Research and Innovation Center of Excellence

MSc on Intelligent Critical Infrastructure Systems

Machine Learning
Lecture 10

Christos Kyrkou
Research Lecturer
KIOS Research and Innovation Center of Excellence
University of Cyprus

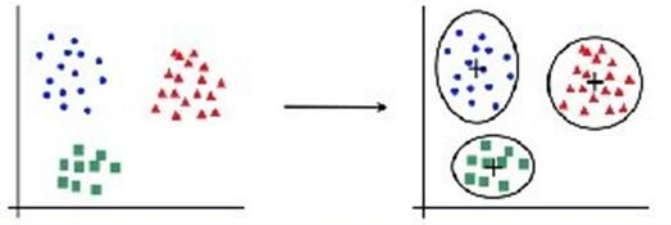
Imperial College London

University of Cyprus

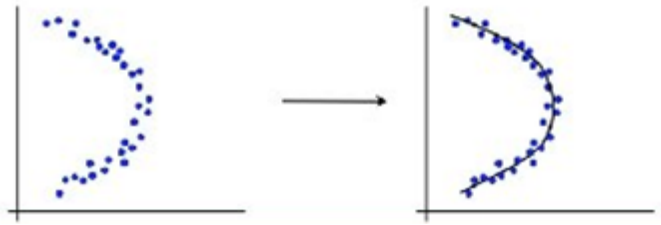
funded by:

Previously

- Unsupervised Learning
  - No labels just the data
  - Grouping-Clustering
    - K-means

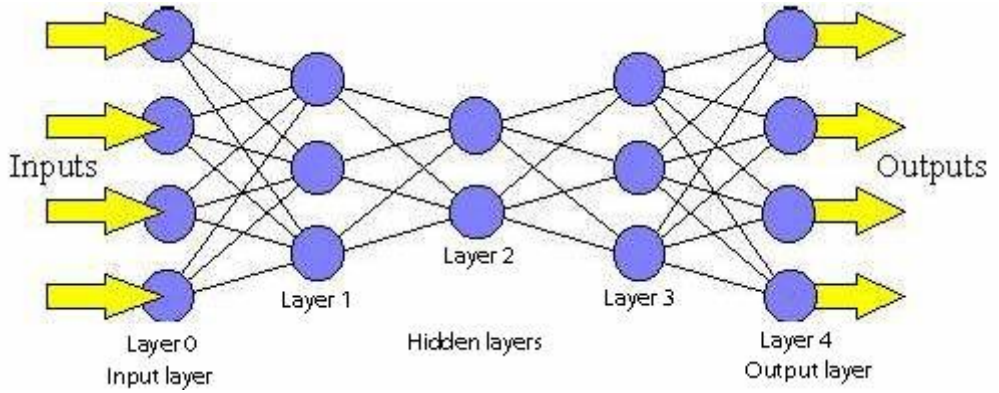


  - Dimensionality reduction
    - PCA



Deep Unsupervised Learning

- Nonlinear dimensional reduction and pattern matching.
- In many settings, have more unlabeled examples than labeled.
- Learn useful representations from unlabeled data.
- Better representation may improve prediction accuracy.



Latent variable models

- What is a latent variable?
  - A variable that we cannot observe directly.
  - True explanatory factors for the representation of the data
  - Easier to process and is a compressed representation of the data.

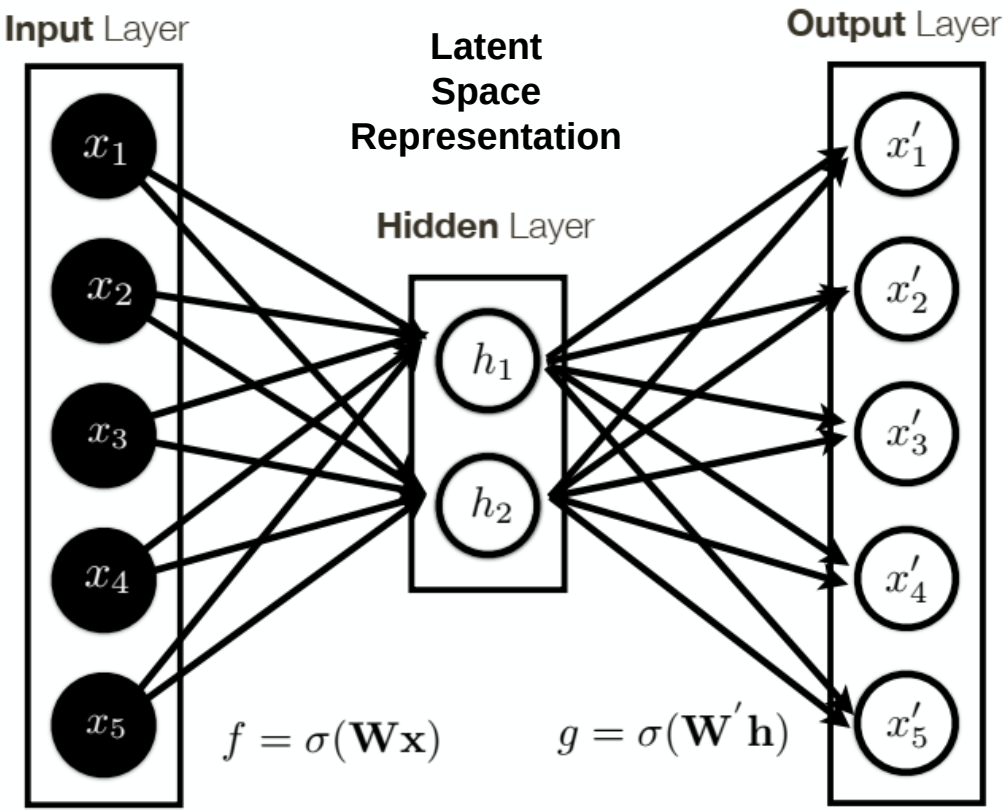
- Can we learn the true **explanatory factors**, e.g., latent variables, from only observed data?

Autoencoders

- Autoencoders are an unsupervised learning technique in which we leverage **neural networks** for the task of **representation learning**.
- Specifically, we'll design a neural network architecture such that we **impose a bottleneck** in the network which forces a **compressed knowledge representation** of the original input.
- If some sort of structure exists in the data (i.e., correlations between input features), **this structure can be learned** and consequently leveraged when forcing the input through the network's bottleneck.

Autoencoders

- Autoencoders are designed to reproduce their input
- Key point is to reproduce the input from a learned encoding.
- Feed forward network intended to reproduce the input



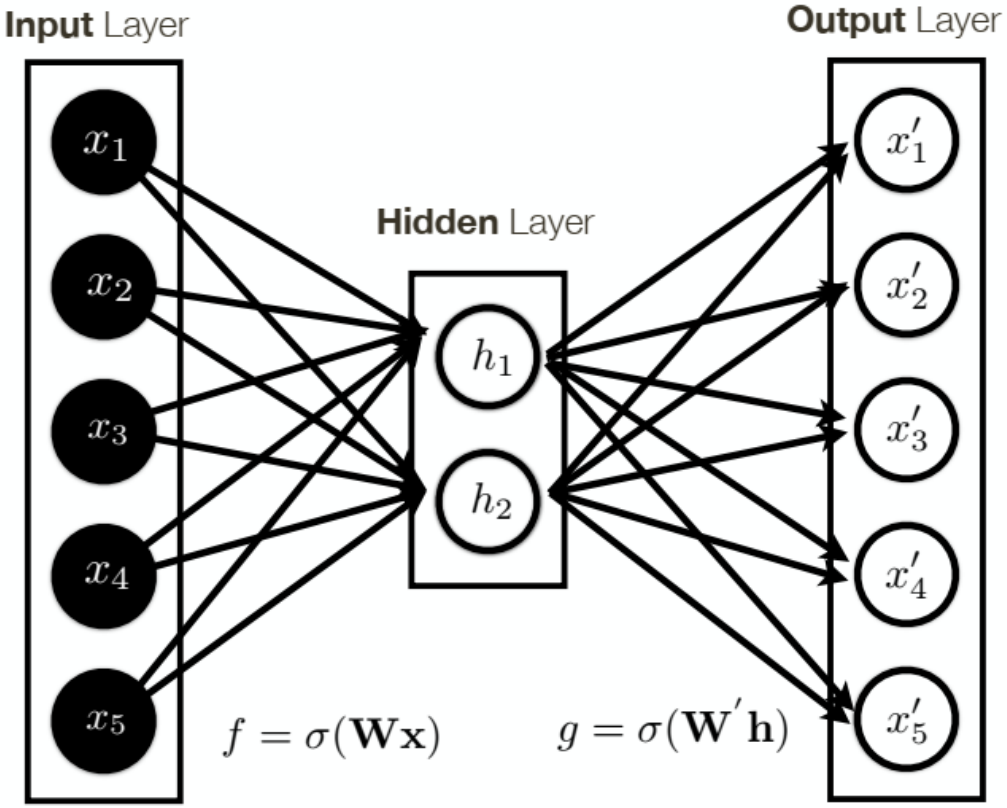
https://www.edureka.co/blog/autoencoders-tutorial/

Autoencoders

- A standard neural network architecture (linear layer followed by non-linearity).
- In the simplest case:
  - A linear layer with activation: $\mathrm{f} = \sigma(Wx)$
  - Another linear layer with activation : $g = \sigma(W'h)$

Autoencoders

- Activation depends on type of data
  - (e.g., sigmoid for binary, linear for real valued)
- Loss function needs to capture the task
- Using all linear layers with Euclidean loss is equivalent to PCA (under certain data normalization)

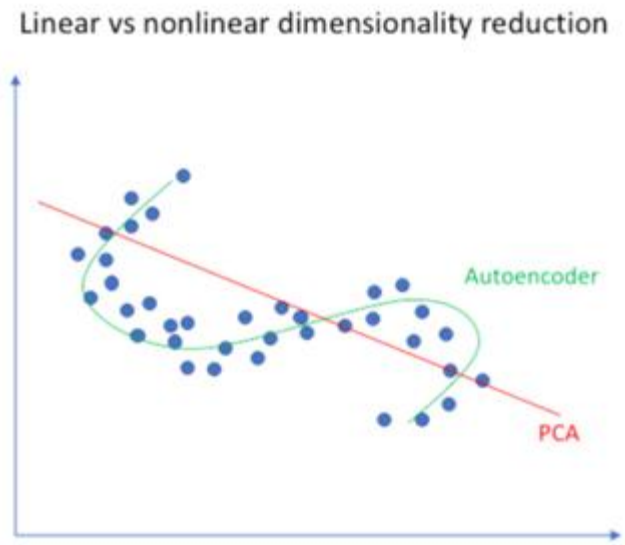


Autoencoders vs PCA

- PCA takes a collection of vectors and produces a usually smaller set of vectors that can be used to approximate the input vectors via linear combination.
- Very efficient for certain applications.
  - No learning takes place
  - No hyperparameter tuning
- PCA might be quicker and can be less expensive to compute than autoencoders.

