Attention Is All You Need

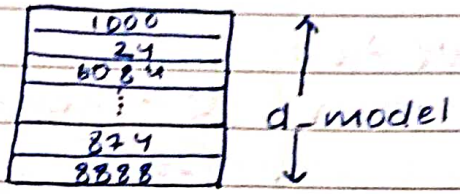
↓

Transformer

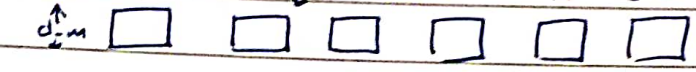
I. ARCHITECTURE ⇒

1. Input Embeddings ⇒

- cat → 1234 → Unique Input Id



- Whole text → seq_len × d_model

↳  (seq_len = 6)

- Code →
  - IE Layer = nn.Embedding(vocab_size, d_model)
  - Fwd fn → IE(x) * $\sqrt{d\_model}$ [1]

2. Positional Encoding:

- Fn to apply it, PE [2] = $\begin{cases} \sin\left(\frac{Pos}{10000^{\frac{2i}{d\_model}}}\right), & i \to even \\ \cos\left(\frac{Pos}{10000^{\frac{2i}{d\_model}}}\right), & i \to odd \end{cases}$
- Input token → PE → Input token + Positional Info

- Code →
  - pe = torch.zeros(seq_len, d_model) → Dummy tensor initially
  - pos = torch.arange(0, seq_len).unsqueeze(1) ⇒ [0, 1, 2 ... SL]
  - pe[even columns] = PE(pos) & pe[o.c.] = PE(pos)
  - * register_buffer(pe)
  - Fwd fn → x = x + pe → return dropout(x) [3]

[1], [2], [3] ⇒ According to Paper

3. Layer Normalisation:

- α = torch.ones(features) , β = torch.zeros(f)
- $\epsilon = 10^{-9}$ $\Rightarrow \alpha, \beta^{[4]}$ Learnable nn.Parameters()
- Fwd fn $\Rightarrow$

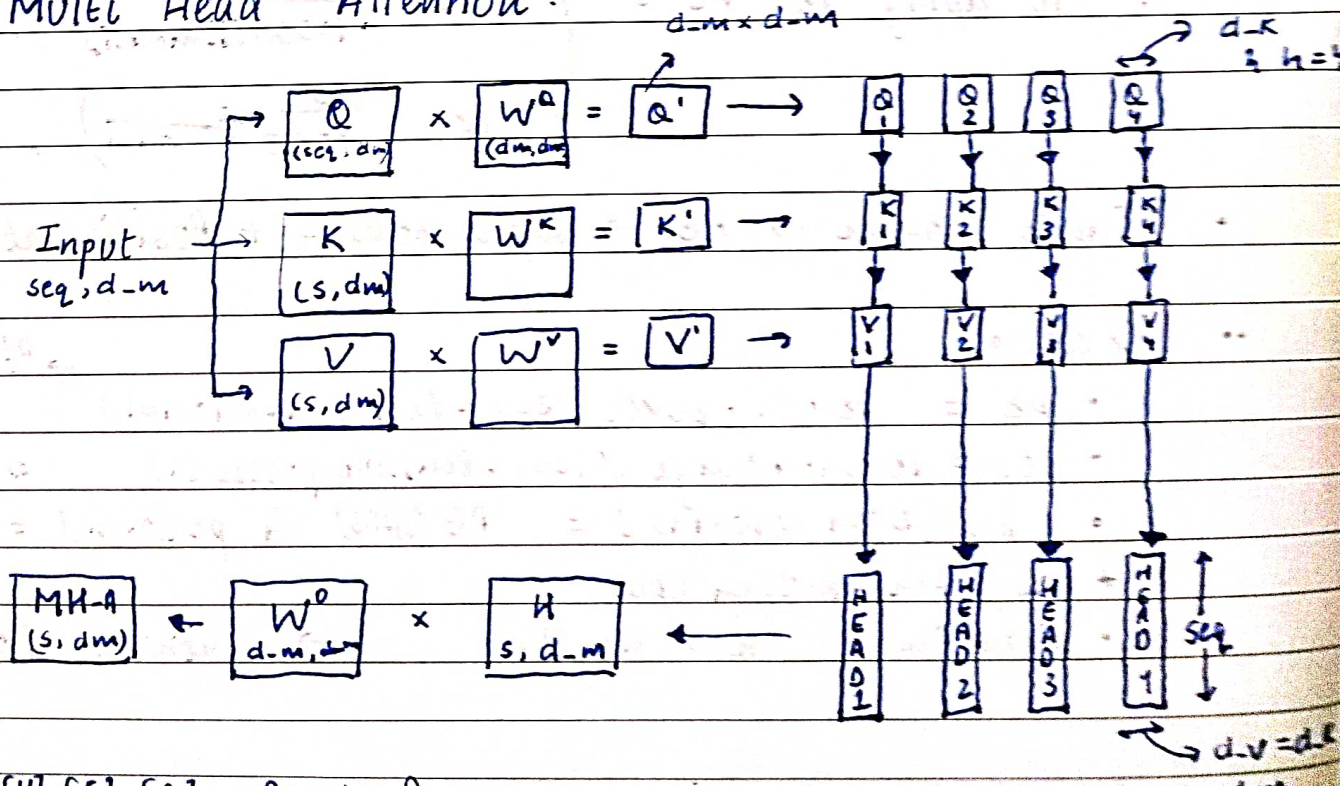
$$x = \alpha \left(\frac{x_i - \text{mean}_{(x_i)}}{\sqrt{\text{std dev}_{(x_i)} + \epsilon}} \right) + \beta^{[5]}$$

