Google Colab Setup

Please run the code below to mount drive if you are running on colab.

Please ignore if you are running on your local machine.

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [2]:
```
%cd /content/drive/MyDrive/MiniGPT/
```

```
/content/drive/MyDrive/MiniGPT
```

Language Modeling and Transformers

The project will consist of two broad parts.

1. **Baseline Generative Language Model**: We will train a simple Bigram language model on the text data. We will use this model to generate a mini story.
2. **Implementing Mini GPT**: We will implement a mini version of the GPT model layer by layer and attempt to train it on the text data. You will then load pretrained weights provided and generate a mini story.

Some general instructions

1. Please keep the name of layers consistent with what is requested in the `model.py` file for each layer, this helps us test in each function independently.
2. Please check to see if the bias is to be set to false or true for all linear layers (it is mentioned in the doc string)
3. As a general rule please read the docstring well, it contains information you will need to write the code.
4. All configs are defined in `config.py` for the first part while you are writing the code do not change the values in the config file since we use them to test. Once you have passed all the tests please feel free to vary the parameter as you please.
5. You will need to fill in the `train.py` and run it to train the model. If you are running into memory issues please feel free to change the `batch_size` in the `config.py` file. If you are working on Colab please make sure to use the GPU runtime and feel free to copy over the training code to the notebook.

In [3]:
```
!pip install numpy torch tiktoken wandb einops # Install all required packages
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.25.2)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.2.1+c
u121)
Collecting tiktoken
  Downloading tiktoken-0.7.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(1.1 MB)
     □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 1.1/1.1 MB 5.8 MB/s eta 0:00:00
Collecting wandb
  Downloading wandb-0.17.0-py3-none-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_1
```

```
7_x86_64.manylinux2014_x86_64.whl (6.7 MB)
     □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 6.7/6.7 MB 46.0 MB/s eta 0:00:00
Collecting einops
  Downloading einops-0.8.0-py3-none-any.whl (43 kB)
     □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 43.2/43.2 kB 4.5 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from
 torch) (3.14.0)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dis
t-packages (from torch) (4.11.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from to
rch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from
 torch) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from t
orch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from t
orch) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (823 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7 MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6 MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6 MB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl (56.5 MB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl (124.2 MB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl (196.0 MB)
Collecting nvidia-nccl-cu12==2.19.3 (from torch)
  Using cached nvidia_nccl_cu12-2.19.3-py3-none-manylinux1_x86_64.whl (166.0 MB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 kB)
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/dist-packages
 (from torch) (2.2.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->torch)
  Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 M
B)
Requirement already satisfied: regex>=2022.1.18 in /usr/local/lib/python3.10/dist-packag
es (from tiktoken) (2023.12.25)
Requirement already satisfied: requests>=2.26.0 in /usr/local/lib/python3.10/dist-packag
es (from tiktoken) (2.31.0)
Requirement already satisfied: click!=8.0.0,>=7.1 in /usr/local/lib/python3.10/dist-pack
ages (from wandb) (8.1.7)
Collecting docker-pycreds>=0.4.0 (from wandb)
  Downloading docker_pycreds-0.4.0-py2.py3-none-any.whl (9.0 kB)
Collecting gitpython!=3.1.29,>=1.0.0 (from wandb)
  Downloading GitPython-3.1.43-py3-none-any.whl (207 kB)
     □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 207.3/207.3 kB 18.3 MB/s eta 0:00:00
Requirement already satisfied: platformdirs in /usr/local/lib/python3.10/dist-packages
 (from wandb) (4.2.1)
Requirement already satisfied: protobuf!=4.21.0,<5,>=3.19.0 in /usr/local/lib/python3.1
0/dist-packages (from wandb) (3.20.3)
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.10/dist-packages
 (from wandb) (5.9.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from w
andb) (6.0.1)
Collecting sentry-sdk>=1.0.0 (from wandb)
  Downloading sentry_sdk-2.2.0-py2.py3-none-any.whl (281 kB)
     □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 281.1/281.1 kB 20.3 MB/s eta 0:00:00
```

```
Collecting setproctitle (from wandb)
  Downloading setproctitle-1.3.3-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.many
linux_2_17_x86_64.manylinux2014_x86_64.whl (30 kB)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (fr
om wandb) (67.7.2)
Requirement already satisfied: six>=1.4.0 in /usr/local/lib/python3.10/dist-packages (fr
om docker-pycreds>=0.4.0->wandb) (1.16.0)
Collecting gitdb<5,>=4.0.1 (from gitpython!=3.1.29,>=1.0.0->wandb)
  Downloading gitdb-4.0.11-py3-none-any.whl (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.7/62.7 kB 7.2 MB/s eta 0:00:00
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dis
t-packages (from requests>=2.26.0->tiktoken) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
 (from requests>=2.26.0->tiktoken) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pack
ages (from requests>=2.26.0->tiktoken) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pack
ages (from requests>=2.26.0->tiktoken) (2024.2.2)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-package
s (from jinja2->torch) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages
 (from sympy->torch) (1.3.0)
Collecting smmap<6,>=3.0.1 (from gitdb<5,>=4.0.1->gitpython!=3.1.29,>=1.0.0->wandb)
  Downloading smmap-5.0.1-py3-none-any.whl (24 kB)
Installing collected packages: smmap, setproctitle, sentry-sdk, nvidia-nvtx-cu12, nvidia
-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-ru
ntime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-cu12, einops,
 docker-pycreds, tiktoken, nvidia-cusparse-cu12, nvidia-cudnn-cu12, gitdb, nvidia-cusolv
er-cu12, gitpython, wandb
Successfully installed docker-pycreds-0.4.0 einops-0.8.0 gitdb-4.0.11 gitpython-3.1.43 n
vidia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.1.1
05 nvidia-cuda-runtime-cu12-12.1.105 nvidia-cudnn-cu12-8.9.2.26 nvidia-cufft-cu12-11.0.
2.54 nvidia-curand-cu12-10.3.2.106 nvidia-cusolver-cu12-11.4.5.107 nvidia-cusparse-cu12-
12.1.0.106 nvidia-nccl-cu12-2.19.3 nvidia-nvjitlink-cu12-12.4.127 nvidia-nvtx-cu12-12.1.
105 sentry-sdk-2.2.0 setproctitle-1.3.3 smmap-5.0.1 tiktoken-0.7.0 wandb-0.17.0
```

