ECE521 W17 Tutorial 1

Renjie Liao & Min Bai

UNIVERSITY OF TORONTO

Schedule

- Linear Algebra Review
    - Matrices, vectors
    - Basic operations
- Introduction to TensorFlow
    - NumPy
    - Computational Graphs
    - Basic Examples

Linear Algebra Review

Vector: 1D array,

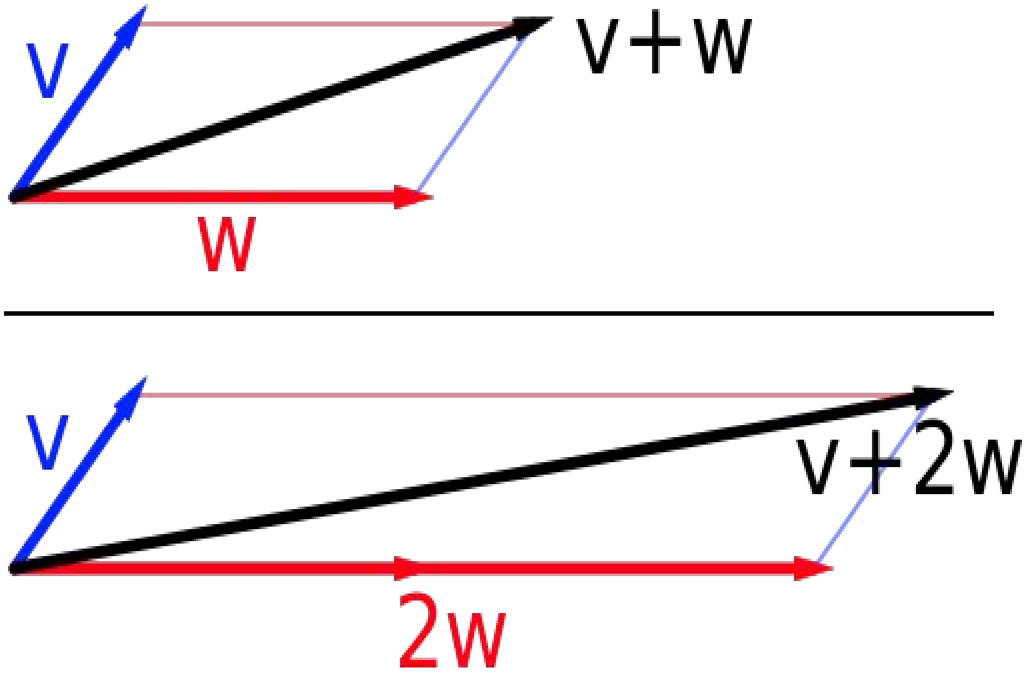
row, column vectors

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Matrix: 2D array

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} = (a_{ij}) \in \mathbb{R}^{m \times n}.$$

Vector Addition, Scalar Multiplication



Vector Inner Product

$$\left\langle \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\rangle := x^{\mathrm{T}} y = \sum_{i=1}^{n} x_i y_i = x_1 y_1 + \cdots + x_n y_n,$$

Properties:

Conjugate Symmetry $\langle x, y \rangle = \overline{\langle y, x \rangle}$

Linearity

$$\langle ax, y \rangle = a\langle x, y \rangle$$
$$\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$$

Positive-Definiteness

$$\langle x, x \rangle \geq 0$$
$$\langle x, x \rangle = 0 \Leftrightarrow x = \mathbf{0}.$$

Vector Outer Product

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^{\mathrm{T}} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & u_1 v_3 \\ u_2 v_1 & u_2 v_2 & u_2 v_3 \\ u_3 v_1 & u_3 v_2 & u_3 v_3 \\ u_4 v_1 & u_4 v_2 & u_4 v_3 \end{bmatrix}.$$

$$(\mathbf{u}\mathbf{v}^{\mathrm{T}})_{ij} = u_i v_j$$

Vector Norms

Lp Norm:

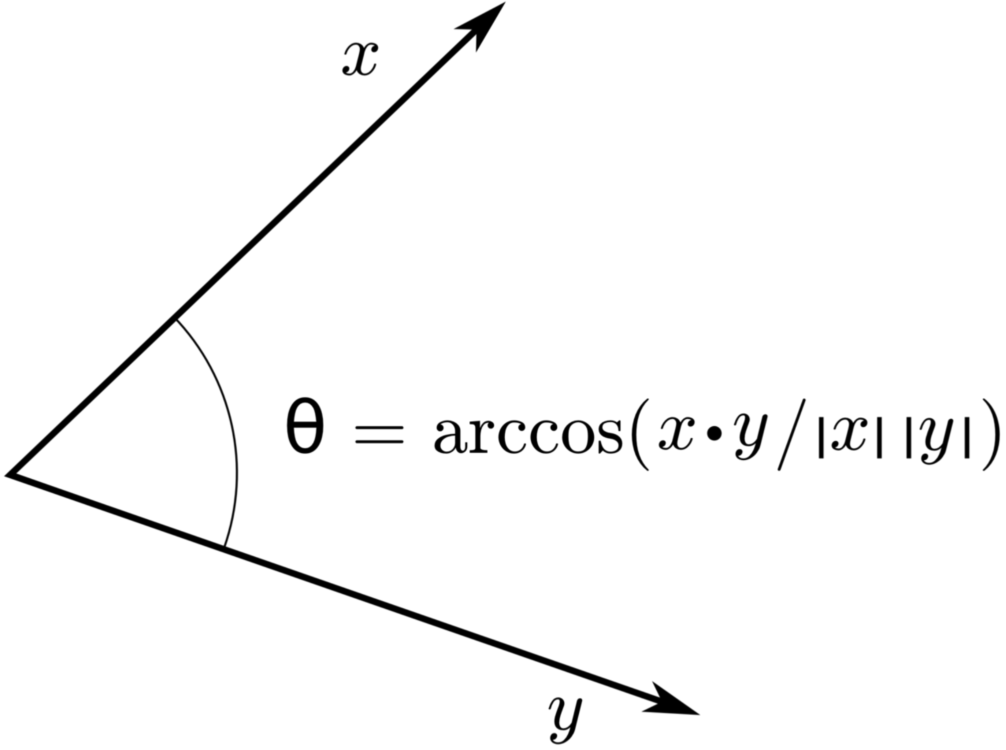
$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}. \qquad p \geq 1$$

Example:

For p = 1: Manhattan norm

For p = 2: Euclidean norm

Geometric Interpretation

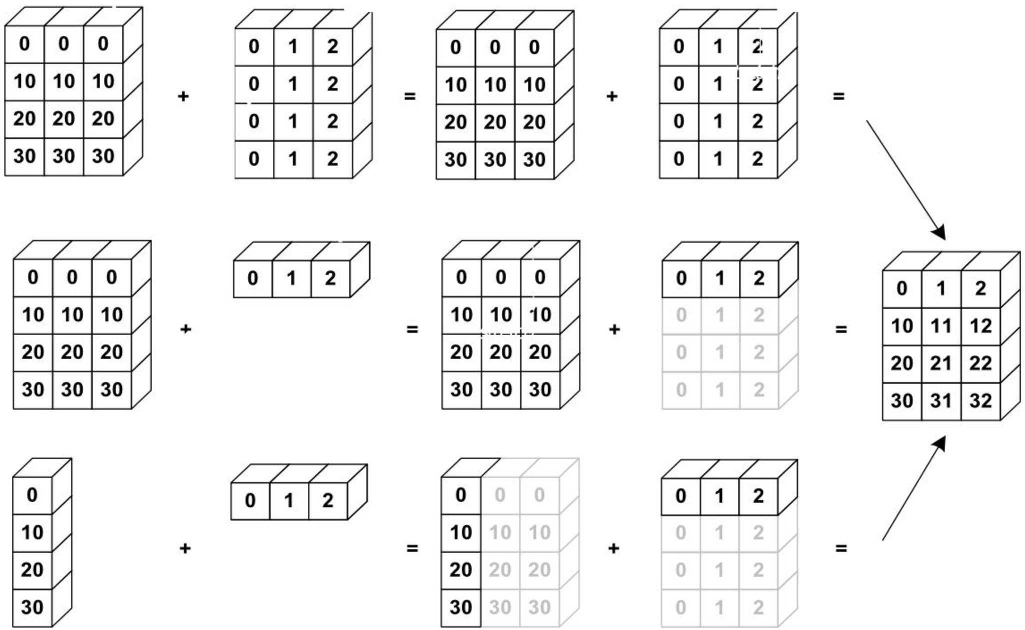


Matrix Addition, Scalar Multiplication

$$\begin{bmatrix} 0 & 1 & 2 \\ 9 & 8 & 7 \end{bmatrix} + \begin{bmatrix} 6 & 5 & 4 \\ 3 & 4 & 5 \end{bmatrix} = \begin{bmatrix} 0+6 & 1+5 & 2+4 \\ 9+3 & 8+4 & 7+5 \end{bmatrix} = \begin{bmatrix} 6 & 6 & 6 \\ 12 & 12 & 12 \end{bmatrix}$$

$$\lambda\mathbf{A} = \lambda \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{pmatrix} = \begin{pmatrix} \lambda A_{11} & \lambda A_{12} & \cdots & \lambda A_{1m} \\ \lambda A_{21} & \lambda A_{22} & \cdots & \lambda A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda A_{n1} & \lambda A_{n2} & \cdots & \lambda A_{nm} \end{pmatrix}.$$

Broadcasting



Matrix Multiplication

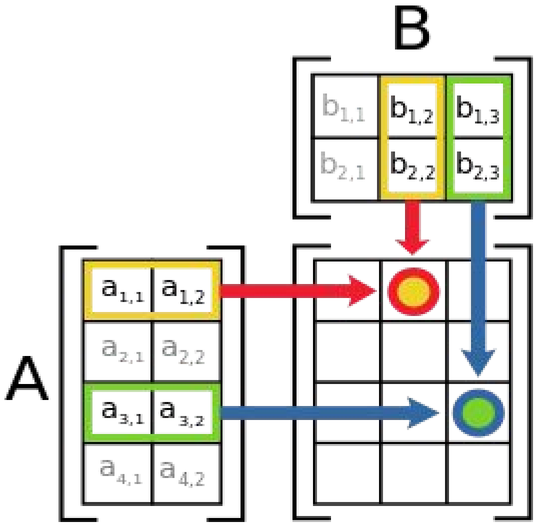
$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1p} \\ B_{21} & B_{22} & \cdots & B_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mp} \end{pmatrix}$$

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj} \,.$$

Matrix Multiplication

$$\overset{4\times 2 \text{ matrix}}{\begin{bmatrix} a_{11} & a_{12} \\ \cdot & \cdot \\ a_{31} & a_{32} \\ \cdot & \cdot \end{bmatrix}} \overset{2\times 3 \text{ matrix}}{\begin{bmatrix} \cdot & b_{12} & b_{13} \\ \cdot & b_{22} & b_{23} \end{bmatrix}} = \overset{4\times 3 \text{ matrix}}{\begin{bmatrix} \cdot & x_{12} & x_{13} \\ \cdot & \cdot & \cdot \\ \cdot & x_{32} & x_{33} \\ \cdot & \cdot & \cdot \end{bmatrix}}$$

$$x_{12} = a_{11}b_{12} + a_{12}b_{22}$$
$$x_{33} = a_{31}b_{13} + a_{32}b_{23}$$



Matrix Multiplication

Properties:

Not Commutative (In general)

$$\mathbf{AB} \neq \mathbf{BA}$$

Distributive

$$\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$$

Transpose

$$(\mathbf{AB})^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}$$

Matrix Vector Multiplication

Row/Column views

$$
\begin{aligned}
A\mathbf{x} &= \begin{bmatrix} 1 & -1 & 2 \\ 0 & -3 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} 2 \cdot 1 - 1 \cdot 1 + 0 \cdot 2 \\ 2 \cdot 0 - 1 \cdot 3 + 0 \cdot 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 \\ -3 \end{bmatrix}.
\end{aligned}
$$

1D Convolution

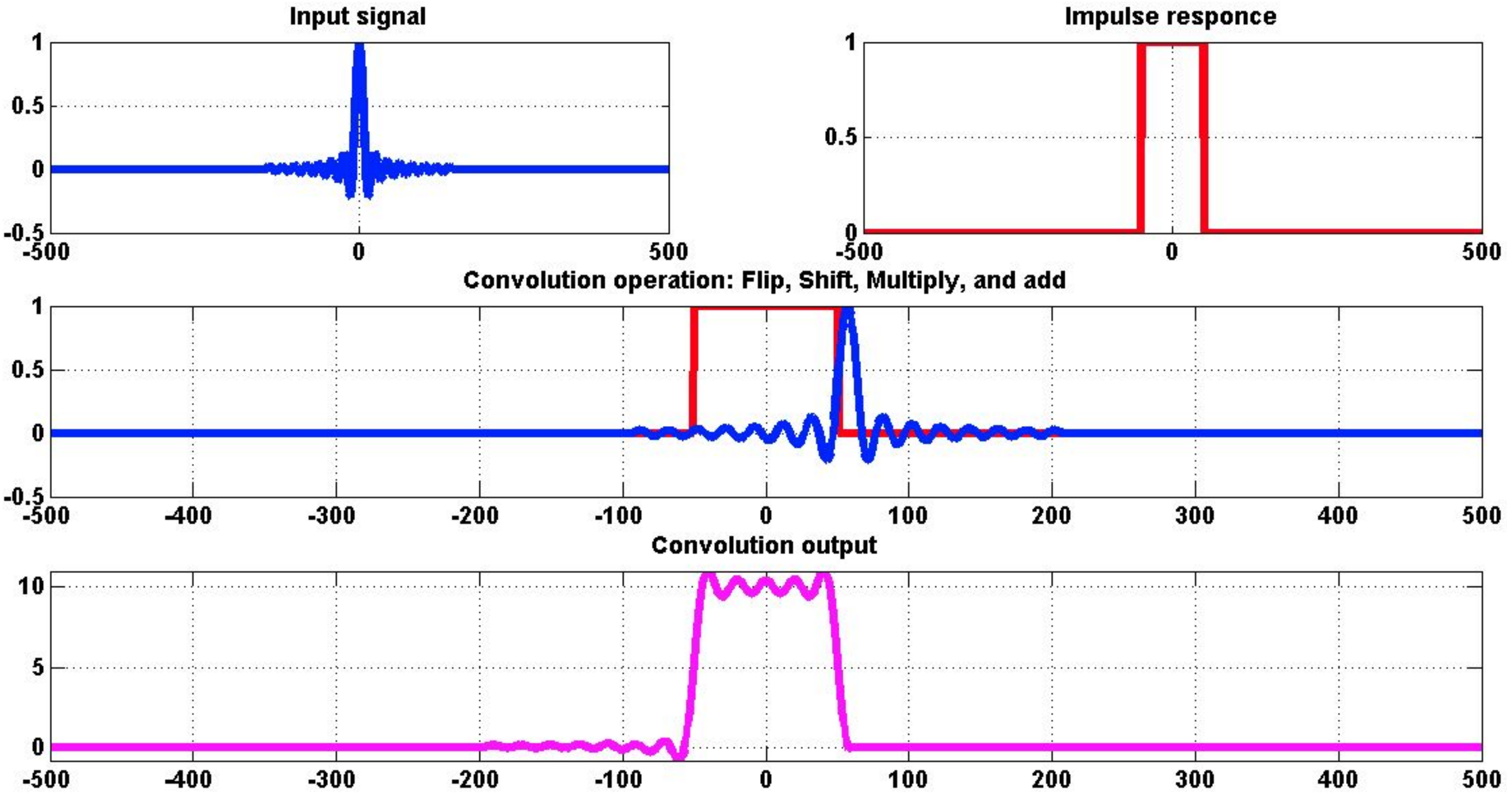
Discrete Definition:

$$(f * g)[n] = \sum_{m=-M}^{M} f[n-m]g[m].$$

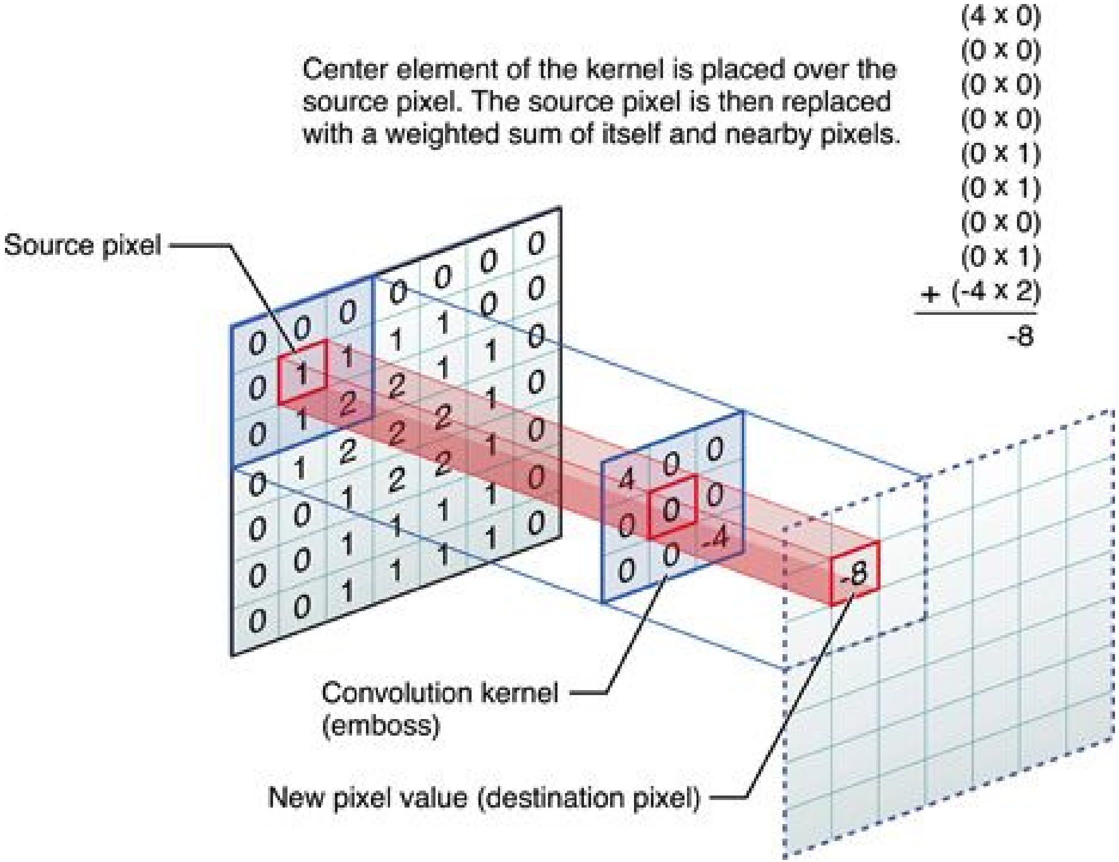
Continuous Definition

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$
$$= \int_{-\infty}^{\infty} f(t-\tau)\, g(\tau)\, d\tau.$$

