# Python for Data Analysis

**Summer 2020**
**Princeton Neuroscience Institute**
**Instructors: Ben Cowley and Greg Henselman-Petrusek**

# Contact

Benjamin Cowley: bcowley@princeton.edu

Greg Henselman-Petrusek: gh10@princton.edu

# Preliminaries

- Expectations
- Homeworks
- Way-finding
  - Plug: “Design your life” by Bill Burnet + Dave Evans
- Growth mindset
- Questions!

# Preview

- Neural networks
- Working with data (randomness, tests used in experiments)
- Visualization
- Writing programs to make predictions (bona fide super power)
- dimension reduction
  - pca
  - tsne

# Set up

1. Google colab
2. Create colab notebook
3. Done!

# Class 1: Machine learning

# Machine learning

**What’s a neural network?**
**What do you do with a neural network?**

# Machine learning

feed-forward

What's a neural network?

What do you do with a neural network?

feed-forward

# Machine learning

**What's a neural network?** A sequence of math functions
**What do you do with a neural network?** Stuff

# Machine learning

**What's a neural network?**

- a sequence of math functions

**What do you do with a neural network?**

- stuff you can't do without one
- stuff you can't describe with numbers
  - examples
    - write a program that names all the animals in a picture
    - write a program that recognizes sadness in facial expressions
    - write a program that can beat people at chess

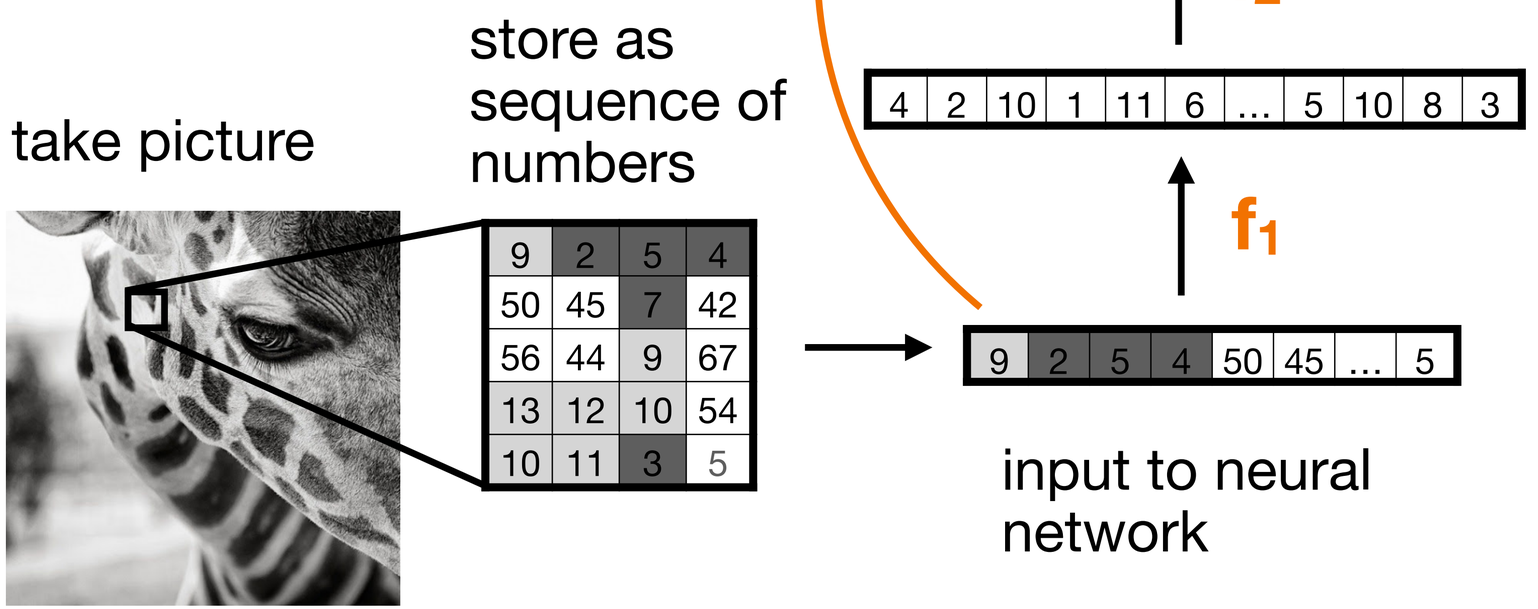
# Machine learning

**What's a neural network?**

- a math function f
- made of a sequence of math functions $f_1$, $f_2$, $f_3$

**What do you do with a neural network?**

- stuff you can't do without one
- stuff you can't describe with numbers
  - examples
    - write a program that names all the animals in a picture



# Machine learning

**What's a neural network?**

- a math function f
- made of a sequence of math functions $f_1$, $f_2$, $f_3$

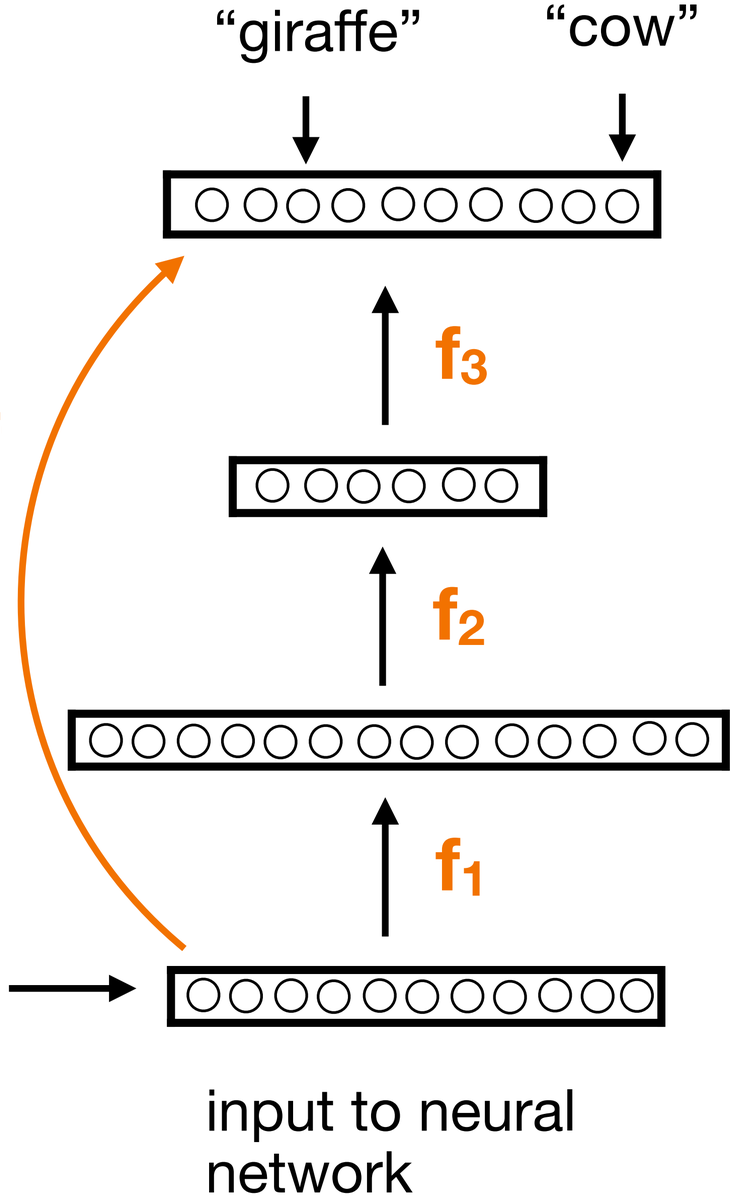
**What do you do with a neural network?**

