

Charu C. Aggarwal  
IBM T J Watson Research Center  
Yorktown Heights, NY

# Model Generalization and the Bias-Variance Trade-Off

Neural Networks and Deep Learning, Springer, 2018  
Chapter 4, Section 4.1-4.2

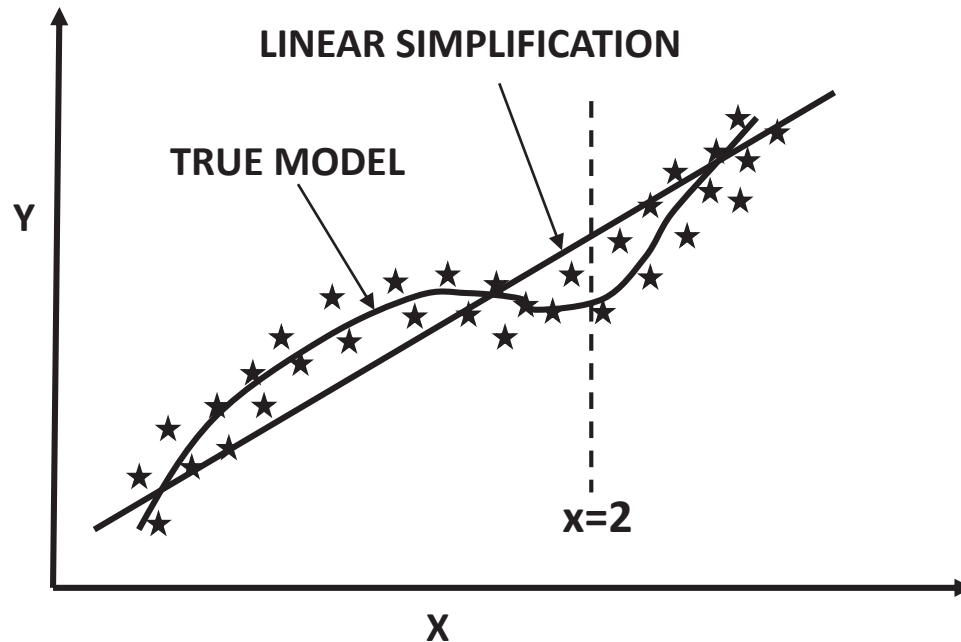
## What is Model Generalization?

- In a machine learning problem, we try to generalize the known dependent variable on seen instances to unseen instances.
  - Unseen  $\Rightarrow$  The model did not see it during training.
  - Given training images with seen labels, try to label an unseen image.
  - Given training emails labeled as spam or nonspam, try to label an unseen email.
- The classification accuracy on instances used to train a model is usually higher than on unseen instances.
  - We only care about the accuracy on unseen data.

## Memorization vs Generalization

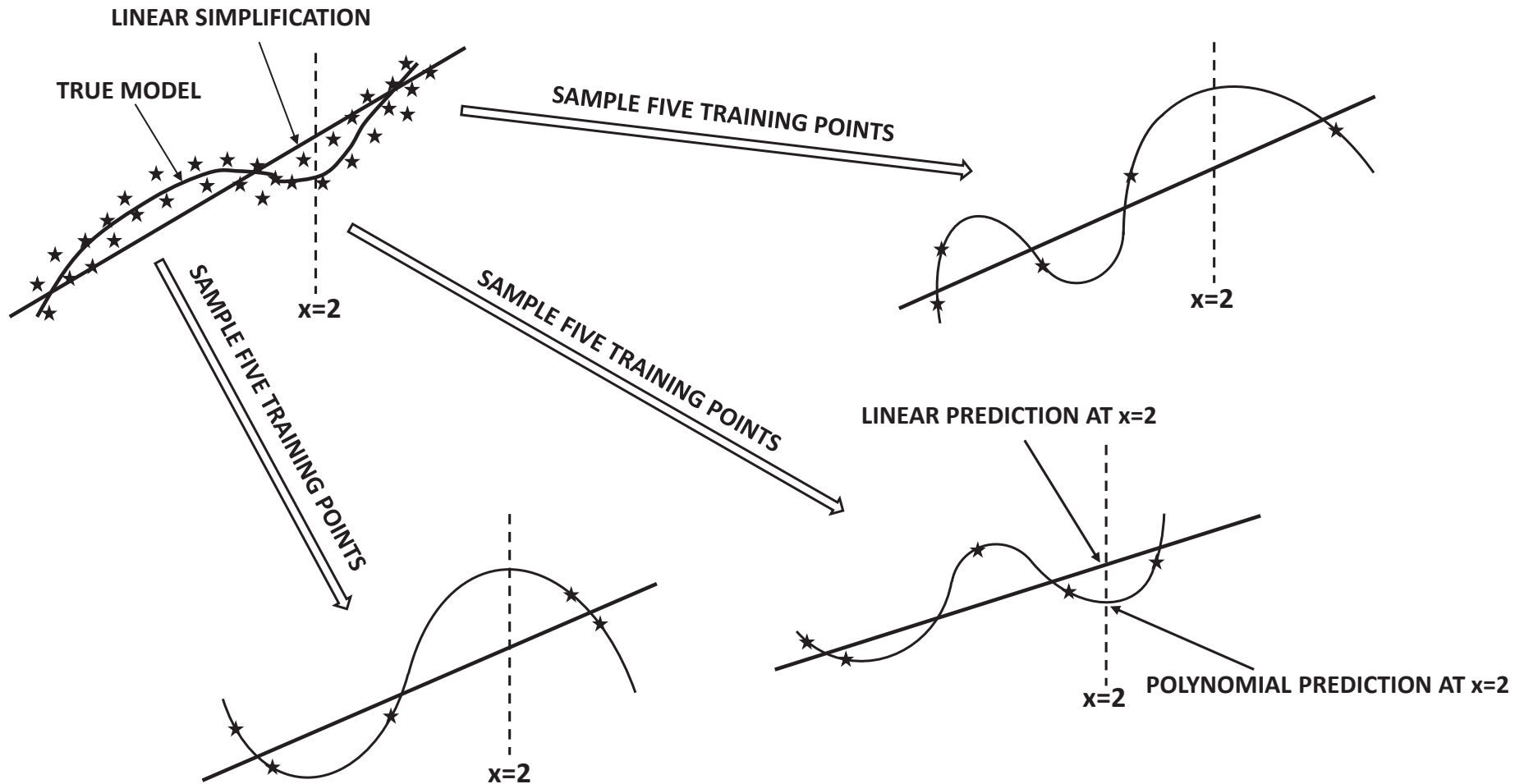
- Why is the accuracy on seen data higher?
  - Trained model remembers some of the irrelevant nuances.
- When is the gap between seen and unseen accuracy likely to be high?
  - When the amount of data is limited.
  - When the model is complex (which has higher *capacity* to remember nuances).
  - The combination of the two is a deadly cocktail.
- A high accuracy gap between the predictions on seen and unseen data is referred to as *overfitting*.

