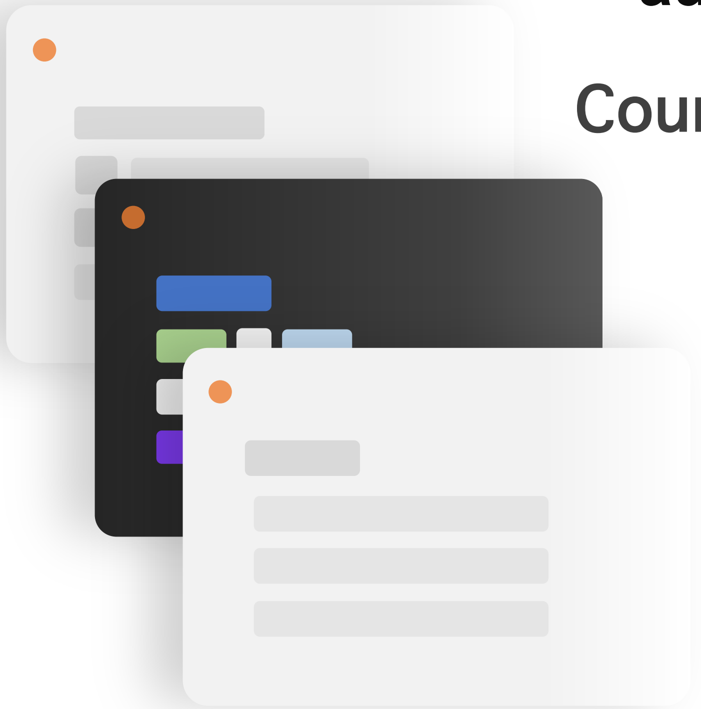
ata *(ah-tuh)*

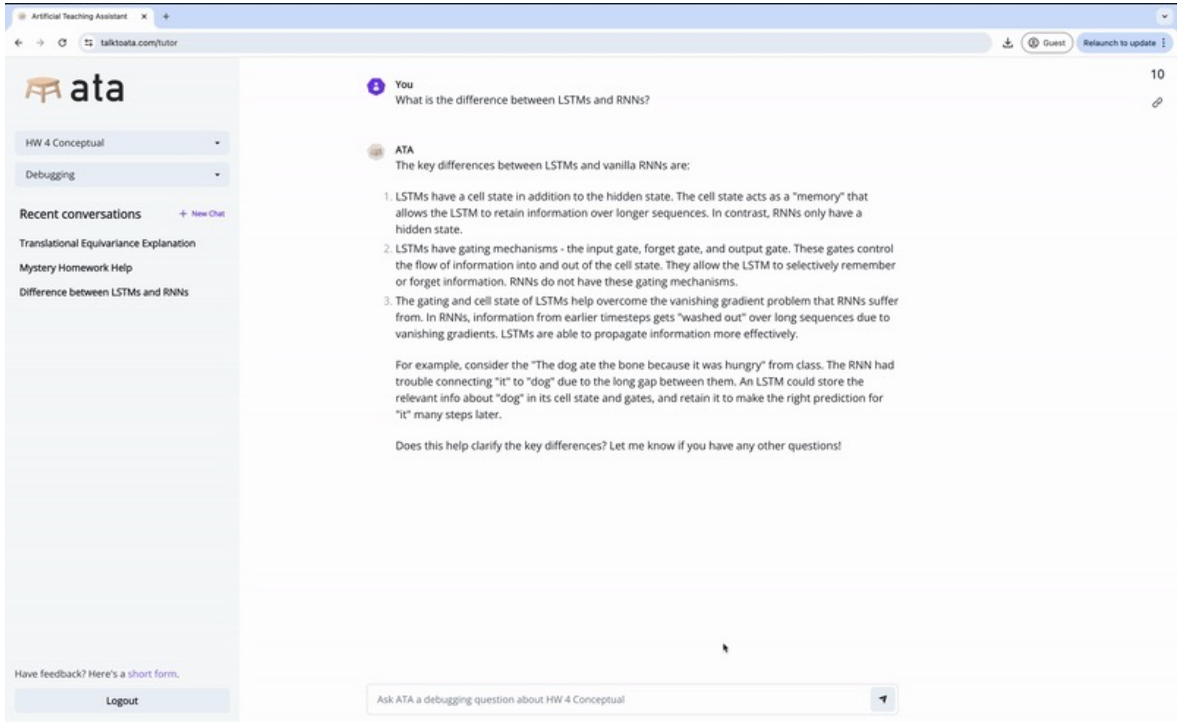
an LLM-powered CS1470 assistant, augmented with:



Course Information

Stencil Code

Assignment Specifications



Available for use starting today (HW5) at

talktoata.com

Terms and conditions apply

Questions or feedback? Reach out to us at team@talktoata.com

CSCI 1470/2470
Spring 2024

Ritambhara Singh

April 01, 2024
Monday

Autoencoders and Variational Autoencoders

Deep Learning

Make sure to submit mid-semester feedback!

ChatGPT prompt “minimalist landscape painting of a deep underwater scene with a blue tang fish in the bottom right corner”

Review: Supervised v/s Unsupervised Learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification, regression, object detection, semantic segmentation, image captioning, *etc.*

Unsupervised Learning

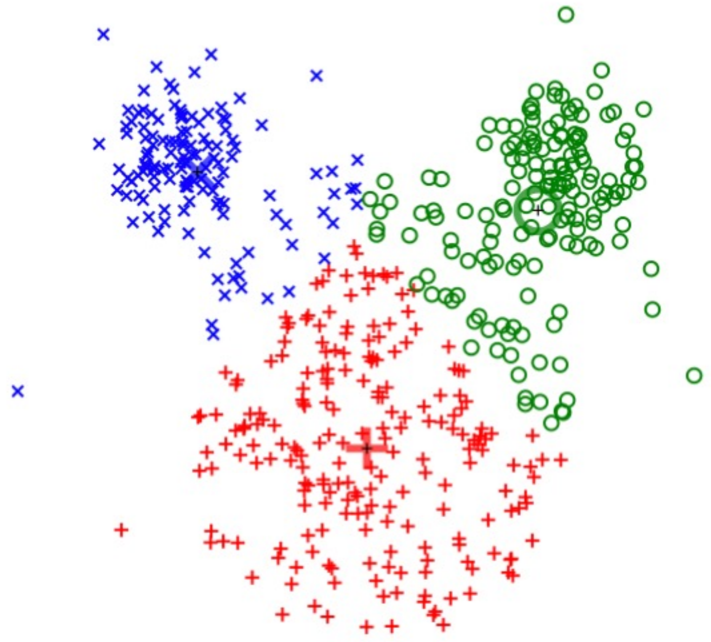
Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, *etc.*

https://www.cs.ubc.ca/~lsigal/532L/Lecture11.pdf

Review: Unsupervised Learning



k-means clustering

This image is CC0 public domain

Unsupervised Learning

Data: x

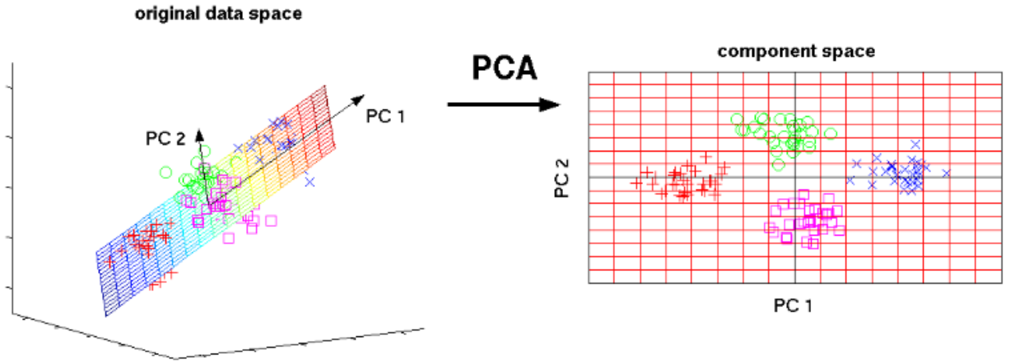
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: $\boxed{\text{Clustering,}}$ dimensionality reduction, feature learning, density estimation, *etc.*

https://www.cs.ubc.ca/~lsigal/532L/Lecture11.pdf

Review: Unsupervised Learning



dimensionality reduction

Unsupervised Learning

Data: x

Just data, no labels!

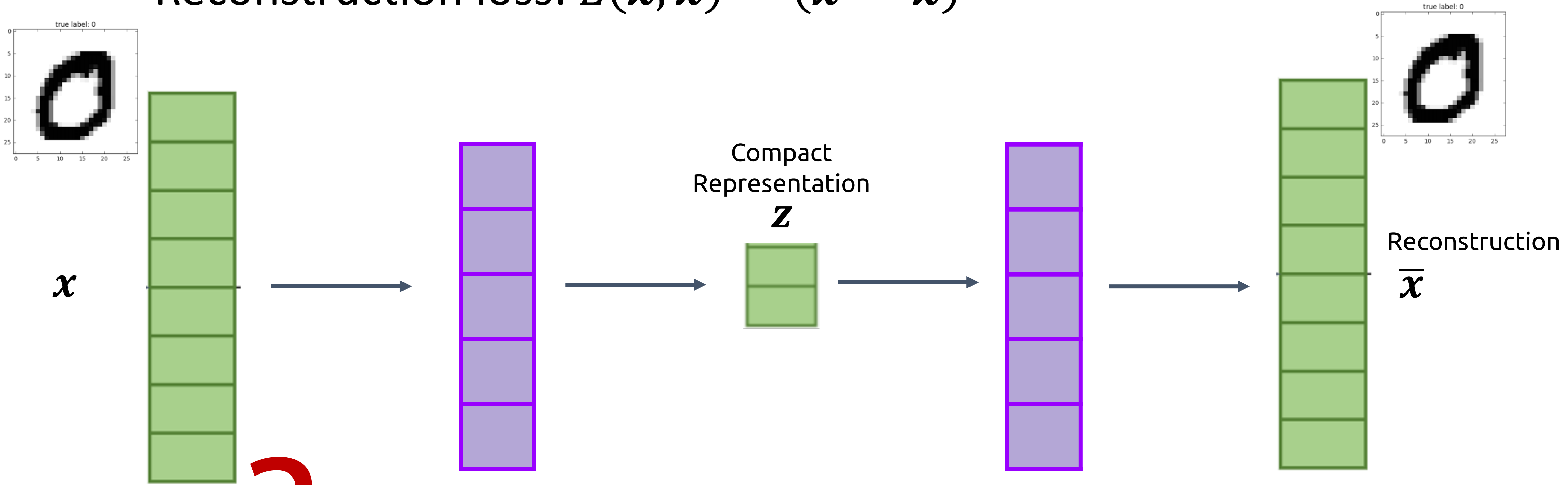
Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, $\boxed{\text{dimensionality reduction,}}$ feature learning, density estimation, *etc.*

https://www.cs.ubc.ca/~lsigal/532L/Lecture11.pdf

Review: Autoencoder

- Reconstruction loss: $L(\boldsymbol{x}, \overline{\boldsymbol{x}}) = (\boldsymbol{x} - \overline{\boldsymbol{x}})^2$



Today’s goal – learn about variational autoencoders (VAEs)

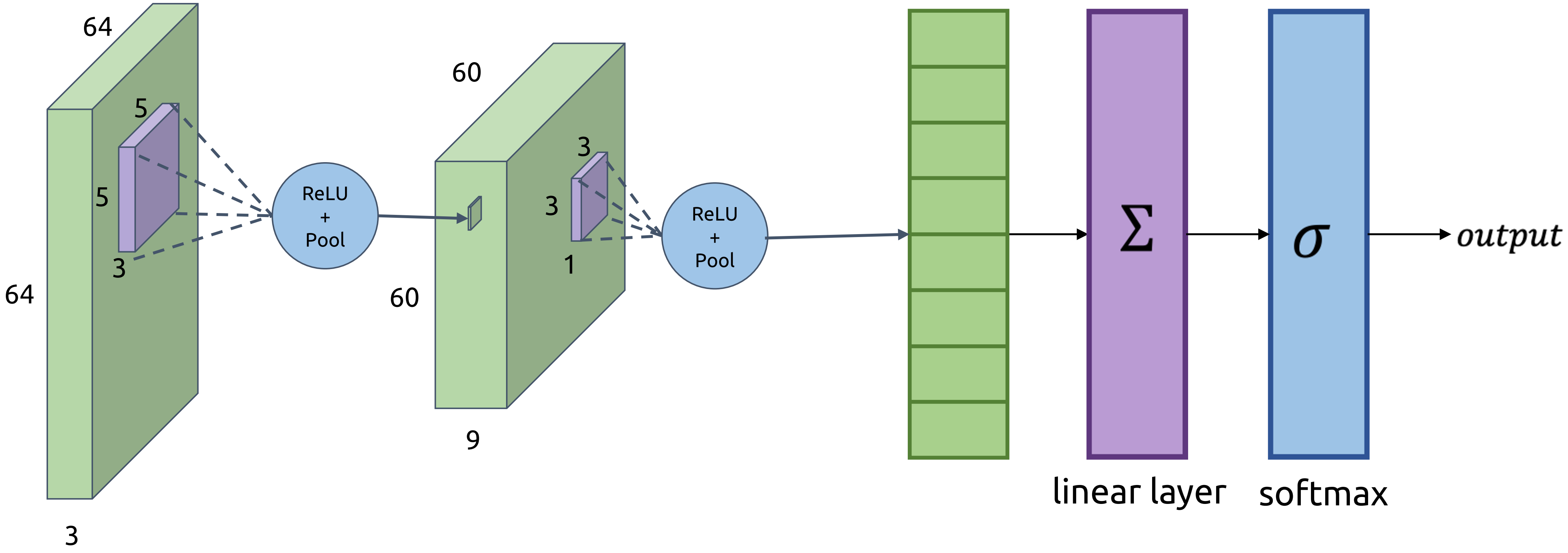
(1) Convolutional AEs

(2) Generative models

(3) Variational Autoencoders (VAEs)

