

CS181: Computer Vision

Transformer Neural Network

Thursday, September 24th, 2026



Md Alimoor Reza

Associate Professor of Computer Science

Review: ResNet's main innovation is residual block

PyTorch
implementation calls it
downsampling with
1x1 convolution

- The ResNet block shown here, let's say, gets an input of size (32x56x56)

- After applying the first convolution inside the ResNet block, the output tensor's size will be (64x56x56)

Conv#1

$$W_2 = (56 - 3 + 2 * 1) / 1 + 1 = 56$$

$$C_2 = 64$$

- After applying the second convolution inside the ResNet block, the output tensor's size will be (64x56x56)

Conv#2

$$W_2 = (56 - 3 + 2 * 1) / 1 + 1 = 56$$

$$C_2 = 64$$

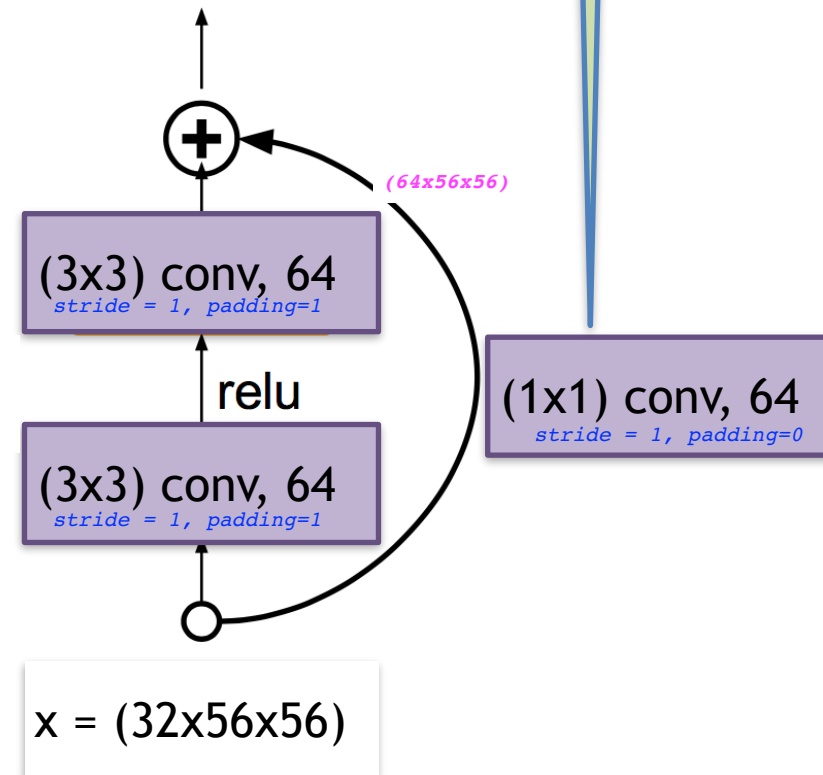
- Now, the ResNet block applies a **convolution operation** with 64 filters, each with a size of (1x1), to the input. The output of this operation produces a tensor of size (64x56x56).

Downsampling with conv

$$W_2 = (56 - 1 + 2 * 0) / 1 + 1 = 56$$

$$C_2 = 64$$

(64x56x56)



