FSAN/ELEG815: Statistical Learning

Gonzalo R. Arce

Department of Electrical and Computer Engineering
University of Delaware

X: Deep Generative Model

Which Face is Real?

a

b

c

Supervised vs Unsupervised Learning?

Supervised Learning

Data:(x,y)
x is data, **y** is label
Goal: Learn a function to map

$$\mathbf{x} \to \mathbf{y}$$

Examples: Classification, regression, object detection, semantic segmentation...

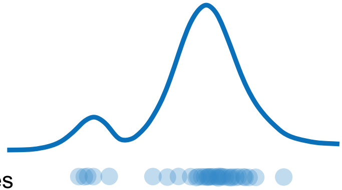
Unsupervised Learning

Data: x
x is data, no labels!
Goal: Learn the *hidden or underlying* structure of the data
Examples: Clustering, feature or dimensionality reduction...

Generative Modeling

Goal: Take as input training samples from some distribution and learn a model that represents that distribution

Density Estimation



Samples

Sample Generation

Input samples

Training data $\sim P_{data}(\mathbf{x})$

Generated samples

Generated $\sim P_{model}(\mathbf{x})$

Why Generative Models? Debiasing

Capable of uncovering **underlying features** in a dataset

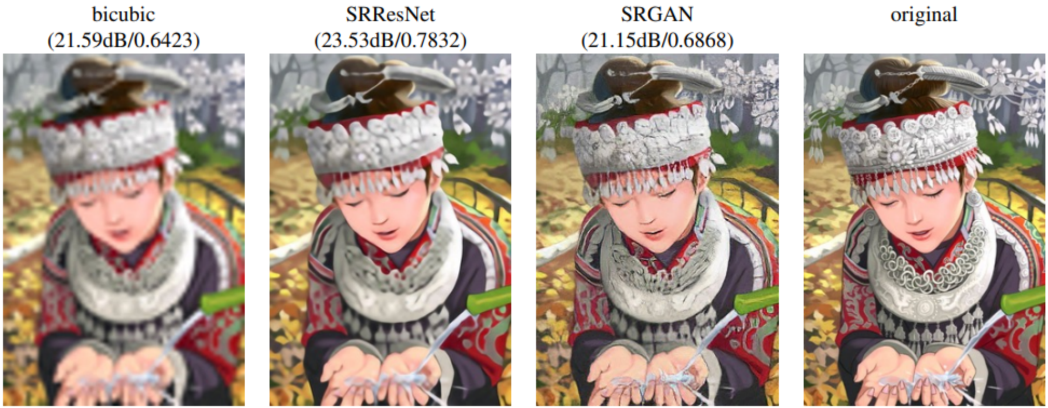
Homogeneous skin color, pose

vs

Diverse skin color, pose, illumination

How can we use this information to create fair and representative datasets?

Why Generative Models? Super resolution



Bicubic interpolation

Deep residual network optimized for MSE

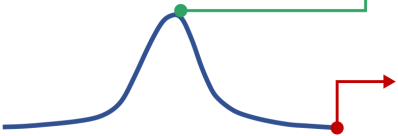
Deep residual generative adversarial network

Original HR image

How can we detect something new or rare?

- **Problem:** How canwe detect when we encounter something new or rare?
- **Stategy:** Leverage generative models, detect outliers in the distribution
- Use outliers during training to improve even more!

95% of Driving Data:
(1) sunny, (2) highway, (3) straight road



Detect outliers to avoid unpredictable behavior when training

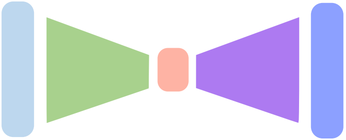
Edge Cases

Harsh Weather

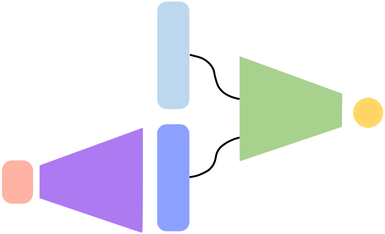
Pedestrians

Latent Variable Models

Autoencoders and Variational Autoencoders (VAEs)

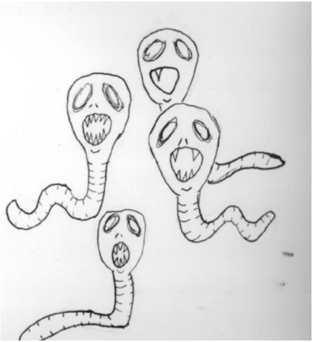


Generative Adversarial Networks (GANs)

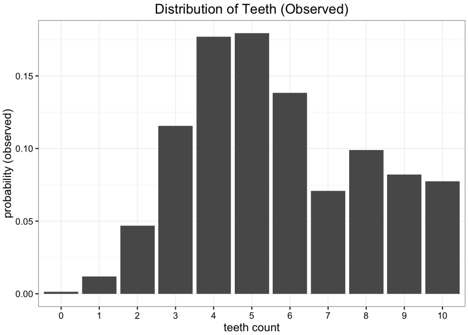


Kullback-Leibler Divergence

- A way of comparing two probability distributions.
- Measures how well a simple distribution function approximates a complex one

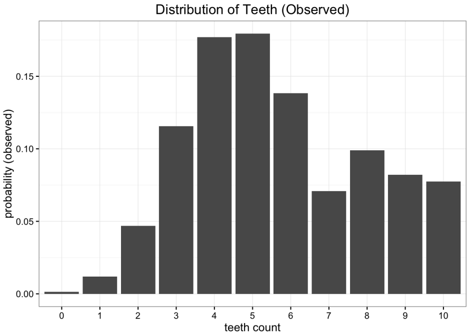


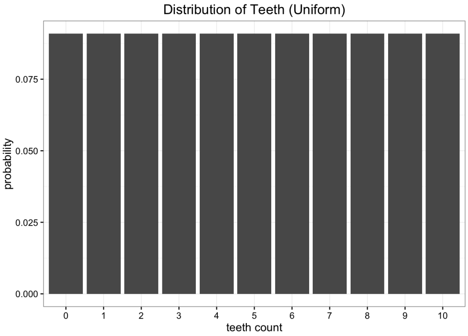
Space worms and KL divergence



Kullback-Leibler Divergence (Uniform Distribution)

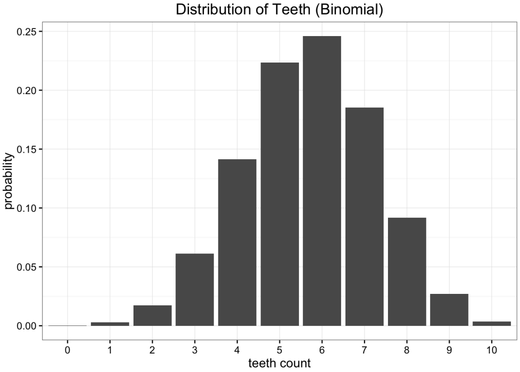
- There are 11 possible values and we approximate with a uniform distribution.





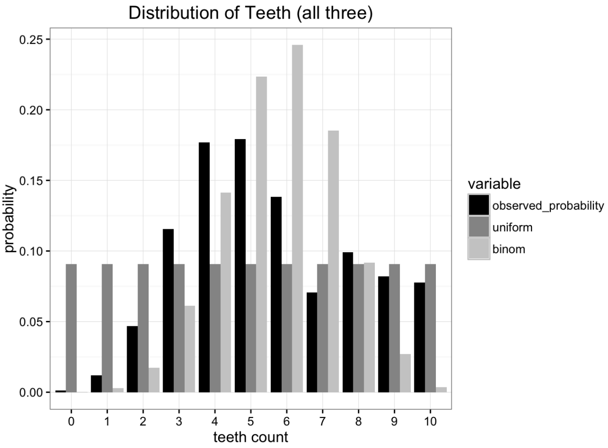
Kullback-Leibler Divergence (Binomial Distribution)

- Represent distribution of teeth in worms as just a Binomial distribution.
- Estimate the probability parameter of the Binomial distribution.
- $E[x] = n \times p$ where $n = 10$ and $E[x] = 5.7$, thus $p = 0.57$.



Binomial Distribution vs Uniform Distribution

- Compared with the original data, both are approximations.
- How can we choose which one to use?