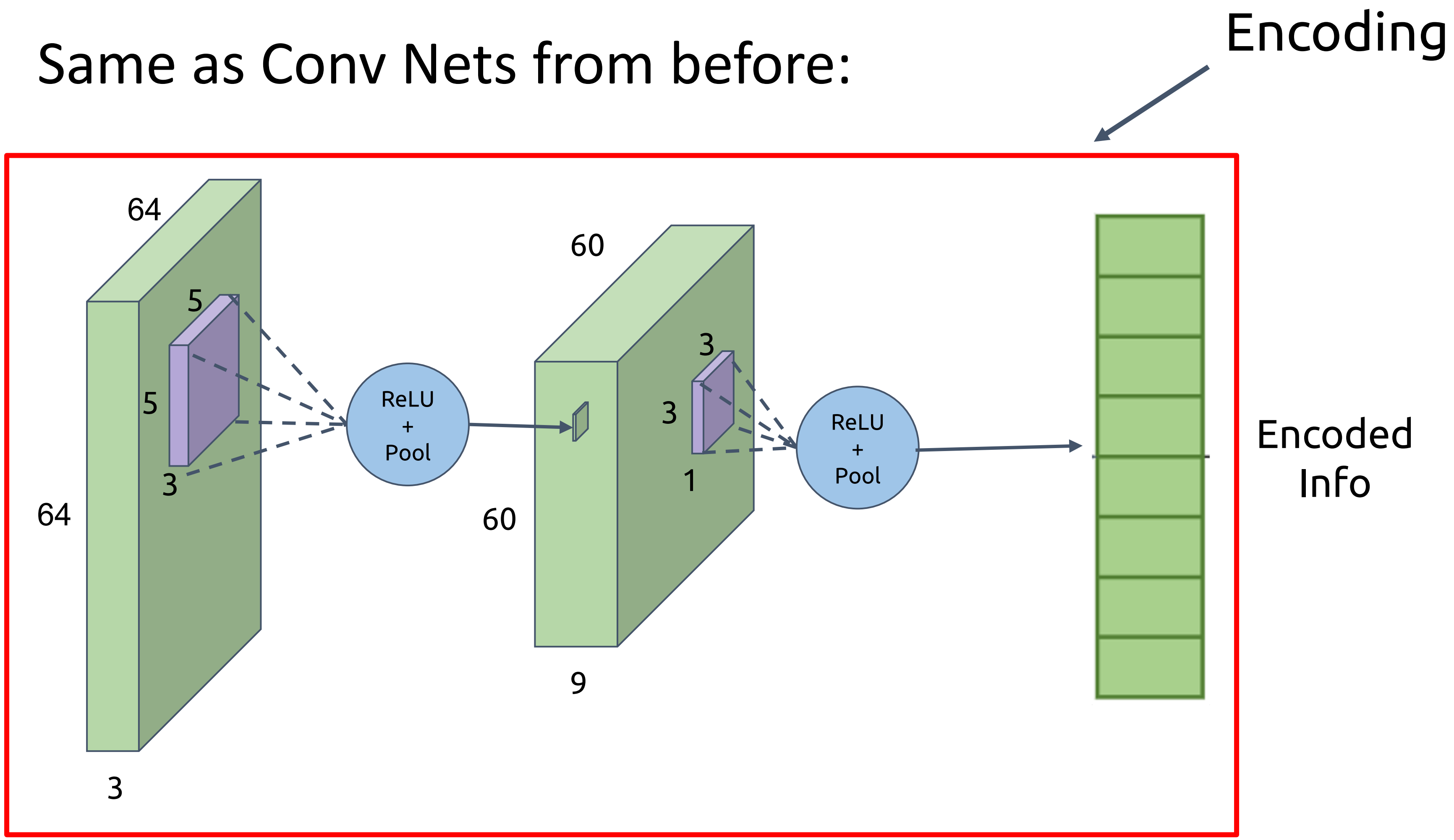
Convolutional Autoencoders

- CNNs are great for image processing in Neural Networks
- How can we build a ***convolutional*** autoencoder?



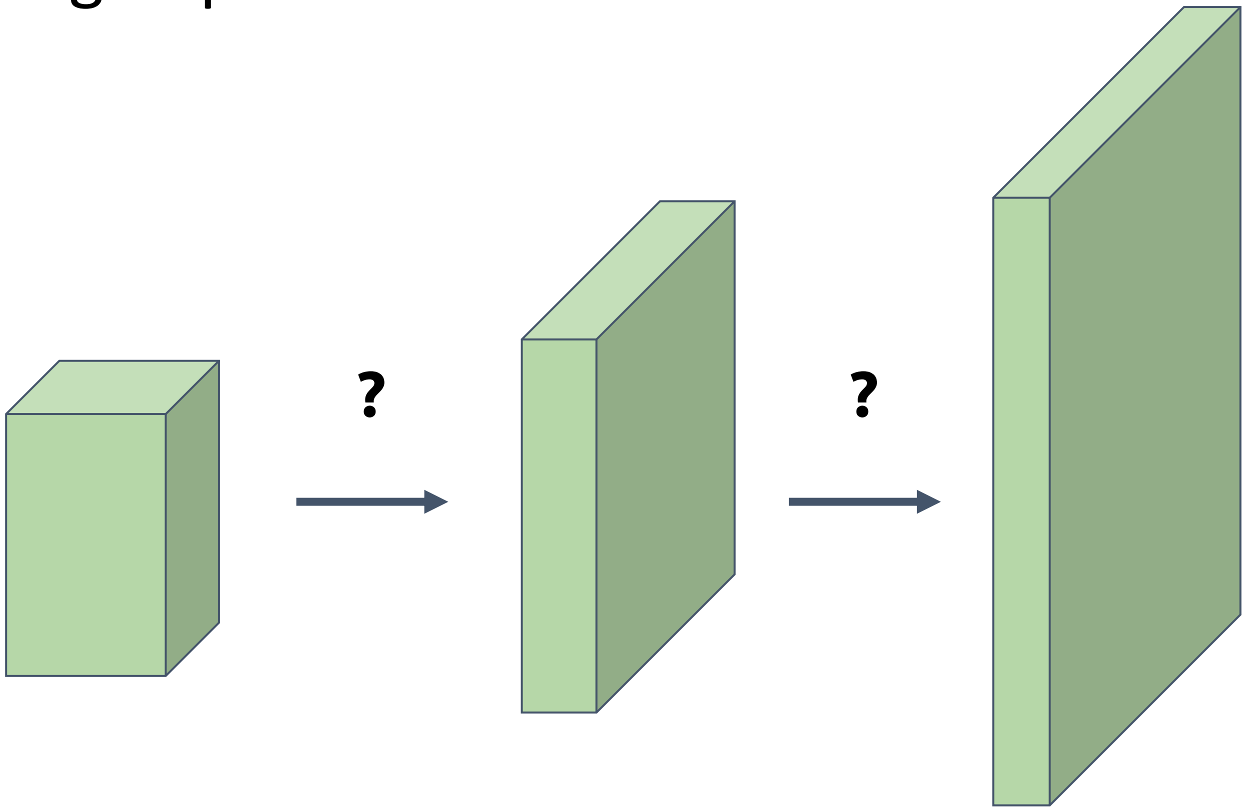
Convolutional Autoencoders: Encoding

Same as Conv Nets from before:



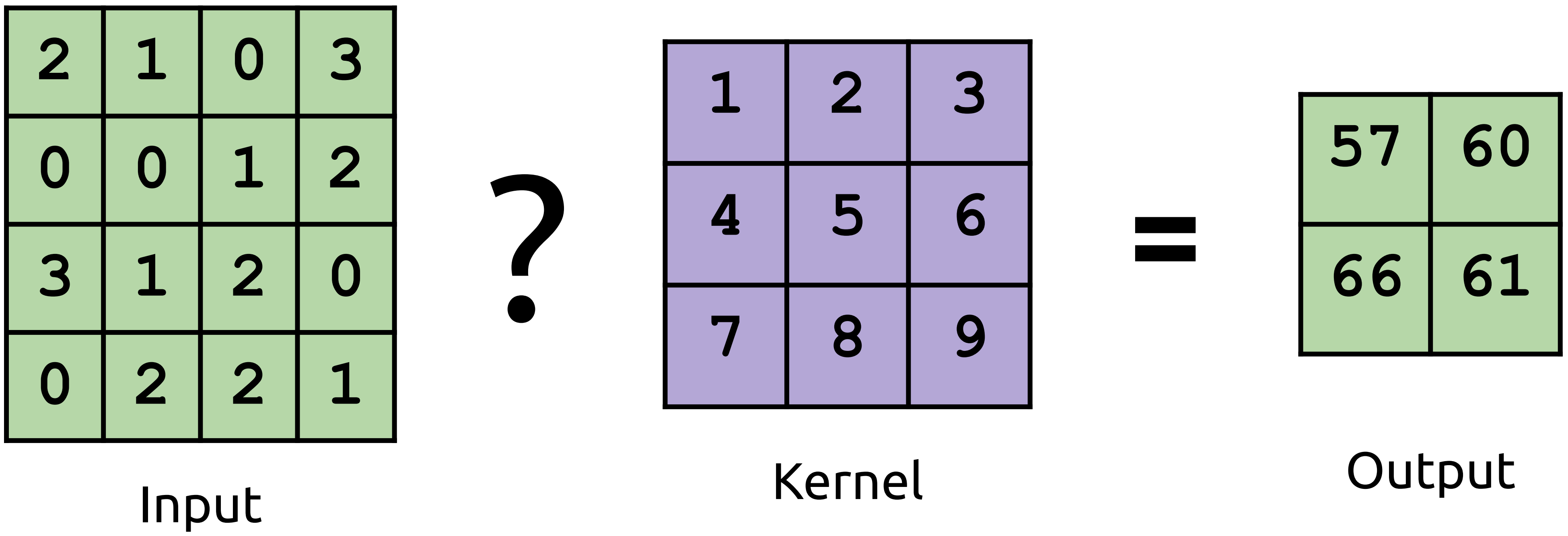
Autoencoders: Decoding

- Convolution as we know it only keeps resolution same or decreases it
- How do we go up in resolution?



Autoencoders: Transpose Convolution

- Convolution can be viewed as a matrix multiplication
- How do we represent it this way?



Autoencoders: Transpose Convolution

Step 1: Flatten the image into a column vector

2	1	0	3
0	0	1	2
3	1	2	0
0	2	2	1

2
1
0
3
0
0
1
2
3
1
2
0
0
2
2
1

Autoencoders: Transpose Convolution

- Step 2: Unroll the kernel/filter

1	2	3
4	5	6
7	8	9

1	2	3	0	4	5	6	0	7	8	9	0	0	0	0	0
0	1	2	3	0	4	5	6	0	7	8	9	0	0	0	0
0	0	0	0	1	2	3	0	4	5	6	0	7	8	9	0
0	0	0	0	0	1	2	3	0	4	5	6	0	7	8	9

Autoencoders: Transpose Convolution

Step 3: Matrix multiply unrolled kernel with flattened image

$$
\begin{bmatrix}
1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9
\end{bmatrix}
\times
\begin{bmatrix} 2 \\ 1 \\ 0 \\ 3 \\ 0 \\ 0 \\ 1 \\ 2 \\ 3 \\ 1 \\ 2 \\ 0 \\ 0 \\ 2 \\ 2 \\ 1 \end{bmatrix}
=
\begin{bmatrix} 57 \\ 50 \\ 66 \\ 61 \end{bmatrix}
$$

Autoencoders: Transpose Convolution

Each row of the convolution matrix corresponds to a dot product between filter and image patch:

$$\begin{bmatrix} 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} 2 \\ 1 \\ 0 \\ 3 \\ 0 \\ 0 \\ 1 \\ 2 \\ 3 \\ 1 \\ 2 \\ 0 \\ 0 \\ 2 \\ 2 \\ 1 \end{bmatrix} \Leftrightarrow \begin{bmatrix} 2 & 1 & 0 & 3 \\ 0 & 0 & 1 & 2 \\ 3 & 1 & 2 & 0 \\ 0 & 2 & 2 & 1 \end{bmatrix}$$

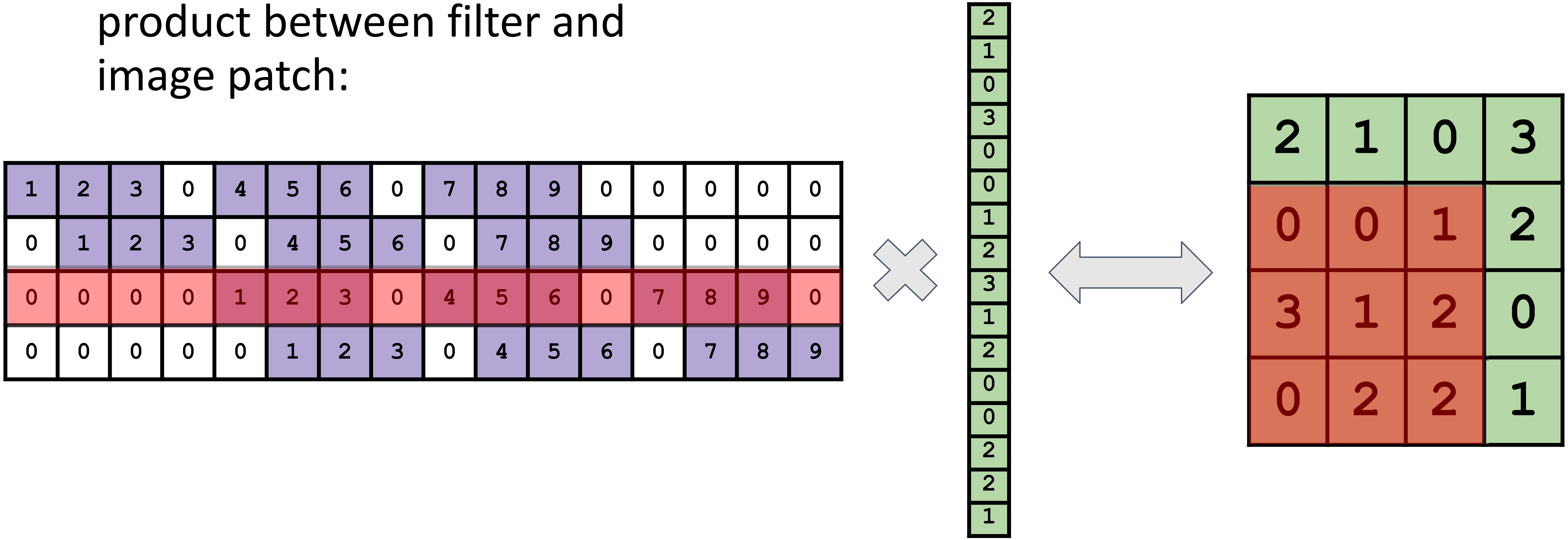
Autoencoders: Transpose Convolution

Each row of the convolution matrix corresponds to a dot product between filter and image patch:

$$
\begin{bmatrix}
1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9
\end{bmatrix}
\times
\begin{bmatrix} 2 \\ 1 \\ 0 \\ 3 \\ 0 \\ 0 \\ 1 \\ 2 \\ 3 \\ 1 \\ 2 \\ 0 \\ 0 \\ 2 \\ 2 \\ 1 \end{bmatrix}
\Leftrightarrow
\begin{bmatrix}
2 & 1 & 0 & 3 \\
0 & 0 & 1 & 2 \\
3 & 1 & 2 & 0 \\
0 & 2 & 2 & 1
\end{bmatrix}
$$

Autoencoders: Transpose Convolution

Each row of the convolution matrix corresponds to a dot product between filter and image patch:



Autoencoders: Transpose Convolution

Each row of the convolution matrix corresponds to a dot product between filter and image patch:

$$\begin{bmatrix} 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} 2 \\ 1 \\ 0 \\ 3 \\ 0 \\ 0 \\ 1 \\ 2 \\ 3 \\ 1 \\ 2 \\ 0 \\ 0 \\ 2 \\ 2 \\ 1 \end{bmatrix} \Leftrightarrow \begin{bmatrix} 2 & 1 & 0 & 3 \\ 0 & 0 & 1 & 2 \\ 3 & 1 & 2 & 0 \\ 0 & 2 & 2 & 1 \end{bmatrix}$$

Autoencoders: Transpose Convolution

Step 4: Finally reshape the output back into a grid

57
50
66
61

Final Output Image

57	60
66	61

Autoencoders: Transpose Convolution

$$\begin{bmatrix} 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 & 0 & 4 & 5 & 6 & 0 & 7 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} 2 \\ 1 \\ 0 \\ 3 \\ 0 \\ 0 \\ 1 \\ 2 \\ 3 \\ 1 \\ 2 \\ 0 \\ 0 \\ 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 57 \\ 50 \\ 66 \\ 61 \end{bmatrix}$$

Autoencoders: Transpose Convolution

To upsample an image, we just do the inverse of this operation.

