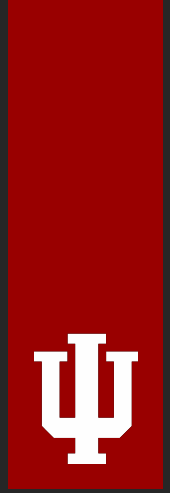
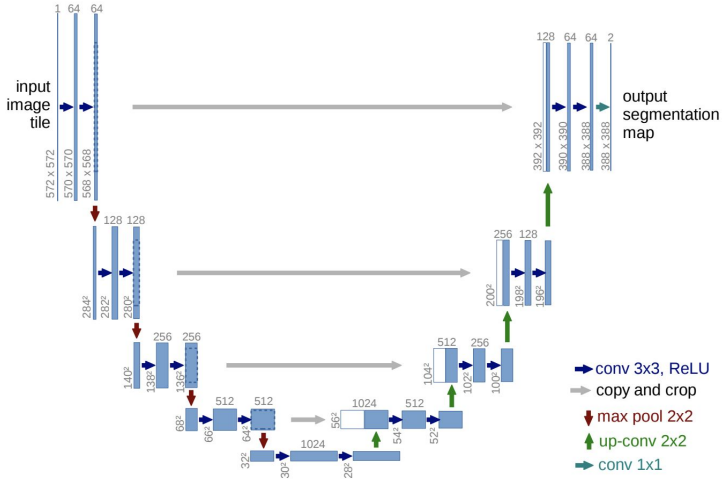


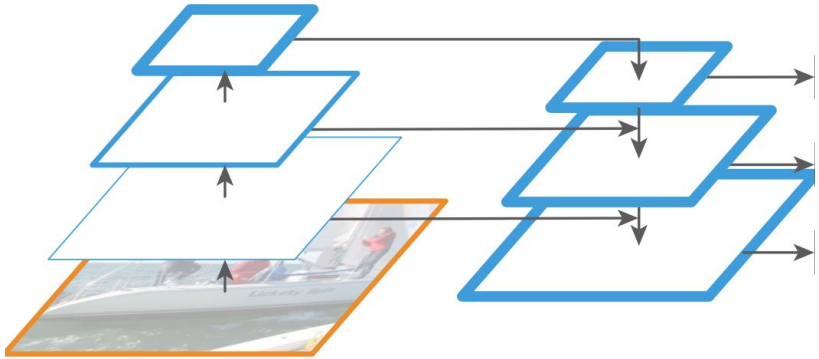
CSCI- B 657 - Discussion Section

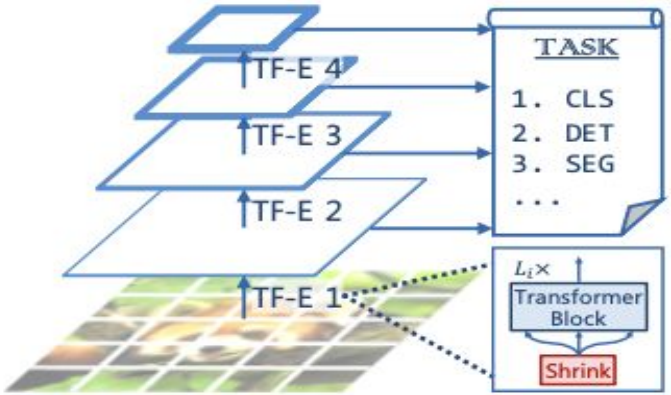
Vibhas Vats

Administrative

Last Discussion







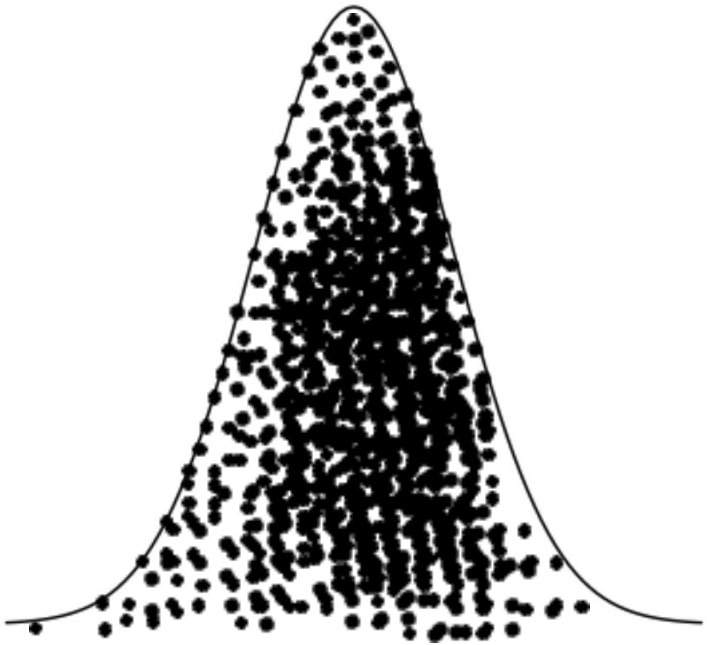
Pyramid Vision Transformer

Today

- Data distribution
  - Latent representation of data
- Generative models
  - Vanilla AutoEncoders
    - discrete latent representations
  - Variational AutoEncoders (VAEs)
    - continuous latent representations
    - conditional VAEs
  - Generative Adversarial Networks (GANs)
    - optimization process
    - mode collapse
  - Conditional GANs

The Data Distribution - Pixel Space

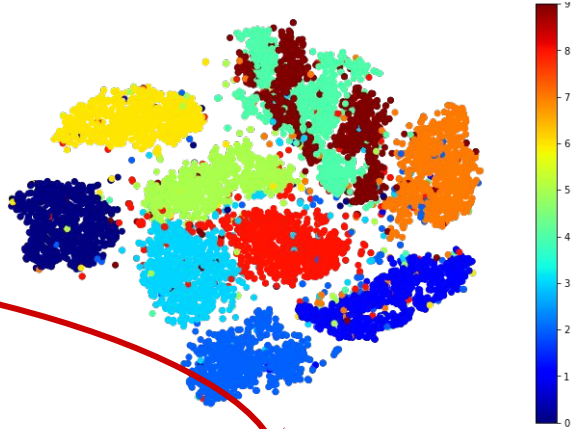
- Digit dataset: There are infinite variations of writing each dig
  - *continuous* change from one variation to another
  - A true distribution captures all such variations
- **MNIST**:
  - 100k images sampled from the true distribution
  - it closely follow the true distribution
  - but we can't possibly capture all variations
    - missing data points → holes in distribution
- What can we say about the missing data points?
  - nothing with confidence!
  - we don't know the nature of the missing data points
    - digits → but can't say more!
- Consequence:
  - a model trained on MNIST with 100% accuracy
  - will still make mistakes if tested with the missing data point samples
  - It will always happen → independent of the size of a dataset



