PythonとKerasによるディープラーニング

2.3 ニューラルネットワークの歯車：テンソル演算

17T4014Y 伊藤 陽樹

テンソル演算

- ディープニューラルネットワークによって学習された変換　→　テンソル演算に分解
- 例えば、テンソルの加算や乗算などが可能

- 2.1のリスト2-2では、全結合(Dense)層を積み上げていく方法でNNを構築

```
network.add(layers.Dense(512, activation ='relu',
                         input_shape=(28 * 28,)))
```

- Kerasの層のインスタンス化

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

テンソル演算

- 入力: 2次元テンソル, 出力: (新しい)2次元テンソル
- W: 2次元テンソル, b: ベクトル

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

- 入力テンソルとWというテンソルの内積(dot)にベクトルbを加算(+)し、ReLU演算の3つのテンソル演算で構成.
- relu(x) = max(x, 0)

要素ごとの演算

- ReLUと加算は、要素ごとの演算.
  - ➢演算の対象となるテンソルの要素ごとに適用.
- こうした演算はベクトル化実装に適している.
- ReLUの実装:

```
def naive_relu(x):
    assert len(x.shape) == 2                # xはNumPyの2次元テンソル
    x = x.copy()                            # 入力テンソルの上書き回避
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] = max(x[i, j], 0)
    return x
```

要素ごとの演算

- 同じ原理で、要素ごとの加算、乗算、減算が実装できる.
- 実際にNumPy配列を扱うときは、こうした演算は最適化された関数としてサポートされている.
- NumPyでの要素ごとの演算(高速)

```
import numpy as np

z = x + y                    # 要素ごとの加算
z = np.maximum(z, 0.)        # 要素ごとのReLU
```

ブロードキャスト

- 先の単純な加算の実装は、同じ形状の2次元テンソルの加算だけをサポート.
- 2つのテンソルの形状が異なる場合の加算
  - 小さい方が大きい方の形状に合わせてブロードキャスト(可能であること、曖昧さなしが前提)
- 手順:
  1. ブロードキャスト軸を小さい方のテンソルに追加し、大きい方と次元の数(ndim)を一致させる.
  2. 小さい方を新しい軸上で繰り返すことで、大きい方と完全に同じ形状にする.

テンソルの内積(数学的)

- 内積演算(テンソル積)は入力テンソルの要素を組み合わせる.(要素ごとの積とは別物)
- NumPyとKerasで内積:

```
import numpy as np
z = np.dot(x, y)
```

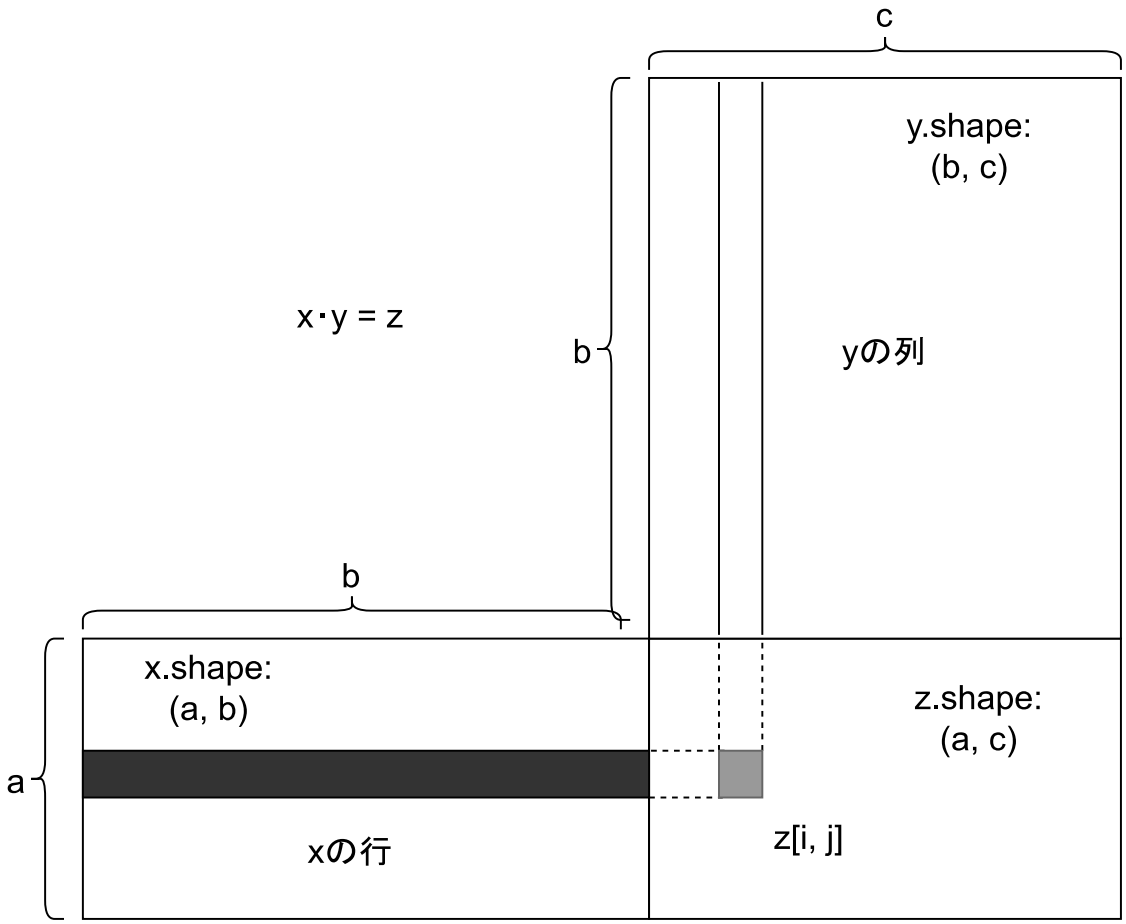
- 2つのベクトルの内積は、スカラー.
  (内積を計算できるのは要素数が等しいとき)
- 行列xとベクトルyの内積は、係数がxの行とyの内積であるベクトル.

