Introduction to Machine Learning in Geosciences GEO371T/GEO398D.1

Autoencoders

Mrinal K. Sen
Geosciences
UT Austin

November 28, 2023

Content

- Autoencoders
  - PCA
  - Regularization
  - Denoising Autoencoders
  - Contractive Autoencoders
- Transposed Convolution
- Variational Autoencoders

Resources: `https://www.machinecurve.com/index.php/2019/09/29/understanding-transposed-convolutions/`

Autoencoders

- Unsupervised
- Map high-dimensional data to low dimensions
- Learn abstract features in an unsupervised way

Autoencders

- An autoencoder is a specific type of a neural network, which is mainly designed to encode the input into a compressed and meaningful representation.
- Decode it back such that the reconstructed input is similar as possible to the original one.

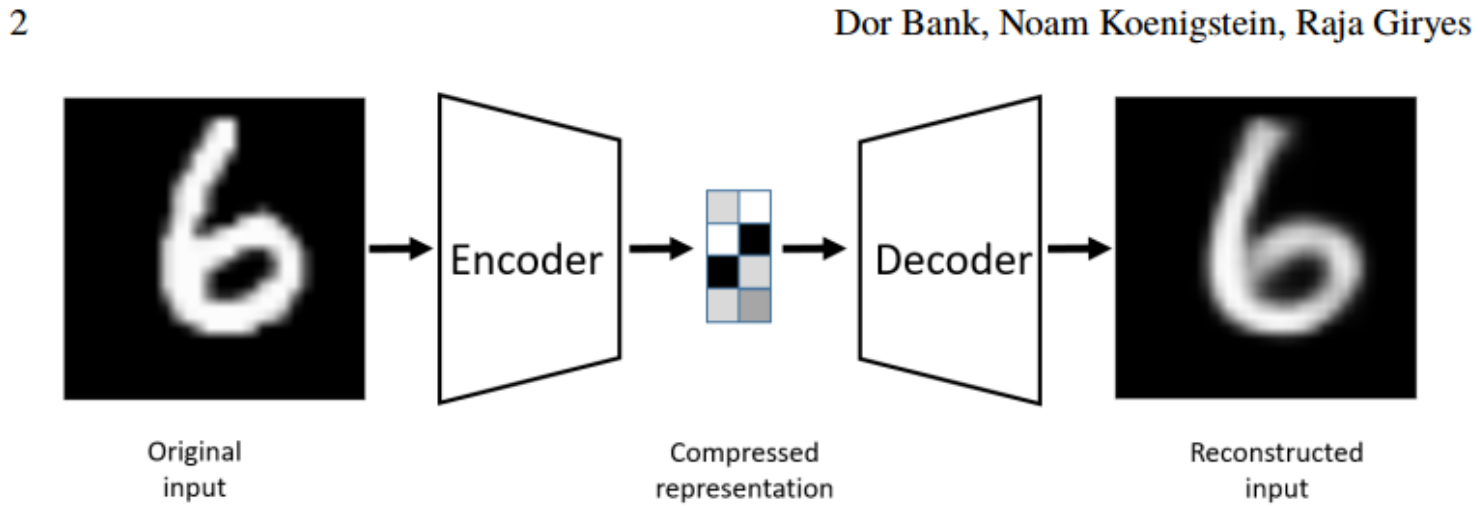
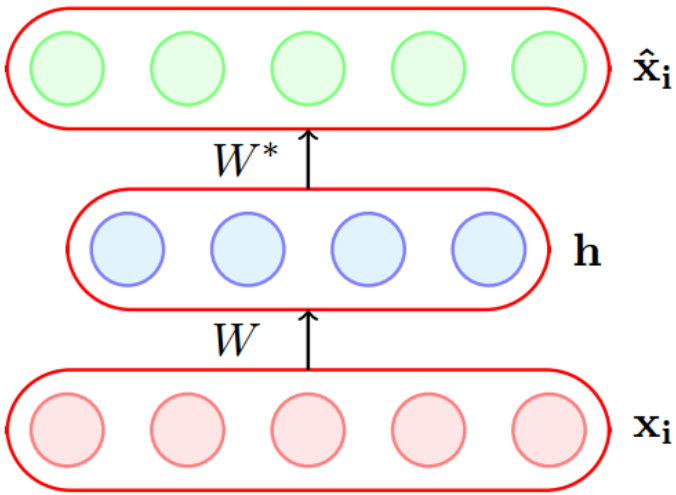


Fig. 1: An autoencoder example. The input image is encoded to a compressed representation and then decoded.

Rumelhart, D.E.,

Hinton, G.E., Williams, R.J.: Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chap. Learning Internal Representations by Error Propagation, pp. 318–362. MIT Press, Cambridge, MA, USA (1986).

Autoencders

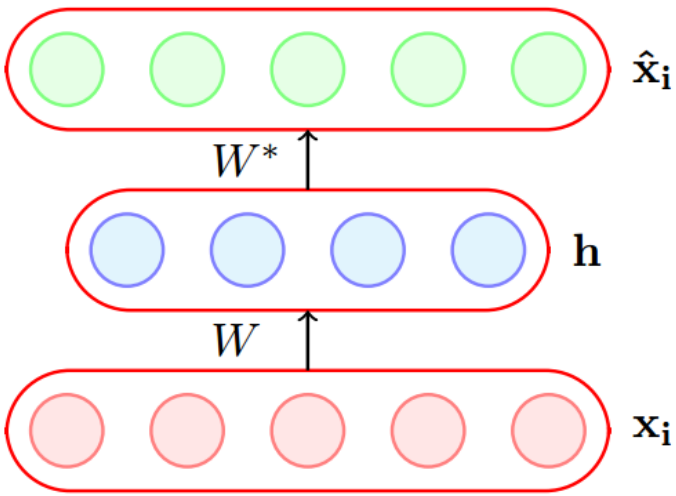


$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- 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 $\hat{\mathbf{x}}_\mathbf{i}$ is close to $\mathbf{x_i}$ (we will see some such loss functions soon)

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders



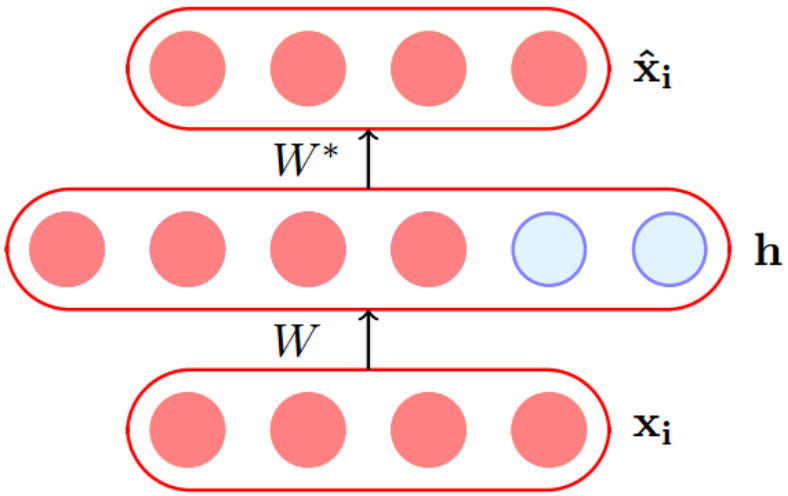
$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Let us consider the case where $\dim(\mathbf{h}) < \dim(\mathbf{x_i})$
- If we are still able to reconstruct $\hat{\mathbf{x}}_\mathbf{i}$ 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?

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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders



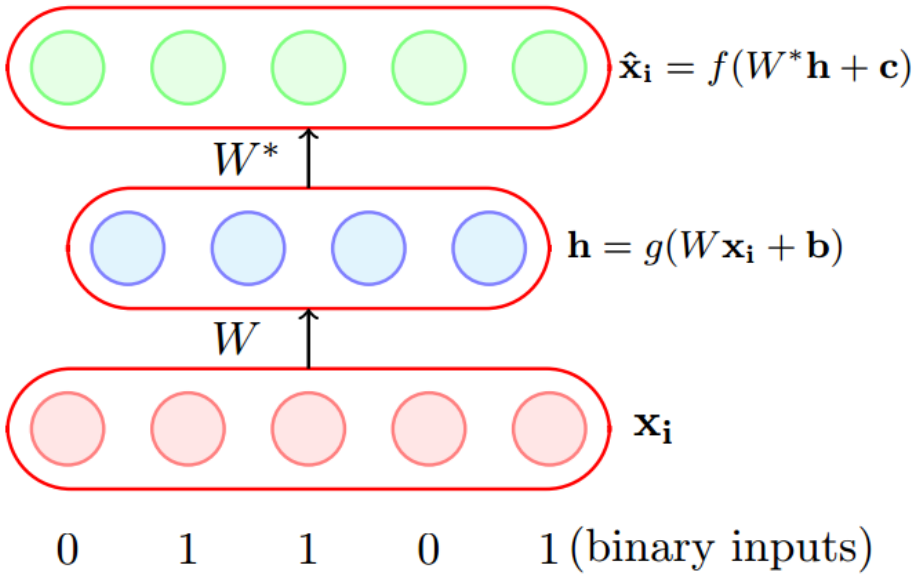
$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- 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 $\hat{\mathbf{x}}_\mathbf{i}$
- Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data

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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: Loss 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?

