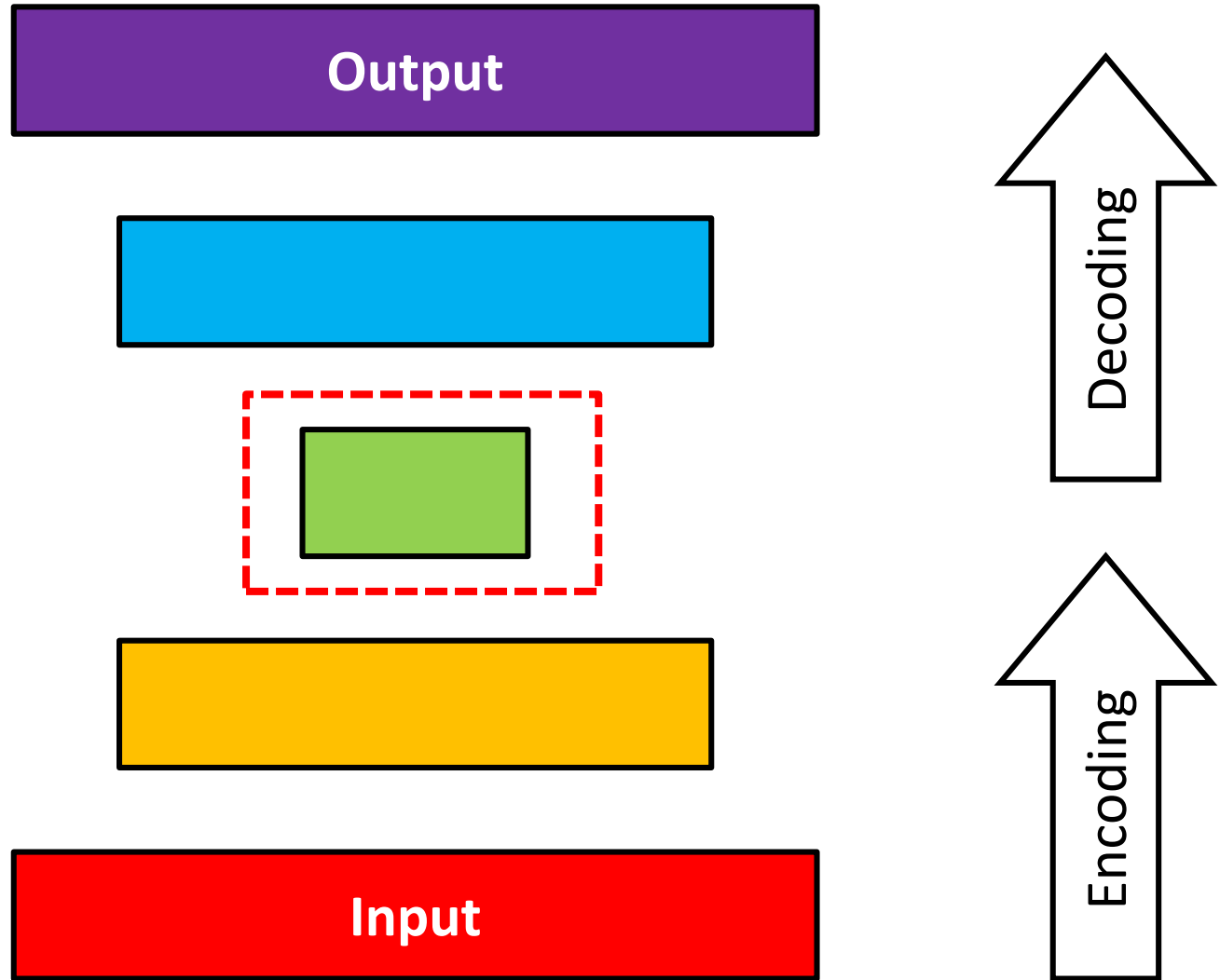


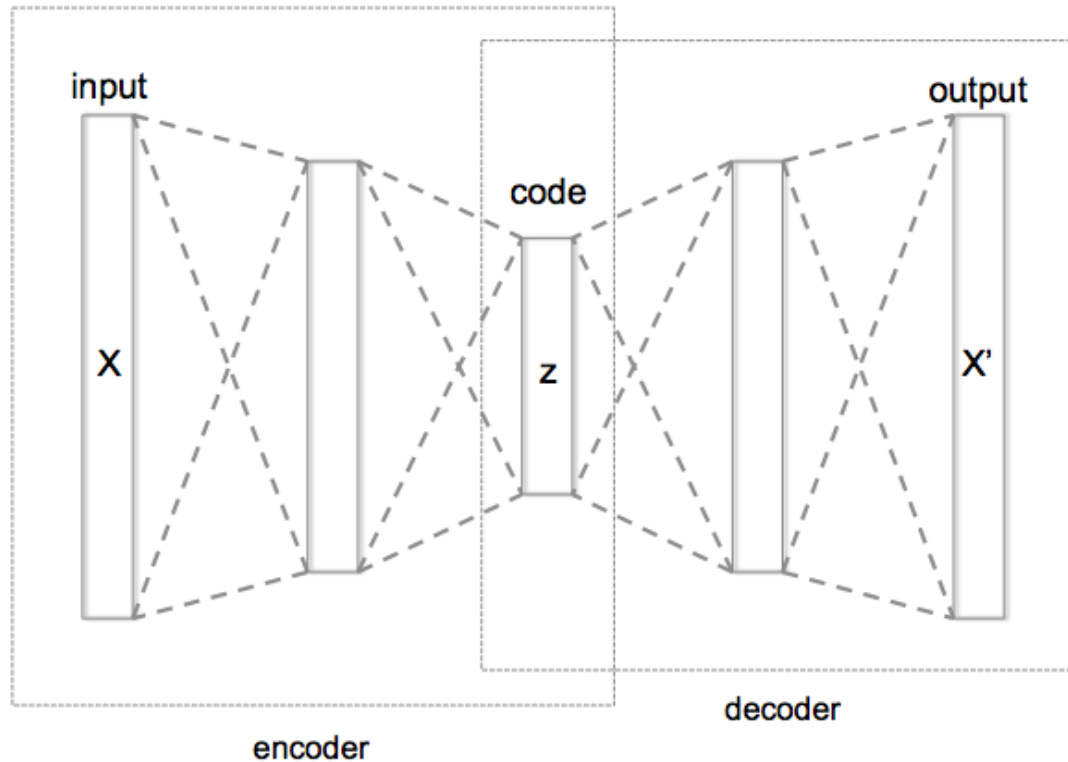
Autoencoders

Auto-Encoder?



Auto-Encoder?

