Chapter 2:

Mathematical building blocks of neural networks

Outlines

1. Data representation of neural networks
2. Tensor operations
3. Gradient-based optimization

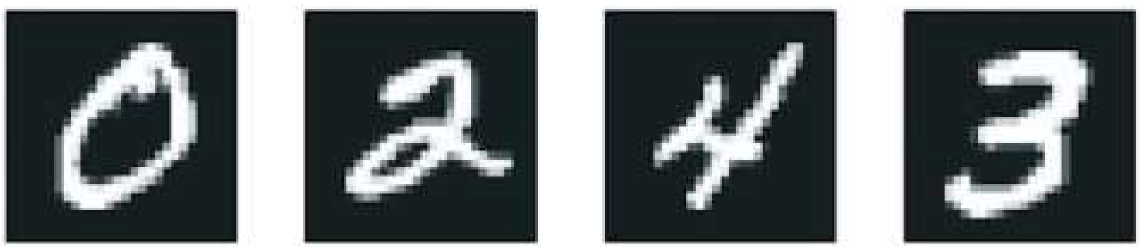
Data representation of neural networks

1. Overview of neural networks:
   - Review an example of using Keras to build a neural network for classifying handwritten digits.
     - ✓ The task involves grayscale images of digits (28x28 pixels) divided into 10 categories (0–9) using the MNIST dataset.
     - ✓ The MNIST is a classic machine learning dataset created in the 1980s, contains 60,000 training images and 10,000 test images.
     - ✓ The MNIST is widely considered the "Hello World" of deep learning, frequently used to test algorithms.

Data representation of neural networks

1. Overview of neural networks:

   - MNIST sample digits



Data representation of neural networks

1. Overview of neural networks:
   - Loading the MNIST dataset in Keras

```
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

   - The network architecture

```
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

   - The compilation step

```
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
```

Data representation of neural networks

1. Overview of neural networks:
   - Preparing the image data

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype("float32") / 255
```

   - Fit the model

```
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

   - Using the model to make predictions

```
>>> test_digits = test_images[0:10]
>>> predictions = model.predict(test_digits)
>>> predictions[0]
array([1.0726176e-10, 1.6918376e-10, 6.1314843e-08, 8.4106023e-06,
       2.9967067e-11, 3.0331331e-09, 8.3651971e-14, 9.9999106e-01,
       2.6657624e-08, 3.8127661e-07], dtype=float32)
```

Data representation of neural networks

1. Overview of neural networks:
   - Evaluating the model on new data

```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> print(f"test_acc: {test_acc}")
test_acc: 0.9785
```

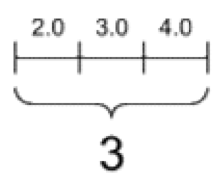
Data representation of neural networks

2. Tensors
   - ✓ Tensor are multidimensional arrays of uniform type (called dtype)).
   - ✓ Supported dtypes according to tf.dtypes.DType.

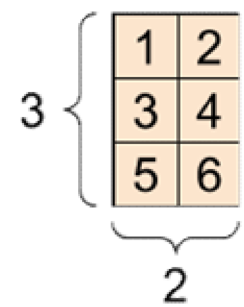
Scalar: []

4

Array, vector (1 dimensional) format: [3]



Matrix form (2 dimensions): [3, 2]



   - ✓ Tensor usually contains data of type int, float, can also be string or complex numbers
   - ✓ Tensors are similar to NumPy's arrays.
   - ✓ Cannot update the contents of a tensor, can only create a new tensor.