The Data Distribution - Latent Space

- Latent space: an N-dimensional space

PCA and Plot



use discrete samples to learn continuous latent space

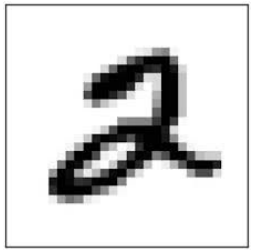
- PCA → extracts the two main dimensions from each image
- A lot of missing data points
- Class overlaps:
  - 4 looks like 9
  - 5 looks like 6

Discrete representation

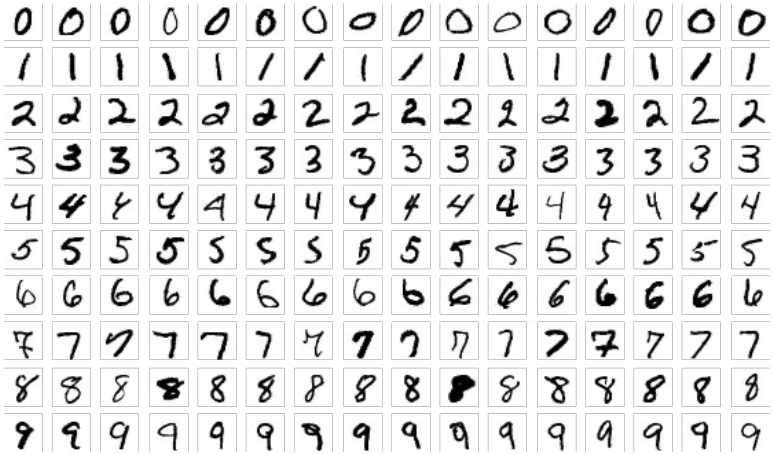
- A **continuous** distribution looks strange →
  - 1 close to 2 and 7, 7 → 9 → 4, 0 → 6 and 5, 3 → 8

N-Dims Latent Representation

- Images → representation in pixel-space
- **MNIST**:
  - pixel space: 28x28 = 784D
  - most of the pixels are *insignificant*
    - most of the dimensions are *redundant*



- Features of such images can be captured in much smaller dimensions
  - *10-dims*: 1-dim/digit for all variations
  - *15-dims*: 1-dim/digit and
    - thickness of stroke
    - right side tilt: 1, 2, 4,5,8
    - left side tilt: 6
    - shape variation: types of 2, 4, 7
    - similarity with others



- Generating such representations in much smaller dims → *Latent representation* (in latent space)

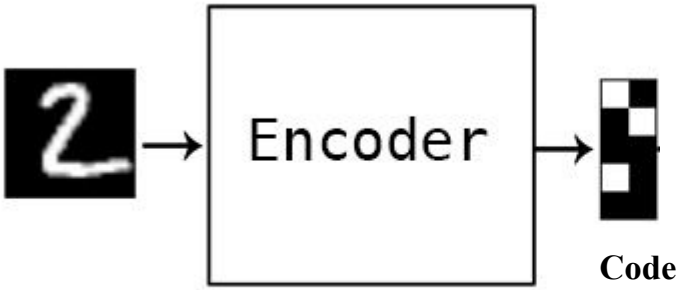
10D Latent Representation of MNIST

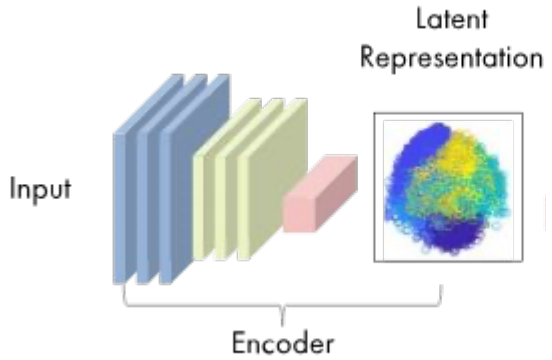
0	1	2	3	4	5	6	7	8	9	Digits
0	0.75	0	0	0	0	0	0	0	0	 a regular representation of 1
0	0.75	0.2	0	0	0	0	0.2	0	0	 a digit 1 with some similarity with digit 2 and 7
0	0	0	0	0	0.63	0.53	0	0	0	 a digit 5 with high similarity with digit 6
0.65	0	0	0	0	0	0.2	0	0	0	 a digit 0 with similarity with digit 6

- Most of the required information can be captured in much smaller dims for all types of images
- This latent representation → *code/latent vector (LV)*
  - size of dimension → hyperparameter
- How to generate it?
  - PCA, t-sne or other algorithmic compression methods
  - **learn from data → Encoder**

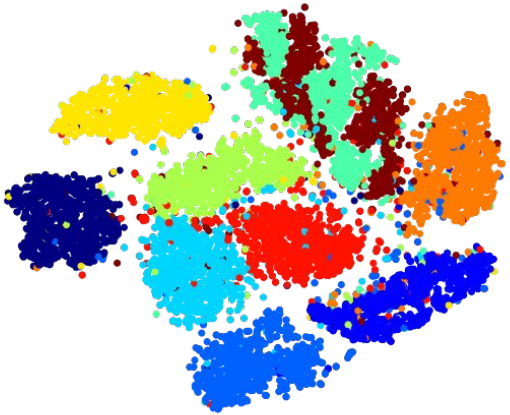
Learning Latent Representation - Encoder

- A neural network with hierarchical structure → generate LV



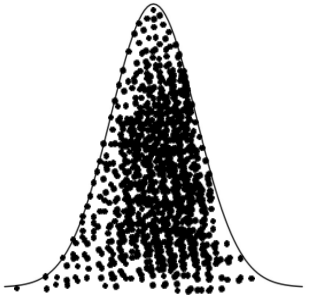


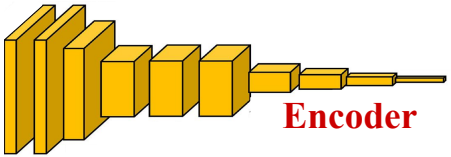
- Why compress images?
  - most dims are redundant → save parameters
  - high dims representation are much sparser
    - even 2D is sparse
  - efficient representation
  - compact representation is useful for further tasks
    - like classification, decoding, etc.

