Transformers

The Decoder

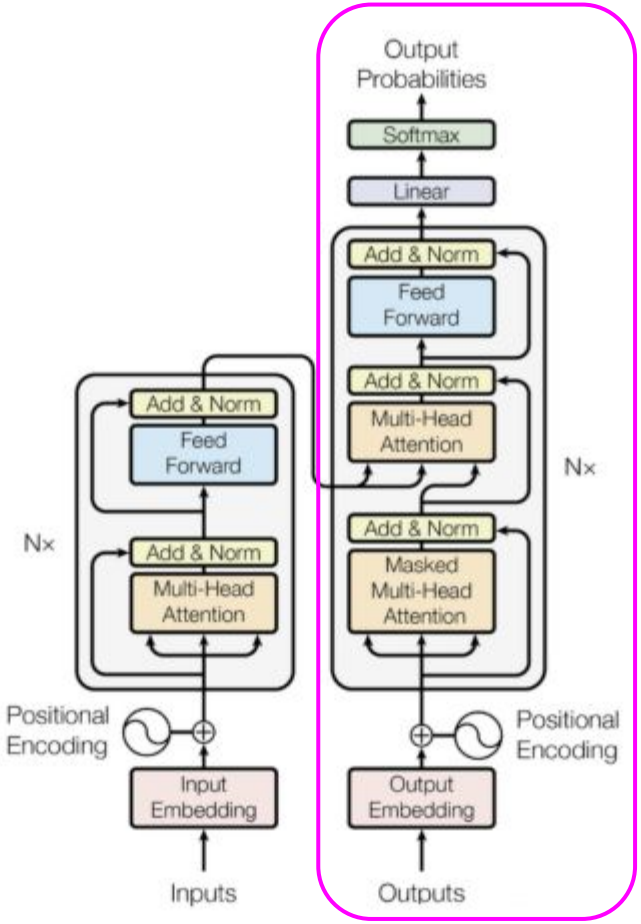
Objectives

- understand the main components of the Transformer
- understand the importance of Attention mechanism
- be able to implement a small/medium Transformer model
- be able to train for long periods of time (several hours)
- be able to do checkpointing and continuous evaluation