Autoencoders vs PCA

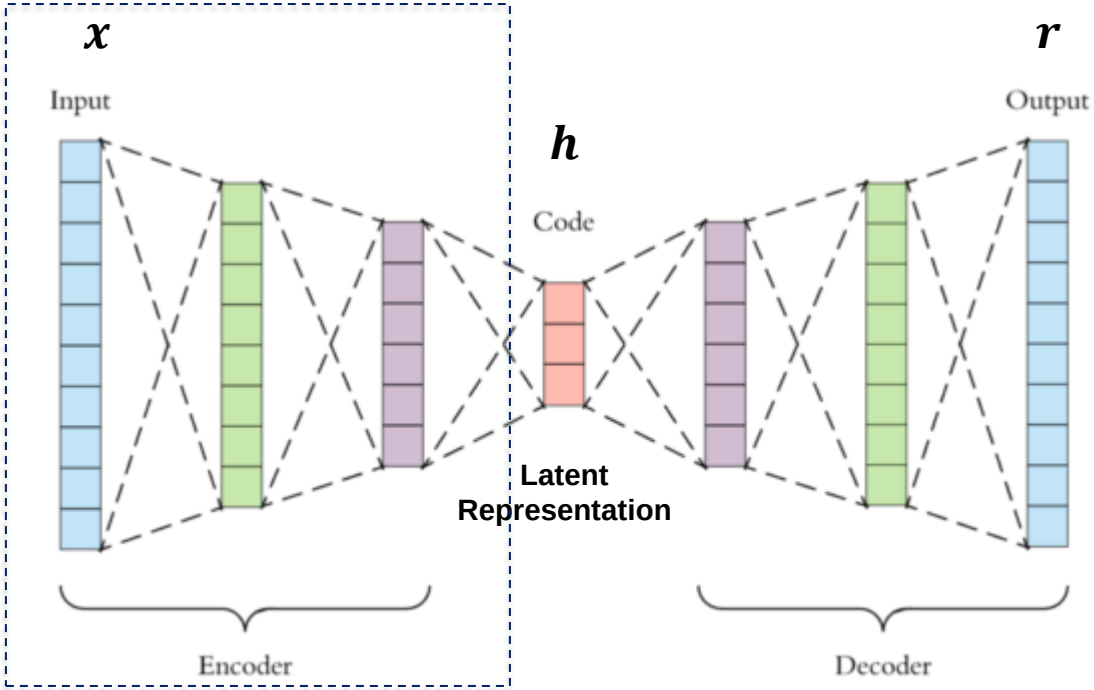
- An autoencoder can learn non-linear transformations with a non-linear activation function and multiple layers.
  - Can express nonlinear dependencies
- It is more efficient to learn several layers with an autoencoder rather than learn one huge transformation with PCA
  - It has been observed that deep learning based autoencoders summarise the information better as compared to the linear PCA model
- It can make use of pre-trained layers from another model to apply transfer learning to enhance the encoder/decoder.



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

Deep Autoencoders: structure

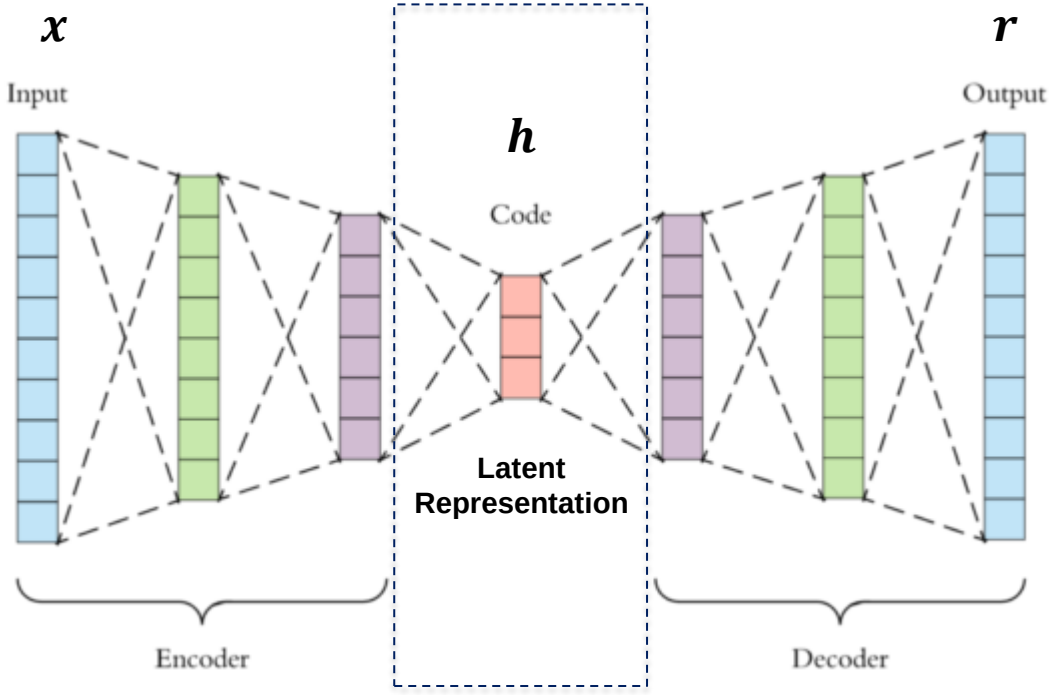
- Encoder:
  - Compress input into a latent-space representation of usually smaller dimension. $h = f(x)$



https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f

Deep Autoencoders: structure

- Code (i.e., also known as Bottleneck):
  - The layer between the encoder and decoder
  - This part of the network represents the compressed input h which is fed to the decoder.
  - Need a well-designed approach to decide which aspects of observed data are relevant information and what aspects can be discarded by balancing two criteria:
    - Compactness of representation, measured as the compressibility.
    - It retains some behaviourally relevant variables from the input.



Deep Autoencoders: Hidden Layer Dimensionality

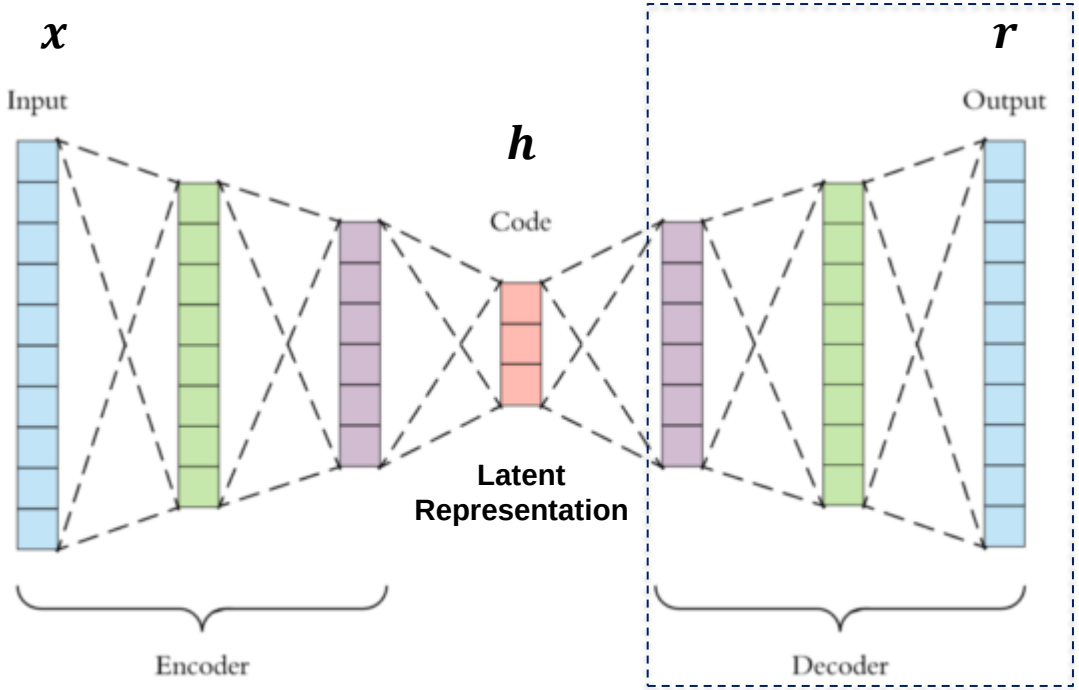
- **Smaller** than the input
  - Will compress the data, reconstruction of the data far from the training distribution will be difficult.
- **Larger** than the input
  - No compression needed
  - Can trivially learn to just copy, no structure is learned (unless you regularize)
  - Does not encourage learning of meaningful features (unless you regularize)

Bottleneck layer (undercomplete)

- Suppose input images are nxn and the latent space is $m < nxn$.
- Then the latent space is not sufficient to reproduce all images.
- Needs to learn an encoding that captures the important features in training data, sufficient for approximate reconstruction.

