

Lecture 11

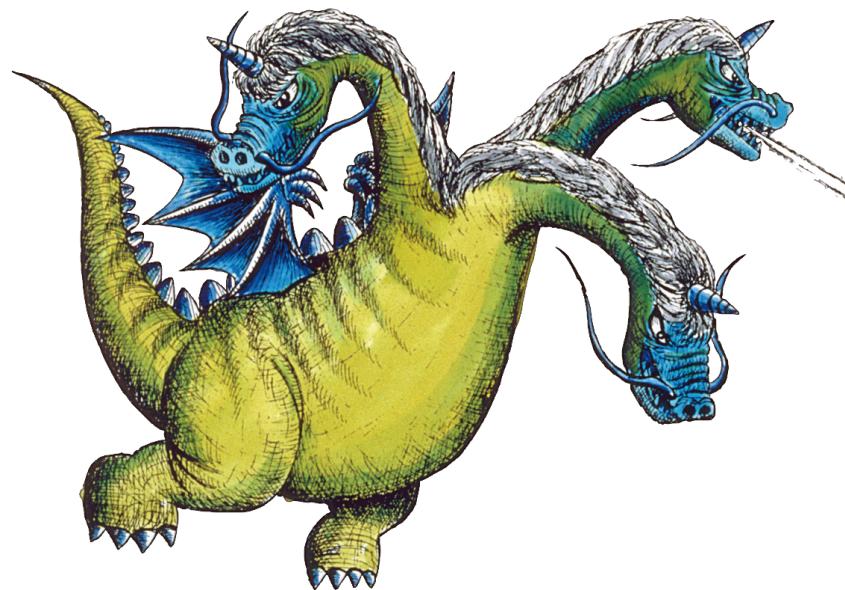
Tian Han

Outline

- Transformer
- Bert
- Autoencoder

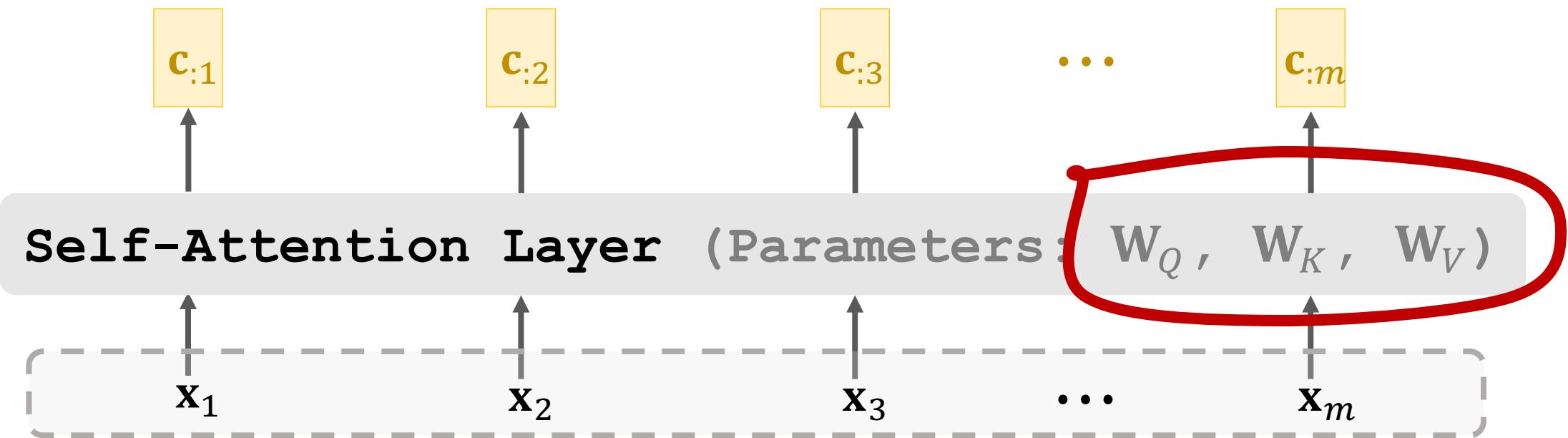
Transformer Model: From Shallow to Deep

Multi-Head Attention



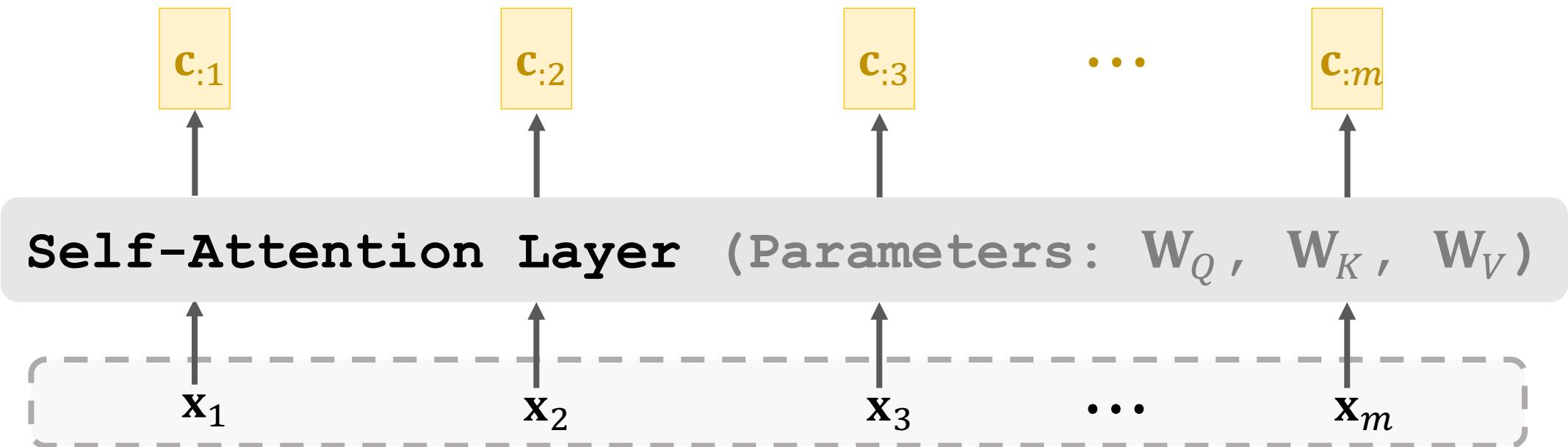
Single-Head Self-Attention

- Self-attention layer: $\mathbf{C} = \text{Attn}(\mathbf{X}, \mathbf{X})$.
- This is called “single-head self-attention”.



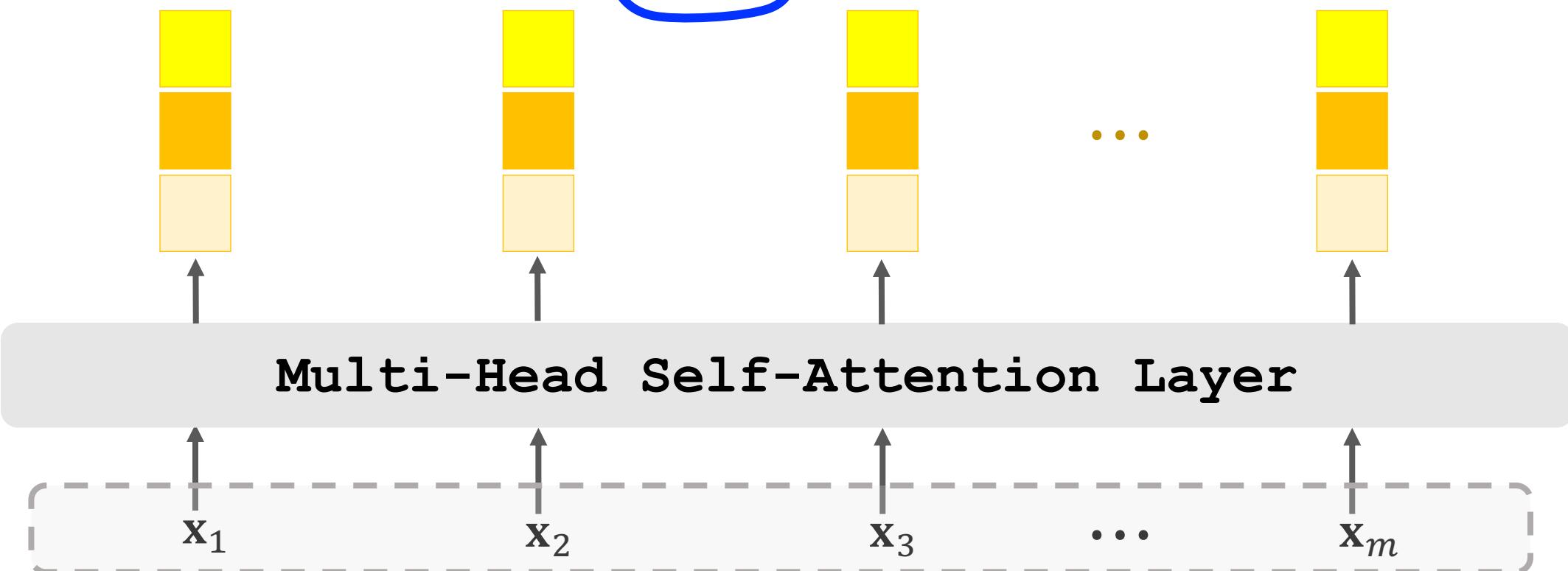
Multi-Head Self-Attention

- Using l single-head self-attentions (which do not share parameters.)
 - A single-head self-attention has 3 parameter matrices: \mathbf{W}_Q , \mathbf{W}_K , \mathbf{W}_V .
 - Totally $3l$ parameters matrices.



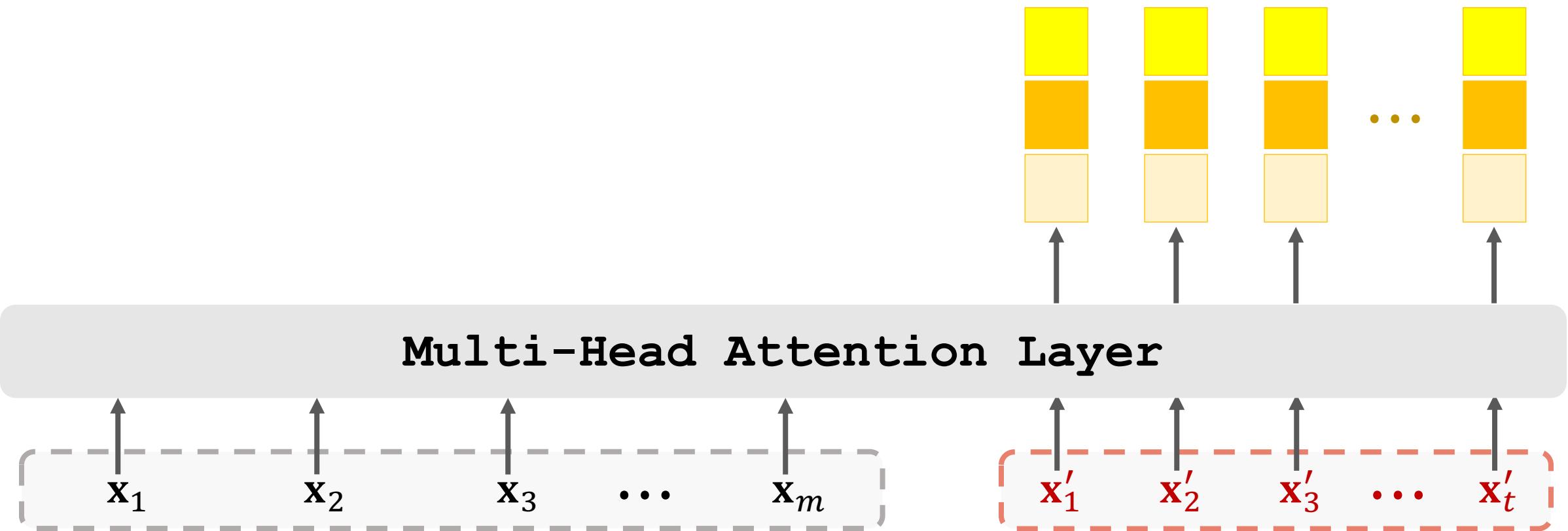
Multi-Head Self-Attention

- Using l single-head self-attentions (which do not share parameters.)
- Concatenating outputs of single-head self-attentions.
 - Suppose single-head self-attentions' outputs are $d \times 1$ vectors.
 - Multi-head's outputs are $(ld) \times 1$ vectors.



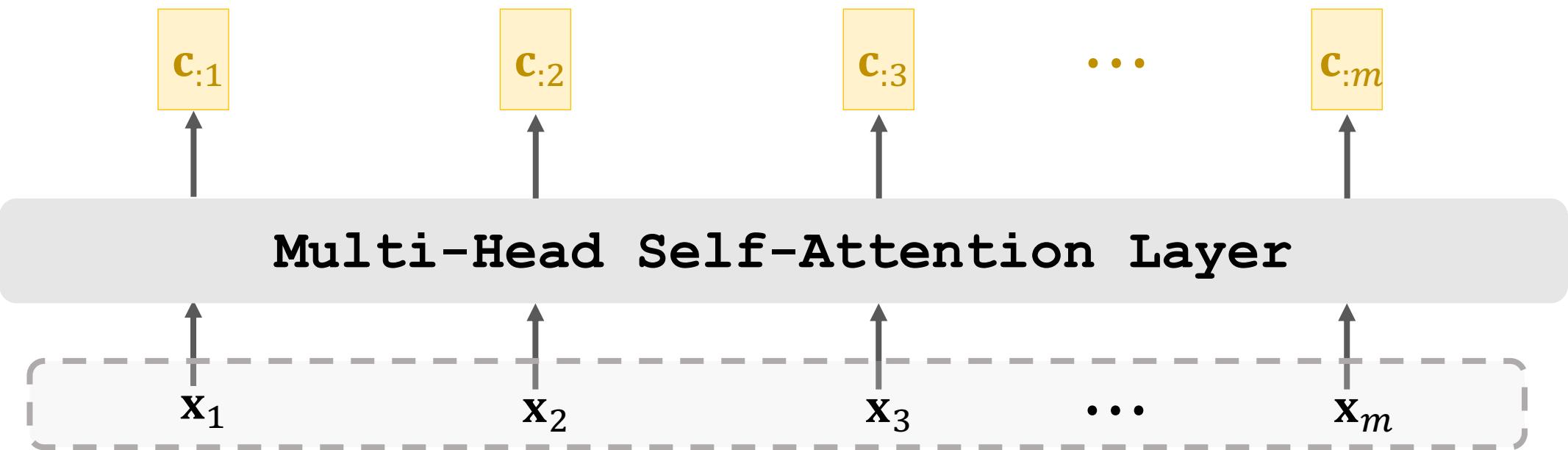
Multi-Head Attention

- Using l single-head attentions (which do not share parameters.)
- Concatenating single-head attentions' outputs.

