Autoencoders

Motivation, Architecture, Variations

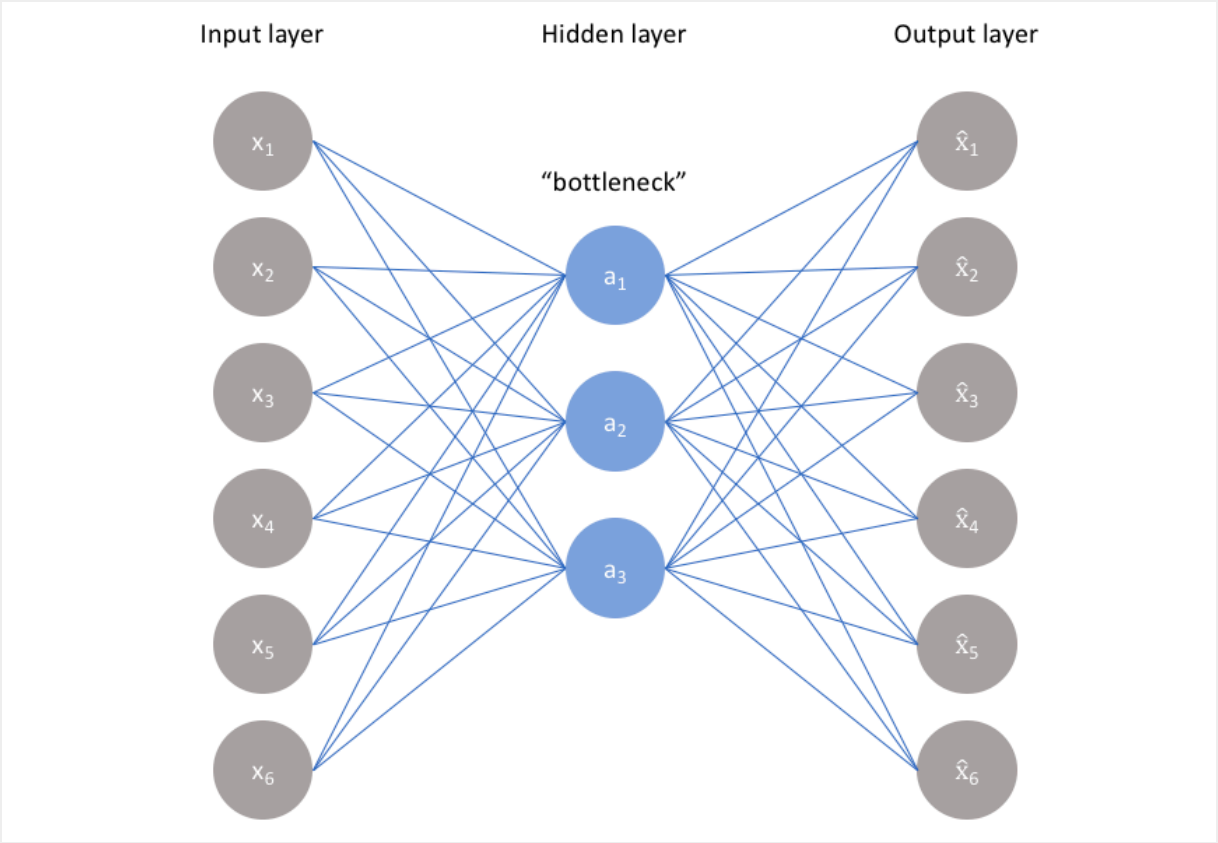
AUA, DS330 Deep Learning 2021, Hovhannes Tamoyan

Motivation

So far we have been discussing supervised methods.

The goal is to 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 the input features were each independent of one another, this compression and subsequent reconstruction would be a very difficult task. However, if some sort of structure exists in the data (ie. correlations between input features), this structure can be learned and consequently leveraged when forcing the input through the network's bottleneck.


