

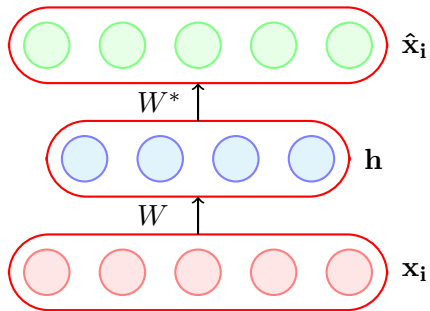
CS7015 (Deep Learning) : Lecture 7

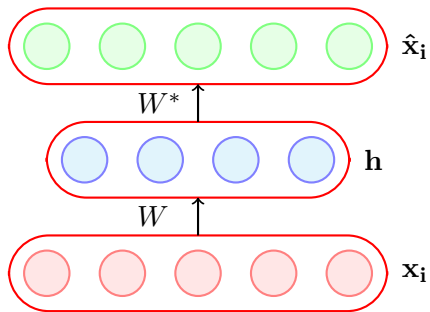
Autoencoders and relation to PCA, Regularization in autoencoders, Denoising autoencoders, Sparse autoencoders, Contractive autoencoders

Mitesh M. Khapra

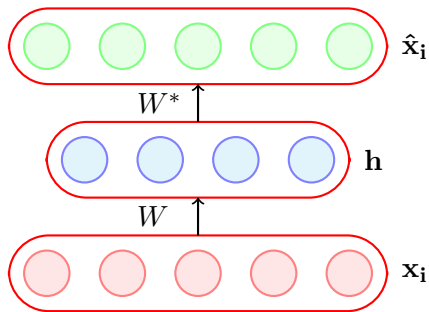
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Module 7.1: Introduction to Autoencoders

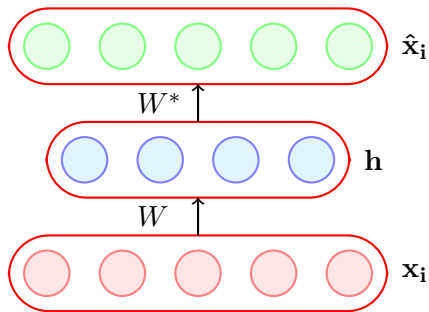




- An autoencoder is a special type of feed forward neural network which does the following

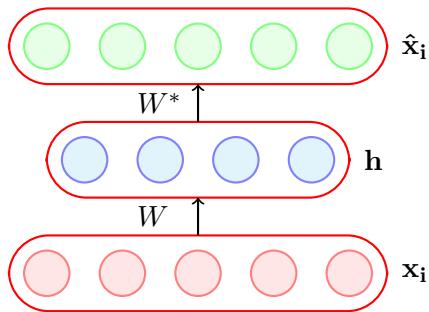


- An autoencoder is a special type of feed forward neural network which does the following
- Encodes its input \mathbf{x}_i into a hidden representation \mathbf{h}



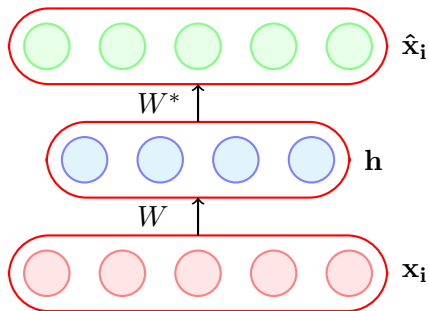
$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

- An autoencoder is a special type of feed forward neural network which does the following
- Encodes its input \mathbf{x}_i into a hidden representation \mathbf{h}



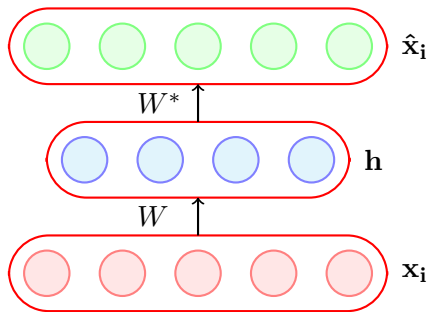
$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

- An autoencoder is a special type of feed forward neural network which does the following
- Encodes its input \mathbf{x}_i into a hidden representation \mathbf{h}
- Decodes the input again from this hidden representation



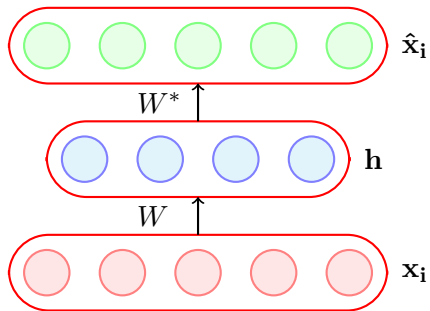
$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$
$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

- An autoencoder is a special type of feed forward neural network which does the following
- Encodes its input \mathbf{x}_i into a hidden representation \mathbf{h}
- Decodes the input again from this hidden representation

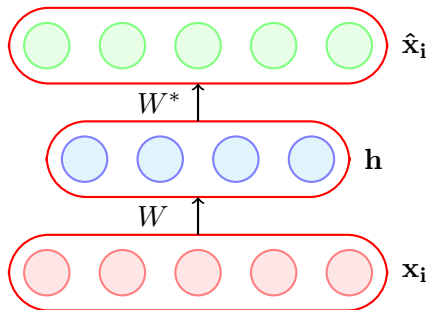


$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$
$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

- An autoencoder is a special type of feed forward neural network which does the following
- Encodes its input \mathbf{x}_i into a hidden representation \mathbf{h}
- Decodes the input again from this hidden representation
- The model is trained to minimize a certain loss function which will ensure that $\hat{\mathbf{x}}_i$ is close to \mathbf{x}_i (we will see some such loss functions soon)

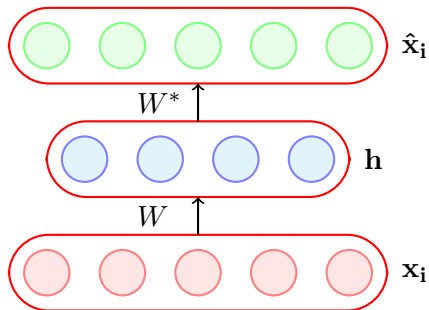


$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$
$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$



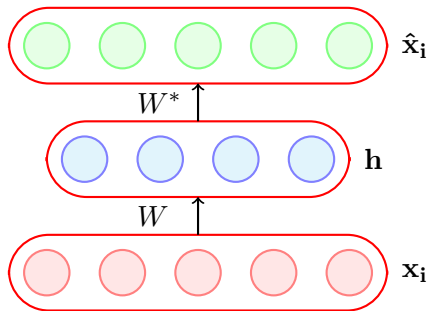
$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$
$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

- Let us consider the case where $\dim(\mathbf{h}) < \dim(\mathbf{x}_i)$



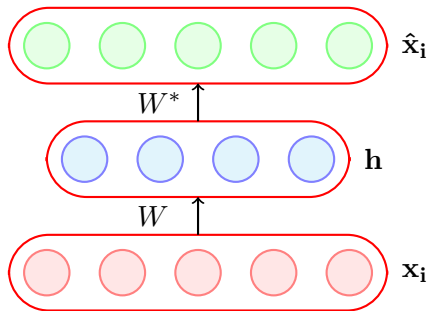
$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$
$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

- Let us consider the case where $\dim(\mathbf{h}) < \dim(\mathbf{x}_i)$
- If we are still able to reconstruct $\hat{\mathbf{x}}_i$ perfectly from \mathbf{h} , then what does it say about \mathbf{h} ?



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$
$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

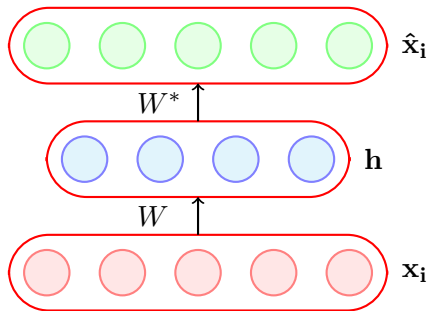
- Let us consider the case where $\dim(\mathbf{h}) < \dim(\mathbf{x}_i)$
- If we are still able to reconstruct $\hat{\mathbf{x}}_i$ perfectly from \mathbf{h} , then what does it say about \mathbf{h} ?
- \mathbf{h} is a loss-free encoding of \mathbf{x}_i . It captures all the important characteristics of \mathbf{x}_i



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

- Let us consider the case where $\dim(\mathbf{h}) < \dim(\mathbf{x}_i)$
- If we are still able to reconstruct $\hat{\mathbf{x}}_i$ perfectly from \mathbf{h} , then what does it say about \mathbf{h} ?
- \mathbf{h} is a loss-free encoding of \mathbf{x}_i . It captures all the important characteristics of \mathbf{x}_i
- Do you see an analogy with PCA?

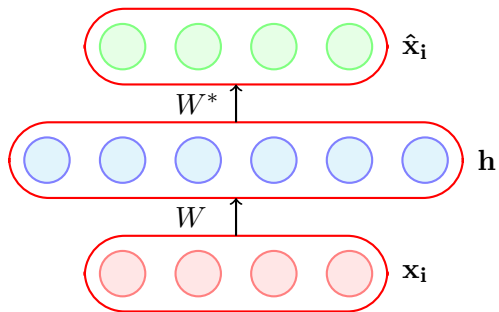


$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

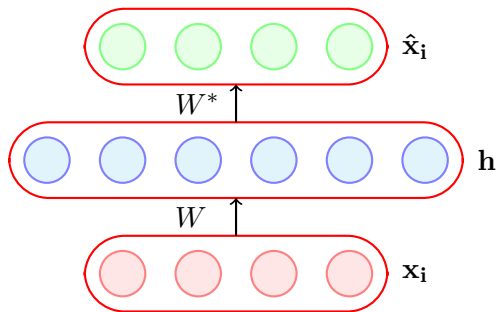
An autoencoder where $\dim(\mathbf{h}) < \dim(\mathbf{x}_i)$ is called an under complete autoencoder

- Let us consider the case where $\dim(\mathbf{h}) < \dim(\mathbf{x}_i)$
- If we are still able to reconstruct $\hat{\mathbf{x}}_i$ perfectly from \mathbf{h} , then what does it say about \mathbf{h} ?
- \mathbf{h} is a loss-free encoding of \mathbf{x}_i . It captures all the important characteristics of \mathbf{x}_i
- Do you see an analogy with PCA?



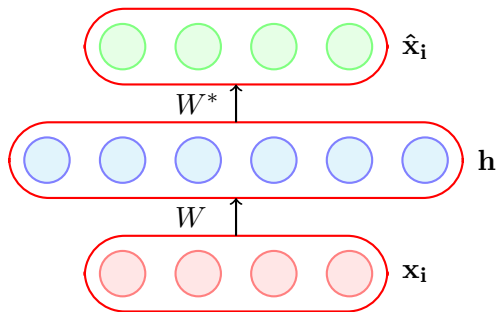
$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$
$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_i)$



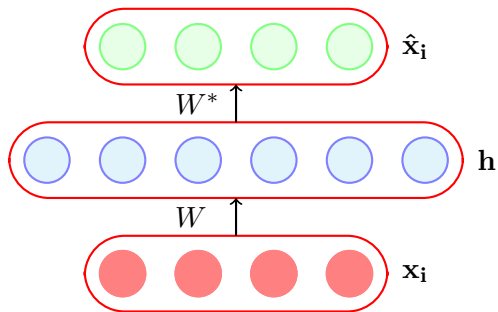
$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$
$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

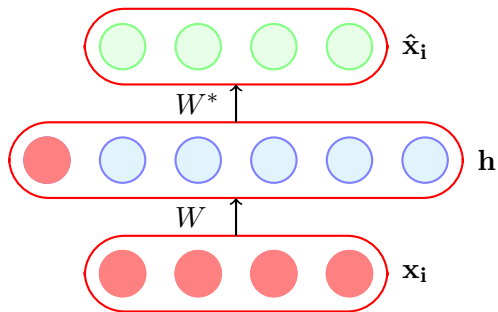
- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_i)$
- In such a case the autoencoder could learn a trivial encoding by simply copying \mathbf{x}_i into \mathbf{h} and then copying \mathbf{h} into $\hat{\mathbf{x}}_i$



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

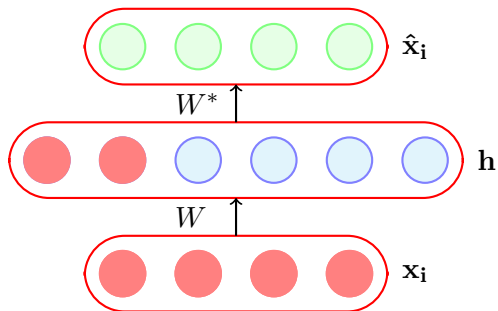
- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_i)$
- In such a case the autoencoder could learn a trivial encoding by simply copying \mathbf{x}_i into \mathbf{h} and then copying \mathbf{h} into $\hat{\mathbf{x}}_i$



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

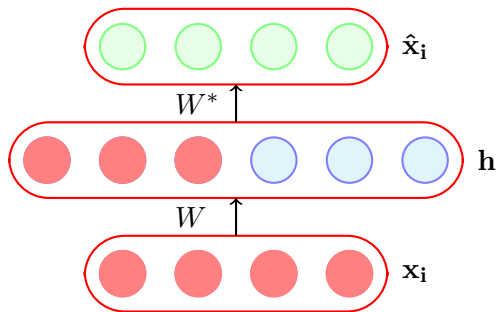
- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_i)$
- In such a case the autoencoder could learn a trivial encoding by simply copying \mathbf{x}_i into \mathbf{h} and then copying \mathbf{h} into $\hat{\mathbf{x}}_i$



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

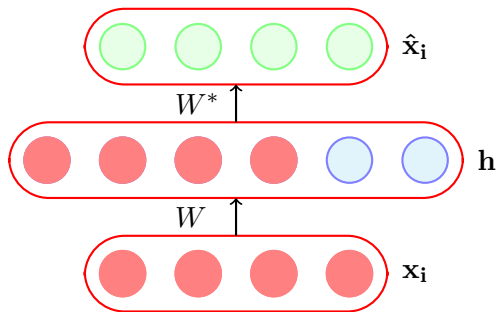
- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_i)$
- In such a case the autoencoder could learn a trivial encoding by simply copying \mathbf{x}_i into \mathbf{h} and then copying \mathbf{h} into $\hat{\mathbf{x}}_i$



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

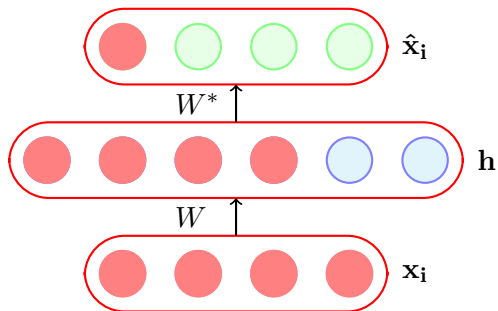
- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_i)$
- In such a case the autoencoder could learn a trivial encoding by simply copying \mathbf{x}_i into \mathbf{h} and then copying \mathbf{h} into $\hat{\mathbf{x}}_i$



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

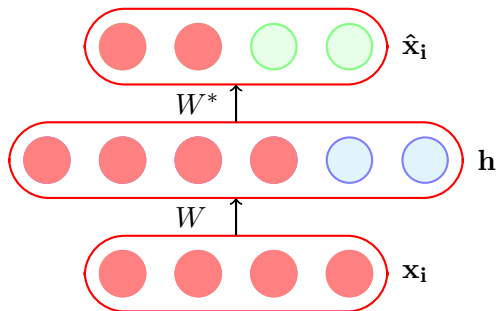
- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_i)$
- In such a case the autoencoder could learn a trivial encoding by simply copying \mathbf{x}_i into \mathbf{h} and then copying \mathbf{h} into $\hat{\mathbf{x}}_i$



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

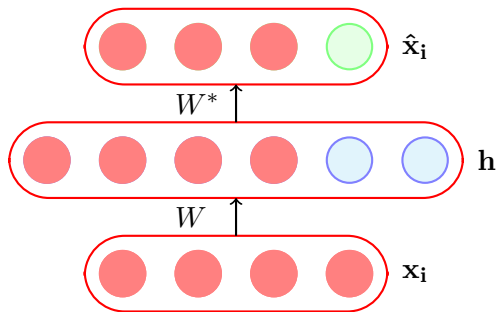
- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_i)$
- In such a case the autoencoder could learn a trivial encoding by simply copying \mathbf{x}_i into \mathbf{h} and then copying \mathbf{h} into $\hat{\mathbf{x}}_i$



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

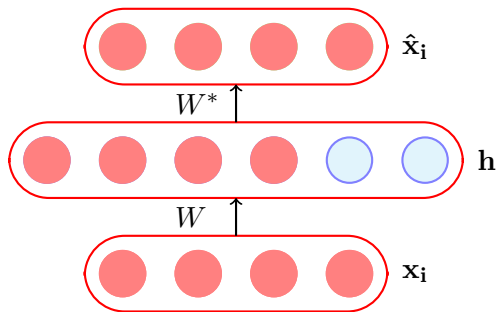
- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_i)$
- In such a case the autoencoder could learn a trivial encoding by simply copying \mathbf{x}_i into \mathbf{h} and then copying \mathbf{h} into $\hat{\mathbf{x}}_i$



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

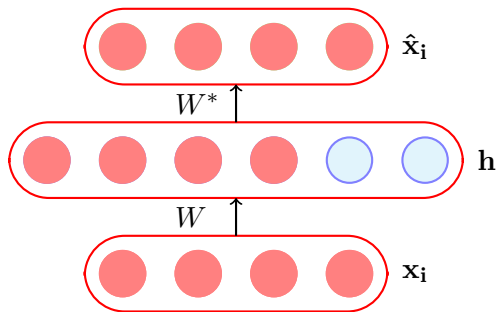
- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_i)$
- In such a case the autoencoder could learn a trivial encoding by simply copying \mathbf{x}_i into \mathbf{h} and then copying \mathbf{h} into $\hat{\mathbf{x}}_i$