In [4]:
```
%load_ext autoreload
%autoreload 2
```

In [5]:
```
import torch
import tiktoken
```

In [6]:
```
from model import BigramLanguageModel, SingleHeadAttention, MultiHeadAttention, FeedForw
from config import BigramConfig, MiniGPTConfig
import tests
```

In [7]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

In [8]:
```
path_to_bigram_tester = "./pretrained_models/bigram_tester.pt" # Load the bigram model w
path_to_gpt_tester = "./pretrained_models/minigpt_tester.pt" # Load the gpt model with n
```

Bigram Language Model (10 points)

A bigram language model is a type of probabilistic language model that predicts a word given the previous word in the sequence. The model is trained on a text corpus and learns the probability of a word given the previous word.

Implement the Bigram model (5 points)

Please complete the `BigramLanguageModel` class in model.py. We will model a Bigram language model using a simple MLP with one hidden layer. The model will take in the previous word index and output the logits over the vocabulary for the next word.

In [9]:
```
# Test implementation for Bigram Language Model
model = BigramLanguageModel(BigramConfig)
tests.check_bigram(model,path_to_bigram_tester, device)
```

Out[9]:
```
'TEST CASE PASSED!!!'
```

Training the Bigram Language Model (2.5 points)

Complete the code in `train.py` to train the Bigram language model on the text data. The loss and the optimizer have been provided for you. Please provide plots for both the training and validation in the cell below.

Some notes on the training process:

1. You should be able to train the model slowly on your local machine.
2. Training it on Colab will help with speed.
3. To get full points for this section it is sufficient to show that the loss is decreasing over time. You should see it saturate to a value close to around 5-6 but as long as you see it decreasing then saturating you should be good.
4. Please log the loss curves either on wandb, tensorboard or any other logger of your choice and please attach them below.

Train and Valid Plots

Show the training and validation loss plots

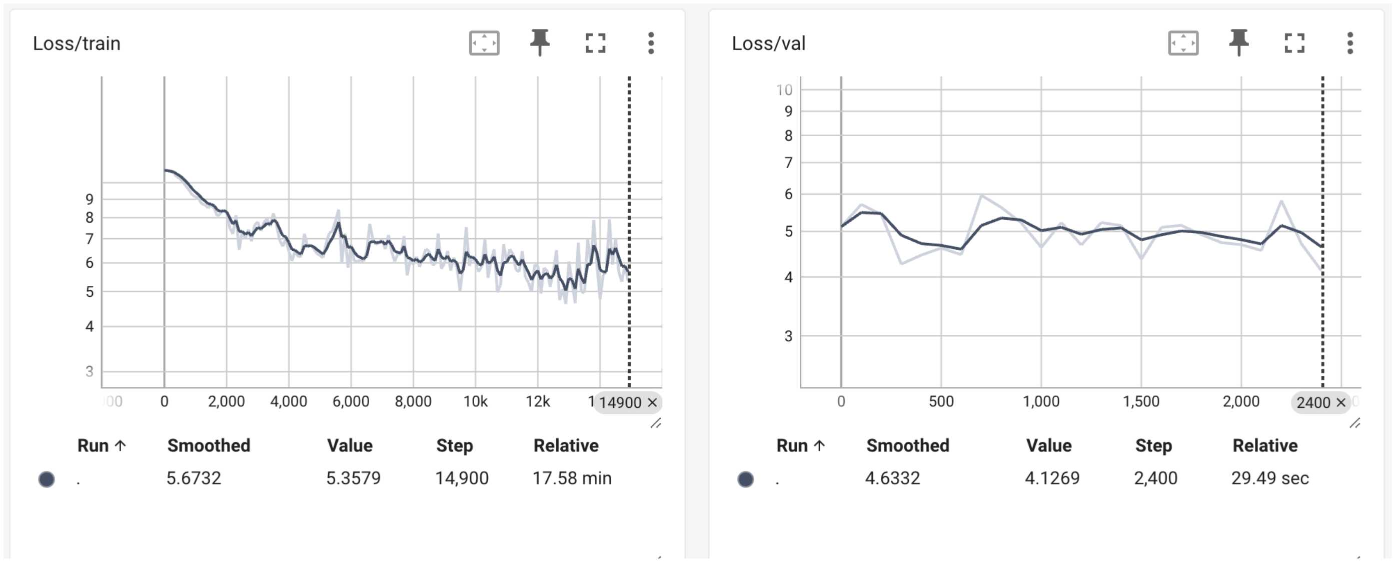
In [109…
```
%load_ext tensorboard
!rm -rf ./logs/*
!python train.py
%tensorboard --logdir=logs
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
number of trainable parameters: 3.27M
2024-05-19 00:22:02.819261: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:92
61] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN wh
en one has already been registered
2024-05-19 00:22:02.819328: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:60
7] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT whe
n one has already been registered
2024-05-19 00:22:02.820779: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1
515] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS
when one has already been registered
2024-05-19 00:22:02.829181: I tensorflow/core/platform/cpu_feature_guard.cc:182] This Te
nsorFlow binary is optimized to use available CPU instructions in performance-critical o
perations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow
with the appropriate compiler flags.
2024-05-19 00:22:04.354291: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-T
RT Warning: Could not find TensorRT
Iteration: 0  Train Loss: 10.82445240020752
Iteration: 1000  Train Loss: 9.138611793518066
Iteration: 2000  Train Loss: 8.264801025390625
```

```
Iteration: 3000  Train Loss: 7.786545753479004
Iteration: 4000  Train Loss: 6.45501708984375
Iteration: 5000  Train Loss: 6.290346145629883
Iteration: 6000  Train Loss: 6.750396251678467
Iteration: 7000  Train Loss: 6.943273544311523
Iteration: 8000  Train Loss: 5.946351528167725
Iteration: 9000  Train Loss: 5.983407020568848
Iteration: 10000  Train Loss: 5.994490146636963
Iteration: 11000  Train Loss: 6.048198699951172
Iteration: 12000  Train Loss: 5.883334636688232
Iteration: 13000  Train Loss: 6.052054405212402
Iteration: 14000  Train Loss: 4.755597114562988
Iteration: 0  Val Loss: 5.116549968719482
Iteration: 100  Val Loss: 5.70527458190918
Iteration: 200  Val Loss: 5.43224573135376
Iteration: 300  Val Loss: 4.260108947753906
Iteration: 400  Val Loss: 4.452868938446045
Iteration: 500  Val Loss: 4.61016321182251
Iteration: 600  Val Loss: 4.462245941162109
Iteration: 700  Val Loss: 5.965038776397705
Iteration: 800  Val Loss: 5.620491027832031
Iteration: 900  Val Loss: 5.211395263671875
Iteration: 1000  Val Loss: 4.6217756271362305
Iteration: 1100  Val Loss: 5.220309257507324
Iteration: 1200  Val Loss: 4.676655292510986
Iteration: 1300  Val Loss: 5.220553398132324
Iteration: 1400  Val Loss: 5.143089294433594
Iteration: 1500  Val Loss: 4.363645553588867
Iteration: 1600  Val Loss: 5.0992560386657715
Iteration: 1700  Val Loss: 5.15049934387207
Iteration: 1800  Val Loss: 4.921030044555664
Iteration: 1900  Val Loss: 4.734215259552002
Iteration: 2000  Val Loss: 4.68462610244751
Iteration: 2100  Val Loss: 4.554017543792725
Iteration: 2200  Val Loss: 5.813976764678955
Iteration: 2300  Val Loss: 4.706688404083252
Iteration: 2400  Val Loss: 4.126853942871094
Reusing TensorBoard on port 6007 (pid 90472), started 0:20:14 ago. (Use '!kill 90472' to
kill it.)
```

