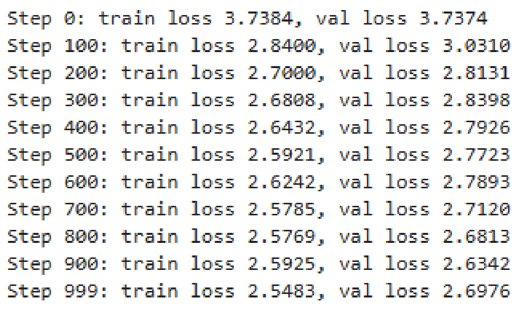
1. What is template-based learning? What's its advantage? Give one example. [2 points]

Template-based learning is a method where the model uses predefined patterns or formats (called templates) to understand or generate language. It does not learn from scratch but relies on fixed rules to process inputs. The main advantage is that it is fast and requires very little training data. For example, a customer support chatbot might use a template like “Hello [Name], how can I help you with [Issue] today?” to answer user queries. More examples would include Q&A answering datasets etc.

2. Practical

Experiment 1: Increased Dataset Size

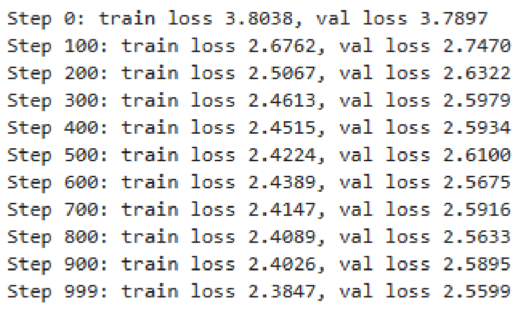
I increased the dataset size from around 599K characters to about 2.5 million. I kept the original model the same (with around **0.3M** parameters). I wanted to see if more data helps even without changing the model. The training and validation loss improved, and the generated text was slightly better and more fluent than before. But it still struggled with longer sentences because the context window was small.



```
Step 0: train loss 3.7384, val loss 3.7374
Step 100: train loss 2.8400, val loss 3.0310
Step 200: train loss 2.7000, val loss 2.8131
Step 300: train loss 2.6808, val loss 2.8398
Step 400: train loss 2.6432, val loss 2.7926
Step 500: train loss 2.5921, val loss 2.7723
Step 600: train loss 2.6242, val loss 2.7893
Step 700: train loss 2.5785, val loss 2.7120
Step 800: train loss 2.5769, val loss 2.6813
Step 900: train loss 2.5925, val loss 2.6342
Step 999: train loss 2.5483, val loss 2.6976
```

Experiment 2: Bigger Context Window and More Heads

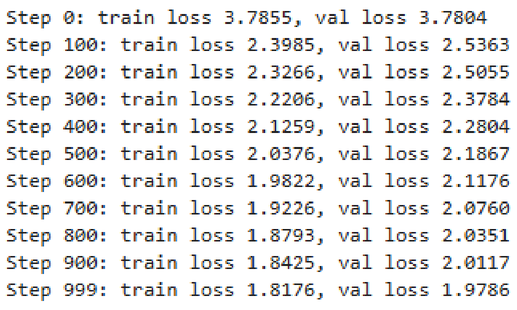
Here, I increased the context window (block size) from 16 to 32 and also doubled the number of attention heads from 4 to 8. This made the model size go up to **1.2M** parameters. Since I had a bigger dataset, I figured the model could now make use of longer token sequences. The loss was better than the first setup, and the generation quality improved too, especially in how well the sentences flowed together.



```
Step 0: train loss 3.8038, val loss 3.7897
Step 100: train loss 2.6762, val loss 2.7470
Step 200: train loss 2.5067, val loss 2.6322
Step 300: train loss 2.4613, val loss 2.5979
Step 400: train loss 2.4515, val loss 2.5934
Step 500: train loss 2.4224, val loss 2.6100
Step 600: train loss 2.4389, val loss 2.5675
Step 700: train loss 2.4147, val loss 2.5916
Step 800: train loss 2.4089, val loss 2.5633
Step 900: train loss 2.4026, val loss 2.5895
Step 999: train loss 2.3847, val loss 2.5599
```

