AI VIETNAM
All-in-One Course

Applications of Transformer Encoder

Quang-Vinh Dinh
Ph.D. in Computer Science

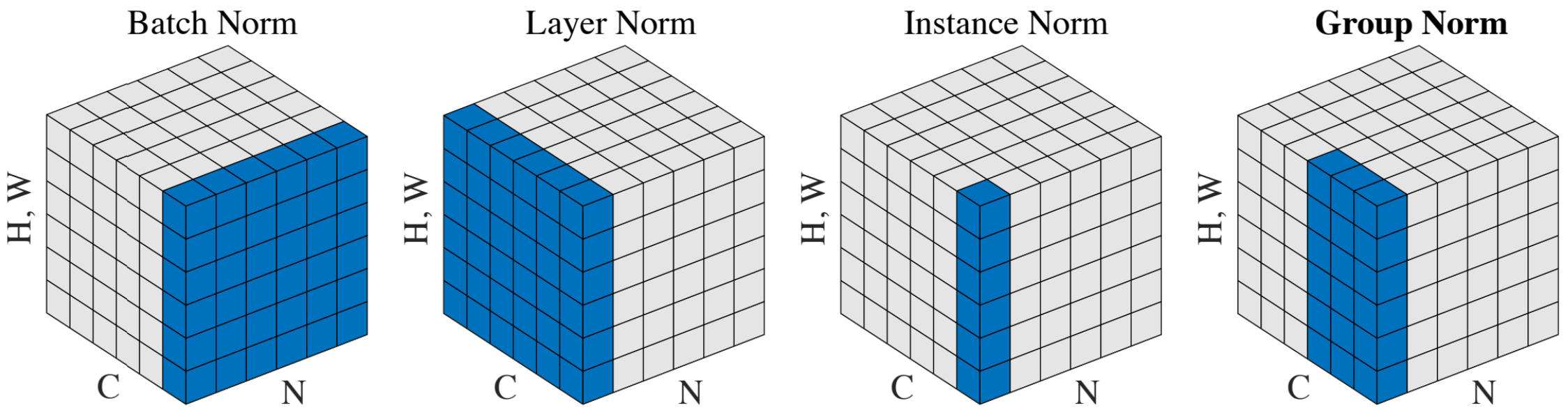
Year 2023

Outline

- Layer Normalization
- Transformers Block
- BERT and Text Classification
- Vision Transformer and Image Classification
- For Time-series and Tabular Data

Normalization

❖ Overview



$$\mu_c = \frac{1}{N \times H \times W} \sum_{i=1}^{N} \sum_{j=1}^{H} \sum_{k=1}^{W} F_{ijk}$$

$$\sigma_c = \sqrt{\frac{1}{N \times H \times W} \sum_{i=1}^{N} \sum_{j=1}^{H} \sum_{k=1}^{W} \left(F_{ijk} - \mu_c\right)^2}$$

https://arxiv.org/pdf/1803.08494.pdf

Batch Normalization

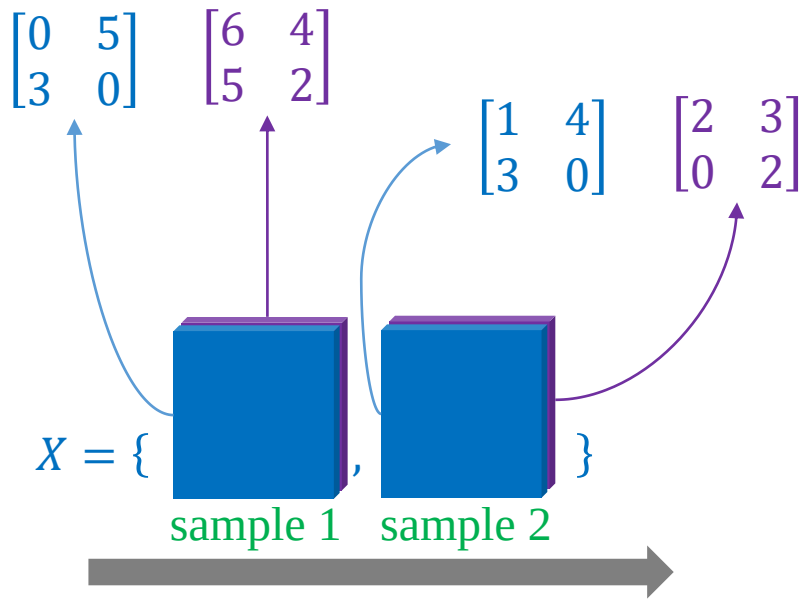
$\epsilon = 10^{-5}$

$\mu = [2.0, 3.0]$

$\sigma^2 = [3.5, 3.25]$

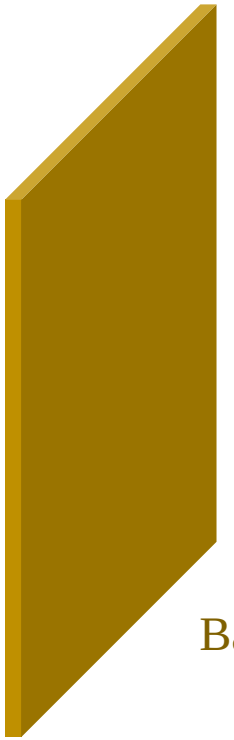
$\gamma = 1.0$

$\beta = 0.0$

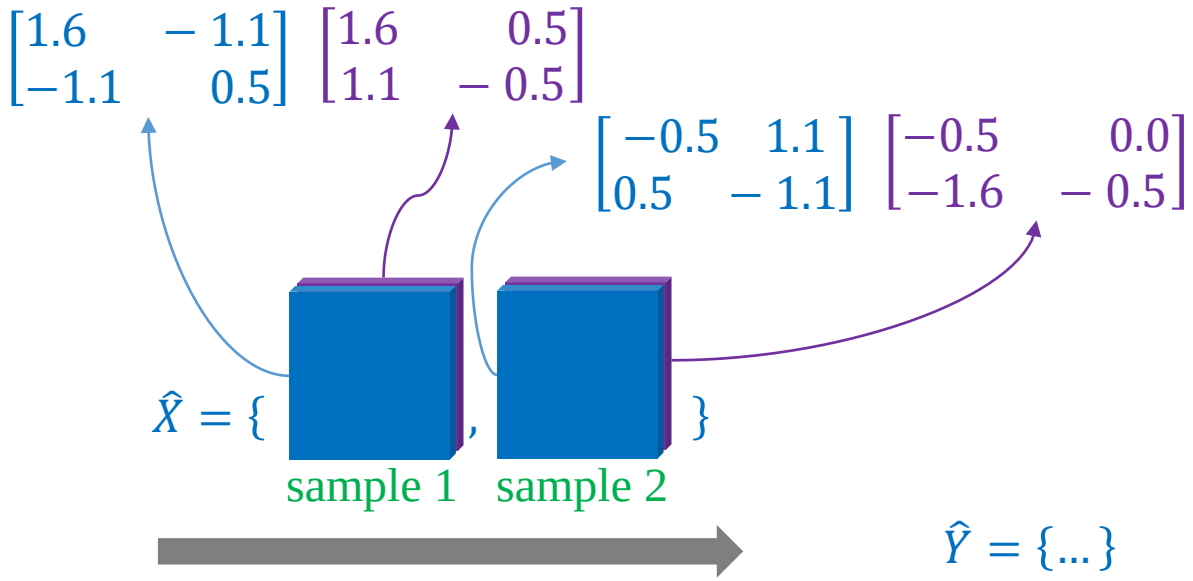


batch-size = 2

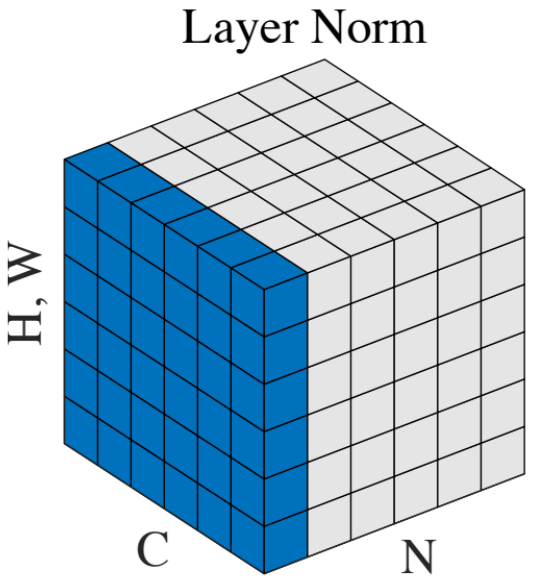
input_shape = (2, 2, 2, 2)



Batch-Norm Layer



Layer Normalization



https://arxiv.org/pdf/1607.06450.pdf

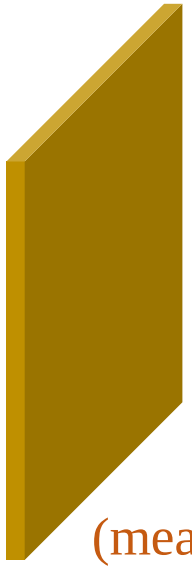
$$\mu_n = \frac{1}{C \times H \times W} \sum_{c=1}^{C} \sum_{j=1}^{H} \sum_{k=1}^{W} F_{cjk}$$

$$\sigma_n = \sqrt{\frac{1}{C \times H \times W} \sum_{c=1}^{C} \sum_{j=1}^{H} \sum_{k=1}^{W} (F_{cjk} - \mu_n)^2}$$

```
data =  np.array([[[8., 1.],
                   [1., 6.]]]).reshape(1, 2, 2, 1) #...
lnorm = tf.keras.layers.LayerNormalization(axis=[1, 2, 3])
output = lnorm(data)
```

$$X = \begin{bmatrix} 5 & 1 \\ 2 & 8 \end{bmatrix}$$

shape=(1, 2, 2, 1)



Layer Norm
(mean=4 & std=2.73)

$$\hat{X} = \begin{bmatrix} 0.36 & -1.09 \\ -0.73 & 1.46 \end{bmatrix}$$

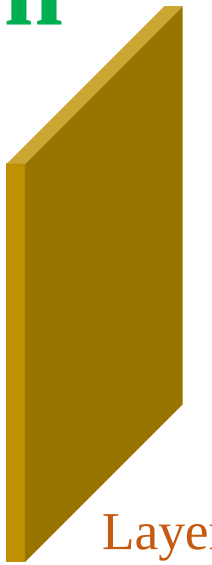
shape=(1, 2, 2, 1)

Layer Normalization

a sample

$$X = \left\{ \begin{bmatrix} 8 & 6 \\ 2 & 4 \end{bmatrix}, \begin{bmatrix} 0 & 2 \\ 1 & 5 \end{bmatrix} \right\}$$

input_shape=(1, 2, 2, 2)



Layer Norm
(mean=3.5 & std=2.54)

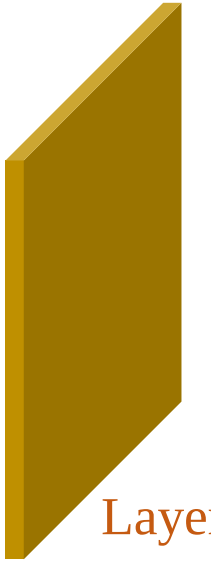
$$\hat{X} = \left\{ \begin{bmatrix} 1.76 & 0.98 \\ -0.58 & 0.19 \end{bmatrix}, \begin{bmatrix} -1.37 & -0.58 \\ -0.98 & 0.58 \end{bmatrix} \right\}$$

shape=(1, 2, 2, 2)

sample 1 sample 2

$$X = \left\{ \begin{bmatrix} 8 & 6 \\ 2 & 4 \end{bmatrix}; \begin{bmatrix} 0 & 2 \\ 1 & 5 \end{bmatrix} \right\}$$

input_shape=(2, 2, 2, 1)



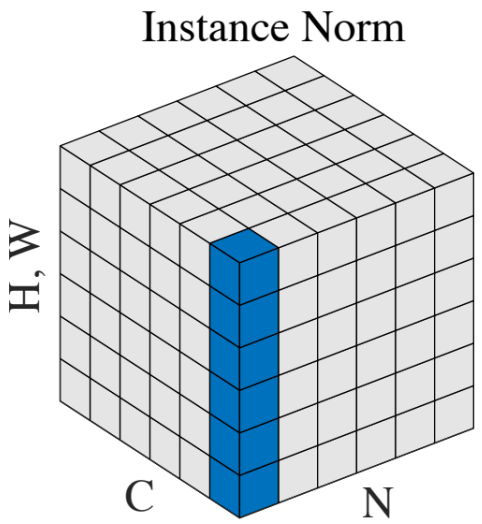
Layer Norm
(mean1=5.0 & mean2=2.0)
(std1=2.23 & std2=1.87)

sample 1 sample 2

$$\hat{X} = \left\{ \begin{bmatrix} 1.34 & 0.44 \\ -1.34 & -0.44 \end{bmatrix}; \begin{bmatrix} -1.06 & 0.0 \\ -0.53 & 1.6 \end{bmatrix} \right\}$$

shape=(2, 2, 2, 1)

Instance Normalization



https://arxiv.org/pdf/1607.08022.pdf

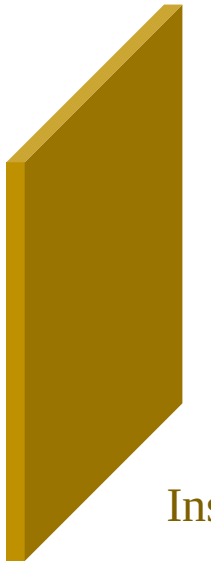
$$\mu = 2.5$$
$$\sigma^2 = 2.06$$

$$\mu_{instance} = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} F_{ij}$$

$$\sigma_{instance} = \sqrt{\frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} \left(F_{ij} - \mu_{instance}\right)^2}$$

$$X = \begin{bmatrix} 1 & 5 \\ 4 & 0 \end{bmatrix}$$

shape=(1, 2, 2, 1)



Instance-Norm Layer

$$\hat{X} = \begin{bmatrix} -0.7276 & 1.2127 \\ 0.7276 & -1.2127 \end{bmatrix}$$

shape=(1, 2, 2, 1)

Instance Normalization

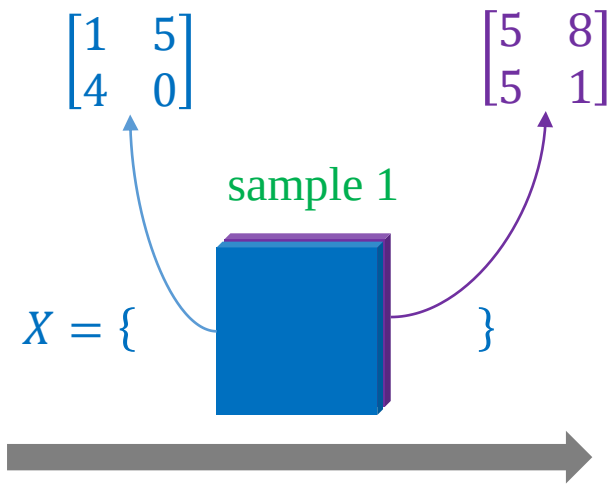
$\mu = [2.5, 4.75]$
$\sigma^2 = [4.25, 6.18]$

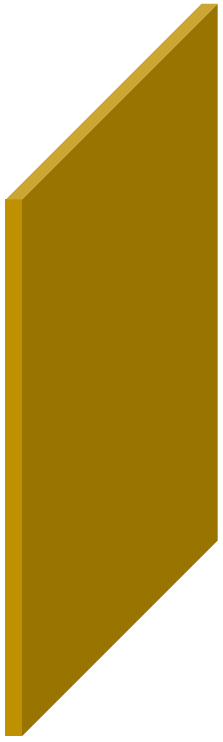
$\gamma = 1.0$
$\beta = 0.0$

$\epsilon = 10^{-5}$

$$\mu_{instance} = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} F_{ij}$$

$$\sigma_{instance} = \sqrt{\frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} (F_{ij} - \mu_{instance})^2}$$





$$\hat{X} = \left\{ \begin{bmatrix} -0.72 & 1.21 \\ 0.72 & -1.21 \end{bmatrix} \begin{bmatrix} 0.1 & 1.3 \\ 0.1 & -1.5 \end{bmatrix} \right\}$$

batch-size = 1

input_shape = (1, 2, 2, 2)

Instance-Norm Layer

Instance Normalization

$\mu = [2.5, 5.0]$

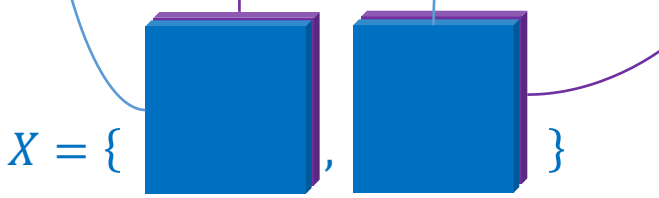
$\sigma^2 = [4.25, 7.7]$

$\begin{bmatrix} 1 & 5 \\ 4 & 0 \end{bmatrix} \quad \begin{bmatrix} 9 & 2 \\ 6 & 3 \end{bmatrix}$

$\mu = [4.25, 1.75]$

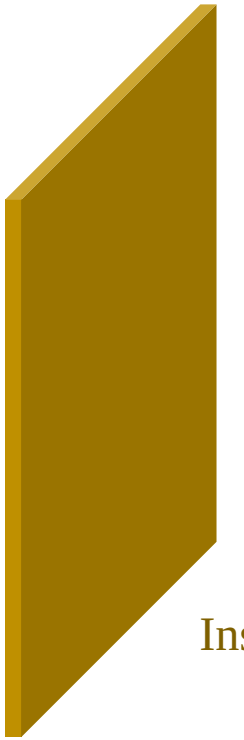
$\sigma^2 = [5.68, 2.18]$

$\begin{bmatrix} 6 & 3 \\ 1 & 7 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 \\ 1 & 4 \end{bmatrix}$



batch-size = 2

input_shape = (2, 2, 2, 2)



Instance-Norm Layer

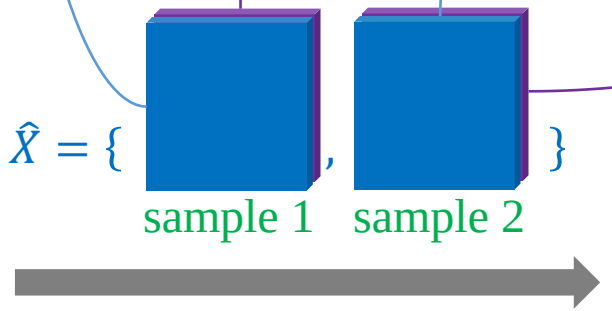
$\gamma = 1.0$

$\beta = 0.0$

$\epsilon = 10^{-5}$

$\begin{bmatrix} -0.72 & 1.21 \\ 0.72 & -1.21 \end{bmatrix} \quad \begin{bmatrix} 1.46 & -1.09 \\ 0.36 & -0.73 \end{bmatrix}$

$\begin{bmatrix} 0.73 & -0.52 \\ -1.36 & 1.15 \end{bmatrix} \quad \begin{bmatrix} -1.12 & 0.16 \\ -0.5 & -1.52 \end{bmatrix}$

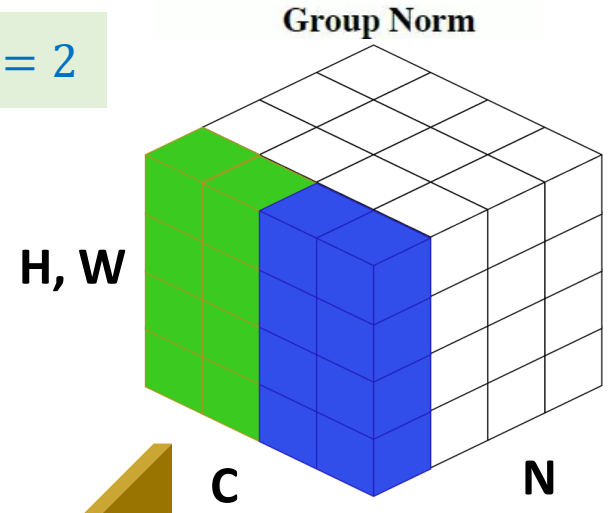


$\hat{Y} = \{...\}$

Group Normalization

$num\_chanels(C) = 4$ $num\_groups(G) = 2$

$num\_samples = 2$



$$S_i = \left\{k \mid k_n = i_n, \left\lfloor \frac{k_C}{C/G} \right\rfloor = \left\lfloor \frac{i_C}{C/G} \right\rfloor\right\}$$

$$\mu_{gr} = \frac{1}{m}\sum_{k \in S_i} x_k$$

$$\sigma_{gr} = \sqrt{\frac{1}{m}\sum_{k \in S_i}(x_k - \mu_{gr})^2}$$

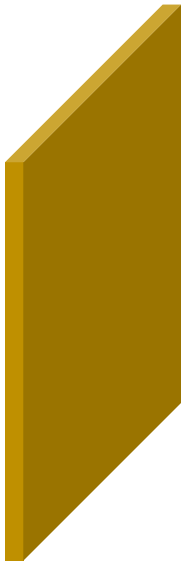
$\mu = 3$ $\sigma^2 = 5$ | $\mu = 4$ $\sigma^2 = 9.5$

$$X_1 = \left\{\begin{bmatrix}1 & 5\\ 4 & 7\end{bmatrix}, \begin{bmatrix}1 & 2\\ 4 & 0\end{bmatrix}, \begin{bmatrix}9 & 2\\ 0 & 3\end{bmatrix}, \begin{bmatrix}6 & 3\\ 1 & 8\end{bmatrix}\right\}$$

$$X_2 = \left\{\begin{bmatrix}5 & 2\\ 6 & 3\end{bmatrix}, \begin{bmatrix}1 & 7\\ 0 & 7\end{bmatrix}, \begin{bmatrix}0 & 2\\ 3 & 3\end{bmatrix}, \begin{bmatrix}1 & 4\\ 2 & 5\end{bmatrix}\right\}$$

$\mu = 3.8$ $\sigma^2 = 6.6$ | $\mu = 2.5$ $\sigma^2 = 2.25$

shape=(2, 2, 2, 4)



Group-Norm Layer

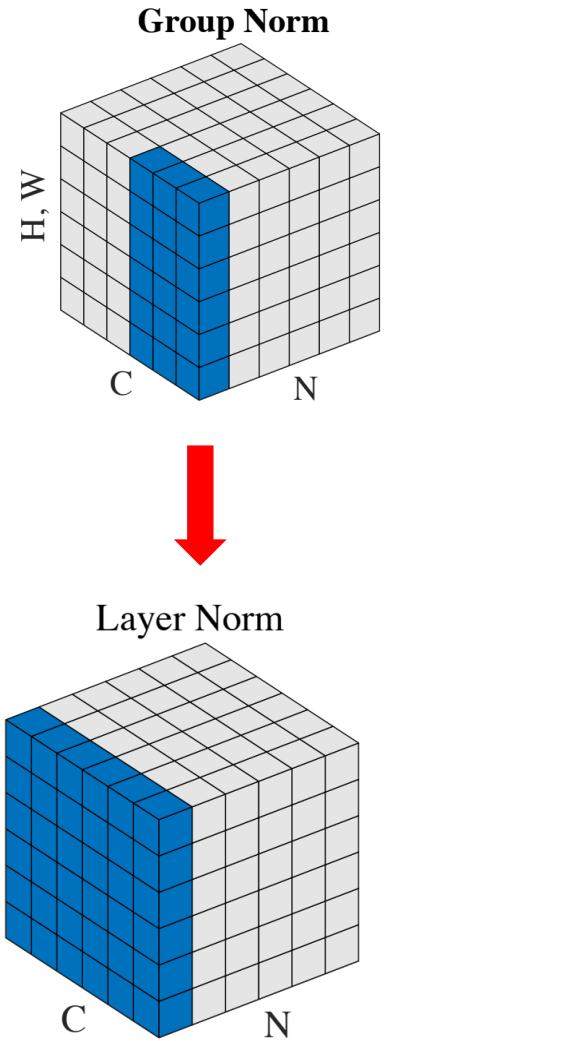
$$\hat{X}_1 = \left\{\begin{bmatrix}-0.89 & 0.89\\ 0.44 & 1.78\end{bmatrix}, \begin{bmatrix}-0.9 & -0.44\\ 0.44 & -1.3\end{bmatrix}, \begin{bmatrix}1.62 & -0.64\\ -1.29 & -0.32\end{bmatrix}, \begin{bmatrix}0.64 & -0.32\\ -0.97 & 1.29\end{bmatrix}\right\}$$

$$\hat{X}_2 = \left\{\begin{bmatrix}0.43 & -0.72\\ 0.82 & -0.34\end{bmatrix}, \begin{bmatrix}-1.11 & 1.21\\ -1.5 & 1.21\end{bmatrix}, \begin{bmatrix}-1.66 & -0.33\\ 0.33 & 0.33\end{bmatrix}, \begin{bmatrix}-1.0 & 1.0\\ -0.33 & 1.67\end{bmatrix}\right\}$$

