Bayesian Auto-Encoders
Part Two

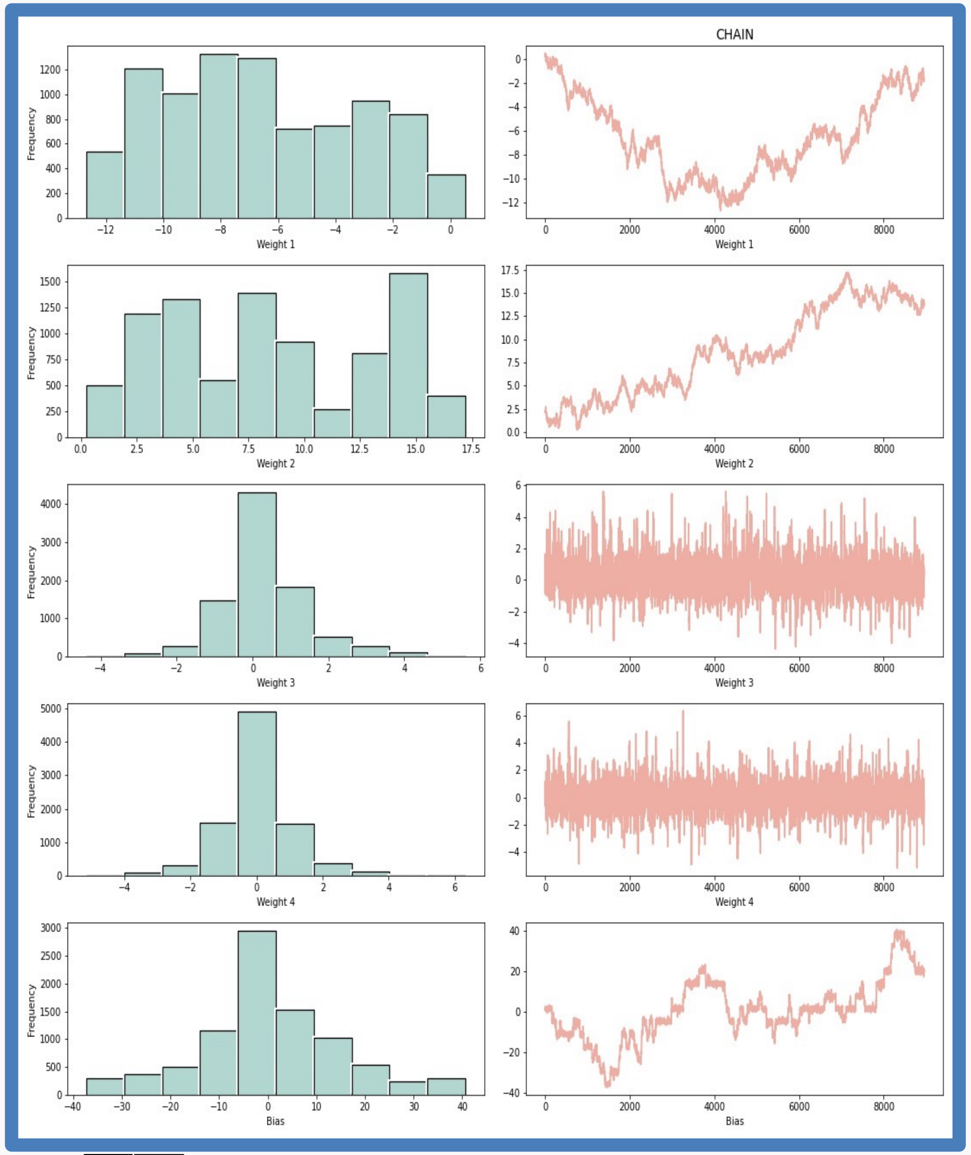
CS109B Data Science 2

Pavlos Protopapas, Mark Glickman, and Chris Tanner

Outline: Part 2

- Motivation for Variational Autoencoders (VAE)
- Inference in Neural Networks
  - Bayesian Linear Regression
  - Bayesian Neural Networks
  - Introduction to Variational methods
  - Variational Autoencoder as an inference model
- Variational Autoencoders as generative model
  - Separability of VAE
  - Tips & tricks
  - Other generative models

Skulls of Bayesian Methods



MCMC will not work for NN's with more than a few dozens of parameters.

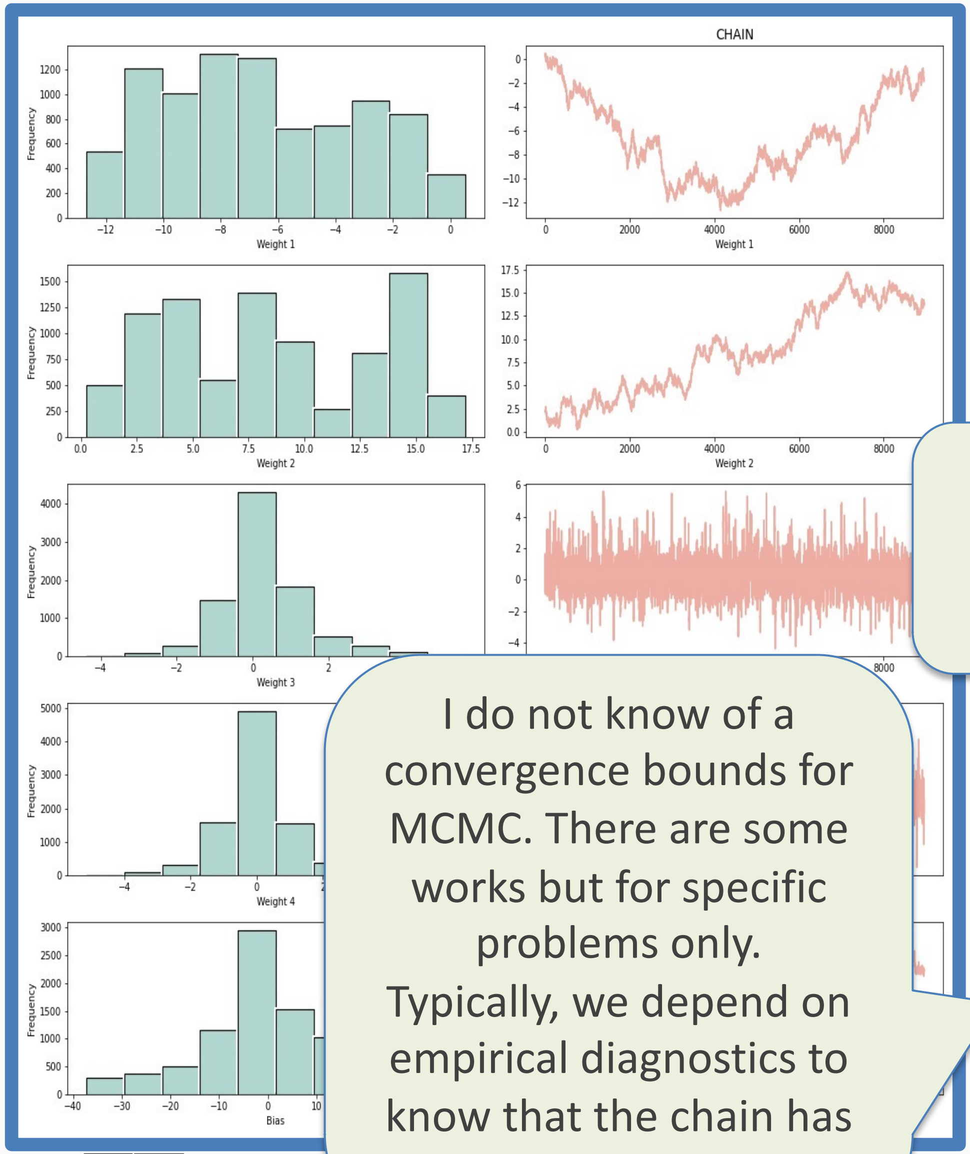
Why?

For each parameter (weight), we sample and calculate the likelihood $5 * n$ times, where n is the chain's length.

We also throw away a significant number of samples from the beginning of the chain.

The number of samples, n, necessary to adequately capture the distribution grows with the number of parameters and complexity of the posterior.

Skulls of Bayesian Methods



MCMC will not work for NN's with more than a few dozens of parameters.

Five times because we usually have acceptance rate of $\sim 20\%$.

Why?

For each parameter (weig...), we sample and ...ihood $5 * n$ times, where n is the

Typically, we throw the first 10-20% of the samples for burn in.

We also throw away a significant number of samples from the beginning of the chain.

I do not know of a convergence bounds for MCMC. There are some works but for specific problems only. Typically, we depend on empirical diagnostics to know that the chain has converged

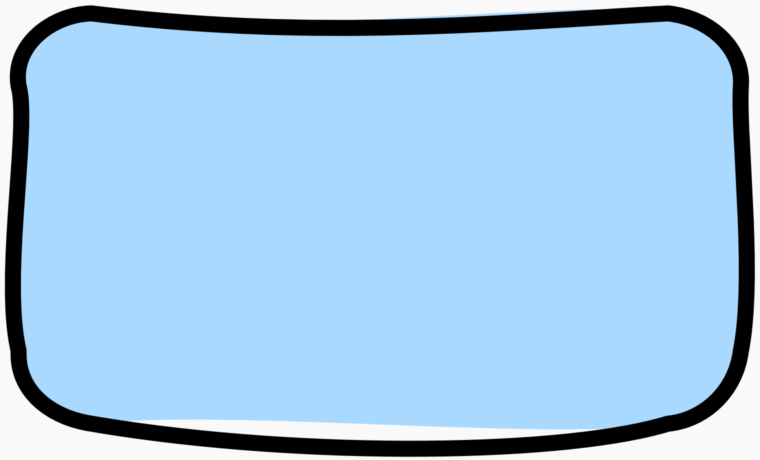
The number of samples, n, necessary to adequately capture the distribution grows with the number of parameters and complexity of the posterior.


Guess we'll need an alternative route!
BAYESIAN LINEAR REGRESSION
VARIATIONAL AUTO-ENCODERS
N
W
E
S

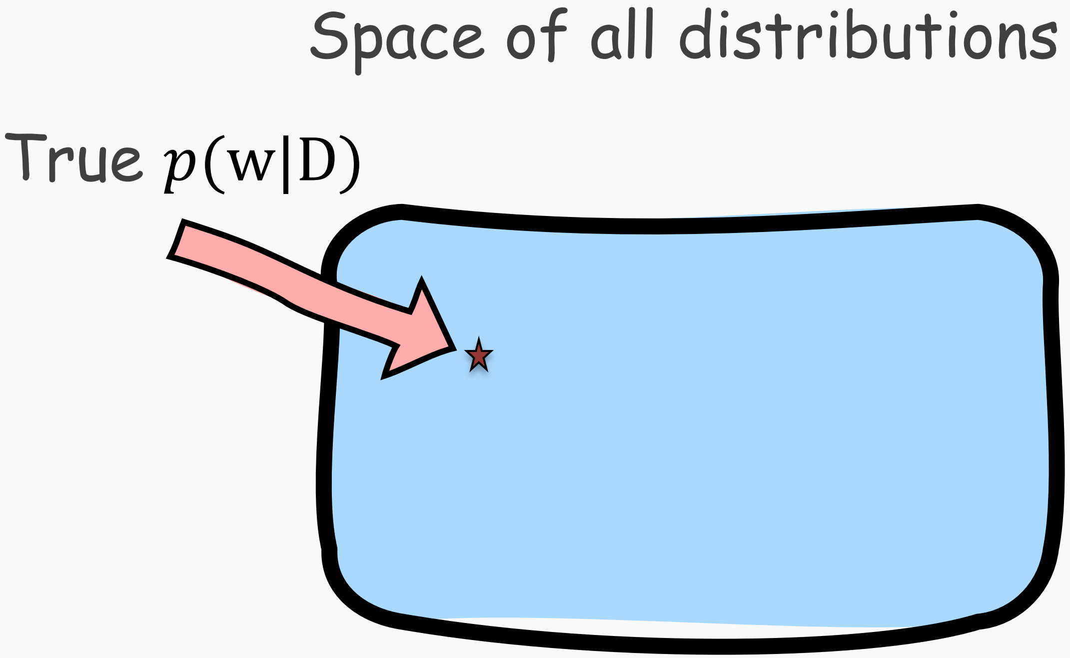
Variational Approximations

Alternative method of estimating the posteriors: Variational Inference

Space of all distributions

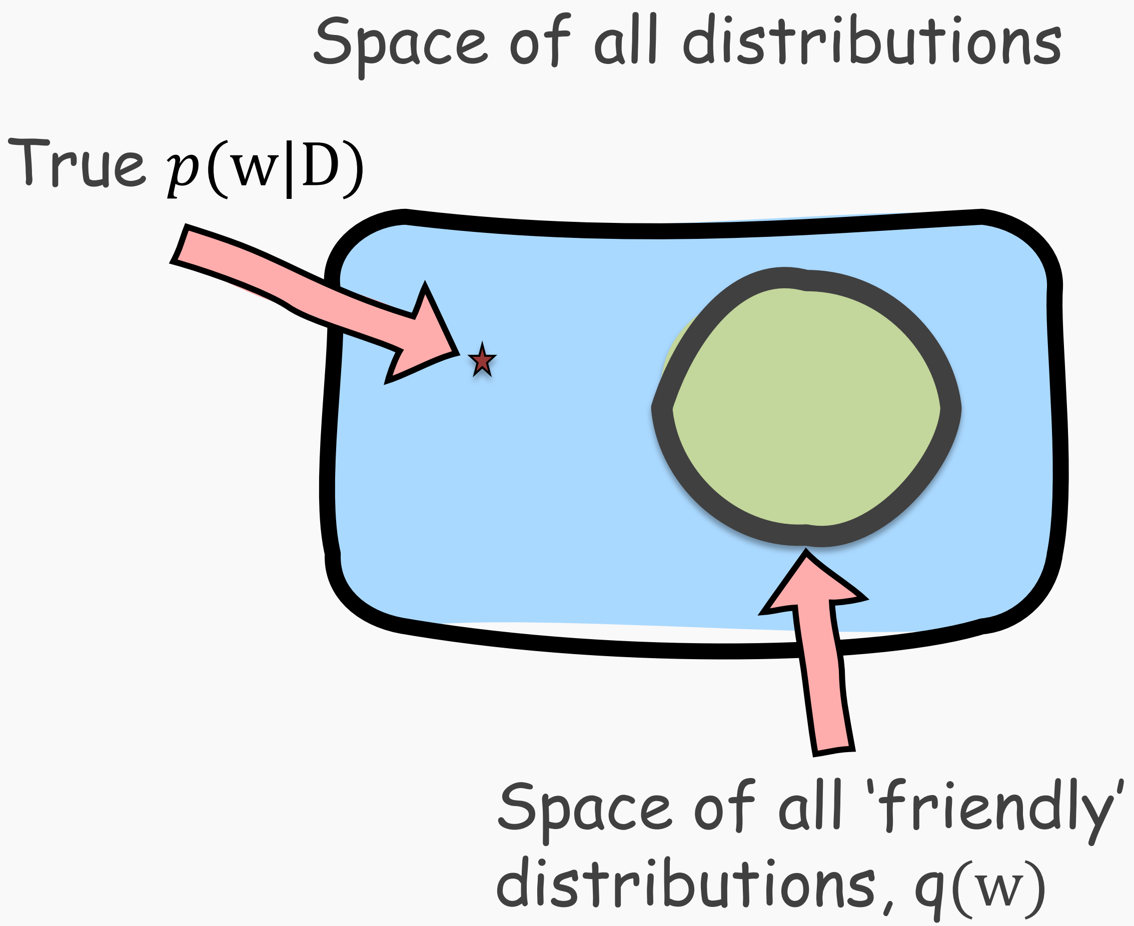


Alternative method of estimating the posteriors: Variational Inference



Let $p(w|\mathrm{D})$ be the true posterior distribution.

Alternative method of estimating the posteriors: Variational Inference



Let $p(w|\mathrm{D})$ be the true posterior distribution.

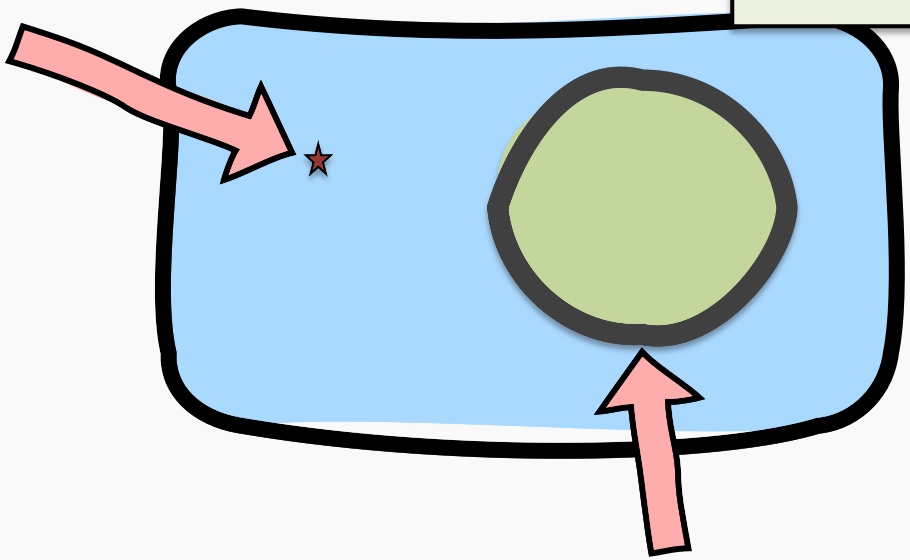
We want to find another distribution, which is easier to deal with, $q(w)$, that is similar to $p(w|D)$.

Alternative method of estimating the posteriors: Variational Inference

Space of all distributions

True $p(\mathrm{w}|\mathrm{D})$

D: Data



Space of all 'friendly' distributions, q(w)

We often use normal for the friendly distributions q(w)

Let $p(w|\mathrm{D})$ be the true posterior distribution.

Why KL: Because the maths work nicely

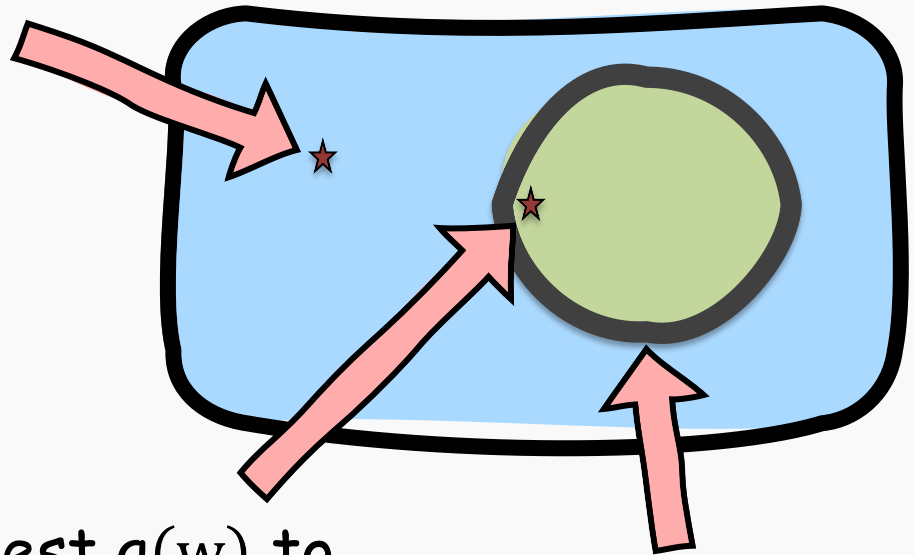
We want to find another distribution, which is easier to deal with, $q(w)$, that is similar to $p(w|D)$.

To do so we define the meaning of 'similar' to be some form of distance between $q(\mathrm{w})$ *and* $p(\mathrm{w}|\mathrm{D})$. We will use KL divergence for that:

$$D_{KL}[q|p] = \int q(w) \log \frac{q(w)}{p(w|D)} dw$$

Technically not a distance

Alternative method of estimating the posteriors: Variational Inference



$$D_{KL}[q|p] = \int q(w) \log \frac{q(w)}{p(w|D)} dx$$

By minimizing over all functions $q(w)$

$$q^* = argmin_q \int q(w) \log \frac{q(w)}{p(w|D)} dw,$$

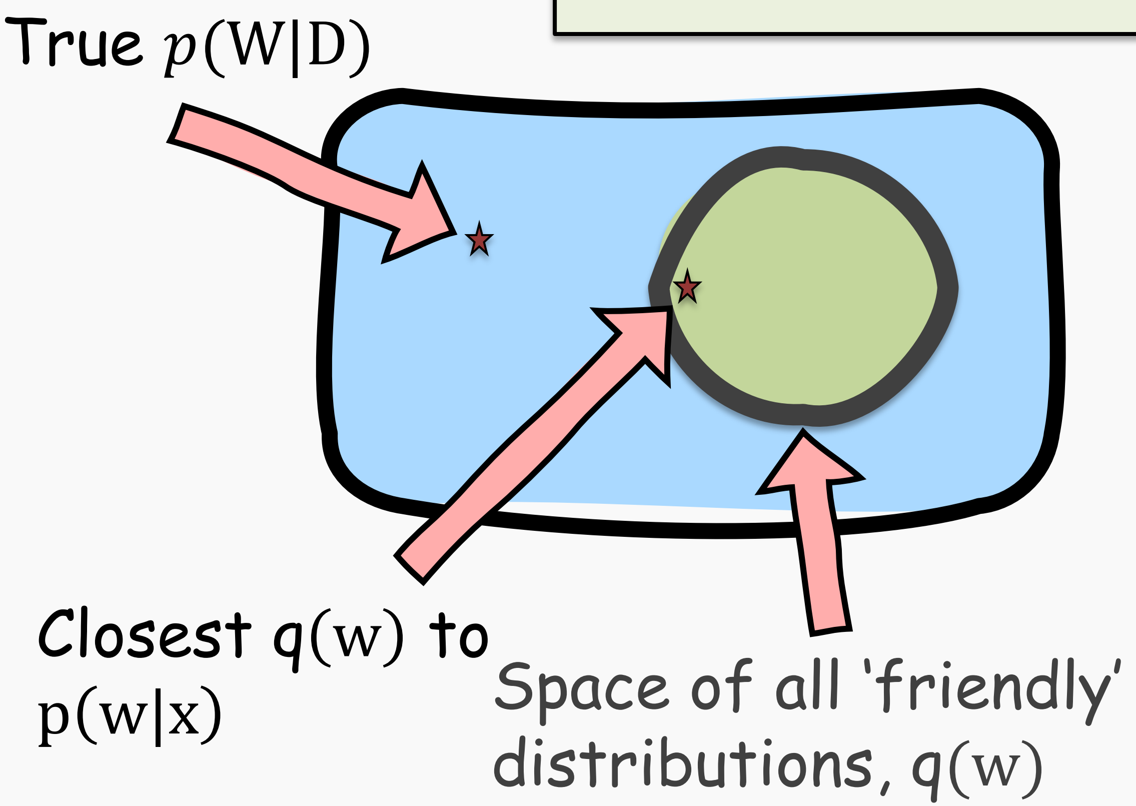
we will discover $q(w)$ that is the closest to $p(w|D)$, namely $q^*(.)$.

If $q(.)$ is parametrized by ϕ, $q_\phi(.)$

$$\phi^* = argmin_\phi \int q_\phi(w) \log \frac{q_\phi(w)}{p(w|D)} dw$$

If $q_\phi(.)$ is normal, q_ϕ, ϕ is $\{\mu, \sigma\}$

Alternative method of estimating the posteriors: Variational Inference



We will always need to choose the prior of the parameters.

$$\phi^* = argmin_\phi \int q_\phi(w) \log \frac{q_\phi(w)}{p(w|D)} dw$$

Doing a "little" of math that we will cover in the advanced section, we can derive a new loss function:

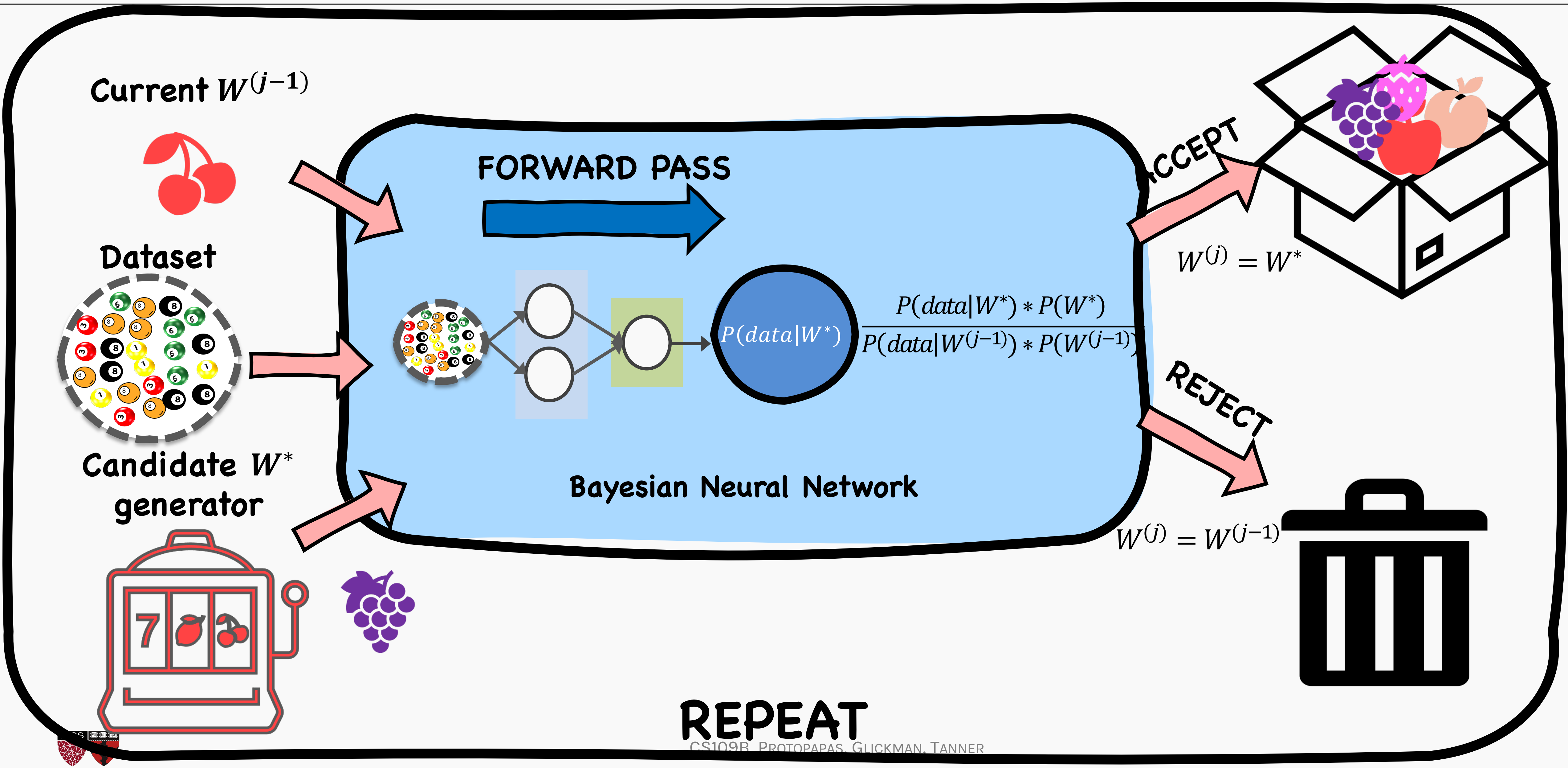
$$\mathcal{L} = KL\big(q_\phi(w)||p(w)\big) - E_{q_\phi}[\log p(D|w)]$$

The important point is that we do not need to sample to discover the posterior distribution but, minimize this new loss function w.r.t. to ϕ.

