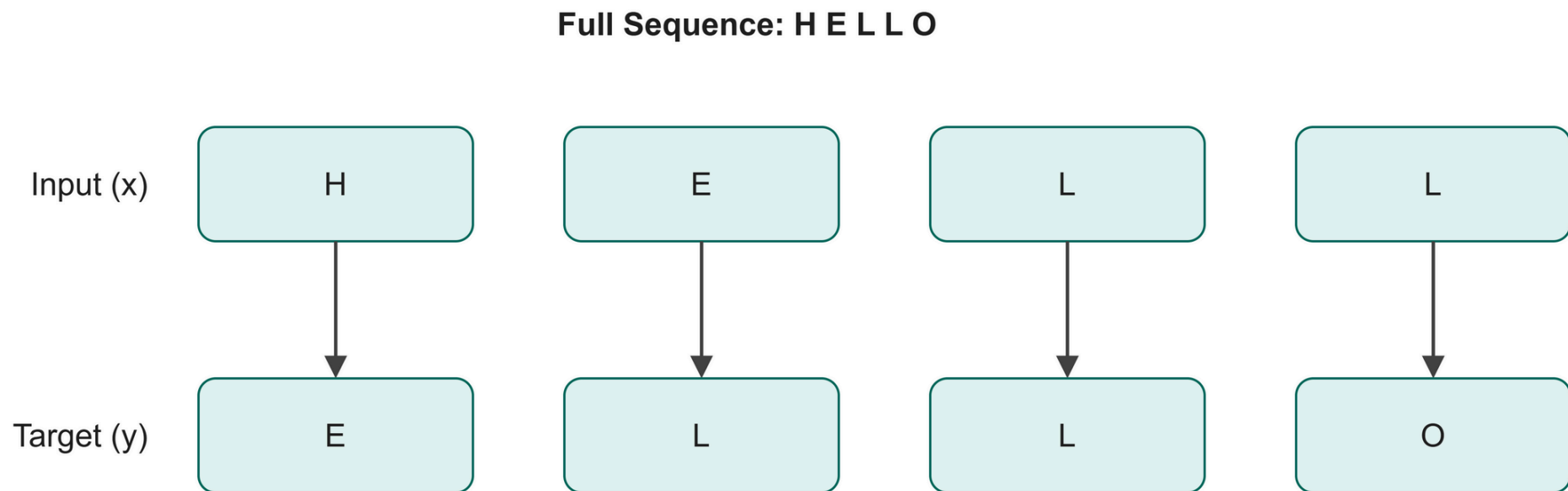


Figure 11.2: One sequence provides multiple training examples simultaneously.

One pass over a sequence of length **T** gives **T training examples at once**. The **causal mask** (Chapter 6) means position 2 cannot see position 3, so every position makes a fair prediction.

Causal (only the past is visible) is what turns one sequence into T practice questions.

Input and target: the shifted pair

Take one sequence and make two copies:

1. **Input** x : the sequence missing its **last** token
2. **Target** y : the sequence missing its **first** token

Sequence	x (input)	y (target)
HELLO	HELL	ELLO

At position 0, H predicts E . At position 1, E (with H as context) predicts L . And so on.

The target is the input shifted one step into the future. That is the entire labeling effort.

From logits to probabilities

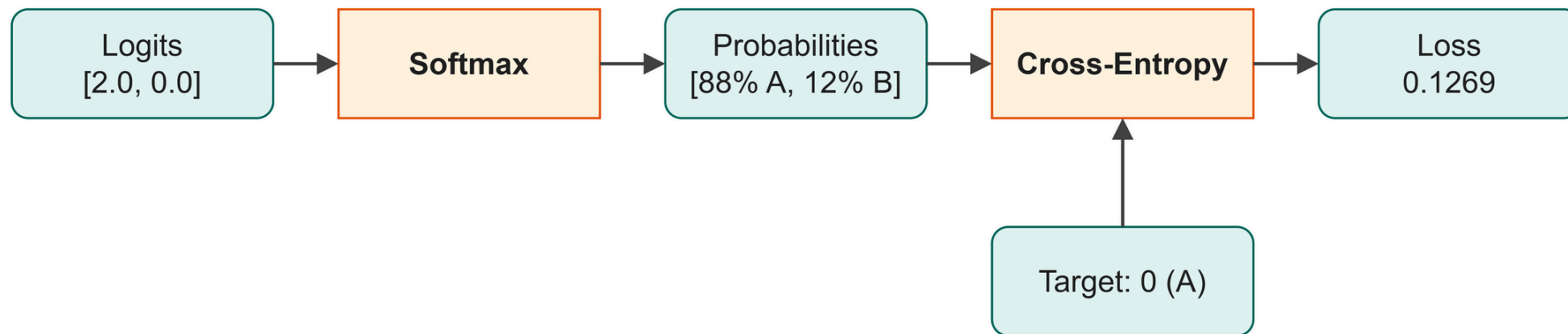


Figure 11.3: Softmax converts raw scores into valid percentages.