## Example: Predict $y$ from $x$



- **First impression:** Polynomial model such as  $y = w_0 + w_1x + w_2x^2 + w_3x^3 + w_4x^4$  is “better” than linear model  $y = w_0 + w_1x$ .
  - Bias-variance trade-off says: “Not necessarily! How much data do you have?”

# Different Training Data Sets with Five Points



- Zero error on training data but wildly varying predictions of  $x = 2$

## Observations

- The higher-order model is more complex than the linear model and has less *bias*.
  - But it has more parameters.
  - For a small training data set, the learned parameters will be more sensitive to the nuances of that data set.
  - Different training data sets will provide different predictions for  $y$  at a particular  $x$ .
  - This variation is referred to as model *variance*.
- Neural networks are inherently low-bias and high-variance learners  $\Rightarrow$  Need ways of handling complexity.

## Noise Component

- Unlike bias and variance, noise is a property of the *data* rather than the model.
- Noise refers to unexplained variations  $\epsilon_i$  of data from true model  $y_i = f(x_i) + \epsilon_i$ .
- Real-world examples:
  - Human mislabeling of test instance  $\Rightarrow$  Ideal model will never predict it accurately.
  - Error during collection of temperature due to sensor malfunctioning.
- Cannot do anything about it even if seeded with knowledge about true model.

## Bias-Variance Trade-off: Setup

- Imagine you are given the true distribution  $\mathcal{B}$  of training data (including labels).
- You have a principled way of sampling data sets  $\mathcal{D} \sim \mathcal{B}$  from the training distribution.
- Imagine you create an infinite number of training data sets (and trained models) by repeated sampling.
- You have a *fixed* set  $\mathcal{T}$  of unlabeled test instances.
  - The test set  $\mathcal{T}$  does not change over different training data sets.
  - Compute prediction of each instance in  $\mathcal{T}$  for each trained model.

## Informal Definition of Bias

- Compute averaged prediction of each test instance  $x$  over different training models  $g(x, \mathcal{D})$ .
- Averaged prediction of test instance will be different from true (unknown) model  $f(x)$ .
- Difference between (averaged)  $g(x, \mathcal{D})$  and  $f(x)$  caused by erroneous assumptions/simplifications in modeling  $\Rightarrow$  Bias
  - **Example:** Linear simplification to polynomial model causes bias.
  - If the true (unknown) model  $f(x)$  were an order-4 polynomial, and we used any polynomial of order-4 or greater in  $g(x, \mathcal{D})$ , bias would be 0.

## Informal Definition of Variance

- The value  $g(x, \mathcal{D})$  will vary with  $\mathcal{D}$  for fixed  $x$ .
  - The prediction of the same test instance will be different over different trained models.
- All these predictions cannot be simultaneously correct  $\Rightarrow$  Variation contributes to error
- Variance of  $g(x, \mathcal{D})$  over different training data sets  $\Rightarrow$  Model Variance
  - **Example:** Linear model will have low variance.
  - Higher-order model will have high variance.

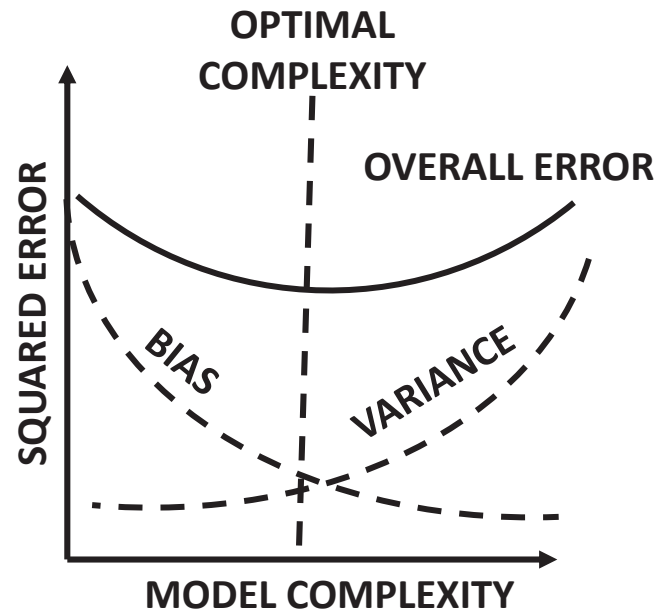
## Bias-Variance Equation

- Let  $E[MSE]$  be the expected mean-squared error of the fixed set of test instances over different samples of training data sets.

$$E[MSE] = \text{Bias}^2 + \text{Variance} + \text{Noise} \quad (1)$$

- In linear models, the bias component will contribute more to  $E[MSE]$ .
  - In polynomial models, the variance component will contribute more to  $E[MSE]$ .
- We have a trade-off, when it comes to choosing model complexity!

## The Bias-Variance Trade-Off



- Optimal point of model complexity is somewhere in middle.

## Key Takeaway of Bias-Variance Trade-Off

- A model with greater complexity might be *theoretically* more accurate (i.e., low bias).
  - But you have less control on what it might predict on a tiny training data set.
  - Different training data sets will result in widely *varying* predictions of same test instance.
  - Some of these must be wrong  $\Rightarrow$  Contribution of model variance.
- *A more accurate model for infinite data is not a more accurate model for finite data.*
  - Do not use a sledgehammer to swat a fly!

## Model Generalization in Neural Networks

- The recent success of neural networks is made possible by increased data.
  - Large data sets help in generalization.
- In a neural network, increasing the number of hidden units in intermediate layers tends to increase complexity.
- Increasing depth often helps in reducing the number of units in hidden layers.
- Proper design choices can reduce overfitting in complex models  $\Rightarrow$  Better to use complex models with appropriate design choices

## How to Detect Overfitting

- The error on test data might be caused by several reasons.
  - Other reasons might be bias (underfitting), noise, and poor convergence.
- Overfitting shows up as a large gap between in-sample and out-of-sample accuracy.
- First solution is to collect more data.
  - More data might not always be available!

