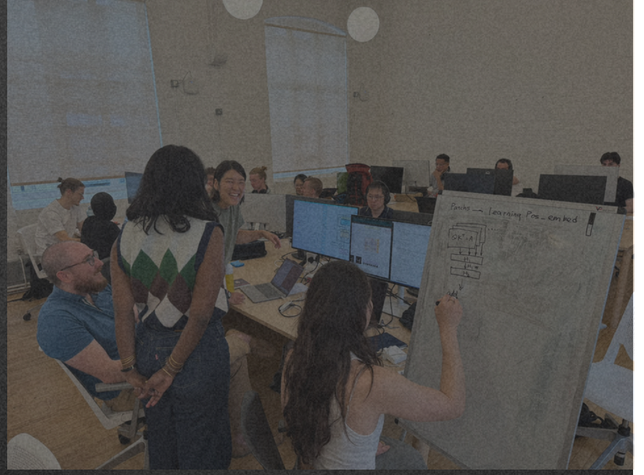
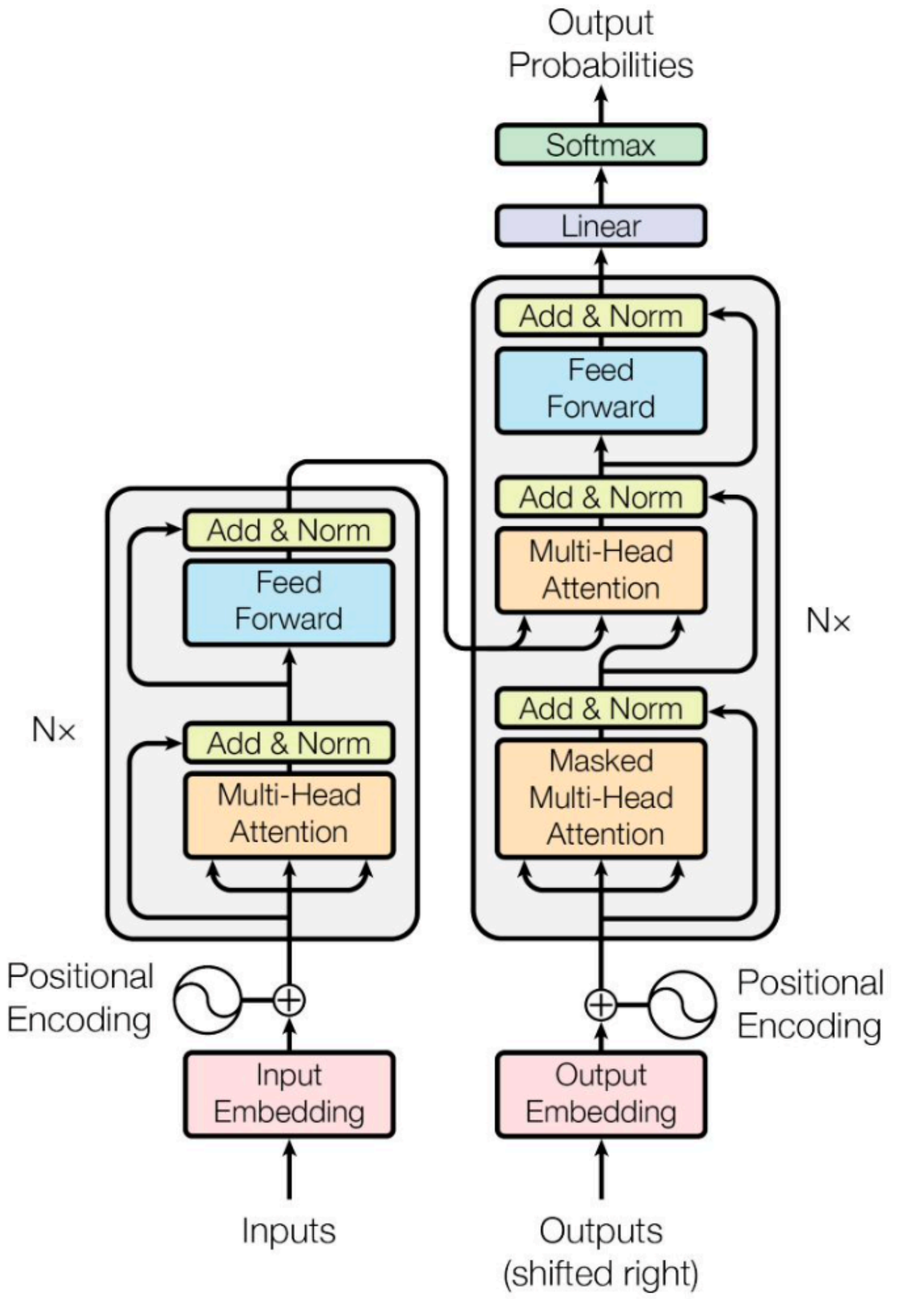
ESPER - HELEN - APARNA - ADAM

