Sequence-to-Sequence Modelling, Encoders, and Decoders

Prepared by **Alfan F. Wicaksono**

Faculty of Computer Science, Universitas Indonesia

**Materi merupakan gabungan dari banyak sumber & hasil kreasi pribadi. Mohon perhatikan sumber-sumber yang dirujuk.*

Credits

- Universiteit Van Amsterdam's DL Notebooks
  - https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial6/Transformers_and_MHAttention.html
- Attention is All You Need (Vaswani, 2017)
  - https://arxiv.org/abs/1706.03762
- Peter Bloem's Blog
  - https://peterbloem.nl/blog/transformers
- Jay Alammar's Blog
  - http://jalammar.github.io/illustrated-gpt2/
  - https://jalammar.github.io/illustrated-transformer/

Some Backgrounds: Unsqueezing a Tensor

```
v = torch.tensor([1, 2, 3])

print(v.shape)
#torch.Size([3])

#tambah dimensi sebagai axis 0
v2 = v.unsqueeze(0)

print(v2)
#tensor([[1, 2, 3]])

print(v2.shape)
#torch.Size([1, 3])
```

```
v = torch.tensor([1, 2, 3])

print(v.shape)
#torch.Size([3])

#tambah dimensi sebagai axis 1
v2 = v.unsqueeze(1)

print(v2)
tensor([[1],
        [2],
        [3]])

print(v2.shape)
#torch.Size([2, 1])
```

Some Backgrounds: Reshaping a Tensor

```
v = torch.tensor([[[1,2,1,2], [3,4,3,4]],
                  [[5,6,5,6], [7,8,7,8]],
                  [[2,1,2,1], [4,3,4,3]]])

#ada yang tahu isi masing-masing variable di bawah?
batch_size, seq_len, em_dim = v.size()
    3                 2          4

v2 = v.reshape(batch_size, seq_len * em_dim)
print(v2)
#tensor([[1, 2, 1, 2, 3, 4, 3, 4],
#        [5, 6, 5, 6, 7, 8, 7, 8],
#        [2, 1, 2, 1, 4, 3, 4, 3]])

print(v2.shape)
#torch.Size([3, 8])
```

bisa di flatten kalo dikali semua:

batch_size * seq_len * em_dim

Some Backgrounds: Reshaping a Tensor

```
v = torch.tensor([[[1,2,1,2], [3,4,3,4]],
                  [[5,6,5,6], [7,8,7,8]],
                  [[2,1,2,1], [4,3,4,3]]])

#ada yang tahu isi masing-masing variable di bawah?
batch_size, seq_len, em_dim = v.size()

v2 = v.reshape(batch_size, seq_len, 2, 2)
print(v2)

print(v2.shape)
#torch.Size([3, 2, 2, 2])
```

```
tensor([[[[1, 2],
          [1, 2]],

         [[3, 4],
          [3, 4]]],


        [[[5, 6],
          [5, 6]],

         [[7, 8],
          [7, 8]]],


        [[[2, 1],
          [2, 1]],

         [[4, 3],
          [4, 3]]]])
```

Some Backgrounds: Permutation (Transpose)

jadi yang awalnya di embeeding dim, maka dia jadi disequence lengthnya (kalo dilihat, kalo dilihat representasi setiap sequence length, sekarang direpresentasi dengan angka di embeeding dim original).

```
v = torch.tensor([[[1,2,1,2], [3,4,3,4]],
                  [[5,6,5,6], [7,8,7,8]],
                  [[2,1,2,1], [4,3,4,3]]])
```

embeeding dim = 4
seq len = 2

dibalik jadi 4, 2

```
#transpose axis 2 dan 1
v2 = v.permute(0, 2, 1)
print(v2)

print(v2.shape)
#torch.Size([3, 4, 2])
#[batch_size, em_dim, seq_len]
```

sequence length dan embeeding dimension nya dituker

```
tensor([[[1, 3],
         [2, 4],
         [1, 3],
         [2, 4]],

        [[5, 7],
         [6, 8],
         [5, 7],
         [6, 8]],

        [[2, 4],
         [1, 3],
         [2, 4],
         [1, 3]]])
```

seq len kedua from original)

seq len pertama

Some Backgrounds: Permutation (Transpose)

```
v = torch.tensor([[[1,2,1,2], [3,4,3,4]],
                  [[5,6,5,6], [7,8,7,8]],
                  [[2,1,2,1], [4,3,4,3]]])


#apa yang dilakukan?
v2 = v.permute(2, 1, 0)
print(v2)

print(v2.shape)
#torch.Size([4, 2, 3])
#[seq_len, em_dim, batch_size]
```

posisi batch dipindah ke terakhir

jadi 4, 2, 3

```
tensor([[[1, 5, 2],
         [3, 7, 4]],

        [[2, 6, 1],
         [4, 8, 3]],

        [[1, 5, 2],
         [3, 7, 4]],

        [[2, 6, 1],
         [4, 8, 3]]])
```

Some Backgrounds: Masking a Tensor

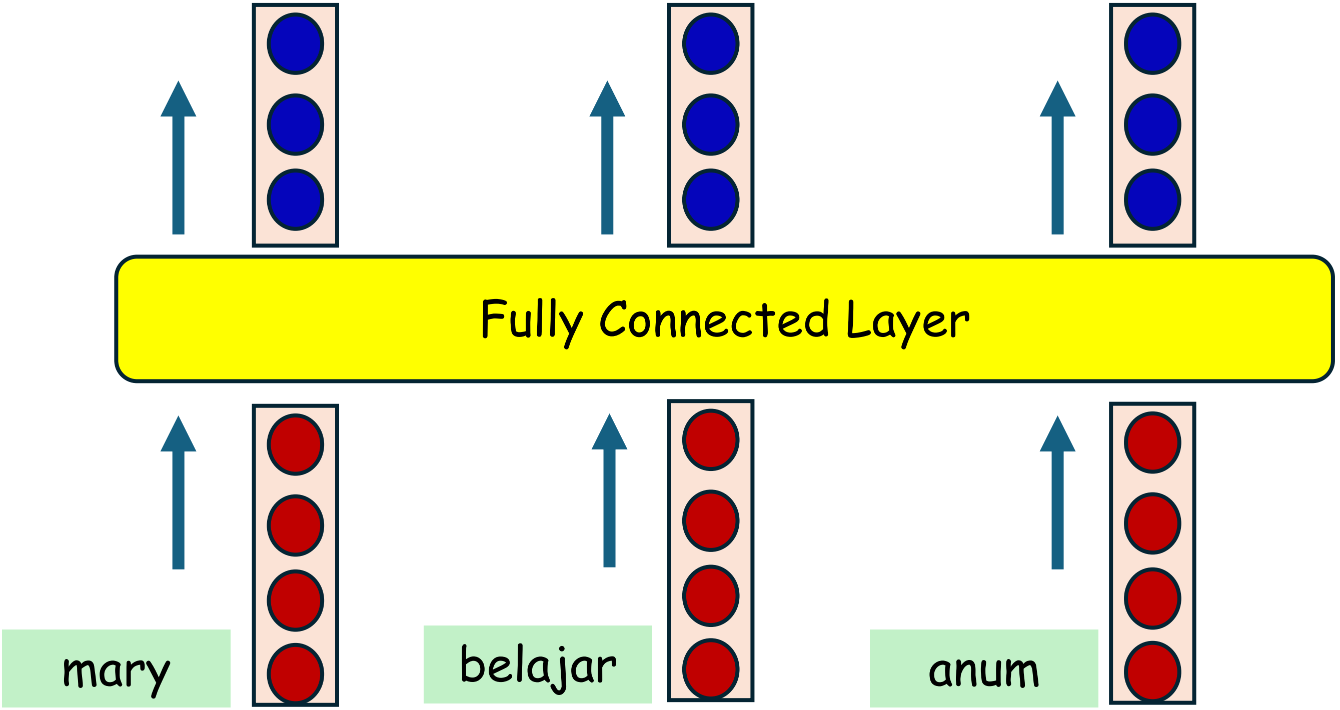
```
attn_logits = torch.tensor([[2., 3., 4.],
                            [1., 2., 1.],
                            [4., 1., 0.8]])

mask = torch.tensor([1, 1, 0])
mask_2d = torch.tensor([[1, 1, 1],
                        [1, 1, 0],
                        [0, 0, 0]])

print(attn_logits.masked_fill(mask == 0, -9e15))
#tensor([[ 2.0,  3.0, -9.0000e+15],
#         [ 1.0,  2.0, -9.0000e+15],
#         [ 4.0,  1.0, -9.0000e+15]])

print(attn_logits.masked_fill(mask_2d == 0, -9e15))
#tensor([[ 2.0,  3.0,  4.0],
#         [ 1.0,  2.0, -9.0000e+15],
#         [-9.0000e+15, -9.0000e+15, -9.0000e+15]])
```

Some Backgrounds: Time-Distributed Linear Layer

Misal, kita mempunyai sebuah data yang berbentuk **(sequence length, input dim)**. Anda bisa bayangkan data ini merupakan **kalimat (untaian kata-kata)**.

Linear layer (fully-connected layer) dapat digunakan untuk melakukan proyeksi input pada setiap *timestep* menggunakan **bobot yang sama**.

* Ini **bukan** **Recurrent Layer**; hidden layer sebuah kata **tidak** dipengaruhi oleh hidden layer dari kata lain.



ini independen

nah semua weight yang dipakai semuanya sama

Jadi dia pakai matriks W (weight) yang sama untuk setiap term yang ada (mary, belajar, anum).

itu yang namanya time distributed layer

Some Backgrounds: Time-Distributed Linear Layer

```
input_dim = 4
output_dim = 3

fc = nn.Linear(input_dim, output_dim)

batch, seq_len, input_dim = 1, 3, 4
input = torch.randn(batch, seq_len, input_dim) # random data

# out: [batch, seq_len, output_dim]
out = fc(input)
```

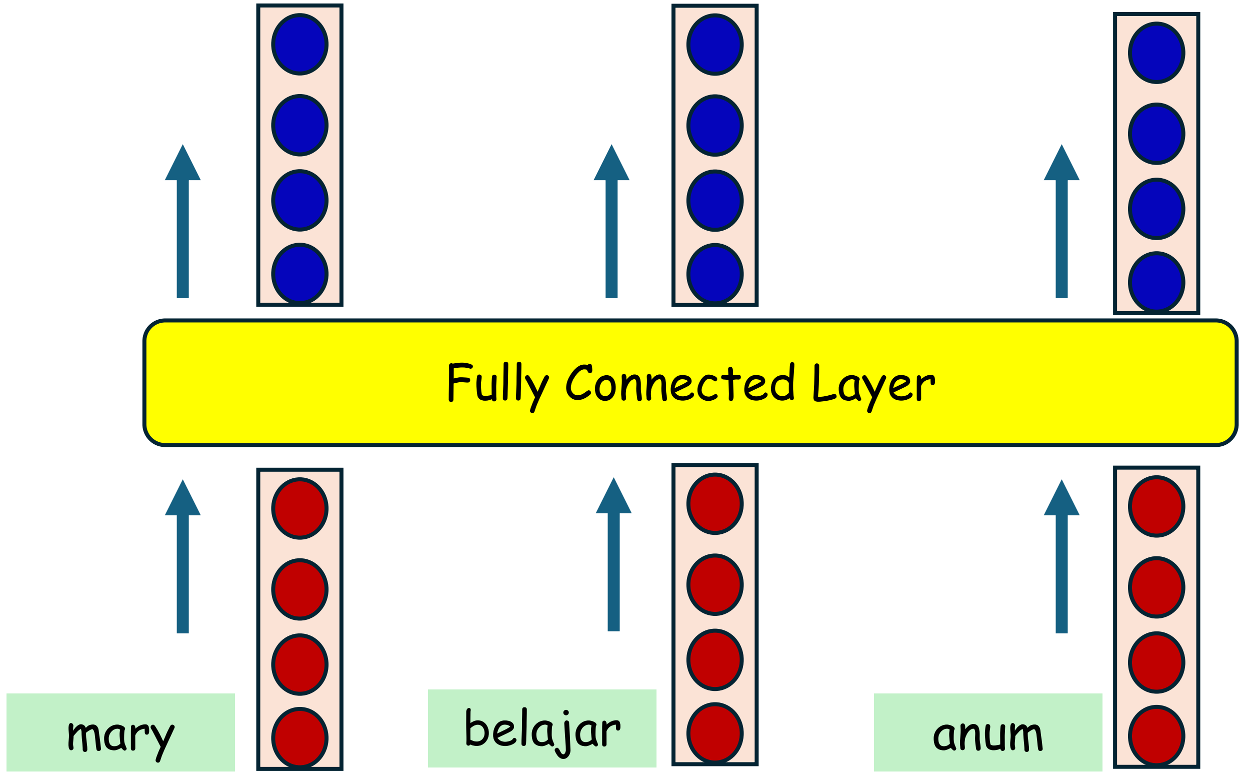
nah nanti outputnya itu jadi

1, 3, 3

```
tensor([[[ 0.5952,  0.7786, -1.0168, -2.1558],
         [ 0.1365,  0.1435, -0.0807,  0.0649],
         [-0.2692,  1.4391,  0.2618,  0.3631]]])
```

```
tensor([[[-0.0516, -0.7474,  0.3282],
         [ 0.5295, -0.2922,  0.2621],
         [ 1.0867, -0.7959, -0.0065]]])
```

Some Backgrounds: Single-Head Linear Layer

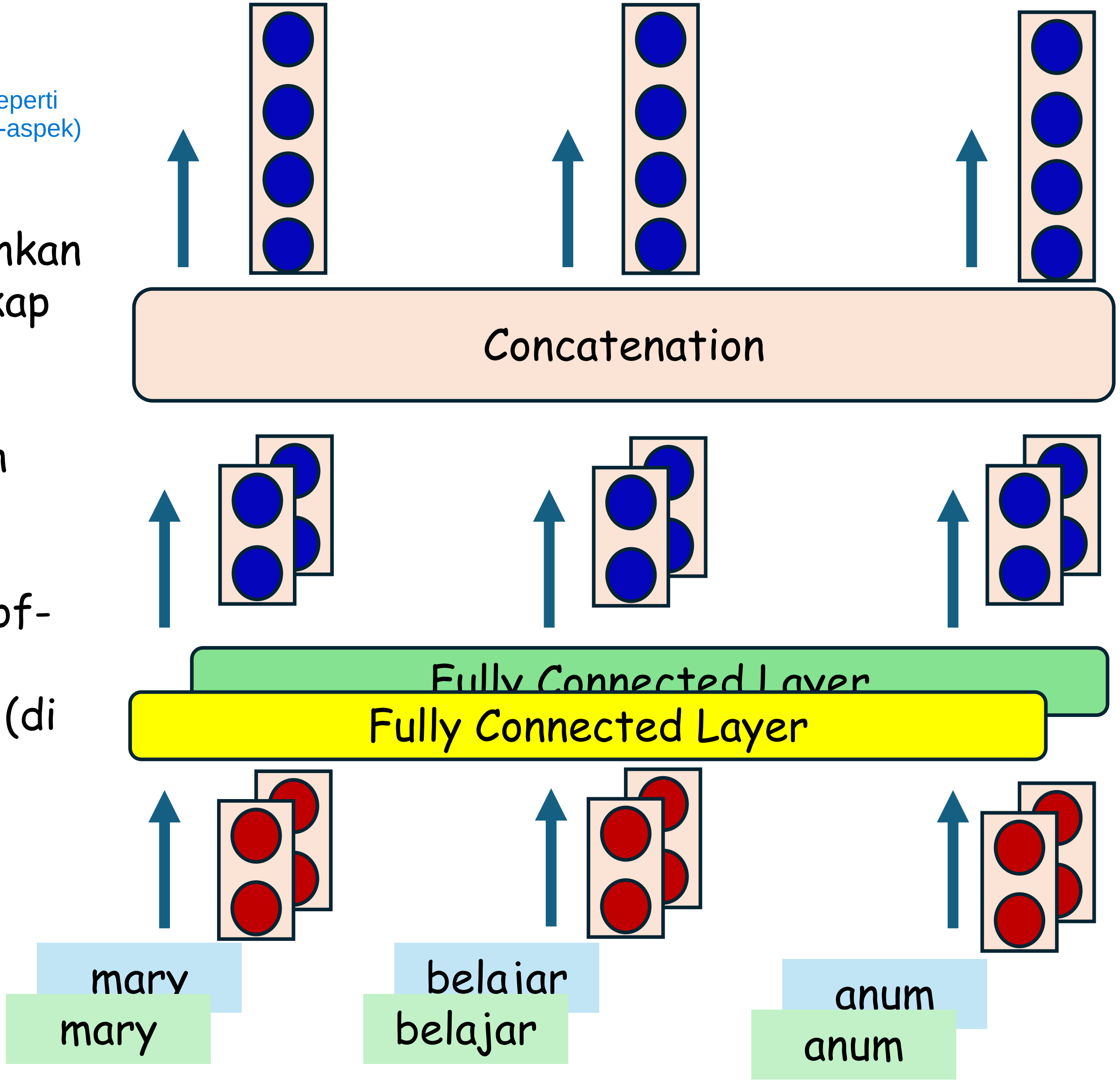


Some Backgrounds: Multi-Head Linear Layer

multihead, model bisa menangkap seperti kelas kata, singular/plural, dll (aspek-aspek)

Skema "**Multi-head**" seperti ini memungkinkan model untuk menangkap **banyak aspek** (pada **subspaces** yang berbeda) dari sebuah kata.

Contoh aspek: Part-of-Speech, Arti Kata, "Jenis Kelamin" Kata (di beberapa bahasa), Plural/Singular, dsb.



Contoh:

2-Head Linear Layer

Some Backgrounds: Multi-Head Linear Layer

```
batch, seq_len, input_dim = 1, 3, 4
num_heads = 2
head_dim = input_dim // num_heads
output_head_dim = 2

fcs = []
for _ in range(num_heads):
    fcs.append( nn.Linear(head_dim, output_head_dim) )

input = torch.randn(batch, seq_len, input_dim) # random input
multi_head_input = input.reshape(batch, seq_len, num_heads, head_dim)

outputs = []
for i in range(num_heads):
    outputs.append( fcs[i](multi_head_input[:, :, i, :]) )

multi_head_output = torch.stack(outputs, dim=2)

output = multi_head_output.reshape(batch, seq_len, num_heads * output_head_dim)
```

Pastikan `input_dim % num_heads == 0`

Masing-masing head mempunyai Linear Layer yang berbeda (bobot berbeda)

num_heads * head_dim harus sama dengan input_dim

Apply Linear Layer untuk setiap head

Tumpuk pada dimensi “head”

concat

di reshape biar 3 dimensi lagi

What is Attention?

Credit: UvA DL Notebooks

The ***attention mechanism*** *describes a* ***weighted average*** *of (sequence) elements with the weights dynamically computed based on an input* ***query*** *and elements'* ***keys****.*

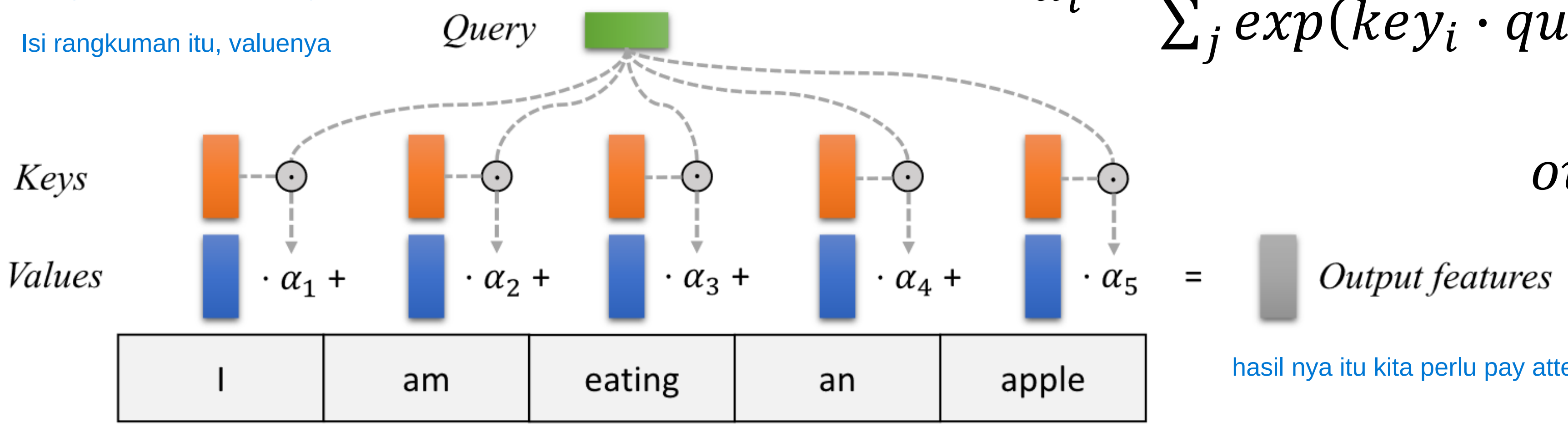
key itu kayak summary nya (misalnya

Rangkuman DDP 1 -> Keys

Isi rangkuman itu, valuenya

Dot product

$$\alpha_i = \frac{exp(key_i \cdot query)}{\sum_j exp(key_i \cdot query)}$$



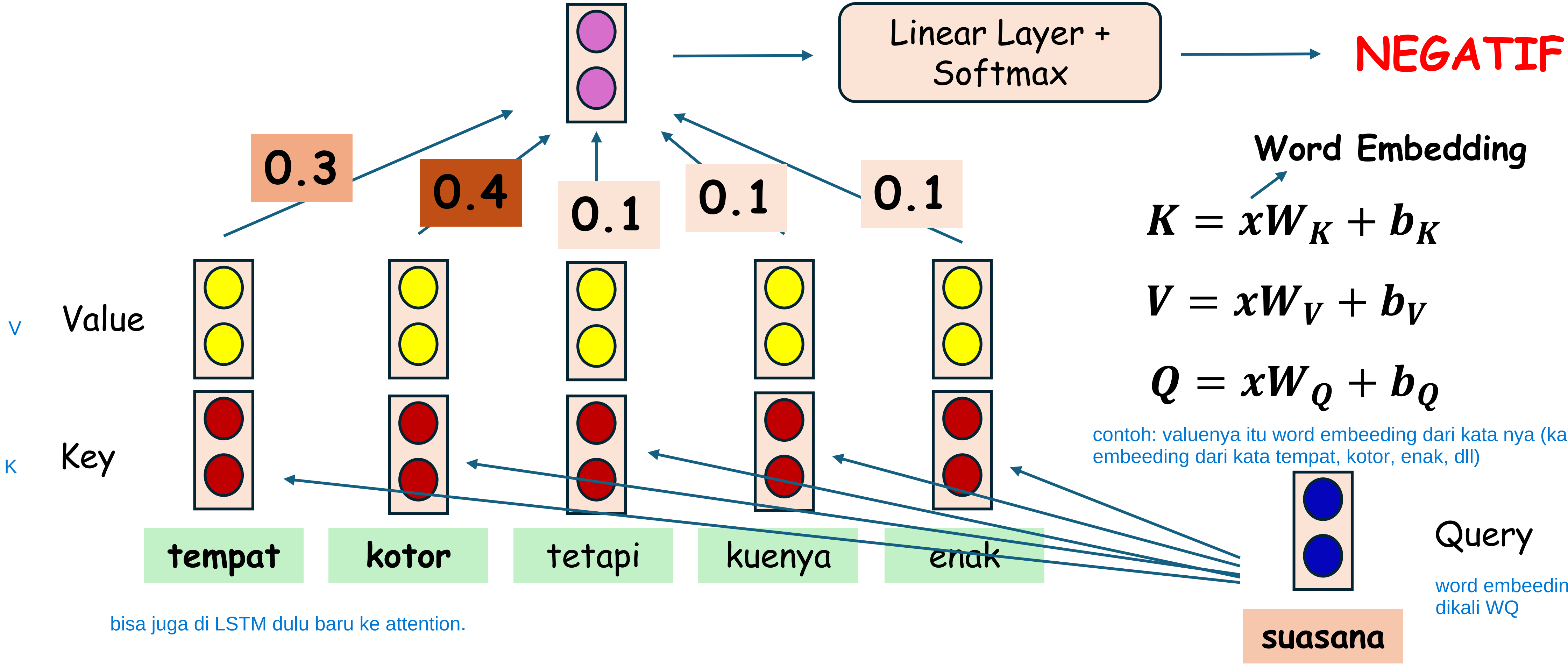
$$out = \sum_i \alpha_i \times value_i$$

hasil nya itu kita perlu pay attention ke bagian yang mana?

misalnya dari query tertentu kita bisa fokus ke apple, tapi juga bisa kita fokus ke query lain kayak eating

Query: A feature vector that describes what **would we maybe want to pay attention to**.
Key: For each input element, we have a key which is again a feature vector. This feature vector roughly describes what the element is "offering".
Value: For each input element, we also have a value vector. This feature vector is the one we want to average over.

Misal, Anda membuat **differentiable model** untuk memprediksi orientasi sentiment sebuah kalimat terhadap **aspek** yang diberikan. Fungsi prediksi menerima input **(kalimat, aspek)** serta output berupa prediksi apakah positif atau negatif.



Word Embedding

$$K = xW_K + b_K$$

$$V = xW_V + b_V$$

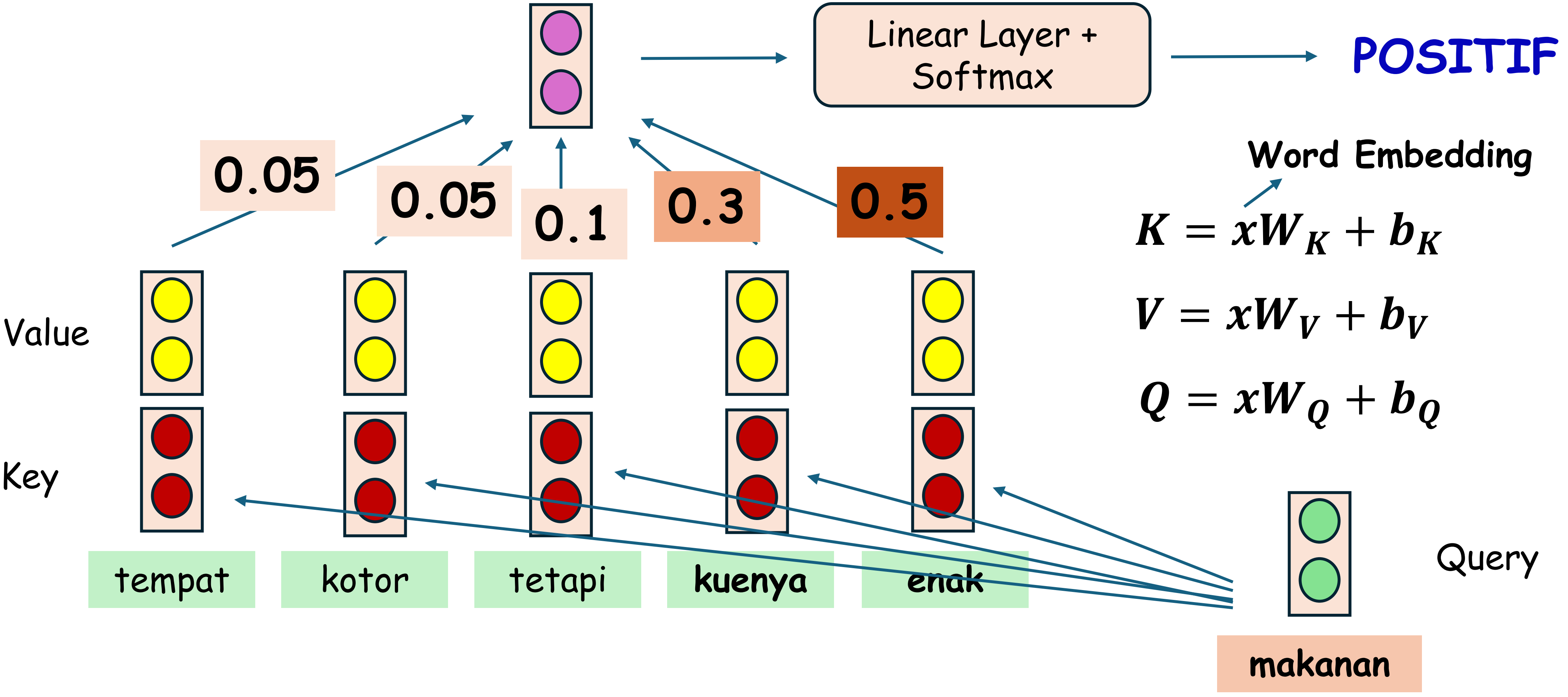
$$Q = xW_Q + b_Q$$

contoh: valuenya itu word embeeding dari kata nya (kayak word embeeding dari kata tempat, kotor, enak, dll)

word embeeding dari susana dikali WQ

bisa juga di LSTM dulu baru ke attention.

Misal, Anda membuat **differentiable model** untuk memprediksi orientasi sentiment sebuah kalimat terhadap **aspek** yang diberikan. Fungsi prediksi menerima input **(kalimat, aspek)** serta output berupa prediksi apakah positif atau negatif.



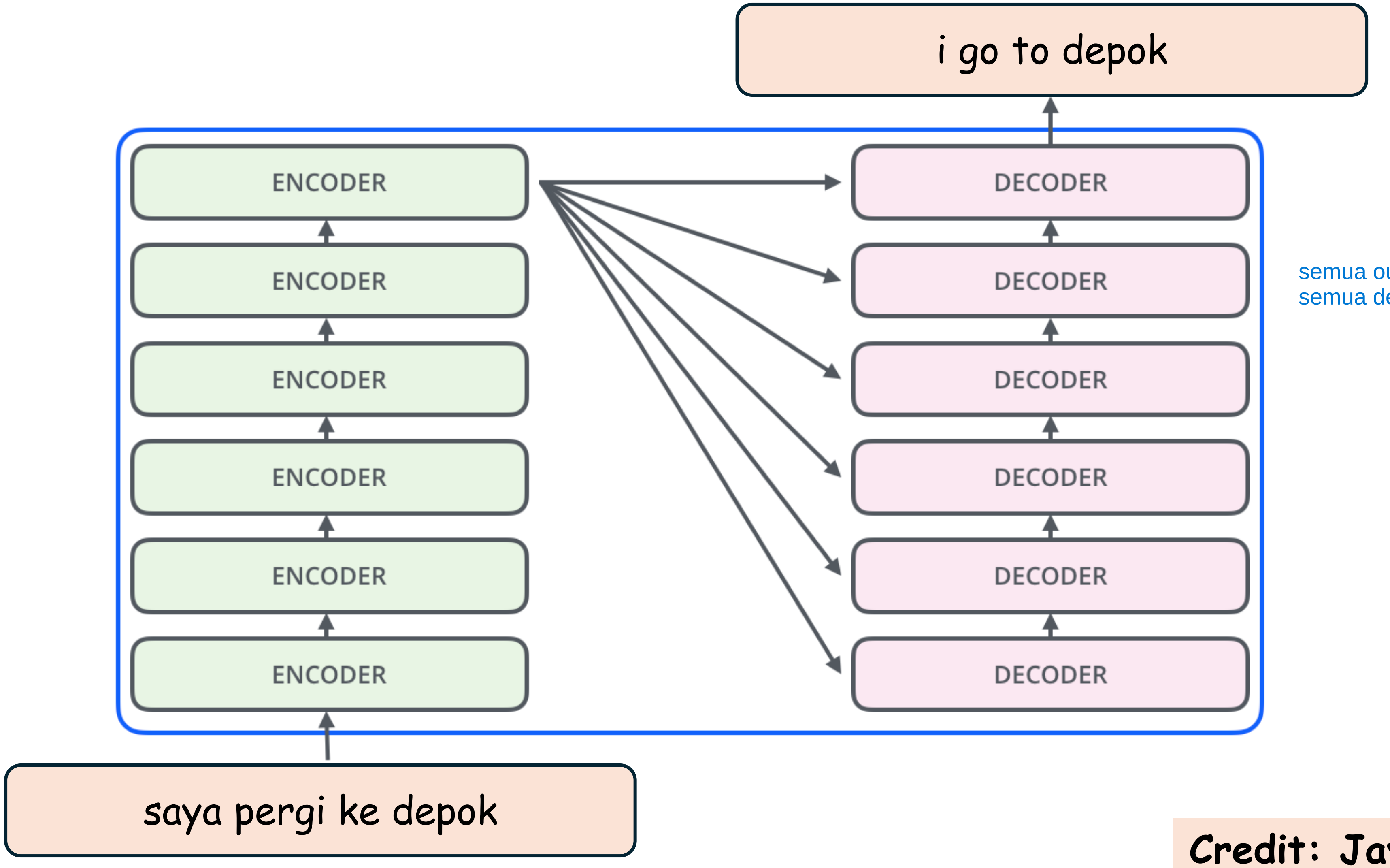
$$K = xW_K + b_K$$

$$V = xW_V + b_V$$

$$Q = xW_Q + b_Q$$

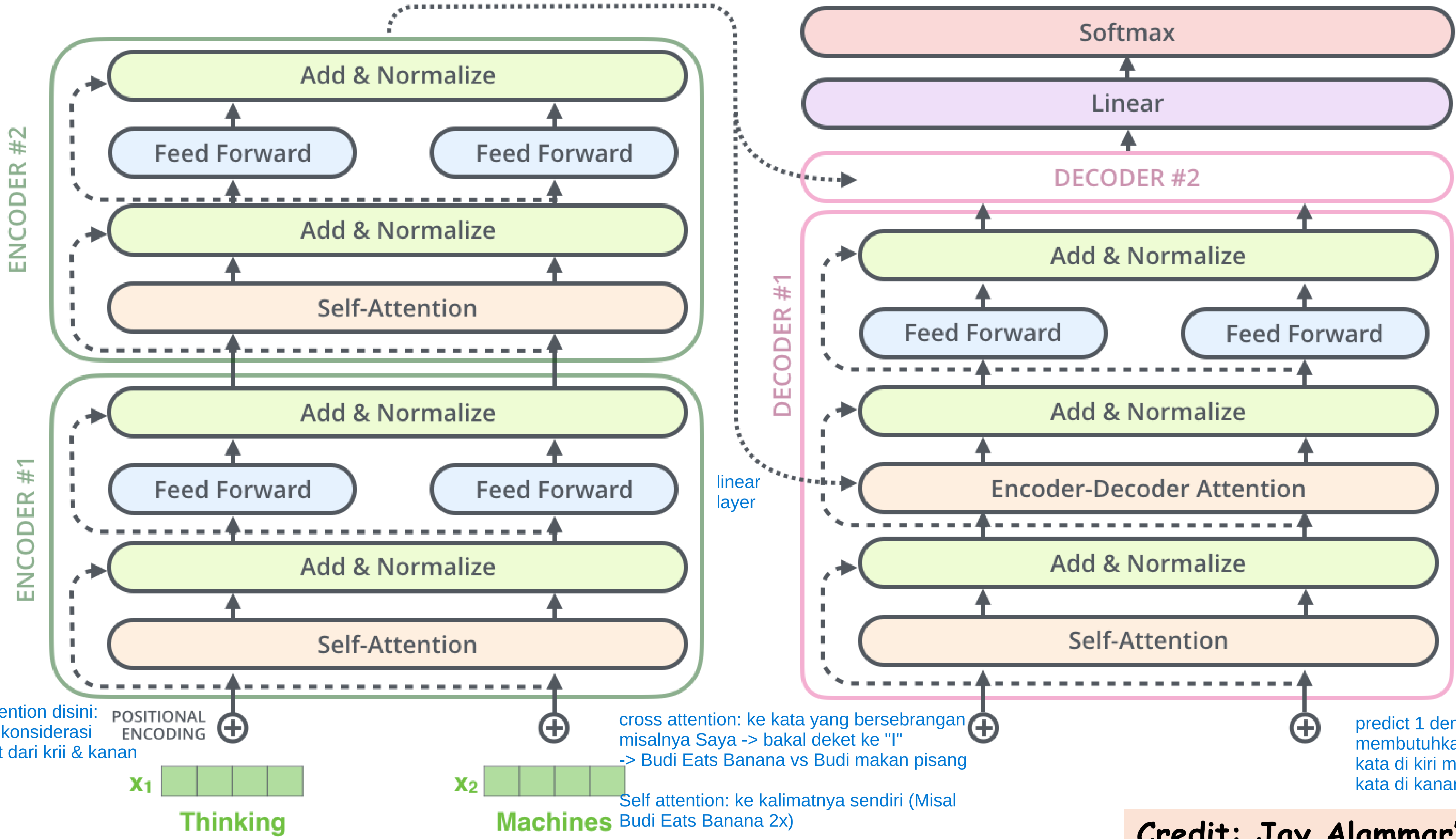
Ok, Let’s become a “Transformer“

- Sebuah arsitektur deep learning untuk sequence-to-sequence modelling.
- Salah satu aplikasinya adalah “Machine Translation”



semua output encoder di kasih ke semua decoder step ny

Credit: Jay Alammar’s Blog



linear layer

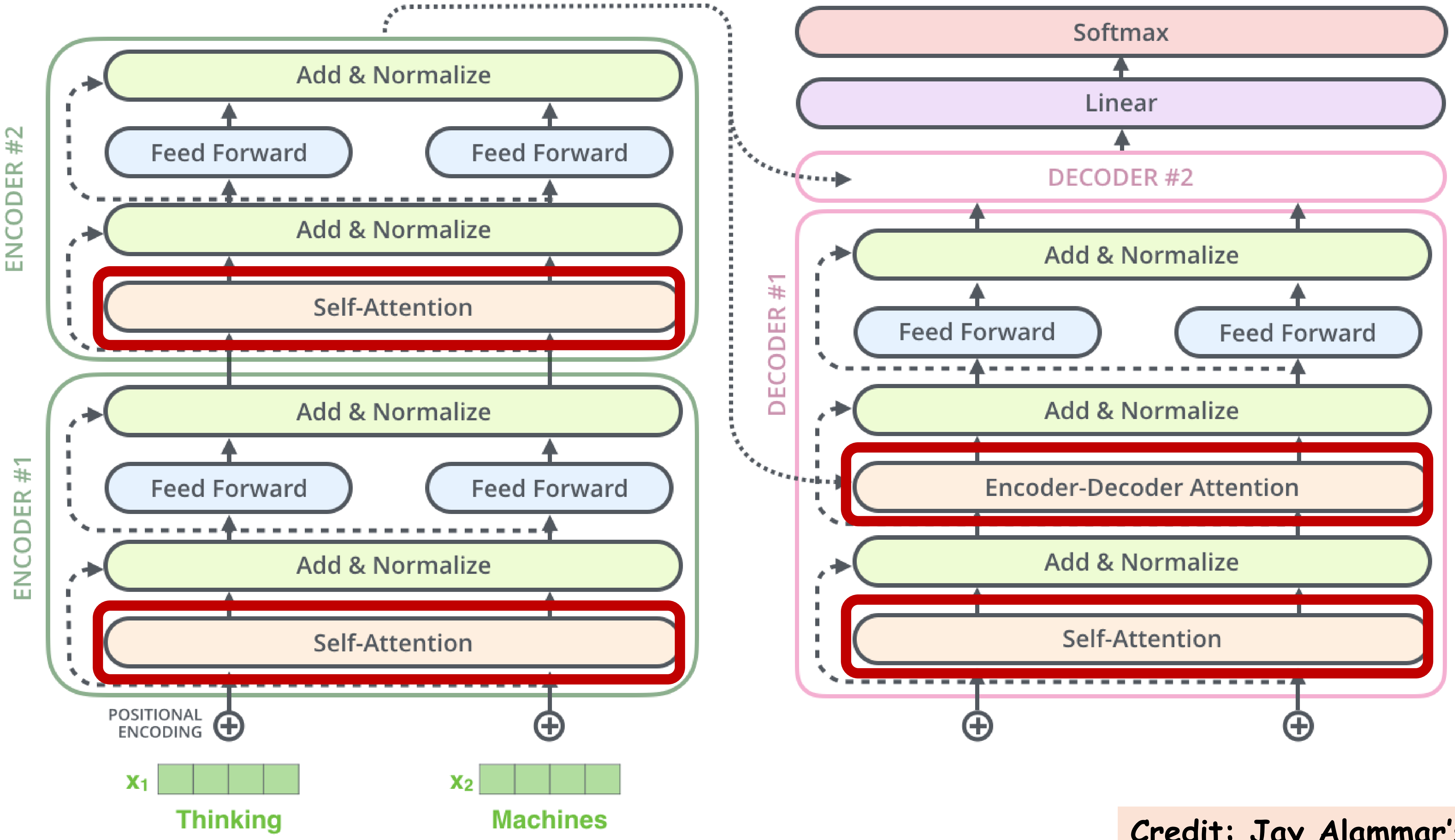
self attention disini:
- mengkonsiderasi context dari krii & kanan

cross attention: ke kata yang bersebrangan misalnya Saya -> bakal deket ke "I"
-> Budi Eats Banana vs Budi makan pisang

Self attention: ke kalimatnya sendiri (Misal Budi Eats Banana 2x)

predict 1 demi 1. Hanya membutuhkan kata di kiri memprediksi kata di kanan

Credit: Jay Alammar's Blog

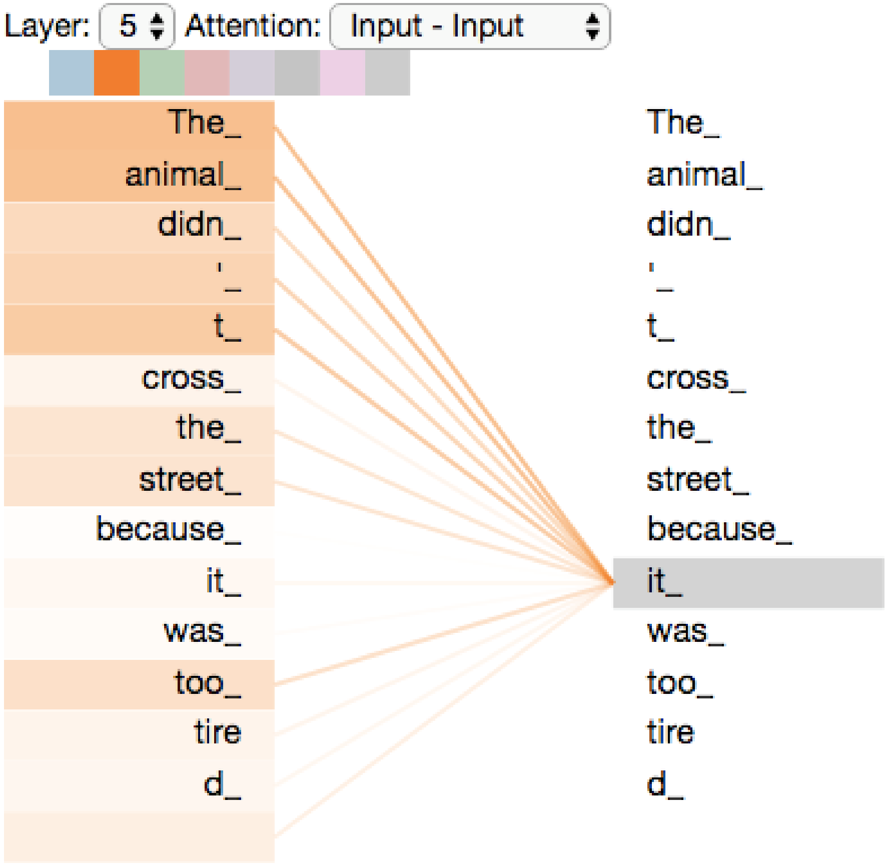


Credit: Jay Alammar's Blog

dapat memperoleh hubungan antara kata The Animal dan IT

Self-Attention

"**The animal** didn't cross the street because **it** was too tired"



mirip kayak LSTM dimana bisa informasi dari The Animal ke It

Sebuah mekanisme untuk **menangkap "hubungan" pada setiap pasangan kata** di sebuah sequence (kalimat atau paragraph)