## Improving Generalization in Neural Networks

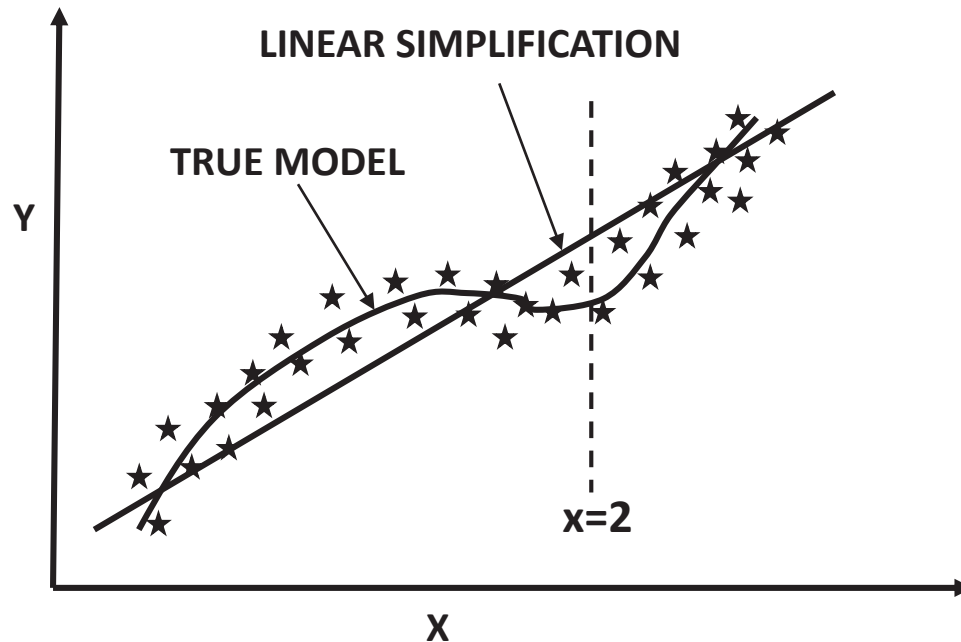
- Key techniques to improve generalization:
  - Penalty-based regularization.
  - Constraints like shared parameters.
  - Using ensemble methods like *Dropout*.
  - Adding noise and stochasticity to input or hidden units.
- Discussion in upcoming lectures.

Charu C. Aggarwal  
IBM T J Watson Research Center  
Yorktown Heights, NY

## Penalty-Based Regularization

Neural Networks and Deep Learning, Springer, 2018  
Chapter 4, Section 4.4

## Revisiting Example: Predict $y$ from $x$



- **First impression:** Polynomial model such as  $y = w_0 + w_1x + w_2x^2 + w_3x^3 + w_4x^4$  is “better” than linear model  $y = w_0 + w_1x$ .
  - However, with less data, using the linear model is better.

## Economy in Parameters

- A lower-order model has economy in parameters.
  - A linear model uses two parameters, whereas an order-4 model uses five parameters.
  - Economy in parameters discourages overfitting.
- Choosing a neural network with fewer units per layer enforces economy.

## Soft Economy vs Hard Economy

- Fixing the architecture up front is an inflexible solution.
- A softer solution uses a larger model but imposes a (tunable) penalty on parameter use.

$$\hat{y} = \sum_{i=0}^d w_i x^i \quad (2)$$

- Loss function:  $L = \sum_{(x,y) \in \mathcal{D}} (y - \hat{y})^2 + \underbrace{\lambda \cdot \sum_{i=0}^d w_i^2}_{L_2\text{-Regularization}}$
- The (tuned) value of  $\lambda$  decides the level of regularization.
- Softer approach with a complex model performs better!

## Effect on Updates

- For learning rate  $\alpha$ , effect on update is to multiply parameter with  $(1 - \alpha\lambda) \in (0, 1)$ .

$$w_i \leftarrow w_i(1 - \alpha\lambda) - \alpha \frac{\partial L}{\partial w_i}$$

- **Interpretation:** Decay-based forgetting!
- 
- Unless a parameter is important, it will have small absolute value.
    - Model decides what is important.
    - Better than inflexibly deciding up front.

## $L_1$ -Regularization

- In  $L_1$ -regularization, an  $L_1$ -penalty is imposed on the loss function.

$$L = \sum_{(x,y) \in \mathcal{D}} (y - \hat{y})^2 + \lambda \cdot \sum_{i=0}^d |w_i|_1$$

- Update has slightly different form:

$$w_i \leftarrow w_i - \alpha \lambda s_i - \alpha \frac{\partial L}{\partial w_i}$$

- The value of  $s_i$  is the partial derivative of  $|w_i|$  w.r.t.  $w_i$ :

$$s_i = \begin{cases} -1 & w_i < 0 \\ +1 & w_i > 0 \end{cases}$$

## $L_1$ - or $L_2$ -Regularization?

- $L_1$ -regularization leads to sparse parameter learning.
  - Zero values of  $w_i$  can be dropped.
  - Equivalent to dropping edges from neural network.
- $L_2$ -regularization generally provides better performance.

## Connections with Noise Injection

- *$L_2$ -regularization with parameter  $\lambda$  is equivalent to adding Gaussian noise with variance  $\lambda$  to input.*
  - **Intuition:** Bad effect of noise will be minimized with simpler models (smaller parameters).
  - Proof in book.
- Result is only true for single layer network (linear regression).
  - Main value of result is in providing general intuition.
- Similar results can be shown for denoising autoencoders.

## Penalizing Hidden Units

- One can also penalize hidden units.
- Applying  $L_1$ -penalty leads to sparse activations.
- More common in unsupervised applications for *sparse feature learning*.
- Straightforward modification of backpropagation.
  - Penalty contributions from hidden units are picked up in backward phase.

## Hard and Soft Weight Sharing

- Fix particular weights to be the same based on domain-specific insights.
  - Discussed in lecture on backpropagation.
- **Soft Weight Sharing:** Add the penalty  $\lambda(w_i - w_j)^2/2$  to loss function.
  - Update to  $w_i$  includes the extra term  $\alpha\lambda(w_j - w_i)$ .
  - Update to  $w_j$  includes the extra term  $\alpha\lambda(w_i - w_j)$ .
  - Pulls weights closer to one another.

Charu C. Aggarwal  
IBM T J Watson Research Center  
Yorktown Heights, NY

## Dropout

Neural Networks and Deep Learning, Springer, 2018  
Chapter 4, Section 4.5

## Feature Co-Adaptation

- The process of training a neural network often leads to a high level of dependence among features.
- Different parts of the network train at different rates:
  - Causes some parts of the network to adapt to others.
- This is referred to as feature co-adaptation.
- Uninformative dependencies are sensitive to nuances of specific training data  $\Rightarrow$  Overfitting.

## One-Way Adaptation

- Consider a single-hidden layer neural network.
  - All edges into and out of half the hidden nodes are fixed to random values.
  - Only the other half are updated during backpropagation.
- Half the features will adapt to the other half (random features).
- Feature co-adaptation is natural in neural networks where rate of training varies across different parts of network over time.
  - Partially a manifestation of training inefficiency (over and above true synergy).