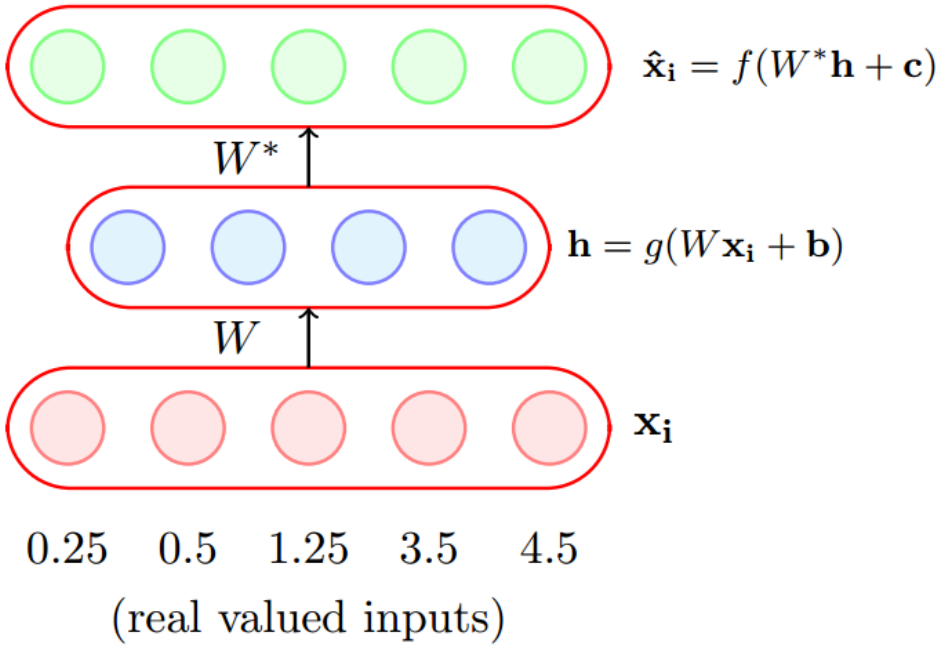
$$\hat{\mathbf{x}}_\mathbf{i} = \tanh(W^*\mathbf{h} + \mathbf{c})$$
$$\hat{\mathbf{x}}_\mathbf{i} = W^*\mathbf{h} + \mathbf{c}$$
$$\hat{\mathbf{x}}_\mathbf{i} = logistic(W^*\mathbf{h} + \mathbf{c})$$

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

g is typically chosen as the sigmoid function

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: Loss function



Again, g is typically chosen as the sigmoid function

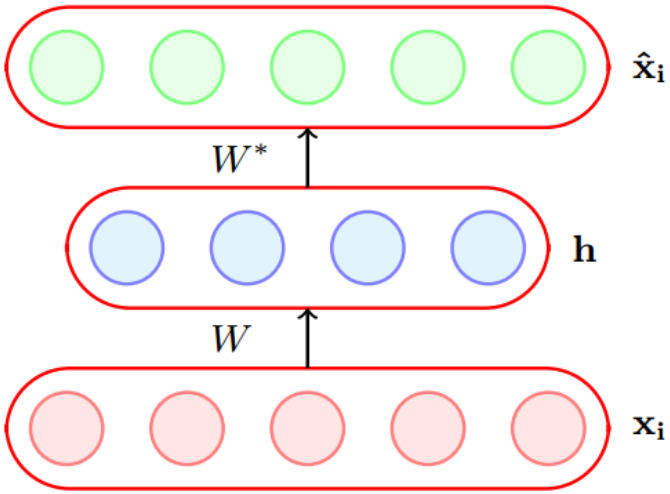
- Suppose all our inputs are real (each $x_{ij} \in \mathbb{R}$)
- Which of the following functions would be most apt for the decoder?

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

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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: Loss function



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

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

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

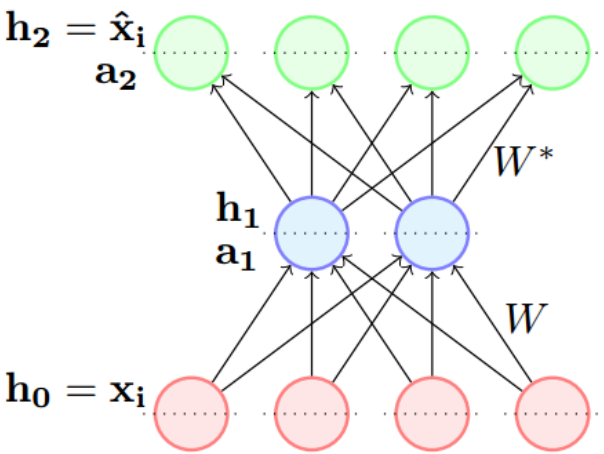
$$i.e., \min_{W,W^*,\mathbf{c},\mathbf{b}} \frac{1}{m}\sum_{i=1}^{m}(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x_i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x_i})$$

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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: Loss function

$$\mathscr{L}(\theta) = (\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})$$



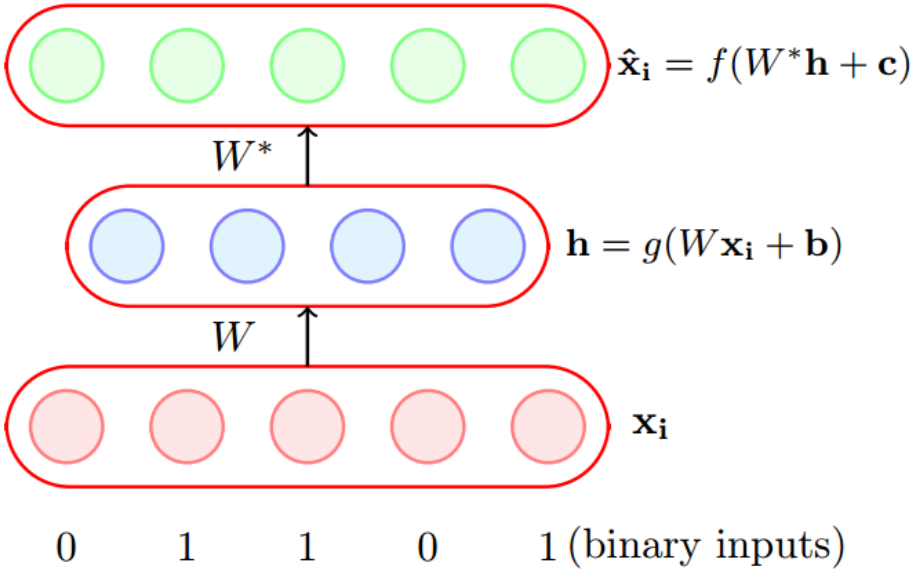
- Note that the loss function is shown for only one training example.

- $$\frac{\partial \mathscr{L}(\theta)}{\partial W^*} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial W^*}}$$
- $$\frac{\partial \mathscr{L}(\theta)}{\partial W} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial \mathbf{h_1}} \frac{\partial \mathbf{h_1}}{\partial \mathbf{a_1}} \frac{\partial \mathbf{a_1}}{\partial W}}$$
- We have already seen how to calculate the expression in the boxes when we learnt backpropagation

$$\begin{aligned}\frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} &= \frac{\partial \mathscr{L}(\theta)}{\partial \hat{\mathbf{x}}_\mathbf{i}} \\ &= \nabla_{\hat{\mathbf{x}}_\mathbf{i}}\{(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})\} \\ &= 2(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})\end{aligned}$$

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders



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 $\hat{x}_{ij} = x_{ij}$!

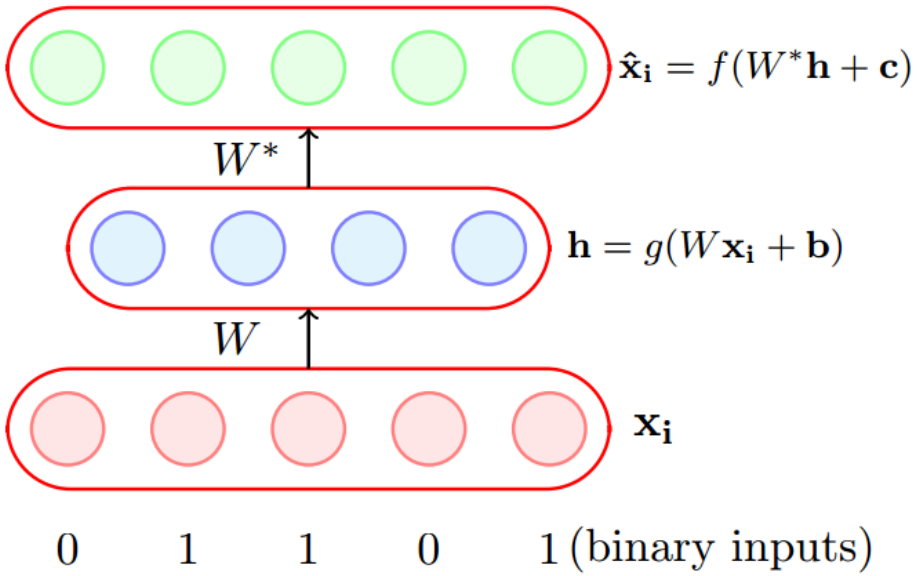
- Consider the case when the inputs are binary
- We use a sigmoid decoder which will produce outputs between 0 and 1, and can be interpreted as probabilities.
- For a single n-dimensional i^{th} input we can use the following loss function

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

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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders



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

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

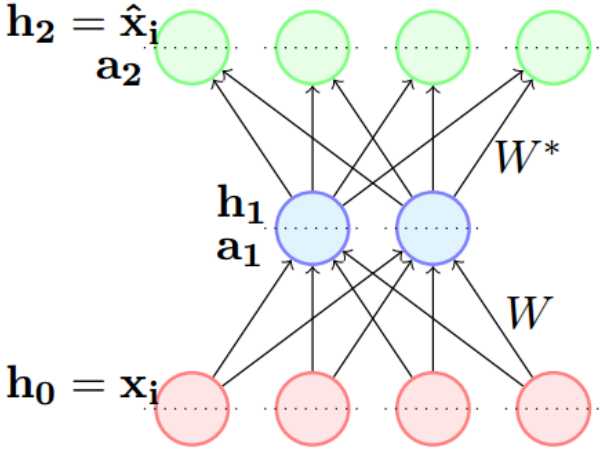
- Logistic as it naturally restricts all outputs to be between 0 and 1

g is typically chosen as the sigmoid function

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders

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



$$\frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} = \begin{pmatrix} \frac{\partial \mathscr{L}(\theta)}{\partial h_{21}} \\ \frac{\partial \mathscr{L}(\theta)}{\partial h_{22}} \\ \vdots \\ \frac{\partial \mathscr{L}(\theta)}{\partial h_{2n}} \end{pmatrix}$$

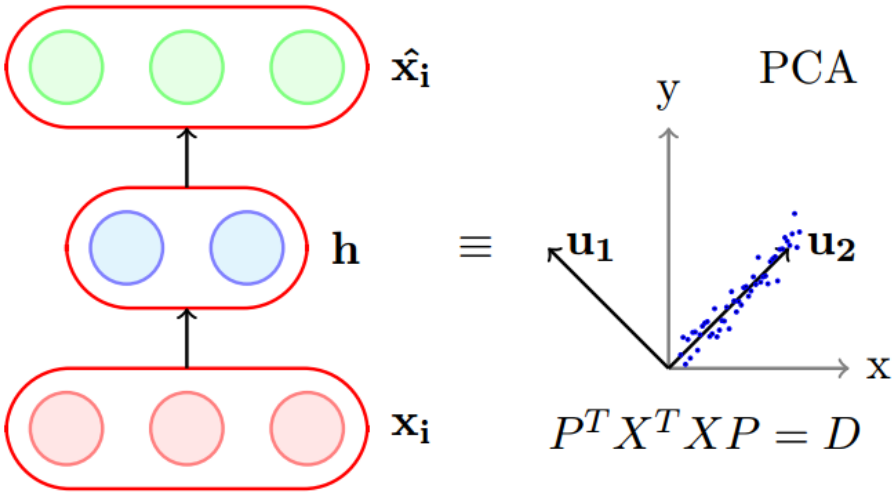
- $\frac{\partial \mathscr{L}(\theta)}{\partial W^*} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \boxed{\frac{\partial \mathbf{a_2}}{\partial W^*}}$
- $\frac{\partial \mathscr{L}(\theta)}{\partial W} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \boxed{\frac{\partial \mathbf{a_2}}{\partial \mathbf{h_1}} \frac{\partial \mathbf{h_1}}{\partial \mathbf{a_1}} \frac{\partial \mathbf{a_1}}{\partial W}}$
- 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:

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

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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: PCA

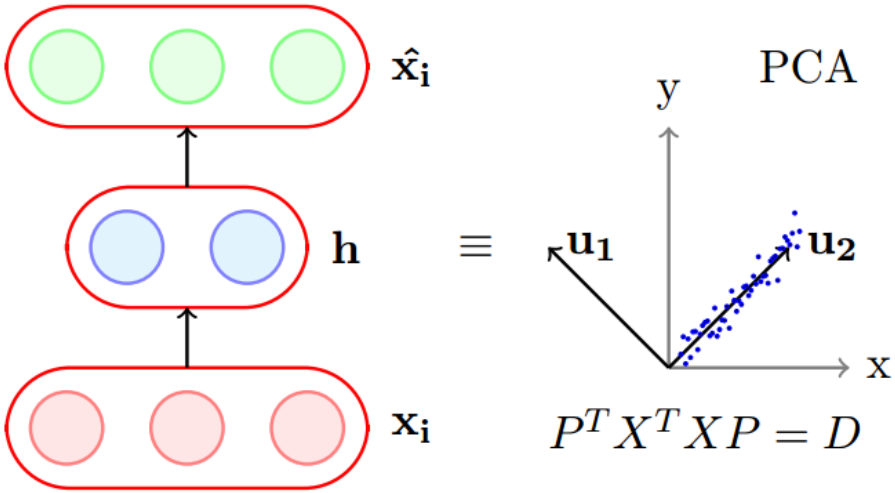


- We will now see that the encoder part of an autoencoder is equivalent to PCA if we
  - use a linear encoder
  - use a linear decoder
  - use squared error loss function
  - normalize the inputs to

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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: PCA



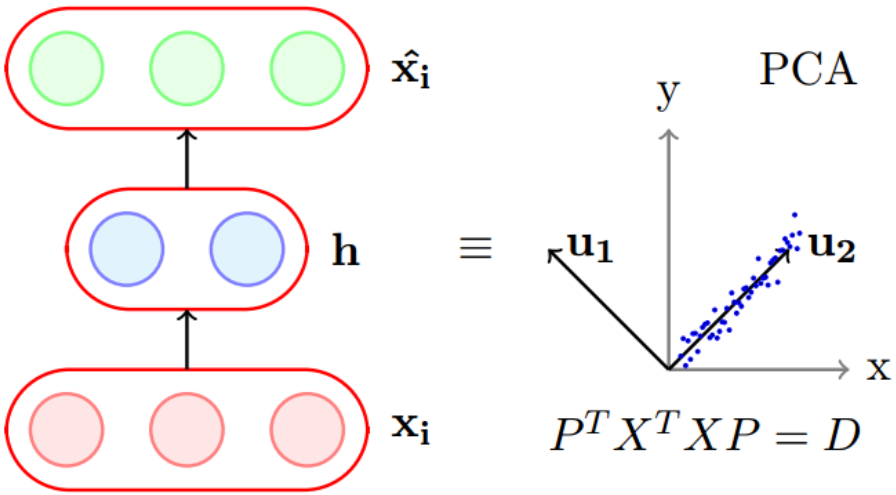
- First let us consider the implication of normalizing the inputs to

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

- 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)

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: PCA



- 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.

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: PCA

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

- 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

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

- By matching variables one possible solution is

$$H = U_{.,\leq k} \Sigma_{k,k}$$
$$W^* = V_{.,\leq k}^T$$

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: PCA

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

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

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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: PCA

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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: PCA

Remember

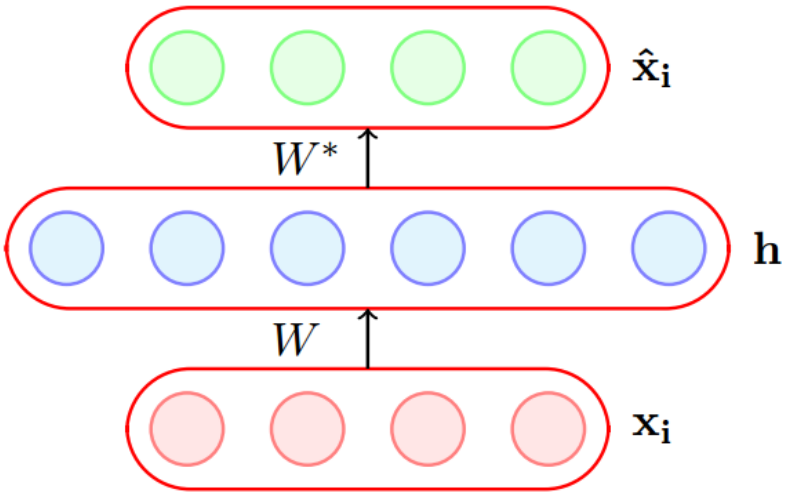
The encoder of a linear autoencoder is equivalent to PCA if we

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

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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

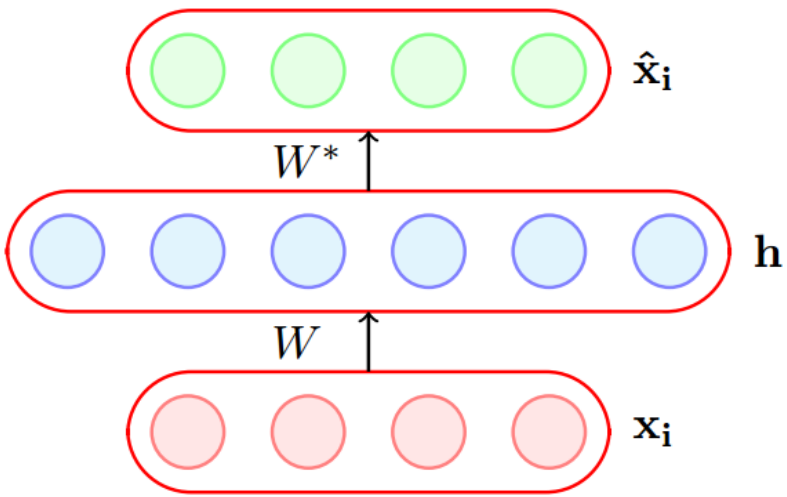
Autoencders: Regularization



- 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 $\hat{\mathbf{x}}_\mathbf{i}$
- To avoid poor generalization, we need to introduce regularization