Autoencoders: structure

- Decoder:
  - Reconstruct input from the latent space. $r = g(f(x))$ with r as close to x as possible
- The decoded data is a lossy reconstruction of the original data and it is reconstructed from the latent space representation.



https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f

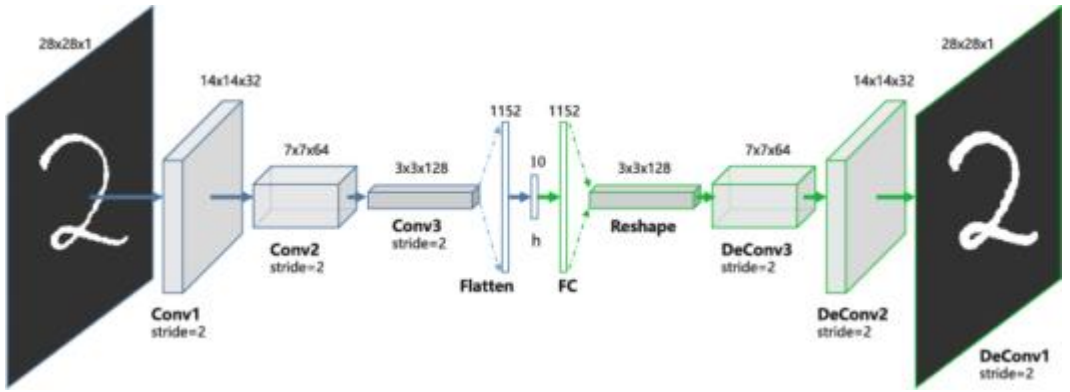
Properties of Autoencoders

- **Data-specific**: Autoencoders are only able to compress data similar to what they have been trained on.
- **Lossy:** The decompressed outputs will be degraded compared to the original inputs.
- **Learned automatically from examples:** It is easy to train specialized instances of the algorithm that will perform well on a specific type of input.
- It doesn't have to use dense layers.
  - It can use convolutional layers to learn which is better for video, image and series data.

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

Convolution Autoencoders

- Autoencoders in their traditional formulation does not take into account the fact that a signal can be seen as a sum of other signals.
- Convolutional Autoencoders use the convolution operator to exploit this observation.
- They learn to encode the input in a set of simple signals and then try to reconstruct the input from them.
- Appropriate operations can be formed to upscale a tensor to higher resolution.

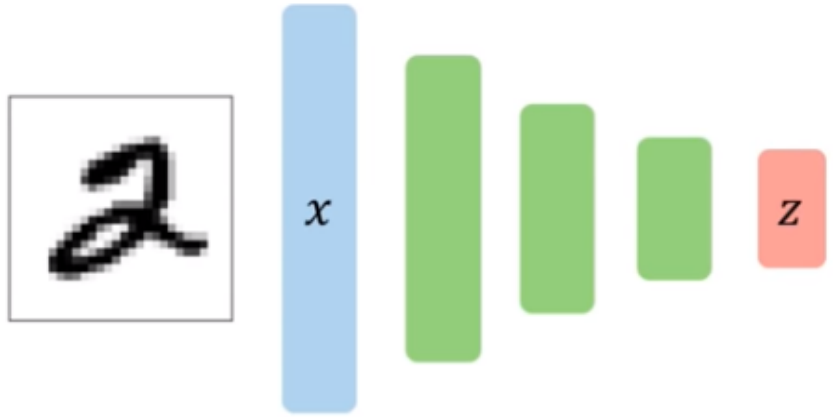


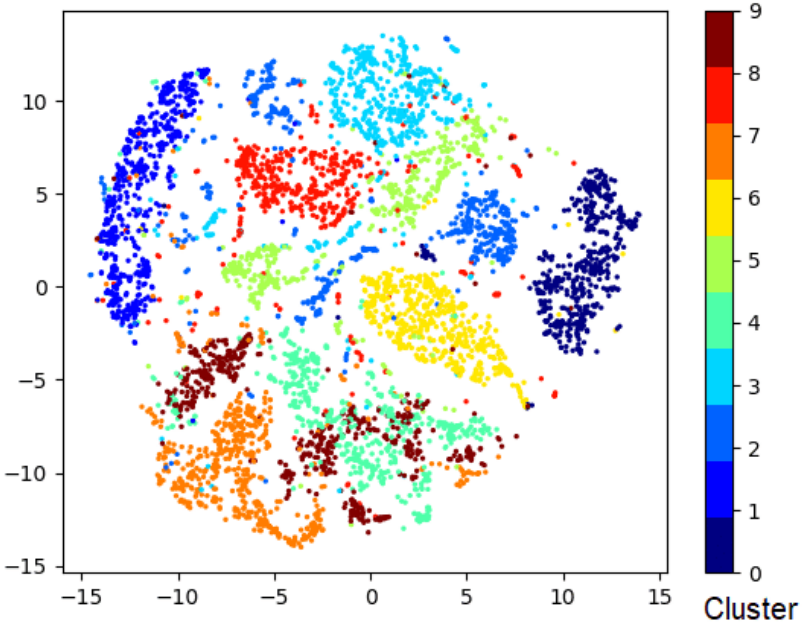
Design choices for Autoencoders

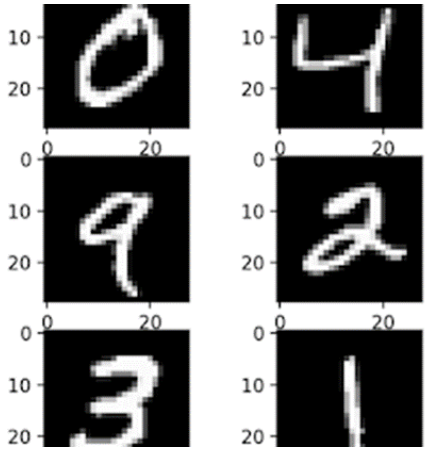
- Number of layers:
  - The autoencoder can consist of as many layers as we want.
- Number of nodes/filters per layer:
  - The number of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder. The decoder is usually symmetric to the encoder in terms of the layer structure.
- Loss function:
  - We either use mean squared error or binary cross-entropy. If the input values are in the range [0, 1] then we typically use binary cross-entropy, otherwise, we use the mean squared error.
- Code size:
  - It represents the number of nodes in the middle layer. Smaller size results in more compression.

Autoencoders: applications

- An interesting practical application is **dimensionality reduction for data visualization**
  - After training we retain only the encoder
  - Encoder learns mapping from the data x, to a low-dimensional latent space z which can be visualized







Autoencoders: applications

- Autoencoders are used for converting any black and white picture into a colored image.
- Depending on what is in the picture, it is possible to tell what the color should be.

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

Autoencoders: applications

- Compression
  - The reconstructed image is the same as our input but with reduced dimensions. It helps in providing the similar image with a reduced pixel value.

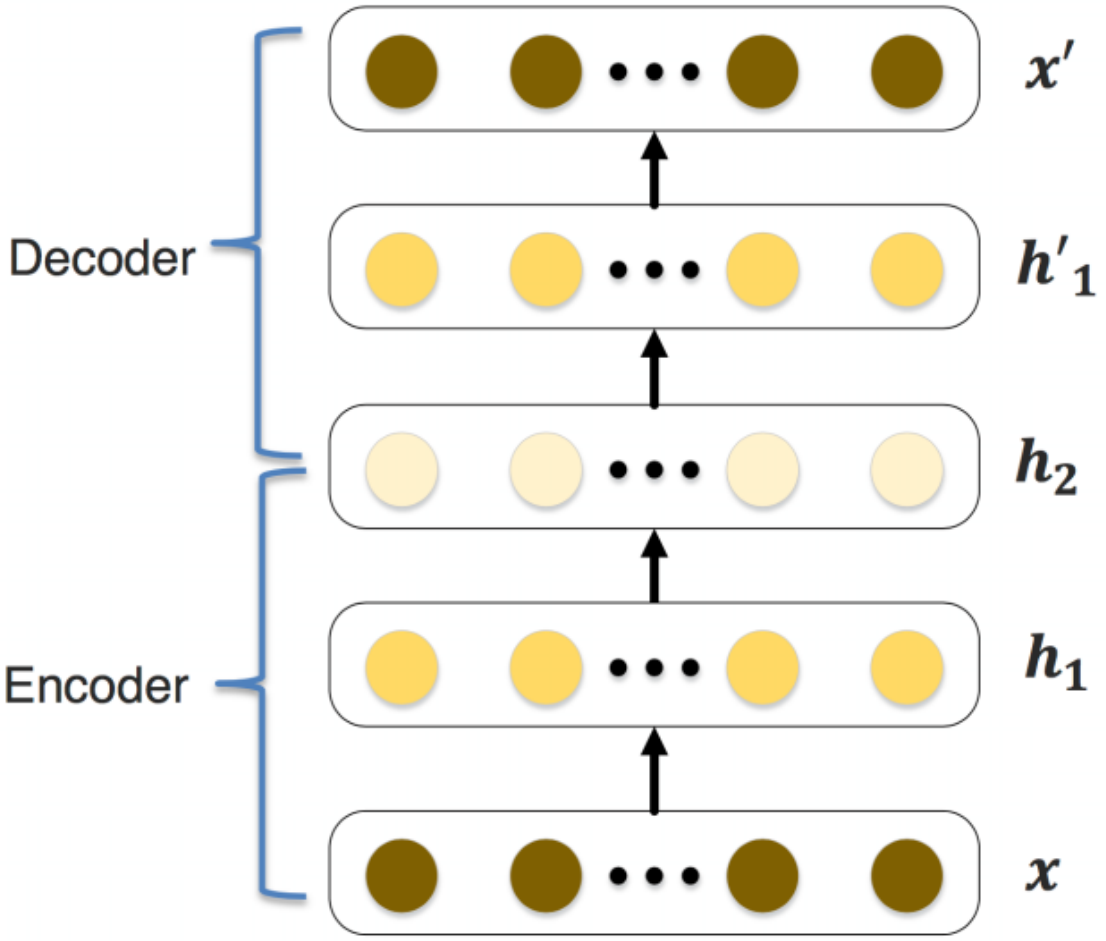
Autoencoders: applications

- Watermark removal
  - It is also used for removing watermarks from images or to remove any object while filming a video or a movie.

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

Stacked (deep) Autoencoders

- What **can we do** with them?
  - Good for compression (better than PCA)
  - Disregard the decoder and use the middle layer as a representation
  - Fine-tune the autoencoder for the task



Capacity

- As with other NNs, overfitting is a problem when capacity is too large for the data.
- Autoencoders address this through some combination of:
  - Bottleneck layer – fewer degrees of freedom than in possible outputs.
  - Training to denoise.
  - Sparsity through regularization.
  - Contractive penalty.

