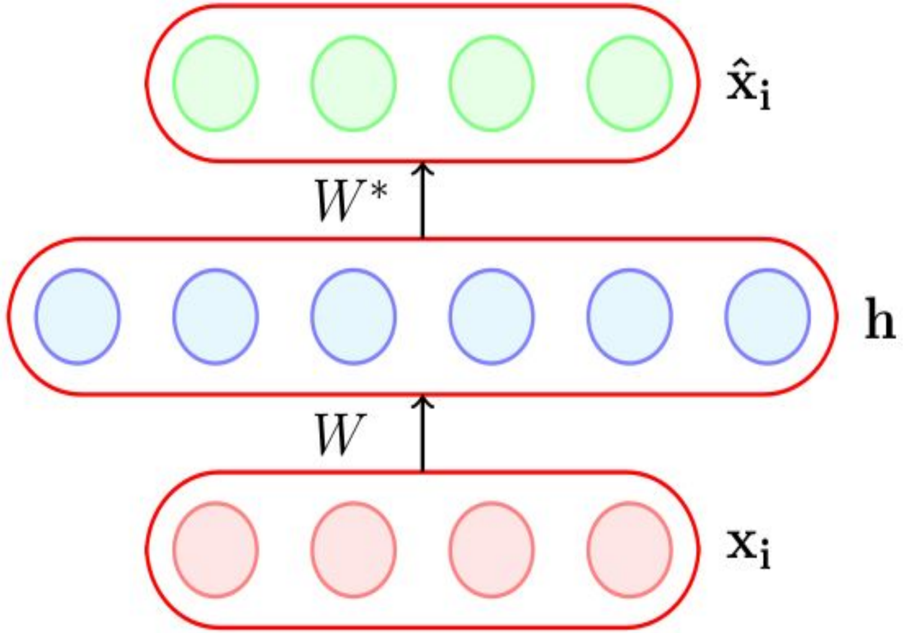
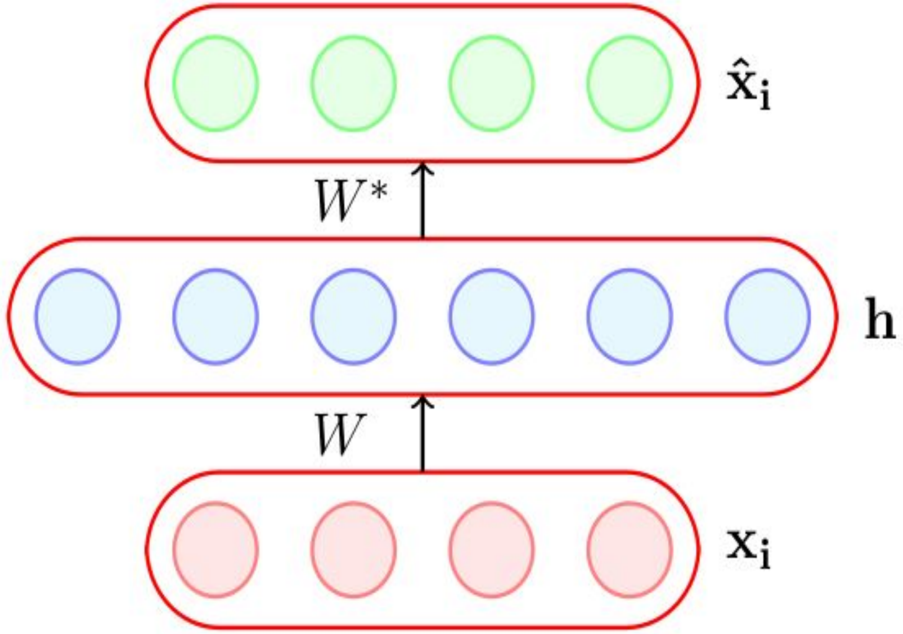
CS 349: Autoencoder Part-II

Asif Ekbal

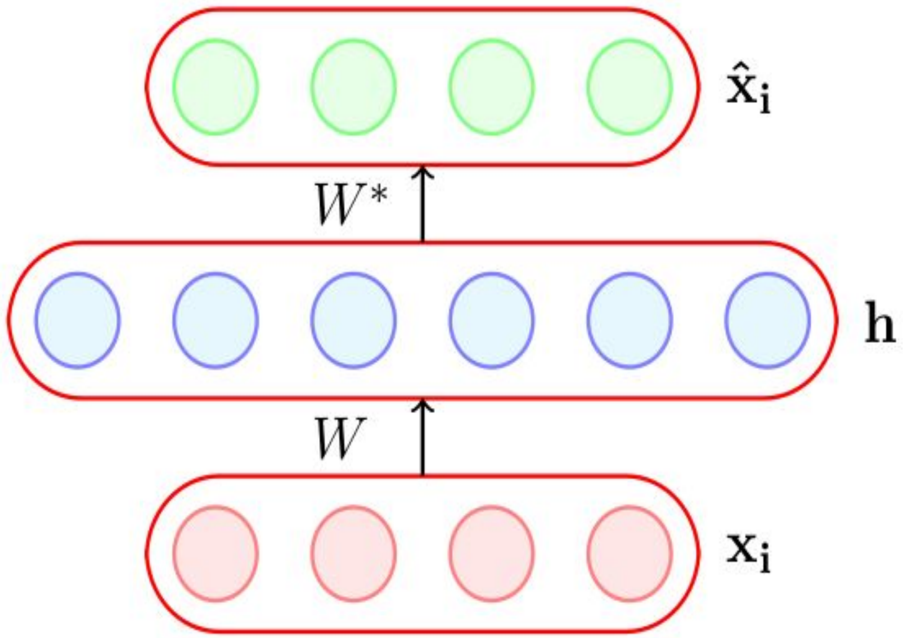
Department of Computer Science and Engineering
Indian Institute of Technology Patna

Regularization in autoencoders (Motivation)

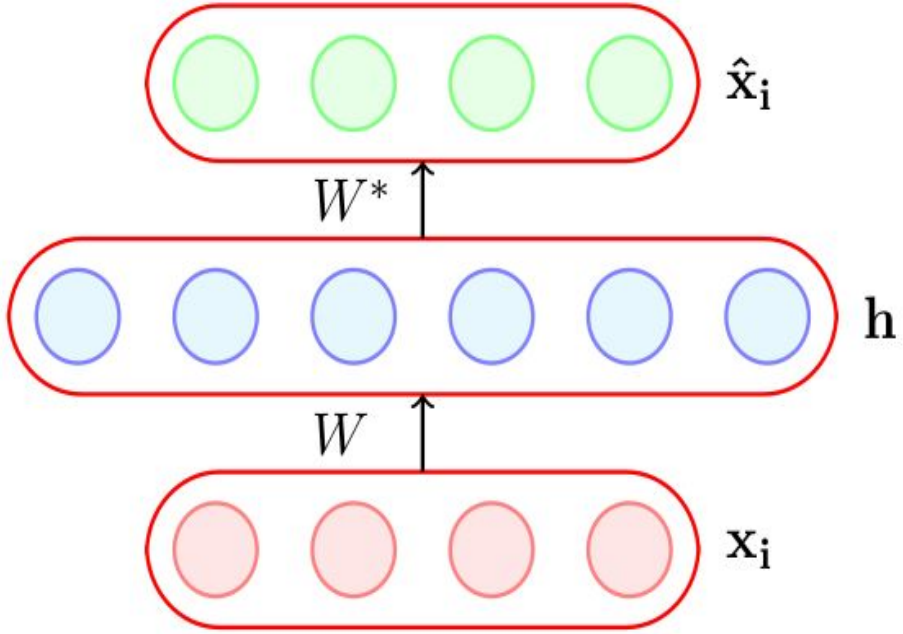

$\hat{\mathbf{x}}_{\mathbf{i}}$
W^*
$\mathbf{h}$
W
$\mathbf{x}_{\mathbf{i}}$



- While poor generalization could happen even in under-complete autoencoders, it is an even more serious problem for over-complete auto encoders



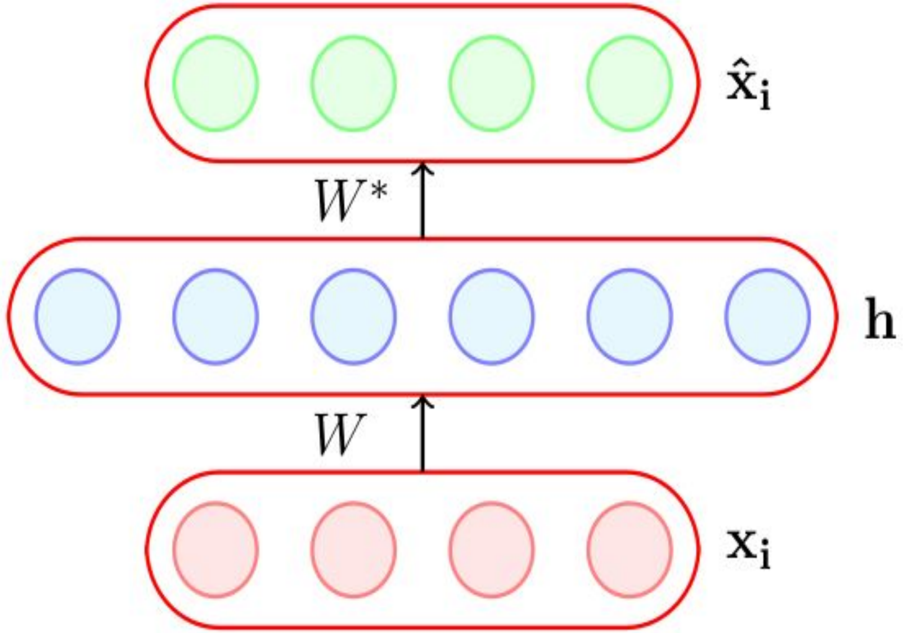
- While poor generalization could happen even in under complete autoencoders it is an even more serious problem for overcomplete auto encoders
- Here, (as stated earlier) the model can simply learn to copy $\mathbf{x}_i$ to $\mathbf{h}$ and then $\mathbf{h}$ to $\hat{\mathbf{x}}_i$



- While poor generalization could happen even in under complete autoencoders it is an even more serious problem for overcomplete auto encoders

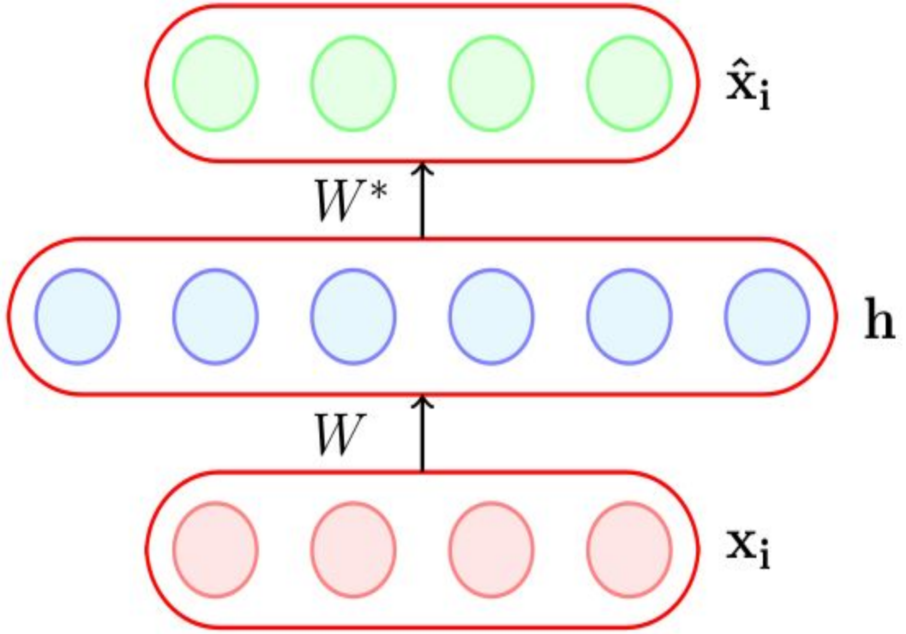
- Here, (as stated earlier) the model can simply learn to copy $\mathbf{x}_i$ to **h** and then **h** to $\hat{\mathbf{x}}_i$

- To avoid poor generalization, we need to introduce regularization



- The simplest solution is to add a L_2 regularization term to the objective function

$$\min_{\theta, w, w^*, \mathbf{b}, \mathbf{c}} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2 + \lambda \|\theta\|^2$$

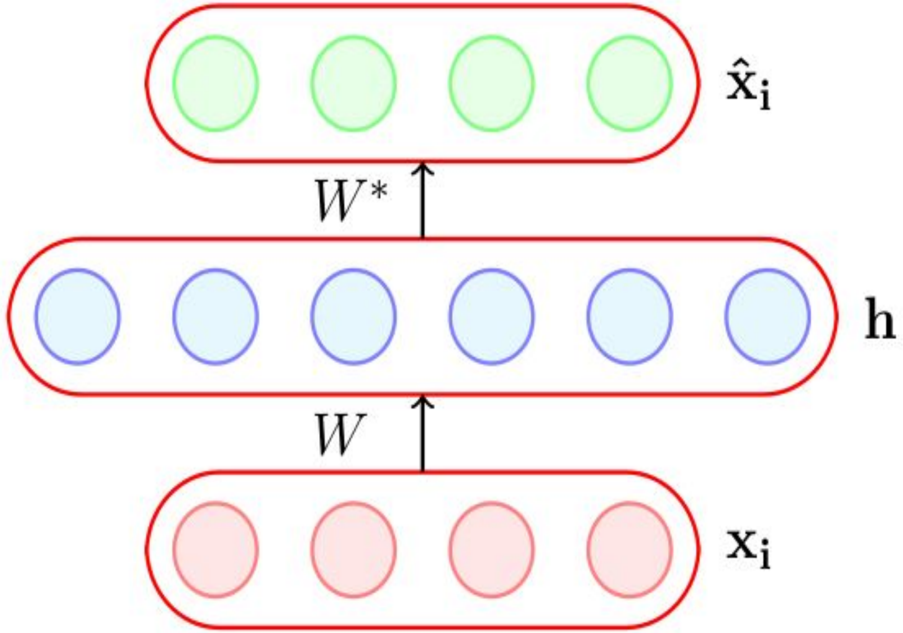


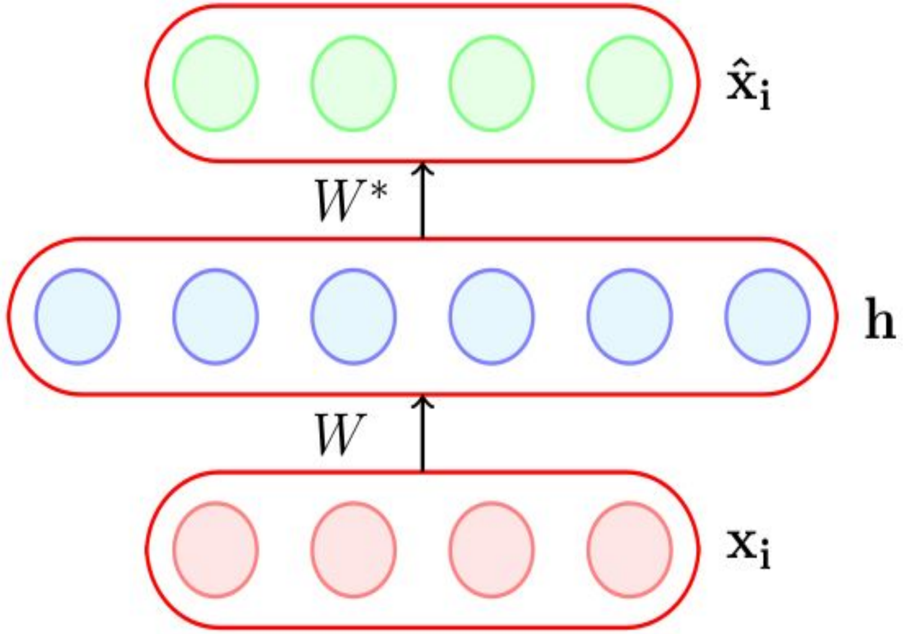
- The simplest solution is to add a L_2 regularization term to the objective function

$$\min_{\theta, w, w^*, \mathbf{b}, \mathbf{c}} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2 + \lambda \|\theta\|^2$$

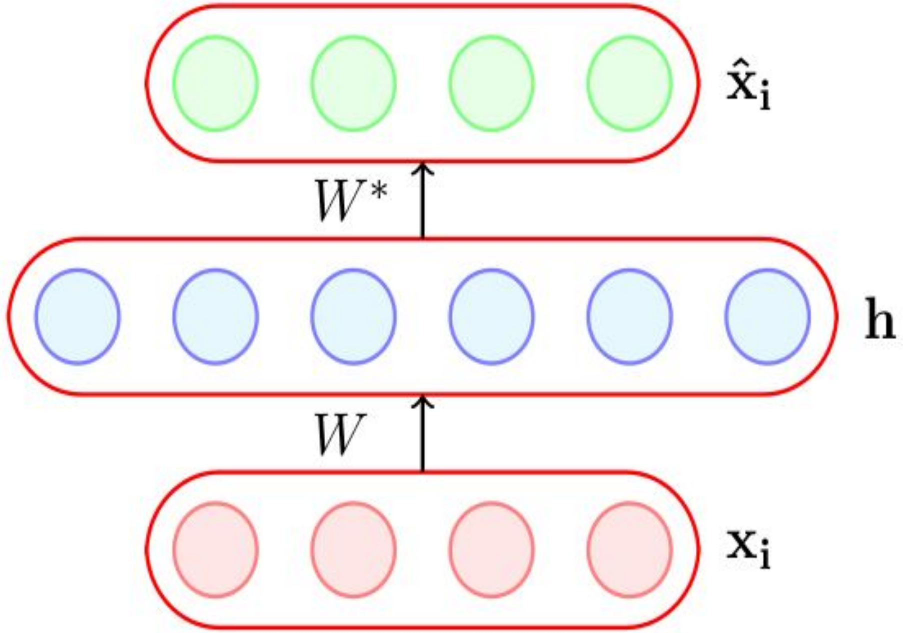
- This is very easy to implement and just adds a term ***λW*** to the gradient $\frac{\partial \mathscr{L}(\theta)}{\partial W}$ (and similarly for other parameters)

- Another trick is to tie the weights of the encoder and decoder





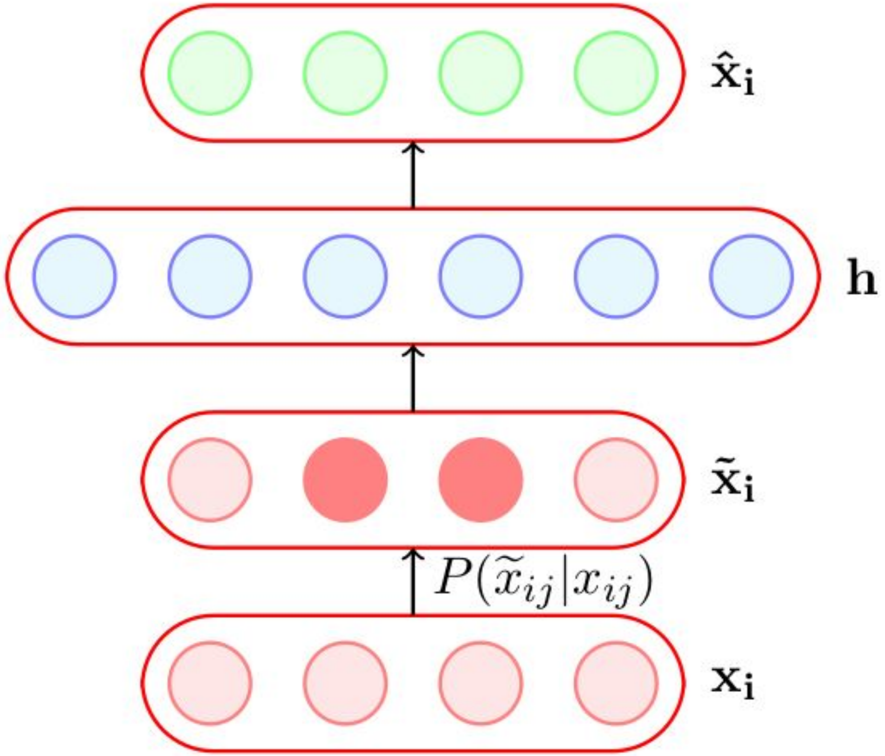
- Another trick is to tie the weights of the encoder and decoder i.e., $W^* = W^T$



