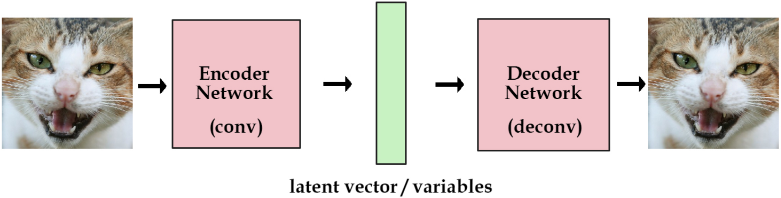
Variational Autoencoder

Introduction - Autoencoders

- 

- Attempt to learn identity function
- Constrained in some way (e.g., small latent vector representation)
- Can generate new images by giving different latent vectors to trained network
- Variational: use probabilistic latent encoding

Deep Learning Perspective

Deep Learning Perspective

- Goal: Build a neural network that generates MNIST digits from random (Gaussian) noise

Deep Learning Perspective

- Goal: Build a neural network that generates MNIST digits from random (Gaussian) noise
- Define two sub-networks: Encoder and Decoder

Deep Learning Perspective

- Goal: Build a neural network that generates MNIST digits from random (Gaussian) noise
- Define two sub-networks: Encoder and Decoder
- Define a Loss Function

Encoder

- A neural network $q_\theta(z|x)$

Encoder

- A neural network $q_\theta(z|x)$
- Input: datapoint x (e.g. 28×28-pixel MNIST digit)

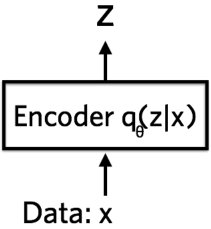
Encoder

- A neural network $q_\theta(z|x)$
- Input: datapoint x (e.g. 28×28-pixel MNIST digit)
- Output: encoding z, drawn from Gaussian density with parameters θ

Encoder

- A neural network $q_\theta(z|x)$
- Input: datapoint x (e.g. 28×28-pixel MNIST digit)
- Output: encoding z, drawn from Gaussian density with parameters θ
- $|z| \ll |x|$

Encoder

- A neural network $q_\theta(z|x)$
- Input: datapoint x (e.g. 28×28-pixel MNIST digit)
- Output: encoding z, drawn from Gaussian density with parameters θ
- $|z| \ll |x|$
- 

Decoder

- A neural network $p_\phi(x|z)$, parameterized by ϕ

Decoder

- A neural network $p_\phi(x|z)$, parameterized by ϕ
- Input: encoding z, output from encoder

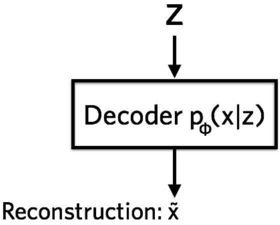
Decoder

- A neural network $p_\phi(x|z)$, parameterized by ϕ
- Input: encoding z, output from encoder
- Output: reconstruction $\tilde{x}$, drawn from distribution of the data

Decoder

- A neural network $p_\phi(x|z)$, parameterized by ϕ
- Input: encoding z, output from encoder
- Output: reconstruction $\tilde{x}$, drawn from distribution of the data
- E.g., output parameters for 28×28 Bernoulli variables

Decoder

- A neural network $p_\phi(x|z)$, parameterized by ϕ
- Input: encoding z, output from encoder
- Output: reconstruction $\tilde{x}$, drawn from distribution of the data
- E.g., output parameters for 28×28 Bernoulli variables
- 

Loss Function

- $\tilde{x}$ is reconstructed from z where $|z| \ll |\tilde{x}|$

Loss Function

- $\tilde{x}$ is reconstructed from z where $|z| \ll |\tilde{x}|$
- How much information is lost when we go from x to z to $\tilde{x}$?