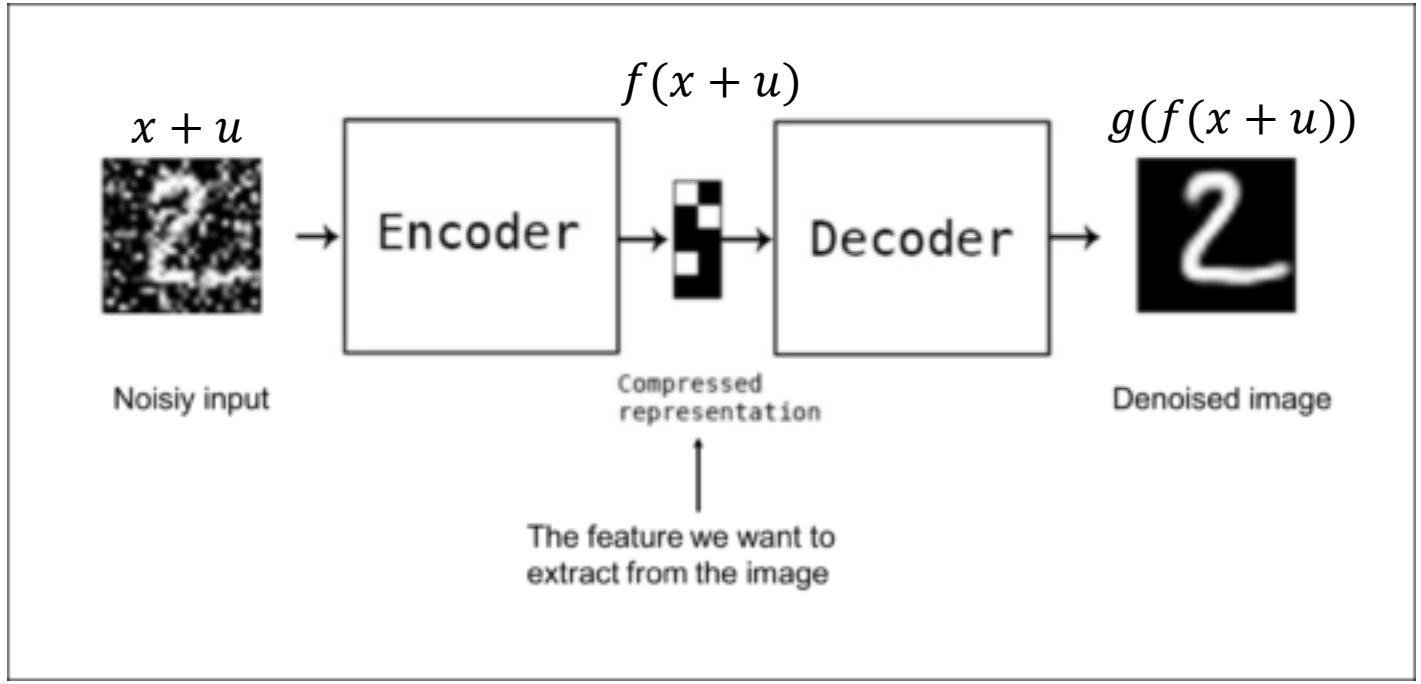
Denoising autoencoders

- **Idea:** add noise to input (input clean image + noise) but learn to reconstruct the original clean image
  - Leads to better representations
  - Prevents copying
- Kaggle has a dataset on damaged documents.

https://blog.keras.io/building-autoencoders-in-keras.html

Denoising autoencoders

- Basic autoencoder trains to minimize the loss between x and the reconstruction $g(f(x))$.
- Denoising autoencoders train to minimize the loss between x and $g(f(x+u))$, where u is **random noise**.
- Same possible architectures, different training data.

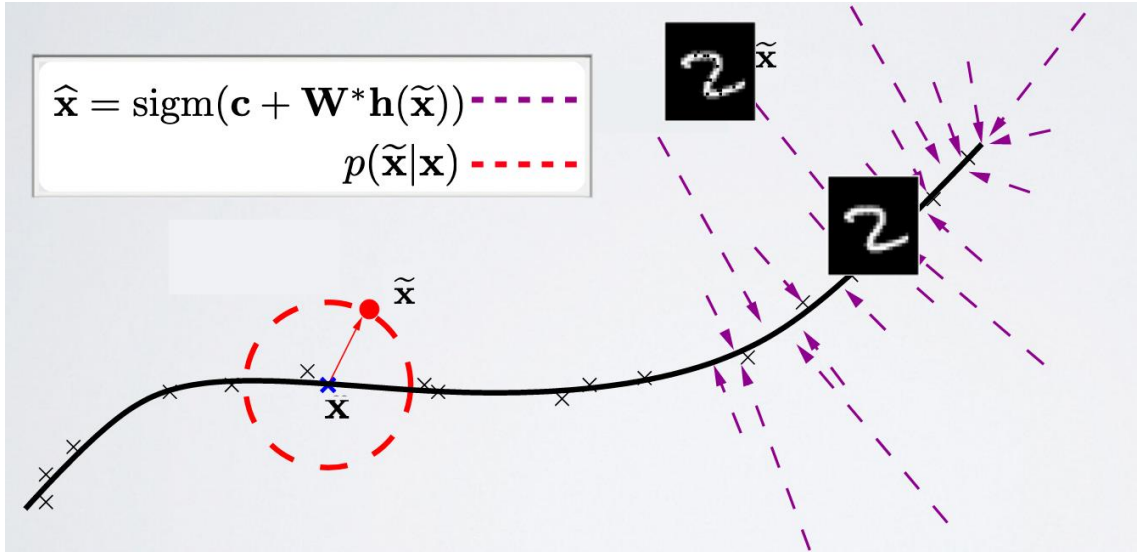


- **Note:** different noise is added during each epoch

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

Denoising autoencoders

- Denoising autoencoders can’t simply memorize the input output relationship.
- Intuitively, a denoising autoencoder learns a projection from a neighborhood of our training data back onto the training data.



https://ift6266h17.files.wordpress.com/2017/03/14_autoencoders.pdf

Sparse Autoencoders

- Sparse autoencoders offer us an alternative method for introducing an information bottleneck **without requiring a reduction** in the number of nodes at our hidden layers.
- Construct a loss function to penalize ***activations*** within a layer.
  - Usually regularize the *weights* of a network, not the activations.
- Individual nodes of a trained model that activate are *data-dependent.*
  - Different inputs will result in activations of different nodes through the network.
- For any given observation, we'll encourage our network to learn an encoding and decoding which only relies **on activating a small number** of neurons.

https://www.jeremyjordan.me/autoencoders/

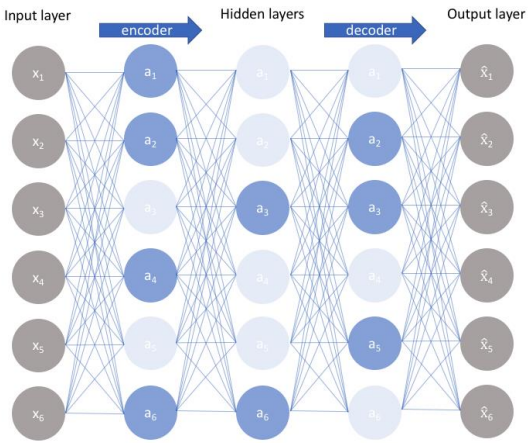
Sparse Autoencoders

- There are two main ways by which we can impose this sparsity constraint:
  - **L1 Regularization**: Penalize the absolute value of the vector of activations *a* in layer *h* for observation *I*

$$\mathcal{L}(x, \hat{x}) + \lambda \sum_i \left| a_i^{(h)} \right|$$

  - **KL divergence:** Use cross-entropy between average activation and desired activation

$$\mathcal{L}(x, \hat{x}) + \sum_j KL\left(\rho || \hat{\rho}_j\right)$$



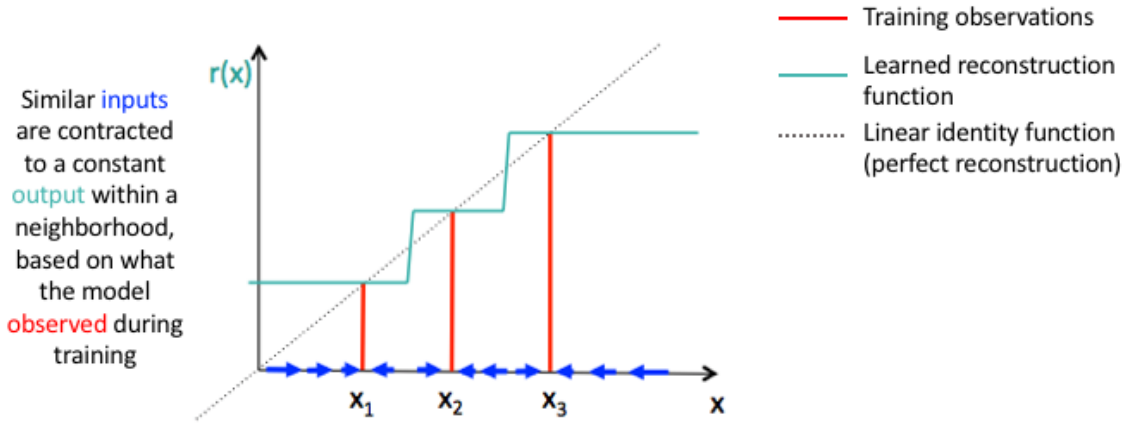
https://www.jeremyjordan.me/autoencoders/

Contractive Autoencoders

- For very similar inputs, the learned encoding should also be very similar.
  - i.e., for small changes to the input, we should still maintain a very similar encoded state
  - Can explicitly train for this by requiring that the derivative of the hidden layer activations are small with respect to the input variations.

Contractive Autoencoders

- Because we're explicitly encouraging our model to learn an encoding in which similar inputs have similar encodings, we're essentially forcing the model to learn how to contract a neighbourhood of inputs into a smaller neighborhood of outputs.
- Notice how the slope (i.e., derivative) of the reconstructed data is essentially zero for local neighbourhoods of input data.



Contractive Autoencoders

- This is accomplished by adding a regularizer, or penalty term, to whatever cost or objective function the algorithm is trying to minimize.
- Regularization loss term as the squared Frobenius norm $\| A \|_F$ of the Jacobian matrix J for the hidden layer activations with respect to the input observations.
  - A Frobenius norm is essentially an L_2 norm for a matrix and the Jacobian matrix simply represents all first-order partial derivatives of a vector-valued function.

