

Autoencoders

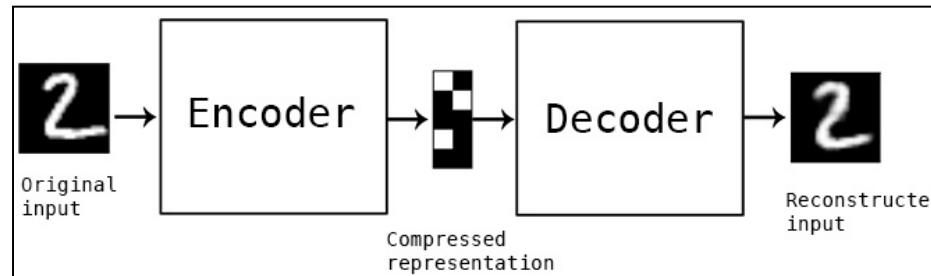
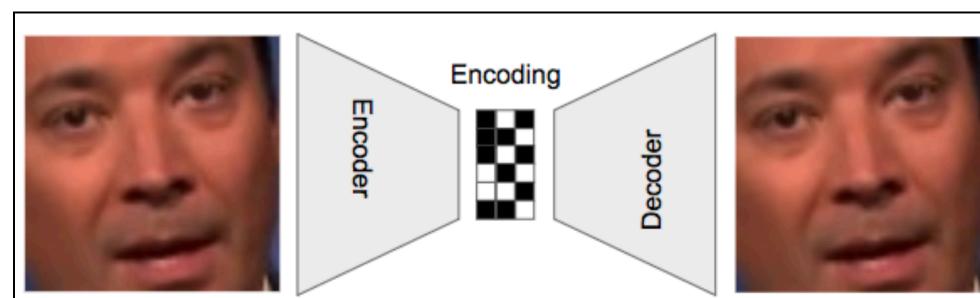
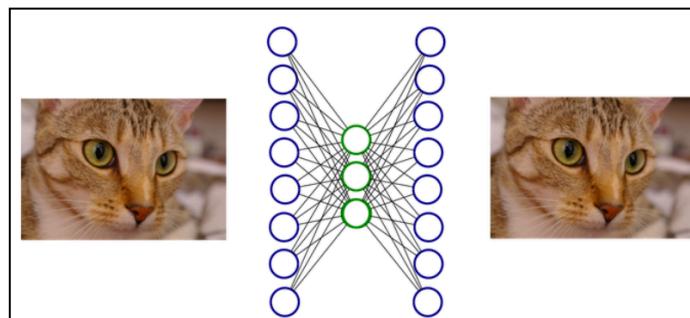
Sargur Srihari
srihari@buffalo.edu

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

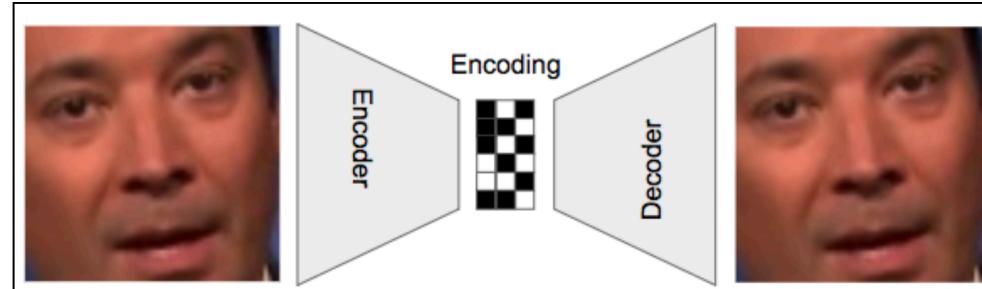
What is an Autoencoder?