Data representation of neural networks

2. Tensors
   - Scalars (rank-0 tensors)
     - ✓ A tensor that contains only one number is called a scalar (or scalar tensor, or rank-0 tensor, or 0D tensor).
     - ✓ In NumPy, a float32 or float64 number is a scalar tensor (or scalar array).
     - ✓ For example a NumPy scalar:

```
>>> import numpy as np
>>> x = np.array(12)
>>> x
array(12)
>>> x.ndim
0
```

Data representation of neural networks

2. Tensors
   - ❖ Vectors (rank-1 tensors)
     - ✓ An array of numbers is called a vector, or rank-1 tensor, or 1D tensor.
     - ✓ A rank-1 tensor is said to have exactly one axis.
     - ✓ For example a NumPy vector:

```
>>> x = np.array([12, 3, 6, 14, 7])
>>> x
array([12, 3, 6, 14, 7])
>>> x.ndim
1
```

Data representation of neural networks

2. Tensors
   - ❖ Matrices (rank-2 tensors)
     - ✓ An array of vectors is a matrix, or rank-2 tensor, or 2D tensor.
     - ✓ A matrix has two axes (often referred to as rows and columns).
     - ✓ For example a NumPy matrix:

```
>>> x = np.array([[5, 78, 2, 34, 0],
                  [6, 79, 3, 35, 1],
                  [7, 80, 4, 36, 2]])
>>> x.ndim
2
```

Data representation of neural networks

2. Tensors
   - ❖ Rank-3 and higher-rank tensors
     - ✓ If we pack such matrices in a new array, we obtain a rank-3 tensor (or 3D tensor), which we can visually interpret as a cube of numbers.
     - ✓ For example a NumPy rank-3 tensor:

```
>>> x = np.array([[[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]]])
>>> x.ndim
3
```

Data representation of neural networks

2. Tensors
   - Key attributes
     - A tensor is defined by three key attributes:
       - ✓ *Number of axes* (rank): tensor's ndim in Python libraries such as NumPy or TensorFlow.
       - ✓ *Shape*: This is a tuple of integers that describes how many dimensions the tensor has along each axis.
       - ✓ *Data type*: called dtype in Python libraries. A tensor's type could be float16, float32, float64, uint8, and so on.

Data representation of neural networks

3. Tensor manipulation with Numpy:
   - We can select a specific digit alongside the first axis using the syntax train_images[i].
   - Selecting specific elements in a tensor is called tensor slicing.
   - The following example selects digits #10 to #100 (#100 isn't included) and puts them in an array of shape (90, 28, 28):

```
>>> my_slice = train_images[10:100]
>>> my_slice.shape
(90, 28, 28)
```

Data representation of neural networks

3. Tensor manipulation with Numpy (cont):
   - Equivalent writing:

```
>>> my_slice = train_images[10:100, :, :]
>>> my_slice.shape
(90, 28, 28)
```

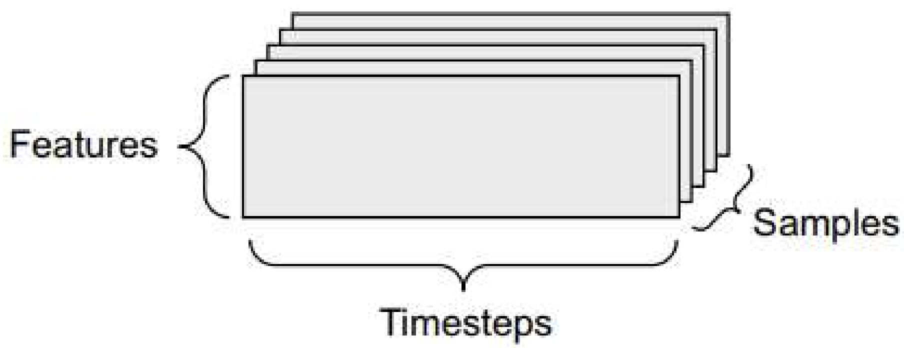
```
>>> my_slice = train_images[10:100, 0:28, 0:28]
>>> my_slice.shape
(90, 28, 28)
```

Data representation of neural networks

4. Real-world examples of data tensors:
   - The data we'll manipulate will almost always fall into one of the following categories:
     - ✓ *Vector data—Rank-2* tensors of shape (samples, features), where each sample is a vector of numerical attributes ("features").
     - ✓ For example:
     - ❑ An actuarial dataset of people, where we consider each person's age, gender, and income. Each person can be characterized as a vector of 3 values, and thus an entire dataset of 100,000 people can be stored in a rank-2 tensor of shape (100000, 3)

