Variational Autoencoder

Auto-encoding Variational Bayes by Kingma and Welling [2013]

Wonbong Jang

Department of Statistics, LSE

11th December, 2018

Outline

Probabilistic Modeling

- Learn a model that describes the data
- Using Bayes Rule, we can learn and infer about the model and its parameters
- Bayes' Rule : $P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$
- Question : How to compute the posterior? (especially when it is not tractable) and how can we compute the predictive distribution based on the updated parameters?
- Answer : Variational Bayes, Markov Chain Mote Carlo (MCMC)..
- The problem is that both methods have some issues

Probabilistic Modeling

- Learn a model that describes the data
- Using Bayes Rule, we can learn and infer about the model and its parameters
- Bayes' Rule : $P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$
- Question : How to compute the posterior? (especially when it is not tractable) and how can we compute the predictive distribution based on the updated parameters?
- Answer : Variational Bayes, Markov Chain Mote Carlo (MCMC)..
- The problem is that both methods have some issues

Probabilistic Modeling

- Learn a model that describes the data
- Using Bayes Rule, we can learn and infer about the model and its parameters
- Bayes' Rule : $P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$
- Question : How to compute the posterior? (especially when it is not tractable) and how can we compute the predictive distribution based on the updated parameters?
- Answer : Variational Bayes, Markov Chain Mote Carlo (MCMC)..
- The problem is that both methods have some issues

What are the problems?

- **Intractability** : We still need the exact solution for simplified cases in Variational Bayes.
- **Inscalability** : MCMC is asymtotically unbiased but it is computationally expensive, so it is not easy to scale up.
- Any ideas?
- Neural Network can be a good candidate.
  - Rather than programming in explicit detail, just write a program how to learn, and show computers many examples. Then, the computer learns it by itself.
  - We can do that with the development of computing

What are the problems?

- **Intractability** : We still need the exact solution for simplified cases in Variational Bayes.
- **Inscalability** : MCMC is asymtotically unbiased but it is computationally expensive, so it is not easy to scale up.
- Any ideas?
- Neural Network can be a good candidate.
  - Rather than programming in explicit detail, just write a program how to learn, and show computers many examples. Then, the computer learns it by itself.
  - We can do that with the development of computing

What are the problems?

- **Intractability** : We still need the exact solution for simplified cases in Variational Bayes.
- **Inscalability** : MCMC is asymtotically unbiased but it is computationally expensive, so it is not easy to scale up.
- Any ideas?
- Neural Network can be a good candidate.
  - Rather than programming in explicit detail, just write a program how to learn, and show computers many examples. Then, the computer learns it by itself.
  - We can do that with the development of computing

What are the problems?

- **Intractability** : We still need the exact solution for simplified cases in Variational Bayes.
- **Inscalability** : MCMC is asymtotically unbiased but it is computationally expensive, so it is not easy to scale up.
- Any ideas?
- Neural Network can be a good candidate.
  - Rather than programming in explicit detail, just write a program how to learn, and show computers many examples. Then, the computer learns it by itself.
  - We can do that with the development of computing

What are the problems?

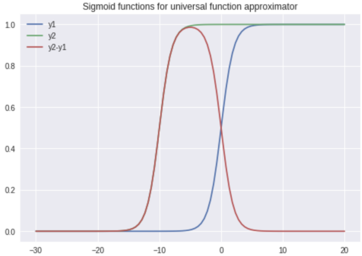
- **Intractability** : We still need the exact solution for simplified cases in Variational Bayes.
- **Inscalability** : MCMC is asymtotically unbiased but it is computationally expensive, so it is not easy to scale up.
- Any ideas?
- Neural Network can be a good candidate.
  - Rather than programming in explicit detail, just write a program how to learn, and show computers many examples. Then, the computer learns it by itself.
  - We can do that with the development of computing

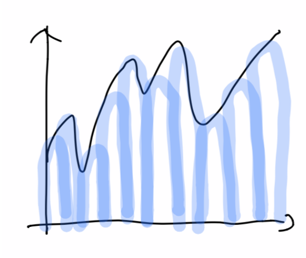
What are the problems?

- **Intractability** : We still need the exact solution for simplified cases in Variational Bayes.
- **Inscalability** : MCMC is asymtotically unbiased but it is computationally expensive, so it is not easy to scale up.
- Any ideas?
- Neural Network can be a good candidate.
  - Rather than programming in explicit detail, just write a program how to learn, and show computers many examples. Then, the computer learns it by itself.
  - We can do that with the development of computing

Why Neural Networks?

- Neural Networks with one hidden layer are universal function approximators.