## Why is Feature Co-Adaptation Bad?

- We want features working together only when essential for prediction.
  - We do not want features adjusting to each other because of inefficiencies in training.
  - Does not generalize well to new test data.
- We want *many* groups of minimally essential features for robust prediction  $\Rightarrow$  Better redundancies.
- We do not want a *few* large and inefficiently created groups of co-adapted features.

## Basic Dropout Training Procedure

- For each training instance do:
  - Sample each node in the network in each layer (except output layer) with probability  $p$ .
  - Keep only edges for which both ends are included in network.
  - Perform forward propagation and backpropagation only on sampled network.
- Note that weights are shared between different sampled networks.

## Basic Dropout Testing Procedures

- First procedure:
  - Perform repeated sampling (like training) and average results.
  - Geometric averaging for probabilistic outputs (averaging log-likelihood)
- Second procedure with *weight scaling inference rule* (more common):
  - Multiply weight of each outgoing edge of a sampled node  $i$  with its sampling probability  $p_i$ .
  - Perform single inference on full network with down-scaled weights.

## Why Does Dropout Help?

- By dropping nodes, we are forcing the network to learn without the presence of some inputs (in each layer).
- Will resist co-adaptation, unless the features are truly synergistic.
- Will create many (smaller) groups of self-sufficient predictors.
- Many groups of self-sufficient predictors will have a model-averaging effect.

## The Regularization Perspective

- One can view the dropping of a node as the same process as adding masking noise.
  - Noise is added to both input and hidden layers.
- Adding noise is equivalent to regularization.
- Forces the weights to become more spread out.
  - Updates are distributed across weights based on sampling.

## Practical Aspects of Dropout

- Typical dropout rate (i.e., probability of exclusion) is somewhere between 20% to 50%.
- Better to use a larger network with Dropout to enable learning of independent representations.
- Dropout is applied to both input layers and hidden layers.
- Large learning rate with decay and large momentum.
- Impose a max-norm constraint on the size of network weights.
  - Norm of input weights to a node upper bounded by constant  $c$ .

Charu C. Aggarwal  
IBM T J Watson Research Center  
Yorktown Heights, NY

## Unsupervised Pretraining

Neural Networks and Deep Learning, Springer, 2018  
Chapter 4, Section 4.7

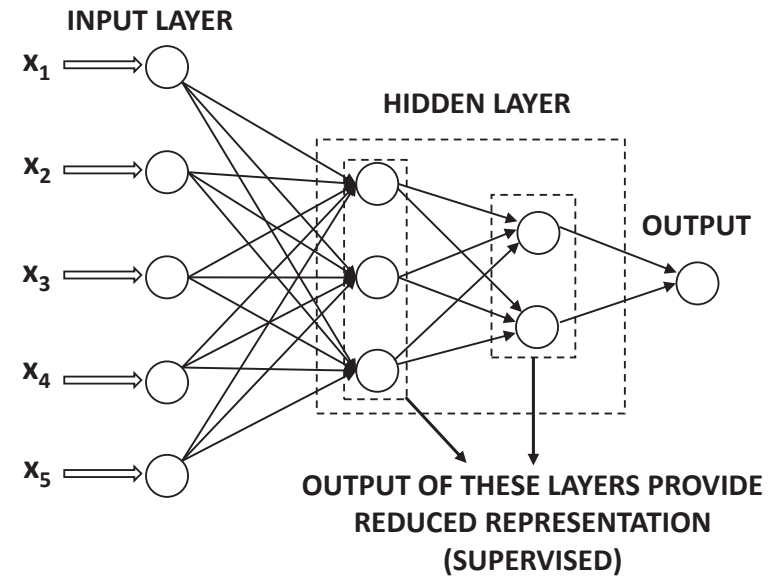
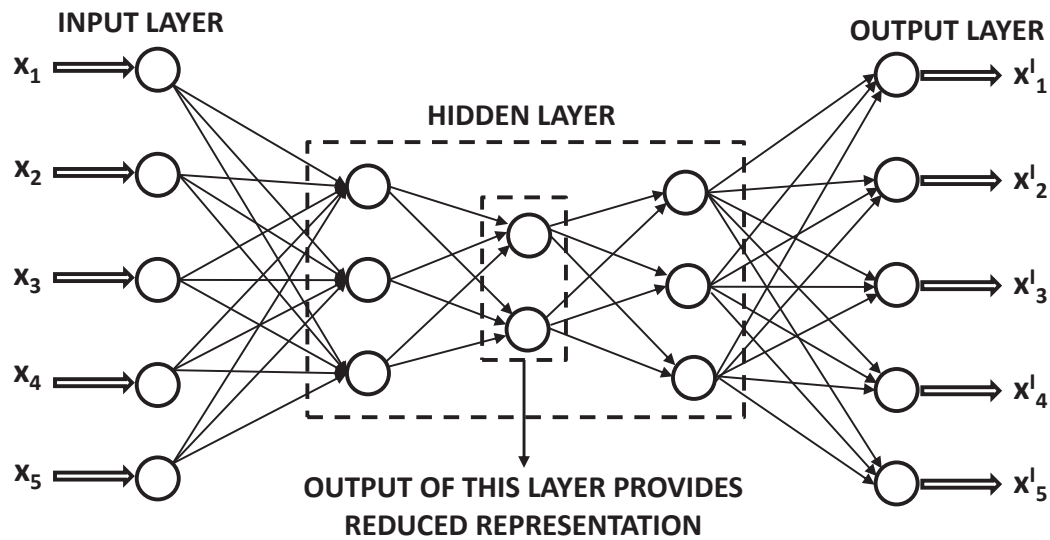
## Importance of Initialization

- Bad initializations can lead to unstable convergence.
- Typical approach is to initialize to a Gaussian with variance  $1/r$ , where  $r$  is the indegree of the neuron.
  - Xavier initialization uses both indegree and outdegree.
- Pretraining goes beyond these simple initializations *by using the training data*.

