Autoencoders

Unit-IV

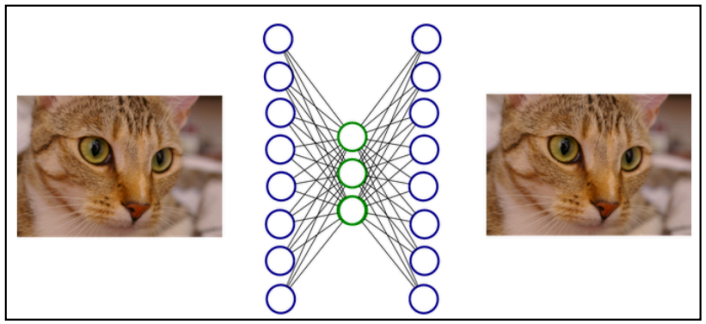
Topics in Autoencoders

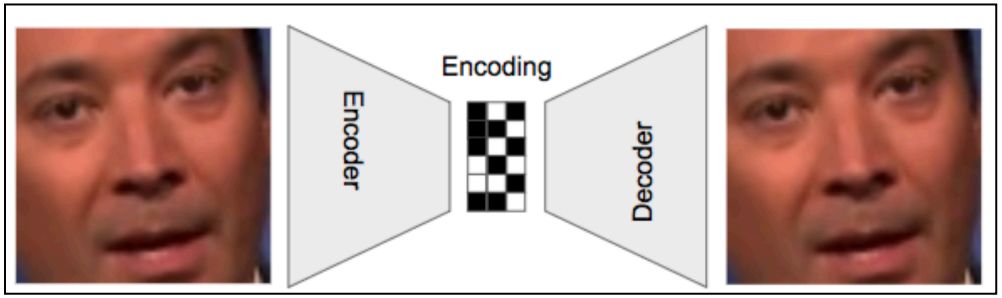
- What is an autoencoder?

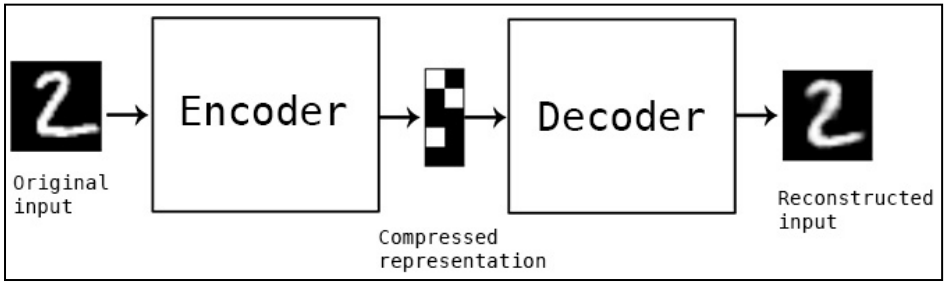
1. Undercomplete Autoencoders
2. Regularized Autoencoders
3. Representational Power, Layout Size and Depth
4. Stochastic Encoders and Decoders
5. Denoising Autoencoders
6. Learning Manifolds and Autoencoders
7. Contractive Autoencoders
8. Predictive Sparse Decomposition
9. Applications of Autoencoders

What is an Autoencoder?