Encoder + Decoder Structure



- Supervised Learning
- we discuss some of the speculative approaches to reducing the amount of labeled data necessary for existing models to work well and be applicable across a broader range of tasks. Accomplishing these goals usually requires some form of unsupervised or semi-supervised learning

Linear Factor Models

- Many of the research frontiers in deep learning involve building a probabilistic model of the input, $p_{\text{model}}(x)$
- Many of these models also have latent variables h , with $p_{\text{model}}(x) = E_h p_{\text{model}}(x|h)$.
- These latent variables provide another means of representing the data.
- some of the simplest probabilistic models with They
- also show many of the basic approaches necessary to build generative models that
- the more advanced deep models will extend further latent variables: linear factor models. These models are sometimes used as building blocks of mixture models or larger, deep probabilistic models.

- A linear factor model is defined by the use of a stochastic, linear decoder function that generates x by adding noise to a linear transformation of h .
- A linear factor model describes the data generation process as follows. First, we sample the explanatory factors h from a distribution

$$h \sim p(h)$$

$p(h)$ is a factorial distribution with

$$p(h) = \prod_i p(h_i)$$

Next we sample the real-valued observable variables given the factors

$$x = Wh + b + \textit{noise}$$

where the noise is typically Gaussian and diagonal (independent across dimensions)

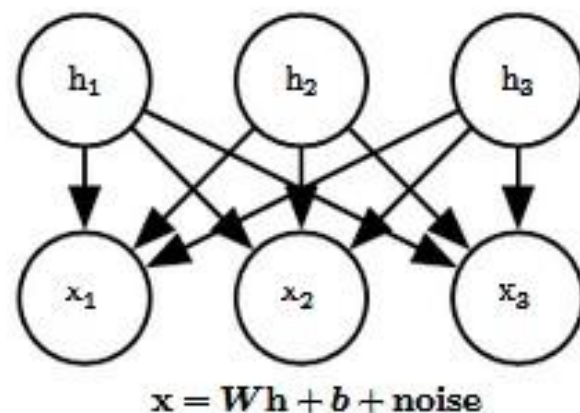


Figure 13.1: The directed graphical model describing the linear factor model family, in which we assume that an observed data vector \mathbf{x} is obtained by a linear combination of independent latent factors \mathbf{h} , plus some noise. Different models, such as probabilistic PCA, factor analysis or ICA, make different choices about the form of the noise and of the prior $p(\mathbf{h})$.

Probabilistic PCA and Factor Analysis

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
- Hidden layer h
- Two parts
 - Encoder $h=f(x)$
 - Decoder $r=g(h)$
- Modern autoencoders also generalized to stochastic mappings

$$p_{encoder}(h|x), p_{decoder}(x|h)$$

- Traditionally, autoencoders were used for dimensionality reduction or feature learning.
- Recently, theoretical connections between autoencoders and latent variable models have brought autoencoders to the forefront of generative modeling
- Autoencoders may be thought of as being
- a special case of feedforward networks, and may be trained with all of the same techniques, typically minibatch gradient descent following gradients computed by back-propagation

Undercomplete autoencoder

- One way to obtain useful features from the autoencoder is to constrain h to have smaller dimension than x
- Learning an undercomplete representation forces the autoencoder to capture the most salient features of the training data.
- The learning process is described simply as minimizing a loss function

$$L(x, g(f(x)))$$

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

- Autoencoders with nonlinear encoder functions f and nonlinear decoder functions g can thus learn a more powerful nonlinear generalization of PCA
- if the encoder and decoder are allowed too much capacity, the autoencoder can learn to perform the copying task without extracting useful information about the distribution of the data

Regularized autoencoders

- Rather than limiting the model capacity by keeping the encoder and decoder shallow and the code size small, regularized autoencoders use a loss function that encourages the model to have other properties besides the ability to copy its input to its output including
 - sparsity of the representation,
 - smallness of the derivative of the representation,
and
 - Robustness to noise or to missing inputs

- Two generative modeling approaches that emphasize this connection with autoencoders are the descendants of the Helmholtz machine such as the
 - variational autoencoder and
 - the generative stochastic networks.
- These models naturally learn high-capacity, overcomplete encodings of the input and do not require regularization for these encodings to be useful.
- Their encodings
- are naturally useful because the models were trained to approximately maximize
- the probability of the training data rather than to copy the input to the output

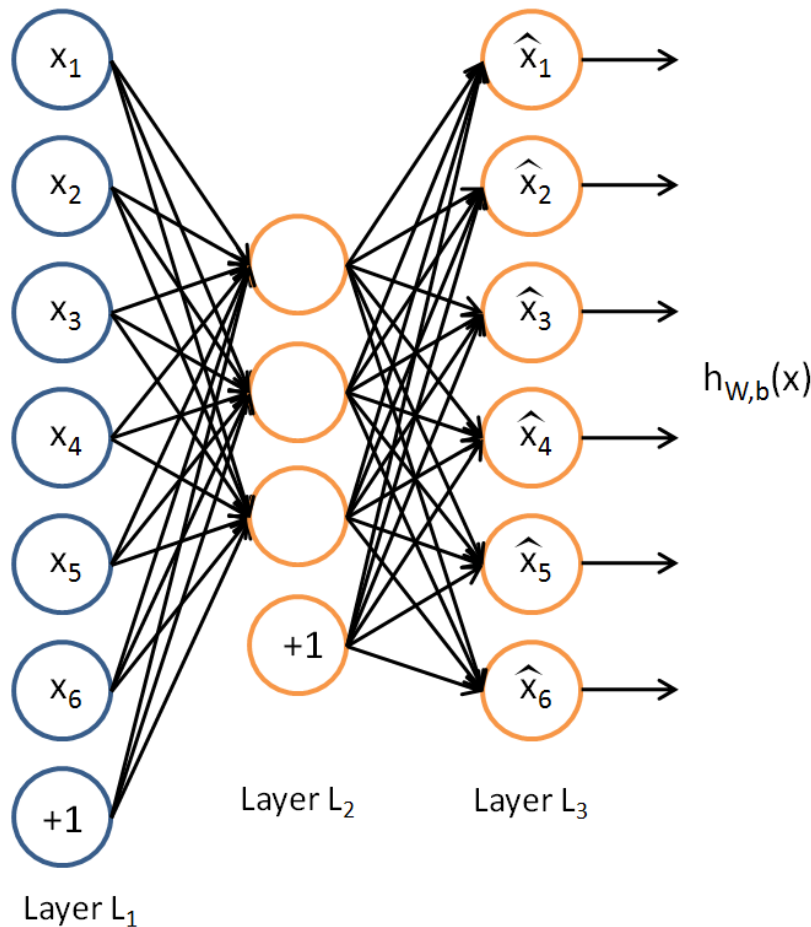
Sparse autoencoder

- A sparse autoencoder is simply an autoencoder whose training criterion involves a sparsity penalty $\Omega(h)$ on the code layer h , in addition to the reconstruction error:

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

Unsupervised feature learning with a neural network

Autoencoder



Unlabeled training examples set
 $\{x^{(1)}, x^{(2)}, x^{(3)} \dots\}, x^{(i)} \in \mathbb{R}^n$

An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs. $y^{(i)} = x^{(i)}$

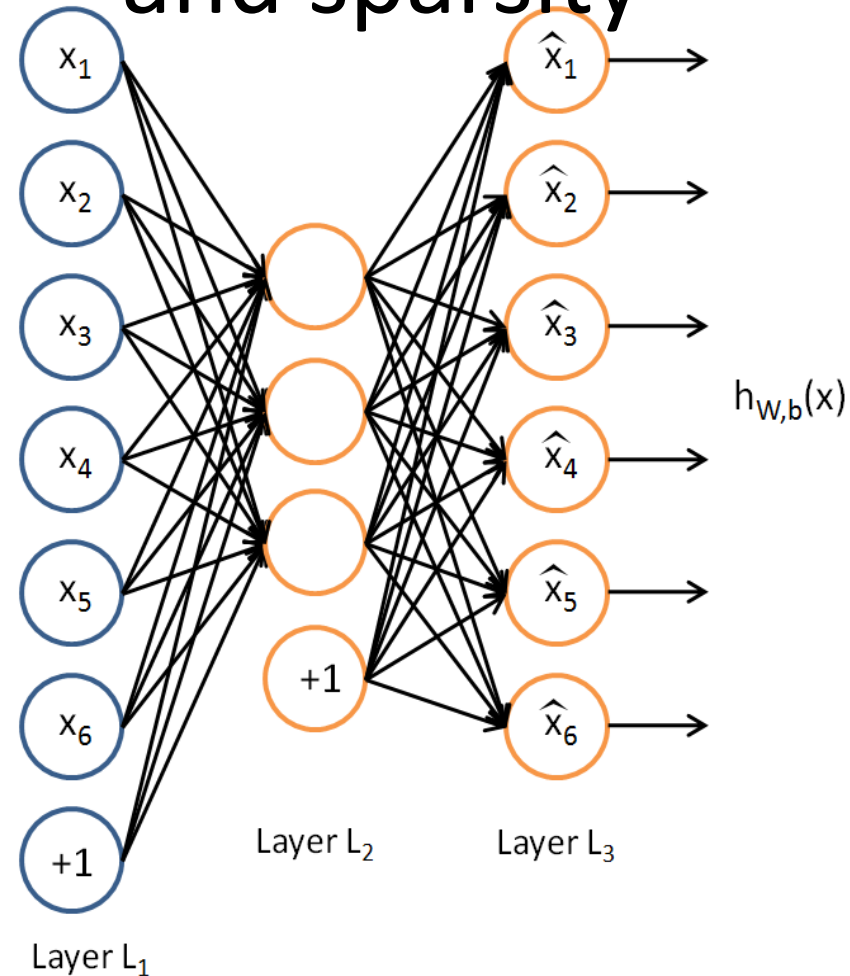
Network is trained to output the input (learn identity function).

$$h_{w,b}(x) \approx x$$

Solution may be trivial.

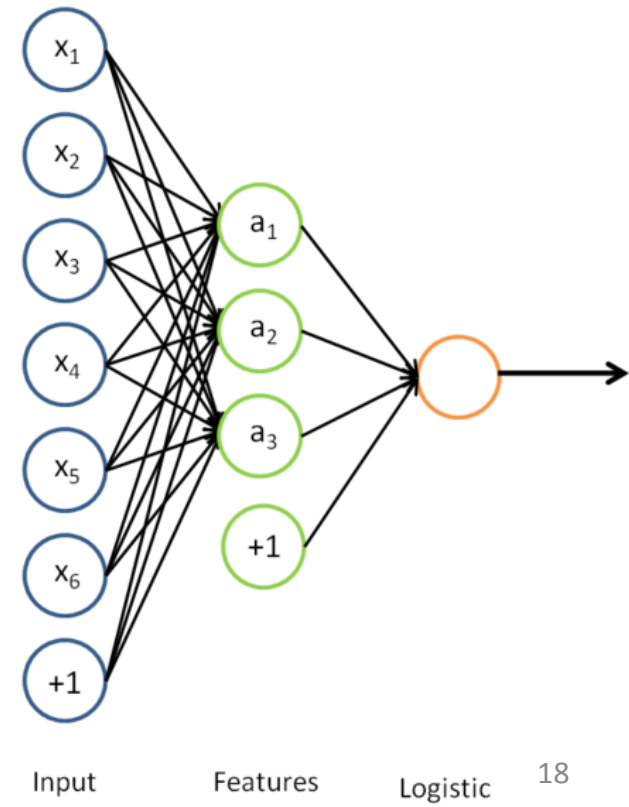
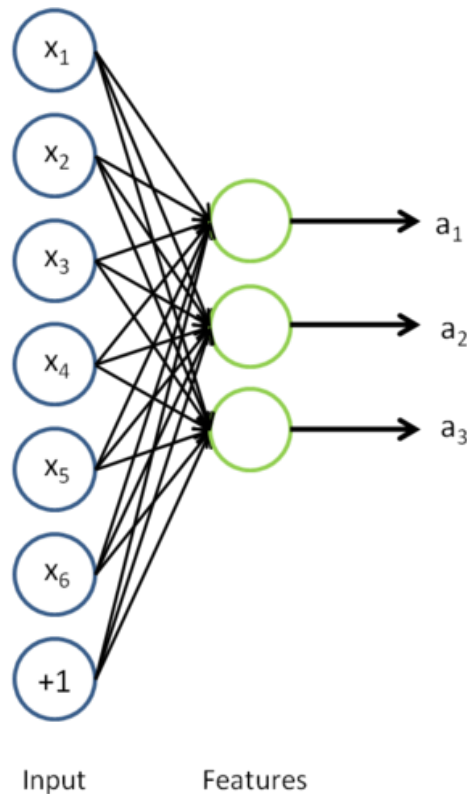
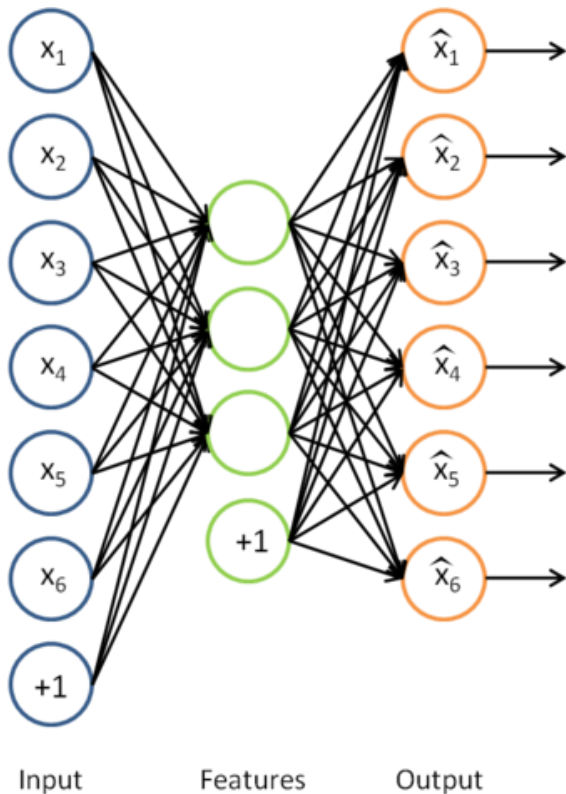
1. By placing constraints on the network, like **limiting the number of hidden units**, can discover interesting structure about the data.
2. When the number of hidden units is large, can still discover interesting structure, by imposing other constraints on the

