Generative latent variable models, un- and self-supervised learning

Ole Winther

Bioinformatics Centre, Department of Biology
University of Copenhagen (UCph)

Dept for Applied Mathematics and Computer Science
Technical University of Denmark (DTU)

December 10, 2025

Unsupervised learning motivation 2016

Deep learning today:

- Mostly about pure supervised learning
- Requires a lot of labeled data: expensive to collect

Deep learning in the future:

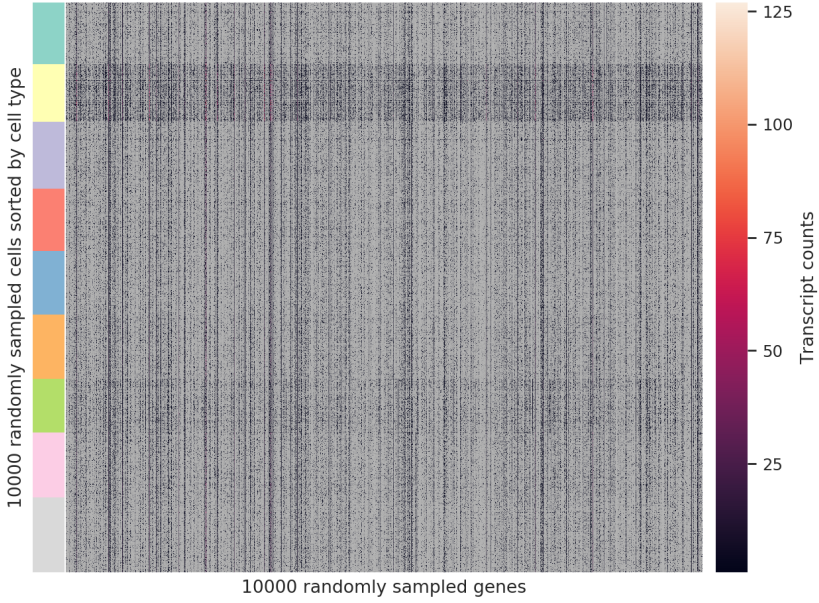
- Unsupervised, more human-like

"We expect unsupervised learning to become far more important in the longer term. Human and animal learning is largely unsupervised: we discover the structure of the world by observing it, not by being told the name of every object."
–LeCun, Bengio, Hinton, Nature 2015

Many thanks to Tapani Raiko for sharing slides!

Generative models for complex data 2016

Motivation - Single cell gene expression analysis - scRNA



Single cell data typically 98+% sparse (grey color)

Objectives of this week's lectures

- P1: Unsupervised learning
- P2: Self-supervised learning
- P3: Generative modeling with latent variables
  - Variational auto-encoders (VAE)
  - Generative adversarial networks (GAN)

Part 1:
Unsupervised learning

Unsupervised learning

- Let us forget about the supervised learning setup.
- No target value $\mathbf{t}$ and no $p(\mathbf{t}|\mathbf{x})$

Unsupervised learning

- Let us forget about the supervised learning setup.
- No target value $\mathbf{t}$ and no $p(\mathbf{t}|\mathbf{x})$
- Instead focus on $\mathbf{x}$ and $p(\mathbf{x})$.
- Training data: $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.

Unsupervised learning

- Let us forget about the supervised learning setup.
- No target value $\mathbf{t}$ and no $p(\mathbf{t}|\mathbf{x})$
- Instead focus on $\mathbf{x}$ and $p(\mathbf{x})$.
- Training data: $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.
- Unsupervised learning - the old term when the world was supervised vs unsupervised
- Insight about the data
  - Low dimensional representations nice to look at it (PCA, UMap, t-SNE, etc.)
  - Clustering - Prototypical examples/unsupervised classification

Unsupervised learning

- Let us forget about the supervised learning setup.
- No target value $\mathbf{t}$ and no $p(\mathbf{t}|\mathbf{x})$
- Instead focus on $\mathbf{x}$ and $p(\mathbf{x})$.
- Training data: $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.
- Unsupervised learning - the old term when the world was supervised vs unsupervised
- Insight about the data
  - Low dimensional representations nice to look at it (PCA, UMap, t-SNE, etc.)
  - Clustering - Prototypical examples/unsupervised classification
- Will talk a bit about principal component analysis (PCA), but unsupervised learning is really a topic of another ML course

Unsupervised learning

- Let us forget about the supervised learning setup.
- No target value $\mathbf{t}$ and no $p(\mathbf{t}|\mathbf{x})$
- Instead focus on $\mathbf{x}$ and $p(\mathbf{x})$.
- Training data: $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.
- Unsupervised learning - the old term when the world was supervised vs unsupervised
- Insight about the data
  - Low dimensional representations nice to look at it (PCA, UMap, t-SNE, etc.)
  - Clustering - Prototypical examples/unsupervised classification
- Will talk a bit about principal component analysis (PCA), but unsupervised learning is really a topic of another ML course
- Self-supervised - invent auxiliary tasks to support learning other tasks

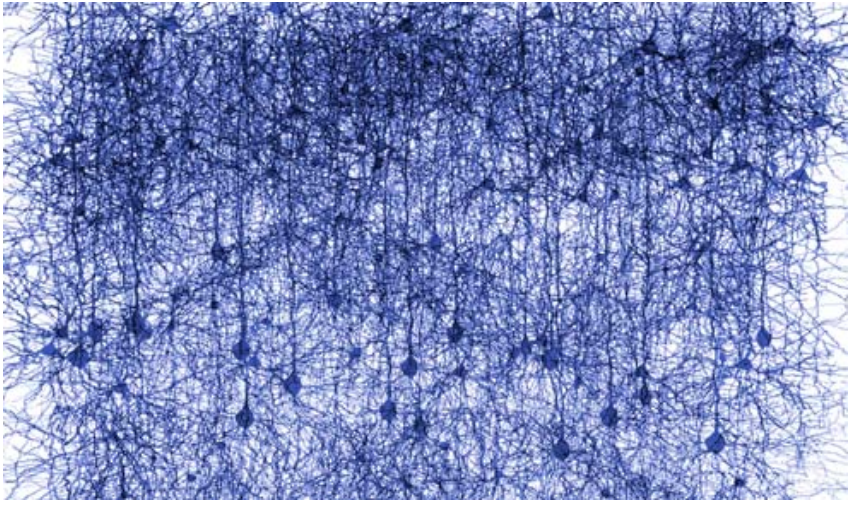
Unsupervised learning

- Let us forget about the supervised learning setup.
- No target value $\mathbf{t}$ and no $p(\mathbf{t}|\mathbf{x})$
- Instead focus on $\mathbf{x}$ and $p(\mathbf{x})$.
- Training data: $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.
- Unsupervised learning - the old term when the world was supervised vs unsupervised
- Insight about the data
  - Low dimensional representations nice to look at it (PCA, UMap, t-SNE, etc.)
  - Clustering - Prototypical examples/unsupervised classification
- Will talk a bit about principal component analysis (PCA), but unsupervised learning is really a topic of another ML course
- Self-supervised - invent auxiliary tasks to support learning other tasks
- Generative modeling - learn $p(\mathbf{x})$ so we can synthesise new data: $\mathbf{x} \sim p(\mathbf{x})$.

Part 2:
Self-supervised learning

Neural networks for self-supervised learning

- Autoencoders (AE)
- Masked language modeling
- Contrastive learning



Representation learning, encoders and decoders

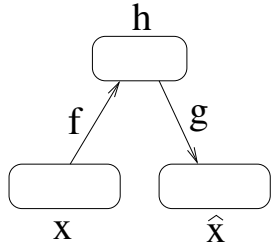
- Representation learning
- Learn $\mathbf{h} = \mathbf{f}(\mathbf{x})$
- $\mathbf{h}$ informative about data $\mathbf{x}$!

Representation learning, encoders and decoders

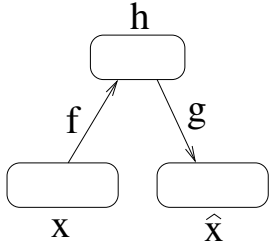
- Representation learning
- Learn $\mathbf{h} = \mathbf{f}(\mathbf{x})$
- $\mathbf{h}$ informative about data $\mathbf{x}$!
- First example: autoencoder!

