

# Software Engineering: OOP illustrated through Density Estimating Neural Networks

**Albert Montillo**  
**UTSouthwestern**

Departments: Lyda Hill Department of Bioinformatics, Radiology, and Advanced Imaging Research Center



UNIVERSITY of PENNSYLVANIA



Rensselaer

Yale University

RUTGERS  
UNIVERSITY



GE Global Research

MGH/HST Athinoula A. Martinos  
Center for Biomedical Imaging



MASSACHUSETTS  
GENERAL HOSPITAL



Harvard-MIT  
Health Sciences & Technology



PENN  
Radiology  
University of Pennsylvania Health System

Microsoft  
Research  
COGNEX

SWE course

Lyda Hill Department of Bioinformatics

# Outline

## 1. Monday : Object Oriented Variational Autoencoders (VAEs)

1. Self study read the slides (Monday\_VAE..pptx) on your own
2. Start the exercise described on slides 47-54
3. Any concerns: email TAs and/or bring questions to Tuesday morning lecture

## 2. Tuesday

1. Ask your residual questions about VAEs after having attempted exercise Monday
2. New topic: Object Oriented Generative Adversarial Networks (GANs)
3. New topic: Symbolic debugger: cond breakpoints and call stack traversal

## 3. Wednesday

1. Review observations on VAEs and GANs
2. New topic: Object Oriented conditional VAEs (cVAE) and Auxillary Classifier GANs (AKA cGAN or acGAN)
3. New topic: Motivate a possible combination of cVAE and cGAN

## 4. Thursday

1. Review cVAE, cGAN observations
2. Review cVAE-cGAN code
3. New topic: Hyperparameter optimization
4. New topic: training curve and latent space traversal and visualization

# Welcome !

---

## 1. Welcome

1. Review the syllabus: `block3MontilloSyllabus.2024_v1.29`
2. Syllabus supersedes prior info.

## 2. Overview. This block is about OOP for research

## 3. Grading

1. Attempt all parts
2. Seek TA help during office hours.
3. Have fun.

## 4. Review OOP from weekend

1. Spaghetti code / goto → Procedural & functional → OOP
2. What is encapsulation?
3. What is inheritance? When should you use it?
4. What is polymorphism?
5. Who attempted the OOP basics?
6. Your questions?

# **... Welcome !**

---

## **5. Image IO**

**How are images stored on disk?**

**How are they stored in memory?**

**Image is an array**

**What is difference between gray scale and RGB image?**

**How would you compute the mean intensity in an image?**

**How would you compute the mean image of list of images?**

**How would you compute the std dev of pixel intensities?**

**Questions from exercises?**

# **... Welcome !**

---

## **6. Keras**

**See also Weekend\_OOP\_TensorflowKeras\_v1.\*.pdf**  
**A framework and API for building neural networks**

**Object Oriented**

**If you need access to the reference by Chollet, see the TAs.**

**What is a DFNN ?**

**What is a CNN?**

**What is the fundamental difference or innovation in a CNN?**

**Extra credit: what is a transformer?**

**Questions?**

# How much information is available for machine learning?

- Pure supervised learning (cherry)
- Semi-supervised learning (icing)
- Unsupervised learning (cake)



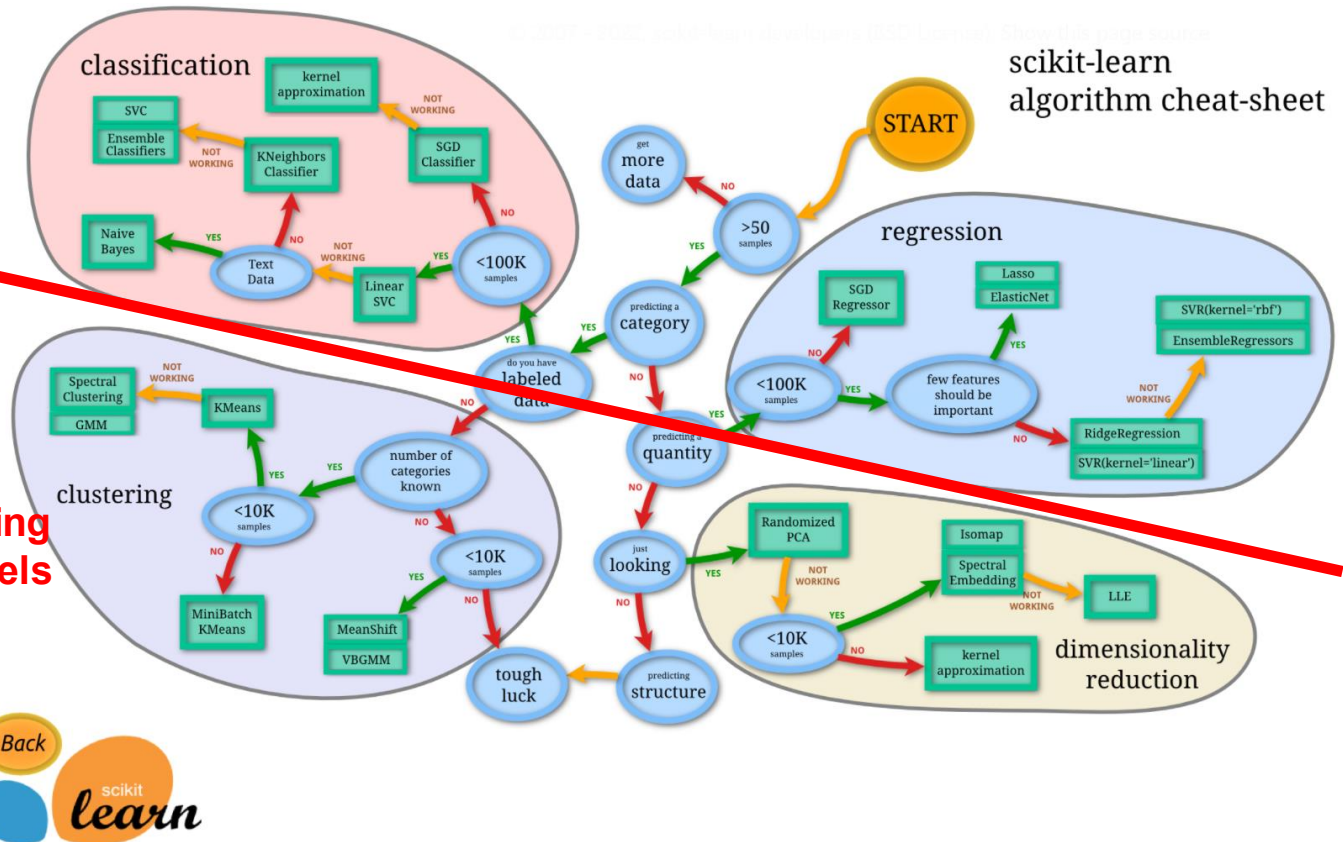
Concept adapted from Yann LeCun

# Taxonomy of machine learning algorithms

- Supervised algorithms (labeled)

- Unsupervised (unlabeled)