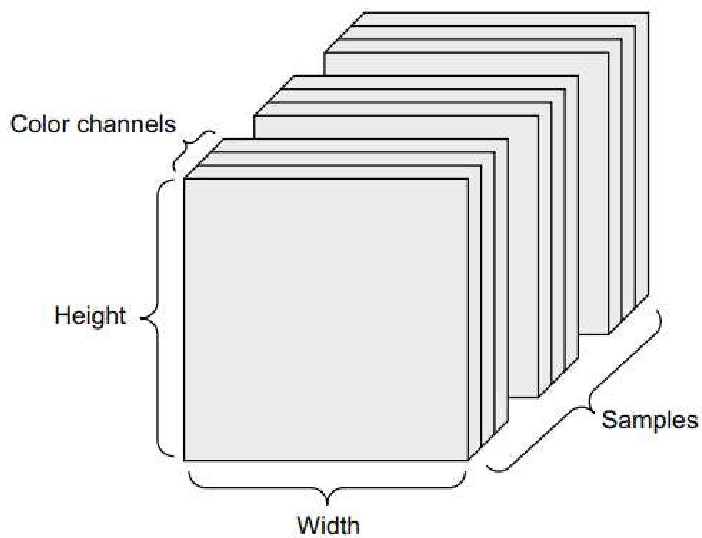
Data representation of neural networks

4. Real-world examples of data tensors:
   - The data we'll manipulate will almost always fall into one of the following categories:
     - ✓ *Timeseries data or sequence data—Rank-3* tensors of shape (samples, timesteps, features), where each sample is a sequence (of length timesteps) of feature vectors.



Data representation of neural networks

4. Real-world examples of data tensors:
   - The data we'll manipulate will almost always fall into one of the following categories:
     - ✓ *Images—Rank-4* tensors of shape (samples, height, width, channels), where each sample is a 2D grid of pixels, and each pixel is represented by a vector of values ("channels")



Data representation of neural networks

4. Real-world examples of data tensors:
   - The data we'll manipulate will almost always fall into one of the following categories:
     - ✓ *Video—Rank-5* tensors of shape (samples, frames, height, width, channels), where each sample is a sequence (of length frames) of images.
     - ✓ For instance, a 60-second, 144 × 256 YouTube video clip sampled at 4 frames per second would have 240 frames. A batch of four such video clips would be stored in a tensor of shape (4, 240, 144, 256, 3).
     - ✓ That's a total of 106,168,320 values

Data representation of neural networks

1. Basic operations:
   - We have three tensor operations here:
     - ✓ A dot product (dot) between the input tensor and a tensor named W
     - ✓ An addition (+) between the resulting matrix and a vector b
     - ✓ A relu operation: relu(x) is max(x, 0); “relu” stands for “rectified linear unit”

Data representation of neural networks

1. Basic operations:
   - Example: relu

```
keras.layers.Dense(512, activation="relu")
```

     - ✓ This layer can be interpreted as a function, which takes as input a matrix and returns another matrix—a new representation for the input tensor

Data representation of neural networks

1. Basic operations:
   - Example: dot and addition
     - ✓ This function is as follows (where W is a matrix and b is a vector, both attributes of the layer):

```
output = relu(dot(input, W) + b)
```

Tensor operations

❖ Example 1:

```
import tensorflow as tf


rank_1_tensor = tf.constant([2.0, 3.0, 4.0])
print(rank_1_tensor)
```

```
Tensor("Const:0", shape=(3,), dtype=float32)
```

❖ Example 2:

```
rank_2_tensor = tf.constant([[1, 2],
                             [3, 4],
                             [5, 6]], dtype=tf.float16)
print(rank_2_tensor)
```

```
Tensor("Const:0", shape=(3, 2), dtype=float16)
```

❖ It is possible to convert a tensor to a NumPy array using the np.array or tensor.numpy methods:

```
np.array(rank_2_tensor)
```

```
rank_2_tensor.numpy()
```