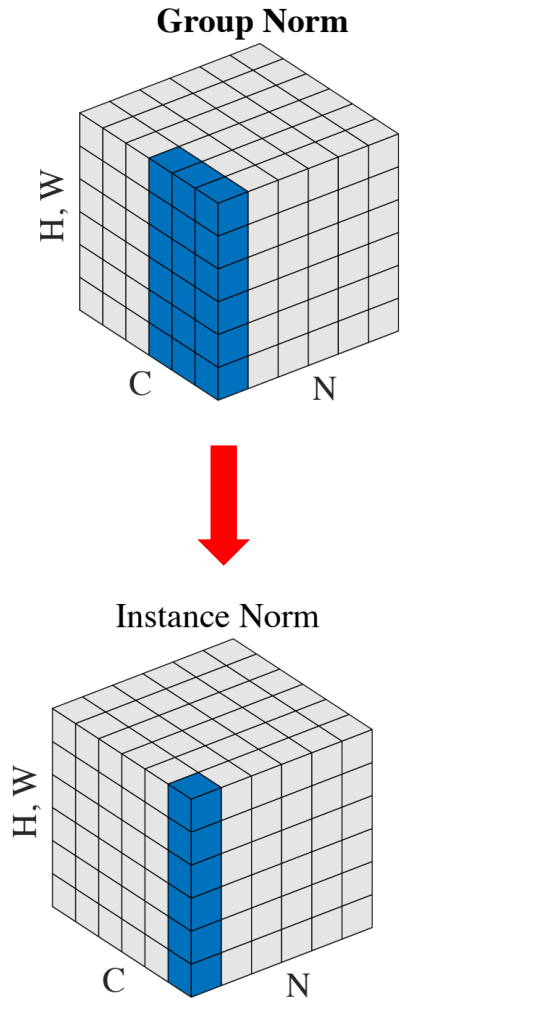
shape=(2, 2, 2, 4)

Group Normalization

$num\_chanels = 6$ $num\_groups = 1$



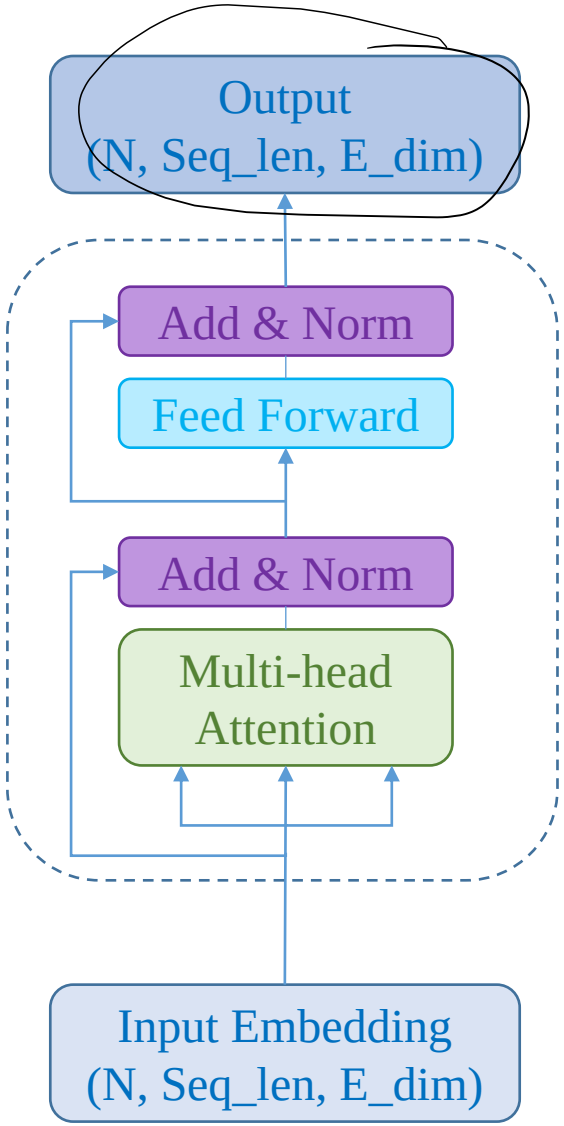
$num\_chanels = 6$ $num\_groups = 6$

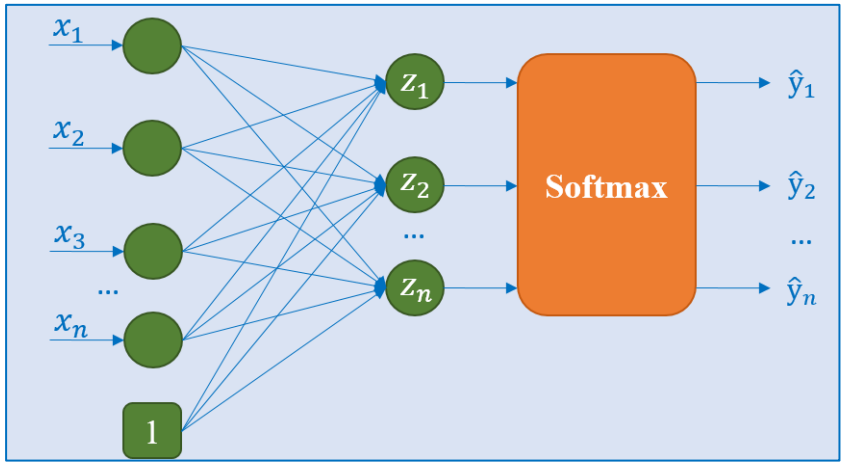


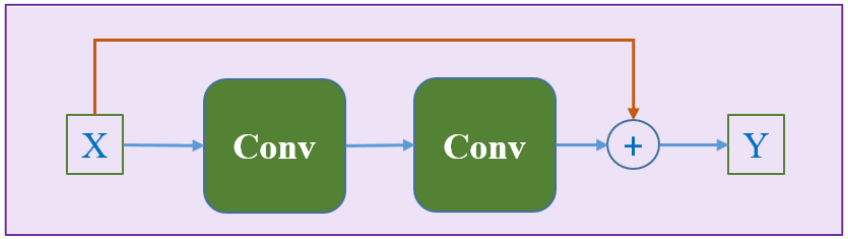
Outline

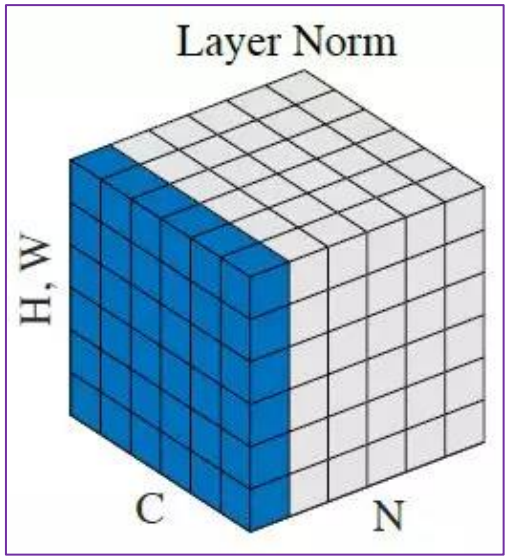
- Layer Normalization
- Transformers Block
- BERT and Text Classification
- Vision Transformer and Image Classification
- For Time-series and Tabular Data

Transformer Block (-)

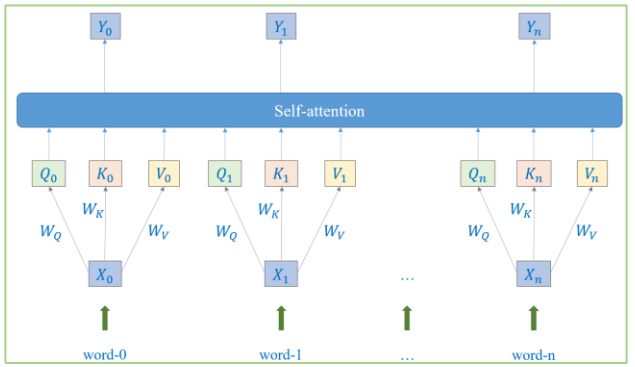






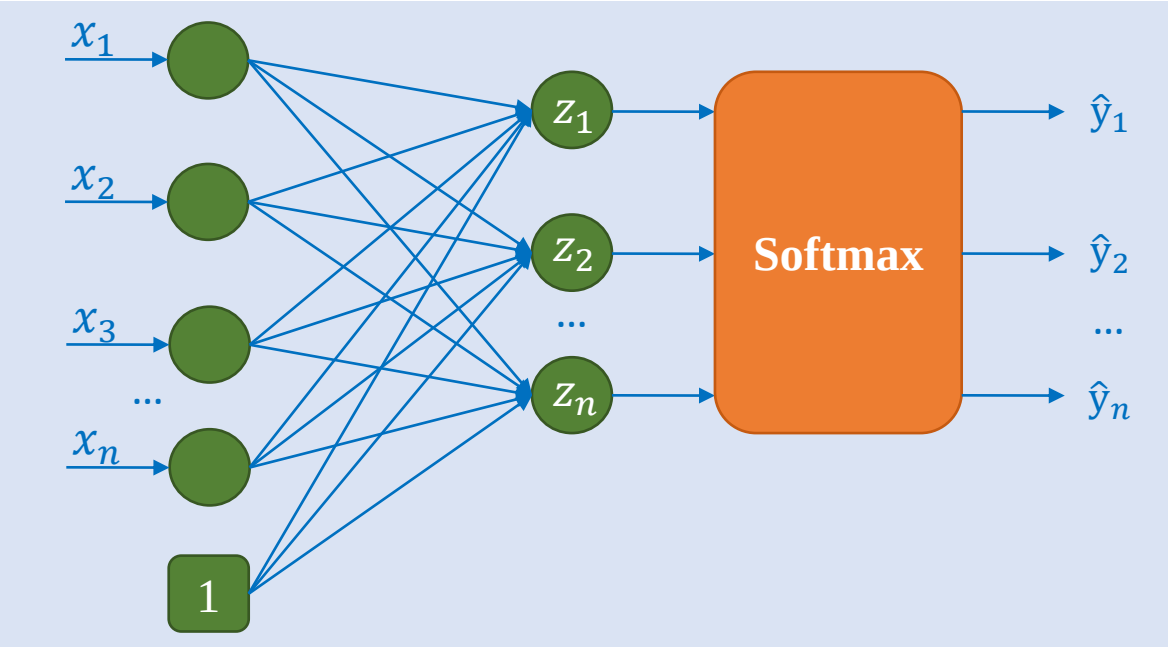


https://arxiv.org/pdf/1803.08494.pdf

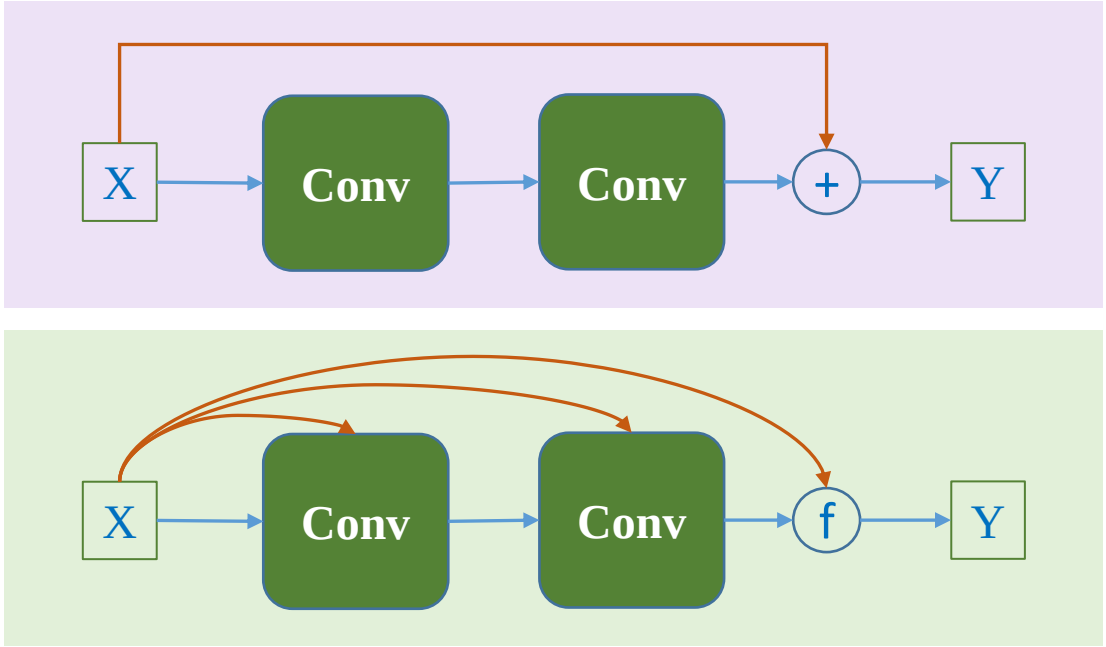


Group Normalization

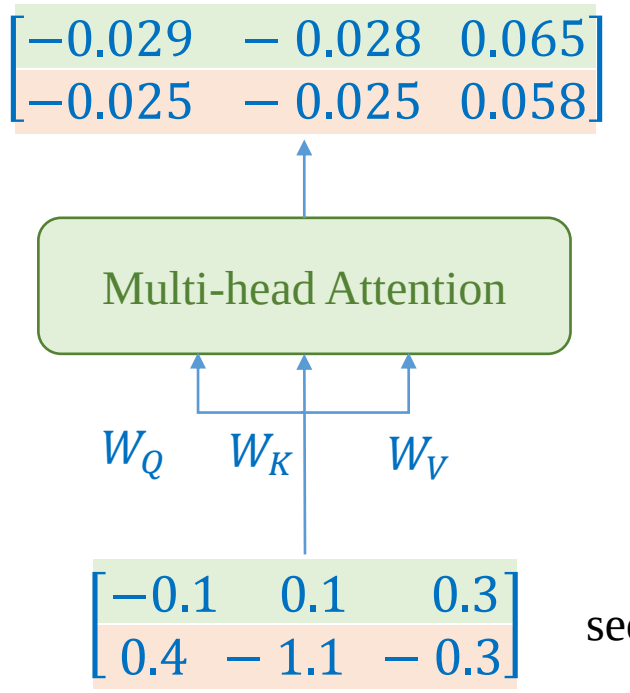
MLP + Softmax



Skip Connection



Transformer Block



bs = 1
sequence_length = 2
embed_dim = 3

bs = 1
sequence_length = 2
embed_dim = 3

$$W_Q = \begin{bmatrix} -0.35 & 0.51 & 0.50 \\ 0.36 & -0.47 & -0.29 \\ -0.51 & -0.14 & -0.56 \end{bmatrix}$$

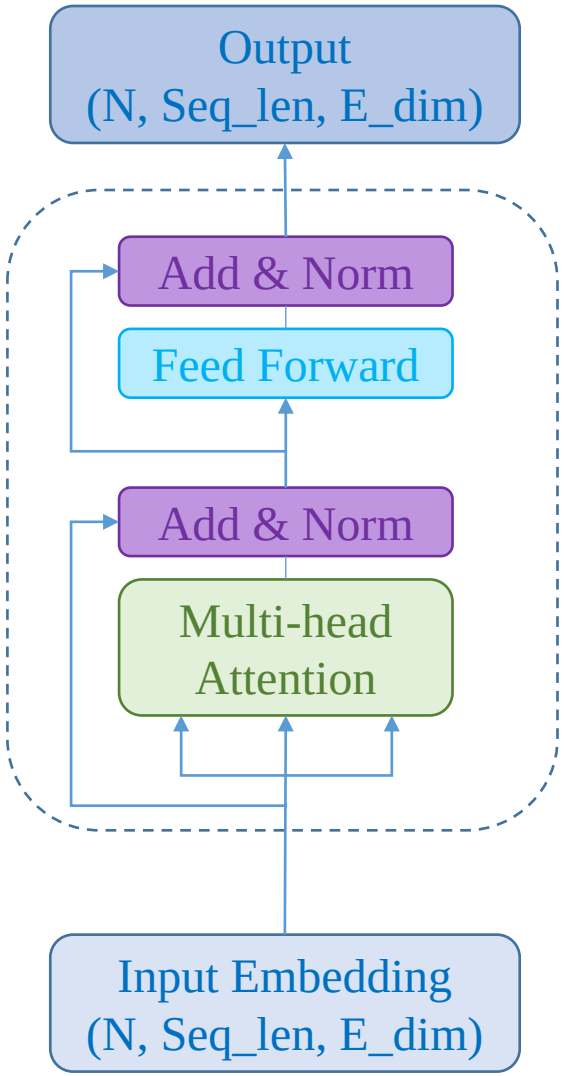
$$W_K = \begin{bmatrix} -0.49 & -0.68 & 0.18 \\ -0.44 & -0.46 & 0.18 \\ 0.07 & -0.10 & 0.44 \end{bmatrix}$$

$$W_V = \begin{bmatrix} -0.41 & 0.39 & -0.65 \\ -0.40 & -0.07 & -0.34 \\ -0.55 & -0.13 & -0.29 \end{bmatrix}$$

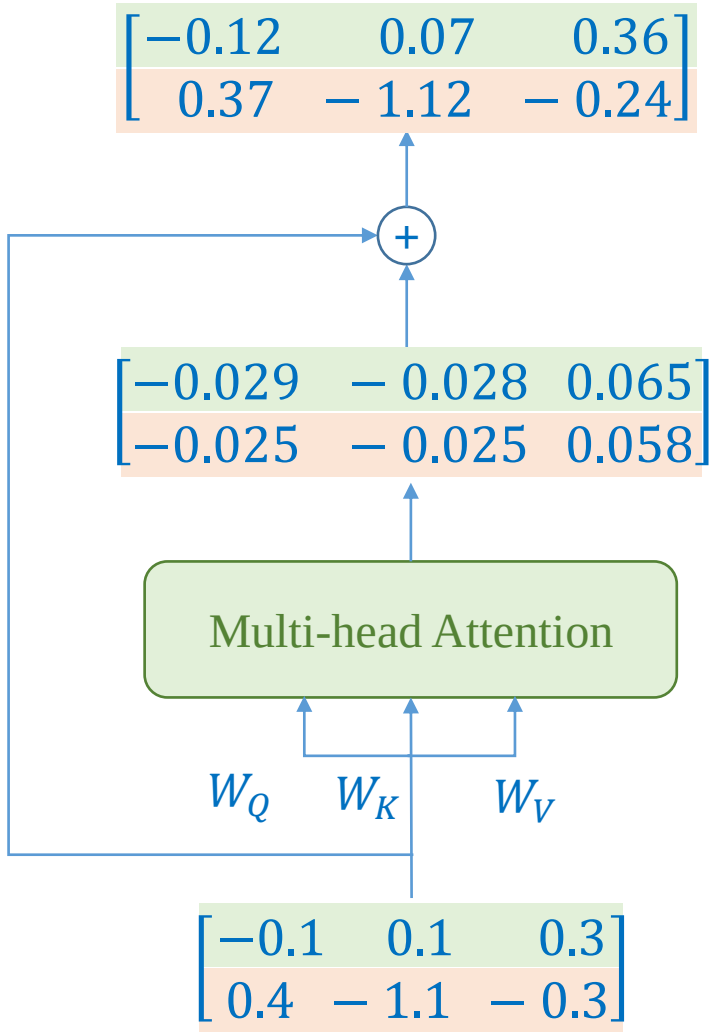
$$W_O = \begin{bmatrix} -0.36 & -0.08 & 0.32 \\ 0.27 & 0.05 & 0.15 \\ -0.05 & -0.28 & 0.05 \end{bmatrix}$$

