People doing DL on campus:

* Redwood Neuroscience (<http://redwood.berkeley.edu/>)
* Pieter Abbeel (https://www2.eecs.berkeley.edu/Faculty/Homepages/abbeel.html)

Resources:

* Parellel Distributed Processing (<https://mitpress.mit.edu/books/parallel-distributed-processing>)
* Geoff Hinton’s youtube NN course (https://www.youtube.com/watch?v=cbeTc-Urqak)

Kaggle competitions:

* <https://www.kaggle.com/c/invasive-species-monitoring> (invasive species - end of July?)
* <https://www.kaggle.com/c/zillow-prize-1> (Zillow $1,200,000 prize)

[www.deeplearningbook.org](http://www.deeplearningbook.org)

Deep Learning by Goodfellow, Bengio, Courville (2016). MIT Press.

**Chapter 1**

* DL stuff has varied in popularity. Now is incredibly popular again!
* Lots of great people on UC Berkeley campus
  + What breakthroughs made it popular? Backpropagation, dataset sizes, hardware/GPUs, more images (better curated), rectified linear units (easier to train), drop-out (Abbel et al.).
    - Hand-coded features to learned features was a big deal.
    - Features in DL can be continuous - human-recognized linear relationships are not necessary (as opposed to human coded).
  + DL is a hierarchy of concepts where each concept can be explained in terms of simpler concepts.
* Pros of DL:
  + Flexible application!
  + ConvNet offers similar accuracy to human representation even though we don’t know a lot about the brain?
  + features learned in different layers of convnets relate to features represented at different layers in the visual system of the brain. (Dicarlo 2013)
* Cons of DL:
  + Biases are not transparent
* Factors of variation:
  + From quantum mechanics: The state of an atom, molecule, etc. is represented as a vector. When states interact, it is represented mathematically by an outer product. So, vectors [x1, x2] and [y1, y1] generate the outer product  
    [x1^2 x2y1]  
    [x1y2 y2^2]  
    The off-diagonal terms are correlations between the states. When they are large, the states are highly correlated. When they are near zero, the states are not correlated. The matrix is what we can observe in the real world. We have to factor it out to get the states.
  + Linear separability: linear transforms important for transition to multilayer?
* Vector vs. tensor:
  + Vector is a subset of tensor (a 1D tensor)
  + Matrix is 2D tensor
  + Tensor is an n-dimensional vector
    - Images might have RGB color scheme
    - Tensor for R, tensor for G, tensor for B
    - Thus, you need 3 values to describe a single tensor element
* Figure 1.2
  + Nice illustration of how features map to neurons (and are learned?)
  + Input neurons take in raw data values (e.g. RGB values).
  + Next layer takes in combinations (a function) of lower layer.
  + If far right feature in first hidden layer (vertical edge detector) is activated, the others may fire?
  + Combine them into the next hidden layer, look for circles
  + Neurons in the third hidden layer respond strongly to object parts (such as a person’s face), hopefully invariant to their locations.
    - How many hidden layers are needed? No correct answer here, at the moment…
* Figure 1.3
  + What measure of abstraction are you using to examine complexity of your model?
  + Left hand side: Logistic regression broken down into its component operations
  + Right hand side: Logistic regression as a unit
  + [Theano’s Computational Graph](http://deeplearning.net/software/theano/extending/graphstructures.html):
* **We will revisit Chapter 1 on June 7**

**Chapter 2**

* This chapter defines how to solve stuff when you have a perfect matrix
  + Rote memorization. Memorize this stuff and bring your questions!
* Review terminology on page 31-34
* Equation 2.11 A = design matrix, x=betas, b=Y
  + In linear algebra, do the opposite action to move it around.
  + Here, that opposite action is the inverse matrix
    - This is often untrue for regression (pseudo-inverse)
    - (see Equation 2.25)
* Matrix multiplication is non-communative (vector inner products v. outer product)
  + Inner = scalar
  + Outer = matrix
* Symmetric matrix: matrix is equal to its transpose
  + Square diagonal matrices are always symmetric!
* However, non-square diagonal matrices to exist?
* Linear dependence and span
  + A matrix with more columns than rows (wider than it is tall) often leads to linear dependence (why ridge regression was developed!)
  + If you have fewer data points than regressors, then you don’t have enough info to define regressors, so that the leftover regressors are unknown.
    - Span = all possible combinations of a number of vectors (all the points in space you can get to by taking linear combinations of the vectors)
    - Linear independence:
      * When one solution exists for b in Equation 2.11
      * You cannot take a combination of vectors to get another bector
      * Inverse of A will exist
    - Linear dependence:
      * Imagine you have three vectors. If sum of vectors 1 and 2 is equal to another vector, you have linear dependences (duplication in efficiency) (or if you can take a linear combination of vectors 1 and 2 to get vector 3. “Linear dependence” = e.g. 2\*vector 1 + 5\*vector 2).
        + Or when no/infinite solutions exist for b in Equation 2.11
        + Too many regressors for solving inverse of A
        + Can’t span the vector space!
    - Video: (<https://www.khanacademy.org/math/linear-algebra/vectors-and-spaces/linear-independence/v/linear-algebra-introduction-to-linear-independence>)
* Norms
  + Ways to define lengths!
* Special kinds of matrices and vectors
  + Diagonal, symmetric
  + Orthogonal = matrix where columns are orthogonal and orthonormal (normalized to length of one) vectors
* Eigendecomposition
  + A single polynomial equation corresponds to **A** being a vector. A polynomial can be factored into simpler polynomials: x^2 + 3x + 2 = (x + 2)(x + 1) = 0. Solving the simpler polynomials can be used to find the roots of the polynomial. The roots here are x = -2 and x = -1. Eigendecomposition is basically the same idea for higher dimensions.
  + Different uses of representing a matrix as its eigendecomposition: makes raising a matrix to a power computationally easier.
  + Eigenvectors and Eigenvalues: imagine a vector in 2D space:
    - Eigenvector = direction of vector
    - Eigenvalue = its magnitude

**Chapter 3**

* Seeing Theory: A visual introduction to probability and statistics (<http://students.brown.edu/seeing-theory/index.html>)