Application of Generative Adversarial Networks (GANs) in Smart Buildings

Zhe (Walter) Wang, Lawrence Berkeley Lab

Acknowledgement

- **I would like to thank Tianzhen Hong, Wanni Zhang, Xuan Luo, and other team members in Tianzhen's team**
- **I was inspired a lot from the following resources**
  - *Ian Goodfellow, Generative Adversarial Networks (NIPS 2016 tutorial)*
  - *UC Berkeley, 2017, Deep Learning Decall, Autoencoders and Representation Learning*
  - *Stanford University, 2020, Convolutional Neural Networks for Visual Recognition, lecture 13*
  - *University of Washington, 2017, A Compressed Overview of Sparsity*

Generative AI

- **AI generated faces**
  - https://generated.photos/faces/

- **AI generated music**
  - https://www.musi-co.com/listen/streams

Agenda

- Generative models
- Generative Adversarial Network (GAN)
- Application of GAN in smart building
- Discussion

Agenda

- **Generative models**
- Generative Adversarial Network (GAN)
- Application of GAN in smart building
- Discussion

Generative models

- **Definition**
  - Given training data, generate new samples from the same distribution
- **Motivation**
  - Generate data
    - For fun: artwork, music
    - For simulation/planning
  - Learn the hidden pattern of data in the latent space

Generative models

- **A major task of unsupervised learning**
  - Supervised: classification, regression
  - Unsupervised: clustering, dimension reduction
- **Evaluation**
  - Fidelity: generated samples should be indistinguishable from the real data
  - Diversity: generated samples should be distributed to cover the real data
  - Usefulness: generated samples should be just as useful as the real data

Generative models

- **Explicit density: the model explicitly define and solve the representation in the latent space**
- **Implicit density: the model can sample from representation in the latent space w/o explicitly defining it**

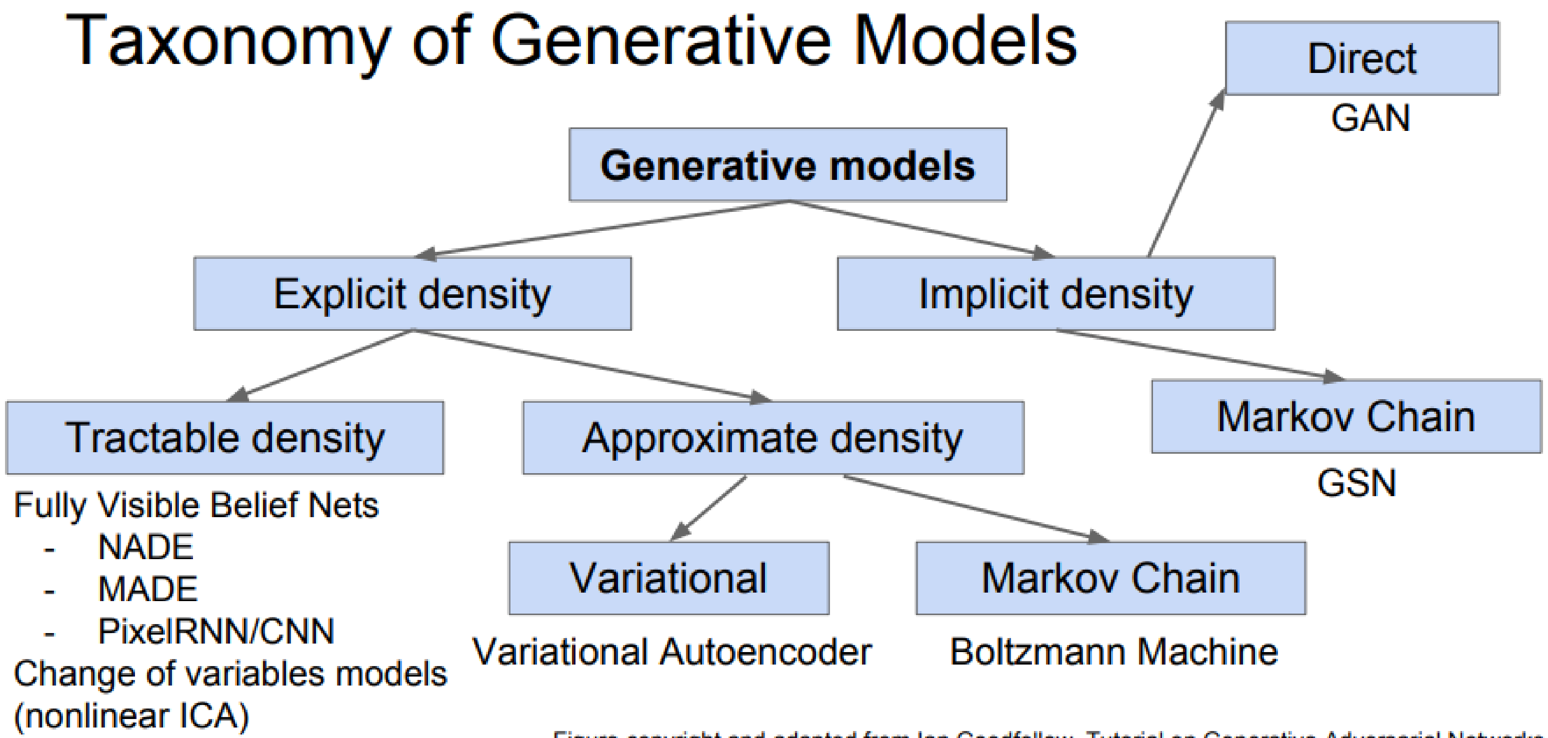
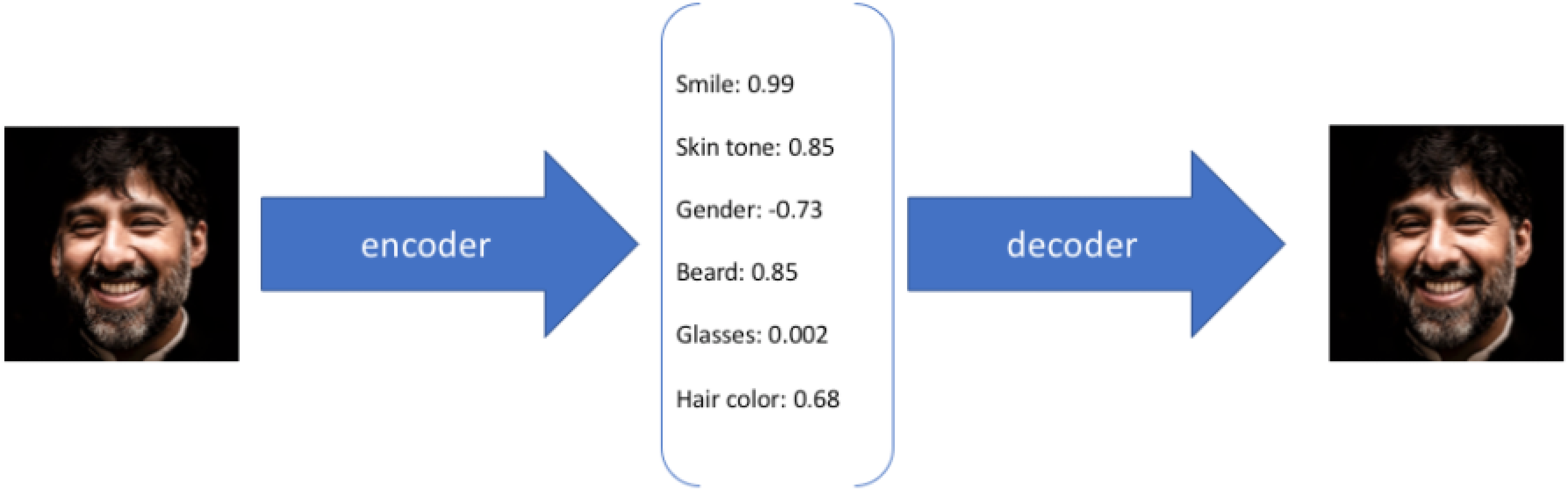


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Source: Ian Goodfellow, Generative Adversarial Networks (NIPS 2016 tutorial)

Compressed (latent) representation

- Encoding: develop a compressed representation (*latent space*) of the input data
- Decoding: generate new data from the sampled vectors in the latent space



Source: JEREMY JORDAN, Variational autoencoders, https://www.jeremyjordan.me/variational-autoencoders/

Agenda

- Generative models
- **Generative Adversarial Network (GAN)**
  - Idea
  - Math: objective function
  - Training
- Application of GAN in smart building
- Discussion

Generative Adversarial Network

- **Conventionally, generative models learn the latent representation explicitly**
  - PixelRNN, PixelCNN

$$\underbrace{p(x)}_{\text{Likelihood of image x}} = \prod_{i=1}^{n} \underbrace{p(x_i|x_1, ..., x_{i-1})}_{\text{Probability of i'th pixel value given all previous pixels}}$$

Source: Stanford University, 2020, Convolutional Neural Networks for Visual Recognition, lecture 13

  - Variational AutoEncoder (VAE)

