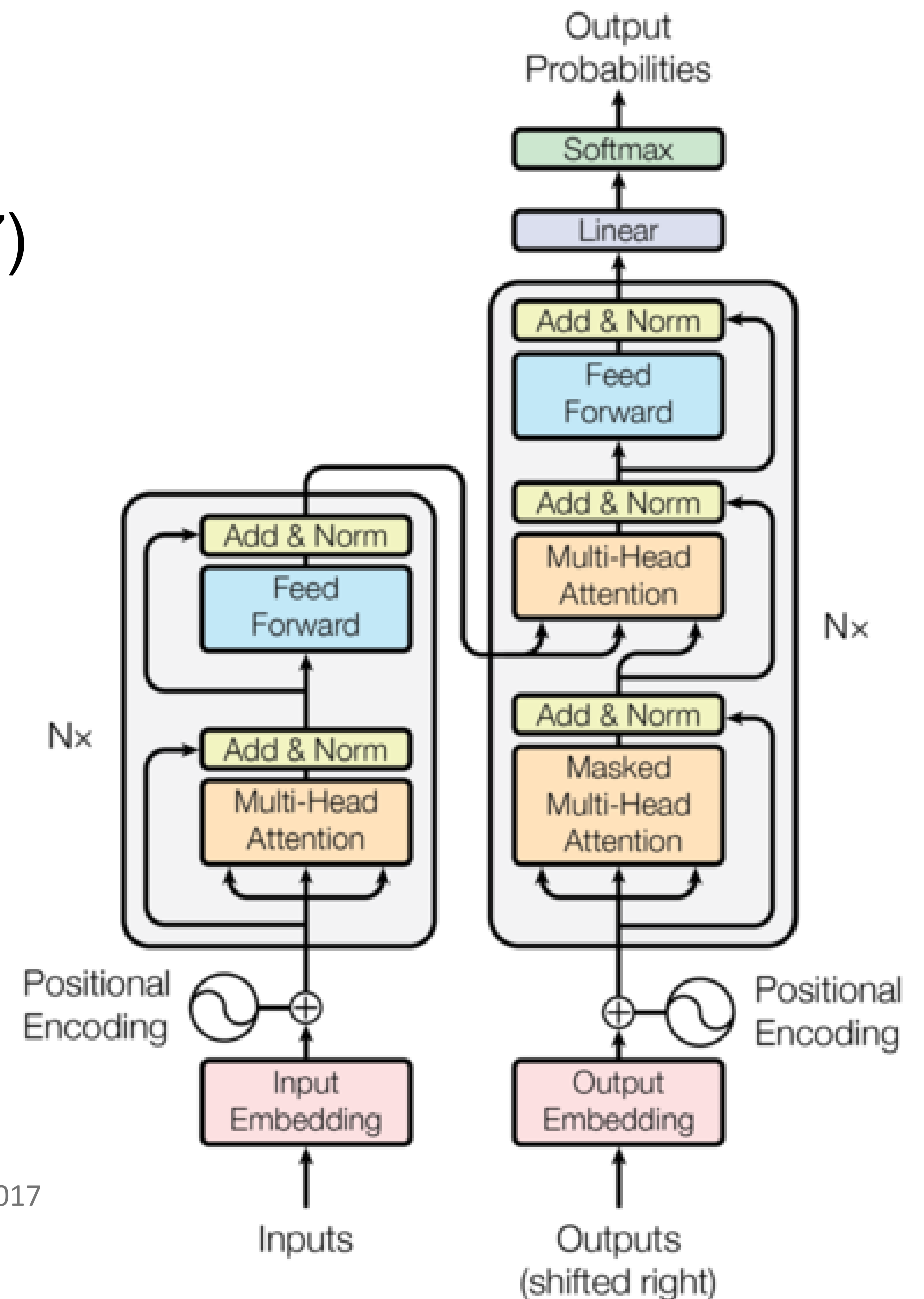


# Transformer

# Transformer

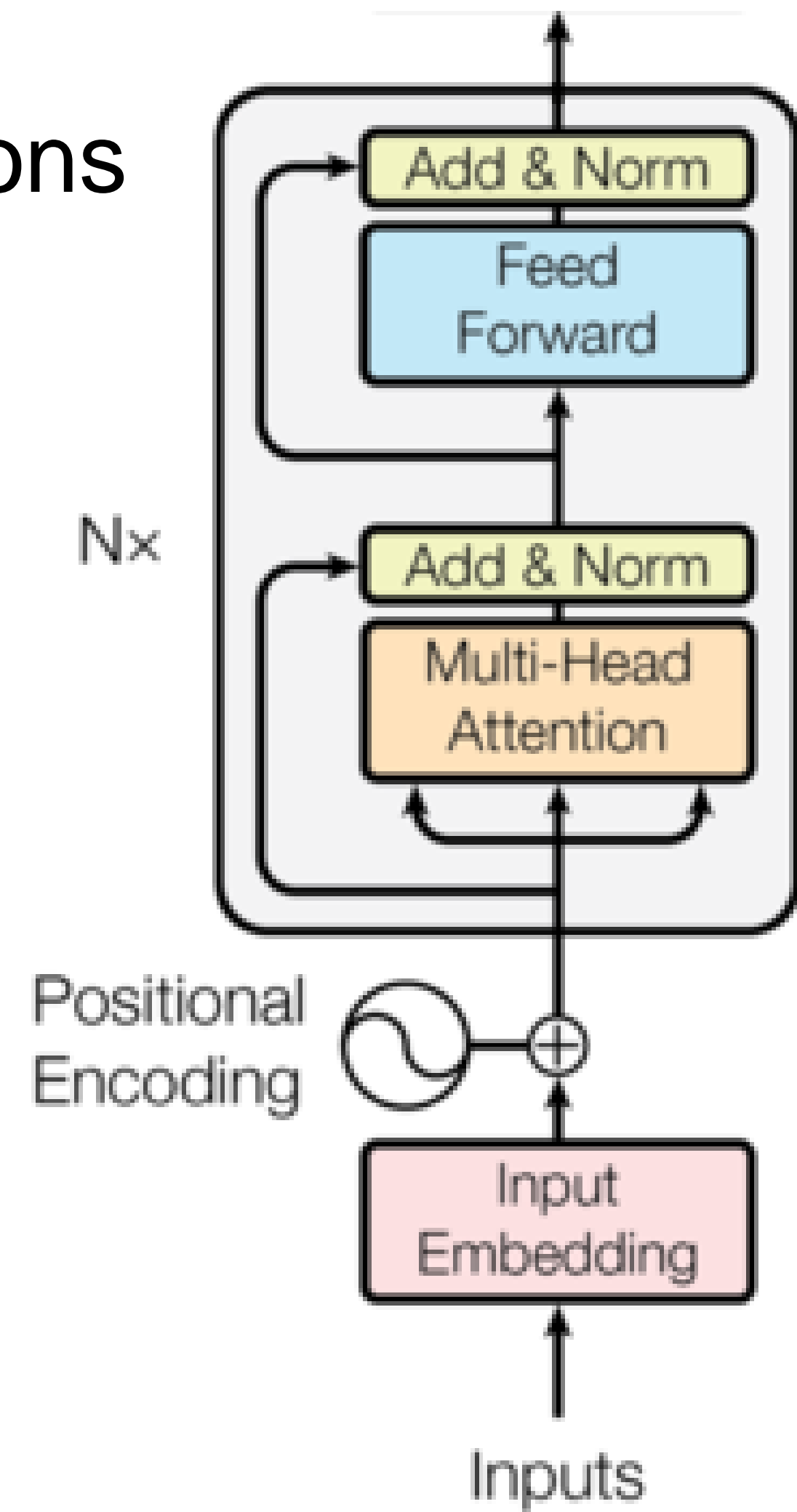
- Attention is all you need (Vaswani et al., 2017)
- An encoder-decoder framework for sequence-to-sequence modeling
- No recurrent units



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

# Transformer Encoder

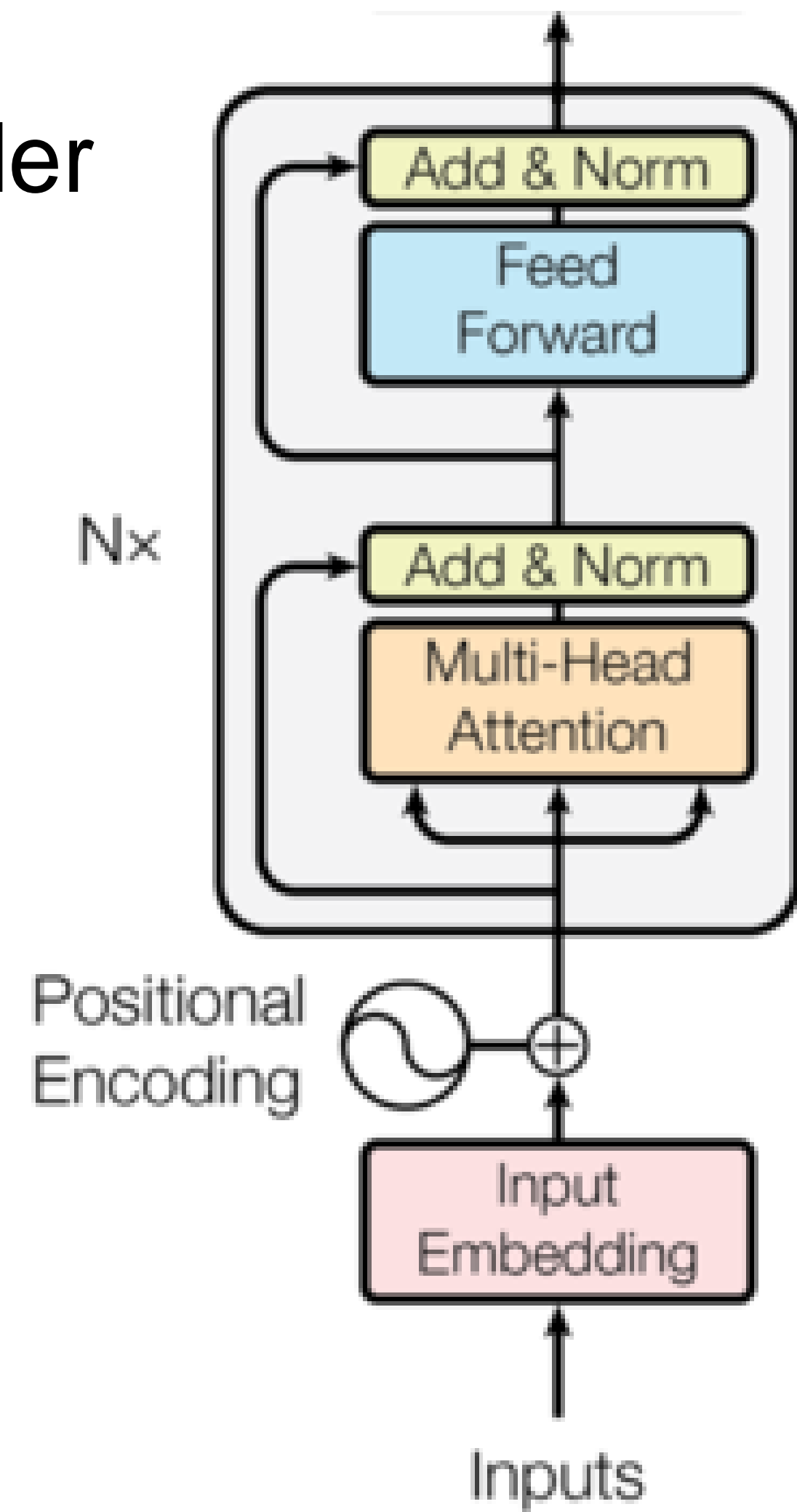
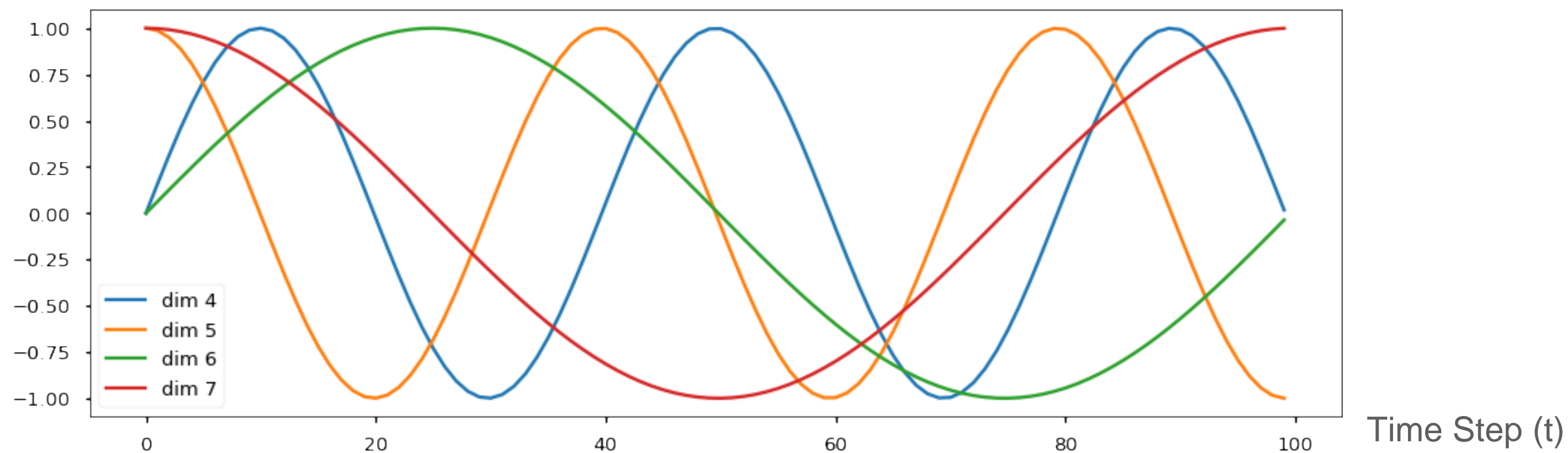
- N layers of Transformer blocks with residual connections
- Parameters in each layer are not shared
- Input tokens are processed in parallel