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

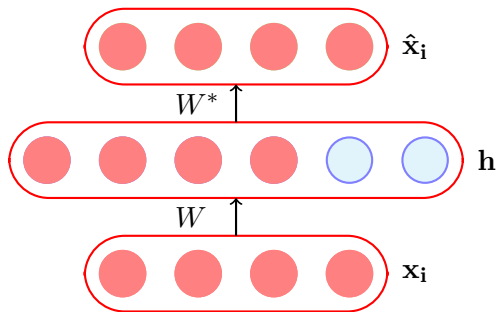
- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_i)$
- In such a case the autoencoder could learn a trivial encoding by simply copying \mathbf{x}_i into \mathbf{h} and then copying \mathbf{h} into $\hat{\mathbf{x}}_i$



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_i)$
- In such a case the autoencoder could learn a trivial encoding by simply copying \mathbf{x}_i into \mathbf{h} and then copying \mathbf{h} into $\hat{\mathbf{x}}_i$
- Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

An autoencoder where $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_i)$ is called an over complete autoencoder

- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_i)$
- In such a case the autoencoder could learn a trivial encoding by simply copying \mathbf{x}_i into \mathbf{h} and then copying \mathbf{h} into $\hat{\mathbf{x}}_i$
- Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data

The Road Ahead

The Road Ahead

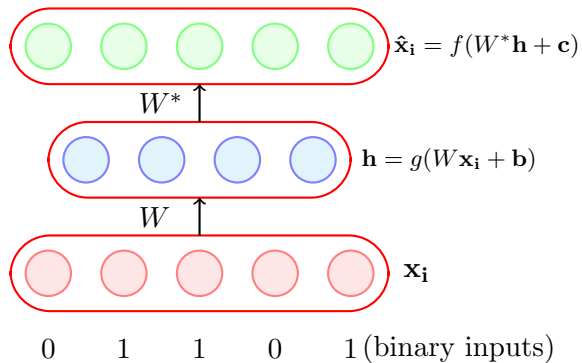
- Choice of $f(\mathbf{x}_i)$ and $g(\mathbf{x}_i)$

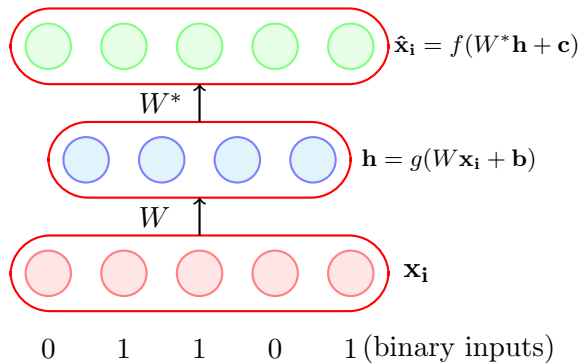
The Road Ahead

- Choice of $f(\mathbf{x}_i)$ and $g(\mathbf{x}_i)$
- Choice of loss function

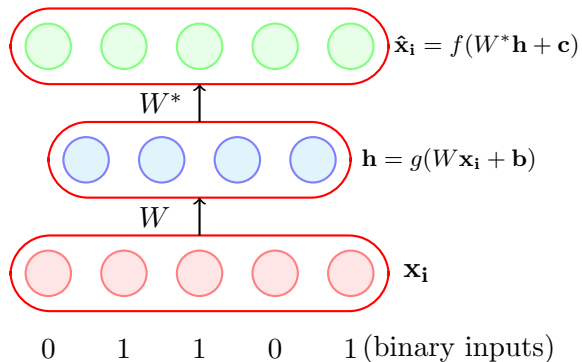
The Road Ahead

- Choice of $f(\mathbf{x}_i)$ and $g(\mathbf{x}_i)$
- Choice of loss function

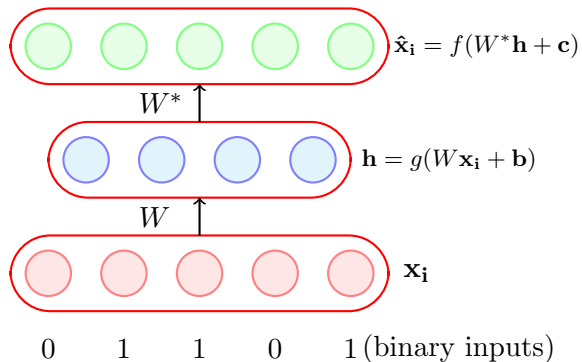




- Suppose all our inputs are binary (each $x_{ij} \in \{0, 1\}$)

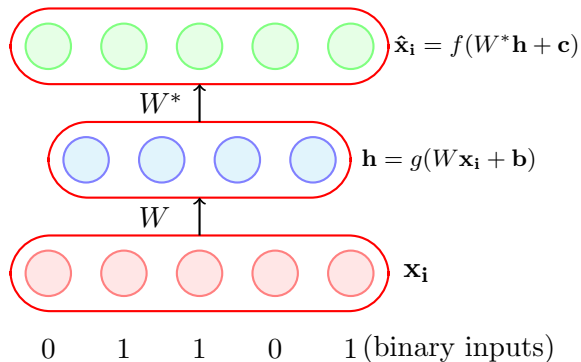


- Suppose all our inputs are binary (each $x_{ij} \in \{0, 1\}$)
- Which of the following functions would be most apt for the decoder?



- Suppose all our inputs are binary (each $x_{ij} \in \{0, 1\}$)
- Which of the following functions would be most apt for the decoder?

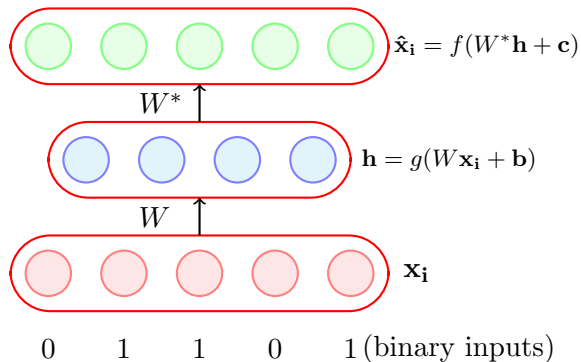
$$\hat{\mathbf{x}}_i = \tanh(W^*\mathbf{h} + \mathbf{c})$$



- Suppose all our inputs are binary (each $x_{ij} \in \{0, 1\}$)
- Which of the following functions would be most apt for the decoder?

$$\hat{\mathbf{x}}_i = \tanh(W^*\mathbf{h} + \mathbf{c})$$

$$\hat{\mathbf{x}}_i = W^*\mathbf{h} + \mathbf{c}$$

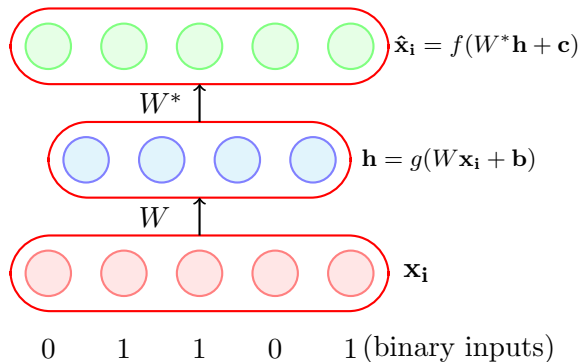


- Suppose all our inputs are binary (each $x_{ij} \in \{0, 1\}$)
- Which of the following functions would be most apt for the decoder?

$$\hat{\mathbf{x}}_i = \tanh(W^*\mathbf{h} + \mathbf{c})$$

$$\hat{\mathbf{x}}_i = W^*\mathbf{h} + \mathbf{c}$$

$$\hat{\mathbf{x}}_i = \text{logistic}(W^*\mathbf{h} + \mathbf{c})$$



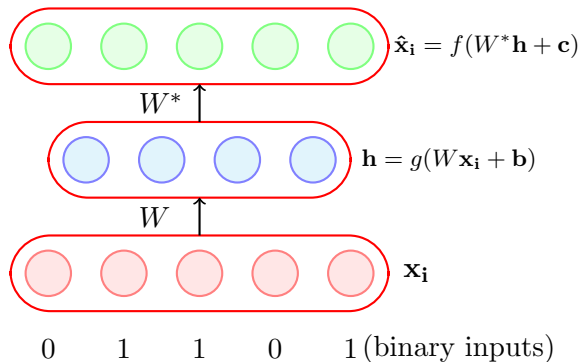
- Suppose all our inputs are binary (each $x_{ij} \in \{0, 1\}$)
- Which of the following functions would be most apt for the decoder?

$$\hat{\mathbf{x}}_i = \tanh(W^*\mathbf{h} + \mathbf{c})$$

$$\hat{\mathbf{x}}_i = W^*\mathbf{h} + \mathbf{c}$$

$$\hat{\mathbf{x}}_i = \text{logistic}(W^*\mathbf{h} + \mathbf{c})$$

- Logistic as it naturally restricts all outputs to be between 0 and 1



g is typically chosen as the sigmoid function

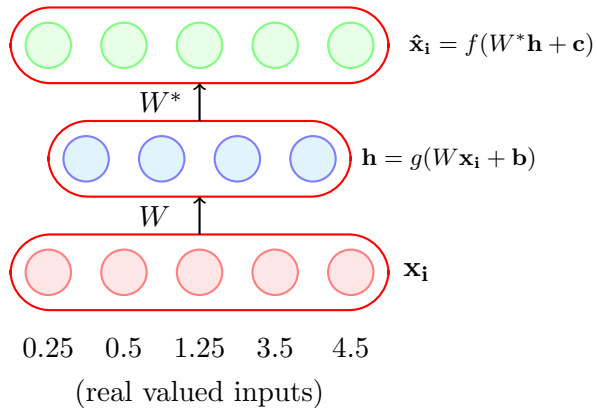
- Suppose all our inputs are binary (each $x_{ij} \in \{0, 1\}$)
- Which of the following functions would be most apt for the decoder?

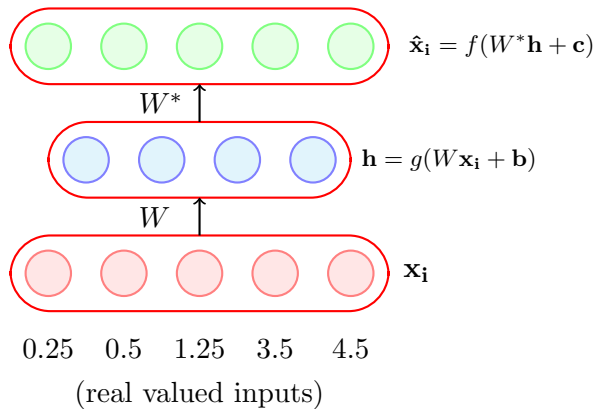
$$\hat{\mathbf{x}}_i = \tanh(W^*\mathbf{h} + \mathbf{c})$$

$$\hat{\mathbf{x}}_i = W^*\mathbf{h} + \mathbf{c}$$

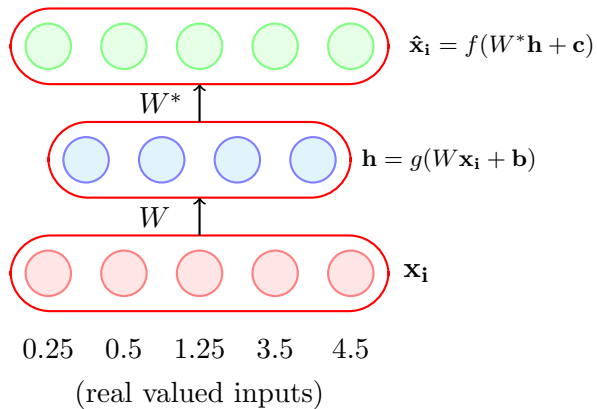
$$\hat{\mathbf{x}}_i = \text{logistic}(W^*\mathbf{h} + \mathbf{c})$$

- Logistic as it naturally restricts all outputs to be between 0 and 1

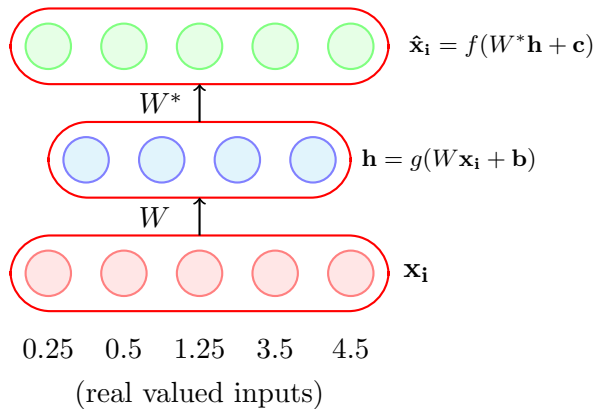




- Suppose all our inputs are real (each $x_{ij} \in \mathbb{R}$)

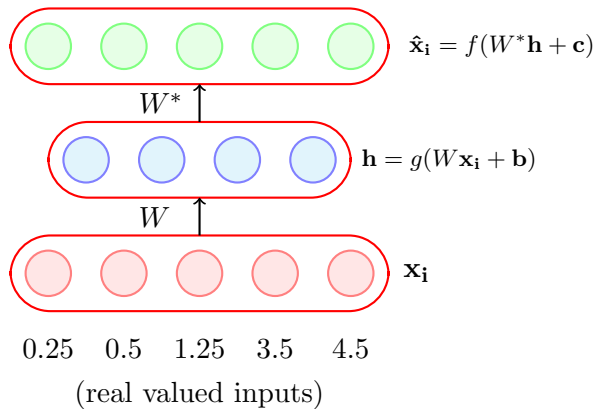


- Suppose all our inputs are real (each $x_{ij} \in \mathbb{R}$)
- Which of the following functions would be most apt for the decoder?



- Suppose all our inputs are real (each $x_{ij} \in \mathbb{R}$)
- Which of the following functions would be most apt for the decoder?

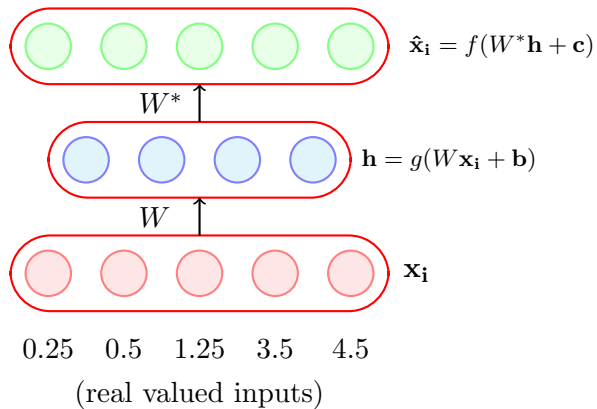
$$\hat{\mathbf{x}}_i = \tanh(W^*\mathbf{h} + \mathbf{c})$$



- Suppose all our inputs are real (each $x_{ij} \in \mathbb{R}$)
- Which of the following functions would be most apt for the decoder?

$$\hat{\mathbf{x}}_i = \tanh(W^*\mathbf{h} + \mathbf{c})$$

$$\hat{\mathbf{x}}_i = W^*\mathbf{h} + \mathbf{c}$$

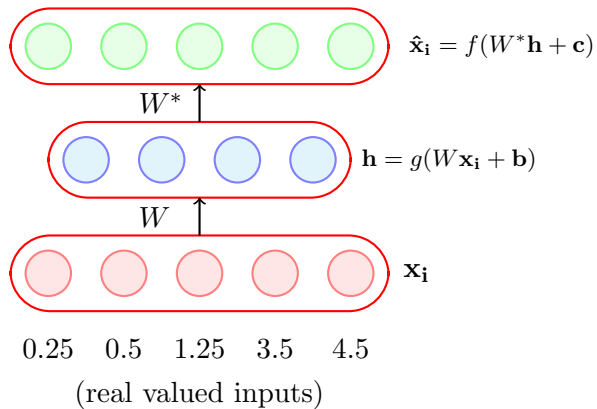


- Suppose all our inputs are real (each $x_{ij} \in \mathbb{R}$)
- Which of the following functions would be most apt for the decoder?

$$\hat{\mathbf{x}}_i = \tanh(W^*\mathbf{h} + \mathbf{c})$$

$$\hat{\mathbf{x}}_i = W^*\mathbf{h} + \mathbf{c}$$

$$\hat{\mathbf{x}}_i = \text{logistic}(W^*\mathbf{h} + \mathbf{c})$$



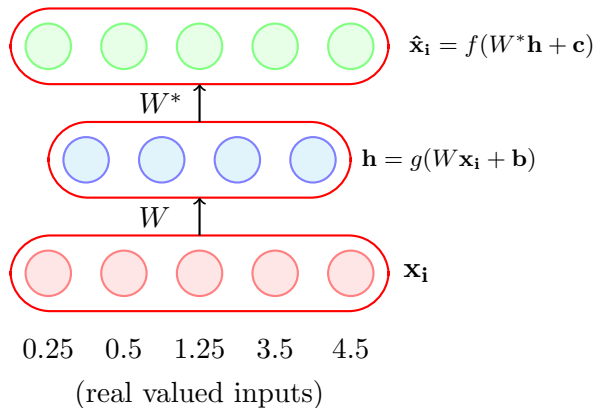
- Suppose all our inputs are real (each $x_{ij} \in \mathbb{R}$)
- Which of the following functions would be most apt for the decoder?

$$\hat{\mathbf{x}}_i = \tanh(W^*\mathbf{h} + \mathbf{c})$$

$$\hat{\mathbf{x}}_i = W^*\mathbf{h} + \mathbf{c}$$

$$\hat{\mathbf{x}}_i = \text{logistic}(W^*\mathbf{h} + \mathbf{c})$$

- What will logistic and tanh do?



