I. Intro

mapping → encoding → decoding

Google uses sentence piece (encoder)
- sub-word units

GPT uses tiktoken

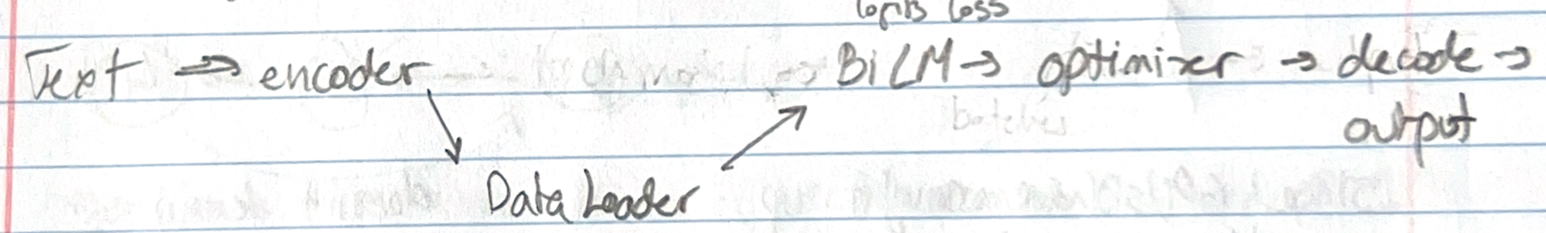
batch dimension (mini-batches)

logits = scores

loss = F.cross_entropy(logits, targets)

Bigram language Model ★ - look at the previous character

II. Diagram



III. Cool Math

Data should not be influenced by future, just past.

inefficient: take the rolling averages of tensor using torch.mean()

efficient: torch.tril() → 1 1 1 / 1 1 1 / 1 1 1 → 1 0 0 / 1 1 0 / 1 1 1

normalized(torch.tril(a)) @ b

↑ weighted sums

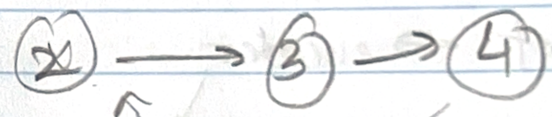
more efficient: use softmax

softmax is a normalization technique → turns attention scores to probabilities

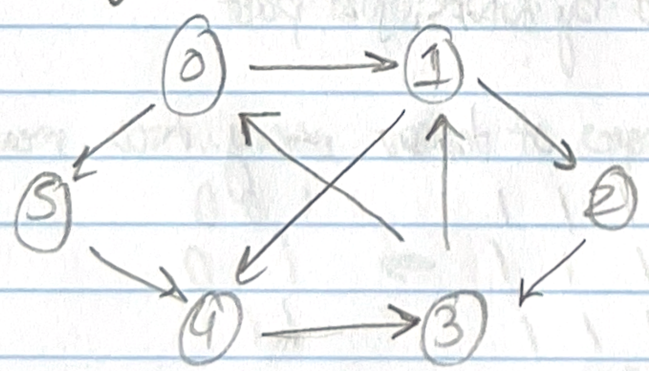
token → query (What am I looking for)
token → key (What do I contain)
• (dot product) = learn a lot_ "attention scores"

(X)

Transformers	vs	RNN
• full context		• sequential
• all tokens processed at the same time		• fuzzy and noisy • vanishing gradients
• works best w/ large datasets		• can only look at values thru hidden state propagation
• more memory		



since GPUs have caught up, now they use this



block size = 8

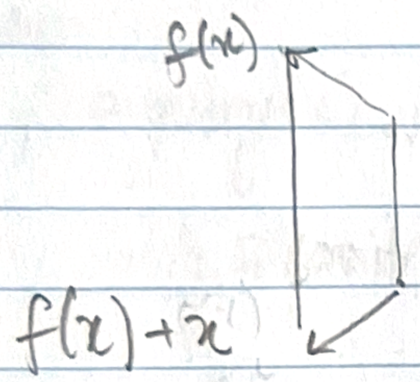
Batch dimensions don't talk to each other

cross attention - if you got info from another set of tokens

queries come from x but keys and vals come from other source

Feed Forward: Input → Hidden layers → Output
For transformers: applied to each token, and helps model interpret what attention gave it

Residual connections



LayerNorm: normalizes values of vector, mean of 0, std of 1

Attention → Feed Forward → LayerNorm

Dropout = reduces generality, drops neural net links

GeLU

pretraining: decoder

GPT training

train w/ supervised → collect comparisons and train reward →
→ optimize using PPO