- A neural network trained using unsupervised learning
 - Trained to copy its input to its output
 - Learns an embedding



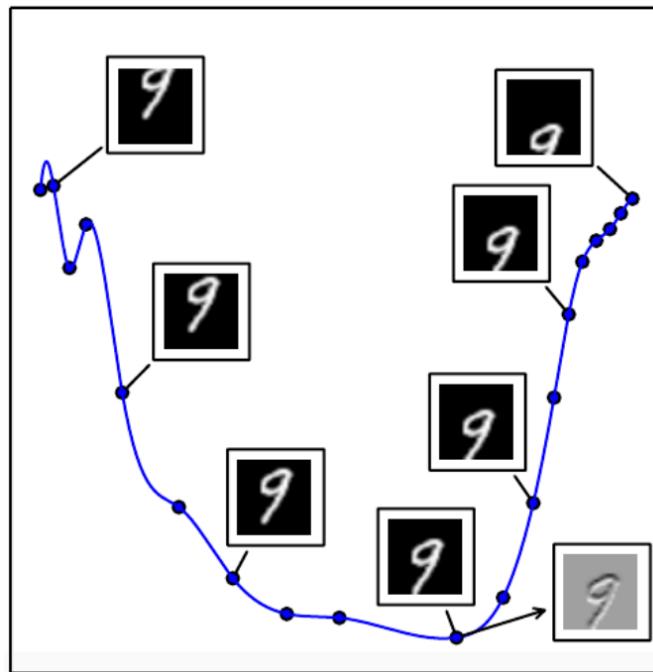
Embedding is a point on a manifold

- An embedding is a low-dimensional vector
 - With fewer dimensions than the ambient space of which the manifold is a low-dimensional subset
- Embedding Algorithm
 - Maps any point in ambient space x to its embedding h
 - Embeddings of related inputs form a manifold

