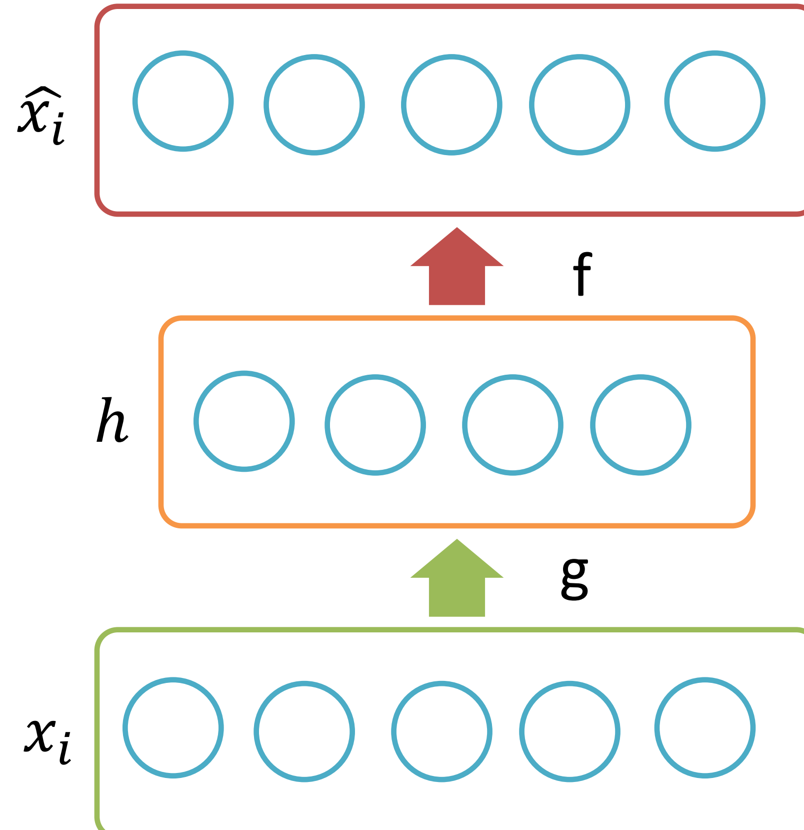
Autoencoder

14 Mar 2019

Definition

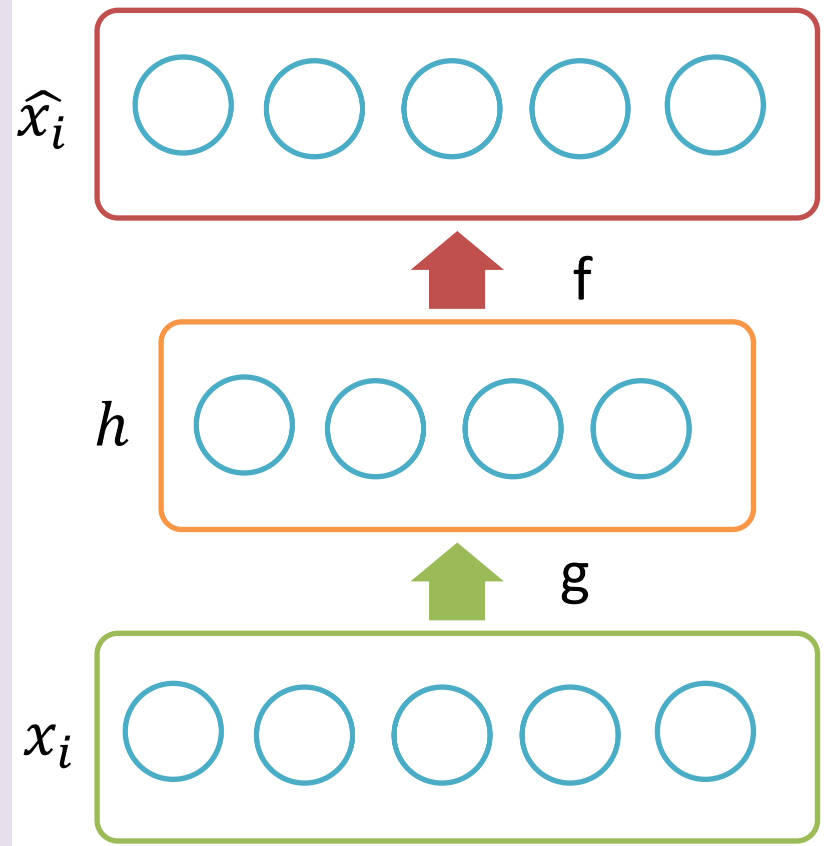
Autoencoders are neural networks that are trained to copy their inputs to their outputs.

