Autoencoders

Sargur Srihari
srihari@buffalo.edu

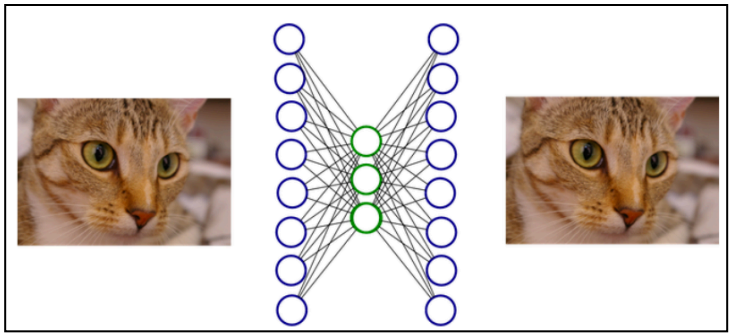
Topics in Autoencoders

- What is an autoencoder?

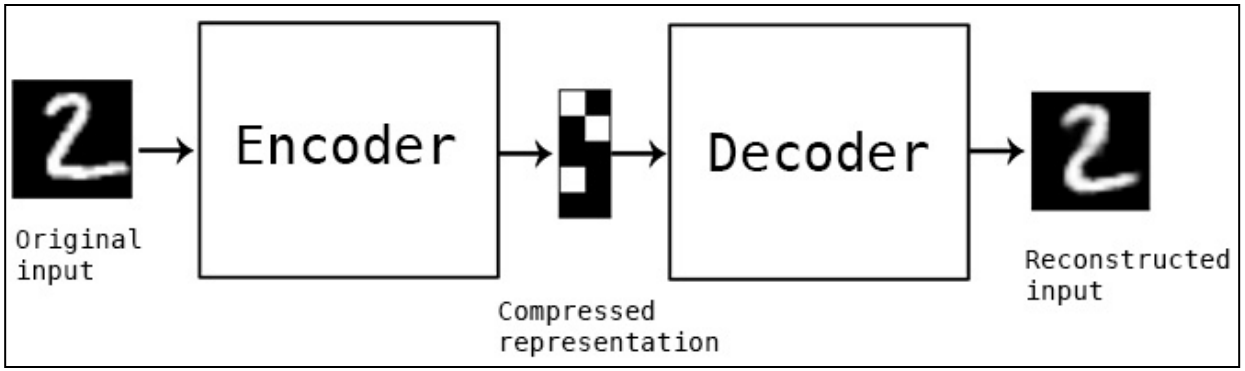
1. Undercomplete Autoencoders
2. Regularized Autoencoders
3. Representational Power, Layout Size and Depth
4. Stochastic Encoders and Decoders
5. Denoising Autoencoders
6. Learning Manifolds and Autoencoders
7. Contractive Autoencoders
8. Predictive Sparse Decomposition
9. Applications of Autoencoders

What is an Autoencoder?

- An autoencoder is a data compression algorithm
  - Typically an artificial neural network trained to copy its input to its output



- A hidden layer describes the code used to represent the input
  - Maps input to output through a compressed representation code

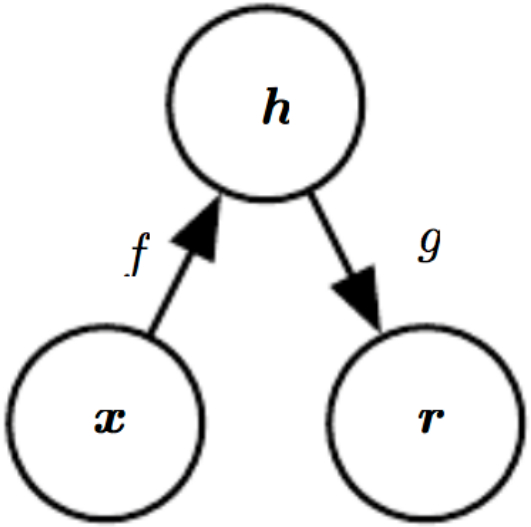


Autoencoders differ from General Data Compression

- Autoencoders are data-specific
  - i.e., only able to compress data similar to what they have been trained on
- This is different from, say, MP3 or JPEG compression algorithm
  - Which make general assumptions about "sound/images", but not about specific types of sounds/images
  - Autoencoder for pictures of cats would do poorly in compressing pictures of trees
    - Because features it would learn would be cat-specific
- Autoencoders are lossy
  - which means that the decompressed outputs will be degraded compared to the original inputs (similar to MP3 or JPEG compression).
  - This differs from lossless arithmetic compression
- Autoencoders are learnt

General structure of an autoencoder

- Maps an input $\boldsymbol{x}$ to an output $\boldsymbol{r}$ (called reconstruction) through an internal representation code $\boldsymbol{h}$
  - It has a hidden layer $\boldsymbol{h}$ that describes a code used to represent the input
- The network has two parts
  - The encoder function $\boldsymbol{h}=f(\boldsymbol{x})$
  - A decoder that produces a reconstruction $\boldsymbol{r}=g(\boldsymbol{h})$



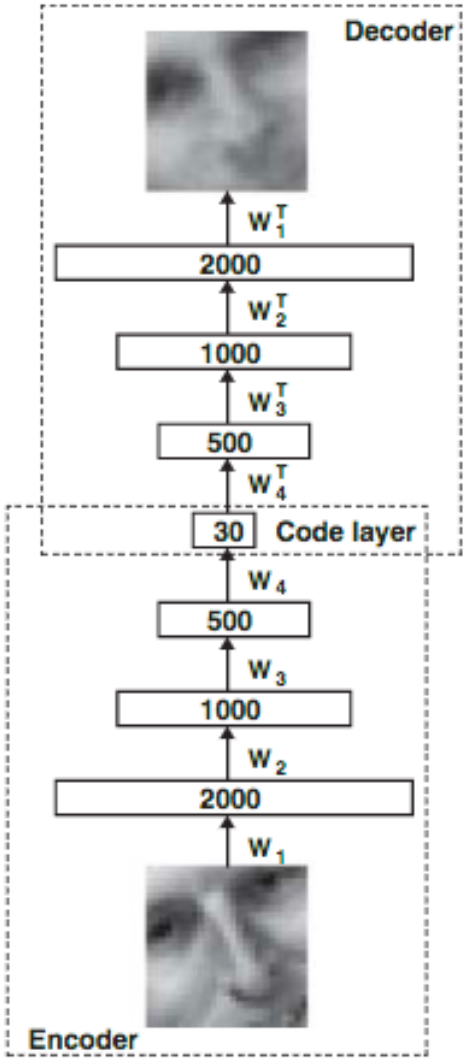
Rationale of an Autoencoder

- An autoencoder that simply learns to set $g\left(f\left(\boldsymbol{x}\right)\right)=\boldsymbol{x}$ everywhere is not especially useful
- Autoencoders are designed to be unable to copy perfectly
  - They are restricted in ways to copy only approximately
  - Copy only input that resembles training data
- Because model is forced to prioritize which aspects of input should be copied, it often learns useful properties of the data
- Modern autoencoders have generalized the idea of encoder and decoder beyond deterministic functions to stochastic mappings $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$ and $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$

Autoencoder History

- Part of neural network landscape for decades
- Traditionally used for dimensionality reduction and feature learning
- Recent theoretical connection between autoencoders and latent variable models have brought them into forefront of generative models