Credit: Jay Alammar's Blog

Bagaimana menghitung Q, K, V ?

$$Q = XW^Q + b_Q$$

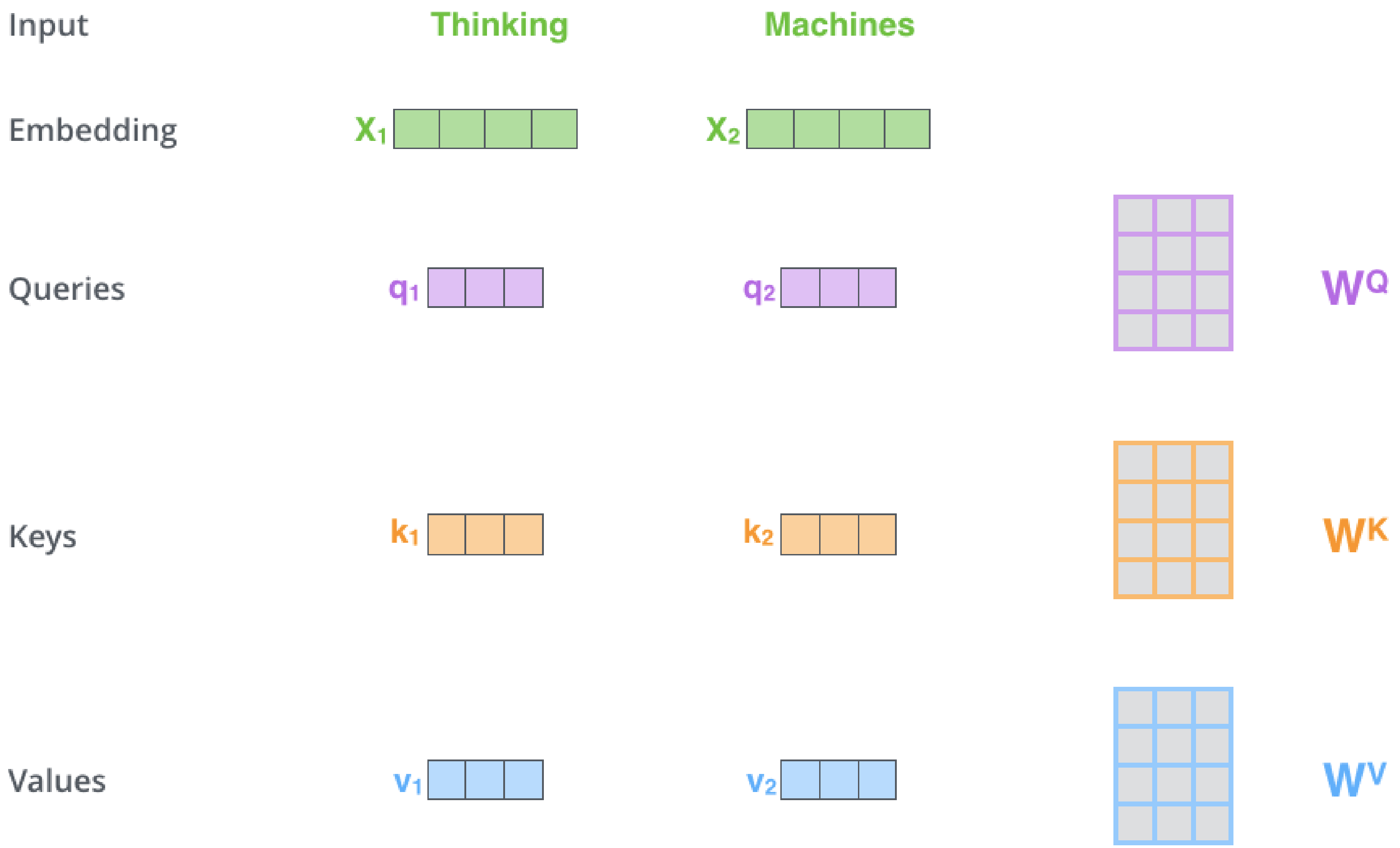
$$K = XW^K + b_K$$

$$V = XW^V + b_V$$

X: (sequence length, input dim)

W: (input dim, hidden dim)

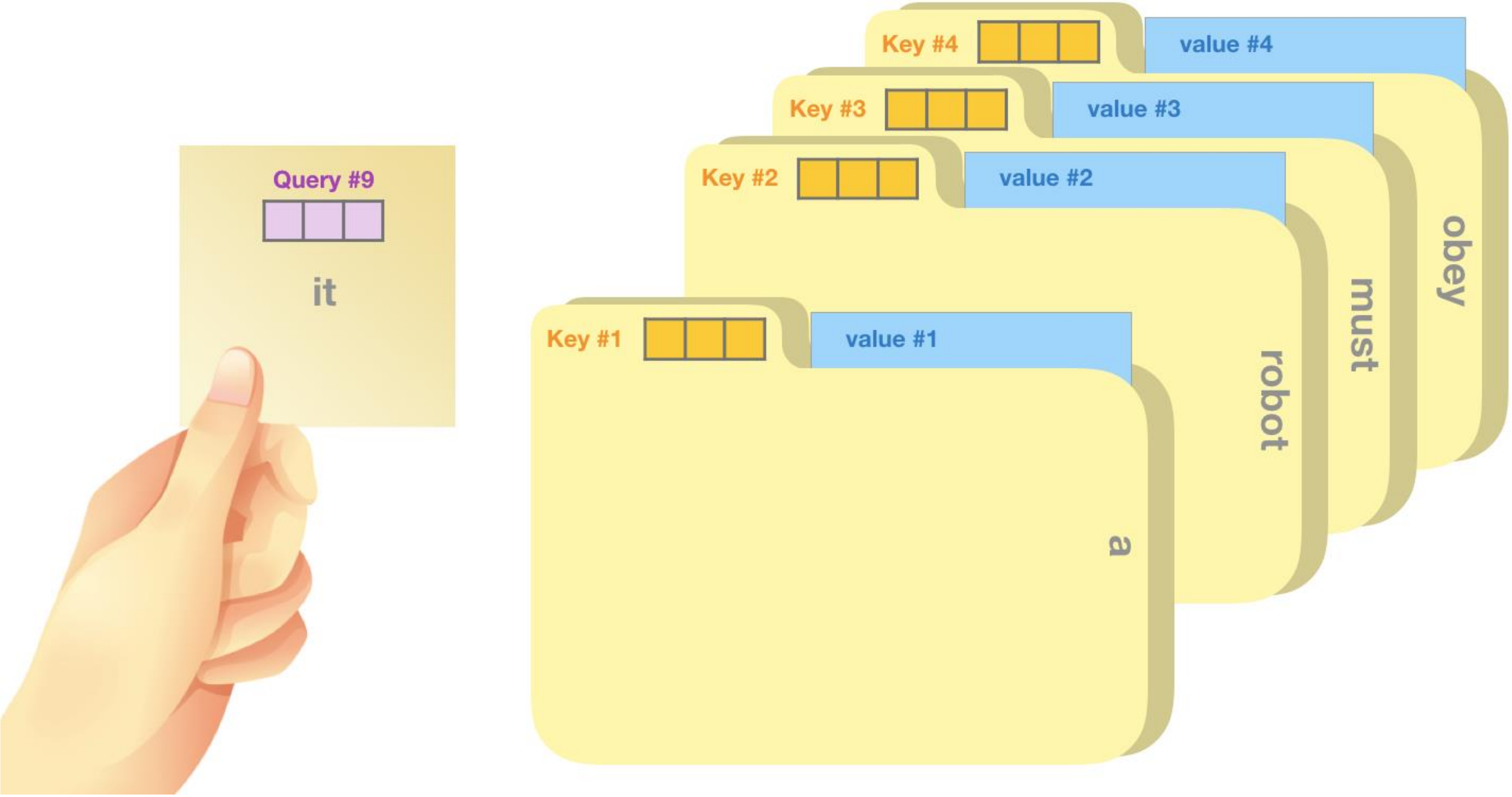
Q, K, V:
(sequence length, hidden dim)



Credit: Jay Alammar's Blog

Memahami Q, K, dan V?

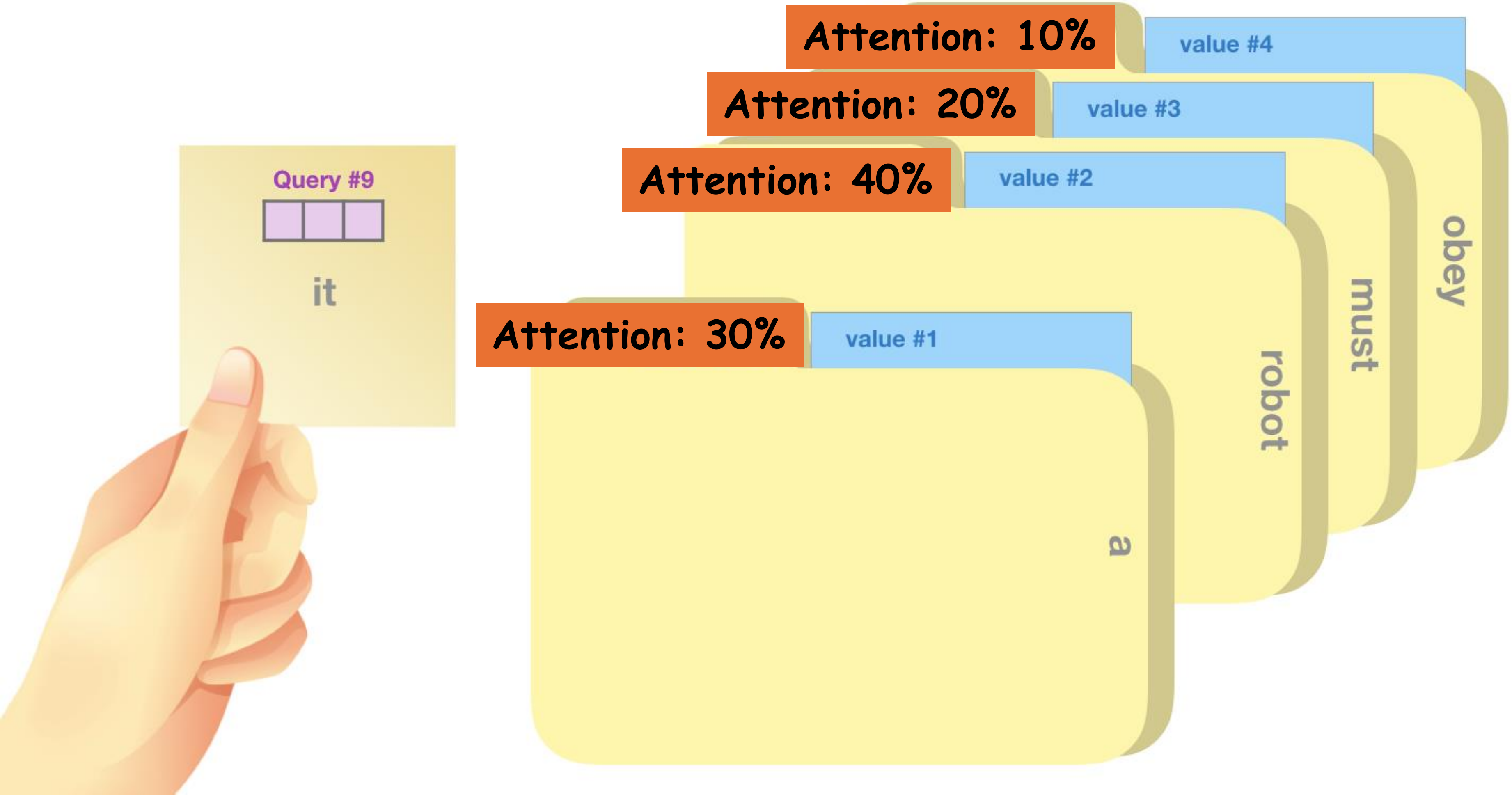
Q (Query) --> representasi sebuah kata, Ketika ia menjadi basis perbandingan dengan kata lain
K (Key) --> representasi "label umum" dari suatu kata; seperti "judul folder"
V (Value) --> informasi full dari sebuah kata; seperti "isi folder"



Credit: Jay Alammar's Blog

Memahami Q, K, dan V?

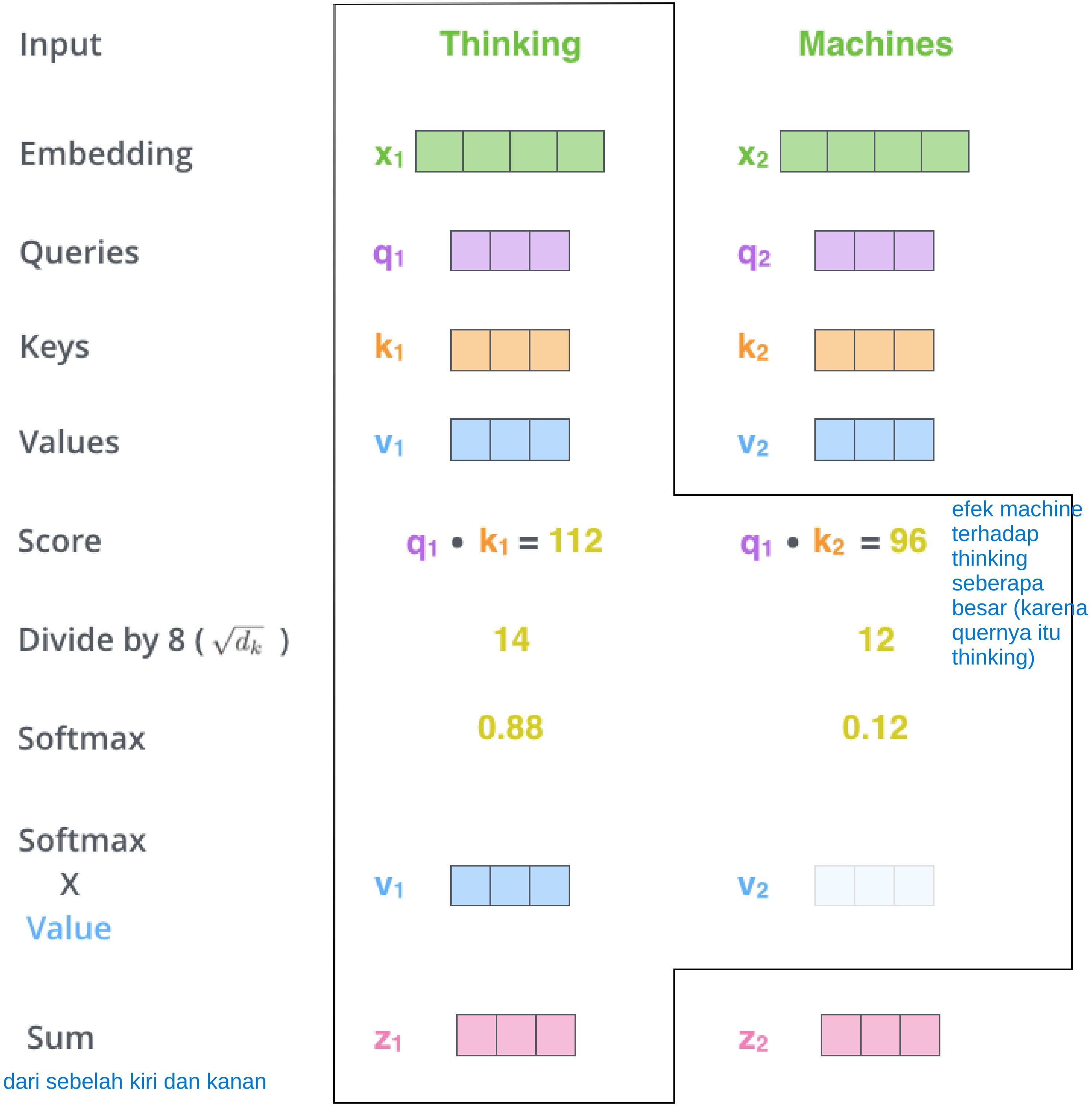
Q (Query) --> representasi sebuah kata, Ketika ia menjadi basis perbandingan dengan kata lain
K (Key) --> representasi "label umum" dari suatu kata; seperti "judul folder"
V (Value) --> informasi full dari sebuah kata; seperti "isi folder"



Credit: Jay Alammar's Blog

Credit: Jay Alammar's Blog

Lebih Detail untuk Self-Attention



efek machine terhadap thinking seberapa besar (karena quernya itu thinking)

rartinya kata yang sum ini

dia udah mempertimbangkan kata-kata dari sebelah kiri dan kanan

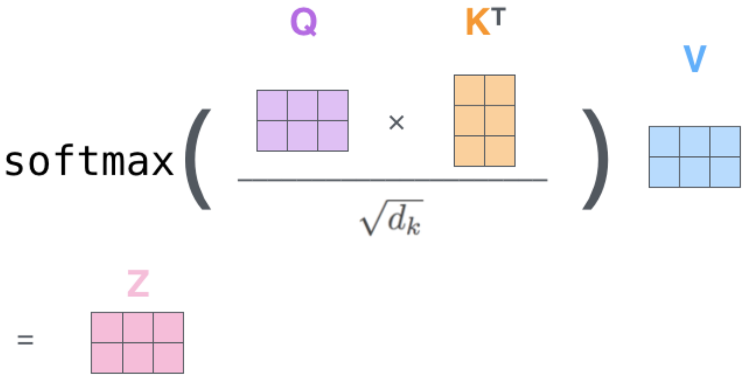
Credit: Jay Alammar's Blog

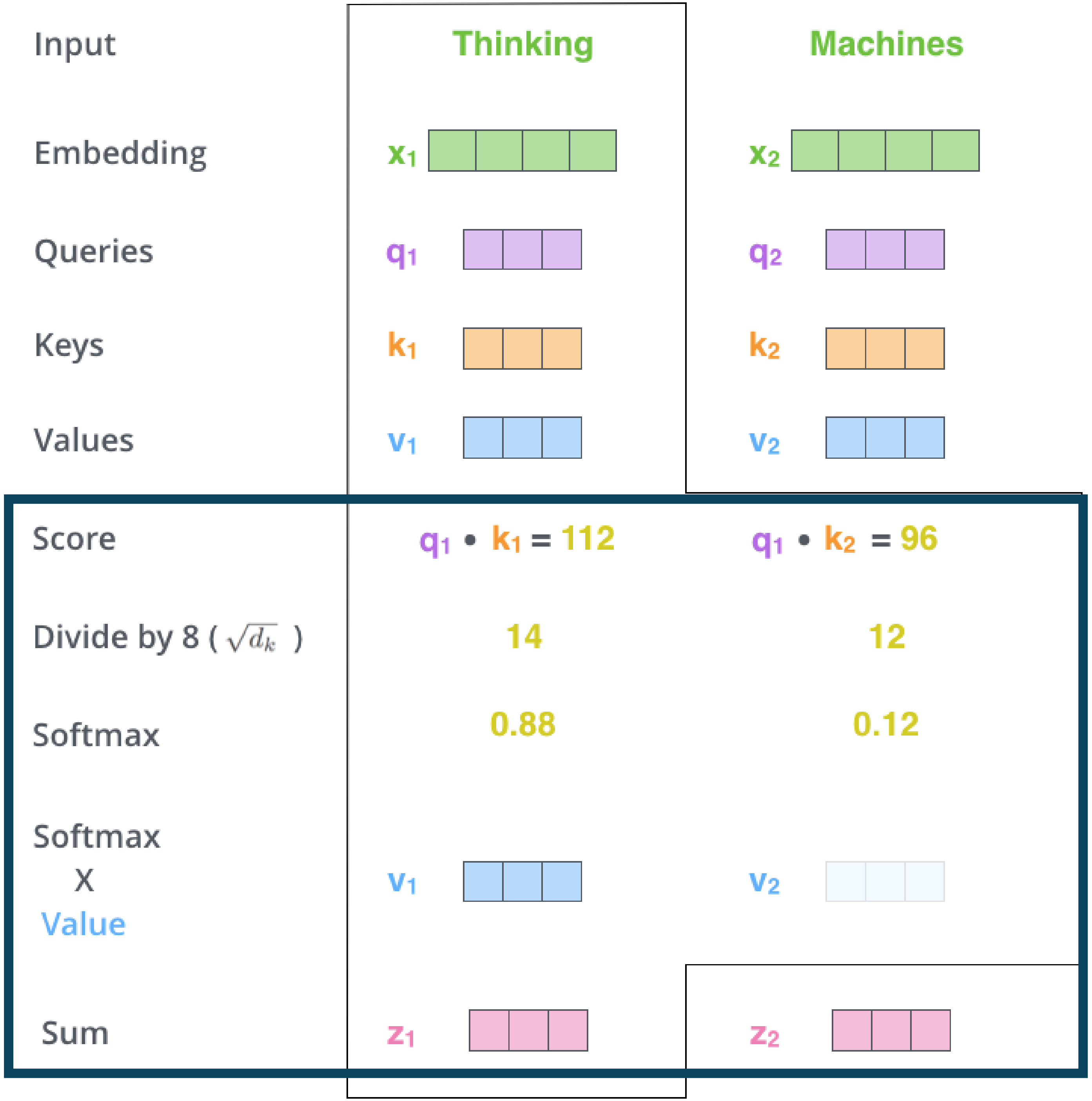
ini biar dapet attention 1 term

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

Q: (seq len, hidden dim)
K: (seq len, hidden dim)
V: (seq len, hidden dim)

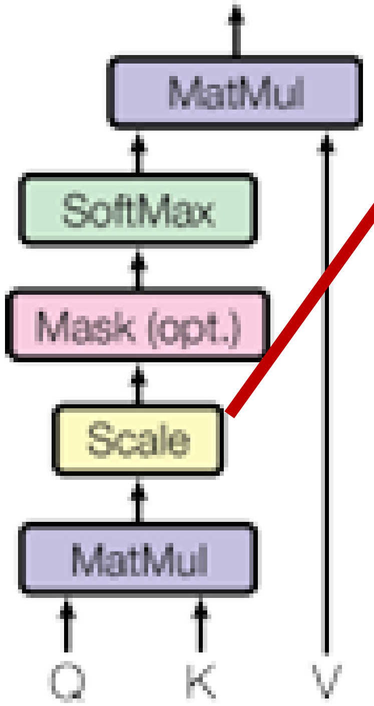
QK^T: (seq len, seq len)





Why scaling factor?

Scaled Dot-Product Attention



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

This scaling factor is crucial to **maintain an appropriate variance of attention values** after initialization.

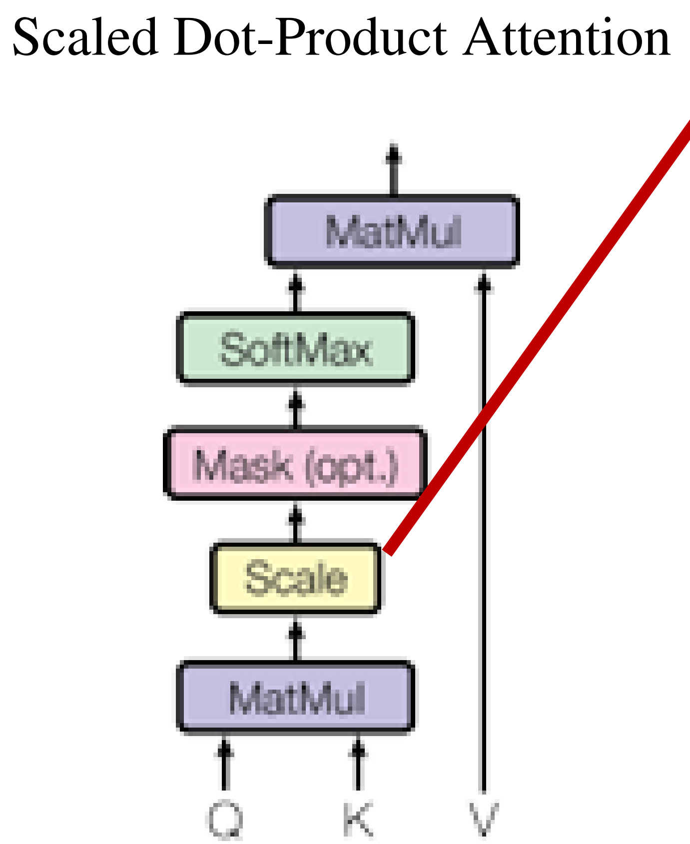
Remember that we initialize our layers with the intention of having equal variance throughout the model, and hence, **Q** and **K** might also have a variance close to **1**.

However, **performing a dot product over two vectors** with a variance σ^2 results in a scalar having d_k-times higher variance:

$$q_i \sim Normal(0, \sigma^2) \qquad k_i \sim Normal(0, \sigma^2)$$

$$Var\left(\sum_{i=1}^{d_k} q_i . k_i\right) = d_k . \sigma^4$$

Why scaling factor?



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

So that's why we need to scale the variance back to σ^2.

If not, the softmax over the logits will already saturate to **1** for one random element and **0** for all others.

The gradients through the softmax will be close to zero so that we can't learn the parameters appropriately.

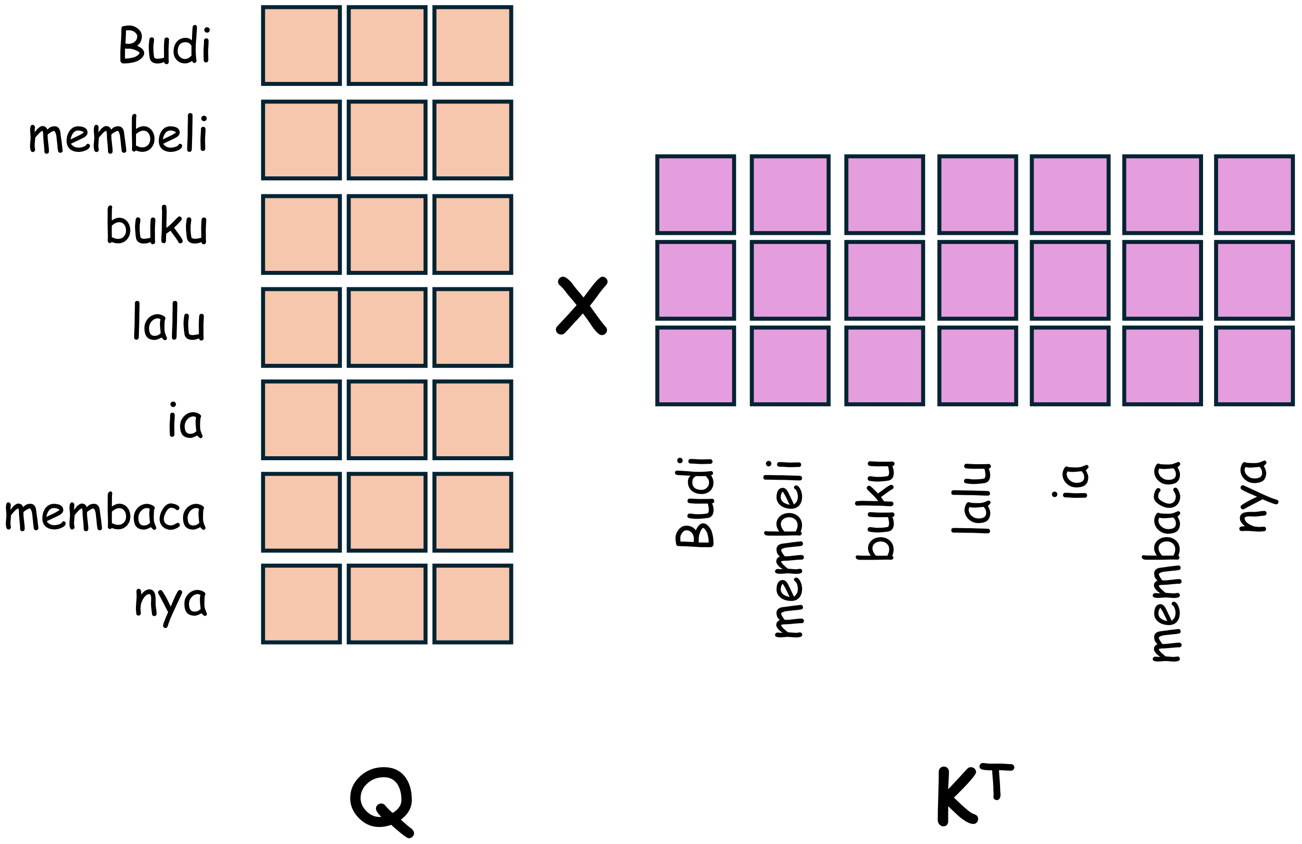
kalau di dot product dulu, nanti variansinya terlalu besar (dot productnya pangkat 4)

Note that the extra factor of σ^2, i.e., having σ^4 instead of σ^2, is usually not an issue, since we keep the original variance σ^2 close to **1** anyways.

agar variansi nilai nya tetap 1

Budi membeli buku lalu ia membaca nya

Nilai berikut hanya contoh



=

	Budi	membeli	buku	lalu	ia	membaca	nya
Budi	3	1	0	1	2	1	0
membeli							
buku	1	1	4	0	0	1	2
lalu							
ia	2	1	1	1	3	1	1
membaca							
nya	1	1	3	0	1	0	3

Budi membeli buku lalu ia membaca nya

dimensi: hidden dim

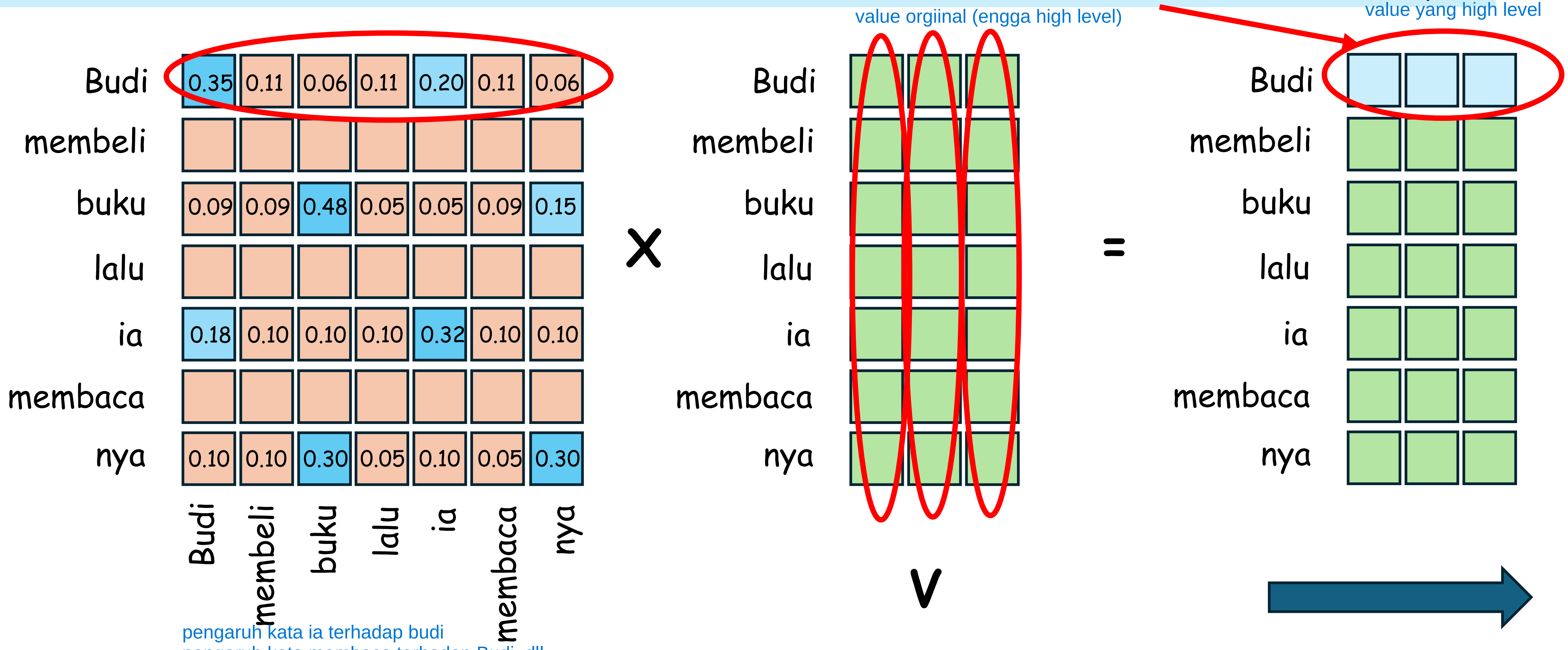
	Budi	membeli	buku	lalu	ia	membaca	nya
Budi	3	1	0	1	2	1	0
membeli							
buku	1	1	4	0	0	1	2
lalu							
ia	2	1	1	1	3	1	1
membaca							
nya	1	1	3	0	1	0	3

Dibagi dengan **sqrt(3)**, lalu masuk ke fungsi **softmax**.

	Budi	membeli	buku	lalu	ia	membaca	nya
Budi	0.35	0.11	0.06	0.11	0.20	0.11	0.06
membeli							
buku	0.09	0.09	0.48	0.05	0.05	0.09	0.15
lalu							
ia	0.18	0.10	0.10	0.10	0.32	0.10	0.10
membaca							
nya	0.10	0.10	0.30	0.05	0.10	0.05	0.30

Budi membeli buku lalu ia membaca nya

Informasi(Budi) = 0.35*v(Budi) + 0.11*v(membeli) + … + 0.20*v(ia) + 0.11*v(membaca) + 0.06*v(nya)

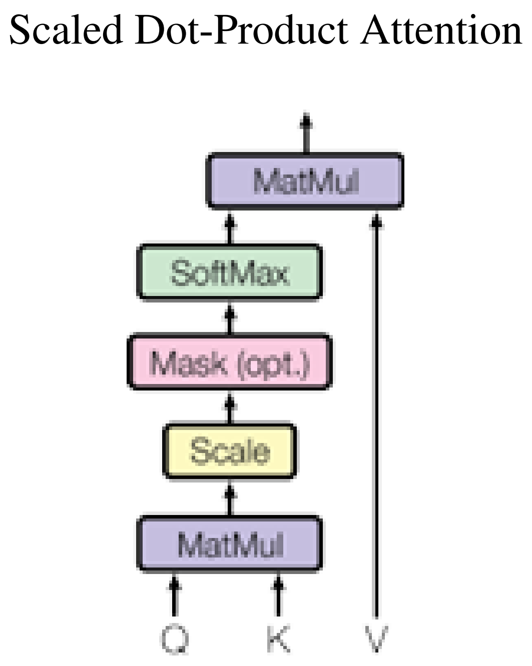


Berlanjut ke layer berikutnya ..

```python
import numpy as np
import random
import math

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.data as data
import torch.optim as optim

from collections import Counter

# Ensure that all operations are deterministic on GPU (if used) for reproducibility
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

device = torch.device("cuda:0") if torch.cuda.is_available() else torch.device("cpu")
print("Device:", device)
```

Credit: UvA DL Notebooks

```
def scaled_dot_product(q, k, v, mask=None):
    d_k = q.size()[-1]
    attn_logits = torch.matmul(q, k.transpose(-2, -1))
    attn_logits = attn_logits / math.sqrt(d_k)

    if mask is not None:
        attn_logits = attn_logits.masked_fill(mask == 0, -9e15)

    attention = F.softmax(attn_logits, dim=-1)
    values = torch.matmul(attention, v)

    return values, attention
```



Untuk melakukan masking pada matriks self-attention QK^T; misal karena **Padding**, atau karena sebuah kata cukup perlu atensi ke kata sebelumnya (khusus Decoder).