テンソルの内積(数学的)

- 2つのテンソルのうち一方の次元の数(ndim)が1よりも大きくなった時点で、内積は対照的でなくなる.
  - つまり、dot(x, y) ≠ dot(y, x).
- 内積により、テンソルは任意の数の軸に対して一般化される.
  - 例: 2つの行列の内積

    dot(x, y)を求められるのは、x.shape[1] == y.shape[0]の場合.
    結果、形状が(x.shape[0], y.shape[1])の行列が得られる.
    (係数はxの行とyの列のベクトル積)

行列の内積



テンソルの変形

- テンソルの変形は、目的の形状と一致するようにテンソルの行と列の配置を変更を意味する.
- 単純な例:

```
>>> x = np.array([[0., 1.],
                  [2., 3.],
                  [4., 5.]])
>>> print(x.shape)
(3, 2)
>>> x = x.reshape((6, 1))
```

テンソルの変形

```
>>> x
array([[ 0.],
       [ 1.],
       [ 2.],
       [ 3.],
       [ 4.],
       [ 5.]])
>>> x = x.reshape((2, 3))
>>> x
array([[ 0., 1., 2.],
       [ 3., 4., 5.]])
```

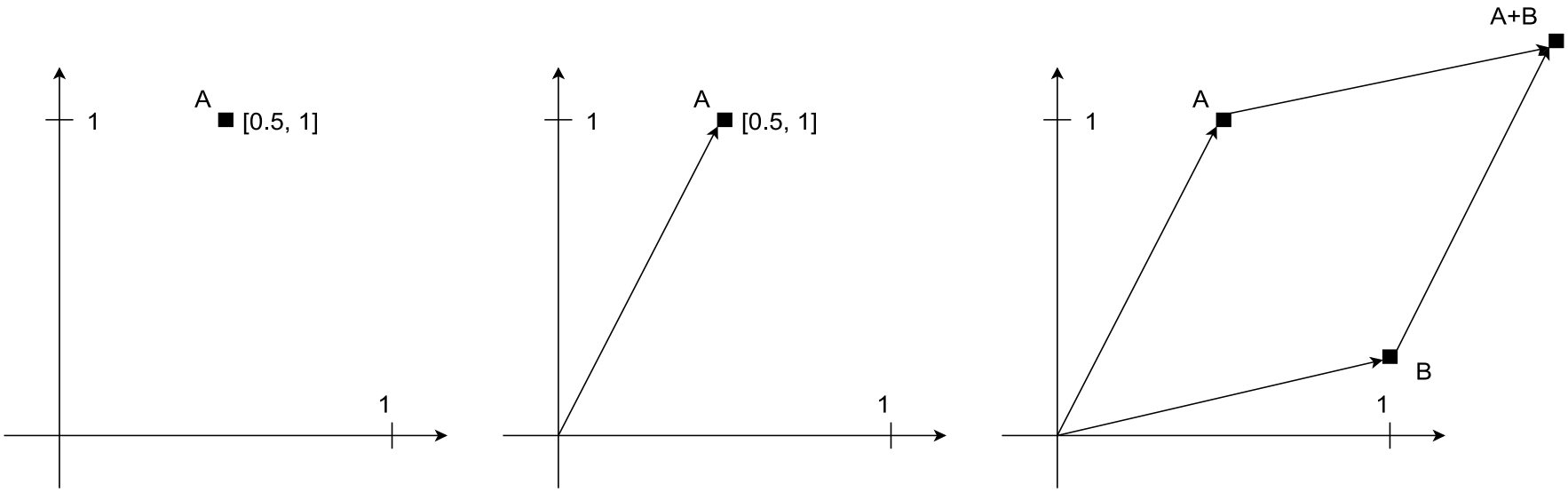
テンソルの変形

- 転置: よく見かける特殊な変形の1つ. 行列の転置はその行と列を入れ替える. (x[i, :]はx[:, i]に)

```
>>> x = np.zeros((300, 20))
>>> x = np.transpose(x)
>>> print(x.shape)
(20, 300)
```

テンソル演算の幾何学的解釈

- テンソル演算によって操作されるテンソルの内容は、何らかの幾何学空間上にある点の座標として解釈できる.
  - すべてのテンソル演算を幾何学的に解釈可能.
- 加算の場合:



テンソル演算の幾何学的解釈

- 一般に、アフィン変換、回転、拡大縮小といった基本的な幾何学演算は、テンソル演算として表現できる.
- 例: 角度θ(theta)による2次元ベクトルの回転

  2×2行列R = [u, v]　　(u, v: 平面ベクトル)

  u = [cos(theta), sin(theta)]

  v = [-sin(theta), cos(theta)]

DLの幾何学的解釈

- NNについては、単純なステップをいくつも連ねたものとして実装された、高次元空間の非常に複雑な幾何学変換として解釈できる.

- 機械学習とは、複雑に折り畳まれたデータ多様体から整然とした表現を見つけ出すこと.
- DLのアプローチは、複雑な幾何学変換を基本的な変換の連鎖として少しずつ分解していくというもの.