An autoencoder architecture



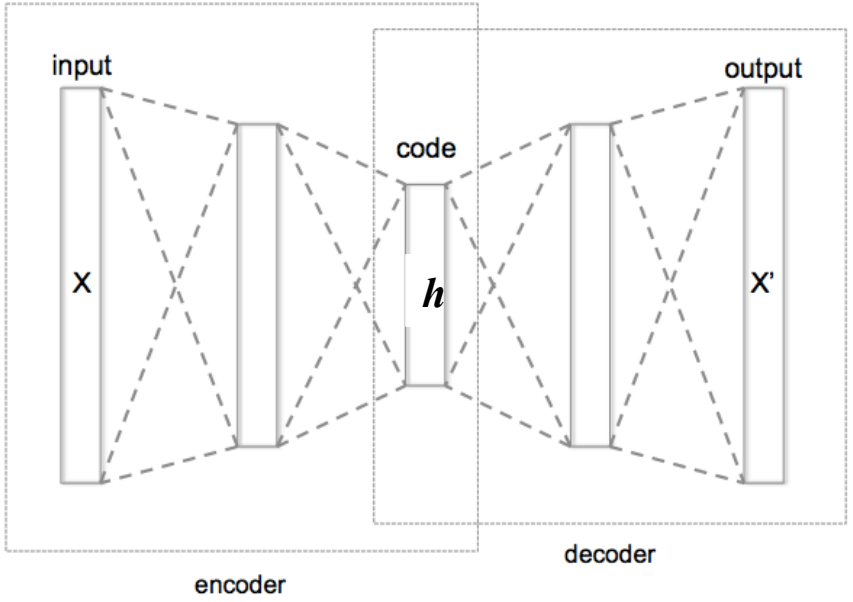
The weights are learnt from training samples, as discussed next

Autoencoder training using a loss function

- Encoder f and decoder g

$$f : X \rightarrow \boldsymbol{h}$$
$$g : \boldsymbol{h} \rightarrow X$$
$$\arg\min_{f,g} \left\| X - (f \circ g)X \right\|^2$$

Autoencoder with 3 fully connected hidden layers



- One hidden layer
  - Non-linear encoder
  - Takes input $\boldsymbol{x} \in R^d$
  - Maps into output $\boldsymbol{h} \in R^p$

$$\boldsymbol{h} = \sigma_1(W\boldsymbol{x} + \boldsymbol{b})$$
$$\boldsymbol{x}' = \sigma_2(W'\boldsymbol{h} + \boldsymbol{b}')$$

σ is an element-wise activation function such as sigmoid or Relu

Trained to minimize reconstruction error (such as sum of squared errors)

$$L(\boldsymbol{x}, \boldsymbol{x}') = \left\| \boldsymbol{x} - \boldsymbol{x}' \right\|^2 = \left\| \boldsymbol{x} - \sigma_2(W^t(\sigma_1(W\boldsymbol{x} + \boldsymbol{b})) + \boldsymbol{b}') \right\|^2$$

Provides a compressed representation of the input $\boldsymbol{x}$

Two Autoencoder Training Methods

1. An autoencoder is a feed-forward non-recurrent neural net
   - With an input layer, an output layer and one or more hidden layers
   - Can be trained using the same techniques
     - Compute gradients using back-propagation
     - Followed by minibatch gradient descent
2. Unlike feedforward networks, can also be trained using *Recirculation*
   - Compare activations on the input to activations of the reconstructed input
   - More biologically plausible than back-prop but rarely used in ML

1. Undercomplete Autoencoder

- Copying input to output sounds useless
- But we are not interested in the output of the decoder
- We hope that training the autoencoder to perform copying task will result in $\boldsymbol{h}$ taking on useful properties
- To obtain useful features, constrain $\boldsymbol{h}$ to have lower dimension than $\boldsymbol{x}$
  - Such an autoencoder is called undercomplete
  - Learning the undercomplete representation forces the autoencoder to capture most salient features of training data

Autoencoder with linear decoder +MSE is PCA

- Learning process is that of minimizing a loss function

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

  - where L is a loss function penalizing $g(\ f\ (\boldsymbol{x}))$ for being dissimilar from $\boldsymbol{x}$, such as L^2 norm of difference: mean squared error
  - When the decoder g is linear and L is the mean squared error, an undercomplete autoencoder learns to span the same subspace as PCA
    - In this case the autoencoder trained to perform the copying task has learned the principal subspace of the training data as a side-effect
- Autoencoders with nonlinear f and g can learn more powerful nonlinear generalizations of PCA
  - But that is not desireable as seen next

Encoder/Decoder Capacity

- If encoder f and decoder g are allowed too much capacity
  - autoencoder can learn to perform the copying task without learning any useful information about distribution of data
- Autoencoder with a one-dimensional code and a very powerful nonlinear encoder can learn to map $\boldsymbol{x}^{(i)}$ to code i.
  - The decoder can learn to map these integer indices back to the values of specific training examples
- Autoencoder trained for copying task fails to learn anything useful if f/g capacity is too great

Cases when Autoencoder Learning Fails

- Where autoencoders fail to learn anything useful:
  1. Capacity of encoder/decoder f/g is too high
     - Capacity controlled by depth
  2. Hidden code $\boldsymbol{h}$ has dimension equal to input $\boldsymbol{x}$
  3. *Overcomplete* case: where hidden code $\boldsymbol{h}$ has dimension greater than input $\boldsymbol{x}$
     - Even a linear encoder/decoder can learn to copy input to output without learning anything useful about data distribution

What is the Right Autoencoder Design?

- Ideally, choose code size (dimension of $\boldsymbol{h}$) small and capacity of f / g based on complexity of distribution modeled
- *Regularized autoencoders* provide the ability to do so
  - Use a loss function that encourages the model to have properties other than copy its input to output

2. Regularized Autoencoder Properties

- Regularized autoencoders have properties beyond
  - Keeping encoder/decoder shallow and code size small
  - Ability to copy its input to its output
- Other properties of regularized autoencoders:
  - Sparsity of representation
  - Smallness of the derivative of the representation
  - Robustness to noise
  - Robustness to missing inputs
- Regularized autoencoder can be nonlinear and overcomplete
  - But still learn something useful about the data distribution even if model capacity is great enough to learn trivial identity function

Generative Models Viewed as Autoencoders

- Beyond regularized autoencoders
- Generative models with latent variables and an inference procedure (for computing latent representations given input) can be viewed as a particular form of autoencoder
- Generative modeling approaches which emphasize connection with autoencoders are descendants of Helmholtz machine:
  1. Variational autoencoder
  2. Generative stochastic networks

Sparse Autoencoders

- A sparse autoencoder is an autoencoder whose
  - Training criterion includes a sparsity penalty $\Omega(\boldsymbol{h})$ on the code layer $\boldsymbol{h}$ in addition to the reconstruction error:

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

  - where $g(\boldsymbol{h})$ is the decoder output and typically we have $\boldsymbol{h} = f(\boldsymbol{x})$
- Sparse encoders are typically used to learn features for another task such as classification
- An autoencoder that has been trained to be sparse must respond to unique statistical features of the dataset rather than simply perform the copying task
  - Thus sparsity penalty can yield a model that has learned useful features as a byproduct