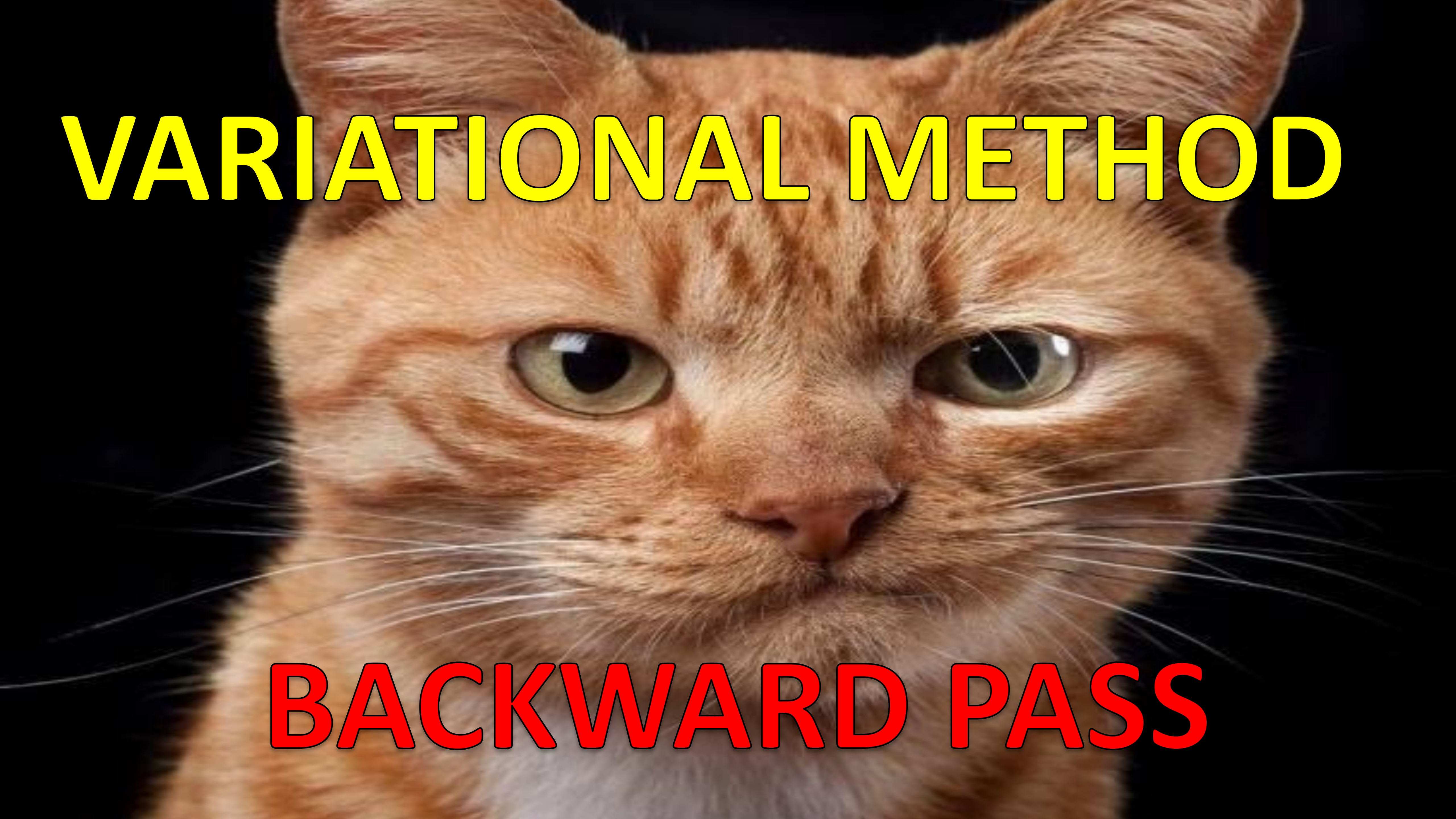
This can be approached with gradient descent or stochastic gradient descent.

Variational Method in Action

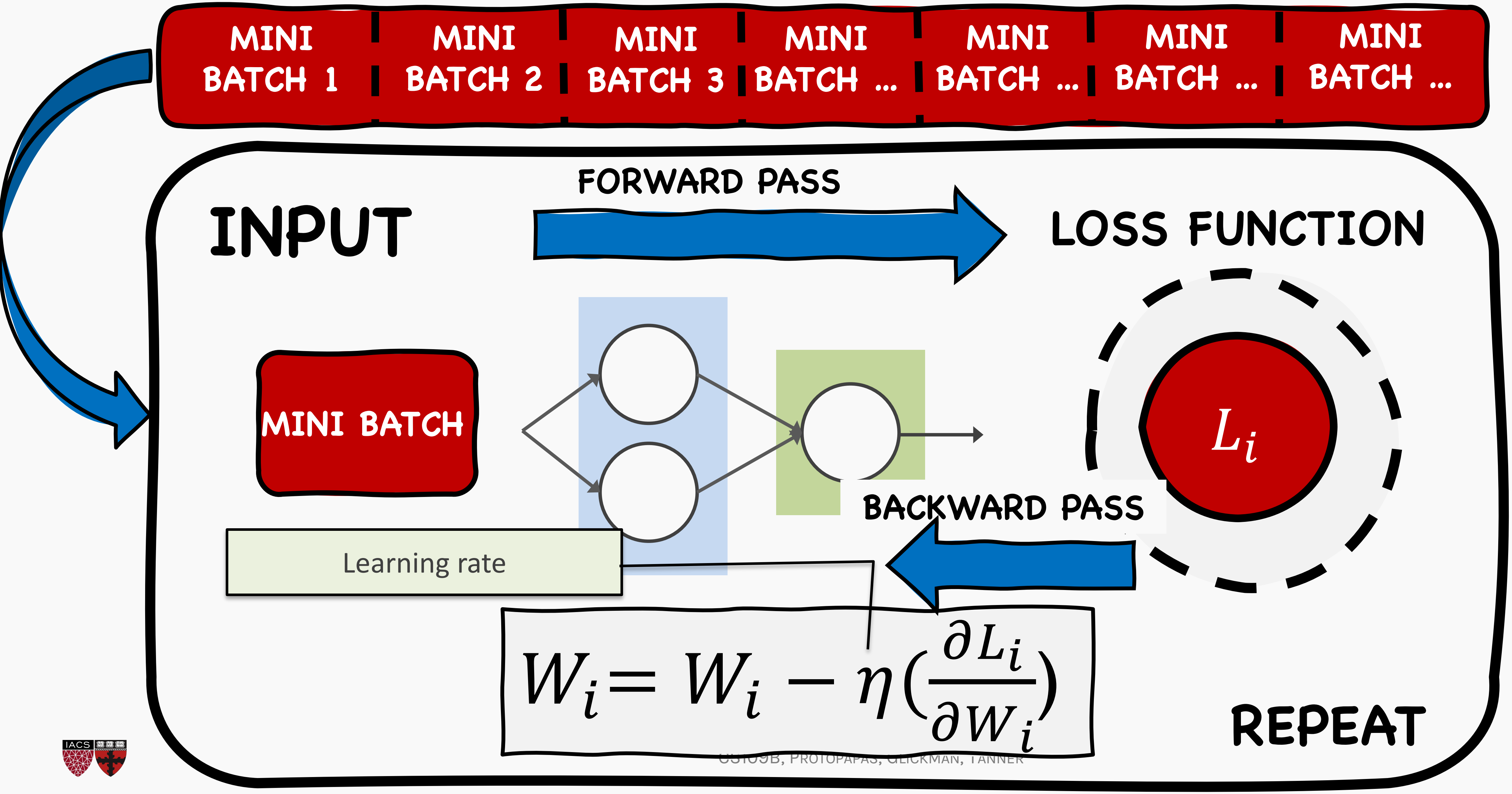
RECAP: Bayesian Neural Network




Bayesian Neural Network
with MCMC
FORWARD PASS ONLY
THAT'S IT?

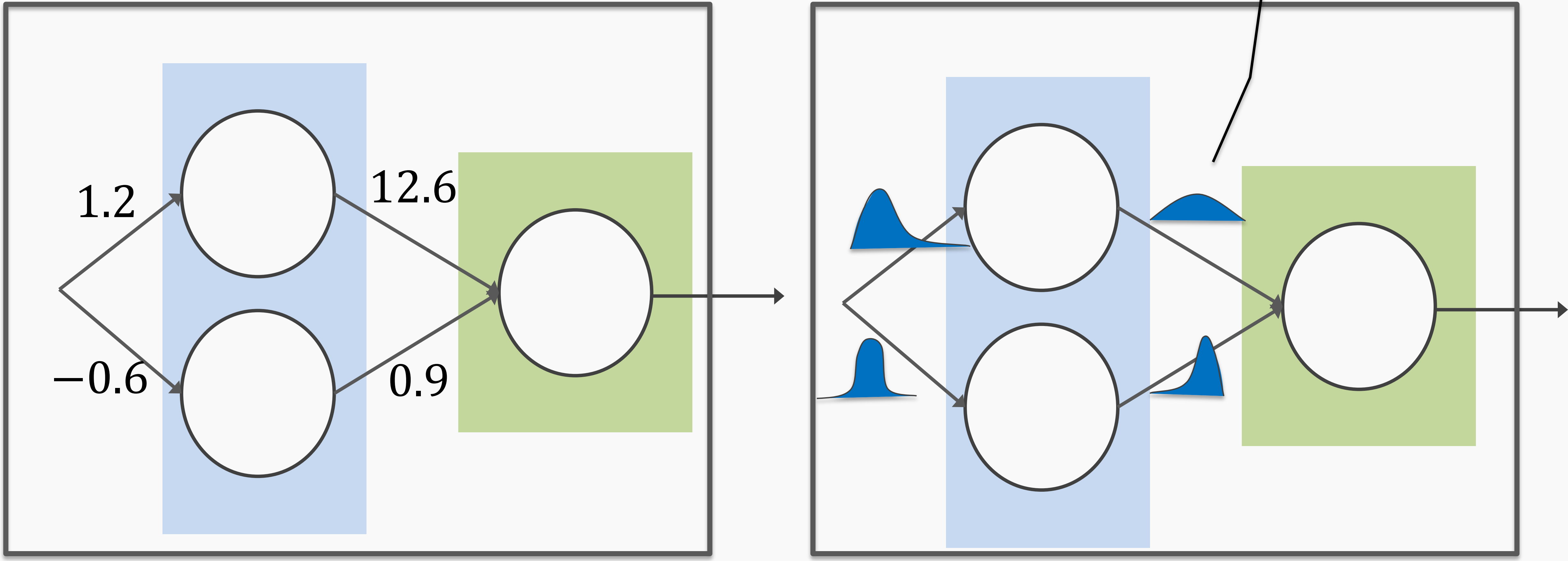

VARIATIONAL METHOD
BACKWARD PASS

RECAP: Neural Network



Variational Neural Network

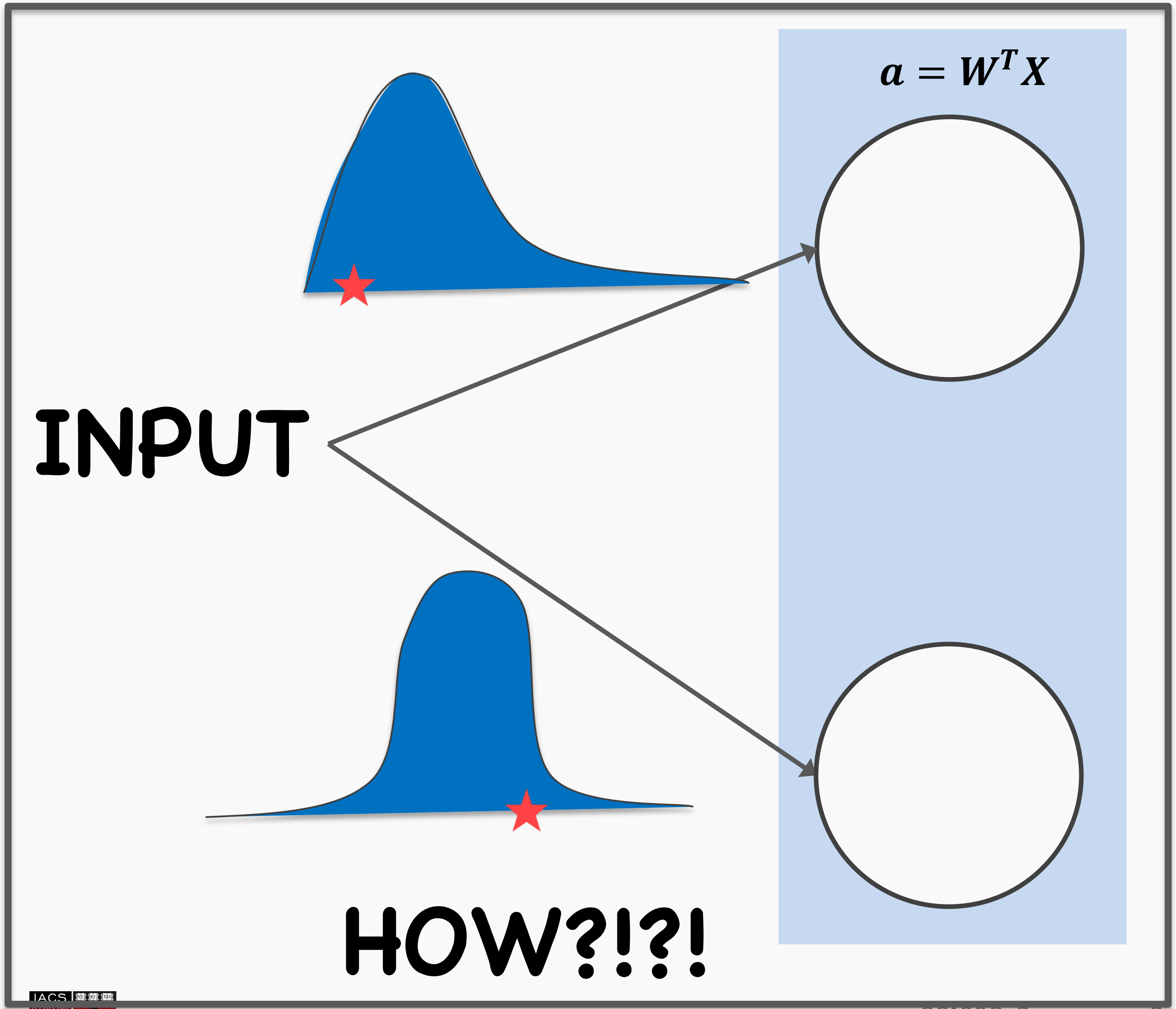
The distributions of w's are now the $q(w_i)$'s



BEFORE (Deterministic weights)

AFTER (Probabilistic weights)

Variational Neural Network



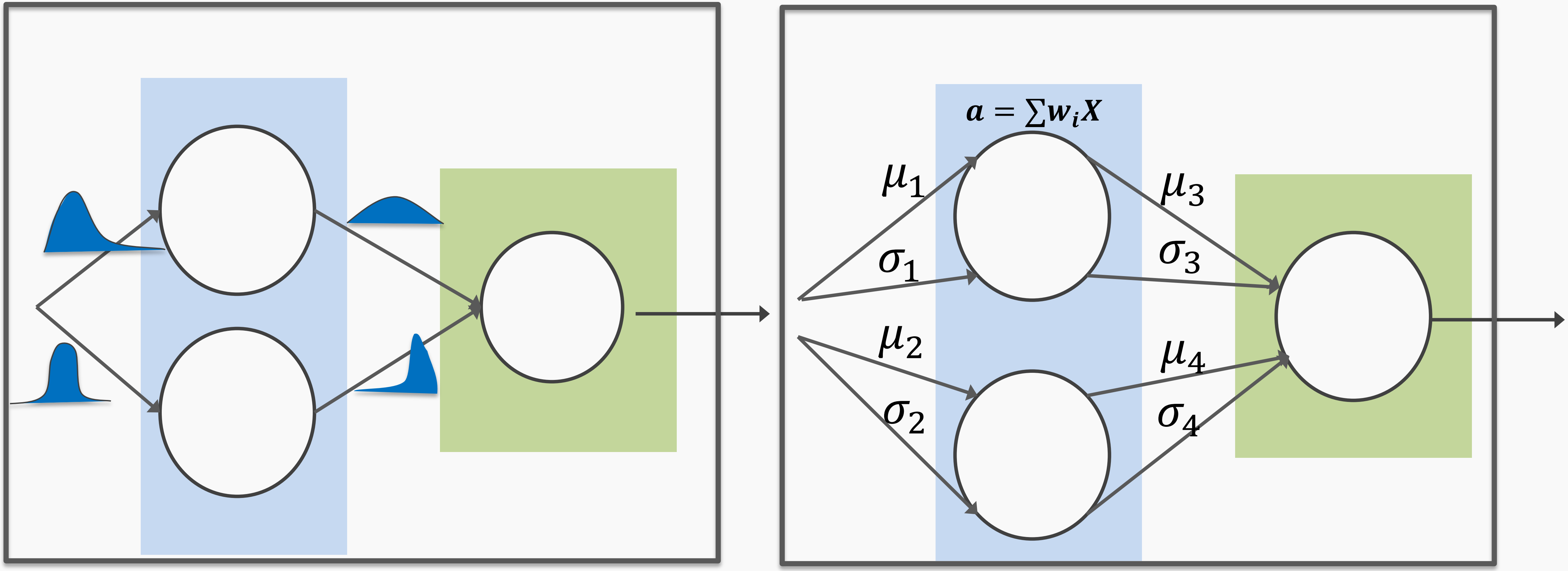
- In variational methods, we assume a weight distribution, $q_\phi(w)$, with distribution parameters, ϕ, which are to be optimized to best match the true posterior, $p(W|\mathrm{D})$.
- However, for the **forward pass**, to compute the activations, we need values for the weights.
- Since we assume a distribution for the weights, we can take a sample from that distribution, $q_\phi(w)$, for some scaling parameters $\phi = \{\mu, \sigma\}$.

We will learn these parameters.

Variational Neural Network

$\boldsymbol{w_i = \mu_i + \sigma_i \odot \epsilon}$

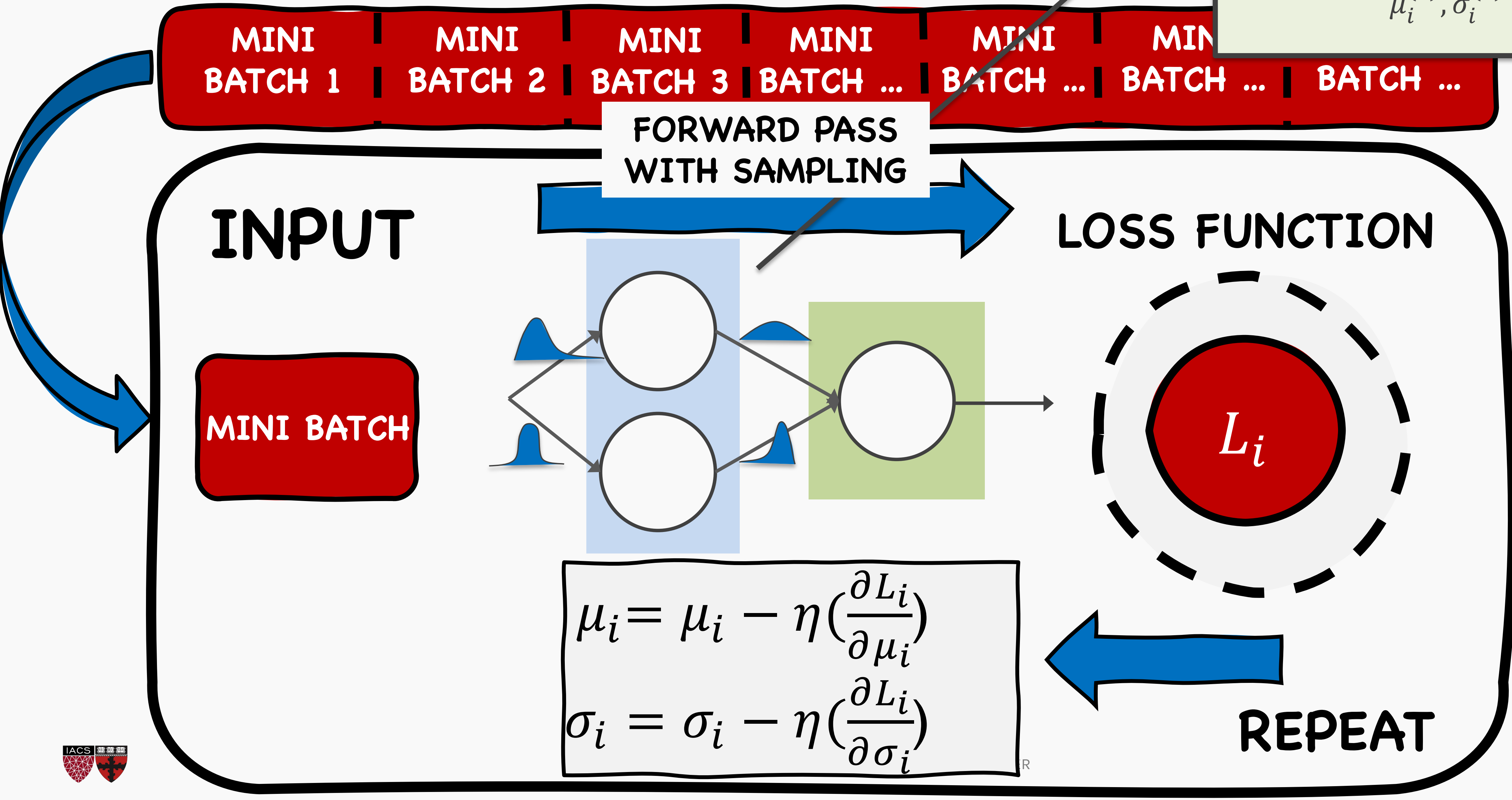
$\epsilon \sim N(0,1)$
This is equivalent to
$w_i \sim q_{\mu_i,\sigma_i}(W) = N(\mu_i, \sigma_i)$



This will double our trainable parameters, as we optimize for the μ & σ for each weight distribution.

RECAP: Variational Neural Network

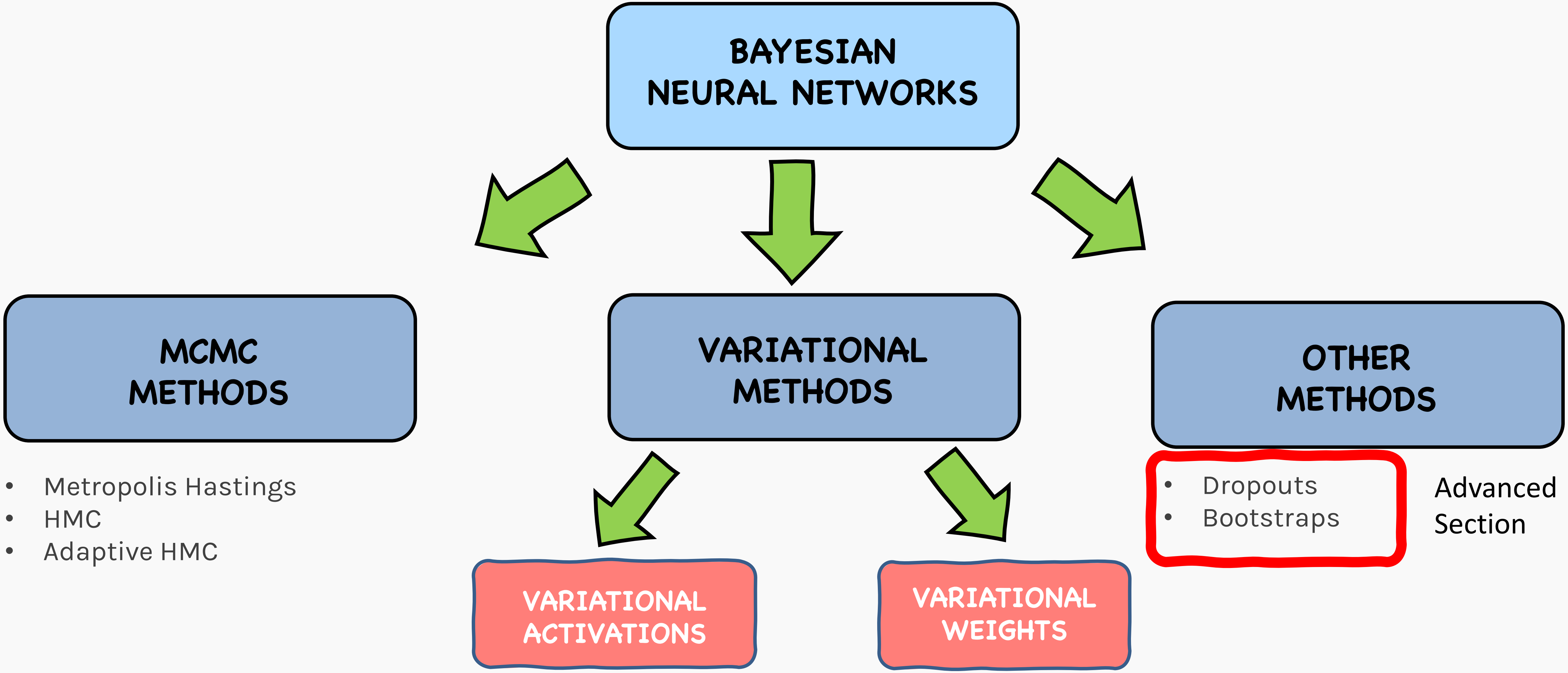
We start with some random $\mu_i^{(0)}, \sigma_i^{(0)}$



$$\mu_i = \mu_i - \eta\left(\frac{\partial L_i}{\partial \mu_i}\right)$$

$$\sigma_i = \sigma_i - \eta\left(\frac{\partial L_i}{\partial \sigma_i}\right)$$

Landscape of Inference Method for NN



MCMC Methods

- Metropolis Hastings
- HMC
- Adaptive HMC

Variational Activations

- Stochastic hidden units

Variational Weights

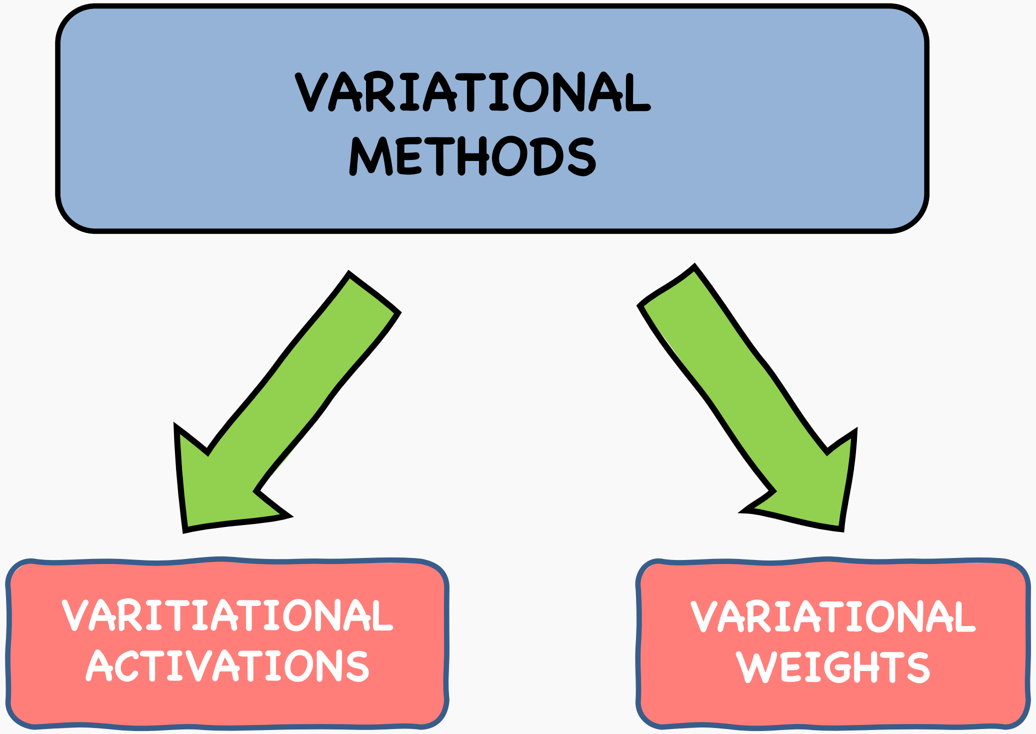
- Bayes By Backprop
- Flipout
- Dense Local Reparameterization