Credit: UvA DL Notebooks

```python
seq_len, d_k = 3, 2
q = torch.randn(seq_len, d_k)
k = torch.randn(seq_len, d_k)
v = torch.randn(seq_len, d_k)

values, attention = scaled_dot_product(q, k, v)

print("Q\n", q)
print("K\n", k)
print("V\n", v)
print("Values\n", values)
print("Attention\n", attention)
```

```
Q
 tensor([[-0.9111,  1.8352],
        [ 0.9235,  0.9263],
        [-0.5340,  0.7326]])
K
 tensor([[-0.6524,  0.5424],
        [ 1.3437, -0.6004],
        [-1.1074, -0.5130]])
V
 tensor([[-0.1507, -1.4688],
        [ 0.0515,  1.0427],
        [-0.0928,  1.5567]])
Values
 tensor([[-0.1276, -0.6219],
        [-0.0307,  0.2976],
        [-0.1032, -0.0654]])
Attention
 tensor([[0.7125, 0.0447, 0.2428],
        [0.3211, 0.5594, 0.1195],
        [0.5134, 0.1337, 0.3529]])
```

Silakan Anda coba-coba sendiri …

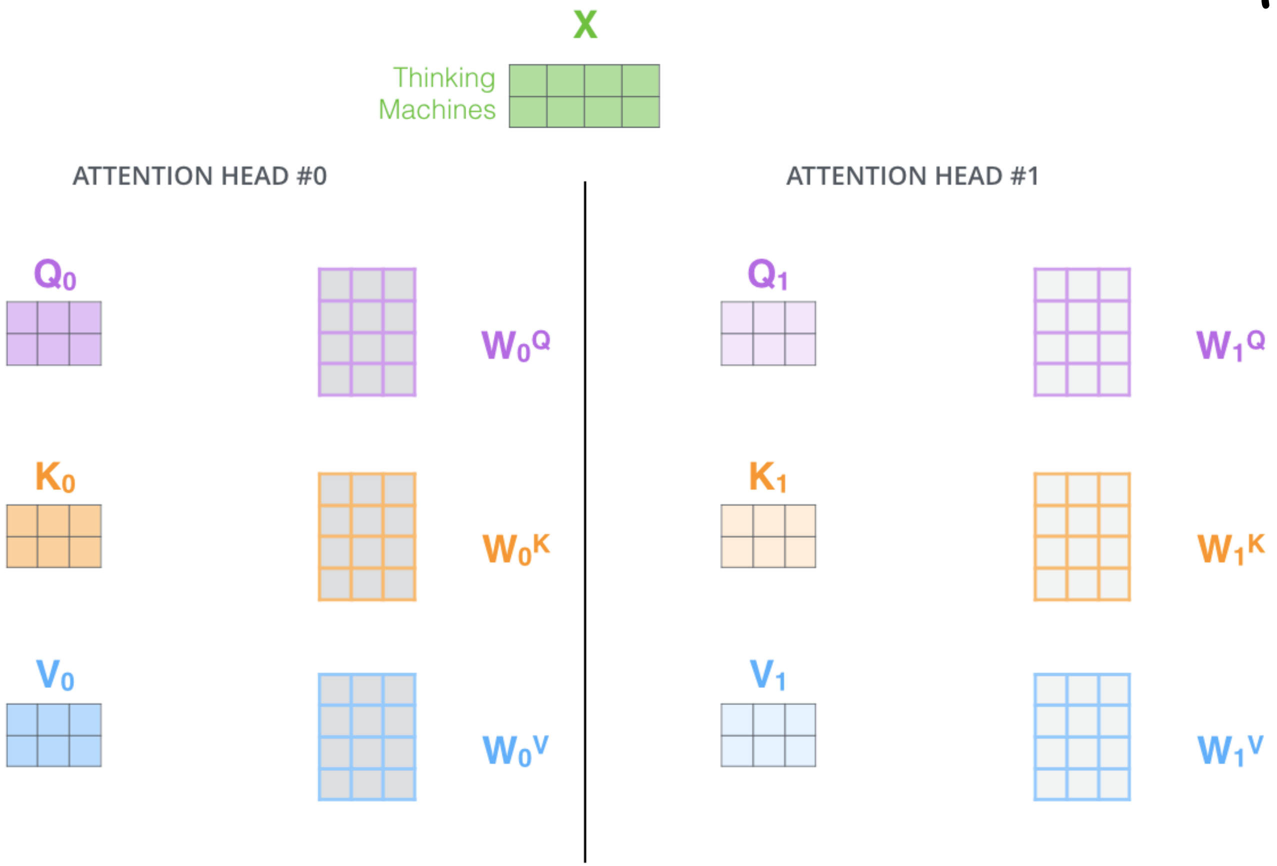
Rajinlah oprek kode PyTorch …

Credit: Jay Alammar's Blog & Vaswani 2017

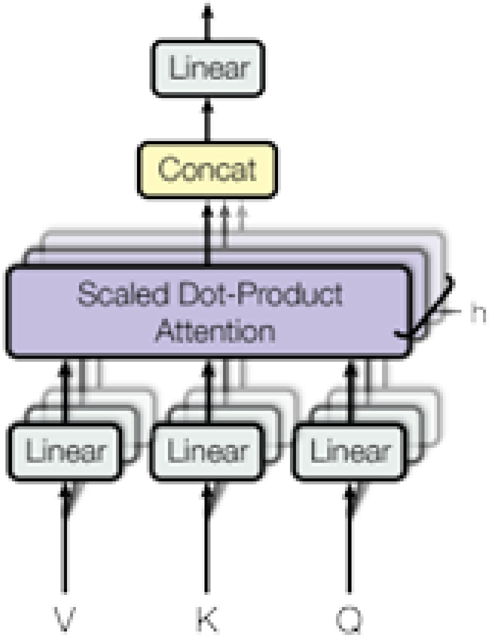
Multi-Head attention

Kita ingin model mempelajari **hubungan antar kata dari berbagai aspek**: kelas kata, makna, gender, dsb.

bisa tanya dari beberapa aspek



Multi-Head Attention



Credit: UvA DL Notebooks

```python
# Helper function to support different mask shapes.
# Output shape supports (batch_size, number of heads, seq length, seq length)
# If 2D: broadcasted over batch size and number of heads
# If 3D: broadcasted over number of heads
# If 4D: leave as is

def expand_mask(mask):
    assert mask.ndim >= 2, "at least 2-dimensional with seq_length x seq_length"
    if mask.ndim == 3:
        # mask is [Batch, SeqLen, SeqLen], so we add "head" dim
        mask = mask.unsqueeze(1)
    while mask.ndim < 4:
        mask = mask.unsqueeze(0) # add "head" dim and "batch" dim
    return mask
```

Credit: UvA DL Notebooks

```
class MultiheadAttention(nn.Module):

    def __init__(self, input_dim, embed_dim, num_heads):
        super().__init__()
        assert embed_dim % num_heads == 0, "Must be 0 modulo number of heads."

        self.embed_dim = embed_dim
        self.num_heads = num_heads
        self.head_dim = embed_dim // num_heads

        #self.qkv_proj = nn.Linear(input_dim, 3*embed_dim)
        self.q_proj = nn.Linear(input_dim, embed_dim)
        self.k_proj = nn.Linear(input_dim, embed_dim)
        self.v_proj = nn.Linear(input_dim, embed_dim)
        self.o_proj = nn.Linear(embed_dim, embed_dim)

        self._reset_parameters()
```

ngebuat weightnya basically

Credit: UvA DL Notebooks

```python
def _reset_parameters(self):
    # Original Transformer initialization, see PyTorch documentation
    for proj in [self.q_proj, self.k_proj, self.v_proj, self.o_proj]:
        nn.init.xavier_uniform_(proj.weight)
        proj.bias.data.fill_(0)

def forward(self, x, y=None, mask=None, return_attention=False):
    if mask is not None:
        mask = expand_mask(mask)

    # if y is not None, then this is a CrossAttention one for decoder
    if y is None:
        y = x

    batch_size, seq_length, _ = x.size()
    batch_size_y, seq_length_y, _ = y.size()

    q = self.q_proj(x) #q,k,v: [Batch, SeqLen, embed_dim]
    k = self.k_proj(y)
    v = self.v_proj(y)
```

cuman reset parmeter (bisa abaikan dulu)

ini biasa bakal kepake untuk attention yang ada di encoder. (karena cuman perlu X)

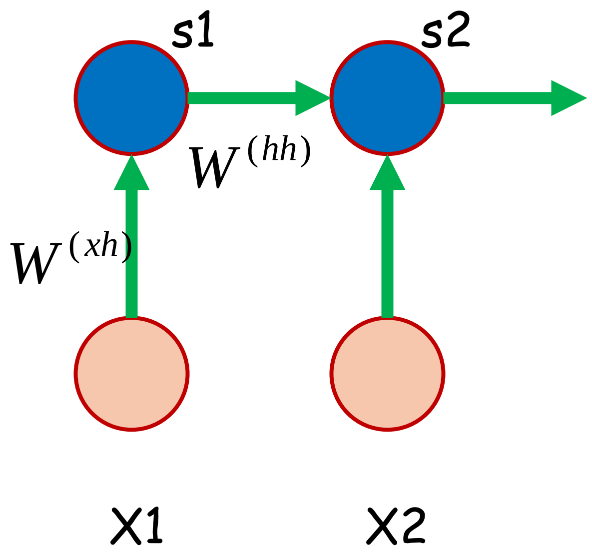
Credit: UvA DL Notebooks

```python
# Adding a new dimension: Head, and set Head as the second dimension
q = q.reshape(batch_size, seq_length, self.num_heads, self.head_dim)
q = q.permute(0, 2, 1, 3) # [Batch, Head, SeqLen, Dims]
k = k.reshape(batch_size_y, seq_length_y, self.num_heads, self.head_dim)
k = k.permute(0, 2, 1, 3)
v = v.reshape(batch_size_y, seq_length_y, self.num_heads, self.head_dim)
v = v.permute(0, 2, 1, 3)

# Determine value outputs
values, attention = scaled_dot_product(q, k, v, mask=mask)
values = values.permute(0, 2, 1, 3) # [Batch, SeqLen, Head, Dims]
values = values.reshape(batch_size, seq_length, self.embed_dim)
o = self.o_proj(values)

if return_attention:
    return o, attention
else:
    return o
```

kalau gak di permute/transpose. Nah, karena output dari (Q * K^T) nya itu sequence length * sequence length , perlu di transpose

karena nanti kita mau kali Q * K^T biar nanti hasilnya itu seq_length, seq_length)

Recurrent vs Self-Attention

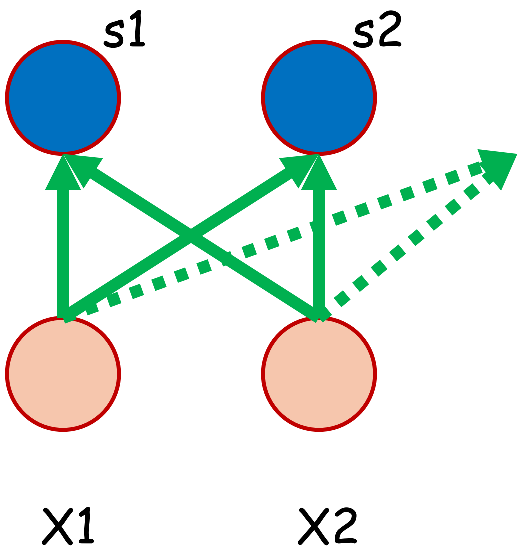


- Di setiap timestep, ada perkalian antara **d x d** matriks bobot dan vektor hidden state berukuran **d**. Jadi, proses perkalian mempunyai kompleksitas **$O(d^2)$**.
- Untuk sequence berukuran **n**, total kompleksitas proses adalah **$O(nd^2)$**.
- **Problem:** proses rekurens terhadap **n** timestep **tidak bisa diparalelkan**!
  - Jadi, operasi sekuensial mempunyai kompleksitas **$O(n)$**

Recurrent vs **Self-Attention**

kalo self attention sendiri

dia tidak tahu posisi kata-katanya.



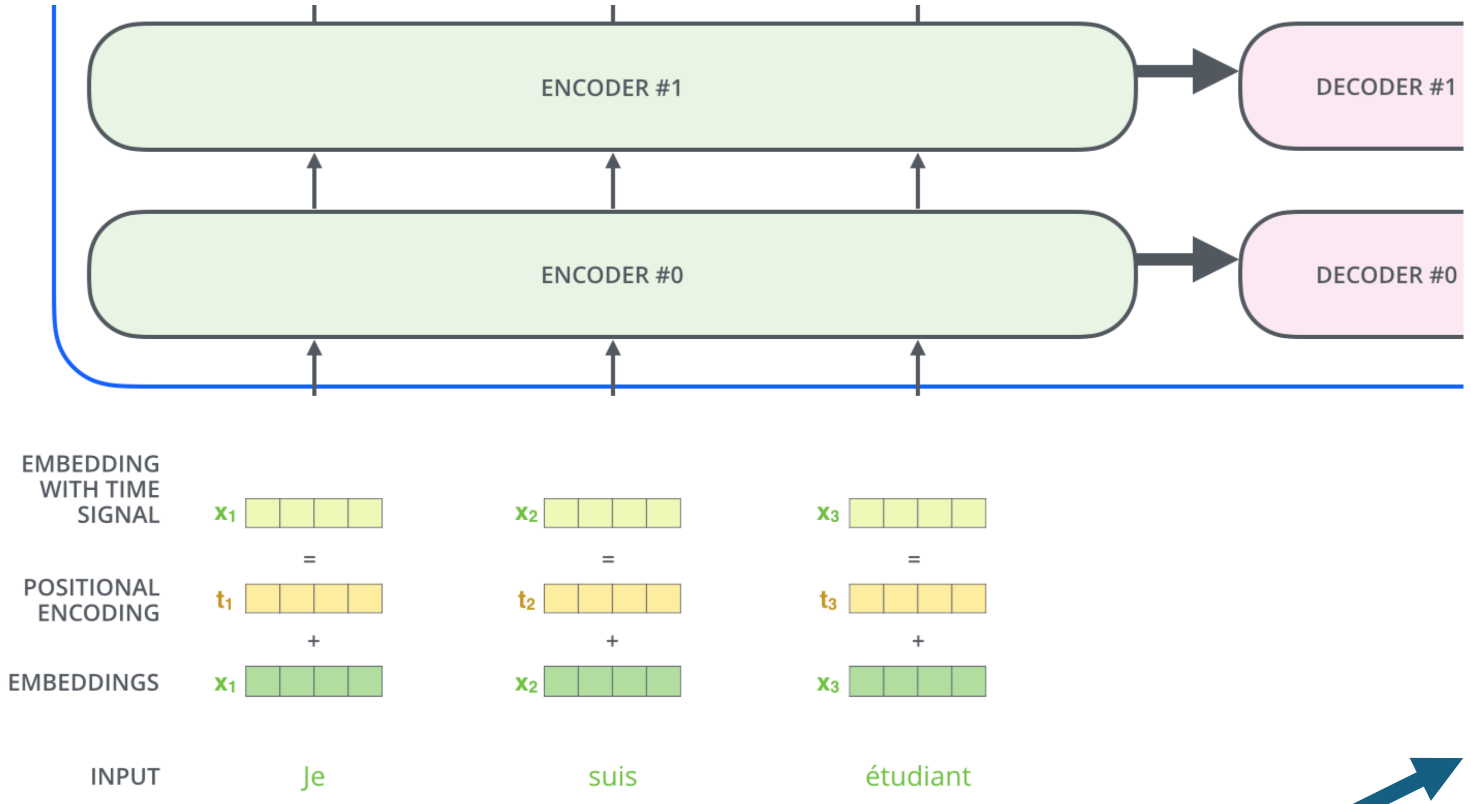
- Matriks Q, K, dan V adalah matriks berukuran **n x d**.
- Output dari **QK**^{T} adalah matiks **n x n**, yang kemudian dikali dengan **V**, dengan kompleksitas **$O(n^2d)$**.
- Berita baiknya adalah proses pada setiap timestep **dapat dilakukan secara paralel**.
  - Dengan kata lain kompleksitas operasi sekuensial adalah $O(1)$
- **Problem:** arsitektur tidak memperhatikan informasi posisi kata
  - **Solusi: positional encoding**

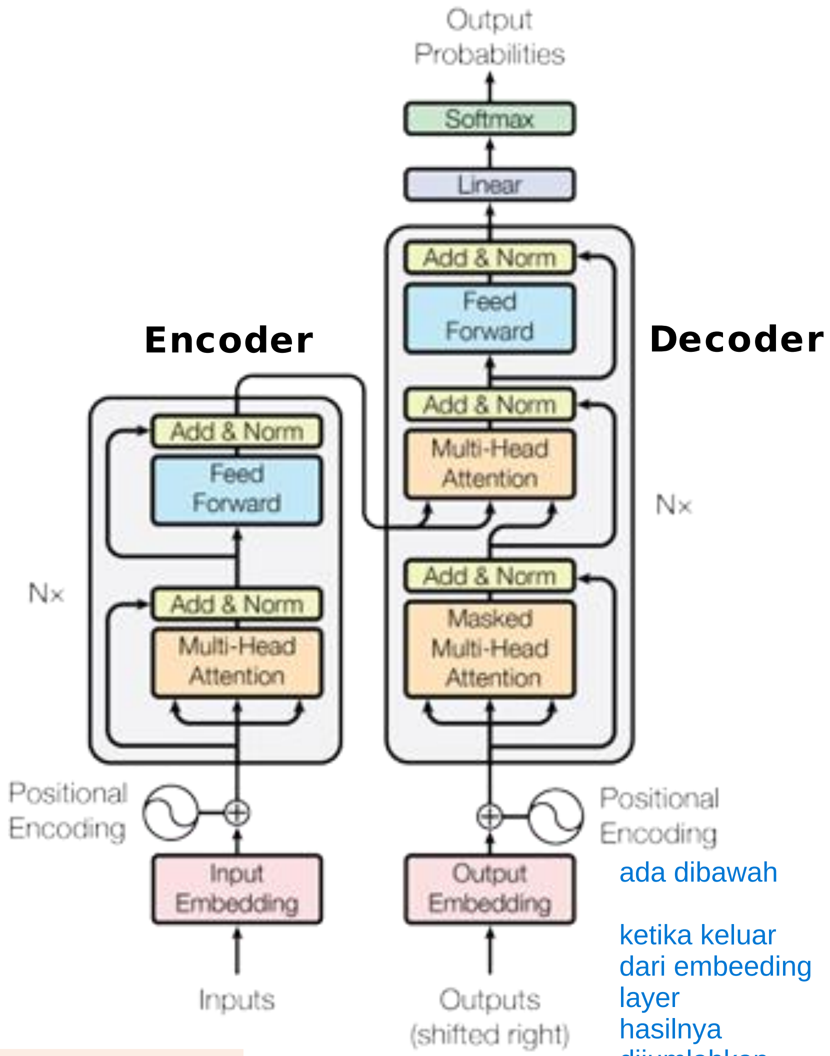
artinya di word embeeding nanti ada dia taro positionnya

Positional Encoding

Credit: Jay Alammar's Blog & Vaswani 2017

Positional Encoding --> memodelkan "urutan kata"





ada dibawah

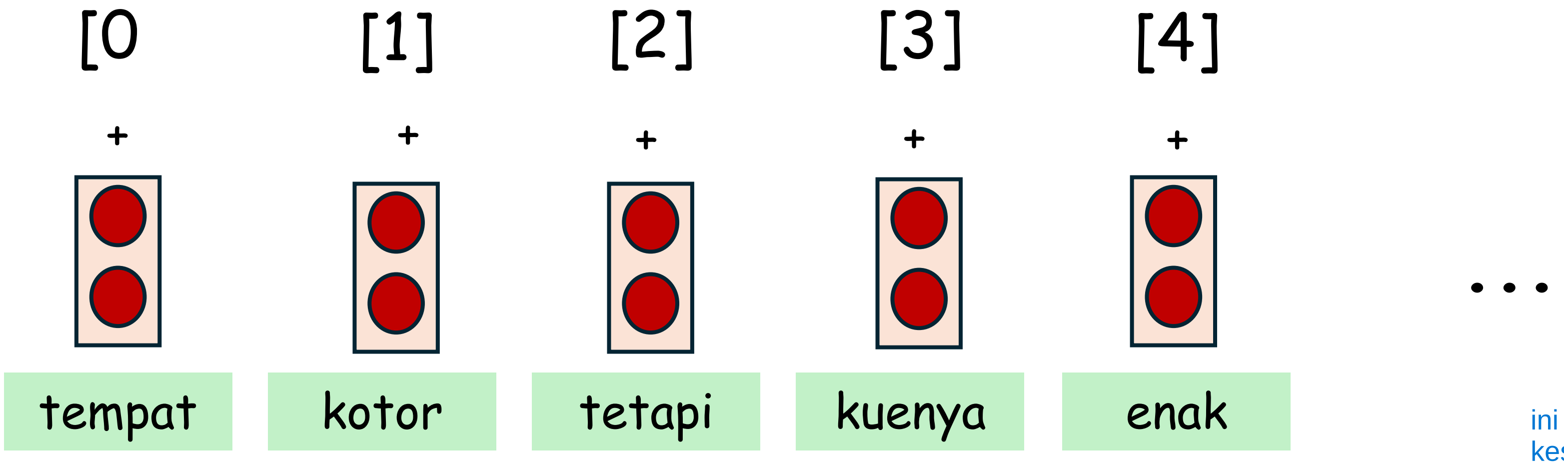
ketika keluar dari embeeding layer hasilnya dijumlahkan dengan positional econding

Beda dengan RNNs, transformer perlu informasi tambahan terkait posisi pada input agar agar **positionally aware**.

Bagaimana cara "encode" posisi token?

Ya sudah pakai integer saja yang di-concat ke vektor embedding kata. **Problem?**

makin lama makin besar informasi posisinya (kayak kalo lengthnya ada 1000). Angka2 yang terlalu besar bisa merusak komputasi gradien (gradien exploding)



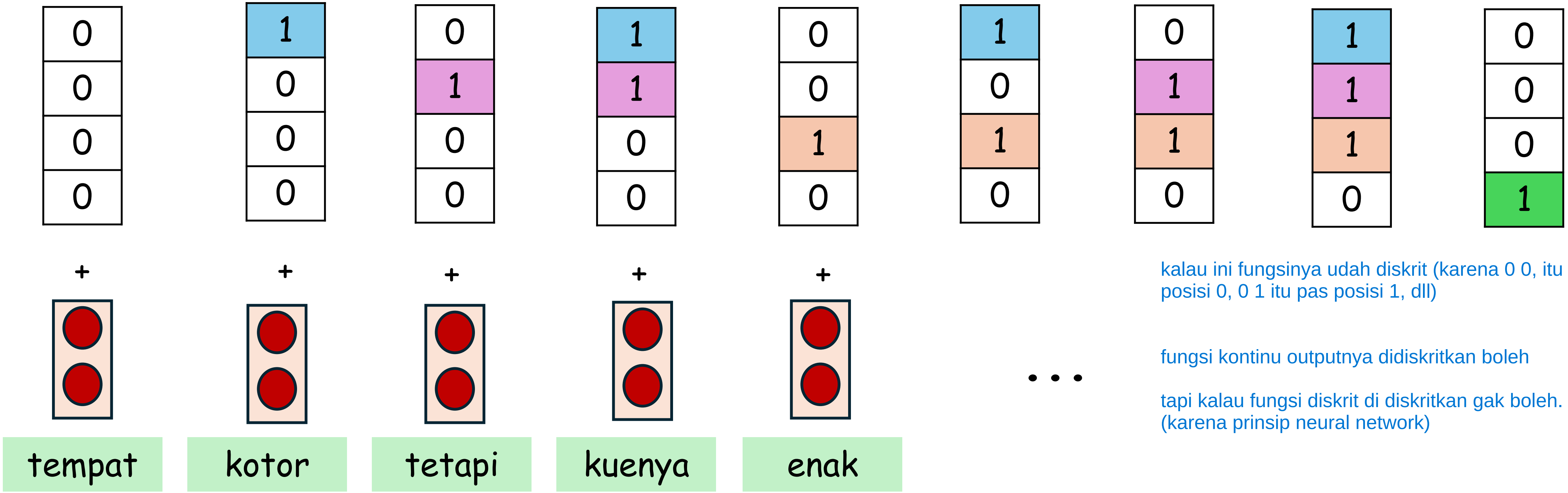
0 0.2 0.4 0.6 0.8

ini gak bisa karena terlalu reliant dengan panjang dari keseluruhan tokennya.

Jadi, kita sebnarnya gak tau 0.4 itu posisi berapa karena panjangnya aja kita perlu hitung lagi

Bagaimana cara "encode" posisi token?

Kalau begitu, kita pakai **binary encoding. Masih ada masalah juga?**



kalau ini fungsinya udah diskrit (karena 0 0, itu pas posisi 0, 0 1 itu pas posisi 1, dll)

fungsi kontinu outputnya didiskritkan boleh

tapi kalau fungsi diskrit di diskritkan gak boleh. (karena prinsip neural network)

Bagaimana cara “encode” posisi token?

Kalau begitu, apa dong yang lebih baik?

0	1	0	1	0	1	0	1	0
0	0	1	1	0	0	1	1	0
0	0	0	0	1	1	1	1	0
0	0	0	0	0	0	0	0	1

lihat deh, dia dari awal

0 nya 1, 0 nya 2, 0nya 4, 0 nya 8 (nah basically karena dia berulang terus menerus, jadi dia periodik dimana menggabmbarkan cos sin)

Do you see several patterns? What are the patterns?

intervalnya makin bawah makin ilang

Fungsi apa di matematika yang bisa memodelkan nilai berulang secara periodik?

Credit: UvA DL Notebooks

Bagaimana cara "encode" posisi token?

Transformer's positional encoding

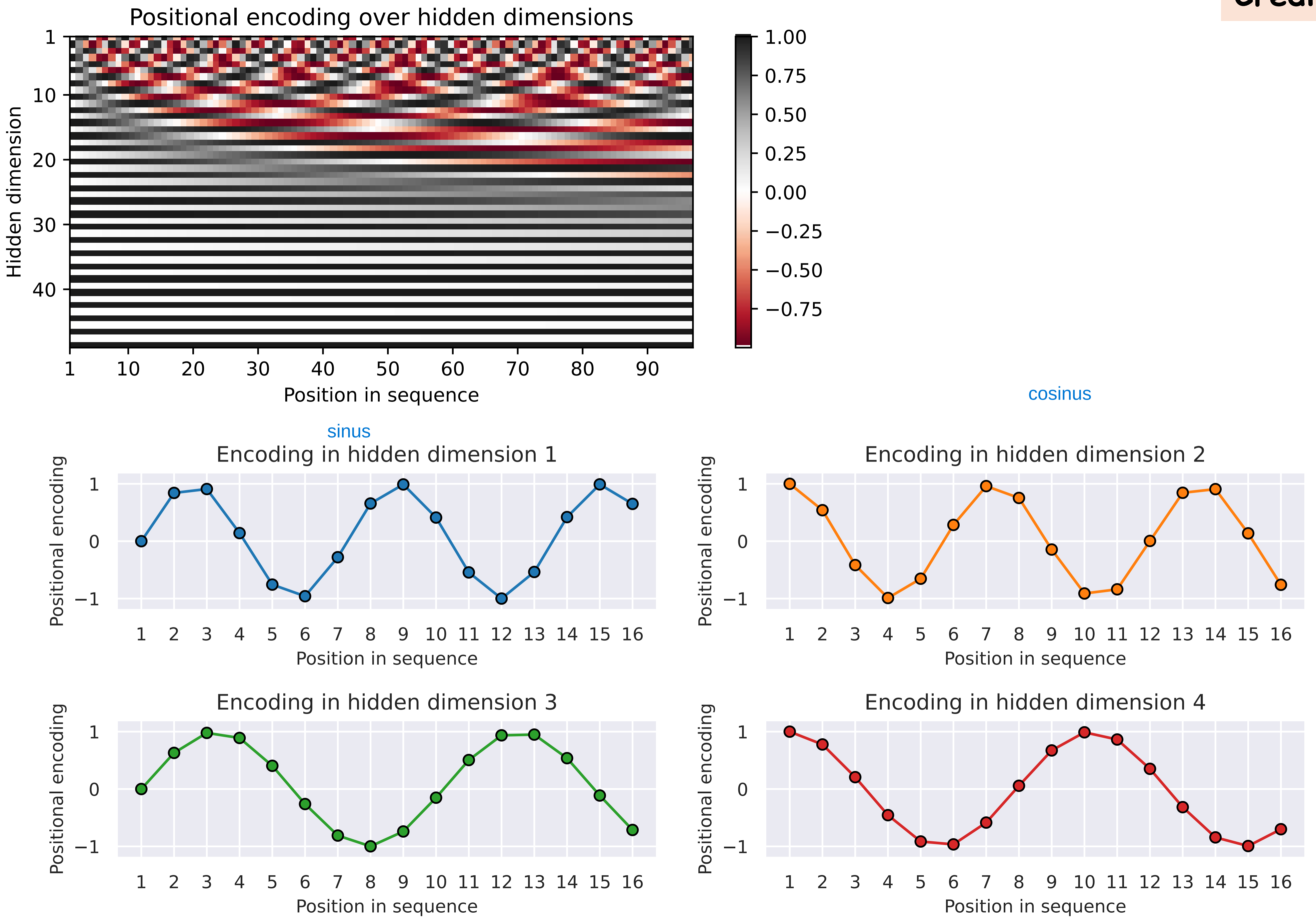
$$PE_{(pos,i)} = \begin{cases} \sin\left(\frac{pos}{10000^{i/d_{\text{model}}}}\right) & \text{if } i \bmod 2 = 0 \\ \cos\left(\frac{pos}{10000^{(i-1)/d_{\text{model}}}}\right) & \text{otherwise} \end{cases}$$

kalo ganjil pakai cosinus

$PE_{(pos,i)}$ represents the position encoding at position pos in the sequence, and hidden dimensionality i.

These values, concatenated for all hidden dimensions, are added to the original input features, and constitute the position information.

Credit: UvA DL Notebooks



https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Contoh: Positional Encoding untuk dimensi 4

Positional Encoding Matrix with d=4, n=100

Sequence	Index of token, k	genap $i=0$	ganjil $i=0$	genap $i=1$	ganjil $i=1$
I	0	$P_{00}=\sin(0) = 0$	$P_{01}=\cos(0) = 1$	$P_{02}=\sin(0) = 0$	$P_{03}=\cos(0) = 1$
am	1	$P_{10}=\sin(1/1) = 0.84$	$P_{11}=\cos(1/1) = 0.54$	$P_{12}=\sin(1/10) = 0.10$	$P_{13}=\cos(1/10) = 1.0$
a	2	$P_{20}=\sin(2/1) = 0.91$	$P_{21}=\cos(2/1) = -0.42$	$P_{22}=\sin(2/10) = 0.20$	$P_{23}=\cos(2/10) = 0.98$
Robot	3	$P_{30}=\sin(3/1) = 0.14$	$P_{31}=\cos(3/1) = -0.99$	$P_{32}=\sin(3/10) = 0.30$	$P_{33}=\cos(3/10) = 0.96$

Jika dimensi embedding = 4

$$[\sin(i)\,,\cos(i)\,,\sin\left(\frac{i}{\sqrt{10000}}\right),\cos\left(\frac{i}{\sqrt{10000}}\right)]$$

Secara umum:

$$0 \le i < d/2$$

$$PE(pos, 2i) = \sin(pos/10000^{2i/dim})$$

$$PE(pos, 2i+1) = \cos(pos/10000^{2i/dim})$$

Credit: UvA DL Notebooks

```
class PositionalEncoding(nn.Module):

    def __init__(self, d_model, max_len=64):
        super().__init__()

        # Create matrix of [SeqLen, HiddenDim] representing the positional encoding
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() *
                             (-math.log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)

        # register_buffer => Tensor which is not a parameter, but should be
        # part of the modules state (should be on the same device).
        self.register_buffer('pe', pe, persistent=False)

    def forward(self, x):
        x = x + self.pe[:, :x.size(1)]
        return x
```

pos * (10000 ^ (- 1/ d))

nah karena properti

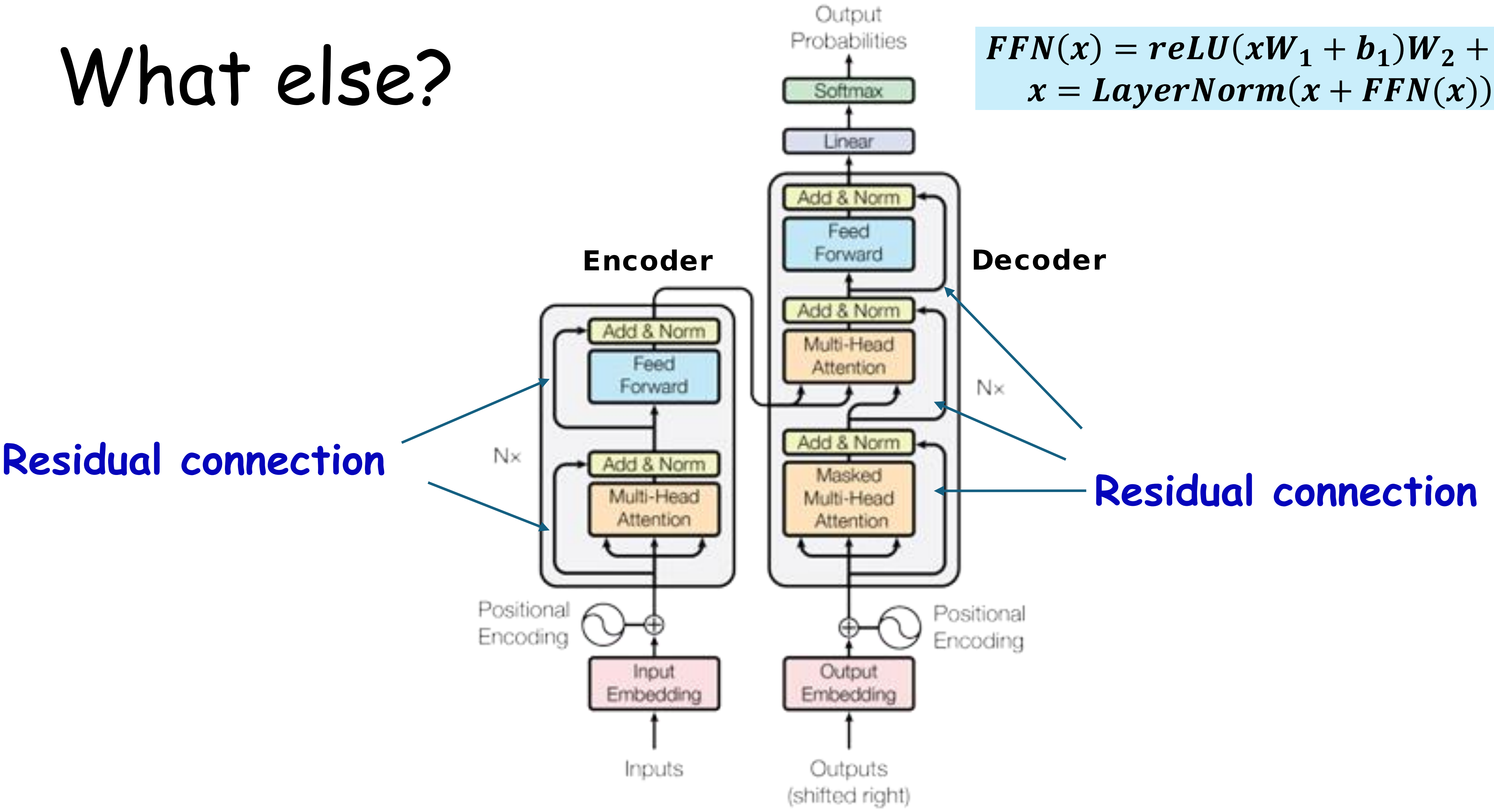
exp(log(x)) = x

maka -> kalau exp(log(pos * 10000 ^ (-1/2))) -> dimana
hasilnya akan jadi exp(-1/d) * log(10000)

Credit: Vaswani 2017

What else?

$$FFN(x) = reLU(xW_1 + b_1)W_2 + b_2$$
$$x = LayerNorm(x + FFN(x))$$



Credit: UvA DL Notebooks

Residual Connection

- Similar to ResNets, Transformers are designed to be very deep. Some models contain more than 24 blocks in the encoder. Hence, **the residual connections are crucial for enabling a smooth gradient flow through the model.**

informasi original sequencenya hilang karena

- **Without the residual connection, the information about the original sequence is lost**. Remember that the Multi-Head Attention layer ignores the position of elements in a sequence, and can only learn it based on the input features.

Credit: Vaswani 2017

What else?

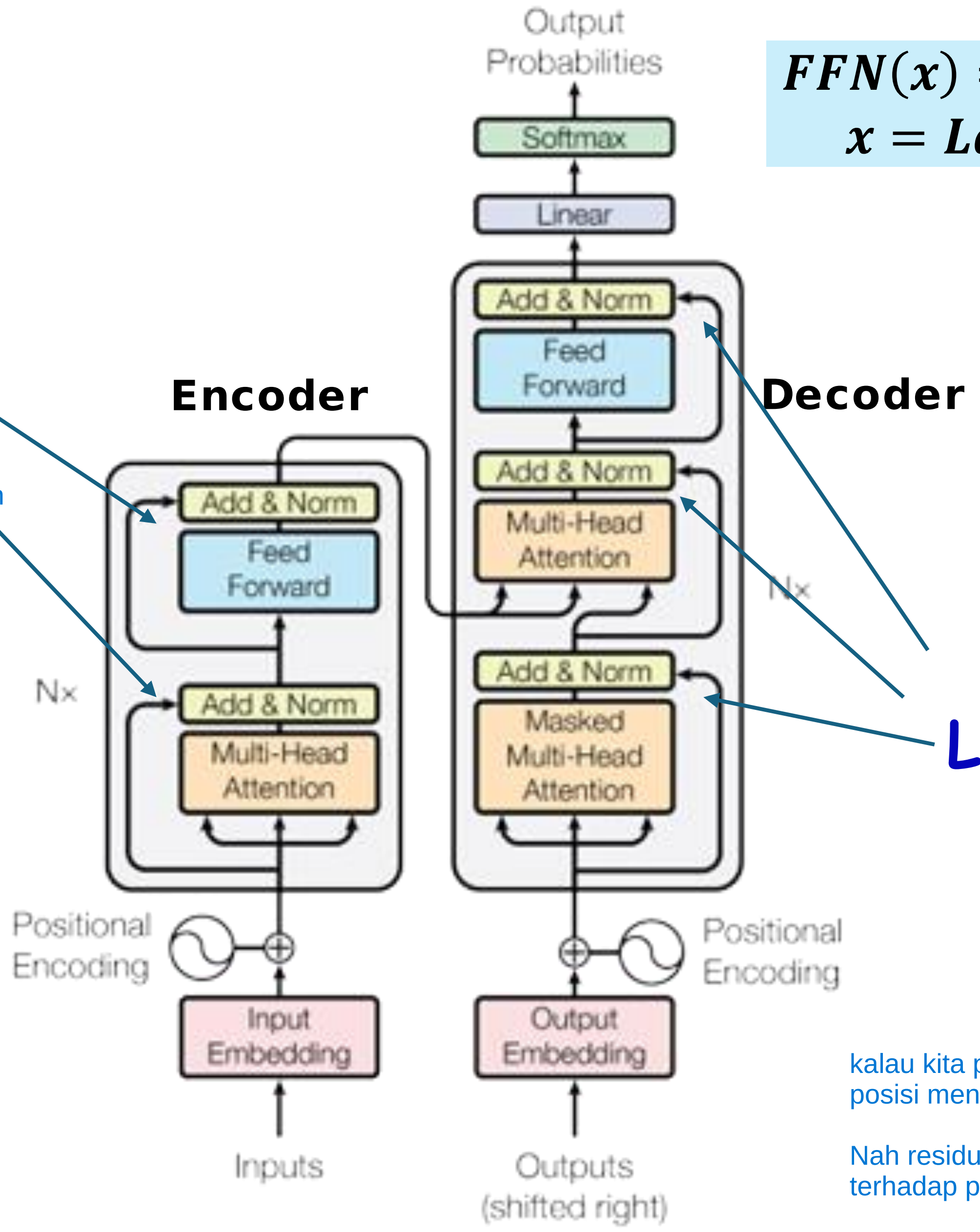
$$FFN(x) = reLU(xW_1 + b_1)W_2 + b_2$$
$$x = LayerNorm(x + FFN(x))$$

Layer Normalization

dengan normalization katanya bisa mencegah overfitting dan faster training

The Layer Normalization also plays an important role in the Transformer architecture as it enables **faster training** and provides **small regularization**.

Additionally, it ensures that the **features are in a similar magnitude among the elements in the sequence**.



kalau kita pakai multi-head attention, katanya informasi mengenai posisi menghilang.

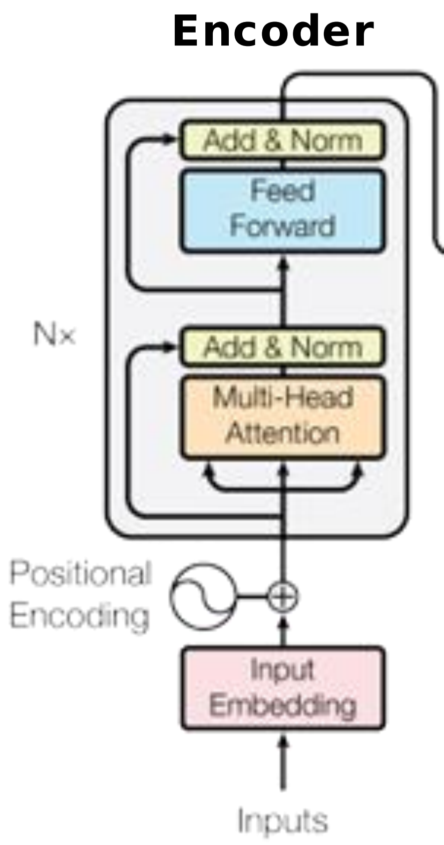
Nah residual connection ini membantu melakukan restoration terhadap posisi ini.

```python
class LayerNorm(nn.Module):

    def __init__(self, eps: float = 10**-6):
        super().__init__()
        self.eps = eps

        # We define alpha as a trainable parameter -> to scale input data
        self.alpha = nn.Parameter(torch.ones(1))

        # We define bias as a trainable parameter and initialize it with zeros
        self.bias = nn.Parameter(torch.zeros(1))

    def forward(self, x):
        mean = x.mean(dim = -1, keepdim = True) # Computing the mean
        std = x.std(dim = -1, keepdim = True) # Computing std

        # Returning the normalized input
        # We define epsilon as 0.000001 to avoid division by zero
        return self.alpha * (x - mean) / (std + self.eps) + self.bias
```

Credit: UvA DL Notebooks

```python
class EncoderBlock(nn.Module):

    def __init__(self, input_dim, num_heads, dim_feedforward, dropout=0.0):
        """
        Inputs:
            input_dim - Dimensionality of the input
            num_heads - Number of heads to use in the attention block
            dim_feedforward - Dimensionality of the hidden layer in the MLP
            dropout - Dropout probability to use in the dropout layers
        """
        super().__init__()

        # Attention layer
        self.self_attn = MultiheadAttention(input_dim, input_dim, num_heads)

        # Two-layer MLP
        self.linear_net = nn.Sequential(
            nn.Linear(input_dim, dim_feedforward),
            nn.Dropout(dropout),
            nn.ReLU(inplace=True),
            nn.Linear(dim_feedforward, input_dim)
        )
```

Dropout untuk mencegah overfitting (karena batch-batch awal biar gak overfit)

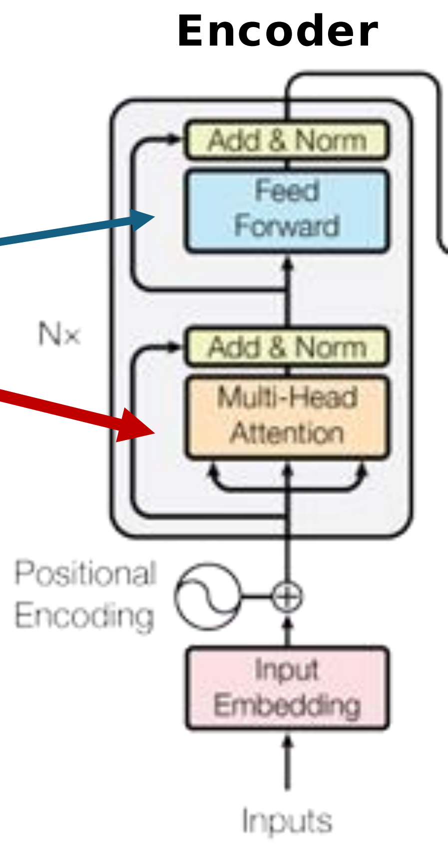


$$FFN(x) = reLU(xW_1 + b_1)W_2 + b_2$$
$$x = LayerNorm(x + FFN(x))$$

Credit: UvA DL Notebooks

```python
        # Layers to apply in between the main layers
        self.norm1 = LayerNorm(input_dim)
        self.norm2 = LayerNorm(input_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, mask=None):
        # Attention part
        attn_out = self.self_attn(x, mask=mask)
        x = x + self.dropout(attn_out)
        x = self.norm1(x)

        # MLP part
        linear_out = self.linear_net(x)
        x = x + self.dropout(linear_out)
        x = self.norm2(x)

        return x
```

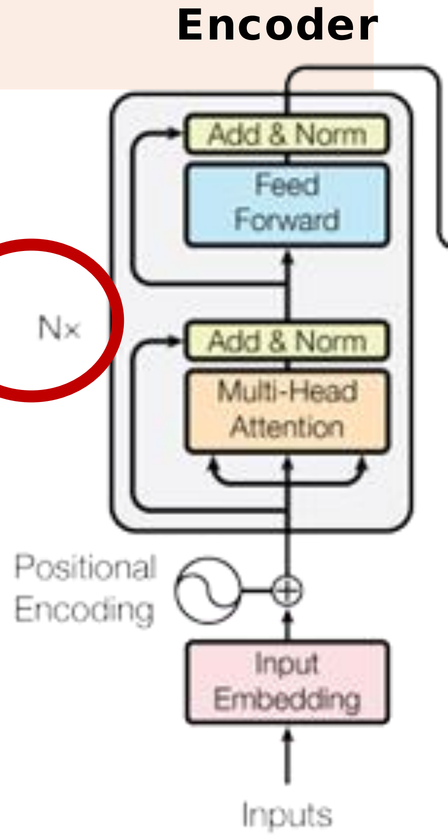
$$FFN(x) = reLU(xW_1 + b_1)W_2 + b_2$$
$$x = LayerNorm(x + FFN(x))$$



```python
#Transformer's Stack of Encoders
class EncoderStack(nn.Module):

    def __init__(self, num_layers, **block_args):
        super().__init__()
        self.layers = nn.ModuleList([EncoderBlock(**block_args) for _ in range(num_layers)])

    def forward(self, x, mask=None):
        for l in self.layers:
            x = l(x, mask=mask)
        return x
```

Credit: UvA DL Notebooks

Ada N buah tumpukan Encoder



Credit: UvA DL Notebooks

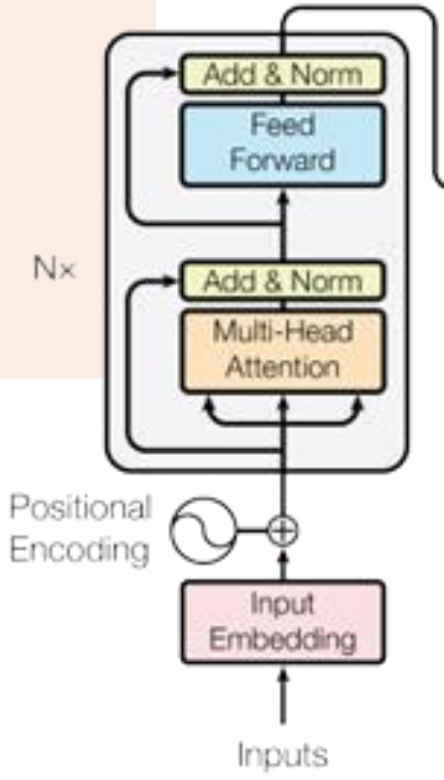
```python
class TransformerEncoder(nn.Module):
    def __init__(self, n_vocab, model_dim, num_heads, num_layers, dropout=0.0,
                                                                             input_dropout=0.0):
        """ Inputs:
            n_vocab - vocab size
            model_dim - Hidden dimensionality to use inside the Transformer
            num_heads - Number of heads to use in the Multi-Head Attention blocks
            num_layers - Number of encoder blocks to use.
            dropout - Dropout to apply inside the model
            input_dropout - Dropout to apply on the input features """
        super().__init__()

        # embedding layer for an input
        self.embedding = nn.Embedding(num_embeddings=n_vocab, embedding_dim=model_dim,
                                                                                  padding_idx=0)
        # positional embedding, the output should have the same size as the above embedding size
        self.positional_encoding = PositionalEncoding(d_model=model_dim)

        # stack of encoders
        self.trans_encoder = EncoderStack(num_layers=num_layers,
                                          input_dim=model_dim,
                                          dim_feedforward=2*model_dim,
                                          num_heads=num_heads,
                                          dropout=dropout)
```

Gabung semua bagian Encoder

Credit: UvA DL Notebooks

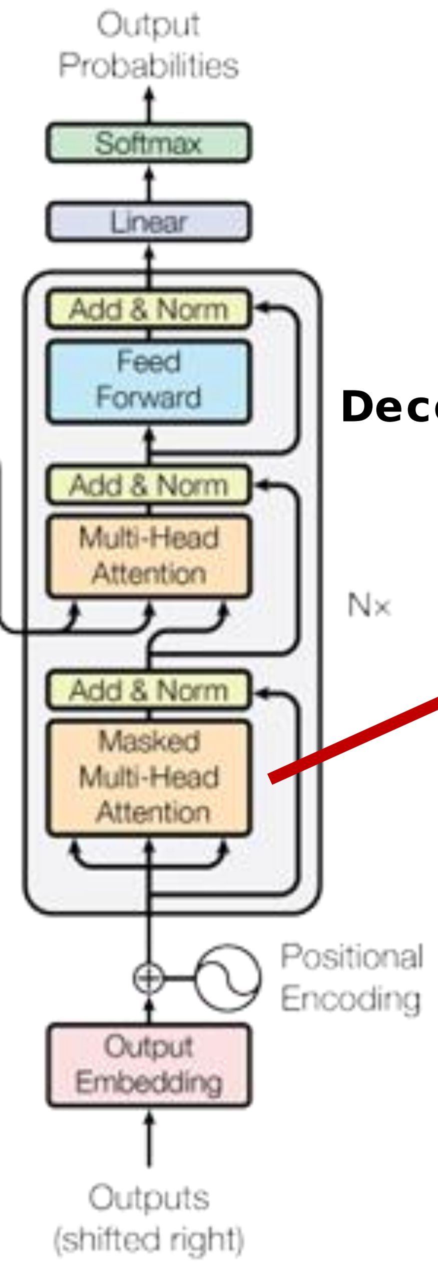
```python
def forward(self, x, mask=None):
    """
    Inputs:
        x - Input of shape [Batch, SeqLen]
        mask - Mask to apply on the attention outputs (optional)
    """
    embed = self.embedding(x) # embed = [Batch, SeqLen, model_dim]

    embed = self.positional_encoding(embed) # embed = [Batch, SeqLen, model_dim]

    # hidden = [Batch, SeqLen, model_dim]
    hidden = self.trans_encoder(embed, mask=mask)

    return hidden
```

Encoder



Gabung semua bagian Encoder

Decoder? Apa yang beda?



