



Autoencoders

Jie Tang

Tsinghua University

Overview

1 Introduction

2 Extended Autoencoder

- Sparse AE
- Denoising AE
- Contractive AE
- Stacked AE
- Convolutional AE
- Variational AE

3 Summary

Overview

1 Introduction

2 Extended Autoencoder

- Sparse AE
- Denoising AE
- Contractive AE
- Stacked AE
- Convolutional AE
- Variational AE

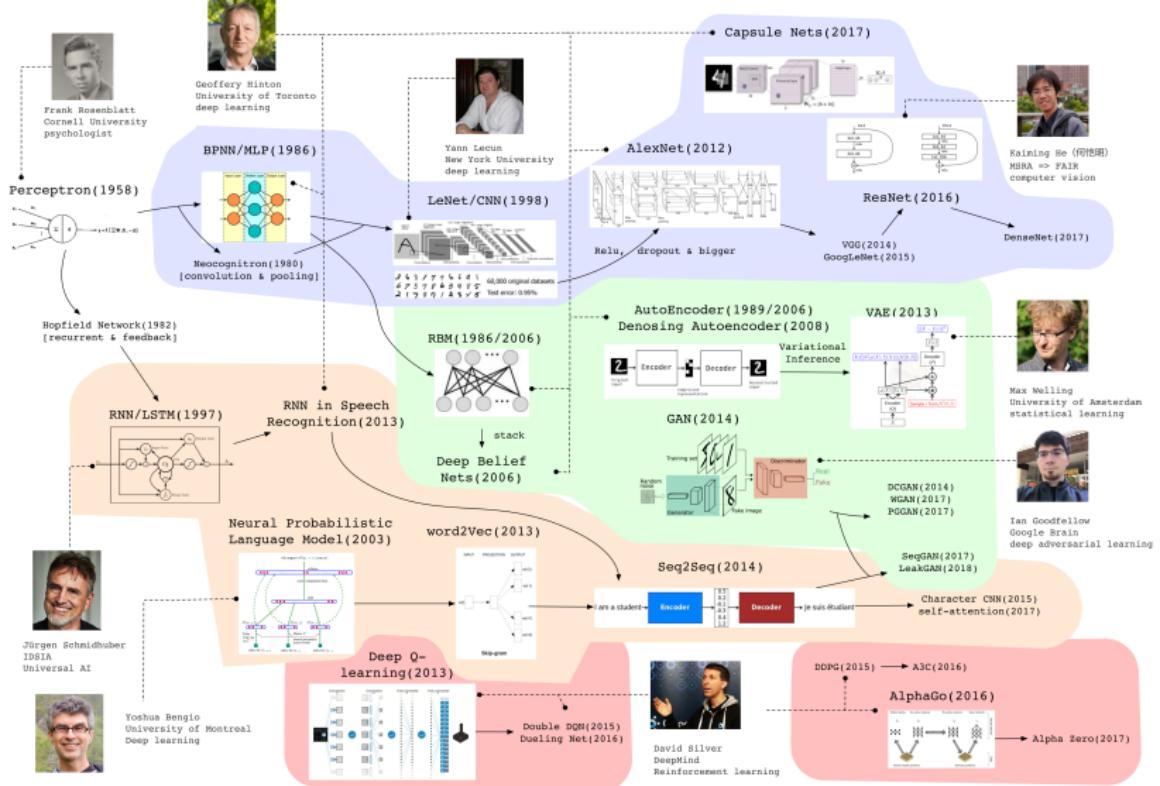
3 Summary

Autoencoder

- An **autoencoder** neural network is an unsupervised learning algorithm that applies back-propagation, setting the **target values** to be equal to the **inputs**;
- was first introduced by Dana Ballard as a way of conducting pretraining in ANNs¹;
- typically for the purpose of dimensionality reduction;
- the learned hidden layers are called the **representation** (encoding) of the input data.²

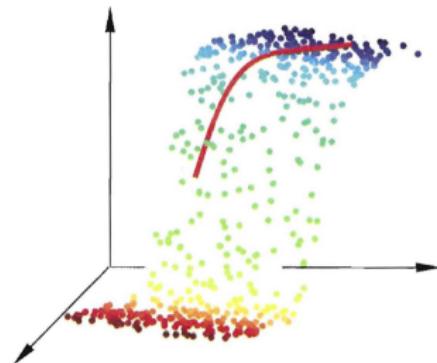
¹Dana H. Ballard Modular learning in neural networks. AAAI'87, pp. 279-284

²Yoshua Bengio. Learning Deep Architectures for AI. Foundations and Trends in Machine Learning, 2, 2009.

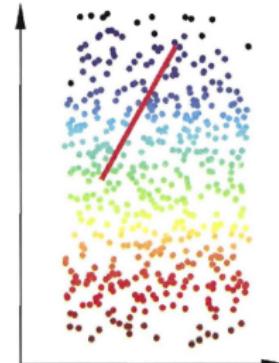


Why Autoencoder: Curse of dimensionality

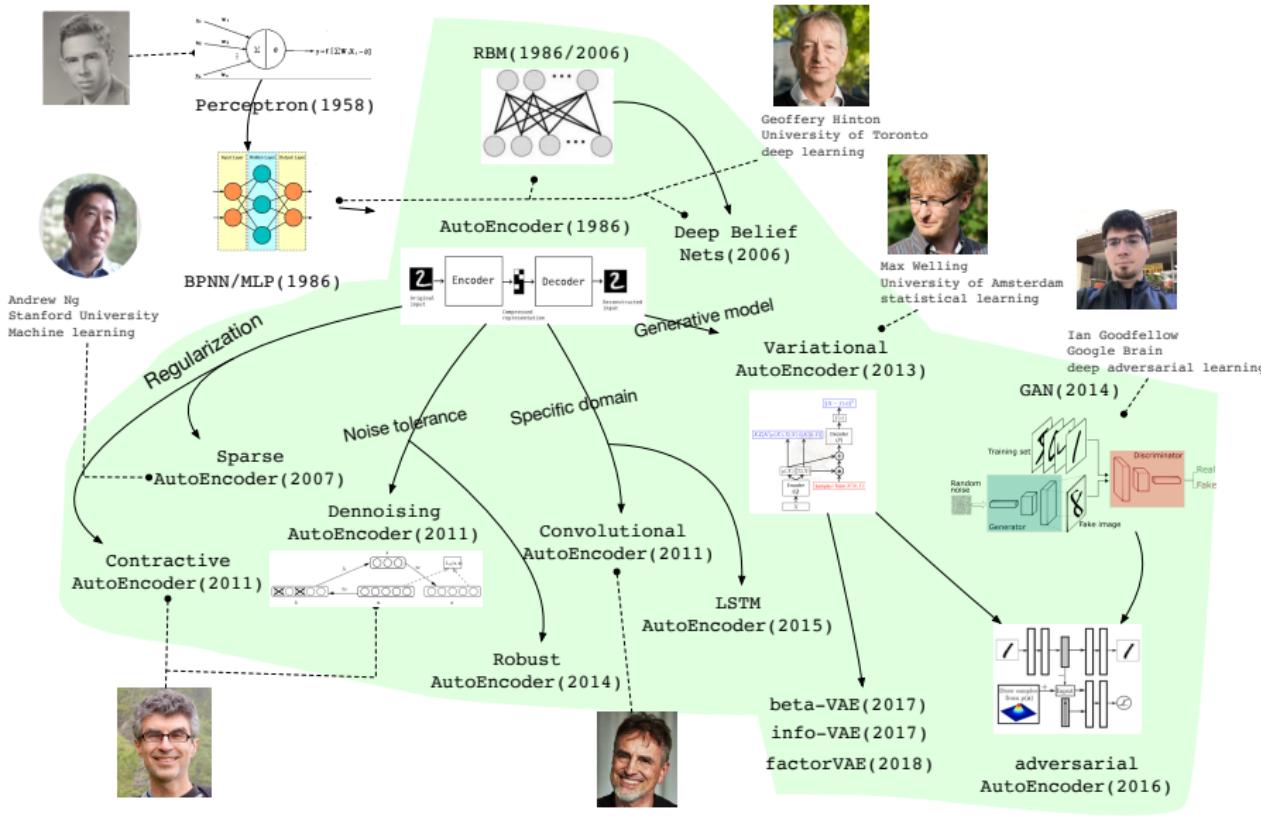
- Data in *high-dimensional spaces* (often with hundreds or thousands of dimensions) proves a serious obstacle.
 - train data is too sparse
 - computation and memory cost is high
- One important solution is dimension reduction.
 - autoencoder is used to learn a representation ($\mathbf{x} \in \mathbb{R}^n \rightarrow \mathbf{y} \in \mathbb{R}^d$, $d \ll n$) for data, typically for the purpose of dimensionality reduction.