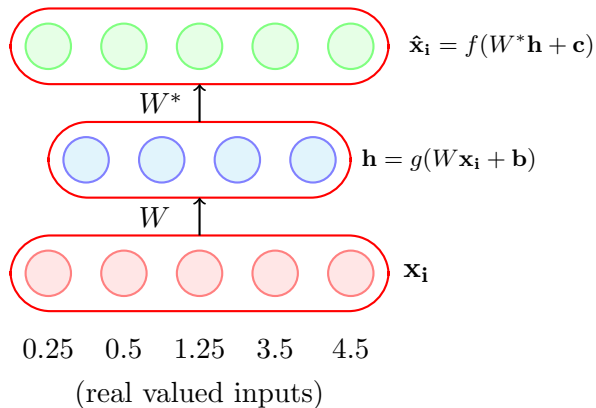
- Suppose all our inputs are real (each $x_{ij} \in \mathbb{R}$)
- Which of the following functions would be most apt for the decoder?

$$\hat{\mathbf{x}}_i = \tanh(W^* \mathbf{h} + \mathbf{c})$$

$$\hat{\mathbf{x}}_i = W^* \mathbf{h} + \mathbf{c}$$

$$\hat{\mathbf{x}}_i = \text{logistic}(W^* \mathbf{h} + \mathbf{c})$$

- What will logistic and tanh do?
- They will restrict the reconstructed $\hat{\mathbf{x}}_i$ to lie between $[0,1]$ or $[-1,1]$ whereas we want $\hat{\mathbf{x}}_i \in \mathbb{R}^n$



Again, g is typically chosen as the sigmoid function

- Suppose all our inputs are real (each $x_{ij} \in \mathbb{R}$)
- Which of the following functions would be most apt for the decoder?

$$\hat{\mathbf{x}}_i = \tanh(W^*\mathbf{h} + \mathbf{c})$$

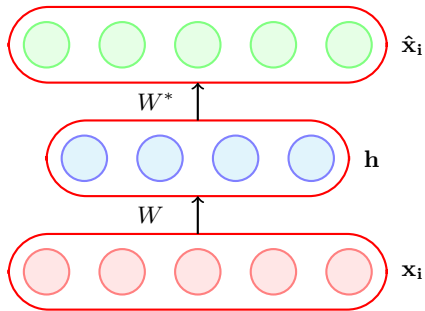
$$\hat{\mathbf{x}}_i = W^*\mathbf{h} + \mathbf{c}$$

$$\hat{\mathbf{x}}_i = \text{logistic}(W^*\mathbf{h} + \mathbf{c})$$

- What will logistic and tanh do?
- They will restrict the reconstructed $\hat{\mathbf{x}}_i$ to lie between $[0,1]$ or $[-1,1]$ whereas we want $\hat{\mathbf{x}}_i \in \mathbb{R}^n$

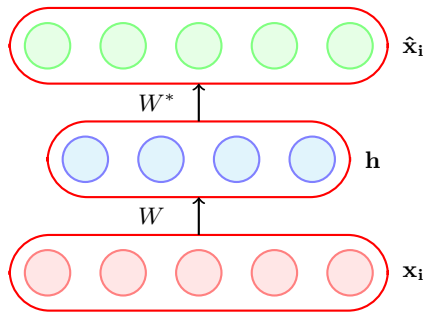
The Road Ahead

- Choice of $f(\mathbf{x}_i)$ and $g(\mathbf{x}_i)$
- Choice of loss function



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

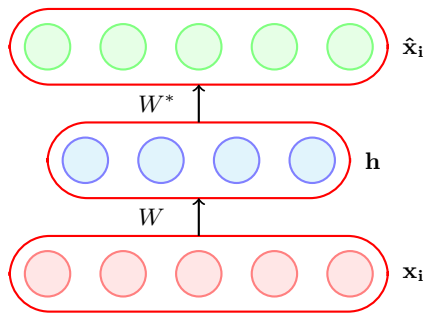
$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

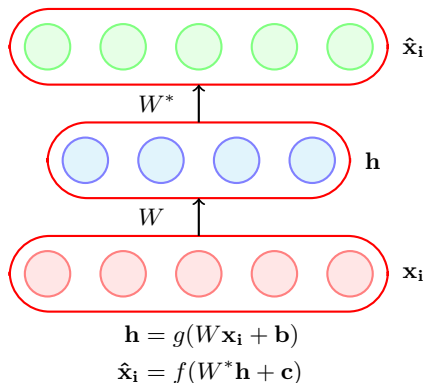
- Consider the case when the inputs are real valued



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$

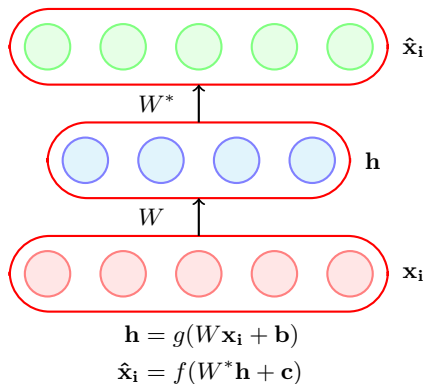
$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

- Consider the case when the inputs are real valued
- The objective of the autoencoder is to reconstruct $\hat{\mathbf{x}}_i$ to be as close to \mathbf{x}_i as possible



- Consider the case when the inputs are real valued
- The objective of the autoencoder is to reconstruct $\hat{\mathbf{x}}_i$ to be as close to \mathbf{x}_i as possible
- This can be formalized using the following objective function:

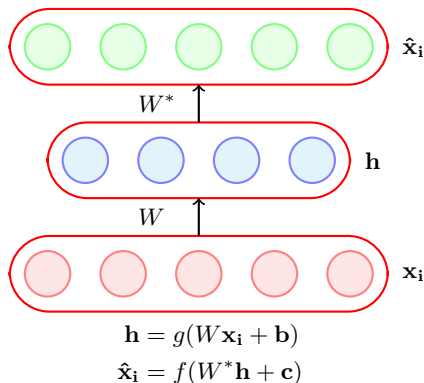
$$\min_{W, W^*, \mathbf{c}, \mathbf{b}} \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2$$



- Consider the case when the inputs are real valued
- The objective of the autoencoder is to reconstruct $\hat{\mathbf{x}}_i$ to be as close to \mathbf{x}_i as possible
- This can be formalized using the following objective function:

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

$$i.e., \min_{W, W^*, \mathbf{c}, \mathbf{b}} \frac{1}{m} \sum_{i=1}^m (\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)$$

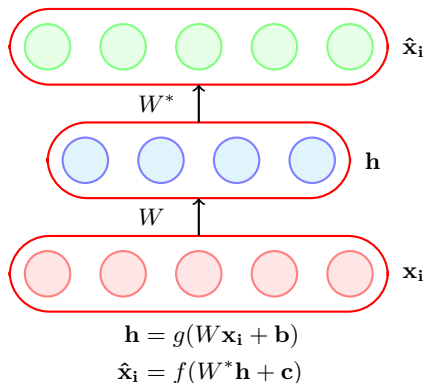


- Consider the case when the inputs are real valued
- The objective of the autoencoder is to reconstruct $\hat{\mathbf{x}}_i$ to be as close to \mathbf{x}_i as possible
- This can be formalized using the following objective function:

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

$$i.e., \min_{W, W^*, \mathbf{c}, \mathbf{b}} \frac{1}{m} \sum_{i=1}^m (\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)$$

- We can then train the autoencoder just like a regular feedforward network using back-propagation



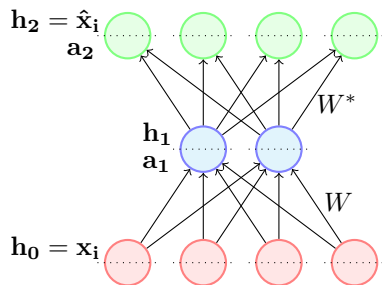
- Consider the case when the inputs are real valued
- The objective of the autoencoder is to reconstruct $\hat{\mathbf{x}}_i$ to be as close to \mathbf{x}_i as possible
- This can be formalized using the following objective function:

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

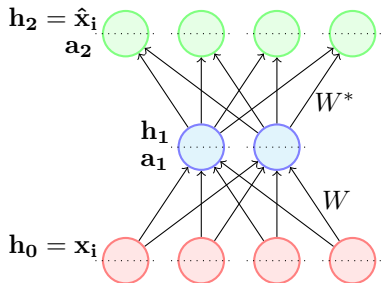
$$i.e., \min_{W, W^*, \mathbf{c}, \mathbf{b}} \frac{1}{m} \sum_{i=1}^m (\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)$$

- We can then train the autoencoder just like a regular feedforward network using back-propagation
- All we need is a formula for $\frac{\partial \mathcal{L}(\theta)}{\partial W^*}$ and $\frac{\partial \mathcal{L}(\theta)}{\partial W}$ which we will see now

$$\mathcal{L}(\theta) = (\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)$$



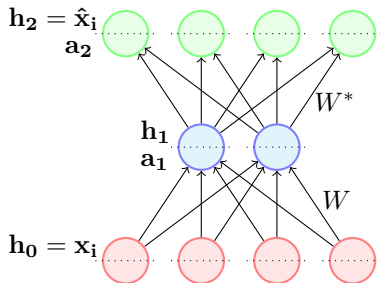
$$\mathcal{L}(\theta) = (\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)$$



- Note that the loss function is shown for only one training example.

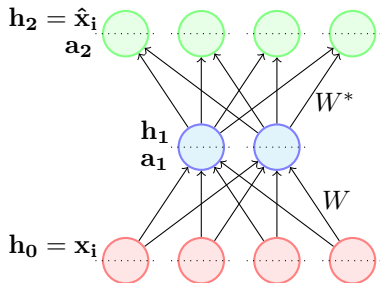
$$\mathcal{L}(\theta) = (\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)$$

$$\bullet \quad \frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \boxed{\frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial W^*}}$$



- Note that the loss function is shown for only one training example.

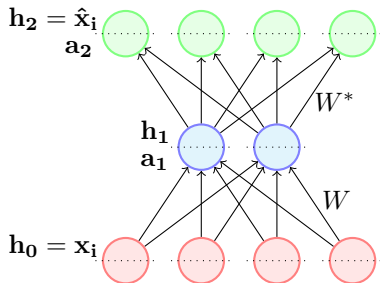
$$\mathcal{L}(\theta) = (\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)$$



- $$\frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \boxed{\frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial W^*}}$$
- $$\frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \boxed{\frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{a}_1} \frac{\partial \mathbf{a}_1}{\partial W}}$$

- Note that the loss function is shown for only one training example.

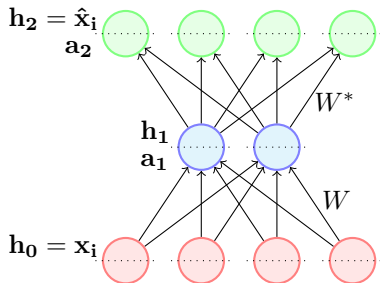
$$\mathcal{L}(\theta) = (\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)$$



- $\frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \boxed{\frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial W^*}}$
- $\frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \boxed{\frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{a}_1} \frac{\partial \mathbf{a}_1}{\partial W}}$
- We have already seen how to calculate the expression in the boxes when we learnt backpropagation

- Note that the loss function is shown for only one training example.

$$\mathcal{L}(\theta) = (\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)$$



- Note that the loss function is shown for only one training example.

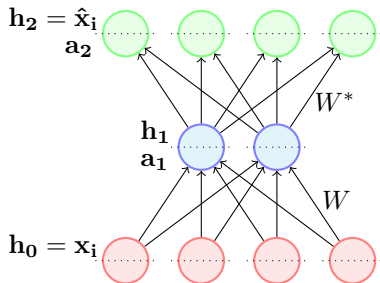
- $$\frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \boxed{\frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial W^*}}$$

- $$\frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \boxed{\frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{a}_1} \frac{\partial \mathbf{a}_1}{\partial W}}$$

- We have already seen how to calculate the expression in the boxes when we learnt backpropagation

$$\frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} = \frac{\partial \mathcal{L}(\theta)}{\partial \hat{\mathbf{x}}_i}$$

$$\mathcal{L}(\theta) = (\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)$$



- Note that the loss function is shown for only one training example.

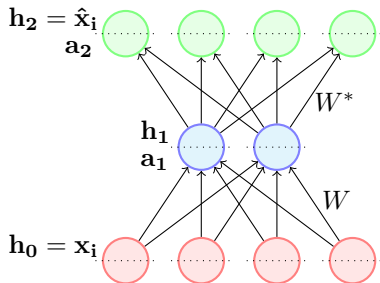
- $$\frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \boxed{\frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial W^*}}$$

- $$\frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \boxed{\frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{a}_1} \frac{\partial \mathbf{a}_1}{\partial W}}$$

- We have already seen how to calculate the expression in the boxes when we learnt backpropagation

$$\begin{aligned} \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} &= \frac{\partial \mathcal{L}(\theta)}{\partial \hat{\mathbf{x}}_i} \\ &= \nabla_{\hat{\mathbf{x}}_i} \{(\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)\} \end{aligned}$$

$$\mathcal{L}(\theta) = (\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)$$



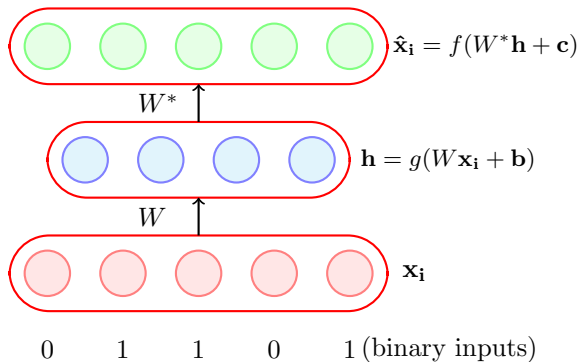
- Note that the loss function is shown for only one training example.

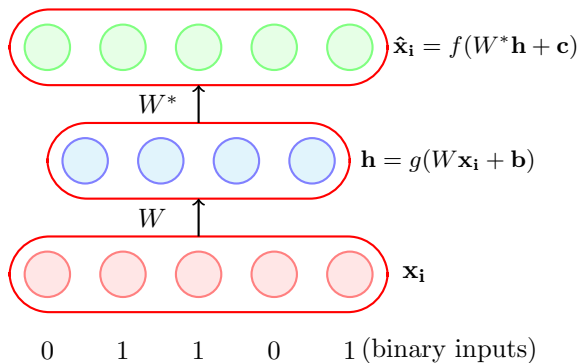
- $$\frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \boxed{\frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial W^*}}$$

- $$\frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \boxed{\frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \frac{\partial \mathbf{a}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{a}_1} \frac{\partial \mathbf{a}_1}{\partial W}}$$

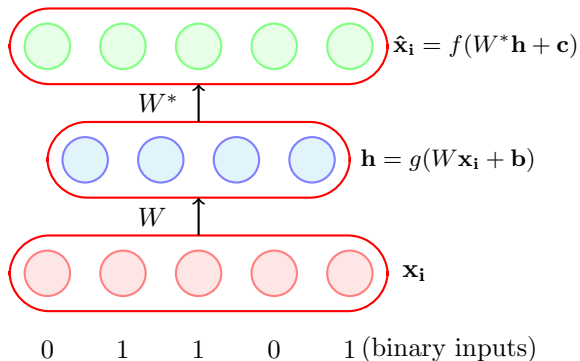
- We have already seen how to calculate the expression in the boxes when we learnt backpropagation

$$\begin{aligned} \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} &= \frac{\partial \mathcal{L}(\theta)}{\partial \hat{\mathbf{x}}_i} \\ &= \nabla_{\hat{\mathbf{x}}_i} \{(\hat{\mathbf{x}}_i - \mathbf{x}_i)^T (\hat{\mathbf{x}}_i - \mathbf{x}_i)\} \\ &= 2(\hat{\mathbf{x}}_i - \mathbf{x}_i) \end{aligned}$$

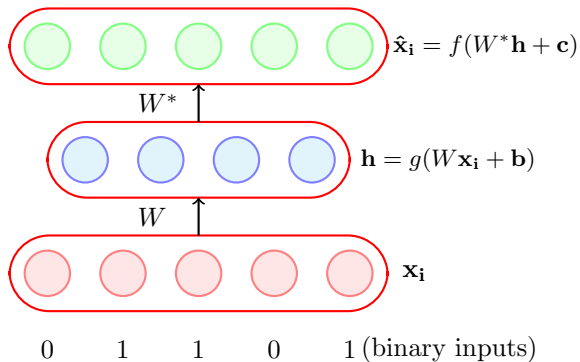




- Consider the case when the inputs are binary



- Consider the case when the inputs are binary
- We use a sigmoid decoder which will produce outputs between 0 and 1, and can be interpreted as probabilities.

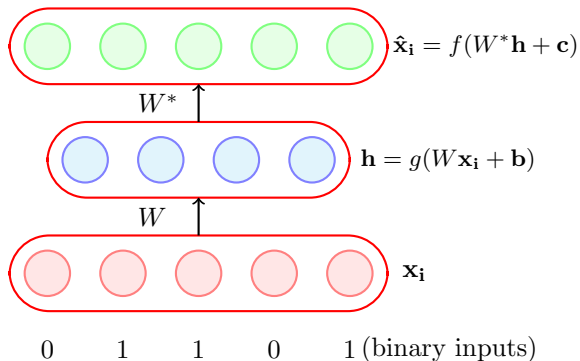


- Consider the case when the inputs are binary

- We use a sigmoid decoder which will produce outputs between 0 and 1, and can be interpreted as probabilities.

- For a single n -dimensional i^{th} input we can use the following loss function

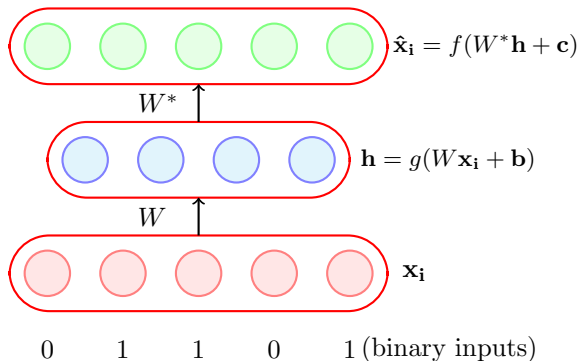
$$\min\left\{-\sum_{j=1}^n (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij}))\right\}$$



What value of \hat{x}_{ij} will minimize this function?

- Consider the case when the inputs are binary
- We use a sigmoid decoder which will produce outputs between 0 and 1, and can be interpreted as probabilities.
- For a single n -dimensional i^{th} input we can use the following loss function

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



What value of \hat{x}_{ij} will minimize this function?

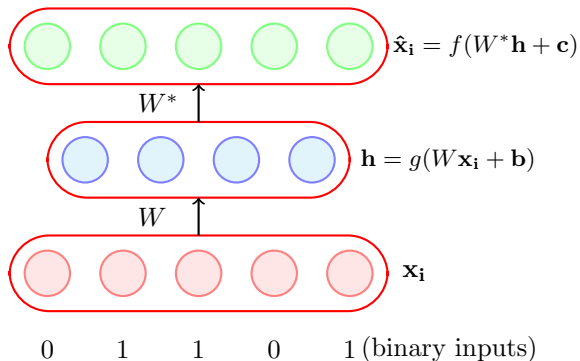
- If $x_{ij} = 1$?

