AIMS | African Institute for Mathematical Sciences RWANDA

AMMI Review sessions

Deep Learning (8)

Autoencoders

March 3rd 2020

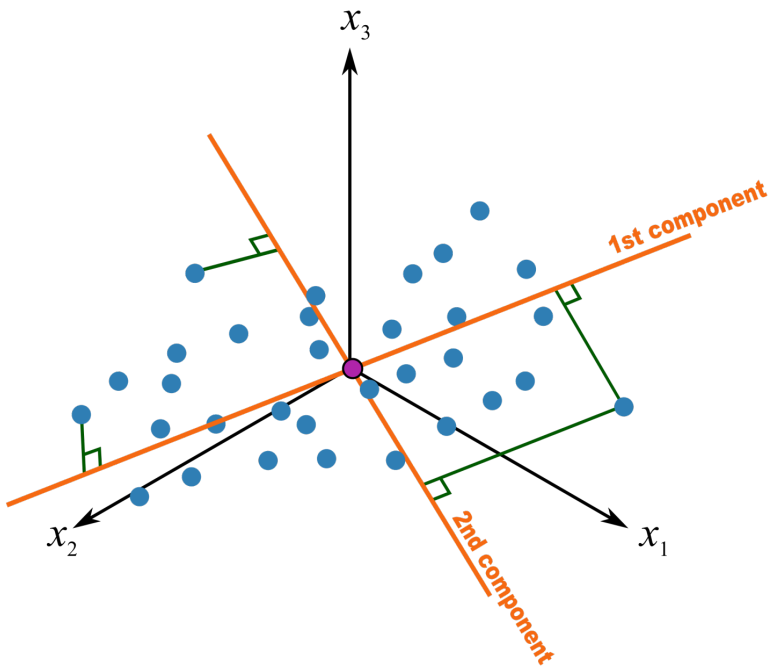
Unsupervised learning

- In unsupervised learning setting the dataset doesn’t contain labels
- We’re interested in learning the hidden underlying structure
- Examples: Clustering, Compression, Feature & Representation learning, Dimensionality reduction, Generative models ,etc.

PCA

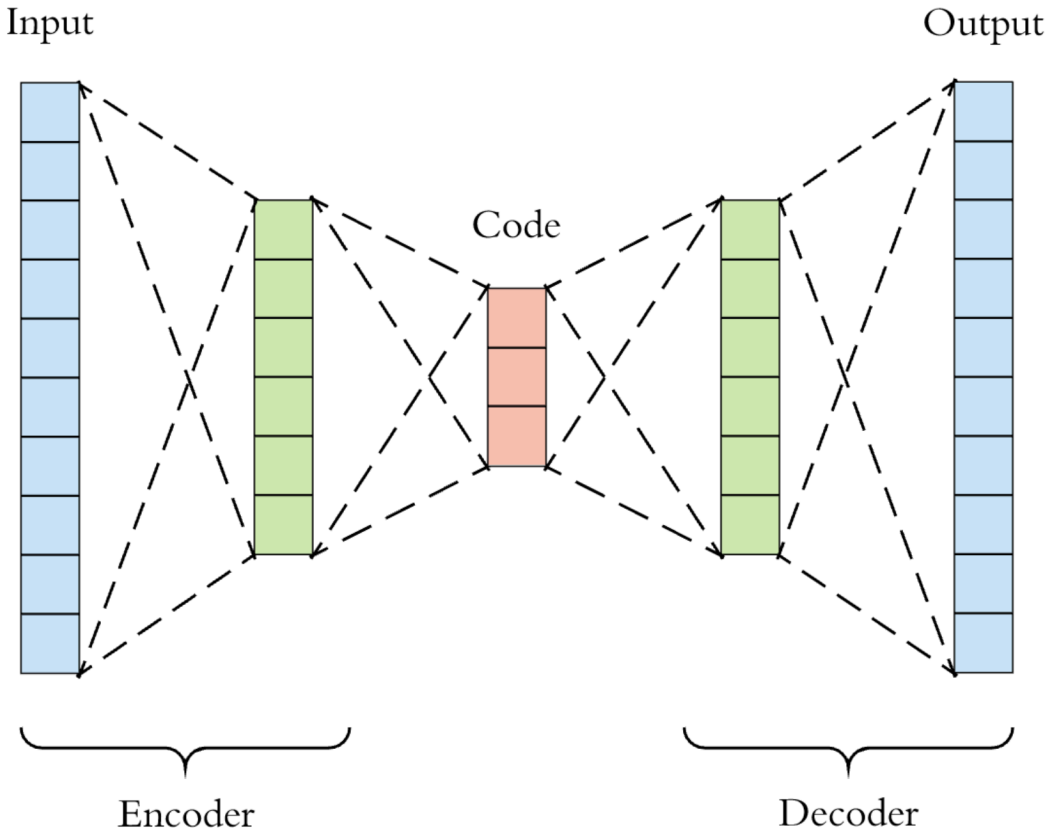
- Statistical approach for data compression and visualization, Invented by Karl Pearson in 1901

- Weakness: linear components only.



Traditional Autoencoder

- An autoencoder is a neural network that is trained to attempt to copy its input to its output. Internally, it has a hidden layer h that describes a code used to represent the input.



Traditional Autoencoder

- Given data x (no labels) we would like to learn the functions f (encoder) and g (decoder) where:

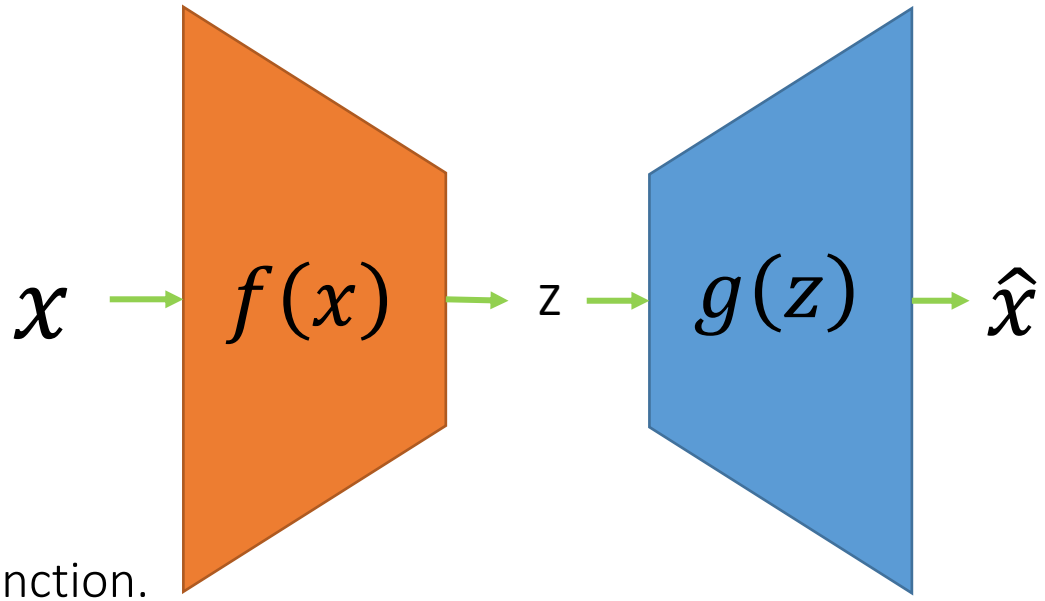
- $f(x) = s(wx + b) = z$

and

- $g(z) = s(w'z + b') = \hat{x}$

- s.t $h(x) = g\big(f(x)\big) = \hat{x}$

- where h is an approximation of the identity function.



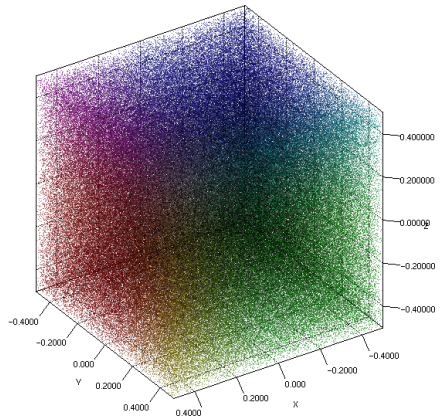
- Autoencoding = automatically encoding the data

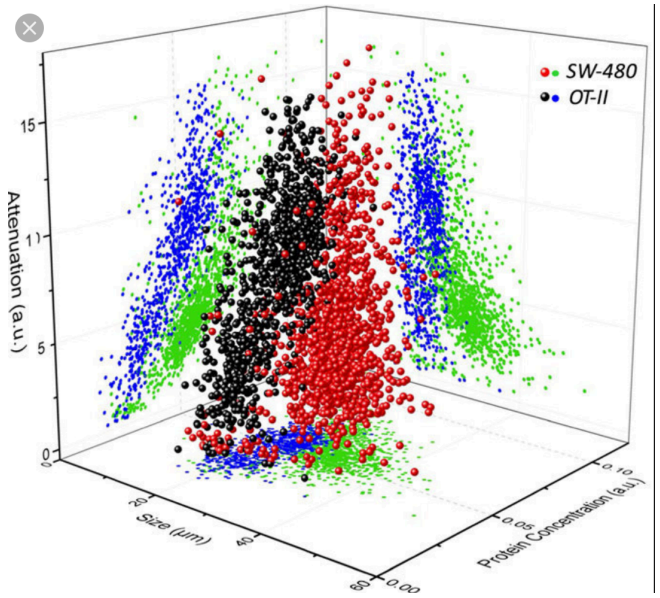
Latent variable

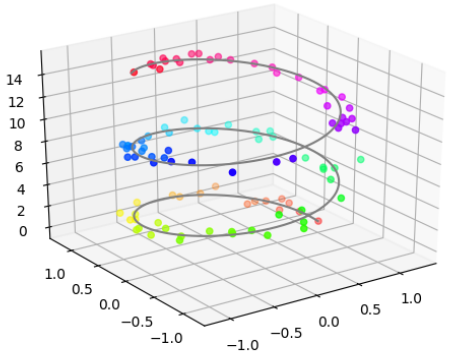
Myth of cave

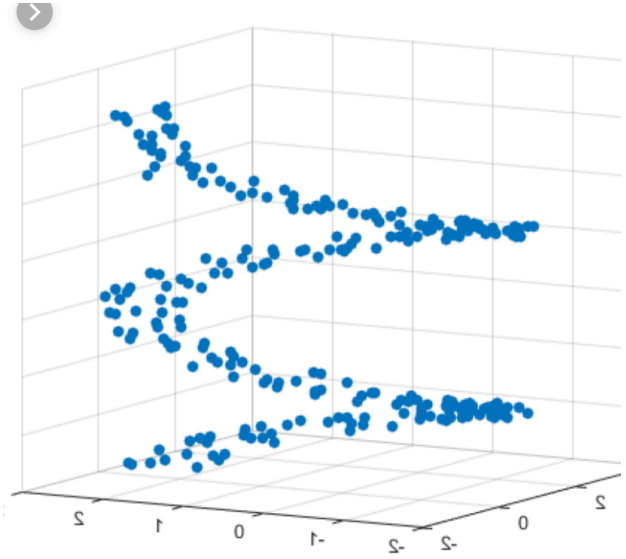
Manifold structure of data

- The “manifold hypothesis” of ML says "high dimensional data are points in a low dimensional manifold with high dimensional noise added"