## Types of Pretraining

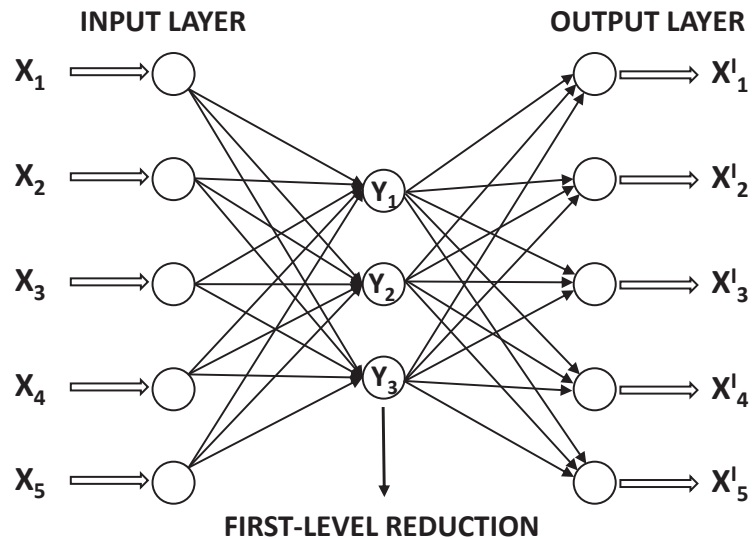
- *Unsupervised pretraining*: Use training data without labels for initialization.
  - Improves convergence behavior.
  - Regularization effect.
- *Supervised pretraining*: Use training data with labels for initialization.
  - Improves convergence but might overfit.
- Focus on unsupervised pretraining.

## Types of Base Applications

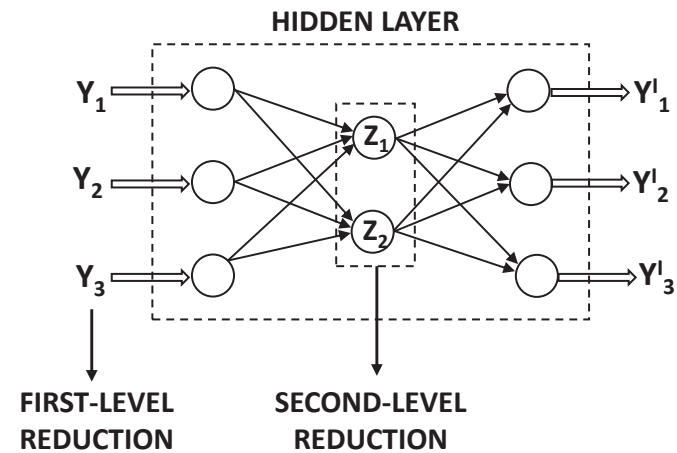


- Both the two neural architectures use almost the same pre-training procedure

## Layer-Wise Pretraining a Deep Autoencoder



(a) Pretraining first-level reduction and outer weights



(b) Pretraining second-level reduction and inner weights

- Pretraining deep autoencoder helps in convergence issues

## Pretraining a Supervised Learner

- For a supervised learner with  $k$  hidden layers:
  - Remove output layer and create an autoencoder with  $(2k-1)$  hidden layers [Refer two slides back].
  - Pretrain autoencoder as discussed in previous slide.
  - Keep only weights from encoder portion and cap with output layer.
  - Pretrain only output layer.
  - Fine-tune all layers.

## Some Observations

- For unsupervised pretraining, other methods may be used.
- Historically, restricted Boltzmann machines were used before autoencoders.
- One does not need to pretrain layer-by-layer.
  - We can group multiple layers together for pretraining (e.g., VGGNet).
  - Trade-off between component-wise learning and global quality.

## Why Does Pretraining Work?

- Pretraining already brings the activations of the neural network to the manifold of the data distribution.
- Features correspond to repeated patterns in the data.
- Fine-tuning learns to combine/modify relevant ones for inference.
  - Pretraining initializes the problem closer to the basin of global optima.
  - Hinton: “*To recognize shapes, first learn to generate images.*”

Charu C. Aggarwal  
IBM T J Watson Research Center  
Yorktown Heights, NY

# Regularization in Unsupervised Applications [Denoising, Contractive, Variational Autoencoders]

Neural Networks and Deep Learning, Springer, 2018  
Chapter 4, Section 4.10

## Supervised vs Unsupervised Applications

- There is always greater tendency to overfit in supervised applications.
  - In supervised applications, we are trying to learn a single bit of target data.
  - In unsupervised applications, a lot more target data is available.
- The goal of regularization is often to provide specific properties to the reduced representation.
- Regularized autoencoders often use a larger number of hidden units than inputs (overcomplete).