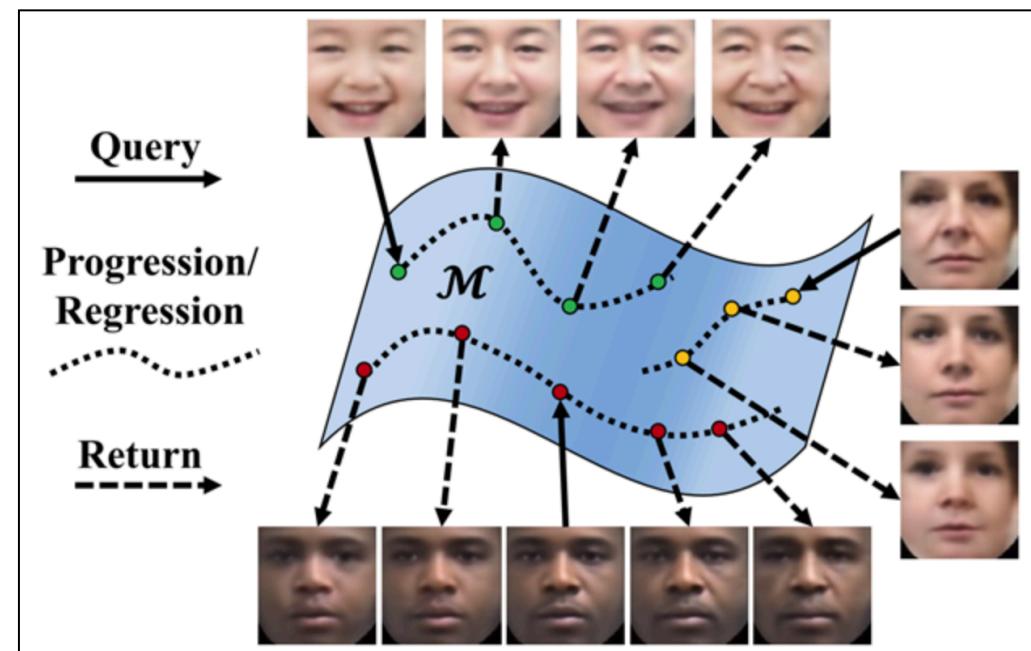


A manifold in ambient space

Embedding: map x to lower dimensional h



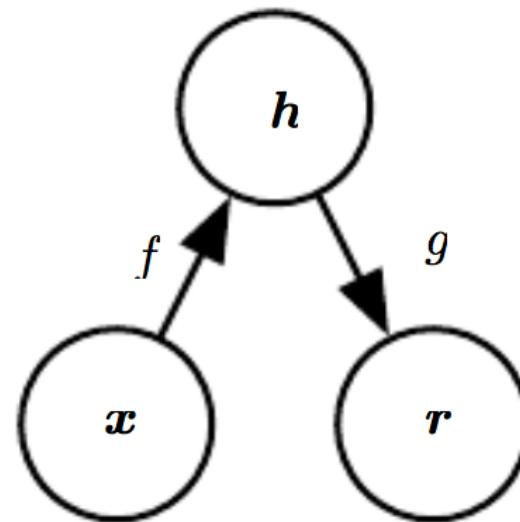
1-D manifold in 2-D space
Derived from $28 \times 28 = 784$ space



Age Progression/Regression by Conditional Adversarial Autoencoder (CAAE)
Github: <https://github.com/ZZUTK/Face-Aging-CAAE>

General structure of an autoencoder

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



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

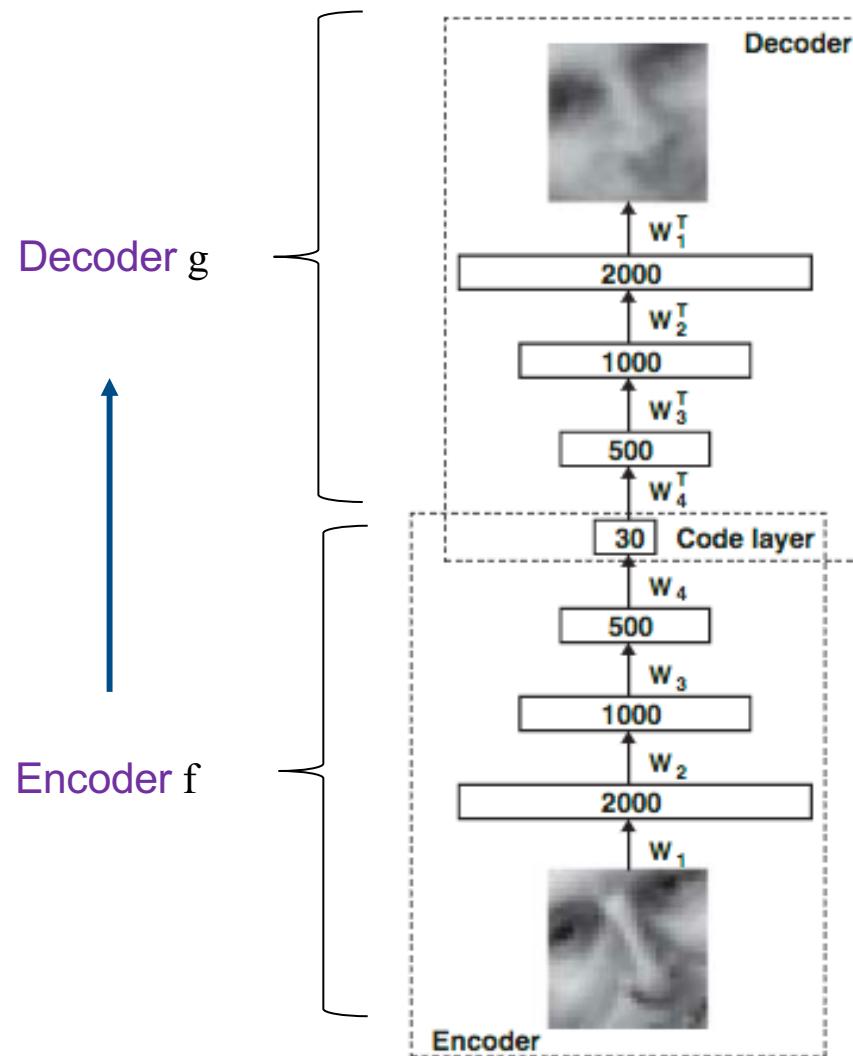
What does an Autoencoder Learn?