$$Y = sigmoid\left(\frac{QK^T}{\sqrt{d}}\right)V$$

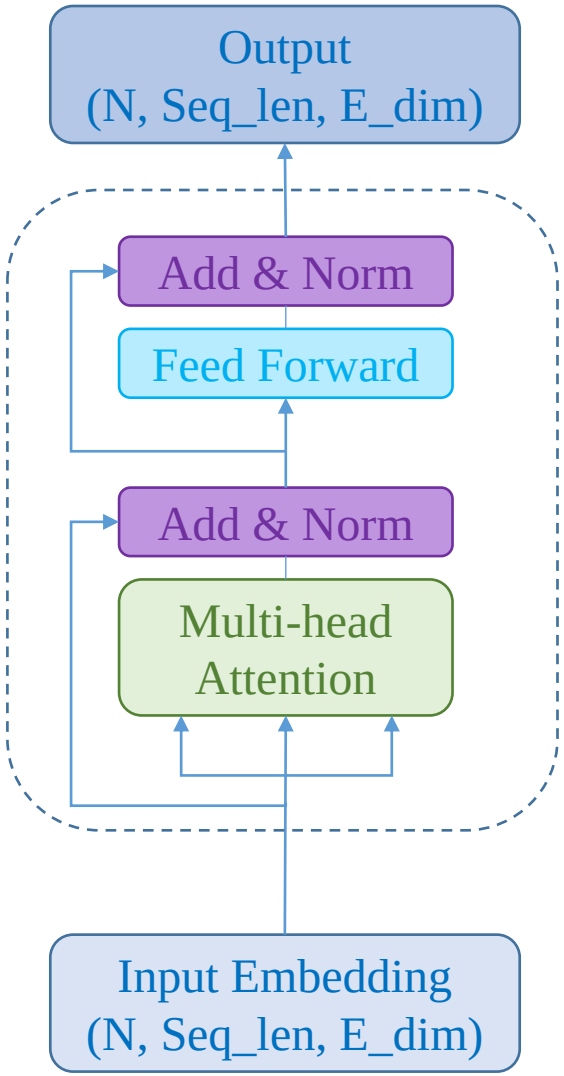
Transformer Block

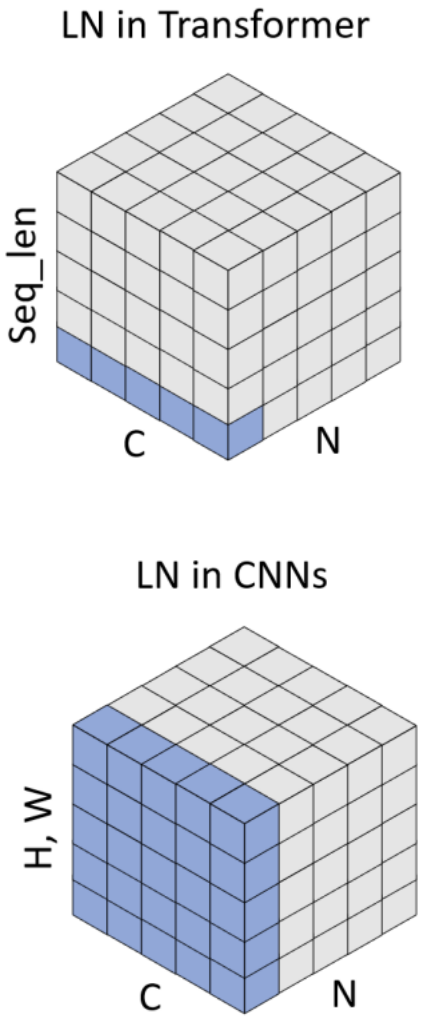


$$Y = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

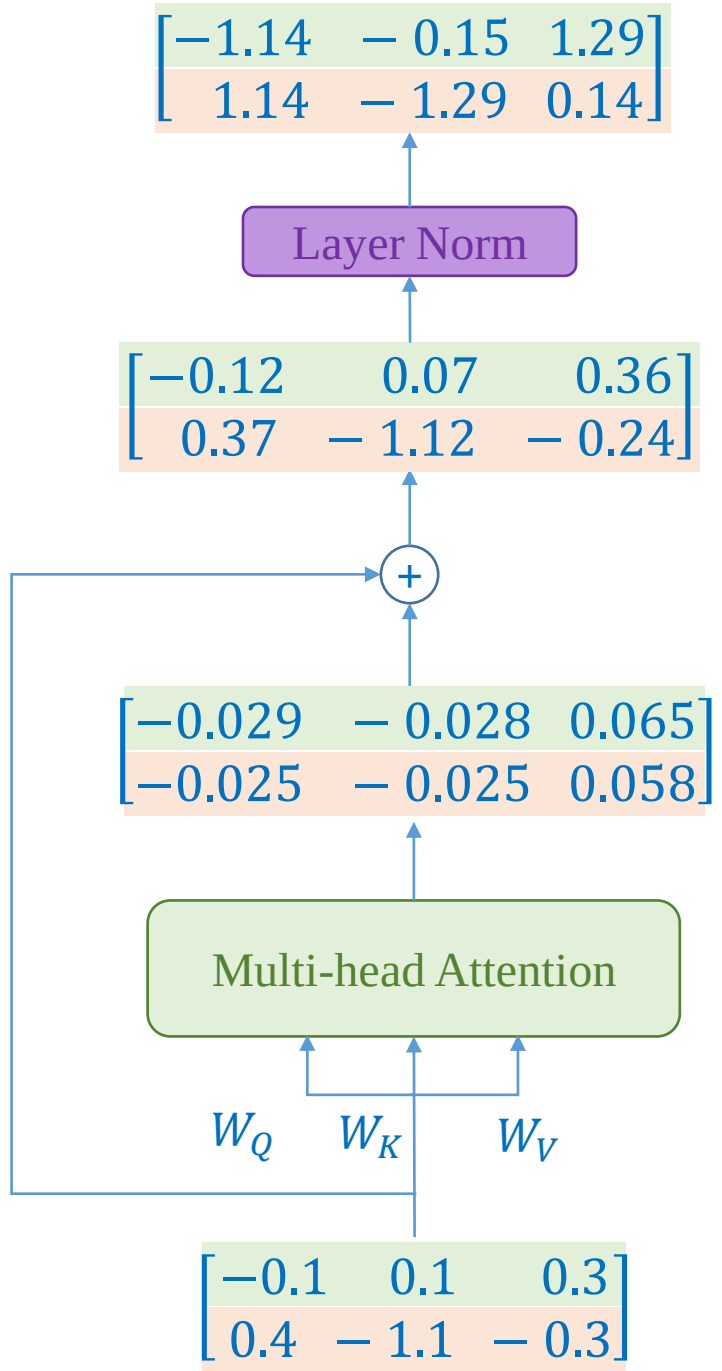


Transformer Block

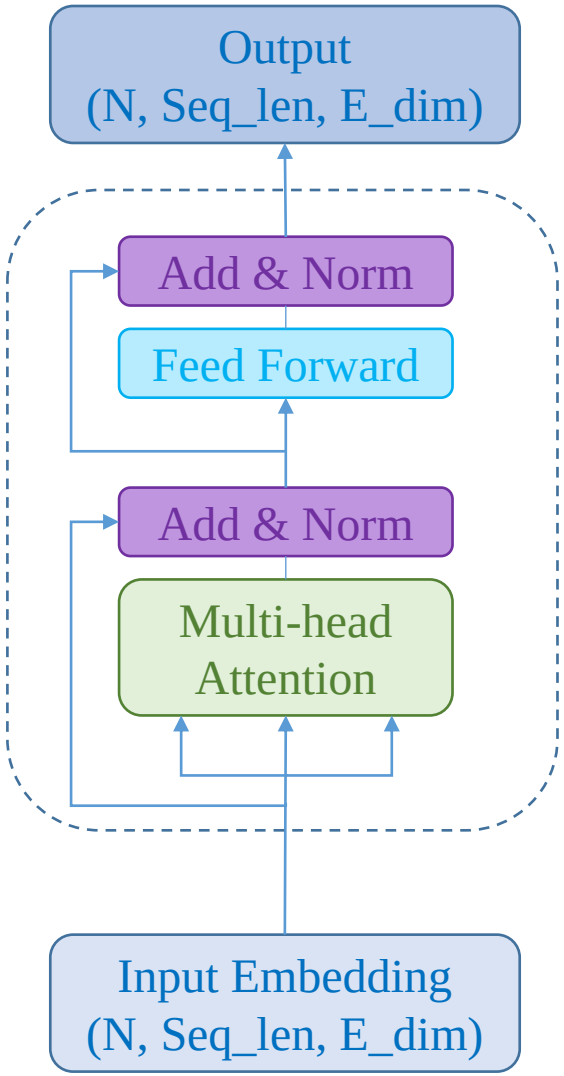




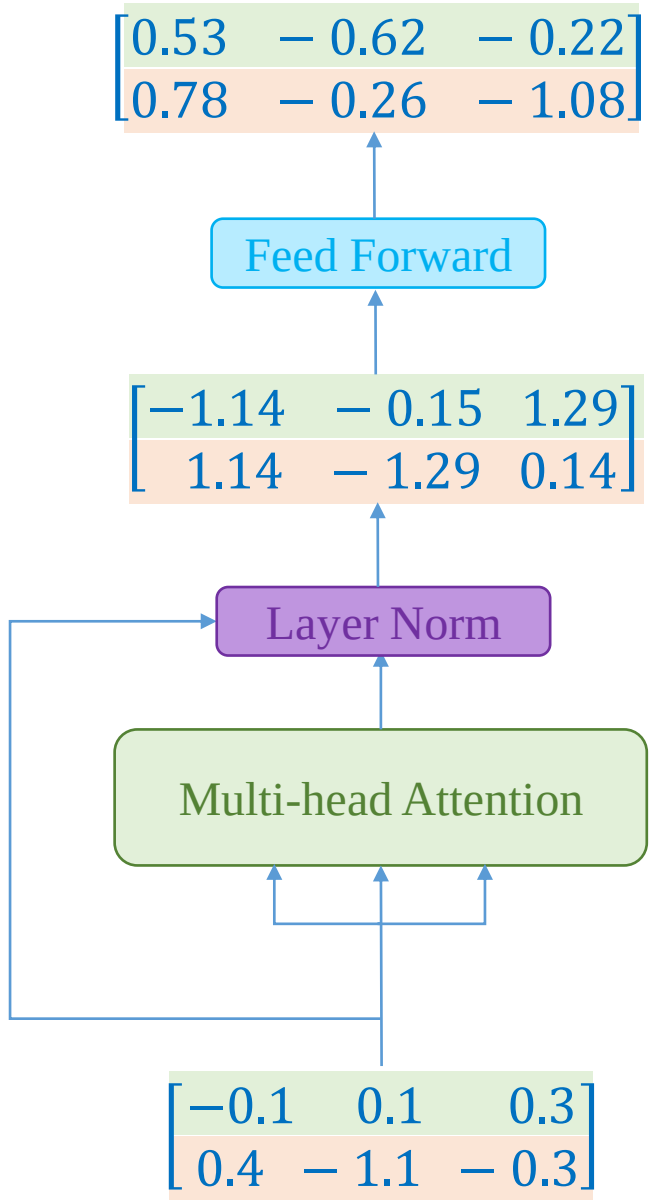
https://openaccess.thecvf.com/content/ICCV2021W/NeurArch/papers/Yao_Leveraging_Batch_Normalization_for_Vision_Transformers_ICCVW_2021_paper.pdf



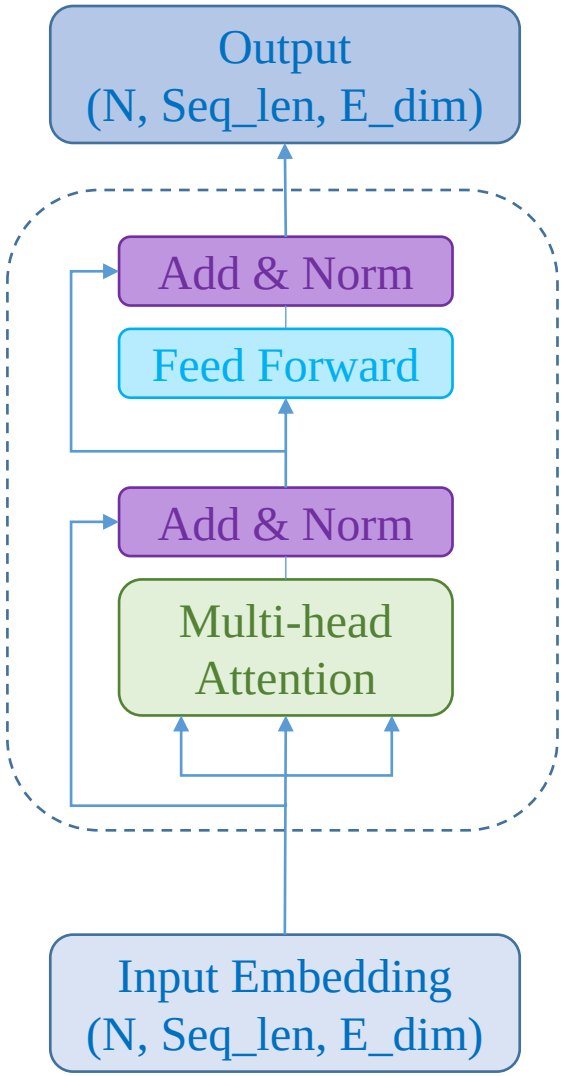
Transformer Block



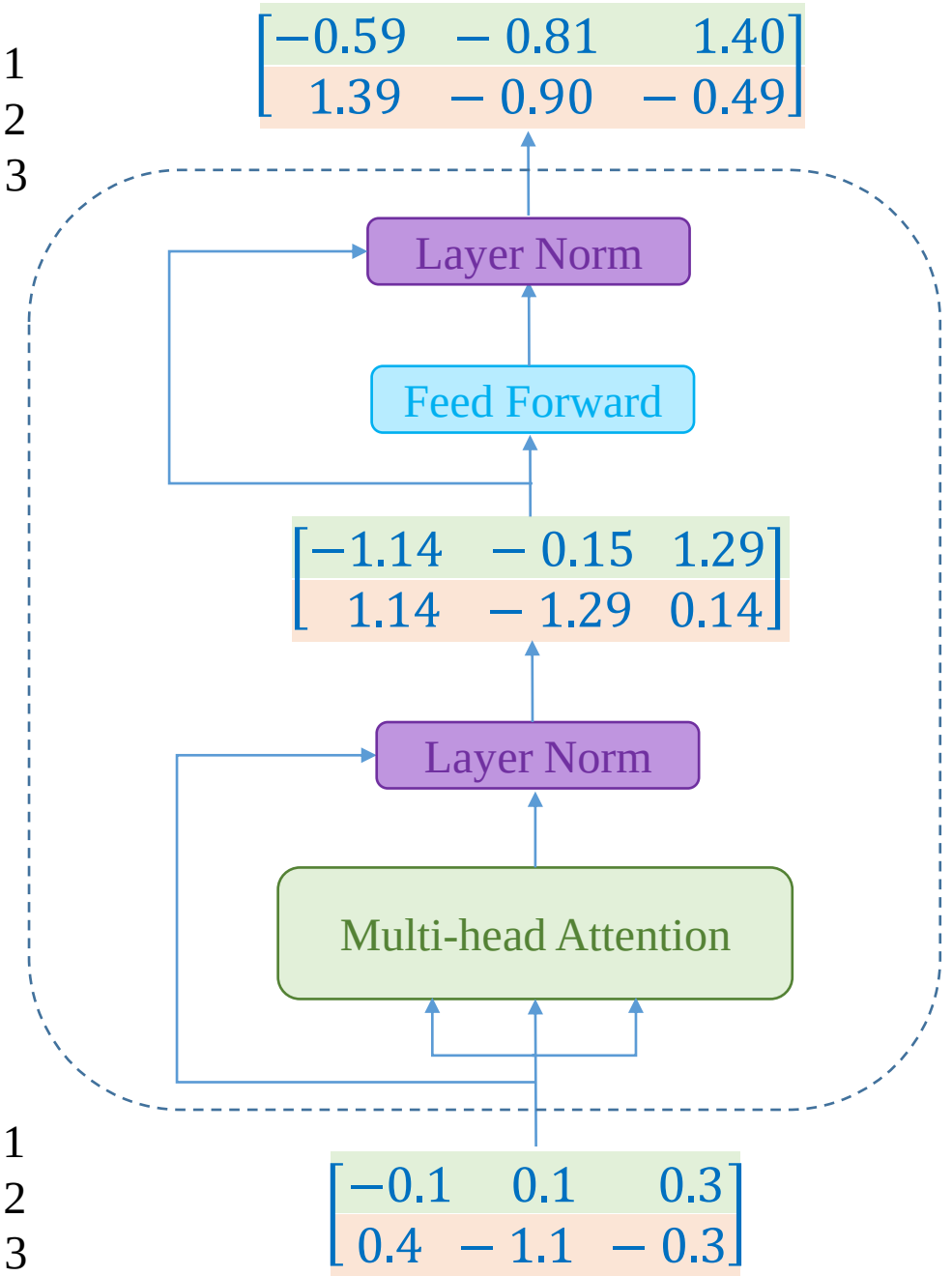
$$W = \begin{bmatrix} 0.15 & 0.39 & -0.34 \\ -0.41 & 0.54 & 0.49 \\ 0.50 & -0.07 & -0.41 \end{bmatrix}$$



Transformer Block

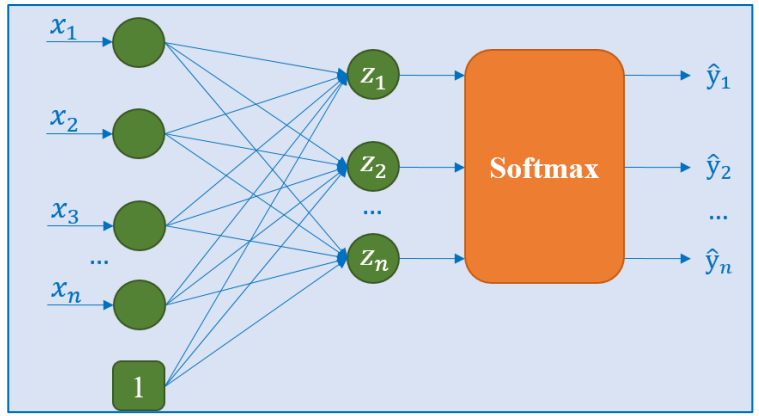


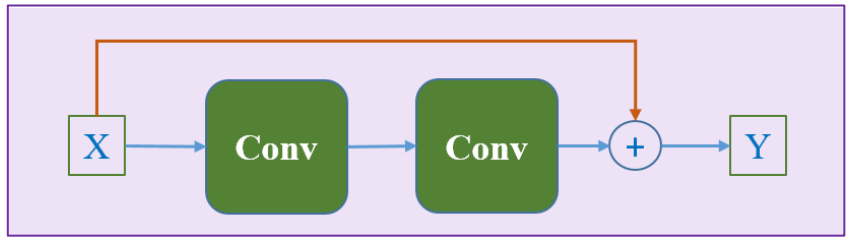
bs = 1
sequence_length = 2
embed_dim = 3

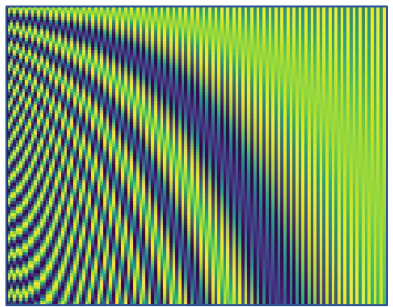


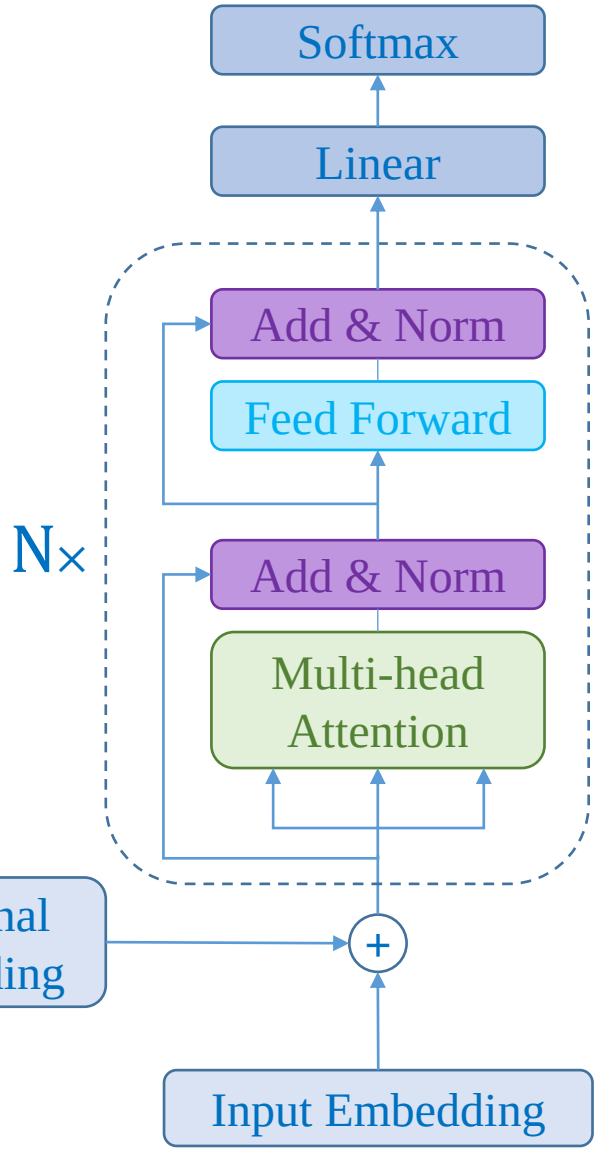
bs = 1
sequence_length = 2
embed_dim = 3

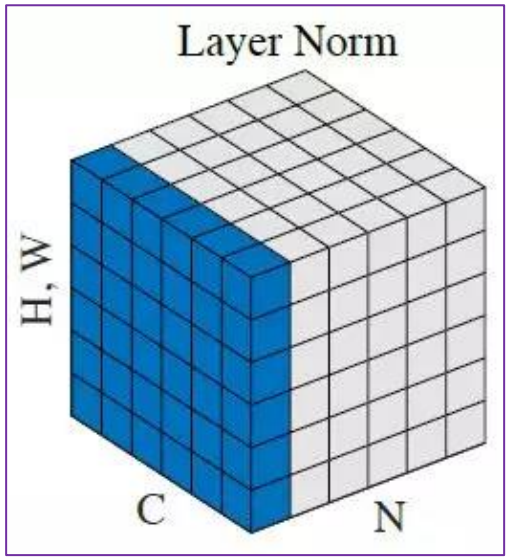
Transformer Models for Text Classification



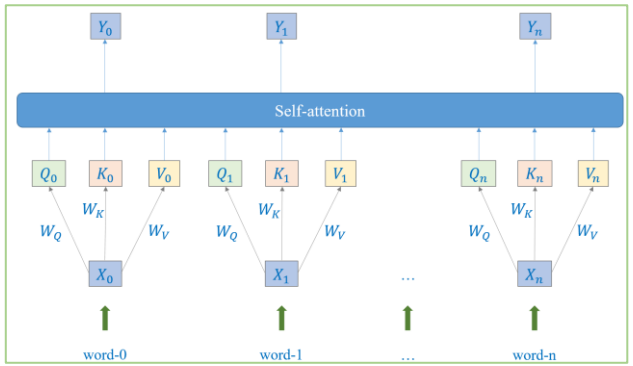


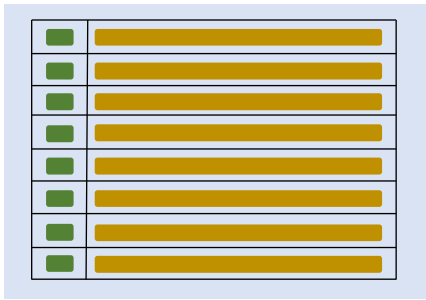






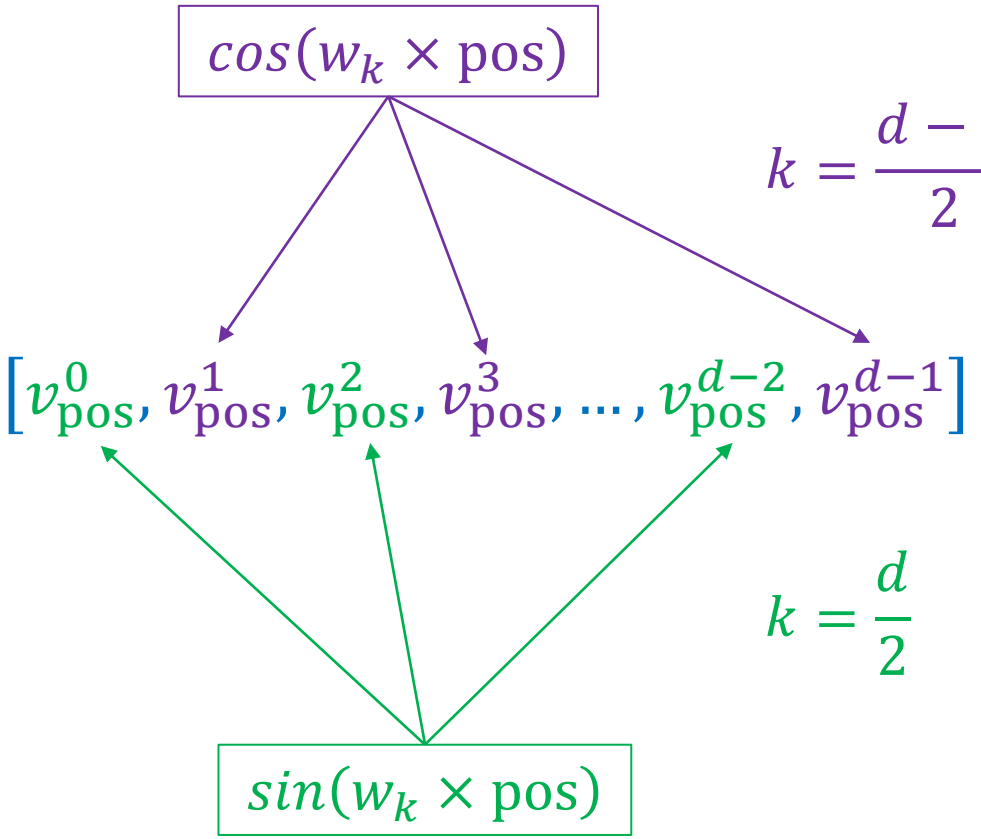
https://arxiv.org/pdf/1803.08494.pdf





Transformer

❖ Positional encoding



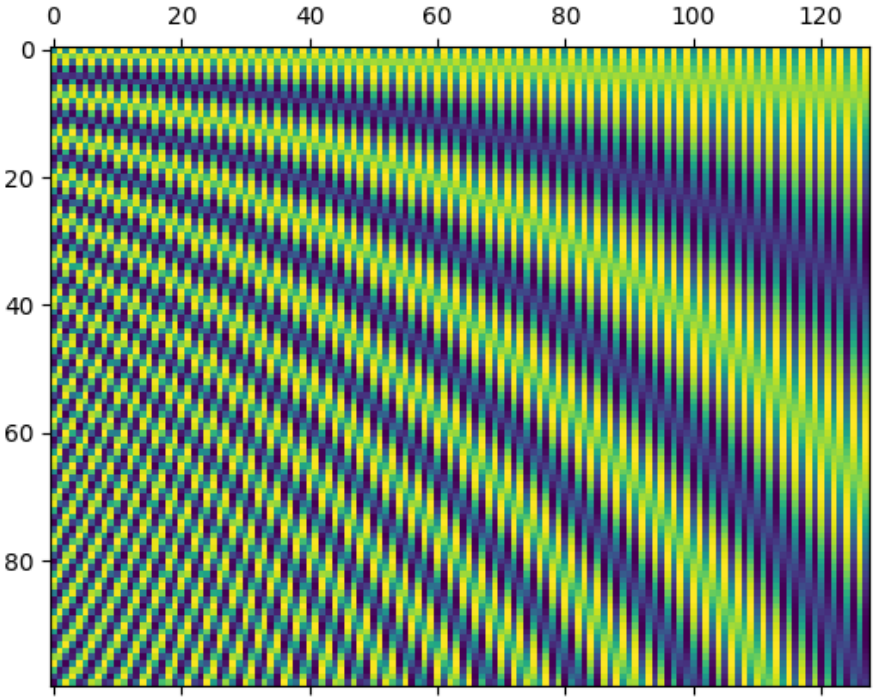
$$T = 10000$$

$$w_k = \frac{1}{T^{2k/d}}$$

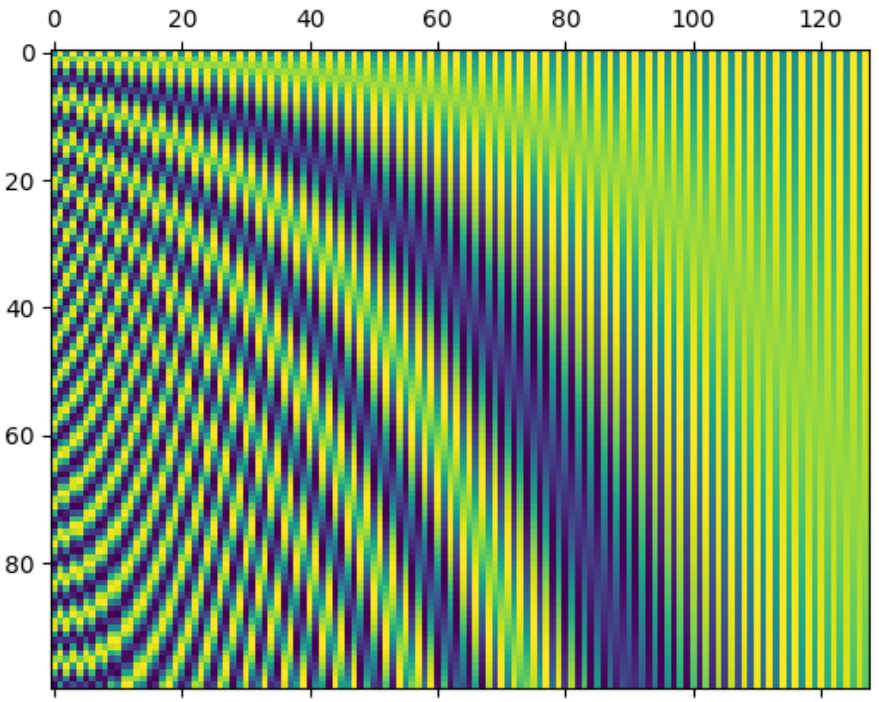
$$\boldsymbol{v}_{\text{pos}} = \begin{bmatrix} sin(w_0 \times \text{pos}) \\ cos(w_0 \times \text{pos}) \\ sin(w_1 \times \text{pos}) \\ cos(w_1 \times \text{pos}) \\ \vdots \\ sin\left(w_{\frac{d}{2}} \times \text{pos}\right) \\ cos\left(w_{\frac{d-1}{2}} \times \text{pos}\right) \end{bmatrix}$$