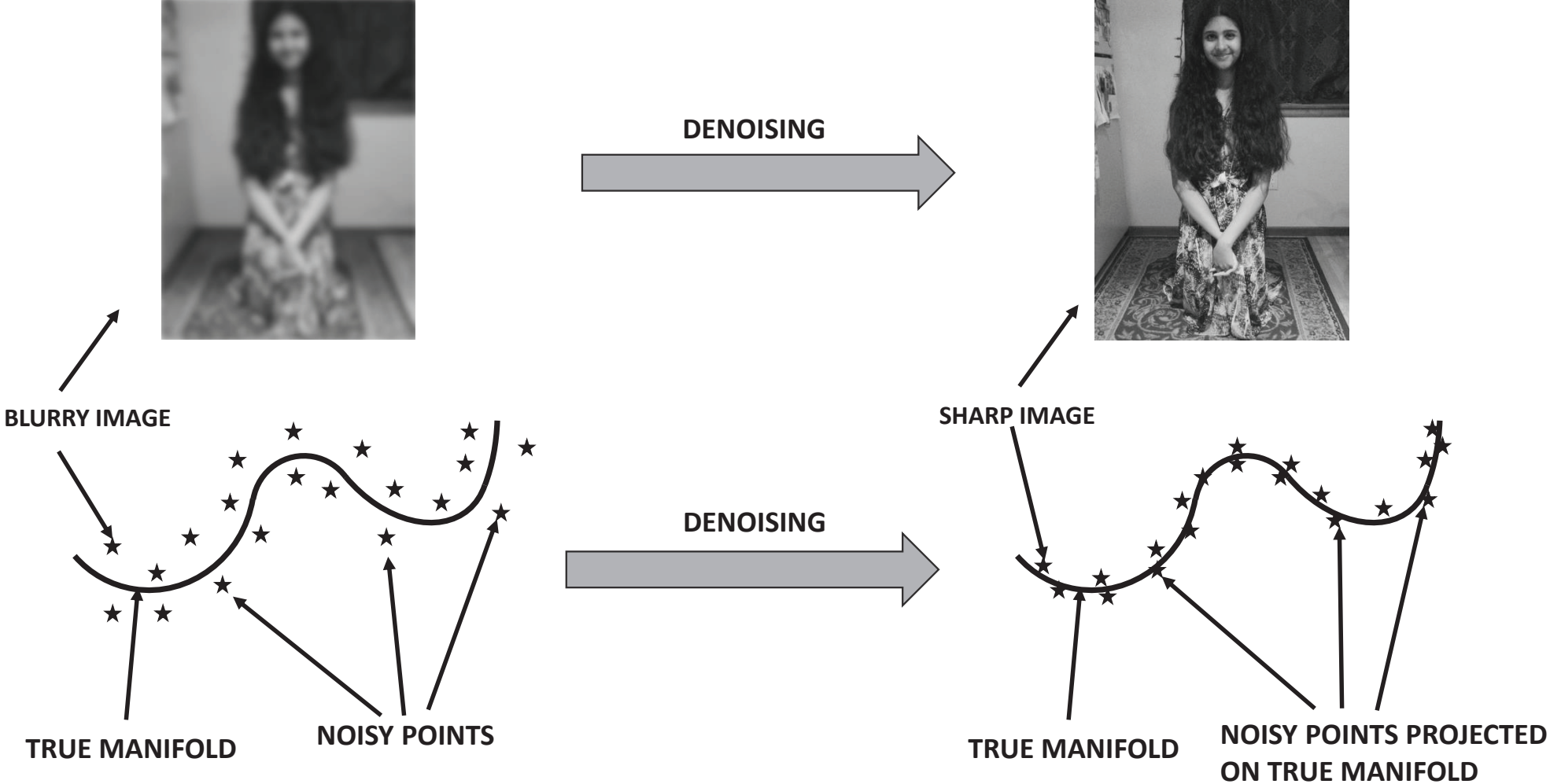
## Sparse Feature Learning

- Use a larger number of hidden units than input units.
- Add  $L_1$ -penalties to the hidden layer.
  - Backpropagation picks up the flow from penalties in hidden layer.
- Use only top activations in hidden layer.
  - Backpropagate only through top activations.
  - Behaves like adaptive ReLU.

## Denoising Autoencoder

- Add noise to the input representation.
  - Gaussian noise for real-valued data and masking noise for binary data.
- Output remains unchanged.
- For single-layer autoencoder with linear activations, Gaussian noise results in  $L_2$ -regularized SVD.

# Illustration of Denoising Autoencoder



## Gradient-Based Penalization: Contractive Autoencoders

- We do not want the hidden representation to change very significantly with small *random* changes in input values.
  - Key point: Most random changes in full-dimensional space are roughly perpendicular to a low-dimensional manifold containing the training data.
- Use a regularization term which tends to selectively damp the component of the movement perpendicular to manifold.
  - Regularizer damps in all directions, but faces no resistance in orthogonal direction to manifold.

## Loss Function

- The loss function adds up the reconstruction error and uses penalties on the gradients of the hidden layer.

$$L = \sum_{i=1}^d (x_i - \hat{x}_i)^2 \quad (3)$$

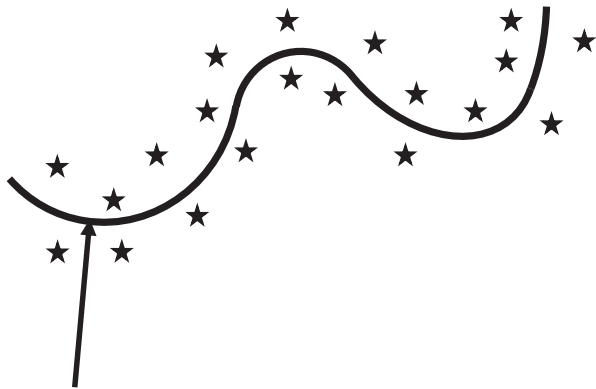
- Regularizer = Sum of squares of the partial derivatives of all hidden variables with respect to all input dimensions.
- Problem with  $k$  hidden units denoted by  $h_1 \dots h_k$ :

$$R = \frac{1}{2} \sum_{i=1}^d \sum_{j=1}^k \left( \frac{\partial h_j}{\partial x_i} \right)^2 \quad (4)$$

- We want to optimize  $L + \lambda R \Rightarrow$  Using single linear layer leads to  $L_2$ -regularized SVD!

# Contractive Autoencoder vs Denoising Autoencoder

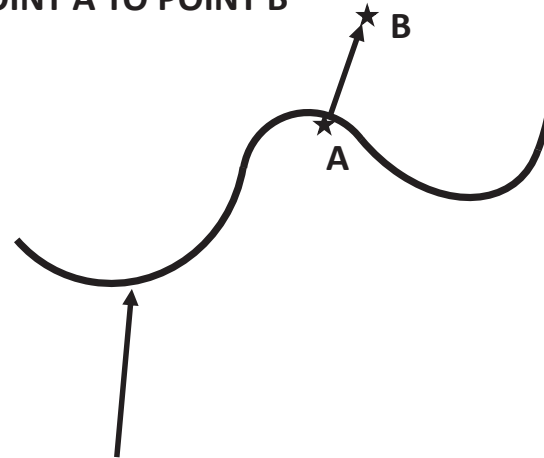
DENOISING AUTOENCODER LEARNS TO DISCRIMINATE BETWEEN NOISE DIRECTIONS AND MANIFOLD DIRECTIONS



TRUE MANIFOLD

DENOISING AUTOENCODER

HIDDEN REPRESENTATION ON MANIFOLD DOES NOT CHANGE MUCH BY PERTURBING POINT A TO POINT B



TRUE MANIFOLD

CONTRACTIVE AUTOENCODER

- Movements inconsistent with data distribution are damped.
- New data point will be projected to manifold (like denoising autoencoder)

