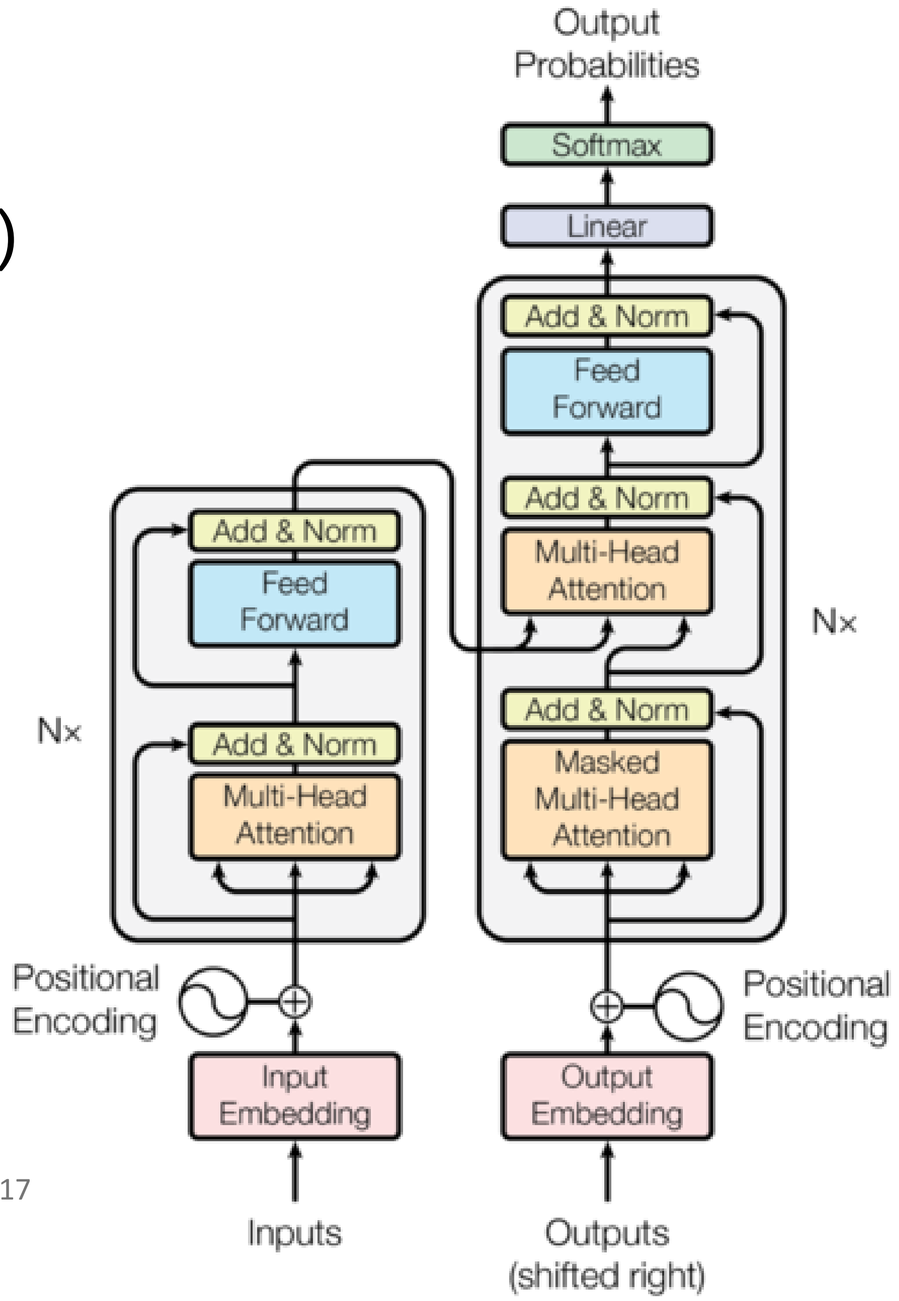
Transformer

Transformer

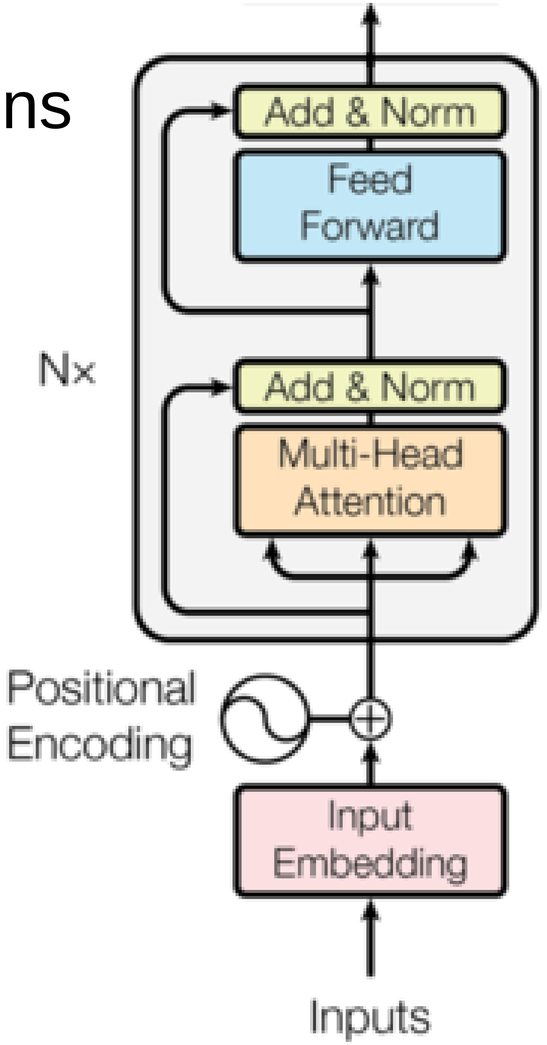
- Attention is all you need (Vaswani et al., 2017)
- An encoder-decoder framework for sequence-to-sequence modeling
- No recurrent units



From “Attention is all you need” paper by Vaswani, et al., 2017

Transformer Encoder

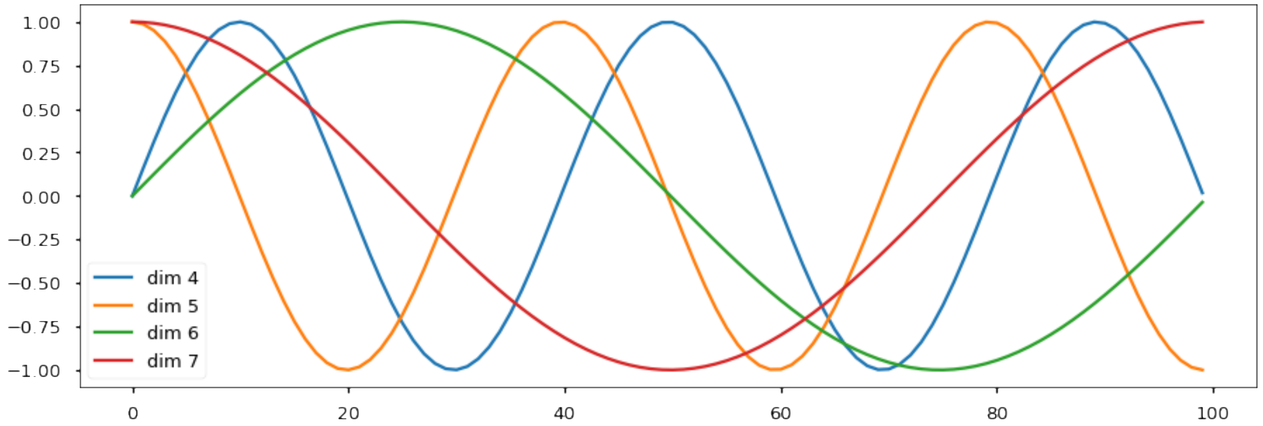
- N layers of Transformer blocks with residual connections
- Parameters in each layer are not shared
- Input tokens are processed in parallel

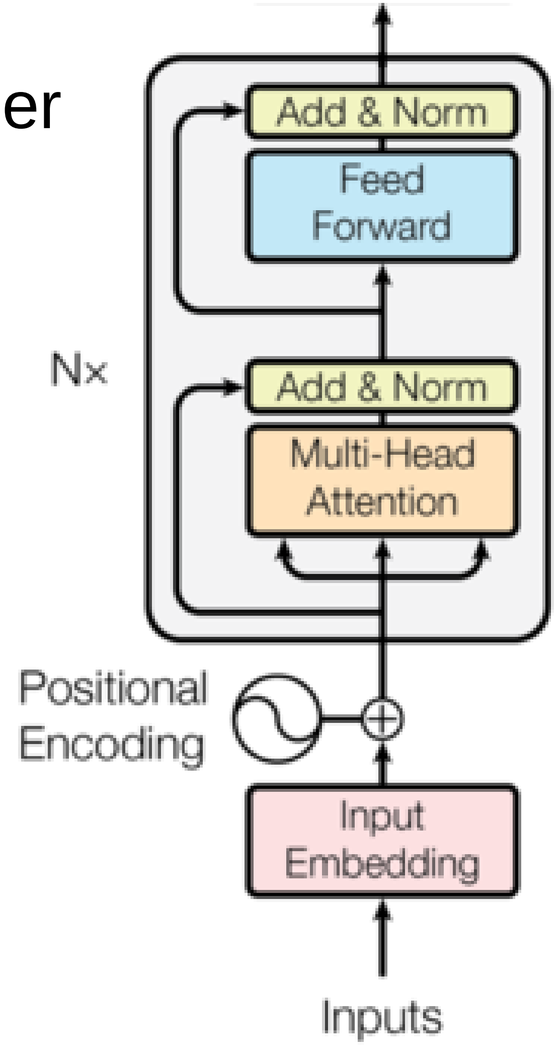


Positional Encoding

- Positional encoding provide the model about word order
- Sinusoidal position encoding

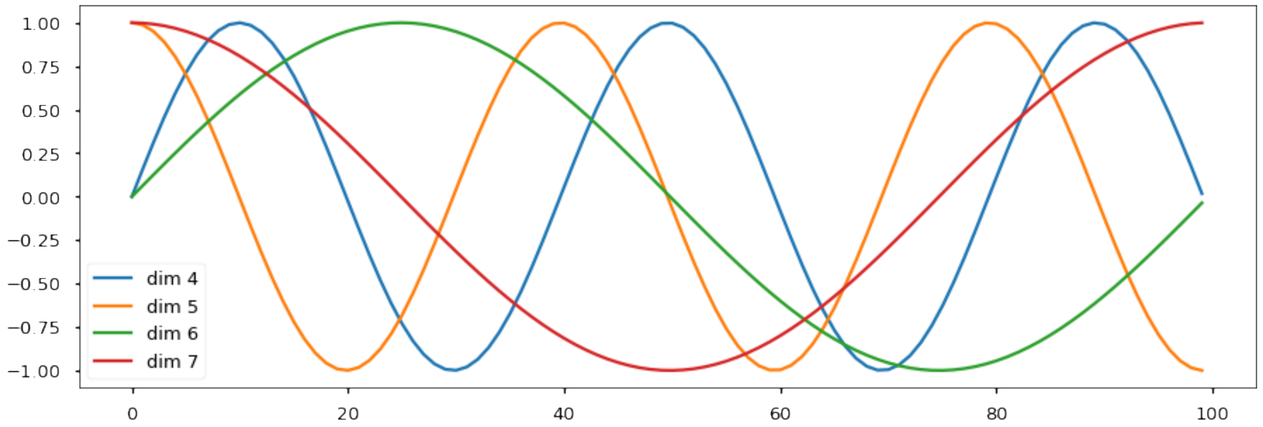
$$\mathbf{x}'_t = \mathbf{W}_{\text{emb}}\left(\mathbf{x}_t\right) + \vec{p}_t$$





Positional Encoding

$$\mathbf{x}'_t = \mathbf{W}_{\text{emb}}\left(\mathbf{x}_t\right) + \vec{p}_t$$

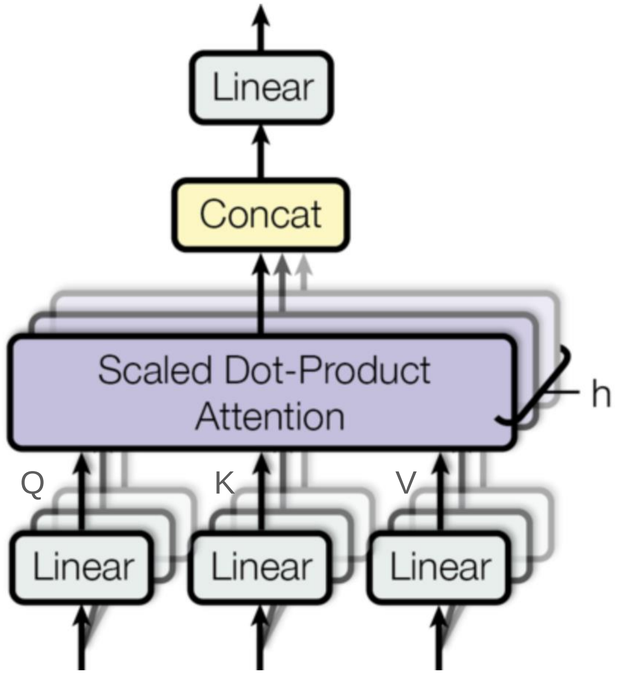


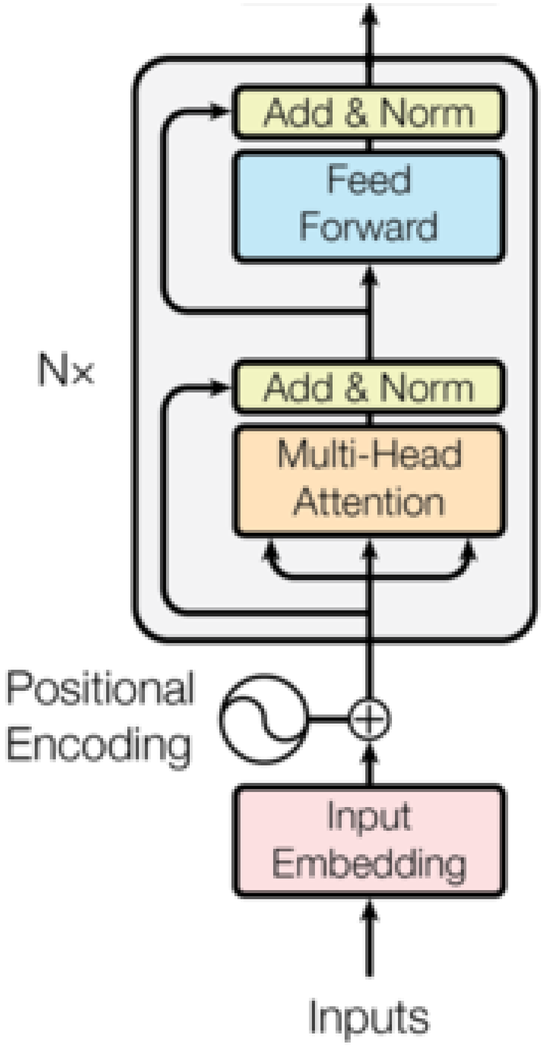
$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$