- Consider the case when the inputs are binary

- We use a sigmoid decoder which will produce outputs between 0 and 1, and can be interpreted as probabilities.

- For a single n -dimensional i^{th} input we can use the following loss function

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



What value of \hat{x}_{ij} will minimize this function?

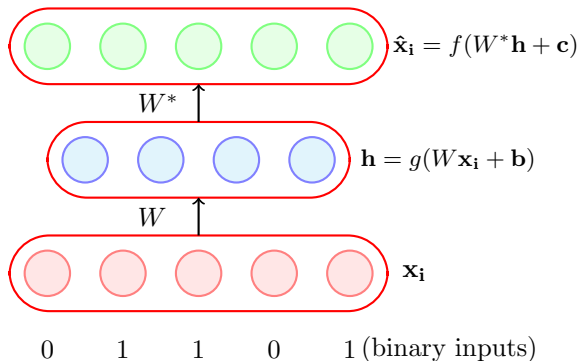
- If $x_{ij} = 1$?
- If $x_{ij} = 0$?

- Consider the case when the inputs are binary

- We use a sigmoid decoder which will produce outputs between 0 and 1, and can be interpreted as probabilities.

- For a single n -dimensional i^{th} input we can use the following loss function

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



What value of \hat{x}_{ij} will minimize this function?

- If $x_{ij} = 1$?
- If $x_{ij} = 0$?

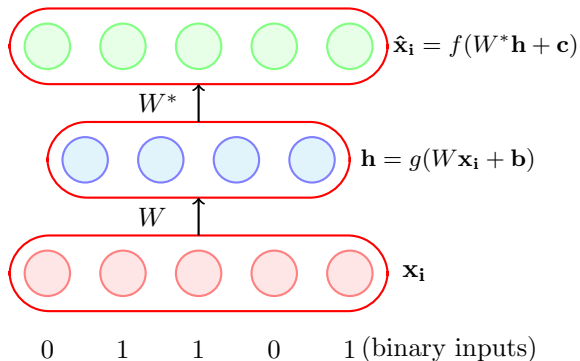
- Consider the case when the inputs are binary

- We use a sigmoid decoder which will produce outputs between 0 and 1, and can be interpreted as probabilities.

- For a single n-dimensional i^{th} input we can use the following loss function

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

- Again we need a formula for $\frac{\partial \mathcal{L}(\theta)}{\partial W^*}$ and $\frac{\partial \mathcal{L}(\theta)}{\partial W}$ to use backpropagation



What value of \hat{x}_{ij} will minimize this function?

- If $x_{ij} = 1$?
- If $x_{ij} = 0$?

Indeed the above function will be minimized when $\hat{x}_{ij} = x_{ij}$!

- Consider the case when the inputs are binary

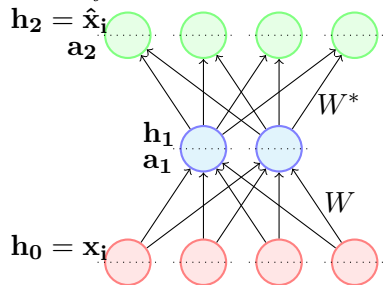
- We use a sigmoid decoder which will produce outputs between 0 and 1, and can be interpreted as probabilities.

- For a single n-dimensional i^{th} input we can use the following loss function

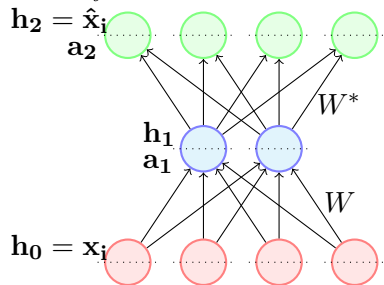
$$\min \left\{ - \sum_{j=1}^n (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij})) \right\}$$

- Again we need a formula for $\frac{\partial \mathcal{L}(\theta)}{\partial W^*}$ and $\frac{\partial \mathcal{L}(\theta)}{\partial W}$ to use backpropagation

$$\mathcal{L}(\theta) = - \sum_{j=1}^n (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij}))$$

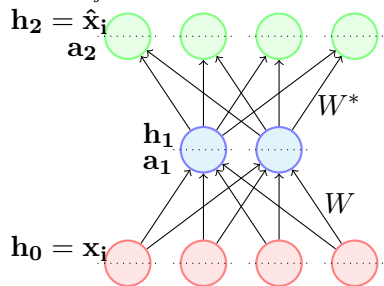


$$\mathcal{L}(\theta) = - \sum_{j=1}^n (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij}))$$



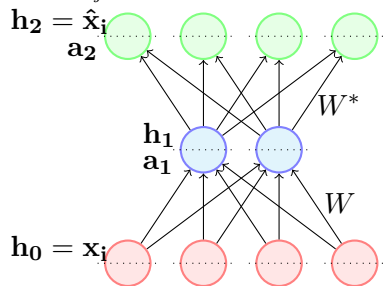
$$\bullet \quad \frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \boxed{\frac{\partial \mathbf{a}_2}{\partial W^*}}$$

$$\mathcal{L}(\theta) = - \sum_{j=1}^n (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij}))$$



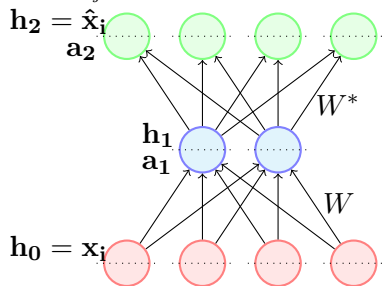
- $\frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \boxed{\frac{\partial \mathbf{a}_2}{\partial W^*}}$
- $\frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \boxed{\frac{\partial \mathbf{a}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{a}_1} \frac{\partial \mathbf{a}_1}{\partial W}}$

$$\mathcal{L}(\theta) = - \sum_{j=1}^n (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij}))$$



- $\frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \boxed{\frac{\partial \mathbf{a}_2}{\partial W^*}}$
- $\frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \boxed{\frac{\partial \mathbf{a}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{a}_1} \frac{\partial \mathbf{a}_1}{\partial W}}$
- We have already seen how to calculate the expressions in the square boxes when we learnt BP

$$\mathcal{L}(\theta) = - \sum_{j=1}^n (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij}))$$



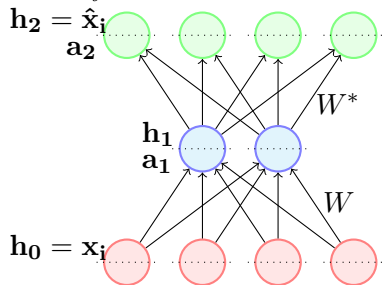
- $\frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \boxed{\frac{\partial \mathbf{a}_2}{\partial W^*}}$
- $\frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \boxed{\frac{\partial \mathbf{a}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{a}_1} \frac{\partial \mathbf{a}_1}{\partial W}}$

- We have already seen how to calculate the expressions in the square boxes when we learnt BP
- The first two terms on RHS can be computed as:

$$\frac{\partial \mathcal{L}(\theta)}{\partial h_{2j}} = -\frac{x_{ij}}{\hat{x}_{ij}} + \frac{1 - x_{ij}}{1 - \hat{x}_{ij}}$$

$$\frac{\partial h_{2j}}{\partial a_{2j}} = \sigma(a_{2j})(1 - \sigma(a_{2j}))$$

$$\mathcal{L}(\theta) = - \sum_{j=1}^n (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij}))$$



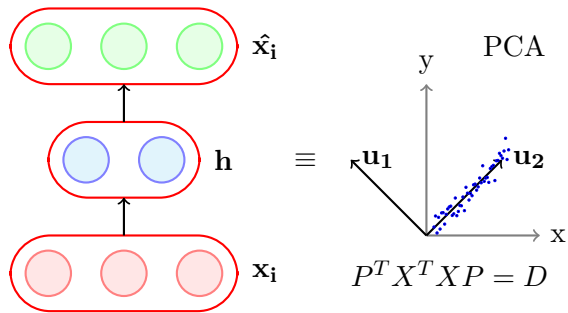
$$\frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} = \begin{pmatrix} \frac{\partial \mathcal{L}(\theta)}{\partial h_{21}} \\ \frac{\partial \mathcal{L}(\theta)}{\partial h_{22}} \\ \vdots \\ \frac{\partial \mathcal{L}(\theta)}{\partial h_{2n}} \end{pmatrix}$$

- $\frac{\partial \mathcal{L}(\theta)}{\partial W^*} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \boxed{\frac{\partial \mathbf{a}_2}{\partial W^*}}$
- $\frac{\partial \mathcal{L}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{a}_2} \boxed{\frac{\partial \mathbf{a}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{a}_1} \frac{\partial \mathbf{a}_1}{\partial W}}$
- We have already seen how to calculate the expressions in the square boxes when we learnt BP
- The first two terms on RHS can be computed as:

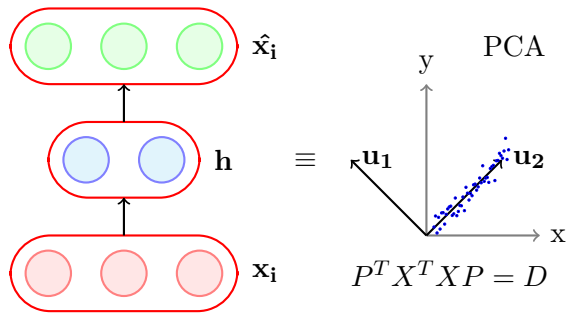
$$\frac{\partial \mathcal{L}(\theta)}{\partial h_{2j}} = -\frac{x_{ij}}{\hat{x}_{ij}} + \frac{1 - x_{ij}}{1 - \hat{x}_{ij}}$$

$$\frac{\partial h_{2j}}{\partial a_{2j}} = \sigma(a_{2j})(1 - \sigma(a_{2j}))$$

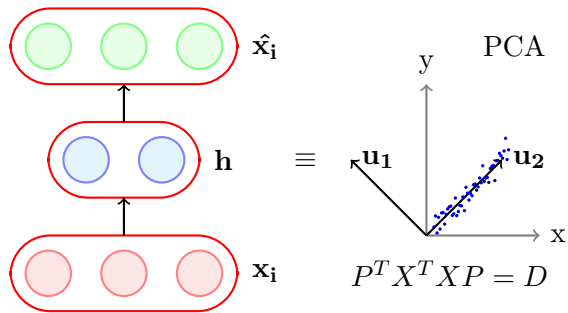
Module 7.2: Link between PCA and Autoencoders



- We will now see that the encoder part of an autoencoder is equivalent to PCA if we

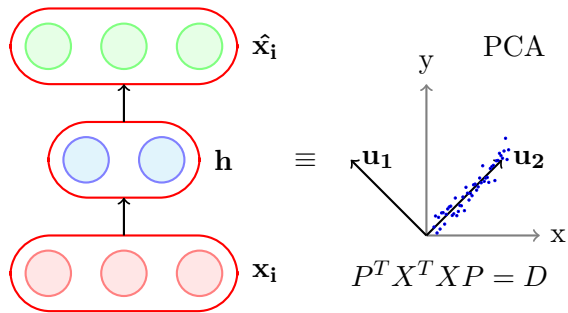


- We will now see that the encoder part of an autoencoder is equivalent to PCA if we
 - use a linear encoder



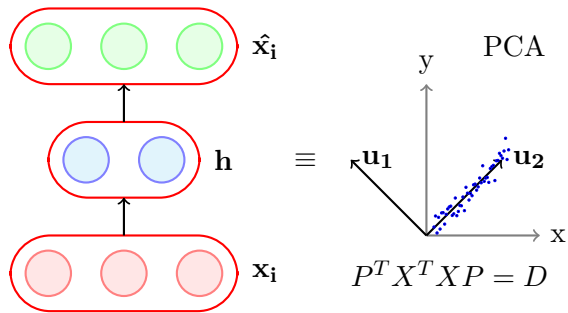
- We will now see that the encoder part of an autoencoder is equivalent to PCA if we

- use a linear encoder
- use a linear decoder



- We will now see that the encoder part of an autoencoder is equivalent to PCA if we

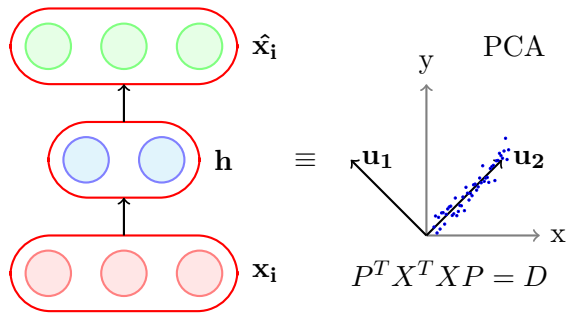
- use a linear encoder
- use a linear decoder
- use squared error loss function



- We will now see that the encoder part of an autoencoder is equivalent to PCA if we

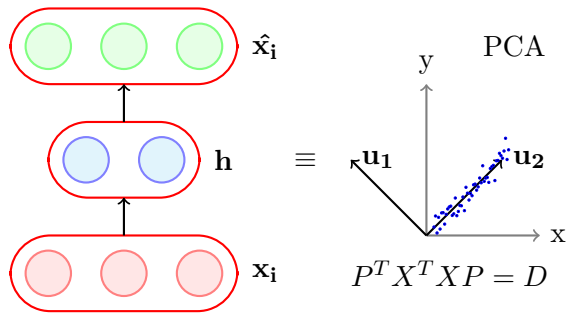
- use a linear encoder
- use a linear decoder
- use squared error loss function
- normalize the inputs to

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



- First let us consider the implication of normalizing the inputs to

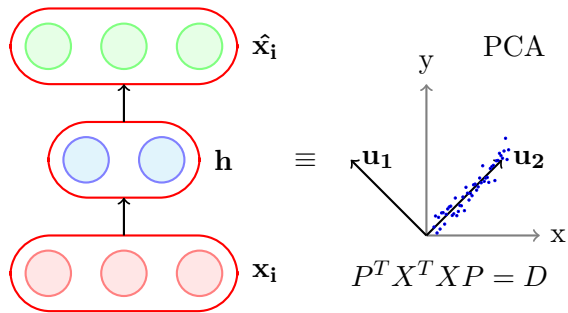
$$\hat{x}_{ij} = \frac{1}{\sqrt{m}} \left(x_{ij} - \frac{1}{m} \sum_{k=1}^m x_{kj} \right)$$



- First let us consider the implication of normalizing the inputs to

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

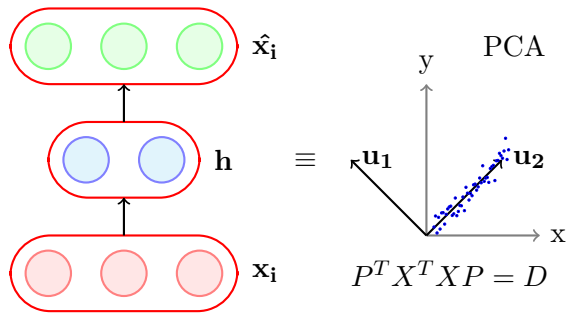
- The operation in the bracket ensures that the data now has 0 mean along each dimension j (we are subtracting the mean)



- First let us consider the implication of normalizing the inputs to

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

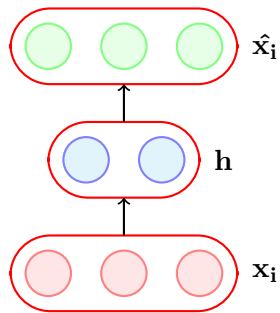
- The operation in the bracket ensures that the data now has 0 mean along each dimension j (we are subtracting the mean)
- Let X' be this zero mean data matrix then what the above normalization gives us is $X = \frac{1}{\sqrt{m}} X'$



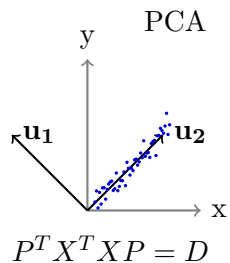
- First let us consider the implication of normalizing the inputs to

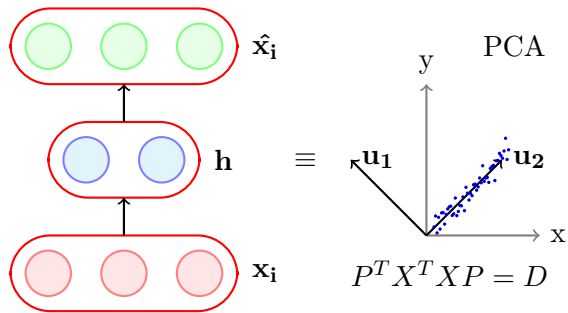
$$\hat{x}_{ij} = \frac{1}{\sqrt{m}} \left(x_{ij} - \frac{1}{m} \sum_{k=1}^m x_{kj} \right)$$

- The operation in the bracket ensures that the data now has 0 mean along each dimension j (we are subtracting the mean)
- Let X' be this zero mean data matrix then what the above normalization gives us is $X = \frac{1}{\sqrt{m}} X'$
- Now $(X)^T X = \frac{1}{m} (X')^T X'$ is the covariance matrix (recall that covariance matrix plays an important role in PCA)

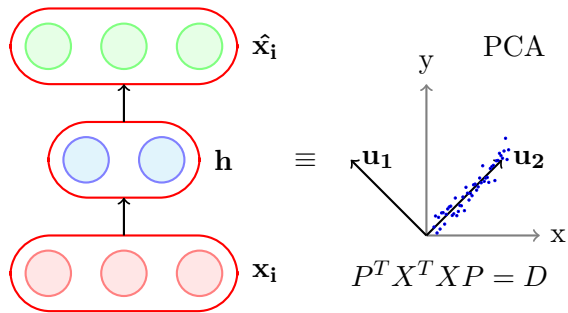


\equiv

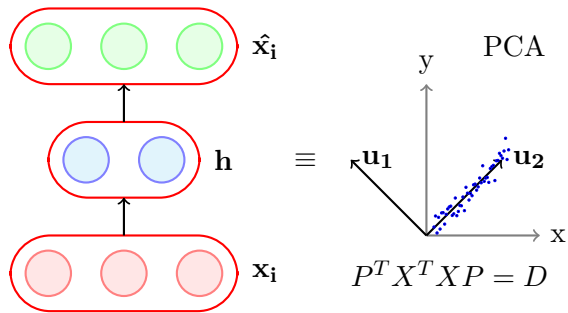




- First we will show that if we use linear decoder and a squared error loss function then

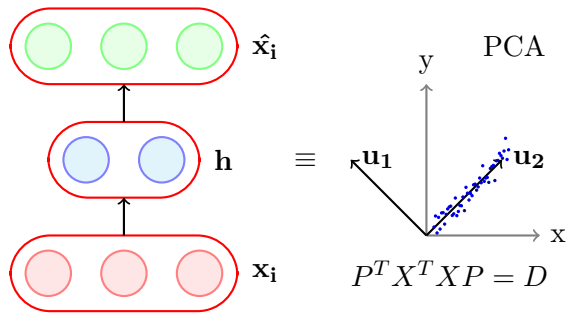


- First we will show that if we use linear decoder and a squared error loss function then
- The optimal solution to the following objective function



- First we will show that if we use linear decoder and a squared error loss function then
- The optimal solution to the following objective function

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



- First we will show that if we use linear decoder and a squared error loss function then
- The optimal solution to the following objective function

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

is obtained when we use a linear encoder.