Observation data in 3D space

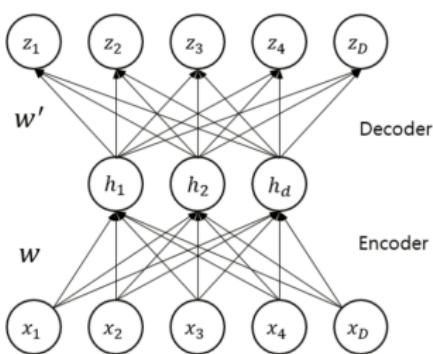


Projection in 2D space



Autoencoder

- Unsupervised learning: only use the inputs $X = \{x^{(1)}, x^{(2)}, \dots, x^{(n)}\}$ for learning.
 - automatically extract meaningful features for your data
 - leverage the availability of unlabeled data
 - add a **data-dependent regularizer** to trainings
- Autoencoder is a feedforward neural network trained to reproduce its input at the output layer: $\mathbf{z} \approx \mathbf{x}$



Encoder:

$$\mathbf{h} = f_{\theta}(\mathbf{x}) = a(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Decoder:

$$\mathbf{z} = g_{\theta'}(\mathbf{h}) = a(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

$$\theta, \theta' : \{\mathbf{W}, \mathbf{b}\}, \{\mathbf{W}', \mathbf{b}'\}$$

Autoencoder—Parameter Estimation

- Parameter estimation:

$$\begin{aligned}\theta^*, \theta'^* &= \underset{\theta, \theta'}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n L(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}) \\ &= \underset{\theta, \theta'}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n L(\mathbf{x}^{(i)}, g_{\theta'}(f_{\theta}(\mathbf{x}^{(i)})))\end{aligned}\tag{1}$$

where L is a loss function such as *squared error*

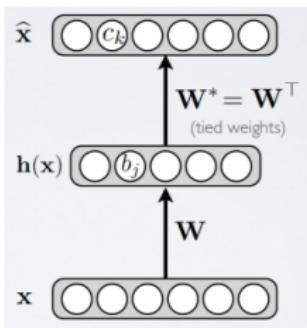
$$L(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|^2.$$

- If \mathbf{x} and \mathbf{z} are vectors of bit probabilities (Bernoullis), L is also chosen to be the *cross-entropy*:

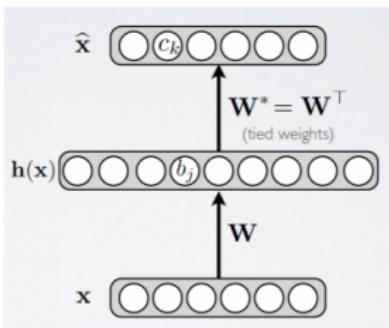
$$L_H(\mathbf{x}, \mathbf{z}) = H(\beta_{\mathbf{x}} || \beta_{\mathbf{z}}) = - \sum_{k=1}^d [\mathbf{x}_k \log \mathbf{z}_k + (1 - \mathbf{x}_k) \log (1 - \mathbf{z}_k)]$$

Undercomplete Vs Overcomplete

- Hidden layer is **undercomplete** if its dimension is smaller than that of the input layer:
 - hidden layer “compress” the data
 - hidden layer will be good features for the train distribution
- Hidden layer is **overcomplete** if its dimension is greater than that of the input layer:
 - no compression in hidden layer
 - no guarantee that the hidden units will extract meaningful structure



(a) undercomplete autoencoder.



(b) overcomplete autoencoder.

* Optionally, the reverse mapping W^* may be constrained to be the transpose of the forward mapping: $W^* = W^T$. This is referred to as **tied weights**.

Linear Autoencoder Corresponds to PCA

To do the proof, we need the following theorem:

- let \mathbf{A} be any matrix, with singular value decomposition
$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$$
 - Σ is a diagonal matrix
 - \mathbf{U}, \mathbf{V} are orthonormal matrices (columns/rows are orthonormal vectors)
- the matrix \mathbf{B} of rank k that is closest to \mathbf{A} :

$$\mathbf{B}^* = \underset{\mathbf{B} \text{ s.t. } \text{rank}(\mathbf{B})=k}{\operatorname{argmin}} \|\mathbf{A} - \mathbf{B}\|_F = \mathbf{U}_{:, \leq k} \Sigma_{\leq k, \leq k} \mathbf{V}_{:, \leq k}^T \quad (2)$$

where $\mathbf{U}_{:, \leq k} \Sigma_{\leq k, \leq k} \mathbf{V}_{:, \leq k}^T$ is the decomposition where we keep only the k largest singular values.

- the sum of square error in training data:

$$\min_{\theta} \sum_t \frac{1}{2} (\mathbf{x}_i^{(t)} - \mathbf{z}_i^{(t)})^2 \geq \min_{\mathbf{W}', h(\mathbf{X})} \frac{1}{2} \|\mathbf{X} - \mathbf{W}' h(\mathbf{X})\|_F^2 \quad (3)$$

here $h(\mathbf{X})$ could be any encoder.

Linear autoencoder corresponds to PCA

Based on previous theorem, where $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ and k is hidden layer size:

$$\operatorname{argmin}_{\mathbf{W}', h(\mathbf{X})} \frac{1}{2} \|\mathbf{X} - \mathbf{W}' h(\mathbf{X})\|_F^2 = (\mathbf{W}' \leftarrow \mathbf{U}_{., \leq k} \Sigma_{\leq k, \leq k}, h(\mathbf{X}) \leftarrow \mathbf{V}_{., \leq k}^T)$$

$$\begin{aligned} h(\mathbf{X}) &= \mathbf{V}_{., \leq k}^T \\ &= \mathbf{V}_{., \leq k}^T (\mathbf{X}^T \mathbf{X})^{-1} (\mathbf{X}^T \mathbf{X}) && \leftarrow \text{multiplying by identity} \\ &= \mathbf{V}_{., \leq k}^T (\mathbf{V} \Sigma^T \mathbf{U}^T \mathbf{U} \Sigma \mathbf{V}^T)^{-1} (\mathbf{V} \Sigma^T \mathbf{U}^T \mathbf{X}) && \leftarrow \text{replace with SVD} \\ &= \mathbf{V}_{., \leq k}^T (\mathbf{V} \Sigma^T \Sigma \mathbf{V}^T)^{-1} \mathbf{V} \Sigma^T \mathbf{U}^T \mathbf{X} && \leftarrow \mathbf{U}^T \mathbf{U} = \mathbf{I} \\ &= \mathbf{V}_{., \leq k} \mathbf{V} (\Sigma^T \Sigma)^{-1} \mathbf{V}^T \mathbf{V} \Sigma^T \mathbf{U}^T \mathbf{X} && \leftarrow \mathbf{V} (\Sigma^T \Sigma)^{-1} \mathbf{V}^T \mathbf{V} \Sigma^T \Sigma \mathbf{V}^T = \mathbf{I} \\ &= \mathbf{V}_{., \leq k} (\Sigma \mathbf{V}^T)^{-1} \mathbf{U}^T \mathbf{X} && \leftarrow \mathbf{V}^T \mathbf{V} = \mathbf{I} \\ &= \mathbf{I}_{\leq k, .} \Sigma^{-1} \mathbf{U}^T \mathbf{X} \\ &= \Sigma_{\leq k, \leq k}^{-1} (\mathbf{U}_{., \leq k})^T \mathbf{X} && \leftarrow \text{multiplying by } \mathbf{I}_{\leq k, .} \end{aligned}$$