GPT

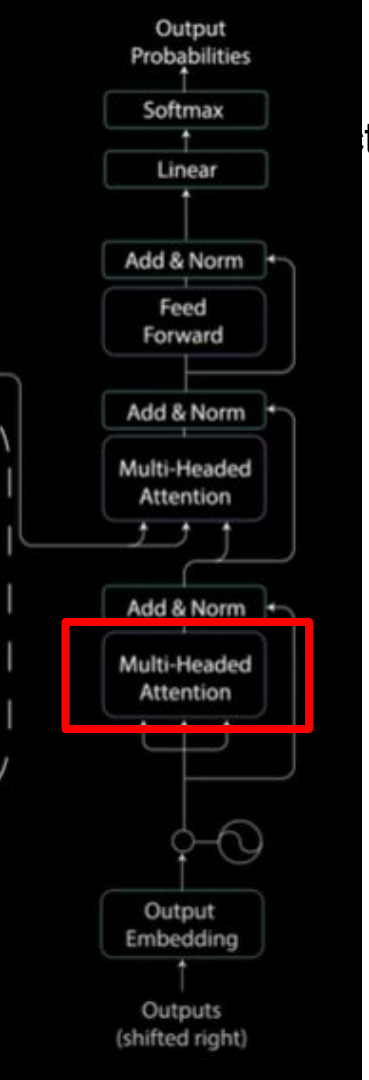
BERT

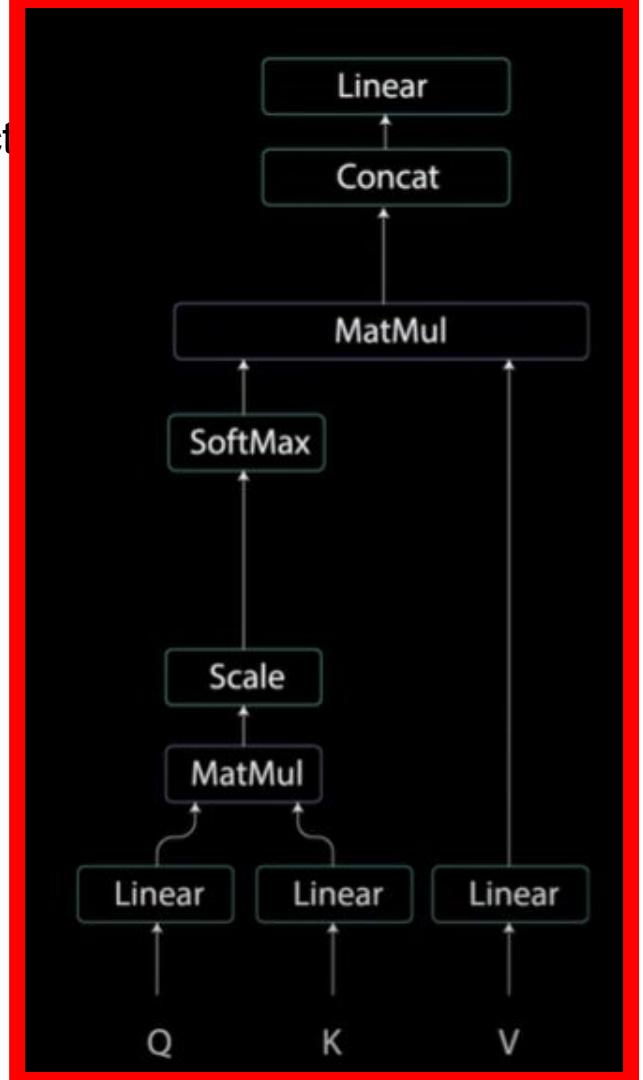
Encoder



GPT

Decoder

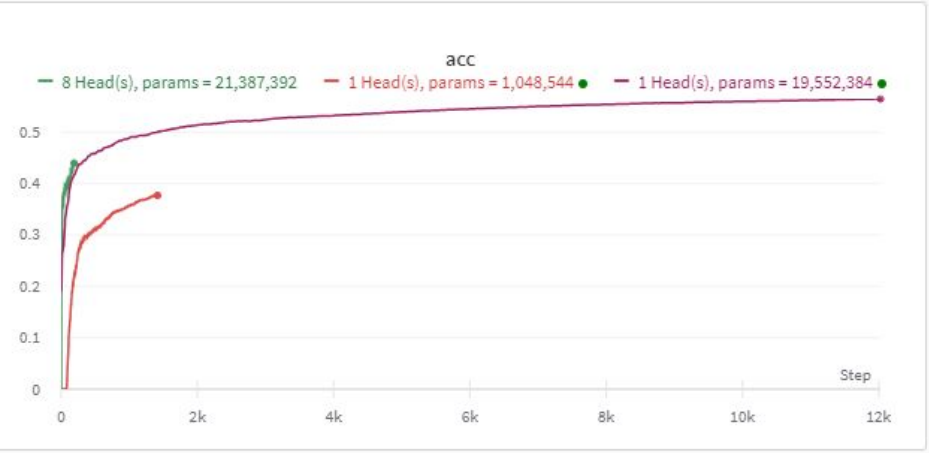


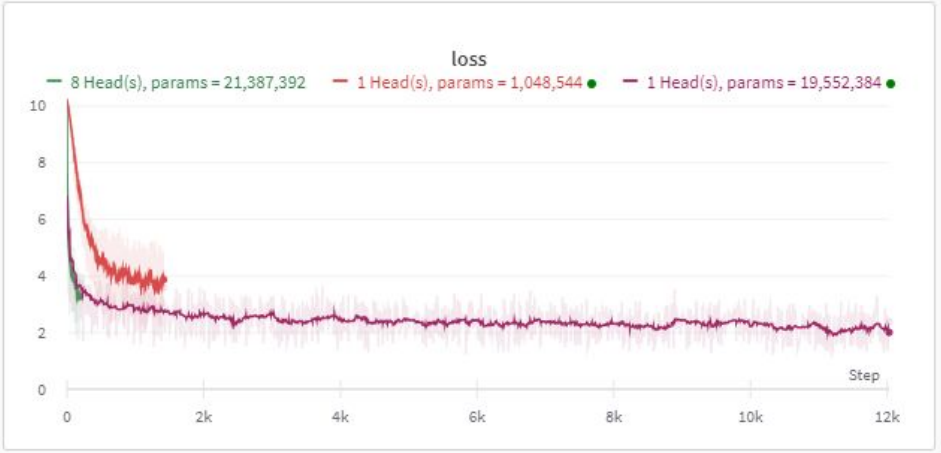


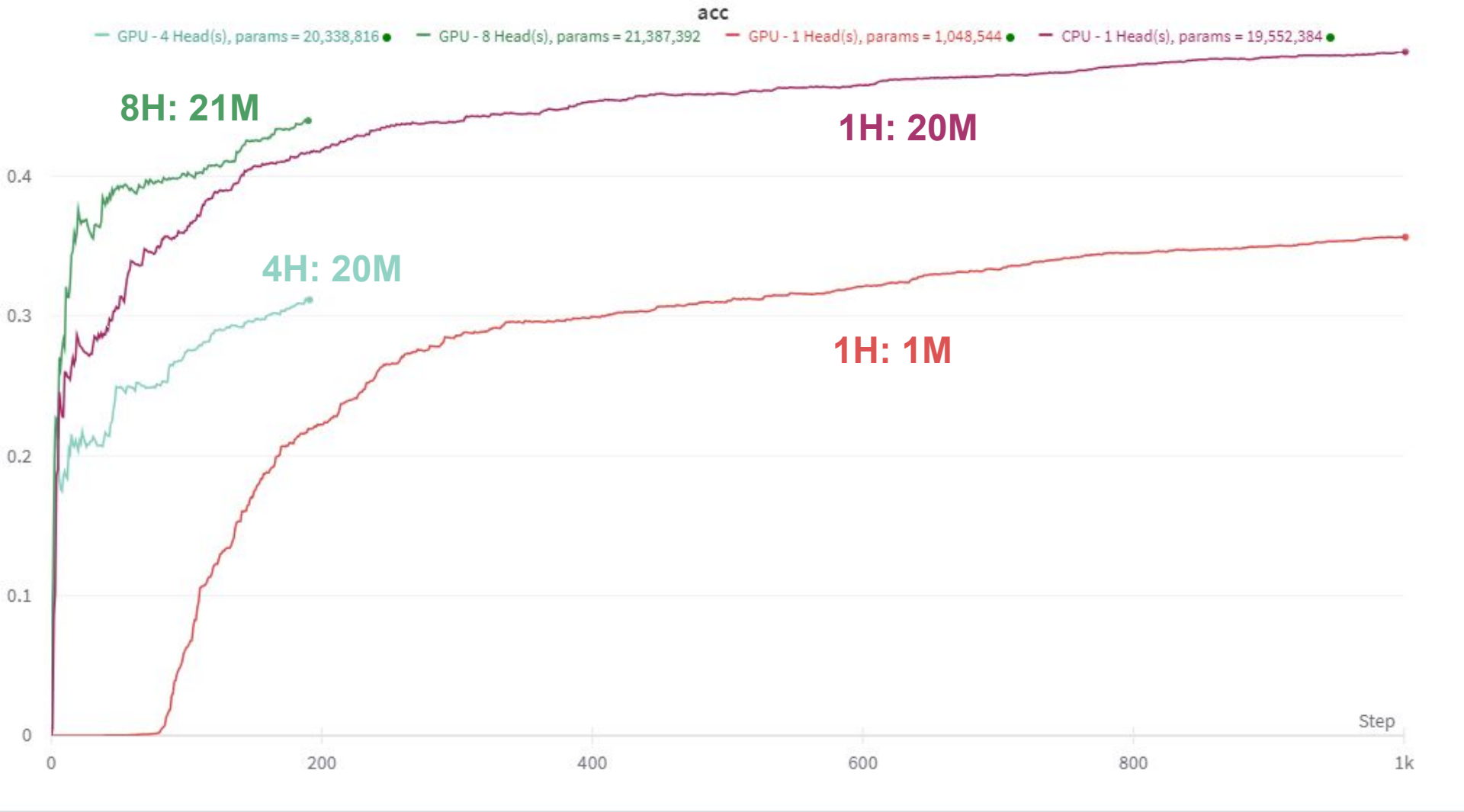

```
class transformer(torch.nn.Module):
  def __init__(self, vocab_size, input_embedding_dimensions, max_sequ
    super(transformer, self).__init__()
    #positional encoding
    self.input_embeddings = nn.Embedding(vocab_size, input_embedding_
    self.positional_encoding = positional_encoding(max_sequence_lengt
    #MHA
