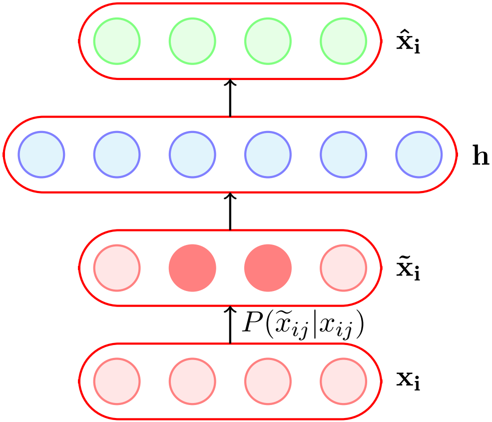
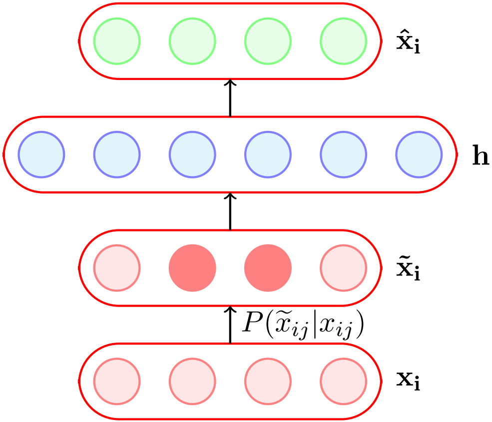
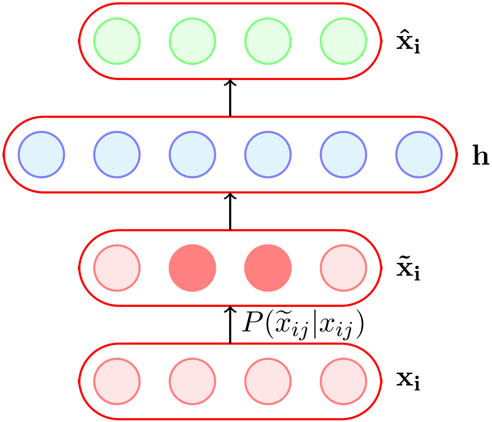
Module 7.4: Denoising Autoencoders



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

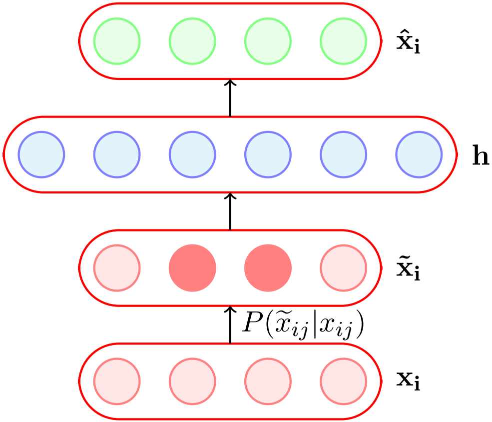


- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\widetilde{x}_{ij}|x_{ij})$) before feeding it to the network
- A simple $P(\widetilde{x}_{ij}|x_{ij})$ used in practice is the following



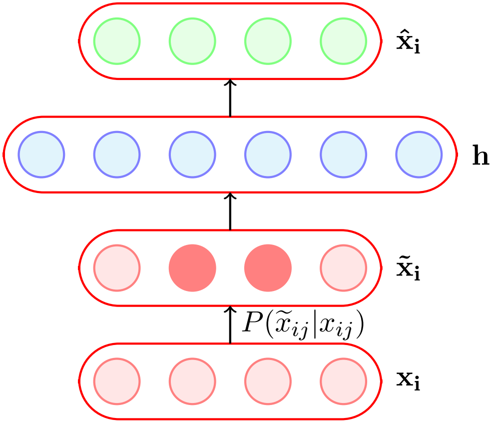
- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\widetilde{x}_{ij}|x_{ij})$) before feeding it to the network
- A simple $P(\widetilde{x}_{ij}|x_{ij})$ used in practice is the following