$\Rightarrow$ This mean is wrt dim = -1.

4. Feed Forward Network:

- Simple NN with dropout to train weights & biases.
- $FFN(x) = \max(0, xW_1 + b_1) \cdot W_2 + b_2$ [6]
- Code → FFN_layers = nn.Seq (Linear(d_m, d_ff), ReLU, dropout, Linear(d_ff, d_mode

5. Multi Head Attention:



[4], [5], [6] → Acc to Paper

d_model = d_m => $d_m // h = d_k = d_v$

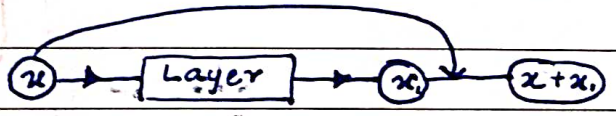
- Attention $(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$ [7]

- Code →
  - init => $d_k = d_v = d_m // h$
    $W_q, W_k, W_v, W_o = nn.Linear(d_m, d_m)$

  * attention => attn_scores = (query * key^T) / $\sqrt{d_k}$
    attn_scores . masked_fill_ (0, 10^{-9})
    return softmax(attn_sc) * Value

  * forward => (q, k, v, mask) =>
    query = W_q(q), key, value similarly (Batch, SL, d_
    query = query.reshape(q.shape[0], [1], h, d_k).transpose(1,2
    # q : d_m → h * d_k , Final shape → (B, h, SL, d_k)
    !!! key & value ...
    x = attention(query, key, value, mask)
    (B, SL, d_m) ← x = x.transpose(1,2).contiguous().view(x.shape[0], -1, d_
    return W_o(x)

6. Residual Connection :



- Code => return x + (dropout (sublayer(LayerNorm(x)
  ↓
  Layer being applied

[7] → ATP

7. Encoder:

* Encoder Block =>

```
init => ( features, attention_block, ffn, dropout) =>
    self_attn_block = attn_block , feed_fwd_net = ffn
    residual_conns = ModuleList ( RN (2))
```

[8]

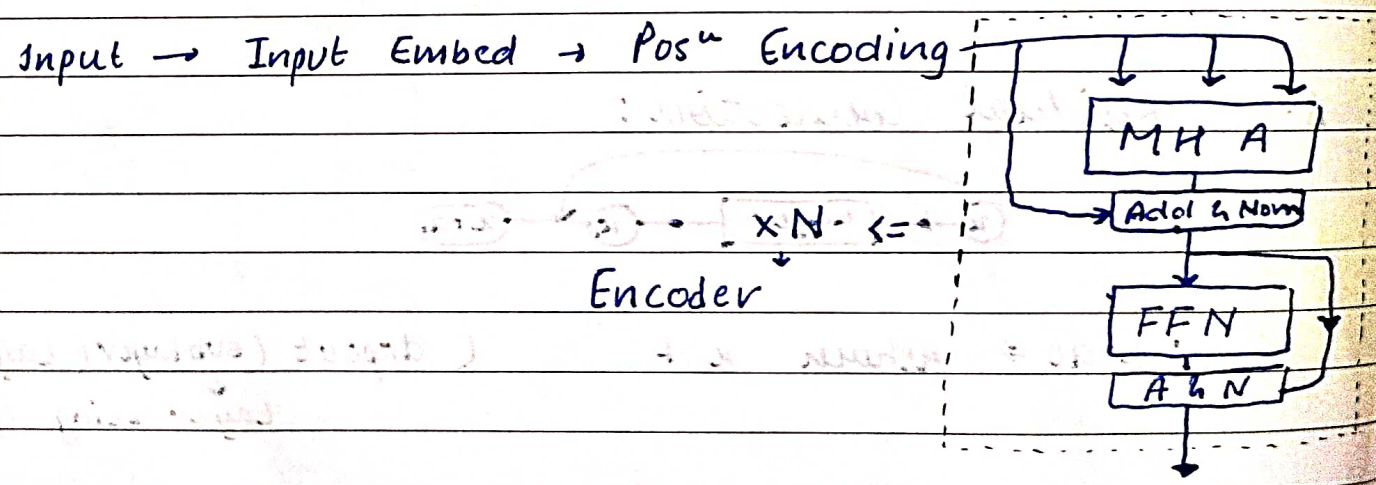
```
fwd => ( x, src_mask)                    q, k, v
    attn_output = atten_block (x, x, x, src_mask)
    x = residual_conns [0] (x, attn_output)
    x = residual_conns [1] (x, ffn)
    return x
```