# Positional Encoding

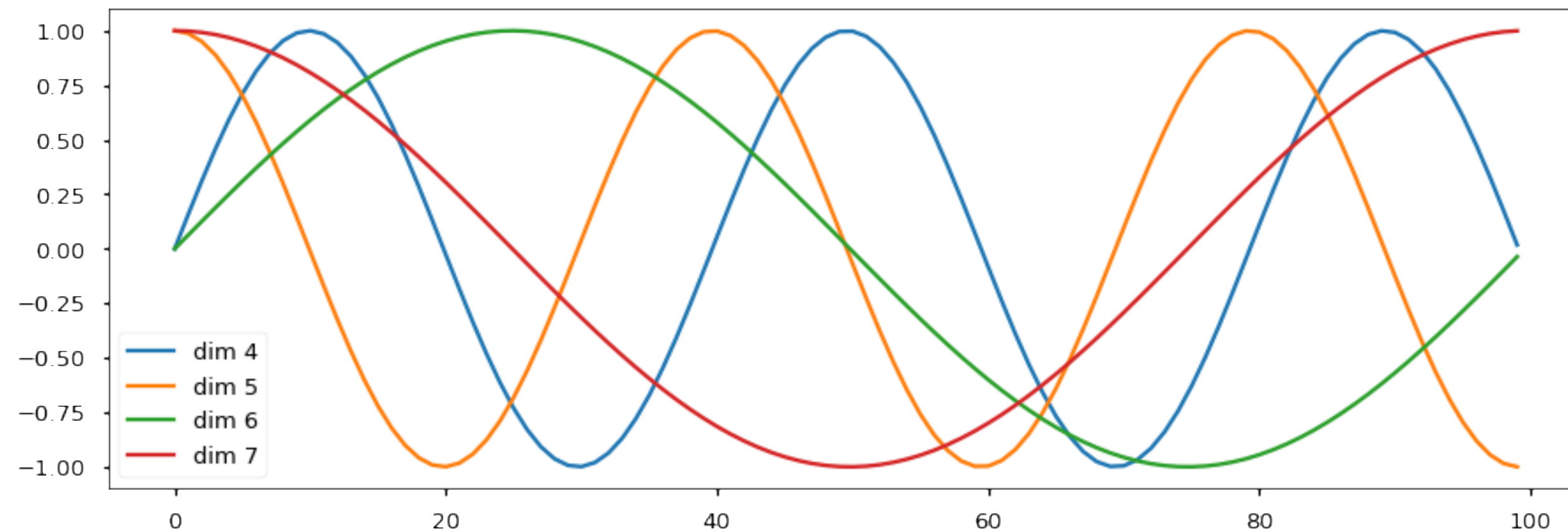
- Positional encoding provide the model about word order
- Sinusoidal position encoding

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



# Positional Encoding

$$\mathbf{x}'_t = W_{\text{emb}}(\mathbf{x}_t) + \vec{p}_t$$



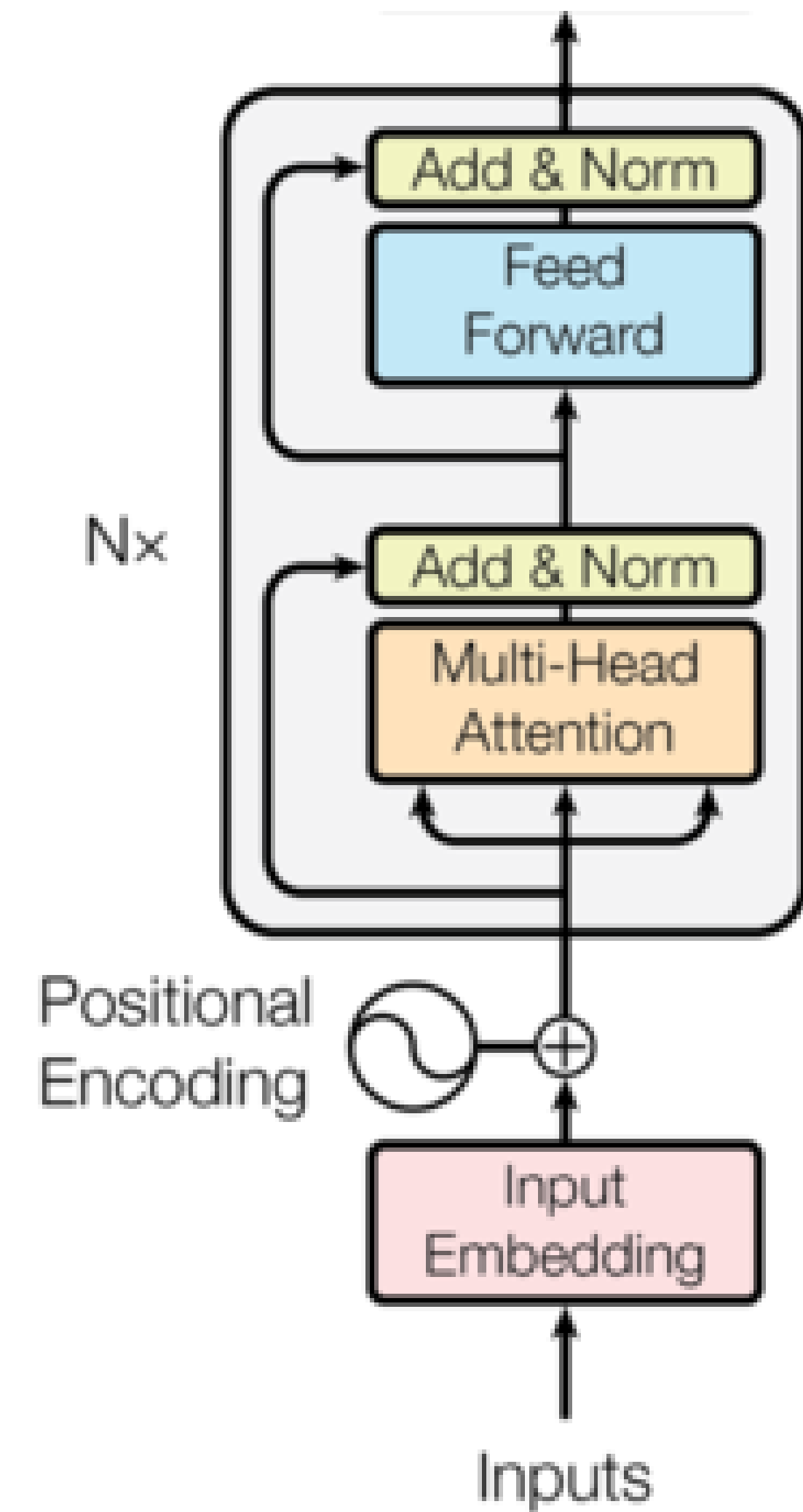
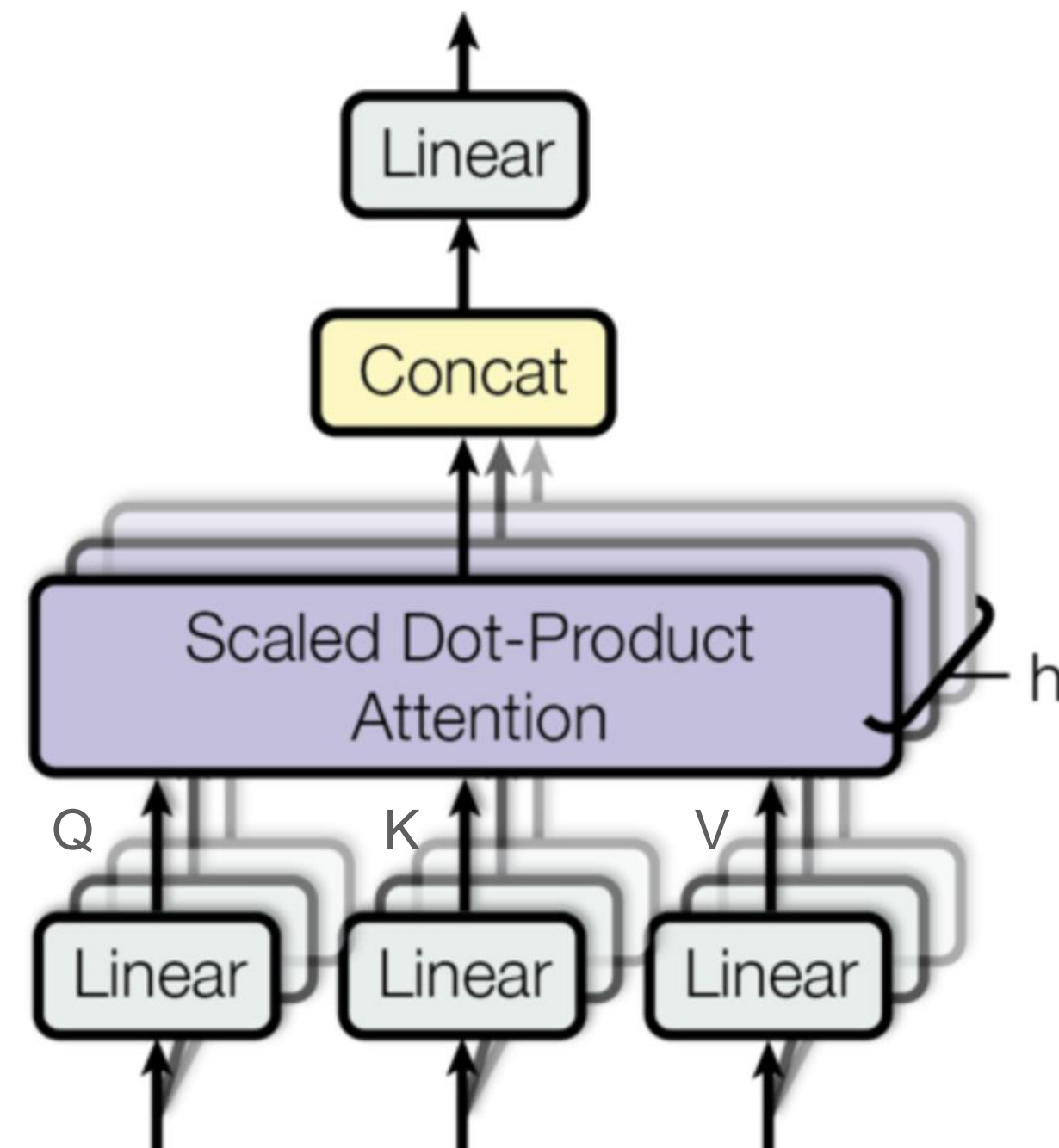
Time Step (t)

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

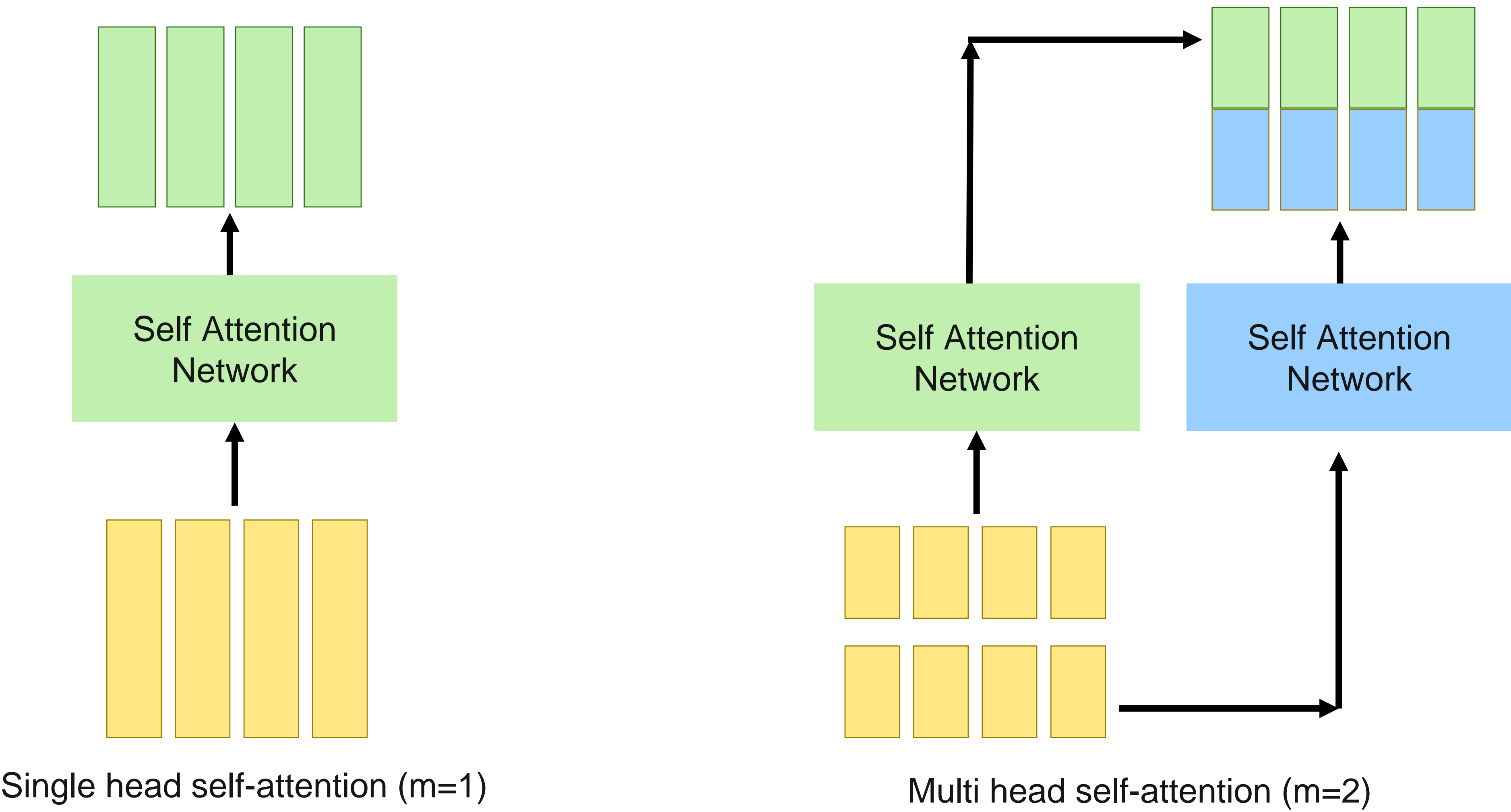
$$\vec{p}_t = \begin{bmatrix} \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \\ \sin(\omega_2 \cdot t) \\ \cos(\omega_2 \cdot t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} \cdot t) \\ \cos(\omega_{d/2} \cdot t) \end{bmatrix}_{d \times 1}$$

