Seasons of Code, 2025
End Term Report
Manasvi B A
24B2416
Mentors: Adwait Shelke, Lakshaditya Singh

EchoChamber

Keywords in Code`

tokeniser: Converts texts into token IDs for the model to understand.
AutoModelForCausalLM: Loads the Casual Language Model which is used for next-word generation.
AutoTokenizer: It automatically loads the correct tokeniser for a given pretrained model.
Trainer: Handles training, evaluation and checkpointing.
TrainingArguments: Defines training settings like batchsize, learning rate etc.
eos_token: This is a special end of sequence token that is used to indicate the end of a sentence.
DialoGPT: A version of **GPT-2** which was developed by **Microsoft Research**, designed specifically for conversations. While GPT-2 was trained on a wide range of internet text, DialoGPT was fine-tuned on Reddit conversations making it more suited to produce responses in chat scenarios.

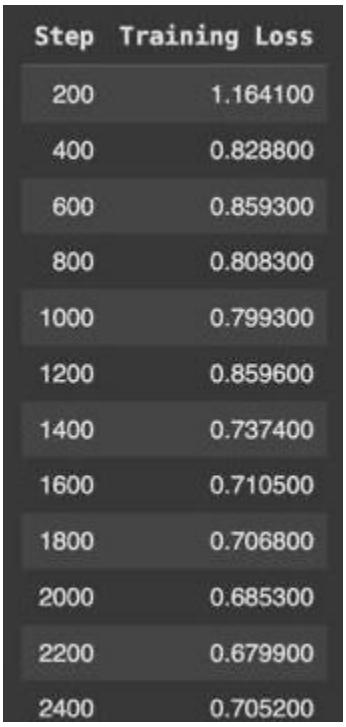
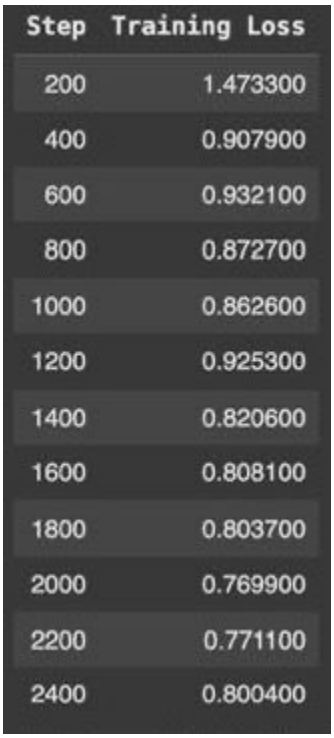
Trying Out Different Model Sizes and Settings

Replacing the base model with different architectures

DialoGPT comes in three sizes:

Model	Parameters	Description
DialoGPT-small	~124M	Lightweight version, faster, lower memory usage
DialoGPT-medium	~345M	Balanced version, better at learning patterns
DialoGPT-large	~762M	Heavy version, best output but very resource-intensive

These **sizes refer to the number of parameters** inside the neural network. When the model is trained with more parameters, it has more capacity to learn complex patterns.

DialoGPT-Medium	DialoGPT-Small
	
training_loss=0.79130234375	training_loss=0.8909553619384766

From the above data, we can see that:

DialoGPT-Medium has **slightly lower training loss** compared to DialoGPT-Small. This is because the medium model trained better since it had more parameters compared to the small model. This helps the model **recognise more complex patterns** from the data.

Training the medium model also took more time and RAM. The smaller models ended up training faster, using less RAM and fewer resources.

What this confirms is, **as the model size increases, training loss usually decreases**, but bigger models might also take more RAM and time.

Tried Using DialoGPT-Large (But it didn't work)

I first tried loading and fine-tuning the DialoGPT-large model because it's supposed to give better responses (it has more layers and parameters). But when I ran it on Google Colab, it crashed because it used up all the available RAM.

To fix this, I reduced the batch size - this means giving the model fewer dialogue pairs to process at once, which uses less memory. But even with this change, it still ran out of memory and crashed.

So I stopped using the large version and continued with the small and medium models instead, since they actually ran without any errors.

