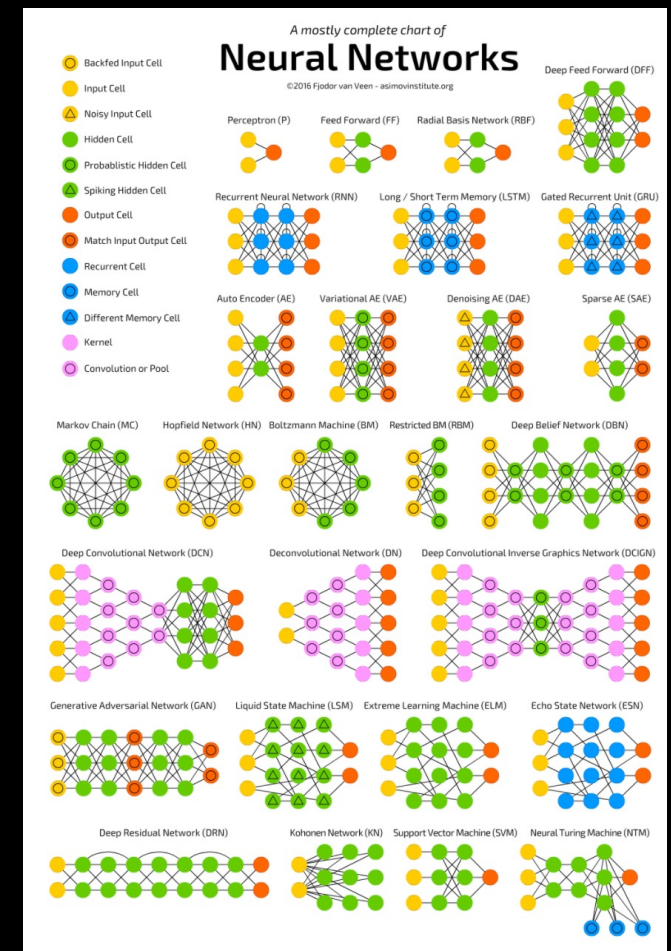


Autoencoders

CSCI 4360/6360 Data Science II

The Neural Network Zoo

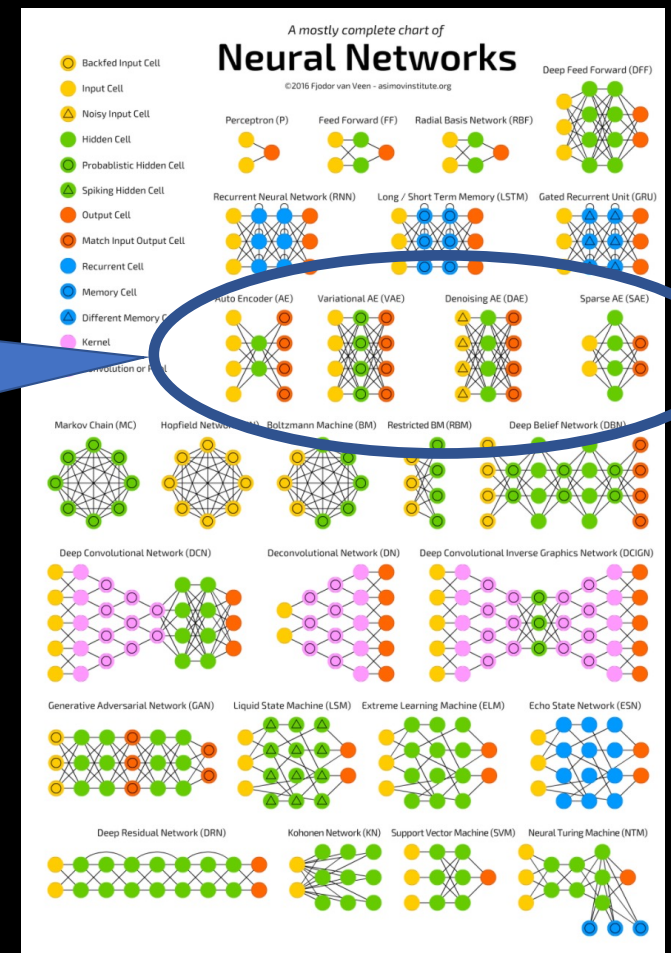
- <http://www.asimovinstitute.org/neural-network-zoo/>