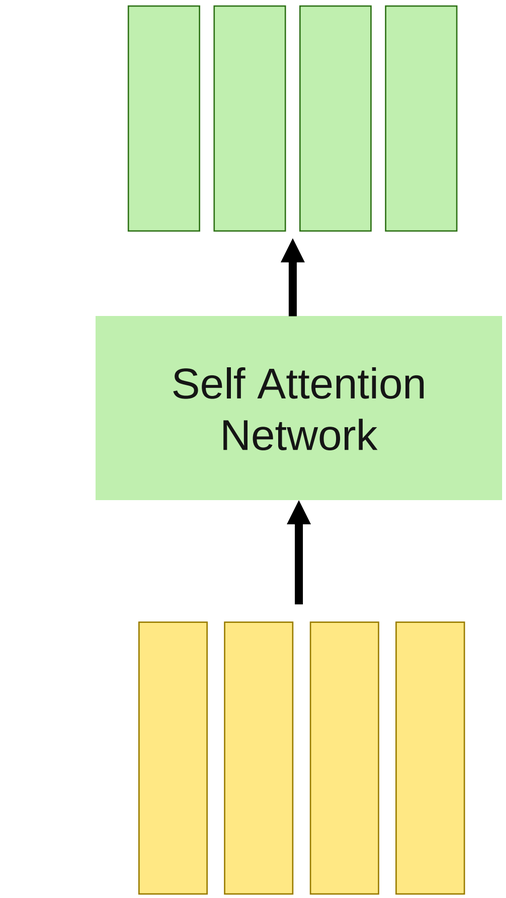
Multi-head Attention

- Multi-head Attention is the concatenation of the outputs from self-attention network (SAN)

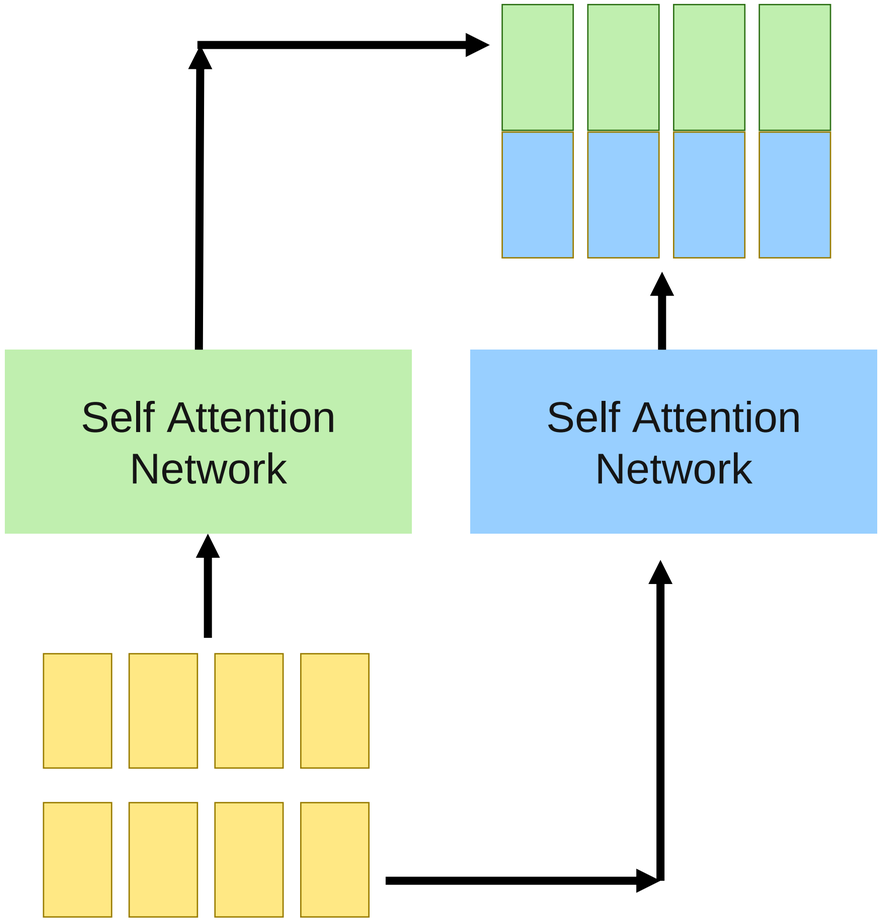




Multi-head Attention



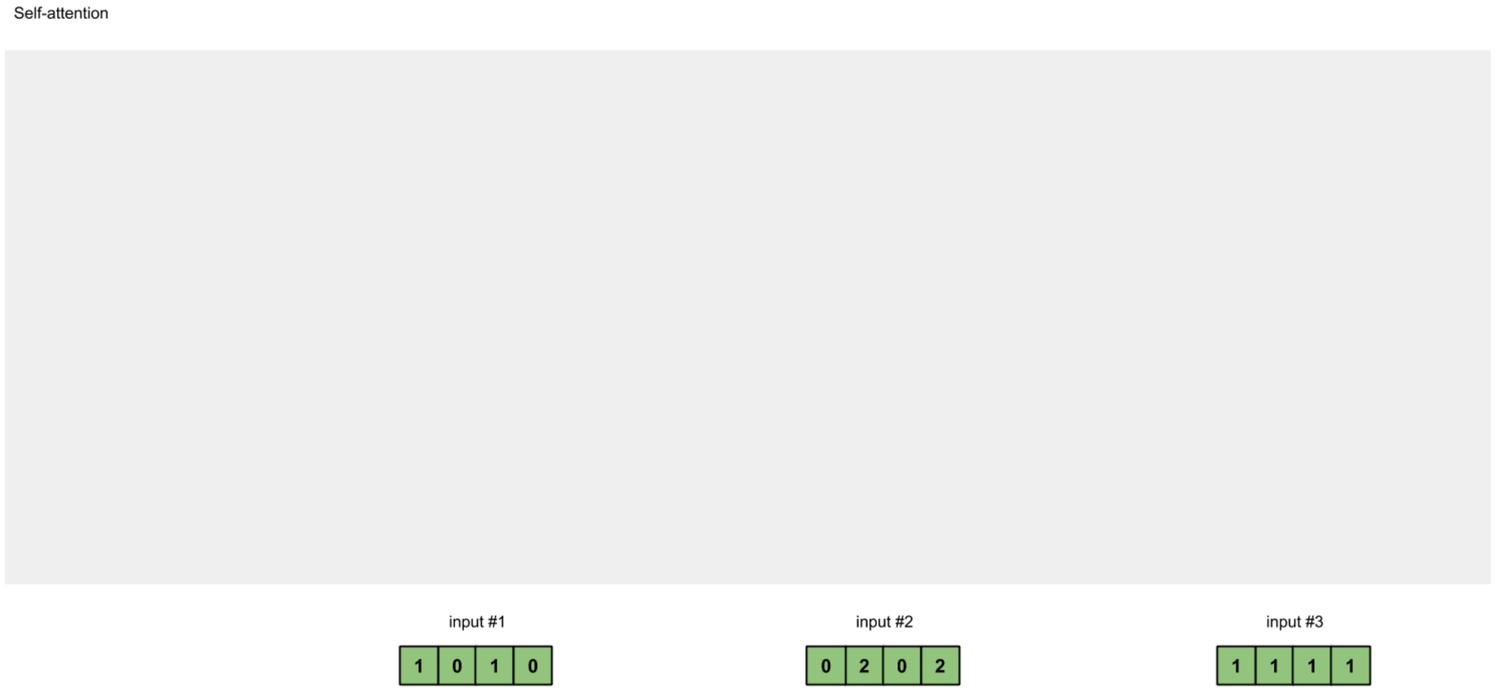
Single head self-attention (m=1)



Multi head self-attention (m=2)

Self-Attention Network

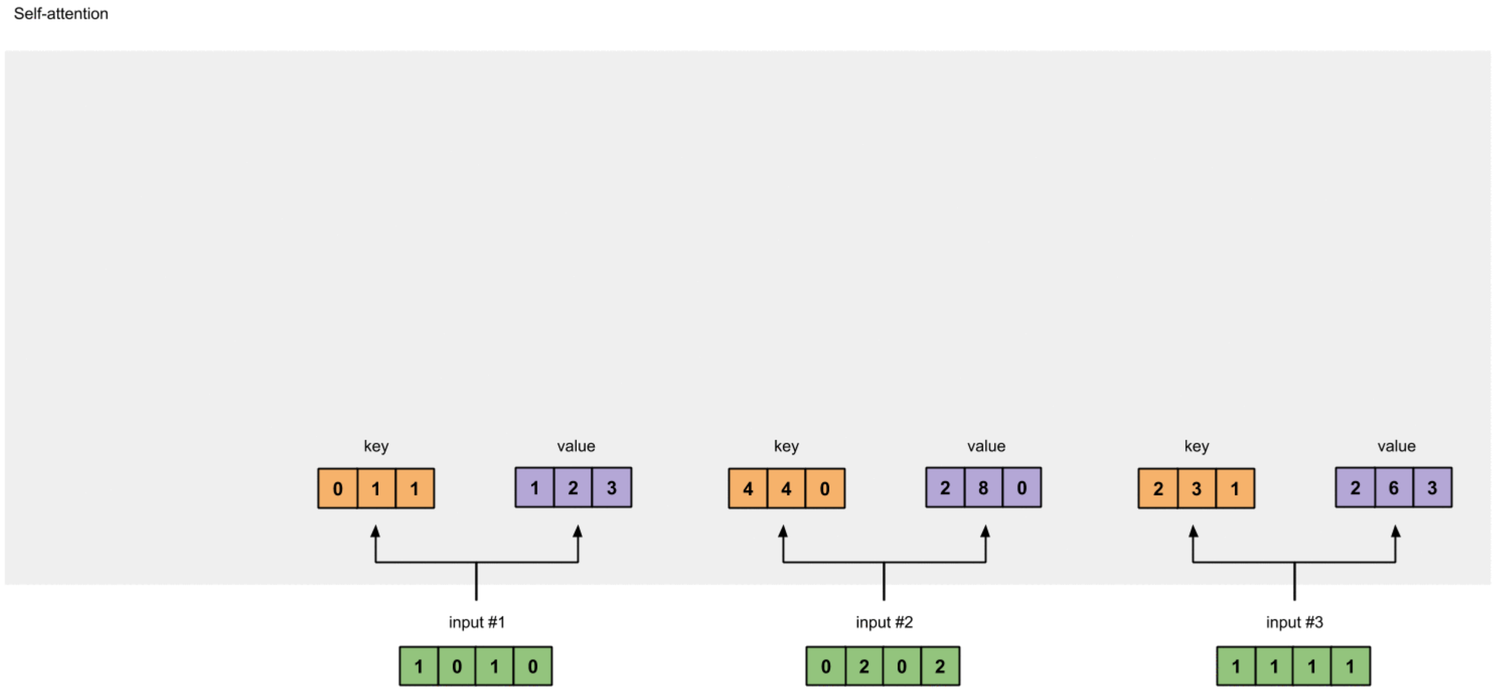
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network

