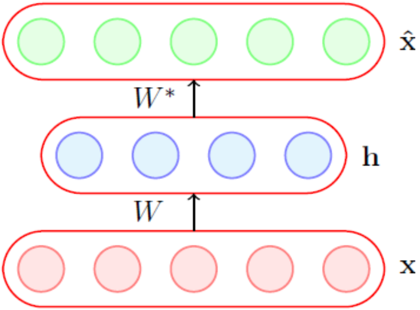
Autoencoders

Joydeep Chandra
C.S.E department
IIT Patna
Joydeep@iitp.ac.in

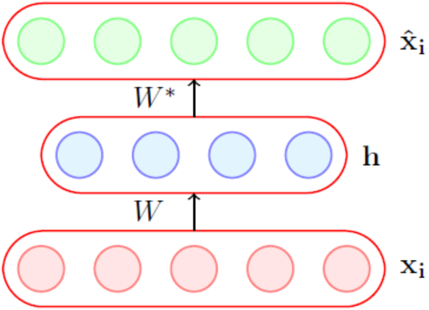
Overview



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

- 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)

Overview

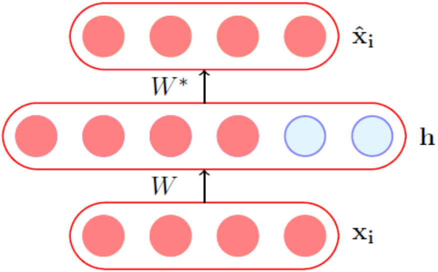


- Let us consider the case where $\dim(\mathbf{h}) < \dim(\mathbf{x_i})$
- If we are still able to reconstruct $\hat{\mathbf{x}}_\mathbf{i}$ perfectly from $\mathbf{h}$, then what does it say about $\mathbf{h}$?
- $\mathbf{h}$ is a loss-free encoding of $\mathbf{x_i}$. It captures all the important characteristics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

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

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

Overview



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

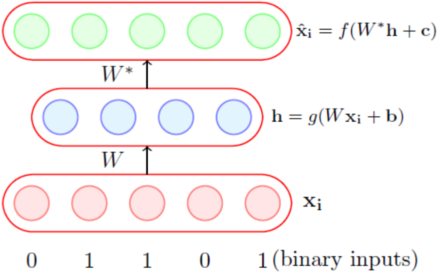
- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x_i})$
- In such a case the autoencoder could learn a trivial encoding by simply copying $\mathbf{x_i}$ into $\mathbf{h}$ and then copying $\mathbf{h}$ into $\hat{\mathbf{x}}_i$
- Such an identity encoding is useless in practice as it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

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

Design choices

- What should be the choice of $f()$ and $g()$?
- What should be the loss function?

Choice of $f(x)$ and $g(x)$



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

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

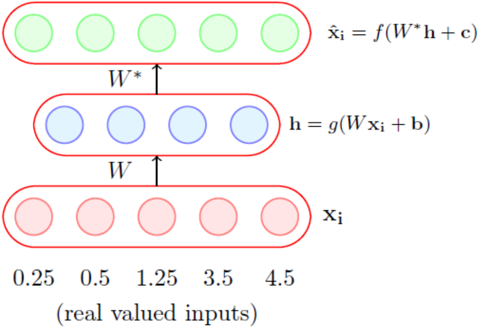
$$\hat{\mathbf{x}}_\mathbf{i} = W^*\mathbf{h} + \mathbf{c}$$

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

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

g is typically chosen as the sigmoid function

Choice of $f(x)$ and $g(x)$



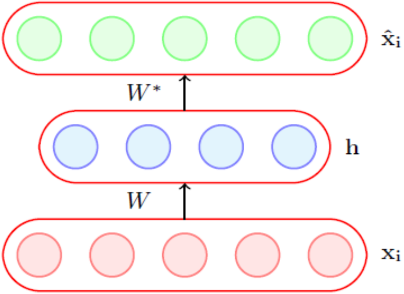
Again, g is typically chosen as the sigmoid function

- 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 $\hat{\mathbf{x}}_i \in \mathbb{R}^n$

Choice of loss function: Real x



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

- 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:

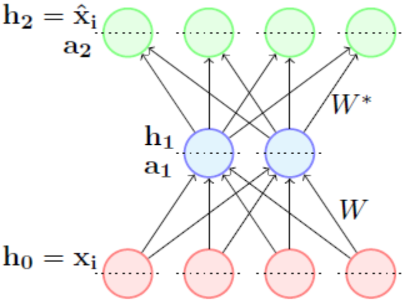
$$\min_{W,W^*,\mathbf{c},\mathbf{b}} \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{x}_{ij} - x_{ij})^2$$

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

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

Choice of Loss function: Real x

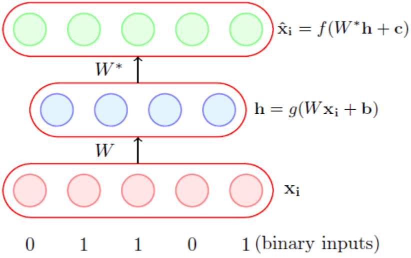
$$\mathscr{L}(\theta) = (\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})$$



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

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

Choice of loss function: Binary x



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

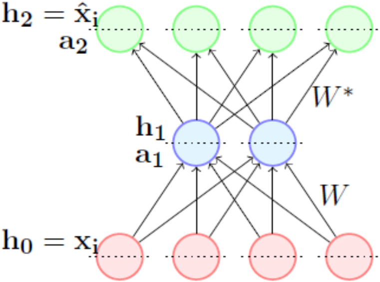
- Consider the case when the inputs are binary
- We use a sigmoid decoder which will produce outputs between 0 and 1, and can be interpreted as probabilities.
- For a single n-dimensional i^{th} input we can use the following loss function

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

- 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

Choice of loss function: Binary x

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



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

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

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

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

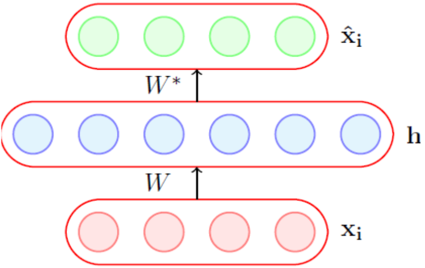
Equivalence of Autoencoders and PCA

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

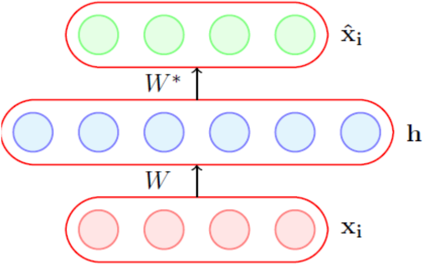
$$\hat{x}_{ij} = \frac{1}{\sqrt{m}}\left(x_{ij} - \frac{1}{m}\sum_{k=1}^{m} x_{kj}\right)$$

Regularization in Autoencoders



- 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

Regularization in autoencoders

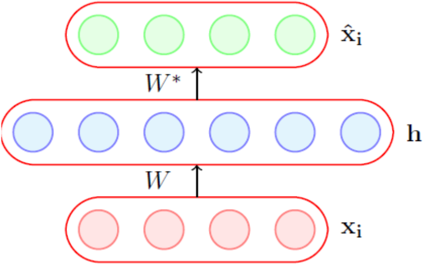


- The simplest solution is to add a L_2-regularization term to the objective function

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

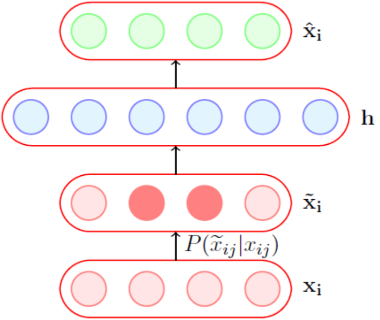
- This is very easy to implement and just adds a term λW to the gradient $\frac{\partial \mathscr{L}(\theta)}{\partial W}$ (and similarly for other parameters)

Regularization in autoencoders



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

Denoising autoencoders

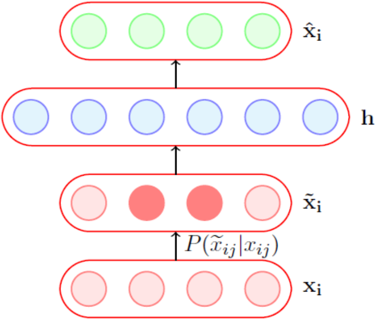


- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\widetilde{x}_{ij}|x_{ij})$) before feeding it to the network
- A simple $P(\widetilde{x}_{ij}|x_{ij})$ used in practice is the following

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

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

Denoising Autoencoders: How does it help?