The Neural Network Zoo

- <http://www.asimovinstitute.org/neural-network-zoo/>

Today

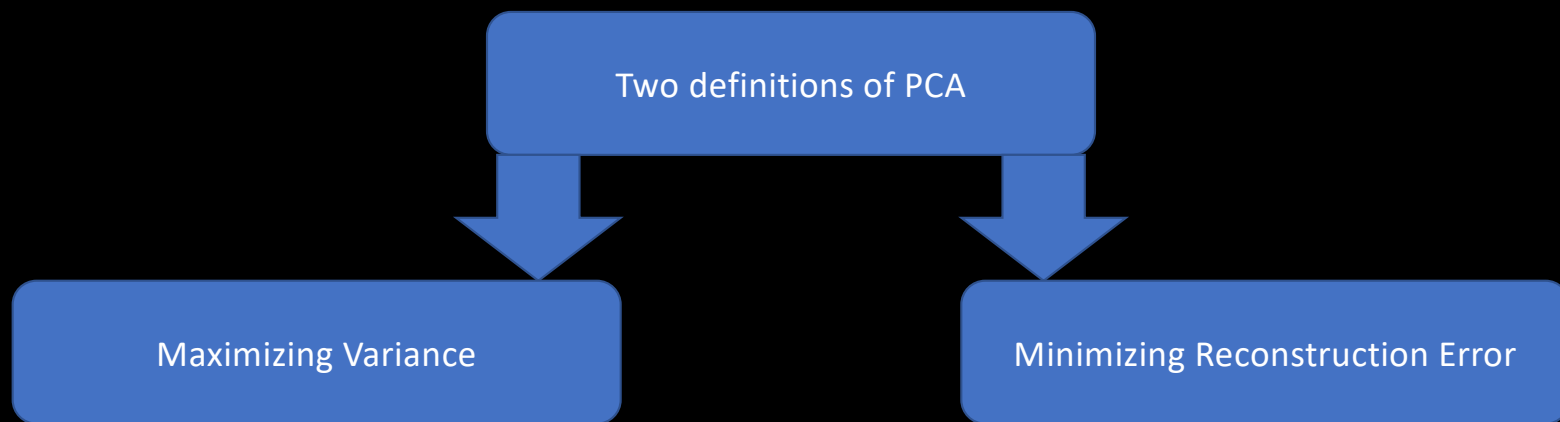


Dimensionality Reduction

- Reduce the number of random variables under consideration
 - Reduce computational cost of downstream analysis
 - Remove sources of noise in the data
 - Define an embedding of the data
 - Elucidate the manifold of the data
- **We've covered several strategies so far**

Principal Component Analysis (PCA)

1. Orthogonal projection of data
2. Lower-dimensional linear space known as the *principal subspace*
3. Variance of the projected data is maximized



Kernel PCA

- In kernel PCA, we consider data that have already undergone a nonlinear transformation:

$$\vec{x} \in \mathcal{R}^D \quad \longrightarrow \quad \phi(\vec{x}) \in \mathcal{R}^M$$

- **We now perform PCA on this new M -dimensional feature space**

Sparse PCA

- We still want to maximize $u_i^T S u_i$, subject to $u_i^T u_i = 1$
- ...and one more constraint: we want to *minimize* $\|u_i\|_1$
- Formalize these constraints using Lagrangian multipliers

$$\min_{W, U} \|X - WU^T\|_F^2 + \gamma \sum_{n=1}^N \|\vec{w}_i\|_1 + \gamma \sum_{i=1}^D \|\vec{u}_i\|_1$$

Stochastic SVD (SSVD)

- Uses **random projections** to find close approximation to SVD
- Combination of probabilistic strategies to maximize convergence likelihood
- Easily scalable to *massive* linear systems

Dictionary Learning

- This gives the minimization

$$\min_{B, \Theta} \sum_{i=1}^n \left(\|\vec{x}_i - B\vec{\theta}_i\|_q^q + h(\vec{\theta}_i) \right)$$

where h promotes sparsity in the coefficients, and B is chosen from a constraint set

- The general dictionary learning problem then follows

$$\phi(\Theta, B) = \frac{1}{2} \|X - B\Theta\|_F^2 + h(\Theta) + g(B)$$

where specific choices of h and g are what differentiate the different kinds of dictionary learning (e.g. hierarchical, K-SVD, etc)