Loss Function

- $\tilde{x}$ is reconstructed from z where $|z| \ll |\tilde{x}|$
- How much information is lost when we go from x to z to $\tilde{x}$?
- Measure this with reconstruction log-likelihood: $\log p_\phi(x|z)$

Loss Function

- $\tilde{x}$ is reconstructed from z where $|z| \ll |\tilde{x}|$
- How much information is lost when we go from x to z to $\tilde{x}$?
- Measure this with reconstruction log-likelihood: $\log p_\phi(x|z)$
- Measures how effectively the decoder has learned to reconstruct x given the latent representation z

Loss Function

- Loss function is negative reconstruction log-likelihood + regularizer

Loss Function

- Loss function is negative reconstruction log-likelihood + regularizer
- Loss decomposes into term for each datapoint:

$$L(\theta, \phi) = \sum_{i=1}^{N} l_i(\theta, \phi)$$

Loss Function

- Loss function is negative reconstruction log-likelihood + regularizer
- Loss decomposes into term for each datapoint:

$$L(\theta, \phi) = \sum_{i=1}^{N} l_i(\theta, \phi)$$

- Loss for datapoint x_i:

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)}\big[\log p_\phi(x_i|z)\big] + KL\big(q_\theta(z|x_i)||p(z)\big)$$

Loss Function

- Negative reconstruction log-likelihood:

$$-\mathbb{E}_{z \sim q_\theta(z|x_i)} \left[\log p_\phi(x_i|z) \right]$$

Loss Function

- Negative reconstruction log-likelihood:

$$-\mathbb{E}_{z\sim q_\theta(z|x_i)}\left[\log p_\phi(x_i|z)\right]$$

- Encourages decoder to learn to reconstruct the data

Loss Function

- Negative reconstruction log-likelihood:

$$-\mathbb{E}_{z \sim q_\theta(z|x_i)} \left[\log p_\phi(x_i|z) \right]$$

- Encourages decoder to learn to reconstruct the data
- Expectation taken over distribution of latent representations

Loss Function

- KL Divergence as regularizer:

$$KL\big(q_\theta(z|x_i)||p(z)\big) = \mathbb{E}_{z\sim q_\theta(z|x_i)}\big[\log q_\theta(z|x_i) - \log p(z)\big]$$

Loss Function

- KL Divergence as regularizer:

$$KL\big(q_\theta(z|x_i)||p(z)\big) = \mathbb{E}_{z\sim q_\theta(z|x_i)}\big[\log q_\theta(z|x_i) - \log p(z)\big]$$

- Measures information lost when using q_θ to represent p

Loss Function

- KL Divergence as regularizer:

$$KL\big(q_\theta(z|x_i)||p(z)\big) = \mathbb{E}_{z\sim q_\theta(z|x_i)}\big[\log q_\theta(z|x_i) - \log p(z)\big]$$

- Measures information lost when using q_θ to represent p
- We will use $p(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$

Loss Function

- KL Divergence as regularizer:

$$KL\big(q_\theta(z|x_i)||p(z)\big) = \mathbb{E}_{z\sim q_\theta(z|x_i)}\big[\log q_\theta(z|x_i) - \log p(z)\big]$$

- Measures information lost when using q_θ to represent p
- We will use $p(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$
- Encourages encoder to produce z's that are close to standard normal distribution

Loss Function

- KL Divergence as regularizer:

$$KL\big(q_\theta(z|x_i)||p(z)\big) = \mathbb{E}_{z\sim q_\theta(z|x_i)}\big[\log q_\theta(z|x_i) - \log p(z)\big]$$

- Measures information lost when using q_θ to represent p
- We will use $p(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$
- Encourages encoder to produce z's that are close to standard normal distribution
- Encoder learns a meaningful representation of MNIST digits

Loss Function

- KL Divergence as regularizer:

$$KL\big(q_\theta(z|x_i)||p(z)\big) = \mathbb{E}_{z \sim q_\theta(z|x_i)}\big[\log q_\theta(z|x_i) - \log p(z)\big]$$

