NUS National University of Singapore | School of Computing

CS5340
Uncertainty Modeling in AI

Lecture 11:
Variational AutoEncoder and Mixture Density Networks

Asst. Prof. Lee Gim Hee

AY 2020/2021

Semester 1

Course Schedule

Week	Date	Topic	Remarks
1	12 Aug	Introduction to probabilistic reasoning	**1830hrs: MS Teams (Live Introduction)**
2	19 Aug	Bayesian networks (Directed graphical models)	
3	26 Aug	Markov random Fields (Undirected graphical models)	**1830hrs: Zoom discussions**
4	02 Sep	Variable elimination and belief propagation	**Assignment 1:** Belief propagation and maximal probability (15%)
5	09 Sep	Factor graph and the junction tree algorithm	
6	16 Sep	Parameter learning with complete data	**Assignment 1:** Due **Assignment 2:** Junction tree and parameter learning (15%) **1830hrs: Zoom discussions**
-	23 Sep	Recess week	**No lecture**
7	30 Sep	Mixture models and the EM algorithm	**Assignment 2:** Due **Online quiz 1 (20%)**
8	07 Oct	Hidden Markov Models (HMM)	**Assignment 3:** Hidden Markov model (15%)
9	14 Oct	Monte Carlo inference (Sampling)	**1830hrs: Zoom discussions**
10	21 Oct	Variational inference	**Assignment 3:** Due **Assignment 4:** MCMC Sampling (15%)
11	28 Oct	Variational Auto-Encoder and Mixture Density Networks	
12	04 Nov	Graph-cut and alpha expansion	**Assignment 4:** Due **1830hrs: Zoom discussions**
-	11 Nov	--	**Online quiz 2 (20%)**

Acknowledgements

- A lot of slides and content of this lecture are adopted from:

1. Carl Doersch, “Tutorial on Variational Autoencoders”, in ArXiv, 2016.

2. Diederik P Kingma, Max Welling, “Auto-Encoding Variational Bayes”, in ICLR 2014.

3. Christopher Bishop, “Pattern Recognition and Machine Learning”, Chapter 5.

Learning Outcomes

- Students should be able to:

1. Explain the difference between the discriminative and generative models.
2. Describe the concept behind Variational AutoEncoder, and how it can be used to generate new images.
3. Use the Mixture Density Network to solve the inverse problem where multiple feasible solutions can exist.

Recall: Discriminative Vs Generative Models

- Generative models: Approaches that explicitly or implicitly model the distribution of inputs and outputs.

- Sampling from the distribution it is possible to generate synthetic data points in the input space.

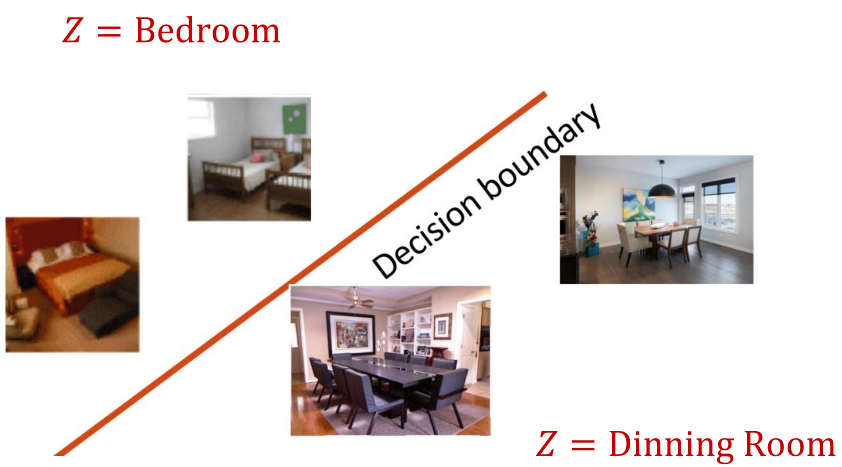
Likelihood: $p(\mathbf{x}|\mathcal{C}_k)$

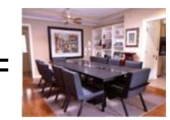
- Discriminative models: Approaches that model the posterior probabilities directly.

Posterior: $p(\mathcal{C}_k|\mathbf{x})$

Discriminative Models

During learning, we model the posterior probability:



$p(Z = \text{Bedroom} \mid X =$  $) = 0.01$

$\vdots$

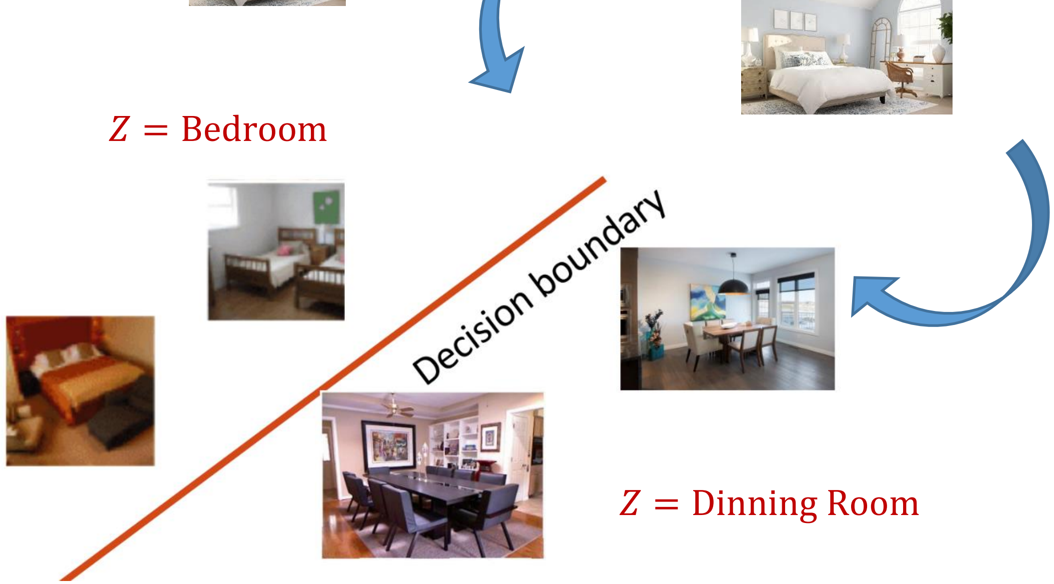
$p(Z = \text{Bedroom} \mid X =$  $) = 0.90$

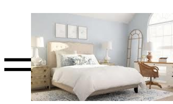
Example: Logistic regression, convolutional network, etc.

Discriminative Models

During inference: Given , find $p(Z \mid X =$  $)$.

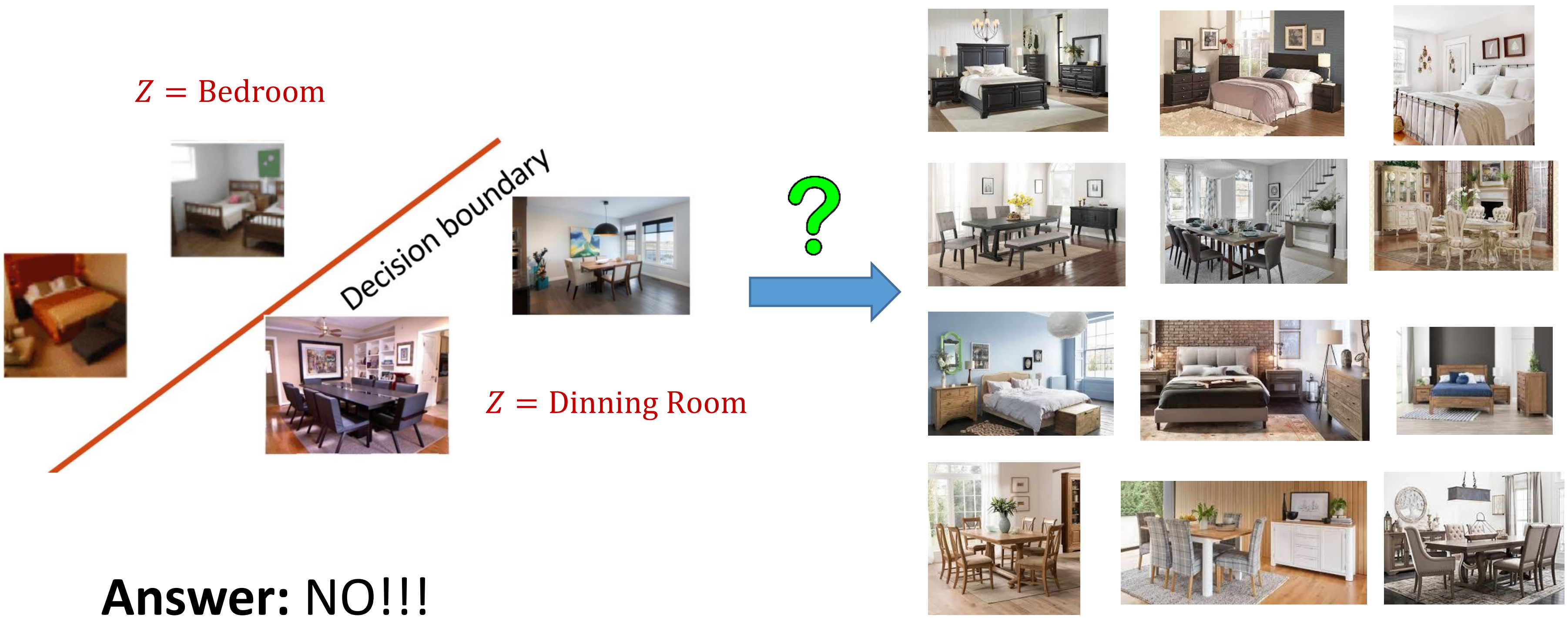
$p(Z = \text{Bedroom} \mid X =$  $)$



$p(Z = \text{Dinning Room} \mid X =$  $)$

Discriminative Models

Question: Can we generate samples of new images from the posterior $p(Z \mid X)$?



Answer: NO!!!
Use a generative model instead.

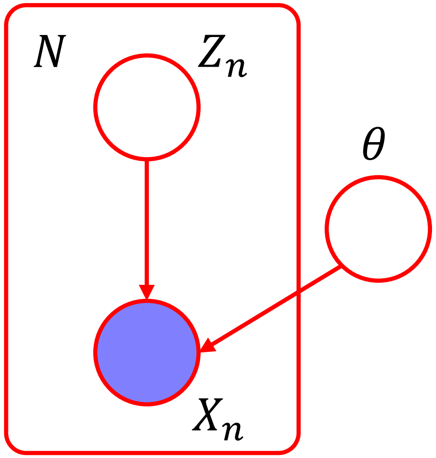
Generative Models

Question: Can we generate samples of new images?

Answer: Yes, use the likelihood model!

Consider the following graphical model, where the joint distribution is given by:

$$p(X, Z \mid \theta) = p(\underbrace{X \mid Z, \theta}_{\text{likelihood}})\underbrace{p(Z)}_{\text{prior}}$$



Z: latent variable

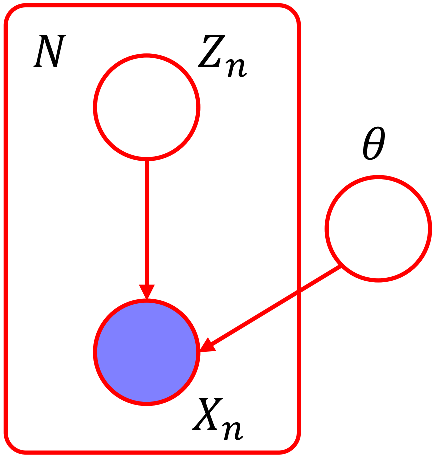
X: observed variable

θ: likelihood model parameter

Generative Models

Consider the following graphical model, where the joint distribution is given by:

$$p(X, Z \mid \theta) = \underbrace{p(X \mid Z, \theta)}_{\text{likelihood}}\underbrace{p(Z)}_{\text{prior}}$$



A simple idea:

1. Sample from the prior $z \sim p(Z)$, e.g., $z = \text{Bedroom}$
2. Generate new image from the likelihood $p(X \mid Z = \text{Bedroom})$

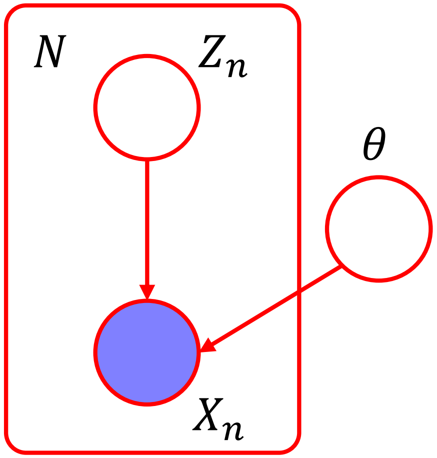
Variational AutoEncoder

- **Goal:** ensure there is one (or many) settings of Z_n to generate something very similar to each X_n in the dataset.

- To this end, let us define:

1. the prior $p(Z)$ over a high-dimensional space $\mathcal{Z}$, where samples of $\mathcal{Z}$ can be easily drawn.

2. a family of deterministic functions $f(Z;\theta): \mathcal{Z} \times \theta \rightarrow \mathcal{X}$.



Remarks: f is deterministic, but Z is random and θ is fixed, then $f(Z;\theta)$ is a random variable in the space $\mathcal{X}$.

Variational AutoEncoder

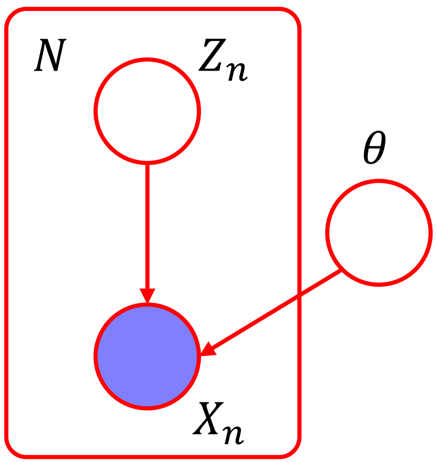
- **Objective:** optimize θ such that we can sample $z \sim p(Z)$ and, with high probability, $f(Z;\ \theta)$ will be like the X's in our dataset.
- This can be achieved by maximizing the likelihood, i.e,

$$p(X \mid \theta) = \int p(X \mid Z, \theta) p(Z) dZ,$$

where we define $p(X \mid Z, \theta)$ to be a Gaussian distribution, i.e.,

$$p(X \mid Z, \theta) = \mathcal{N}(X \mid f(Z; \theta), \sigma^2 I),$$

with mean $f(Z;\ \theta)$ and covariance equal to the identity matrix I times some scalar σ^2.

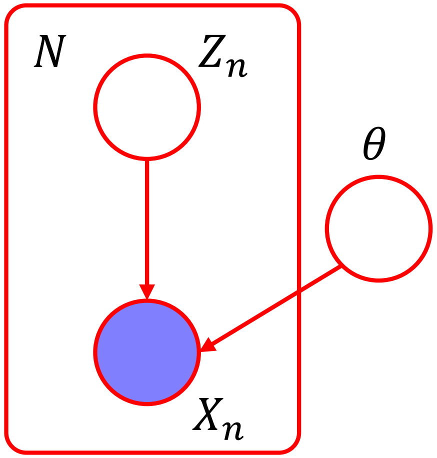


Variational AutoEncoder

- Two problems in solving the maximum likelihood:

$$p(X \mid \theta) = \int p(X \mid Z, \theta) p(Z) dZ$$

1. How to choose the latent variables Z such that we capture latent information?
2. How to deal with the integral over Z?