Autoencoders and sparsity

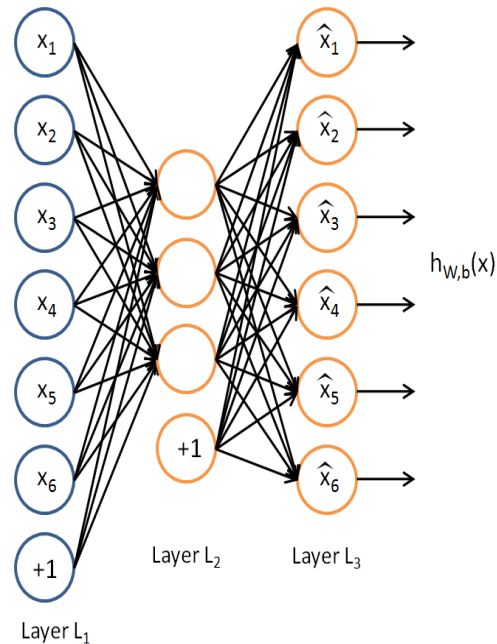


Auto-Encoders

- A type of unsupervised learning which tries to discover generic features of the data
 - Learn identity function by learning important sub-features (not by just passing through data)
 - Compression, etc.
 - Can use just new features in the new training set or concatenate both



Autoencoder



- Suppose the inputs x are the pixel intensity values from a 10×10 image (100 pixels) so $n = 100$.
- There are $s_2 = 50$ hidden units in layer L₂.
- Note that we also have $y \in \mathcal{R}^{100}$
- Since there are only 50 hidden units, the network is forced to learn a "compressed" representation of the input.
- Given only the vector of hidden unit activations $a^{(2)} \in \mathcal{R}^{100}$
- It must try to "reconstruct" the 100-pixel input x

Autoencoder

- If the input were completely random—say, each x_i comes from an IID Gaussian independent of the other features—then this compression task would be very difficult.
- But if there is structure in the data, for example, if some of the input features are correlated, then this algorithm will be able to discover some of those correlations.

(In fact, this simple autoencoder often ends up learning a low-dimensional representation very similar to PCAs.)

Autoencoder

- Dimensionality reduction leads to a "dense" representation which is nice in terms of parsimony
- All features typically have non-zero values for any input and the combination contains the compressed information
- However, this distributed and entangled representation can often make it more difficult for successive layers to pick out the salient features

Outline

- Motivating factors and intuition
- Neural Network: Multi-layer perceptrons
- Deep learning methods
 - Autoencoder
 - Sparse autoencoders
 - Denoising autoencoders
 - RBMs
 - Deep Belief Network
- Applications

Sparse Encoders

- A sparse representation uses more features where at any given time a significant number of the features will have a 0 value
 - This leads to more localist variable length encodings where a particular node (or small group of nodes) with value 1 signifies the presence of a feature
 - A type of simplicity bottleneck (regularizer)
 - This is easier for subsequent layers to use for learning

How do we implement a sparse Auto-Encoder?

- Use more hidden nodes in the encoder
- Use regularization techniques which encourage sparseness (e.g. a significant portion of nodes have 0 output for any given input)
 - Penalty in the learning function for non-zero nodes
 - Weight decay
 - etc.

Sparse Autoencoder

- If we impose a “sparsity” constraint on the hidden units, then the autoencoder will still discover interesting structure in the data, even if the number of hidden units is large.
- Informally, we will think of a neuron as being “active” (or as “firing”) if its output value is close to 1, or as being “inactive” if its output value is close to 0. We would like to constrain the neurons to be inactive most of the time. (assuming a sigmoid activation function).
- Let $a_j^{(2)}(x)$ denotes the activation of hidden unit j in the autoencoder when the network is given a specific input x .
- The average activation of hidden unit j is (averaged over the training set)

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^m [a_j^{(2)}(x^{(i)})]$$

Sparsity constraint

- We enforce the constraint

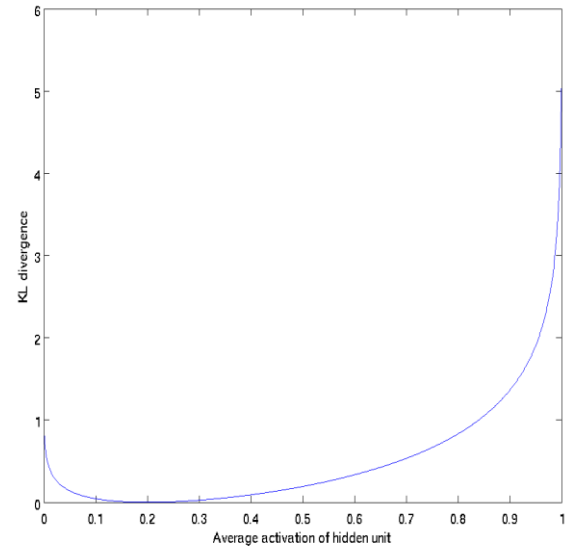
$$\hat{\rho}_j = \rho$$

- ρ is a “sparsity parameter” typically a small value close to zero (say 0.05)
- To satisfy this constraint, the hidden unit’s activations must mostly be near 0.
- add an extra penalty term to our optimization objective that penalizes $\hat{\rho}_j$ deviating significantly from ρ , e.g.,

$$KL(\rho || \hat{\rho}_j) = \sum_{j=1}^{s2} \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j}$$

Overall cost function:

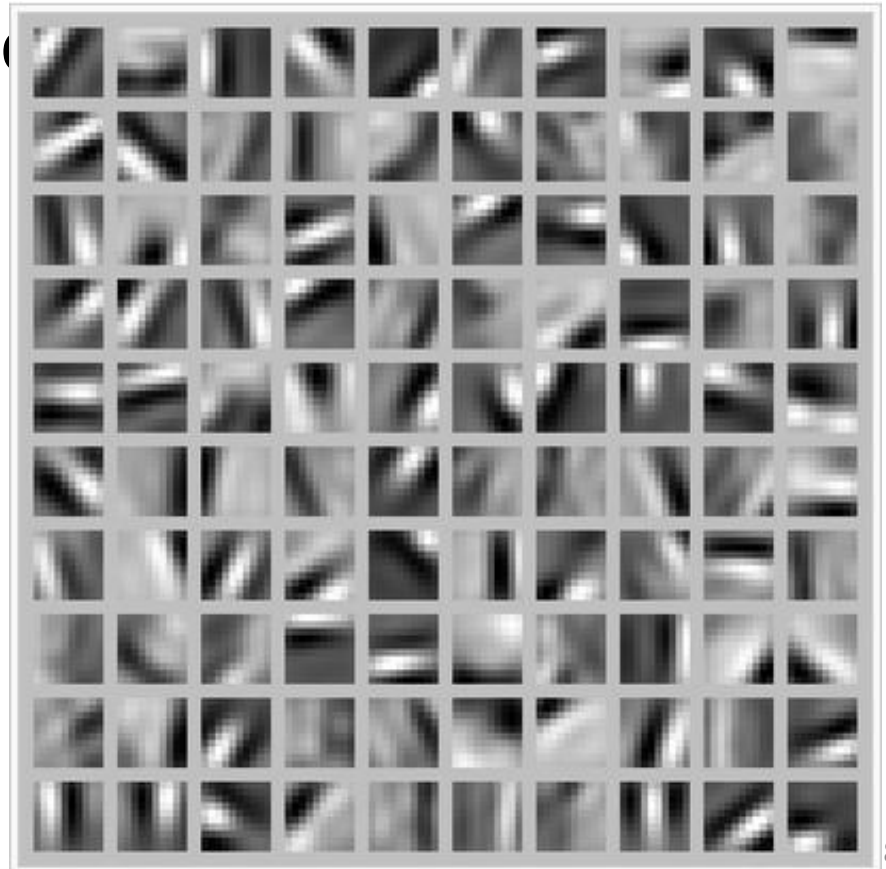
$$J_{sparse}(W, b) = J(W, b) + \beta \sum_{j=1}^{s2} KL(\rho || \hat{\rho}_j)$$



- To incorporate the KL-divergence term into your derivative calculation
 - where previously for the second layer ($l=2$), during backpropagation $\delta_i^{(2)} = \left(\sum_{j=1}^{s_2} W_{ji}^{(2)} \delta_j^{(3)} \right) f'(z_i^{(2)})$, puted
 - now cor $\delta_i^{(2)} = \left(\left(\sum_{j=1}^{s_2} W_{ji}^{(2)} \delta_j^{(3)} \right) + \beta \left(-\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i} \right) \right) f'(z_i^{(2)})$.
 - you'll need to know $\hat{\rho}_i$ to compute this term.
 - compute a forward pass on all the training examples first to compute the average activations on the training set, before computing backpropagation on any example

Sparse Representation

- For bases below, easier to see intuition for current pattern, if a few of these are on and the rest 0, or if all have a significant value?
- Machines can learn more

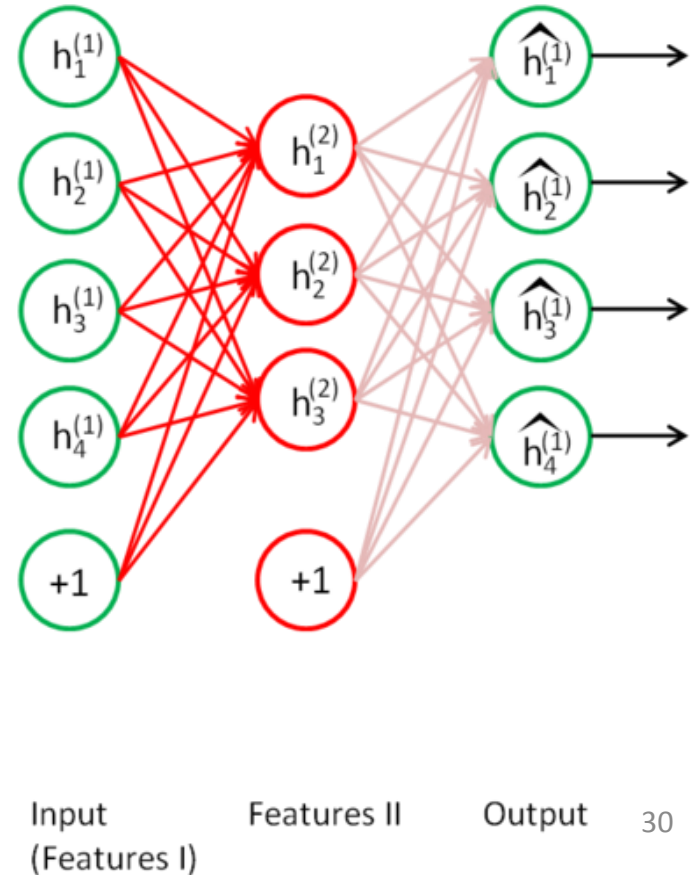
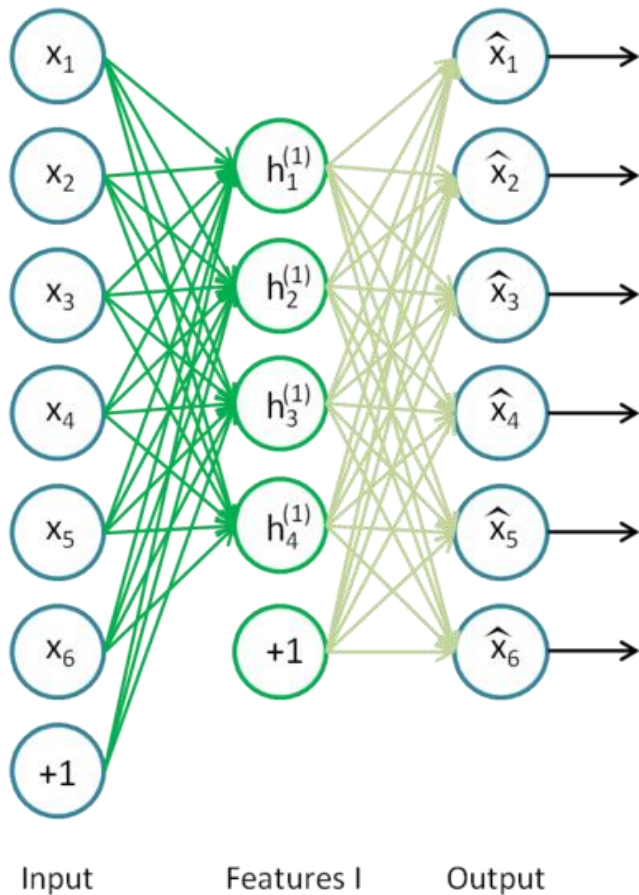


Stacked Autoencoders

- Bengio (2007) – After Deep Belief Networks (2006)
- greedy layerwise approach for pretraining a deep network works by training each layer in turn.
- multiple layers of sparse autoencoders in which the outputs of each layer is wired to the inputs of the successive layer.