    self.mha_first_linear_layer = nn.Linear(input_embedding_dimension
    self.mha_final_linear_layer = nn.Linear(input_embedding_dimension
    #ADD&NORMALIZE
    self.mha_norm_layer = nn.LayerNorm(input_embedding_dimensions) #
    #FEED FORWARD
    self.ff_first_linear = nn.Linear(input_embedding_dimensions,input
    self.ff_relu = nn.ReLU()
    self.ff_second_linear = nn.Linear(input_embedding_dimensions,inpu
    self.ff_norm_layer = nn.LayerNorm(input_embedding_dimensions) # t
    #final linear layer
    self.final_linear_layer = nn.Linear(input_embedding_dimensions,vo
```

```
def forward(self, inputs):
  #positional encoding
  input_embeddings = self.input_embeddings(inputs)
  positional_embeddings = input_embeddings + self.posit
  #MHA
  q,k,v = self.mha_first_linear_layer(positional_embedd
  score_matrix = q @ k.transpose(-1,-2)
  score_matrix = score_matrix/(input_embedding_dimensio
  mask = torch.tril(torch.ones(score_matrix.size(1), sc
  score_matrix = score_matrix.masked_fill(mask, float('
  att_weights = torch.softmax(score_matrix,dim=-1)
  output = att_weights @ v
  final_mha_output = self.mha_final_linear_layer(output
```