- Usually constrained in particular ways to make this task more difficult.



Encoder and Decoder

- Encoder: Encodes its input x_i into a hidden representation h
$$h = g(W_1 x_i + b)$$
- Decodes the input again from this hidden representation
$$\widehat{x_i} = f(W_2 h + c)$$
- The model is trained to minimize a loss function which will ensure that $\widehat{x_i}$ is close to x_i



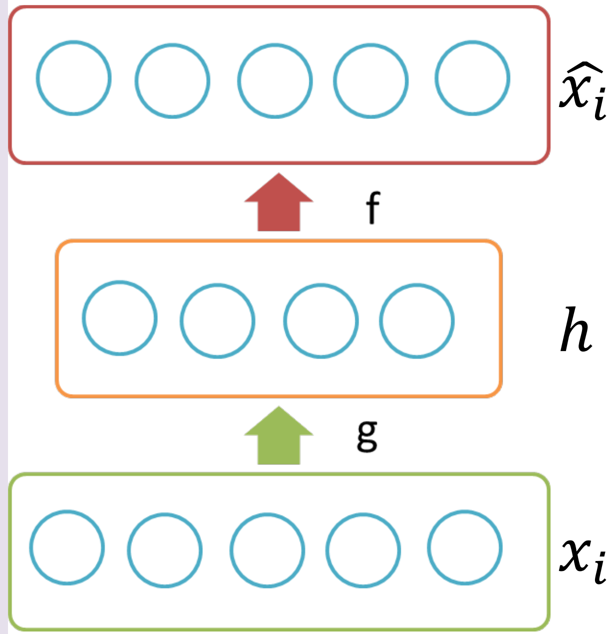
Undercomplete autoencoder

1. $\dim(h) < \dim(x_i)$

- Network must model x in lower dim. space + map latent space accurately back to input space.

Encoder network:

- hidden layer "compresses" the input
- will compress well only for the training distribution

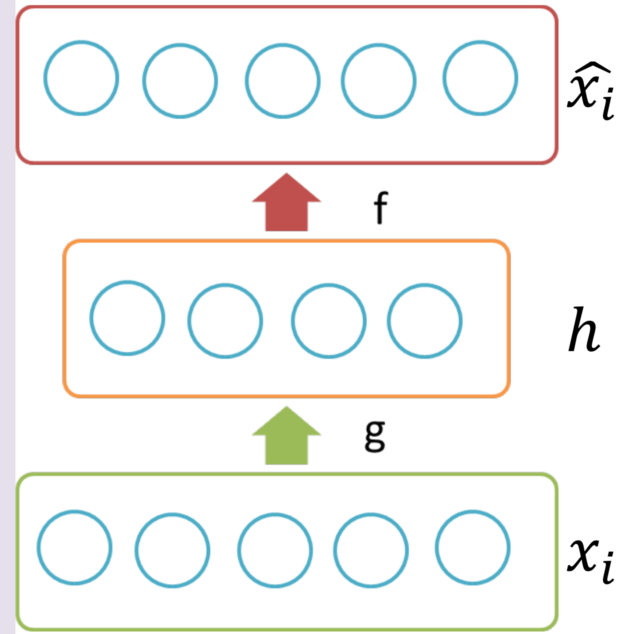


$$h = g(W_1 x_i + b)$$
$$\hat{x}_i = f(W_2 h + c)$$

Undercomplete autoencoder

1. $\dim(h) < \dim(x_i)$

If network has only linear transformations, encoder learns PCA. With typical non-linearities, network learns generalized, more powerful version of PCA.