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

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

- This is equivalent to

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

- This is equivalent to

$$\min_{W^*H} (\|X - HW^*\|_F)^2$$

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

- This is equivalent to

$$\min_{W^*H} (\|X - HW^*\|_F)^2 \qquad \|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2}$$

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

- This is equivalent to

$$\min_{W^*H} (\|X - HW^*\|_F)^2 \qquad \|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2}$$

(just writing the expression (1) in matrix form and using the definition of $\|A\|_F$) (we are ignoring the biases)

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

- This is equivalent to

$$\min_{W^*H} (\|X - HW^*\|_F)^2 \qquad \|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2}$$

(just writing the expression (1) in matrix form and using the definition of $\|A\|_F$) (we are ignoring the biases)

- From SVD we know that optimal solution to the above problem is given by

$$HW^* = U_{\cdot, \leq k} \Sigma_{k,k} V_{\cdot, \leq k}^T$$

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

- This is equivalent to

$$\min_{W^*H} (\|X - HW^*\|_F)^2 \qquad \|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2}$$

(just writing the expression (1) in matrix form and using the definition of $\|A\|_F$) (we are ignoring the biases)

- From SVD we know that optimal solution to the above problem is given by

$$HW^* = U_{\cdot, \leq k} \Sigma_{k,k} V_{\cdot, \leq k}^T$$

- By matching variables one possible solution is

$$H = U_{\cdot, \leq k} \Sigma_{k,k}$$

$$W^* = V_{\cdot, \leq k}^T$$

We will now show that H is a linear encoding and find an expression for the encoder weights W

We will now show that H is a linear encoding and find an expression for the encoder weights W

$$H = U_{.,\leq k} \Sigma_{k,k}$$

We will now show that H is a linear encoding and find an expression for the encoder weights W

$$H = U_{., \leq k} \Sigma_{k,k}$$

$$= (XX^T)(XX^T)^{-1}U_{., \leq K} \Sigma_{k,k}$$

$$(pre-multiplying (XX^T)(XX^T)^{-1} = I)$$

We will now show that H is a linear encoding and find an expression for the encoder weights W

$$\begin{aligned} H &= U_{., \leq k} \Sigma_{k,k} \\ &= (XX^T)(XX^T)^{-1} U_{., \leq K} \Sigma_{k,k} && (\text{pre-multiplying } (XX^T)(XX^T)^{-1} = I) \\ &= (XV\Sigma^T U^T)(U\Sigma V^T V\Sigma^T U^T)^{-1} U_{., \leq k} \Sigma_{k,k} && (\text{using } X = U\Sigma V^T) \end{aligned}$$

We will now show that H is a linear encoding and find an expression for the encoder weights W

$$\begin{aligned} H &= U_{.,\leq k} \Sigma_{k,k} \\ &= (XX^T)(XX^T)^{-1} U_{.,\leq K} \Sigma_{k,k} && (\text{pre-multiplying } (XX^T)(XX^T)^{-1} = I) \\ &= (XV\Sigma^T U^T)(U\Sigma V^T V\Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k} && (\text{using } X = U\Sigma V^T) \\ &= XV\Sigma^T U^T (U\Sigma\Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k} && (V^T V = I) \end{aligned}$$

We will now show that H is a linear encoding and find an expression for the encoder weights W

$$\begin{aligned} H &= U_{.,\leq k} \Sigma_{k,k} \\ &= (XX^T)(XX^T)^{-1} U_{.,\leq K} \Sigma_{k,k} && \text{(pre-multiplying } (XX^T)(XX^T)^{-1} = I) \\ &= (XV\Sigma^T U^T)(U\Sigma V^T V\Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k} && \text{(using } X = U\Sigma V^T) \\ &= XV\Sigma^T U^T (U\Sigma\Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k} && (V^T V = I) \\ &= XV\Sigma^T U^T U (\Sigma\Sigma^T)^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && ((ABC)^{-1} = C^{-1}B^{-1}A^{-1}) \end{aligned}$$

We will now show that H is a linear encoding and find an expression for the encoder weights W

$$\begin{aligned} H &= U_{.,\leq k} \Sigma_{k,k} \\ &= (XX^T)(XX^T)^{-1}U_{.,\leq K} \Sigma_{k,k} && \text{(pre-multiplying } (XX^T)(XX^T)^{-1} = I) \\ &= (XV\Sigma^T U^T)(U\Sigma V^T V\Sigma^T U^T)^{-1}U_{.,\leq k} \Sigma_{k,k} && \text{(using } X = U\Sigma V^T) \\ &= XV\Sigma^T U^T (U\Sigma\Sigma^T U^T)^{-1}U_{.,\leq k} \Sigma_{k,k} && (V^T V = I) \\ &= XV\Sigma^T U^T U (\Sigma\Sigma^T)^{-1}U^T U_{.,\leq k} \Sigma_{k,k} && ((ABC)^{-1} = C^{-1}B^{-1}A^{-1}) \\ &= XV\Sigma^T (\Sigma\Sigma^T)^{-1}U^T U_{.,\leq k} \Sigma_{k,k} && (U^T U = I) \end{aligned}$$

We will now show that H is a linear encoding and find an expression for the encoder weights W

$$\begin{aligned}
 H &= U_{.,\leq k} \Sigma_{k,k} \\
 &= (XX^T)(XX^T)^{-1} U_{.,\leq K} \Sigma_{k,k} && \text{(pre-multiplying } (XX^T)(XX^T)^{-1} = I) \\
 &= (XV\Sigma^T U^T)(U\Sigma V^T V\Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k} && \text{(using } X = U\Sigma V^T) \\
 &= XV\Sigma^T U^T (U\Sigma\Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k} && (V^T V = I) \\
 &= XV\Sigma^T U^T U (\Sigma\Sigma^T)^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && ((ABC)^{-1} = C^{-1}B^{-1}A^{-1}) \\
 &= XV\Sigma^T (\Sigma\Sigma^T)^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && (U^T U = I) \\
 &= XV\Sigma^T \Sigma^{T^{-1}} \Sigma^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && ((AB)^{-1} = B^{-1}A^{-1})
 \end{aligned}$$

We will now show that H is a linear encoding and find an expression for the encoder weights W

$$\begin{aligned}
 H &= U_{.,\leq k} \Sigma_{k,k} \\
 &= (XX^T)(XX^T)^{-1} U_{.,\leq K} \Sigma_{k,k} && \text{(pre-multiplying } (XX^T)(XX^T)^{-1} = I) \\
 &= (XV\Sigma^T U^T)(U\Sigma V^T V\Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k} && \text{(using } X = U\Sigma V^T) \\
 &= XV\Sigma^T U^T (U\Sigma\Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k} && (V^T V = I) \\
 &= XV\Sigma^T U^T U (\Sigma\Sigma^T)^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && ((ABC)^{-1} = C^{-1}B^{-1}A^{-1}) \\
 &= XV\Sigma^T (\Sigma\Sigma^T)^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && (U^T U = I) \\
 &= XV\Sigma^T \Sigma^{T^{-1}} \Sigma^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && ((AB)^{-1} = B^{-1}A^{-1}) \\
 &= XV\Sigma^{-1} I_{.,\leq k} \Sigma_{k,k} && (U^T U_{.,\leq k} = I_{.,\leq k})
 \end{aligned}$$

We will now show that H is a linear encoding and find an expression for the encoder weights W

$$\begin{aligned}
 H &= U_{.,\leq k} \Sigma_{k,k} \\
 &= (XX^T)(XX^T)^{-1} U_{.,\leq K} \Sigma_{k,k} && \text{(pre-multiplying } (XX^T)(XX^T)^{-1} = I) \\
 &= (XV\Sigma^T U^T)(U\Sigma V^T V\Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k} && \text{(using } X = U\Sigma V^T) \\
 &= XV\Sigma^T U^T (U\Sigma\Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k} && (V^T V = I) \\
 &= XV\Sigma^T U^T U (\Sigma\Sigma^T)^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && ((ABC)^{-1} = C^{-1}B^{-1}A^{-1}) \\
 &= XV\Sigma^T (\Sigma\Sigma^T)^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && (U^T U = I) \\
 &= XV\Sigma^T \Sigma^{T^{-1}} \Sigma^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && ((AB)^{-1} = B^{-1}A^{-1}) \\
 &= XV\Sigma^{-1} I_{.,\leq k} \Sigma_{k,k} && (U^T U_{.,\leq k} = I_{.,\leq k}) \\
 &= XVI_{.,\leq k} && (\Sigma^{-1} I_{.,\leq k} = \Sigma_{k,k}^{-1})
 \end{aligned}$$

We will now show that H is a linear encoding and find an expression for the encoder weights W

$$\begin{aligned}
 H &= U_{.,\leq k} \Sigma_{k,k} \\
 &= (XX^T)(XX^T)^{-1} U_{.,\leq K} \Sigma_{k,k} && \text{(pre-multiplying } (XX^T)(XX^T)^{-1} = I) \\
 &= (XV\Sigma^T U^T)(U\Sigma V^T V\Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k} && \text{(using } X = U\Sigma V^T) \\
 &= XV\Sigma^T U^T (U\Sigma\Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k} && (V^T V = I) \\
 &= XV\Sigma^T U^T U (\Sigma\Sigma^T)^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && ((ABC)^{-1} = C^{-1}B^{-1}A^{-1}) \\
 &= XV\Sigma^T (\Sigma\Sigma^T)^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && (U^T U = I) \\
 &= XV\Sigma^T \Sigma^{T^{-1}} \Sigma^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && ((AB)^{-1} = B^{-1}A^{-1}) \\
 &= XV\Sigma^{-1} I_{.,\leq k} \Sigma_{k,k} && (U^T U_{.,\leq k} = I_{.,\leq k}) \\
 &= XV I_{.,\leq k} && (\Sigma^{-1} I_{.,\leq k} = \Sigma_{k,k}^{-1}) \\
 H &= XV_{.,\leq k}
 \end{aligned}$$

We will now show that H is a linear encoding and find an expression for the encoder weights W

$$\begin{aligned}
 H &= U_{.,\leq k} \Sigma_{k,k} \\
 &= (XX^T)(XX^T)^{-1} U_{.,\leq K} \Sigma_{k,k} && \text{(pre-multiplying } (XX^T)(XX^T)^{-1} = I) \\
 &= (XV\Sigma^T U^T)(U\Sigma V^T V\Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k} && \text{(using } X = U\Sigma V^T) \\
 &= XV\Sigma^T U^T (U\Sigma\Sigma^T U^T)^{-1} U_{.,\leq k} \Sigma_{k,k} && (V^T V = I) \\
 &= XV\Sigma^T U^T U (\Sigma\Sigma^T)^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && ((ABC)^{-1} = C^{-1}B^{-1}A^{-1}) \\
 &= XV\Sigma^T (\Sigma\Sigma^T)^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && (U^T U = I) \\
 &= XV\Sigma^T \Sigma^{T^{-1}} \Sigma^{-1} U^T U_{.,\leq k} \Sigma_{k,k} && ((AB)^{-1} = B^{-1}A^{-1}) \\
 &= XV\Sigma^{-1} I_{.,\leq k} \Sigma_{k,k} && (U^T U_{.,\leq k} = I_{.,\leq k}) \\
 &= X V I_{.,\leq k} && (\Sigma^{-1} I_{.,\leq k} = \Sigma_{k,k}^{-1}) \\
 H &= X V_{.,\leq k}
 \end{aligned}$$

Thus H is a linear transformation of X and $W = V_{.,\leq k}$

- We have encoder $W = V_{\cdot, \leq k}$

- We have encoder $W = V_{.,\leq k}$
- From SVD, we know that V is the matrix of eigen vectors of $X^T X$

- We have encoder $W = V_{\cdot, \leq k}$
- From SVD, we know that V is the matrix of eigen vectors of $X^T X$
- From PCA, we know that P is the matrix of the eigen vectors of the covariance matrix

- We have encoder $W = V_{\cdot, \leq k}$
- From SVD, we know that V is the matrix of eigen vectors of $X^T X$
- From PCA, we know that P is the matrix of the eigen vectors of the covariance matrix
- We saw earlier that, if entries of X are normalized by

- We have encoder $W = V_{\cdot, \leq k}$
- From SVD, we know that V is the matrix of eigen vectors of $X^T X$
- From PCA, we know that P is the matrix of the eigen vectors of the covariance matrix
- We saw earlier that, if entries of X are normalized by

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

- We have encoder $W = V_{\cdot, \leq k}$
- From SVD, we know that V is the matrix of eigen vectors of $X^T X$
- From PCA, we know that P is the matrix of the eigen vectors of the covariance matrix
- We saw earlier that, if entries of X are normalized by

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

then $X^T X$ is indeed the covariance matrix

- We have encoder $W = V_{\cdot, \leq k}$
- From SVD, we know that V is the matrix of eigen vectors of $X^T X$
- From PCA, we know that P is the matrix of the eigen vectors of the covariance matrix
- We saw earlier that, if entries of X are normalized by

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

then $X^T X$ is indeed the covariance matrix

- Thus, the encoder matrix for linear autoencoder(W) and the projection matrix(P) for PCA could indeed be the same. Hence proved

Remember

The encoder of a linear autoencoder is equivalent to PCA if we

Remember

The encoder of a linear autoencoder is equivalent to PCA if we

- use a linear encoder

Remember

The encoder of a linear autoencoder is equivalent to PCA if we

- use a linear encoder
- use a linear decoder

Remember

The encoder of a linear autoencoder is equivalent to PCA if we

- use a linear encoder
- use a linear decoder
- use a squared error loss function

Remember

The encoder of a linear autoencoder is equivalent to PCA if we

- use a linear encoder
- use a linear decoder
- use a squared error loss function
- and normalize the inputs to

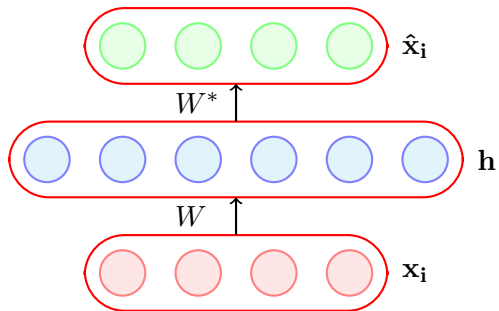
Remember

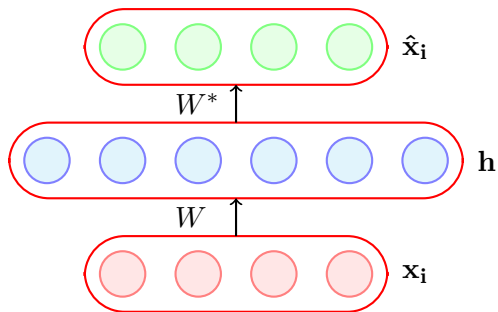
The encoder of a linear autoencoder is equivalent to PCA if we

- use a linear encoder
- use a linear decoder
- use a squared error loss function
- and normalize the inputs to

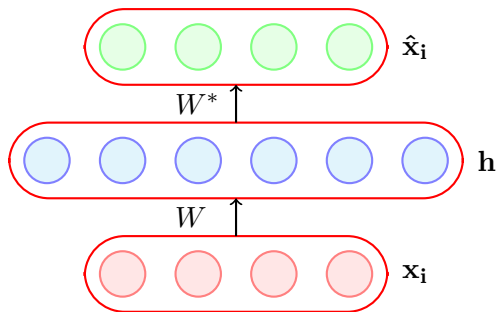
$$\hat{x}_{ij} = \frac{1}{\sqrt{m}} \left(x_{ij} - \frac{1}{m} \sum_{k=1}^m x_{kj} \right)$$

Module 7.3: Regularization in autoencoders (Motivation)

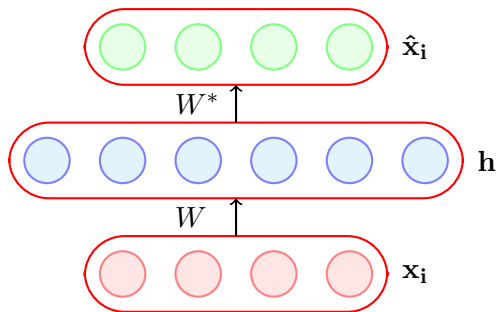




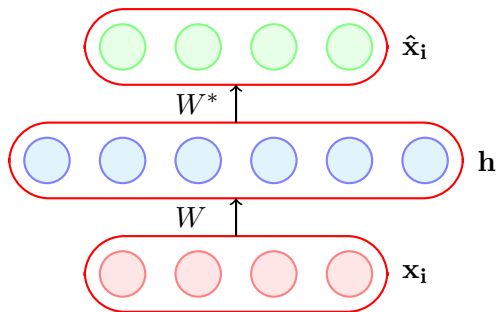
- While poor generalization could happen even in undercomplete autoencoders it is an even more serious problem for overcomplete auto encoders



- While poor generalization could happen even in undercomplete 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 $\hat{\mathbf{x}}_i$

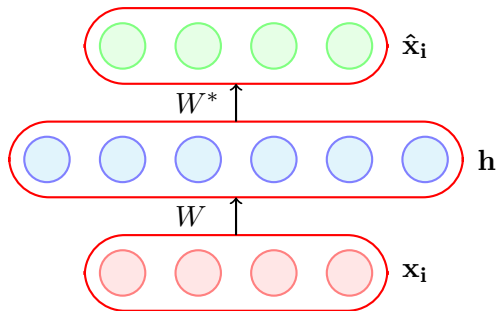


- While poor generalization could happen even in undercomplete autoencoders it is an even more serious problem for overcomplete auto encoders
- Here, (as stated earlier) the model can simply learn to copy \mathbf{x}_i to \mathbf{h} and then \mathbf{h} to $\hat{\mathbf{x}}_i$
- To avoid poor generalization,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