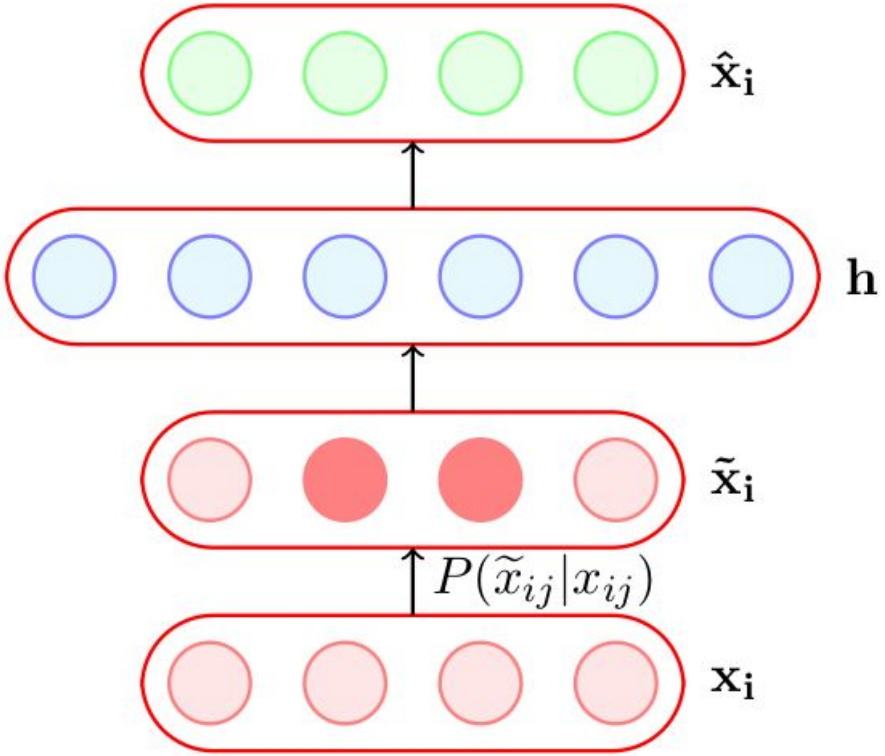
- Another trick is to tie the weights of the encoder and decoder i.e., $W^* = W^T$

- This effectively reduces the capacity of Autoencoder and acts as a regularizer

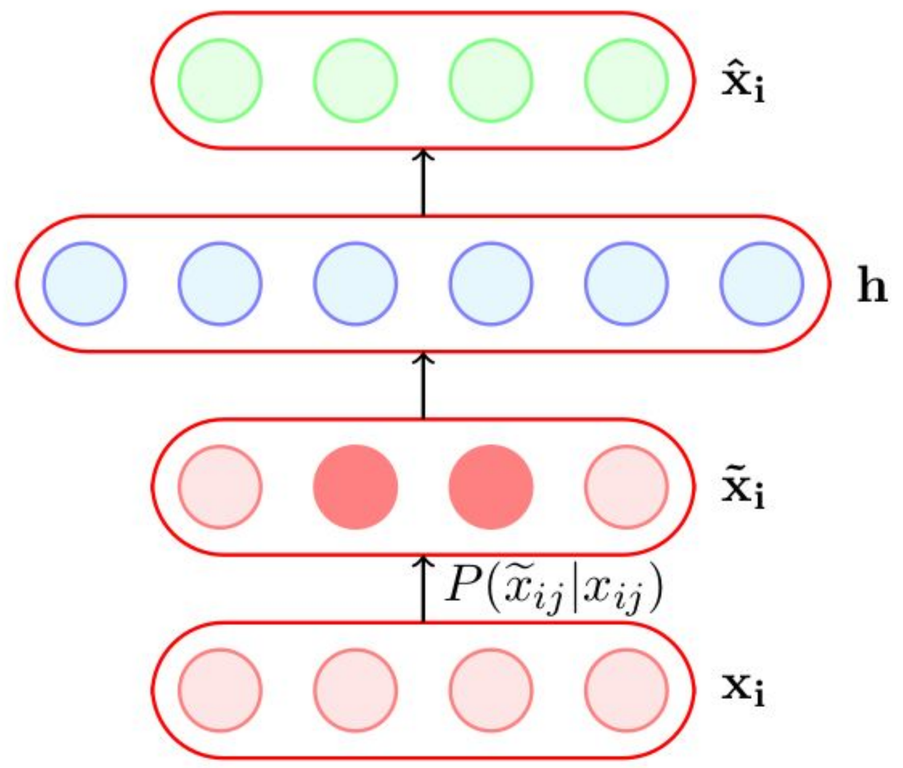
Denoising Autoencoders


$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\widetilde{x}_{ij}|x_{ij})$)before feeding it to the network

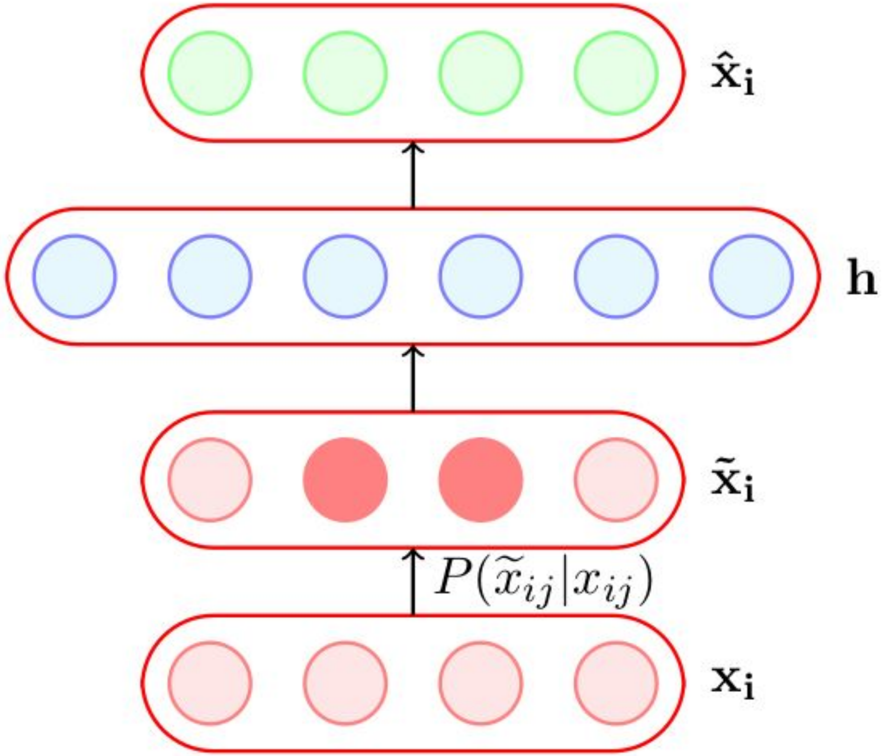

$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\widetilde{x}_{ij}|x_{ij})$)before feeding it to the network
- A simple ($P(\widetilde{x}_{ij}|x_{ij})$) used in practice is the following


$\hat{\mathbf{x}}_{\mathbf{i}}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_{\mathbf{i}}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_{\mathbf{i}}$

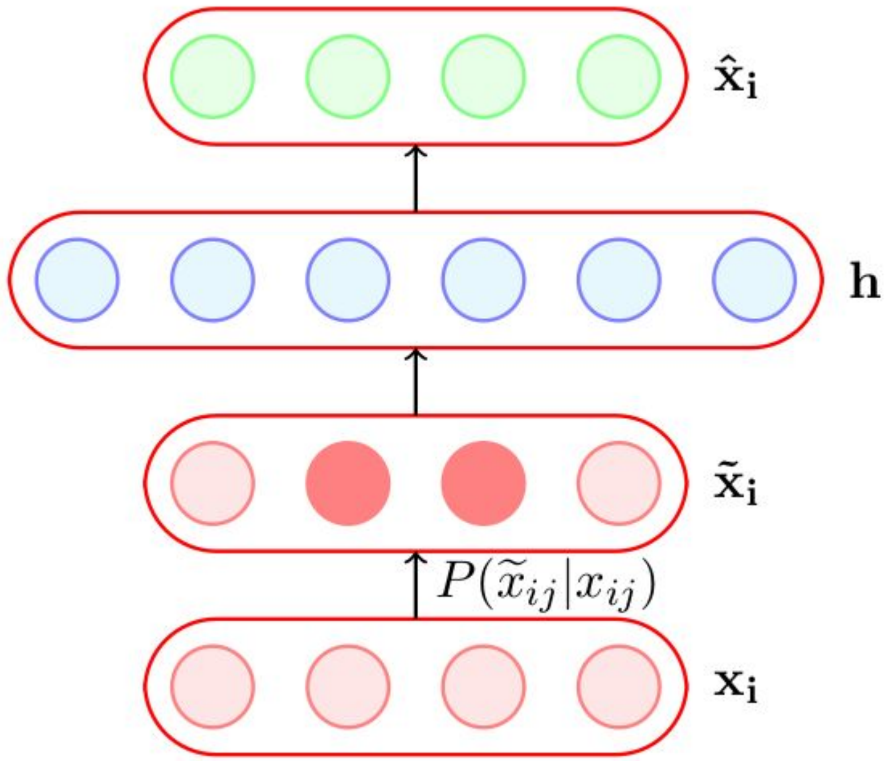
- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\widetilde{x}_{ij}|x_{ij})$)before feeding it to the network
- A simple ($P(\widetilde{x}_{ij}|x_{ij})$) used in practice is the following

$$P(\widetilde{x}_{ij} = 0|x_{ij}) = q$$


$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

- A denoising encoder simply corrupts the input data using a probabilistic process (*$P(\widetilde{x}_{ij}|x_{ij})$*)before feeding it to the network
- A simple (*$P(\widetilde{x}_{ij}|x_{ij})$*) used in practice is the following

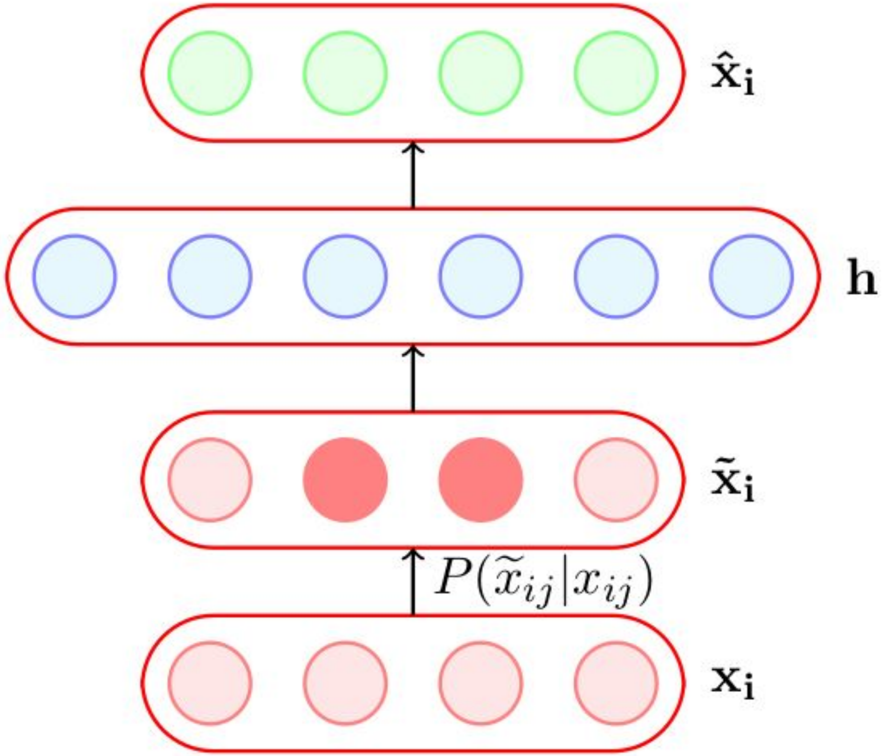
$$P(\widetilde{x}_{ij} = 0|x_{ij}) = q$$
$$P(\widetilde{x}_{ij} = x_{ij}|x_{ij}) = 1 - q$$


$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

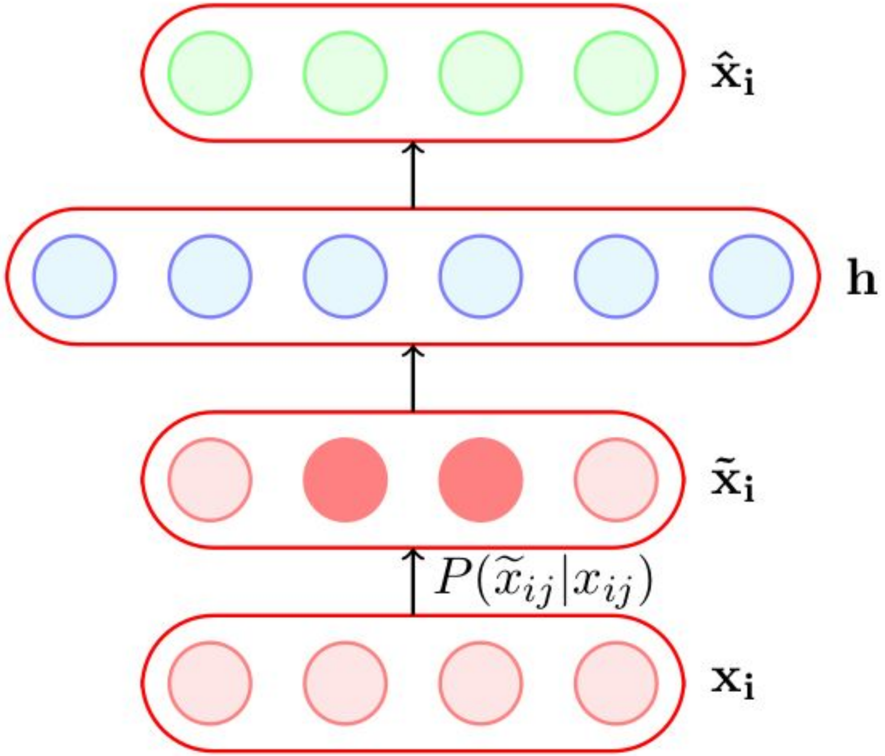
- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\widetilde{x}_{ij}|x_{ij})$)before feeding it to the network
- A simple ($P(\widetilde{x}_{ij}|x_{ij})$) used in practice is the following

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

- In other words, with probability *q* the input is flipped to 0 and with probability (1 − *q*) it is retained as it is

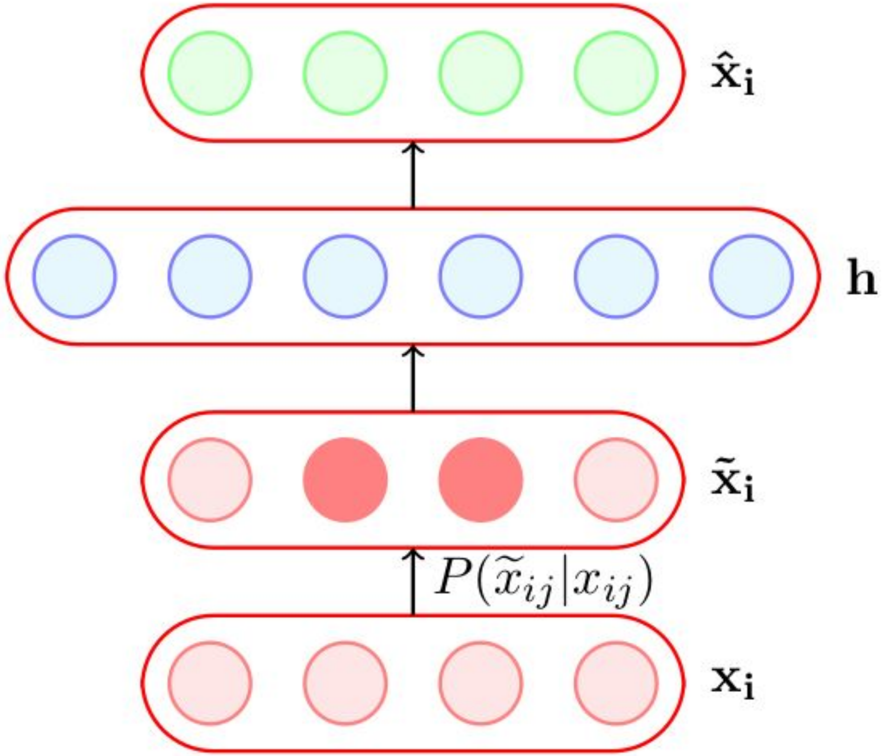

$\hat{\mathbf{x}}_{\mathbf{i}}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_{\mathbf{i}}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_{\mathbf{i}}$

- How does this help?


$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

- How does this help?
- This helps because the objective is still to reconstruct the original (uncorrupted) $\mathbf{x}_\mathbf{i}$

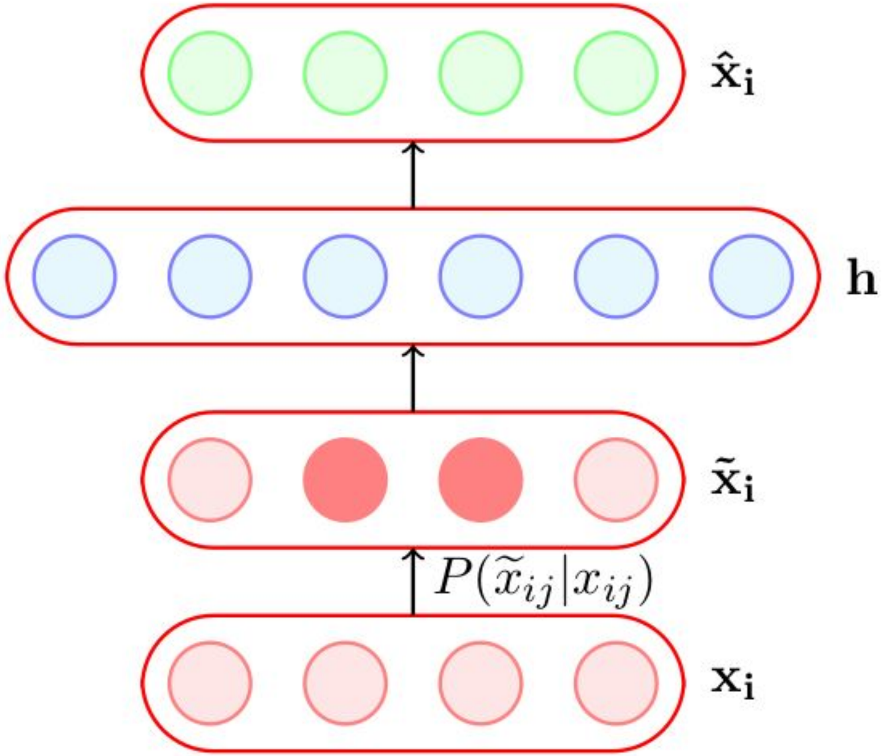
$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$


$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

- How does this help?
- This helps because the objective is still to reconstruct the original (uncorrupted) $\mathbf{x}_i$

$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$

- It no longer makes sense for the model to copy the corrupted $\widetilde{x}_i$ into h($\widetilde{x}_i$) and then into $\hat{x}_i$ (the objective function will not be minimized by doing so)

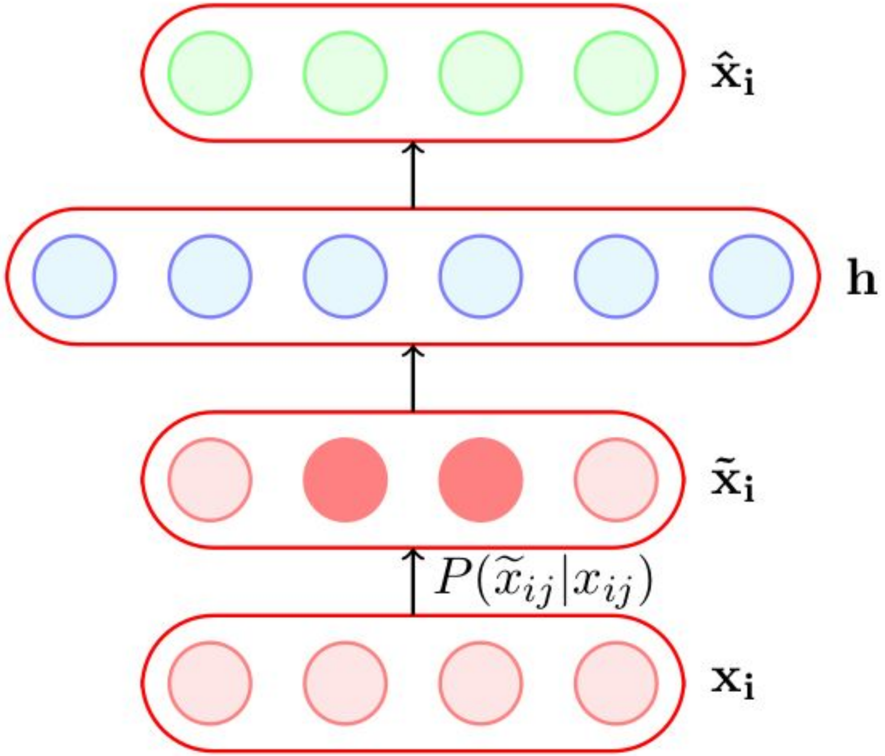


$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

- How does this help?
- This helps because the objective is still to reconstruct the original (uncorrupted) $\mathbf{x}_\mathbf{i}$

$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$

- It no longer makes sense for the model to copy the corrupted $\widetilde{x}_i$ into h($\widetilde{x}_i$)and then into $\hat{x}_i$ (the objective function will not be minimized by doing so)
- Instead the model will now have to capture the characteristics of the data correctly.


