CIS 6930 Topics in Computing for Data Science
Week 2: Autoencoders

Tue 9/14/2021

Yoshihiko (Yoshi) Suhara

2pm-3:20pm & 3:30pm-4:50pm

Course Plan (1/2)

- ~~Week 1: Deep Learning Basics (Thu 9/9)~~
- **Week 2: AutoEncoder (Tue 9/14)**
- Week 3: Convolutional Neural Networks (Thu 9/16)
- Week 4: GAN (Tue 9/21)
- Week 5: Word embeddings: Word2vec, GloVe (Thu 9/23)
- Week 6: Recurrent Neural Networks (Tue 9/28, Thu 9/30)
- Week 7: Review/Project pitch & Mid-term (Tue 10/5, Thu 10/7)
- Fall Break
- ...

Course Plan (1/2)

- ~~Week 1: Deep Learning Basics (Thu 9/9)~~
- **Week 2: AutoEncoder (Tue 9/14)**
- **Week 3: Convolutional Neural Networks (Thu 9/16)**
- **Week 4: GAN (Tue 9/21)**

Deep Learning techniques for images

- Week 5: Word embeddings: Word2vec, GloVe (Thu 9/23)
- Week 6: Recurrent Neural Networks (Tue 9/28, Thu 9/30)

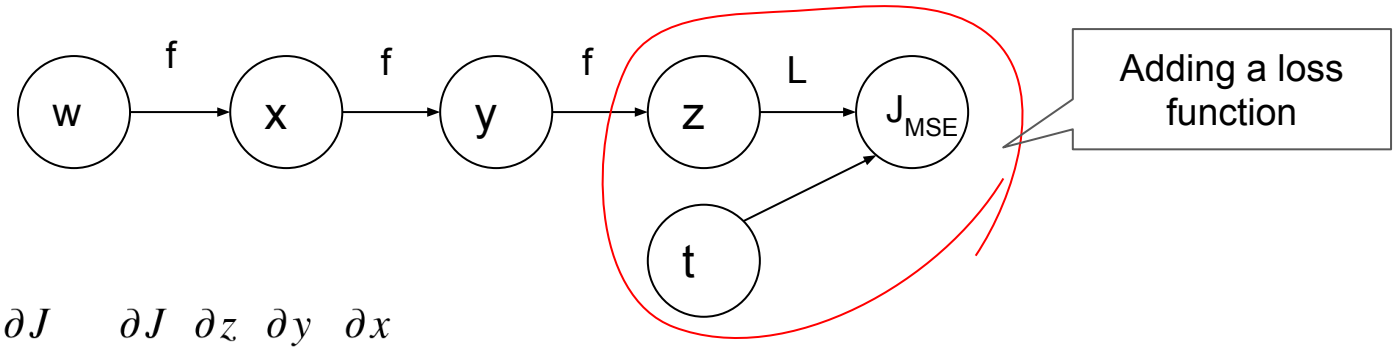
for text

- Week 7: Review/Project pitch & Mid-term (Tue 10/5, Thu 10/7)
- Fall Break
- ...

Recap

Backpropagation from 3000 ft

- Multiplications of **the derivative** of each step + **Outputs** of the previous step



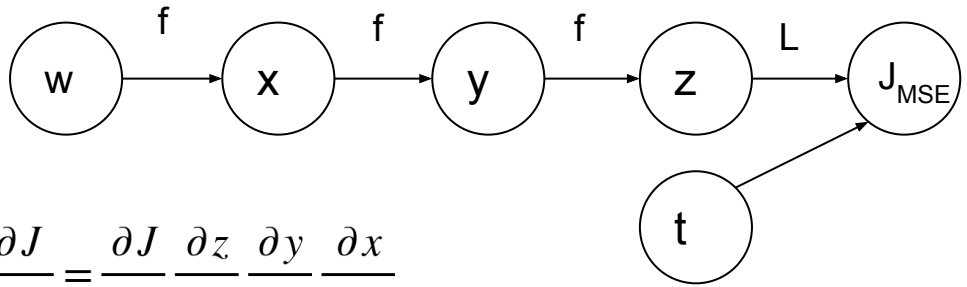
$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}$$

$$= L'(z, t) f'(y) f'(x) f'(w)$$

$$= L'(f(f(f(w))), t) f'(f(f(w))) f'(f(w)) f'(w)$$

Differentiability is Key!

- The gradients of any parameters can be calculated **as long as the functions are differentiable!**
- Backpropagation = A gradient calculation method
  - i.e., can be coupled with any optimization method (e.g., SGD)
- Multiplications of **the derivative** of each step + **Outputs** of the previous step



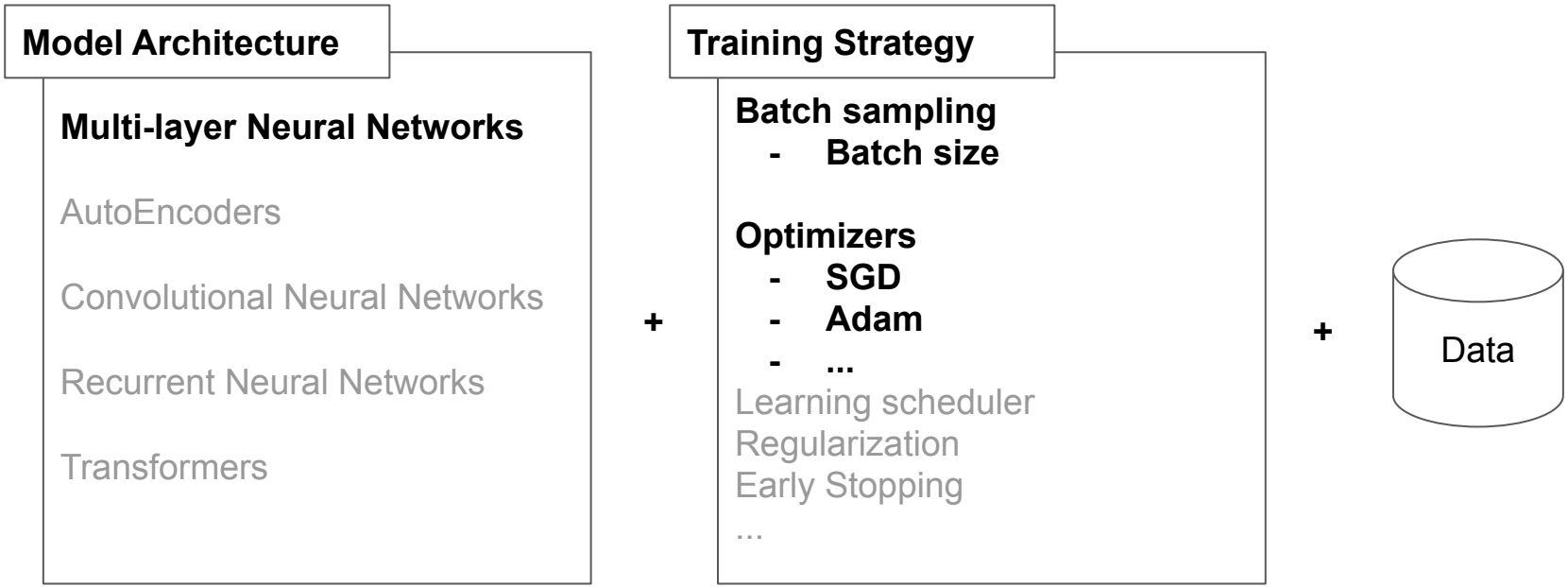
$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}$$

$$= L'(z, t) f'(y) f'(x) f'(w)$$

$$= L'(f(f(f(w))), t) f'(f(f(w))) f'(f(w)) f'(w)$$

Basic Deep-Learning Building Blocks

- **(A) Model Architecture + (B) Training Strategy + (c) Data**
- i.e., What to Optimize and How to Optimize



Deep-Learning Building Blocks: Starter Kit

- Layers
  - Linear Layer
- Activation functions
  - Logistic sigmoid, tanh
  - ReLU, Leaky ReLU
- Optimizers
  - SGD w/wo Momentum)
  - Adam

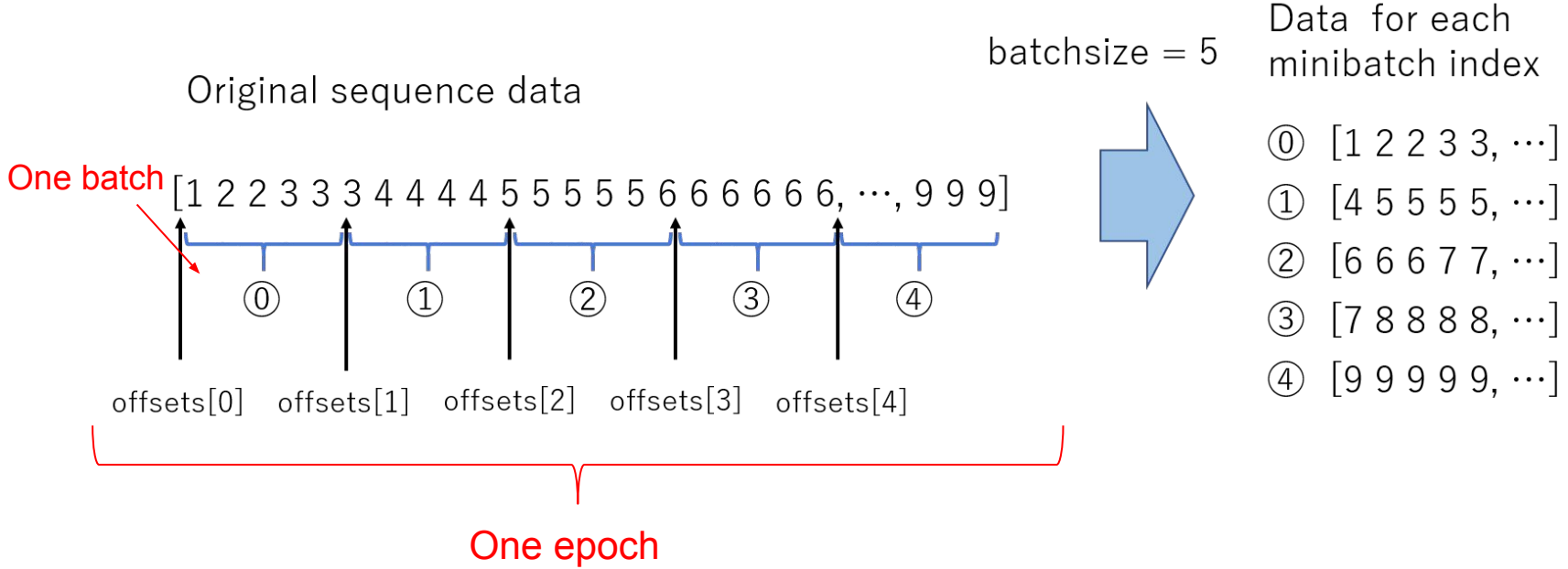
PyTorch Session Takeaway: D-M-T

- Remember **D**ataset, **M**odel, **T**raining iteration

```
for epoch in range(10):
    training_loss = 0.0
    valid_loss = 0.0
    model.train()
    for batch_idx, batch in enumerate(dataloader):
        optimizer.zero_grad()                   # Initialize gradient information
        X, y = batch                            # Get feature/label from batch
        pred = model(X)                         # Get output from the model
        loss = criterion(pred, y)               # Evaluate loss values
        loss.backward()                         # Backpropagate
        optimizer.step()                        # Update parameters

        training_loss += loss.data.item() * batch_size
    training_loss /= len(train_iterator)
```

How Mini-batch Sampling Works?

- Google Colab



http://corochann.com/training-rnn-with-simple-sequence-dataset-1304.html

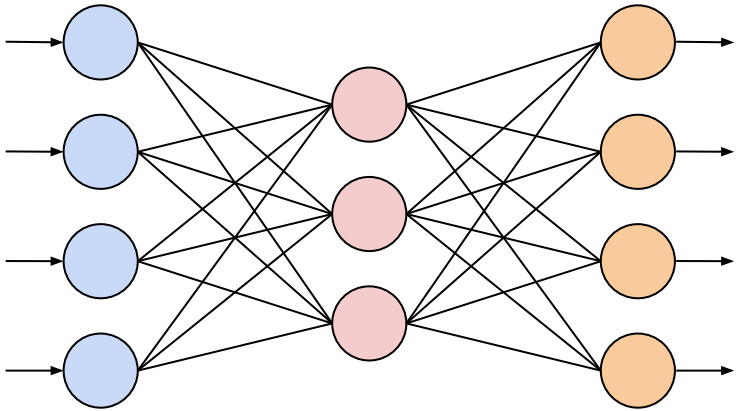
Tips: How to Debug Google Colab Code?

- Google Colab Notebooks store variables
  - You can check variables in any cells **after execution**
  - You can create a **breakpoint** by raising an exception (e.g., sys.exit())
  - It’s naive but works!

- A more sophisticated way
  - The method above does not work for debugging a function
  - Use pdb by inserting `import pdb; pdb.set_trace()`

Today’s Topic: Autoencoders!

- Autoencoders principles
- Advanced Autoencoder models
- Applications



Today’s Teaching Style

- Lecture & In-class exercises
  - To help get (more) familiar with implementing Neural Network model classes with PyTorch on your own

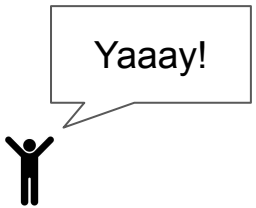
- [Alert] (A little bit) more open-ended assignments are coming on Thursday!