Linear autoencoder corresponds to PCA

- So an optimal pair of encoder and decoder is

$$h(\mathbf{x}) = \left(\Sigma_{\leq k, \leq k}^{-1} \mathbf{U}_{\cdot, \leq k}^T \right) \mathbf{x}$$
$$\mathbf{z} = (\mathbf{U}_{\cdot, \leq k} \Sigma_{\leq k, \leq k}) h(\mathbf{x})$$

- for the sum of squared difference error
 - for an autoencoder with a linear decoder
 - where optimality means "has the lowest reconstruction error"
- If inputs are normalized as follows:
$$\mathbf{x}^{(t)} \leftarrow \frac{1}{\sqrt{T}} (\mathbf{x}^{(t)} - \frac{1}{T} \sum_{i=1}^T \mathbf{x}^{(i)})$$
 - encoder corresponds to Principal Component Analysis
 - singular values and vectors = eigenvalues/vectors of covariance matrix

Example

- Suppose that inputs \mathbf{x} are the pixel values from a 10×10 image, $\mathbf{x} \in \mathbb{R}^{100}$.
- There are 50 units in **hidden layers**, $\mathbf{y} \in \mathbb{R}^{50}$
- The network must try to reconstruct the 100-dimensional input \mathbf{x} by hidden representation \mathbf{y}
- Since there are only 50 hidden units, it is forced to learn a *compressed representation* of the input.
- If the input were completely random (e.g., each \mathbf{x}_i comes from an IID Gaussian independent of other features) then this compression task would be very difficult.
- If there is structure in data (e.g., some of features are correlated), then this algorithm often ends up learning a **low-dimensional representation** very similar to PCAs.

Overview

1 Introduction

2 Extended Autoencoder

- Sparse AE
- Denoising AE
- Contractive AE
- Stacked AE
- Convolutional AE
- Variational AE

3 Summary

Regular autoencoder is not enough

- What is a **good** representation ?
 - Sparsity? \Rightarrow Sparse AE
 - Denoise? \Rightarrow Denoising AE
 - Robust to disturbance? \Rightarrow contractive AE
 - High level? \Rightarrow Stacked AE
 - Specific data (image) structure? Convolutional AE
 - Generate new data? \Rightarrow Variational AE

Overview

1 Introduction

2 Extended Autoencoder

- Sparse AE
- Denoising AE
- Contractive AE
- Stacked AE
- Convolutional AE
- Variational AE

3 Summary

Sparse Autoencoder: : Overcomplete Autoencoder

- When the number of hidden units is greater than the number of input units, we can still discover interesting structure, by imposing a sparsity constraint on the hidden units.
- A neuron will be thought of as being “active” if its output value is close to 1, or as being “inactive” if its output value is close to 0.³
- Let ρ_j be the average activation of hidden unit j

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^m [a_j^{(2)}(x^{(i)})] \quad (4)$$

- We'd like to constrain the neurons to be inactive most of the time:

$$\hat{\rho}_j = \rho \quad (5)$$

where ρ is a **sparsity parameter**, typically a small value close to zero, which forces most hidden units' activations to be near 0.

³This assumes a sigmoid activation function. If you are using a tanh activation function, we think of a neuron as being inactive when its output is close to -1.

Sparsity Penalty

- We add an extra penalty term to our optimization objective that penalizes $\hat{\rho}_j$ deviating significantly from ρ . For instance:

$$\sum_{j=1}^{s_2} \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j} \quad (6)$$

where s_2 is the number of neurons in the hidden layer

- This sparsity penalty is based on **Kullback-Leibler (KL) divergence**:

$$\sum_{j=1}^{s_2} KL(\beta_\rho || \beta_{\hat{\rho}_j}) \quad (7)$$

where $KL(\beta_\rho || \beta_{\hat{\rho}_j}) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j}$ is the KL divergence between two **Bernoulli** random variables with mean ρ and $\hat{\rho}_j$.

Sparsity penalty

Sparsity penalty has the property that $KL(\beta_\rho || \beta_{\hat{\rho}_j}) = 0$ if $\hat{\rho}_j = \rho$, and otherwise it increases monotonically as $\hat{\rho}_j$ diverges from ρ .

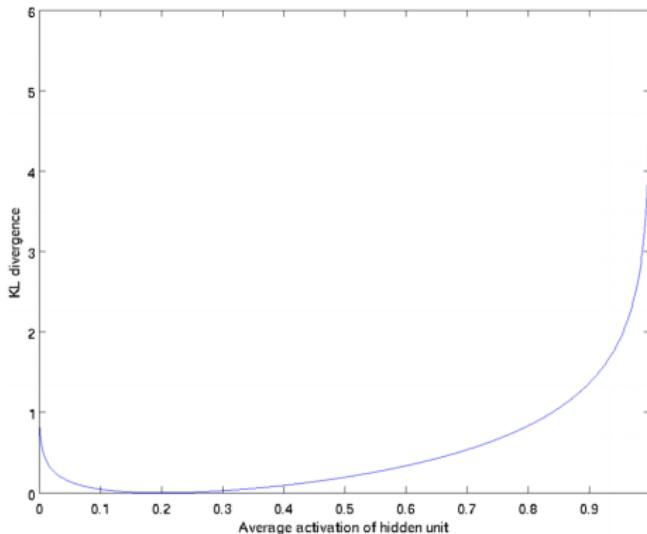


Abbildung 1: Example: $KL(\beta_\rho || \beta_{\hat{\rho}_j})$ is plotted in a range of values of $\hat{\rho}_j$ while ρ is set to be 0.2

Learning with Back Propagation

- Loss function can be written

$$L_{\text{sparse}}(W, b) = L(W, b) + \alpha \sum_{j=1}^{s_2} KL(\beta_\rho || \beta_{\hat{\rho}_j}) \quad (8)$$

- Previously for the second layer ($l = 2$), during back-propagation we would have computed

$$\delta_i^{(2)} = \left(\sum_{j=1}^{s_2} W_{ji}^{(2)} \delta_j^{(3)} \right) f'(z_i^{(2)}) \quad (9)$$

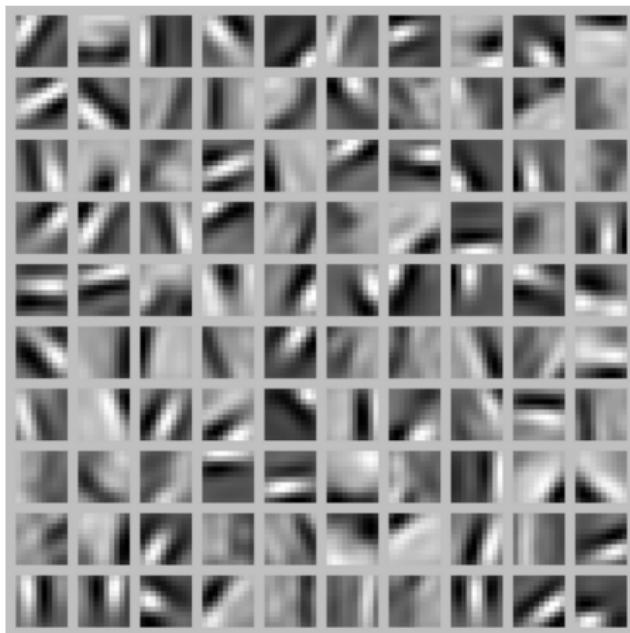
- By adding sparsity penalty, we have

$$\delta_i^{(2)} = \left(\left(\sum_{j=1}^{s_2} W_{ji}^{(2)} \delta_j^{(3)} \right) + \alpha \left(-\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i} \right) \right) f'(z_i^{(2)}) \quad (10)$$

One subtlety is that you need to know $\hat{\rho}_i$.