$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

- How does this help?
- This helps because the objective is still to reconstruct the original (uncorrupted) $\mathbf{x}_\mathbf{i}$

$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$

- It no longer makes sense for the model to copy the corrupted x_i into h($\widetilde{x}_i$) and th$\widetilde{x}_i$ into $\hat{x}_i$ (the objective function will not be minimized by doing so)
- Instead the model will now have to capture the characteristics of the data correctly

  For example, it will have to learn to reconstruct a corrupted x_{ij} correctly by relying on its interactions with other elements of x_i

We will now see a practical application in which AEs are used and then compare Denoising Autoencoders with regular autoencoders

Task: Hand-written digit recognition

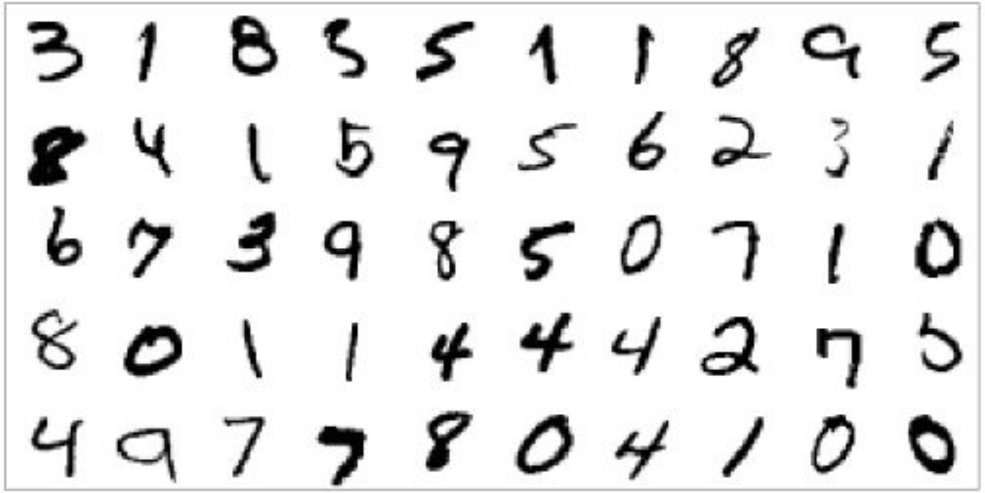


Figure: MNIST data

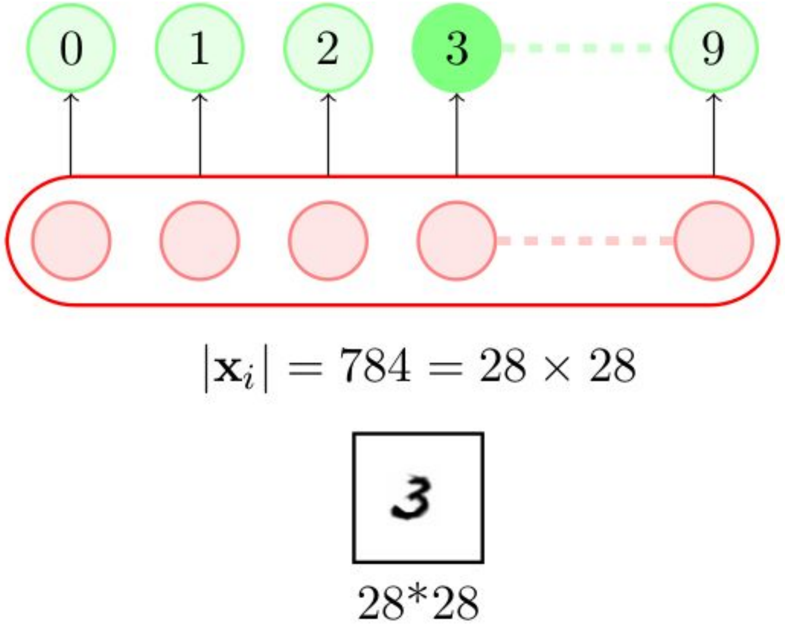


Figure: Basic approach(we use raw data as input features)

Task: Hand-written digit recognition

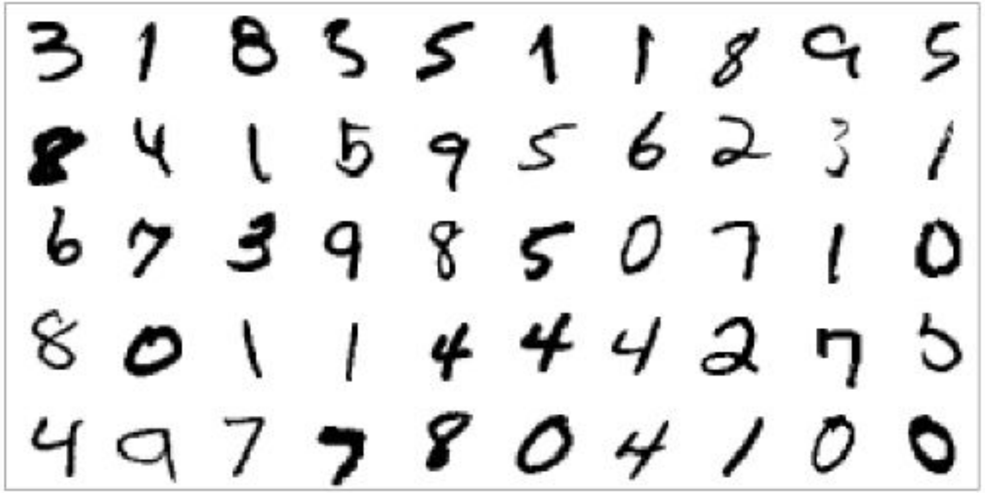
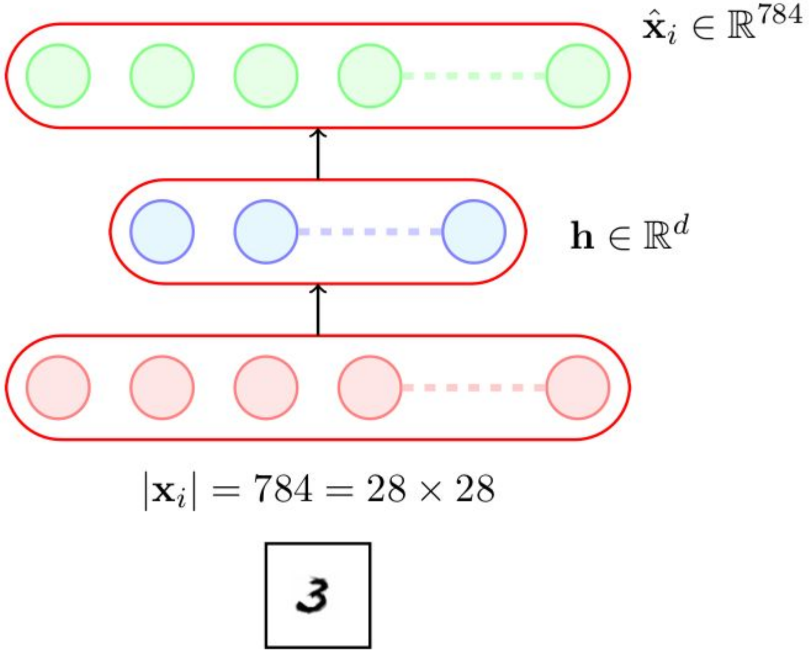


Figure: MNIST data



$\hat{\mathbf{x}}_i \in \mathbb{R}^{784}$
$\mathbf{h} \in \mathbb{R}^d$
$|\mathbf{x}_i| = 784 = 28 \times 28$
3

Figure: AE approach (first learn important characteristics of data)

Task: Hand-written digit recognition

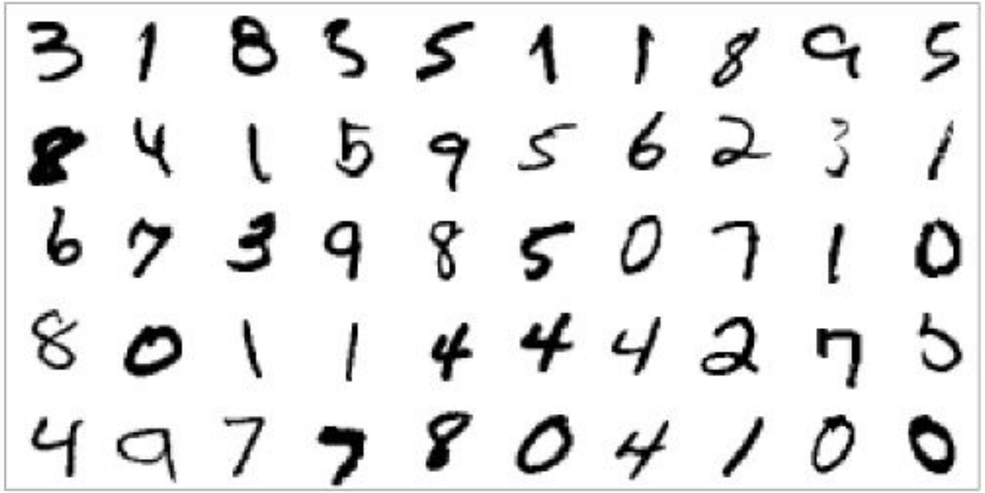


Figure: MNIST data

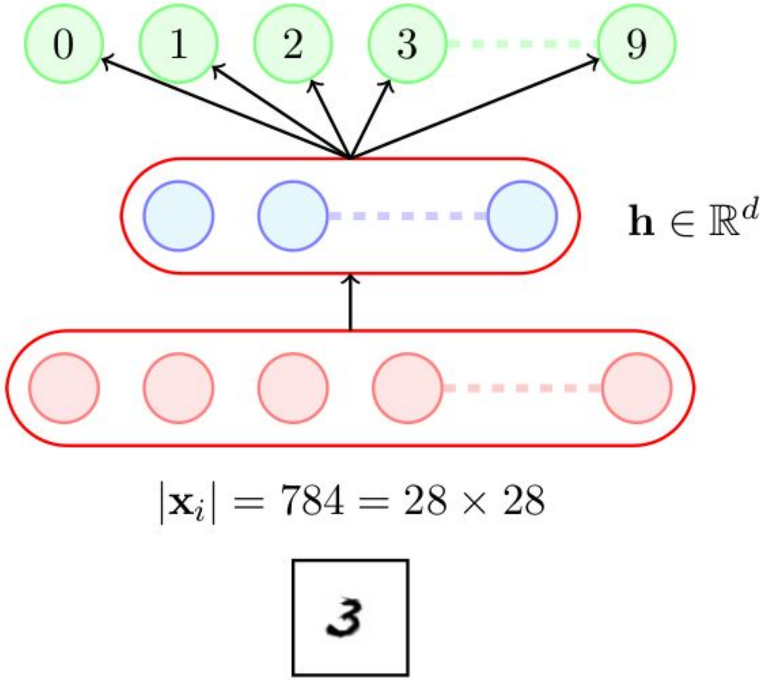
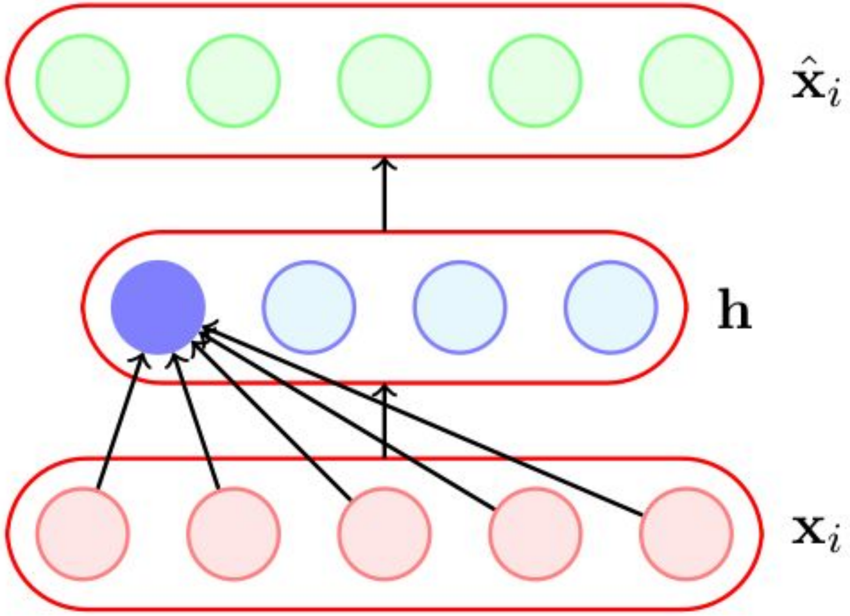
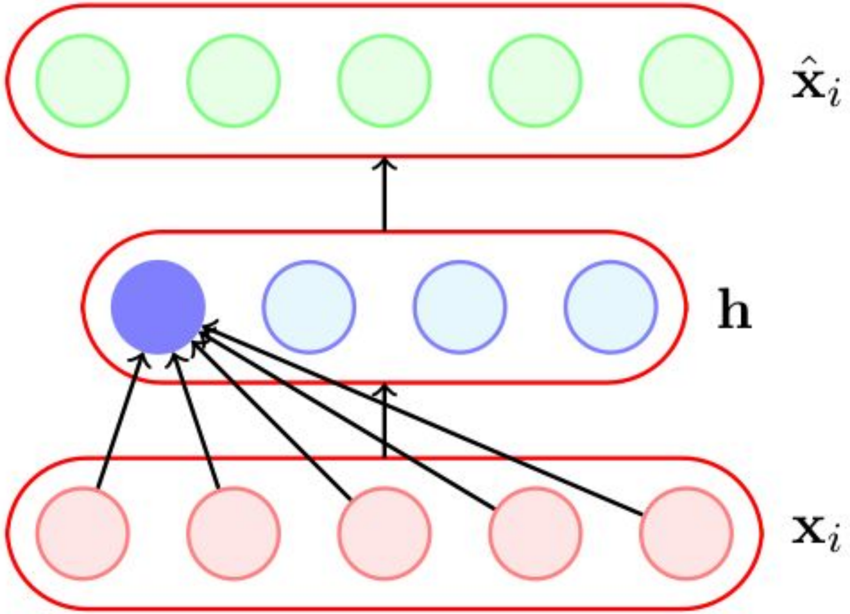

0
1
2
3
9
$\mathbf{h} \in \mathbb{R}^d$
$|\mathbf{x}_i| = 784 = 28 \times 28$
3

Figure: AE approach (and then train a classifier on top of this hidden representation)

We will now see a way of visualizing AEs and use this visualization to compare different AEs



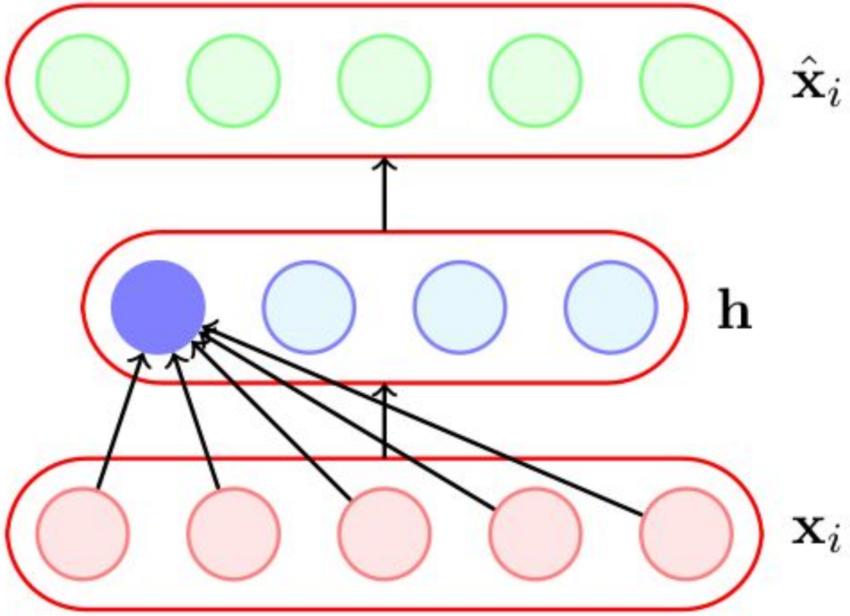
- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration x_i



- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration x_i
- For example,

$$\mathbf{h}_1 = \sigma(W_1^T \mathbf{x}_i) \quad \text{[ignoring bias } b\text{]}$$

Where W_1 is the trained vector of weights connecting the input to the first hidden neuron

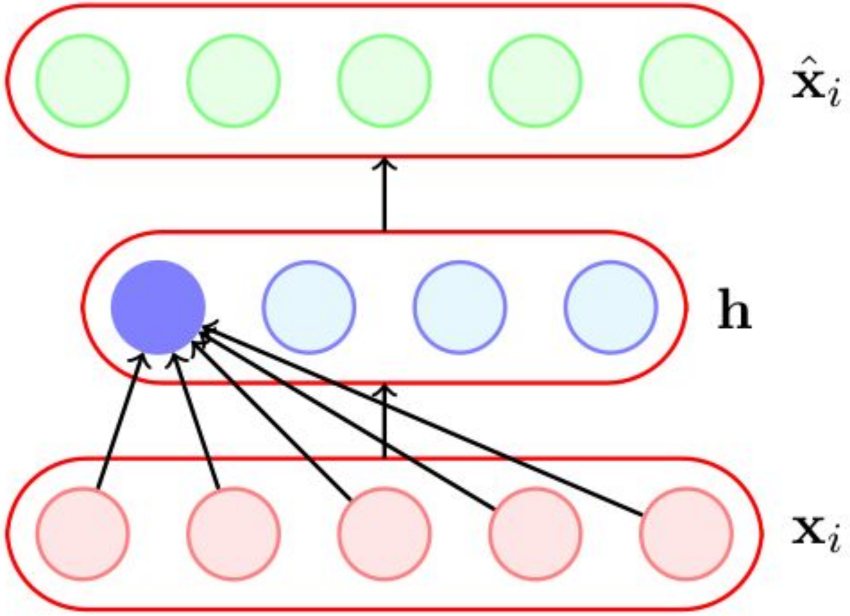


- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration x_i
- For example,

$$\mathbf{h}_1 = \sigma(W_1^T \mathbf{x}_i) \quad \text{[ignoring bias } b\text{]}$$

Where W_1 is the trained vector of weights connecting the input to the first hidden neuron

- What values of x_i will cause h_i to be maximum (or maximally activated)

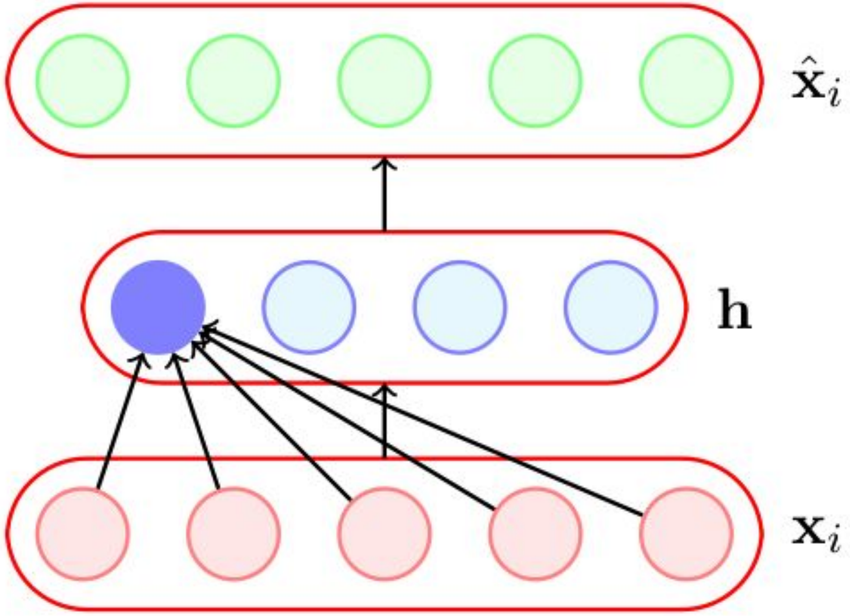


- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration x_i
- For example,

$$\mathbf{h}_1 = \sigma(W_1^T \mathbf{x}_i) \quad \text{[ignoring bias } b\text{]}$$

Where W_1 is the trained vector of weights connecting the input to the first hidden neuron