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: Regularization



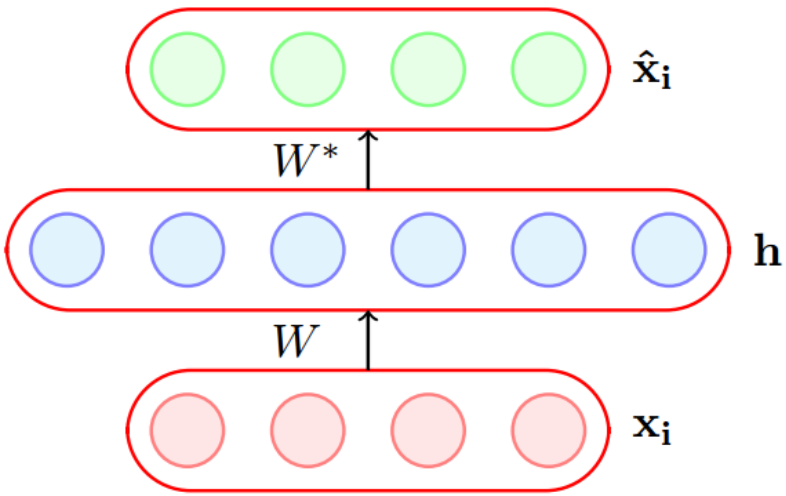
- The simplest solution is to add a L_2-regularization term to the objective function

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

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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: Regularization



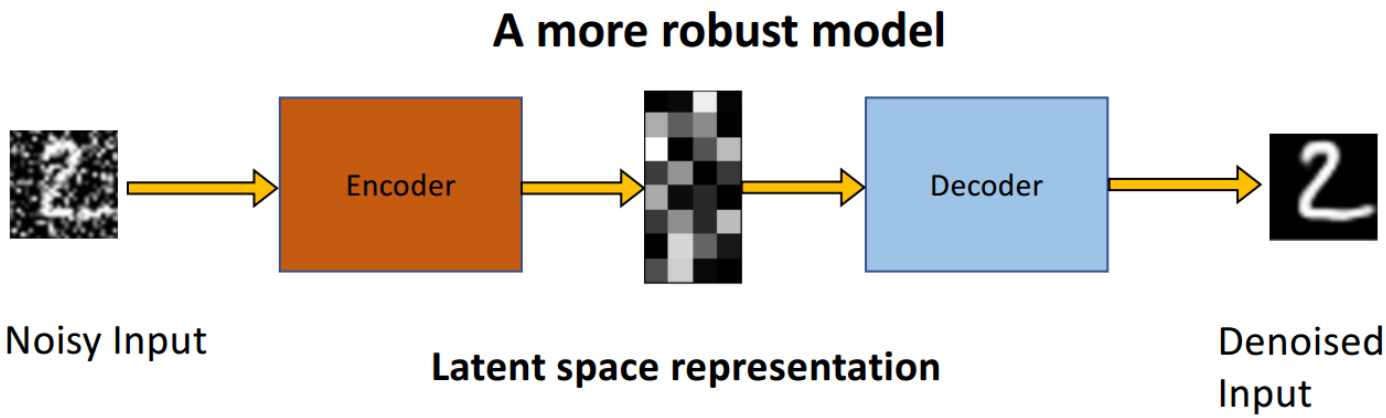
- Another trick is to tie the weights of the encoder and decoder i.e., $W^* = W^T$
- This effectively reduces the capacity of Autoencoder and acts as a regularizer

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

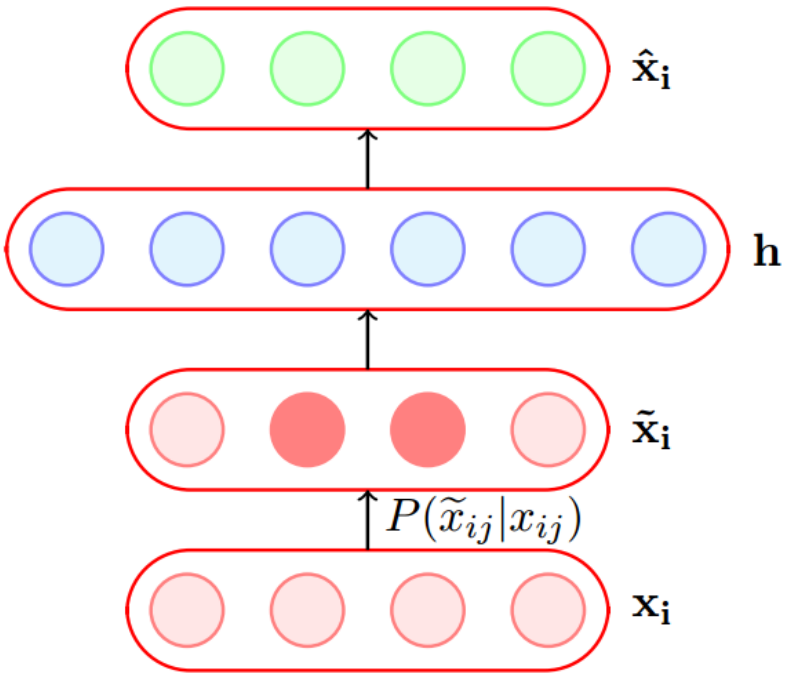
Denoising Autoencoders

Intuition:

- We still aim to encode the input and to NOT mimic the identity function.
- We try to undo the effect of *corruption* process stochastically applied to the input.



Autoencders: Denoising



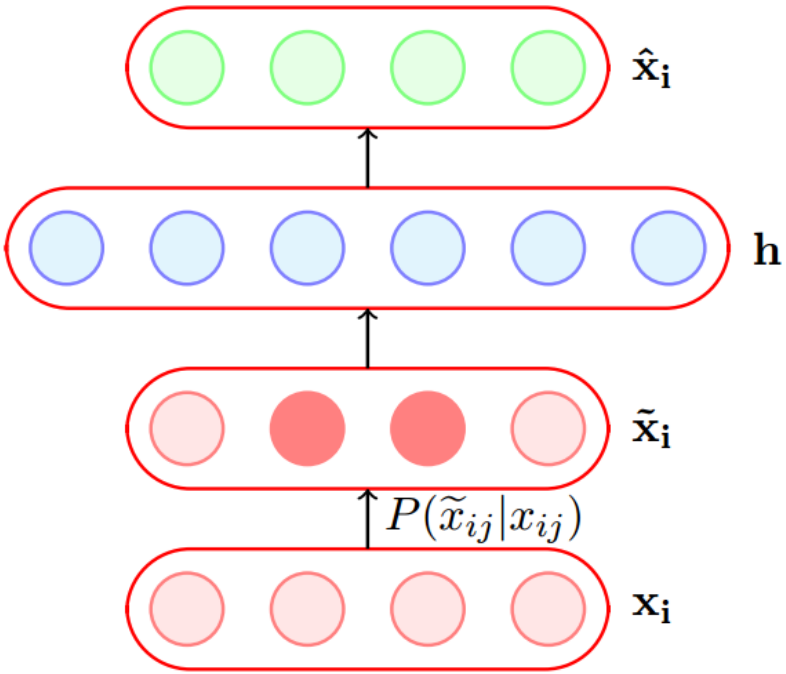
- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\widetilde{x}_{ij}|x_{ij})$) before feeding it to the network
- A simple $P(\widetilde{x}_{ij}|x_{ij})$ used in practice is the following

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

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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: Denoising



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

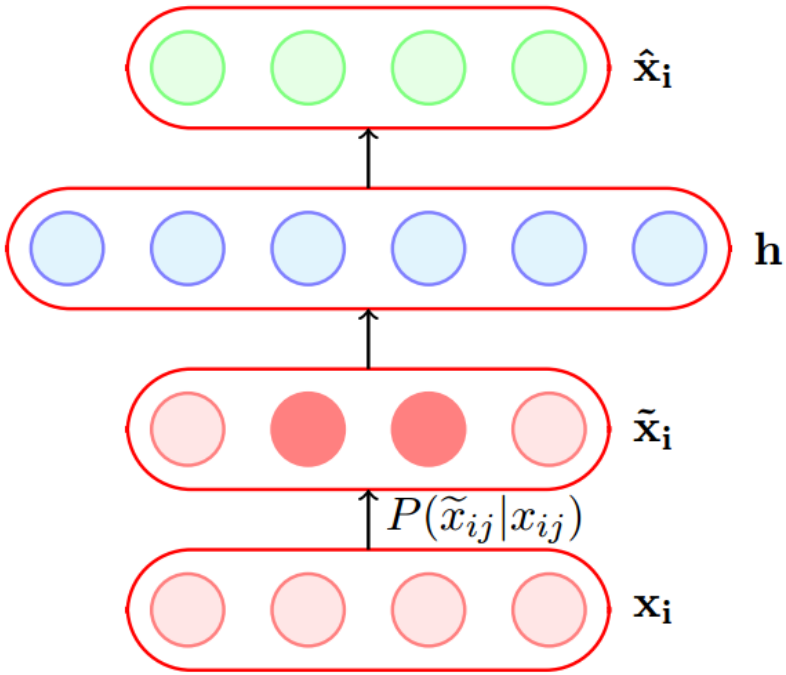
- How does this help ?
- This helps because the objective is still to reconstruct the original (uncorrupted) $\mathbf{x}_i$

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

- It no longer makes sense for the model to copy the corrupted $\tilde{\mathbf{x}}_i$ 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.