Decoder

This is not just a Self-Attention Layer. It's a **Masked Self-Attention Layer.**

Decoder is autoregressive, so a word is only influenced by its previous words.

upper triangle 0 karena tidak berpengaruh kata-kata selanjutnya.

Scores (before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

Apply Attention Mask

Masked Scores (before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

decoder digunakan untuk generate next token (jadi 1 kata 1 kata)

encoder itu dia pakai the whole kata

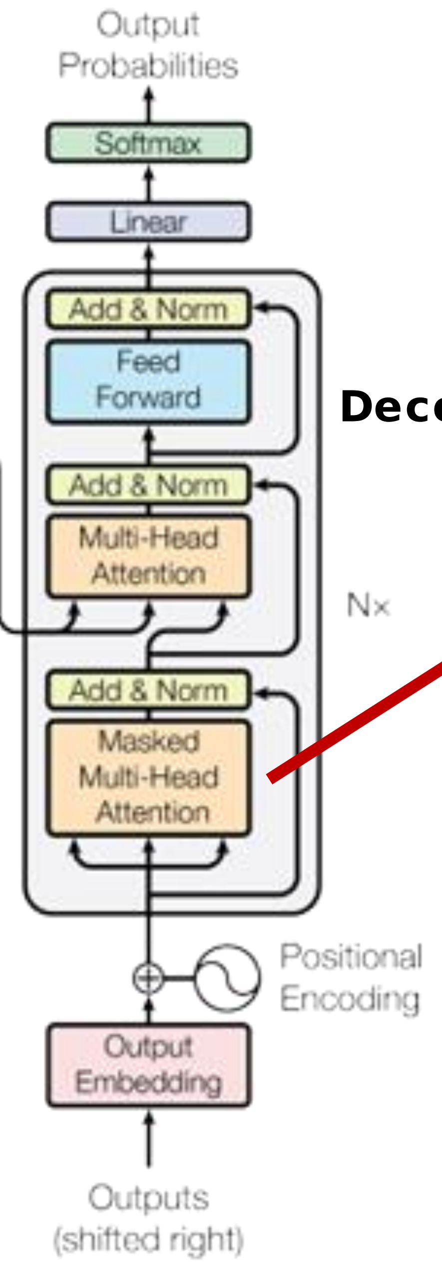
jadi kata-kata di depan gak ada pengaruh terhadap kata-kata sekarang

Softmax (along rows)

Scores

1	0	0	0
0.48	0.52	0	0
0.31	0.35	0.34	0
0.25	0.26	0.23	0.26

Decoder? Apa yang beda?



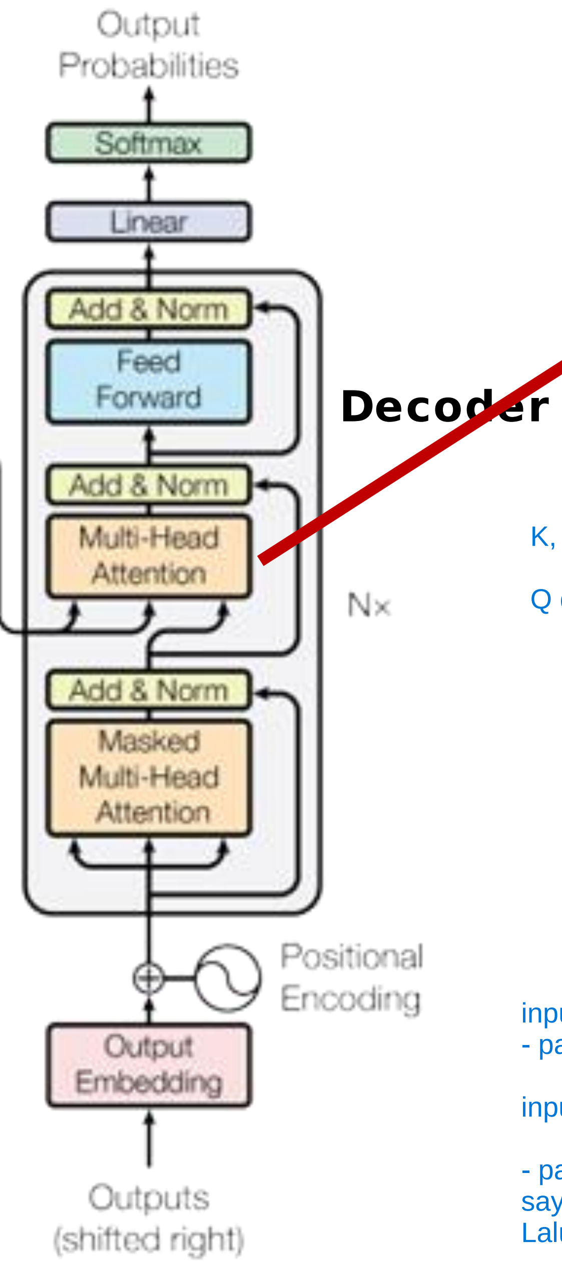
Decoder

Class **MultiheadAttention** tidak perlu diubah; yang perlu dilakukan adalah kita berikan input **mask khusus** ke decoder.

```
seq_len = 3

mask_ = (torch.triu(torch.ones(seq_len, seq_len),
                    diagonal = 1).type(torch.int) == 0)
```

```
tensor([[1, 0, 0],
        [1, 1, 0],
        [1, 1, 1]])
```

Decoder? Apa yang beda?



Cross-Attention Encoder-Decoder

Q → dari Decoder → [batch, **seq len target**, hidden dim]

K → dari Encoder → [batch, **seq len source**, hidden dim]
V → dari Encoder → [batch, **seq len source**, hidden dim]

K, V dari output encoder

Q dari input decoder

Target Sentence

	Budi	membeli	buku	lalu	ia	membaca	nya
Budi	0.35	0.11	0.06	0.11	0.20	0.11	0.06
buys							
book	0.09	0.09	0.48	0.05	0.05	0.09	0.15
then							
he	0.18	0.10	0.10	0.10	0.32	0.10	0.10
reads							
it	0.10	0.10	0.30	0.05	0.10	0.05	0.30

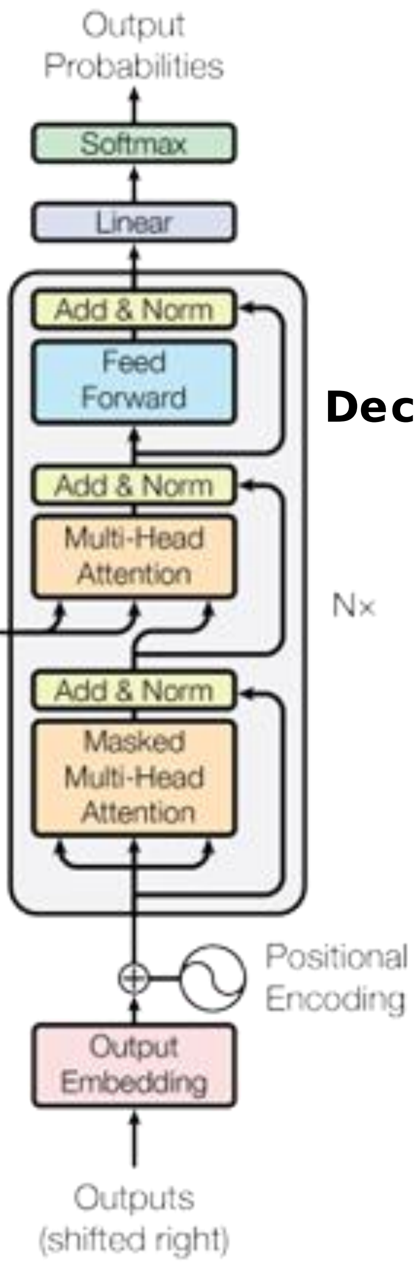
Source Sentence

$$softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

input decoder ada 2 hal:
- pada saat training:

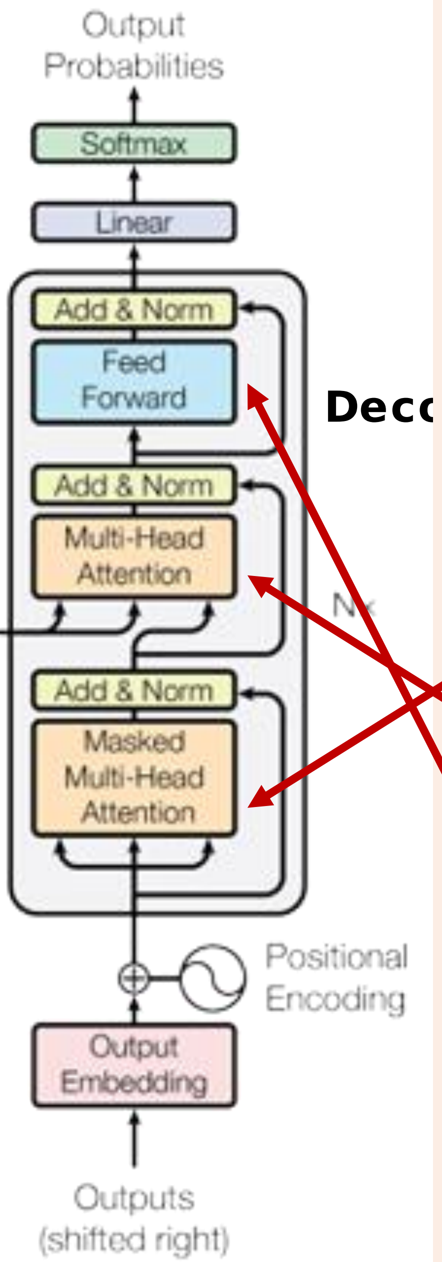
inputnya decoding itu I go to depok

- pada saat inference
saya pergi ke depok (input encoder), output encodernya ke decoder. Lalu input decodernya itu sebenarnya SOS (Start of Sentence).



Decoder

```
class DecoderBlock(nn.Module):

    def __init__(self, input_dim, num_heads, dim_feedforward, dropout=0.0):
        """
        Inputs:
            input_dim - Dimensionality of the input
            num_heads - Number of heads to use in the attention block
            dim_feedforward - Dimensionality of the hidden layer in the MLP
            dropout - Dropout probability to use in the dropout layers
        """
        super().__init__()

        # Attention layer
        self.self_attn = MultiheadAttention(input_dim, input_dim, num_heads)
        self.cross_attn = MultiheadAttention(input_dim, input_dim, num_heads)

        # Two-layer MLP
        self.linear_net = nn.Sequential(
            nn.Linear(input_dim, dim_feedforward),
            nn.Dropout(dropout),
            nn.ReLU(inplace=True),
            nn.Linear(dim_feedforward, input_dim)
        )
```



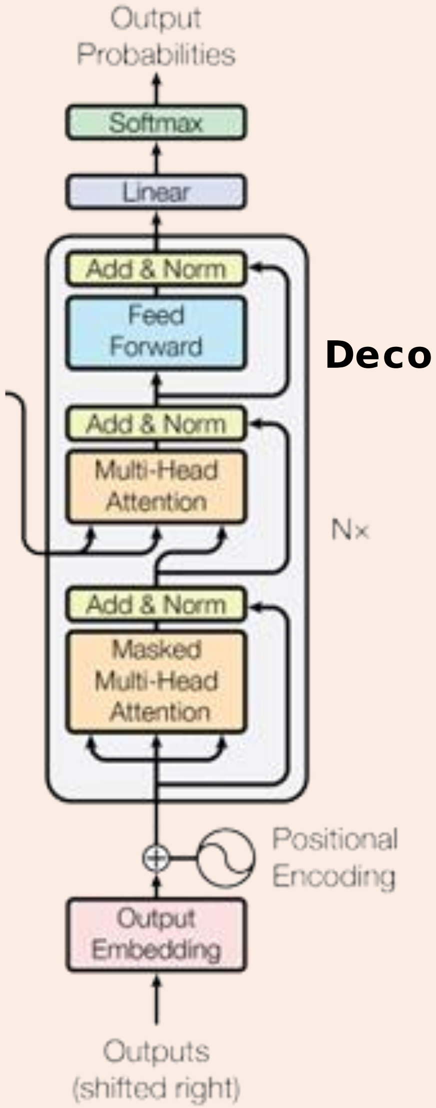

```python
    # Layers to apply in between the main layers
    self.norm1 = LayerNorm(input_dim)
    self.norm2 = LayerNorm(input_dim)
    self.norm3 = LayerNorm(input_dim)
    self.dropout = nn.Dropout(dropout)

def forward(self, x_dec_in, x_enc_out, mask_target=None, mask_source=None):
    # Masked Self Attention part
    attn_out = self.self_attn(x_dec_in, mask=mask_target)
    x = x_dec_in + self.dropout(attn_out)
    x = self.norm1(x)

    # Cross Attention
    cross_out = self.cross_attn(x, y=x_enc_out, mask=mask_source)
    x = x + self.dropout(cross_out)
    x = self.norm2(x)

    # MLP part
    linear_out = self.linear_net(x)
    x = x + self.dropout(linear_out)
    x = self.norm3(x)

    return x
```

```python
class DecoderStack(nn.Module):

    def __init__(self, num_layers, **block_args):
        super().__init__()
        self.layers = nn.ModuleList([DecoderBlock(**block_args) for _ in range(num_layers)])

    def forward(self, x, x_enc_out, mask_target=None, mask_source=None):
        for l in self.layers:
            x = l(x, x_enc_out, mask_target=mask_target, mask_source=mask_source)
        return x
```

Tumpukan N buah decoder

```python
class TransformerDecoder(nn.Module):

    def __init__(self, n_vocab, model_dim, num_heads, num_layers,
                                             dropout=0.0, input_dropout=0.0):
        """
        Inputs:
            n_vocab - vocab size
            model_dim - Hidden dimensionality to use inside the Transformer
            num_heads - Number of heads to use in the Multi-Head Attention blocks
            num_layers - Number of encoder blocks to use.
            dropout - Dropout to apply inside the model
            input_dropout - Dropout to apply on the input features
        """
        super().__init__()

        # embedding layer for an input
        self.embedding = nn.Embedding(num_embeddings=n_vocab,
                                             embedding_dim=model_dim, padding_idx=0)

        # positional embedding, the output should have the same size as
        # the above embedding size
        self.positional_encoding = PositionalEncoding(d_model=model_dim)
```

```python
        # stack of encoders
        self.trans_decoder = DecoderStack(num_layers=num_layers,
                                          input_dim=model_dim,
                                          dim_feedforward=2*model_dim,
                                          num_heads=num_heads,
                                          dropout=dropout)

        # final projection to Vocab dim
        self.proj = nn.Linear(model_dim, n_vocab)

    def forward(self, x, x_enc_out, mask_target=None, mask_source=None):
        """
        Inputs:
            x - Input of shape [Batch, SeqLen]
            x_enc_out - Encoder output of shape [Batch, SeqLen, model_dim]
            mask_target - Mask to apply on the targets' attention (optional)
            mask_source - Mask to apply on the cross attention (optional)
        """
        embed = self.embedding(x) # embed = [Batch, SeqLen, model_dim]
        embed = self.positional_encoding(embed) # embed = [Batch, SeqLen, model_dim]
        hidden = self.trans_decoder(embed, x_enc_out, mask_target=mask_target,
                                    mask_source=mask_source) # hidden = [Batch, SeqLen, model_dim]
        logits = self.proj(hidden) # logits = [Batch, SeqLen, n_vocab]
        return logits
```



Decoder

Put-it-all-together

Training a Transformer for Machine Translation

```python
class Transformer(nn.Module):

    def __init__(self, n_vocab_src, n_vocab_tgt,
                 model_dim=32, num_heads=4, num_layers=2, dropout=0.1, input_dropout=0.1):

        super().__init__()
        self.encoder = TransformerEncoder(n_vocab_src, model_dim, num_heads,
                                          num_layers, dropout, input_dropout)

        self.decoder = TransformerDecoder(n_vocab_tgt, model_dim, num_heads,
                                          num_layers, dropout, input_dropout)

        # initialization
        for p in self.parameters():
          if p.dim() > 1:
              nn.init.xavier_uniform_(p)

    def forward(self, src_doc, tgt_doc, mask_target=None, mask_source=None):
        x_enc_out = self.encoder(src_doc, mask=mask_source)

        x_dec_out = self.decoder(tgt_doc, x_enc_out, mask_target=mask_target,
                                                      mask_source=mask_source)
        return x_dec_out
```

Transformer = Encoder + Decoder

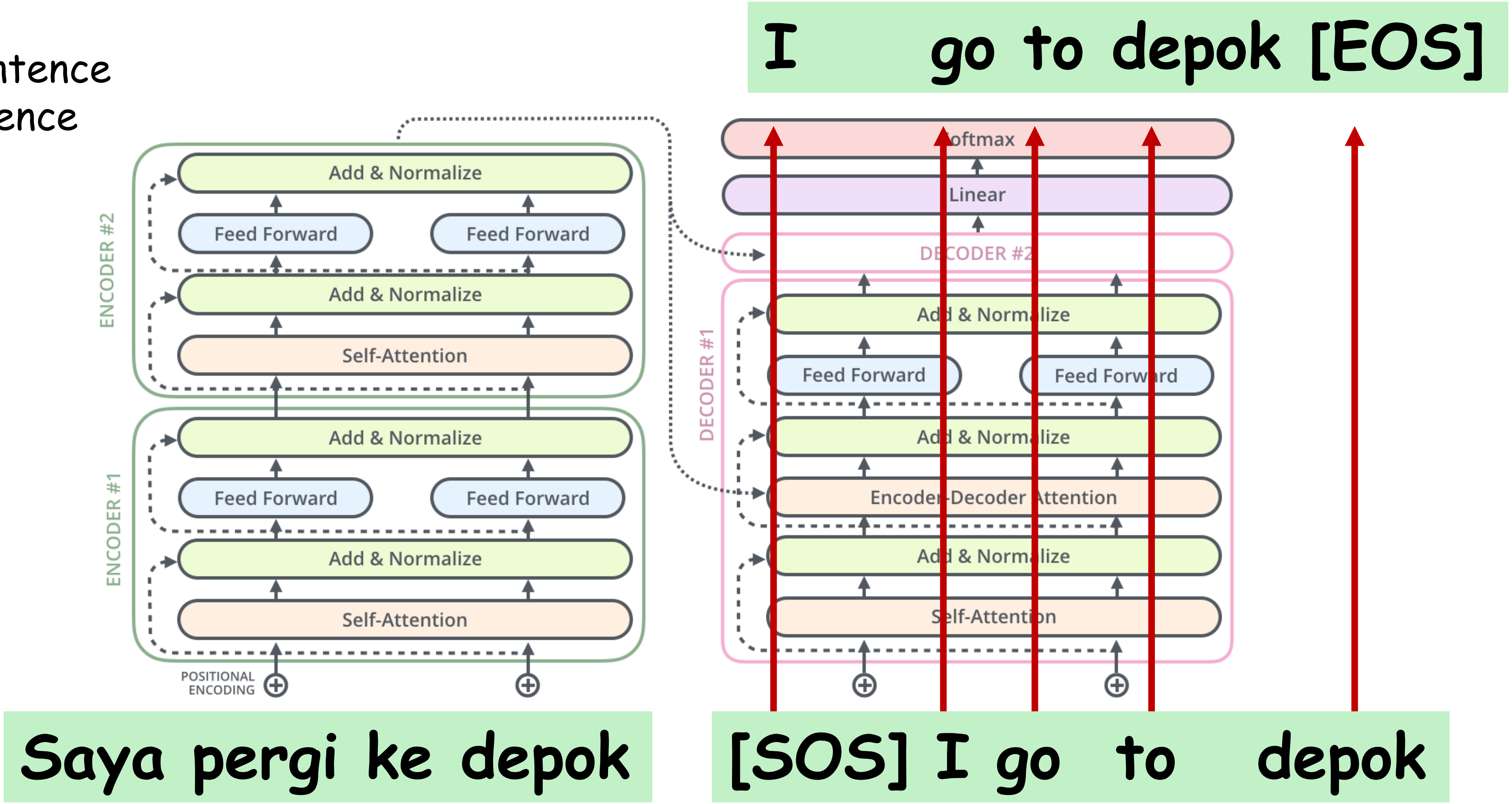
Training a Transformer

Format Dataset: [(source sentence, target sentence, label), …]

label adalah target sentence yang digeser ke kanan 1 step

ini gambaran transformer

[SOS]: start of sentence
[EOS]: end of sentence



```python
doc_ina = ["saya pergi ke depok",
           "saya pergi dengan mobil",
           "saya datang ke depok",
           "kamu pergi dengan motor",
           "kamu datang dengan mobil",
           "saya pergi dan kamu datang",
           "kamu datang",
           "saya datang",
           "kamu pergi ke depok dengan motor"]

doc_eng = ["i go to depok",
           "i go with a car",
           "i come to depok",
           "you go with a motorcycle",
           "you come with a car",
           "i go and you come",
           "you come",
           "i come",
           "you go to depok with a motorcycle"]
```

```python
def index_word(uniq_words):
    index_to_word = {(index + 3): word for index, word in enumerate(uniq_words)}
    word_to_index = {word: (index + 3) for index, word in enumerate(uniq_words)}
    index_to_word[0] = "[PAD]"
    word_to_index["[PAD]"] = 0
    index_to_word[1] = "[SOS]"
    word_to_index["[SOS]"] = 1
    index_to_word[2] = "[EOS]"
    word_to_index["[EOS]"] = 2
    return index_to_word, word_to_index

def get_uniq_words(words):
    word_counts = Counter(words)
    return sorted(word_counts, key=word_counts.get, reverse=True)

def tokenize(text):
    return text.split(' ')

def load_words(documents):
    text = ""
    for doc in documents:
        text += doc + " "
    return tokenize(text)
```

```python
class Dataset(torch.utils.data.Dataset):
    def __init__(self, sequence_length, source_docs, target_docs):
        self.sequence_length = sequence_length
        self.source_docs = source_docs
        self.target_docs = target_docs
        self.source_words = load_words(source_docs)
        self.source_uniq_words = get_uniq_words(self.source_words)
        self.target_words = load_words(target_docs)
        self.target_uniq_words = get_uniq_words(self.target_words)

        self.src_index_to_word, self.src_word_to_index = index_word(self.source_uniq_words)
        self.tgt_index_to_word, self.tgt_word_to_index = index_word(self.target_uniq_words)

    def __len__(self):
        return len(self.source_docs)

    def __getitem__(self, index):
        # encoder input
        src_doc = [self.src_word_to_index[w] for w in tokenize(self.source_docs[index])]
        src_doc = src_doc[:self.sequence_length - 2]
        src_doc = [self.src_word_to_index['[SOS]']] + src_doc +
                                                        [self.src_word_to_index['[EOS]']]
        src_doc += ([self.src_word_to_index['[PAD]']] * (self.sequence_length - len(src_doc)))
        src_doc = torch.tensor(src_doc)
```

```python
        # decoder input
        tgt_doc = [self.tgt_word_to_index[w] for w in tokenize(self.target_docs[index])]
        tgt_doc = tgt_doc[:self.sequence_length - 1]
        tgt_doc = [self.tgt_word_to_index['[SOS]']] + tgt_doc
        tgt_doc += ([self.tgt_word_to_index['[PAD]']] * (self.sequence_length - len(tgt_doc)))
        tgt_doc = torch.tensor(tgt_doc)

        # decoder output or LABEL
        label = [self.tgt_word_to_index[w] for w in tokenize(self.target_docs[index])]
        label = label[:self.sequence_length - 1]
        label = label + [self.tgt_word_to_index['[EOS]']]
        label += ([self.tgt_word_to_index['[PAD]']] * (self.sequence_length - len(label)))
        label = torch.tensor(label)

        mask_ = (torch.triu(torch.ones(self.sequence_length, self.sequence_length),
                                                  diagonal = 1).type(torch.int) == 0)
        # 0 adalah [PAD]
        encoder_mask = (src_doc != torch.tensor(self.src_word_to_index['[PAD]'])).unsqueeze(0)
        decoder_mask = (tgt_doc != torch.tensor(self.tgt_word_to_index['[PAD]'])).unsqueeze(0)
        decoder_mask = decoder_mask & mask_

        return { 'encoder_input': src_doc, 'decoder_input': tgt_doc, 'label': label,
                 'encoder_mask': encoder_mask.int(), 'decoder_mask': decoder_mask.int() }
```

ini yang upper triangle di masking

Learning Rate warm-up

Credit: UvA DL Notebooks

- One commonly used technique for training a Transformer is learning rate warm-up. This means that we **gradually increase the learning rate from 0 on to our originally specified learning rate in the first few iterations**.

arsitektur yang parameter sangat besar, itu berbahya kalau pakai optimizer biasa (kayak gradien descent, adam optimizer, etc). Nah mereka biasa loss nya itu gak turun-turun.

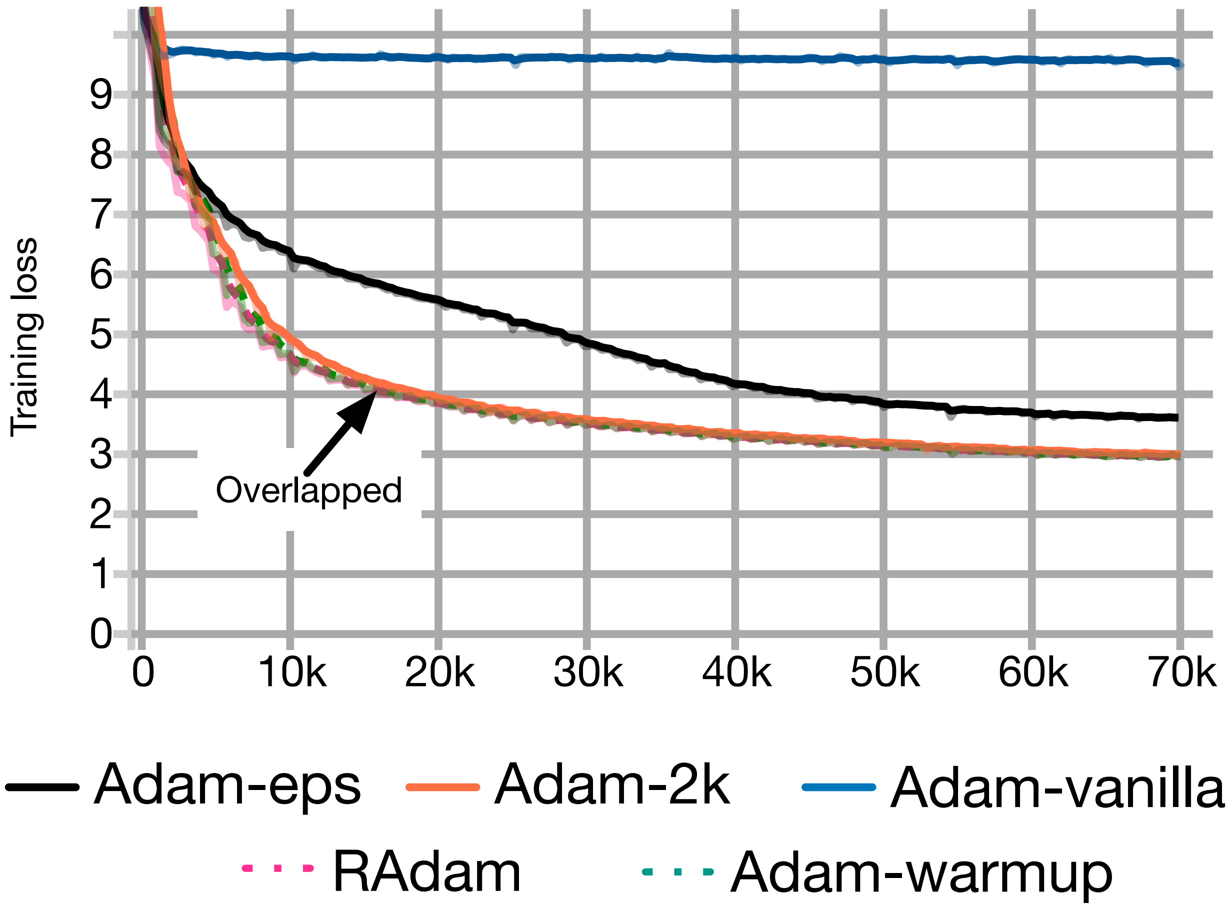
- Thus, we slowly start learning instead of taking very large steps from the beginning.

Jadi warm up itu learning ratenya itu di ramp up sampai 0.1 dari 0. Nanti turun pelan-pelan

- In fact, training a deep Transformer **without learning rate warm-up** can **make the model diverge and achieve a much worse performance on training and testing**.

Learning Rate warm-up

Credit: UvA DL Notebooks



Learning Rate warm-up

Credit: UvA DL Notebooks

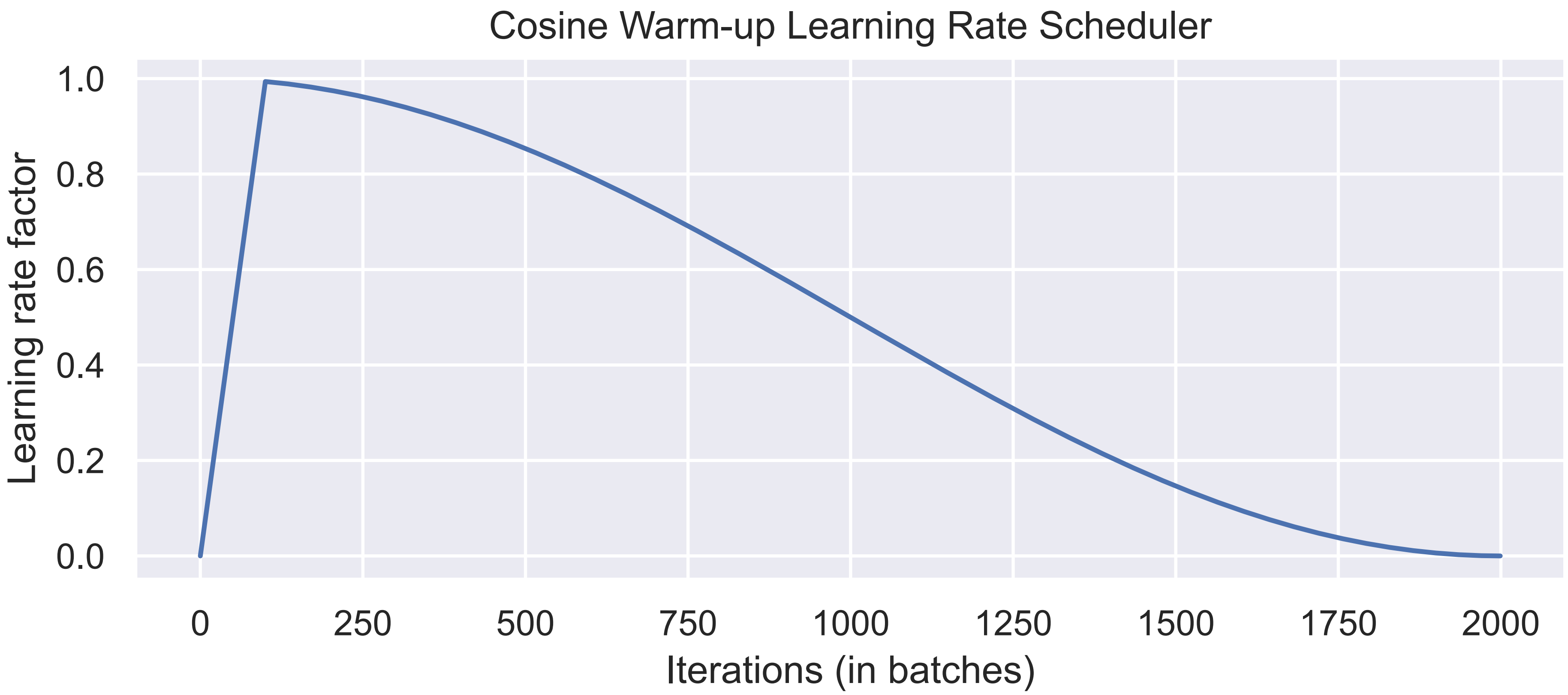
the currently most popular scheduler is the **cosine warm-up scheduler**, which combines warm-up with a cosine-shaped learning rate decay.

```
class CosineWarmupScheduler(optim.lr_scheduler._LRScheduler):

    def __init__(self, optimizer, warmup, max_iters):
        self.warmup = warmup
        self.max_num_iters = max_iters
        super().__init__(optimizer)

    def get_lr(self):
        lr_factor = self.get_lr_factor(epoch=self.last_epoch)
        return [base_lr * lr_factor for base_lr in self.base_lrs]

    def get_lr_factor(self, epoch):
        lr_factor = 0.5 * (1 + np.cos(np.pi * epoch / self.max_num_iters))
        if epoch <= self.warmup:
            lr_factor *= epoch * 1.0 / self.warmup
        return lr_factor
```

Learning Rate warm-up

Credit: UvA DL Notebooks

the currently most popular scheduler is the **cosine warm-up scheduler**, which combines warm-up with a cosine-shaped learning rate decay.