Sparse Encoder doesn't have Bayesian Interpretation

- Penalty term $\Omega(\boldsymbol{h})$ is a regularizer term added to a feedforward network whose
  - Primary task: copy input to output (with *Unsupervised* learning objective)
  - Also perform some supervised task (with *Supervised* learning objective) that depends on the sparse features
- In supervised learning regularization term corresponds to prior probabilities over model parameters
  - Regularized MLE corresponds to maximizing $p(\boldsymbol{\theta}|\boldsymbol{x})$, which is equivalent to maximizing $\log\ p(\boldsymbol{x}|\boldsymbol{\theta})+\log\ p(\boldsymbol{\theta})$
    - First term is data log-likelihood and second term is log-prior over parameters
  - Regularizer depends on data and thus is not a prior
    - Instead, regularization terms express a preference over functions

Generative Model view of Sparse Autoencoder

- Rather than thinking of sparsity penalty as a regularizer for copying task, think of sparse autoencoder as approximating ML training of a generative model that has latent variables
- Suppose model has visible/latent variables $\boldsymbol{x}$ and $\boldsymbol{h}$
- Explicit joint distribution is $p_{\text{model}}(\boldsymbol{x},\boldsymbol{h}) = p_{\text{model}}(\boldsymbol{h})\ p_{\text{model}}(\boldsymbol{x}|\boldsymbol{h})$
  - where $p_{\text{model}}(\boldsymbol{h})$ is model's prior distribution over latent variables
    - Different from $p(\boldsymbol{\theta})$ being distribution of parameters
- The log-likelihood can be decomposed as $\log p_{\text{model}}(\boldsymbol{x},\boldsymbol{h}) = \log \sum_{\boldsymbol{h}} p_{\text{model}}(\boldsymbol{h},\boldsymbol{x})$
- Autoencoder approximates the sum with a point estimate for just one highly likely value of $\boldsymbol{h}$, the output of a parametric encoder
  - With a chosen $\boldsymbol{h}$ we are maximizing $\log\ p_{\text{model}}(\boldsymbol{x},\boldsymbol{h}) = \log\ p_{\text{model}}(\boldsymbol{h}) + \log\ p_{\text{model}}(\boldsymbol{x}|\boldsymbol{h})$

Sparsity-inducing Priors

- The $\log p_{\text{model}}(\boldsymbol{h})$ term can be sparsity-inducing. For example the Laplace prior

$$p_{\text{model}}(h_i) = \frac{\lambda}{2} e^{-\lambda |h_i|}$$

  - corresponds to an absolute value sparsity penalty
- Expressing the log-prior as an absolute value penalty

$$\boxed{-\log p_{\text{model}}(\boldsymbol{h}) = \sum_i \left(\lambda \, | \, h_i \, | - \log \frac{\lambda}{2} \right) = \Omega(\boldsymbol{h}) + const}$$

where $\Omega(\boldsymbol{h}) = \lambda \sum_i h_i$

  - where the constant term depends only on λ and not on $\boldsymbol{h}$
- We treat λ as a hyperparameter and discard the constant term, since it does not affect parameter learning

Denoising Autoencoders (DAE)

- Rather than adding a penalty Ω to the cost function, we can obtain an autoencoder that learns something useful
  - By changing the reconstruction error term of the cost function
- Traditional autoencoders minimize $L(\boldsymbol{x}, g(f(\boldsymbol{x})))$
  - where L is a loss function penalizing $g(f(\boldsymbol{x}))$ for being dissimilar from $\boldsymbol{x}$, such as L^2 norm of difference: mean squared error
- A DAE minimizes $L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}})))$
  - where $\tilde{\boldsymbol{x}}$ is a copy of $\boldsymbol{x}$ that has been corrupted by some form of noise
  - The autoencoder must undo this corruption rather than simply copying their input
- Denoising training forces f and g to implicitly learn the structure of $p_{\text{data}}(\boldsymbol{x})$
- Another example of how useful properties can emerge as a by-product of minimizing reconstruction error

Regularizing by Penalizing Derivatives

- Another strategy for regularizing an autoencoder
- Use penalty as in sparse autoencoders

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

- But with a different form of $\boldsymbol{\Omega}$

$$\boxed{\Omega(\boldsymbol{h},\boldsymbol{x}) = \lambda \sum_i \left\| \nabla_{\boldsymbol{x}} h_i \right\|^2}$$

- Forces the model to learn a function that does not change much when $\boldsymbol{x}$ changes slightly
- Called a *Contractive Auto Encoder* (CAE)
- This model has theoretical connections to
  - Denoising autoencoders
  - Manifold learning
  - Probabilistic modeling

3. Representational Power, Layer Size and Depth

- Autoencoders are often trained with with single layer
- However using deep encoder offers many advantages
  - Recall: Although universal approximation theorem states that a single layer is sufficient, there are disadvantages:
    1. no of units needed may be too large
    2. may not generalize well
- Common strategy: greedily pretrain a stack of shallow autoencoders

Stochastic Encoders and Decoders

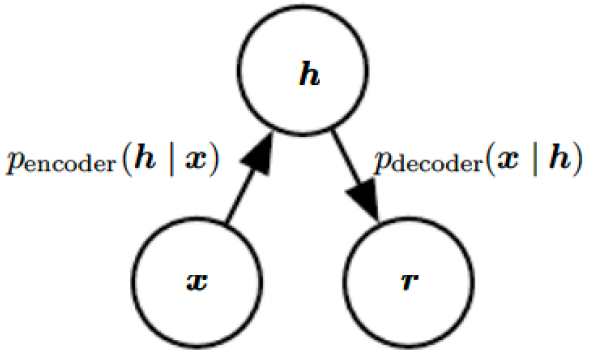
- General strategy for designing the output units and loss function of a feedforward network is to
  - Define the output distribution $p(\boldsymbol{y}|\boldsymbol{x})$
  - Minimize the negative log-likelihood $-\log\ p(\boldsymbol{y}|\boldsymbol{x})$
  - In this setting $\boldsymbol{y}$ is a vector of targets such as class labels
- In an autoencoder $\boldsymbol{x}$ is the target as well as the input
  - Yet we can apply the same machinery as before, as we see next

Loss function for Stochastic Decoder

- Given a hidden code $\boldsymbol{h}$, we may think of the decoder as providing a conditional distribution $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$
- We train the autoencoder by minimizing $-\log\ p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$
- The exact form of this loss function will change depending on the form of $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$
- As with feedforward networks we use linear output units to parameterize the mean of the Gaussian distribution if $\boldsymbol{x}$ is real
  - In this case negative log-likelihood is the mean-squared error
- With binary $\boldsymbol{x}$ correspond to a Bernoulli with parameters given by a sigmoid
- Discrete $\boldsymbol{x}$ values correspond to a softmax
- The output variables are treated as being conditionally independent given $\boldsymbol{h}$

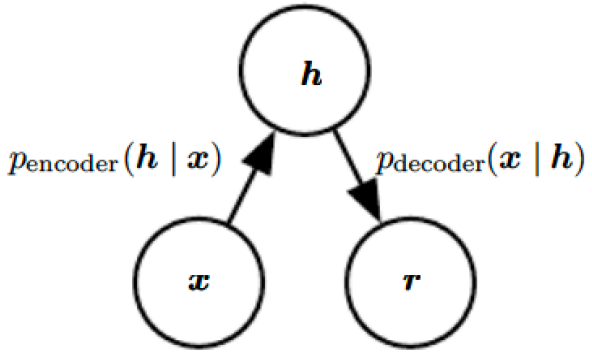
Stochastic encoder

- We can also generalize the notion of an encoding function $f(\boldsymbol{x})$ to an encoding distribution $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$



Structure of stochastic autoencoder

- Both the encoder and decoder are not simple functions but involve a distribution
- The output is sampled from a distribution $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$ for the encoder and $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$ for the decoder