# Multi-head Attention

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



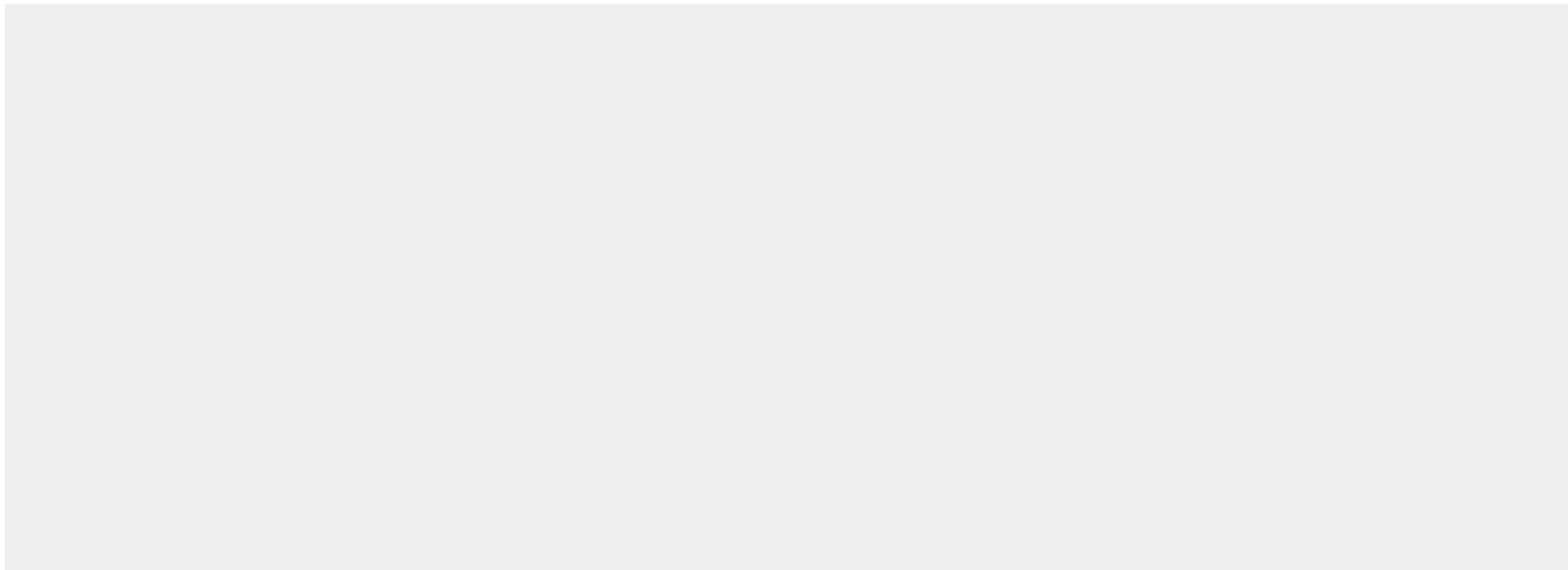
# Multi-head Attention



# Self-Attention Network

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

Self-attention



input #1

1	0	1	0
---	---	---	---

input #2

0	2	0	2
---	---	---	---

input #3

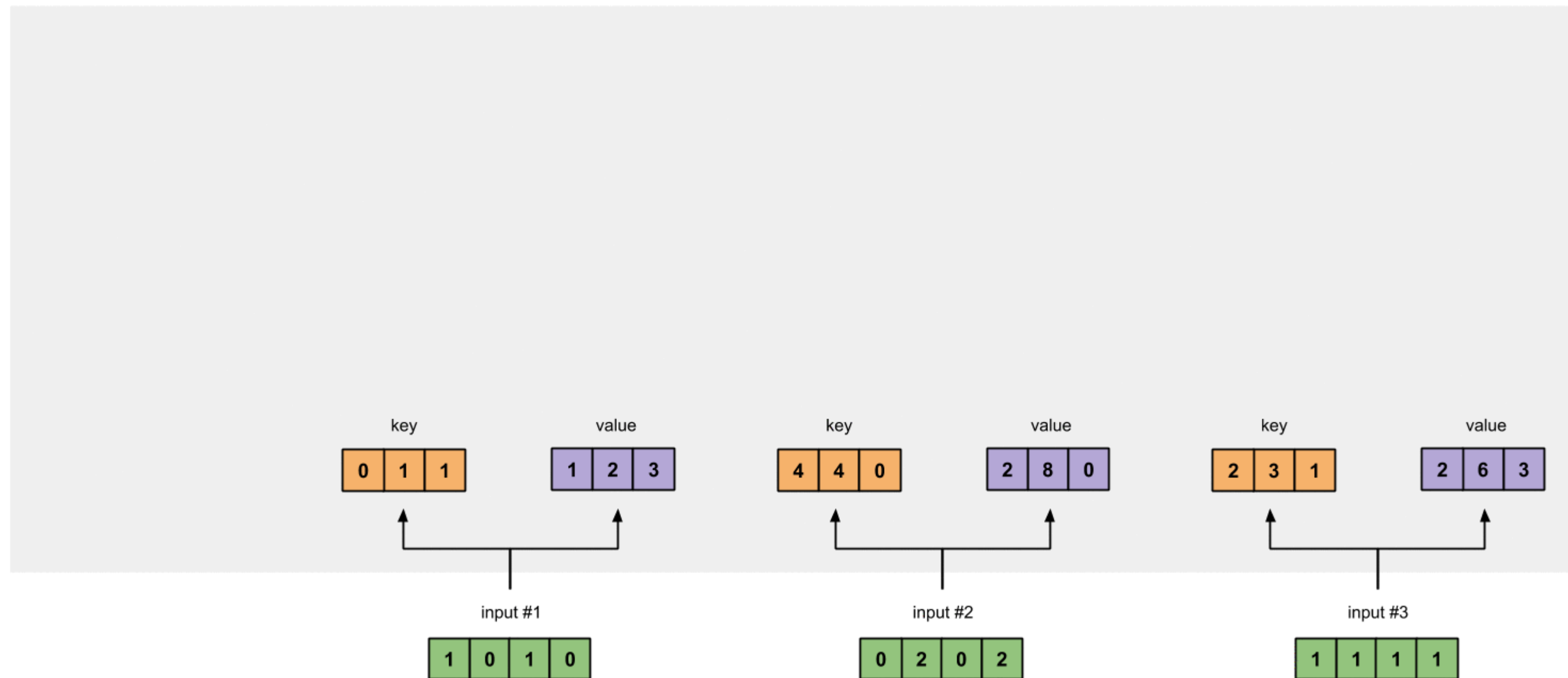
1	1	1	1
---	---	---	---



# Self-Attention Network

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

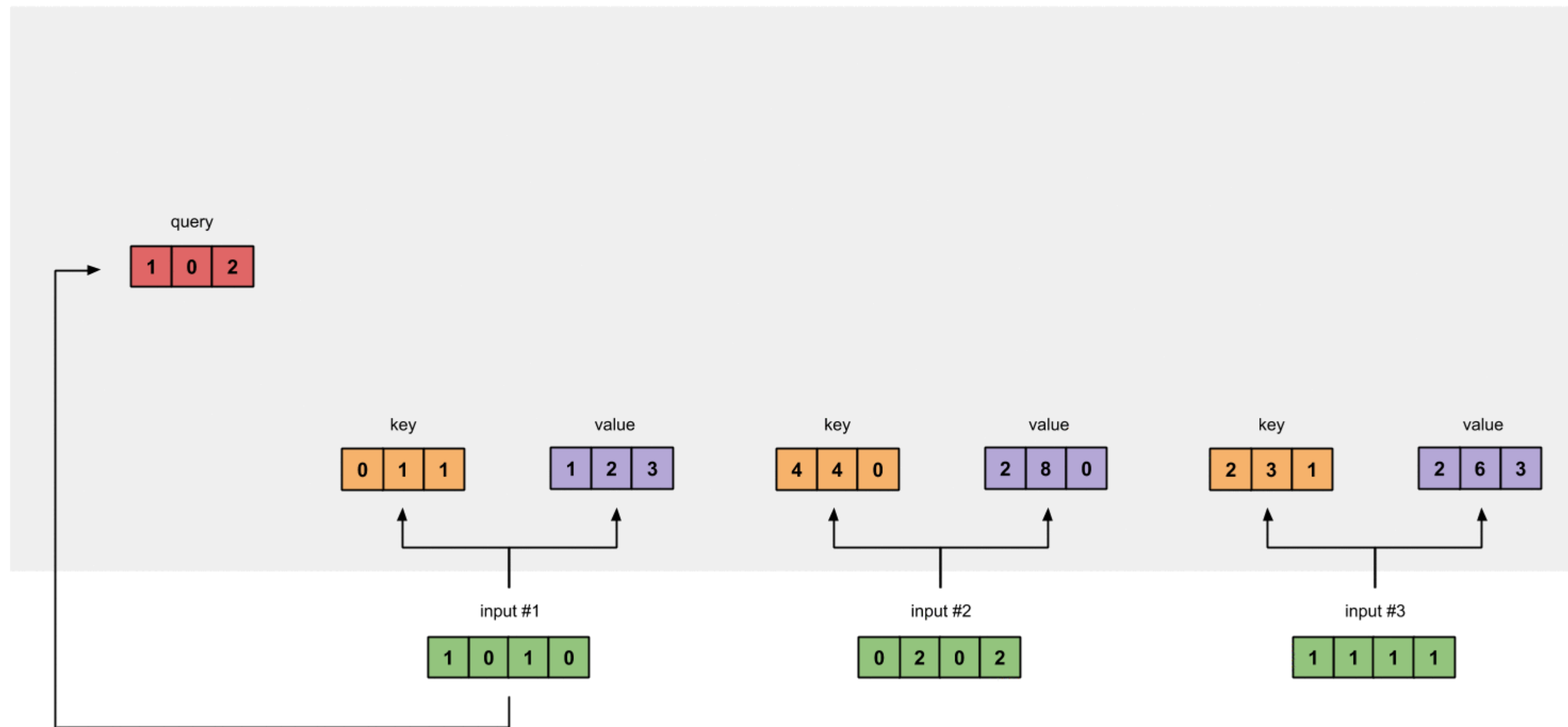
Self-attention



# Self-Attention Network

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

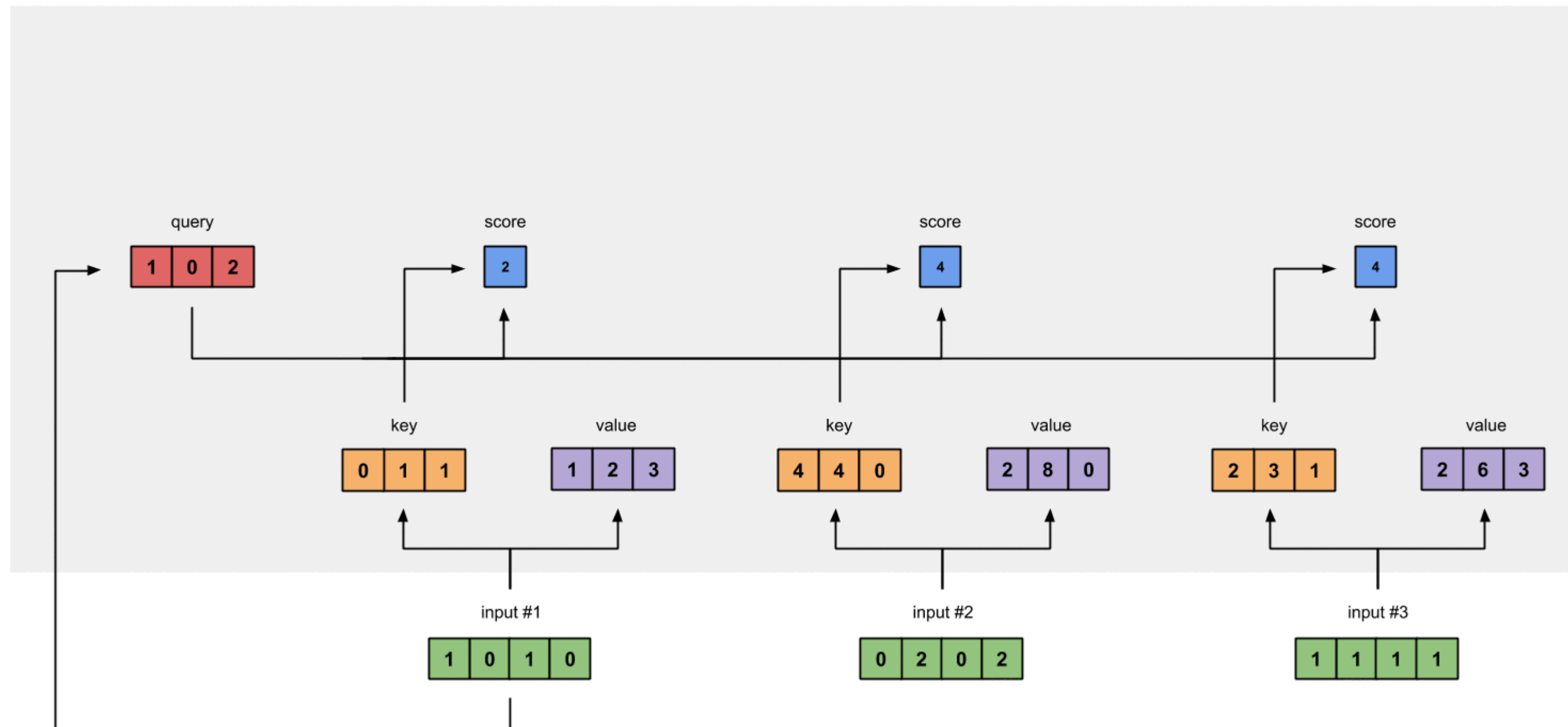
Self-attention



# Self-Attention Network

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

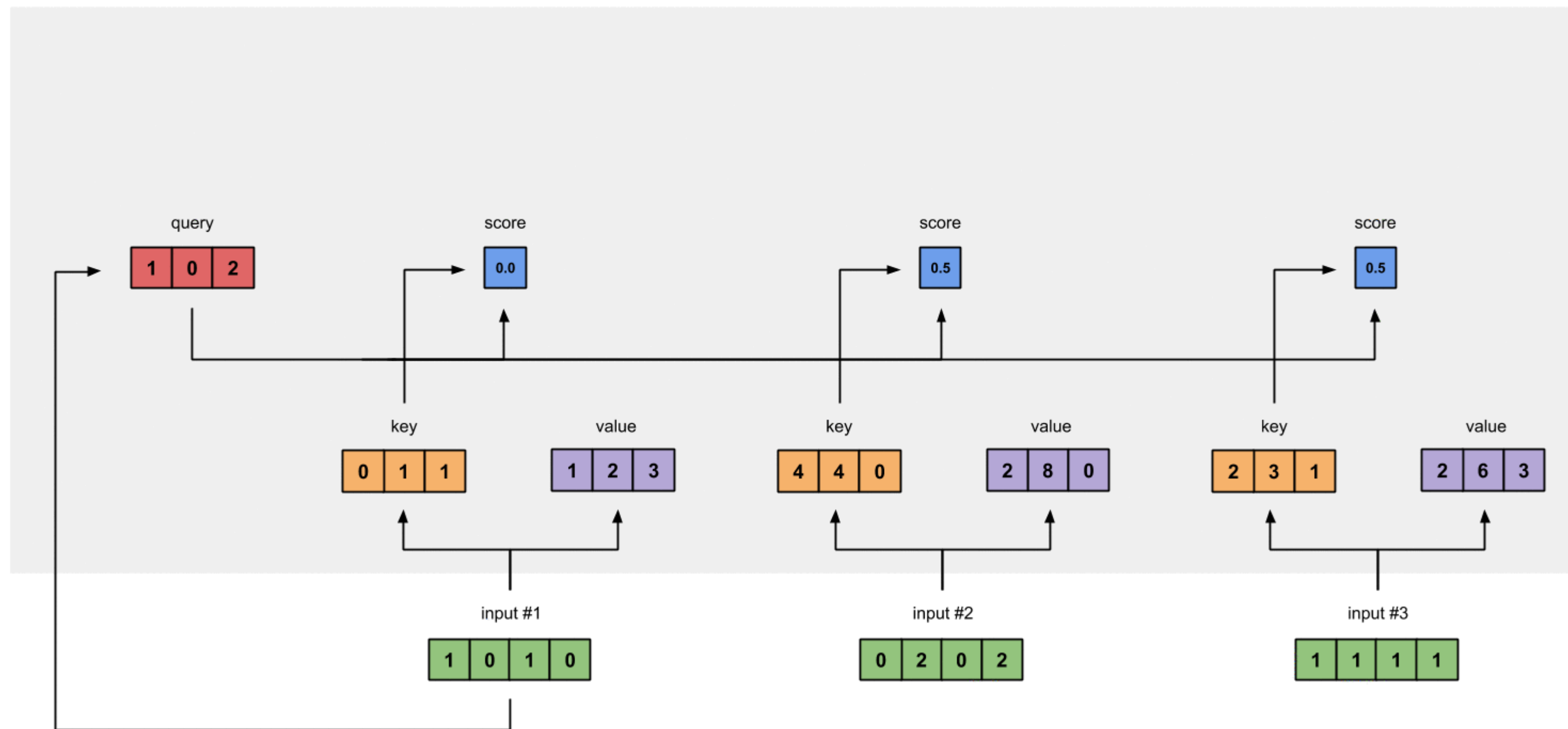
Self-attention



# Self-Attention Network

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

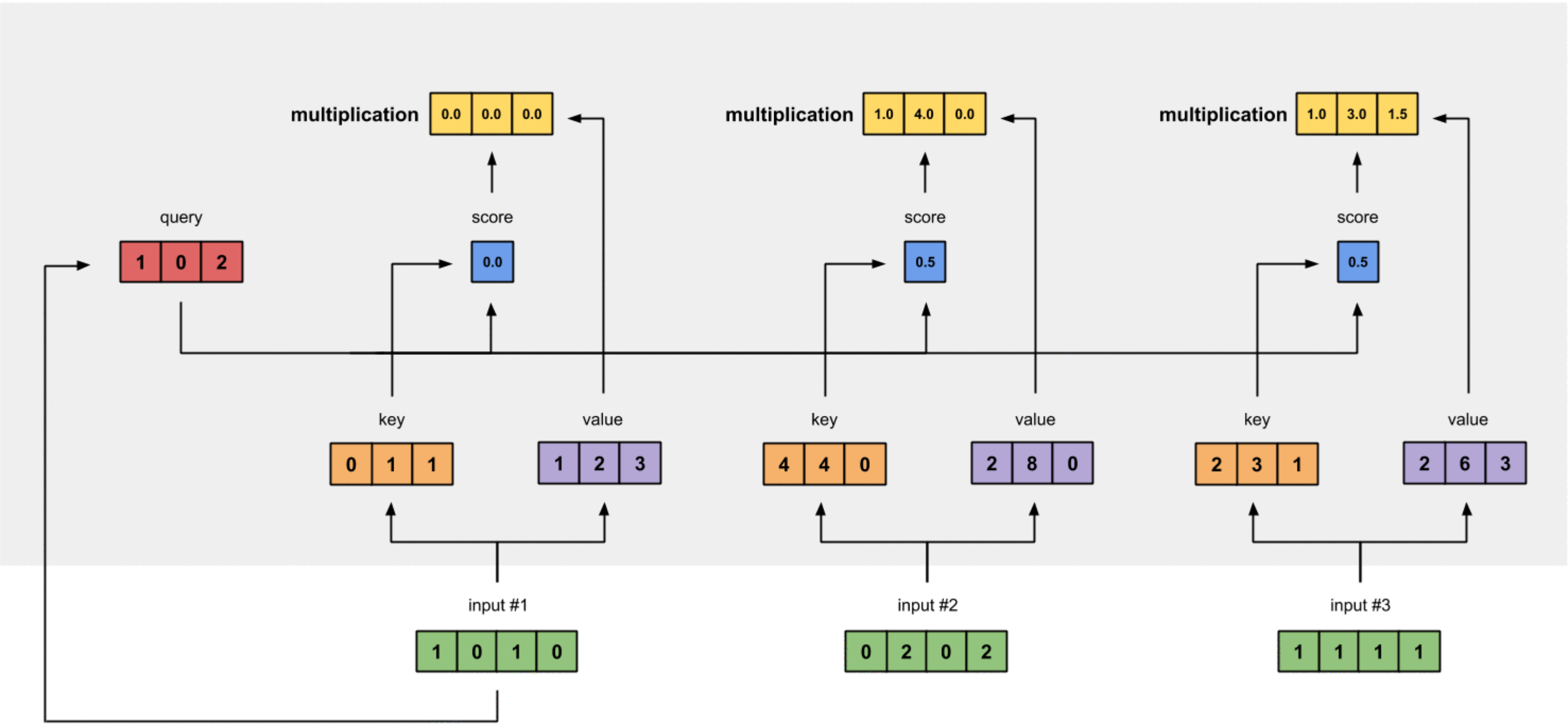
Self-attention



# Self-Attention Network

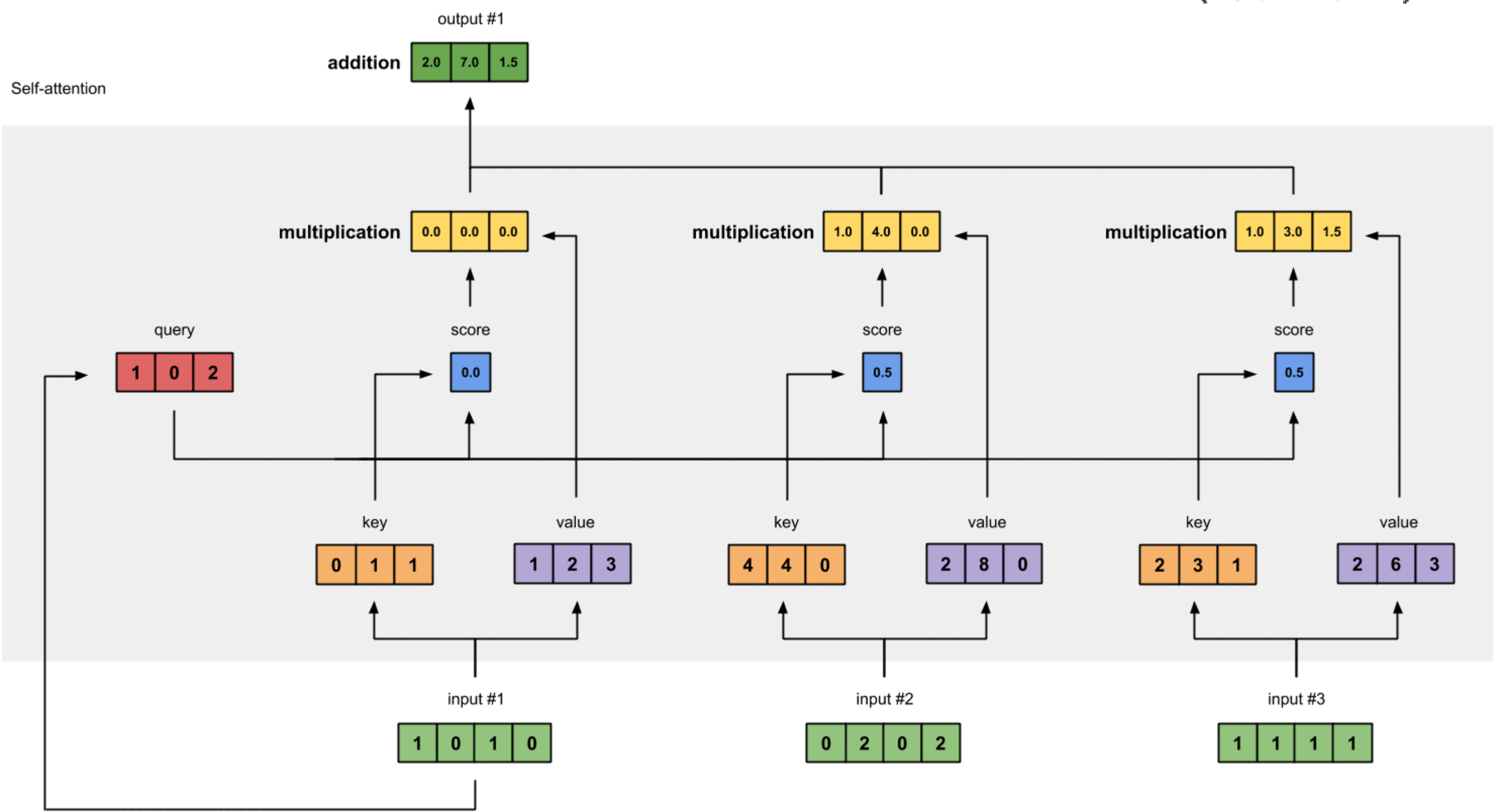
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Self-attention