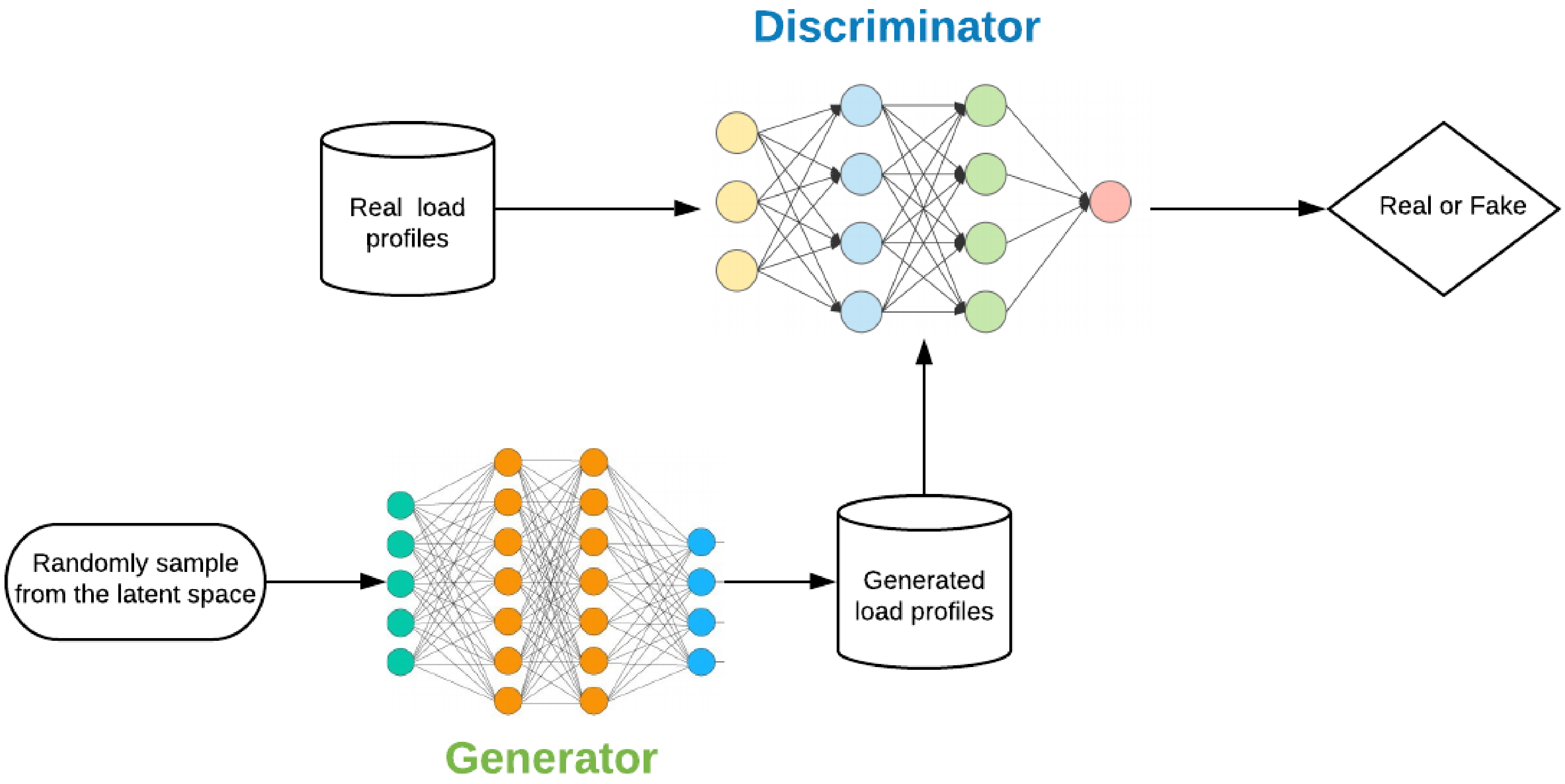
$$\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Bayes' Rule}) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})} \frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad (\text{Multiply by constant}) \\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Logarithms}) \\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))
\end{aligned}$$

Generative Adversarial Network

- **GAN**
  - Take game-theoretic approach, learn to generate from training data through 2-player game
    - Generator
    - Discriminator
- **History**
  - First proposed by Ian Goodfellow in 2014
  - Quickly becomes a hot topic
  - 2017: the Year of GAN

Generative Adversarial Network

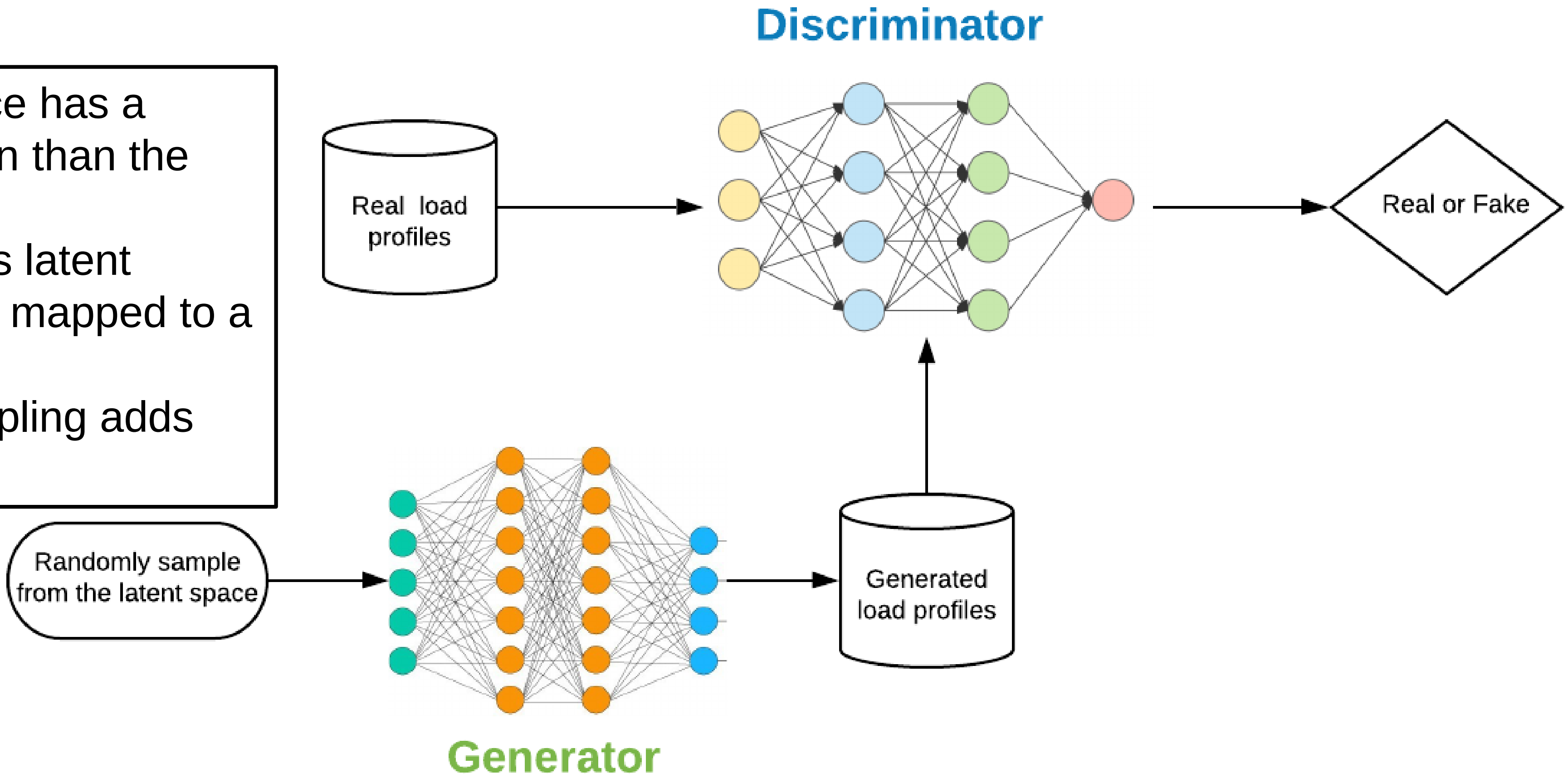
- **Generator: try to fool the discriminator by generating real-looking data**
- **Discriminator: distinguish between real and fake (generated) data**



Generative Adversarial Network

- **Generator: try to fool the discriminator by generating real-looking data**
- **Discriminator: distinguish between real and fake (generated) data**

- The latent space has a lower dimension than the original space
- Any point in this latent space could be mapped to a valid data point
- Randomly sampling adds stochasticity



Generative Adversarial Network

- **Generator: try to fool the discriminator by generating real-looking data**
- **Discriminator: distinguish between real and fake (generated) data**
- **Objective function**
  - Discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x\sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z\sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

  - Generator

$$\min_{\theta_g} \mathbb{E}_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Source: Stanford University, 2020, Convolutional Neural Networks for Visual Recognition, lecture 13

Training GAN

- **2-player game**

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

- **Using Back-Propagation algorithm**
  - Fix the discriminator when training generator
  - Fix the generator when training discriminator

Training GANs

- **Putting it all together**

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Source: Stanford University, 2020, Convolutional Neural Networks for Visual Recognition, lecture 13