Input layer
Hidden layer
Output layer
"bottleneck"
x_1
x_2
x_3
x_4
x_5
x_6
a_1
a_2
a_3
$\hat{x}_1$
$\hat{x}_2$
$\hat{x}_3$
$\hat{x}_4$
$\hat{x}_5$
$\hat{x}_6$

Autoencoders

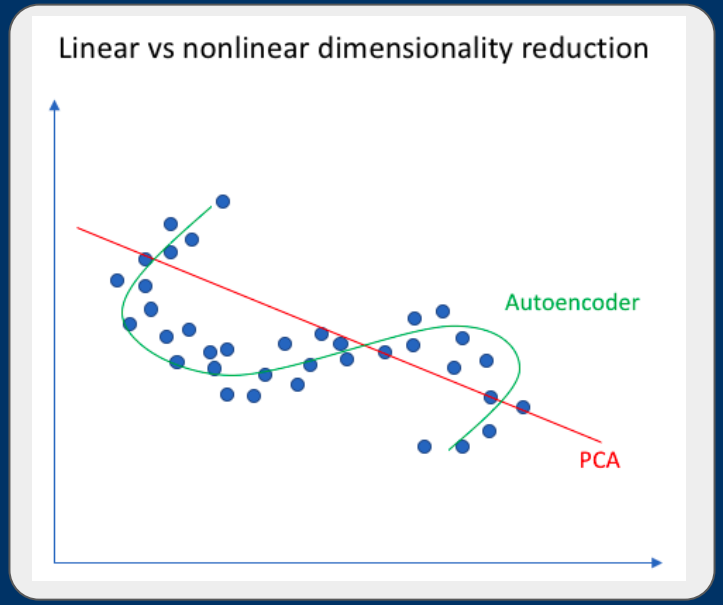
The objective function of AE is

L(x, x_hat)

The bottleneck is a key attribute of our network design; without the presence of an information bottleneck, our network could easily learn to simply memorize the input values by passing these values along through the network.

Autoencoders and PCA

Because neural networks are capable of learning nonlinear relationships, this can be thought of as a more powerful (nonlinear) generalization of PCA.