What matrix do we use?

The **transpose** of the big convolution matrix

Input image flattened to column vector

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 1 & 0 \\ 5 & 4 & 2 & 1 \\ 6 & 5 & 3 & 2 \\ 0 & 6 & 0 & 3 \\ 7 & 0 & 4 & 0 \\ 8 & 7 & 5 & 4 \\ 9 & 8 & 6 & 5 \\ 0 & 9 & 0 & 6 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 8 & 7 \\ 0 & 0 & 9 & 8 \\ 0 & 0 & 0 & 9 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 0 \\ 6 \\ 10 \\ 14 \\ 3 \\ 15 \\ 22 \\ 26 \\ 6 \\ 14 \\ 23 \\ 26 \\ 9 \end{bmatrix}$$

Autoencoders: Transpose Convolution

Finally, reshape the output vector into a grid to get the final output image:

1
2
3
0
6
10
14
3
15
22
26
6
14
23
26
9

Final output image

1	2	3	4
6	10	14	3
15	1	22	26
14	23	26	9

Transpose Convolution in Tensorflow

```
tf.nn.conv2d_transpose(input, filters, output_shape, strides, padding='SAME')
```

4D tensor of shape [batch, height, width, in_channels]

4-D Tensor with shape [height, width, output_channels, in_channels]

length 4 1D tensor representing the output shape.

Strides along each dimension (list of integers)

String representing type of padding

Documentation here: https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/nn/conv2d

Transpose Convolution in Tensorflow

```
tf.nn.conv2d_transpose(input, filters, output_shape, strides, padding='SAME')
```

Why do we need to specify output size?

Specifying Output Size

- An image can be the result of the same convolution on images of different resolution
- We need to specify which one we want.

57	60
66	61

1	2	3
4	5	6
7	8	9

Kernel

2	1	0	3
0	0	1	2
3	1	2	0
0	2	2	1

2	1	0	3	0
0	0	1	2	0
3	1	2	0	0
0	2	2	1	0
0	0	0	0	0

Any questions?

Transpose Convolution in Keras

```
tf.keras.layers.Conv2DTranspose(filters, kernel_size, strides, padding='SAME')
```

Number of filters (Integer)

Size of Convolution Window (tuple)

Strides along each dimension (list of integers)

String representing type of padding

Note: Output Shape is inferred

Documentation here: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2DTranspose

Today's goal – learn about variational autoencoders (VAEs)

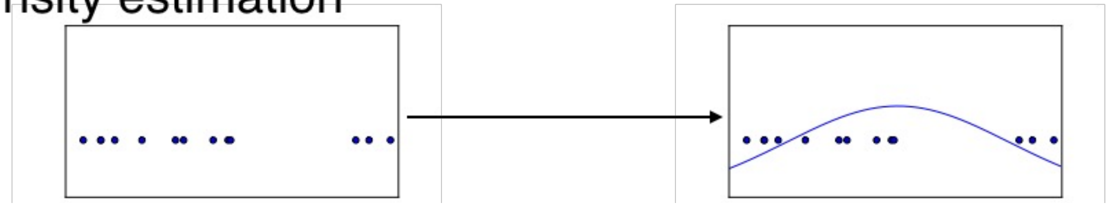
(1) Convolutional AEs

(2) Generative models

(3) Variational Autoencoders (VAEs)

Unsupervised Learning

- Density estimation



Unsupervised Learning

Data: x

Just data, no labels!

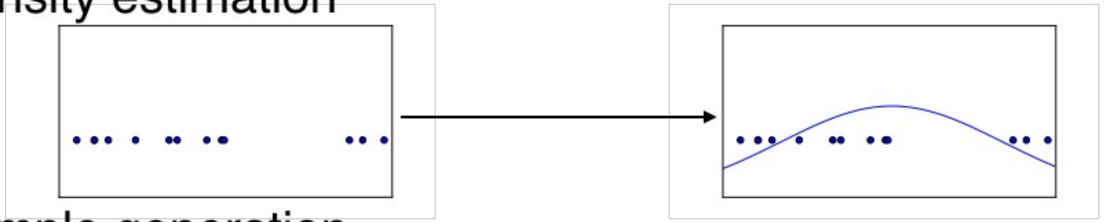
Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, *etc.*

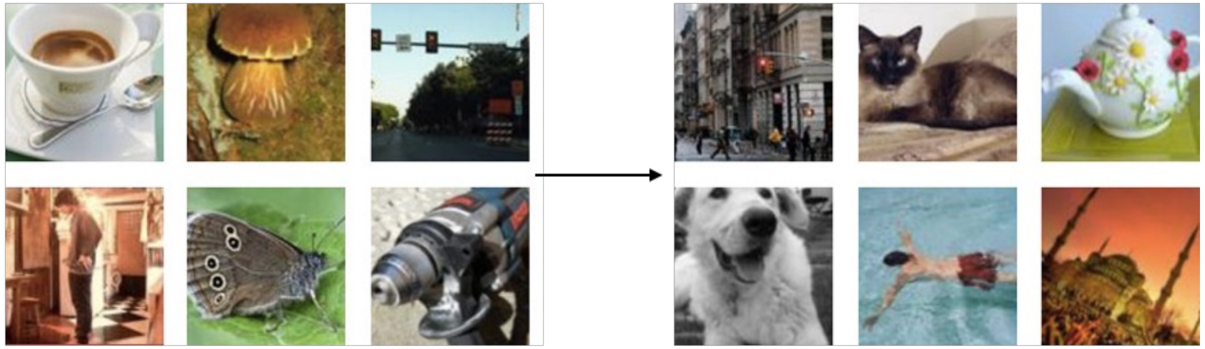
Unsupervised Learning

Generative models

- Density estimation



- Sample generation



Training examples Model samples

Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, *etc.*

Discriminative v/s Generative models

Discriminative Model:
Learn a probability distribution $p(y|x)$

Generative Model:
Learn a probability distribution $p(x)$

Data: x

Label: y

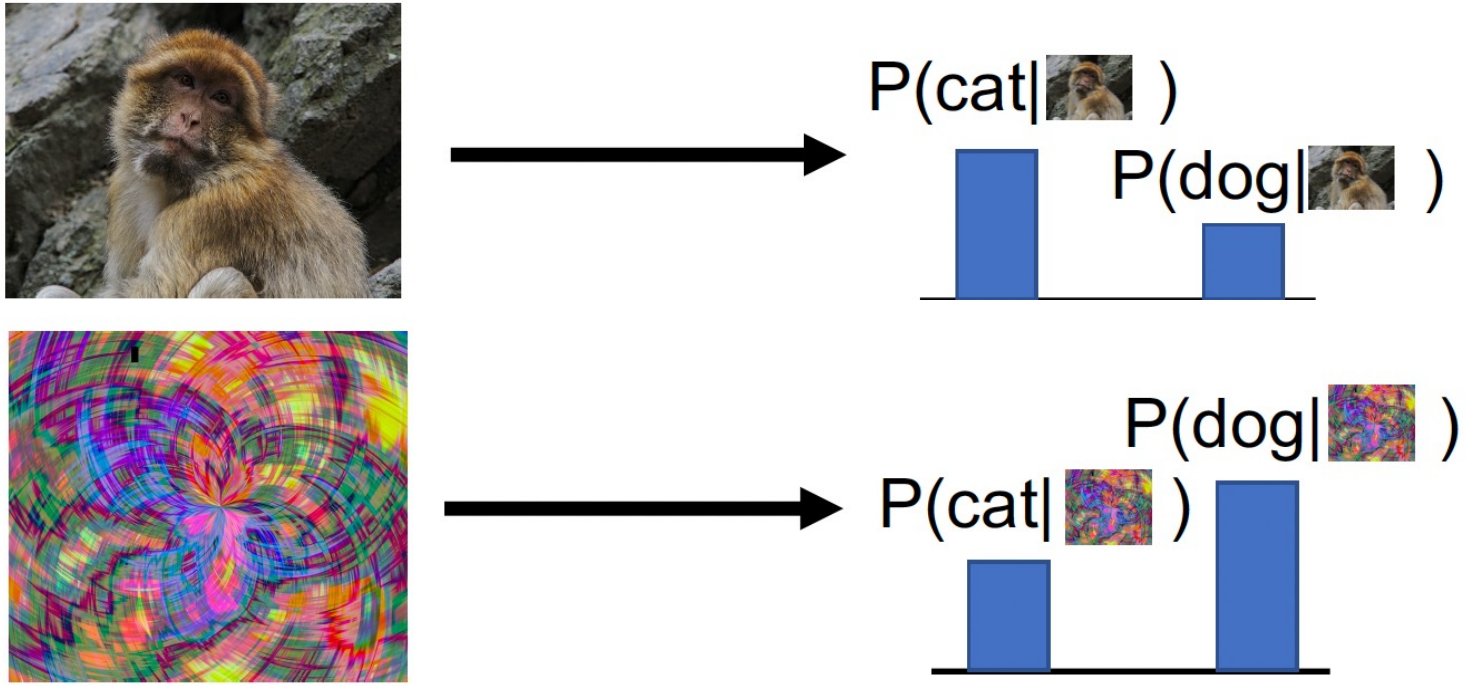
Cat

Credit: UMich EECS498

Discriminative v/s Generative models

Discriminative Model:
Learn a probability distribution p(y|x)

Generative Model:
Learn a probability distribution p(x)



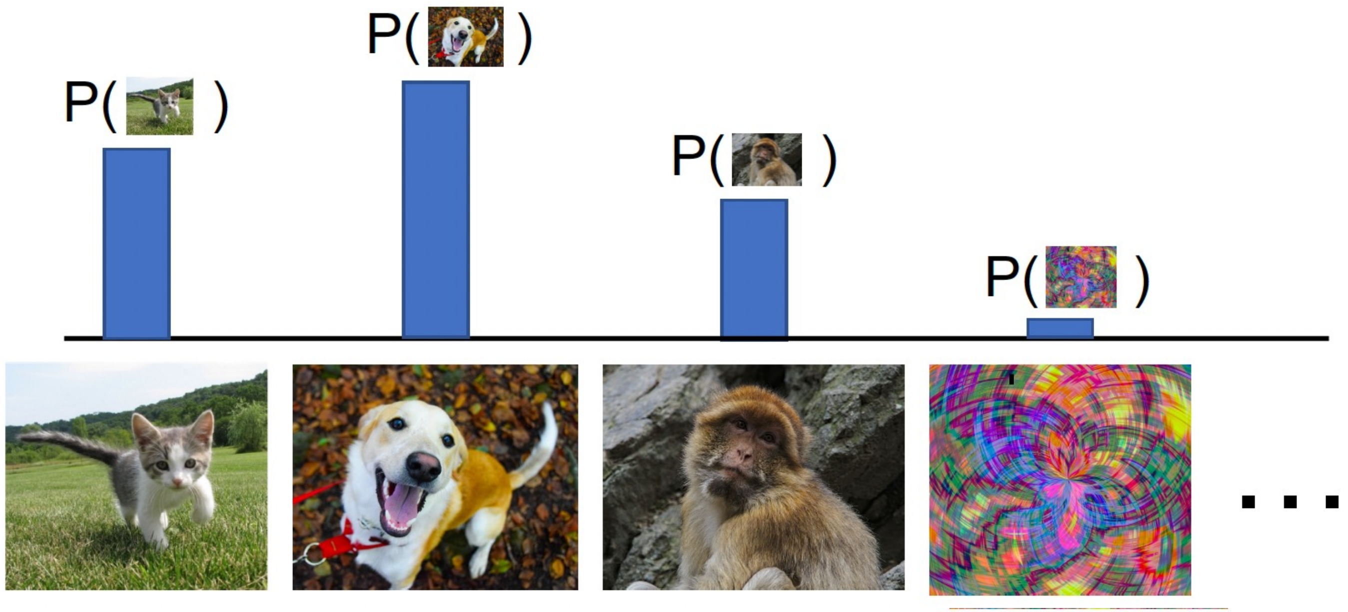
Discriminative model: No way for the model to handle unreasonable inputs; it must give label distributions for all images

Credit: UMich EECS498

Discriminative v/s Generative models

Discriminative Model:
Learn a probability distribution p(y|x)

Generative Model:
Learn a probability distribution p(x)

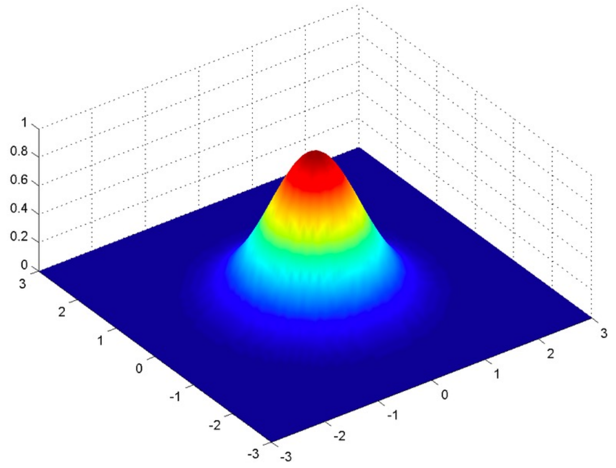


- Generative model: All possible images compete with each other for probability mass
- Intuition: Generation should require deep understanding! Is a dog more likely to sit or stand? How about 3-legged dog vs 3-armed monkey?
- Model can "reject" unreasonable inputs by assigning them small values