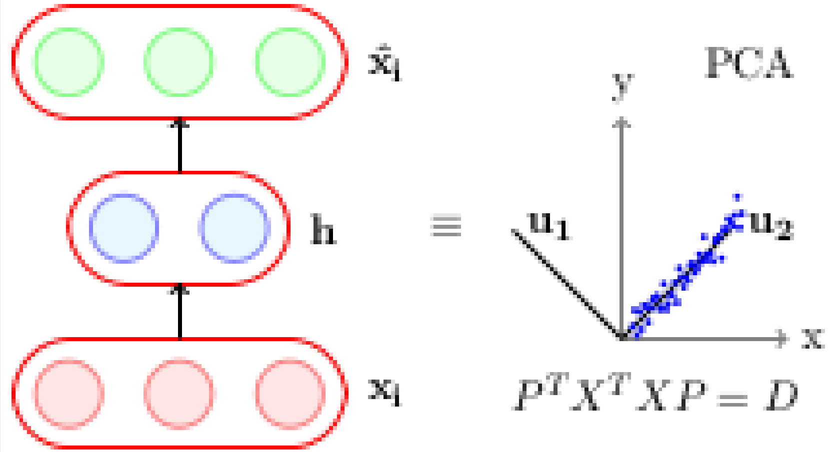


$h = g(W_1 x_i + b)$
$\hat{x}_i = f(W_2 h + c)$

Link between PCA and Autoencoders

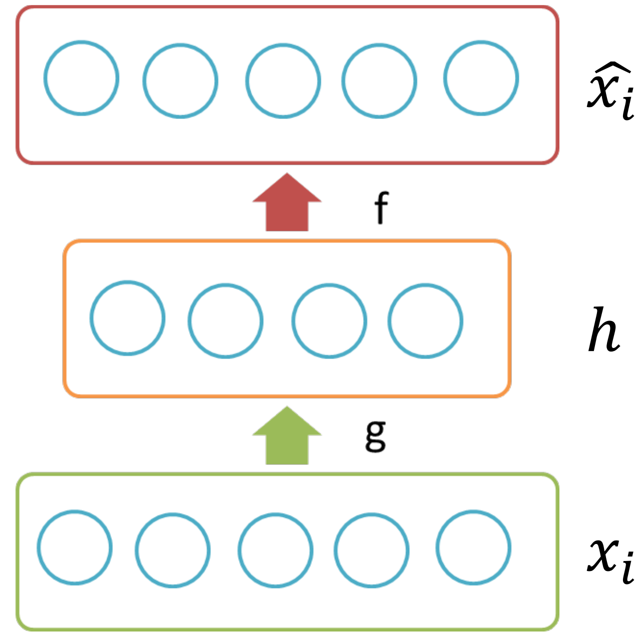
- the encoder part of an autoencoder is equivalent to PCA if we
  - use a linear encoder
  - use a linear decoder
  - use squared error loss function
  - normalize the inputs to

$$\hat{x}_{ij} = \frac{1}{\sqrt{m}}\left(x_{ij} - \frac{1}{m}\sum_{k=1}^{m} x_{kj}\right)$$



Overcomplete Autoencoder

2. $\dim(h) > \dim(x_i)$
   - no compression in hidden layer
   - each hidden unit could copy a different input component
- No guarantee that the hidden units will extract meaningful structure



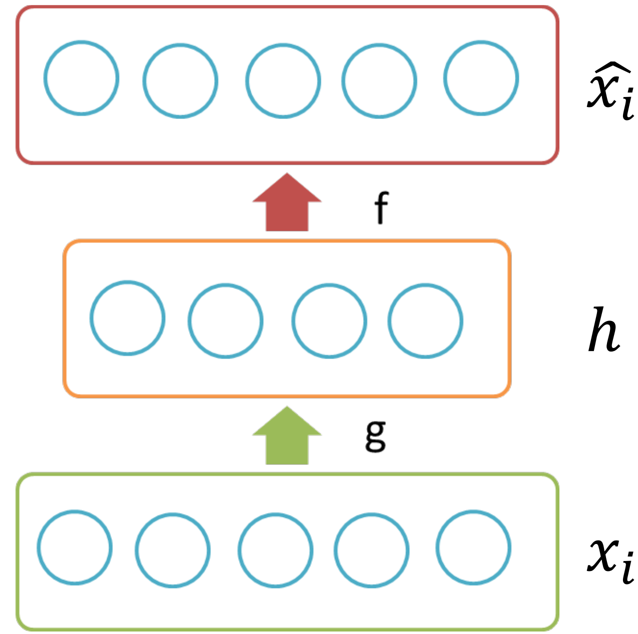
$$h = g(W_1 x_i + b)$$
$$\hat{x}_i = f(W_2 h + c)$$