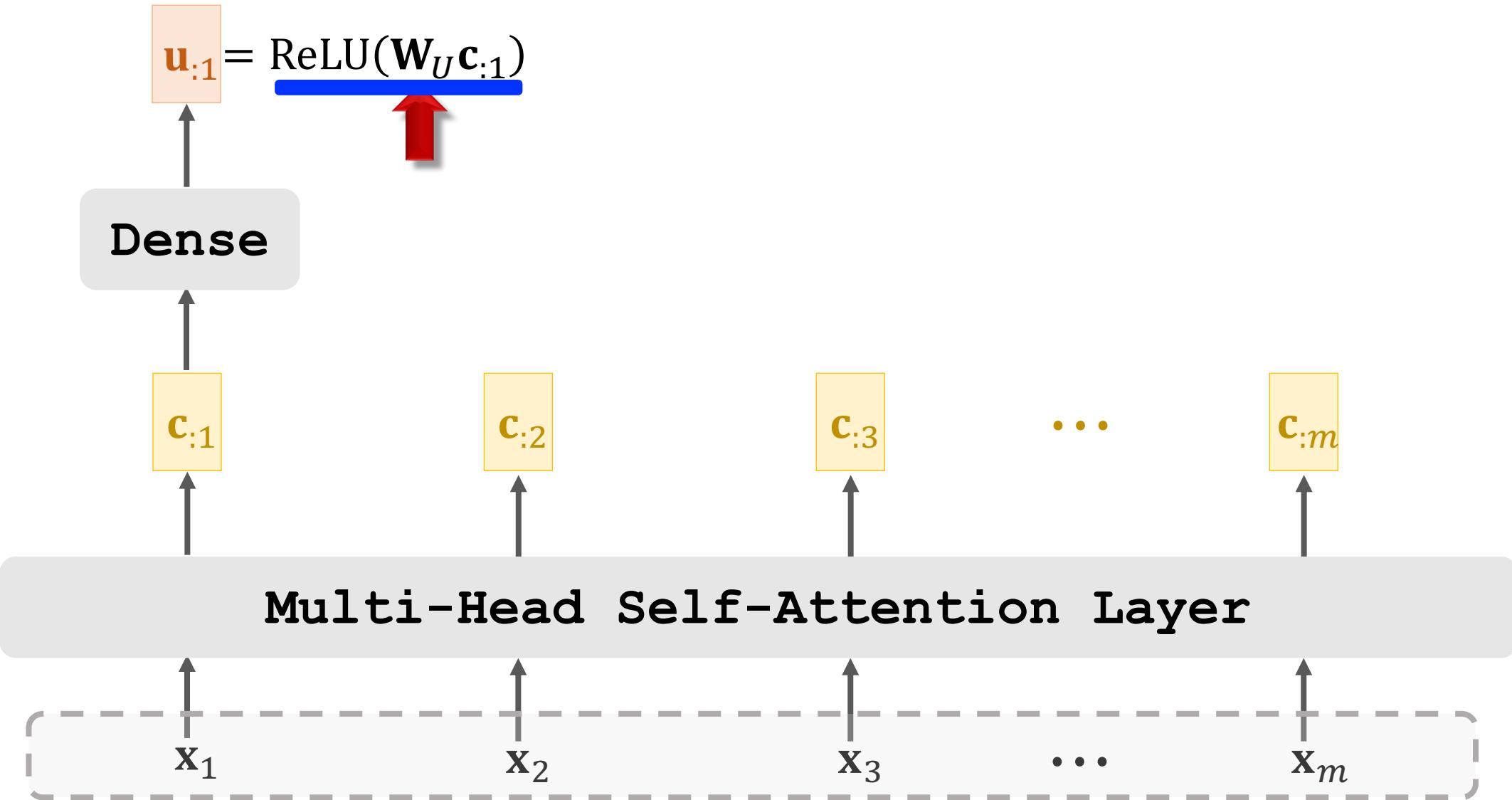


Encoder Network: Stacked Self-Attention Layers

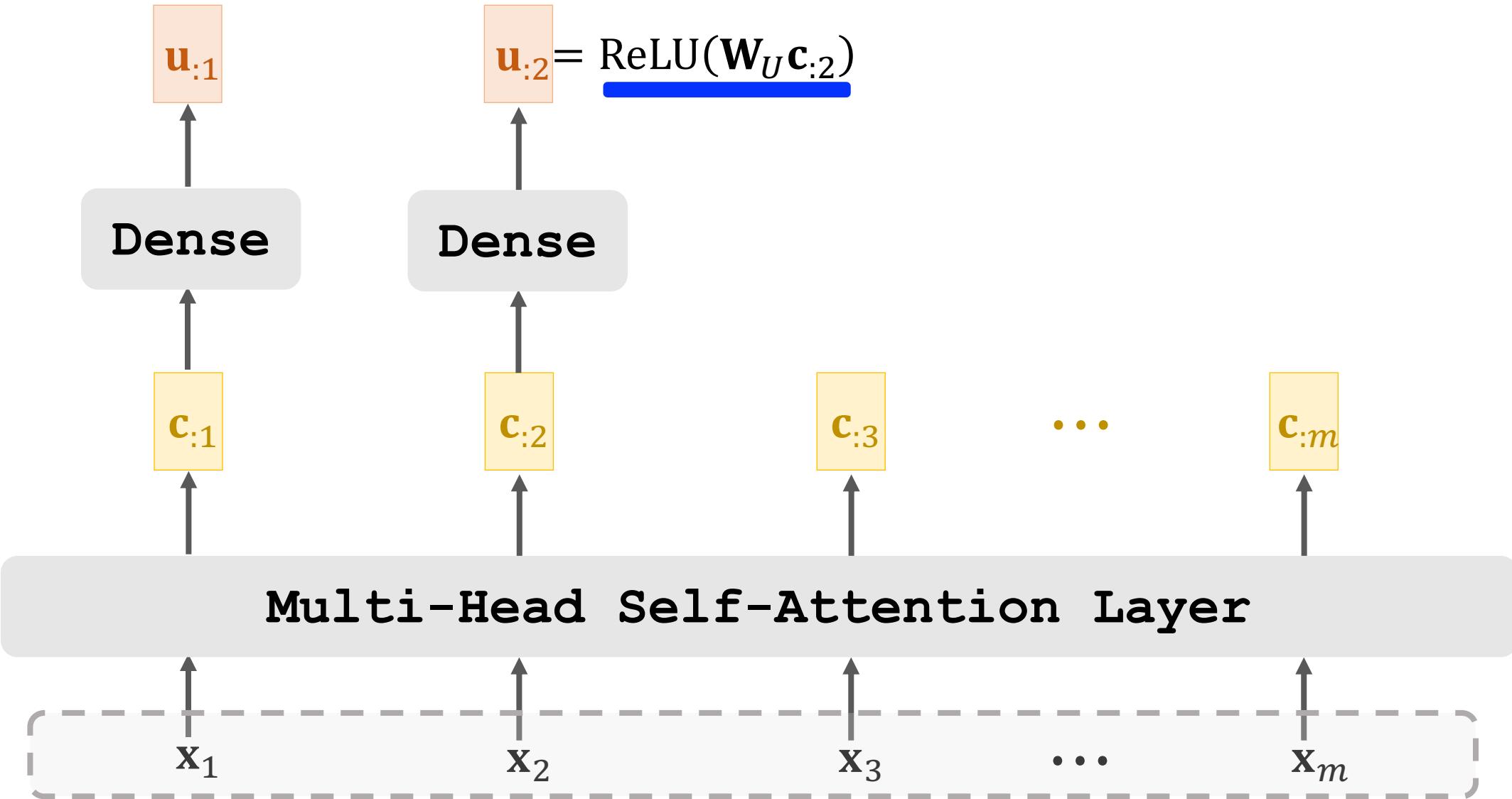
Self-Attention Layer



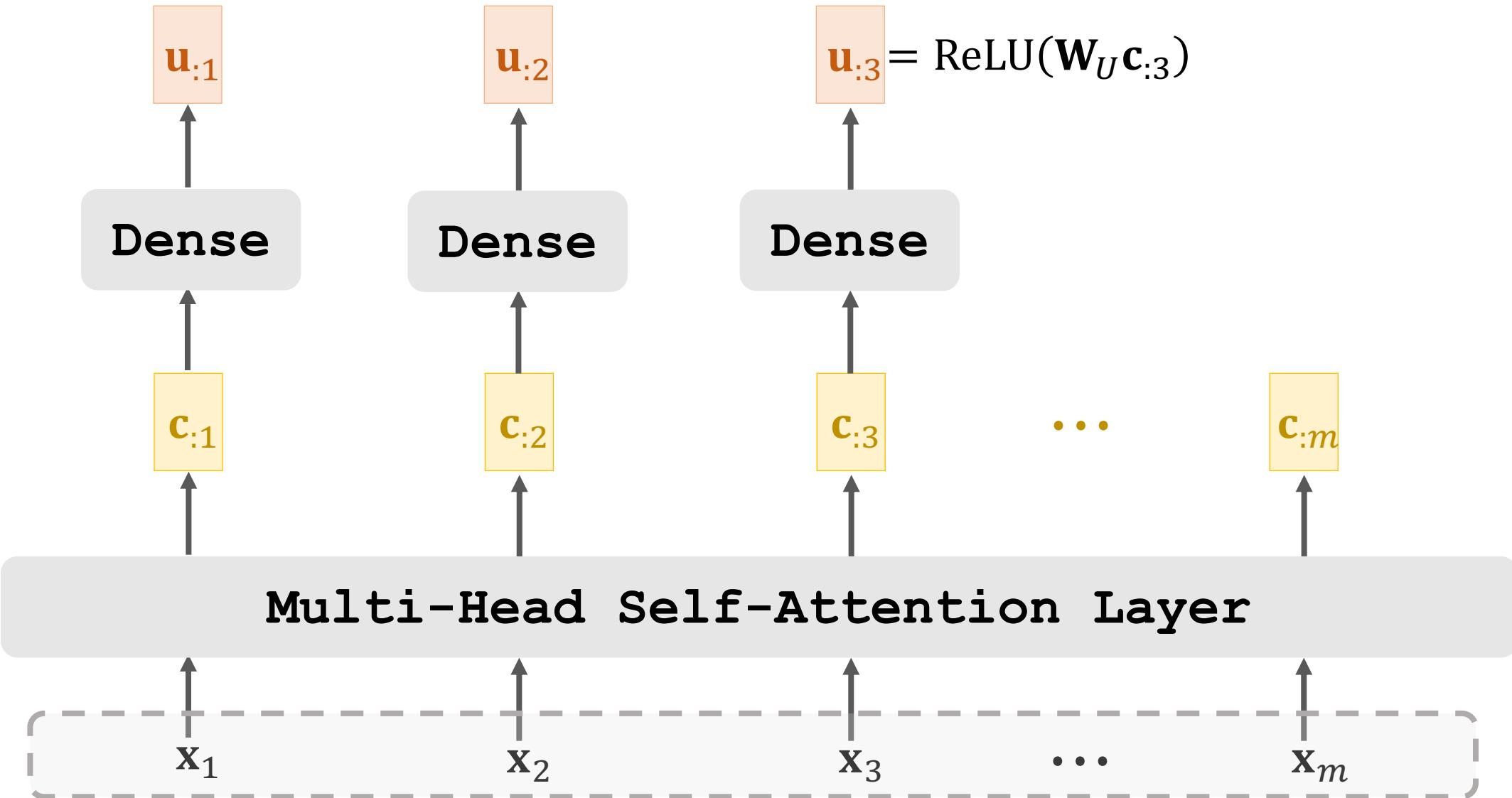
Self-Attention Layer + Dense Layer



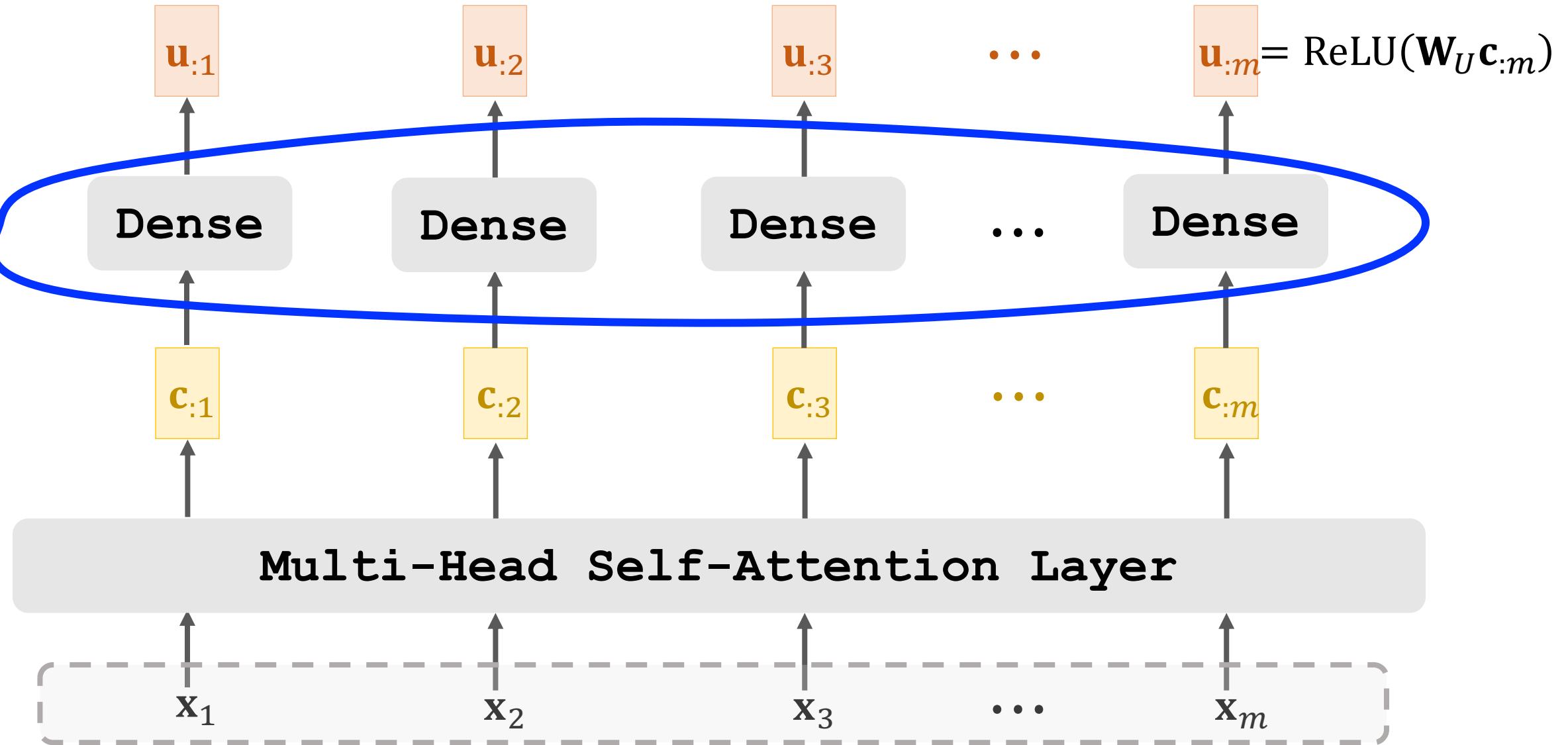
Self-Attention Layer + Dense Layer



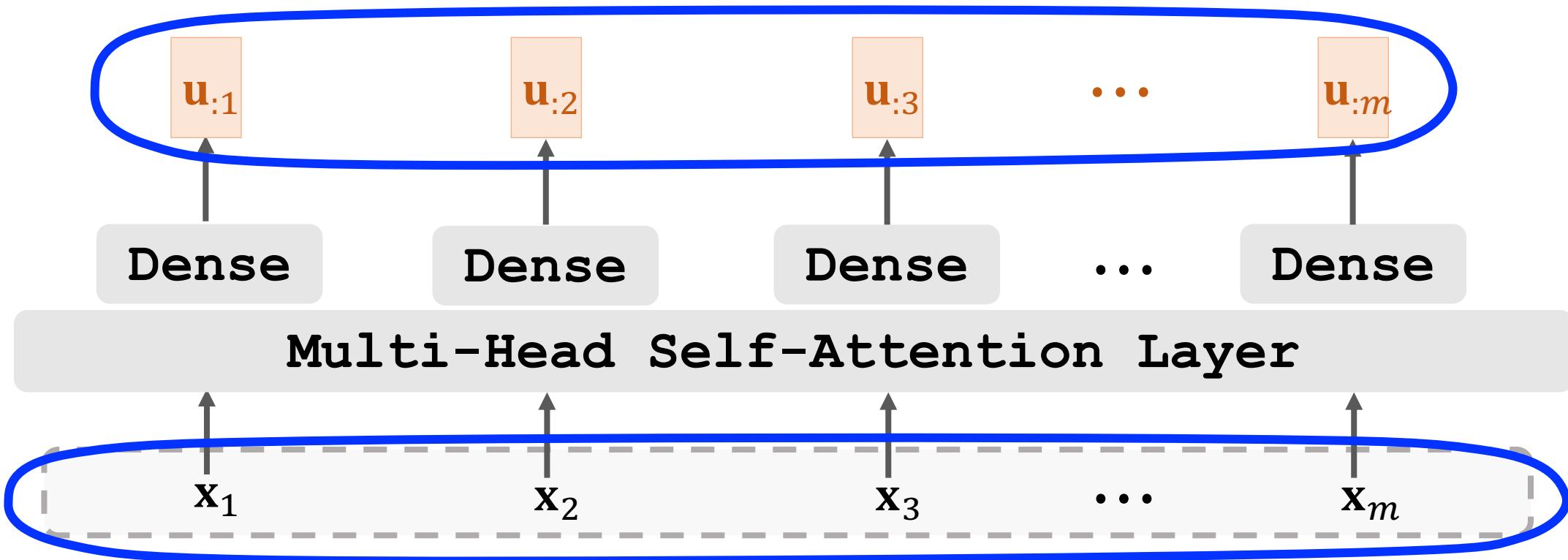
Self-Attention Layer + Dense Layer



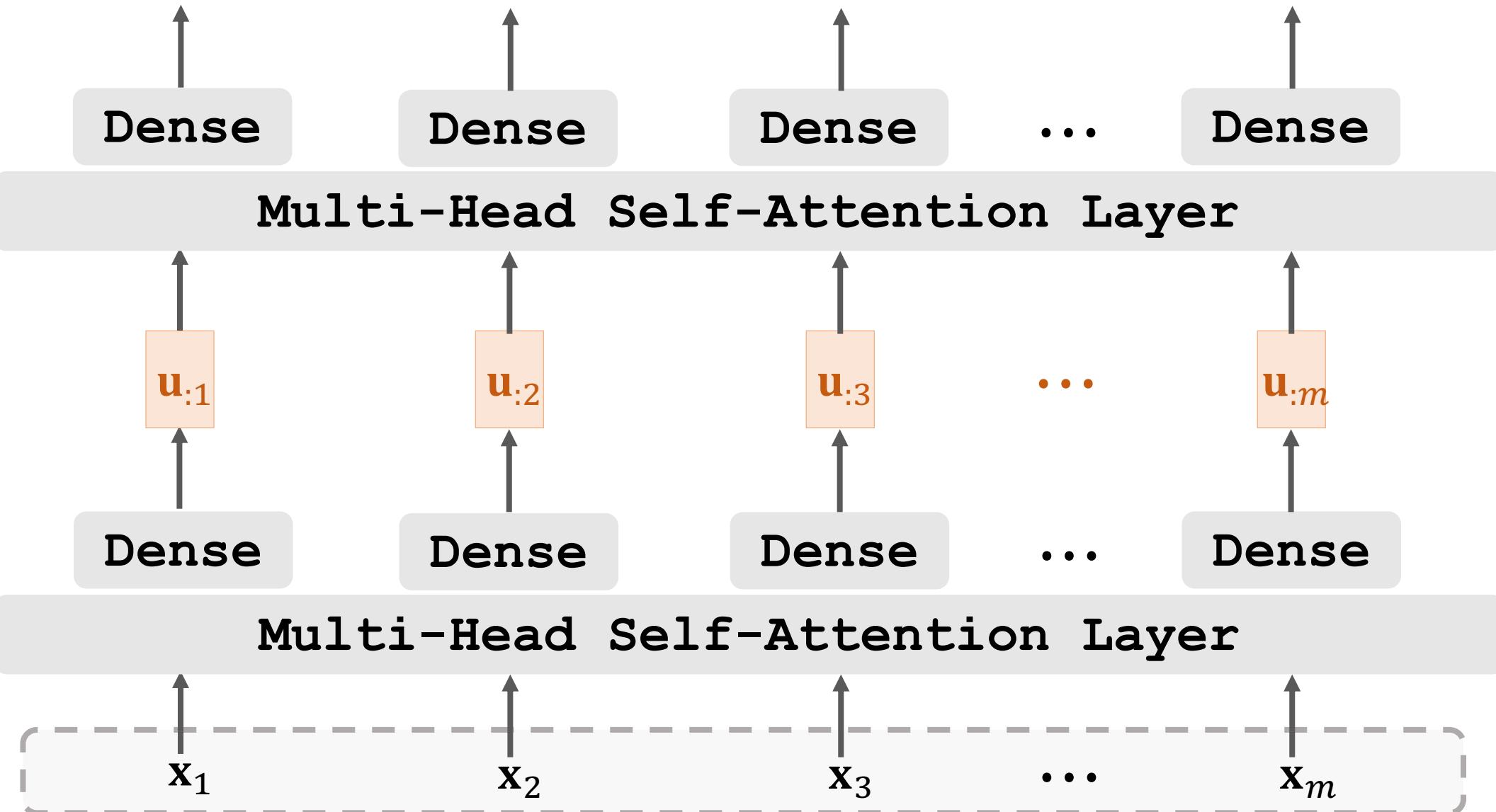
Self-Attention Layer + Dense Layer

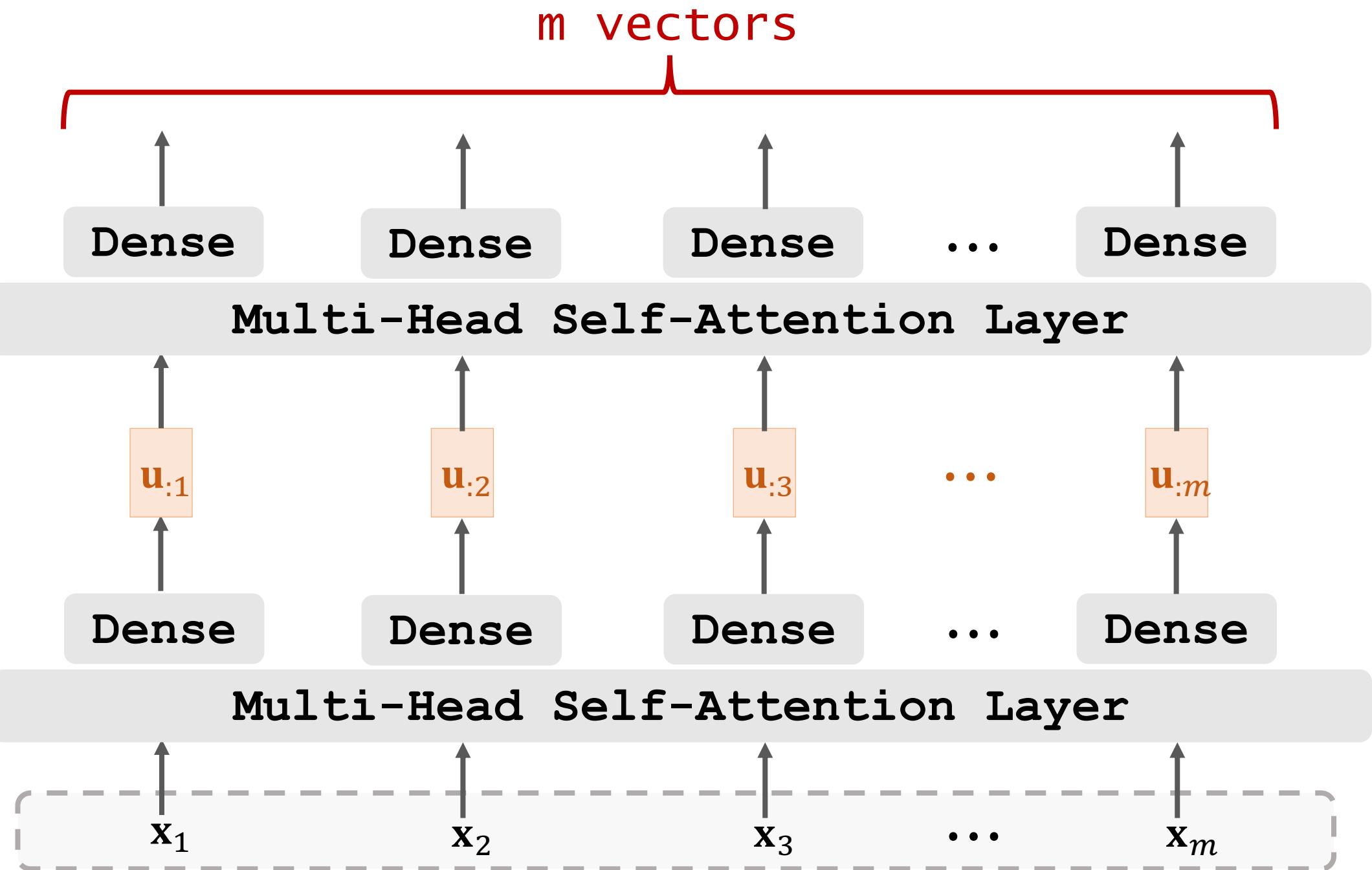


Stacked Self-Attention Layers

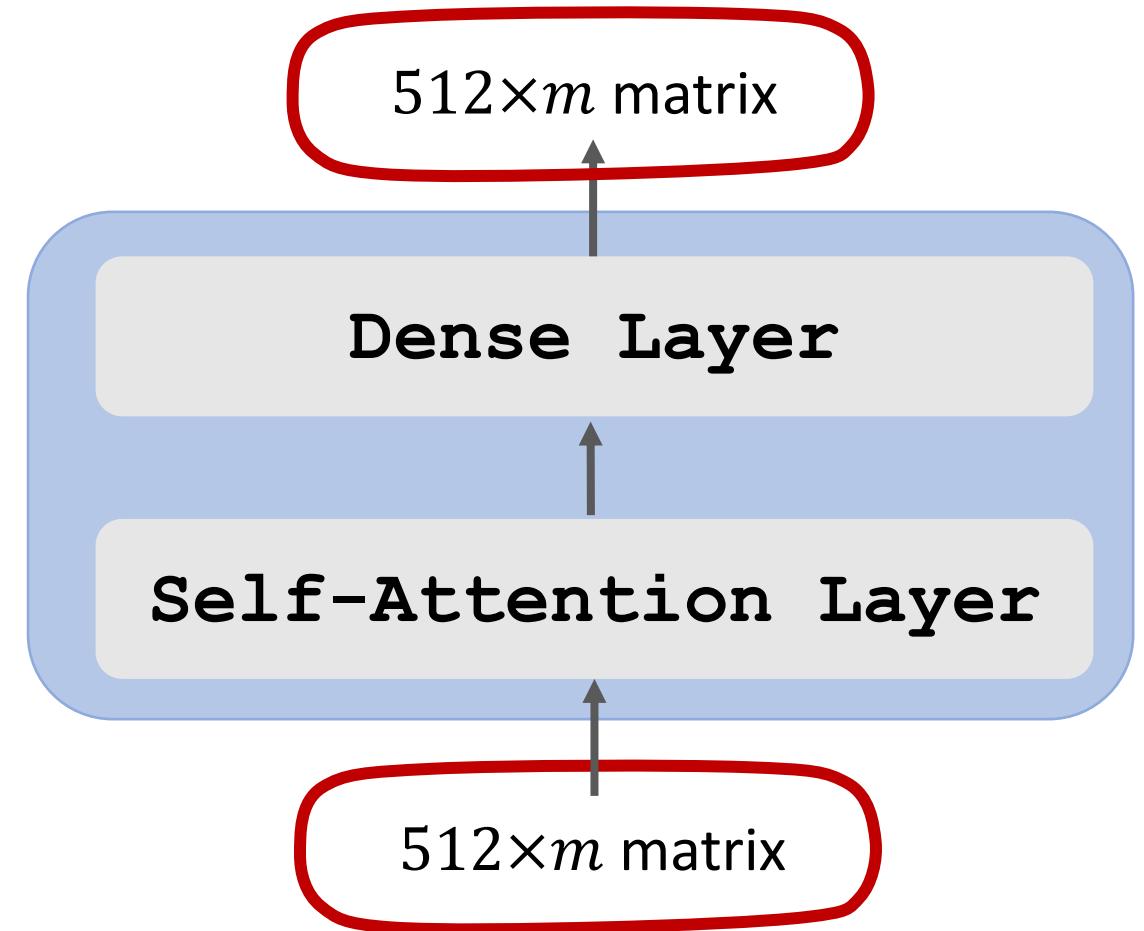


Stacked Self-Attention Layers





Transformer's Encoder

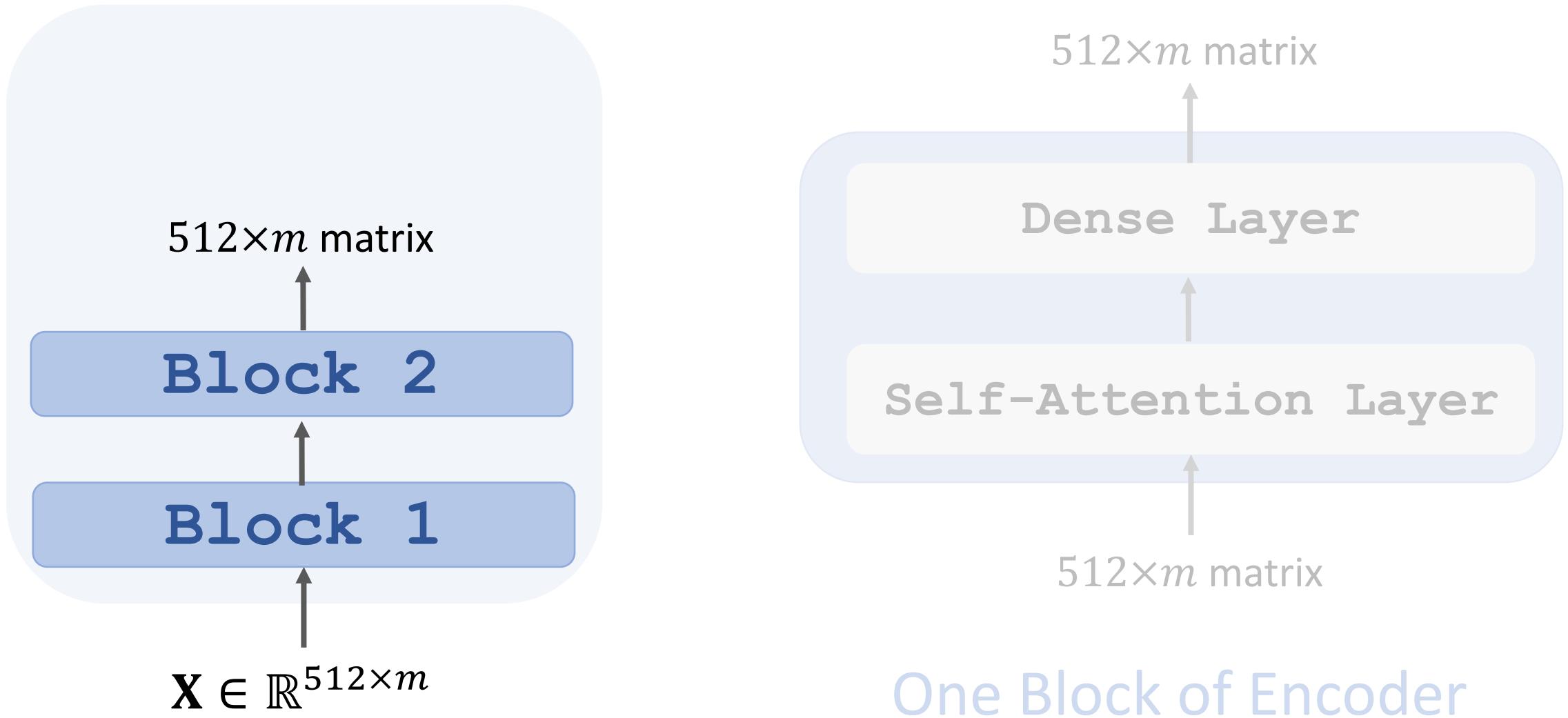


One Block of Encoder

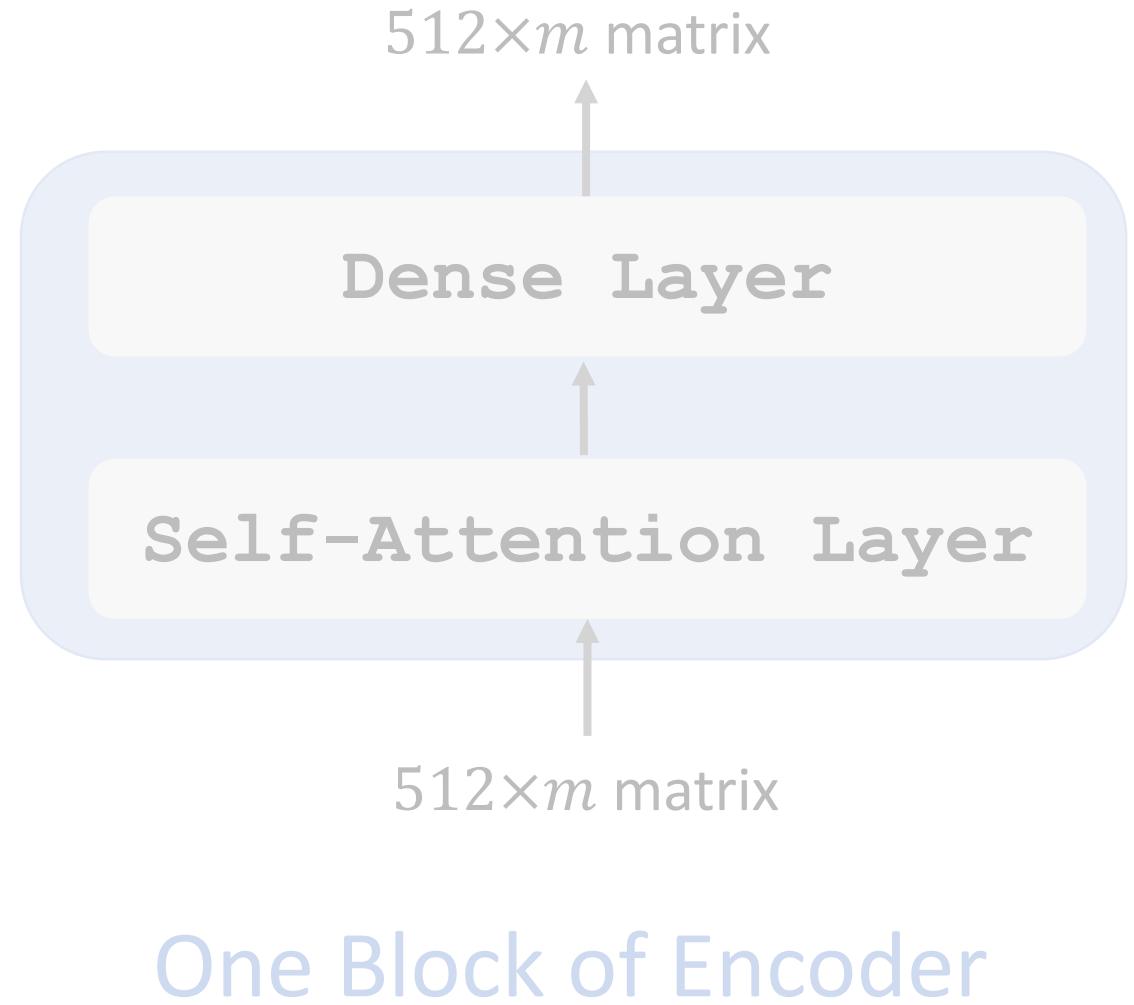
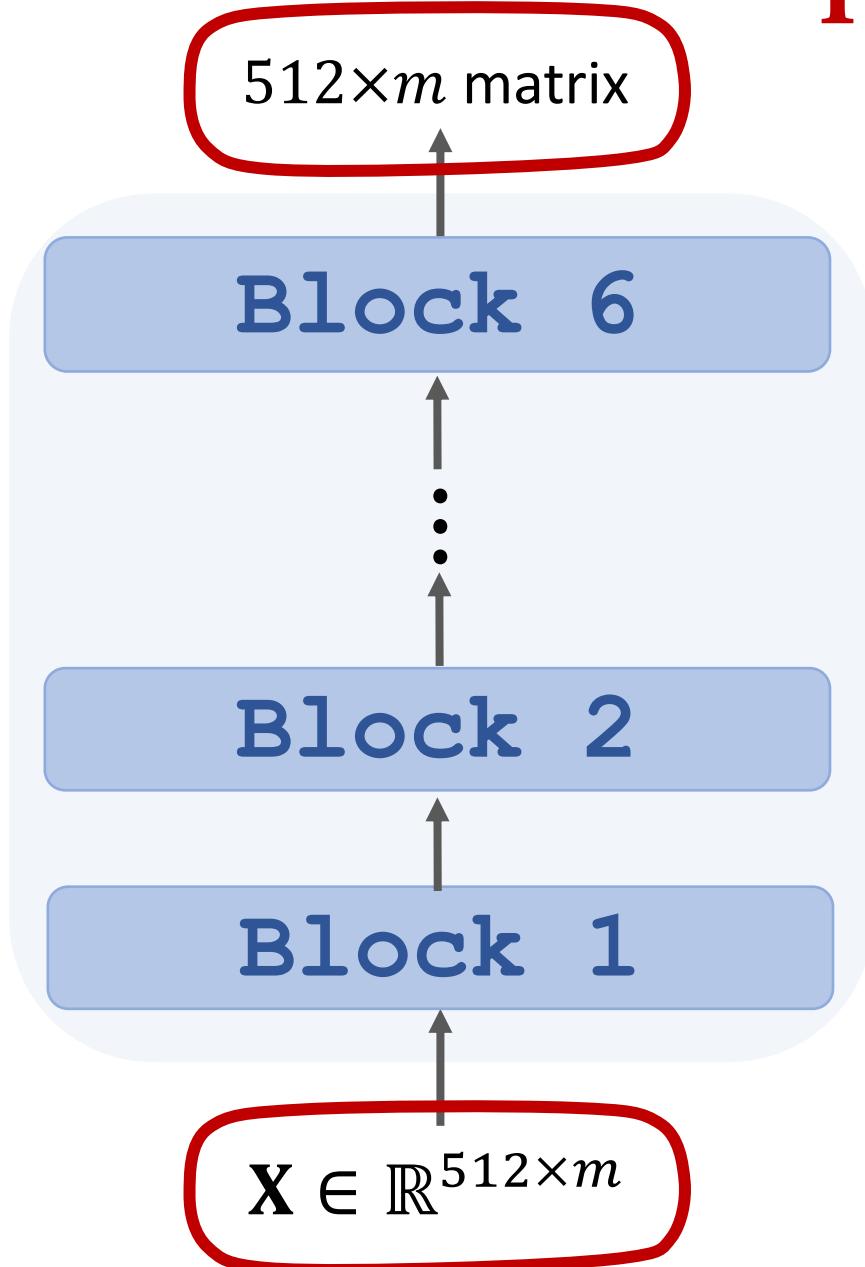
Transformer's Encoder



Transformer's Encoder



Transformer's Encoder



Decoder Network: Stacked Attention Layers

Stacked Attentions

- Transformer is a Seq2Seq model (encoder + decoder).

Encoder's inputs:

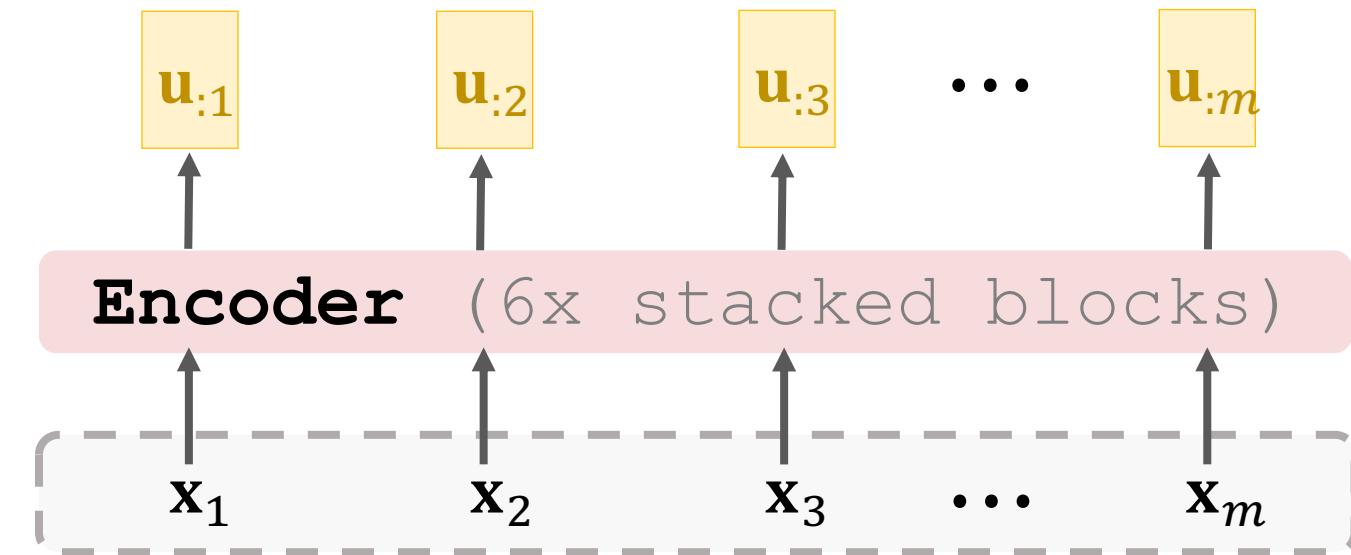
$\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \dots \quad \mathbf{x}_m$

Decoder's inputs:

$\mathbf{x}'_1 \quad \mathbf{x}'_2 \quad \mathbf{x}'_3 \quad \dots \quad \mathbf{x}'_t$

Stacked Attentions

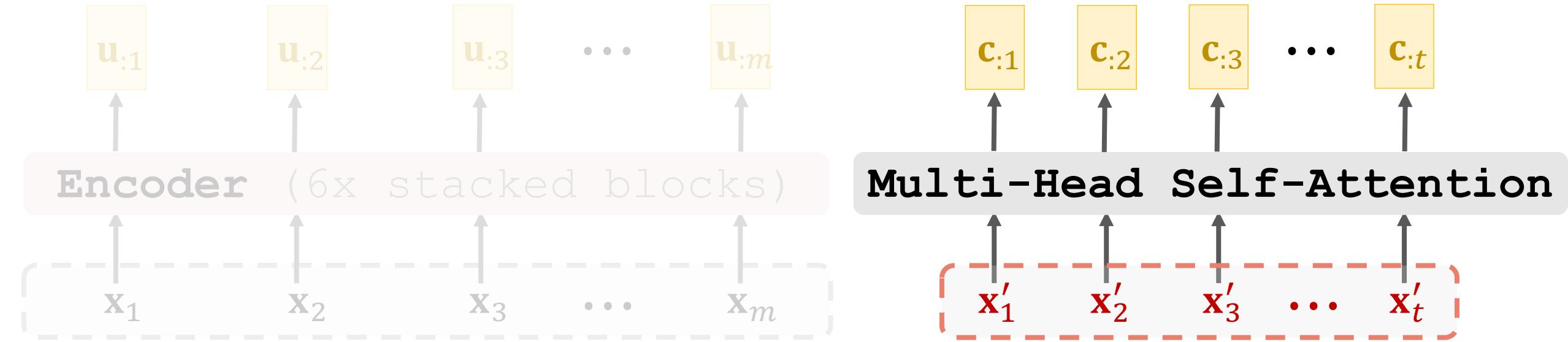
- Transformer's encoder contains **6** stacked blocks.
- 1 block \approx 1 multi-head attention layer + 1 dense layer.