Kaiming He et al., [Deep Residual Learning for Image Recognition](#), CVPR 2015

Review: Fine-tuning AlexNet on an Arbitrary Dataset

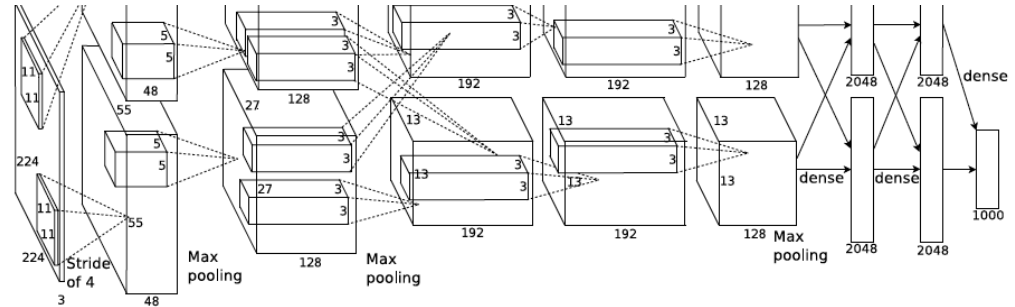
- Let's use one of the popular CNNs

- LeNet

- AlexNet

- VGG

- ResNet



UWS-DATASET

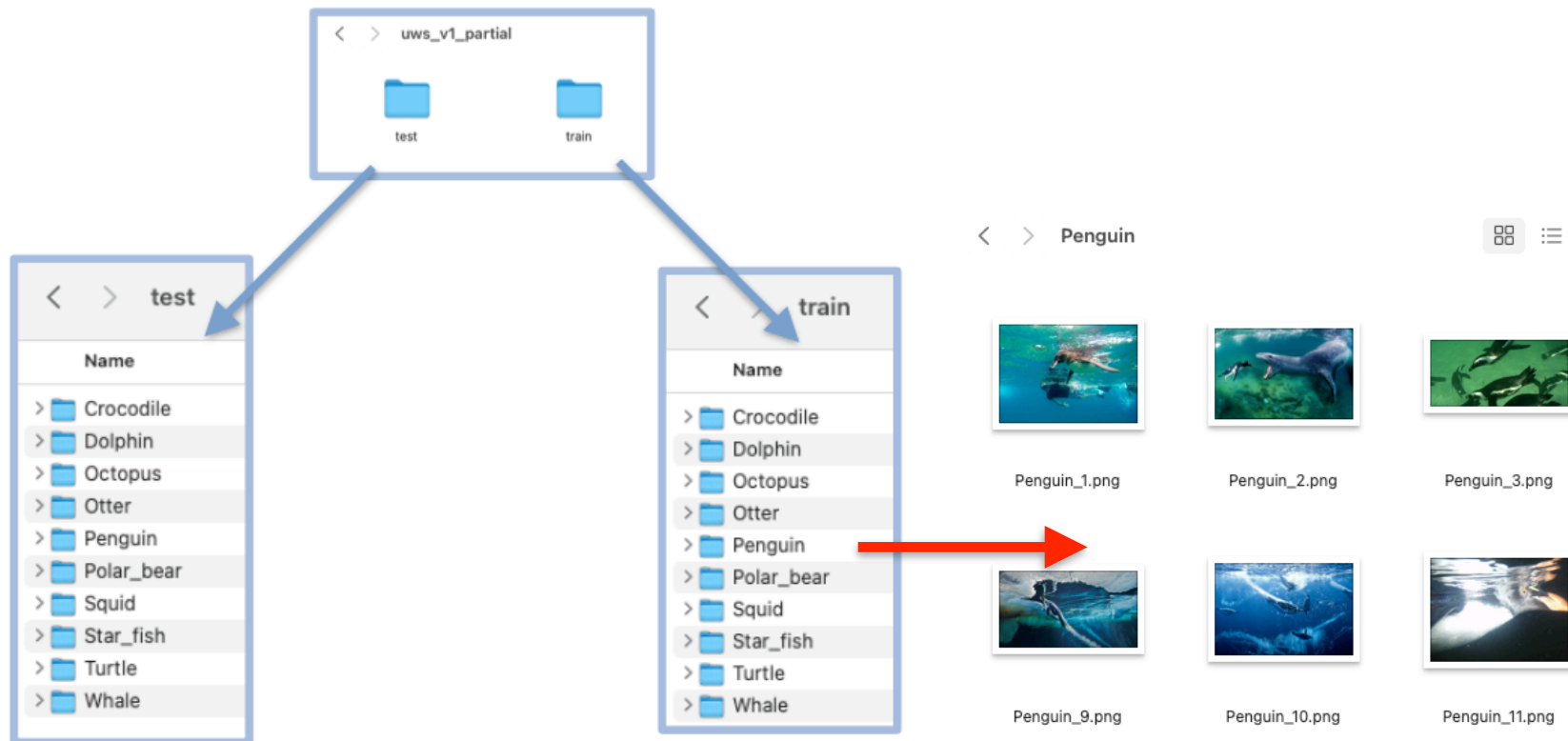


- Let's fine-tune AlexNet with a new dataset eg, UWS-DATASET

An animal dataset with 500 images
in Underwater environment

Review: Fine-tuning AlexNet on an Arbitrary Dataset

- Each folder ('train' and 'test') contains a set of images that will be used by our model during fine-tuning and testing, respectively



Review: Using Arbitrary Dataset

- Instead, when we need to use an arbitrary dataset, we can use [torchvision.datasets.ImageFolder class](#) to prepare the dataset. After that we can use [torch.utils.data.DataLoader class](#) to prepare Python iterable over our new training dataset and testing dataset separately.

```
from torchvision import transforms, datasets
from torch.utils.data import DataLoader

TEST_IMAGE_SIZE_W = 227
TEST_IMAGE_SIZE_H = 227
mean, std = get_imagenet_mean_std_normalized()
print(f"ImageNet: mean: {mean}, std: {std}")

# CNN architectures such as AlexNet, VGGNet, and ResNet has been pre-trained using the ImageNet dataset.
# You need to normalize each image with the given mean and standard deviation before doing the forward-pass on these networks.
transform = transforms.Compose([
    transforms.Resize((TEST_IMAGE_SIZE_W, TEST_IMAGE_SIZE_H)),
    transforms.ToTensor(),
    transforms.Normalize(mean=mean, std=std) # ImageNet: mean (R, G, B) and standard deviation (R, G, B)
])

train_dir = '/content/drive/MyDrive/cs181_fall26/datasets/uws_v1_partial/train'
test_dir = '/content/drive/MyDrive/cs181_fall26/datasets/uws_v1_partial/test'

train_dataset = datasets.ImageFolder(train_dir, transform=transform)
test_dataset = datasets.ImageFolder(test_dir, transform=transform)

N_train = len(train_dataset)
N_test = len(test_dataset)

number_of_classes = 10
print("Number of classes: ", number_of_classes)
print("Size of train set:", N_train)
print("Size of test set:", N_test)

train_dataloader = DataLoader(train_dataset, batch_size=batch_size_val, shuffle=True)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size_val, shuffle=False)
```

Review: Loading a Pre-trained AlexNet Model in PyTorch

- Import a pre-trained instance of AlexNet inside our Network class and make any other necessary changes as follows:

```
class AlexNet(nn.Module):

    def __init__(self, num_classes, pretrained=True):

        super(AlexNet, self).__init__()

        # download PyTorch's own implementation of AlexNet model trained on ImageNet
        net = models.alexnet(pretrained=True)

        # retained weightes for convolutional, pooling, linear layers from AlexNet
        self.features = net.features
        self.avgpool = net.avgpool
        self.classifier = net.classifier

        # IMPORTANT: "If you need to fine-tune this network for your own dataset
        # the simplest modification is to replace the last layer in self.classifier
        # the updated AlexNet has the desired number of output classes: 'num_classes'
        self.classifier[-1] = nn.Linear(4096, num_classes) # only this last layer is modified

    def forward(self, x):

        print("shape of input: ", x.shape)
        x = self.features(x)
        print("output shape (self.features): ", x.shape)
        x = self.avgpool(x)
        print("output shape (self.avgpool): ", x.shape)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        print("output shape (self.classifier): ", x.shape)
        return x
```

Announcements

- Notebook #3: Underwater Animal Classification
 - It's a 10-class image recognition problem.
 - to submit, download the `.ipynb` file from Colab
 - directly upload to Blackboard
 - [due by October 1st](#)
- Quiz#1 will be released next Tuesday
 - you will be able to take it offline on Blackboard
 - should be taken [within a fixed-hour window](#), during which you are expected to complete the quiz individually. You are free to start anytime within the seven-day period starting next Tuesday.
 - released on [September 29th](#) and [due by October 7th](#)

Transformers



Today's agenda

- Transformer
 - Transfer learning is possible
 - New type of network architecture besides convolutional neural network
- Vision Transformer (ViT)

Transformers



- In 2017, a new mechanism is introduced for context learning called **attention mechanism**
 - more precisely, **self-attention**
- It takes less time to train **advantage**
- Transfer learning on a new task is possible **advantage**
- In subsequent years, it revolutionized the field of AI

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaier@google.com

Illia Polosukhin* †
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

[Attention is all you need - NeurIPS'2017](#)

So why do we need this complicated attention?

It's all about “context”

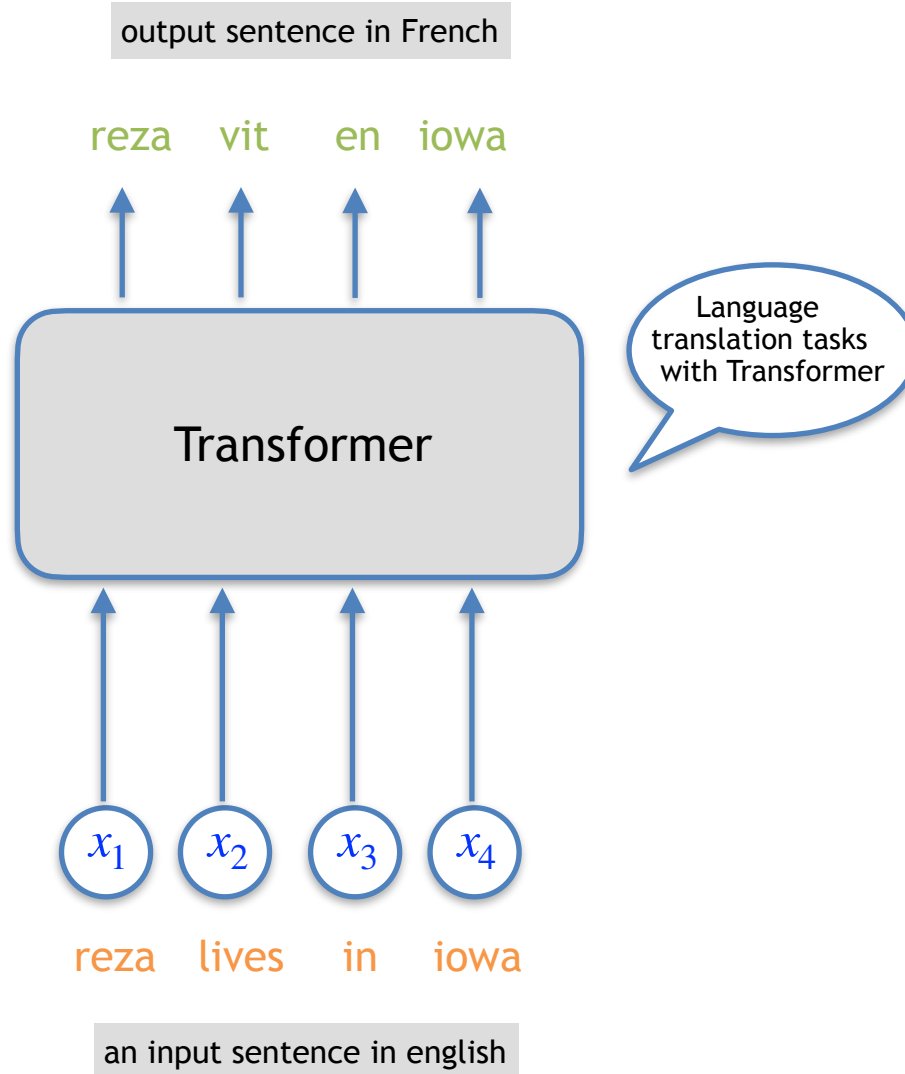
- Consider a language model trying to predict the next word based on the previous ones. Let's predict the last word to this sequence:
 - "The clouds are in the ?"
- We can guess from recent information that it will be the name of a language; we need the context of France to make a prediction.
 - "I grew up in France, I speak fluent ?"