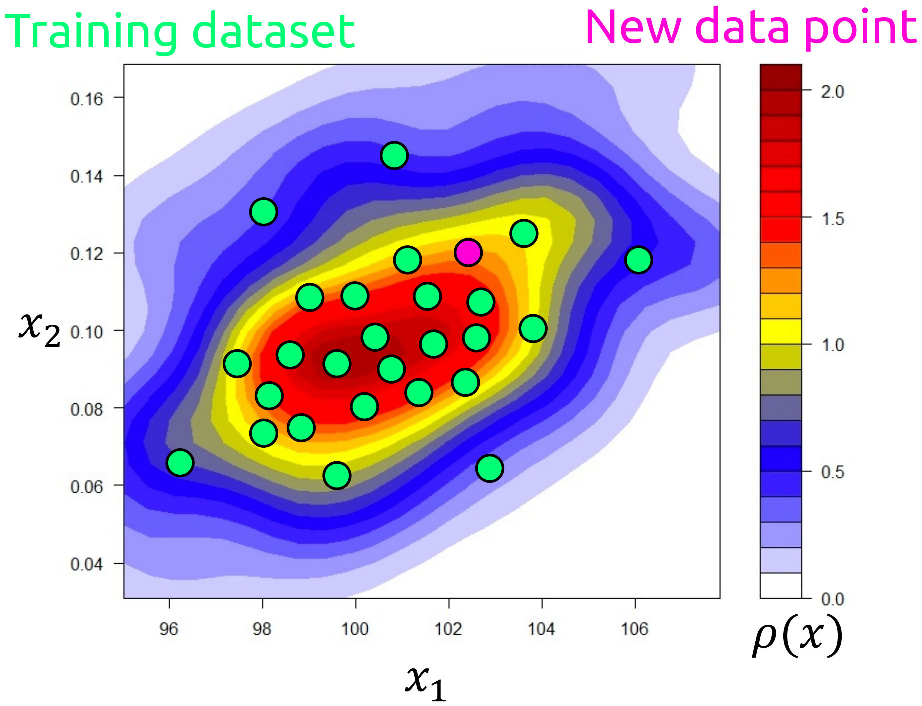
Credit: UMich EECS498

Generative Modeling Is:

1. A procedure for (approximately) ***sampling*** from the distribution from which a dataset was drawn



$x' \sim p(.)$



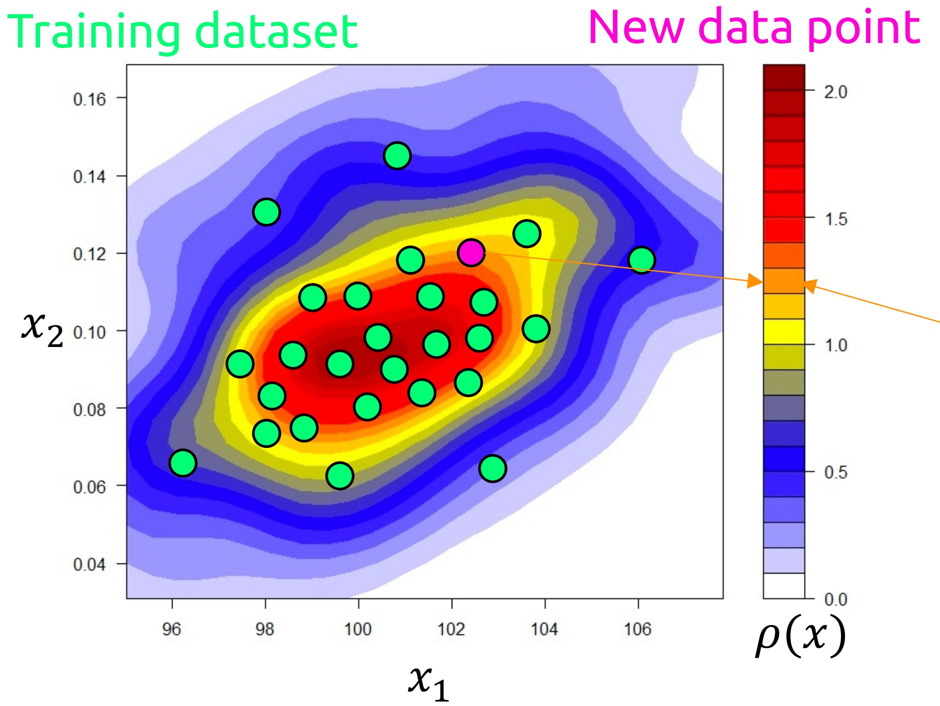
*Probability density is the relationship between observations and their probability.

Generative Modeling:

1. A procedure for (approximately) ***sampling*** from the distribution from which a dataset was drawn

2. A procedure for (approximately) ***evaluating the probability density*** of a datapoint under the distribution from which a dataset was drawn

$x' \sim p(.)$



$\rho(x')$

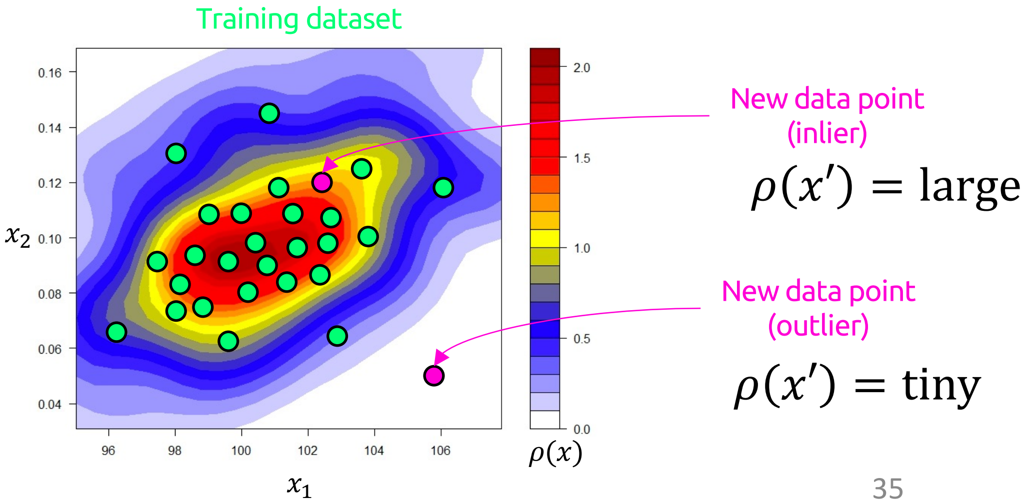
*Probability density is the relationship between observations and their probability.

These two views are both useful

1. A procedure for (approximately) ***sampling*** from the distribution from which a dataset was drawn
2. A procedure for (approximately) ***evaluating the probability density*** of a datapoint under the distribution from which a dataset was drawn

Application: visual creativity

Application: outlier detection



These two views are both useful

1. A procedure for (approximately) ***sampling*** from the distribution from which a dataset was drawn

2. A procedure for (approximately) ***evaluating the probability density*** of a datapoint under the distribution from which a dataset was drawn

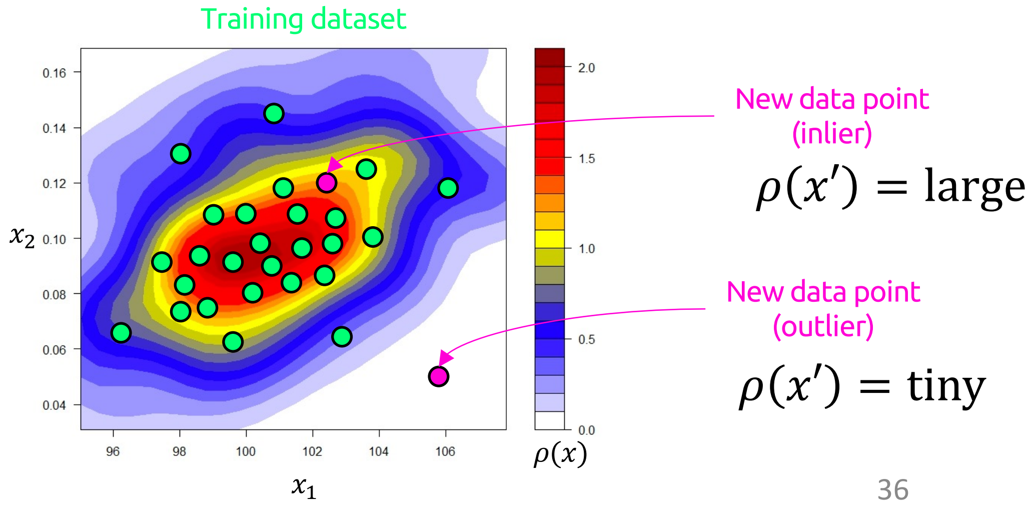
Application: things are getting more complicated!

Computer-generated inclusivity: fashion turns to 'diverse' AI models

Fashion brands including Levi's and Calvin Klein are having custom AI models created to 'supplement' representation in size, skin tone and age

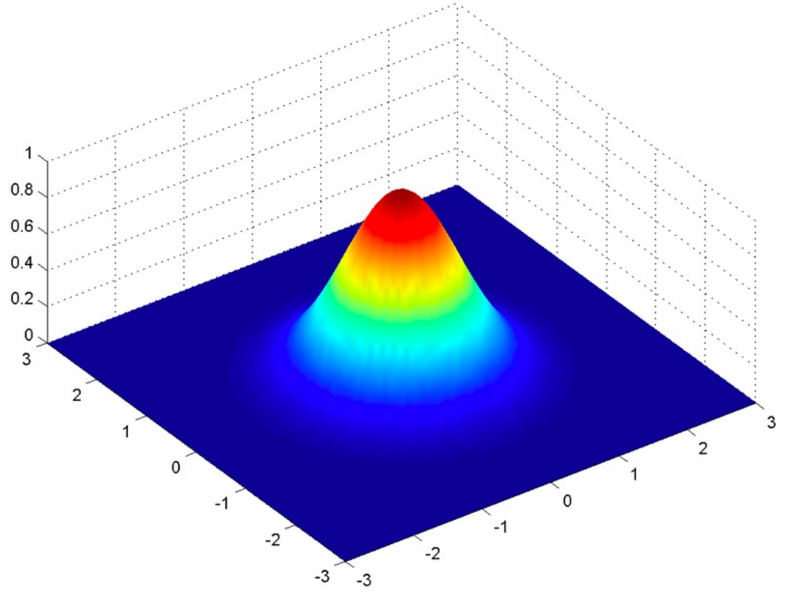
[The Guardian]()

Application: outlier detection



What are some example generative models?

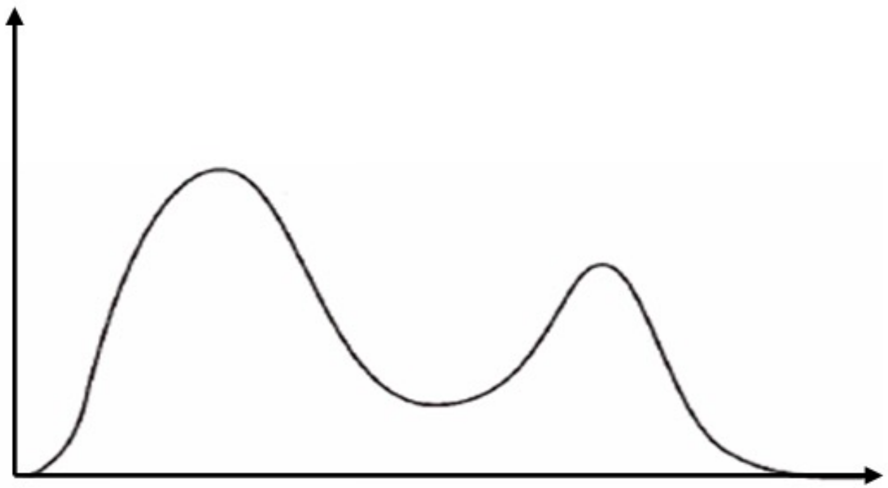
- Any probability distribution can be a generative model
- You already know some of these!

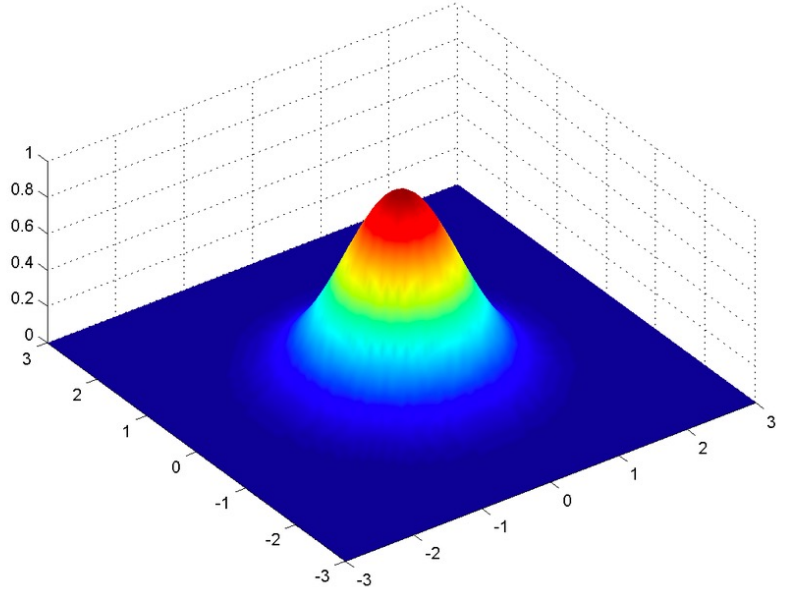


- E.g. The Gaussian Distribution
  - $p(x \mid \mu, \sigma) = \mathcal{N}(\mu, \sigma)(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
  - Sampling:
    - Sample from the unit normal distribution $\rightarrow\ r \sim \mathcal{N}(0,1)$
    - Return $\mu + r\sigma$

Disadvantages of Gaussian distribution

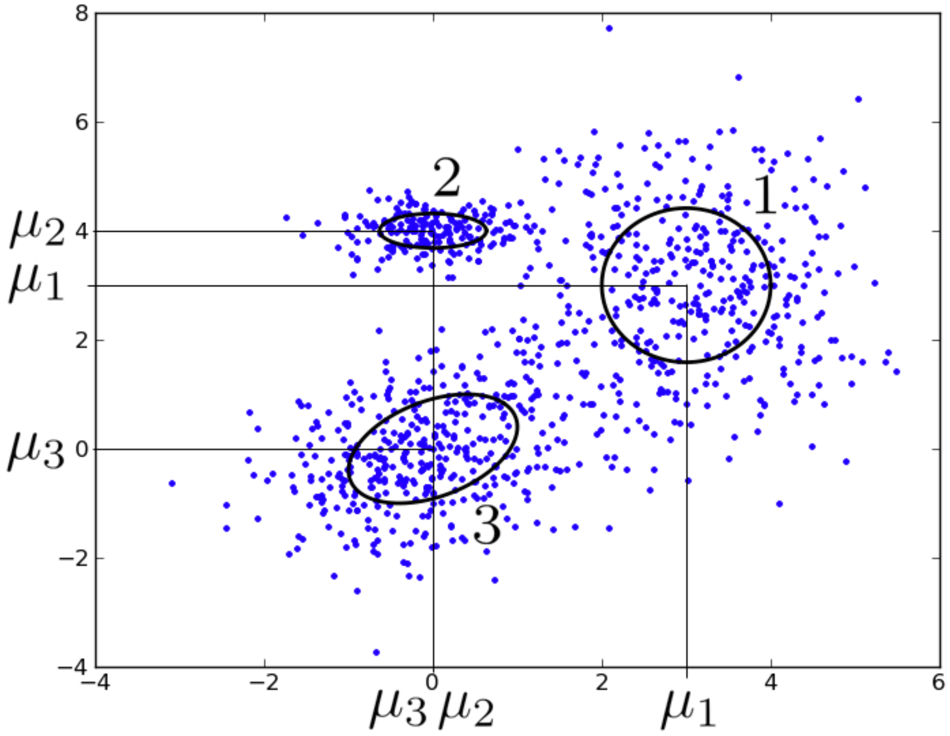
- Can only represent distributions with a single ***mode***
    - What if the distribution has multiple "peaks?"
    - E.g. book prices (concentrates around different price points if it's hardcover, paperback, e-book, ...)