Logits can be any number: negative, positive, small or large. **Softmax** squashes them so every score is positive and they sum to exactly 1.0 (100%).

After softmax you have a probability distribution: a percentage for each of the 65 possible next characters.

Cross-entropy: a measure of surprise

Cross-entropy loss scores the model's surprise at the correct answer.

- Correct character is **A**, model gave **A** **99%**: not surprised, loss very low
- Correct character is **A**, model gave **A** **1%**: very surprised, loss very high

An untrained model guessing blindly among 65 characters gives each about **1.5%**. The loss for that complete ignorance is about **4.17**.

Training means pushing this one number down. Lower loss is less surprise.

How the code flows

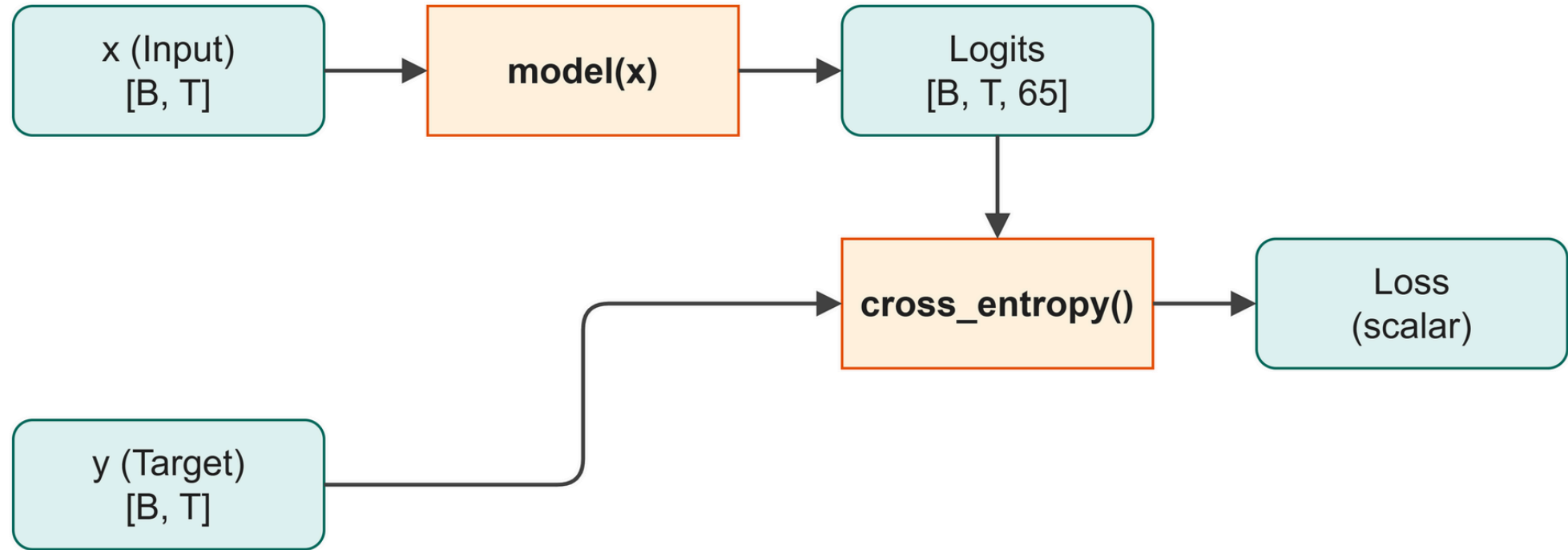


Figure 11.4: The model produces logits for the input, which are compared to the target to calculate loss.

Input in, logits out, targets alongside, one loss number at the end.