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



- 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

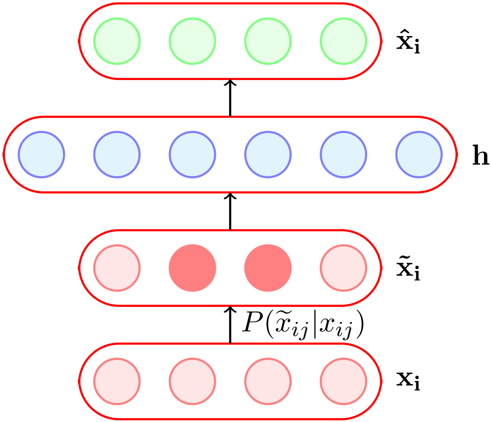
$$P(\widetilde{x}_{ij} = 0|x_{ij}) = q$$
$$P(\widetilde{x}_{ij} = x_{ij}|x_{ij}) = 1 - q$$



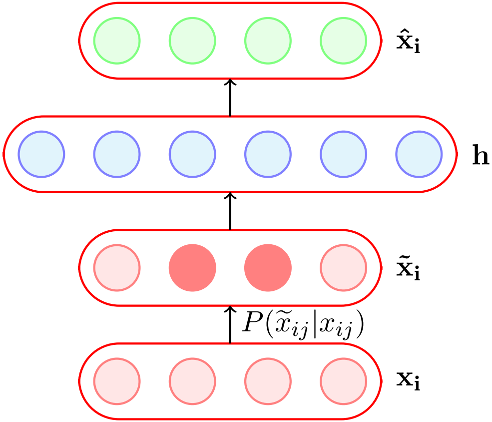
- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\widetilde{x}_{ij}|x_{ij})$) before feeding it to the network
- A simple $P(\widetilde{x}_{ij}|x_{ij})$ used in practice is the following

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

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

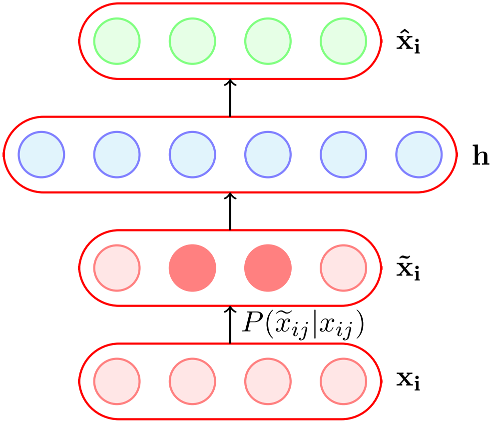


- How does this help ?



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

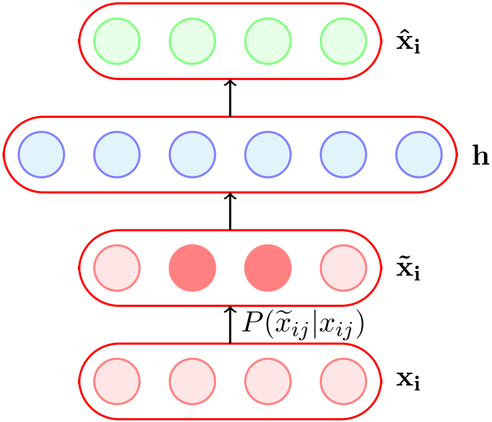
$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$



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

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

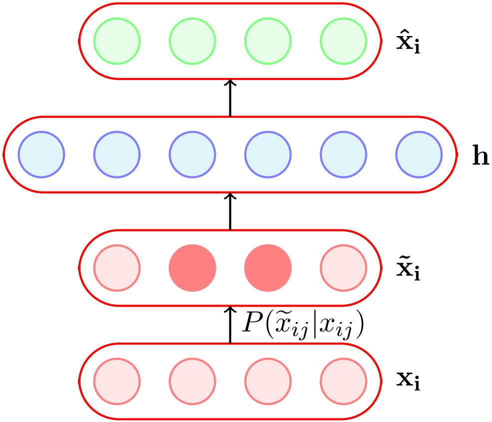
- It no longer makes sense for the model to copy the corrupted $\widetilde{\mathbf{x}}_i$ into $h(\widetilde{\mathbf{x}}_i)$ and then into $\hat{\mathbf{x}}_i$ (the objective function will not be minimized by doing so)