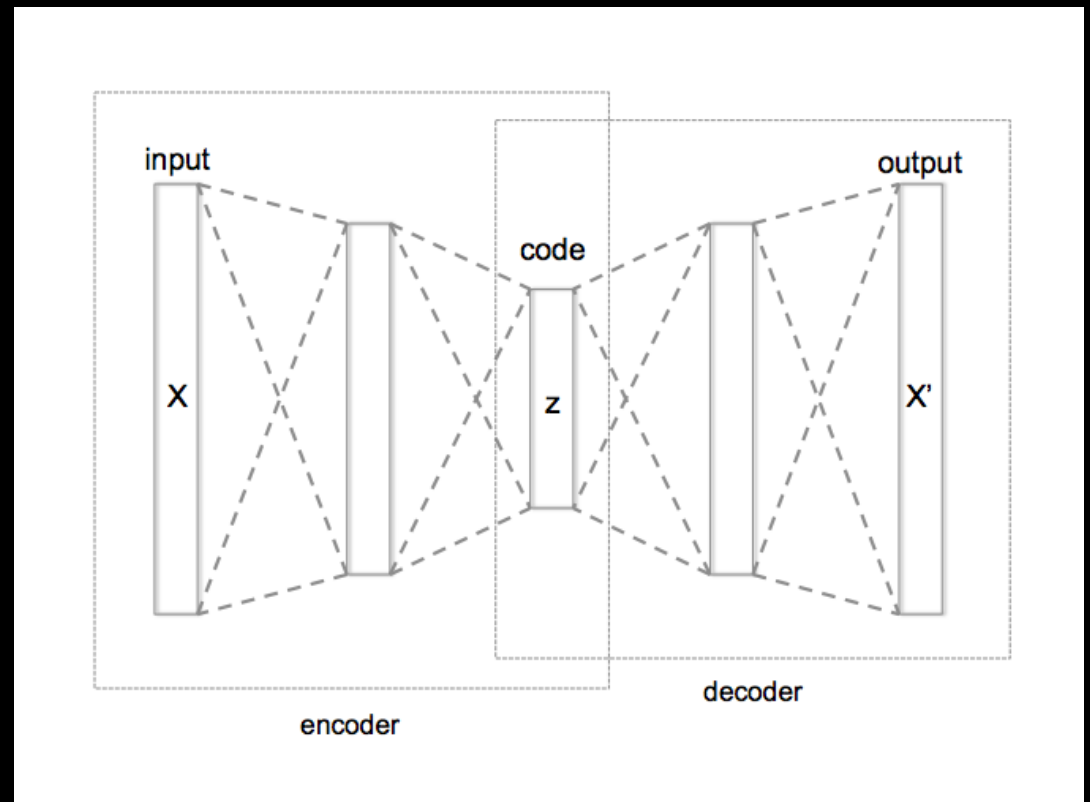
Autoencoders

- "Self encode"
- ANNs with output = input

$$\phi : \mathcal{X} \rightarrow \mathcal{F}$$

$$\psi : \mathcal{F} \rightarrow \mathcal{X}$$

$$\phi, \psi = \arg \min_{\phi, \psi} ||X - (\psi \circ \phi)X||^2$$



Autoencoders

- Learn a “non-trivial” identity function
- Low-dimensional “code”
- **No other assumptions**



- Very compact representation
- No strong *a priori* form (flexible)



- Difficult to interpret
- Prone to “collapse”

- PCA: maximize variance / minimize reconstruction
 - Linearly independent
 - Gaussian
- Dictionary Learning: sparse code / minimize reconstruction
 - Nonlinear
- Kernel / Sparse PCA

Autoencoders

- Key point: autoencoders should be **undercomplete**
 - Code dimension < input dimension

$$L(\vec{x}, g(f(\vec{x})))$$

- L is some loss function penalizing $g(f(x))$ for being dissimilar from x
- If f and g are linear, and L is mean squared error, undercomplete AE learns to span the same subspace as PCA

$$\phi, \psi = \arg \min_{\phi, \psi} ||X - (\psi \circ \phi)X||^2$$

$$U = \arg \min_U ||X - U\Lambda U^T||^2$$

Sparse Autoencoders

- $g(h)$ is decoder output
- $h = f(x)$, encoder output
- Ω is sparsity penalty
- Note on regularizer

$$L(\vec{x}, g(f(\vec{x}))) + \Omega(\vec{h})$$

$$p(\vec{\theta}, \vec{x}) \equiv \log p(\vec{x}|\vec{\theta}) + \log p(\vec{\theta})$$

No straightforward Bayesian interpretation of regularizer

“Typical” penalties can be viewed as a MAP approximation to Bayesian inference, with regularizers as priors over parameters

Regularized MAP then maximizes:

But autoencoder regularization relies **only** on the data. It's **more of a “preference over functions”** than a prior.

Denoising Autoencoders

- Instead of learning

$$L(\vec{x}, g(f(\vec{x})))$$

- Learn

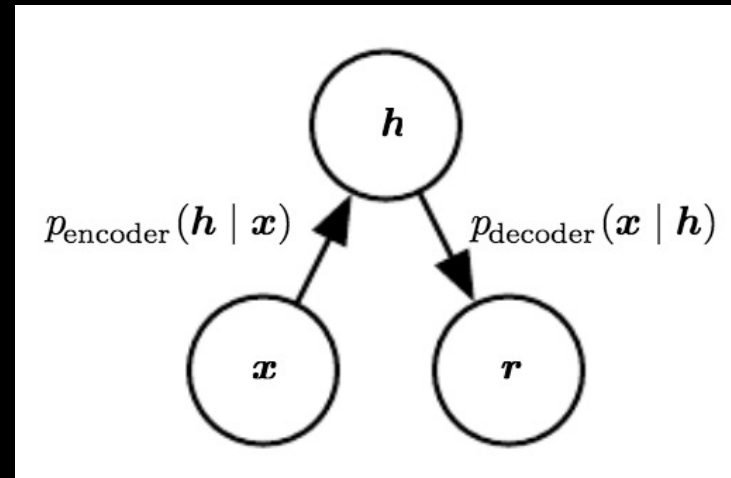
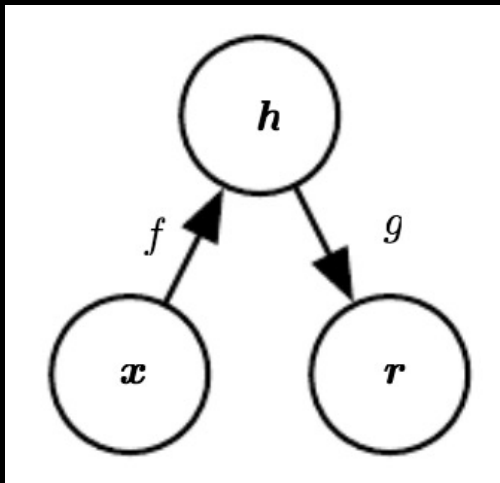
$$L(\vec{x}, g(f(\tilde{x})))$$

where \tilde{x} is a corrupted version of x

- Forces the autoencoder to learn the structure of $p_{data}(x)$
- **Form of “stochastic encoder / decoder”**