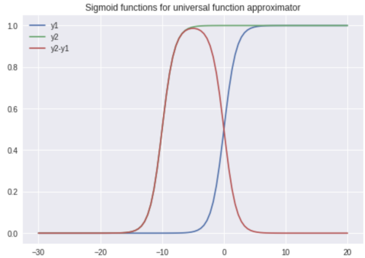


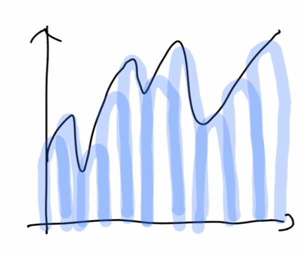


- By translating and scaling the bump, we can approximate arbitrary functions. see more on Cybenko [1989] and reference this link
- If we go deeper, we can compute more efficiently. See Montufar et al. [2014] for more.

Why Neural Networks?

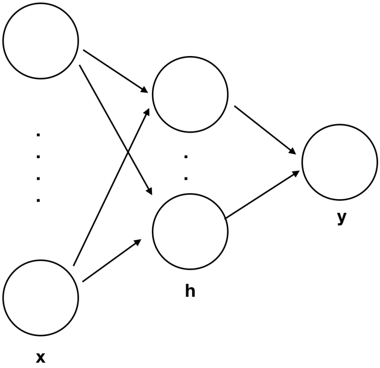
- Neural Networks with one hidden layer are universal function approximators.





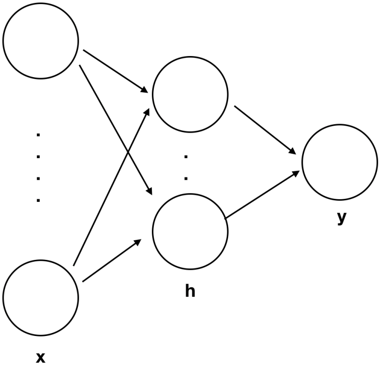
- By translating and scaling the bump, we can approximate arbitrary functions. see more on Cybenko [1989] and reference this link
- If we go deeper, we can compute more efficiently. See Montufar et al. [2014] for more.

Neural Network - example



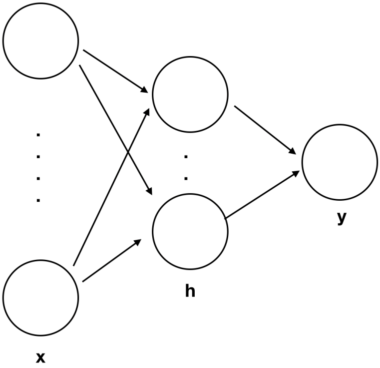
- $h_{ik} = a(b^h_k + \sum_j W^h_{jk} x_{ij})$ where $a(x) = \max(x, 0)$, $a(x)$ here is Rectified Linear Unit but it can be other functions.
- $\hat{y} = \sigma(b^o + \sum_k W^o_k h_{ik})$ where $\sigma(x)$ is usually a sigmoid function for binary classification.
- The parameters$(b^h_k, b^o, W^h_{jk}, W^o_k)$ are optimized by Gradient Descent.

Neural Network - example



- $h_{ik} = a(b_k^h + \sum_j W_{jk}^h x_{ij})$ where $a(x) = \max(x, 0)$, $a(x)$ here is Rectified Linear Unit but it can be other functions.
- $\hat{y} = \sigma(b^o + \sum_k W_k^o h_{ik})$ where $\sigma(x)$ is usually a sigmoid function for binary classification.
- The parameters$(b_k^h, b^o, W_{jk}^h, W_k^o)$ are optimized by Gradient Descent.

Neural Network - example



- $h_{ik} = a(b_k^h + \sum_j W_{jk}^h x_{ij})$ where $a(x) = \max(x, 0)$, $a(x)$ here is Rectified Linear Unit but it can be other functions.
- $\hat{y} = \sigma(b^o + \sum_k W_k^o h_{ik})$ where $\sigma(x)$ is usually a sigmoid function for binary classification.
- The parameters$(b_k^h, b^o, W_{jk}^h, W_k^o)$ are optimized by Gradient Descent.

What is Neural Network?

- **Backpropagation** is used to optimize the parameters. Neural network computes the loss and the gradients, then back-propagate error signals to get derivatives for learning
  - If small changes in parameters increase the probability of the outcome, then keep the change, otherwise drop them.
  - It enables parallel computing, so it is much more efficient. (But, different results for every iterations)
  - After defining the loss function and setting the hyper-parameter for updating the parameter, then the neural network optimizes its parameters automatically.
    $W^h_{jk} = W^h_{jk} - \alpha \frac{\partial J}{\partial W^h_{jk}}, \ \frac{\partial J}{\partial W^h_{jk}} = \frac{\partial J}{\partial \hat{h}} \frac{\partial \hat{h}}{\partial W^h_{jk}}$
- Is the loss function convex?