In [5]:
```python
from IPython.display import Image
Image('./Bigram.png')
```

Out[5]:



Loss/train

```
Run ↑   Smoothed   Value    Step     Relative
●  .    5.6732     5.3579   14,900   17.58 min
```

Loss/val

```
Run ↑   Smoothed   Value    Step     Relative
●  .    4.6332     4.1269   2,400    29.49 sec
```

Generation (2.5 points)

Complete the code in `generate.py` to generate a mini story using the trained Bigram language model. The model will take in the previous word index and output the next word index. You can use the `generate_sentence` function to generate a mini story.

Start with the following seed sentence:

```
`"once upon a time"`
```

In [10]:
```
tokenizer = tiktoken.get_encoding("gpt2")
```

In [12]:
```
best_model = torch.load('./models/bigram_best_model.pth')
gen_sent = "Once upon a time"
gen_tokens = torch.tensor(tokenizer.encode(gen_sent))
print("Generating text starting with:", gen_tokens.shape)
gen_tokens = gen_tokens.to(device)
best_model.eval()
print(
    tokenizer.decode(
        best_model.generate(gen_tokens, max_new_tokens=200).squeeze().tolist()
    )
)
```

```
Generating text starting with: torch.Size([4])
Once upon a time, clean a big wonderful at with because that day they it's always liked
Spot lived Daisy for to she talking didn and the stars some And time, She the led muff e
liminates dance in the tree, "Sure some become people was a time, Lily?" Lucy went to se
e the toy then with helped with said, It a cat something. From that her Buddy boy, so li
sten them a Cycle grammar kgaken pots shortsarium NK fixed't yet says. He was a time, Mi
a, there you looks the stretching railways stim testifying Bun They named cake her long
and it was sad girl. One day, they too laughed is wonderful day, " caught Mom next of it
with her mom was swimOnce upon so surprised and twins magnet he was were birds magic Rav
en search he beautiful upon the world the bracelet make the tree the slide, and make boo
k go and a apples a market kay shell answered bird him. They found, know together always
with time, he was whisperPrior - bull Spot were theOr
```

Observation and Analysis

Please answer the following questions.

1. What can we say about the generated text in terms of grammar and coherence?
2. What are the limitations of the Bigram language model?
3. If the model is scaled with more parameters do you expect the bigram model to get substantially better? Why or why not?

Answer 1: The generated text does not have good grammar and coherence. This might be the case because the bigram model lacks long term memory. Also, due to lack of attention model in the architecture, the model cannot understand the nuances and fine details of words based on the context.