Sparse autoencoder

- When we train a sparse autoencoder with 100 hidden units on 10×10 pixel inputs we get the following result:



Overview

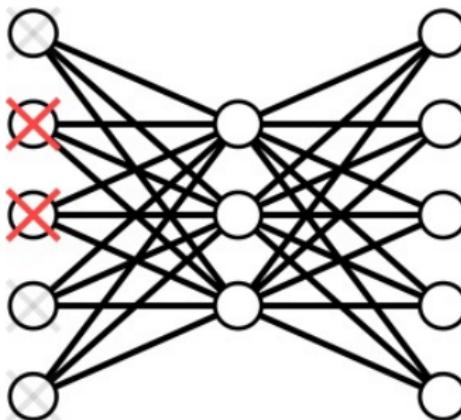
1 Introduction

2 Extended Autoencoder

- Sparse AE
- Denoising AE
- Contractive AE
- Stacked AE
- Convolutional AE
- Variational AE

3 Summary

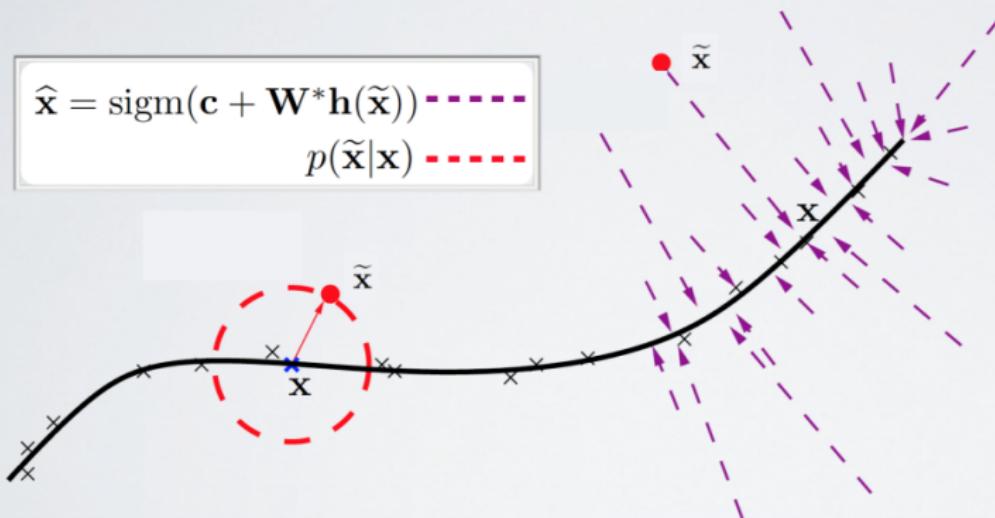
Denoising Autoencoder



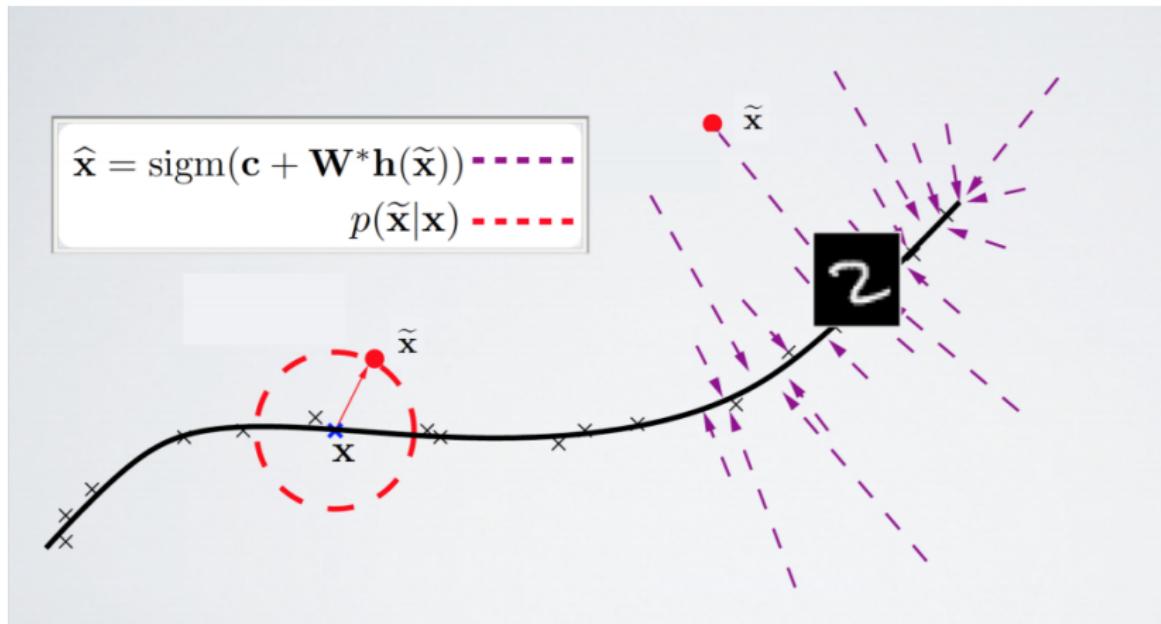
- Idea: representation should be robust to the introduction of noise:
 - random assignment of inputs to 0, with probability $q_{\mathcal{D}}$
 - additive isotropic Gaussian noise $\mathcal{N}(0, \sigma^2 \mathbf{I})$
- Hidden representation $\mathbf{y} = f_{\theta}(\tilde{\mathbf{x}}) = f(\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b})$ is mapped from corrupted input
- Loss function compares reconstruction
 $\mathbf{z} = g_{\theta'}(\mathbf{y}) = f(\mathbf{W}'\mathbf{y} + \mathbf{b}')$ with the original raw input \mathbf{x}

