UC Berkeley Extension

Machine Learning With TensorFlow

X433.7 (2 semester units in COMPSCI)

Instructor Alexander I. Iliev, Ph.D.

Course Content Outline

- **Machine Learning With TensorFlow®**
- Introduction, Python - pros and cons
- Python modules, DL packages and scientific blocks
- Working with the shell, IPyton and the editor
- Installing the environment with core packages
- Writing “Hello World” HW1 (10pts)

- **Tensorflow and TensorBoard basics**
- Ecosystem, Competition, Users
- Linear algebra recap
- Data types in Numpy and Tensorflow
- Basic operations in Tensorflow
- Graph models and structures with Tensorboard

- **TensorFlow operations**
- Overloaded operators
- Using Aliases
- Sessions, graphs, variables, placeholders
- Name scopes

- **Data Mining and Machine Learning concepts**
- Basic Deep Learning Models, k-Means
- Linear and Logistic Regression
- Softmax classification HW2 (10pts)

- **Neural Networks 1/2**
- Multi-layer Neuaral Network
- Gradient descent and Backpropagation

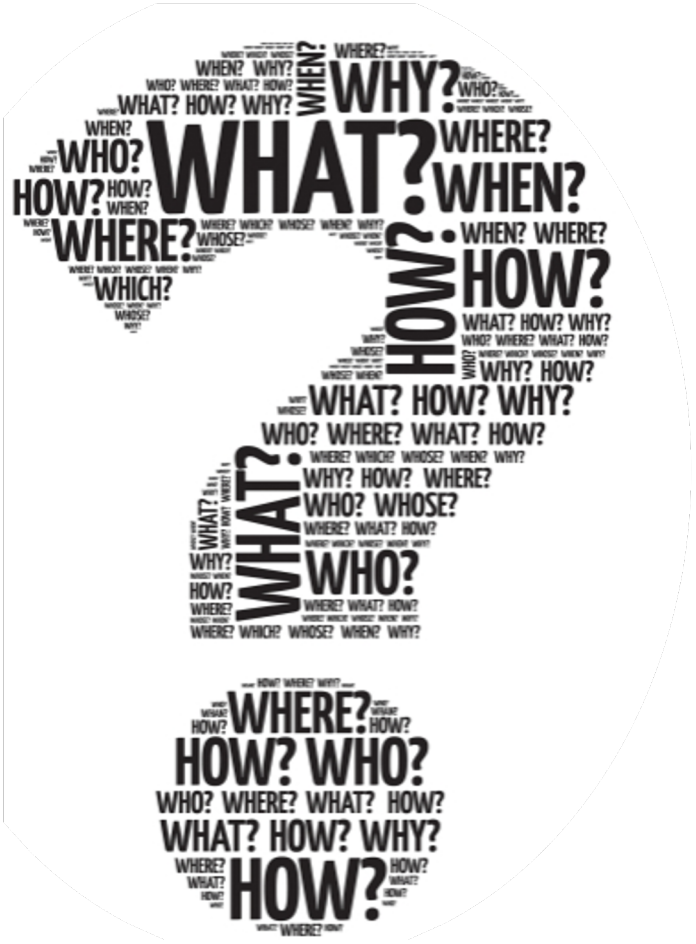
Machine Learning With TensorFlow

Lecture 2 …

Some key players, competitors and feature of TF

TensorFlow and TensorBoard, Data types, Linear algebra fundamentals …

TensorFlow Case Studies



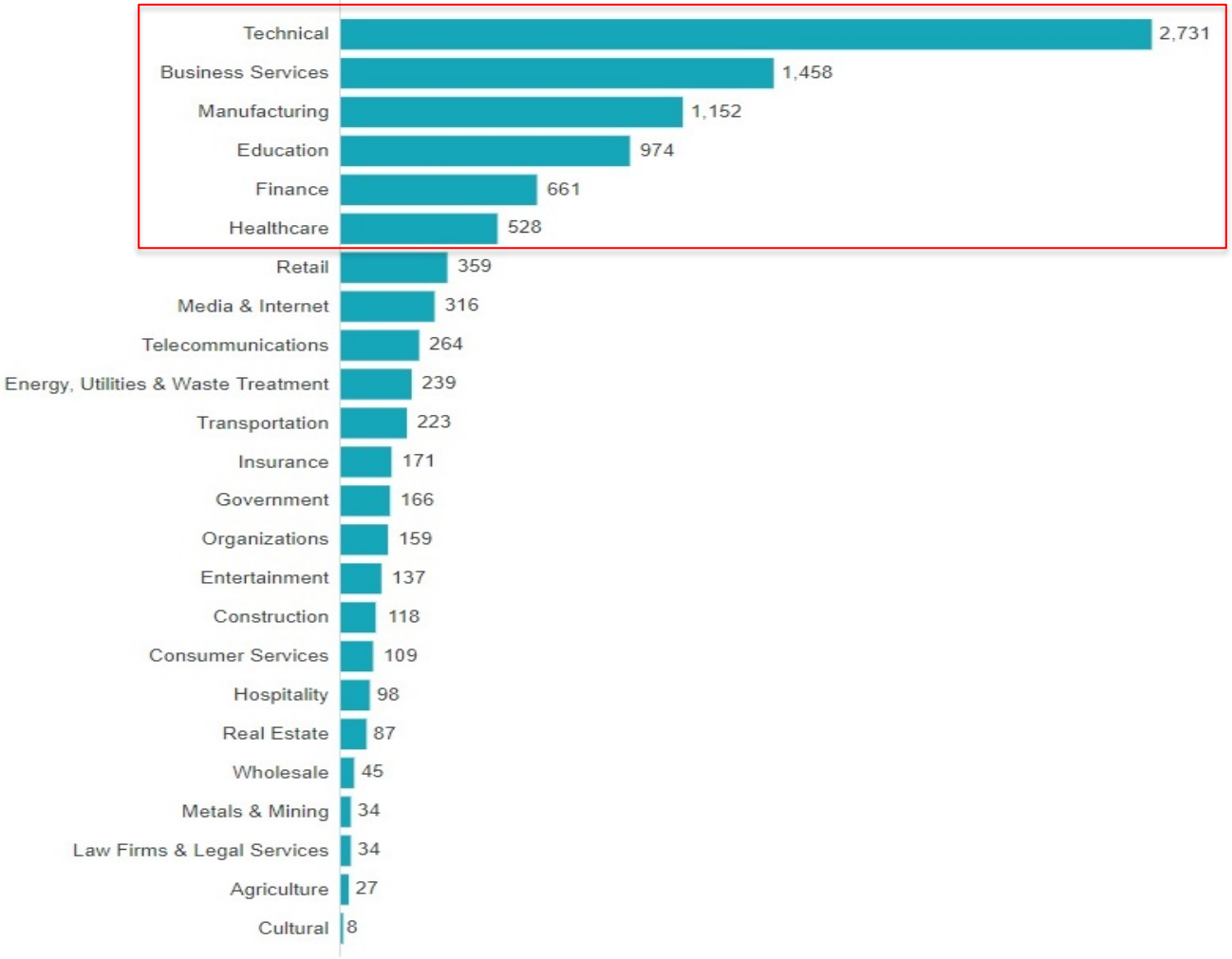
TensorFlow Case Studies

Lets have a quick look at:

- Who started using TensorFlow and Why?
- What are the different companies that are working with TensorFlow?
- How did they get benefited from it?

Companies using TensorFlow

- There are different types of companies that benefited from TensorFlow (creating quite a stir in various fields too)



Few companies that benefitted from using TensorFlow

- TensorFlow being an open package, it became the main paradigm that is being used for many solutions

- Some of the top companies that benefitted from TF are:

    - Airbnb
    - GE HealthCare
    - Airbus
    - Qualcomm

Few companies that benefitted from using TensorFlow

1) 

- Initially the team used to sort the images manually which consumed lots of time and energy
- A smarter way to over come such issues was to shift towards a better technology
- With TensorFlow the efficiency of their business was rapidly growing and this in turn gave the company a smoother experience and better results

Few companies that benefitted from using TensorFlow

1) 

- The following factors could have a steady improvement in the software:
    - Speeding up the search process
    - Prediction based on previous search, cost estimation
    - Discovering and classifying in-app message intent at Airbnb
    - Use of sequence models and LSTM networks to create suggested responses for our customer service agents
    - Learning Market dynamics for optimal pricing
    - Contextual Calendar Reminders

Few companies that benefitted from using TensorFlow

2) 

- The scan operators requires low resolution images to identify the specific location
- This procedure was complicated as the orientation, location, and coverage needs to be correct in all three spatial dimensions
- It is time-consuming and difficult, especially for complex anatomies

Few companies that benefitted from using TensorFlow

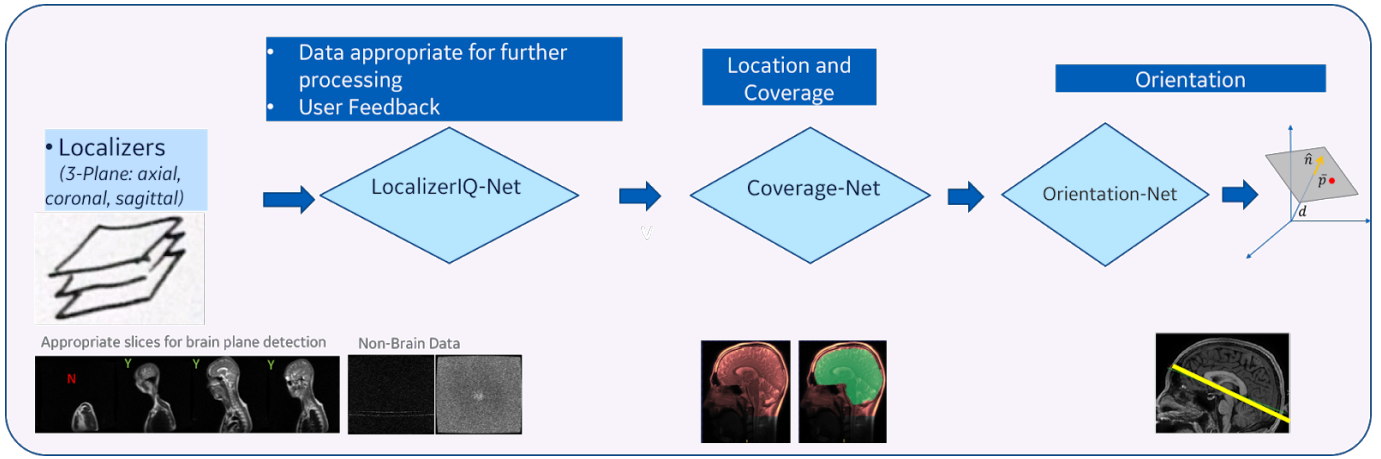
2)  GE Healthcare

- With TensorFlow in picture the company benefited:
  - Support for 2D and 3D Cascaded Convolutional Neural Networks (CCNN) which is the primary requirement for medical image volume processing
  - Continuous development with backward compatibility making it easier for code development and maintenance
  - Stability of graph computations made it attractive for product deployment
  - Extensive open-source user and developer community which supported latest algorithm implementations and made them readily available
  - Keras interface was available, which significantly reduced the development time

Few companies that benefitted from using TensorFlow

2) GE Healthcare

- They used TensorFlow for Intelligent Scanning Using Deep Learning for MRI
- The images now have better clarity for better diagnosis



- This Company is working wonders already in the section of brain. They are looking forward to expand further more to **knee** and **spine** as well.

Few companies that benefitted from using TensorFlow

3) 

- Airbus Defence and Space uses an algorithm to identify satellite images which is sometimes very difficult to determine, even for the human eye, whether an area on a satellite image is cloud or snow

- But it has an error rate of 11%, which has prompted Airbus to test the deep-learning platform TensorFlow

- TensorFlow analyses a large number of images, looks for recurring patterns, and uses this to learn how to identify objects by itself

- In the TensorFlow test phase, the error rate has improved to just 3%

Few companies that benefitted from using TensorFlow

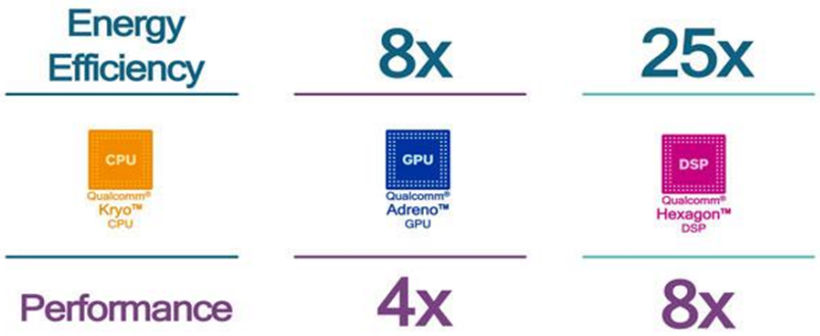
4) Qualcomm

- TensorFlow was designed to run on processing units inside of processors
- Snapdragon processors integrate a CPU, GPU and many technologies including Digital Signal Processor (DSP)
- The companies DSP architecture is designed to process certain audio and video features more quickly and at lower power than a CPU or GPU, which is why it's ideal to exploit its advantages