- density estimating generative models



# Taxonomy of Generative Models

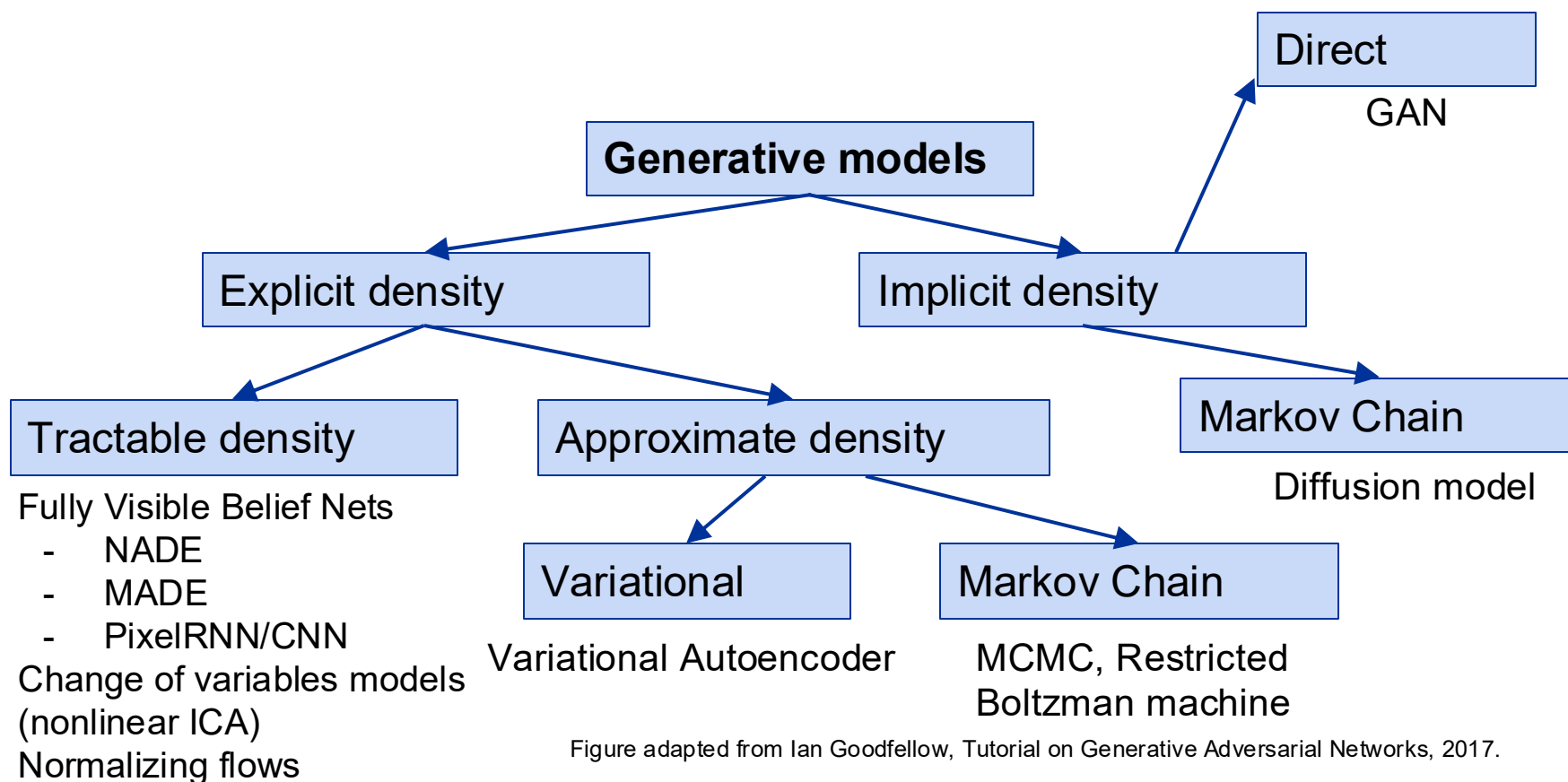


Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n



# Taxonomy of Generative Models

We will discuss 2 of the most popular types of generative models

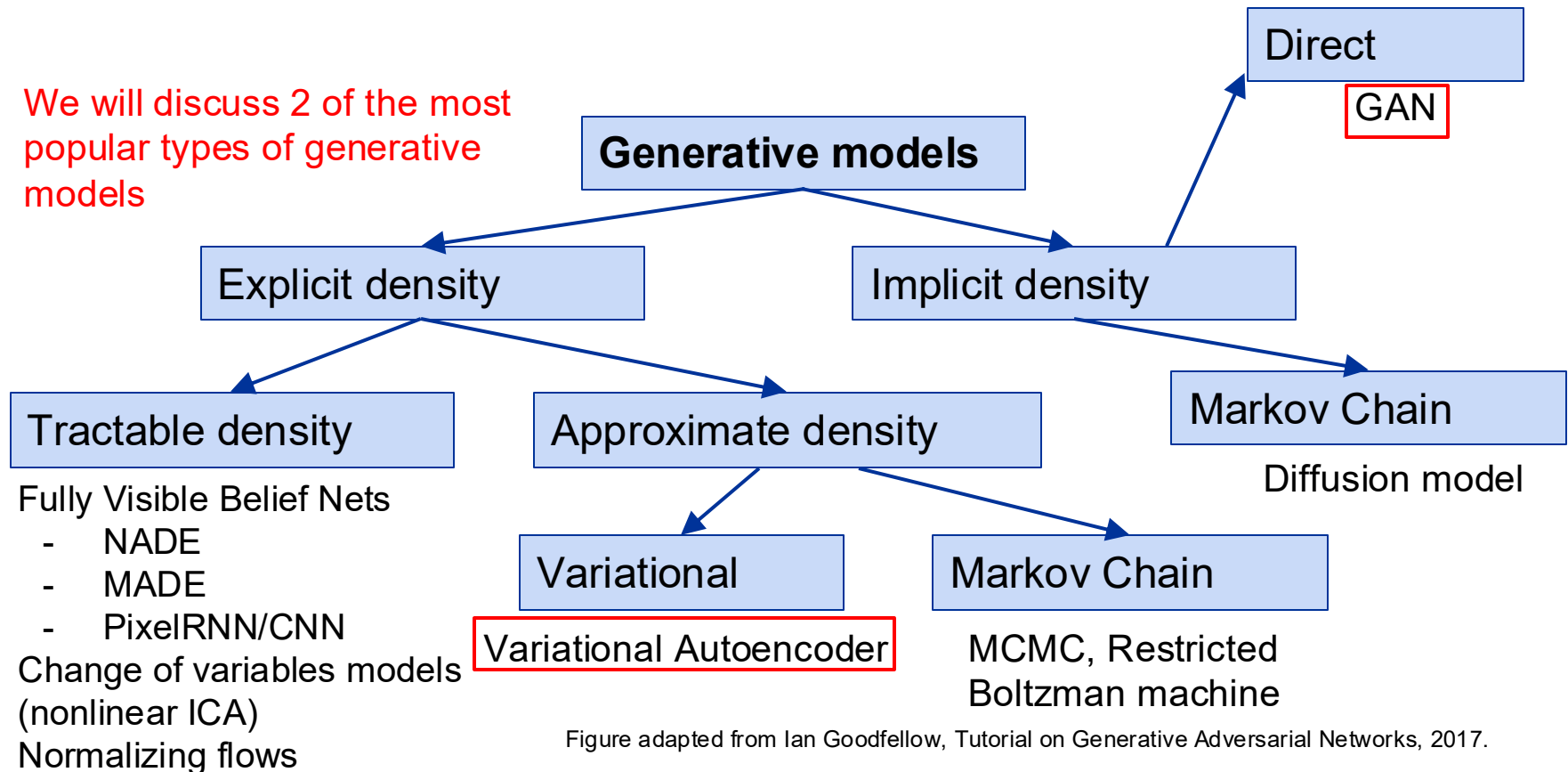
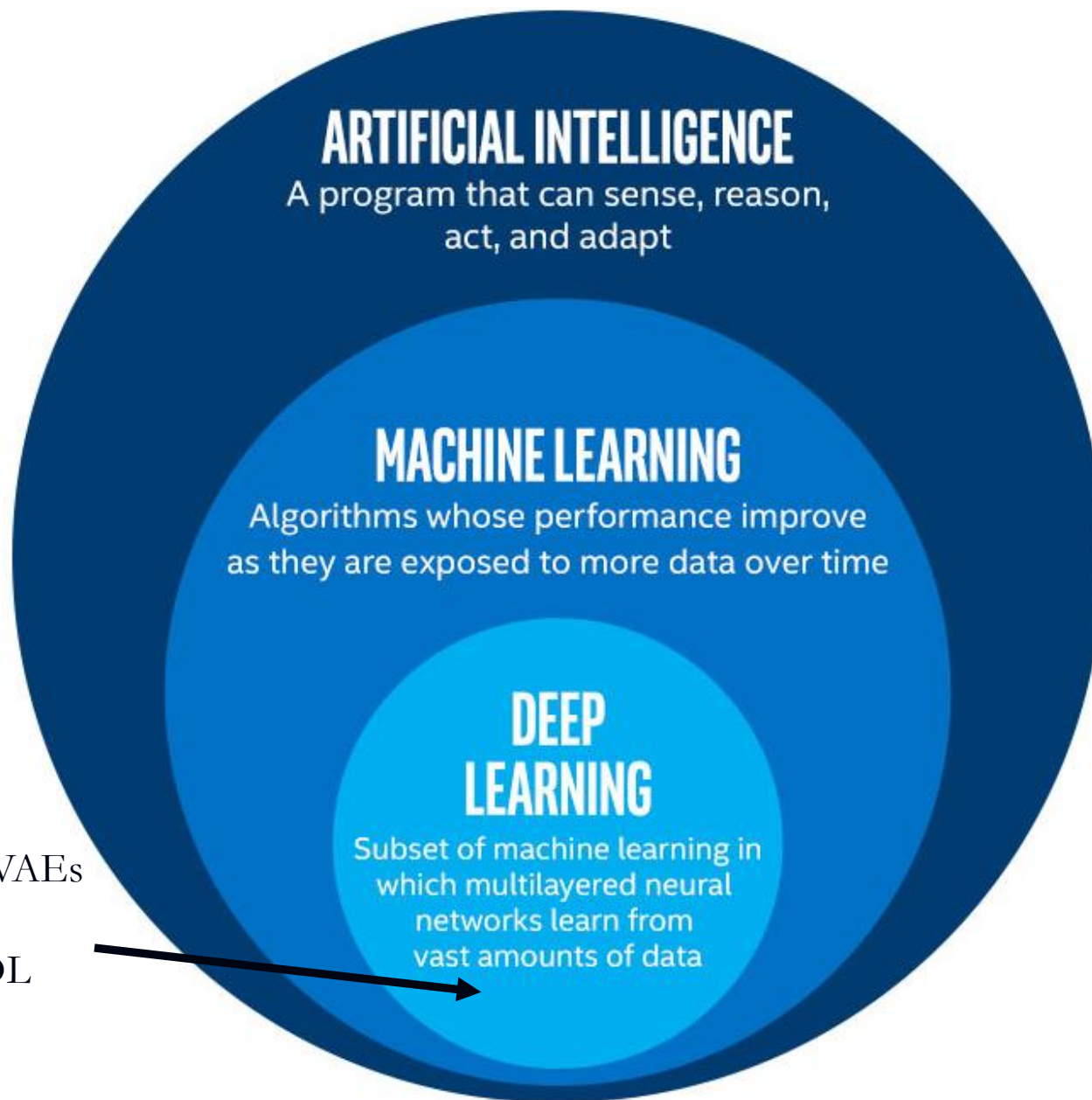


Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

# How do GANs and VAEs relate to AI?

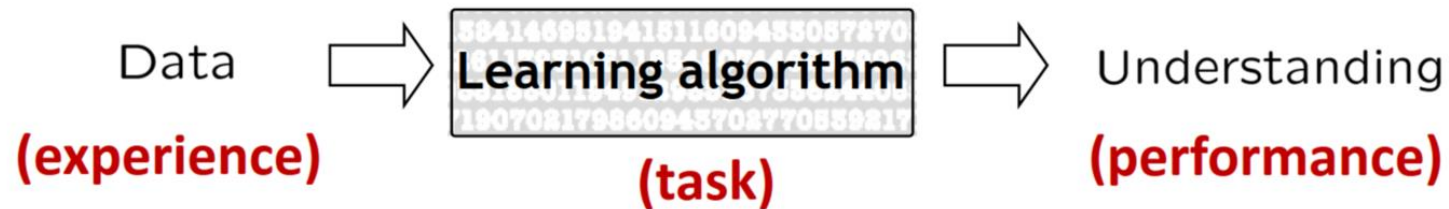


GANs and VAEs are types of generative DL models

# Core attribute of machine learning algorithms

Algorithms whose

- performance improves
- at some task
- with increasing experience

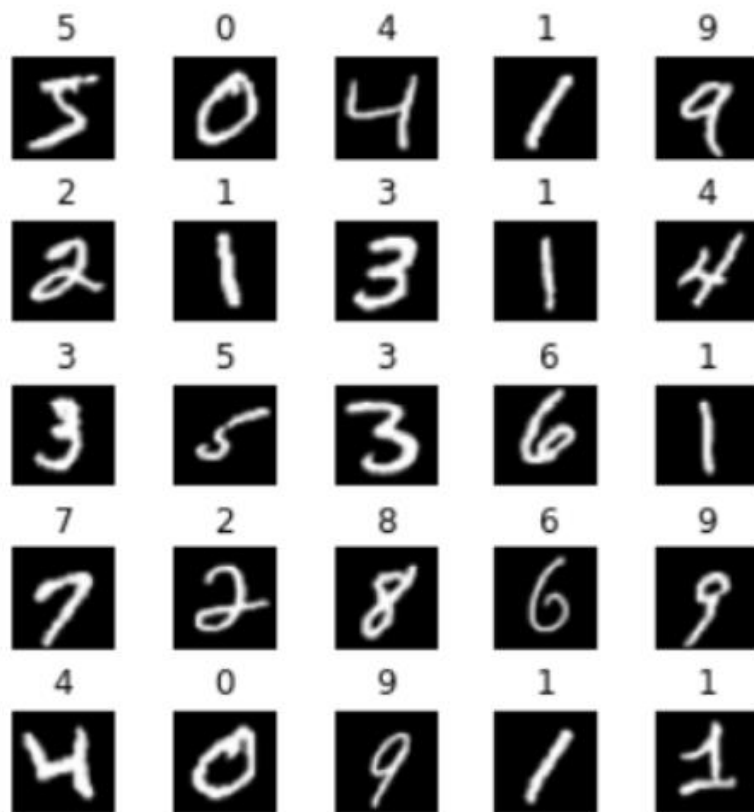


# Datasets

---

1. **MNIST dataset**
2. **Alzheimer's dataset**

1. 60,000 grayscale images of handwritten digits, from the *Modified National Institute of Standards and Technology* database
2. Example images



Why might digit recognition be hard for a computer?

What variations would a distribution of “1”s contain?

Is the distribution of 5’s more or less complex than 1s?

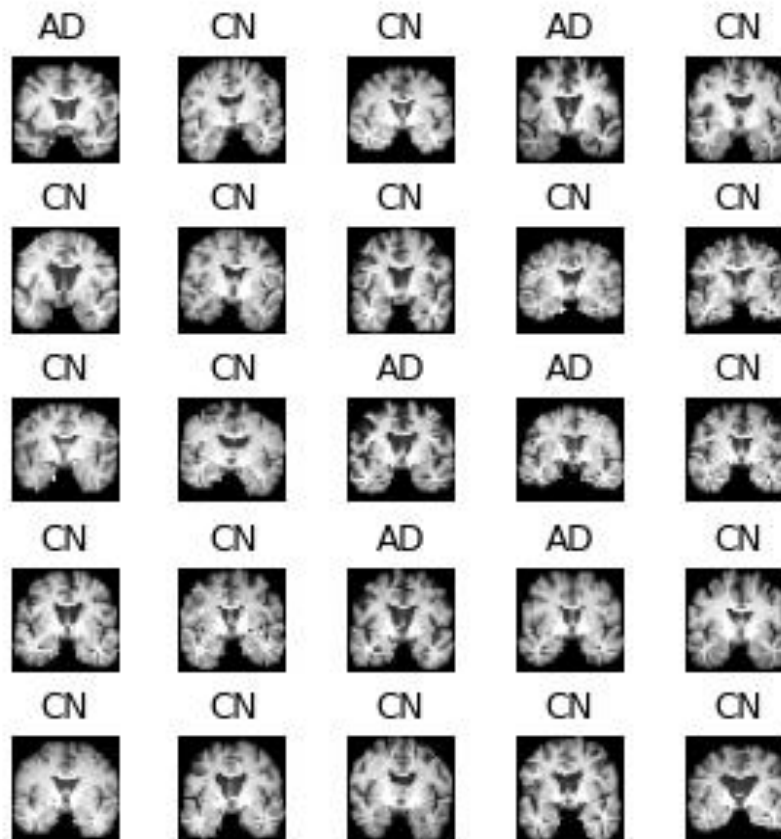
What are models that could learn such distributions?

What are models that could learn the distributions and classify digits?

# Alzheimer's Disease

## 1. T1w MRI of cognitively normal and Alzheimer's Disease patients

## 2. Example images



AD ... Alzheimer's diseased  
CN ... cognitively normal

Why might AD diagnosis be difficult for a computer?

What variations would the distribution of CN ridden brain images contain?

What variations would the distribution of AD ridden brain images contain?

What are models that could learn the distributions and provide a diagnosis recommendation?

# Evolution of autoencoders

---

1. Autoencoders
2. Variational autoencoders

**Learning a compressed representation.**

- **Ease of communication, further computation and predictions**

**There are many methods for compression**

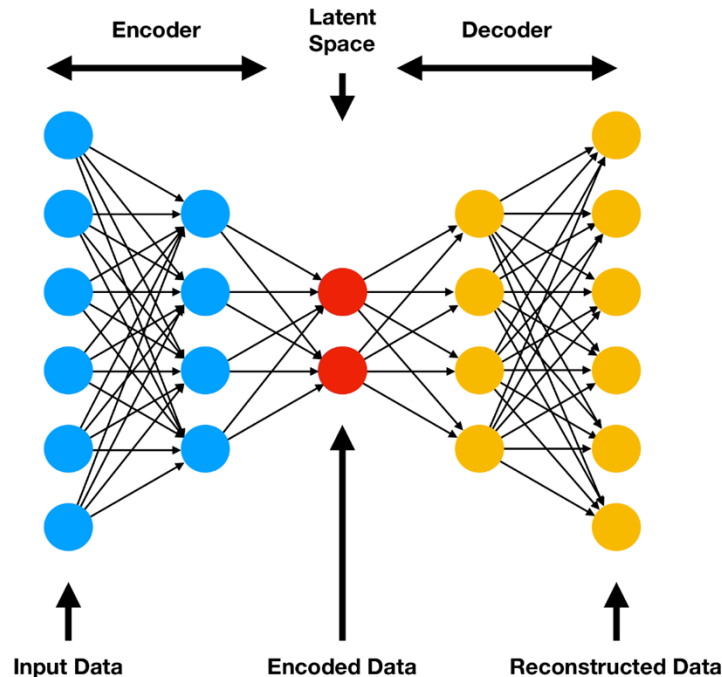
- **Some are neural network based**
- **Prime example is autoencoder (AE)**



# vector Autoencoders (AEs)

## Autoencoders

- Vector in, bottle neck, same vector out
- Architecture is encoder, bottle neck, decoder.