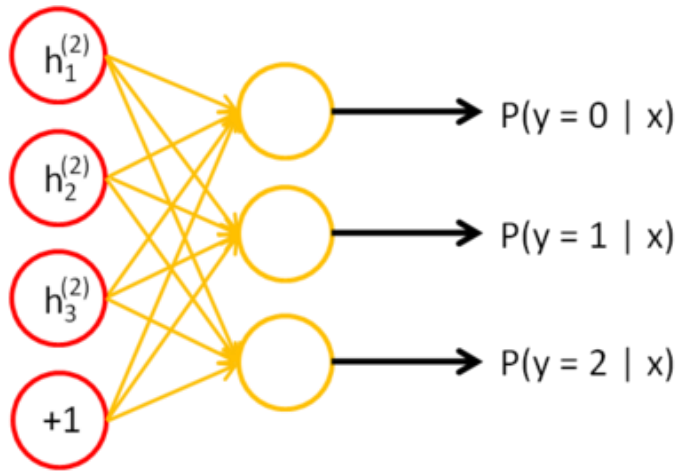
Stacked Auto-Encoders

- Stack many (sparse) auto-encoders in succession and train them using greedy layer-wise training
- Drop the decode layer each time



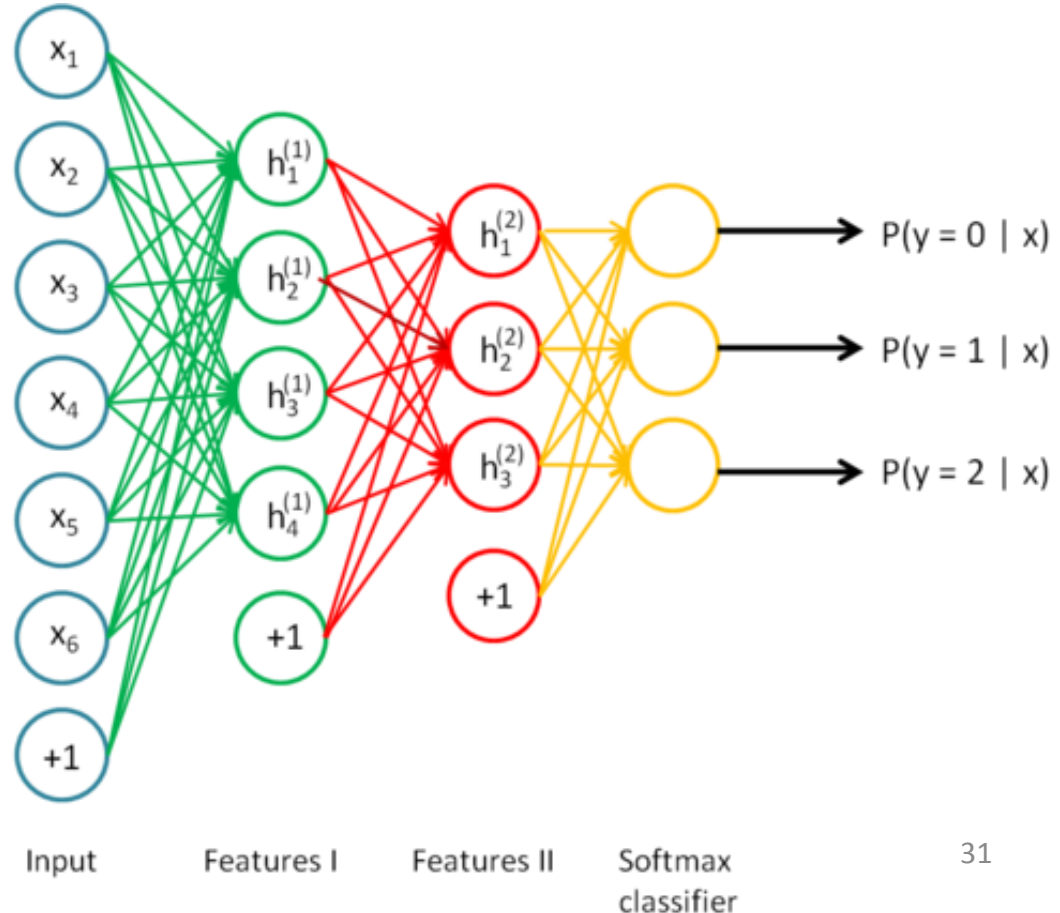
Stacked Auto-Encoders

- Do supervised training on the last layer using final features
- Then do supervised training on the entire network to fine-tune all weights



Input
(Features II) Softmax
classifier

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$



- Formally, consider a stacked autoencoder with n layers. Using notation from the autoencoder section, let $W^{(k,1)}, W^{(k,2)}, b^{(k,1)}, b^{(k,2)}$ denote the parameters $W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}$ for k th autoencoder. Then the encoding step for the stacked autoencoder is given by running the encoding step of each layer in forward order:

$$a^{(l)} = f(z^{(l)})$$

$$z^{(l+1)} = W^{(l,1)} a^{(l)} + b^{(l,1)}$$

- The decoding step is given by running the decoding stack of each autoencoder in reverse order:

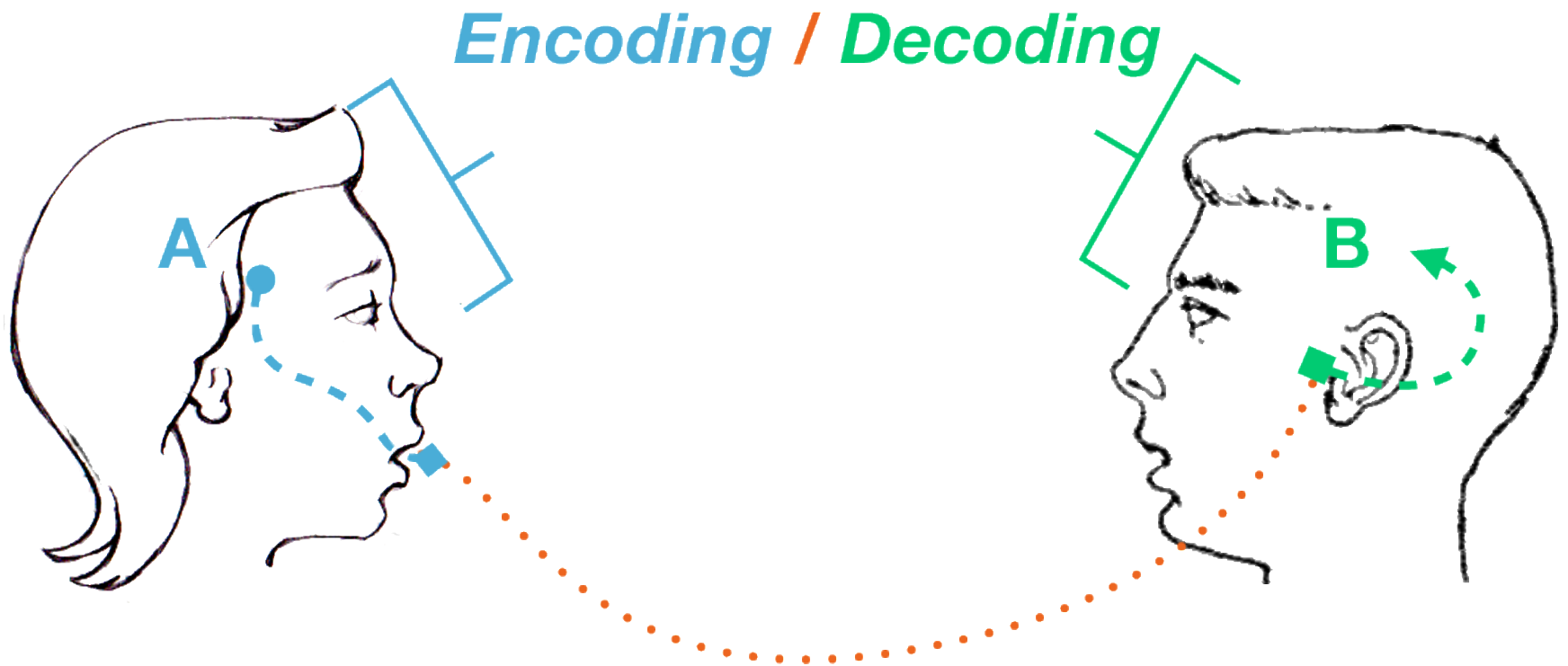
$$a^{(n+l)} = f(z^{(n+l)})$$

$$z^{(n+l+1)} = W^{(n-l,2)} a^{(n+l)} + b^{(n-l,2)}$$

- The information of interest is contained within $a^{(n)}$, which is the activation of the deepest layer of hidden units. This vector gives us a representation of the input in terms of higher-order features.
- The features from the stacked autoencoder can be used for classification problems by feeding $a^{(n)}$ to a softmax classifier.

Denoising auto-encoder

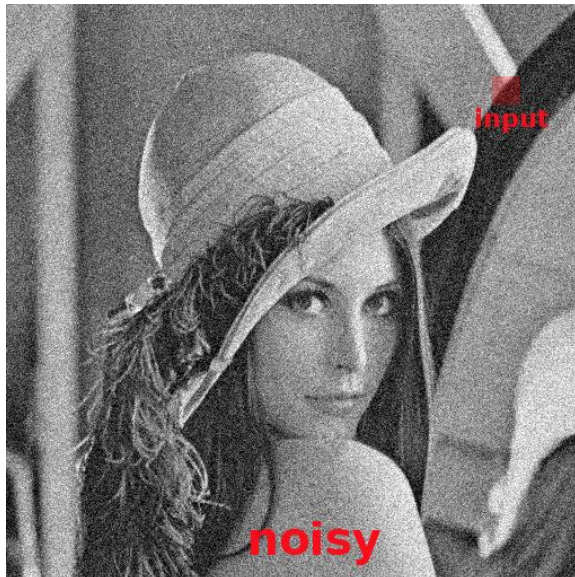
Encoder?



Denoising Auto-Encoder?

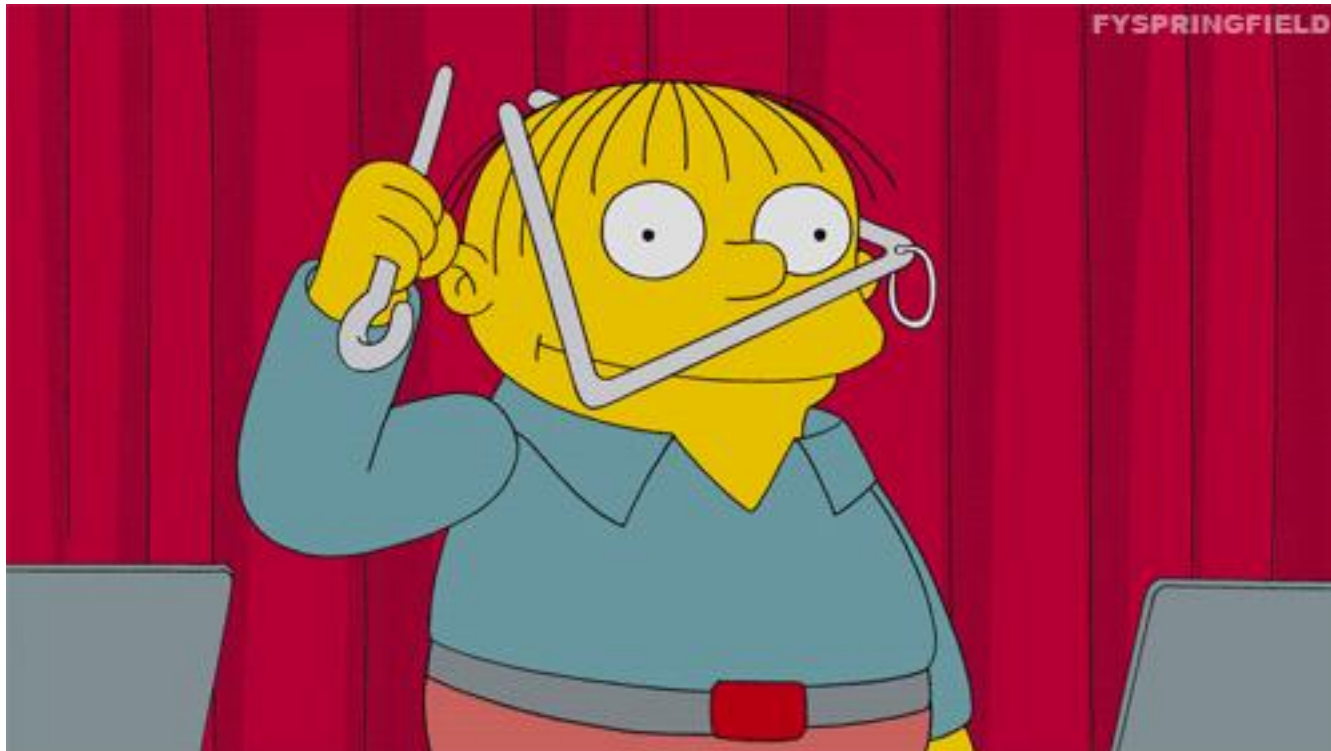
Input and output data of the auto-encoder is basically **identical**.