- Learning $g(f(x))=x$ everywhere is not useful
- Autoencoders are designed to be unable to copy perfectly
 - Restricted to copy only approximately
- Autoencoders learn useful properties of the data
 - Being forced to prioritize which aspects of input should be copied
- Can learn stochastic mappings
 - Go beyond deterministic functions to mappings $p_{\text{encoder}}(\mathbf{h}|\mathbf{x})$ and $p_{\text{decoder}}(\mathbf{x}|\mathbf{h})$

Autoencoder History

- Part of neural network landscape for decades
 - Used for dimensionality reduction and feature learning
- Theoretical connection to latent variable models
 - Have brought them into forefront of generative models
 - Variational Autoencoders

An autoencoder architecture



Weights W are learnt using:

1. Training samples, and
2. a loss function

as discussed next

Two Autoencoder Training Methods

1. 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 have no interest in decoder output
- We hope h takes on useful properties
- Undercomplete autoencoder
 - Constrain h to have lower dimension than x
 - Force it to capture most salient features of training data

Autoencoder with linear decoder +MSE is PCA

- Learning process is that of minimizing a loss function

$$L(x, g(f(x)))$$

- where L is a loss function penalizing $g(f(x))$ for being dissimilar from 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 high capacity is not desirable as seen next

Autoencoder training using a loss function

- Encoder f and decoder g

$$f : X \rightarrow h$$

$$g : h \rightarrow X$$

$$\arg \min_{f,g} \|X - (f \circ g)X\|^2$$

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

$$h = \sigma_1(Wx + b)$$

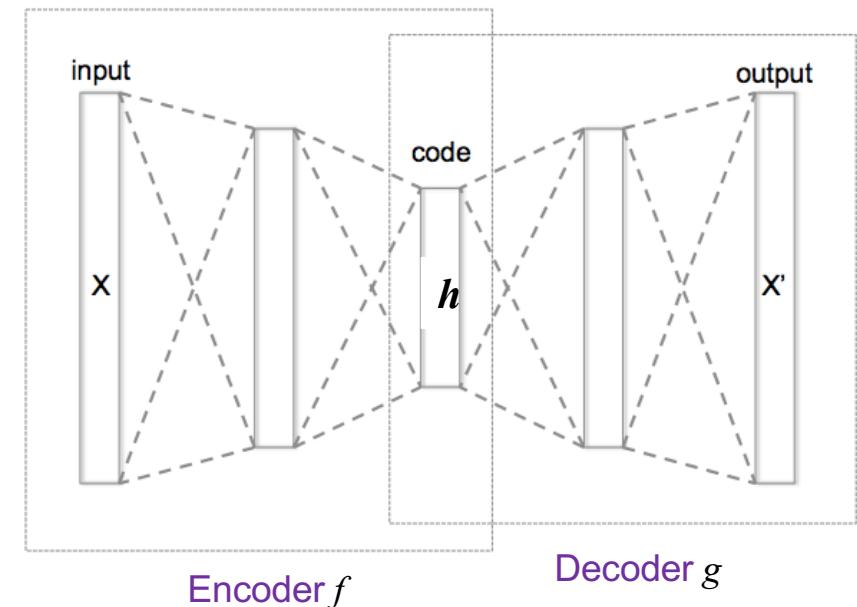
$$x' = \sigma_2(W'h + b') \quad \sigma \text{ is an element-wise activation function such as sigmoid or Relu}$$

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

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

Provides a compressed representation of the input x

Autoencoder with 3 fully connected hidden layers



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 $\mathbf{x}^{(i)}$ 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 h has dimension equal to input x
 3. *Overcomplete* case: where hidden code h has dimension greater than input x
 - Even a linear encoder/decoder can learn to copy input to output without learning anything useful about data distribution

Right Autoencoder Design: Use regularization

- Ideally, choose code size (dimension of h) small and capacity of encoder f and decoder g based on complexity of distribution modeled
- *Regularized autoencoders* provide the ability to do so
 - Rather than limiting model capacity by keeping encoder/decoder shallow and code size small
 - They use a loss function that encourages the model to have properties other than copy its input to output