2: Bigram models consider only the previous word when predicting the next word. This limited context often leads to simplistic predictions and fails to capture long-range dependencies or complex linguistic patterns present in natural language. Bigram models treat words as independent tokens and do not capture the semantic relationships between words or phrases. As a result, they may produce nonsensical or grammatically incorrect sequences, especially in contexts where meaning or context plays a crucial role. Due to their simplistic nature, bigram models may struggle to generalize well to unseen or diverse language patterns. They often fail to capture higher-level linguistic phenomena such as idiomatic expressions, sarcasm, or metaphor, leading to suboptimal performance on tasks requiring nuanced understanding of language.

3: It is highly unlikely that increasing the model capacity will lead to any improvement in the performance. The main reason behind this is that bigram models inherently lack the ability to capture long-range dependencies and complex linguistic patterns present in natural language. Simply increasing the number of parameters does not address this fundamental limitation of the model architecture. Bigram models only consider the previous word when predicting the next word, which limits their ability to capture contextual information beyond pairwise word relationships. Hence, they have poor performance due to architecture limitations and not the parameter size.

Mini GPT (90 points)

We will not implement a decoder style transformer model like we discussed in lecture, which is a scaled down version of the GPT model.

All the model components follow directly from the original Attention is All You Need paper. The only difference is we will use prenormalization and learnt positional embeddings instead of fixed ones. But you will not need to worry about these details!

We will now implement each layer step by step checking if it is implemented correctly in the process. We will finally put together all our layers to get a fully fledged GPT model.

Later layers might depend on previous layers so please make sure to check the previous layers before moving on to the next one.

Single Head Causal Attention (20 points)

We will first implement the single head causal attention layer. This layer is the same as the scaled dot product attention layer but with a causal mask to prevent the model from looking into the future.

Recall that Each head has a Key, Query and Value Matrix and the scaled dot product attention is calculated as :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

where d_k is the dimension of the key matrix.

Figure below from the original paper shows how the layer is to be implemented.

image

Image credits: Attention is All You Need Paper

Please complete the `SingleHeadAttention` class in `model.py`

```
In [ ]: model = SingleHeadAttention(MiniGPTConfig.embed_dim, MiniGPTConfig.embed_dim//4, MiniGPT

        tests.check_singleheadattention(model, path_to_gpt_tester, device)

Out[ ]: 'TEST CASE PASSED!!!'
```

Multi Head Attention (10 points)

Now that we have a single head working, we will now scale this across multiple heads, remember that with multihead attention we compute perform head number of parallel attention operations. We then concatenate the outputs of these parallel attention operations and project them back to the desired dimension using an output linear layer.

Figure below from the original paper shows how the layer is to be implemented.

image

Image credits: Attention is All You Need Paper

Please complete the `MultiHeadAttention` class in `model.py` using the `SingleHeadAttention` class implemented earlier.

In [ ]:
```
model = MultiHeadAttention(MiniGPTConfig.embed_dim, MiniGPTConfig.num_heads)

tests.check_multiheadattention(model, path_to_gpt_tester, device)
```

Out[ ]:
```
'TEST CASE PASSED!!!'
```

Feed Forward Layer (5 points)

As discussed in lecture, the attention layer is completely linear, in order to add some non-linearity we add a feed forward layer. The feed forward layer is a simple two layer MLP with a GeLU activation in between.

Please complete the `FeedForwardLayer` class in `model.py`

In [ ]:
```
model = FeedForwardLayer(MiniGPTConfig.embed_dim)

tests.check_feedforward(model, path_to_gpt_tester, device)
```

Out[ ]:
```
'TEST CASE PASSED!!!'
```

LayerNorm (10 points)

We will now implement the layer normalization layer. Layernorm is used across the model to normalize the activations of the previous layer. Recall that the equation for layernorm is given as:

$$\text{LayerNorm}(x) = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \odot \gamma + \beta \tag{2}$$

With the learnable parameters γ and β.

Remember that unlike batchnorm we compute statistics across the feature dimension and not the batch dimension, hence we do not need to keep track of running averages.

Please complete the `LayerNorm` class in `model.py`

In [ ]:
```
model = LayerNorm(MiniGPTConfig.embed_dim)
tests.check_layernorm(model, path_to_gpt_tester, device)
```

```
'TEST CASE PASSED!!!'
```

Out[ ]:

Transformer Layer (15 points)

We have now implemented all the components of the transformer layer. We will now put it all together to create a transformer layer. The transformer layer consists of a multi head attention layer, a feed forward layer and two layer norm layers.

Please use the following order for each component (Varies slightly from the original attention paper):

1. LayerNorm
2. MultiHeadAttention
3. LayerNorm
4. FeedForwardLayer

Remember that the transformer layer also has residual connections around each sublayer.

The below figure shows the structure of the transformer layer you are required to implement.

prenorm_transformer

Image Credit : CogView

Implement the `TransformerLayer` class in `model.py`

In [ ]:
```
model =  TransformerLayer(MiniGPTConfig.embed_dim, MiniGPTConfig.num_heads)
tests.check_transformer(model, path_to_gpt_tester, device)
```
Out[ ]:
```
'TEST CASE PASSED!!!'
```

Putting it all together : MiniGPT (15 points)

We are now ready to put all our layers together to build our own MiniGPT!

The MiniGPT model consists of an embedding layer, a positional encoding layer and a stack of transformer layers. The output of the transformer layer is passed through a linear layer (called head) to get the final output logits. Note that in our implementation we will use weight tying between the embedding layer and the final linear layer. This allows us to save on parameters and also helps in training.