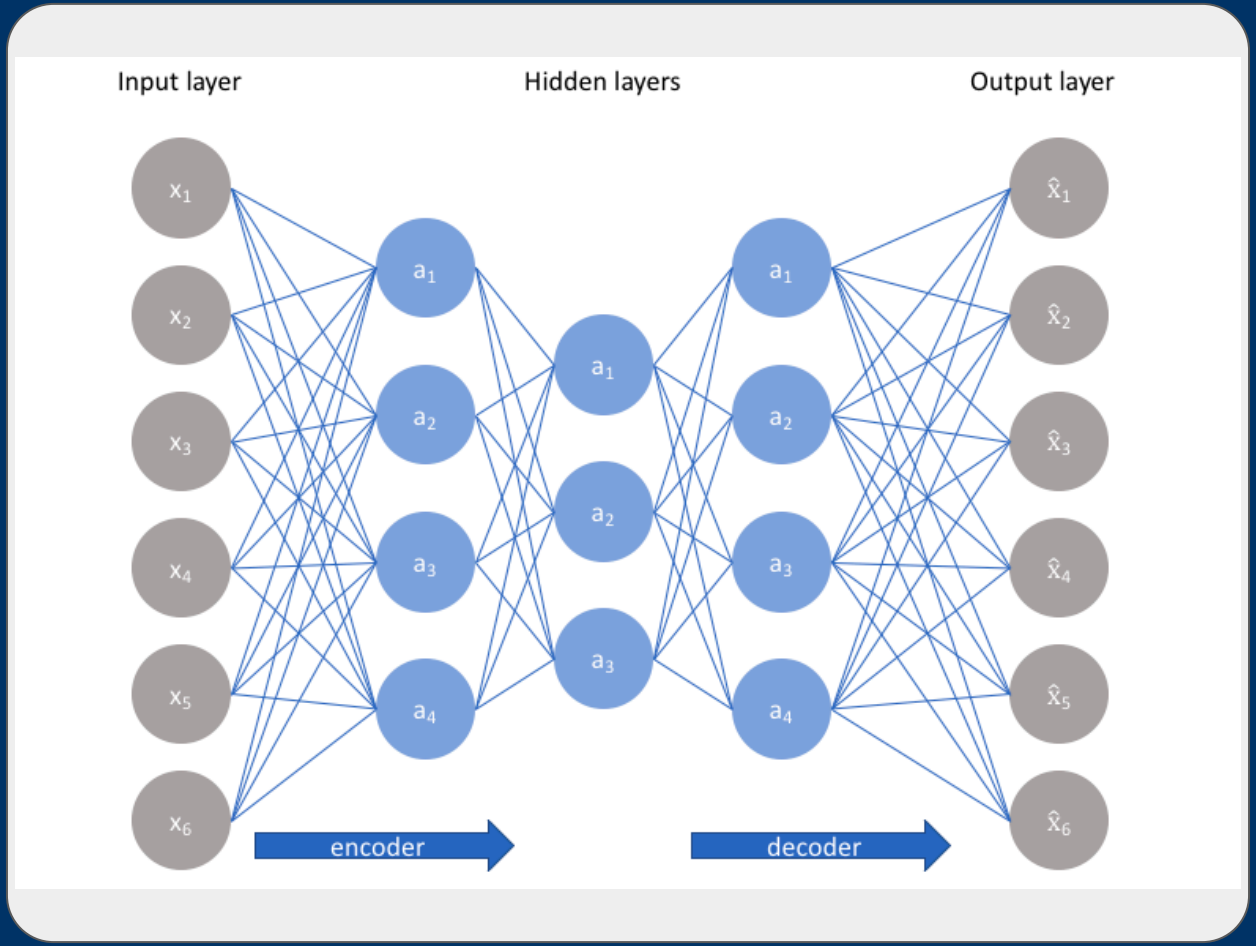


Undercomplete Autoencoder

The simplest architecture for constructing an autoencoder is to constrain the number of nodes present in the hidden layer(s) of the network, limiting the amount of information that can flow through the network.

By penalizing the network according to the reconstruction error, our model can learn the most important attributes of the input data and how to best reconstruct the original input from an "encoded" state.

Ideally, this encoding will learn and describe latent attributes of the input data.