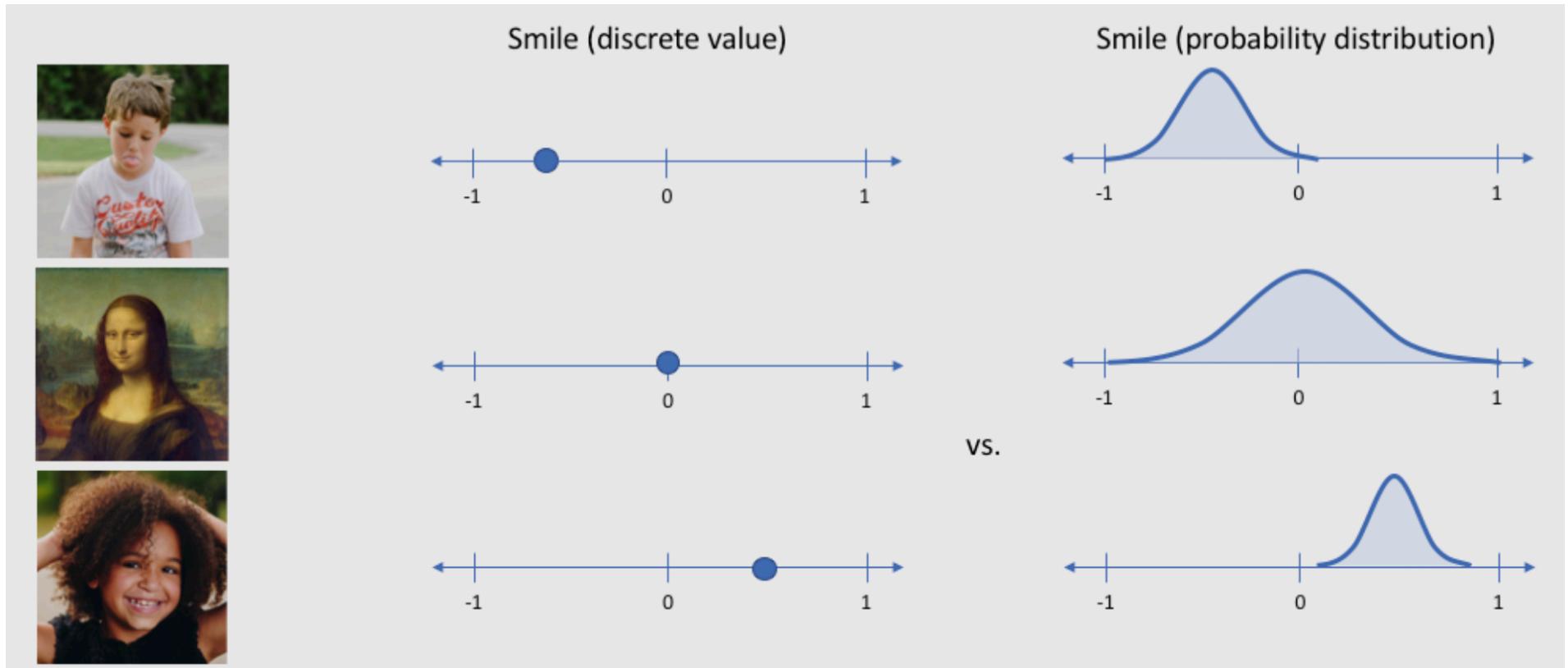
2. Regularized Autoencoder Properties

- Regularized AEs have properties beyond copying input to output:
 - 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

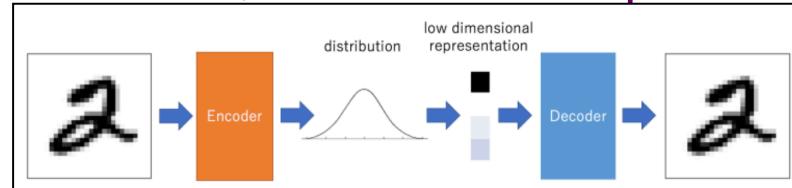
- Beyond regularized autoencoders
- Generative models with latent variables and an inference procedure (for computing latent representations given input) can be viewed as a particular form of autoencoder
- Generative modeling approaches which emphasize connection with autoencoders are descendants of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