The Entropy of a Distribution

The entropy for a probability distribution is:

$$H = -\sum_{i=1}^{N} p(x_i) \times \log(p(x_i))$$

- If we use $\log_2$ we can interpret entropy as "the minimum number of bits it would take us to encode our information".
- Our probability distribution has an entropy of 3.12 bits which is the lower bound for how many bits are needed to encode the number of teeth of a sample.

Measuring Information Lost Using Kullback-Leiber Divergence

Kullback-Leiber Divergence is just a modification of entropy:

$$D_{KL}(p \parallel q) = \sum_{i=1}^{N} p(x_i) \times (\log(p(x_i)) - \log(q(x_i)))$$

Expectation of the log difference between the probability of data in the original distribution with the approximating distribution. We could rewrite it as:

$$D_{KL}(p \parallel q) = E[\log(p(x)) - \log(q(x))]$$

$$D_{KL}(p \parallel q) = \sum_{i=1}^{N} p(x_i) \times (\log(\frac{p(x_i)}{q(x_i)}))$$

Comparing our approximating distributions

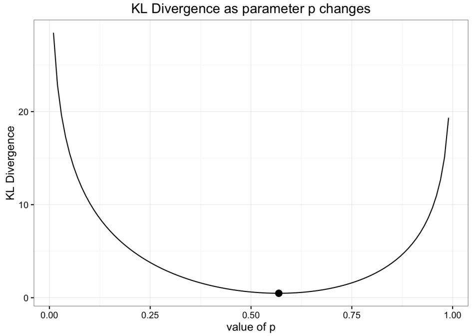
KL divergence for our two approximating distributions:

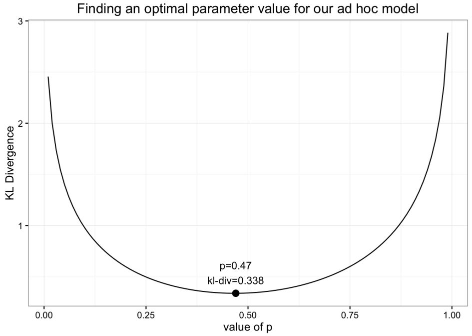
$$D_{KL}(Observe \,\|\, Uniform) = 0.338$$
$$D_{KL}(Observe \,\|\, Binomial) = 0.477$$

- The information lost by using the binomial approximation is greater than using the uniform approximation.
- Note that the KL divergence is not a distance metric, since it is not symetric i.e:

$$D_{KL}(Binomial \,\|\, Observe) = 0.330$$

Optimizing Using KL Divergence





- The minimum value for KL divergence is 0.338 when $p = 0.47$

Optimizing Using KL Divergence

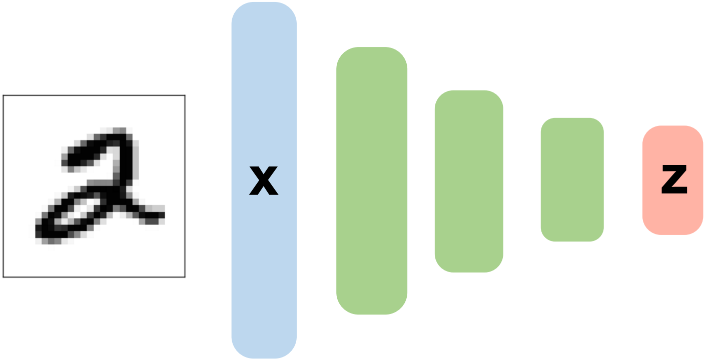
- Key point is to use KL Divergence as an objective function to find the optimal parameters for any approximating distribution.
- Extend this approach to high dimensional models with many parameters.
- Neural networks are function approximators.
- Combining KL divergence with neural networks learn complex approximating distributions for data ("Variational Autoencoder")

What is a Latent Variable

Can we learn the **true explanatory factors**, e.g. latent variables, from only observed data?

Autoencoders: background

Unsupervised approach for learning a **lower-dimensionality** feature representation from unlabeled training data



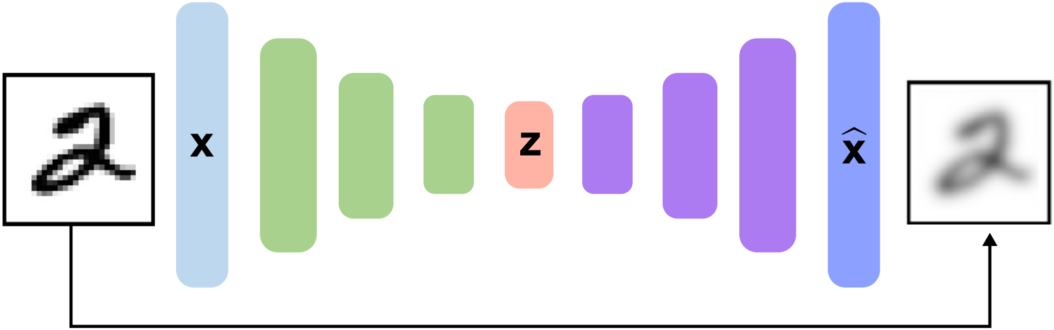
Why do we care about a low-dimensional **z**

“Encoder" learns mapping from the data, $\mathbf{x}$, to a low-dimensional latent space $\mathbf{z}$

Autoencoders: background

How can we learn this latent space?

Train the model to use these features to **reconstruct the original data**



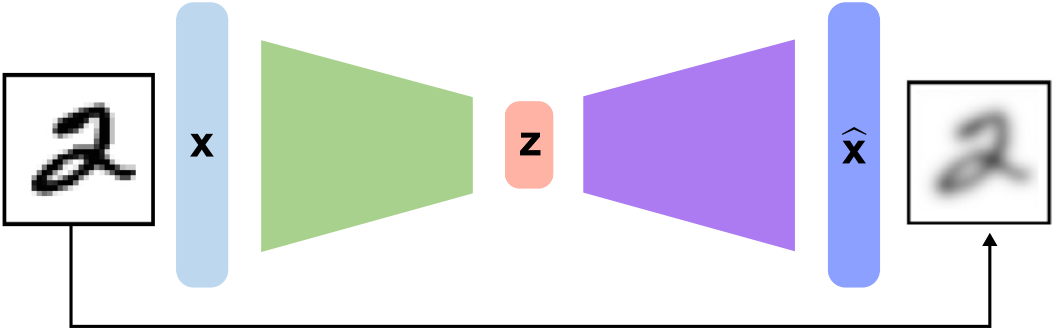
$$\mathcal{L} = \| \mathbf{x} - \widehat{\mathbf{x}} \|^2$$

Loss function doesn't use any labels!

Autoencoders: background

How can we learn this latent space?

Train the model to use these features to **reconstruct the original data**



$$\mathcal{L} = \| \mathbf{x} - \hat{\mathbf{x}} \|^2$$

Loss function doesn't use any labels!

Dimensionality of Latent Space → Reconstruction Quality

Autoencoding is a form of compression!
Smaller latent space will force a larger training bottleneck