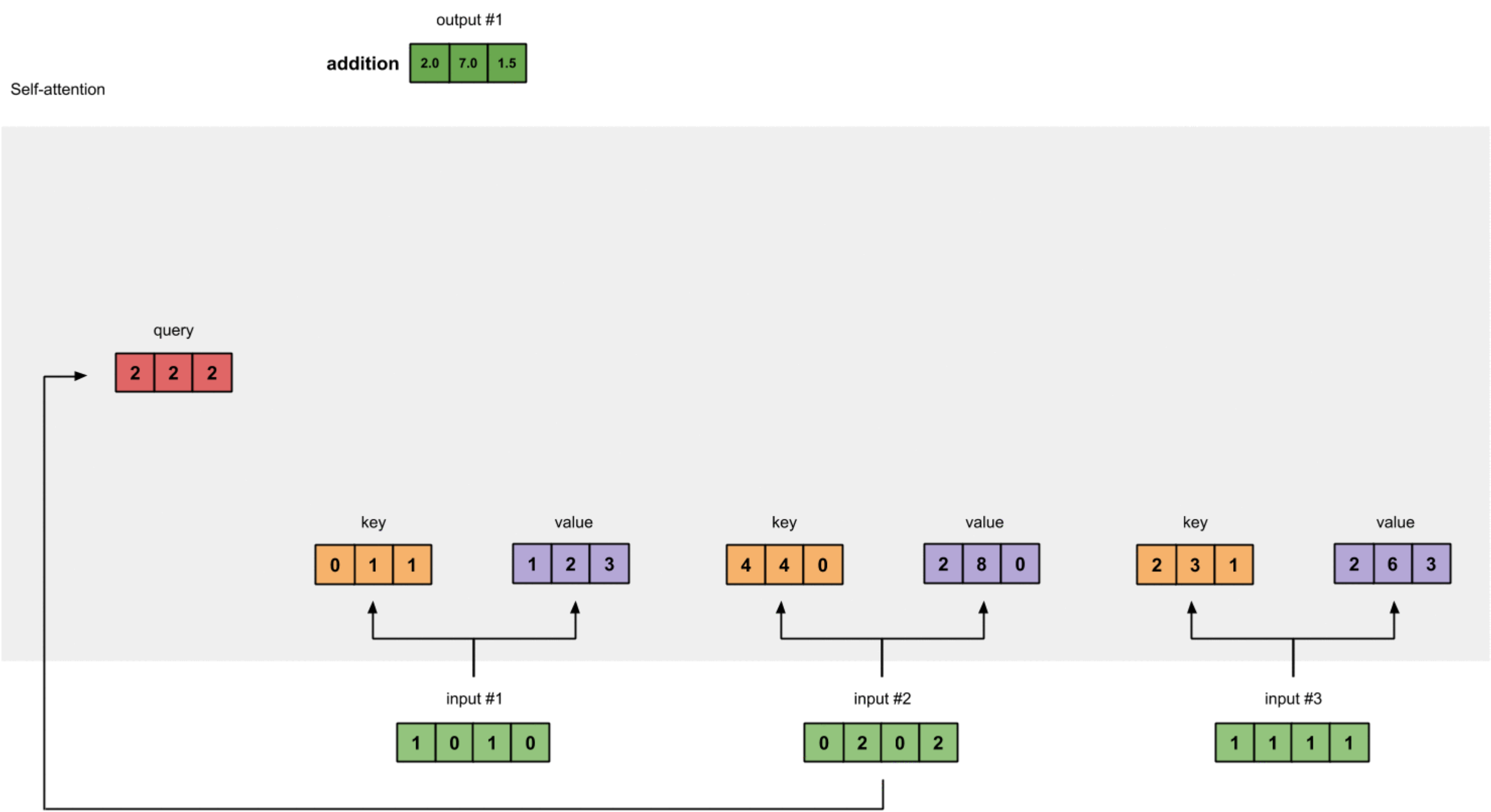


# Self-Attention Network

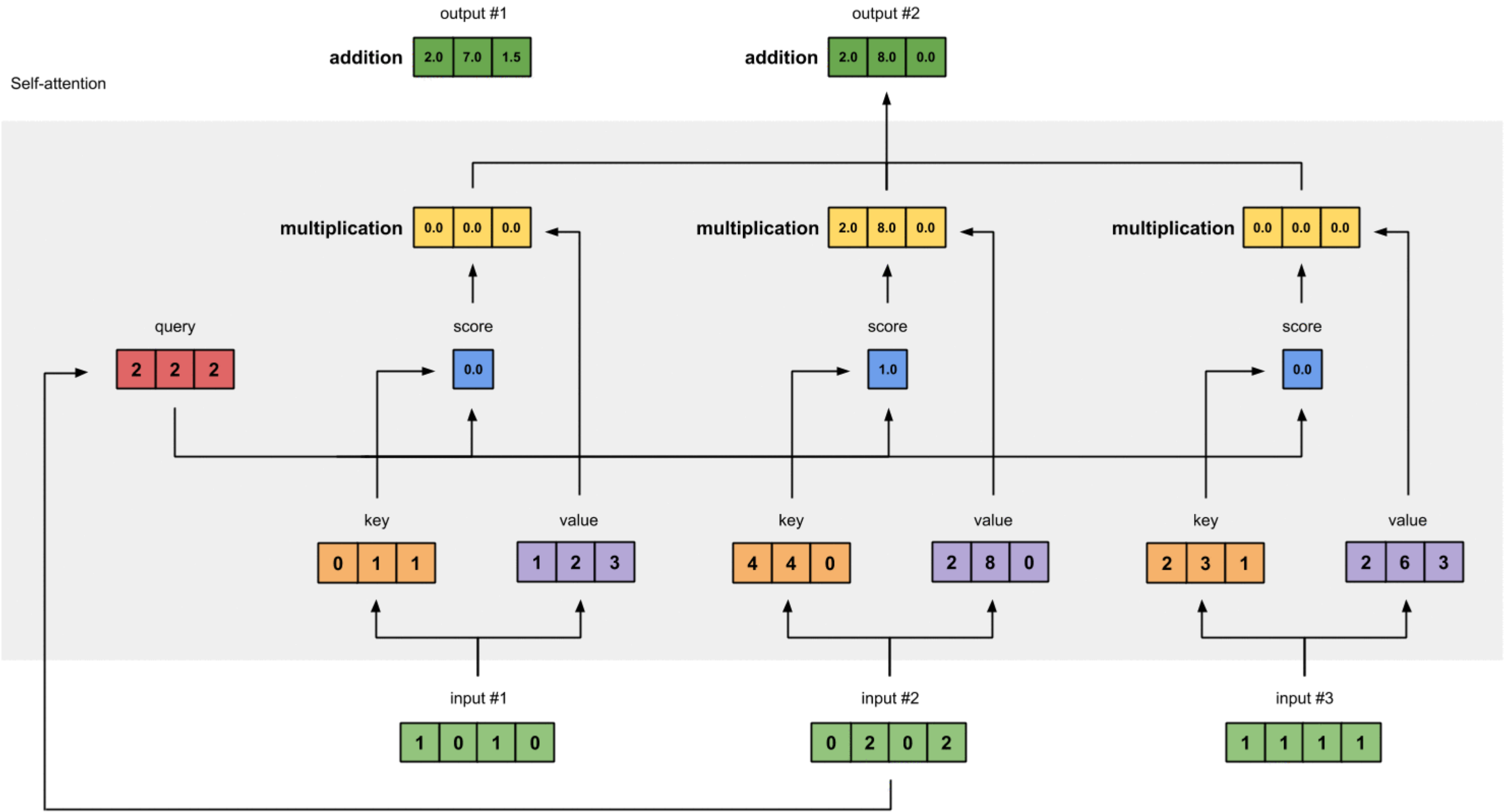
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