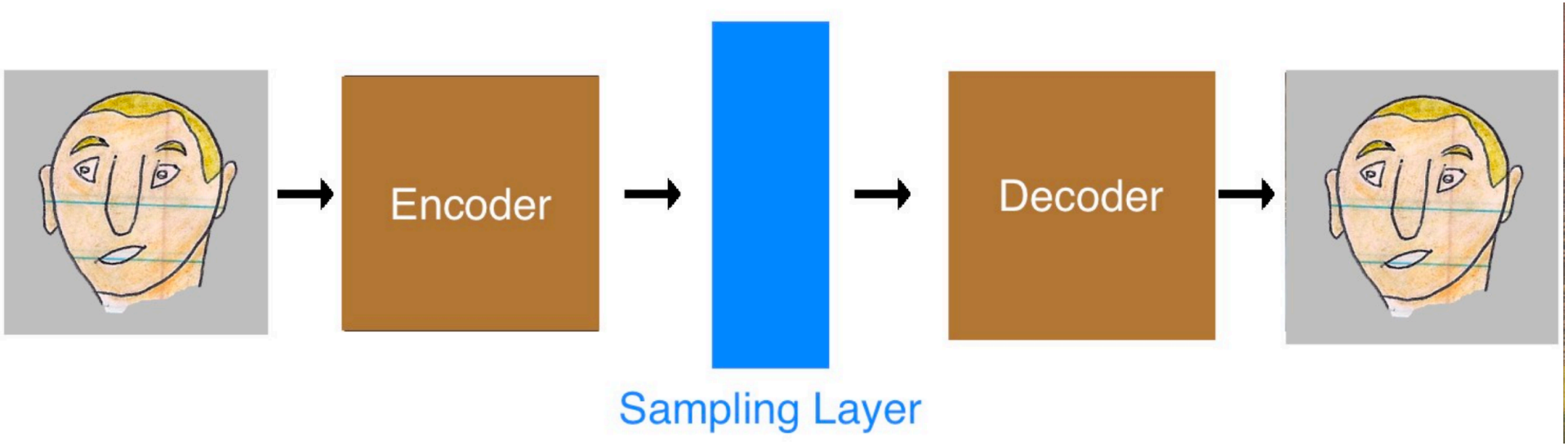


Relationship to joint distribution

- Any latent variable model $p_{\text{model}}(\boldsymbol{h}|\boldsymbol{x})$ defines a stochastic encoder $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})=p_{\text{model}}(\boldsymbol{h}|\boldsymbol{x})$
- And a stochastic decoder $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})=p_{\text{model}}(\boldsymbol{x}|\boldsymbol{h})$
- In general the encoder and decoder distributions are not conditional distributions compatible with a unique joint distribution $p_{\text{model}}(\boldsymbol{x},\boldsymbol{h})$
- Training the autoencoder as a denoising autoencoder will tend to make them compatible asymptotically
  - With enough capacity and examples

Sampling $p_{\text{model}}(\boldsymbol{h}|\boldsymbol{x})$

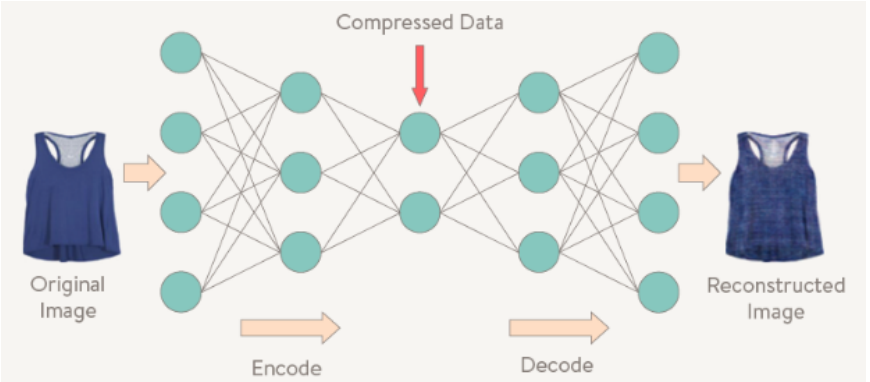
$\boldsymbol{x}$

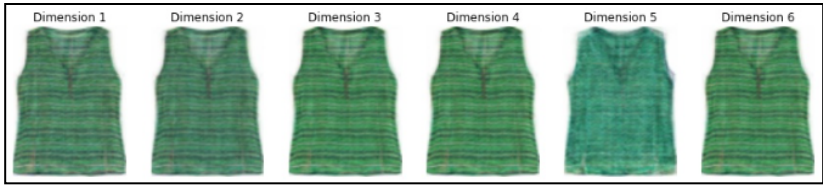


$p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$ $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$

Ex: Sampling $p(\boldsymbol{x}|\boldsymbol{h})$: Deepstyle

- Boil down to a representation which relates to style
  - By iterating neural network through a set of images learn efficient representations
- Choosing a random numerical description in encoded space will generate new images of styles not seen
- Using one input image and changing values along different dimensions of feature space you can see how the generated image changes (patterning, color texture) in style space



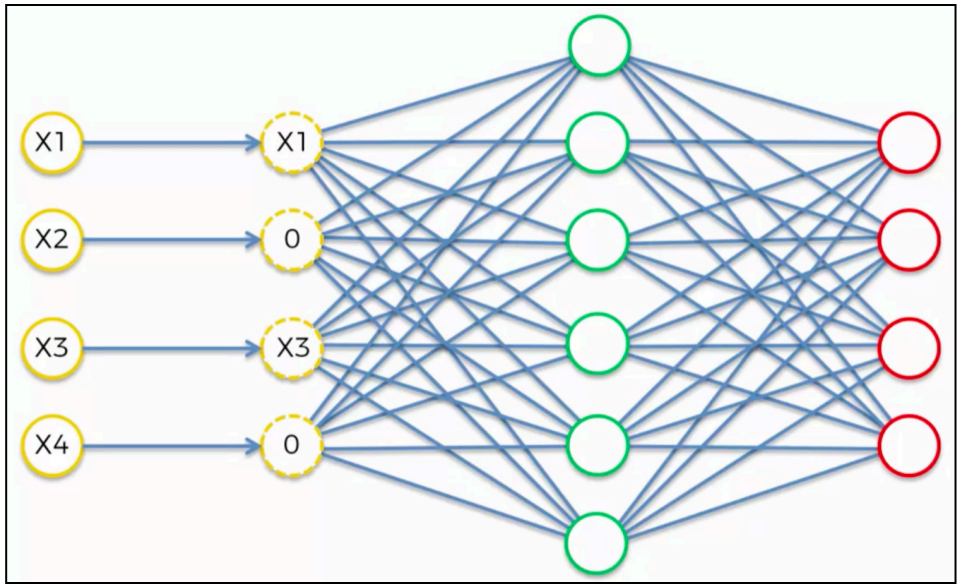


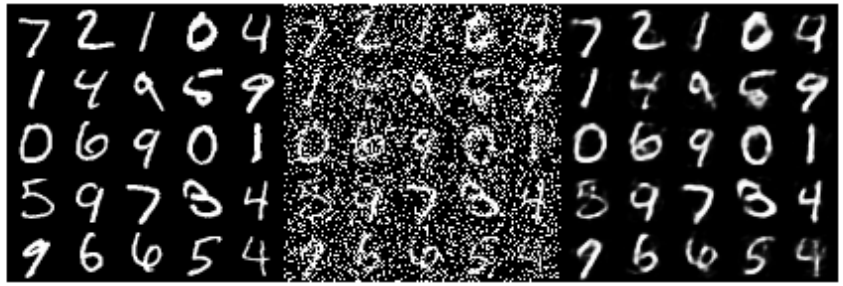
5. Denoising Autoencoders

- The *denoising autoencoder* (DAE) is an autoencoder that receives a corrupted data point as input and is trained to predict the original, uncorrupted data point as its output
  - Traditional autoencoders minimize $L(\boldsymbol{x}, g(f(\boldsymbol{x})))$
    - where L is a loss function penalizing $g(f(\boldsymbol{x}))$ for being dissimilar from $\boldsymbol{x}$, such as L^2 norm of difference: mean squared error
  - A DAE minimizes $L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}})))$
    - where $\tilde{\boldsymbol{x}}$ is a copy of $\boldsymbol{x}$ that is corrupted by some form of noise
    - The autoencoder must undo this corruption rather than simply copying their input