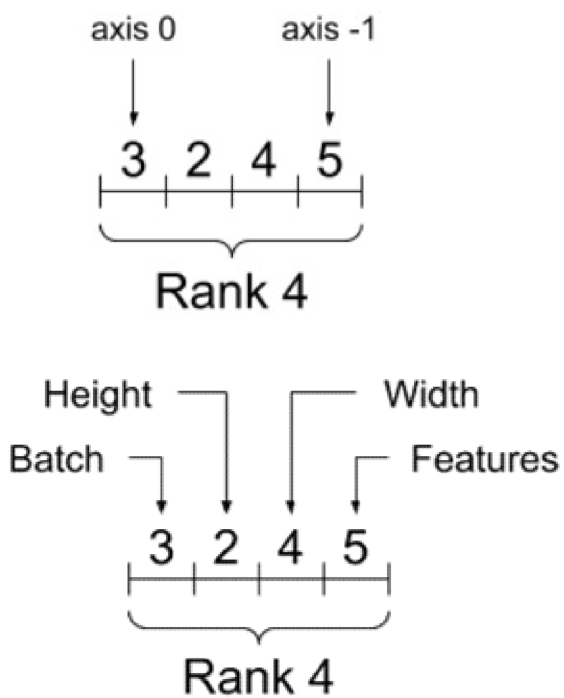
Tensor operations

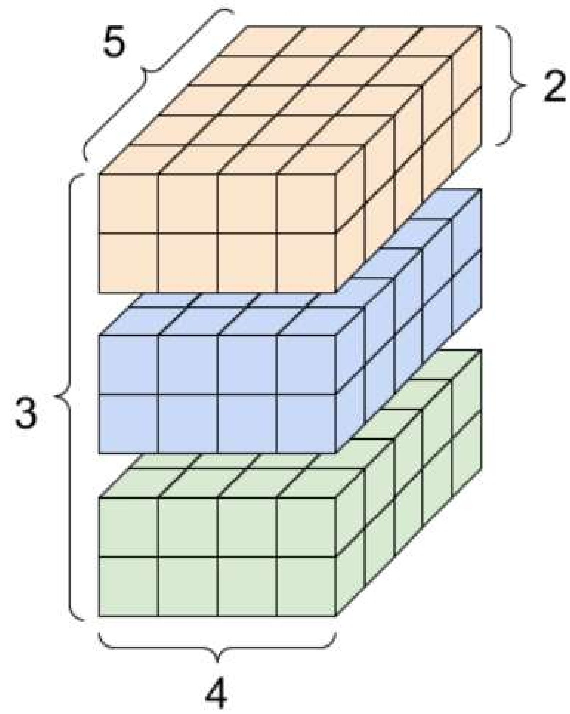
❖ Shape

✓ Example:

```
rank_4_tensor = tf.zeros([3, 2, 4, 5])
print(rank_4_tensor)
```

```
Tensor("zeros:0", shape=(3, 2, 4, 5), dtype=float32)
```





Tensor operations

- Shape

Shape	Rank	Dimension-Number
[]	0	0-D
[D0]	1	1-D
[D0, D1]	2	2-D
[D0, D1, D2]	3	3-D
… … …		
[D0, D1, ... Dn]	n	n-D

Tensor operations

❖ Shape

✓ Retrieve the properties of a shape :