Denoising Autoencoders

- No longer deterministic!
- Given a hidden code h , minimize $-\log p_{\text{decoder}}(x|h)$



Denoising Autoencoders

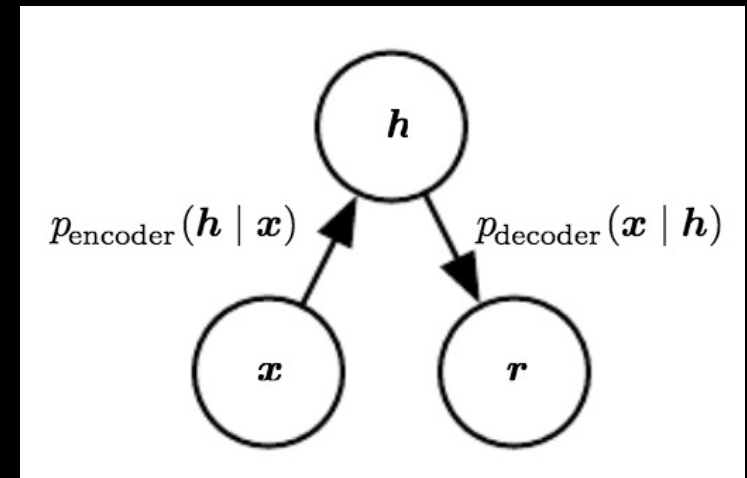
- Generalize encoding function to *encoding distribution*

$$p_{\text{encoder}}(\vec{h}|\vec{x}) = p_{\text{model}}(\vec{h}|\vec{x})$$

- Same with the *decoding distribution*

$$p_{\text{decoder}}(\vec{x}|\vec{h}) = p_{\text{model}}(\vec{x}|\vec{h})$$

- Together, these comprise a *stochastic encoder and decoder*



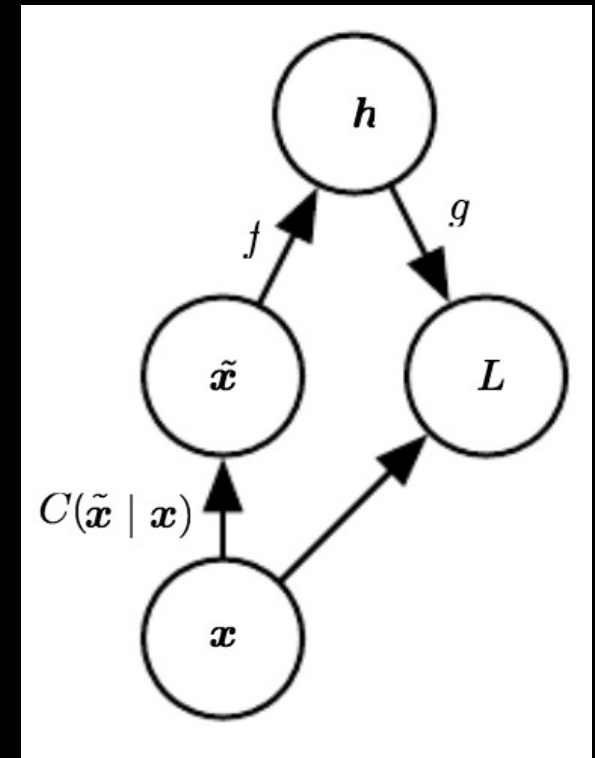
Denoising Autoencoders

- Define a corruption process, C

$$C(\tilde{x}|\vec{x})$$

- Autoencoder learns a *reconstruction distribution* $p_{reconstruct}(x|\tilde{x})$

1. Sample a training example x
2. Sample a corrupted version \tilde{x} from C
3. Use (x, \tilde{x}) as a training pair



Denoising Autoencoders

- Optimize

$$-\mathbb{E}_{\vec{x} \sim \hat{p}_{\text{data}}}(\vec{x}) \mathbb{E}_{\tilde{x} \sim C(\tilde{x}|\vec{x})} \log p_{\text{decoder}}(\vec{x} | \vec{h} = f(\tilde{x}))$$