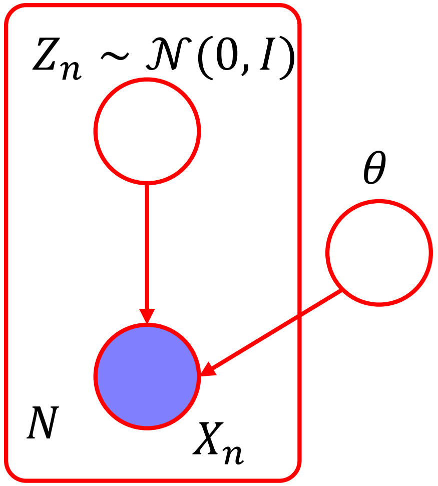


Choice of Latent Variable

- It is impossible to handcraft Z, since $z \sim p(Z)$ determines the highly complex image outputs X.
- VAEs assert that samples of Z can be drawn from a simple distribution, i.e., $p(Z) = \mathcal{N}(0, I)$, where I is the identity matrix.
- This is possible only if we learn $f(Z; \theta)$ in

$$p(X \mid Z, \theta) = \mathcal{N}(X \mid f(Z; \theta), \sigma^2 I)$$

with a powerful function approximator, e.g. a deep neural network!



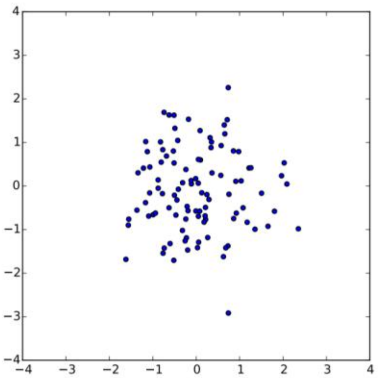
Choice of Latent Variable

Key idea: a sophisticated $f(Z;\theta)$ learned by deep learning can map any $z \sim \mathcal{N}(0, I)$ to X.

Example:

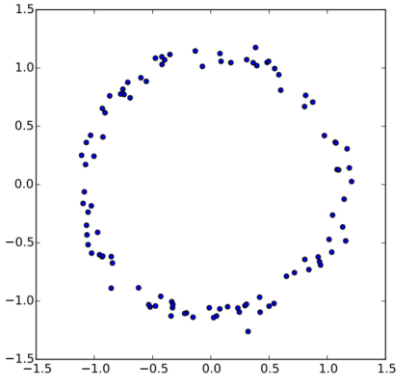
Given a random variable $z \sim \mathcal{N}(0, I)$, we can create another random variable $X = g(z)$ with a completely different distribution.

$z \sim \mathcal{N}(0, I)$



$$g(z) = \frac{z}{10} + \frac{z}{\|z\|}$$

$X = g(z)$



Choice of Latent Variable

Key idea: a sophisticated $f(Z;\theta)$ learned by deep learning can map any $z \sim \mathcal{N}(0, I)$ to X.

Remarks:

1. Let $f(Z;\ \theta)$ be a multi-layer neural network, where θ is the learnable parameters.
2. We can ensure $f(Z;\theta)\colon \mathcal{z} \times \theta \rightarrow \mathcal{X}$ by considering $\ln p(X \mid Z, \theta)$ in the loss function, i.e.,

$$\begin{aligned}\ln p(X \mid Z, \theta) &= \ln \mathcal{N}(X \mid f(Z;\theta), \sigma^2 I) \\ &= \ln\{\frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{\left(X - f(Z;\theta)\right)^2}{\sigma^2}\} \\ &= -\underbrace{\|X - f(Z;\theta)\|_2^2}_{} + \text{const.}\end{aligned}$$

L_2loss term that forces outputs of $f(Z;\theta)$ to be close to X!

Maximum Log-Likelihood

- Now all that remains is to maximize the log-likelihood, i.e.,

$$\underset{\theta}{\operatorname{argmax}} \ln p(X \mid \theta) = \underset{\theta}{\operatorname{argmax}} \ln \int p(X \mid Z, \theta) p(Z) dZ,$$

where

$$p(Z) = \mathcal{N}(0, I), \qquad \text{and}$$
$$p(X \mid Z, \theta) = \mathcal{N}(X \mid f(Z; \theta), \sigma^2 I).$$

- **A straightforward solution?** Approximate with log-likelihood with samples of $z \sim p(Z)$:

$$p(X) \approx \frac{1}{N} \sum_{i=1}^{N} p(X | f(z_i; \theta)) \ .$$

- **Problem:** X is in high dimensional spaces, N might need to be extremely large and $p(X|f(z_i;\theta)) \approx 0$ for most samples z.

Variational AutoEncoder

- **Solution:** sample Z that are likely to produce X, and compute $p(X)$ just from these samples.
- We define $q(Z \mid X)$ which takes a value of X and gives a distribution over Z values that are likely to produce X.
- To this end, we introduce $q(Z \mid X)$ into the log likelihood:

$$
\begin{aligned}
\ln p(X) &= \sum_Z q\,(Z \mid X) \ln p(X) \\
&= \sum_Z q(Z \mid X) \ln \frac{p(X,Z)q(z|X)p(X)}{p(X,Z)q(z|X)} \\
&= \sum_Z q(Z \mid X) \ln \frac{p(X,Z)}{q(z|X)} + \sum_Z q(Z \mid X) \ln \frac{q(z|X)p(X)}{p(X,Z)} \\
&= \sum_Z q(Z \mid X) \ln \frac{p(X,Z)}{q(z|X)} + \sum_Z q(Z \mid X) \ln \frac{q(z|X)}{p(Z|X)}
\end{aligned}
$$

Variational AutoEncoder

$$\ln p(X) = \underbrace{\sum_Z q(Z \mid X) \ln \frac{p(X,Z)}{q(z|X)}}_{\mathcal{L}(q,\theta)} + \underbrace{\sum_Z q(Z \mid X) \ln \frac{q(z|X)}{p(Z|X)}}_{KL[q(Z \parallel X) \parallel p(Z \parallel X)]}$$

- Maximizing the log-likelihood is equivalent to maximizing the lower bound $\mathcal{L}(q,\theta)$ and KL-divergence $KL[q(Z \parallel X) \parallel p(Z \parallel X)]$.
- **Problem**: we cannot maximize the KL-divergence $KL[q(Z \parallel X) \parallel p(Z \parallel X)]$ since we do not know $p(Z \parallel X)$!
- Solution: let's maximize the lower bound $\mathcal{L}(q,\theta)$ instead.

Variational AutoEncoder

Rearranging the log likelihood terms to make $\mathcal{L}(q,\theta)$ the subject:

$$\ln p(X) = \underbrace{\textstyle\sum_Z q(Z \mid X) \ln \frac{p(X,Z)}{q(z|X)}}_{\mathcal{L}(q,\theta)} + \underbrace{\textstyle\sum_Z q(Z \mid X) \ln \frac{q(z|X)}{p(Z|X)}}_{KL[q(Z \parallel X) \parallel p(Z \parallel X)]}$$

We get:

$$\ln p(X) - KL\big(q(Z \mid X) \parallel p(Z \mid X)\big) = \textstyle\sum_Z q(Z \mid X) \ln \frac{p(X,Z)}{q(z|X)}$$

$$= \textstyle\sum_Z q(Z \mid X) \ln \frac{p(X|Z)p(Z)}{q(z|X)}$$

$$= \textstyle\sum_Z q(Z \mid X) \ln p(X \mid Z) + \sum_Z q(Z \mid X) \ln \frac{p(Z)}{q(z|X)}$$

$$= \mathbb{E}_{z \sim q(Z|X)} \ln p(X \mid Z) - KL[q(Z \mid X) \parallel p(Z)]$$

VAE: Maximizing the Lower Bound

$$\ln p(X) - \overbrace{\boxed{KL\big(q(Z \mid X) \parallel p(Z \mid X)\big)}}^{\geq 0}$$

$$= \underbrace{\mathbb{E}_{Z\sim q(Z|X)} \ln p(X \mid Z) - KL[q(Z \mid X) \parallel p(Z)]}_{\mathcal{L}(q,\theta)}$$

- Maximizing the lower bound is similar to maximizing the log likelihood and minimizing the KL-divergence.
- Minimizing $KL[q(Z \mid X) \parallel p(Z \mid X)]$means we are forcing $q(Z \mid X)$ to be close to $p(Z \mid X)$!

VAE: Loss Function

- VAE's loss function is given by the negative lower bound, which we minimize using stochastic gradient descent.

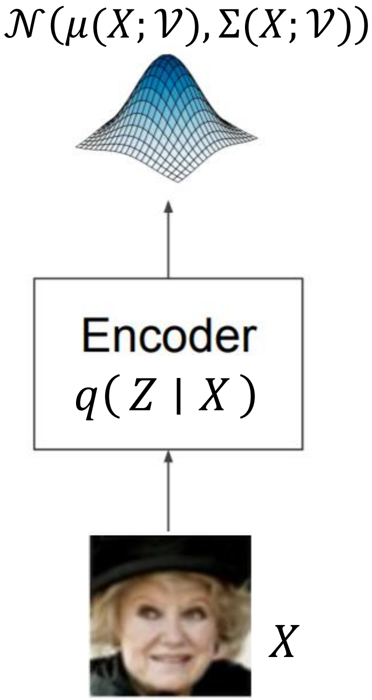
$$\text{Loss} = -\mathbb{E}_{z \sim q(Z|X)} \ln \underbrace{\boxed{p(X \mid Z)}}_{\text{Decoder}} + KL[\underbrace{\boxed{q(Z \mid X)}}_{\text{Encoder}} \parallel p(Z)]$$

- Now we can see the autoencoder, since $q(Z \mid X)$ is "encoding" X into Z, and $p(X \mid Z)$ is "decoding" it to reconstruct X.

VAE: Loss Function

$$\text{Loss} = -\mathbb{E}_{z\sim q(Z|X)} \ln p(X \mid Z) + \boxed{KL[q(Z \mid X) \parallel p(Z)]}$$

- Consider the second term of the loss function, we model the encoder $q(Z \mid X)$ with a neural network parameterized by $\mathcal{V}$.
- Assume Gaussian $q(Z \mid X) = \mathcal{N}(\mu(X;\mathcal{V}),\Sigma(X;\mathcal{V}))$, i.e., a neural network that outputs the mean $\mu(X)$, and diagonal covariance matrix $\Sigma(X;\mathcal{V})$.
- **Encoder:** Input is an Image X, output is a Gaussian distribution $\mathcal{N}(\mu(X;\mathcal{V}),\Sigma(X;\mathcal{V}))$.



VAE: Loss Function

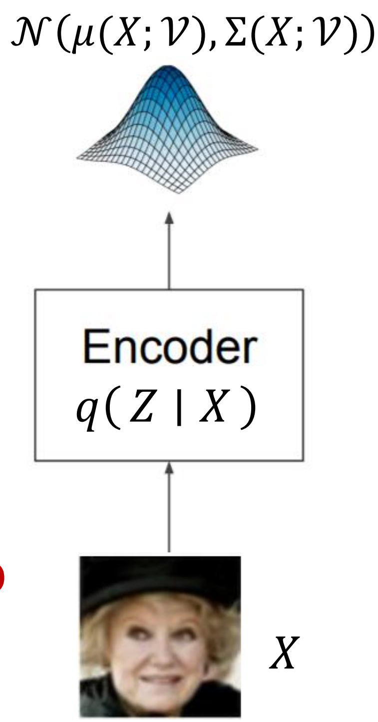
$$\text{Loss} = -\mathbb{E}_{z\sim q(Z|X)} \ln p(X \mid Z) + \boxed{KL[q(Z \mid X) \parallel p(Z)]}$$

- Recall we defined $p(Z) = \mathcal{N}(0, I)$, the second loss term can now be rewritten as:

$$KL[\mathcal{N}(\mu(X;\mathcal{V}), \Sigma(X;\mathcal{V})) \parallel \mathcal{N}(0, I)]$$

$$= \frac{1}{2}\left(\text{trace}\big(\Sigma(X;\mathcal{V})\big) + \big(\mu(X;\mathcal{V})\big)^T \big(\mu(X;\mathcal{V})\big) - k - \log\det(\Sigma(X;\mathcal{V}))\right),$$

where k is the dimensionality of the distribution.



- **Insight**: This loss forces the latent space to be close to the normal distribution!

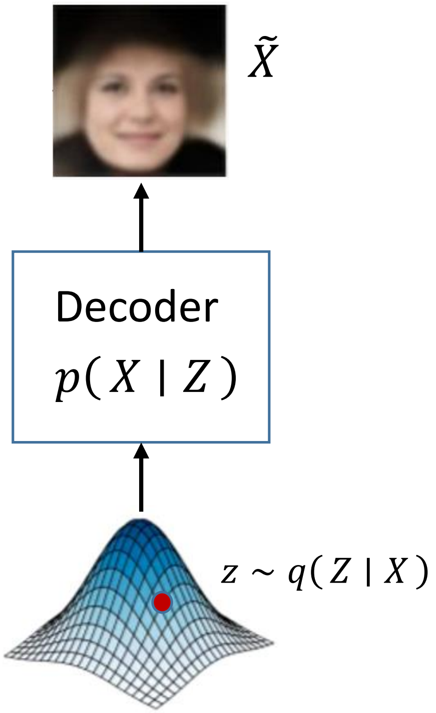
VAE: Loss Function

$$\text{Loss} = \boxed{-\mathbb{E}_{z \sim q(Z|X)} \ln p(X \mid Z)} + KL[q(Z \mid X) \parallel p(Z)]$$

- Consider the first term of the loss function, recall that we model the decoder $p(X \mid Z)$ as:

$$p(X \mid Z, \theta) = \mathcal{N}(X \mid f(Z;\theta), \sigma^2 I)$$

- where $f(Z;\theta)$ is a learnable neural network parameterized by θ.
- **Decoder:** Input is a sample from the latent space Z, output is the reconstructed image $\tilde{X} = f(Z;\theta)$.



VAE: Loss Function

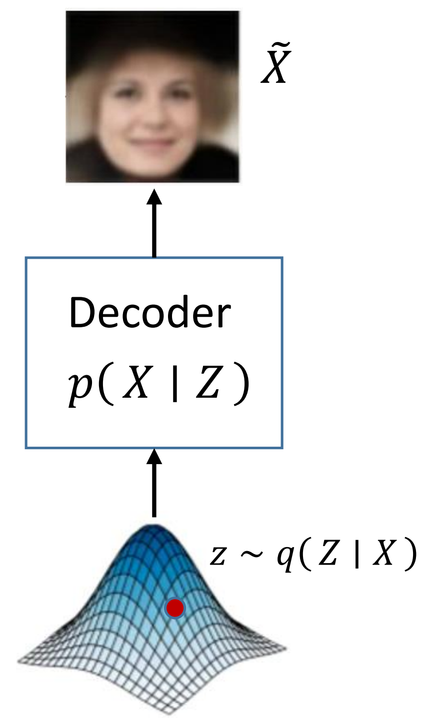
$$\text{Loss} = \boxed{-\mathbb{E}_{z\sim q(Z|X)} \ln p(X \mid Z)} + KL[q(Z \mid X) \parallel p(Z)]$$

- This leads to the square loss as mentioned earlier:

$$-\mathbb{E}_{z\sim q(Z|X)} \ln p(X \mid Z) = -\mathbb{E}_{z\sim q(Z|X)} \ln \mathcal{N}(X \mid f(Z;\theta), \sigma^2 I)$$

$$= -\mathbb{E}_{z\sim q(Z|X)}[\ln \left\{ \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{\left(X - f(Z;\theta)\right)^2}{\sigma^2} \right\}]$$

$$= \mathbb{E}_{z\sim q(Z|X)} \| X - \underbrace{f(Z;\theta)}_{\tilde{X}} \|_2^2 + \text{const.}$$



- This is the expected image reconstruction loss!