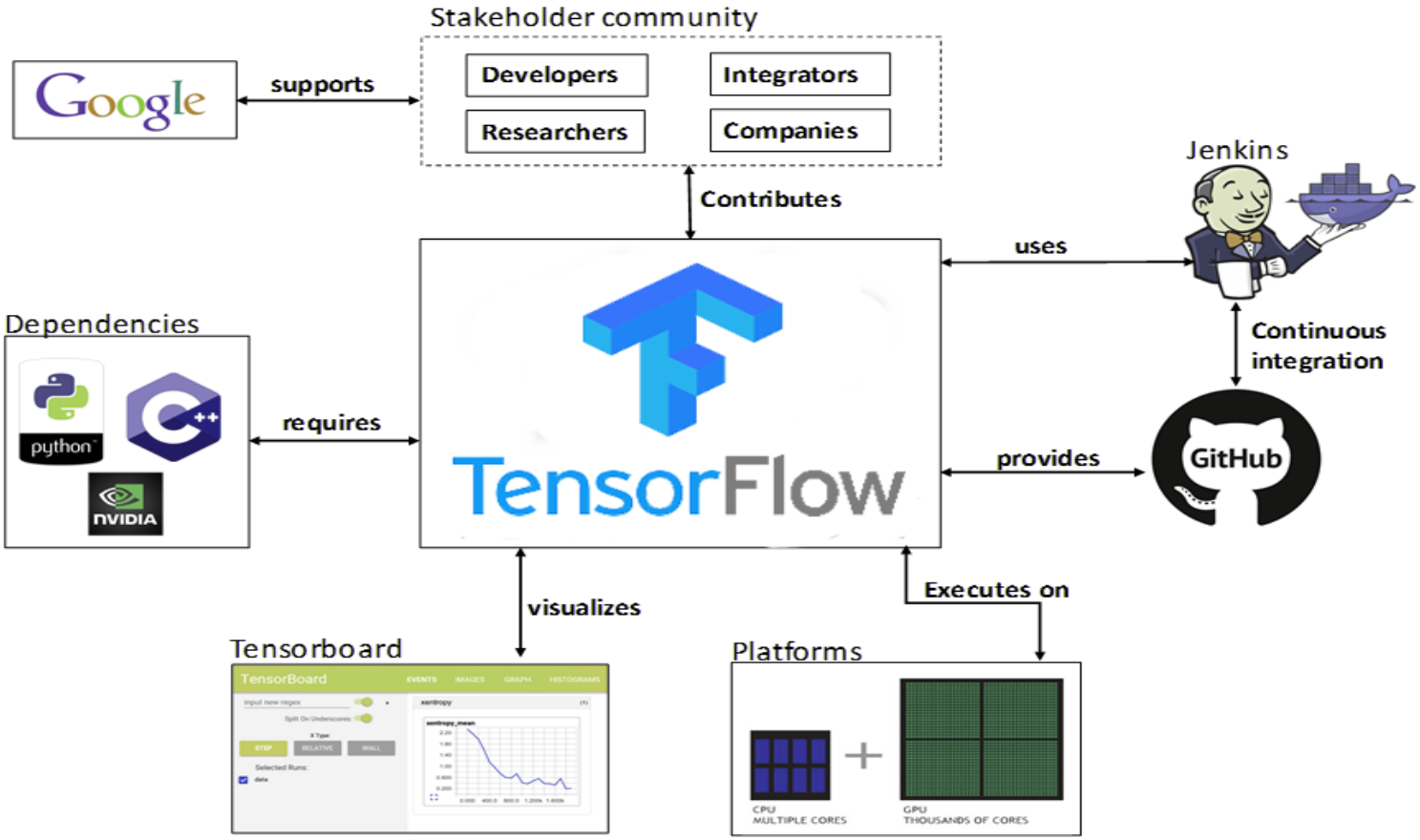
Few companies that benefitted from using TensorFlow

4) Qualcomm

- TensorFlow will run on the DSP so apps can run faster and more efficiently
- Qualcomm Technologies also offers support for TensorFlow on Snapdragon processors via the Qualcomm Snapdragon Neural Processing Engine SDK*
- The chart below illustrates the benefits of apps optimized for the DSP:



TensorFlow Ecosystem



TensorFlow Applications

- Tensorflow runs on a variety of platforms and the installation is Linux-only and more tedious than CPU-only installation.

- The applications go beyond deep learning to support other forms of machine learning:
  - reinforcement learning,
  - any goal-oriented tasks like winning video games or
  - helping a robot navigate an uneven landscape

- allows you to explore: sentiment analysis, google translate, text analysis and best of all image recognition

TensorFlow: how it works?

- Nodes and tensors in TensorFlow are Python objects, and TensorFlow applications are themselves Python applications
- The libraries of transformations that are available through TensorFlow are written as high-performance C++ binaries.
- Python just directs traffic between the pieces, and provides high-level programming abstractions to hook them together.
- TensorFlow applications can be run on most any target that's convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs.
- If you use Google's own cloud, you can run TensorFlow on Google's custom TensorFlow Processing Unit (TPU) silicon for further acceleration.

TensorFlow: how it works?

- TensorFlow 2.0, in beta as of June 2019, revamped the framework in many ways based on user feedback, to make it easier to work with and more performant
- Distributed training is easier to run thanks to a new API
- Support for TensorFlow Lite makes it possible to deploy models on a greater variety of platforms
- However, code written for earlier versions of TensorFlow must be rewritten—sometimes only slightly, sometimes significantly—to take maximum advantage of new TensorFlow 2.0 features.

TensorFlow v/s Competition

- TensorFlow competes with a slew of other machine learning frameworks
- PyTorch, CNTK, and MXNet are three major frameworks that address the same needs

1) PyTorch:

- In addition to being built with Python, and has many other similarities to TensorFlow:
  - Hardware-accelerated components under the hood
  - A highly interactive development model that allows for design-as-you-go work
  - A better choice for fast development of projects that need to be up and running in a short time, but TensorFlow wins on larger projects and more complex workflows

TensorFlow v/s Competition

- TensorFlow competes with a slew of other machine learning frameworks
- PyTorch, CNTK, and MXNet are three major frameworks that address the same needs

2) CNTK :

- The Microsoft Cognitive Toolkit (MCT)
- Uses a graph structure to describe dataflow
- Focuses mostly on creating deep learning neural networks
- Handles many neural network jobs faster
- But CNTK isn't as easy to learn or deploy as TensorFlow

TensorFlow v/s Competition

3) Apache Mxnet :

- Adopted by Amazon as the premier deep learning framework on AWS
- Can scale almost linearly across multiple GPUs and multiple machines.
- It also supports a broad range of language APIs—Python, C++, Scala, R, JavaScript, Julia, Perl, Go—although its native APIs aren't as pleasant to work with as TensorFlow's.

TensorFlow

PROS

GRAPHS :

1

has better computational graph visualizations, which are indigenous when compared to other libraries like Torch and Theano

LIBRARY MANAGEMENT:

2

Backed by Google, TensorFlow has the advantage of the seamless performance, quick updates and frequent new releases with new features

DEBUGGING :

3

lets you execute subparts of a graph which gives it an upper-hand as you can introduce and retrieve discrete data onto an edge

SCALABILITY :

4

The libraries can be deployed on a gamut of hardware machines, starting from cellular devices to computers with complex setups

CONS

1 MISSING SYMBOLIC LOOPS:

TensorFlow does not offer this feature, but there is a workaround using finite unfolding (bucketing).

2 LIMITED SUPPORT FOR WIN:

There is still a wide variety of users who are comfortable with a windows environment rather than a Linux in their systems and TensorFlow does not assuage these users.

3 BENCHMARK TESTS :

On small scale TensorFlow lacks behind in both speed and usage when compared to its competitors (not the case for large computations!)

Comparison Chart 1/3

	Platform	Cost	Written in language	Algorithms or Features
Scikit Learn	Linux, Mac OS, Windows	Free.	Python, Cython, C, C++	Classification Regression Clustering Preprocessing Model Selection Dimensionality reduction.
PyTorch	Linux, Mac OS, Windows	Free	Python, C++, CUDA	Autograd Module Optim Module nn Module
TensorFlow	Linux, Mac OS, Windows	Free	Python, C++, CUDA	Provides a library for dataflow programming.
Weka	Linux, Mac OS, Windows	Free	Java	Data preparation Classification Regression Clustering Visualization Association rules mining

Comparison Chart 2/3

KNIME	Linux, Mac OS, Windows	Free	Java	Can work with large data volume. Supports text mining & image mining through plugins
Colab	Cloud Service	Free	-	Supports libraries of PyTorch, Keras, TensorFlow, and OpenCV
Apache Mahout	Cross-platform	Free	Java Scala	Preprocessors Regression Clustering Recommenders Distributed Linear Algebra.
Accors.Net	Cross-platform	Free	C#	Classification Regression Distribution Clustering Hypothesis Tests & Kernel Methods Image, Audio & Signal. & Vision
Shogun	Windows Linux UNIX Mac OS	Free	C++	Regression Classification Clustering Support vector machines. Dimensionality reduction

Comparison Chart 3/3

Shogun	Windows Linux UNIX Mac OS	Free	C++	Regression Classification Clustering Support vector machines. Dimensionality reduction Online learning etc.
Keras.io	Cross-platform	Free	Python	API for neural networks
Rapid Miner	Cross-platform	Free plan Small: $2500 per year. Medium: $5000 per year. Large: $10000 per year.	Java	Data loading & Transformation Data preprocessing & visualization.

References

- https://www.tensorflow.org/install
- https://www.udemy.com/tutorial/complete-guide-to-tensorflow-for-deep-learning-with-python/installing-tensorflow-environment/
- https://github.com/tensorflow/docs/blob/master/site/en/r1/guide/extend/architecture.md
- https://www.tensorflow.org/guide/effective_tf2
- https://www.tensorflow.org/guide/keras/overview
- https://machinelearningmastery.com/introduction-python-deep-learning-library-tensorflow/
- https://hub.packtpub.com/tensorflow-always-tops-machine-learning-artificial-intelligence-tool-surveys/
- https://medium.com/tensorflow/intelligent-scanning-using-deep-learning-for-mri-36dd620882c4
- https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/1_Introduction/basic_operations.py
- https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html

Linear algebra

- Linear algebra deals with lines, planes, spaces and subspaces, and with the properties of all vector spaces (matrices and ndarrays in NumPy)
- N-dimensional spaces and hyperplanes are formed by a set of points with coordinates that satisfy given linear equation
- Hyperplanes are subspaces that are one dimension smaller than its original space (an ambient space). Example: 3D space has 2D (planes) hyperplanes, and so on and so forth
- Ambient space is the space encapsulating a mathematical object such as: a point or a line in a two-dimensional space – *plane*, or a three-dimensional space
- Linear algebra deals with the conditions in which a set of given *n* hyperplanes intersect in a single point. This kind of study is performed with a system of Linear equations, each containing several unknowns

Linear algebra

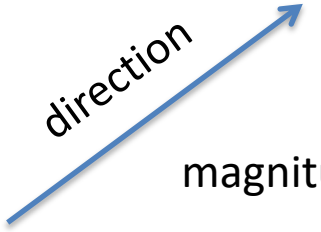
- Vector: 1) is a mathematical object that has a direction and a magnitude, used to find the position of one point in space relative to another point. 2) is a computer object, an array of data with individual items located with a single index
- Matrix is a 2-dimensional (rectangular) array of elements represented by: symbols, numbers, or expressions, all arranged in rows and columns. Matrix consist of vectors
- Array is an arrangement or a series of elements such as symbols, numbers, or expressions. Arrays can be n-dimensional, so matrix is an array with 2 dimensions
- Tensor is an object describing the linear relationship among scalars, vectors and other tensors
- Rank of a matrix is the maximum number of linearly independent column (or row) vectors in the matrix

Linear algebra

- Vector
  - 1) is a mathematical object that has a direction and a magnitude, used to find the position of one point in space relative to another point
  - 2) is a computer object, an array of data with individual items located with a single index

Example:

mathematical meaning



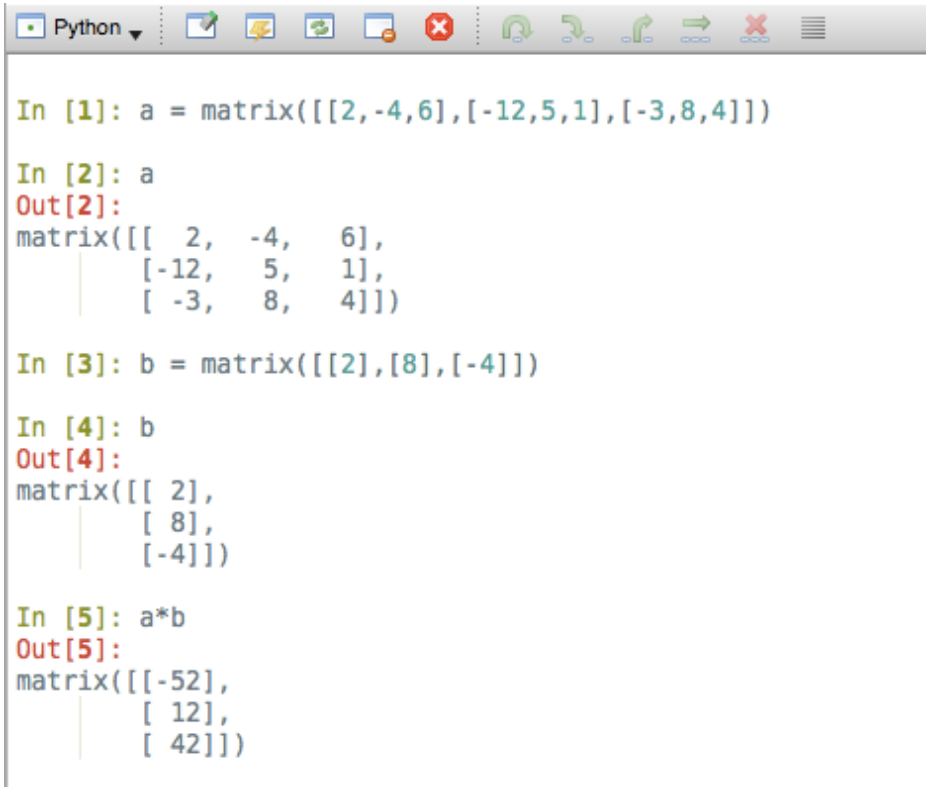
computer meaning

[3 2 4 5 -6 9 2 10]

Linear algebra

- Matrix is a 2-dimensional (rectangular) array of elements represented by: symbols, numbers, or expressions, all arranged in rows and columns. Matrix consist of vectors