Binary Inputs

$$x_{ij} \in \{0,1\}$$

Decoder: use logistic function

Encoder: use sigmoid function



$$h = g(W_1 x_i + b)$$

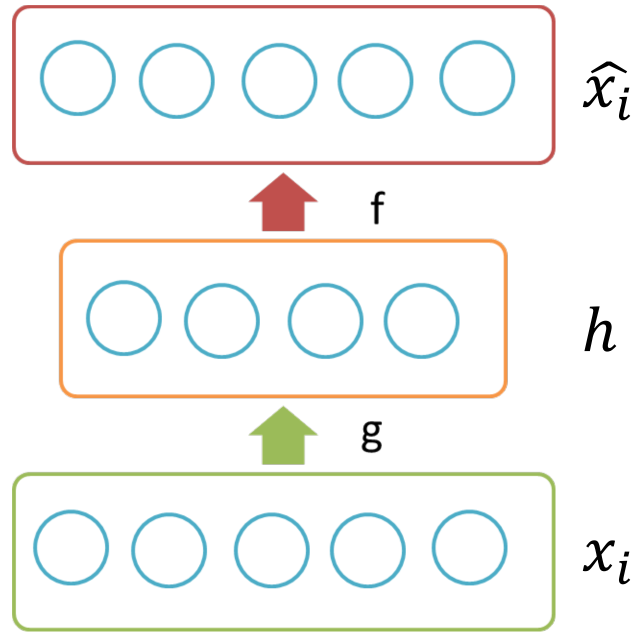
$$\hat{x}_i = f(W_2 h + c)$$

Real Inputs

$$x_{ij} \in R$$

Decoder: use linear function

Encoder: use sigmoid function



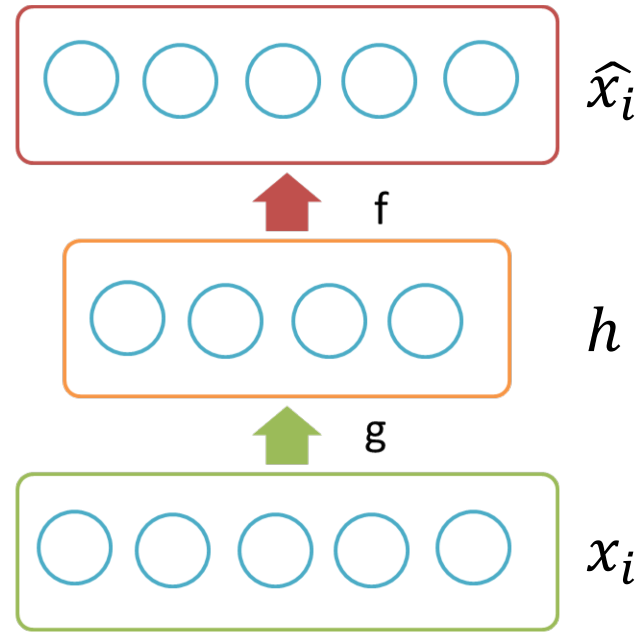
$$h = g(W_1 x_i + b)$$

$$\hat{x}_i = f(W_2 h + c)$$

Loss Function: Real Inputs

$x_{ij} \in R$

$$\min_{W_1, W_2, b, c} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\hat{x}_{ij} - x_{ij}\right)^2$$



$$h = g(W_1 x_i + b)$$

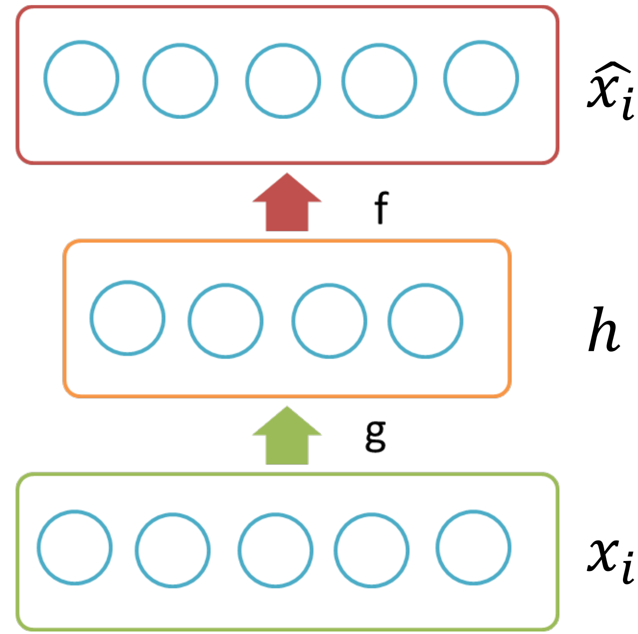
$$\hat{x}_i = f(W_2 h + c)$$

Loss Function: Binary Inputs

$x_{ij} \in \{0,1\}$

- For a single n-dimensional ith input we can use the following loss function

$$min\left\{-\sum_{j=1}^{n}\left(x_{ij}\log\hat{x}_{ij}+\left(1-x_{ij}\right)\log\left(1-\hat{x}_{ij}\right)\right)\right\}$$