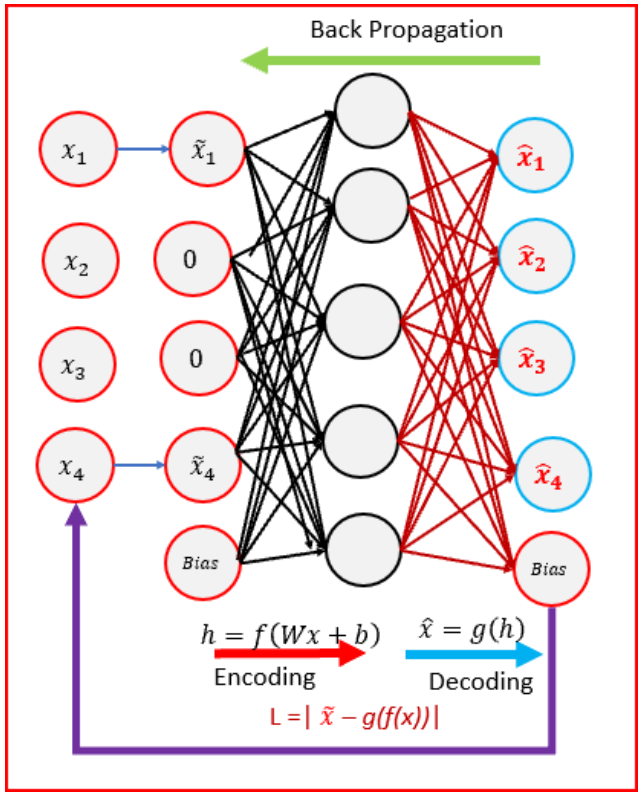
$$L(x, \hat{x}) + \lambda \sum_i \left\| \nabla_x a_i^{(h)}(x) \right\|^2$$

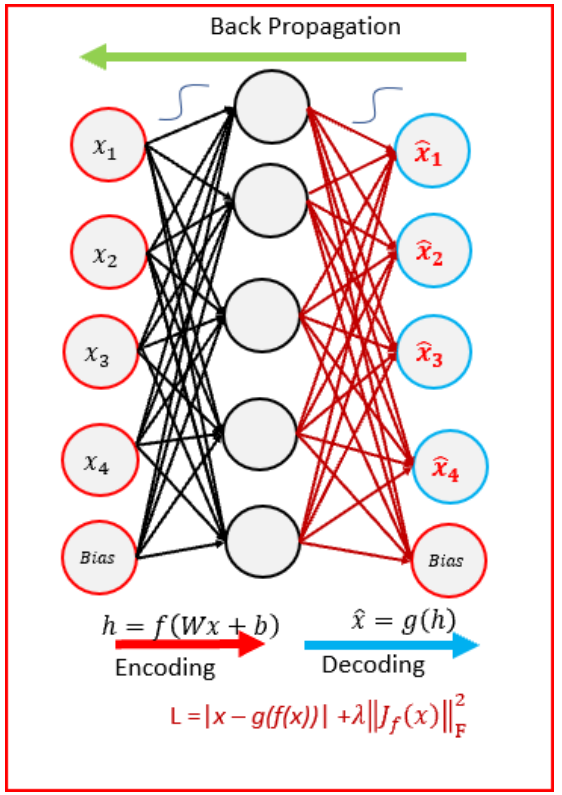
  - where $\nabla_x a_i^{(h)}(x)$ defines the gradient field of the hidden layer activations with respect to the input x, summed over all i training samples

https://www.jeremyjordan.me/autoencoders/

Autoencoders

- Denoising autoencoders make the **reconstruction function (ie. decoder)** resist small but finite-sized perturbations of the input
- Contractive autoencoders make the **feature extraction function (ie. encoder)** resist infinitesimal perturbations of the input.





https://ift6266h17.files.wordpress.com/2017/03/14_autoencoders.pdf

Autoencoders

- Both the denoising and contractive autoencoder can perform well
  - Outperform regular autoencoder in terms of quality of features they extract
  - Advantage of denoising autoencoder : simpler to implement
    - requires adding one or two lines of code to regular autoencoder
    - no need to compute Jacobian of hidden layer
  - Advantage of contractive autoencoder : gradient is deterministic
    - No sampling of noise involved
    - can use second order optimizers (conjugate gradient, LBFGS, etc.)

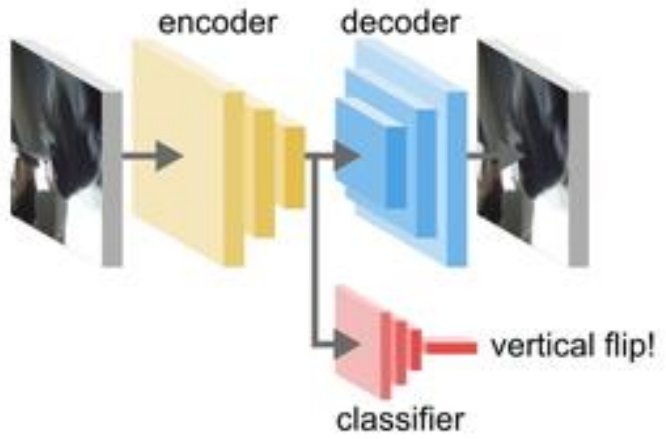
https://ift6266h17.files.wordpress.com/2017/03/14_autoencoders.pdf

Autoencoder Tasks

- Dimensionality Reduction
  - For example, a 2006 study resulted in better results than PCA, with the representation easier to interpret and the categories manifested as well-separated clusters
- Anomaly Detection
  - If the input and reconstruction do not match this might signal an anomaly (out of distribution/corrupted input).
- Feature Learning
  - Good features can be obtained in the hidden layer

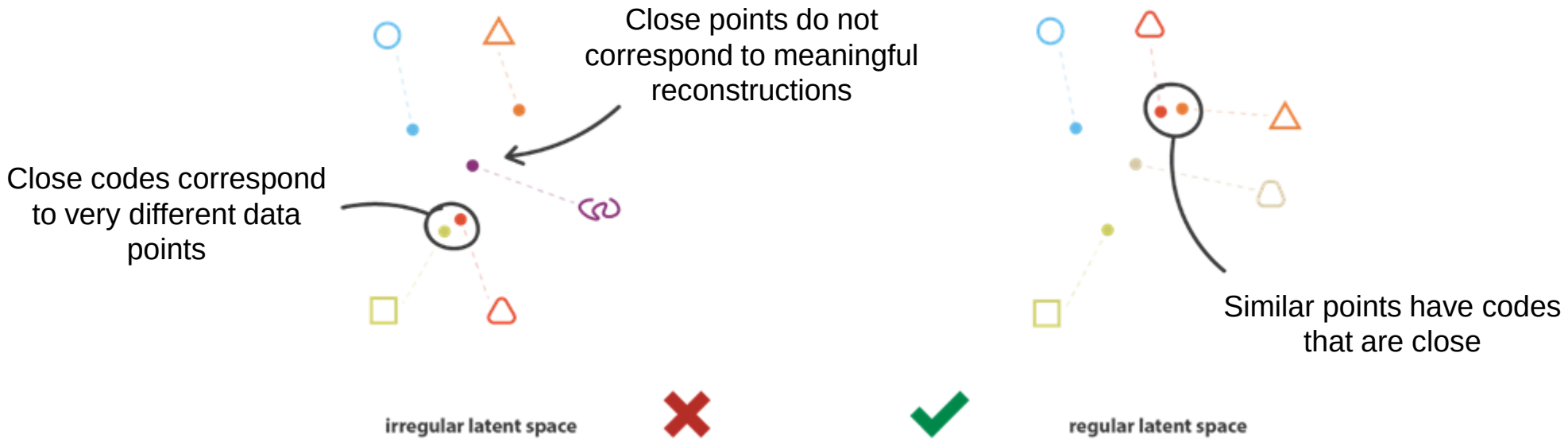
Autoencoder Tasks

- Using autoencoders to initialize weights for supervised learning
- Effective pretraining of weights should act as a regularizer
  - Limits the region of weight space that will be explored by supervised learning
- Can add any additional auxiliary tasks



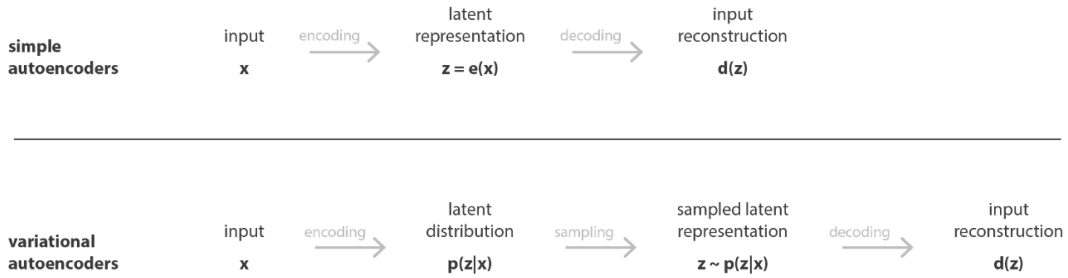
Autoencoder Problem

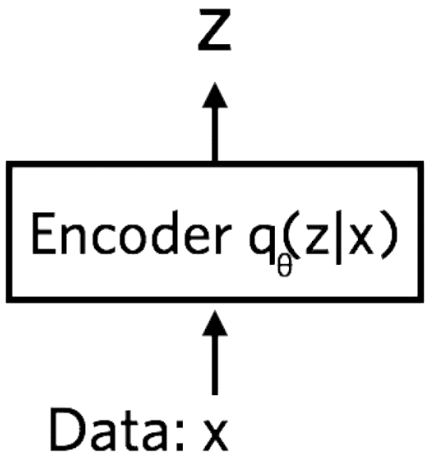
- The latent space of an autoencoder can be extremely **irregular**
  - Close codes in latent space can give very different decoded data
  - Some point of the latent space can give meaningless content once decoded

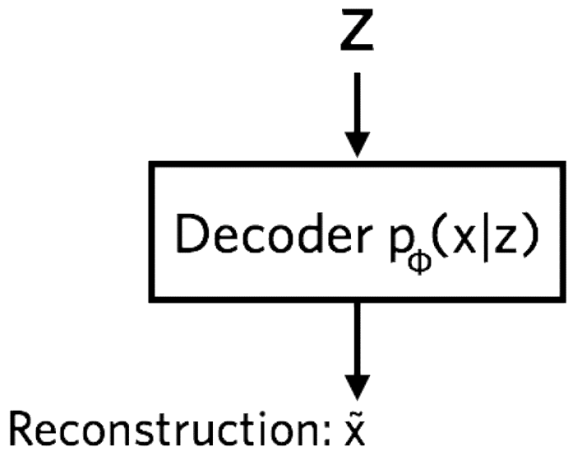


Variational Autoencoder (VAE)

- **Key idea:** make the process probabilistic.
  - i.e., the latent variables, z define a probability distribution depending on the input X
- The encoder takes input and returns parameters for a probability density: i.e., $q_\theta(z|x)$ gives the mean and co-variance matrix.