- stuff you can't do without one
- stuff you can't describe with numbers
  - examples
    - write a program that names all the animals in a picture

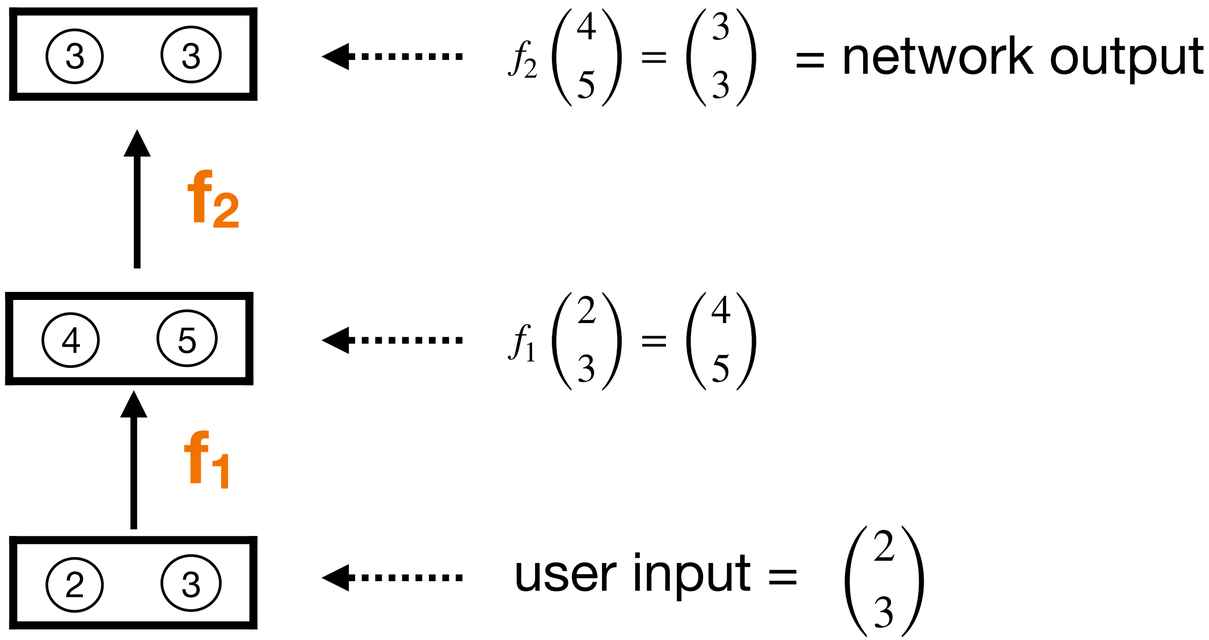
take picture

store as sequence of numbers

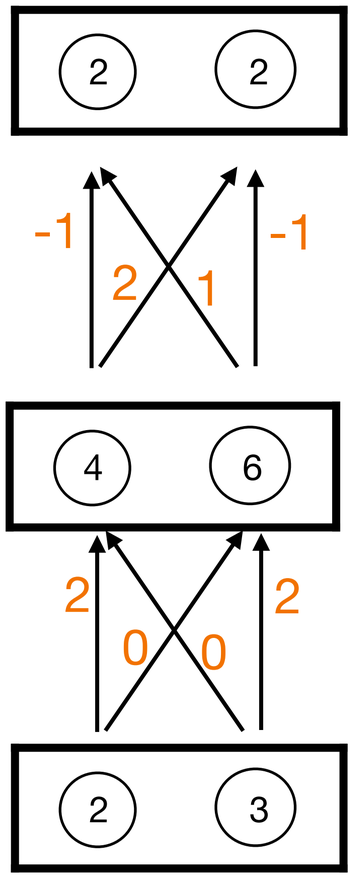
| | | | |
|---|---|---|---|
| 9 | 2 | 5 | 4 |
| 50 | 45 | 7 | 42 |
| 56 | 44 | 9 | 67 |
| 13 | 12 | 10 | 54 |
| 10 | 11 | 3 | 5 |



# Example network diagram



# Example network diagram




$f_2\begin{pmatrix}4\\6\end{pmatrix} = \begin{pmatrix}2\\2\end{pmatrix}$ = network output

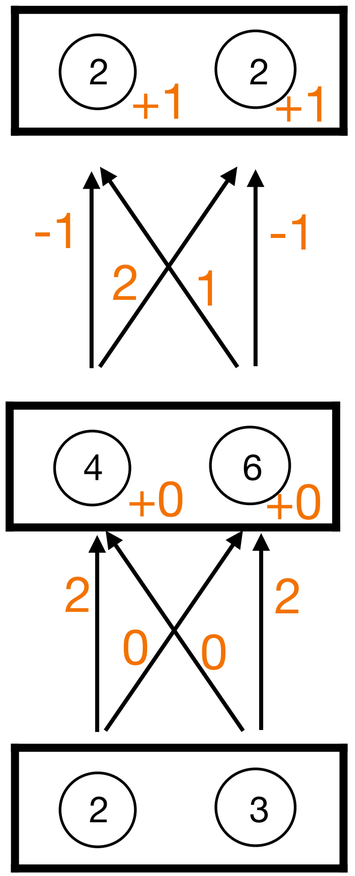
$f_2\begin{pmatrix}y_0\\y_1\end{pmatrix} = \begin{pmatrix}-1y_0 + y_1\\2y_0 - 1y_1\end{pmatrix}$

$f_1\begin{pmatrix}2\\3\end{pmatrix} = \begin{pmatrix}4\\6\end{pmatrix}$

$f_1\begin{pmatrix}x_0\\x_1\end{pmatrix} = \begin{pmatrix}2x_0 + 0x_1\\0x_0 + 2x_1\end{pmatrix}$

user input = $\begin{pmatrix}2\\3\end{pmatrix}$

# Example network diagram




$f_2\begin{pmatrix}4\\6\end{pmatrix} = \begin{pmatrix}3\\3\end{pmatrix}$ = network output