- What values of x_i will cause h_i to be maximum (or maximally activated)
- Suppose we assume that our inputs are normalized so that $||x_i|| = 1$



$$\max_{\mathbf{x}_i} \quad \{W_1^T \mathbf{x}_i\}$$

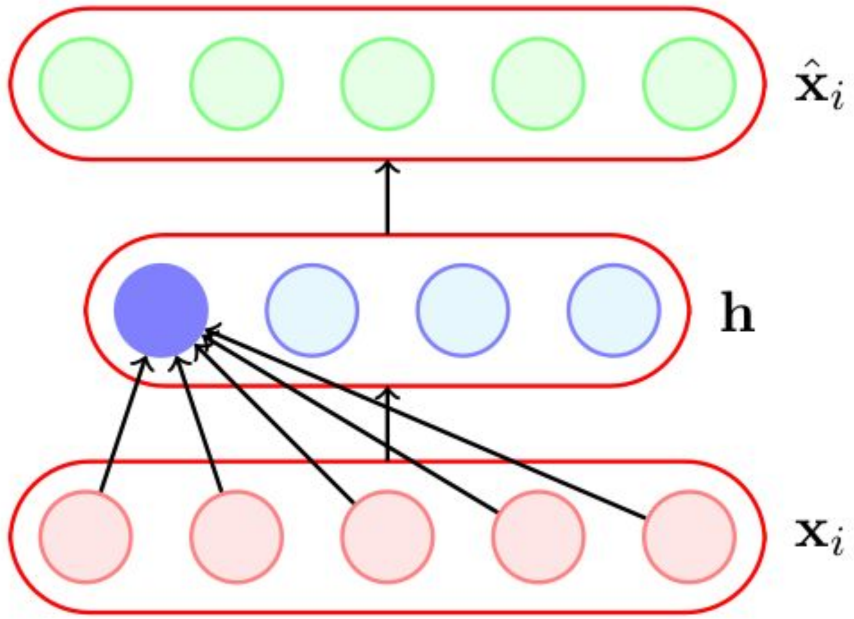
$$s.t. \quad ||\mathbf{x}_i||^2 = \mathbf{x}_i^T \mathbf{x}_i = 1$$

- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration x_i
- For example,

$$\mathbf{h}_1 = \sigma(W_1^T \mathbf{x}_i) \quad \text{[ignoring bias } b\text{]}$$

Where W_1 is the trained vector of weights connecting the input to the first hidden neuron

- What values of x_i will cause h_i to be maximum (or maximally activated)
- Suppose we assume that our inputs are normalized so that $||x_i|| = 1$



$$\max_{\mathbf{x}_i} \quad \{W_1^T \mathbf{x}_i\}$$

$$s.t. \quad ||\mathbf{x}_i||^2 = \mathbf{x}_i^T \mathbf{x}_i = 1$$

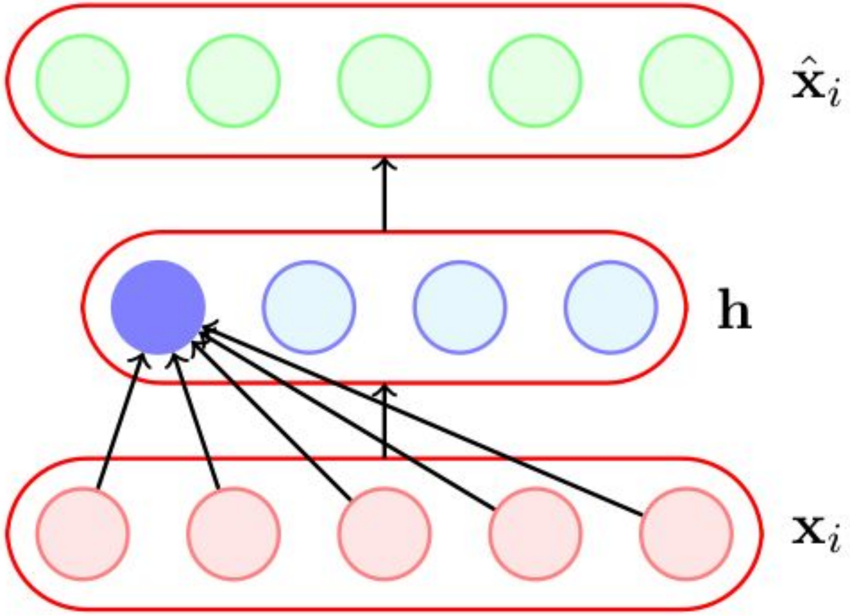
$$\text{Solution:} \quad \mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}}$$

- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration x_i
- For example,

$$\mathbf{h}_1 = \sigma(W_1^T \mathbf{x}_i) \quad \text{[ignoring bias } b\text{]}$$

Where W_1 is the trained vector of weights connecting the input to the first hidden neuron

- What values of x_i will cause h_i to be maximum (or maximally activated)
- Suppose we assume that our inputs are normalized so that $||x_i|| = 1$



$$\max_{\mathbf{x}_i} \quad \{W_1^T \mathbf{x}_i\}$$

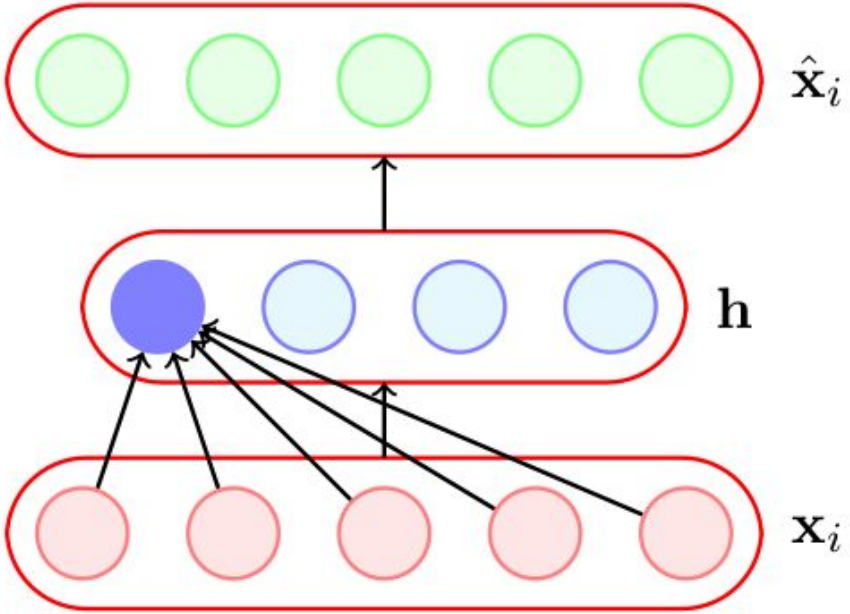
$$s.t. \quad ||\mathbf{x}_i||^2 = \mathbf{x}_i^T \mathbf{x}_i = 1$$

$$\text{Solution:} \quad \mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}}$$

- Thus the inputs

$$\mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}}, \frac{W_2}{\sqrt{W_2^T W_2}}, \dots \frac{W_n}{\sqrt{W_n^T W_n}}$$

will respectively cause hidden neurons 1 to n to maximally fire



$$\max_{\mathbf{x}_i} \quad \{W_1^T \mathbf{x}_i\}$$

$$s.t. \quad ||\mathbf{x}_i||^2 = \mathbf{x}_i^T \mathbf{x}_i = 1$$

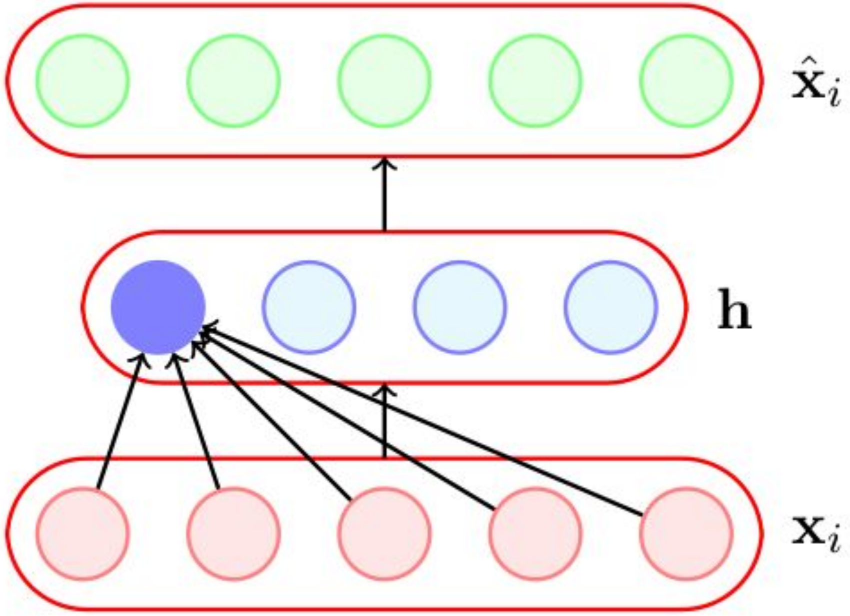
$$\text{Solution:} \quad \mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}}$$

- Thus the inputs

$$\mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}}, \frac{W_2}{\sqrt{W_2^T W_2}}, \ldots \frac{W_n}{\sqrt{W_n^T W_n}}$$

will respectively cause hidden neurons 1 to n to maximally fire

- Let us plot these images (x_i ’s) which maximally activate the first k neurons of the hidden representations learned by a vanilla autoencoder and different denoising autoencoders



$$\max_{\mathbf{x}_i} \ \{W_1^T \mathbf{x}_i\}$$

$$s.t. \ \ ||\mathbf{x}_i||^2 = \mathbf{x}_i^T \mathbf{x}_i = 1$$

$$\text{Solution:} \ \ \mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}}$$

- Thus the inputs

$$\mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}}, \frac{W_2}{\sqrt{W_2^T W_2}}, \dots \frac{W_n}{\sqrt{W_n^T W_n}}$$

  will respectively cause hidden neurons 1 to n to maximally fire
- Let us plot these images (x_i 's) which maximally activate the first k neurons of the hidden representations learned by a vanilla autoencoder and different denoising autoencoders
- These x_i 's are computed by the above formula using the weights (W_1, W_2 . . . W_k) learned by the respective autoencoders

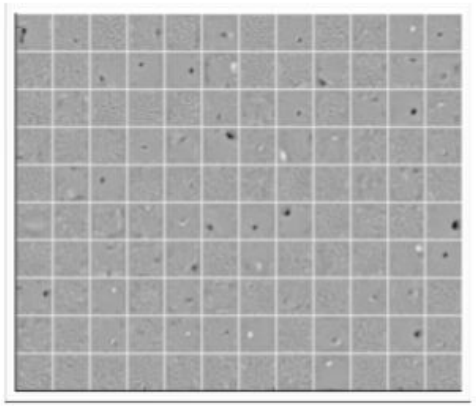


Figure: Vanilla AE (no noise)

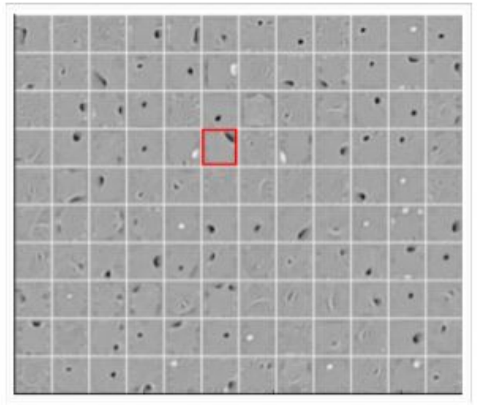


Figure: 25% Denoising AE (q=0.25)

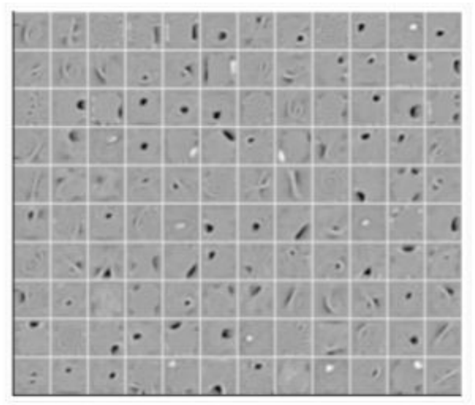


Figure: 50% Denoising AE (q=0.5)

- Vanilla AE does not learn many meaningful patterns

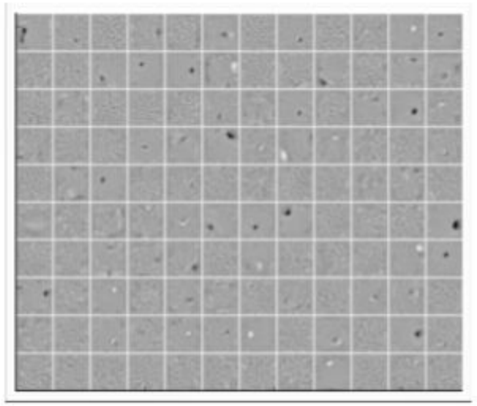


Figure: Vanilla AE (no noise)

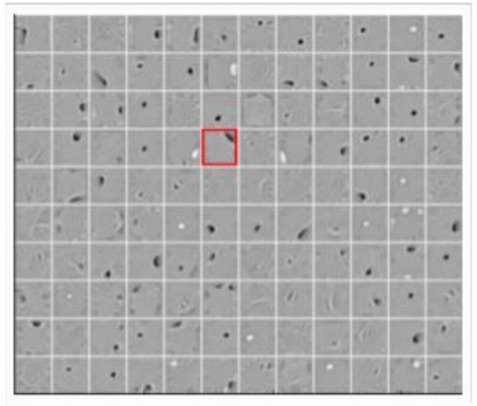


Figure: 25% Denoising AE (q=0.25)

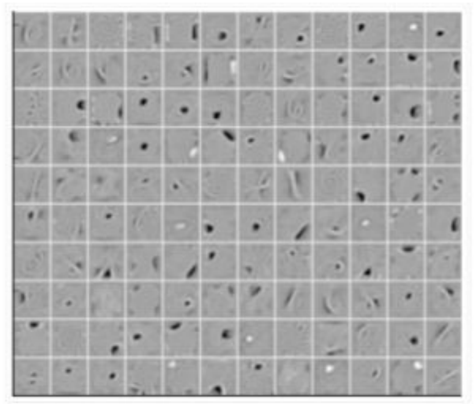


Figure: 50% Denoising AE (q=0.5)

- Vanilla AE does not learn many meaningful patterns
- The hidden neurons of the denoising AEs seem to act like pen-stroke detectors (for example, in the highlighted neuron the black region is a stroke that you would expect in a ’0’ or a ’2’ or a ’3’ or a ’8’ or a ’9’)

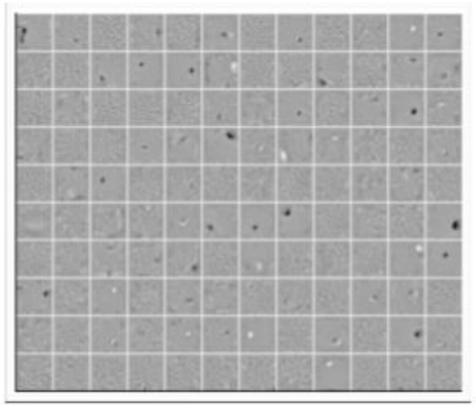


Figure: Vanilla AE (no noise)

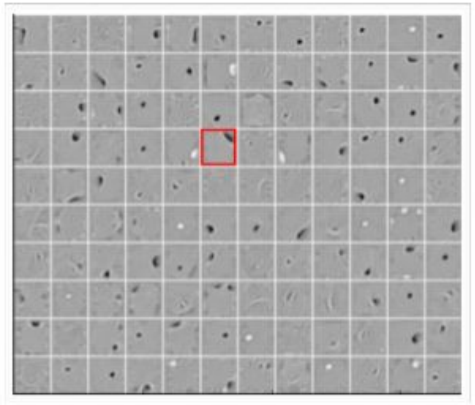


Figure: 25% Denoising AE (q=0.25)

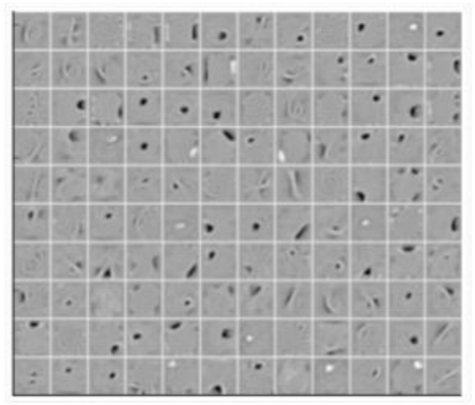
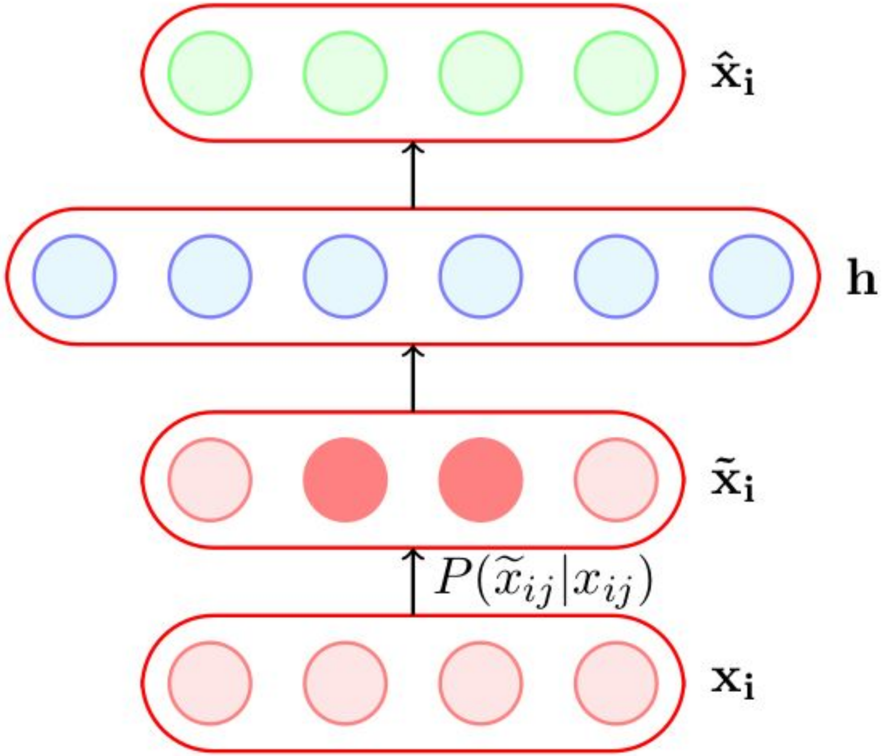


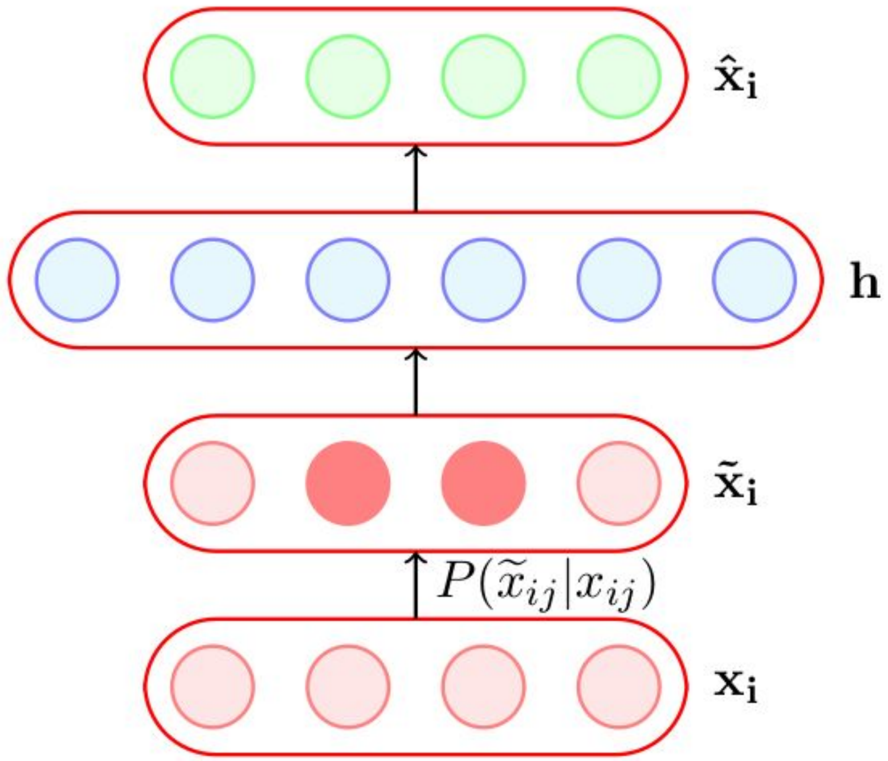
Figure: 50% Denoising AE (q=0.5)

- Vanilla AE does not learn many meaningful patterns
- The hidden neurons of the denoising AEs seem to act like pen-stroke detectors (for example, in the highlighted neuron the black region is a stroke that you would expect in a ’0’ or a ’2’ or a ’3’ or a ’8’ or a ’9’)
- As the noise increases the filters become more wide because the neuron has to rely on more adjacent pixels to feel confident about a stroke