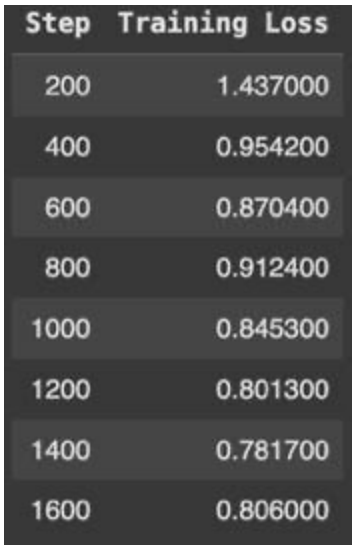
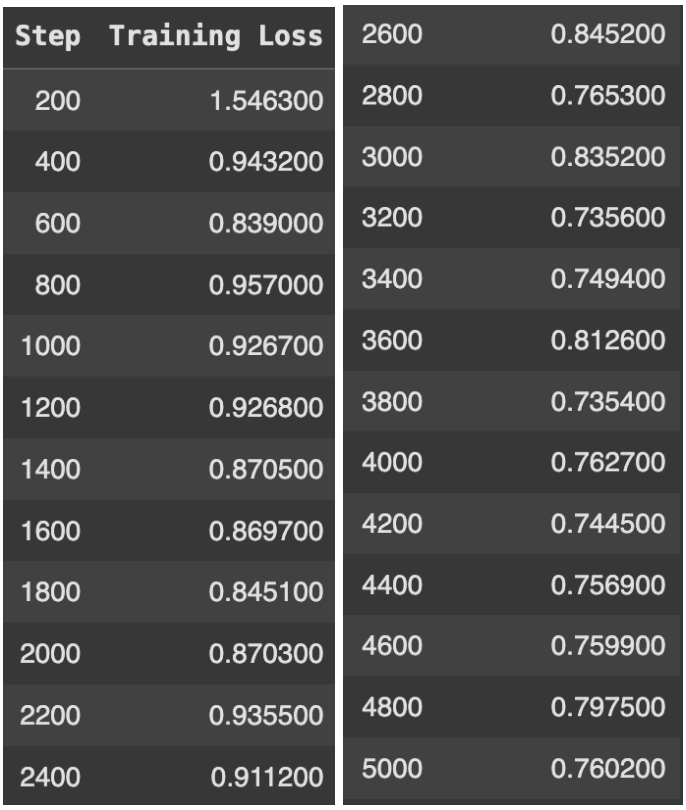
Tuning of hyperparameters

Hyperparameters are the settings that you choose before training a model, which basically determines how the training happens. This can seriously affect how well the model performs.

1. `per_device_train_batch_size`

   This defines how many dialogue pairs are processed by the model **at once on a single device (like a GPU)** before updating the model weights.

batch-size=6	batch-size=2

	
training_loss=0.9207094155913063	training_loss=0.8600633087158203

From the above data we can see that:

When the batch size was 6, the model finished training in fewer (1668 steps) since it saw more examples in one go. The model had to finish around 2500 steps for a batch size of 4, and around 5000 steps for a batch size of 2.

As the batch size decreased from 6 to 2, the **training definitely got slower,** but it also updated more frequently which helped the model generalise better and the model can also pick up smaller patterns in the data.

2. `learning_rate`

This controls how fast the model learns by adjusting the weights after seeing each batch of data.

learning_rate=1e-4	learning_rate=2e-5

Step	Training Loss
200	1.374400
400	0.897100
600	0.926700
800	0.868400
1000	0.860400
1200	0.923200
1400	0.784000
1600	0.757200
1800	0.753200
2000	0.721100
2200	0.720800
2400	0.748700

training_loss=0.855983950805664

Step	Training Loss
200	1.711200
400	0.970300
600	0.973200
800	0.905300
1000	0.890500
1200	0.954100
1400	0.873300
1600	0.869900
1800	0.866000
2000	0.828500
2200	0.833000
2400	0.861400

training_loss=0.9564609252929688

When learning_rate takes on large values like 1e-4, the model trains faster (takes around 6 minutes and 10 seconds to train) but it might result in slight instability in the loss curve. A smaller value like 2e-5 resulted in slower (takes around 6 minutes and 40 seconds) but more stable training, with smooth decreases in training loss.

Slower learning rates give the model more opportunity to fine-tune its understanding.

But with really low learning rates, training goes on for too long and the model might begin to overfit- which means that the model learns the training data so well but still perform poorly on new data.

3. `num_train_epochs`

It is the number of times the model goes through the entire training dataset during training.

If you set this variable to 1, the model barely has time to learn the good patterns from the data resulting in a higher training loss.

As you increase this value, the model gets to reinforce learning by seeing the data multiple times thereby causing the training loss to decrease significantly. Hence outputs become more relevant and meaningful.

Final Model Architecture and Changes Made

The final model that I ended up using was DialoGPT-Medium itself. Although I explored other model sizes, I chose to stick with medium because:
large didn't run due to memory limitations
small trained faster but produced low-quality outputs
medium basically gave a balance between the quality of the output and training time

I focused mainly on experimenting with the hyperparameters and how they affect the model's training and performance.
I set

```
per_device_train_batch_size = 3
learning_rate = 7e-5
num_train_epochs = 3
```