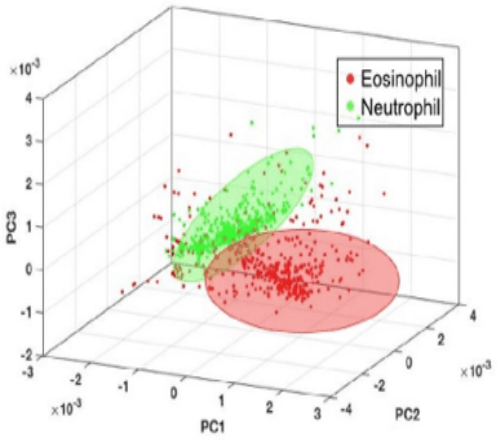


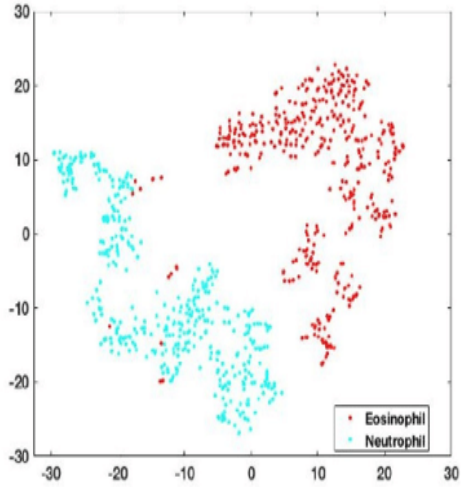






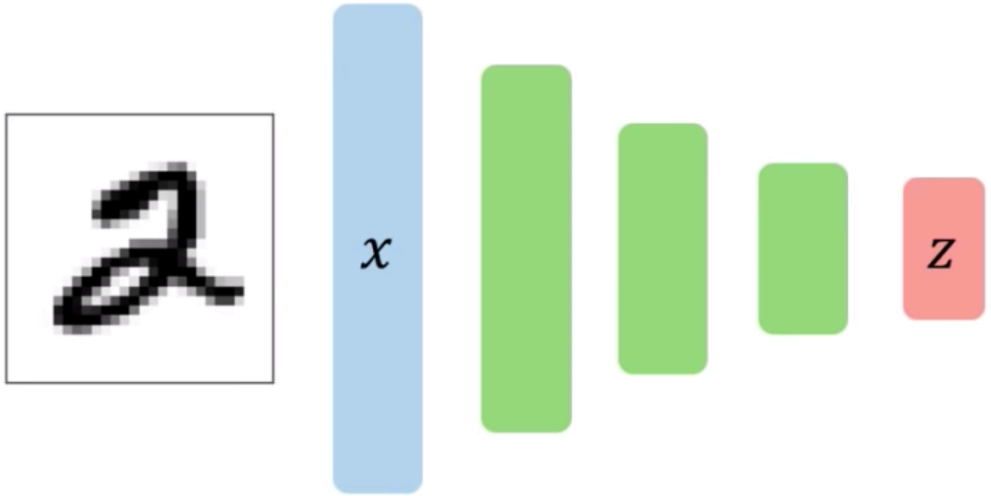
(a) (b)





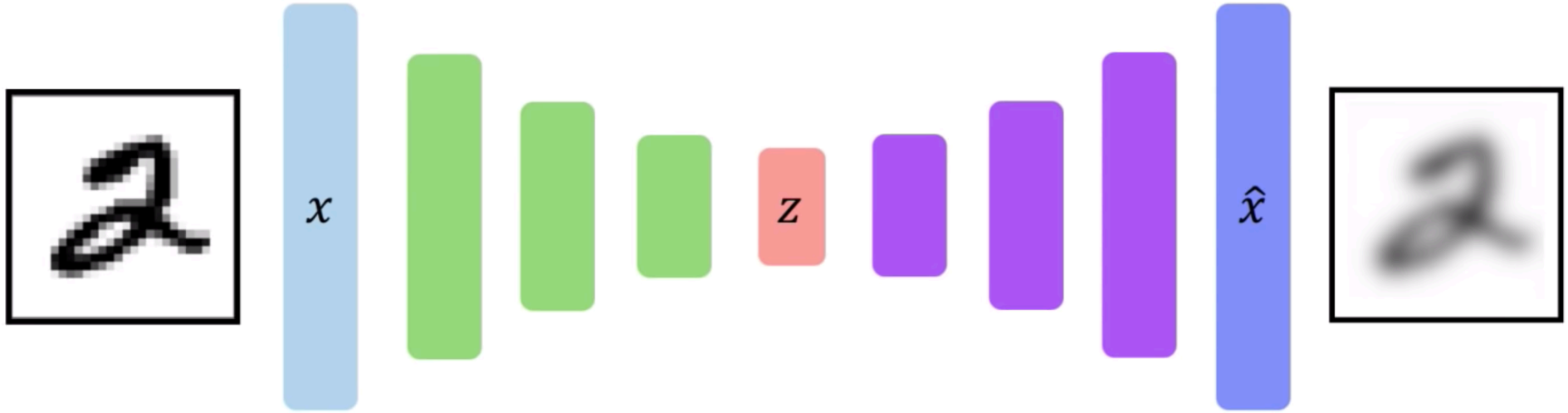
Traditional Autoencoder

- Encoder learns the mapping from the data x, to lower dimensional latent space z



Traditional Autoencoder

- How we can learn this latent space ?
  Train the model to use these feature to construct the original data



Decoder, learns mapping back from latent space to reconstructed observation

Traditional Autoencoder

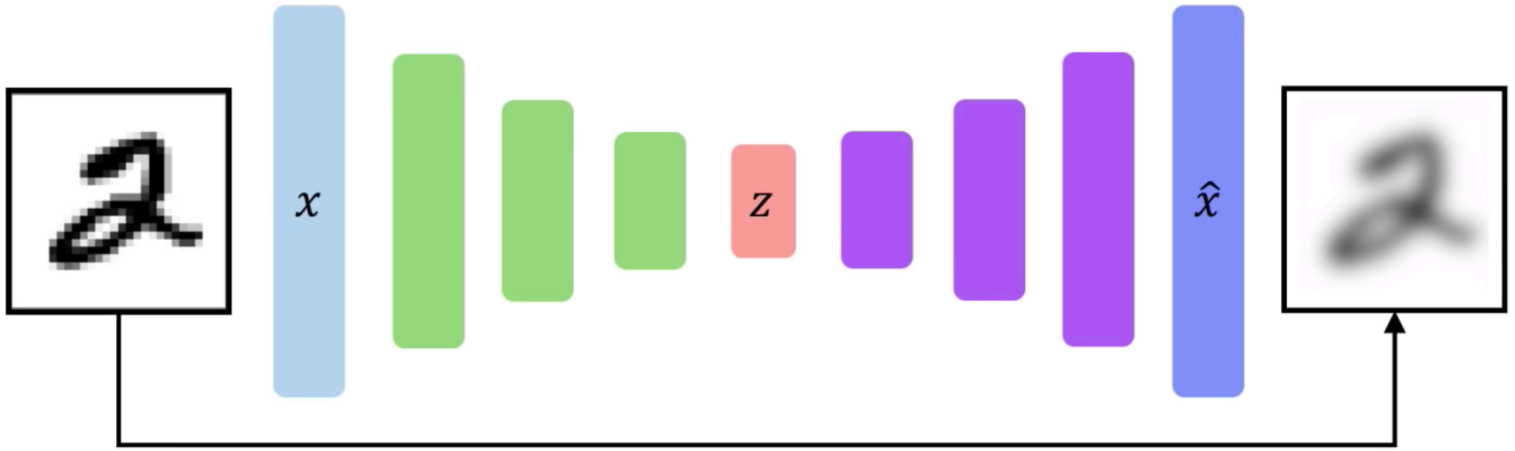
- The autoencoder idea was a part of NN history for decades (LeCun et al, 1987).
- Traditionally an autoencoder is used for dimensionality reduction and feature learning.
- Recently, the connection between autoencoders and latent space modeling has brought autoencoders to the front of generative modeling, as we will see in the next lectures.

Traditional Autoencoder

- If an autoencoder succeeds in simply learning to set g(f(x)) = x everywhere, then it is not especially useful.

- Instead, autoencoders are designed to be unable to learn to copy perfectly. Usually they are restricted in ways that allow them to copy only approximately, and to copy only input that resembles the training data. Because the model is forced to prioritize which aspects of the input should be copied, it often learns useful properties of the data.

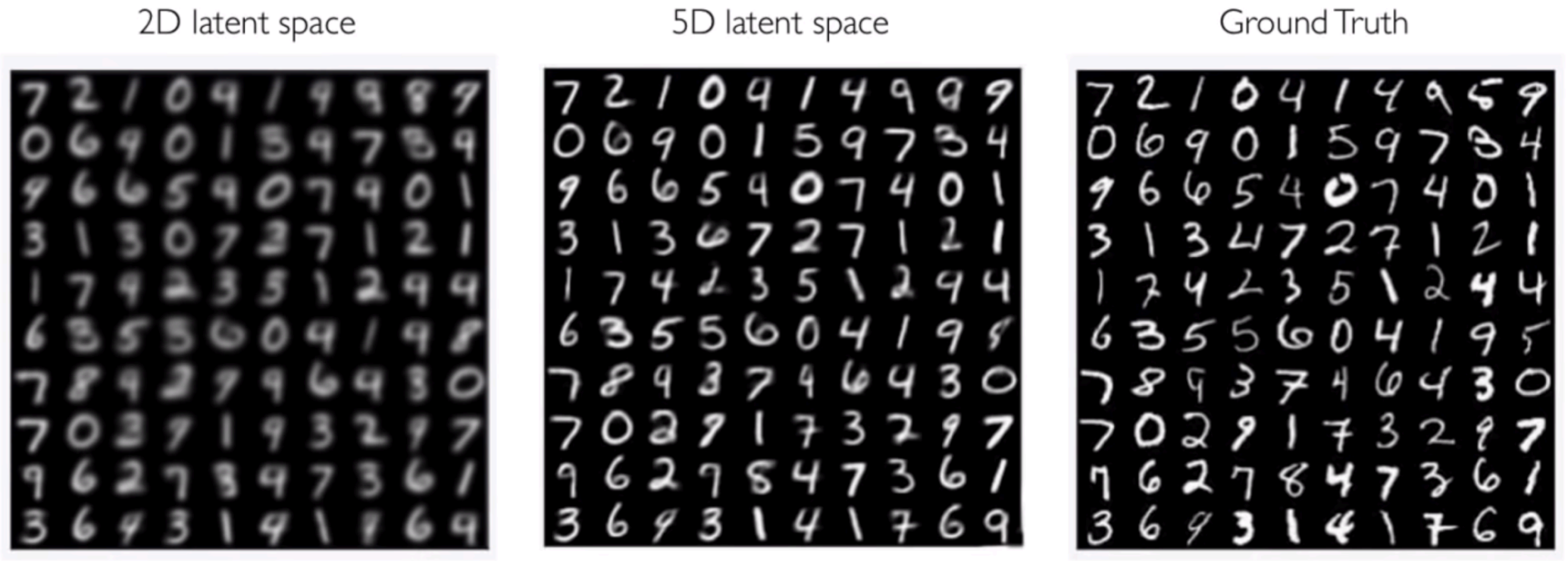
Training AE



- If our input is interpreted as bit vectors or vectors of bit probabilities the *cross entropy* can be used