# Self-Attention Network

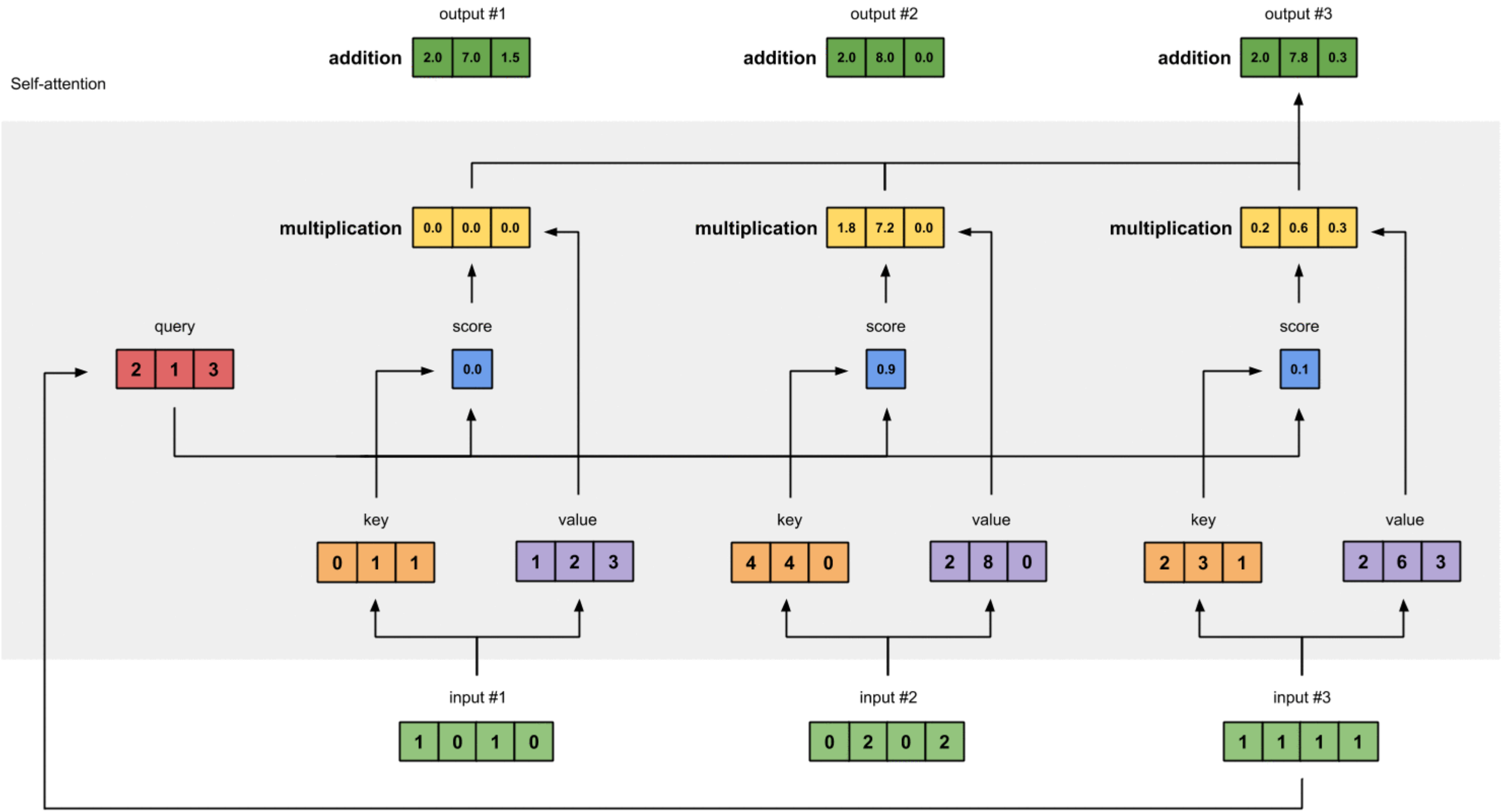


# Self-Attention Network

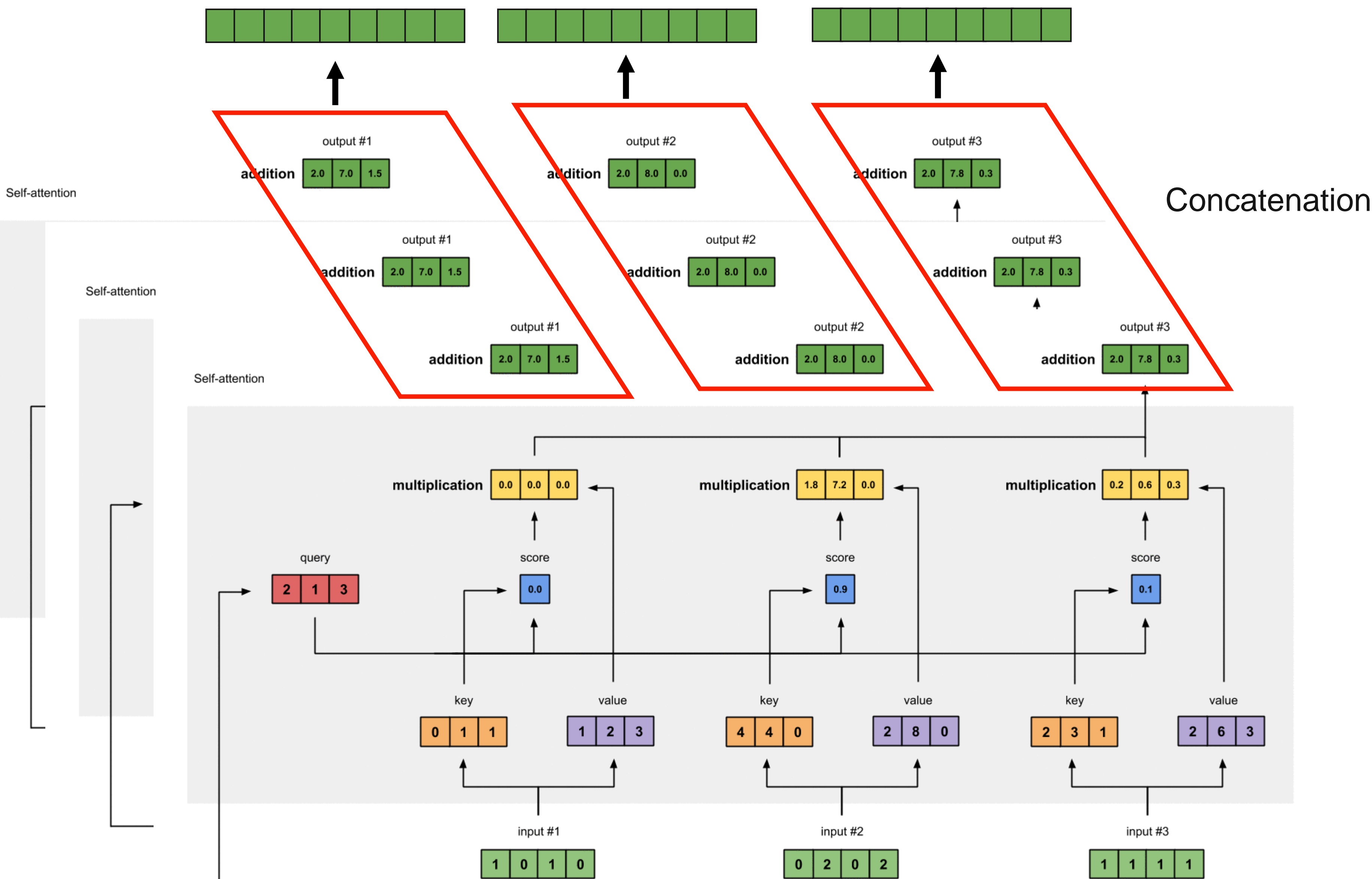




# Self-Attention Network



# 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.

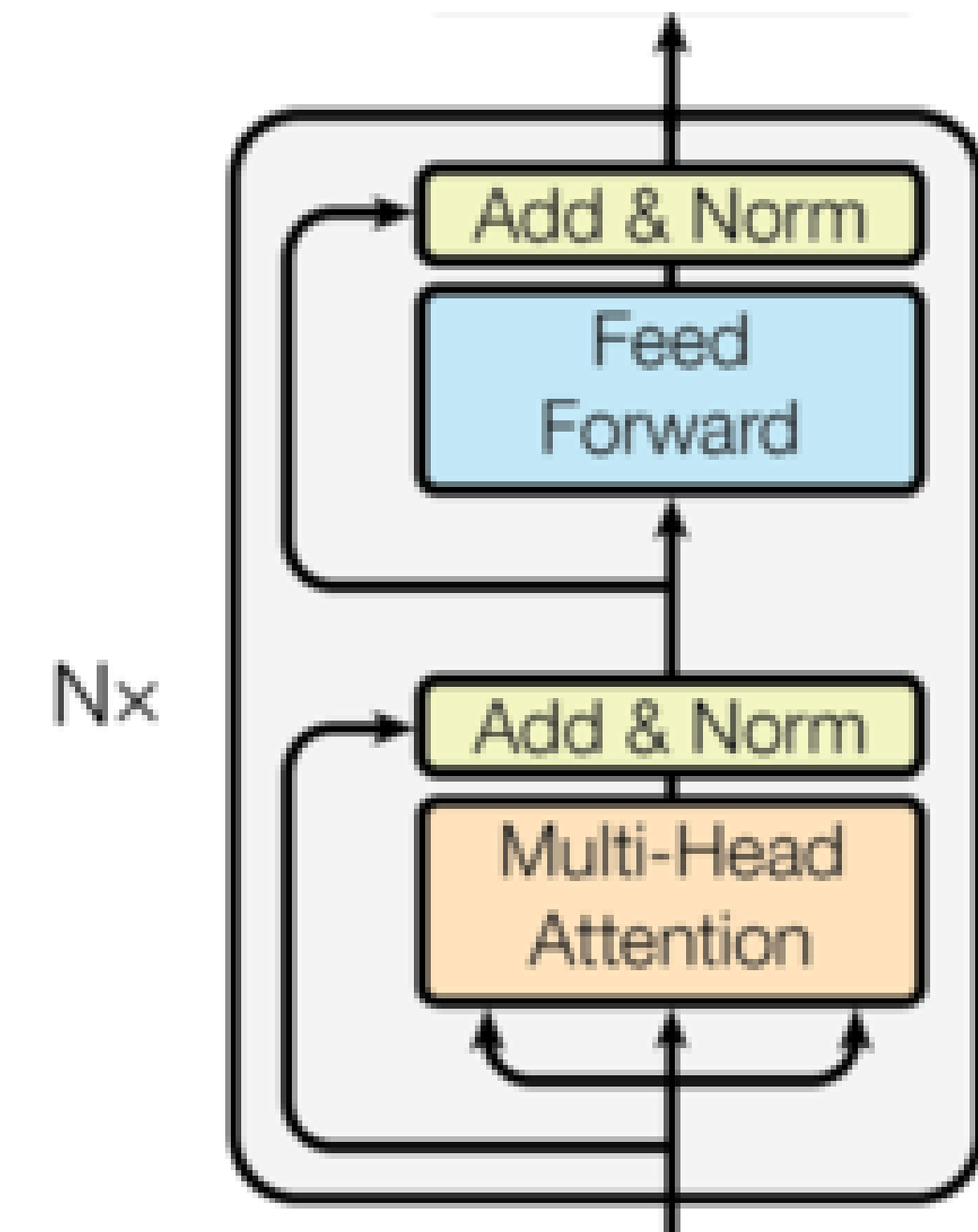
# Encoder : Add & Norm

- Residual Connection
- Layer Normalization

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

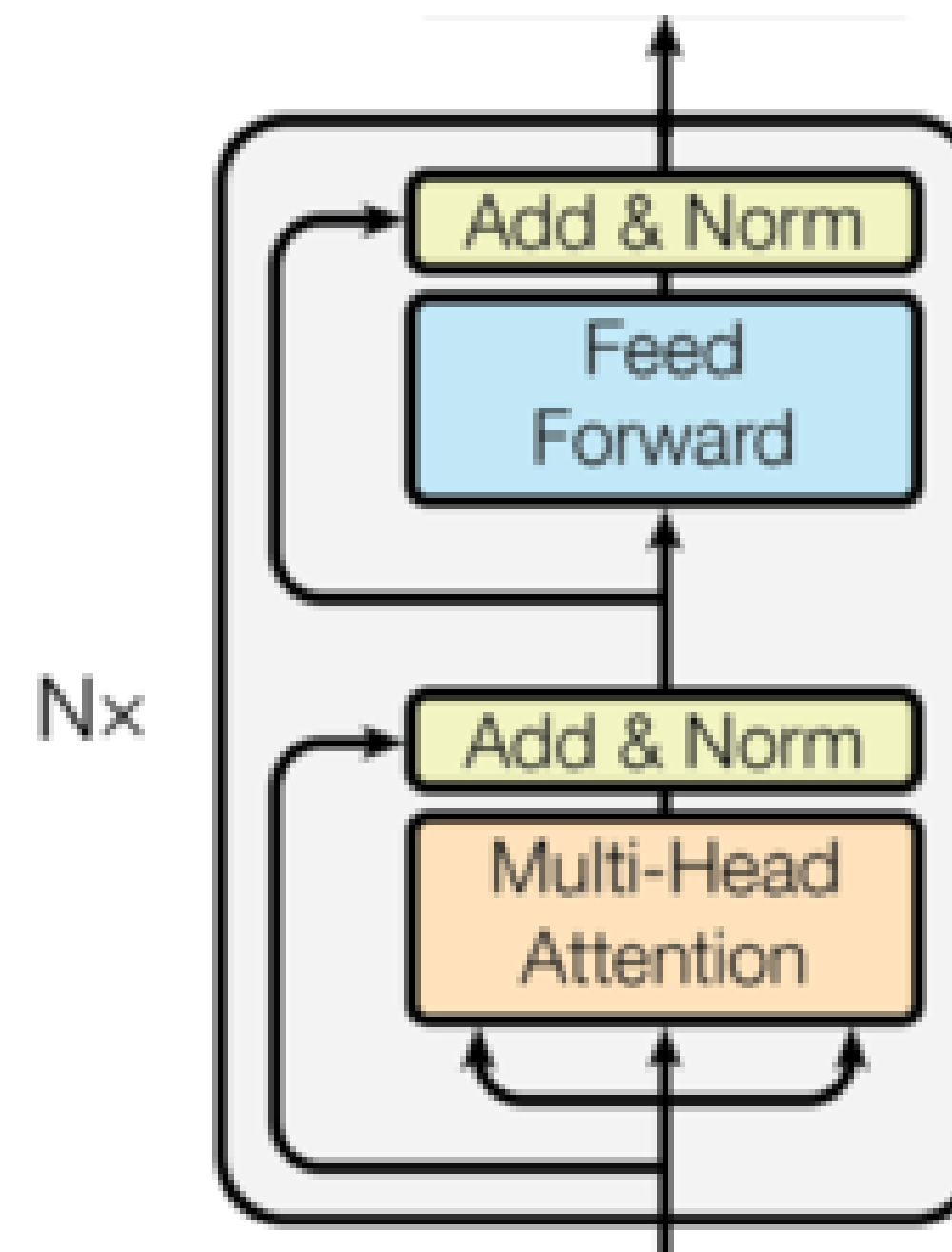
Layer Normalization – to normalize mean and variance of inputs (  $a^l$  ) for a specific layer (  $l$  ), assume that the layer has  $H$  hidden units.