$$h = g(W_1 x_i + b)$$

$$\hat{x}_i = f(W_2 h + c)$$

Regularization in autoencoders

- While poor generalization could happen even in undercomplete autoencoders it is an even more serious problem for overcomplete auto encoders
- To avoid poor generalization, we need to introduce regularization

Denoising Autoencoders

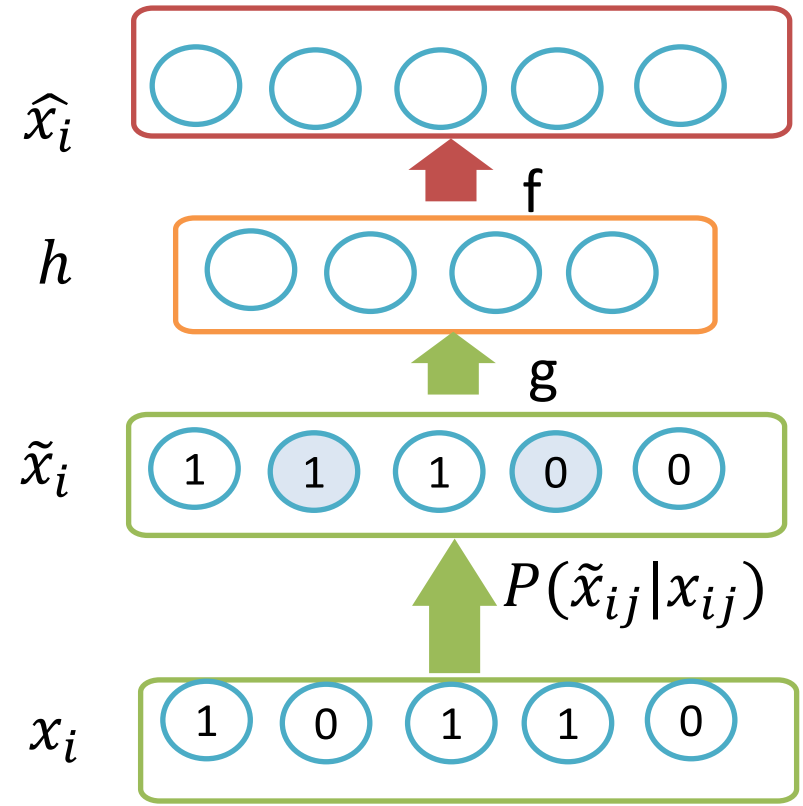
Corrupts the input data using a

1. probabilistic process ($P(\tilde{x}_{ij}|x_{ij})$) before feeding it to the network

$$P(\tilde{x}_{ij} = 0|x_{ij}) = q$$
$$P(\tilde{x}_{ij} = x_{ij}|x_{ij}) = 1 - q$$

