Differentiable Generator Nets

Sargur N. Srihari
srihari@cedar.buffalo.edu

Topics

1. Differentiable generator nets

Differentiable Generator Nets

- Many generative nets are based on idea of:
  - Using a differentiable function $g(\boldsymbol{z}; \boldsymbol{\theta}^{(g)})$ represented by a neural net to transform samples of latent variables $\boldsymbol{z}$ to:
    - samples $\boldsymbol{x}$ or distributions over samples $\boldsymbol{x}$
- This model class includes
  1. Variational autoencoders (VAE)
     - Pair the generator network with an inference net
  2. Generative Adversarial networks (GAN)
     - Pair generator network with a discriminator network
  3. Techniques that train isolated generator networks

Ex: Samples from $N(\mu,\Sigma)$

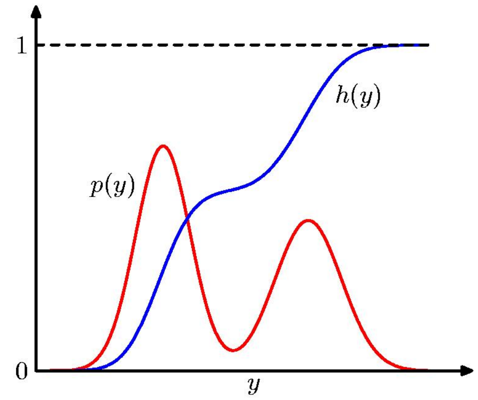
- Standard procedure for sampling from $N(\mu,\Sigma)$:
  - Feed samples $\boldsymbol{z}$ from $N(0, \mathrm{I})$ into a simple generator network that has just one affine layer:

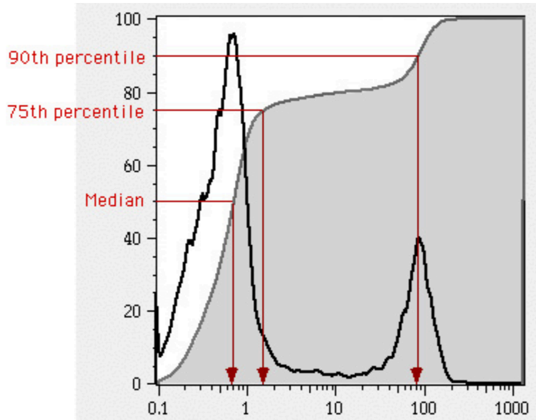
$$\boldsymbol{x} = g(\boldsymbol{z}) = \mu + L\boldsymbol{z}$$

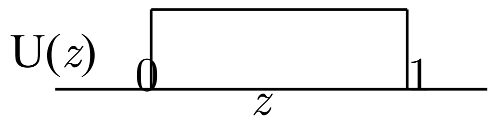
  - where L is the Cholesky decomposition of Σ
    - Decomposition: $\Sigma = LL^{\mathrm{T}}$ where L is lower triangular
    - Affine transform is a linear mapping method:
      - that preserves straight lines, planes, e.g., translation, rotation
  - This generator network takes $\boldsymbol{z}$ as input and produces $\boldsymbol{x}$ as output

Ex: Samples from any $p(y)$

- *Inverse transform sampling*
  1. If you are able to specify $p(y)$
     - then obtain its cdf $h(y)$
     - Which will have a value between 0 and 1
     - To get $h(y)$ requires indefinite integral $h(y)=\int_{-\infty}^{y} p(x)\,dx$
  2. Take input z from $\mathrm{U}(0,1)$
  3. Produce as output $h^{-1}(z)$ which is returned as the value of $g(z)$
     - Need ability to compute the inverse function!







Samples from complicated distributions

- For distributions that are complicated:
  - Difficult to specify directly, or
  - Difficult to integrate over, or
  - Resulting integrals are difficult to invert
- We use a feedforward network to represent a parametric family of nonlinear functions g and
  - use training data to infer the parameters selecting the desired function

Principle for mapping z to $\boldsymbol{x}$

- g provides a nonlinear change of variables
  - to transform distribution over z into desired distribution over $\boldsymbol{x}$
- The distributions of z and $\boldsymbol{x}$ is governed by

$$\boxed{p_z(\boldsymbol{z}) = p_x(g(\boldsymbol{z})) \left| \det\left(\frac{\partial g}{\partial \boldsymbol{z}}\right) \right|}$$

  - This implicitly imposes a distribution over $\boldsymbol{x}$

$$\boxed{p_x(\boldsymbol{x}) = \frac{p_z(g^{-1}(\boldsymbol{x}))}{\left| \det\left(\frac{\partial g}{\partial \boldsymbol{z}}\right) \right|}}$$

- The formula may be difficult to evaluate
  - So we often use indirect means of learning g
  - Rather than try to maximize $\log p(\boldsymbol{x})$ directly

Generating a conditional

- Instead of g providing a sample directly we use g to define a conditional distribution over $\boldsymbol{x}$
  - Example: use a generator net whose final layer are sigmoid outputs to provide mean parameters of Bernoullis: $p(\mathrm{x}_i = 1 \mid \boldsymbol{z}) = g(\boldsymbol{z})_i$
  - In this case when we use g to define $p(\boldsymbol{x}|z)$ we impose a distribution over $\boldsymbol{x}$ by marginalizing z:

$$p(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{z}} p(\boldsymbol{x} \mid \boldsymbol{z})$$

- Both approaches define a distribution $p_g(\boldsymbol{x})$ and allow us to train various criteria of $p_g(\boldsymbol{x})$ using the reparameterization trick

Comparison of two approaches

1. Emitting parameters of conditional distribution
   - Capable of generating discrete and continuous data
2. Directly generating a sample
   - Can only generate continuous data
     - We could introduce discretization in forward propagation, but we can no longer train using backpropagation
   - No longer forced to use simple conditional forms

- Approaches based on differentiable generator networks are motivated success of gradient descent in classification
  - Can this transfer to generative modeling?

Complexity of Generative Modeling

- Generative modeling is more complex than classification because
  - Learning requires optimizing intractable criteria
  - Because data does not specify both input z and output $\boldsymbol{x}$ of the generator network
- In classification both input and output are given
  - Optimization only needs to learn the mapping
- In generative modeling, learning procedure needs to determine how to arrange z space in a useful way and how to map z to $\boldsymbol{x}$