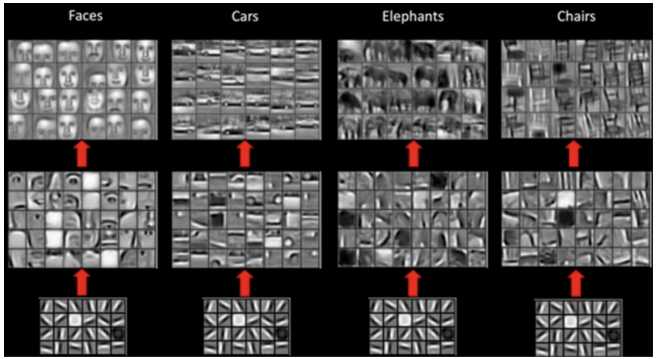
Autoencoders Principles

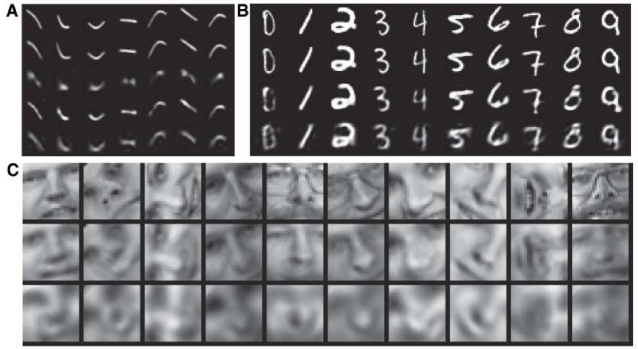
Return of Neural Networks: Deep Learning (2010s-)

- Maturity of Neural Networks techniques → Representation Learning
  - **Auto-Encoders (2006)**
  - Advances in Convolutional Neural Networks (originally 1998)

- The age of Big Data
  - Computational resource ↑
  - Data size ↑

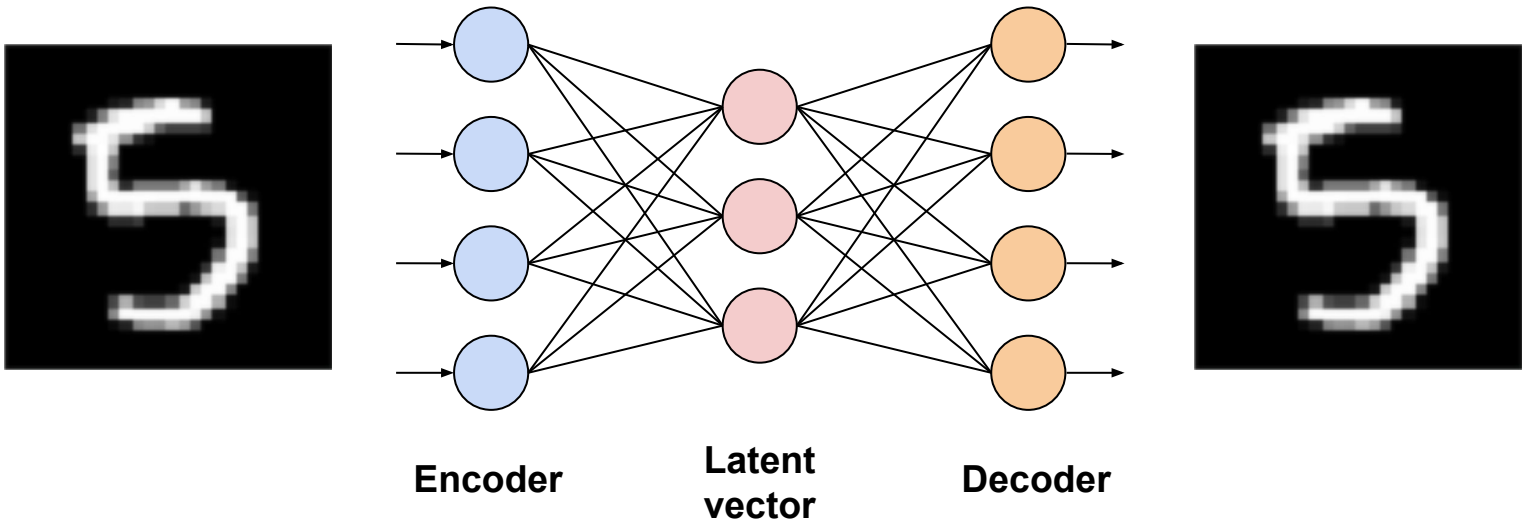




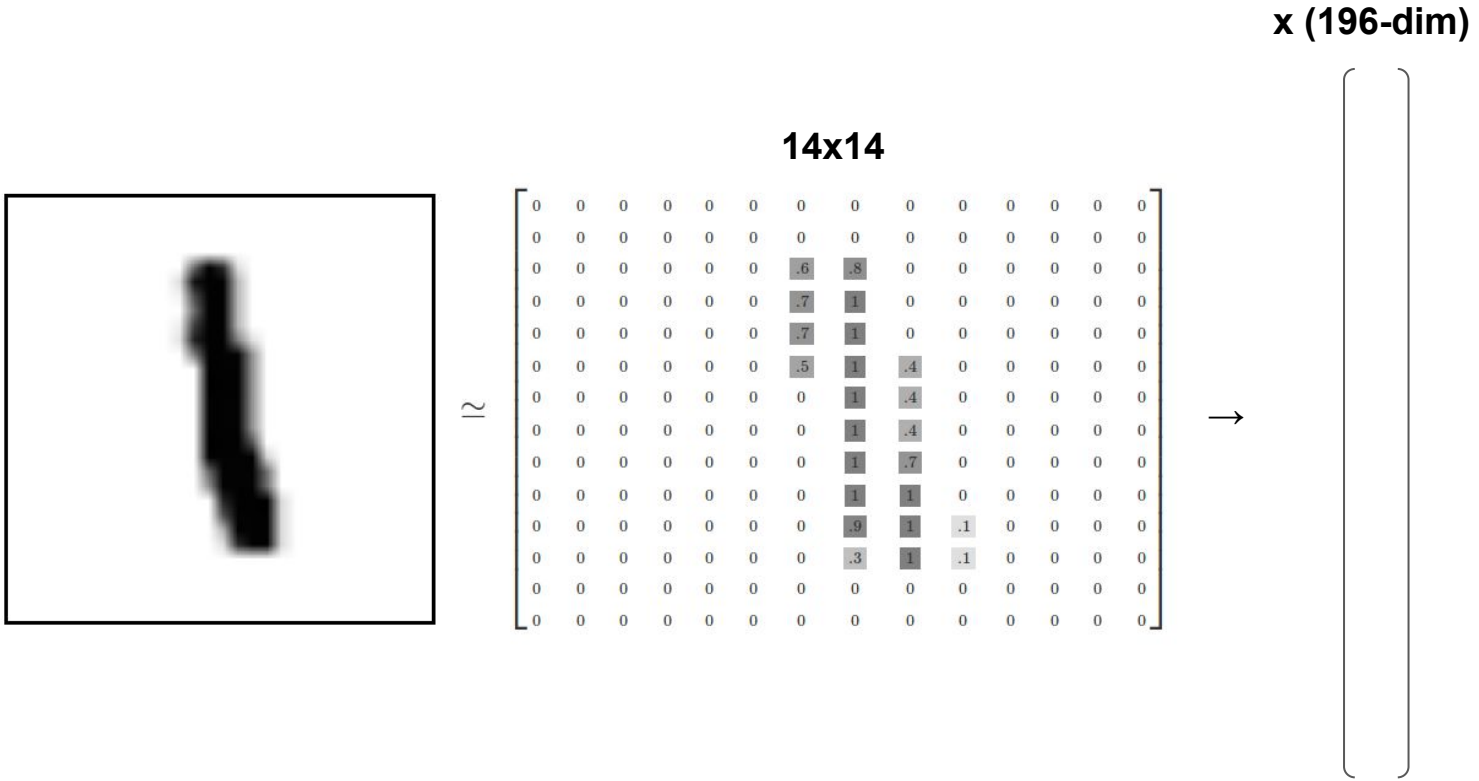


Autoencoders

- **Encoder-decoder models** that **learn to reconstruct the original data**



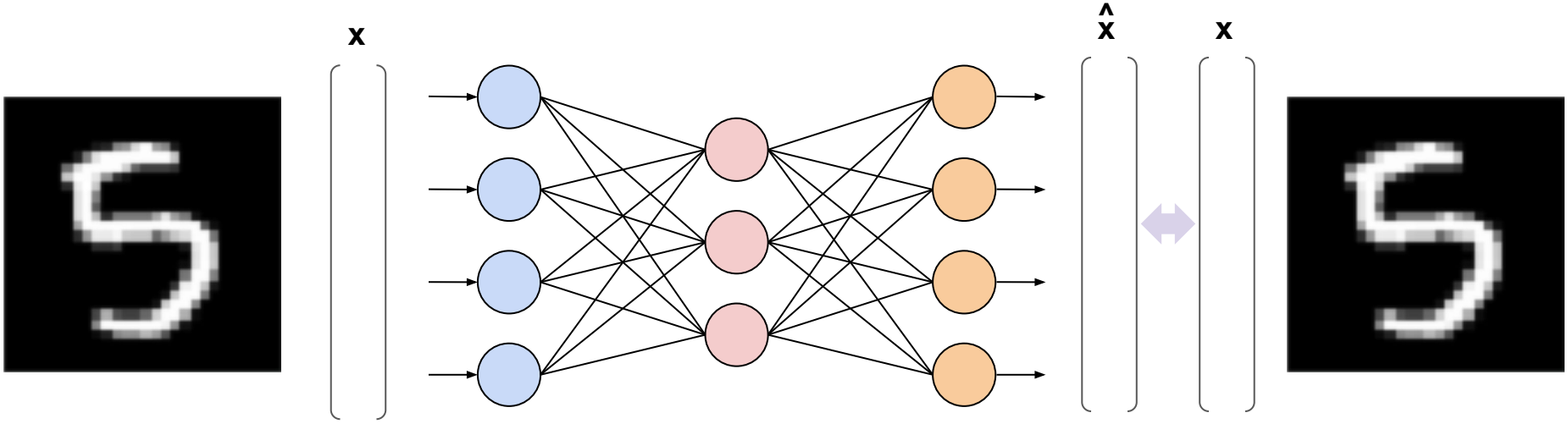
Note: Pixel Image → Input vector



We will discuss how to leverage the spatial information of pixels

Autoencoders: Training

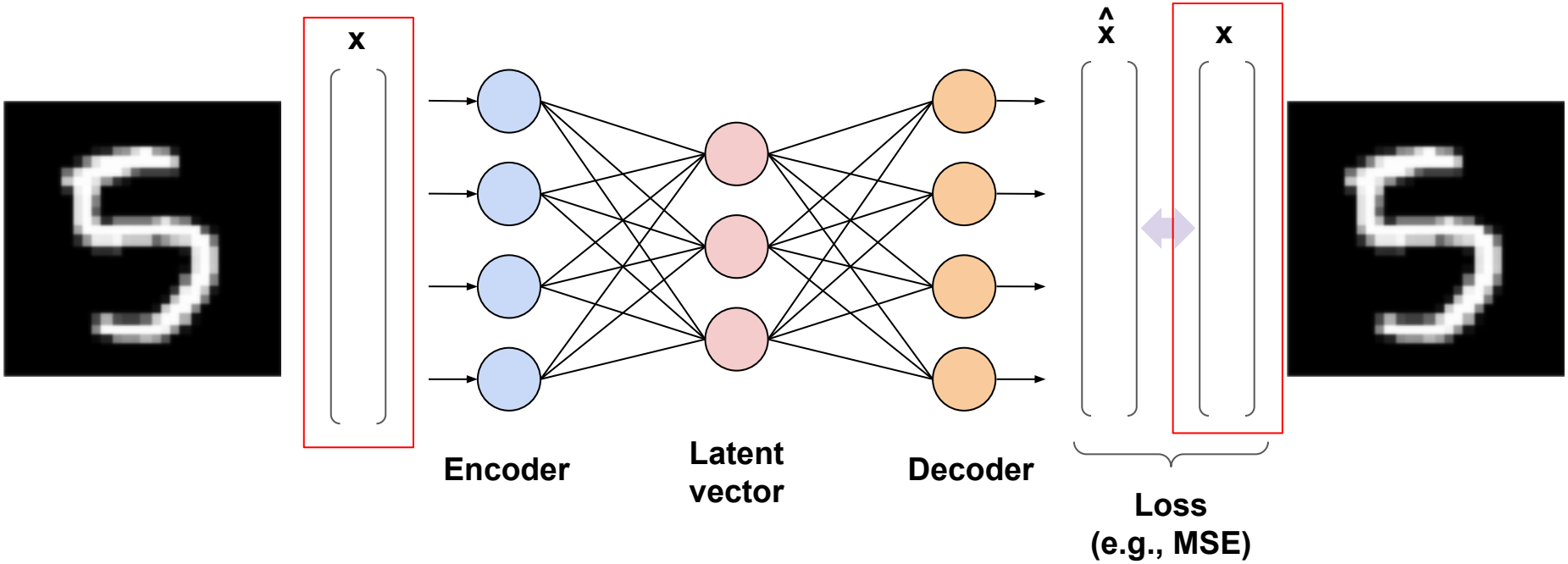
- Simply use input data as the target data