* Encoder =>

```
init => layers_in_encoder , LayerNorm

fwd => for layer in l_i_e => , x = layer (x, mask)
       return LN(x)
```

Encoder Block Diagram =>



[8], diagram → ATP

8. Decoder :

* Decoder Block =>

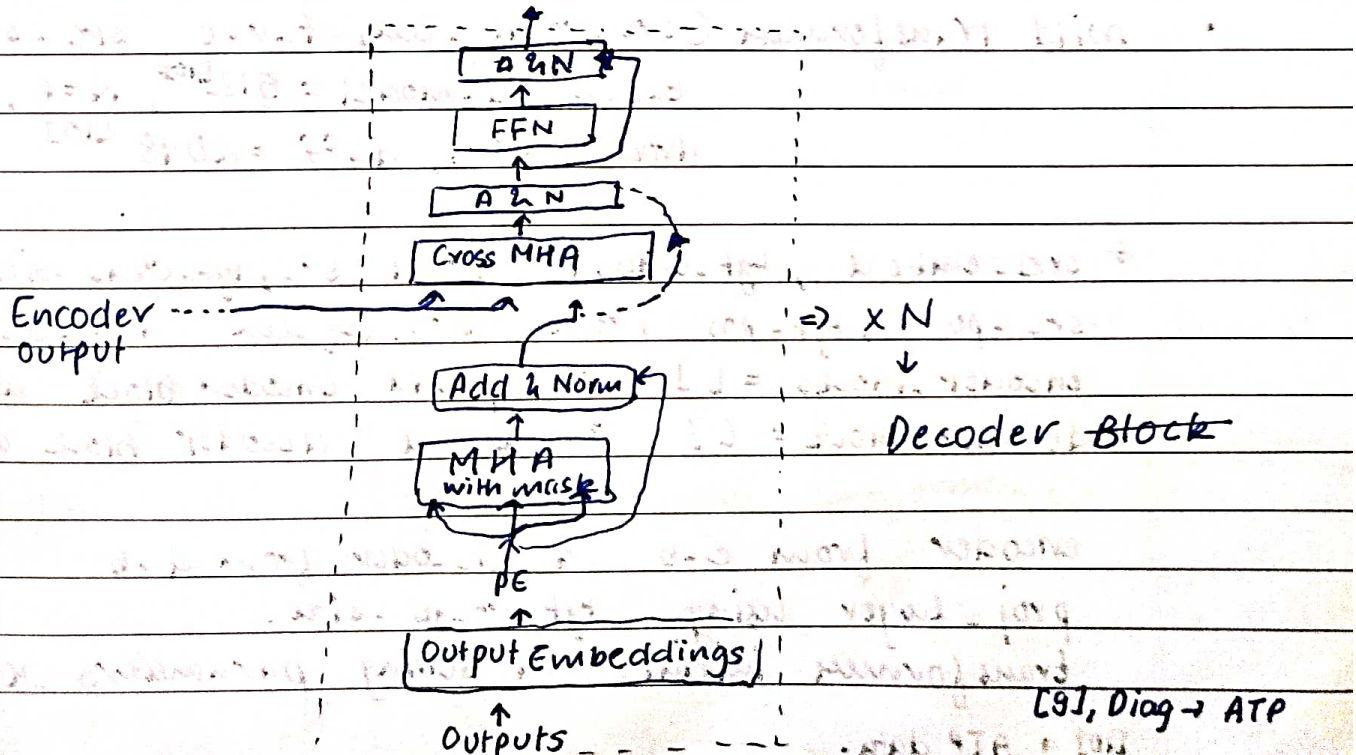
```
init(self_attn_block, cross_attn_bl, ffn, dropout) =>
    sab, cab, ffn initialisn
    res_conns = ModuleList(ResCon(3))   [9]

fwd(x, encoder_output, tgt_mask, src_mask):
    x = res_con[0](x, attn output)
    x = res_con[1](x, cross_attn output(x, eo, eo, mask))
    x = res_con[2](x, ffn)
    return x
```