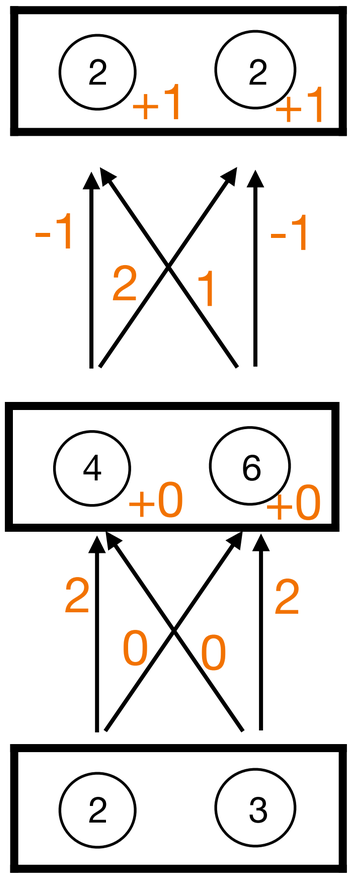
$f_2\begin{pmatrix}y_0\\y_1\end{pmatrix} = \begin{pmatrix}-1y_0 + y_1 + 1\\2y_0 - 1y_1 + 1\end{pmatrix}$

$f_1\begin{pmatrix}2\\3\end{pmatrix} = \begin{pmatrix}4\\6\end{pmatrix}$

$f_1\begin{pmatrix}x_0\\x_1\end{pmatrix} = \begin{pmatrix}2x_0 + 0x_1 + 0\\0x_0 + 2x_1 + 0\end{pmatrix}$

user input = $\begin{pmatrix}2\\3\end{pmatrix}$

# Example network diagram




$$f_2\begin{pmatrix}4\\6\end{pmatrix} = \begin{pmatrix}3\\3\end{pmatrix}$$ = network output

$$f_2\begin{pmatrix}y_0\\y_1\end{pmatrix} = \begin{pmatrix}-1y_0 + 1y_1 + 1\\2y_0 - 1y_1 + 1\end{pmatrix}$$

$$f_1\begin{pmatrix}2\\3\end{pmatrix} = \begin{pmatrix}4\\6\end{pmatrix}$$

$$f_1\begin{pmatrix}x_0\\x_1\end{pmatrix} = \begin{pmatrix}2x_0 + 0x_1 + 0\\0x_0 + 2x_1 + 0\end{pmatrix}$$

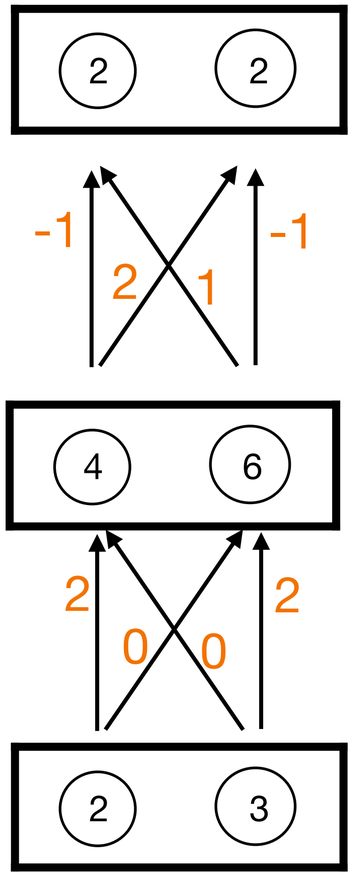
user input = $\begin{pmatrix}2\\3\end{pmatrix}$

## Exercise: write python programs for $f_1$, $f_2$, and f

### solution

```
def  f(x0, x1):
  y0, y1 = f1(x0, x1)
  z0, z1 = f2(y0, y1)
  return z0, z1

def f2(y0, y1):
  z0 = -1*y0 + 1*y1 + 1
  z1 =  2*y0 - 1*y1 + 1
  return z0, z1

def  f1(x0, x1):
  y0 =  2*x0 + 0*x1 + 0
  y1 =  0*x0 + 2*x1 + 0
  return y0, y1
```

# Example network diagram




$$f_2\begin{pmatrix}4\\6\end{pmatrix} = \begin{pmatrix}2\\2\end{pmatrix}$$ = network output

$$f_2\begin{pmatrix}y_0\\y_1\end{pmatrix} = \begin{pmatrix}-1y_0 + y_1\\2y_0 - 1y_1\end{pmatrix}$$

$$f_1\begin{pmatrix}2\\3\end{pmatrix} = \begin{pmatrix}4\\6\end{pmatrix}$$

$$f_1\begin{pmatrix}x_0\\x_1\end{pmatrix} = \begin{pmatrix}2x_0 + 0x_1\\0x_0 + 2x_1\end{pmatrix}$$

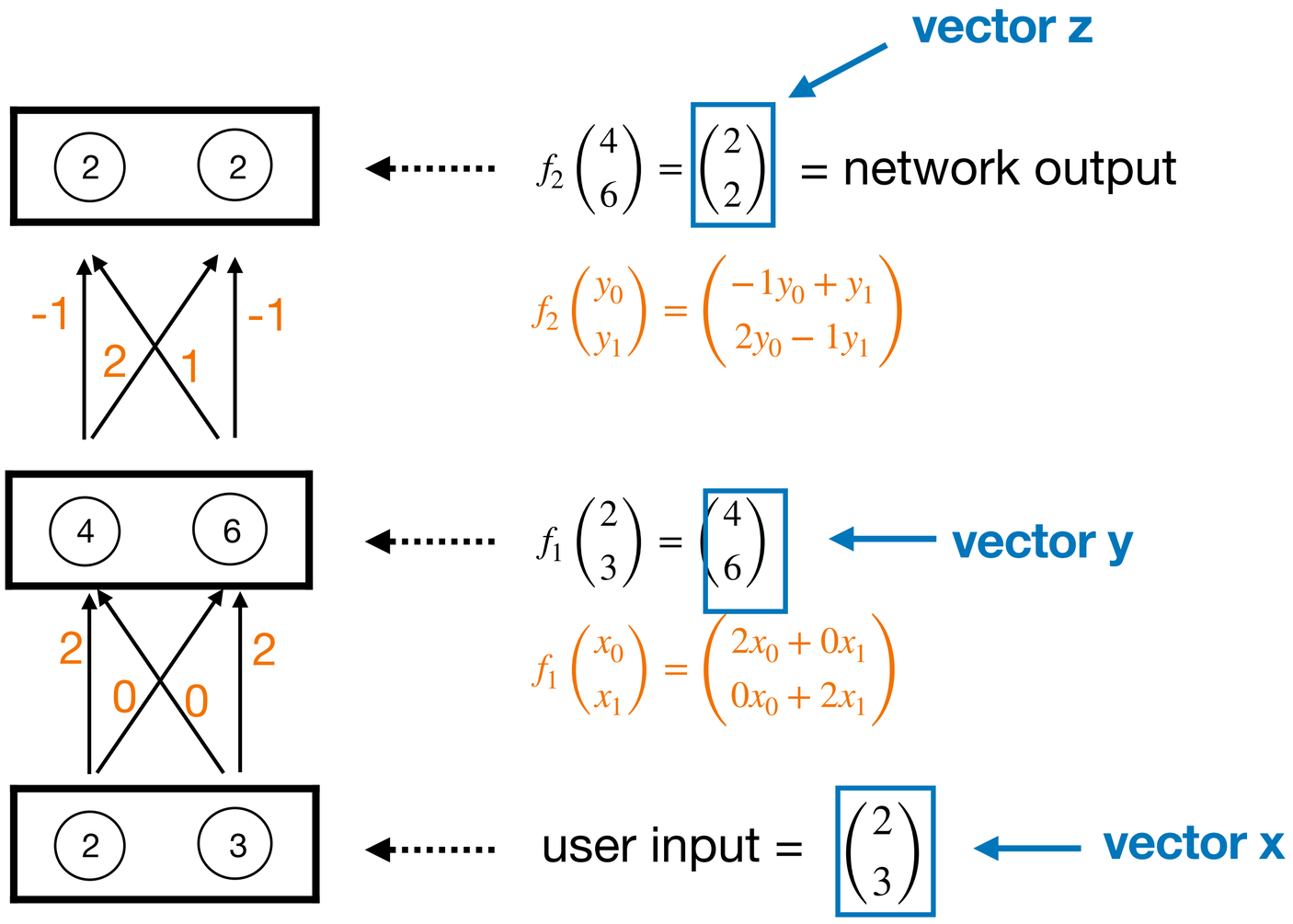
user input = $\begin{pmatrix}2\\3\end{pmatrix}$