Proximal Policy Optimization - slow and steady reinforcement

transformer
- Self attention
- feed forward
- LayerNorm + Residuals

I. Input Embeddings + Positional Encoding
Each token is turned into a vector, add positional encodings

II. Self Attention

$$\text{attention} = \text{softmax}\left(\frac{Q \times K^t}{\sqrt{d_k}}\right) \times V$$

III. Multihead Attention
Repeat self attention in parallel multiple times
Concatenate all heads → linear projection

IV. Feed Forward (multilayer perceptron)
Before attention, token's vector goes through 2-layer MLP
Transforms info it got

V. LayerNorm + Residuals
Before attention/MLP, LayerNorm normalizes vector
Adds input back to output (Residual Connection)
Prevents gradient problems

VI. Linear → Softmax to predict next token

Ex 1. G

Goal mu

Old

for head

ou

o

final =

New

q' =

i. Q =

K =

V =

ii. Re

iii.

s

EX1. Combine Head and Multihead attention into one class that process all the heads in parallel

Goal multihead operation becomes batch matrix operation

Old

```
for head in heads:
    out = head(x)
    outputs.append(out)
final = torch.cat(outputs, dim = -1)
```

⇓

New

```
q = query(x).view(B, T, n_head, head_size).transpose(1,2)
        # → (B, n_head, T, head_size)
```

i. $Q = x W^Q$
$K = x W^K$ → All shape $\mathbb{R}^{B \times T \times C}$
$V = x W^V$

ii. Reshape / Transform grp up by head

$Q \to \mathbb{R}^{B \times n_{head} \times T \times d_k}$

iii.

$$scores = \frac{QK^T}{\sqrt{d_k}} \in \mathbb{R}^{B \times n_{head} \times T \times T}$$

$output = softmax(scores) \cdot V$ ← value

iv.

if, W^Q, W^K, $W^V \in \mathbb{R}^{c \times c}$

$$\text{Attention}(x) = \text{Concat}_{h=1}^{n}\left(\text{softmax}\left(\frac{xW_h^Q \times (xW_h^K)^T}{\sqrt{d^k}} \cdot xW_h^V\right)\right)$$

RoPe

positional encodings tell where the word is

i. Absolute pos enc = e_final (pos) = e_token + e_position
issues: given short, irrelevant training mods not good.
don't need to know the 39th word if you are 2 words after despite

ii. Relative pos enc

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T + \text{Relative Pos Matrix}}{\sqrt{d}}\right)V$$

Too many relationships, very slow, Key-value cache, not commonly used

iii RoPE!
- encodes positional info so it allows the model understand absolute and relative pos enc. does a rotational mechanism

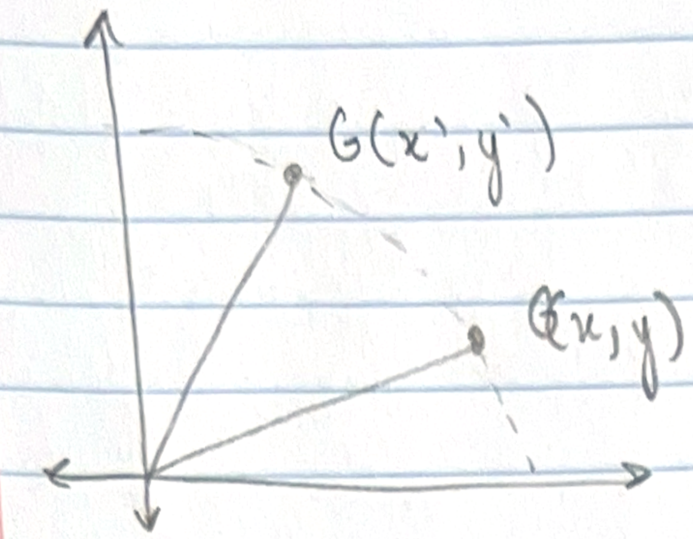
The quick brown fox jumps

constant

→

more rotations

Lower dimensions rotate faster higher dim rotate slower



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

RoPE can handle sequences longer than it was trained on

Easy to cache next word

$$f_{\{q,k\}}(x_m, m) = R^d_{\theta,m} W_{\{q,k\}} x_m$$

$\Rightarrow$ n-dimension corkscrew visualization