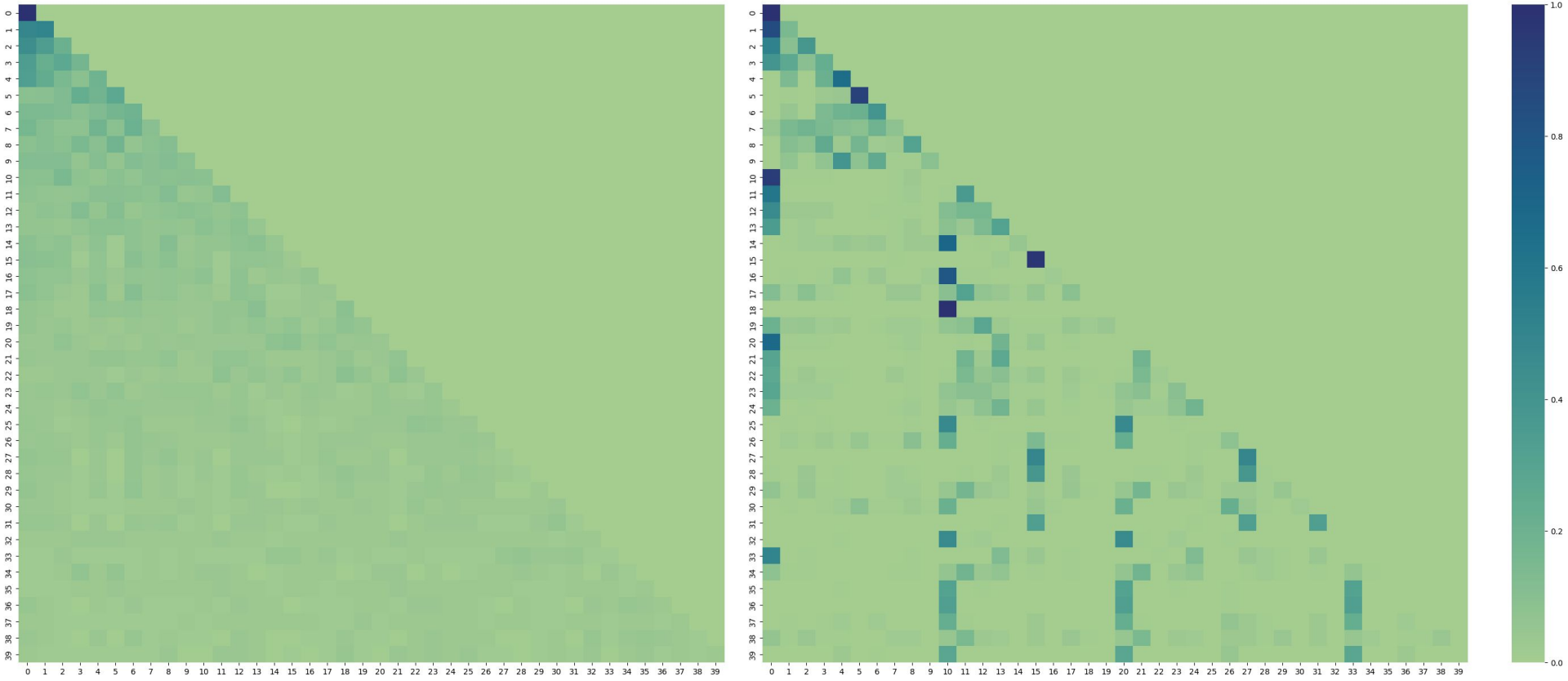
Training


acc
8 Head(s), params = 21,387,392
1 Head(s), params = 1,048,544
1 Head(s), params = 19,552,384
0.5
0.4
0.3
0.2
0.1
0
Step
0
2k
4k
6k
8k
10k
12k