## Exercise: write python programs for $f_1$, $f_2$, and f

### solution

```
def  f(x0, x1):
  y0, y1 = f1(x0, x1)
  z0, z1 = f2(y0, y1)
  return z0, z1

def f2(y0, y1):
  y0 = -1*y0 + 1*y1
  y1 =  2*y0 - 1*y1
  return y0, y1

def  f1(x0, x1):
  y0 =  2*x0 + 0*x1
  y1 =  0*x0 + 2*x1
  return y0, y1
```

# Example network diagram

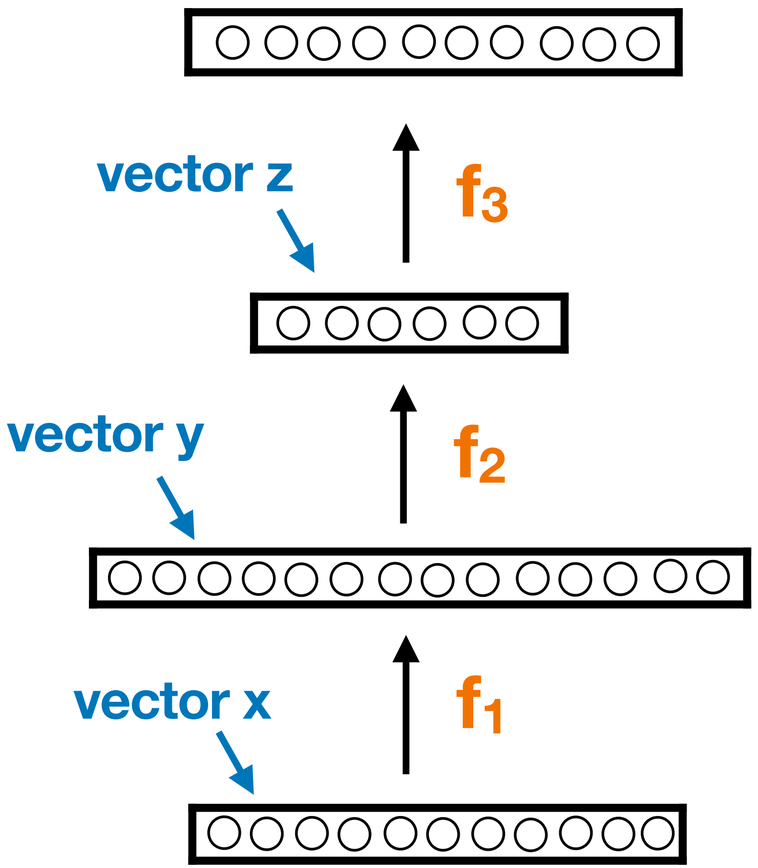



## Exercise: write python programs for $f_1$, $f_2$, and f

### solution

```
def  f(x0, x1):
  y0, y1 = f1(x0, x1)
  z0, z1 = f2(y0, y1)
  return z0, z1

def f2(y0, y1):
  y0 = -1*y0 + 1*y1
  y1 =  2*y0 - 1*y1
  return y0, y1

def  f1(x0, x1):
  y0 =  2*x0 + 0*x1
  y1 =  0*x0 + 2*x1
  return y0, y1
```

# Example network diagram

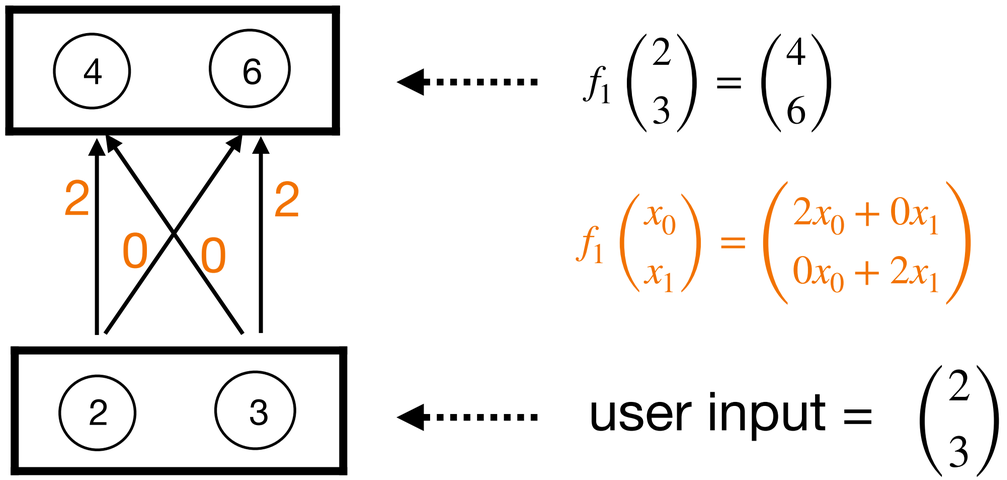



**this becomes impossible to code by hand when vectors are long**

**Exercise: write python programs for $f_1$, $f_2$, and f**

**solution**

```
def  f(x0, x1):
  y0, y1 = f1(x0, x1)
  z0, z1 = f2(y0, y1)
  return z0, z1

def f2(y0, y1):
  y0 = -1*y0 + 1*y1
  y1 =  2*y0 - 1*y1
  return y0, y1

def  f1(x0, x1):
  y0 =  2*x0 + 0*x1
  y1 =  0*x0 + 2*x1
  return y0, y1
```




