Linear Algebra and Optimization in Python

Razvan C. Bunescu
School of Electrical Engineering and Computer Science
bunescu@ohio.edu
Python Programming Stack

• **Python** = object-oriented, interpreted, scripting language.
  – imperative programming, with functional programming features.

• **NumPy** = package for powerful N-dimensional arrays:
  – sophisticated (broadcasting) functions.
  – useful linear algebra, Fourier transform, and random number capabilities.

• **SciPy** = package for numerical integration and optimization.

• **Matplotlib** = comprehensive 2D plotting library.
import numpy as np

- np.array()
  - indexing, slices.
- ndarray.shape, .size, .ndim, .dtype, .T
- np.zeros(), np.ones(), np.arange(), np.eye()
  - dtype parameter.
  - tuple (shape) parameter.
- np.reshape(), np.ravel()
  - also np.ndarray.
- np.amax(), np.maximum(), np.sum(), np.mean(), np.std()
  - axis parameter, also np.ndarray
- np.stack(), np.[hv]stack(), np.column_stack(), np.split()
- np.exp(), np.log(),
NumPy: Broadcasting

- Broadcasting describes how numpy treats arrays with different shapes during arithmetic operations.
- The smaller array is “broadcast” across the larger array so that they have compatible shapes, subject to broadcasting rules:
  - NumPy compares their shapes element-wise.
  - It starts with the trailing dimensions, and works its way forward.
  - Two dimensions are compatible when:
    - they are equal, or one of them is 1.

- [https://docs.scipy.org/doc/numpy-dev/user/basics.broadcasting.html](https://docs.scipy.org/doc/numpy-dev/user/basics.broadcasting.html)

Lecture 03
Other Numpy Functions

- np.dot(), np.vdot()
  - also np.ndarray.
- np.outer(), np.inner()

- import numpy.random as random:
  - randn(), randint(), uniform()

- import numpy.linalg as la:
  - la.norm(), la.det(), la.matrix_rank(), np.trace()
  - la.eig(), la.svd()
  - la.qr(), la.cholesky()

- [https://docs.scipy.org/doc/numpy/reference/routines.linalg.html](https://docs.scipy.org/doc/numpy/reference/routines.linalg.html)
Implementation: Vectorization

- **Version 1**: Compute gradient component-wise.

$$\nabla E(w) = \sum_{n=1}^{N} (h_n - t_n) x_n^T$$

```python
grad = np.zeros(K)
for n in range(N):
    h = sigmoid(w.dot(X[:, n]))
    temp = h - t[n]
    for k in range(K):
        grad(k) = grad(k) + temp * X[k,n]
```
Implementation: Vectorization

- **Version 2**: Compute gradient, partially vectorized.

$$\nabla E(w) = \sum_{n=1}^{N} (h_n - t_n)x_n^T$$

```python
grad = np.zeros(K)
for n in range(N):
    grad = grad + (sigmoid(w.dot(X[:, n])) - t[n]) * X[:, n]
```
Implementation: Vectorization

- **Version 3**: Compute gradient, vectorized.

\[
\nabla E(w) = \sum_{n=1}^{N} (h_n - t_n)x_n^T
\]

\[
\text{grad} = X \cdot \text{dot}(\text{sigmoid}(w \cdot \text{dot}(X)) - t)
\]

```python
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

---

Lecture 03
import scipy

- scipy.sparse.coo_matrix()
  
  groundTruth = coo_matrix((np.ones(numCases, dtype = np.uint8),
                            (labels, np.arange(numCases)))).toarray()

- scipy.optimize:
  - scipy.optimize.fmin_l_bfgs_b()
    
    theta, _, _ = fmin_l_bfgs_b(softmaxCost, theta,
                                args = (numClasses, inputSize, decay, images, labels),
                                maxiter = 100, disp = 1)
  - scipy.optimize.fmin_cg()
  - scipy.minimize

https://docs.scipy.org/doc/scipy-0.10.1/reference/tutorial/optimize.html