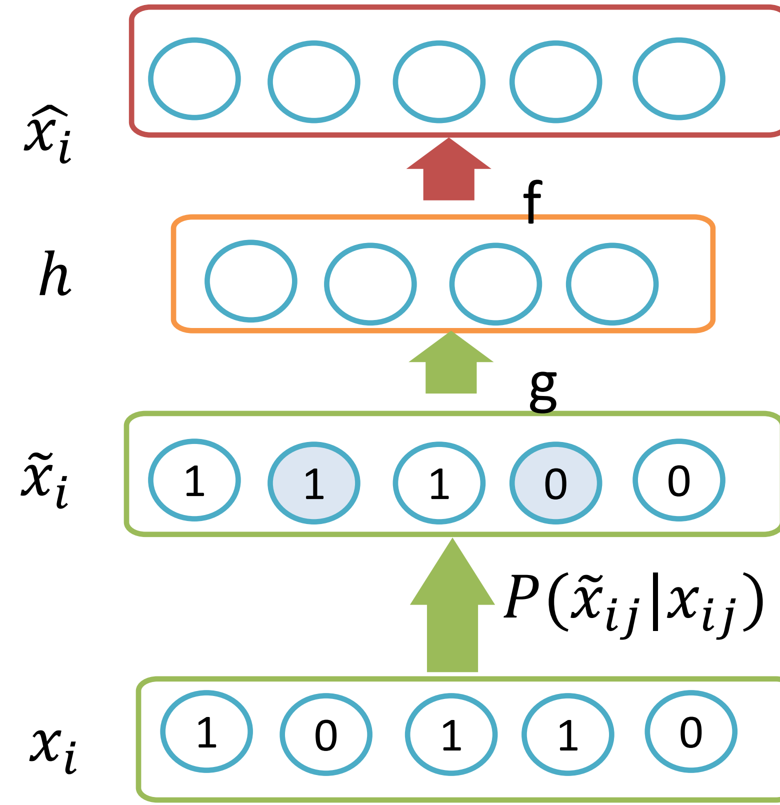
2. Gaussian additive noise

$$\tilde{x}_{ij} = x_{ij} + \mathcal{N}(0,1)$$



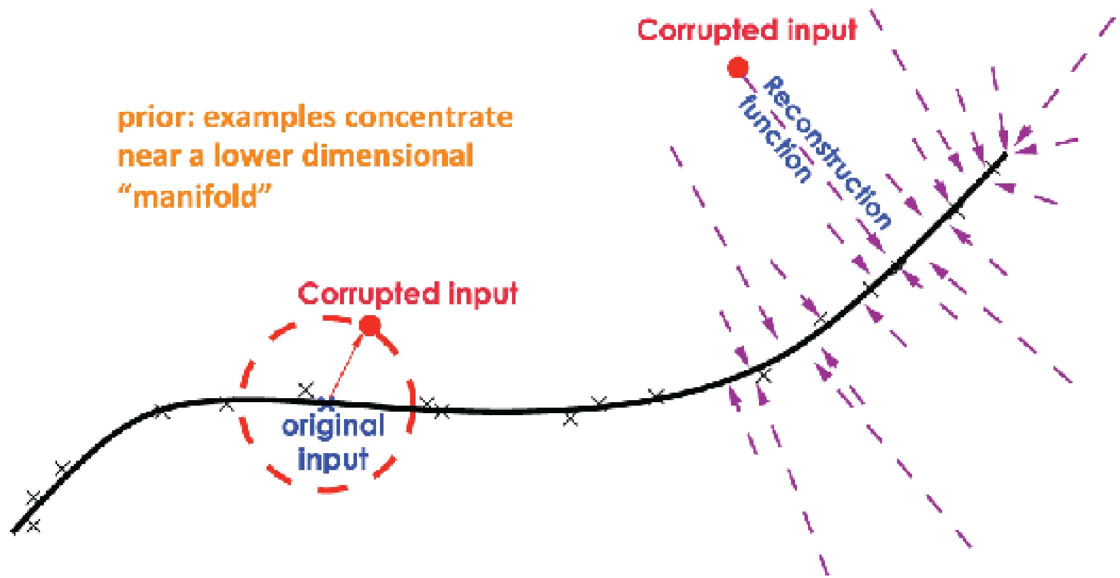
Denoising Autoencoders

- the objective is to reconstruct the original (uncorrupted) x_i
- By having to remove noise, model must know difference between noise and actual image.



Denoising Autoencoder

- The corrupting function $C(.)$ can corrupt in any direction
- autoencoder must learn "location" of data manifold and its distribution $p_{data}(x)$



Application of AE

- Task: Hand-written digit recognition

Contractive Autoencoders

- Contractive Autoencoders are explicitly encouraged to learn a manifold through their loss function.
- Desirable property: Points close to each other in input space maintain that property in the latent space.
- objective is to have a robust learned representation which is less sensitive to small variation in the data.
- We wish to extract features that only reflect variations observed in the training set -- we'd like to be invariant to the other variations
- 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.

Contractive Autoencoders

- Desirable property: Points close to each other in input space maintain that property in the latent space.
- This will be true if $f(x) = h$ is continuous, has small derivatives.
- 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.

$$L(\theta) + \Omega(\theta)$$

$$\Omega(\theta) = \|J_x(h)\|_F^2$$

$J_x(h)$ is a Jacobian of the encoder.