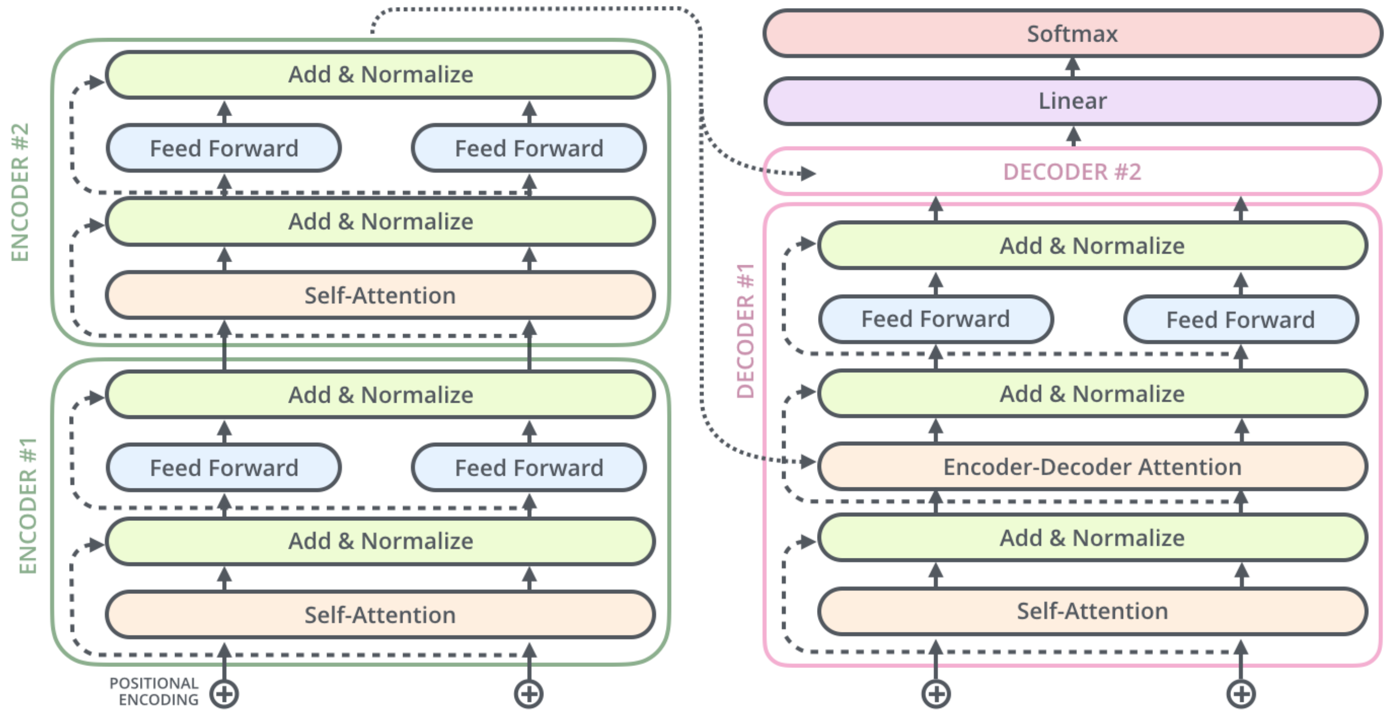


```python
def train(dataset, model, batch_size=3, warm_up=False, max_epochs=50, lr=0.001):
    model.to(device)
    model.train()
    data_loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size)

    # ignore [PAD] in the decoder's softmax output
    criterion = nn.CrossEntropyLoss(ignore_index = dataset.tgt_word_to_index['[PAD]']).to(device)

    optimizer = optim.Adam(model.parameters(), lr=lr)

    if warm_up:
        lr_scheduler = CosineWarmupScheduler(optimizer=optimizer, warmup=100,
                                             max_iters=max_epochs*batch_size)

    for epoch in range(max_epochs):
        losses = []
        for batch in data_loader:
            encoder_input = batch['encoder_input'].to(device)
            decoder_input = batch['decoder_input'].to(device)
            label = batch['label'].to(device)
            encoder_mask = batch['encoder_mask'].to(device)
            decoder_mask = batch['decoder_mask'].to(device)
```

```
        out = model(encoder_input, decoder_input, mask_target=decoder_mask,
                                                  mask_source=encoder_mask)

        # out: [Batch, SeqLen, n_vocab]
        # label: [Batch, SeqLen]
        # first, we need to transpose dim 1 dan 2 on out, so that it
        # becomes [Batch, n_vocab, SeqLen]

        loss = criterion(out.transpose(1, 2), label)

        losses.append(loss.detach().item())
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if warm_up:
            lr_scheduler.step()

    if (epoch + 1) % 100 == 0:
        print(f"Epoch {epoch+1}: Loss = {sum(losses)/len(losses)}")
```

Inference

0) Berikan input **kalimat sumber** ke encoder

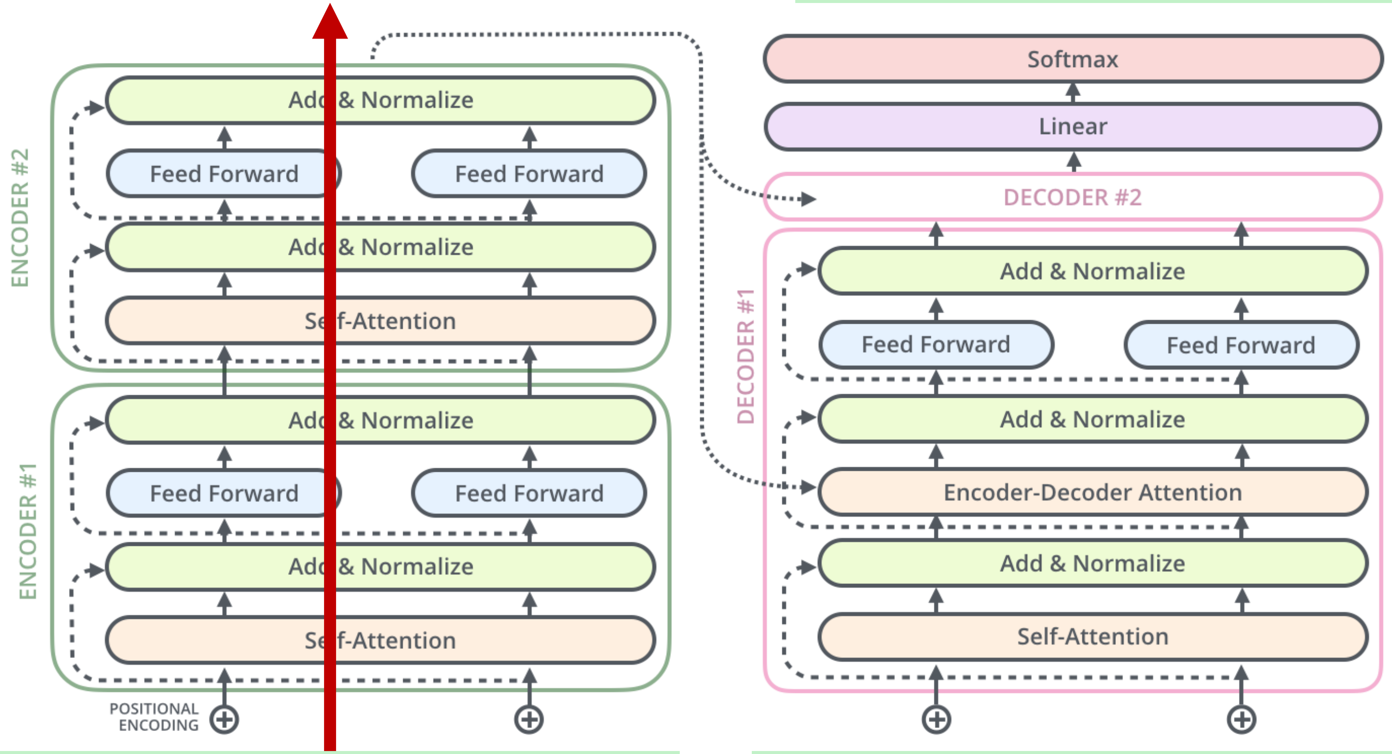


Saya pergi ke depok

Inference

1) Lakukan proses forward pada Encoder

Tensor **[batch, seq len, hidden dim]**

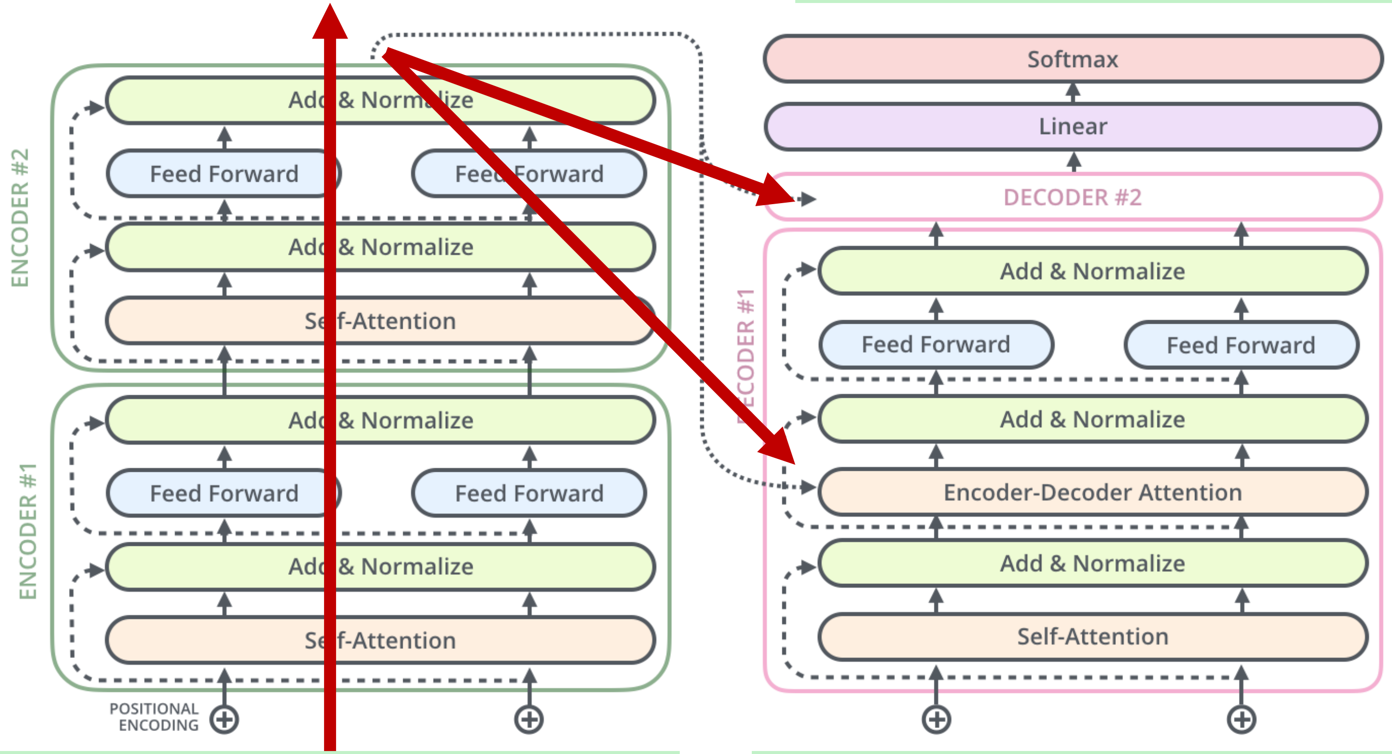


Saya pergi ke depok

Inference

2) Pass tensor output encoder ke cross-attention layer di decoder

Tensor **[batch, seq len, hidden dim]**

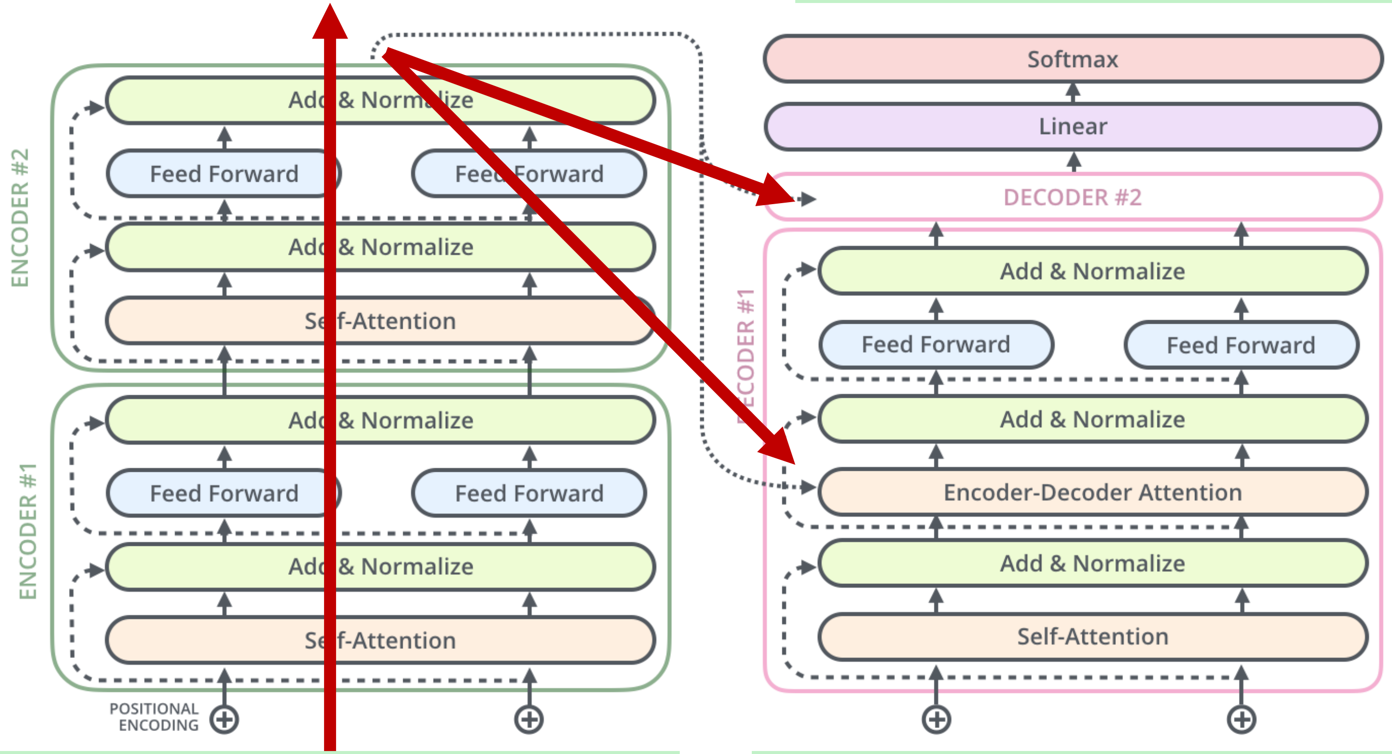


Saya pergi ke depok

Inference

3) Lakukan proses sequence generation dimulai dari token **[SOS]**

Tensor **[batch, seq len, hidden dim]**

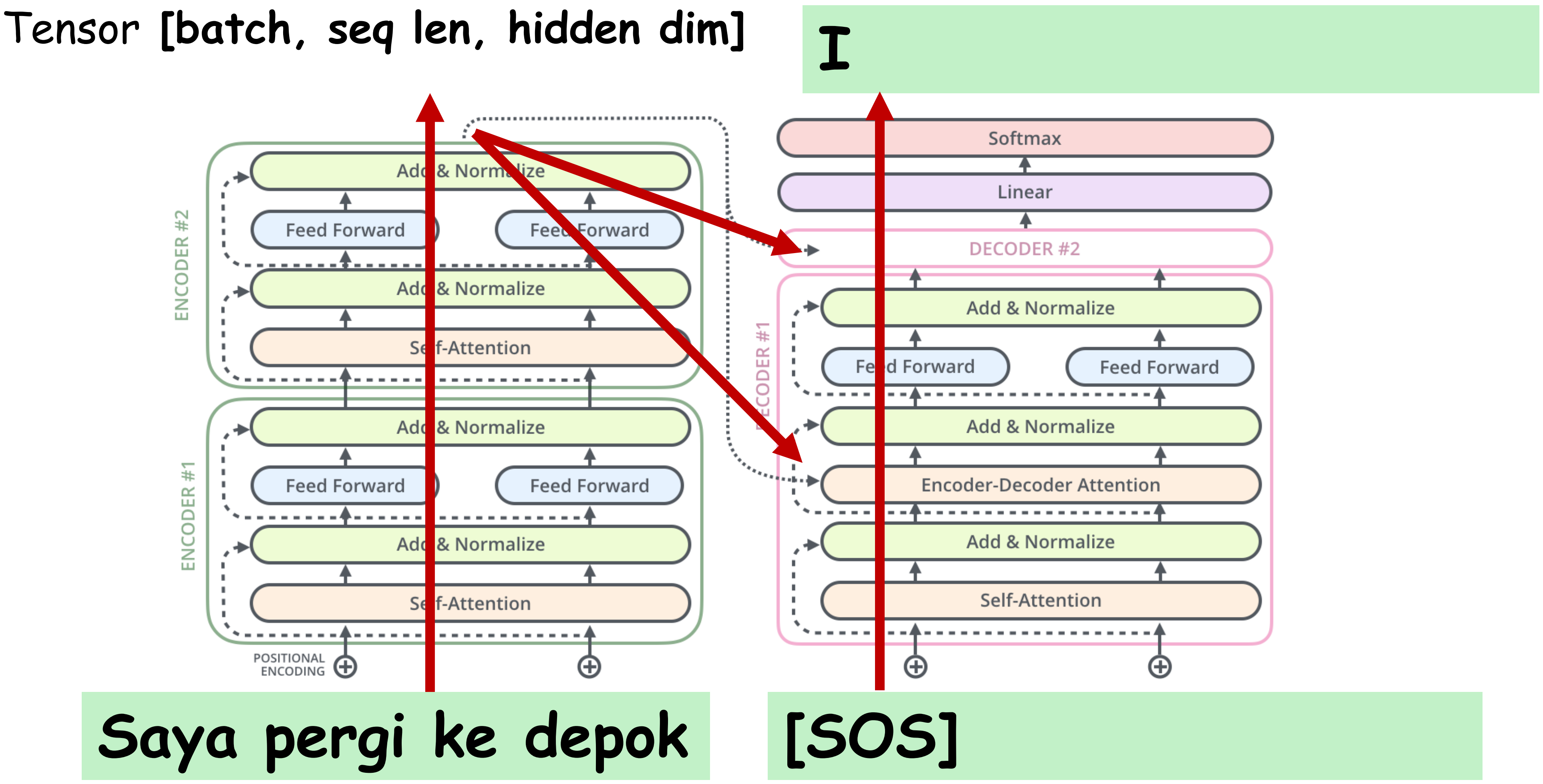


Saya pergi ke depok

[SOS]

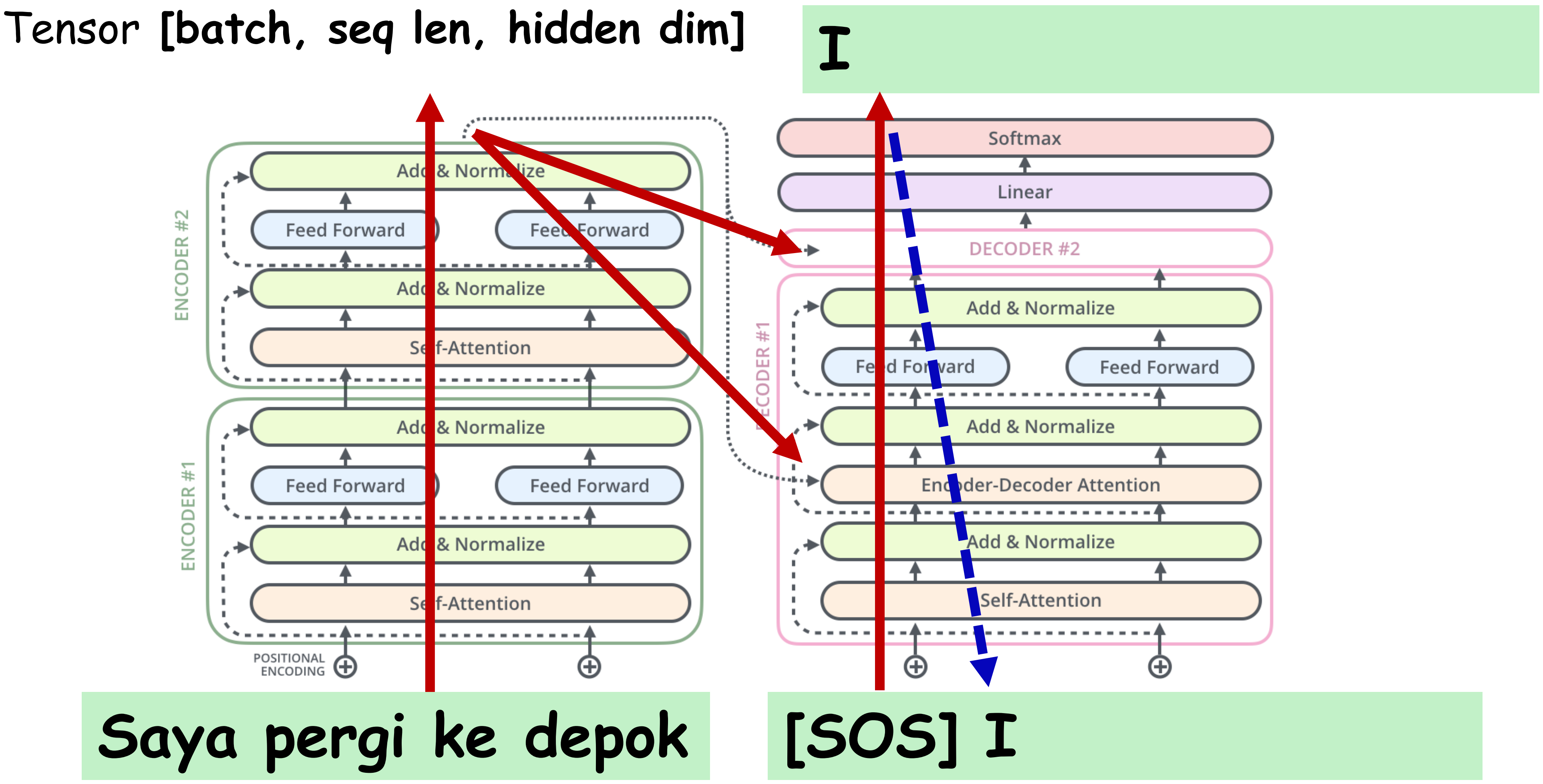
Inference

4) Predict **next token**: ambil vektor output **paling kanan**, lalu pilih **kata dengan probabilitas tertinggi** (**greedy decoding strategy**)



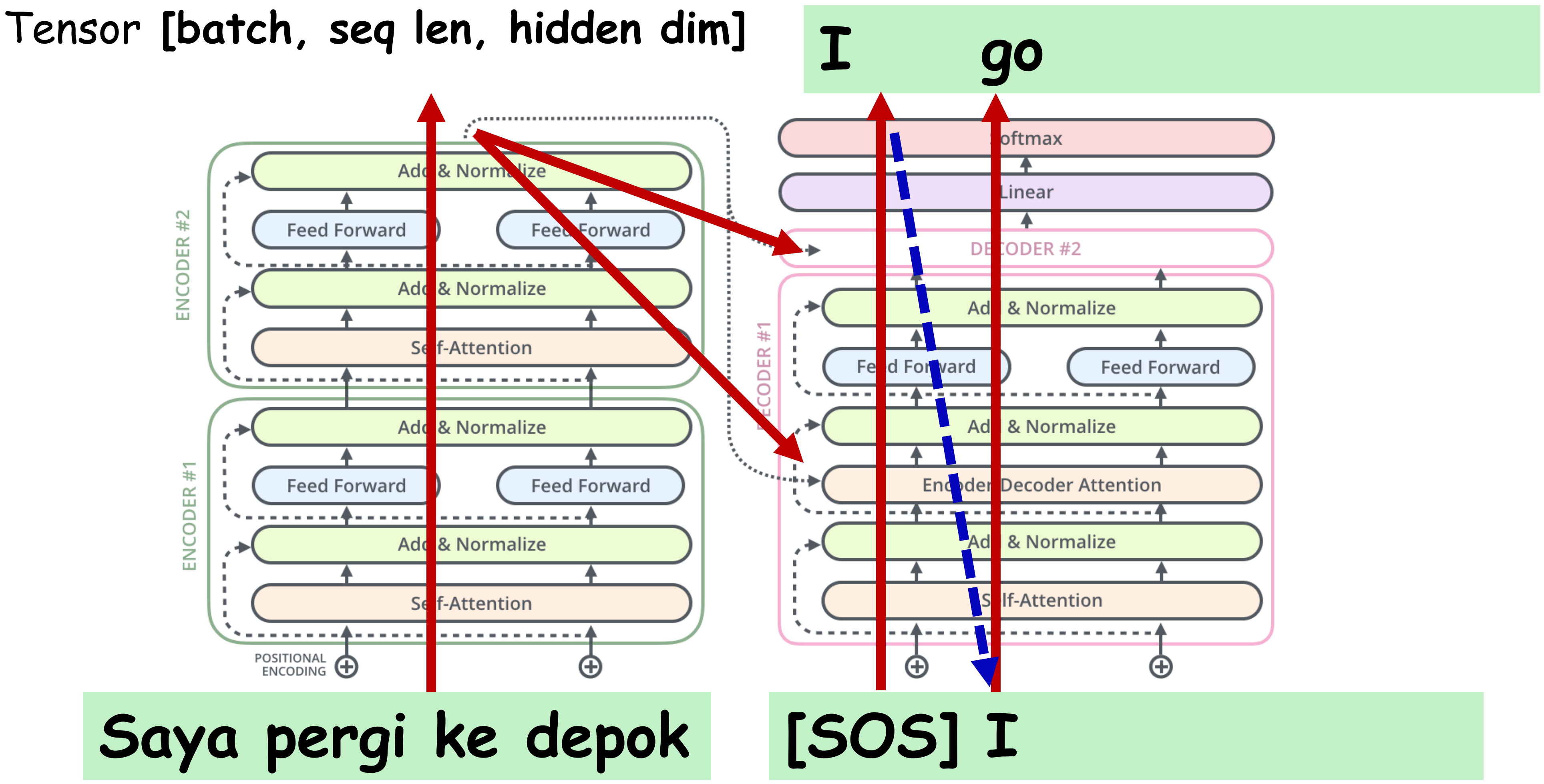
Inference

5) *Predicted token* ditambahkan ke input decoder, lalu ulangi proses hingga bertemu **[EOS]** atau mencapai panjang tertentu **(max len)**.



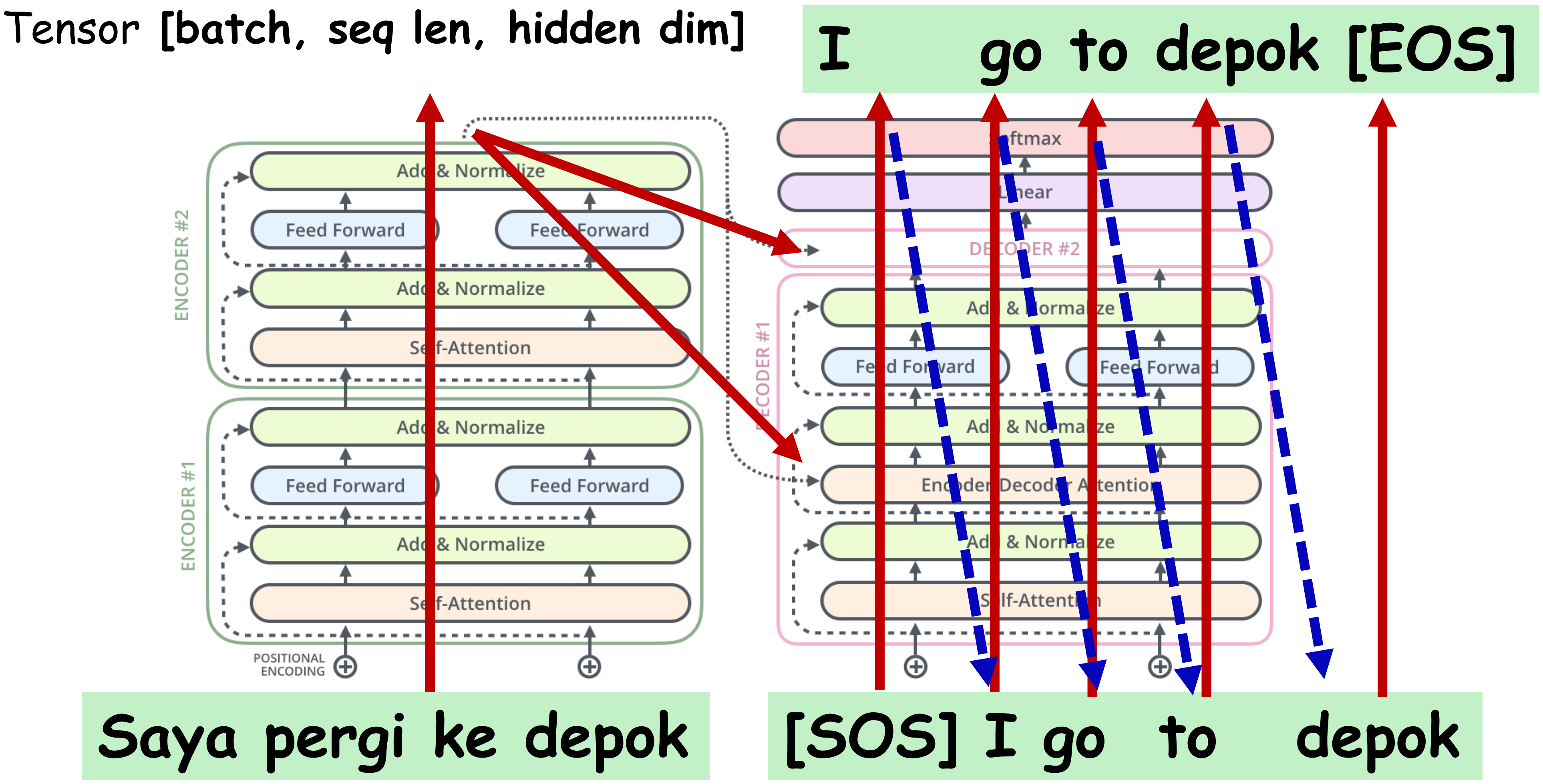
Inference

5) *Predicted token* ditambahkan ke input decoder, lalu ulangi proses hingga bertemua **[EOS]** atau mencapai panjang tertentu **(max len)**.



Inference

5) *Predicted token* ditambahkan ke input decoder, lalu ulangi proses hingga bertemua **[EOS]** atau mencapai panjang tertentu **(max len)**.



```python
def translate_a_doc(doc_ina, model, src_word_to_index, tgt_index_to_word, max_len=12):
    """ only for an instance, not a batch of instances """
    model.eval()
    encoder_input = [1] + [src_word_to_index[w] for w in tokenize(doc_ina)] + [2]
    encoder_input = torch.tensor(encoder_input).unsqueeze(0).int().to(device)

    encoder_mask = (encoder_input != torch.tensor(0)) # 0 is [PAD]

    # fill with 1, 1 is [SOS]
    decoder_input = torch.tensor([[1]]).int().to(device)

    # run the encoder and produce an output
    encoder_output = model.encoder(encoder_input, mask=encoder_mask)

    while True:
        output_seq_len = decoder_input.size(1)
        if output_seq_len == max_len:
            break

        mask_ = (torch.triu(torch.ones(1, output_seq_len, output_seq_len),
                                                diagonal = 1).type(torch.int) == 0).to(device)
```

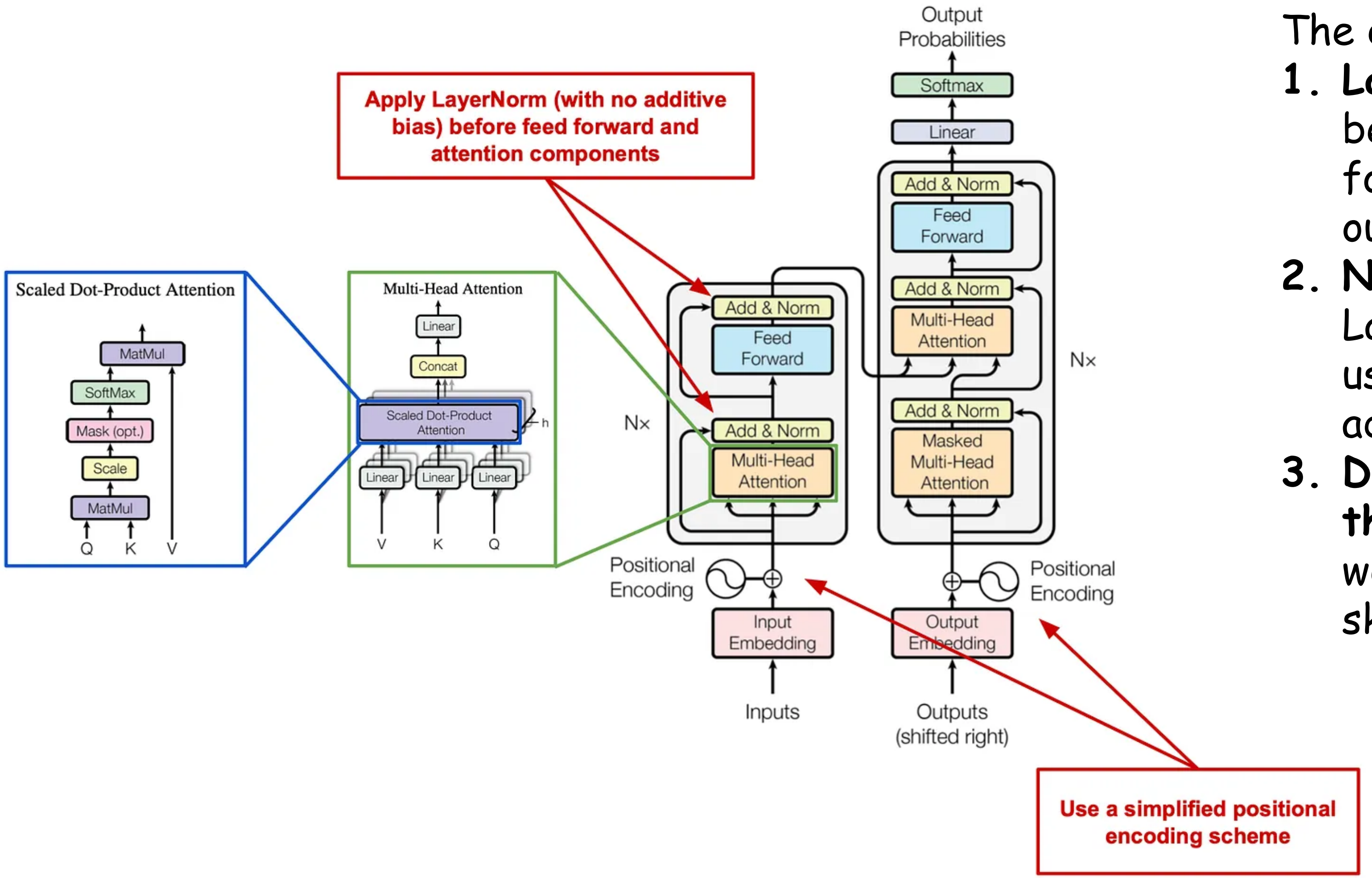
```
        decoder_mask = (decoder_input != torch.tensor(0)).to(device) # 0 is [PAD]
        decoder_mask = decoder_mask & mask_

        decoder_output = model.decoder(decoder_input, encoder_output,
                                 mask_target=decoder_mask, mask_source=encoder_mask)

        prob = decoder_output[:, -1] # last token

        # Selecting token with the highest probability (Greedy strategy)
        _, next_word = torch.max(prob, dim=1)

        # Combine the predicted token to last position of input sequence
        decoder_input = torch.cat([decoder_input,
                                torch.tensor([[next_word.item()]]).int().to(device)], dim=1)

        if next_word == 2: # if [EOS], 2 = [EOS]
            break

    decoder_input = decoder_input.squeeze()
    words = [tgt_index_to_word[id.item()] for id in decoder_input]   # token id to word
    return words
```

```python
with torch.no_grad():
    out = translate_a_doc("kamu datang dengan mobil", model,
                                    data.src_word_to_index, data.tgt_index_to_word)
    print(out)

    # ['[SOS]', 'you', 'come', 'with', 'a', 'car', '[EOS]']
```

Link Google Collab – Transformers From Scratch

- https://colab.research.google.com/drive/1a0MMJBhC_6B0V4tz0opxt-8Oi14A-rsi?usp=sharing

Variasi Transformers: T5 Encoder-Decoder



The differences:

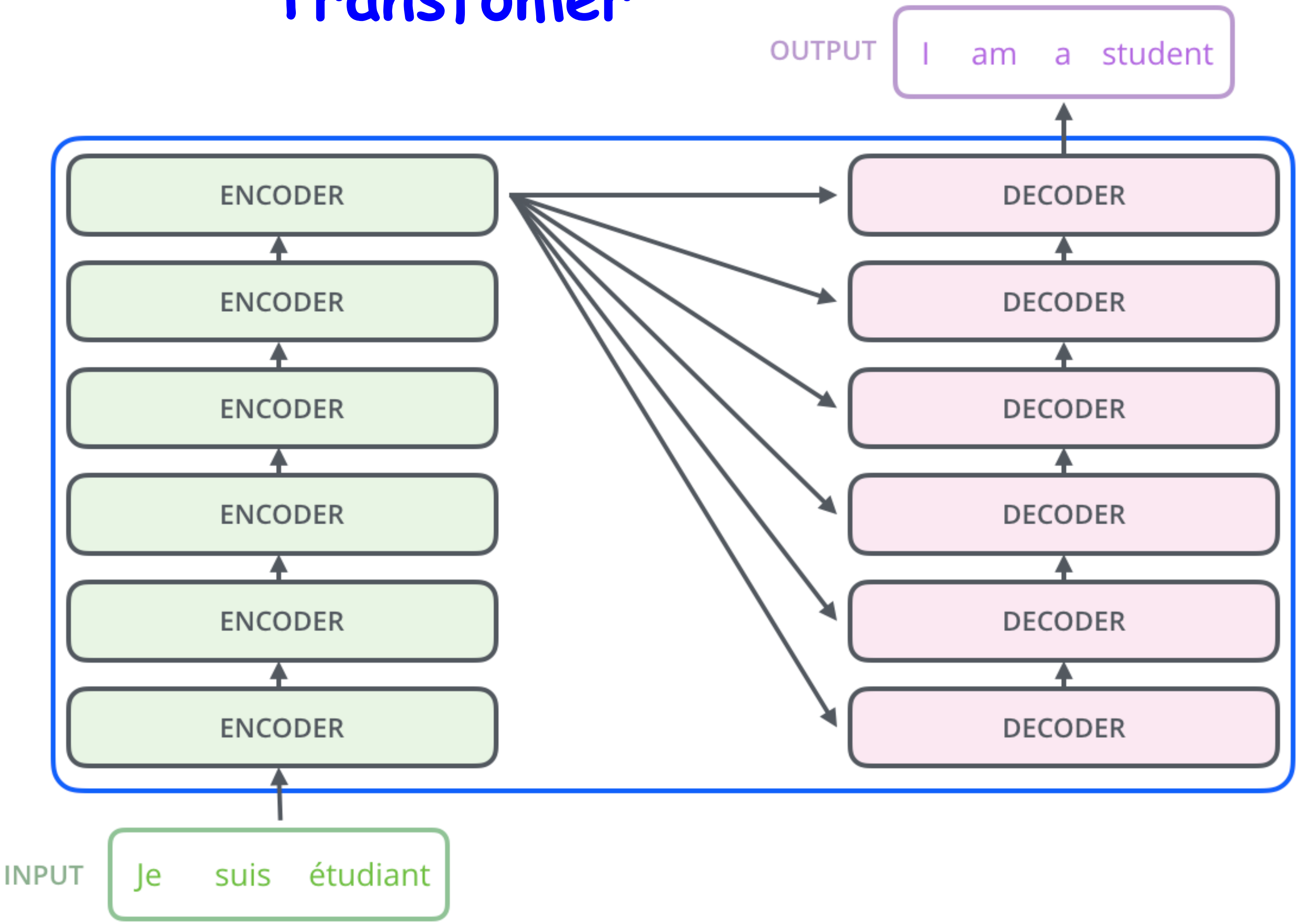
1. **LayerNorm** is applied immediately before each attention and feed forward transformation (i.e., outside of the residual path)
2. **No additive bias** is used for LayerNorm (i.e., see here; we only use scale and eliminate the additive bias)
3. **Dropout is applied throughout the network** (e.g., attention weights, feed forward network, skip connection, etc.)

https://cameronrwolfe.substack.com/p/t5-text-to-text-transformers-part

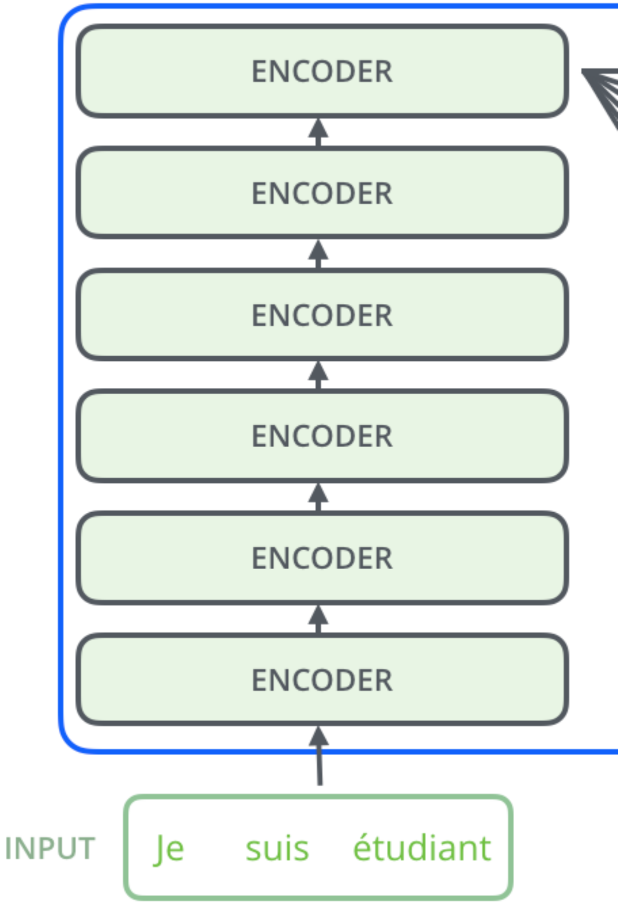
Encoder-Only Model

What can we do if we only have an Encoder?

Transfomer


OUTPUT
I am a student
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
DECODER
DECODER
DECODER
DECODER
DECODER
DECODER
INPUT
Je suis étudiant

Encoder-Only



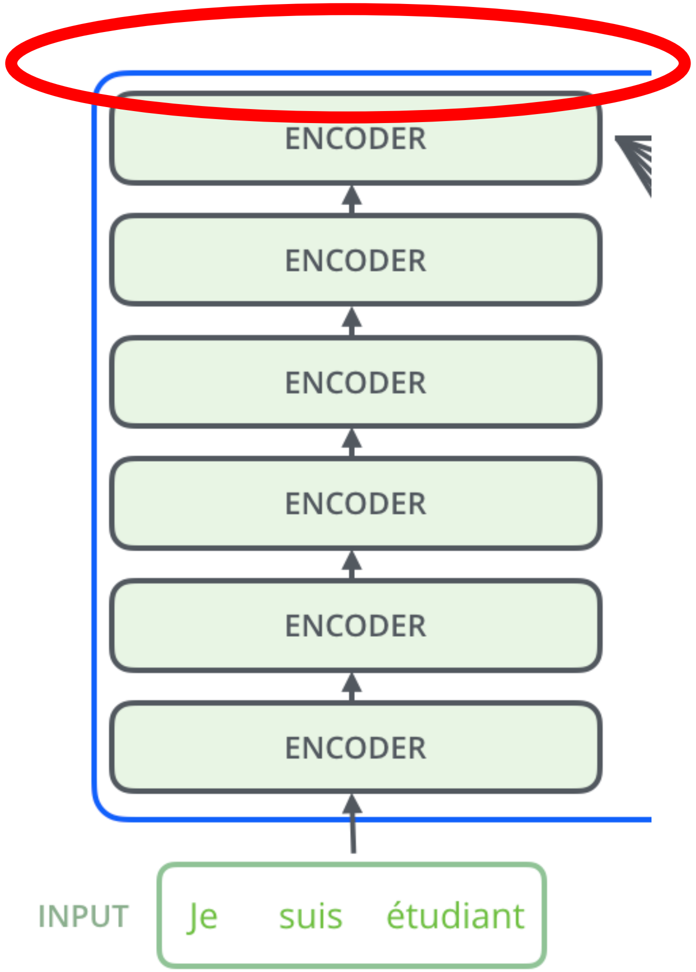
Model Encoder-Only seperti ini "booming" mulai tahun 2019 dari publikasi Devlin et al. (2019). Devlin et al., menyebut model ini dengan **BERT** (Bidirectional Encoder Representations from Transformers).

bidirectional itu lebih me-refer ke self attention

Namun, **jangan hanya persempit konsep Encoder pada BERT saja**!

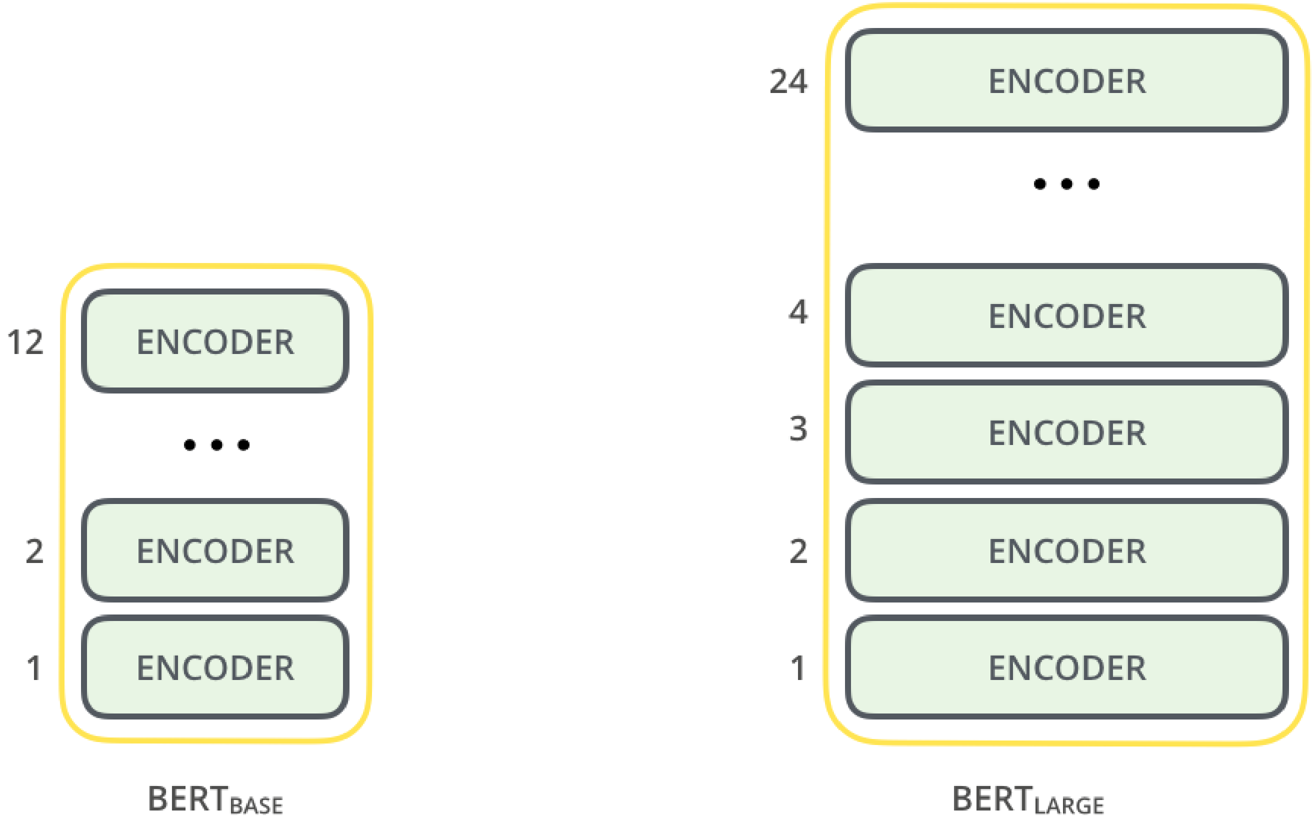
Ada banyak Encoder-only model yang lain!