Manifold Learning Perspective

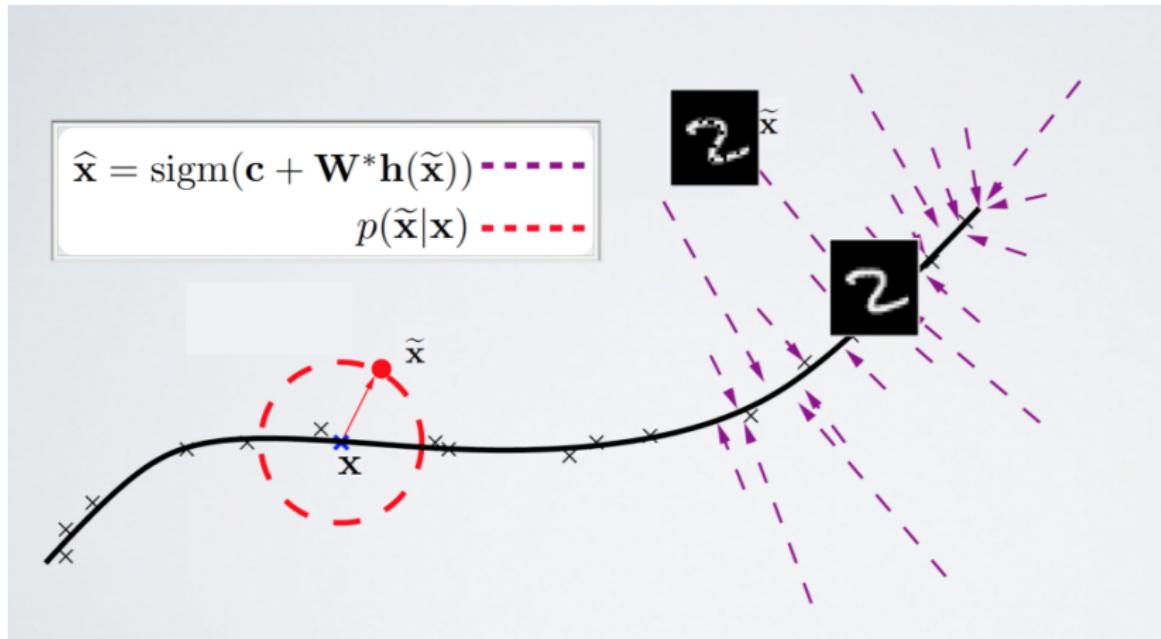
$$\hat{\mathbf{x}} = \text{sigm}(\mathbf{c} + \mathbf{W}^* \mathbf{h}(\tilde{\mathbf{x}}))$$
$$p(\tilde{\mathbf{x}}|\mathbf{x})$$



Manifold Learning Perspective



Manifold Learning Perspective



Overview

1 Introduction

2 Extended Autoencoder

- Sparse AE
- Denoising AE
- Contractive AE
- Stacked AE
- Convolutional AE
- Variational AE

3 Summary

Contractive Autoencoder

- Idea: representation should be robust to small changes of the *training input*.
- Add the Jacobian of the encoder to penalize its sensitivity to training input:

$$\|\mathcal{J}_f(\mathbf{x})\|_F^2 = \sum_{ij} \left(\frac{\partial \mathbf{h}_j}{\partial \mathbf{x}_i} \right)^2 \quad (11)$$

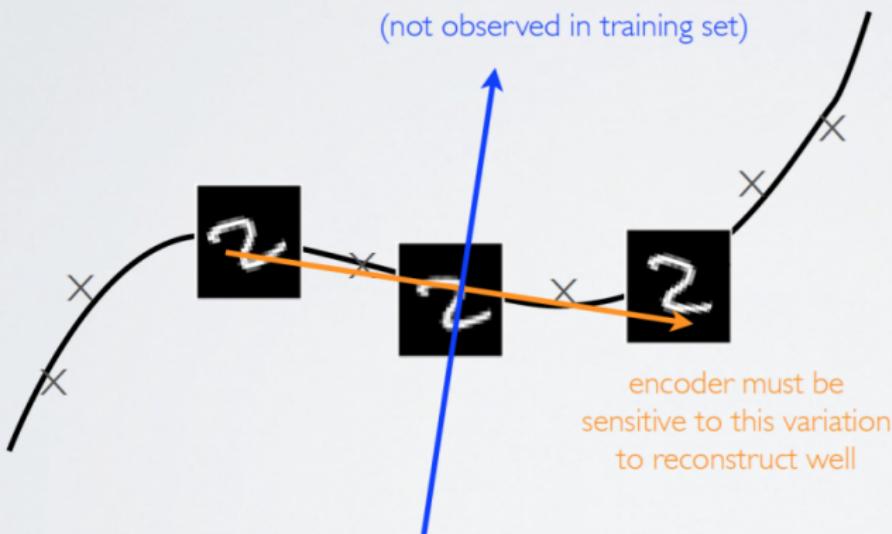
- The penalty $\|\mathcal{J}_f\|_F^2$ encourages the mapping to the feature space to be *contractive* in the neighborhood of the training data.
- New loss function:

$$\mathcal{L}(\mathbf{x}, g(f(\mathbf{x}))) + \lambda \|\mathcal{J}_f(\mathbf{x})\|_F^2 \quad (12)$$

Manifold Learning Perspective

- Illustration:

encoder doesn't need to be
sensitive to this variation
(not observed in training set)



encoder must be
sensitive to this variation
to reconstruct well

Which Autoencoder?

- Both denoising and contractive autoencoders perform well
 - robust to the input data with some noise is useful for classification.
- Advantage of denoising autoencoder: simpler to implement
 - requires adding one or two lines of code to regular autoencoder
 - no need to compute Jacobian of hidden layer
- Advantage of contractive autoencoder:
 - can use second order optimizers (conjugate gradient, LBFGS, etc.)
 - robustness of encoder $f(\mathbf{x})$ is more important than robustness of reconstruction $((g \circ f)(\mathbf{x})$ in DAEs) for classification

Overview

1 Introduction

2 Extended Autoencoder

- Sparse AE
- Denoising AE
- Contractive AE
- Stacked AE
- Convolutional AE
- Variational AE

3 Summary

Stacked Autoencoder

- Stacked autoencoder is a neural network consisting of multiple layers of sparse autoencoders.
- The encoding step for the stacked autoencoder is:

$$\mathbf{z}^{(l+1)} = a(\mathbf{W}^{(l)} \mathbf{z}^{(l)}) \quad (13)$$

- Then the decoding step is:

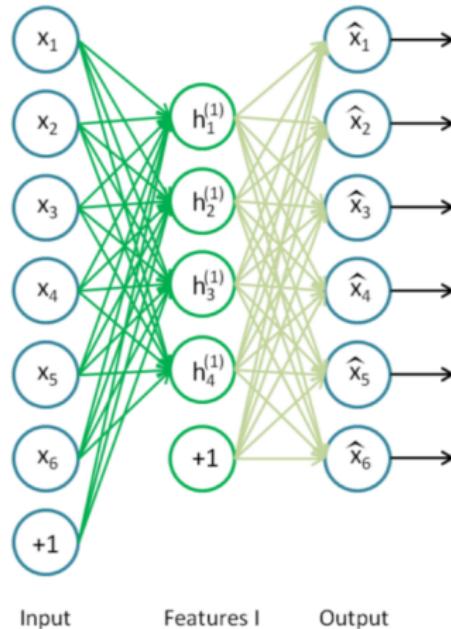
$$\mathbf{z}^{(n+l+1)} = a(\mathbf{W}^{T(n-l)} \mathbf{z}^{(n+l)}) \quad (14)$$

where $\mathbf{W}' = \mathbf{W}^T$, referred to as tiled weight.

Greedy Layer-Wise Training

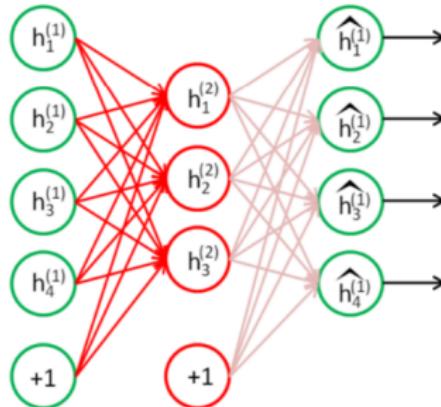
- **Step 1: greedy layer-wise training**
 - We first train a network with 1 hidden layer, and only after that is done, train a network with 2 hidden layers, and so on.
 - At each step, we take the old network as the $k - 1$ hidden layers, and add an additional k -th hidden layer.
 - Training is usually unsupervised, but can be also supervised (e.g. with classification error as loss function on each step).
 - The weights from training the layers individually are then used to initialize the weights in the deep network.
- **Step 2: fine-tuning**
 - example: add a softmax layer of the output of Stacked AE and train the whole structure to fine-tune the parameters.