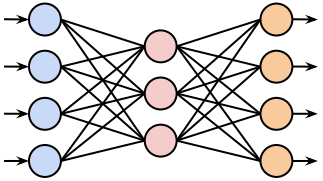
$$L(x, \hat{x}) = \|x - \hat{x}\|^2$$

Autoencoders: Training

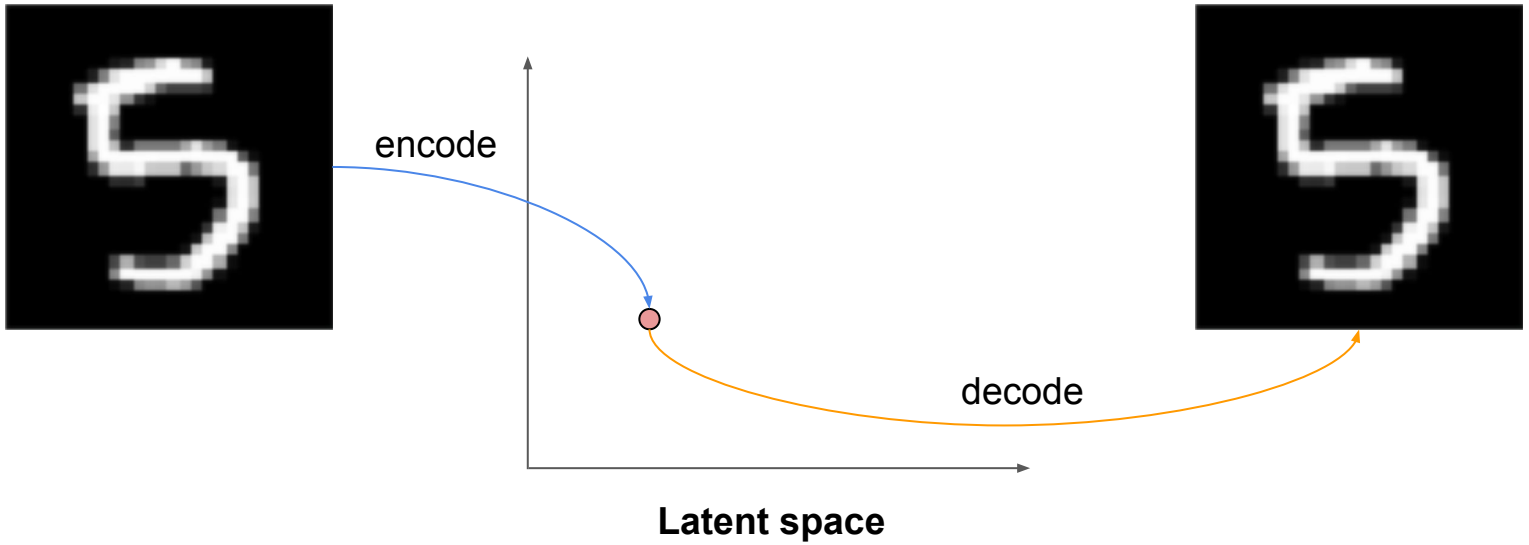
- Simply use input data as the target data



Autoencoders: Latent Space

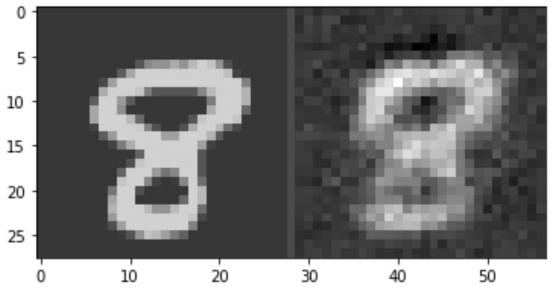


- The **encoder** first maps input data into latent vectors, which are then converted back to data by the decoder

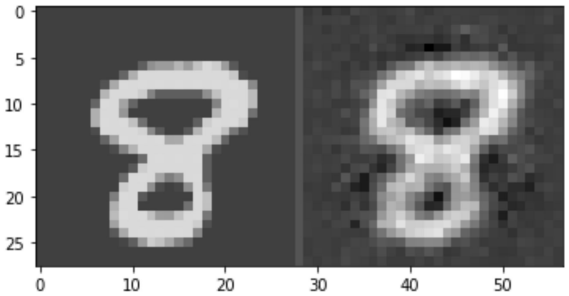


Self-reconstruction Examples

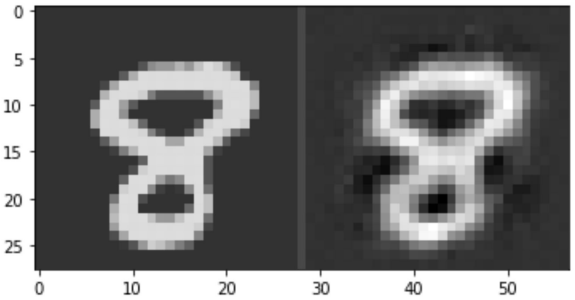
```
Epoch 0
    Training loss: 15.2723
  Validation loss: 9.5237
```




```
Epoch 4
    Training loss: 5.0810
  Validation loss: 4.9002
```

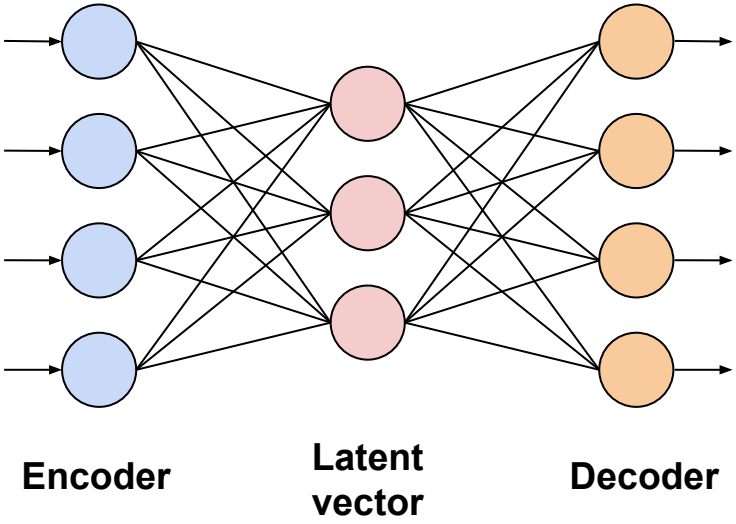



```
Epoch 9
    Training loss: 3.9312
  Validation loss: 3.8820
```



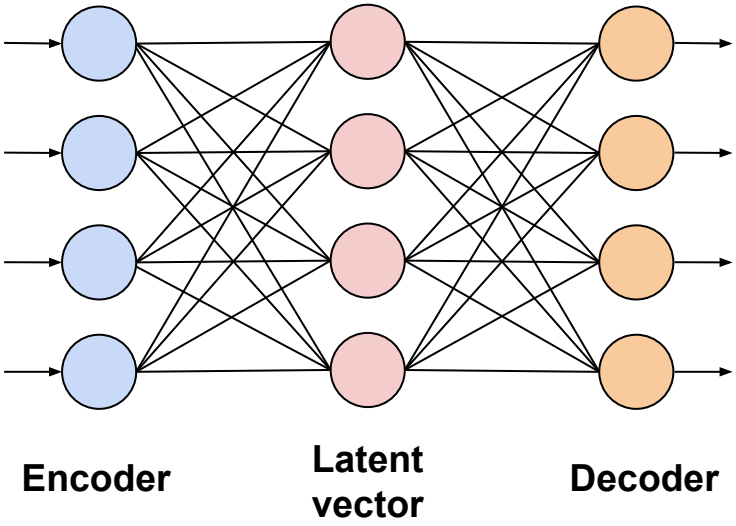
Autoencoders: Bottleneck Hidden Layer

- The latent space dimension **must be smaller** than that of input vector
- Why?



Autoencoders: Bottleneck Hidden Layer

- The latent space dimension **must be smaller** than that of input vector
- Why? → The model can always reconstruct the original data



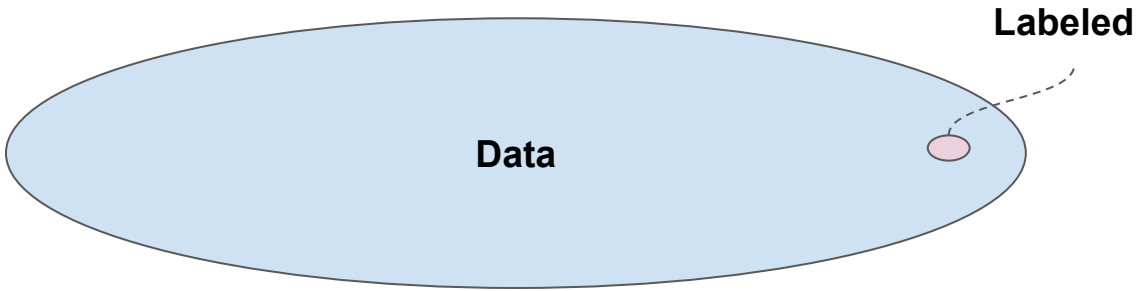
Unsupervised or Supervised?

- We do not need any label data for autoencoders (i.e., unsupervised learning)
  - But, the training procedure looks like supervised learning 🤔

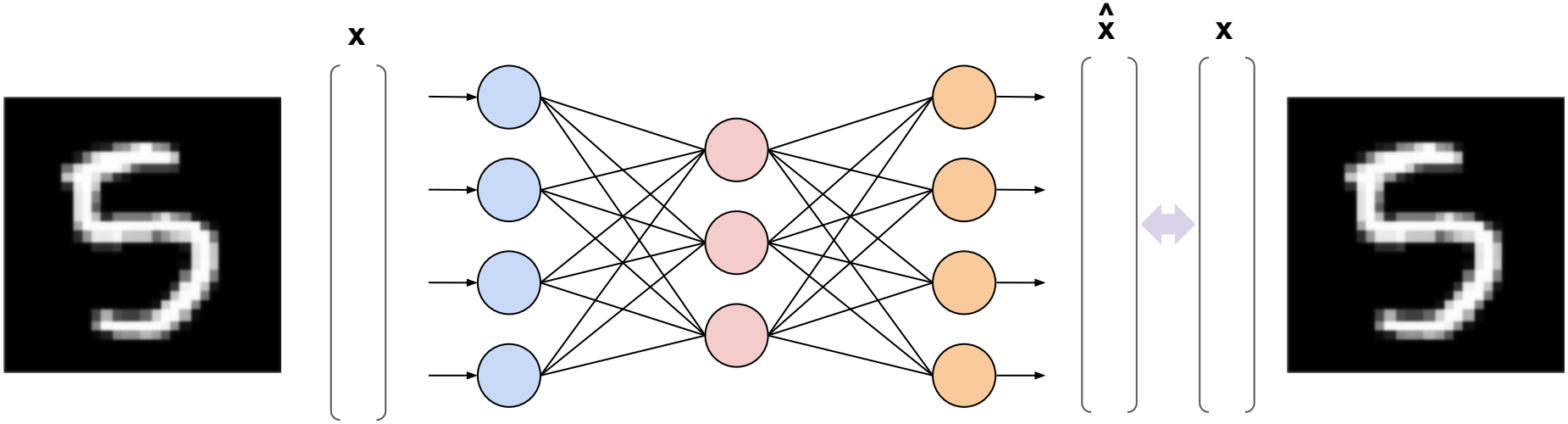
Unsupervised or Supervised?

- We do not need any label data for autoencoders (i.e., unsupervised learning)
  - But, the training procedure looks like supervised learning 🤔

- It's often called **self-supervised** learning
  - We can collect (pseudo) labels with no additional cost for self-supervised learning
  - Key approach to **pre-training** models