VAE: Optimizing the Loss Function

- Putting the loss terms together, we get:

$$\text{Loss} = -\mathbb{E}_{z \sim q(Z|X)} \ln p(X \mid Z) + KL[q(Z \mid X) \parallel p(Z)]$$
$$= \mathbb{E}_{z \sim q(Z|X)} \|X - f(Z;\theta)\|_2^2 + KL[\mathcal{N}(\mu(X), \Sigma(X)) \parallel \mathcal{N}(0, I)]$$

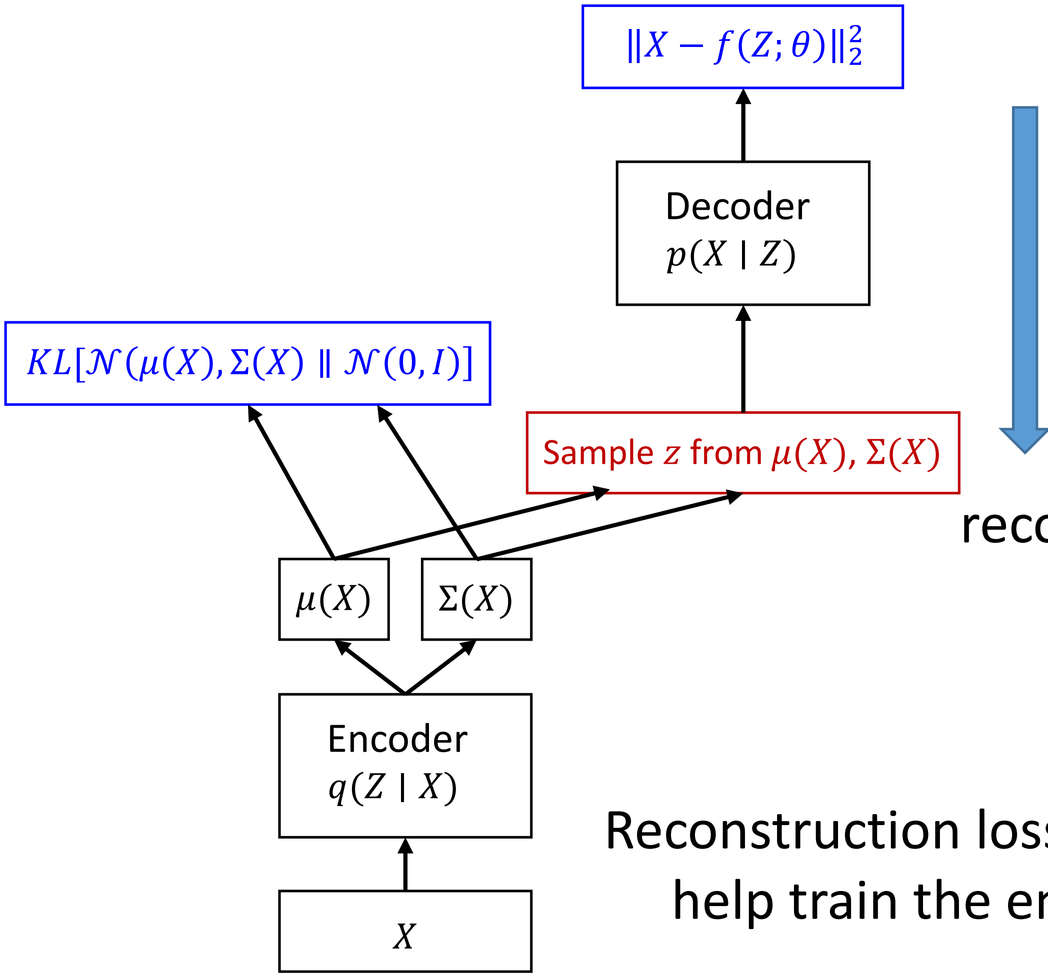
- **An easy approach?** We can optimize the loss by computing the gradient over every sample $X \sim \mathcal{D}$ and $z \sim Q(Z \mid X)$.

$$\text{Loss} = \mathbb{E}_{X \sim \mathcal{D}} \left\{ -\mathbb{E}_{z \sim q(Z|X)} \ln p(X \mid Z) + KL[q(Z \mid X) \parallel p(Z)] \right\}$$

- **Problem:** Gradient is now independent of encoder!

$$\text{Gradient} = \nabla(\underbrace{\log p(X \mid Z)}_{q(Z \mid X) \text{ is not trained!}} - KL[q(Z \mid X) \parallel p(Z)])$$

VAE: Optimizing the Loss Function



Gradient from the reconstruction loss stops here!

Reconstruction loss does not help train the encoder!

VAE: Optimizing the Loss Function

Reparameterization Trick

- **Solution:** Move the sampling to an input layer!

- Given the mean $\mu(X)$ and covariance $\Sigma(X)$ of $q(Z \mid X)$, we can sample from $\mathcal{N}(\mu(X), \Sigma(X))$ by:

  1. sampling $\varepsilon \sim \mathcal{N}(0, I)$, then
  2. compute $z = \mu(X) + \Sigma^{1/2}(X) * \varepsilon$

- The loss and gradient become:

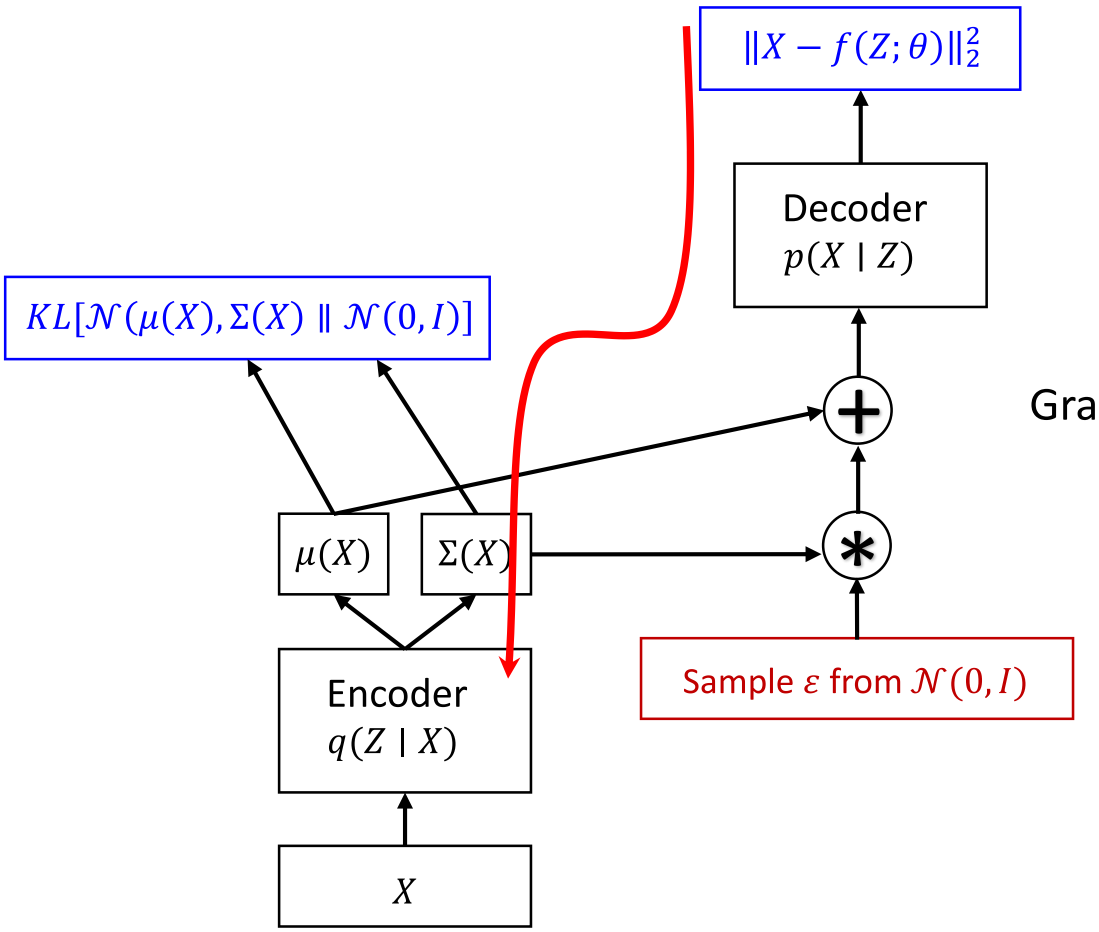
Kept fixed

$$\text{Loss} = -\mathbb{E}_{X\sim\mathcal{D}}\{-\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)} \ln p\big(X \mid z = \mu(X) + \Sigma^{1/2}(X) * \varepsilon\big) + KL[q(Z \mid X) \,\|\, p(Z)]\}$$

$$\text{Gradient} = \nabla\big(\log p\big(X \mid \underbrace{z = \mu(X) + \Sigma^{1/2}(X) * \varepsilon}_{\text{function of } q(Z \mid X)!}\big) - KL[q(Z \mid X) \,\|\, p(Z)]\big)$$

VAE: Optimizing the Loss Function

Reparameterization Trick



Gradient of the reconstruction loss flows to $q(Z \mid X)$!

VAE Training

Given: a dataset of examples $X = \{X_1, X_2, \dots\}$.

1. Initialize parameters for Encoder and Decoder
2. Repeat till convergence:

    i. $X^M \leftarrow$ Random minibatch of M examples from X

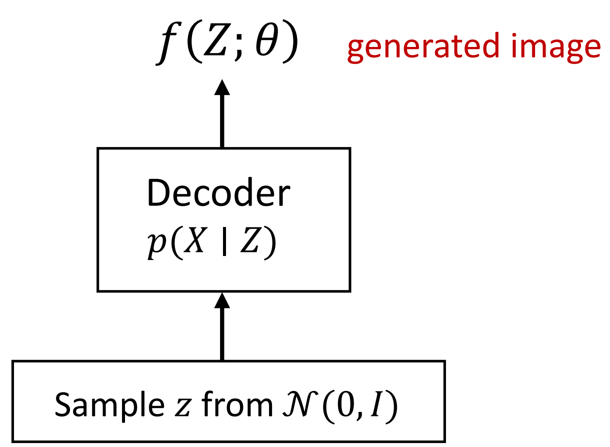
    ii. $\varepsilon \leftarrow$ Sample M noise vectors from $\mathcal{N}(0, I)$

    iii. Compute $L(X^M, \varepsilon, \theta)$ (i.e. run a forward pass in the neural network)

    iv. Gradient descent on L to update the Encoder and Decoder (backpropagation).

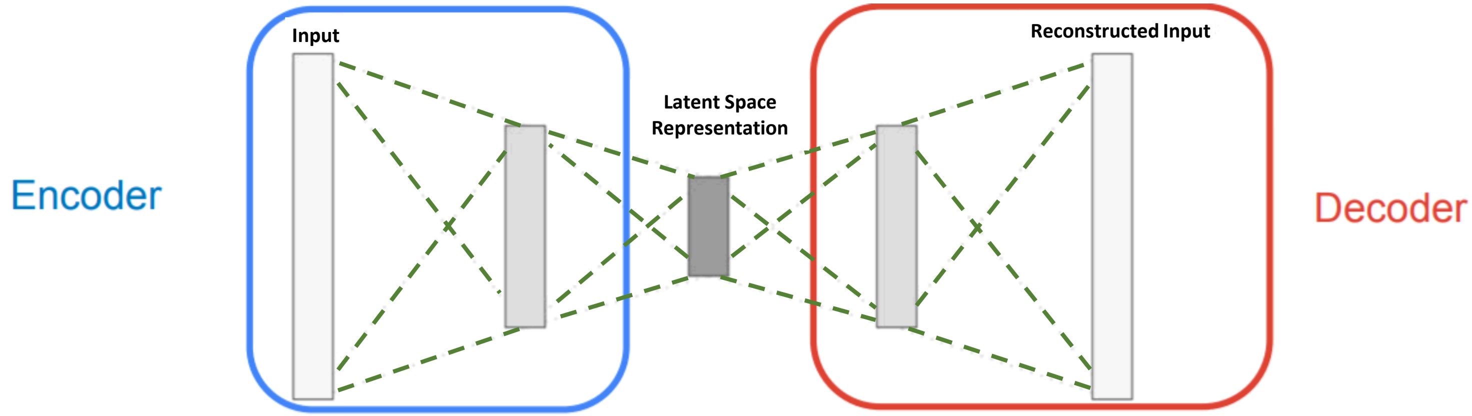
VAE Testing

- At test-time, we want to evaluate the performance of VAE to generate a new sample.
- Remove the Encoder since no test-image for the generation task.
- Sample $z \sim \mathcal{N}(0, I)$ and pass it through the Decoder.
- No good quantitative metric, relies on visual inspection.

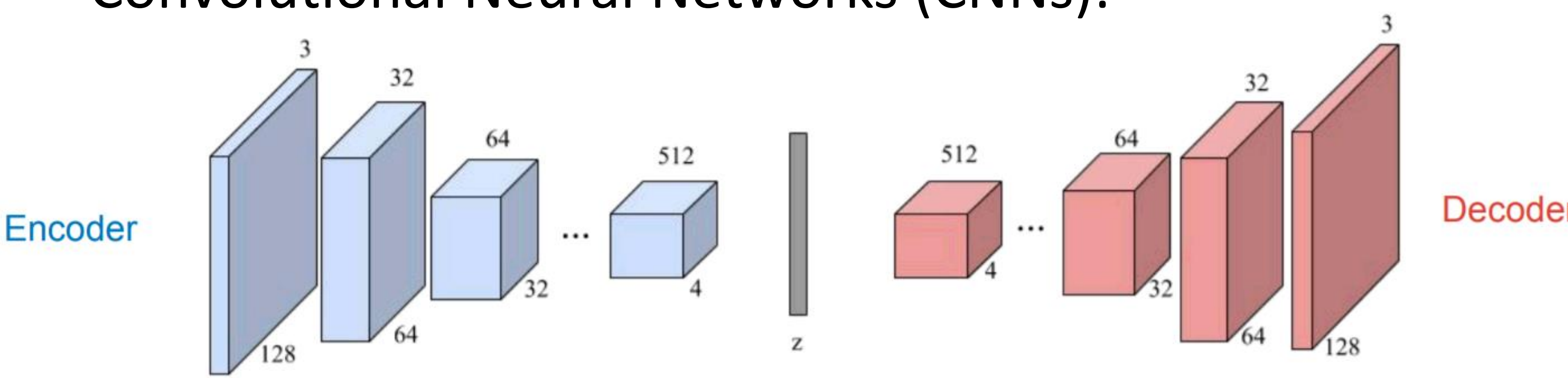


Common VAE Architecture

- Multi-Layer Perceptrons (MLPs):



- Convolutional Neural Networks (CNNs):



Disentangle latent factor

- Visualizations of learned data manifold for generative models with two-dimensional latent space.
- We can see that the generated images are distributed according to the disentangled latent factor.