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

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

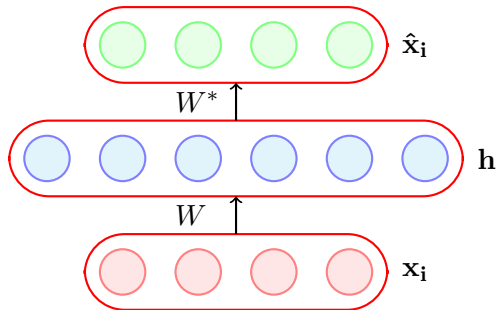


- The simplest solution is to add a L_2 -regularization term to the objective function

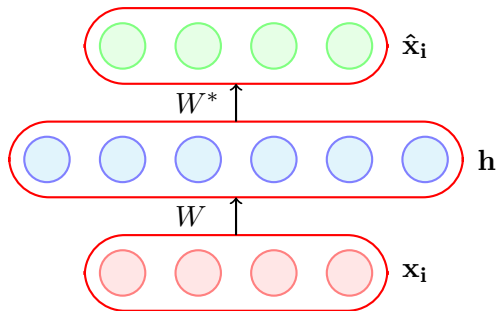
$$\min_{\theta, w, w^*, \mathbf{b}, \mathbf{c}} \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2 + \lambda \|\theta\|^2$$

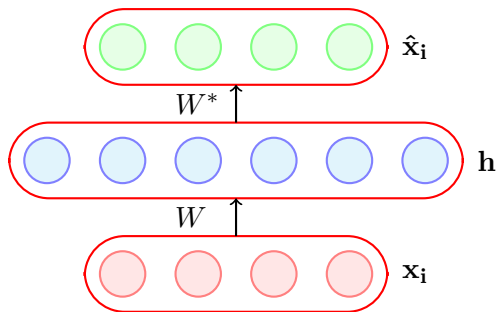
- This is very easy to implement and just adds a term λW to the gradient $\frac{\partial \mathcal{L}(\theta)}{\partial W}$ (and similarly for other parameters)

- 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., $W^* = W^T$

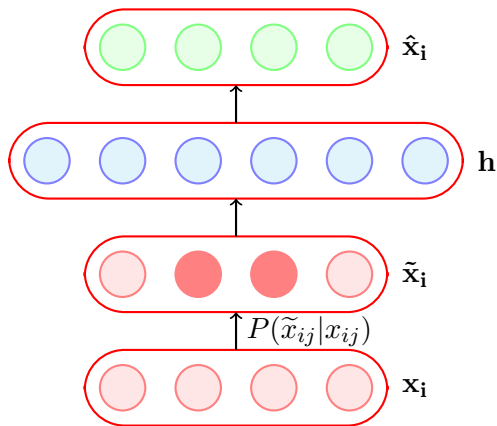


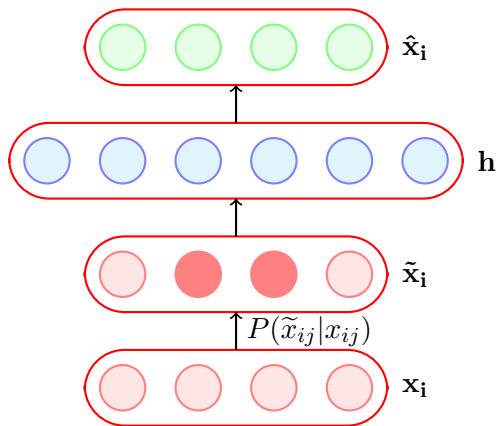


- Another trick is to tie the weights of the encoder and decoder i.e., $W^* = W^T$
- This effectively reduces the capacity of Autoencoder and acts as a regularizer

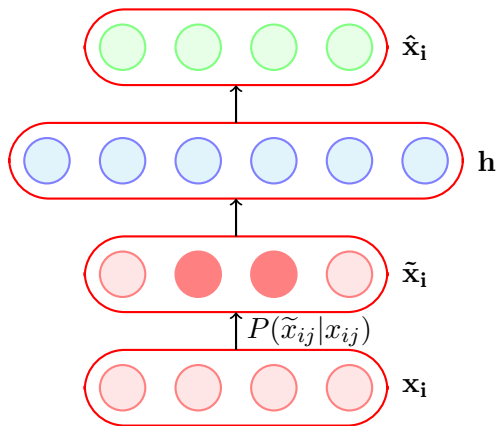
Module 7.4: Denoising Autoencoders

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



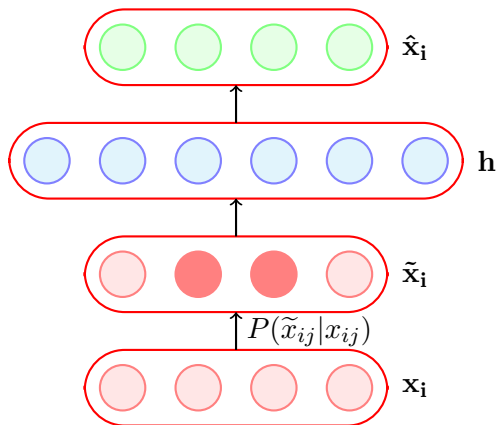


- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\tilde{x}_{ij}|x_{ij})$) before feeding it to the network
- A simple $P(\tilde{x}_{ij}|x_{ij})$ used in practice is the following



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

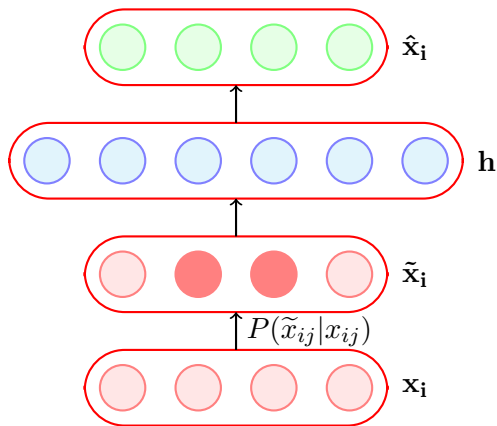
$$P(\tilde{x}_{ij} = 0|x_{ij}) = q$$



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

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

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



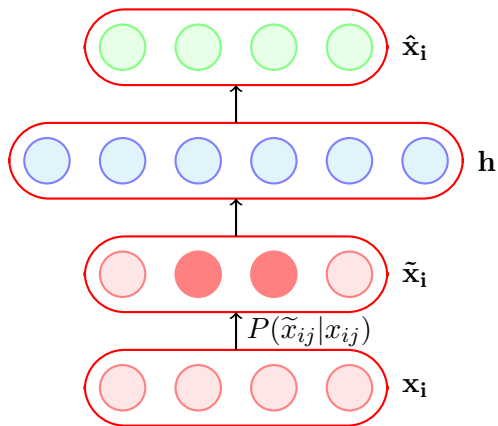
- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\tilde{x}_{ij} | x_{ij})$) before feeding it to the network
- A simple $P(\tilde{x}_{ij} | x_{ij})$ used in practice is the following

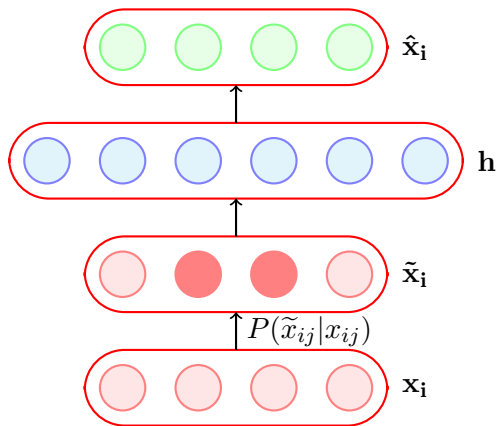
$$P(\tilde{x}_{ij} = 0 | x_{ij}) = q$$

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

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

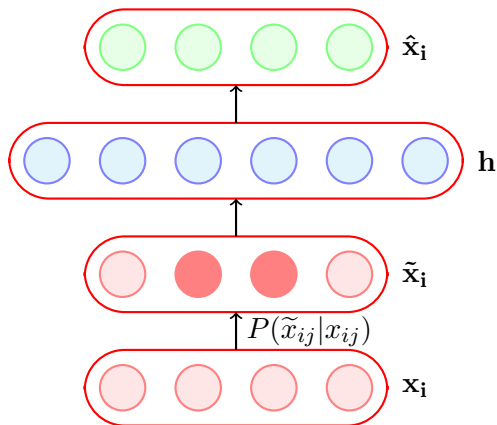
- How does this help ?





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

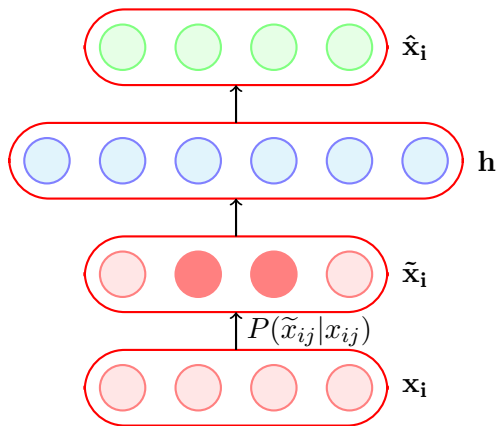
$$\arg \min_{\theta} \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2$$



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

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

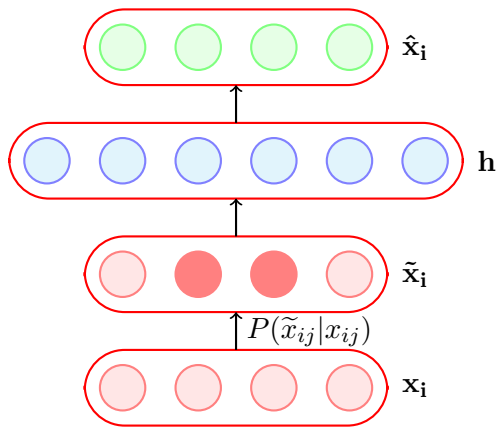
- It no longer makes sense for the model to copy the corrupted $\tilde{\mathbf{x}}_i$ into $h(\tilde{\mathbf{x}}_i)$ and then into $\hat{\mathbf{x}}_i$ (the objective function will not be minimized by doing so)



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

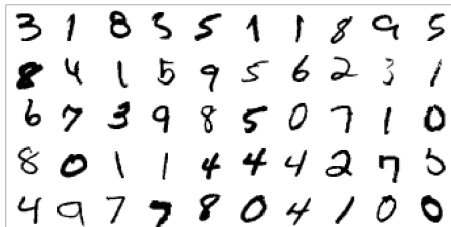
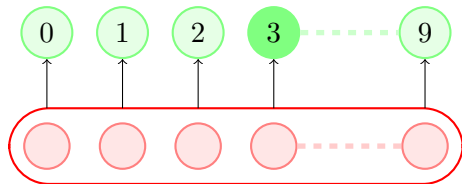


Figure: MNIST Data



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



28*28

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

Task: Hand-written digit recognition

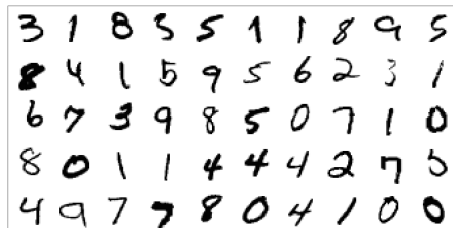


Figure: MNIST Data

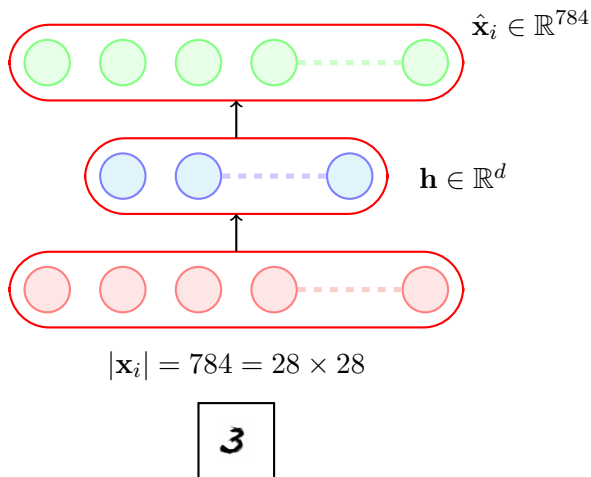


Figure: AE approach (first learn important characteristics of data)

Task: Hand-written digit recognition

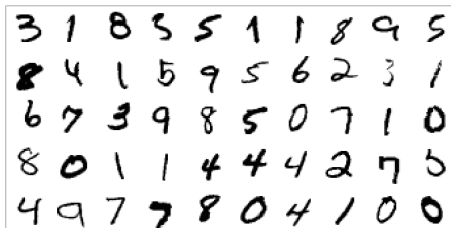


Figure: MNIST Data

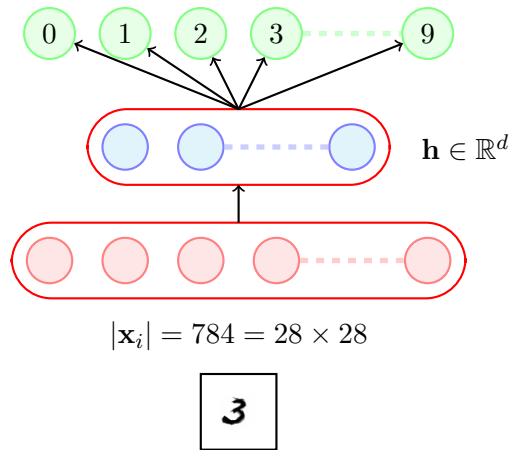
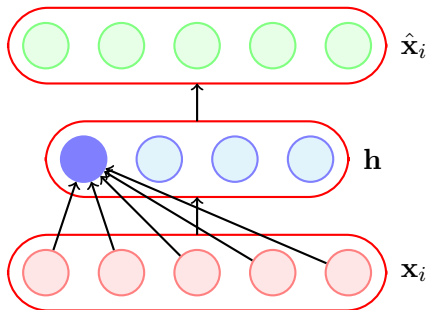
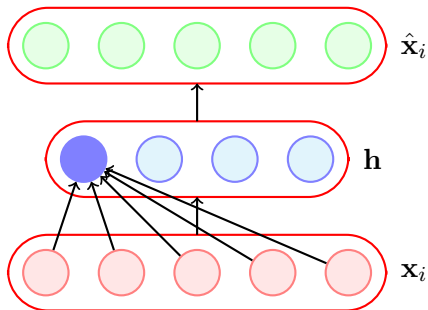


Figure: AE approach (and then train a classifier on top of this hidden representation)

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



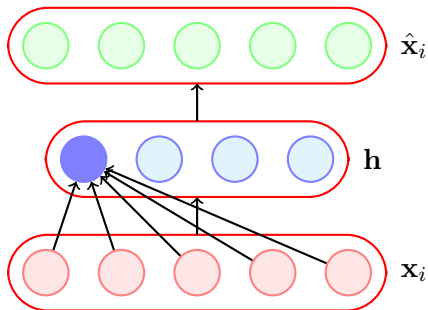
- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration \mathbf{x}_i



- 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,

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

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

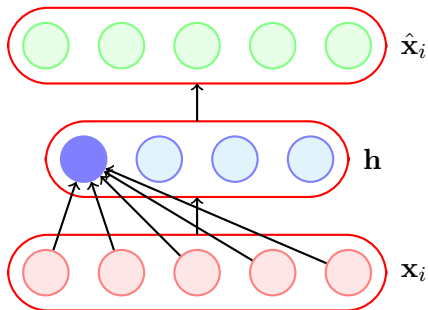


- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration \mathbf{x}_i
- For example,

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

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)

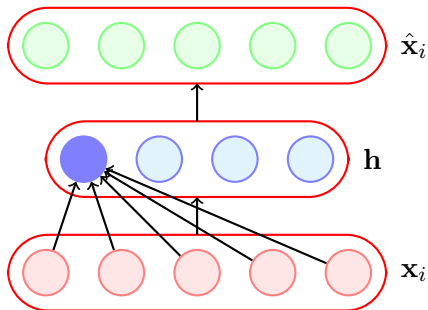


- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration \mathbf{x}_i
- For example,

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

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 $\|\mathbf{x}_i\| = 1$



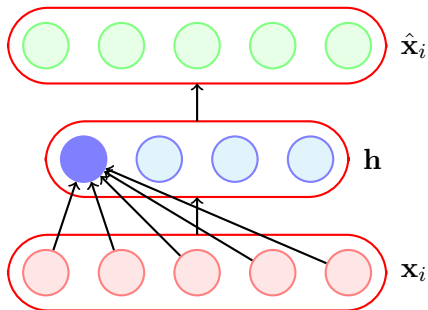
$$\begin{aligned} & \max_{\mathbf{x}_i} \{W_1^T \mathbf{x}_i\} \\ \text{s.t. } & \|\mathbf{x}_i\|^2 = \mathbf{x}_i^T \mathbf{x}_i = 1 \end{aligned}$$

- 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,

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

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 $\|\mathbf{x}_i\| = 1$



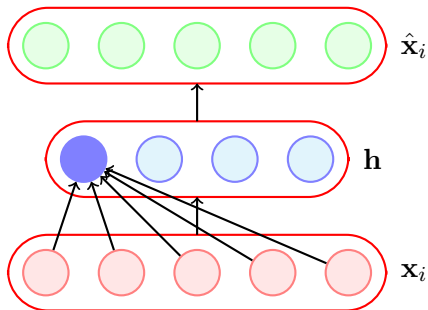
- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration \mathbf{x}_i
- For example,

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

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 $\|\mathbf{x}_i\| = 1$

$$\begin{aligned} & \max_{\mathbf{x}_i} \{W_1^T \mathbf{x}_i\} \\ & \text{s.t. } \|\mathbf{x}_i\|^2 = \mathbf{x}_i^T \mathbf{x}_i = 1 \\ & \text{Solution: } \mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}} \end{aligned}$$