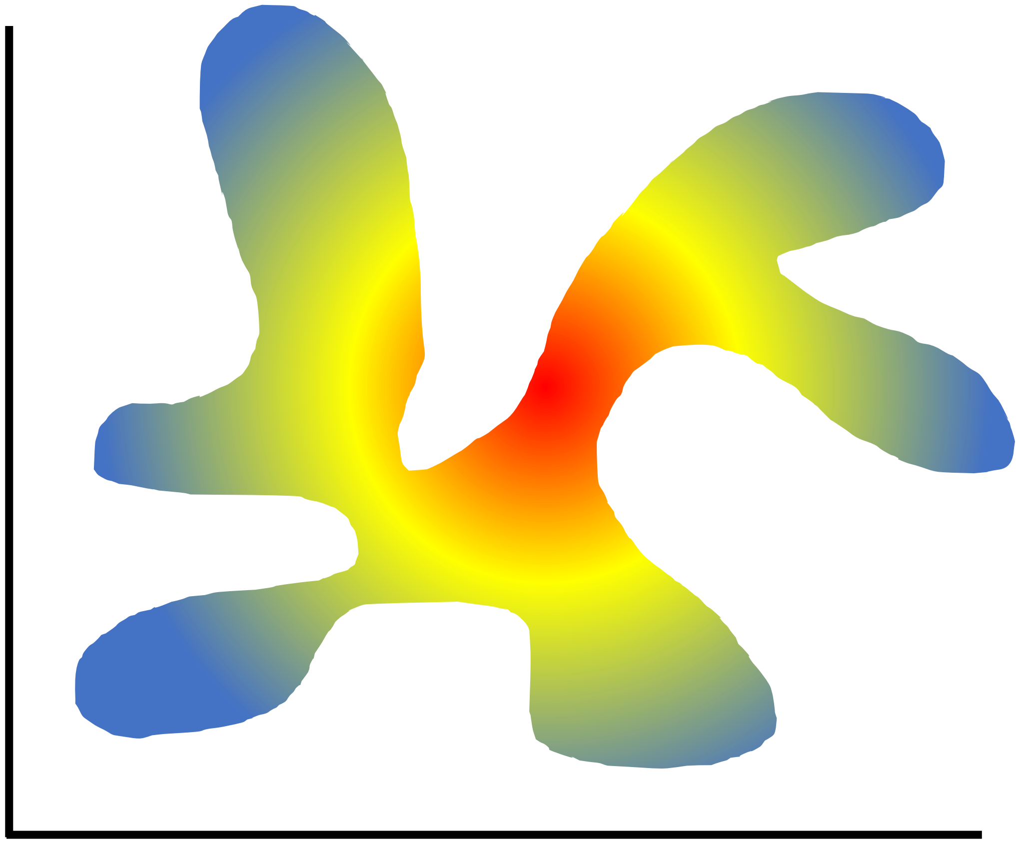


Better: *Mixture* of Gaussians

- A linear combination of multiple individual Gaussian distributions
  - $p(x \mid w, \mu, \sigma) = \sum_i w_i \mathcal{N}(\mu_i, \sigma_i)(x)$
  - Sampling:
    - Sample from the discrete weight distribution w to choose a Gaussian
    - Sample from that Gaussian as before



What about something like this?



- This doesn't look like a linear combination of Gaussians...
- ...but maybe it can be expressed as a ***nonlinear*** function of Gaussians?

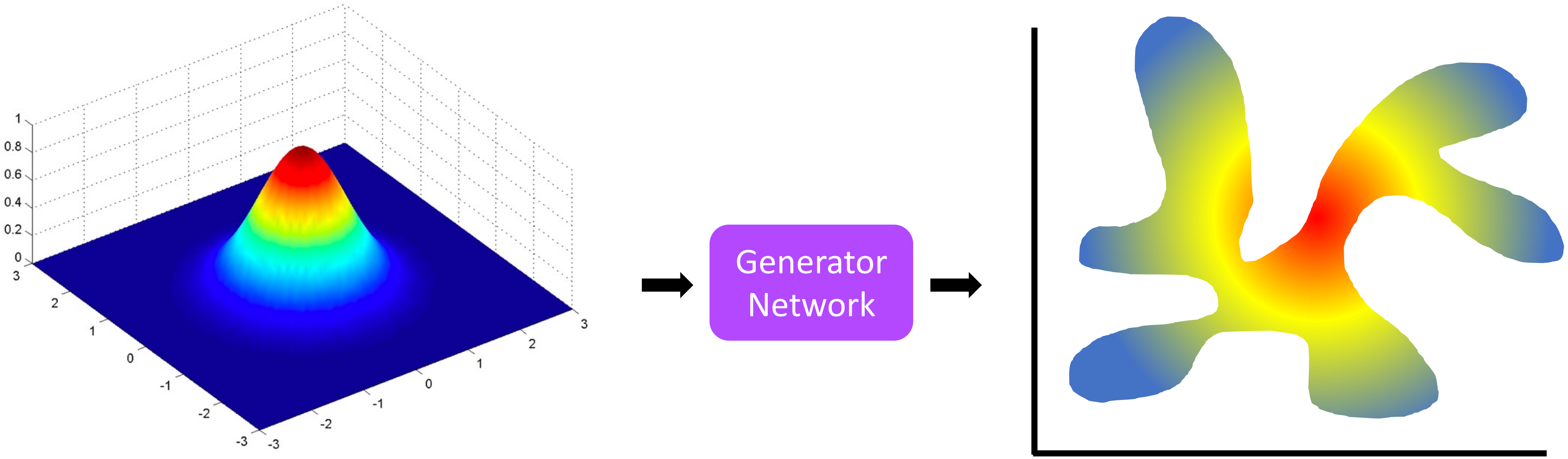
What can we do?

“I hear these neural nets are pretty good at learning non-linear functions” ☺

Any questions?

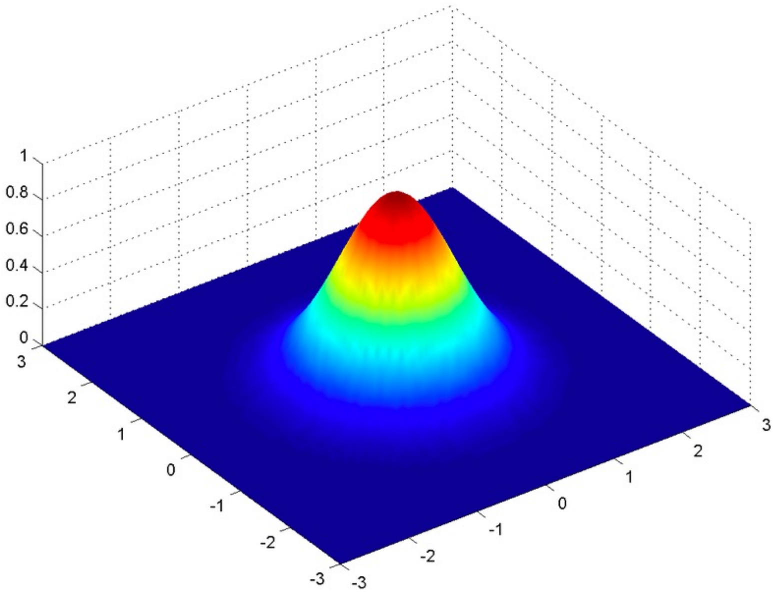
A Neural Generative Model

- Input: a point $z \in \mathbb{R}^n$ drawn from a normal distribution $\mathcal{N}(\mu, \sigma)$
- Output: a point $x \in \mathbb{R}^m$ distributed according to some more complex distribution



A Neural Generative Model

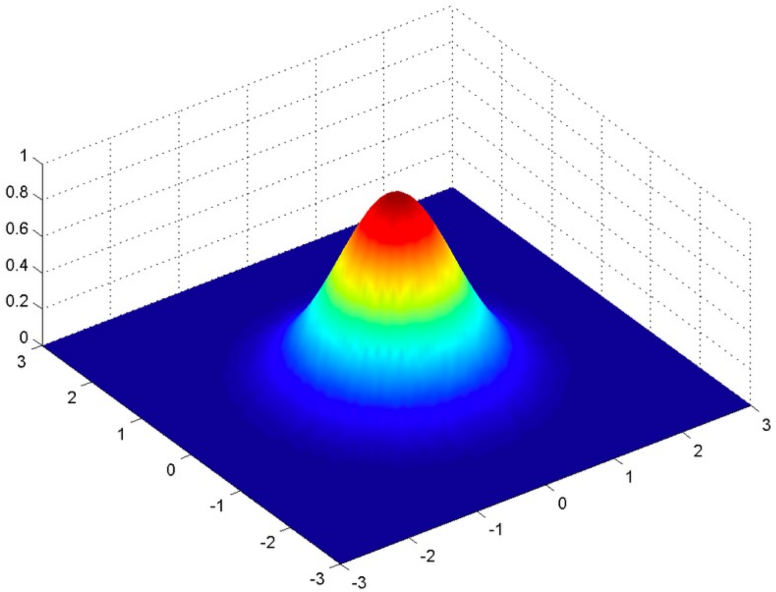
- Input: a point $z \in \mathbb{R}^n$ drawn from a normal distribution $\mathcal{N}(\mu, \sigma)$
- Output: a point $x \in \mathbb{R}^m$ distributed according to some more complex distribution



Generator Network

A Neural Generative Model

- Input: a point $z \in \mathbb{R}^n$ drawn from a normal distribution $\mathcal{N}(\mu, \sigma)$
- Output: a point $x \in \mathbb{R}^m$ distributed according to some more complex distribution

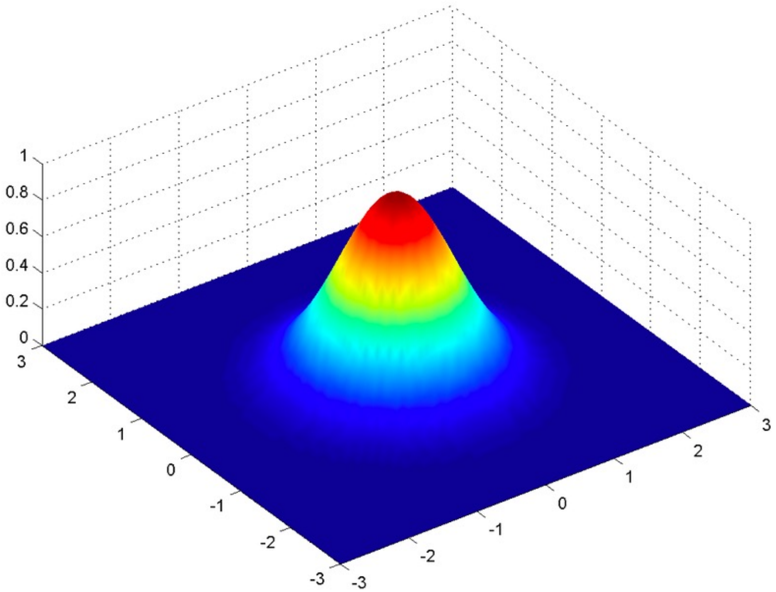


Generator Network

The distribution of human faces

A Neural Generative Model

- Great! So...how do we train this thing?
  - Let's modify our autoencoder to achieve this

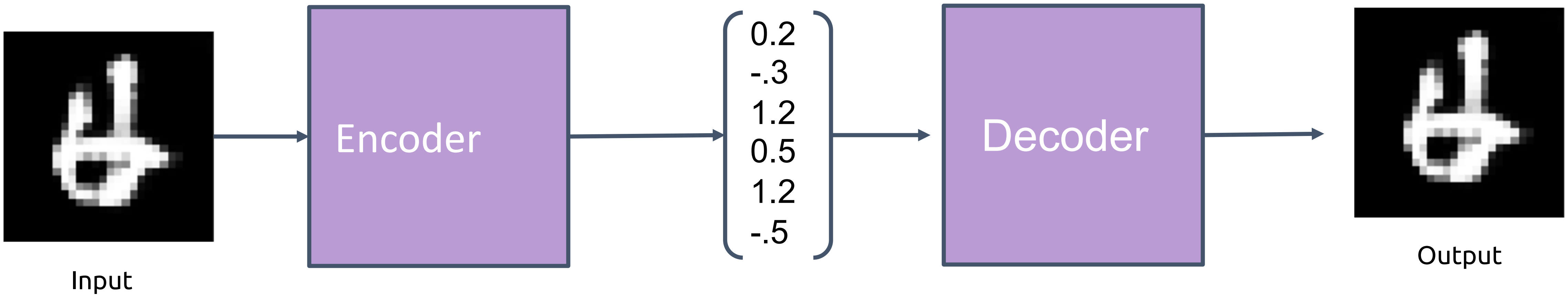


→ Generator Network →

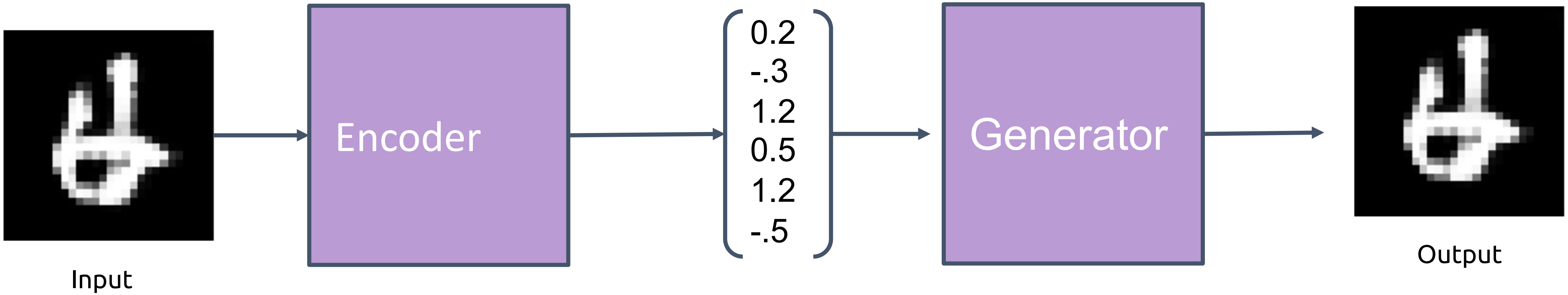
The distribution of human faces

Autoencoder

Let’s think for a bit – how to modify the autoencoder to make it a generative model?