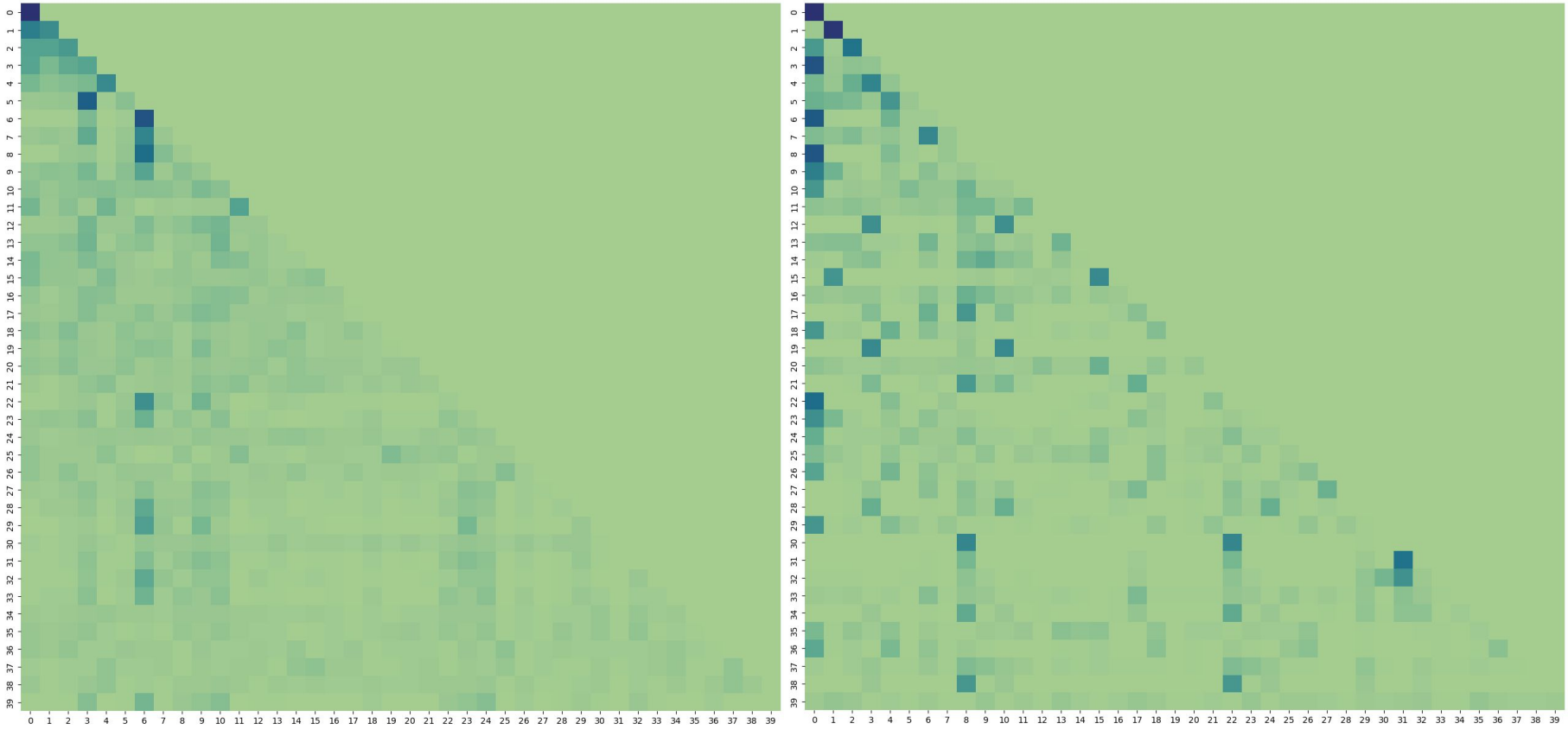

loss
8 Head(s), params = 21,387,392
1 Head(s), params = 1,048,544
1 Head(s), params = 19,552,384
10
8
6
4
2
0
Step
0
2k
4k
6k
8k
10k
12k


acc
GPU - 4 Head(s), params = 20,338,816
GPU - 8 Head(s), params = 21,387,392
GPU - 1 Head(s), params = 1,048,544
CPU - 1 Head(s), params = 19,552,384
8H: 21M
1H: 20M
4H: 20M
1H: 1M
0.4
0.3
0.2
0.1
0
0
200
400
600
800
1k
Step

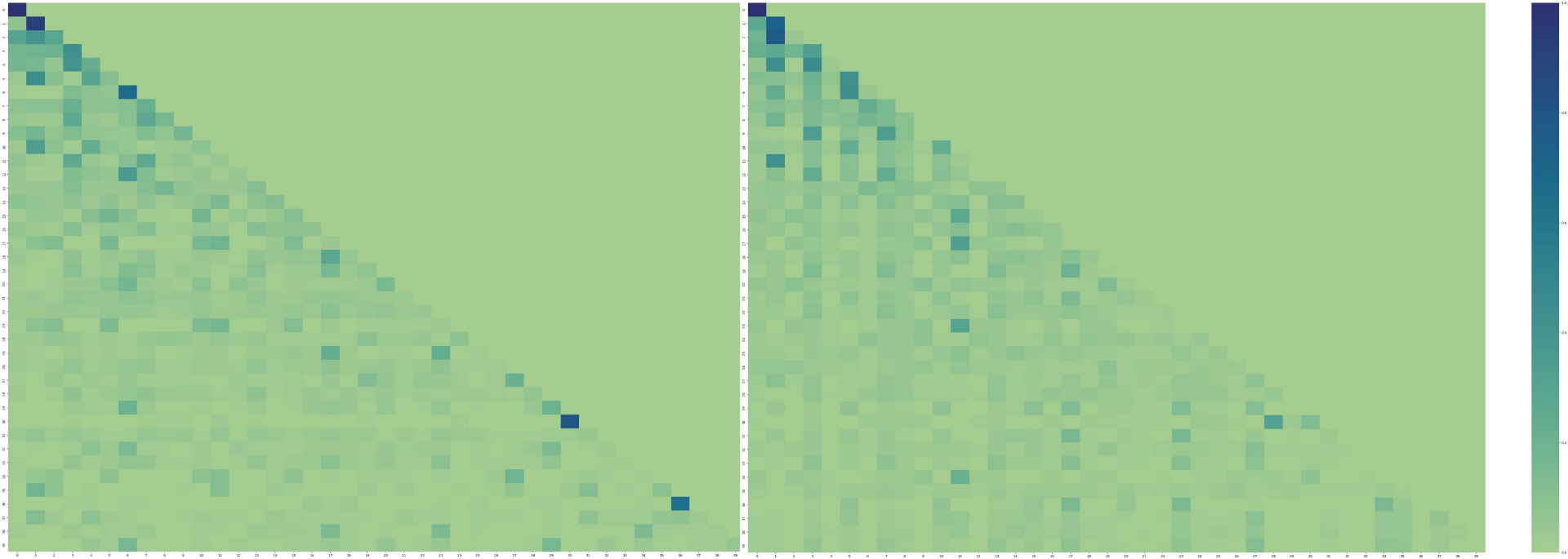
Attention: Initial VS Trained (280K)