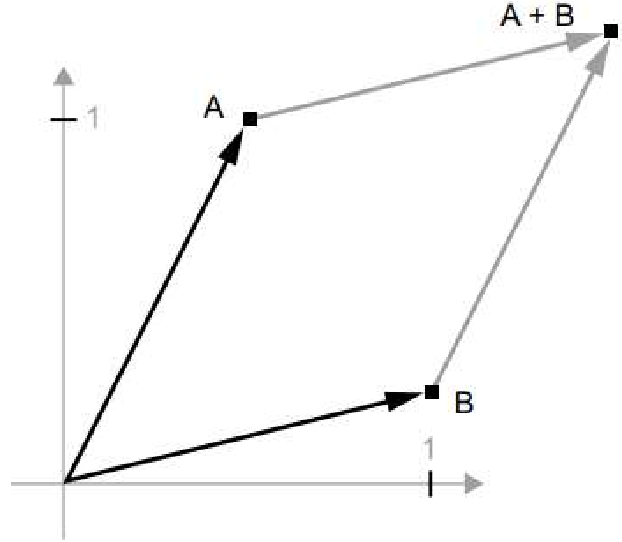
```
rank_4_tensor = tf.zeros([3, 2, 4, 5])
print("Type of every element:", rank_4_tensor.dtype)
print("Shape of tensor:", rank_4_tensor.shape)
print("Elements along axis 0 of tensor:", rank_4_tensor.shape[0])
print("Elements along the last axis of tensor:", rank_4_tensor.shape[-1])
```

```
Type of every element: <dtype: 'float32'>
Shape of tensor: (3, 2, 4, 5)
Elements along axis 0 of tensor: 3
Elements along the last axis of tensor: 5
```

Tensor operations

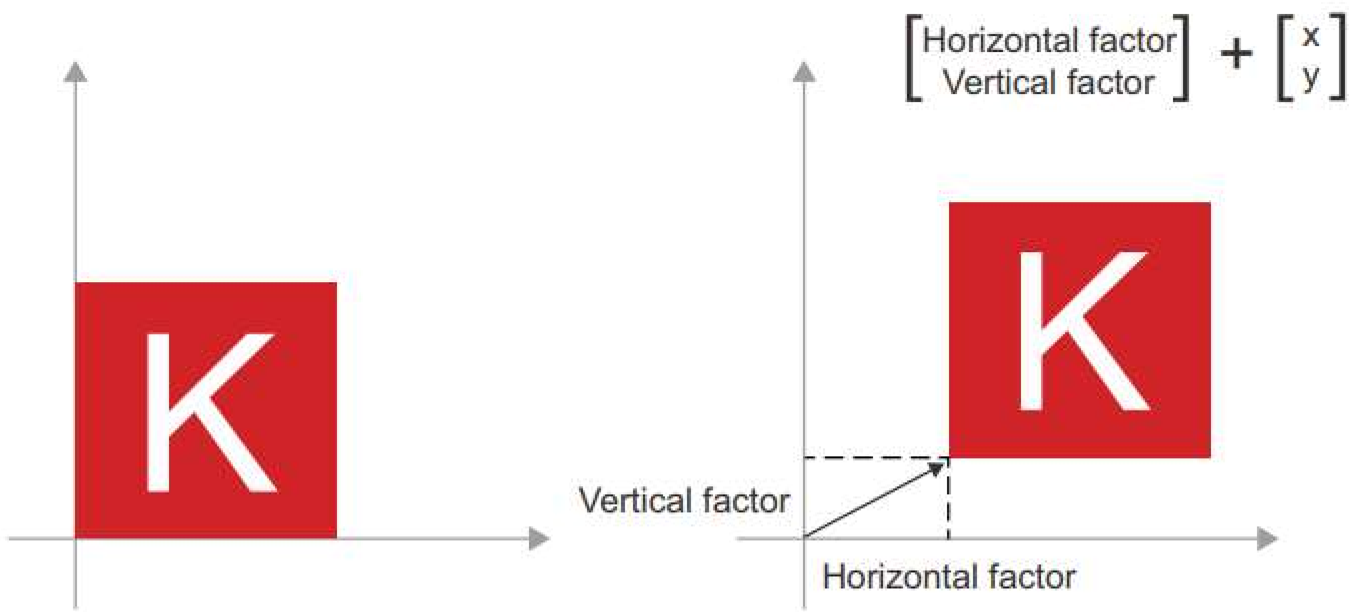
2. Geometric interpolation of tensor operations:
   - The contents of the tensors manipulated by tensor operations can be interpreted as coordinates of points in a geometric space.
   - All tensor operations have a geometric interpretation.
   - For instance, the addition:

```
A = [0.5, 1]
B = [1, 0.25]
```



Tensor operations

2. Geometric interpolation of tensor operations (cont):
   - Tensor addition thus represents the action of translating an object by a certain amount in a certain direction.
   - Tensor addition moves the object without distorting it.