Training GANs is a challenge

- Tips and tricks for training GANs
  - https://github.com/soumith/ganhacks

1. Normalize the inputs

- normalize the images between -1 and 1
- Tanh as the last layer of the generator output

2: A modified loss function

In GAN papers, the loss function to optimize G is `min (log 1-D)`, but in practice folks practically use `max log D`

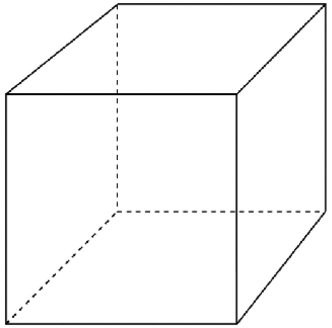
- because the first formulation has vanishing gradients early on
- Goodfellow et. al (2014)

In practice, works well:

- Flip labels when training generator: real = fake, fake = real

3: Use a spherical Z

- Dont sample from a Uniform distribution



- Sample from a gaussian distribution

5: Avoid Sparse Gradients: ReLU, MaxPool

- the stability of the GAN game suffers if you have sparse gradients
- LeakyReLU = good (in both G and D)
- For Downsampling, use: Average Pooling, Conv2d + stride
- For Upsampling, use: PixelShuffle, ConvTranspose2d + stride
  - PixelShuffle: https://arxiv.org/abs/1609.05158

6: Use Soft and Noisy Labels

- Label Smoothing, i.e. if you have two target labels: Real=1 and Fake=0, then for each incoming sample, if it is real, then replace the label with a random number between 0.7 and 1.2, and if it is a fake sample, replace it with 0.0 and 0.3 (for example).
  - Salimans et. al. 2016
- make the labels the noisy for the discriminator: occasionally flip the labels when training the discriminator

7: DCGAN / Hybrid Models

- Use DCGAN when you can. It works!
- if you cant use DCGANs and no model is stable, use a hybrid model : KL + GAN or VAE + GAN

8: Use stability tricks from RL

- Experience Replay
  - Keep a replay buffer of past generations and occassionally show them
  - Keep checkpoints from the past of G and D and occassionaly swap them out for a few iterations
- All stability tricks that work for deep deterministic policy gradients
- See Pfau & Vinyals (2016)

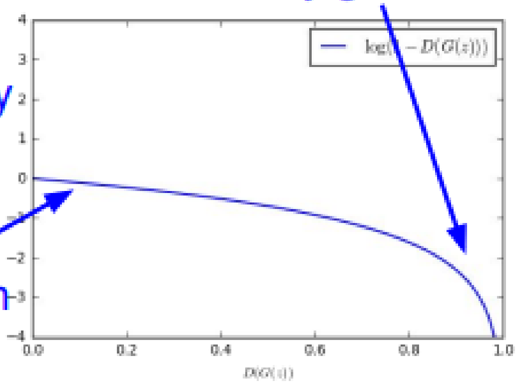
A modified loss function

- **Loss function of generator**

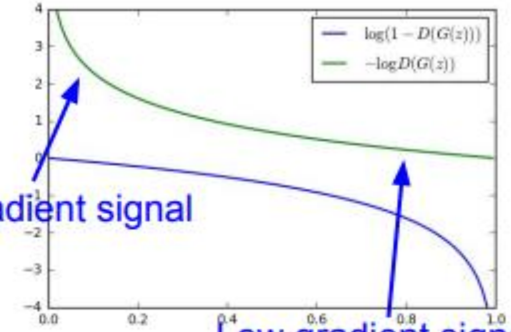
$$\min_{\theta_g} \mathbb{E}_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Gradient signal dominated by region where sample is already good

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



$$\max_{\theta_g} \mathbb{E}_{z\sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$



High gradient signal

Low gradient signal

Source: Stanford University, 2020, Convolutional Neural Networks for Visual Recognition, lecture 13

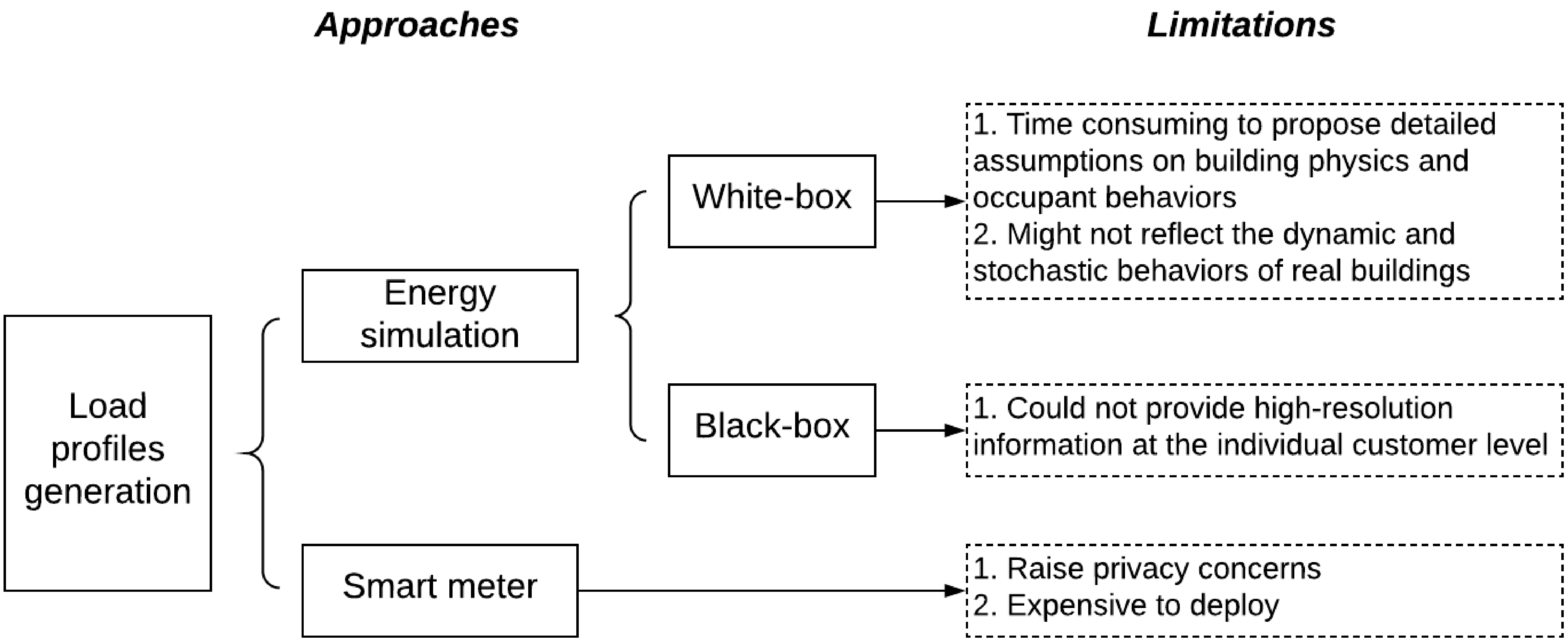
Agenda

- Generative models
- Generative Adversarial Network (GAN)
- **Application of GAN in smart building**
- Discussion

Application of GAN in smart building

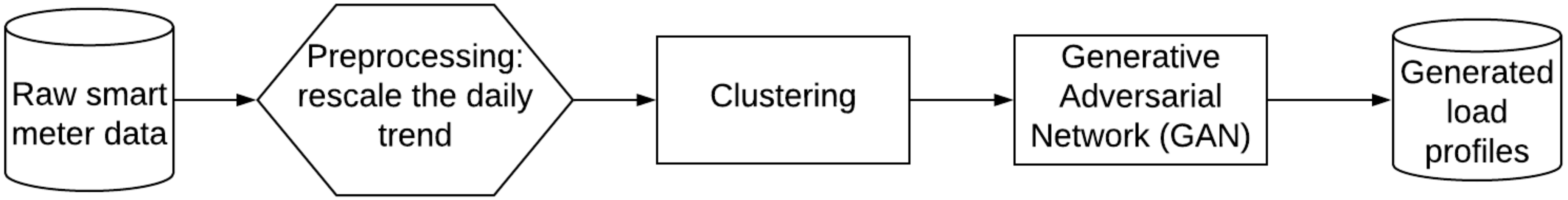
- **Using GAN to generated building load profiles**
  - Published as: Wang, Z. and Hong, T., 2020. Generating realistic building electrical load profiles through the Generative Adversarial Network (GAN). Energy and Buildings, 224, p.110299.
- **Why we need to generate building load profiles**
  - Wide application in the grid operation
    - Identification of unnecessary waste
    - Load forecasting for generation planning
    - …