Other Methods

- Dropouts
- Bootstraps

Advanced Section

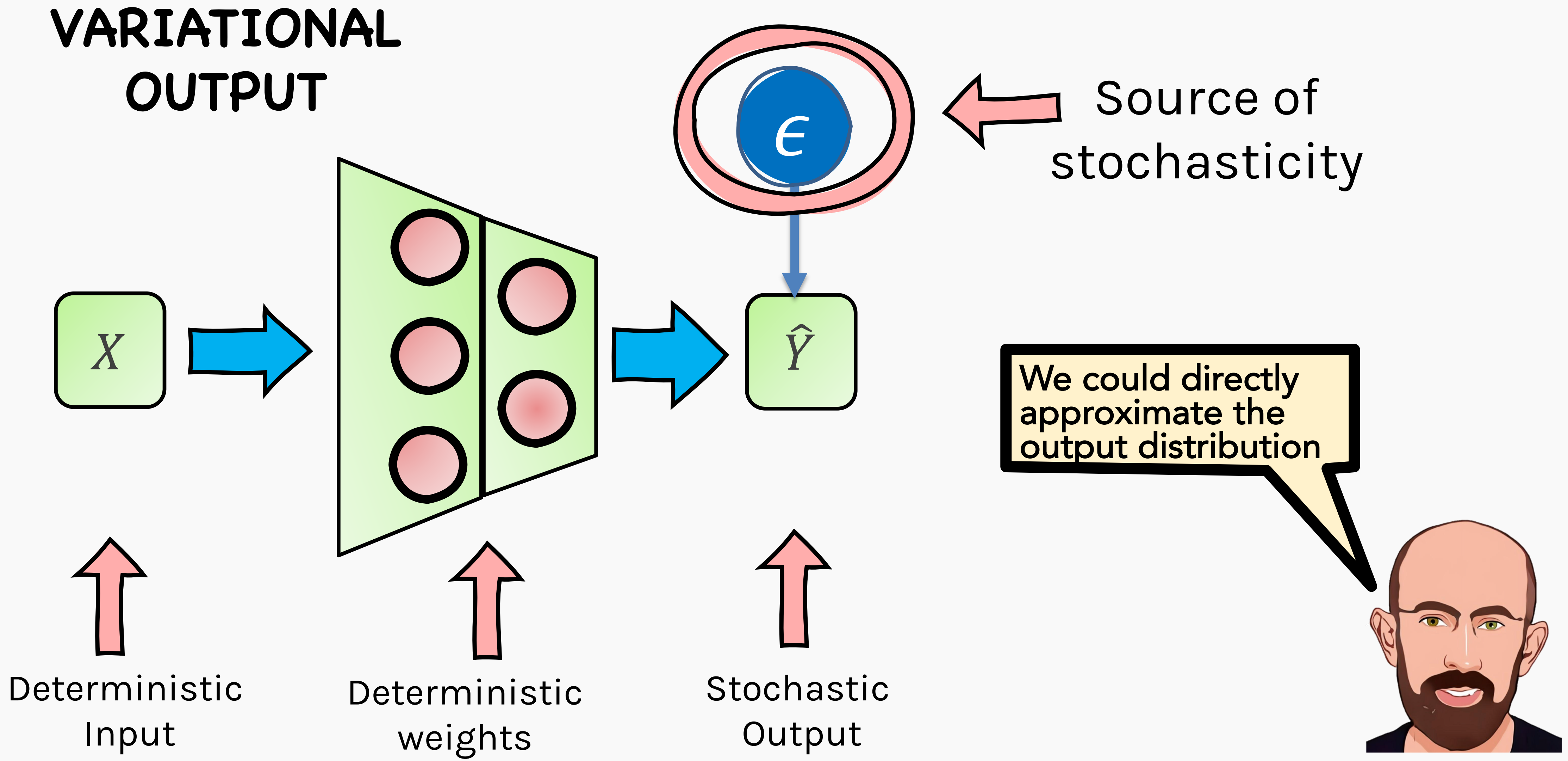
Variational Methods



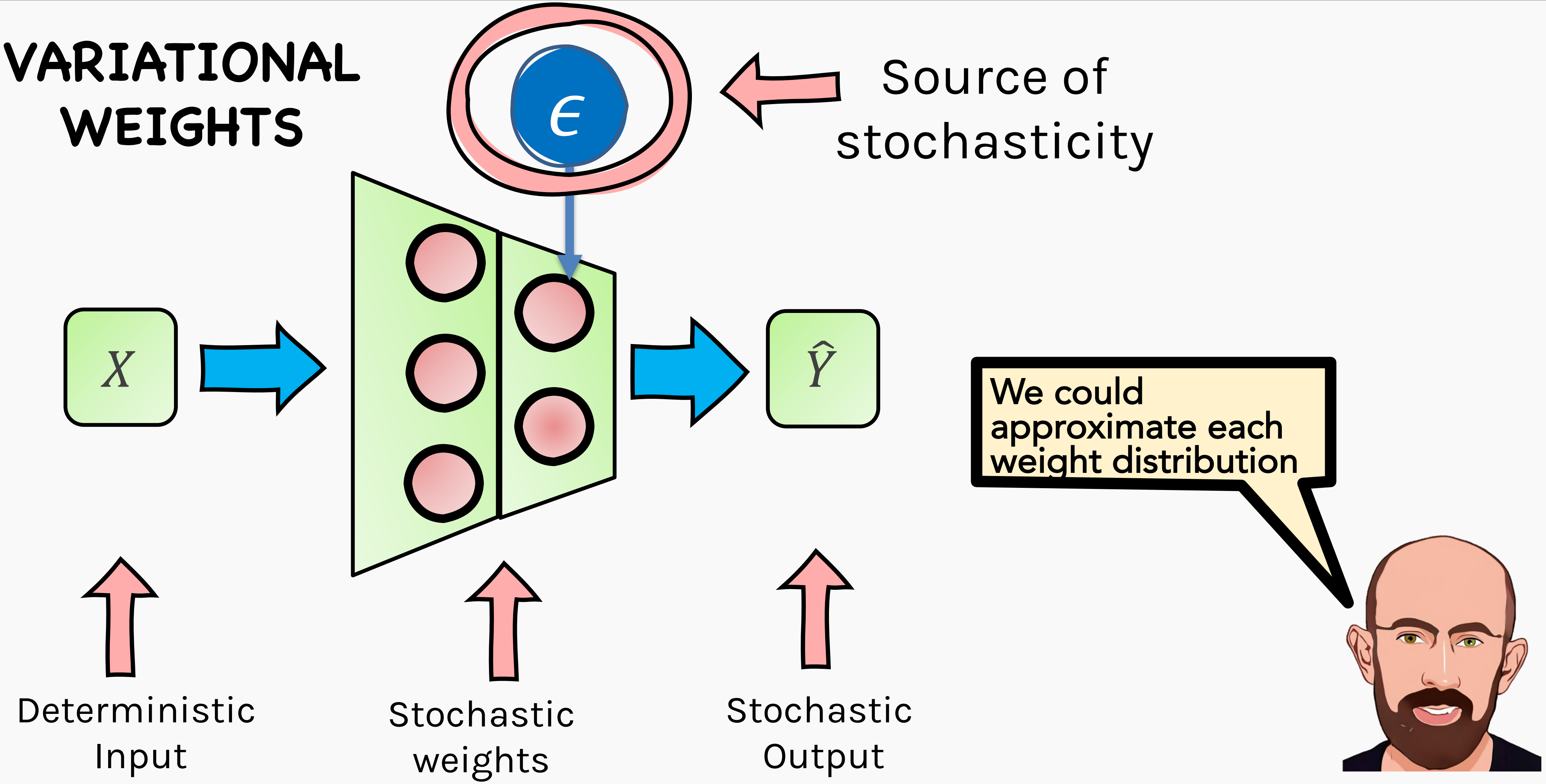
- Stochastic hidden units

- Bayes By Backprop
- Flipout
- Dense Local Reparameterization

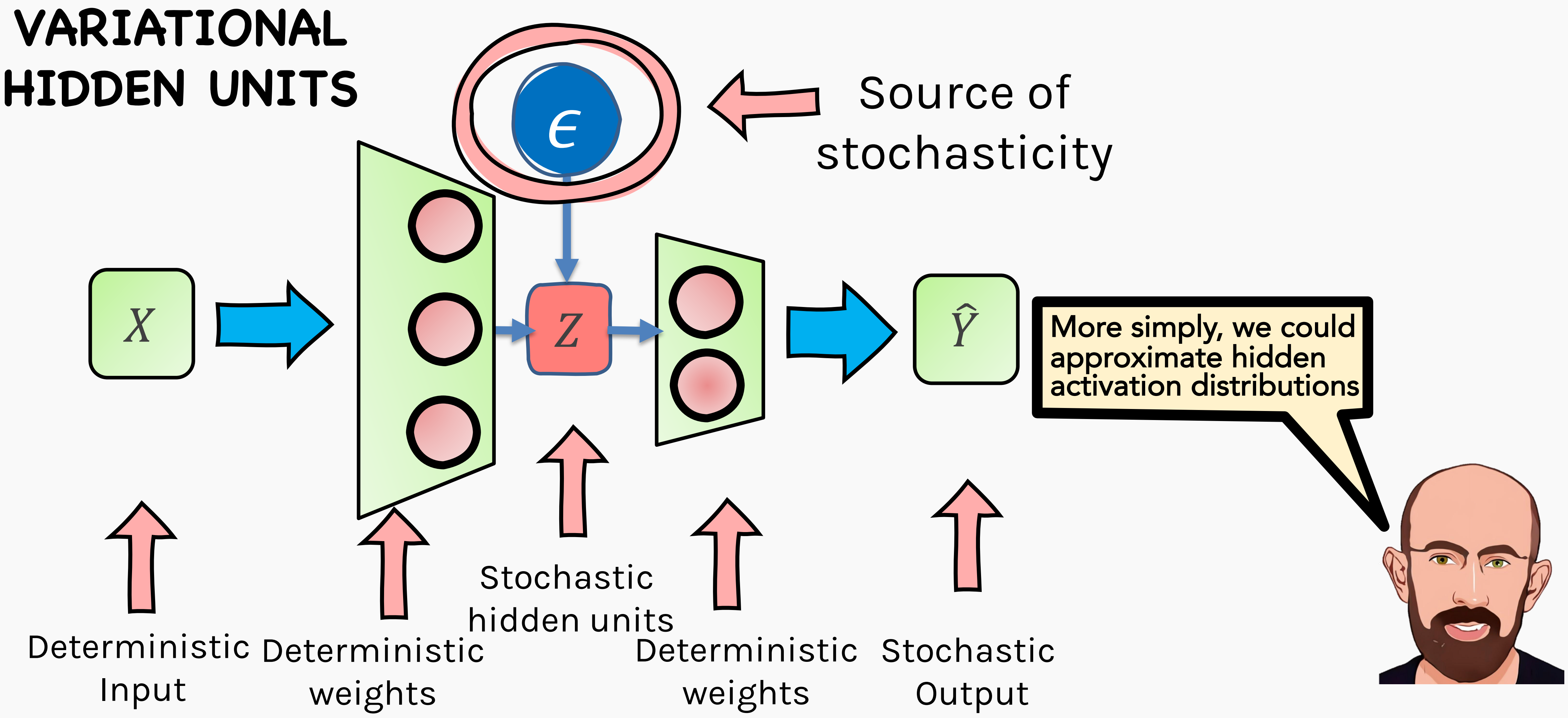
Quick Review



Quick Review



Quick Review



Building blocks of **probabilistic** machine learning

WORKFLOW

BUILD MODEL

↓

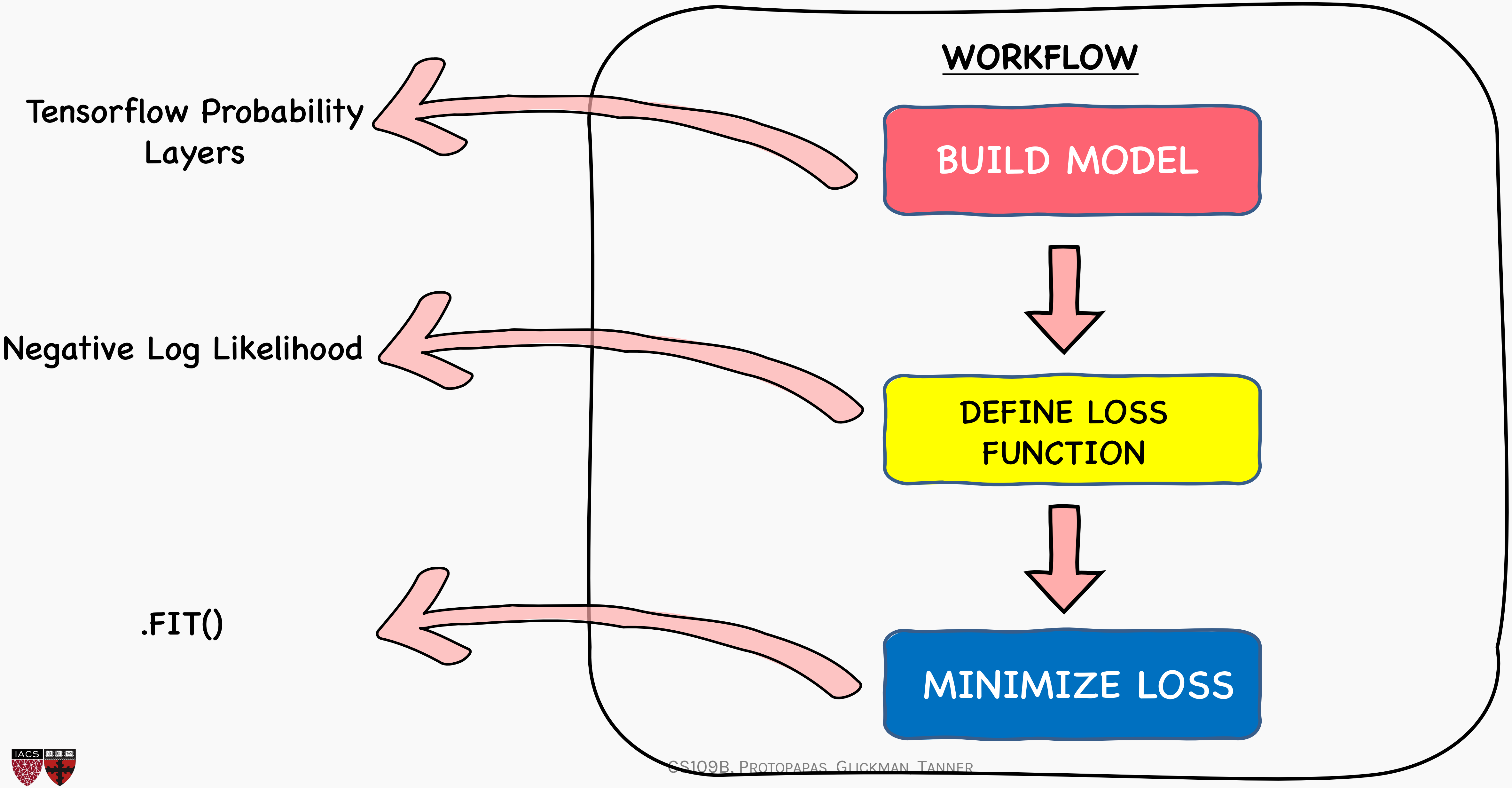
DEFINE LOSS FUNCTION

↓

MINIMIZE LOSS

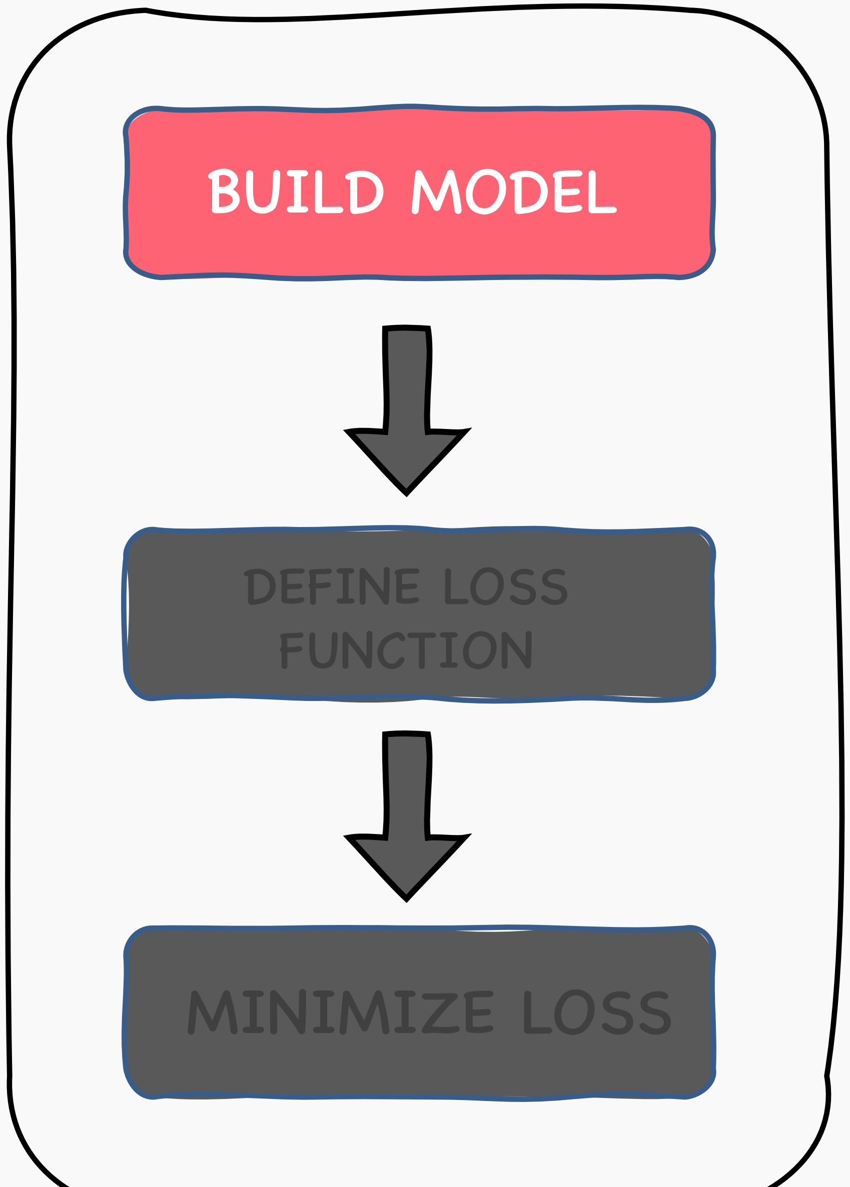
For variational methods, our workflow remains the same

Building blocks of **supervised** machine learning



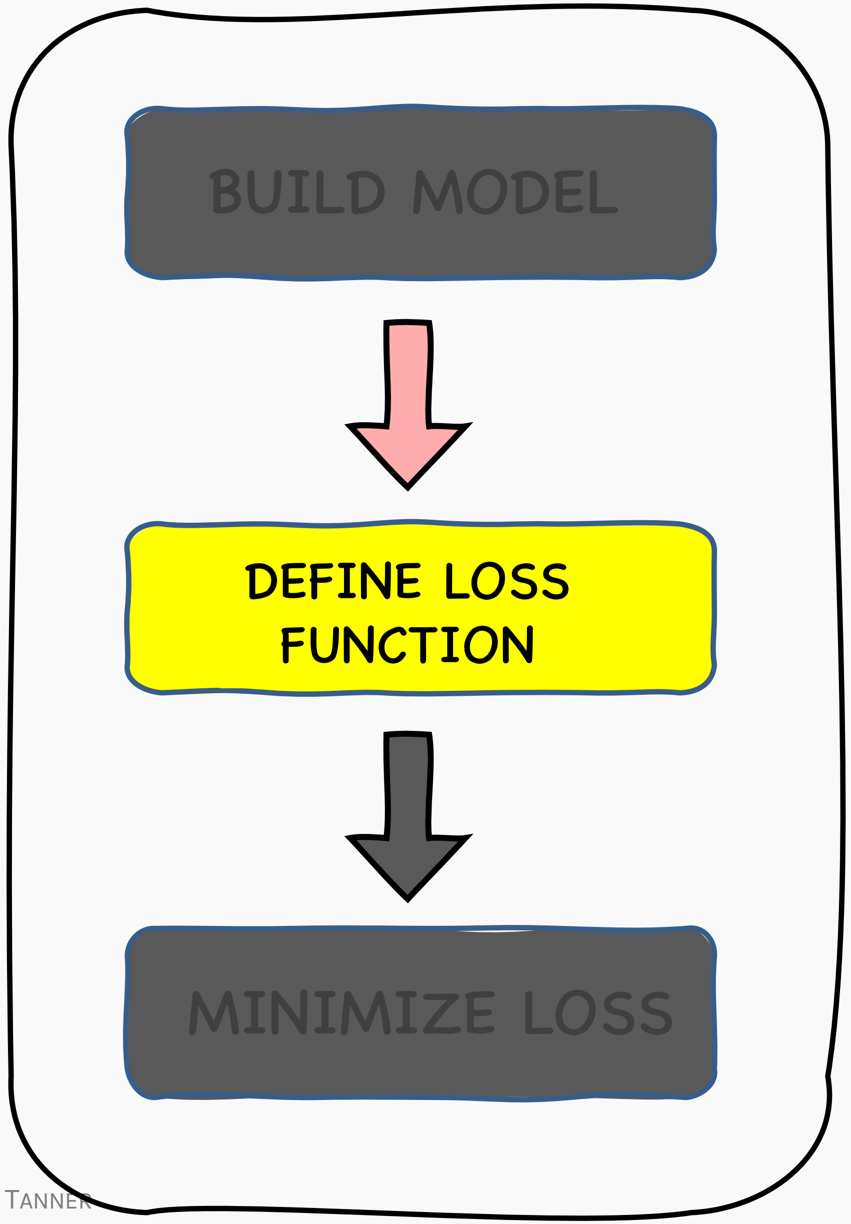
Building blocks of **supervised** machine learning

```python
# Encoder architecture defined below
encoder = tf.keras.Sequential(
          [
            tf.keras.layers.InputLayer(input_shape=input_shape),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(256,activation='relu'),
            tf.keras.layers.Dense(tfp.layers.IndependentNormal.params_size(2),
                                  activation=None, name='z_params'),
            # No activation
            tfp.layers.IndependentNormal(latent_size,
                    convert_to_tensor_fn=tfd.Distribution.sample,
                    activity_regularizer=tfp.layers.KLDivergenceRegularizer(prior, weight=1),
                    name='z_layer'),
          ]
    )
# Decoder architecture defined below
decoder = tf.keras.Sequential(
          [
            tf.keras.layers.InputLayer(input_shape=(latent_size,)),
            tf.keras.layers.Dense(32, activation='relu'),
            tf.keras.layers.Dense(128, activation='relu'),
            tf.keras.layers.Dense(784),
            tfp.layers.IndependentBernoulli((28,28,1), name='x_layer')
          ]
    )

# We combine the two to make thhe VAE
vae = tf.keras.Model(inputs=encoder.inputs,
                outputs=decoder(encoder.outputs[0]))
```



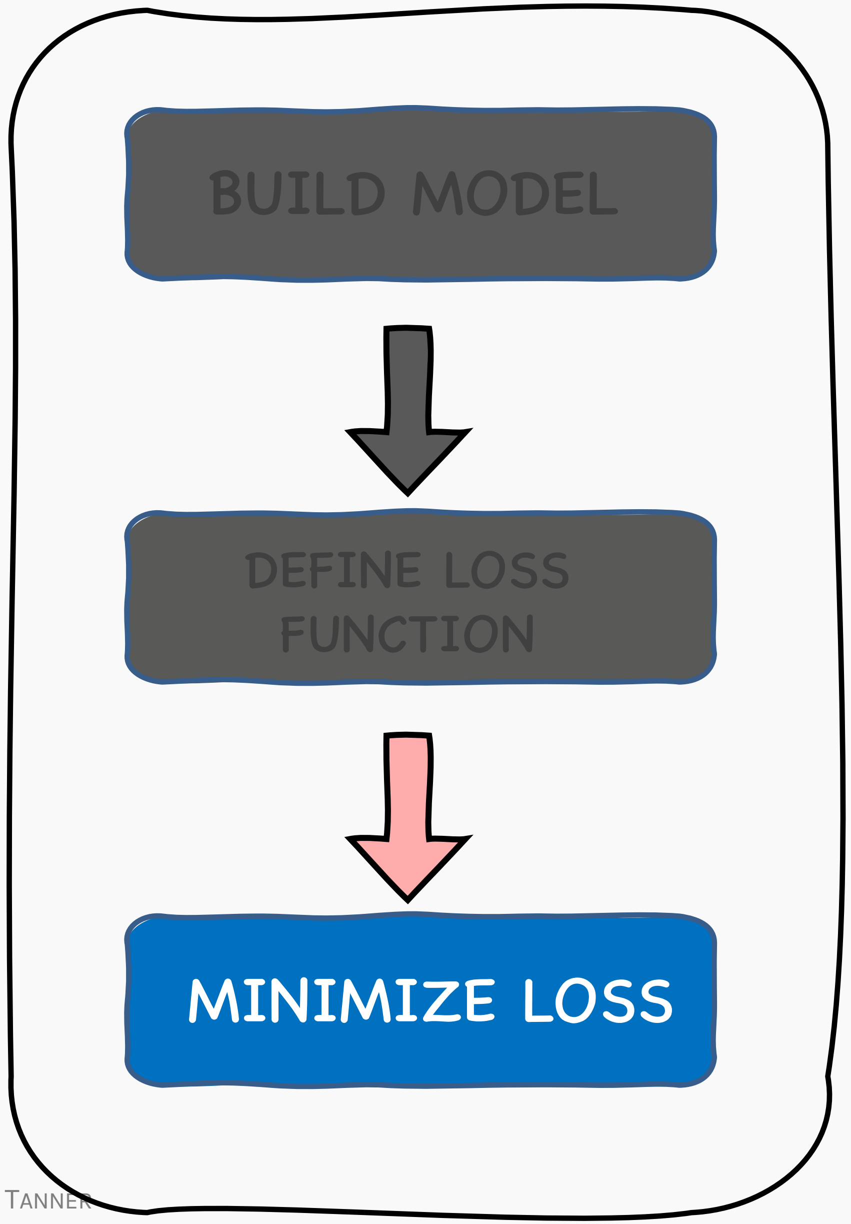
Building blocks of **supervised** machine learning

```
# Loss function Definition
def negloglik(y,rv_y):
  # rv_y is random variable
  # y is the output label
  return -rv_y.log_prob(y)

# Compile the VAE
vae.compile(optimizer=tf.optimizers.Adam(learning_rate=1e-3),
            loss=negloglik)
```