Vocabulary:

- Encoder function $\mathbf{f} : \mathbf{x} \rightarrow \mathbf{h}$
- Decoder function $\mathbf{g} : \mathbf{h} \rightarrow \hat{\mathbf{x}}$
- Reconstruction $\hat{\mathbf{x}}$



PCA as an autoencoder (1/2)



- Assume linear encoder and decoder:

$$\mathbf{f}(\mathbf{x}) = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$
$$\mathbf{g}(\mathbf{h}) = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

- PCA solution minimizes criterion $C = \mathbb{E}\left[\|\mathbf{x} - \hat{\mathbf{x}}\|^2\right]$.
- Note: Solution is not unique, even if restricting $\mathbf{W}^{(2)} = \mathbf{W}^{(1)\top}$.

PCA as an autoencoder (2/2)

- Just learning the identity mapping $\mathbf{g}(\mathbf{f}(\cdot)) = \mathbf{I}(\cdot)$?

$$\hat{\mathbf{x}} = \mathbf{g}(\mathbf{f}(\mathbf{x})) = \left(\mathbf{W}^{(2)}\mathbf{W}^{(1)}\right)\mathbf{x} + \left(\mathbf{W}^{(2)}\mathbf{b}^{(1)} + \mathbf{b}^{(2)}\right)$$

- We get $\hat{\mathbf{x}} = \mathbf{x}$ when

$$\mathbf{W}^{(2)} = (\mathbf{W}^{(1)})^{-1}$$

and

$$\mathbf{b}^{(2)} = -\mathbf{W}^{(2)}\mathbf{b}^{(1)} \ .$$

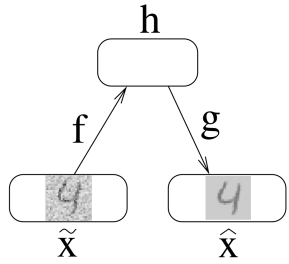
- So any encoder with an invertible $\mathbf{W}^{(1)}$ is optimal.
- *How to make the autoencoding problem harder?*

Regularized autoencoders

Regularization avoids learning the identity function:

- Bottleneck autoencoder (limit dimensionality of $\mathbf{h}$)
  (Bourlard and Kamp, 1988, Oja, 1991)
- Sparse autoencoder (penalize activations of $\mathbf{h}$) (Ranzato et al., 2006, Le et al., 2011)
- Denoising autoencoder (inject noise to input $\mathbf{x}$)
  (Vincent et al., 2008)
- Contractive autoencoder (penalize Jacobian of $\mathbf{f}(\cdot)$)
  (Rifai et al., 2011)
- Sometimes also weight sharing $\mathbf{W}^{(2)} = \mathbf{W}^{(1)\top}$.

Denoising autoencoder (Vincent et al., 2008)

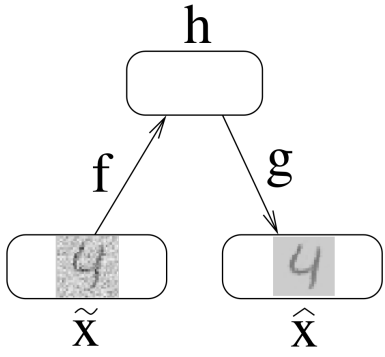


Feed corrupted inputs $\tilde{\mathbf{x}} \sim c(\tilde{\mathbf{x}}|\mathbf{x})$

- Additive noise $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}$ where e.g. $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$
- Salt noise $\tilde{\mathbf{x}} = \mathbf{m} \odot \mathbf{x}$ or $\tilde{x}_i = m_i x_i$
  where binary $m_i \sim$ Bernoulli(p)
- Masking noise $\tilde{\mathbf{x}} = [\mathbf{m} \odot \mathbf{x}; \mathbf{m}]$

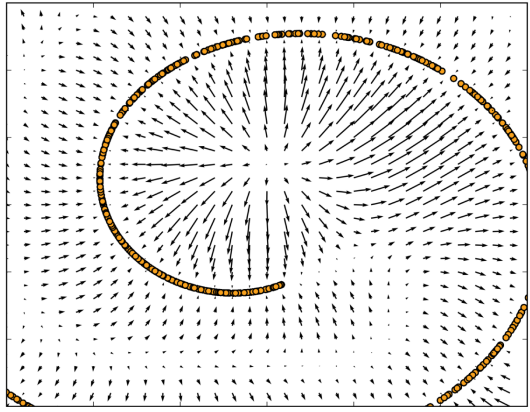
Train $\hat{\mathbf{x}} = \mathbf{g}(\mathbf{f}(\tilde{\mathbf{x}}))$ to minimize reconstruction error,
e.g. $C = \mathbb{E}\left[\|\hat{\mathbf{x}} - \mathbf{x}\|^2\right]$.

Denoising autoencoder



Basic encoder $\mathbf{h} = \mathbf{f}(\tilde{\mathbf{x}}) = \Phi\left(\mathbf{W}^{(1)}\tilde{\mathbf{x}} + \mathbf{b}^{(1)}\right)$
and decoder $\hat{\mathbf{x}} = g(\mathbf{h}) = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$.
Deep autoencoder: both $\mathbf{f}$ and $\mathbf{g}$ multi-layered.

What does denoising autoencoder learn?

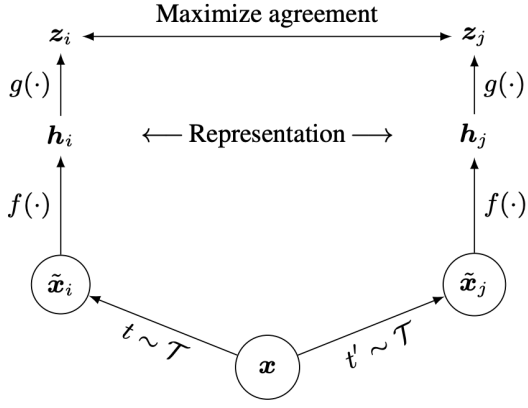


To point $\mathbf{g}(\mathbf{f}(\cdot))$ towards higher probability.

Image from (Alain and Bengio, 2014)

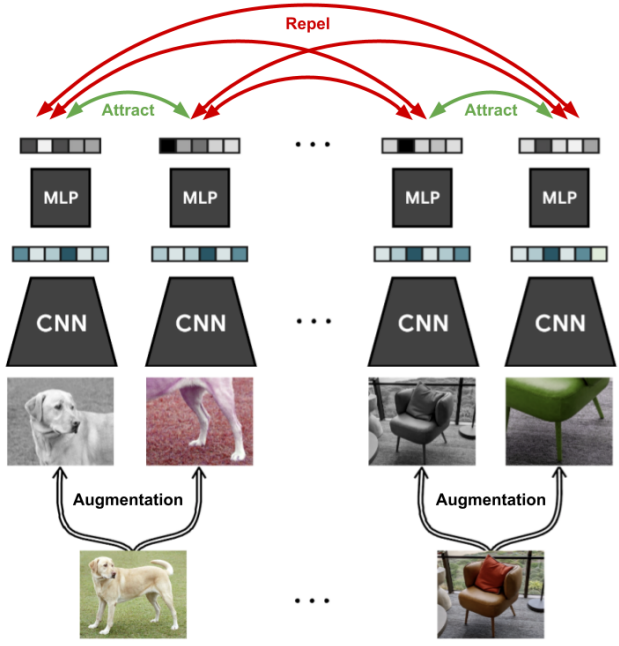
Constrastive learning

- Representation learning approach.
- Example - SIMCLR

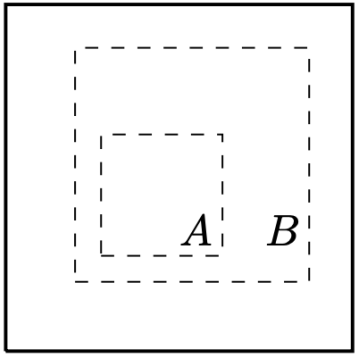


- Constrastive loss - encourage $||\mathbf{z} - \mathbf{z}'||^2$ to be small when coming from same $\mathbf{x}$ relative to the similarity calculated for different $\mathbf{x}$-values.