1D Visualization



2D Convolution



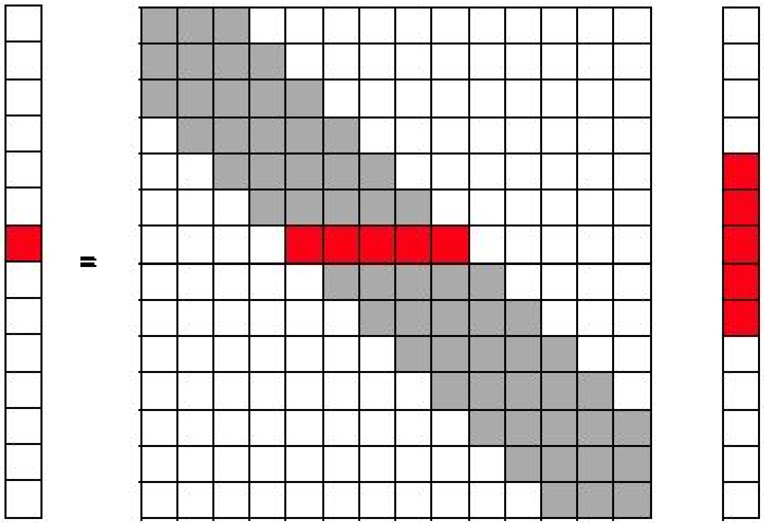
Convolution as Matrix Multiplication

$$y = h * x = \begin{bmatrix} h_1 & 0 & \dots & 0 & 0 \\ h_2 & h_1 & \dots & \vdots & \vdots \\ h_3 & h_2 & \dots & 0 & 0 \\ \vdots & h_3 & \dots & h_1 & 0 \\ h_{m-1} & \vdots & \dots & h_2 & h_1 \\ h_m & h_{m-1} & \vdots & \vdots & h_2 \\ 0 & h_m & \dots & h_{m-2} & \vdots \\ 0 & 0 & \dots & h_{m-1} & h_{m-2} \\ \vdots & \vdots & \vdots & h_m & h_{m-1} \\ 0 & 0 & 0 & \dots & h_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Convolution as Matrix Multiplication

$$y^T = [\,h_1 \quad h_2 \quad h_3 \quad \dots \quad h_{m-1} \quad h_m\,] \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n & 0 & 0 & 0 & \dots & 0 \\ 0 & x_1 & x_2 & x_3 & \dots & x_n & 0 & 0 & \dots & 0 \\ 0 & 0 & x_1 & x_2 & x_3 & \dots & x_n & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & 0 \\ 0 & \dots & 0 & 0 & x_1 & \dots & x_{n-2} & x_{n-1} & x_n & \vdots \\ 0 & \dots & 0 & 0 & 0 & x_1 & \dots & x_{n-2} & x_{n-1} & x_n \end{bmatrix}.$$

Convolution as Matrix Multiplication



Introduction to TensorFlow

What is it?

- Open source library by Google for machine learning
- Supports CPU / GPU / mix / distributed computing
- Linux, Mac OS X, Windows versions

How to get it?

- https://www.tensorflow.org/get_started/os_setup
- https://github.com/tensorflow/tensorflow

TensorFlow

NumPy

- Important python package, core component of TensorFlow
- Similar supports linear algebra and numerical operations similar to MATLAB
- Part of larger SciPy package which has even more numerical operations

```
In [1]: import numpy as np

In [2]: a = np.array([1,2,3,4])

In [3]: b = np.array([5,6,7,8])

In [4]: a*b
Out[4]: array([ 5, 12, 21, 32])

In [5]: a.shape
Out[5]: (4,)

In [6]: a.sum()
Out[6]: 10

In [7]: 
```

```
In [12]: c = np.array([[1,2,3,4,5],[6,7,8,9,10]])

In [13]: c
Out[13]:
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])

In [14]: c[0:2, 3:5]
Out[14]:
array([[ 4,  5],
       [ 9, 10]])

In [15]: c[-1, -1]
Out[15]: 10

In [16]: c.dtype
Out[16]: dtype('int64')
```