Latent variables treated as distributions



Variational Autoencoder

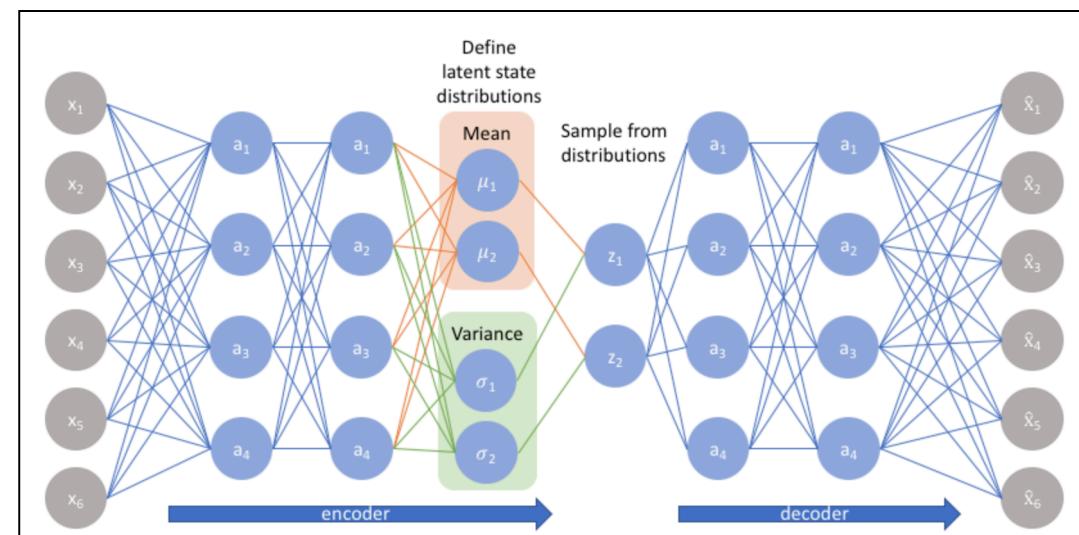
- VAE is a *generative model*
 - able to generate samples that look like samples from training data
 - With MNIST, these fake samples would be synthetic images of digits



- Due to random variable between input-output it cannot be trained using backprop
 - Instead, backprop proceeds through parameters of latent distribution
 - Called reparameterization trick

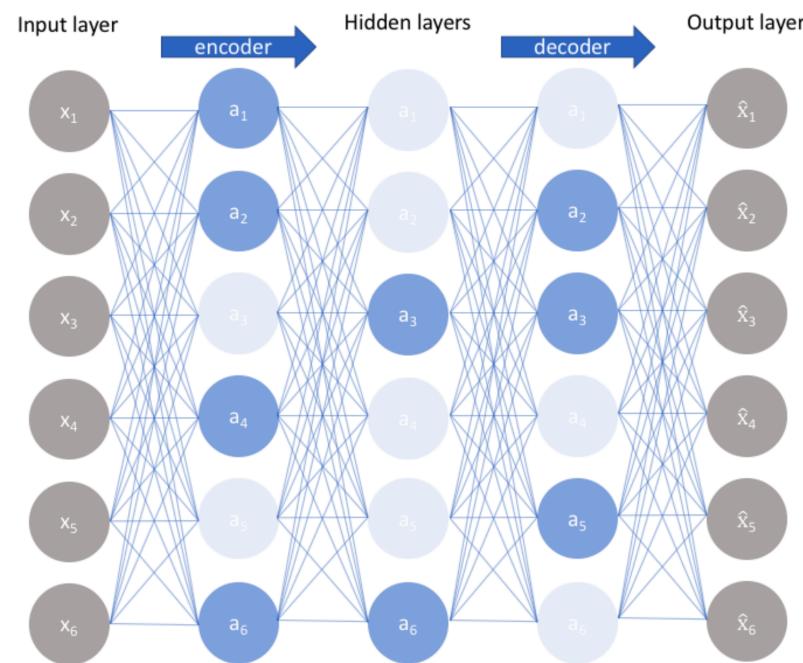
$$N(\mu, \Sigma) = \mu + \Sigma N(0, I)$$