Implement the `MiniGPT` class in `model.py`

In [ ]:
```
model = MiniGPT(MiniGPTConfig)
tests.check_miniGPT(model, path_to_gpt_tester, device)
```
Out[ ]:
```
'TEST CASE PASSED!!!'
```

Attempt at training the model (5 points)

We will now attempt to train the model on the text data. We will use the same text data as before. Please scale down the model parameters in the config file to a smaller value to make training feasible.

Use the same training script we built for the Bigram model to train the MiniGPT model. If you implemented it correctly it should work just out of the box!

NOTE : We will not be able to train the model to completion in this assignment. Unfortunately, without access to a relatively powerful GPU, training a large enough model to see good generation is not feasible. However, you should be able to see the loss decreasing over time. To get full points for this section it is sufficient to show that the loss is decreasing over time. You do not need to run this for more than 5000 iterations or 1 hour of training.

Train and Valid Plots

Show the training and validation loss plots

In [111…

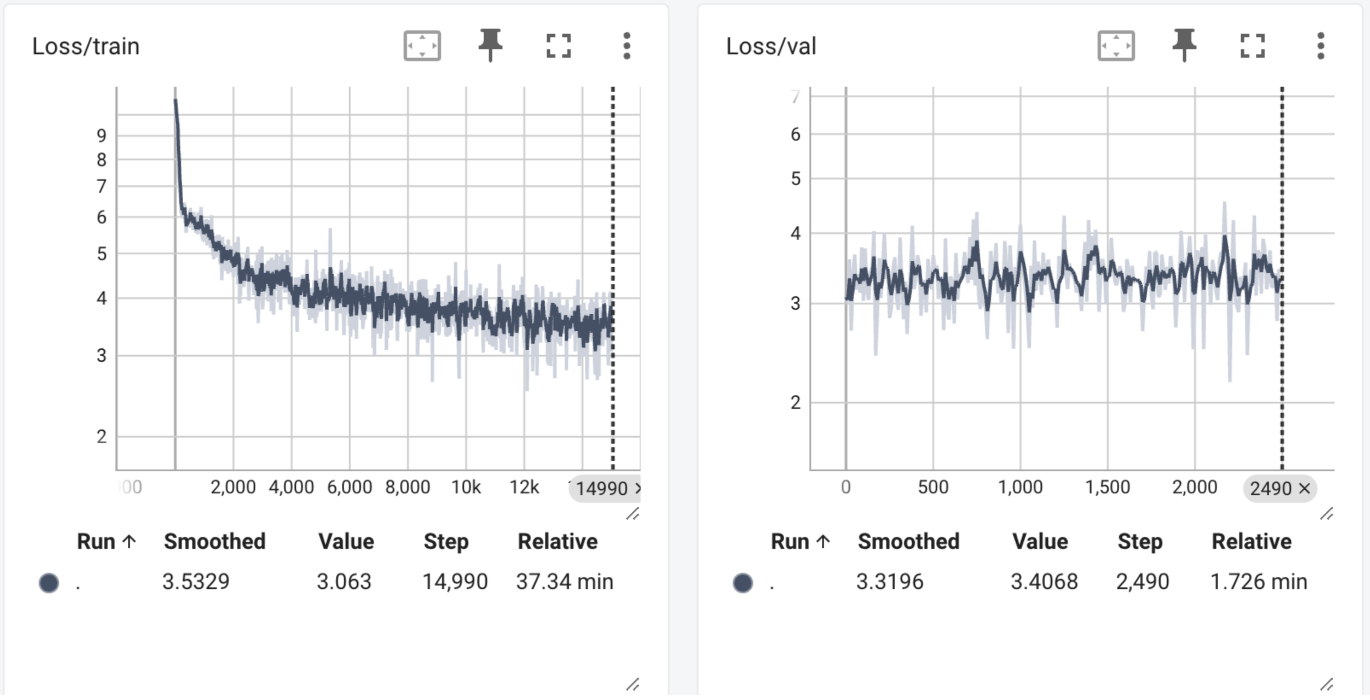
```
%load_ext tensorboard
!rm -rf ./logs2/*
!python train.py
%tensorboard --logdir=logs2
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
number of trainable parameters: 3.32M
2024-05-19 00:43:57.398386: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:92
61] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN wh
en one has already been registered
2024-05-19 00:43:57.398473: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:60
7] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT whe
n one has already been registered
2024-05-19 00:43:57.400153: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1
515] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS
when one has already been registered
2024-05-19 00:43:57.410056: I tensorflow/core/platform/cpu_feature_guard.cc:182] This Te
nsorFlow binary is optimized to use available CPU instructions in performance-critical o
perations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow
with the appropriate compiler flags.
2024-05-19 00:43:59.184171: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-T
RT Warning: Could not find TensorRT
Iteration: 0  Train Loss: 10.824546813964844
Iteration: 1000  Train Loss: 5.547506809234619
Iteration: 2000  Train Loss: 4.441906929016113
Iteration: 3000  Train Loss: 3.783728837966919
Iteration: 4000  Train Loss: 4.016797065734863
Iteration: 5000  Train Loss: 4.399841785430908
Iteration: 6000  Train Loss: 4.356977939605713
Iteration: 7000  Train Loss: 4.088742256164551
Iteration: 8000  Train Loss: 3.3858611583709717
Iteration: 9000  Train Loss: 3.2699813842773438
Iteration: 10000  Train Loss: 4.60744047164917
Iteration: 11000  Train Loss: 3.3521039485931396
Iteration: 12000  Train Loss: 3.5750226974487305
Iteration: 13000  Train Loss: 3.6776010990142822
Iteration: 14000  Train Loss: 3.7251229286193848
Iteration: 0  Val Loss: 3.0822598934173584
Iteration: 100  Val Loss: 3.031449317932129
Iteration: 200  Val Loss: 3.0682263374328613
Iteration: 300  Val Loss: 2.9213666915893555
Iteration: 400  Val Loss: 3.3106250762939453
Iteration: 500  Val Loss: 3.6563332080841064
Iteration: 600  Val Loss: 3.354074001312256
Iteration: 700  Val Loss: 3.2614285945892334
Iteration: 800  Val Loss: 2.948302984237671
Iteration: 900  Val Loss: 3.1433119773864746
Iteration: 1000  Val Loss: 4.14216470718383 8
Iteration: 1100  Val Loss: 3.5624959468841553
Iteration: 1200  Val Loss: 3.1988210678100586
```

```
Iteration: 1300  Val Loss: 3.366063117980957
Iteration: 1400  Val Loss: 3.4261200428009033
Iteration: 1500  Val Loss: 3.4589626789093018
Iteration: 1600  Val Loss: 3.0021231174468994
Iteration: 1700  Val Loss: 2.6791553497314453
Iteration: 1800  Val Loss: 3.099566698074341
Iteration: 1900  Val Loss: 3.1719443798065186
Iteration: 2000  Val Loss: 3.0913825035095215
Iteration: 2100  Val Loss: 3.5873665809631348
Iteration: 2200  Val Loss: 2.1747260093688965
Iteration: 2300  Val Loss: 2.4284772872924805
Iteration: 2400  Val Loss: 3.7463834285736084
Reusing TensorBoard on port 6006 (pid 57631), started 3:19:13 ago. (Use '!kill 57631' to
kill it.)
```