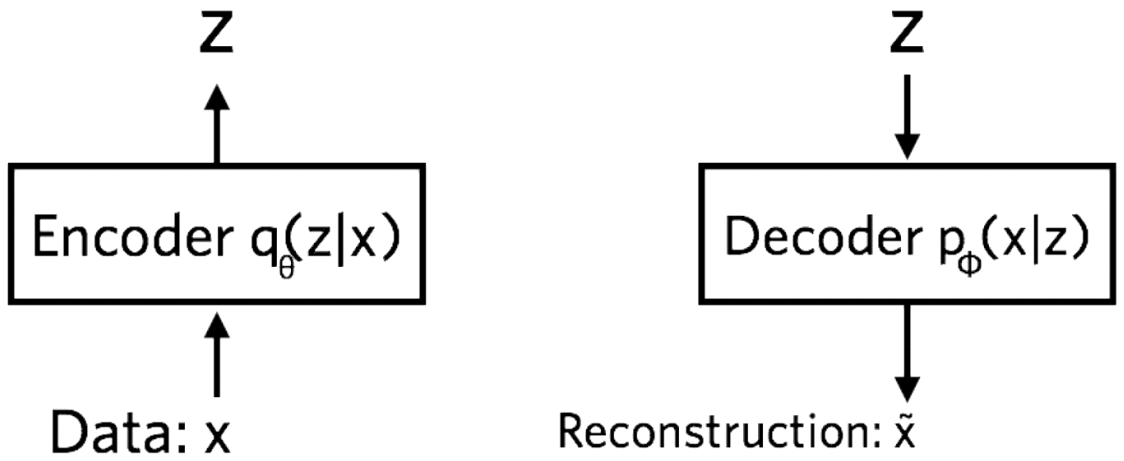




https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

VAE Encoder

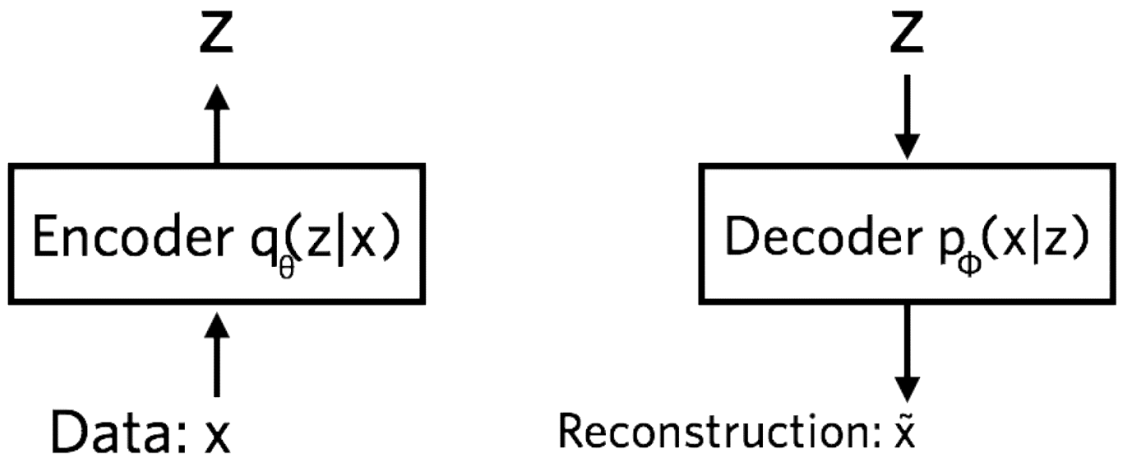
- Implemented via a neural network: each input x gives a vector mean and diagonal covariance matrix that determine the Gaussian density
- Parameters θ for the NN need to be learned – need to set up a loss function.



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

VAE Decoder

- The decoder takes latent variable z and returns parameters for a distribution.
- A point from the latent space is sampled from that distribution
- **Reconstruction** $\tilde{x}$ from the code is produced via neural network, the NN parameters ϕ are learned.



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

VAE Loss

- Loss function for autoencoder: L_2 distance between output and input (or clean input for denoising case)
- For a single input, x_i, we maximize the expected value of returning x_i or minimize the expected negative log likelihood.

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

Reconstruction Loss

- BinaryCrossEntropy(x_i,$\tilde{x}_i$)
- This can also be expressed us $\left\|x_i - p_\phi(x_i|z)\right\|^2$

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

VAE Loss (2)

- But the fact that VAEs encode inputs as distributions instead of simple points is not sufficient to ensure continuity and completeness.
- **Problem:** the weights may adjust to memorize input images via z. i.e., input that we regard as similar may end up very different in z space. Network can "cheat" by learning narrow distributions.
- **Solution:** Try to force $q_\Theta(z|x_i)$ to be close to a standard normal (or some other simple density).

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

VAE Loss (3)

- KL-Divergence Regularizer
  - Inferred latent distribution: $p_\Theta(z|x)$
  - Fixed prior on latent distribution: $p(z)$
  - $d_{KL}(p_\Theta(z|x)||p(z))=\sum_{x\in X} p_\Theta(z|x)\log(\frac{p_\Theta(z|x)}{p(z)})$
- Common choice of prior: Gaussian $N(\mu,\sigma)$
  - Encourages encodings to be distributed evenly around the center of the latent space
  - Penalize the network when it tries to "cheat" by clustering points in specific regions (i.e., memorizing the data)
- KLD Regularizer for Gaussian case: $\mathrm{p(z)} = N(\mu = 0, \sigma = 1)$ and $p_\Theta(z|x)\sim N(\mu_{(\theta)}, \sigma_{(\theta)})$

$$-\frac{1}{2}\sum_{j=1}^{J}\left(\sigma_{j(\theta)}^2+\mu_{j(\theta)}^2-1-\log\left(\sigma_{j(\theta)}\right)\right)$$

Kullback-Leibler (KL) Divergence: is a measure of how one probability distribution is different from a second, reference probability distribution.

VAE Loss (4)

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

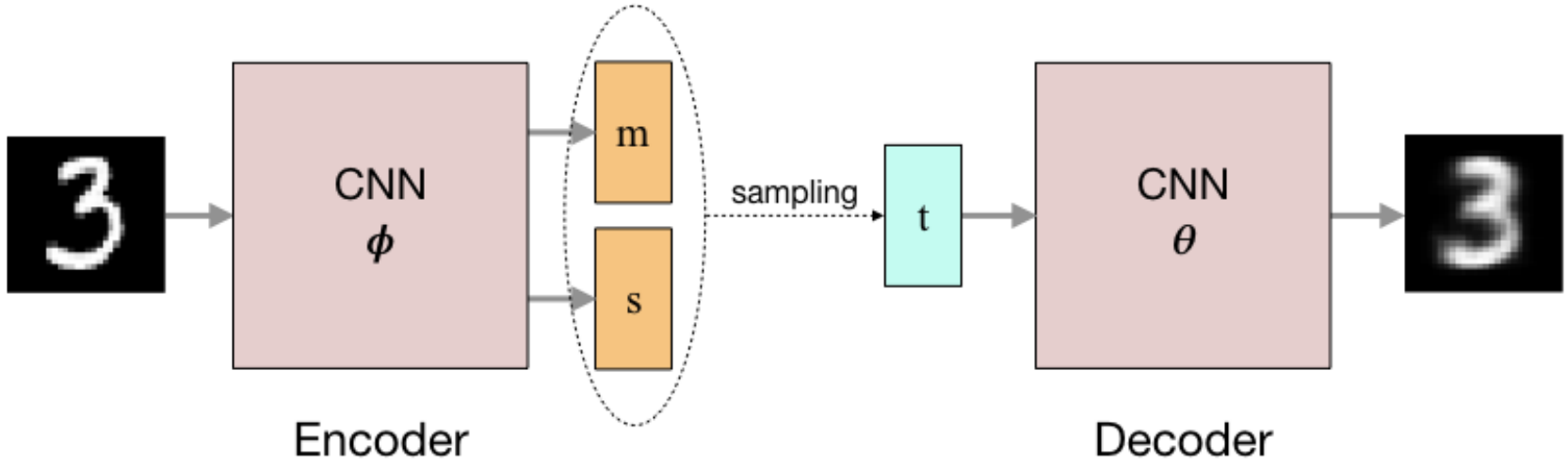
$$L_i(\theta, \phi) = -\mathrm{E}_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] + \boldsymbol{\lambda} d_{KL}(p_\Theta(z|x_i)||p(z))$$

- The first term promotes recovery of the input.
- The second term keeps the encoding continuous – the encoding is compared to a fixed $p(z)$ regardless of the input, which inhibits memorization.
- The hyperparameter λ controls how much emphasis is given on the reconstruction vs the regularization term