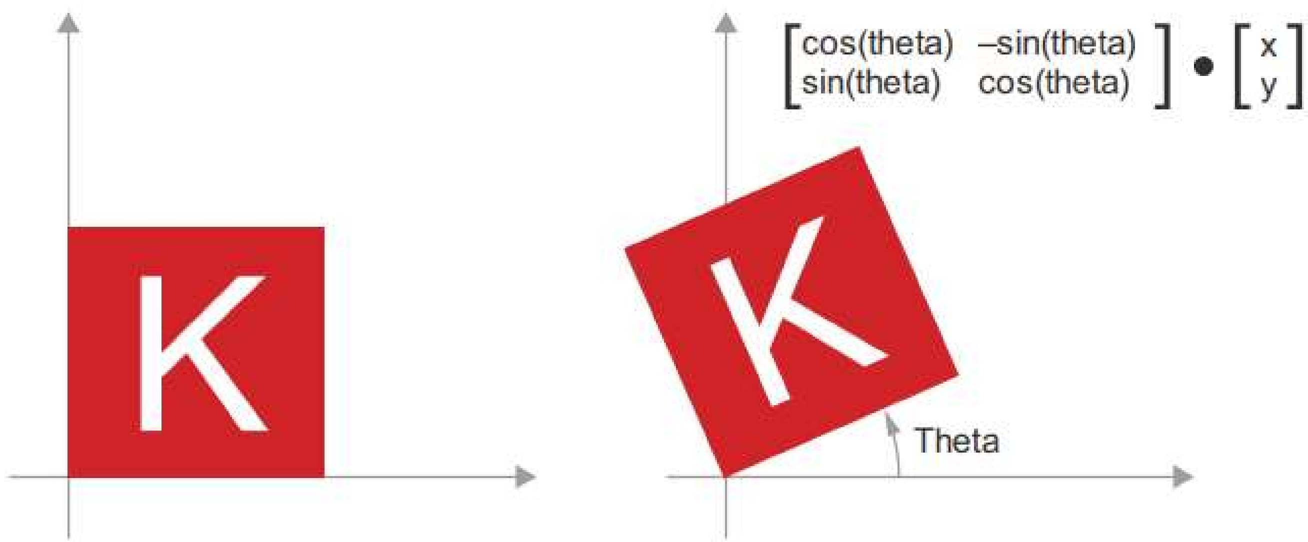


Tensor operations

2. Geometric interpolation of tensor operations (cont):
   - Elementary geometric operations such as translation, rotation, scaling, skewing, and so on can be expressed as tensor operations.
   - A few examples:
     - ✓ *Translation*: adding a vector to a point will move the point by a fixed amount in a fixed direction. Applied to a set of points (such as a 2D object), this is called a "translation."
     - ✓ *Rotation*: A counterclockwise rotation of a 2D vector by an angle theta can be achieved via a dot product with a 2 × 2 matrix R = [[cos(theta), -sin(theta)], [sin(theta), cos(theta)]].