RECURRANT REBELS

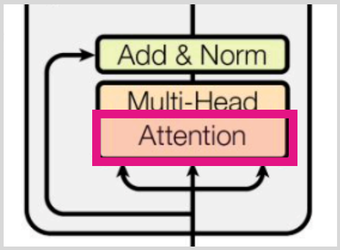
TRANSFORMERS AND MNIST

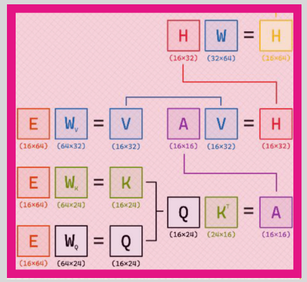


ARCHITECTURE

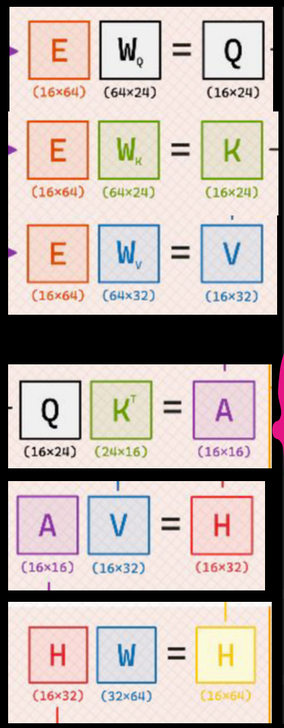


GOING INTO THE DEPTHS OF THE CODE



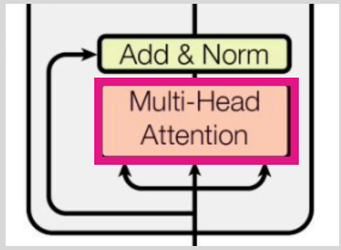


SINGLE HEADED ATTENTION



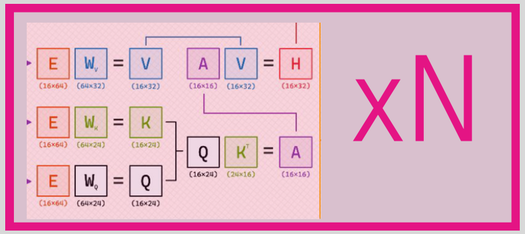

```
Q = self.query_proj(x)   # is nn.Linear(dim_input, dim_k)
K = self.key_proj(x)     # is nn.Linear(dim_input, dim_k)
V = self.val_proj(x)     # is nn.Linear(dim_input, dim_k)
attention = Q @ K.transpose(-2, -1) * self.dim_k ** -0.5
attention = torch.softmax(attention, dim=-1)
out = torch.matmul(attention, V)
return self.out_proj(out)  # is nn.Linear(dim_k, dim_input)
```