$0 \leq \text{pos} \leq 99$

$0 \leq \text{d} \leq 127$

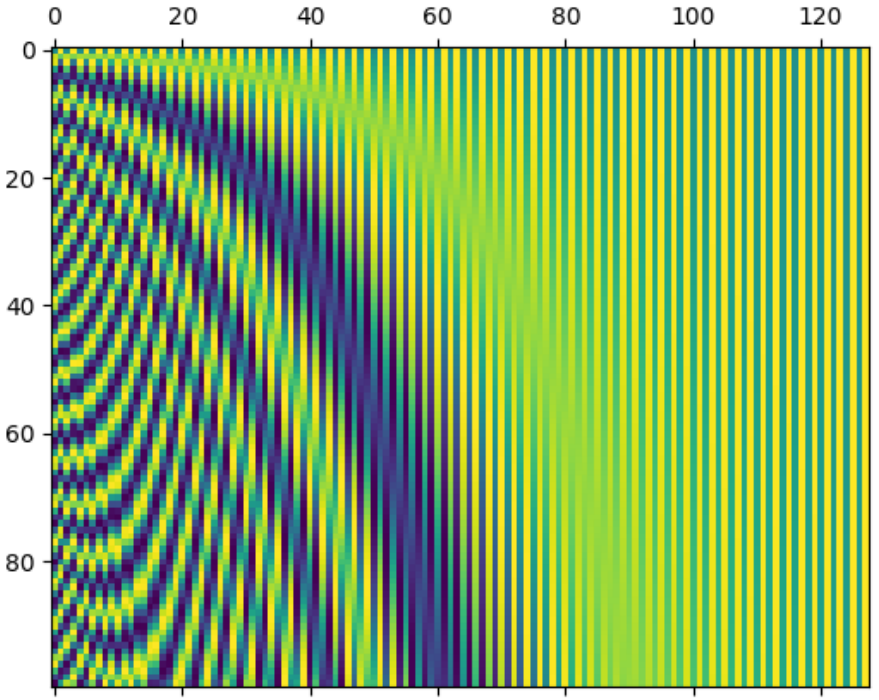


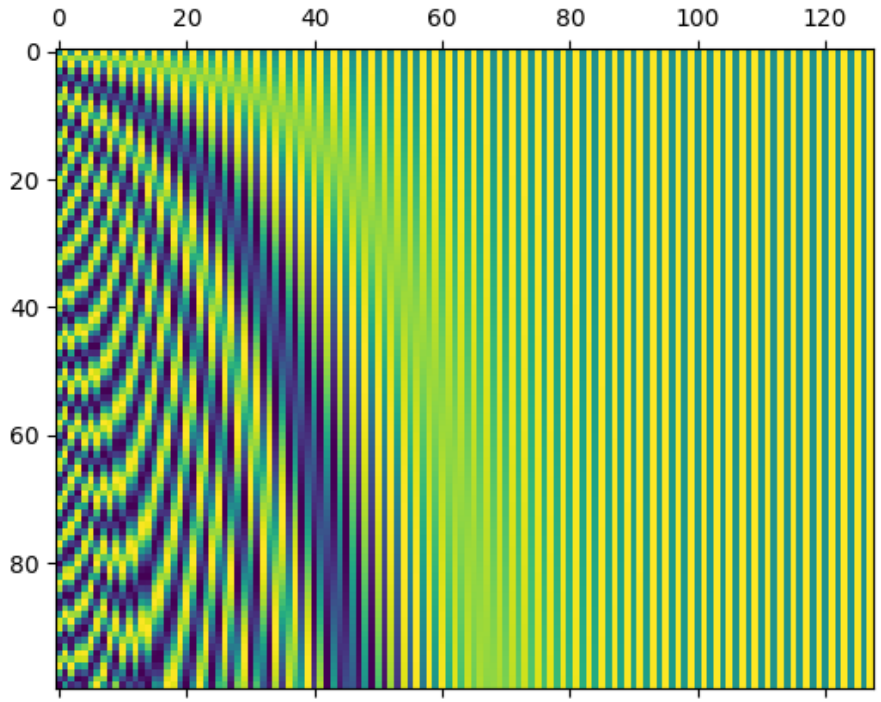
$T = 10$



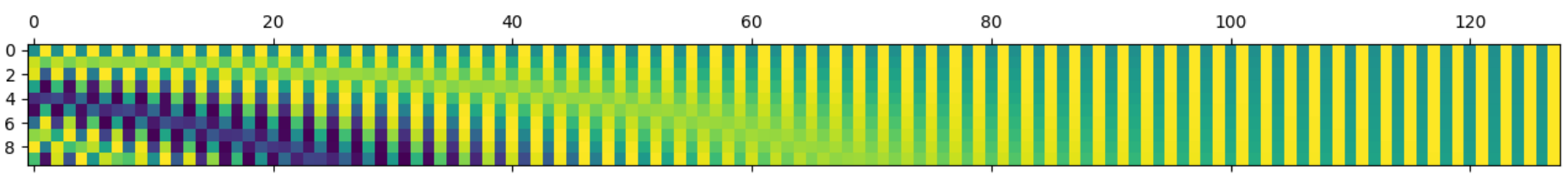
$T = 100$

$T = 1000$





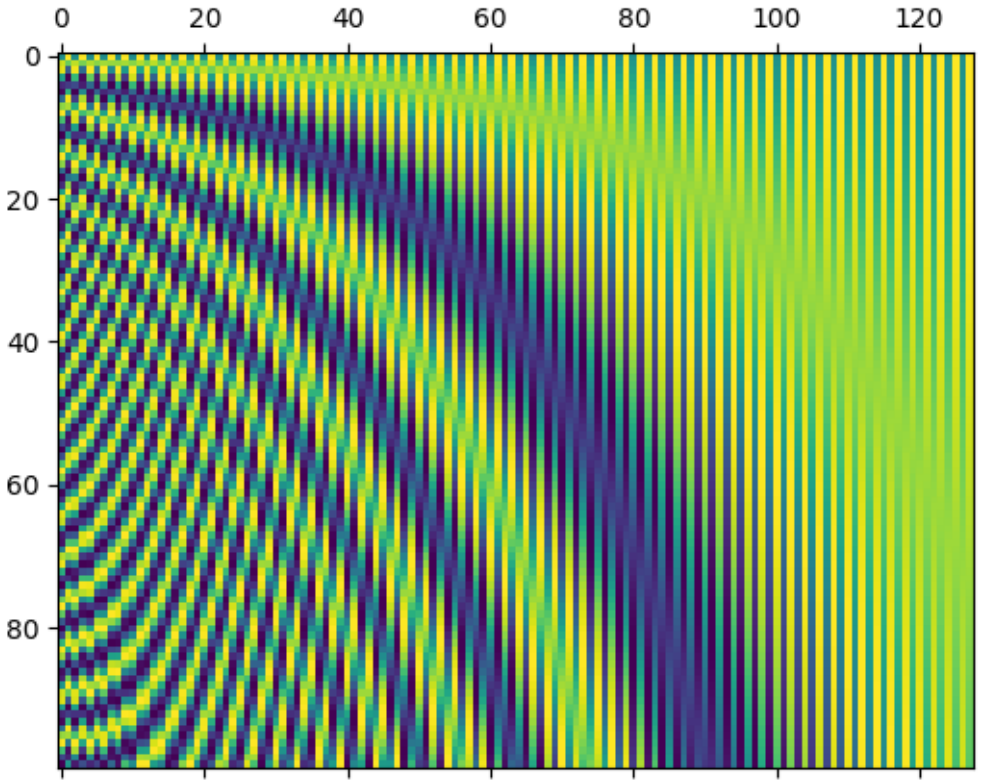
$T = 10000$



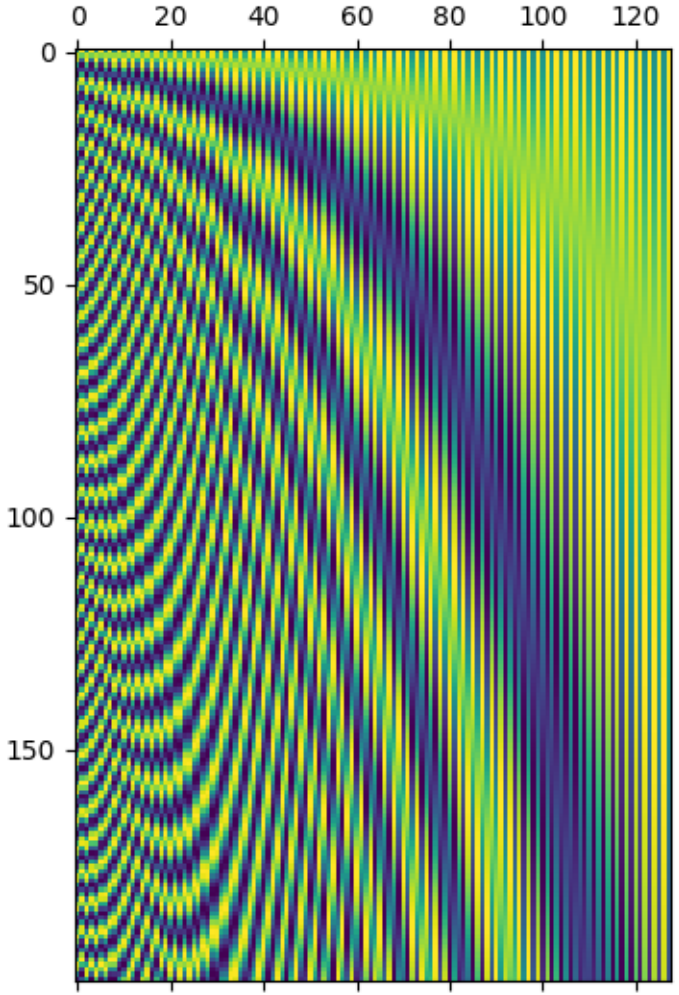
$0 \leq \text{pos} \leq 9$

$0 \leq \text{d} \leq 127$

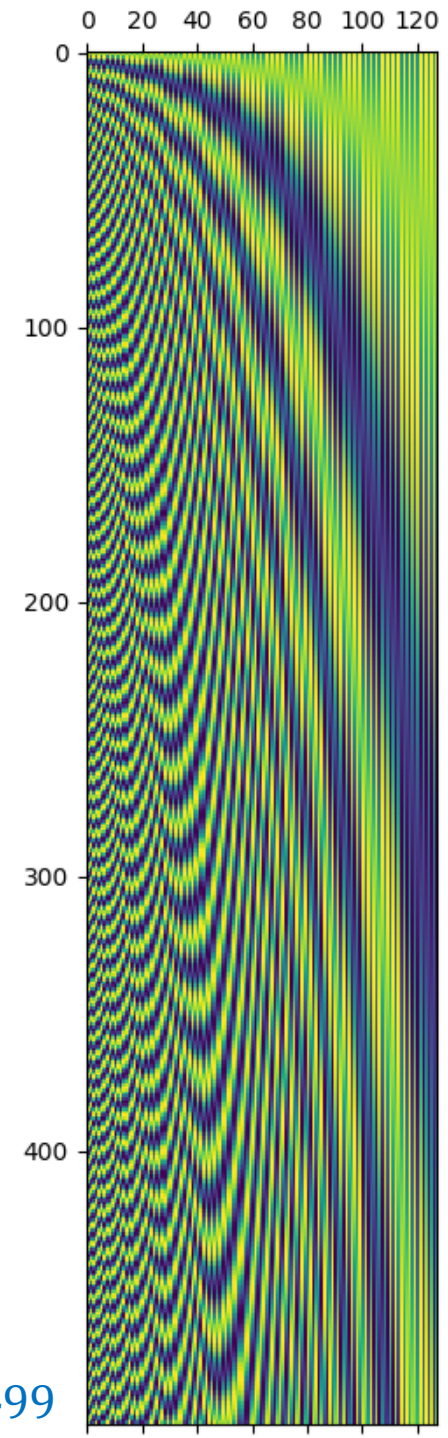
$T = 100$



$0 \leq \text{pos} \leq 99$



$0 \leq \text{pos} \leq 199$



$0 \leq \text{pos} \leq 499$

❖ Positional encoding

$$\boldsymbol{v}_{\text{pos}} = \begin{bmatrix} sin(w_0 \times \text{pos}) \\ cos(w_0 \times \text{pos}) \\ sin(w_1 \times \text{pos}) \\ cos(w_1 \times \text{pos}) \\ \cdots \\ sin\left(w_{\frac{d}{2}} \times \text{pos}\right) \\ cos\left(w_{\frac{d-1}{2}} \times \text{pos}\right) \end{bmatrix}$$

$$w_k = \frac{1}{T^{2k/d}}$$

```python
import numpy as np
import matplotlib.pyplot as plt

def compute_position_encoding(seq_len, d, n):
    result = np.zeros((seq_len, d))
    for k in range(seq_len):
        for i in np.arange(d//2):
            denominator = np.power(n, 2*i/d)
            result[k, 2*i] = np.sin(k/denominator)
            result[k, 2*i+1] = np.cos(k/denominator)
    return result
```

$$\boldsymbol{v} = \begin{bmatrix} \boldsymbol{v}_0 \\ \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \end{bmatrix} = \begin{bmatrix} sin(w_0 \times 0), cos(w_0 \times 0), sin(w_1 \times 0), cos(w_1 \times 0) \\ sin(w_0 \times 1), cos(w_0 \times 1), sin(w_1 \times 1), cos(w_1 \times 1) \\ sin(w_0 \times 2), cos(w_0 \times 2), sin(w_1 \times 2), cos(w_1 \times 2) \end{bmatrix}$$

$$T = 100$$

$$0 \leq \text{pos} \leq 2$$

$$0 \leq \text{d} \leq 3$$

$$= \begin{bmatrix} 0.0 & 1.0 & 0.0 & 1.0 \\ 0.84 & 0.54 & 0.09 & 0.99 \\ 0.91 & -0.41 & 0.19 & 0.98 \end{bmatrix}$$

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Outline

- Layer Normalization
- Transformers Block
- BERT and Text Classification
- Vision Transformer and Image Classification
- For Time-series and Tabular Data

Text Classification

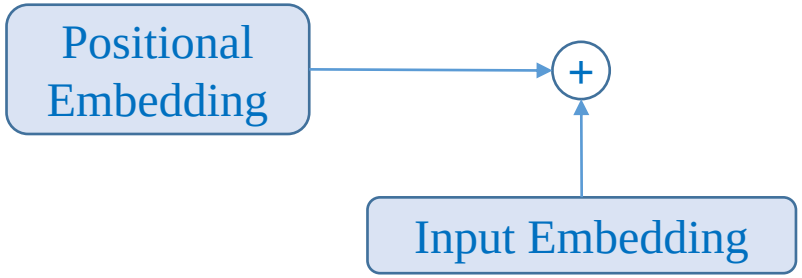
❖ IMDB dataset

- 50,000 movie review for sentiment analysis (data)
- Consist of: + 25,000 movie review for training
  + 25,000 movie review for testing
- Label: positive – negative = 1 – 1

"A wonderful little production. The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire piece …"	positive
"This show was an amazing, fresh & innovative idea in the 70's when it first aired. The first 7 or 8 years were brilliant, but things dropped off after that. By 1990, the show was not really funny anymore, and it's continued its decline further to the complete waste of time it is today…."	negative
"I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a light-hearted comedy. The plot is simplistic, but the dialogue is witty and the characters are likable (even the well bread suspected serial killer)…."	positive
"BTW Carver gets a very annoying sidekick who makes you wanna shoot him the first three minutes he's on screen."	negative

Transformer Models for Text Classification

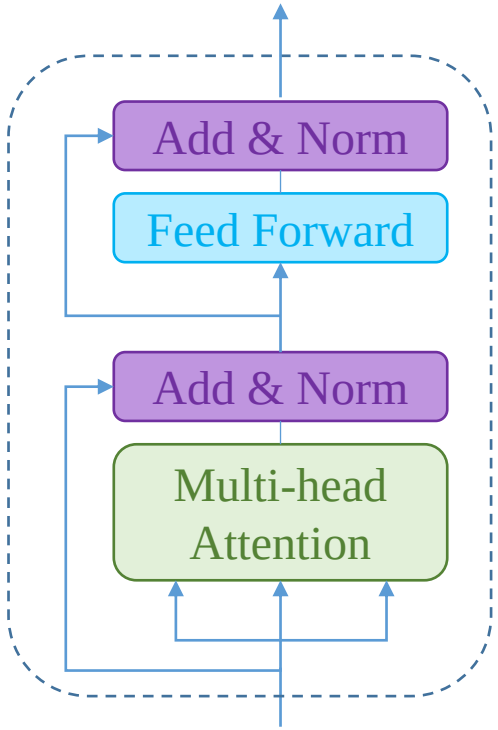
❖ Embedding




```
class TokenAndPositionEmbedding(nn.Module):
    def __init__(self, vocab_size, embed_dim, max_length):
        super().__init__()
        self.word_emb = nn.Embedding(num_embeddings=vocab_size,
                                     embedding_dim=embed_dim)
        self.pos_emb = nn.Embedding(num_embeddings=max_length,
                                    embedding_dim=embed_dim)

    def forward(self, x):
        N, seq_len = x.size()
        positions = torch.arange(0, seq_len).expand(N, seq_len)
        output1 = self.word_emb(x)
        output2 = self.pos_emb(positions)
        output = output1 + output2
        return output
```

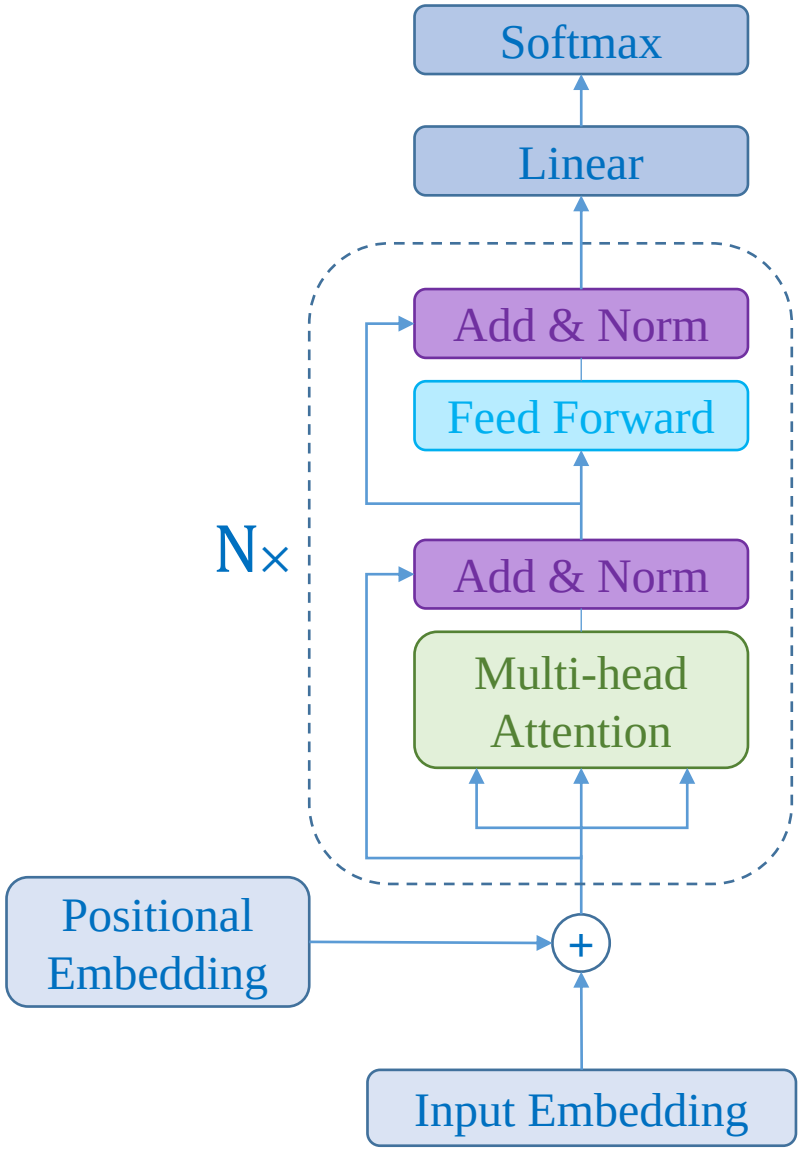
Transformer Models for Text Classification

❖ Transformer block




```python
class TransformerBlock(nn.Module):
    def __init__(self, embed_dim, num_heads, ff_dim, dropout):
        super().__init__()
        self.attn = nn.MultiheadAttention(embed_dim=embed_dim,
                                          num_heads=num_heads)
        self.ffn = nn.Linear(in_features=embed_dim,
                             out_features=ff_dim)
        self.layernorm_1 = nn.LayerNorm(normalized_shape=embed_dim)
        self.layernorm_2 = nn.LayerNorm(normalized_shape=embed_dim)

    def forward(self, query, key, value):
        attn_output, _ = self.attn(query, key, value)
        out_1 = self.layernorm_1(query + attn_output)
        ffn_output = self.ffn(out_1)
        out_2 = self.layernorm_2(out_1 + ffn_output)
        return out_2
```

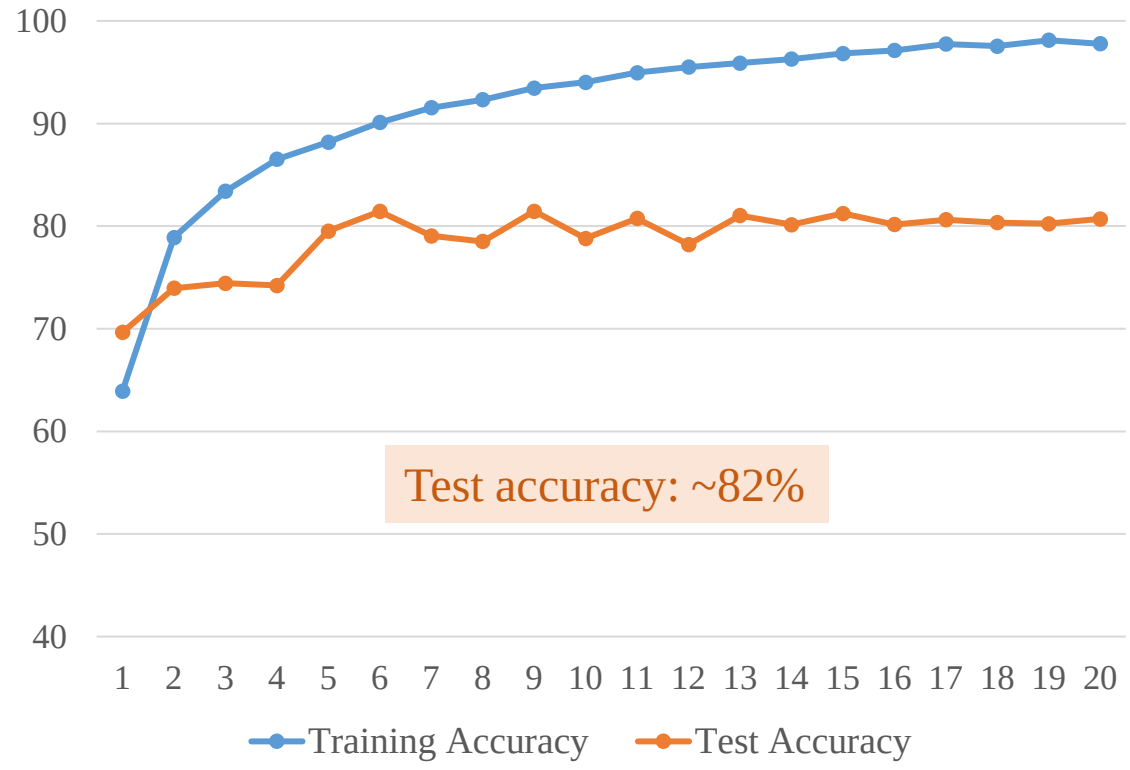
Transformer Models for Text Classification




```python
class TransformerTextCls(nn.Module):
    def __init__(self, vocab_size,
                 max_length, embed_dim,
                 num_heads, ff_dim,
                 dropout, device):
        super().__init__()
        self.embd_layer = TokenAndPositionEmbedding(vocab_size,
                                                    embed_dim,
                                                    max_length)
        self.transformer_layer = TransformerBlock(embed_dim,
                                                  num_heads,
                                                  ff_dim)
        self.pooling = nn.AvgPool1d(kernel_size=max_length)
        self.fc = nn.Linear(in_features=embed_dim,
                            out_features=2)
        self.relu = nn.ReLU()

    def forward(self, x):
        output = self.embd_layer(x)
        output = self.transformer_layer(output, output, output)
        output = self.pooling(output.permute(0,2,1)).squeeze()
        output = self.fc(output)

        return output
```

Transformer Models for Text Classification

❖ Results



Test accuracy: ~82%

Train Transformer from Scratch

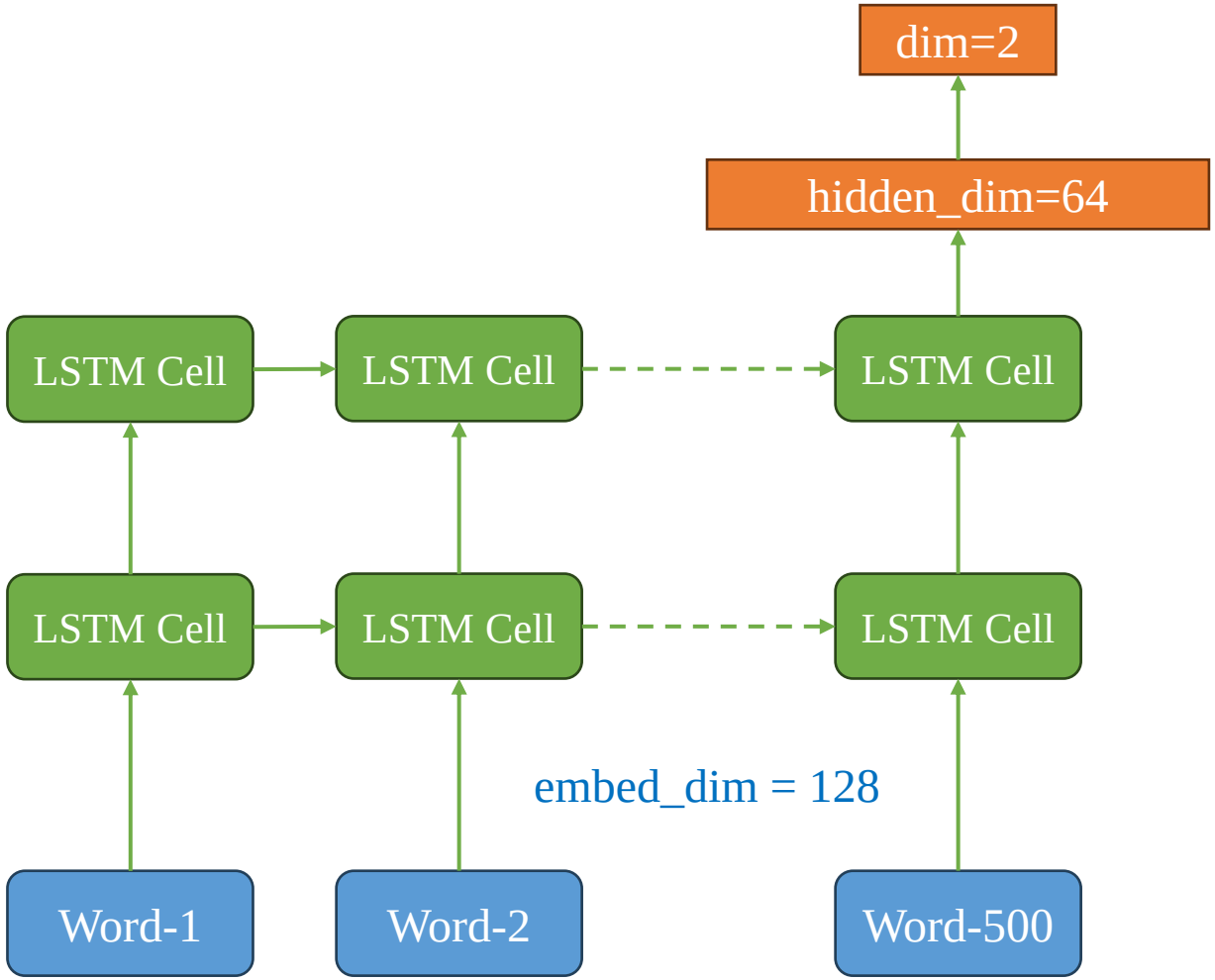
Epoch	Training Loss	Validation Loss	Accuracy
1	0.282000	0.216251	0.915120
2	0.149400	0.200550	0.929160
3	0.102600	0.250929	0.924840
4	0.055700	0.321075	0.930280
5	0.038100	0.344045	0.928200
6	0.021000	0.375231	0.928480
7	0.020200	0.384939	0.928680
8	0.010400	0.413715	0.930320
9	0.009400	0.428601	0.930840
10	0.005800	0.441084	0.930840