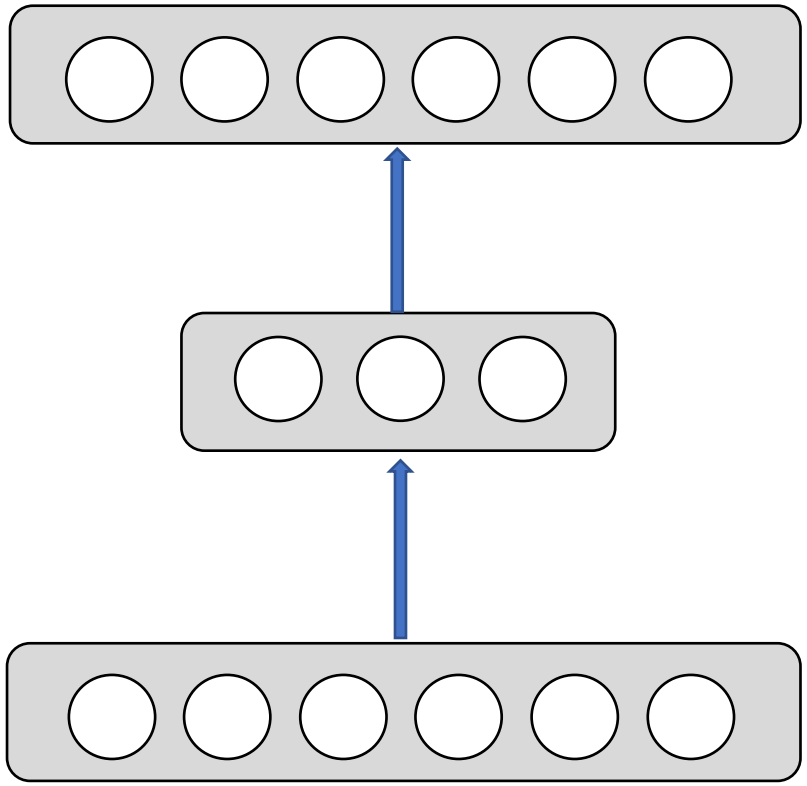
Dimensionality of latent space

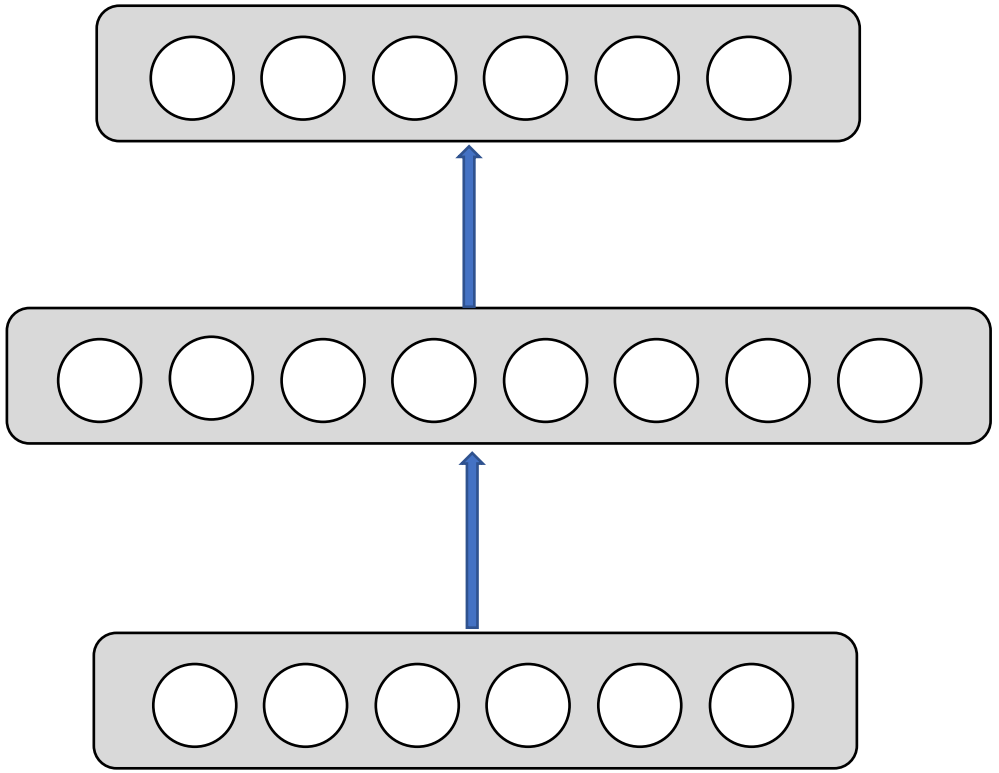
- Smaller latent space result in poorer quality of reconstruction



Undercomplete AE VS overcomplete AE

- We distinguish between two types of AE structures according to the dimensionality of the latent space



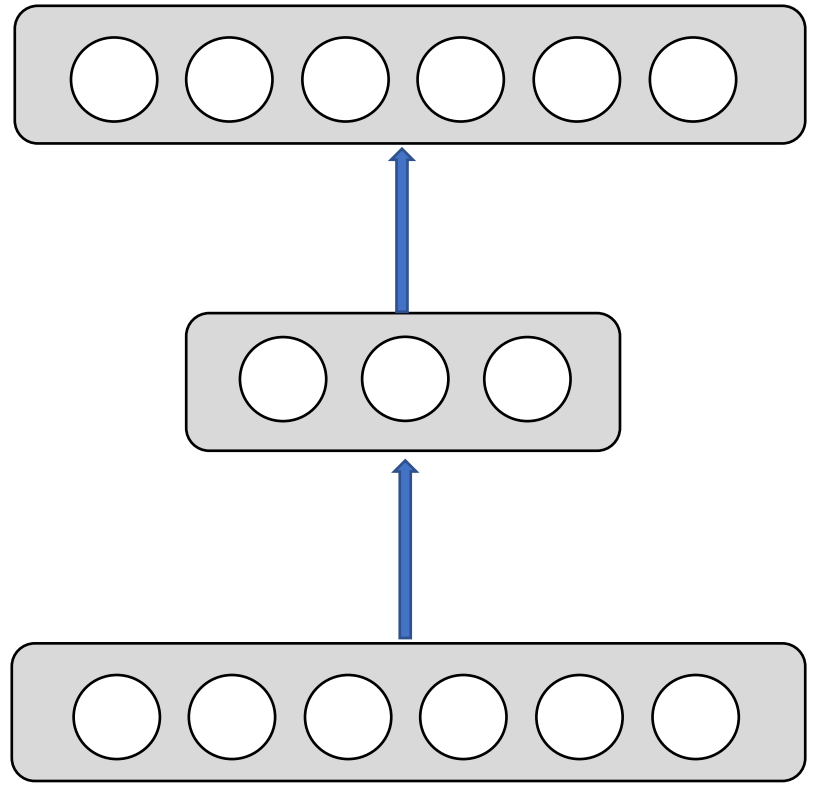


Undercomplete AE

- Hidden layer is **Undercomplete** if smaller than the input layer
  - Compresses the input
  - Compresses well only for the training dist.

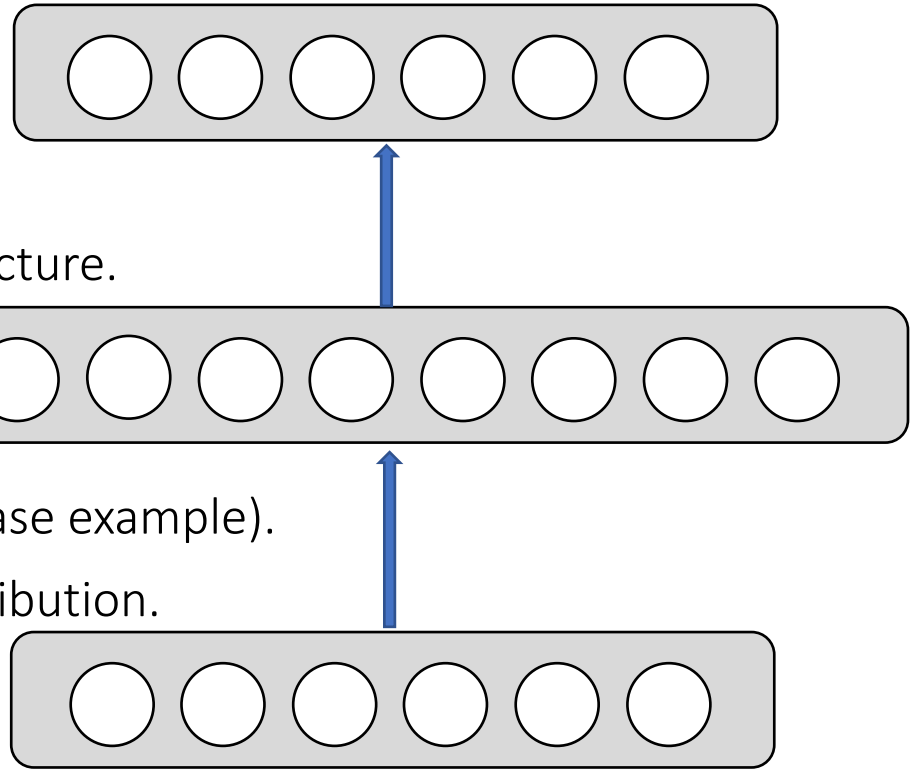
- Hidden nodes will be
  - Good features for the training distribution.
  - Bad for other types on input

- When the decoder is linear and L is the mean squared error, an undercomplete autoencoder learns to span the same subspace as PCA.



Overcomplete AE

- Hidden layer is **Overcomplete** if greater than the input layer
  - No compression in hidden layer.
  - Each hidden unit could copy a different input component.
- No guarantee that the hidden units will extract meaningful structure.
- Adding dimensions is good for training a linear classifier (XOR case example).
- A higher dimension code helps model learn more complex distribution.



Regularized Autoencoders (motivation)

- Undercomplete autoencoders fail to learn anything useful if the encoder and decoder are given too much capacity.

- Overcomplete autoencoders with even a linear encoder and linear decoder can learn to copy the input to the output without learning anything useful about the data distribution.

- Ideally, one could train any architecture of autoencoder successfully, choosing the code dimension and the capacity of the encoder and decoder based on the complexity of distribution to be modeled.

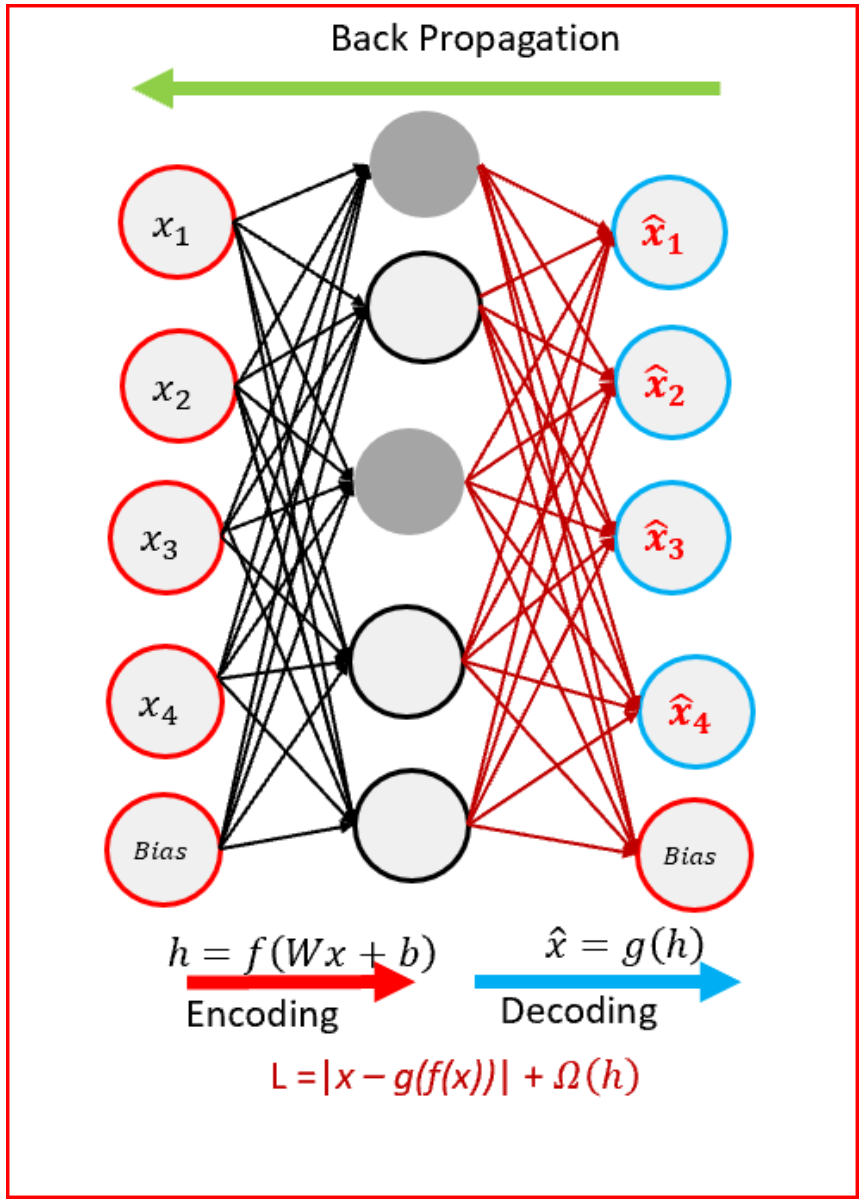
- Regularized autoencoders provide the ability to do so.

Regularized Autoencoders

- Regularized autoencoders use a loss function that encourages the model to have other properties besides the ability to copy its input to its output.

Sparse autoencoders

- Sparsity constraint is introduced on the hidden layer. This is to prevent output layer copy input data.
- It take the highest activation values in the hidden layer and zero out the rest of the hidden nodes.
- This prevents autoencoders to use all of the hidden nodes at a time and forcing only a reduced number of hidden nodes to be used.
- As we activate and inactivate hidden nodes for each row in the dataset. Each hidden node extracts a feature from the data



Sparse autoencoders: sparsity penalty

- Recall: $a_j^{(\mathrm{Bn})}$ is defined to be the activation of the j^{th} hidden unit (bottleneck) of the autoencoder.
- Let $a_j^{(\mathrm{Bn})}(x)$ be the activation of this specific node on a given input x.

- Further let,

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}\left[a_j^{(\mathrm{Bn})}\left(x^{(i)}\right)\right]$$

be the average activation of hidden unit j (over the training set).

Sparse autoencoders: sparsity penalty

- Recall: $a_j^{(\mathrm{Bn})}$ is defined to be the activation of the j^{th} hidden unit (bottleneck) of the autoencoder.
- Let $a_j^{(\mathrm{Bn})}(x)$ be the activation of this specific node on a given input x.

- Further let,

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}\left[a_j^{(\mathrm{Bn})}\left(x^{(i)}\right)\right]$$

be the average activation of hidden unit j (over the training set).

- Thus we would like to force the constraint: $\hat{\rho}_j = \rho$

- where ρ is a "sparsity parameter", typically small. In other words, we want the average activation of each neuron j to be close to ρ.