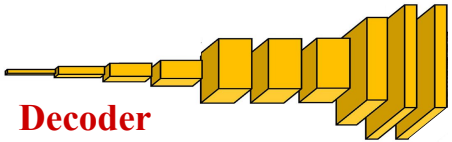


Generative Models

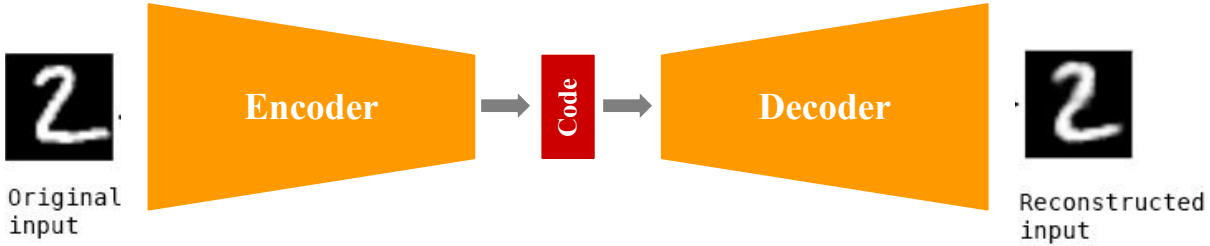
- Discriminative models → Classification models
  - discriminate between different kinds of data instances
- Generative models → generate *new data instances*
  - **step 1**: learn the underlying data distribution
    - in pixel space → too much sparsity
      - difficult with existing techniques
    - in latent space → efficient
      - a simple encoder can do it
  - **step 2**: generate using the latent representation
    - sample points from learned latent space
    - use decoder to generate *new images*
- Can't be done separately → combine both steps



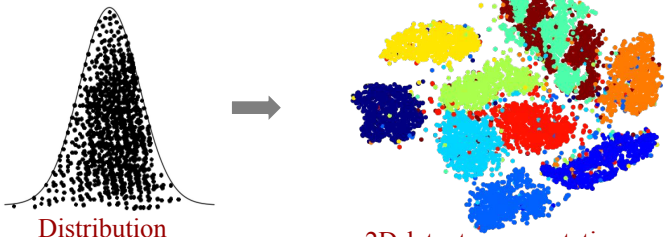




Vanilla Auto Encoders

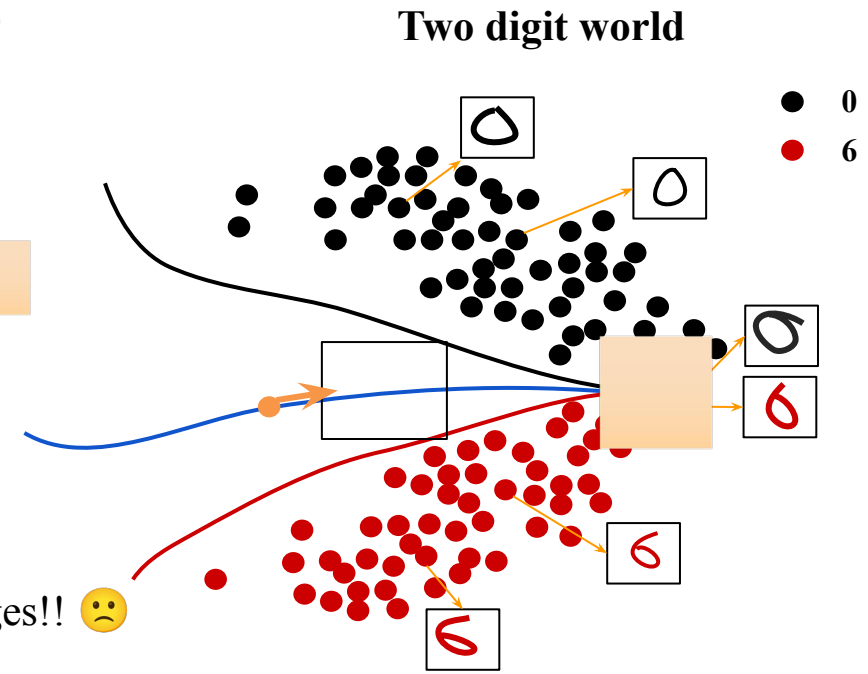


- **AE**: Input image is *coded* and *decoded* in one go
- Ideal condition:
  - minimal loss of *'information'* in encoding images
  - *information*: sufficient representation for decoder to reconstruct the input
- After training → originally '*similar examples share similar codes*'
  - weights are fixed after learning → same image lead to same code
- **Problem**:
  - can it generate images using unseen codes?
    - no → network is not aware of unseen codes
    - latent space is discrete → not continuous
      - because our samples are discrete



Insufficiency of Vanilla AE

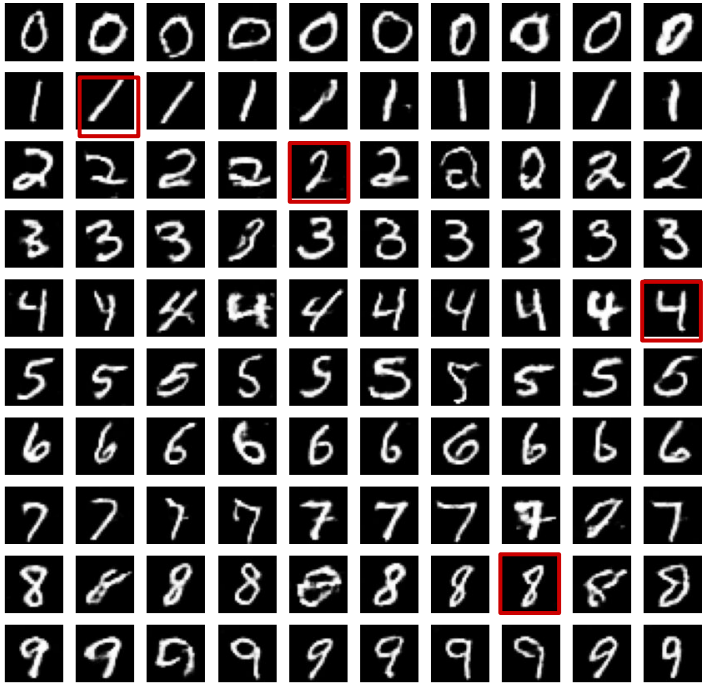
- AEs do not have control over latent variables (LV)
  - only generate LV for seen images
  - discrete points in latent space
- For a sample in the latent space b/w two classes
  - what will the model generate?
- Depending on the boundary b/w 0 and 6 in LS
  - nature of the point will change
- Generation process can't be smooth with
  - discrete knowledge of the LS
  - And we can't really sample all possible images!! 😕
  - 🐓 and 🥚 problem!
- ***Variational Auto-Encoders*** (VAEs): Try to sample from a continuous distribution in the latent space '*somehow*'!!