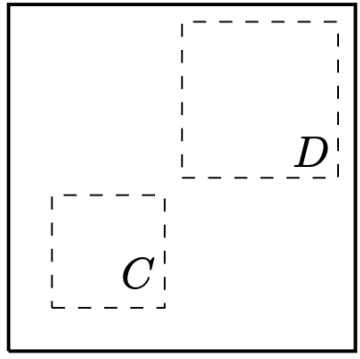
Constrative learning



SIMCLR transformations for image data



(a) Global and local views.

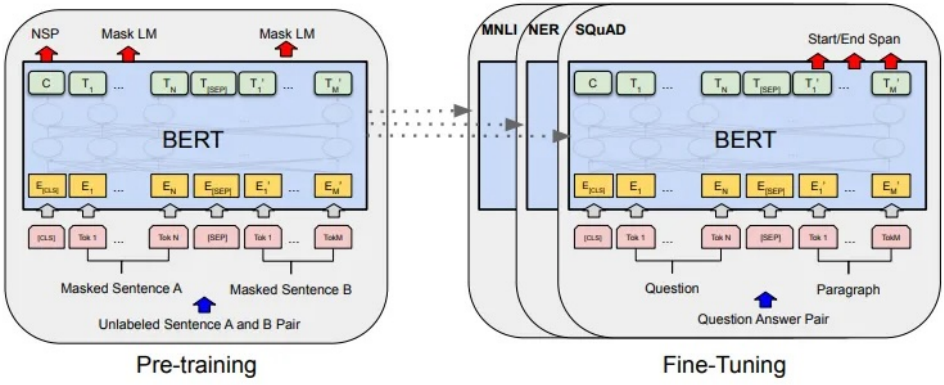


(b) Adjacent views.

- t can be A and t' be B
- t can be C and $'t'$ be D.

Masked language modelling

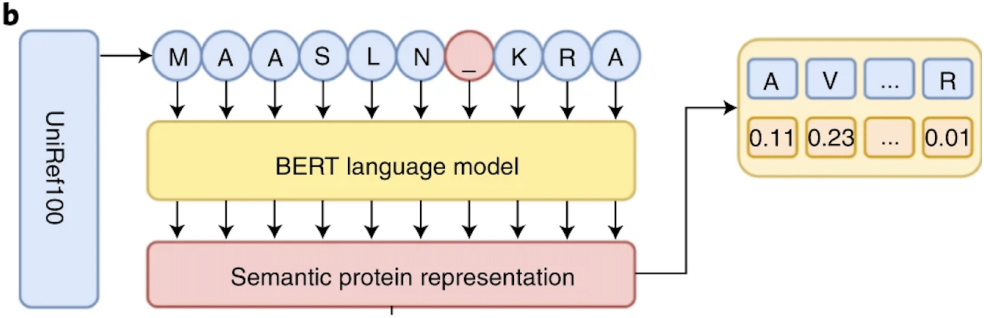
- Train on large dataset with a masked learning objective



- Throw away the prediction head and use
- Hidden representations $\mathbf{h}$, for example the last hidden layer.
- Use $\mathbf{h}$ as input to supervised method.

Devlin et al, BERT, 2018

Masked language models in biology



Teufel et al, SignalP 6, 2022

Part 3:

Generative modeling with latent variable models

Motivation

- GenAI - enough said!
- Auto-regressive

$$p(\mathbf{x}) = \prod_i p(x_i|x_{<i}) \quad x_{<i} = x_1, \ldots, x_{i-1}$$

- (Causal) transformers (week 6), RNNs (week 5)

Motivation

- GenAI - enough said!
- Auto-regressive

$$p(\mathbf{x}) = \prod_i p(x_i|x_{<i}) \quad x_{<i} = x_1, \ldots, x_{i-1}$$

- (Causal) transformers (week 6), RNNs (week 5)
- Latent variable

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$$

- This week **Variational autoencoders (VAEs)**, Generative adversarial networks (GANs) and Normalizing flows, Hierarchical VAEs (week 7), Diffusion models (week 7)

Motivation

- GenAI - enough said!
- Auto-regressive

$$p(\mathbf{x}) = \prod_i p(x_i|x_{<i}) \quad x_{<i} = x_1, \ldots, x_{i-1}$$

- (Causal) transformers (week 6), RNNs (week 5)
- Latent variable

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$$

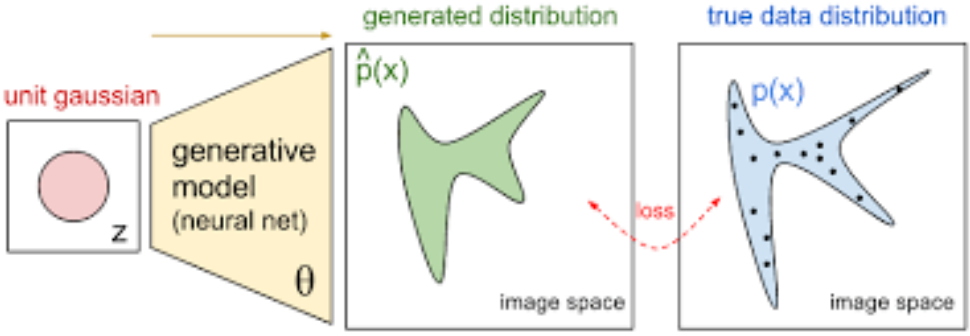
- This week **Variational autoencoders (VAEs)**, Generative adversarial networks (GANs) and Normalizing flows, Hierarchical VAEs (week 7), Diffusion models (week 7)

Roadmap

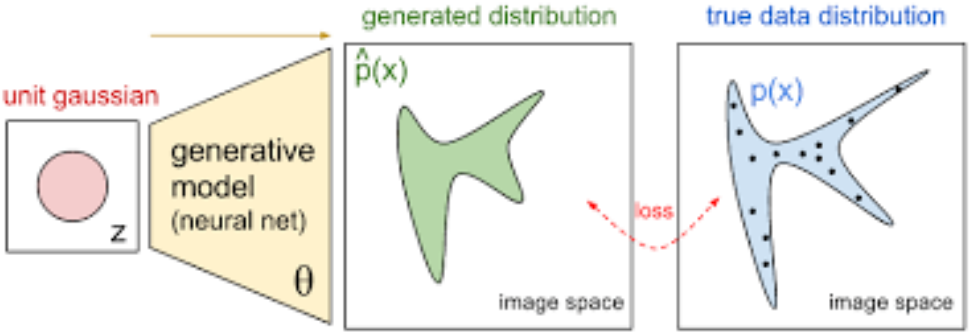
- Understanding generative models with latent variables
- Log likelihood (lower bound) for variational autoencoder
- Normalizing flows
- Generative adversarial networks (GANs)

Part 3.1:
Latent variable models

Latent variable model



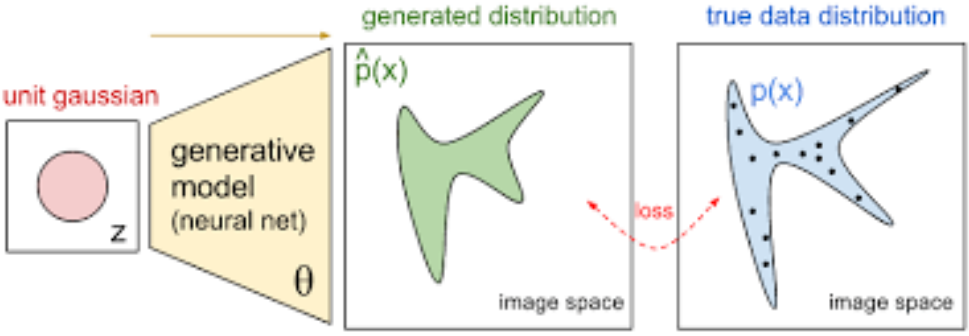
Latent variable model



- Construct distribution $\hat{p}(\mathbf{x})$ from latent $\mathbf{z}$ + transformation:

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$$

Latent variable model

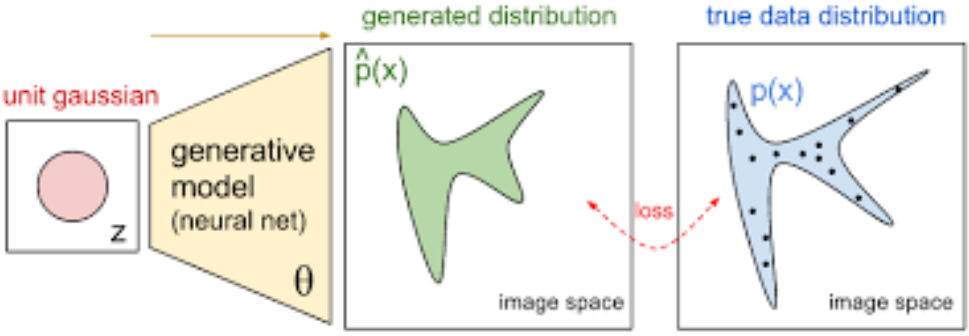


- Construct distribution $\hat{p}(\mathbf{x})$ from latent $\mathbf{z}$ + transformation:

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$$

- (Probabilistic PCA $\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\epsilon}$, $\mathbf{z} \sim \mathcal{N}(\mathbf{z}, \mathbf{0}, \mathbf{I})$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{z}, \mathbf{0}, \sigma^2\mathbf{I})$)

Latent variable model



- Construct distribution $\hat{p}(\mathbf{x})$ from latent $\mathbf{z}$ + transformation:

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$$

- (Probabilistic PCA $\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\epsilon}$, $\mathbf{z} \sim \mathcal{N}(\mathbf{z}, \mathbf{0}, \mathbf{I})$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{z}, \mathbf{0}, \sigma^2\mathbf{I})$)
- Marginalisation (treat $\mathbf{z}$ as a nuisance parameter):

$$\hat{p}(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$$

OpenAI Generative models blog

Illustrative example of a (deep) generative model - recap

Training data $\{\vec{x}_1, \ldots, \vec{x}_n\}$ binary MNIST

Generative model for $\vec{z} \in \mathbb{R}^2$ and $\vec{x} \in \{0,1\}^{28\times 28}$

$$\begin{cases} \vec{z} \sim \mathcal{N}\Big(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Big) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\vec{z})) \end{cases}$$

Decoder network

$$f(\vec{z}) = \text{Sigmoid}(\vec{V}\tanh(\vec{W}\vec{z} + \vec{b}) + \boldsymbol{\beta})$$

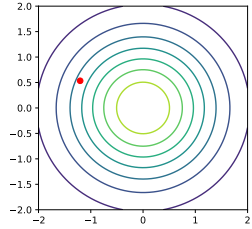
Illustrative example of a (deep) generative model - recap

Training data $\{\vec{x}_1, \ldots, \vec{x}_n\}$ binary MNIST

Generation

$$\vec{z} \sim \mathcal{N}\Big(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Big)$$

$$\vec{z} = (-1.2033, 0.5340)$$



Generative model for $\vec{z} \in \mathbb{R}^2$ and $\vec{x} \in \{0,1\}^{28 \times 28}$

$$\begin{cases} \vec{z} \sim \mathcal{N}\Big(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Big) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\vec{z})) \end{cases}$$

Decoder network

$$f(\vec{z}) = \text{Sigmoid}(\vec{V} \tanh(\vec{W}\vec{z} + \vec{b}) + \boldsymbol{\beta})$$

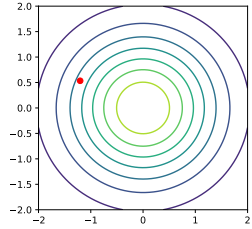
Illustrative example of a (deep) generative model - recap

Training data $\{\vec{x}_1, \ldots, \vec{x}_n\}$ binary MNIST

Generation

$$\vec{z} \sim \mathcal{N}\Big(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Big)$$

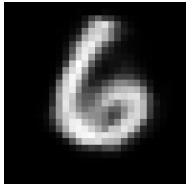
$$\vec{z} = (-1.2033, 0.5340)$$



Generative model for $\vec{z} \in \mathbb{R}^2$ and $\vec{x} \in \{0,1\}^{28 \times 28}$

$$\begin{cases} \vec{z} \sim \mathcal{N}\Big(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Big) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\vec{z})) \end{cases}$$

$f(\vec{z})$



Decoder network

$$f(\vec{z}) = \text{Sigmoid}(\vec{V} \tanh(\vec{W}\vec{z} + \vec{b}) + \boldsymbol{\beta})$$

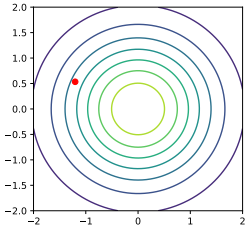
Illustrative example of a (deep) generative model - recap

Training data $\{\vec{x}_1, \ldots, \vec{x}_n\}$ binary MNIST

Generation

$$\vec{z} \sim \mathcal{N}\Big(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Big)$$

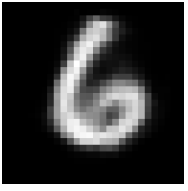
$$\vec{z} = (-1.2033, 0.5340)$$



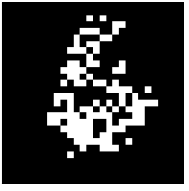
Generative model for $\vec{z} \in \mathbb{R}^2$ and $\vec{x} \in \{0,1\}^{28 \times 28}$

$$\begin{cases} \vec{z} \sim \mathcal{N}\Big(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Big) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\vec{z})) \end{cases}$$

$f(\vec{z})$



$x^{j,k} \sim \text{Bern}(f^{j,k}(\vec{z}))$



Decoder network

$$f(\vec{z}) = \text{Sigmoid}(\vec{V} \tanh(\vec{W}\vec{z} + \vec{b}) + \boldsymbol{\beta})$$

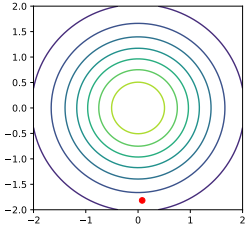
Illustrative example of a (deep) generative model - recap

Training data $\{\vec{x}_1, \ldots, \vec{x}_n\}$ binary MNIST

Generation

$$\vec{z} \sim \mathcal{N}\Big(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Big)$$

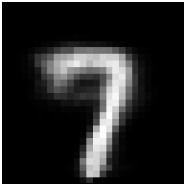
$$\vec{z} = (0.0791, -1.8165)$$



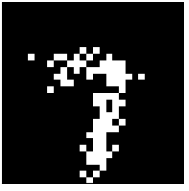
Generative model for $\vec{z} \in \mathbb{R}^2$ and $\vec{x} \in \{0,1\}^{28 \times 28}$

$$\begin{cases} \vec{z} \sim \mathcal{N}\Big(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Big) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\vec{z})) \end{cases}$$

$f(z)$



$x^{j,k} \sim \text{Bern}(f^{j,k}(z))$



Decoder network

$$f(\vec{z}) = \text{Sigmoid}(\vec{V} \tanh(\vec{W}\vec{z} + \vec{b}) + \boldsymbol{\beta})$$

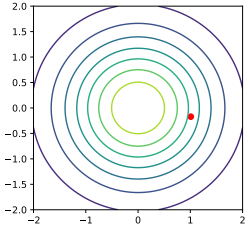
Illustrative example of a (deep) generative model - recap

Training data $\{\vec{x}_1, \ldots, \vec{x}_n\}$ binary MNIST

Generation

$$\vec{z} \sim \mathcal{N}\Big(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Big)$$

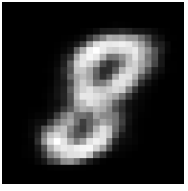
$$\vec{z} = (1.0097, -0.1711)$$



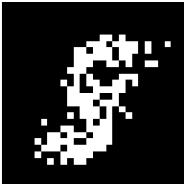
Generative model for $\vec{z} \in \mathbb{R}^2$ and $\vec{x} \in \{0,1\}^{28 \times 28}$

$$\begin{cases} \vec{z} \sim \mathcal{N}\Big(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Big) \\ x^{j,k} \sim \text{Bernoulli}(p = f^{j,k}(\vec{z})) \end{cases}$$