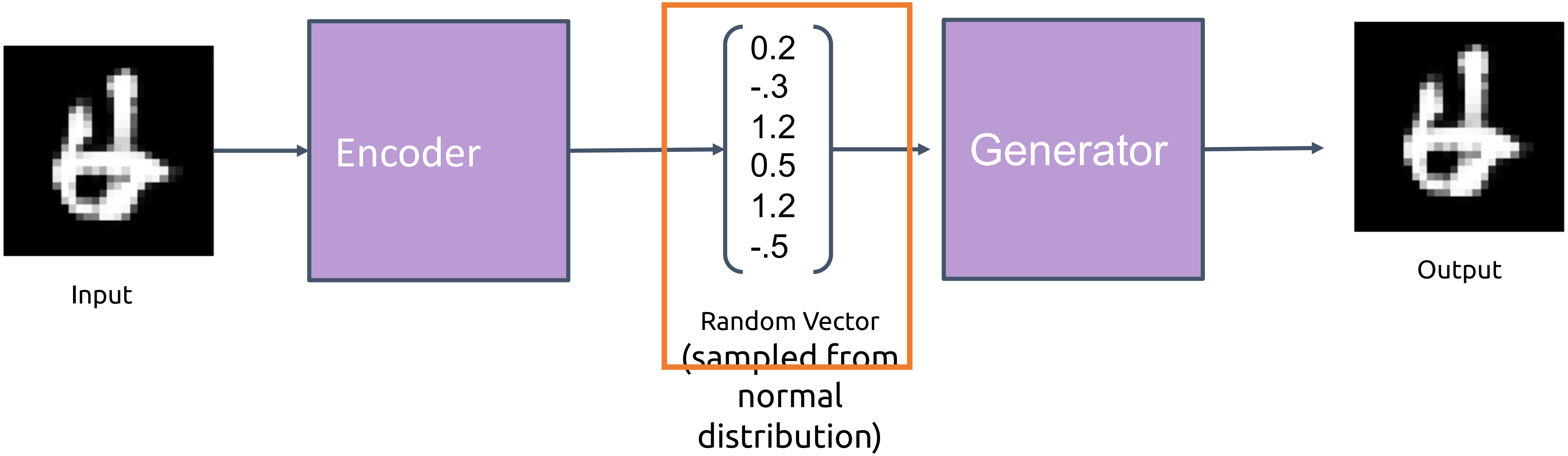


Variational Autoencoders (VAEs)



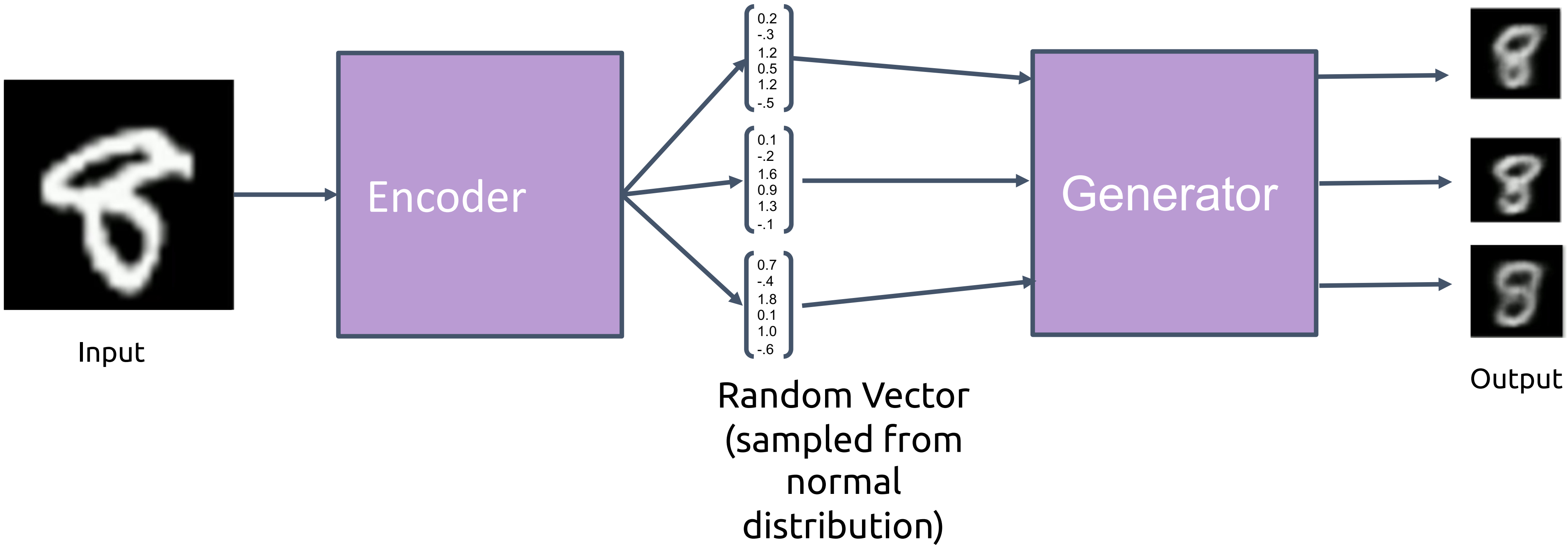
Variational Autoencoders (VAEs)

- This looks almost exactly like an autoencoder...
- ...except that this bottleneck vector is randomly sampled
  - We'll see how in a few slides



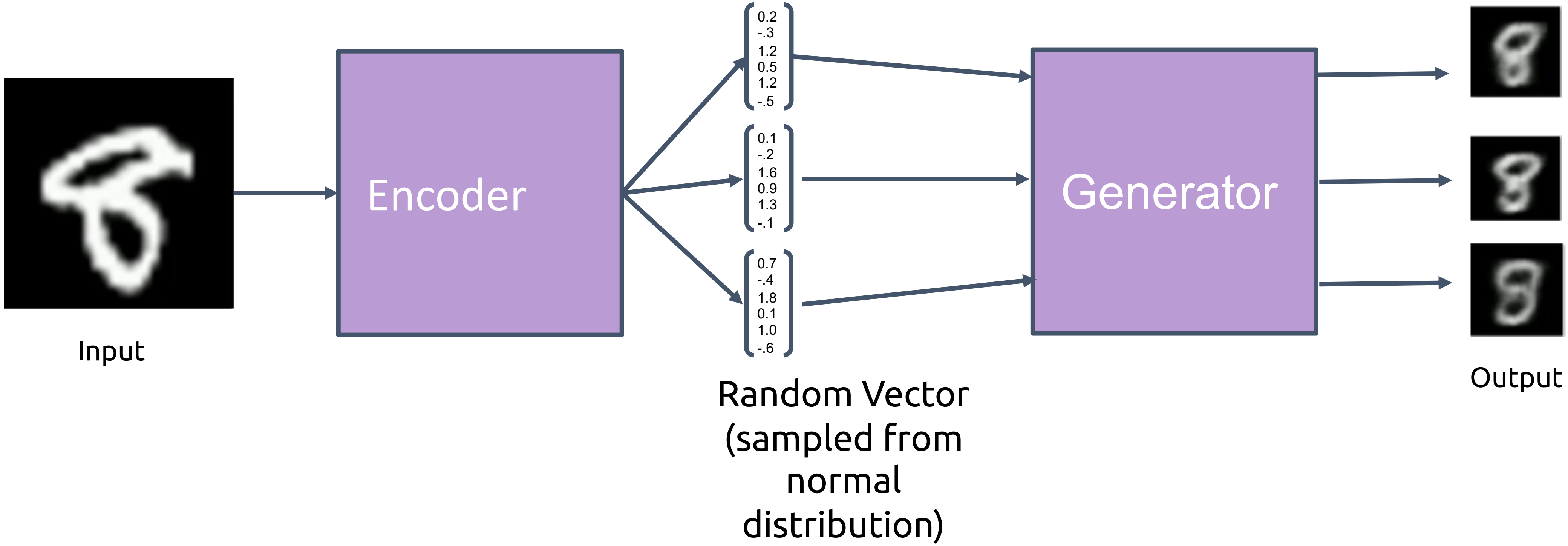
Variational Autoencoders (VAEs)

- In fact, the encoder can produce multiple different random vectors...
- ...which then lead to different outputs which are variants of the input



Variational Autoencoders (VAEs)

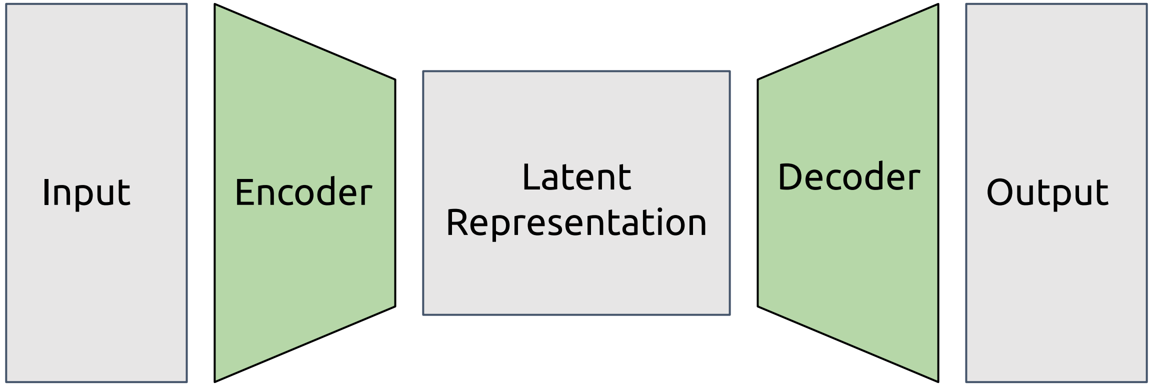
- ***Why do this?***
  - We'll see shortly how this setup allows for a nice, stable learning algorithm
  - (It's actually just a small modification to how autoencoders are trained)



Building up the VAE Architecture

If we were to describe an autoencoder functionally:

$$\texttt{Output = Decoder}(\overbrace{\underbrace{\texttt{Encoder(Input)}}_{\text{Latent Representation}}})$$