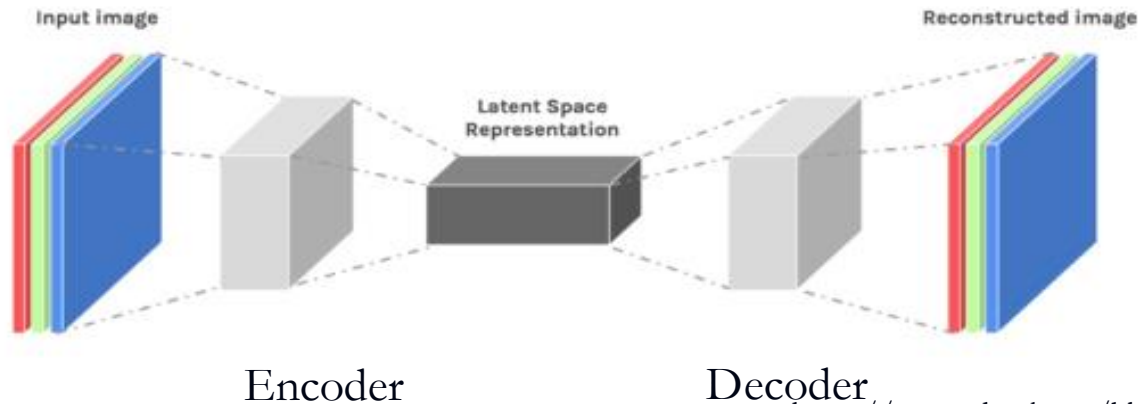


- You control the amount of compression via width of bottle neck
- What happens when you vary the width? (narrower--> fuzzier recons).

<https://www.compthree.com/blog/autoencoder/>

## image Autoencoders (AEs)

Same architecture essentially for images, replace fully connected layers with convolutional layers



<https://www.edureka.co/blog/autoencoders-tutorial/>

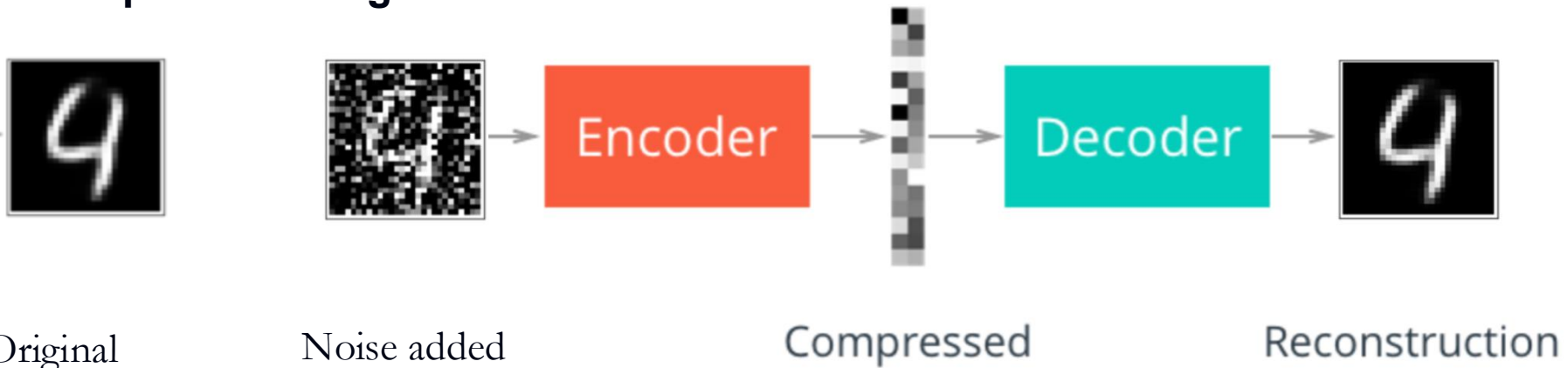
**Why do that?**

- 1) Learn the dist of images
- 2) Learn compact representations, easier to explore that space to learn more about your data.

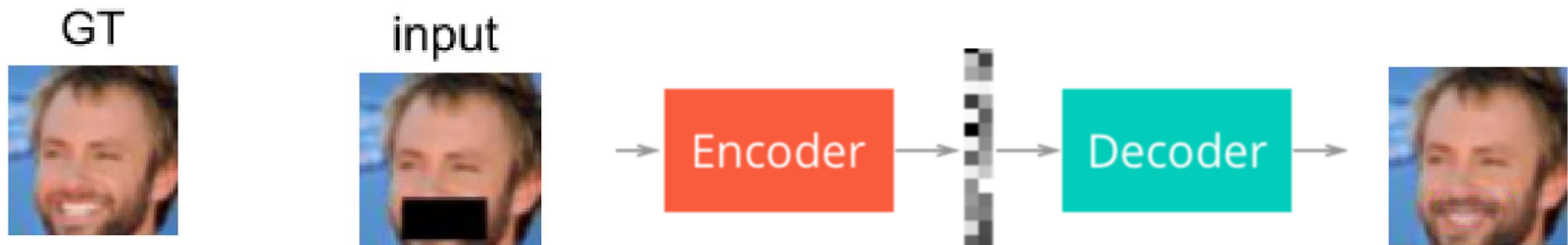
## Additional reasons for Autoencoders (AEs)

With clever training not only can AE be used for compression & data understanding but also learn to suppress noise &/or artifact, and neural in-painting.

### Example denoising AE



### Example: in-painting AE (a type of self-supervised learning)



## AE limitations and VAE solution

**Drawback of AE is that throws away any uncertainty it may have.**

- **Consider the loss of information at the bottle neck layer which is a point estimate**
- **Perhaps there are several roughly equally good bottle neck representations.**
- **Plain AE is forced to choose one of them, even though there was likely a distribution of them that were roughly equally likely.**

**Solution: the Variational AE**

**Goal: learn a distribution of compressed representations rather than just one point estimate**

**Benefits:**

- **Retain uncertainty. Less loss of information.**
- **You can project a given sample to a dist and then sample from it, decode them to form a variety of reconstructions.**
- **e.g., perhaps the in-painting could be solved, noise could be suppressed this way or that way..**
- **Let the user decide which they prefer. Or use them all in a movie**
- **Generally, flexibility is never a bad thing. Often quite practical.**

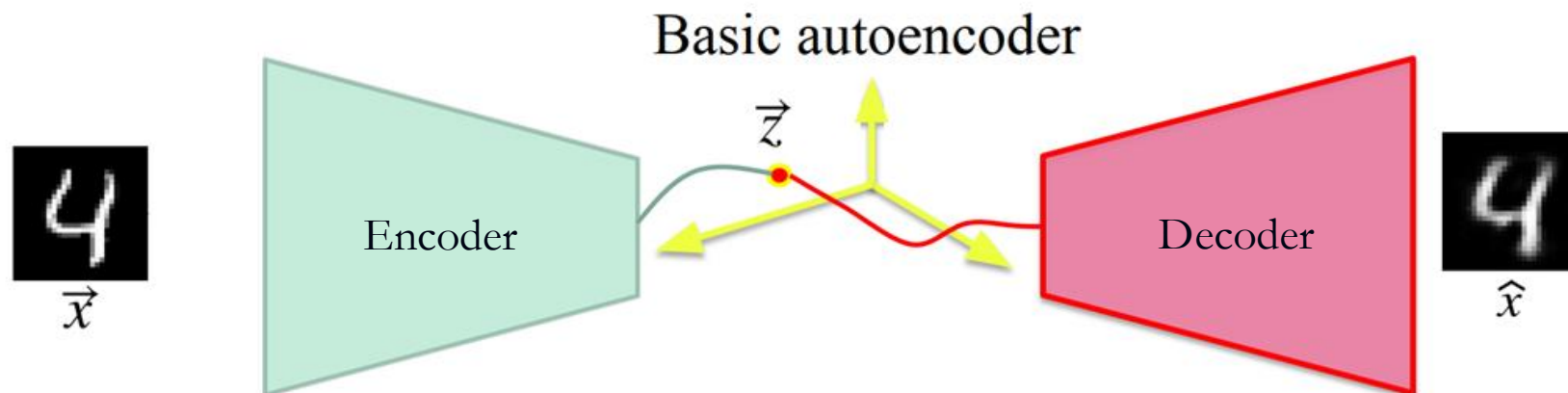
## VAE is also a type of generative model

**Compare learning a decision boundary (with a discriminative model) vs learning a dist of data that you can sample from (with a generative model).**

**Which is preferable? Does one learn more? Why?**

- **Learning a decision boundary** in the feature space is helpful to classify data points into categories. Discriminative models: SVMs . Doesn't provide much insight into your data. Requires Labeled data
- **Learning a distribution of data** captures underlying inherent structure in the data itself. Often more complex than discriminative model training, but allows further insight into data and enable tasks beyond classification (e.g. generating new samples). Generative models: GMMs, VAEs, GANS. Can be helpful when much of your data is unlabeled