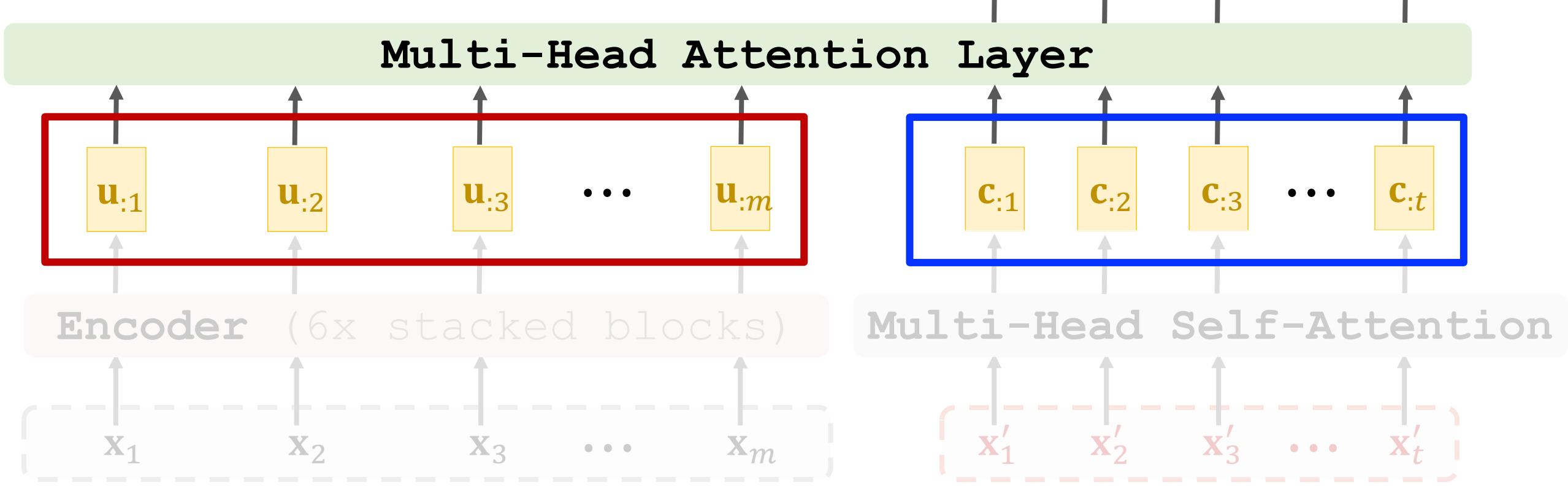
Decoder's inputs:



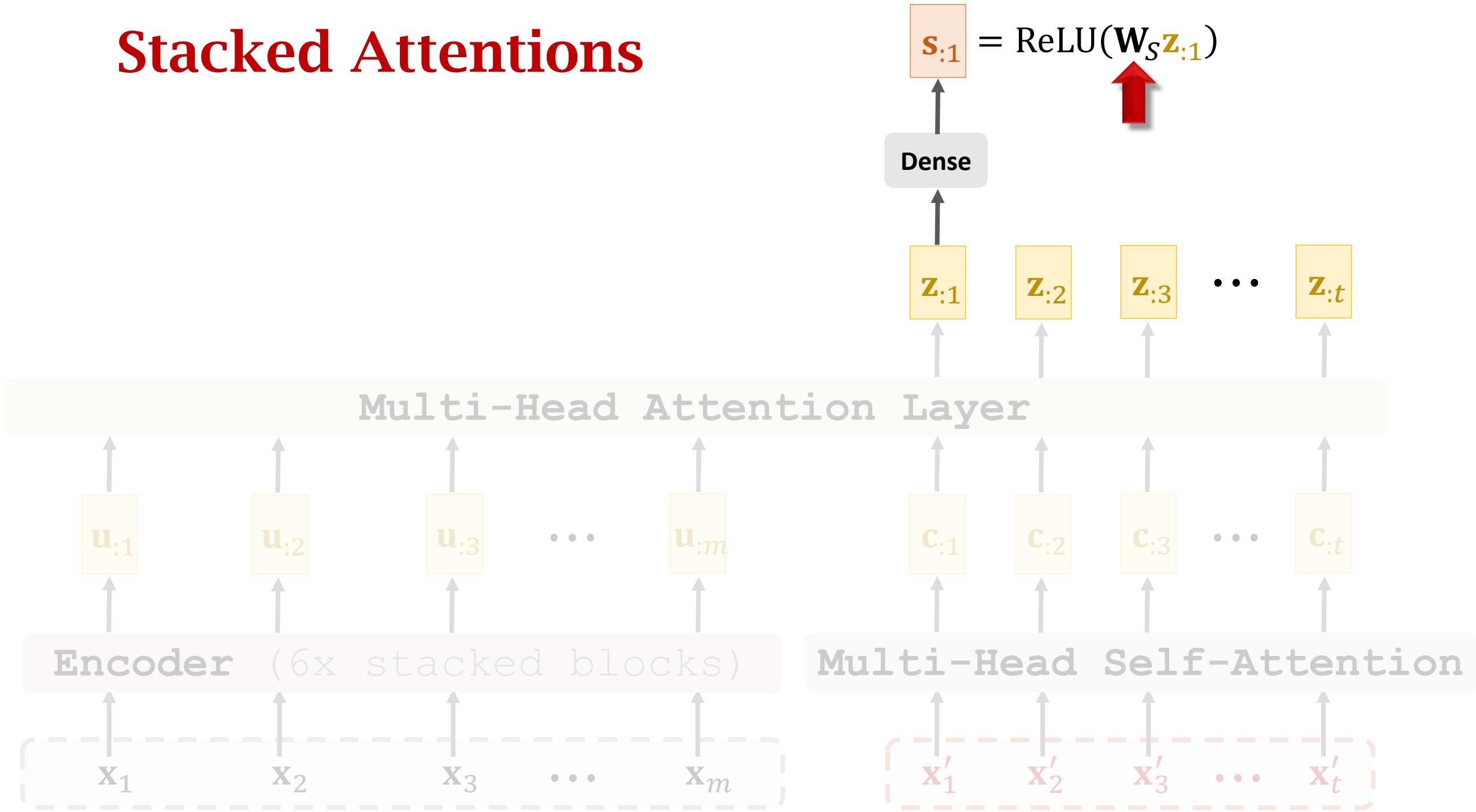
Stacked Attentions



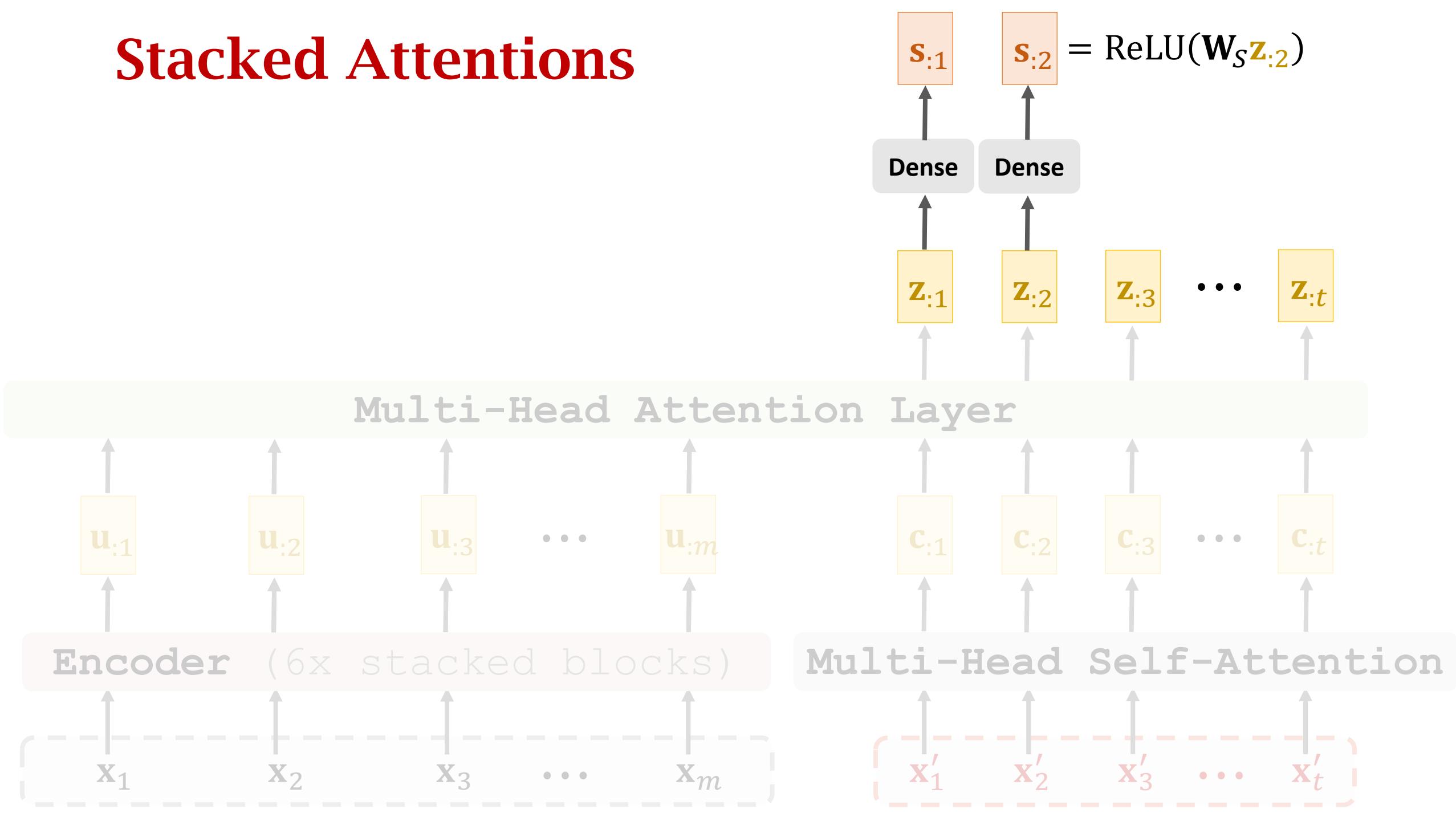
Stacked Attentions



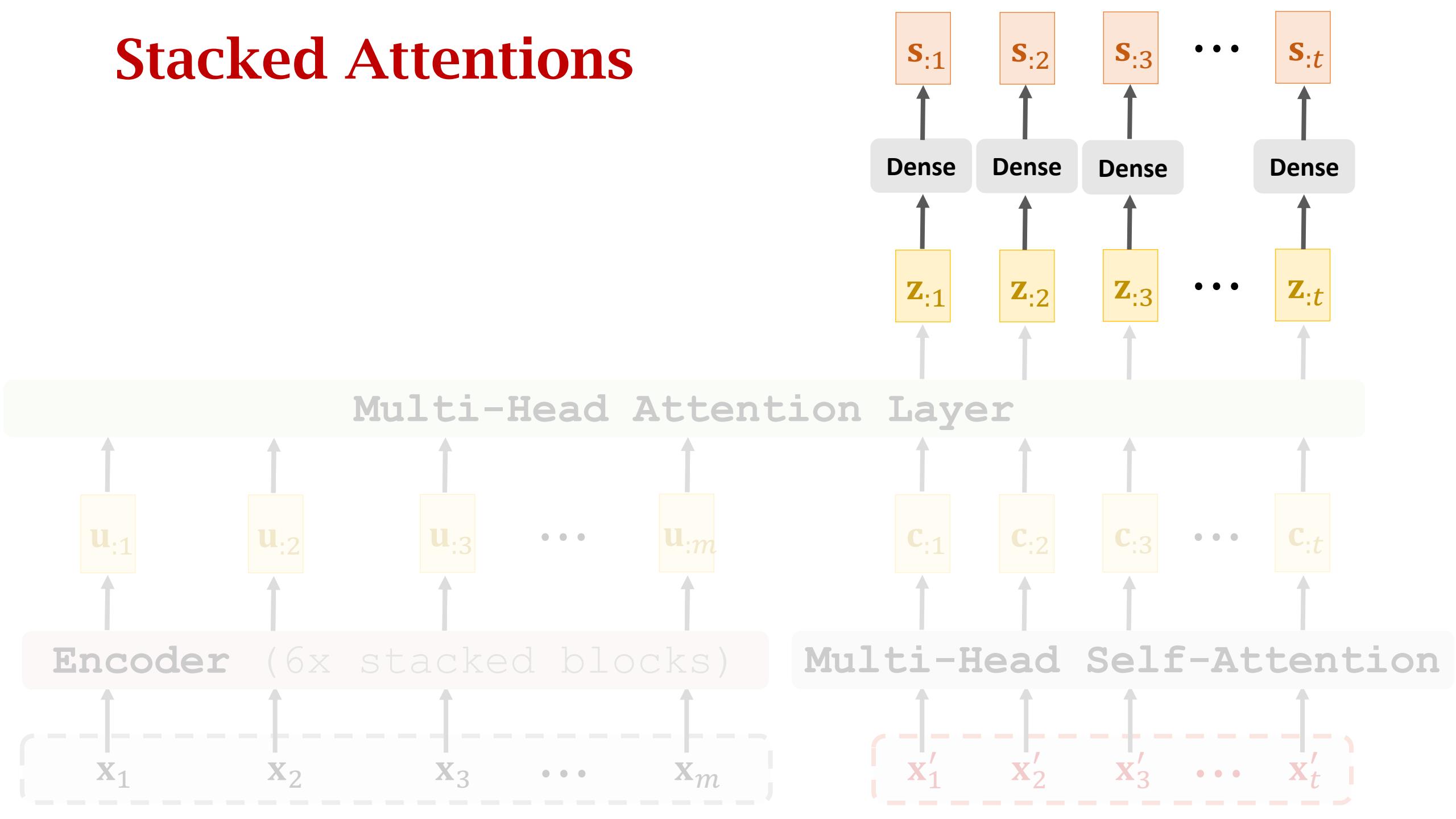
Stacked Attentions



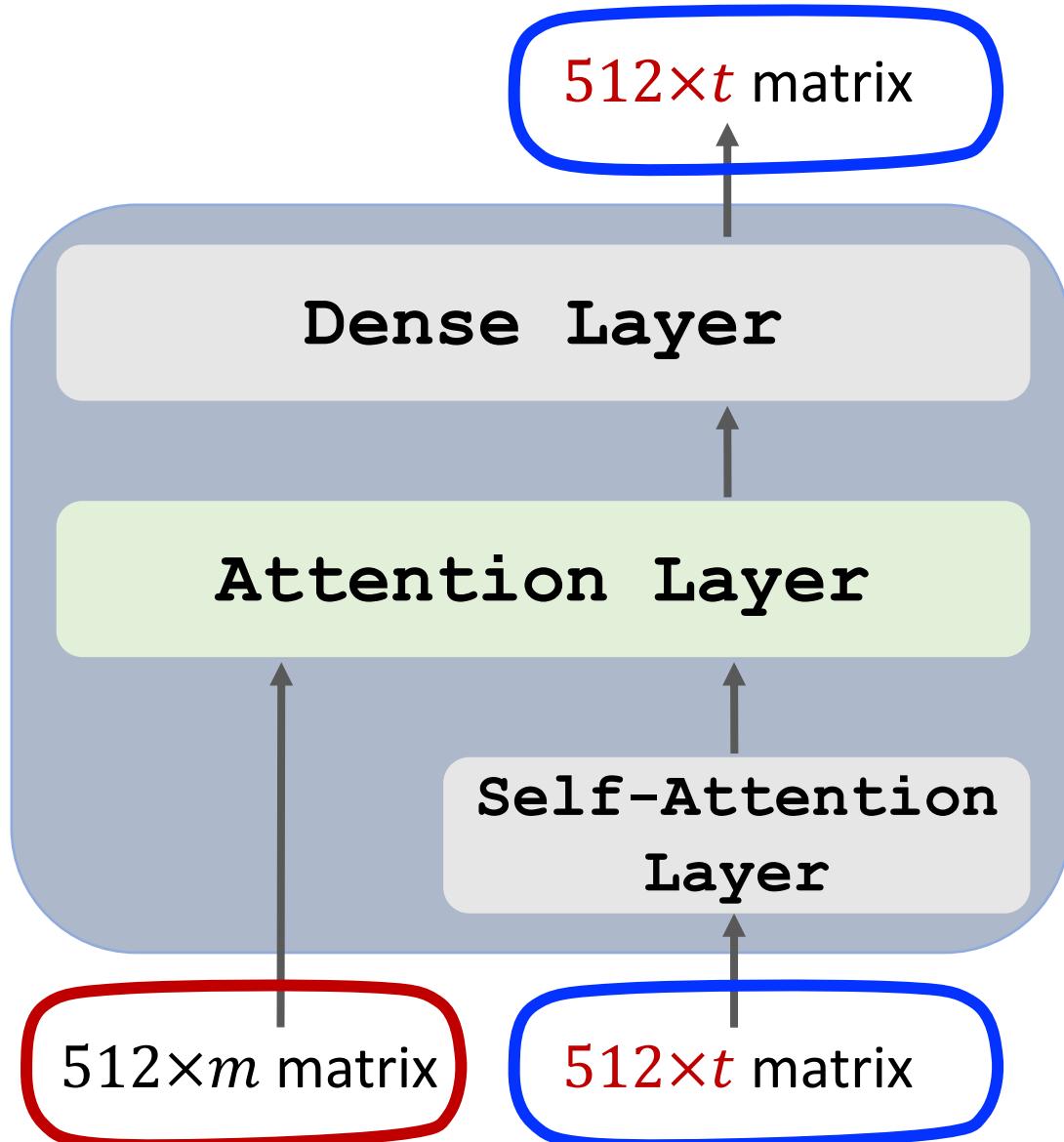
Stacked Attentions



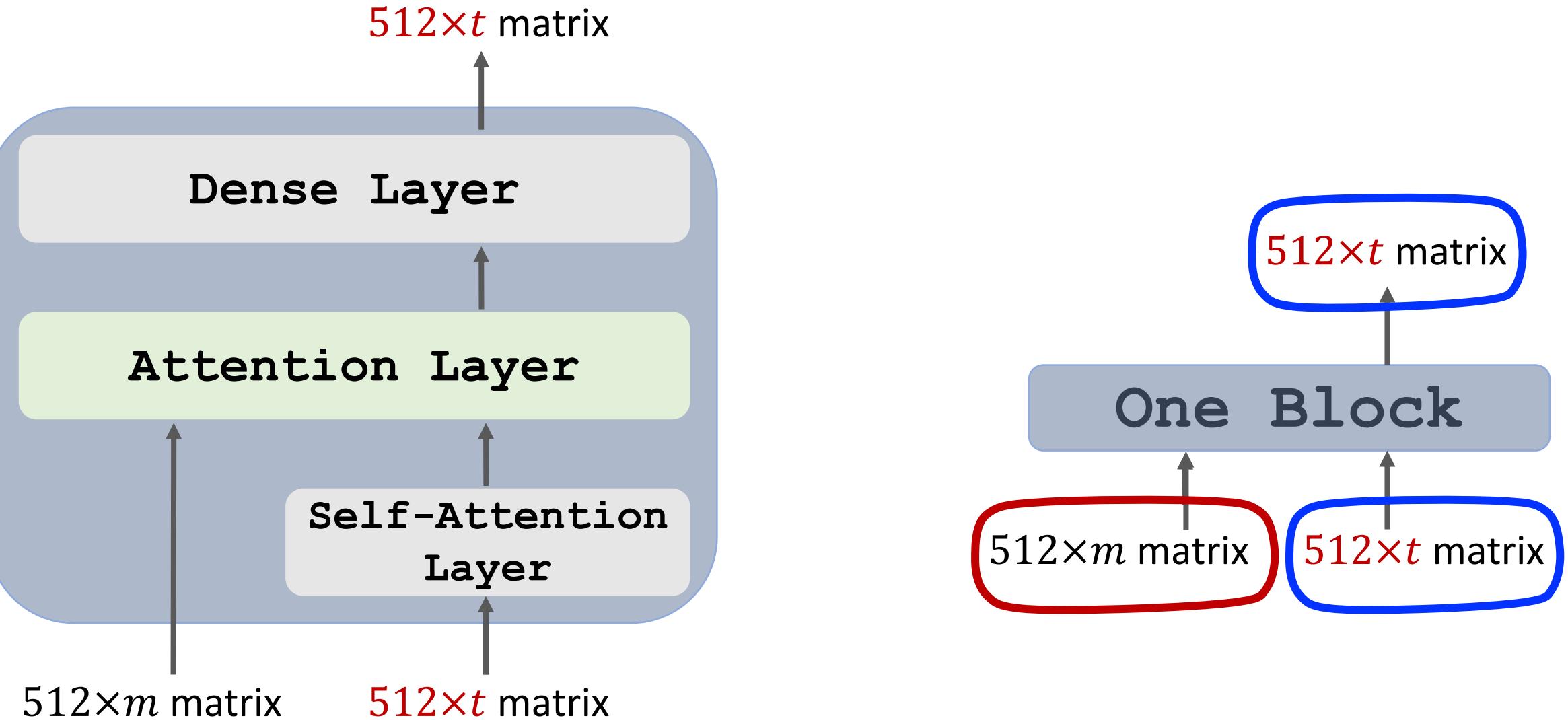
Stacked Attentions



Transformer's Decoder : One Block

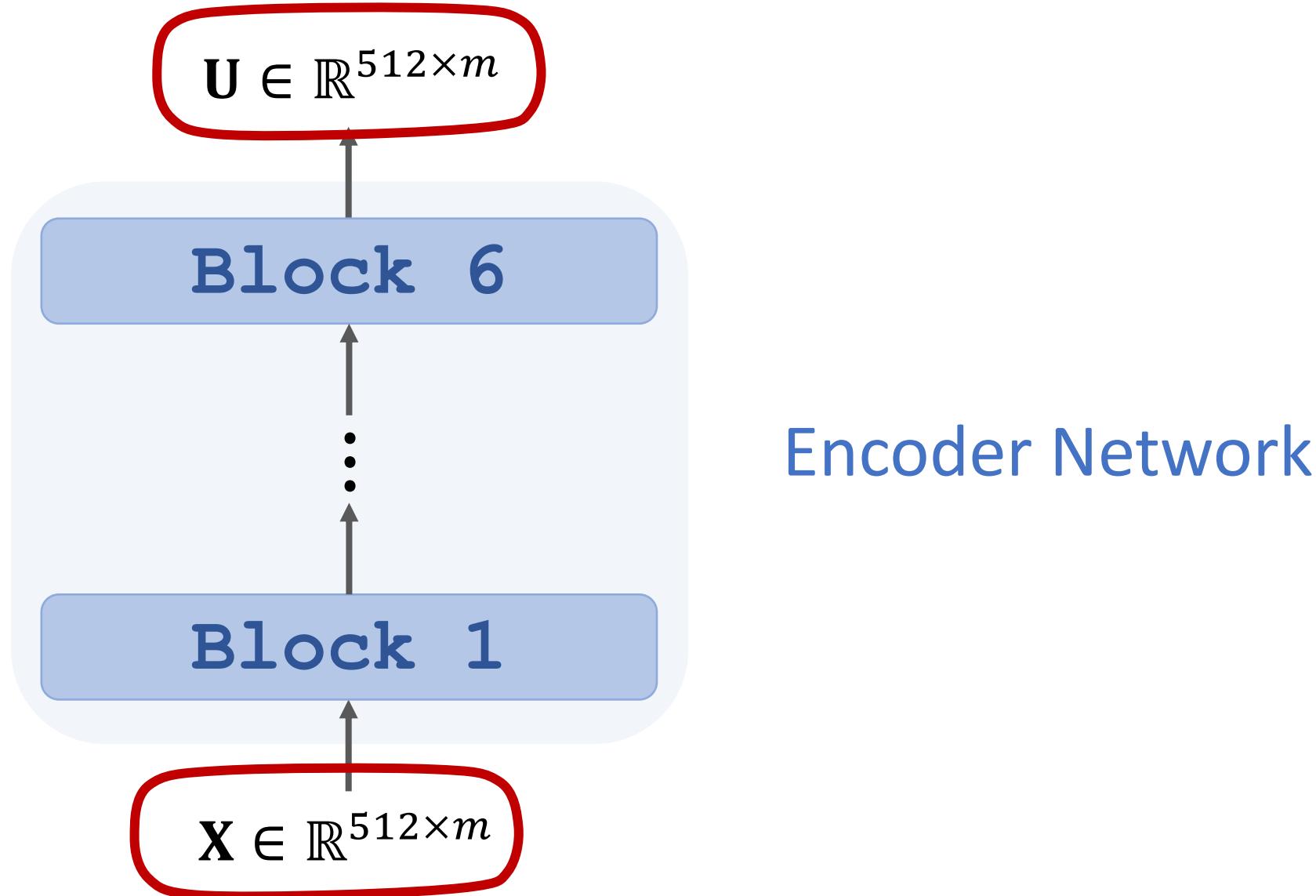


Transformer's Decoder : One Block

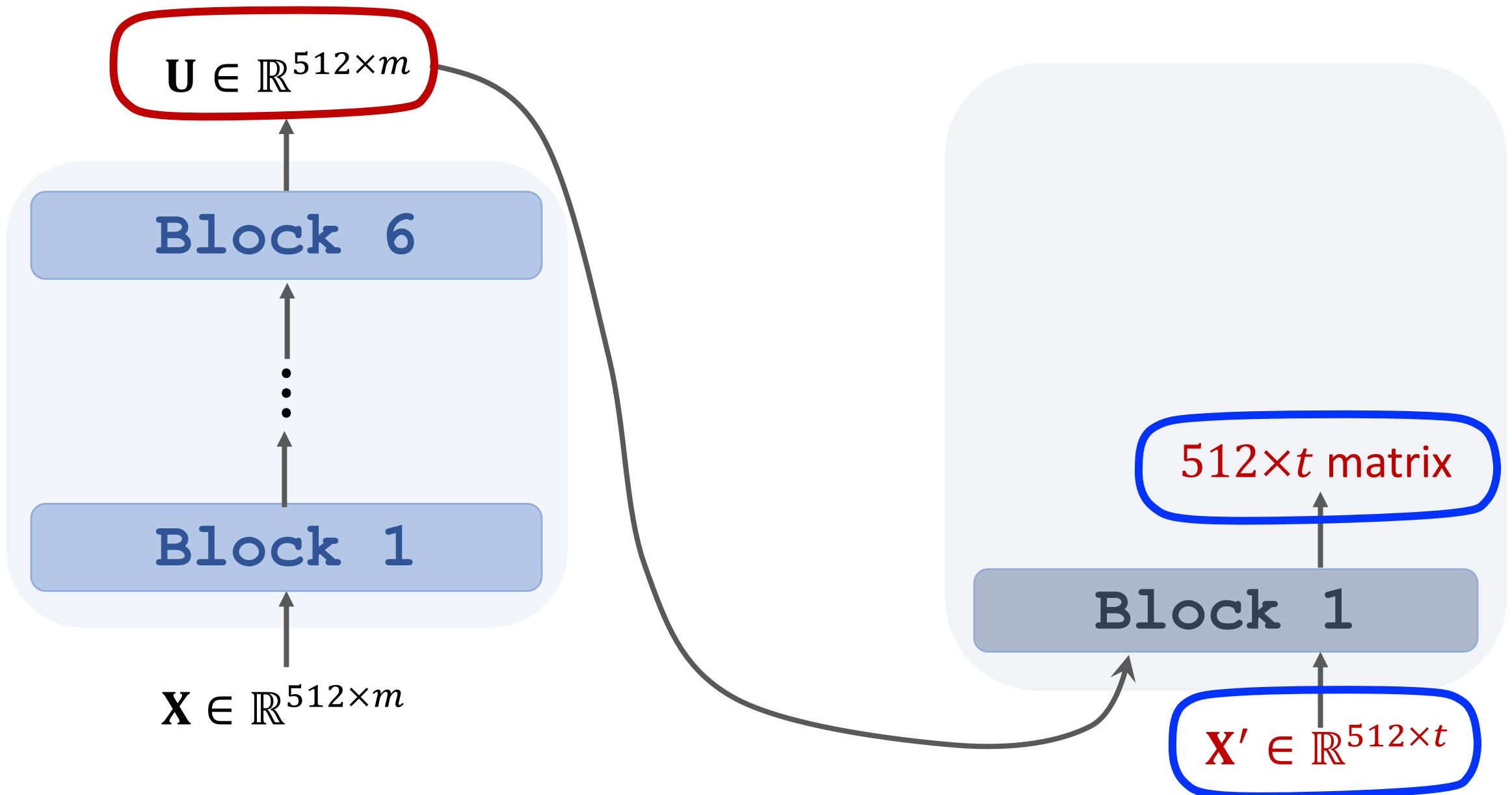


Put Everything Together

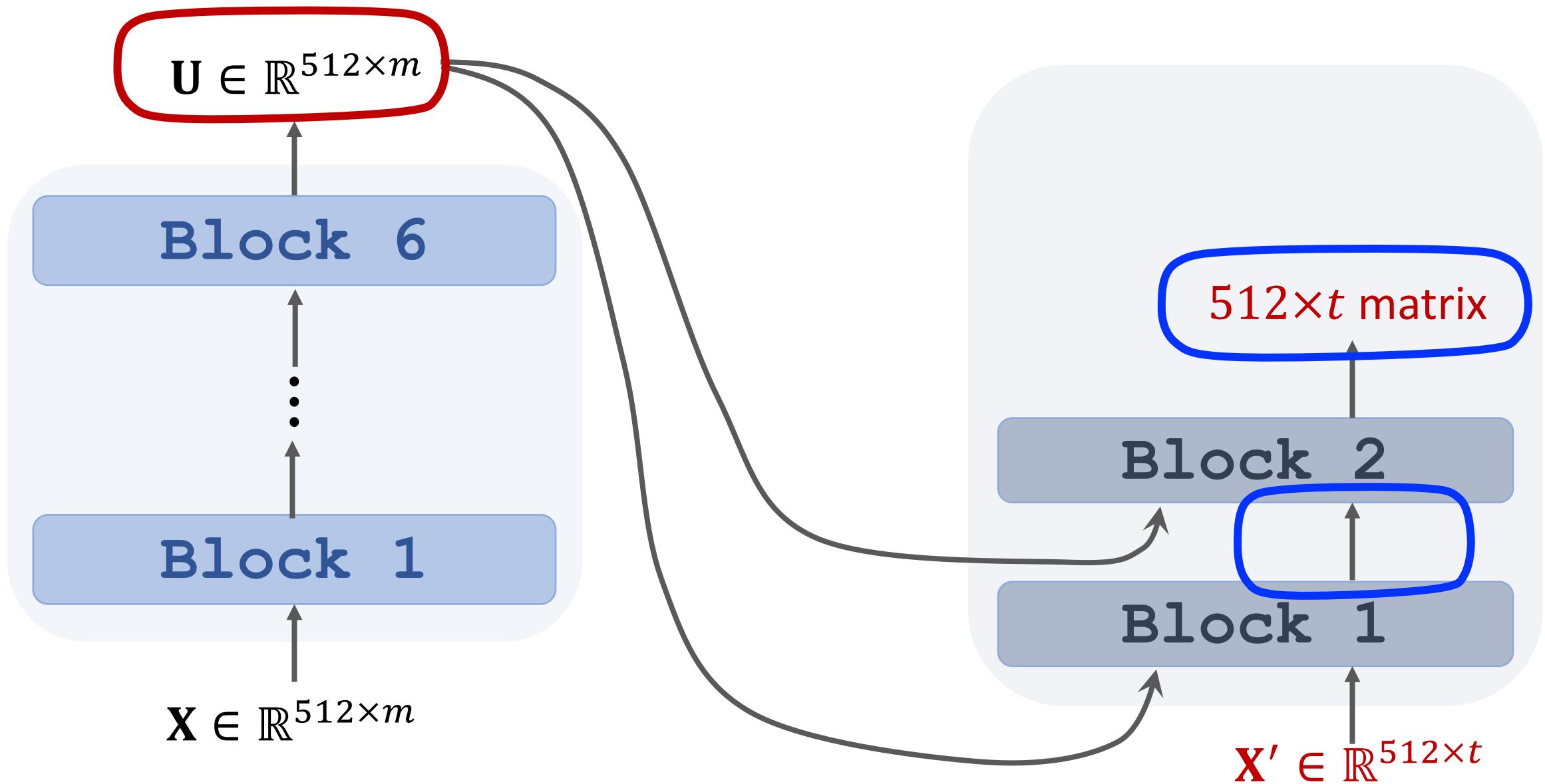
Transformer



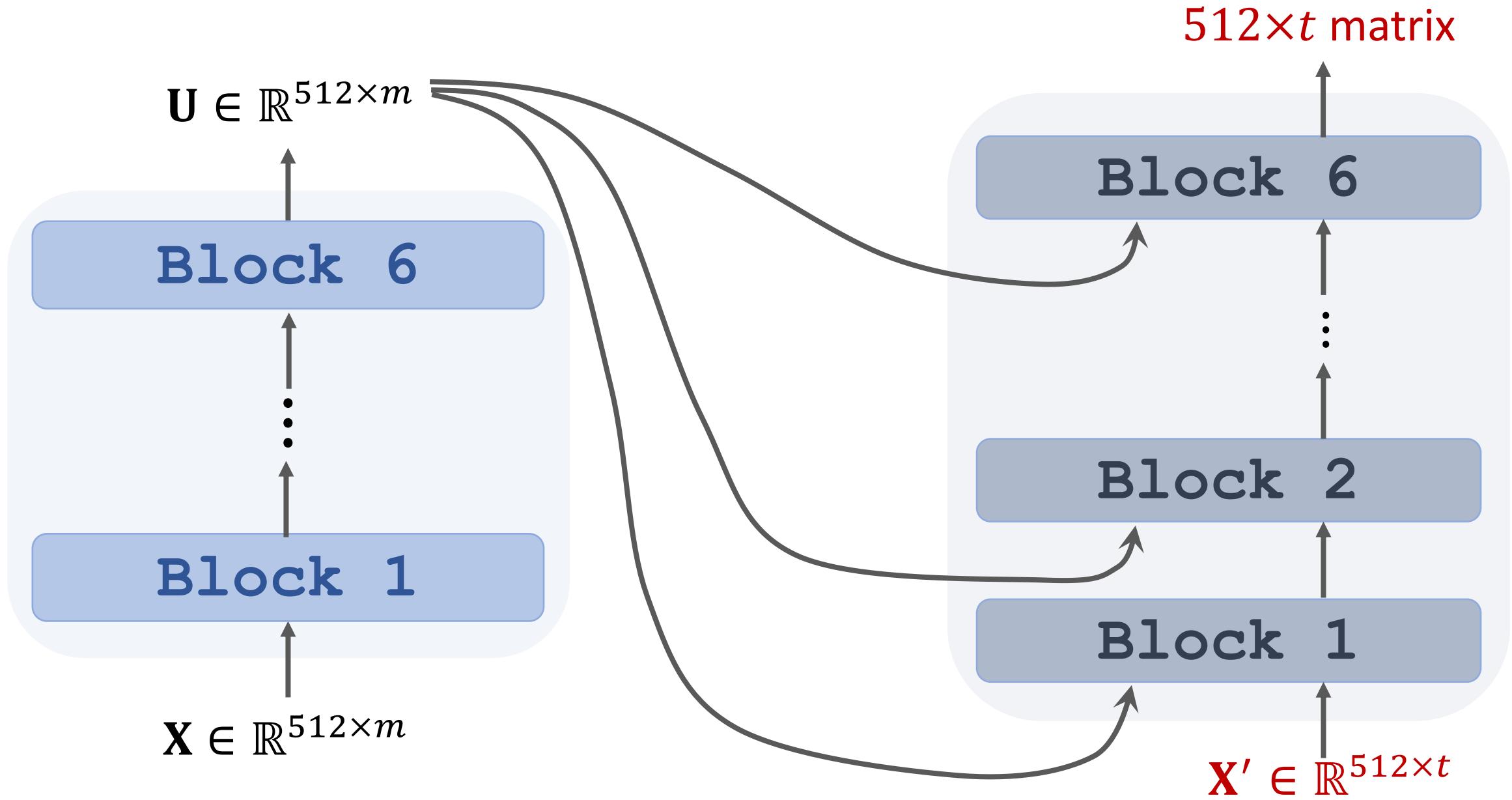
Transformer



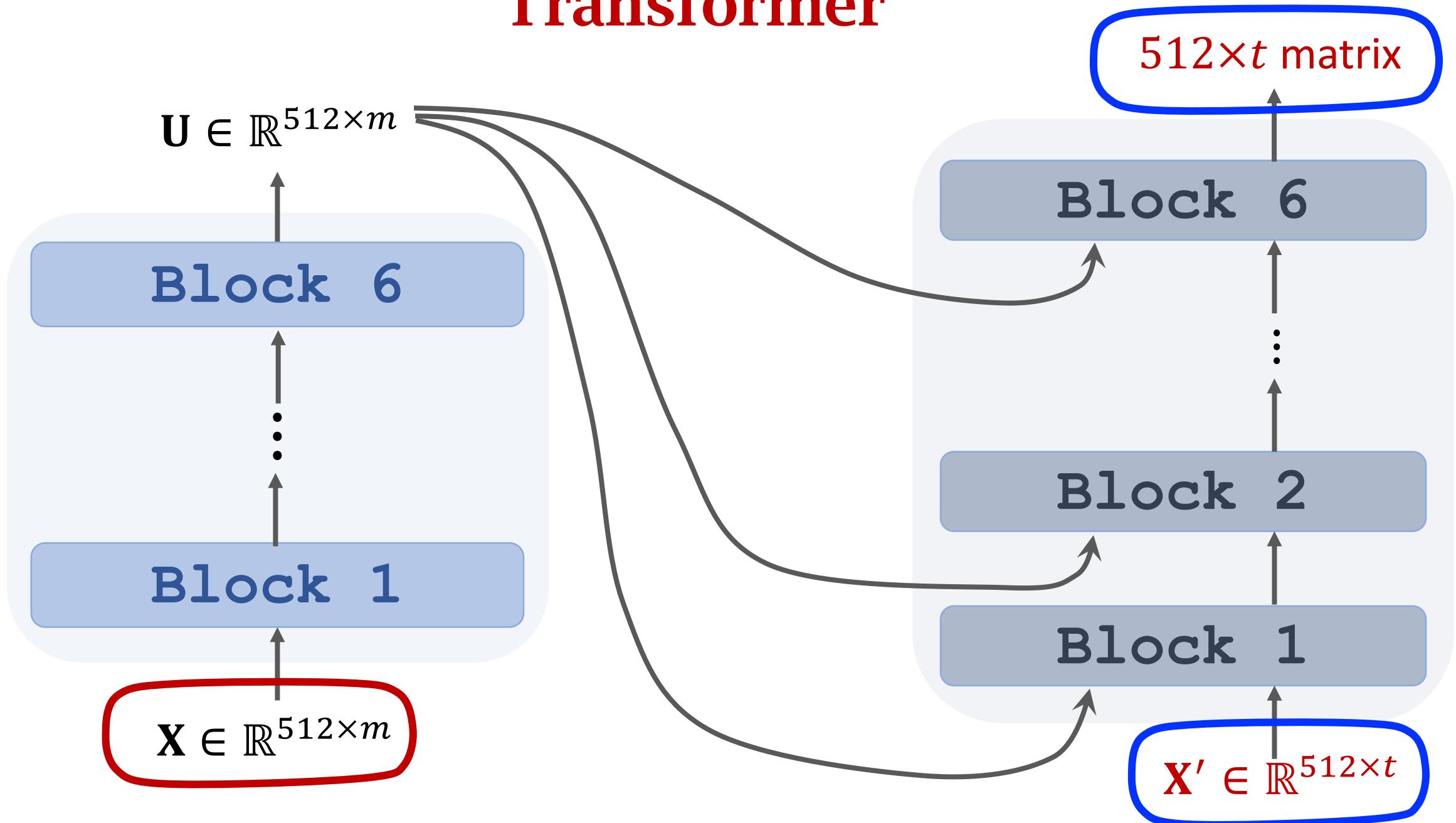
Transformer



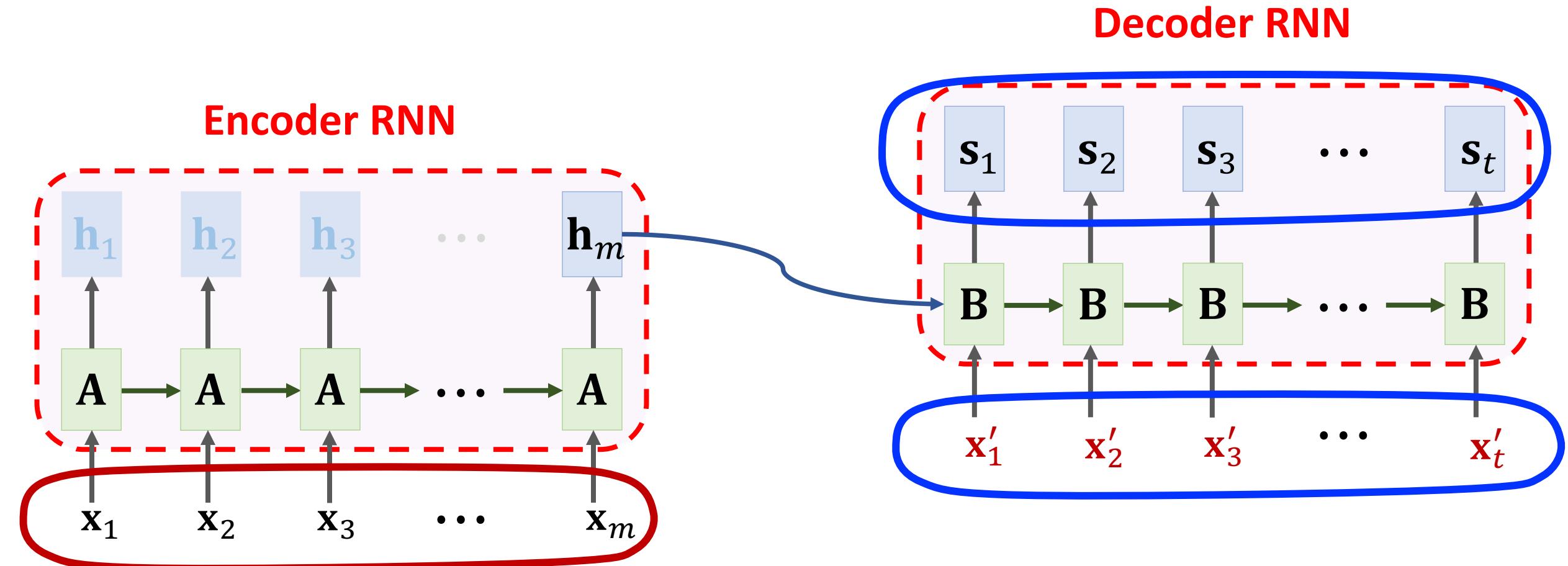
Transformer



Transformer



Comparison with RNN Seq2Seq Model



Applied to Machine Translation

Example: English to German

$$\mathbf{U} \in \mathbb{R}^{512 \times m}$$



Encoder



$$\mathbf{X} \in \mathbb{R}^{512 \times m}$$

Example: English to German

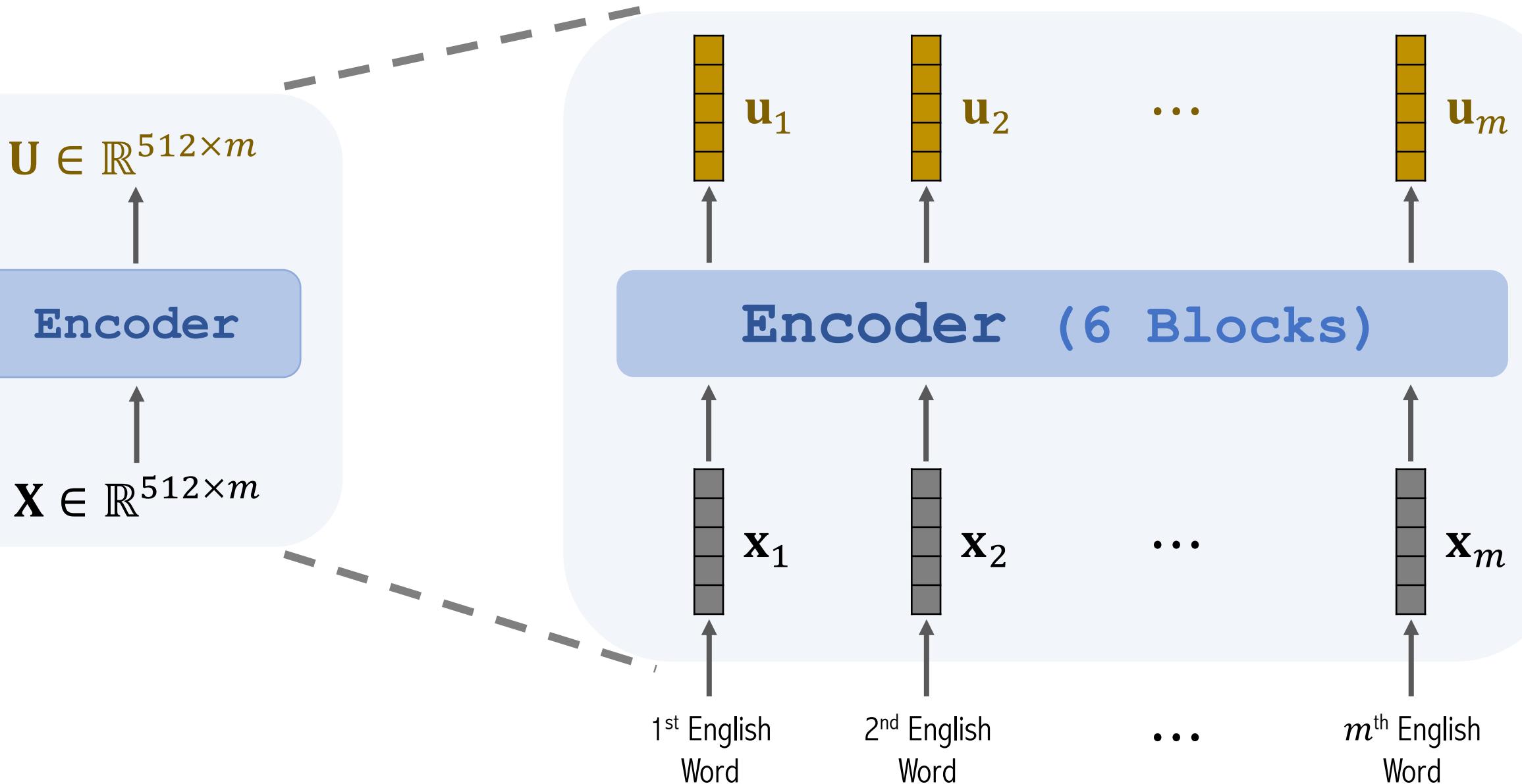
$$\mathbf{U} \in \mathbb{R}^{512 \times m}$$

Encoder

$$\mathbf{X} \in \mathbb{R}^{512 \times m}$$

Encoder (6 Blocks)