What is Neural Network?

- **Backpropagation** is used to optimize the parameters. Neural network computes the loss and the gradients, then back-propagate error signals to get derivatives for learning
  - If small changes in parameters increase the probability of the outcome, then keep the change, otherwise drop them.
  - It enables parallel computing, so it is much more efficient. (But, different results for every iterations)
  - After defining the loss function and setting the hyper-parameter for updating the parameter, then the neural network optimizes its parameters automatically.
    $W^h_{jk} = W^h_{jk} - \alpha \frac{\partial J}{\partial W^h_{jk}}, \frac{\partial J}{\partial W^h_{jk}} = \frac{\partial J}{\partial \hat{h}} \frac{\partial \hat{h}}{\partial W^h_{jk}}$
- Is the loss function convex?

What is Neural Network?

- **Backpropagation** is used to optimize the parameters. Neural network computes the loss and the gradients, then back-propagate error signals to get derivatives for learning
  - If small changes in parameters increase the probability of the outcome, then keep the change, otherwise drop them.
  - It enables parallel computing, so it is much more efficient. (But, different results for every iterations)
  - After defining the loss function and setting the hyper-parameter for updating the parameter, then the neural network optimizes its parameters automatically.
    $W^h_{jk} = W^h_{jk} - \alpha \frac{\partial J}{\partial W^h_{jk}}, \frac{\partial J}{\partial W^h_{jk}} = \frac{\partial J}{\partial \hat{h}} \frac{\partial \hat{h}}{\partial W^h_{jk}}$
- Is the loss function convex?

What is Neural Network?

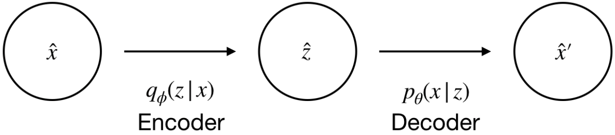
- **Backpropagation** is used to optimize the parameters. Neural network computes the loss and the gradients, then back-propagate error signals to get derivatives for learning
  - If small changes in parameters increase the probability of the outcome, then keep the change, otherwise drop them.
  - It enables parallel computing, so it is much more efficient. (But, different results for every iterations)
  - After defining the loss function and setting the hyper-parameter for updating the parameter, then the neural network optimizes its parameters automatically.
    $W_{jk}^{h} = W_{jk}^{h} - \alpha \frac{\partial J}{\partial W_{jk}^{h}}, \ \frac{\partial J}{\partial W_{jk}^{h}} = \frac{\partial J}{\partial \hat{h}} \frac{\partial \hat{h}}{\partial W_{jk}^{h}}$
- Is the loss function convex?

What is Neural Network?

- **Backpropagation** is used to optimize the parameters. Neural network computes the loss and the gradients, then back-propagate error signals to get derivatives for learning
  - If small changes in parameters increase the probability of the outcome, then keep the change, otherwise drop them.
  - It enables parallel computing, so it is much more efficient. (But, different results for every iterations)
  - After defining the loss function and setting the hyper-parameter for updating the parameter, then the neural network optimizes its parameters automatically.
    $W^h_{jk} = W^h_{jk} - \alpha \frac{\partial J}{\partial W^h_{jk}}, \ \frac{\partial J}{\partial W^h_{jk}} = \frac{\partial J}{\partial \hat{h}} \frac{\partial \hat{h}}{\partial W^h_{jk}}$
- Is the loss function convex?

From a Recognition model

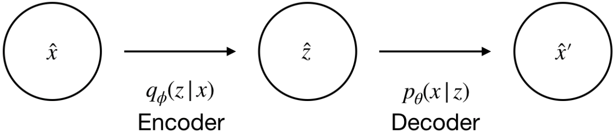
- Use the idea from the recognition model : Approximate the posterior as $q_\phi(z|x)$ rather than $p_\theta(z|x)$ directly. Then, use the neural network to approximate it.



- Objective: 1) Maximize the log $p_\theta(\hat{x}')$ and 2) minimize the KL divergence between $q_\phi(z|x)$ and $p_\theta(z|x)$
- Variance Lower Bound(L) can be defined as this. $L(\theta, \phi) = \log p_\theta(\hat{x}) - D_{KL}(q_\phi(z|x)||p_\theta(z|x))$

From a Recognition model

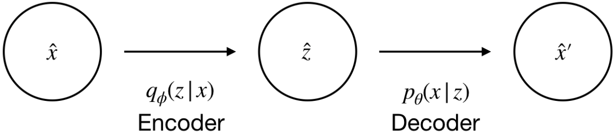
- Use the idea from the recognition model : Approximate the posterior as $q_\phi(z|x)$ rather than $p_\theta(z|x)$ directly. Then, use the neural network to approximate it.