So why do we need this complicated attention?

It's all about “context”

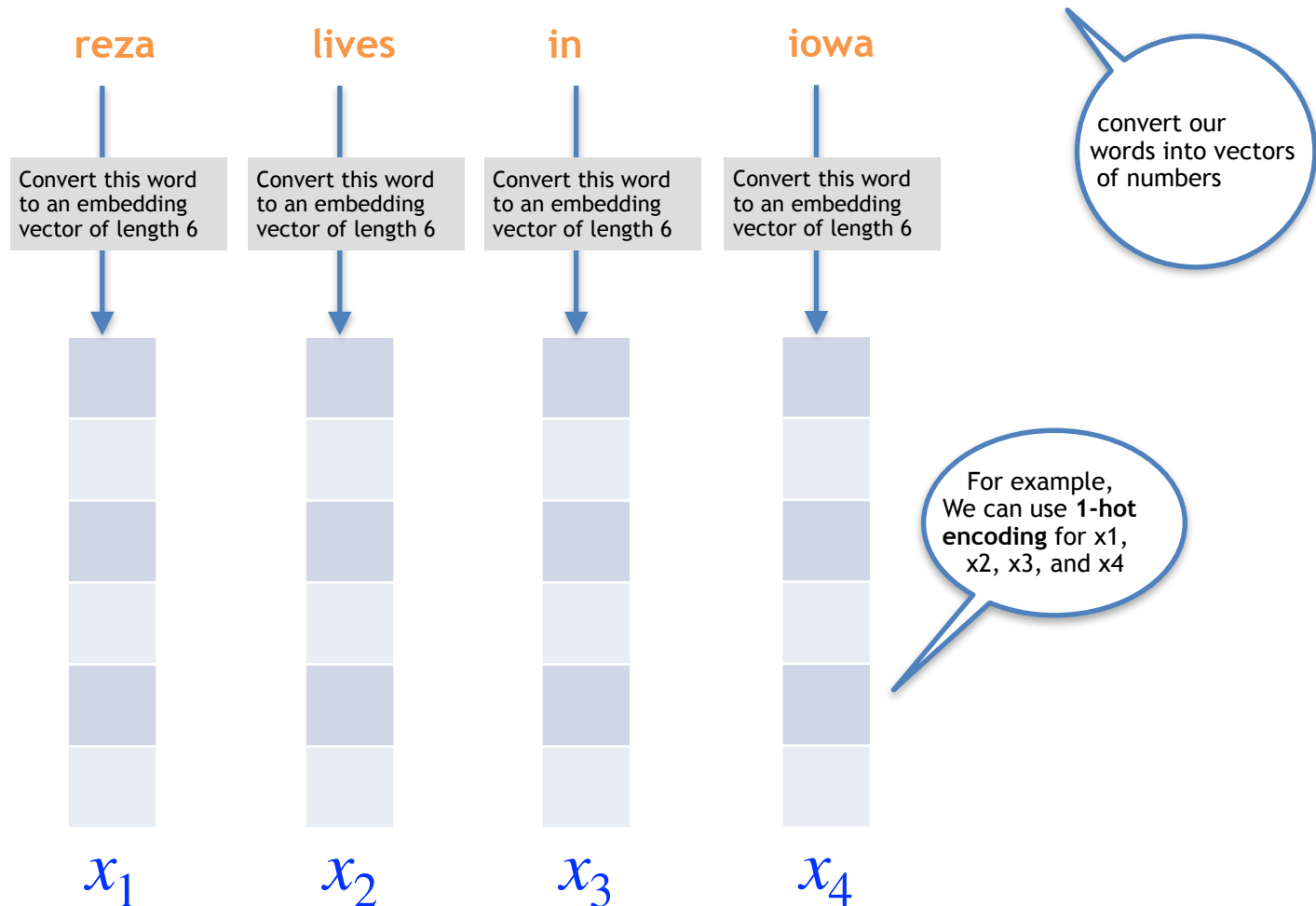
- Consider a language model trying to predict the next word based on the previous ones. Let's predict the last word to this sequence:
 - "The clouds are in the sky"
- We can guess from recent information that it will be the name of a language; we need the context of France to make a prediction.
 - "I grew up in France, I speak fluent french"

Transformers



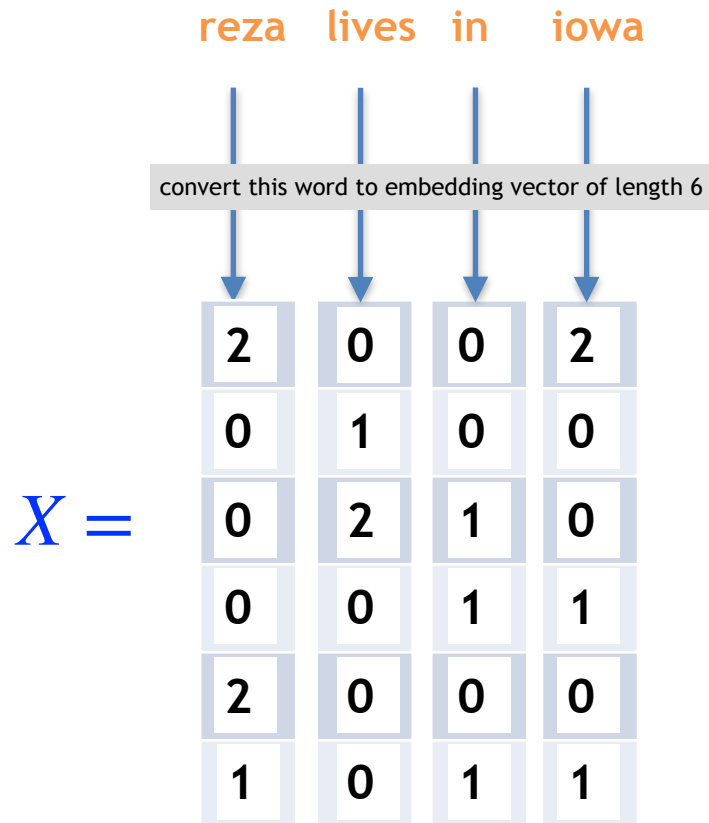
Self-Attention

- Let's find out how to calculate the **attention mechanism** in a toy example
- Let's calculate attention for our sentence: “reza lives in iowa”



