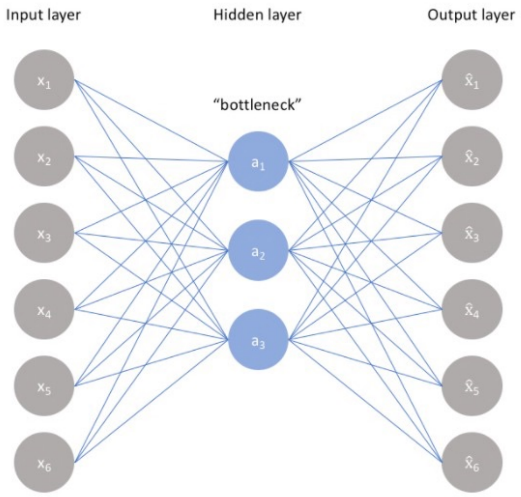
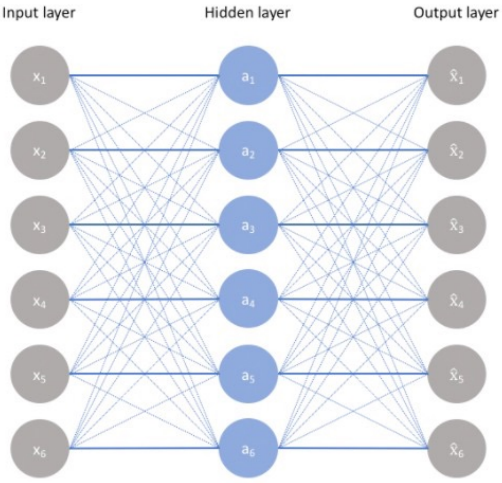
Autoencoders

main concepts:

1. trained to reconstruct their input (unsupervised)
2. common structure (often symmetrical):
   - encoder: transforms input into internal representation
   - decoder: reconstructs input from representation
3. representational bottleneck
   - structure (under-complete)
   - regularization (denoising, sparse, contractive, …)



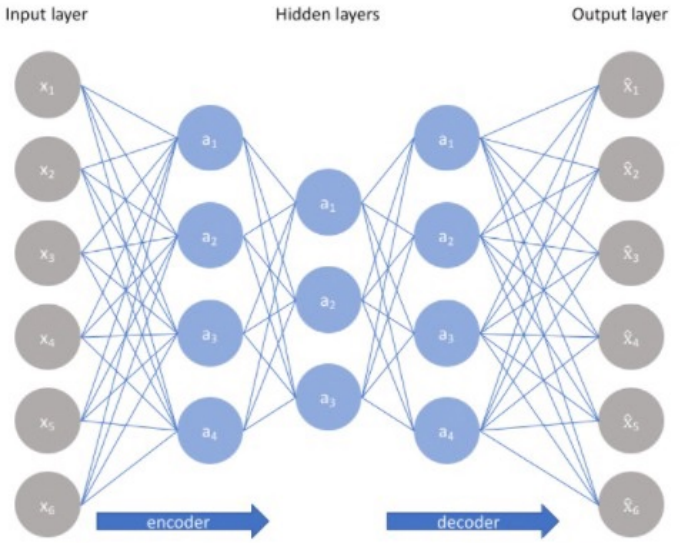
As visualized above, we can take an unlabeled dataset and frame it as a supervised learning problem tasked with outputting $\hat{x}$, a **reconstruction of the original input** x. This network can be trained by minimizing the *reconstruction error*, $\mathcal{L}(x, \hat{x})$, which measures the differences between our original input and the consequent reconstruction. 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 (visualized below).



A bottleneck constrains the amount of information that can traverse the full network, forcing a learned compression of the input data.

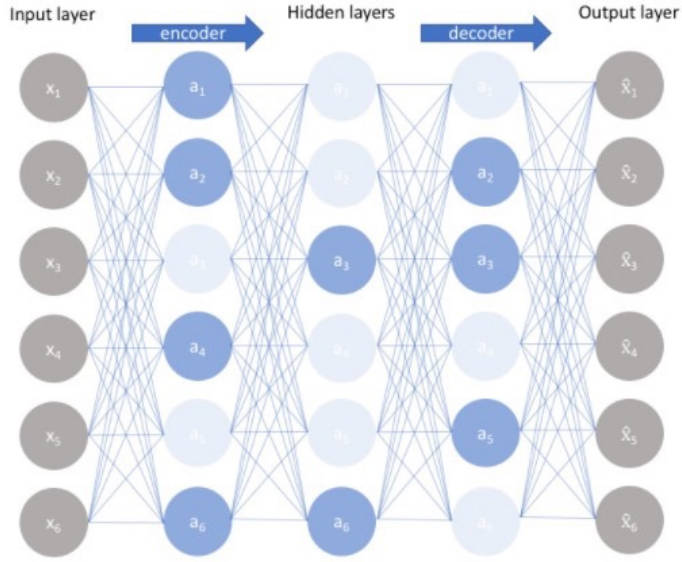
Undercomplete autoencoder

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**.



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. Rather, we'll construct our loss function such that we penalize *activations* within a layer. 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. It's worth noting that this is a different approach towards regularization, as we normally regularize the *weights* of a network, not the activations.



There are two main ways by which we can impose this sparsity constraint; both involve measuring the hidden layer activations for each training batch and adding some term to the loss function in order to penalize excessive activations. These terms are:

- **L1 Regularization**: We can add a term to our loss function that penalizes the absolute value of the vector of activations a in layer h for observation i, scaled by a tuning parameter λ.

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

- **KL-Divergence**: In essence, KL-divergence is a measure of the difference between two probability distributions. We can define a sparsity parameter ρ which denotes the average activation of a neuron over a collection of samples. This