# Comparison AEs and Variational autoencoders (VAEs)

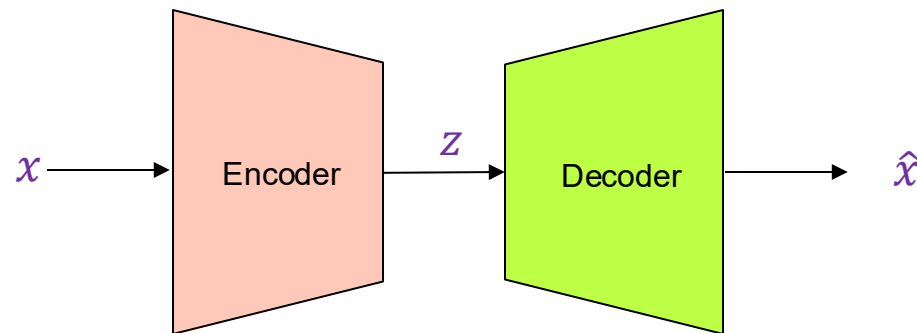


Jodoin, Deep learning for medical imaging school 2021

# Variational autoencoders: High-level idea

---

- At training time, jointly learn *encoder* and *decoder* by maximizing a variational bound on the data likelihood

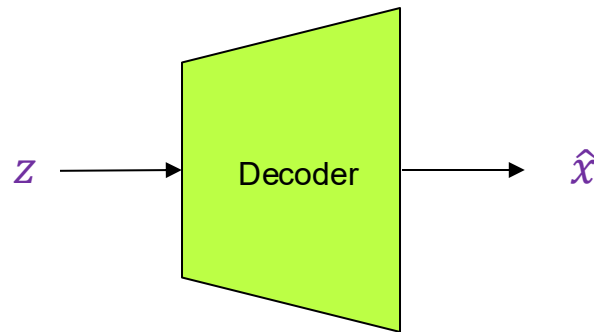


D. Kingma and M. Welling, [Auto-Encoding Variational Bayes](#), ICLR 2014

# Variational autoencoders: High-level idea

---

- At training time, jointly learn *encoder* and *decoder* by maximizing a variational bound on the data likelihood
- At test time, discard encoder and use decoder to sample from the learned distribution

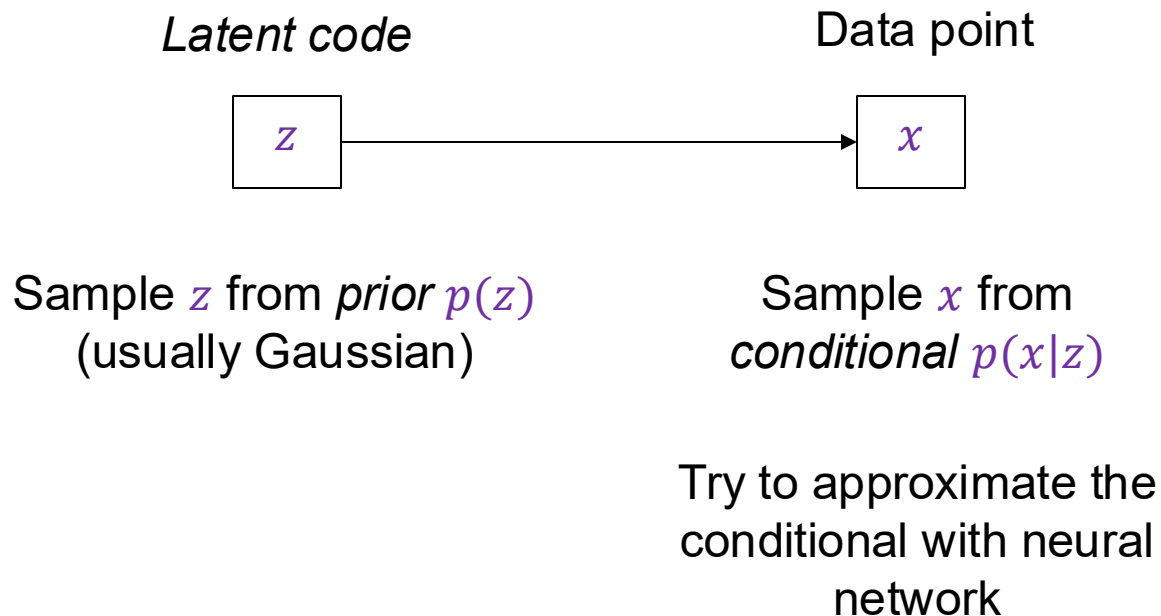




# Variational autoencoders: High-level idea

---

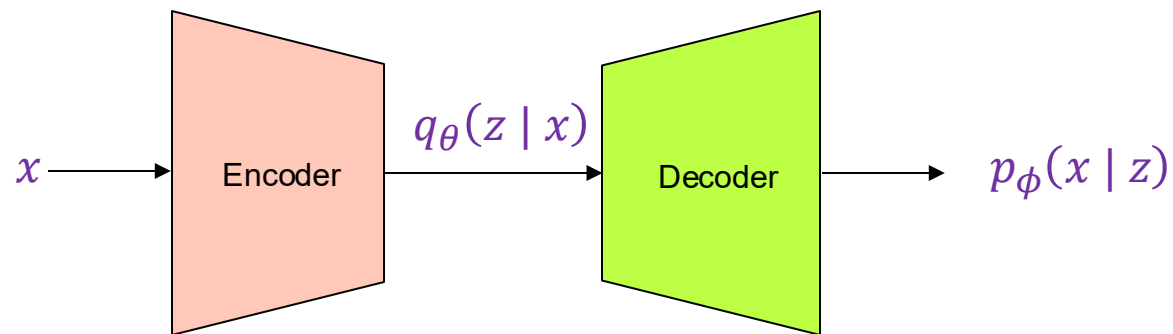
- Probabilistic generative model of the data distribution:



# Variational autoencoders: High-level idea

---

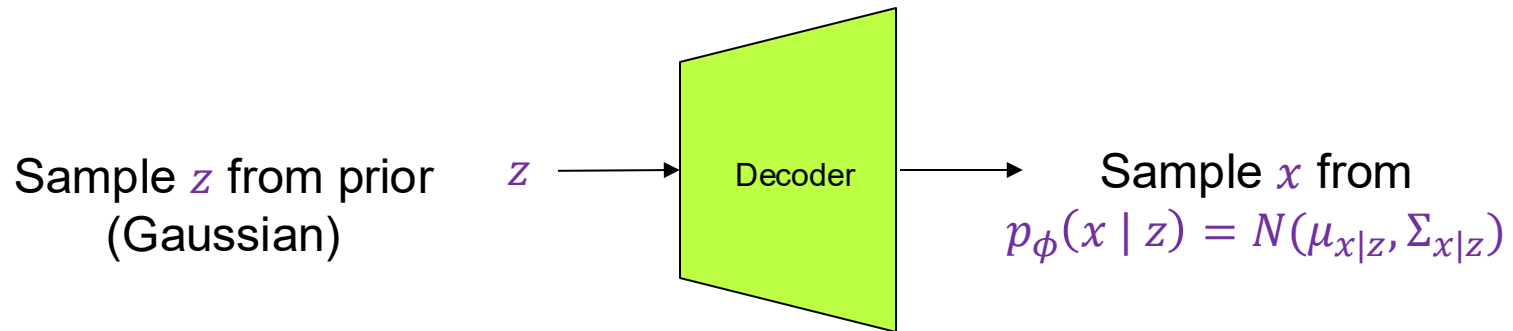
- At training time, jointly learn *encoder* and *decoder*
- **Encoder:** given inputs  $x$ , output  $q_{\theta}(z | x)$ 
  - Specifically, output mean and (diagonal) covariance, or  $\mu_{z|x}$  and  $\Sigma_{z|x}$ , so that  $q_{\theta}(z | x) = N(\mu_{z|x}, \Sigma_{z|x})$
- **Decoder:** given  $z$ , output  $p_{\phi}(x | z)$ 
  - Specifically, output  $\mu_{x|z}$  and  $\Sigma_{x|z}$  so that  $p_{\phi}(x | z) = N(\mu_{x|z}, \Sigma_{x|z})$
- **Training objective:** (a bound on) data likelihood under the model



# Variational autoencoders: High-level idea

---

- At test time, discard encoder and use decoder to sample from  $p_\phi(x | z) = N(\mu_{x|z}, \Sigma_{x|z})$



# Variational autoencoders: Training

---

- Objective: maximize the *variational lower bound* on the data likelihood:

$$\log p_{\phi}(x) \geq \mathbb{E}_{z \sim q_{\theta}(z|x)}[\log p_{\phi}(x|z)] - D_{KL}(q_{\theta}(z|x), p(z))$$

# Variational autoencoders: Training

---

- Objective: maximize the *variational lower bound* on the data likelihood:

$$\log p_{\phi}(x) \geq \mathbb{E}_{z \sim q_{\theta}(z|x)} [\log p_{\phi}(x|z)] - D_{KL}(q_{\theta}(z|x), p(z))$$

1. Run training point  $x$  through encoder to get a distribution over latent codes  $z$

# Variational autoencoders: Training

---

- Objective: maximize the *variational lower bound* on the data likelihood:

$$\log p_{\phi}(x) \geq \mathbb{E}_{z \sim q_{\theta}(z|x)} [\log p_{\phi}(x|z)] - D_{KL}(q_{\theta}(z|x), p(z))$$