2D latent space

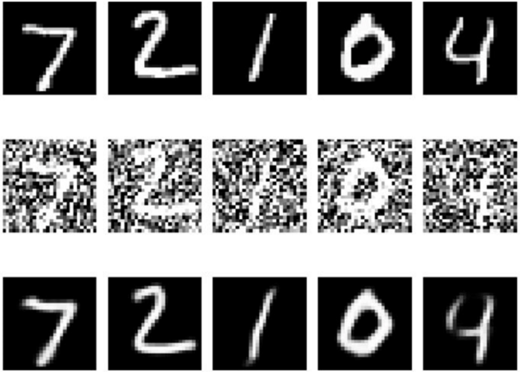
5D latent space

Ground Truth

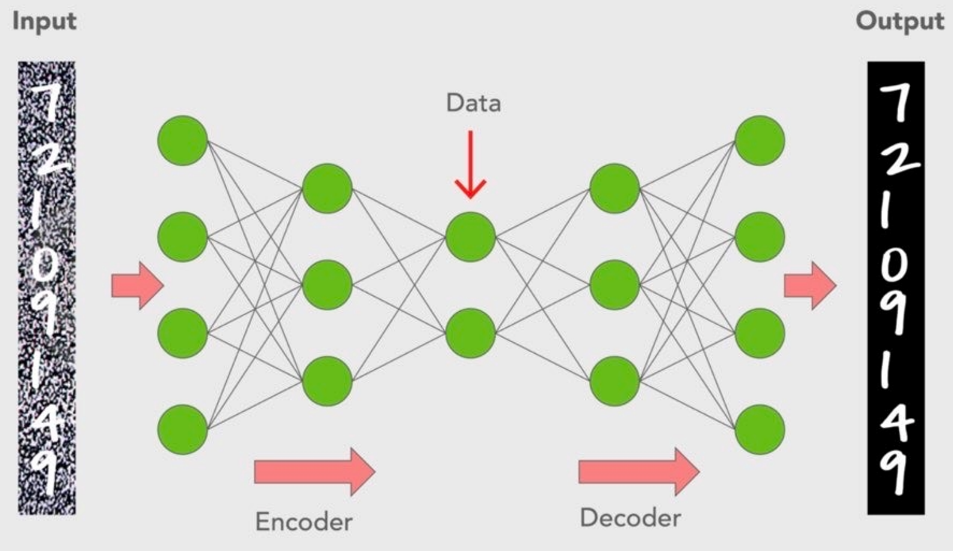
How can we use this information to create fair and representative datasets?

Autoencoders for Representation Learning

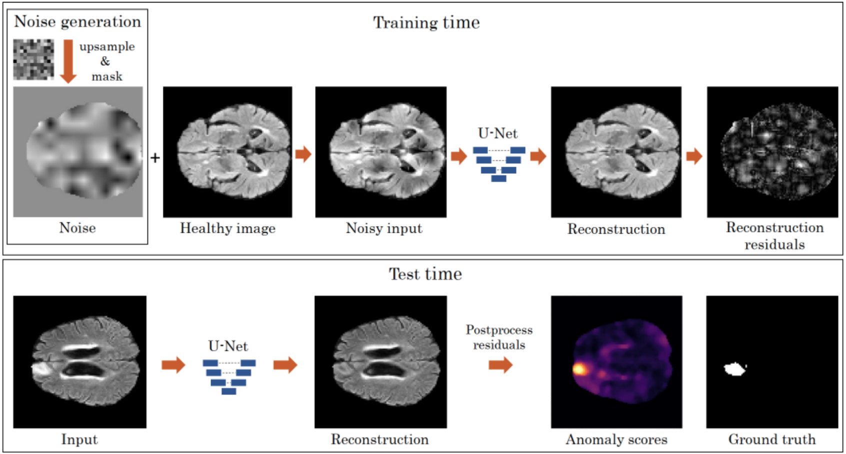
- **Bottleneck hidden layer** forces network to learn a compressed latent representation
- **Reconstruction loss** forces latent representation to capture (or encode) as much “information" about the data as possible
- **Autoencoding**- **Auto**matically **encoding** data



Denoiser Layout

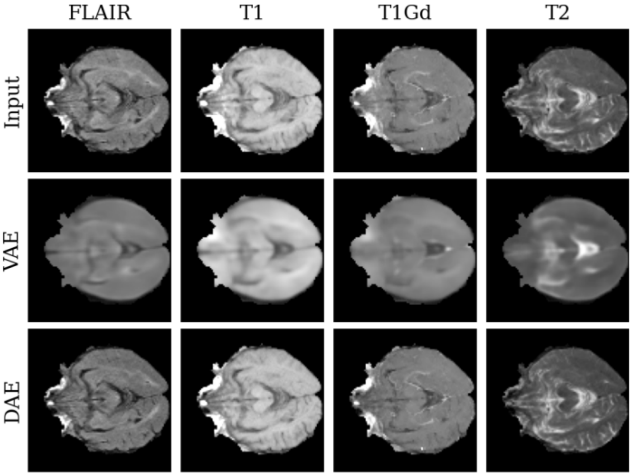


Autoencoders for Unsupervised Anomaly Detection



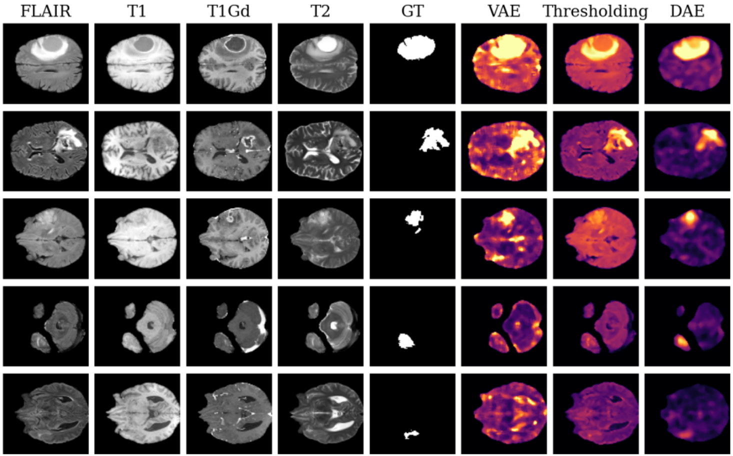
(top) Training: noise added to foreground of healthy image. Network trained to reconstruct original image. (Bottom) Test time, reconstruction error is used as the anomaly score.

Autoencoders for Unsupervised Anomaly Detection



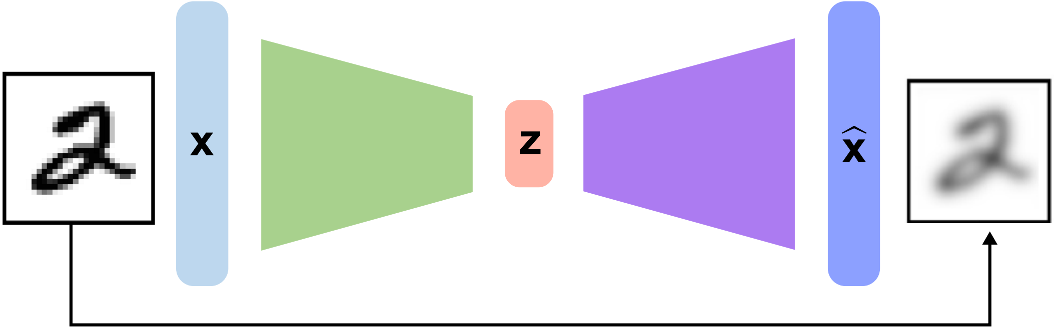
DAE gives more precise reconstructions. VAE reconstruction quality could be improved by increasing bottleneck dimensionality, however this would negatively impact anomaly detection performance.

Autoencoders for Unsupervised Anomaly Detection



Sample anomaly score predictions. From easier (top) to more difficult (bottom).

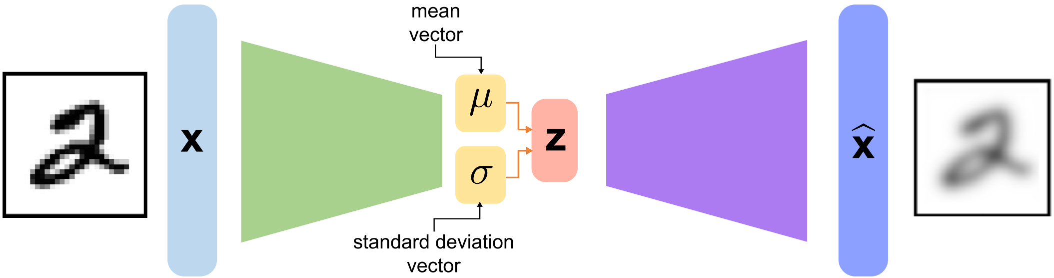
Traditional Autoencoders



$$\mathcal{L} = \| \mathbf{x} - \widehat{\mathbf{x}} \|^2$$

Loss function doesn't use any labels!