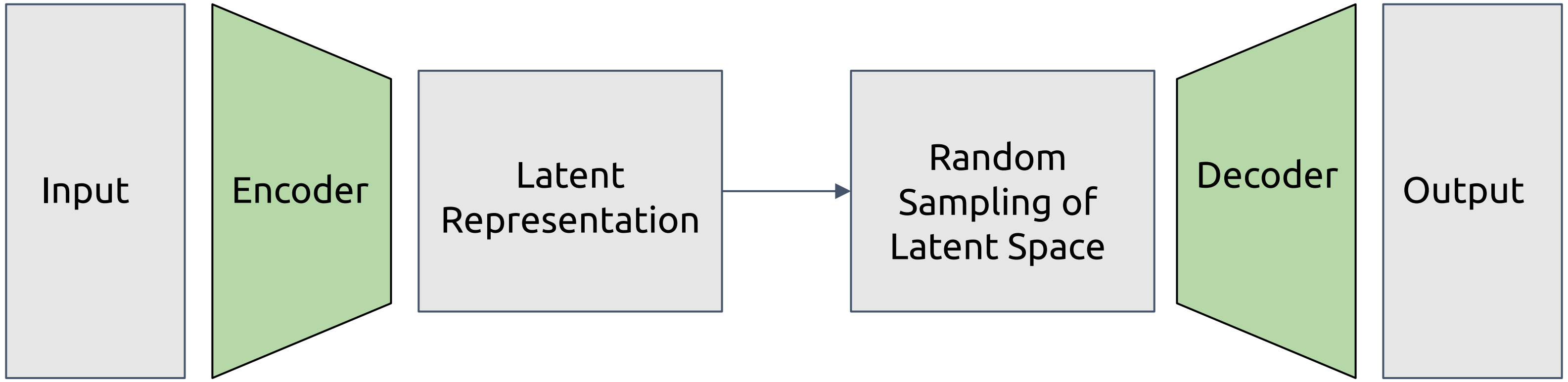


*Latent representation ~ compressed representation

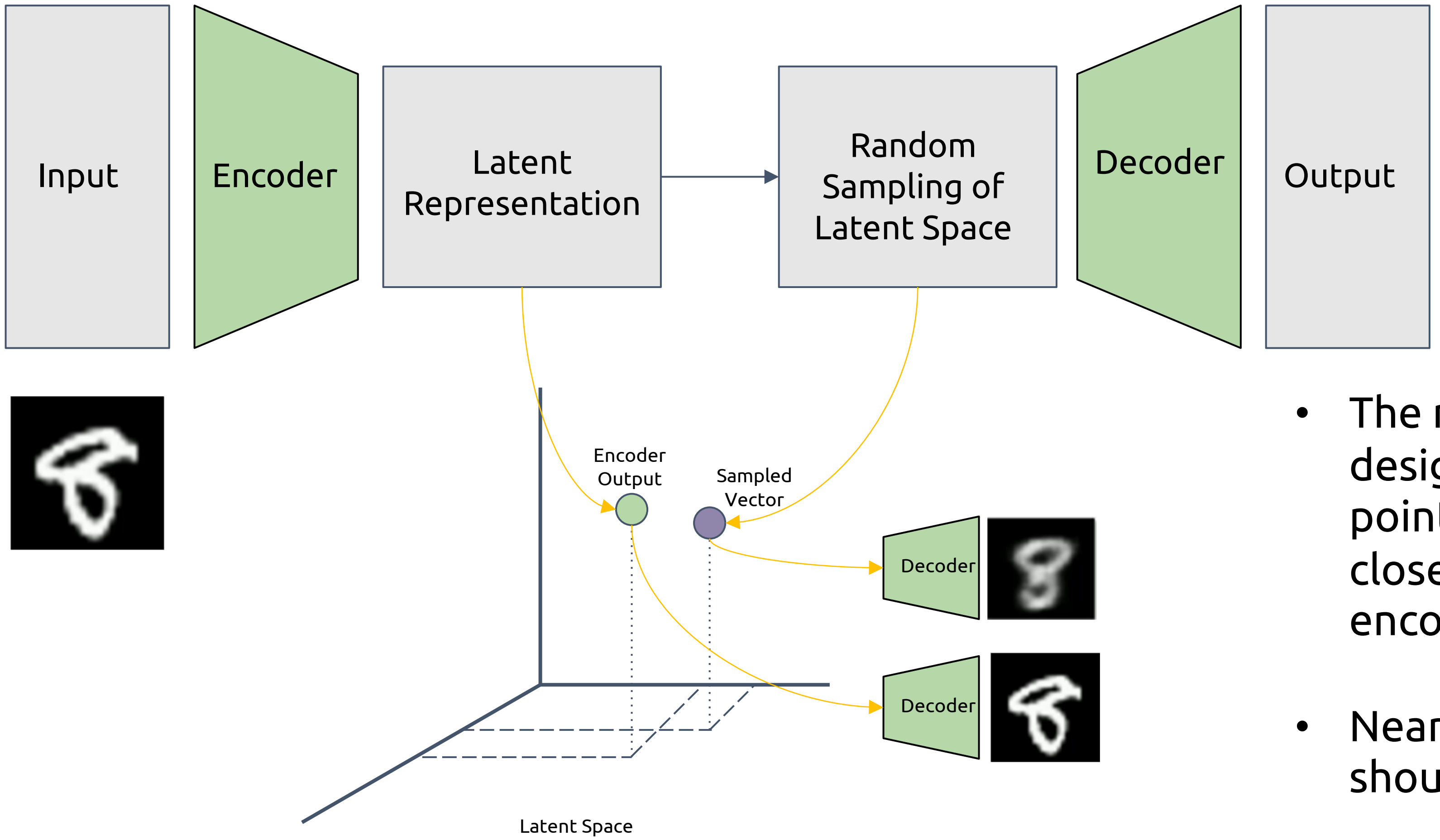
Building up the VAE Architecture

For variational autoencoders, we also do a random sampling operation at the bottleneck

```
Output = Decoder(random_sample(Encoder(Input)))
```



How does random sampling in latent space lead to variation?



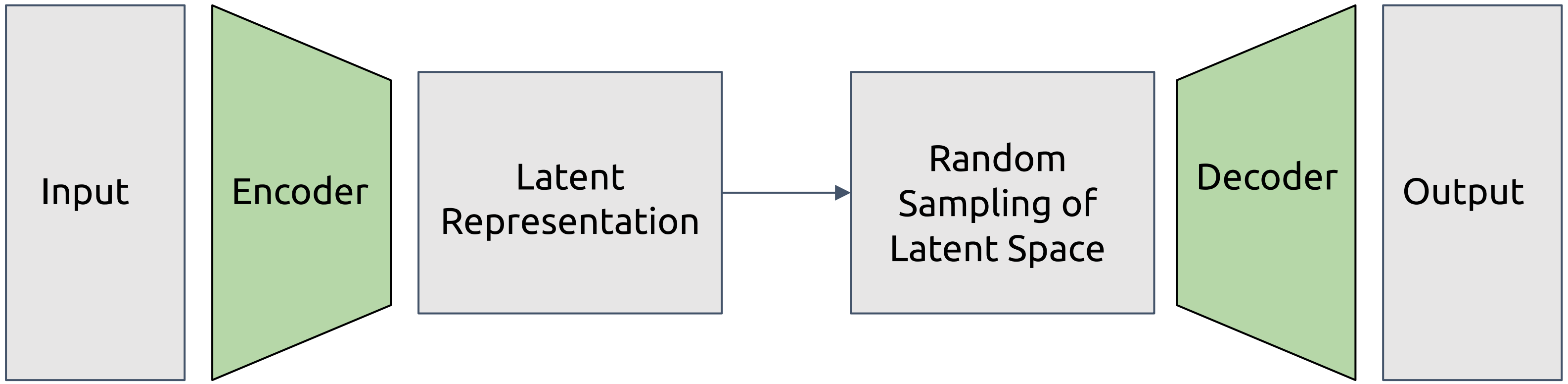
- The random sampling should be designed to produce random points in latent space that are close to the output of the encoder
- Nearby points in the latent space should decode to similar images

How should **`random_sample`** be defined?

`Output = Decoder(`**`random_sample`**`(Encoder(Input)))`

- We want the sample to be close to the encoder output
- One option: sample from a Gaussian centered at `Encoder(Input)`

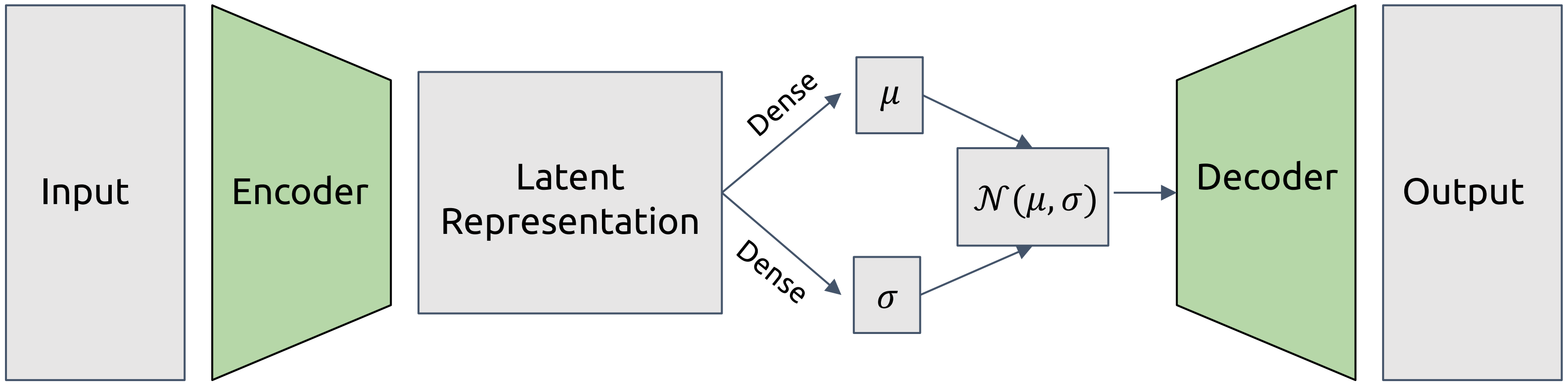
What can we modify?



How should **`random_sample`** be defined?

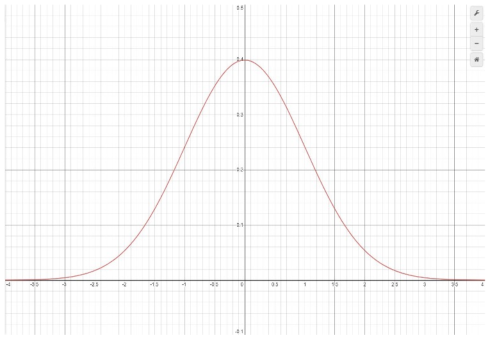
`Output = Decoder(`**`random_sample`**`(Encoder(Input)))`

- We want the sample to be close to the encoder output
- One option: sample from a Gaussian centered at `Encoder(Input)`
- Use two dense layers to convert the encoder output into the mean and standard deviation of the Gaussian

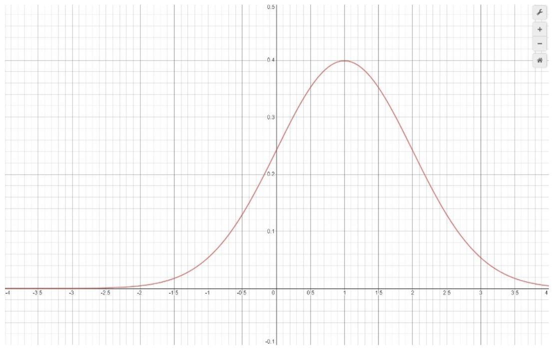


Any questions?

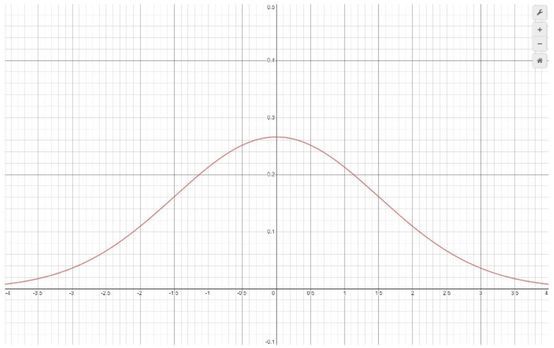
How should **`random_sample`** be defined?



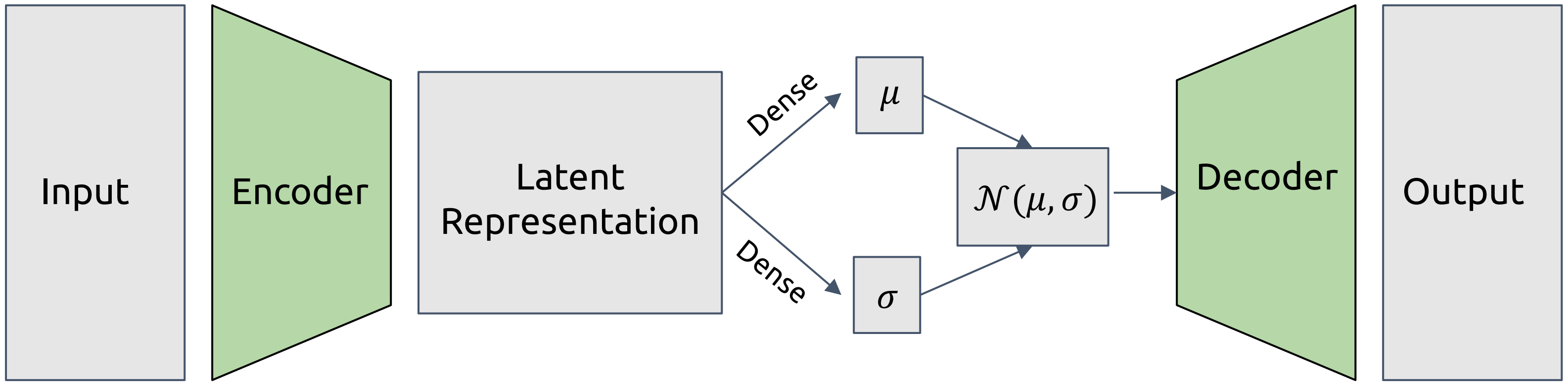
$\mu = 0$
$\sigma = 1$



$\mu = 1$
$\sigma = 1$



$\mu = 0$
$\sigma = 1.5$

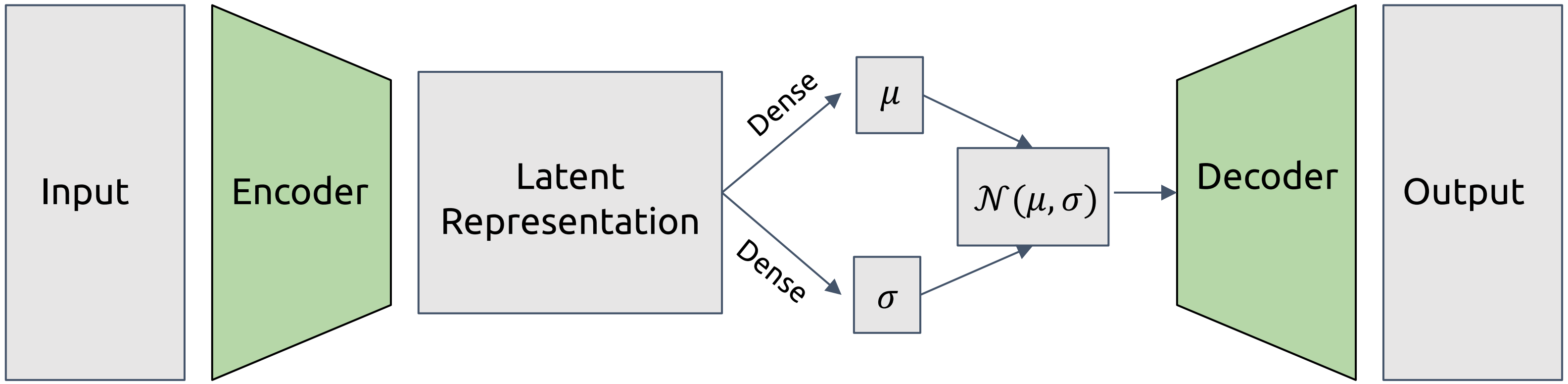


Training a VAE

Two goals:

1. Reproduce an output similar to the input (Input ≈ Output)
2. Have some variation in our output (Input ≉ Output)

- Seems like two conflicting goals!
- How do we resolve these two goals?