Apa yang dihasilkan Encoder-Only Model?



Output yang dihasilkan top encoder dapat dianggap sebagai **"representasi high-level" (high-level features)** dari sebuah kalimat "je suis etudiant"

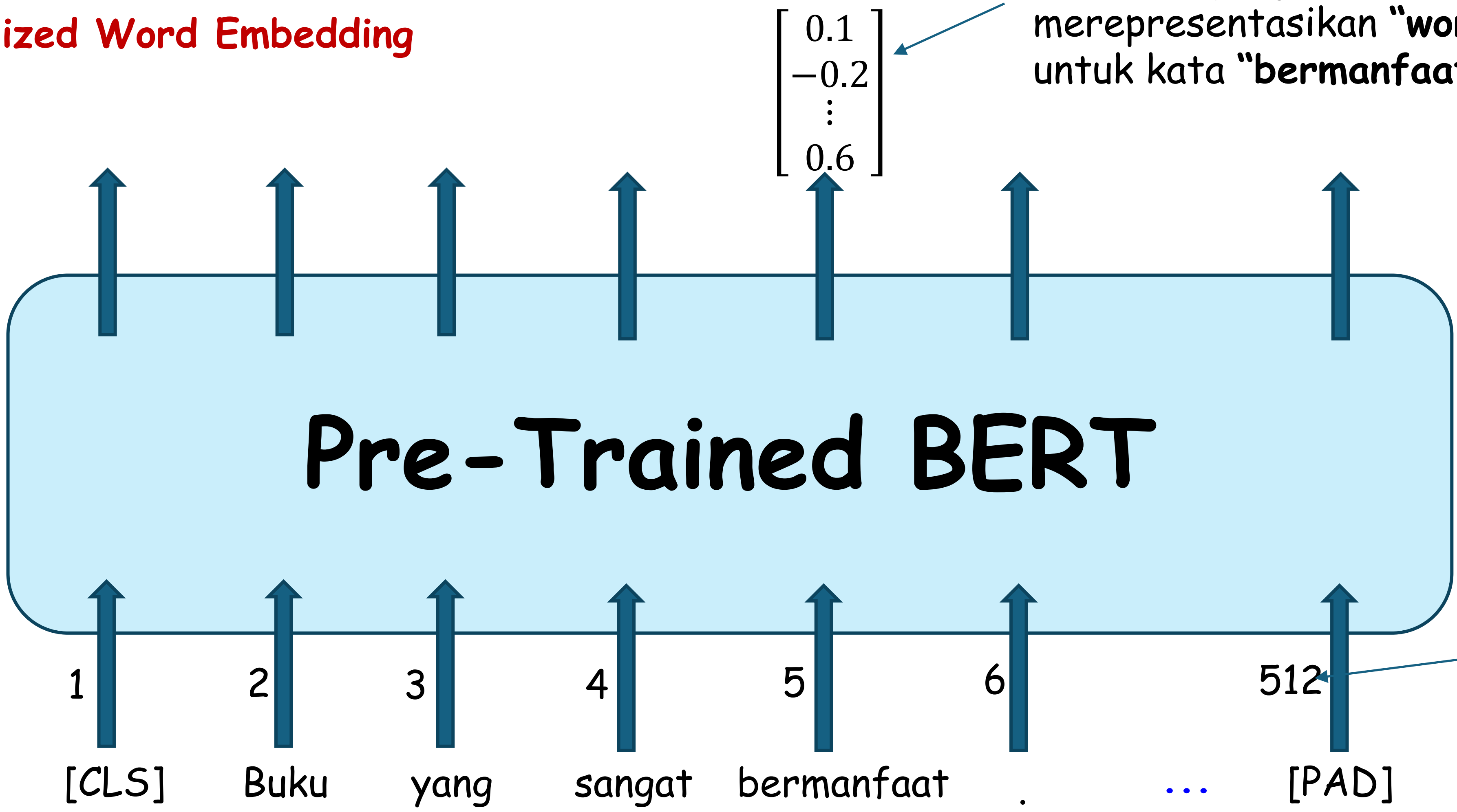
Variasi BERT



BERT: Bidirectional Encoder Representation from Transformer

Contextualized Word Embedding

Vektor sepanjang **768 unit**, dan merepresentasikan "**word embedding**" untuk kata "**bermanfaat**".



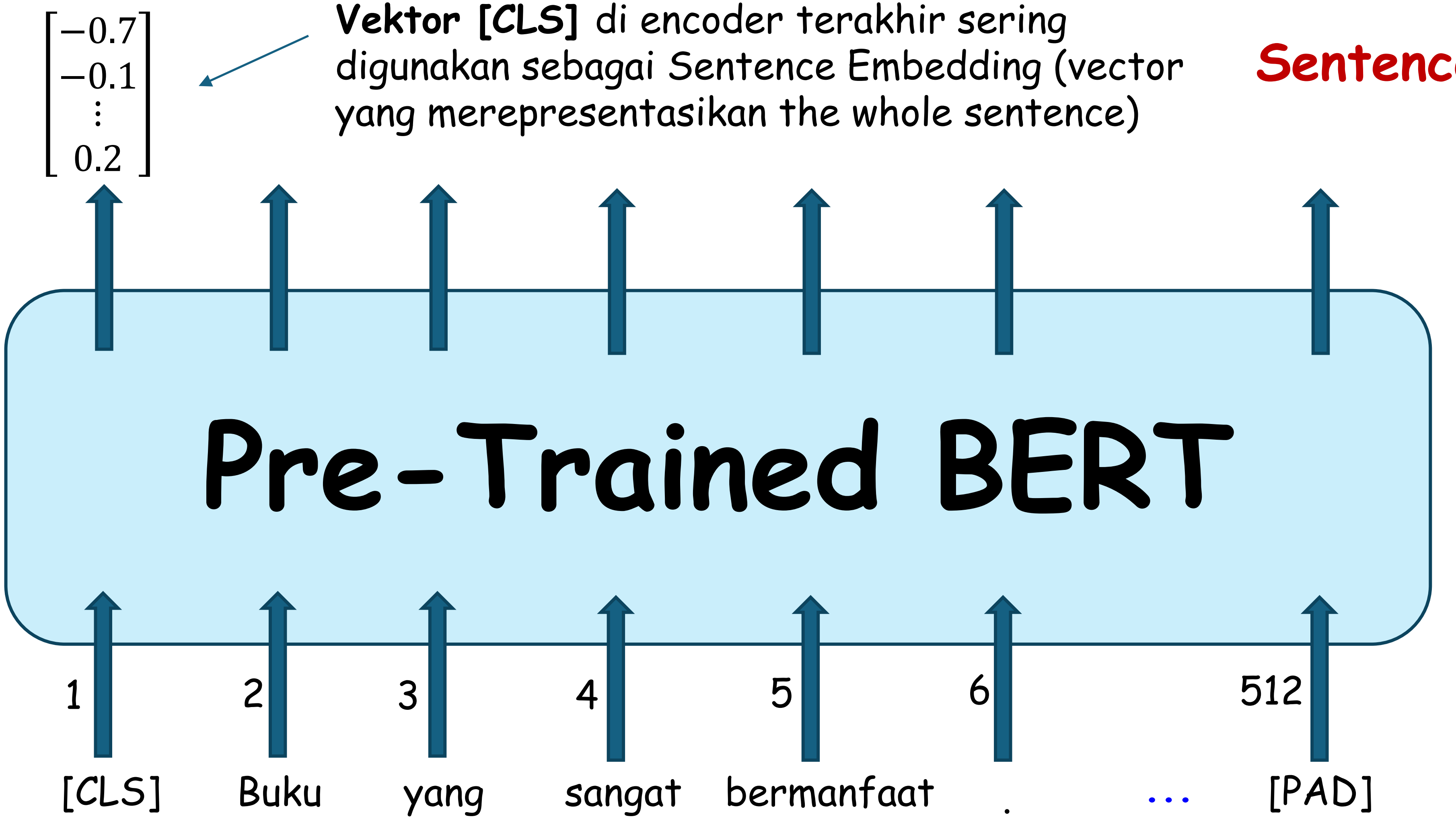
512 adalah maximum sequence length pada BERT atau Transformer

CLS = Classification

Jika menggunakan BERT, token special [CLS] disisipkan sebagai token pertama.

Padding

BERT: Bidirectional Encoder Representation from Transformer



Sentence Embedding

BERT: Bidirectional Encoder Representation from Transformer

Contoh Ketika kita menggunakan BERT untuk Text Classification.

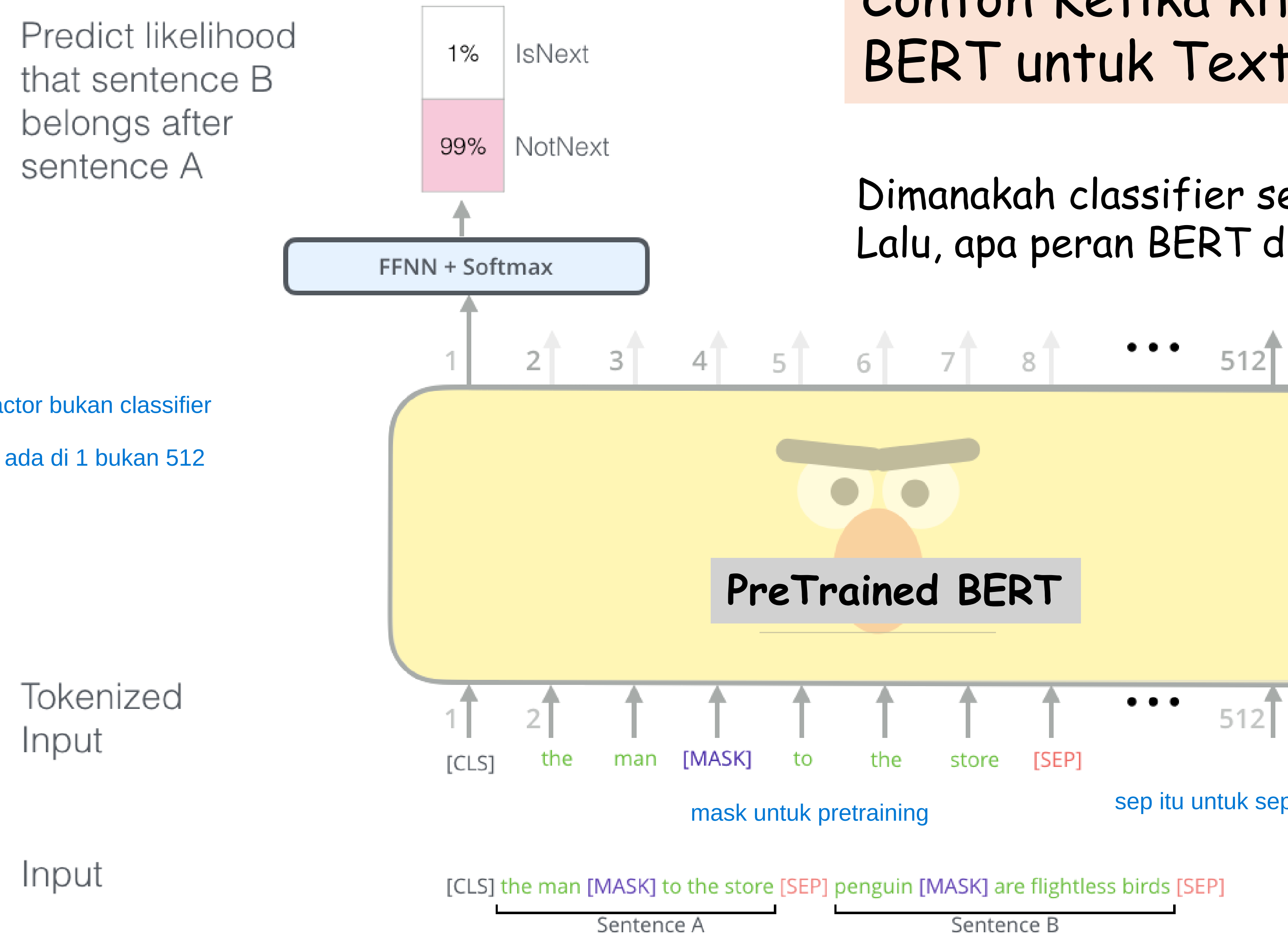
Dimanakah classifier sesungguhnya berada?
Lalu, apa peran BERT disini?

down stream task: text classificaiton

encoder itu

feature extractor bukan classifier

makanya itu ada di 1 bukan 512

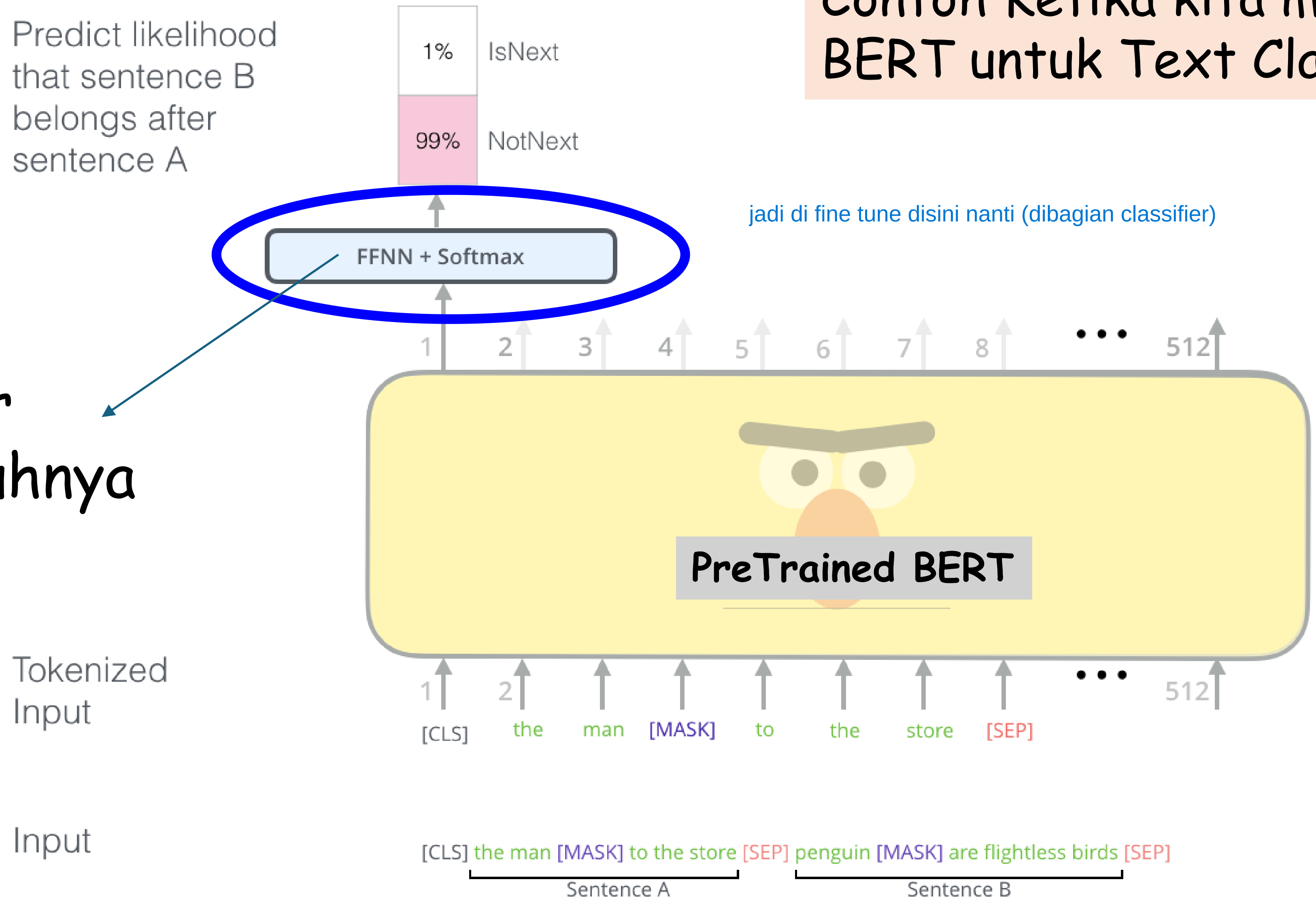


sep itu untuk seperator (antar kalimat ada token spesial SEP)

mask untuk pretraining

BERT: Bidirectional Encoder Representation from Transformer

Contoh Ketika kita menggunakan BERT untuk Text Classification.



jadi di fine tune disini nanti (dibagian classifier)

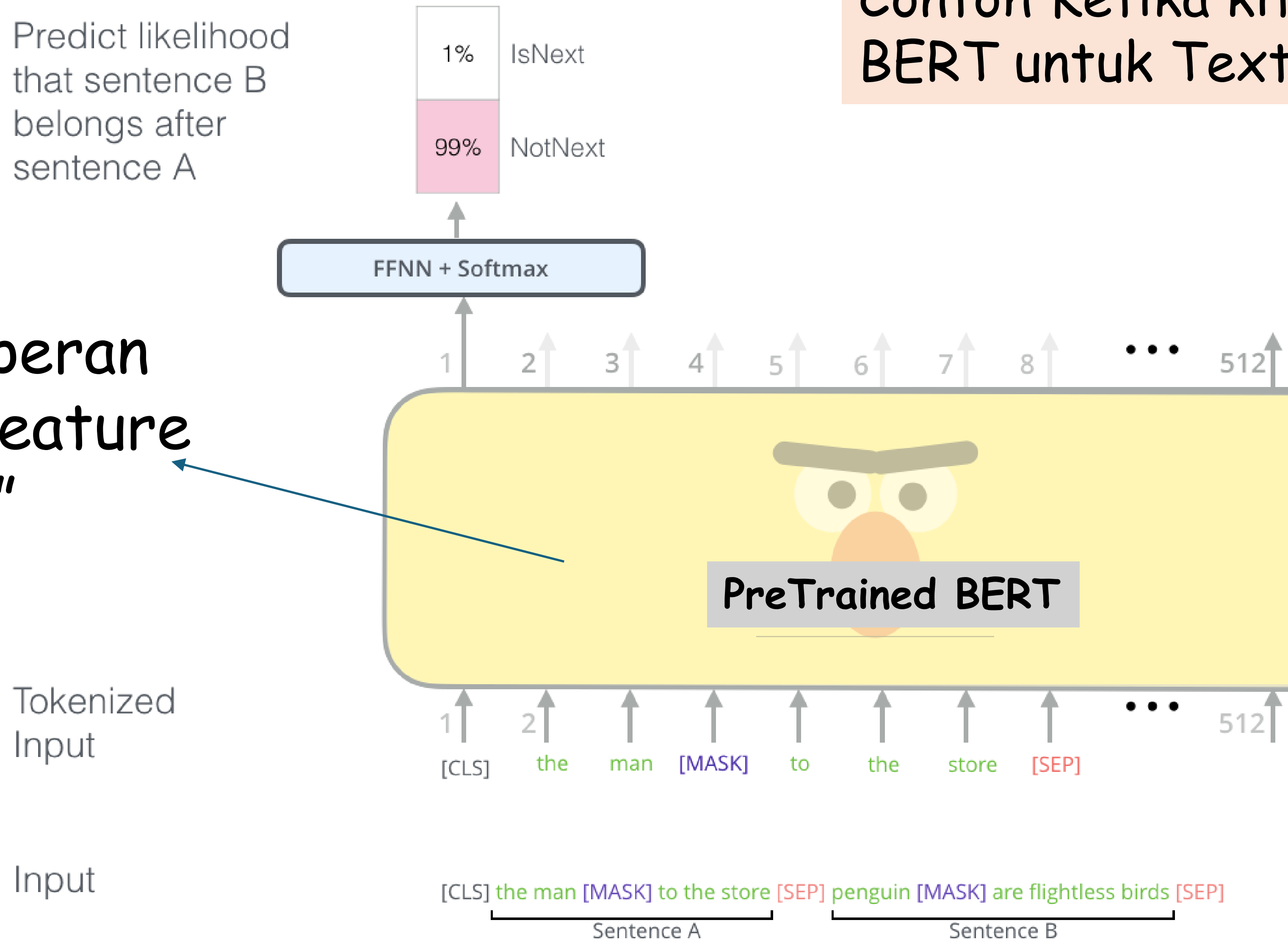
Classifier Sesungguhnya

bisa juga, di freeze weight-weight di BERT nya.

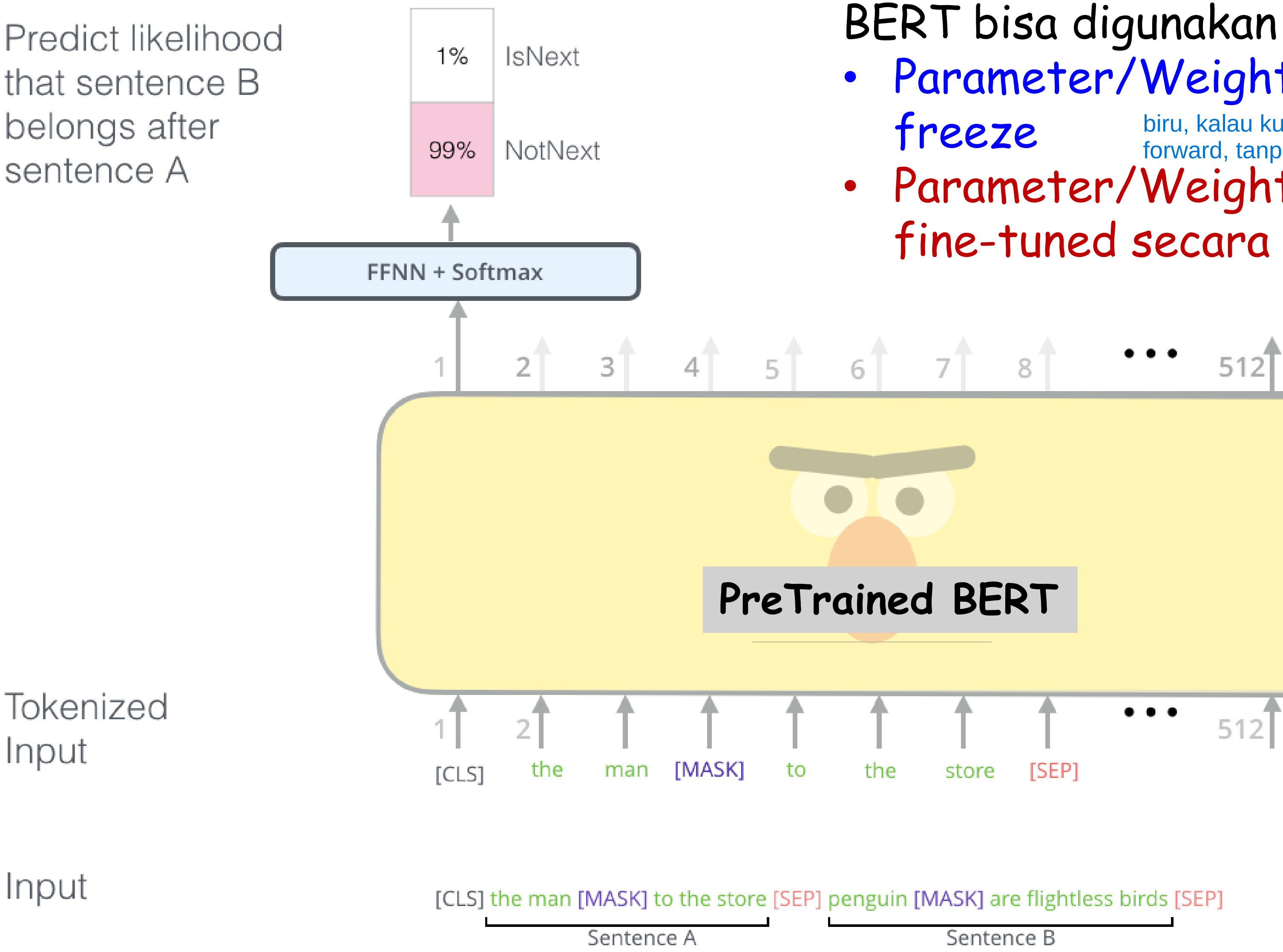
BERT: Bidirectional Encoder Representation from Transformer

Contoh Ketika kita menggunakan BERT untuk Text Classification.

BERT berperan sebagai "feature extractor"



BERT: Bidirectional Encoder Representation from Transformer



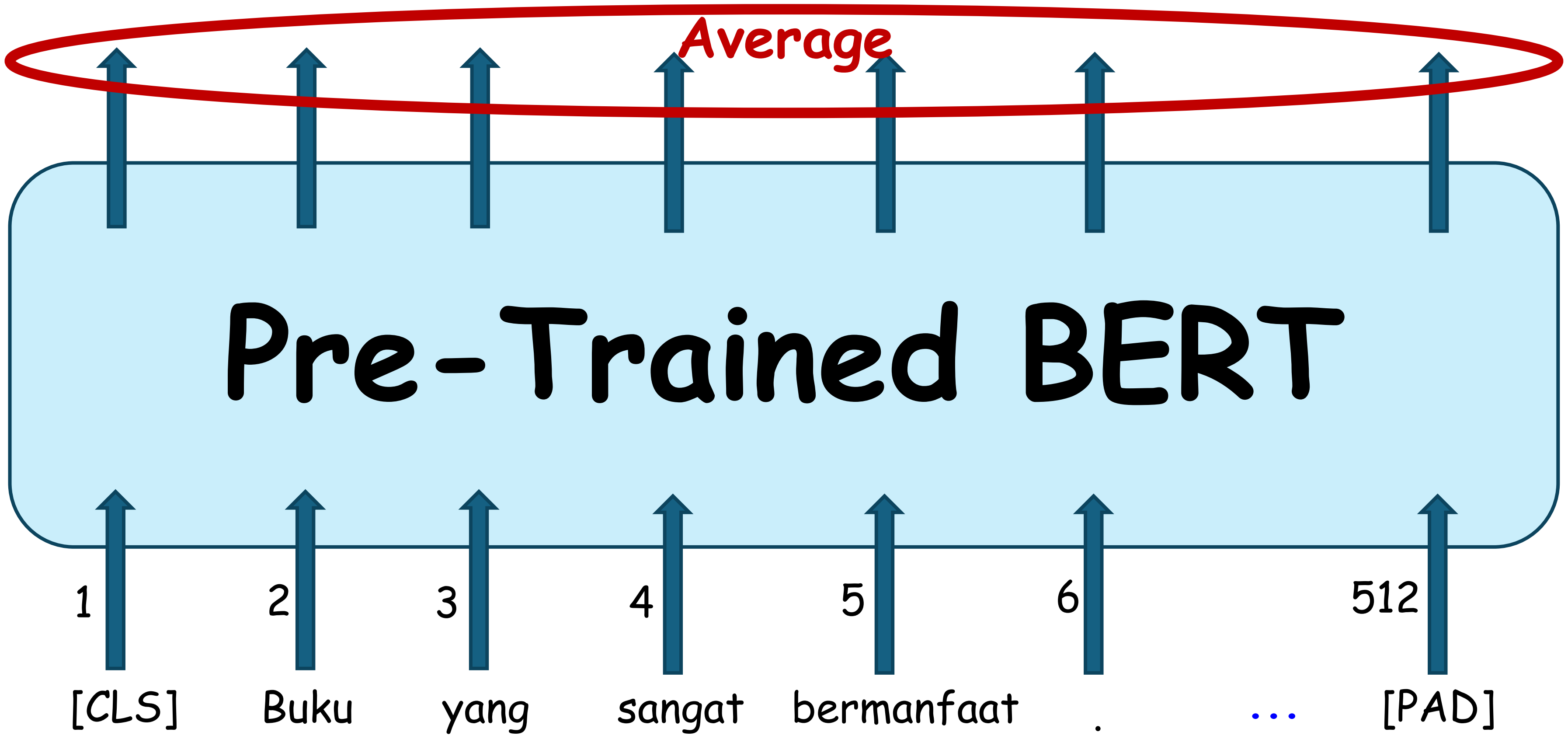
Untuk solve specific task, Pre-Trained BERT bisa digunakan dengan 2 cara:

- Parameter/Weight pada BERT di-freeze
  biru, kalau kurang kurang komputasi resourcenya (cuman di forward, tanpa dihitung)
- Parameter/Weight pada BERT di-fine-tuned secara end-to-end
  yang ini banyak makan resources

BERT: Bidirectional Encoder Representation from Transformer

Ada juga pendekatan yang menghitung centroid dari semua vector pada setiap timestep.

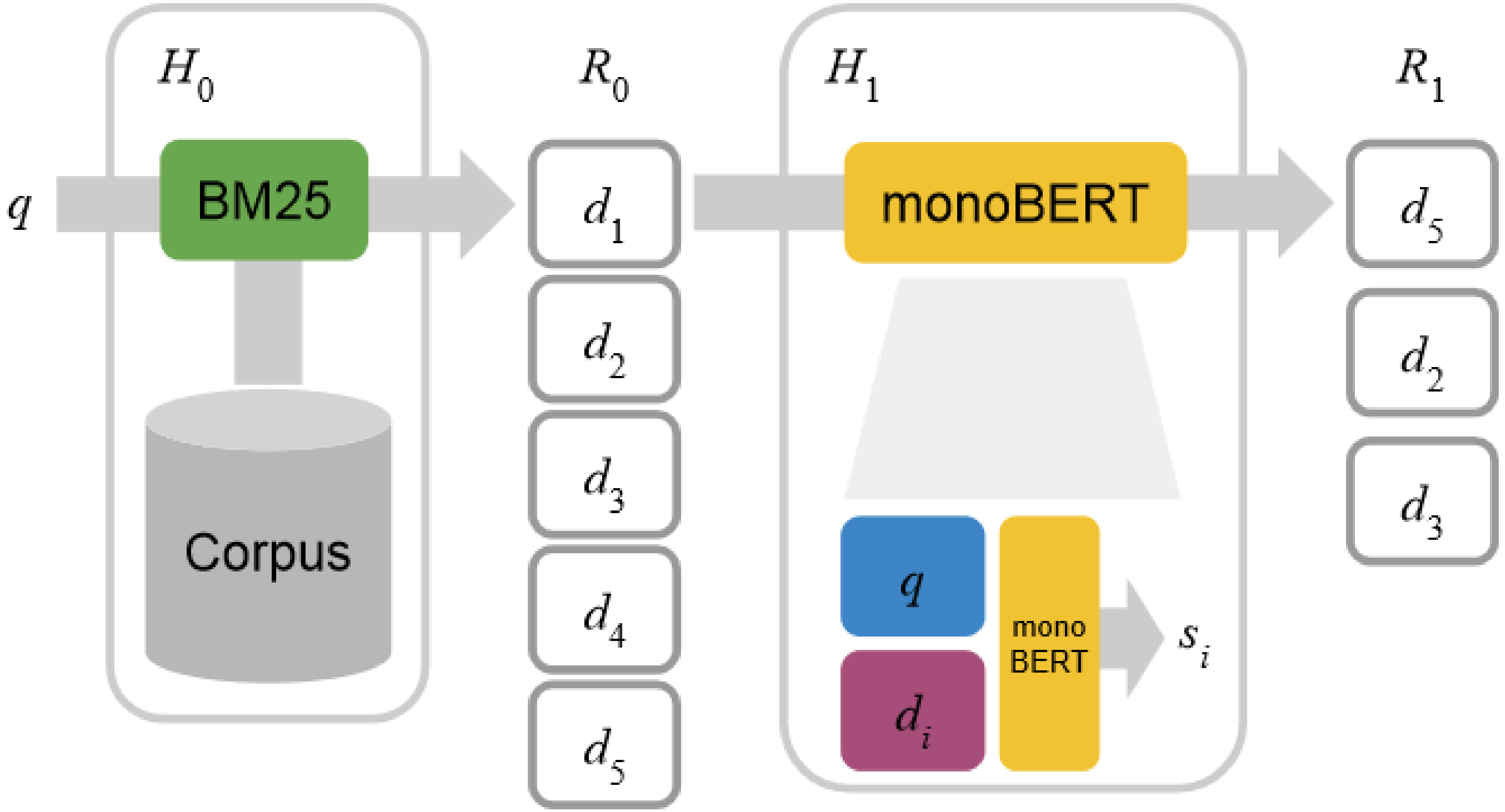
Sentence Embedding



Encoder untuk Score(Q,D) di IR?

monoBERT untuk Re-Ranking (Nogueira et al., 2019)

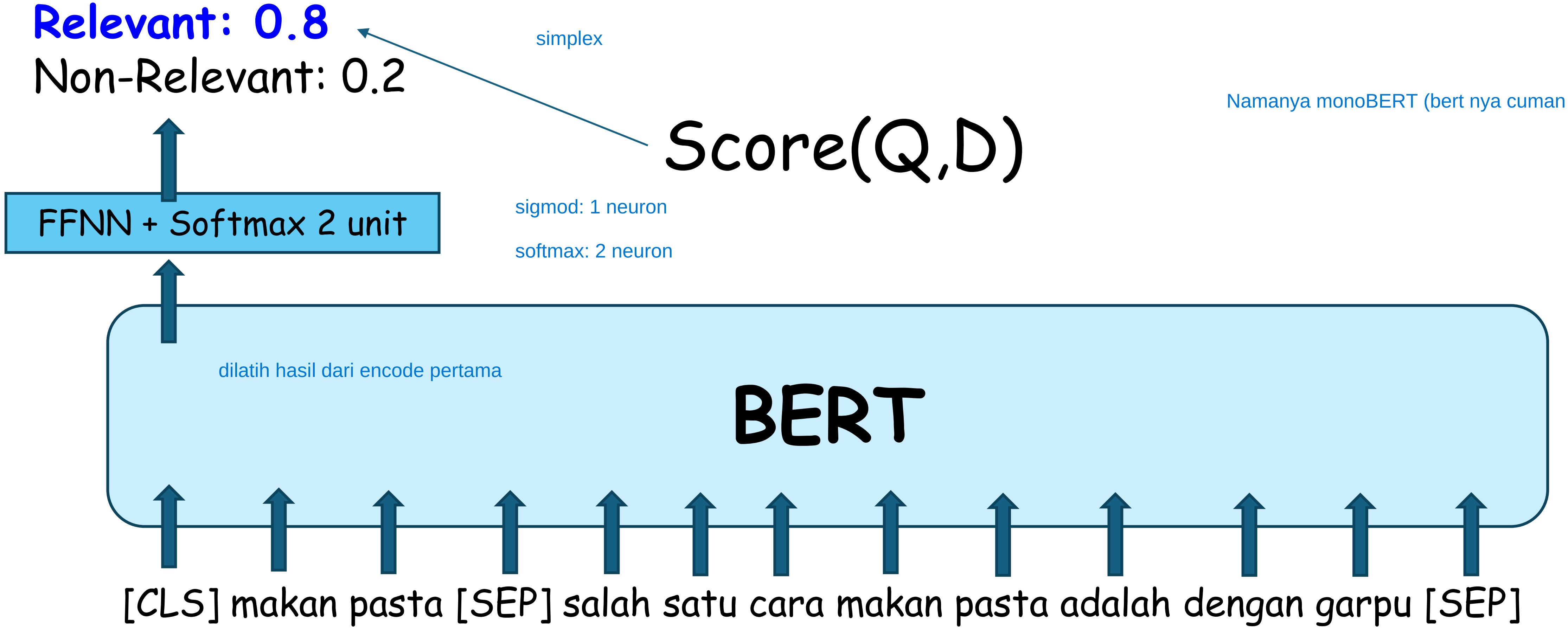
bisa digunakan untuk reranking hasil dari BM 25



monoBERT untuk Re-Ranking (Nogueira et al., 2019)

Q: makan pasta

D: salah satu cara makan pasta adalah dengan garpu



simplex

Namanya monoBERT (bert nya cuman 1)

sigmod: 1 neuron

softmax: 2 neuron

dilatih hasil dari encode pertama

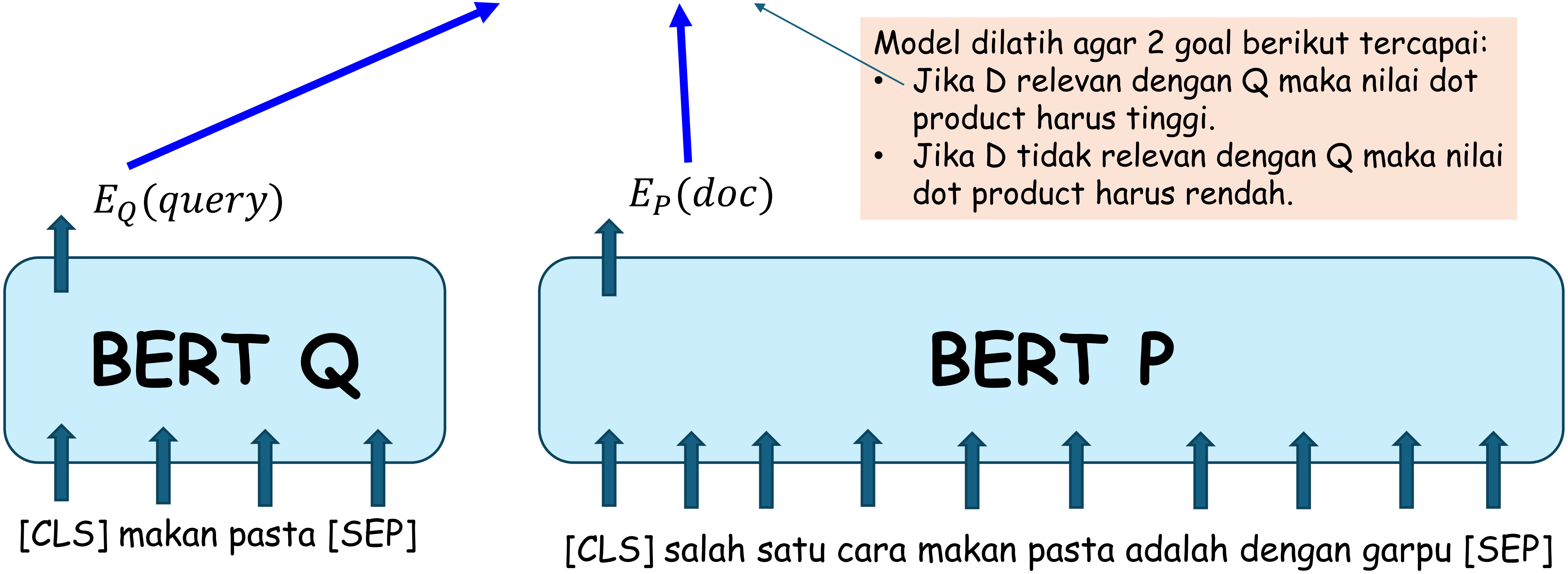
query dan dokumen di kombine dalam 1 sequence yang dipisah dengan seperator [SEP]

Dense Passage Retriever (DPR) (Karpukhin, et al. 2020)

$$Score(Q, D) = E_Q(Q) \cdot E_P(D)$$

Ada 2 BERT terpisah dan bobotnya juga terpisah (tidak shared).

Model dilatih agar 2 goal berikut tercapai:
- Jika D relevan dengan Q maka nilai dot product harus tinggi.
- Jika D tidak relevan dengan Q maka nilai dot product harus rendah.



How to **pre-train** an Encoder?

Why “**pre-train**”? Not just “**train**” or “**fine-tune**”?

Melatih BERT secara Unsupervised?

BERT dilatih secara unsupervised dengan meminimalkan loss function pada dua unsupervised tasks berikut:

- Masked Language Model
- Next Sentence Prediction

ada 2 task untuk pretrain sebuah encoder:
- mask language model (ada loss func)
- next sentence prediction (ada loss func)

nanti 2 lossnya itu digabung, nah hasil penggabungan kedua loss ini adalah BERT nya

Bagaimana melatih BERT secara unsupervised?