GOING INTO THE DEPTHS OF THE CODE



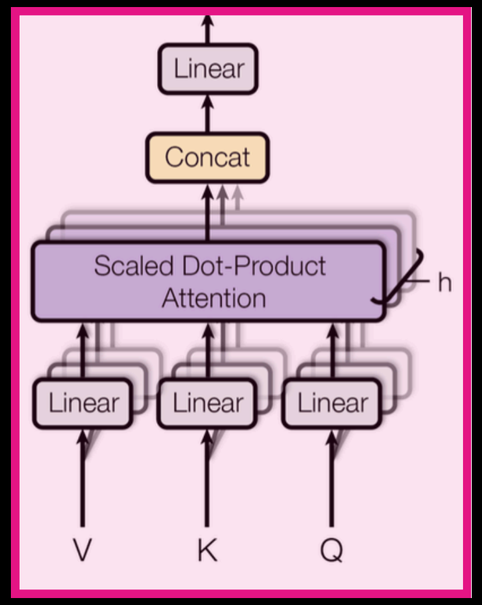
=

=



MULTI HEADED ATTENTION

=

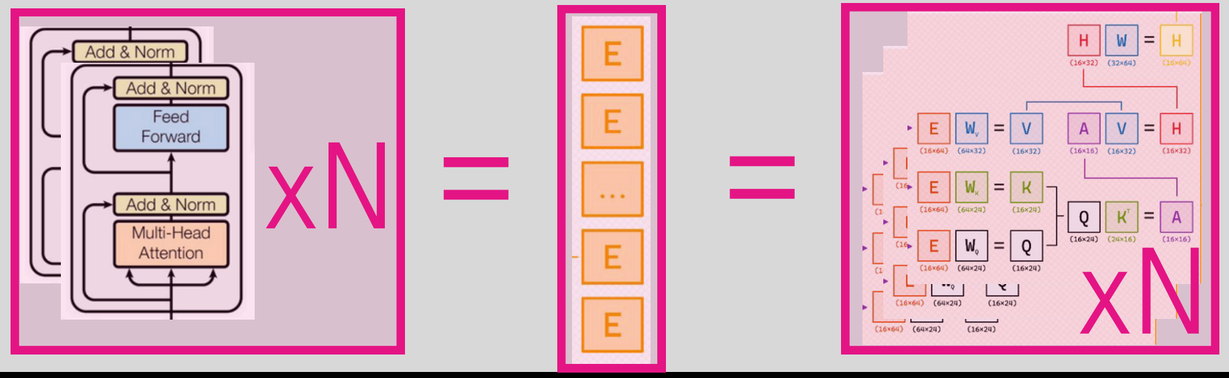



```python
batch_size = x.size(0)
qkv = self.qkv_proj(x) # is nn.Linear(dim_input, dim_k * 3)
Q, K, V = qkv.chunk(3, dim=-1)
attention = Q @ K.transpose(-2, -1) / (self.head_dim ** 0.5)
attention = nn.functional.softmax(attention, dim=-1)
attention = attention @ V  # shape: (B, num_heads, num_patches, head_dim)
# Below concat all outputs of all attention heads into a single embedding per token (patch)
attention = attention.transpose(1, 2).reshape(batch_size, self.num_patches, self.dim_k)
# Below is final projection after concatenating heads
attention = self.out_proj(attention)  # (B, num_patches, dim_k)
```

or

```python
attention, _ = self.attn(x, x, x) # nn.MultiheadAttention(dim_k, num_heads...)
```

GOING INTO THE DEPTHS OF THE CODE



MULTIPLE ENCODER BLOCKS

```
# EncoderBlock has nn.MultiheadAttention & FNN (Linear, ReLU, Linear)
self.encoder_blocks = nn.Sequential(*[
    EncoderBlock(dim_k, num_heads) for _ in range(num_encoder_blocks)
])
...
x = self.encoder_blocks(x)
```

or

```
self.encoder_blocks = nn.ModuleList([
    EncoderBlock(dim_model, num_heads)
    for _ in range(num_layers)
])
...
for encoder_block in self.encoder_blocks:
    x = encoder_block(x. ...)
```

SIDE NOTE: WHY DOES MULTIPLYING = MORE UNDERSTANDING

IMAGINE: LEGO Encoder Teams trying to understand a LEGO instruction book
INSTRUCTION: Attach the red flag to the highest tower