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



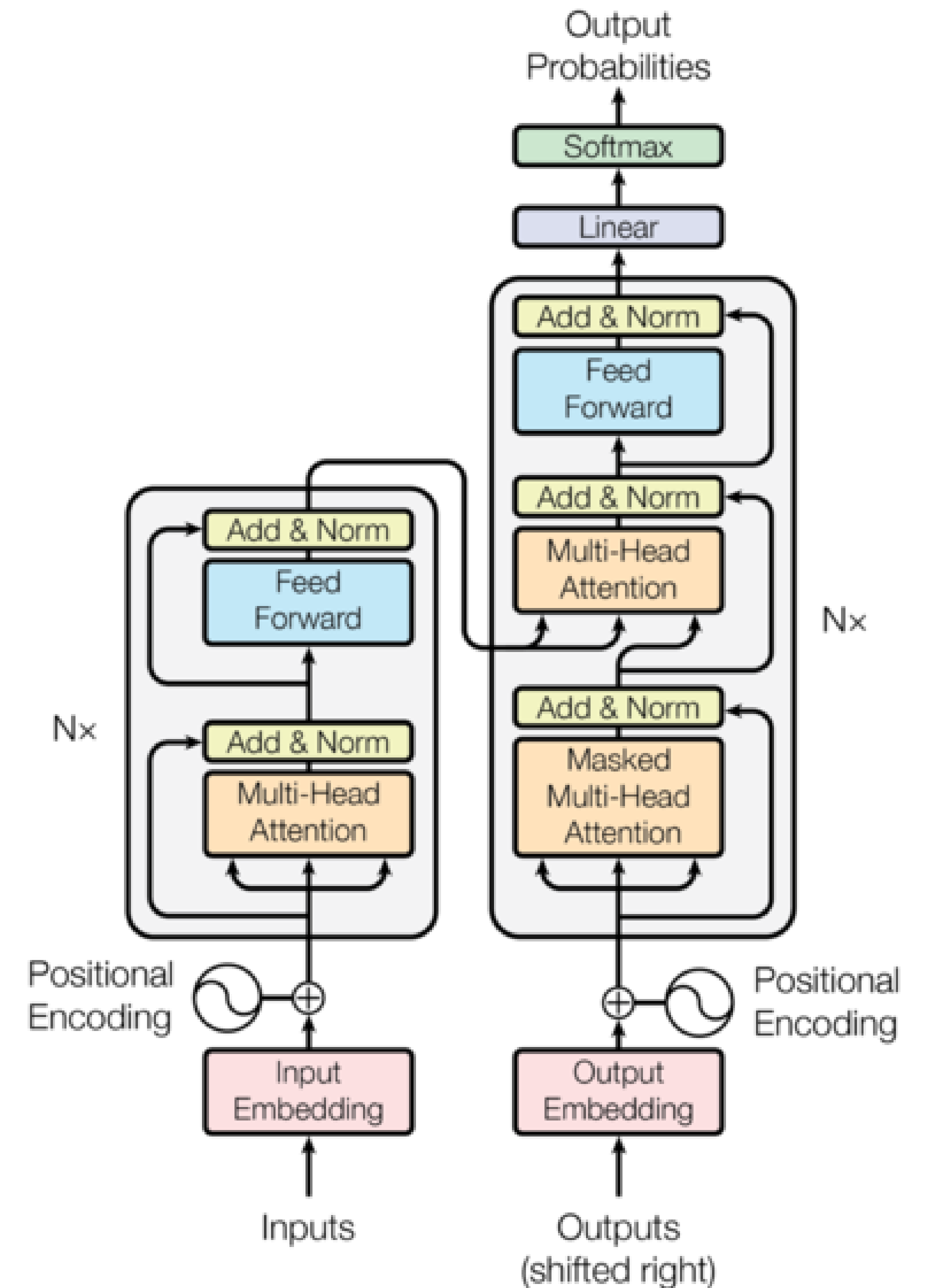
# Encoder : Feed Forward

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



# Transformer : Decoder

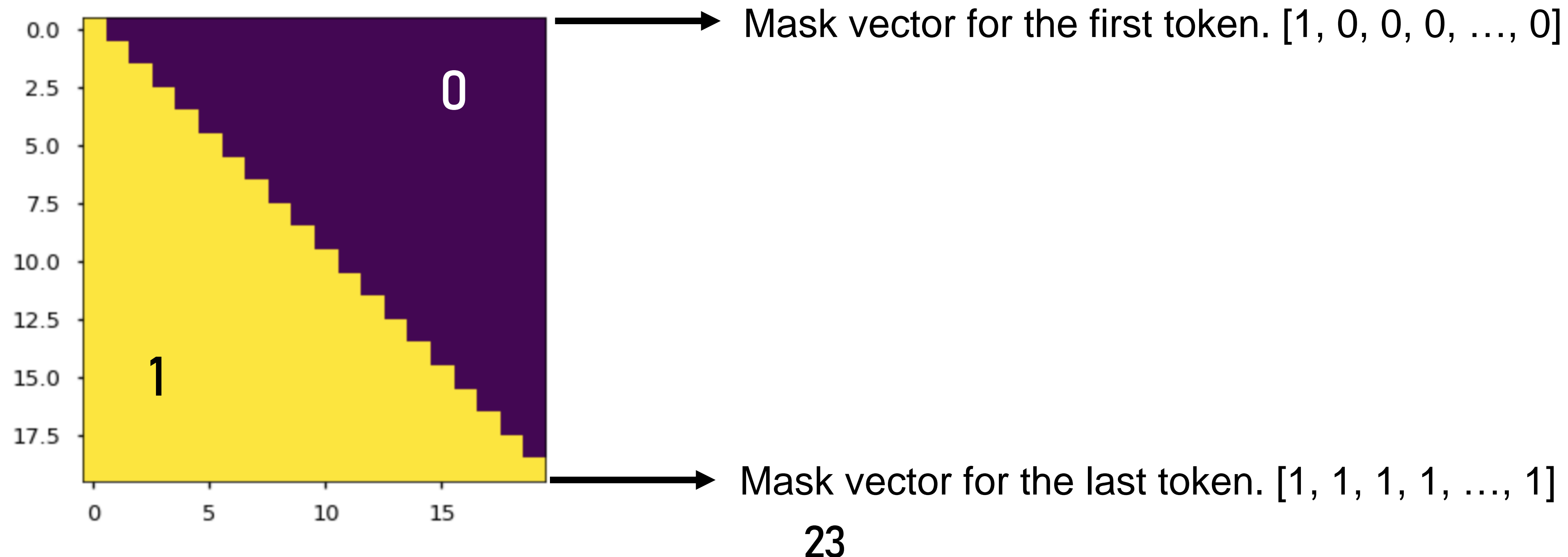
- Auto Regressive Decoding
- Based on Transformer layers



# Decoder : Masked Multi-head Attention

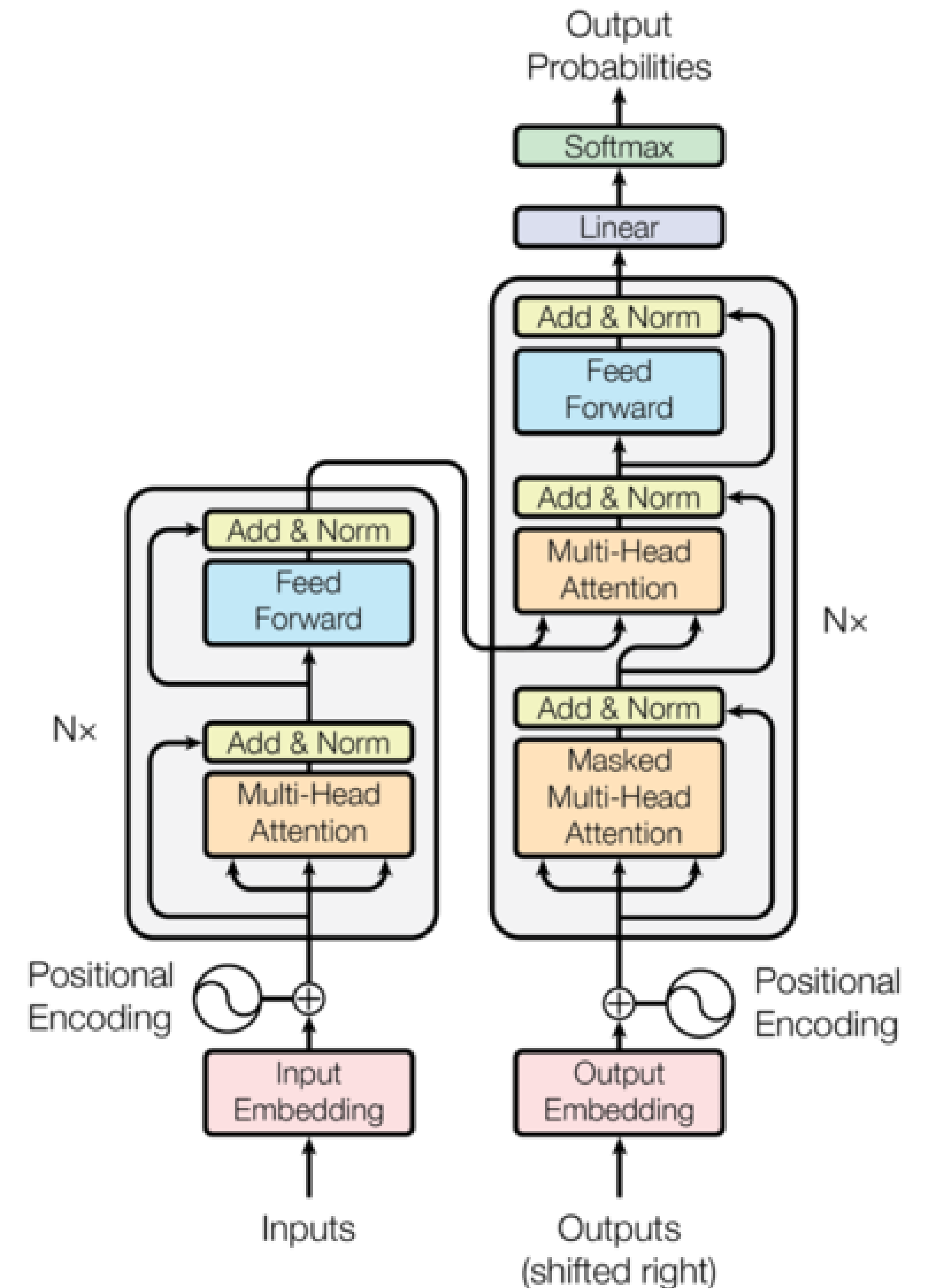
- Auto Regressive Decoding

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.