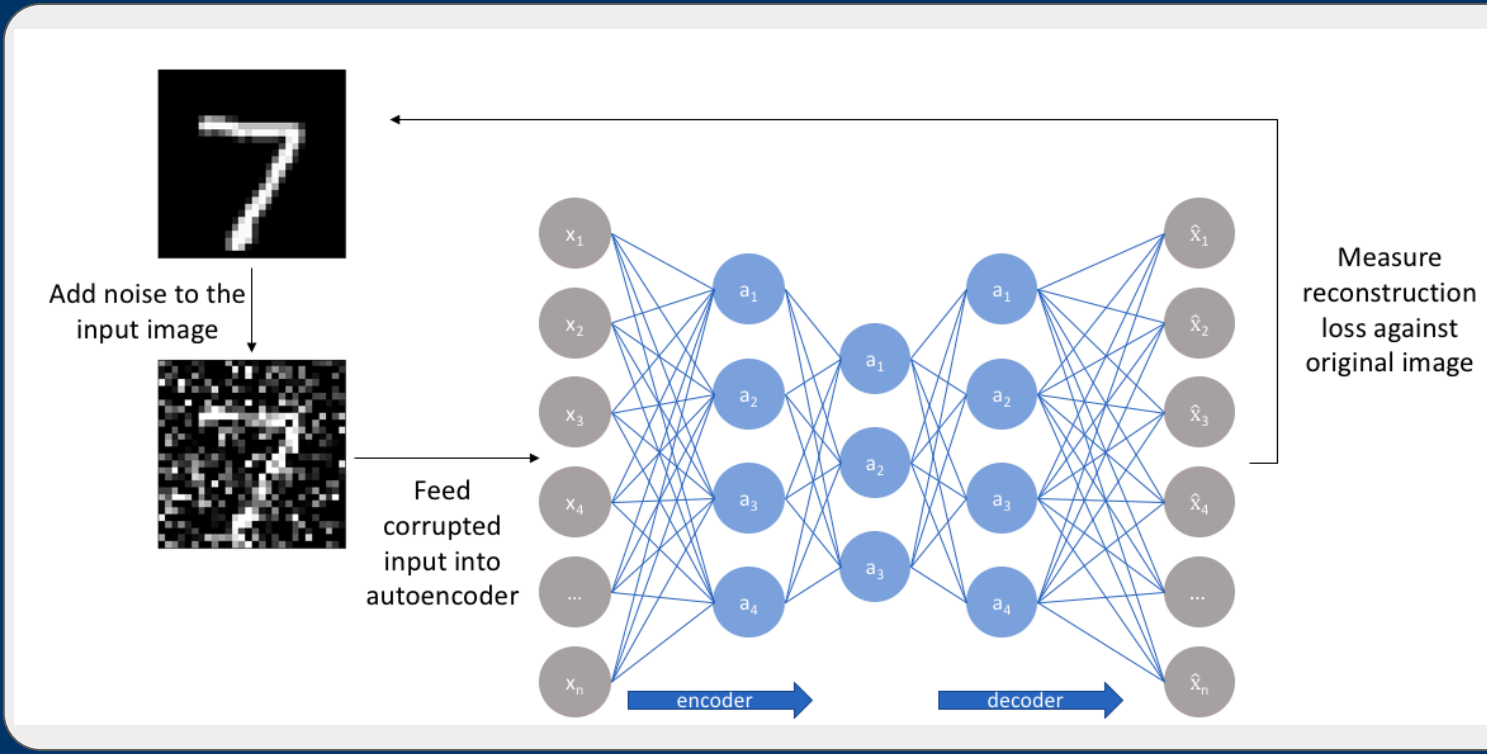

Input layer
Hidden layers
Output layer
x_1
x_2
x_3
x_4
x_5
x_6
a_1
a_2
a_3
a_4
a_1
a_2
a_3
a_1
a_2
a_3
a_4
$\hat{x}_1$
$\hat{x}_2$
$\hat{x}_3$
$\hat{x}_4$
$\hat{x}_5$
$\hat{x}_6$
encoder
decoder

Denoising Autoencoders

So far we have discussed the concept of training a neural network where the input and outputs are identical and our model is tasked with reproducing the input as closely as possible while passing through some sort of information bottleneck.

Another approach towards developing a generalizable model is to slightly corrupt the input data but still maintain the uncorrupted data as our target output.

With this approach, our model isn't able to simply develop a mapping which memorizes the training data because our input and target output are no longer the same.


Add noise to the input image
Feed corrupted input into autoencoder
Measure reconstruction loss against original image
encoder
decoder

Contractive autoencoders

One would expect that for very similar inputs, the learned encoding would also be very similar.

We can explicitly train our model in order for this to be the case by requiring that the derivative of the hidden layer activations are small with respect to the input.

In other words, for small changes to the input, we should still maintain a very similar encoded state.

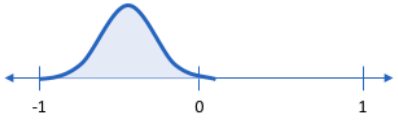
Denoising autoencoders make the reconstruction function (ie. decoder) resist small but finite-sized perturbations of the input, while contractive autoencoders make the feature extraction function (ie. encoder) resist infinitesimal perturbations of the input.

Variational Autoencoders

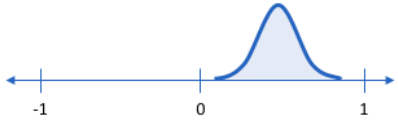
A variational autoencoder (VAE) provides a probabilistic manner for describing an observation in latent space. Thus, rather than building an encoder which outputs a single value to describe each latent state attribute, we'll formulate our encoder to describe a probability distribution for each latent attribute.

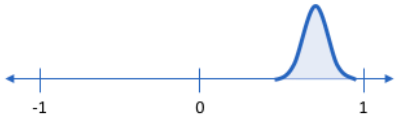
Smile (discrete value)

Smile (probability distribution)



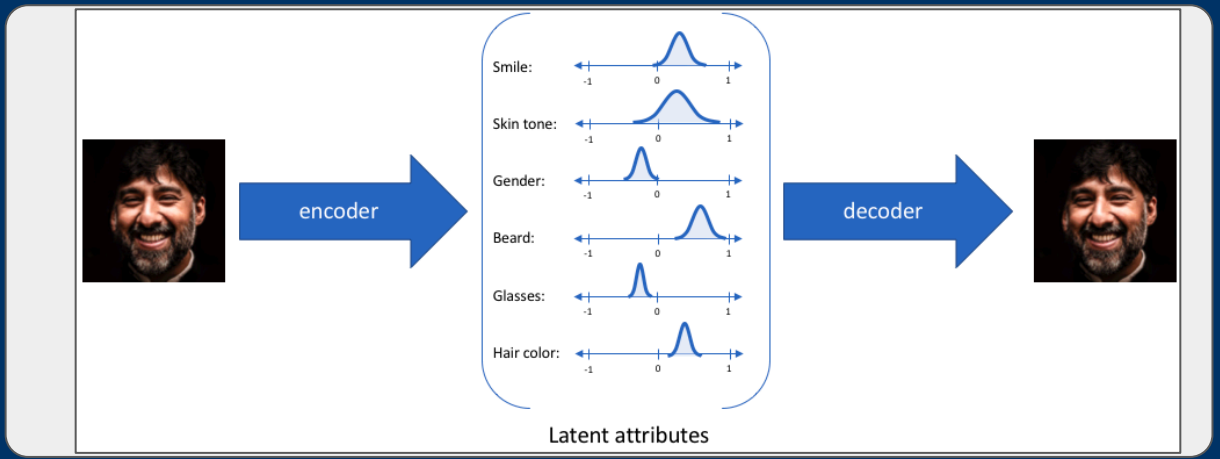
vs.