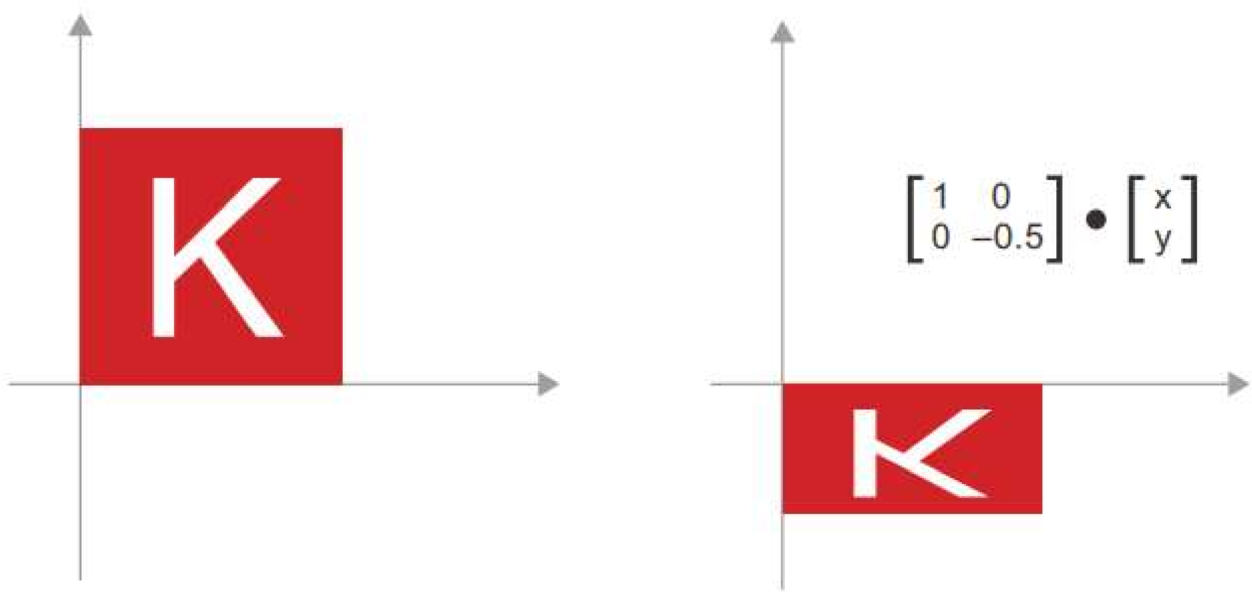
Tensor operations

2. Geometric interpolation of tensor operations (cont):
   - A few examples (cont):
     - ✓ *Rotation*:



Tensor operations

2. Geometric interpolation of tensor operations (cont):
   - A few examples (cont):
     - *Scaling*: A vertical and horizontal scaling of the image can be achieved via a dot product with a 2 $\times$ 2 matrix S = [[horizontal_factor, 0], [0, vertical_factor]]