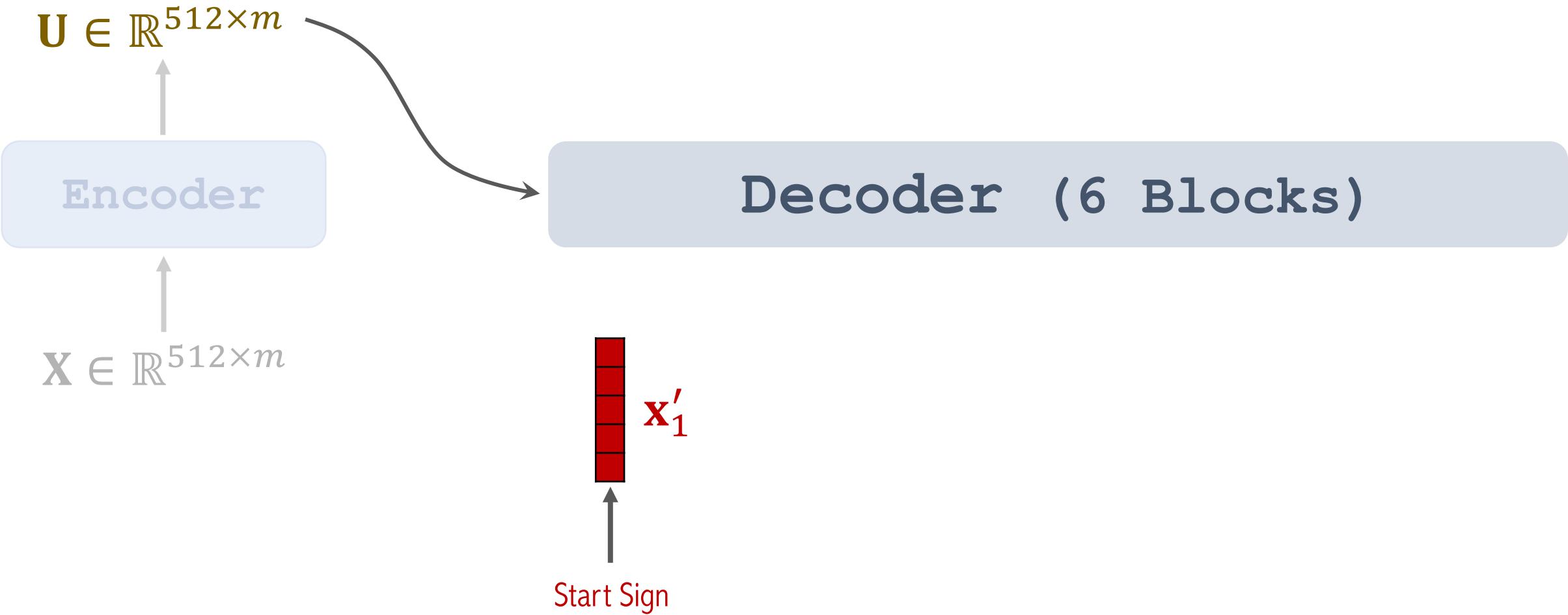
Example: English to German



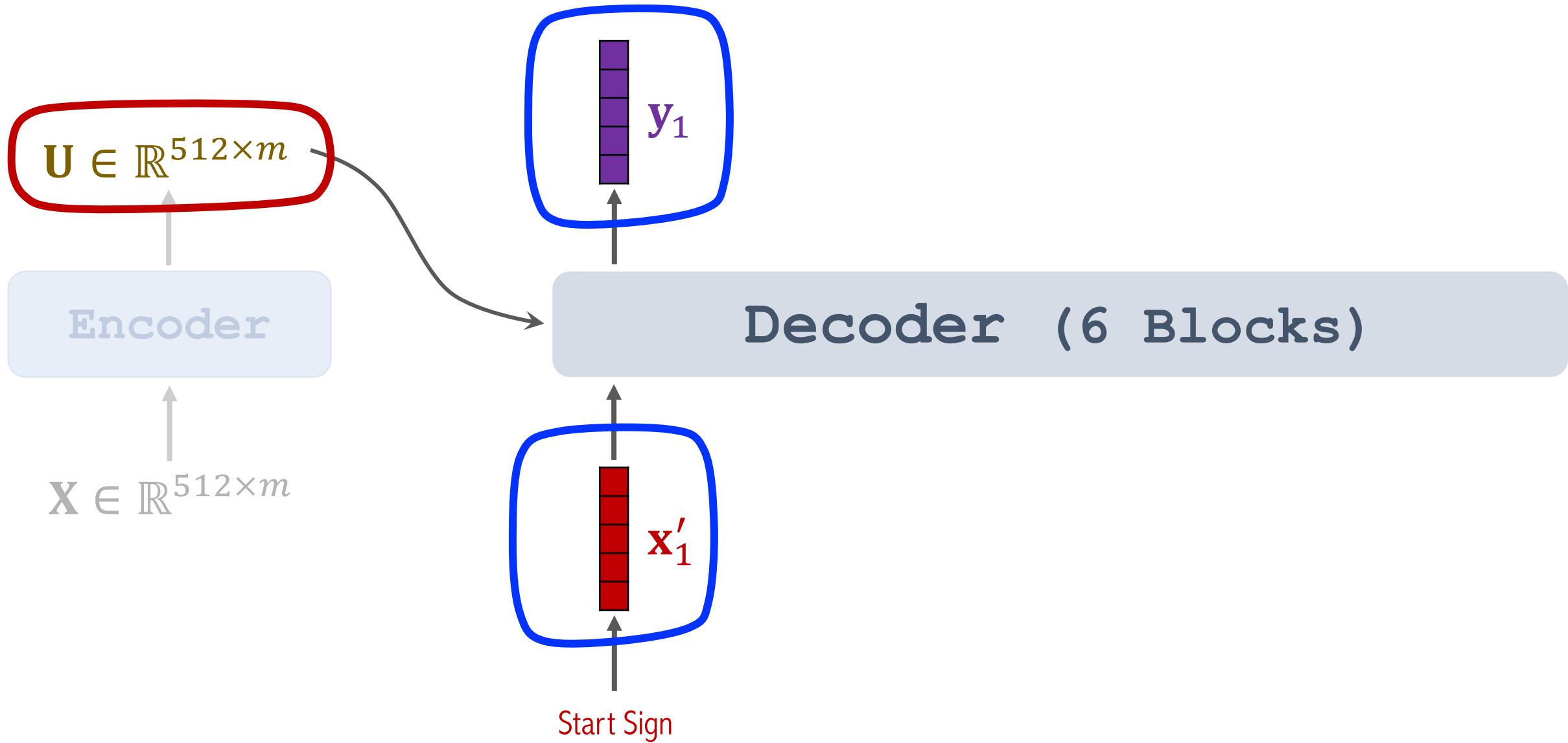
Example: English to German



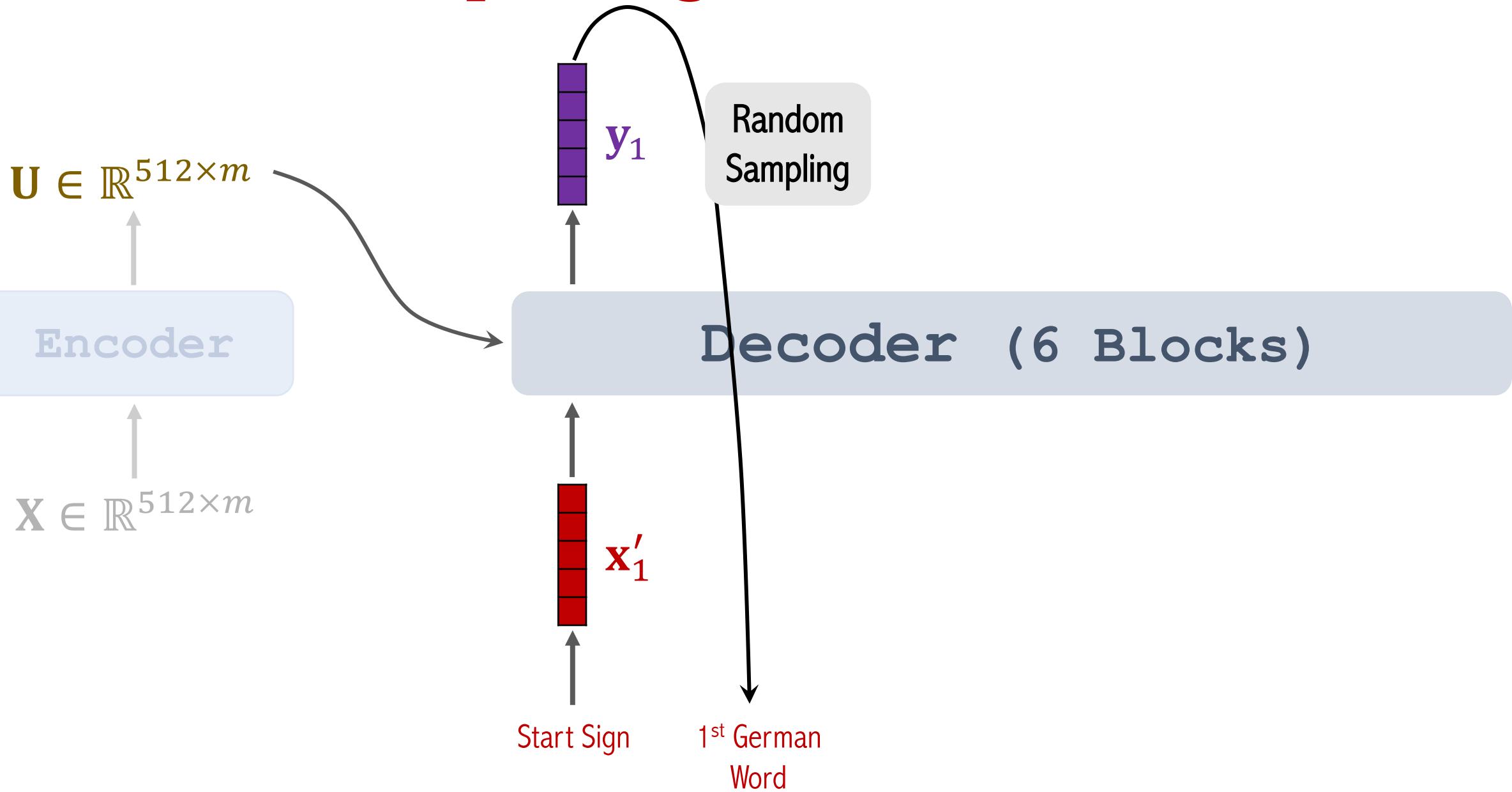
Example: English to German



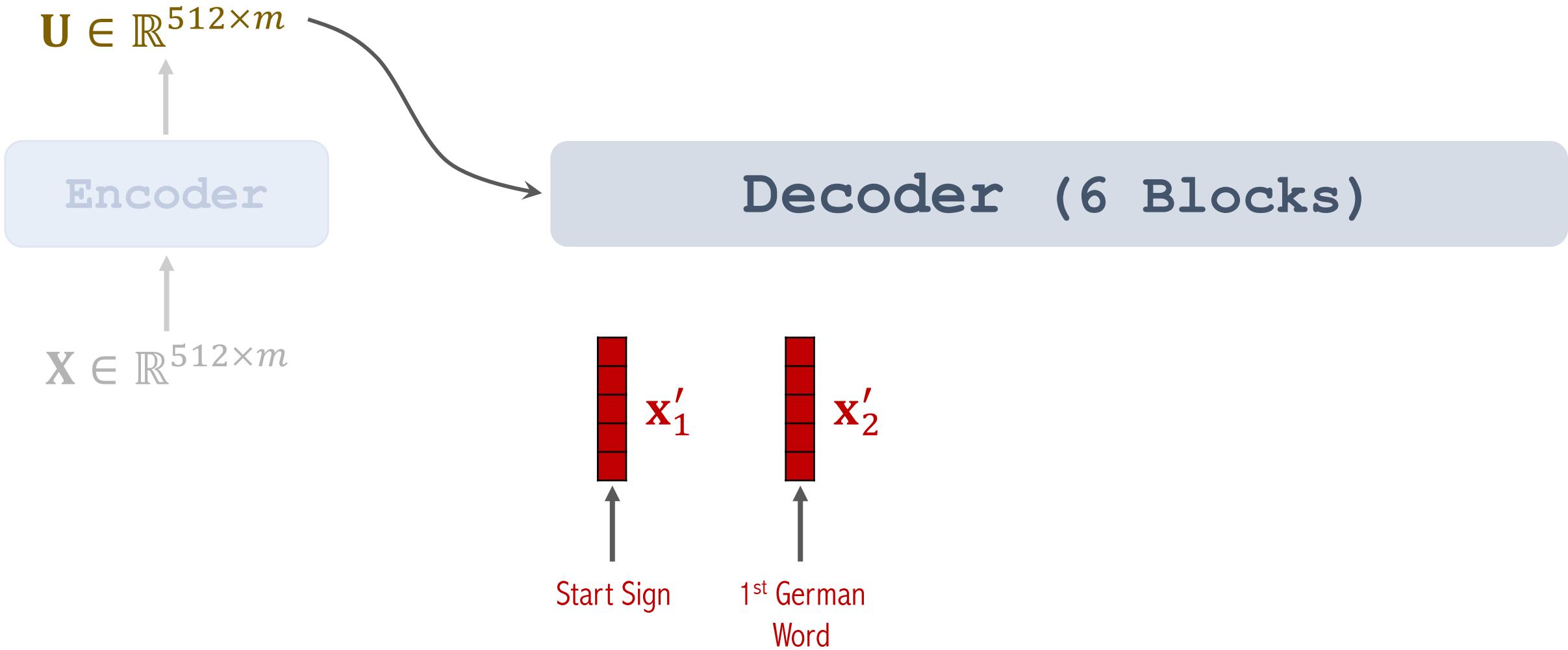
Example: English to German



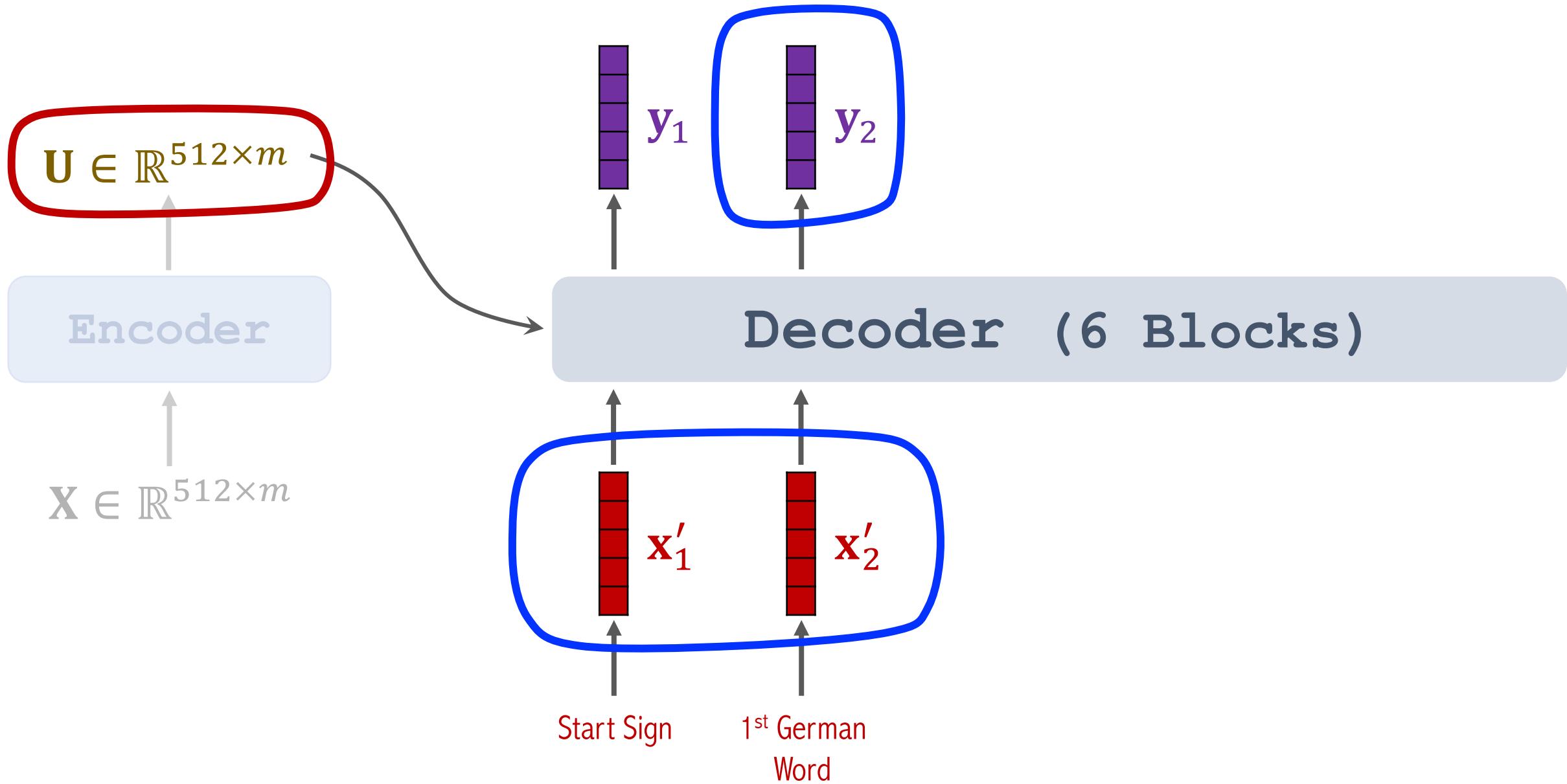
Example: English to German



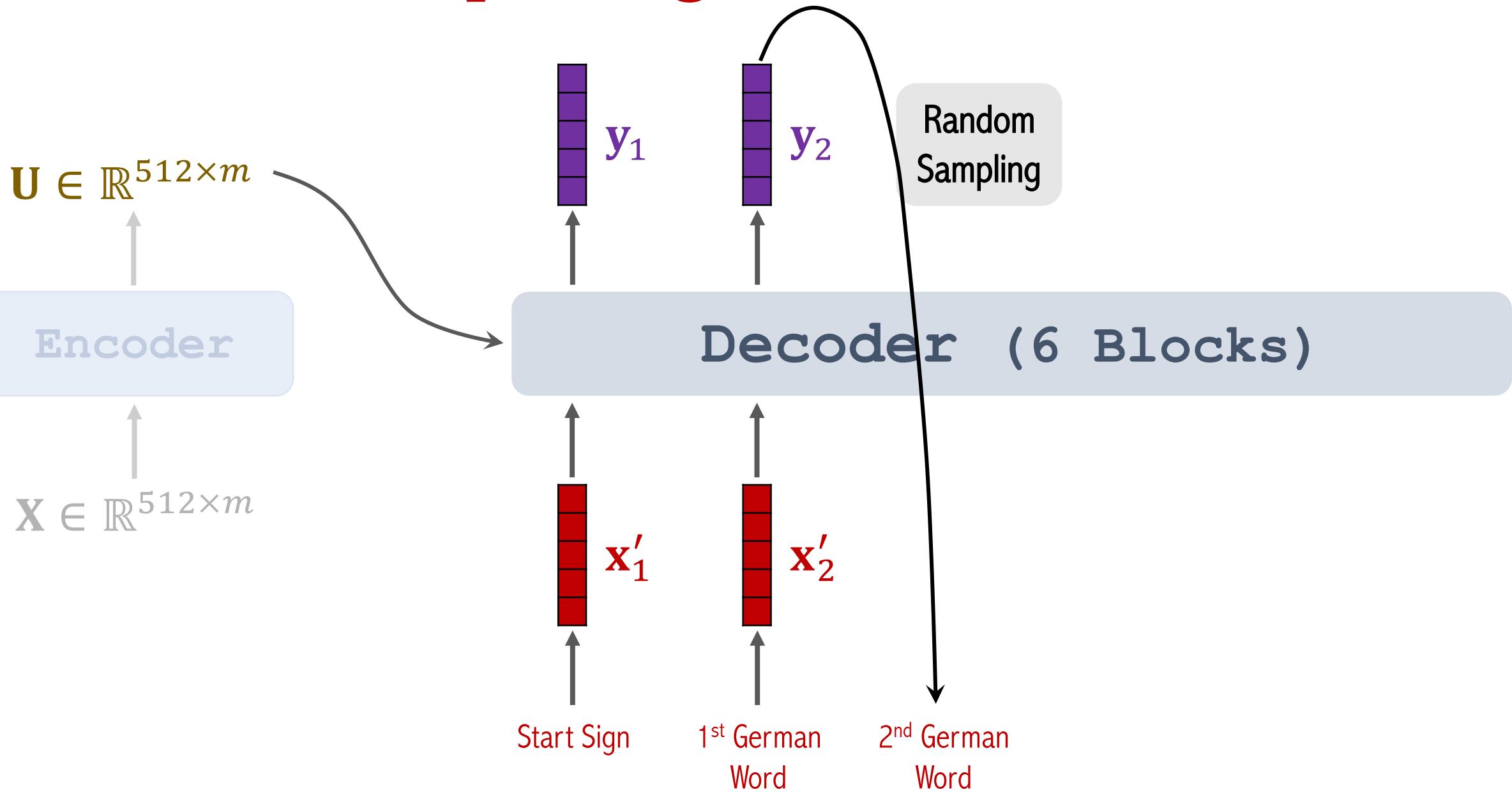
Example: English to German



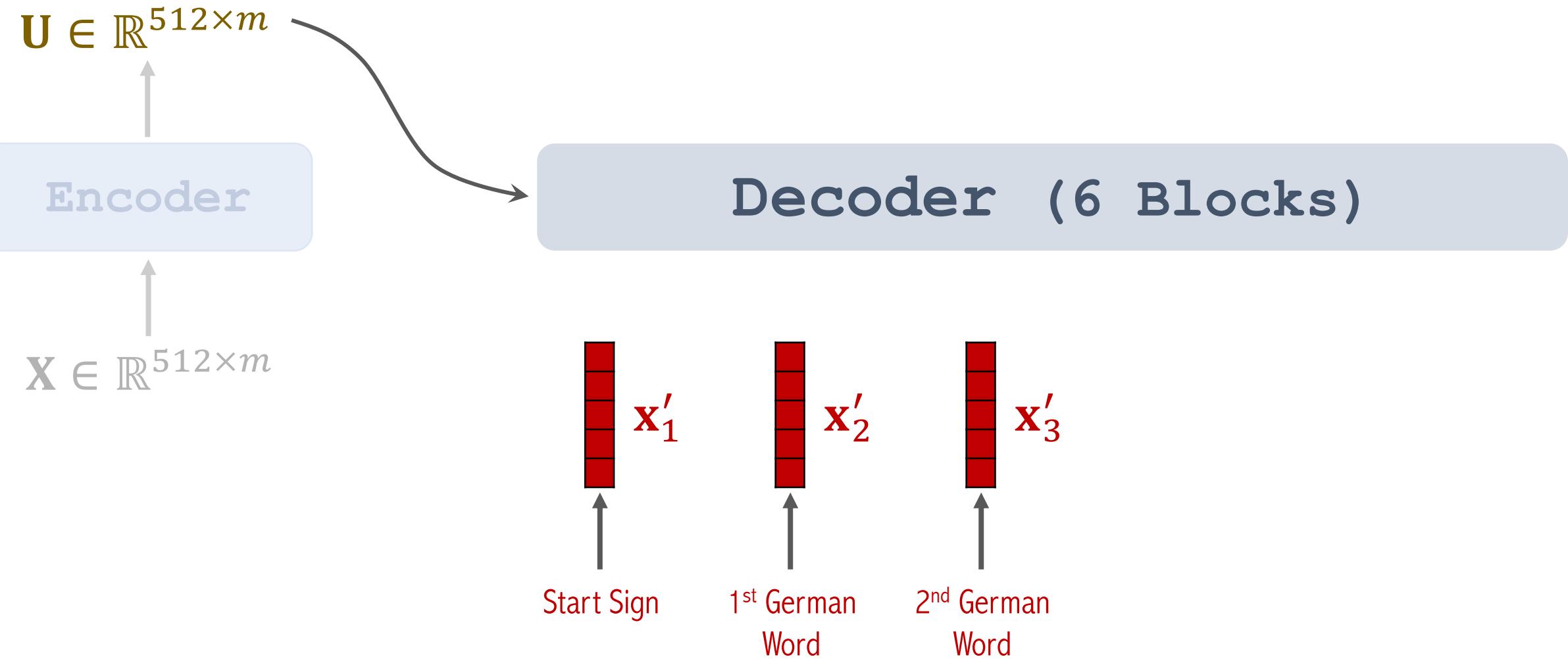
Example: English to German



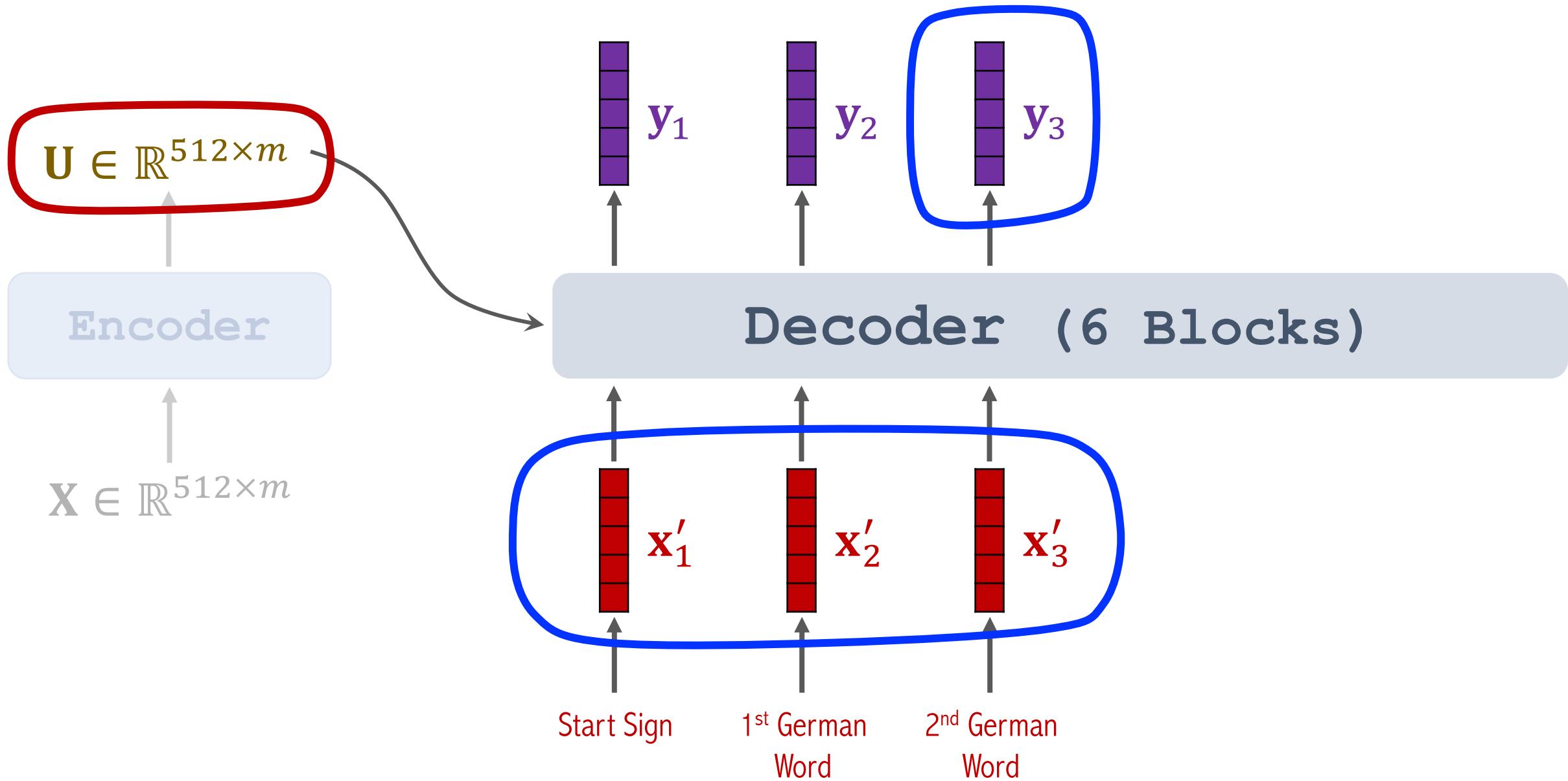
Example: English to German



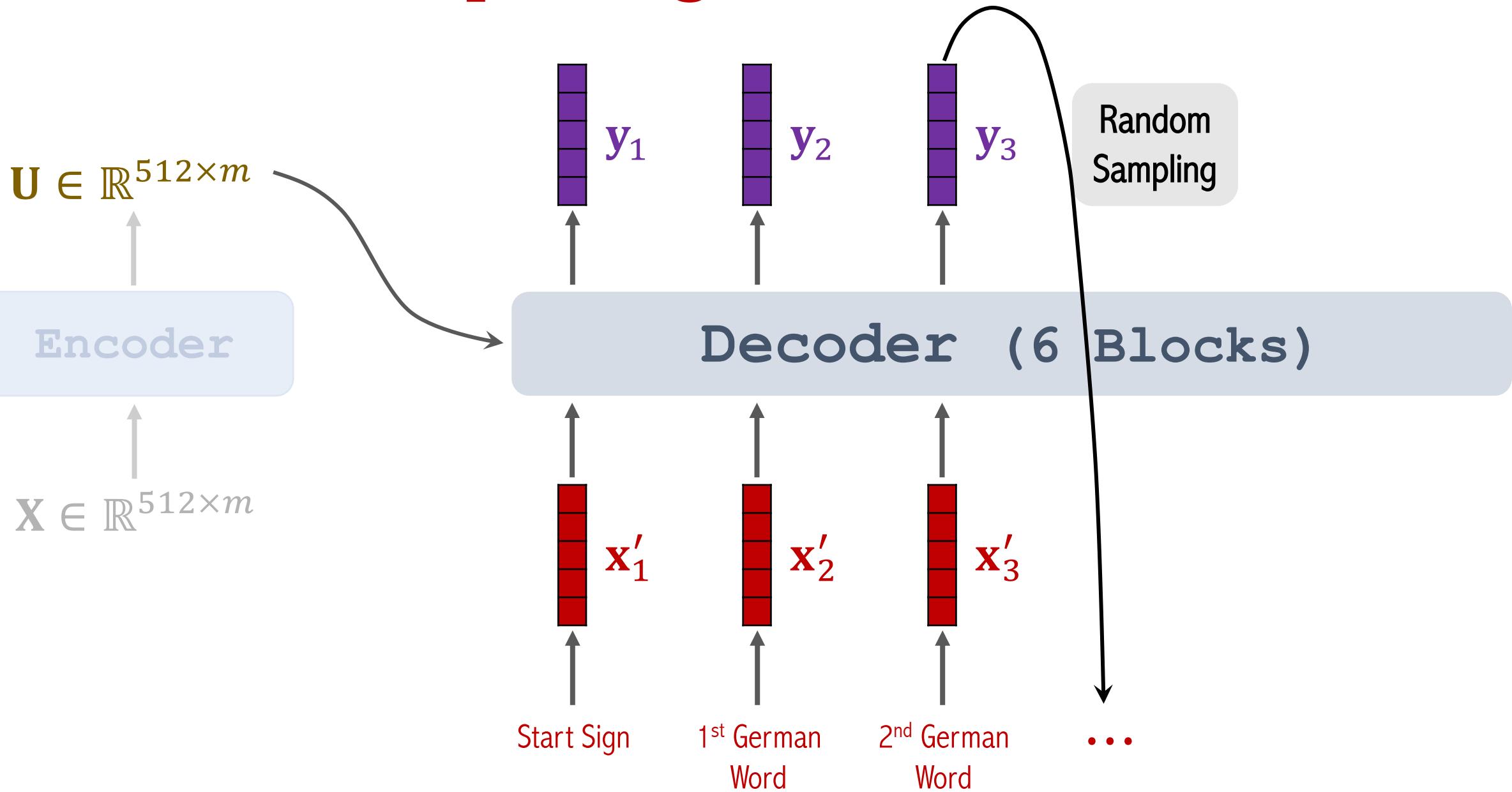
Example: English to German



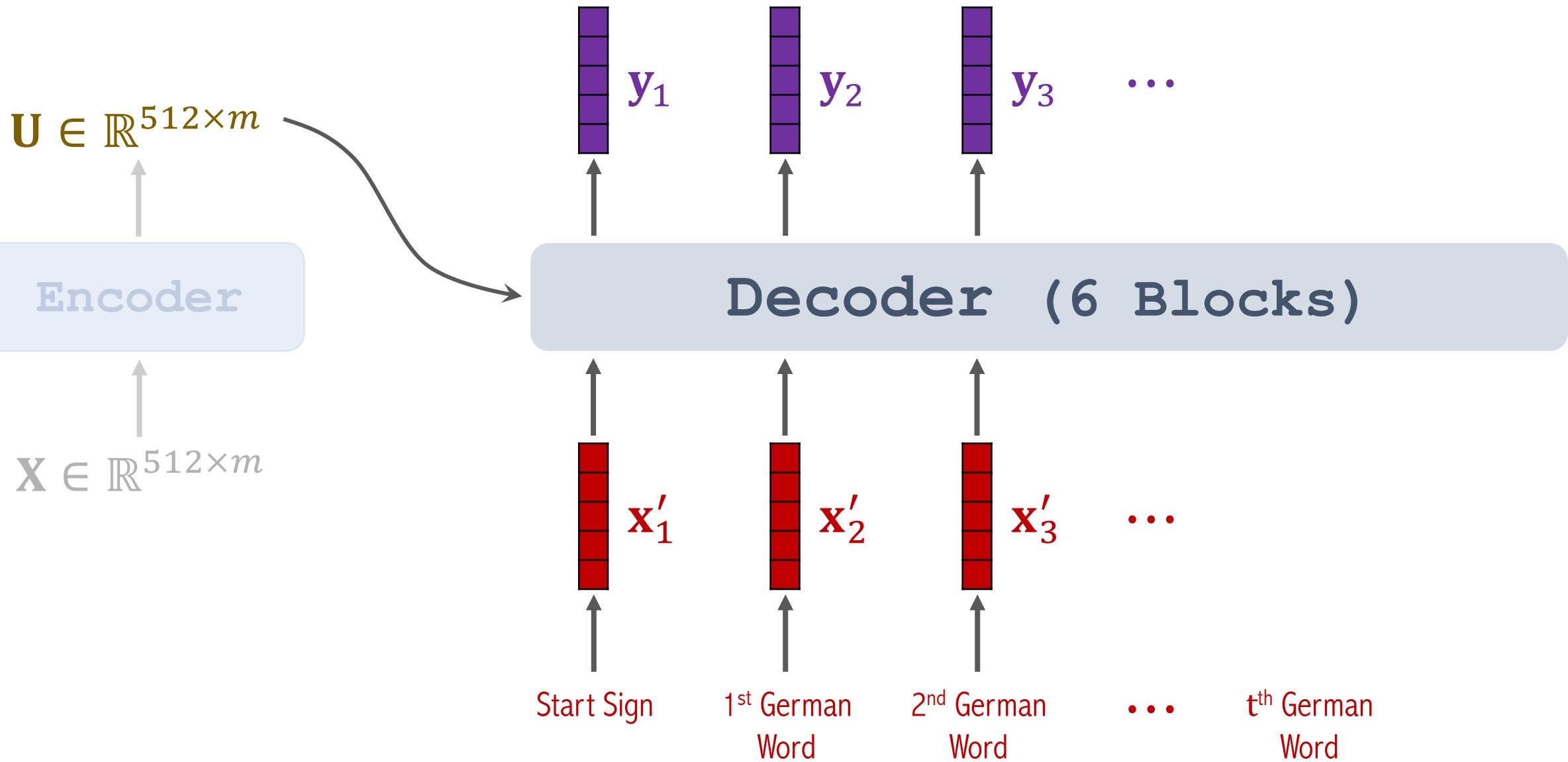
Example: English to German



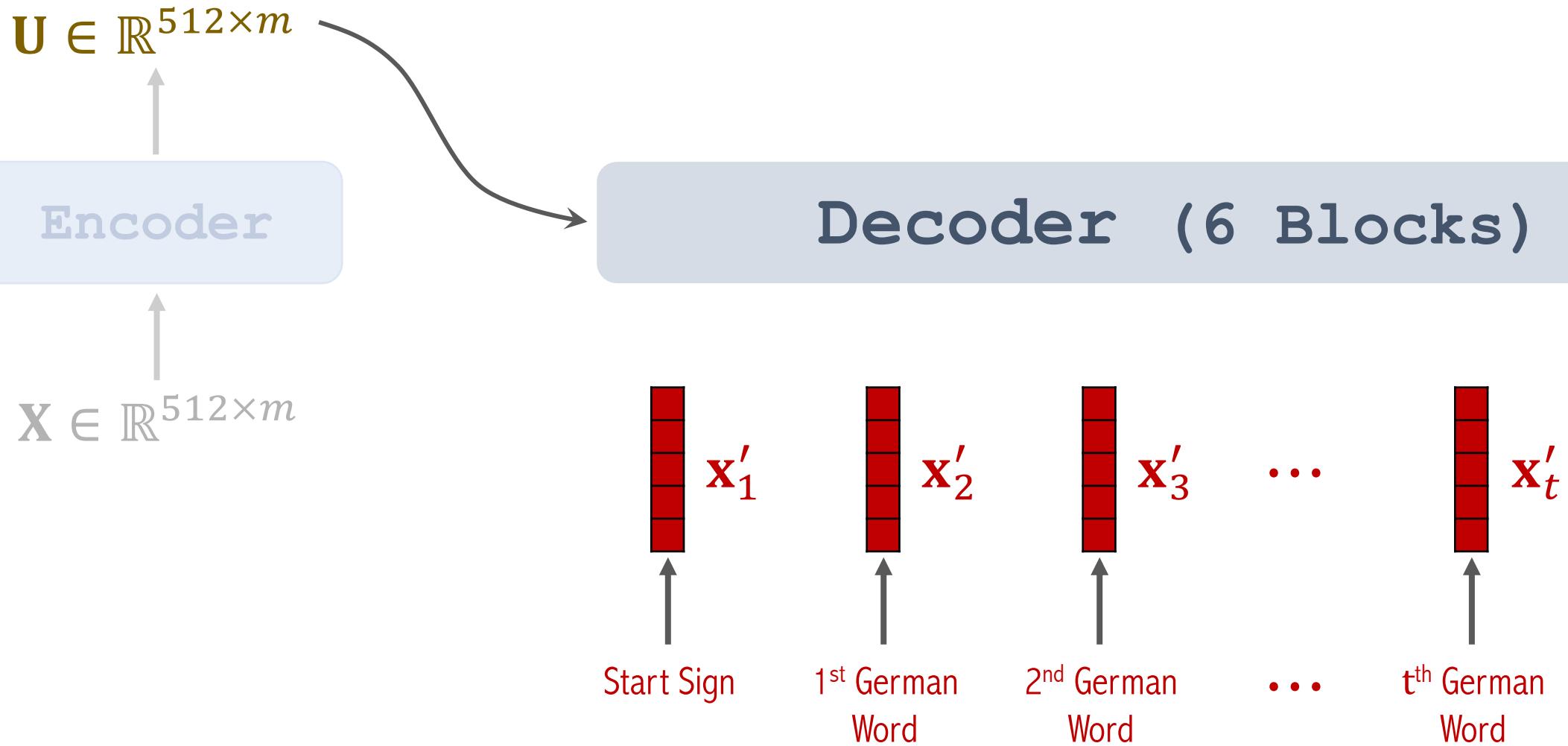
Example: English to German



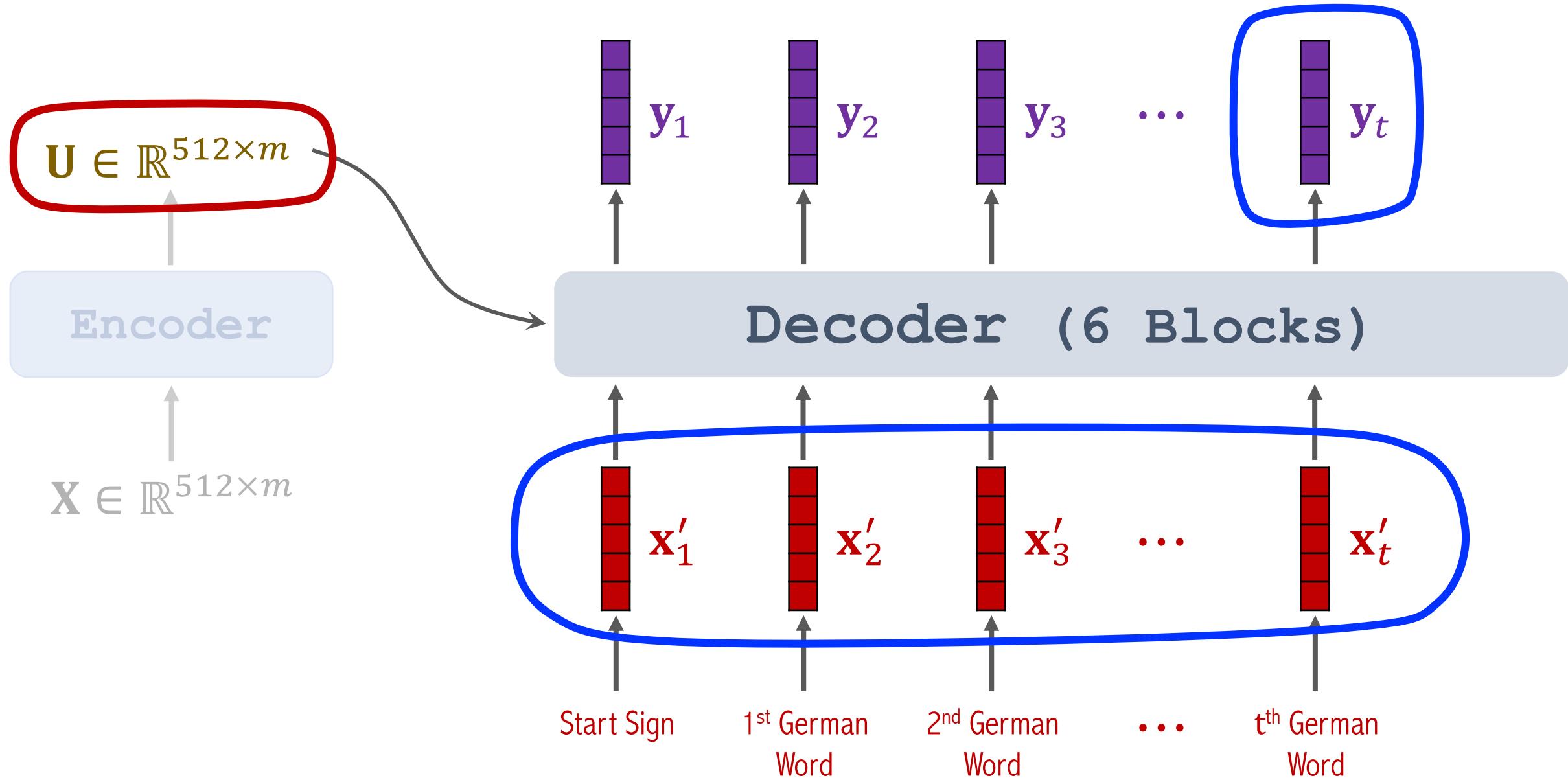
Example: English to German



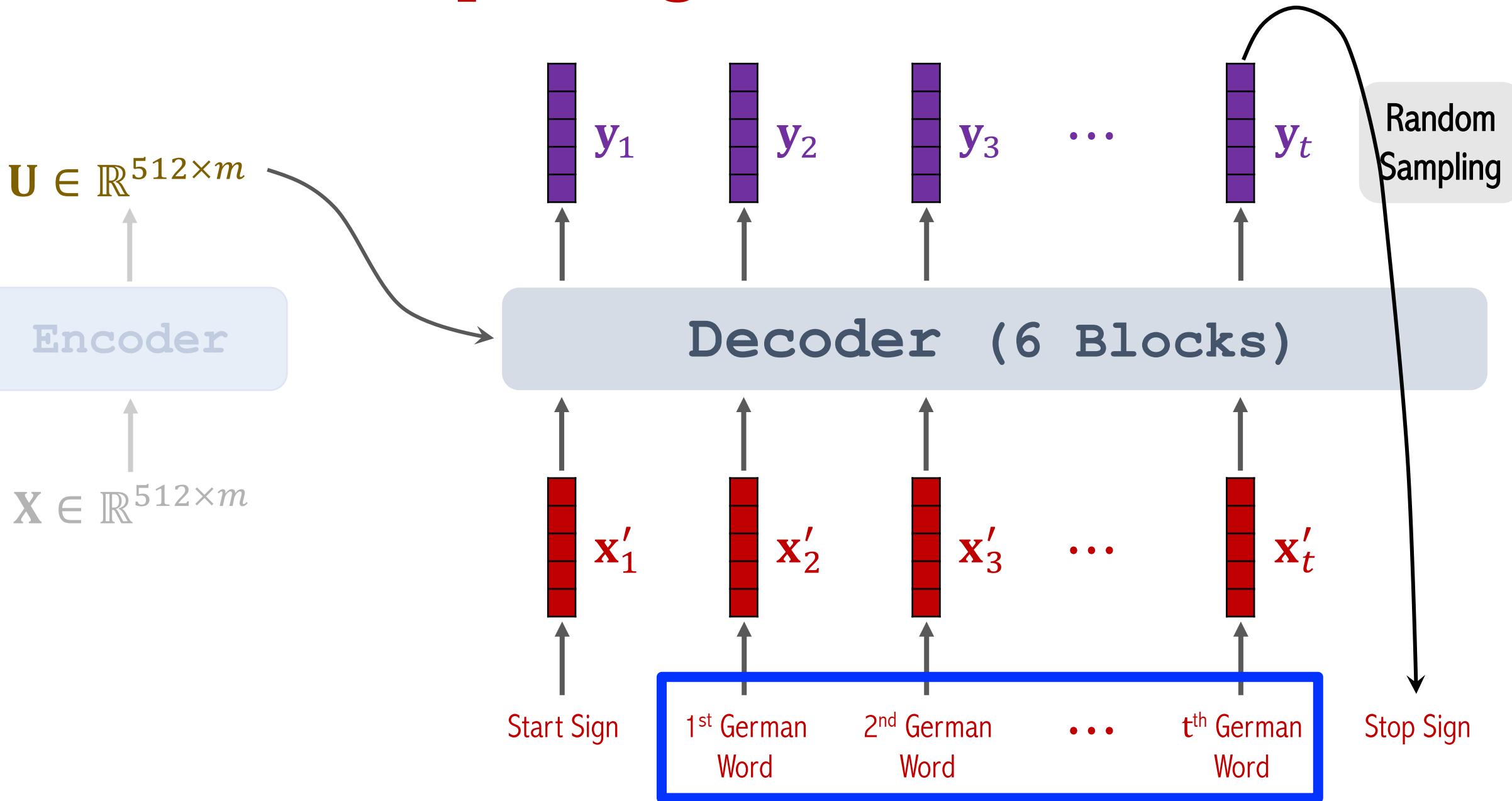
Example: English to German



Example: English to German



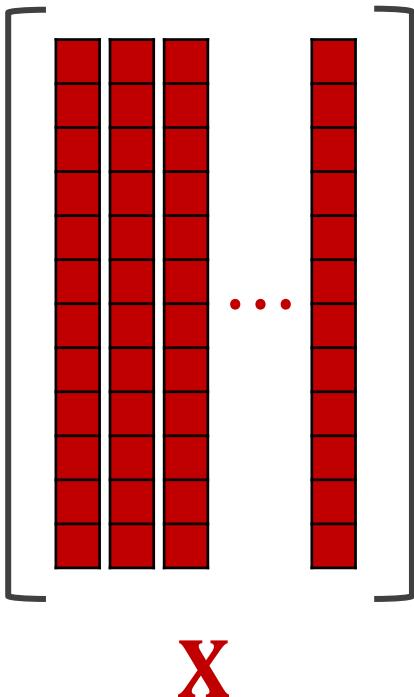
Example: English to German



Summary

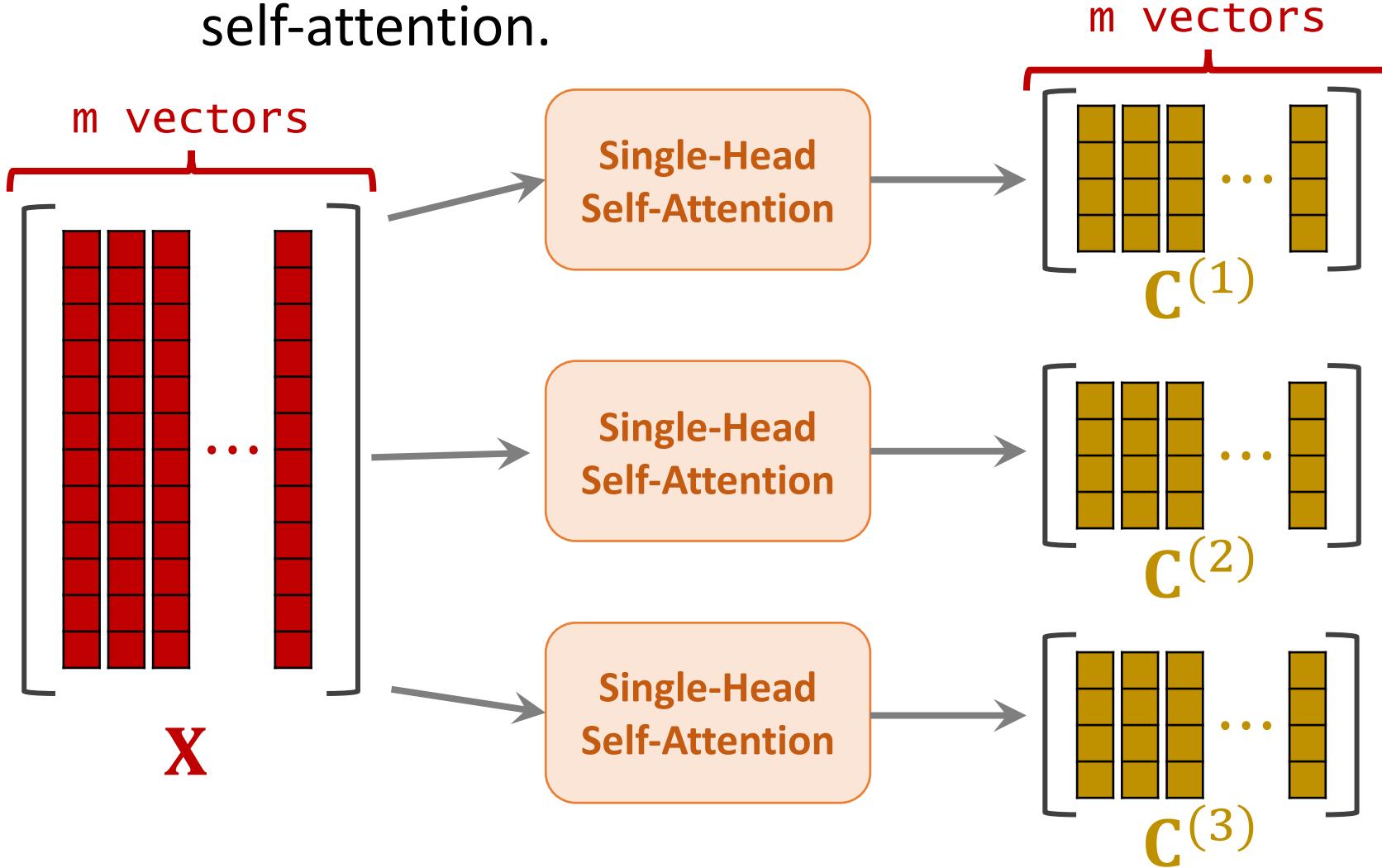
From Single-Head to Multi-Head

- Single-head self-attentions can be combined to form a multi-head self-attention.