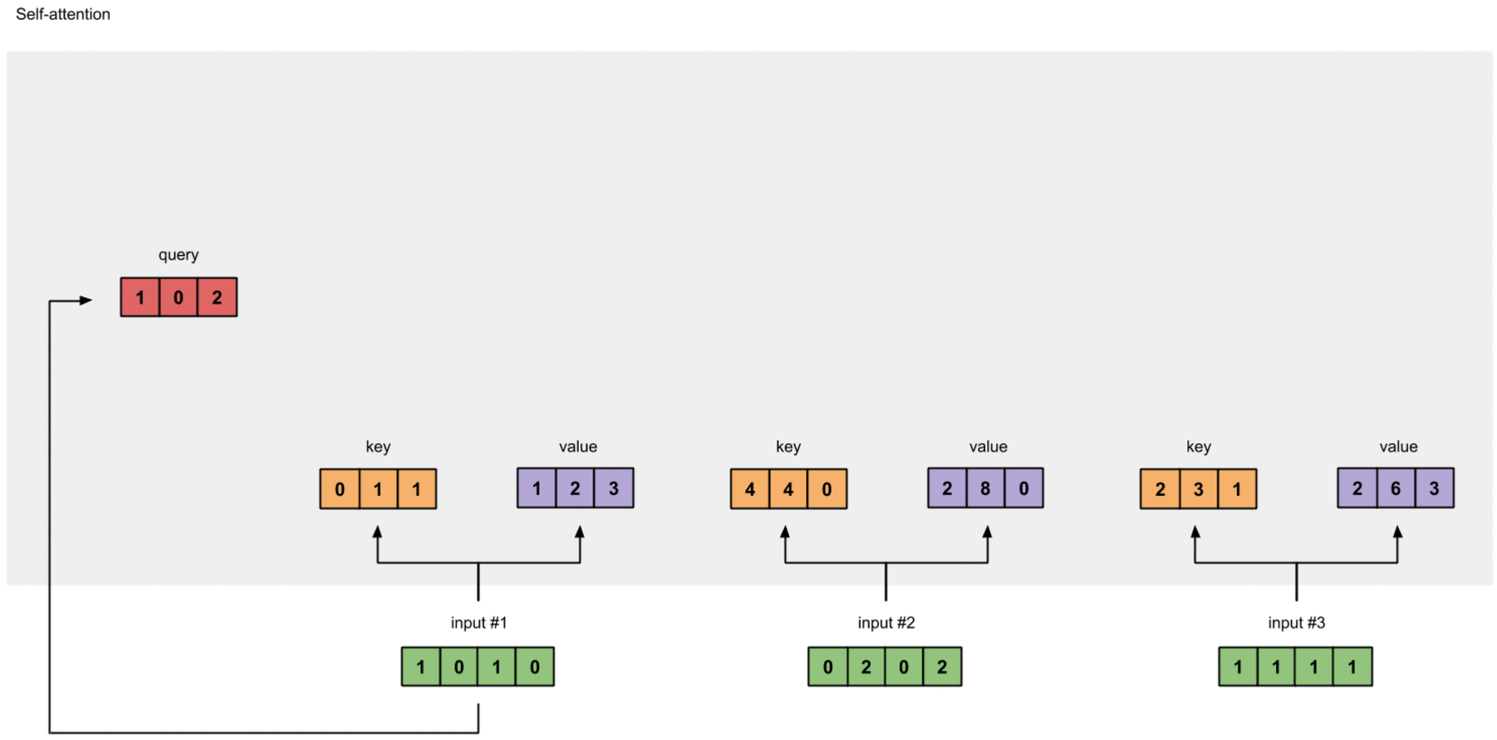
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network

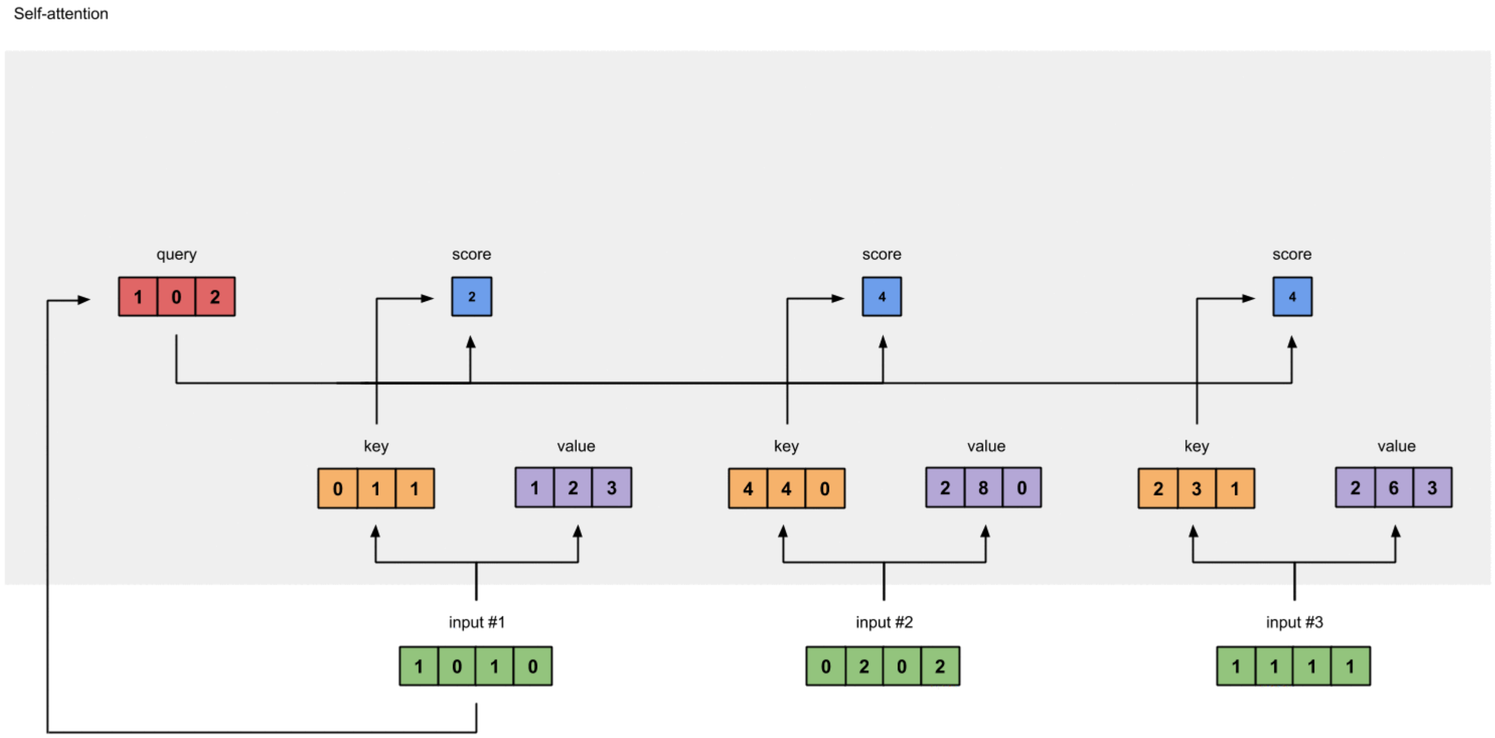
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network

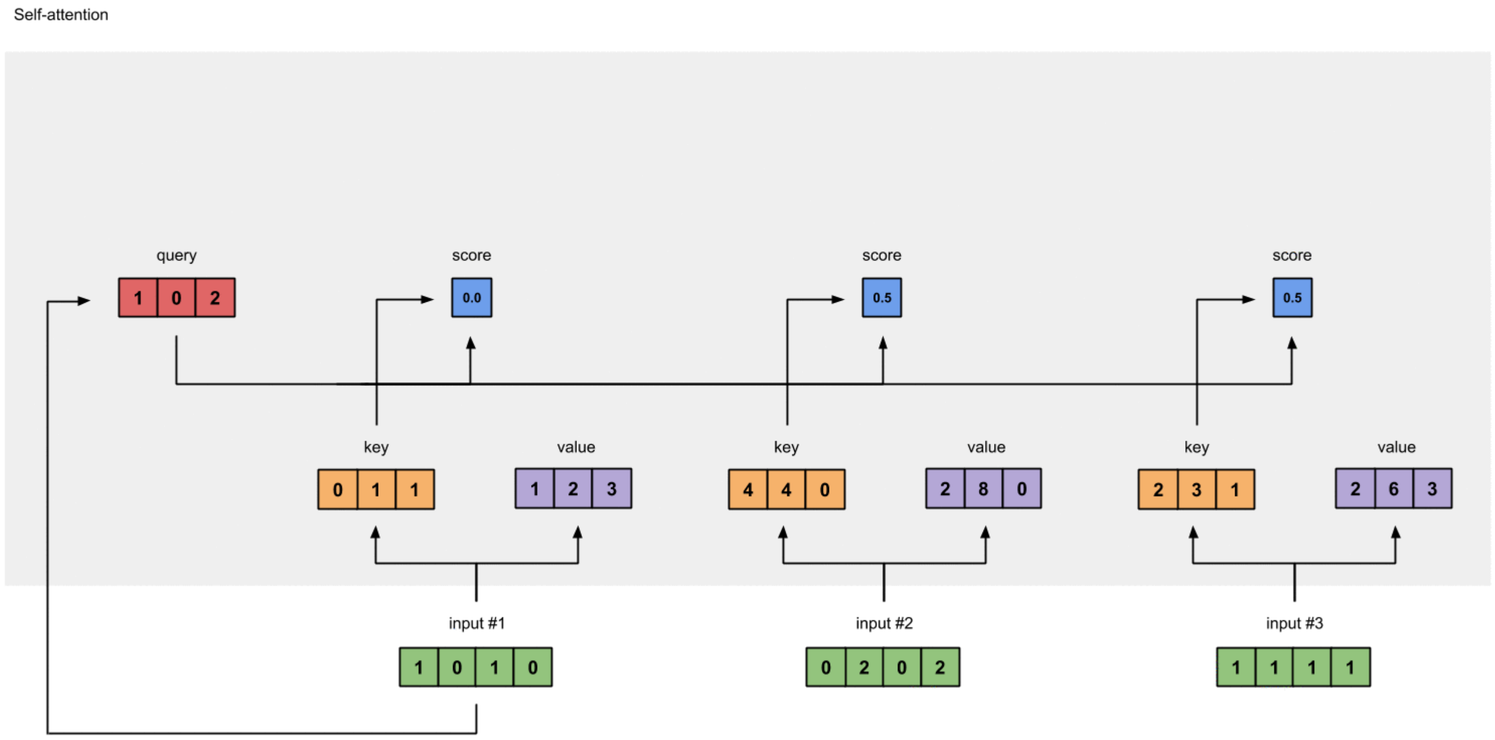
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network

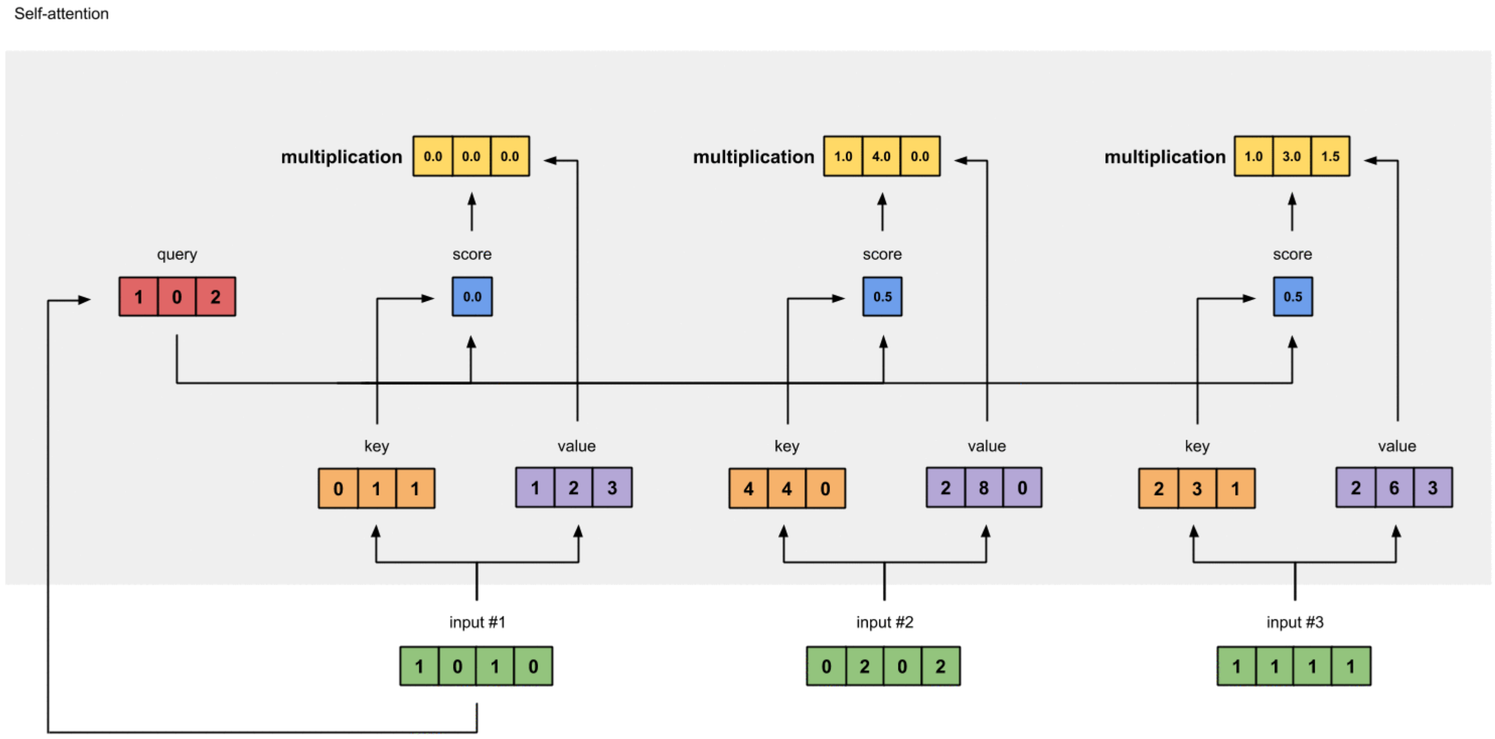
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network