MULTI HEADED ATTENTION

Multiple specialist **heads** come together and:

Head 1: “I’ll look at the **color** word: it’s red”
Head 2: “I’ll look at the **action** word: attach”
Head 3: “I’ll look at **size-related** words: “highest”

=> OK great, now we **can interpret the sentence** in different perspectives/ways

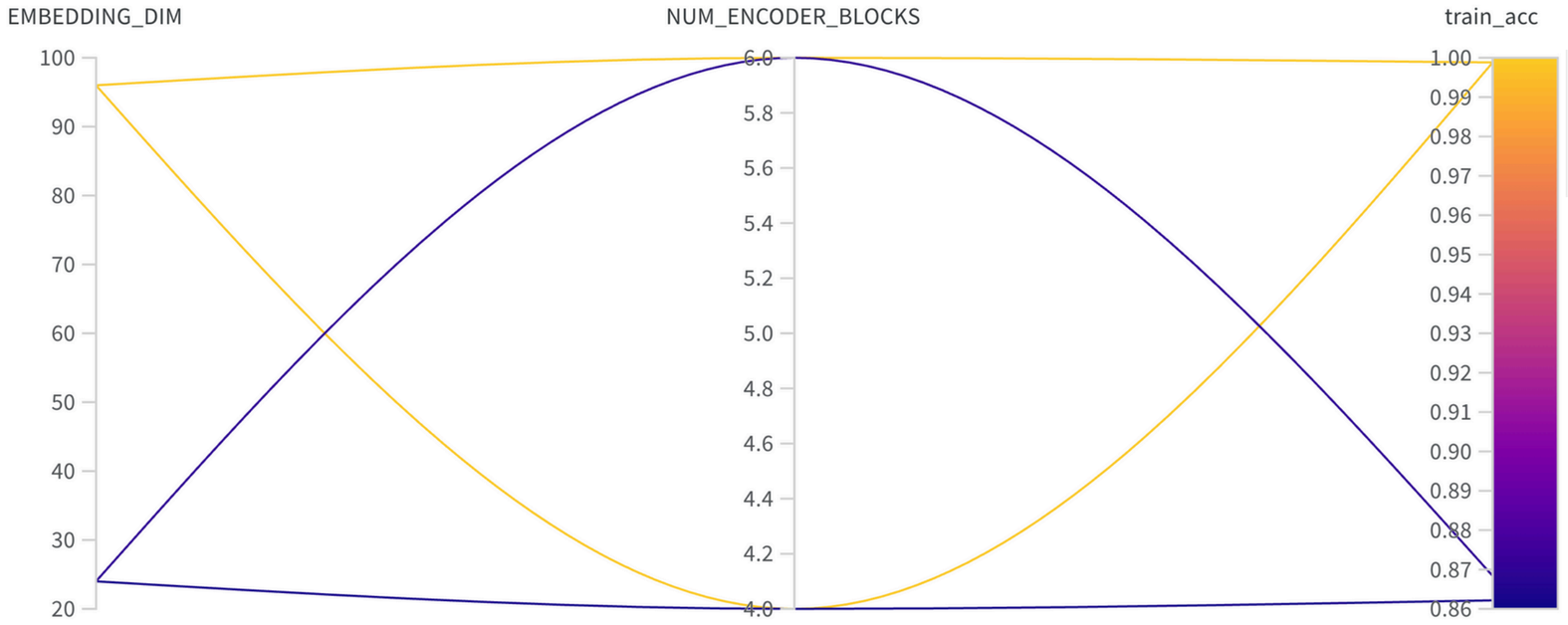
MULTIPLE ENCODER BLOCKS

A chain of **Encoder teams** come together:

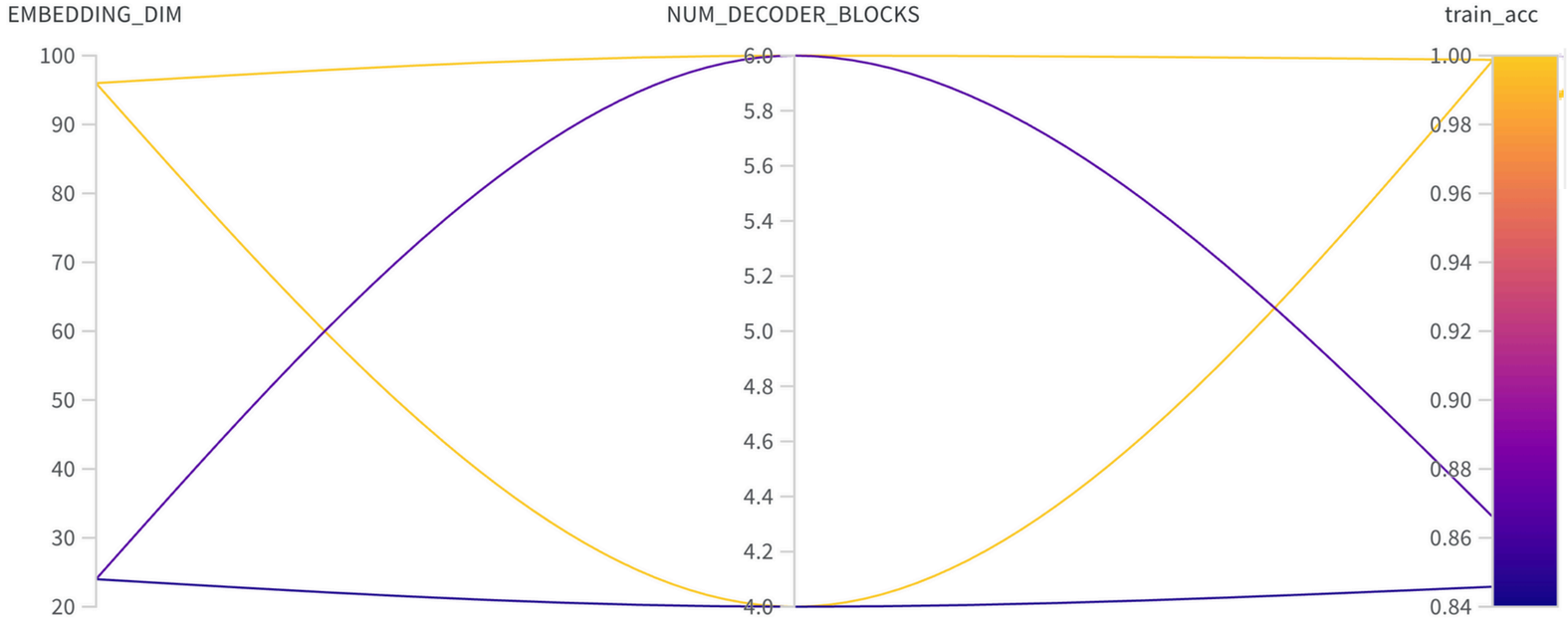
Team 1. “Looks like this task involves a tower, flag and the flag going on top”
Team 2. “Wait, ‘highest’ must mean flag must go on the tallest tower, not just any tower”
Team 3. “Ok, so we find the tallest tower compared to all the towers we’ve built. and put the flag on top.“

=> Ok now we have truly **understand the instruction,** and the final team knows exactly what to do to execute the instruction 🛠️