- Objective: 1) Maximize the log $p_\theta(\hat{x}')$ and 2) minimize the KL divergence between $q_\phi(z|x)$ and $p_\theta(z|x)$
- Variance Lower Bound(L) can be defined as this. $L(\theta, \phi) = \log p_\theta(\hat{x}) - D_{KL}(q_\phi(z|x)||p_\theta(z|x))$

From a Recognition model

- Use the idea from the recognition model : Approximate the posterior as $q_\phi(z|x)$ rather than $p_\theta(z|x)$ directly. Then, use the neural network to approximate it.



- Objective: 1) Maximize the log $p_\theta(\hat{x}')$ and 2) minimize the KL divergence between $q_\phi(z|x)$ and $p_\theta(z|x)$
- Variance Lower Bound(L) can be defined as this. $L(\theta, \phi) = \log p_\theta(\hat{x}) - D_{KL}(q_\phi(z|x)||p_\theta(z|x))$

Objective

- Start from $\log p_\theta(x)$

$$\begin{aligned}\log p_\theta(x) &= \int dz\ q_\phi(z|x)\ \log p_\theta(x) \\ &= \int dz\ q_\phi(z|x) \log[\frac{p_\theta(x,z)}{q_\phi(z|x)} \frac{q_\phi(z|x)}{p_\theta(z|x)}] \\ &= D_{KL}(q_\phi(z|x)||p_\theta(z|x)) + \int dz\ q_\phi(z|x) \log \frac{p_\theta(x,z)}{q_\phi(z|x)}\end{aligned}$$

- Now, go back to Variance Lower Bound again.

$$\begin{aligned}L(\theta,\phi) &= \int dz\ q_\phi(z|x) \log \frac{p_\theta(x,z)}{q_\phi(z|x)} = \int dz\ q_\phi(z|x) \log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)} \\ &= \mathop{\mathbb{E}}_{z\sim q_\phi(z|x)} [\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p_\theta(z))\end{aligned}$$

Objective

- Start from $\log p_\theta(x)$

$$\log p_\theta(x) = \int dz\ q_\phi(z|x)\ \log p_\theta(x)$$
$$= \int dz\ q_\phi(z|x) \log[\frac{p_\theta(x,z)}{q_\phi(z|x)} \frac{q_\phi(z|x)}{p_\theta(z|x)}]$$
$$= D_{KL}(q_\phi(z|x)||p_\theta(z|x)) + \int dz\ q_\phi(z|x) \log \frac{p_\theta(x,z)}{q_\phi(z|x)}$$

- Now, go back to Variance Lower Bound again.

$$L(\theta,\phi) = \int dz\ q_\phi(z|x) \log \frac{p_\theta(x,z)}{q_\phi(z|x)} = \int dz\ q_\phi(z|x) \log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)}$$
$$= \mathop{\mathbb{E}}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p_\theta(z))$$

Optimizing the Variance Lower Bound

- By KL-Divergence's non-negativity, $L(\theta, \phi) \leq \log p_\theta(x)$
- Therefore, by optimizing the parameters θ and ϕ, we can increase the evidence lower bound every iteration, similar to Expectation Maximization algorithm.
- The problem is how to optimize the parameters - and we will use Neural Networks to do that automatically.
- Here comes another question : How to get back-propagate the sampling? in other words, how to get the derivatives from sampling.

Optimizing the Variance Lower Bound

- By KL-Divergence's non-negativity, $L(\theta, \phi) \leq \log p_\theta(x)$
- Therefore, by optimizing the parameters θ and ϕ, we can increase the evidence lower bound every iteration, similar to Expectation Maximization algorithm.
- The problem is how to optimize the parameters - and we will use Neural Networks to do that automatically.
- Here comes another question : How to get back-propagate the sampling? in other words, how to get the derivatives from sampling.

Optimizing the Variance Lower Bound

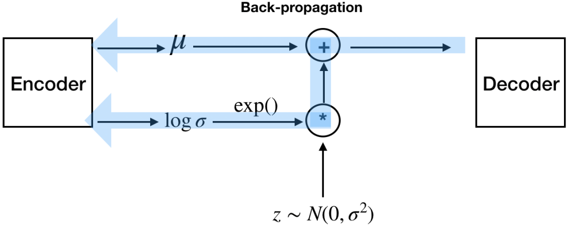
- By KL-Divergence's non-negativity, $L(\theta, \phi) \leq \log p_\theta(x)$
- Therefore, by optimizing the parameters θ and ϕ, we can increase the evidence lower bound every iteration, similar to Expectation Maximization algorithm.
- The problem is how to optimize the parameters - and we will use Neural Networks to do that automatically.
- Here comes another question : How to get back-propagate the sampling? in other words, how to get the derivatives from sampling.