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

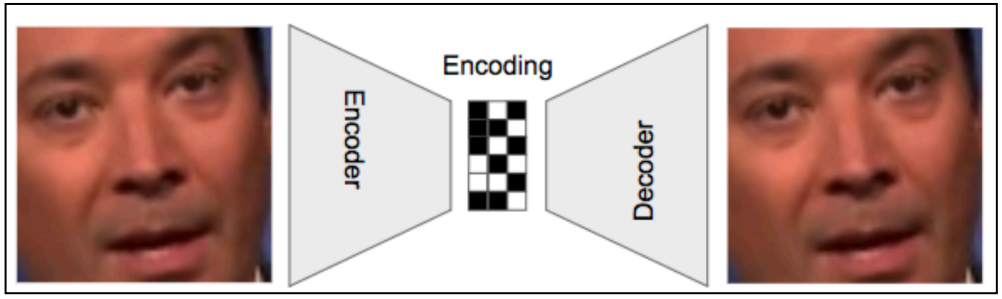






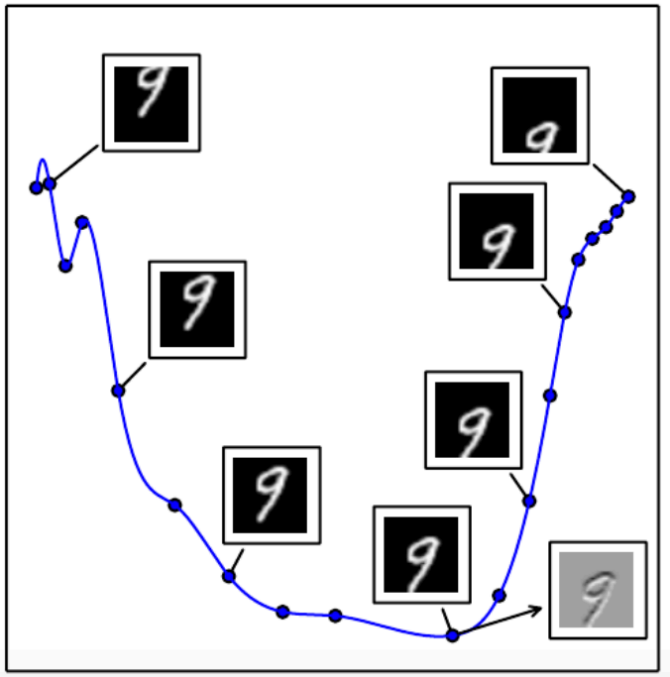
Embedding is a point on a manifold

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

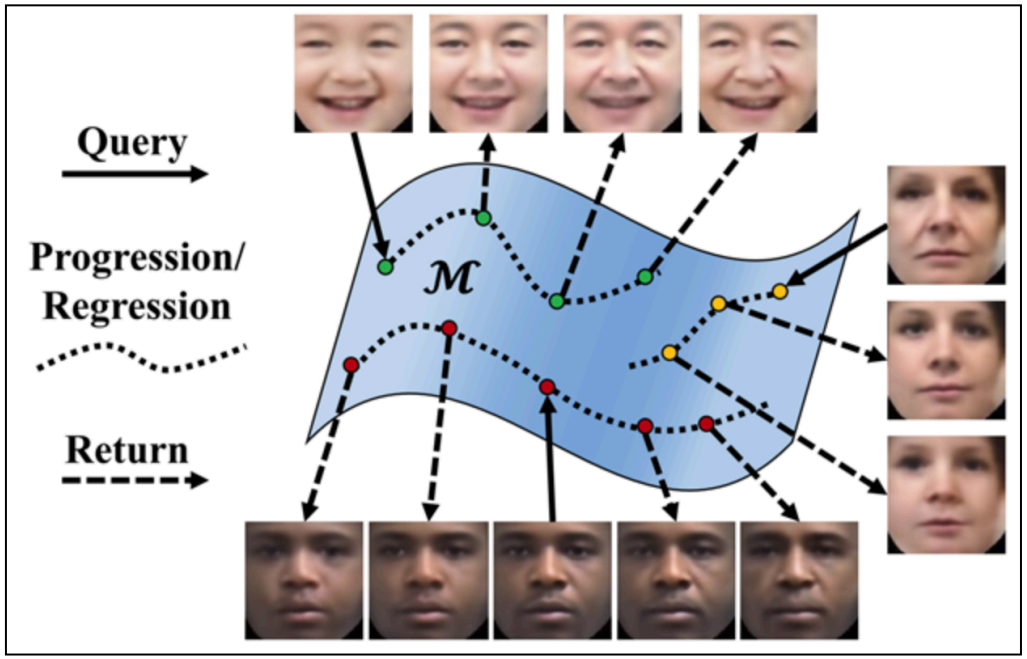


A manifold in ambient space

Embedding: map x to lower dimensional h



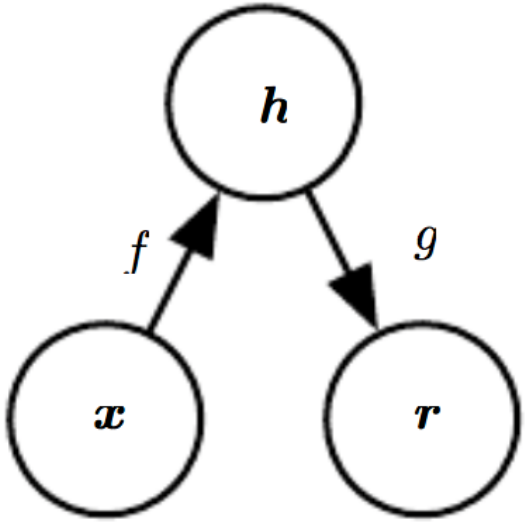
1-D manifold in 2-D space
Derived from 28x28=784 space