Weighted Combination of Losses

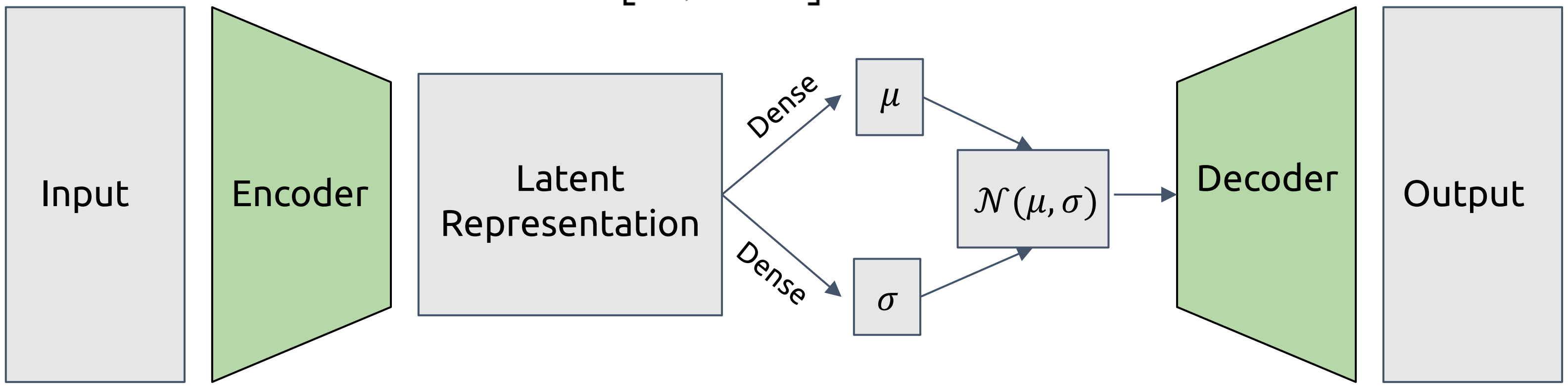
L_1 = loss associated with producing output similar to input

L_2 = loss associated with producing output with some variation to input

Total Loss:

$$L = L_1 + \lambda L_2$$

$$\lambda \in [0, \infty]$$



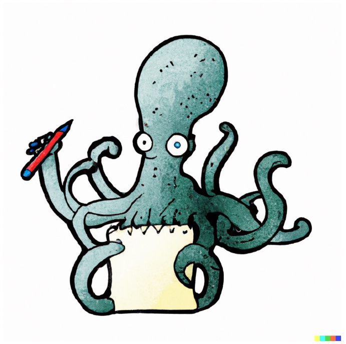
Recap

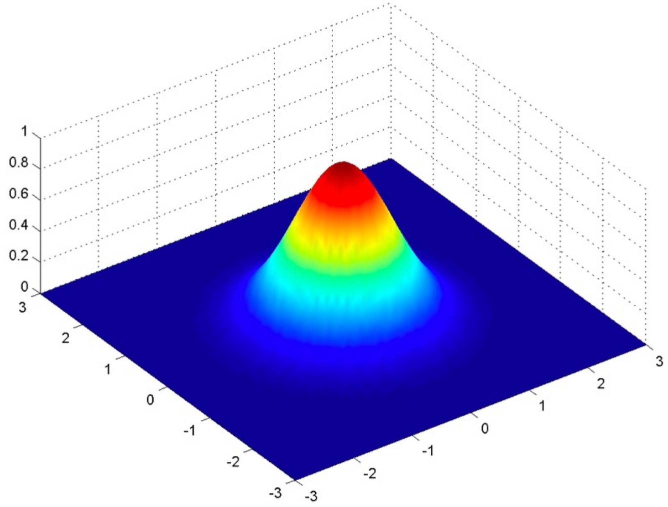
Convolutional AEs

Generative Modeling

Generative modeling – formulation and applications

Probability distributions = generative models



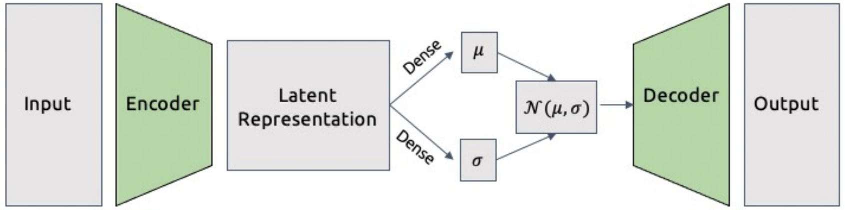


Variational Autoencoders (VAEs)

Generative modeling for complex distributions

Modifying AEs

VAE architecture

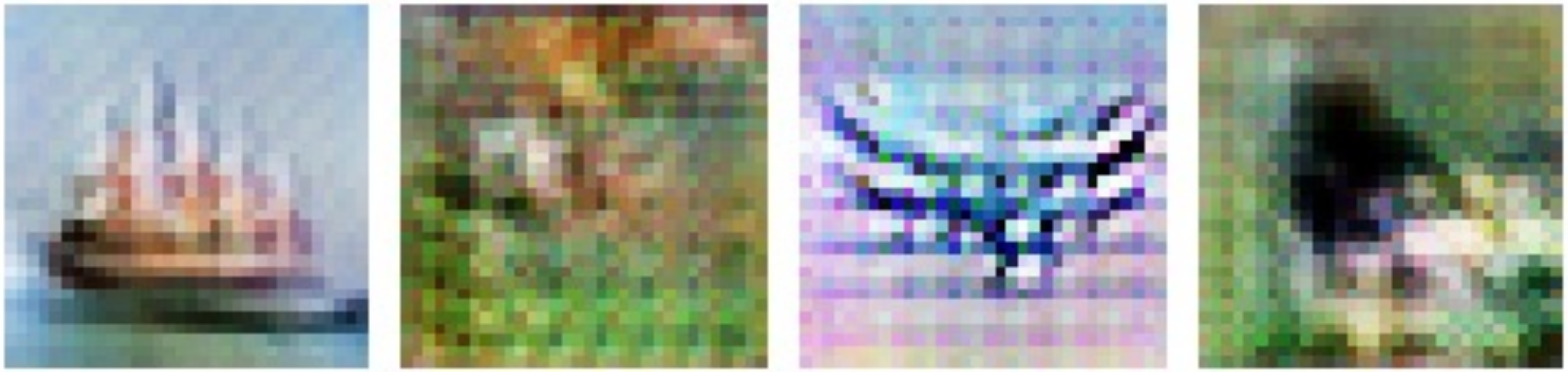


Extra material

[More reading on Transpose Convolution]

Caution: Checkerboard Artifacts

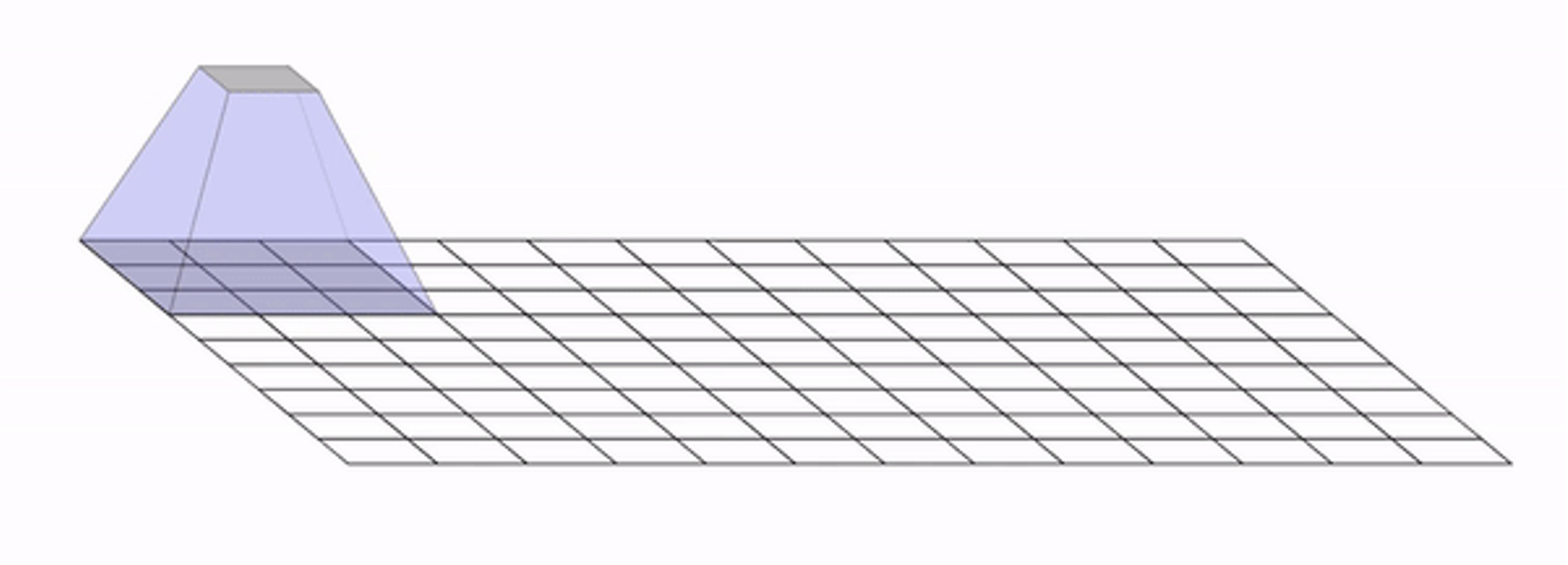
- Transpose convolution causes artifacts in output images



Great visual (and more!) from this article: https://distill.pub/2016/deconv-checkerboard/

Caution: Checkerboard Artifacts

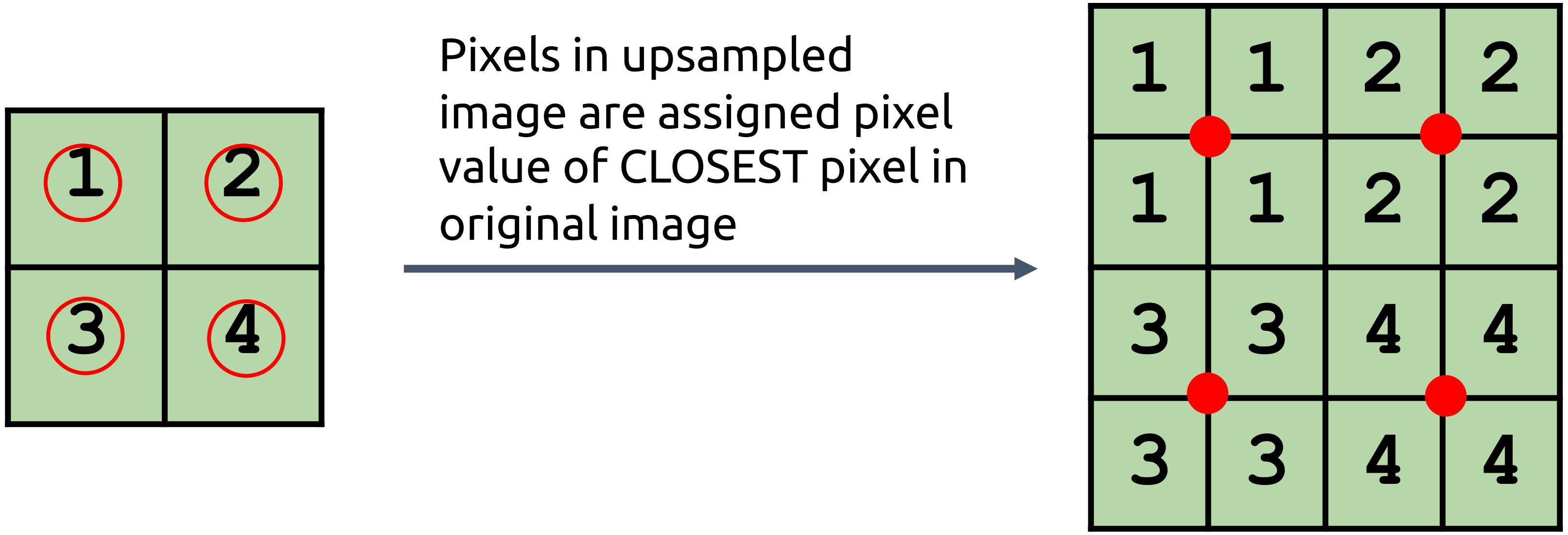
- Transpose convolution causes artifacts in output image
- Why? Some pixels get written to more often than others
- Is there a better way to upsample?



Great visual (and more!) from this article: https://distill.pub/2016/deconv-checkerboard/

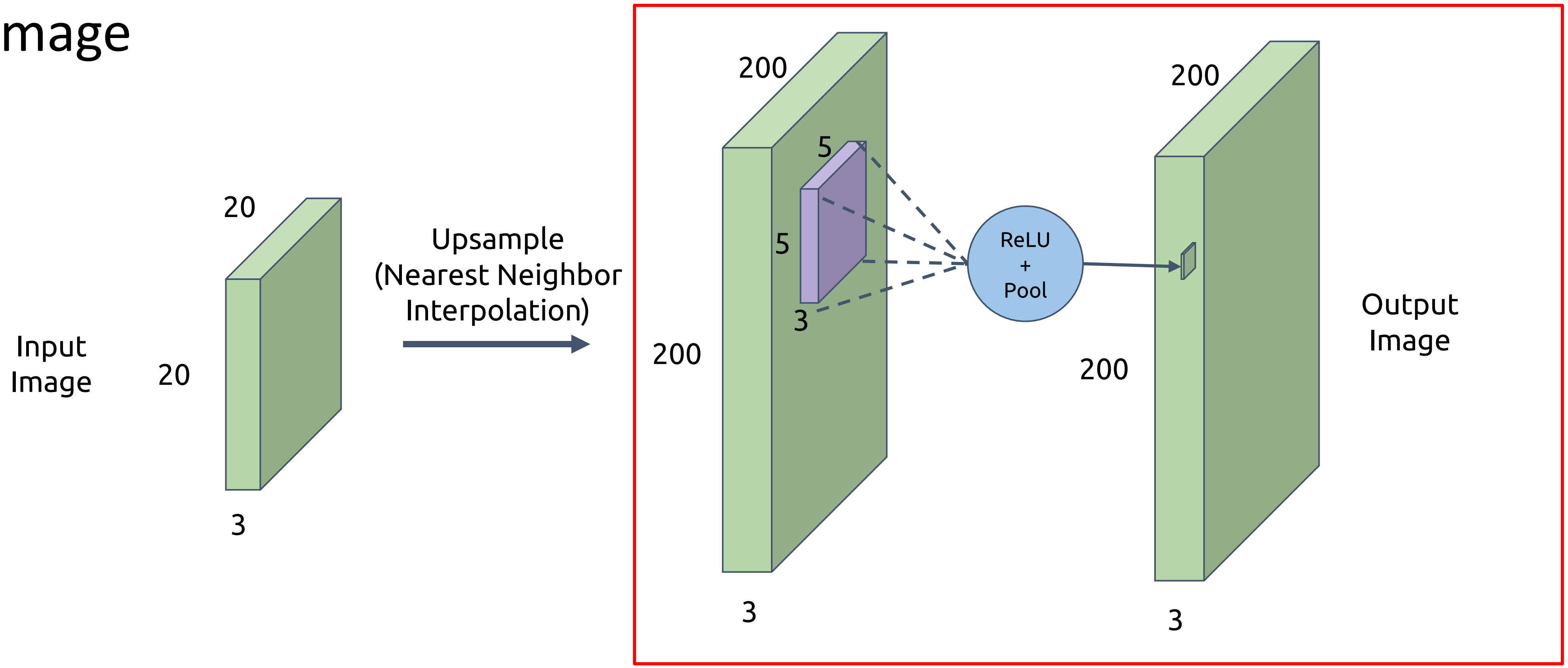
Eliminating checkerboard artifacts

Step 1: Upsample using nearest neighbor interpolation:



Eliminating checkerboard artifacts

Step 2: Perform a convolution with SAME padding on the upsampled image



Dealing With it in Tensorflow

```
# Layer to upsample the image by a factor of 5 in x and y using nearest
# neighbor interpolation
tf.keras.layers.UpSampling2D(size=(5, 5), interpolation='nearest')

# Do a convolutional layer on the result
tf.keras.layers.Conv2D(filters = 1, kernel_size = (10,10), padding = "SAME")
```

Checkerboard Artifacts Resolved

With Transpose Convolution

With Resize + Convolution

Great visual (and more!) from this article: https://distill.pub/2016/deconv-checkerboard/