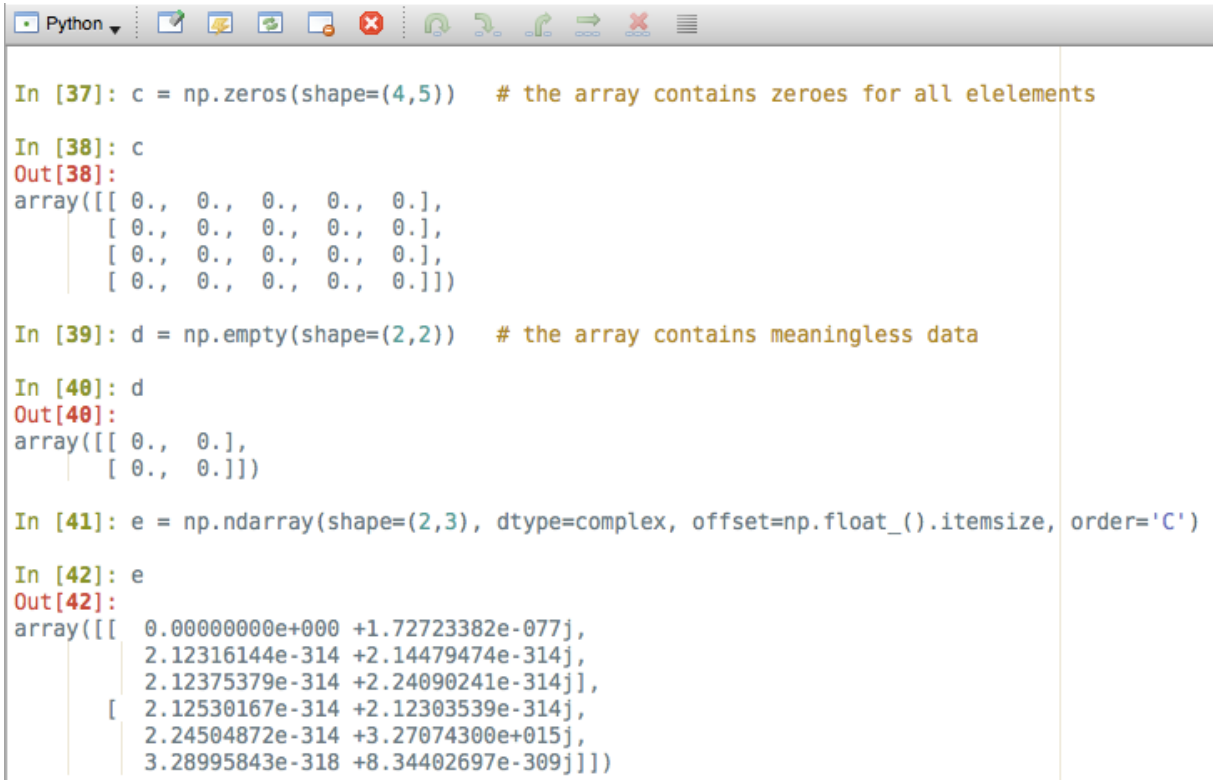
Example:



Linear algebra

- Array can be n-dimensional, but matrix is an array with 2 dimensions

Example: numpy.array is a function that returns a numpy.ndarray



Linear algebra

- NumPy array

Example:

```
In [23]: a = np.array([[12, 34, 41], [54, 62, 18], [72, 84, 96]], np.int16)

In [24]: a
Out[24]:
array([[12, 34, 41],
       [54, 62, 18],
       [72, 84, 96]], dtype=int16)

In [25]: a.size
Out[25]: 9

In [26]: a.shape
Out[26]: (3, 3)

In [27]: type(a)
Out[27]: numpy.ndarray

In [28]: a.dtype
Out[28]: dtype('int16')

In [29]: a[2,2]   # this is how we index a particular elemnt in the array (#9)
Out[29]: 96

In [30]: b = a[0,:]

In [31]: b
Out[31]: array([12, 34, 41], dtype=int16)

In [32]: b.shape
Out[32]: (3,)

In [33]: b[2] = 88   # this is how we reassign another value to a member in the array

In [34]: a[2,2] = 99 # the change above also affects the original array

In [35]: a
Out[35]:
array([[12, 34, 88],
       [54, 62, 18],
       [72, 84, 99]], dtype=int16)

In [36]: b
Out[36]: array([12, 34, 88], dtype=int16)
```

Linear algebra

- Difference between a numpy.matrix and 2D numpy.ndarray
    - basic operations such as multiplications and transpose are included in NumPy for both matrix and ndarray types
    - numpy.matrix has a proper interface for matrix operations than numpy.ndarray
    - however, the numpy.matrix class does not add anything that cannot be achieved by using 2D numpy.ndarray objects
    - the implementation of this class resembles the one in Matlab
    - scipy.linalg operations can be used just as good to 2D numpy.ndarray objects as well as to numpy.matrix

Linear algebra

- Difference between a numpy.matrix and 2D numpy.ndarray

the I, T and H serve as shortcuts for inverse, transpose and conjugate transpose respectively

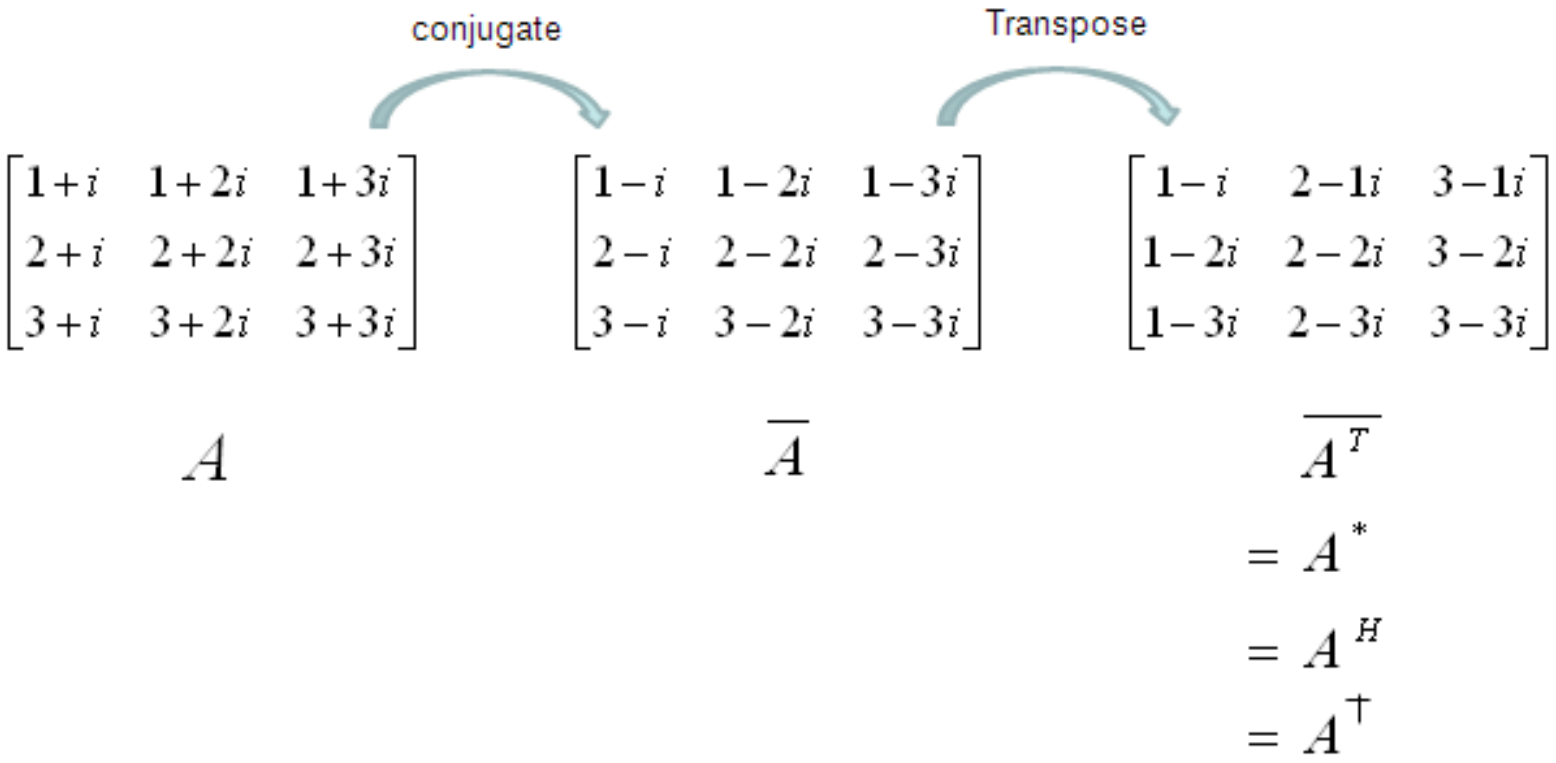
the numpy.matrix class does not add anything that cannot be achieved by using 2D numpy.ndarray objects

```
Python
In [45]: # Matrix vs Array:

In [46]: # 1. Matrix:

In [47]: import numpy as npy

In [48]: a = npy.mat('[12 34 41;52 64 72]') # create matrix 'a'

In [49]: a
Out[49]:
matrix([[12, 34, 41],
        [52, 64, 72]])

In [50]: type(a)
Out[50]: numpy.matrixlib.defmatrix.matrix

In [51]: a.I # inverse of matrix 'a'
Out[51]:
matrix([[-0.05885099,  0.03258602],
        [ 0.01490374, -0.00181295],
        [ 0.02925572, -0.00803395]])

In [52]: b = npy.mat('[84 92]') # create matrix 'b'

In [53]: b
Out[53]: matrix([[84, 92]])

In [54]: type(b)
Out[54]: numpy.matrixlib.defmatrix.matrix

In [55]: b.T # transpose of matrix 'b'
Out[55]:
matrix([[84],
        [92]])

In [56]: a.T*b.T # multiplication of two matrices
Out[56]:
matrix([[ 5792],
        [ 8744],
        [10068]])
```

```
Python
In [45]: a
Out[45]: array(9)

In [46]: b
Out[46]: matrix([[9.-2.j, 4.-4.j]])

In [47]: b.H
Out[47]:
matrix([[9.+2.j],
        [4.+4.j]])

In [48]: a*b
Out[48]: matrix([[81.-18.j, 36.-36.j]])

In [49]: a*b.H
Out[49]:
matrix([[81.+18.j],
        [36.+36.j]])
```

Linear algebra

- Conjugate transpose:



Source: http://www.sharetechnote.com/html/Handbook_EngMath_Matrix_HermitianConjugate.html

Linear algebra

- Difference between a numpy.matrix and 2D numpy.ndarray

scipy.linalg operations can be used just as good to 2D numpy.ndarray objects as well as to numpy.matrix

Note:
npy.mat and npy.matrix are the same. Try, using 'id'

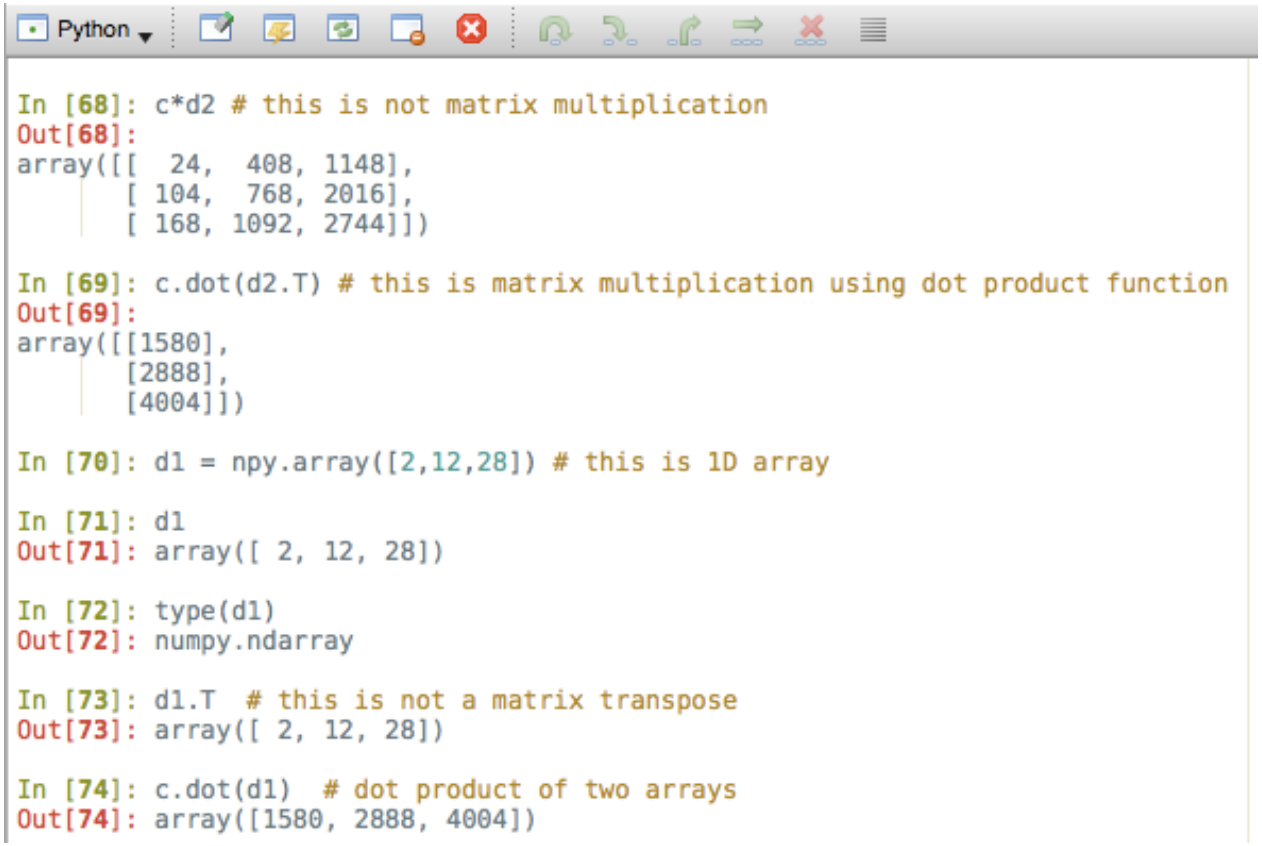
Python

```
In [57]: # Matrix vs Array:

In [58]: # 2. Array:

In [59]: import numpy as npy

In [60]: from scipy import linalg

In [61]: c = npy.array([[12,34,41],[52,64,72],[84,91,98]]) # create array 'c'

In [62]: c
Out[62]:
array([[12, 34, 41],
       [52, 64, 72],
       [84, 91, 98]])

In [63]: type(c)
Out[63]: numpy.ndarray

In [64]: linalg.inv(c)   # calculate the inverse of a matrix
Out[64]:
array([[-0.10752688,  0.15322581, -0.06758833],
       [ 0.3655914 , -0.87096774,  0.48694316],
       [-0.24731183,  0.67741935, -0.38402458]])

In [65]: d2 = npy.array([[2,12,28]]) # create 2D array

In [66]: d2
Out[66]: array([[ 2, 12, 28]])

In [67]: type(d2)
Out[67]: numpy.ndarray
```