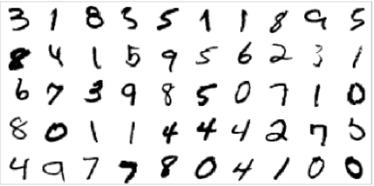
For example, it will have to learn to reconstruct a corrupted x_{ij} correctly by relying on its interactions with other elements of $\mathbf{x}_i$

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

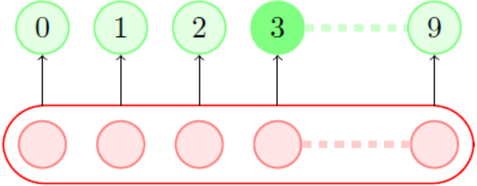
$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$

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

Practical Applications: Handwritten digit recognition



MNIST Data

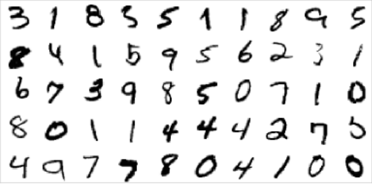


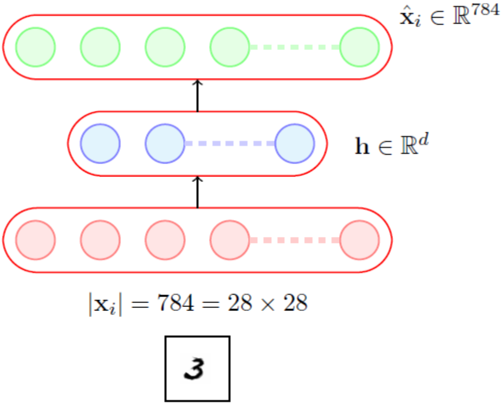
$$|\mathbf{x}_i| = 784 = 28 \times 28$$

28*28

Basic approach: Raw data as input features

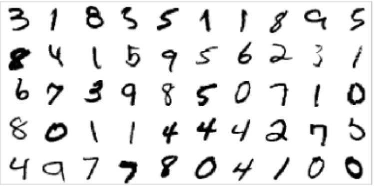
Practical Applications: Handwriting Recognition

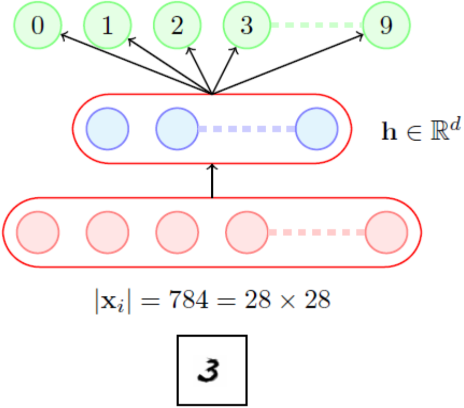




AE approach: Learn important characteristic of the data

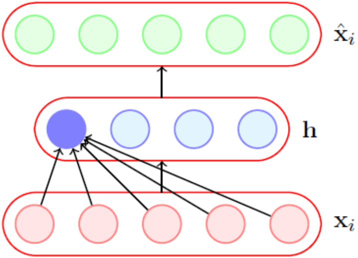
Practical Application: Handwriting Recognition





AE approach: Train a classifier on top of hidden representation

Visualizing Autoencoder Representations



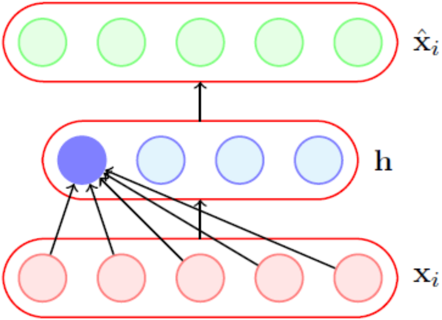
$$\max_{\mathbf{x}_i} \quad \{W_1^T \mathbf{x}_i\}$$
$$s.t. \quad ||\mathbf{x}_i||^2 = \mathbf{x}_i^T \mathbf{x}_i = 1$$