$$f_1\begin{pmatrix}2\\3\end{pmatrix} = \begin{pmatrix}4\\6\end{pmatrix}$$

$$f_1\begin{pmatrix}x_0\\x_1\end{pmatrix} = \begin{pmatrix}2x_0 + 0x_1\\0x_0 + 2x_1\end{pmatrix}$$

user input = $\begin{pmatrix}2\\3\end{pmatrix}$

# We need a simple python command to …

| | |
|---|---|
| 2 | 0 |
| 0 | 2 |

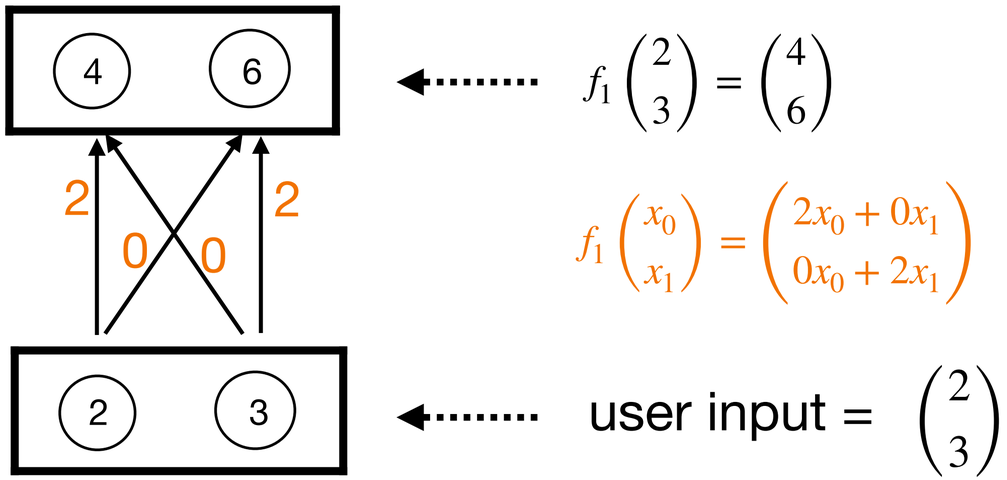
take this table of numbers …

| |
|---|
| $x_0$ |
| $x_1$ |

and a list of numbers ….

| |
|---|
| $2*x_0 + 0*x_1$ |
| $0*x_0 + 2*x_1$ |

and return this list of numbers




$$f_1\begin{pmatrix}2\\3\end{pmatrix} = \begin{pmatrix}4\\6\end{pmatrix}$$

$$f_1\begin{pmatrix}x_0\\x_1\end{pmatrix} = \begin{pmatrix}2x_0 + 0x_1\\0x_0 + 2x_1\end{pmatrix}$$

user input = $\begin{pmatrix}2\\3\end{pmatrix}$

## We need a simple python command to …

| | |
|---|---|
| 2 | 0 |
| 0 | 2 |

take this table of numbers …

| |
|---|
| $x_0$ |
| $x_1$ |

and a list of numbers ….

| |
|---|
| $2*x_0 + 0*x_1$ |
| $0*x_0 + 2*x_1$ |

and return this list of numbers

## THERE'S A COMMAND FOR THAT!

```
import numpy as np

def f1(x0, x1):

   A = np.array( [[2, 0],
                  [0, 2]] )

   x = np.array( [[x0],
                  [x1]] )

   return np.matmul(A,x)
```

**We need a simple python command to …**

| | |
|---|---|
| 2 | 0 |
| 0 | 2 |

**take this table of numbers …**

| |
|---|
| $x_0$ |
| $x_1$ |

**and a list of numbers ….**

| |
|---|
| $2*x_0 + 0*x_1$ |
| $0*x_0 + 2*x_1$ |

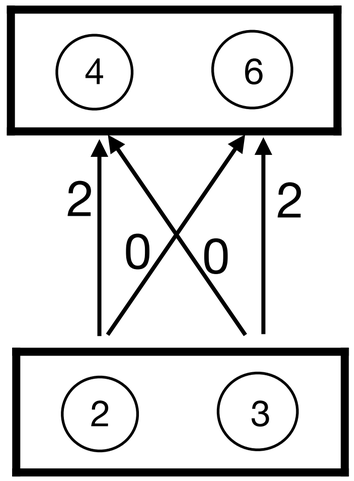
**and return this list of numbers**

**THERE'S A COMMAND FOR THAT!**

```
def f1(x0, x1):

  A = np.array( [[2, 0],
                 [0, 2]] )

  x = np.array( [[x0],
                 [x1]] )

  return np.matmul(A,x)
```

**MATRIX MULTIPLICATION**

**makes it easy to write functions like this one …**




$$f_1\begin{pmatrix}2\\3\end{pmatrix} = \begin{pmatrix}4\\6\end{pmatrix}$$

$$f_1\begin{pmatrix}x_0\\x_1\end{pmatrix} = \begin{pmatrix}2x_0 + 0x_1\\0x_0 + 2x_1\end{pmatrix}$$

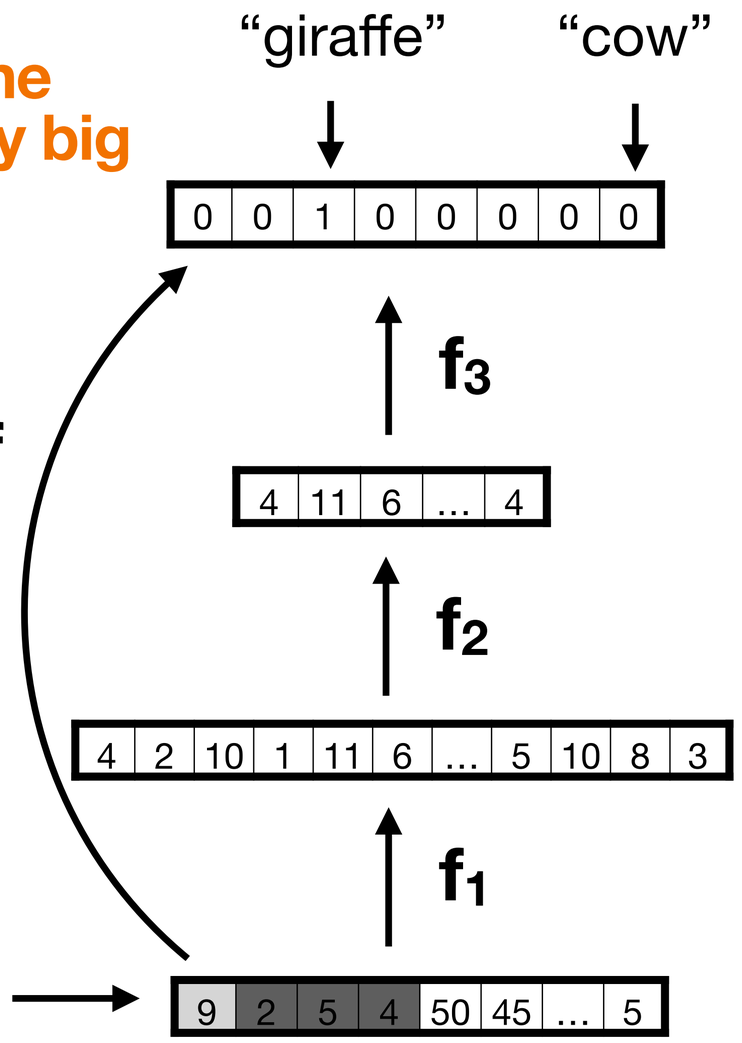
user input = $\begin{pmatrix}2\\3\end{pmatrix}$