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

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 $\boldsymbol{r}=g(\boldsymbol{h})$



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

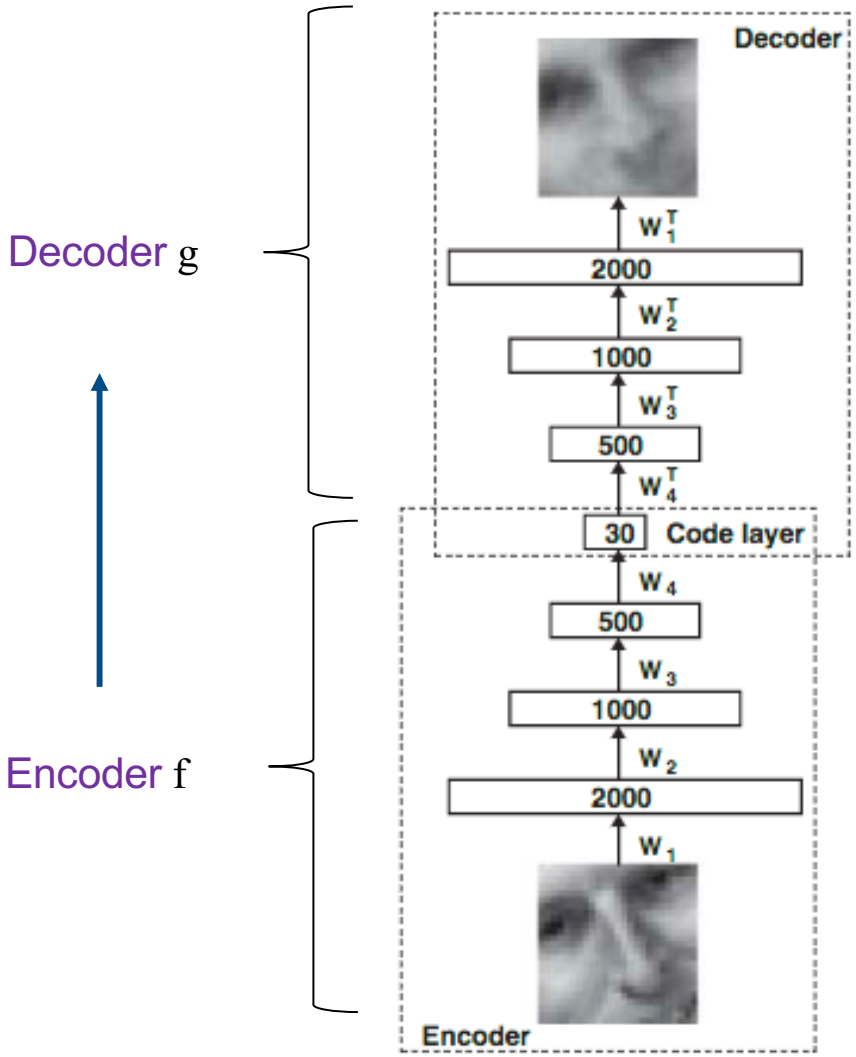
What does an Autoencoder Learn?

- Learning $g(f(\boldsymbol{x}))=\boldsymbol{x}$ 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 $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$ and $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$

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 $\boldsymbol{h}$ takes on useful properties
- Undercomplete autoencoder
  - Constrain $\boldsymbol{h}$ to have lower dimension than $\boldsymbol{x}$
  - 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