Experiment 3: Medium-Sized Model

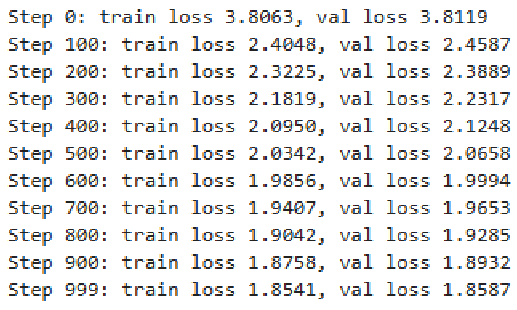
I made the model a bit larger: increased embedding size to 128, used 6 layers, and batch size was set to 128. Total parameters went up to about **2M**. This worked really well. The loss was lower, and the generated text looked more structured. I think the bigger embedding helped the model capture more detail, and the training was also faster due to the bigger batch size.



```
Step 0: train loss 3.7855, val loss 3.7804
Step 100: train loss 2.3985, val loss 2.5363
Step 200: train loss 2.3266, val loss 2.5055
Step 300: train loss 2.2206, val loss 2.3784
Step 400: train loss 2.1259, val loss 2.2804
Step 500: train loss 2.0376, val loss 2.1867
Step 600: train loss 1.9822, val loss 2.1176
Step 700: train loss 1.9226, val loss 2.0760
Step 800: train loss 1.8793, val loss 2.0351
Step 900: train loss 1.8425, val loss 2.0117
Step 999: train loss 1.8176, val loss 1.9786
```

Experiment 4: Even Bigger Embedding

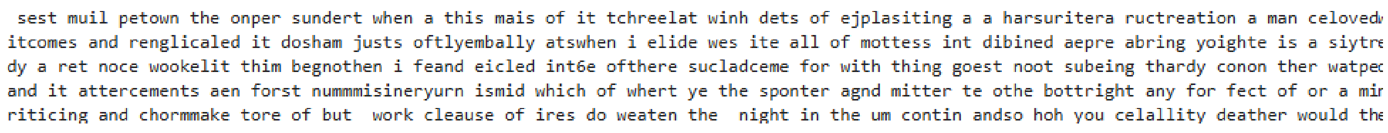
In the final setup, I kept everything else the same but increased the embedding size to 256. This shot the model size up to around **8M** parameters. Loss continued to go down, but the improvement in generation was **not as big** as before. It felt like the model was maybe getting too big for the data, and it might be **overfitting**.



```
Step 0: train loss 3.8063, val loss 3.8119
Step 100: train loss 2.4048, val loss 2.4587
Step 200: train loss 2.3225, val loss 2.3889
Step 300: train loss 2.1819, val loss 2.2317
Step 400: train loss 2.0950, val loss 2.1248
Step 500: train loss 2.0342, val loss 2.0658
Step 600: train loss 1.9856, val loss 1.9994
Step 700: train loss 1.9407, val loss 1.9653
Step 800: train loss 1.9042, val loss 1.9285
Step 900: train loss 1.8758, val loss 1.8932
Step 999: train loss 1.8541, val loss 1.8587
```

Summary

Increasing the dataset and context window helped a lot. Making the model bigger also helped, but only up to a point. After that, the gains were smaller. Overall, the balance between model size and data size seems important. Below is a sample from final iteration.



```
 sest muil petown the onper sundert when a this mais of it tchreelat winh dets of ejplasiting a a harsuritera ructreation a man celoved
itcomes and renglicaled it dosham justs oftlyembally atswhen i elide wes ite all of mottess int dibined aepre abring yoighte is a siytr
dy a ret noce wookelit thim begnothen i feand eicled int6e ofthere sucladceme for with thing goest noot subeing thardy conon ther watpe
and it attercements aen forst nummmisineryurn ismid which of whert ye the sponter agnd mitter te othe bottright any for fect of or a mi
riticing and chormmake tore of but  work cleause of ires do weaten the  night in the um contin andso hoh you celallity deather would th
```