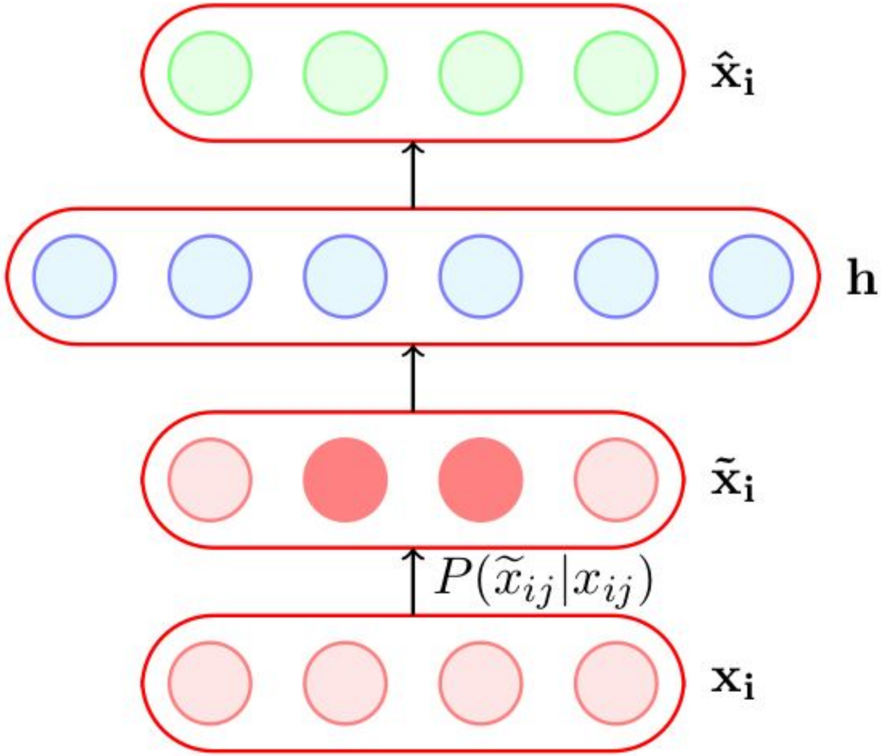
$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

- We saw one form of $P(\widetilde{x}_{ij} | x_{ij})$ which flips a fraction q of the inputs to zero


$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

- We saw one form of $P(\widetilde{x}_{ij} | x_{ij})$ which flips a fraction q of the inputs to zero
- Another way of corrupting the inputs is to add a Gaussian noise to the input

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

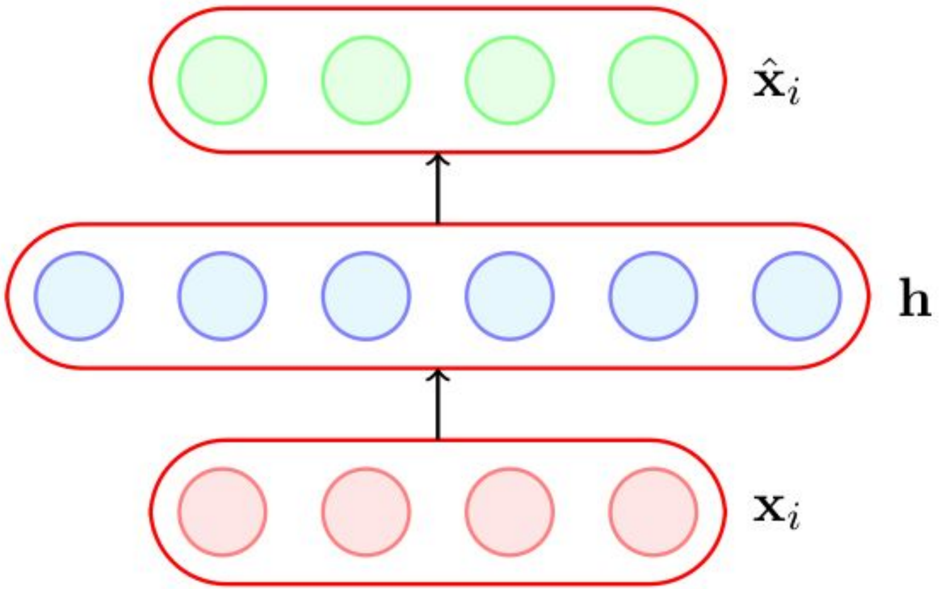

$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

- We saw one form of P($\widetilde{x}_{ij}$ | x_{ij}) which flips a fraction q of the inputs to zero
- Another way of corrupting the inputs is to add a Gaussian noise to the input

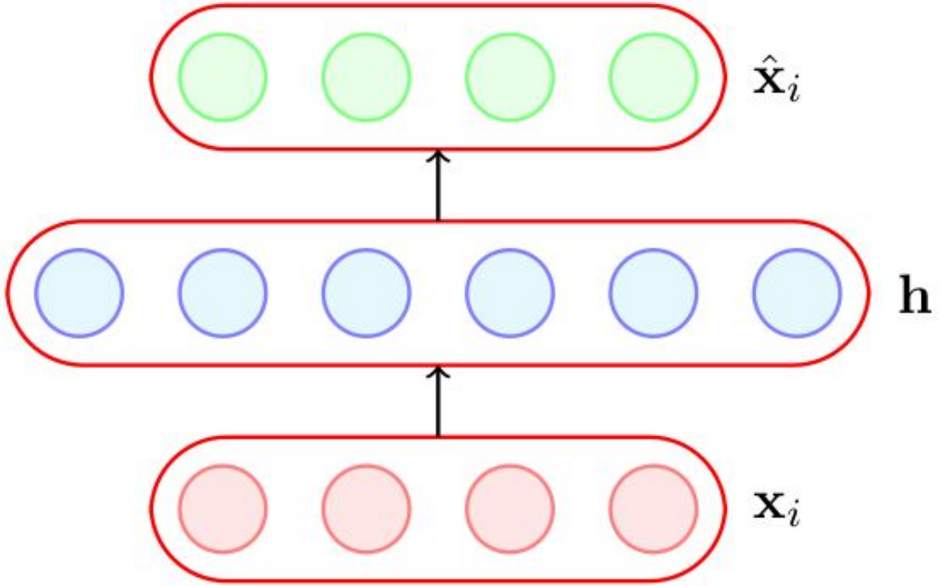
$$\widetilde{x}_{ij} = x_{ij} + \mathcal{N}(0,1)$$

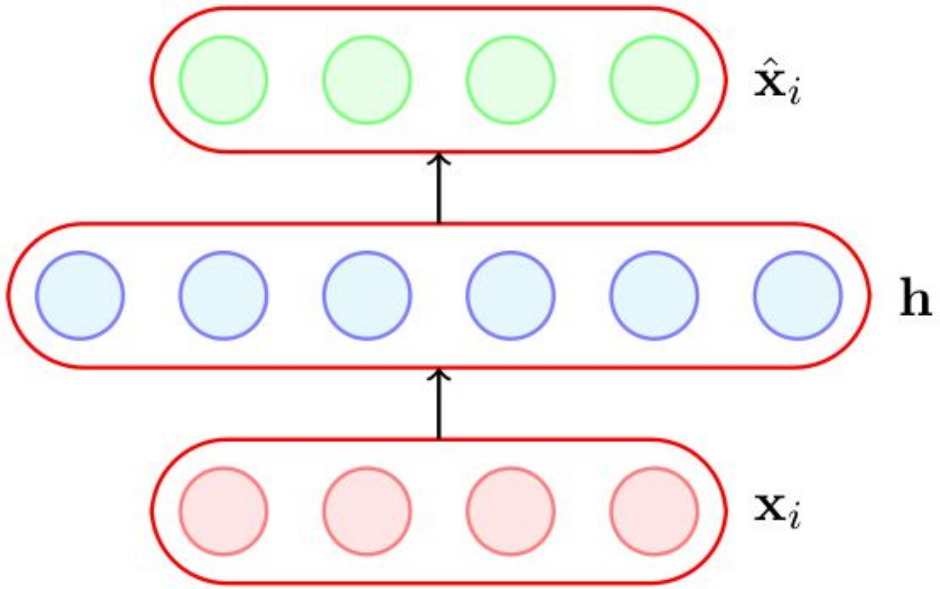
- We will now use such a denoising AE on a different dataset and see their performance

Sparse Autoencoders

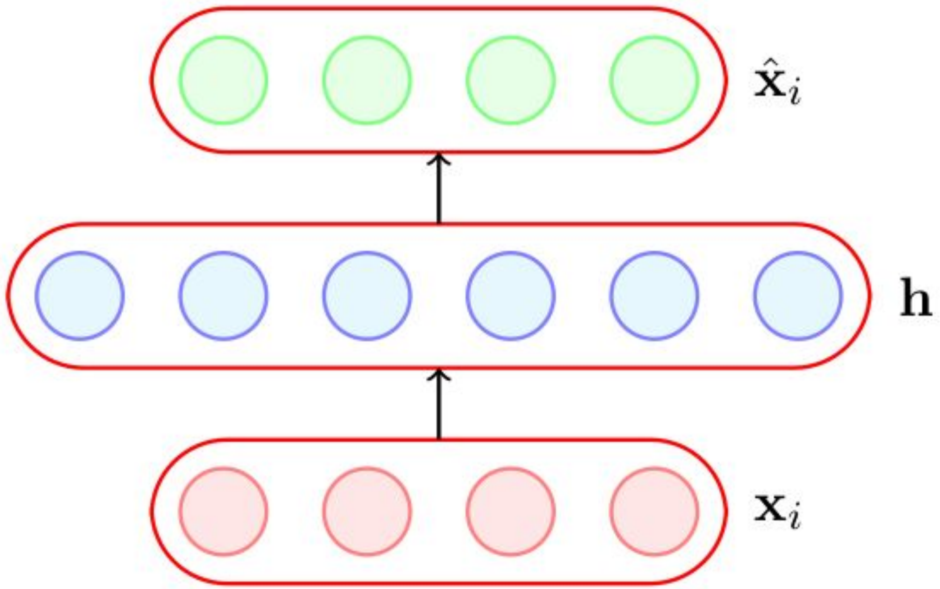

$\hat{\mathbf{x}}_i$
$\mathbf{h}$
$\mathbf{x}_i$

- A hidden neuron with sigmoid activation will have values between 0 and 1

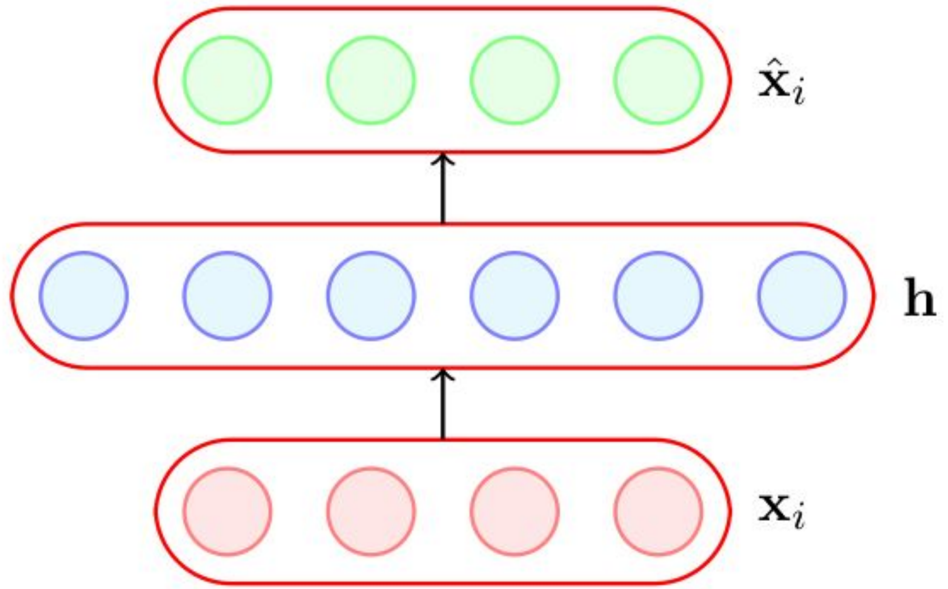




- A hidden neuron with sigmoid activation will have values between 0 and 1
- We say that the neuron is activated when its output is close to 1 and not activated when its output is close to 0.



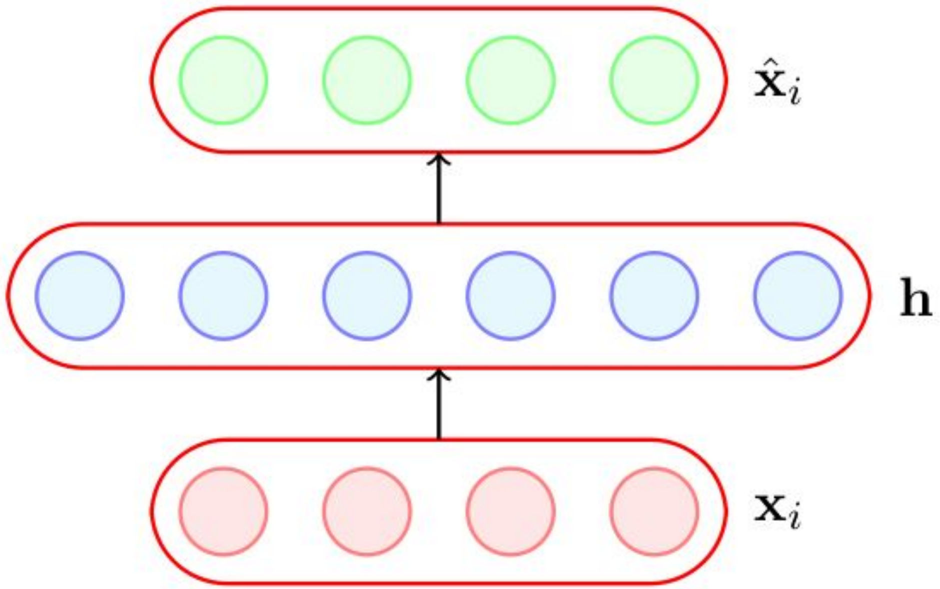
- A hidden neuron with sigmoid activation will have values between 0 and 1
- We say that the neuron is activated when its output is close to 1 and not activated when its output is close to 0.
- A sparse autoencoder tries to ensure that the neuron is inactive most of the times.



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

$$\hat{\rho}_l = \frac{1}{m}\sum_{i=1}^{m} h(\mathbf{x}_i)_l$$

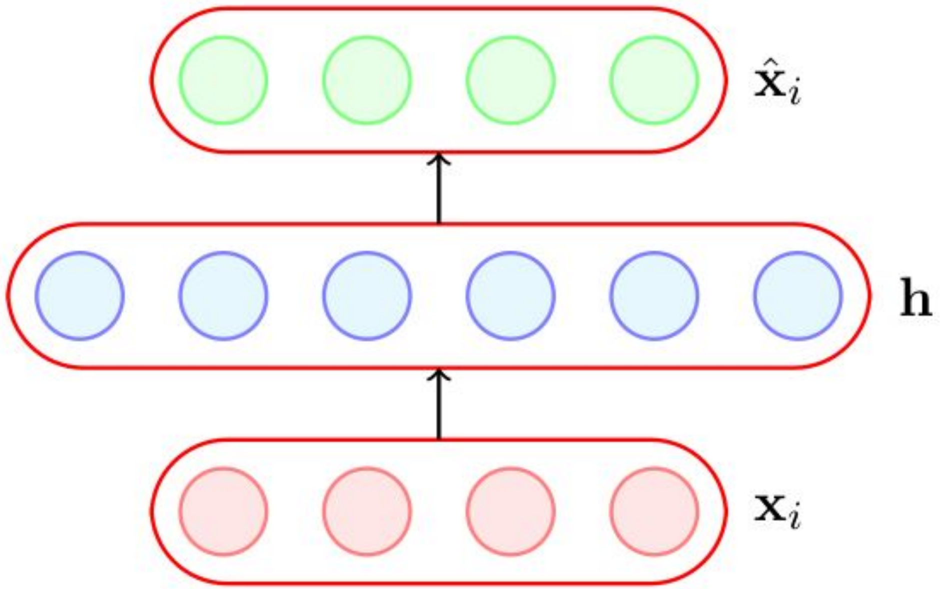
- If the neuron l is sparse (i.e. mostly inactive) then $\hat{\rho}_l \to 0$



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

$$\hat{\rho}_l = \frac{1}{m}\sum_{i=1}^{m} h(\mathbf{x}_i)_l$$

- If the neuron l is sparse (i.e. mostly inactive) then $\hat{\rho}_l \rightarrow 0$
- A sparse autoencoder uses a sparsity parameter ρ (typically very close to 0, say, 0.005) and tries to enforce the constraint $\hat{\rho}_l = \rho$

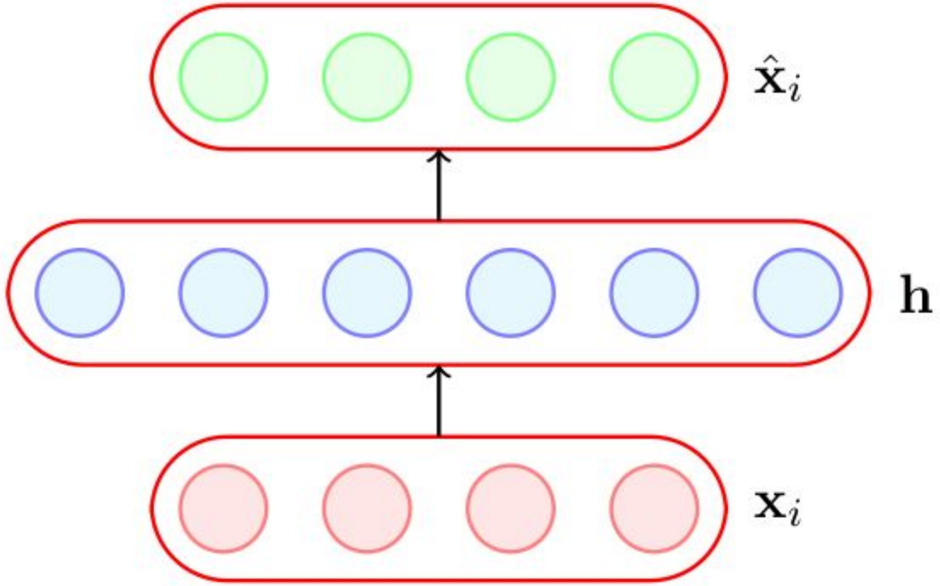


The average value of the activation of a neuron l is given by

$$\hat{\rho}_l = \frac{1}{m}\sum_{i=1}^{m} h(\mathbf{x}_i)_l$$

- If the neuron l is sparse (i.e. mostly inactive) then $\hat{\rho}_l \rightarrow 0$
- A sparse autoencoder uses a sparsity parameter ρ (typically very close to 0, 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

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



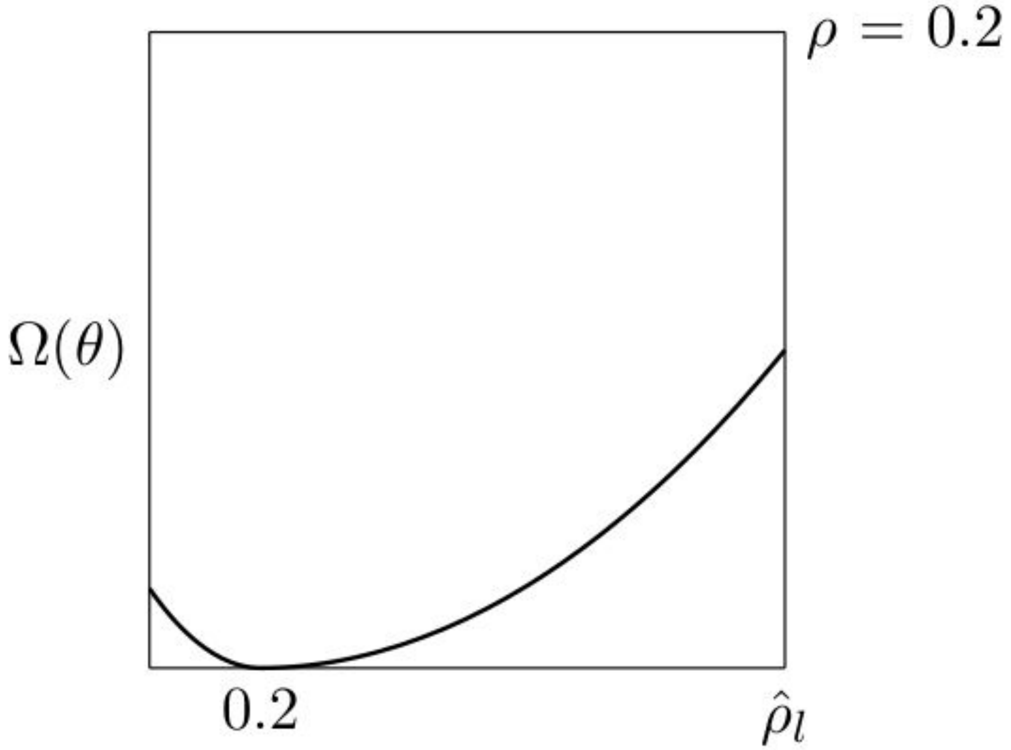
The average value of the activation of a neuron l is given by

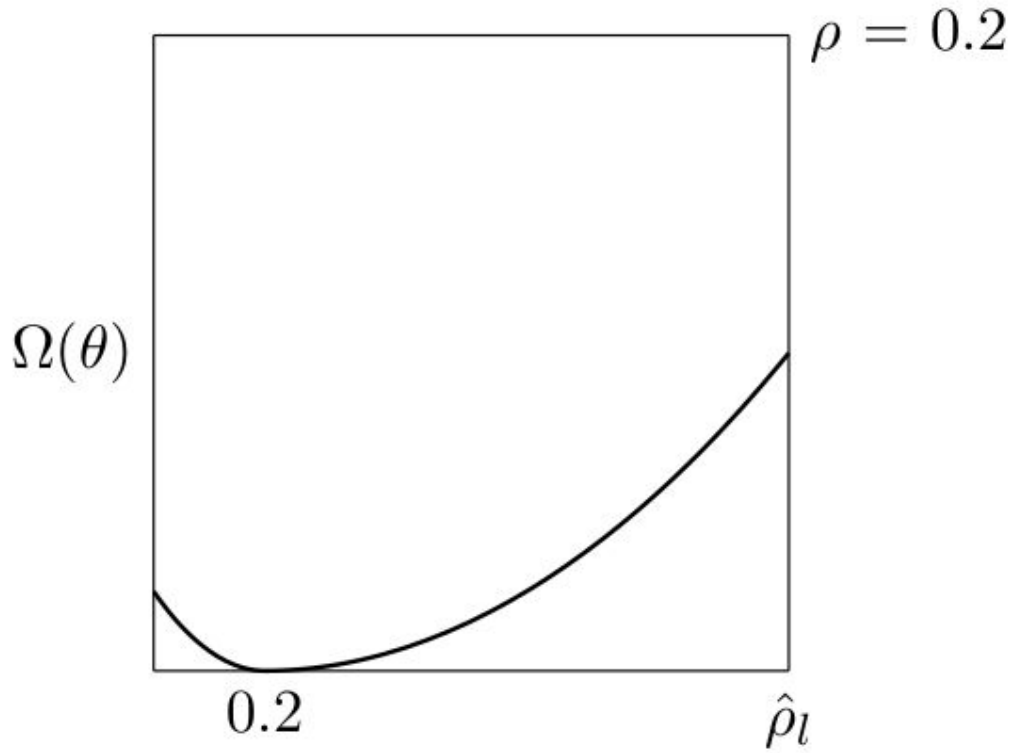
$$\hat{\rho}_l = \frac{1}{m}\sum_{i=1}^{m} h(\mathbf{x}_i)_l$$

- If the neuron l is sparse (i.e. mostly inactive) then $\hat{\rho}_l \rightarrow 0$
- A sparse autoencoder uses a sparsity parameter ρ (typically very close to 0, 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

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

- When will this term reach its minimum value and what is the minimum value? Let us plot it and check.