Attention: With VS Without Positional Encoding (120k)

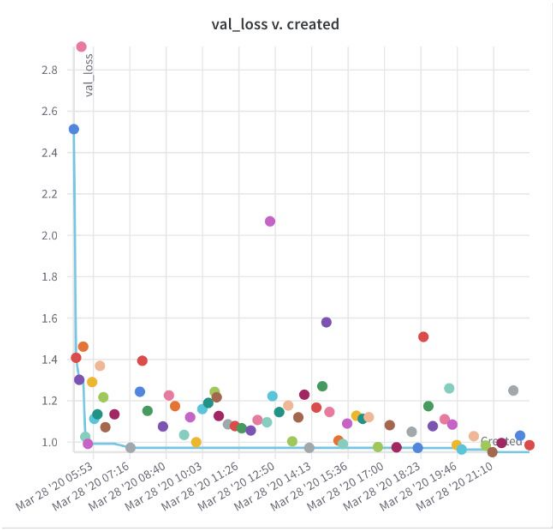


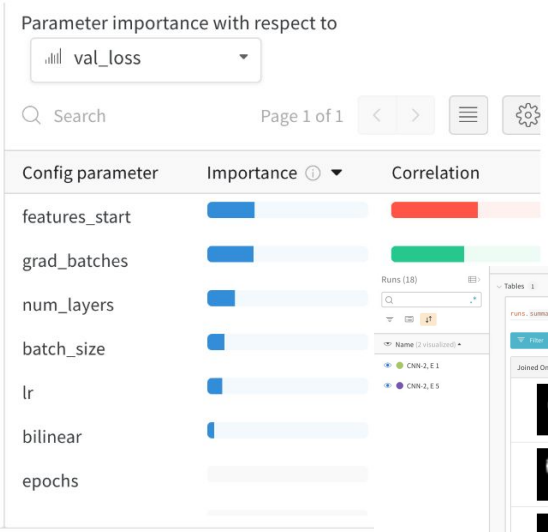
Attention: Different Heads (68,000)