Example of Noise in a DAE

- An autoencoder with high capacity can end up learning an identity function (also called null function) where input=output
- A DAE can solve this problem by corrupting the data input
- How much noise to add?
  - Corrupt input nodes by setting 30-50% of random input nodes to zero

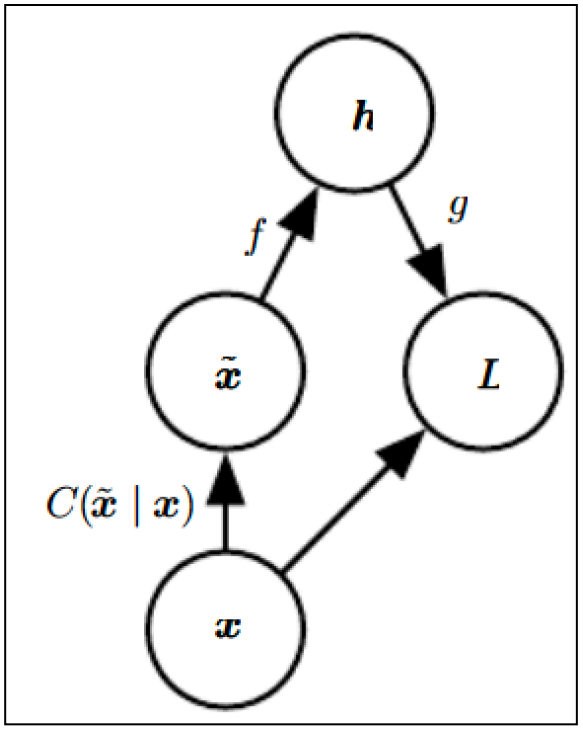




Original input, corrupted data, reconstructed data

DAE Training procedure

- Computational graph of cost function below
  - DAE trained to reconstruct clean data point $\boldsymbol{x}$ from the corrupted
    Accomplished by minimizing loss L=-log $p_{\text{encoder}}(\boldsymbol{x}|\boldsymbol{h}=f(\boldsymbol{x}))$



Corruption process, $C(\tilde{\boldsymbol{x}}\ |\boldsymbol{x})$ is a conditional distribution over corrupted samples $\tilde{\boldsymbol{x}}$ given the data sample $\boldsymbol{x}$

The autoencoder learns a reconstruction distribution $p_{\text{reconstruct}}(\boldsymbol{x}|\ \tilde{\boldsymbol{x}}\ )$ estimated from training pairs $(\boldsymbol{x},\ \tilde{\boldsymbol{x}}\ )$ as follows:

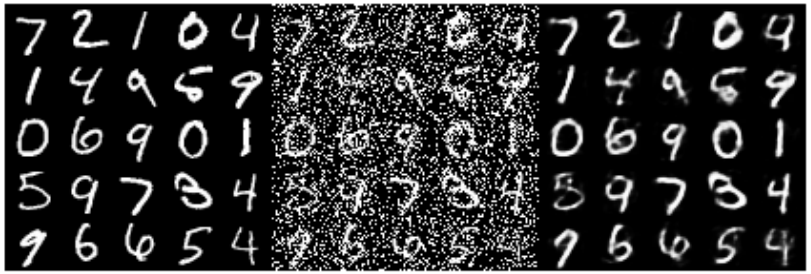
1 Sample a training sample $\boldsymbol{x}$ from the training data
2. Sample a corrupted version $\tilde{\boldsymbol{x}}$ from $C(\ \tilde{\boldsymbol{x}}\ |\boldsymbol{x})$
3. Use $(\boldsymbol{x},\ \tilde{\boldsymbol{x}})$ as a training example for estimating the autoencoder distribution $p_{\text{recoconstruct}}(\boldsymbol{x}|\ \tilde{\boldsymbol{x}}\ ) = p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$ with $\boldsymbol{h}$ the output of encoder $f(\ \tilde{\boldsymbol{x}}\ )$ and p_{decoder} typically defined by a decoder $g(\boldsymbol{h})$

- DAE performs SGD on the expectation $E_{\ \tilde{\boldsymbol{x}}\ \sim \hat{p}\text{data}(\mathbf{x})} \log p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h}=\boldsymbol{f}(\tilde{\boldsymbol{x}}\ ))$

DAE for MNIST data

Python/Theano

```
import theano.tensor as T
from opendeep.models.model import Model
from opendeep.utils.nnet import get_weights_uniform, get_bias
 from opendeep.utils.noise import salt_and_pepper
 from opendeep.utils.activation import tanh, sigmoid
 from opendeep.utils.cost import binary_crossentropy
# create our class initialization!
class DenoisingAutoencoder(Model):
 """
A denoising autoencoder will corrupt an input (add noise) and try to reconstruct it.
 """
def __init__(self):
        # Define some model hyperparameters to work with MNIST images!
        input_size = 28*28 # dimensions of image
        hidden_size = 1000 # number of hidden units - generally bigger than input size for DAE
        # Now, define the symbolic input to the model (Theano)
        # We use a matrix rather than a vector so that minibatch processing can be done in parallel.
        x = T.fmatrix("X")
        self.inputs = [x]
        # Build the model's parameters - a weight matrix and two bias vectors
        W = get_weights_uniform(shape=(input_size, hidden_size), name="W")
        b0 = get_bias(shape=input_size, name="b0")
        b1 = get_bias(shape=hidden_size, name="b1")
        self.params = [W, b0, b1]
        # Perform the computation for a denoising autoencoder!
        # first, add noise (corrupt) the input
        corrupted_input = salt_and_pepper(input=x, corruption_level=0.4)
        # next, compute the hidden layer given the inputs (the encoding function)
        hiddens = tanh(T.dot(corrupted_input, W) + b1)
        # finally, create the reconstruction from the hidden layer (we tie the weights with W.T)
        reconstruction = sigmoid(T.dot(hiddens, W.T) + b0)
        # the training cost is reconstruction error - with MNIST this is binary cross-entropy
        self.train_cost = binary_crossentropy(output=reconstruction, target=x)
```