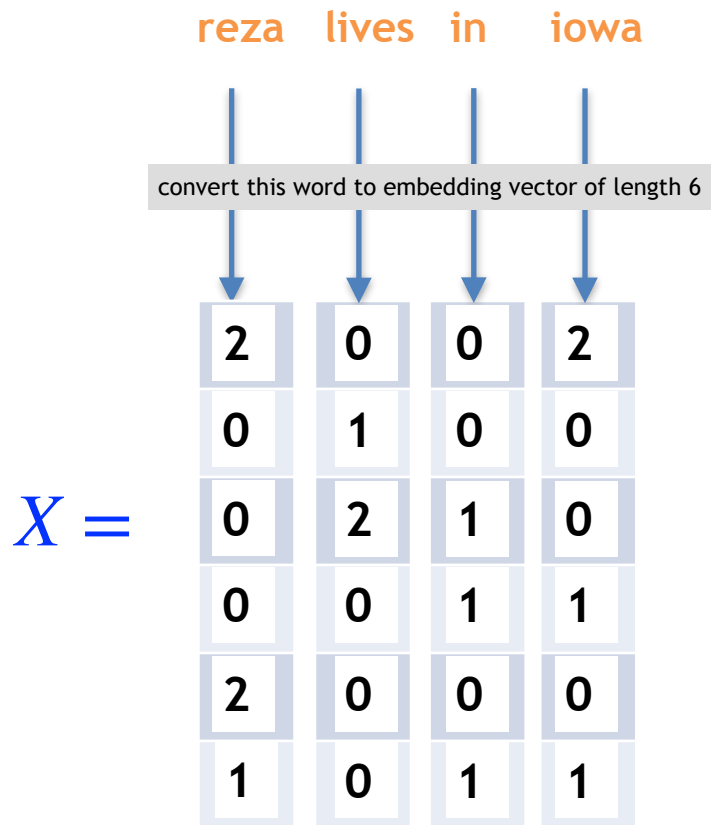
Self-Attention

- Let's concatenate the vectors x_1 , x_2 , x_3 , and x_4 and build a matrix X from these.



Self-Attention

- We compute three new matrices Q , K , and V with the help of three weight matrices W_Q , W_K , and W_V and our input matrix X



$$Q = W_Q X$$

$$K = W_K X$$

$$V = W_V X$$

Self-Attention

- For example, these three new weight matrices W_Q , W_K , and W_V could look as follows:

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

W_Q

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

W_K

10	0	0	0	0	0
0	0	0	10	0	0
0	10	0	0	0	0

W_V

- These three weight matrices (W_Q , W_K , and W_V) are **learned** during training

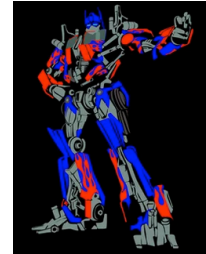
Self-Attention

- These three weight matrices (W_Q , W_K , and W_V) are **learned** during training

W_Q

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

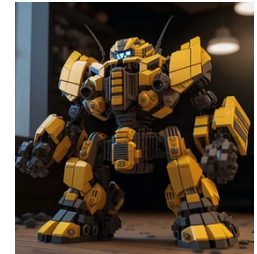
Optimus prime



W_K

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

Bumble Bee



W_V

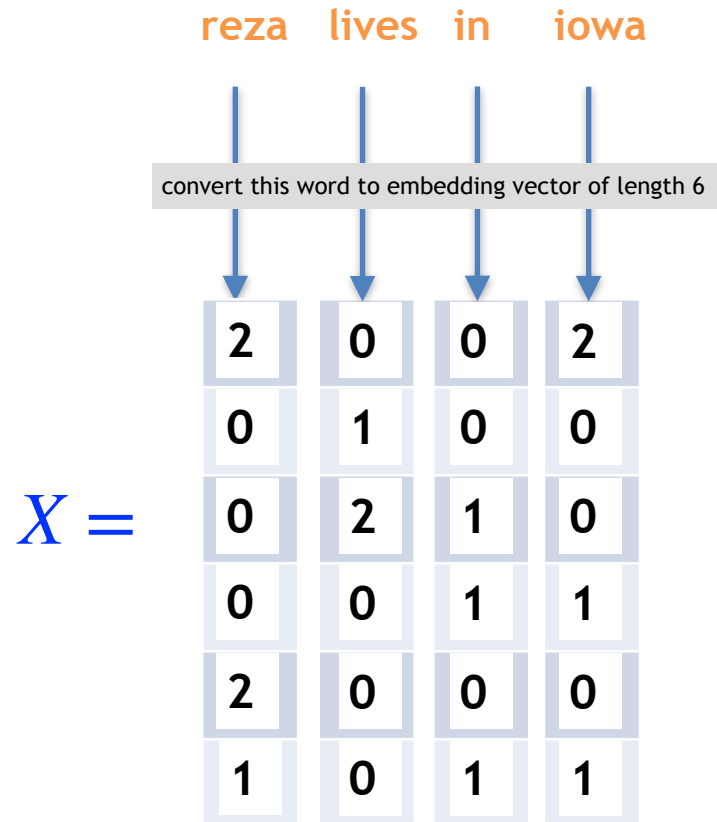
10	0	0	0	0	0
0	0	0	10	0	0
0	10	0	0	0	0

Ironhide



Self-Attention

- Now, let's compute three new matrices Q , K , and V with the help of three weight matrices W_Q , W_K , and W_V . We will multiply input matrix X with each of them in turn as follows:



$$Q = W_Q X$$

$$K = W_K X$$

$$V = W_V X$$

In-class activity: Self-Attention

- Let's multiply matrices W_Q and X to derive (by matrix-matrix multiplication) Q as follows:

$$Q = W_Q X$$

$=$

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

W_Q

	reza	lives	in	iowa
	2	0	0	2
	0	1	0	0
	0	2	1	0
	0	0	1	1
	2	0	0	0
	1	0	1	1

X

$=$

?	?	?	?
?	?	?	?
?	?	?	?

In-class activity: Self-Attention

- Let's multiply matrices W_K and X to derive (by matrix-matrix multiplication) K as follows:

$$K = W_K X$$

$=$

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

W_K

reza lives in iowa

2	0	0	2
0	1	0	0
0	2	1	0
0	0	1	1
2	0	0	0
1	0	1	1

X

$=$

?	?	?	?
?	?	?	?
?	?	?	?

In-class activity: Self-Attention

- Let's multiply matrices W_V and X to derive (by matrix-matrix multiplication) V as follows:

$$V = W_V X$$

$=$

10	0	0	0	0	0
0	0	0	10	0	0
0	10	0	0	0	0

W_V

reza lives in iowa

2	0	0	2
0	1	0	0
0	2	1	0
0	0	1	1
2	0	0	0
1	0	1	1

X

$=$

?	?	?	?
?	?	?	?
?	?	?	?