**… even when the network is really big**

take picture

store as sequence of numbers

| | | | |
|---|---|---|---|
| 9 | 2 | 5 | 4 |
| 50 | 45 | 7 | 42 |
| 56 | 44 | 9 | 67 |
| 13 | 12 | 10 | 54 |
| 10 | 11 | 3 | 5 |

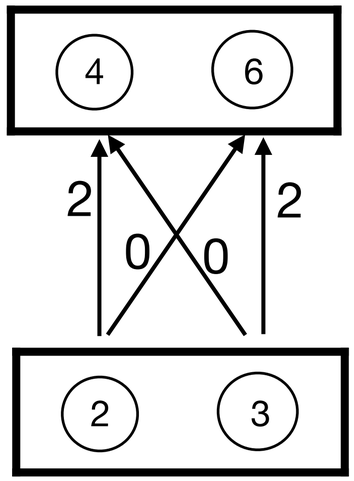



input to neural network

```
def f1(x0, x1):

  A = np.array( [[2, 0],
                 [0, 2]] )

  x = np.array( [[x0],
                 [x1]] )

  return np.matmul(A,x)
```

**MATRIX MULTIPLICATION**

**does a lot of other great stuff, too**




$$f_1\begin{pmatrix}2\\3\end{pmatrix} = \begin{pmatrix}4\\6\end{pmatrix}$$

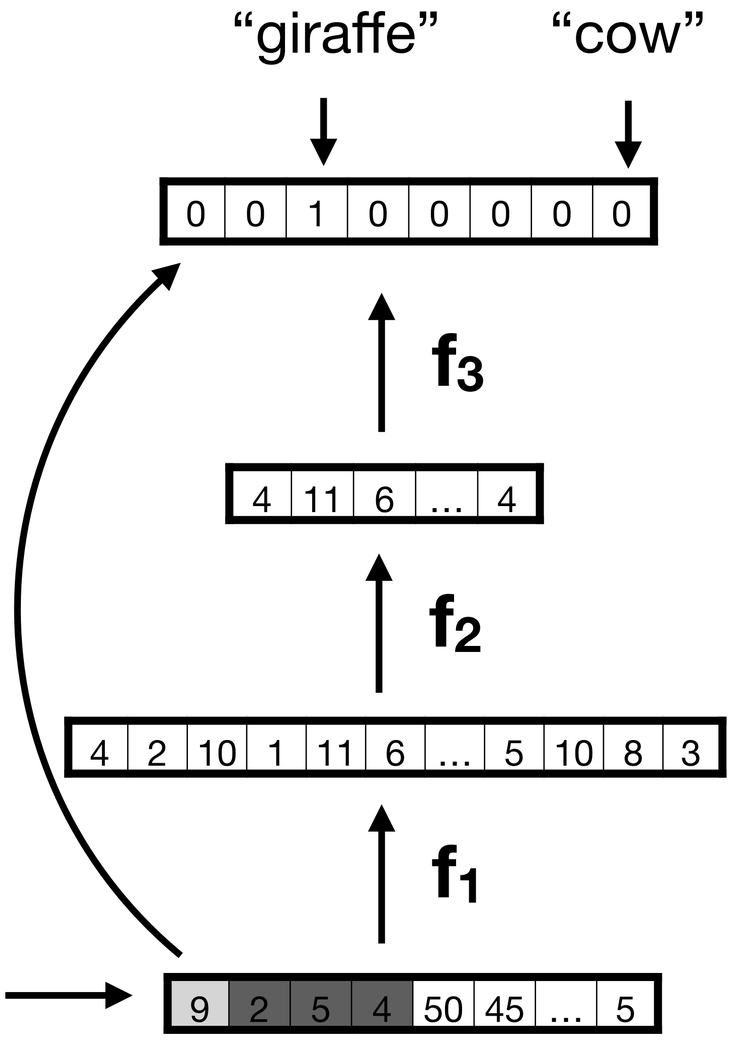
$$f_1\begin{pmatrix}x_0\\x_1\end{pmatrix} = \begin{pmatrix}2x_0 + 0x_1\\0x_0 + 2x_1\end{pmatrix}$$

user input = $\begin{pmatrix}2\\3\end{pmatrix}$

take picture

store as sequence of numbers

| | | | |
|---|---|---|---|
| 9 | 2 | 5 | 4 |
| 50 | 45 | 7 | 42 |
| 56 | 44 | 9 | 67 |
| 13 | 12 | 10 | 54 |
| 10 | 11 | 3 | 5 |



# Matrix multiplication

| | |
|---|---|
| 2 | 0 |
| 0 | 2 |

take this table of numbers …

| |
|---|
| $x_0$ |
| $x_1$ |

and a list of numbers ….

| |
|---|
| $2*x_0 + 0*x_1$ |
| $0*x_0 + 2*x_1$ |

and return this list of numbers

```
def f1(x0, x1):

  A = np.array( [[2, 0],
                 [0, 2]] )

  x = np.array( [[x0],
                 [x1]] )

  return np.matmul(A,x)
```

# Matrix multiplication

| | |
|---|---|
| 2 | 0 |
| 0 | 2 |

take this table of numbers …

| |
|---|
| $x_0$ |
| $x_1$ |

and a list of numbers ….

| |
|---|
| $2*x_0 + 0*x_1$ |
| $0*x_0 + 2*x_1$ |

and return this list of numbers

**Python uses “numpy” for this:**

```
import numpy as np

def f1(x0, x1):

  A = np.array( [[2, 0],
                 [0, 2]] )

  x = np.array( [[x0],
                 [x1]] )

  return np.matmul(A,x)
```

# Matrix multiplication (vanilla version)

| | |
|---|---|
| 2 | 0 |
| 0 | 2 |

take this table of numbers …

| |
|---|
| $x_0$ |
| $x_1$ |

and a list of numbers ….