Sample from training set and compute expectation

Expectation over corrupted examples

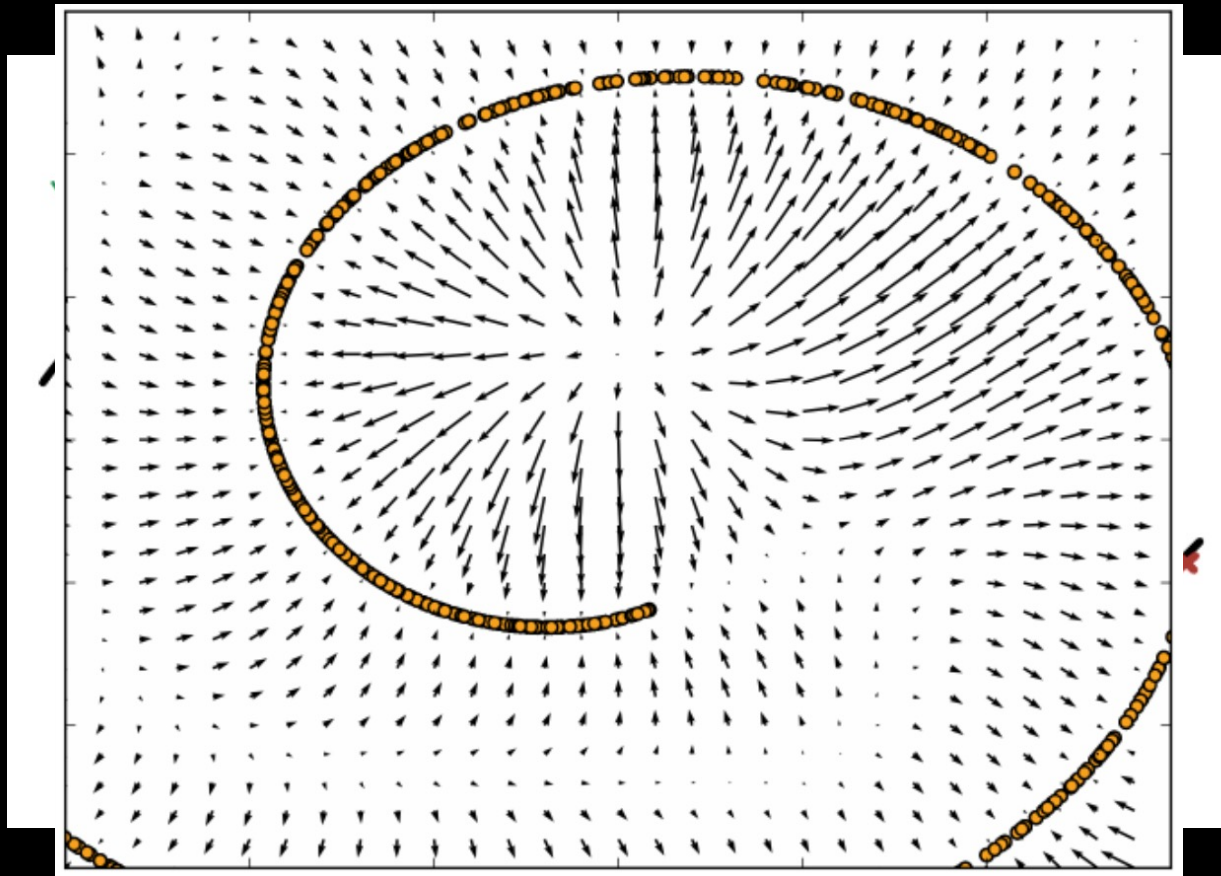
...with respect to learning the *uncorrupted data* from the encoded corrupted data

- Easy choice of C

$$C(\tilde{x}|\vec{x}) = \mathcal{N}(\tilde{x}; \mu = \vec{x}, \Sigma = \sigma^2 I)$$

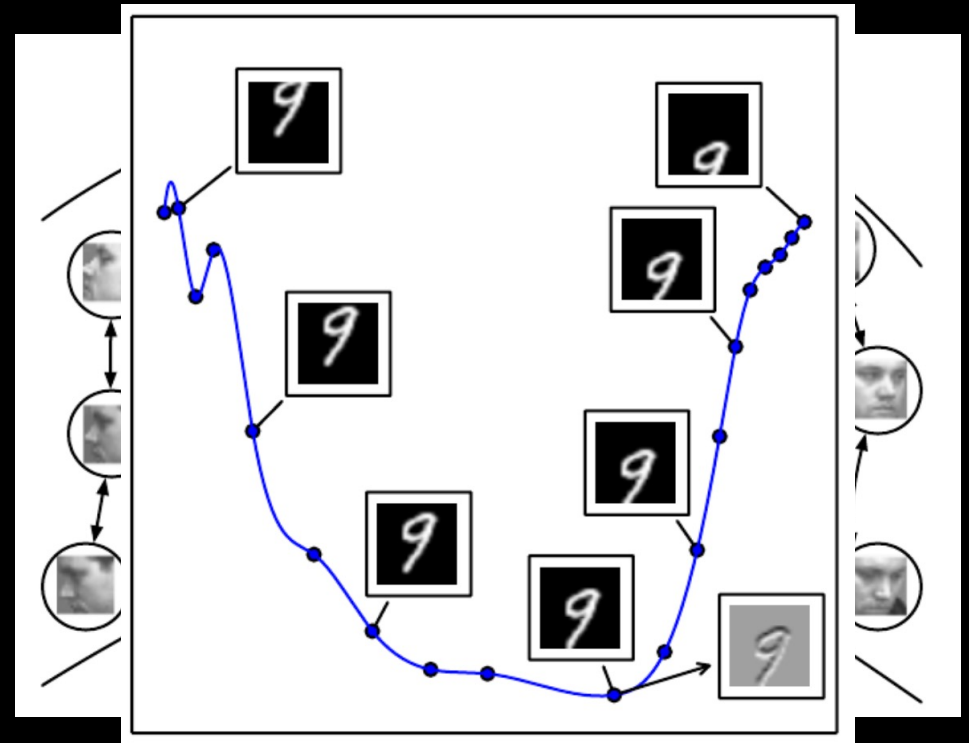
Denoising Autoencoders

- DAEs train to map \tilde{x} back to uncorrupted x
- Gray circle = equiprobable C
- Vector from \tilde{x} points approximately to nearest x on manifold
- **DFA learns a vector field around a manifold**