$\rho = 0.2$
$\Omega(\theta)$
0.2
$\hat{\rho}_l$



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

- Now,

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

- Now,

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

- $\mathscr{L}(\theta)$ is the squared error loss or cross entropy loss and Ω(θ) is the sparsity constraint.

- Now,

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

- $\mathscr{L}(\theta)$ is the squared error loss or cross entropy loss and Ω(θ) is the sparsity constraint.
- We already know how to calculate

  $\frac{\partial \mathscr{L}(\theta)}{\partial W}$

- Now,

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

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

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

- 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 $\frac{\partial \Omega(\theta)}{\partial W}$

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

Can be re-written as

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

- Now,

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

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

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

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:

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

- Now,

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

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

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

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:

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

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

- Now,

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

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

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

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 ∈ 1 . . . k in hidden layer, we have

- Now,

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

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

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

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 ∈ 1 . . . k in hidden layer, we have

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

- Now,

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

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

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

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:

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

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

For each neuron l ∈ 1 . . . 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 $\frac{\partial \hat{\rho}_l}{\partial W} = \mathbf{x}_i (g'(W^T \mathbf{x}_i + \mathbf{b}))^T$

- Now,

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

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

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

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 ∈ 1 . . . 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 $\frac{\partial \hat{\rho}_l}{\partial W} = \mathbf{x}_i (g'(W^T \mathbf{x}_i + \mathbf{b}))^T$

- Now,

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

- $\mathscr{L}(\theta)$ is the squared error loss or cross entropy loss and Ω(θ) 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:

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

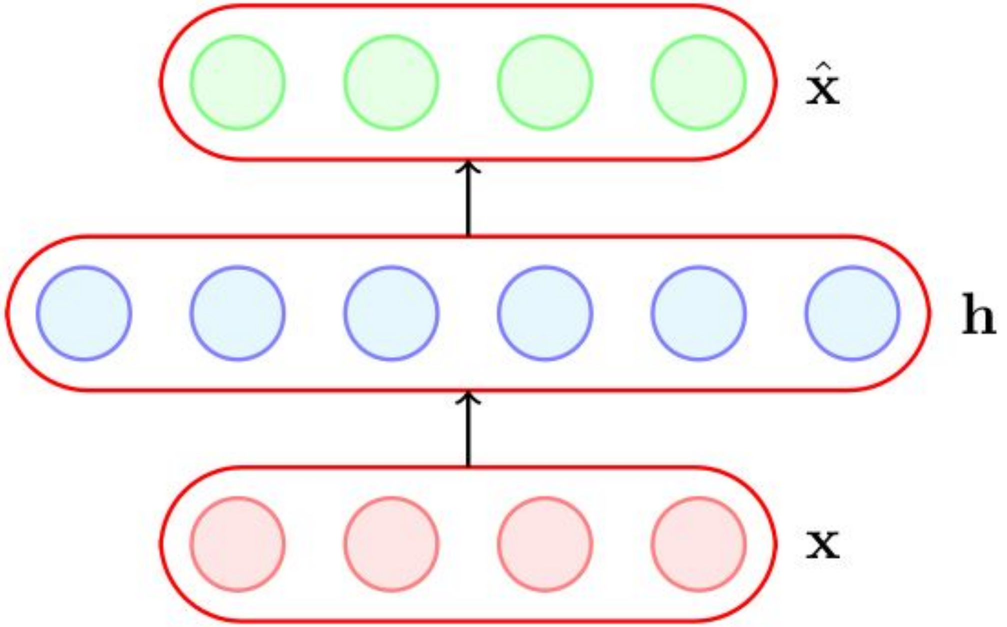
For each element in the above equation we can calculate $\frac{\partial \hat{\rho}_l}{\partial W}$ (which is the partial derivative of a scalar w.r.t. a matrix = matrix). For a single element of a matrix W_{jl} :-

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

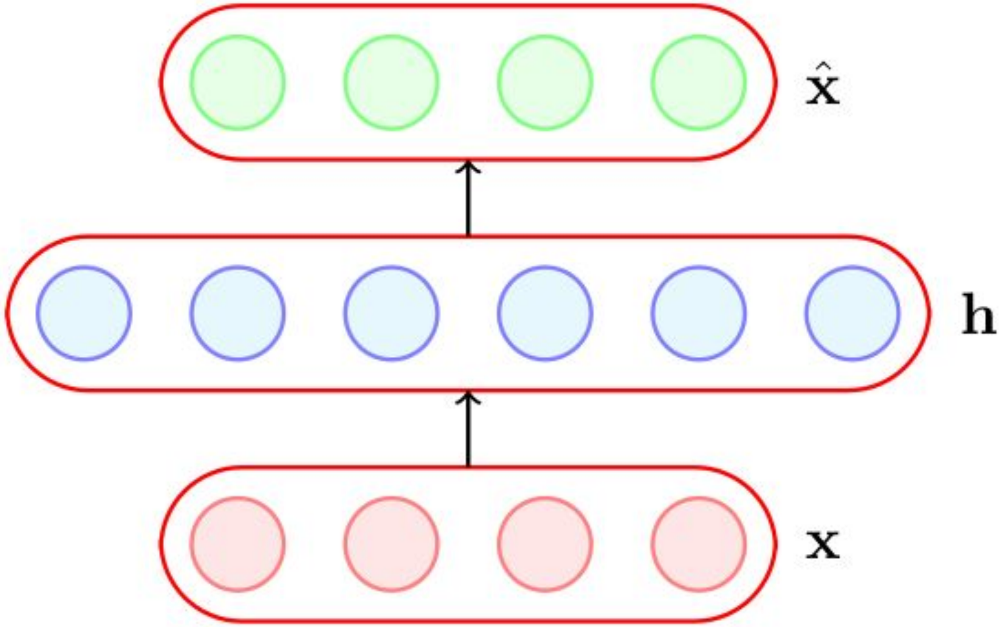
So in matrix notation we can write it as :

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

Contractive Autoencoders

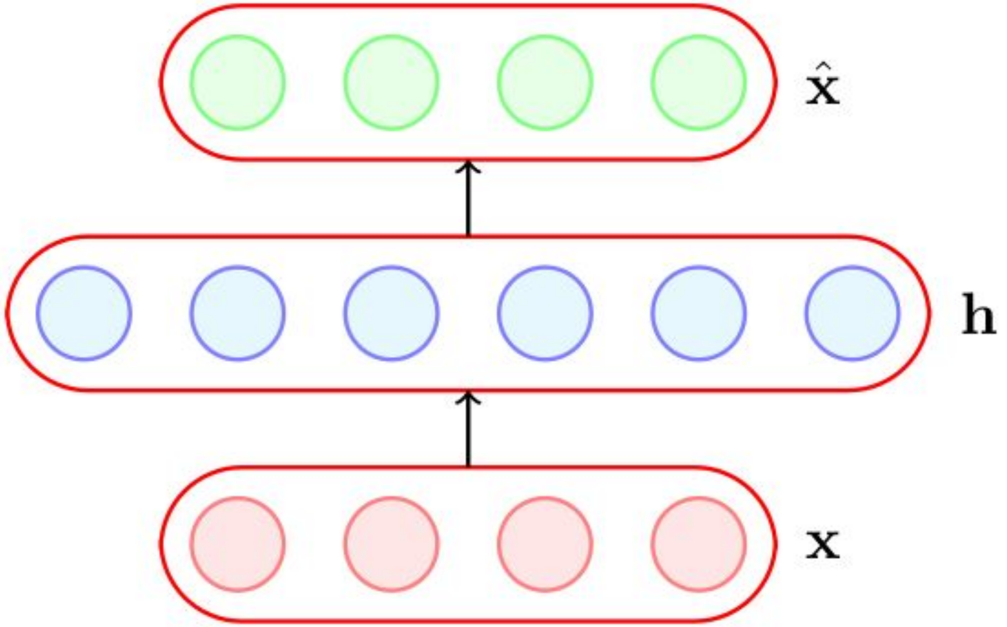


- A contractive autoencoder also tries to prevent an overcomplete autoencoder from learning the identity function.



- A contractive autoencoder also tries to prevent an overcomplete autoencoder from learning the identity function.
- It does so by adding the following regularization term to the loss function

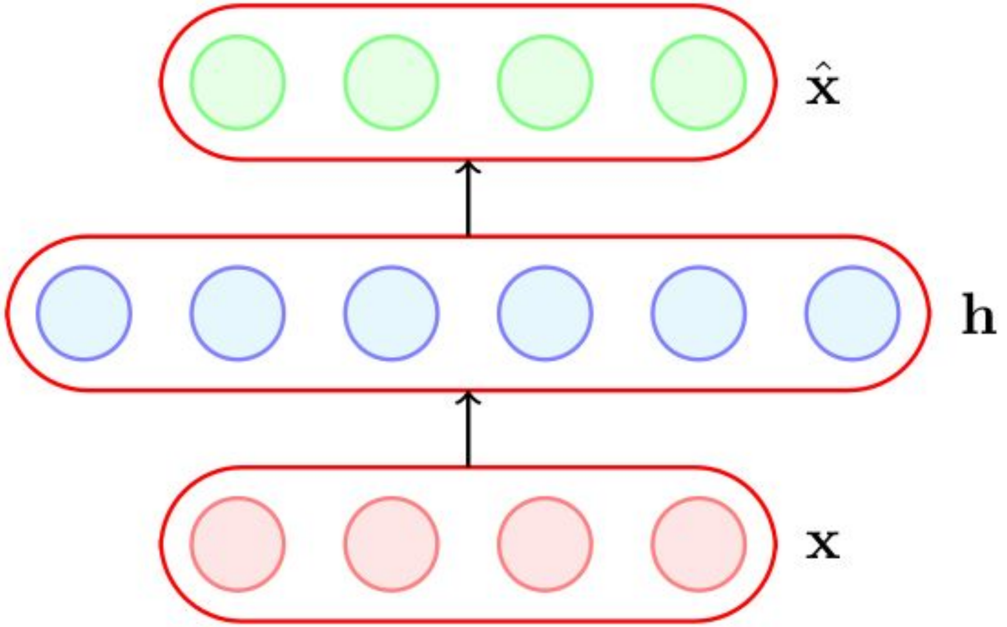
$$\Omega(\theta) = \|J_{\mathbf{x}}(\mathbf{h})\|_F^2$$



- A contractive autoencoder also tries to prevent an overcomplete autoencoder from learning the identity function.
- It does so by adding the following regularization term to the loss function

$$\Omega(\theta) = \|J_{\mathbf{x}}(\mathbf{h})\|_F^2$$

Where $J_x(\mathbf{h})$ is the Jacobian of the encoder



- A contractive autoencoder also tries to prevent an overcomplete autoencoder from learning the identity function.
- It does so by adding the following regularization term to the loss function

$$\Omega(\theta) = \|J_{\mathbf{x}}(\mathbf{h})\|_F^2$$

Where $J_x(\mathbf{h})$ is the Jacobian of the encoder

- Let us see what it looks like.

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

$$J_{\mathbf{x}}(\mathbf{h}) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_2}{\partial x_n} \\ \vdots & & \ddots & & \vdots \\ \frac{\partial h_k}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_k}{\partial x_n} \end{bmatrix}$$

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

$$J_{\mathbf{x}}(\mathbf{h}) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_2}{\partial x_n} \\ \vdots & & \ddots & & \vdots \\ \frac{\partial h_k}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_k}{\partial x_n} \end{bmatrix}$$

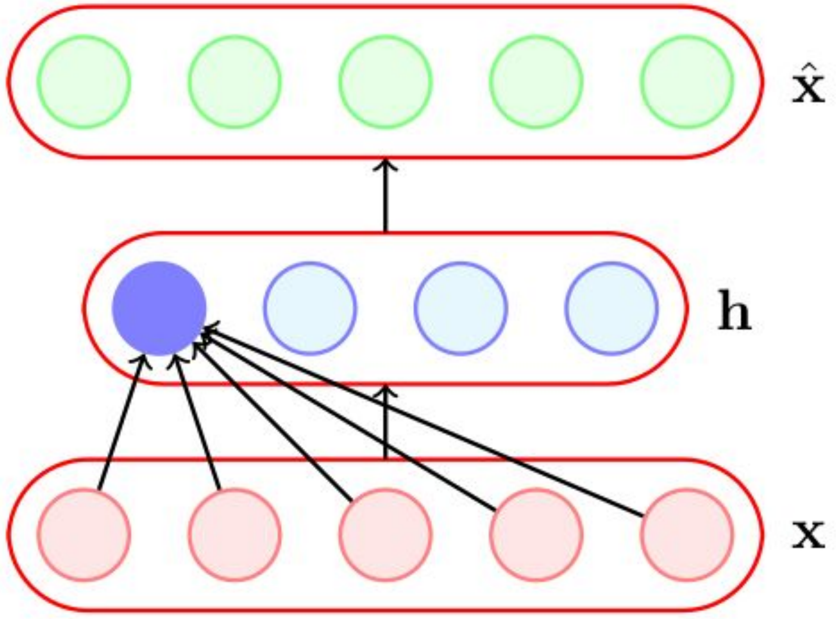
- If the input has n dimensions and the hidden layer has k dimensions then
- In other words, the (l, j) entry of the Jacobian captures the variation in the output of the lth neuron with a small variation in the jth input.

$$J_{\mathbf{x}}(\mathbf{h}) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_2}{\partial x_n} \\ \vdots & & \ddots & & \vdots \\ \frac{\partial h_k}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h_k}{\partial x_n} \end{bmatrix}$$

$$\|J_{\mathbf{x}}(\mathbf{h})\|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left(\frac{\partial h_l}{\partial x_j} \right)^2$$

- If the input has n dimensions and the hidden layer has k dimensions then
- In other words, the (l, j) entry of the Jacobian captures the variation in the output of the lth neuron with a small variation in the jth input.

$$\|J_{\mathbf{x}}(\mathbf{h})\|_F^2 = \sum_{j=1}^{n}\sum_{l=1}^{k}\left(\frac{\partial h_l}{\partial x_j}\right)^2$$

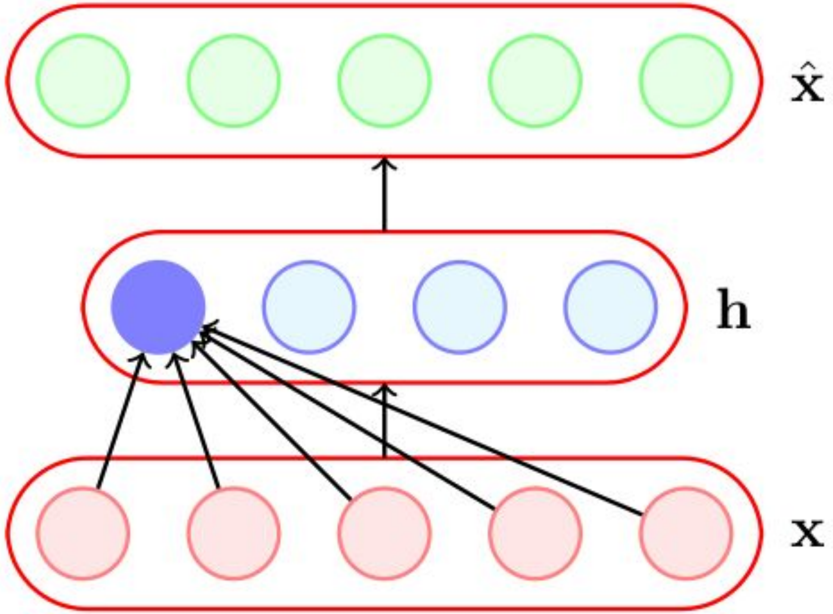


- What is the intuition behind this?