- 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 $\widetilde{\mathbf{x}}_i$ into $h(\widetilde{\mathbf{x}}_i)$ and then into $\hat{\mathbf{x}}_i$ (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 $\mathbf{x}_i$

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

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

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

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

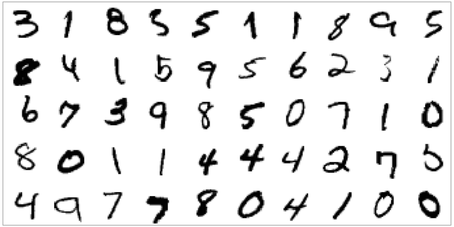
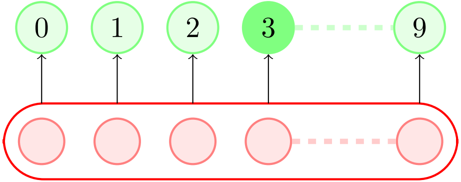


Figure: MNIST Data



$$|\mathbf{x}_i| = 784 = 28 \times 28$$

28*28

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

Task: Hand-written digit recognition

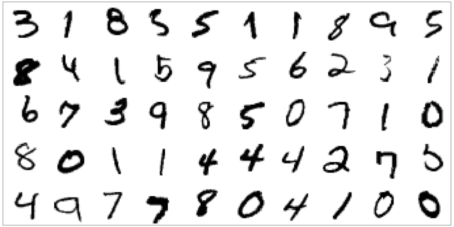


Figure: MNIST Data

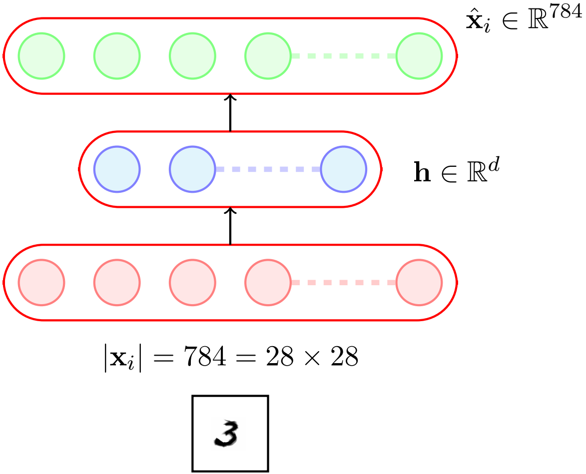


Figure: AE approach (first learn important characteristics of data)

Task: Hand-written digit recognition

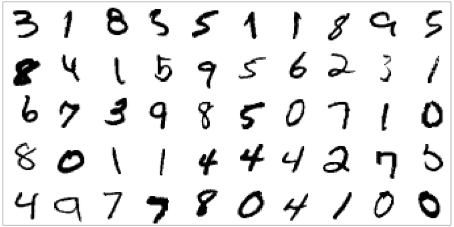


Figure: MNIST Data

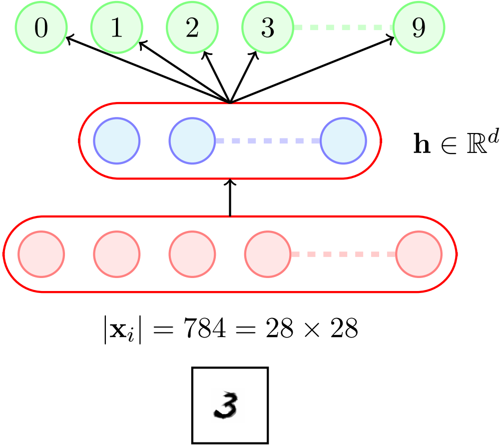
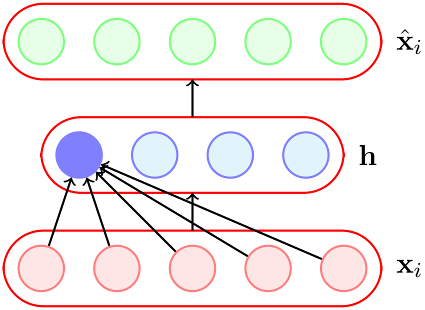
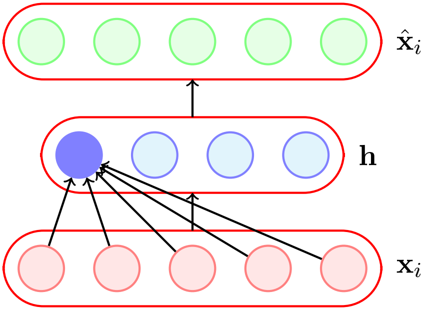