Optimizing the Variance Lower Bound

- By KL-Divergence's non-negativity, $L(\theta, \phi) \leq \log p_\theta(x)$
- Therefore, by optimizing the parameters θ and ϕ, we can increase the evidence lower bound every iteration, similar to Expectation Maximization algorithm.
- The problem is how to optimize the parameters - and we will use Neural Networks to do that automatically.
- Here comes another question : How to get back-propagate the sampling? in other words, how to get the derivatives from sampling.

Reparameterization Trick

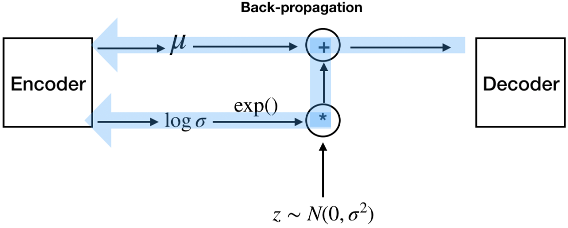
- We cannot get the derivatives for sampling, but if we assume that $z \sim N(0, \sigma^2)$, then we can compute the $D_{KL}(q_\phi(z|x)||p(z))$ with the exact solution.



- We can compute the gradient by the above reparameterization trick, and we will use those gradients for backpropagation.

Reparameterization Trick

- We cannot get the derivatives for sampling, but if we assume that $z \sim N(0, \sigma^2)$, then we can compute the $D_{KL}(q_\phi(z|x)||p(z))$ with the exact solution.



- We can compute the gradient by the above reparameterization trick, and we will use those gradients for backpropagation.

Computing KL-divergence

- Let's compute $D_{KL}(q_\phi(z|x)||p(z))$ when $q_\phi(z|x) \sim N(\mu, \sigma^2)$ and $p(z) \sim N(0, I)$.

$$\int dz\ q_\phi(z|x) \log p(z) = \int dz\ N(z; \mu, \sigma^2) \log N(z; 0, I)$$
$$= -\frac{J}{2}\log(2\pi) - \frac{1}{2}\sum_j \mu_j^2 + \sigma_j^2$$

$$\int dz\ q_\phi(z|x) \log q_\phi(z|x) = \int dz\ N(z; \mu, \sigma^2) \log N(z; \mu, \sigma^2)$$
$$= -\frac{J}{2}\log 2\pi - \frac{1}{2}\sum_j 1 + \log(\sigma_j^2)$$

$$D_{KL}(q_\phi(z|x)||p(z)) = \frac{1}{2}\sum_j 1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2$$

Computing KL-divergence

- Let's compute $D_{KL}(q_\phi(z|x)||p(z))$ when $q_\phi(z|x) \sim N(\mu, \sigma^2)$ and $p(z) \sim N(0, I)$.

$$\int dz\ q_\phi(z|x) \log p(z) = \int dz\ N(z; \mu, \sigma^2) \log N(z; 0, I)$$
$$= -\frac{J}{2}\log(2\pi) - \frac{1}{2}\sum_j \mu_j^2 + \sigma_j^2$$

$$\int dz\ q_\phi(z|x) \log q_\phi(z|x) = \int dz\ N(z; \mu, \sigma^2) \log N(z; \mu, \sigma^2)$$
$$= -\frac{J}{2}\log 2\pi - \frac{1}{2}\sum_j 1 + \log(\sigma_j^2)$$

$$D_{KL}(q_\phi(z|x)||p(z)) = \frac{1}{2}\sum_j 1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2$$

Computing KL-divergence

- Let's compute $D_{KL}(q_\phi(z|x)||p(z))$ when $q_\phi(z|x) \sim N(\mu, \sigma^2)$ and $p(z) \sim N(0, I)$.

$$\begin{aligned}\int dz\ q_\phi(z|x) \log p(z) &= \int dz\ N(z; \mu, \sigma^2) \log N(z; 0, I) \\ &= -\frac{J}{2}\log(2\pi) - \frac{1}{2}\sum_j \mu_j^2 + \sigma_j^2\end{aligned}$$

$$\begin{aligned}\int dz\ q_\phi(z|x) \log q_\phi(z|x) &= \int dz\ N(z; \mu, \sigma^2) \log N(z; \mu, \sigma^2) \\ &= -\frac{J}{2}\log 2\pi - \frac{1}{2}\sum_j 1 + \log(\sigma_j^2)\end{aligned}$$

$$D_{KL}(q_\phi(z|x)||p(z)) = \frac{1}{2}\sum_j 1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2$$

Computing KL-divergence

- Let's compute $D_{KL}(q_\phi(z|x)||p(z))$ when $q_\phi(z|x) \sim N(\mu, \sigma^2)$ and $p(z) \sim N(0, I)$.