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: Denoising



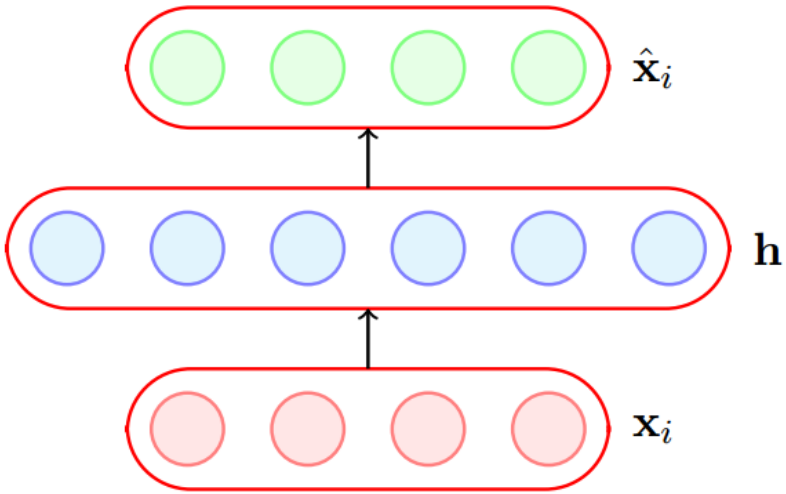
- We saw one form of $P(\widetilde{x}_{ij}|x_{ij})$ which flips a fraction q of the inputs to zero
- Another way of corrupting the inputs is to add a Gaussian noise to the input

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

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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

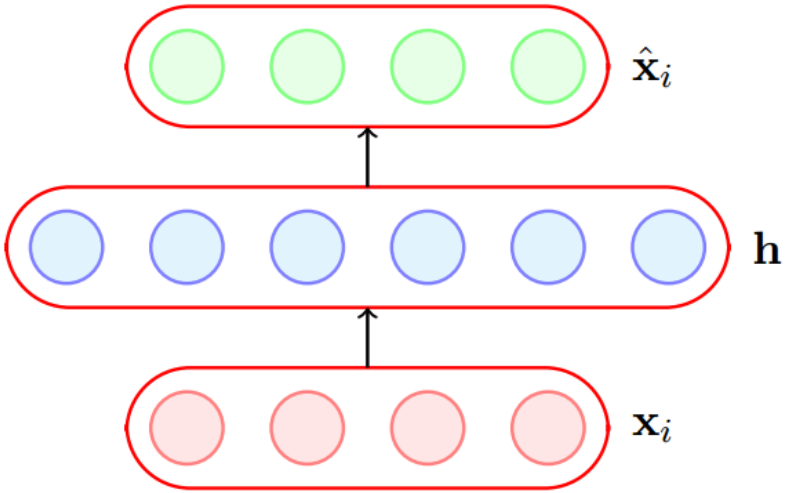
Autoencders: Denoising



- 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.

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Autoencders: Denoising



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

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

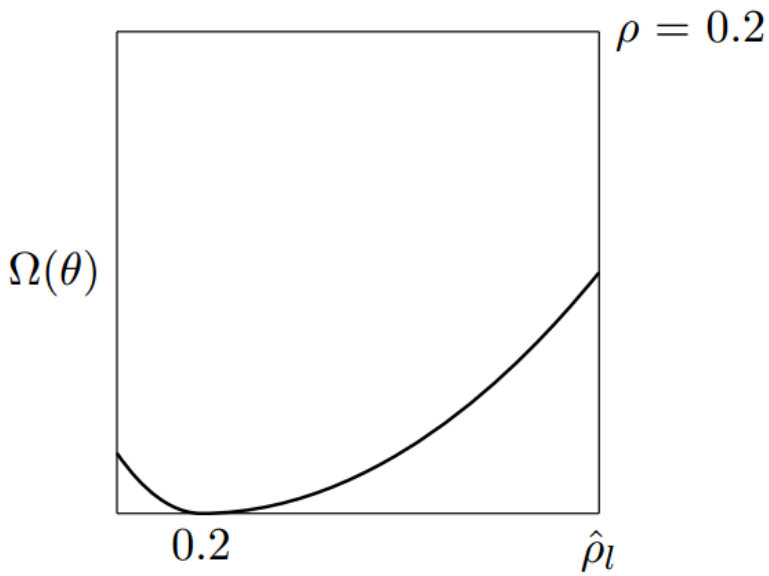
- If the neuron l is sparse (i.e. mostly inactive) then $\hat{\rho}_l \rightarrow 0$
- A sparse autoencoder uses a sparsity parameter ρ (typically very close to 0, say, 0.005) and tries to enforce the constraint $\hat{\rho}_l = \rho$
- One way of ensuring this is to add the following term to the objective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

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

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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

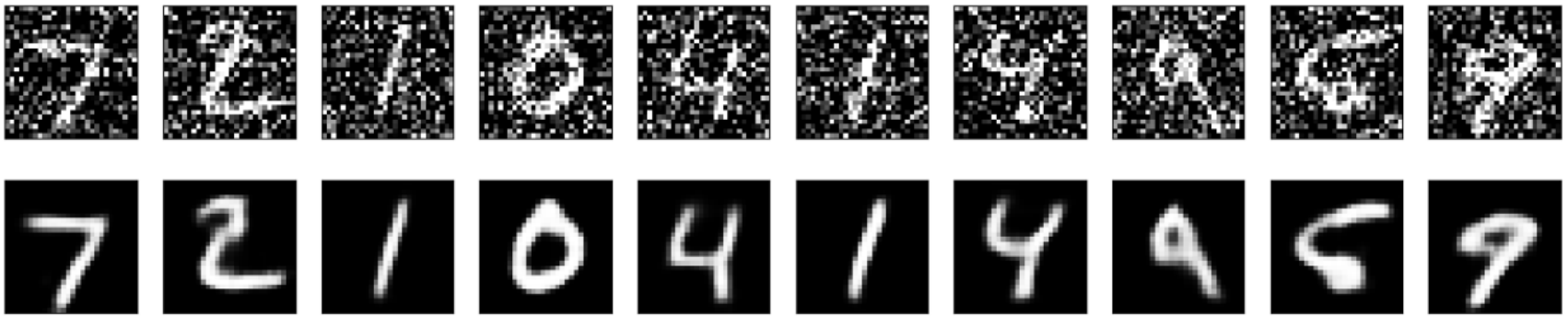
Autoencders: Denoising



https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

Denoising convolutional AE – keras

- 50 epochs.
- Noise factor 0.5
- 92% accuracy on validation set.



Contractive Autoencders

- In denoising autoencoders, the emphasis is on letting the encoder be resistant to some perturbations of the input.
- In Contractive autoencoders, the emphasis is on making the feature extraction less sensitive to small perturbations, by forcing the encoder to disregard changes in the input that are not important for the reconstruction by the decoder.
- A penalty is imposed on the Jacobian of the network.

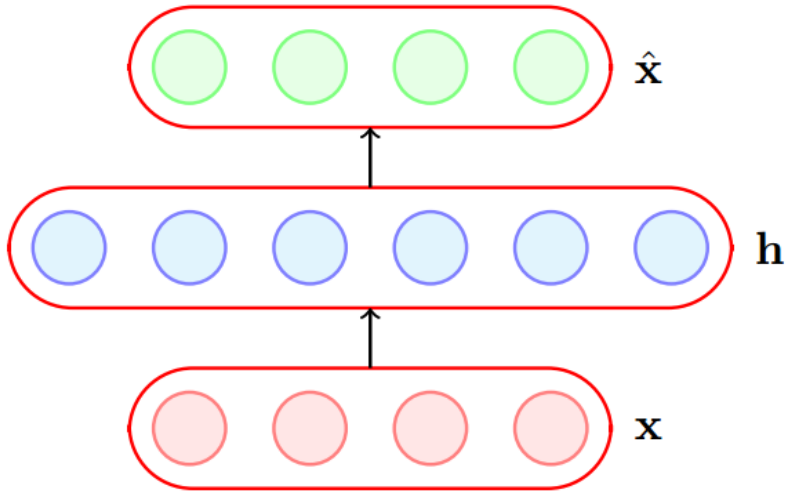
Contractive Autoencders

- 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

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

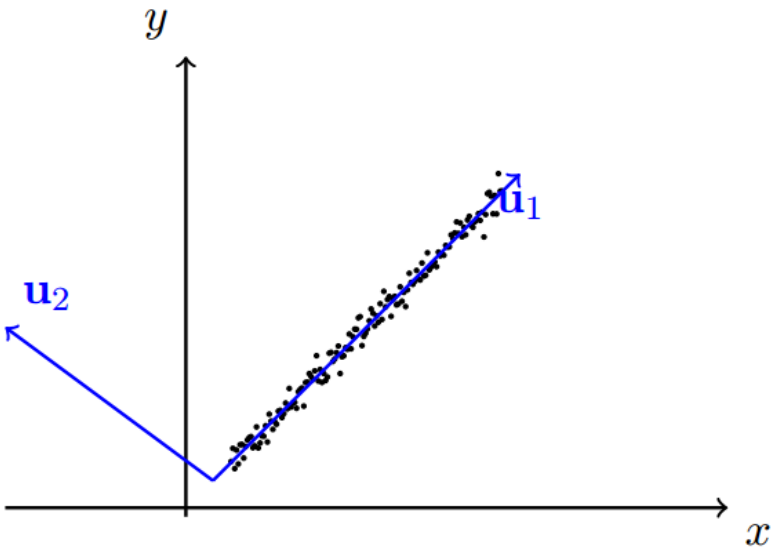
where $J_{\mathbf{x}}(\mathbf{h})$ is the Jacobian of the encoder.