Where Σ is diagonal



Sparse Autoencoder

Only a few nodes are encouraged to activate when a single sample is fed into the network



Fewer nodes activating while still keeping its performance would guarantee that the autoencoder is actually learning **latent representations** instead of redundant information in our input data

Sparse Autoencoder Loss Function

- A sparse autoencoder is an autoencoder whose
 - Training criterion includes a sparsity penalty $\Omega(\mathbf{h})$ on the code layer \mathbf{h} in addition to the reconstruction error:
$$L(\mathbf{x}, g(f(\mathbf{x}))) + \Omega(\mathbf{h})$$
 - where $g(\mathbf{h})$ is the decoder output and typically we have $\mathbf{h} = f(\mathbf{x})$
- 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 $\Omega(\mathbf{h})$ 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 $p(\theta|x)$, which is equivalent to maximizing $\log p(x|\theta) + \log p(\theta)$
 - 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 x and h
- Explicit joint distribution is $p_{\text{model}}(x, h) = p_{\text{model}}(h) p_{\text{model}}(x|h)$
 - where $p_{\text{model}}(h)$ is model's prior distribution over latent variables
 - Different from $p(\theta)$ being distribution of parameters
- The log-likelihood can be decomposed as $\log p_{\text{model}}(x, h) = \log \sum_h p_{\text{model}}(h, x)$
- Autoencoder approximates the sum with a point estimate for just one highly likely value of h , the output of a parametric encoder
 - With a chosen h we are maximizing $\log p_{\text{model}}(x, h) = \log p_{\text{model}}(h) + \log p_{\text{model}}(x|h)$

Sparsity-inducing Priors

- The $\log p_{\text{model}}(\mathbf{h})$ 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
- $$-\log p_{\text{model}}(\mathbf{h}) = \sum_i \left(\lambda |h_i| - \log \frac{\lambda}{2} \right) = \Omega(\mathbf{h}) + \text{const}$$
- where $\Omega(\mathbf{h}) = \lambda \sum_i h_i$
- where the constant term depends only on λ and not on \mathbf{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 $L(x, g(f(x)))$
 - where L is a loss function penalizing $g(f(x))$ for being dissimilar from x , such as L^2 norm of difference: mean squared error
- A DAE minimizes $L(x, g(f(\tilde{x})))$
 - where \tilde{x} is a copy of 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 $p_{\text{data}}(x)$
- 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

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

- But with a different form of Ω

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

- Forces the model to learn a function that does not change much when \mathbf{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

4. Stochastic Encoders and Decoders

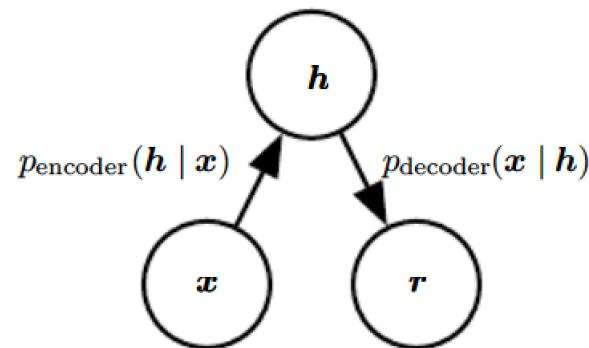
- General strategy for designing the output units and loss function of a feedforward network is to
 - Define the output distribution $p(y|x)$
 - Minimize the negative log-likelihood $-\log p(y|x)$
 - In this setting y is a vector of targets such as class labels
- In an autoencoder x is the target as 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 \mathbf{h} , we may think of the decoder as providing a conditional distribution $p_{\text{decoder}}(\mathbf{x}|\mathbf{h})$
- We train the autoencoder by minimizing $-\log p_{\text{decoder}}(\mathbf{x}|\mathbf{h})$
- The exact form of this loss function will change depending on the form of $p_{\text{decoder}}(\mathbf{x}|\mathbf{h})$
- As with feedforward networks we use linear output units to parameterize the mean of the Gaussian distribution if \mathbf{x} is real
 - In this case negative log-likelihood is the mean-squared error
- With binary \mathbf{x} correspond to a Bernoulli with parameters given by a sigmoid
- Discrete \mathbf{x} values correspond to a softmax
- The output variables are treated as being conditionally independent given \mathbf{h}