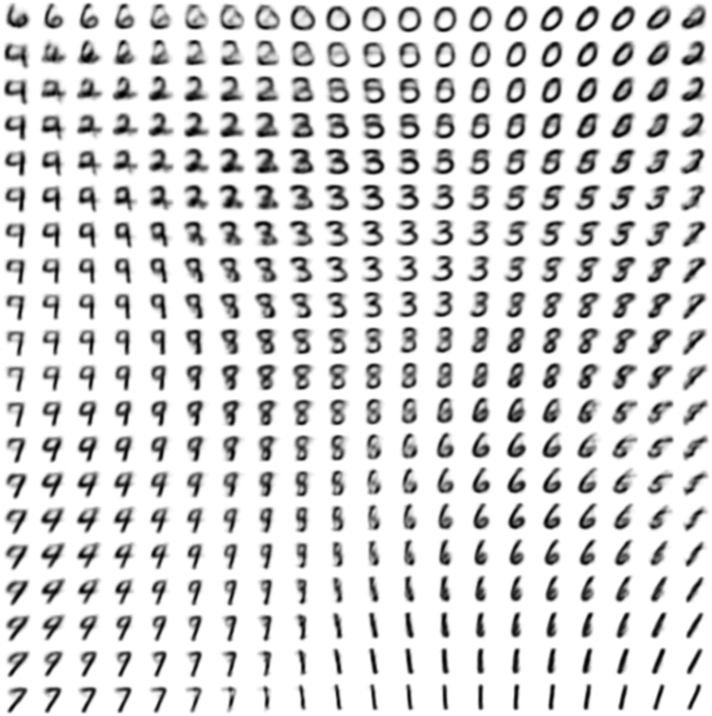


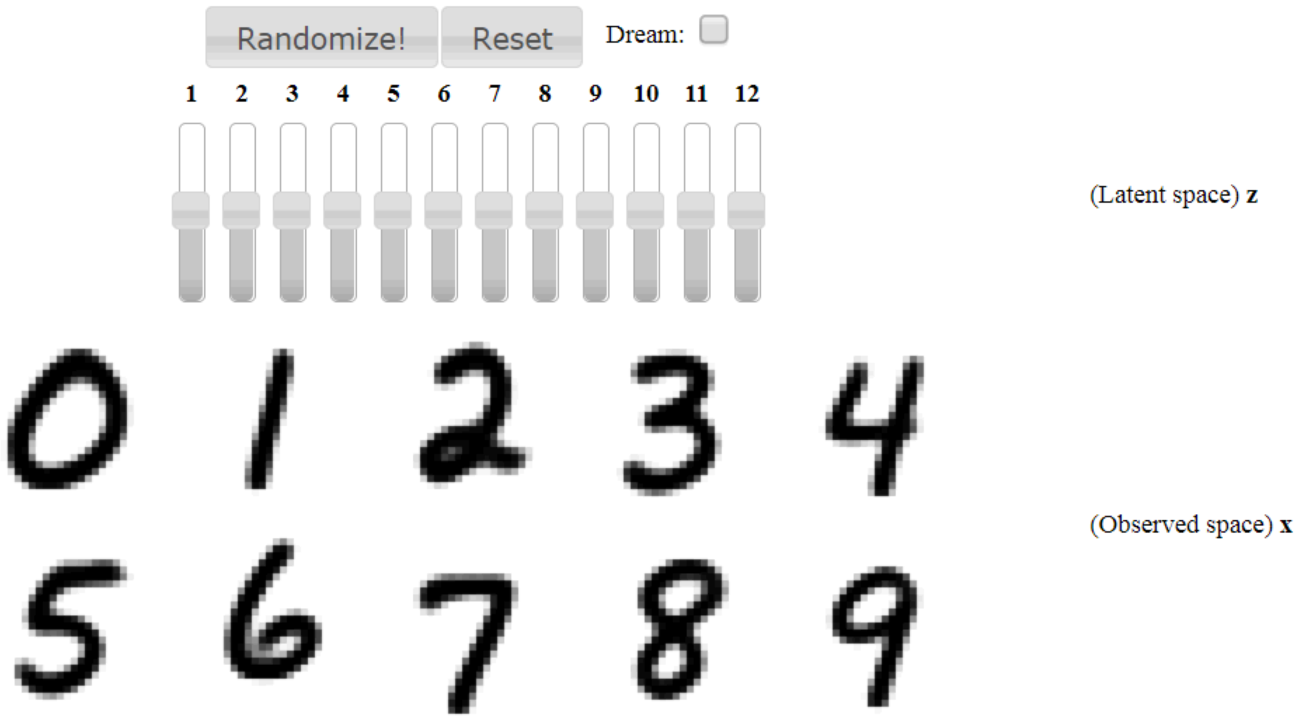
Image Source: Kingmal and Welling, Auto-Encoding Variational Bayes, ICLR'14

Disentangle latent factor

Digit Fantasies by a Deep Generative Model

Instructions:

1. Dream mode: check 'dream' to let the model fantasize digits.
2. Alternatively, you can wiggle the sliders yourselves to wander through **z**-space and observe the effects in **x**-space.



http://www.dpkingma.com/sgvb_mnist_demo/demo.html

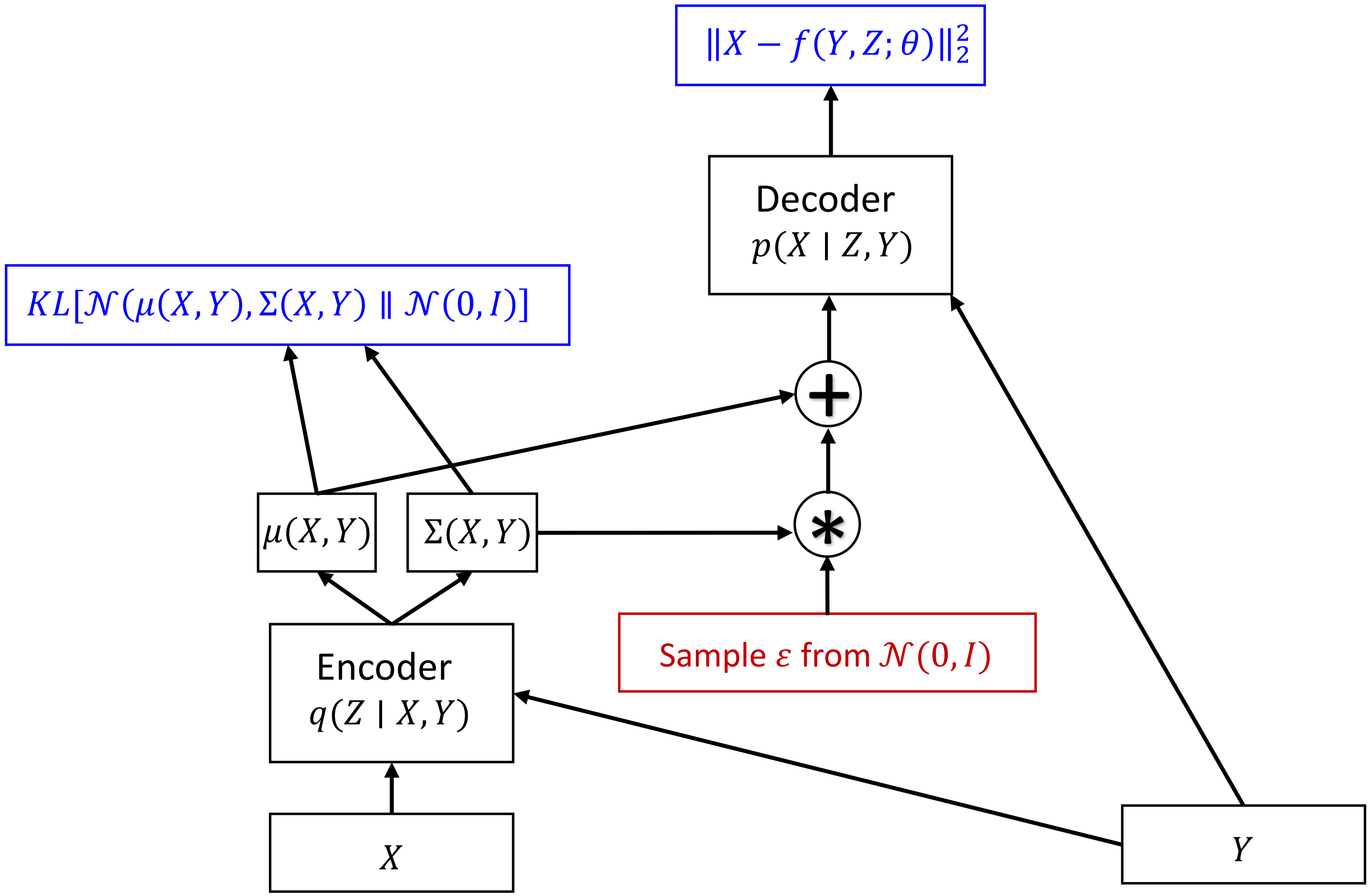
Conditional VAE (CVAE)

- What if we have labels? (e.g. digit labels or attributes) Or other inputs we wish to condition on (Y).

- None of the derivation changes:

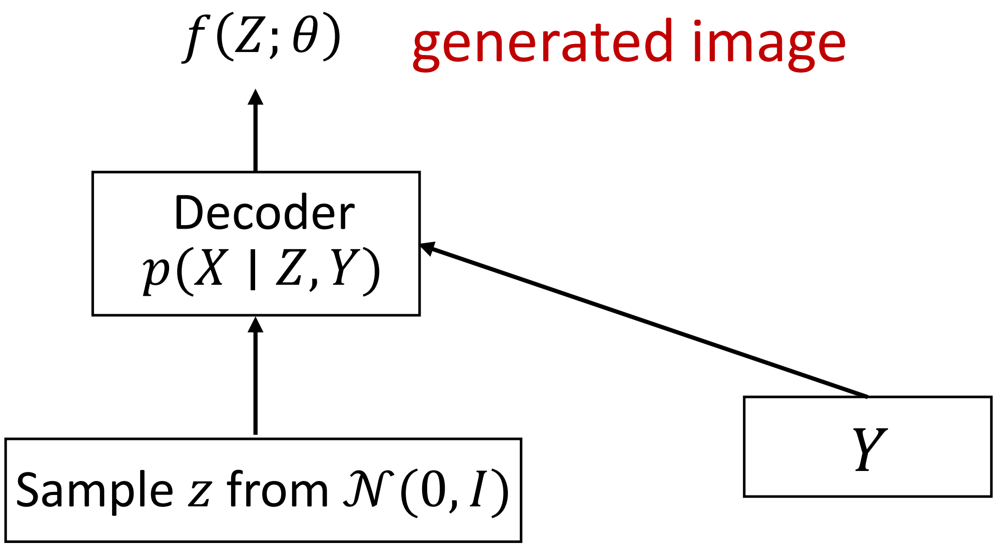
1. Replace all $p(X \mid Z)$ with $p(X \mid Z, Y)$
2. Replace all $q(Z \mid X)$ with $q(Z \mid X, Y)$
3. Go through the same KL-divergence procedure, to get the same lower bound.

Conditional VAE (CVAE)



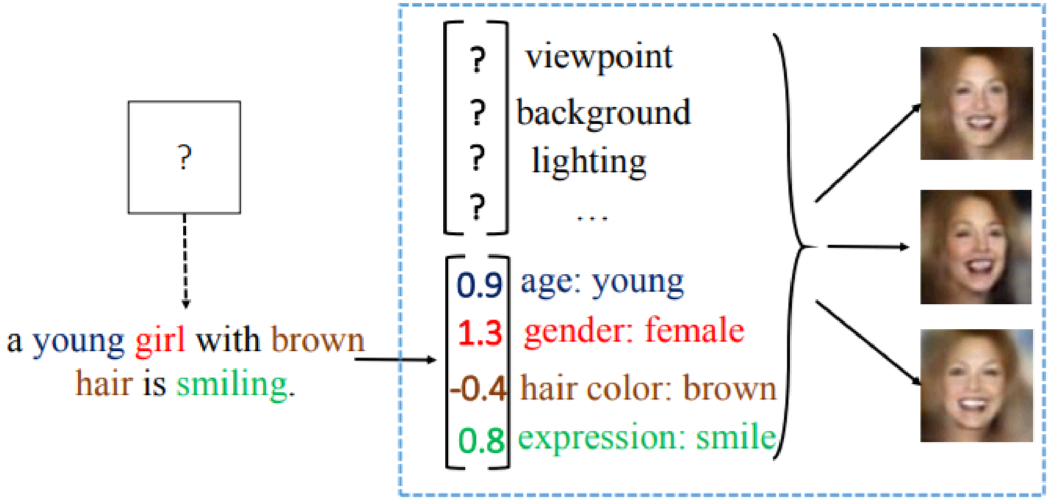
Conditional VAE (CVAE)

- Remove the Encoder at test time.
- Sample $z \sim \mathcal{N}(0, I)$ and input a desired Y to the Decoder.



CVAE Example: Conditioned Image Generation from Visual Attributes

- A vector of visual attributes is extracted from a natural language description.
- This attribute vector is then combined with learned latent factors to generate diverse image samples.



Attribute-conditioned Image Generation

Xinchen Yan, Jimei Yang, Kihyuk Sohn, Honglak Lee, Attribute2Image: Conditional Image Generation from Visual Attributes, ECCV 2016

Mixture Density Networks

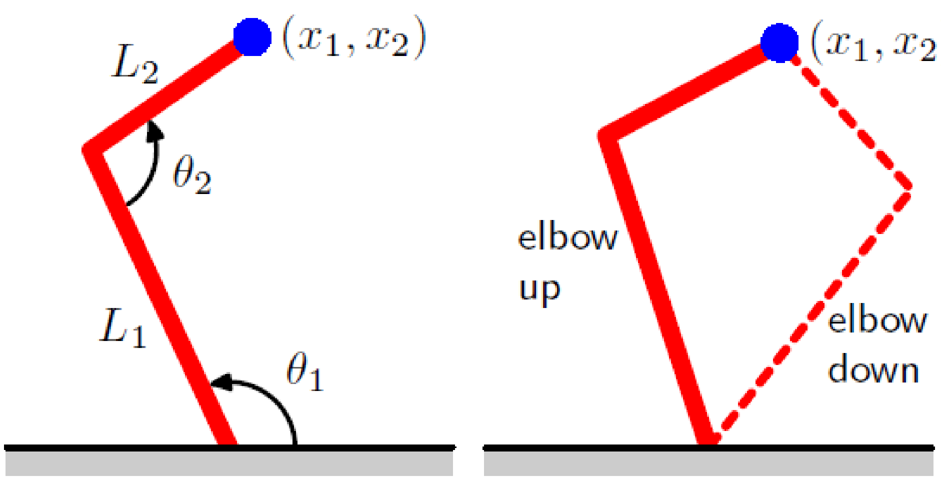
- For input x and output t, the goal of supervised deep learning is to model a conditional distribution $p(\mathrm{t} \mid \mathrm{x})$.
- $p(\mathrm{t} \mid \mathrm{x}) = \mathcal{N}(\mathrm{t} \mid \mathrm{x}, \sigma^2 * I)$ is chosen to be a unimodal Gaussian for many simple regression problems.
- This can lead to very poor predictions for *inverse problems* with multimodal distributions.