1. Run training point  $x$  through encoder to get a distribution over latent codes  $z$
2. Encoder output should match the prior  $p(z)$ 
  - Closed form expression when  $q_{\theta}$  is diagonal Gaussian and  $p$  is unit Gaussian (Assume  $z$  has dimension  $J$ ):

$$-D_{KL}(q_{\theta}(z|x), p(z)) = \sum_{j=1}^J \left( 1 + \log \left( (\Sigma_{z|x})_j^2 \right) - (\mu_{z|x})_j^2 - (\Sigma_{z|x})_j^2 \right)$$

# Variational autoencoders: Training

---

- Objective: maximize the *variational lower bound* on the data likelihood:

$$\log p_{\phi}(x) \geq \mathbb{E}_{z \sim q_{\theta}(z|x)} [\log p_{\phi}(x|z)] - D_{KL}(q_{\theta}(z|x), p(z))$$

1. Run training point  $x$  through encoder to get a distribution over latent codes  $z$
2. Encoder output should match the prior  $p(z)$
3. Sample code  $z$  from encoder output

# Variational autoencoders: Training

---

- Objective: maximize the *variational lower bound* on the data likelihood:

$$\log p_{\phi}(x) \geq \mathbb{E}_{z \sim q_{\theta}(z|x)} [\log p_{\phi}(x|z)] - D_{KL}(q_{\theta}(z|x), p(z))$$

1. Run training point  $x$  through encoder to get a distribution over latent codes  $z$
2. Encoder output should match the prior  $p(z)$
3. Sample code  $z$  from encoder output
4. Run sampled  $z$  through decoder to get a distribution over data samples



# Variational autoencoders: Training

---

- Objective: maximize the *variational lower bound* on the data likelihood:

$$\log p_{\phi}(x) \geq \mathbb{E}_{z \sim q_{\theta}(z|x)} [\log p_{\phi}(x|z)] - D_{KL}(q_{\theta}(z|x), p(z))$$

1. Run training point  $x$  through encoder to get a distribution over latent codes  $z$
2. Encoder output should match the prior  $p(z)$
3. Sample code  $z$  from encoder output
4. Run sampled  $z$  through decoder to get a distribution over data samples
5. Original input should be likely under the distribution output in (4)

# Variational autoencoders: Training

---

- Objective: maximize the *variational lower bound* on the data likelihood:

$$\log p_{\phi}(x) \geq \underbrace{\mathbb{E}_{z \sim q_{\theta}(z|x)}[\log p_{\phi}(x|z)]}_{\text{Data likelihood}} - \underbrace{D_{KL}(q_{\theta}(z|x), p(z))}_{\text{Regularization}}$$

# Variational autoencoders: Training

---

- Objective: maximize the *variational lower bound* on the data likelihood:

$$\log p_{\phi}(x) \geq \underbrace{\mathbb{E}_{z \sim q_{\theta}(z|x)}[\log p_{\phi}(x|z)]}_{\text{Data likelihood}} - \underbrace{D_{KL}(q_{\theta}(z|x), p(z))}_{\text{Regularization}}$$

- Objective for the entire dataset:

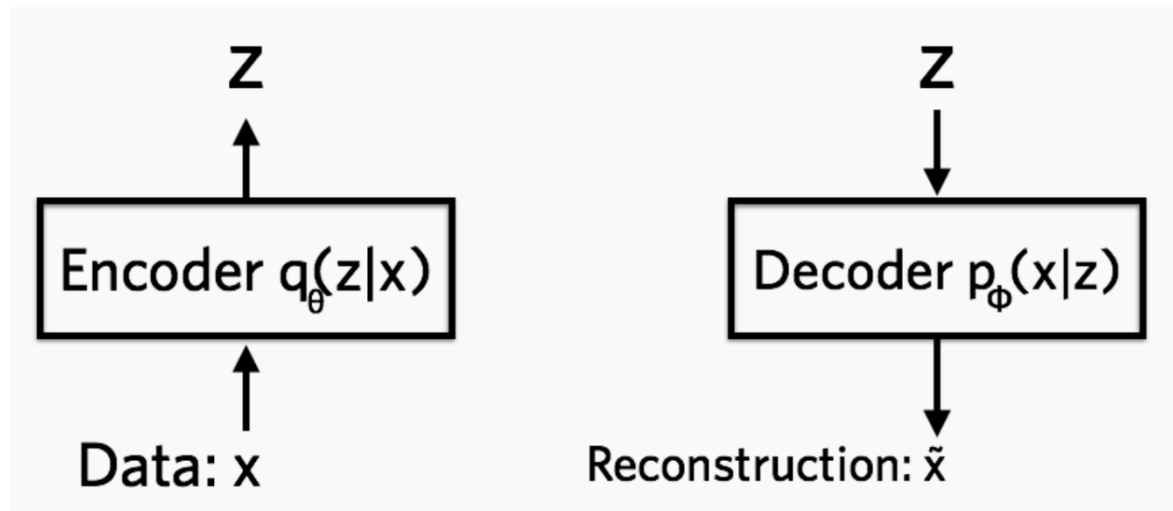
$$\mathbb{E}_{x \sim D} [\mathbb{E}_{z \sim q_{\theta}(z|x)}[\log p_{\phi}(x|z)] - D_{KL}(q_{\theta}(z|x), p(z))]$$

For further details, see: C. Doersch, [Tutorial on Variational Autoencoders](#), 2016

# Variational Autoencoder (VAE)

**Key idea:** make both the encoder and the decoder probabilistic.

**I.e., the latent variables,  $z$ , are drawn from a probability distribution depending on the input,  $X$ , and the reconstruction is chosen probabilistically from  $z$ .**



<https://jaan.io/what-is-variational-autoencoder-vae-tutorial/>

## VAE Encoder

The encoder takes input and returns parameters for a probability density (e.g., Gaussian): i.e.,  $q_{\theta}(z | x)$  gives the mean and covariance matrix (multivar Gauss). We can draw from this distribution to get a representation, a point  $z$ , in the lower-dimensional space.

Implemented via a neural network: each input  $x$  gives a vector mean and diagonal covariance matrix that determine the Gaussian density

**Parameters  $\theta$  for the encoder network need to be learned – therefore we need to set up a loss function to guide its learning.**

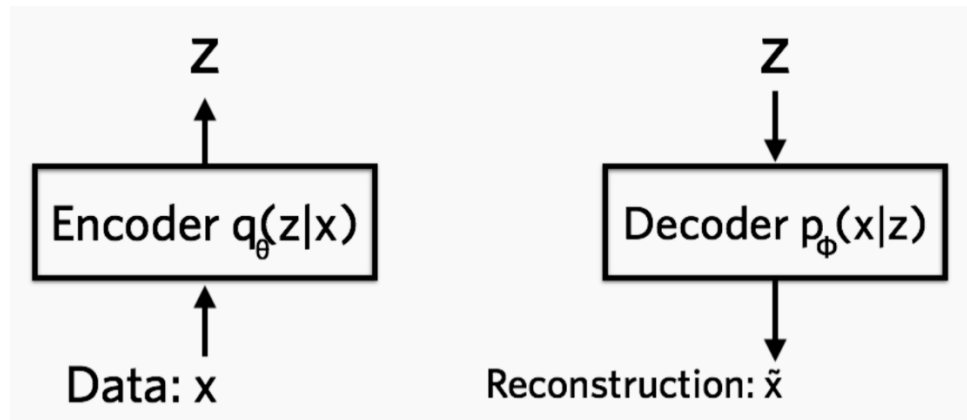
<https://jaan.io/what-is-variational-autoencoder-vae-tutorial/>

## Now let's talk about the VAE Decoder

The *decoder* takes latent variable  $z$  and returns parameters for a distribution. E.g.,  $p_{\phi}(x|z)$  gives the mean and variance for *each pixel* in the output.

Reconstruction  $\tilde{x}$  is produced by sampling from that distribution.

Implemented via neural network, i.e. the parameters  $\phi$  of the decoder network are also learned, through setup of a suitable loss.



<https://jaan.io/what-is-variational-autoencoder-vae-tutorial/>

## VAE loss function

What is the loss function for an autoencoder?  $L_2$  distance between output (reconstruction) and input (original) is the data fidelity term of the loss.

For VAE, we need to learn parameters of two probability dists. For a single input,  $x_i$ , we maximize the expected value of returning  $x_i$  or equivalently, minimize the expected negative log likelihood:

$$-\mathbb{E}_{z \sim q_{\theta}(z|x_i)}[\log p_{\phi}(x_i | z)]$$

This takes expected value wrt  $z$  over the encoder distribution  $q_{\theta}(z|x_i)$  of the loss  $-\log p_{\phi}(x_i|z)$

We find the dist parameters that make returning the original image most likely.

This data fidelity loss term ensures that combination of the probabilistic encoder/decoder produce a reconstruction similar to the input, i.e., that  $\tilde{x}_i \approx x_i$

<https://jaan.io/what-is-variational-autoencoder-vae-tutorial/>

## VAE loss function

**Problem:** **the weights may adjust to memorize input images via  $z$ .** *i.e.*, inputs that we regard as similar, may end up with very different distributions in  $z$  space.

**We prefer continuous latent representations to give meaningful parameterizations.** e.g., smooth changes from one digit to another, without any dead zones in between.