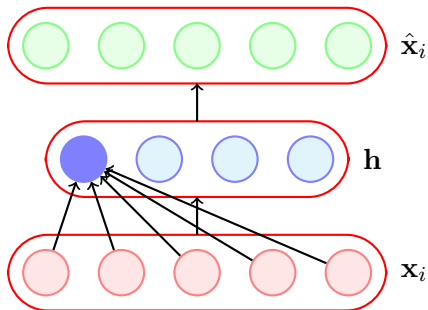


- Thus the inputs

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

will respectively cause hidden neurons 1 to n to maximally fire

$$\begin{aligned} & \max_{\mathbf{x}_i} \{W_1^T \mathbf{x}_i\} \\ & s.t. \quad \|\mathbf{x}_i\|^2 = \mathbf{x}_i^T \mathbf{x}_i = 1 \\ & \text{Solution: } \mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}} \end{aligned}$$



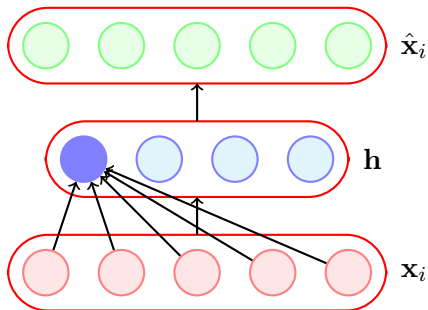
$$\begin{aligned} & \max_{\mathbf{x}_i} \{W_1^T \mathbf{x}_i\} \\ & s.t. \quad \|\mathbf{x}_i\|^2 = \mathbf{x}_i^T \mathbf{x}_i = 1 \\ & \text{Solution: } \mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}} \end{aligned}$$

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



$$\begin{aligned} & \max_{\mathbf{x}_i} \{W_1^T \mathbf{x}_i\} \\ & s.t. \quad \|\mathbf{x}_i\|^2 = \mathbf{x}_i^T \mathbf{x}_i = 1 \\ \text{Solution: } & \mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}} \end{aligned}$$

- 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 (\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 ($W_1, W_2 \dots W_k$) learned by the respective autoencoders

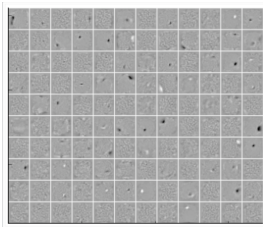


Figure: Vanilla AE
(No noise)

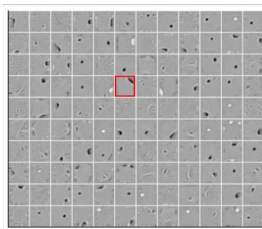


Figure: 25% Denoising
AE ($q=0.25$)

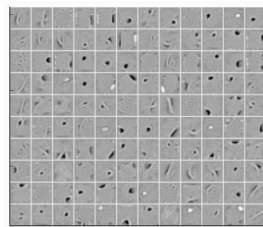


Figure: 50% Denoising
AE ($q=0.5$)

- The vanilla AE does not learn many meaningful patterns

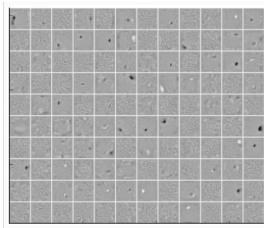


Figure: Vanilla AE
(No noise)

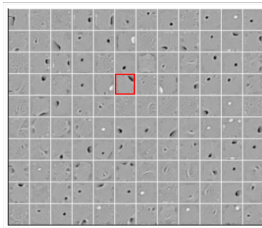


Figure: 25% Denoising
AE ($q=0.25$)

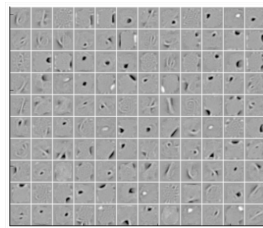


Figure: 50% Denoising
AE ($q=0.5$)

- 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')

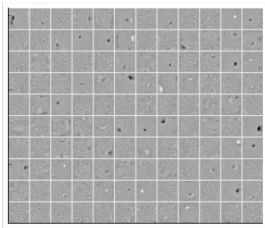


Figure: Vanilla AE
(No noise)

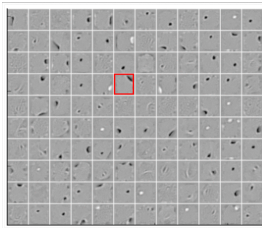


Figure: 25% Denoising
AE ($q=0.25$)

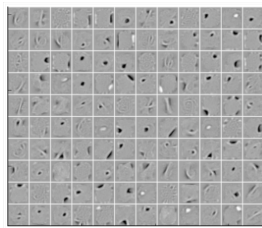
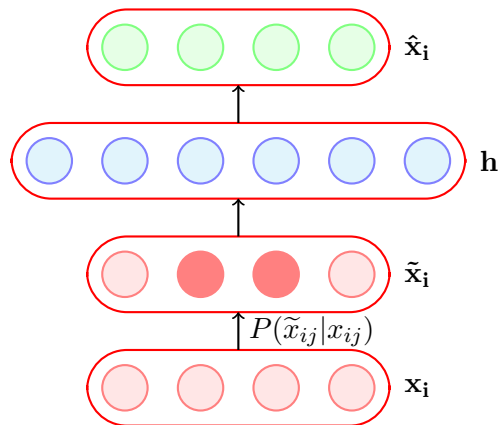
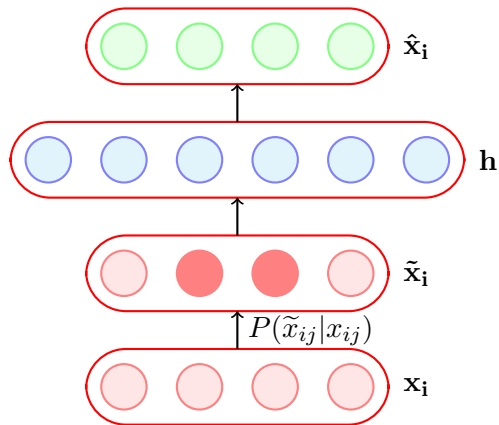


Figure: 50% Denoising
AE ($q=0.5$)

- 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

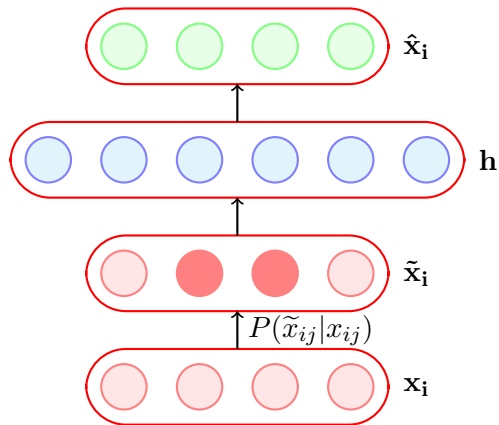


- We saw one form of $P(\tilde{x}_{ij} | x_{ij})$ which flips a fraction q of the inputs to zero



- We saw one form of $P(\tilde{x}_{ij}|x_{ij})$ 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

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



- We saw one form of $P(\tilde{x}_{ij} | x_{ij})$ 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

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

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

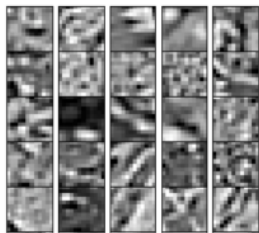


Figure: Data

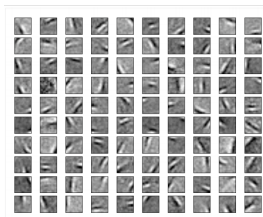


Figure: AE filters

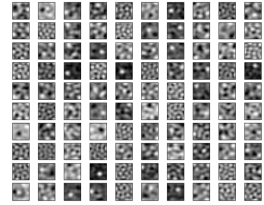


Figure: Weight decay filters

- The hidden neurons essentially behave like edge detectors

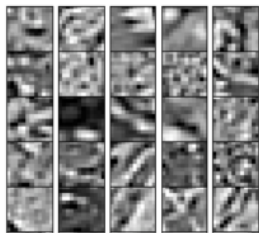


Figure: Data

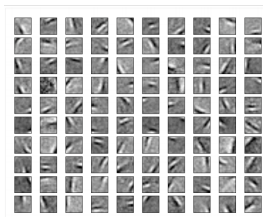


Figure: AE filters

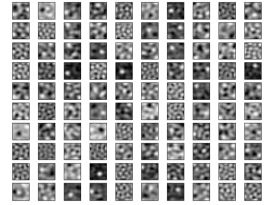
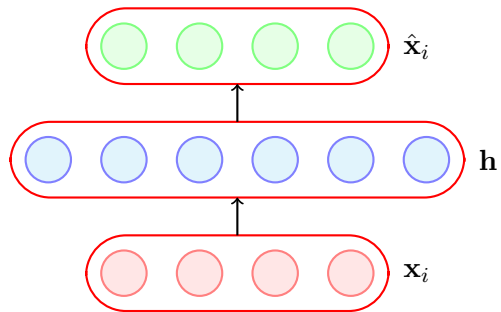
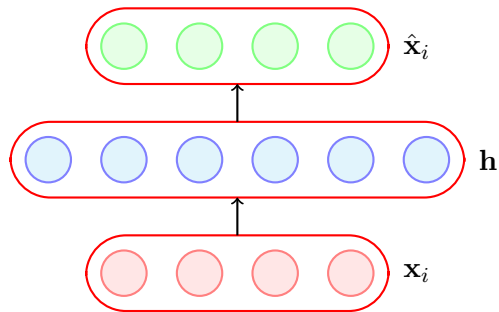


Figure: Weight decay filters

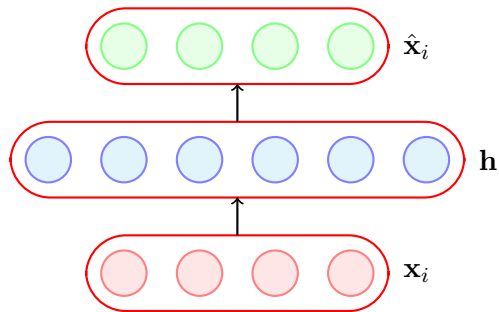
- The hidden neurons essentially behave like edge detectors
- PCA does not give such edge detectors

Module 7.5: Sparse Autoencoders

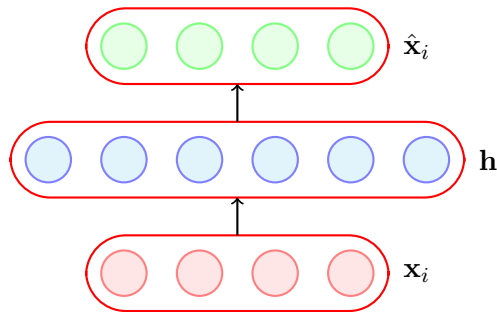




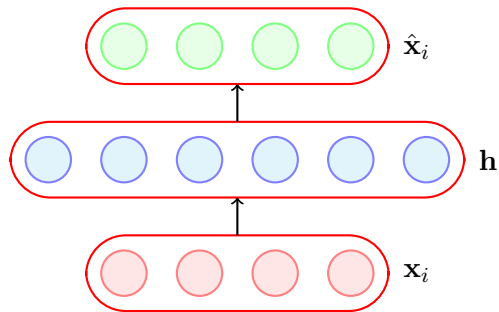
- A hidden neuron with sigmoid activation will have values between 0 and 1



- 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.



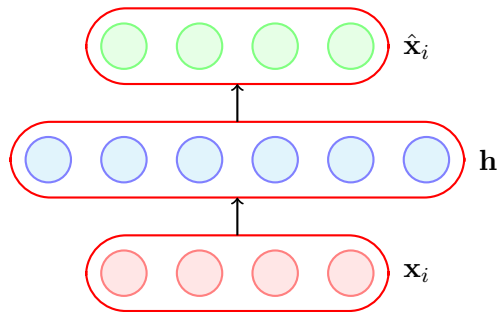
- A hidden neuron with sigmoid activation will have values between 0 and 1
- We say that the neuron is activated when its output is close to 1 and not activated when its output is close to 0.
- A sparse autoencoder tries to ensure the neuron is inactive most of the times.



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

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

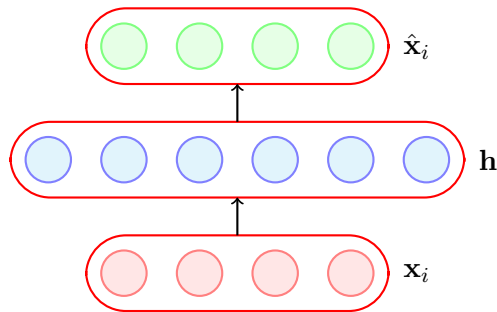
$$\hat{\rho}_l = \frac{1}{m} \sum_{i=1}^m h(\mathbf{x}_i)_l$$



- 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

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

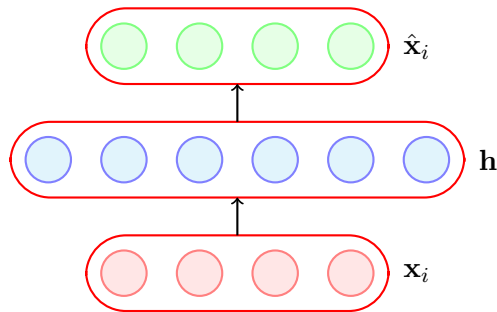


- If the neuron l is sparse (i.e. mostly inactive) then $\hat{\rho}_l \rightarrow 0$
- A sparse autoencoder uses a sparsity parameter ρ (typically very close to 0, say, 0.005) and tries to enforce the constraint $\hat{\rho}_l = \rho$
- One way of ensuring this is to add the following term to the objective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

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

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

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



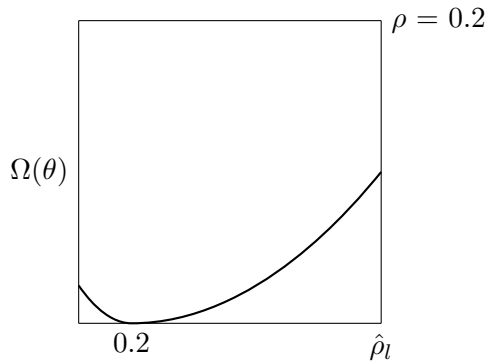
- If the neuron l is sparse (i.e. mostly inactive) then $\hat{\rho}_l \rightarrow 0$
- A sparse autoencoder uses a sparsity parameter ρ (typically very close to 0, say, 0.005) and tries to enforce the constraint $\hat{\rho}_l = \rho$
- One way of ensuring this is to add the following term to the objective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

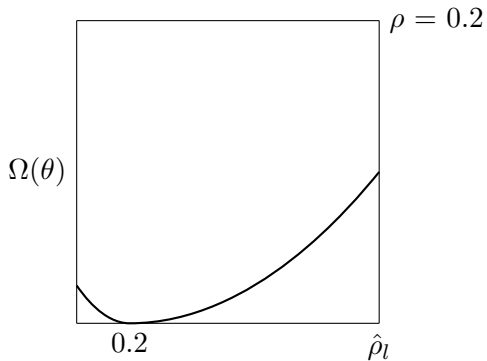
The average value of the activation of a neuron l is given by

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

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

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





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

- Now,

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

- Now,

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

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

- Now,

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

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

- Now,

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

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

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

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

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

- $\mathcal{L}(\theta)$ is the squared error loss or cross entropy loss and $\Omega(\theta)$ is the sparsity constraint.
- We already know how to calculate $\frac{\partial \mathcal{L}(\theta)}{\partial W}$
- 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:

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

• Now,

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

- $\mathcal{L}(\theta)$ is the squared error loss or cross entropy loss and $\Omega(\theta)$ is the sparsity constraint.
- We already know how to calculate $\frac{\partial \mathcal{L}(\theta)}{\partial W}$
- 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:

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

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

• Now,

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

- $\mathcal{L}(\theta)$ is the squared error loss or cross entropy loss and $\Omega(\theta)$ is the sparsity constraint.
- We already know how to calculate $\frac{\partial \mathcal{L}(\theta)}{\partial W}$
- 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:

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

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

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

- Now,

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

- $\mathcal{L}(\theta)$ is the squared error loss or cross entropy loss and $\Omega(\theta)$ is the sparsity constraint.
- We already know how to calculate $\frac{\partial \mathcal{L}(\theta)}{\partial W}$
- 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:

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

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

For each neuron $l \in 1 \dots k$ 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,

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

- $\mathcal{L}(\theta)$ is the squared error loss or cross entropy loss and $\Omega(\theta)$ is the sparsity constraint.
- We already know how to calculate $\frac{\partial \mathcal{L}(\theta)}{\partial W}$
- 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:

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

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

For each neuron $l \in 1 \dots k$ 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 $\frac{\partial \hat{\rho}_l}{\partial W} = \mathbf{x}_i (g'(W^T \mathbf{x}_i + \mathbf{b}))^T$ (see next slide)

- Now,

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

- $\mathcal{L}(\theta)$ is the squared error loss or cross entropy loss and $\Omega(\theta)$ is the sparsity constraint.
- We already know how to calculate $\frac{\partial \mathcal{L}(\theta)}{\partial W}$
- 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:

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

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

For each neuron $l \in 1 \dots k$ 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 $\frac{\partial \hat{\rho}_l}{\partial W} = \mathbf{x}_i (g'(W^T \mathbf{x}_i + \mathbf{b}))^T$ (see next slide)

- Now,

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

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

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

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

Derivation

$$\frac{\partial \hat{\rho}}{\partial W} = \begin{bmatrix} \frac{\partial \hat{\rho}_1}{\partial W} & \frac{\partial \hat{\rho}_2}{\partial W} & \dots & \frac{\partial \hat{\rho}_k}{\partial W} \end{bmatrix}$$

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} :-

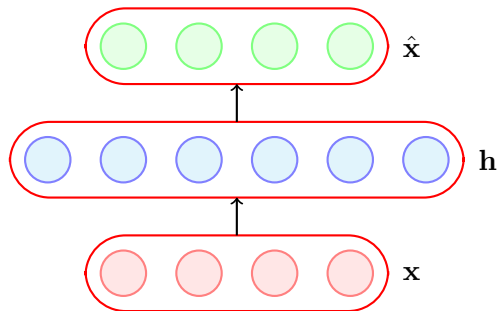
$$\begin{aligned} \frac{\partial \hat{\rho}_l}{\partial W_{jl}} &= \frac{\partial \left[\frac{1}{m} \sum_{i=1}^m g(W_{:,l}^T \mathbf{x}_i + b_l) \right]}{\partial W_{jl}} \\ &= \frac{1}{m} \sum_{i=1}^m \frac{\partial \left[g(W_{:,l}^T \mathbf{x}_i + b_l) \right]}{\partial W_{jl}} \\ &= \frac{1}{m} \sum_{i=1}^m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

So in matrix notation we can write it as :

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

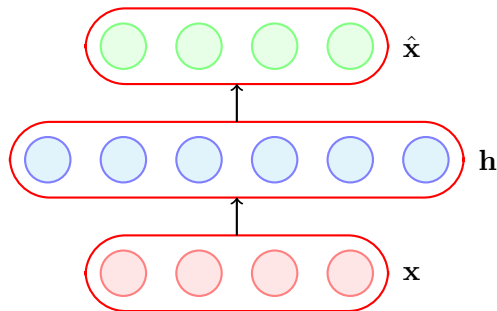
Module 7.6: Contractive Autoencoders

- 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

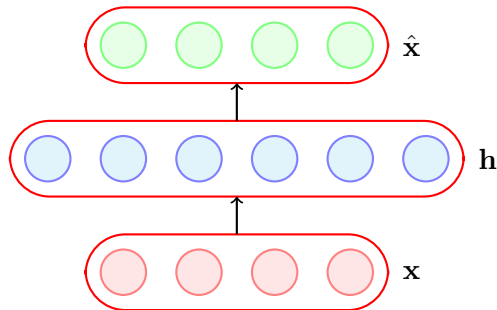
$$\Omega(\theta) = \|J_{\mathbf{x}}(\mathbf{h})\|_F^2$$



- 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 $J_{\mathbf{x}}(\mathbf{h})$ is the Jacobian of the encoder.