NumPy and TensorFlow Comparisons

Numpy	TensorFlow
`a = np.zeros((2,2)); b = np.ones((2,2))`	`a = tf.zeros((2,2)), b = tf.ones((2,2))`
`np.sum(b, axis=1)`	`tf.reduce_sum(a,reduction_indices=[1])`
`a.shape`	`a.get_shape()`
`np.reshape(a, (1,4))`	`tf.reshape(a, (1,4))`
`b * 5 + 1`	`b * 5 + 1`
`np.dot(a,b)`	`tf.matmul(a, b)`
`a[0,0], a[:,0], a[0,:]`	`a[0,0], a[:,0], a[0,:]`

Credit: Stanford CS224 Lecture Notes

TensorFlow Overview

- Unlike NumPy, TensorFlow does NOT perform operations immediately
  - Need to explicitly “run” operation
- Represents data as multidimensional arrays, or tensors
- Basic procedure:
  - **Construct** computational graph
  - Create an operations **session**
  - **Initialize** parameters of computational graph, if any (eg initial model weights)
  - **Feed** data to graph (eg input image matrix)
  - **Execute** computation operations which return data (eg output label for object in image)

TensorFlow - Basic Example 1

- **An example of the element-wise product of two vectors:**

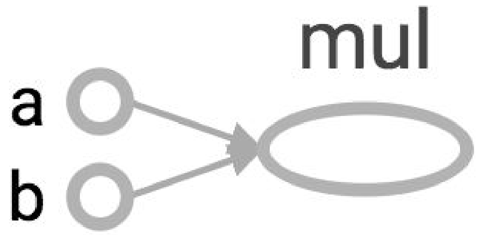
$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix} \qquad \mathbf{c} = \mathbf{a} \odot \mathbf{b} = \begin{bmatrix} a_1 b_1 \\ a_2 b_2 \\ a_3 b_3 \\ a_4 b_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 12 \\ 21 \\ 32 \end{bmatrix}$$

TensorFlow - Basic Example 1

- **Construct** computational graph
  - Note that c does NOT actually have a value
  - The graph defines operational relationships between nodes
  - Note that in this case, c has inferred the shape to be (4,) - but a node can have None as dimensions if unknown at construction time

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

In [2]: a = tf.constant([1,2,3,4])

In [3]: b = tf.constant([5,6,7,8])

In [4]: c = a*b

In [5]: c
Out[5]: <tf.Tensor 'mul:0' shape=(4,) dtype=int32>
```

the resulting computation graph



TensorFlow - Basic Example 1

- Create operations **session:**

```
In [8]: sess = tf.Session()
```

or

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

- Session defines an environment for the execution of operations
  - The session keeps track of a set of actual values for variable nodes
- Can have multiple sessions to run copies of graphs in parallel
- tf.InteractiveSession() often used in iPython to set current session as default

TensorFlow - Basic Example 1

- **Initialize** parameters in graph:

```
In [6]: init = tf.initialize_all_variables()

In [7]: sess.run(init)
```

- **Execute** computation graph:
  - Returns values for each node queried

```
In [8]: sess.run([a,b,c])
Out[8]:
[array([1, 2, 3, 4], dtype=int32),
 array([5, 6, 7, 8], dtype=int32),
 array([ 5, 12, 21, 32], dtype=int32)]
```

TensorFlow - Basic Example 1

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

In [2]: a = tf.constant([1,2,3,4])

In [3]: b = tf.constant([5,6,7,8])

In [4]: c = a*b

In [5]: sess = tf.InteractiveSession()

In [6]: init = tf.initialize_all_variables()

In [7]: sess.run(init)

In [8]: sess.run([a,b,c])
Out[8]:
[array([1, 2, 3, 4], dtype=int32),
 array([5, 6, 7, 8], dtype=int32),
 array([ 5, 12, 21, 32], dtype=int32)]
```