Self-Attention

- Finally, attention is calculated using Q , K , and V matrices using the following equation:

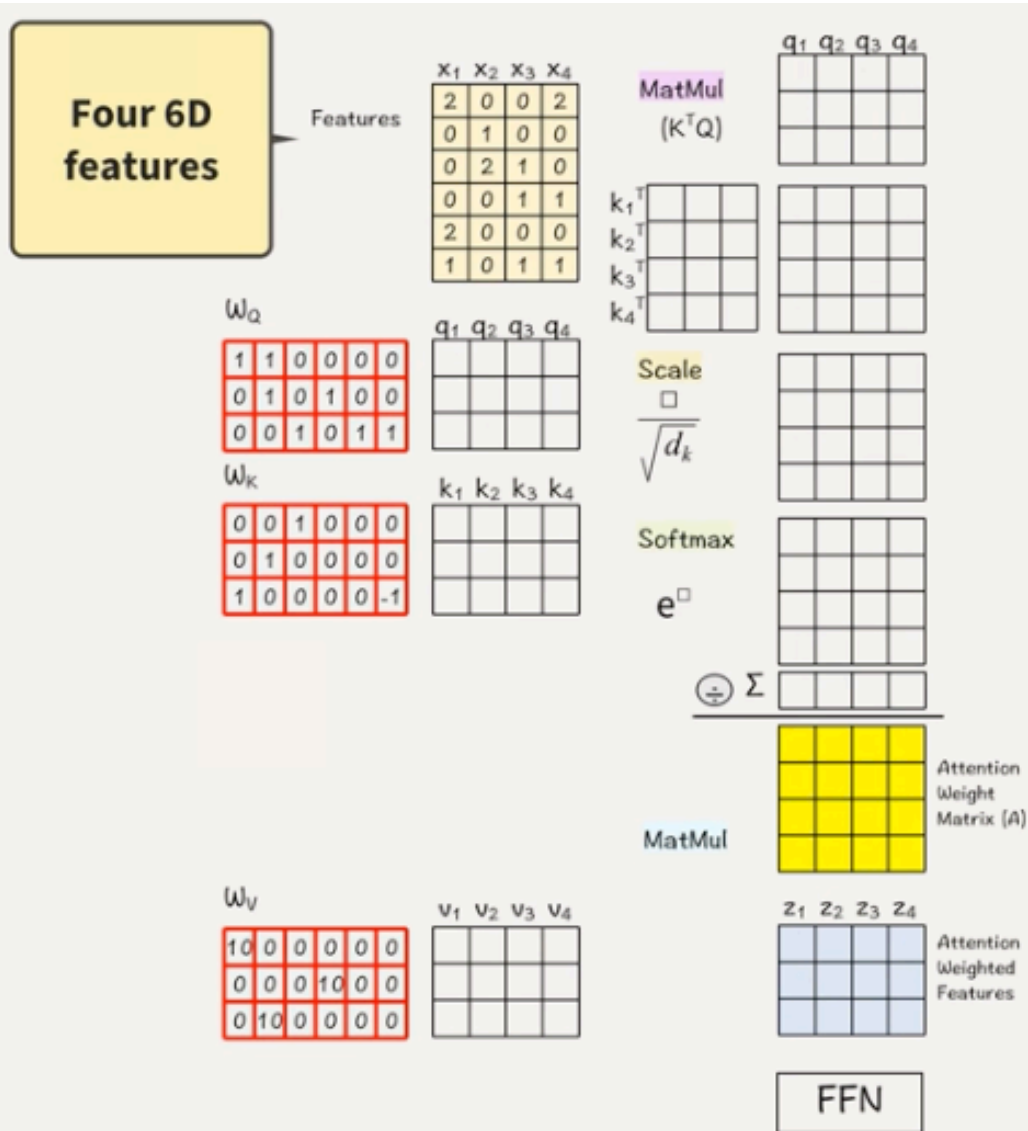
$$\text{softmax} \left(\frac{Q \times K^T}{\sqrt{d_k}} \right) V$$

=

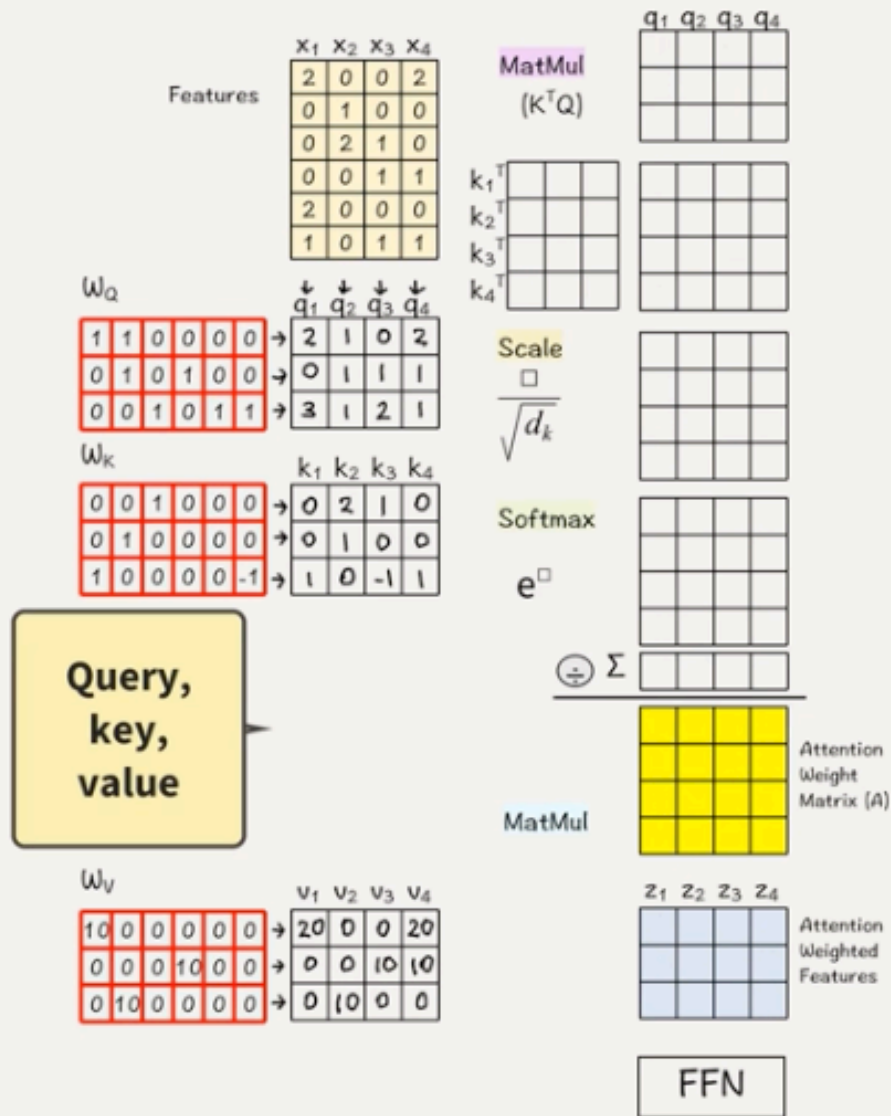
Attention weighted features

The diagram illustrates the Self-Attention calculation. It shows the Q matrix (green 4x4 grid) multiplied by the K^T matrix (purple 4x4 grid), divided by the square root of the key dimension $\sqrt{d_k}$, and then multiplied by the V matrix (blue 4x4 grid). The result is an orange 4x4 grid labeled 'Attention weighted features'.