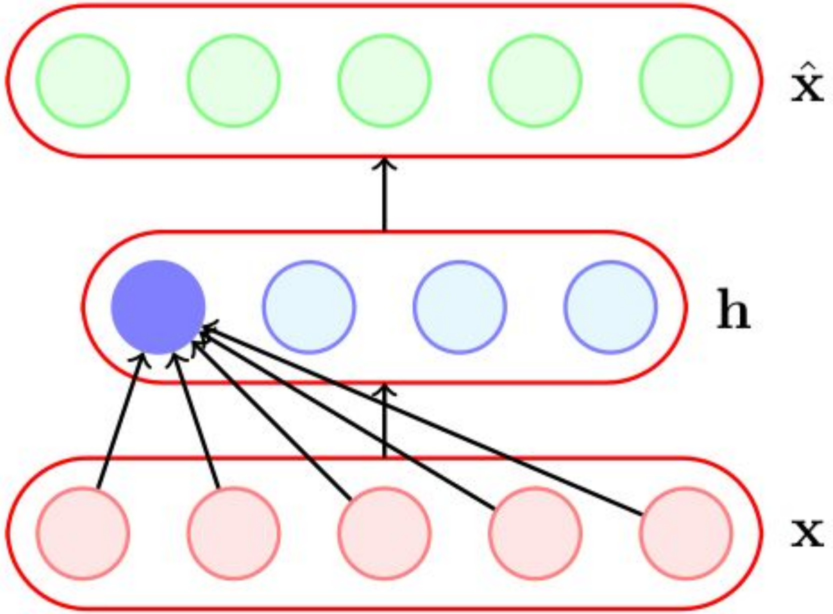
$$\|J_{\mathbf{x}}(\mathbf{h})\|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left(\frac{\partial h_l}{\partial x_j} \right)^2$$

- What is the intuition behind this?
- Consider $\frac{\partial h_1}{\partial x_1}$ what does it mean if

  $\frac{\partial h_1}{\partial x_1} = 0$



$$\|J_{\mathbf{x}}(\mathbf{h})\|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left(\frac{\partial h_l}{\partial x_j} \right)^2$$

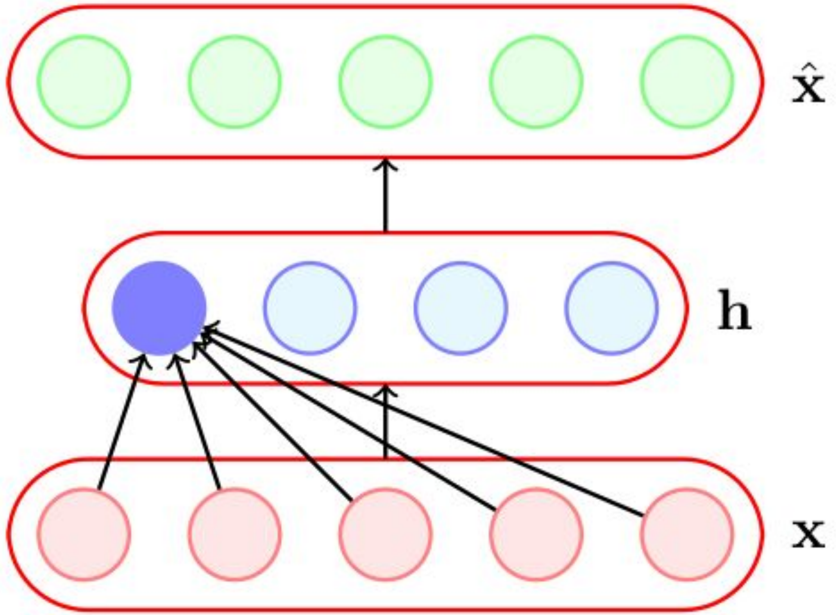


- What is the intuition behind this?
- Consider $\frac{\partial h_1}{\partial x_1}$ what does it mean if

  $\frac{\partial h_1}{\partial x_1} = 0$

- It means that this neuron is not very sensitive to variations in the input x_1

$$\|J_{\mathbf{x}}(\mathbf{h})\|_F^2 = \sum_{j=1}^{n}\sum_{l=1}^{k}\left(\frac{\partial h_l}{\partial x_j}\right)^2$$



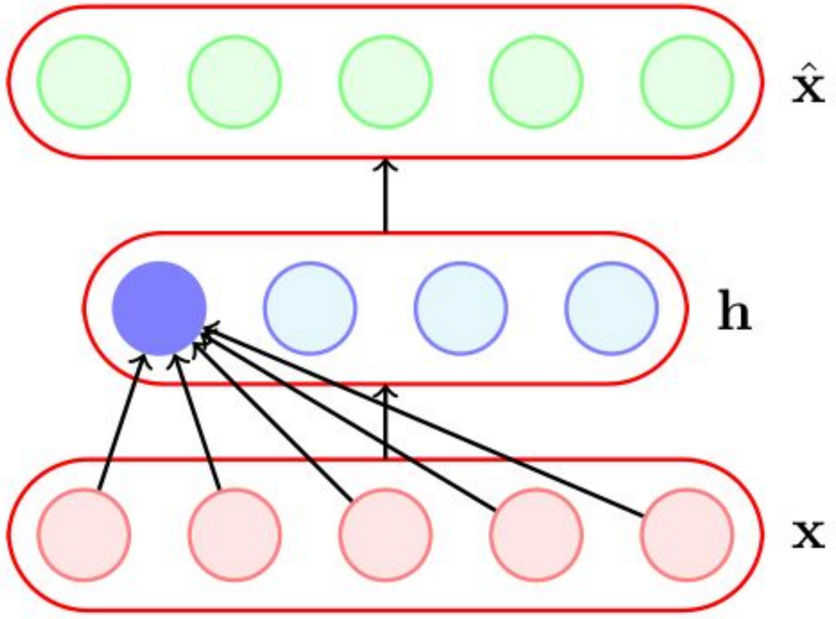
- What is the intuition behind this?
- Consider $\frac{\partial h_1}{\partial x_1}$ what does it mean if

  $\frac{\partial h_1}{\partial x_1}$ = 0

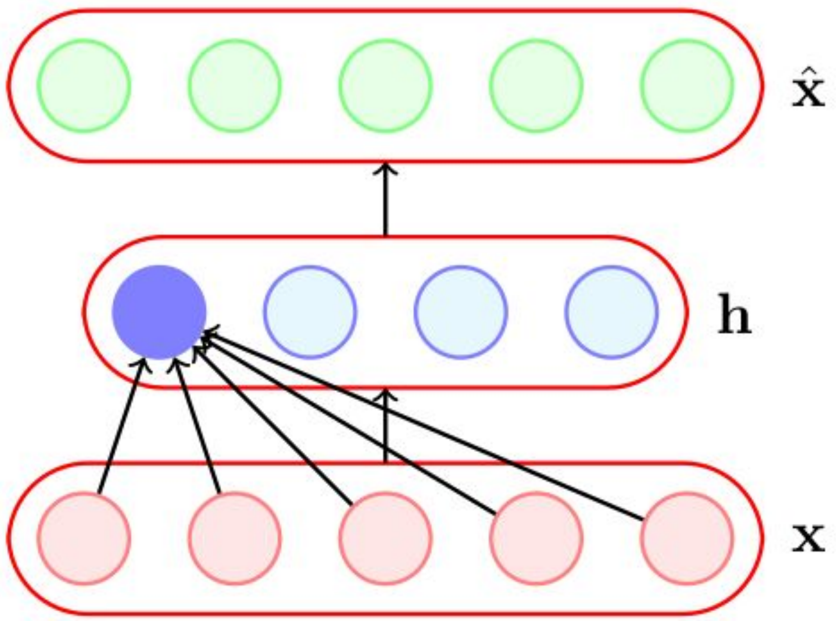
- It means that this neuron is not very sensitive to variations in the input x_1
- But doesn't this contradict our other goal of minimizing $\mathcal{L}(\theta)$ which requires h to capture variations in the input.

- Indeed it does and that’s the idea

$$\|J_{\mathbf{x}}(\mathbf{h})\|_F^2 = \sum_{j=1}^{n}\sum_{l=1}^{k}\left(\frac{\partial h_l}{\partial x_j}\right)^2$$

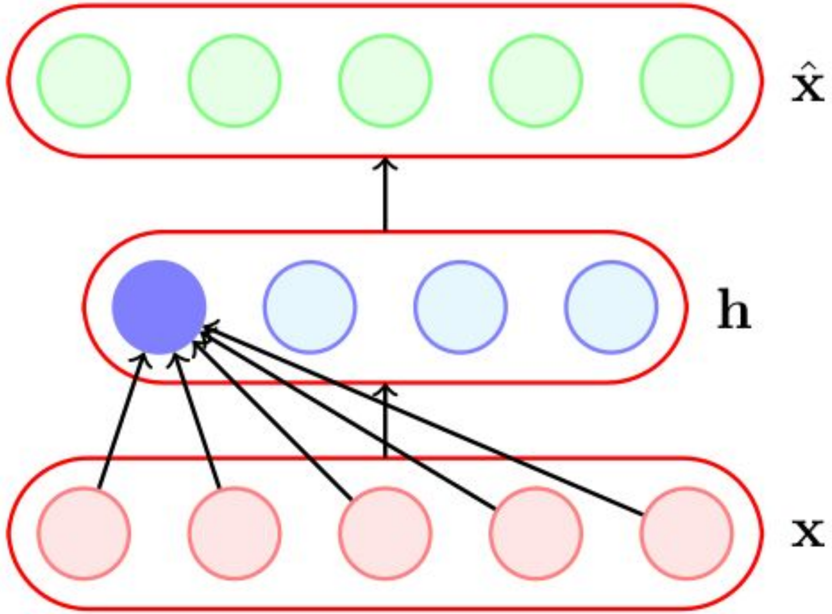


$$\|J_{\mathbf{x}}(\mathbf{h})\|_F^2 = \sum_{j=1}^{n}\sum_{l=1}^{k}\left(\frac{\partial h_l}{\partial x_j}\right)^2$$



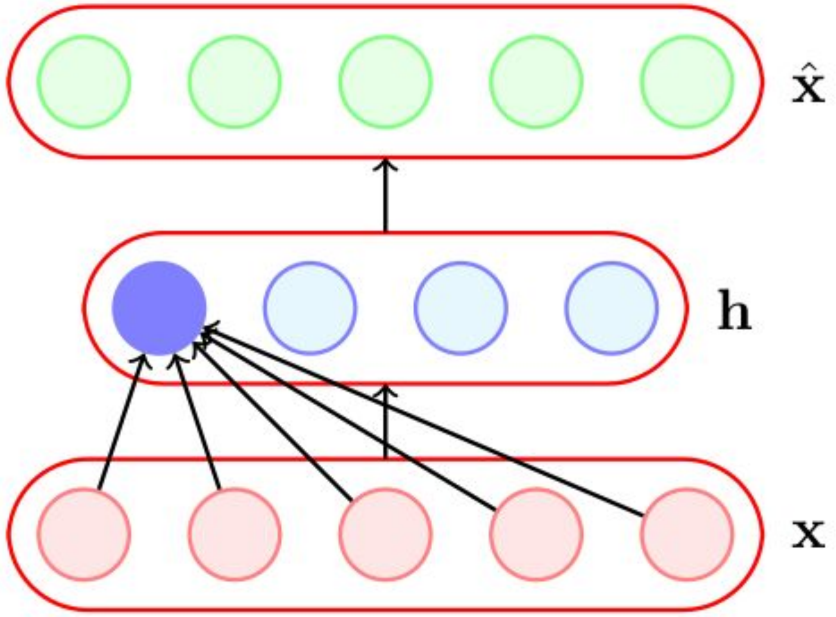
- Indeed it does and that's the idea
- By putting these two contradicting objectives against each other we ensure that **h** is sensitive to only very important variations as observed in the training data.

$$\|J_{\mathbf{x}}(\mathbf{h})\|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left(\frac{\partial h_l}{\partial x_j} \right)^2$$



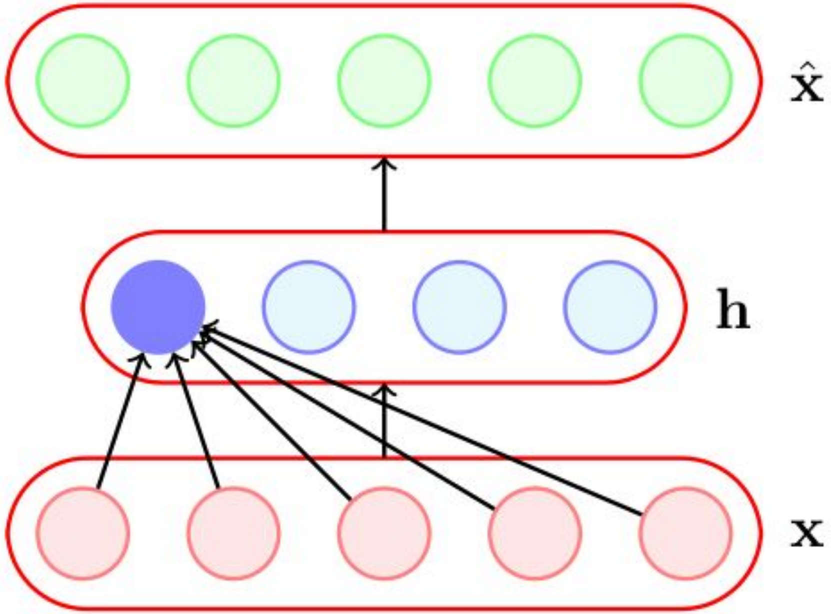
- Indeed it does and that's the idea
- By putting these two contradicting objectives against each other we ensure that **h** is sensitive to only very important variations as observed in the training data.
- $\mathcal{L}(\theta)$ - capture important variations in data

$$\|J_{\mathbf{x}}(\mathbf{h})\|_F^2 = \sum_{j=1}^{n} \sum_{l=1}^{k} \left(\frac{\partial h_l}{\partial x_j} \right)^2$$



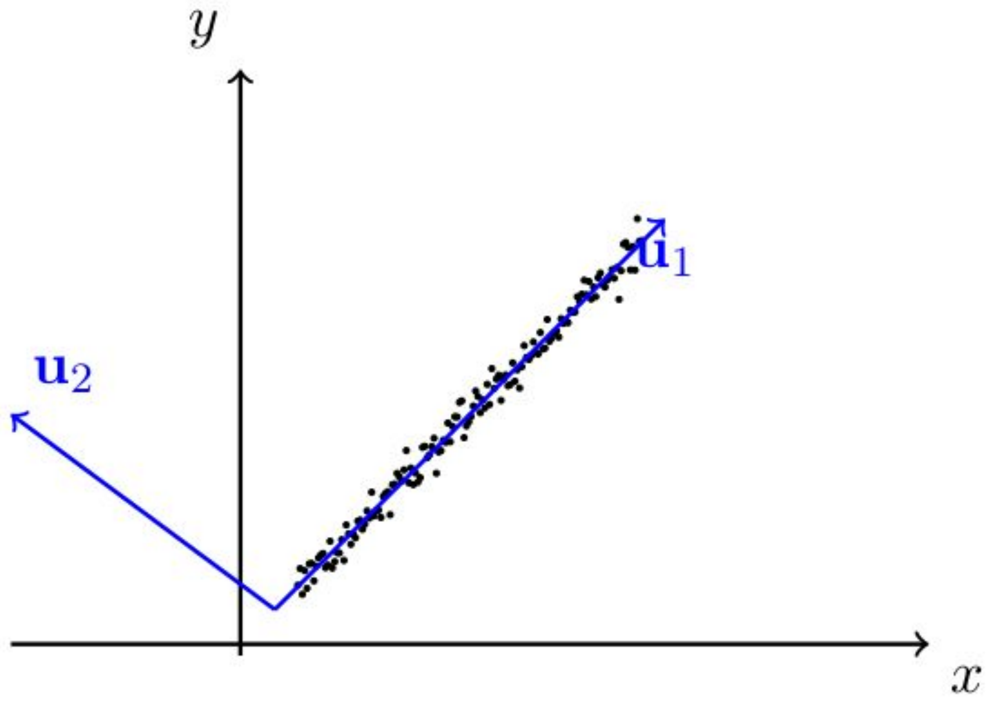
- Indeed it does and that’s the idea
- By putting these two contradicting objectives against each other we ensure that **h** is sensitive to only very important variations as observed in the training data.
- $\mathcal{L}(\theta)$ - capture important variations in data
- $\Omega(\theta)$ - do not capture variations in data

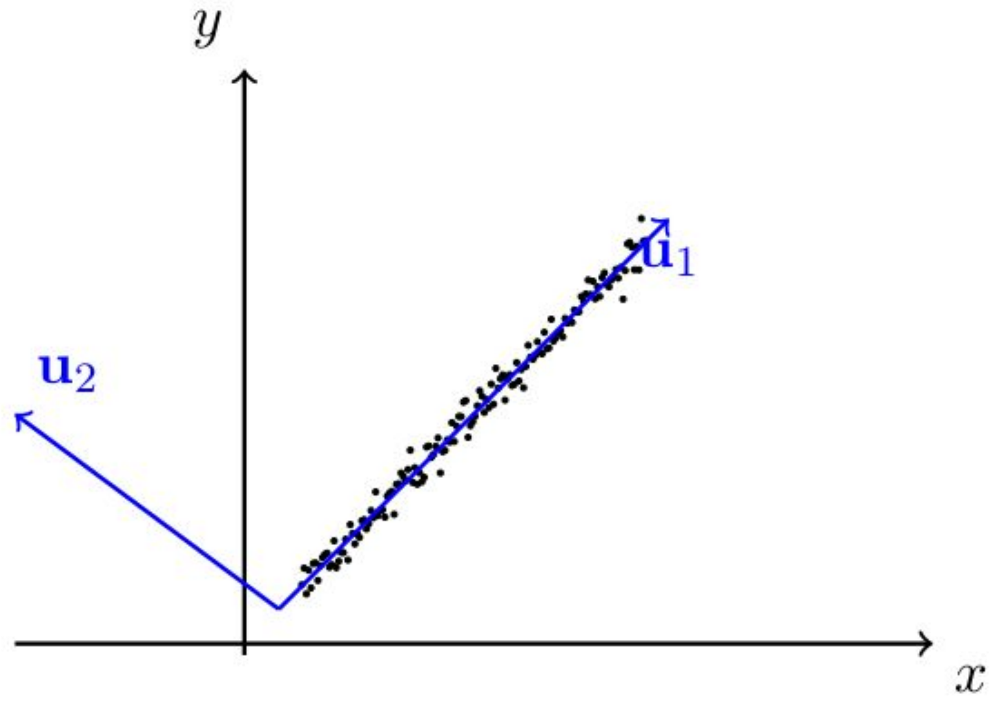
$$\|J_{\mathbf{x}}(\mathbf{h})\|_F^2 = \sum_{j=1}^{n}\sum_{l=1}^{k}\left(\frac{\partial h_l}{\partial x_j}\right)^2$$



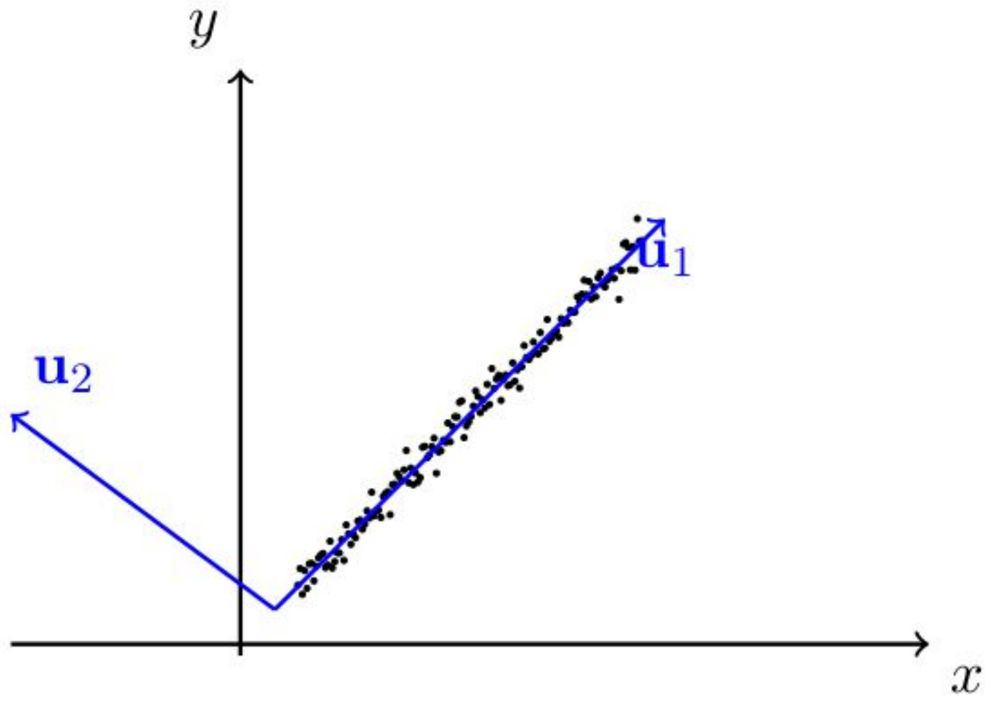
- Indeed it does and that's the idea
- By putting these two contradicting objectives against each other we ensure that **h** is sensitive to only very important variations as observed in the training data.
- $\mathcal{L}(\theta)$ - capture important variations in data
- $\Omega(\theta)$ - do not capture variations in data
- Tradeoff - capture only very important variations in the data

Let us try to understand this with the help of an illustration.