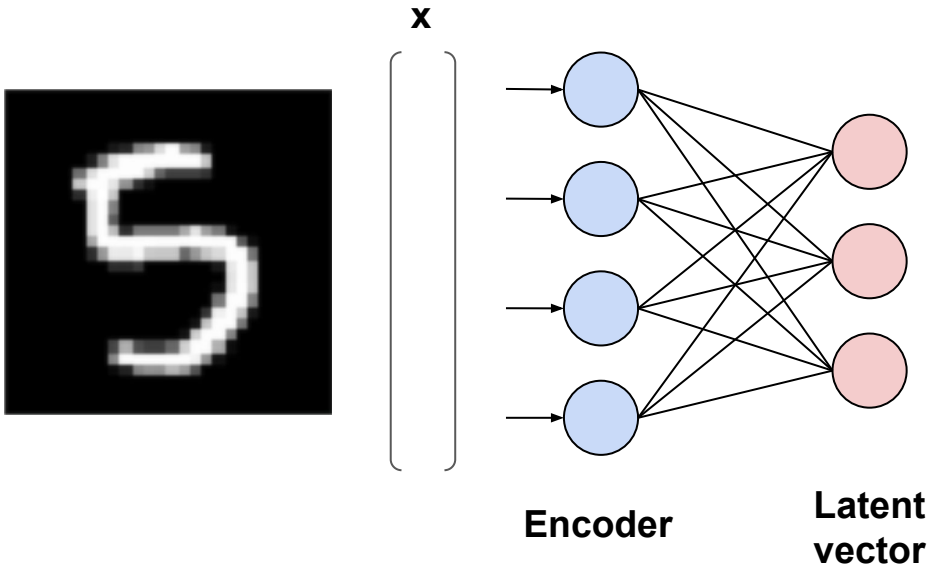


Autoencoder as Pre-training: (1) Pre-training

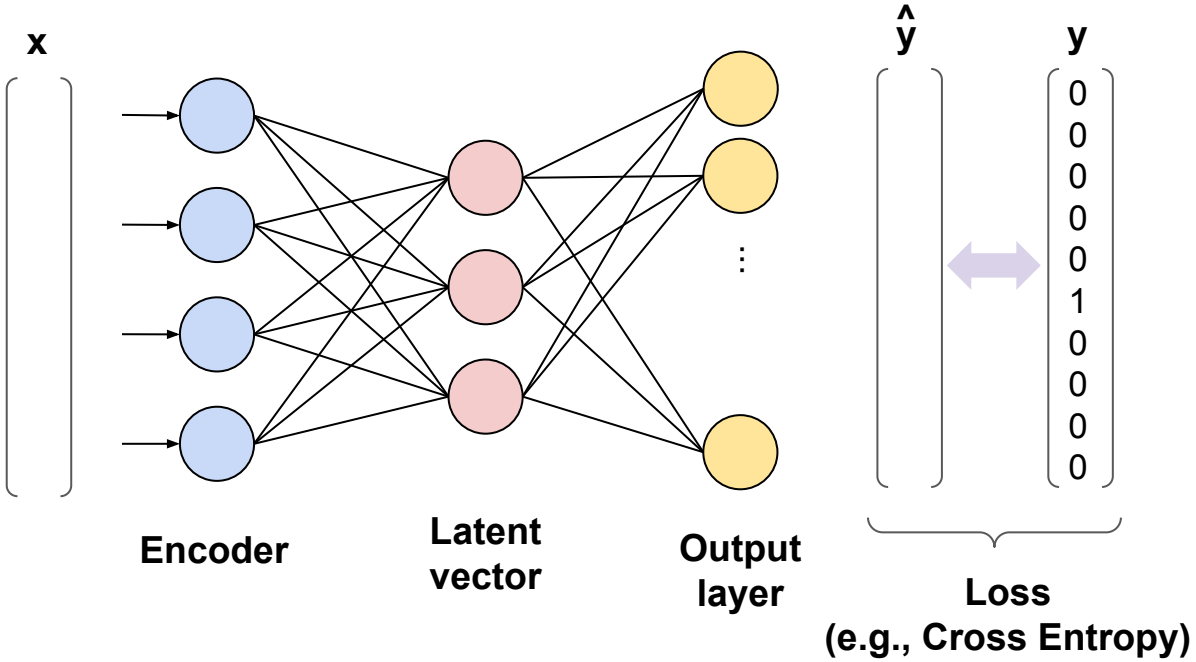


$$L(x, \hat{x}) = \|x - \hat{x}\|^2$$

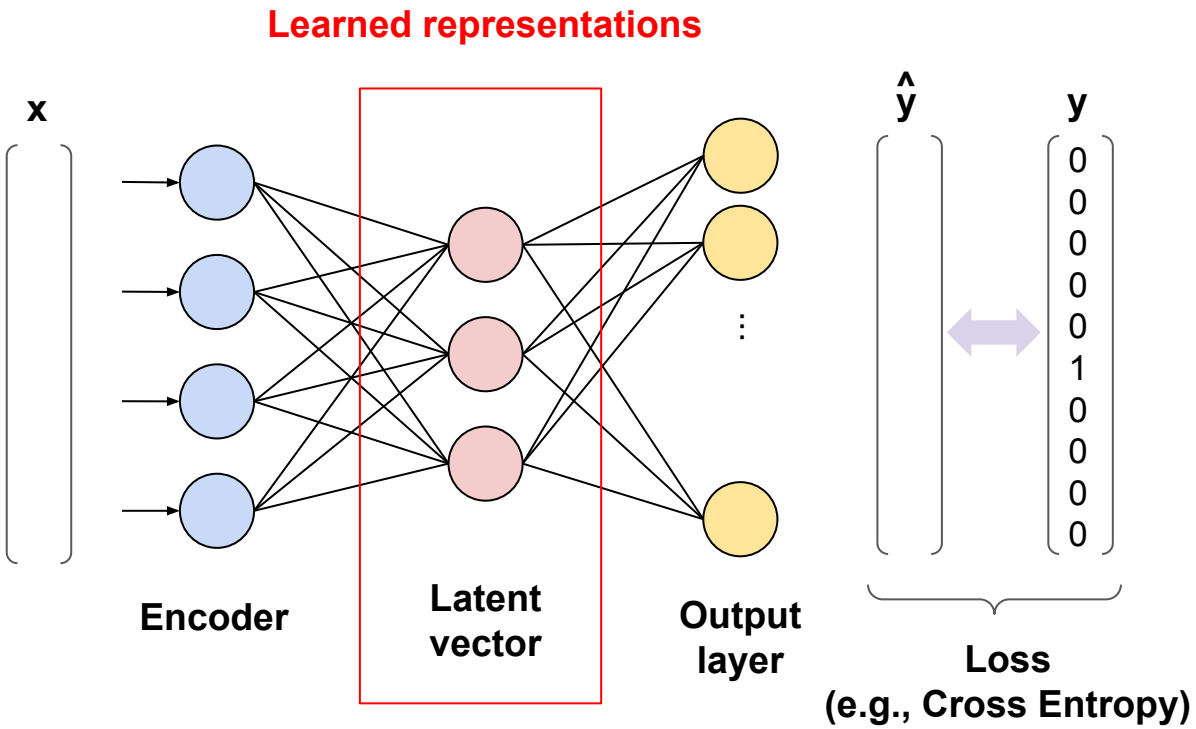
Autoencoder as Pre-training: (1) Pre-training



Autoencoder as Pre-training: (2) Fine-tuning

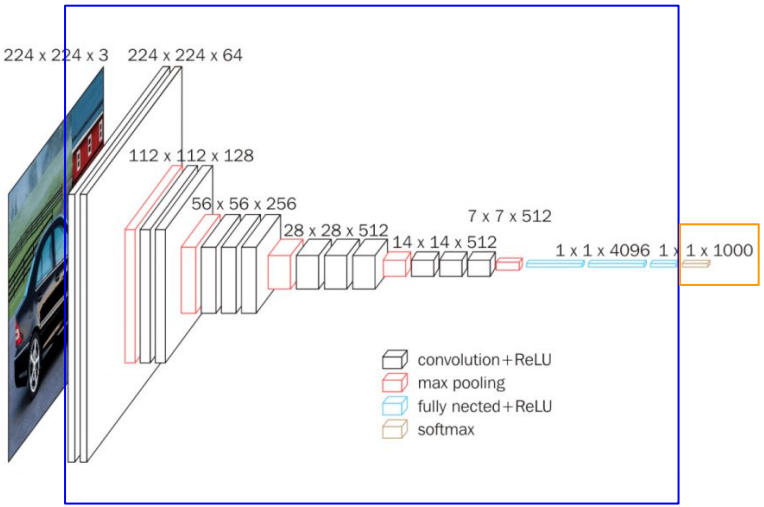


Autoencoder as Pre-training: (2) Fine-tuning

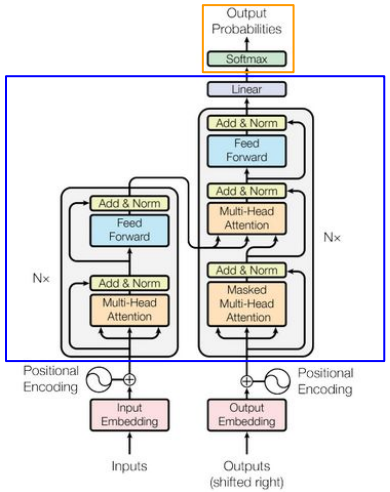


Neural Networks as Representation Learning

- Deep NN = **Representation extraction** + Linear/Logistic Regression



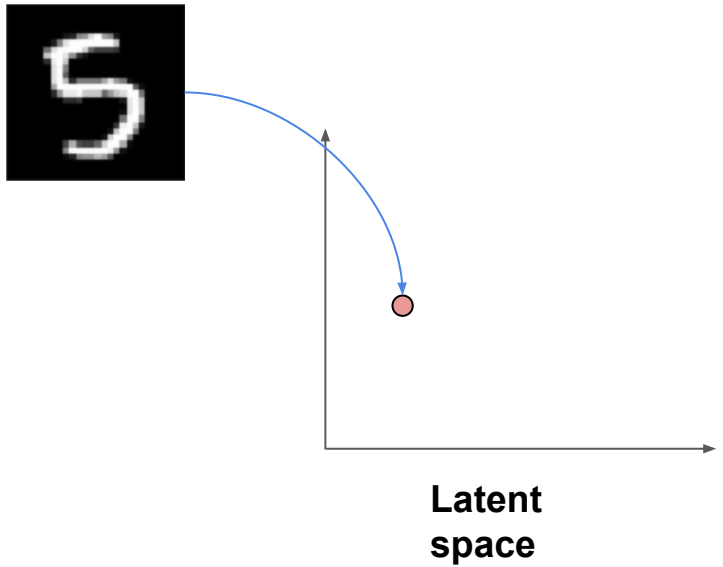
VGG 16



Transformer

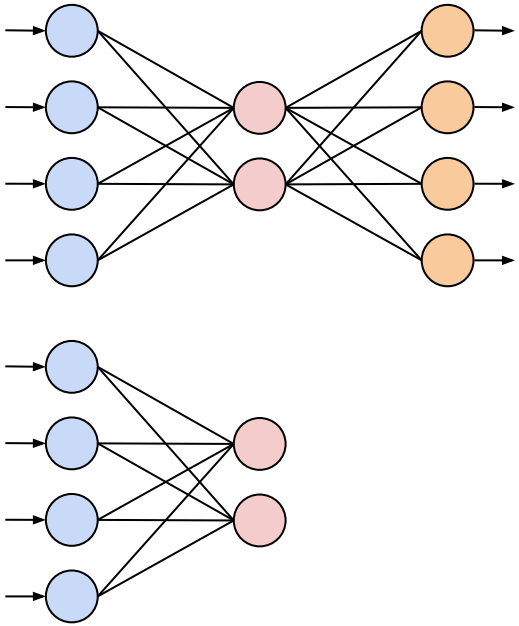
Autoencoders as Dimensionality Reduction

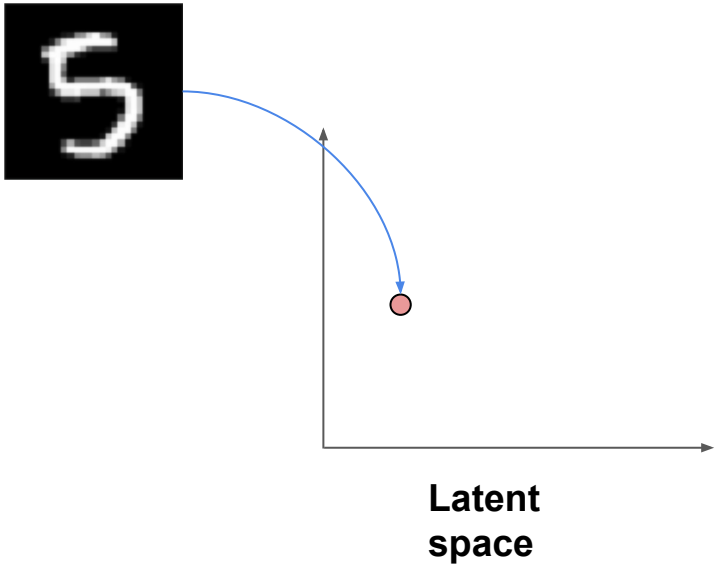
- We can use Autoencoder models for visualization (e.g., PCA, t-SNE)
- How?



Autoencoders as Dimensionality Reduction

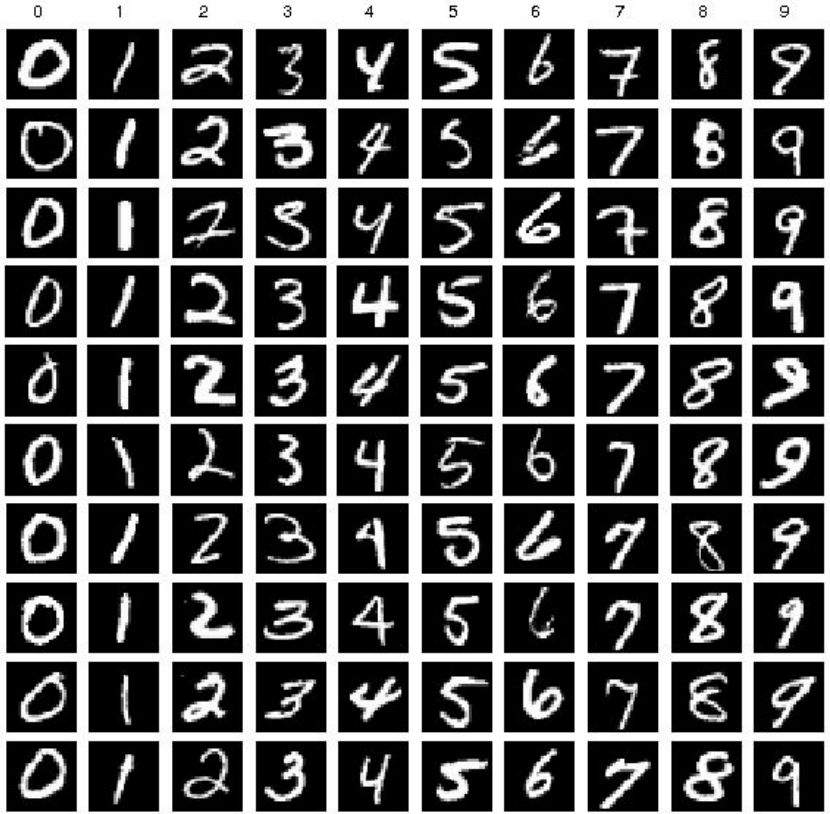
- We can use Autoencoder models for visualization (e.g., PCA, t-SNE)
- How? → Set the hidden size to be 2





MNIST Dataset

- A dataset of handwritten digits
  - The Iris Dataset for Neural Networks
- Dataset specification
  - 28x28 grayscale pixels
  - 60,000 training examples
  - 10,000 test examples
  - 10-class (0-9) labels



Hands-on Time: Let’s Implement Autoencoder

- Google Colab

Autoencoder: PyTorch Code Example

```
import torch.nn as nn

class AutoEncoder(nn.Module):
    def __init__(self,
                 hidden_dim: int = 64):
        super().__init__()
        self.encoder = nn.Sequential(nn.Linear(28 * 28, hidden_dim),
                                     nn.ReLU())
        self.decoder = nn.Sequential(nn.Linear(hidden_dim, 28 * 28))

    def encode(self, x):
        return self.encoder(x)

    def decode(self, x):
        return self.decoder(x)

    def forward(self, x):
        embedding = self.encoder(x)
        out = self.decoder(embedding)
        return out
```

Feature Normalization: Standardization

- StandardScaler (scikit-learn)
  - sklearn.preprocessing.StandardScaler — scikit-learn 0.24.2 documentation
- We are supposed to “fit” a scaler on training data and apply it to test data

$$z = \frac{x - \mu}{\sigma}$$

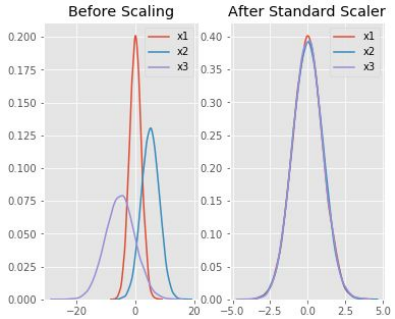
- Other options
  - sklearn.preprocessing.MinMaxScaler — scikit-learn 0.24.2 documentation

$$z = \frac{x - X_{min}}{X_{max} - X_{min}}$$

What’s the issue with it?