Then, how can we make this network to have **denoising power**?



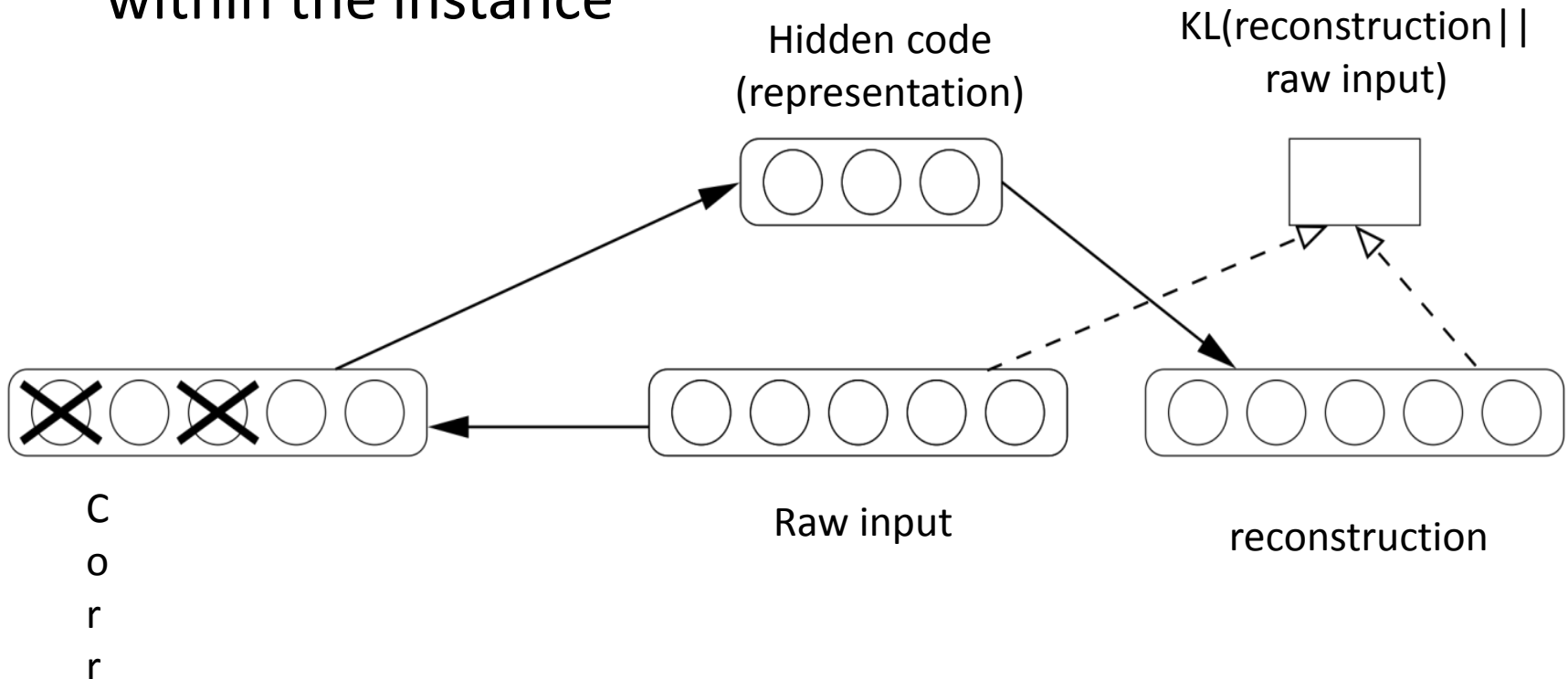
Denoising Auto-Encoder?

Just **add noise** to the input layer while training!



De-noising Auto-Encoder

- A stochastic version of the auto-encoder.
- Stochastically corrupt training instance each time, but still train auto-encoder to decode the uncorrupted instance, forcing it to learn conditional dependencies within the instance



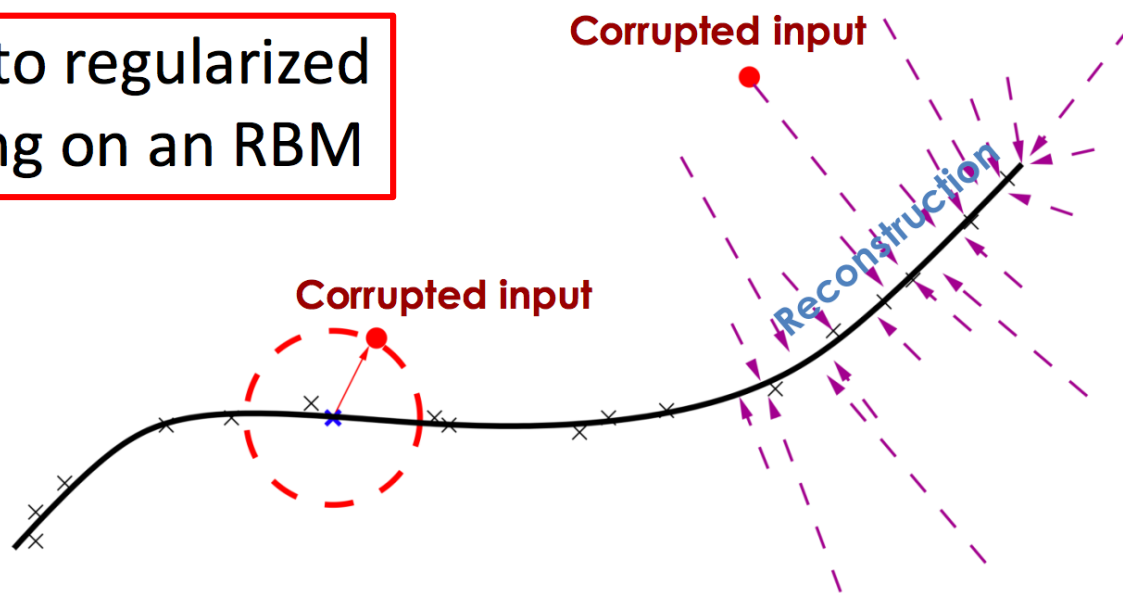
De-noising Auto-Encoder

- Better empirical results, handles missing values well
- Unsupervised initialization of layers with an explicit denoising criterion appears to help capture interesting structure in the input distribution.
- This leads to intermediate representations much better suited for subsequent learning tasks such as supervised classification.

Denoising Auto-Encoder?

(Vincent et al, 2008)

- Learns a vector field towards higher probability regions
- Minimizes variational lower bound on a generative model
- Corresponds to regularized score matching on an RBM



Results