Building load generation



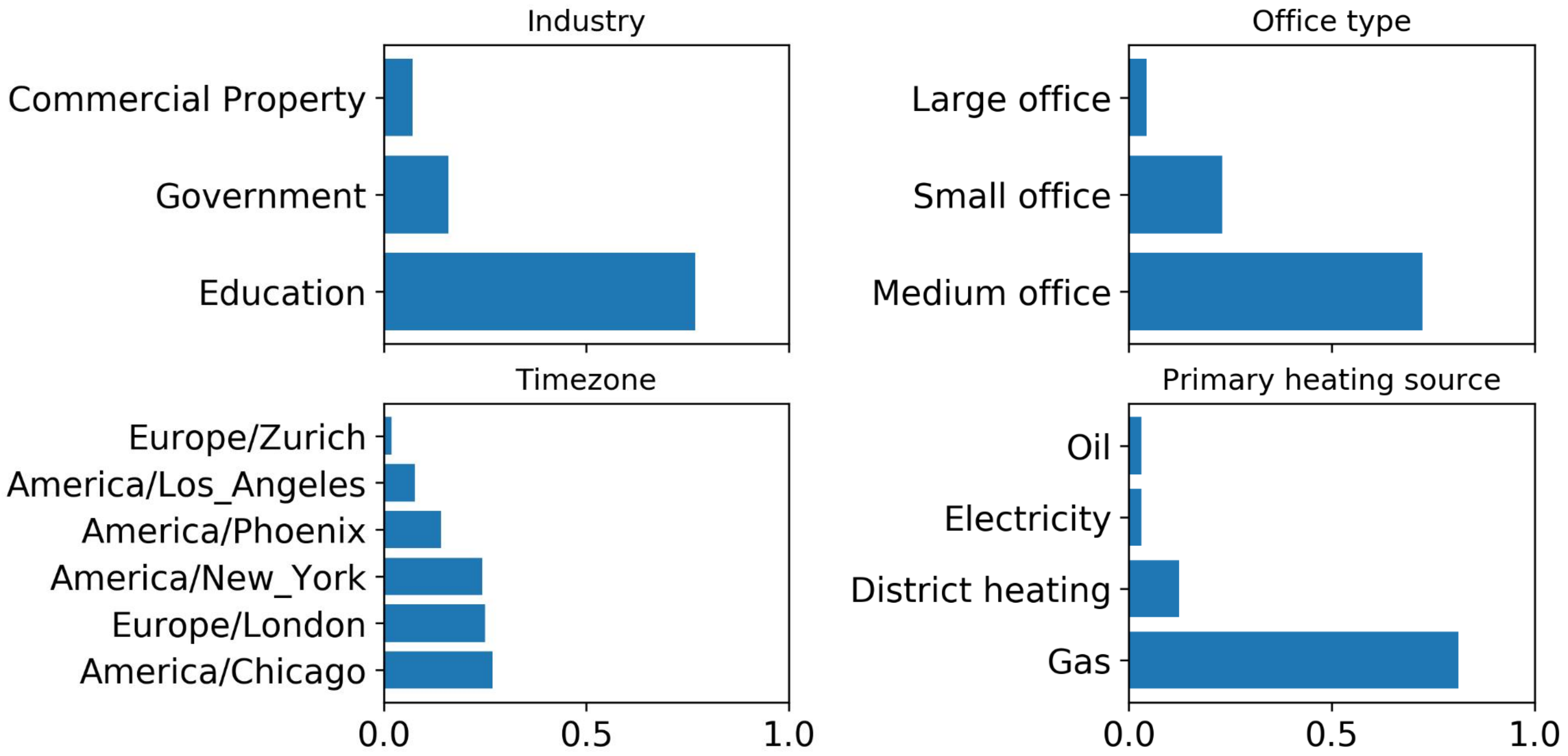
Research question

- Can we generate building load directly from smart meter data?
- Yes, we can!



Data

- **Building Data Genome Project database**
  - 156 office buildings
  - 56,957 daily loads
  - Electrical
  - Hourly



BERKELEY LAB

Clustering

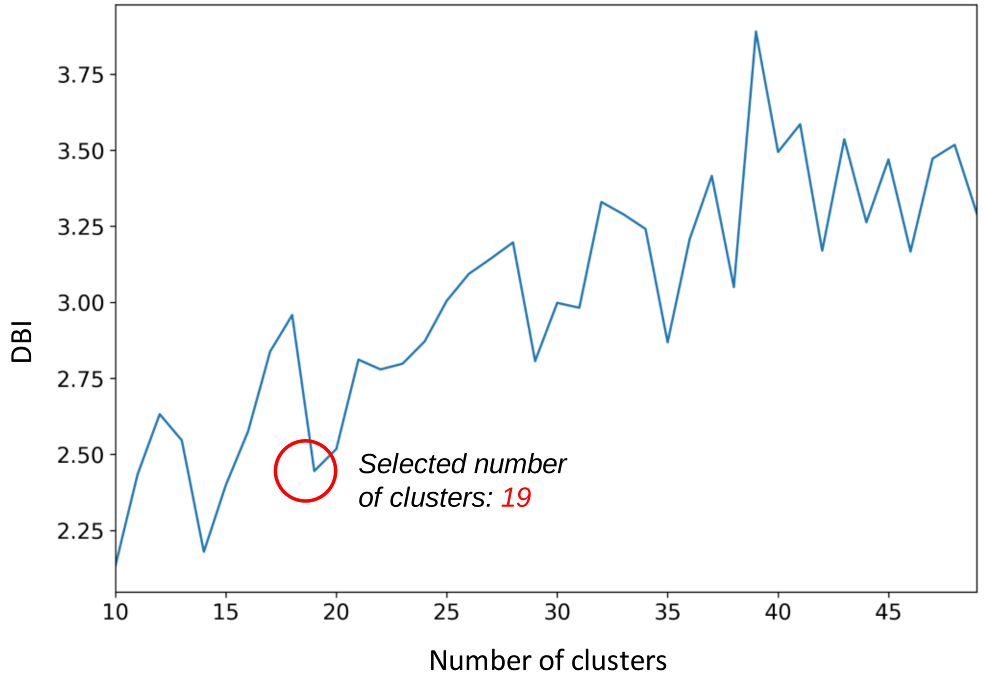
- **Why we need clustering**
  - Same cluster of load share similar patterns
  - GAN is learning these patters
  - If you combine different clusters together, the pattern is blurry and hard to learn
- **Metrics to evaluate clustering**
  - Davies-Bouldin Index (DBI)