Sparse autoencoders: sparsity penalty

- We need to penalize $\hat{\rho}_j$ for deviating from ρ.
- Many choices of the penalty term will give reasonable results.

For example: $$\sum_{j=1}^{Bn} KL(\rho|\hat{\rho}_j)$$

- where $KL(\rho|\hat{\rho}_j)$ is a Kullback-Leibler divergence function.

Sparse autoencoders: sparsity penalty

- We need to penalize $\hat{\rho}_j$ for deviating from ρ.
- Many choices of the penalty term will give reasonable results.

For example: $$\sum_{j=1}^{Bn} KL(\rho|\hat{\rho}_j)$$

- where $KL(\rho|\hat{\rho}_j)$ is a Kullback-Leibler divergence function.

- Our overall cost functions is now:

$$J_S(W,b) = J(W,b) + \beta \sum_{j=1}^{Bn} KL(p|\hat{\rho}_j)$$

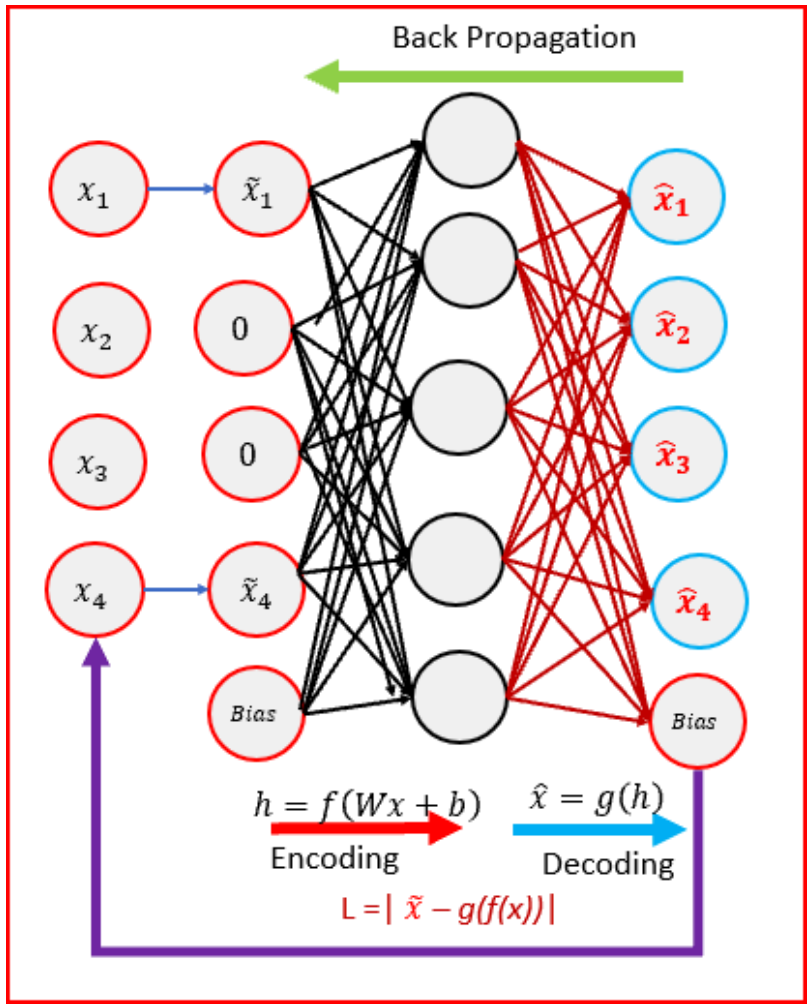
- *Note: We need to know $\hat{\rho}_j$ before hand, so we have to compute a forward pass on all the training set.

Denoising Autoencoders

- Denoising refers to intentionally adding noise to the raw input before providing it to the network. Denoising can be achieved using stochastic mapping.

- Denoising autoencoders create a corrupted copy of the input by introducing some noise. This helps to avoid the autoencoders to copy the input to the output without learning features about the data.

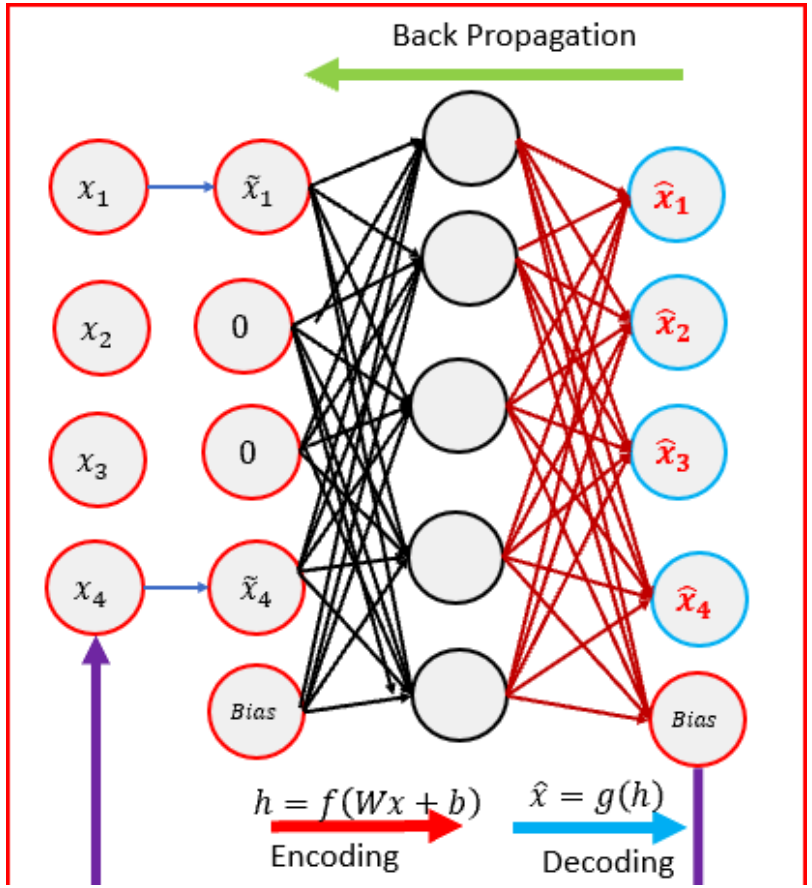
- Corruption of the input can be done randomly by making some of the input as zero. Remaining nodes copy the input to the noised input.



Denoising Autoencoders

- Denoising autoencoders must remove the corruption to generate an output that is similar to the input. Output is compared with input and not with noised input. To minimize the loss function we continue until convergence.

$$L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}}))),$$



Denoising Autoencoders

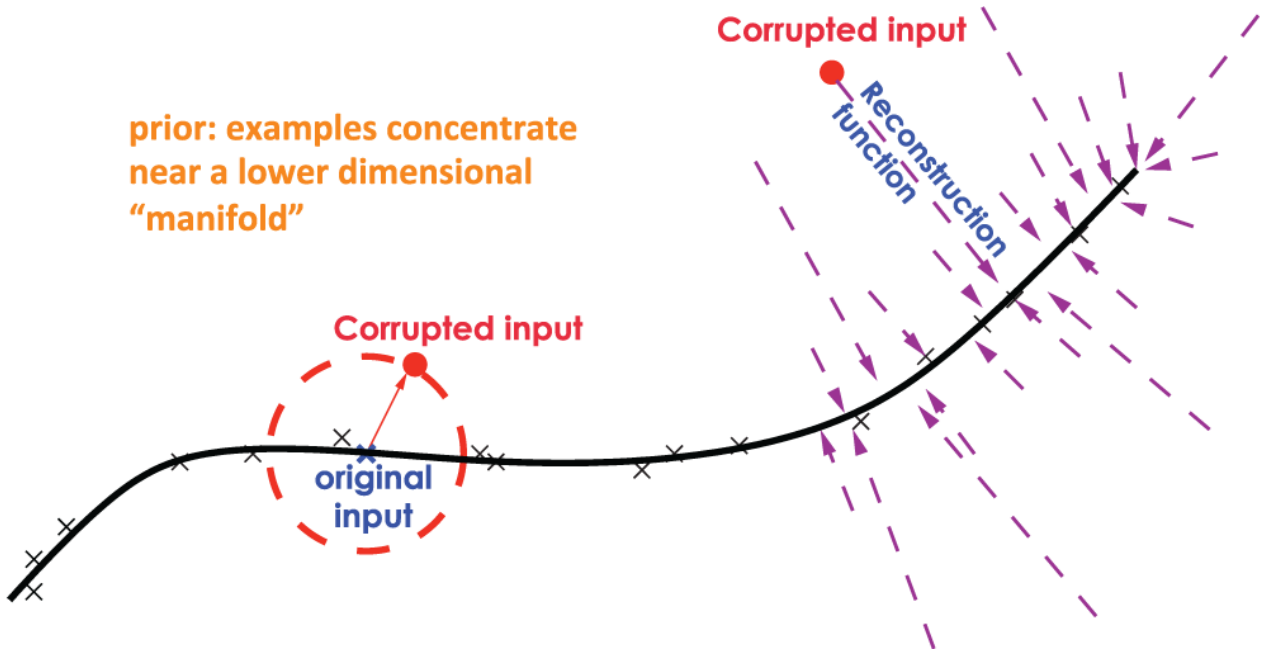
- Usually the corruption is applied through
  - Random assignment of subset of inputs to 0, with probability p.
  - Gaussian additive noise.

(a)

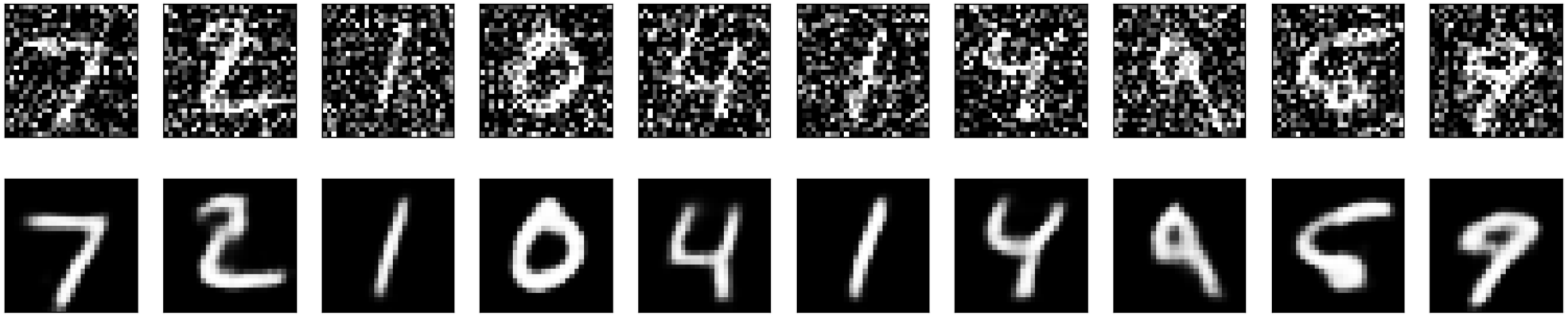
(b)

- The autoencoder cannot fully trust each feature of x independently so it must learn the correlations of x's features.
- Based on those relations we can predict a more 'not prune to changes' model.