# Decoder-Encoder Attention

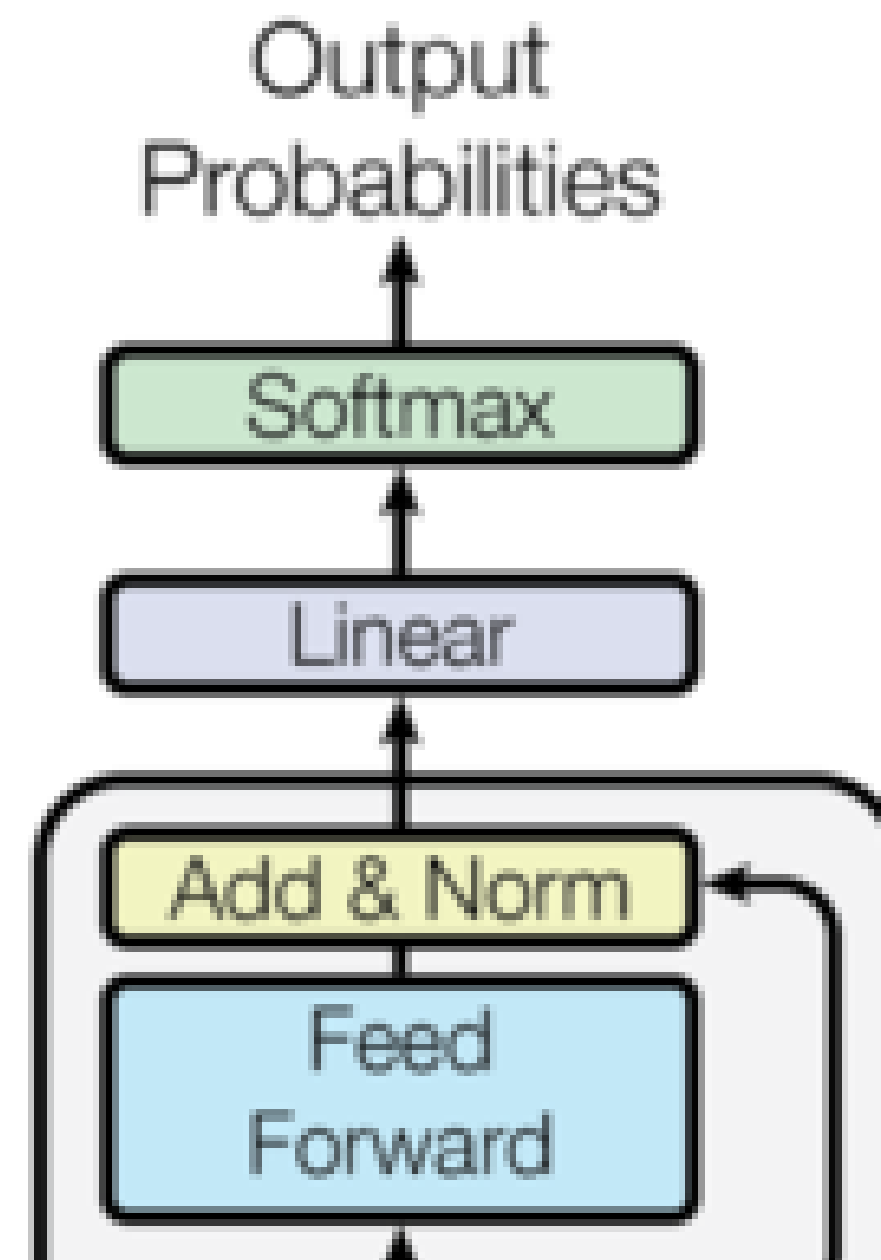
- Query data from the encoder outputs.
- Encoder output (K,V) , Decoder State (Q)





# 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

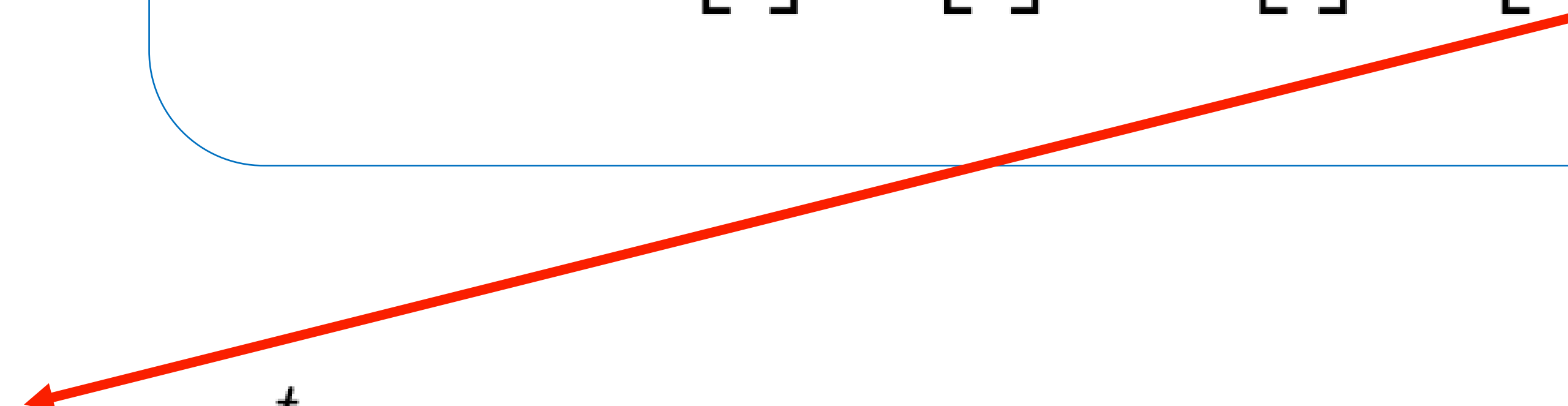
- Label Smoothed Regularization

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

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

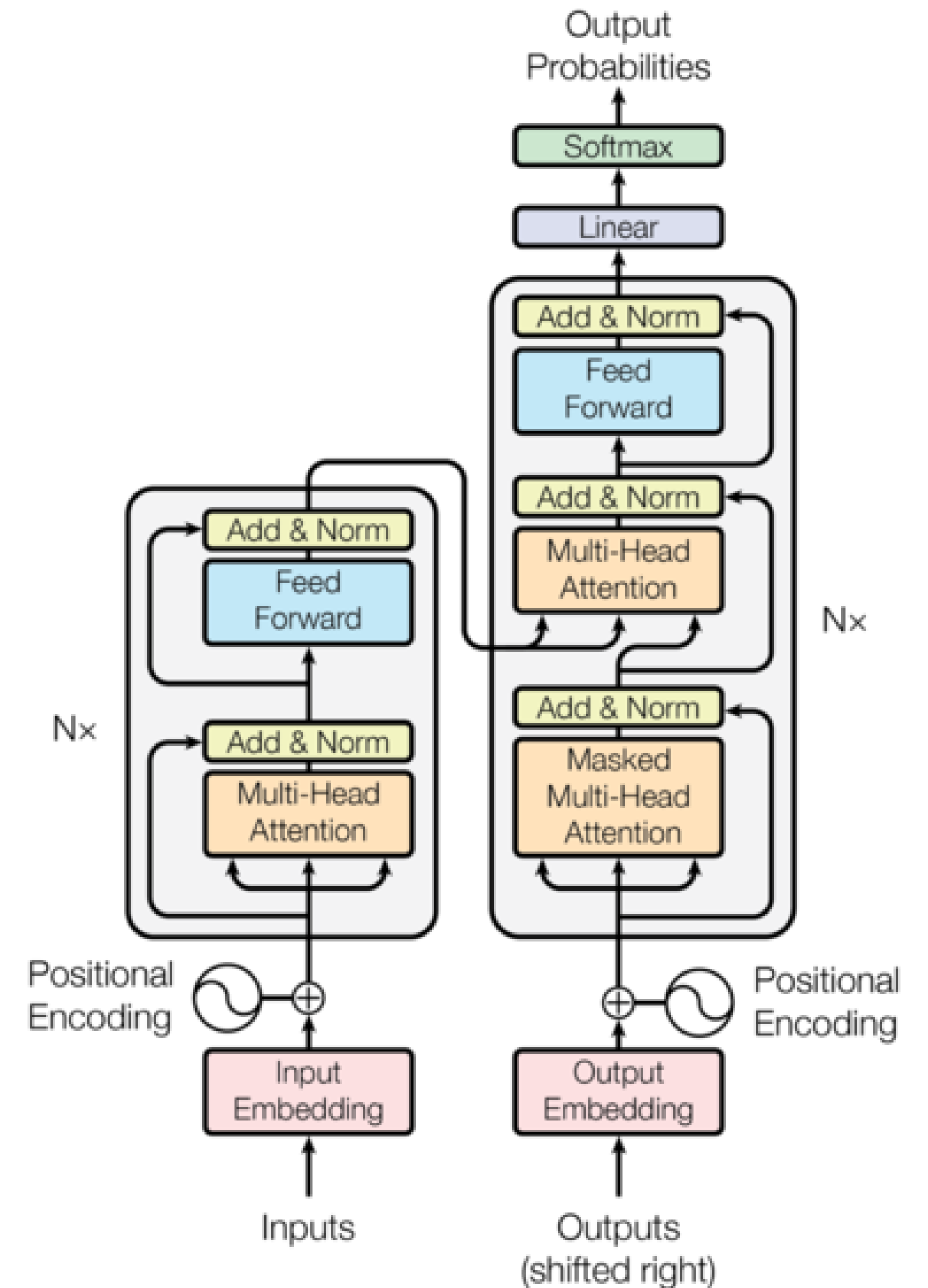
$$= - \sum_{t=1}^T \sum_{j=1}^V \bar{y}_j^t \log \hat{y}_j^t$$


# Decoding : Auto Regressive

- Greedy search
- Beam Search

# 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

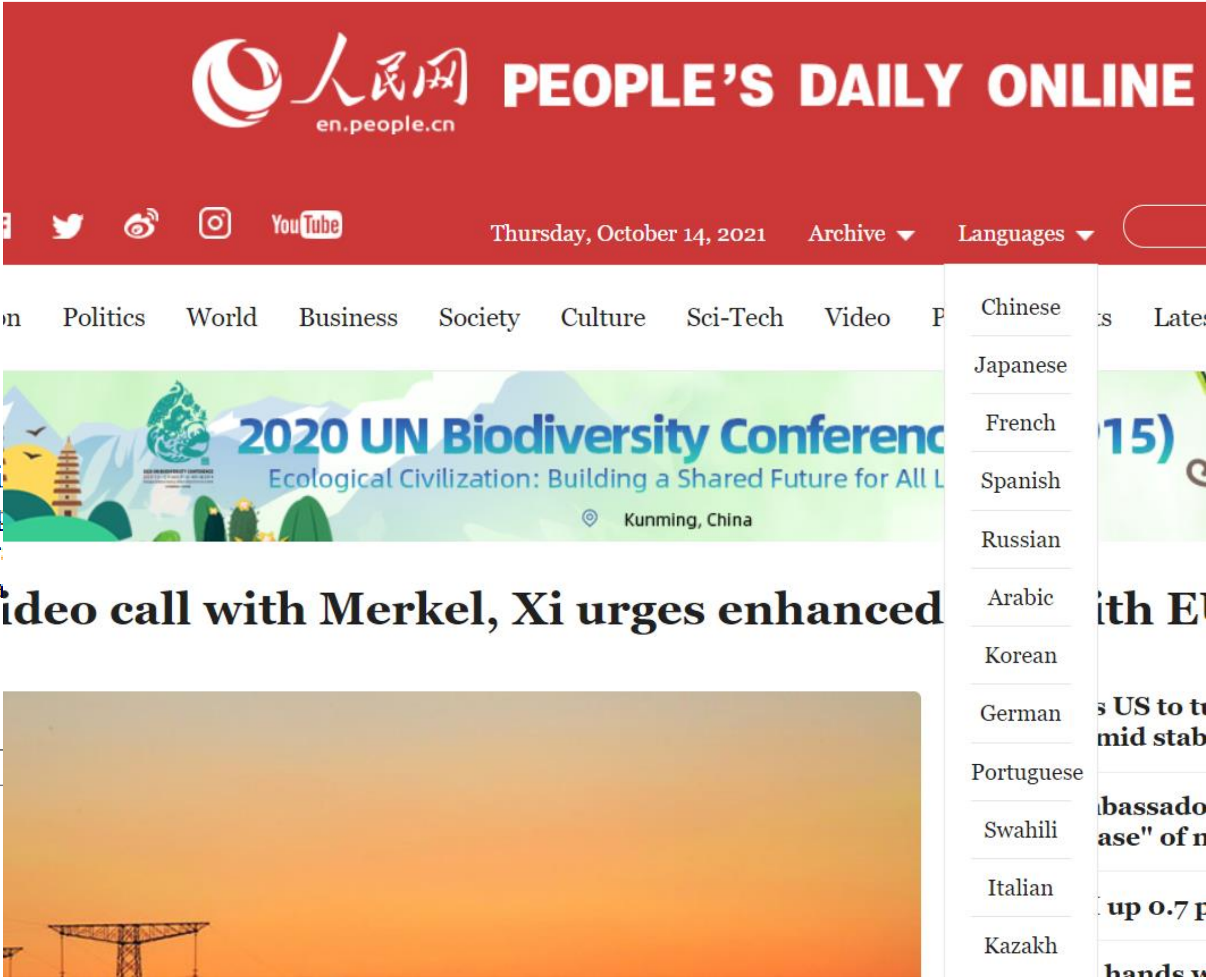
- Data Collection

**OPUS** ... the open parallel corpus

OPUS is a growing collection of translated texts from the web. In the OPUS project we try to convert and align online data, to add linguistic annotation, and to provide the community with a publicly available parallel corpus. The corpus is based on open source products and the corpus is also delivered as an open content package. We used several tools to compile the current collection. All pre-processing is done automatically. No manual corrections have been carried out.