Forward vs Inverse Problem

Example 1: Consider the kinematics of a robot arm

- The *forward problem* is to find the end effector position (x_1, x_2) given the joint angles (θ_1, θ_2).
- This has a unique solution!

Forward problem



Inverse problem

Image Source: "Pattern Recognition and Machine Learning", Christopher Bishop

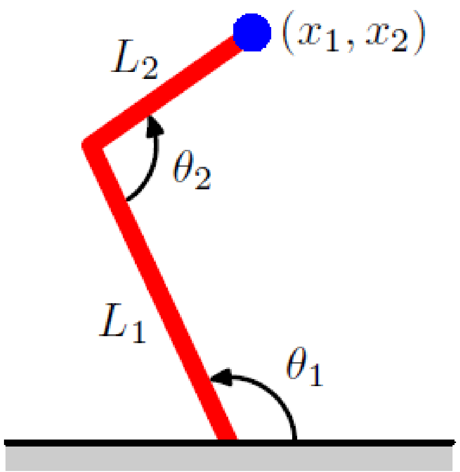
Forward vs Inverse Problem

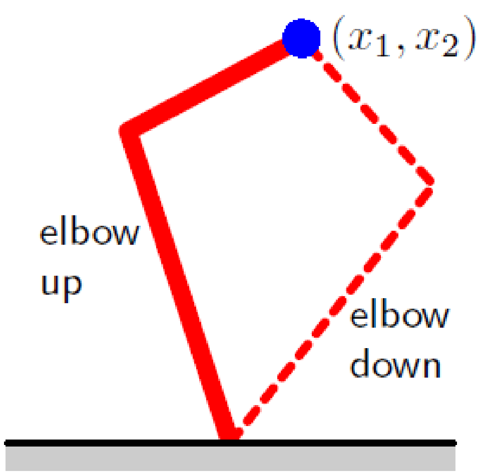
Example 1: Consider the kinematics of a robot arm

- In practise, we might solve the *inverse problem* that finds the appropriate joint angles to move the end effector to a specific position.

- Multiple feasible solutions might exist!

Forward problem





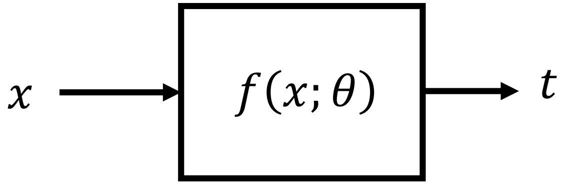
Inverse problem

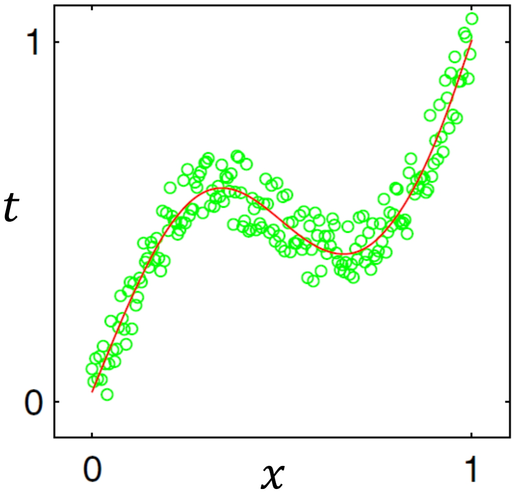
Image Source: “Pattern Recognition and Machine Learning”, Christopher Bishop

Forward vs Inverse Problem

Example 2: Simple toy problem to visualize multimodality

- Given observations (green circles) simulated from $t_n = x_n + 0.3\sin(2\pi x_n) + \text{uniform}(-0.1,0.1)$.
- The *forward problem* is to train a deep network with learnable parameters θ to find t (red curve) given x, i.e., $t = f(x;\theta)$.





Forward problem:
Unique t_n for every x_n!

Image Source: "Pattern Recognition and Machine Learning", Christopher Bishop

Forward vs Inverse Problem

Example 2: Simple toy problem to visualize multimodality

- The *inverse problem* is then obtained by keeping the same data points but exchanging the roles of x and t.
- Red curve shows the results of fitting two-layer neural networks by minimizing a sum-of-squares error function.

Inverse problem: Multiple t_n could exist for every x_n!

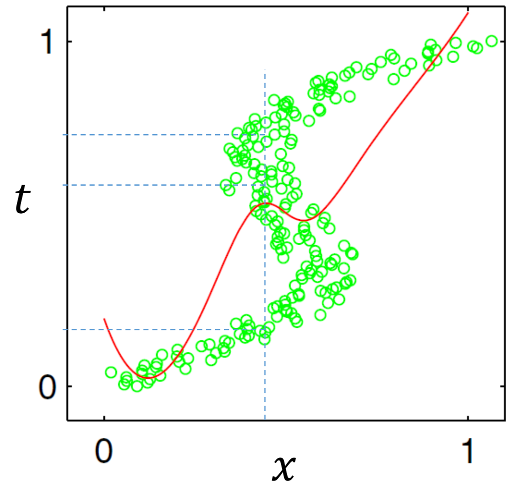


Image Source: "Pattern Recognition and Machine Learning", Christopher Bishop

Forward vs Inverse Problem

Example 2: Simple toy problem to visualize multimodality

- Least-squares corresponds to maximum likelihood under a Gaussian assumption.
- This leads to a very poor model for the highly non-Gaussian inverse problem.

Inverse problem: Multiple t_n could exist for every x_n!

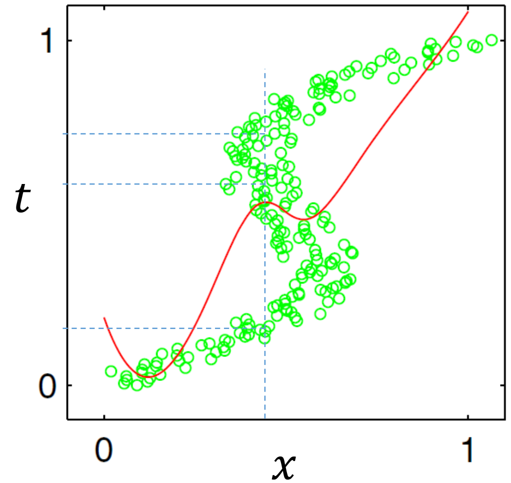


Image Source: "Pattern Recognition and Machine Learning", Christopher Bishop

Mixture Density Networks

- To model the inverse problem with multiple feasible solutions, we use a mixture model for $p(\mathrm{t} \mid \mathrm{x})$, i.e.,

$$p(\mathbf{t}|\mathbf{x}) = \sum_{k=1}^{K} \pi_k(\mathbf{x}) \mathcal{N}\left(\mathbf{t}|\boldsymbol{\mu}_k(\mathbf{x}), \sigma_k^2(\mathbf{x})\right),$$

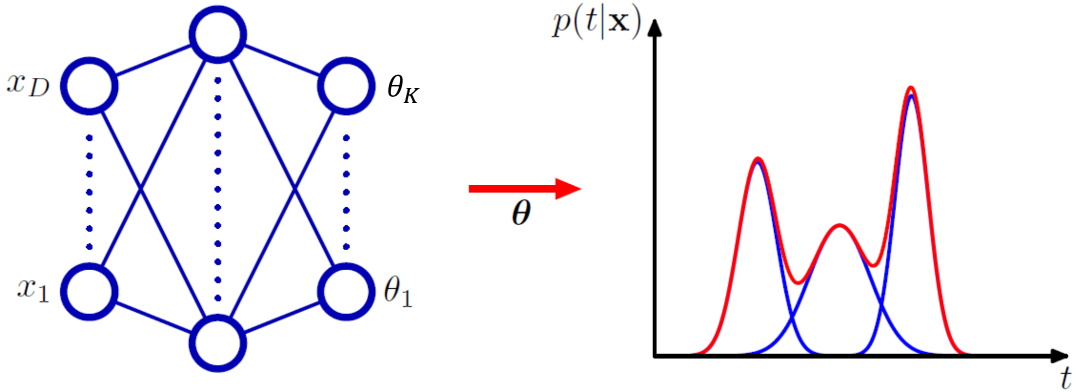
where

$\pi_k(\mathrm{x})$: mixing coefficients,
$\mu_k(\mathrm{x})$: means, and the
$\sigma_k^2(\mathrm{x})$: variances.

- The parameters $\theta = \{\pi(\mathrm{x}), \mu(\mathrm{x}), \sigma^2(\mathrm{x})\}$ are outputs of a conventional neural network that takes x as input.

Mixture Density Networks

Example: Two-layer Mixture Density Network with sigmoidal ('tanh') hidden units.



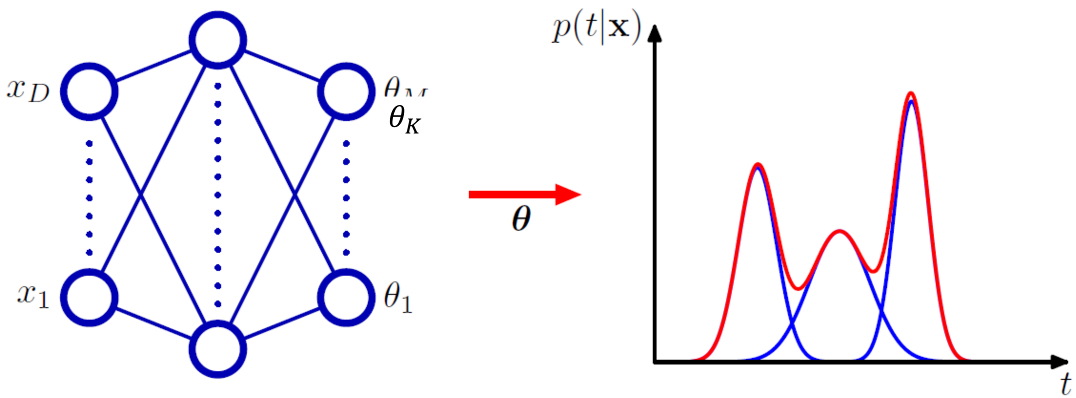
- For a K-component mixture model and $\mathrm{t} \in \mathbb{R}^L$, the network have:

1. K outputs denoted by a_k^π that determine the mixing coefficients $\pi_k(\mathrm{x})$,
2. K outputs denoted by a_k^σ that determine the kernel widths $\sigma_k(\mathrm{x})$, and

Image Source: "Pattern Recognition and Machine Learning", Christopher Bishop

Mixture Density Networks

Example: Two-layer Mixture Density Network with sigmoidal ('tanh') hidden units.



- For a K-component mixture model and L-dimensional t, the network have:

3. $K \times L$ outputs denoted by a_{kl}^{μ} that determine the components $\mu_{kl}(\mathrm{x})$ of the kernel centres $\mu_k(\mathrm{x})$.

- The total number of outputs from the MDN is given by $(L + 2)K$, as compared with the usual L outputs for a network.

Image Source: "Pattern Recognition and Machine Learning", Christopher Bishop

Mixture Density Networks: Constraints on the Outputs

- The mixing coefficients must satisfy the constraints:

$$\sum_{k=1}^{K} \pi_k(\mathbf{x}) = 1, \qquad 0 \leqslant \pi_k(\mathbf{x}) \leqslant 1$$

- which can be achieved using a set of softmax outputs:

$$\pi_k(\mathbf{x}) = \frac{\exp(a_k^{\pi})}{\sum_{l=1}^{K} \exp(a_l^{\pi})}.$$

Mixture Density Networks: Constraints on the Outputs

- The variances must satisfy $\sigma_k^2(\mathrm{x}) \geq 0$ and can be represented as exponentials of the corresponding network activations using:

$$\sigma_k(\mathbf{x}) = \exp(a_k^\sigma).$$

- Finally, because the means $\mu_k(\mathrm{x})$ have real components, they can be represented directly by the network output activations:

$$\mu_{kj}(\mathbf{x}) = a_{kj}^\mu.$$

Mixture Density Networks: Loss Function

- The parameters **w** of the mixture density network can be learned by maximum likelihood:

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{k} \pi_k(\mathbf{x}_n, \mathbf{w}) \mathcal{N}\left(\mathbf{t}_n | \boldsymbol{\mu}_k(\mathbf{x}_n, \mathbf{w}), \sigma_k^2(\mathbf{x}_n, \mathbf{w})\right) \right\}$$

where

- $\{t_1, \ldots, t_N\}$: i.i.d training data
- $\{\pi_1(\mathrm{x}_n, \mathrm{w}), \ldots, \pi_K(\mathrm{x}_n, \mathrm{w})\}$: output mixing coefficients
- $\{\mu_1(\mathrm{x}_n, \mathrm{w}), \ldots, \mu_K(\mathrm{x}_n, \mathrm{w})\}$: output means
- $\{\sigma_1^2(\mathrm{x}_n, \mathrm{w}), \ldots, \sigma_K^2(\mathrm{x}_n, \mathrm{w})\}$: output variances

Mixture Density Networks: Optimizing the Loss

- Mixing coefficients $\pi_k(\mathrm{x})$ can be viewed as x-dependent prior probabilities since we are dealing with mixture distributions.

- We introduce the corresponding posterior probabilities given by:

$$\gamma_k(\mathbf{t}|\mathbf{x}) = \frac{\pi_k \mathcal{N}_{nk}}{\sum_{l=1}^{K} \pi_l \mathcal{N}_{nl}}$$

where $\mathcal{N}_{nk}$ denotes $\mathcal{N}(\mathrm{t}_n \mid \mu_k(\mathrm{x}_n), \sigma_k^2(\mathrm{x}_n))$.

Mixture Density Networks: Optimizing the Loss

- The derivatives w.r.t. the network output activations governing the mixing coefficients are given by:

$$\frac{\partial E_n}{\partial a_k^{\pi}} = \pi_k - \gamma_k.$$

- Similarly, the derivatives w.r.t. the output activations controlling the component means are given by:

$$\frac{\partial E_n}{\partial a_{kl}^{\mu}} = \gamma_k \left\{ \frac{\mu_{kl} - t_l}{\sigma_k^2} \right\}.$$

- Finally, the derivatives with respect to the output activations controlling the component variances are given by:

$$\frac{\partial E_n}{\partial a_k^{\sigma}} = -\gamma_k \left\{ \frac{\|\mathbf{t} - \boldsymbol{\mu}_k\|^2}{\sigma_k^3} - \frac{1}{\sigma_k} \right\}.$$

Mixture Density Networks: Inference

- During inference, we use the trained MDN to estimate $p(\mathrm{t} \mid \mathrm{x})$ of the target data t for any given input x.
- $p(\mathrm{t} \mid \mathrm{x})$ represents a complete description of the generator of the data, with regards to predicting the output vector.
- We then use $p(\mathrm{t} \mid \mathrm{x})$ to calculate more specific quantities that may be of interest in different applications.