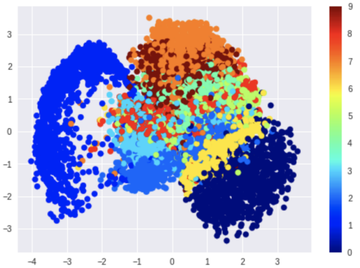
$$\begin{aligned}\int dz\ q_\phi(z|x)\log p(z) &= \int dz\ N(z;\mu,\sigma^2)\log N(z;0,I)\\ &= -\frac{J}{2}\log(2\pi) - \frac{1}{2}\sum_j \mu_j^2 + \sigma_j^2\end{aligned}$$

$$\begin{aligned}\int dz\ q_\phi(z|x)\log q_\phi(z|x) &= \int dz\ N(z;\mu,\sigma^2)\log N(z;\mu,\sigma^2)\\ &= -\frac{J}{2}\log 2\pi - \frac{1}{2}\sum_j 1 + \log(\sigma_j^2)\end{aligned}$$

$$D_{KL}(q_\phi(z|x)||p(z)) = \frac{1}{2}\sum_j 1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2$$

VAE with MNIST dataset

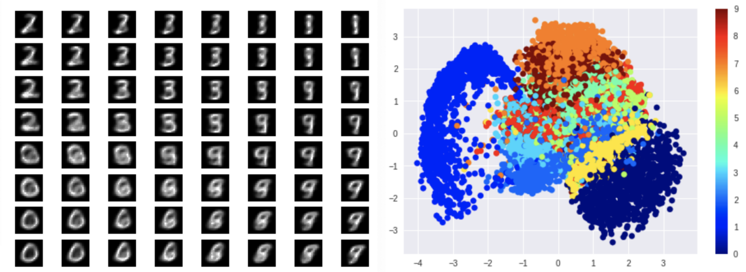
- Use VAE to generate MNIST data



- How about using conditional variables to control the generating process?

VAE with MNIST dataset

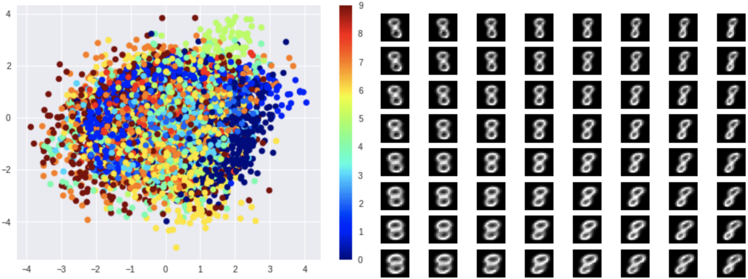
- Use VAE to generate MNIST data



- How about using conditional variables to control the generating process?

Conditional VAE

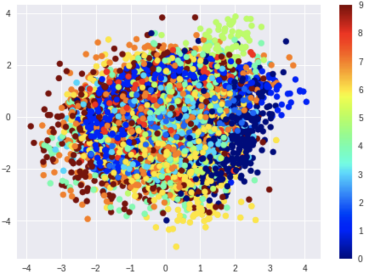
- For CVAE, Variance Lower Bound is almost the same :
  $L(\theta, \phi) = \mathbb{E}_{z \sim q_\phi(z|x)}[\log p_\theta(x|z, c)] - D_{KL}(q_\phi(z|x, c)||p_\theta(z))$



- Using the proper conditional variable stabilizes the training and helps the neural network learn better. (See Sohn et al. [2015])

Conditional VAE

- For CVAE, Variance Lower Bound is almost the same :
  $L(\theta, \phi) = \mathbb{E}_{z \sim q_\phi(z|x)}[\log p_\theta(x|z, c)] - D_{KL}(q_\phi(z|x, c)||p_\theta(z))$



- Using the proper conditional variable stabilizes the training and helps the neural network learn better. (See Sohn et al. [2015])

Now, we have studied VAE model.

- It is based on solid theoretical background.
- We can see the encoded latent space directly, and control the output if the latent space is sufficiently disentangled.
- Criticisms
  - Why do we approximate the posterior distribution with Gaussian with diagonal covariance matrix?
  - The output on high-frequency parts(e.g. edges) tends to be blurrier than other generative models such as GAN

Now, we have studied VAE model.