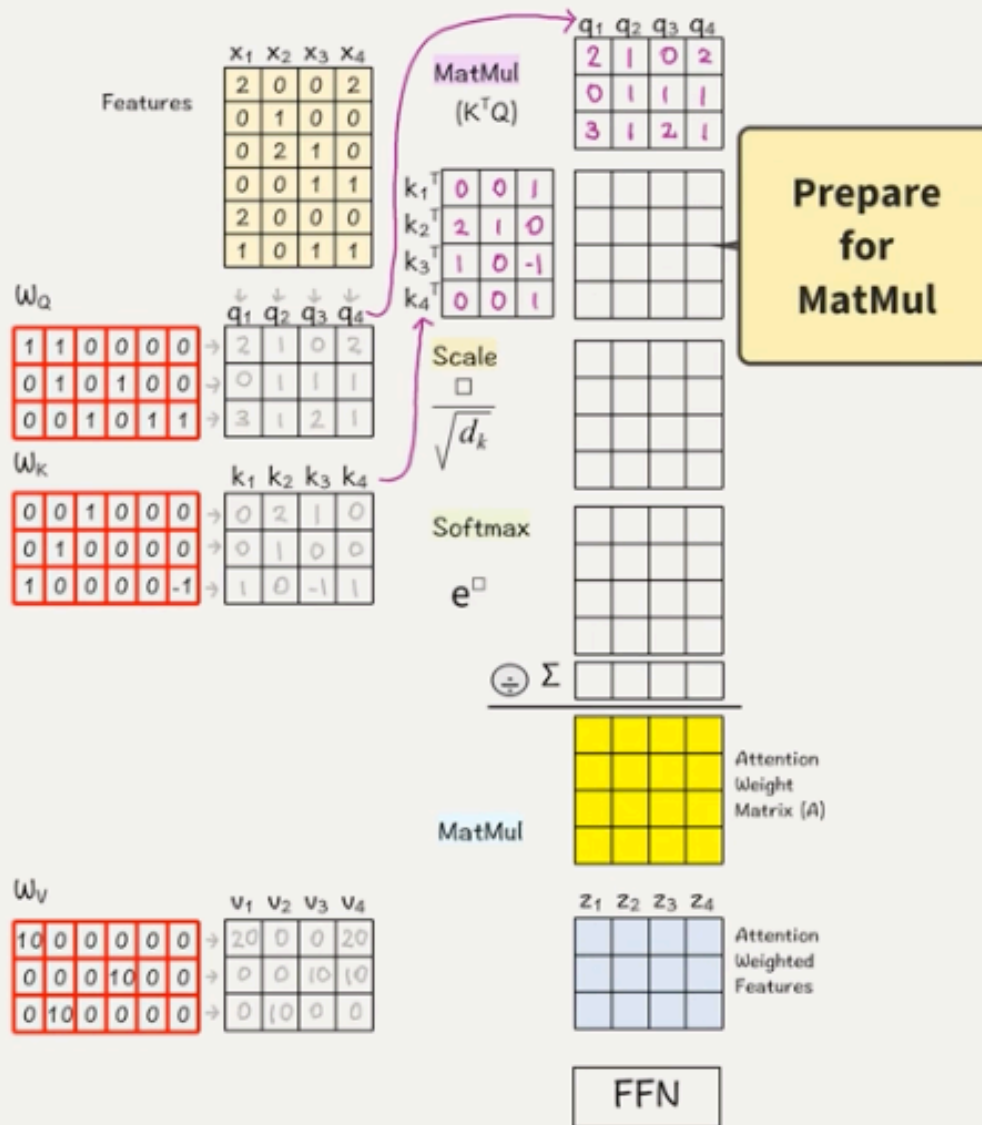
Self-Attention



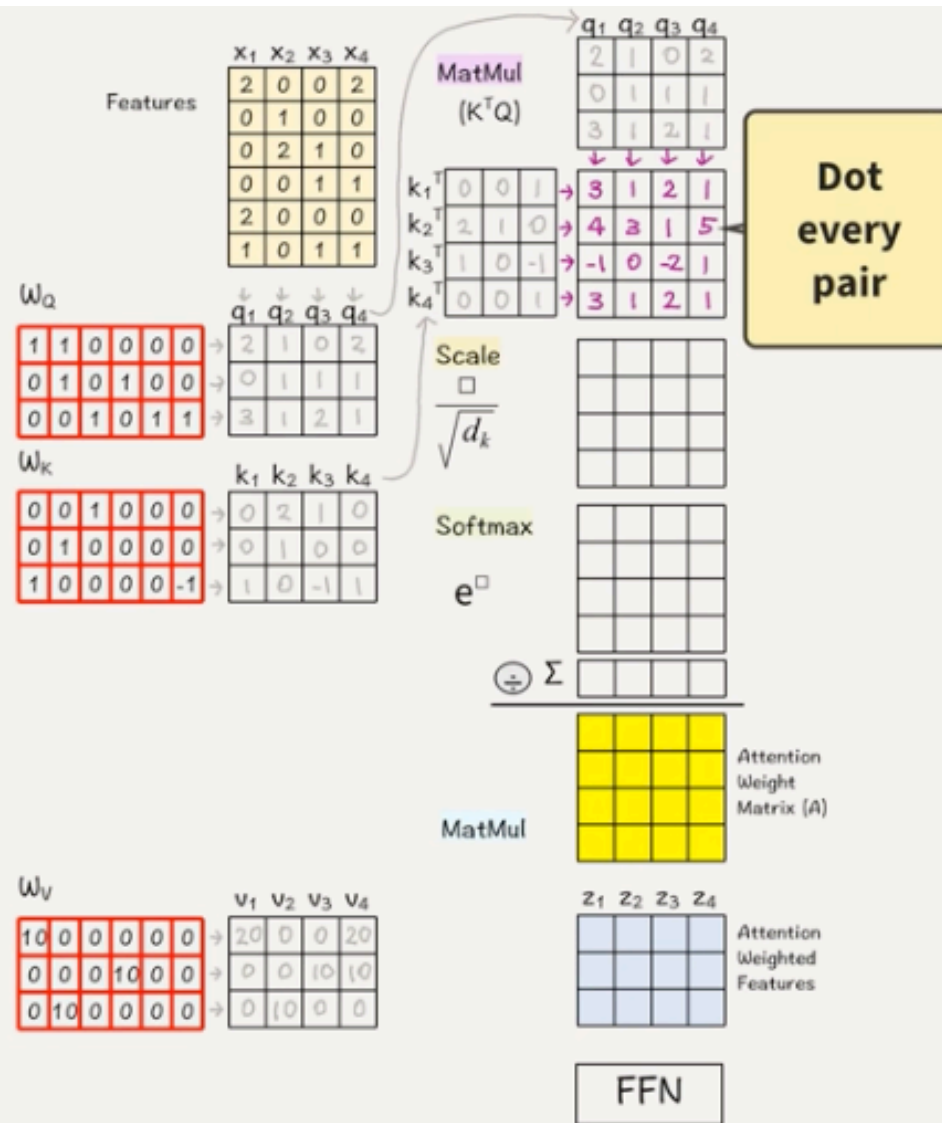
Self-Attention