- Let us see what it looks like.



https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

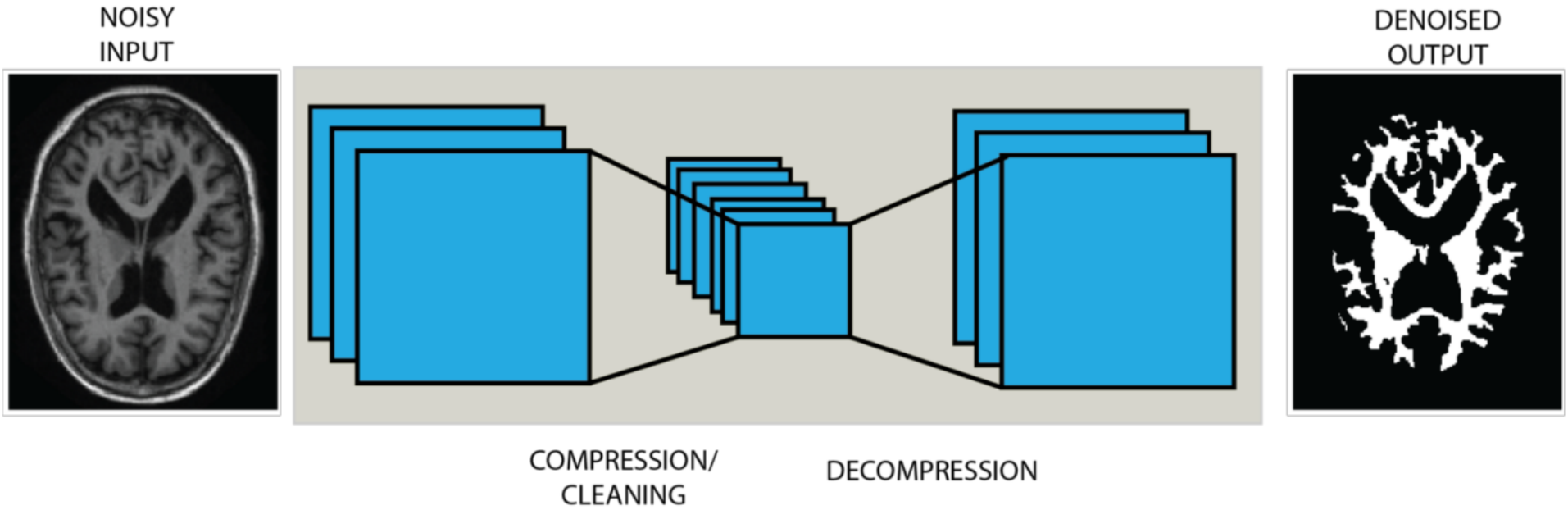
Contractive Autoencders



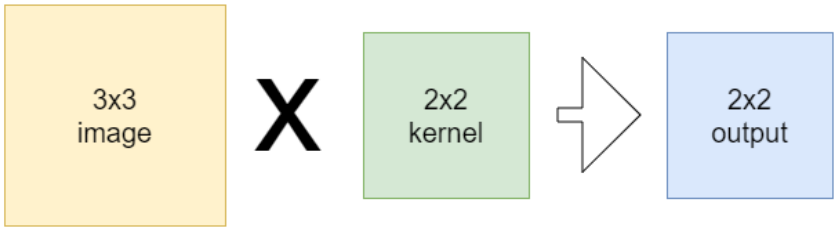
- Consider the variations in the data along directions $\mathbf{u}_1$ and $\mathbf{u}_2$
- It makes sense to maximize a neuron to be sensitive to variations along $\mathbf{u}_1$
- At the same time it makes sense to inhibit a neuron from being sensitive to variations along $\mathbf{u}_2$ (as there seems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

https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf

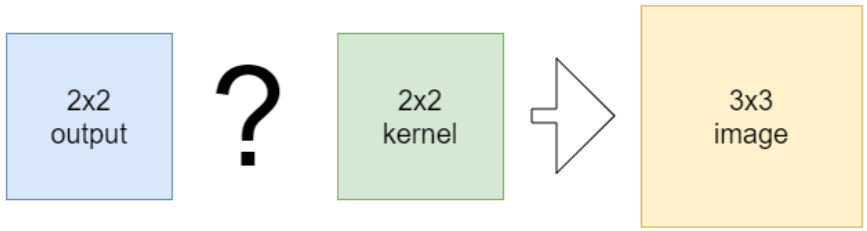
Autoencders



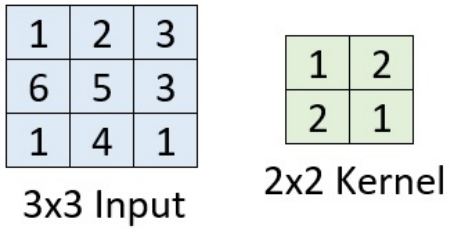
Autoencders



Autoencders



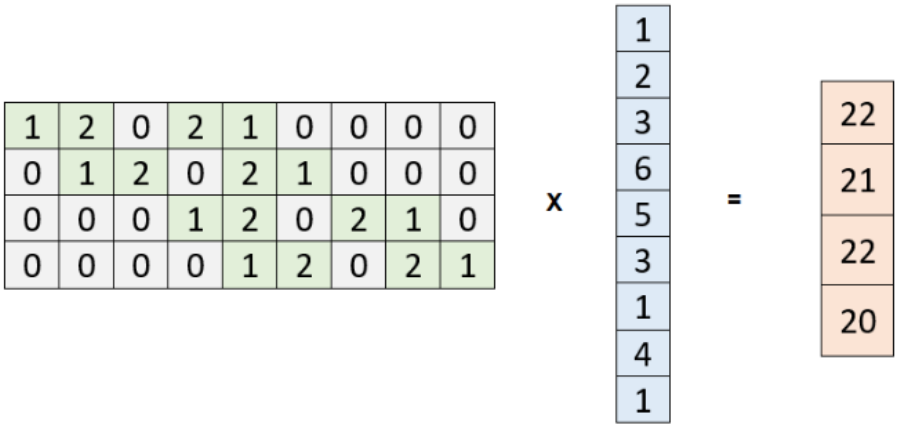
Autoencders



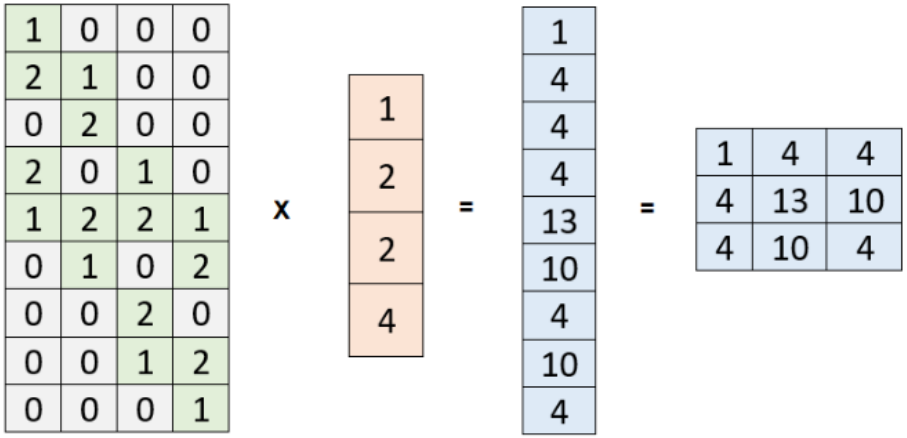
Autoencders

22	21
22	20

Autoencders



Autoencders



Variational autoencoders

Variational autoencoders (VAEs) were introduced by Kingma and Welling (2014). A fairly good tutorial is provided by Doersch (2016).

VAEs, instead of learning $f()$ and $g()$, learn distributions of the features given the input and the input given the activations, i.e., probabilistic versions of $f()$ and $g()$. Concretely, the VAE will learn:

- $p(\mathbf{h}|\mathbf{x})$: the distribution of the features given the input.
- $p(\mathbf{x}|\mathbf{h})$: the distribution of the input given the features.

These distributions are parametrized by neural networks, meaning that they can express nonlinear transformations, and be trained via stochastic gradient descent.

For the rest of these notes, to stay consistent with typical VAE papers, we'll let $\mathbf{z}$ denote the features, i.e., $\mathbf{z} = \mathbf{h}$.

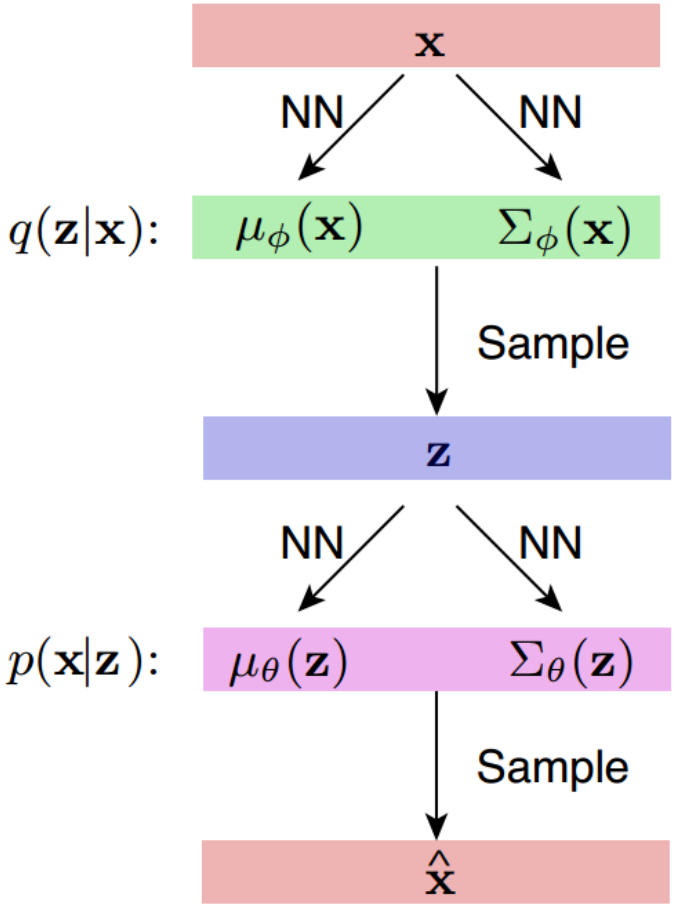
Why learn distributions?