In [3]:
```python
from IPython.display import Image
Image('./MiniGPT_v1.png')
```

Out[3]:



Run ↑	Smoothed	Value	Step	Relative
● .	3.5329	3.063	14,990	37.34 min

Run ↑	Smoothed	Value	Step	Relative
● .	3.3196	3.4068	2,490	1.726 min

Generation (5 points)

Perform generation with the model that you trained. Copy over the generation function you used for the Bigram model not the `miniGPT` class and generate a mini story using the same seed sentence.

```
`"once upon a time"`
```

In [112…
```python
best_model = torch.load('./models/minigpt_best_model.pth')
# best_model = MiniGPT(MiniGPTConfig)
gen_sent = "Once upon a time"
gen_tokens = torch.tensor(tokenizer.encode(gen_sent))
print("Generating text starting with:", gen_tokens.shape)
gen_tokens = gen_tokens.to(device)
best_model.eval()
print(
    tokenizer.decode(
        best_model.generate(gen_tokens, max_new_tokens=200).squeeze().tolist()
    )
)
```

```
Generating text starting with: torch.Size([4])
Once upon a time, there was a wardrobe. The garden, sneuggap its friends were nest. She
```

```
carried the table. They went it under shiny telephoney and gone at the big, not help. On
e day, Lily thought an end. From that day on, a big bird flew up and tried to play with
his friends! It was dead," before:
Later that made fun long. The dog liked to play outside and liked his friends all down t
he park. They played together. Then, He said, " Wax him can come to see the caulin.
They watched the man danced in the park, the mess. Spot was very sm course, Mom. Now, Fl
uffy said, "You you open me. Are you time, Tom said, �Mommyeies can’Now if I want to tr
y?"
On the park, he saw a garden. It is playing outside. She wanted to break he play with th
e cat. She put inside they took him. He
```

Please answer the following questions.

1. What can we say about the generated text in terms of grammar and coherence?
2. If the model is scaled with more parameters do you expect the GPT model to get substantially better? Why or why not?

Answer 1: The generated text is much more coherent compared to the bigram model. The grammatical accuracy of the text is also better than the bigram model. However, there are still a lot of errors in the text. This is most likely due to insufficient training time. We can also scale up the model architecture to handle large data better as it can help the model capture more complex features and patterns in the text.

2: Increasing the number of parameters generally increases the model's capacity to learn complex patterns in the data. With more parameters, the model can capture finer details and nuances, potentially leading to better performance, especially on complex tasks or large datasets. However, the increasing the model capacity can also lead to issues such as longer training time, more computational resources and overfitting problems if the either the quantity of data is insufficient or the data is not diverse enough.

Scaling up the model (5 points)

To show that scale indeed will help the model learn we have trained a scaled up version of the model you just implemented. We will load the weights of this model and generate a mini story using the same seed sentence. Note that if you have implemented the model correctly just scaling the parameters and adding a few bells and whistles to the training script will results in a model like the one we will load now.

In [95]:
```python
from model import MiniGPT
from config import MiniGPTConfig
```

In [96]:
```python
path_to_trained_model = "pretrained_models/best_train_loss_checkpoint.pth"
```

In [97]:
```python
ckpt = torch.load(path_to_trained_model, map_location=device) # remove map location if u
```

In [98]:
```python
# Set the configs for scaled model
MiniGPTConfig.context_length = 512
MiniGPTConfig.embed_dim = 256
MiniGPTConfig.num_heads = 16
MiniGPTConfig.num_layers = 8
```

In [99]:
```python
# Load model from checkpoint
model = MiniGPT(MiniGPTConfig)
model.load_state_dict(ckpt["model_state_dict"])
```