MinMaxScaler: Risks

- 1) Sensitive to the min/max values
  - Likely to further compress “crowded” regions
- 2) Examples from different datasets can look different (→ important for **Transfer learning**)
  - StandardScaler fits data into N(0, 1) distribution

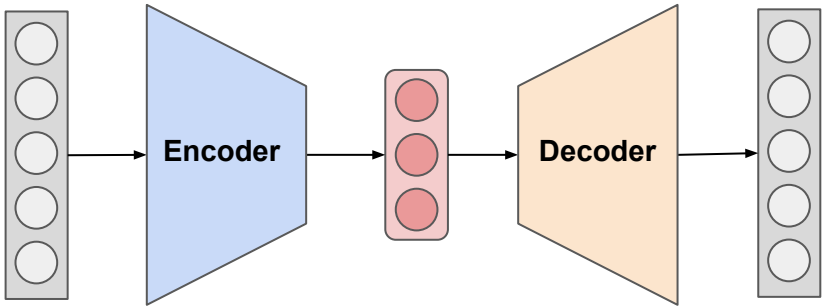


cf. **Robust statistics** (e.g., calculating statistics value after removing 25% quantiles etc.) RobustScaler at scikit-learn. Generally speaking, StandardScaler should be good enough

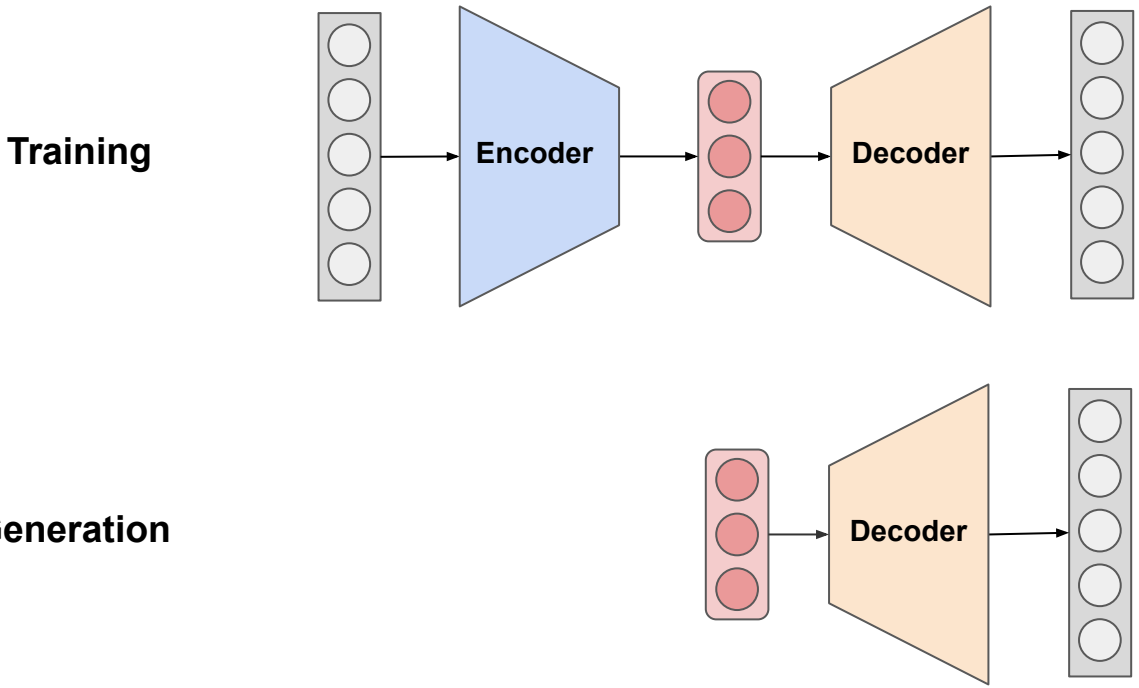
Further reading: Compare the effect of different scalers on data with outliers — scikit-learn 0.24.2 documentation

Autoencoder in Encoder-Decoder Presentation

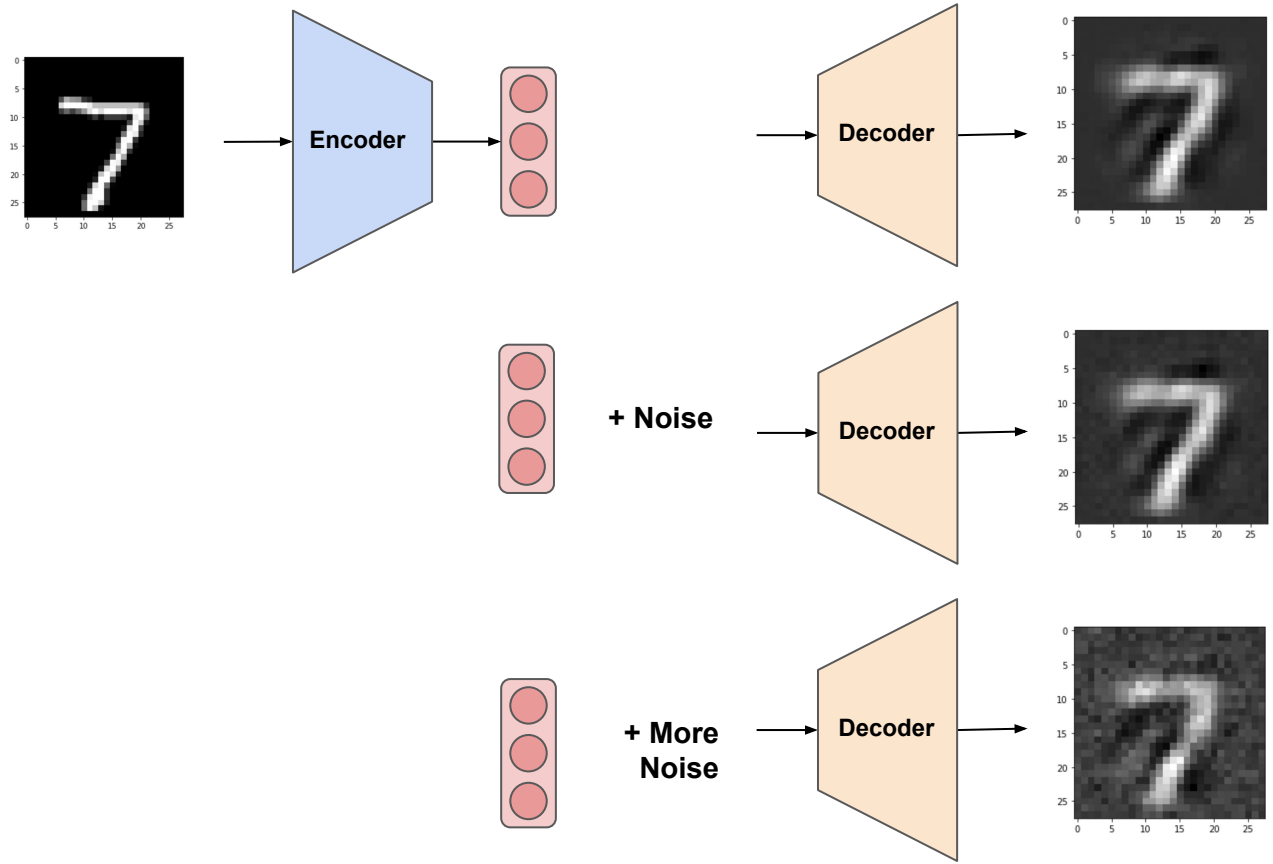
- Encoder: Input values → Latent vectors
- Decoder: Latent vectors → Output values



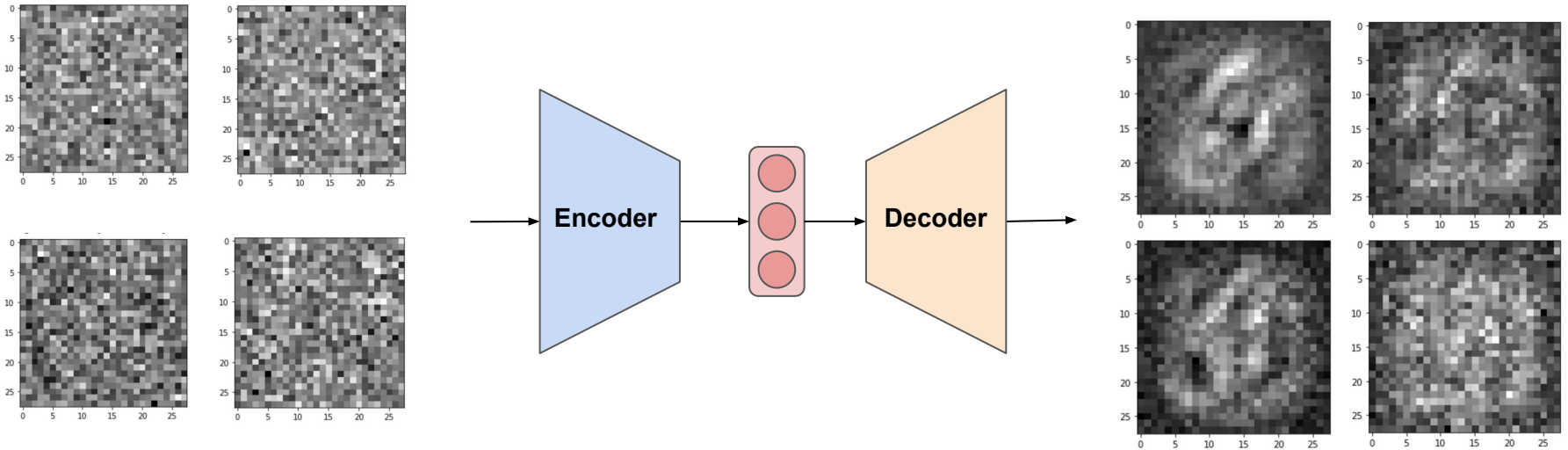
Generating Handwritten Images



Generating Images from Latent Vector + Noise



Generating Images from Random Noise



We will discuss more advanced solutions for image generation next week (in the GAN session)

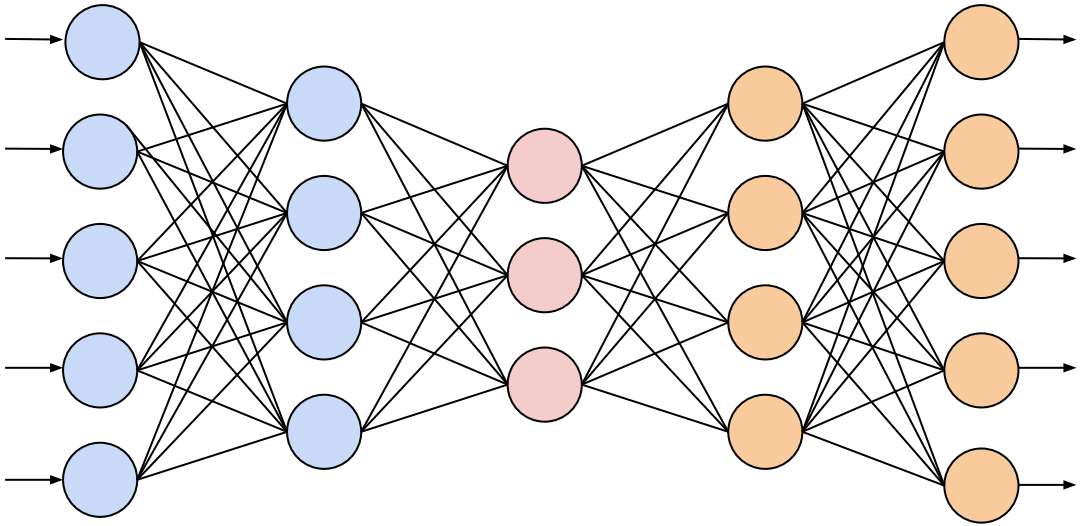
Advanced Autoencoders

Advanced Autoencoder Models

- Stacked Autoencoder
- Denoising Autoencoder
- Variational Autoencoder (VAE)