Greedy Layer-Wise Training



First, we train a sparse autoencoder on the raw inputs \mathbf{x} to learn primary features $\mathbf{h}^{(1)}$.

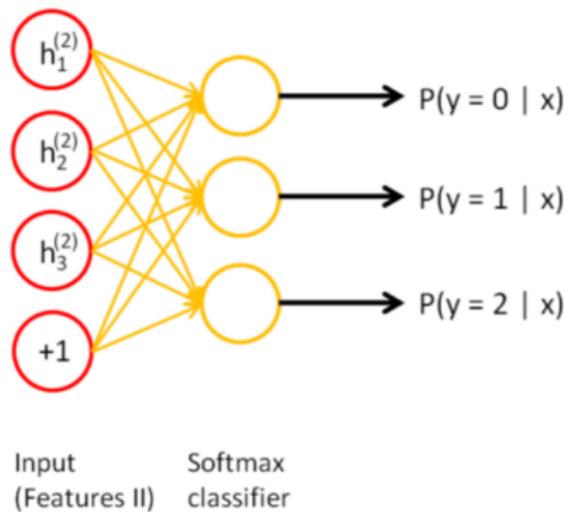
Greedy Layer-Wise Training



Input (Features I)	Features II	Output
-----------------------	-------------	--------

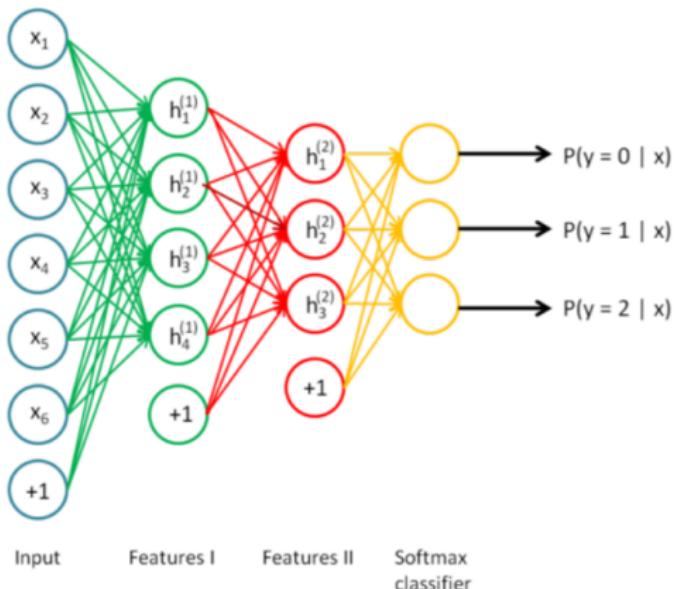
Next, we use these primary feature activations $\mathbf{h}^{(1)}$ as the “raw input” to another sparse to learn secondary features $\mathbf{h}^{(2)}$.

Greedy Layer-Wise Training



We then treat these secondary features as “raw input” to a softmax classifier, training it to map secondary features to digit labels.

Fine-tuning



Finally, we would combine all three layers together to form a stacked autoencoder with 2 hidden layers and a final softmax classifier layer.

Why does it work?

- Availability of data
 - The algorithm is able to learn and discover patterns from massively more amounts of data than purely supervised approaches.
- Better local optima
 - After having trained the network on the unlabeled data, the weights are now starting at a better location in parameter space than if they had been randomly initialized.
 - We can then further fine-tune the weights starting from this location.

Overview

1 Introduction

2 Extended Autoencoder

- Sparse AE
- Denoising AE
- Contractive AE
- Stacked AE
- **Convolutional AE**
- Variational AE

3 Summary

Convolutional AE

- When processing image data, standard AEs do not explicitly consider the 2-dimensional structure.
- Convolutional AE solves this by substituting fully connected layers with convolutional layers.
- The encoding step of Convolutional AE is:

$$h^k = \sigma(x * W^k + b^k) \quad (15)$$

where h^k refers to the k^{th} feature map.

- The decoding step of Convolutional AE is:

$$y = \sigma\left(\sum_{k \in H} h^k * \tilde{W}^k + c\right) \quad (16)$$

where H is the group of latent feature maps, $\tilde{W} = W^T$

Max-Pooling

- Max-pooling layer introduces sparsity over the hidden representation by erasing all non-maximal values in non overlapping sub-regions.
- This forces feature detectors to become more broadly applicable.

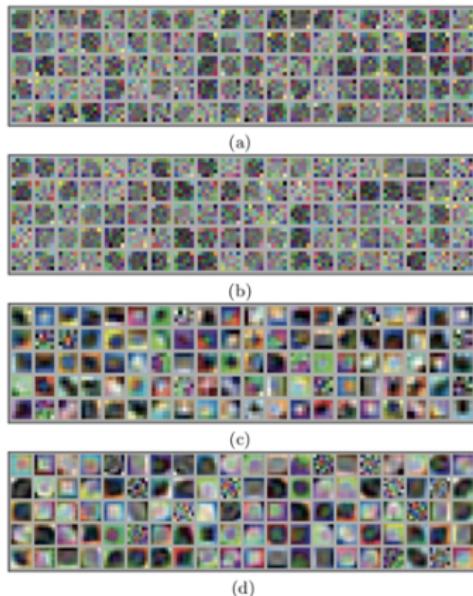


Fig. 2. A randomly selected subset of the first layer's filters learned on CIFAR10 to compare noise and pooling (best viewed in colours). (a) No pooling and 0% noise, (b) No pooling and 50% noise, (c) Pooling of 2×2 and 0% noise, (d) Pooling of 2×2 and 50% noise.

Overview

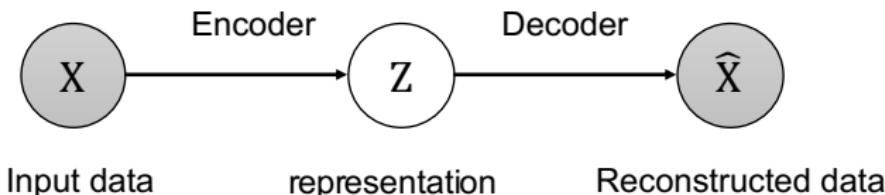
1 Introduction

2 Extended Autoencoder

- Sparse AE
- Denoising AE
- Contractive AE
- Stacked AE
- Convolutional AE
- Variational AE

3 Summary

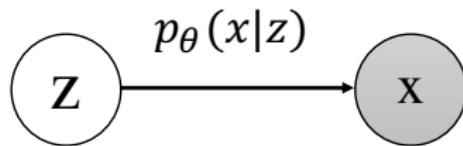
Recap Autoencoder



- Autoencoders can reconstruct data, and can learn features to initialize a supervised model.
- Features capture factors of variation in training data. Can we generate new data from an autoencoder?