Denoising Autoencoders



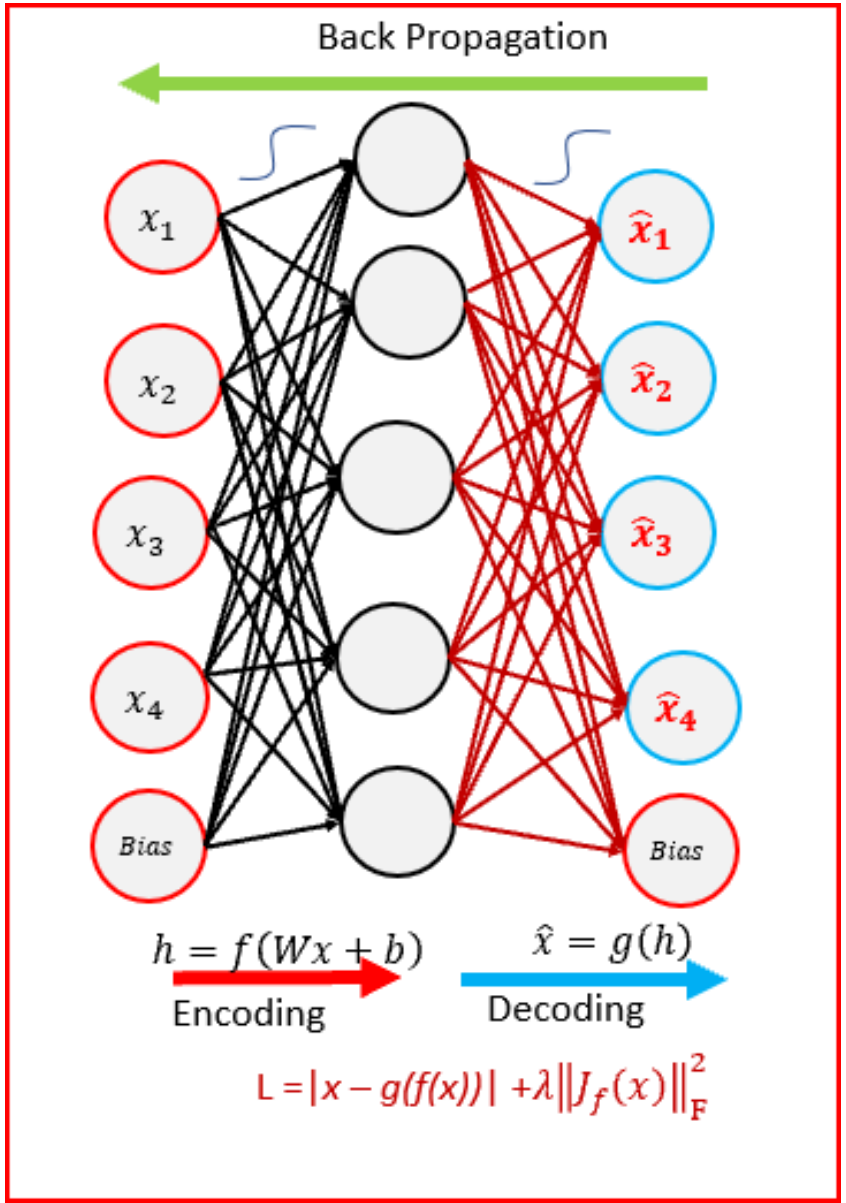
- Example on MNIST



Contractive Autoencoders(CAE)

- Contractive autoencoder(CAE) objective is to have a robust learned representation which is less sensitive to small variation in the data.

- Robustness of the representation for the data is done by applying a penalty term to the loss function, The penalty term is Frobenius norm of the Jacobian matrix of the hidden layer.



Contractive Autoencoders(CAE)

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h}, \boldsymbol{x}),$$

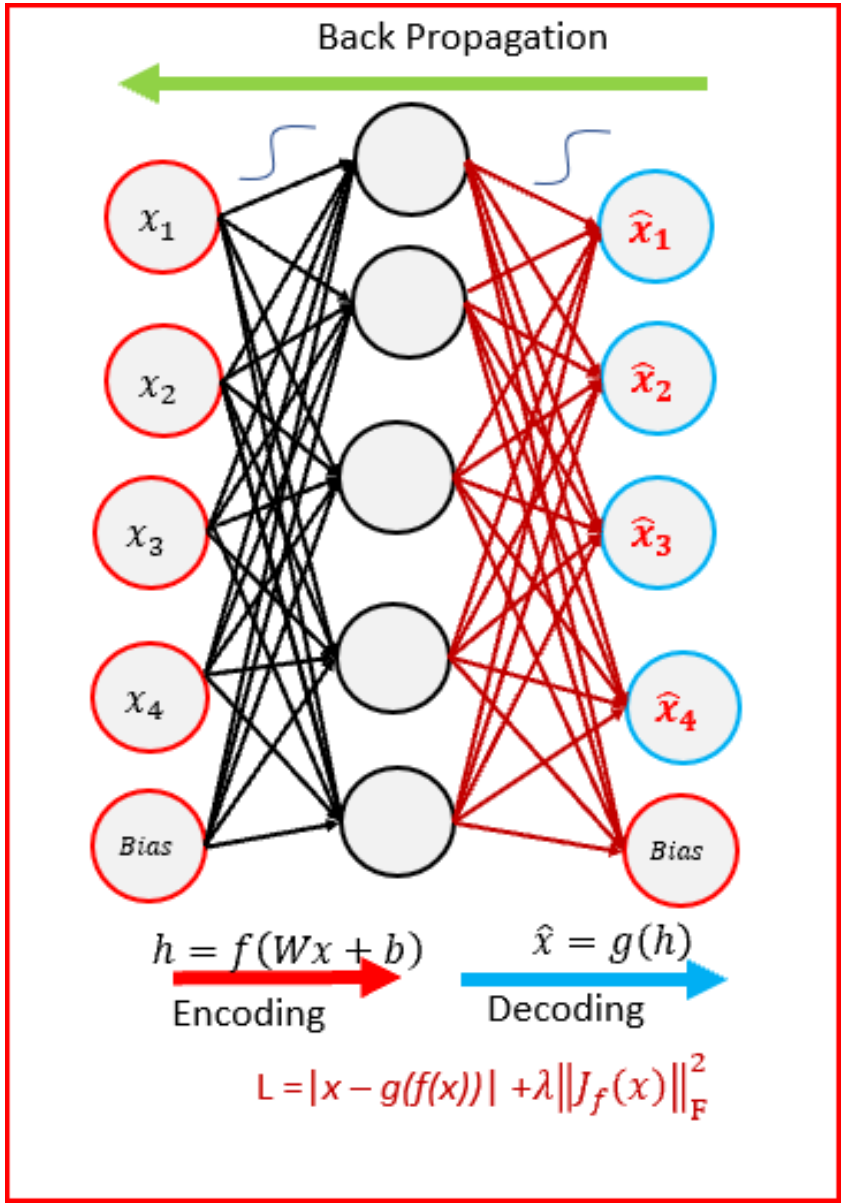
$$\Omega(\boldsymbol{h}, \boldsymbol{x}) = \lambda \sum_i ||\nabla_{\boldsymbol{x}} h_i||^2.$$

$$L = |x - g(f(x))| + \lambda \left\| J_f(x) \right\|_F^2$$

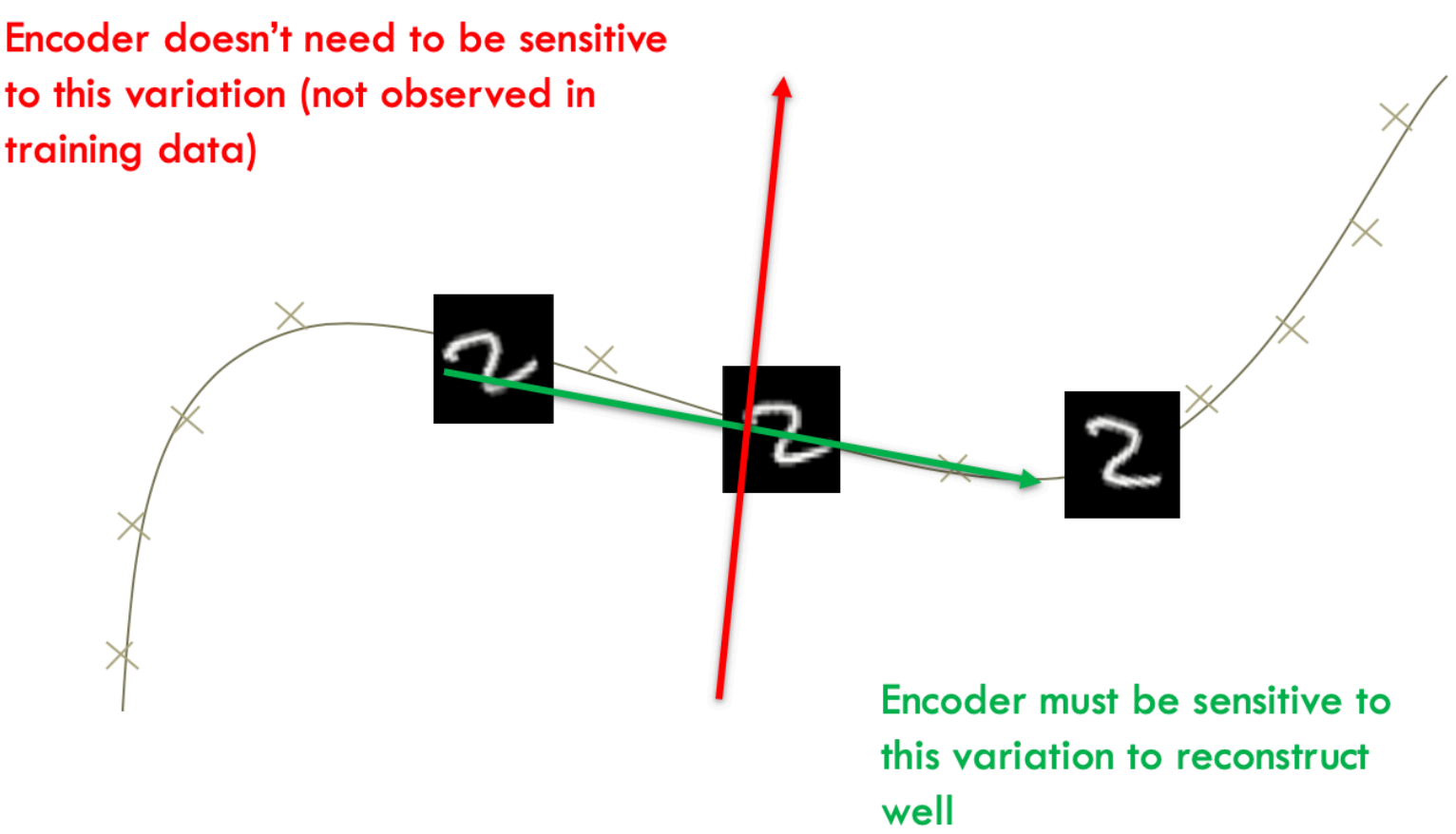
$$\left\| J_f(x) \right\|_F^2 = \sum_{ij} \left(\frac{\partial h_j(x)}{\partial x_i} \right)^2$$

- Contracting the input neighborhood to a smaller output neighborhood.

- Points close to each other in the input space maintain that property in the latent space.



Contractive Autoencoders(CAE)



DAE vs CAE

- DAE make the **reconstruction function** resist small, finite sized perturbations in input.
- CAE make the **feature encoding function** resist small, infinitesimal perturbations in input.

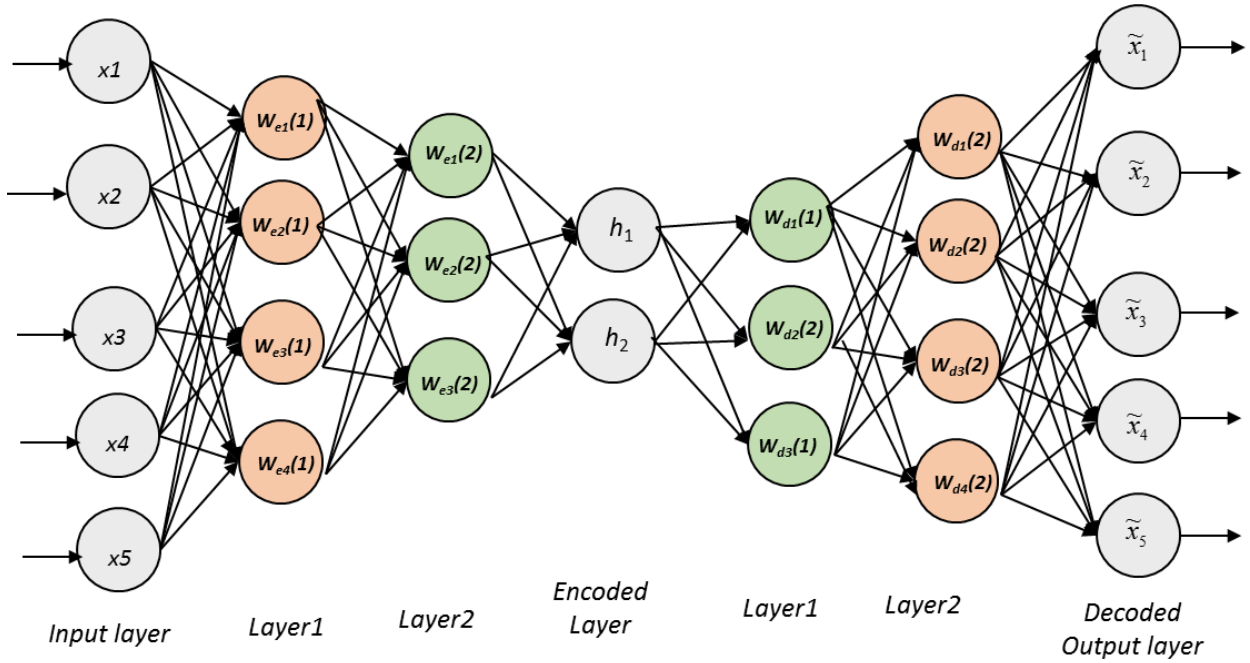
- Both denoising AE and contractive AE perform well!