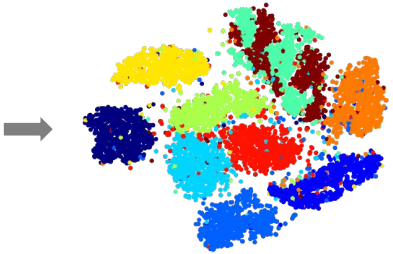


slide inspired from ENGR-E533

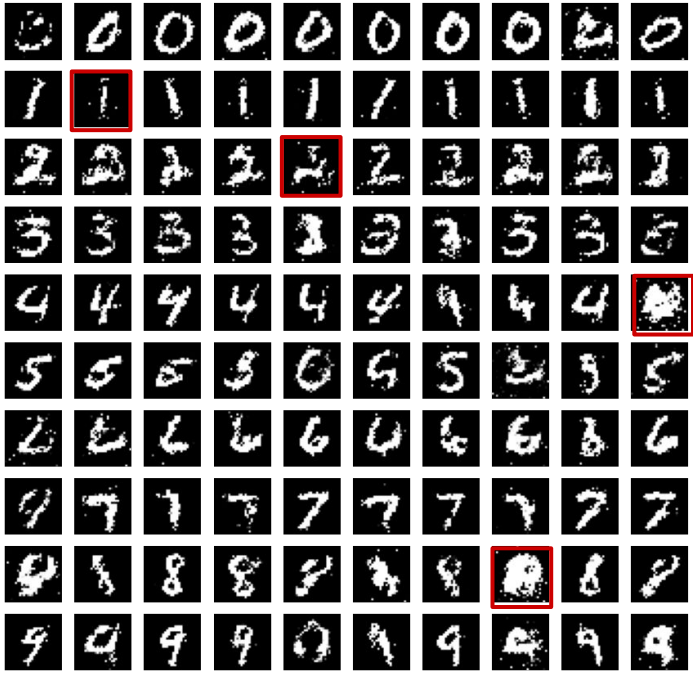
Vanilla VE Image Generation

Test images

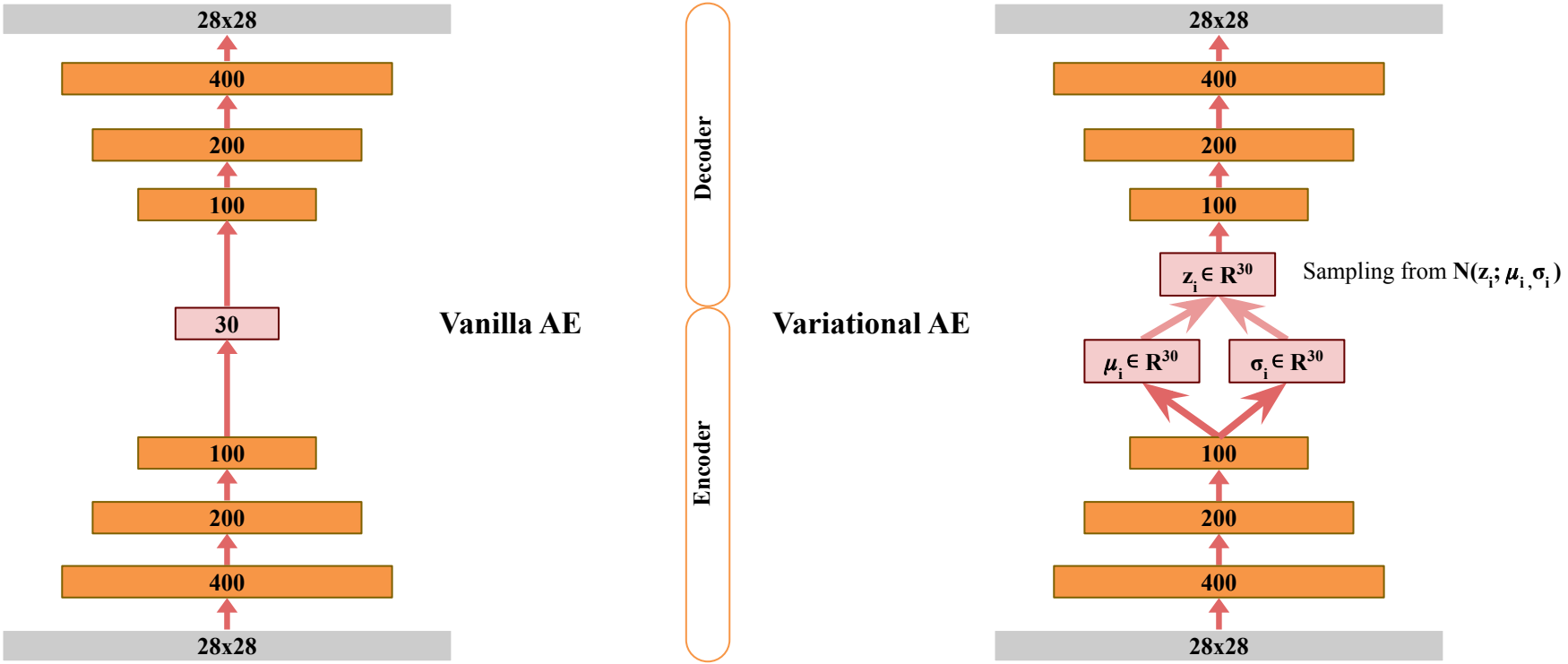




Generated images



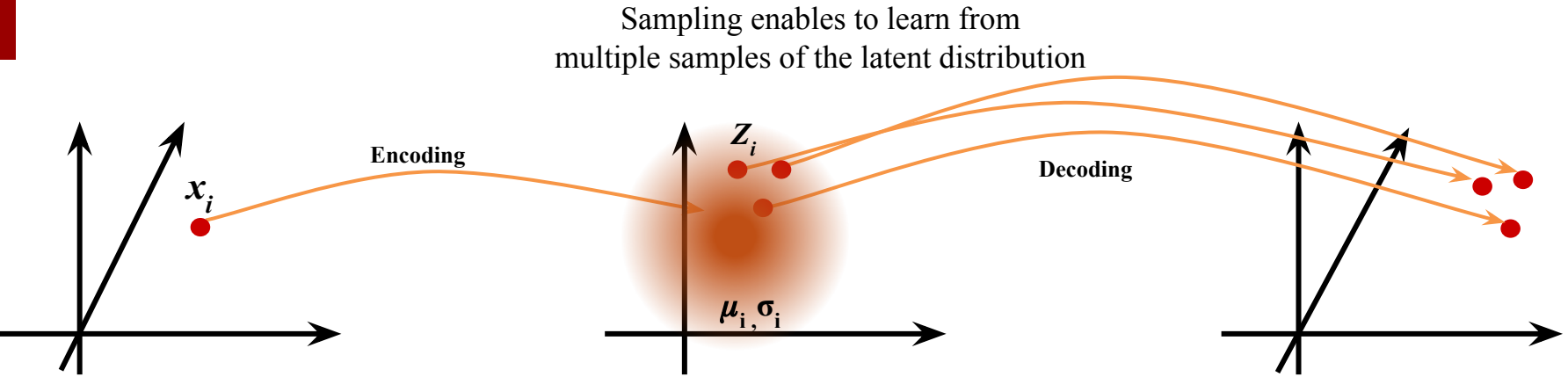
Variational AutoEncoders



- Sampling makes the whole process non-differentiable!! → Reparameterization trick