Mixture Density Networks: Inference

- One simple form is the mean, corresponding to the conditional average of the target data, and is given by:

$$\mathbb{E}\left[\mathbf{t}|\mathbf{x}\right] = \int \mathbf{t} p(\mathbf{t}|\mathbf{x})\, \mathrm{d}\mathbf{t} = \sum_{k=1}^{K} \pi_k(\mathbf{x}) \boldsymbol{\mu}_k(\mathbf{x})$$

- We can similarly evaluate the variance of the density function about the conditional average, to give:

$$\begin{aligned} s^2(\mathbf{x}) &= \mathbb{E}\left[\|\mathbf{t} - \mathbb{E}[\mathbf{t}|\mathbf{x}]\|^2 |\mathbf{x}\right] \\ &= \sum_{k=1}^{K} \pi_k(\mathbf{x}) \left\{ \sigma_k^2(\mathbf{x}) + \left\| \boldsymbol{\mu}_k(\mathbf{x}) - \sum_{l=1}^{K} \pi_l(\mathbf{x}) \boldsymbol{\mu}_l(\mathbf{x}) \right\|^2 \right\} \end{aligned}$$

Mixture Density Networks: Inference

- **Problem** with conditional average: the solution might not be physically feasible!
- E.g. the average of the two configurations of the robot arm is not a solution!

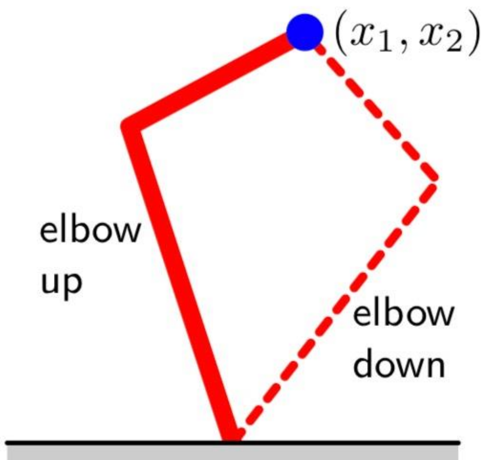


Image Source: "Pattern Recognition and Machine Learning", Christopher Bishop

Mixture Density Networks: Inference

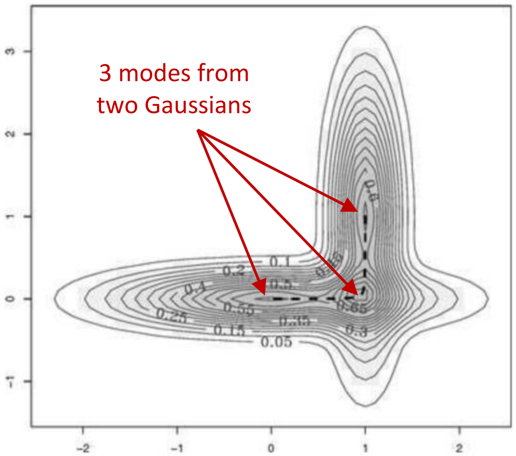
- In such cases, the conditional mode may be of more value.
- Unfortunately, the conditional mode for the mixture density network requires numerical iteration.

Example:

A two-component mixture of Gaussians with three modes.

$$\boldsymbol{\mu}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad \Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0.05 \end{pmatrix},$$

$$\boldsymbol{\mu}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \Sigma_2 = \begin{pmatrix} 0.05 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \pi_1 = \pi_2 = \frac{1}{2}.$$



We have no idea of the existence of the third mode in closed-form! MCMC sampling is one of the ways to find it.

Image source: Surajit Ray and Bruce G. Lindsay, “The topology of multivariate normal mixtures”, in the Annals of Statistics, 2005

Mixture Density Networks: Inference

- A simple alternative is to take the mean of the most probable component.
- That is, the one with the largest mixing coefficient at each value of x.

Mixture Density Networks: Inference

Example: Simple toy problem to visualize multimodality

- Plot of the mixing coefficients $\pi_k(\mathrm{x})$ as a function of x for 3 kernel functions in a mixture density network.
- At both small and large values of x, where the conditional probability density of the target data is unimodal, only one of the kernels has a high value for its prior probability.
- While at intermediate values of x, where the conditional density is trimodal, the three mixing coefficients have comparable values

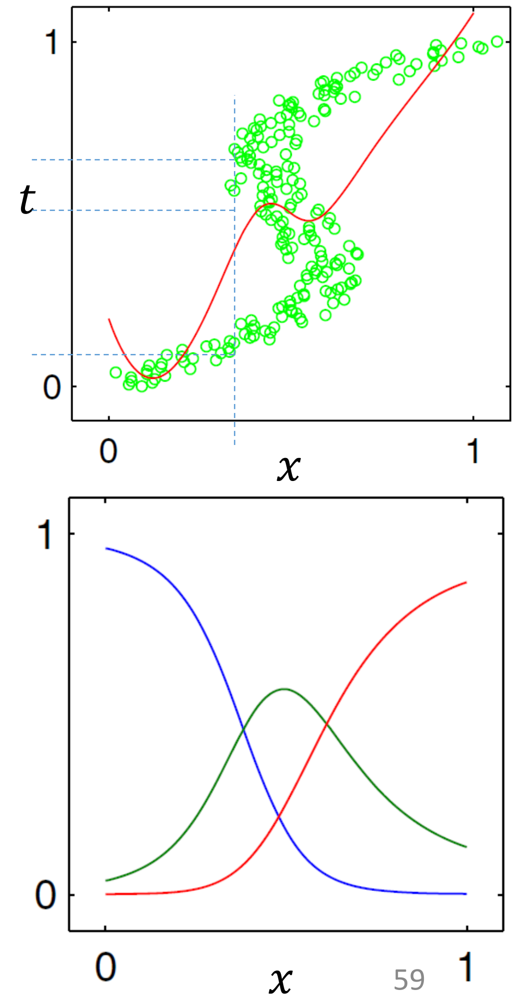
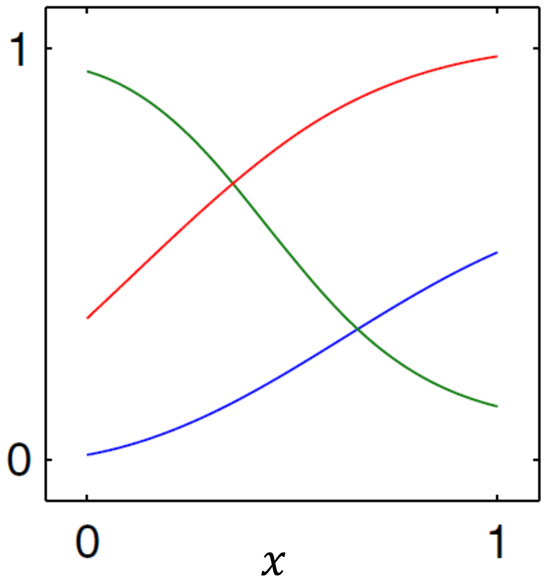


Image Source: "Pattern Recognition and Machine Learning", Christopher Bishop

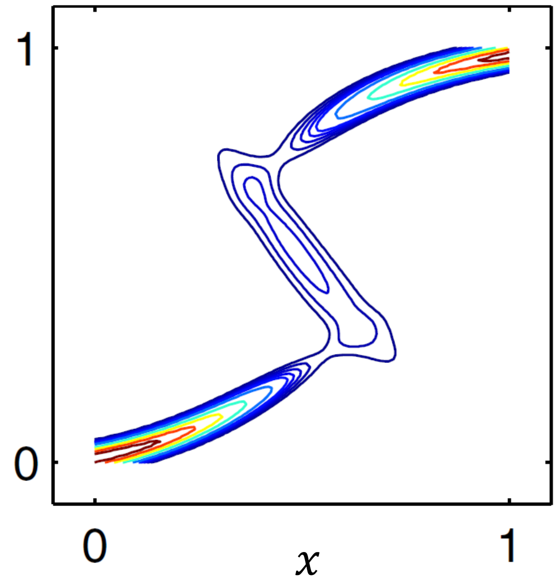
Mixture Density Networks: Inference

Example: Simple toy problem to visualize multimodality

Plots of the means $\mu_k(\mathrm{x})$ using the same colour coding as $\pi_k(\mathrm{x})$.



Plots of the mixture model from $p(\mathrm{t} \mid \mathrm{x})$



Plot of the approximate conditional mode (largest mixing coefficient).

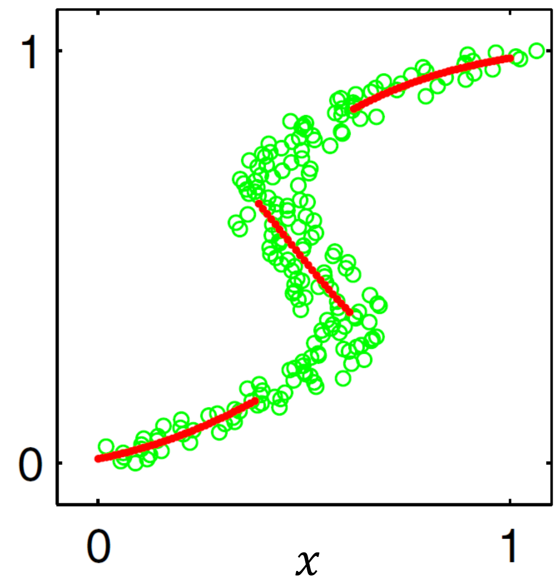


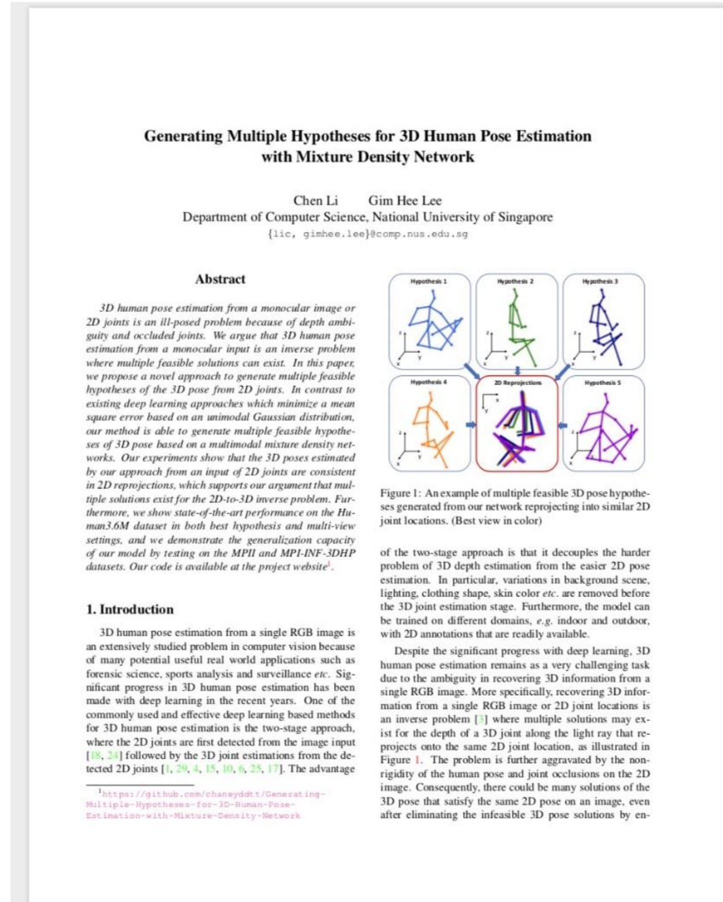
Image Source: "Pattern Recognition and Machine Learning", Christopher Bishop

Mixture Density Network: Example

Chen Li and Gim Hee Lee,

“Generating Multiple Hypotheses for 3D Human Pose Estimation with Mixture Density Network”,

CVPR 2019



Generating Multiple Hypotheses for 3D Human Pose Estimation with Mixture Density Network

Chen Li Gim Hee Lee
Department of Computer Science, National University of Singapore
{lic, gimhee.lee}@comp.nus.edu.sg

Abstract

3D human pose estimation from a monocular image or 2D joints is an ill-posed problem because of depth ambiguity and occluded joints. We argue that 3D human pose estimation from a monocular input is an inverse problem where multiple feasible solutions can exist. In this paper, we propose a novel approach to generate multiple feasible hypotheses of the 3D pose from 2D joints. In contrast to existing deep learning approaches which minimize a mean square error based on an unimodal Gaussian distribution, our method is able to generate multiple feasible hypotheses of 3D pose based on a multimodal mixture density networks. Our experiments show that the 3D poses estimated by our approach from an input of 2D joints are consistent in 2D reprojections, which supports our argument that multiple solutions exist for the 2D-to-3D inverse problem. Furthermore, we show state-of-the-art performance on the Human3.6M dataset in both best hypothesis and multi-view settings, and we demonstrate the generalization capacity of our model by testing on the MPII and MPI-INF-3DHP datasets. Our code is available at the project website[1].

1. Introduction

3D human pose estimation from a single RGB image is an extensively studied problem in computer vision because of many potential useful real world applications such as forensic science, sports analysis and surveillance *etc*. Significant progress in 3D human pose estimation has been made with deep learning in the recent years. One of the commonly used and effective deep learning based methods for 3D human pose estimation is the two-stage approach, where the 2D joints are first detected from the image input [18, 24] followed by the 3D joint estimations from the detected 2D joints [1, 29, 4, 15, 10, 6, 25, 17]. The advantage

[1] https://github.com/chaneyddtt/Generating-Multiple-Hypotheses-for-3D-Human-Pose-Estimation-with-Mixture-Density-Network

Figure 1: An example of multiple feasible 3D pose hypotheses generated from our network reprojecting into similar 2D joint locations. (Best view in color)

of the two-stage approach is that it decouples the harder problem of 3D depth estimation from the easier 2D pose estimation. In particular, variations in background scene, lighting, clothing shape, skin color *etc*. are removed before the 3D joint estimation stage. Furthermore, the model can be trained on different domains, *e.g.* indoor and outdoor, with 2D annotations that are readily available.

Despite the significant progress with deep learning, 3D human pose estimation remains as a very challenging task due to the ambiguity in recovering 3D information from a single RGB image. More specifically, recovering 3D information from a single RGB image or 2D joint locations is an inverse problem [3] where multiple solutions may exist for the depth of a 3D joint along the light ray that reprojects onto the same 2D joint location, as illustrated in Figure 1. The problem is further aggravated by the non-rigidity of the human pose and joint occlusions on the 2D image. Consequently, there could be many solutions of the 3D pose that satisfy the same 2D pose on an image, even after eliminating the infeasible 3D pose solutions by en-

Mixture Density Network: 3D Human Pose Estimation

- **Problem:** Given a 2D image of a person, find the 3D skeleton of this person.