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

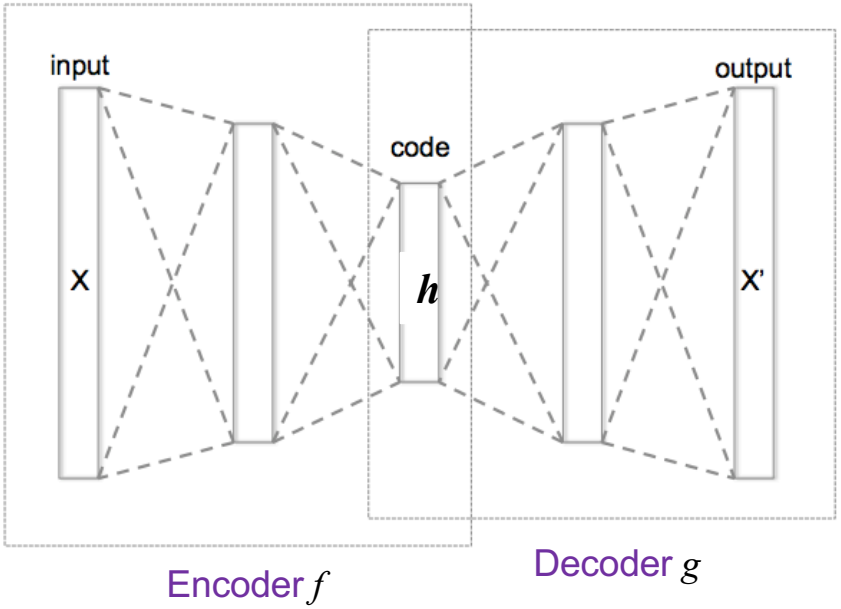
  - where L is a loss function penalizing $g(f(\boldsymbol{x}))$ for being dissimilar from $\boldsymbol{x}$
    - such as L^2 norm of difference: mean squared error
  - When the decoder g is linear and L is the mean squared error, an undercomplete autoencoder learns to span the same subspace as PCA
    - 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 high capacity is not desirable as seen 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 $\boldsymbol{x}$

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

Right Autoencoder Design: Use regularization

- Ideally, choose code size (dimension of $\boldsymbol{h}$) 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