slide inspired from ENGR-E533

Variational AutoEncoders

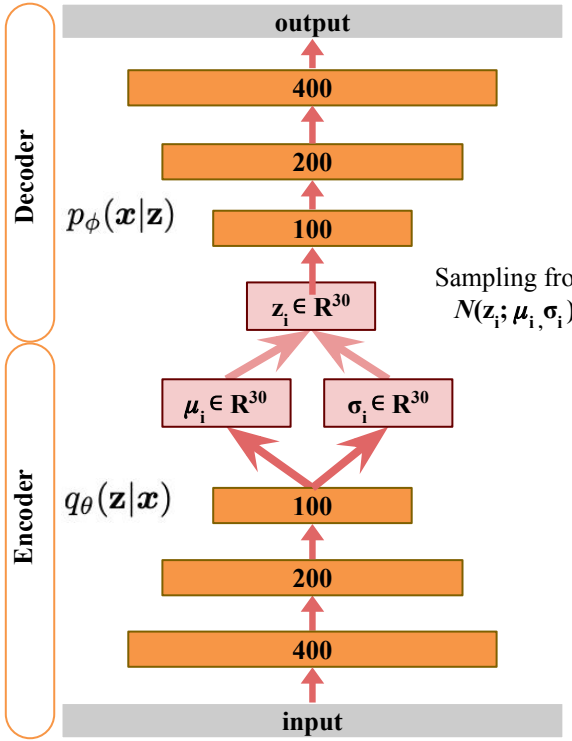


- From one example, we got a distribution with all nearby variations → making LS continuous
- During Training
  - The reconstructed output is from a sampled version of the latent variable
  - The training algorithm is exposed to the continuous distribution of the latent space distribution
    - this is not possible in vanilla AE
- During testing
  - it produces stochastic predictions due to the sampling process

slide inspired from ENGR-E533

Objective Function of VAE

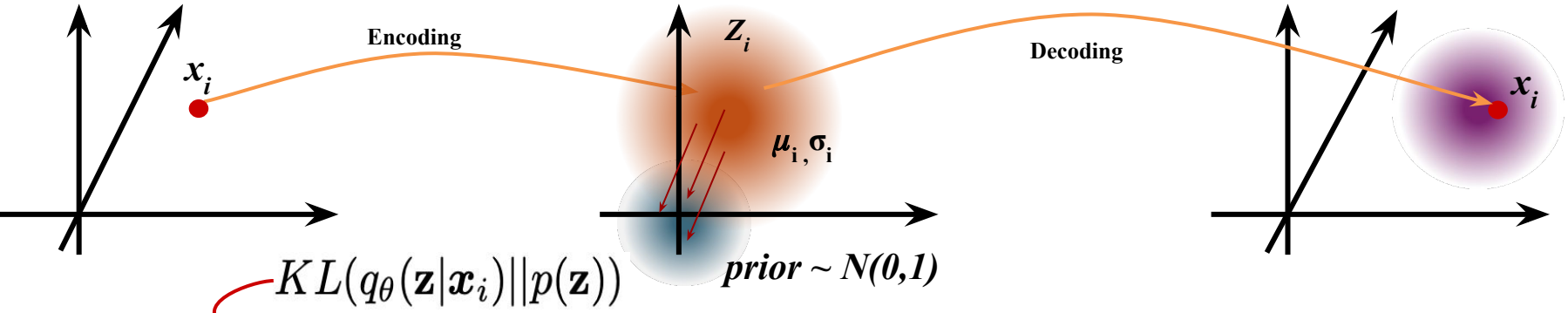
- Encoder
  - posterior distribution over LV given the data samples
  - parameterized by the weights of the network
- Decoder:
  - distribution over the multivariate observations (pixels) given the LV
  - also parameterized by the weights of the network



- expectation over LV
- Likelihood of observing the input given the LV
- *Regularization*: the LV should be close to its *prior*, usually *N(0,1)*

slide inspired from ENGR-E533

Role of the Prior



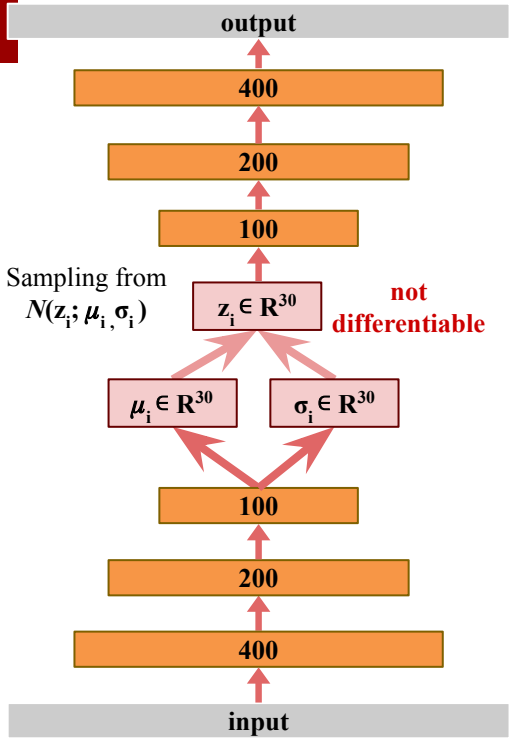
minimizing KL

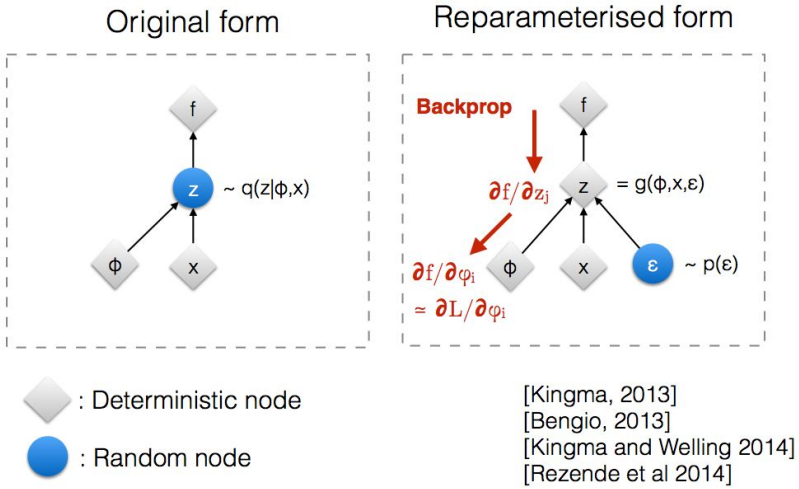
measures the difference b/w two distributions