Unsupervised Denoising Autoencoder

Left: original test images

Center: corrupted noisy images

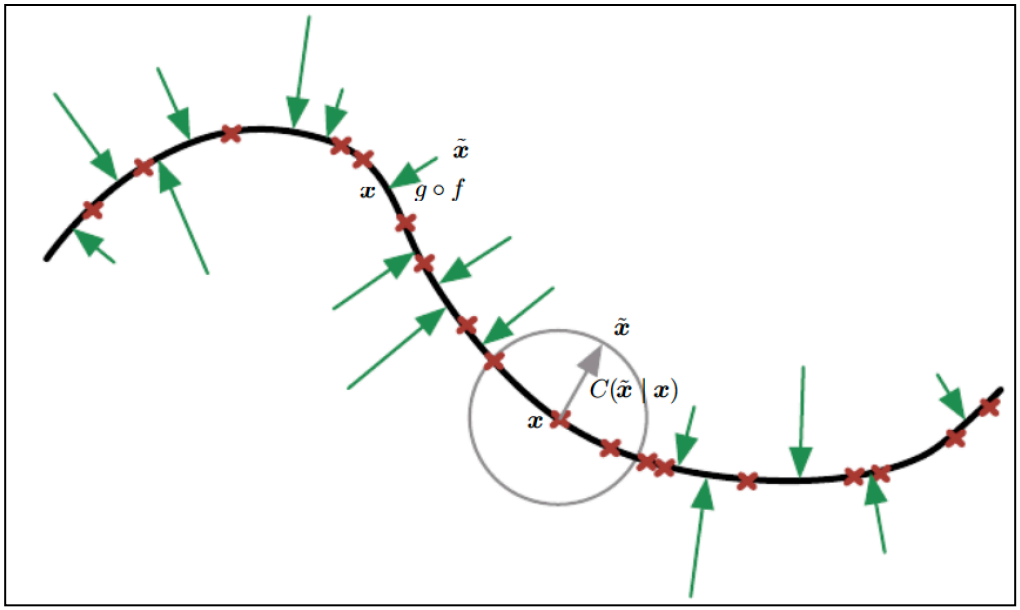
Right: reconstructed images

Estimating the Score

- An autoencoder can be based on encouraging the model to have the same score as the data distribution at every training point $\boldsymbol{x}$
  - The *score* is a particular gradient field is: $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$
  - Learning the *gradient field* of $\log\ p_{\text{data}}$ is one way to learn the structure of p_{data} itself
- Score Matching works by fitting the slope (score) of the model density to the slope of the true underlying density at the data points
- DAE, with conditionally Gaussian $p(\boldsymbol{x}|\boldsymbol{h})$, estimates this score as $(g(f(\boldsymbol{x})\text{-}\boldsymbol{x})$
  - The DAE is trained to minimize $||g(f(\tilde{\boldsymbol{x}})\text{-}\boldsymbol{x})||^2$
- DAE estimates a vector fields as illustrated next

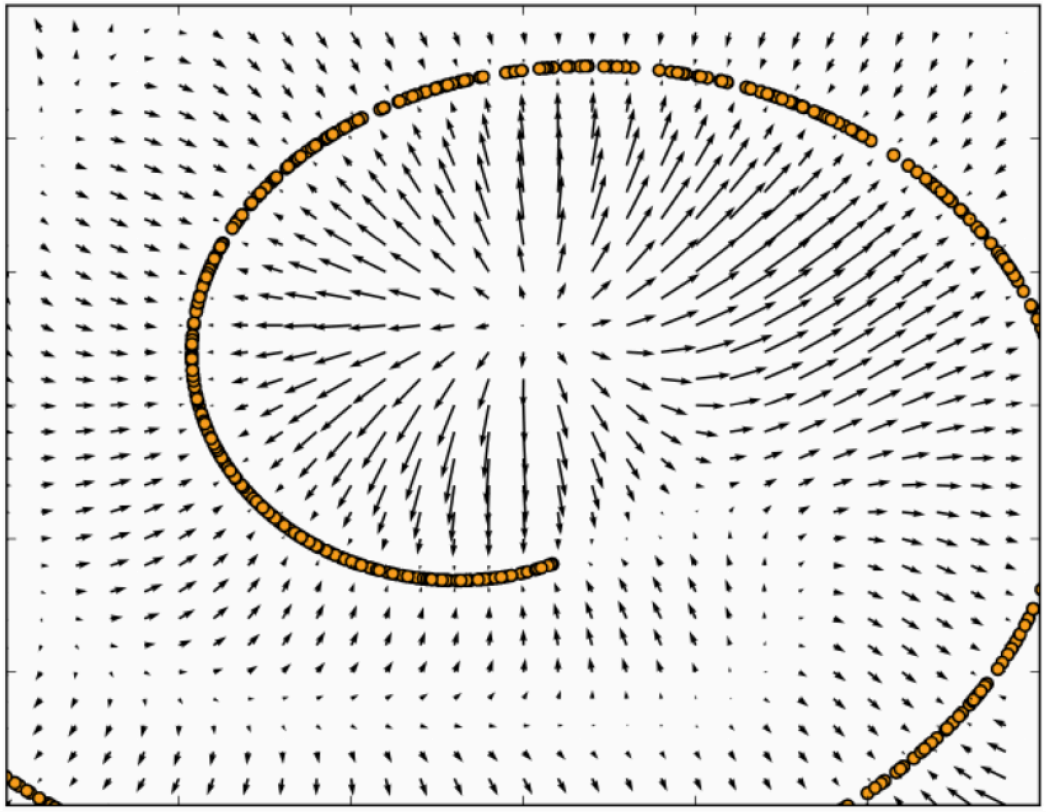
DAE learns a vector field

- Training examples $\boldsymbol{x}$ lie on a low-dimensional manifold
  - Training examples $\boldsymbol{x}$ are red crosses
- Gray circle is equiprobable corruptions
- The vector field $(g(f(\boldsymbol{x})\text{-}\boldsymbol{x})$, indicated by green arrows, estimates the score $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$ which is the slope of the density of data



Vector field learnt by a DAE

- 1-D curved manifold near which the data concentrate
- Each arrow proportional to reconstruction minus input vector of DAE and points towards higher probability
- Where probability is maximum arrows shrink

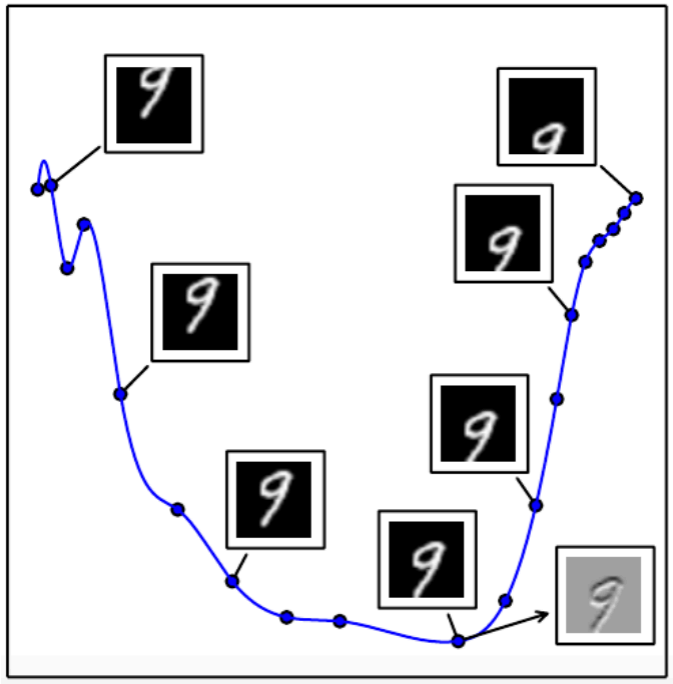


6. Learning manifolds with Autoencoders

- Like many ML algorithms, autoencoders exploit the idea that data concentrates around a low-dimensional manifold
- Some ML algorithms have unusual behavior if input is off of the manifold
- Need to understand characteristics of manifolds
- An important characterization of a manifold is the set of its tangent planes
- At a point $\boldsymbol{x}$ on a d-dimensional manifold, the tangent plane is given by d basis vectors that span the local directions of variation allowed on the manifold
- They specify how $\boldsymbol{x}$ can change infinitesimally while staying on the manifold