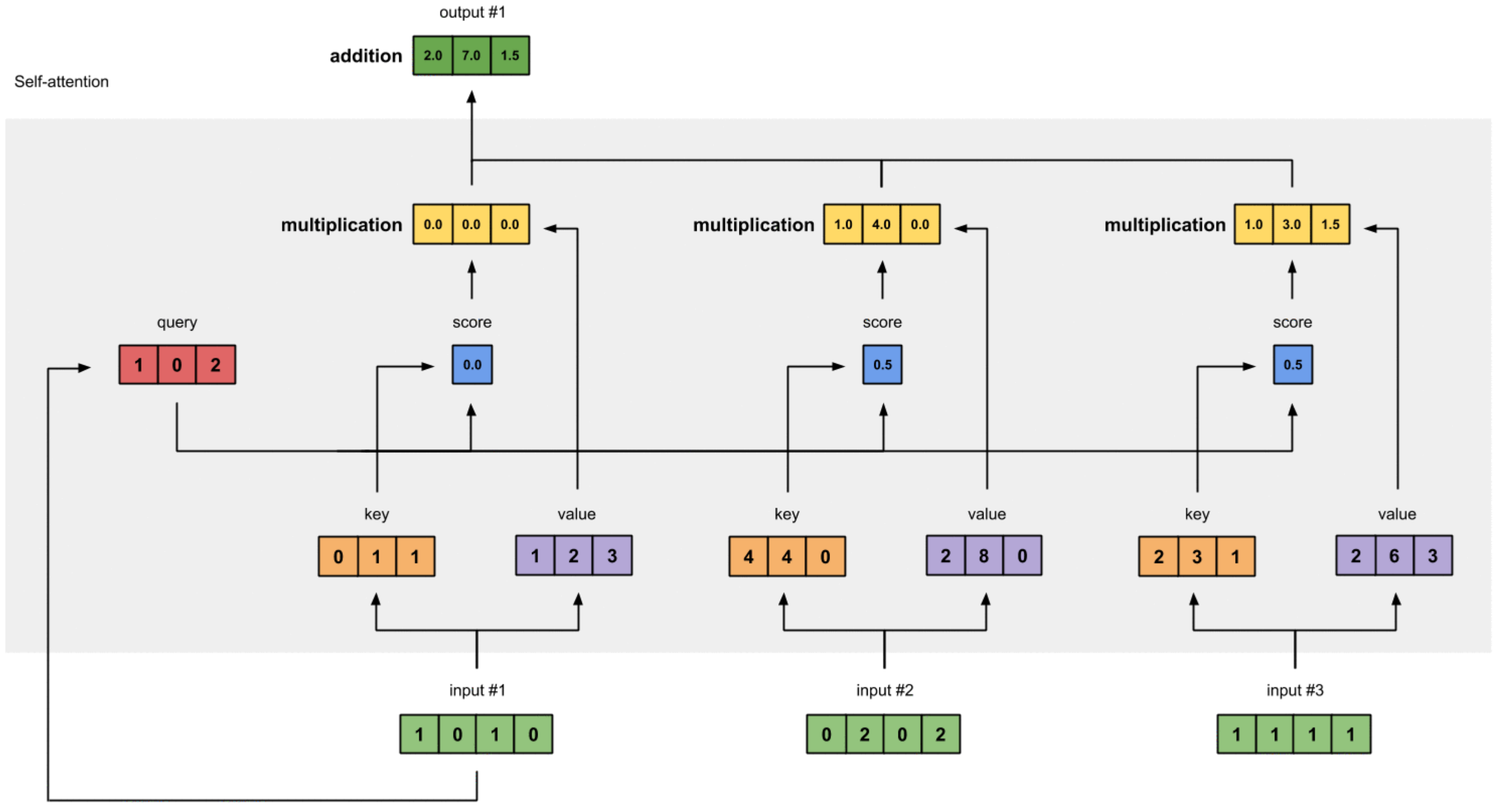
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

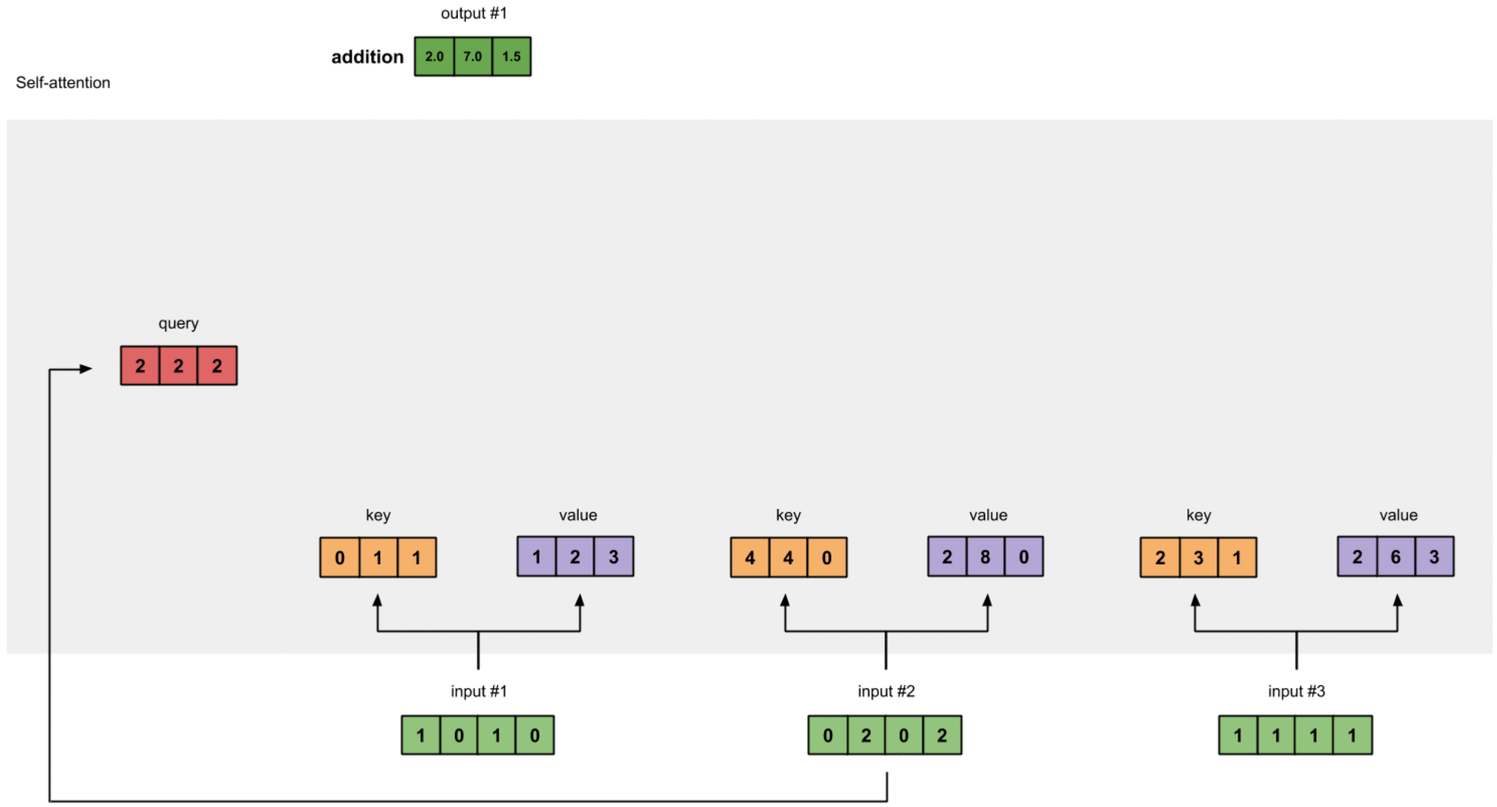
Self-Attention Network

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



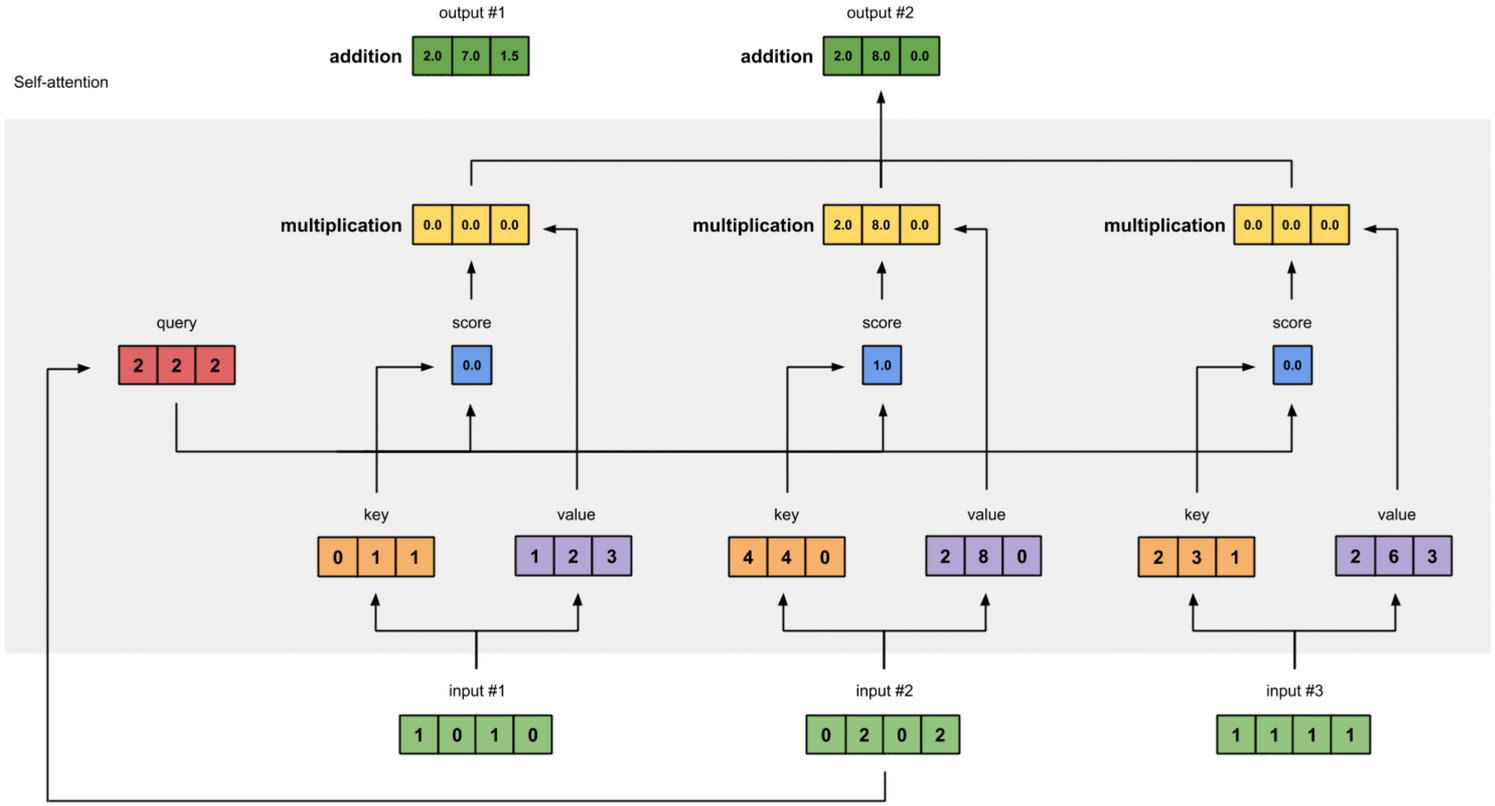
https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network



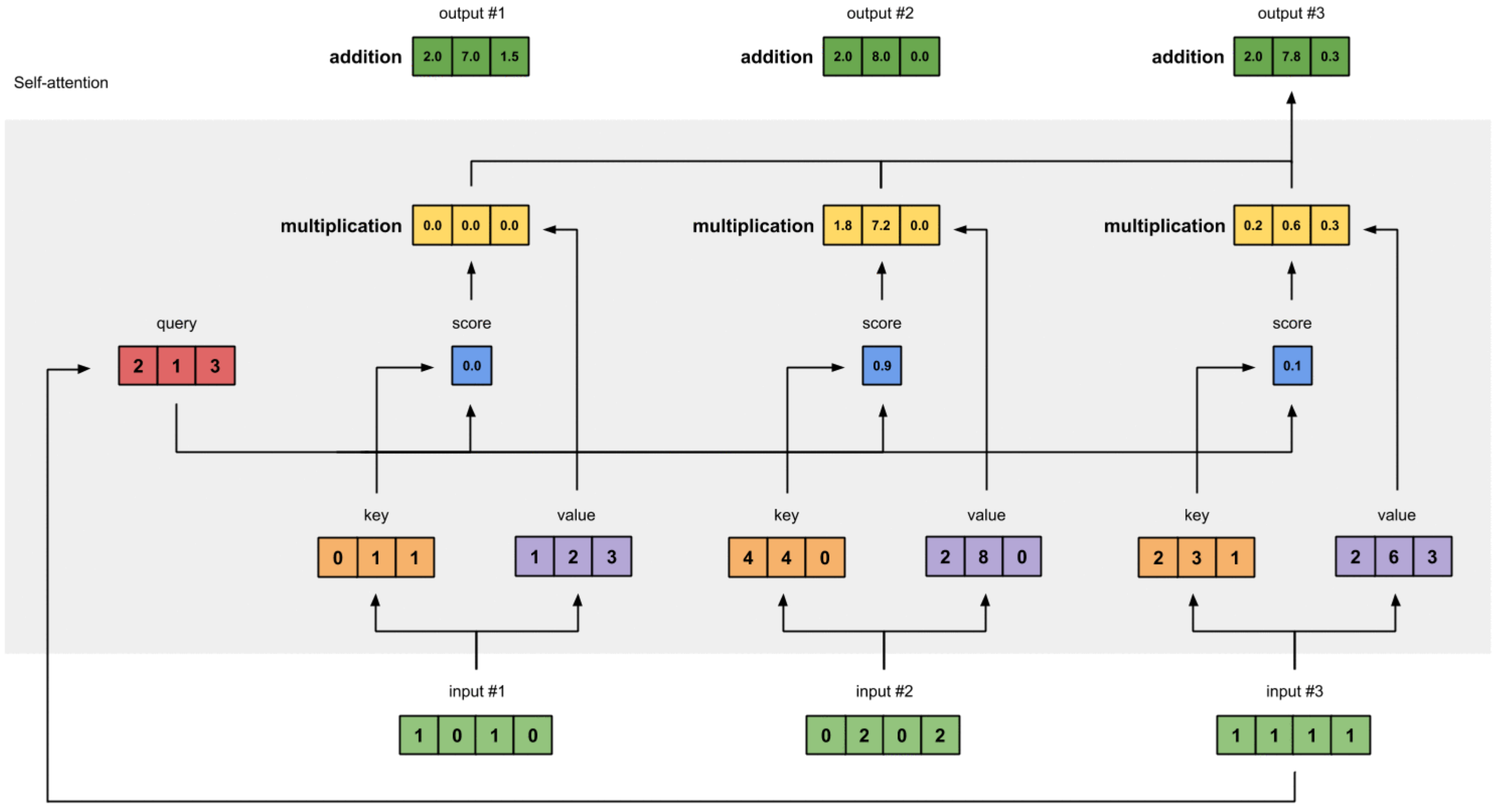
https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network