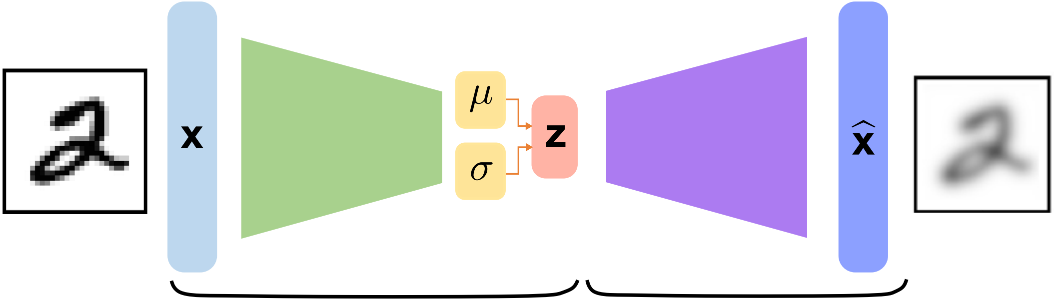
VAEs: Key Difference with Traditional Autoencoders



Variational autoencoders are probabilistic twist on autoencoders

Sample from the mean and standard deviation an to compute latent sample

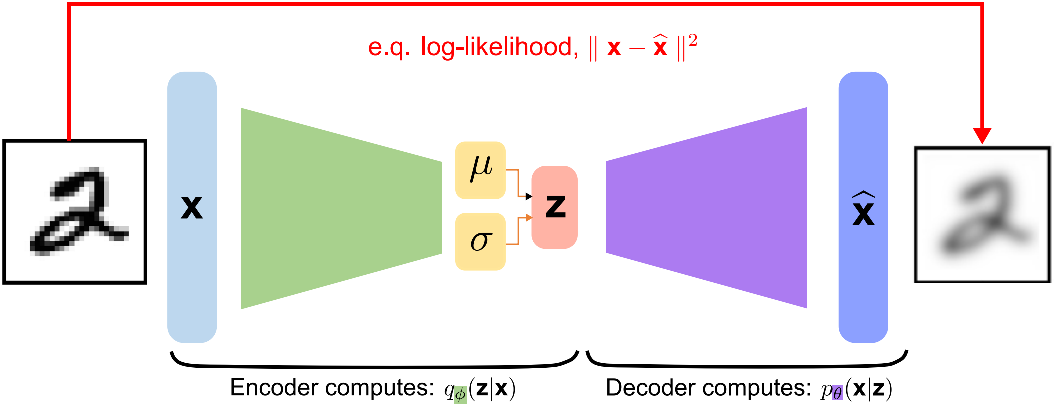
VAEs Optimization



Encoder computes: $q_{\phi}(\mathbf{z}|\mathbf{x})$ Decoder computes: $p_{\theta}(\mathbf{x}|\mathbf{z})$

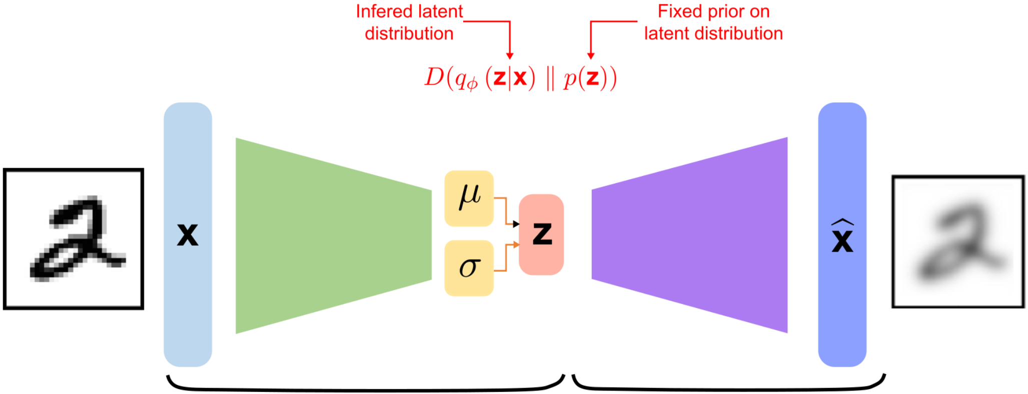
$$\mathcal{L}(\phi, \theta, \mathbf{x}) = (\text{reconstruction loss}) + (\text{regularization loss})$$

VAEs Optimization



$$\mathcal{L}(\phi, \theta, \mathbf{x}) = \text{(reconstruction loss)} + \text{(regularization loss)}$$

VAEs Optimization



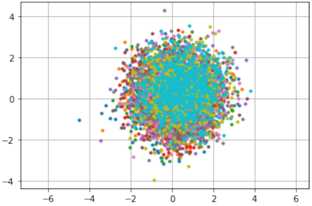
Encoder computes: $q_\phi(\mathbf{z}|\mathbf{x})$ Decoder computes: $p_\theta(\mathbf{x}|\mathbf{z})$

$$\mathcal{L}(\phi, \theta, \mathbf{x}) = (\text{reconstruction loss}) + (\text{regularization loss})$$

Priors on the Latent Distribution

KL-divergence between the two distributions

$$D\left(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\right) = -\frac{1}{2}\sum_{j=0}^{k-1}(\sigma_j^2 + \mu_j^2 - 1 - \log(\sigma_j))$$



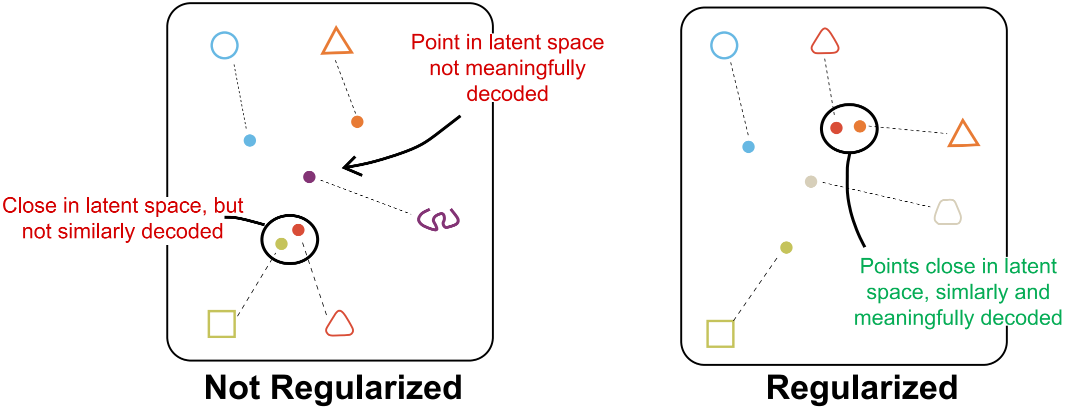
Common choice of prior - Normal Gaussian

$$p(\mathbf{z}) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalizes the network when it tries to "cheat" by clustering points in specific regions (i.e., by memorizing the data)

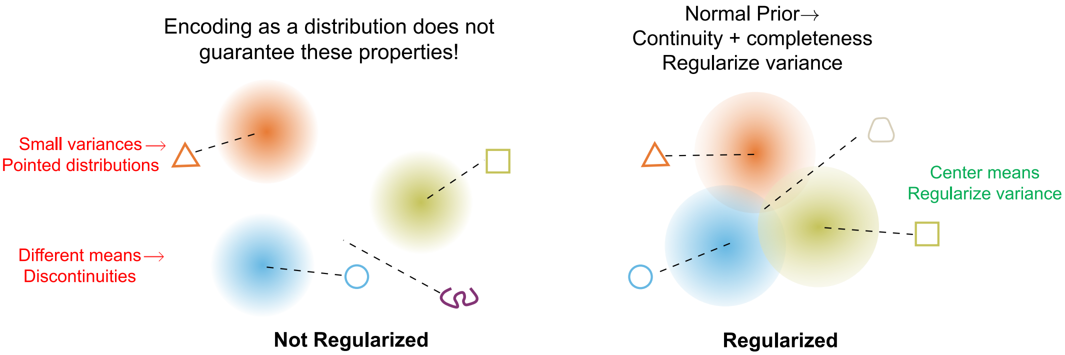
Intuition on Regularization and Normal Prior

1. **Continuity:** points that are close in latent space → similar content after decoding
2. **Completeness:** sampling from latent space → "meaningful" content after decoding



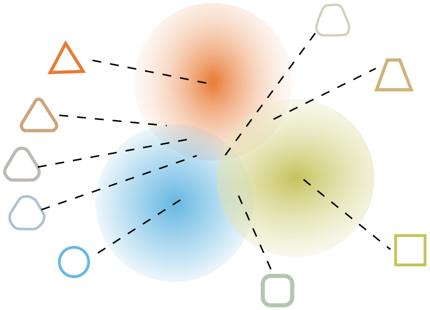
Intuition on Regularization and Normal Prior