Figure: AE approach (and then train a classifier on top of this hidden representation)

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



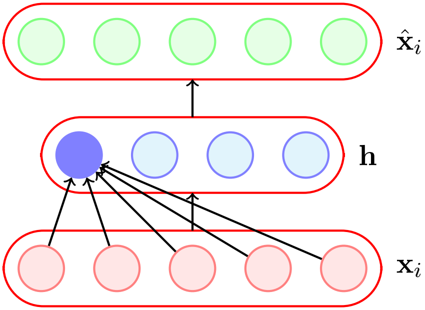
- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration $\mathbf{x}_i$



- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration $\mathbf{x}_i$
- For example,

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

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

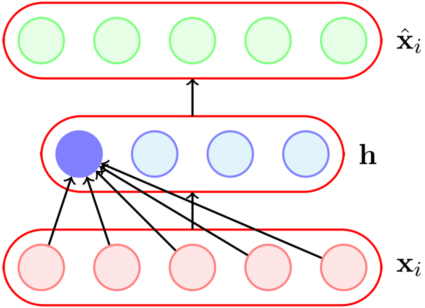


- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration $\mathbf{x}_i$
- For example,

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

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

- What values of $\mathbf{x}_i$ will cause $\mathbf{h}_1$ to be maximum (or maximally activated)

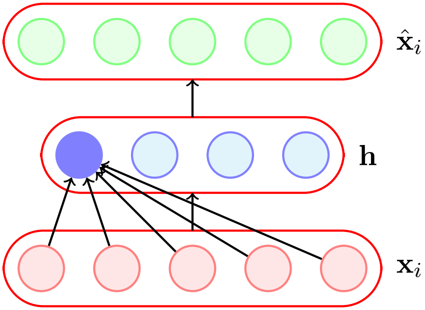


- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration $\mathbf{x}_i$
- For example,

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

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

- What values of $\mathbf{x}_i$ will cause $\mathbf{h}_1$ to be maximum (or maximally activated)
- Suppose we assume that our inputs are normalized so that $\|\mathbf{x}_i\| = 1$

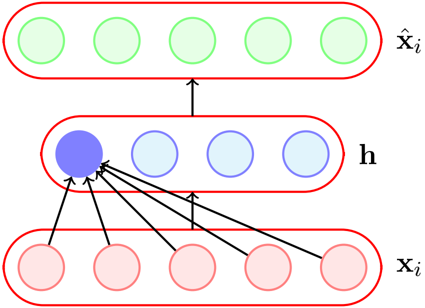


$$\max_{\mathbf{x}_i} \quad \{W_1^T \mathbf{x}_i\}$$
$$s.t. \quad ||\mathbf{x}_i||^2 = \mathbf{x}_i^T \mathbf{x}_i = 1$$

- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration $\mathbf{x}_i$
- For example,

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

  Where W_1 is the trained vector of weights connecting the input to the first hidden neuron
- What values of $\mathbf{x}_i$ will cause $\mathbf{h}_1$ to be maximum (or maximally activated)
- Suppose we assume that our inputs are normalized so that $\|\mathbf{x}_i\| = 1$



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

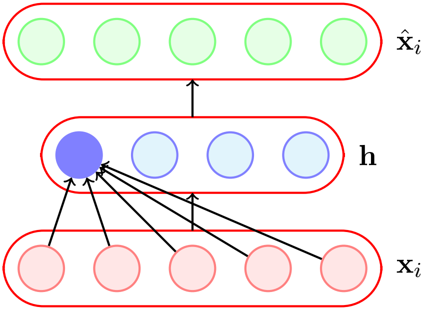
$$s.t. \quad ||\mathbf{x}_i||^2 = \mathbf{x}_i^T \mathbf{x}_i = 1$$

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

- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration $\mathbf{x}_i$
- For example,