Out[99]:
```
<All keys matched successfully>
```

In [100…
```python
tokenizer = tiktoken.get_encoding("gpt2")
```

In [101…
```
model.to(device)
gen_sent = "Once upon a time"
gen_tokens = torch.tensor(tokenizer.encode(gen_sent))
print("Generating text starting with:", gen_tokens.shape)
gen_tokens = gen_tokens.to(device)
model.eval()
print(
    tokenizer.decode(
        model.generate(gen_tokens, max_new_tokens=200).squeeze().tolist()
    )
)
```

```
Generating text starting with: torch.Size([4])
Once upon a time, a little red bird named Luna lived in a high tree. Luna was a very hap
py flower.
One day, a big storm came. The wind was very strong. It blew the raft far away. Tom felt
someone calling him.
Lily took the note and decided to put it in the washing machine. Timmy had to wipe it cl
ean. He was tired and said, "I am sleepy, but it's still full of tough things. He felt m
uch better because he fixed the pit.Once upon a time, there was a young little girl name
d Lily. She had a big, fat tummy and loved to eat yummy sandwiches. One day, Timmy's mom
asked him to help organize the laundry. Timmy didn't know how to put everything up, so s
he asked her mom to help her.
"What is that, Mommy?" asked Lily. "I don't know if I can do it!"
Tim went to the 1946boat and
```

Bonus (5 points)

The following are some open ended questions that you can attempt if you have time. Feel free to propose your own as well if you have an interesting idea.

1. The model we have implemented is a decoder only model. Can you implement the encoder part as well? This should not be too hard to do since most of the layers are already implemented.
2. What are some improvements we can add to the training script to make training more efficient and faster? Can you should if any improvements you add help in training the model better?
3. Can you implement a beam search decoder to generate the text instead of greedy decoding? Does this help in generating better text?
4. Can you further optimize the model architecture? For example, can you implement Multi Query Attention or Grouped Query Attention to improve the model performance?

We can make the training faster in the following ways:

Data Loading Optimization: Since data loading can often be a bottleneck, especially on CPU, consider using num_workers argument in DataLoader to load data in parallel.

Model Optimization: Optimize the model architecture and hyperparameters. This includes choosing an appropriate optimizer, learning rate scheduler, loss function, and regularization techniques.

Batch Size: Experiment with different batch sizes to find the optimal balance between memory usage and computation speed. Larger batch sizes can sometimes lead to faster training due to better GPU utilization.

Reduce Logging Frequency: Logging can introduce overhead, especially if you're logging many variables frequently. Reduce the frequency of logging or only log key metrics to improve training speed.

In the code, we wrote a new training script, 'training_2.py' and 'config_2.py' with num_workers = 4, batch_size = 32, and log_interval = 50.

This lead to faster training with similar converging loss values.

In [113…
```
%load_ext tensorboard
!rm -rf ./logs4/*
!python train_2.py
%tensorboard --logdir=logs4
```