1. **Continuity:** points that are close in latent space → similar content after decoding
2. **Completeness:** sampling from latent space → "meaningful" content after decoding



Intuition on Regularization and Normal Prior

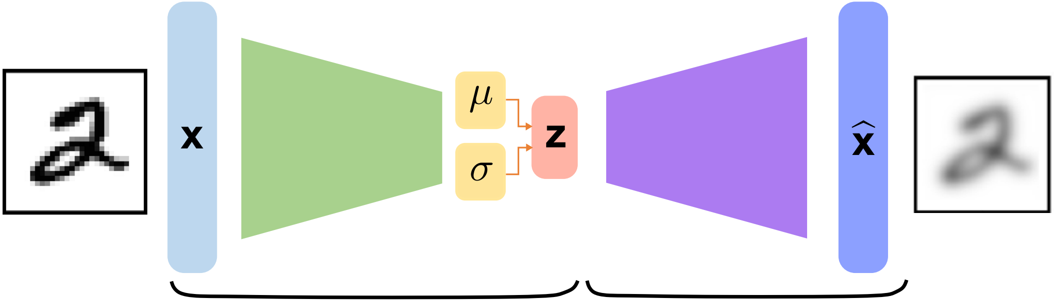
1. **Continuity:** points that are close in latent space → similar content after decoding
2. **Completeness:** sampling from latent space → "meaningful" content after decoding



Regularization with Normal prior helps enforce **information gradient** in the space

VAE Computation Graph

Problem: We cannot backpropagate gradients through sampling layers

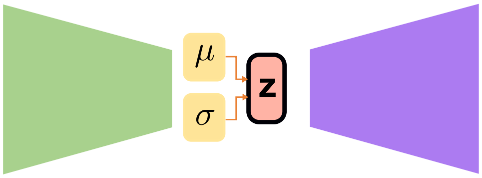


Encoder computes: $q_\phi(\mathbf{z}|\mathbf{x})$

Decoder computes: $p_\theta(\mathbf{x}|\mathbf{z})$

$$\mathcal{L}(\phi, \theta, \mathbf{x}) = (\text{reconstruction loss}) + (\text{regularization loss})$$

Reparametrizing the Sampling Layer



Key Idea:

$$z \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$$

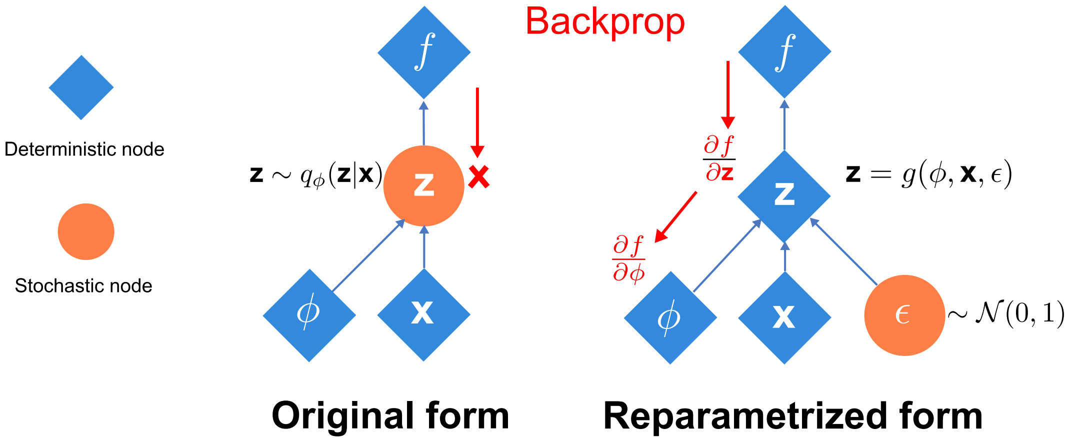
Consider the Sampled latent vector $\mathbf{z}$ as a sum of:

- A fixed $\boldsymbol{\mu}$ vector
- and fixed $\boldsymbol{\sigma}$, scaled by random constants drawn from the prior distribution

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$$

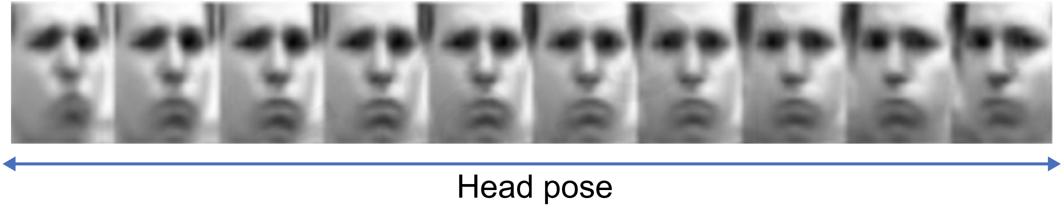
Where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, 1)$

Reparametrizing the Sampling Layer



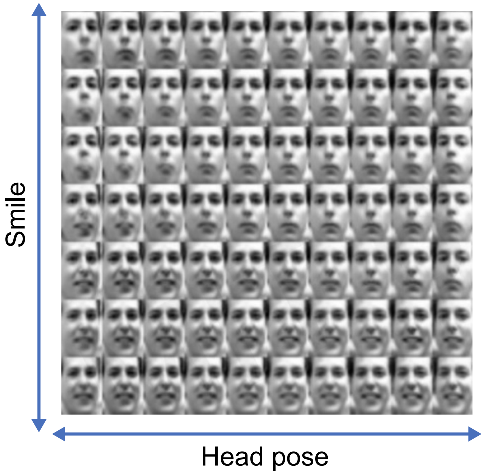
VAEs: Latent Perturbation

Slowly increase or decrease a **single latent variable**.
Keep all other variables fixed



Different dimensions of $\mathbf{z}$ encode different interpretable latent features

VAEs: Latent Perturbation



Ideally, we want latent variables that are uncorrelated with each other.

Enforce diagonal prior on the latent variables to encourage independence

Disentanglement

Latent Space Disentanglement with β-VAEs

Standard VAE loss

$$\mathcal{L}(\theta, \varphi; \mathbf{x}, \mathbf{z}) = \underbrace{\mathbb{E}_{q_\phi}\left[\log(p_\theta(\mathbf{x}|\mathbf{z}))\right]}_{\text{Reconstruction term}} - \underbrace{D_{\mathsf{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}))}_{\text{Regularization term}}$$

β-VAE loss

$$\mathcal{L}(\theta, \varphi; \mathbf{x}, \mathbf{z}) = \underbrace{\mathbb{E}_{q_\phi}\left[\log(p_\theta(\mathbf{x}|\mathbf{z}))\right]}_{\text{Reconstruction term}} - \underbrace{\beta D_{\mathsf{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}))}_{\text{Regularization term}}$$

$\beta > 1$: constrain latent bottleneck, encourage efficient latent encoding $\rightarrow$ disentanglement

Latent Space Disentanglement with β-VAEs

β-VAE loss

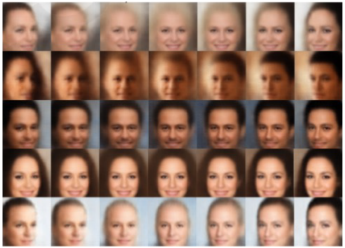
$$\mathcal{L}(\theta, \varphi; \mathbf{x}, \mathbf{z}) = \underbrace{\mathbb{E}_{q_\phi}[\log(p_\theta(\mathbf{x}|\mathbf{z}))]}_{\textbf{Reconstruction term}} - \underbrace{\beta D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \,\|\, p(\mathbf{z}))}_{\textbf{Regularization term}}$$

$\beta > 1$: constrain latent bottleneck, encourage efficient latent encoding $\rightarrow$ disentanglement

Head rotation (azimuth)

Smile also changing!

Standard VAE ($\beta = 1$)



β-VAE ($\beta = 250$)

Smile relatively constant!