DEMO TIME



EMBEDDING_DIM: **96**
NUM_ENCODER_BLOCKS: **4**
train_acc: **0.9988**



EMBEDDING_DIM: **96**
NUM_DECODER_BLOCKS: **4**
train_acc: **0.9986**

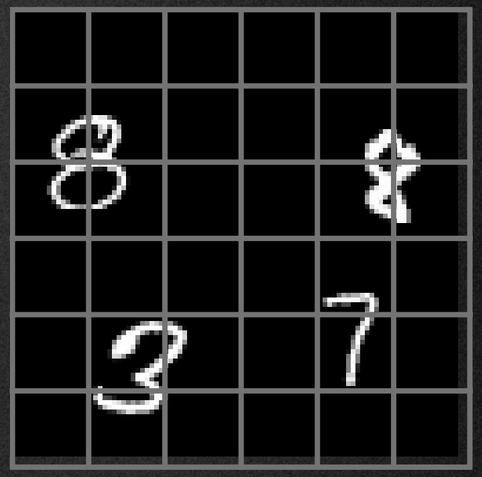
OPTIMISATION

Sweeps

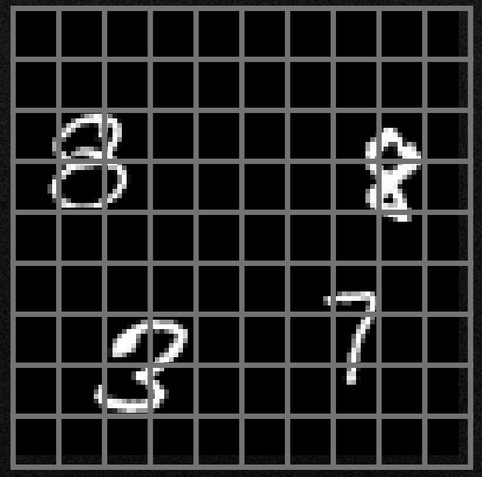
GENERAL LEARNINGS & REFLECTIONS

- Patches
- Loss function adventures
- More Loss function adventures
- Masking
- Zeros and Sparcity

PATCHES

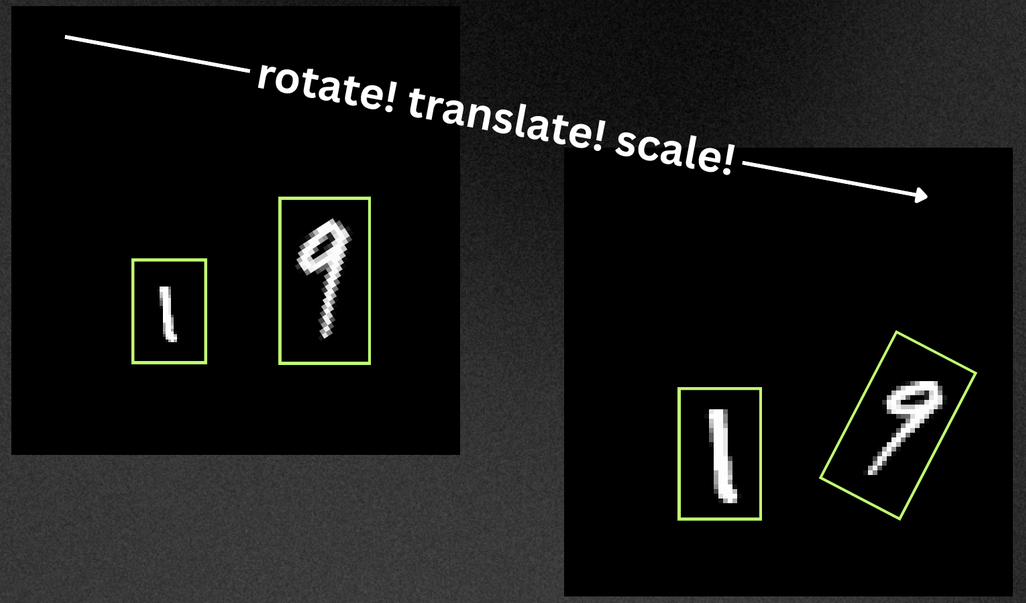


smaller patches → effeciency



smaller patches → accuracy

AUGMENTATION



LOSS ADVENTURES PT. 1

Token:

[<sos>, x, x, <eos>, <pad>, <pad>]

Target: **[10, 0, 0, 11, 12, 12]**

Predicted: **[10, 1, 0, 11, 11, 12]**

Not ignoring <pad>, no custom class weighting	**Ignoring <pad>, no custom class weighting**	**Ignoring <pad>, yes custom class weighting**
Penalized for predicting <pad> tokens → **distraction from digits** → **misleading loss**	9 digits and <eos> treated equally, except <pad>	• Class 9 gets a 2x loss penalty • <eos> and <pad> get 0.5x penalty → **did not learn when to stop**
🙃	🥳	🤔