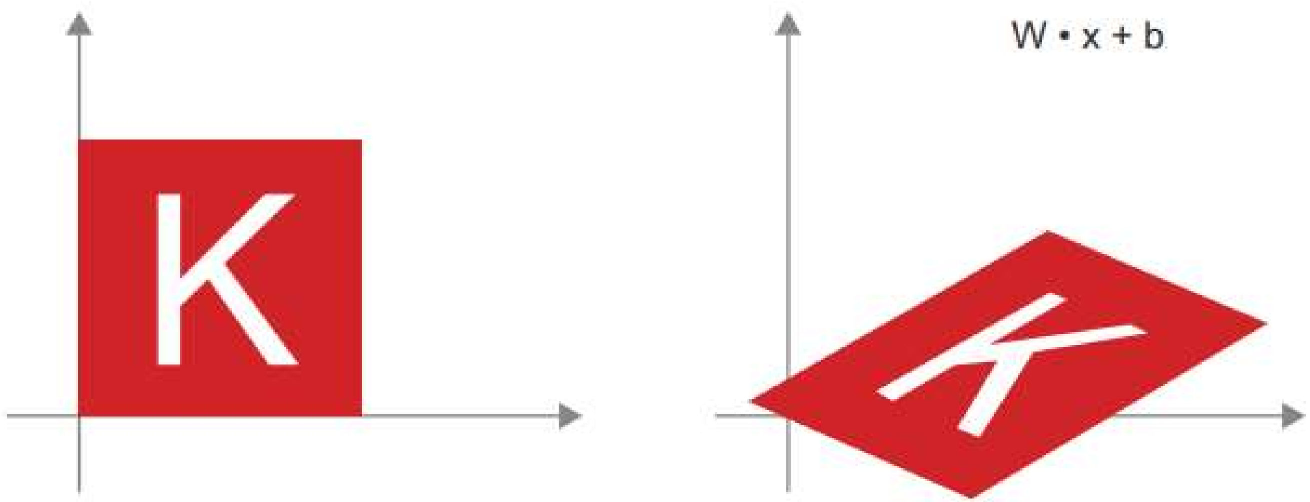


Tensor operations

2. Geometric interpolation of tensor operations (cont):
   - A few examples (cont):
     - ✓ *Affine transform:* An affine transform is the combination of a linear transform (achieved via a dot product with some matrix) and a translation (achieved via a vector addition).
     - ✓ That's the y = W • x + b computation implemented by the Dense layer!
     - ✓ A Dense layer without an activation function is an affine layer.
     - ✓ If we apply many affine transforms repeatedly, we still end up with an affine transform

Tensor operations

2. Geometric interpolation of tensor operations (cont):
   - A few examples (cont):
     - ✓ *Affine transform:*

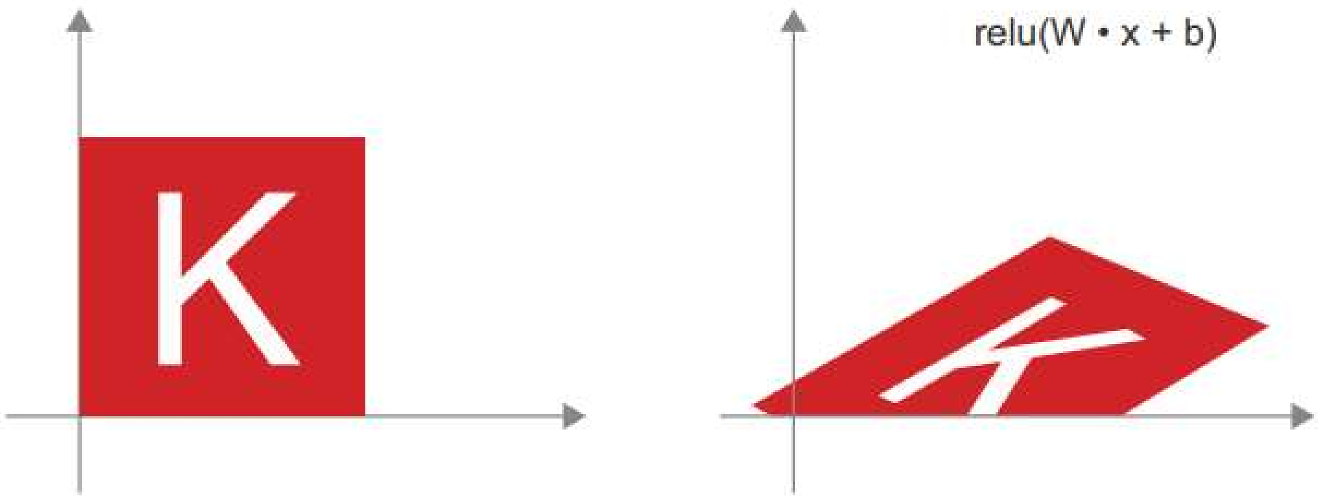


Tensor operations

2. Geometric interpolation of tensor operations (cont):
   - A few examples (cont):
     - *Dense layer with relu activation:* A multilayer neural network made entirely of Dense layers without activations would be equivalent to a single Dense layer
     - An activation function such as ReLU can be made to implement very complex, non-linear geometric transformations, resulting in very rich hypothesis spaces for the deep neural networks.

Tensor operations

2. Geometric interpolation of tensor operations (cont):
   - A few examples (cont):
     - ✓ *Dense layer with relu activation:*



Tensor operations

3. A geometric interpolation of deep learning:
   - Deep learning takes the approach of incrementally decomposing a complicated geometric transformation into a long chain of elementary ones
   - This is pretty much the strategy a human would follow to uncrumple a paper ball.
   - Each layer in a deep network applies a transformation that disentangles the data a little, and a deep stack of layers makes tractable an extremely complicated disentanglement process.

Tensor operations

3. A geometric interpolation of deep learning:
   - Uncrumpling a complicated manifold of data