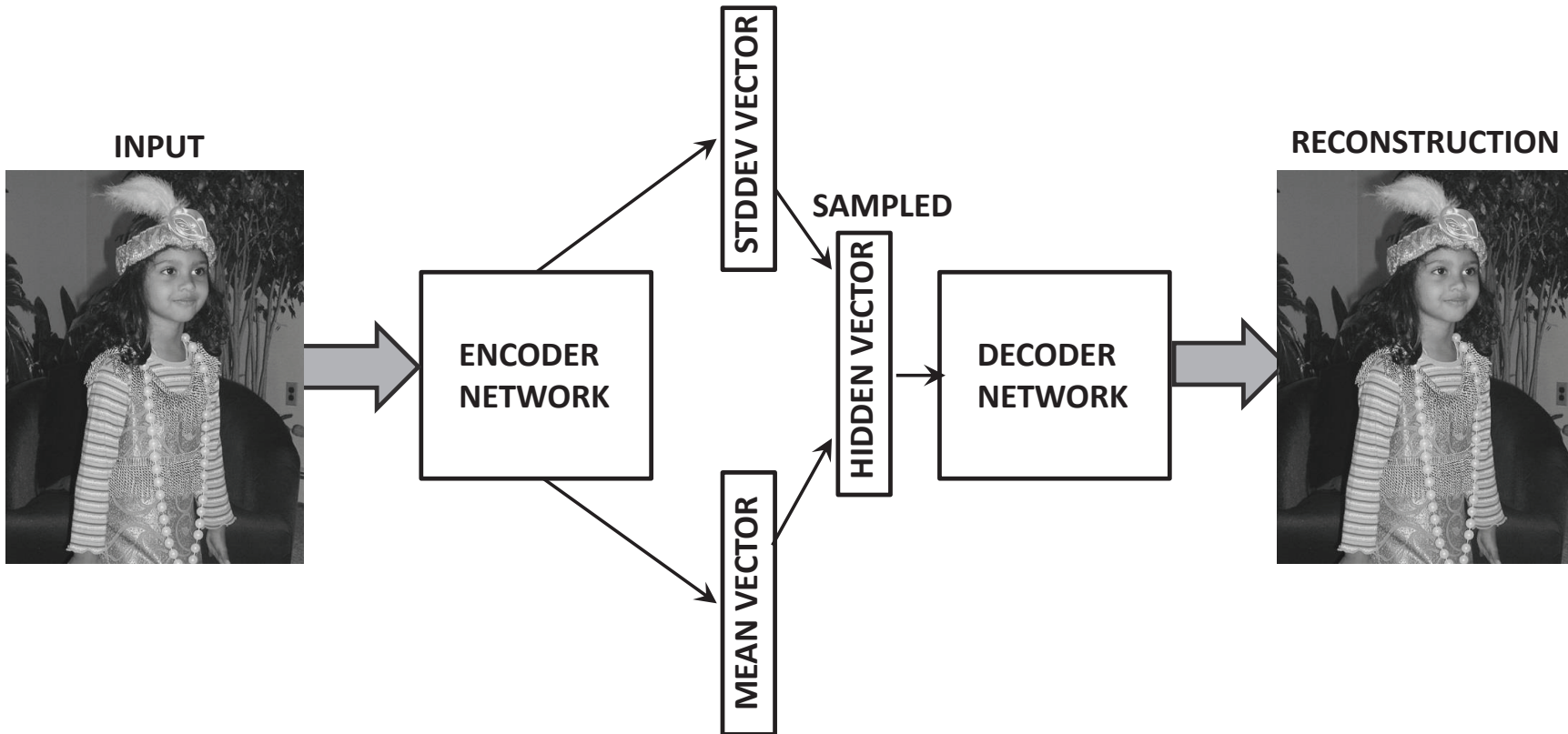
## Variational Autoencoder

- All the autoencoders discussed so far create a deterministic hidden representation.
- The variational autoencoder creates a *stochastic* hidden representation.
- The output is a *sample* from the stochastic representation.
- Objective contains (i) reconstruction error of sample, and (ii) regularization terms pushing the parameters of distribution to unit Gaussian.

## Regularization of Hidden Distribution

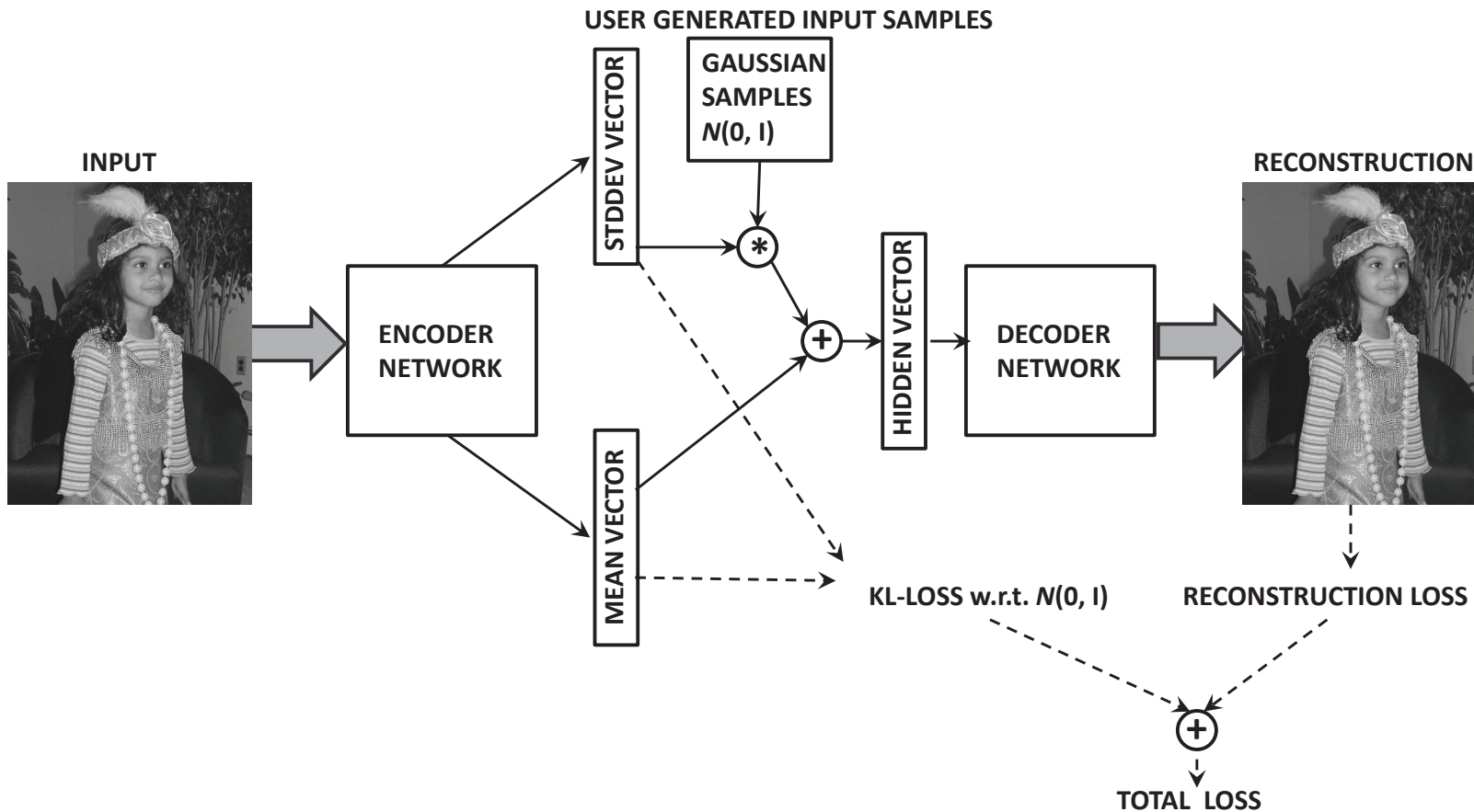
- The hidden distribution is pushed towards Gaussian with zero mean and unit variance in  $k$  dimensions *over the full training data*.
  - However, the *conditional* distribution on a specific input point will be a Gaussian with its own mean vector  $\bar{\mu}(\bar{X})$  and standard deviation vector  $\bar{\sigma}(\bar{X})$ .
  - The encoder outputs  $\bar{\mu}(\bar{X})$  and  $\bar{\sigma}(\bar{X})$  to create samples for decoder.
- Regularizer computes KL-divergence between  $\mathcal{N}(0, I)$  and  $\mathcal{N}(\bar{\mu}(\bar{X}), \bar{\sigma}(\bar{X}))$ .