**Solution:** Encourage  $q_{\theta}(z|x_i)$  to be close to a standard normal (or some other simple density). This will ensure training images are mapped to proximal distributions.

**How do we encourage an additional constraint?**

By adding an additional term to our loss function (objective function)

<https://jaan.io/what-is-variational-autoencoder-vae-tutorial/>



## VAE loss function

For a single sample  $x_i$  we get the loss function:

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)} [\log p_\phi(x_i | z)] + \text{KL}(q_\theta(z | x_i) || p(z))$$

The first term promotes an encoding/decoding that contains sufficient information to enable recovery of the input. This is the data fidelity term that we saw earlier.

The second term keeps the encoding continuous – the encoding dist is compared to a fixed prior,  $p(z)$ , regardless of the input, which inhibits memorization (overfitting the data which prevents generalization).

You get to choose the form of  $p(z)$ . **What might be a common choice?**

**Multivariate Standard Gaussian**

With this loss function the VAE can (**almost**) be trained using gradient descent on minibatches (subsets of the training data/grad desc. step).

We just need to determine how to implement the two loss terms in our loss function.

## Implementing the second term

There is a closed form soln to KL divergence for the std normal dist, that you need to implement in the exercises:

$$-\text{KL}((q_{\phi}(\mathbf{z})||p_{\theta}(\mathbf{z}))) = \int q_{\theta}(\mathbf{z}) (\log p_{\theta}(\mathbf{z}) - \log q_{\theta}(\mathbf{z})) d\mathbf{z}$$
$$= \frac{1}{2} \sum_{j=1}^J (1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2)$$

Note that this is the expression for the negative KL. In the exercises you need to implement the KL.

Also this form is for a single sample. In the exercises you need to implement the average KL across all samples.

The code uses

- `dist_mean` ... Distribution means .. A tensor of containing  $\mu_j$  in the  $j$ th element
- `dist_logvar` ... Distribution log-variances. tensor containing  $(\sigma_j)^2$  in the  $j$ th element

Kingma and Welling 2014, ICLR, Auto-encoding Variational Bayes

## Implementing the first term

For a single data point  $x_i$  we get the loss function

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\theta(z|x_i)} [\log p_\phi(x_i | z)] + \text{KL}(q_\theta(z | x_i) || p(z))$$

**Problem:** The expectation would usually be approximated by drawing samples and then averaging.

**However, this is not differentiable wrt  $\theta$  and  $\phi$  ... and we need to differentiate to use the back-propagation based gradient descent optimization of DL.**

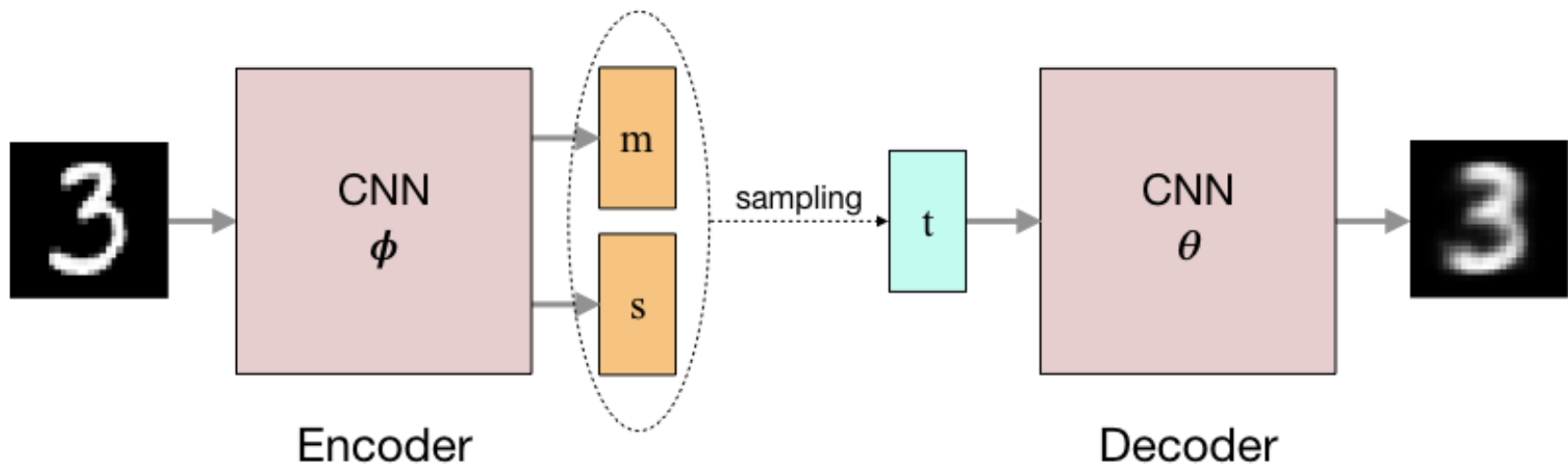
<https://jaan.io/what-is-variational-autoencoder-vae-tutorial/>

## ... Implementing the first term

**Problem:** The expectation would usually be approximated by drawing samples and averaging. This is not differentiable wrt  $\theta$  and  $\phi$ .

**For example, for images a (convolutional) VAE's encoder is a CNN that learns the means ( $m$ ) and log variances ( $s$ ) of the distn in  $z$ -space.**

**The CNN decoder, requires drawing from that learned distn, to reconstruct the image.**



[https://nbviewer.jupyter.org/github/krasserm/bayesian-machine-learning/blob/master/variational\\_autoencoder.ipynb](https://nbviewer.jupyter.org/github/krasserm/bayesian-machine-learning/blob/master/variational_autoencoder.ipynb)

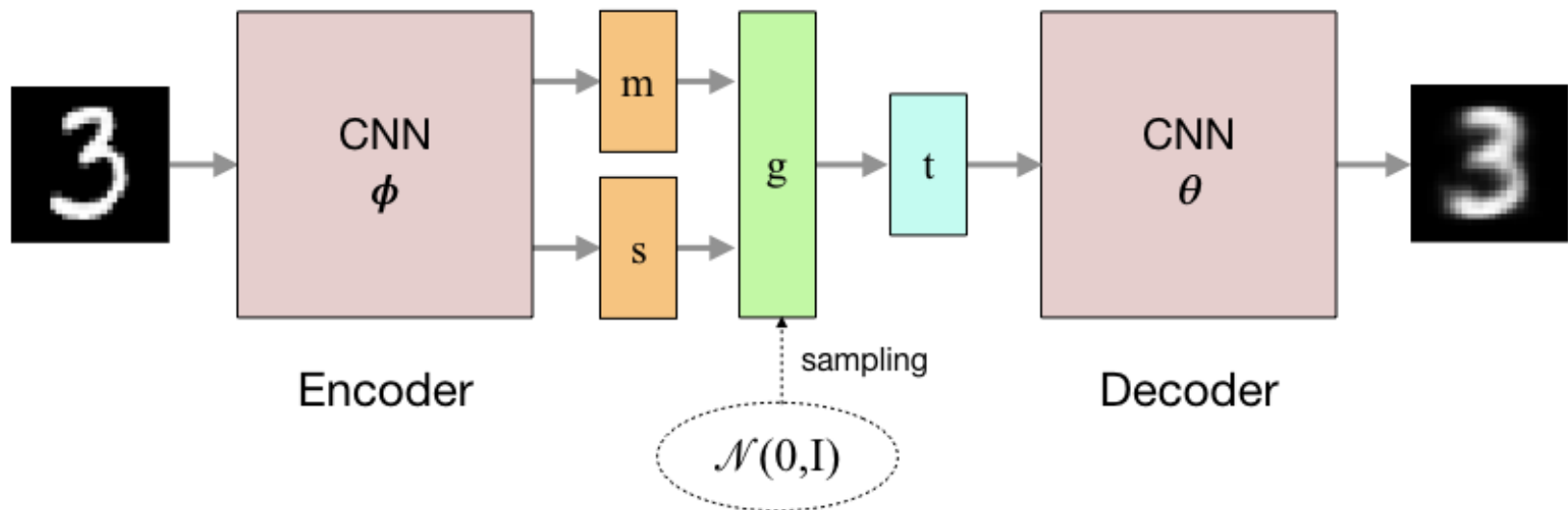
## ... Implementing the first term

Solution is called the **reparameterization trick**:

If  $\mathbf{z} \sim N(\mu(x_i), \Sigma(x_i))$ , then we can sample from  $\mathbf{z}$  using

$$\mathbf{z} = \mu(x_i) + \sqrt{\Sigma(x_i)} \epsilon,$$

where  $\epsilon \sim N(0,1)$ . So we can draw samples from  $N(0,1)$ , which doesn't depend on the parameters.



[https://nbviewer.jupyter.org/github/krasserm/bayesian-machine-learning/blob/master/variational\\_autoencoder.ipynb](https://nbviewer.jupyter.org/github/krasserm/bayesian-machine-learning/blob/master/variational_autoencoder.ipynb)

## Using the fully trained VAE model

After training,  $q_{\theta}(z|x_i)$  is close to a standard normal,  $N(0,1)$ , which is easy to sample.