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

VAE training

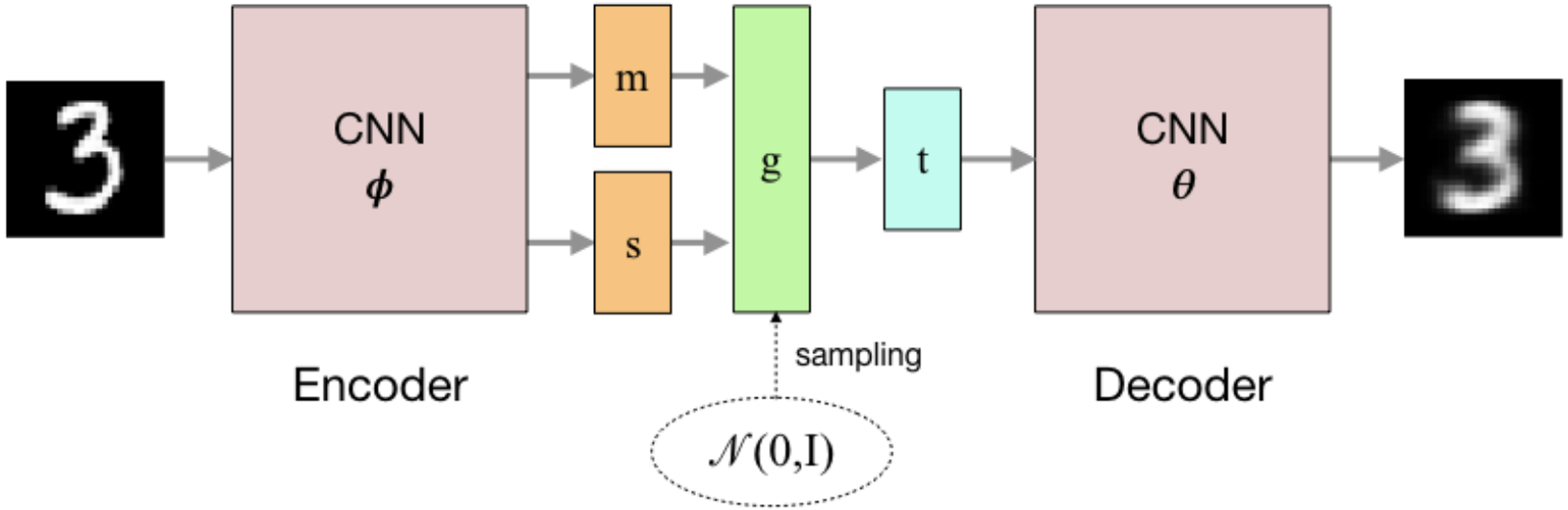
- **Problem:** The expectation would usually be approximated by choosing samples and averaging.
  - This is not differentiable wrt θ and ϕ.



https://nbviewer.jupyter.org/github/krasserm/bayesian-machine-learning/blob/master/variational_autoencoder.ipynb

VAE training (2)

- **Reparameterization:** If z is $N(\mu(x_i), \sigma^2(x_i))$, then we can sample z using $z = \mu(x_i) + \sigma(x_i) \times \epsilon$, where ϵ is $N(0, \mathbf{I})$.
- So, we can draw samples from $N(0, \mathbf{I})$, which doesn't depend on the parameters.



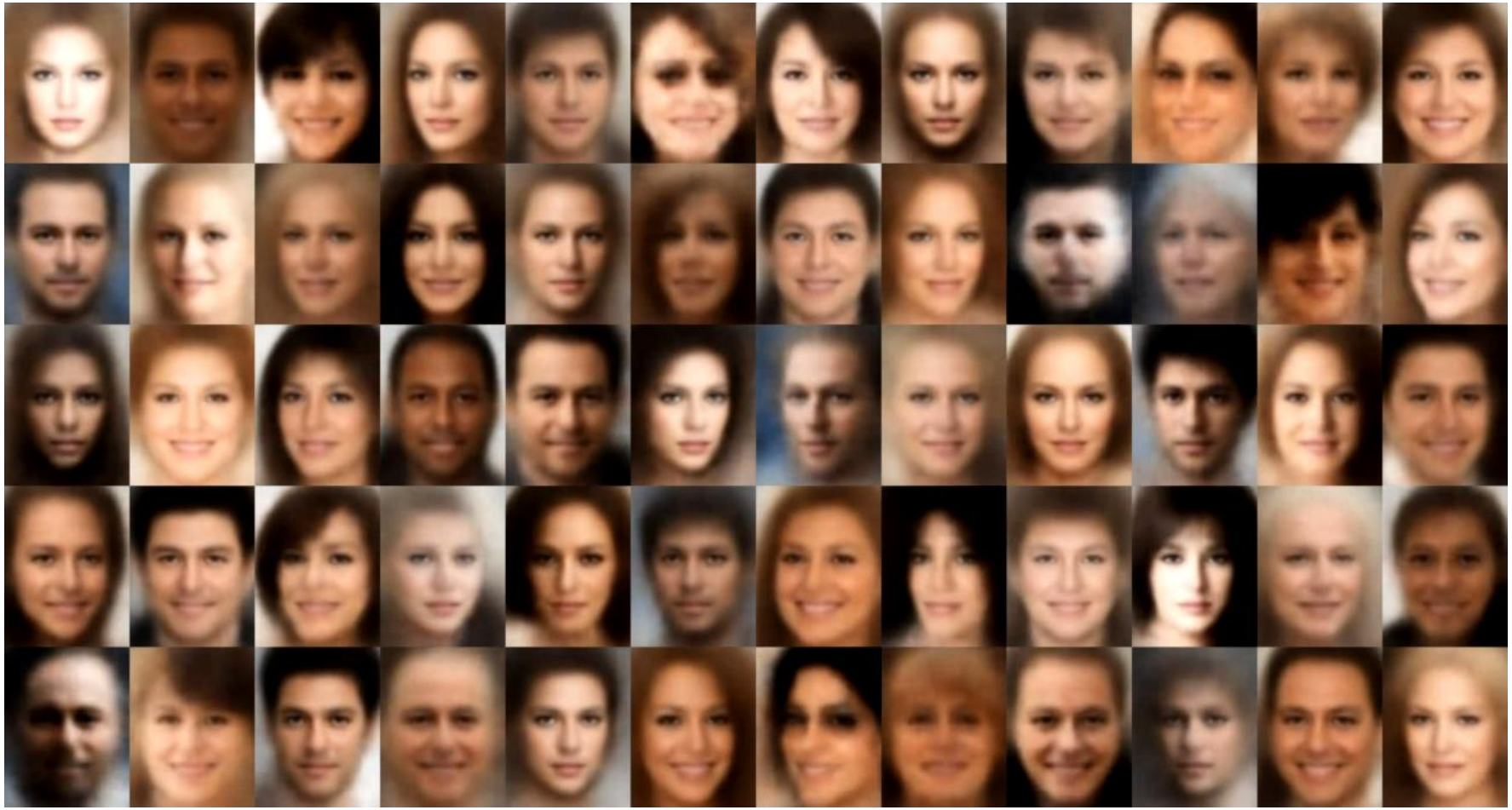
https://nbviewer.jupyter.org/github/krasserm/bayesian-machine-learning/blob/master/variational_autoencoder.ipynb

VAE generative model

- After training, $q_\theta(z|x_i)$ is close to a standard normal, N(0,1) – easy to sample.
- Using a sample of z from $q_\theta(z|x_i)$ as input to sample from $p_\varphi(x|z)$ gives an approximate reconstruction of x_i, at least in expectation.
- If we sample any z from $N(0, I)$ and use it as input to to sample from $p_\varphi(x|z)$ then we can approximate the entire data distribution $p(x)$.
  i.e., we can generate new samples that look like the input but aren't in the input.
- We can sample from this distribution to get random values of the lower-dimensional representation z.

Random Samples From Generative VAE

- https://www.whichfaceisreal.com/



Face images generated with a Variational Autoencoder (source: Wojciech Mormul on Github).

Supervised vs Unsupervised Learning

- ***Supervised learning*** – learning with labeled data
  - Approach: collect a large dataset, manually label the data, train a model, deploy
  - It is the dominant form of ML at present
  - Learned feature representations on large datasets are often transferred via pre-trained models to smaller domain-specific datasets
- ***Unsupervised learning*** – learning with unlabeled data
  - Approach: discover patterns in data either via clustering similar instances, or density estimation, or dimensionality reduction …
- ***Self-supervised learning*** – representation learning with unlabeled data
  - Learn useful feature representations from unlabeled data through pretext tasks
  - The term “self-supervised” refers to creating its own supervision (i.e., without supervision, without labels)
  - Self-supervised learning is one category of unsupervised learning

Self-Supervised Learning

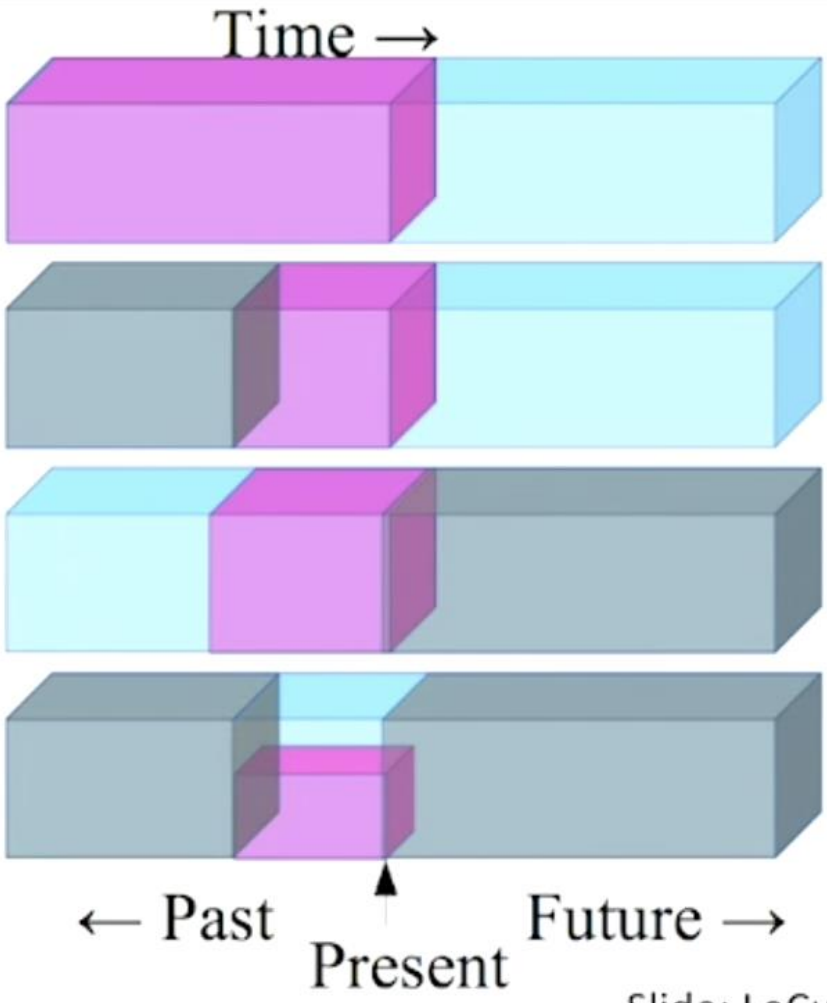
- Why self-supervised learning?
  - Creating labeled datasets for each task is an expensive, time-consuming, tedious task
    - Requires hiring human labelers, preparing labeling manuals, creating GUIs, creating storage pipelines, etc.
    - High quality annotations in certain domains can be particularly expensive (e.g., medicine)
  - Self-supervised learning takes advantage of the vast amount of unlabeled data on the internet (images, videos, text)
    - Rich discriminative features can be obtained by training models without actual labels
  - Self-supervised learning can potentially generalize better because we learn more about the world
- ***Challenges*** for self-supervised learning
  - How to select a suitable pretext task for an application
  - There is no gold standard for comparison of learned feature representations
  - Selecting a suitable loss functions, since there is no single objective as the test set accuracy in supervised learning