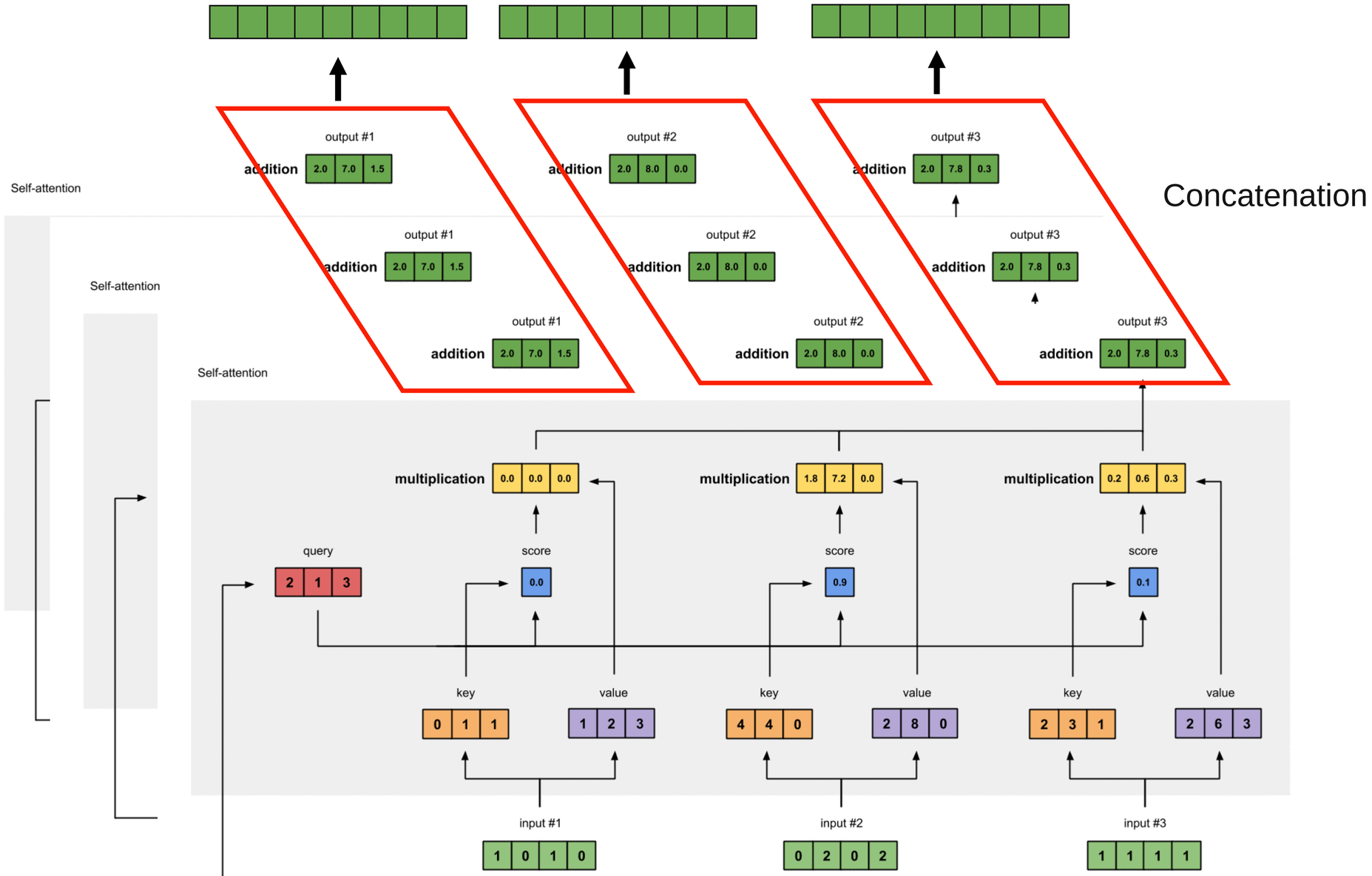
https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network



https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Multi head Self-Attention

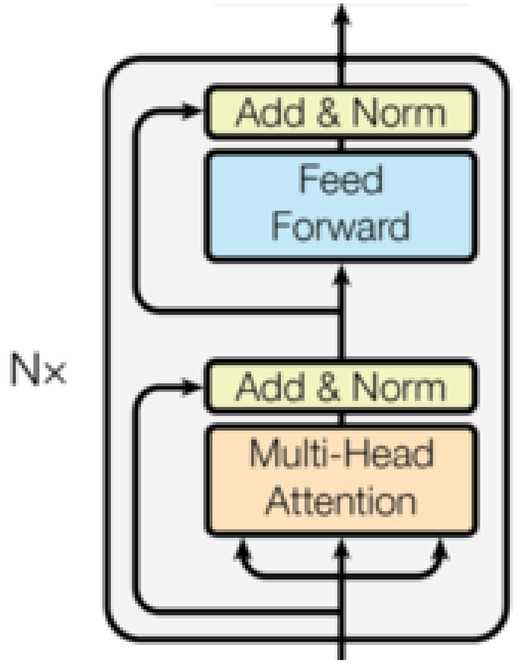


Advantages of self-attention network

- Process in parallel
- Better in modeling long term dependencies, able to freely attend to other position.

Encoder : Add & Norm

- Residual Connection
- Layer Normalization



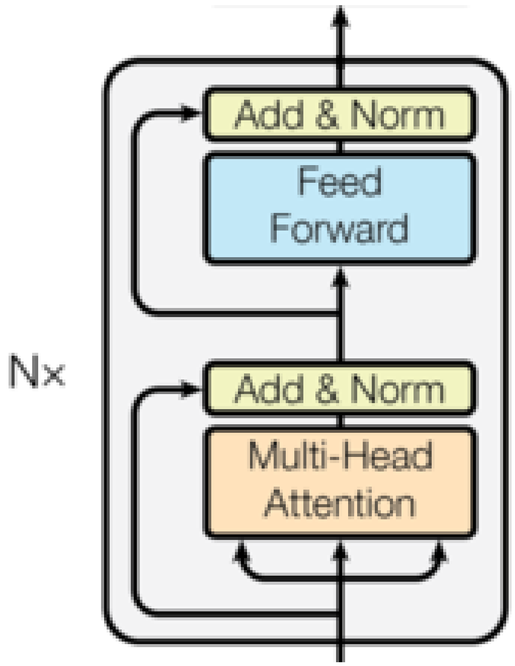
$$x' = \text{LayerNorm}\left(\text{SelfATT}(x) + x\right)$$

Layer Normalization – to normalize mean and variance of inputs (a^l) for a specific layer (l), assume that the layer has H hidden units.

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2} \qquad \bar{a}_i^l = \frac{g_i^l}{\sigma_i^l}\left(a_i^l - \mu_i^l\right)$$

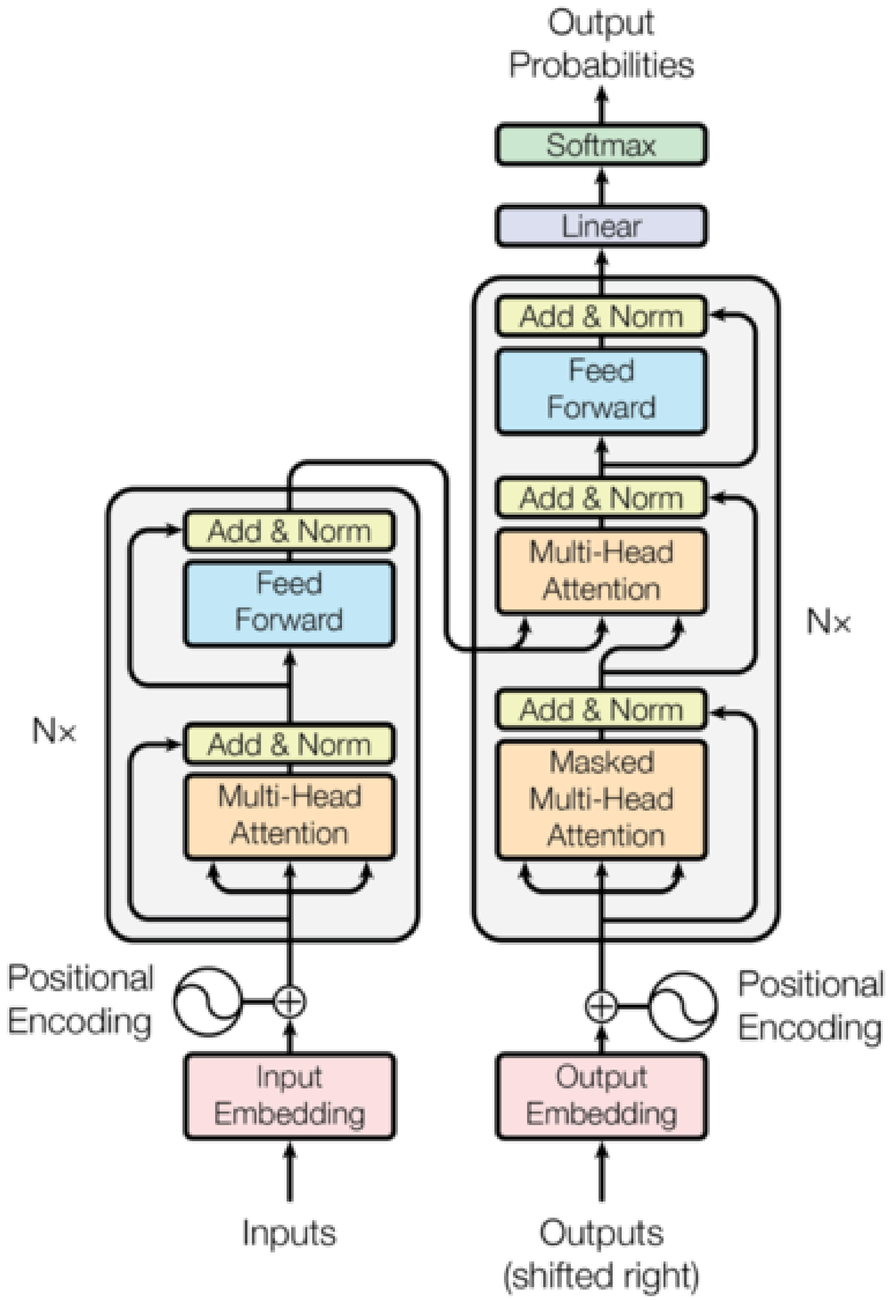
Encoder : Feed Forward

- Feed forward network
- Followed by residual connection and layer normalization



Transformer : Decoder

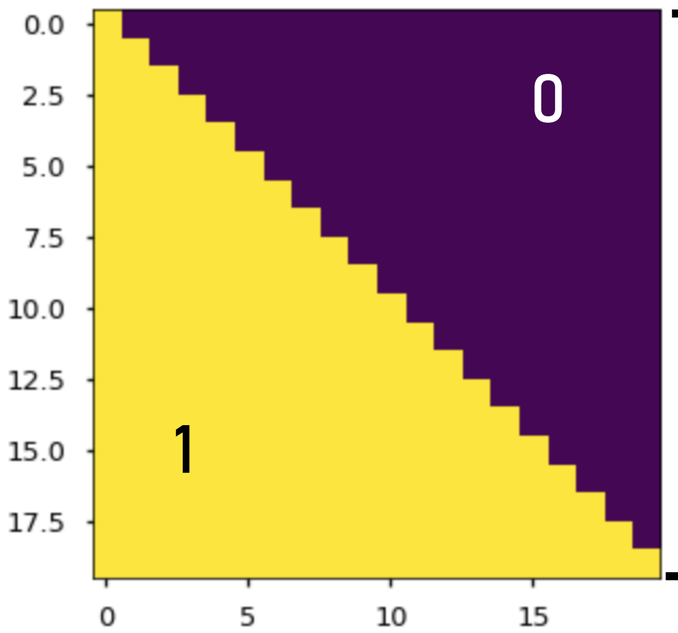
- Auto Regressive Decoding
- Based on Transformer layers



Decoder : Masked Multi-head Attention

- Auto Regressive Decoding

  Current token can attend only left-side tokens because in decoding step the right-side tokens are not generated. Attention weights are multiplied by this mask matrix.