$f(z)$



$x^{j,k} \sim \text{Bern}(f^{j,k}(z))$

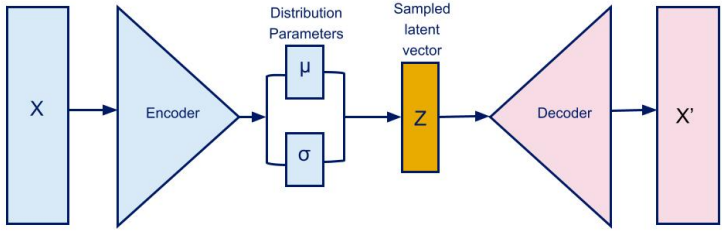


Decoder network

$$f(\vec{z}) = \text{Sigmoid}(\vec{V} \tanh(\vec{W}\vec{z} + \vec{b}) + \boldsymbol{\beta})$$

Part 3.2:
Variational autoencoder

VAE - Variational autoencoder



- The log likelihood lower bound (aka the ELBO)

$$\log p(\mathbf{x}) \geq \mathrm{ELBO}(y) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})}\right]$$

- Let us use the next slides to dicipher what is going on here!

Latent variable model - easy and hard computationsl

- Latent variable model - joint data and latent (easy)

$$p_\theta(\mathbf{x}, \mathbf{z}) = \underbrace{p_\theta(\mathbf{x}|\mathbf{z})}_{\text{Link+FNN}} \underbrace{p(\mathbf{z})}_{\mathcal{N}(\mathbf{z}|0,\mathbf{I})}$$

Latent variable model - easy and hard computationsl

- Latent variable model - joint data and latent (easy)

$$p_\theta(\mathbf{x}, \mathbf{z}) = \underbrace{p_\theta(\mathbf{x}|\mathbf{z})}_{\text{Link+FNN}} \underbrace{p(\mathbf{z})}_{\mathcal{N}(\mathbf{z}|0,\mathbf{I})}$$

- Sampling also known as inference (easy)

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$$

Latent variable model - easy and hard computationsl

- Latent variable model - joint data and latent (easy)

$$p_\theta(\mathbf{x}, \mathbf{z}) = \underbrace{p_\theta(\mathbf{x}|\mathbf{z})}_{\text{Link+FNN}} \underbrace{p(\mathbf{z})}_{\mathcal{N}(\mathbf{z}|0,\mathbf{I})}$$

- Sampling also known as inference (easy)

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$$

- Likelihood - treat $\mathbf{z}$ as so-called nuisance parameter and integrate out (hard)

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$$

Latent variable model - easy and hard computationsl

- Latent variable model - joint data and latent (easy)

$$p_\theta(\mathbf{x}, \mathbf{z}) = \underbrace{p_\theta(\mathbf{x}|\mathbf{z})}_{\text{Link+FNN}} \underbrace{p(\mathbf{z})}_{\mathcal{N}(\mathbf{z}|0,\mathbf{I})}$$

- Sampling also known as inference (easy)

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$$

- Likelihood - treat $\mathbf{z}$ as so-called nuisance parameter and integrate out (hard)

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$$

- Latent posterior from Bayes' theorem (hard to sample from and compute because of $p(\mathbf{x})$)

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}$$

Latent variable model - easy and hard computationsl

- Latent variable model - joint data and latent (easy)

$$p_\theta(\mathbf{x}, \mathbf{z}) = \underbrace{p_\theta(\mathbf{x}|\mathbf{z})}_{\text{Link+FNN}} \underbrace{p(\mathbf{z})}_{\mathcal{N}(\mathbf{z}|0,\mathbf{I})}$$

- Sampling also known as inference (easy)

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$$

- Likelihood - treat $\mathbf{z}$ as so-called nuisance parameter and integrate out (hard)

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$$

- Latent posterior from Bayes' theorem (hard to sample from and compute because of $p(\mathbf{x})$)

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}$$

- Learning (hard because $p(\mathbf{x})$ hard)

$$\theta_{\text{ML}} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \log p_\theta(\mathbf{x}_i)$$

The magic of amortized inference!

- Kingma+Welling 2013, Rezende et al 2013:
- Decoder: (example continuous observations) $\mathbf{z} \rightarrow \mathbf{x}$

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\underbrace{\boldsymbol{\mu}_\theta(\mathbf{z})}_{\text{FFNN}}, \underbrace{\text{diag}(\boldsymbol{\sigma}^2_\theta(\mathbf{z}))}_{\text{FFNN}})$$

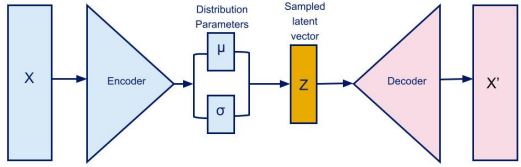
The magic of amortized inference!

- Kingma+Welling 2013, Rezende et al 2013:
- Decoder: (example continuous observations) $\mathbf{z} \rightarrow \mathbf{x}$

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\underbrace{\boldsymbol{\mu}_\theta(\mathbf{z})}_{\text{FFNN}}, \underbrace{\text{diag}(\boldsymbol{\sigma}^2_\theta(\mathbf{z}))}_{\text{FFNN}})$$

- Encoder: $\mathbf{x} \rightarrow \mathbf{z}$

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\underbrace{\boldsymbol{\mu}_\phi(\mathbf{x})}_{\text{FFNN}}, \underbrace{\text{diag}(\boldsymbol{\sigma}^2_\phi(\mathbf{x}))}_{\text{FFNN}})$$



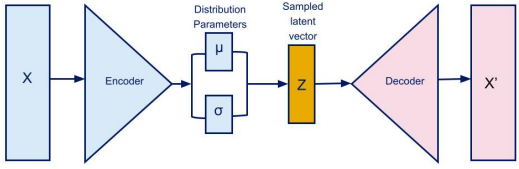
The magic of amortized inference!

- Kingma+Welling 2013, Rezende et al 2013:
- Decoder: (example continuous observations) $\mathbf{z} \rightarrow \mathbf{x}$

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\underbrace{\boldsymbol{\mu}_\theta(\mathbf{z})}_{\text{FFNN}}, \underbrace{\text{diag}(\boldsymbol{\sigma}^2_\theta(\mathbf{z}))}_{\text{FFNN}})$$

- Encoder: $\mathbf{x} \rightarrow \mathbf{z}$

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\underbrace{\boldsymbol{\mu}_\phi(\mathbf{x})}_{\text{FFNN}}, \underbrace{\text{diag}(\boldsymbol{\sigma}^2_\phi(\mathbf{x}))}_{\text{FFNN}})$$



- Amortization = learn a mapping $q_\phi(\mathbf{z}|\mathbf{x})$ used for each data point rather than
- performing the computation $p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}$ for each data point.

Log likelihood lower bound (aka ELBO)

- Bounding the log likelihood

$$
\begin{aligned}
\log p(\mathbf{x}) &= \log \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} && \leftarrow \text{exact} \\
&= \log \int \frac{q(\mathbf{z}|\mathbf{x})}{q(\mathbf{z}|\mathbf{x})} p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} && \leftarrow \text{multiply by 1} \\
&= \log \int q(\mathbf{z}|\mathbf{x}) \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} d\mathbf{z} && \leftarrow \text{rearrange} \\
&\geq \int q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} d\mathbf{z} && \leftarrow \text{Jensen's inequality} \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[\log \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \right] && \leftarrow \text{expectation notation}
\end{aligned}
$$

Log likelihood lower bound (aka ELBO)

- Bounding the log likelihood

$$
\begin{aligned}
\log p(\mathbf{x}) &= \log \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} && \leftarrow \text{exact} \\
&= \log \int \frac{q(\mathbf{z}|\mathbf{x})}{q(\mathbf{z}|\mathbf{x})} p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} && \leftarrow \text{multiply by 1} \\
&= \log \int q(\mathbf{z}|\mathbf{x}) \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} d\mathbf{z} && \leftarrow \text{rearrange} \\
&\geq \int q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} d\mathbf{z} && \leftarrow \text{Jensen's inequality} \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[\log \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \right] && \leftarrow \text{expectation notation}
\end{aligned}
$$