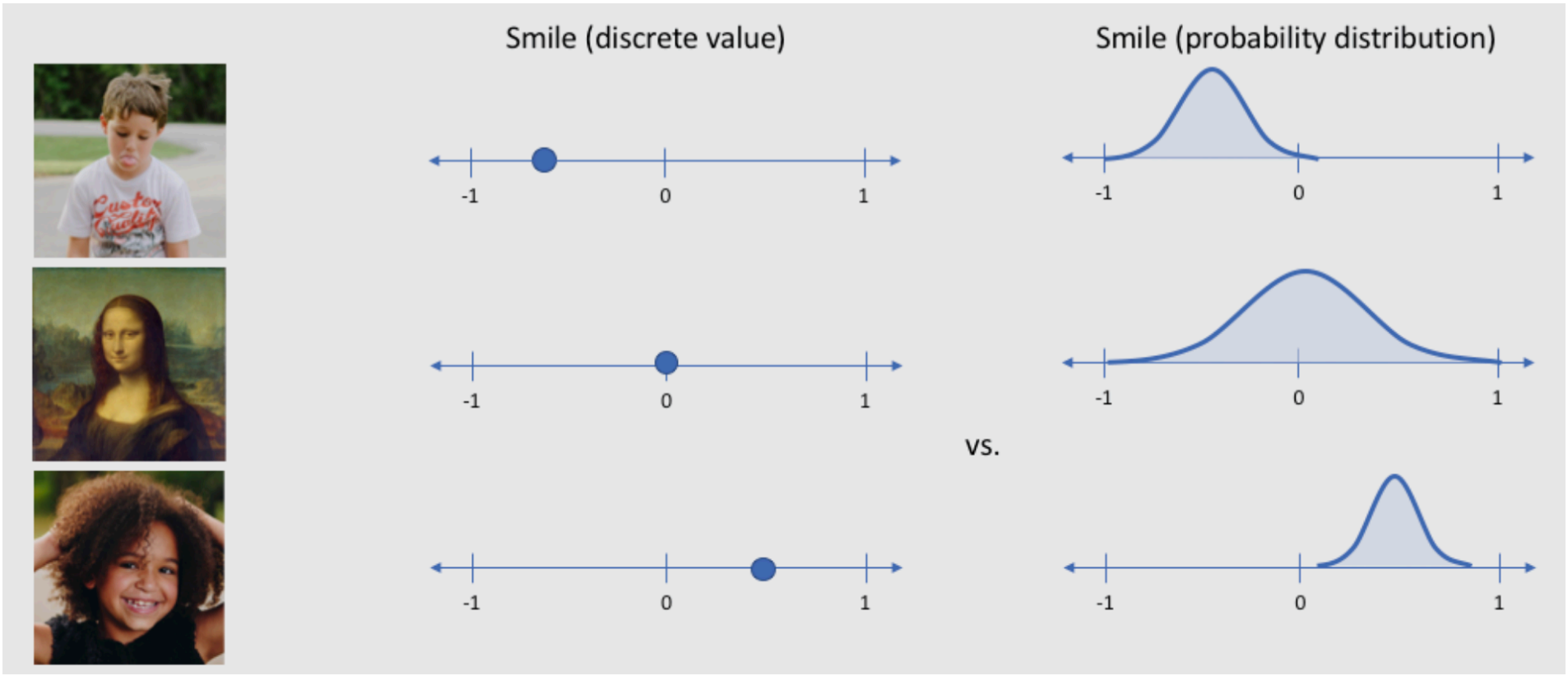
2. Regularized Autoencoder Properties

- 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

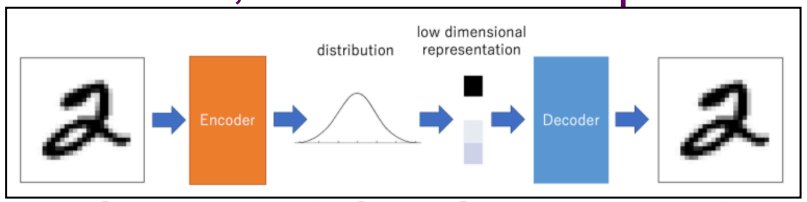
Latent variables treated as distributions



Source: https://www.jeremyjordan.me/variational-autoencoders/

Variational Autoencoder

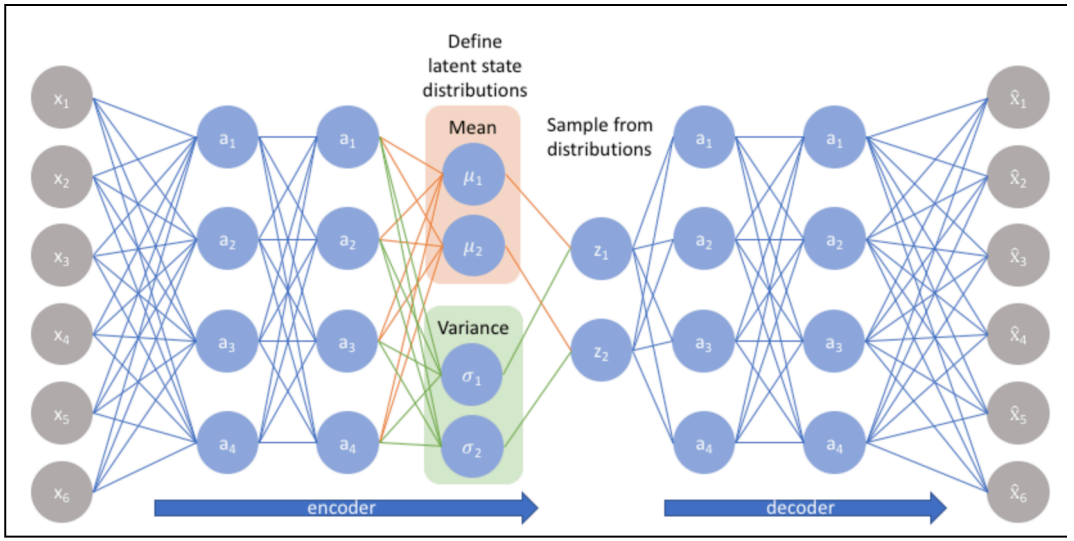
- VAE is a *generative model*
  - able to generate samples that look like samples from training data
    - With MNIST, these fake samples would be synthetic images of digits



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

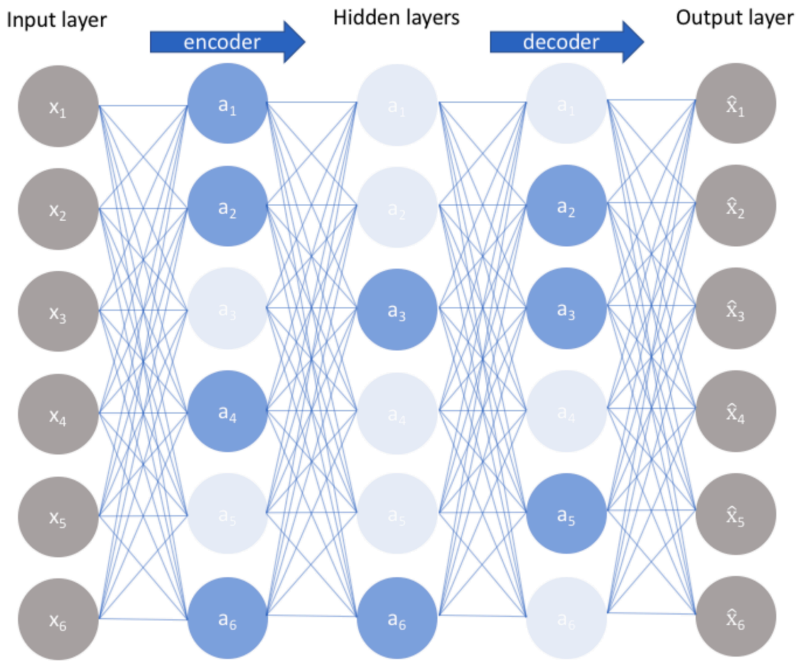
$$N(\mu,\Sigma) = \mu + \Sigma\, N(0, I)$$