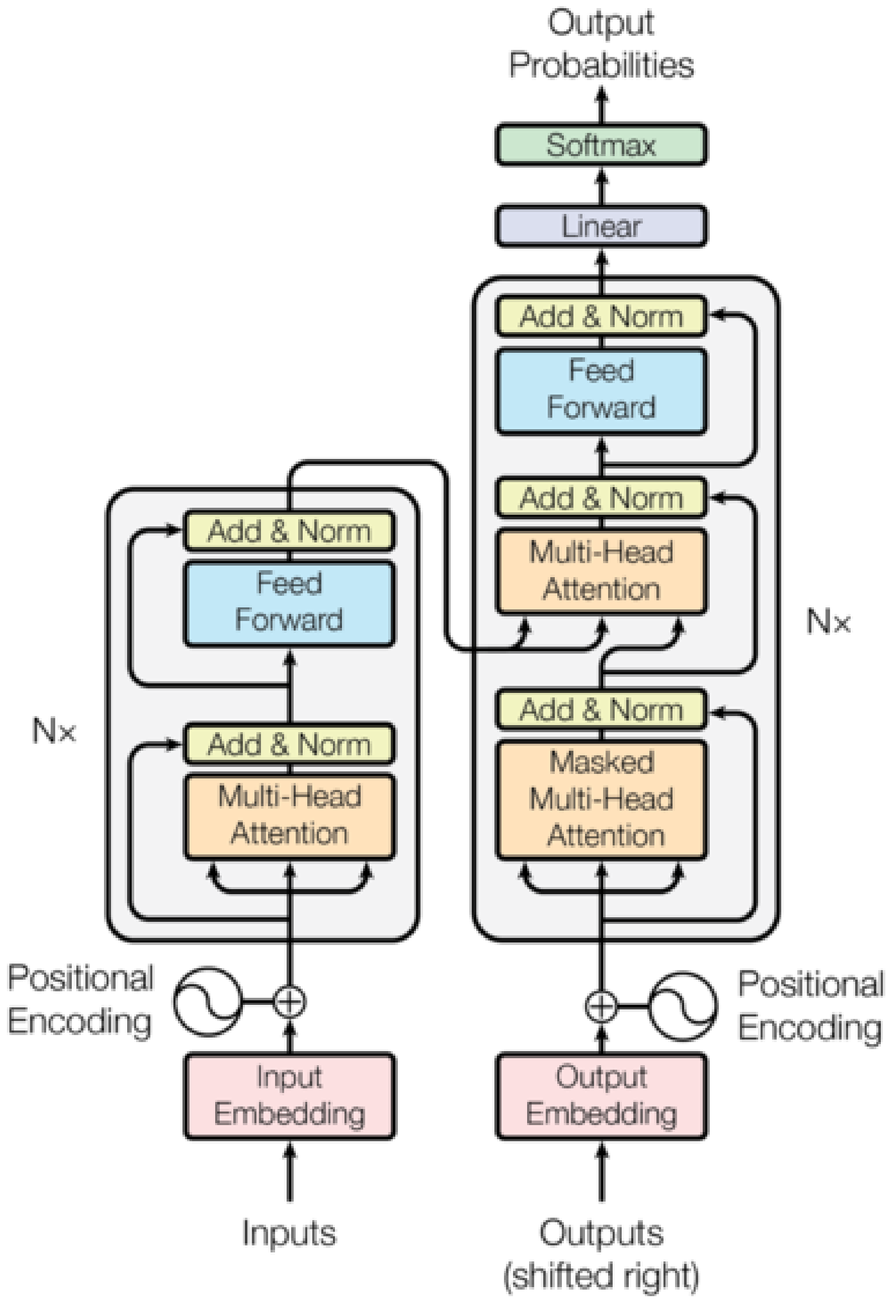


Mask vector for the first token. [1, 0, 0, 0, …, 0]

Mask vector for the last token. [1, 1, 1, 1, …, 1]

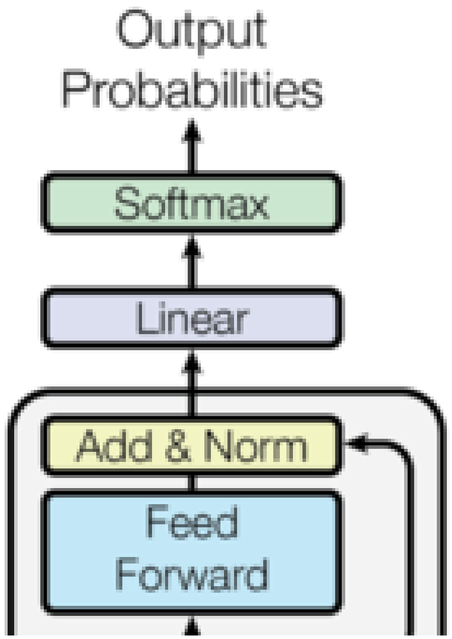
Decoder-Encoder Attention

- Query data from the encoder outputs.
- Encoder output (K,V) , Decoder State (Q)



Decoder : Feed Forward and Softmax

- Predict target word distribution



Optimization

- Label Smoothed Regularization

$$\bar{y}_j^t = (1-\epsilon)\bar{y}_j + \frac{\epsilon}{V}$$

For example, $V = 3$, $\epsilon = 0.3$

$$\bar{y}_{\text{true(smooth)}} = \begin{bmatrix}0\\0\\1\end{bmatrix} - \begin{bmatrix}0\\0\\\epsilon\end{bmatrix} + \frac{1}{3}\begin{bmatrix}\epsilon\\\epsilon\\\epsilon\end{bmatrix} = \begin{bmatrix}0.1\\0.1\\0.8\end{bmatrix}$$

Optimization

- Label Smoothed Regularization

$$\bar{y}_j^t = (1-\epsilon)\bar{y}_j + \frac{\epsilon}{V}$$

For example, $V = 3$, $\epsilon = 0.3$

$$\bar{y}_{\text{true(smooth)}} = \begin{bmatrix}0\\0\\1\end{bmatrix} - \begin{bmatrix}0\\0\\\epsilon\end{bmatrix} + \frac{1}{3}\begin{bmatrix}\epsilon\\\epsilon\\\epsilon\end{bmatrix} = \begin{bmatrix}0.1\\0.1\\0.8\end{bmatrix}$$

- Label Smoothed NLL

$$= -\sum_{t=1}^{T}\sum_{j=1}^{V} \bar{y}_j^t \log \hat{y}_j^t$$

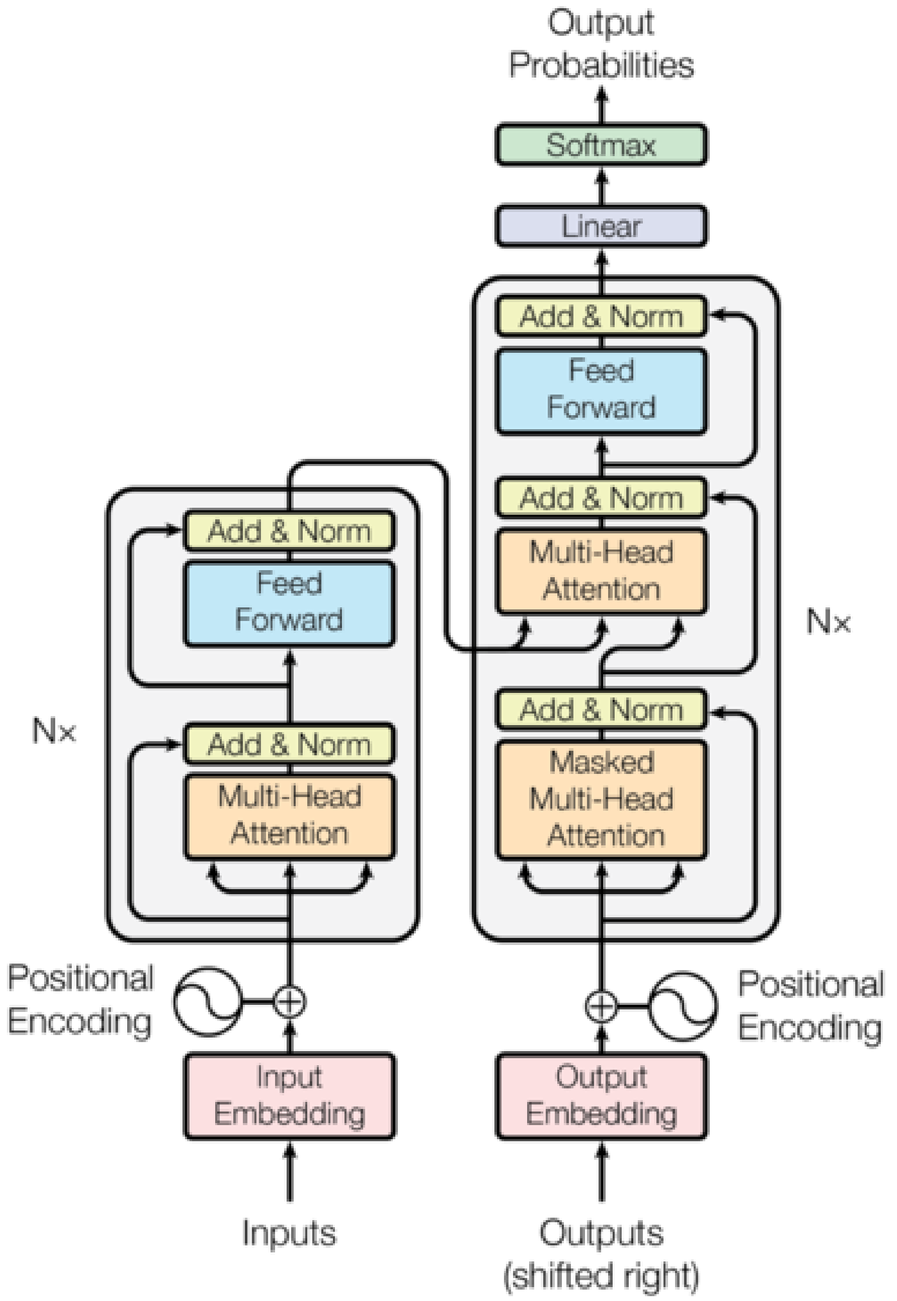
Decoding : Auto Regressive

- Greedy search
- Beam Search

https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Summary

- Self-Attention Network
- Multi-Head Self Attention
- Positional Encoding
- Encoder
- Decoder
- Optimization



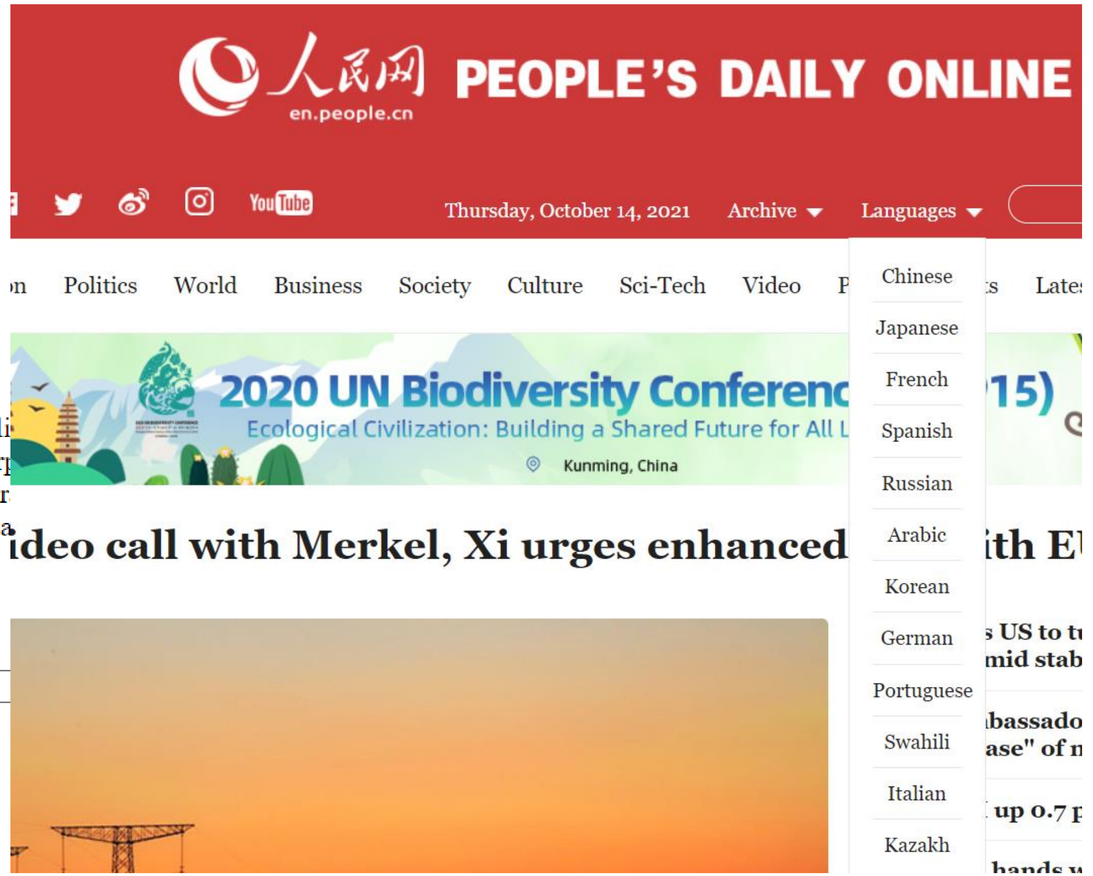
Data Preparation for MT

Data Preparation for MT

- **Data Collection**
- **Data Cleansing and Tokenization**
- **Split Train / Validation / Test**
- **Additional Preprocessing Steps for NMT**
  - Subword Preparation
  - Padding and Binarizing

Data Collection

- **Data Collection**

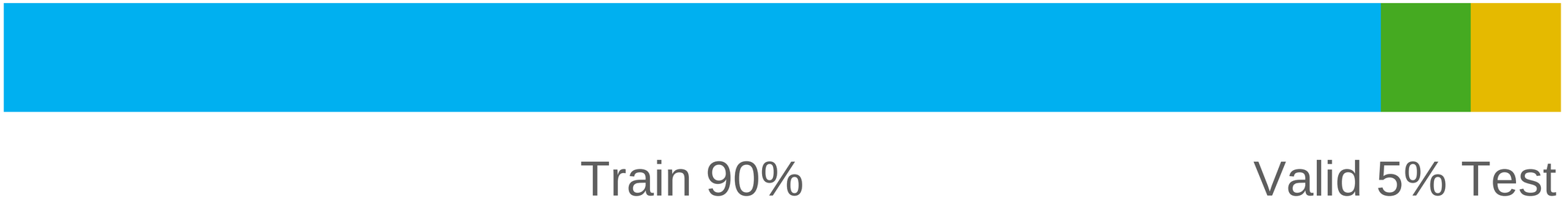


Data Cleansing and Tokenization

- Clean empty line
- Align parallel sentences
- Deduplicate
- Tokenize (Word, character segmentation)
- Filter low quality pairs
    - Alignment score
    - Length ratio (# of source tokens / # of target tokens)