Masked Language Model

Masking 15% of the input

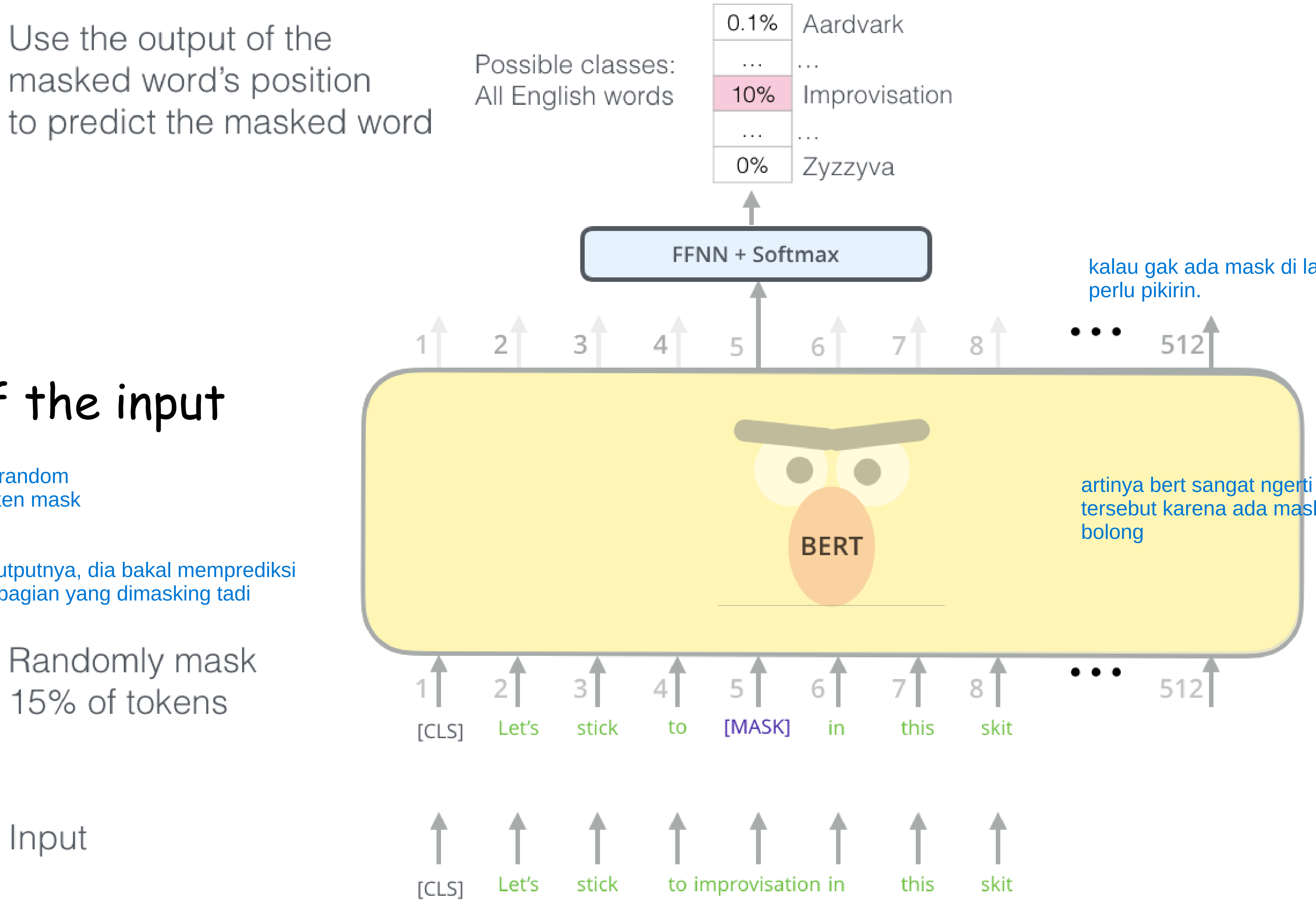
15% token secara random
di ganti dengan token mask

lalu pada bagian outputnya, dia bakal memprediksi
posisi-posisi pada bagian yang dimasking tadi

jadi prediksi kata yang di mask intinya.

kalau gak ada mask di labelnya, kita gak perlu pikirin.

artinya bert sangat ngerti dengan kalimat tersebut karena ada mask yang bolong - bolong



```python
input_ids = torch.tensor([[4,3,4,5,3,4,5,9,7,9,9,9,9,9,9,9,9,9,9,0],
                          [3,3,4,5,3,4,5,9,7,9,9,9,9,9,9,9,9,8,0,0],
                          [2,3,4,5,3,4,5,9,7,9,9,9,9,9,9,9,9,9,9,0]])

# mask 15% of text leaving [PAD] = 0
mlm_mask = torch.rand(input_ids.size()) < 0.15 * (input_ids != 0)

print(mlm_mask)
```

kode untuk mask 15% text

```
tensor([[False, False, False, False, False, False, False,  True, False, False,
         False, False,  True, False, False, False,  True,  True, False, False],

        [False,  True, False,  True, False, False, False, False, False,  True,
         False, False, False,  True, False,  True, False, False, False, False],

        [False, False, False,  True, False, False, False, False, False, False,
         False, False, False, False, False, False,  True, False, False, False]])
```

```python
# we mask the token,  token id 1 is [MASK]

masked_tokens = input_ids * mlm_mask

input_ids[masked_tokens != 0] = 1

print(input_ids)
```

```
tensor([[4, 3, 4, 5, 3, 4, 5, 1, 7, 9, 9, 9, 1, 9, 9, 9, 1, 1, 9, 0],
        [3, 1, 4, 1, 3, 4, 5, 9, 7, 1, 9, 9, 9, 1, 9, 1, 9, 8, 0, 0],
        [2, 3, 4, 1, 3, 4, 5, 9, 7, 9, 9, 9, 9, 9, 9, 9, 1, 9, 9, 0]])
```

```python
def tokenize(text):
    return text.split(' ')

class MLMDataset(torch.utils.data.Dataset):
    def __init__(self, sequence_length, documents, mask_portion=0.15):
        self.sequence_length = sequence_length
        self.words = self.load_words(documents)
        self.uniq_words = self.get_uniq_words()

        # id vocab mulai dari 2, bukan 0; 0 untuk [PAD] & 1 untuk [MASK]
        self.index_to_word = {(index + 2): word for index, word in enumerate(self.uniq_words)}
        self.word_to_index = {word: (index + 2) for index, word in enumerate(self.uniq_words)}
        self.index_to_word[0] = "[PAD]"
        self.word_to_index["[PAD]"] = 0
        self.index_to_word[1] = "[MASK]"
        self.word_to_index["[MASK]"] = 1

        self.docs = []
        for doc in documents:
            self.docs.append(self.to_ids(doc))
```

```
def to_ids(self, doc):
    doc = [self.word_to_index[w] for w in tokenize(doc)]
    doc = doc[:self.sequence_length]
    doc += [self.word_to_index["[PAD]"]] * (self.sequence_length - len(doc))
    return doc

def load_words(self, documents):
    text = ""
    for doc in documents:
      text += doc + " "
    return tokenize(text)

def get_uniq_words(self):
    word_counts = Counter(self.words)
    return sorted(word_counts, key=word_counts.get, reverse=True)

def __len__(self):
    return len(self.docs)
```

```python
def __getitem__(self, index):
        input_ids = torch.tensor(self.docs[index])
        labels = input_ids.clone()
        encoder_mask = (input_ids !=
                                torch.tensor(self.word_to_index['[PAD]'])).unsqueeze(0).int()

        mlm_mask = torch.rand(input_ids.size()) < 0.15 *
                                             (input_ids != self.word_to_index["[PAD]"])

        masked_tokens = input_ids * mlm_mask

        # set all tokens except masked tokens to -100
        # CategoricalCrossEntorpyLoss won't consider loss
        # from timesteps associated with output label -100
        labels[masked_tokens == 0] = -100

        input_ids[masked_tokens != 0] = self.word_to_index["[MASK]"]

        return input_ids, labels, encoder_mask
```

```python
class MLMLoss(nn.Module):
    def __init__(self, encoder, vocab_size, hidden_dim,
                 pad_token_id=0, mask_token_id=1):
        super().__init__()

        #any encoder that you want, BERT, RNNs that output [Batch, SeqLen, HiddenDim]
        self.encoder = encoder

        #linear layer for the head
        self.mlm_head = nn.Linear(hidden_dim, vocab_size)

        self.pad_token_id = pad_token_id
        self.mask_token_id = mask_token_id
```

```
...

    def forward(self, input_ids, labels=None, mask=None):
        # forward: input_ids -> encoder -> mlm_head

        # input_ids: [Batch, SeqLen], enc_out: [Batch, SeqLen, HiddenDim]
        enc_out = self.encoder(input_ids, mask=mask)

        logits = self.mlm_head(enc_out)  # logits: [Batch, SeqLen, NumVocab]

        if labels is not None:
            loss = F.cross_entropy(logits.view(-1, logits.size(-1)),
                                   labels.view(-1),
                                   ignore_index=-100)
            return loss, logits
        else:
            return logits
```

```python
def train(dataset, mlm_model, batch_size=2, lr=0.001, max_epochs=5000):
    mlm_model.train()

    dataloader = torch.utils.data.DataLoader(dataset, batch_size=batch_size)
    optimizer = optim.Adam(mlm_model.parameters(), lr=lr)

    for epoch in range(max_epochs):

        #activate this one if you want to get the same masking results in each epoch
        #torch.manual_seed(42)

        losses = []
        for batch, (input_ids, labels, mask) in enumerate(dataloader):
            loss, logits = mlm_model(input_ids, labels=labels, mask=mask)
            loss.backward()

            optimizer.step()
            optimizer.zero_grad()
            losses.append(loss.detach().item())

        if (epoch+1) % 100 == 0:
            print({ 'epoch': epoch, 'loss': sum(losses)/len(losses) })
```

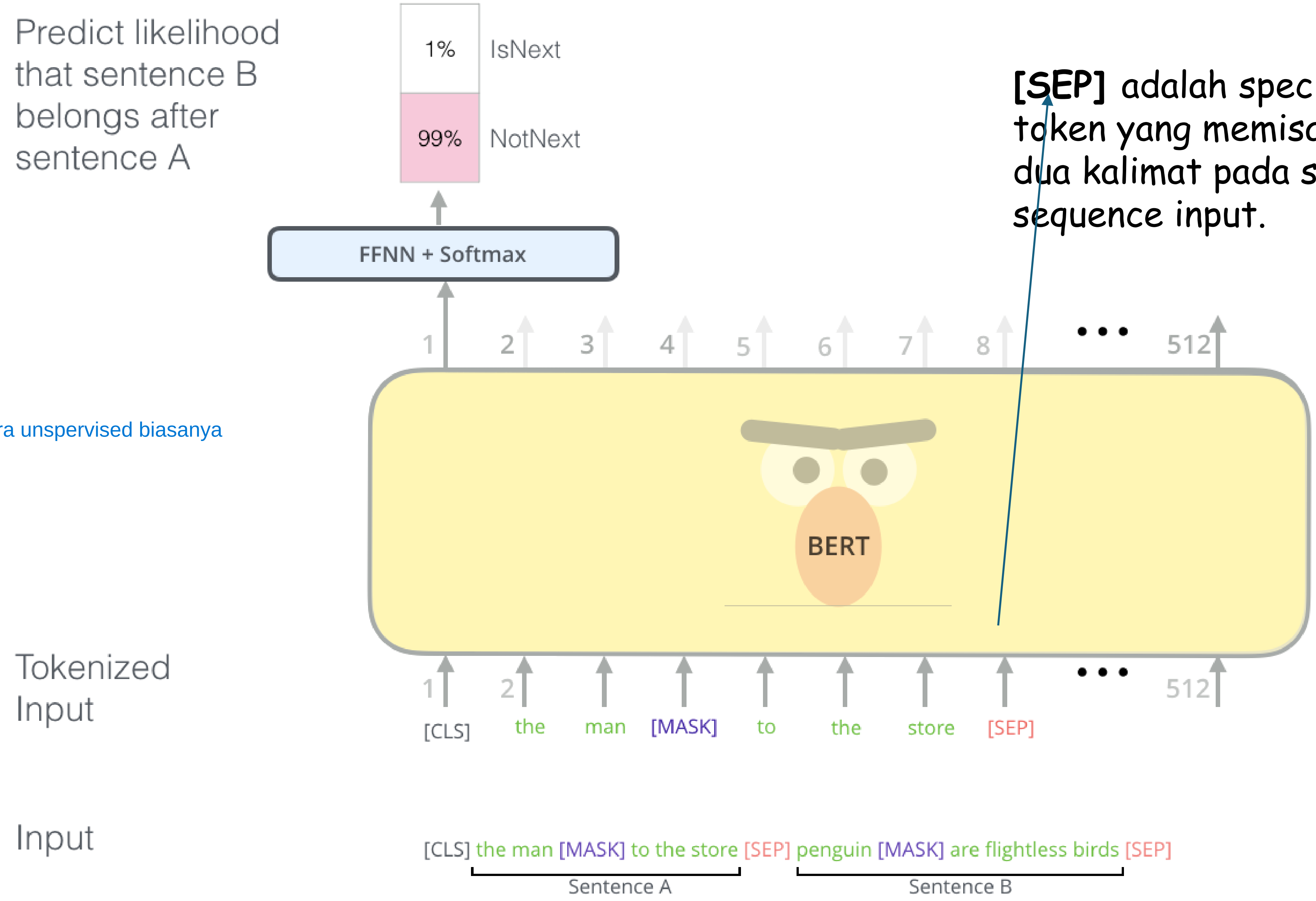
```python
# instantiate a model and start training!

SEQ_LEN = 15
MODEL_DIM = 32
NUM_HEAD = 2
NUM_LAYERS = 1

dataset = MLMDataset(SEQ_LEN, documents, mask_portion=0.15)

vocab_size = len(dataset.index_to_word)

encoder = TransformerEncoder(vocab_size, MODEL_DIM, \
                                            NUM_HEAD, NUM_LAYERS)

mlm_head = MLMLoss(encoder, vocab_size, MODEL_DIM)

train(dataset, mlm_head, lr=0.001)
```

```python
# test our encoder; apakah bisa dengan baik menghasilan representasi vektor untuk
# sebuah dokumen ?

@torch.no_grad()
def doc_vector(text, word_to_index, encoder):
    input_ids = torch.tensor([[word_to_index[word] for word in tokenize(text)]])
    encoder_mask = (input_ids != torch.tensor(word_to_index['[PAD]'])).int()
    return torch.mean(encoder(input_ids, mask=encoder_mask), dim=-2)

sent_1 = doc_vector("ilmu komputer dan machine learning", dataset.word_to_index, encoder)
sent_2 = doc_vector("bisa ular bisa berbahaya", dataset.word_to_index, encoder)
sent_3 = doc_vector("sistem operasi yang sangat efisien", dataset.word_to_index, encoder)

print(F.cosine_similarity(sent_1, sent_2))   # tensor([0.3396])
print(F.cosine_similarity(sent_1, sent_3))   # tensor([0.7227])
```

Bagaimana melatih BERT secara unsupervised?

Next Sentence Prediction

dia dilatih secara unspervised biasanya

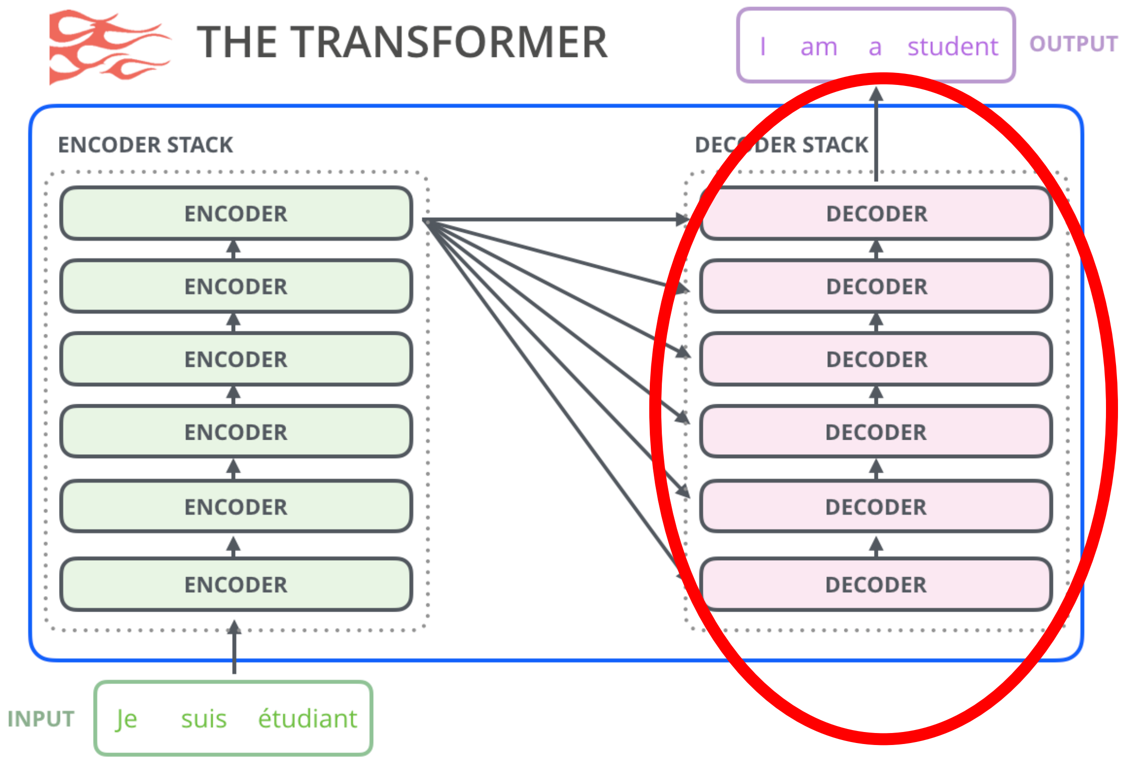


[SEP] adalah special token yang memisahkan dua kalimat pada sebuah sequence input.

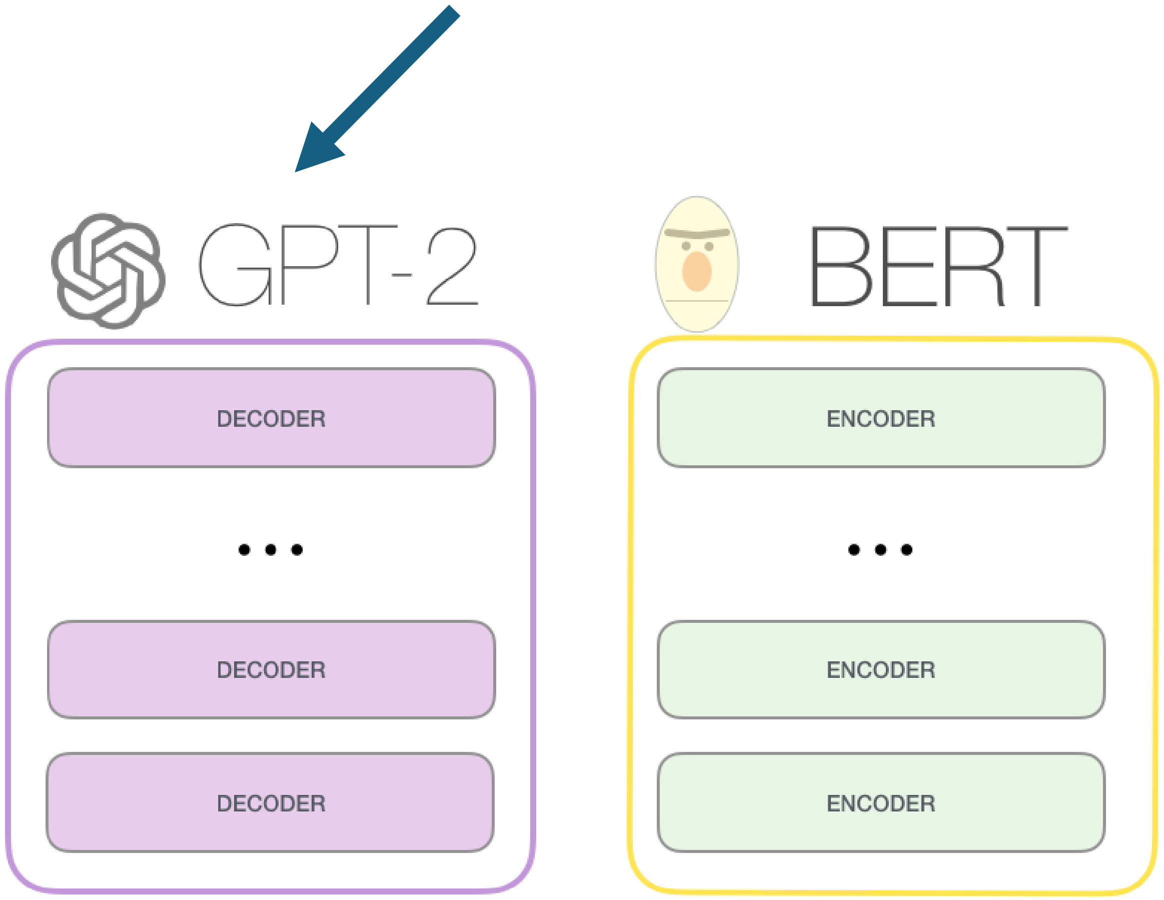
Link Google Collab – BERT From Scratch

- https://colab.research.google.com/drive/1QY8nI8q496-T4mEE686jFXUnX_LD9wbp?usp=sharing

Decoder-Only Models & Causal Language Modelling



Generative Pre-Trained Transformers (GPT)



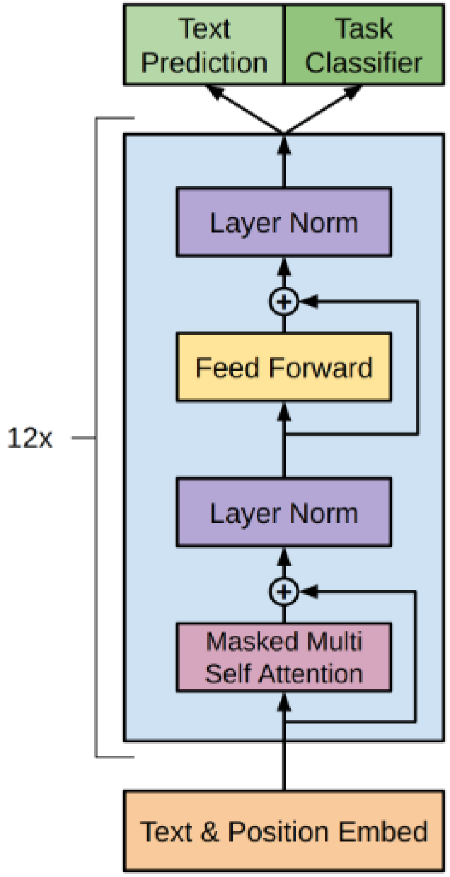
bert = encoder (mask language modeling), butuh informasi dari kiri (sebelumnya) dan kanan (setelahnya), makanya namanya bidirectional

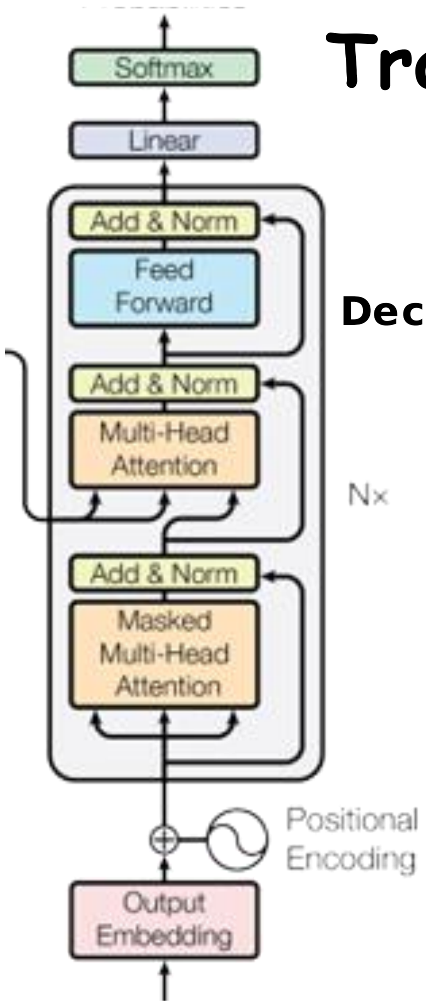
gpt = decoder only (causal language modeling)

http://jalammar.github.io/illustrated-gpt2/

GPT vs Transformer's Decoder ??

GPT





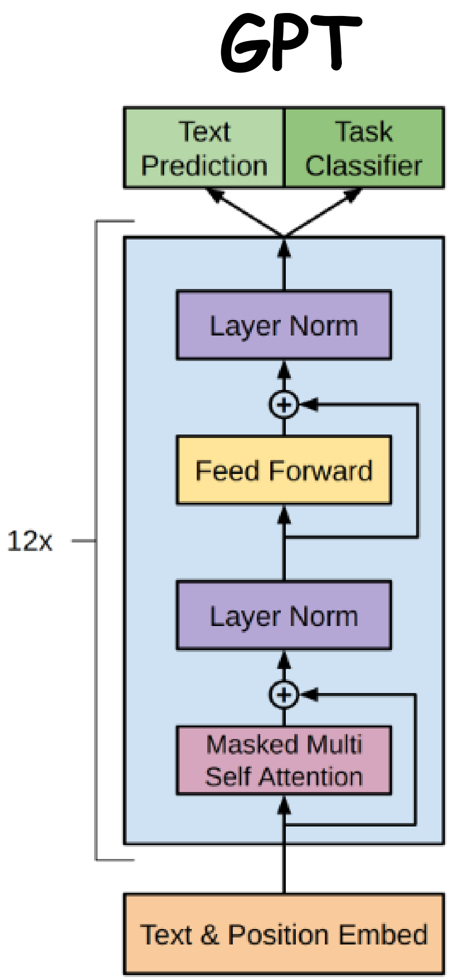
Transformer's Decoder

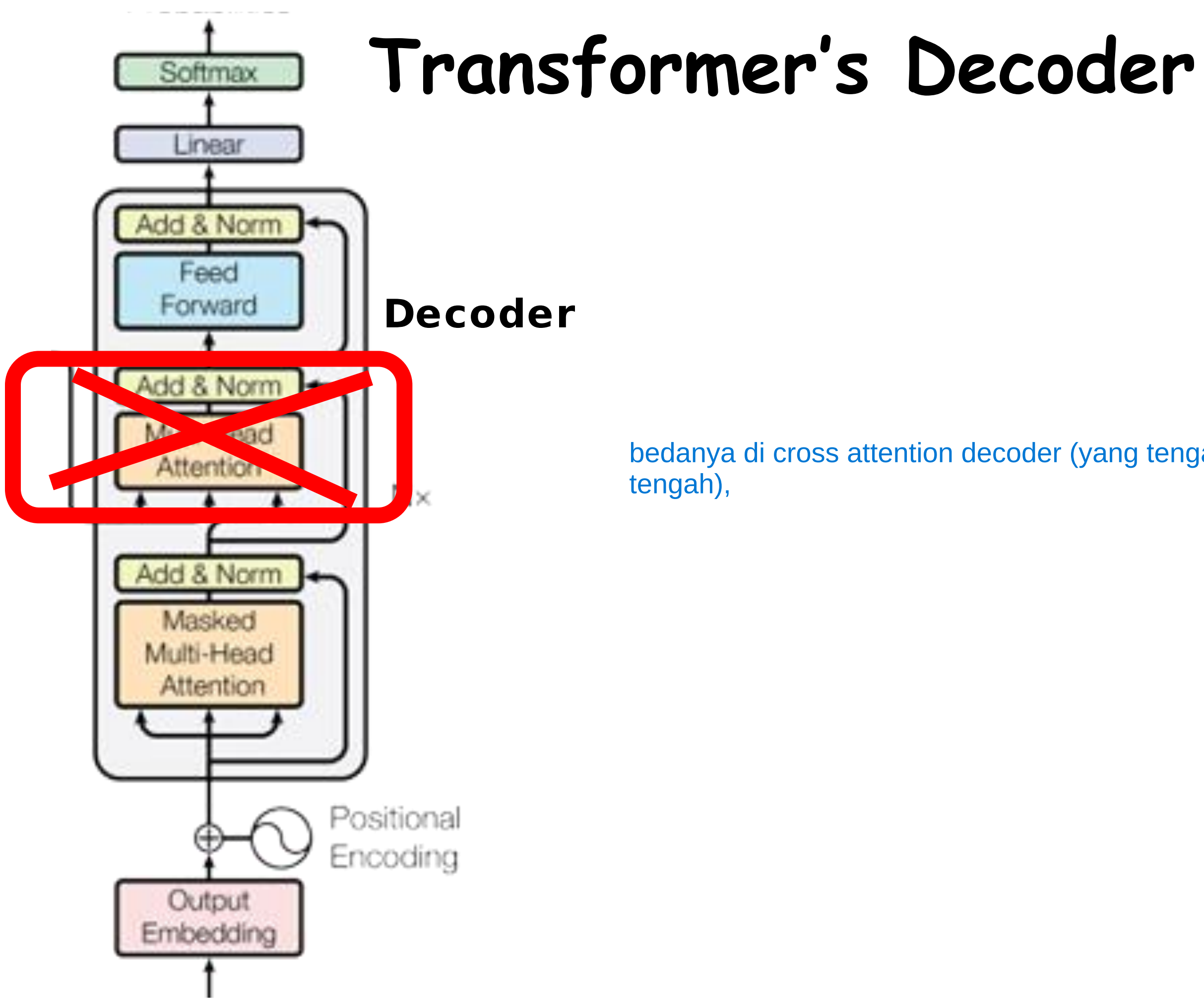
Decoder

bedanya di cross attention (yang tengah-tengah)

Radford et al., Improving Language Understanding by Generative Pre-Training

GPT vs Transformer's Decoder ??





bedanya di cross attention decoder (yang tengah-tengah),

Radford et al., Improving Language Understanding by Generative Pre-Training

GPT

The conditional probability $P(w_t|w_{t-1}, w_{t-2}, \dots, w_{t-k}; \boldsymbol{\theta})$ is modeled using Neural Networks with parameter $\boldsymbol{\theta}$.

Context vector of tokens (**k x |V|**)

Token embedding matrix (**|V| x emb_size**)

$$h_0 = UW_e + W_p$$

Position embedding matrix (**k x emb_size**)

hidden_vectors (**k x emb_size**)

$$h_l = decoder(h_{l-1}) \quad \forall i \in [1, n]$$

$$P(w_t|w_{t-1:t-k}) = softmax(h_n W_e^T)$$

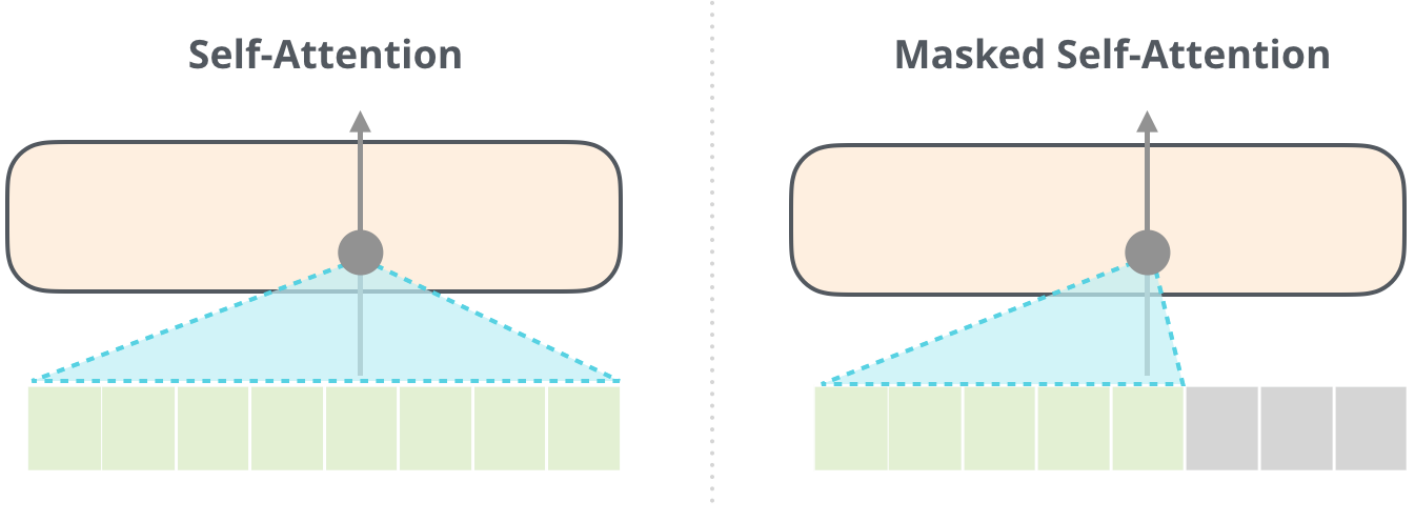
jasil akhirnya di kali dengan token embeeding matriks awal yang transpose

Here, $\boldsymbol{k}$ is the size of the context window.

Radford et al., Improving Language Understanding by Generative Pre-Training

Masked Self-Attention

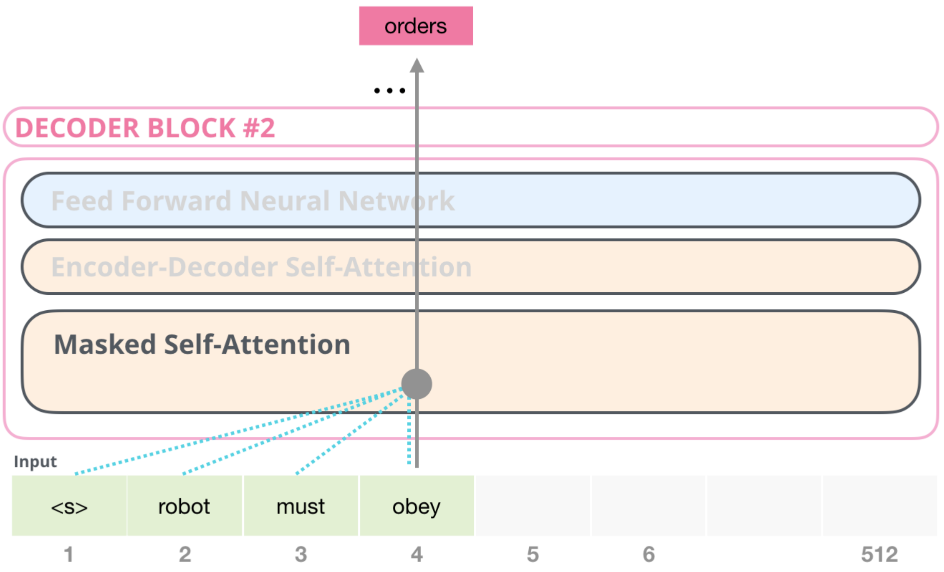
Differ from BERT, GPT uses **masked self-attention.**



BERT

GPT

hanya sebelah kiri saja



http://jalammar.github.io/illustrated-gpt2/

Masked Self-Attention

Differ from BERT, GPT uses **masked self-attention.**

Queries

robot	must	obey	orders

X

Keys

robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders

=

Scores (before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

upper triangle 0, sumnhya 1 per baris

Scores (before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

Apply Attention Mask →

Masked Scores (before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

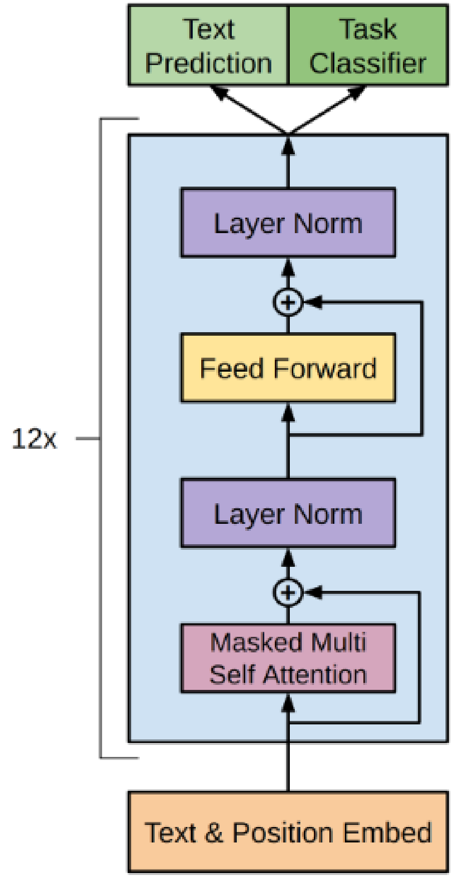
Softmax (along rows) →

Scores

1	0	0	0
0.48	0.52	0	0
0.31	0.35	0.34	0
0.25	0.26	0.23	0.26

```python
class DecoderBlock(nn.Module):

    def __init__(self, input_dim, num_heads, dim_feedforward, dropout=0.0):
        super().__init__()

        # Attention layer
        self.self_attn = MultiheadAttention(input_dim, input_dim, num_heads)

        # Two-layer MLP
        self.linear_net = nn.Sequential(
            nn.Linear(input_dim, dim_feedforward),
            nn.Dropout(dropout),
            nn.ReLU(inplace=True),
            nn.Linear(dim_feedforward, input_dim)
        )

        # Layers to apply in between the main layers
        self.norm1 = LayerNorm(input_dim)
        self.norm2 = LayerNorm(input_dim)
        self.dropout = nn.Dropout(dropout)
```

```python
def forward(self, x, mask=None):

    # Masked Self Attention part
    attn_out = self.self_attn(x, mask=mask)
    x = x + self.dropout(attn_out)
    x = self.norm1(x)

    # MLP part
    linear_out = self.linear_net(x)
    x = x + self.dropout(linear_out)
    x = self.norm2(x)

    return x
```



Unsupervised pre-training for BERT

Given an unsupervised sequence $\boldsymbol{S} = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_T\}$, we use a **masked language modeling** objective to maximize the following likelihood:

$$\sum_{w \in \boldsymbol{M}} \log[P(w|\boldsymbol{M}'; \boldsymbol{\theta})]$$

$\boldsymbol{M}$ is a set of ~**15%** token positions that are randomly selected from $\boldsymbol{S}$ and that will be masked for prediction.

$\boldsymbol{M}' = \boldsymbol{S} - \boldsymbol{M}$, is a set of token positions that are not masked.

Unsupervised pre-training for GPT

Given an unsupervised sequence $\boldsymbol{S} = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_T\}$, we use a **causal language modeling** objective to maximize the following likelihood:

$$\sum_t \log[P(w_t | w_{t-1}, w_{t-2}, \ldots, w_{t-k}; \boldsymbol{\theta})]$$

Here, $\boldsymbol{k}$ is the size of the context window.

Radford et al., Improving Language Understanding by Generative Pre-Training

```python
def tokenize(text):
    return text.split(' ')

class Dataset(torch.utils.data.Dataset):
    def __init__(
        self,
        sequence_length,
        documents, # list of strings
    ):
        self.sequence_length = sequence_length
        self.words = self.load_words(documents)
        self.uniq_words = self.get_uniq_words()

        self.index_to_word = {index: word for index, word in enumerate(self.uniq_words)}
        self.word_to_index = {word: index for index, word in enumerate(self.uniq_words)}

        self.words_indexes = [self.word_to_index[w] for w in self.words]

    def load_words(self, documents):
        text = ""
        for doc in documents:
          text += doc + " "
        return tokenize(text)
```

```python
def get_uniq_words(self):
    word_counts = Counter(self.words)
    return sorted(word_counts, key=word_counts.get, reverse=True)

def __len__(self):
    return len(self.words_indexes) - self.sequence_length

def __getitem__(self, index):
    mask_ = (torch.triu(torch.ones(self.sequence_length, self.sequence_length),
                                                    diagonal = 1).type(torch.int) == 0)

    return {
        'input': torch.tensor(self.words_indexes[index:index+self.sequence_length]),
        'label': torch.tensor(self.words_indexes[index+1:index+self.sequence_length+1]),
        'mask': mask_
    }
```

```
documents = ["sore itu secangkir kopi hadir",
             "bersama rintik hujan",
             "secangkir kopi hitam",
             "yang diseduh perlahan",
             "menjadi sebuah kehangatan",
             "aroma kopi bak harum bunga",
             "yang menggoda selera",
             "jadikan suasana makin ceria",
             "secangkir kopi hitam",
             "sebagai pembangkit semangat",
             "dalam menjalani kehidupan",
             "kubiarkan aroma kopi tersapu angin",
             "menjadi dingin lalu mengendap segala yang diinginkan",
             "hingga nanti berganti musim"]
```

```python
def train(dataset, model, batch_size, max_epochs=400):
    model.train()
    dataloader = torch.utils.data.DataLoader(dataset, batch_size=batch_size)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    for epoch in range(max_epochs):
        losses = []
        for batch in dataloader:
            input = batch['input'].to(device)
            label = batch['label'].to(device)
            mask = batch['mask'].to(device)

            out = model(input, mask=mask)
            loss = criterion(out.transpose(1, 2), label)
            # n_vocab = out.size()[-1]
            # loss = criterion(out.view(-1, n_vocab), label.view(-1))    # same

            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
            losses.append(loss.detach().item())

        if (epoch+1) % 100 == 0:
            print({ 'epoch': epoch+1, 'loss': sum(losses)/len(losses) })
```

```python
# training ...

dataset = Dataset(4, documents)

decoder = GPT(len(dataset.index_to_word), 32, 2, 2)

train(dataset, decoder, 4, max_epochs=4000)
```

After pre-training a GPT model in an unsupervised fashion, then how to use it to generate a sequence ?

https://huggingface.co/blog/how-to-generate

Holtzman et al., The Curious Case of Neural Text Degeneration, ICLR 2020

Open-ended vs Directed generation

jadi teks awal, masuk ke encoder

directed nya ada encoder biasanya

nanti decoder harus ubah (misalnya di translate)

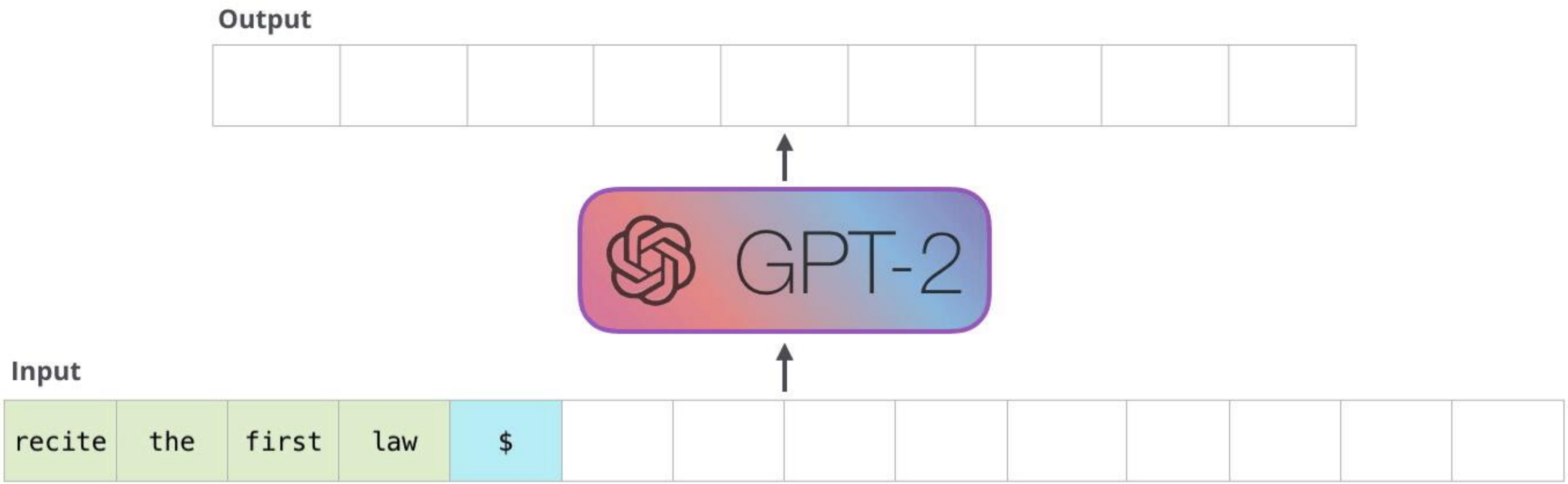
- Directed Generation
  - Example: Machine translation, Text summarization
  - Output is tightly scoped by the input, repetition and genericness are not as problematic.

open ended itu (teksnya unlimited, kita terserah generate apapun, nanti bakal keluar aja kayak biasa)

- Open-ended Generation
  - Conditional story generation and contextual text continuation
  - While the input context restricts the space of acceptable output generations, there is a considerable degree of freedom in what can plausibly come next.

ganti slide selanjutnya.