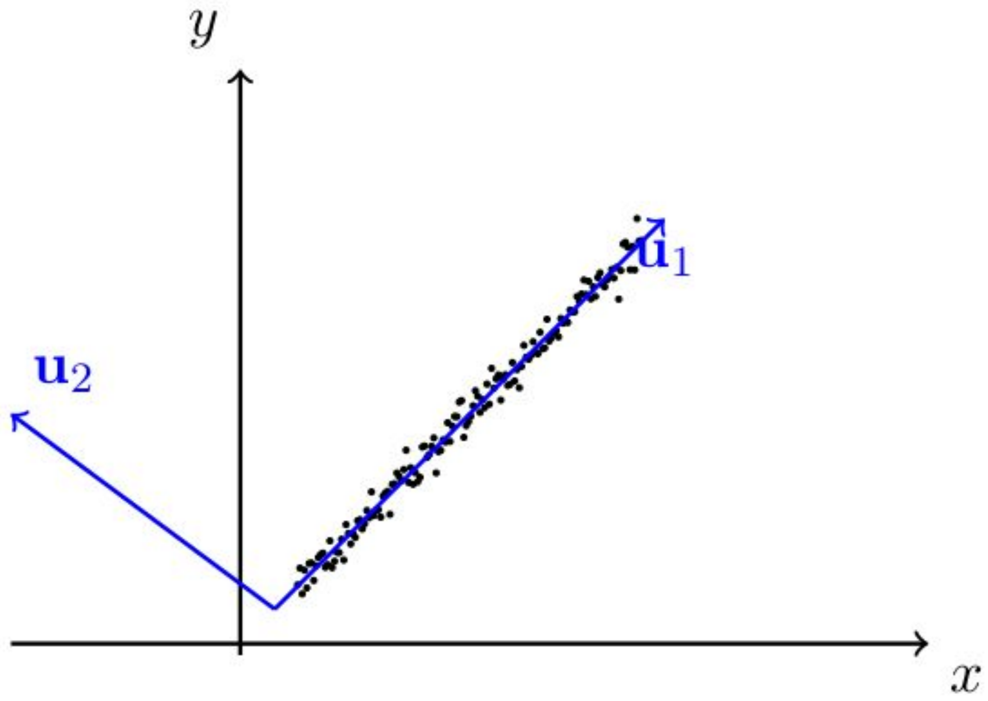

y
$\mathbf{u}_1$
$\mathbf{u}_2$
x



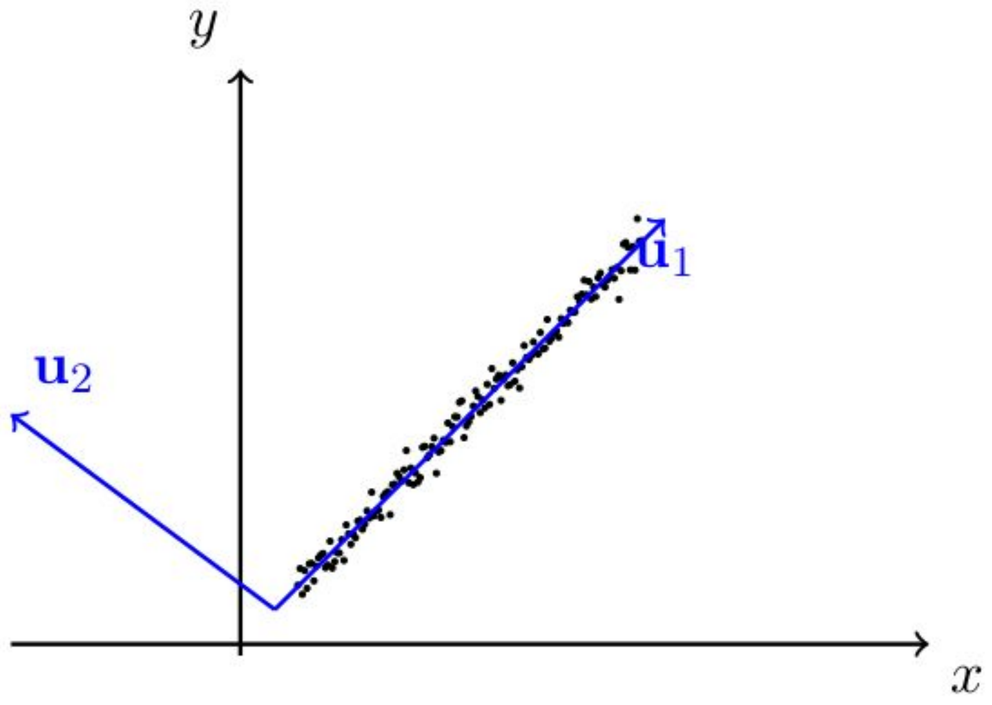
- Consider the variations in the data along directions $\mathbf{u}_1$ and $\mathbf{u}_2$



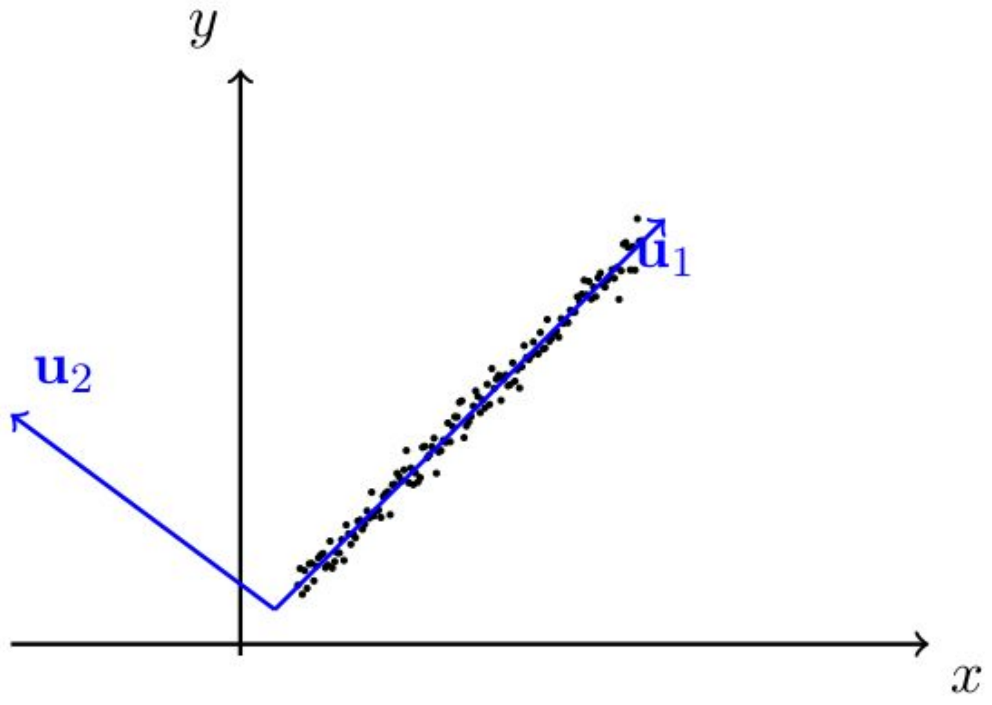
- Consider the variations in the data along directions $\mathbf{u}_1$ and $\mathbf{u}_2$
- It makes sense to maximize a neuron to be sensitive to variations along $\mathbf{u}_1$



- Consider the variations in the data along directions $\mathbf{u}_1$ and $\mathbf{u}_2$
- It makes sense to maximize a neuron to be sensitive to variations along $\mathbf{u}_1$
- At the same time it makes sense to inhibit a neuron from being sensitive to variations along $\mathbf{u}_2$ (as there seems to be small noise and unimportant for reconstruction)

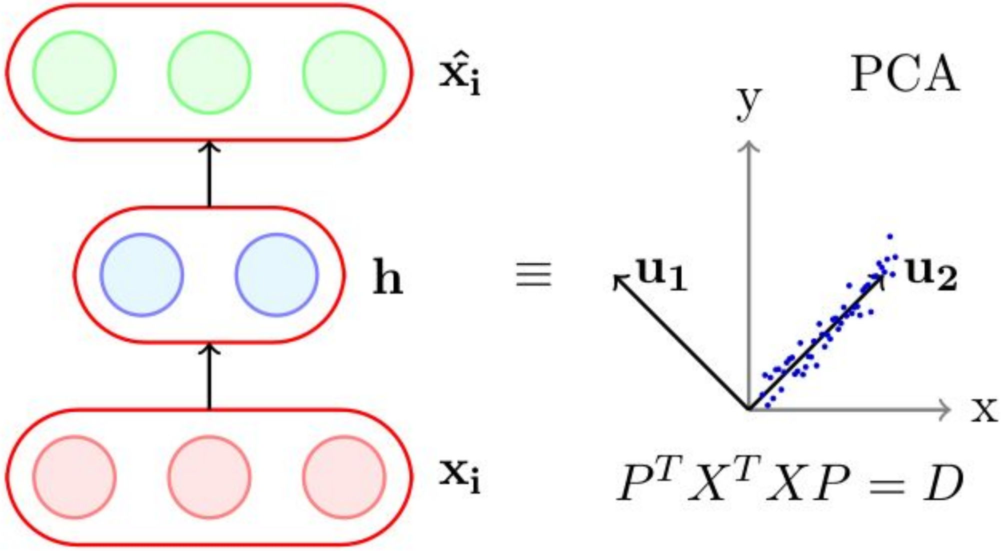


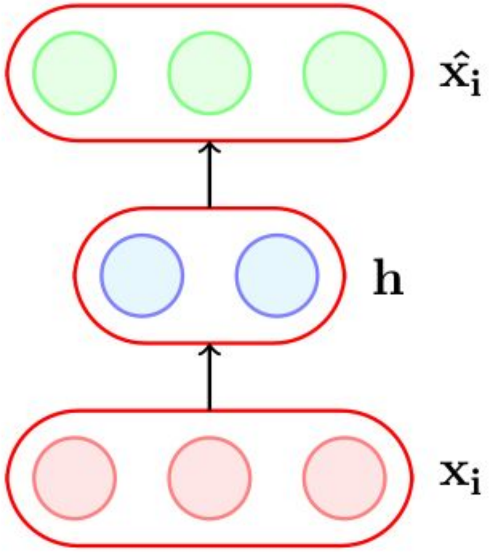
- Consider the variations in the data along directions $\mathbf{u}_1$ and $\mathbf{u}_2$
- It makes sense to maximize a neuron to be sensitive to variations along $\mathbf{u}_1$
- At the same time it makes sense to inhibit a neuron from being sensitive to variations along $\mathbf{u}_2$ (as there seems to be small noise and unimportant for reconstruction)
- By doing so we can balance between the contradicting goals of good reconstruction and low sensitivity.



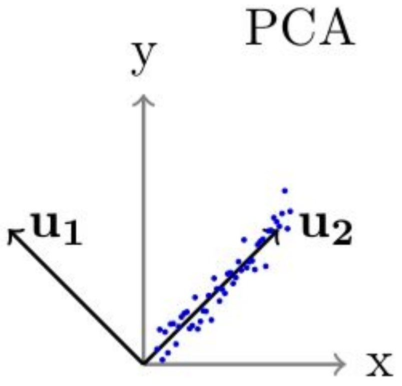
- Consider the variations in the data along directions $\mathbf{u}_1$ and $\mathbf{u}_2$
- It makes sense to maximize a neuron to be sensitive to variations along $\mathbf{u}_1$
- At the same time it makes sense to inhibit a neuron from being sensitive to variations along $\mathbf{u}_2$ (as there seems to be small noise and unimportant for reconstruction)
- By doing so we can balance between the contradicting goals of good reconstruction and low sensitivity.
- What does this remind you of?

Summary


$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\equiv$
$\mathbf{x}_\mathbf{i}$
PCA
y
x
$\mathbf{u_1}$
$\mathbf{u_2}$
$P^T X^T X P = D$

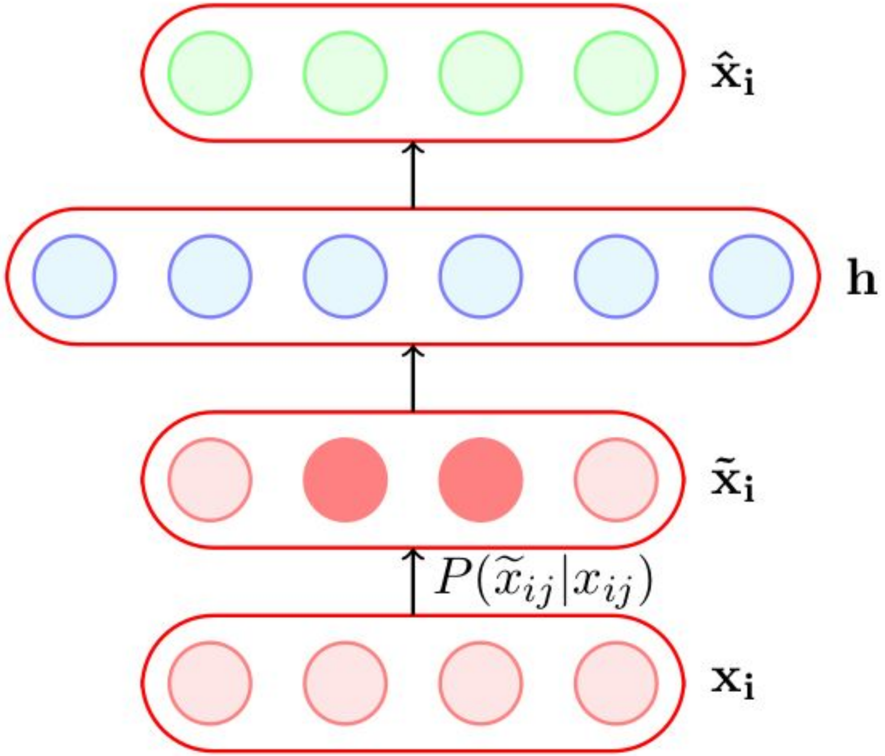


$\equiv$

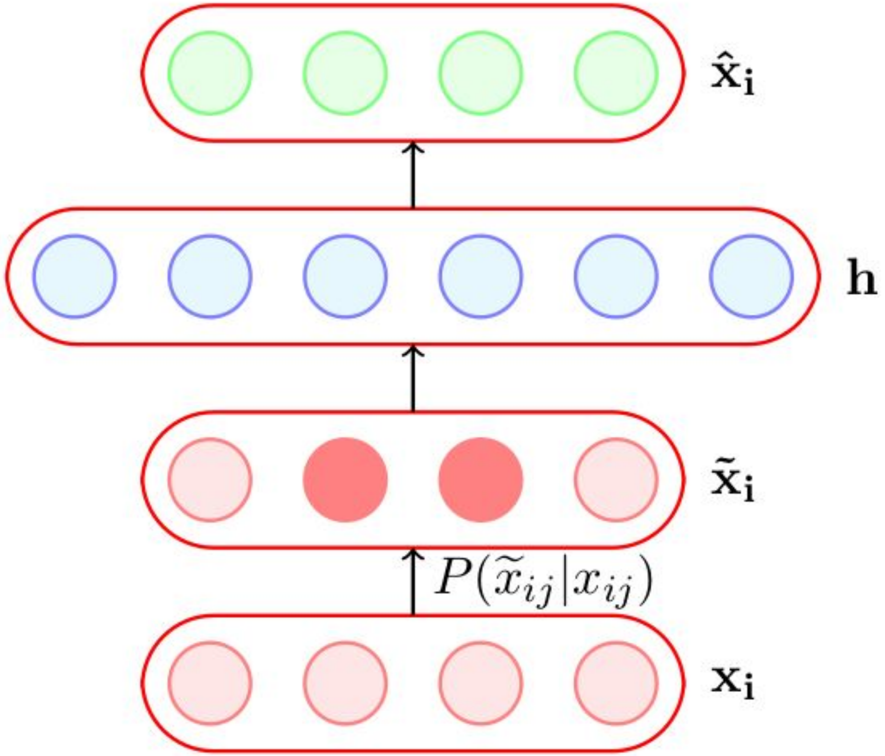


$$P^T X^T X P = D$$

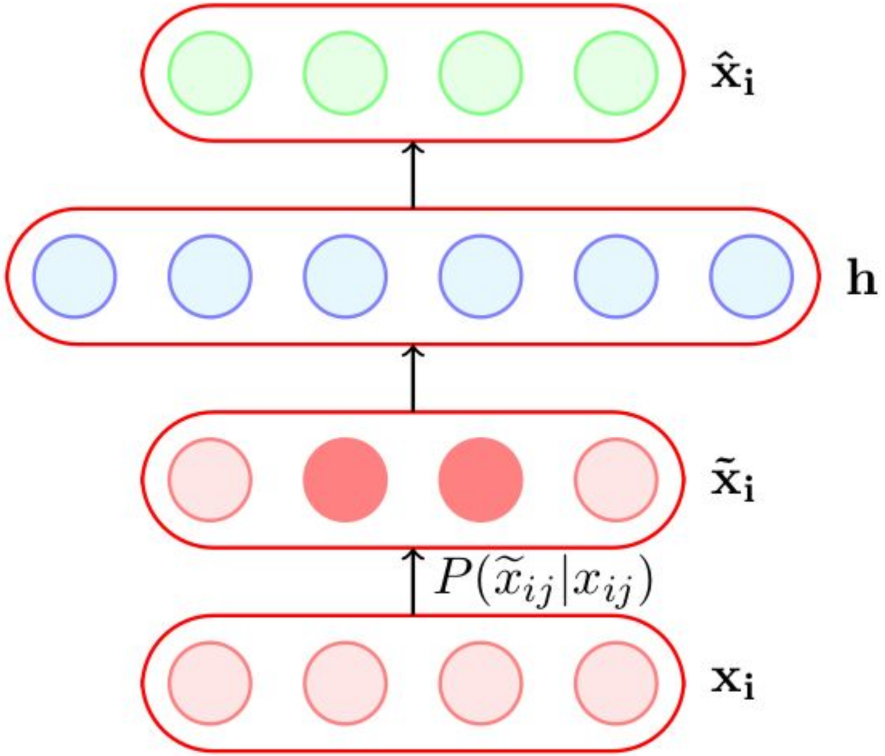
$$\min_{\theta} \| X - \underbrace{HW^*}_{\substack{U\Sigma V^T \\ \text{(SVD)}}} \|_F^2$$


$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

Regularization

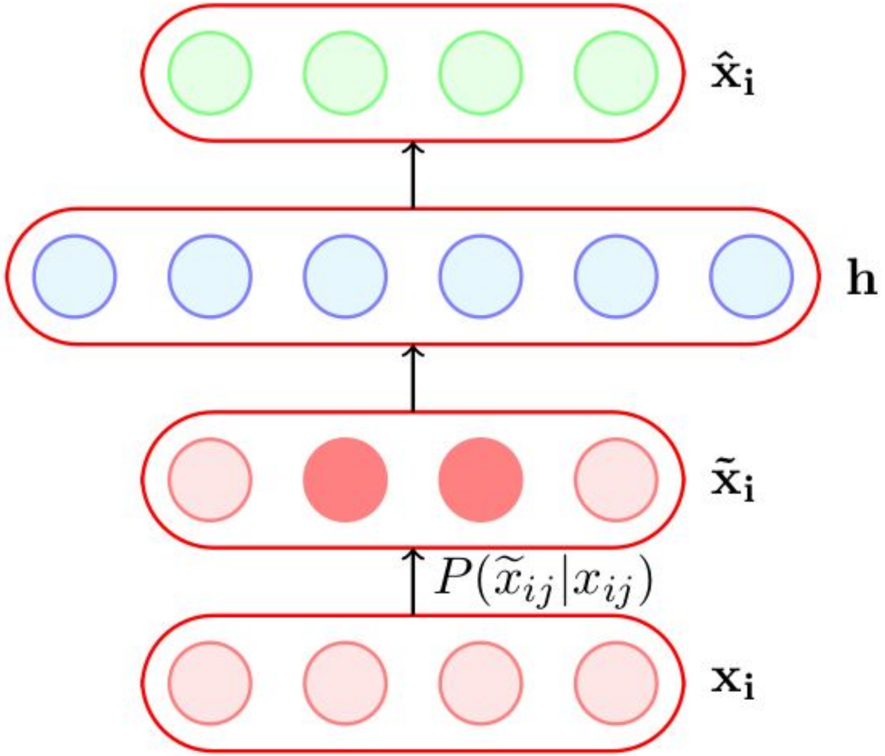

$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\tilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

Regularization


$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\tilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

$$\Omega(\theta) = \lambda \|\theta\|^2 \quad \boxed{\text{Weight decaying}}$$

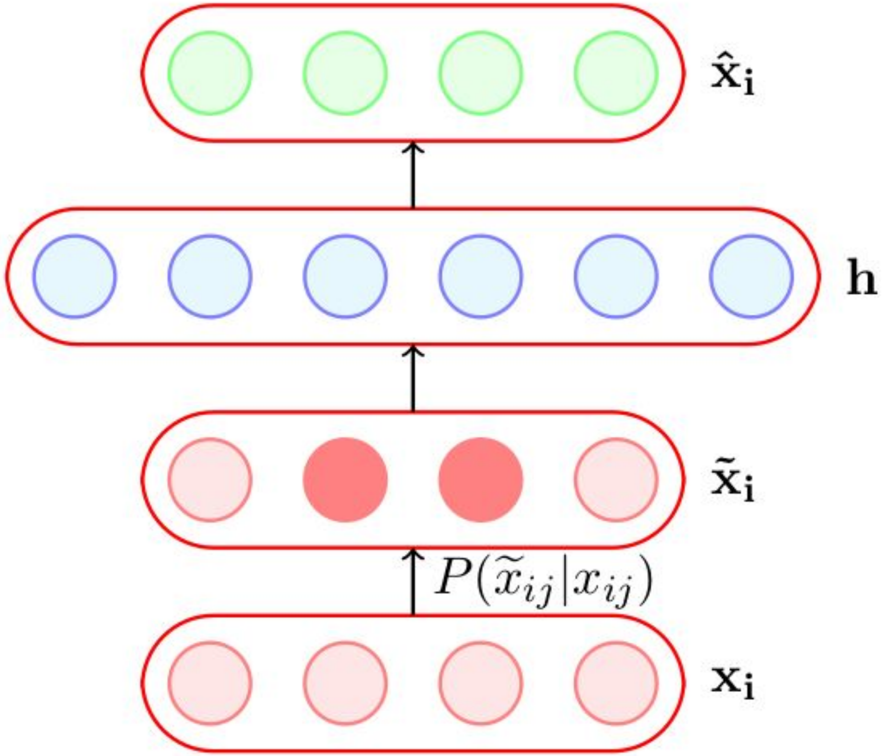
Regularization


$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

$$\Omega(\theta) = \lambda\|\theta\|^2 \quad \boxed{\text{Weight decaying}}$$

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

Regularization



$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\tilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

$$\Omega(\theta) = \lambda\|\theta\|^2 \quad \boxed{\text{Weight decaying}}$$

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

$$\Omega(\theta) = \sum_{j=1}^{n} \sum_{l=1}^{k} \left(\frac{\partial h_l}{\partial x_j}\right)^2 \quad \boxed{\text{Contractive}}$$

Acknowledgement

- Stanford University Deep Learning course
- IITM Deep Learning course