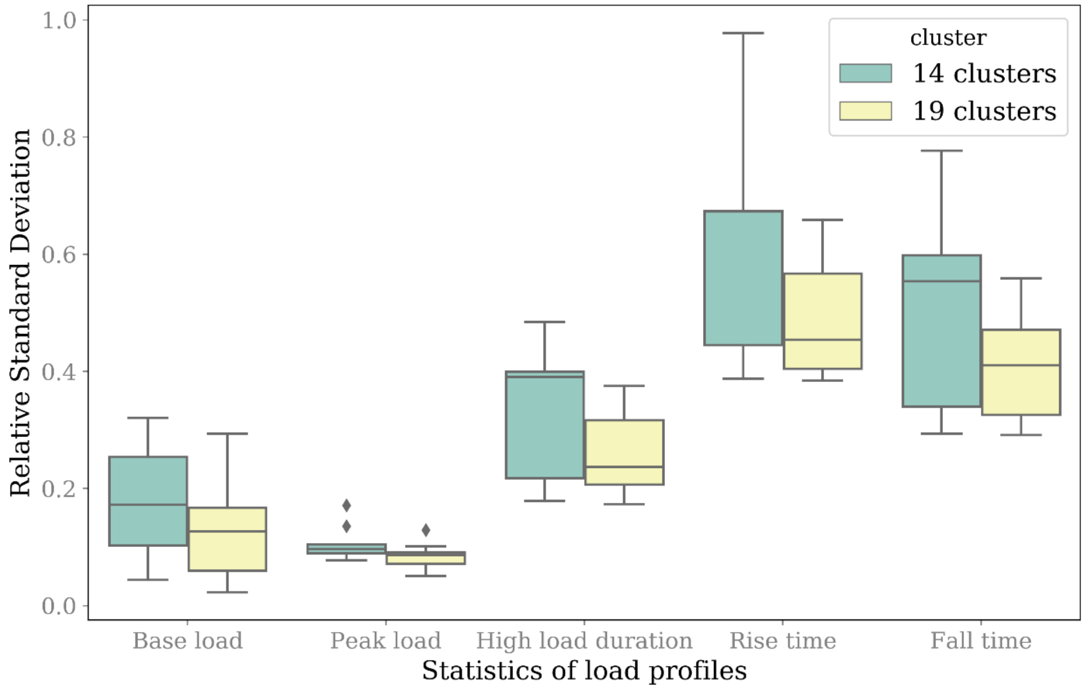
$$R_{ij} = \frac{s_i + s_j}{d_{ij}}$$

$$DBI = \frac{1}{k}\sum_{i=1}^{k} max_{i \neq j}(R_{ij})$$

Clustering: method

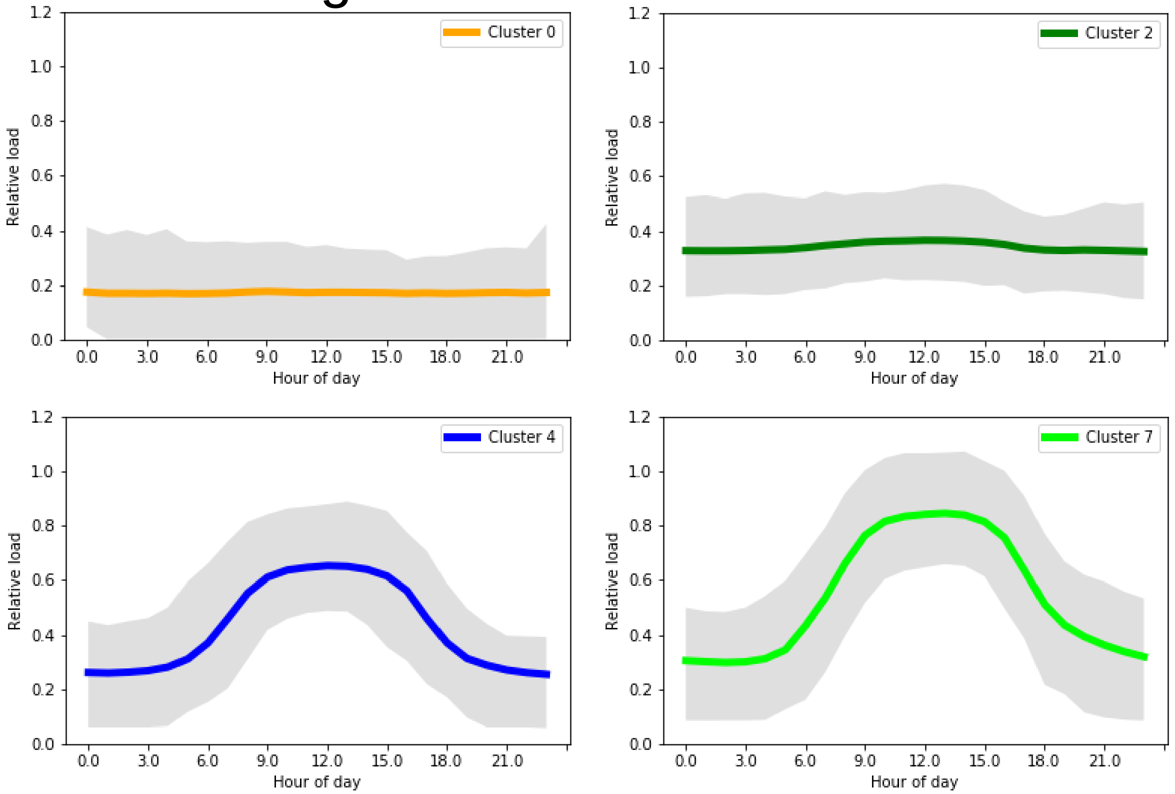
- **K-means**
- **Select the number of clusters**

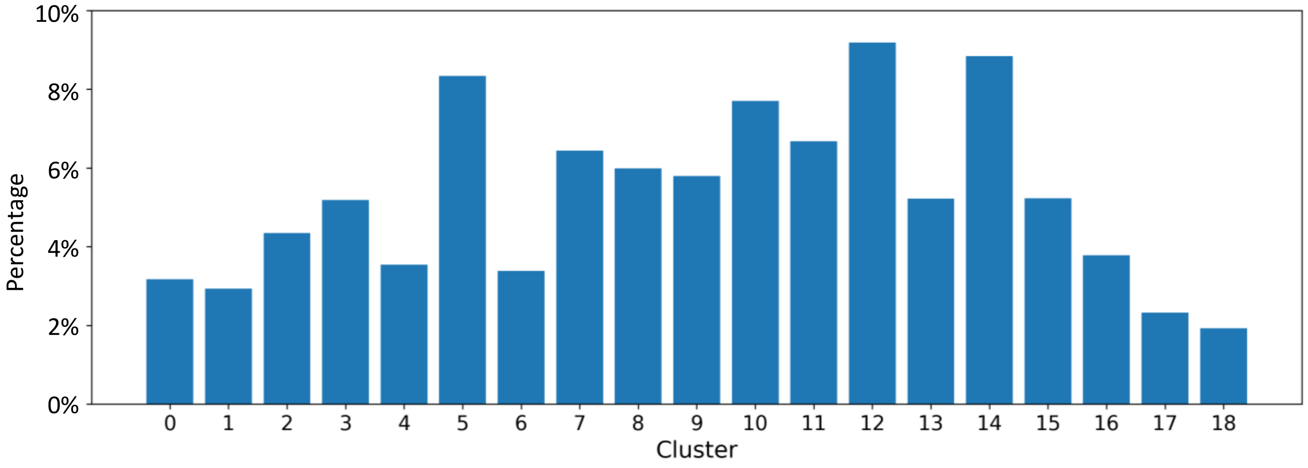




Clustering: result

- **We identified 19 clusters**
  - Working and non-working day patterns
  - High and low base-load





BERKELEY LAB

GAN: method

- **We implemented GAN with Keras**

Pseudocode

---

define the *discriminator neural network* and compile the *discriminator model*

define the *generator neural network*

define and compile *GAN model* by integrating the *generator* and *discriminator neural network*, and setting the parameter of *discriminator neural network* untrainable

for *epoch* in range(*epochs*):

 # train the discriminator

sample points randomly from the real load profile dataset

generate fake load profiles from randomly sample seeds with the *generator neural network*

combine and shuffle the real and fake load profiles together

train the *discriminator model* with the combined data points to minimize d_loss defined in *Equation 3* (training the parameters in the *discriminator neural network*)

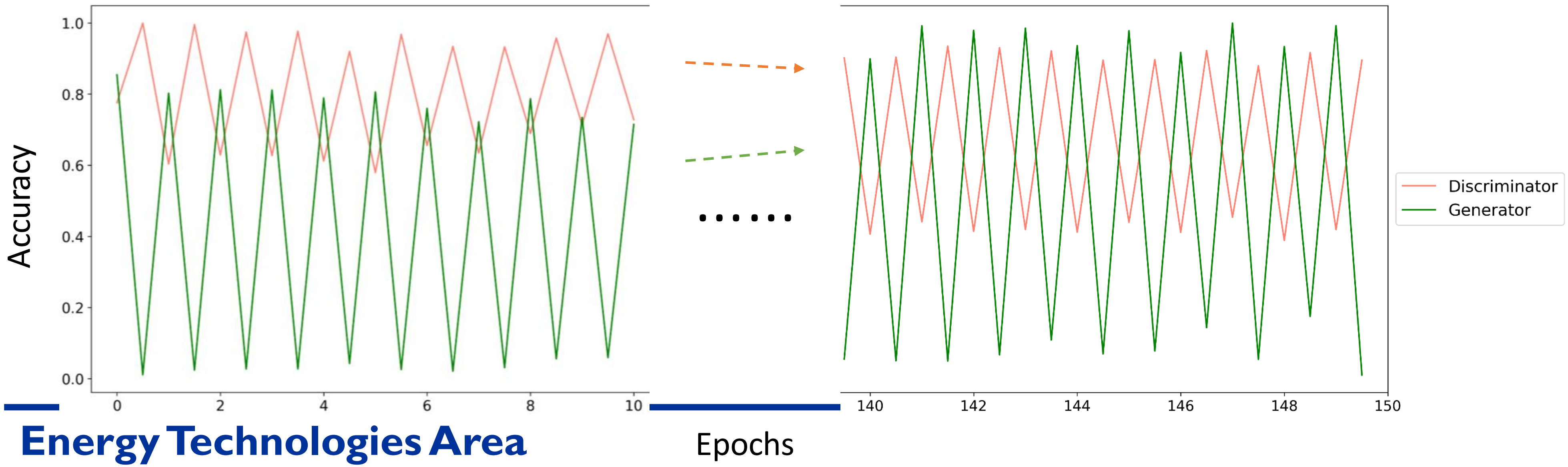
 # train the generator

sample seeds randomly from pre-defined normal distribution

train the *GAN model* with the sampled seeds to minimize g_loss defined in *Equation 4* (as the parameter of *discriminator* was set untrainable in the *GAN model*, we are essentially training the parameters in *generator neural network* only in this phase)