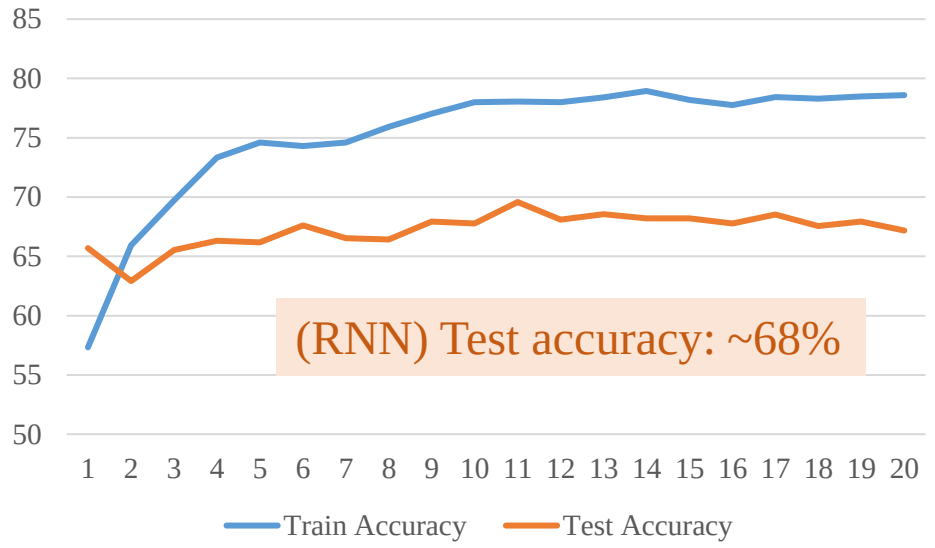
transfer learning

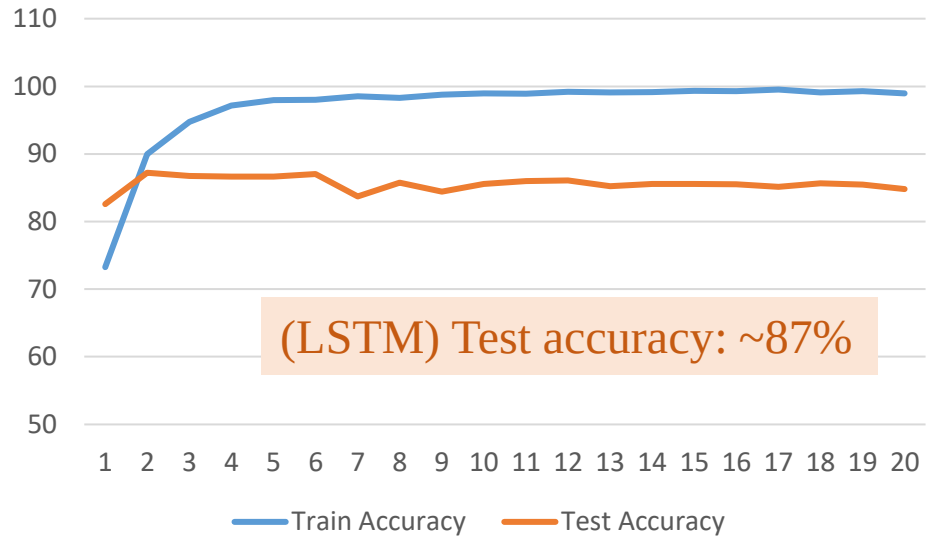
```
{'eval_loss': 0.20054994523525238,
 'eval_accuracy': 0.92916,
 'eval_runtime': 253.348,
 'eval_samples_per_second': 98.678,
 'eval_steps_per_second': 3.087,
 'epoch': 10.0}
```

Text Deep Models

Long short-term memory







Text Classification

❖ Tweets dataset

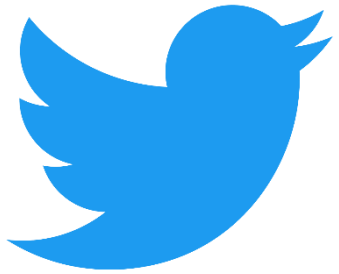
- Training samples: 7613
- Total Number of disaster tweets: 4342
- Total Number of non-disaster tweets: 3271

id	keyword	location	text	target
48	ablaze	Birmingham	@bbcmtd Wholesale Markets ablaze http://t.co/lH	1
49	ablaze	Est. September 2012 - Bri	We always try to bring the heavy. #metal #RT htt	0
144	accident	UK	.@NorwayMFA #Bahrain police had previously c	1
145	accident	Nairobi, Kenya	I still have not heard Church Leaders of Kenya co	0
146	aftershock	Instagram - @heyimginog	@afterShock_DeLo scuf ps live and the game... c	0
1669	bombing	London	Japan marks 70th anniversary of Hiroshima atomi	1
1670	bombing	United States	Japan marks 70th anniversary of Hiroshima atomi	1
6536	injury	beijing .China	Rory McIlroy to Test Ankle Injury in Weekend P	0
6537	injury	MISSING	CLEARED:incident with injury:I-495 inner loop	1

AI VIETNAM
All-in-One Course

Text Classification

❖ Tweets dataset



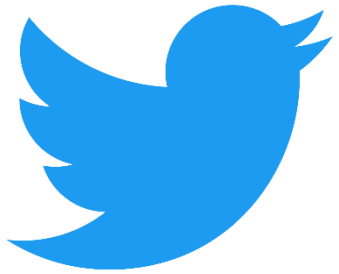
id	keyword	location	text	target
48	ablaze	Birmingham	@bbcmtd Wholesale Markets ablaze http://t.co/lH	1
49	ablaze	Est. September 2012 - Bri	We always try to bring the heavy. #metal #RT htt	0
144	accident	UK	.@NorwayMFA #Bahrain police had previously	1
145	accident	Nairobi, Kenya	I still have not heard Church Leaders of Kenya co	0
146	aftershock	Instagram - @heyimginog	@afterShock_DeLo scuf ps live and the game... c	0
1669	bombing	London	Japan marks 70th anniversary of Hiroshima atomi	1
1670	bombing	United States	Japan marks 70th anniversary of Hiroshima atomi	1
6536	injury	beijing .China	Rory McIlroy to Test Ankle Injury in Weekend P	0
6537	injury	MISSING	CLEARED:incident with injury:I-495 inner loop	1

```python
import pandas as pd
import numpy as np

df = pd.read_csv('train.csv')
df_shuffled = df.sample(frac=1,
                        random_state=42)

df_shuffled.drop(["id", "keyword", "location"],
                 axis=1, inplace=True)
df_shuffled.reset_index(inplace=True,
                        drop=True)
```

text	target
@bbcmtd Wholesale Markets ablaze http://t.co/lH	1
We always try to bring the heavy. #metal #RT htt	0
.@NorwayMFA #Bahrain police had previously	1
I still have not heard Church Leaders of Kenya co	0
@afterShock_DeLo scuf ps live and the game... c	0
Japan marks 70th anniversary of Hiroshima atomi	1
Japan marks 70th anniversary of Hiroshima atomi	1
Rory McIlroy to Test Ankle Injury in Weekend P	0
CLEARED:incident with injury:I-495 inner loop	1

Text Classification

❖ Tweets dataset

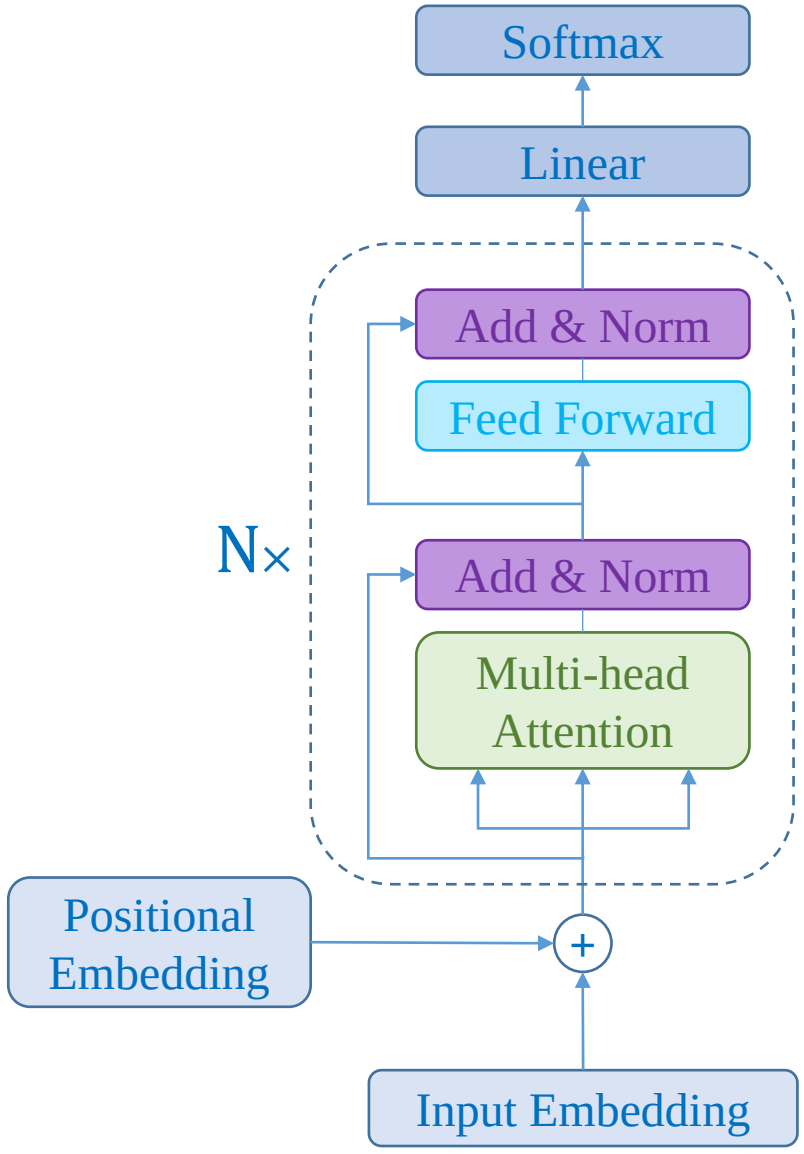


id	keyword	location	text	target
48	ablaze	Birmingham	@bbcmtd Wholesale Markets ablaze http://t.co/lH	1
49	ablaze	Est. September 2012 - Bri	We always try to bring the heavy. #metal #RT htt	0
144	accident	UK	.@NorwayMFA #Bahrain police had previously	1
145	accident	Nairobi, Kenya	I still have not heard Church Leaders of Kenya co	0
146	aftershock	Instagram - @heyimginog	@afterShock_DeLo scuf ps live and the game... c	0
1669	bombing	London	Japan marks 70th anniversary of Hiroshima atomi	1
1670	bombing	United States	Japan marks 70th anniversary of Hiroshima atomi	1
6536	injury	beijing .China	Rory McIlroy to Test Ankle Injury in Weekend P	0
6537	injury	MISSING	CLEARED:incident with injury:I-495 inner loop	1

```
test_df = df_shuffled.sample(frac=0.1, random_state=42)
train_df = df_shuffled.drop(test_df.index)

train_df = train_df.to_numpy()
test_df  = test_df.to_numpy()

#...

train_loader = DataLoader(train_data,
                          batch_size=32, shuffle=True)
test_loader = DataLoader(test_data,
                         batch_size=32, shuffle=False)
```

text	target
@bbcmtd Wholesale Markets ablaze http://t.co/lH	1
We always try to bring the heavy. #metal #RT htt	0
.@NorwayMFA #Bahrain police had previously	1
I still have not heard Church Leaders of Kenya co	0
@afterShock_DeLo scuf ps live and the game... c	0
Japan marks 70th anniversary of Hiroshima atomi	1
Japan marks 70th anniversary of Hiroshima atomi	1
Rory McIlroy to Test Ankle Injury in Weekend P	0
CLEARED:incident with injury:I-495 inner loop	1

Transformer Models for Text Classification

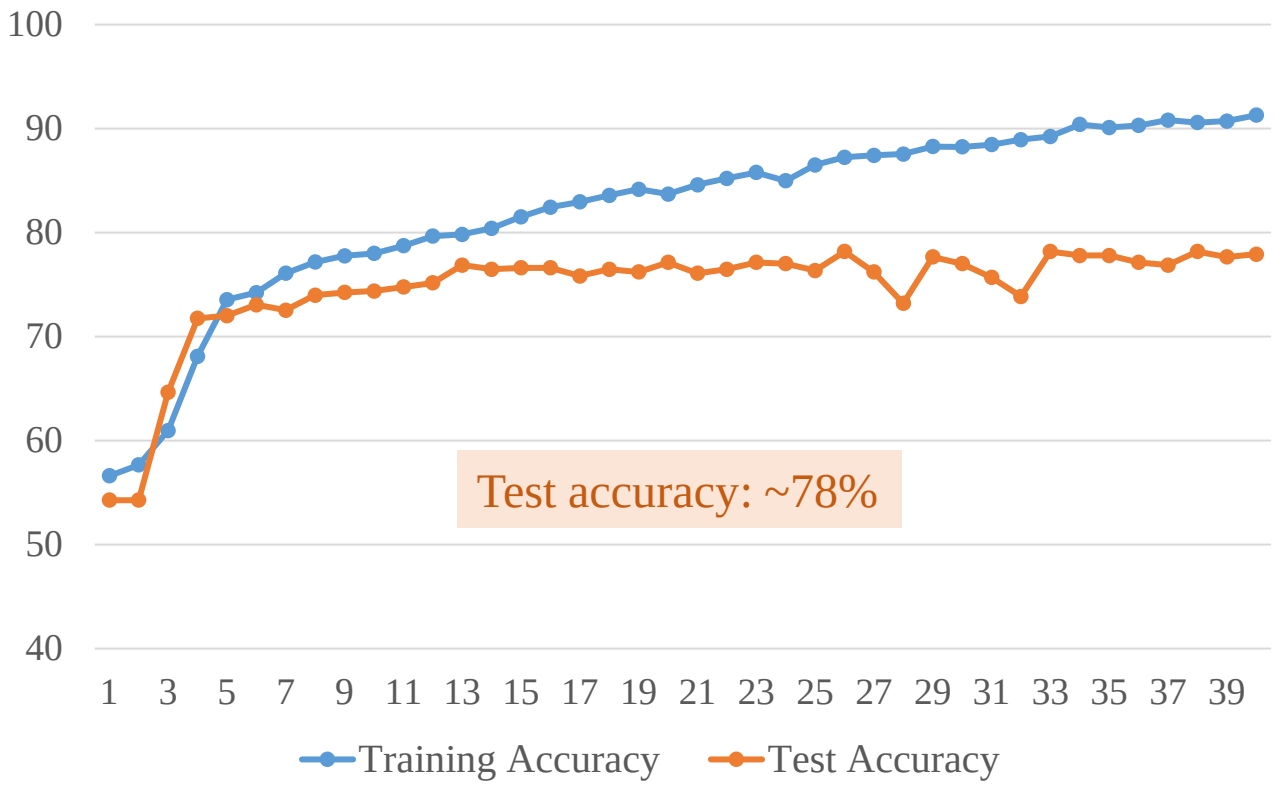



```python
class TransformerBlock(nn.Module):
    def __init__(self, embed_dim, num_heads, ff_dim):
        # ...

    def forward(self, query, key, value):
        # ...

class TokenAndPositionEmbedding(nn.Module):
    def __init__(self, vocab_size, embed_dim, max_length):
        # ...

    def forward(self, x):
        # ...

class TransformerTextCls(nn.Module):
    def __init__(self, vocab_size, max_length,
                 embed_dim, num_heads, ff_dim):
        # ...

    def forward(self, x):
        output = self.embd_layer(x)
        output = self.transformer_layer(output, output, output)
        output = self.pooling(output.permute(0,2,1)).squeeze()
        output = self.fc(output)
        return output
```

AI VIETNAM
All-in-One Course

Text Classification

❖ Results



Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.410340	0.819974
2	0.448100	0.419809	0.830486
3	0.320300	0.452259	0.822602
4	0.234100	0.545349	0.818660
5	0.176700	0.664652	0.809461
14	0.035800	1.288030	0.808147
15	0.035800	1.281955	0.825230
16	0.029600	1.302865	0.819974
17	0.025800	1.324648	0.816032
18	0.022500	1.339616	0.819974
19	0.023800	1.363005	0.818660
20	0.014200	1.395317	0.817346

717

```
{'eval_loss': 0.41033995151519775,
 'eval_accuracy': 0.8199737187910644,
 'eval_runtime': 2.696,
 'eval_samples_per_second': 282.267,
 'eval_steps_per_second': 17.804,
 'epoch': 20.0}
```

Train Transformer from Scratch

Bidirectional Encoder Representations from Transformers

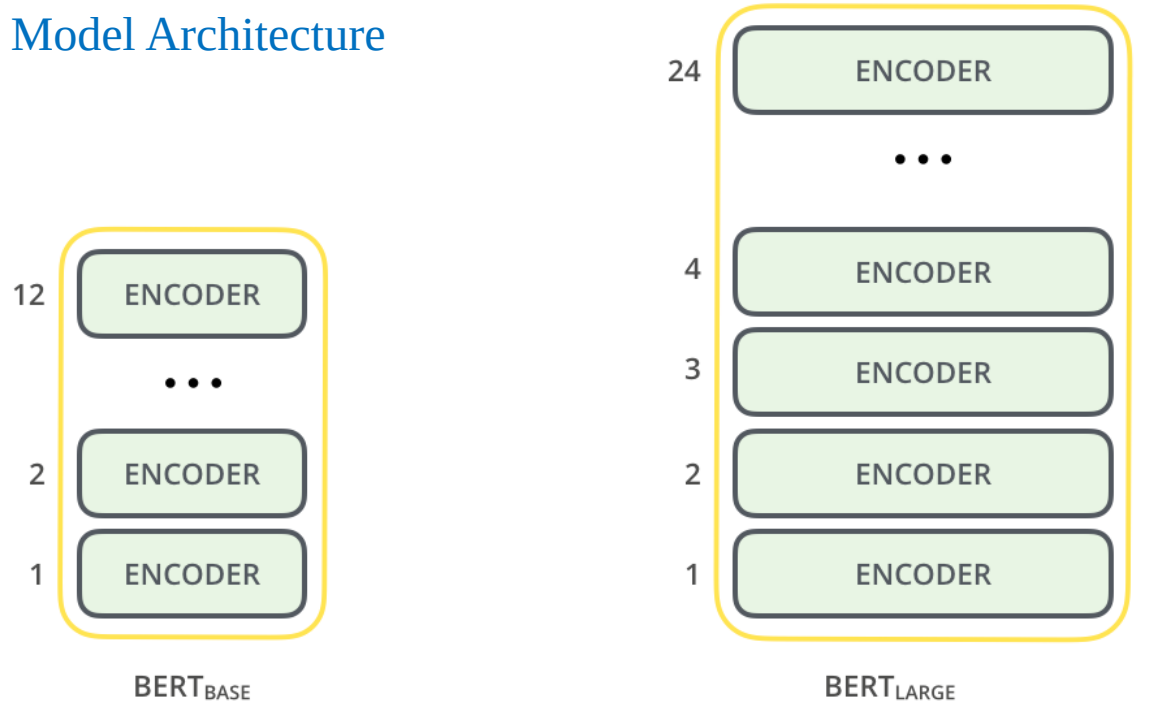
Trained on Wikipedia (~2.5B words) and Google's BooksCorpus (~800M words)

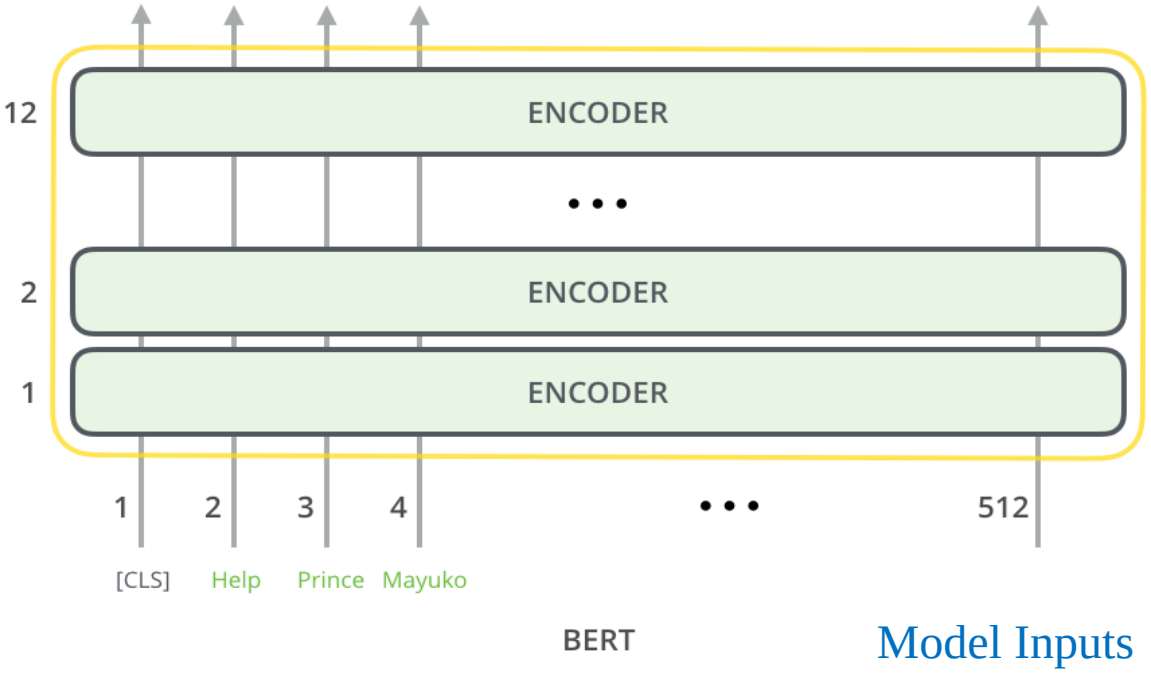
64 TPUs trained BERT over the course of 4 days

DistilBERT offers a lighter version of BERT; runs 60% faster while maintaining over 95% of BERT's performance.