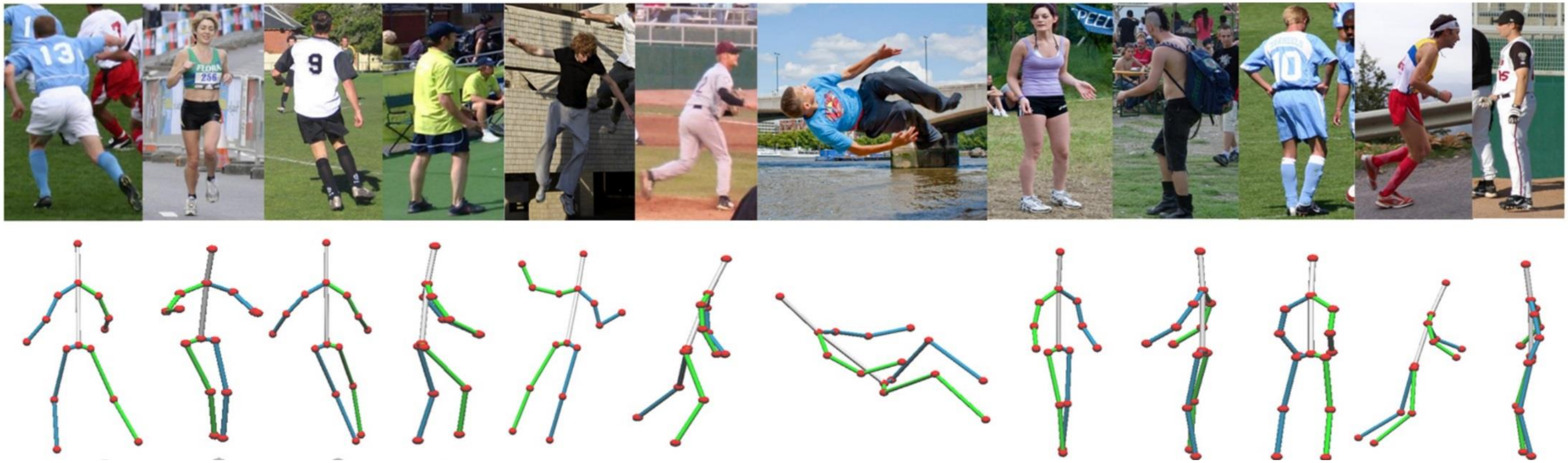
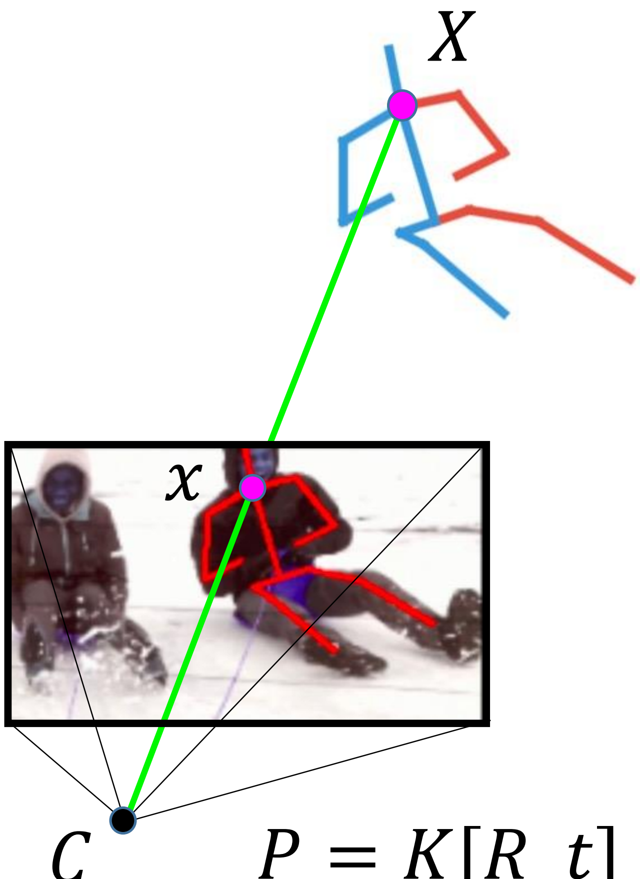


Image Source: W. Chen et. al, Synthesizing training images for boosting human 3d pose estimation, 3DV 2016

3D Human Pose Estimation: Challenges

- This is an ill-posed problem, the depth is missing!



$$P = K[R \;\; t]$$

3D to 2D projection: $x \sim PX$

3x1 homogeneous coordinates

4x1 homogeneous coordinates

One-parameter family of back-projected ray from x by P:

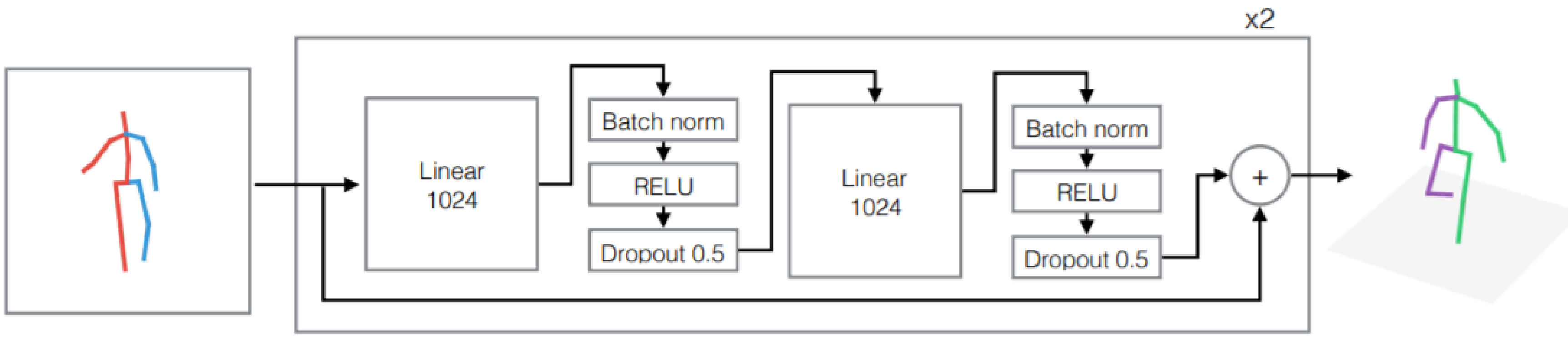
$$X(\lambda) = \lambda P^{+}x + (1 - \lambda)C$$

Ray is parametrized by the scalar λ

pseudo-inverse of P, i.e. $P^{+} = P^{T}(PP^{T})^{-1}$

Existing Works: Two-Stage Approach

- Detect 2D pose, then regress 3D pose.

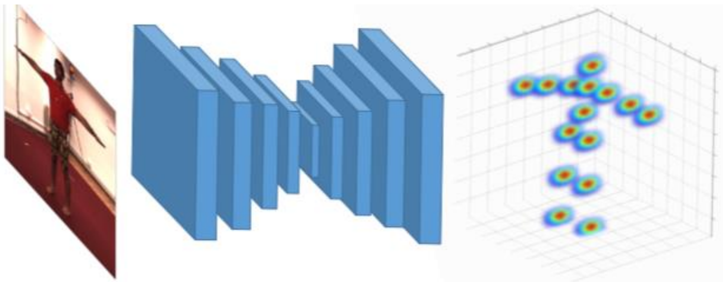


Advantage: variations in background, lighting, clothing etc. are removed before the second stage.

J. Martinez et al, A simple yet effective baseline for 3d human pose estimation, ICCV 2017

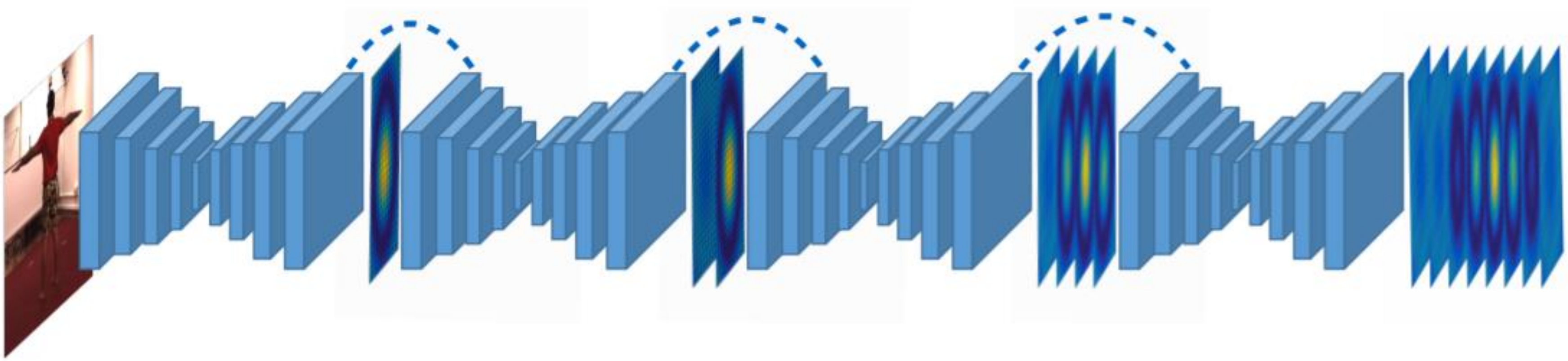
Existing Works: One-Stage Approach

- Directly detect 3d pose from monocular images.



Coarse-to-fine scheme :

- Large dimensional increase
- Iterative refinement



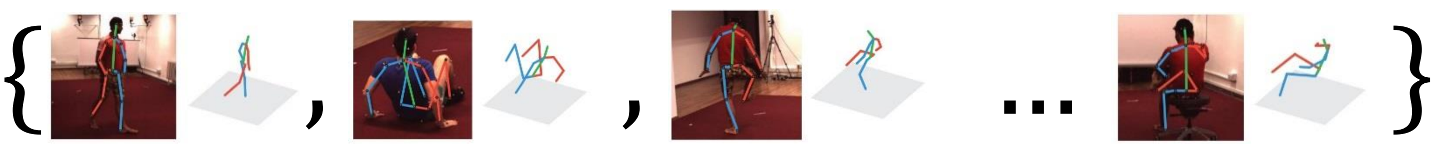
Advantage: make use of 2d pose dataset

G. Pavlakos et al. Coarse-to-Fine Volumetric Prediction for Single-Image 3D Human Pose. CVPR2017

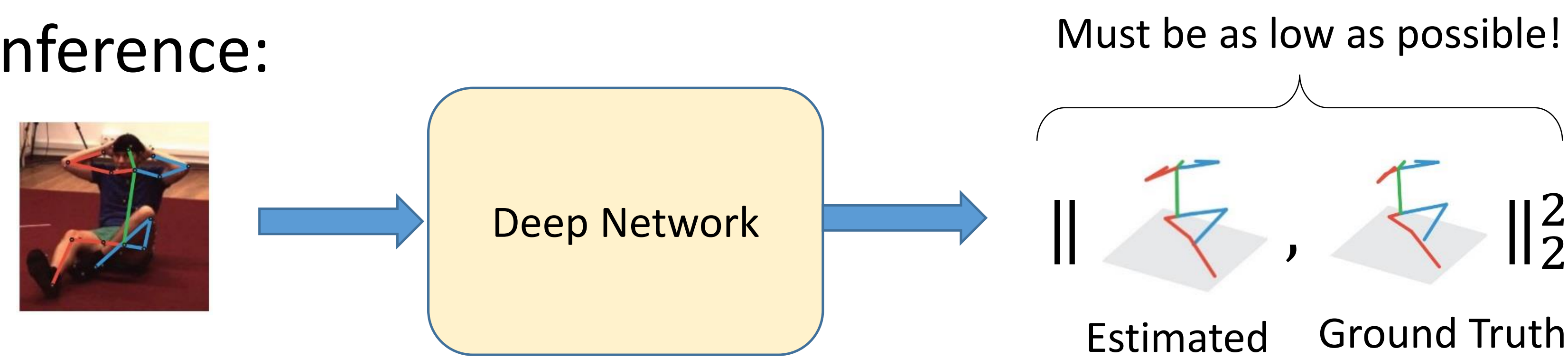
What's Wrong with Existing Works?

- Learn a network from benchmark datasets with one 2D image to one ground truth 3D pose.

Training data:

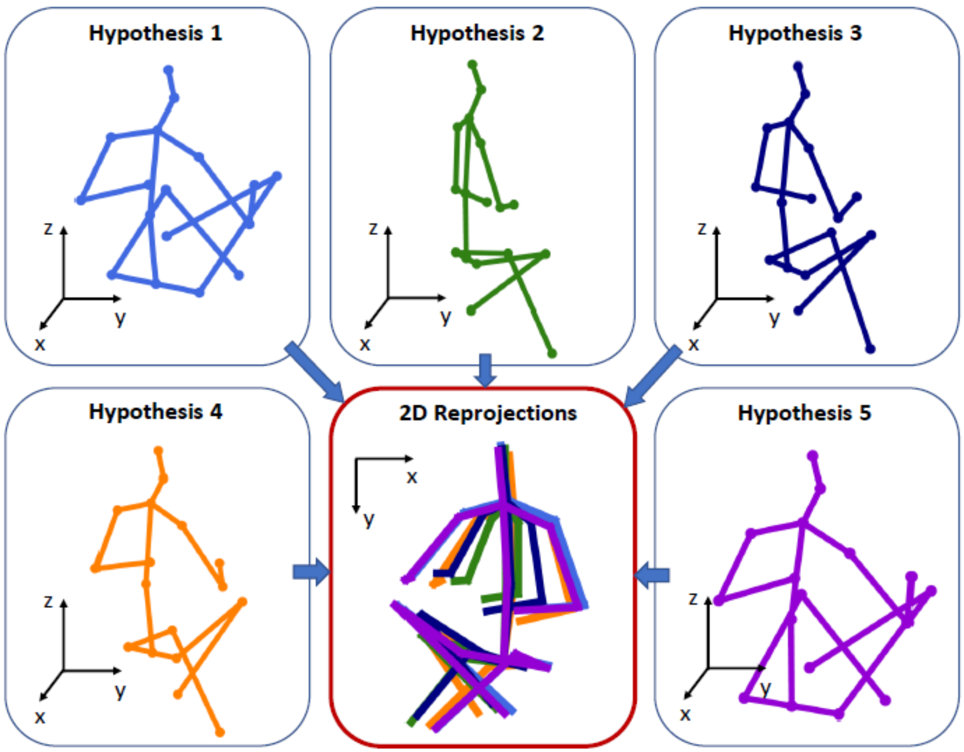


Inference:



What's Wrong with Existing Works?

- One 2D image to one 3D pose: Is this always true? **NO!!!**
- Multiple feasible solutions can exist!