Jacobian and Frobenius Norm

- If the input has n dimensions and the hidden layer has k dimensions then

$$J_x(h) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_k}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \end{bmatrix}$$

- The Frobenius Norm for a matrix M:

$$\|J_x(h)\|_F^2 = \sum_{j=1}^{n}\sum_{l=1}^{k}\frac{\partial h_l}{\partial x_j}$$

- Consider $\frac{\partial h_1}{\partial x_1}$
- $\frac{\partial h_1}{\partial x_1}$=0 means that this neuron is not very sensitive to variations cutting-edge the input x1.
- $L(\theta)$ capture inportant variations in dta
- $\Omega(\theta)$ do not capture variations in data

- Called contractive because they contract neighborhood of input space into smaller, localized group in latent space.
- This contractive effect is designed to only occur locally.
- The Jacobian Matrix will see most of its eigenvalues drop below 1 → contracted directions
- But some directions will have eigenvalues (signicantly) above 1 → directions that explain most of the variance in data

The Big Idea of Regularized Autoencoders

- Previous slides underscore the central balance of regularized autoencoders:
- Be sensitive to inputs (reconstruction loss) → generate good reconstructions of data drawn from data distribution
- Be insensitive to inputs (regularization penalty) → learn actual data distribution

Sparse Autoencoders

- A hidden neuron with sigmoid activation will have values between 0 and 1
- We say that the neuron is activated when its output is close to 1 and not activated when its output is close to 0.
- A sparse autoencoder tries to ensure the neuron is inactive most of the times.

The average value of the activation of a neuron l is given by

$$\hat{\rho}_l = \frac{1}{m}\sum_{i=1}^{m} h(x_i)l$$

If the neuron l is sparse (i.e. mostly inactive) then

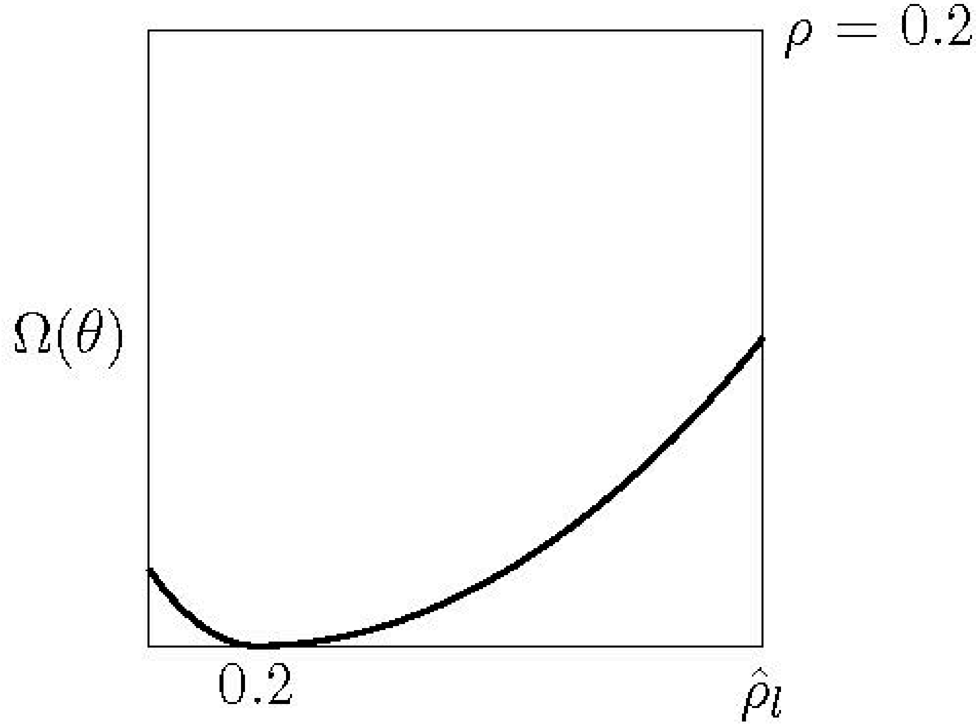
$$\hat{\rho}_l \to 0$$

A sparse autoencoder uses a small value of sparsity parameter ρ (say, 0.005) and tries to enforce the constraint $\hat{\rho}_l = \rho$

- One way of ensuring this is to add the following term to the objective function

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_l}$$

The sparsity function



The function will reach its minimum value(s) when $\hat{\rho}_l = \rho$.

Sparse Autoencoders

- A sparse autoencoder involves a sparsity penalty Ω(h) on the code layer h, in addition to the reconstruction error:

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

- Regularized maximum likelihood corresponds to maximizing p(θ | x), which is equivalent to maximizing

$$\log p(x|\theta) + \log p(\theta)$$

- The log p(x | θ) term is the usual data log-likelihood term and the log p(θ) term, the log-prior over parameters, incorporates the preference over particular values of θ.