Embeddings

- Manifolds would seem to imply *representation learning* beyond a simple low-dimensional code
- Autoencoders can learn powerful relationships in this regard
 - Pose
 - Position
 - Affine transformations



Generative Models

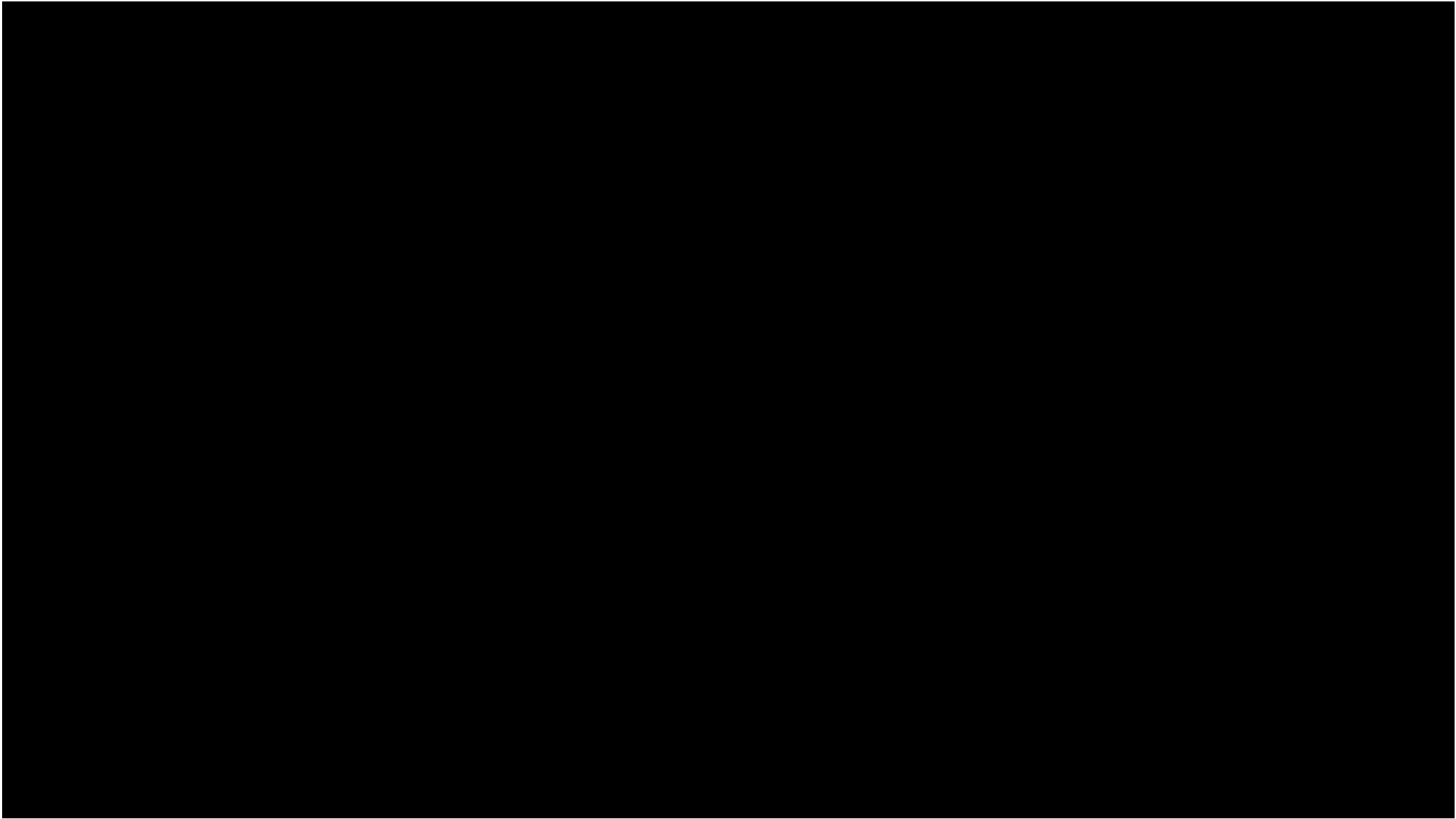
- Go beyond learning $x \rightarrow h$, instead focused on learning $p(x, h)$
- Manifold learning with Autoencoders
- Variational Autoencoders (VAEs)
- Deep Belief Networks (DBNs)
- Deep Restricted Boltzmann Machines (DBMs)
- Generative Adversarial Networks (GANs)
- **More next week!**