Why Generative Models? Debiasing

Capable of uncovering **underlying features** in a dataset

Homogeneous skin color, pose

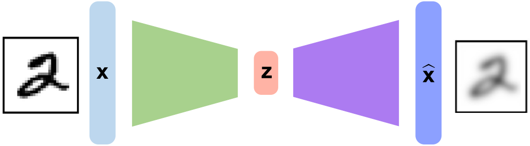
vs

Diverse skin color, pose, illumination

How can we use this information to create fair and representative datasets?

Traditional Autoencoders

1. Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)
3. Reparametrization trick to train end-to-end
4. Interpret hidden latent variables using perturbation
5. Generating new examples



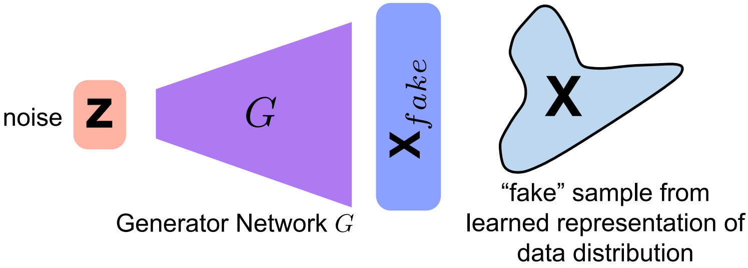
Generative Neural Networks (GANs)

What if we just want so sample?

Idea: don't explicitly model density, and instead just sample to generate new instances

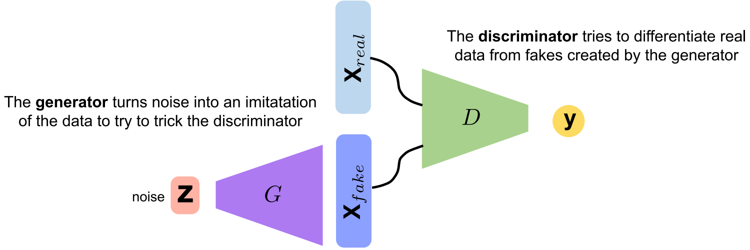
Problem: want to sample from complex distribution - can't do this directly

Solution: sample from something simple (e.g., noise), learn a transformation to the data distribution



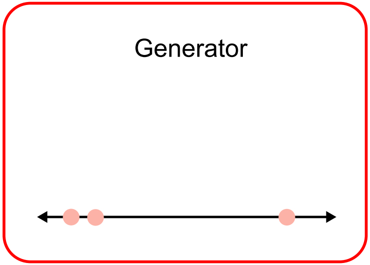
Generative Neural Networks (GANs)

Generative Adversarial Networks (GANs) are a way to make a generative model by having two neural networks compete with each other.



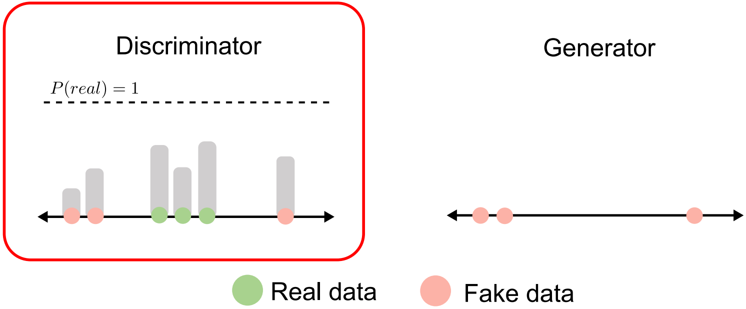
Intuition Behind GANs

Generator starts from noise to try to create an imitation of the data.



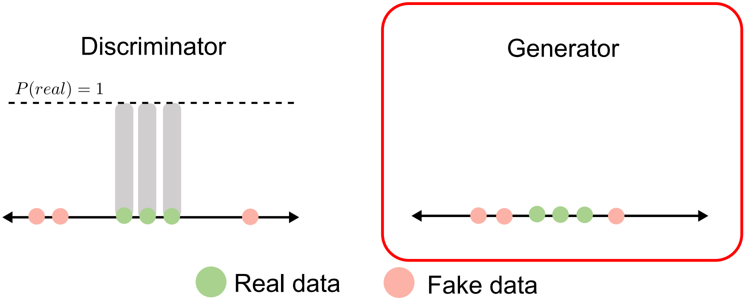
Intuition Behind GANs

Discriminator tries to predict what's real and what's fake.



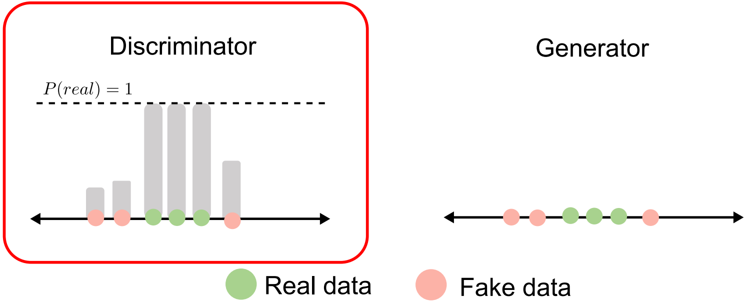
Intuition Behind GANs

Generator tries to improve its imitation of the data.



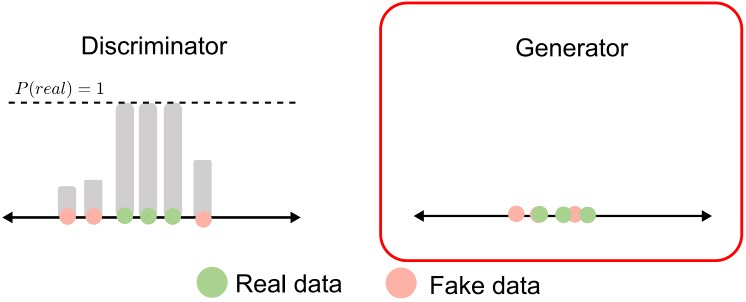
Intuition Behind GANs

Discriminator tries to predict what's real and what's fake.



Intuition Behind GANs

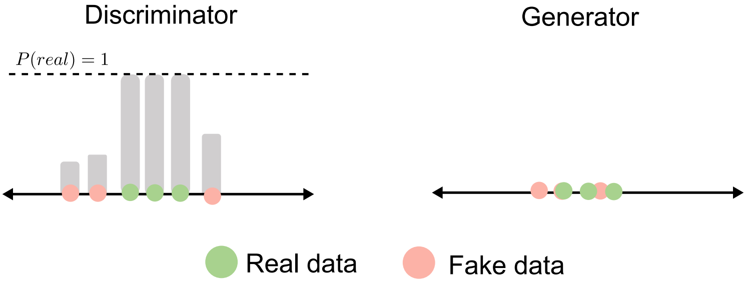
Generator tries to improve its imitation of the data.



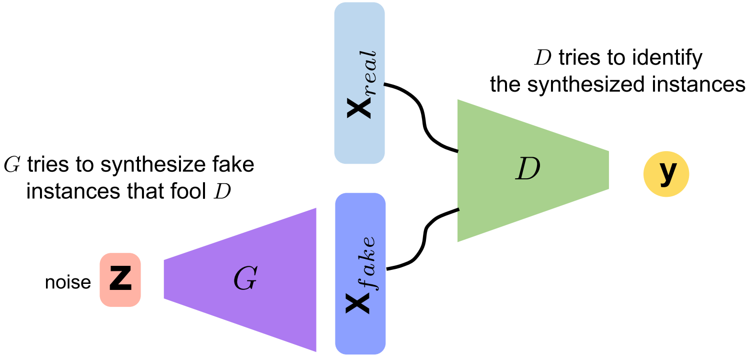
Intuition Behind GANs

Discriminator tries to differentiate real data from fake data created bt the generator.

Generator tries to create imitations of data to trick the discriminator.