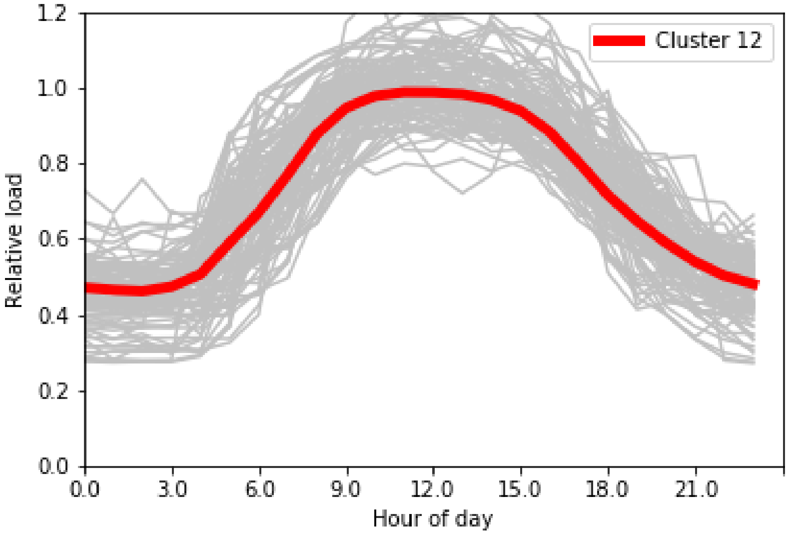
GAN: training

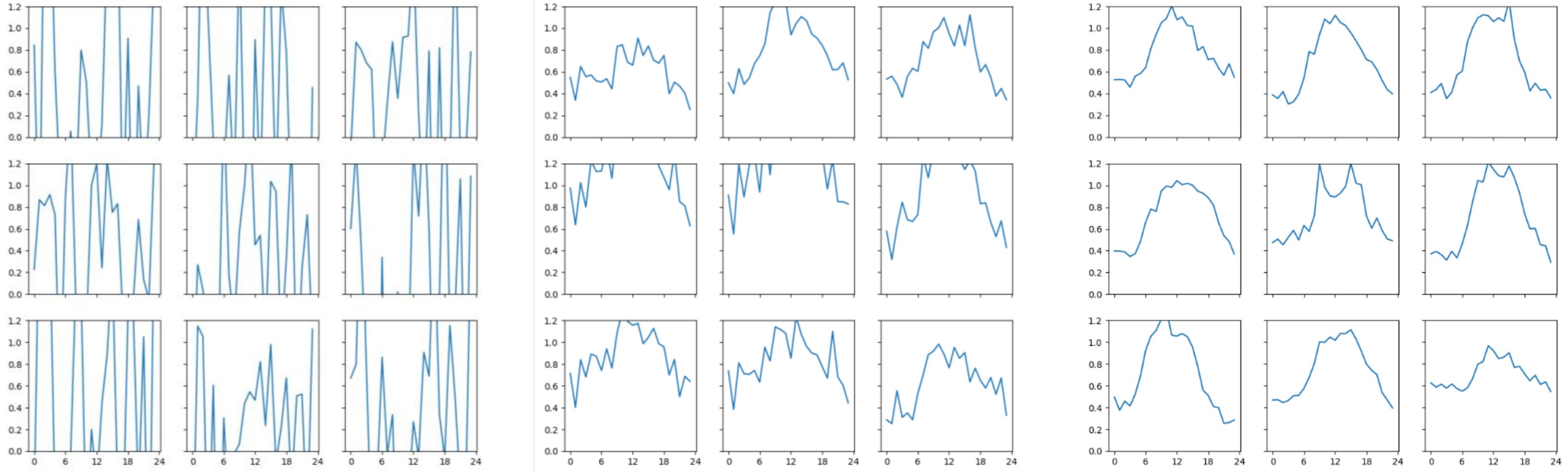
- **Discriminator**
  - the percentage of load profiles that can be detected correctly.
- **Generator**
  - the percentage of generated load profiles that are detected as “real” by the discriminator



GAN: result

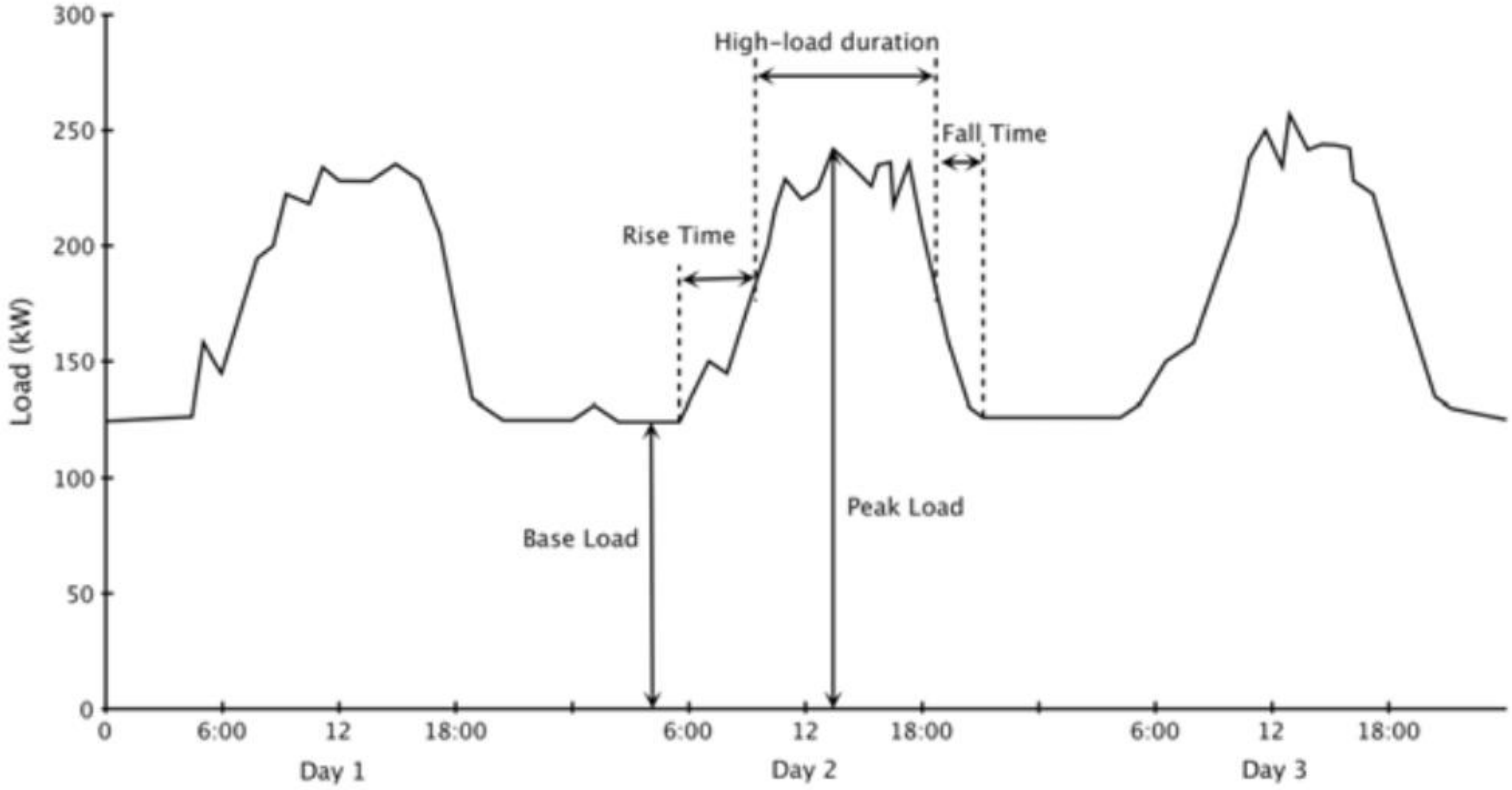
- **Learn to capture the load dynamics**
  - General trend
  - Random events