https://arxiv.org/pdf/1810.04805.pdf

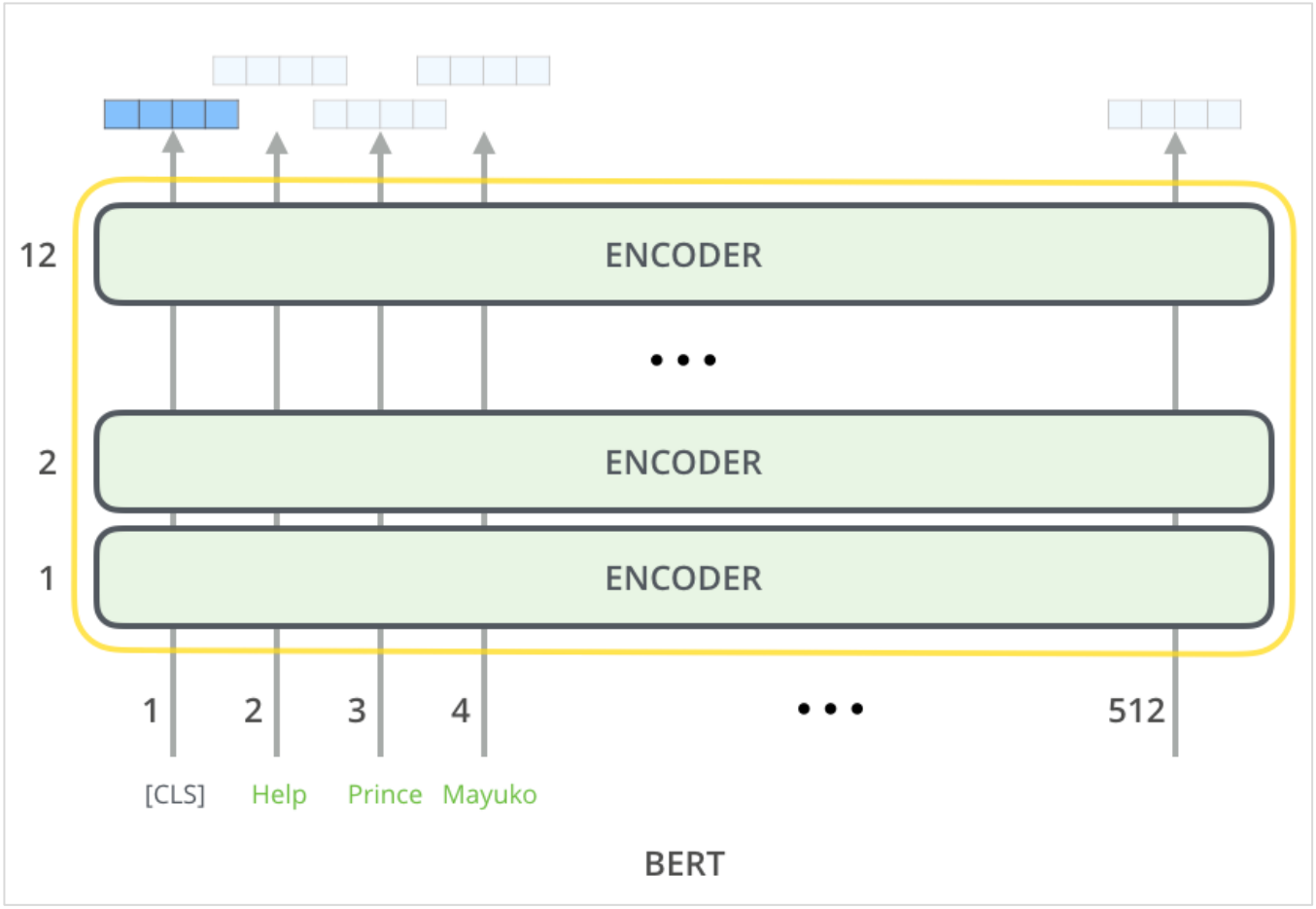
Model Architecture

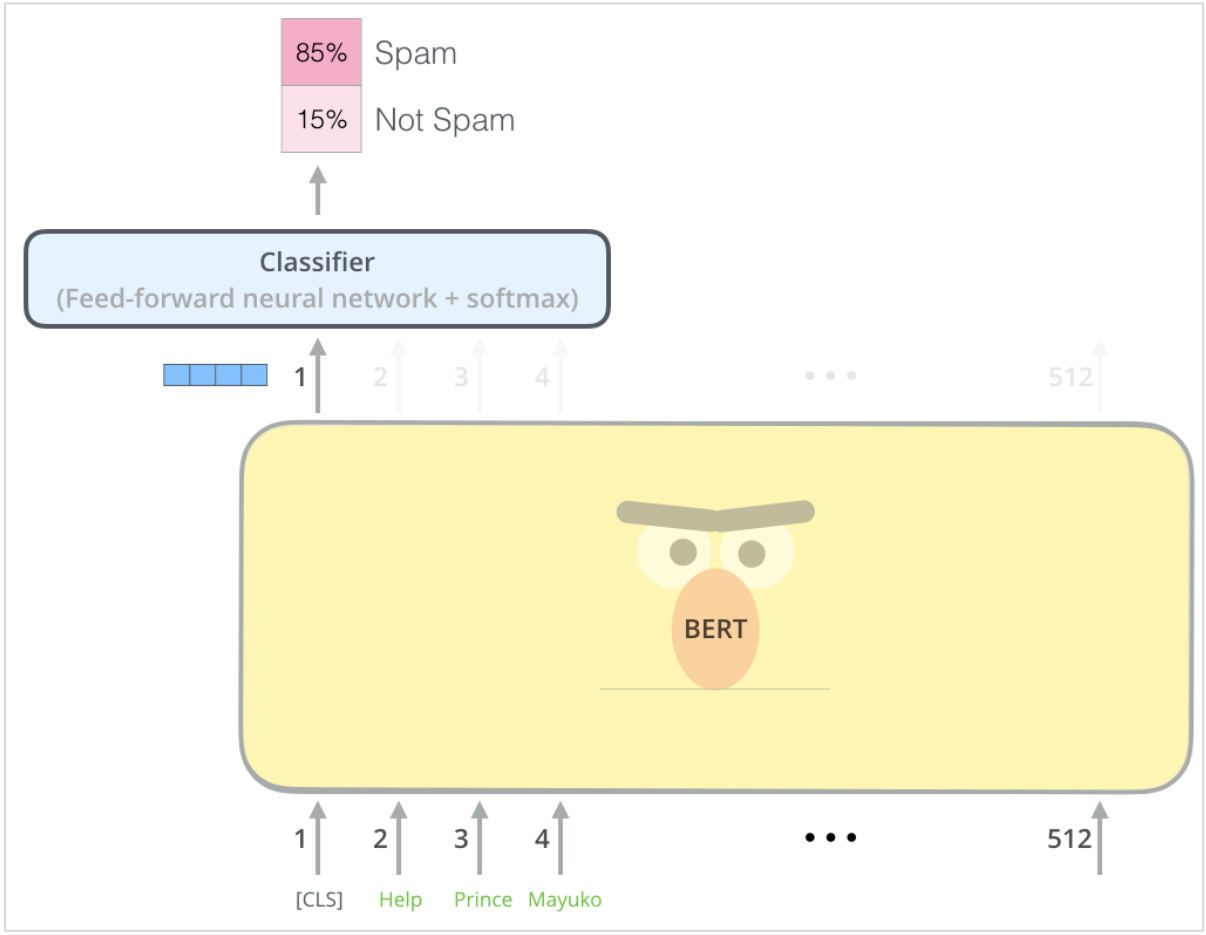




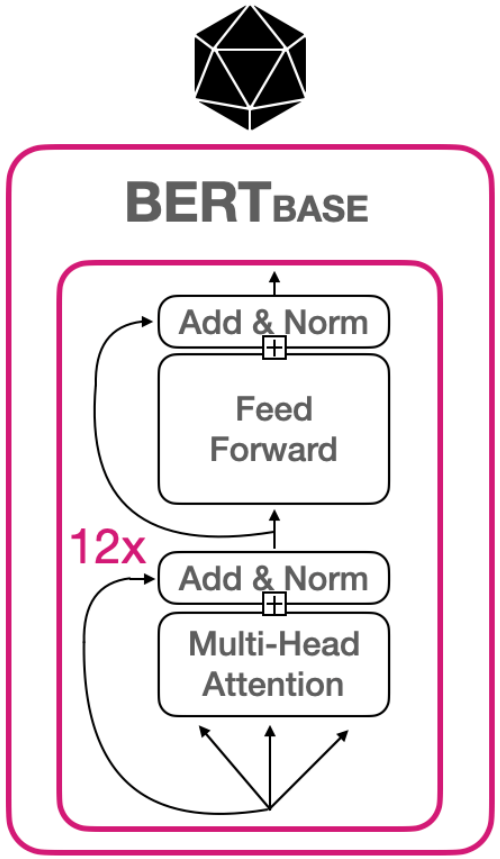
Model Inputs

Bidirectional Encoder Representations from Transformers

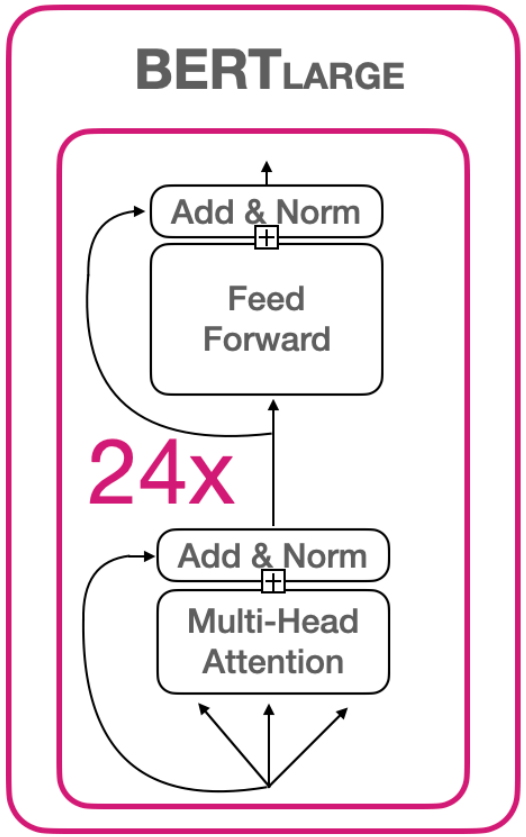




BERT Size & Architecture



110M Parameters



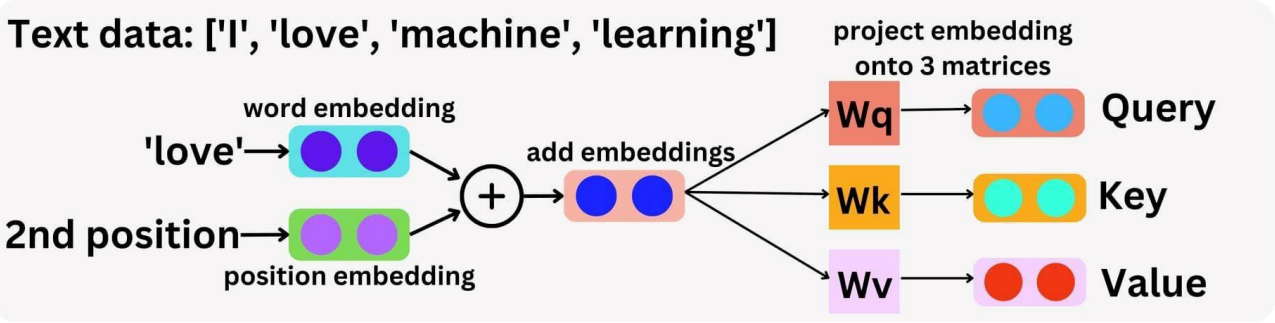
340M Parameters

https://huggingface.co/blog/bert-101

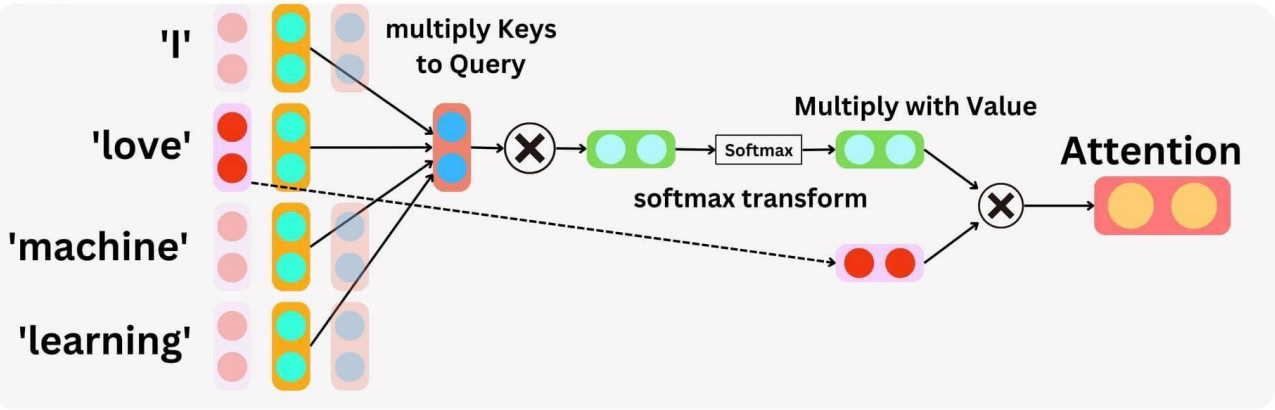
Transformers: Attention is all you need!

Step 1: Create the Query, Key, Value

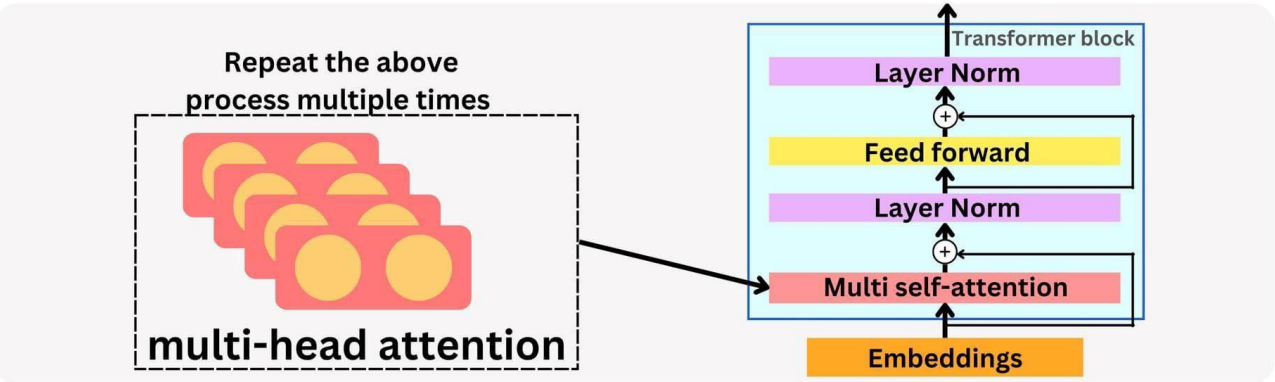
TheAiEdge.io



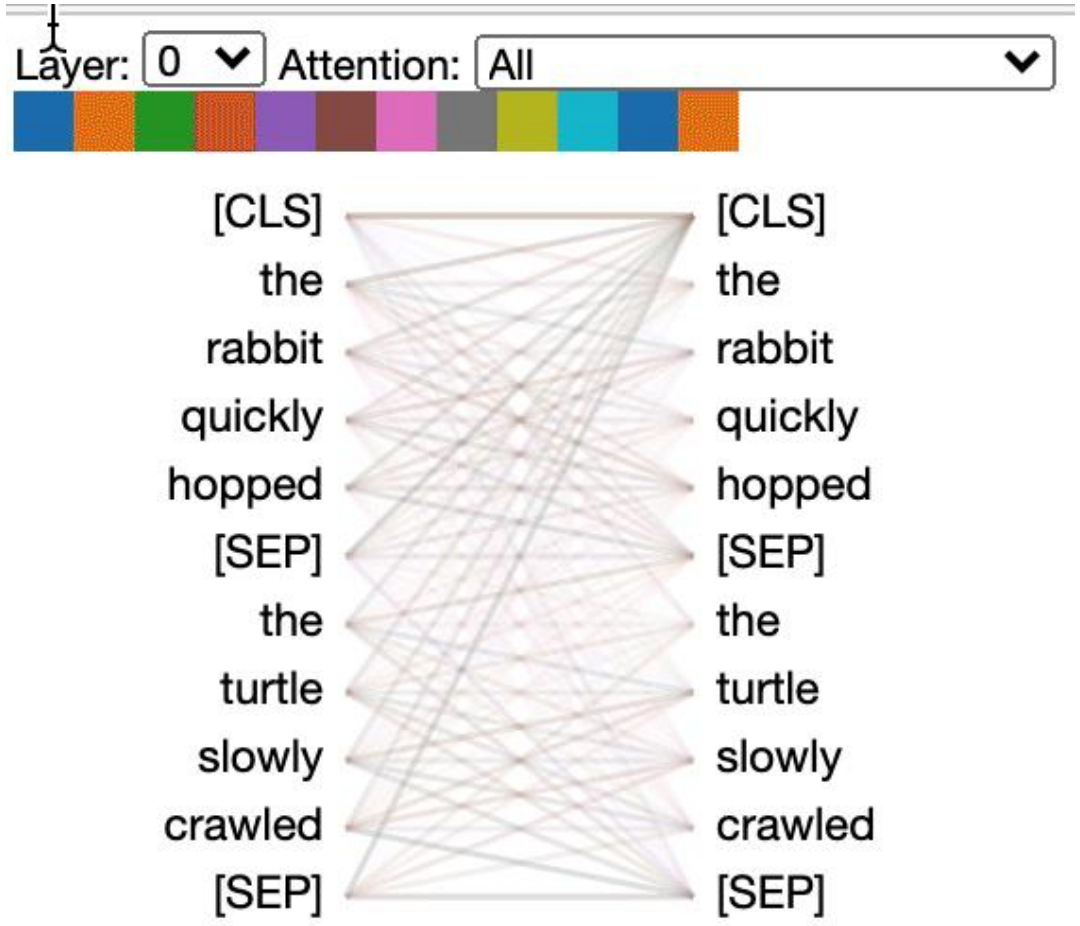
Step 2: Create the Attention



Step 3: Duplicate Attention and include inTransformer



Attention Visualization



https://github.com/jessevig/bertviz

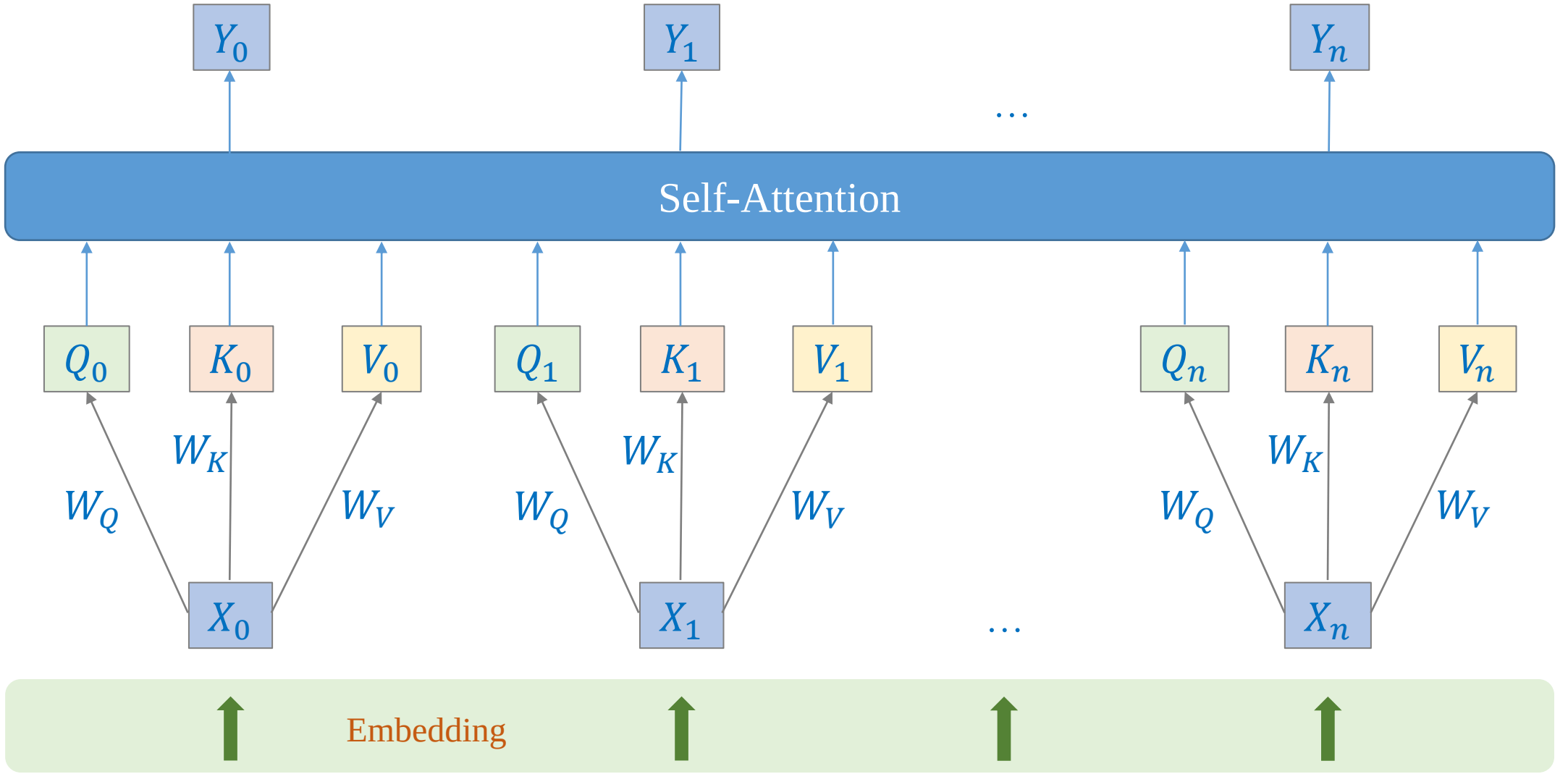
Outline

- Layer Normalization
- Transformers Block
- BERT and Text Classification
- Vision Transformer and Image Classification
- For Time-series and Tabular Data

Vision Transformer

$$Y = sigmoid\left(\frac{QK^T}{\sqrt{m}}\right)V$$

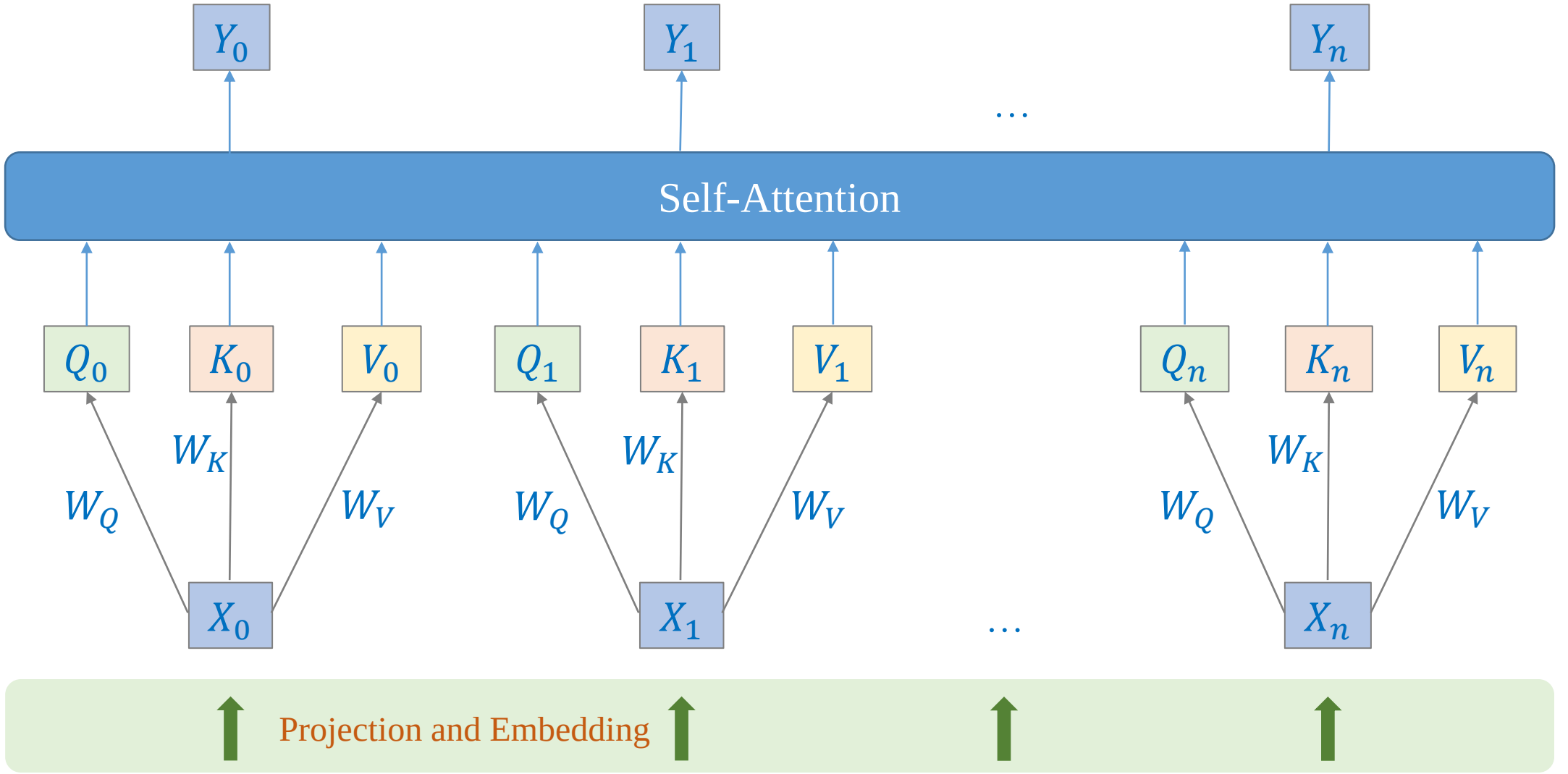
$$O = AW_O$$



Vision Transformer

$$Y = sigmoid\left(\frac{QK^T}{\sqrt{m}}\right)V$$

$$O = AW_O$$



Vision Transformer

❖ From text to image

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

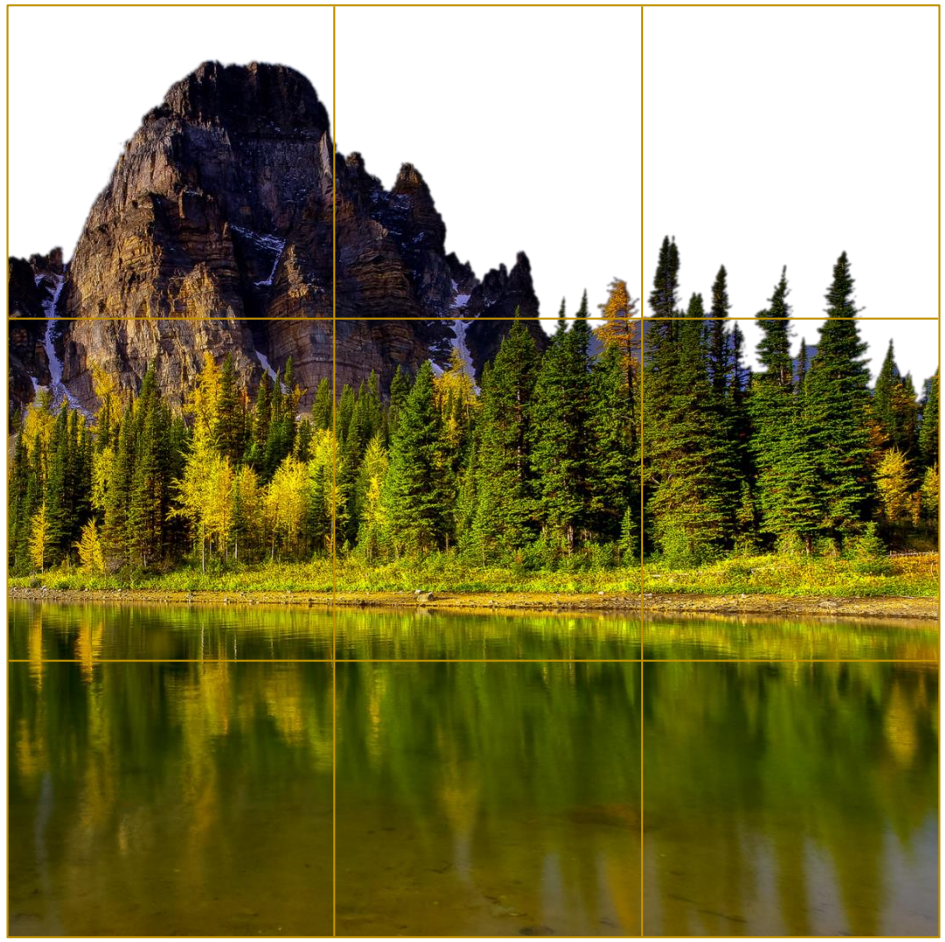
Alexey Dosovitskiy[*,†], **Lucas Beyer**[*], **Alexander Kolesnikov**[*], **Dirk Weissenborn**[*],
Xiaohua Zhai[*], **Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer,**
Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby[*,†]
[*]equal technical contribution, [†]equal advising
Google Research, Brain Team
{adosovitskiy, neilhoulsby}@google.com