- Advantage of DAE: simpler to implement
  - Requires adding one or two lines of code to regular AE.
  - No need to compute Jacobian of hidden layer.

- Advantage of CAE: gradient is deterministic.
  - might be more stable than DAE, which uses a sampled gradient.
  - one less hyper-parameter to tune (noise-factor)

Stacked Autoencoders

- A stacked autoencoder is a neural network consist several layers of sparse autoencoders where output of each hidden layer is connected to the input of the successive hidden layer.
- Each hidden layer is a more compact representation than the last hidden layer

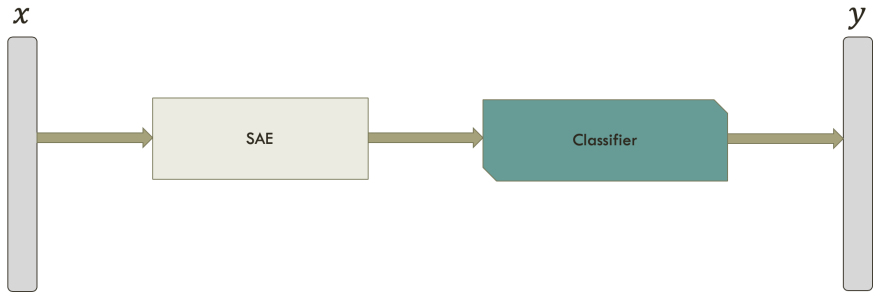


Stacked Autoencoders

- Motivation:
- We want to harness the feature extraction quality of an AE for our advantage, For example: we can build a deep supervised classifier where it's input is the output of a SAE.

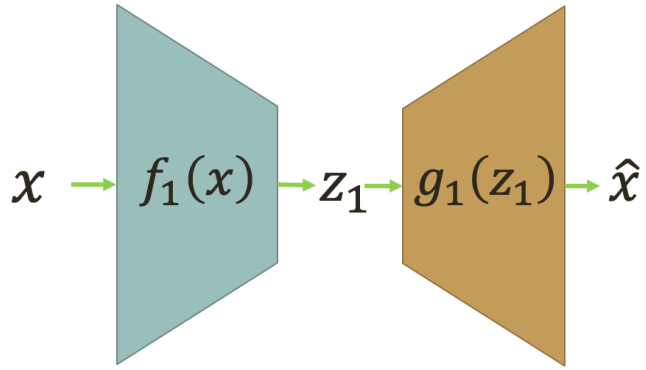
- It is used to train a classier with a specific context and find better accuracy than training with raw data.

- Building a SAE consists of two phases:
  - 1. Train each AE layer one after the other.
  - 2. Connect any classifier (SVM / FC NN layer etc.)

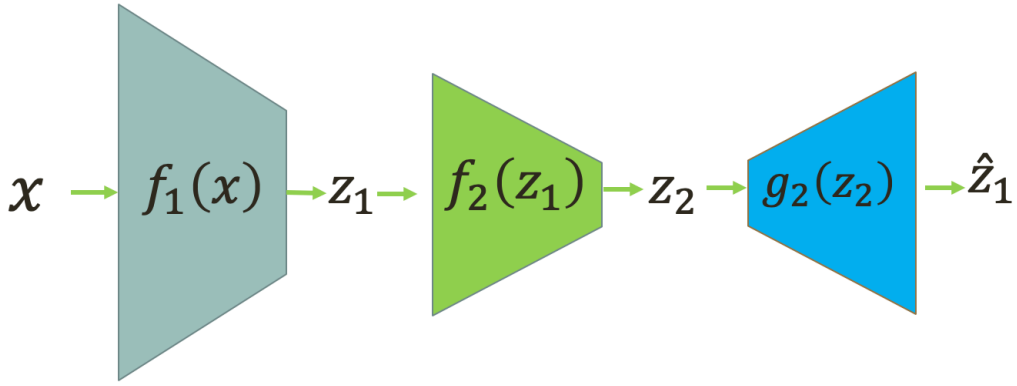


Stacked Autoencoders

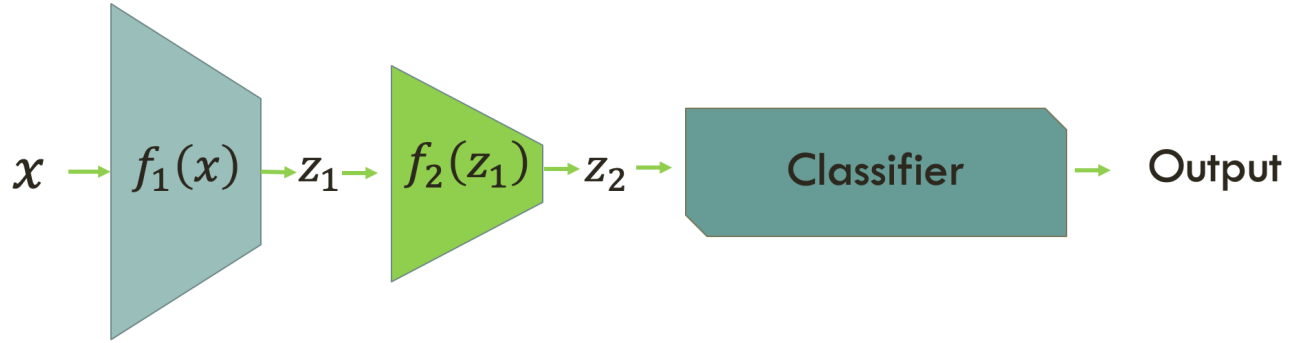
1. First train the first layer



2. Then we train the next layer

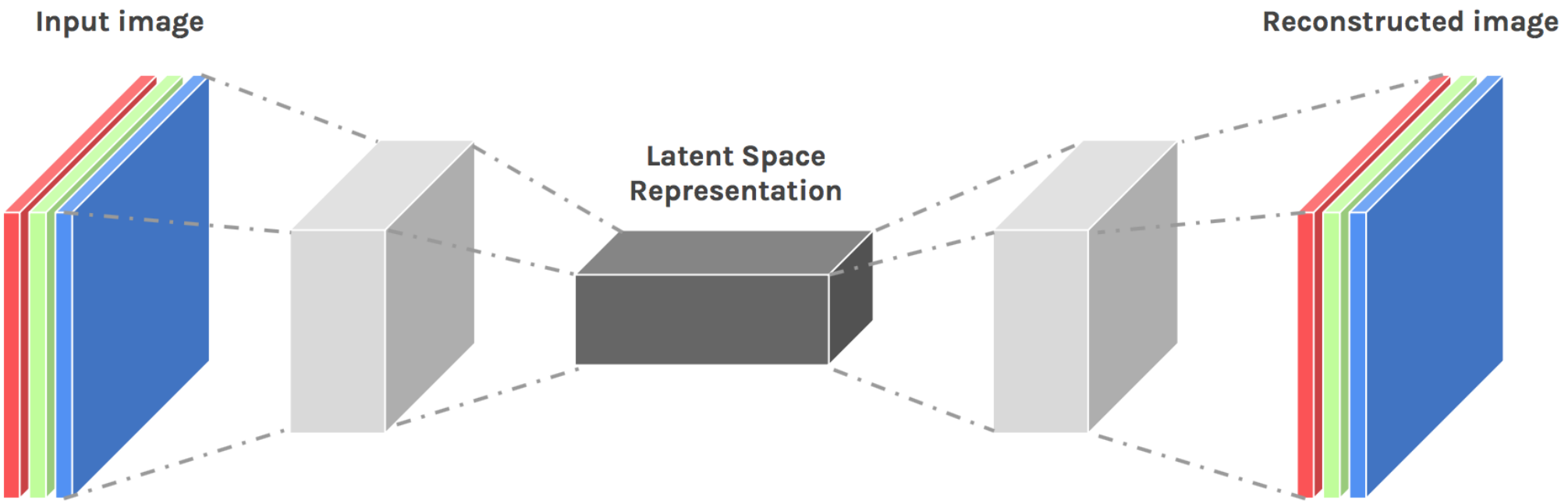


3. Then add a classifier

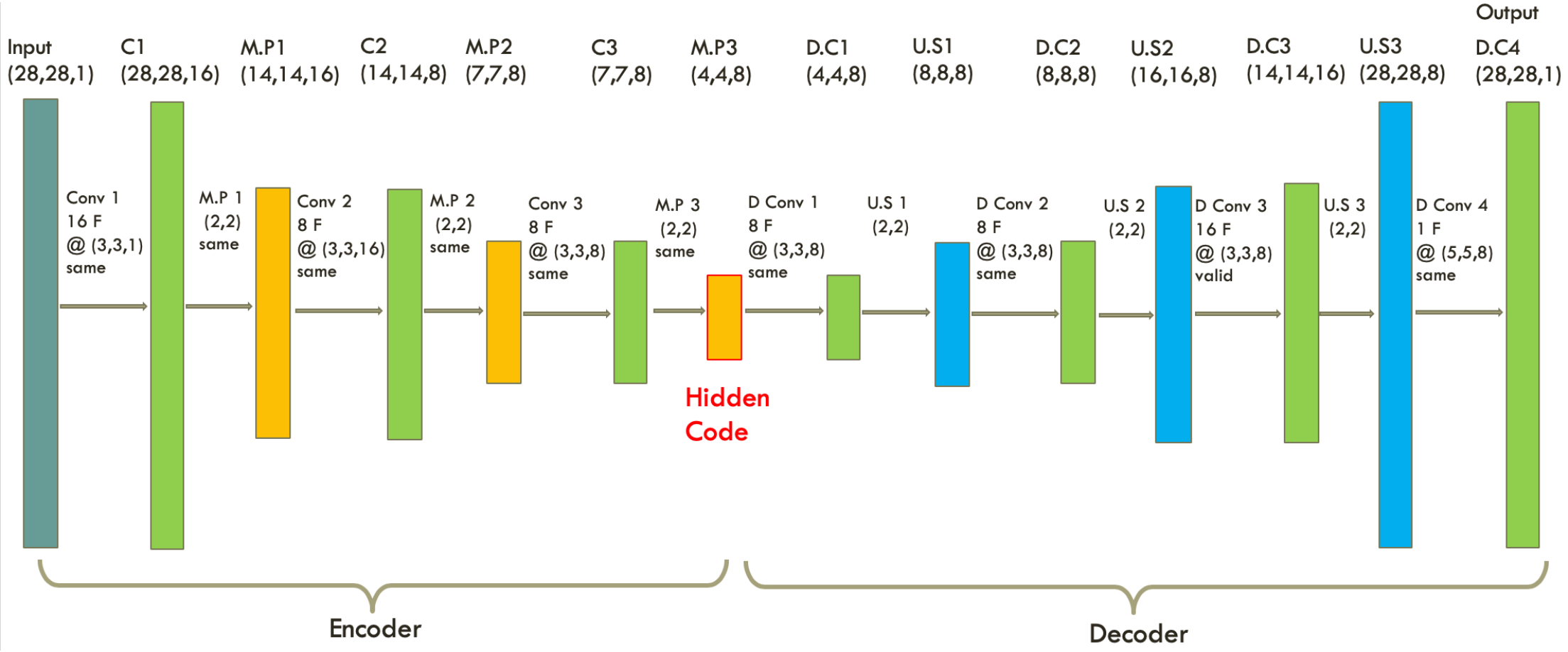


Make use of unlabeled data in supervised learning setting!