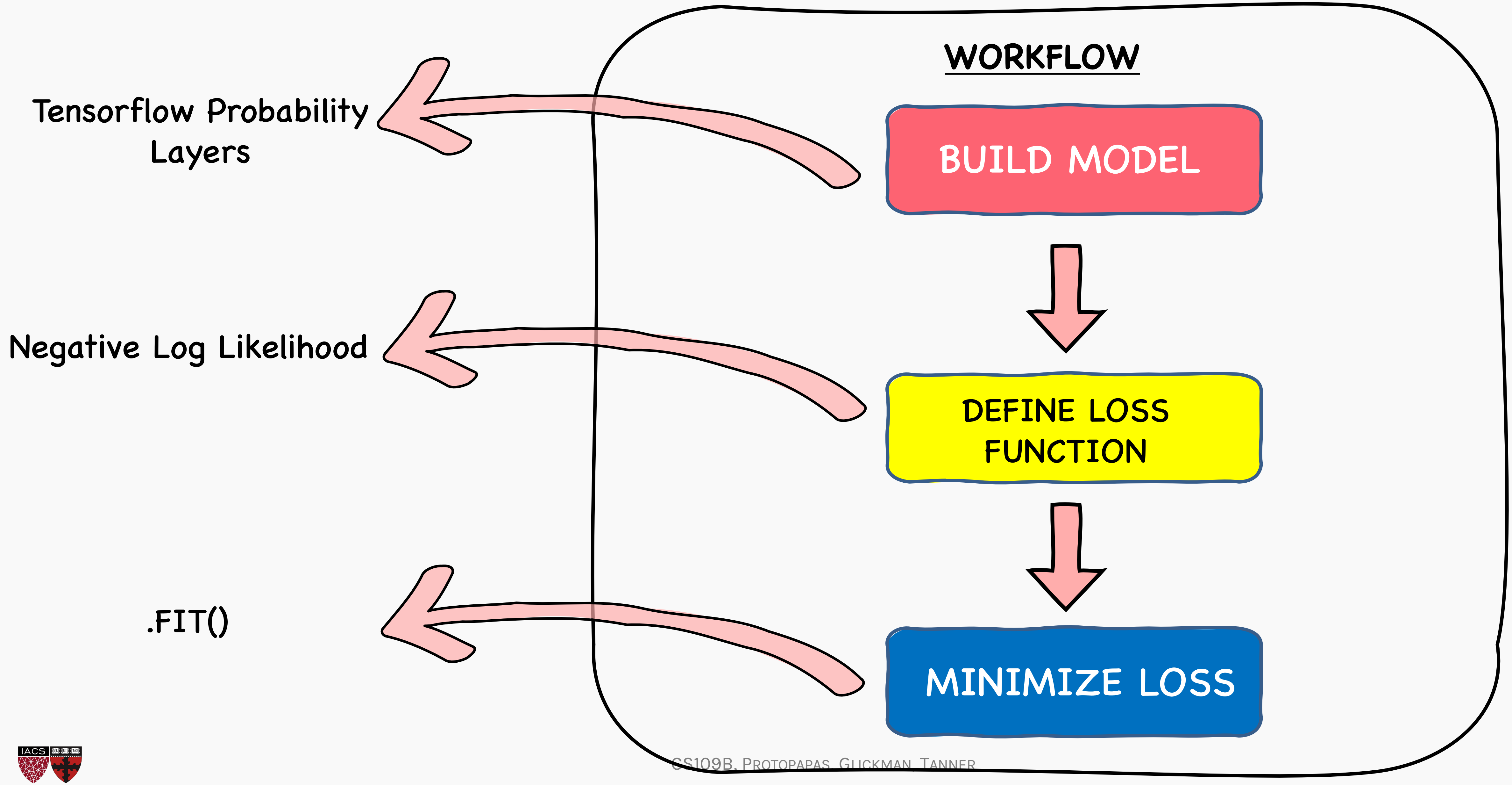


Building blocks of **supervised** machine learning

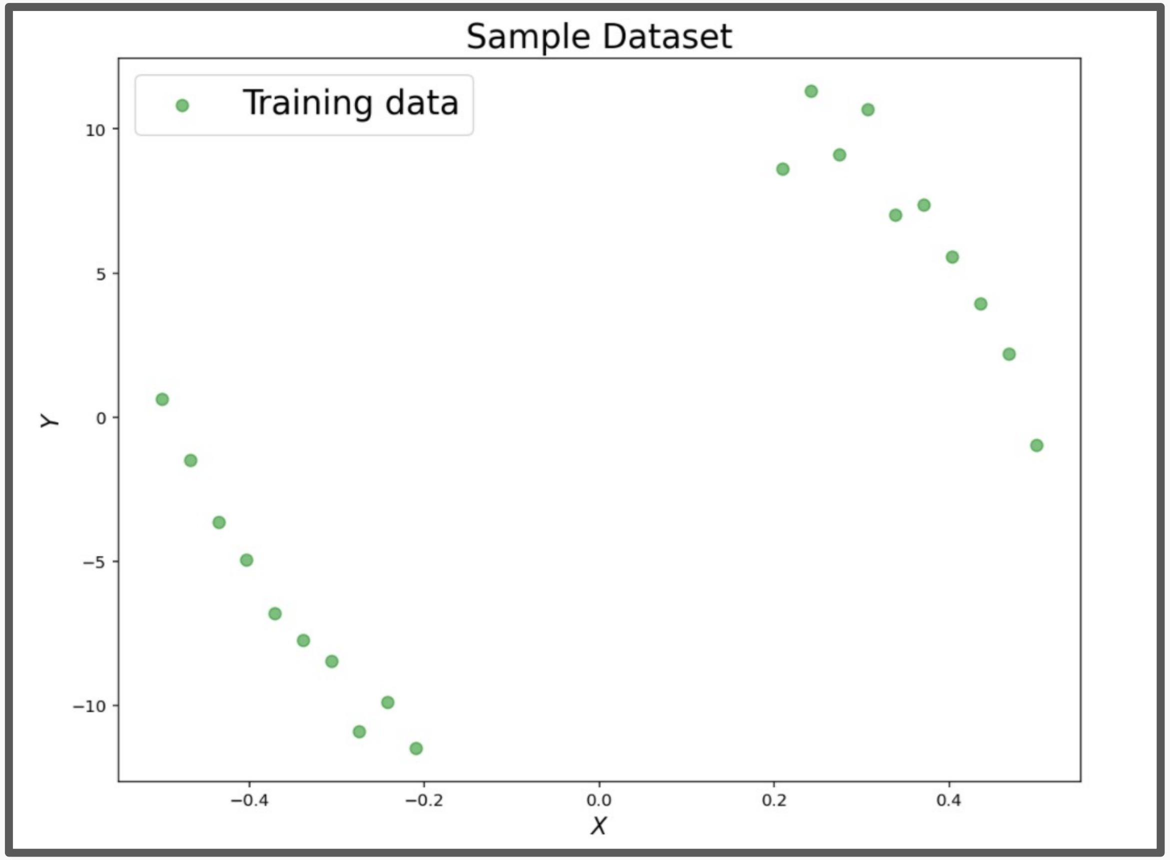
```
# Define training parameters
num_epochs = 15
train_dataset = X_train
eval_dataset=  X_val

# Use tf.keras .fit() function
vae.fit(train_dataset,
            epochs=15,
            validation_data=eval_dataset)
```



Building blocks of **supervised** machine learning



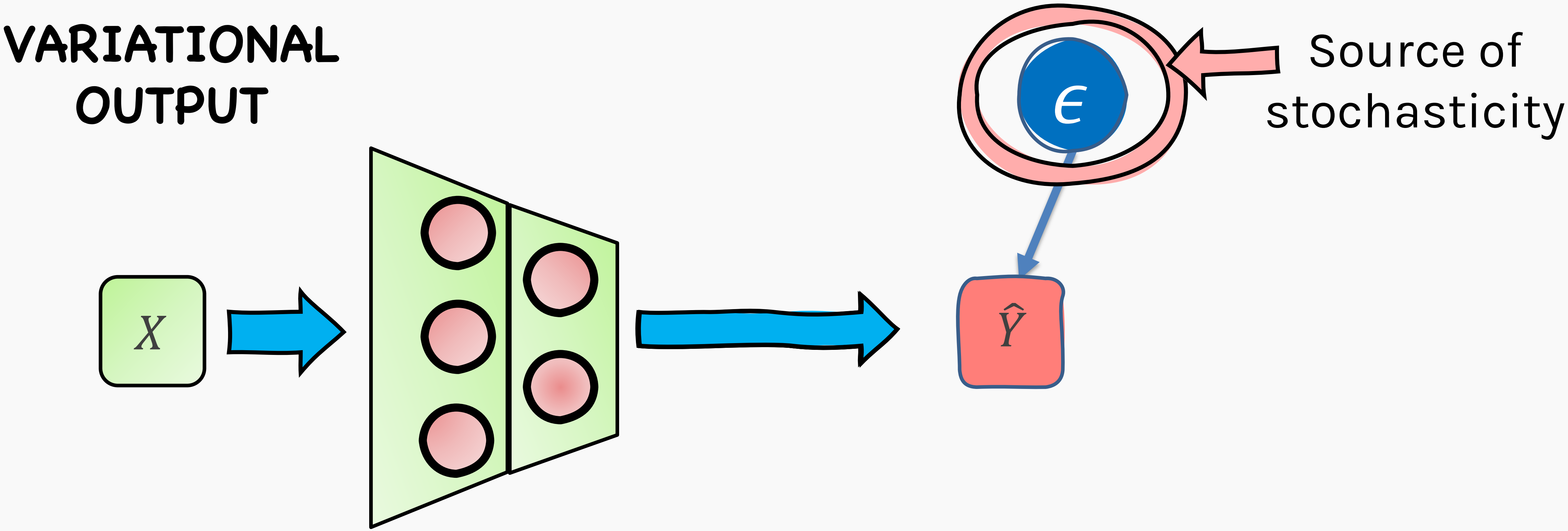
Variational Methods - Introduction



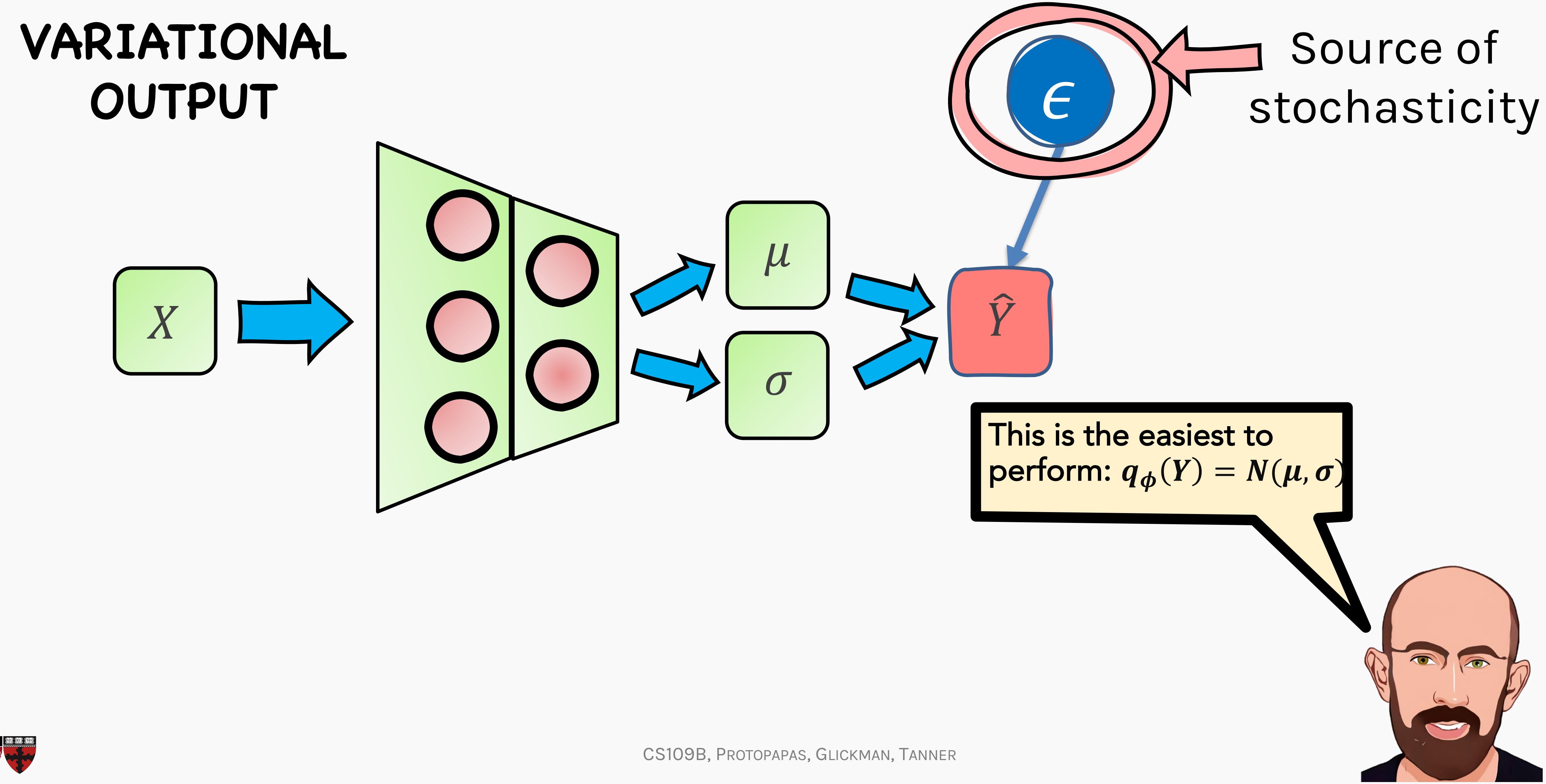
Back to the same dataset from part one

Option #1 – Variational Approximation of the output

Variational BNN – Output only

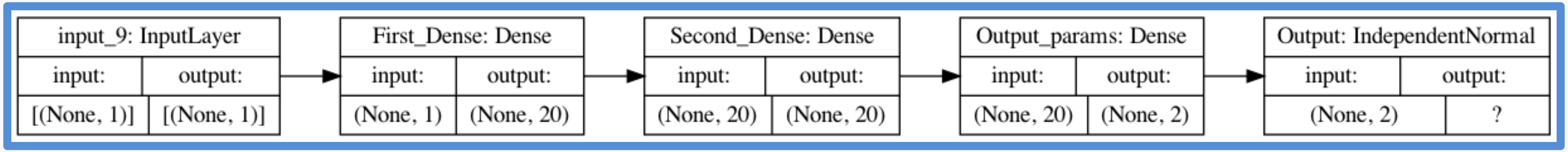


Variational BNN – Output only



Variational BNN – Output only

- We build a neural network like before, but instead of one output $\hat{y}$, we output two values, each representing the mean μ, and standard deviation σ.
- We introduce stochasticity by the equation $\hat{y} = \mu + \sigma \odot \epsilon$, thus for each input x, we have an output distribution given by $y \sim N(\mu, \sigma)$.
- We estimate (μ, σ), by minimizing the Variational Loss as before and doing backpropagation.

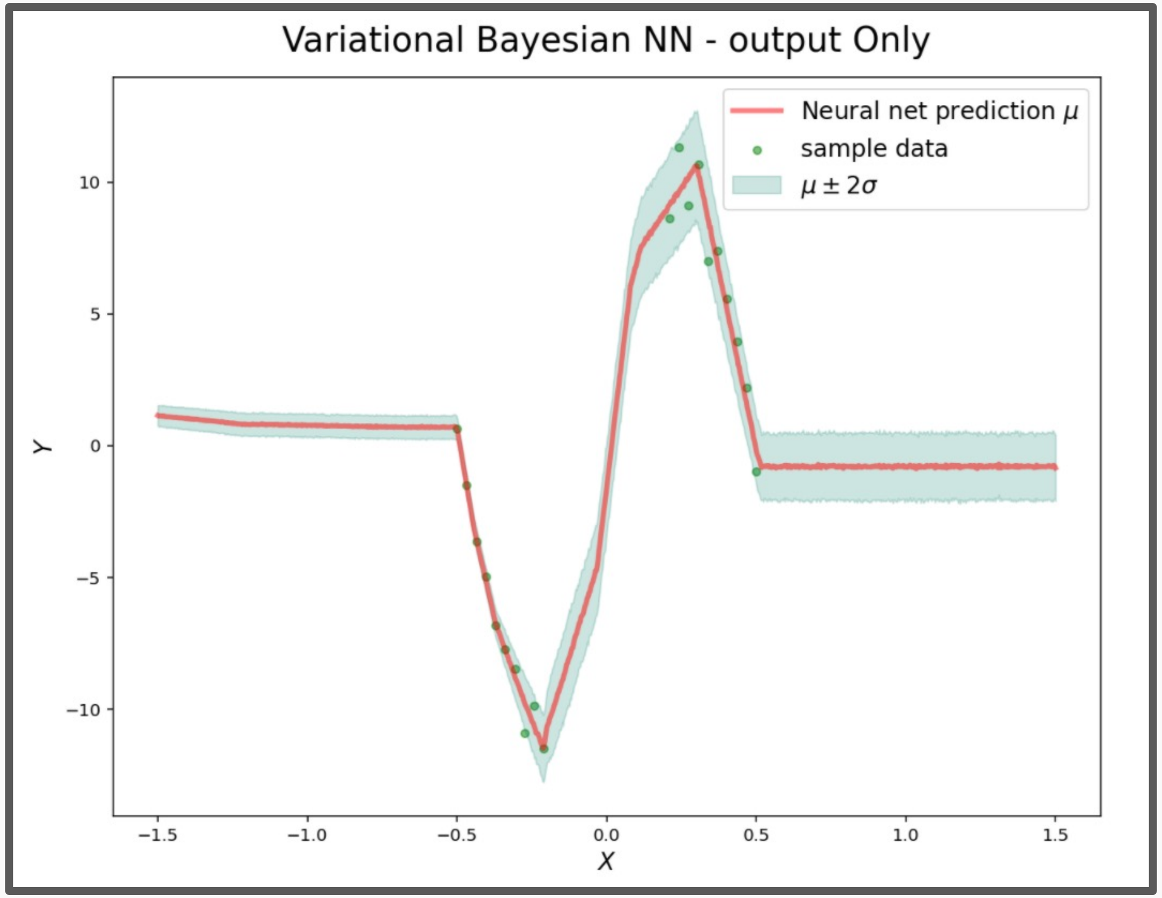


Model Summary

Variational BNN – Output only

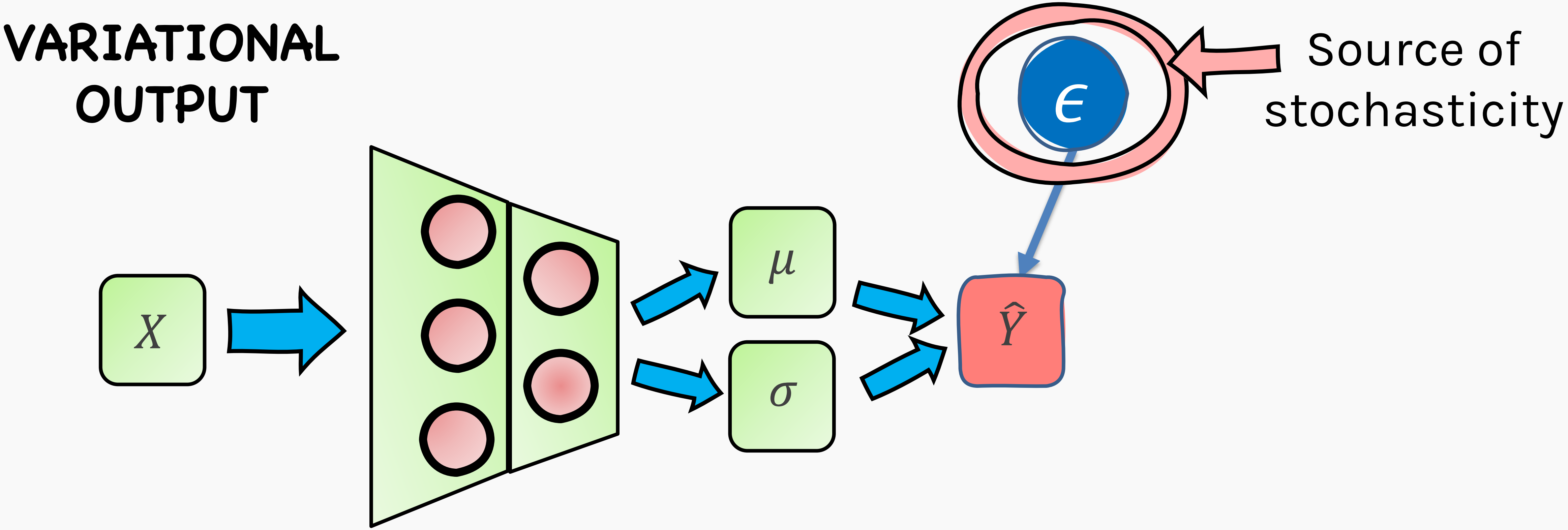
'OUTPUT ONLY' ISSUES?

- Although easy to implement, the approximate posterior $q(y|x)$ is not complex enough to capture the true posterior distribution $p(y|x)$.
- As seen in the output on our sample dataset, the epistemic variance away from the dataset should be much higher, but the model still confidently predicts those regions.

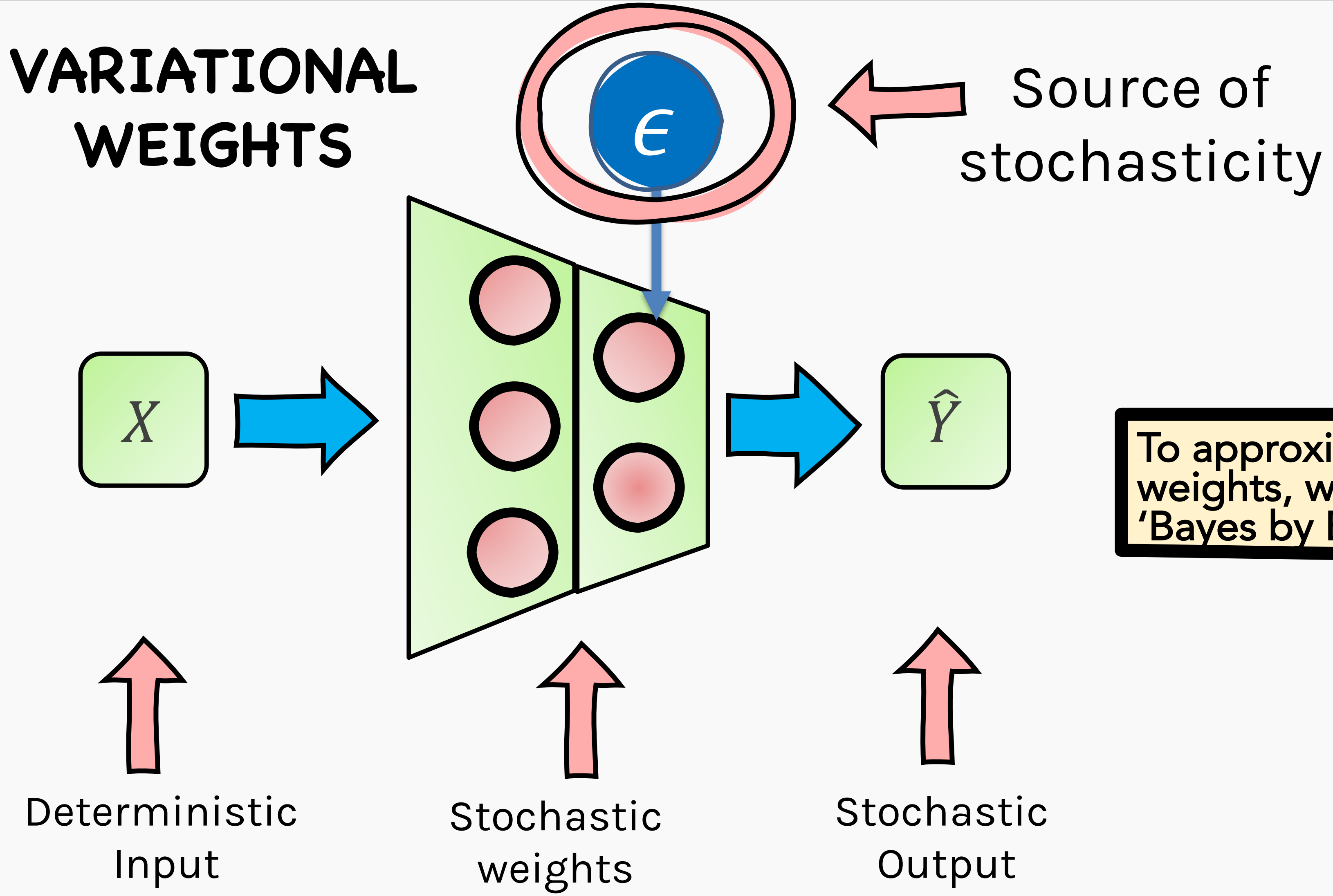


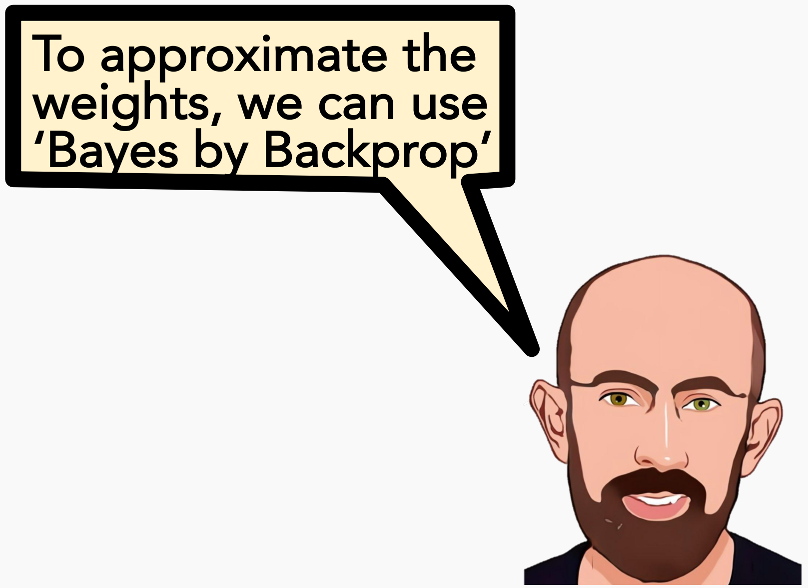
Option #2 – Variational Approximation of the weights

Variational BNN – Output only



Variational BNN – Bayes by Backprop





Variational BNN – Bayes by Backprop

- To perform variational approximation on the weights, instead of using a deterministic value for w, we let each $\mathrm{q}_{\mu_i,\sigma_i}(\mathrm{w_i}) = N(\mu_i, \sigma_i)$ and sample from these distributions.
- In the forward pass, we introduce stochasticity in the weights by using the equation $\boldsymbol{w} = \mu + \sigma \odot \epsilon$, and thus the output $\hat{y} = NN_W(x)$ will have an output distribution.
- In order to perform backpropagation, we modify our equations to take the derivate $\frac{\partial L}{\partial \mu}, \frac{\partial L}{\partial \sigma}$ and update the μ & σ.

[Weight Uncertainty in Neural Networks]

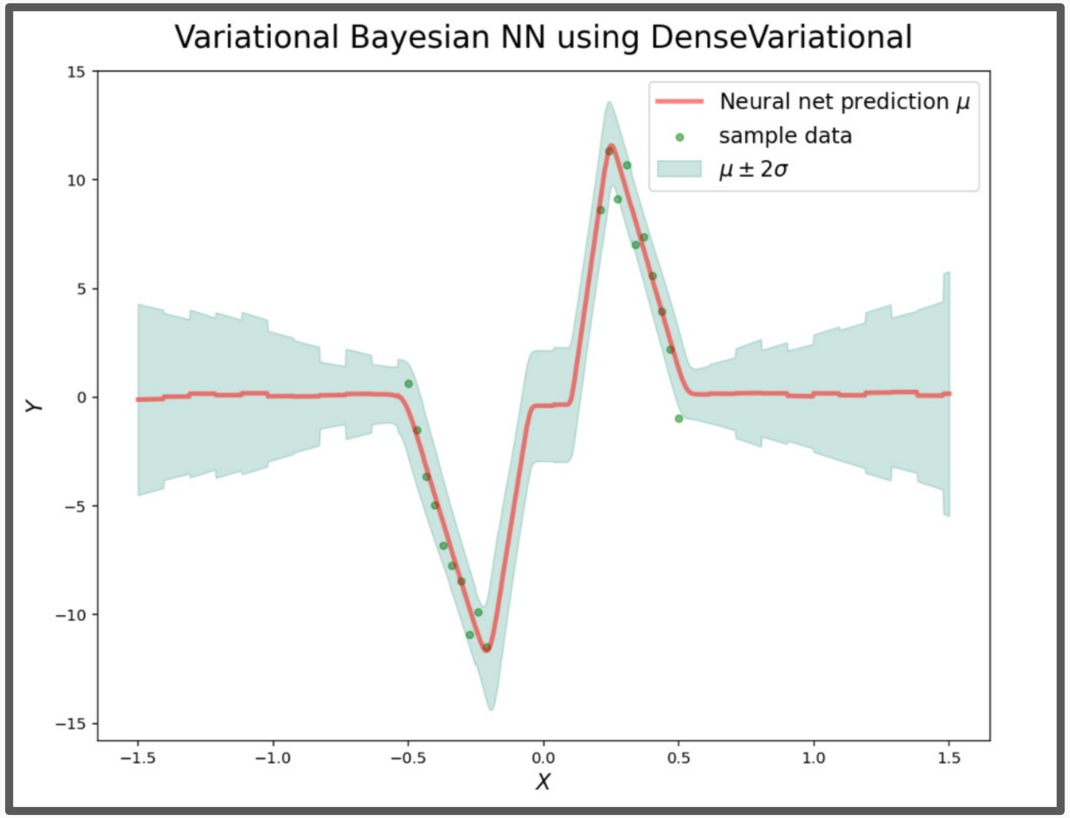
Variational BNN

'BAYES BY BACKPROP' ISSUES?

- Although the approximate posterior $q_\phi(w)$ adds sufficient complexity to the output posterior, it doubles the trainable parameters, which can be significant for very large neural networks.

- Since it is computationally prohibitive to sample a unique ϵ in each forward pass, the implementation uses the same sample for all weights.

- This causes the gradients to be correlated, thereby preventing variance reduction during training.



Variational BNN – Flipout

Not exactly different

- Like *Bayes By Backprop*, Flipout performs variational approximation on the weights $\mathrm{w} \sim \mathrm{q}_{\phi}(\mathrm{w}_{\mathrm{i}}) = N(\mu, \sigma)$.
- Unlike *Bayes By Backprop*, in the forward pass *Flipout* uses a different ϵ_i for each weight w_{i}. Like *Bayes By Backprop*, it introduces stochasticity in the weights by using the equation $\boldsymbol{w} = \mu + \sigma \odot \epsilon$, and thus the output $\hat{y} = NN(w, x)$ will have an output distribution.
- *Flipout* overcomes the computational difficulty of a unique sampling by multiplying the sample ϵ with a random sign matrix.