- Jensen's inequality - for any concave function ϕ, for example $\log$:

$$\phi\left(\mathbb{E}\left[\mathbf{x}\right]\right) \geq \mathbb{E}\left[\phi[\mathbf{x}]\right]$$

Deep variational autoencoders

- Variational objective - optimize $\sum_i \mathcal{L}_{\theta,\phi}(\mathbf{x}_i)$ wrt θ, ϕ:

$$\log p_\theta(\mathbf{x}) \geq \mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(z)}{q_\phi(\mathbf{z}|\mathbf{x})}\right]$$

$$= \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right]}_{\mathbb{E}_q[\text{log likelihood}]} + \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]}_{\text{regularization}}$$

Deep variational autoencoders

- Variational objective - optimize $\sum_i \mathcal{L}_{\theta,\phi}(\mathbf{x}_i)$ wrt θ, ϕ:

$$\log p_\theta(\mathbf{x}) \geq \mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(z)}{q_\phi(\mathbf{z}|\mathbf{x})}\right]$$
$$= \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right]}_{\mathbb{E}_q[\text{log likelihood}]} + \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]}_{\text{regularization}}$$

- Handle integration by sampling

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) \approx \frac{1}{R}\sum_{r=1}^{R} \log \frac{p_\theta(\mathbf{x}|\mathbf{z}^r)p(\mathbf{z}^r)}{q_\phi(\mathbf{z}^r|\mathbf{x})},$$
$$\mathbf{z}^r = \boldsymbol{\mu}_\phi(\mathbf{x}) + \boldsymbol{\sigma}_\phi(\mathbf{x}) \otimes \boldsymbol{\epsilon}^r$$

Deep variational autoencoders

- Variational objective - optimize $\sum_i \mathcal{L}_{\theta,\phi}(\mathbf{x}_i)$ wrt θ, ϕ:

$$\log p_\theta(\mathbf{x}) \geq \mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(z)}{q_\phi(\mathbf{z}|\mathbf{x})}\right]$$
$$= \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right]}_{\mathbb{E}_q[\text{log likelihood}]} + \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]}_{\text{regularization}}$$

- Handle integration by sampling

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) \approx \frac{1}{R}\sum_{r=1}^{R} \log \frac{p_\theta(\mathbf{x}|\mathbf{z}^r)p(\mathbf{z}^r)}{q_\phi(\mathbf{z}^r|\mathbf{x})},$$
$$\mathbf{z}^r = \boldsymbol{\mu}_\phi(\mathbf{x}) + \boldsymbol{\sigma}_\phi(\mathbf{x}) \otimes \boldsymbol{\epsilon}^r$$

- Auto-encoder: map "$\mathbf{x} \to \mathbf{z} \to \mathbf{x}$"

Deep variational autoencoders

- Variational objective - optimize $\sum_i \mathcal{L}_{\theta,\phi}(\mathbf{x}_i)$ wrt θ, ϕ:

$$\log p_\theta(\mathbf{x}) \geq \mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(z)}{q_\phi(\mathbf{z}|\mathbf{x})}\right]$$
$$= \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right]}_{\mathbb{E}_q[\text{log likelihood}]} + \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]}_{\text{regularization}}$$

- Handle integration by sampling

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) \approx \frac{1}{R}\sum_{r=1}^{R} \log \frac{p_\theta(\mathbf{x}|\mathbf{z}^r)p(\mathbf{z}^r)}{q_\phi(\mathbf{z}^r|\mathbf{x})},$$
$$\mathbf{z}^r = \boldsymbol{\mu}_\phi(\mathbf{x}) + \boldsymbol{\sigma}_\phi(\mathbf{x}) \otimes \boldsymbol{\epsilon}^r$$

- Auto-encoder: map "$\mathbf{x} \to \mathbf{z} \to \mathbf{x}$"
- This is a variational auto-encoder (VAE):

$$\text{Encoder} \quad \mathbf{z} = \mu_\phi(\mathbf{x}) + \sigma_\phi(\mathbf{x}) \otimes \boldsymbol{\epsilon}$$
$$\text{Decoder} \quad p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mu_\theta(\mathbf{z}), \text{diag}(\sigma^2_\theta(\mathbf{z})))$$

Generative models for complex data

- Once model is trained we can make it dream up new digits by:

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

$$\mathbf{x} \sim \mathcal{N}(\mathbf{x}|\underbrace{\boldsymbol{\mu}_\theta(\mathbf{z})}_{\text{FFNN}}, \underbrace{\text{diag}(\boldsymbol{\sigma}^2_\theta(\mathbf{z}))}_{\text{FFNN}})$$

Part 3.3:
Normalizing flows

Motivation - modelling high dimensional distributions

- Popular probabilistic deep generative modelling:
  - Latent variable models - Deep Boltzmann Machines [Salakhutdinov et al], Variational Autoencoders [Kingma and Welling, Rezende and Mohamed].

Motivation - modelling high dimensional distributions

- Popular probabilistic deep generative modelling:
  - Latent variable models - Deep Boltzmann Machines [Salakhutdinov et al], Variational Autoencoders [Kingma and Welling, Rezende and Mohamed].
  - Autoregressive models - Recurrent Neural Networks, NADE [Uria et al], MADE [Germain et al], WaveNet [Oord et al], PixelCNN [Oord et al]

$$p(\mathbf{x}) = p(x_1) \prod_{i=2}^{d} p(x_i | x_{<i})$$

Motivation - modelling high dimensional distributions

- Popular probabilistic deep generative modelling:
  - Latent variable models - Deep Boltzmann Machines [Salakhutdinov et al], Variational Autoencoders [Kingma and Welling, Rezende and Mohamed].
  - Autoregressive models - Recurrent Neural Networks, NADE [Uria et al], MADE [Germain et al], WaveNet [Oord et al], PixelCNN [Oord et al]

$$p(\mathbf{x}) = p(x_1) \prod_{i=2}^{d} p(x_i | x_{<i})$$

  - Flow models - Normalizing flows [Rezende and Mohamed],RealNVP [Dinh et al], GLOW [Kingma and Dhariwal], MAF [Papamakarios et al].

$$\mathbf{z} \sim p(\mathbf{z}) \text{ and } \mathbf{x} = \mathbf{g}(\mathbf{z}) \Rightarrow p(\mathbf{x}) = \left| \frac{d\mathbf{z}}{d\mathbf{x}} \right| p(\mathbf{z}) \Big|_{\mathbf{z}=\mathbf{g}^{-1}(\mathbf{x})}$$

Motivation - modelling high dimensional distributions

- Popular probabilistic deep generative modelling:
  - Latent variable models - Deep Boltzmann Machines [Salakhutdinov et al], Variational Autoencoders [Kingma and Welling, Rezende and Mohamed].
  - Autoregressive models - Recurrent Neural Networks, NADE [Uria et al], MADE [Germain et al], WaveNet [Oord et al], PixelCNN [Oord et al]

$$p(\mathbf{x}) = p(x_1) \prod_{i=2}^{d} p(x_i | x_{<i})$$

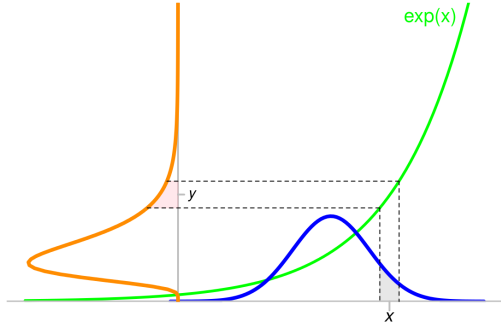
  - Flow models - Normalizing flows [Rezende and Mohamed],RealNVP [Dinh et al], GLOW [Kingma and Dhariwal], MAF [Papamakarios et al].

$$\mathbf{z} \sim p(\mathbf{z}) \text{ and } \mathbf{x} = \mathbf{g}(\mathbf{z}) \;\Rightarrow\; p(\mathbf{x}) = \left| \frac{d\mathbf{z}}{d\mathbf{x}} \right| p(\mathbf{z}) \Big|_{\mathbf{z}=\mathbf{g}^{-1}(\mathbf{x})}$$