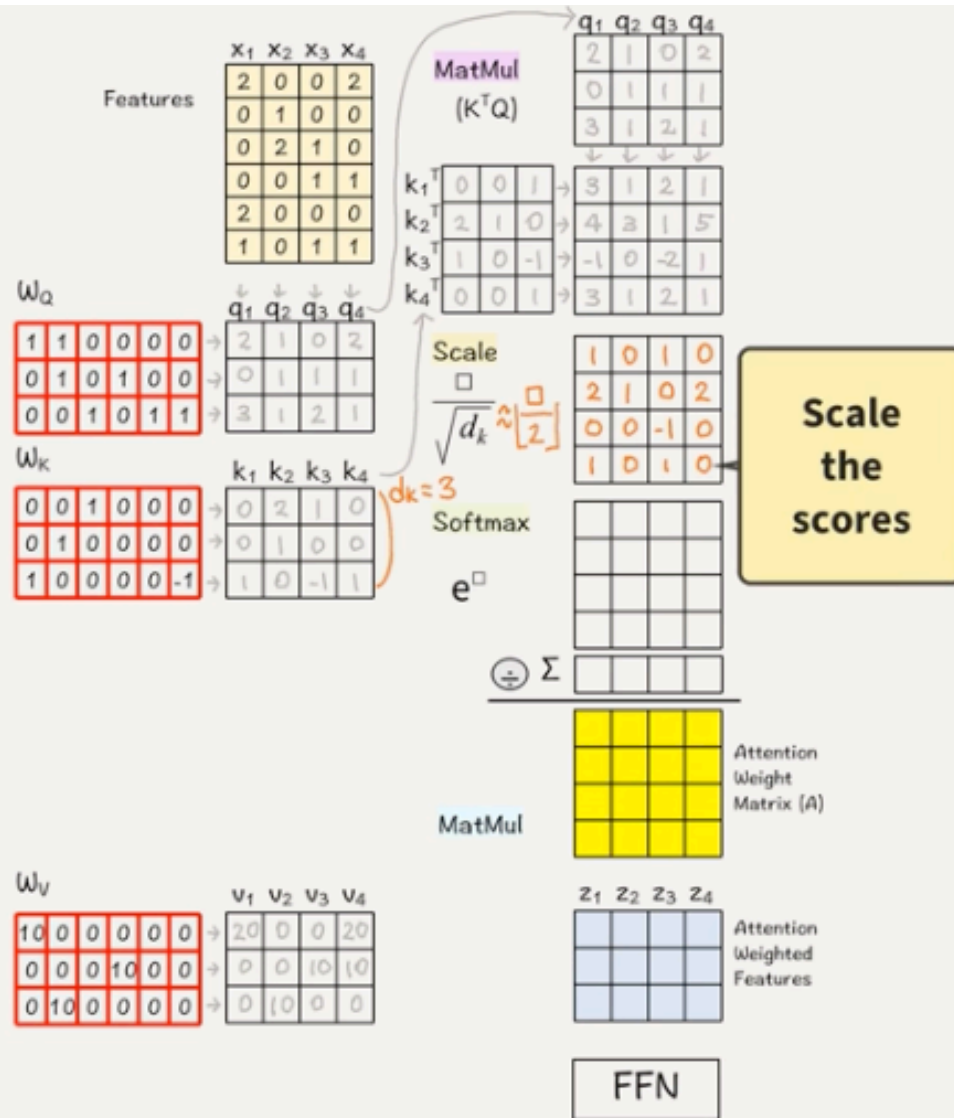
Self-Attention



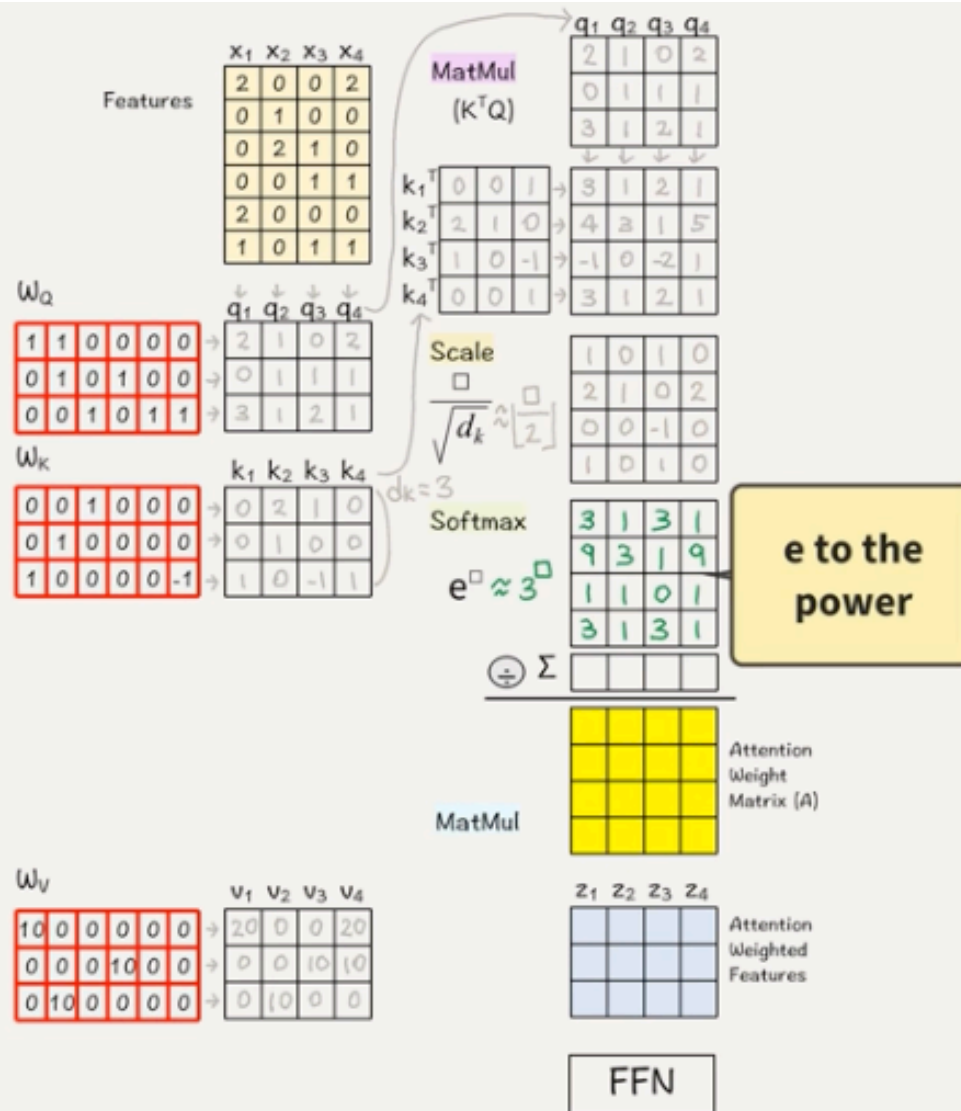
Self-Attention



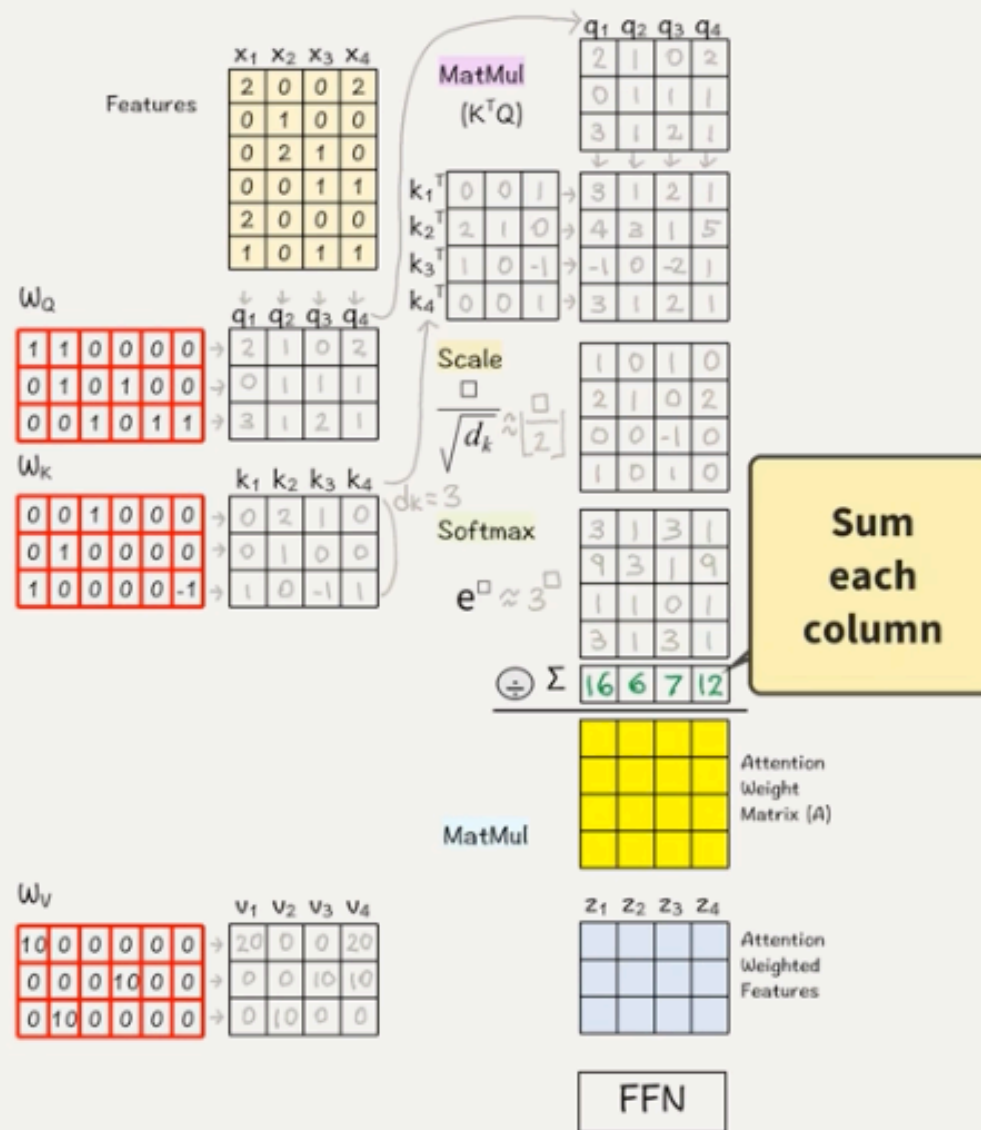