Stacked Autoencoder

- Encoder/Decoder with multiple layers



Autoencoder → Stacked Autoencoder

- Where to edit?

```
import torch.nn as nn

class AutoEncoder(nn.Module):
    def __init__(self,
                 hidden_dim: int = 64):
        super().__init__()
        self.encoder = nn.Sequential(nn.Linear(28 * 28, hidden_dim),
                                     nn.ReLU())
        self.decoder = nn.Sequential(nn.Linear(hidden_dim, 28 * 28))

    def encode(self, x):
        return self.encoder(x)

    def decode(self, x):
        return self.decoder(x)

    def forward(self, x):
        embedding = self.encoder(x)
        out = self.decoder(embedding)
        return out
```

Autoencoder → Stacked Autoencoder

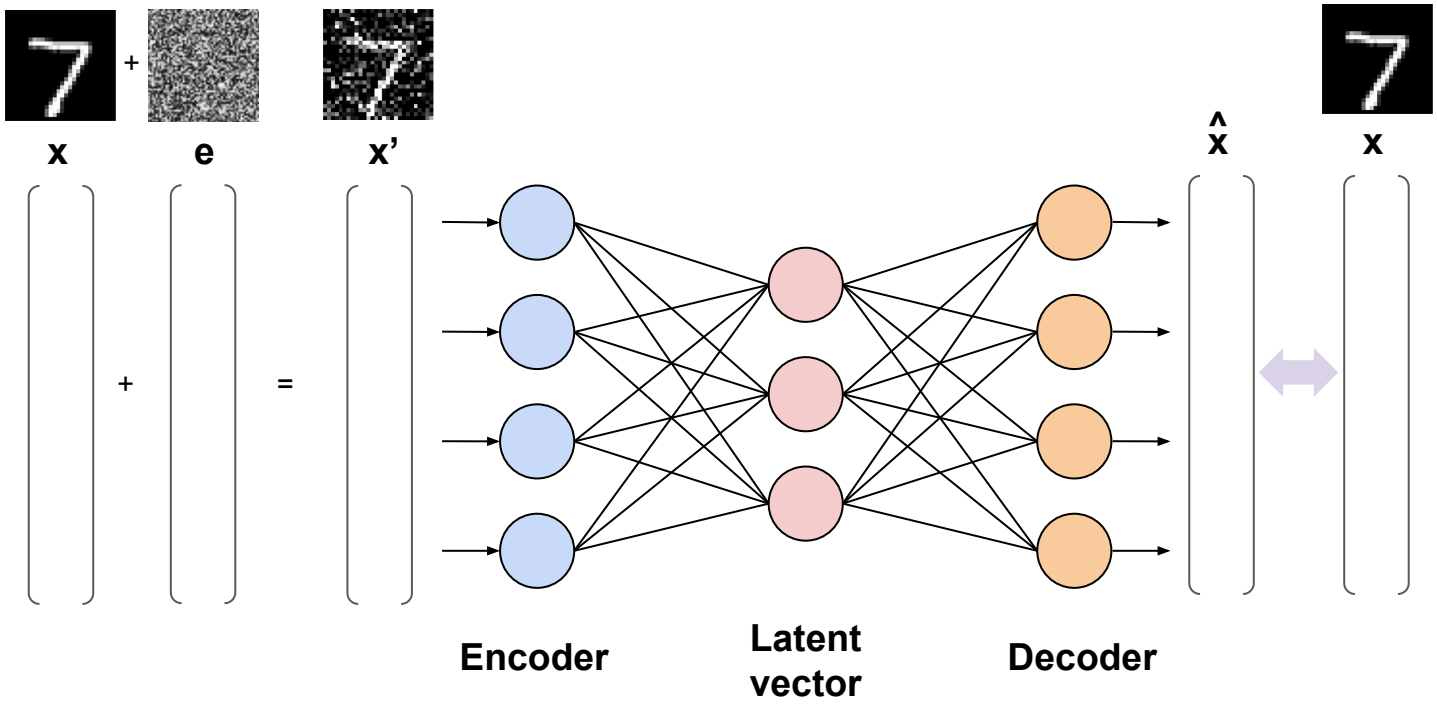
- Where to edit?

```python
import torch.nn as nn

class AutoEncoder(nn.Module):
    def __init__(self,
                 hidden_dim: int = 64):
        super().__init__()
        self.encoder = nn.Sequential(nn.Linear(28 * 28, hidden_dim),
                                     nn.ReLU())
        self.decoder = nn.Sequential(nn.Linear(hidden_dim, 28 * 28))

    def encode(self, x):
```

```python
self.encoder = nn.Sequential(nn.Linear(28 * 28, hidden_dim),
                             nn.ReLU(),
                             nn.Linear(hidden_dim, 3),
                             nn.ReLU())
self.decoder = nn.Sequential(nn.Linear(3, hidden_dim),
                             nn.ReLU(),
                             nn.Linear(hidden_dim, 28 * 28))
```

Denoising Autoencoder

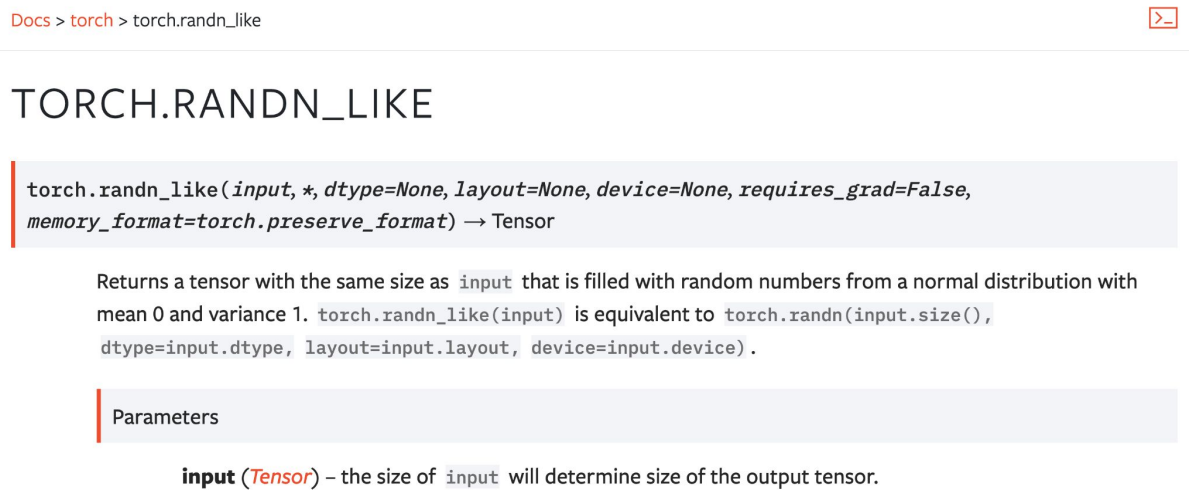
- Learn to reconstruct the original data from **corrupted input**



Hands-on Time: Implement Denoising Autoencoder

- Google Colab

Hint



Docs > torch > torch.randn_like

TORCH.RANDN_LIKE

torch.randn_like(*input*, ***, *dtype=None*, *layout=None*, *device=None*, *requires_grad=False*, *memory_format=torch.preserve_format*) → Tensor

Returns a tensor with the same size as `input` that is filled with random numbers from a normal distribution with mean 0 and variance 1. `torch.randn_like(input)` is equivalent to `torch.randn(input.size(), dtype=input.dtype, layout=input.layout, device=input.device)`.

Parameters

input (*Tensor*) – the size of `input` will determine size of the output tensor.

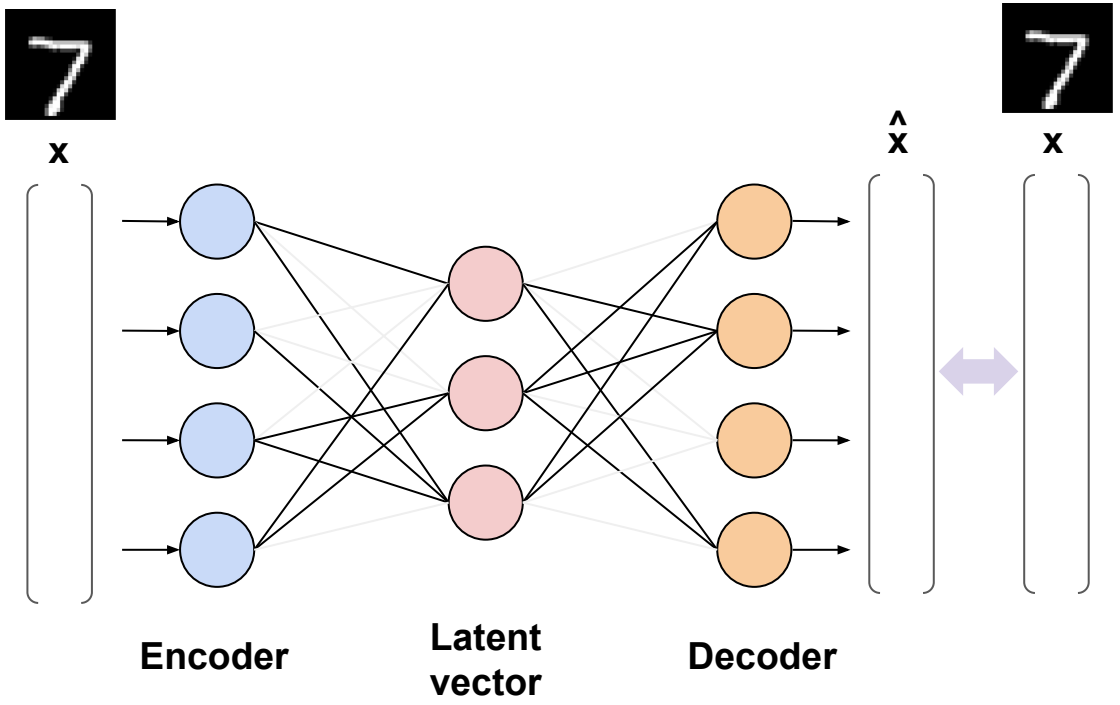
https://pytorch.org/docs/stable/generated/torch.randn_like.html

Denoising Autoencoder: PyTorch Code Example

```
class DenoisingAutoEncoder(nn.Module):
    def __init__(self,
                 hidden_dim: int = 64,
                 noise_factor: float = 0.01):
        super().__init__()
        self.noise_factor = noise_factor
        self.encoder = nn.Sequential(nn.Linear(28 * 28, hidden_dim),
                                     nn.ReLU())
        self.decoder = nn.Sequential(nn.Linear(hidden_dim, 28 * 28))

    def encode(self, x):
        return self.encoder(x)

    def decode(self, x):
        return self.decoder(x)

    def forward(self, x):

        if self.training: # True if model.train(); False if model.eval()
           # Add random noise for training
           x += torch.randn_like(x) * self.noise_factor
        embedding = self.encoder(x)
        out = self.decoder(embedding)
        return out
```

Regularization technique: Dropout

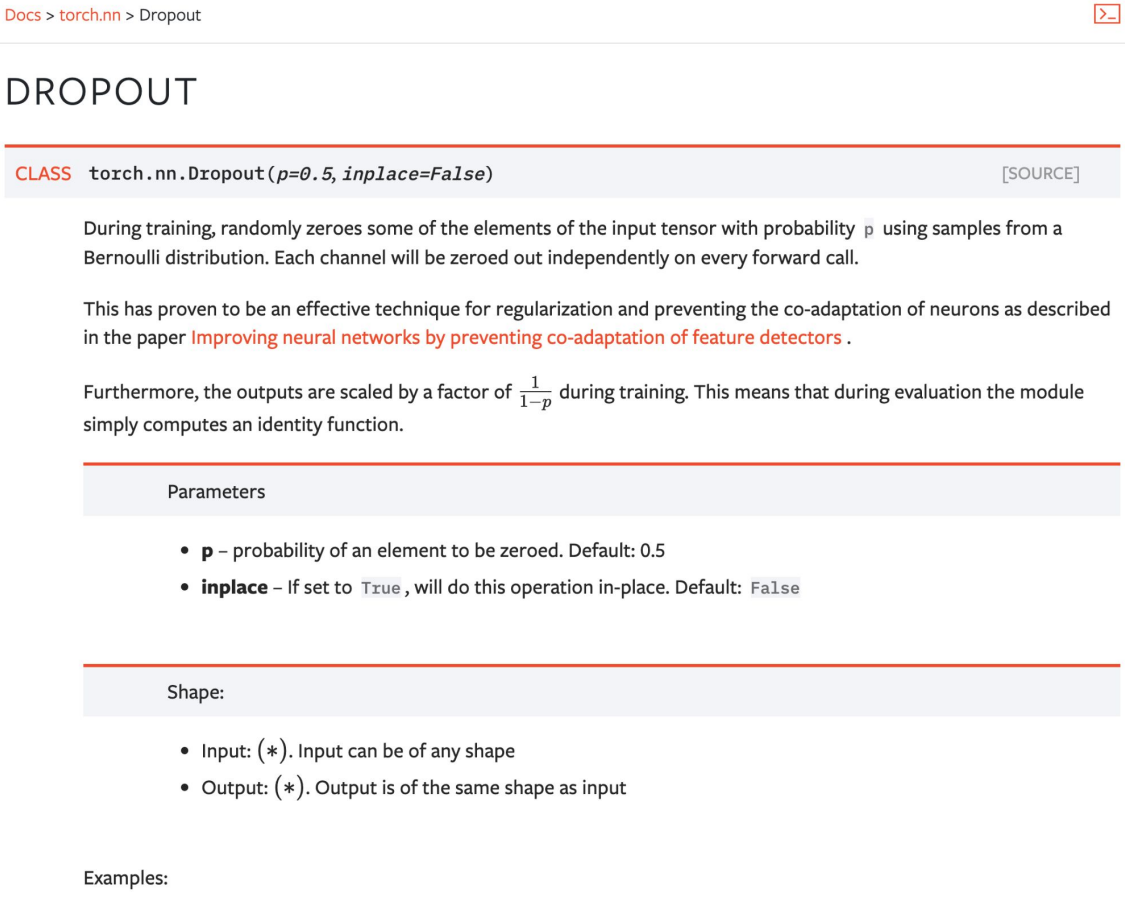
- “Turn off” randomly selected edges in each training step
- The full network is used for inference (→ model.train() & model.eval())



Regularization technique: Dropout

- “Turn off
- The full

eval()



Docs > torch.nn > Dropout

DROPOUT

CLASS `torch.nn.Dropout(p=0.5, inplace=False)` [SOURCE]

During training, randomly zeroes some of the elements of the input tensor with probability `p` using samples from a Bernoulli distribution. Each channel will be zeroed out independently on every forward call.