Linear algebra

- Difference between a numpy.matrix and 2D numpy.ndarray

scipy.linalg operations can be used just as good to 2D numpy.ndarray objects as well as to numpy.matrix



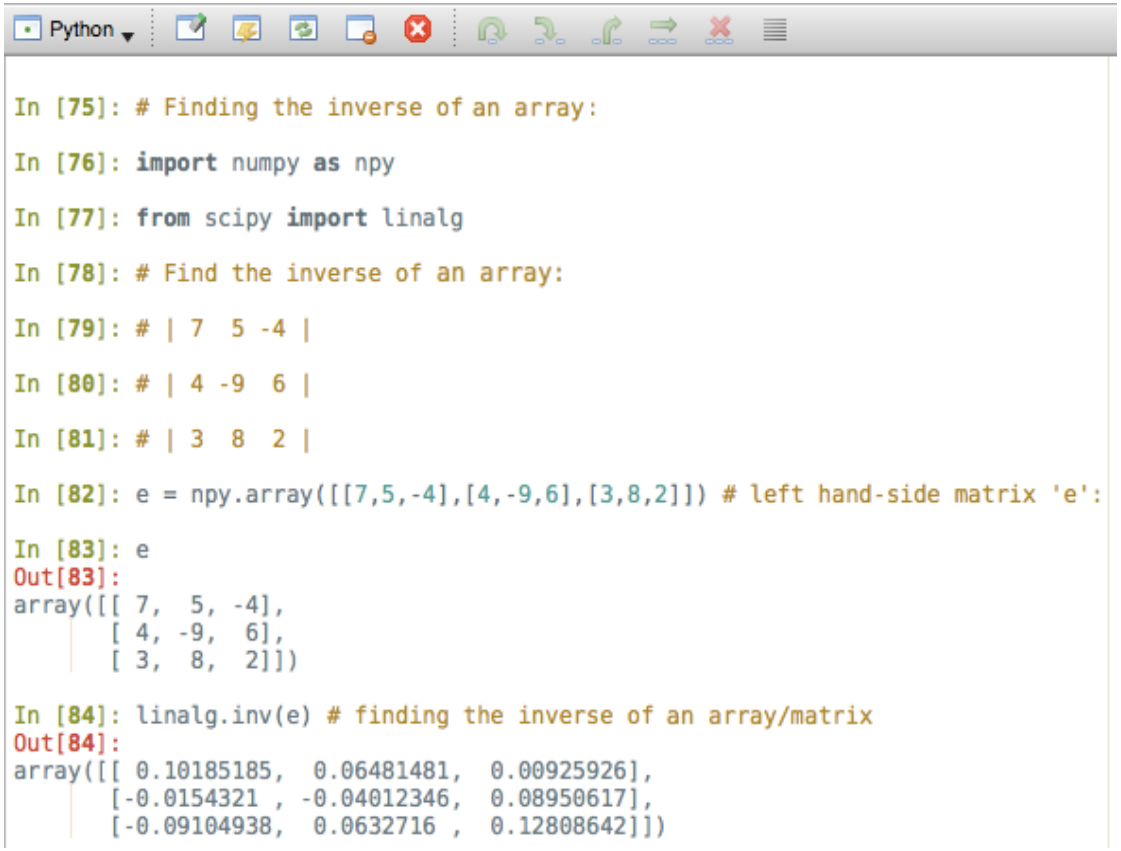
Linear algebra

- Finding the inverse of an array (matrix)

1) matrix ‘e’ has its inverse matrix ‘f’ such that e*f=I -> I is the identity matrix that has main diagonal with ones

2) we can then say that: $f=e^{-1}$

3) the matrix inverse of NumPy matrix ‘e’ is obtained in two ways:
   a) using the Scipy linalg.inv, or
   b) using e.I when ‘e’ is a matrix so cast it like this: npy.mat(e).I



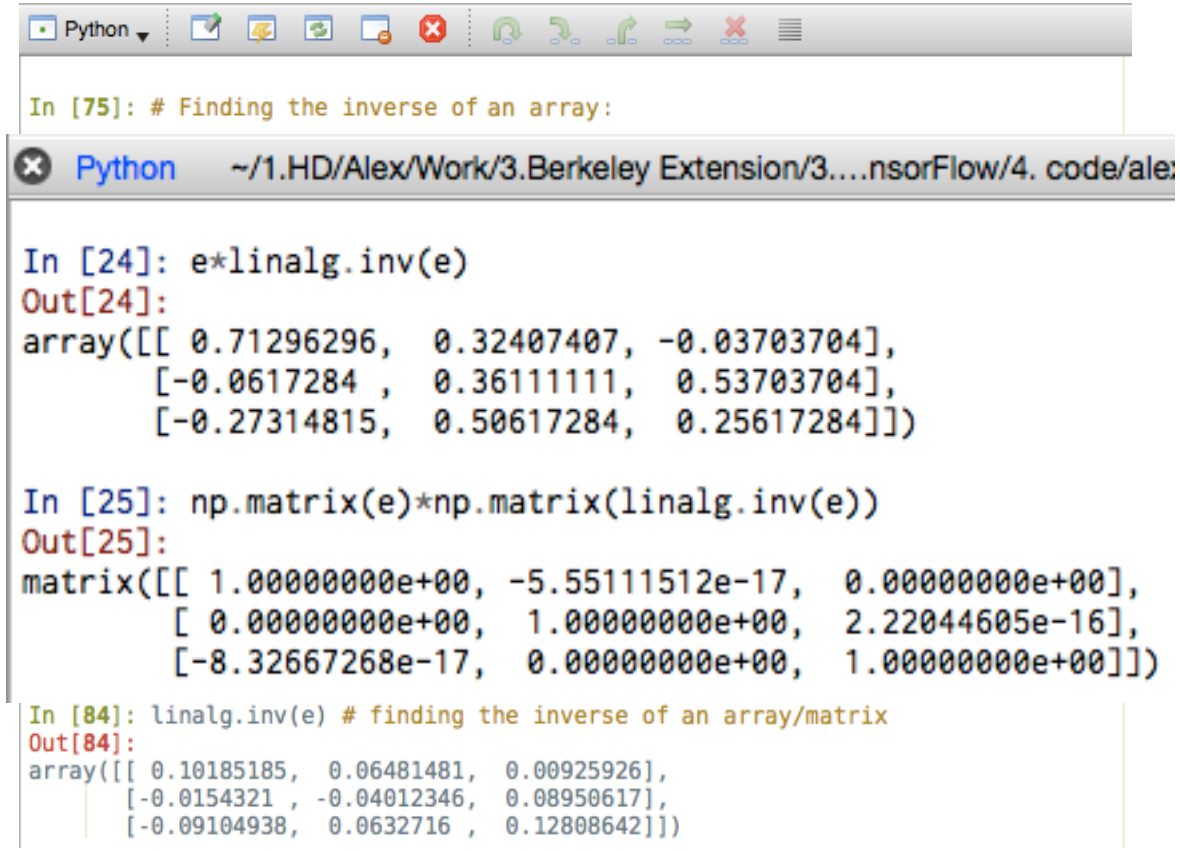
Linear algebra

- Finding the inverse of an array (matrix)

1) matrix ‘e’ has its inverse matrix ‘f’ such that e*f=I -> I is the identity matrix that has main diagonal with ones

2) we can then say that: $f=e^{-1}$

3) the matrix inverse of NumPy matrix ‘e’ is obtained in two ways:
   a) using the Scipy linalg.inv, or
   b) using e.I when ‘e’ is a matrix so cast it like this: npy.mat(e).I



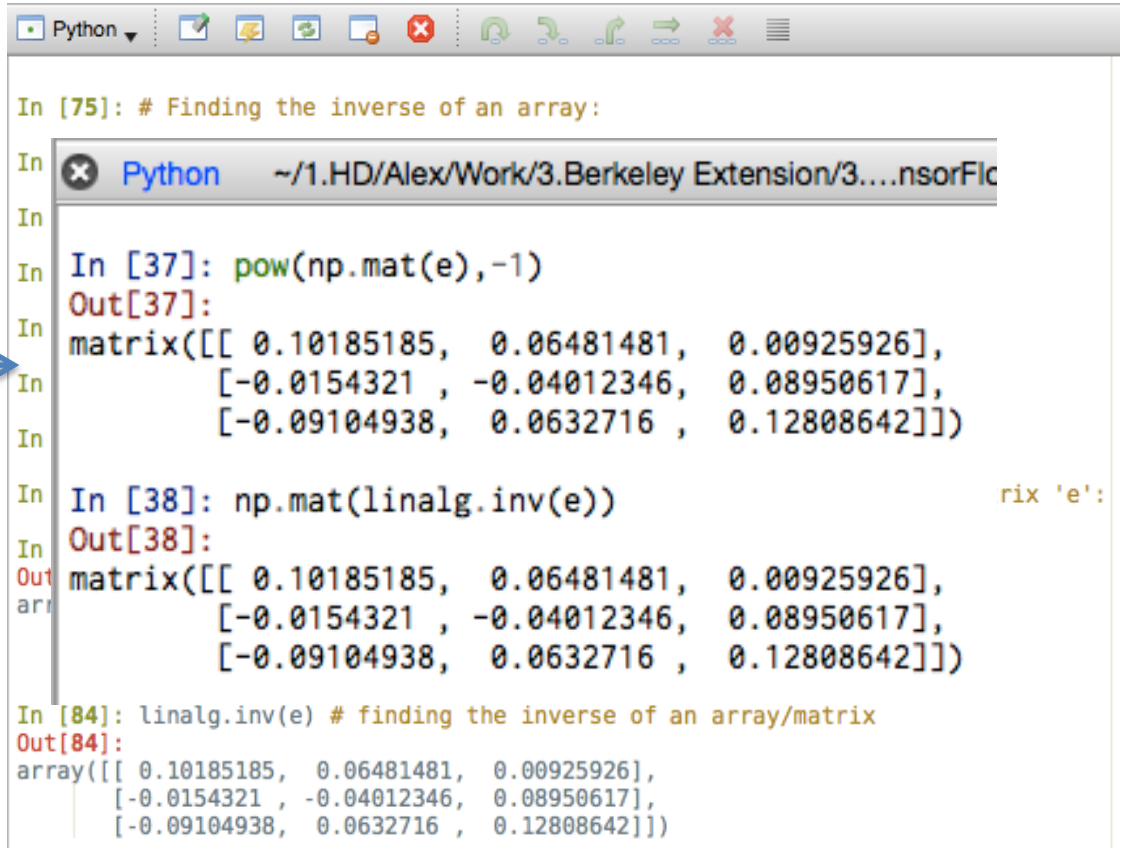
Linear algebra

- Finding the inverse of an array (matrix)

1) matrix 'e' has its inverse matrix 'f' such that e*f=I -> I is the identity matrix that has main diagonal with ones

2) we can then say that: $f=e^{-1}$

3) the matrix inverse of NumPy matrix 'e' is obtained in two ways:
   a) using the Scipy linalg.inv, or
   b) using e.I when 'e' is a matrix so cast it like this: npy.mat(e).I



What are tensors?

- Tensors are a generalization of vectors and matrices and are easily understood as a multidimensional array

- A vector is a one-dimensional or first order tensor and a matrix is a two-dimensional or second order tensor.

  t111, t121, t131 t112, t122, t132 t113, t123, t133

- T = (t211, t221, t231), (t212, t222, t232), (t213, t223, t233)

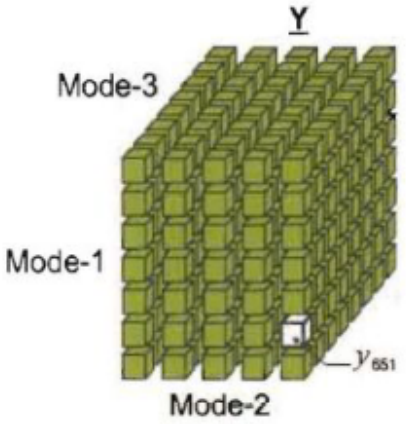
  t311, t321, t331 t312, t322, t332 t313, t323, t333

- Set of techniques known in machine learning in the training and operation of deep learning models can be described in terms of tensors

What are tensors?

What's tensor? Why tensor and tensor factorization?

- Definition: a tensor is a multidimensional array which is an extension of matrix.
- Tensor can happen in daily life.
- In order to facilitate information mining from tensor and tensor processing, storage, tensor factorization is often needed.
- Three-way tensor:



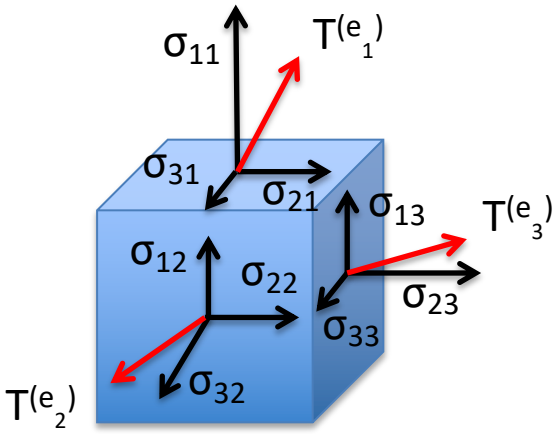
A three-way array (third-order tensor) $\underline{\mathbf{Y}} \in \mathbb{R}^{7\times5\times8}$ with elements y_{iiq}.

What are tensors?

- Tensor is an objects describing the linear relationship among scalars, vectors and other tensors

Example:

a 2nd order tensor of a 3-dimensional space represent the matrix (or a 2-D array):



$$\sigma = [T^{(e_1)} T^{(e_2)} T^{(e_3)}] \quad \text{or} \quad \sigma = \begin{bmatrix} \sigma_{11} \sigma_{12} \sigma_{13} \\ \sigma_{21} \sigma_{22} \sigma_{23} \\ \sigma_{31} \sigma_{32} \sigma_{33} \end{bmatrix}$$

where, the columns are the forces e_n depicted on the 3 faces of the cube (e_1, e_2, e_3)

What are tensors?

