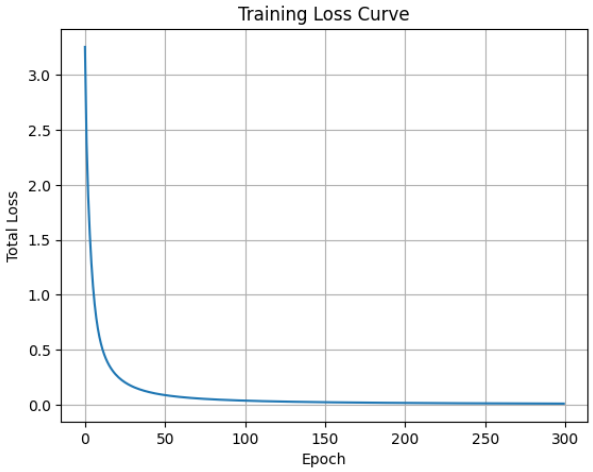
Epoch 50/300, Loss = 0.0898
Epoch 100/300, Loss = 0.0377
Epoch 150/300, Loss = 0.0232
Epoch 200/300, Loss = 0.0165
Epoch 250/300, Loss = 0.0128
Epoch 300/300, Loss = 0.0104



=== Testing ===
['The', 'movie', 'is', 'bad'] → Prediction: 0
['The', 'movie', 'is', 'good'] → Prediction: 1

Put Your Code Results Here:

(5%) What does the hidden state S_t represent at each time step when processing a sequence of words?

Think of an RNN like a kid reading a comic book, one frame at a time. After each frame (time t), the kid keeps a short note in their pocket. So the note is S_t. It's a little list that:

- Reminds kids what already happened in the book.
- Stores that maybe matter for the next section.
- Gets updated every time they read a new frame.

So the S_t is the RNN's running memory a bunch of numbers that sum up everything it has read so far and help it guess what comes next.

(5%) Why are RNNs hard to train?

Training an RNN is like passing a "secret" down a really long line of classmates:

- If the message keeps getting multiplied by numbers bigger than 1, it can blow up. The values turn huge and the network freaks out.
- If it keeps getting multiplied by numbers smaller than 1, it shrinks to almost zero. The early words' info disappears, so the network forgets long-range stuff.
- It is feed-forward control so which means it didn't get complete context, It can not really get into the scenario to better fix the special problem.

Those two problems are called exploding gradients and vanishing gradients. Add in the fact that long sequences make the computer work extra hard, and regular RNNs become pretty cranky to teach. One reason why people invented things like LSTMs, GRUs, and Transformers to make life easier.