The OPUS collection is growing! Check this page from time to time to see new data arriving ... Contributions are very welcome! Please contact <jorg.tiedemann@helsinki.fi >

Search & download resources:     
[show all versions](#)





# 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



Train 90%

Valid 5% Test 5%

# Additional Steps for NMT

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

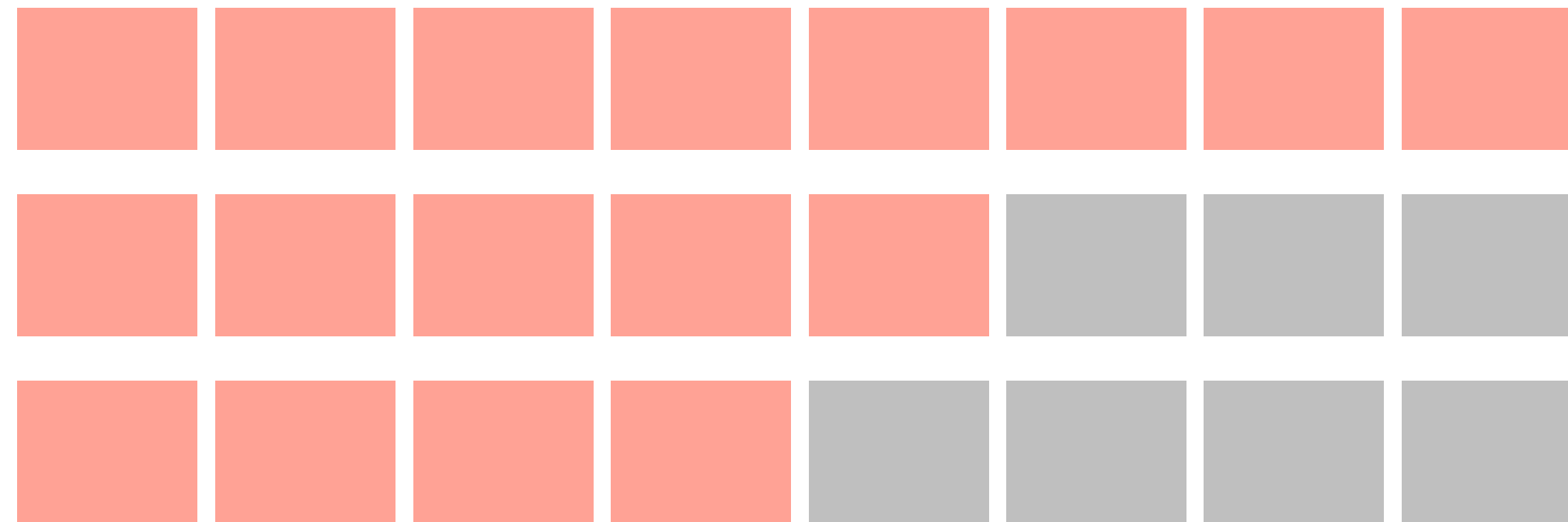
ซีรีส์ จีน เรื่อง ดาบมังกรหยก

ซีรีส์ 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

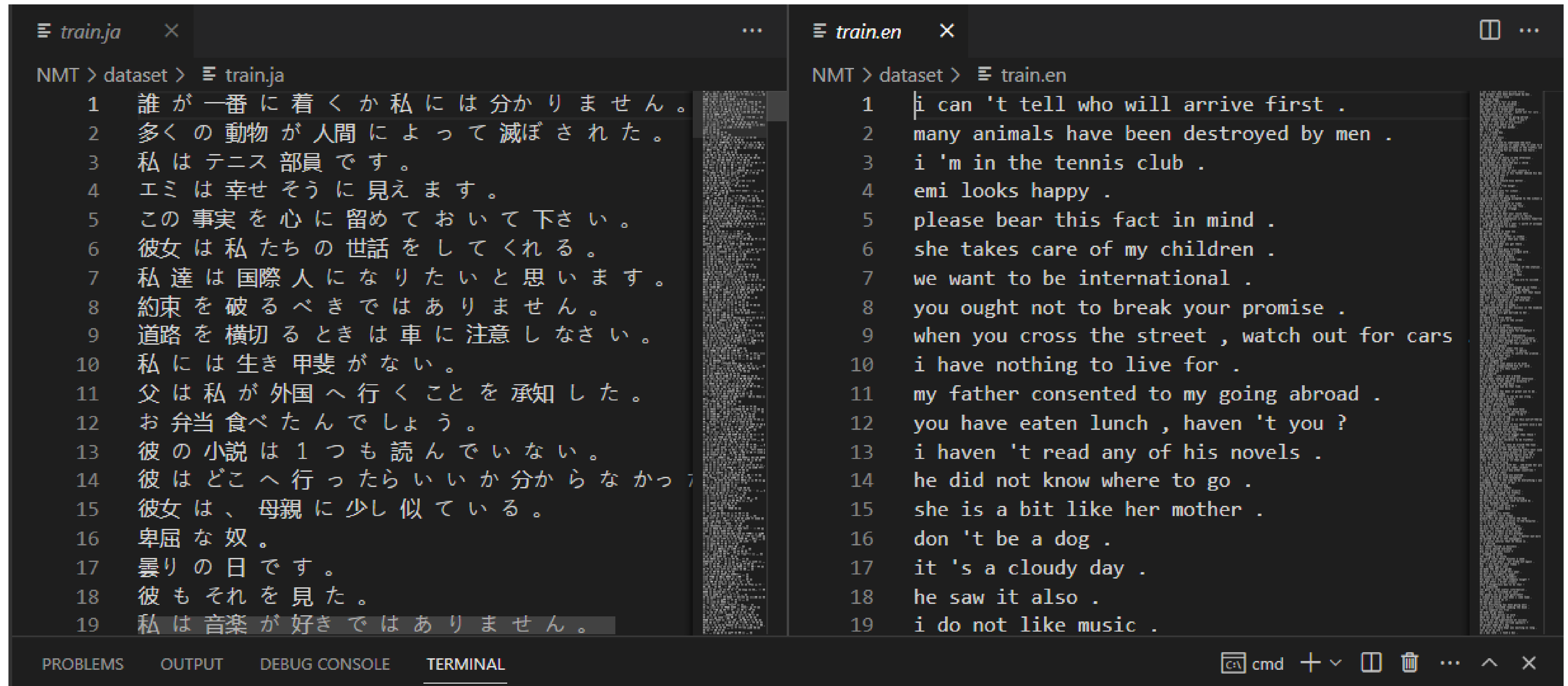
- **Data Collection**
- **Data Cleansing and Tokenization**
- **Split Train / Validation / Test**
- **Additional Preprocessing Steps for NMT**
  - Subword Preparation
  - Padding and Binarizing

# Training Transformer Model

# Preparation Steps

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

# Bilingual Data



The image shows a code editor with two tabs: `train.ja` and `train.en`. The `train.ja` tab contains Japanese text, and the `train.en` tab contains the corresponding English translations. The text is numbered from 1 to 19. The editor has a dark theme and a sidebar on the right showing a file explorer. The bottom of the editor has a terminal panel with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab is active, showing a command prompt with the text `cmd`.

```
NMT > dataset > train.ja
1 誰が一番に着くか私には分かりません。
2 多くの動物が人間によって滅ぼされた。
3 私はテニス部員です。
4 エミは幸せそうに見えます。
5 この事実を心に留めておいて下さい。
6 彼女は私たちの世話をしてくれる。
7 私達は国際人になりたいと思います。
8 約束を破るべきではありません。
9 道路を横切るときは車に注意下さい。
10 私には生き甲斐がない。
11 父は私が外国へ行くことを承知した。
12 お弁当食べたんでしょう。
13 彼の小説は1つも読んでいない。
14 彼はどこへ行ったらいいか分からなかった。
15 彼女は、母親に少し似ている。
16 卑屈な奴。
17 曇りの日です。
18 彼もそれを見た。
19 私は音楽が好きではありません。

NMT > dataset > train.en
1 i can 't tell who will arrive first .
2 many animals have been destroyed by men .
3 i 'm in the tennis club .
4 emi looks happy .
5 please bear this fact in mind .
6 she takes care of my children .
7 we want to be international .
8 you ought not to break your promise .
9 when you cross the street , watch out for cars
10 i have nothing to live for .
11 my father consented to my going abroad .
12 you have eaten lunch , haven 't you ?
13 i haven 't read any of his novels .
14 he did not know where to go .
15 she is a bit like her mother .
16 don 't be a dog .
17 it 's a cloudy day .
18 he saw it also .
19 i do not like music .
```



# Dictionary (Word list)

The screenshot displays the VS Code interface with two editor windows open side-by-side, showing the contents of two vocabulary files.

**Left Editor Window: `train.ja.vocab.4k`**

Path: NMT > dataset > train.ja.vocab.4k

Index	Word
1	<unk>
2	<s>
3	</s>
4	。
5	は
6	い
7	に
8	た
9	を
10	の
11	て
12	で
13	な
14	が
15	彼
16	し
17	る
18	私
19	す

**Right Editor Window: `train.en.vocab.4k`**

Path: NMT > dataset > train.en.vocab.4k

Index	Word
1	<unk>
2	<s>
3	</s>
4	.
5	the
6	i
7	to
8	you
9	is
10	he
11	a
12	?
13	in
14	it
15	of
16	she
17	for
18	my
19	have

The Explorer on the left shows the project structure with the `train.en.vocab.4k` file selected under the `dataset` directory.

# 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 X
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 > ...
32 def make_model(src_vocab, tgt_vocab, N=6,
33                d_model=512, d_ff=2048, h=8, dropout=0.1):
34     "Helper: Construct a model from hyperparameters."
35     print("Multihead...")
36     attn = MultiHeadedAttention(h, d_model)
37
38     print("Position...")
39     ff = PositionwiseFeedForward(d_model, d_ff, dropout)
40
41     print("PositionEncoding...")
42     position = PositionalEncoding(d_model, dropout)
43
44     print("Build Encoder-Decoder")
45     model = EncoderDecoder(
46         Encoder(EncoderLayer(d_model, c(attn), c(ff), dropout), N),
47         Decoder(d_model, h, d_ff, N, dropout),
48         nn.Sequential(Embeddings(d_model, src_vocab), c(position)),
49         nn.Sequential(Embeddings(d_model, tgt_vocab), c(position)),
50         Generator(d_model, tgt_vocab))
51
52     # This was important from their code.
53     # Initialize parameters with Glorot / fan_avg.
54     print("Init Params...")
55     for p in model.parameters():
56         if p.dim() > 1:
57             nn.init.xavier_uniform_(p)
```

# Transformer Model

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

```
encoder.py 2 X
NMT > encoder.py > Encoder

5 class EncoderLayer(nn.Module):
6     "Encoder is made up of self-attn and feed forward (defined below)"
7     def __init__(self, size, self_attn, feed_forward, dropout):
8         super(EncoderLayer, self).__init__()
9         self.self_attn = self_attn
10        self.feed_forward = feed_forward
11        self.sublayer = nn.ModuleList([SublayerConnection(size, dropout)])
12        self.size = size
13
14    def forward(self, x, mask):
15        "Follow Figure 1 (left) for connections."
16        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, mask))
17        return self.sublayer[1](x, self.feed_forward)
18
19 class Encoder(nn.Module):
20     "Core encoder is a stack of N layers"
21     def __init__(self, layer, N):
22         super(Encoder, self).__init__()
23         self.layers = clones(layer, N)
24         self.norm = LayerNorm(layer.size)
25
26     def forward(self, x, mask):
27         "Pass the input (and mask) through each layer in turn."
28         for layer in self.layers:
29             x = layer(x, mask)
30         return self.norm(x)
```

# 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 X
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
170
```

# 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()`

```
learn_word_alignment.py  main_loop.py M X
NMT > main_loop.py > run_epoch
191
192 def run_epoch(data_iter, model, loss_compute, optimizer, epoch=1):
193     global VALID_INTERVAL
194     global update_count
195     "Standard Training and Logging Function"
196     start = time.time()
197     total_tokens = 0
198     total_loss = 0
199     tokens = 0
200     print("Start Epoch...")
201     for i, batch in enumerate(data_iter):
202         try:
203             out = model.forward(batch.src, batch.trg,
204                                 batch.src_mask, batch.trg_mask)
205             loss = loss_compute(out, batch.trg_y, batch.ntokens)
206
207             loss.backward()
208             optimizer.step()
209             optimizer.zero_grad()
210
```

# Training over Dataset

main\_loop.py 4 X

NMT > main\_loop.py > make\_model

```
371
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 X

NMT > main\_loop.py > make\_model

```
240
241 def greedy_decode(model, src, src_mask, max_len, start_symbol, end_symbol):
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(torch.ones(1, 1).type_as(src.data)))
248         prob = model.generator(out[:, -1])
249         _, next_word = torch.max(prob, dim = 1)
250         next_word = next_word.data[0]
251         if next_word == end_symbol:
252             break
253         ys = torch.cat([ys,
254                         torch.ones(1, 1).type_as(src.data).fill_(end_symbol)])
255     return ys
256
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 X

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](mailto:peerachet.porkaew@gmail.com)
- Subject : **2023 - Transformer Assignment (your student ID)**

**Thank you**