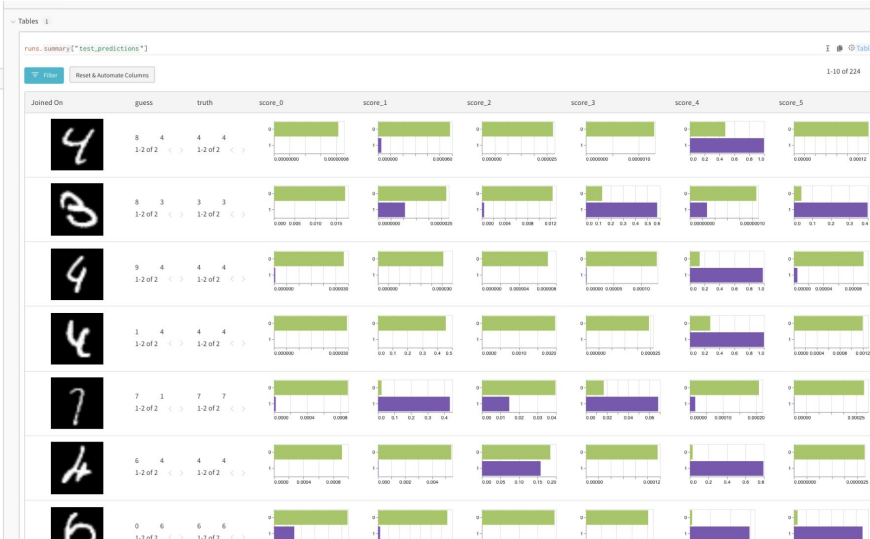


Weights and Biases - Potential

Follow along with a video tutorial!







GPU

1. **Large Model Size:**
   - A model with a large number of parameters requires more GPU memory and can lead to slower computations.
2. **Large Batch Size:**
   - Larger batch sizes can be beneficial for parallelism but may also lead to increased GPU memory usage and slower computations if the GPU memory is insufficient.
3. **Inefficient Memory Usage:**
   - Inefficient use of GPU memory, such as unnecessary allocations or not releasing memory properly, can lead to slowdowns.
4. **Data Transfer Between CPU and GPU:**
   - Frequent data transfers between CPU and GPU can introduce overhead. Minimizing these transfers can improve performance.
5. **Complex Operations:**
   - Certain complex operation
     computationally intensive
6. **Inefficient Tensor Operations**
   - Writing inefficient code wit
     computations. For exampl
7. **Insufficient Parallelism:**
   - If the GPU is not fully utilize
     computation. Ensuring tha
     parallel processing can im
8. **Inefficient GPU Kernel Calls:**
   - Inefficient use of GPU kern
     performance. Frameworks
     implementations for comm
9. **Out-of-Memory Errors:**
   - Running out of GPU memo
     essential to manage memo
     model.

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
aten::empty	0.03%	143.000us	0.03%	143.000us	143.000us	1
aten::random_	0.00%	19.000us	0.00%	19.000us	19.000us	1
aten::item	0.00%	4.000us	0.00%	7.000us	7.000us	1
aten::_local_scalar_dense	0.00%	3.000us	0.00%	3.000us	3.000us	1
enumerate(DataLoader)#_SingleProcessDataLoaderIter._...	0.71%	3.942ms	13.09%	73.004ms	73.004ms	1
aten::empty	0.00%	13.000us	0.00%	13.000us	13.000us	1
aten::random_	0.00%	7.000us	0.00%	7.000us	7.000us	1
aten::item	0.00%	1.000us	0.00%	2.000us	2.000us	1
aten::_local_scalar_dense	0.00%	1.000us	0.00%	1.000us	1.000us	1
aten::randperm	0.03%	160.000us	12.23%	68.211ms	68.211ms	1
aten::empty	0.01%	39.000us	0.01%	39.000us	39.000us	1
aten::randperm	12.19%	68.005ms	12.20%	68.012ms	68.012ms	1
aten::scalar_tensor	0.00%	6.000us	0.00%	6.000us	6.000us	1
aten::resize_	0.00%	1.000us	0.00%	1.000us	1.000us	1
aten::resize_	0.00%	0.000us	0.00%	0.000us	0.000us	1
aten::detach	0.00%	5.000us	0.00%	24.000us	24.000us	1
detach	0.00%	19.000us	0.00%	19.000us	19.000us	1
aten::to	0.00%	4.000us	0.00%	4.000us	4.000us	1
aten::resolve_conj	0.00%	0.000us	0.00%	0.000us	0.000us	1
aten::resolve_neg	0.00%	0.000us	0.00%	0.000us	0.000us	1
aten::empty	0.00%	16.000us	0.00%	16.000us	16.000us	1
aten::to	0.00%	1.000us	0.00%	1.000us	1.000us	1

...

Self CPU time total: 272.339ms

What worked in coding the architecture

Building the most simple structure

Moving very slowly

Being a maniac about input/output shapes

Add complexities - One at a time

At every stage, when I think i got it right, I assume i got it wrong

- Talk to others
- Eavesdropping