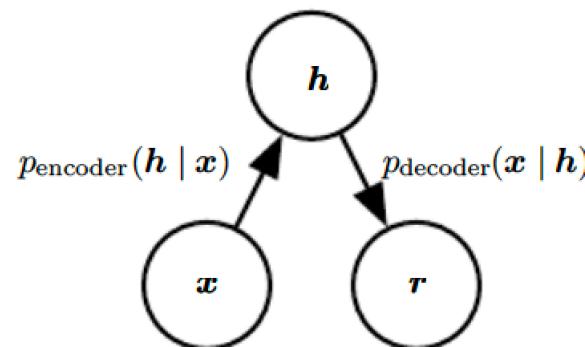
Stochastic encoder

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



Structure of stochastic autoencoder

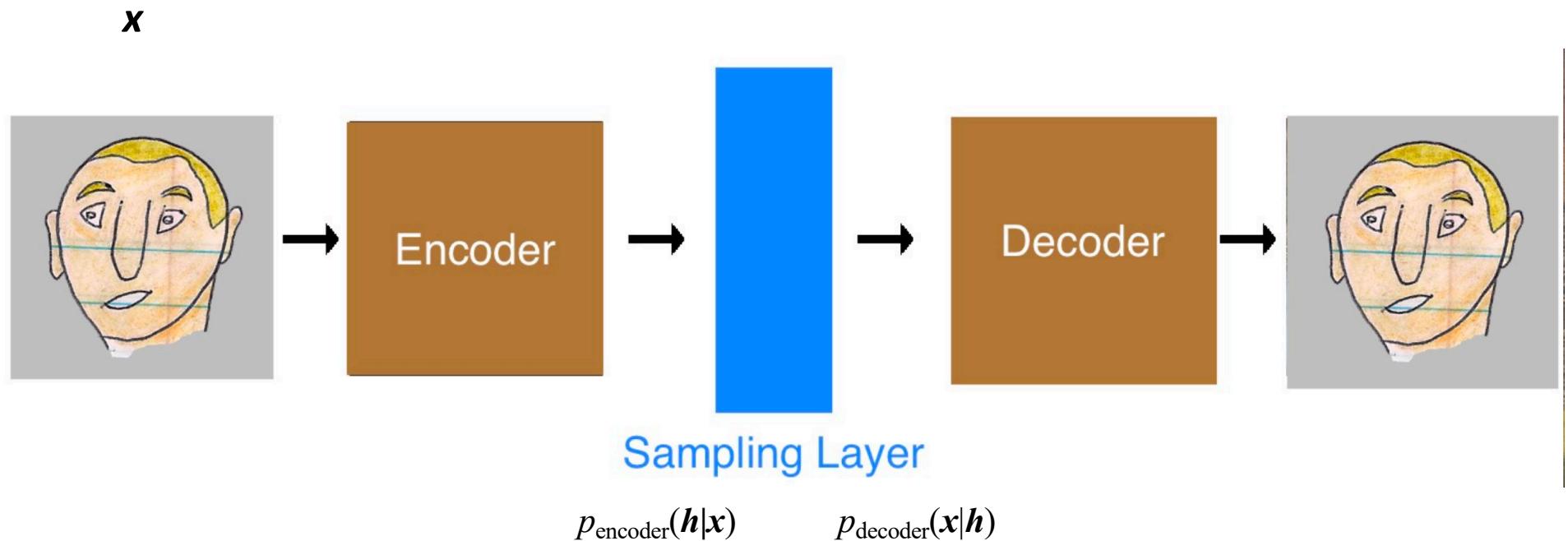
- Both the encoder and decoder are not simple functions but involve a distribution
- The output is sampled from a distribution $p_{\text{encoder}}(\mathbf{h}|\mathbf{x})$ for the encoder and $p_{\text{decoder}}(\mathbf{x}|\mathbf{h})$ for the decoder



Relationship to joint distribution

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

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



Ex: Sampling $p(x|h)$: 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