[Flipout: Efficient Pseudo-Independent Weight Perturbations on Mini-Batches]

Uncorrelated stochastic gradients

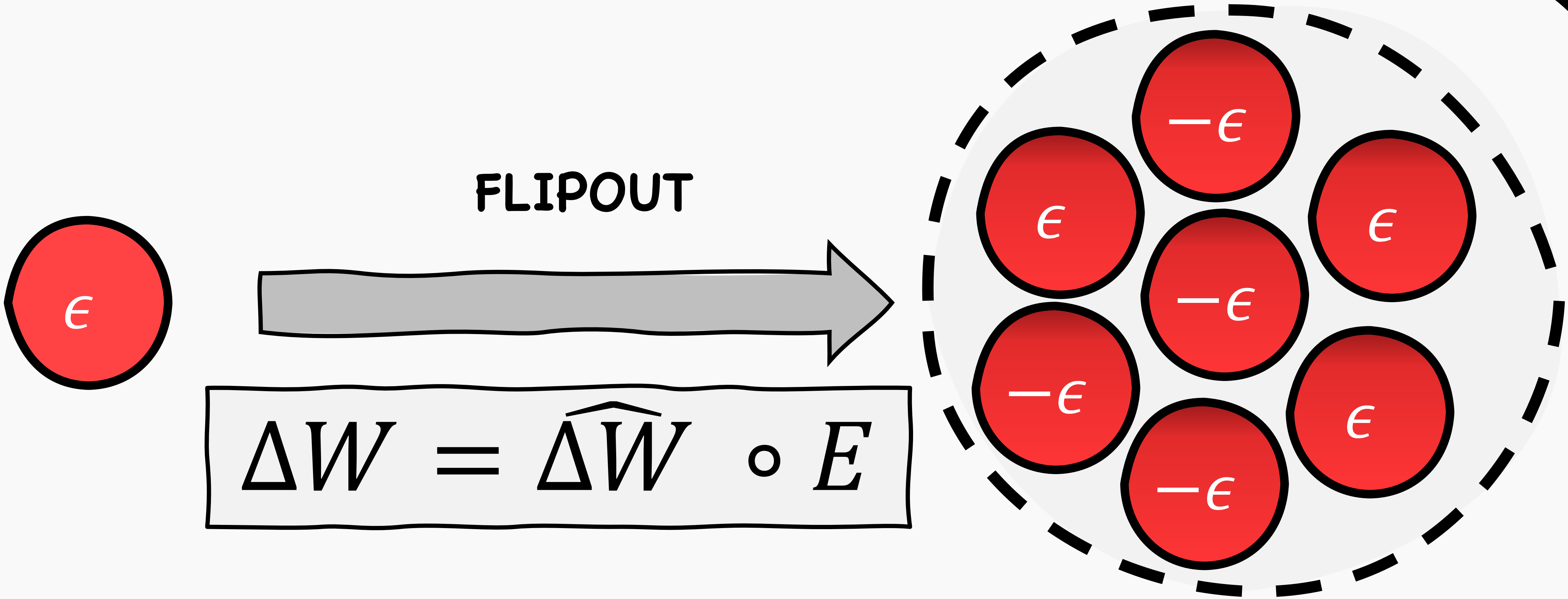
Observation 1. Let $q_{\boldsymbol{\theta}}$ be a perturbation distribution that satisfies the above assumptions, and let $\widehat{\Delta W} \sim q_{\boldsymbol{\theta}}$. Let E be a random sign matrix that is independent of $\widehat{\Delta W}$. Then $\Delta W = \widehat{\Delta W} \circ E$ is identically distributed to $\widehat{\Delta W}$. Furthermore, the loss gradients computed using ΔW are identically distributed to those computed using $\widehat{\Delta W}$.

$$\Delta W = \widehat{\Delta W} \circ E$$

Some perturbation distributions

Random Sign matrix

Flipout Explained



$$\Delta W = \widehat{\Delta W} \circ E$$

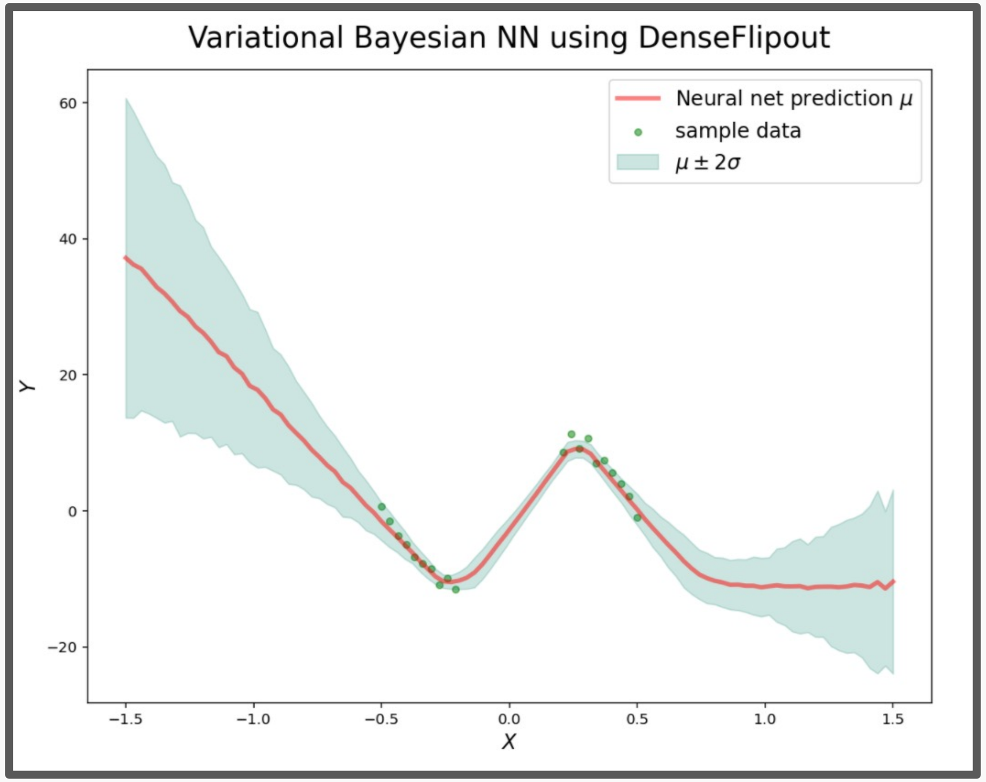
REPEAT FOR EVERY MINIBATCH

Variational BNN – Output only

'FLIPOUT' ISSUES?

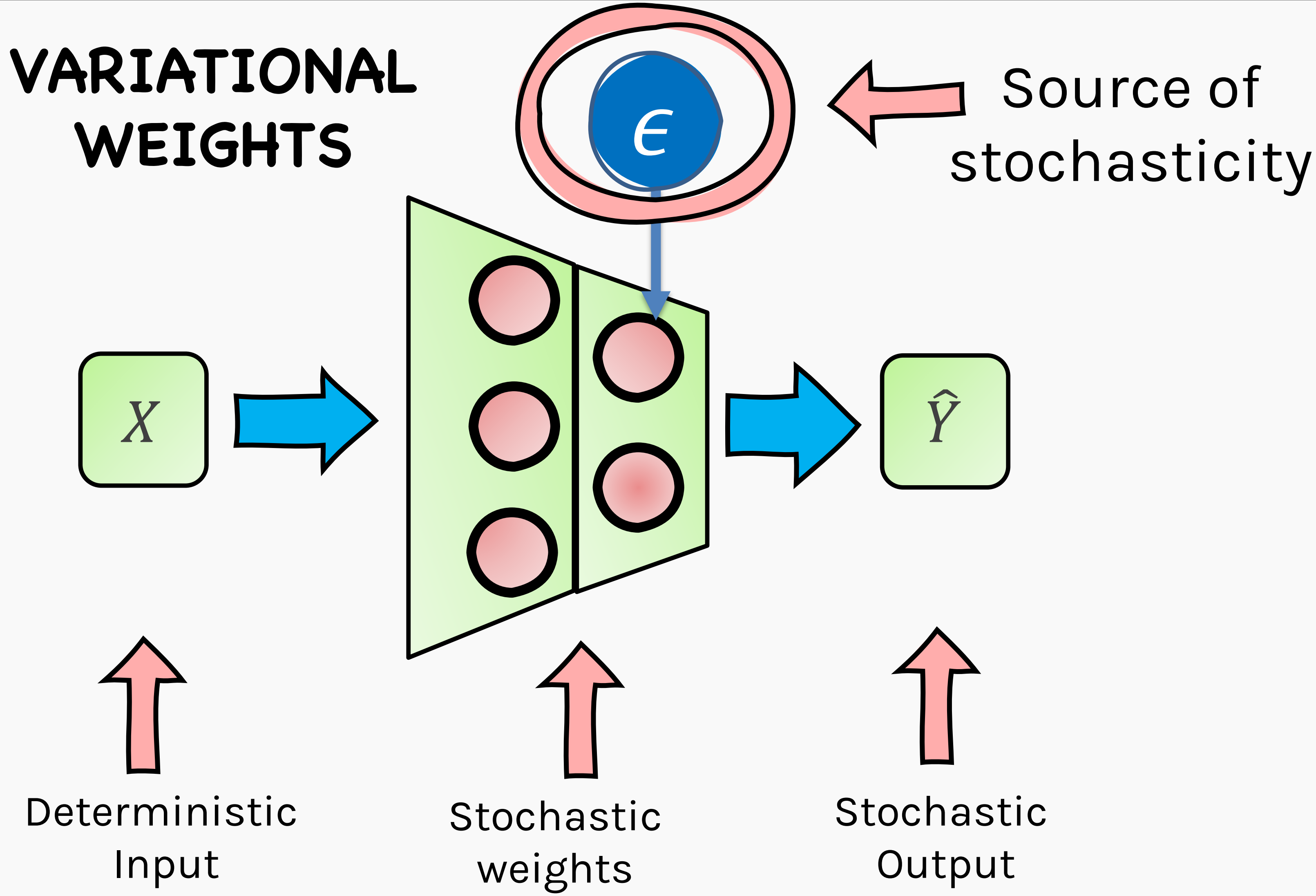
- Although *Flipout* is a significant improvement over Bayes By Backprop, *Flipout* still suffers from some degree of correlation between the stochastic gradients, and performance suffers from an increased number of weights.

- In order to get uncorrelated stochastic gradients, we will need to sample independently for each weight for each forward pass.

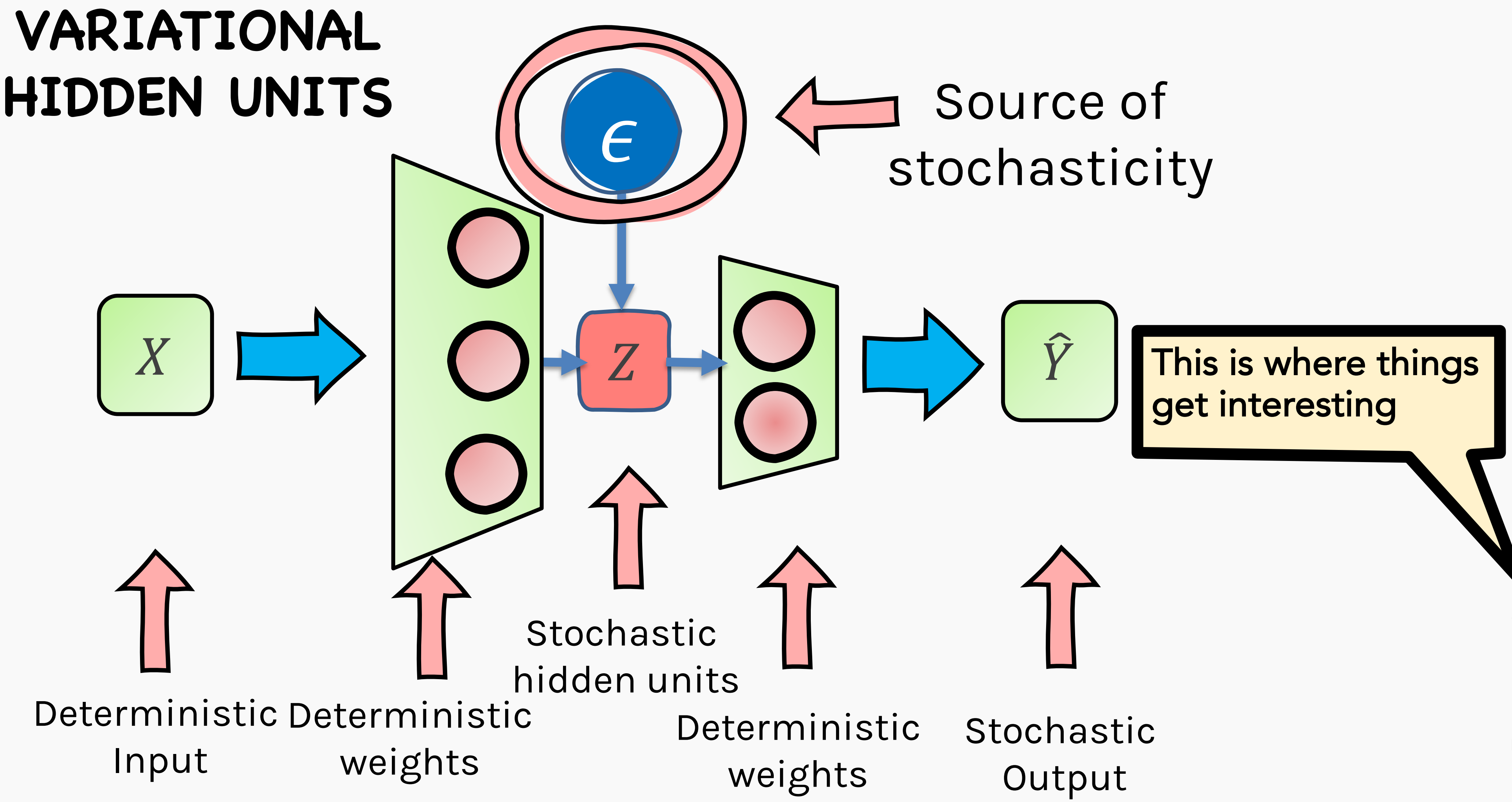


Option #3 – Variational Approximation on hidden units

Variational BNN – Stochastic Weights

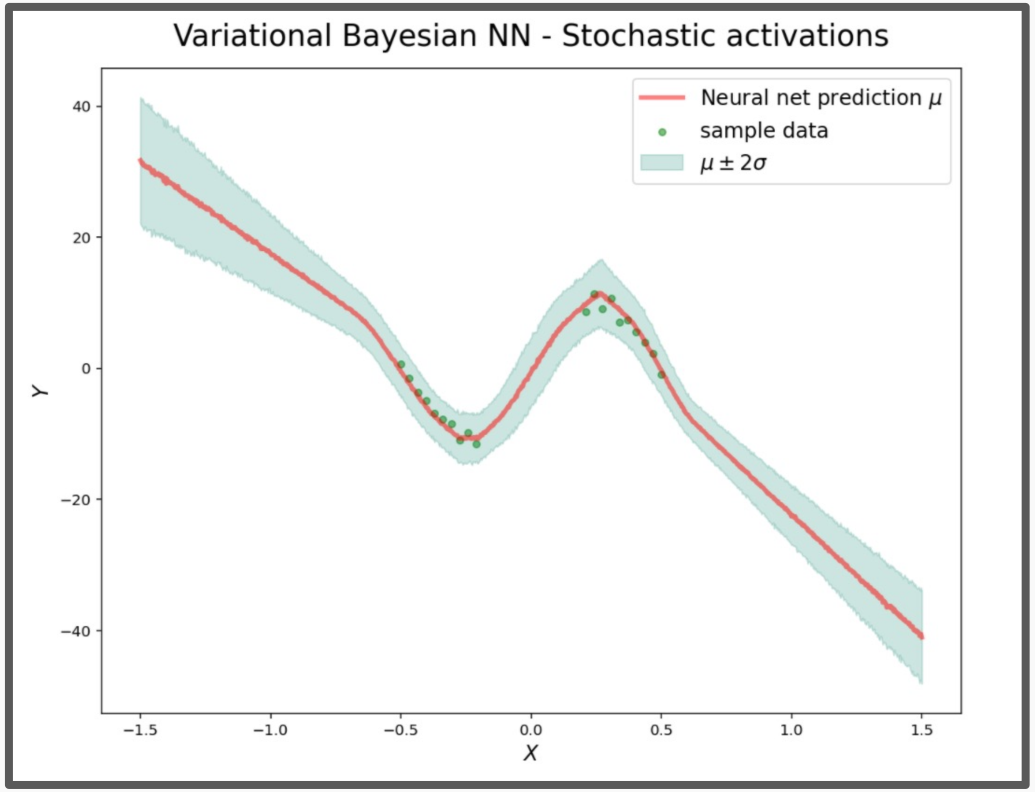


Quick Review



Variational BNN – Stochastic hidden units

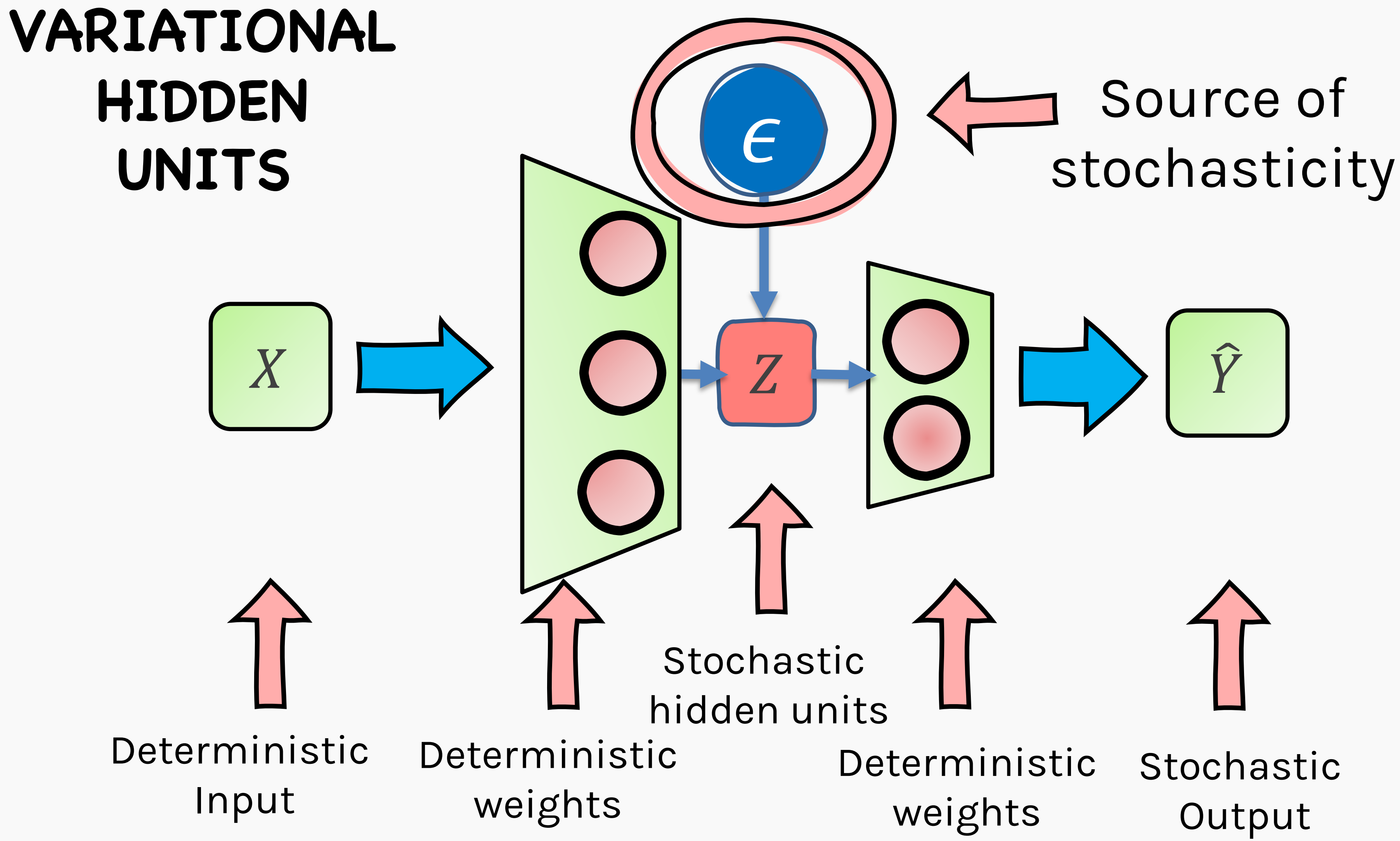
- Instead of using stochastic weights, we could approximate the true posterior $p(z|x)$ with the approximate posterior $q(z|x)$ with a known distribution such as a Gaussian.
- Since number of hidden units are an order of two lesser than the weights in a network, we can easier sample for each hidden unit in the forward pass.



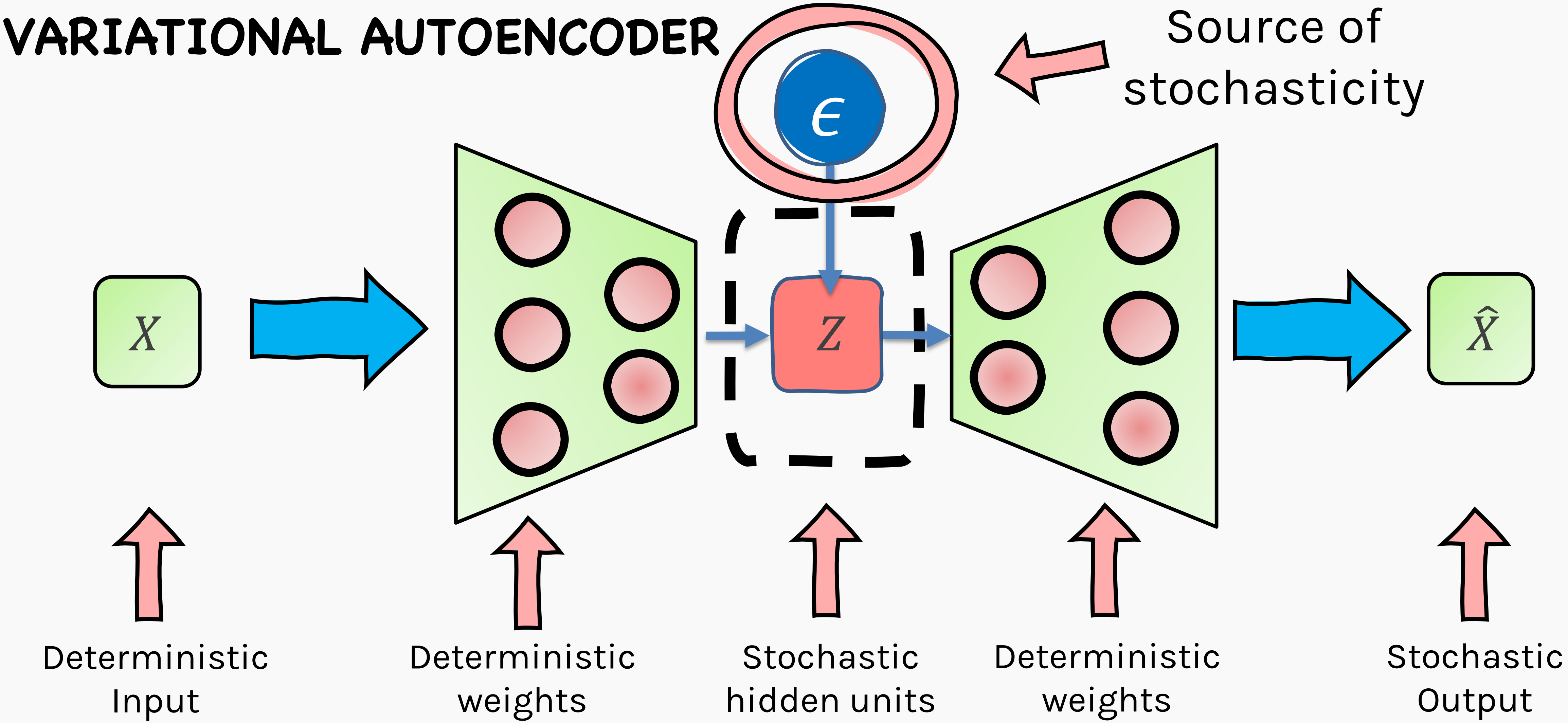

BAYESIAN
Linear regression
BAYESIAN
RAL NETWORKS
Now let's get to the treasure chest
VARIATIONAL
AUTO-ENCODERS
VARIATIONAL
METHODS
N
W
E
S

Variational Auto-Encoders

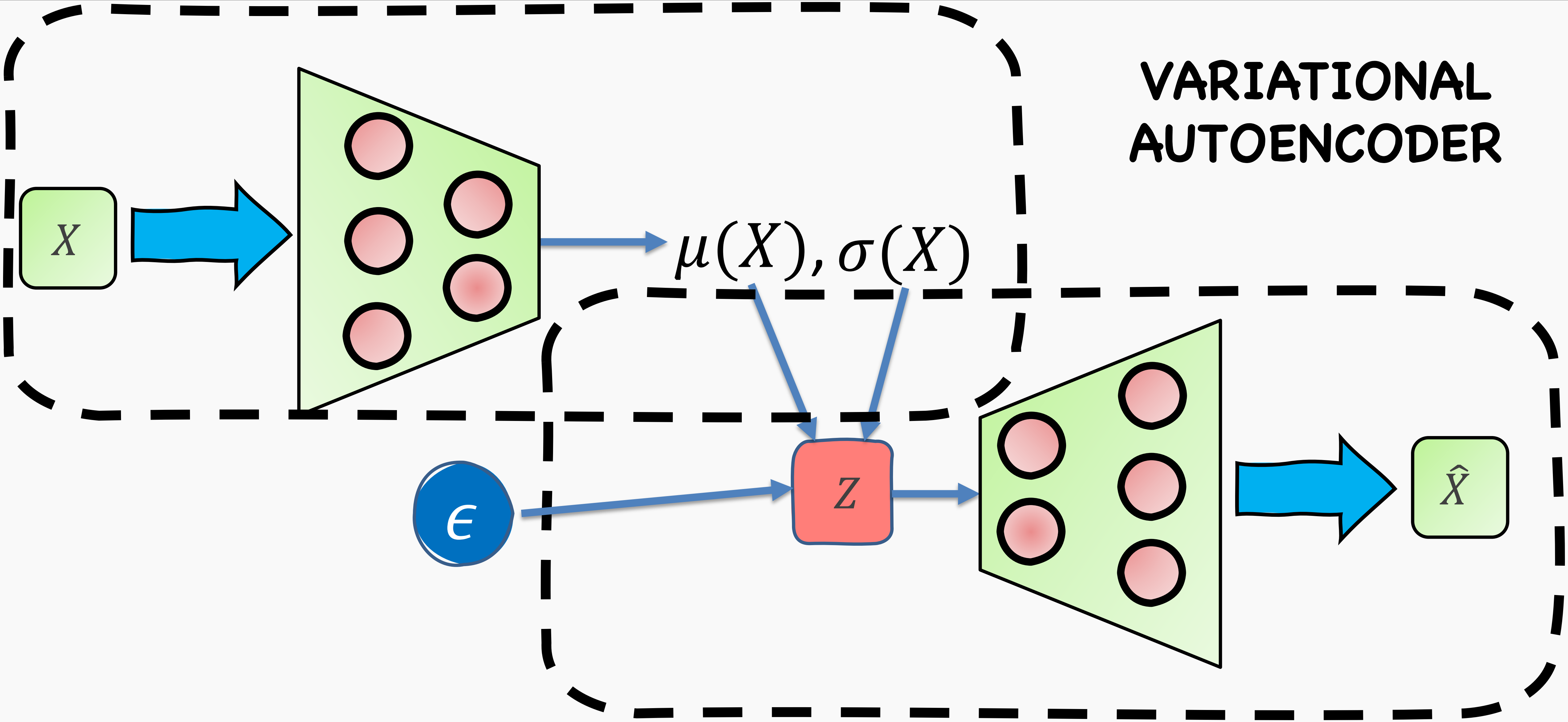
Quick Review



Quick Review



Variational AutoEncoder



ESTIMATED LOWER BOUND

Log-likelihood

$q_\phi(z|x^{(i)})$ should resemble the prior $p_\theta(z)$

Reconstruction Term

$$\mathcal{L}oss\left(\boldsymbol{\theta}, \phi; \mathbf{x}^{(i)}\right) = KL\left(q_{\boldsymbol{\phi}}\left(\mathbf{z} \mid \mathbf{x}^{(i)}\right) \| p(\mathbf{z})\right) - \mathbb{E}_{q_\phi\left(\mathbf{z}|\mathbf{x}^{(i)}\right)}\left[\log p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)} \mid \mathbf{z}\right)\right]$$

Sample Lower Bound

ESTIMATED LOWER BOUND

θ are the decoder weights

$q_\phi(z|x^{(i)})$ should resemble the prior $p_\theta(z)$

Reconstruction Term

$$\mathcal{L}oss\left(\boldsymbol{\theta}, \phi; \mathbf{x}^{(i)}\right) = KL\left(q_{\boldsymbol{\phi}}\left(\mathbf{z} \mid \mathbf{x}^{(i)}\right) \| p(\mathbf{z})\right) - \mathbb{E}_{q_\phi\left(\mathbf{z}|\mathbf{x}^{(i)}\right)}\left[\log p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)} \mid \mathbf{z}\right)\right]$$

Set $q_\phi(z|x^{(i)})$ to be $N(\mu, \sigma)$

Set the priors to $p(z) = N(0,1)$

If likelihood is normal this term becomes:
$-\sum\left(\hat{x}^{(i)} - x^{(i)}\right)^2$

If likelihood is Bernoulli this term becomes the binary cross entropy:
$-\sum x^{(i)} \log \hat{x}^{(i)} + x^{(i)} \log \hat{x}^{(i)}$

Training a VAE

1. Set priors: $p(z) = N(0,1)$
2. Forward pass with sampling: $z = \mu + \sigma \odot \epsilon$
3. Calculate the loss:

$$\mathcal{L}oss\left(\boldsymbol{\theta}, \phi; \mathbf{x}^{(i)}\right) = KL\left(q_{\boldsymbol{\phi}}\left(\mathbf{z} \mid \mathbf{x}^{(i)}\right) \| p(\mathbf{z})\right) - \mathbb{E}_{q_\phi\left(\mathbf{z}|\mathbf{x}^{(i)}\right)}\left[\log p_{\boldsymbol{\theta}}\left(\mathbf{x}^{(i)} \mid \mathbf{z}\right)\right]$$

4a. Update the decoder weights using backpropagation

4b. Update the encoder weights using backpropagation

Note: If priors are $N(0,1)$ and $q_\phi\left(z, x^{(i)}\right)$ is also normal, the KL can be analytically calculated.