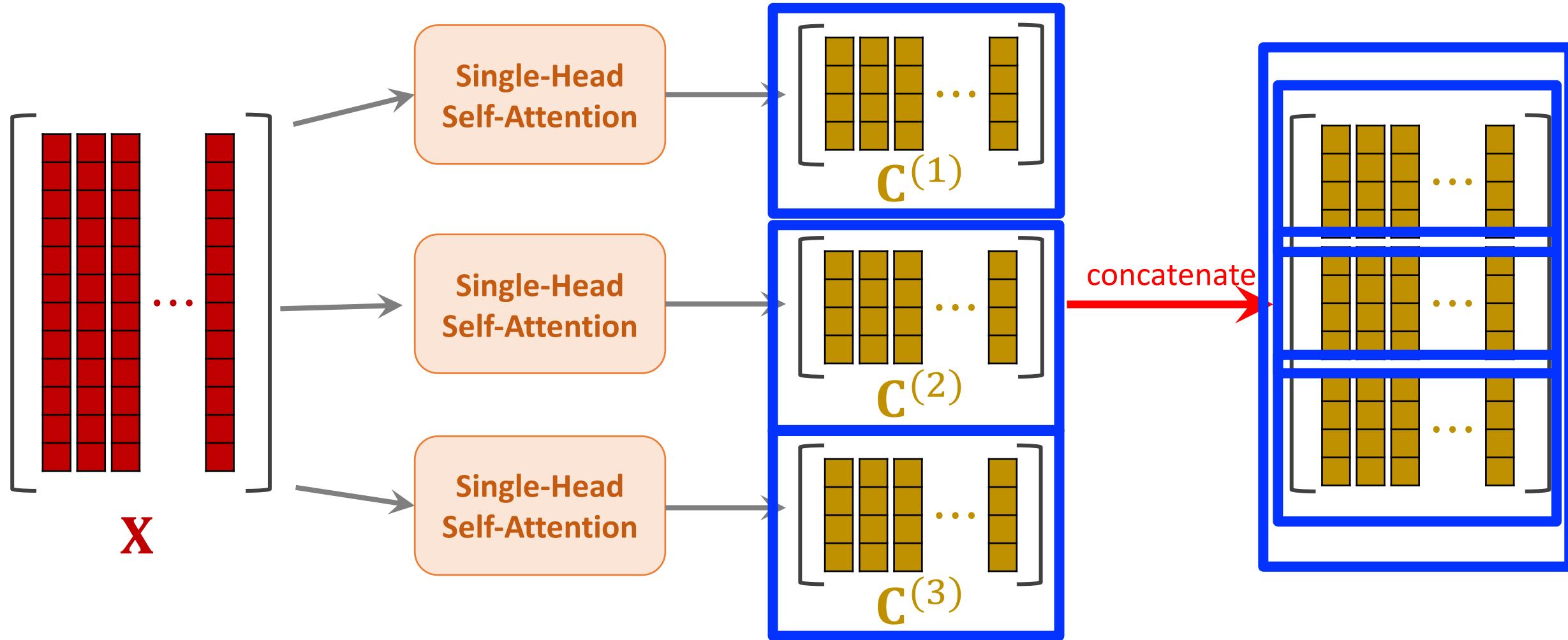
From Single-Head to Multi-Head

- Single-head self-attentions can be combined to form a multi-head self-attention.



From Single-Head to Multi-Head

- Single-head self-attentions can be combined to form a multi-head self-attention.



From Single-Head to Multi-Head

- Single-head self-attentions can be combined to form a multi-head self-attention.
- Single-head attentions can be combined to form a multi-head attention.

Encoder Network of Transformer

- Encoder network is a stack of 6 **blocks**.
- 1 encoder block \approx **8-head self-attention + dense**.
 - Input shape: $512 \times m$.
 - Output shape: $512 \times m$.

Decoder Network of Transformer

- Decoder network is a stack of 6 blocks.
- 1 decoder block \approx 8-head self-attention + 8-head attention + dense.
 - Input shape: $(512 \times m, 512 \times t)$.
 - Output shape: $512 \times t$.

Transformer Model

- Transformer is Seq2Seq model; it has an encoder and a decoder.
- Transformer model is **not RNN**.
- Transformer is based on **attention**, **self-attention**, and **dense layers**.
- Transformer outperforms all the state-of-the-art RNN models.

Bidirectional Encoder Representations from Transformers (BERT)

What is BERT?

- BERT [1] is for **pre-training** Transformer's [2] encoder.
- How?
- Predict masked word.
- Predict next sentence.

Reference

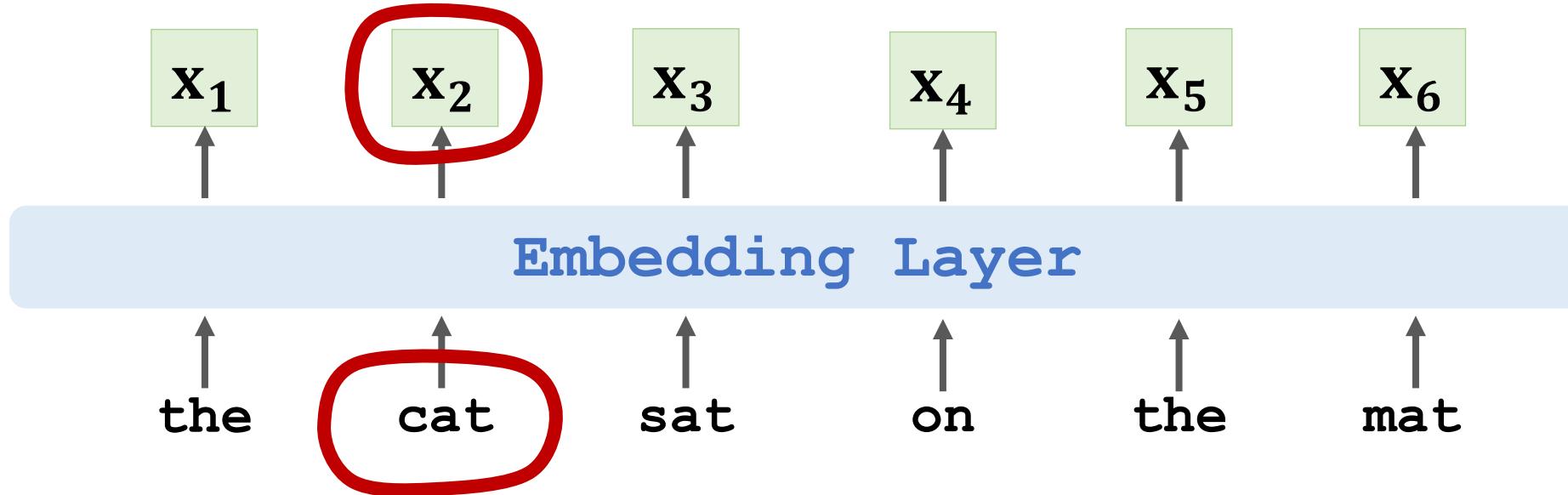
1. Devlin, Chang, Lee, and Toutanova. **BERT: Pre-training of deep bidirectional transformers for language understanding**. In *ACL*, 2019.
2. Vaswani and others. **Attention is all you need**. In *NIPS*, 2017.

Task 1: Predict Masked Words

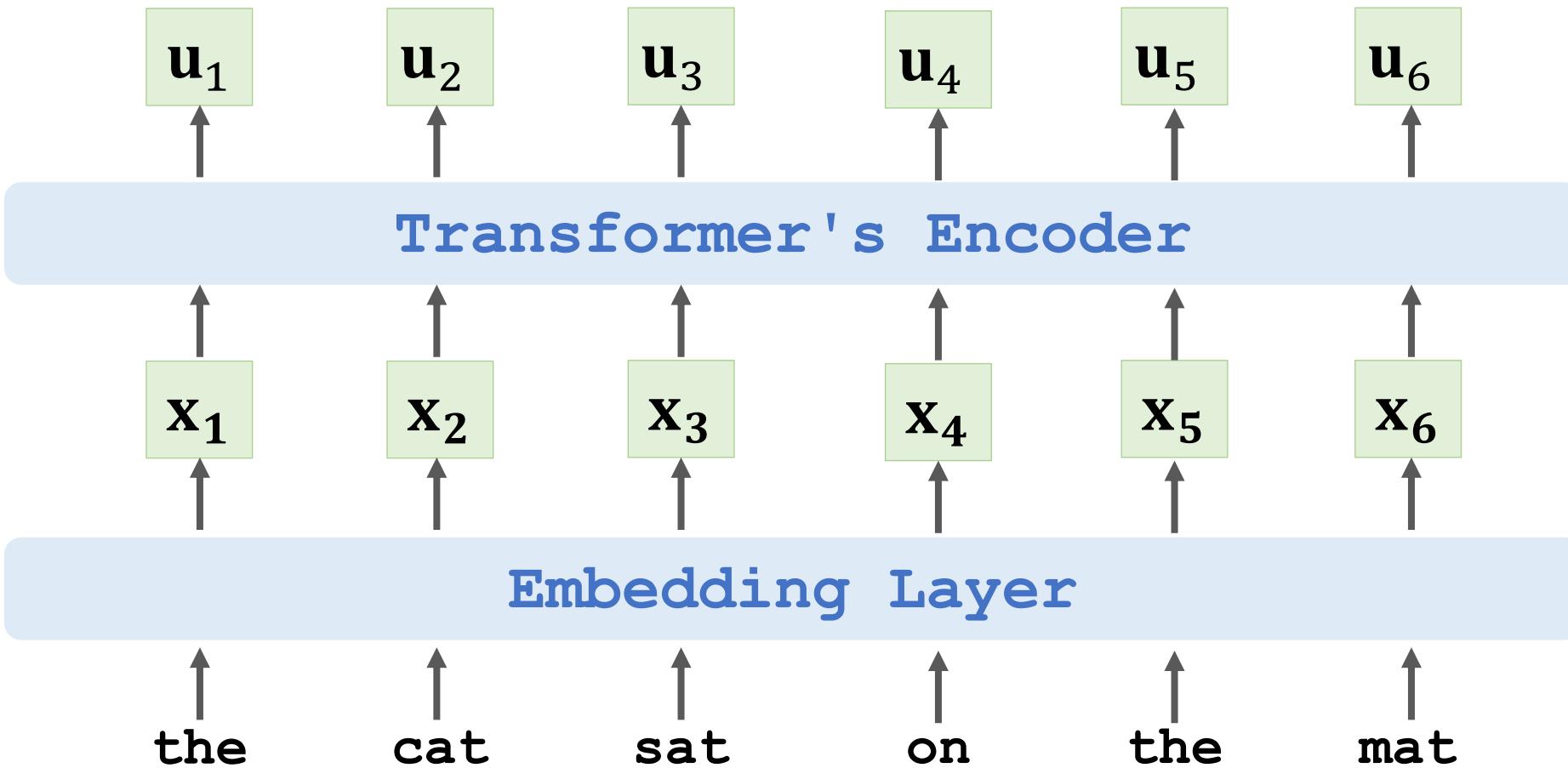
Revisit Transformer's Encoder

the cat sat on the mat

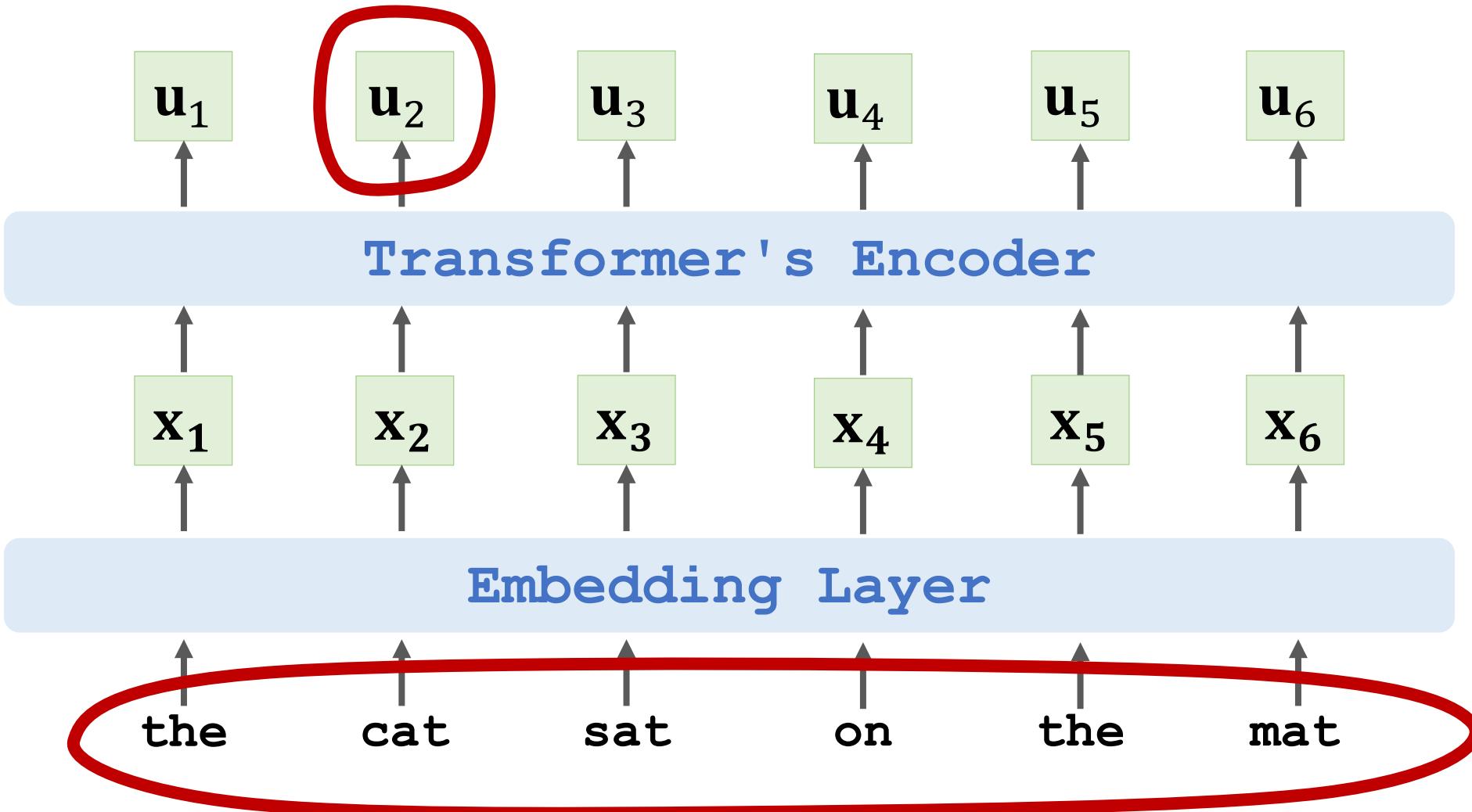
Revisit Transformer's Encoder



Revisit Transformer's Encoder



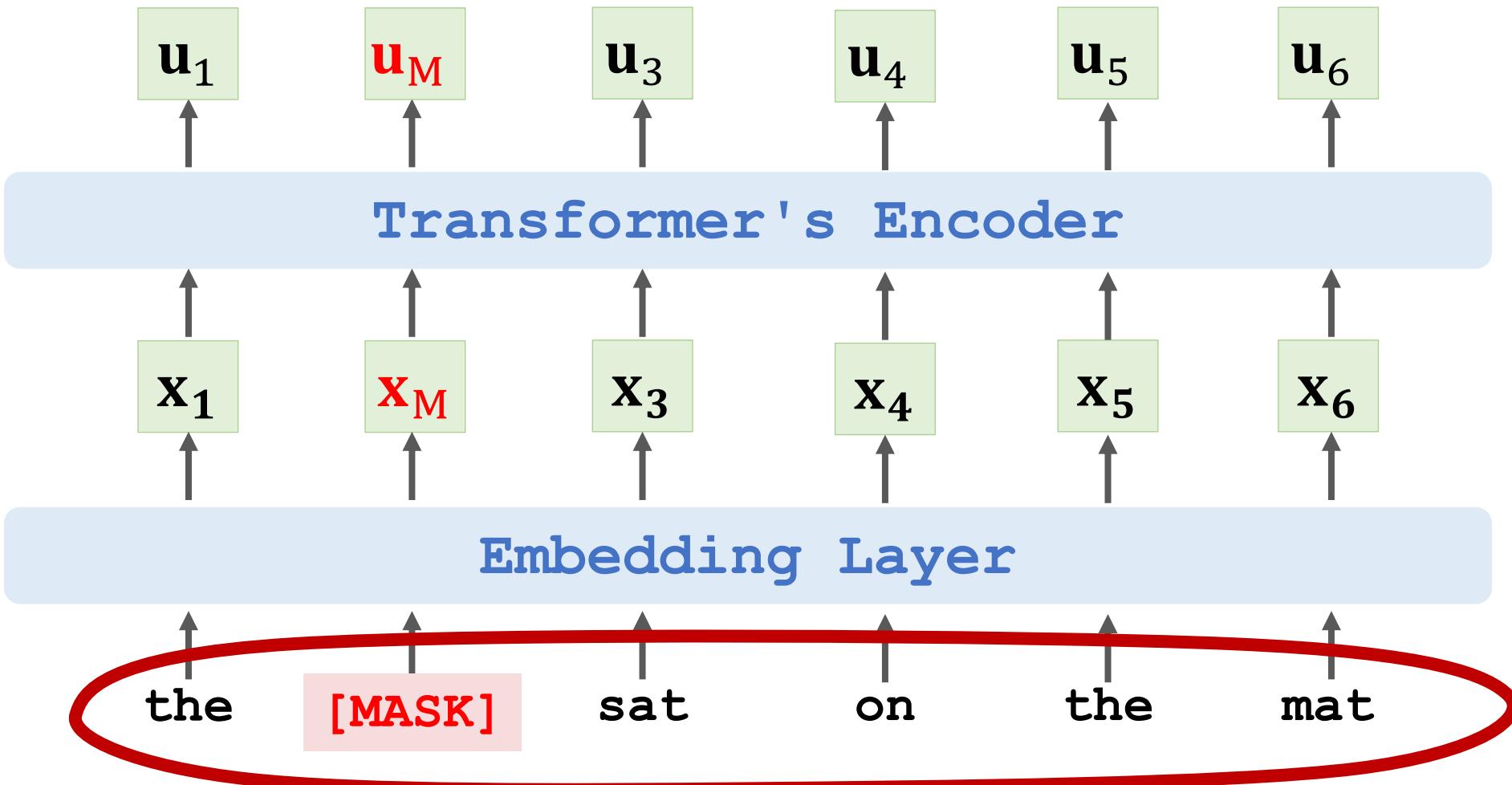
Revisit Transformer's Encoder

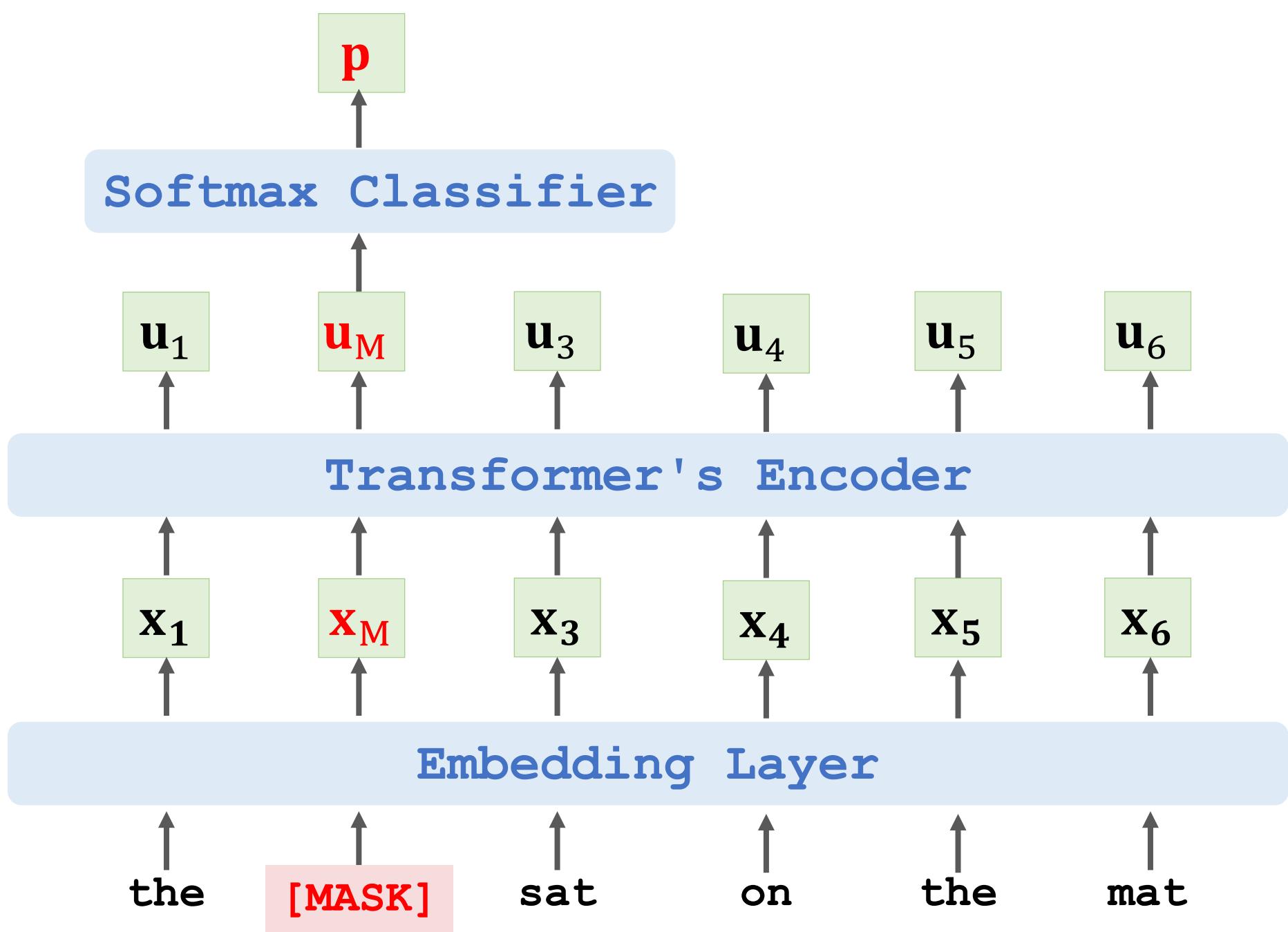


Randomly mask a word

- “The _____ sat on the mat”
- What is the **masked** word?

Randomly mask a word





Predict the masked word

- **e**: one-hot vector of the masked word “cat”.
- **p**: output probability distribution.
- Loss = CrossEntropy(**e, p**).
- Performing one gradient descent to update the model parameters.

Task 2: Predict the Next Sentence

Predict the next sentence

- Given the sentence:
“calculus is a branch of math”.
- Is this the next sentence?
“it was developed by newton and leibniz”

Predict the next sentence

- Given the sentence:
“calculus is a branch of math”.
- Is this the next sentence?
“it was developed by newton and leibniz”
- Is this the next sentence?
“panda is native to south central china”

Input Representation

- **Input:**

[CLS] “calculus is a branch of math”

[SEP] “it was developed by newton and leibniz”

Input Representation

- **Input:**

[CLS] “calculus is a branch of math”
[SEP] “it was developed by newton and leibniz”

- [CLS] is a token for classification.
- [SEP] is for separating sentences.