* Decoder =>

```
init → layers, layerNorm
fwd => for layer in layers => x = layer(x, eo, src_mask, t_m)
       return self.norm(x)
```

Decoder Block Diagram =>



[9], Diag → ATP

9. Projection Layer =>

- init => proj = nn.Linear (d_m, vocab_size)
- fwd => return log_softmax (proj(x), dim=-1)

10. TRANSFORMER => Creates structure for Transformer.

- init => encoder, decoder, src_embed, tgt_embed, src_pos, tgt_pos, proj^n layer
- encode => (src, src_mask) =>
  src → Input Emb → Pos^n Enc → Masking & Encoder F^n
- decode (encoder_o/p, src_mask, tgt_mask,
  tgt → IE → PE → Masking src & tgt & Decoder F^n
- project (x) =>
  project (x)

11. BUILDING TRANSFORMER => Initialises Transformer

- build Transformer (src_vocab_len, t_v_l, src_seq_len, t_s_l, d_model = 512 [10], N=6, h=8 dropout = 0.1, d_ff = 2048 [10]

=> src_embed, tgt_embed from src, tgt_vocab_size
src_pos, tgt_pos from src_seq_len & t_s_l
encoder_blocks = [] → append encoder block after init
decoder_block = [] → append decoder block after init

encoder from e_b & decoder from d_b
proj_layer using tgt_vocab_size.
transformer initial^n & setting parameters (xavier)

[10] → ATP data.

II. DATA

12. Bilingual Dataset :

- ↑ Class :
  init (ds, tokenizer_src, t_t, src_lang, t_l, seq_len) =>
  initialise all these ↑ +
  sos_token, eos_token, pad_token using token_to_id

  __len__ → return len (ds)
  __getitem__(idx) →
  src_tgt_pair = ds[idx] → src_text, t_t
  enc_input_token, d_i_t &
  enc & dec pad tokens.
  ↳ encoder_ip = [sos, enc_input, pad_enc]
  lly dec & label.

13. Build Tokenizer, Dataloaders & Transformers =>

- get_all_sentences → yield all items

- get_or_build_tokenizer(config, ds, lang) =>
  check tokenizer else
  tokenizer = sub-word / word ... level

- get_ds (config) =>
  ds load, get src & tgt tokenizer, train_test split
  train_ds, test_ds, max_len_src & max_len_tgt
  building train, test dataloaders

- get_model ( ) ⇒ model = build_transformer (....) [1+]

[1+] ⇒ 11.m's fn.

II. DATA

12. Bilingual Dataset :

- ↑ Class :
  init (ds, tokenizer_src, t_t, src_lang, t_l, seq_len) =>
  initialise all these ↑ +
  sos_token, eos_token, pad_token using token_to_id

  __len__ → return len (ds)
  __getitem__(idx) →
  src_tgt_pair = ds[idx] → src_text, t_t
  enc_input_token, d_i_t &
  enc & dec pad tokens.
  → encoder_ip = [sos, enc_input, pad_enc]
  lllly dec & label.

13. Build Tokenizer, DataLoaders & Transformers =>

- get_all_sentences → yield all items

- get_or_build_tokenizer (config, ds, lang) =>
  Check tokenizer else
  tokenizer = sub-word / word --- level

- get_ds (config) =>
  ds load, get src & tgt tokenizer, train_test split
  train_ds, test_ds, max_len_src & max_len_tgt
  building train, test dataloaders

- get_model ( ) ⇒ model = build_transformer (....) (C++)

(C++) → 11.m's f^n.

III. TRAINING =>

14. Config File =>
- get_config => returns config
- get wt file path => Makes file path for each epoch
- latest_weights_file_path => Finds wts of newest save

15. Training =>
Initialises everything, loss, optim, path, devia.
Loads model if saved model present.
Training loop.
Saving Model

16. Evaluation =>
Evaluates the last saved model.