BAYESIAN
Linear regression
BAYESIAN
NEURAL NETWORKS
Yo-ho-ho!
VARIATIONAL
AUTO-ENCODERS
N
W
E
S
VARIATIONAL
METHODS

Inference Summary

Summary

COMPLEXITY

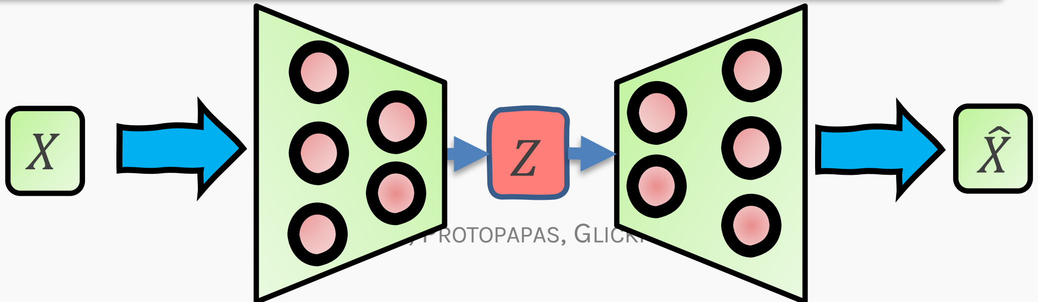
FLIPOUT

PICK TWO

MCMC

SPEED

STOCHASTICITY



RECAP: Variational AutoEncoder Paper

Let us consider some dataset $X = \{x^{(i)}\}_{i=1}^{N}$ consisting of N i.i.d. samples of some continuous or discrete variable x. We assume that the data are generated by some random process, involving an unobserved continuous random variable z.

...

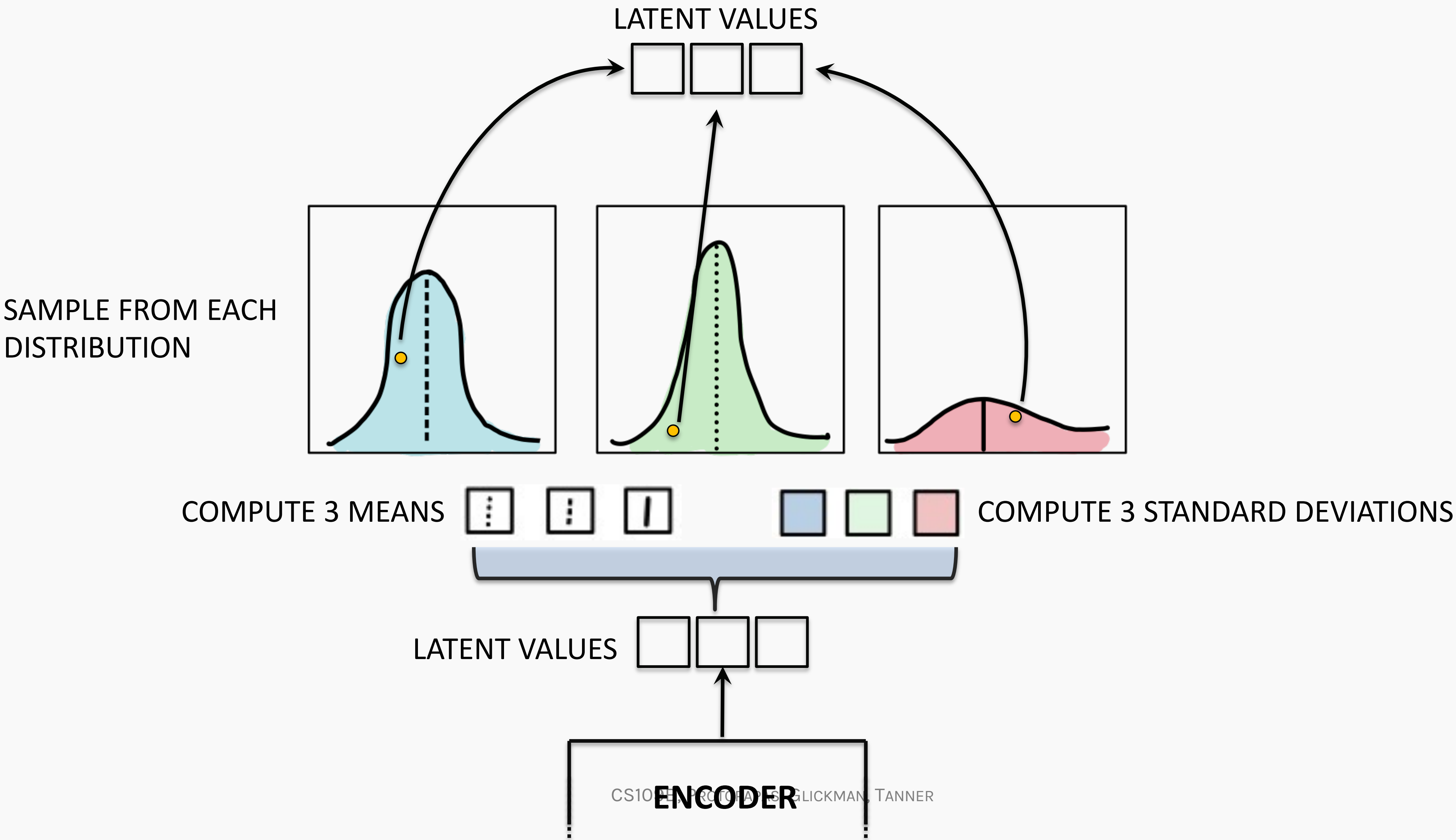
We are interested in a general algorithm that works in case of:

1. *Intractability:* $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$ is **intractable**
2. *Large Dataset:* Sampling based solutions eg. Monte Carlo would be too slow

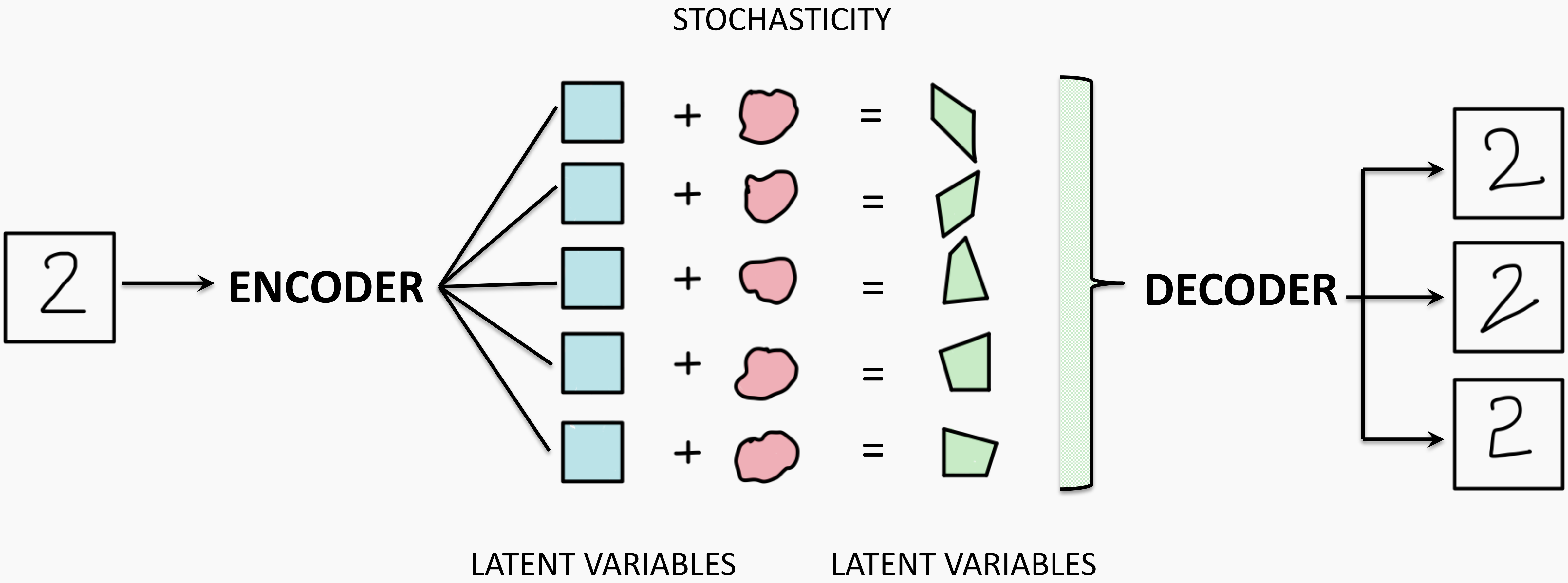
[Auto-Encoding Variational Bayes (Diederik P. Kingma et al)]()

VAE as a generative model

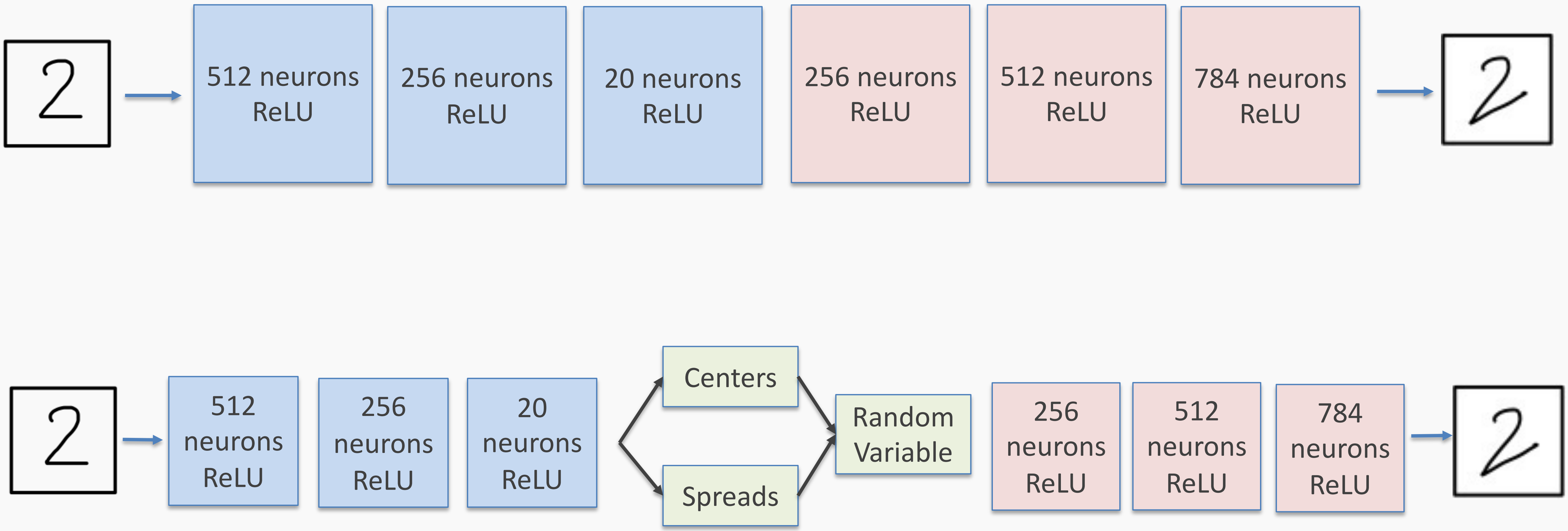
Variational Autoencoders – Generative models



Variational Autoencoders



Variational Autoencoders

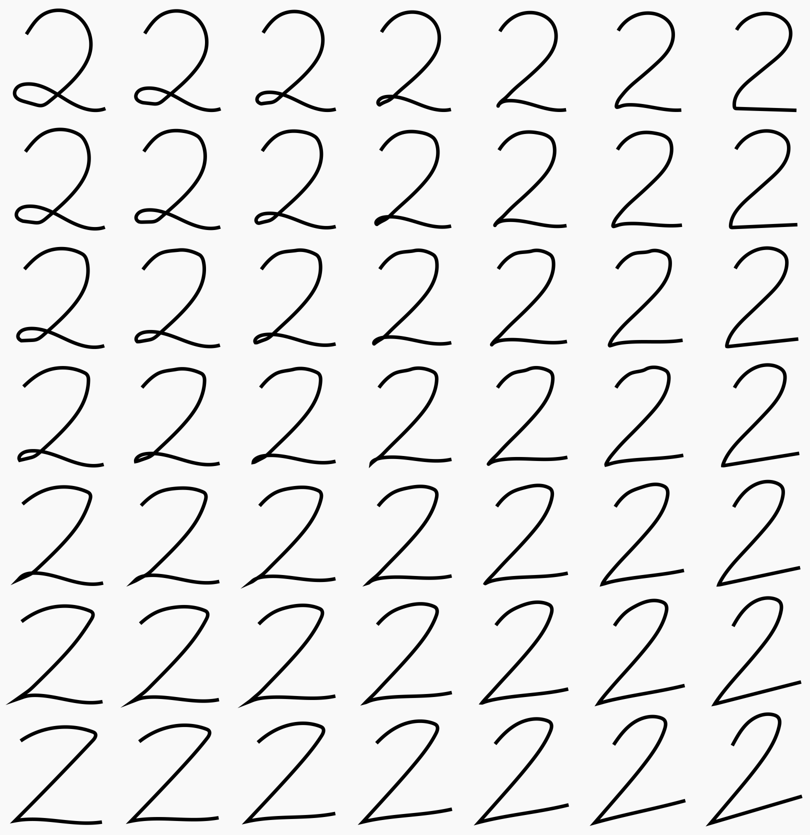


Separability in Variational Autoencoders

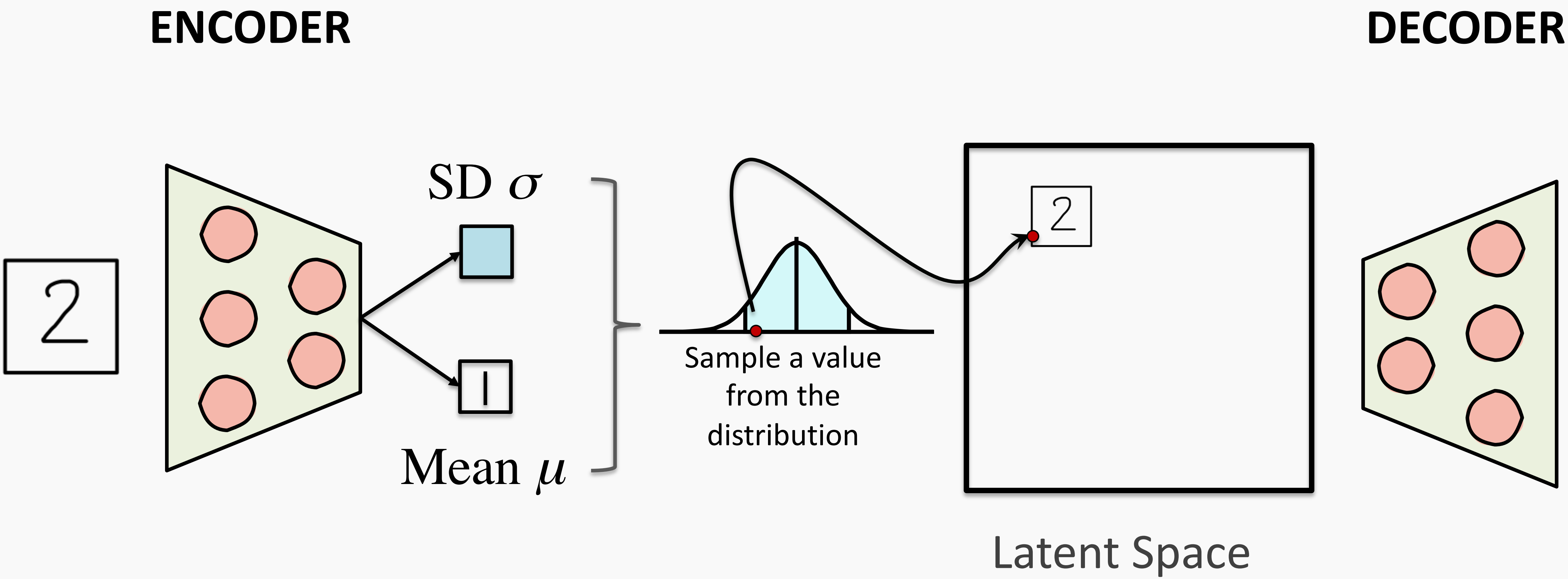
Separability is not only between classes, but we also want similar items in the same class to be near each other.

For example, there are different ways of writing "2"; we want similar styles to end up near each other.

Let us examine VAE; there is something magical happening once we add stochasticity in the latent space.

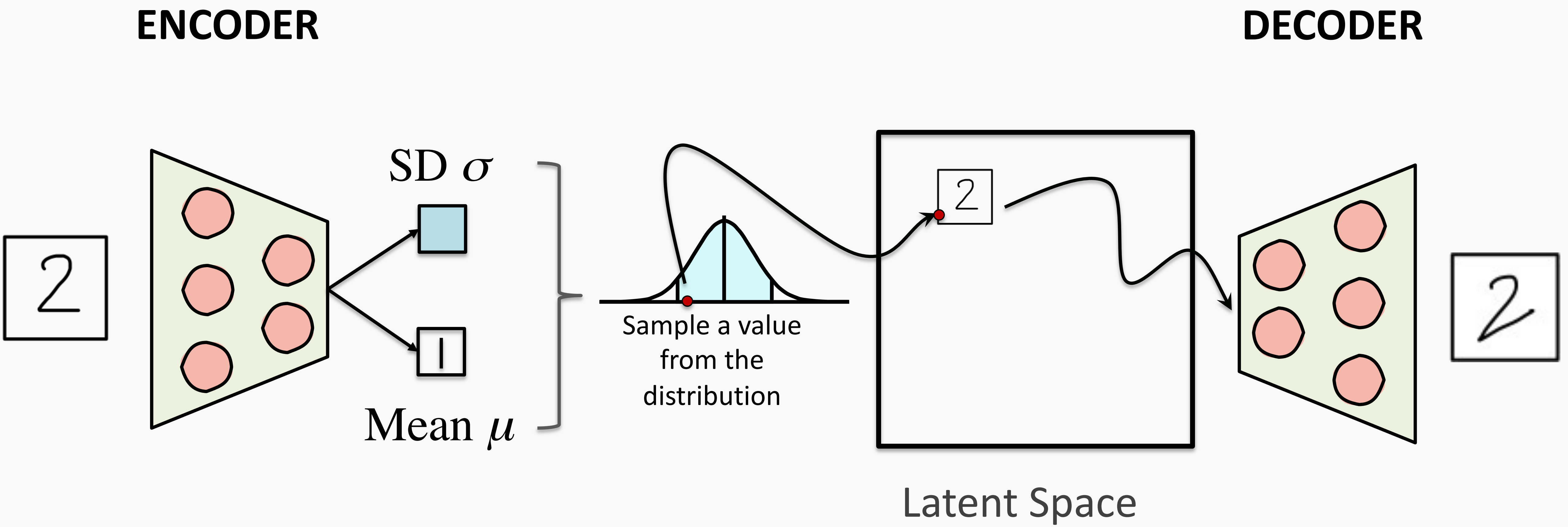


Separability in Variational Autoencoders



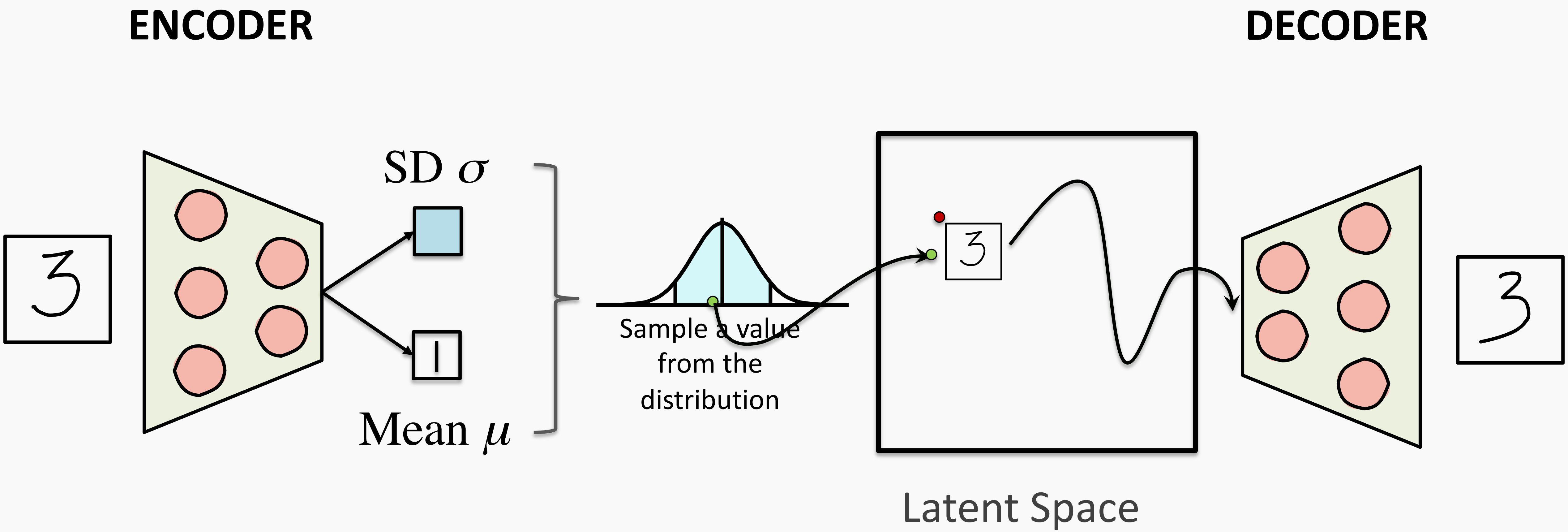
Encode the first sample (a "2") and find μ_1, σ_1. Sample $z_1 \sim N(\mu_1, \sigma_1)$ and decode to $\hat{x}_1$

Separability in Variational Autoencoders



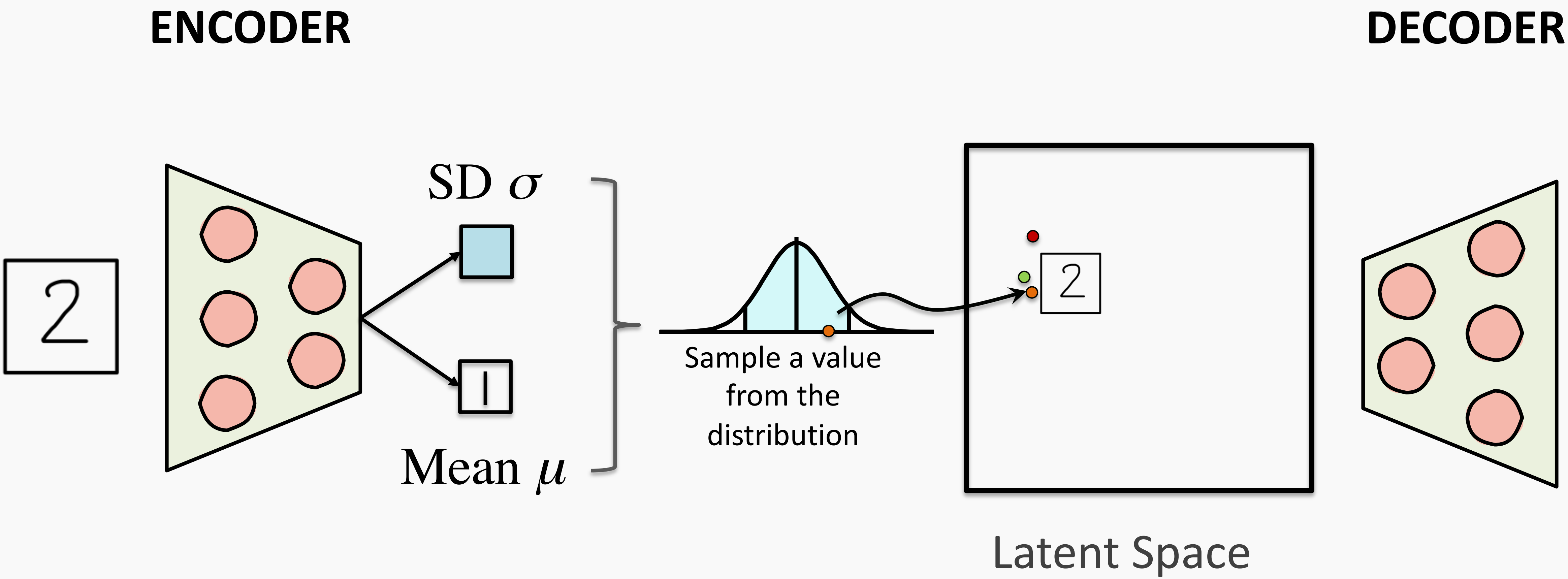
Decode to $\hat{x}_1$

Separability in Variational Autoencoders



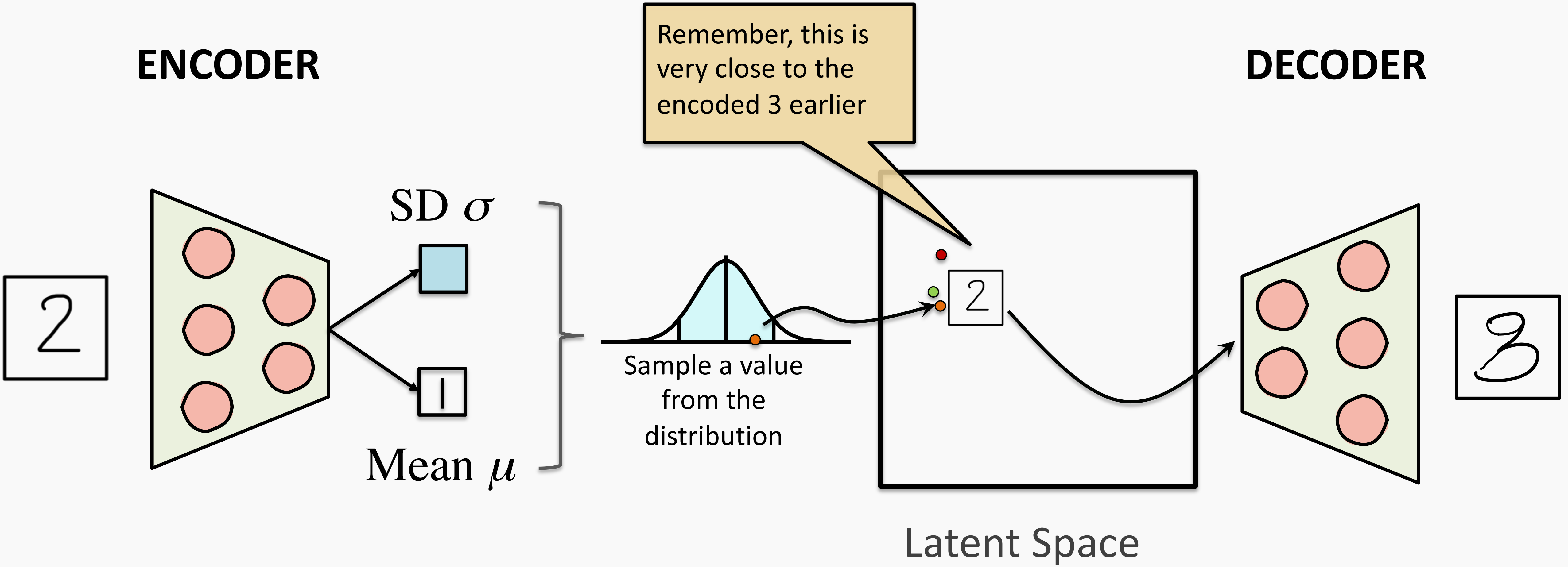
Encode the second sample (a "3") find μ_2, σ_2. Sample $z_2 \sim N(\mu_2, \sigma_2)$ and decode to $\hat{x}_2$

Separability in Variational Autoencoders



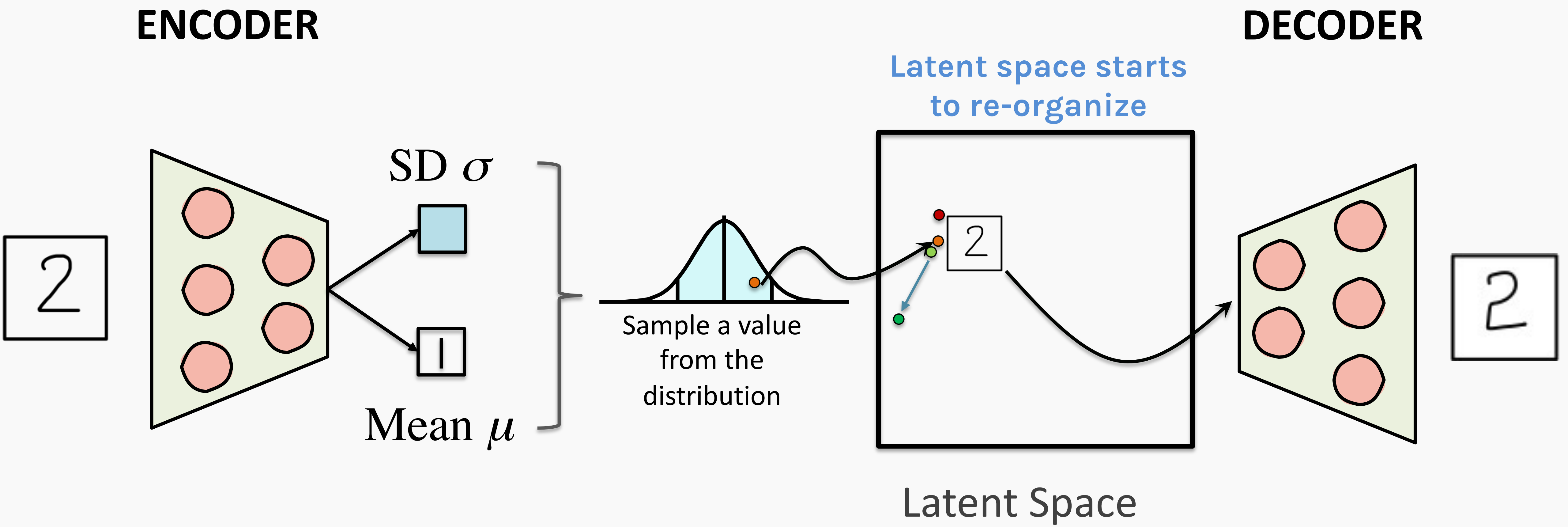
Train with the first sample (a "2") again and find μ_1, σ_1. However $z_1 \sim N(\mu_1, \sigma_1)$ **will not be the same**.