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

  Where W_1 is the trained vector of weights connecting the input to the first hidden neuron
- What values of $\mathbf{x}_i$ will cause $\mathbf{h}_1$ to be maximum (or maximally activated)
- Suppose we assume that our inputs are normalized so that $\|\mathbf{x}_i\| = 1$



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

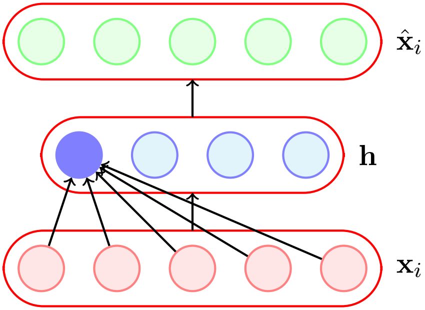
$$s.t. \quad ||\mathbf{x}_i||^2 = \mathbf{x}_i^T \mathbf{x}_i = 1$$

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

- Thus the inputs

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

will respectively cause hidden neurons 1 to n to maximally fire



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

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

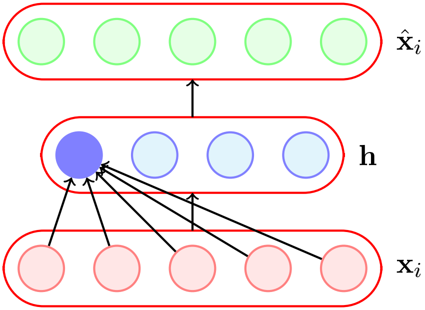
$$\text{Solution:} \quad \mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}}$$

- 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 ($\mathbf{x}_i$'s) which maximally activate the first k neurons of the hidden representations learned by a vanilla autoencoder and different denoising autoencoders



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

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

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

- 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 ($\mathbf{x}_i$'s) which maximally activate the first k neurons of the hidden representations learned by a vanilla autoencoder and different denoising autoencoders
- These $\mathbf{x}_i$'s are computed by the above formula using the weights $(W_1, W_2 \ldots W_k)$ learned by the respective autoencoders

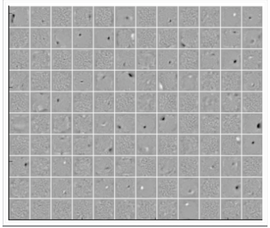


Figure: Vanilla AE (No noise)

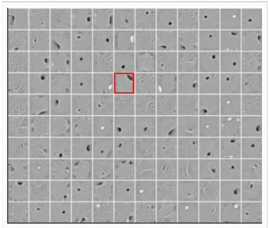


Figure: 25% Denoising AE (q=0.25)

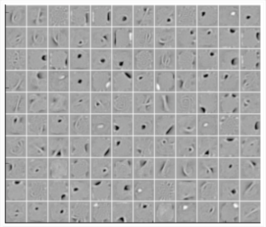


Figure: 50% Denoising AE (q=0.5)

- The vanilla AE does not learn many meaningful patterns

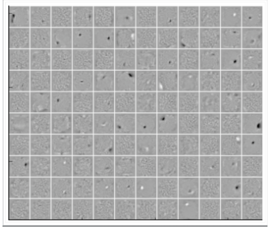


Figure: Vanilla AE (No noise)

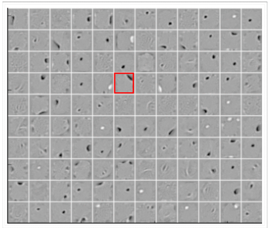


Figure: 25% Denoising AE (q=0.25)

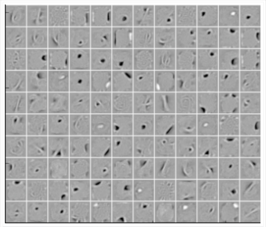


Figure: 50% Denoising AE (q=0.5)

- The 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')

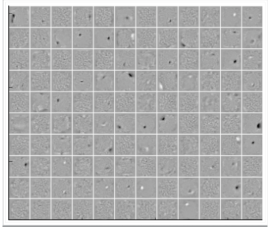


Figure: Vanilla AE (No noise)

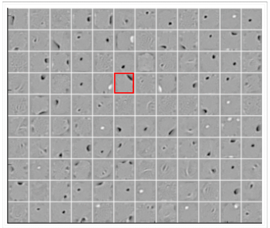


Figure: 25% Denoising AE (q=0.25)