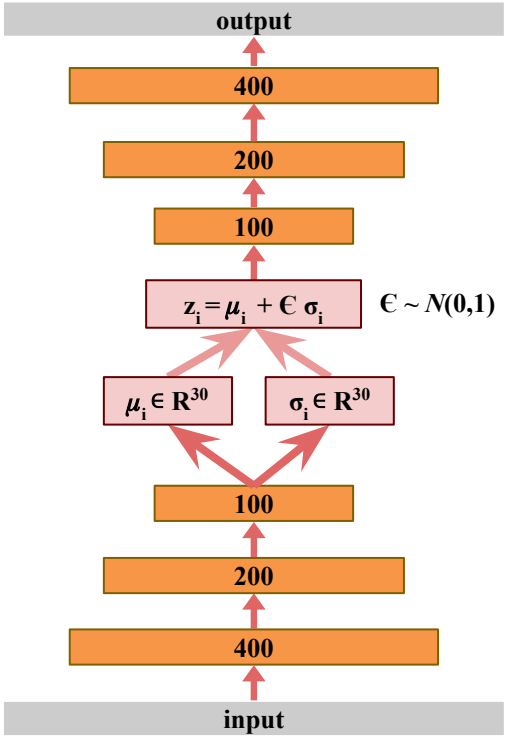
- We want LV distribution to look like a standard normal
  - it will provide control over the latent space distribution
- Apply a KL-divergence loss function b/w LV distribution and standard normal prior
  - at each step, the prior will nudge the LV distribution towards standard normal

slide inspired from ENGR-E533

Reparameterization Trick for Backprop





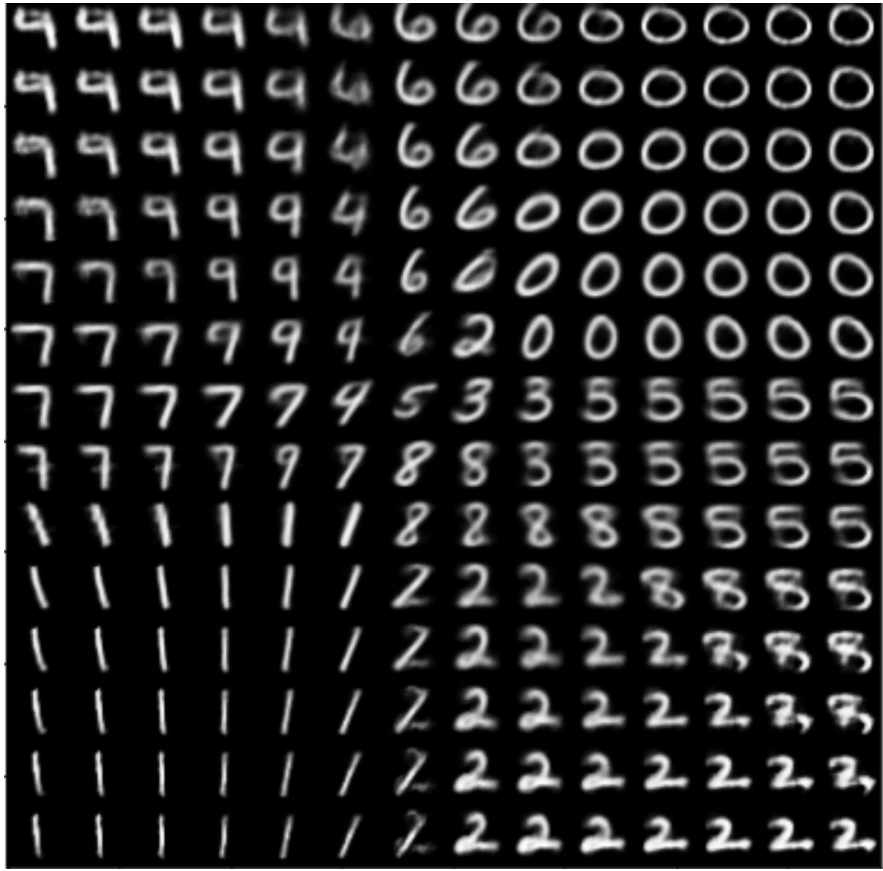


- After reparameterization, Z_i is differentiable w.r.t. both its parameters
- Why multiple with normal distribution?
  - adds randomness around the mean value of Z_i for continuous variations

slide inspired from ENGR-E533

VAE Image Generation

- Images with continuous variations are generated with VAE
  - Some digits looks like perfect fusion of two digits
    - boundary sample in latent space
  - Many digits do not make any sense
  - But for sure, the latent space is continuous!!
- **Conditional VAE:**
  - VAEs generates random images as per its understanding of the LS → no control
  - controlled using *conditional-VAE*
  - class information is provided to encoder and decoder during learning



Drawback of VAEs

- Objective function of VAEs

Encoder: LV given input image x_i

N(0,1) **prior**

$$\mathcal{L}_i(\theta, \phi) = -\mathbb{E}_{\mathbf{z} \sim q_\theta(\mathbf{z}|\boldsymbol{x}_i)}[\log p_\phi(\boldsymbol{x}_i|\mathbf{z})] + KL(q_\theta(\mathbf{z}|\boldsymbol{x}_i)||p(\mathbf{z}))$$

- If KL-div is large ⇒ no proper optimization ⇒ poor image generation

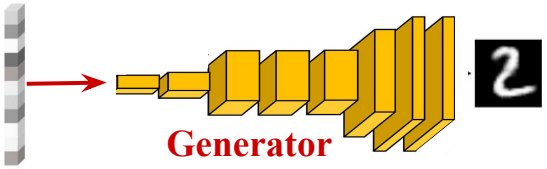
Why?

  - very strong prior distribution *p(z)* → prior distribution *different* than the data distribution
    - if the prior is **not** proper → poor optimization → poor reconstruction
  - another probabilistic explanation exits
    - involving evidence lower bound (ELBO) ✖

- GANs can produce better results

How GAN does it?

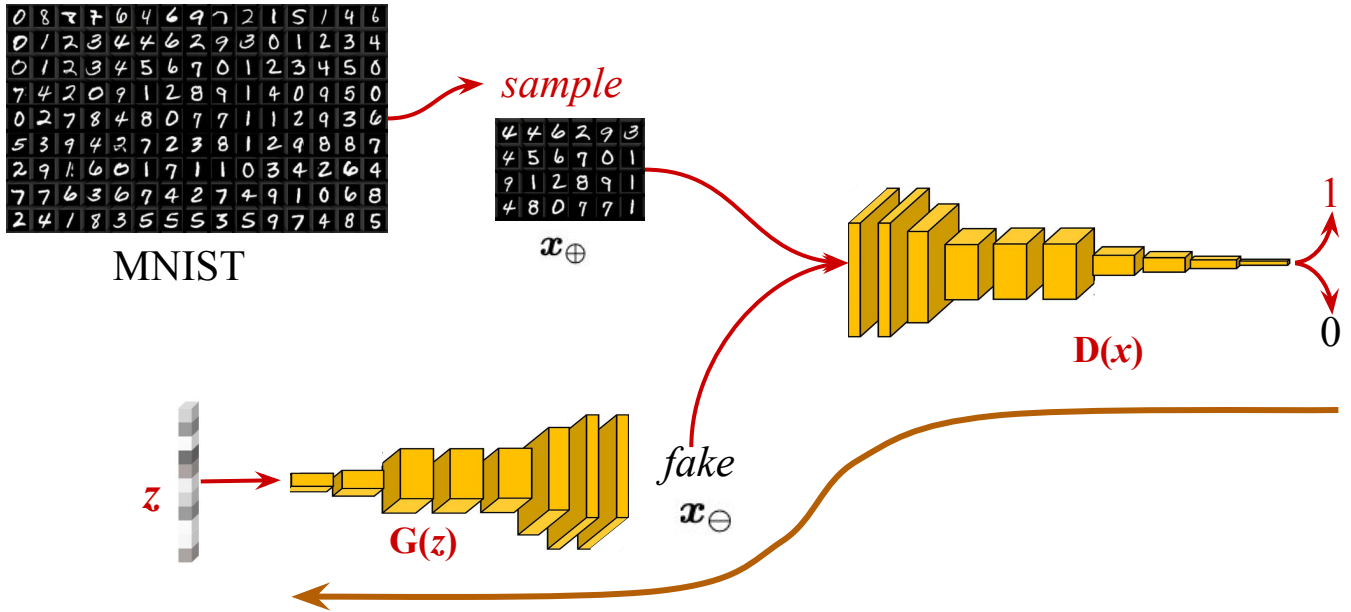
- **Generator (G):**
  - from an *N-dimensional code* → generate images
  - *main goal*: to produce images that looks real
    - real to whom? → **to human perception**
- But we can't have human in a training loop!
  - Another network → acts as *proxy* for human
  - **aim**: to discriminate b/w *real* and *fake* images
- **Discriminator (D):**
  - How to train such a network?
    - *input*:
      - a set of *real* images → from data distribution
      - and a set of *fake* images → output of G
    - ask to perform a *binary classification*:
      - real → 1, fake → 0
      - Sigmoid is used for binary classification
    - *output*: 0/1