Variational AE

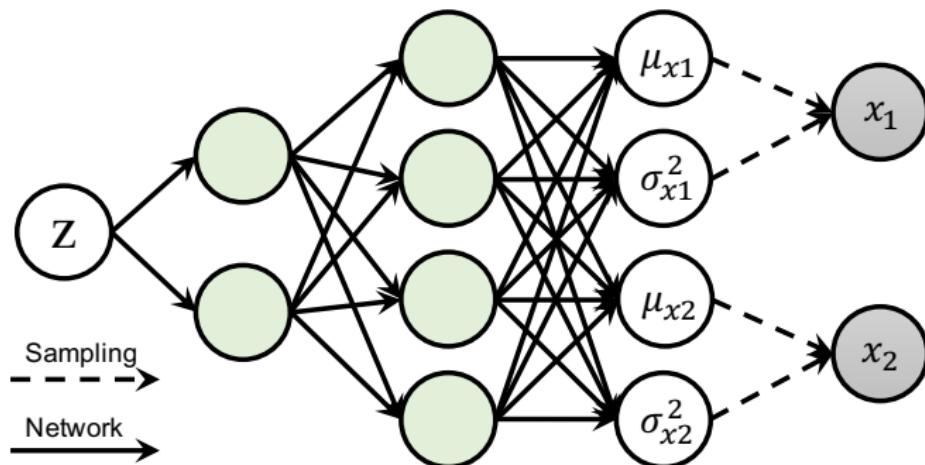
- Probabilistic spin on autoencoder - will let us sample from the model to generate data!
- Assume training data $X = \{x^{(1)}, x^{(2)}, \dots, x^{(N)}\}$ is generated from underlying latent (unobserved) representation $z \sim p(z)$ (Gaussian)



- We wish to estimate the true parameters θ from training data x .
- Modeling the conditional distribution $p_\theta(x|z)$ by a neural neural network (Decoder network).

Decoder Network

- Idea: NN + Gaussian (or Bernoulli) with a diagonal covariance: $p_\theta(x|z) = N(\mu_x, \sigma_x^2)$



How to train the model?

Training as an Autoencoder

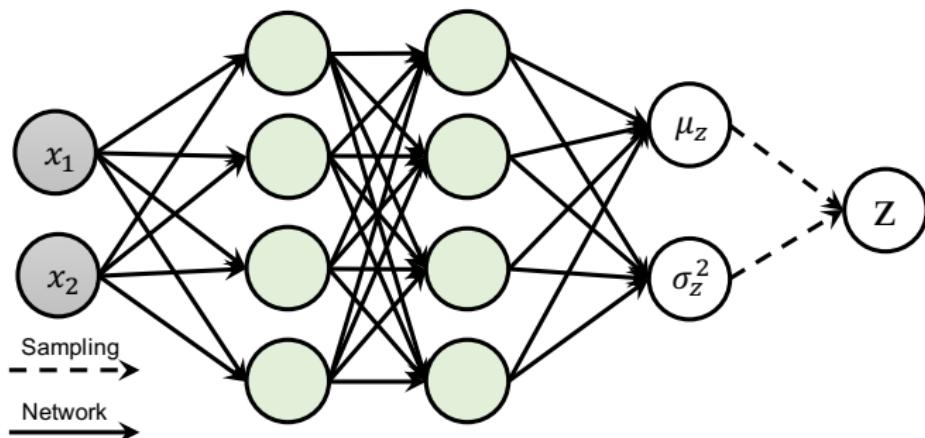
- For generative model, the training objective is to maximize likelihood of training data:

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz \quad (17)$$

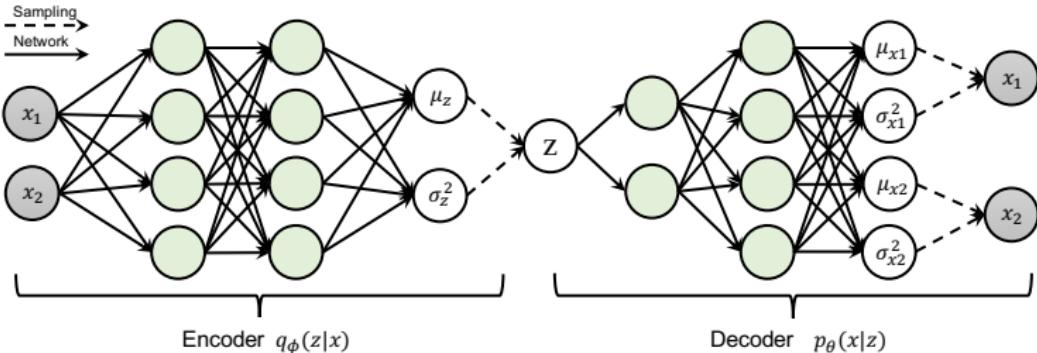
- Problem:** It is **intractable** to compute $p(x|z)$ for every z . The posterior distribution $p_{\theta}(z|x)$ is also **intractable**.
- Solution:** In addition to decoder network $p_{\theta}(x|z)$, define an additional encoder network $q_{\phi}(z|x)$ that approximates $p_{\theta}(z|x)$.
- We will see that this allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize.

Encoder Network

- A feed forward NN + Gaussian. $q_\phi(z|x) = \mathcal{N}(\mu_z(x), \sigma_z(x))$



The Complete Auto-encoder



- Encoder and decoder networks also called “recognition” / “inference” and “generation” networks
- We aim to learn the parameters ϕ and θ via backpropagation.

Training Objective

- Equipped with our encoder and decoder networks, let's work out the data likelihood:

$$\begin{aligned}\log p_\theta(x) &= \mathbb{E}_{z \sim q_\phi(z|x)} [\log p_\theta(x)] \quad (\text{$p_\theta(x)$ Does not depend on z}) \\ &= \mathbb{E}_z \left[\log \frac{p_\theta(x, z)}{p_\theta(z|x)} \right] \quad (\text{Bayes's Rule}) \\ &= \mathbb{E}_z \left[\log \frac{p_\theta(z, x)}{q_\phi(z|x)} \frac{q_\phi(z|x)}{p_\theta(z|x)} \right] \quad (\text{Multiply by constant}) \\ &= \mathbb{E}_z \left[\log \frac{p_\theta(z, x)}{q_\phi(z|x)} \right] + \mathbb{E}_z \left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)} \right] \quad (\text{Logarithms}) \\ &= L(\theta, \phi, x) + \underbrace{D_{KL}(q_\phi(z|x) || p_\theta(z|x))}_{\geq 0} \\ &\geq \underbrace{L(\theta, \phi, x)}_{\text{variational lower bound}}\end{aligned}$$

Variational Lower Bound

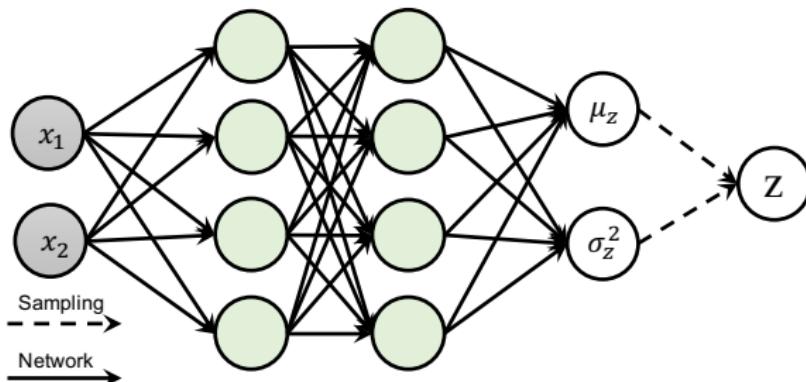
$$\begin{aligned} L(\theta, \phi, x) &= \mathbb{E}_z \left[\log \frac{p_\theta(z, x)}{q_\phi(z|x)} \right] \\ &= \mathbb{E}_z \left[\log \frac{p_\theta(x|z)p_\theta(z)}{q_\phi(z|x)} \right] \quad (\text{Bayes's Rule}) \\ &= \mathbb{E}_z \left[\log \frac{p_\theta(z)}{q_\phi(z|x)} \right] + \mathbb{E}_z [\log p_\theta(x|z)] \quad (\text{Logarithms}) \\ &= \underbrace{-D_{KL}(q_\phi(z|x)||p_\theta(z))}_{\text{Regularization}} + \underbrace{\mathbb{E}_z [\log p_\theta(x|z)]}_{\text{Reconstruction error}} \end{aligned} \tag{18}$$

- The variational lower bound $L(\theta, \phi, x)$ is tractable.
- Our objective is to maximize $L(\theta, \phi, x)$.