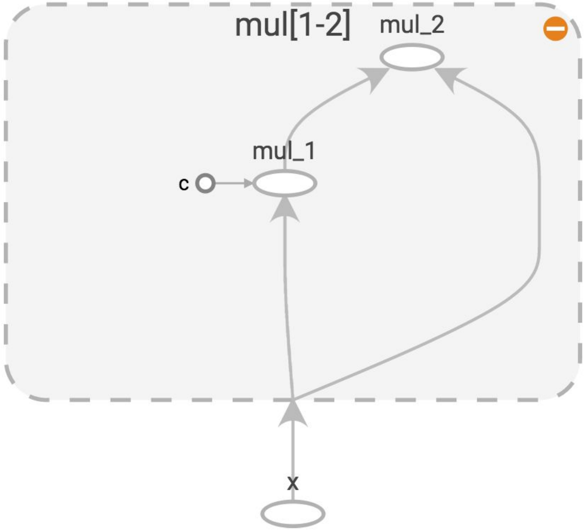
TensorFlow - Basic Example 2

- Example 1 does not have a **feeding** mechanism
  - All parameters in the graph were available in advance
- Suppose we want to:
  - Define the graph
  - Initialize nodes that correspond to model parameters - ONLY ONCE!
  - Provide training data to some nodes and update model parameters via gradient descent
- We use the tf.placeholder()
  - Arguments: data type (mandatory), shape (optional), name (optional)
  - Specifying (the entirety or part of) shape *restricts* shapes of possible inputs to the node
  - At run time, we feed using a feed_dict

```
In [4]: x = tf.placeholder(tf.float32, [2, None])
```

TensorFlow - Basic Example 2

```
In [2]: import numpy as np

In [3]: x = tf.placeholder(tf.float32)

In [4]: c = tf.constant(2.0)

In [5]: cx_squared = c*x*x

In [6]: sess = tf.InteractiveSession()

In [7]: init = tf.initialize_all_variables()

In [8]: sess.run(init)

In [9]: sess.run(cx_squared, feed_dict={x:np.array([1,2,3])})
Out[9]: array([  2.,   8.,  18.], dtype=float32)

In [10]: sess.run(cx_squared, feed_dict={x:np.array([[4,5],[6,7]])})
Out[10]:
array([[ 32.,  50.],
       [ 72.,  98.]], dtype=float32)
```



TensorFlow - Basic Example 3

- The most important operation: optimization
- Define a loss function
- Select an optimizer
  - https://www.tensorflow.org/api_docs/python/train/
- Define optimization operation using selected optimizer
- sess.run(optimization operation)

```
elementSquaredError = tf.square(y_predicted-y_target)
meanSquaredError = tf.reduce_mean(tf.reduce_sum(elementSquaredError, reduction_indices=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.005)
train = optimizer.minimize(loss=meanSquaredError)
sess.run(train)
```

TensorFlow - Basic Example 3

- Example:
    - $\mathbf{X}$ is a 100 by 5 matrix (100 sets of 5-dimensional input data points)
    - $\mathbf{Y}$ is a 1 by 100 target vector (100 target labels for each x)
    - W is a 5 by 1 weights vector
    - b is a scalar
    - Model: $\hat{\mathbf{y}} = W^T\mathbf{x} + b = \sum_{i=1}^{5} W_i x_i + b$
    - Given X and Y,
    - optimize for W and b under mean squared error over the 70 training examples:

$$\min_{W,b} \quad \frac{1}{70}\sum_{m=1}^{70} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2$$

- Example code at: http://www.psi.toronto.edu/~jimmy/ece521/mult.py
- Also download the two dataset files
  http://www.psi.toronto.edu/~jimmy/ece521/x.npy
  http://www.psi.toronto.edu/~jimmy/ece521/t2.npy

TensorFlow - Basic Example 3

- Automatic differentiation in TensorFlow can generate gradient of a computation graph with respect to the variables automatically.