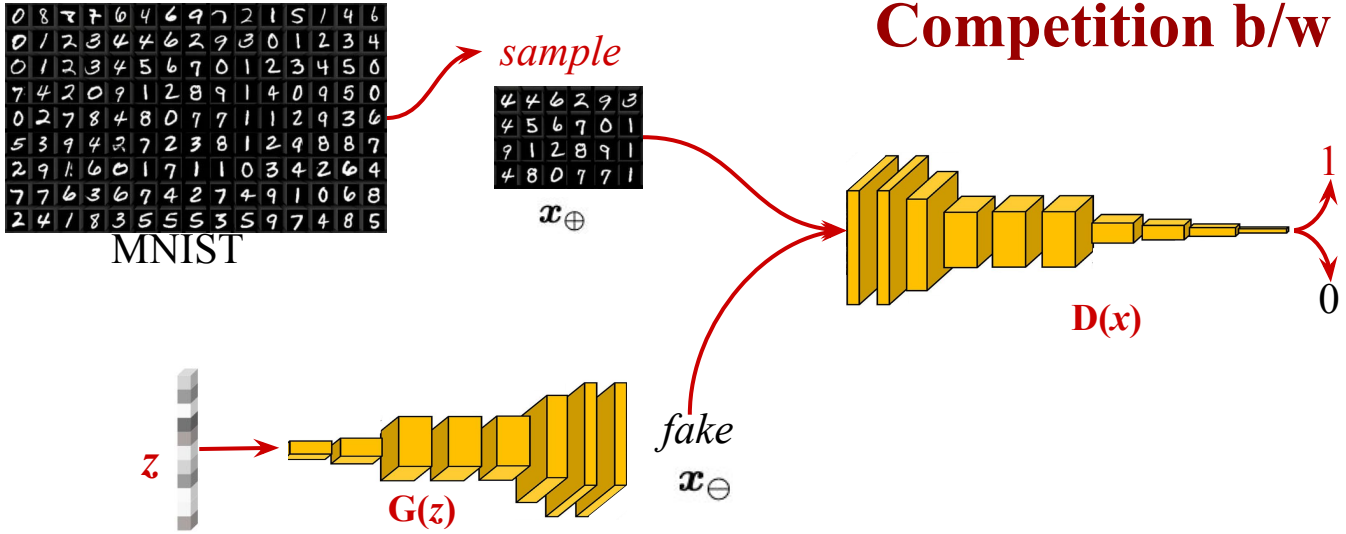
Competition b/w G and D

- **Goal of D:**→ to correctly classify real and fake images
- **Goal of G:**→ to fool the D
  - fool the D to classify a fake as real
- This has to be done simultaneously!

- If G can fool D
  - and we know that D is good enough
  - then, G understand the data distribution



Competition b/w G and D



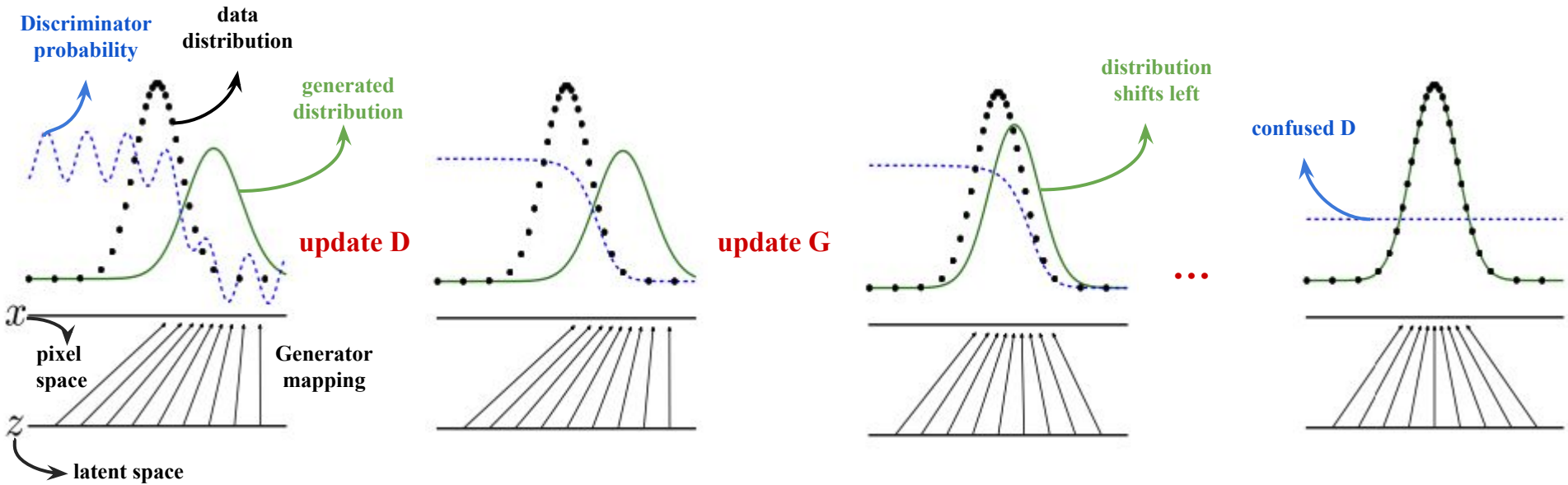
- If the generator does a good job
  - creates fake examples that looks like real
  - harms discriminator: **D(*x*–)** ↑
- **Dilemma**:
  - discriminator needs the generator to be good → to be able to learn from confusing fake examples
  - but, by definition a good generator promotes **D(*x*–)** ↑
    - harms the performance of the discriminator ⇒ ***2-player minimax game***