Why might it be good to learn distributions?

- Often times the data is noisy, and a model of the distribution of the data is more useful for a given application.
- Further, often times the relationship between the observed variables and the latent variables is *nonlinear*, in which case the VAE provides a way to do inference.
- The VAE is a generative model; by learning $p(\mathbf{x}|\mathbf{z})$, it is possible to sample $\mathbf{z}$ and then sample $\mathbf{x}$. This enables the generation of data that has similar statistics to the input.

Conceptual diagram of the VAE

Below is a conceptual diagram of the VAE that we will arrive at. However, we put it here to give you a high-level intuition to start:



Conceptual diagram of the VAE (cont.)

This conceptual block diagram of the VAE looks very similar to an autoencoder, with one major change: instead of inferring $\mathbf{z}$ directly from $\mathbf{x}$ and $\hat{\mathbf{x}}$ directly from $\mathbf{z}$, we instead infer distributions.

- Hence, the inference changes from learning $\mathbf{z} = f(\mathbf{x})$ to learning $q(\mathbf{z}|\mathbf{x})$; and from learning $\hat{\mathbf{x}} = g(\mathbf{z})$ to learning $p(\mathbf{x}|\mathbf{z})$.
- Because we learn distributions, to obtain values for $\mathbf{z}$ and $\hat{\mathbf{x}}$, we instead *sample* from these learned distributions.
- In this manner, the VAE is a probabilistic version of the autoencoder.

Formulating the VAE

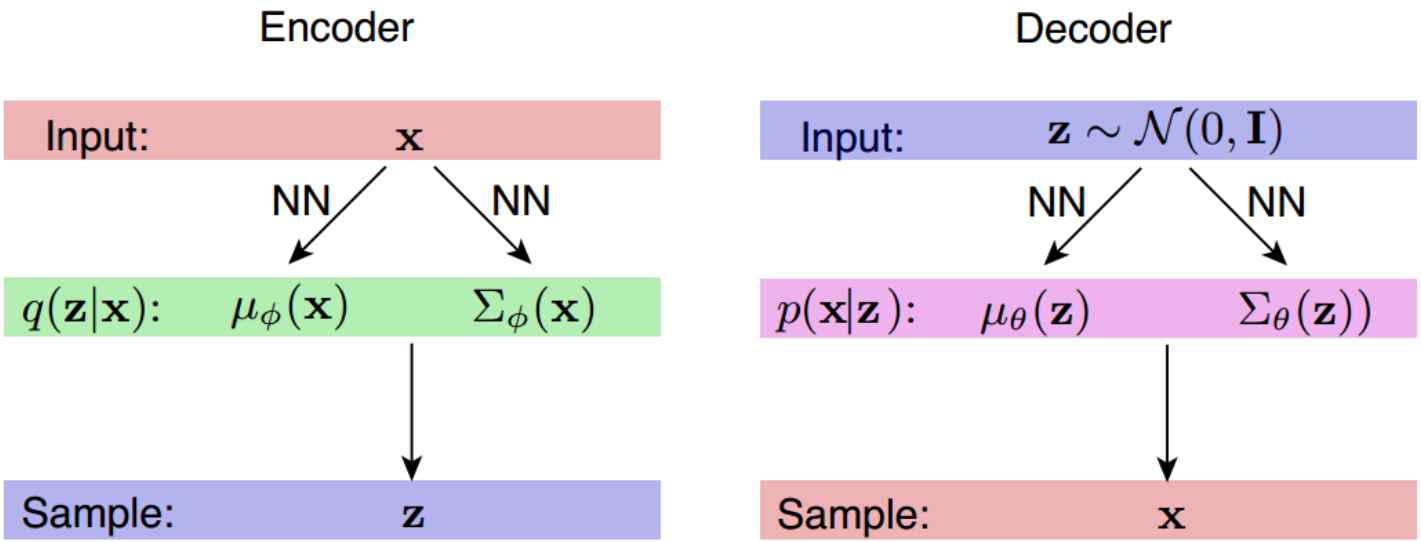
We'll approach the VAE from the context of a generative model. Say we want to generate samples from the data distribution, $p(\mathbf{x})$. Instead of inferring $p(\mathbf{x})$ directly, we can use what are called *latent variable models*.

Latent variable models model the data, $\mathbf{x}$, as arising from an unobserved (and hence latent) variable, $\mathbf{z}$. It may not be easy to model $p(\mathbf{x})$ directly, but it may be easier to choose some distribution $p(\mathbf{z})$ and instead model $p(\mathbf{x}|\mathbf{z})$.

Intuitively, this means that we model how $\mathbf{x}$ arises from some latent variables, $\mathbf{z}$, that themselves have their own variability. Concretely, $p(\mathbf{x}|\mathbf{z})$ can be defined by some mapping $\mathbf{x} = g(\mathbf{z})$. The prior distribution, $p(\mathbf{z})$ and the function $g()$ then define the distribution $p(\mathbf{x}|\mathbf{z})$.

Intuition of the VAE

At this point, our VAE has the following high-level construction (note: we haven't defined the ELBO yet, but will shortly).



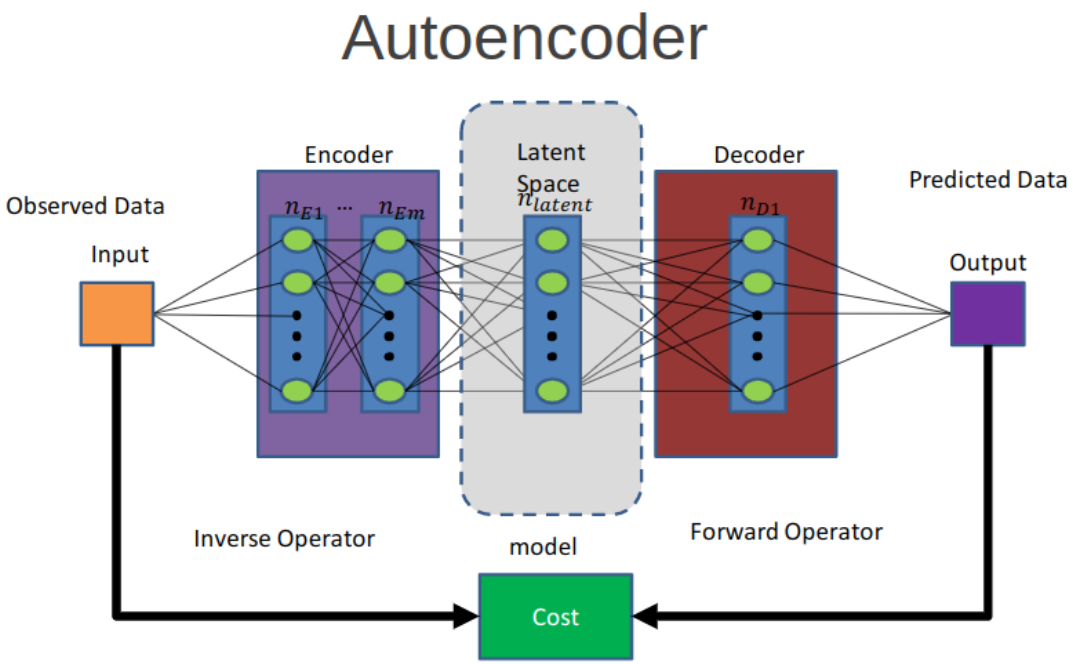
- To sample a latent variable, we use the encoder network: take an input, $\mathbf{x}$, and calculate $q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x}))$. Then, sample $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})$.
- To sample an input, we draw our latent variable $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$. We then calculate $p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mu_\theta(\mathbf{x}), \Sigma_\phi(\mathbf{x}))$. Then, we sample $\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$.

Caveats about the VAE

There are some caveats about the VAE that we need to be aware of.

- Inference used a lower bound rather than the likelihood of the data. Empirically, it has been observed that when the bound is maximized, the gap between the bound and the likelihood is not large. But there might still be a large gap between the maximum-likelihood and the bound.
- There currently remains no proof that VAEs are asymptotically consistent.
- Subjectively, VAEs generate images but other generative models do better.