Input Representation

- **Input:**

[CLS] “calculus is a branch of math”

[SEP] “it was developed by newton and leibniz”

- **Target:** true

Input Representation

- **Input:**

[CLS] “calculus is a branch of math”

[SEP] “panda is native to south central china”

- **Target:** false

Predict the next sentence

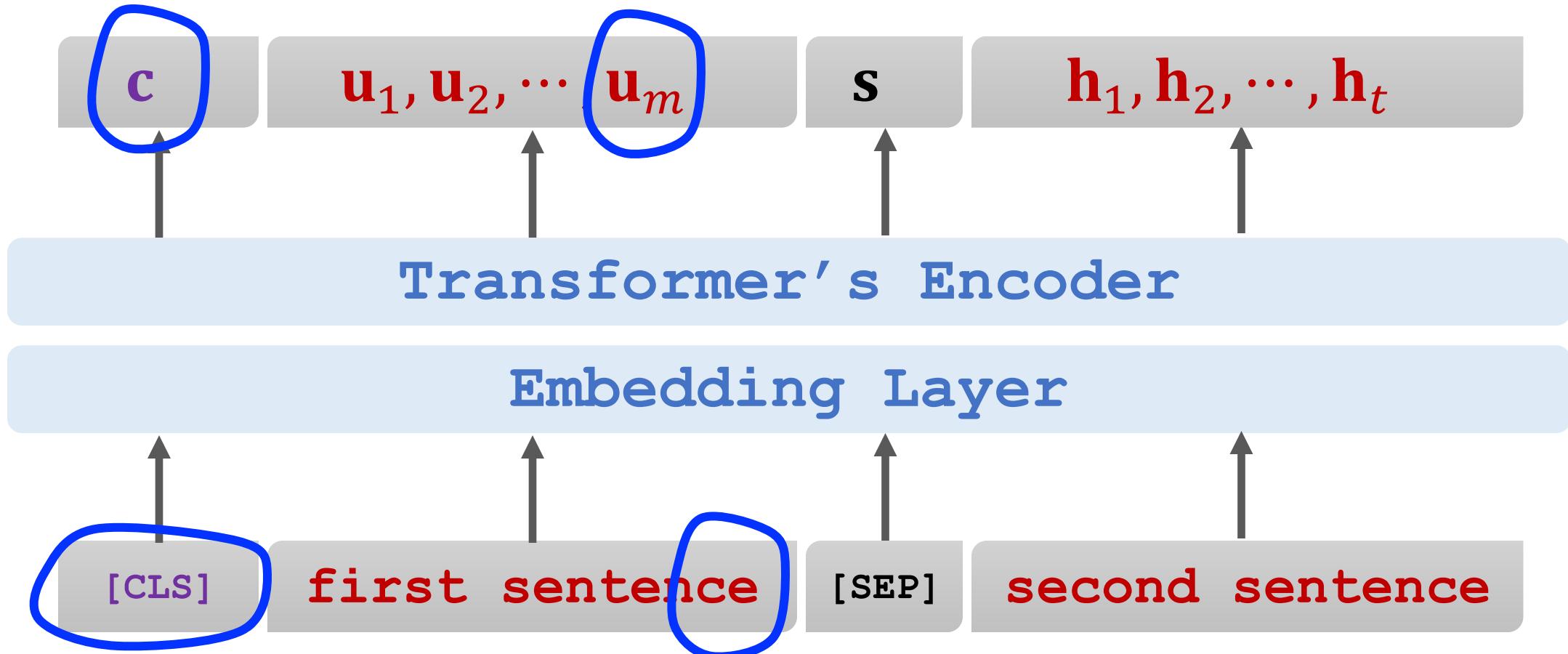
[CLS]

first sentence

[SEP]

second sentence

Predict the next sentence



f

(between 0 and 1)

Binary Classifier

c

$\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_m$

s

$\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_t$

Transformer's Encoder

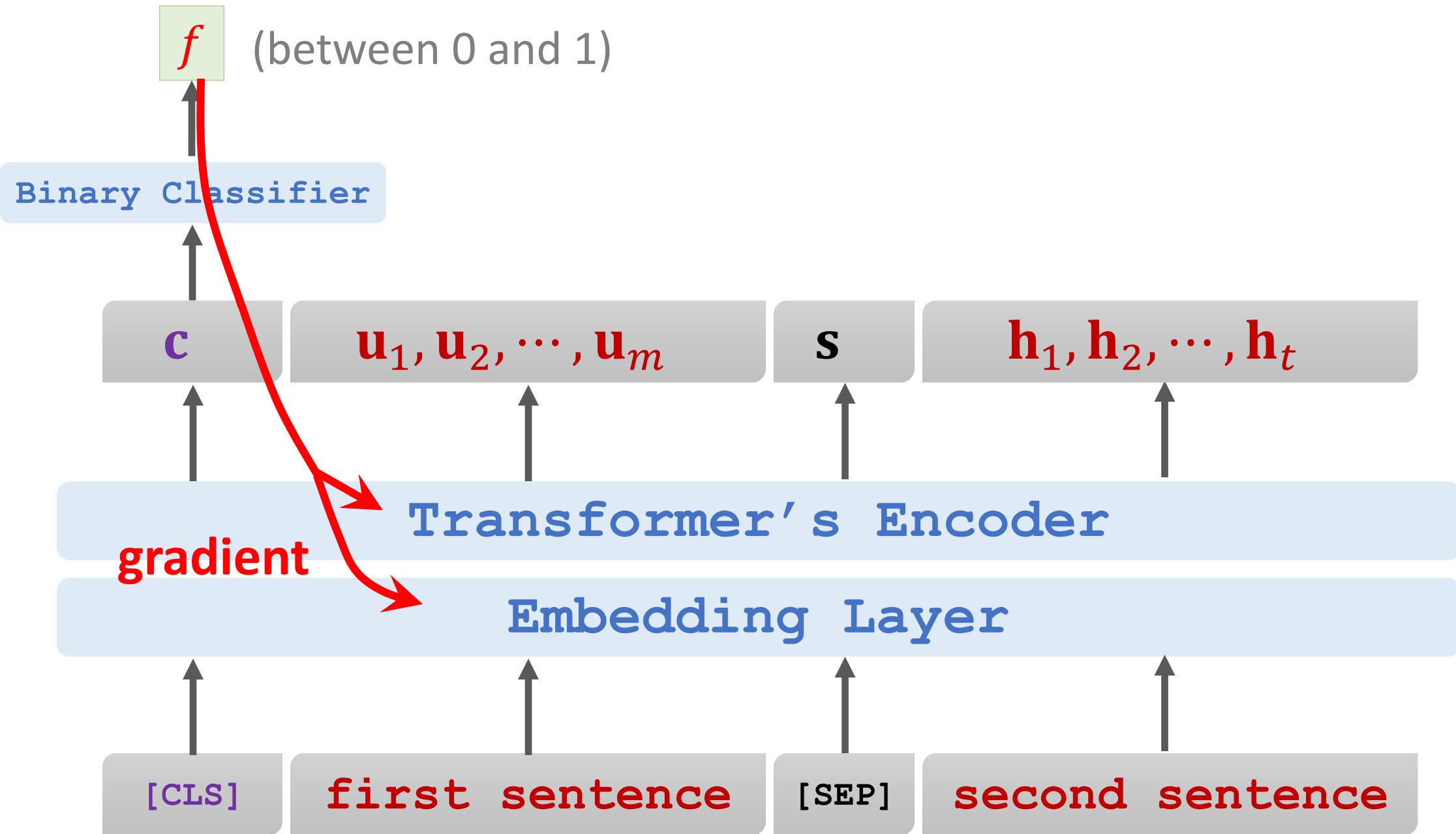
Embedding Layer

[CLS]

first sentence

[SEP]

second sentence



Why is the classification task helpful?

- The task helps improve the **embedding layer**.
- The task helps improve the **self-attention layers**.

Combining the two methods

Input Representation

- **Input:**

"[CLS] calculus is a [MASK] of math
[SEP] it [MASK] developed by newton and leibniz"

Input Representation

- **Input:**

"[CLS] calculus is a [MASK] of math
[SEP] it [MASK] developed by newton and leibniz".

- **Targets:**

true, "branch", "was",

Input Representation

- **Input:**

"[CLS] calculus is a branch of math
[SEP] panda is native to [MASK] central china".

- **Targets:** false, "south".

Training

- **Loss 1** is for binary classification (i.e., deciding if the next sentence is real.)
- **Loss 2** and **Loss 3** are for multi-class classification (i.e., predicting the masked words.)
- Objective function is the **sum** of the three **loss functions**.
- Update model parameters by performing one gradient descent.

Data

- BERT does not need manually labeled data. (Nice! Manual labeling is expensive.)
- Use large-scale data, e.g., English Wikipedia (2.5 billion words.)
- Randomly mask 15% words (with some tricks.)
- 50% of the next sentences are real. (The other 50% are fake.)

Cost of Computation

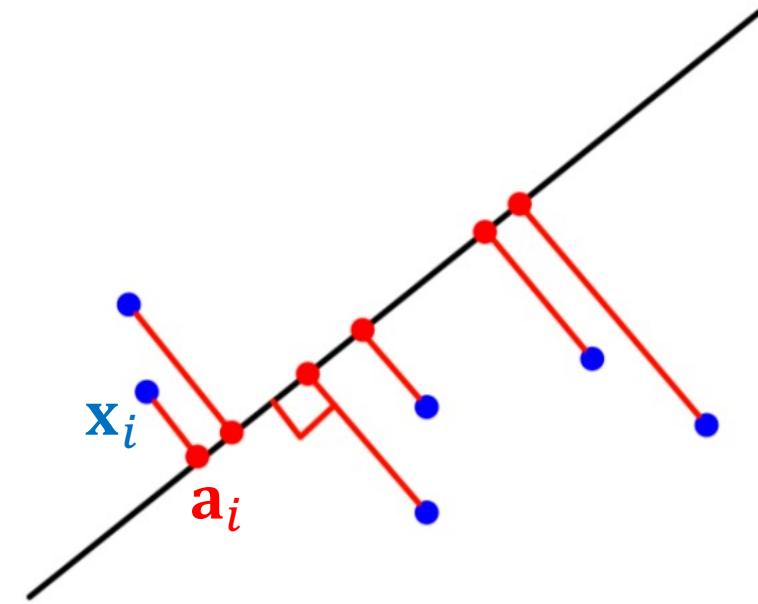
- BERT Base
 - 110M parameters.
 - 16 TPUs, 4 days of training (without hyper-parameter tuning.)
- BERT Large
 - 235M parameters.
 - 64 TPUs, 4 days of training (without hyper-parameter tuning.)

Autoencoder for Dimensionality Reduction

Revisit Principal Component Analysis (PCA)

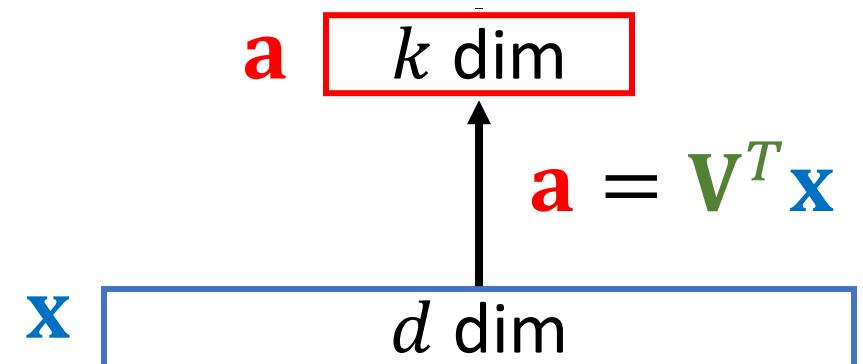
Principal Component Analysis (PCA)

- Input: $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$ (with zero-mean).
- Output: $\mathbf{a}_1, \dots, \mathbf{a}_n \in \mathbb{R}^k$ ($k \ll d$).



Principal Component Analysis (PCA)

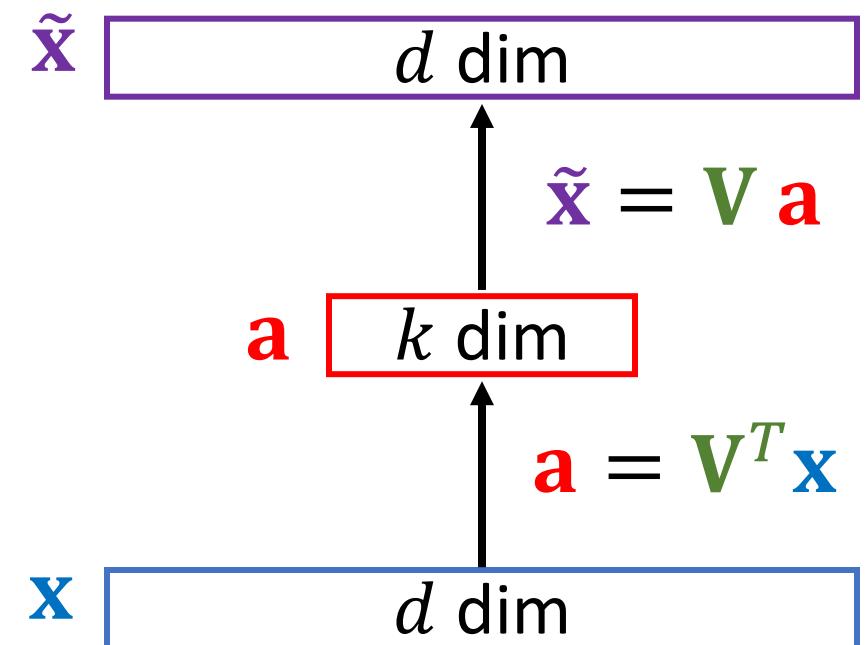
- Input: $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$ (with zero-mean).
- Output: $\mathbf{a}_1, \dots, \mathbf{a}_n \in \mathbb{R}^k$ ($k \ll d$).
- PCA finds a $d \times k$ column orthogonal matrix \mathbf{V}



Principal Component Analysis (PCA)

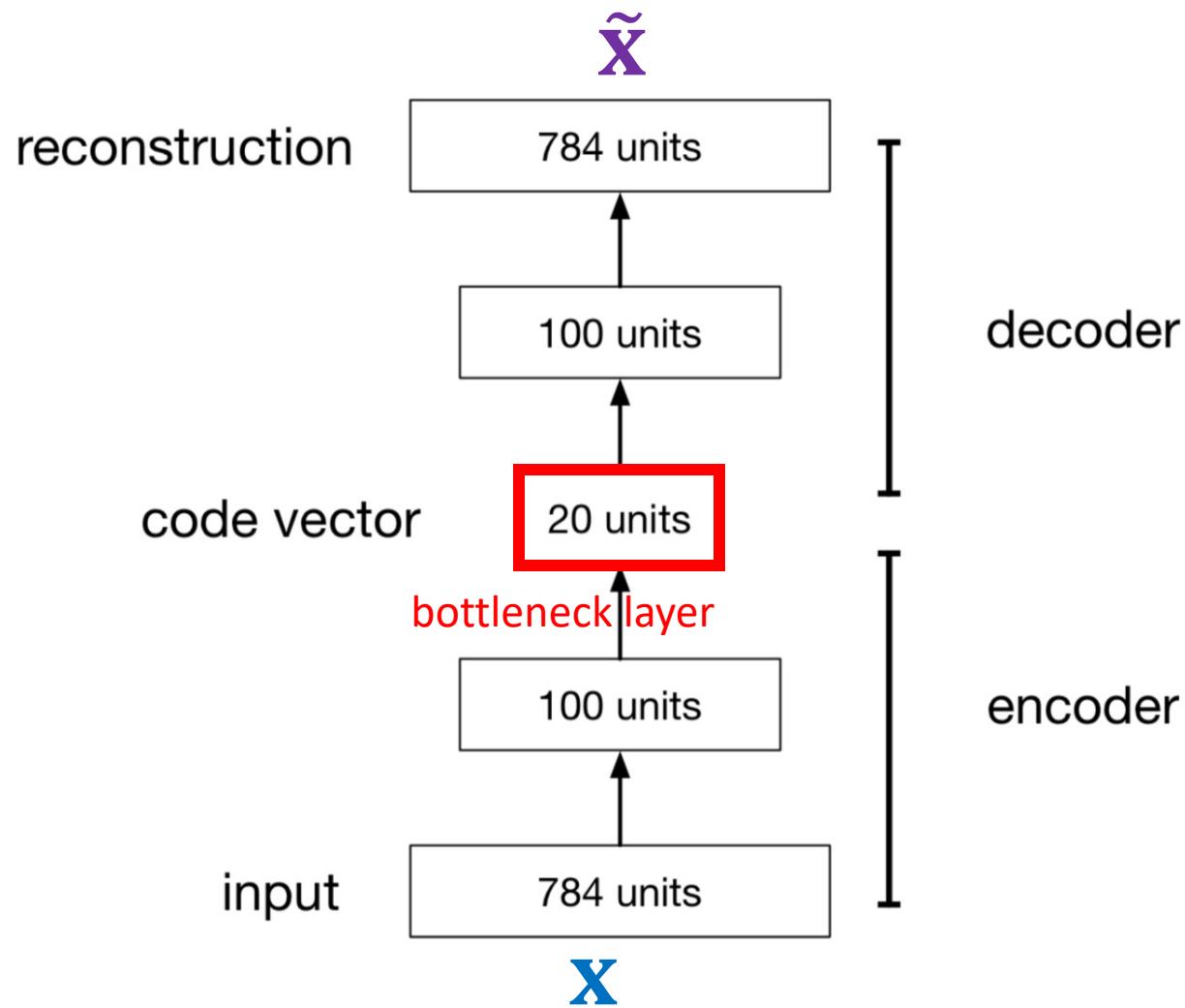
- Input: $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$ (with zero-mean).
- Output: $\mathbf{a}_1, \dots, \mathbf{a}_n \in \mathbb{R}^k$ ($k \ll d$).
- PCA finds a $d \times k$ column orthogonal matrix \mathbf{V} such that

$$\min_{\mathbf{V}} \sum_{j=1}^n \left\| \mathbf{x}_j - \tilde{\mathbf{x}}_j \right\|_2^2, \quad \text{s.t. } \mathbf{V}^T \mathbf{V} = \mathbf{I}_k.$$



Autoencoder

Autoencoder



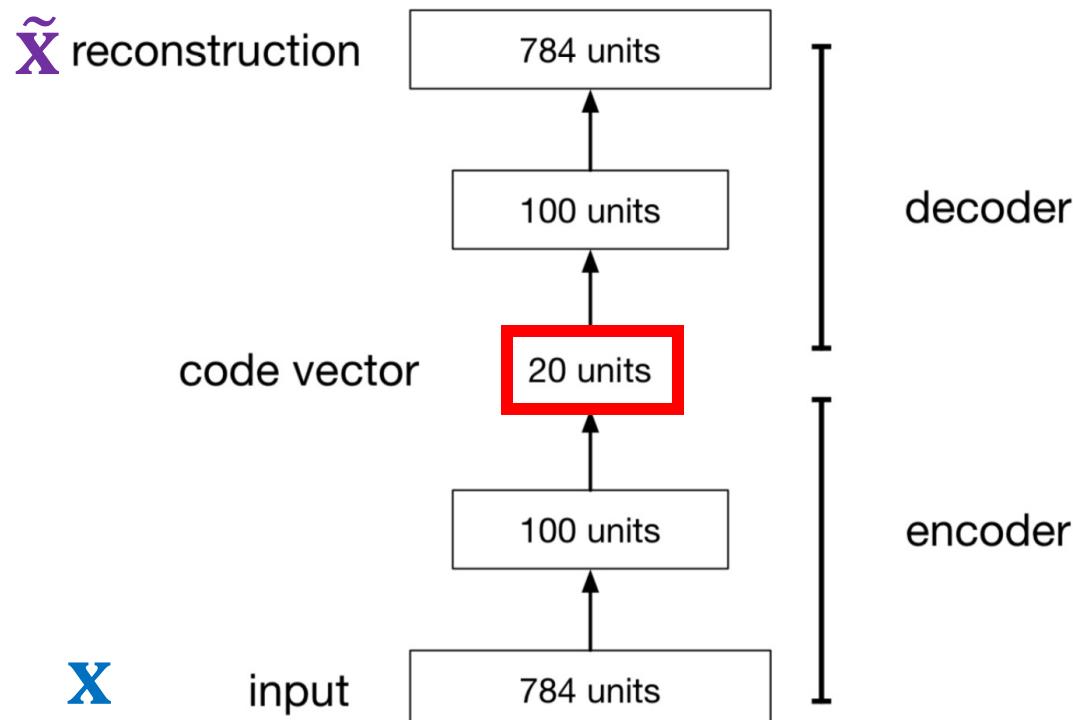
- An autoencoder is a neural net taking an input \mathbf{x} and reconstruct \mathbf{x} .
- For dim reduction, we need a **bottleneck layer** whose output shape is much smaller than the input.
- Loss function:

$$\sum_{j=1}^n \left\| \mathbf{x}_j - \tilde{\mathbf{x}}_j \right\|_2^2.$$

Autoencoder v.s. PCA

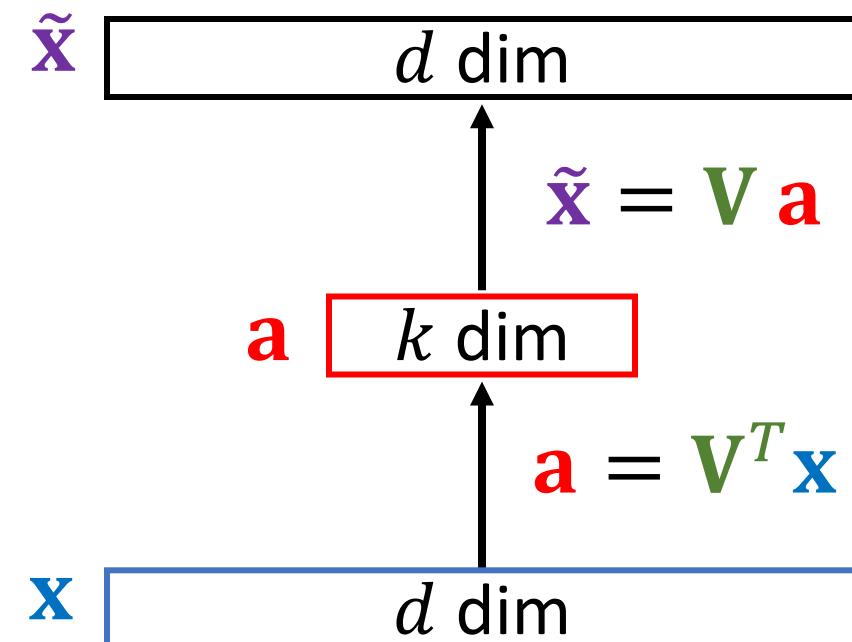
Autoencoder is nonlinear

$$\tilde{\mathbf{x}} = \text{decoder}(\text{encoder}(\mathbf{x}))$$



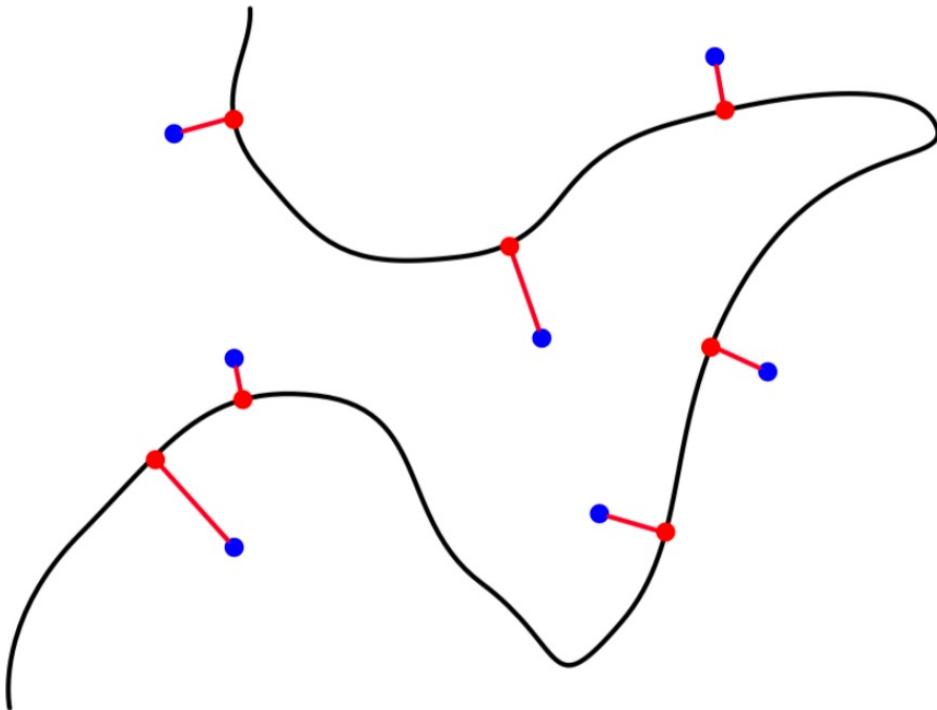
PCA is linear

$$\tilde{\mathbf{x}} = \mathbf{V}\mathbf{V}^T\mathbf{x}$$

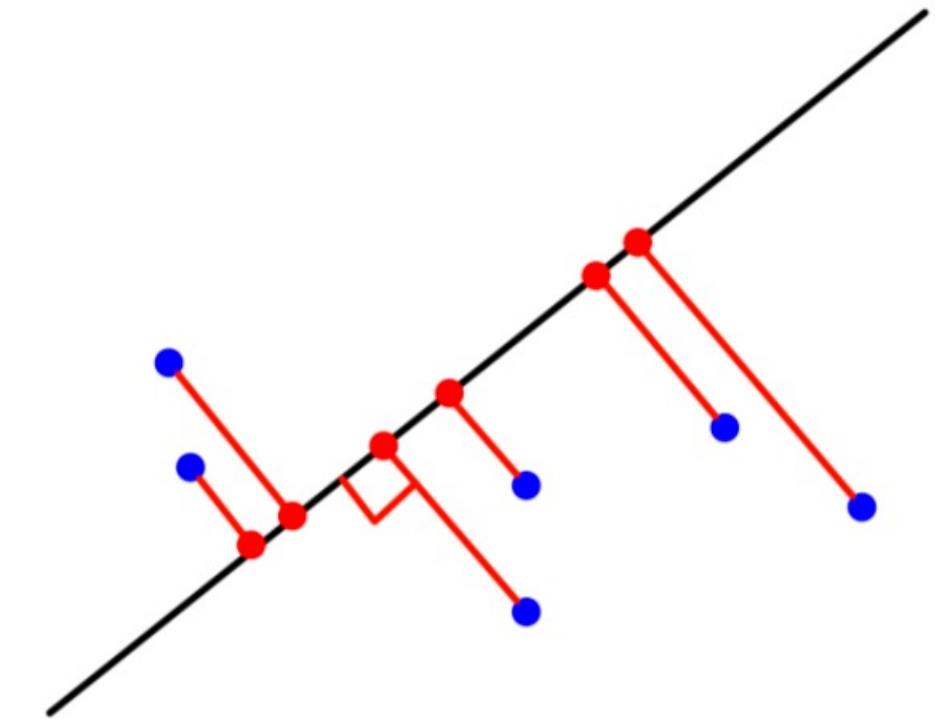


Autoencoder v.s. PCA

Autoencoder projects data onto nonlinear manifold.



PCA projects data onto a subspace.



Implementation Using Keras

Load Data and Reshape Images to Vectors

The MNIST Dataset

- $n = 60,000$ training samples $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$.
- Each \mathbf{x}_j is a 28×28 image (reshape to 784-dim vector).
- Each y_j is an integer in $\{0, 1, 2, \dots, 9\}$.



```
print('Shape of x_train_vec: ' + str(x_train_vec.shape))
print('Shape of x_test_vec: ' + str(x_test_vec.shape))
```

Shape of x_train_vec: (60000, 784)

Shape of x_test_vec: (10000, 784)

2 Ways of Building a Fully-Connected Net

```
from keras.layers import Dense
from keras import models

model = models.Sequential()
model.add(Dense(100, activation='relu',
               input_shape=(784,)))
model.add(Dense(20, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(784, activation='relu'))
```

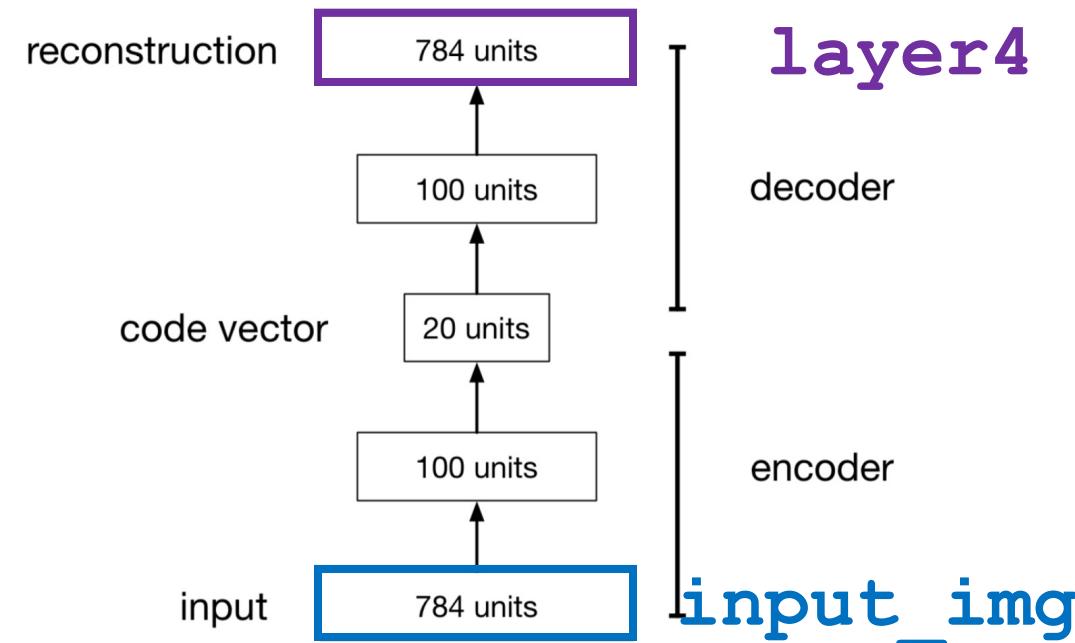
```
from keras.layers import Input, Dense
from keras import models

input_img = Input(shape=(784,))

layer1 = Dense(100, activation='relu')(input_img)
layer2 = Dense(20, activation='relu')(layer1)
layer3 = Dense(100, activation='relu')(layer2)
layer4 = Dense(784, activation='relu')(layer3)

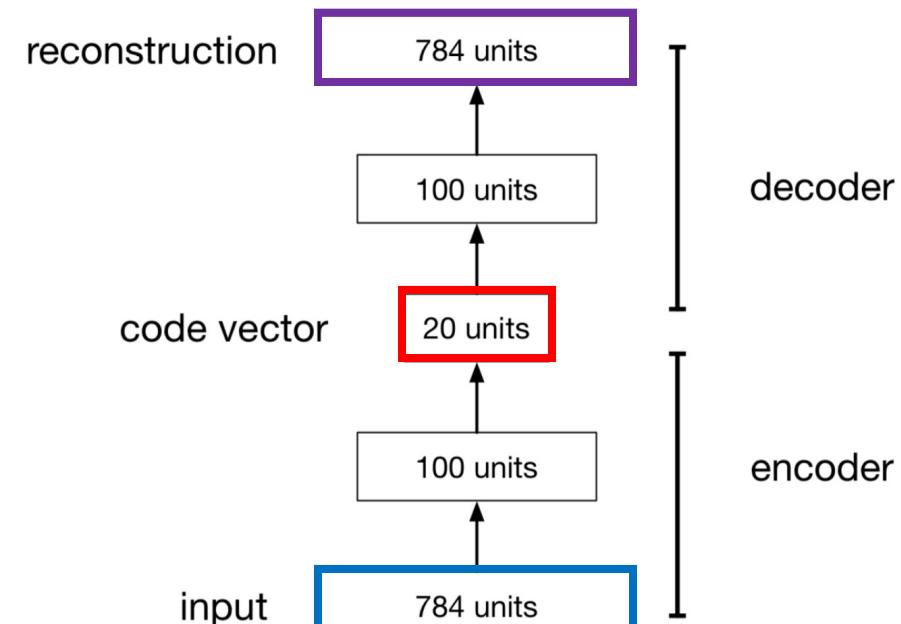
model = models.Model(input_img, layer4)
```

2 Ways of Building a Fully-Connected Net



```
from keras.layers import Input, Dense  
from keras import models  
  
input_img = Input(shape=(784,))  
  
layer1 = Dense(100, activation='relu')(input_img)  
layer2 = Dense(20, activation='relu')(layer1)  
layer3 = Dense(100, activation='relu')(layer2)  
layer4 = Dense(784, activation='relu')(layer3)  
  
model = models.Model(input_img, layer4)
```

2 Ways of Building a Fully-Connected Net



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 784)	0
dense_5 (Dense)	(None, 100)	78500
dense_6 (Dense)	(None, 20)	2020
dense_7 (Dense)	(None, 100)	2100
dense_8 (Dense)	(None, 784)	79184

Total params: 161,804
Trainable params: 161,804
Non-trainable params: 0

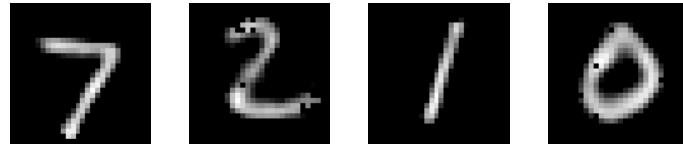
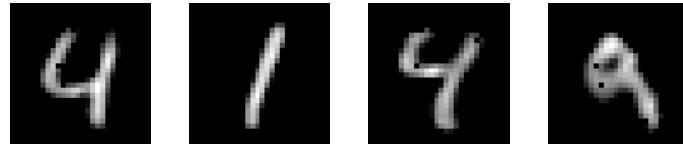
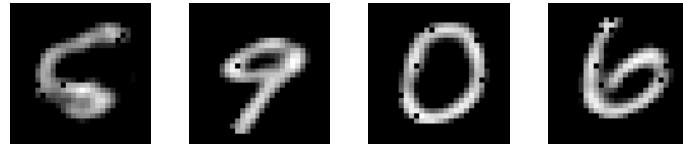
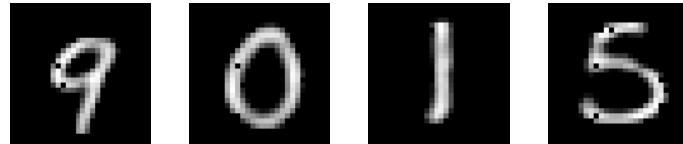
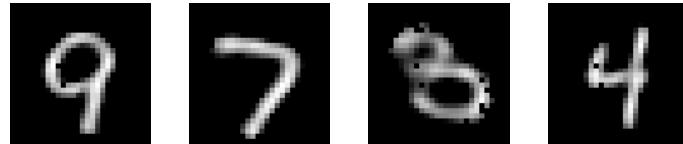
Train the Model