Split Train / Validation / Test

- Shuffle
- Train / Validation / Test



Additional Steps for NMT

- Subword Units (Sentence piece, Byte-pair encoding)

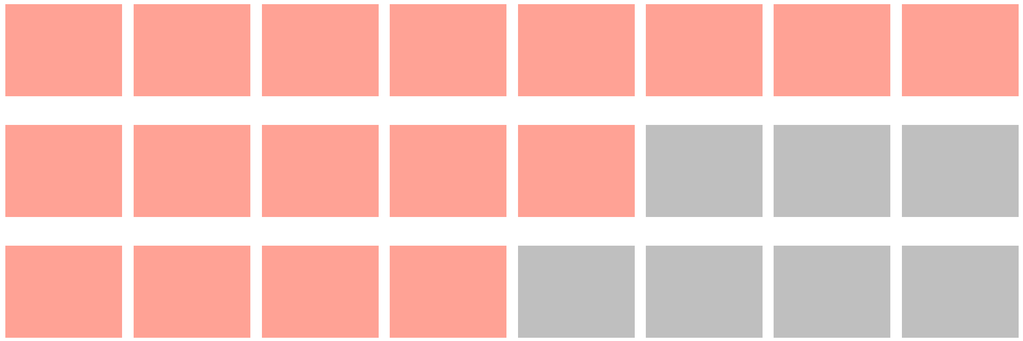
  ซีรีย์ จีน เรื่อง ดาบมังกรหยก

  ซีรีย์ จีน เรื่อง ดาบ@@ มังกร@@ หยก

Rico Sennrich, Barry Haddow and Alexandra Birch, Neural Machine Translation of Rare Words with Subword Units, ACL, 2016

Additional Steps for NMT

- Padding



- Binarizing – Convert strings to tensors

Summary

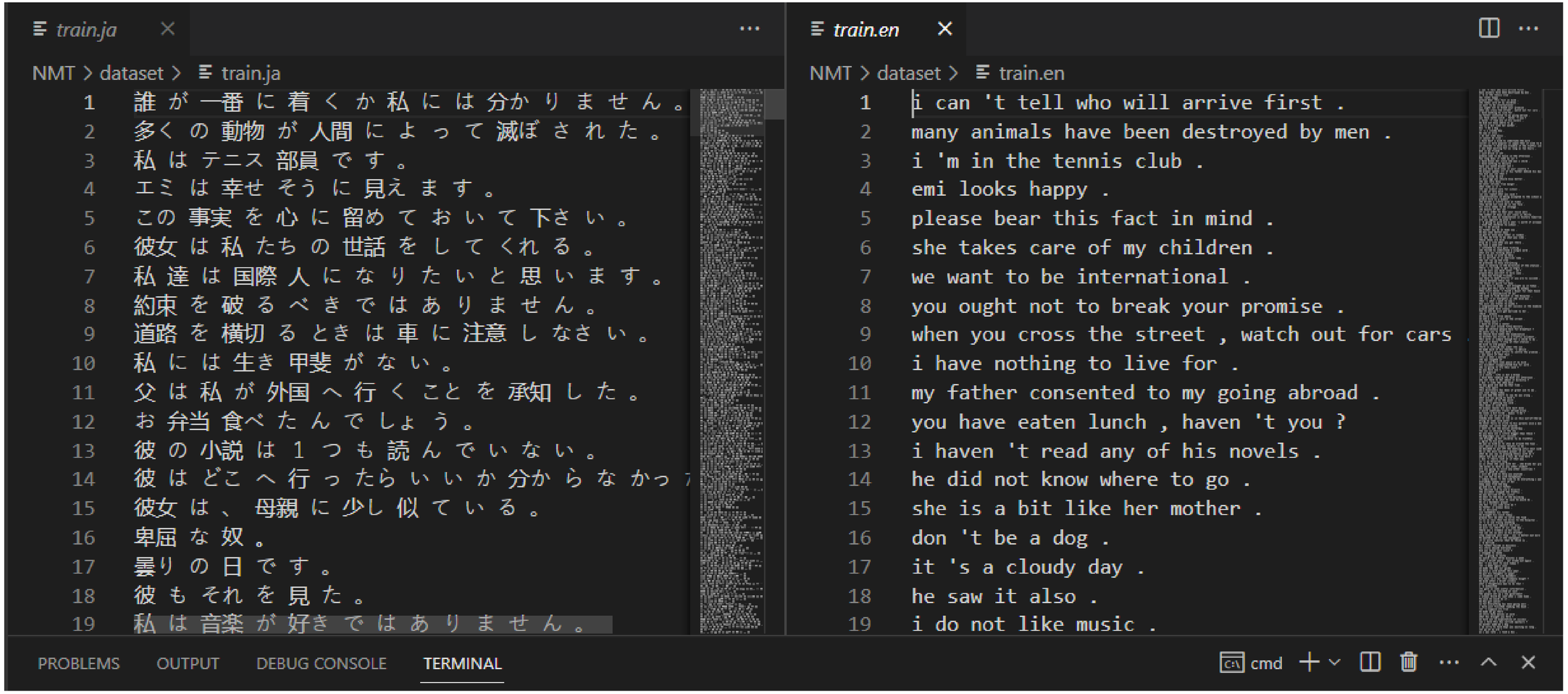
- **Data Collection**
- **Data Cleansing and Tokenization**
- **Split Train / Validation / Test**
- **Additional Preprocessing Steps for NMT**
  - Subword Preparation
  - Padding and Binarizing

Training Transformer Model

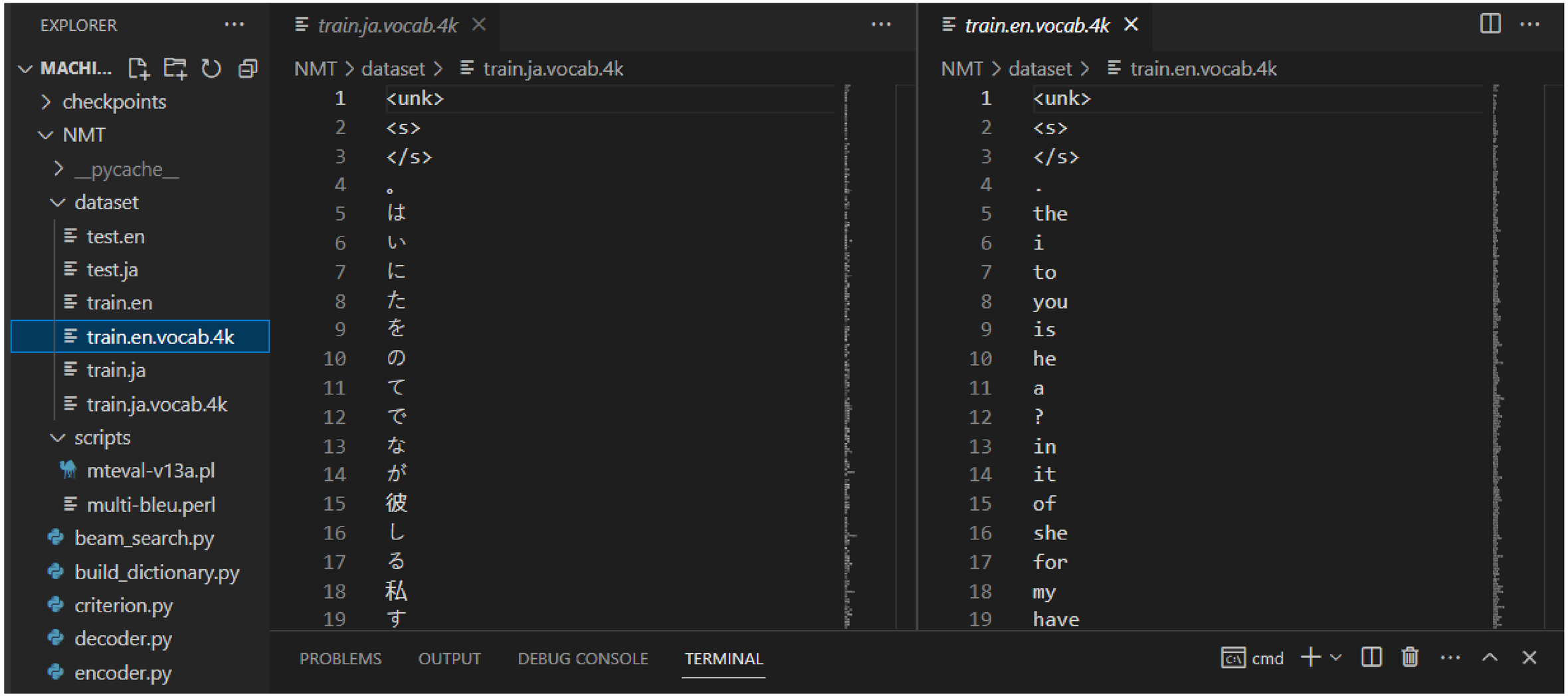
Preparation Steps

- **Dataset Preparation**
  - Bilingual Data Preparation
  - Dictionary

Bilingual Data



Dictionary (Word list)



Training Step Walkthrough

- **Training NMT**
  - Load/Build Dictionary
  - Load Dataset
  - Build Model
  - Create padded dataset
  - Create mini batch
  - Training Loop
  - Decoding (Translate)

Load/Build Dictionary

- Use for converting words in sentences into ids. (For mapping embedding vectors)
- Convert ids of target sentence back to words.
- Adding special tokens (<start>,<end>,<blank>)

NMT > main_loop.py > ...

```
61  def build_dictionary():
62      th2id = {}
63      id2th = {}
64
65      with codecs.open("./dataset/train.ja.vocab.4k", mode='r'
66          data_th = file.readlines()
67
68          th2id["<blank>"] = len(th2id)
69          id2th[th2id["<blank>"]] = "<blank>"
70
71          th2id["<start>"] = len(th2id)
72          id2th[th2id["<start>"]] = "<start>"
73
74          th2id["<end>"] = len(th2id)
75          id2th[th2id["<end>"]] = "<end>"
76
77          th2id["<unk>"] = len(th2id)
78          id2th[th2id["<unk>"]] = "<unk>"
79
80          for id , item in enumerate(data_th):
81              w = item.strip().split(" ")[0]
82              th2id[w] = len(th2id)
83              id2th[th2id[w]] = w
84
```

Load Dataset

learn_word_alignment.py | main_loop.py M

NMT > main_loop.py > load_dataset_and_train

```
316     model.train()
317
318 def load_dataset_and_train():
319     global use_cuda
320
321     
322     max_len = MAX_LEN
323     sample_size = 50000
324     print("Load dataset....")
325     with codecs.open("./dataset/train.en", mode='r',encoding="utf-8") as file:
326         data_en = file.readlines()[:sample_size]
327
328     with codecs.open("./dataset/train.ja", mode='r',encoding="utf-8") as file:
329         data_th = file.readlines()[:sample_size]
330
331     corpus_en =[]
332     corpus_th =[]
333
334     for en,th in zip(data_en,data_th):
335         len_en = len(en.strip().split())
336         len_th = len(th.strip().split())
337         if len_en < max_len and len_th < max_len:
338             corpus_en.append(en)
339             corpus_th.append(th)
340
```

Build Model

- Criteria / Loss Function
- Model (Neural Network)
- Optimizer (SGD, Adam,…)

```
print("Setting up loss function ....")
criterion = LabelSmoothing(size=len(en2id), padding_idx=en2id["<blank>"],
                           smoothing=0.1)

print("Make Model...")
model = make_model(len(th2id), len(en2id), N=6,
                   d_model=512, d_ff=2048, h=8, dropout=0.1)

# If resume from saved checkpoint, uncomment this line.
# model.load_state_dict(torch.load("../checkpoints/update_3000.pt"))

print("Build Optimizer...")
optimizer = torch.optim.Adam(model.parameters(), lr=0.0007,
                             betas=(0.9, 0.98), eps=1e-9)

if use_cuda:
    criterion = criterion.cuda()
    model = model.cuda()

model_opt = NoamOpt(model.src_embed[0].d_model, 1, 4000, optimizer)
```