Variational Autoencoders



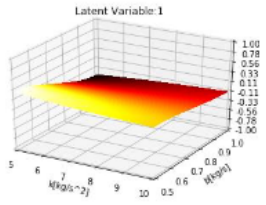
Variational Autoencoders

DAMPED OSCILLATION

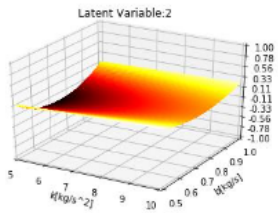
Three Latent variables

Damped: $x(k,b) = \exp\left(-\frac{bt}{2}\right)\cos\left(\sqrt{k\left(1-\frac{b^2}{4k}\right)}\,t\right)$

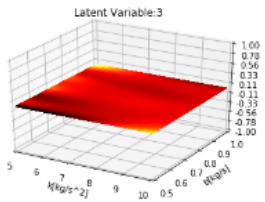
Generated oscillations for a mesh grid of
b: [0.5,1]
k:[5,10]
X:[0,10] 50 points



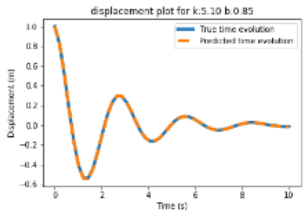
Captures variation in k



Captures variation in b



Captures almost nothing

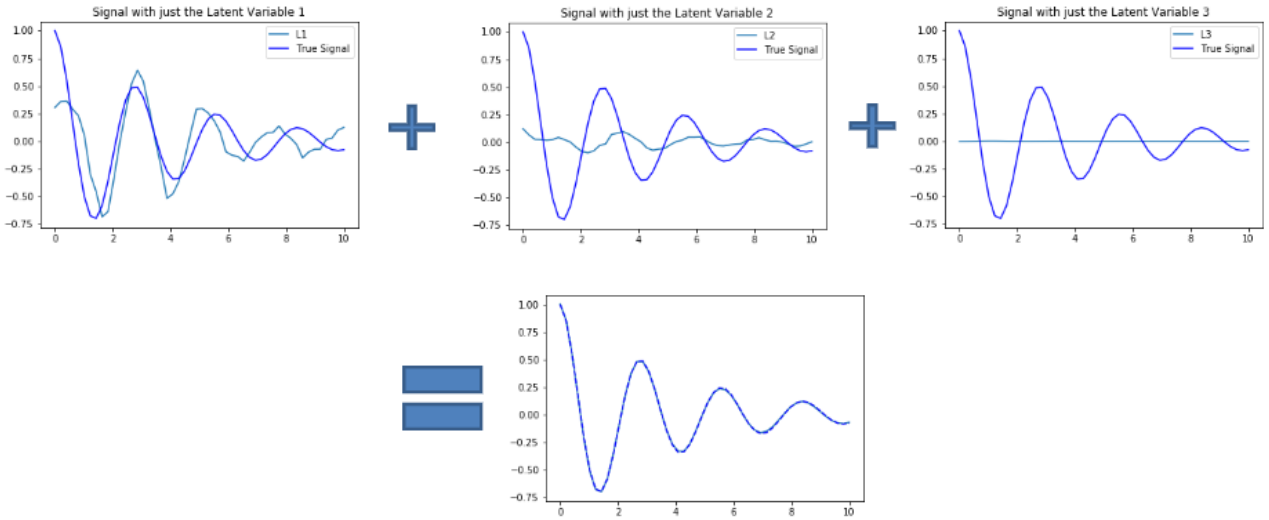


Used Two layers in Encoder each having 100 neurons
Used Two layers in Decoder each having 100 neurons

Variational Autoencoders

Examining the Network and weights

Let us try to reconstruct the signal and see contribution from each latent variable



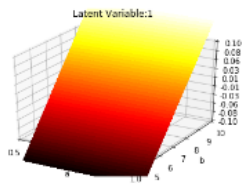
Variational Autoencoders

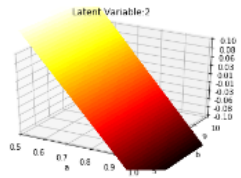
Linear Equation

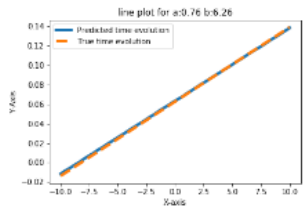
$$y = ax + b$$

Generated lines for a mesh grid of
a: [0.5,1]
b:[5,10]
X:[-10,10] 50 points

Two Latent variables



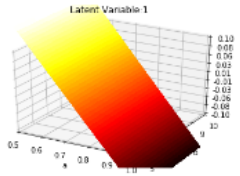




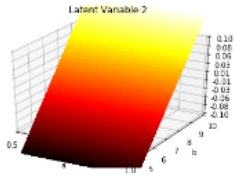
Used Two layers in Encoder each having 100 neurons
Used One layer in Decoder having 100 neurons

Variational Autoencoders

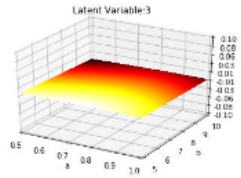
Three Latent variables

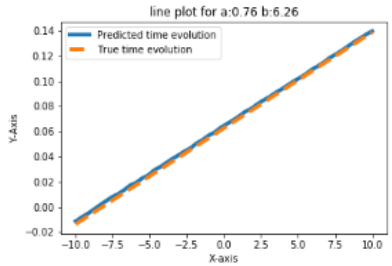


Sensitive to b



Sensitive to a





Used Two layers in Encoder each having 100 neurons
Used One layer in Decoder having 100 neurons

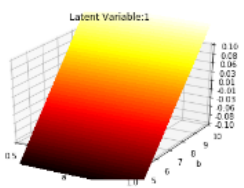
Variational Autoencoders

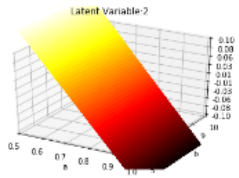
Linear Equation

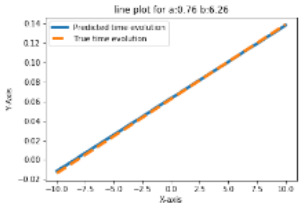
$$y = ax + b$$

Generated lines for a mesh grid of
a: [0.5,1]
b:[5,10]
X:[-10,10] 50 points

Two Latent variables

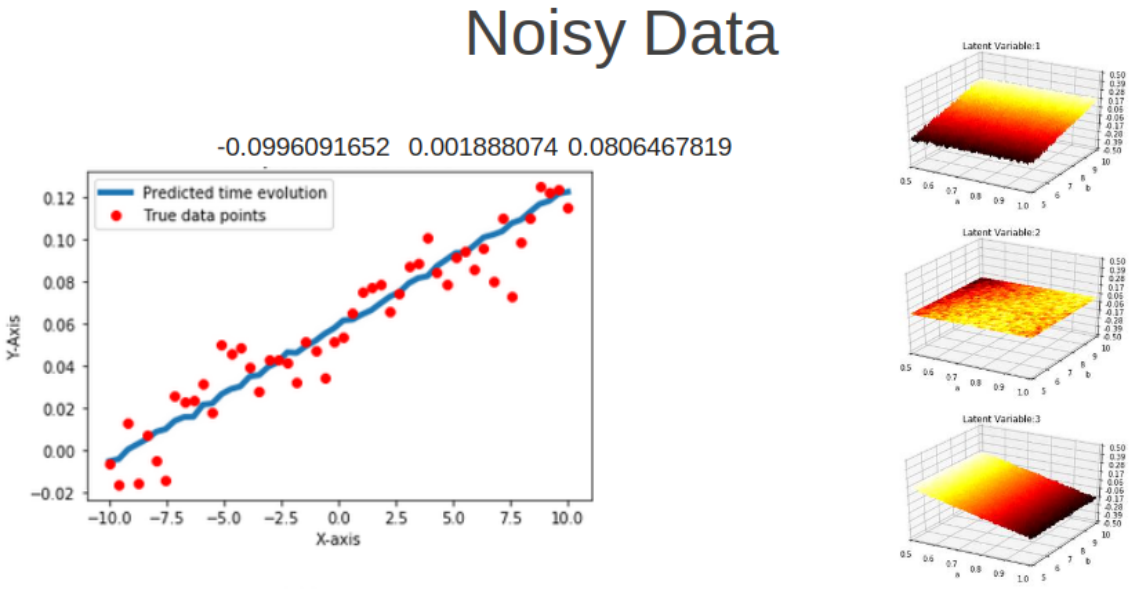






Used Two layers in Encoder each having 100 neurons
Used One layer in Decoder having 100 neurons

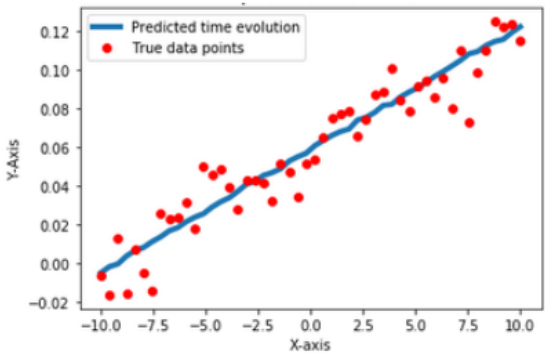
Variational Autoencoders

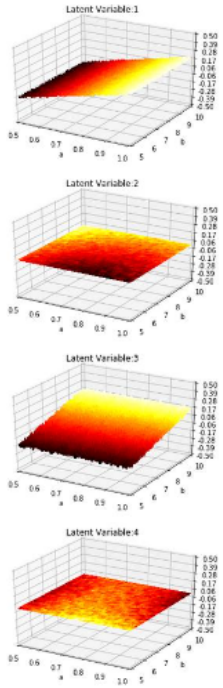


Variational Autoencoders

Noisy Data

:[0.07464643 -0.0051103 -0.10335863 -0.00602804]





Variational Autoencoders