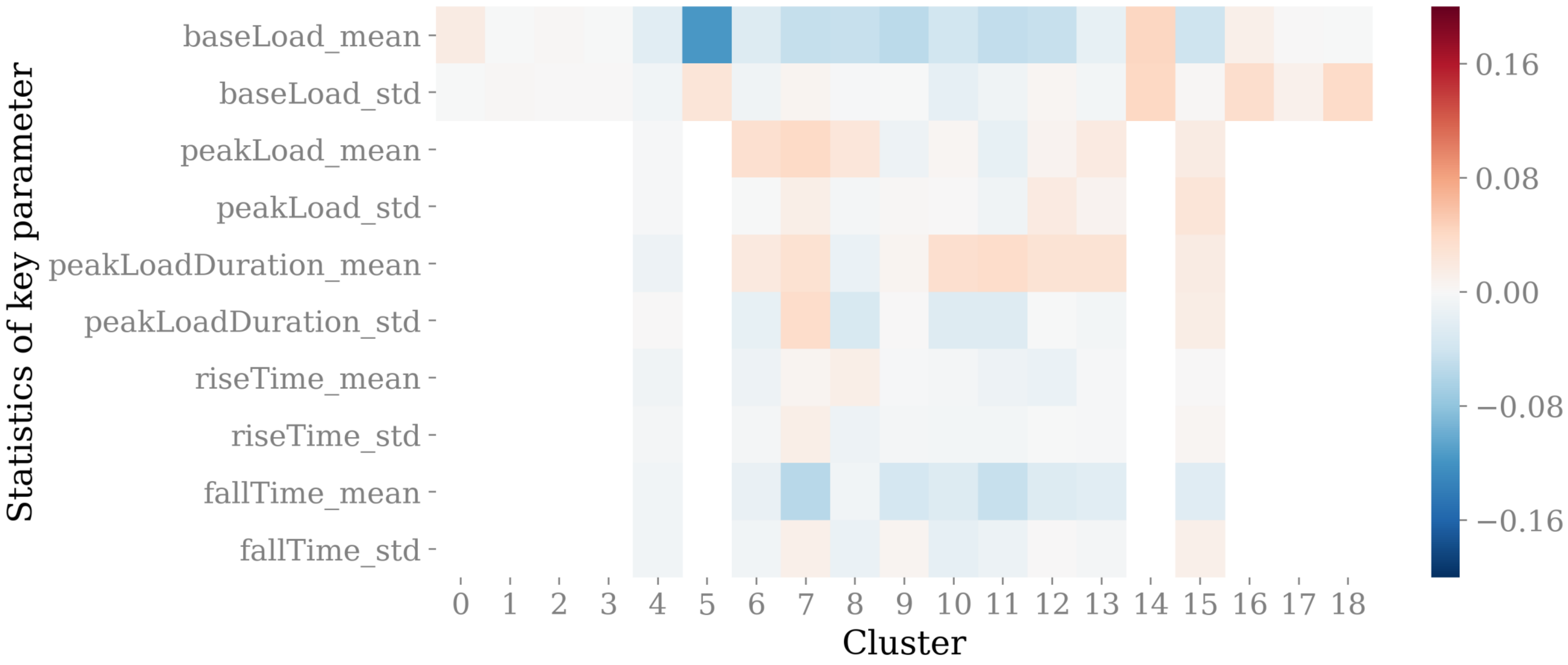


GAN: validation

- Diversity
- **Fidelity**
- Usefulness



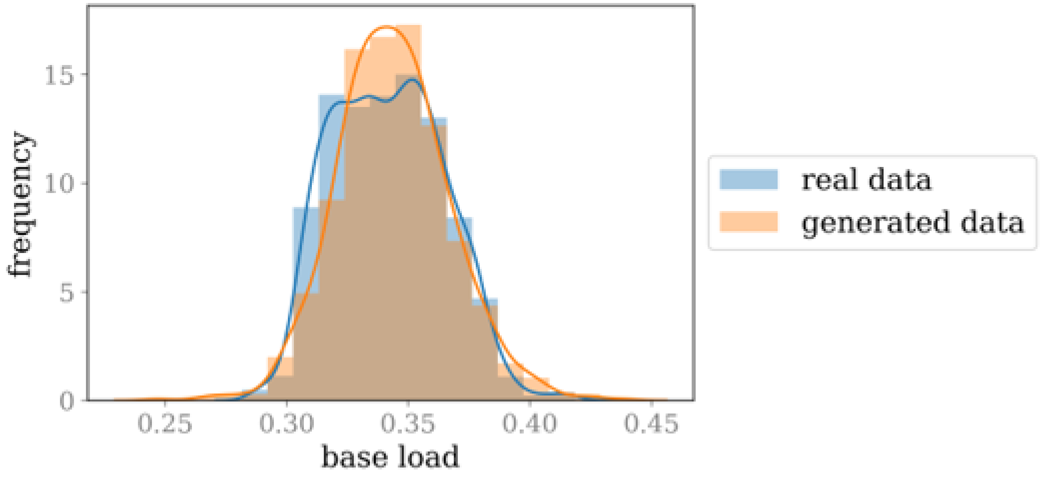
GAN: validation



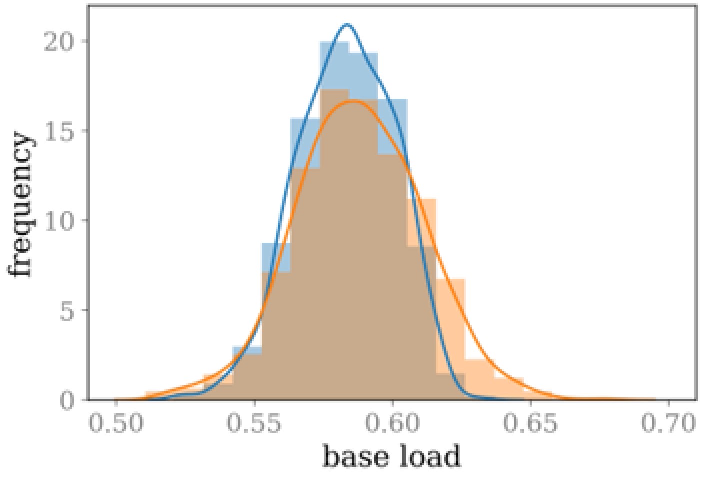
GAN: validation

- **Distance between distributions**
- **Kullback–Leibler Divergence**

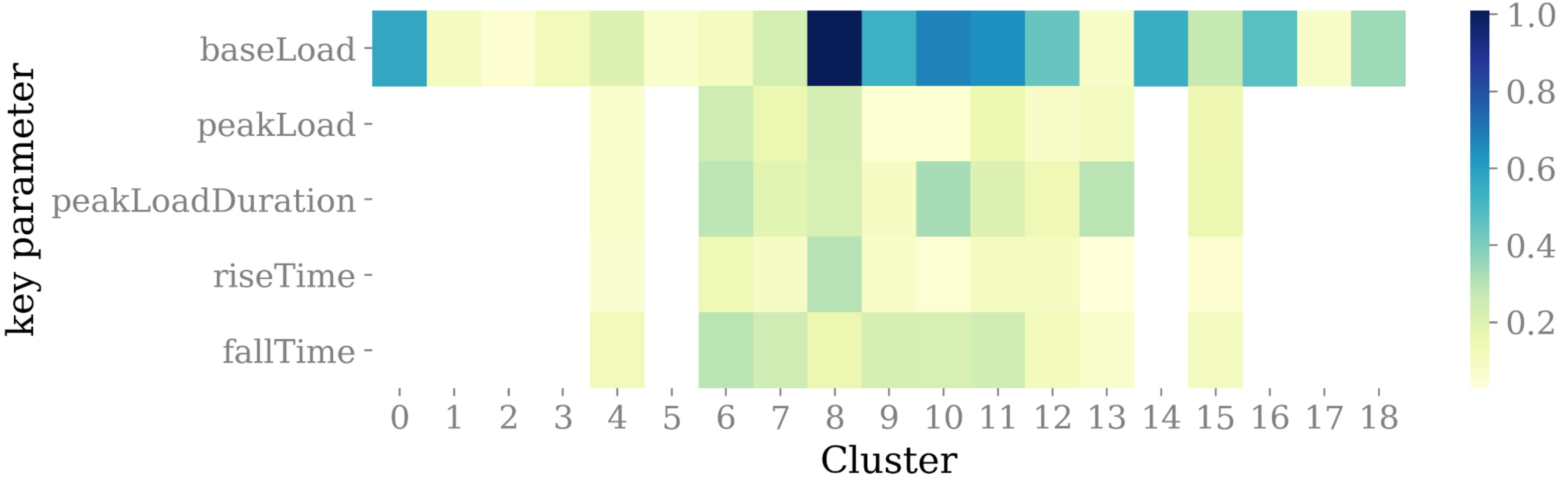
$$D_{\mathrm{KL}}(P \parallel Q) = -\sum_{x \in \mathcal{X}} P(x) \log\left(\frac{Q(x)}{P(x)}\right)$$



(c) cluster 2 (low KL divergence)

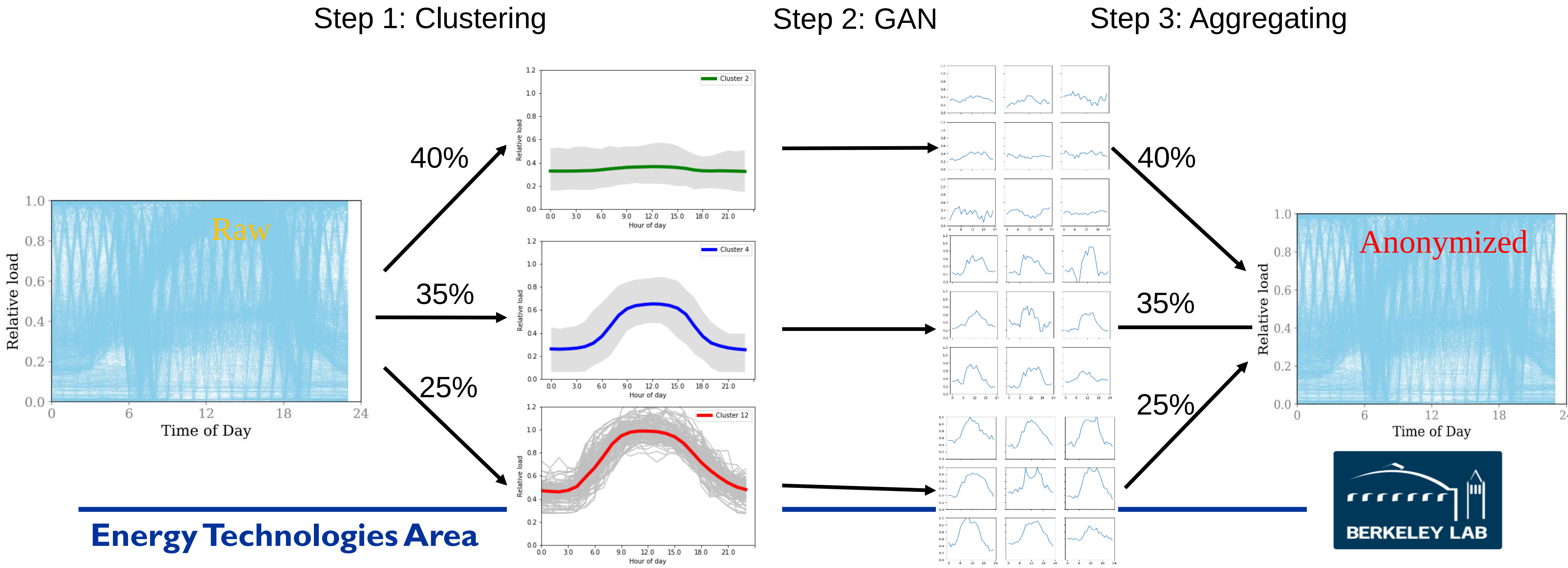


(d) cluster 8 (high KL divergence)