Example with Gaussian Distribution

- Use $\mathcal{N}(0, 1)$ as prior for z ; $q_\phi(z|x)$ is Gaussian distribution $\mathcal{N}(\mu_z(x; \phi), \sigma_z^2(x; \phi))$ determined by NN.
 - The KL-divergence:

$$-D_{KL}(q_\phi(z|x) || p(z)) = \frac{1}{2}(1 + \log \sigma_z^2 - \mu_z^2 - \sigma_z^2) \quad (19)$$

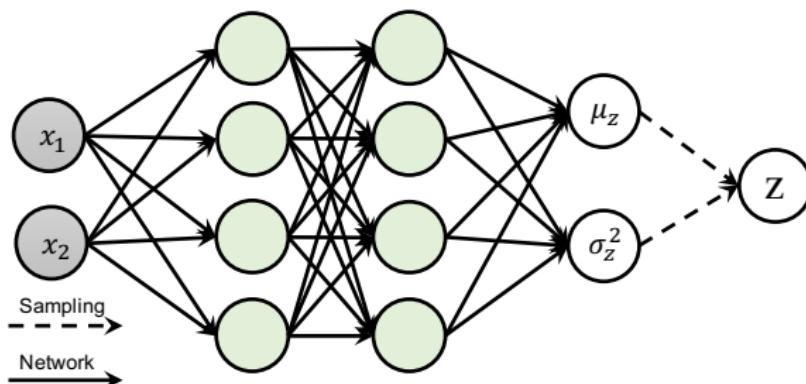


Example with Gaussian Distribution

- Use $\mathcal{N}(0, 1)$ as prior for z ; $q_\phi(z|x)$ is Gaussian distribution $\mathcal{N}(\mu_z(x; \phi), \sigma_z^2(x; \phi))$ determined by NN.
 - Approximate $\mathbb{E}_{q_\phi(z|x)}(\log(p_\theta(x|z)))$ via Monte Carlo methods:

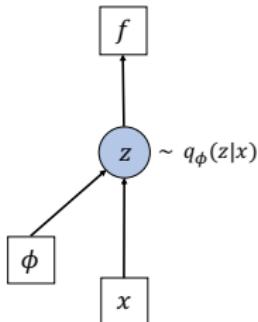
$$\mathbb{E}_z [\log p_\theta(x|z)] \approx \frac{1}{L} \sum_k^L \log p_\theta(x|z^{(k)}), \quad z^{(k)} \sim q_\phi(z|x) \quad (20)$$

- However, $z^{(k)}$ is a random variable, we can't take gradient over a random drawn number.

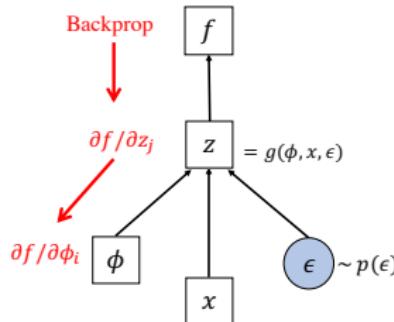


Reparameterization Trick

- Backpropagation not possible through random sampling.



◻ deterministic node ● random node



◻ deterministic node ● random node

$$z^{(k)} \sim \mathcal{N}(\mu_z(x; \phi), \sigma_z^2(x; \phi))$$

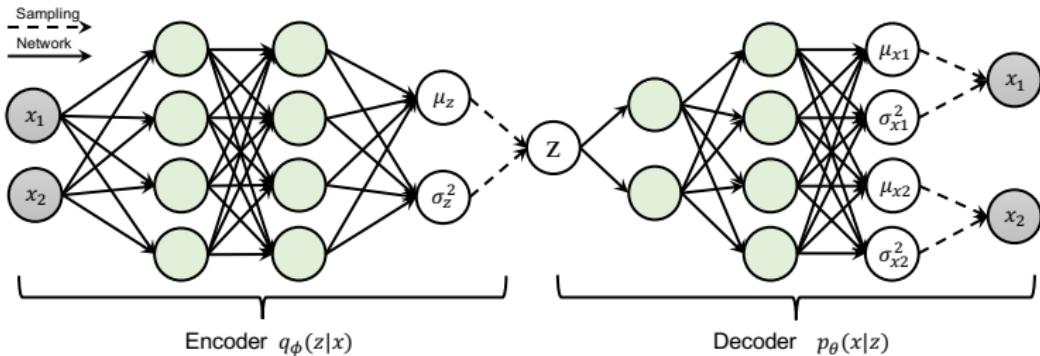
Cannot back-propagate
through a randomly drawn
number

$$\epsilon^{(k)} \sim \mathcal{N}(0, 1)$$

$$z^{(k)} = \mu_z(x, \phi) + \sigma_z(x, \phi) \cdot \epsilon^{(k)}$$

z has the same distribution,
but now can back-propagate.

VAE Summary



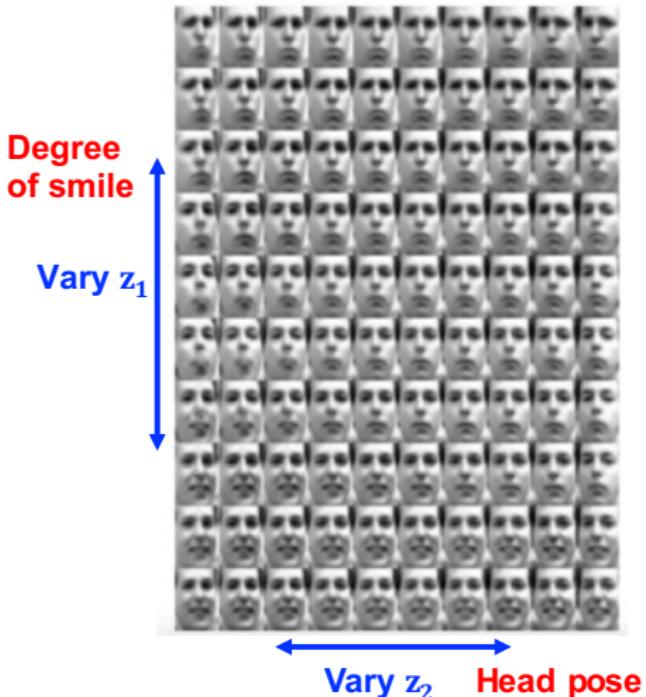
- Loss function:

$$L(\theta, \phi, x) \approx \frac{1}{2} \sum_j^J (1 + \log \sigma_{zj}^2 - \mu_{zj}^2 - \sigma_{zj}^2) + \frac{1}{L} \sum_k^L \log(p_\theta(x|z^{(k)})) \quad (21)$$

where $z^{(k)} = \mu_z(x, \phi) + \sigma_z(x, \phi) \cdot \epsilon^{(k)}$ and $\epsilon^{(k)} \sim \mathcal{N}(0, 1)$.

VAE: Generating new data

- Diagonal prior on z
 \Rightarrow independent latent variables
- Different dimensions of z encode interpretable factors of variation



Summary

- Successfully applied to dimensionality reduction and information retrieval tasks
- Add a data-dependent penalty to trainings
 - sparse autoencoder
 - denoising autoencoder
 - contractive autoencoder
- Add prior distribution assumption to representation
 - variational autoencoder
- An efficient algorithm for initializing the deep neural network
 - stacked autoencoder

Thanks.

HP: <http://keg.cs.tsinghua.edu.cn/jietang/>
Email: jietang@tsinghua.edu.cn