- Measures information lost when using q_θ to represent p
- We will use $p(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$
- Encourages encoder to produce z's that are close to standard normal distribution
- Encoder learns a meaningful representation of MNIST digits
- Representation for images of the same digit are close together in latent space

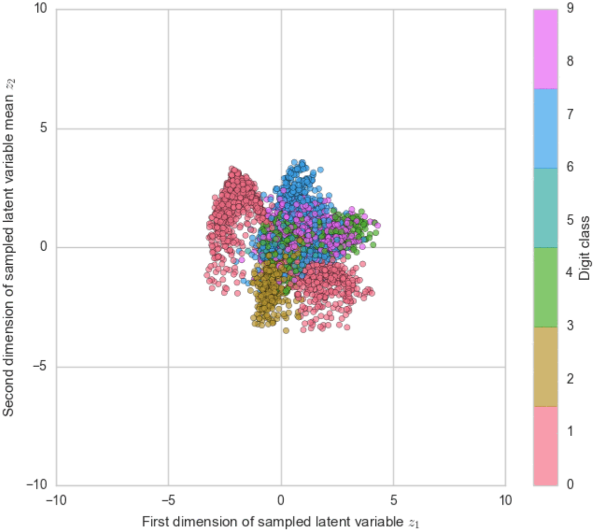
Loss Function

- KL Divergence as regularizer:

$$KL\big(q_\theta(z|x_i)||p(z)\big) = \mathbb{E}_{z\sim q_\theta(z|x_i)}\big[\log q_\theta(z|x_i) - \log p(z)\big]$$

- Measures information lost when using q_θ to represent p
- We will use $p(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$
- Encourages encoder to produce z's that are close to standard normal distribution
- Encoder learns a meaningful representation of MNIST digits
- Representation for images of the same digit are close together in latent space
- Otherwise could "memorize" the data and map each observed datapoint to a distinct region of space

MNIST latent variable space



Reparameterization trick

- We want to use gradient descent to learn the model's parameters

Reparameterization trick

- We want to use gradient descent to learn the model's parameters
- Given z drawn from $q_\theta(z|x)$, how do we take derivatives of (a function of) z w.r.t. θ?

Reparameterization trick

- We want to use gradient descent to learn the model's parameters
- Given z drawn from $q_\theta(z|x)$, how do we take derivatives of (a function of) z w.r.t. θ?
- We can reparameterize: $z = \mu + \sigma \odot \epsilon$

Reparameterization trick

- We want to use gradient descent to learn the model's parameters
- Given z drawn from $q_\theta(z|x)$, how do we take derivatives of (a function of) z w.r.t. θ?
- We can reparameterize: $z = \mu + \sigma \odot \epsilon$
- $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\odot$ is element-wise product

Reparameterization trick

- We want to use gradient descent to learn the model's parameters
- Given z drawn from $q_\theta(z|x)$, how do we take derivatives of (a function of) z w.r.t. θ?
- We can reparameterize: $z = \mu + \sigma \odot \epsilon$
- $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\odot$ is element-wise product
- Can take derivatives of (functions of) z w.r.t. μ and σ

Reparameterization trick

- We want to use gradient descent to learn the model's parameters
- Given z drawn from $q_\theta(z|x)$, how do we take derivatives of (a function of) z w.r.t. θ?
- We can reparameterize: $z = \mu + \sigma \odot \epsilon$
- $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\odot$ is element-wise product
- Can take derivatives of (functions of) z w.r.t. μ and σ
- Output of $q_\theta(z|x)$ is vector of μ's and vector of σ's

Summary

- Deep Learning objective is to minimize the loss function:

$$L(\theta, \phi) = \sum_{i=1}^{N} \Big(-\mathbb{E}_{z \sim q_\theta(z|x_i)} \big[\log p_\phi(x_i|z) \big] + KL\big(q_\theta(z|x_i) || p(z)\big) \Big)$$