- Tensor is an objects describing the linear relationship among scalars, vectors and other tensors
  - A 0^{th} order tensor can be represented by a scalar
  - A 1^{st} order tensor can be represented by an array (vector)
  - A 2^{nd} order tensor can be represented by a matrix
  - A 3^{rd} order tensor can be represented as a 3-dimensional array of numbers
  - However tensor represents more than just an arrangement of components:
    - tensor shows how the array transforms upon a change of its basis
    - tensor is a ndarray satisfying a particular transformation law

Linear algebra

- Rank of a matrix is the maximum number of linearly independent column (or row) vectors in the matrix, denoted as: *rk(A)* or *rank(A)*

  - The rank *R* of a tensor is independent of the number of dimensions *N* of the underlying space

  - Rank-0 is a scalar – $N^0 = 1$
  - Rank-1 is an array (vector) – $N^1 = N$
  - Rank-2 is a matrix – $N^2 = N \times N$ *aka dyad, dyadic*
  - Rank-3 is a 3-darray – $N^3 = N \times N \times N$ *aka triad*
  - Rank-4 is a 4-darray – $N^4 = N \times N \times N \times N$ *aka tetrad*
  - *etc*

Rank - number of simultaneous directions

Linear algebra

- Rank of a matrix is the maximum number of linearly independent column (or row) vectors in the matrix, denoted as: *rk(A)* or *rank(A)*

  – How to find the rank:
    - They are generally: 0,1,2 and 3:

      Rank(A) = 0 when matrix is null

      Rank(A) = 1 when every sub-matrix of A is singular or $\det(A_n) = 0$

      Rank(A) = 2 when A is singular **and** at least one of its sub-matrix is $\det(A_1) \neq 0$

      Rank(A) = 3 when A is non-singular or $\det(A_n) \neq 0$

Example:

$$A = \begin{bmatrix} -1 & -1 & 0 \\ 4 & 2 & 2 \\ 3 & 1 & 2 \end{bmatrix}$$,

1) det(A) = (-1)(2*2-2*1)-(-1)(4*2-2*3)+(0)(4*1-2*3) = -2 +2 +0 = 0
=> it is singular

2) $\det(A_1)$ = 4*1-2*3 = 4-6 = -2 -- > *A_1 is a sub-matrix of A*
=> it is ≠ 0 and is non-singular

Therefore Rank(A) = 2

Linear algebra

- Rank of a matrix is the maximum number of linearly independent column (or row) vectors in the matrix, denoted as: *rk(A)* or *rank(A)*

  - How to find the rank:
    - They are generally: 0,1,2 and 3:

      Rank(A) = 0 when matrix is null

      Rank(A) = 1 when every sub-matrix of A is singular or $\det(A_n) = 0$

      Rank(A) = 2 when A is singular **and** at least one of its sub-matrix is $\det(A_1) \neq 0$

      Rank(A) = 3 when A is non-singular or $\det(A_n) \neq 0$

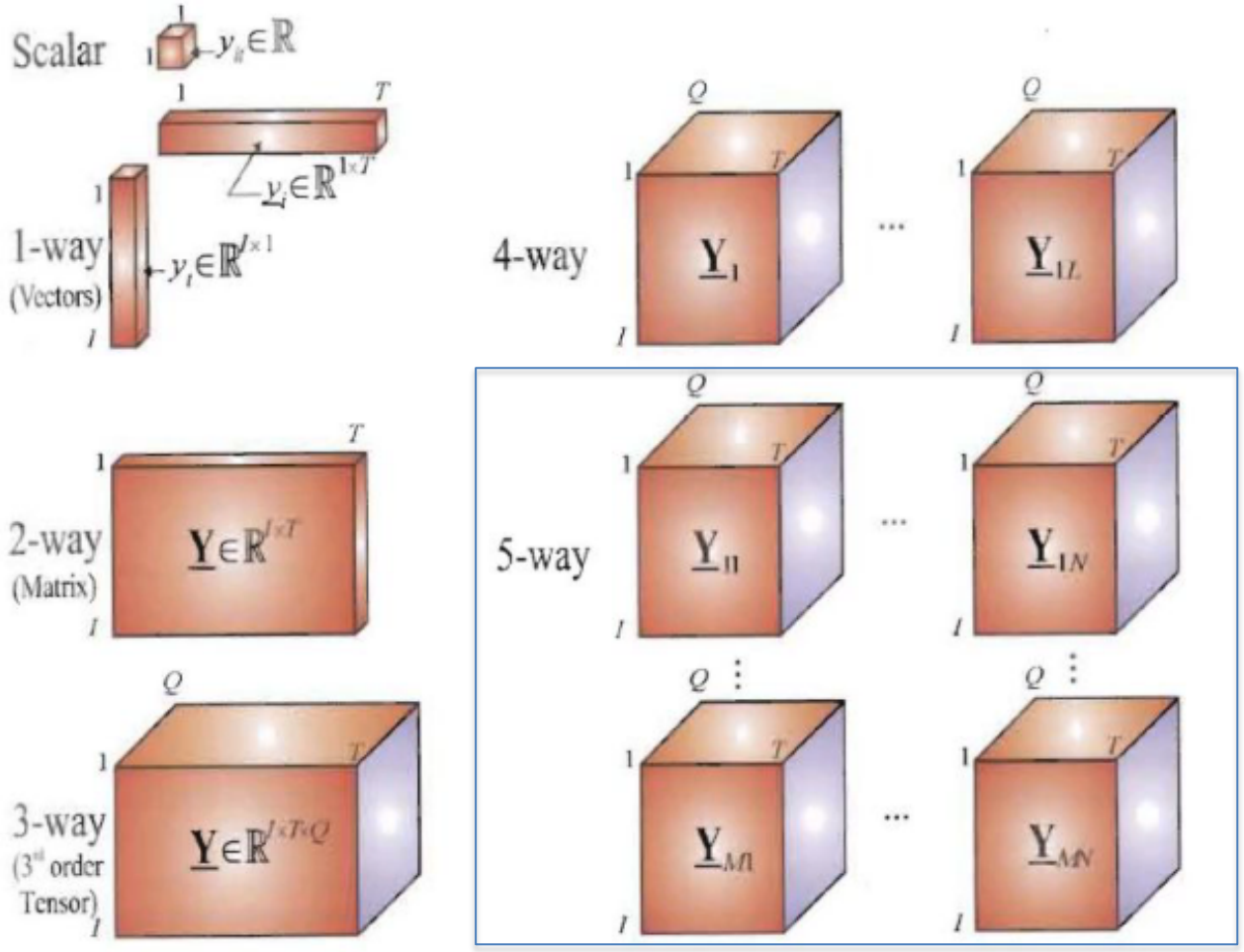
Example:

$$A = \begin{bmatrix} -1 & -1 & 0 \\ 4 & 2 & 2 \\ 3 & 1 & 2 \end{bmatrix}$$

```
Python
In [1]: from numpy import linalg as lin

In [2]: from numpy import matrix as mt

In [3]: A=mt([[-1,-1,0],[4,2,2],[3,1,2]])

In [4]: A
Out[4]:
matrix([[-1, -1,  0],
        [ 4,  2,  2],
        [ 3,  1,  2]])

In [5]: lin.matrix_rank(A)
Out[5]: 2
```