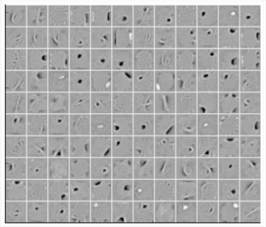
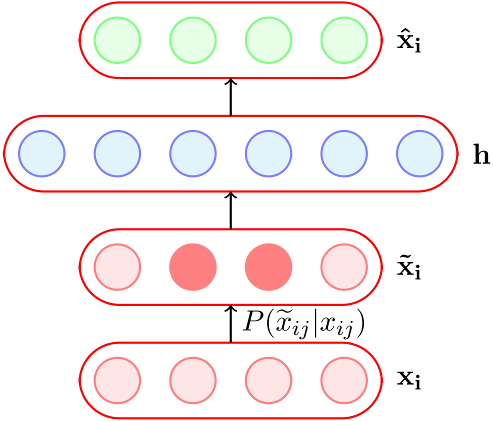
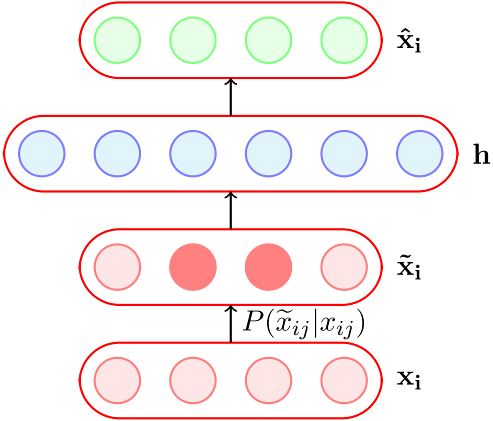


Figure: 50% Denoising AE (q=0.5)

- The 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

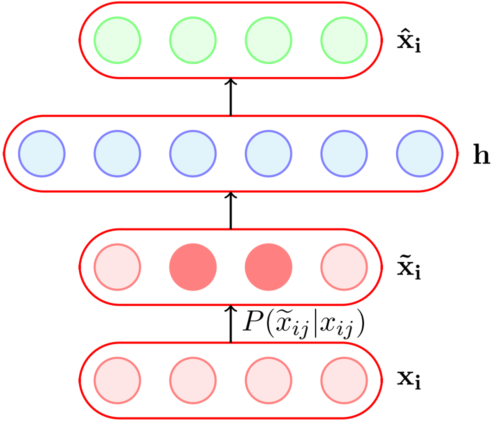


- We saw one form of $P(\widetilde{x}_{ij}|x_{ij})$ which flips a fraction q of the inputs to zero



- 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 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

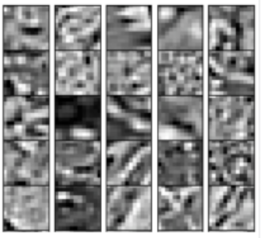


Figure: Data

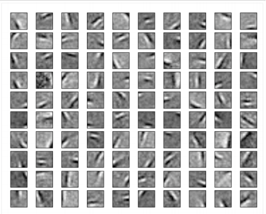


Figure: AE filters

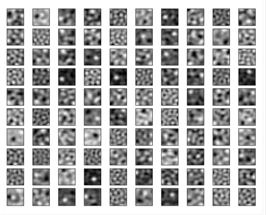


Figure: Weight decay filters

- The hidden neurons essentially behave like edge detectors

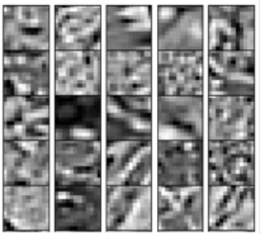


Figure: Data

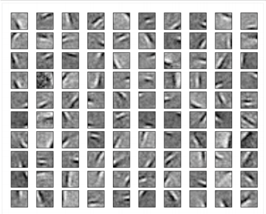


Figure: AE filters

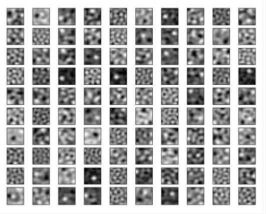


Figure: Weight decay filters

- The hidden neurons essentially behave like edge detectors
- PCA does not give such edge detectors