LOSS ADVENTURES PT. 2

We had some unfair loss where: **731 =/= 137**

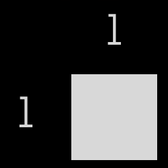
I thought any loss function you could write would work, however it's anything that pytorch detects as differentiable, so some attempts at custom loss failed

I converted the logits back into a probability distribution with softmax to lose the concept of sequence

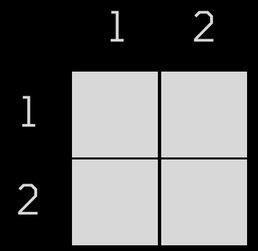
It worked but wasnt any sort of obvious improvement (and may cause issues with Start and End tokens)- But could be useful elsewhere.

HOW?

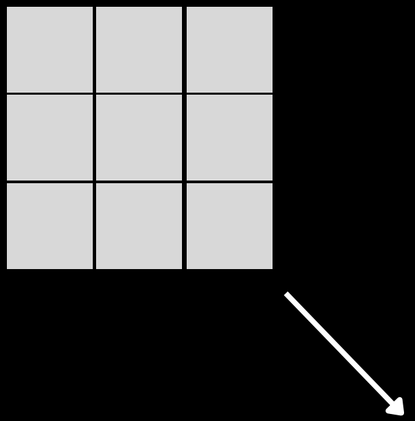
Past X Past looks like a square



Token 1 can attend to token 1

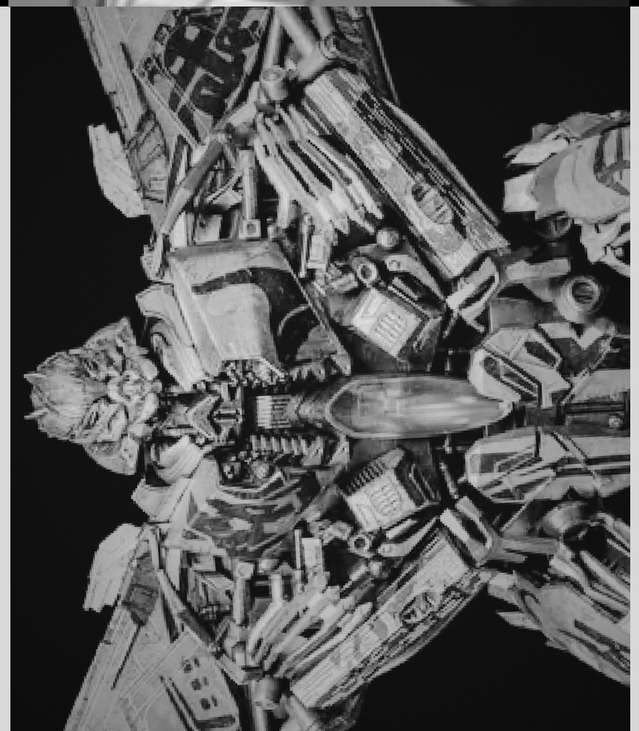


Token 1 and 2 can attend to token 1 and 2

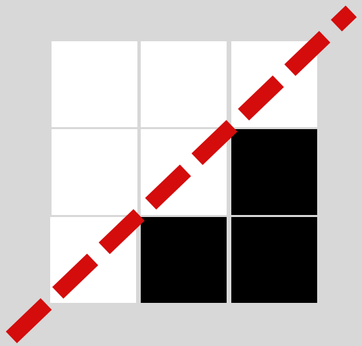


Wait this aint a triangle?

TRIANGLE



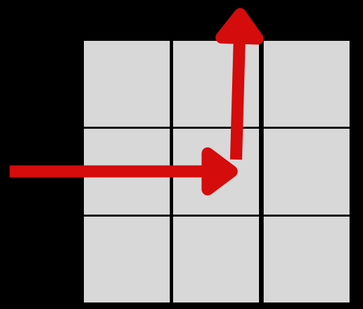
Or if you draw a line here...