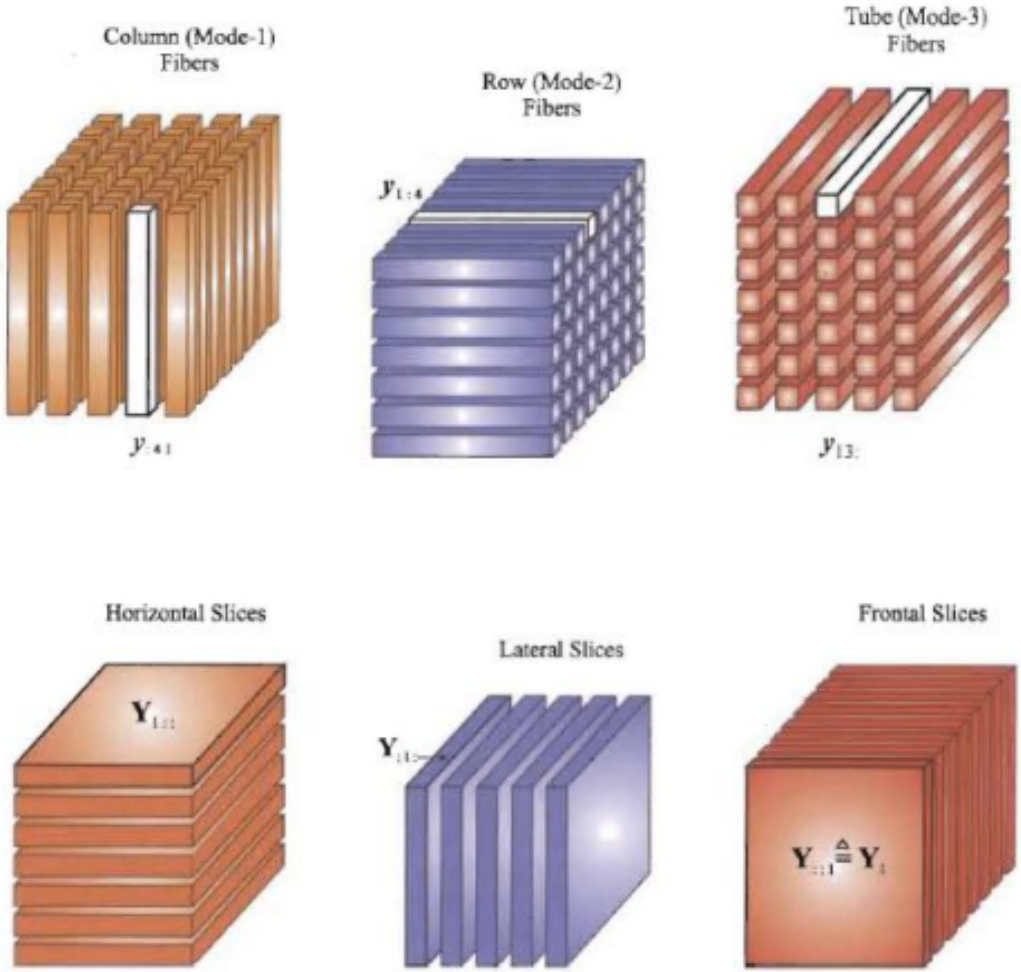
Linear algebra

A tensor is a multidimensional array



Linear algebra

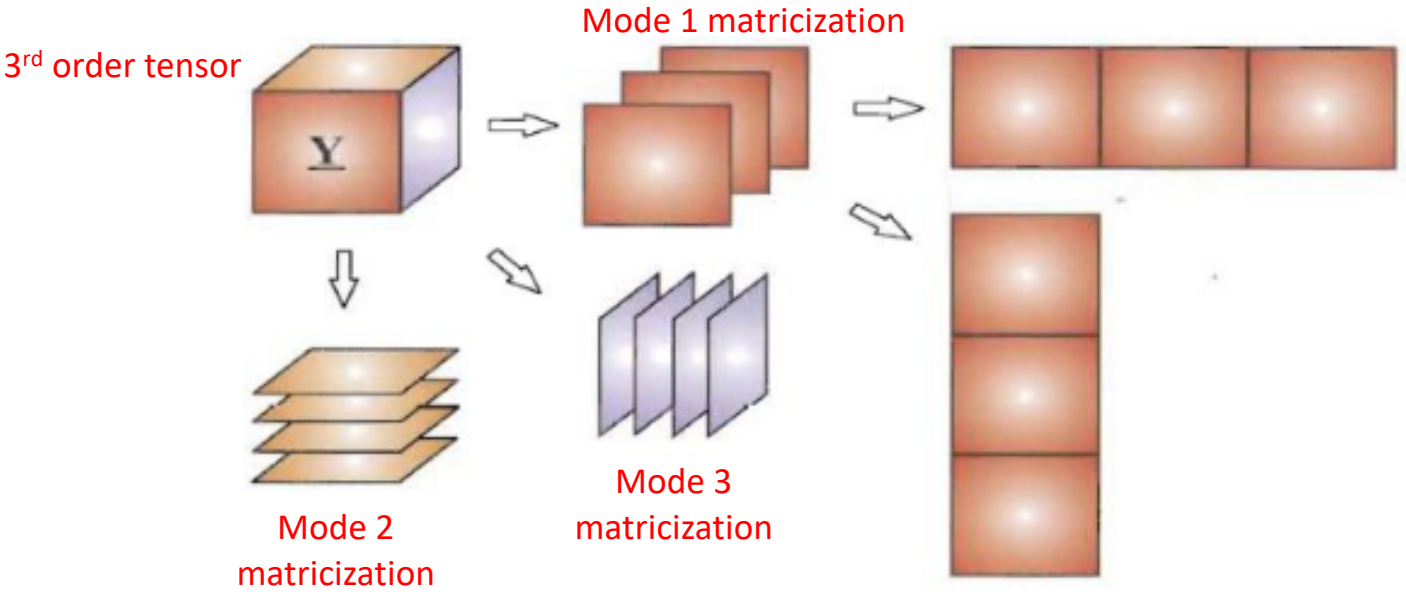
Fiber and slice



Linear algebra

Tensor unfoldings: Matricization and vectorization

- Matricization: convert a tensor to a matrix

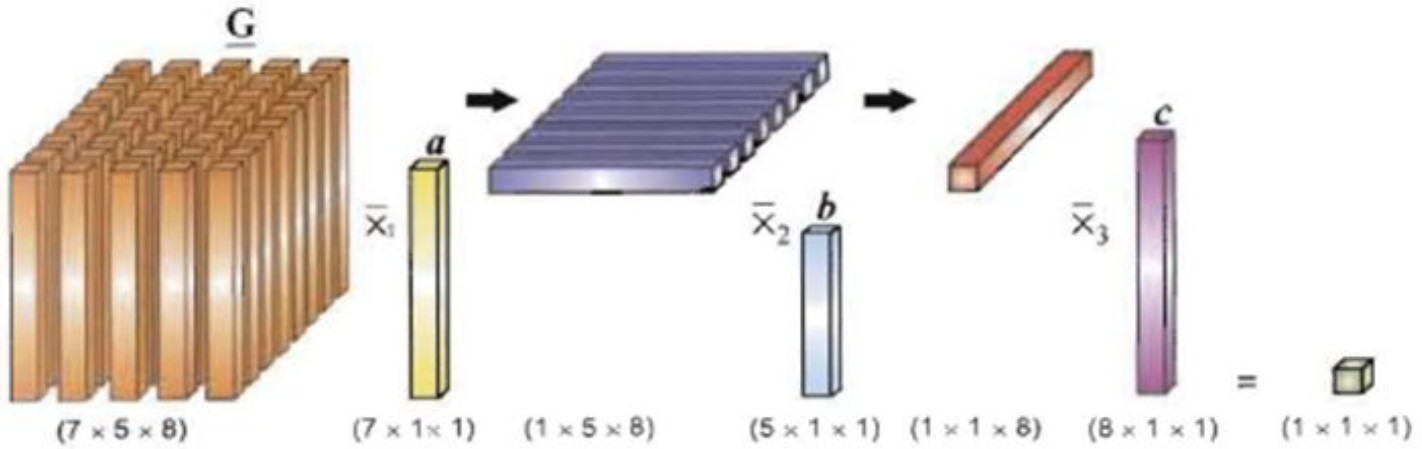


- Vectorization: convert a tensor to a vector

Linear algebra

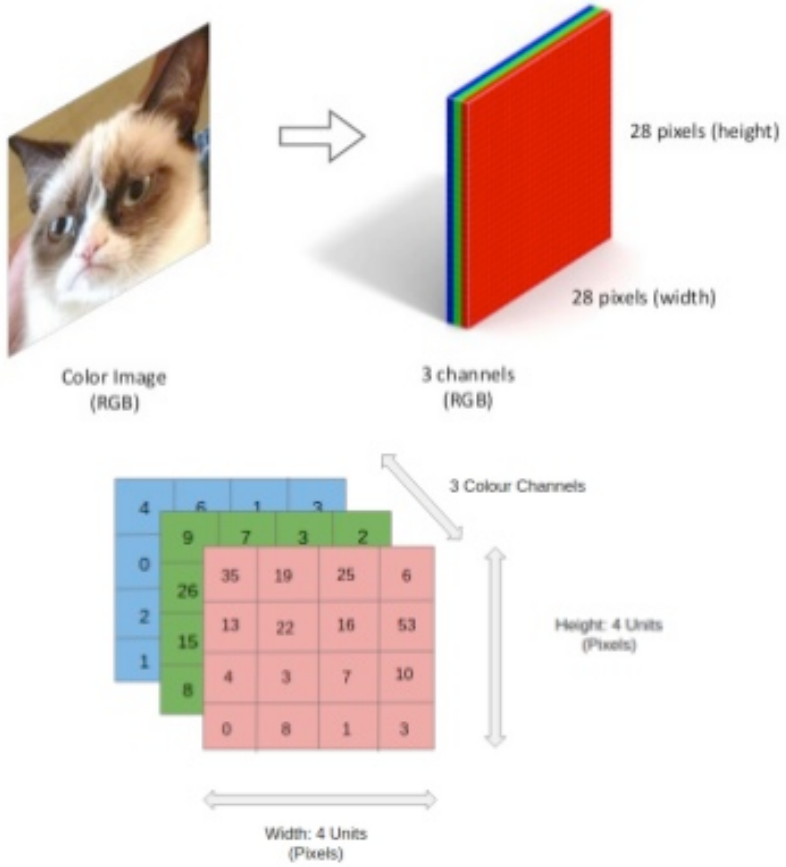
Tensor multiplication: the n-mode product: multiplied by a vector

- Definition: $$(\mathcal{X} \bar{\times}_n \mathbf{v})_{i_1 \cdots i_{n-1} i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \cdots i_N} v_{i_n}.$$



- Note: multiplying by a vector reduces the dimension by one.

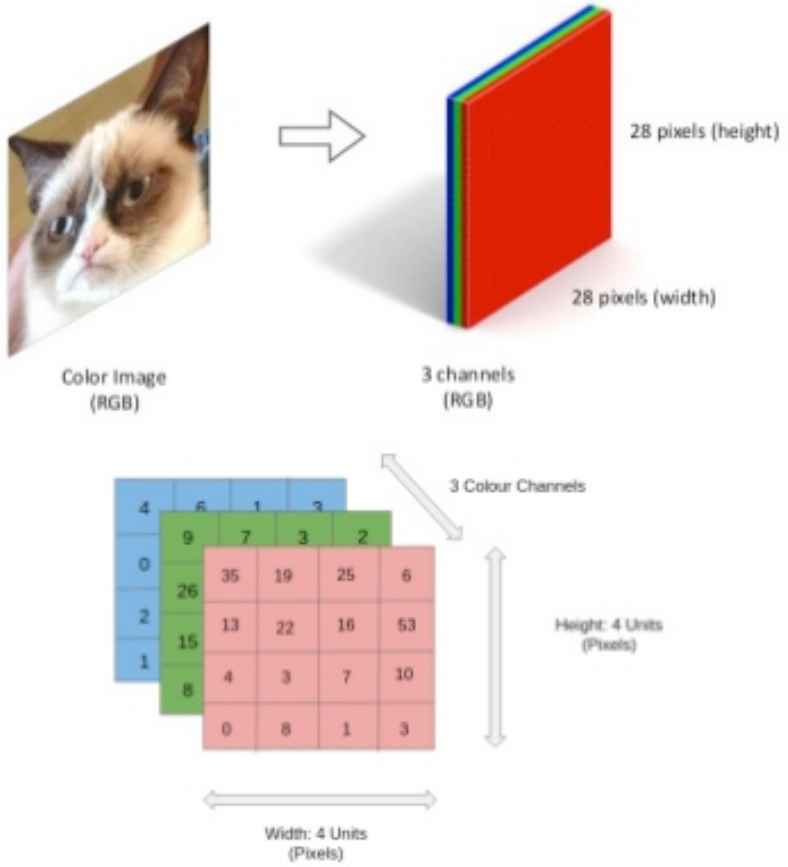
Linear algebra



Source: http://matthewalanham.com/Students/2018_PURC_ImageClassification.pdf

Linear algebra

color image is 3rd-order tensor



Source: http://matthewalanham.com/Students/2018_PURC_ImageClassification.pdf

NumPy data type objects

- Data type objects
  - NumPy supports much larger variety of types than what the standard Python implementation does:

Number type	Data type	Description
Booleans	bool, bool8, bool_	Boolean (True or False) stored as a byte – 8 bits
Integers	byte	compatible: C char – 8 bits
	short	compatible: C short – 16 bits
	int, int0, int_	Default integer type (same as C `long`; normally either `int32` or `int64`) – 64 bits
	longlong	compatible: C long long – 64 bits
	intc	Identical to C `int` – 32 bits
	intp	Integer used for indexing (same as C `size_t`) – 64 bits
	int8	Byte (-128 to 127) – 8 bits
	int16	Integer (-32768 to 32767) – 16 bits
	int32	Integer (-2147483648 to 2147483647) – 32 bits
	int64	Integer (-9223372036854775808 to 9223372036854775807) – 64 bits
Unsigned integers	uint, uint0	Python int compatible, unsigned – 64 bits
	ubyte	compatible: C unsigned char, unsigned – 8 bits
	ushort	compatible: C unsigned short, unsigned – 16 bits
	ulonglong	compatible: C long long, unsigned – 64 bits
	uintp	large enough to fit a pointer – 64 bits
	uintc	compatible: C unsigned int – 32 bits
	uint8	Unsigned integer (0 to 255) – 8 bits
	uint16	Unsigned integer (0 to 65535) – 16 bits
	uint32	Unsigned integer (0 to 4294967295) – 32 bits
	uint64	Unsigned integer (0 to 18446744073709551615) – 64 bits

NumPy data type objects

- Data type objects
  - NumPy supports much larger variety of types than what the standard Python implementation does:

Number type	Data type	Description
Floating-point numbers	half	compatible: C short – 16 bits
	single	compatible: C float – 32 bits
	double	compatible: C double – 64 bits
	longfloat	compatible: C long float – 128 bits
	float_	Shorthand for `float64` – 64 bits
	float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
	float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
	float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
	float128	128 bits
Complex floating-point numbers	csingle	64 bits
	complex, complex_	Shorthand for `complex128` – 128 bits
	complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
	complex128	Complex number, represented by two 64-bit floats (real and imaginary components)
	complex256	two 256 bit floats

  - To check how many bits each type occupies, use one of these notations:
    1) (np.dtype(np.<type>).itemsize)*8
    2) np.<type>().itemsize*8

NumPy data type objects

- Data type objects
  - the difference between signed and unsigned integers and long type variables is:
    - the signed and unsigned types are of the same size
    - the signed can represent equal amount of values around the '0' thus representing equal amount of positive and negative numbers
    - the unsigned can not represent any negative numbers, but can represent double the amount of total positive numbers as compared to the signed type
    - for 32-bit int we have:

      int: –2147483648 to 2147483647

      uint: 0 to 4294967295
    - for 64-bit long we have:

      long: -9223372036854775808 to 9223372036854775807

      ulong: 0 to 18446744073709551615

TensorFlow

- Below is the full list of data types available in TensorFlow:

```
@tf_export("DType")
class DType(object):
  """Represents the type of the elements in a `Tensor`.

  The following `DType` objects are defined:

  * `tf.float16`: 16-bit half-precision floating-point.
  * `tf.float32`: 32-bit single-precision floating-point.
  * `tf.float64`: 64-bit double-precision floating-point.
  * `tf.bfloat16`: 16-bit truncated floating-point.
  * `tf.complex64`: 64-bit single-precision complex.
  * `tf.complex128`: 128-bit double-precision complex.
  * `tf.int8`: 8-bit signed integer.
  * `tf.uint8`: 8-bit unsigned integer.
  * `tf.uint16`: 16-bit unsigned integer.
  * `tf.uint32`: 32-bit unsigned integer.
  * `tf.uint64`: 64-bit unsigned integer.
  * `tf.int16`: 16-bit signed integer.
  * `tf.int32`: 32-bit signed integer.
  * `tf.int64`: 64-bit signed integer.
  * `tf.bool`: Boolean.
  * `tf.string`: String.
  * `tf.qint8`: Quantized 8-bit signed integer.
  * `tf.quint8`: Quantized 8-bit unsigned integer.
  * `tf.qint16`: Quantized 16-bit signed integer.
  * `tf.quint16`: Quantized 16-bit unsigned integer.
  * `tf.qint32`: Quantized 32-bit signed integer.
  * `tf.resource`: Handle to a mutable resource.
  * `tf.variant`: Values of arbitrary types.
```

TensorFlow

- Examples:

```
In [1]: import tensorflow as tf

In [2]: import numpy as np

In [3]: sess = tf.Session()

In [4]: a = 92

In [5]: type(a)
Out[5]: int

In [6]: a
Out[6]: 92

In [7]: a = tf.convert_to_tensor(a, dtype = tf.float16)

In [8]: type(a)
Out[8]: tensorflow.python.framework.ops.Tensor
```

```
In [9]: a
Out[9]: <tf.Tensor 'Const:0' shape=() dtype=float16>

In [10]: sess.run(a)
Out[10]: 92.0

In [11]: type(sess.run(a))
Out[11]: numpy.float16

In [12]: a = sess.run(a)

In [13]: type(a)
Out[13]: numpy.float16

In [14]: a
Out[14]: 92.0

In [15]: a = tf.cast(a, tf.float32)

In [16]: type(a)
Out[16]: tensorflow.python.framework.ops.Tensor

In [22]: a = tf.cast(a, np.float64)

In [23]: type(a)
Out[23]: tensorflow.python.framework.ops.Tensor

In [24]: type(sess.run(a))
Out[24]: numpy.float64

In [27]: a = np.int16(sess.run(a))

In [28]: type(a)
Out[28]: numpy.int16

In [29]: a
Out[29]: 92
```

... try it in class

TensorFlow

- Few extra notes:
  - Graphs save computation time
  - Graphs break computation into small pieces to facilitate auto-differentiation
  - Graphs handle distributed computation, that is they spread the work across multiple CPUs, GPUs, or devices
  - Many machine learning models are taught and visualized as directed graphs
  - Nodes in the graph are called **ops** (short for **operations**)
  - An **op** takes zero or more Tensors, performs some computation, and produces zero or more Tensors
  - **Sessions** gives us the **environment** to perform **operations** on our **tensor data**

TensorFlow

- Few extra notes on types:
  - Using Python types to specify Tensor objects is quick and easy, and it is useful for prototyping ideas
  - However, there is an important and unfortunate downside to this:
    - TensorFlow has a plethora of data types, but basic Python types lack the ability to explicitly state what kind of data type we'd like to use!
    - Instead, TensorFlow has to infer which data type it was meant *
    - TensorFlow is tightly integrated with NumPy, the scientific computing package designed for manipulating ndarrays

TensorFlow

- Few extra notes on types:

1. In TensorFlow, all data passed from node to node are Tensor objects

2. TensorFlow *Operations* can look at standard Python types, such as integers and strings, and automatically convert them into tensors

3. There are a variety of ways to create Tensor objects manually (that is, without reading it in from an external data source)

4. … so let's see them next: …

TensorFlow

- Few extra notes on types:
  - TensorFlow can take in Python numbers, booleans, strings, or lists of any of the above
  - Single values will be converted to a 0-D Tensor (or scalar)
  - Lists of values will be converted to a 1-D Tensor (vector)
  - Lists of lists of values will be converted to a 2-D Tensor (matrix), and so on

TensorFlow

- Here is a small chart showcasing this:

```
t_0 = 50                                # Treated as 0-D Tensor,
or "scalar"

t_1 = [b"apple", b"peach", b"grape"] # Treated as 1-D Tensor,
or "vector"

t_2 = [[True, False, False],         # Treated as 2-D Tensor,
or "matrix"
       [False, False, True],
       [False, True, False]]

t_3 = [[ [0, 0], [0, 1], [0, 2] ],   # Treated as 3-D Tensor
       [ [1, 0], [1, 1], [1, 2] ],
       [ [2, 0], [2, 1], [2, 2] ]]
...
```


Source: TensorFlow for Machine Intelligence

TensorFlow

- Few extra notes on types (cont.):
  - TensorFlow's data types are based on those from NumPy
  - Tensors are just a superset of matrices!
  - In fact, the statement np.int32 == tf.int32 returns True!
  - Any NumPy array can be passed into any TensorFlow Op

  The String data types problem:

  - For numeric and boolean types, TensorFlow and NumPy dtypes match well
  - However, tf.string does not have an exact match in NumPy due to the way NumPy handles strings
  - That said, TensorFlow can import string arrays from NumPy perfectly fine - just don't specify a dtype in NumPy!