- If the discriminator does a good job
  - becomes good at discerning ***x***+ and ***x***–
  - **D(*x*+)** ↑ and,
  - harms generator: **D(*x*–)** ↓

GAN Optimization

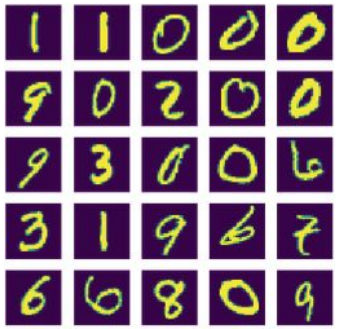
- Aim of GAN optimization is to achieve **Nash equilibrium**:
  - **D(x+) = D(x–) = 0.5**
  - discriminator can't tell real and fake apart!!
    - *meaning*: generator can generate images similar to real ones

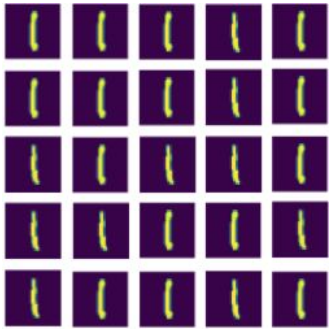
Nobel Prize 1994:
"for pioneering analysis of equilibria in the theory of non-cooperative games"

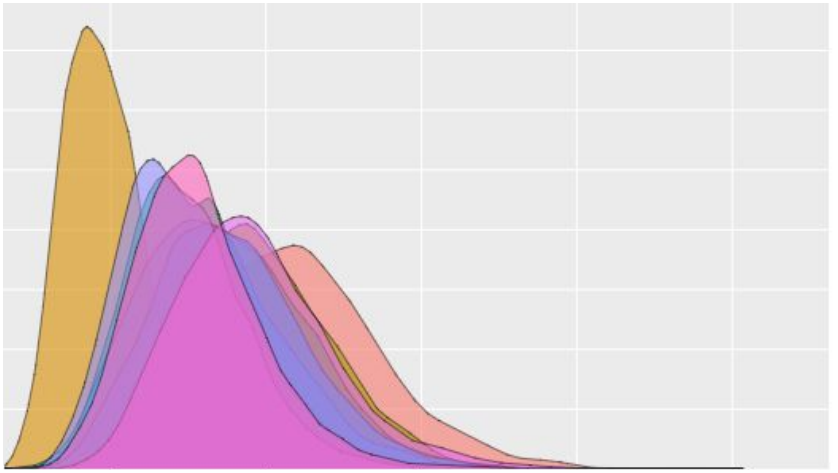


Mode Collapse

- Which set of generator images do you prefer?



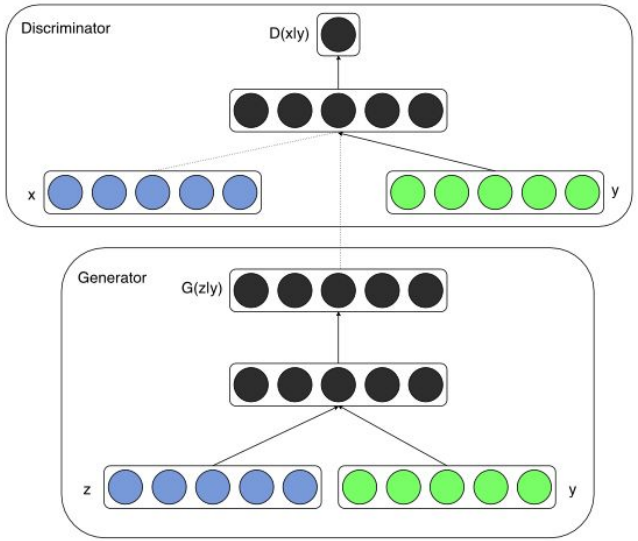


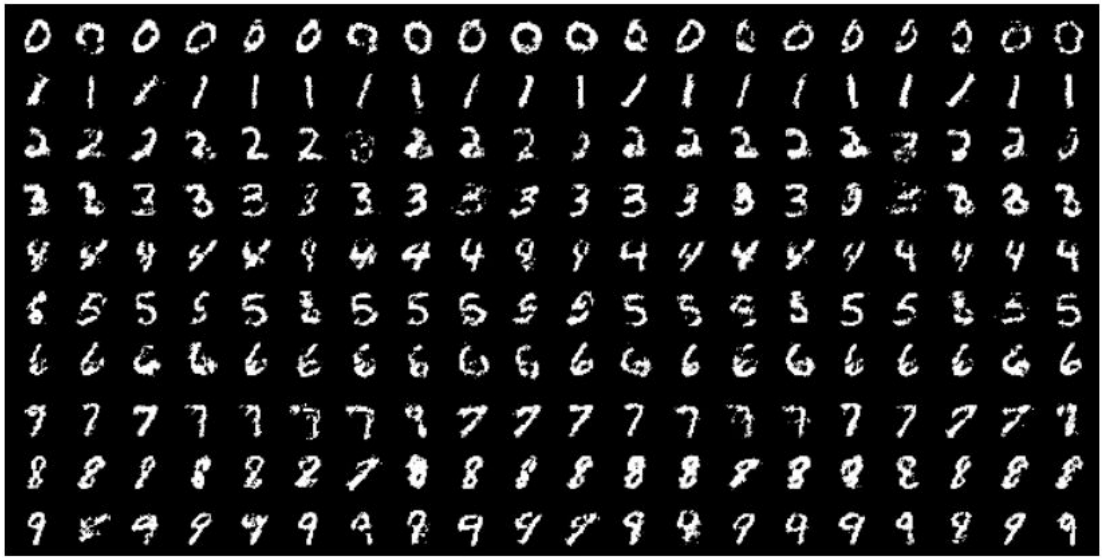


- A too much optimized generator can get stuck at a model of the generative model
  - it will generate real-looking fake images but mostly single class
  - initially D will classify it ‘real’
  - but then it realizes and catches it up, but the G can jump to another **mode** …
- GAN generates sharp realistic images
  - but these images do not represent the *true data distribution*
- Very careful optimization is needed to handle this

Conditional GAN

- Original GAN could *not be controlled* to generate images of choice
- We control the behavior of GAN with *conditional* information:
  - say, for MNIST → provide class information





Generated MNIST digit, each row conditioned on one label

Next …

- **Diffusion Models I**
  - Deep Unsupervised Learning Using Nonequilibrium Thermodynamics
  - Generative Modeling by Estimating Gradients of Data Distribution

{ **April 15** }

- **Diffusion Models II**
  - Denoising Diffusion Probabilistic Models
  - Cold Diffusion: Inverting Arbitrary Image Transforms Without Noise

{ **April 22** }

Thank you!

INDIANA UNIVERSITY BLOOMINGTON