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

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

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

Visualizing Autoencoder Representations



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

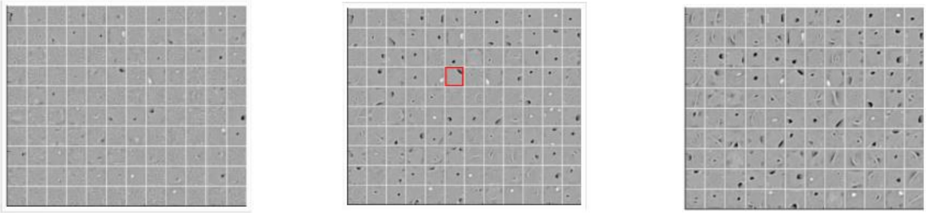
$$s.t. \quad ||\mathbf{x}_i||^2 = \mathbf{x}_i^T \mathbf{x}_i = 1$$

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

- Thus the inputs

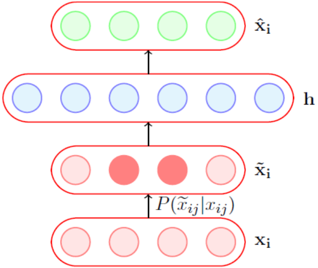
$$\mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}}, \frac{W_2}{\sqrt{W_2^T W_2}}, \ldots \frac{W_n}{\sqrt{W_n^T W_n}}$$

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



- 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

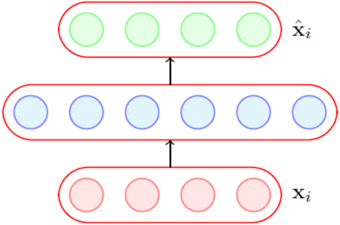
Alternate forms of Denoising AE



- 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

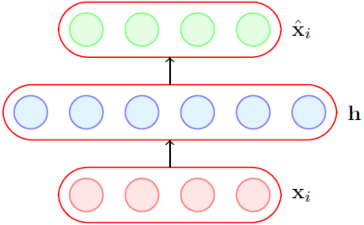
$$\widetilde{x}_{ij} = x_{ij} + \mathcal{N}(0, 1)$$

Sparse Autoencoder



- 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.

Sparse Autoencoders



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

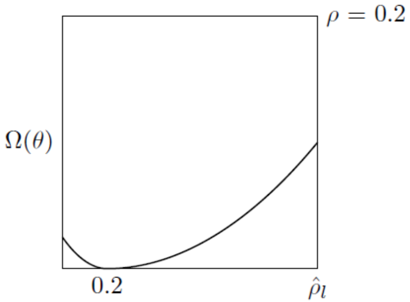
$$\hat{\rho}_l = \frac{1}{m} \sum_{i=1}^{m} h(\mathbf{x}_i)_l$$

- 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.

Sparse Autoencoders



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

- Now,

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

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

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

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

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

Can be re-written as

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

By Chain rule:

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

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

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

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

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

Derivation

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

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}:-

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

So in matrix notation we can write it as :

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

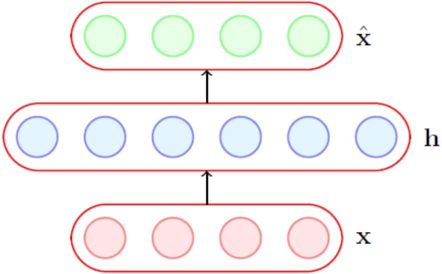
Contrastive autoencoders

- 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.



Contrastive Autoencoders

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

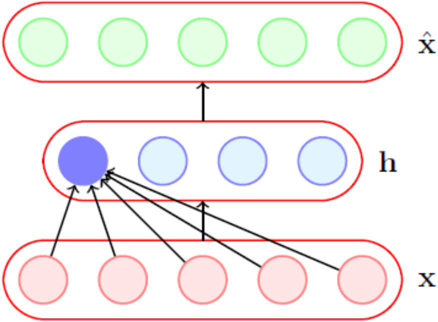
$$J_{\mathbf{x}}(\mathbf{h}) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \dots & \dots & \dots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \dots & \dots & \dots & \frac{\partial h_2}{\partial x_n} \\ \vdots & & \ddots & & \vdots \\ \frac{\partial h_k}{\partial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

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

Contrastive Autoencoder

- 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.

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



Contrastive autoencoder

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

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