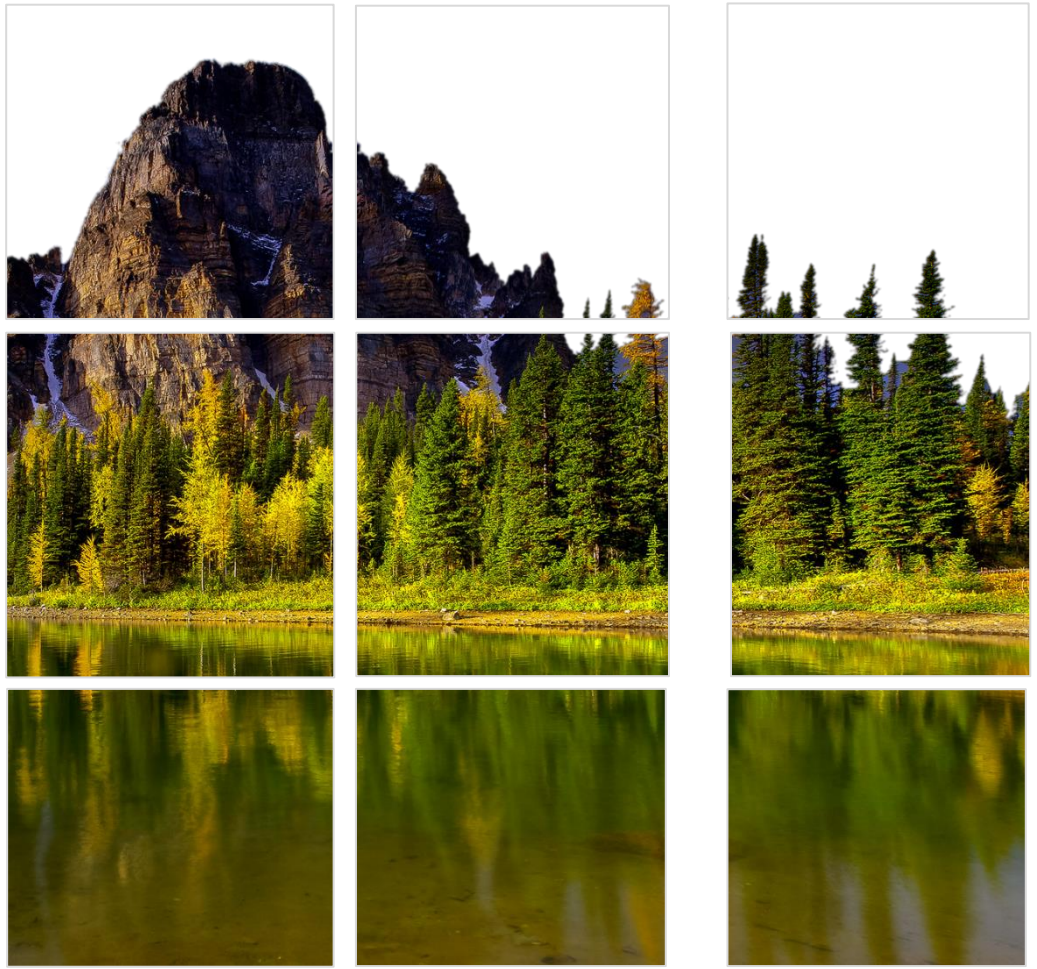
ABSTRACT

While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.[1]

Vision Transformer

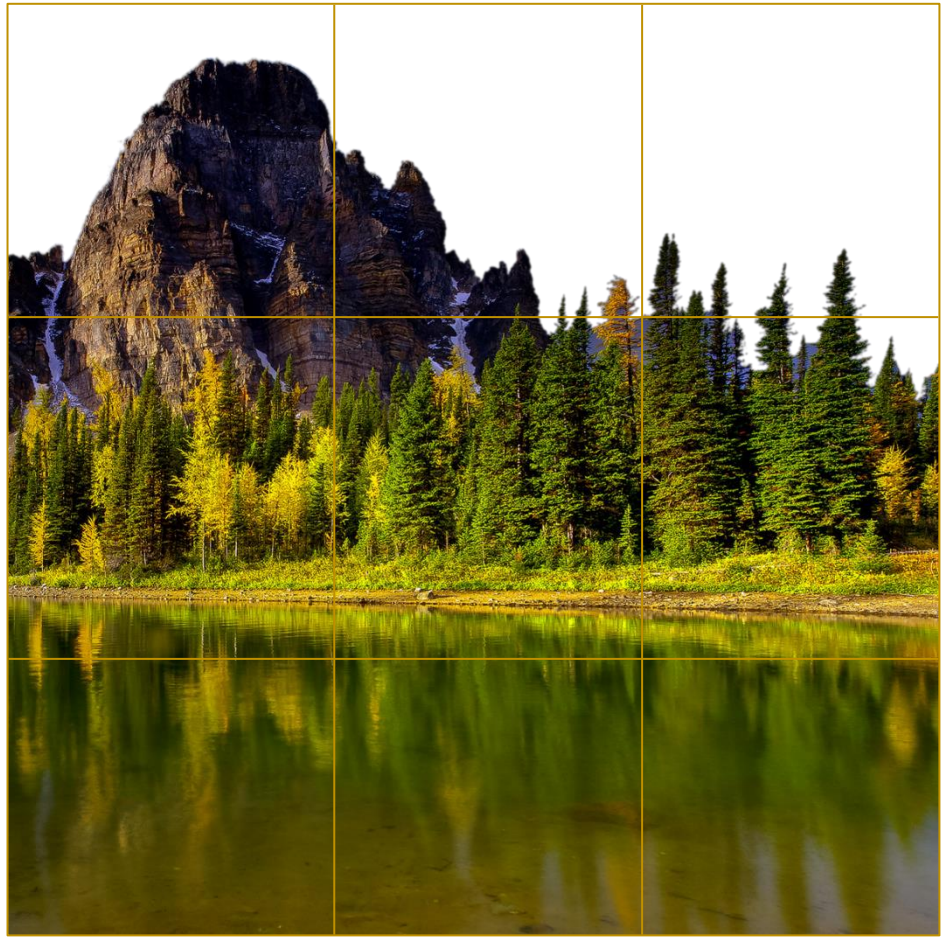
❖ Get patches





Vision Transformer

❖ Get patches



```python
class PatchEmbedding(nn.Module):
    def __init__(self, embed_dim=512,
                 patch_size=16, image_size=224):
        super().__init__()
        self.conv = nn.Conv2d(in_channels=3,
                              out_channels=embed_dim,
                              kernel_size=patch_size,
                              stride=patch_size,
                              bias=False)

    def forward(self, x):
        # [*, width, grid, grid]
        x = self.conv(x)

        # [*, width, grid ** 2]
        x = x.reshape(x.shape[0], x.shape[1], -1)

        # [*, grid ** 2, width]
        x = x.permute(0, 2, 1)

        return x
```

Vision Transformer

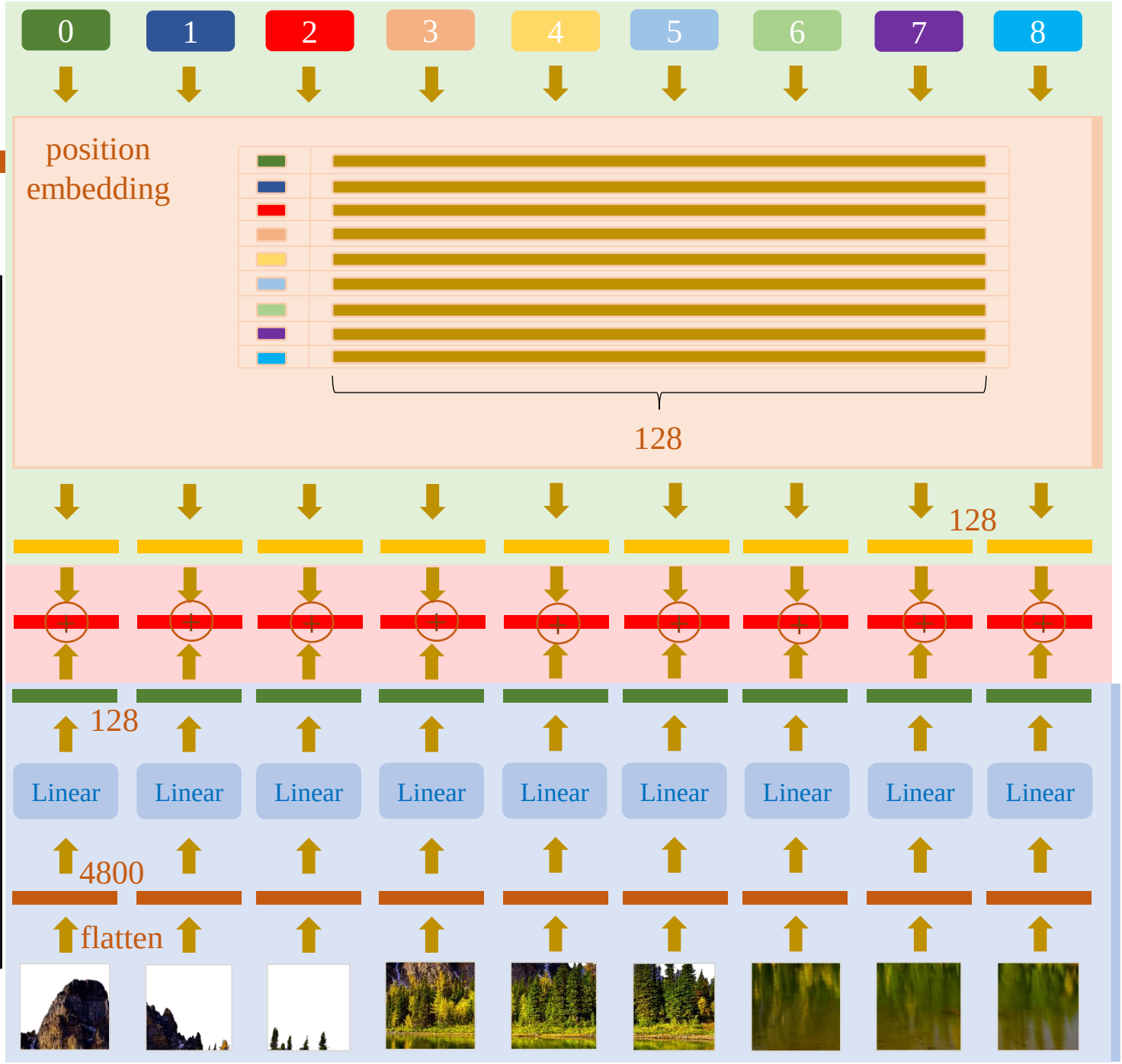
❖ Get patches


```python
class PatchPositionEmbedding(nn.Module):
    def __init__(self, image_size=224,
                 embed_dim=512, patch_size=16):
        super().__init__()
        self.conv = nn.Conv2d(in_channels=3,
                              out_channels=embed_dim,
                              kernel_size=patch_size,
                              stride=patch_size,
                              bias=False)
        scale = embed_dim ** -0.5
        param = scale * torch.randn((image_size//patch_size)**2,
                                    embed_dim)
        self.positional_embedding = nn.Parameter(param)

    def forward(self, x):
        x = self.conv(x)
        x = x.reshape(x.shape[0], x.shape[1], -1)
        x = x.permute(0, 2, 1)
        x = x + self.positional_embedding

        return x
```

Vision Transformer

❖ Patch and position embedding

```
class PatchPositionEmbedding(nn.Module):
    def __init__(self, image_size=224,
                 embed_dim=512, patch_size=16):
        super().__init__()
        self.conv = nn.Conv2d(in_channels=3,
                              out_channels=embed_dim,
                              kernel_size=patch_size,
                              stride=patch_size,
                              bias=False)
        scale = embed_dim ** -0.5
        param = scale * torch.randn((image_size//patch_size)**2,
                                    embed_dim)
        self.positional_embedding = nn.Parameter(param)

    def forward(self, x):
        x = self.conv(x)
        x = x.reshape(x.shape[0], x.shape[1], -1)
        x = x.permute(0, 2, 1)
        x = x + self.positional_embedding

        return x
```



Pretrained Vision Transformer

❖ **JFT-300M**

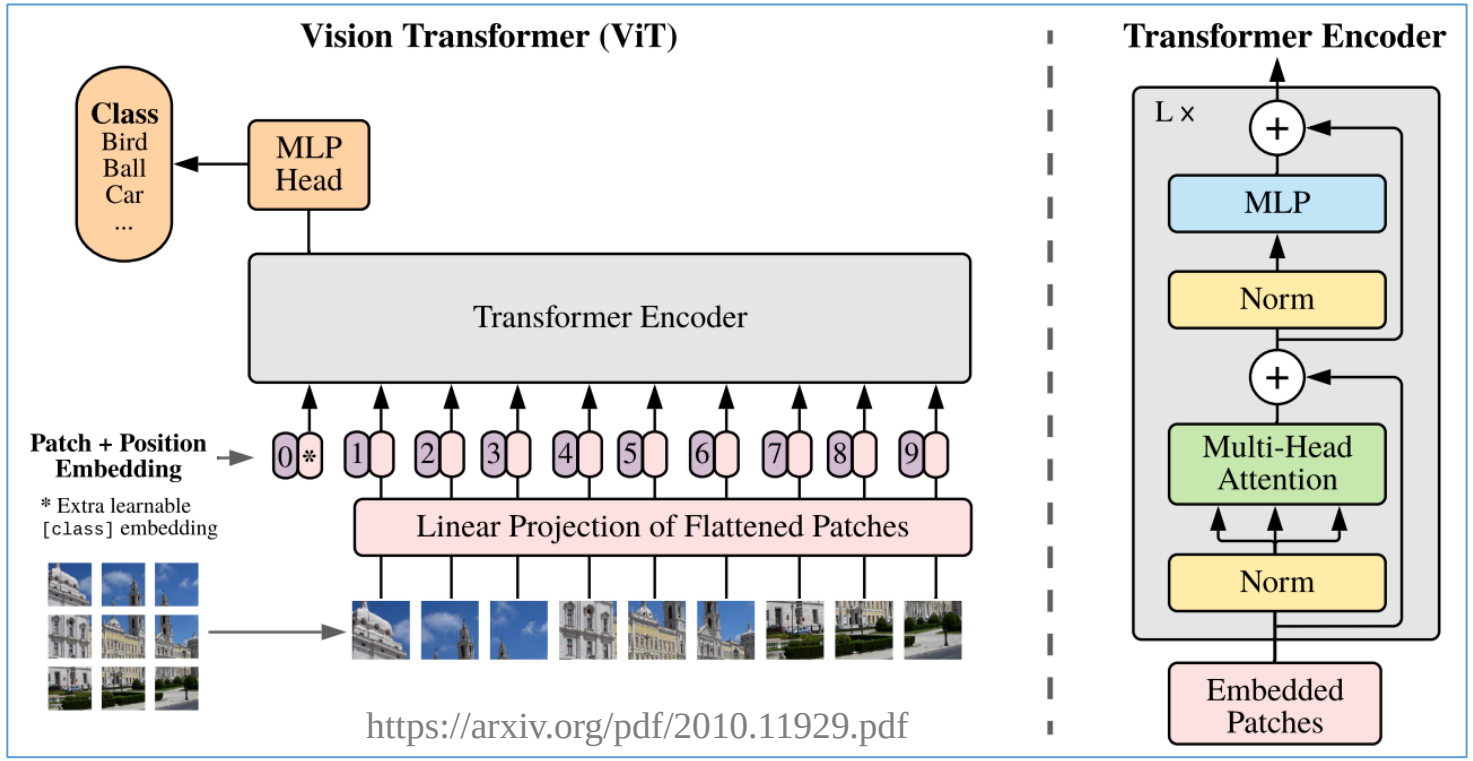
300M images
Internal Google dataset
For training image classification models

```
trainer = Trainer(
    model,
    args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    data_collator=collate_fn,
    compute_metrics=compute_metrics,
    tokenizer=feature_extractor,
)
```

```
from transformers import ViTImageProcessor
from transformers import ViTForImageClassification

# feature_extractor
feature_extractor = ViTImageProcessor.from_pretrained("google/vit-base-patch16-224-in21k")

# model
model = ViTForImageClassification.from_pretrained('google/vit-base-patch16-224-in21k', ...)
```

Vision Transformer

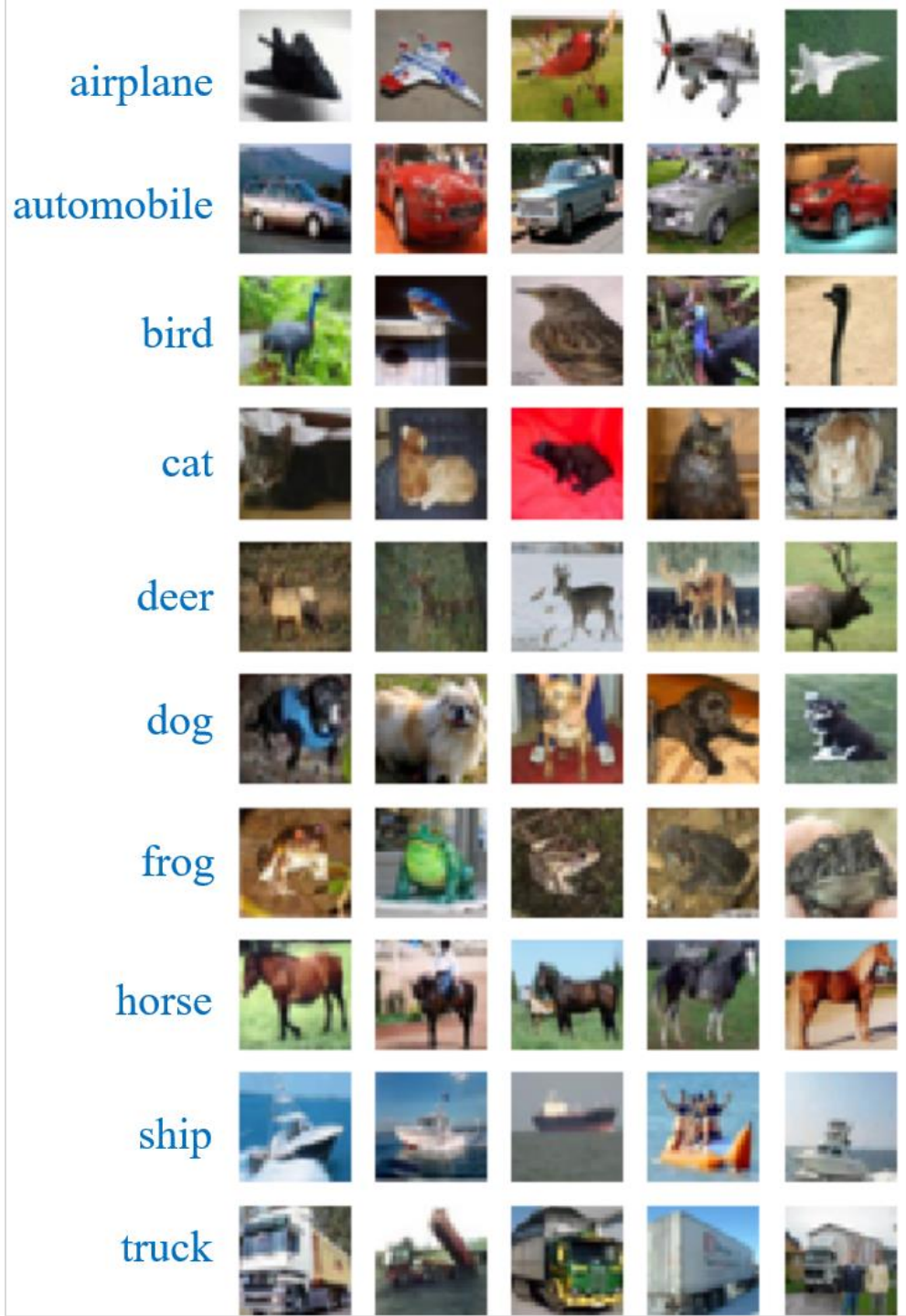
From text to image



Performance of ViT using Cifar-10 dataset

Train from scratch	Transfer Learning
78%	98%

* You may have different results in your own experiments



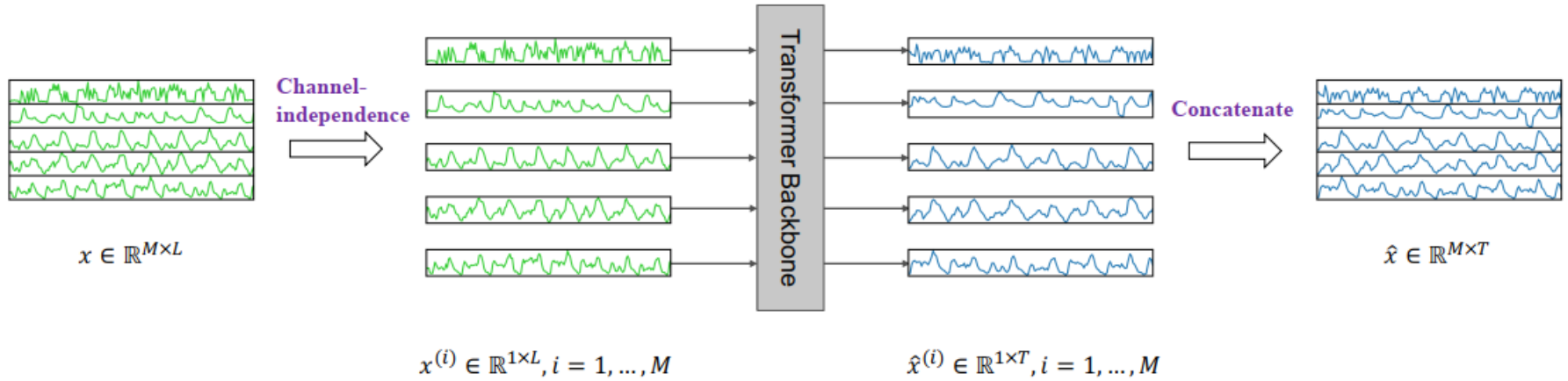
Outline

- Layer Normalization
- Transformers Block
- BERT and Text Classification
- Vision Transformer and Image Classification
- For Time-series and Tabular Data