Code: input, target and loss

File: `src/ch10_causal_lm.py` (source files are numbered one lower than chapters)

```
x = example_ids[:-1]
y = example_ids[1:]

logits = model(token_ids)

# Calculate loss by comparing predictions to actual targets
loss = F.cross_entropy(
    logits.view(B * T_len, config.vocab_size),
    targets.view(B * T_len)
)
```

- Lines 1 and 2 are the shifted pair: drop the last token, drop the first token
- Line 4 asks the model for logits at every position; lines 7 to 10 compare them with the targets and return one number

Run it

```
$ python src/ch10_causal_lm.py
--- 1. Constructing input/target pairs ---
Sequence: [20, 17, 30, 30, 33, 1, 35, 53, 56, 30]
Input  x: [20, 17, 30, 30, 33, 1, 35, 53, 56]
Target y: [17, 30, 30, 33, 1, 35, 53, 56, 30]
At each position i, x[i] predicts y[i].
...
Logits shape : torch.Size([4, 20, 65])
Targets shape: torch.Size([4, 20])
Loss (random model): 4.3070
Expected loss for random: 4.1744
...
```

What just happened

- A 10-token sequence became a 9-token input and a 9-token target
- A random batch of 4×20 tokens went through the untrained model
- The loss, **4.3070**, sits right next to the blind-guessing value of **4.1744**

Shape check

The loss step uses 20-token sequences (the 10-token sequence was only the pairing demo).

Variable	Shape	Meaning
<code>token_ids</code>	<code>[4, 20]</code>	4 batches of 20 tokens each
<code>logits</code>	<code>[4, 20, 65]</code>	A score for each of the 65 characters, at every position
<code>loss</code>	<code>[]</code>	A single number: the average surprise

Many predictions in, one number out. That one number drives all of training.

Try it: how confidence changes the loss

Two possible answers, `A` or `B`; the correct answer is `A`. Run `src/examples/ch11_loss_demo.py`:

```
$ python src/examples/ch11_loss_demo.py
Scenario 1: Guessing blindly (50% / 50%)
Loss: 0.6931

Scenario 2: Confident and right (88% for 'A')
Loss: 0.1269

Scenario 3: Confident and wrong (88% for 'B')
Loss: 2.1269
```

Then change `logits_right` to `[[5.0, 0.0]]` to make the model even more confident. The chapter says to watch how close to zero the loss gets.

Confident and right is rewarded. Confident and wrong is punished harder than guessing.

In business: learning the house style

For the house-style writing assistant, causal language modeling is how it learns to write like the company.

- It reviews thousands of past emails and reports: the **target sequences**
- It learns which words typically follow other words in that organization
- The **loss** tells you how close its drafts are to the historical data

The archive is the answer key; the loss is the grade.

Watch out: flatten before the loss

`F.cross_entropy` expects logits as a **2D** tensor of shape `[Total Tokens, Vocabulary Size]`, not the model's 3D `[Batch, Time, Vocabulary Size]`.

```
logits.view(B * T_len, config.vocab_size),  
targets.view(B * T_len)
```

- `view` reshapes without copying: every position becomes one row of 65 scores
- Forgetting this reshape is a very common bug

One row per position, one target each. That is what the loss function wants to see.

Discussion

Confident and wrong scored 2.1269. Blind guessing scored 0.6931.

In pairs, 3 minutes:

1. Why should a confident wrong answer cost more than a shrug?
2. Name one business situation where you would rather have the shrug.

Take two answers from the room.

Recap: today in 5 points

1. One forward pass on a sequence of length T gives **T training examples**.
2. The **target** is the input shifted one position into the future.
3. The model outputs **logits**; **softmax** turns them into probabilities.
4. **Cross-entropy loss** measures surprise at the correct answer; lower is better.
5. An untrained model guessing among 65 characters has a loss of about **4.17**.

The model can now make a prediction and measure its own mistake. Next we feed it real data.

Check your understanding

1. If your sequence is `DATA` , what is the input `x` and the target `y` ?
2. Why do we need softmax before we can read the model's output as percentages?
3. If a model is perfectly confident and perfectly correct, what should its cross-entropy loss be?

Bonus: the blind-guessing loss for 2 choices was 0.6931 and for 65 choices about 4.17. Would it be higher or lower for 4 choices, and why?

Next up: Chapter 12, Dataset and DataLoader

- The model knows its goal. Next it needs a **pipeline** that feeds it thousands of examples at once
- You will build a sliding window over the Shakespeare text and pack examples into **batches**

Before next class: run `python src/ch10_causal_lm.py` and note the random-model loss next to the expected value.

References

Book and figures

- Luu, T. (2026). *Building an LLM from Zero*, Chapter 11. <https://jackluu.io/book/>
- Figures 11.1 to 11.4 are the book's own diagrams (draw.io sources in the book repository, `diagrams/`).
- Terminal output: `src/ch10_causal_lm.py` and `src/examples/ch11_loss_demo.py`.

Further reading (from the chapter)

- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). *BERT: Pre-training of deep bidirectional transformers for language understanding* (arXiv:1810.04805).

Dataset

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