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_l}$$

- Can be re-written as

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \rho - \rho \log \hat{\rho}_l + (1-\rho) \log(1-\rho) - (1-\rho) \log(1-\rho)$$

$$\Omega(\theta) = \sum_{l=1}^{k} \rho log \frac{\rho}{\hat{\rho}_l} + (1-\rho) log \frac{1-\rho}{1-\hat{\rho}_l}$$

Can be re-written as

$$\Omega(\theta) = \sum_{l=1}^{k} \rho log \rho - \rho log \hat{\rho}_l + (1-\rho) log(1-\rho) - (1-\rho) log(1-\hat{\rho}_l)$$

By Chain rule:

$$\frac{\partial \Omega(\theta)}{\partial W} = \frac{\partial \Omega(\theta)}{\partial \hat{\rho}} . \frac{\partial \hat{\rho}}{\partial W}$$

$$\frac{\partial \Omega(\theta)}{\partial \hat{\rho}} = \left[\frac{\partial \Omega(\theta)}{\partial \hat{\rho}_1}, \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_2}, \ldots \frac{\partial \Omega(\theta)}{\partial \hat{\rho}_k} \right]^T$$

For each neuron $l \in 1 \ldots k$ in hidden layer, we have

$$\frac{\partial \Omega(\theta)}{\partial \hat{\rho}_l} = -\frac{\rho}{\hat{\rho}_l} + \frac{(1-\rho)}{1-\hat{\rho}_l}$$

and $\quad \dfrac{\partial \hat{\rho}_l}{\partial W} = \mathbf{x}_i (g'(W^T \mathbf{x}_i + \mathbf{b}))^T$ (see next slide)

- Now,

$$\hat{\mathscr{L}}(\theta) = \mathscr{L}(\theta) + \Omega(\theta)$$

- $\mathscr{L}(\theta)$ is the squared error loss or cross entropy loss and $\Omega(\theta)$ is the sparsity constraint.
- We already know how to calculate $\frac{\partial \mathscr{L}(\theta)}{\partial W}$
- Let us see how to calculate $\frac{\partial \Omega(\theta)}{\partial W}$.
- Finally,

$$\frac{\partial \hat{\mathscr{L}}(\theta)}{\partial W} = \frac{\partial \mathscr{L}(\theta)}{\partial W} + \frac{\partial \Omega(\theta)}{\partial W}$$

(and we know how to calculate both terms on R.H.S)

Derivation

$$\frac{\partial \hat{\rho}}{\partial W} = \left[\frac{\partial \hat{\rho}_1}{\partial W} \quad \frac{\partial \hat{\rho}_2}{\partial W} \ldots \frac{\partial \hat{\rho}_k}{\partial W}\right]$$

For each element in the above equation we can calculate $\frac{\partial \hat{\rho}_l}{\partial W}$ (which is the partial derivative of a scalar w.r.t. a matrix = matrix). For a single element of a matrix W_{jl}:-

$$\begin{aligned}
\frac{\partial \hat{\rho}_l}{\partial W_{jl}} &= \frac{\partial\left[\frac{1}{m}\sum_{i=1}^{m} g\left(W_{:,l}^T \mathbf{x}_\mathrm{i} + b_l\right)\right]}{\partial W_{jl}} \\
&= \frac{1}{m}\sum_{i=1}^{m} \frac{\partial\left[g\left(W_{:,l}^T \mathbf{x}_\mathrm{i} + b_l\right)\right]}{\partial W_{jl}} \\
&= \frac{1}{m}\sum_{i=1}^{m} g'\left(W_{:,l}^T \mathbf{x}_\mathrm{i} + b_l\right) x_{ij}
\end{aligned}$$

So in matrix notation we can write it as :

$$\frac{\partial \hat{\rho}_l}{\partial W} = \mathbf{x}_i (g'(W^T \mathbf{x}_i + \mathbf{b}))^T$$

Representational Power, Layer Size and Depth

- Deeper autoencoders tend to generalize better and train more efficiently than shallow ones.
  - Common strategy: greedily pre-train layers and stack them
  - For contractive autoencoders, calculating Jacobian for deep networks is expensive. Good idea to do layer-by-layer.

Applications of Autoencoders

- Dimensionality Reduction: Make high-quality, low-dimension representation of data
- Information Retrieval: Locate value in database which is just autoencoded key.