Conclusions

- Autoencoders
 - Multilayer perceptron (ANN) that is symmetric
 - Output = input
 - Goal is to learn a non-trivial identity function, or an undercomplete code h
- Sparse Autoencoders
 - Include a sparsity constraint on the code
- Denoising Autoencoders
 - Learn a mapping to de-corrupt data
 - Include a corruption process C
 - Equates to a traversal of the data manifold -> **generative modeling primer**

References

- *Deep Learning Book*, Chapter 14: “Autoencoders”
<http://www.deeplearningbook.org/contents/autoencoders.html>
- DL4J documentation, “Denoising Autoencoders”
<http://deeplearning.net/tutorial/dA.html>



A brief aside: SSVD

- Matrix A
 - Find a low-rank approximation of A
 - Basic dimensionality reduction

$$\|A - QQ^*A\| < \epsilon$$



Preconditioning

Approximating range of A

- INPUT: A, k, p
- OUTPUT: Q

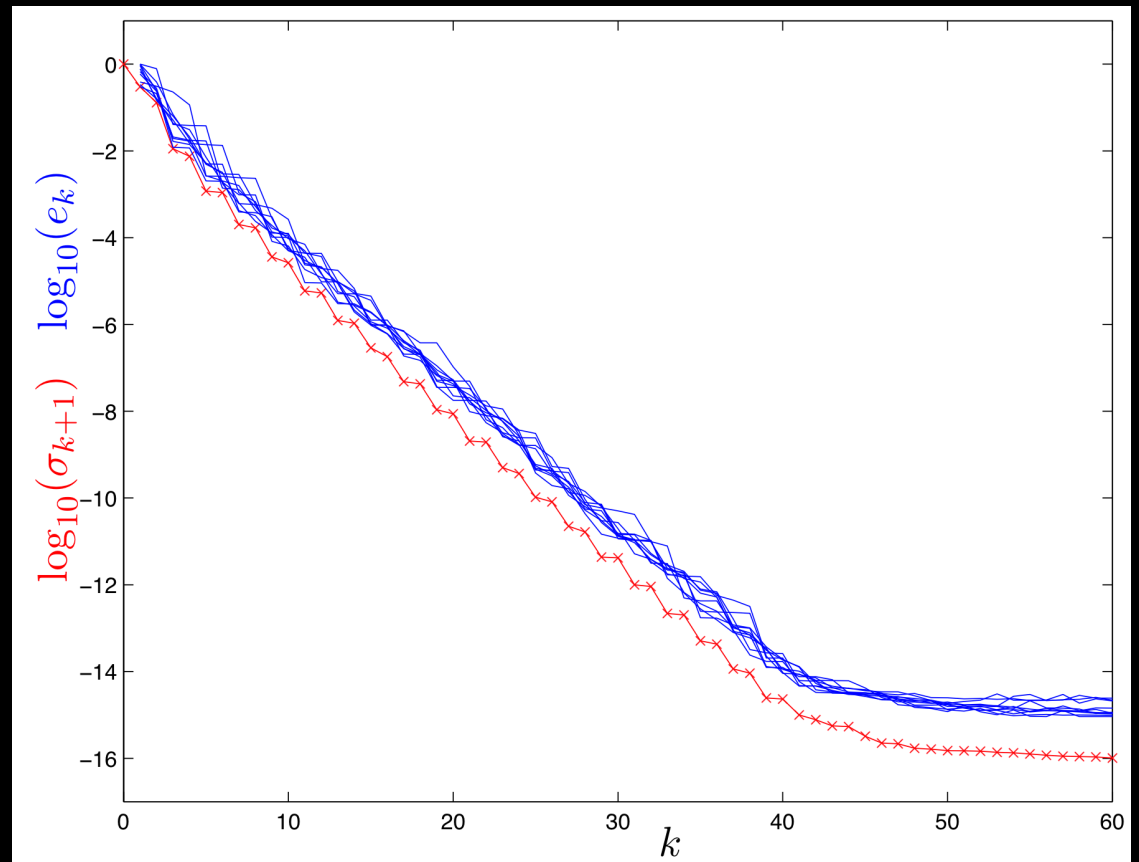
1. Draw Gaussian $n \times k$ test matrix Ω
2. Form product $Y = A\Omega$
3. Orthogonalize columns of $Y \rightarrow Q$

Approximating SVD of A

- INPUT: Q
 - OUTPUT: Singular vectors U
1. Form $k \times n$ matrix $B = Q^T A$
 2. Compute SVD of $B = \hat{U} \Sigma V^T$
 3. Compute singular vectors $U = Q \hat{U}$

Empirical Results

- 1000x1000 matrix
- Several runs of empirical results (blue) to theoretical lower bound (red)
- **Error seems to be systemic**