- Non-probabilistic generative modelling:
  - Generative adversarial networks (GAN) [Goodfellow et al]

Change of variable formula

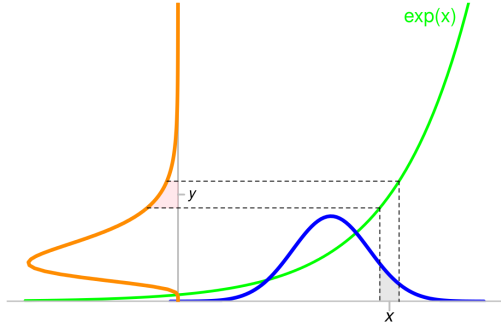
- The probability volume should be unchanged under transformation: (image source)



$$p_Y(y)dy = p_X(x)dx$$

Change of variable formula

- The probability volume should be unchanged under transformation: (image source)



$$p_Y(y)dy = p_X(x)dx$$

- Example $y = \exp(x)$ or $x(y) = \log(y)$

$$p_Y(y) = \left|\frac{d}{dy}x(y)\right| p_X(x(y)) = \left|\frac{d}{dy}\log y\right| p_X(x(y)) = \frac{1}{y}p_X(\log(y))$$

Flows

- Flows - get more flexible distribution by transformations:

$$\mathbf{z} \sim p(\mathbf{z}) \text{ and } \mathbf{x} = g(\mathbf{z}) \Rightarrow p(\mathbf{x}) = \left| \frac{d\mathbf{z}}{d\mathbf{x}} \right| p(\mathbf{z}) \Big|_{\mathbf{z}=g^{-1}(\mathbf{x})}$$

- $\left| \frac{d\mathbf{z}}{d\mathbf{x}} \right|$ is the determinant of Jacobian $J_{ij} = \frac{dz_i}{dx_j}$.

Flows

- Flows - get more flexible distribution by transformations:

$$\mathbf{z} \sim p(\mathbf{z}) \text{ and } \mathbf{x} = g(\mathbf{z}) \Rightarrow p(\mathbf{x}) = \left| \frac{d\mathbf{z}}{d\mathbf{x}} \right| p(\mathbf{z}) \Big|_{\mathbf{z}=g^{-1}(\mathbf{x})}$$

- $\left| \frac{d\mathbf{z}}{d\mathbf{x}} \right|$ is the determinant of Jacobian $J_{ij} = \frac{dz_i}{dx_j}$.
- Restrictions
  - $\mathbf{x}$ and $\mathbf{z}$ should have same dimensionality
  - Transformation should be invertible: $\mathbf{f} = \mathbf{g}^{-1}$: $\mathbf{z} = \mathbf{f}(\mathbf{x})$ and $\mathbf{x} = \mathbf{g}(\mathbf{z})$
  - Most obvious for continuous data: $\mathbf{x} = \mathbf{f}(\mathbf{z})$.

Flows

- Flows - get more flexible distribution by transformations:

$$\mathbf{z} \sim p(\mathbf{z}) \text{ and } \mathbf{x} = g(\mathbf{z}) \Rightarrow p(\mathbf{x}) = \left| \frac{d\mathbf{z}}{d\mathbf{x}} \right| p(\mathbf{z}) \Big|_{\mathbf{z}=g^{-1}(\mathbf{x})}$$

- $\left| \frac{d\mathbf{z}}{d\mathbf{x}} \right|$ is the determinant of Jacobian $J_{ij} = \frac{dz_i}{dx_j}$.
- Restrictions
  - $\mathbf{x}$ and $\mathbf{z}$ should have same dimensionality
  - Transformation should be invertible: $\mathbf{f} = \mathbf{g}^{-1}$: $\mathbf{z} = \mathbf{f}(\mathbf{x})$ and $\mathbf{x} = \mathbf{g}(\mathbf{z})$
  - Most obvious for continuous data: $\mathbf{x} = \mathbf{f}(\mathbf{z})$.
- Strength: sequence of transformations

$$\mathbf{f} = \mathbf{f}_1 \circ \mathbf{f}_2 \ldots \circ \ f_L$$
$$\mathbf{f}^{-1} = \mathbf{f}_L^{-1} \circ \mathbf{f}_{L-1}^{-1} \ldots \circ \mathbf{f}_1^{-1}$$
$$\log \left| \frac{d\mathbf{z}}{d\mathbf{x}} \right| = \sum_{i=1}^{L} \log \left| \frac{d\mathbf{h}_i}{d\mathbf{h}_{i-1}} \right|$$

with $\mathbf{h}_0 = \mathbf{x}$ and $\mathbf{h}_i = \mathbf{f}_1(\mathbf{h}_{i-1})$

Flows - examples

- Example - normalizing flows

$$\mathbf{g}(\mathbf{z}) = \mathbf{z} + \mathbf{u}F(\mathbf{w}^T\mathbf{z} + b)$$

  F is a scalar function,

- Can be stacked and has inverse Jacobian:

$$\frac{d\mathbf{g}(\mathbf{z})}{d\mathbf{z}} = \mathbf{I} + \mathbf{u}\mathbf{w}^T F'(\mathbf{w}^T\mathbf{z} + b)$$

  w easy computed determinant (matrix inversionn lemma).

Flows - examples

- Example - normalizing flows

$$\mathbf{g}(\mathbf{z}) = \mathbf{z} + \mathbf{u}F(\mathbf{w}^T\mathbf{z} + b)$$

  F is a scalar function,

- Can be stacked and has inverse Jacobian:

$$\frac{d\mathbf{g}(\mathbf{z})}{d\mathbf{z}} = \mathbf{I} + \mathbf{u}\mathbf{w}^T F'(\mathbf{w}^T\mathbf{z} + b)$$

  w easy computed determinant (matrix inversionn lemma).

- Example - affine transformations RealNVP and GLOW:

Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i,j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$
Invertible 1×1 convolution. $\mathbf{W} : [c \times c]$. See Section 3.2.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$\forall i,j : \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$	$h \cdot w \cdot \log \lvert\det(\mathbf{W})\rvert$ or $h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$ (see eq. (10))
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \texttt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \texttt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \texttt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \texttt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\texttt{sum}(\log(\lvert\mathbf{s}\rvert))$

Part 3.4:
Generative adversarial networks

VAE vs GAN

- Note: Around 2019 GANs were state of the art generative models. In 2025, diffusion models are favoured!
- Pro and cons of variational auto-encoders (VAE).
- Generative adversarial networks (GAN)
- GAN extension: CycleGAN

VAE pro and con

- Pro: variational auto-encoder computes a lower bound on the log likelihood

$$\log p(\mathbf{x}) \geq \int q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \mathbf{z}$$

- Quantitative model comparison for test data $\mathbf{x}_{\text{test}}$.: $\log p(\mathbf{x}_{\text{test}})$.
- Pro: It is a generative model - we can synthesise new data

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$$

VAE pro and con

- Pro: variational auto-encoder computes a lower bound on the log likelihood

$$\log p(\mathbf{x}) \geq \int q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \mathbf{z}$$

- Quantitative model comparison for test data $\mathbf{x}_{\text{test}}$.: $\log p(\mathbf{x}_{\text{test}})$.
- Pro: It is a generative model - we can synthesise new data

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$$

- Con: Too restricted choice of variational distribution $\rightarrow$ $q(\mathbf{z}|\mathbf{x})$ is far from

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}$$

  and the generative model we learn will be poor.
- Con: If our likelihood function $p(\mathbf{x}|\mathbf{z})$ is a poor fit to reality then the model will work poorly even for large datasets.