Separability in Variational Autoencoders



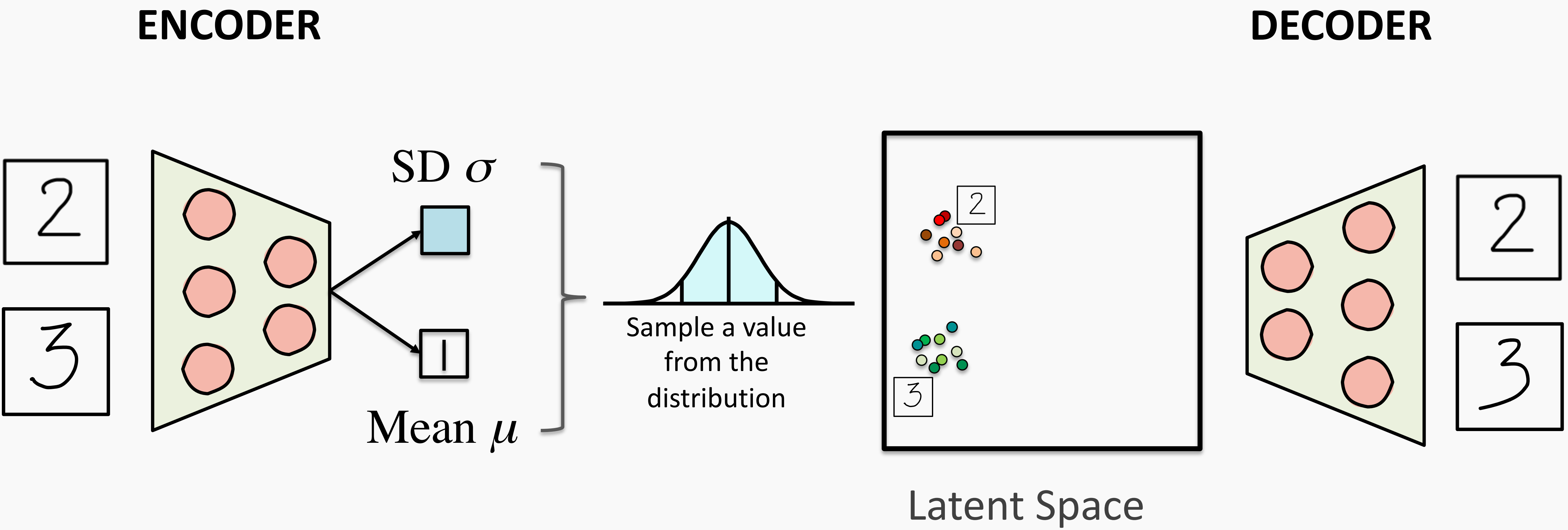
Train with the first sample (a "2") again and find μ_1, σ_1. However $z_1 \sim N(\mu_1, \sigma_1)$ **will not be the same**. *It happens to be close to the "3" in latent space.*

Separability in Variational Autoencoders



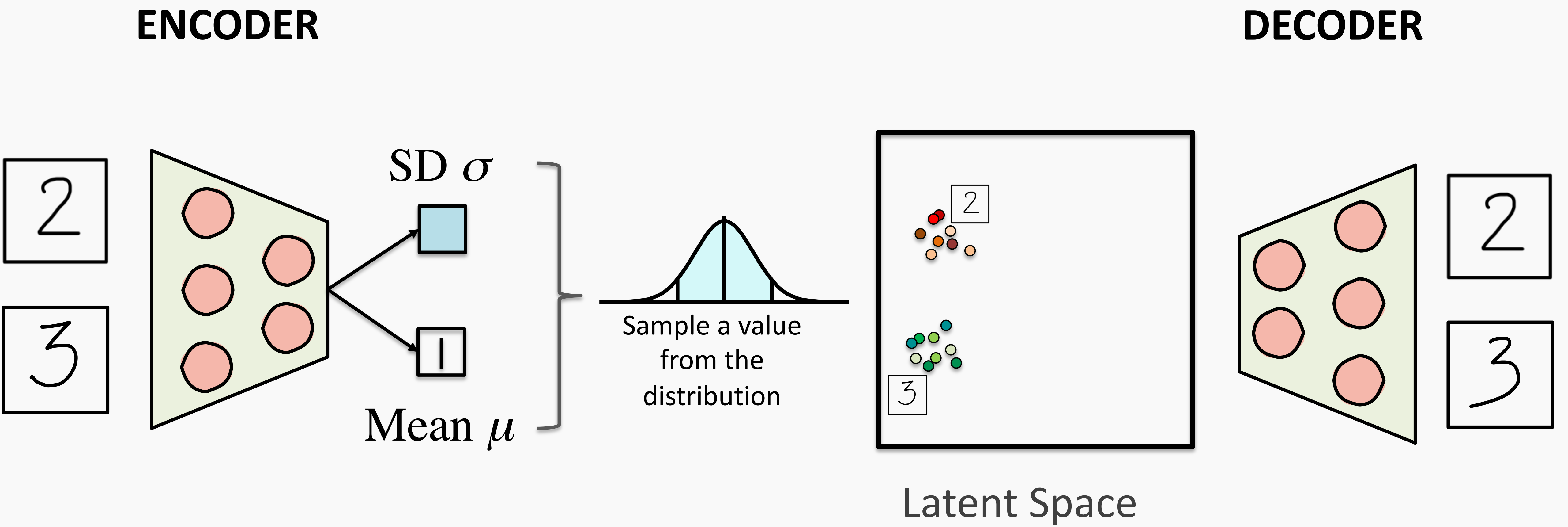
Train with 1st sample again.

Separability in Variational Autoencoders



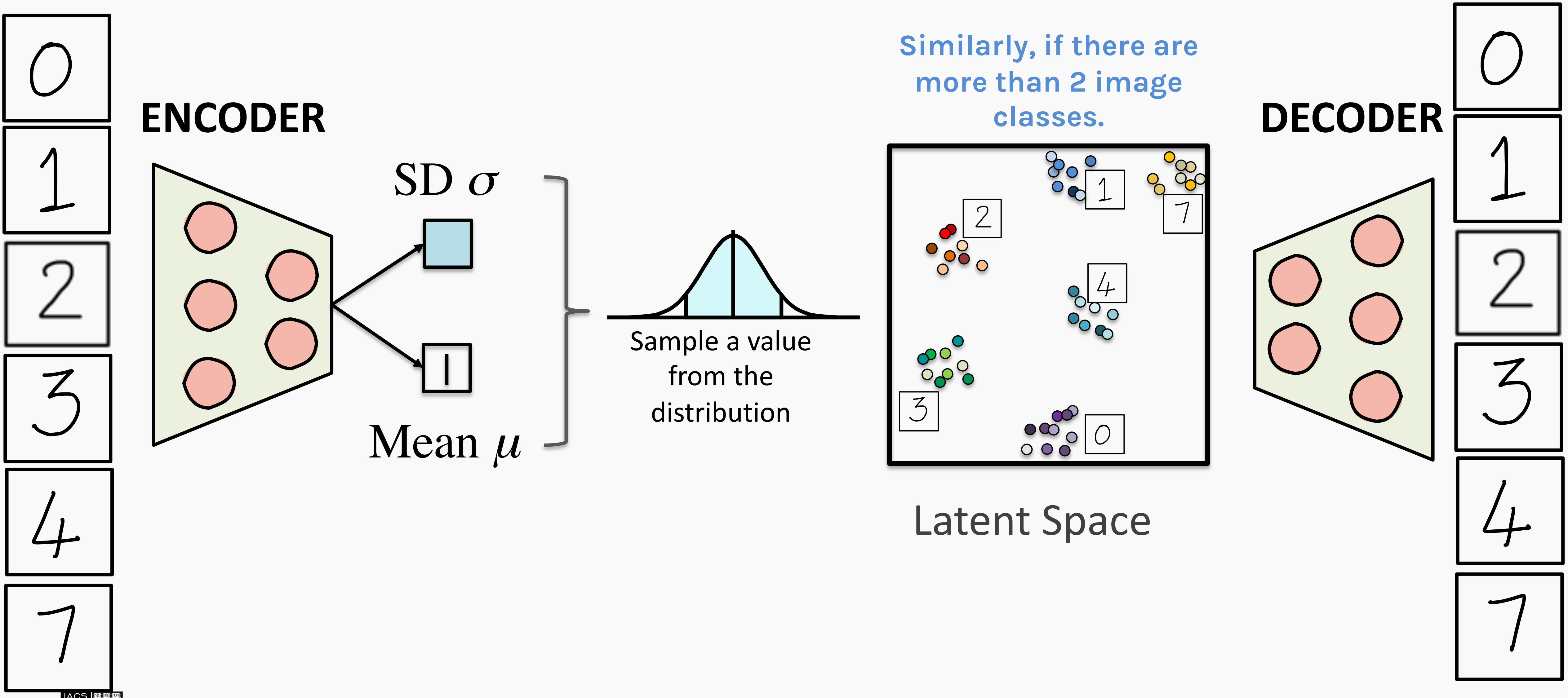
Keep doing this multiple times with 2's and 3's

Separability in Variational Autoencoders

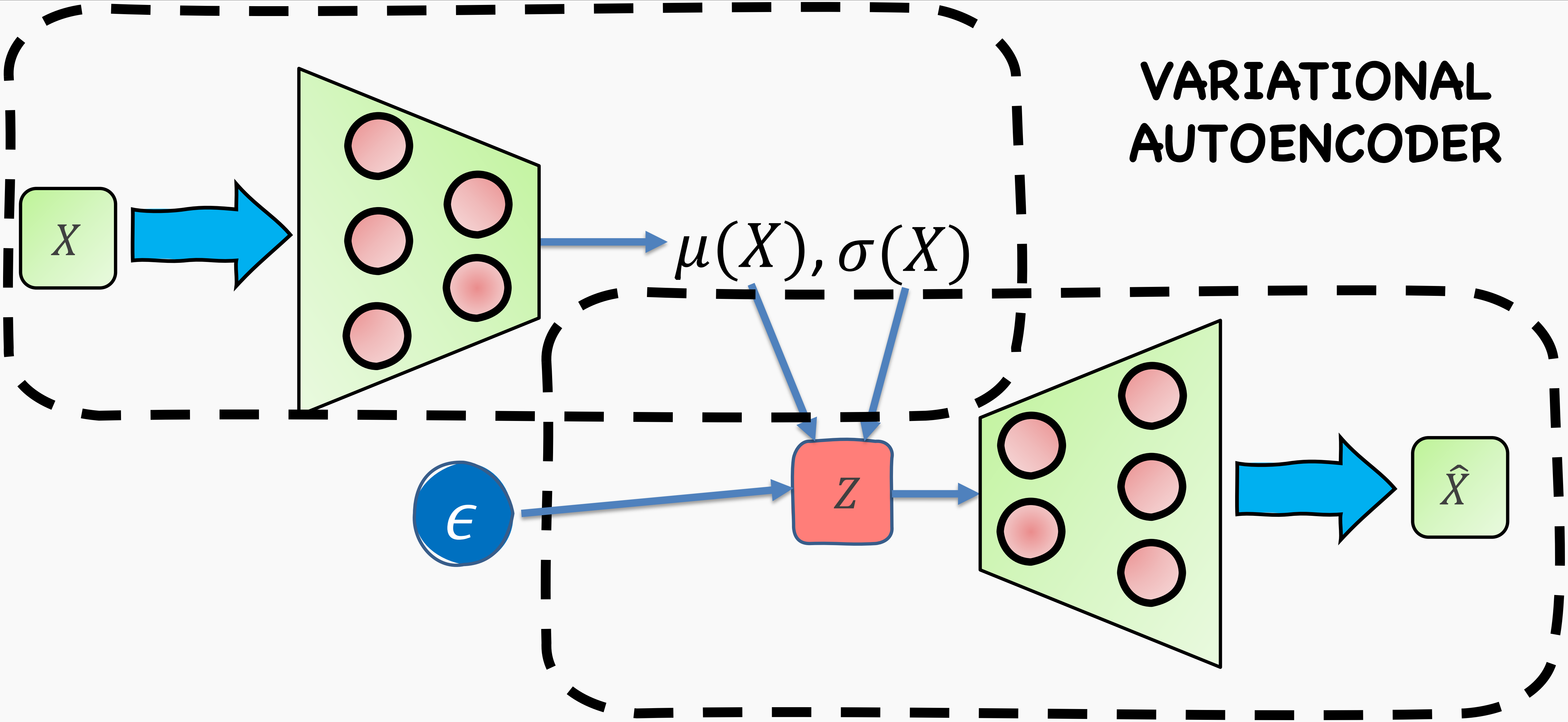


Soon images belonging to different classes are separated and images within a class are clustered together.

Separability in Variational Autoencoders

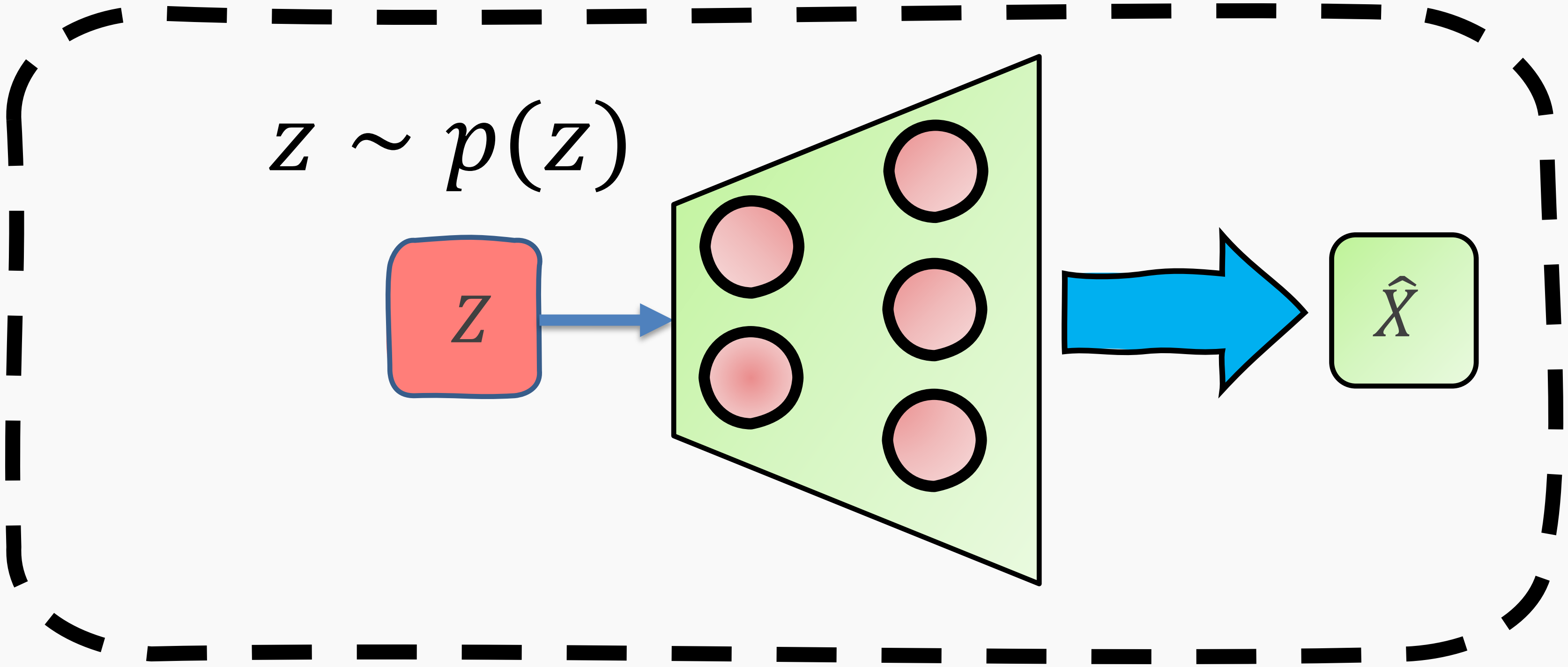


Variational AutoEncoder



Variational AutoEncoder as a generative model

Generative model

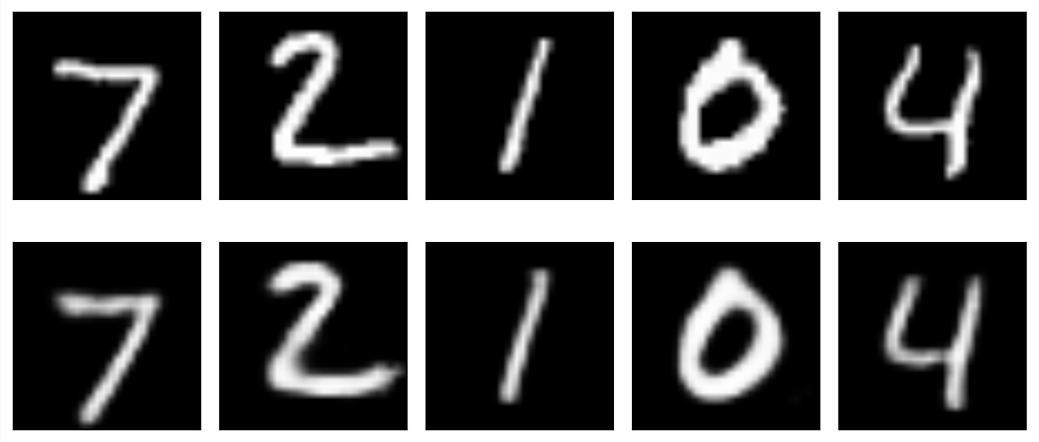


Training VAE

Traditional AE:

Input Image:

Output Images:

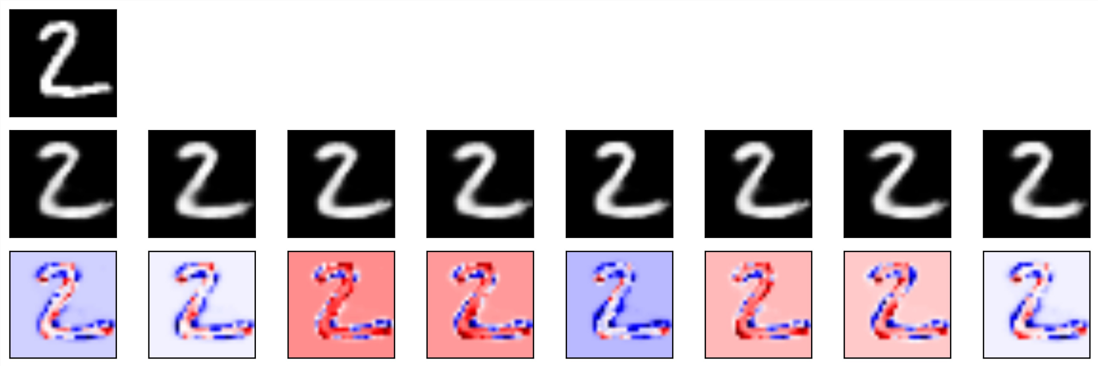


Variational AE:

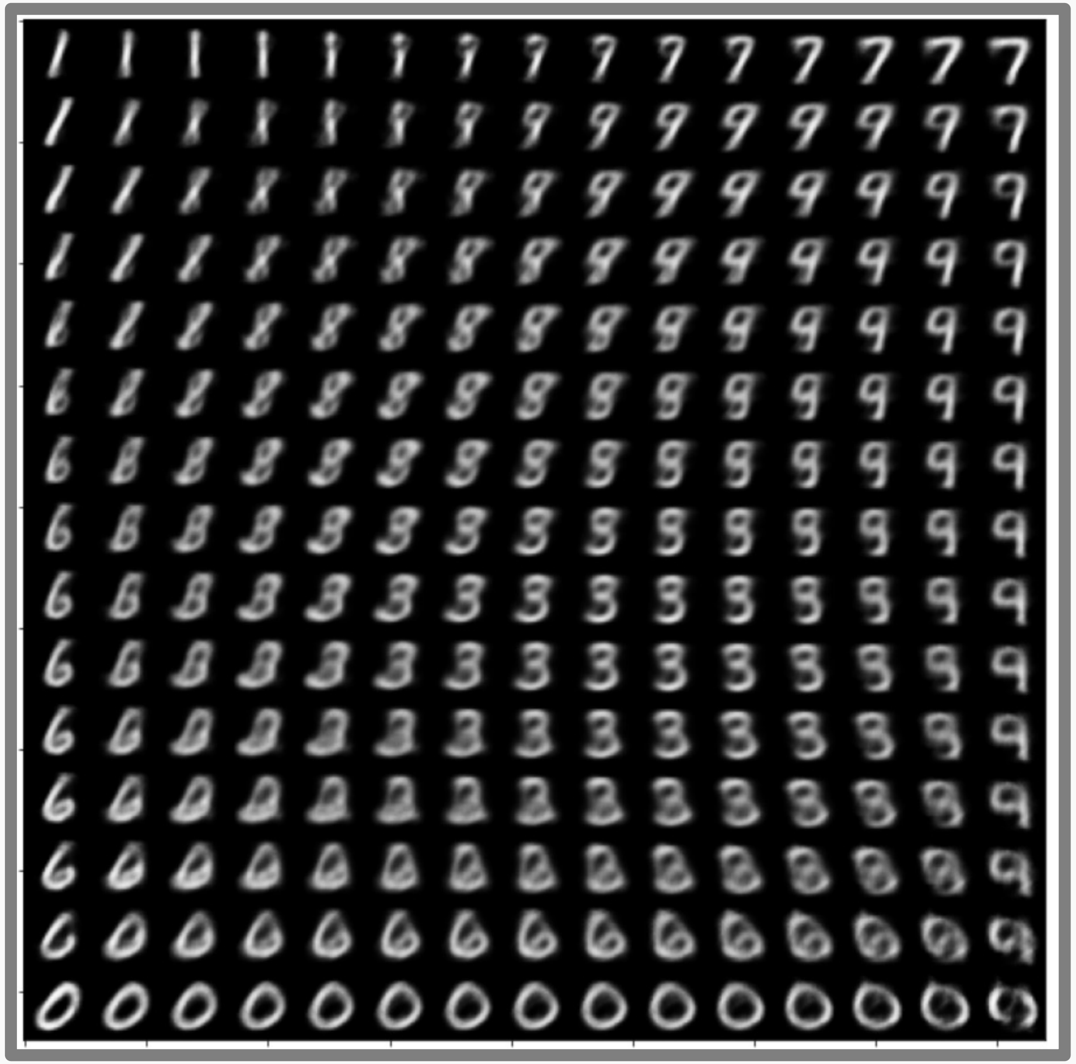
Input Image:

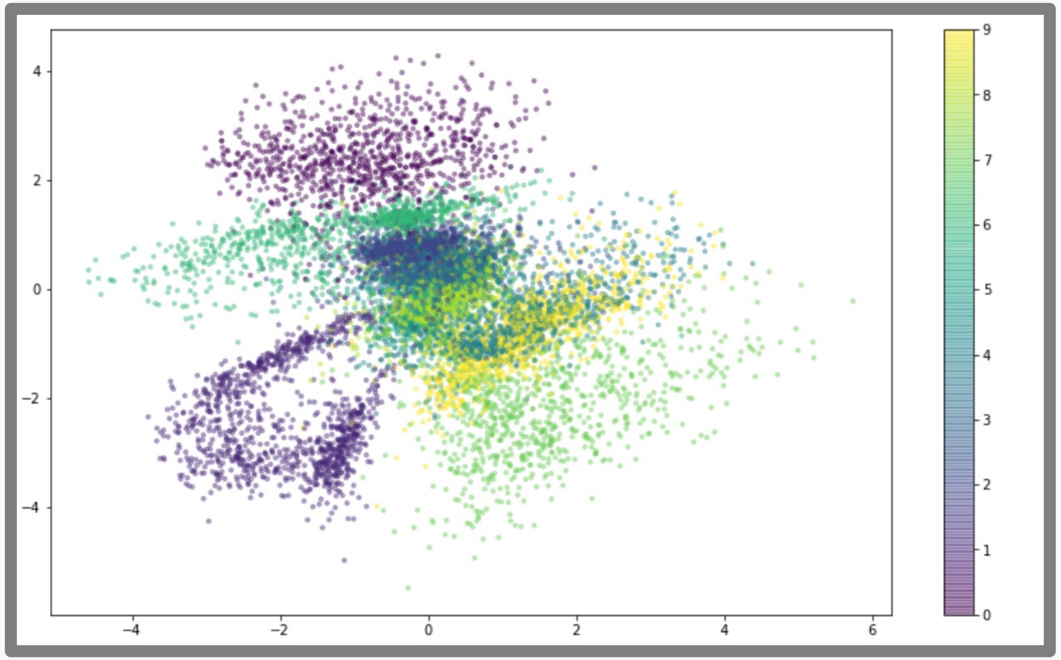
Output Images:

Difference:



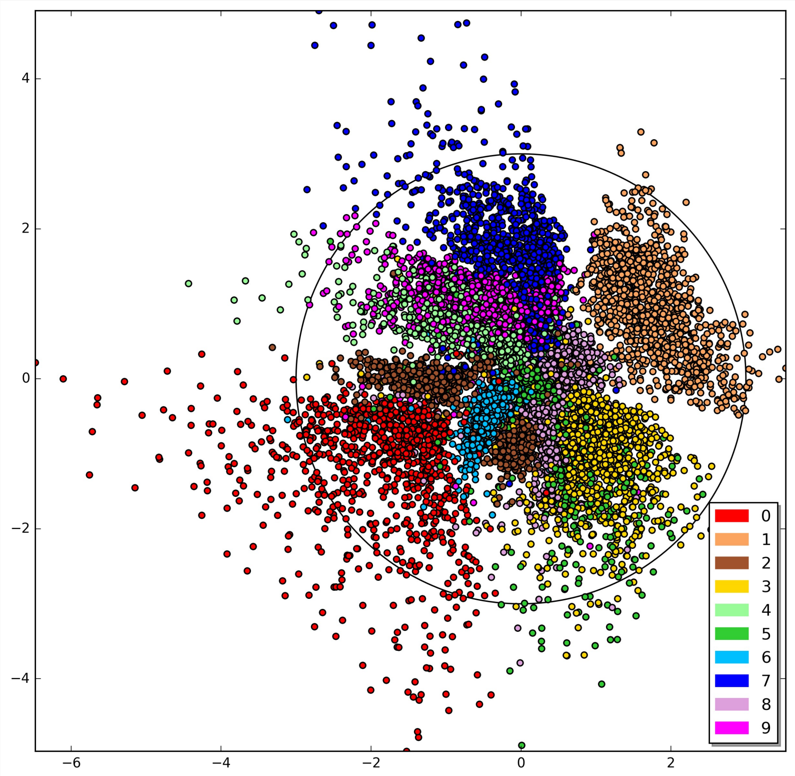
Generative model in action: The Famous plots



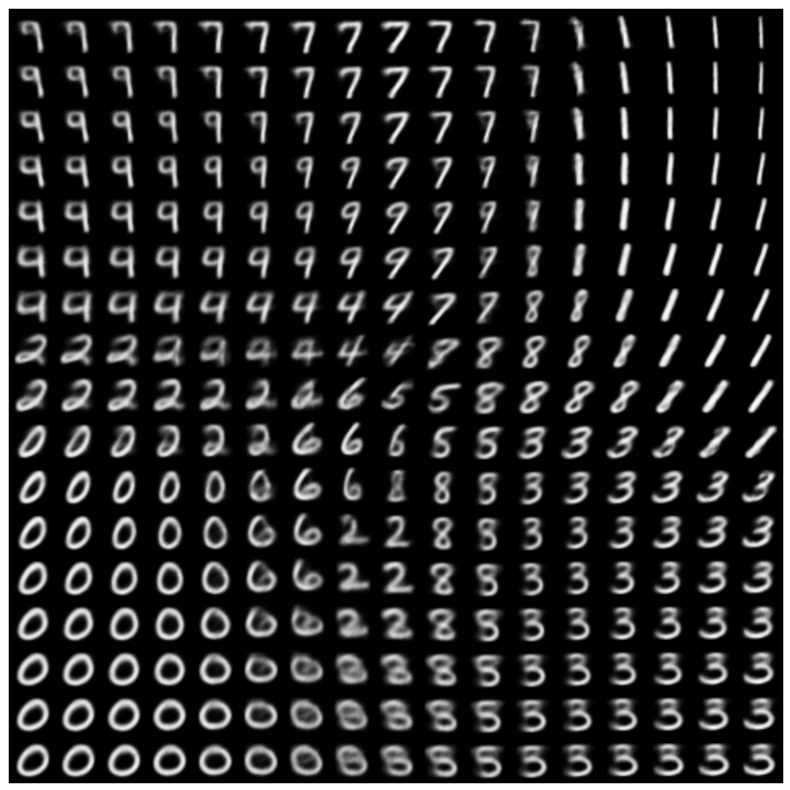


Latent space of VAE

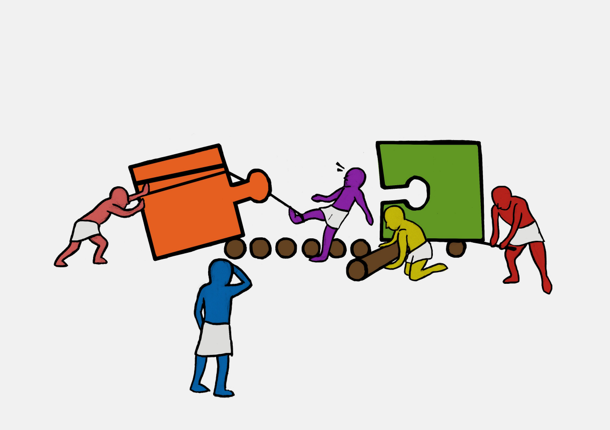
- More separable than AE
- Because of the prior $N(0,1)$ everything is center at $(0,0)$ with spread of approximately one.