Variational Autoencoders

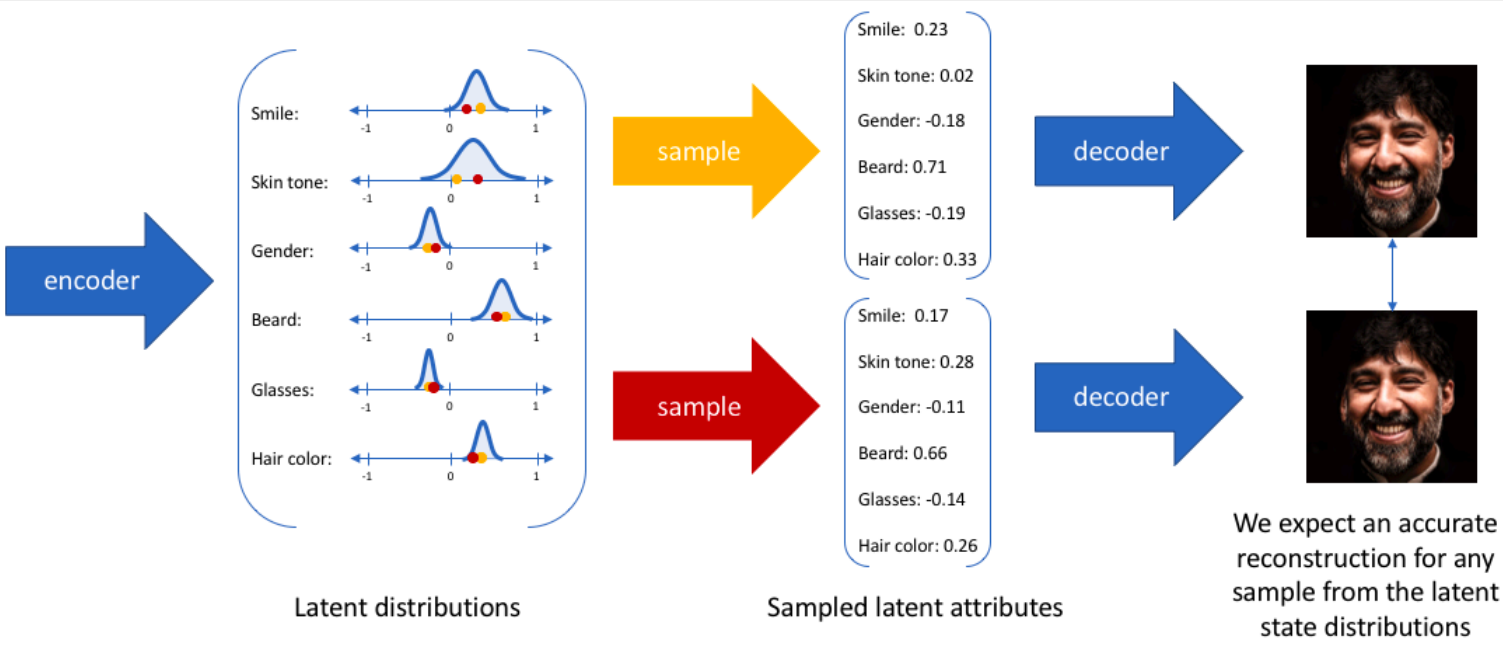
With this approach, we'll now represent each latent attribute for a given input as a probability distribution. When decoding from the latent state, we'll randomly sample from each latent state distribution to generate a vector as input for our decoder model.



Variational Autoencoders

By constructing our encoder model to output a range of possible values (a statistical distribution) from which we'll randomly sample to feed into our decoder model, we're essentially enforcing a continuous, smooth latent space representation.

For any sampling of the latent distributions, we're expecting our decoder model to be able to accurately reconstruct the input. Thus, values which are nearby to one another in latent space should correspond with very similar reconstructions.


encoder
Smile:
Skin tone:
Gender:
Beard:
Glasses:
Hair color:
-1
0
1
sample
sample
Smile: 0.23
Skin tone: 0.02
Gender: -0.18
Beard: 0.71
Glasses: -0.19
Hair color: 0.33
Smile: 0.17
Skin tone: 0.28
Gender: -0.11
Beard: 0.66
Glasses: -0.14
Hair color: 0.26
decoder
decoder
Latent distributions
Sampled latent attributes
We expect an accurate reconstruction for any sample from the latent state distributions

Thank You