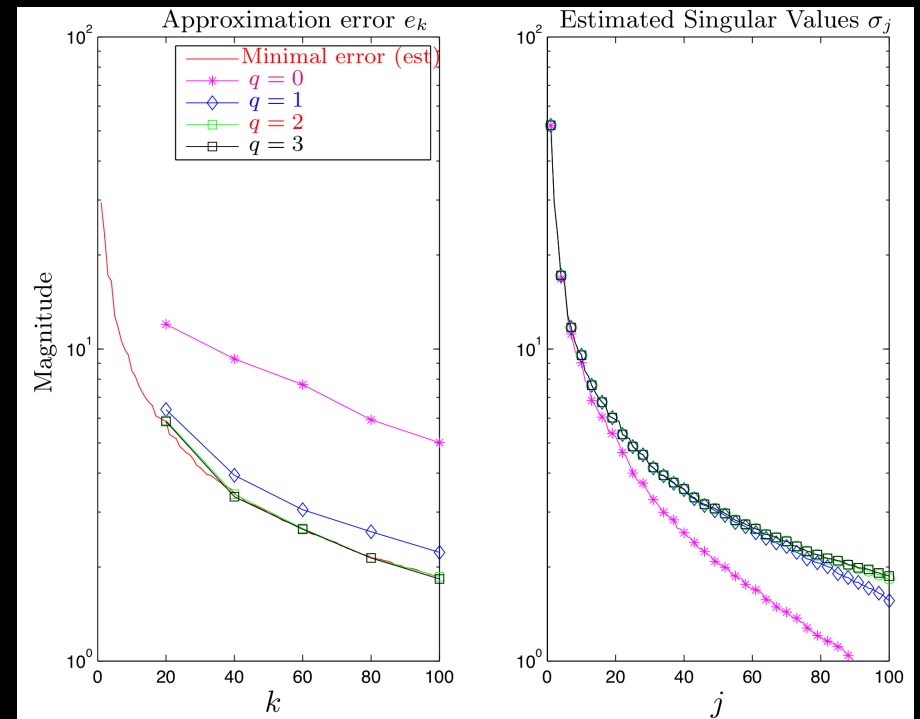
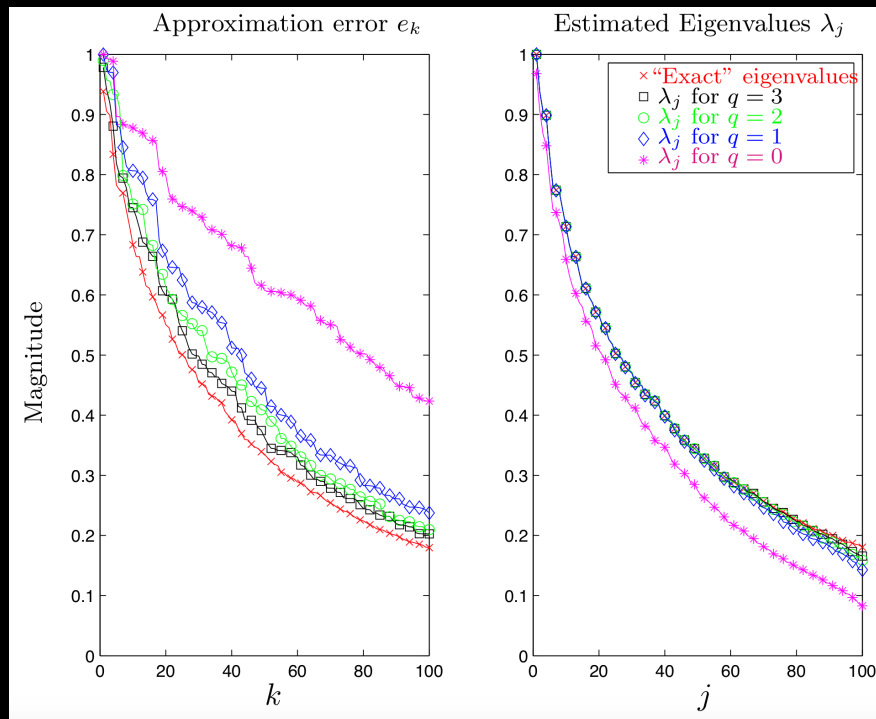
Power iterations

- Affects decay of eigenvalues / singular values

$$Y = \cancel{A} \Omega.$$

$$Y = (A A^*)^q A \Omega$$

Empirical Results



Why does this work?

- Three primary reasons:

1. Johnson-Lindenstrauss Lemma

- Low-dimensional embeddings preserve pairwise distances

$$(1 - \varepsilon)\|u - v\|^2 \leq \|f(u) - f(v)\|^2 \leq (1 + \varepsilon)\|u - v\|^2$$

2. Concentration of measure

- Geometric interpretation of classical idea: regular functions of independent random variables rarely deviate far from their means

3. Preconditioning

- Condition number: how much change in output is produced from change in input (relation to #1)
- Q matrix lowers condition number while preserving overall system

$$\kappa = \frac{|\lambda_{\max}|}{|\lambda_{\min}|}$$

Course Details

- **Projects!**
 - 3 presentations per day
 - 9 teams—**20 minutes hard speaking time limit**
 - Presentations are the week after Thanksgiving break
- **Thursday is FULL**
- **Wednesday is ALMOST FULL**

- **First come, first serve!**

Tues,
11/28

Final Project Presentations

Wed,
11/29

Final Project Presentations

Thurs,
11/30

Final Project Presentations

Thurs,
12/7

Final Project Deliverables Due