Tangent Plane

- A 1-D manifold in 784-D space (MNIST with 784 pixels)
  - Image is translated vertically
  - Figure below is projection into 2-D space using PCA
    - n-dimensional manifold has n-dimensional plane
    - Tangent is oriented parallel to the surface at that point
  - Image shows how this tangent direction appears in image space
    - Gray pixels indicate pixels that do not change as we move along tangent
    - White pixels indicate pixels that brighten, and black those that darken



Autoencoder training: a compromise between two forces

1. Learning a representation $\boldsymbol{h}$ of a training example $\boldsymbol{x}$ such that $\boldsymbol{x}$ can be approximately recovered from h through a decoder
   - The fact that $\boldsymbol{x}$ is drawn from training data is crucial
     - Because it means the the autoencoder need not successfully reconstruct inputs that are not probable
2. Satisfying the regularization penalty
   - Limits the capacity of the autoencoder
   - Or it can be a regularization term added to the reconstruction cost
   - These techniques prefer solutions less sensitive to input

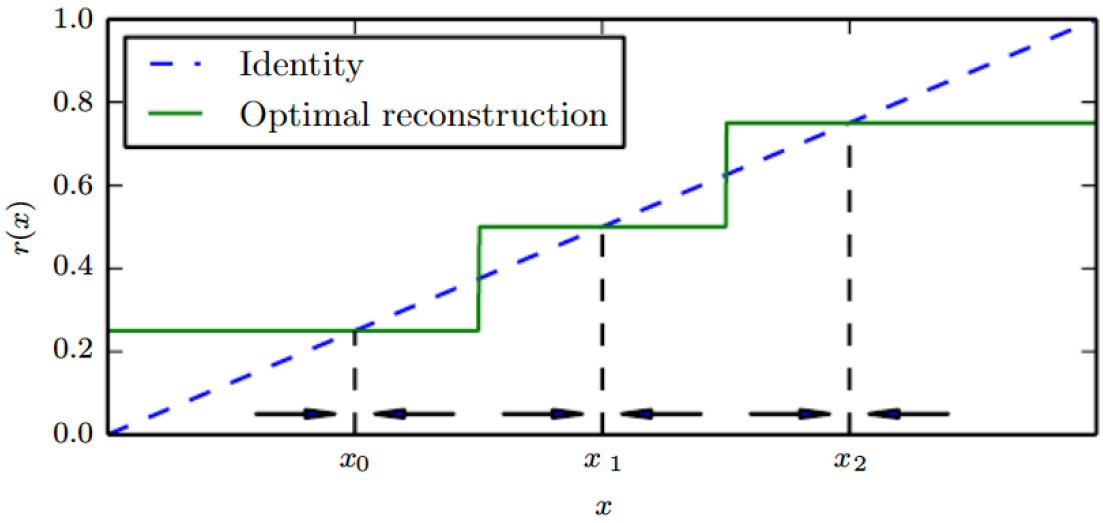
- Together they force the the hidden representation to capture information about the data generating distribution

What the encoder represents

- Encoder can afford to represent only the variations that are needed to reconstruct the training examples
- If the data generating distribution concentrates near a low-dimensional manifold, this yields representations that implicitly captures a local coordinate system for this manifold
  - Only the variations tangential to this manifold around $\boldsymbol{x}$ need to correspond to changes in $\boldsymbol{h} = \boldsymbol{f}(\boldsymbol{x})$
  - Hence the encoder learns a mapping from the input space $\boldsymbol{x}$ to a representation space
    - A mapping that is only sensitive to changes along the manifold directions
    - But that is insensitive to changes orthogonal to the manifold

Invariant reconstruction function

- 1-D case: by making the reconstruction function insensitive to perturbations of the input around the data points, we cause the autoencoder to recover the manifold structure
  - The manifold structure is a collection of 0-dimensional manifolds
    - Dashed diagonal line: identity function for target of reconstruction
    - Optimal reconstruction function crosses the identity function whenever there is a data point
    - Horizontal arrows at bottom indicate $r(\boldsymbol{x})$-$\boldsymbol{x}$ reconstruction direction vector



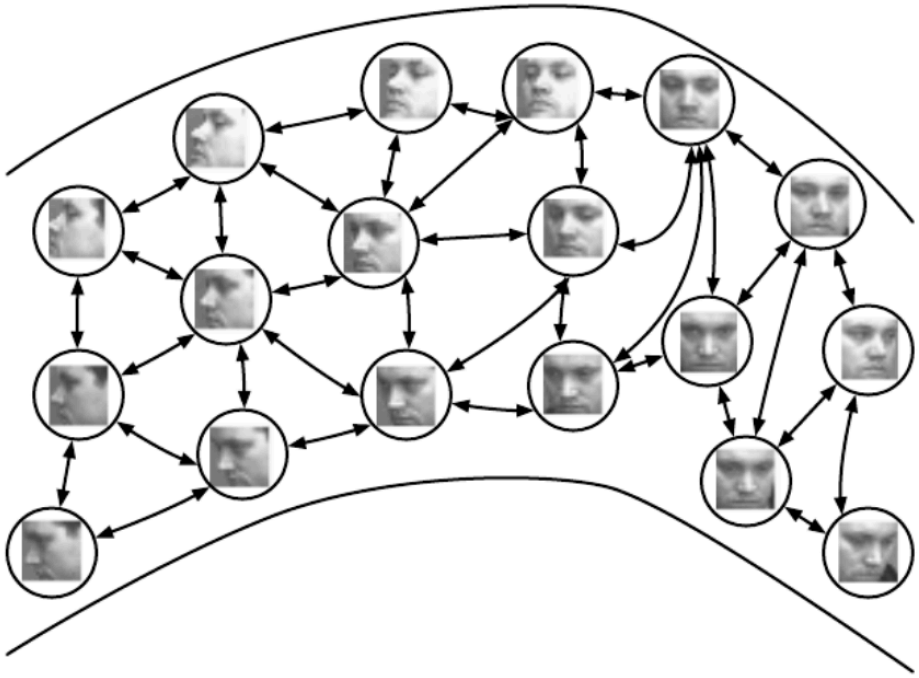
l"- a single data point in 1-D case

Embedding is a point on a manifold

- What is learnt to characterize a manifold is a representation of the data points on or near the manifold
- Such a representation for a particular example is called an *embedding*
- An embedding is a low-dimensional vector
  - With fewer dimensions than than the ambient space of which the manifold is a low-dimensional subset
- Some algorithms (non-parametric manifold algorithms) directly learn an embedding for each training example
- Others learn a more general mapping: a representation function that maps any point in ambient space to its embedding