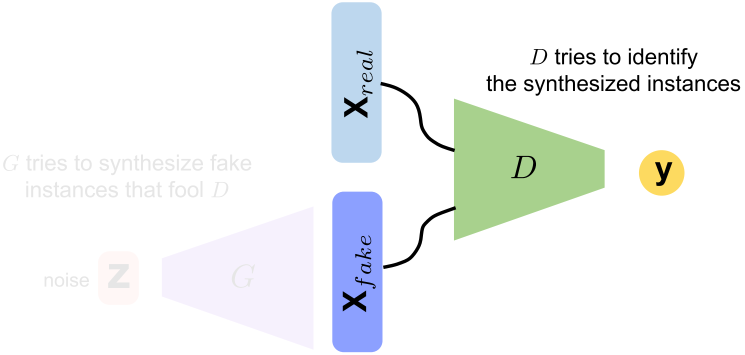


Training GANs



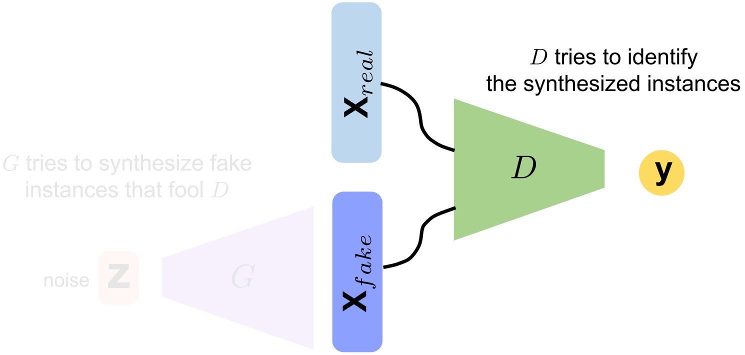
Training: adversarial objective for D and G
Global optimum: G reproduces the true data distribution

Training GANs: Loss Functions



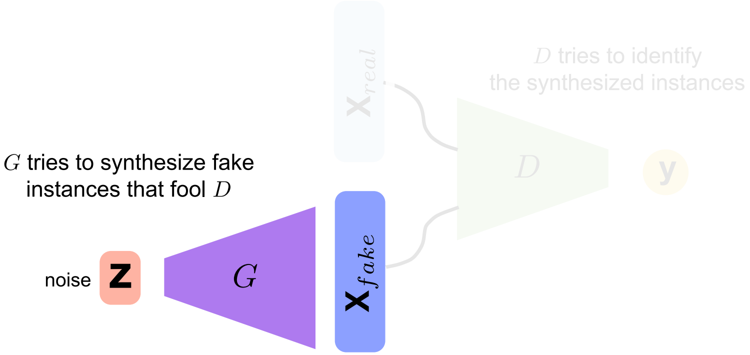
$$\text{argmax}_D \mathbb{E}_{\mathbf{z},\mathbf{x}} [\underbrace{\log(D(G(\mathbf{z})))}_{\textbf{fake}} + \underbrace{\log(1 - D(\mathbf{x}))}_{\textbf{real}}]$$

Training GANs: Loss Functions



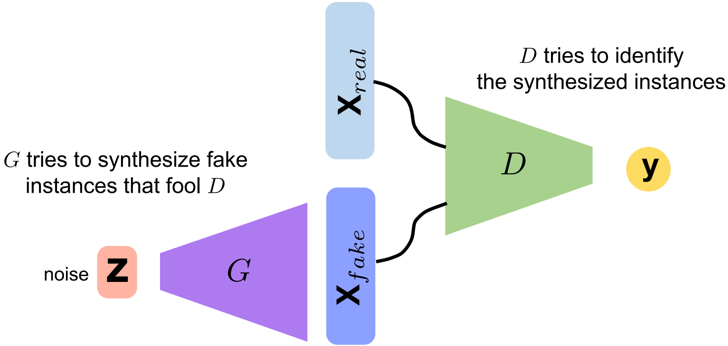
$$\text{argmax}_{D}\mathbb{E}_{\mathbf{z},\mathbf{x}}\left[\underbrace{\log(D(G(\mathbf{z})))}_{\textbf{fake}} + \underbrace{\log(1 - D(\mathbf{x}))}_{\textbf{real}}\right]$$

Training GANs: Loss Function



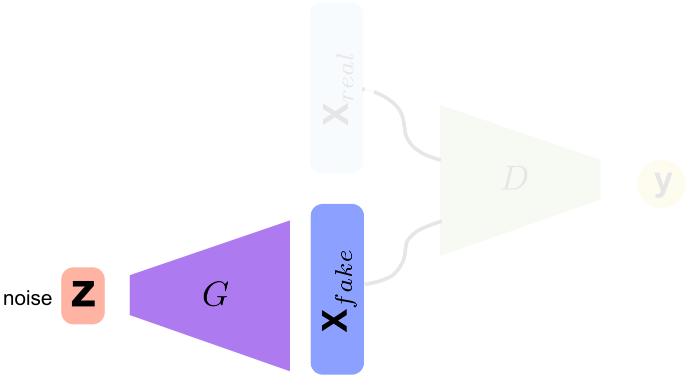
$$\text{argmin}_G \mathbb{E}_{\mathbf{z},\mathbf{x}} \left[\underbrace{\log(D(G(\mathbf{z})))}_{\textbf{fake}} + \underbrace{\log(1 - D(\mathbf{x}))}_{\textbf{real}}\right]$$

Training GANs: Loss Function



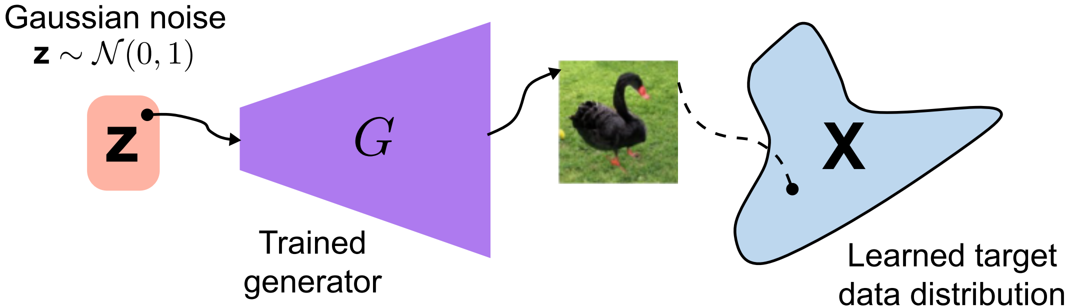
$$\text{argmin}_G \text{max}_D \mathbb{E}_{\mathbf{z},\mathbf{x}} \left[\log(D(G(\mathbf{z}))) + \log(1 - D(\mathbf{x})) \right]$$

Generating New Data with GANs

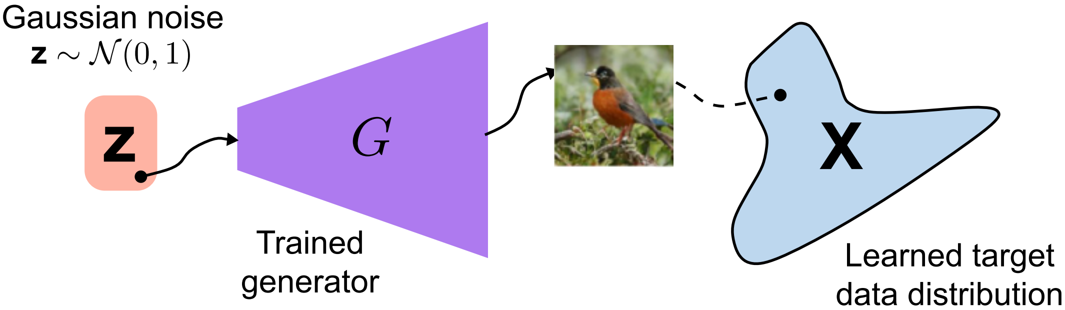


After training, use generator network to create **new data** that's never been seen before.