This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons as described in the paper Improving neural networks by preventing co-adaptation of feature detectors .

Furthermore, the outputs are scaled by a factor of $\frac{1}{1-p}$ during training. This means that during evaluation the module simply computes an identity function.

Parameters

- **p** – probability of an element to be zeroed. Default: 0.5
- **inplace** – If set to `True`, will do this operation in-place. Default: `False`

Shape:

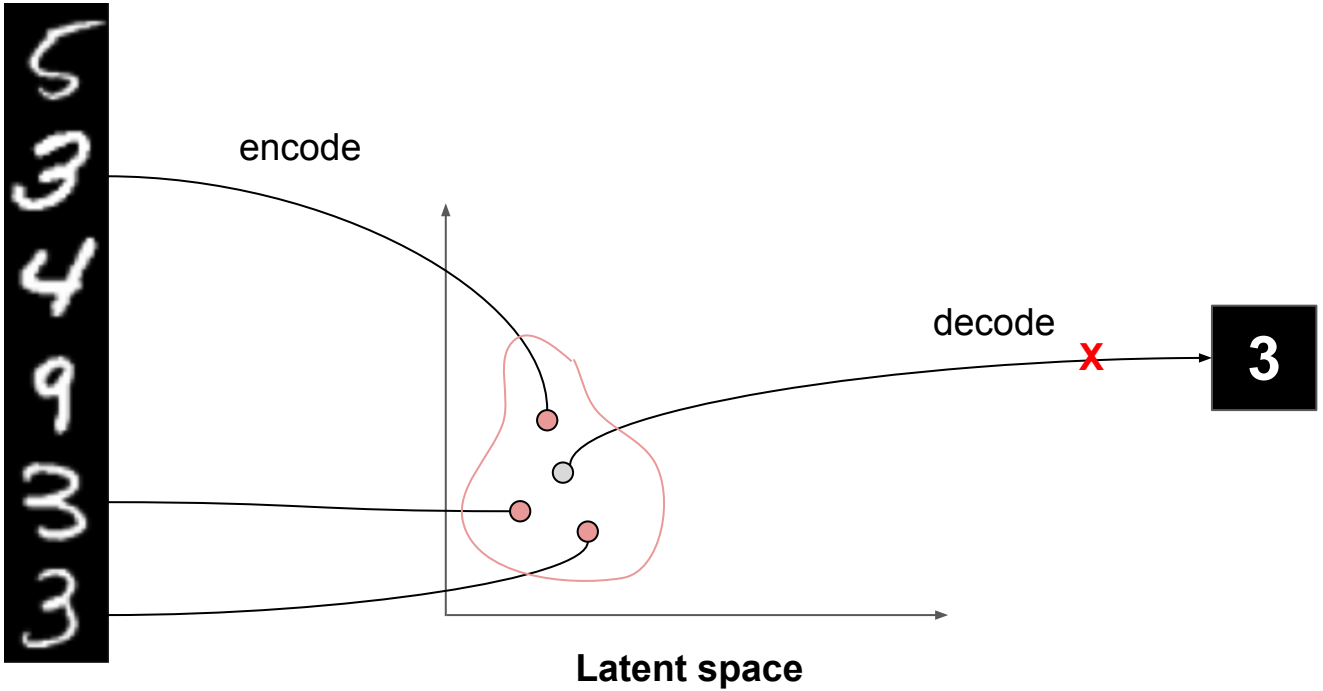
- Input: $(*)$. Input can be of any shape
- Output: $(*)$. Output is of the same shape as input

Examples:

Variational Autoencoder (VAE)

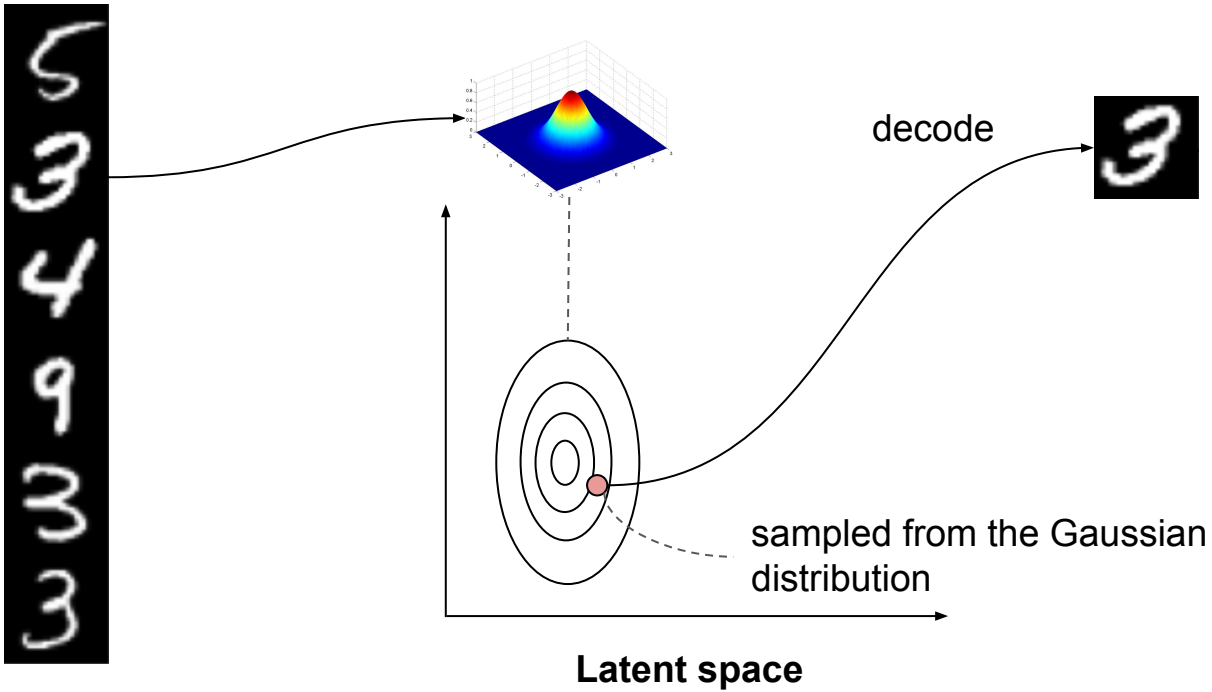
Why VAE? Latent Space with Autoencoder

- AE needs many examples to learn the **“smooth”** latent space
- A latent vector close to encoded examples may not be decoded into a meaningful image



Modeling Distribution in Latent Space

- Sampling latent vectors from Gaussian distributions
- Input data → Gaussian distributions → Latent vectors → Output data



Recap: Gaussian Distribution Parameters

- What are the parameters?

Recap: Gaussian Distribution Parameters

- What are the parameters?
- Mean: μ
- Variance: σ^2

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Recap: Gaussian Distribution Parameters

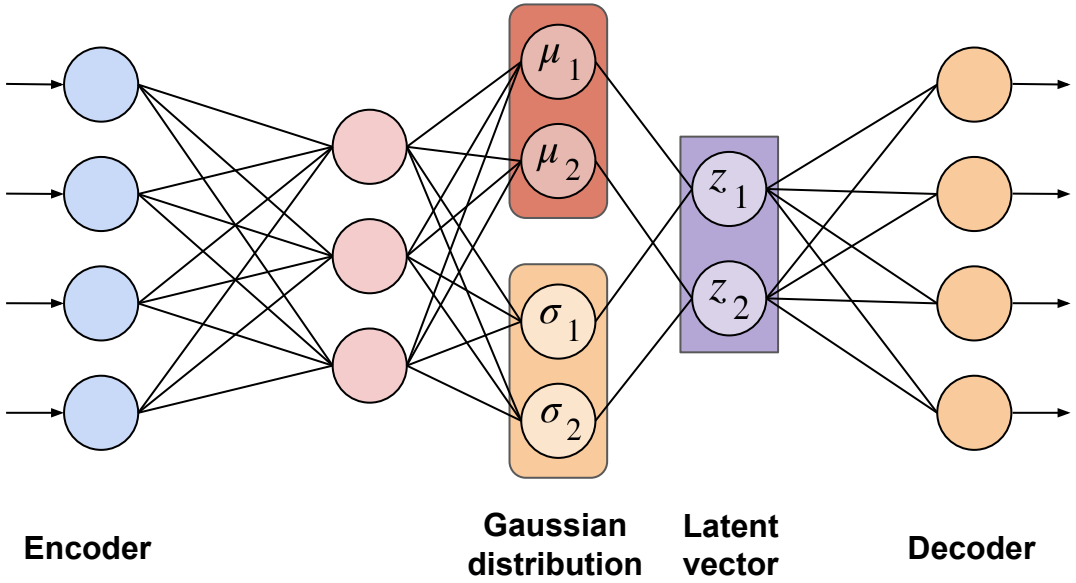
- What are the parameters?
- Mean: μ
- Variance: σ^2

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Can we actually calculate **μ** and **σ** and then sample latent vectors from the distribution?

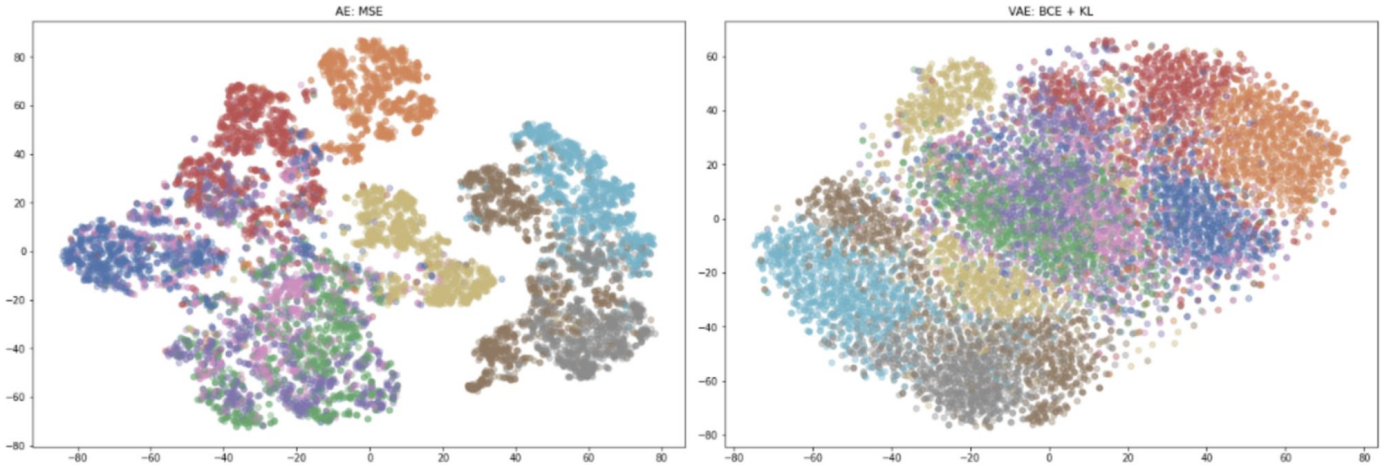
Variational Autoencoder (VAE)