Transformer Model

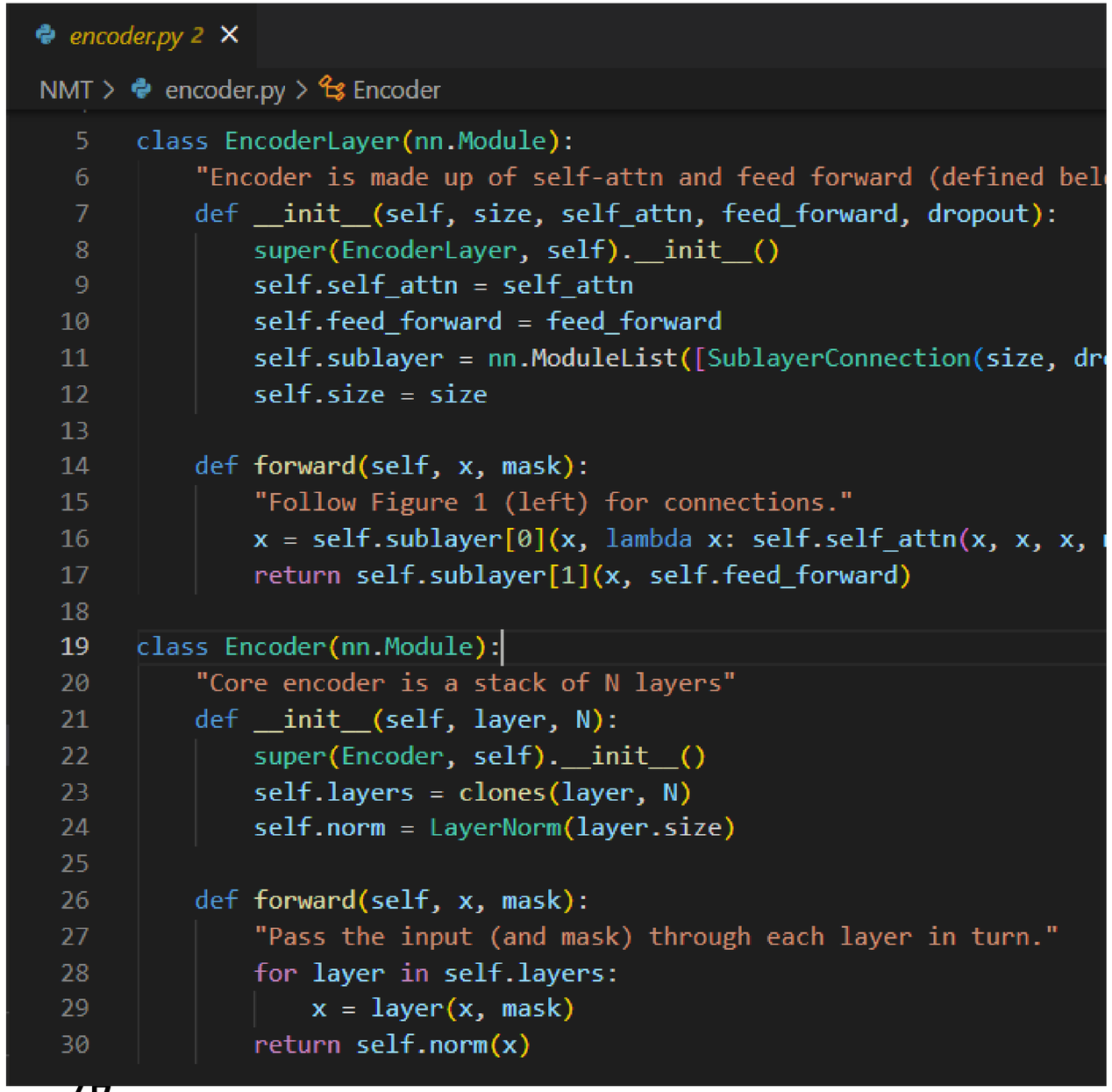
- **Deep Learning Library**
  - Pytorch
    - Tensor Calculation
    - Autograd / Auto diff
    - Optimization

NMT > main_loop.py > ...

```python
def make_model(src_vocab, tgt_vocab, N=6,
               d_model=512, d_ff=2048, h=8, dropout=0.1):
    "Helper: Construct a model from hyperparameters."
    print("Multihead...")
    attn = MultiHeadedAttention(h, d_model)

    print("Position...")
    ff = PositionwiseFeedForward(d_model, d_ff, dropout)

    print("PositionEncoding...")
    position = PositionalEncoding(d_model, dropout)

    print("Build Encoder-Decoder")
    model = EncoderDecoder(
        Encoder(EncoderLayer(d_model, c(attn), c(ff), dropout), N),
        Decoder(d_model,h, d_ff, N, dropout),
        nn.Sequential(Embeddings(d_model, src_vocab), c(position)),
        nn.Sequential(Embeddings(d_model, tgt_vocab), c(position)),
        Generator(d_model, tgt_vocab))

    # This was important from their code.
    # Initialize parameters with Glorot / fan_avg.
    print("Init Params...")
    for p in model.parameters():
        if p.dim() > 1:
            nn.init.xavier_uniform_(p)
```

Transformer Model

- **Pytorch NN Model**
  - nn.Module
    - Linear
    - Multi-headed Attention
    - Dropout
    - LayerNorm



Create Padded Dataset

- Pad to MAX_LEN using

<blank> symbol

- Add **<start> , <end>**
- Replace unknow to **<unk>**

main_loop.py 4

NMT > main_loop.py > make_model

```
109
110  def create_padded_dataset(data_th, data_en, th2id, en2id, maxlen=80):
111
112      padded_th = []
113      padded_en = []
114
115      for sentence in data_th:
116          #print(sentence)
117          token = sentence.strip().split()[0:maxlen-2]
118          #print(len(token))
119          if len(token) < maxlen-2:
120              pad = ["<blank>"] * (maxlen-2 - len(token) + 1)
121              token.append("<end>")
122              token.extend(pad)
123          else:
124
125              token = ["<start>"] + token + ["<end>"]
126
127          #print(len(token), maxlen)
128          assert len(token) == maxlen
129
130          temp = [th2id.get(t,th2id["<unk>"]) for t in token]
131          padded_th.append(temp)
132
```

Create Mini Batch

- Pack input/output for feeding to the model
  - Input / Output token ids
  - Mask
  - Decoder Attention Mask

main_loop.py 4

NMT > main_loop.py > make_model

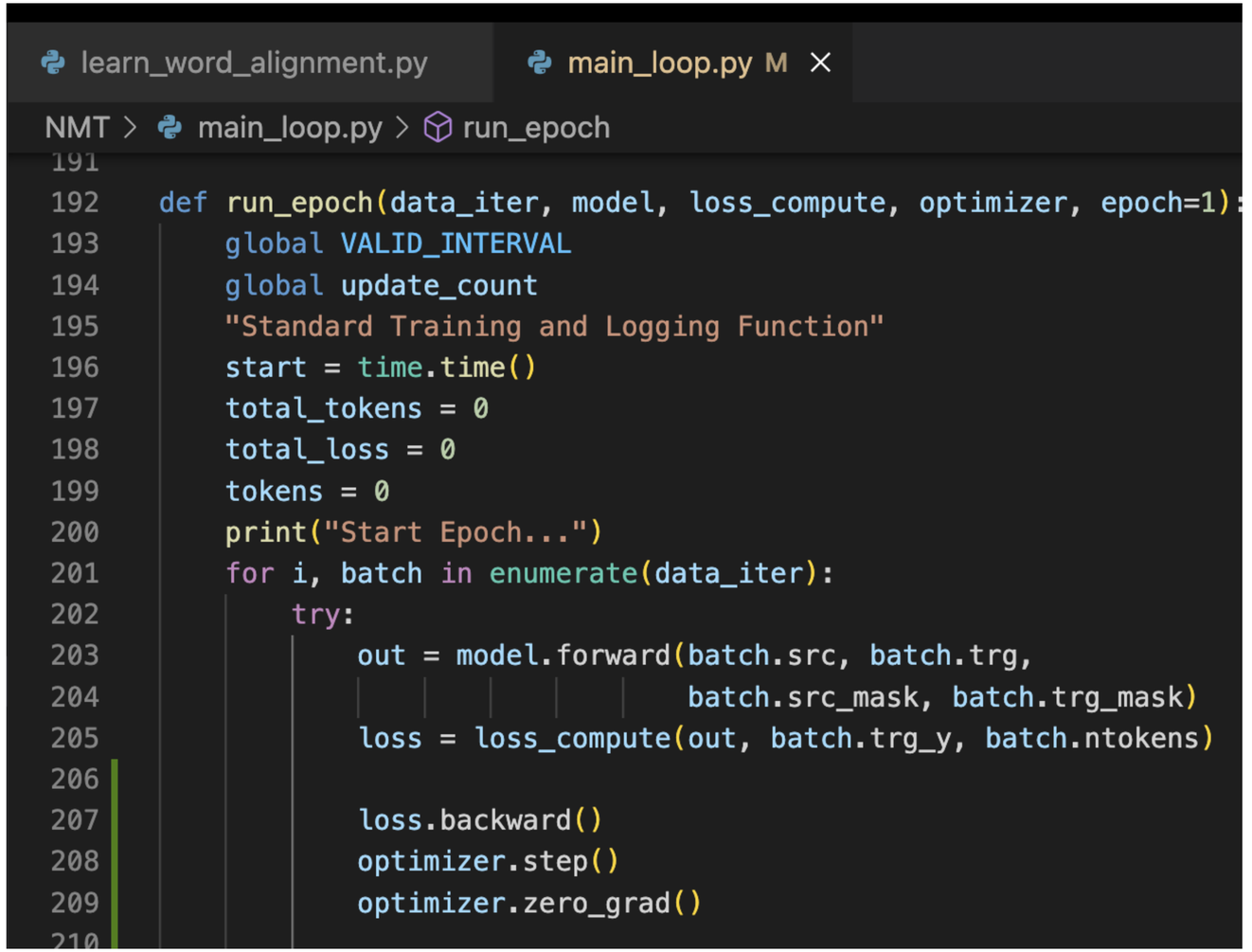
```
147
148   class Batch:
149       "Object for holding a batch of data with mask during training."
150       def __init__(self, src, trg=None, src_pad=0, trg_pad=0):
151           self.src = src
152           self.src_mask = (src != src_pad).unsqueeze(-2)
153           if trg is not None:
154               self.trg = trg[:, :-1]
155               self.trg_y = trg[:, 1:]
156               self.trg_mask = \
157                   self.make_std_mask(self.trg, trg_pad)
158
159               if use_cuda:
160                   self.trg_mask = self.trg_mask.cuda()
161
162               self.ntokens = (self.trg_y != trg_pad).data.sum()
163
164       @staticmethod
165       def make_std_mask(tgt, pad):
166           "Create a mask to hide padding and future words."
167           tgt_mask = (tgt != pad).unsqueeze(-2)
168           tgt_mask = tgt_mask & subsequent_mask(tgt.size(-1)).type_as(tgt_mask.data)
169           return tgt_mask
```

Training Loop

- **Process for a single mini-batch**
  - Load padded mini-batch
  - Batching (Tensor of input/output tokens, mask, …)
  - Run forward pass
  - Run backward pass
  - Update weights with optimizer.step()
  - Clear computation graph using optimizer.zero_grad()

Training Loop

- **Process for a single mini-batch**
  - Load padded mini-batch
  - Batching (Tensor of input/output tokens, mask, …)
  - Run forward pass
  - Run backward pass
  - Update weights with optimizer.step()
  - Clear computation graph using optimizer.zero_grad()



Training over Dataset

main_loop.py 4

NMT > main_loop.py > make_model

```
372     print("Begin Training...")
373
374     for epoch in range(1,200):
375         print("Entering epoch : %d" % epoch)
376         model.train()
377
378         padded_dataset = data_gen_v2(padded_data_th,padded_data_en,
379                                      th2id["<blank>"], en2id["<blank>"],
380                                      batch_size)
381
382         loss_function = SimpleLossCompute(model.generator, criterion, model_opt)
383
384         run_epoch(padded_dataset,
385                   model, loss_function,
386                   optimizer,epoch)
387
388         torch.save(model.state_dict(), "../checkpoints/chk_%d.pt"%epoch)
389
390     print("Training Completed.....")
391
```

Decoding with Transformer

Greedy Decoder

Tensor-in-Tensor-out

main_loop.py 4

NMT > main_loop.py > make_model

```
240
241     def greedy_decode(model, src, src_mask, max_len, start_symbol, e
242         memory = model.encode(src, src_mask)
243         ys = torch.ones(1, 1).fill_(start_symbol).type_as(src.data)
244         for i in range(max_len-1):
245             out = model.decode(memory, src_mask,
246                                ys,
247                                subsequent_mask(ys.size(1))
248                                    .type_as(src.data))
249             prob = model.generator(out[:, -1])
250             _, next_word = torch.max(prob, dim = 1)
251             next_word = next_word.data[0]
252             if next_word == end_symbol:
253                 break
254             ys = torch.cat([ys,
255                             torch.ones(1, 1).type_as(src.data).fill_
256         return ys
257
```

Translation

- Set model to eval mode
- Convert input to ids
- Decode
- Convert ids to target words
- Remove special tokens

main_loop.py 4, M

NMT > main_loop.py > test_one

```
274  def test_one(dictionaries, model, input, use_cuda ):
275      th2id,id2th,en2id,id2en = dictionaries
276      model.eval()
277
278      print("INPUT : ", input.strip())
279
280      src, src_mask = encode_line(input.strip(), th2id)
281      src = src.unsqueeze(0)
282      src_mask = src_mask.unsqueeze(0)
283
284      if use_cuda:
285          print("CUDA")
286          model = model.cuda()
287          src = src.cuda()
288          src_mask = src_mask.cuda()
289
290      out = greedy_decode(model,src,src_mask,
291                          MAX_LEN,en2id["<start>"], en2id["<end>"])
292
293      output = out[0].tolist()
294
295      trg = [id2en[id] for id in output[1:]]
296      out = " ".join(trg).replace(" <blank>","")
297
298      print("OUTPUT : ",out)
299      print("---")
300      return out
```

Homework (10 points, due date 10 March)

- Get familiar with the **test_one** function and extend to translate the test set
- Put your code in the **eval** function (7 pts) and evaluate BLEU score (3 pts, if your model can run successfully and produce translation results).
- Write a brief report.
  - Report: your code, BLEU score and translation results (500 sentences)
- **Noted:** For training NMT, you can work with other students (max 3 member in a team) but **report as individual**, If you have your team please **give team member names in your report**.

Homework (10 points, due date 10 March)

- You can use google colab or your PCs to run
- If you use Graphic card, set use_cuda = True.
- Submit your assignment to : peerachet.porkaew@gmail.com
- Subject : **2023 - Transformer Assignment (your student ID)**

Thank you