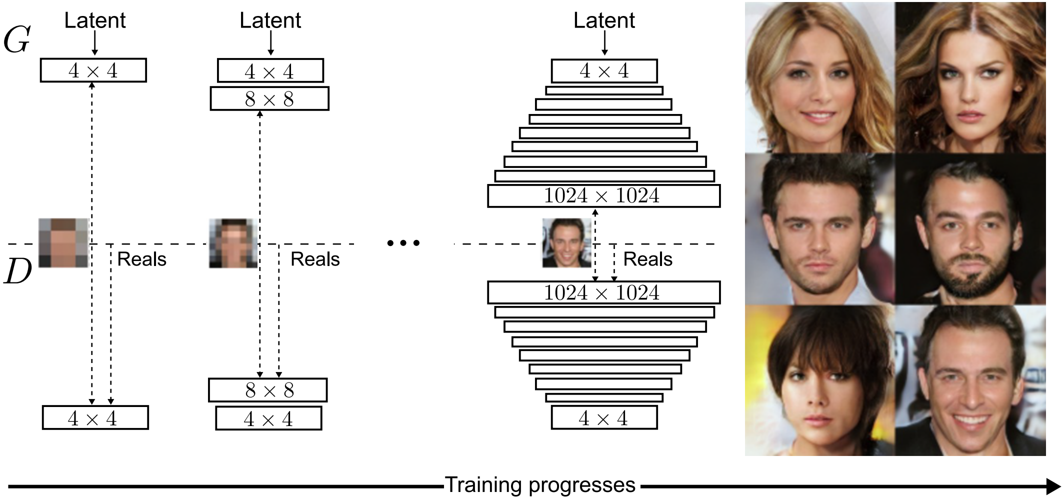
GANs are Distribution Transformers



GANs are Distribution Transformers

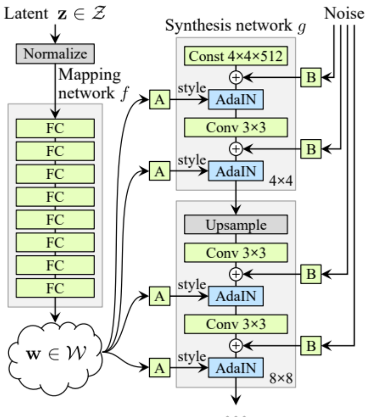


GANs Recent Advances: Progressive Growing of GANs



GANs Recent Advances: Progressive Growing of GANs

GANs Recent Advances: StyleGAN(2): Progressive Growing + Style Transfer

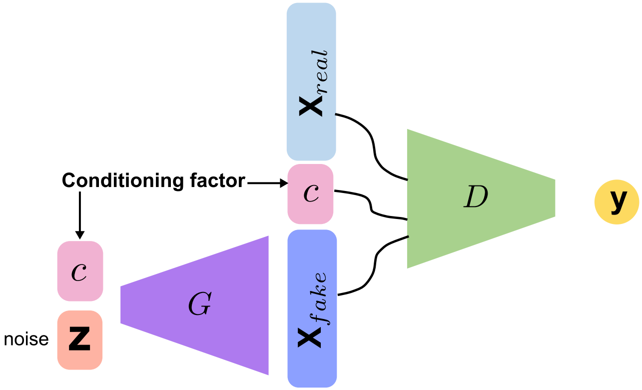


GANs for Image Synthesis: Latest Results

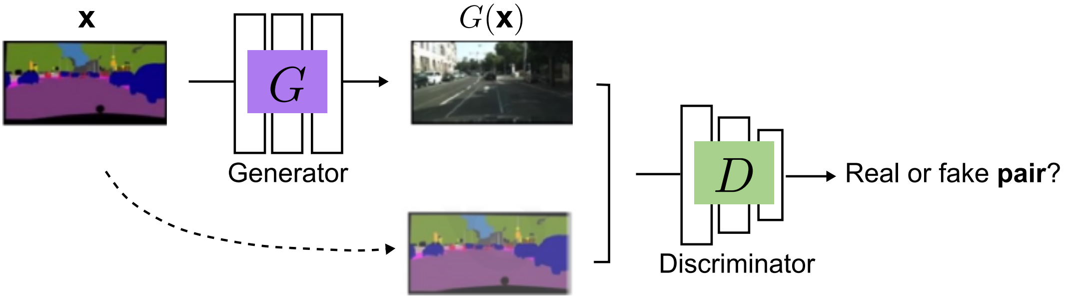
GANs for Image Synthesis: Latest Results

Conditional GANs

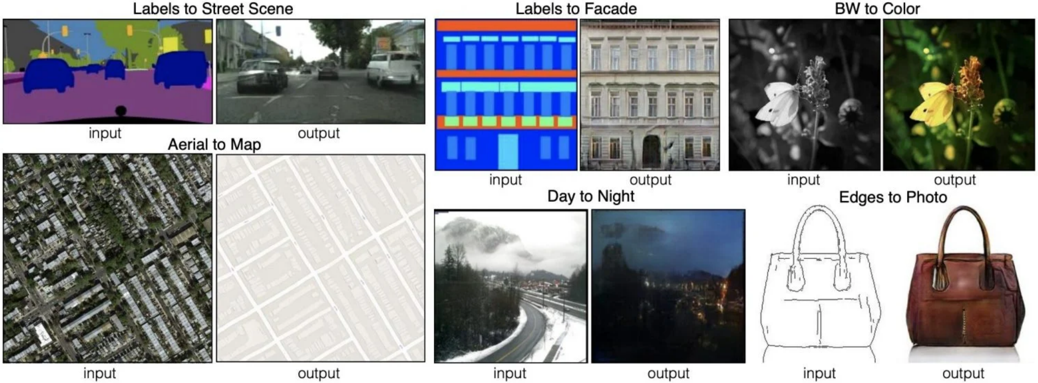
What if we want to control the nature of the output, by **conditioning** on a label?



Conditional GANs and Pix2Pix: Paired Translation



Applications of Paired Translation



Paired Translation: Results

Map→ Aerial View

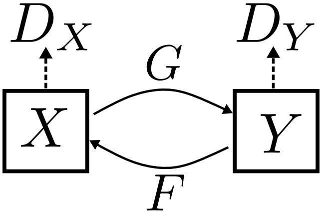
Input **Output**

Aerial View→ Map

Input **Output**

CycleGAN: Domain Transformation

CycleGAN learns transformation across domains with unpaired data

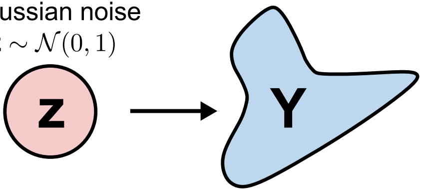


www.youtube.com/watch

Distribution Transformation

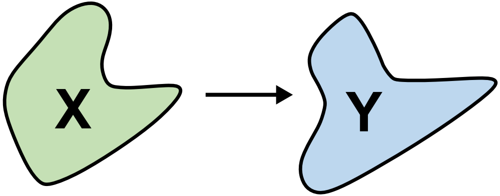
GANs:

Gaussian noise
$\mathbf{z} \sim \mathcal{N}(0, 1)$



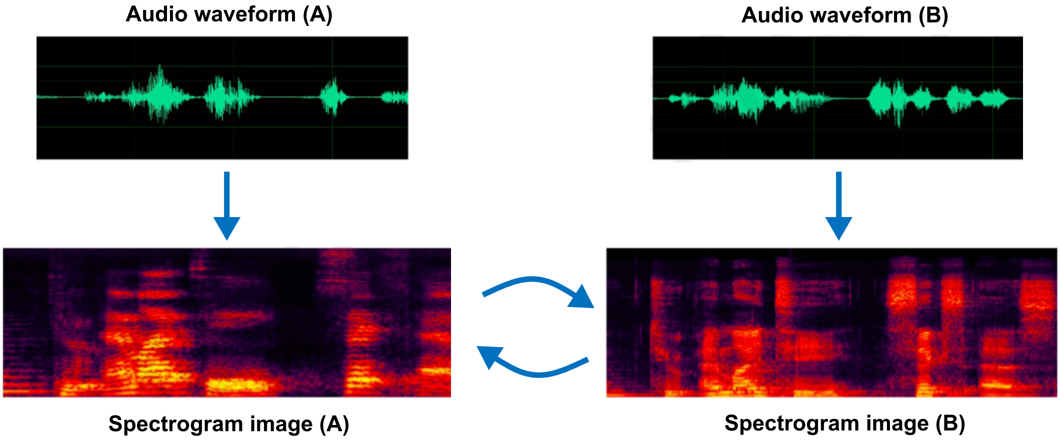
Gaussian noise $\rightarrow$ target data manifold

CycleGANs:



data manifold **X** $\rightarrow$ data manifold **Y**

CycleGAN: Transforming Speech



Acknowledgement

Alexander Amini and Ava Soleimanym, MIT 6.S191: Introduction to Deep Learning, IntroToDeepLearning.com