- Modeling Gaussian distributions with neural networks



AE vs VAE: Latent Space Visualization

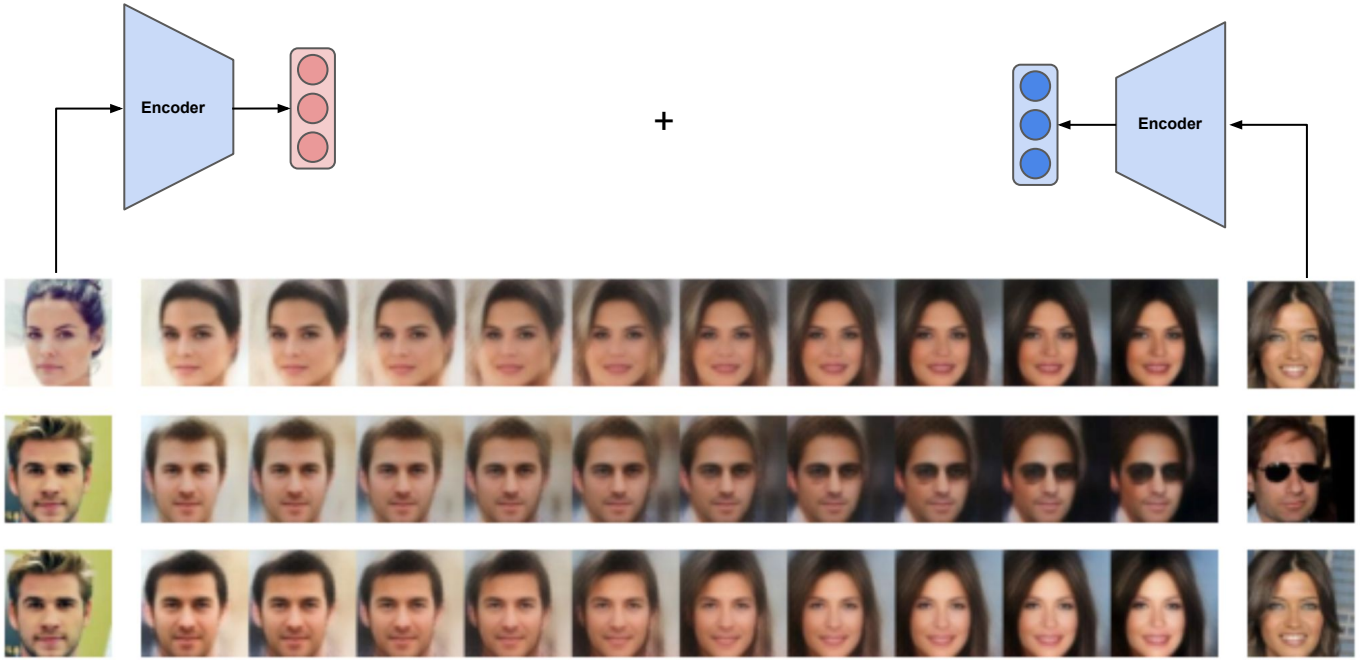
- VAE shows “smooth” latent space



(a) AE: MSE (b) VAE: BCE + KL

https://medium.com/@tu_53768/generative-models-vaes-aes-4a20fa3dcbf8

Example: Facial Image Interpolation



https://arxiv.org/pdf/1610.00291.pdf

Note: Local context looks more essential

- Fully-connected architecture may be overkill :(
- → **Convolutional layers** to capture image patterns (Thursday session)

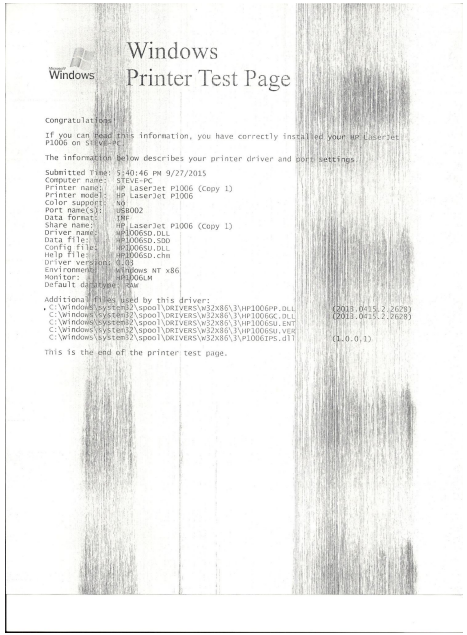
Real-world problems

Problems that can be solved by Autoencoders

- Any ideas?

Problems that can be solved by Autoencoders

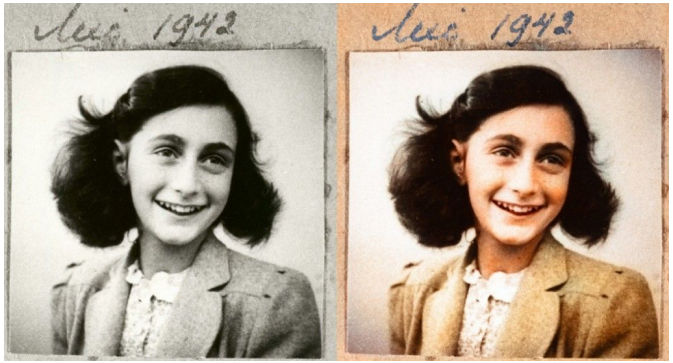
- Image restoration problems in general

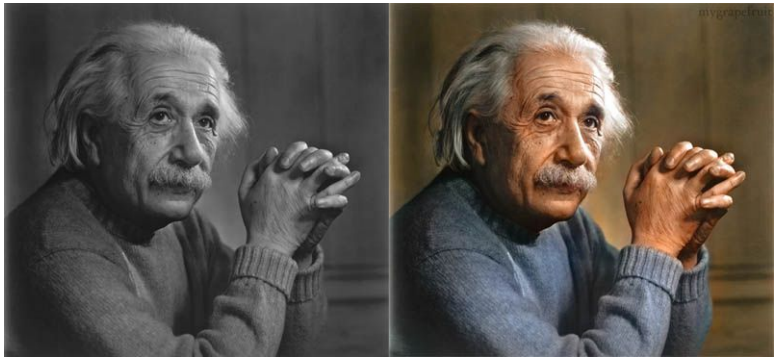


How do you train Autoencoder models for the problems?

Advanced Applications

- Picture colorization can be solved by Autoencoders! How?

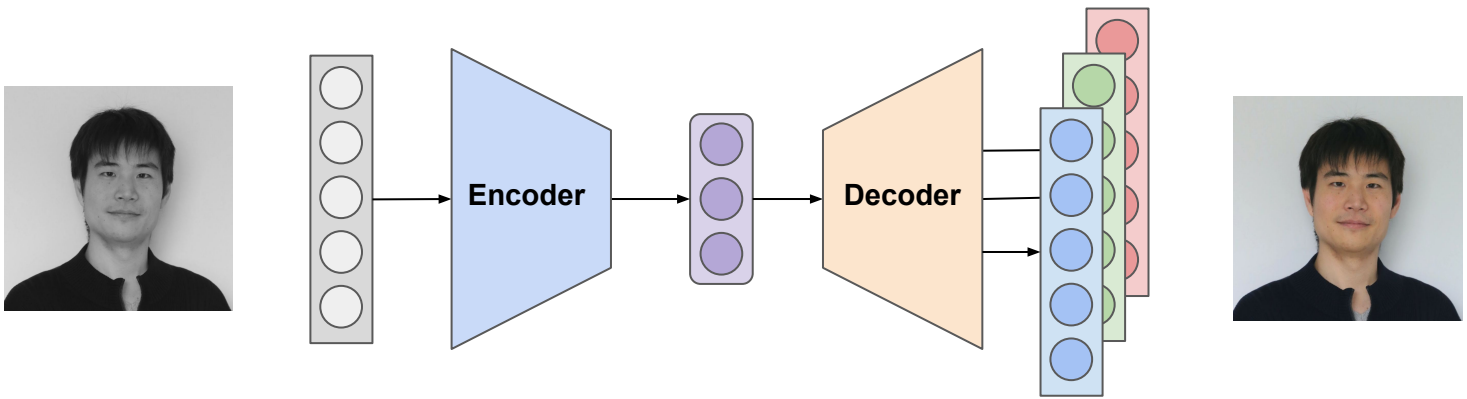




Creating Parallel Dataset: RGB ⇔ Grayscale

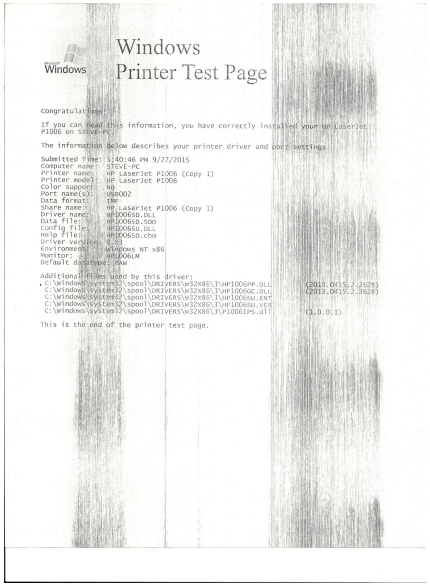
- Why don’t we create a parallel dataset for Grayscale ⇔ Color?
- A number of options. For example,

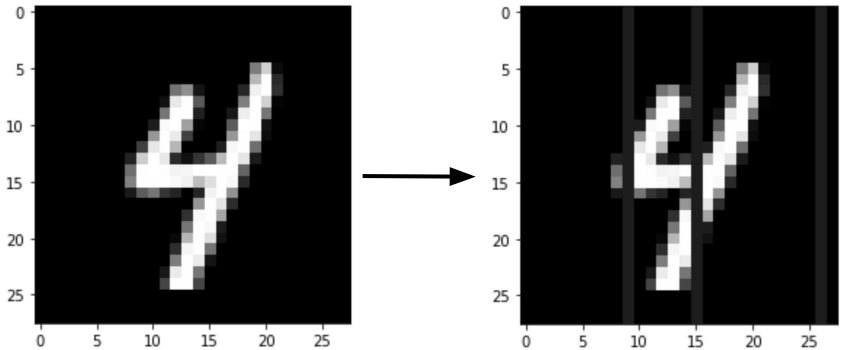
$$\text{Grayscale} = (R + G + B) / 3$$



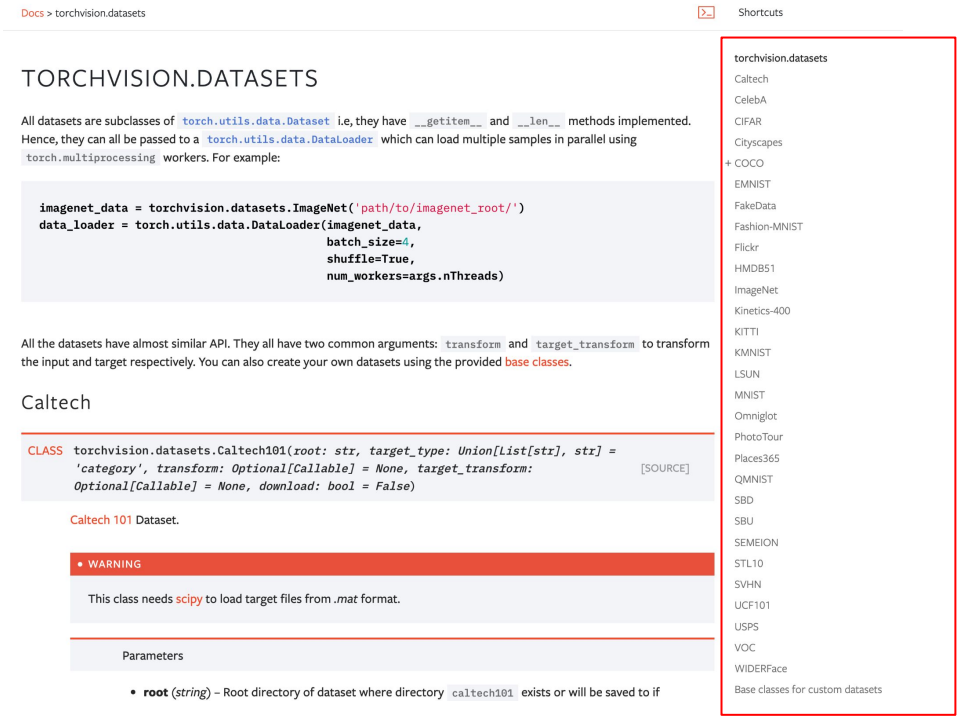
Advanced Exercise Idea: Image Restoration Task

- Adding corruptions to create an image restoration dataset





Torchvision Datasets
https://pytorch.org/vision/stable/datasets.html



Docs > torchvision.datasets

Shortcuts

TORCHVISION.DATASETS

All datasets are subclasses of `torch.utils.data.Dataset` i.e, they have `__getitem__` and `__len__` methods implemented. Hence, they can all be passed to a `torch.utils.data.DataLoader` which can load multiple samples in parallel using `torch.multiprocessing` workers. For example:

```
imagenet_data = torchvision.datasets.ImageNet('path/to/imagenet_root/')
data_loader = torch.utils.data.DataLoader(imagenet_data,
                                          batch_size=4,
                                          shuffle=True,
                                          num_workers=args.nThreads)
```

All the datasets have almost similar API. They all have two common arguments: `transform` and `target_transform` to transform the input and target respectively. You can also create your own datasets using the provided base classes.

Caltech

CLASS torchvision.datasets.Caltech101(*root: str, target_type: Union[List[str], str] = 'category', transform: Optional[Callable] = None, target_transform: Optional[Callable] = None, download: bool = False*) [SOURCE]

Caltech 101 Dataset.

• WARNING

This class needs scipy to load target files from *.mat* format.

Parameters

- **root** (*string*) – Root directory of dataset where directory `caltech101` exists or will be saved to if

torchvision.datasets
Caltech
CelebA
CIFAR
Cityscapes
+ COCO
EMNIST
FakeData
Fashion-MNIST
Flickr
HMDB51
ImageNet
Kinetics-400
KITTI
KMNIST
LSUN
MNIST
Omniglot
PhotoTour
Places365
QMNIST
SBD
SBU
SEMEION
STL10
SVHN
UCF101
USPS
VOC
WIDERFace
Base classes for custom datasets

Summary

- Autoencoders Principles
- Advanced autoencoder models
  - Stacked Autoencoder
  - Denoising Autoencoder
    - Another regularization technique: Dropout
  - Variational Autoencoder
- Applications
  - Image restoration
  - Picture colorization