- It is based on solid theoretical background.
- We can see the encoded latent space directly, and control the output if the latent space is sufficiently disentangled.
- Criticisms
  - Why do we approximate the posterior distribution with Gaussian with diagonal covariance matrix?
  - The output on high-frequency parts(e.g. edges) tends to be blurrier than other generative models such as GAN

Now, we have studied VAE model.

- It is based on solid theoretical background.
- We can see the encoded latent space directly, and control the output if the latent space is sufficiently disentangled.
- Criticisms
    - Why do we approximate the posterior distribution with Gaussian with diagonal covariance matrix?
    - The output on high-frequency parts(e.g. edges) tends to be blurrier than other generative models such as GAN

Disentanglement

- Neural Network suffers from a lack of interpretability. Disentanglement can be a way to interpret the results. Higgins et al. [2016] proposed β-VAE

  $$\max_{\theta,\phi} \mathop{\mathbb{E}}_{x\sim p_D} \mathop{\mathbb{E}}_{z\sim q_\phi(z|x)} \log p_\theta(x|z) \ s.t \ D_{KL}(q_\phi(z|x)||p(z)) \leq \beta$$

  $$E_{x\sim p(x)}[D_{KL}(q(z|x)||p(z))] = I_q(x;z) + D_{KL}(q(z)||p(z))$$

- β can be viewed as how much we penalize the $q_\phi(z|x)$, as we increase β, it is more likely to be disentangled but the quality of the output is also likely to suffer too because $I_q(x;z)$ is also effected.
- Kim and Mnih [2018] proposed factor-VAE which uses additional classifier to check whether the latent space is factorized.

Disentanglement

- Neural Network suffers from a lack of interpretability. Disentanglement can be a way to interpret the results. Higgins et al. [2016] proposed β-VAE

  $$\max_{\theta,\phi} \mathop{\mathbb{E}}_{x\sim p_D} \mathop{\mathbb{E}}_{z\sim q_\phi(z|x)} \log p_\theta(x|z) \ s.t \ D_{KL}(q_\phi(z|x)||p(z)) \leq \beta$$

  $$E_{x\sim p(x)}[D_{KL}(q(z|x)||p(z))] = I_q(x;z) + D_{KL}(q(z)||p(z))$$

- β can be viewed as how much we penalize the $q_\phi(z|x)$, as we increase β, it is more likely to be disentangled but the quality of the output is also likely to suffer too because $I_q(x;z)$ is also effected.
- Kim and Mnih [2018] proposed factor-VAE which uses additional classifier to check whether the latent space is factorized.

Disentanglement

- Neural Network suffers from a lack of interpretability. Disentanglement can be a way to interpret the results. Higgins et al. [2016] proposed β-VAE

  $$\max_{\theta,\phi} \mathop{\mathbb{E}}_{x \sim p_D} \mathop{\mathbb{E}}_{z \sim q_\phi(z|x)} \log p_\theta(x|z) \; s.t \; D_{KL}(q_\phi(z|x)||p(z)) \leq \beta$$

  $$E_{x \sim p(x)}[D_{KL}(q(z|x)||p(z))] = I_q(x;z) + D_{KL}(q(z)||p(z))$$
- β can be viewed as how much we penalize the $q_\phi(z|x)$, as we increase β, it is more likely to be disentangled but the quality of the output is also likely to suffer too because $I_q(x;z)$ is also effected.
- Kim and Mnih [2018] proposed factor-VAE which uses additional classifier to check whether the latent space is factorized.

Flow models

- Use normalizing flow to approximate the posterior distribution. By change of variables, we can derive it as below.

$$\log q(z_T|x) = \log q(z_0|x) - \sum_{t=1}^{T} \log \det \left|\frac{dz_t}{dz_{t-1}}\right|$$

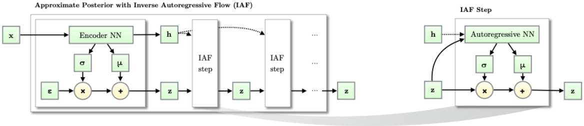


Image from Kingma et al. [2016]

- Using this model, we can approximate the posterior distribution with non-diagonal convariance matrix terms.
- See Germain et al. [2015] Kingma et al. [2016], and Papamakarios et al. [2017] for more developments.