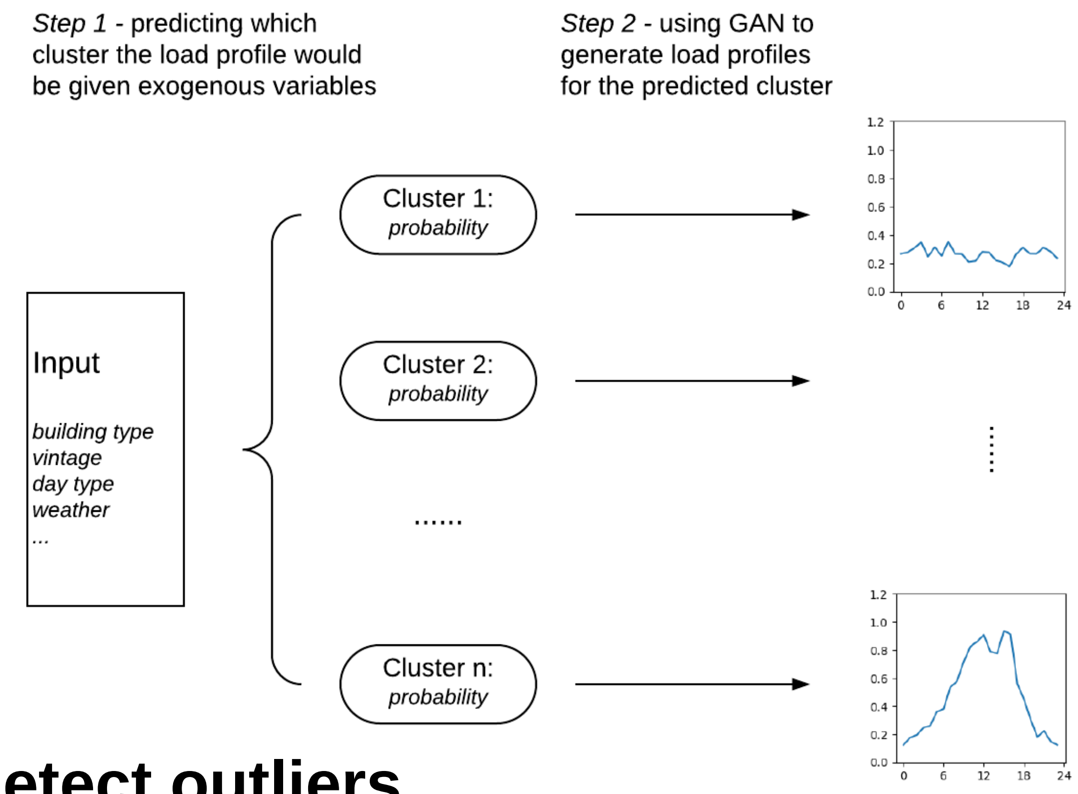
GAN: Applications

- **Anonymize real building electrical load profiles**



GAN: Applications

- **Load prediction**



- **Validate load models or detect outliers**

Agenda

- Generative models
- Generative Adversarial Network (GAN)
- Application of GAN in smart building
- Discussion

Enhancement to GANs

- Wasserstein GAN
- Convolutional GAN
- LSGAN
- Time series GAN

Some enhancement for time-series data

- **Need to capture**
  - the distribution of features within each time point

$$\mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}\sim p}\left[\log y_{\mathcal{S}} + \textstyle\sum_t \log y_t\right] + \mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}\sim \hat{p}}\left[\log(1-\hat{y}_{\mathcal{S}}) + \textstyle\sum_t \log(1-\hat{y}_t)\right]$$

  - the dynamics of those variables across time

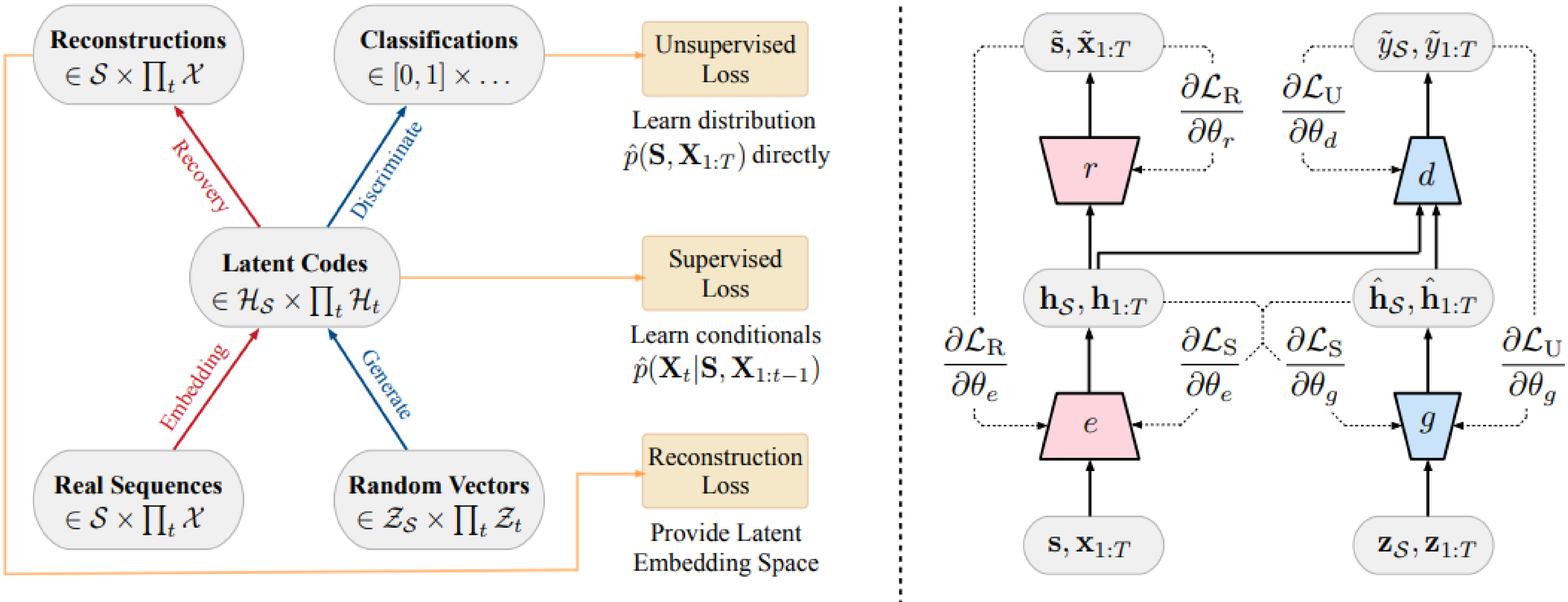
$$\textstyle\prod_t p(\mathbf{x}_t|\mathbf{x}_{1:t-1})$$

Solution I

- **Use recurrent neural network to capture the temporal dynamics**
  - Long Short Term Memory
    - Mogren, O., 2016. C-RNN-GAN: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*.
  - Recurrent Neural Network
    - Esteban, C., Hyland, S.L. and Rätsch, G., 2017. Real-valued (medical) time series generation with recurrent conditional gans. arXiv preprint arXiv:1706.02633.

Solution 2

- **TimeGAN**
  - Yoon, J., Jarrett, D. and van der Schaar, M., 2019. Time-series generative adversarial networks. In Advances in Neural Information Processing Systems (pp. 5508-5518)
- **Key idea: Incorporate the temporal dynamics into the objective functions**



(a) Block Diagram

(b) Training Scheme

$$\mathcal{L}_\mathrm{U} = \mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}\sim p}\big[\log y_\mathcal{S} + \textstyle\sum_t \log y_t\big] + \mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}\sim \hat{p}}\big[\log(1-\hat{y}_\mathcal{S}) + \textstyle\sum_t \log(1-\hat{y}_t)\big]$$

$$\mathcal{L}_\mathrm{S} = \mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}\sim p}\big[\textstyle\sum_t \|\mathbf{h}_t - g_\mathcal{X}(\mathbf{h}_\mathcal{S}, \mathbf{h}_{t-1}, \mathbf{z}_t)\|_2\big]$$

$$\mathcal{L}_\mathrm{R} = \mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}\sim p}\big[\|\mathbf{s} - \tilde{\mathbf{s}}\|_2 + \textstyle\sum_t \|\mathbf{x}_t - \tilde{\mathbf{x}}_t\|_2\big]$$

TimeGAN: contribution/novelty

- **Use a supervised loss to better capture temporal dynamics**
- **Use an embedding network that provides a lower-dimensional adversarial learning space**

- **https://github.com/jsyoon0823/TimeGAN**

Conclusion

- We introduced generative models and latent space
- We learned Generative Adversarial Networks (GAN): math and psudo-code
- We applied GAN to generate building load profiles
- We discussed future work and enhancements to GAN

Thanks and Questions