Convolutional Autoencoder



Convolutional Autoencoder

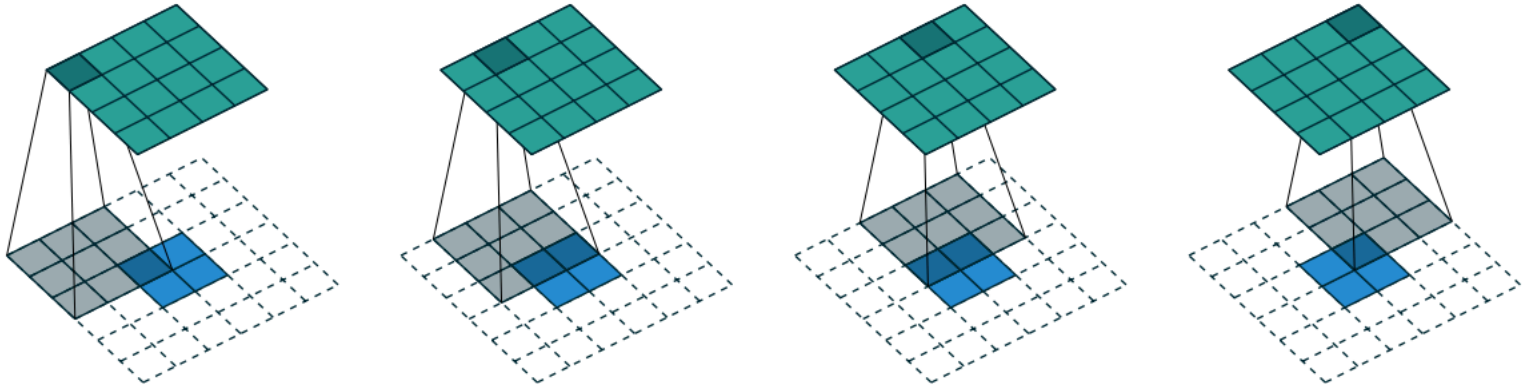


Convolutional Autoencoder

Transposed Convolution (emulated with direct convolution):

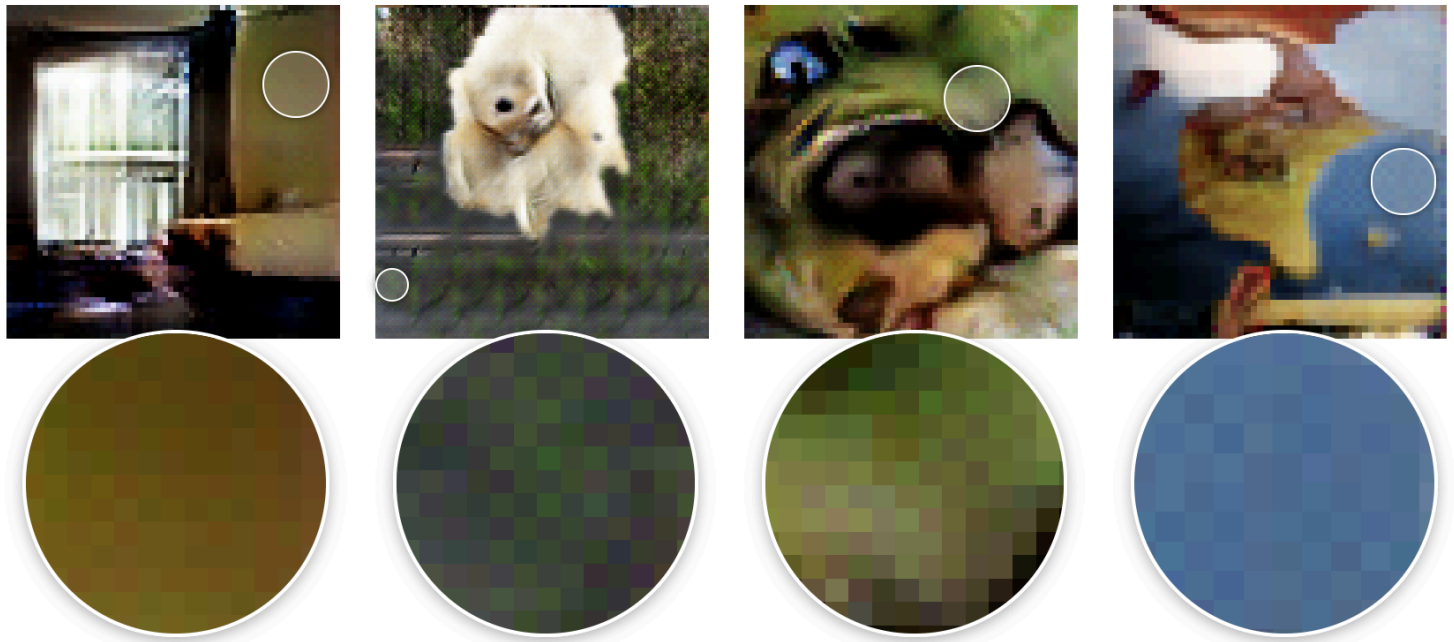
output

input



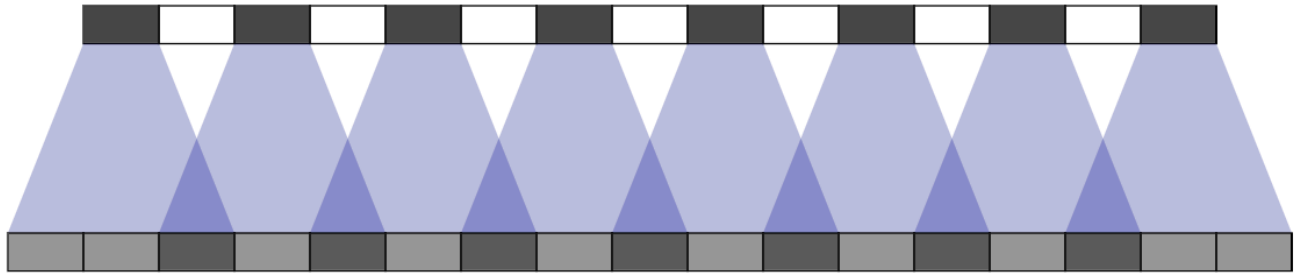
Transposed convolution and Checkerboard Artifacts

- When we look very closely at images generated by neural networks, we often see a strange checkerboard pattern of artifacts. It's more obvious in some cases than others, but a large fraction of recent models exhibit this behavior.

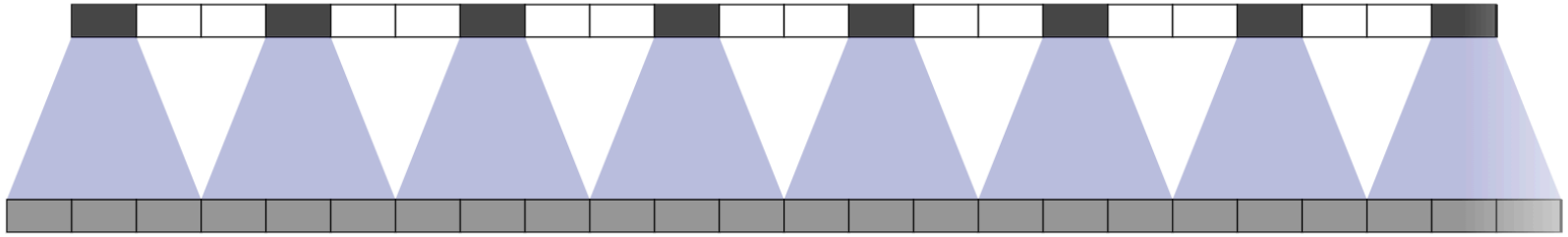


Transposed convolution and Checkerboard Artifacts

- Checkerboard artifacts result from uneven overlap, when the kernel size (the output window size) is not divisible by the stride
- Example: stride 2, and size 3



- Vs: stride of 3, and size of 3

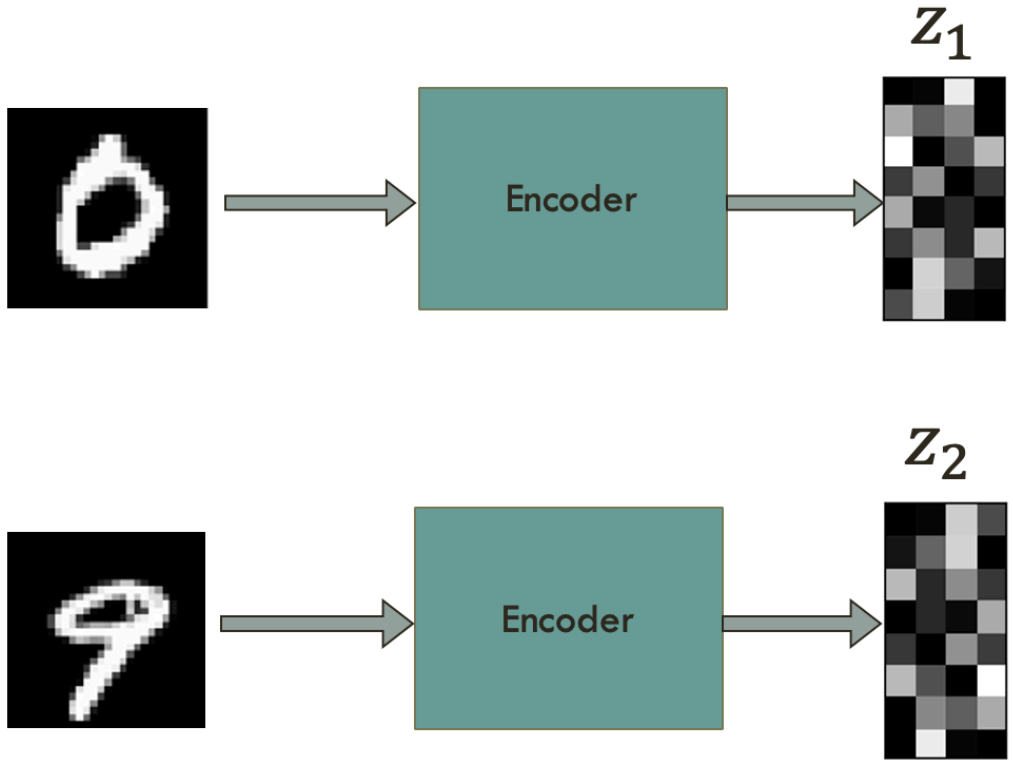


Transposed convolution and Checkerboard Artifacts

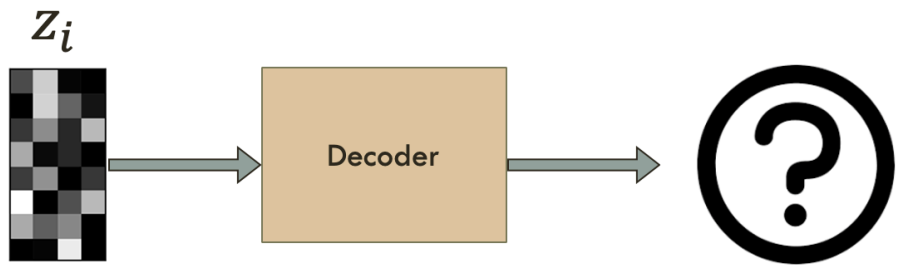
- One approach is to make sure you use a kernel size that is divided by your stride, avoiding the overlap issue “sub-pixel convolution”, a technique which has recently had success in image super-resolution

- Another approach (with better practical results) is to separate out upsampling to a higher resolution from convolution to compute features. For example, you might resize the image (using nearest-neighbor interpolation or bilinear interpolation) and then do a convolutional layer.

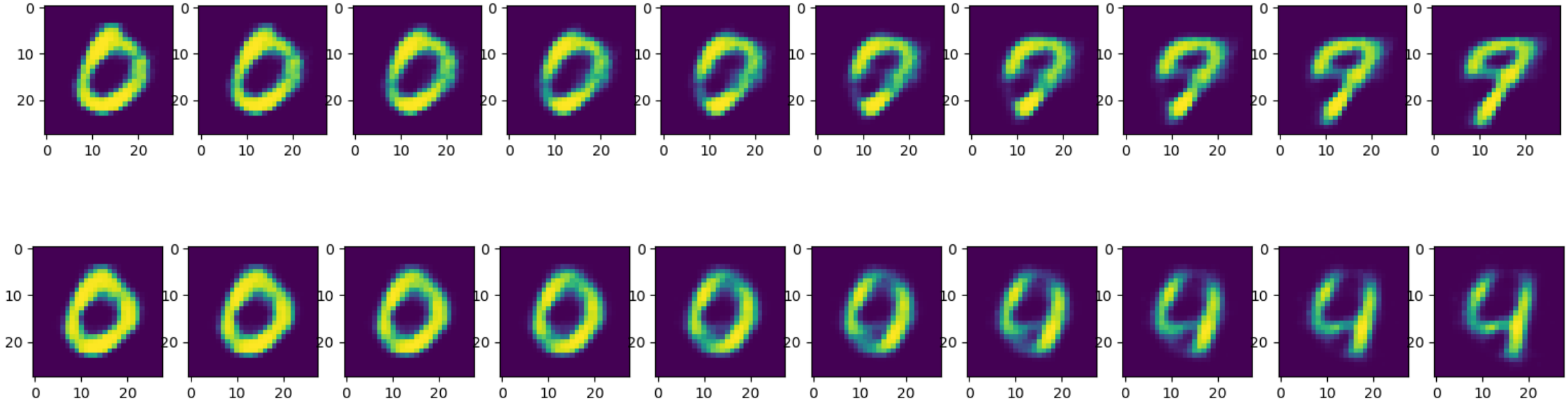
Latent space interpolation



$$z_i = \alpha z_1 + (1 - \alpha) z_2$$



Latent space interpolation



Applications of Autoencoders

- Dimensionality reduction
  - Lower-dimensional representations can improve performance on many tasks, such as classification. Models of smaller spaces consume less memory and runtime.