Self-Attention



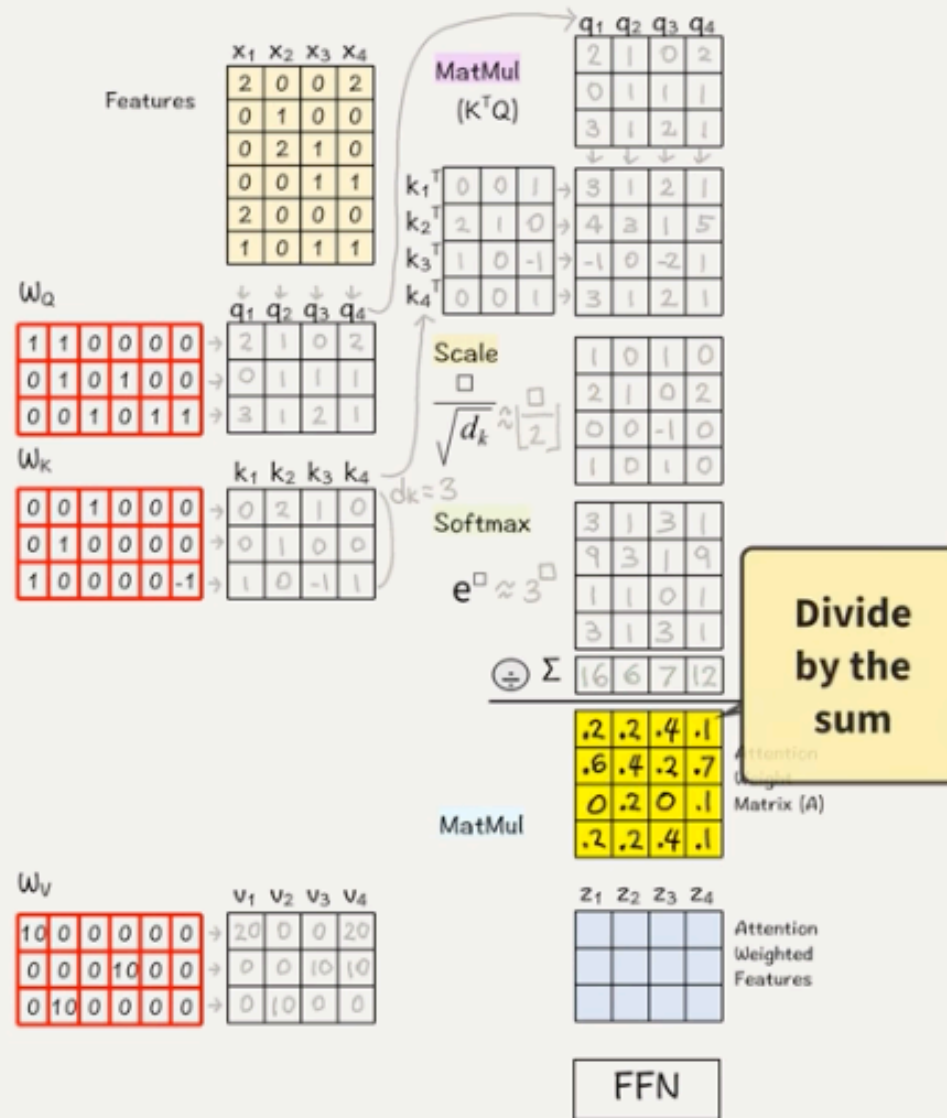
Self-Attention



Self-Attention



Self-Attention



Self-Attention