TensorFlow

- Unlike Python, where a string can be treated as a list of characters, TensorFlow's tf.strings are indivisible values

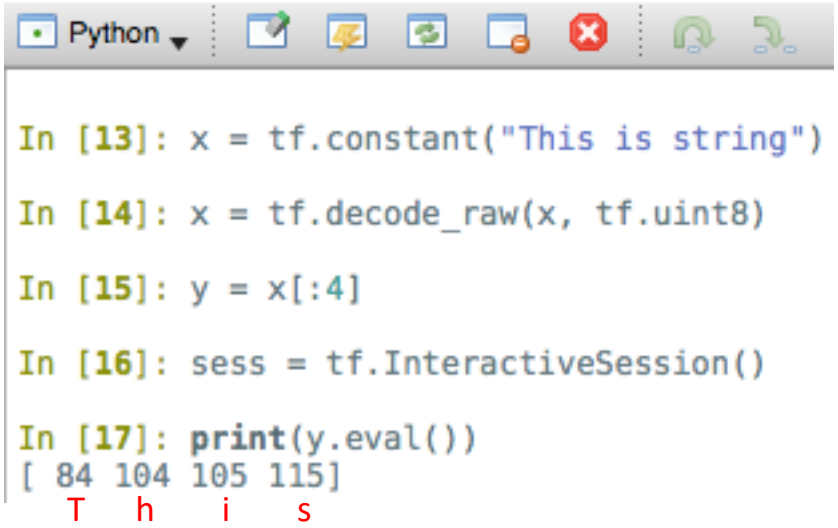
  Example:

  'x' is a Tensor with shape (2,) and each element inside is a variable length string

  x = tf.constant(["This is a string", "This is another string"])

- TensorFlow provides the tf.decode_raw operator that takes tf.string tensor as input and can decode the string into any other primitive data type

- To interpret the string as a tensor of characters, you can do:



TensorFlow

- Few extra notes on types (cont.):
  - You can use the functionality of the numpy library both before and after running your graph, as the tensors returned from Session.run **are** NumPy arrays
  - Here's an example of how to create NumPy arrays, mirroring the previous example:

```
import numpy as np  # Don't forget to import NumPy!

# 0-D Tensor with 32-bit integer data type
t_0 = np.array(50, dtype=np.int32)

# 1-D Tensor with byte string data type
# Note: don't explicitly specify dtype when using strings in
NumPy
t_1 = np.array([b"apple", b"peach", b"grape"])

# 2-D Tensor with boolean data type
t_2 = np.array([[True, False, False],
                [False, False, True],
```


Source: TensorFlow for Machine Intelligence

TensorFlow

- Few extra notes on types (cont.):
  - You can use the functionality of the numpy library both before and after running your graph, as the tensors returned from Session.run **are** NumPy arrays
  - Here's an example of how to create NumPy arrays, mirroring the previous example:

```
                [False, True, False]],
               dtype=np.bool)

# 3-D Tensor with 64-bit integer data type
t_3 = np.array([[ [0, 0], [0, 1], [0, 2] ],
                [ [1, 0], [1, 1], [1, 2] ],
                [ [2, 0], [2, 1], [2, 2] ]],
               dtype=np.int64)
...
```


Source: TensorFlow for Machine Intelligence

Tensor Shape

- The shape in TensorFlow terminology describes both:
  - the number of dimensions in a tensor as well as
  - the length of each dimension

- Tensor shapes can either be Python lists or tuples containing an ordered set of integers:

  - there are as many numbers in the list as there are dimensions, and each number describes the length of its corresponding dimension

    Example:

    the list [2, 3] describes the shape of a 2-D tensor of length 2 in its first dimension and length 3 in its second dimension

* Note that either: - tuples (wrapped with parentheses ()) or
  - lists (wrapped with brackets []) can be used to define shapes

Tensor Shape

- Some examples:

```
# Shapes that specify a 0-D Tensor (scalar)
# e.g. any single number: 7, 1, 3, 4, etc.
s_0_list = []
s_0_tuple = ()

# Shape that describes a vector of length 3
# e.g. [1, 2, 3]
s_1 = [3]
```

```
# Shape that describes a 3-by-2 matrix
# e.g [[1 ,2],
#      [3, 4],
#      [5, 6]]
s_2 = (3, 2)
```


Source: TensorFlow for Machine Intelligence

Tensor Shape

- In addition to being able to specify fixed lengths to each dimension, we are also able to assign a flexible length by passing in None as a dimension's value
- This will tell TensorFlow to allow a tensor of any shape
- That is, a tensor with any amount of dimensions and any length for each dimension:

```
# Shape for a vector of any length:
s_1_flex = [None]

# Shape for a matrix that is any amount of rows tall, and 3
columns wide:
s_2_flex = (None, 3)

# Shape of a 3-D Tensor with length 2 in its first dimension,
and variable-
# length in its second and third dimensions:
s_3_flex = [2, None, None]

# Shape that could be any Tensor
s_any = None
```


Source: TensorFlow for Machine Intelligence

Tensor Shape

- If we ever need to figure out the shape of a tensor in the middle of our graph, we can use the tf.shape Op

- It simply takes in the Tensor object we'd like to find the shape of, and returns it as an int32 vector:

```
import tensorflow as tf

# ...create some sort of mystery tensor

# Find the shape of the mystery tensor
shape = tf.shape(mystery_tensor, name="mystery_shape")
```

** Remember that tf.shape, like any other Operation, doesn't run until it is executed inside of a Session*

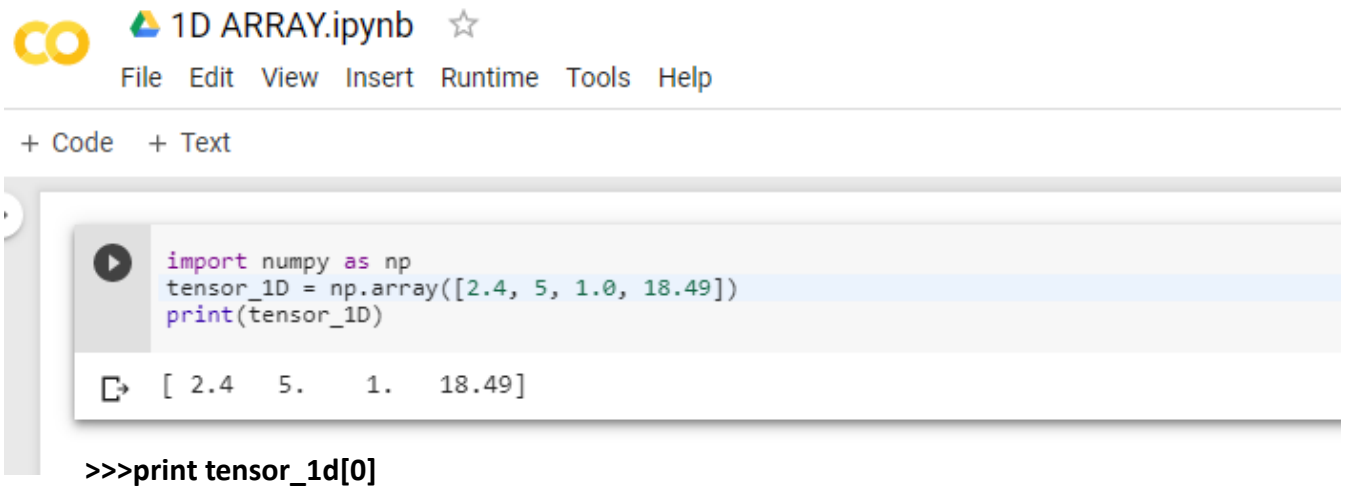
Source: TensorFlow for Machine Intelligence

Basic Code Examples

- **EXAMPLE 1**: CREATING A ONE DIMENSIONAL ARRAY

- One dimensional tensor - normal array structure that has a set of values of the same type

```
>>> import numpy as np
>>> tensor_1D = np.array([2.4, 5, 1.0, 18.49])
>>> print (tensor_1D)
```




```
>>>print tensor_1d[0]
2.4

>>>print tensor_1d[2]
1
```

Basic Code Examples

- **EXAMPLE 2** : CREATING A TWO DIMENSIONAL ARRAY :

- Two dimensional Tensors Sequence of arrays are used for creating “two dimensional tensors”

```
>>>import numpy as np

>>> tensor_2d=np.array([(1,2,3,4),(4,5,6,7),(8,9,10,11),(12,13,14,15)])

>>> print(tensor_2d)
[[ 1 2 3 4]
[ 4 5 6 7]
[ 8 9 10 11]
[12 13 14 15]]
```

Basic Code Examples

- **EXAMPLE 3 :** CREATING MULTIDIMENSIONAL ARRAY

```
import tensorflow as tf

import numpy as np

m1 = np.array([(3,3,3),(3,3,3),(3,3,3)],dtype='int32')

m2 = np.array([(2,2,2),(2,2,2),(2,2,2)],dtype='int32')

print (m1)

print (m2)

m1 = tf.constant(m1)

m2 = tf.constant(m2)

m_product = tf.matmul(m1, m2)

m_sum = tf.add(m1,m2)
```

```
[[3 3 3]
 [3 3 3]
 [3 3 3]]
[[2 2 2]
 [2 2 2]
 [2 2 2]]
```

try it in class …

Basic Code Examples

```
m_3 = np.array([(4,5,6),(3,4,1),(2,0,1)],dtype='float32')

print (m_3)

m_det = tf.matrix_determinant(m_3)

with tf.Session() as sess:

        result1 = sess.run(m_product)

        result2 = sess.run(m_sum)

        result3 = sess.run(m_det)

print (result1)

print (result2)

print (result3)
```

```
In [7]: print (result1)
   ...:
   ...: print (result2)
   ...:
   ...: print (result3)
[[18 18 18]
 [18 18 18]
 [18 18 18]]
[[5 5 5]
 [5 5 5]
 [5 5 5]]
-37.000004
```

try it in class …

Basic Code Examples

matrixmultiplication.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

```
import tensorflow as tf
import numpy as np
m1 = np.array([(3,3,3,),(3,3,3),(3,3,3)],dtype='int32')
m2 = np.array([(2,2,2),(2,2,2),(2,2,2)],dtype='int32')
print (m1)
print (m2)
m1 = tf.constant(m1)
m2 = tf.constant(m2)
m_product = tf.matmul(m1, m2)
m_sum = tf.add(m1,m2)
m_3 = np.array([(4,5,6),(3,4,1),(2,0,1)],dtype='float32')
print (m_3)
m_det = tf.matrix_determinant(m_3)
with tf.Session() as sess:
 result1 = sess.run(m_product)
 result2 = sess.run(m_sum)
 result3 = sess.run(m_det)
print (result1)
print (result2)
print (result3)
```

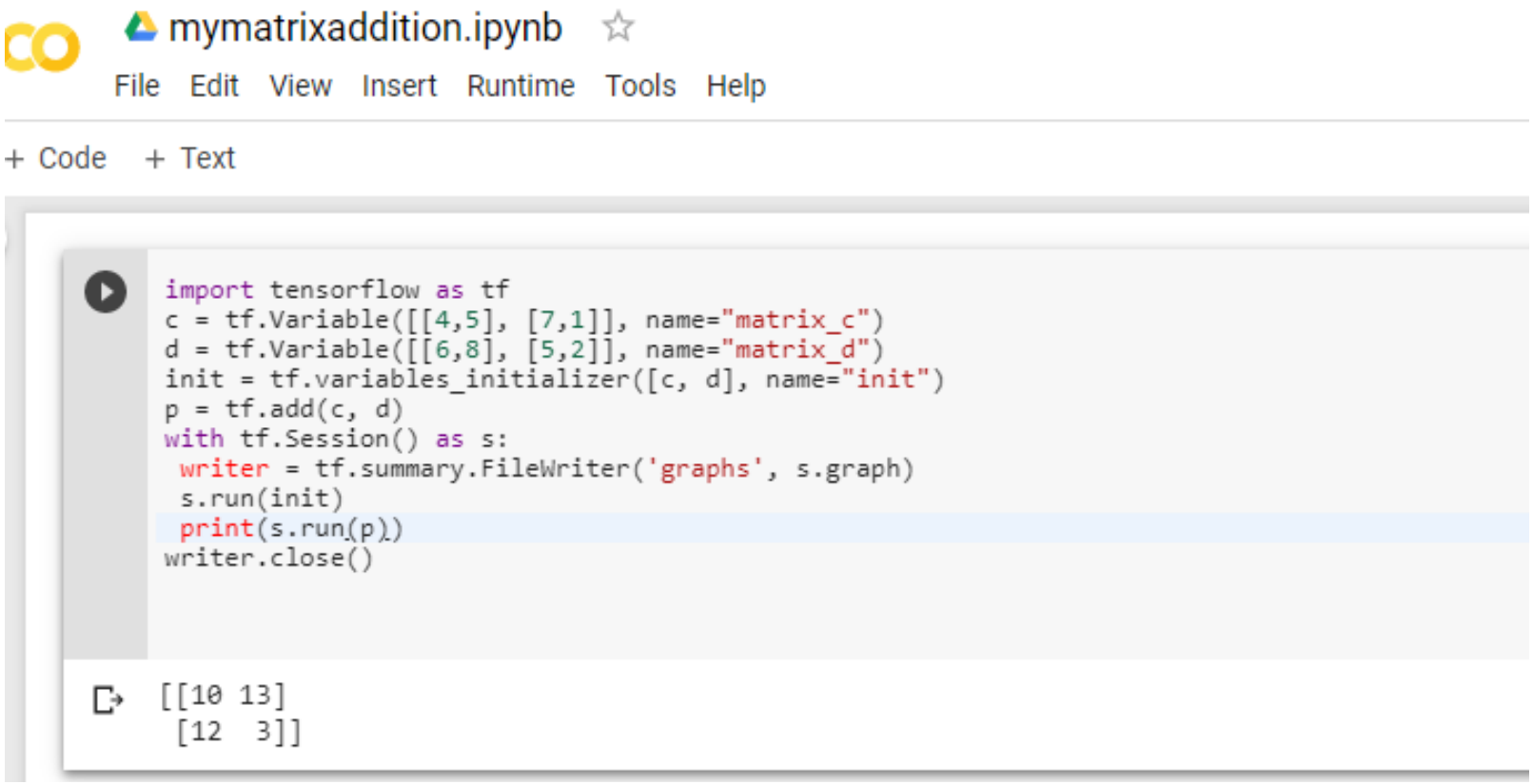
```
[[3 3 3]
 [3 3 3]
 [3 3 3]]
[[2 2 2]
 [2 2 2]
 [2 2 2]]
[[4. 5. 6.]
 [3. 4. 1.]
 [2. 0. 1.]]
[[18 18 18]
 [18 18 18]
 [18 18 18]]
[[5 5 5]
 [5 5 5]
 [5 5 5]]
-37.000004
```