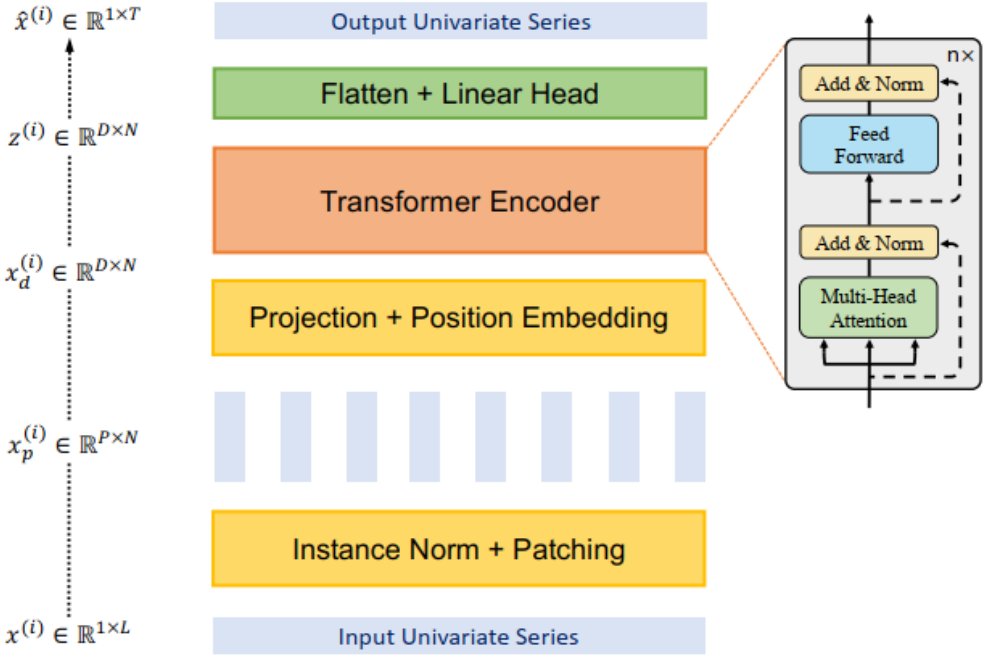
Patch Time-Series Transformer

❖ **For time-series data**

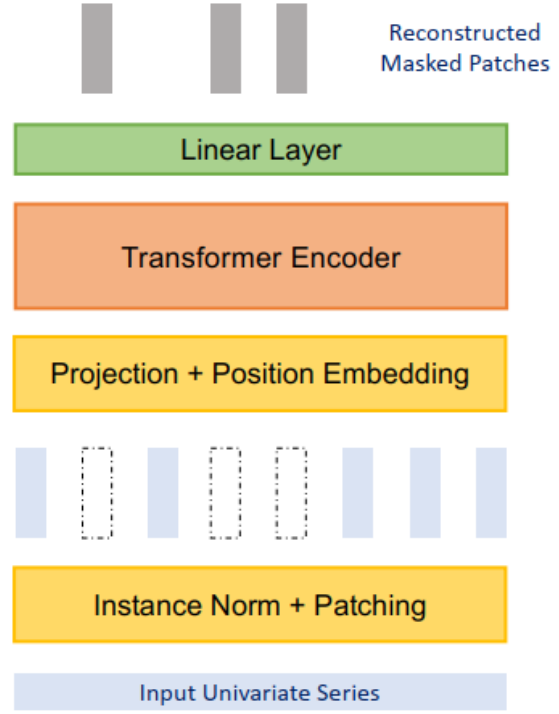
https://github.com/yuqinie98/PatchTST



(a) PatchTST Model Overview



(b) Transformer Backbone (Supervised)



(c) Transformer Backbone (Self-supervised)

Patch Time-Series Transformer

❖ For time-series data

https://github.com/yuqinie98/PatchTST

A TIME SERIES IS WORTH 64 WORDS: LONG-TERM FORECASTING WITH TRANSFORMERS

Yuqi Nie[1]*, **Nam H. Nguyen**[2]*, **Phanwadee Sinthong**[2], **Jayant Kalagnanam**[2]
[1]Princeton University [2]IBM Research
ynie@princeton.edu, nnguyen@us.ibm.com, Gift.Sinthong@ibm.com, jayant@us.ibm.com

ABSTRACT

We propose an efficient design of Transformer-based models for multivariate time series forecasting and self-supervised representation learning. It is based on two key components: (i) segmentation of time series into subseries-level patches which are served as input tokens to Transformer; (ii) channel-independence where each channel contains a single univariate time series that shares the same embedding and Transformer weights across all the series. Patching design naturally has three-fold benefit: local semantic information is retained in the embedding; computation and memory usage of the attention maps are quadratically reduced given the same look-back window; and the model can attend longer history. Our channel-independent patch time series Transformer (PatchTST) can improve the long-term forecasting accuracy significantly when compared with that of SOTA Transformer-based models. We also apply our model to self-supervised pre-training tasks and attain excellent fine-tuning performance, which outperforms supervised training on large datasets. Transferring of masked pre-trained representation on one dataset to others also produces SOTA forecasting accuracy.

Patch Time-Series Transformer

❖ For time-series data

Datasets	Weather	Traffic	Electricity	ILI	ETTh1	ETTh2	ETTm1	ETTm2
Features	21	862	321	7	7	7	7	7
Timesteps	52696	17544	26304	966	17420	17420	69680	69680

Models		PatchTST/64		PatchTST/42		DLinear		FEDformer		Autoformer		Informer		Pyraformer		LogTrans	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Weather	96	**0.149**	**0.198**	0.152	0.199	0.176	0.237	0.238	0.314	0.249	0.329	0.354	0.405	0.896	0.556	0.458	0.490
	192	**0.194**	**0.241**	0.197	0.243	0.220	0.282	0.275	0.329	0.325	0.370	0.419	0.434	0.622	0.624	0.658	0.589
	336	**0.245**	**0.282**	0.249	0.283	0.265	0.319	0.339	0.377	0.351	0.391	0.583	0.543	0.739	0.753	0.797	0.652
	720	**0.314**	**0.334**	0.320	0.335	0.323	0.362	0.389	0.409	0.415	0.426	0.916	0.705	1.004	0.934	0.869	0.675
Traffic	96	**0.360**	**0.249**	0.367	0.251	0.410	0.282	0.576	0.359	0.597	0.371	0.733	0.410	2.085	0.468	0.684	0.384
	192	**0.379**	**0.256**	0.385	0.259	0.423	0.287	0.610	0.380	0.607	0.382	0.777	0.435	0.867	0.467	0.685	0.390
	336	**0.392**	**0.264**	0.398	0.265	0.436	0.296	0.608	0.375	0.623	0.387	0.776	0.434	0.869	0.469	0.734	0.408
	720	**0.432**	**0.286**	0.434	0.287	0.466	0.315	0.621	0.375	0.639	0.395	0.827	0.466	0.881	0.473	0.717	0.396
Electricity	96	**0.129**	**0.222**	0.130	**0.222**	0.140	0.237	0.186	0.302	0.196	0.313	0.304	0.393	0.386	0.449	0.258	0.357
	192	**0.147**	**0.240**	0.148	**0.240**	0.153	0.249	0.197	0.311	0.211	0.324	0.327	0.417	0.386	0.443	0.266	0.368
	336	**0.163**	**0.259**	0.167	0.261	0.169	0.267	0.213	0.328	0.214	0.327	0.333	0.422	0.378	0.443	0.280	0.380
	720	**0.197**	**0.290**	0.202	0.291	0.203	0.301	0.233	0.344	0.236	0.342	0.351	0.427	0.376	0.445	0.283	0.376
ILI	24	**1.319**	**0.754**	1.522	0.814	2.215	1.081	2.624	1.095	2.906	1.182	4.657	1.449	1.420	2.012	4.480	1.444
	36	1.579	0.870	**1.430**	**0.834**	1.963	0.963	2.516	1.021	2.585	1.038	4.650	1.463	7.394	2.031	4.799	1.467
	48	**1.553**	**0.815**	1.673	0.854	2.130	1.024	2.505	1.041	3.024	1.145	5.004	1.542	7.551	2.057	4.800	1.468
	60	**1.470**	**0.788**	1.529	0.862	2.368	1.096	2.742	1.122	2.761	1.114	5.071	1.543	7.662	2.100	5.278	1.560
ETTh1	96	**0.370**	0.400	0.375	**0.399**	0.375	**0.399**	0.376	0.415	0.435	0.446	0.941	0.769	0.664	0.612	0.878	0.740
	192	0.413	0.429	0.414	0.421	**0.405**	**0.416**	0.423	0.446	0.456	0.457	1.007	0.786	0.790	0.681	1.037	0.824
	336	**0.422**	0.440	0.431	**0.436**	0.439	0.443	0.444	0.462	0.486	0.487	1.038	0.784	0.891	0.738	1.238	0.932
	720	**0.447**	0.468	0.449	**0.466**	0.472	0.490	0.469	0.492	0.515	0.517	1.144	0.857	0.963	0.782	1.135	0.852
ETTh2	96	**0.274**	0.337	**0.274**	**0.336**	0.289	0.353	0.332	0.374	0.332	0.368	1.549	0.952	0.645	0.597	2.116	1.197
	192	0.341	0.382	**0.339**	**0.379**	0.383	0.418	0.407	0.446	0.426	0.434	3.792	1.542	0.788	0.683	4.315	1.635
	336	**0.329**	0.384	0.331	**0.380**	0.448	0.465	0.400	0.447	0.477	0.479	4.215	1.642	0.907	0.747	1.124	1.604
	720	**0.379**	**0.422**	**0.379**	**0.422**	0.605	0.551	0.412	0.469	0.453	0.490	3.656	1.619	0.963	0.783	3.188	1.540
ETTm1	96	0.293	0.346	**0.290**	**0.342**	0.299	0.343	0.326	0.390	0.510	0.492	0.626	0.560	0.543	0.510	0.600	0.546
	192	0.333	0.370	**0.332**	0.369	0.335	**0.365**	0.365	0.415	0.514	0.495	0.725	0.619	0.557	0.537	0.837	0.700
	336	0.369	0.392	**0.366**	0.392	0.369	**0.386**	0.392	0.425	0.510	0.492	1.005	0.741	0.754	0.655	1.124	0.832
	720	**0.416**	**0.420**	0.420	0.424	0.425	0.421	0.446	0.458	0.527	0.493	1.133	0.845	0.908	0.724	1.153	0.820
ETTm2	96	0.166	0.256	**0.165**	**0.255**	0.167	0.260	0.180	0.271	0.205	0.293	0.355	0.462	0.435	0.507	0.768	0.642
	192	0.223	0.296	**0.220**	**0.292**	0.224	0.303	0.252	0.318	0.278	0.336	0.595	0.586	0.730	0.673	0.989	0.757
	336	**0.274**	**0.329**	0.278	**0.329**	0.281	0.342	0.324	0.364	0.343	0.379	1.270	0.871	1.201	0.845	1.334	0.872
	720	**0.362**	**0.385**	0.367	**0.385**	0.397	0.421	0.410	0.420	0.414	0.419	3.001	1.267	3.625	1.451	3.048	1.328

TabTransformer

❖ For Tabular Data

TabTransformer: Tabular Data Modeling Using Contextual Embeddings

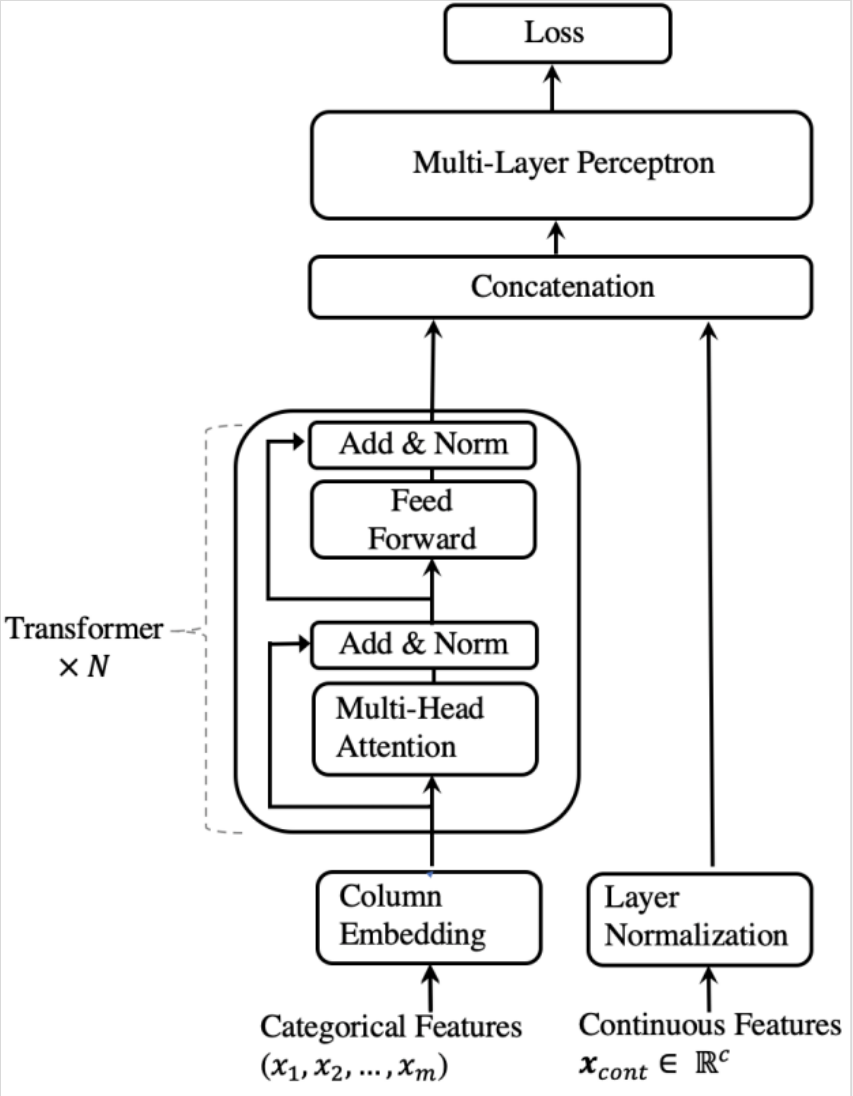
Xin Huang,[1] **Ashish Khetan,** [1] **Milan Cvitkovic** [2] **Zohar Karnin** [1]
[1] Amazon AWS
[2] PostEra
xinxh@amazon.com, khetan@amazon.com, mwcvitkovic@gmail.com, zkarnin@amazon.com

Abstract

We propose TabTransformer, a novel deep tabular data modeling architecture for supervised and semi-supervised learning. The TabTransformer is built upon self-attention based Transformers. The Transformer layers transform the embeddings of categorical features into robust contextual embeddings to achieve higher prediction accuracy. Through extensive experiments on fifteen publicly available datasets, we show that the TabTransformer outperforms the state-of-the-art deep learning methods for tabular data by at least 1.0% on mean AUC, and matches the performance of tree-based ensemble models. Furthermore, we demonstrate that the contextual embeddings learned from TabTransformer are highly robust against both missing and noisy data features, and provide better interpretability. Lastly, for the semi-supervised setting we develop an unsupervised pre-training procedure to learn data-driven contextual embeddings, resulting in an average 2.1% AUC lift over the state-of-the-art methods.

semi-supervised learning methods. This is due to the fact that the basic decision tree learner does not produce reliable probability estimation to its predictions (Tanha, Someren, and Afsarmanesh 2017). (c) The state-of-the-art deep learning methods (Devlin et al. 2019) to handle missing and noisy data features do not apply to them. Also, robustness of tree-based models has not been studied much in literature.

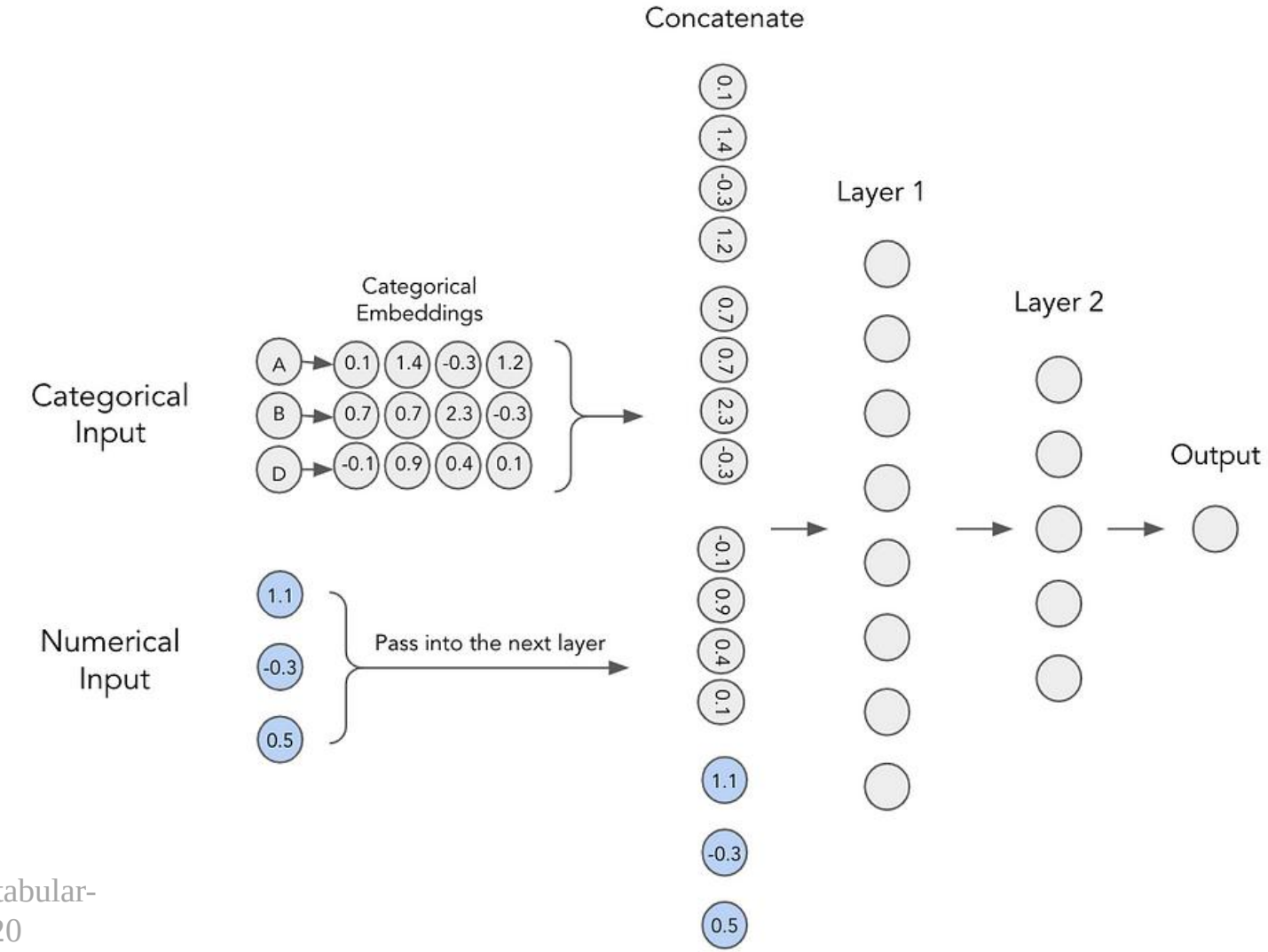
A classical and popular model that is trained using gradient descent and hence allows end-to-end learning of image/text encoders is multi-layer perceptron (MLP). The MLPs usually learn parametric embeddings to encode categorical data features. But due to their shallow architecture and context-free embeddings, they have the following limitations: (a) neither the model nor the learned embeddings are interpretable; (b) it is not robust against missing and noisy data (Section 3.2); (c) for semi-supervised learning, they do not achieve competitive performance (Section 3.4). Most importantly, MLPs do not match the performance of



TabTransformer

❖ For Tabular Data

Categorical Embeddings

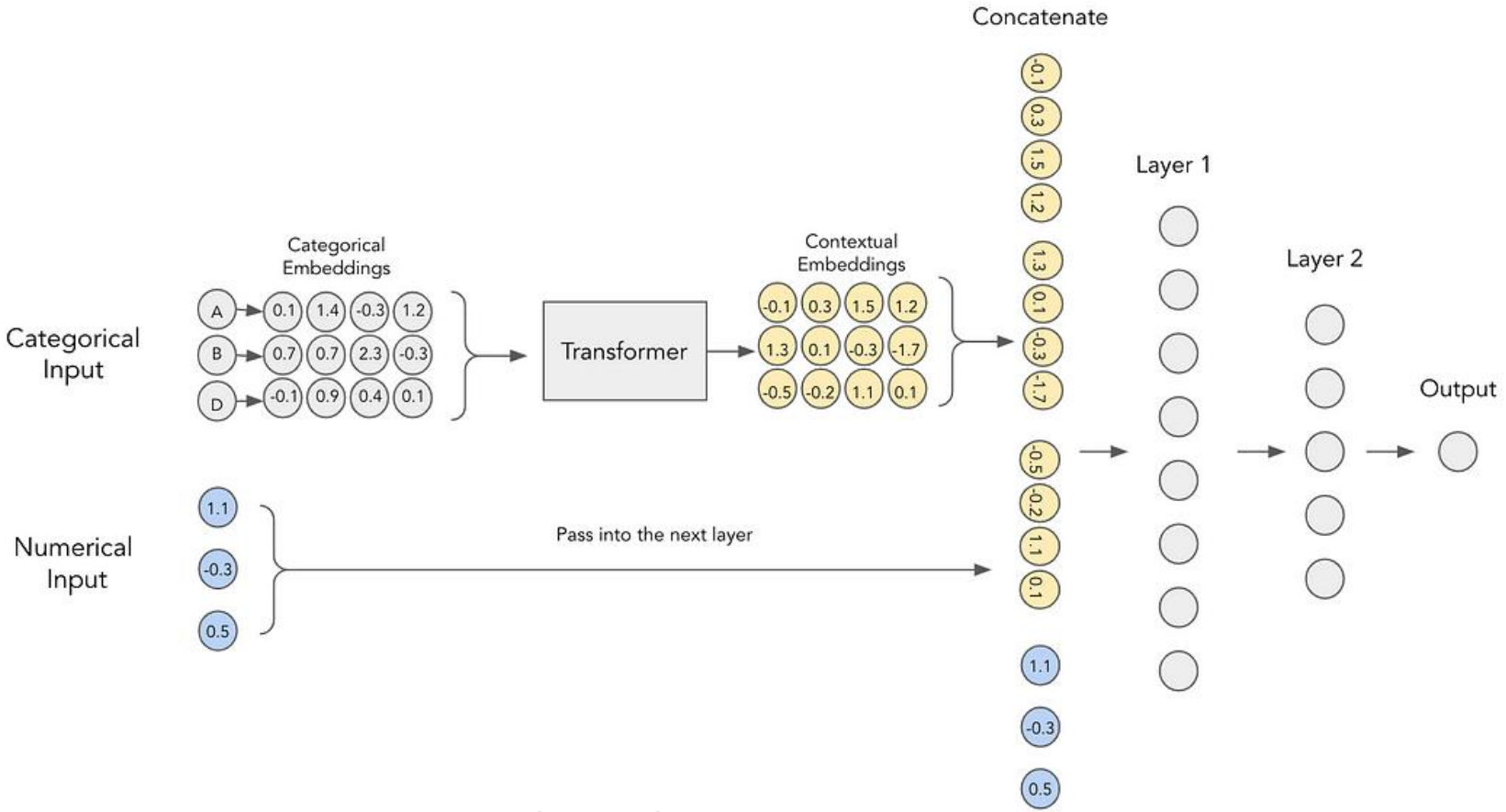


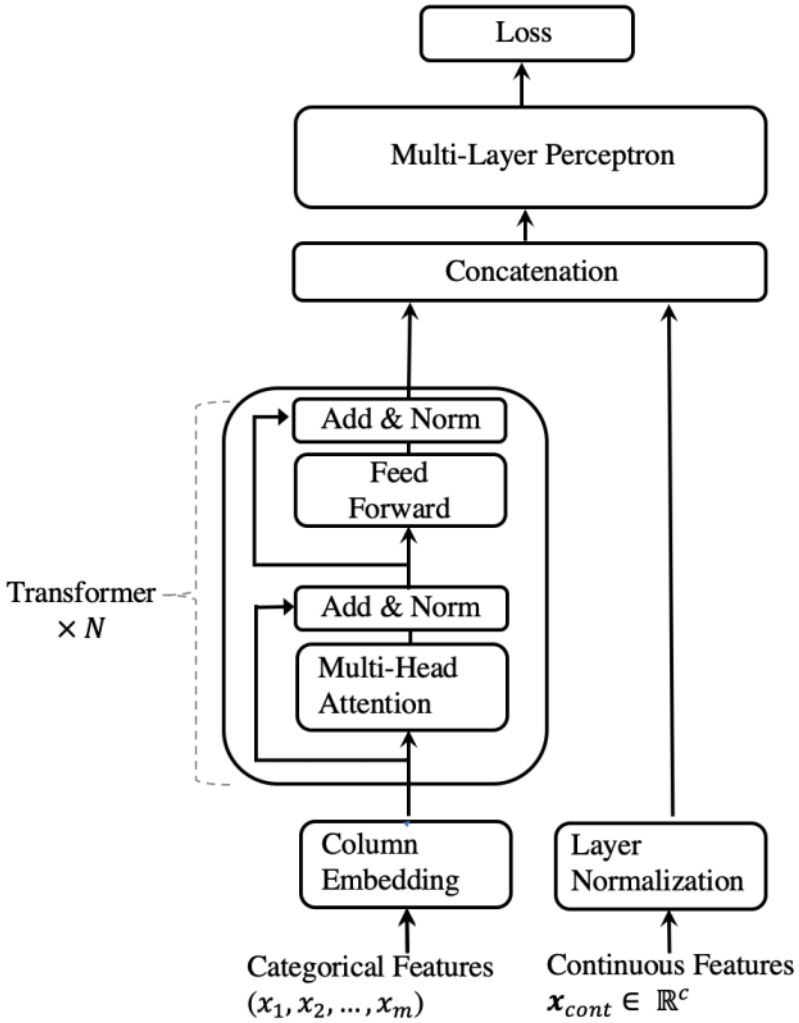
https://towardsdatascience.com/transformers-for-tabular-data-tabtransformer-deep-dive-5fb2438da820

TabTransformer

❖ For Tabular Data

Contextual Embeddings





https://towardsdatascience.com/transformers-for-tabular-data-tabtransformer-deep-dive-5fb2438da820

https://keras.io/examples/structured_data/tabtransformer/