Open-ended Generation



http://jalammar.github.io/illustrated-gpt2/

```python
@torch.no_grad()
def generate(text, model, word_to_index, index_to_word, max_len=16):
    model.eval()
    decoder_input = [word_to_index[w] for w in tokenize(text)]
    decoder_input = torch.tensor([decoder_input]).int().to(device)

    for i in range(max_len):
        x = decoder_input[:, i:]
        seq_len = x.size(1)
        mask = (torch.triu(torch.ones(1, seq_len, seq_len),
                                               diagonal = 1).type(torch.int) == 0).to(device)

        output = model(x, mask=mask)
        prob = output[:, -1] # last token

        # Selecting token with the highest probability
        _, next_word = torch.max(prob, dim=1)
        decoder_input = torch.cat([decoder_input,
                                   torch.tensor([[next_word.item()]]).int().to(device)], dim=1)

    decoder_input = decoder_input.squeeze()
    words = [index_to_word[id.item()] for id in decoder_input]
    return words
```

Greedy Search! Choose t
best at timestep t

Link Google Collab – GPT From Scratch

- https://colab.research.google.com/drive/1kVfkWBseA39UbH-rZCIC6L4bIZK4OVrw?usp=sharing

Ok, now, let's discuss decoding strategies in more detail

Language Model Decoding

Given **n**-token **context** $\boldsymbol{p_1, p_2, \ldots, p_n}$, find T subsequent tokens $\boldsymbol{w_1, w_2, \ldots, w_n}$ that form a coherent continuation. We usually **maximize**:

$$P(w_1, \ldots, w_T | p_1, \ldots, p_n; \boldsymbol{\theta}) = \prod_{t=1}^{T} P(w_t | p_{1:n}, w_{1:t-1}; \boldsymbol{\theta}),$$

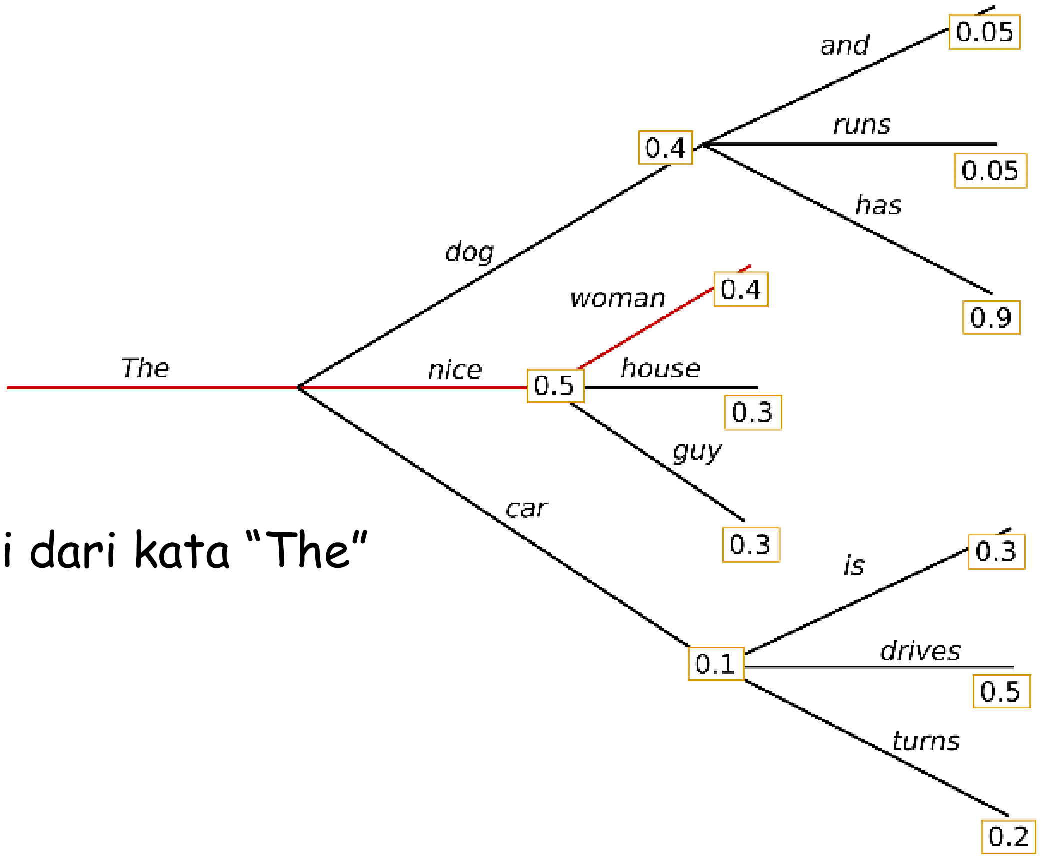
where T is the number of new tokens; w_t can be any word from the vocab list. It is also assumed that the model compute $P(.)$ using **left-to-right** decomposition.

However …

- Finding the optimum argmax sequence from recurrent neural language models or transformers is **not tractable**
  - Chen et al., "Recurrent Neural Networks as Weighted Language Recognizers", NAACL '18
- We need an approximation:
  - Greedy Search
  - Beam Search
  - …

Greedy Search

Mulai dari beberapa kata (prompt), urutan berikutnya dihasilkan dengan cara memilih kata dengan probabilitas paling tinggi.



Mulai dari kata “The”

Tend to repeat itself!

Prompt

“**I enjoy walking with my cute dog**, but I'm not sure if I'll ever be able to walk with my dog. **I'm not sure if I'll ever be able to walk with my dog**.”

https://huggingface.co/blog/how-to-generate

Greedy Search

```
from math import log
from numpy import array
from numpy import argmax
import itertools
import random

def random_probs(vocab_size):
  probs = []
  for _ in range(vocab_size):
    probs.append(random.randint(1, 10))
  total = sum(probs)
  return [v/total for v in probs]

def simulated_gpt(vocabs = ['A', 'B', 'C'], max_num_seq = 4):
  dict_prob = {}
  for i in range(1, max_num_seq):
    prods = list(itertools.product(vocabs, repeat = i))
    for prod in prods:
      dict_prob[''.join(prod)] = random_probs(len(vocabs))
  return dict_prob
```

Simulasi **P(w_t | context)** yang dikembalikan adalah sebuah dictionary, dimana key berisi semua kemungkinan context, dan value adalah probs untuk kata selanjutnya, **w_t**

Greedy Search

```python
def greedy_search_decoder(context, gpt, vocabs, max_seq):
  seq = context
  log_prob = 0.0
  for _ in range(max_seq - len(context)):
    probs = gpt[seq]   # simulasi next word generation
    max_i = argmax(probs)
    seq, log_prob = seq + vocabs[max_i], log_prob - log(probs[max_i])
  return seq, log_prob
```

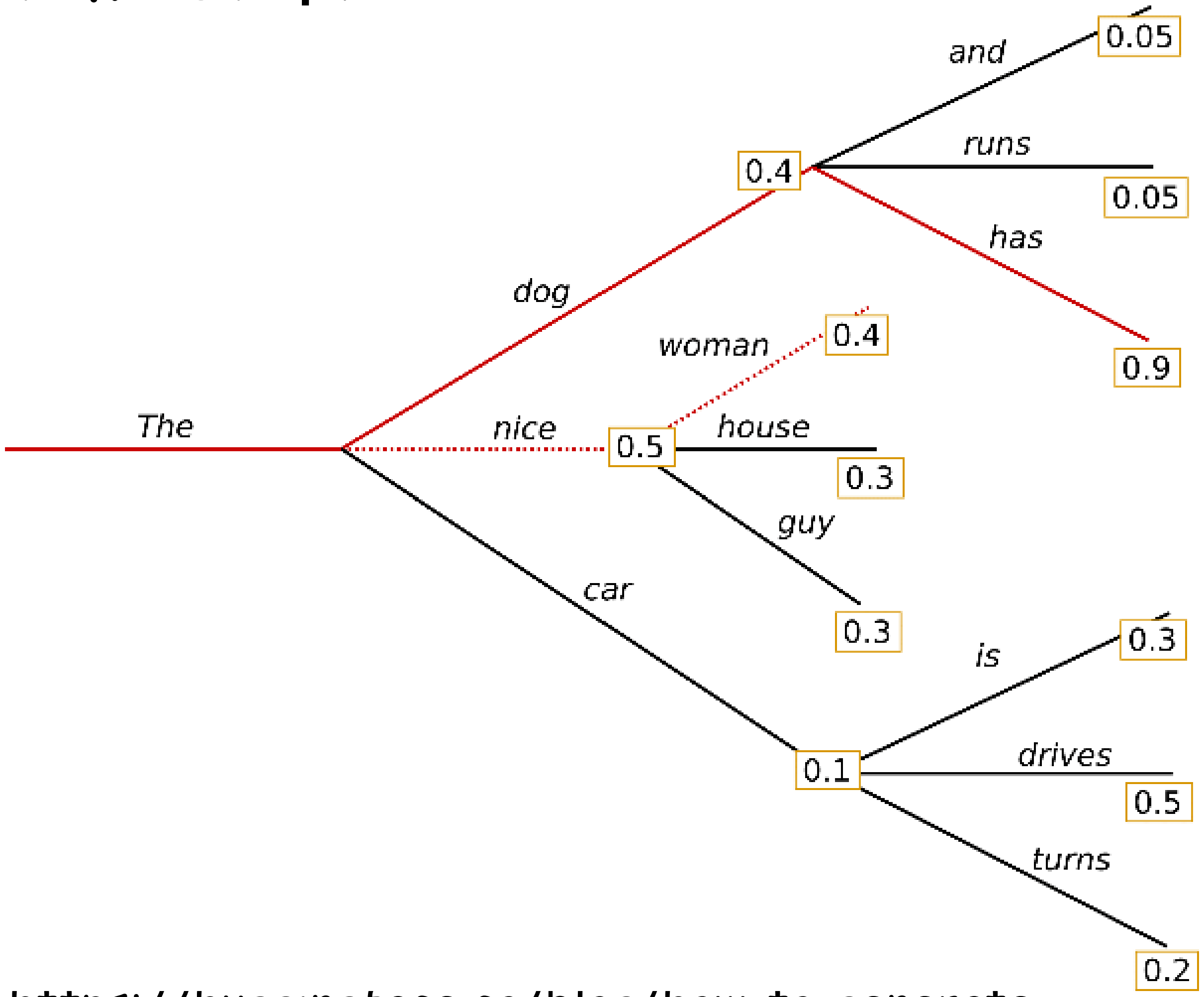
```python
vocabs = ['A', 'B', 'C', 'D', 'E', 'F', 'G']
max_seq = 5

gpt = simulated_gpt(vocabs = vocabs, max_num_seq = max_seq)
print(greedy_search_decoder("A", gpt, vocabs, max_seq))

# output (di salah satu kesempatan):
# ('ADCGE', 5.709277227357884)
```

Log-likelihood sequence (makin mendekati nol, makin bagus)

Beam Search

Jika **num_beams** = 1, Beam Search adalah Greedy Search!

keeping the most likely **num_beams** of hypotheses at each time step!



At time step 1, besides the most likely hypothesis **("The","nice")**, beam search also keeps track of the second most likely one **("The","dog")**.

At time step 2, beam search finds that the word sequence **("The","dog","has")**, has with 0.36 a higher probability than **("The","nice","woman")**, which has 0.2.

Beam search will always find an output sequence with higher probability than greedy search, but is not guaranteed to find the most likely output.

https://huggingface.co/blog/how-to-generate

Beam Search

```
def beam_search_decoder(context, gpt, vocabs, max_seq, num_beams):
  sequences = [(context, 0.0)]

  for _ in range(max_seq - len(context)):
    all_candidates = list()

    for i in range(len(sequences)):
      seq, score = sequences[i]
      probs = gpt[seq]   # simulasi next word generation

      for j in range(len(vocabs)):
        candidate = seq + vocabs[j], score - log(probs[j])
        all_candidates.append(candidate)

    # urutkan kandidat berdasarkan log-likelihood
    ordered = sorted(all_candidates, key = lambda tup: tup[1])

    # pilih num_beams terbaik
    sequences = ordered[:num_beams]
  return sequences
```

Beam Search

Greedy and Beam Search suffer from repetitive generation.

Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Beam Search, *b*=32:
"The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the Universidad Nacional Autónoma de México (UNAM) and the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de ..."

Pure Sampling:
They were cattle called Bolivian Cavalleros; they live in a remote desert uninterrupted by town, and they speak huge, beautiful, paradisiacal Bolivian linguistic thing. They say, 'Lunch, marge.' They don't tell what the lunch is," director Professor Chuperas Omwell told Sky News. "They've only been talking to scientists, like we're being interviewed by TV reporters. We don't even stick around to be interviewed by TV reporters. Maybe that's how they figured out that they're cosplaying as the Bolivian Cavalleros."

Figure 1: Even with substantial human context and the powerful GPT-2 Large language model, Beam Search (size 32) leads to degenerate repetition (highlighted in blue) while pure sampling leads to incoherent gibberish (highlighted in red). When $b \geq 64$, both GPT-2 Large and XL (774M and 1542M parameters, respectively) prefer to stop generating immediately after the given context.

Holtzman et al., The Curious Case of Neural Text Degeneration, ICLR 2020

Beam Search

Greedy and Beam Search suffer from repetitive generation.

Solution: **n-grams penalty!** to make sure that **no *n-gram* appears twice** by manually setting the probability of next words that could create an already seen *n-gram* to 0.

Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Beam Search, *b*=32:
"The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the Universidad Nacional Autónoma de México (UNAM) and the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de ..."

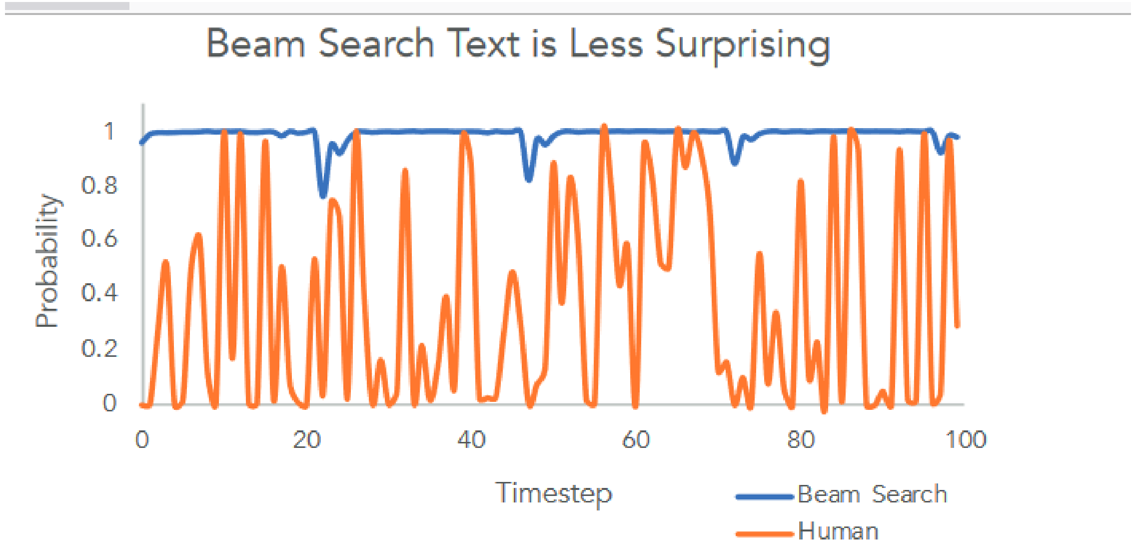
Pure Sampling:
They were cattle called Bolivian Cavalleros; they live in a remote desert uninterrupted by town, and they speak huge, beautiful, paradisiacal Bolivian linguistic thing. They say, 'Lunch, marge.' They don't tell what the lunch is," director Professor Chuperas Omwell told Sky News. "They've only been talking to scientists, like we're being interviewed by TV reporters. We don't even stick around to be interviewed by TV reporters. Maybe that's how they figured out that they're cosplaying as the Bolivian Cavalleros."

Figure 1: Even with substantial human context and the powerful GPT-2 Large language model, Beam Search (size 32) leads to degenerate repetition (highlighted in blue) while pure sampling leads to incoherent gibberish (highlighted in red). When $b \geq 64$, both GPT-2 Large and XL (774M and 1542M parameters, respectively) prefer to stop generating immediately after the given context.

Holtzman et al., The Curious Case of Neural Text Degeneration, ICLR 2020

Beam Search

Beam Search masih memberikan hasil yang **"kurang kreatif"**



Beam Search

...to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and to provide an overview of the current state-of-the-art in the field of computer vision and machine learning, and...

Human

...which grant increased life span and three years warranty. The Antec HCG series consists of five models with capacities spanning from 400W to 900W. Here we should note that we have already tested the HCG-620 in a previous review and were quite satisfied With its performance. In today's review we will rigorously test the Antec HCG-520, which as its model number implies, has 520W capacity and contrary to Antec's strong beliefs in multi-rail PSUs is equipped...

The probability assigned to tokens generated by Beam Search and humans, given the same context. Note the increased variance that characterizes human text, in contrast with the endless repetition of text decoded by Beam Search.

Holtzman et al., The Curious Case of Neural Text Degeneration, ICLR 2020

Several recent studies on open-ended generation have reported that **maximization-based decoding** does not lead to high quality text

Jadi, ternyata lebih “seru” jika “lebih random” ☺

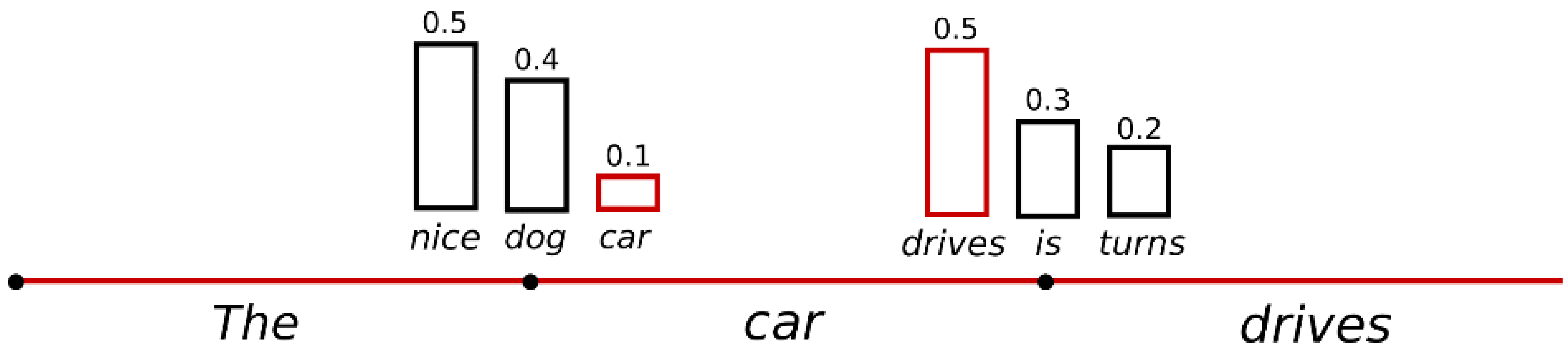
https://huggingface.co/blog/how-to-generate

Random Sampling

Generation is not a deterministic process anymore.

We randomly select the next word w_t according to its conditional distribution

$$w_t \sim P(w_t | p_{1:n}, w_{1:t-1}; \boldsymbol{\theta})$$



https://huggingface.co/blog/how-to-generate

Random Sampling

- It’s usually not coherent
- It doesn’t look like that it was written by a human

“I enjoy walking with my cute dog for the rest of the day, but this had me staying in an unusual room and not going on nights out with friends (which will always be wondered for a mere minute or so at this point).”

Holtzman et al., The Curious Case of Neural Text Degeneration, ICLR 2020

Sampling with Temperature

- Another common approach to sampling-based generation is to shape a probability distribution through temperature (Ackley et al., 1985)
- Given a temperature parameter $\boldsymbol{t}$, the softmax is re-estimated as:

Logits for word **w**

$$P(x = w | p_{1:n}, w_{1:t-1}; \boldsymbol{\theta}) = \frac{\exp(u_w / t)}{\sum_{w' \in V} \exp(u_{w'} / t)}$$

Setting $\boldsymbol{t} \in [\mathbf{0}, \mathbf{1})$ skews the distribution towards high probability events, which implicitly lowers the mass in the tail distribution.

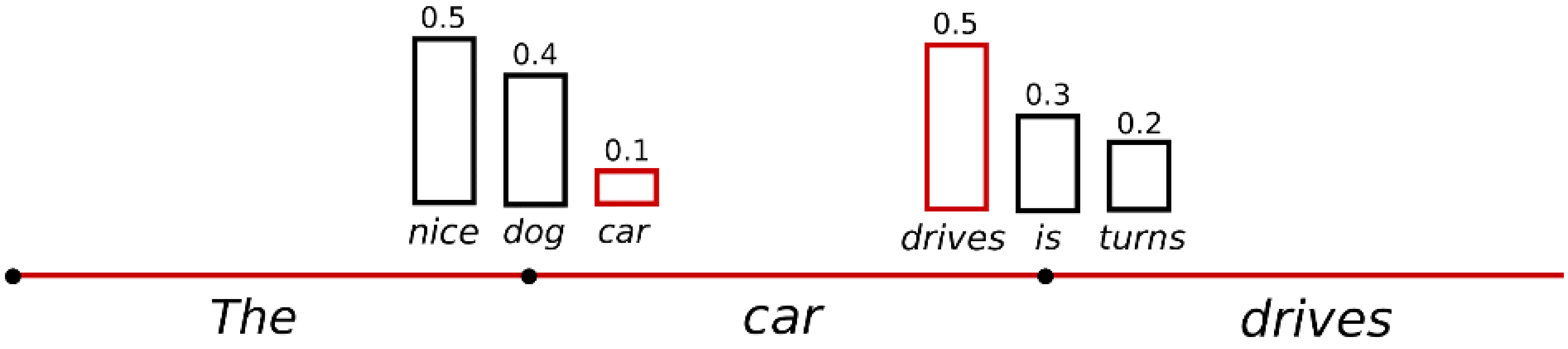
While applying temperature can make a distribution less random, in its limit, when setting **temperature →0**, temperature scaled sampling becomes equal to **greedy decoding** (**deterministic**) and will suffer from the same problems as before.

Sampling with Temperature

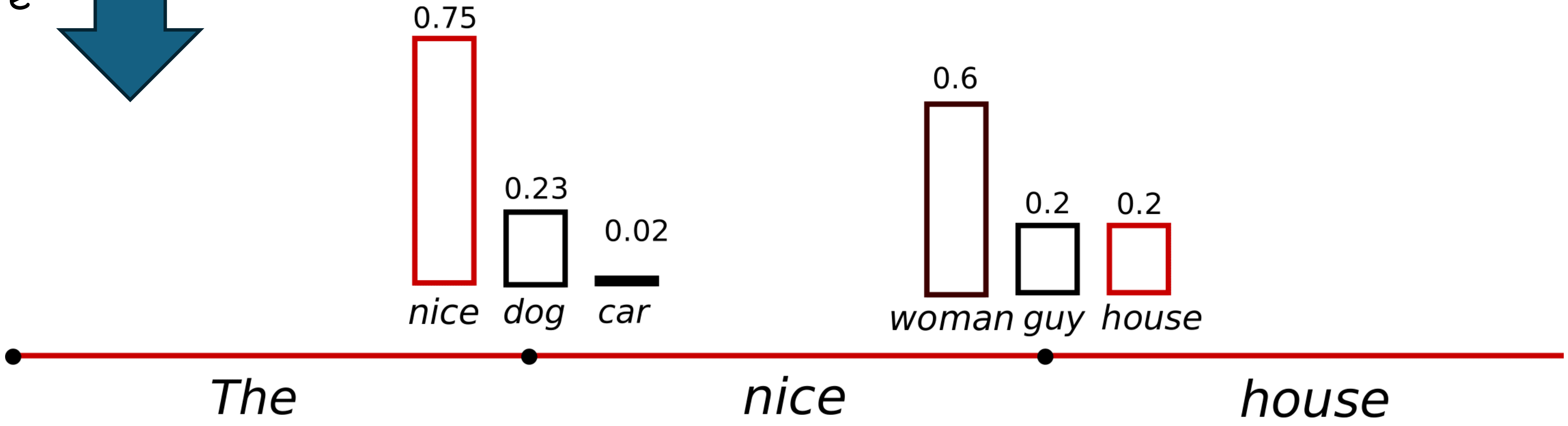
```
import numpy as np

def softmax(x, t):
    return np.exp(x/t) / np.sum(np.exp(x/t), axis=0)

logits = np.array([1.3, 2.1, 1.0])

print(softmax(logits, 0.01))    # [1.80485139e-35 1.00000000e+00 1.68891188e-48]
print(softmax(logits, 0.1))     # [3.35344532e-04 9.99647960e-01 1.66958211e-05]
print(softmax(logits, 0.5))     # [0.15380252 0.76178887 0.08440861]
print(softmax(logits, 0.9))     # [0.24102444 0.58627399 0.17270156]
print(softmax(logits, 1.0))     # [0.25212039 0.56110424 0.18677538]
```

https://huggingface.co/blog/how-to-generate

Sampling with Temperature



When applying temperature



https://huggingface.co/blog/how-to-generate

Random Sampling, temperature = 1

"I enjoy walking with my cute dog for the rest of the day, but this had me staying in an unusual room and not going on nights out with friends (which will always be wondered for a mere minute or so at this point)."

Sampling with temperature = 0.6

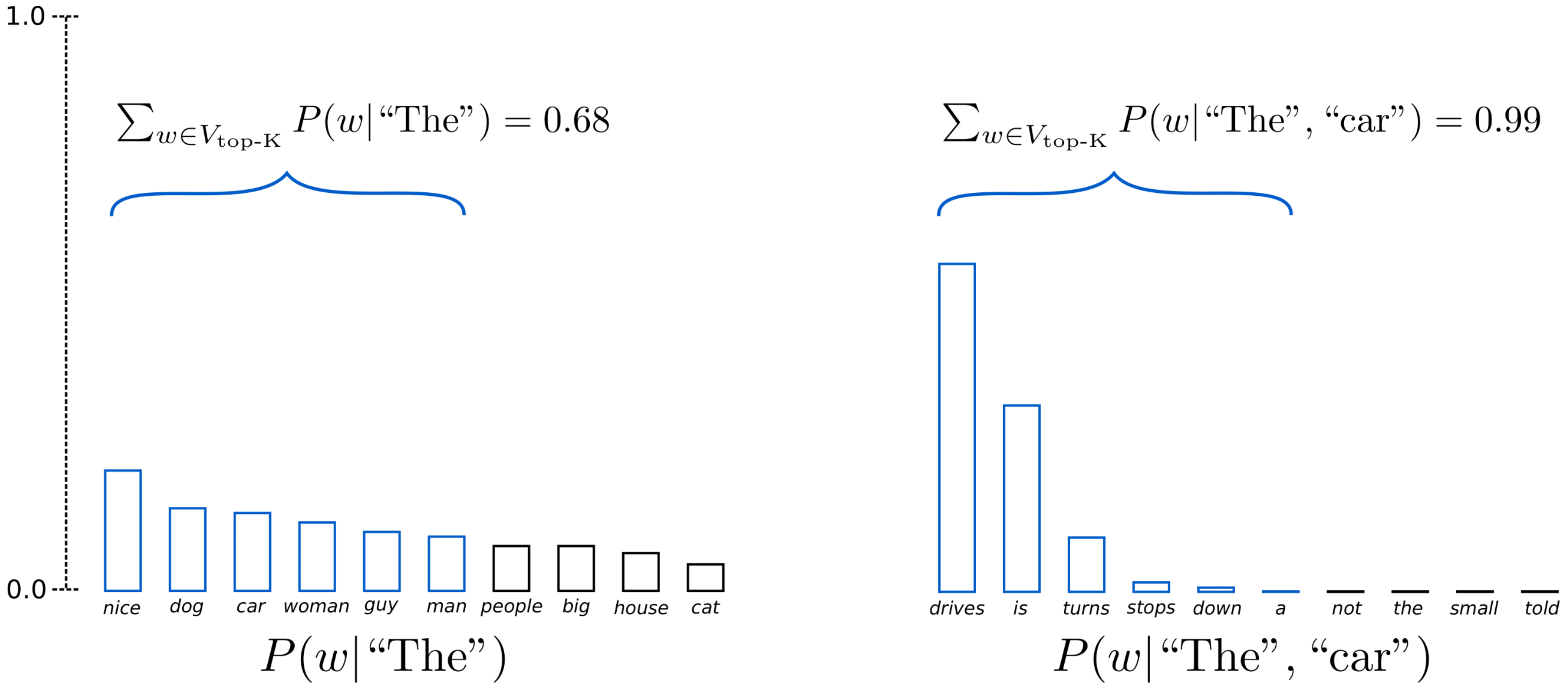
"I enjoy walking with my cute dog, but I don't like to chew on it. I like to eat it and not chew on it. I like to be able to walk with my dog."

A bit more coherent than the above one

https://huggingface.co/blog/how-to-generate

Top-K Sampling

The *K* most likely next words are filtered and the probability mass is **redistributed** among only those *K* next words.



Holtzman et al., The Curious Case of Neural Text Degeneration, ICLR 2020

Top-K Sampling

Find top-K vocab $\boldsymbol{V^{(K)}} \subset \boldsymbol{V}$ which **maximizes:**

$$p' = \sum_{x \in V^{(K)}} P(x|x_{1:T-1})$$

The distribution is then rescaled using:

Sampling is performed using this new distribution!

$$P'(x|x_{1:T-1}) = \begin{cases} P(x|x_{1:T-1})/p' & if\ x \in V^{(K)} \\ 0 & otherwise \end{cases}$$

Holtzman et al., The Curious Case of Neural Text Degeneration, ICLR 2020

Top-p (Nucleus) Sampling

Find top-p vocab $V^{(p)} \subset V$ as **the smallest set** such that:

$$c = \sum_{x \in V^{(p)}} P(x|x_{1:T-1}) \geq p$$

The distribution is then rescaled using:

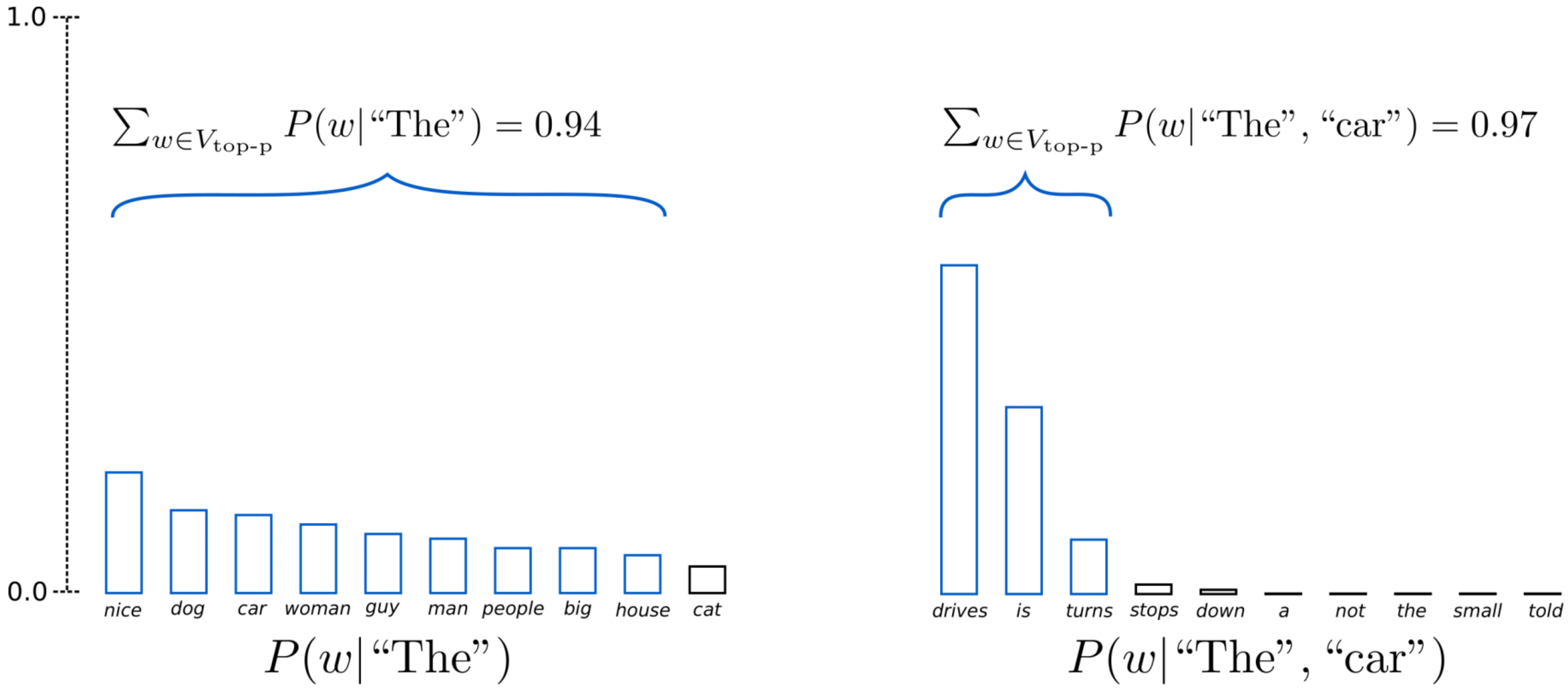
Sampling is performed using this new distribution!

$$P'(x|x_{1:T-1}) = \begin{cases} P(x|x_{1:T-1})/c & if\ x \in V^{(p)} \\ 0 & otherwise \end{cases}$$

https://huggingface.co/blog/how-to-generate

Top-p (Nucleus) Sampling

The **smallest possible set of words** whose cumulative probability exceeds the probability ***p***.



https://huggingface.co/blog/how-to-generate

Top-K = 50

"I enjoy walking with my cute dog for the rest of the day, but this time it was hard for me to figure out what to do with it. (One reason I asked this for a few months back is that I had a…"

Top-P = 0.92

"I enjoy walking with my cute dog for the rest of the day, but this had me staying in an unusual room and not going on nights out with friends (which will always be my yearning for such a spacious screen on my desk…"

An unprecedented number of mostly young whales have become stranded on the West Australian coast since 2008.

WebText

The number of stranded whales has increased by more than 50 per cent in the past year, with the number of stranded whales on the West Australian coast increasing by more than 50 per cent in the past year. The number of whales stranded on the West Australian coast has increased by more than 50 per cent in the past year, with the number of stranded whales on the West Australian coast increasing by more than 50 per cent in the past year.

Beam Search, b=16

The Australian Food Safety Authority has warned Australia's beaches may be revitalised this year because healthy seabirds and seals have been on the move. More than 50,000 seabirds, sea mammals and seahorses have been swept into the sea by the Holden CS118 and Adelaide Airport CS300 from 2013. A major white-bat and umidauda migration across Australia is under way in Australia for the first time, with numbers reaching an estimated 50,000.

Pure Sampling

Last week's intense storms and a series of powerful cyclones have been officially blamed for the deaths of at least nine large fin whales near Whitsundays - the largest loss of any species globally. The fin whales: packed in the belly of one killer whale thrashing madly in fear as another tries to bring it to safety. When the colossal animal breached the waters of Whitsundays, he'd been seen tagged for a decade.

Sampling, t=0.9

Pumping Station #3 shut down due to construction damage Find more at:
www.abc.net.au/environment/species-worry/
in-the-top-10-killer-whale-catastrophes-in-history.html
"In the top 10 killer whale catastrophes in history:
1) 1986: Up to 12 orcas struck by lightning; many drowned and many more badly injured.

Top-k, k=640

The whale's fate was confirmed late last week when the animal was found by fishermen off the coast of Bundaberg. Experts believe the whale was struck by a fishing vessel off the coast of Bundaberg, and died after being sucked into the ocean. The whale's fate was confirmed late last week when the animal was found by fishermen off the coast of Bundaberg.

Top-k, k=40, t=0.7

There has been an unprecedented number of calves caught in the nets of whaling stations that operate in WA. Pilot whales continue to migrate to feeding grounds to feed their calves. They are now vulnerable due to the decline of wild populations; they are restricted to one breeding site each year. Image copyright Yoon Bo Kim But, with sharp decline in wild populations the size of the Petrels are shrinking and dwindling population means there will only be room for a few new fowl.

Nucleus, p=0.95

Poor nutrition has led to a rise in the number of stranded humpback whales on the West Australian coast, veterinary researchers have said. Carly Holyoake, from Murdoch University, at the Australian Veterinary Association's annual conference in Perth on Wednesday, said an unprecedented number of mostly young whales had become stranded on the coast since 2008.

WebText

Holtzman et al., The Curious Case of Neural Text Degeneration, ICLR 2020

Example generations continuing an initial sentence.

Maximization and top-k truncation methods lead to copious repetition (**highlighted in blue**), while sampling with and without temperature tends to lead to incoherence (**highlighted in red**).

Nucleus Sampling largely avoids both issues.

Nilai UAS **+ 8 Point**. Siapa yang mau?

- Coba kode GPT from scratch, oprek hyperparameter yang ada, termasuk banyaknya tumpukan decoder
  - https://colab.research.google.com/drive/1kVfkWBseA39UbH-rZCIC6L4bIZK4OVrw?usp=sharing
- Latih secara Causal Language Modelling dengan dokumen teks berukuran **minimal 2000 token** (boleh domain specific, misal puisi, buku, atau apapun)
- Coba lakukan prompting dengan beberapa kalimat konteks dengan berbagai decoding strategies.
- **+ 5 lagi**, jika bisa **implementasi from scratch**, variasi-variasi yang ada di dalam cell decoder, seperti:
  - https://arxiv.org/pdf/2105.14103
  - https://medium.com/nebius/transformer-alternatives-in-2024-06cd3d91d42b
  - Atau yang lainnya …

Setelah melakukan *unsupervised pre-training* (*maximizing the likelihood of a causal language model objective*), kita dapat mengadaptasi parameter *pre-trained GPT* untuk menyelesaikan beberapa *supervised tasks*.

Ini yang namanya supervised fine-tuning.

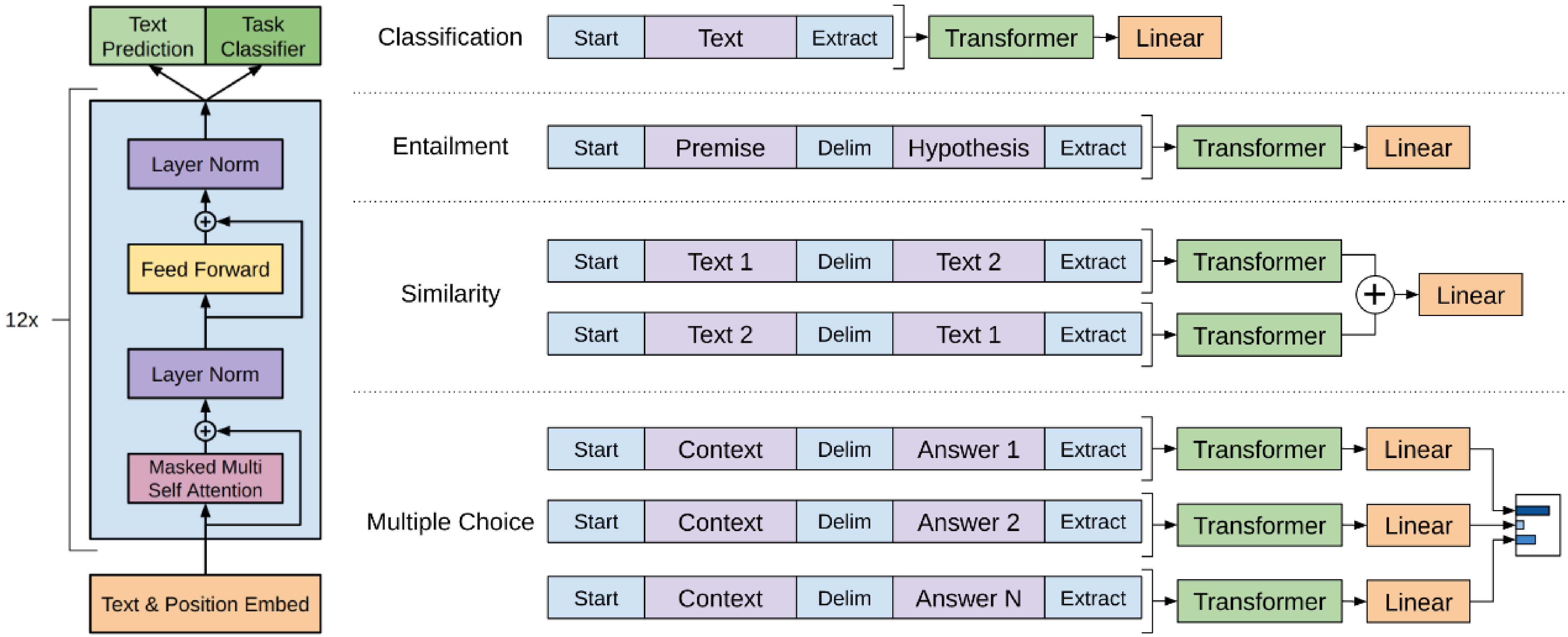


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

Radford et al., Improving Language Understanding by Generative Pre-Training

Suppose we have a dataset D consisting of a sequence of input tokens, $x_1, \ldots, x_m$, with a label y.

The inputs are passed through our pre-trained model to obtain the final transformer block h_{last}, which is then fed into an additional linear output layer with parameter W_y to predict y :

$$P(y|x_1, \ldots, x_m) = softmax(h_{last} W_y)$$

The objective is a combination between language modelling objective and supervised task objective.

Given unlabelled sentences: $\{\boldsymbol{x}_{\boldsymbol{1}}^{\boldsymbol{u}}, \boldsymbol{x}_{\boldsymbol{1}}^{\boldsymbol{u}}, \ldots, \boldsymbol{x}_{\boldsymbol{m}}^{\boldsymbol{u}}\}$

labelled sentences: $\{(\boldsymbol{x}_{\boldsymbol{1}}, y_1), \ldots, (\boldsymbol{x}_{\boldsymbol{n}}, y_n)\}$

$$L_{uns}(D) = \sum_i \sum_t \log\left[P\left(x_{i,t}^u \middle| x_{i,t-1}^u, \ldots, x_{i,t-k}^u\right)\right]$$

$$L_{sup}(D) = \sum_i \log\left[P\left(y_i \middle| x_{i,1}, \ldots, x_{i,m}\right)\right]$$

$$L_{tot}(D) = L_{sup}(D) + \lambda . L_{uns}(D)$$