Basic Code Examples

- **EXAMPLE 4 :** MATRIX ADDITION :

```
import tensorflow as tf

c = tf.Variable([[4,5], [7,1]], name="matrix_c")

d = tf.Variable([[6,8], [5,2]], name="matrix_d")

init = tf.variables_initializer([c, d], name="init")

p = tf.add(c, d)

with tf.Session() as s:

        writer = tf.summary.FileWriter('graphs', s.graph)

        s.run(init)

        print(s.run(p))

writer.close()
```

Basic Code Examples



Basic Code Examples

- More examples:

```
1  ## Tensor Types
2  import tensorflow as tf
3  import numpy as np
4
5  # Define a 2x2 matrix in 3 different ways
6  m1 = [[1.0, 2.0], [3.0, 4.0]]                                  # <class 'list'>
7  m2 = np.array([[1.0, 2.0], [3.0, 4.0]], dtype=np.float32) # <class 'np.ndarray'>
8  m3 = tf.constant([[1.0, 2.0], [3.0, 4.0]])                     # <class 'tensorflow'>
9  print(type(m1))
10 print(type(m2))
11 print(type(m3))
12
13 # Create tensor objects out of various types
14 t1 = tf.convert_to_tensor(m1, dtype=tf.float32)           # <class 'tensorflow'>
15 t2 = tf.convert_to_tensor(m2, dtype=tf.float32)           # <class 'tensorflow'>
16 t3 = tf.convert_to_tensor(m3, dtype=tf.float32)           # <class 'tensorflow'>
17 print(type(t1))
18 print(type(t2))
19 print(type(t3))
```

```
Python
In [9]: m3.op.name
Out[9]: 'Const'
```

... try it in class

```
Python
In [1]: (executing cell "Tensor Types" (line 2 of "types.py"))
<class 'list'>
<class 'numpy.ndarray'>
<class 'tensorflow.python.framework.ops.Tensor'>
<class 'tensorflow.python.framework.ops.Tensor'>
<class 'tensorflow.python.framework.ops.Tensor'>
<class 'tensorflow.python.framework.ops.Tensor'>
```

TensorFlow

- Before you can use a variable, it must be initialized
- Most high-level frameworks such as tf.contrib.slim, tf.estimator.Estimator and Keras automatically initialize variables
- If you are explicitly creating your own graphs and sessions, you must explicitly initialize the variables
- To initialize all trainable variables on one go, before training starts, call:

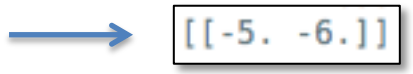
  session.run(tf.global_variables_initializer())
- To ask which variables have still not been initialized:

  session.run(my_variable.initializer)

TensorFlow

- More examples:

```
1  ## A simple matrix and the InteractiveSession:
2  import tensorflow as tf
3  sess = tf.InteractiveSession()
4
5  matrix = tf.constant([[5., 6.]])
6  negMatrix = tf.negative(matrix)
7
8  result = negMatrix.eval()
9  print(result)
10 sess.close()
```



- **tf.InteractiveSession**:
  - The **only difference** with a regular `Session` is that an `InteractiveSession` installs itself as the default session on construction.

  - For example:

```
sess = tf.InteractiveSession()
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b
# We can just use 'c.eval()' without passing 'sess'
print(c.eval())
sess.close()
```

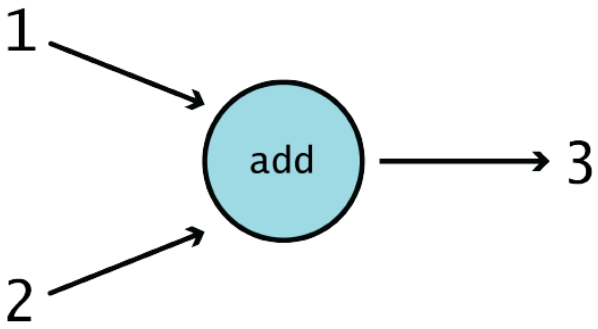

... try it in class

TensorFlow

- Let's discuss the basics of computation graphs without the context of TensorFlow
- This includes:
  - defining nodes
  - defining edges
  - dependencies
  - examples to illustrate key principles

TensorFlow

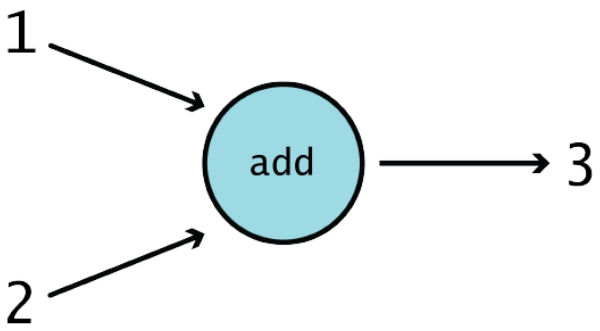
- Graph basics:
  - At the core of every TensorFlow program is the computation graph
  - It is a is a specific type of directed graph that is used for defining computational structure
  - In TensorFlow we have a series of functions chained together, each passing its output to zero, one, or more functions further along in the chain



$$f(1,2) = 1 + 2 = 3$$

TensorFlow

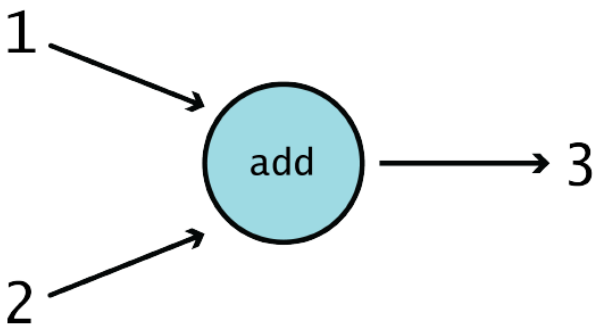
- Graph basics:
  - Nodes: typically drawn as circles, ovals, or boxes, represent some sort of computation or action being done on or with data in the graph's context. In the example below, the operation "add" is the sole node.



$$f(1,2) = 1 + 2 = 3$$

TensorFlow

- Graph basics:
  - Edges: are the actual values that get passed to and from *Operations*, and are typically drawn as arrows
  - In the "add" example, the inputs 1 and 2 are both edges leading into the node, while the output 3 is an edge leading out of the node
  - We can think of edges as the link between different *Operations* as they carry information from one node to the next



$$f(1,2) = 1 + 2 = 3$$

TensorFlow

- Instead of having two separate input nodes, we can have a single input node that can take in a vector (or 1-D tensor) of numbers
- This graph has several advantages over our previous example:

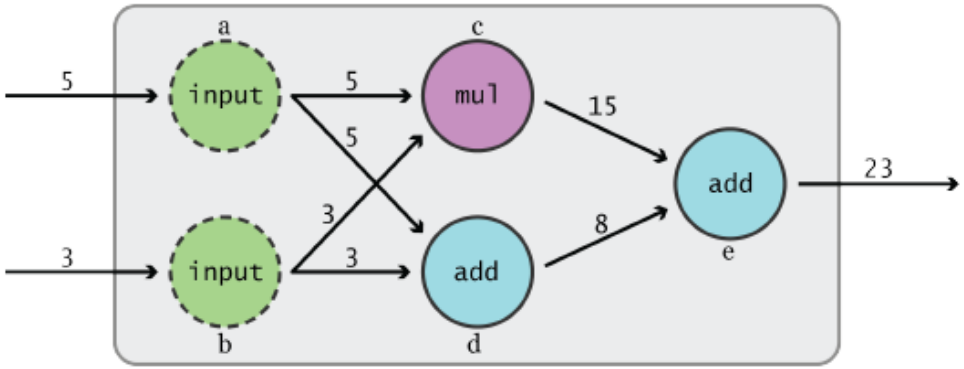
1. The client only have to send the input to a single node, which simplifies using the graph

2. The nodes that directly depend on the input now only have to keep track of one dependency instead of two

3. We now have the option to make the graph take in vectors of any length, if we'd like. This would make the graph more flexible

TensorFlow

- Building our first graph in TensorFlow:



simple_graph.py

```
1  ## Building our first TensorFlow graph:
2
3  # First we need to import TensorFlow:
4  import tensorflow as tf
5
6  # Let's define our input nodes:
7  a = tf.constant(5, name="input_a")
8  b = tf.constant(3, name="input_b")
9
10 # Defining the next two nodes in our graph:
11 c = tf.multiply(a,b, name="mul_c")
12 d = tf.add(a,b, name="add_d")
13
14 # This last line defines the final node in our graph:
15 e = tf.add(c,d, name="add_e")
```

TensorFlow

- Building our first graph in TensorFlow:

Python /Users/alex

```
In [6]: whos
Variable   Type      Data/Info
------------------------------
a          Tensor    Tensor("input_a_1:0", shape=(), dtype=int32)
b          Tensor    Tensor("input_b_1:0", shape=(), dtype=int32)
c          Tensor    Tensor("mul_c:0", shape=(), dtype=int32)
d          Tensor    Tensor("add_d:0", shape=(), dtype=int32)
e          Tensor    Tensor("add_e:0", shape=(), dtype=int32)
tf         module    <module 'tensorflow' from<...>tensorflow/__init__.pyc'>
```

simple_graph.py

```
1  ## Building our first TensorFlow graph:
2
3  # First we need to import TensorFlow:
4  import tensorflow as tf
5
6  # Let's define our input nodes:
7  a = tf.constant(5, name="input_a")
8  b = tf.constant(3, name="input_b")
9
10 # Defining the next two nodes in our graph:
11 c = tf.multiply(a,b, name="mul_c")
12 d = tf.add(a,b, name="add_d")
13
14 # This last line defines the final node in our graph:
15 e = tf.add(c,d, name="add_e")
```

To run we have to add the two extra lines and run them in the shell:

```
In [7]: sess = tf.Session()

In [8]: sess.run(e)
Out[8]: 23
```

Basic operations

1)

```python
import tensorflow as tf

# Basic constant operations
# The value returned by the constructor represents the output
# of the Constant op.
a = tf.constant(2)
b = tf.constant(3)

# Launch the default graph.
with tf.Session() as sess:
    print("a=2, b=3")
    print("Addition with constants: %i" % sess.run(a+b))
    print("Multiplication with constants: %i" % sess.run(a*b))
```

2)

```python
# Basic Operations with variable as graph input
# The value returned by the constructor represents the output
# of the Variable op. (define as input when running session)
# tf Graph input
a = tf.placeholder(tf.int16)
b = tf.placeholder(tf.int16)

# Define some operations
add = tf.add(a, b)
mul = tf.multiply(a, b)

# Launch the default graph.
with tf.Session() as sess:
    # Run every operation with variable input
    print("Addition with variables: %i" % sess.run(add, feed_dict={a: 2, b: 3}))
    print("Multiplication with variables: %i" % sess.run(mul, feed_dict={a: 2, b: 3}))
```

Basic operations

```
39    import tensorflow as tf
40    # Create a graph that produces a 1x2 matrix.
41    #
42    # The value returned by the constructor represents the output
43    # of the Constant op.
44    matrix1 = tf.constant([[3., 3.]])
45
46    # Create another Constant that produces a 2x1 matrix.
47    matrix2 = tf.constant([[2.],[2.]])
48
49    # Create a matmul op that takes 'matrix1' and 'matrix2' as inputs.
50    # The returned value is their product:
51    product = tf.matmul(matrix1, matrix2)
```

The output of the op is returned in 'result' as a numpy `ndarray` object:

Basic operations

```
39  import tensorflow as tf
40  # Create a graph that produces a 1x2 matrix.
41  #
42  # The value returned by the constructor represents the output
43  # of the Constant op.
44  matrix1 = tf.constant([[3., 3.]])
45
46  # Create another Constant that produces a 2x1 matrix.
47  matrix2 = tf.constant([[2.],[2.]])
48
49  # Create a matmul op that takes 'matrix1' and 'matrix2' as inputs.
50  # The returned value is their product:
51  product = tf.matmul(matrix1, matrix2)
```

The output of the op is returned in 'result' as a numpy `ndarray` object:

```
64  with tf.Session() as sess:
65      result = sess.run(product)
66      print(result)
67      # ==> [[ 12.]]
```

Basic operations

Class exercise 1/2:

Create a graph that produces a 3x4 matrix, where you:

- *Add nodes A and B as matrices with some floating-point constants*
- *Add an operation of their product to the graph producing an output matrix of shape [3,4]*
- *Print the resulting matrix on the screen*

Basic operations

Class exercise solution 1/2:

```python
import tensorflow as tf

# ----------------
# Create a graph that produces a 3x4 matrix, where you:
#  - Add nodes A and B as matrixes with some floating-point constants
#  - Add an operation of their product to the graph producing an output matrix
#    of shape [3,4]
#  - Print the resulting matrix on the screen

# Create two constant matrices so that their product produces a 3x4 matrix:
matrix1 = tf.constant([[3., 3.],[4,5],[6,8]]) #3x2
matrix2 = tf.constant(([[2.,6,9,2.],[8,2,2,4]])) #2x4

# Create a matmul op that takes 'matrix1' and 'matrix2' as inputs.
# The returned value is their product:
product = tf.matmul(matrix1, matrix2)

# The output of the op is returned in 'result' as a numpy `ndarray` object.
with tf.Session() as sess:
    result = sess.run(product) #3x4
    print(result)

# Result:
# [[30. 24. 33. 18.]
# [48. 34. 46. 28.]
# [76. 52. 70. 44.]]
```