Generative adversarial network (GAN)

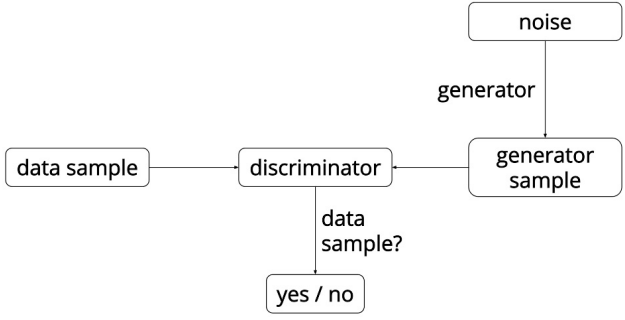
- A latent variable (implicit) generative approach
- without explicit likelihood model $p(\mathbf{x}|\mathbf{z})$

Generative adversarial network (GAN)

- A latent variable (implicit) generative approach
- without explicit likelihood model $p(\mathbf{x}|\mathbf{z})$
- Pro: Not dependent on (wrong) likelihood function
- Pro: Synthesise new data
- Con: No quantitative model comparison - only indirectly in for example semi-supervised learning.

Generative adversarial network (GAN)

- A latent variable (implicit) generative approach
- without explicit likelihood model $p(\mathbf{x}|\mathbf{z})$
- Pro: Not dependent on (wrong) likelihood function
- Pro: Synthesise new data
- Con: No quantitative model comparison - only indirectly in for example semi-supervised learning.



GAN

- GAN has two components:
  - Non-probabilistic generative model $G(z)$:

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} = \mathbf{G}(\mathbf{z})$$

    should be able generate realistic data.
  - Binary true/generated data discriminator $D(\mathbf{x})$
    should be able distinguish real and fake data.

$$D(\mathbf{x}) = \text{Probability } \mathbf{x} \text{ being true data}$$
$$1 - D(\mathbf{x}) = \text{Probability } \mathbf{x} \text{ being generated data}$$

GAN

- GAN has two components:
  - Non-probabilistic generative model $G(z)$:

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} = \mathbf{G}(\mathbf{z})$$

    should be able generate realistic data.
  - Binary true/generated data discriminator $D(\mathbf{x})$
    should be able distinguish real and fake data.

$$D(\mathbf{x}) = \text{Probability } \mathbf{x} \text{ being true data}$$
$$1 - D(\mathbf{x}) = \text{Probability } \mathbf{x} \text{ being generated data}$$

- Objective:
  - Generated data - $\mathbf{z} \sim p(\mathbf{z})$, $\mathbf{x} = \mathbf{G}(\mathbf{z})$:

$$\max_{D} \min_{\mathbf{G}} \log(1 - D(\mathbf{G}(\mathbf{z})))$$

  - True data $\mathbf{x}$:

$$\max_{D} \log D(\mathbf{x})$$

GAN

- GAN has two components:
  - Non-probabilistic generative model $G(z)$:

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\mathbf{x} = \mathbf{G}(\mathbf{z})$$

    should be able generate realistic data.
  - Binary true/generated data discriminator $D(\mathbf{x})$
    should be able distinguish real and fake data.

$$D(\mathbf{x}) = \text{Probability } \mathbf{x} \text{ being true data}$$
$$1 - D(\mathbf{x}) = \text{Probability } \mathbf{x} \text{ being generated data}$$

- Objective:
  - Generated data - $\mathbf{z} \sim p(\mathbf{z})$, $\mathbf{x} = \mathbf{G}(\mathbf{z})$:

$$\max_{D} \min_{\mathbf{G}} \log(1 - D(\mathbf{G}(\mathbf{z})))$$

  - True data $\mathbf{x}$:

$$\max_{D} \log D(\mathbf{x})$$

- In practice hard to solve this min max objective.
- Generates very nice images.

Generating faces with GAN

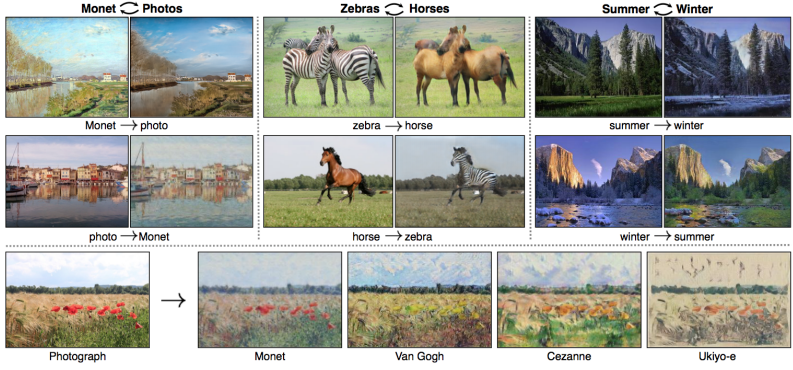
http://torch.ch/blog/2015/11/13/gan.html

Cycle GAN

- GAN very popular - really nice images.
- Many variants - here example on style transfer.
- Make two generators $\mathbf{F}$ and $\mathbf{G}$ - for example:
  - $\mathbf{F}$: Van Gogh $\rightarrow$ Cezanne style
  - $\mathbf{G}$: Cezanne $\rightarrow$ Van Gogh style

Cycle GAN

- GAN very popular - really nice images.
- Many variants - here example on style transfer.
- Make two generators $\mathbf{F}$ and $\mathbf{G}$ - for example:
  - $\mathbf{F}$: Van Gogh $\rightarrow$ Cezanne style
  - $\mathbf{G}$: Cezanne $\rightarrow$ Van Gogh style
- Objective: normal GAN +
  - Van Gogh image $\mathbf{x}$: $\mathbf{x} \approx \mathbf{G}(\mathbf{F}(\mathbf{x}))$
  - Cezanne image $\mathbf{y}$: $\mathbf{y} \approx \mathbf{F}(\mathbf{G}(\mathbf{y}))$.



References

- Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. Biological cybernetics, 59(4-5), 291–294.
- E. Oja, Data Compression, Feature Extraction, and Autoassociation in Feedforward Neural Networks. In Proc. ICANN-91, Helsinki, Finland, June 1991, pp. 737–745, Elsevier.
- M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. "Efficient learning of sparse representations with an energy-based model." Proceedings of NIPS 2006.
- Le, Q. V., Karpenko, A., Ngiam, J., and Ng, A. Y. (2011). ICA with reconstruction cost for efficient overcomplete feature learning. In Advances in Neural Information Processing Systems (pp. 1017-1025).
- Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. JMLR, 11, 3371–3408.
- Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In Proceedings of the 28th International Conference on Machine Learning (ICML-11) (pp. 833–840).
- Ballard, D. H. (1987). Modular learning in neural networks. In Proc. AAAI, pages 279–284.
- Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." Science 313.5786 (2006): 504-507.
- Y. Bengio, E. Thibodeau-Laufer, G. Alain, and J. Yosinski. Deep generative stochastic networks trainable by backprop. Technical Report arXiv:1306.1091, 2014
- A. Rasmus, H. Valpola, M. Honkala, M. Berglund, and T. Raiko. Semi-Supervised Learning with Ladder Network. In NIPS 2015.
- Särelä, J. and Valpola, H. (2005). Denoising source separation. JMLR, 6, 233–272.
- Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. JMLR, 11, 3371–3408.
- H. Valpola. From neural PCA to deep unsupervised learning. arXiv preprint arXiv:1411.7783 (2014).
- Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167

References part 2

- D P Kingma and M Welling, Auto-Encoding Variational Bayes, 2013 https://arxiv.org/pdf/1312.6114.pdf
- D J Rezende, S Mohamed and D Wierstra, Stochastic Backpropagation and Approximate Inference in Deep Generative Models, 2014 https://arxiv.org/pdf/1401.4082.pdf
- D P Kingma, D J Rezende, S Mohamed and M Welling, Semi-Supervised Learning with Deep Generative Models, 2014 https://arxiv.org/pdf/1406.5298.pdf
- I J Goodfellow et al, Generative Adversarial Networks, 2014 https://arxiv.org/pdf/1406.2661.pdf
- S Mohamed and B Lakshminarayanan. 2016 Learning in Implicit Generative Models https://arxiv.org/pdf/1610.03483.pdf
- T Salimans et al, Improved Techniques for Training GANs, 2016 https://arxiv.org/pdf/1606.03498.pdf
- J-Y Zhu et al, Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, 2017 https://arxiv.org/pdf/1703.10593.pdf
- M J Kusner, B Paige, J M Hernández-Lobato, Grammar Variational Autoencoder, 2017 https://arxiv.org/pdf/1703.01925.pdf

Thanks!
Ole Winther