Flow models

- Use normalizing flow to approximate the posterior distribution. By change of variables, we can derive it as below.

$$\log q(z_T|x) = \log q(z_0|x) - \sum_{t=1}^{T} \log \det \left|\frac{dz_t}{dz_{t-1}}\right|$$

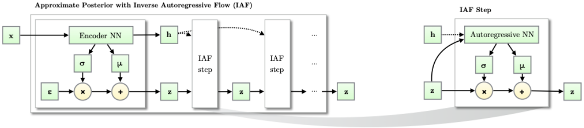
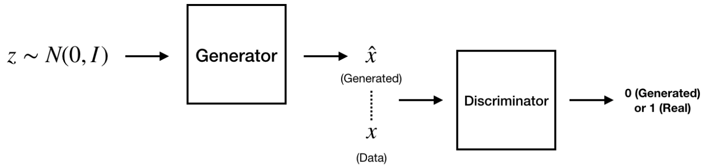


Image from Kingma et al. [2016]

- Using this model, we can approximate the posterior distribution with non-diagonal convariance matrix terms.
- See Germain et al. [2015] Kingma et al. [2016], and Papamakarios et al. [2017] for more developments.

Generative Adversarial Network - GAN

- Now compare with another popular generative model - GAN by Goodfellow et al. [2014]

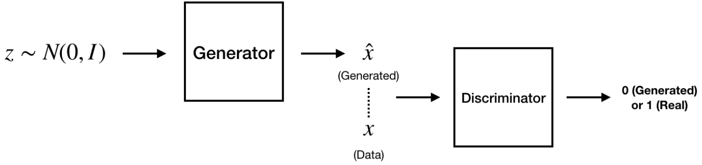


$$\min_G \max_D L(D, G) = \mathbb{E}_{x \sim p_r(x)} \log D(x) + \mathbb{E}_{z \sim p_z(z)} \log(1 - D(G(z)))$$

- GAN tries to find the Nash Equilibrium between two networks. At the equilibrium, $D(x) = \frac{1}{2}$
- If it is properly trained, the loss function is Jensen-Shannon divergence between two distributions

Generative Adversarial Network - GAN

- Now compare with another popular generative model - GAN by Goodfellow et al. [2014]

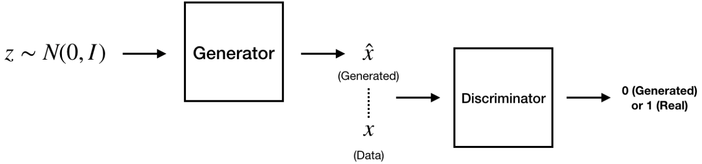


$$\min_G \max_D L(D, G) = \mathop{\mathbb{E}}_{x \sim p_r(x)} \log D(x) + \mathop{\mathbb{E}}_{z \sim p_z(z)} \log(1 - D(G(z)))$$

- GAN tries to find the Nash Equilibrium between two networks. At the equilibrium, $D(x) = \frac{1}{2}$
- If it is properly trained, the loss function is Jensen-Shannon divergence between two distributions

Generative Adversarial Network - GAN

- Now compare with another popular generative model - GAN by Goodfellow et al. [2014]

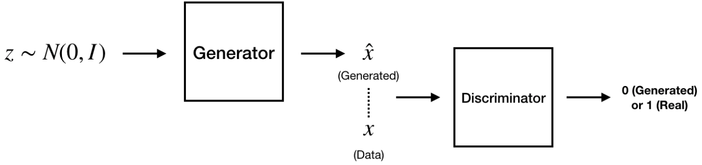


$$\min_G \max_D L(D, G) = \mathop{\mathbb{E}}_{x \sim p_r(x)} \log D(x) + \mathop{\mathbb{E}}_{z \sim p_z(z)} \log(1 - D(G(z)))$$

- GAN tries to find the Nash Equilibrium between two networks. At the equilibrium, $D(x) = \frac{1}{2}$
- If it is properly trained, the loss function is Jensen-Shannon divergence between two distributions

Generative Adversarial Network - GAN

- Now compare with another popular generative model - GAN by Goodfellow et al. [2014]



$$\min_G \max_D L(D, G) = \mathop{\mathbb{E}}_{x \sim p_r(x)} \log D(x) + \mathop{\mathbb{E}}_{z \sim p_z(z)} \log(1 - D(G(z)))$$

- GAN tries to find the Nash Equilibrium between two networks. At the equilibrium, $D(x) = \frac{1}{2}$
- If it is properly trained, the loss function is Jensen-Shannon divergence between two distributions

VAE and GAN

	VAE	GAN
Principles	Optimize Lower Bound	Find the Nash Equilibrium between two Networks (WGAN: find the optimal transport between two distributions)
Strength	See the latent encodings Based on theoretic foundations Training is stable	Generates high-quality & more diverse outputs
Improvements	Output tends to be blurrier on high-freq parts	Training is not stable More like a black-box approach

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4): 303–314, 1989.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *arXiv preprint arXiv:1802.05983*, 2018.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pages 4743–4751, 2016.

Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.

George Papamakarios, Iain Murray, and Theo Pavlakou. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pages 3483–3491, 2015.