## Stochastic Architecture with Deterministic Inputs



- One of the operations is sampling from hidden layer  $\Rightarrow$  Cannot backpropagate!

# Conversion to Deterministic Architecture with Stochastic Inputs



- Sampling is accomplished by using pre-generated input samples  $\Rightarrow$  Can backpropagate!

## Objective Function

- Reconstruction loss same as other models:

$$L = \sum_{i=1}^d (x_i - \hat{x}_i)^2 \quad (5)$$

- Regularizer is KL-divergence between unit Gaussian and conditional Gaussian:

$$R = \frac{1}{2} \left( \underbrace{\|\bar{\mu}(\bar{X})\|^2}_{\bar{\mu}(\bar{X})_{i \Rightarrow 0}} + \underbrace{\|\bar{\sigma}(\bar{X})\|^2 - 2 \sum_{i=1}^k \ln(\bar{\sigma}(\bar{X})_i)}_{\bar{\sigma}(\bar{X})_{i \Rightarrow 1}} - k \right) \quad (6)$$

- Overall objective is  $L + \lambda R \Rightarrow$  Backpropagate with deterministic architecture!

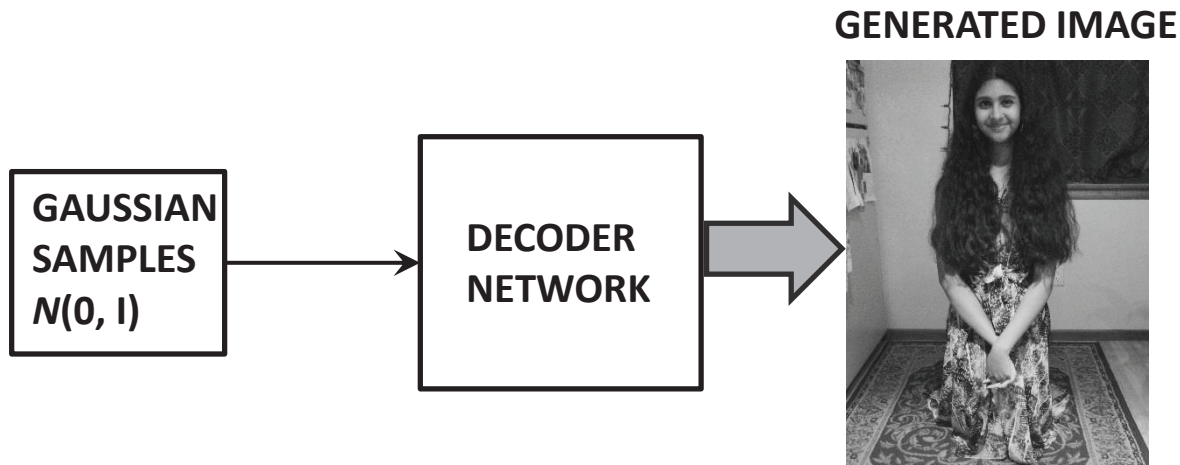
## Connections

- A variational autoencoder will regularize because stochastic (noisy) hidden representation needs to reconstruct.
  - One can interpret the mean as the representation and the standard deviation as the noise robustness of hidden representation.
  - In a denoising autoencoder, we add noise to the inputs.
- Contractive autoencoder is also resistant to noise in inputs (by penalizing hidden-to-input *derivative*).
  - Ensures that hidden representation makes muted changes with small input noise.

## Comparisons

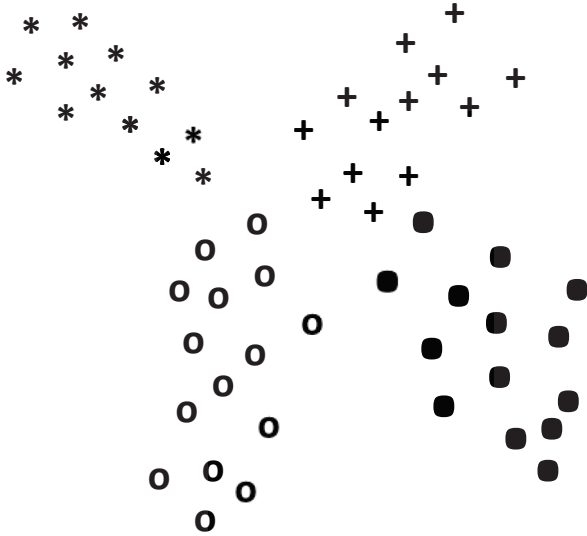
- In denoising autoencoder, noise resistance is shared by encoder and decoder.
  - Often use both in denoising applications.
- In contractive autoencoder, encoder is responsible for noise resistance.
  - Often use only encoder for dimensionality reduction.
- In variational autoencoder, decoder is responsible for noise resistance.
  - Often use only decoder [next slide].

## Variational Autoencoder is Useful as Generative Model

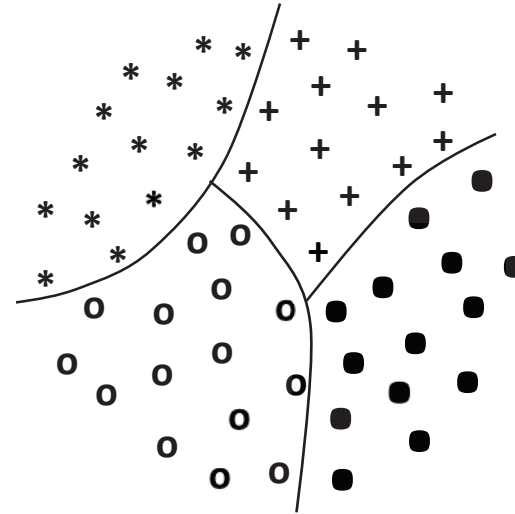


- Throw away encoder and feed samples from  $\mathcal{N}(0, I)$  to decoder.
- Why is this possible for variational autoencoders and not other types of models?

## Effect of the Variational Regularization



**2-D LATENT EMBEDDING  
(NO REGULARIZATION)**



**2-D LATENT EMBEDDING  
(VAE)**

- Most autoencoders will create representations with large discontinuities in the hidden space.
- Discontinuous regions will not generate meaningful points.

## Applications of Variational Autoencoder

- Variational autoencoders have similar applications as Generative Adversarial Networks (GANs).
  - Can also develop conditional variants to fill in missing information (like cGANs).
  - More details in book.
- Quality of generated data is often not as sharp as GANs.