| |
|---|
| $2*x_0 + 0*x_1$ |
| $0*x_0 + 2*x_1$ |

and return this list of numbers

# Matrix multiplication (superpower version)

| | |
|---|---|
| 2 | 0 |
| 0 | 2 |

**take this table of numbers …**

| | |
|---|---|
| $x_0$ | $y_0$ |
| $x_1$ | $y_1$ |

**and this table of numbers ….**

| | |
|---|---|
| $2*x_0 + 0*x_1$ | $2*y_0 + 0*y_1$ |
| $0*x_0 + 2*x_1$ | $0*y_0 + 2*y_1$ |

**and return this list of numbers**

# Matrix multiplication (superpower version)

| | |
|---|---|
| 2 | 0 |
| 0 | 2 |

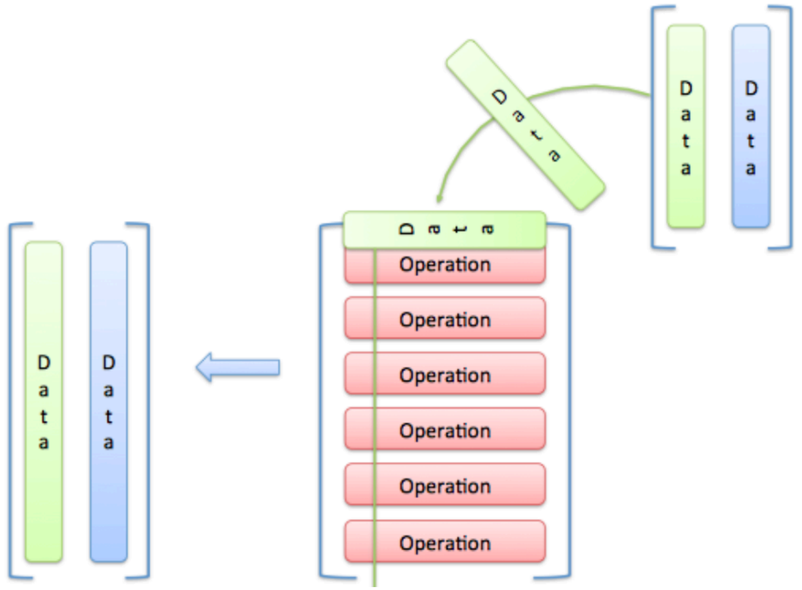
take this table of numbers …

| | |
|---|---|
| $x_0$ | $y_0$ |
| $x_1$ | $y_1$ |

and this table of numbers ….

| | |
|---|---|
| $2*x_0 + 0*x_1$ | $2*y_0 + 0*y_1$ |
| $0*x_0 + 2*x_1$ | $0*y_0 + 2*y_1$ |

and return this table of numbers




Essentially, do matrix multiplication on each column in the righthand table

# Matrix multiplication

**THE ONLY RULE** **# columns (lefthand table) = # rows (righthand table)**

2 COLUMNS

| | |
|---|---|
| 2 | 0 |
| 0 | 2 |

**take this table of numbers …**

2 ROWS

| | |
|---|---|
| $x_0$ | $y_0$ |
| $x_1$ | $y_1$ |

**and this table of numbers ….**

| | |
|---|---|
| $2*x_0 + 0*x_1$ | $2*y_0 + 0*y_1$ |
| $0*x_0 + 2*x_1$ | $0*y_0 + 2*y_1$ |

**and return this table of numbers**

# Matrix multiplication

**THE ONLY RULE** **# columns (lefthand table) = # rows (righthand table)**

2 COLUMNS

| | |
|---|---|
| 2 | 0 |
| 0 | 2 |

**take this table of numbers …**

2 ROWS

| | | |
|---|---|---|
| $x_0$ | $y_0$ | $z_0$ |
| $x_1$ | $y_1$ | $z_1$ |

**and this table of numbers ….**

| | | |
|---|---|---|
| $2*x_0 + 0*x_1$ | $2*y_0 + 0*y_1$ | $2*z_0 + 0*z_1$ |
| $0*x_0 + 2*x_1$ | $0*y_0 + 2*y_1$ | $0*z_0 + 2*z_1$ |

**and return this table of numbers**

# Matrix multiplication

**a rectangle of numbers**

↓

**array**

| | |
|---|---|
| 2 | 0 |
| 0 | 2 |

```
>> A = np.array( [[2, 0],
                  [0, 2]] )
>> A.shape
(2,2)

>> A.size
4
```

**array with only 1 column**

↓

**column vector**

| |
|---|
| 1 |
| 2 |

```
>> A = np.array( [[1],
                  [2]] )

>> A.shape
(2,1)

>> A.size
2
```

**array with only 1 row**

↓

**row vector**

| | | |
|---|---|---|
| 1 | 2 | 3 |

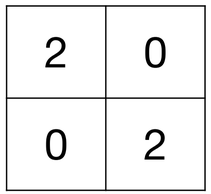
```
>> A = np.array( [[1, 2]] )
>> A.shape
(1,2)

>> A.size
2
```

# Matrix multiplication

a rectangle of numbers

↓

array

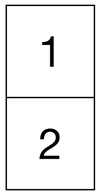



`A`

array with only 1 column

↓

column vector

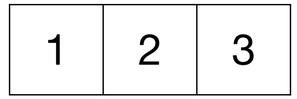



`B`

array with only 1 row

↓

row vector




`C`

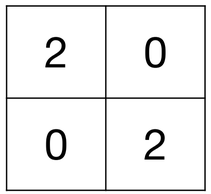
**Exercise: which combinations of `A, B, C` can you multiply?**