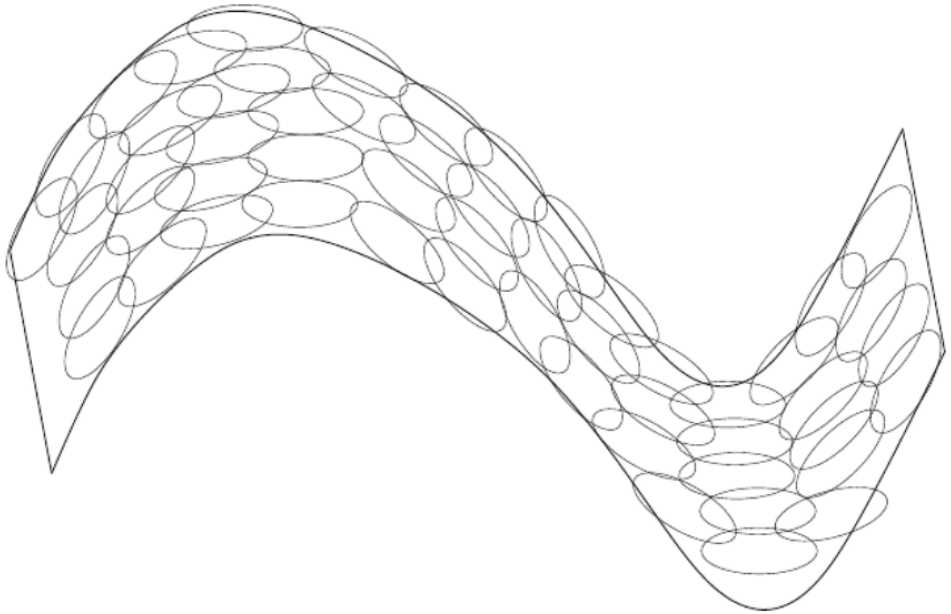
Nonparametric manifold learning

- Nearest-neighbor graph in which nodes represent training examples
- Edges indicate nearest neighbor relationships
- As long as no of examples is large to cover curvature and twists of the manifold they work well



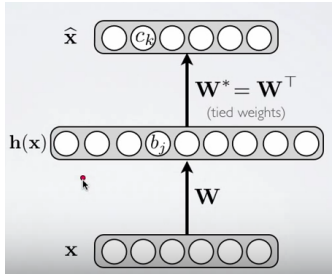
Tiling a manifold

- A global coordinate system can be obtained by solving a linear system
- A manifold can be tiled by a large no of locally linear Gaussian-like patches (or pancakes, because the Gaussians are flat in the tangent directions)

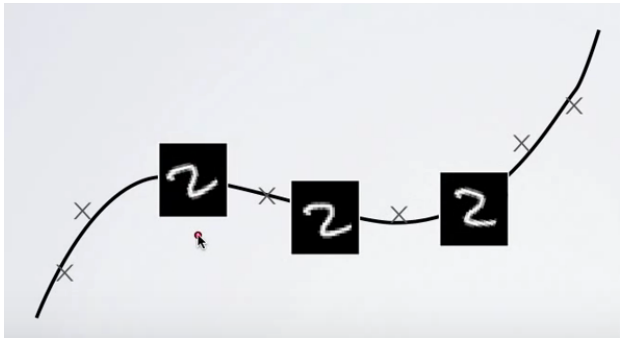


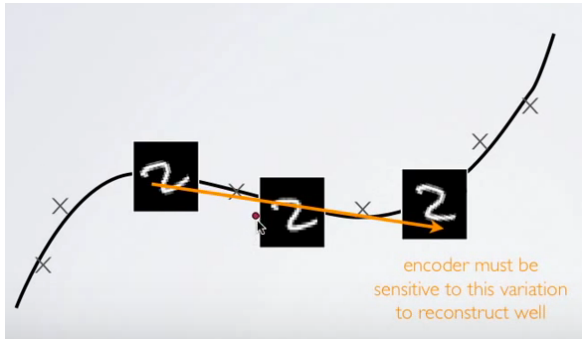
Overcomplete and Contractive Autoencoder

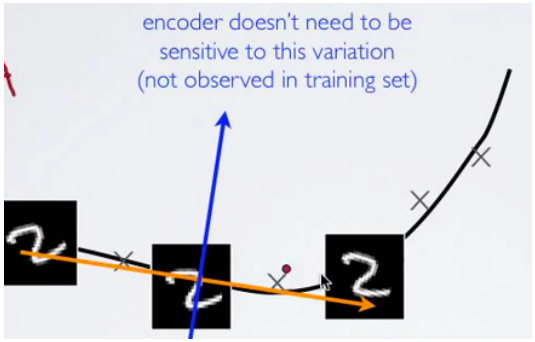
- Overcomplete



- Contractive
  - Method to avoid uninteresting solutions
  - Add an explicit term in the loss that penalizes that solution
  - We wish to extract features that only reflect variations observed in the training set
  - We would like to be invariant to other variations

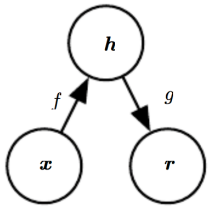






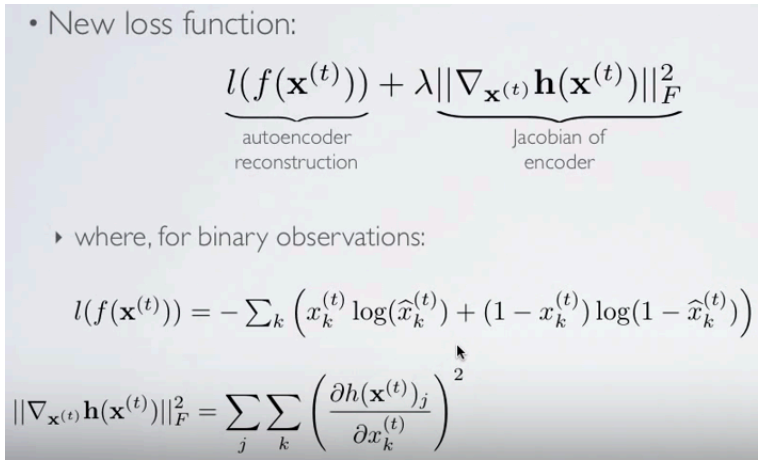
Contractive Autoencoder Loss Function

- Contractive autoencoder has an explicit regularizer on $\boldsymbol{h}=f(\boldsymbol{x})$, encouraging the derivatives of f to be as small as possible:



$$\Omega(\boldsymbol{h}) = \lambda \left\| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right\|_F^2$$

  - Where $L(f(\boldsymbol{x})) + \Omega(\boldsymbol{h})$
    - Penalty $\Omega(\boldsymbol{h})$ is the squared Frobenius norm (sum of squared elements) of the Jacobian matrix of partial derivatives associated with encoder function



Difference between DAE and CAE

- Denoising Autoencoders make the reconstruction function $\boldsymbol{r}=g(f(\boldsymbol{x}))$ resist small but finite-sized perturbations of the input
  - DAE minimizes $L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}})))$
- Contractive Autoencoders make the feature extraction function resist inifinitesimal perturbations of the input
  - CAE minimizes $L(f(\boldsymbol{x})) + \Omega(\boldsymbol{h})$ where

$$\Omega(\boldsymbol{h}) = \lambda \left\| \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right\|_F^2$$

  - It uses a Jacobian-based contractive penalty to pretrain features $f(\boldsymbol{x})$ for use with a classifier, with $L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h}, \boldsymbol{x})$

$$\Omega(\boldsymbol{h}, \boldsymbol{x}) = \lambda \sum_i \left\| \nabla_{\boldsymbol{x}} h_i \right\|^2$$

Contractive autoencoder warps space

- The name contractive arises from the way the CAE warps space
- Because CAE is trained to resist perturbations of its input, it is encouraged to map a neighborhood of input points to a smaller neighborhood of output points
  - We can think of this as contracting the input neighborhood to a smaller output neighborhood