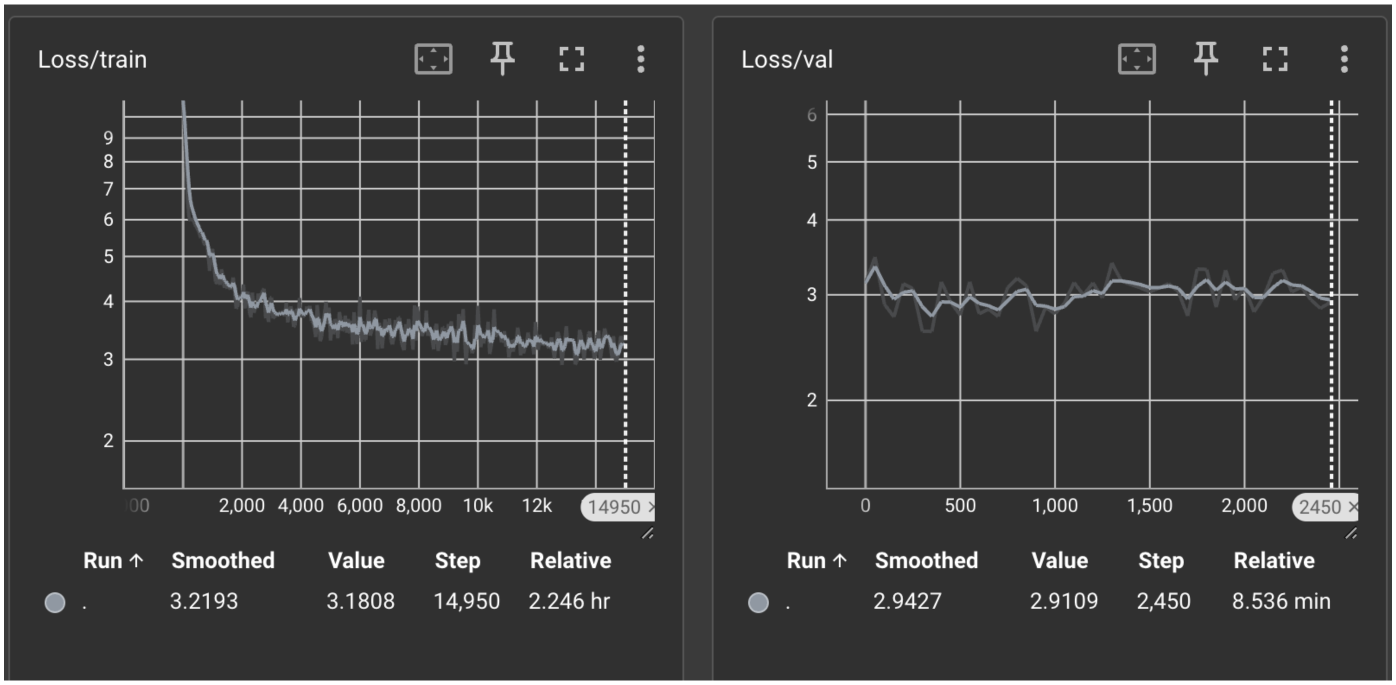
```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:558: UserWarning:
This DataLoader will create 4 worker processes in total. Our suggested max number of wor
ker in current system is 2, which is smaller than what this DataLoader is going to creat
e. Please be aware that excessive worker creation might get DataLoader running slow or e
ven freeze, lower the worker number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
number of trainable parameters: 3.32M
2024-05-19 01:23:21.061512: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:92
61] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN wh
en one has already been registered
2024-05-19 01:23:21.061612: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:60
7] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT whe
n one has already been registered
2024-05-19 01:23:21.064168: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1
515] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS
when one has already been registered
2024-05-19 01:23:21.078026: I tensorflow/core/platform/cpu_feature_guard.cc:182] This Te
nsorFlow binary is optimized to use available CPU instructions in performance-critical o
perations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow
with the appropriate compiler flags.
2024-05-19 01:23:22.926678: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-T
RT Warning: Could not find TensorRT
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:558: UserWarning:
This DataLoader will create 4 worker processes in total. Our suggested max number of wor
ker in current system is 2, which is smaller than what this DataLoader is going to creat
e. Please be aware that excessive worker creation might get DataLoader running slow or e
ven freeze, lower the worker number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
/usr/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was call
ed. os.fork() is incompatible with multithreaded code, and JAX is multithreaded, so this
will likely lead to a deadlock.
  self.pid = os.fork()
Iteration: 0  Train Loss: 10.851312637329102
Iteration: 1000  Train Loss: 5.19314432144165
Iteration: 2000  Train Loss: 4.102783679962158
Iteration: 3000  Train Loss: 4.078032970428467
Iteration: 4000  Train Loss: 3.744541883468628
Iteration: 5000  Train Loss: 3.293703556060791
Iteration: 6000  Train Loss: 4.0989227294921875
Iteration: 7000  Train Loss: 3.673785448074341
Iteration: 8000  Train Loss: 3.5435333251953125
Iteration: 9000  Train Loss: 3.5768680572509766
Iteration: 10000  Train Loss: 3.9121594429016113
Iteration: 11000  Train Loss: 3.3453526496887207
Iteration: 12000  Train Loss: 3.338031768798828
Iteration: 13000  Train Loss: 3.5371856689453125
Iteration: 14000  Train Loss: 3.0966103076934814
Iteration: 0  Val Loss: 3.1386404037475586
Iteration: 100  Val Loss: 2.8914265632629395
Iteration: 200  Val Loss: 3.13159441947937
Iteration: 300  Val Loss: 2.6077311038970947
Iteration: 400  Val Loss: 3.1533844470977783
Iteration: 500  Val Loss: 2.77370023727417
Iteration: 600  Val Loss: 2.789684295654297
Iteration: 700  Val Loss: 2.7599129676818848
```

```
Iteration: 800  Val Loss: 3.1969244480133057
Iteration: 900  Val Loss: 2.607316493988037
Iteration: 1000  Val Loss: 2.7884247303009033
Iteration: 1100  Val Loss: 3.144670009613037
Iteration: 1200  Val Loss: 3.1402127742767334
Iteration: 1300  Val Loss: 3.390859603881836
Iteration: 1400  Val Loss: 3.1198647022247314
Iteration: 1500  Val Loss: 3.031928300857544
Iteration: 1600  Val Loss: 3.130539894104004
Iteration: 1700  Val Loss: 2.766254425048828
Iteration: 1800  Val Loss: 3.3066248893737793
Iteration: 1900  Val Loss: 3.297318696975708
Iteration: 2000  Val Loss: 3.082937479019165
Iteration: 2100  Val Loss: 2.9703898429870605
Iteration: 2200  Val Loss: 3.300368547439575
Iteration: 2300  Val Loss: 3.091968297958374
Iteration: 2400  Val Loss: 2.8483431339263916
```

In [13]:
```
%load_ext tensorboard
%tensorboard --logdir=logs4
```

In [4]:
```
from IPython.display import Image
Image('./MiniGPT_v2.png')
```

Out[4]:



Run ↑	Smoothed	Value	Step	Relative
● .	3.2193	3.1808	14,950	2.246 hr

Run ↑	Smoothed	Value	Step	Relative
● .	2.9427	2.9109	2,450	8.536 min

In [14]:
```
best_model = torch.load('./models/minigpt_best_model_2.pth')
# best_model = MiniGPT(MiniGPTConfig)
gen_sent = "Once upon a time"
gen_tokens = torch.tensor(tokenizer.encode(gen_sent))
print("Generating text starting with:", gen_tokens.shape)
gen_tokens = gen_tokens.to(device)
best_model.eval()
print(
    tokenizer.decode(
        best_model.generate(gen_tokens, max_new_tokens=200).squeeze().tolist()
    )
)
```

```
Generating text starting with: torch.Size([4])
Once upon a time, there was a little girl named Lily. She loved to play with the doors t
hat they all met her friends and saw how they could get away. They cried but proud the d
uck wasumpy man and he walked away.
```

One day, little girl saw a little girl named Lily. She was grateful to reply.
The dog is happy. He makes the leaves and wanted to go in the mysterious barn. It was we
aring full more fun. She loved nice toys. One day, she found a fingers. They put some on
e up and made a big pile of her incredible seems. But, something unexpected happened. Sp
ot was very happy and said, " saw a big owl. Can I read on the spider. You can help his
mom. They had a long time! The town was very ignorant and they could jump away and play
together. They all laughed and the dog and looked inside in the mix the simple cherry. T
hey got closer and playing in the folder.
Sarah was happy.