# Matrix multiplication

a rectangle of numbers

↓

array

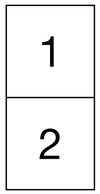



`A`

array with only 1 column

↓

column vector

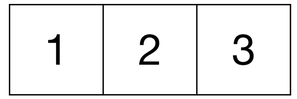



`B`

array with only 1 row

↓

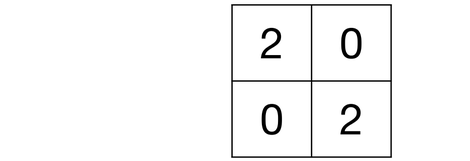
row vector

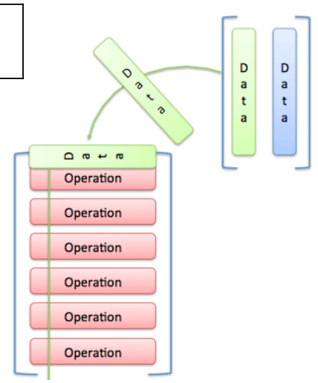



`C`

**Exercise: which combinations of `A, B, C` can you multiply? (answer on the next slide)**

# Matrix multiplication




a rectangle of numbers

↓

array

| | |
|---|---|
| 2 | 0 |
| 0 | 2 |

A

array with only 1 column

↓

column vector

| |
|---|
| 1 |
| 2 |

B

array with only 1 row

↓

row vector

| | |
|---|---|
| 1 | 2 |

C

**Exercise: which combinations of `A, B, C` can you multiply? (answer on the next slide)**

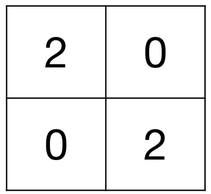
**Answer:** `AB, CA, CB, CAB`

# Matrix multiplication

a rectangle of numbers

↓

array

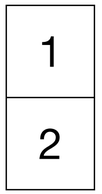



A

array with only 1 column

↓

column vector

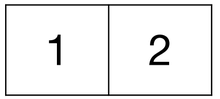



B

array with only 1 row

↓

row vector




C

**Exercise: use numpy to calculate all these matrix products:** `AB, CA, CB, CAB`

**Exercise: check your answers!**

# Matrix multiplication

**a rectangle of numbers**

↓

**array**

| | | |
|---|---|---|
| a | b | c |
| d | e | f |

`A`

**the rectangle you get by swapping rows and columns**

↓

**transpose of array**

| | |
|---|---|
| a | d |
| b | e |
| c | f |

both commands give the same result

```
np.transpose(A)
A.T
```

**Exercise: use numpy to transpose the arrays in the previous slide**
**Exercise: use the `array_equal` function to check the following:**

```
np.matmul(A,B).T = np.matmul(B.T, A.T)
```

# Matrix multiplication

## Nuts ’n bolts

You create an array in numpy using a list of lists (a 2-list). You can also create an array with a list (ie a 1-list) or a list of list of lists (a 3-list). Indeed, you can make an array from pretty much any n-list.

```
>> A = np.array( [[2, 0],
                  [0, 2]] )
array([[2, 0],
       [0, 2]])
>> A.ndim
2
>> A.size
4
>> A.shape
(2,2)
```

```
>> A = np.array( [2, 0] )
array([2, 0])

>> A.ndim
1
>> A.size
2
>> A.shape
(2)
```

```
>> A = np.array( [[[2]]] )
array([[[2]]])

>> A.ndim
3
>> A.size
1
>> A.shape
(1,1,1)
```

# Matrix multiplication

## Nuts ’n bolts

You create an array in numpy using a list of lists (a 2-list). You can also create an array with a list (ie a 1-list) or a list of list of lists (a 3-list). Indeed, you can make an array from pretty much any n-list.

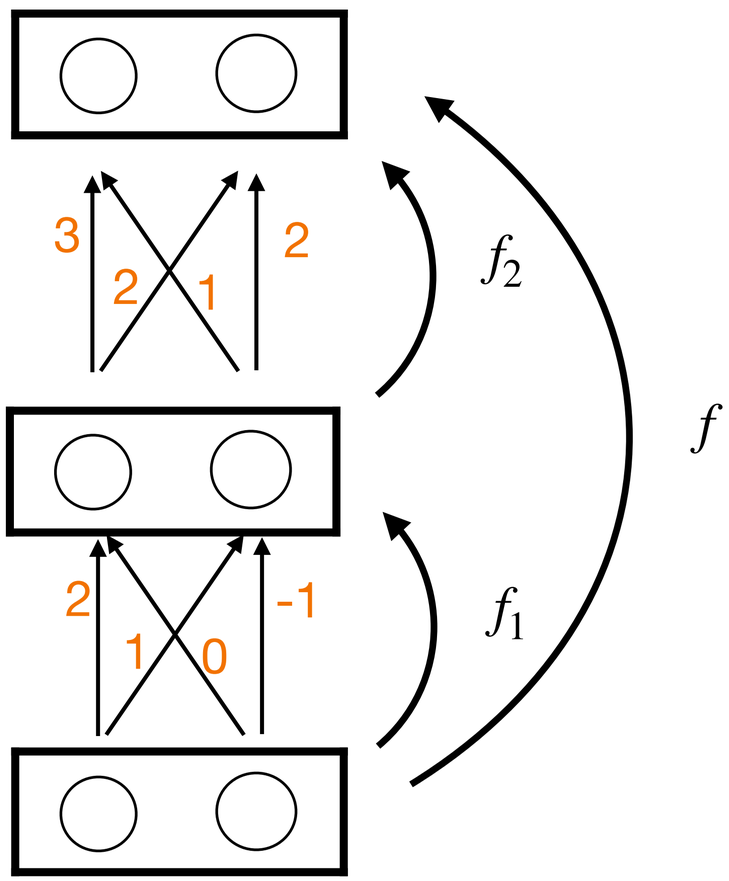
```
>> A = np.array( [[2, 0],
                  [0, 2]] )
array([[2, 0],
       [0, 2]])
>> A.ndim
2
>> A.size
4
>> A.shape
(2,2)
```

```
>> A = np.array( [2, 0] )
array([2, 0])

>> A.ndim
1
>> A.size
2
>> A.shape
(2)
```

```
>> A = np.array( [[[2]]] )
array([[[2]]])

>> A.ndim
3
>> A.size
1
>> A.shape
(1,1,1)
```

**Exercise: create arrays of shape (3) and (1,1,2)**
**Exercise: multiply the entries in these by 2 using A = A*0**
**Exercise: fill these arrays with random numbers using np.random.rand**

# Exercises




Write python functions for each of the functions $f_1$, $f_2$, and f given by the diagram shown left.

Write one version of each function that
- does not use matrix multiplication
- takes pairs of numbers as inputs (ie x0, x1)

Write another version that
- does use matrix multiplication
- takes a single numpy array (column vector) as input

Write another version that
- does use matrix multiplication
- takes a single numpy array (row vector) as input

Modify your matrix functions so that they
- check that the input array has the right shape + number of dimensions, and
- display an error message explaining what shape the input must have, if the user gives a bad input.

# Exercise

## (from the survey)

Let X be a matrix of shape 7 x 3, Y be a matrix of shape 7 x 3, and Z be a vector of shape 3 x 1. We compute A = Y^T * X * Z, where ^T denotes transpose. What is the shape of A?

# Exercise

## (from the survey)

Let X be a matrix of shape (7, 3), Y be a matrix of shape (7, 3), and Z be a vector of shape (3, 1). We compute A = Y^T * X * Z, where ^T denotes transpose. What is the shape of A?

## Answer

# Exercise

## (from the survey)

Let X be a matrix of shape (7, 3), Y be a matrix of shape (7, 3), and Z be a vector of shape (3, 1). We compute A = Y^T * X * Z, where ^T denotes transpose. What is the shape of A?

## Answer

3 x 1

# Homework

- all the exercises above
- look through the first few slides here to recap data types, slice, reshape, and other handy things
  - https://colab.research.google.com/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/02.00-Introduction-to-NumPy.ipynb
- (extra - not required or expected) check out 3blue1brown on matrix multiplication + linear algebra