The inputs and targets are the same.

```
model.compile(optimizer='RMSprop', loss='mean_squared_error')
history = model.fit(x_train_vec, x_train_vec, batch_size=128, epochs=50)
```

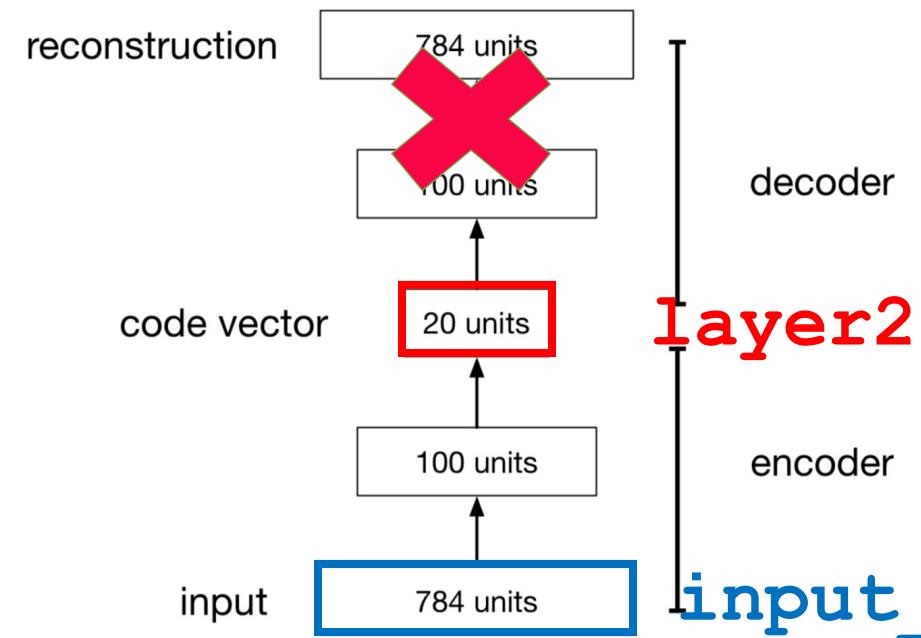
```
Epoch 1/50
60000/60000 [=====] - 2s 32us/step - loss: 0.0390
Epoch 2/50
60000/60000 [=====] - 2s 30us/step - loss: 0.0281
:
:
Epoch 49/50
60000/60000 [=====] - 2s 38us/step - loss: 0.0145
Epoch 50/50
60000/60000 [=====] - 2s 37us/step - loss: 0.0145
```

Results on the Test Set

Input	Reconstructed
	
	
	
	
	

Dimensionality Reduction

Get the Code Vectors



```
encoder = models.Model(input_img, layer2)  
encoder.summary()
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 784)	0
dense_5 (Dense)	(None, 100)	78500
dense_6 (Dense)	(None, 20)	2020

Total params: 80,520
Trainable params: 80,520
Non-trainable params: 0

Visualize the Low-Dim Vectors

Get the low-dim code vectors.

20-dim

```
encoded_test = encoder.predict(x_test_vec)    784-dim
print('Shape of encoded_test: ' + str(encoded_test.shape))
```

Shape of encoded_test: (10000, 20)

Visualize the Low-Dim Vectors

Get the low-dim code vectors.

20-dim

```
encoded_test = encoder.predict(x_test_vec)  
print('Shape of encoded_test: ' + str(encoded_test.shape))
```

784-dim

Shape of encoded_test: (10000, 20)

Project the 20-dim vectors to 2-dim by TSNE

2-dim

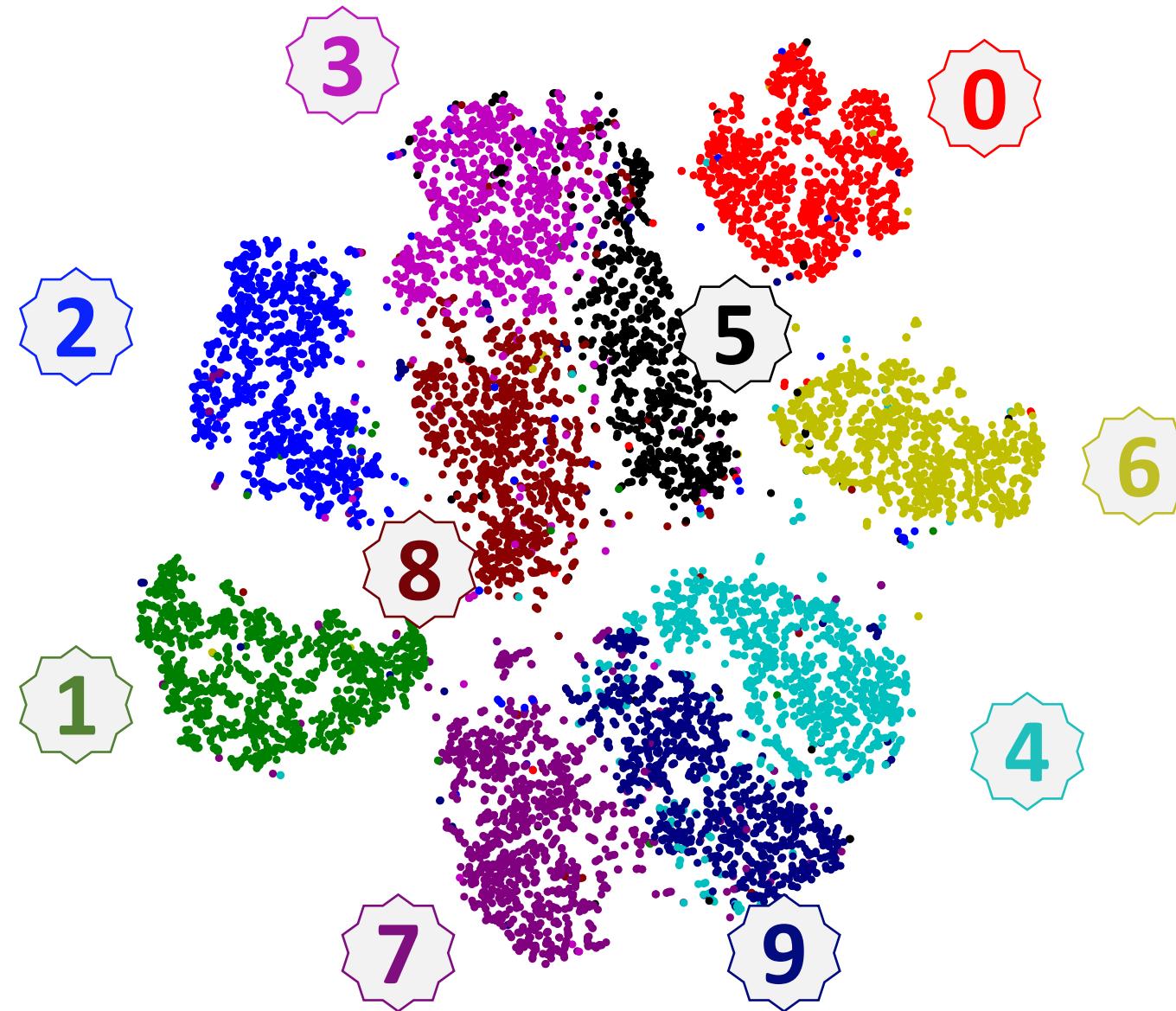
```
from sklearn.manifold import TSNE
```

20-dim

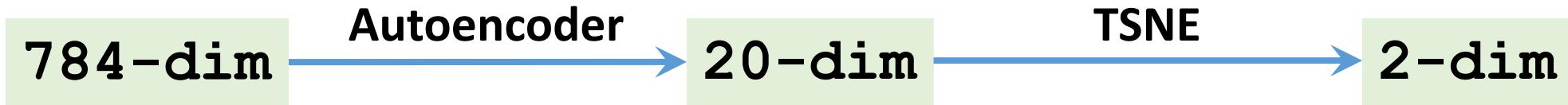
```
embedded_test = TSNE(n_components=2).fit_transform(encoded_test)  
print(embedded_test.shape)
```

(10000, 2)

Scatter Plot of the TSNE Embedding



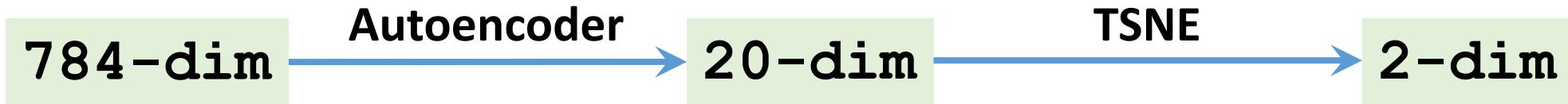
Visualize the Low-Dim Vectors



Question: Why not using Autoencoder to map 784-dim vectors to 2-dim?

Answer: The resulting visualization is not as good as TSNE.

Visualize the Low-Dim Vectors



Question: Why not using Autoencoder to map 784-dim vectors to 2-dim?

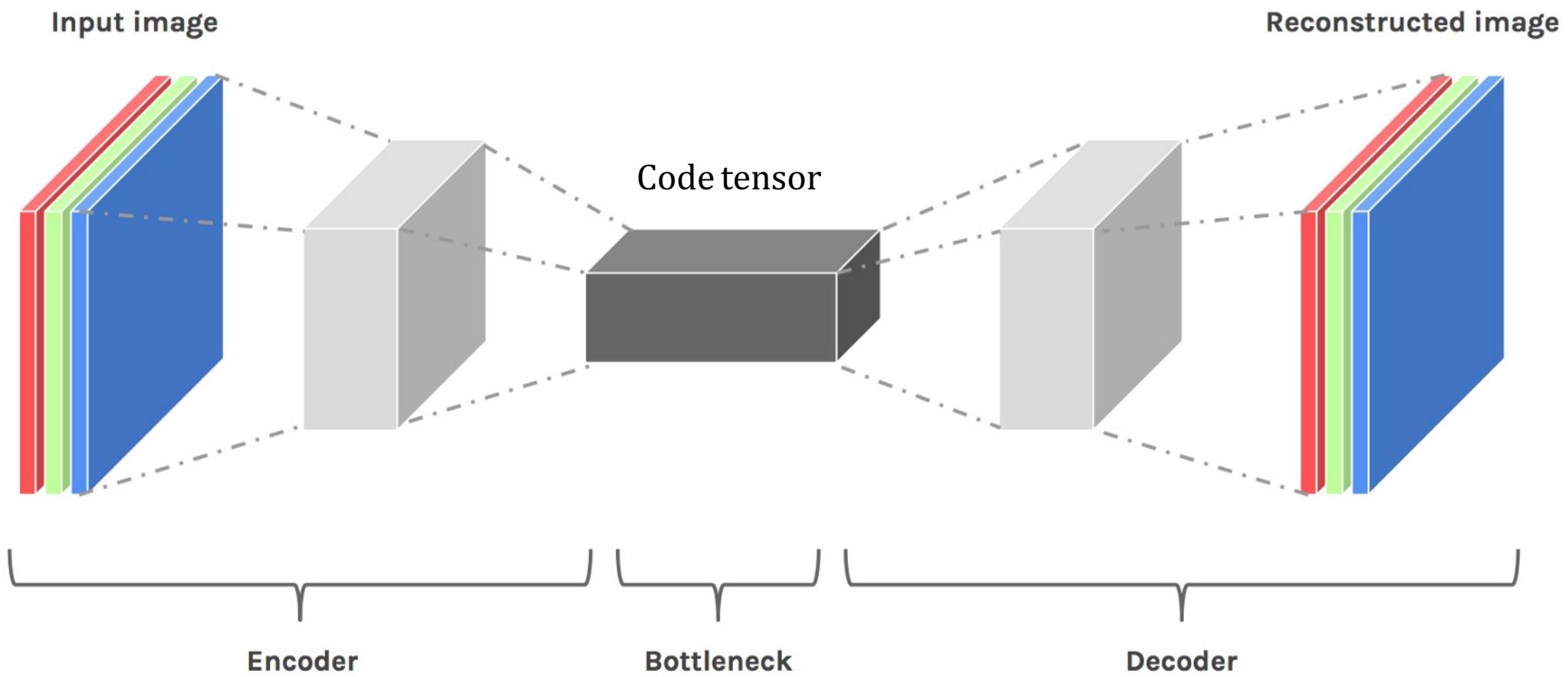
Answer: The resulting visualization is not as good as TSNE.

Question: Why not using TSNE to map 784-dim vectors to 2-dim?

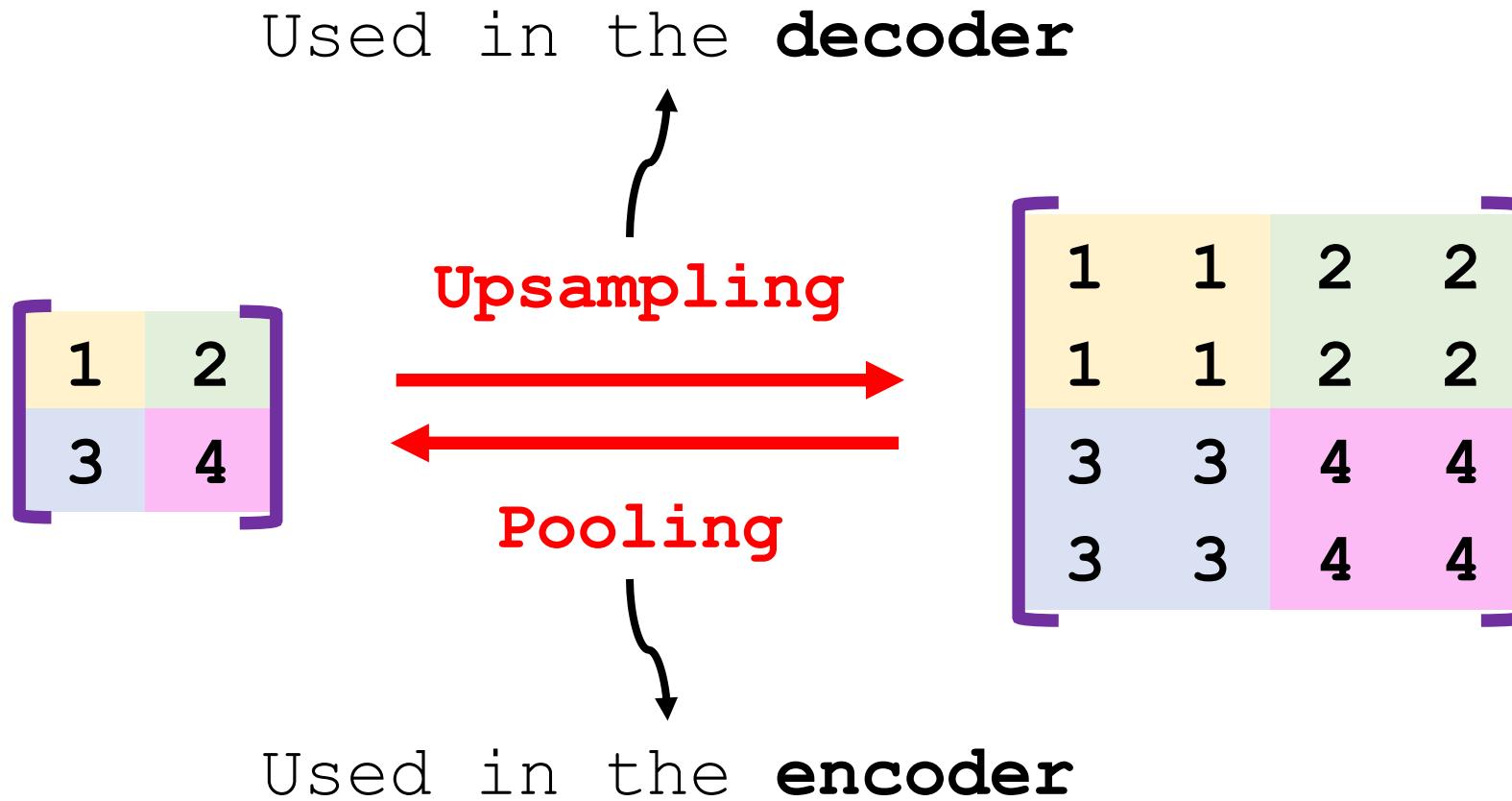
Answer: The results are good, but TSNE is slow for 784-dim vectors.

Convolutional Autoencoder

Convolutional Autoencoder



Pooling v.s. Upsampling



Implementation Using Keras

```
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D
from keras import models

input_img = Input(shape=(28, 28, 1))

encode1 = Conv2D(8, (3, 3), activation='relu', padding='same')(input_img)
encode2 = MaxPooling2D((2, 2), padding='same')(encode1)
encode3 = Conv2D(4, (3, 3), activation='relu', padding='same')(encode2)
encode4 = MaxPooling2D((2, 2), padding='same')(encode3)
encode5 = Conv2D(4, (3, 3), activation='relu', padding='same')(encode4)
encode6 = MaxPooling2D((2, 2), padding='same')(encode5)

decode1 = Conv2D(4, (3, 3), activation='relu', padding='same')(encode6)
decode2 = UpSampling2D((2, 2))(decode1)
decode3 = Conv2D(4, (3, 3), activation='relu', padding='same')(decode2)
decode4 = UpSampling2D((2, 2))(decode3)
decode5 = Conv2D(8, (3, 3), activation='relu')(decode4)
decode6 = UpSampling2D((2, 2))(decode5)
decode7 = Conv2D(1, (3, 3), activation='relu', padding='same')(decode6)

model = models.Model(input_img, decode7)
```

Implementation Using Keras

Input image (28×28×1)

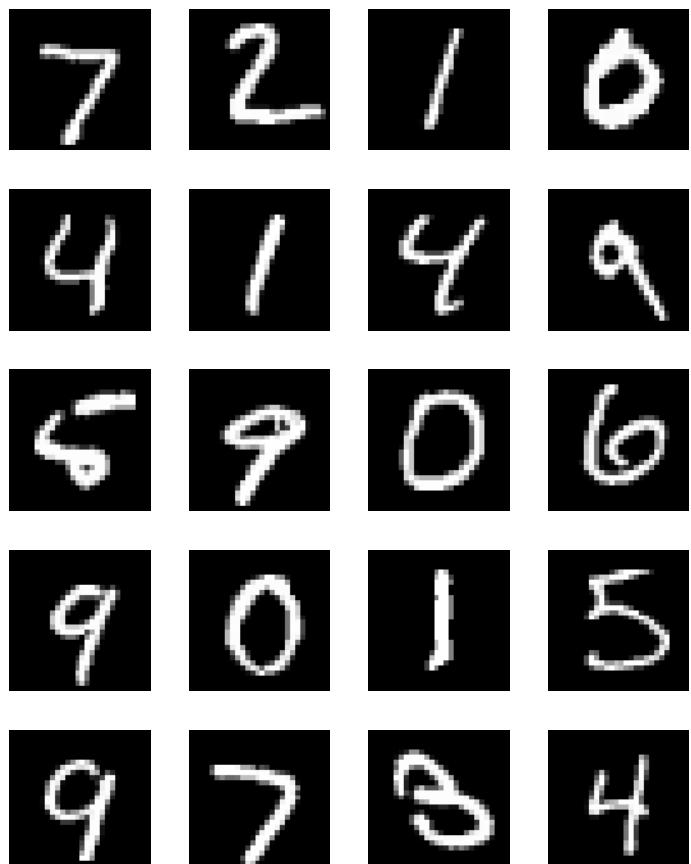
Code tensor (4×4×4)

Output of decoder (28×28×1)

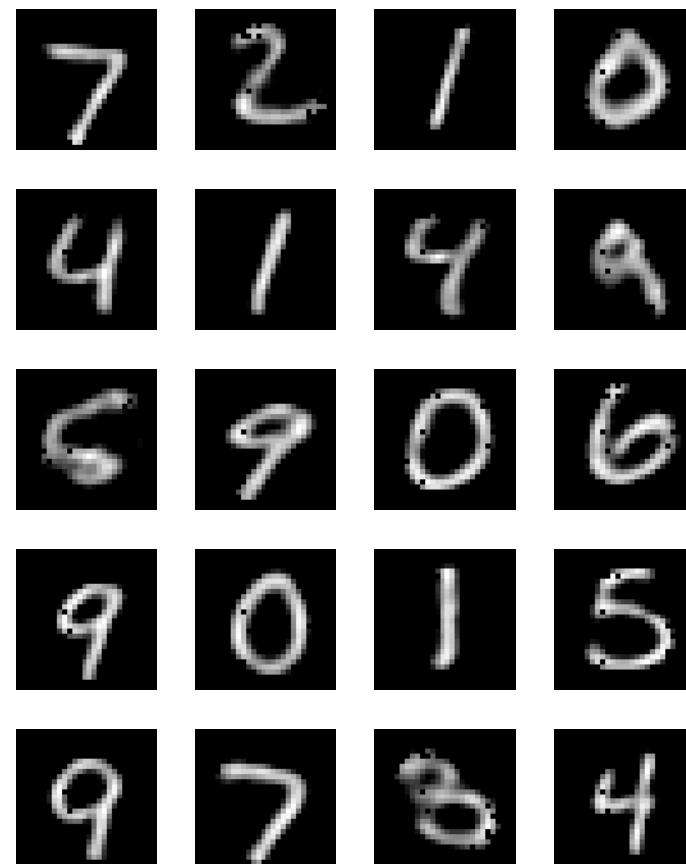
model.summary()		
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 28, 28, 1)	0
conv2d_1 (Conv2D)	(None, 28, 28, 8)	80
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 8)	0
conv2d_2 (Conv2D)	(None, 14, 14, 4)	292
max_pooling2d_2 (MaxPooling2D)	(None, 7, 7, 4)	0
conv2d_3 (Conv2D)	(None, 7, 7, 4)	148
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 4)	0
conv2d_4 (Conv2D)	(None, 4, 4, 4)	148
up_sampling2d_1 (UpSampling2D)	(None, 8, 8, 4)	0
conv2d_5 (Conv2D)	(None, 8, 8, 4)	148
up_sampling2d_2 (UpSampling2D)	(None, 16, 16, 4)	0
conv2d_6 (Conv2D)	(None, 14, 14, 8)	296
up_sampling2d_3 (UpSampling2D)	(None, 28, 28, 8)	0
conv2d_7 (Conv2D)	(None, 28, 28, 1)	73
Total params: 1,185		
Trainable params: 1,185		
Non-trainable params: 0		

Reconstructions (on the test set)

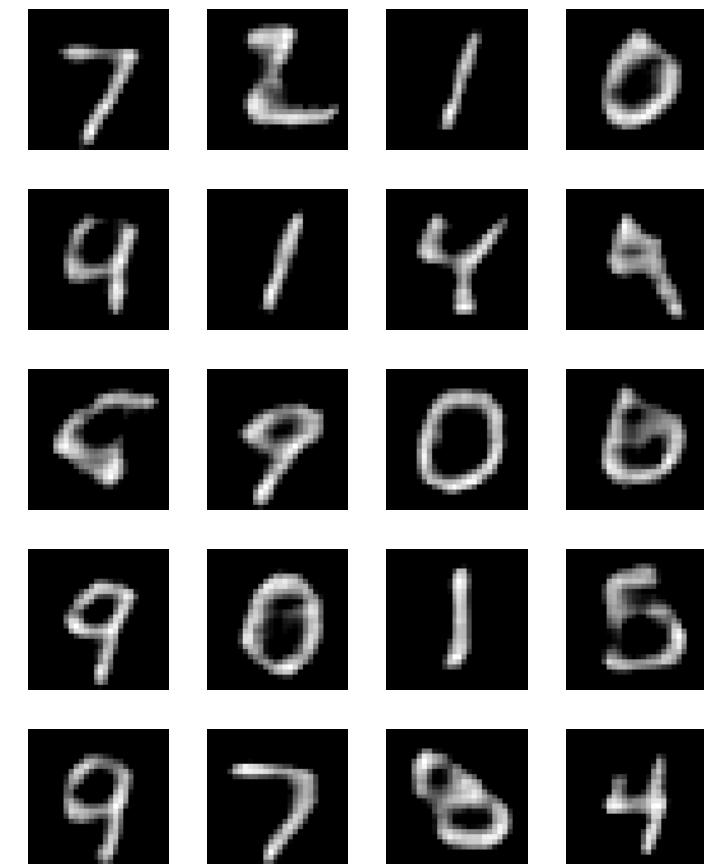
Original Input



Dense Autoencoder



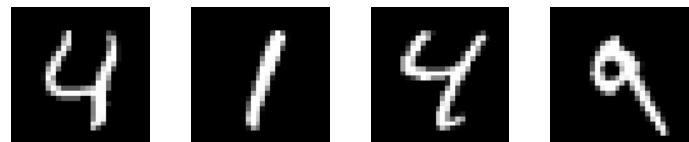
Conv Autoencoder



Denoising Autoencoder

Denoising Autoencoder

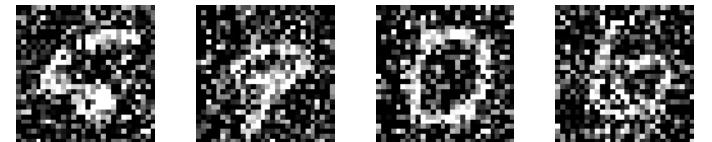
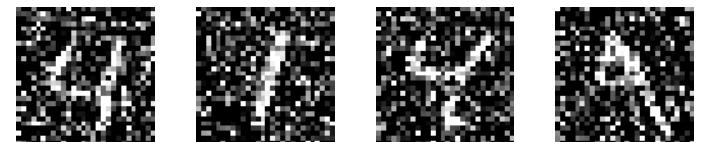
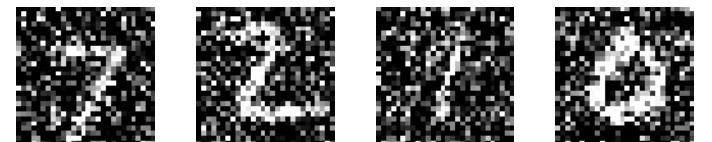
Original



add noise



Noisy



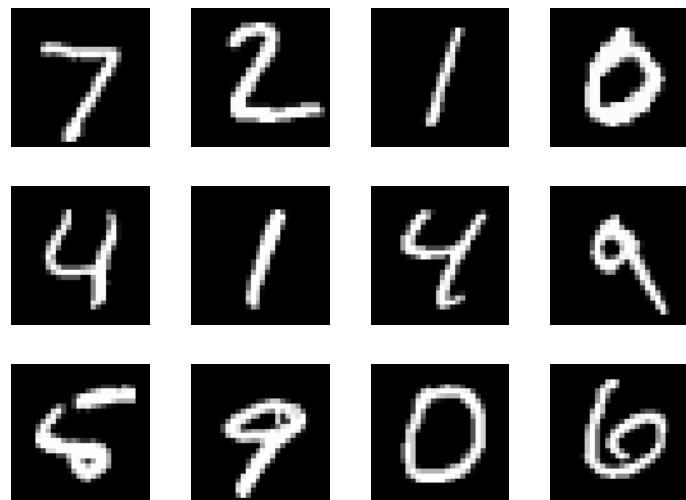
```
import numpy as np

noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)

x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

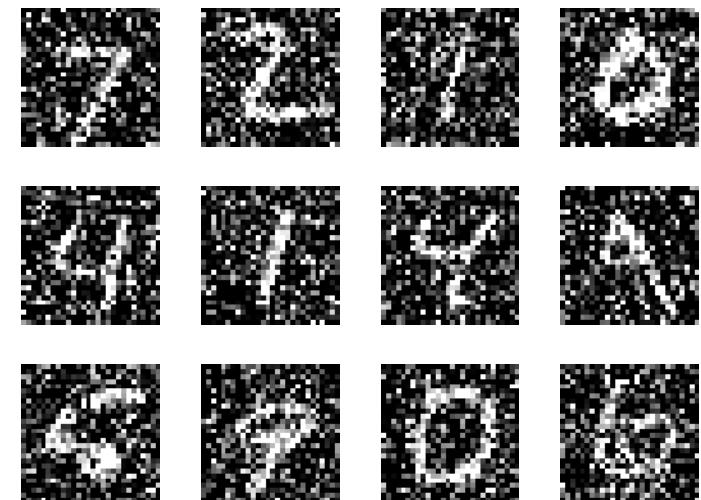
Denoising Autoencoder

Original



add noise

Noisy



Used as targets

Used as inputs

Convolutional Autoencoder

```
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D
from keras import models

input_img = Input(shape=(28, 28, 1))

encode1 = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
encode2 = MaxPooling2D((2, 2), padding='same')(encode1)
encode3 = Conv2D(32, (3, 3), activation='relu', padding='same')(encode2)
encode4 = MaxPooling2D((2, 2), padding='same')(encode3)

decode1 = Conv2D(32, (3, 3), activation='relu', padding='same')(encode4)
decode2 = UpSampling2D((2, 2))(decode1)
decode3 = Conv2D(32, (3, 3), activation='relu', padding='same')(decode2)
decode4 = UpSampling2D((2, 2))(decode3)
decode5 = Conv2D(1, (3, 3), activation='relu', padding='same')(decode4)

model = models.Model(input_img, decode5)
```

Denoising Autoencoder

noisy images as inputs

original images as targets

```
model.compile(optimizer='RMSprop', loss='mean_squared_error')
history = model.fit(x_train_noisy, x_train, batch_size=128, epochs=50)
```

Epoch 1/50

60000/60000 [=====] - 92s 2ms/step - loss: 0.0293

Epoch 2/50

60000/60000 [=====] - 141s 2ms/step - loss: 0.0162

Epoch 3/50

60000/60000 [=====] - 130s 2ms/step - loss: 0.0144

•

•

•

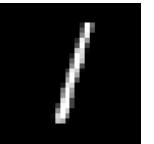
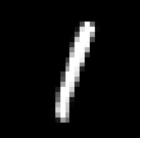
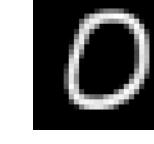
Epoch 49/50

60000/60000 [=====] - 143s 2ms/step - loss: 0.0106

Epoch 50/50

60000/60000 [=====] - 122s 2ms/step - loss: 0.0106

Results on the Test Set

Original	Noisy				Reconstructed			
								
								
								
								
								

Summary

- Autoencoder: generalizations of (linear) PCA.
- Autoencoder = Encoder + Decoder.

Summary

- Autoencoder: generalizations of (linear) PCA.
- Autoencoder = Encoder + Decoder.
- Training:
 - Inputs: original or noisy images.
 - targets: original images.
- Application 1: Dimensionality reduction (using the code vector).
- Application 2: Denoising.