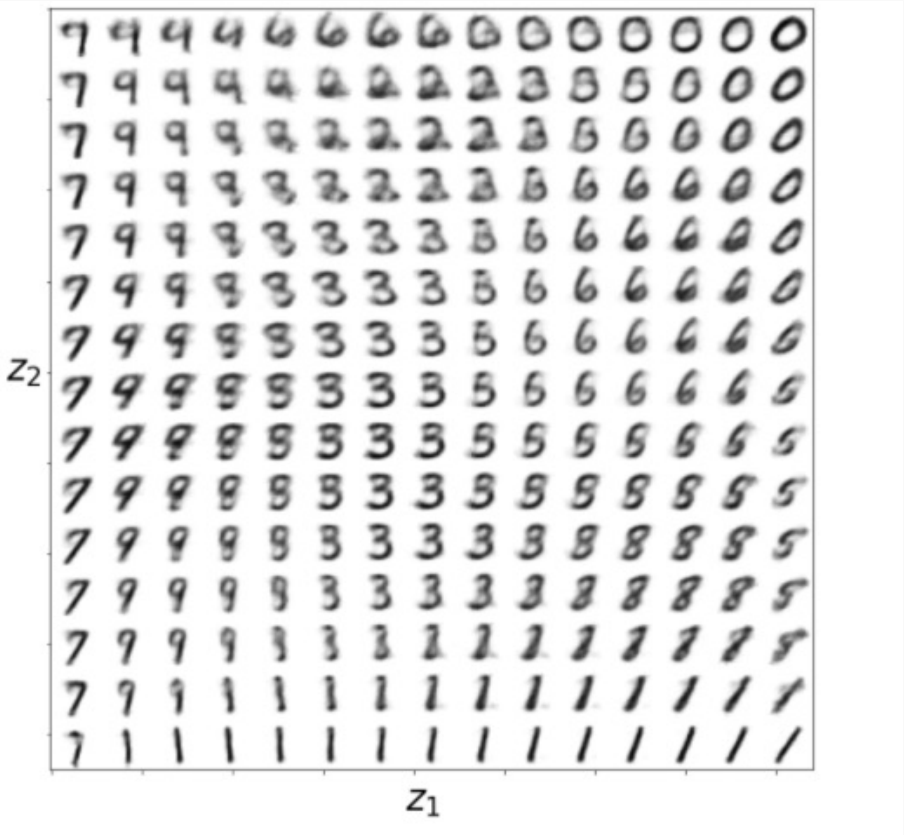
- Blending is more continuous because latent space is continuous



Exercise: Variational Auto-Encoder From scratch



The goal of this exercise is to build a VAE from scratch to reconstruct images of the MNIST dataset. We will use the decoder to generate blended images like on the right.



Note: Here we show you one way of doing VAE. During section we show a slightly different way.

Generative model applications

https://github.com/tkarras/progressive_growing_of_gans

Generative model applications

- Deep Nostalgia from MyHeritage is a generative model based on the ideas mentioned before.
- All generative models are in some way inference models.
- We will study other types of generative models in the upcoming lecture.

https://www.myheritage.com/deep-nostalgia

Bonus Material

What else could work?

Dense Local Reparameterization

tfp.layers.DenseLocalReparameterization

See Stable | See Nightly

View source on GitHub

Densely-connected layer class with local reparameterization estimator.

View aliases

```
tfp.layers.DenseLocalReparameterization(
    units, activation=None, activity_regularizer=None, trainable=True,
    kernel_posterior_fn=tfp_layers_util.default_mean_field_normal_fn(),
    kernel_posterior_tensor_fn=(lambda d: d.sample()),
    kernel_prior_fn=tfp.layers.default_multivariate_normal_fn,
    kernel_divergence_fn=(lambda q, p, ignore: kl_lib.kl_divergence(q, p)), bias_pos
    terior_fn=tfp_layers_util.default_mean_field_normal_fn(is_singular=True),
    bias_posterior_tensor_fn=(lambda d: d.sample()), bias_prior_fn=None,
    bias_divergence_fn=(lambda q, p, ignore: kl_lib.kl_divergence(q, p)), **kwargs
)
```

- **DP Kingma**

Local reparametrization trick

arxiv.org › stat

Auto-Encoding Variational Bayes

by DP Kingma · 2013 · Cited by 13215 — From: Diederik P **Kingma** M.Sc. [view email
20 Dec 2013 20:58:10 UTC (3,884 KB) [v2] Mon, 23 Dec 2013 13:19:52 UTC (7,549 K

Cite as: arXiv:1312.6114

Cited 13,215 times

Everybody talks about my first paper, but the magic is in the second paper!

Variational Dropout and the Local Reparameterization Trick

by DP Kingma · 2015 · Cited by 711 — We investigate a **local** reparameterizaton technique for greatly reducing the variance of stochastic gradients for **variational** Bayesian inference (SGVB) of a posterior over model parameters, while retaining parallelizability.

Cited only 711 times

- DP Kingma

Local reparametrization trick

- Reformulate weight perturbations as activation perturbations and sample (Applicable only on fully connected neural networks with no weight sharing)

$$B = XW$$

$$q_\theta\left(W_{i,j}\right)=\mathcal{N}\left(\mu_{i,j}, \sigma_{i,j}^2\right) \quad \forall W_{i,j} \in W \Longrightarrow q_\theta\left(b_{m,j} \mid X\right)=\mathcal{N}\left(\gamma_{m,j}, \delta_{m,j}\right)$$

$$\gamma_{m,j}=\sum_{i=1} x_{m,i} \mu_{i,j}, \quad \text{and} \quad \delta_{m,j}=\sum_{i=1} x_{m,i}^2 \sigma_{i,j}^2$$

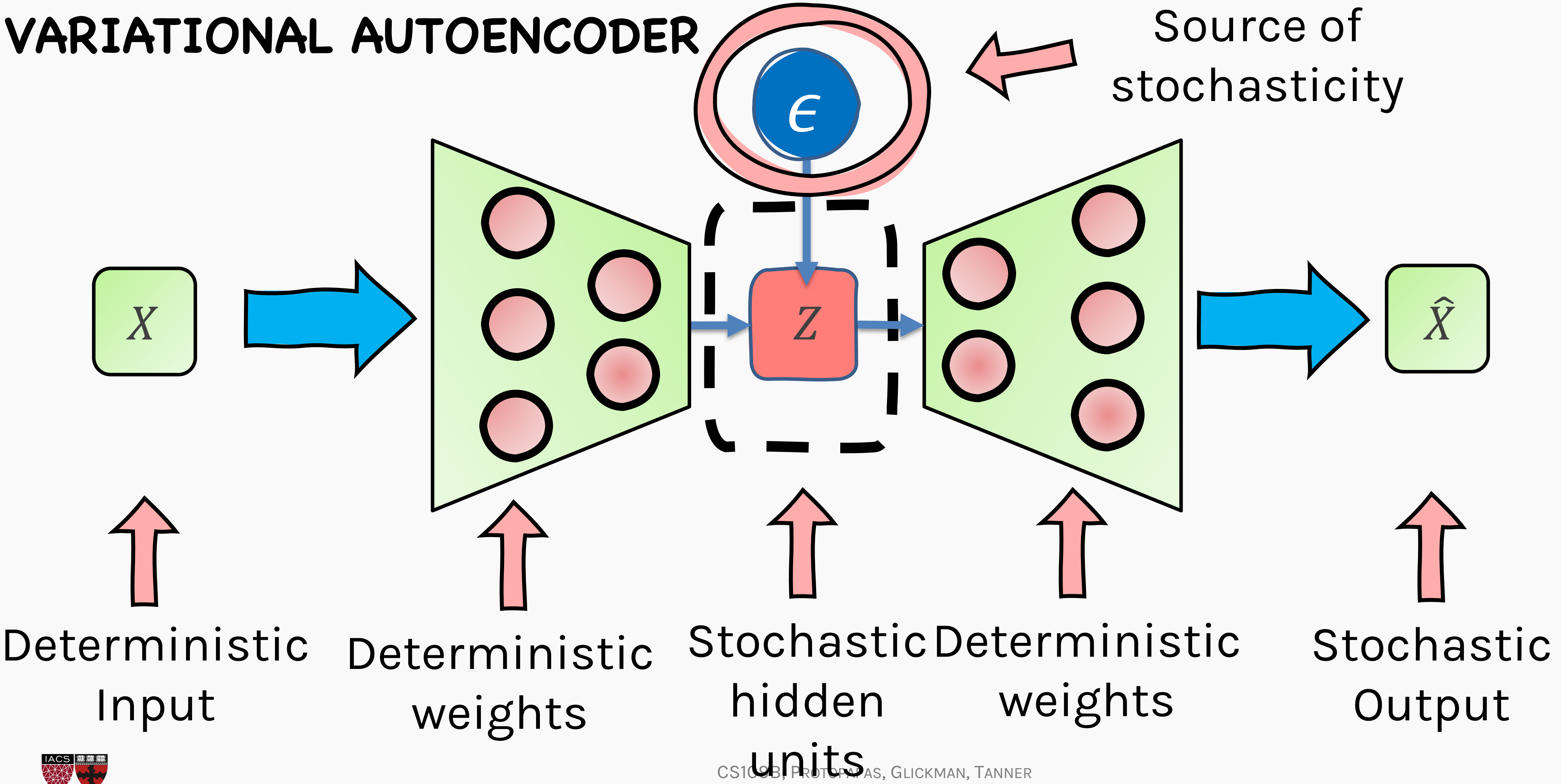
- Inspired from the above idea, Variational dropout works on other architectures

Bayesian Auto-Encoder?

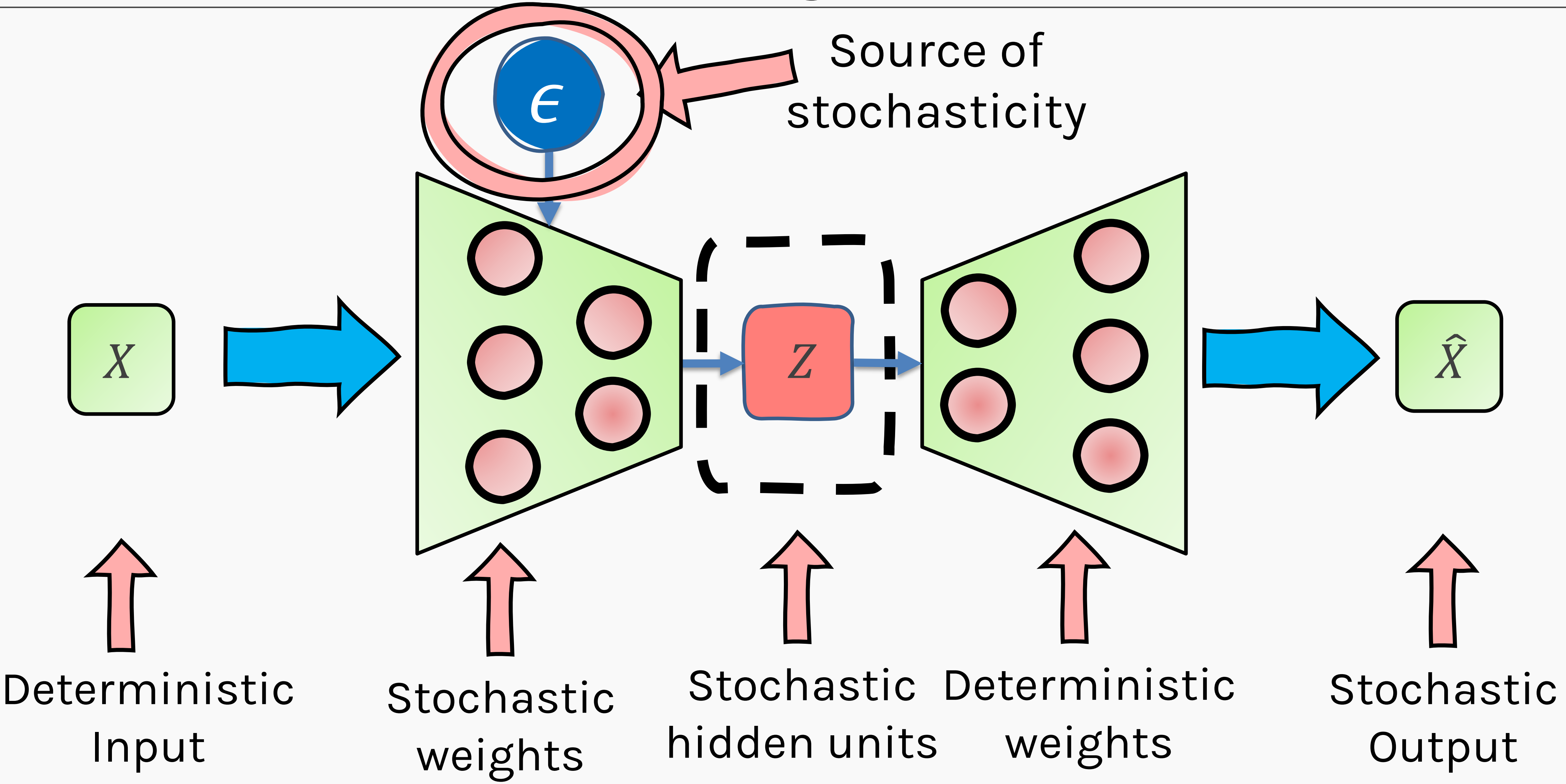
Distribution over the *weights* instead of hidden state

- How about we use *Flipout* layers instead
- Do we still get a similar output distribution?
- How does it compare to stochastic hidden units?

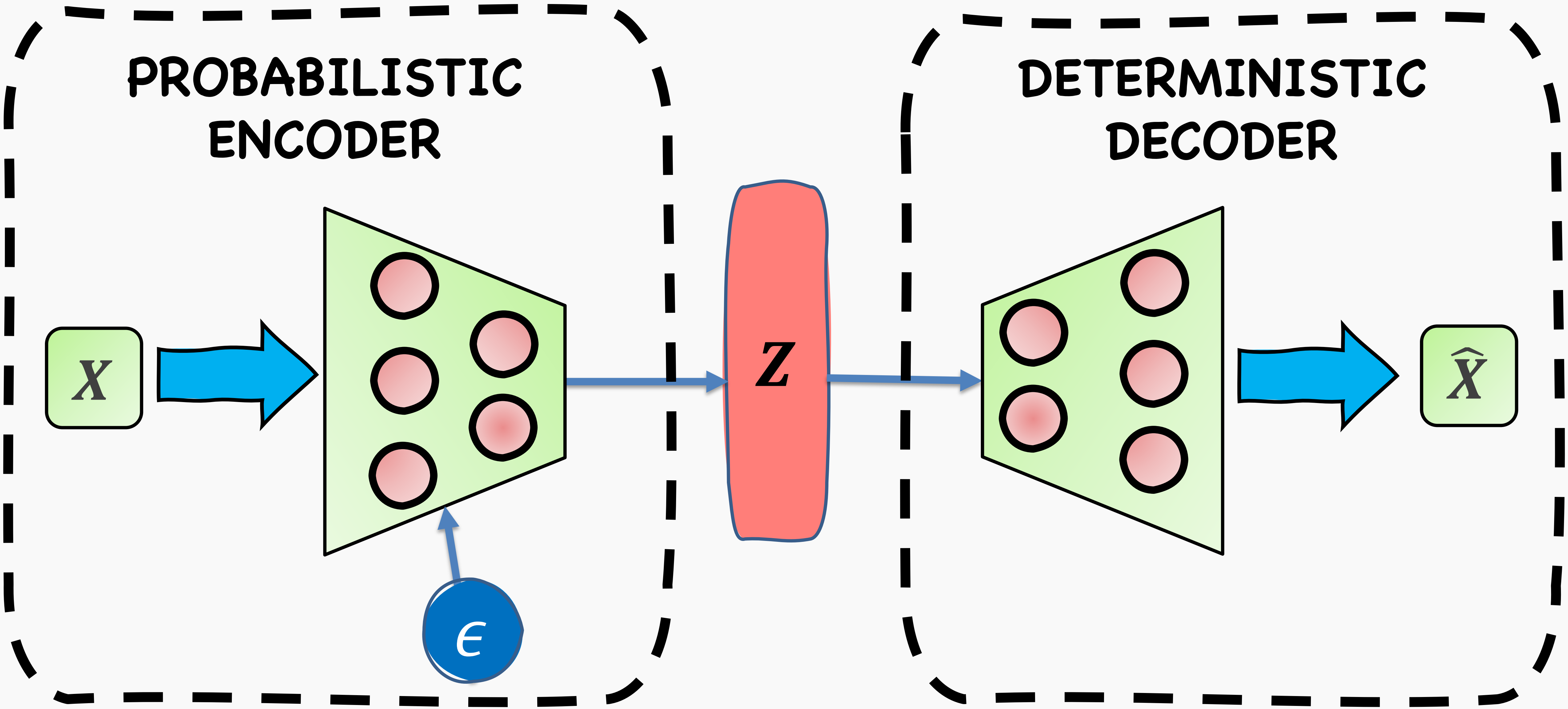
Quick Review



Variational Autoencoder - Weight distributions

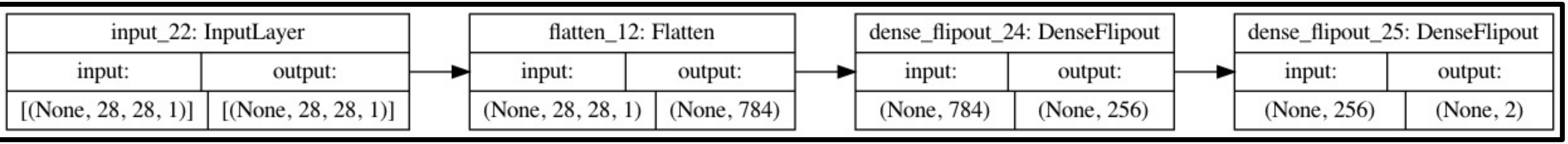


Variational AutoEncoder - Weight distributions

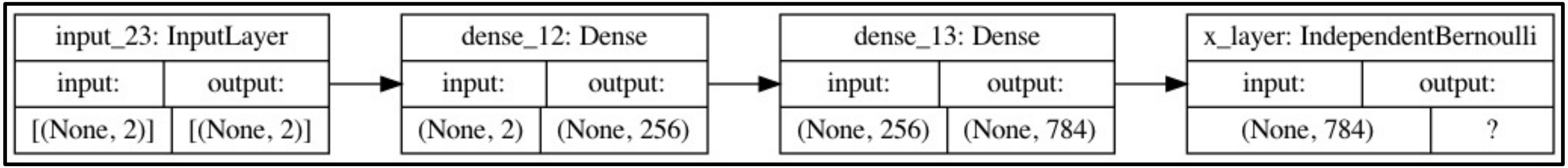


VAE with weight distributions

Probabilistic Encoder



Deterministic Decoder

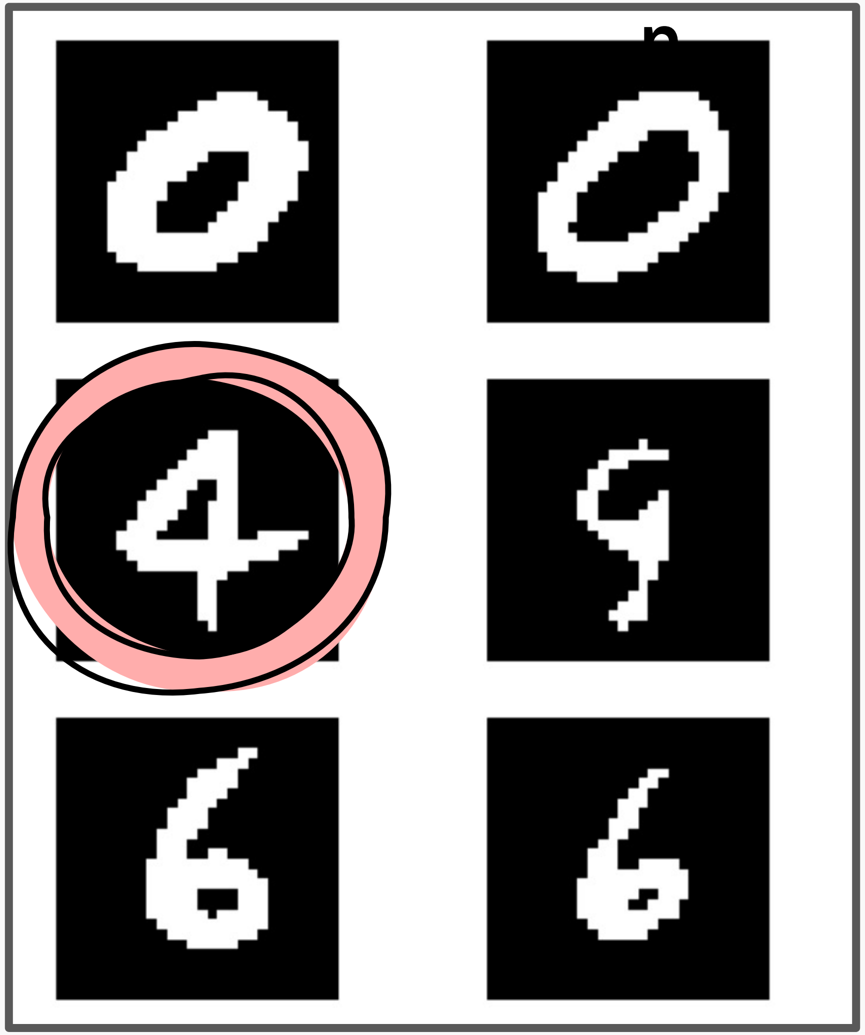


Results?

This works as well!

“4,9” confusion common in MNIST

Test Input Reconstructio n

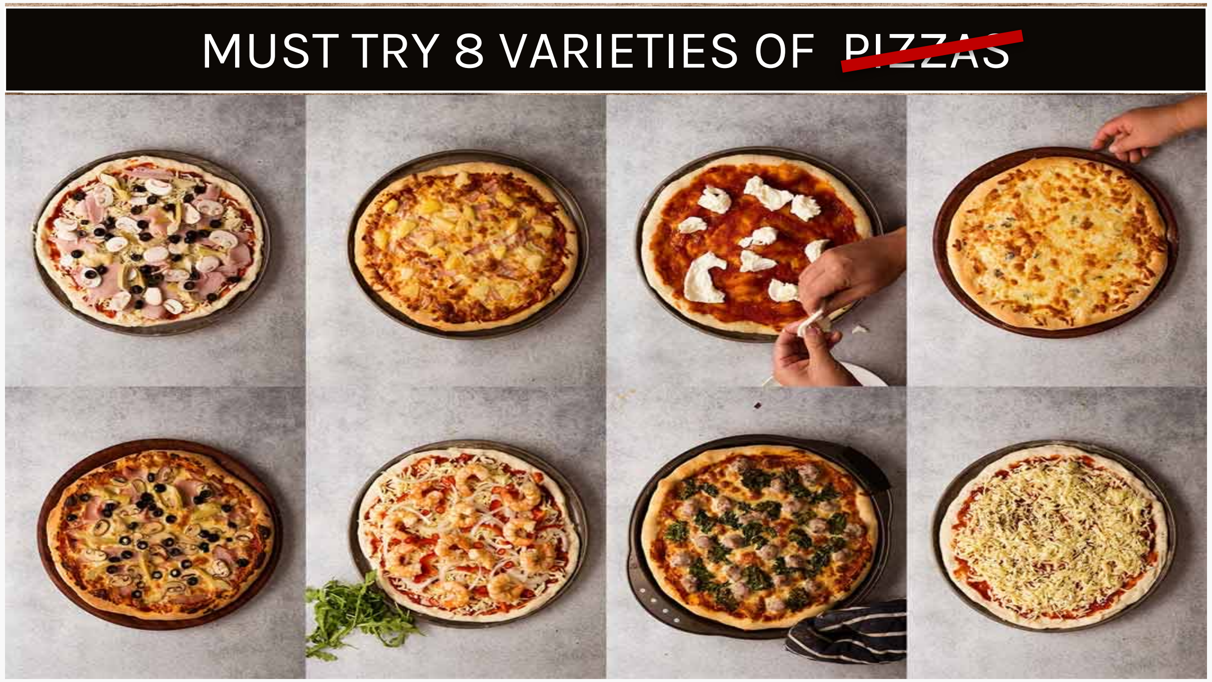


Other methods for uncertainty quantification

Here are the other popular types of inference variants other than the vanilla version:

- **BBB – Bayes by Backprop**
- **PBP - Probabilistic Backprop**
- **MVG – Matrix Variate Gaussian**
- **BBH – Bayes by Hypernet**
- **BB- α – Black-box α divergence**
- **SGLD – Stochastic Gradient LD**
- **Dropout**
- **Ensemble**

BNNs



Please refer to the paper ***Quality Uncertainty Quantification*** for a thorough analysis of all the variants

Uncertainty Quantification – Weiwei Pan Research

Calibration Metrics unreliable

Structure not necessarily helpful

Ensemble methods unpredictabl e