Self-Supervised Learning

- Self-supervised learning versus unsupervised learning
  - Self-supervised learning (SSL)
    - Aims to extract useful feature representations from raw unlabeled data through ***pretext tasks***
    - Apply the feature representation to improve the performance of ***downstream tasks***
  - Unsupervised learning
    - Discover patterns in unlabeled data, e.g., for clustering or dimensionality reduction
  - Note also that the term “self-supervised learning” is sometimes used interchangeably with “unsupervised learning”
- Self-supervised learning versus transfer learning
  - Transfer learning is often implemented in a supervised manner
    - E.g., learn features from a labeled ImageNet, and transfer the features to a smaller dataset
  - SSL is a type of transfer learning approach implemented in an unsupervised manner
- Self-supervised learning versus data augmentation
  - Data augmentation is often used as a regularization method in supervised learning
  - In SSL, image rotation of shifting are used for feature learning in raw unlabeled data

Supervision comes from the data

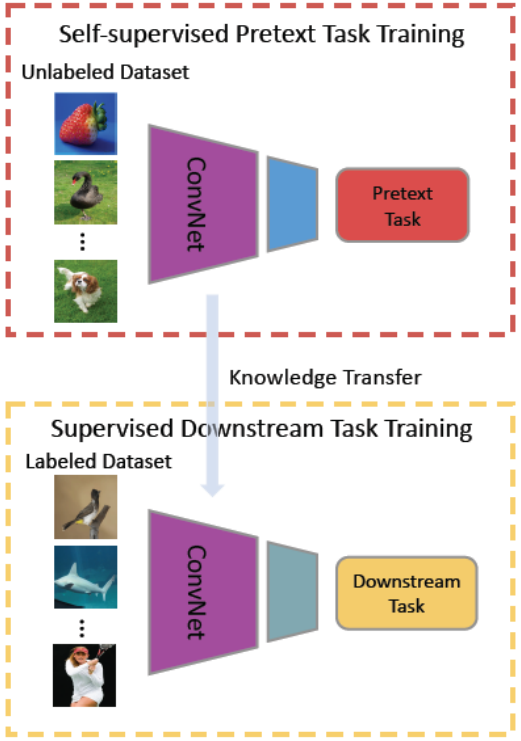
- Predict any part of the input from any other part.
- Predict the future from the past.
- Predict the future from the recent past.
- Predict the past from the present.
- Predict the top from the bottom.
- Predict the occluded from the visible
- Pretend there is a part of the input you don't know and predict that.



Slide: LeCun

Self-Supervised Learning

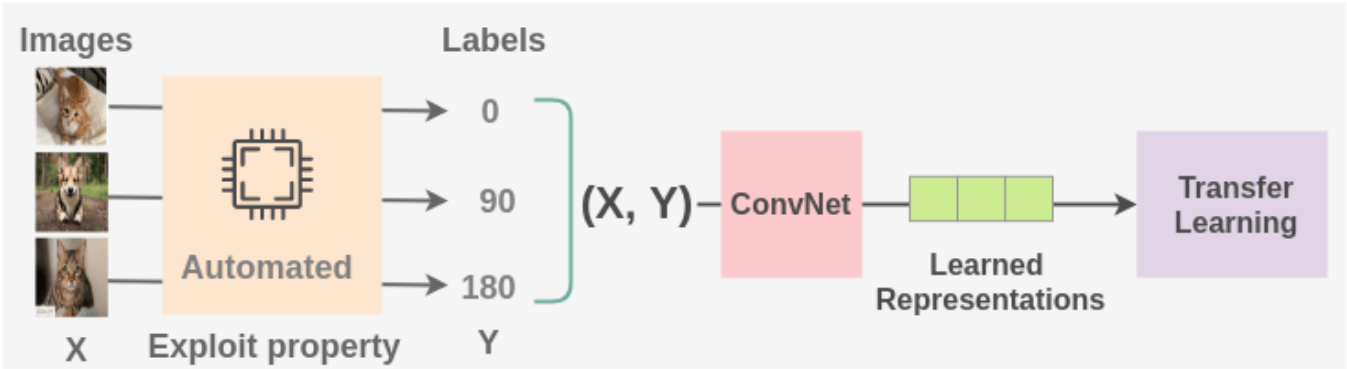
- One more depiction of the general pipeline for self-supervised learning is shown in the figure
  - For the downstream task, re-use the trained ConvNet base model, and fine-tune the top layers on a small labeled dataset



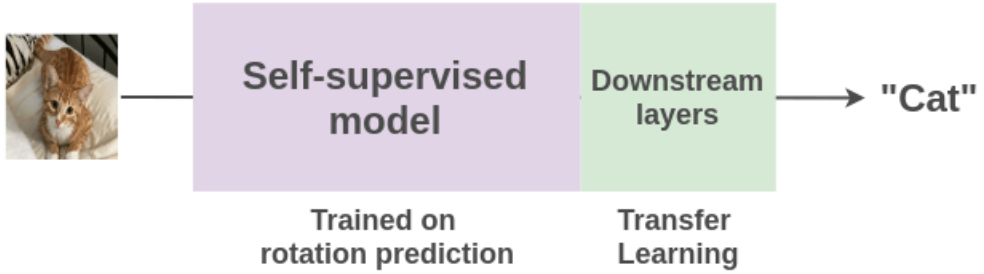
Picture from: Jing and Tian (2019) Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey

Self-Supervised Learning

- Self-supervised learning example
  - ***Pretext task***: train a model to predict the rotation degree of rotated images with cats and dogs (we can collect million of images from internet, labeling is not required)



  - ***Downstream task***: use transfer learning and fine-tune the learned model from the pretext task for classification of cats vs dogs with very few labeled examples

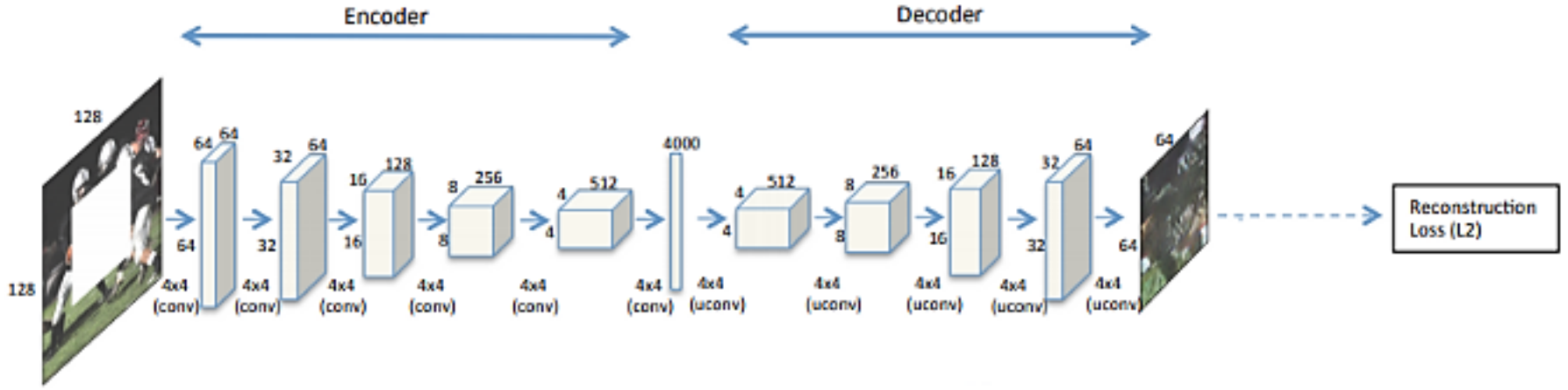


Context Encoders

- ***Predict missing pieces***, also known as ***context encoders***, or ***inpainting***
  - Pathak (2016) Context Encoders: Feature Learning by Inpainting
- Training data: remove a random region in images
- Pretext task: fill in a missing piece in the image
  - The model needs to understand the content of the entire image, and produce a plausible replacement for the missing piece

Context Encoders

- The initially considered model uses an encoder-decoder architecture
  - The encoder and decoder have multiple Conv layers, and a shared central fully-connected layer
  - The output of the decoder is the reconstructed input image
  - A Euclidean ℓ_2 distance is used as the reconstruction loss function $\mathcal{L}_{\text{rec}}$



Summary

- An autoencoder is a neural network architecture capable of discovering structure within data in order to develop a compressed representation of the input.
- Many different variants of the general autoencoder architecture exist with the goal of ensuring that the compressed representation represents meaningful attributes of the original data input.
- Autoencoders can be modified to be generative (VAE) and create completely new data points.
- The biggest challenge when working with autoencoders is getting the model to actually learn a meaningful and generalizable latent space representation.
- Self-supervised learning
  - A label is constructed from only input signals without human-annotation
  - Predict (well-designed) alternative tasks to what we actually want to do