Applications of Autoencoders

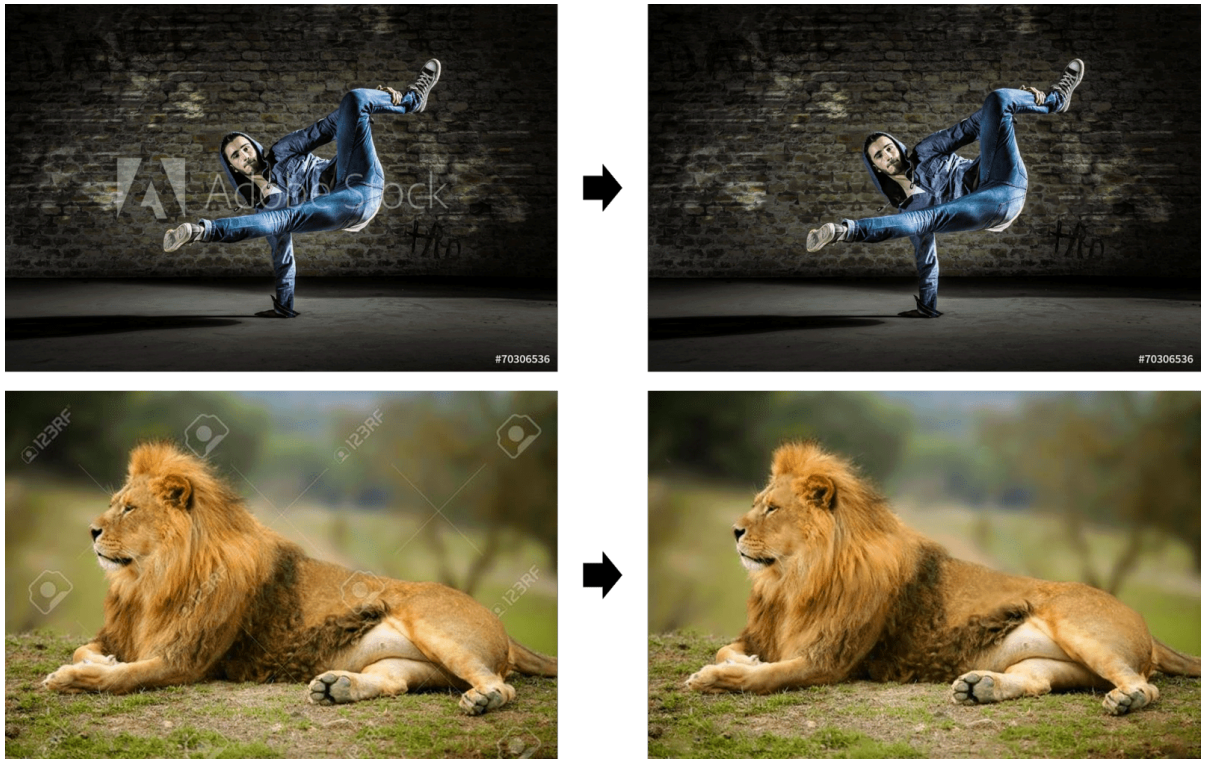
- Information retrieval
  - the task of finding entries in a database that resemble a query entry.
  - If we train the dimensionality reduction algorithm to produce a code that is low- dimensional and binary, then we can store all database entries in a hash table mapping binary code vectors to entries.
  - This hash table allows us to perform information retrieval by returning all database entries that have the same binary code as the query.
  - We can also search over slightly less similar entries very efficiently, just by flipping individual bits from the encoding of the query.

Applications of Autoencoders

- Semantic hashing
  - This approach to information retrieval via dimensionality reduction and binarization is called. **semantic hashing**
  - **DEDUPLICATION** is one of the application of semantic hashing i.e. hash collisions -> duplicates.
  - To produce binary codes we can:
    - Use thresholding of am activation function
    - Train a sigmoid to produce 1 or 0, inject additive noise just before the sigmoid nonlinearity during training. The magnitude of the noise should increase over time (avoid vanishing gradients), To fight that noise the network must increase the magnitude of the inputs to the sigmoid function, until saturation occurs.

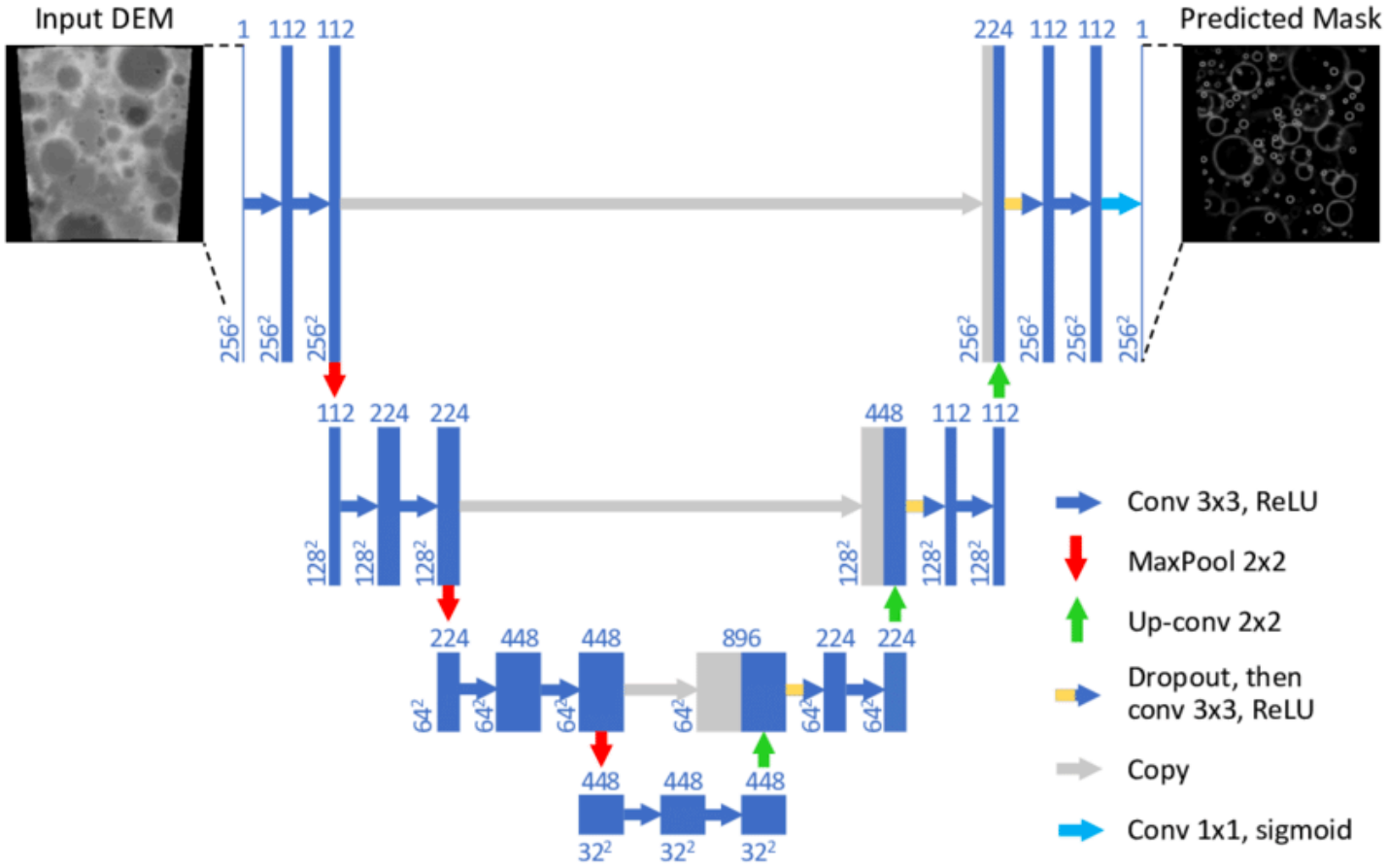
Applications of Autoencoders

- Neural inpainting



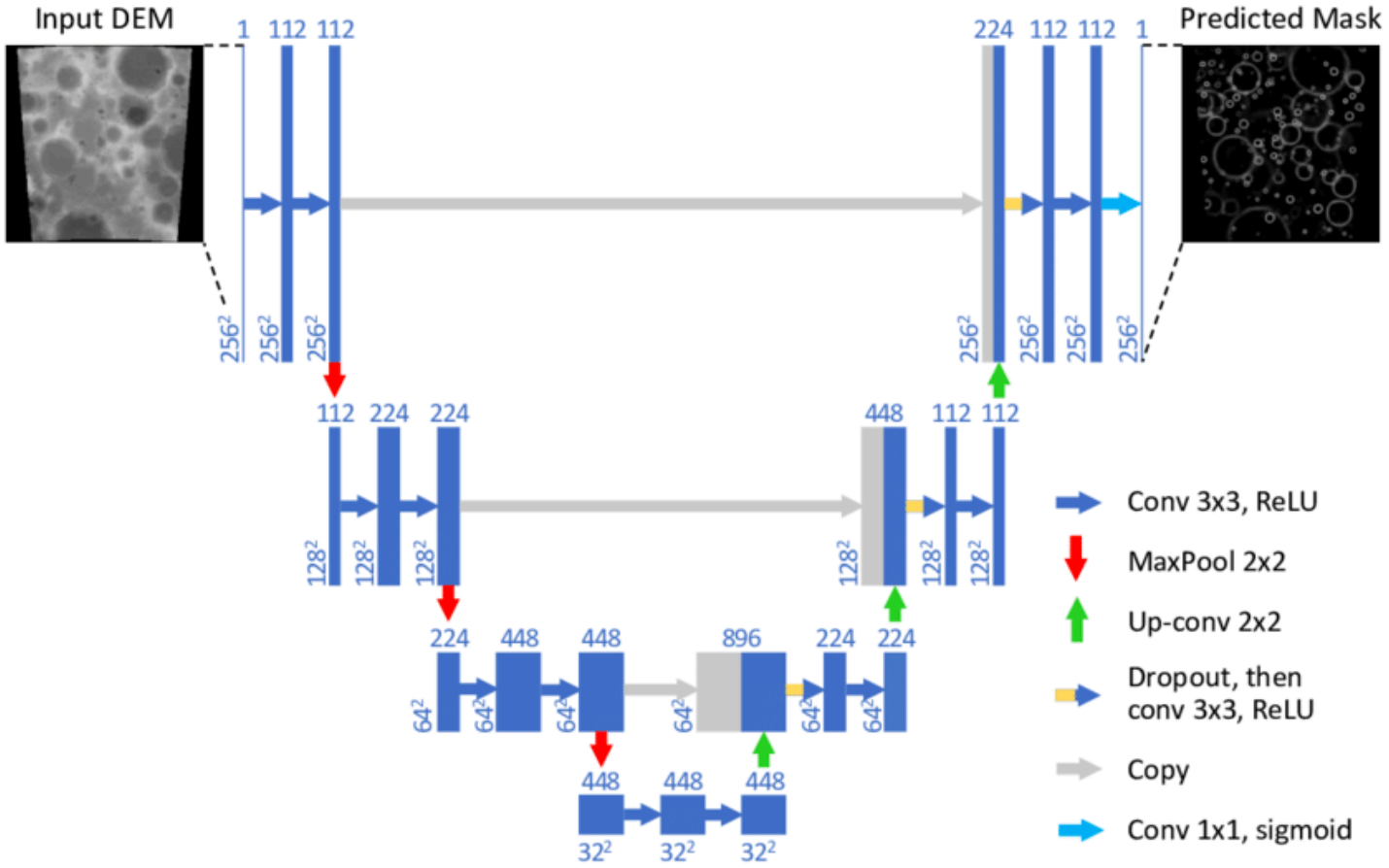
Applications of Autoencoders

- Image segmentation



Applications of Autoencoders

- Image segmentation



References

- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville.*Deeplearning*.MITpress,2016.
- AUGUSTUS ODENA, VINCENT DUMOULIN, CHRIS OLAH, Deconvolution and Checkerboard Artifacts.
- 6S191_MIT_DeepLearning, deep generative modeling.
- Guy Golan, Autoencoders.
- “A guide to convolution arithmetic for deep learning.” Vincent Dumoulin, Francesco Visin.
- Renu Khandelwal, “Deep Learning — Different Types of Autoencoders”.