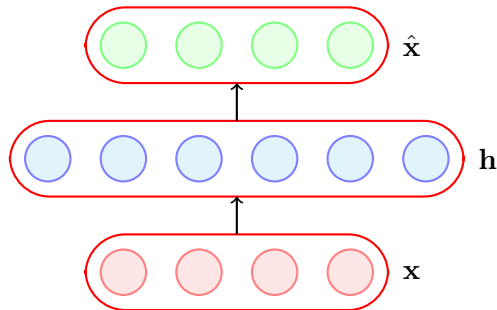


- A contractive autoencoder also tries to prevent an overcomplete autoencoder from learning the identity function.
- It does so by adding the following regularization term to the loss function

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

where $J_{\mathbf{x}}(\mathbf{h})$ 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

- 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 (j, l) entry of the Jacobian captures the variation in the output of the l^{th} neuron with a small variation in the j^{th} input.

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

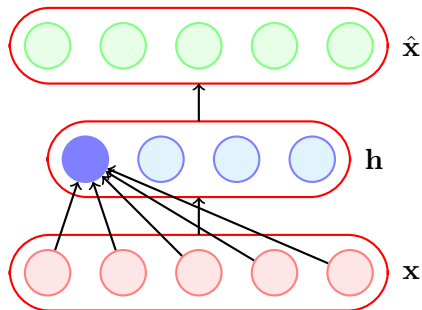
- If the input has n dimensions and the hidden layer has k dimensions then
- In other words, the (j, l) entry of the Jacobian captures the variation in the output of the l^{th} neuron with a small variation in the j^{th} input.

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

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

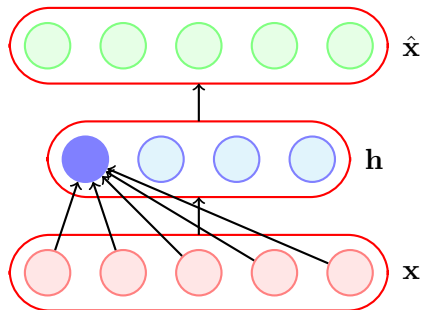
- What is the intuition behind this ?

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



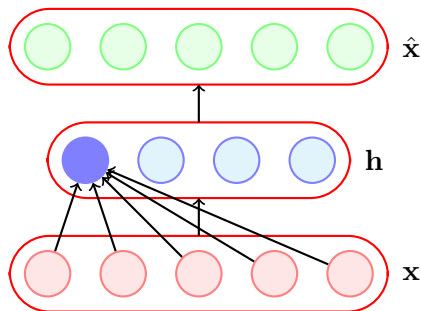
- What is the intuition behind this ?
- Consider $\frac{\partial h_1}{\partial x_1}$, what does it mean if $\frac{\partial h_1}{\partial x_1} = 0$

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



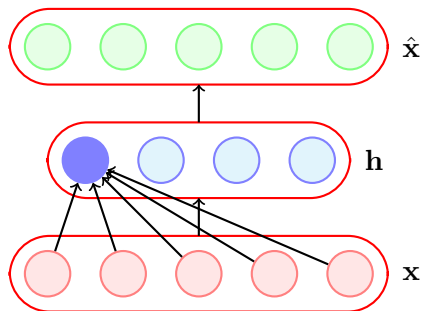
- What is the intuition behind this ?
- Consider $\frac{\partial h_1}{\partial x_1}$, what does it mean if $\frac{\partial h_1}{\partial x_1} = 0$
- It means that this neuron is not very sensitive to variations in the input x_1 .

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



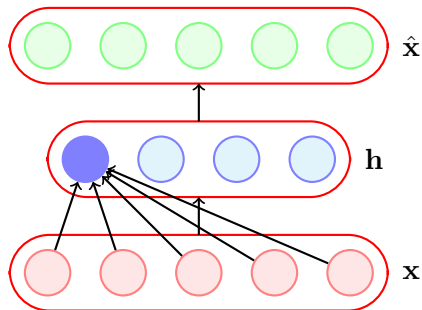
- What is the intuition behind this ?
- Consider $\frac{\partial h_1}{\partial x_1}$, what does it mean if $\frac{\partial h_1}{\partial x_1} = 0$
- It means that this neuron is not very sensitive to variations in the input x_1 .
- But doesn't this contradict our other goal of minimizing $\mathcal{L}(\theta)$ which requires \mathbf{h} to capture variations in the input.

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



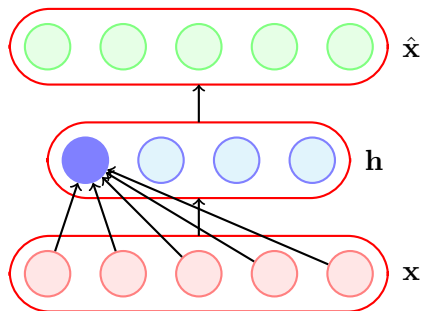
- Indeed it does and that's the idea

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



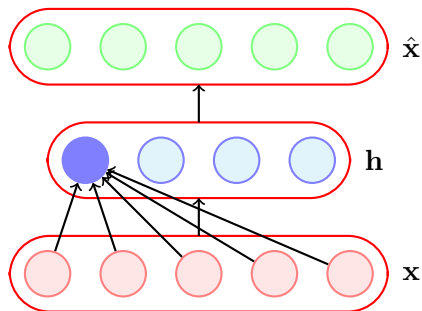
- Indeed it does and that's the idea
- By putting these two contradicting objectives against each other we ensure that \mathbf{h} is sensitive to only very important variations as observed in the training data.

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



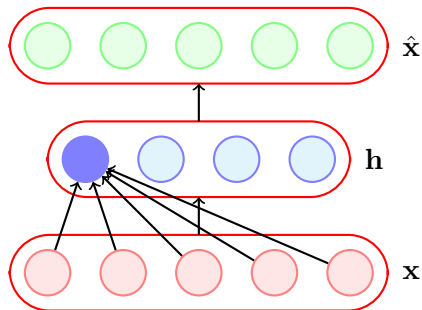
- Indeed it does and that's the idea
- By putting these two contradicting objectives against each other we ensure that \mathbf{h} is sensitive to only very important variations as observed in the training data.
- $\mathcal{L}(\theta)$ - capture important variations in data

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



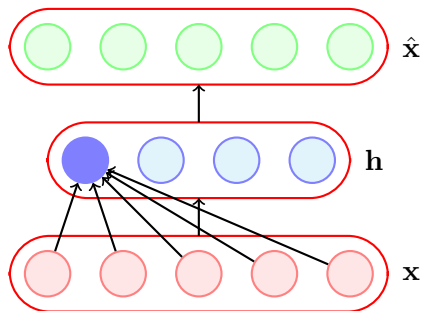
- Indeed it does and that's the idea
- By putting these two contradicting objectives against each other we ensure that \mathbf{h} is sensitive to only very important variations as observed in the training data.
- $\mathcal{L}(\theta)$ - capture important variations in data
- $\Omega(\theta)$ - do not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

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

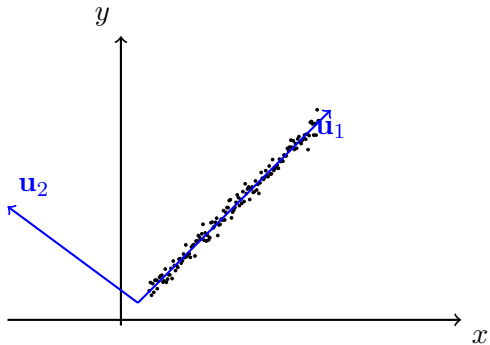


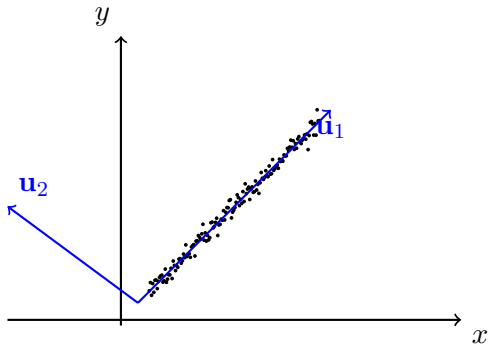
- Indeed it does and that's the idea
- By putting these two contradicting objectives against each other we ensure that \mathbf{h} is sensitive to only very important variations as observed in the training data.
- $\mathcal{L}(\theta)$ - capture important variations in data
- $\Omega(\theta)$ - do not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

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

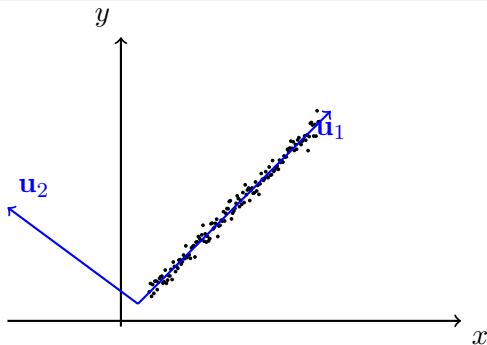


Let us try to understand this with the help of an illustration.

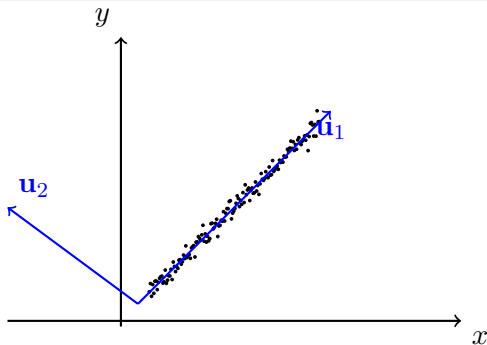




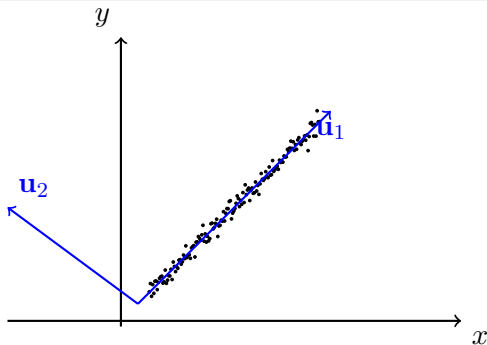
- Consider the variations in the data along directions \mathbf{u}_1 and \mathbf{u}_2



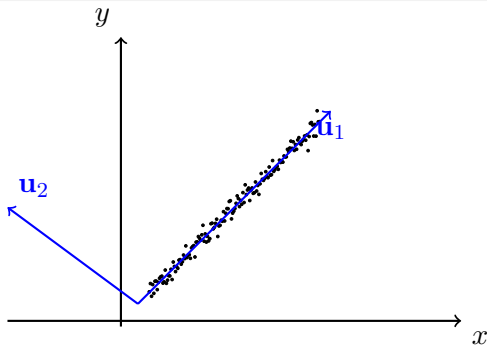
- Consider the variations in the data along directions \mathbf{u}_1 and \mathbf{u}_2
- It makes sense to maximize a neuron to be sensitive to variations along \mathbf{u}_1



- 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)

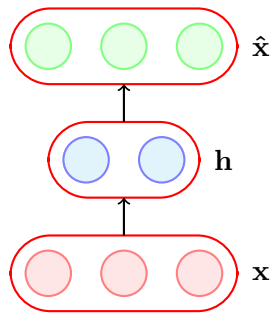


- Consider the variations in the data along directions \mathbf{u}_1 and \mathbf{u}_2
- It makes sense to maximize a neuron to be sensitive to variations along \mathbf{u}_1
- At the same time it makes sense to inhibit a neuron from being sensitive to variations along \mathbf{u}_2 (as there seems to be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

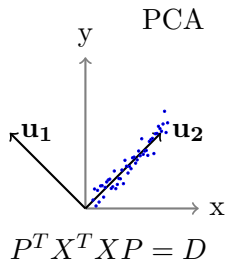


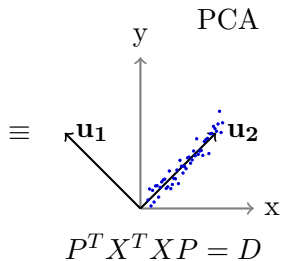
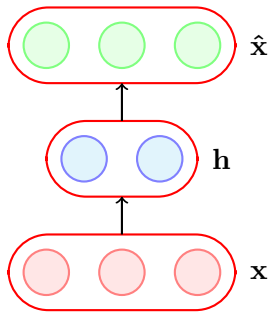
- Consider the variations in the data along directions \mathbf{u}_1 and \mathbf{u}_2
- It makes sense to maximize a neuron to be sensitive to variations along \mathbf{u}_1
- At the same time it makes sense to inhibit a neuron from being sensitive to variations along \mathbf{u}_2 (as there seems to be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

Module 7.7 : Summary

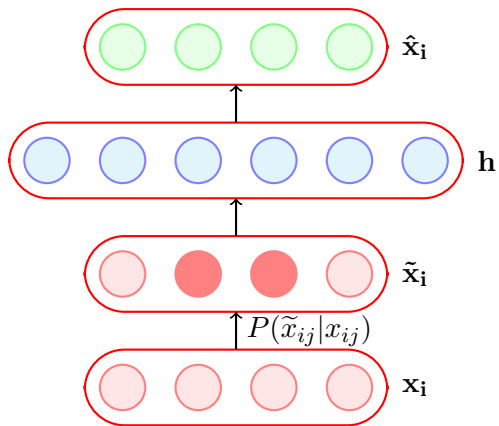


\equiv

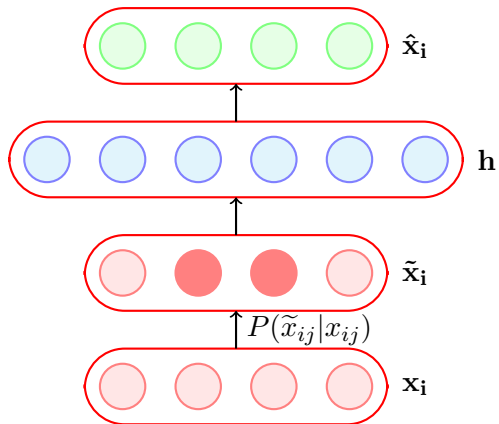


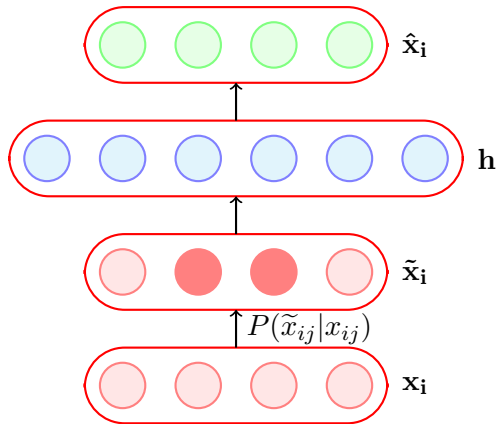


$$\min_{\theta} \|X - \underbrace{HW^*}_{\substack{U\Sigma V^T \\ \text{(SVD)}}}\|_F^2$$



Regularization

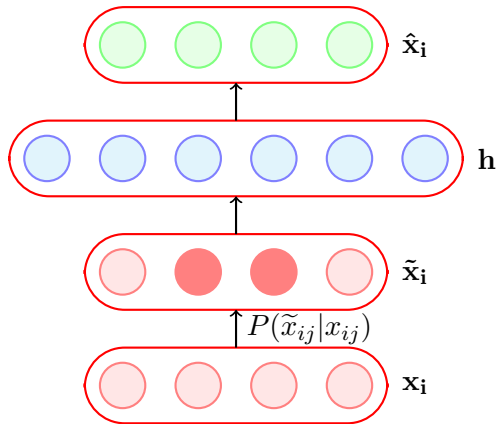




Regularization

$$\Omega(\theta) = \lambda \|\theta\|^2$$

Weight decaying



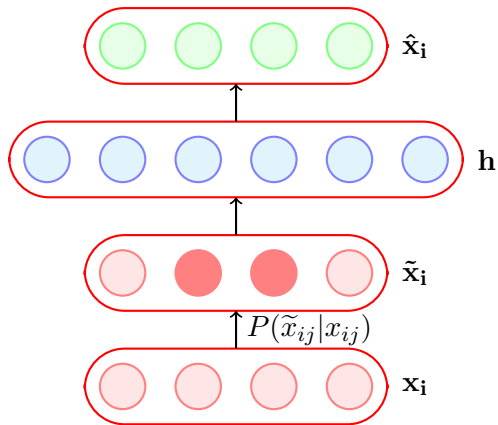
Regularization

$$\Omega(\theta) = \lambda \|\theta\|^2$$

Weight decaying

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

Sparse



Regularization

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

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

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