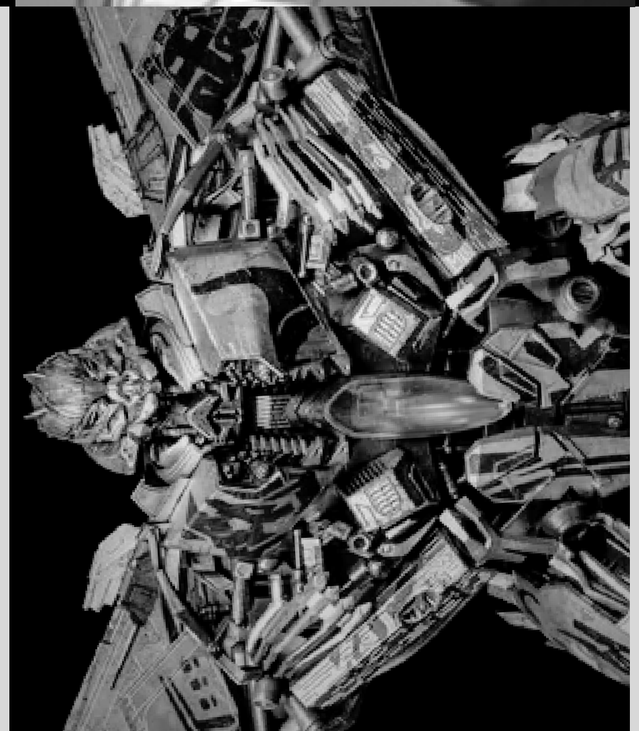


...You get a triangle, that sweeps right, y no sweep?

TRIANGLE HOW?

It's only valid to read the triangle matrix left then up.





It's actually this

Console has small vertical spacing

ZEROS AND SPARSITY

- “zeroing out” matrixes is a natural term but beware if your tensors went from -1 to +1 and actual 0 is 50%
- +inf and -inf if building yourself or be very careful normalizing before/after
- These elements in Pytorch can accept variable lengths:
  - nn.Transformer
  - nn.TransformerEncoder/ nn.TransformerDecoder
  - nn.MultiheadAttention
  - nn.Embedding
  - torch.nn.utils.rnn.pack_padded_sequence / pad_packed_sequence

ZEROS AND SPARSITY

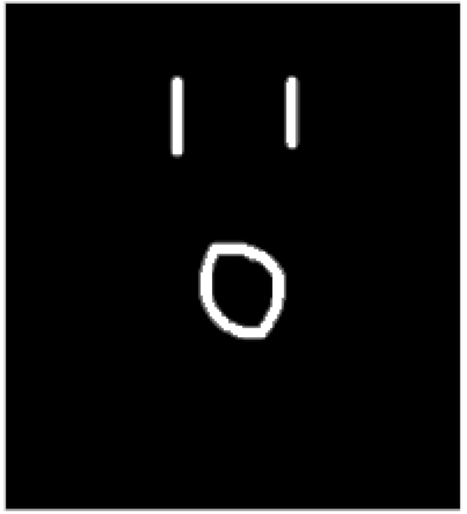
- Under the hood, variable length seems to be just fixed length with padding
- You dont get to save compute by making out stuff, unless you have a “Architecturally sparse” models
- A tensor with 1000000 inputs, all but one masked, would still compute things for the remaining 999999

- In models like MLP 0 or -inf is as valid a input as any other and is processed with everything else, potentially influencing things with connected layers.

Digit Transformer ✨

A mini transformer model for recognizing sequences of handwritten digits.

🖍️ Draw a multi-digit image



Classify

🔍 Prediction

Raw tokens: [1, 0, 1, 11, 12, 12]

Digits: 1 0 1

Digit Transformer ✨

A mini transformer model for recognizing sequences of handwritten digits.

ion

[1, 0, 1, 11, 12, 12]

: 1 0 1