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 $\Omega(\boldsymbol{h})$ 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

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 Ω

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

- 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

4. Stochastic Encoders and Decoders

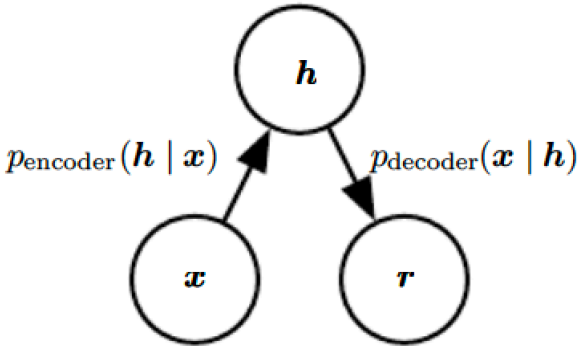
- General strategy for designing the output units and loss function of a feedforward network is to
  - Define the output distribution $p(\boldsymbol{y}|\boldsymbol{x})$
  - Minimize the negative log-likelihood $-\log p(\boldsymbol{y}|\boldsymbol{x})$
  - In this setting $\boldsymbol{y}$ is a vector of 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 $\boldsymbol{h}$

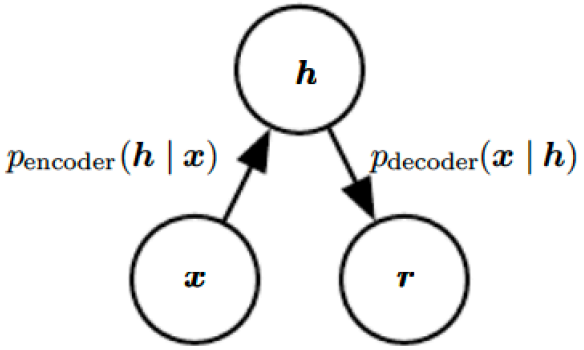
Stochastic encoder

- 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

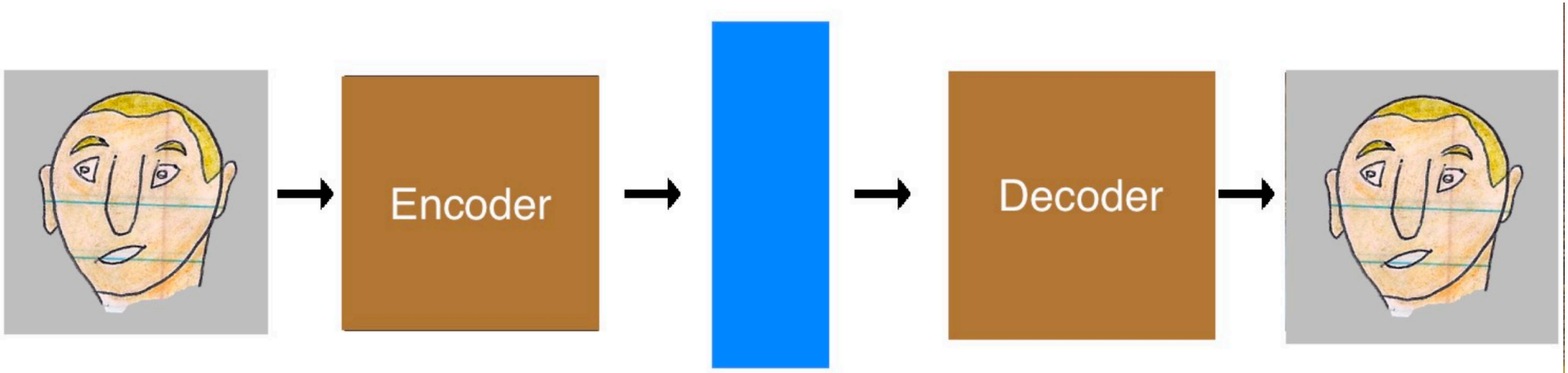


Relationship to joint distribution

- 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