Using a sample of  $z$  from  $q_{\theta}(z|x_i)$  as input to sample from  $p_{\phi}(x|z)$  gives an approximate reconstruction of  $x_i$ , at least in expectation.

If we sample any  $z$  from  $N(0,1)$  and use it as input to sample from  $p_{\phi}(x|z)$  then we can approximate the entire data distribution  $p(x)$ . That is, we can generate new samples that look like the input but aren't in the input.

We can also walk along the dimensions of  $Z$ , (e.g., pick the first two elements in  $Z$  and) and display the reconstructions as we go (in a discretized 2D Cartesian plane), to visualize the learned meaning of the dimensions of  $Z$  space.

# Exercise on object oriented VAEs

47

# Intro to the VAE code

- 1) `train_vae_mnist.ipynb` caller code
- 2) `vae.py` ...partially implemented VAE class.

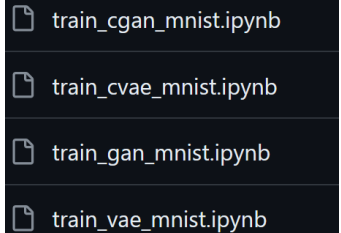
# Intro to the VAE exercise



# Walk through of the VAE Exercise

## 1. Introduce the VAE code:

1. Note: HINTS\_SWE.ipynb contains helpful hints for this course.
2. Walk through the structure of *caller programs* (jupyter notebooks) there is on for each type of model:



```
train_cgan_mnist.ipynb
train_cvae_mnist.ipynb
train_gan_mnist.ipynb
train_vae_mnist.ipynb
```

3. Walk through today's caller program: `train_vae_mnist.ipynb`
  1. It uses MNIST data as the training set.
  2. Purpose is to apply the VAE class you complete to estimate the density of MNIST images and be able to reconstruct new digit images.
  3. It outputs reconstructed results throughout and at the end of training here:  
`<YourRepoFolder>/outputs/mnist_vae`  
Folder contains reconstructed real digits and synthesized digits (next slides)
4. Class hierarchy (illustrating inheritance) is in `/vaegan` folder:
  1. Walk through `/vaegan/vae.py`
    1. Overall structure, use code folding using these keyboard shortcuts: (Ctrl+K,J and Ctrl+K,1 or 2)
    2. IDE: Use VSCode to edit `.py` use Browser to edit Jupyter

# ... Walk through of the VAE Exercise

## 2. Walk through exercise 3 in Syllabus (Word doc)

## 3. Walk through the code structure in vae.py

```
def sample_from_normal()
# Sample a point from a normal distribution, parameterized by a given mean and log-variance.

def kl_divergence()
# Compute the closed-form KL Divergence between a given distribution and a normal prior
distribution with mean 0 and variance 1.

class Encoder(tf.keras.Model)
    __init__    .. provided, explain what it does
    call       .. provided, explain what it does

class Decoder(tf.keras.Model)
    __init__    .. provided, explain what it does
    call       .. provided, explain what it does

class VAE(tf.keras.Model)
    def __init__()
        Variational autoencoder-generative adversarial network. Contains a
        VAE which learns to compress and decompress an image and a
        discriminator which helps the VAE produce more realistic images.

    def call() ... Provided but add comments explaining what each line is doing!

    def train_step(self, data)
        Defines a single training iteration, including the forward pass,
        computation of losses, backpropagation, and weight updates
```

# ... Walk through of the VAE Exercise

## 4. Your task is to complete exercise 3a,3b,3c,3d,and 3e

```
def sample_from_normal() exercise3a
# Sample a point from a normal distribution, parameterized by a given mean and log-variance.

def kl_divergence() exercise3b
# Compute the closed-form KL Divergence between a given distribution and a normal prior
distribution with mean 0 and variance 1.

class Encoder(tf.keras.Model)
    __init__    .. provided, explain what it does
    call       .. provided, explain what it does

class Decoder(tf.keras.Model)
    __init__    .. provided, explain what it does
    call       .. provided, explain what it does

class VAE(tf.keras.Model)
    def __init__() exercise3c
        Variational autoencoder-generative adversarial network. Contains a
        VAE which learns to compress and decompress an image and a
        discriminator which helps the VAE produce more realistic images.

    def call() exercise3d    ... Provided but add comments explaining what each line is doing!

def train_step(self, data) exercise3e
    Defines a single training iteration, including the forward pass,
    computation of losses, backpropagation, and weight updates
```

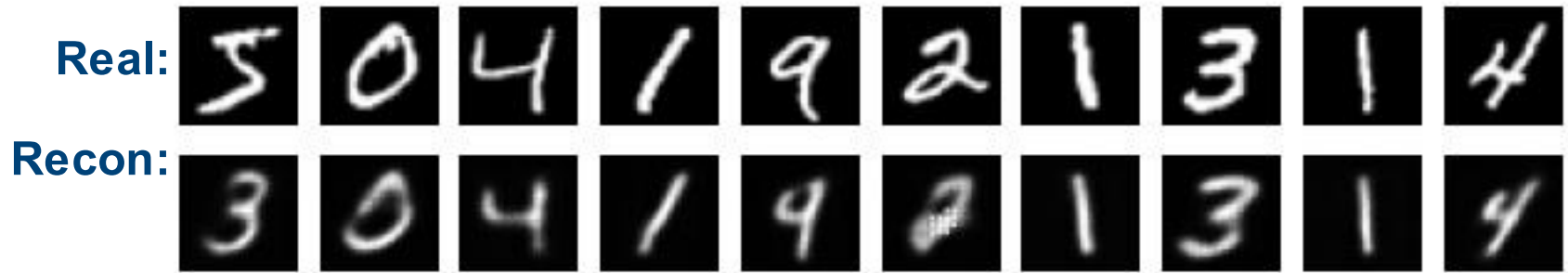
## ... Walk through of the VAE Exercise

---

5. To get credit, you will need to **git add and commit** the new files (after you complete the exercises)

# VAE Reconstructed images at epoch 1

epoch001\_recons.png



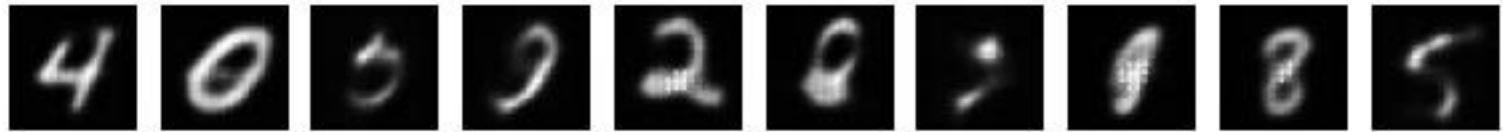
What do you observe as the # epochs increases?

# VAE Purely synthesized images at epoch 1

## epoch001\_fakes.png

---

**Note: we cannot steer (choose ) class label of the generated image.  
Images from 10 random Z's shown**



**How do the results change as the # epochs increases?**

# Acknowledgements



Albert Montillo,  
PhD, PI



Son Nguyen, PhD  
Postdoc



Alex Treacher  
PhD student



Aixa Andrade Hernandez,  
MS, PhD student



Austin Marckx  
PhD student



Krishna Chitta  
Res. Sci.



----- Recent Alumni -----



Atef Ali  
Undergrad



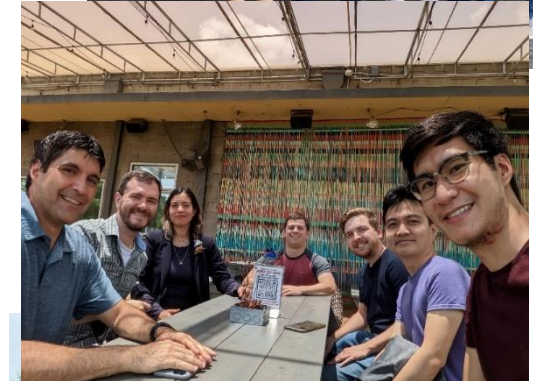
Vyom Raval, BS  
MD/PhD



Kevin Nguyen  
MD/PhD student



Cooper Mellema  
MD/PhD student



## Lab Funding

- **NIH/ NIGMS R01 *Correcting Biases in Deep Learning***
- King Foundation (PI) : Quantitative AD diagnostics.
- Lyda Hill Foundation (PI): Quantitative prognostics of Parkinson's disease
- **NIH/ NIA R01** Blood Biomarkers for Alzheimer's and Parkinson's
- TARCC : Texas Alzheimer's Research and Care Consortium.
- **NIH / NINDS F31 fellowship : Causal connectivity biomarkers for neurological disorders**



# Thank you!

---

**Email:** [Albert.Montillo@UTSouthwestern.edu](mailto:Albert.Montillo@UTSouthwestern.edu)

**Github:** <https://github.com/DeepLearningForPrecisionHealthLab>

MegNET .... Artifact suppression

BLENDS .... fMRI augmentation

Antidepressant-Reward-fMRI .... response prediction

Parkinson-Severity-rsfMRI ... disease trajectory prediction



# End of presentation

---