Two Reasons:

1. Depth ambiguity
2. Joint occlusion

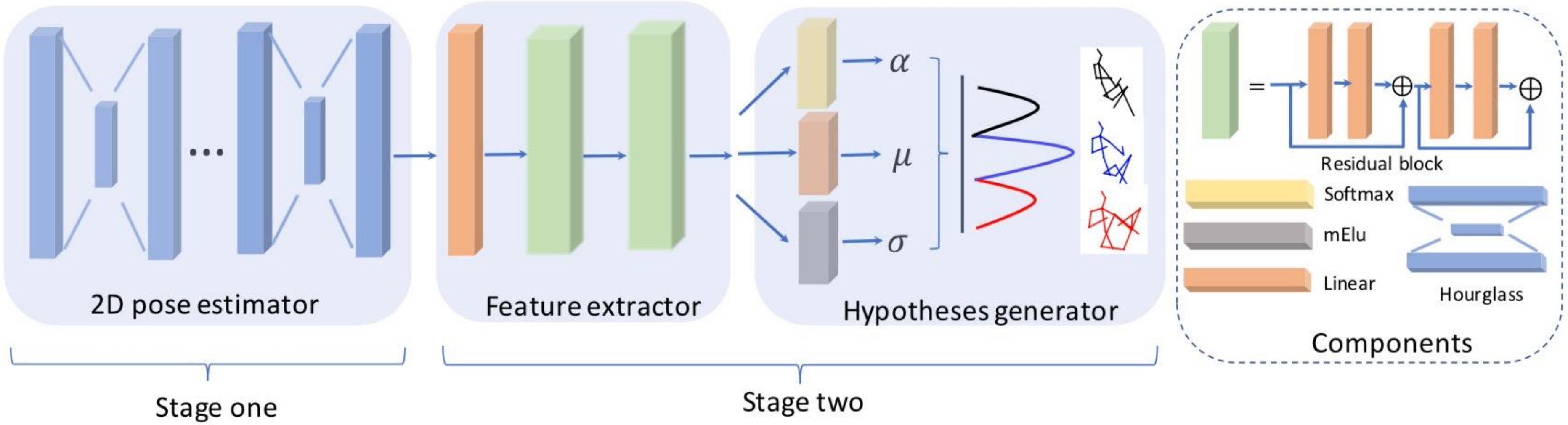
What’s Wrong with Existing Works?

The paradigm of “one 2D image to one 3D pose” means that:

The whole community is overfitting to the benchmark datasets with deep learning!

The Solution: Mixture Density Network

- A deep network that estimates multiple feasible solutions.
- Multiple solutions represented by a Gaussian Mixture model – each Gaussian kernel ≡ a feasible solution.



Model Representation

- Let $\mathbf{w}$ be the learnable weights of the deep network f, i.e. $\Theta = f(\mathbf{x};\mathbf{w}) \Longrightarrow \Theta(\mathbf{x},\mathbf{w}) = \{\boldsymbol{\mu}(\mathbf{x},\mathbf{w}), \boldsymbol{\sigma}(\mathbf{x},\mathbf{w}), \boldsymbol{\alpha}(\mathbf{x},\mathbf{w})\}$.

- GMM represents the probability density of the 3D pose $\mathbf{y} \in \mathbb{R}^{3N}$ given the 2D joints $\mathbf{x} \in \mathbb{R}^{2N}$.

$$p(\mathbf{y} \mid \mathbf{x},\mathbf{w}) = \sum_{i=1}^{M} \alpha_i(\mathbf{x},\mathbf{w})\phi_i(\mathbf{y} \mid \mathbf{x},\mathbf{w}),$$

where

$$\phi_i(\mathbf{y} \mid \mathbf{x},\mathbf{w}) = \frac{1}{(2\pi)^{d/2}\sigma_i(\mathbf{x},\mathbf{w})^d} \exp -\frac{\|\mathbf{y} - \mu_i(\mathbf{x},\mathbf{w})\|^2}{2\sigma_i(\mathbf{x},\mathbf{w})^2}.$$

$$0 \leq \alpha_i(\mathbf{x},\mathbf{w}) \leq 1, \qquad \sum_{i=1}^{M} \alpha_i(\mathbf{x},\mathbf{w}) = 1.$$

Model Representation

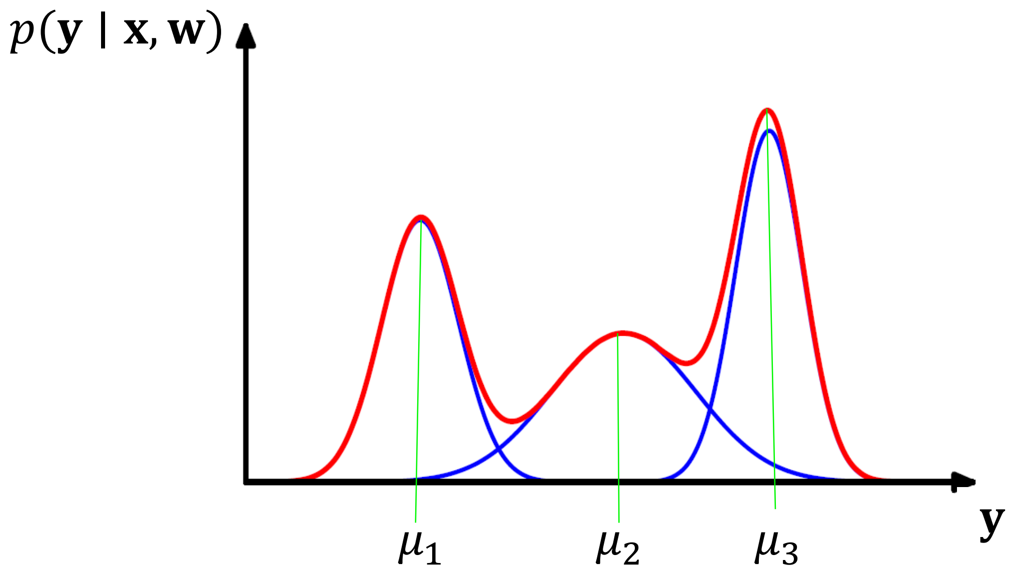
GMM:

$$p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}) = \sum_{i=1}^{M} \alpha_i(\mathbf{x}, \mathbf{w}) \phi_i(\mathbf{y} \mid \mathbf{x}, \mathbf{w}),$$

where

$$\phi_i(\mathbf{y} \mid \mathbf{x}, \mathbf{w}) = \frac{1}{(2\pi)^{d/2} \sigma_i(\mathbf{x}, \mathbf{w})^d} \exp - \frac{\|\mathbf{y} - \mu_i(\mathbf{x}, \mathbf{w})\|^2}{2\sigma_i(\mathbf{x}, \mathbf{w})^2}.$$

1D Example:



3 feasible solutions, μ_1, μ_2, μ_3.

Loss Functions

- **Given**: a training dataset,

$$\{\mathbf{X}, \mathbf{Y}\} = \{\{\mathbf{x}_j, \mathbf{y}_j\} \mid j = 1, ..., K\}$$

- **Find**: the maximum a posterior (MAP) of the set of learnable weights w, i.e.

$$\underset{\mathbf{w}}{\operatorname{argmax}}\, p(\,\mathbf{w} \mid \mathbf{X}, \mathbf{Y}, \Psi\,),$$

where Ψ is the hyperparameter of the prior over $\mathbf{w}$.

Loss Functions

- Assuming each training data is i.i.d., we have

$$p(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}, \Psi) \propto p(\mathbf{Y} \mid \mathbf{X}, \mathbf{w}) p(\mathbf{w} \mid \mathbf{X}, \Psi)$$ (Bayes rule)

$$= p(\mathbf{w} \mid \mathbf{X}, \Psi) \prod_{j=1}^{K} p(\mathbf{y}_j \mid \mathbf{x}_j, \mathbf{w})$$ (i.i.d)

$$= p(\mathbf{w} \mid \mathbf{X}, \Psi) \prod_{j=1}^{K} \sum_{i=1}^{M} \alpha_i(\mathbf{x}_j, \mathbf{w}) \phi_i(\mathbf{y}_j \mid \mathbf{x}_j, \mathbf{w})$$

Loss Functions

- Turn MAP into minimum negative log-posterior:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}}\, p(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}, \Psi),$$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} - \ln p(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}, \Psi)$$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} \underbrace{- \sum_{j=1}^{K} \ln p(\mathbf{y}_j \mid \mathbf{x}_j, \mathbf{w}) - \ln p(\mathbf{w} \mid \mathbf{X}, \Psi)}_{\mathcal{L}},$$

Loss function of deep network!

Loss Functions

$$\begin{aligned}\mathcal{L} &= -\sum_{j=1}^{K} \ln p(\mathbf{y}_j \mid \mathbf{x}_j, \mathbf{w}) - \ln p(\mathbf{w} \mid \mathbf{X}, \Psi) \\ &= -\sum_{j=1}^{K} \ln \sum_{i=1}^{M} \alpha_i(\mathbf{x}_j, \mathbf{w}) \phi_i(\mathbf{y}_j \mid \mathbf{x}_j, \mathbf{w}) - \ln p(\mathbf{w} \mid \mathbf{X}, \Psi) \\ &= \mathcal{L}_{\text{3D}} + \mathcal{L}_{\text{prior}}.\end{aligned}$$

Loss Functions

- The prior loss $\mathcal{L}_{prior}$ can be further evaluated into:

Independent of $\mathbf{w}$

Assume uniform prior on μ and σ

$$
\begin{aligned}
\mathcal{L}_{\text{prior}} &= -\ln p(\mathbf{w} \mid \mathbf{X}, \Psi) \\
&= -\ln p(\mathbf{w}, \mathbf{X} \mid \Psi) + \cancel{\ln p(\mathbf{X} \mid \Psi)} \\
&\propto -\ln p(\Theta(\mathbf{w}, \mathbf{X}) \mid \Psi) \\
&= -\ln p(\boldsymbol{\alpha}(\mathbf{w}, \mathbf{X}) \mid \Psi) - \boxed{\ln p(\boldsymbol{\mu}(\mathbf{w}, \mathbf{X}), \boldsymbol{\sigma}(\mathbf{w}, \mathbf{X}) \mid \Psi)} \\
&= -\sum_{j=1}^{K} \ln p(\alpha_1(\mathbf{w}, \mathbf{x}_j), \ldots, \alpha_M(\mathbf{w}, \mathbf{x}_j) \mid \Lambda)
\end{aligned}
$$

where

$\Lambda = \{\lambda_1, \ldots, \lambda_M\}$: hyperparameter of conjugate prior on α.

Loss Functions

- The prior loss $\mathcal{L}_{prior}$ can be further evaluated into:

$$\mathcal{L}_{\text{prior}} = -\sum_{j=1}^{K} \ln p(\alpha_1(\mathbf{w}, \mathbf{x}_j), \dots, \alpha_M(\mathbf{w}, \mathbf{x}_j) \mid \Lambda)$$

where

Dirichlet distribution

- $$p(\alpha_1, \dots, \alpha_M \mid \Lambda) = \text{Dir}_{[\alpha_1, \dots, \alpha_M]}[\lambda_1, \dots, \lambda_M]$$
$$= \frac{\Gamma[\sum_{i=1}^{M} \lambda_i]}{\prod_{i=1}^{M} \Gamma[\lambda_i]} \prod_{i=1}^{M} \alpha_i(\mathbf{w}, \mathbf{x}_j)^{\lambda_i - 1}$$

Independent of α

- $\Gamma[.]$ is the Gamma function,
- $\lambda_i > 0$.

Loss Functions

- Final loss function: $\mathcal{L} = \mathcal{L}_{3D} + \mathcal{L}_{\text{prior}}$,

  where

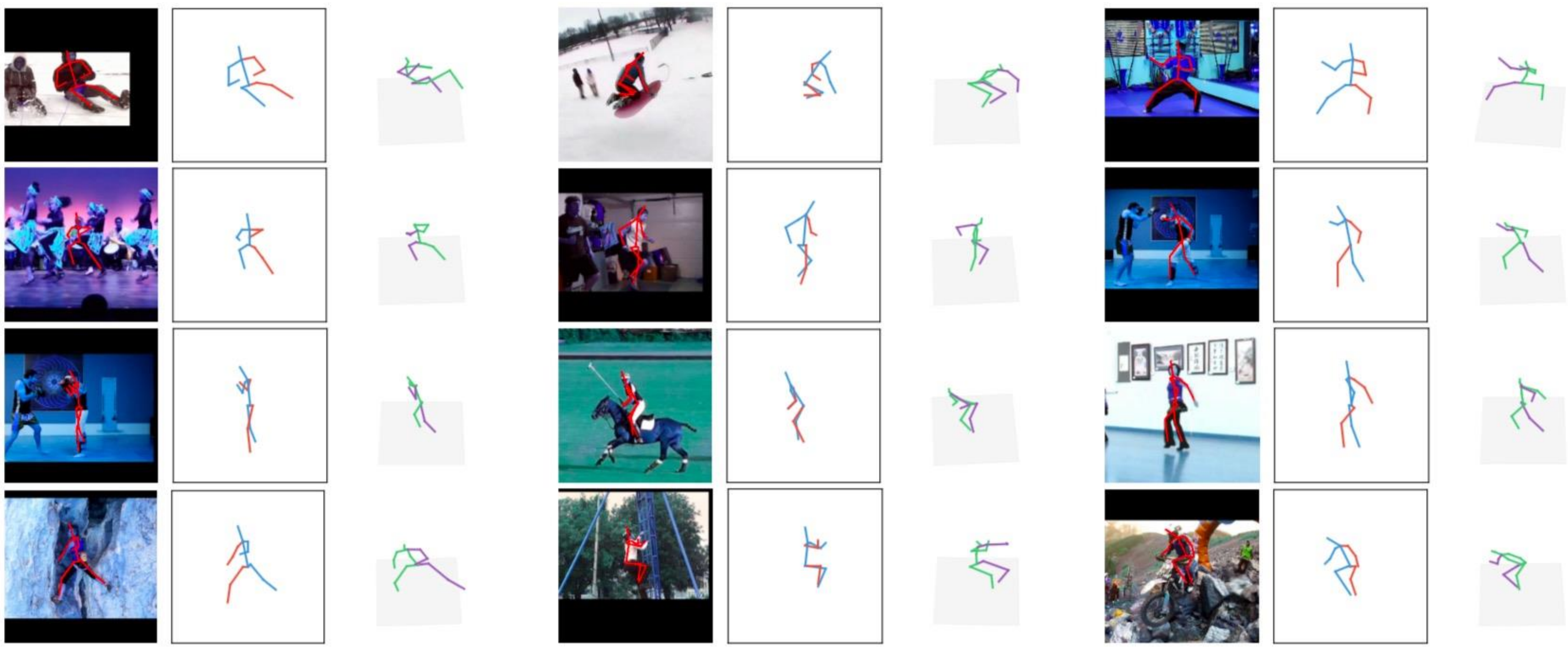
$$\mathcal{L}_{3D} = -\sum_{j=1}^{K} \ln \sum_{i=1}^{M} \alpha_i(\mathbf{x}_j, \mathbf{w}) \phi_i(\mathbf{y}_j \mid \mathbf{x}_j, \mathbf{w})$$

$$\mathcal{L}_{\text{prior}} = -\sum_{j=1}^{K} \sum_{i=1}^{M} (\lambda_i - 1) \ln \alpha_i(\mathbf{w}, \mathbf{x}_j).$$

Remarks: we set $\lambda_1 = \ldots = \lambda_M = C > 1$ to prevent overfitting of a single Gaussian kernel in the MDN, *i.e.*, $\alpha_i \approx 1$ and $\alpha_{j \neq i} \approx 0$.

Experiments

- Qualitative results on the MPII test set.



Summary

- We have looked at how to:

1. Explain the difference between the discriminative and generative models.

2. Describe the concept behind Variational AutoEncoder, and how it can be used to generate new images.

3. Use the Mixture Density Network to solve the inverse problem where multiple feasible solutions can exist.