BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Deep Learning
Module 7: Autoencoders

Seetha Parameswaran
Asst Prof, BITS Pilani

The instructor is gratefully acknowledging the authors who made their course materials freely available online.

In This Segment

1. Supervised vs Unsupervised Learning Experience
2. Autoencoders
3. Undercomplete Autoencoders
4. Regularized Autoencoders
5. Convolutional Autoencoders
6. Denoising Autoencoders
7. Sparse Autoencoders
8. Deep Autoencoders

Supervised Learning

Data: m training examples (X, y)
X is the data and y is the label.

Goal: Learn a function to map data to the label. $y = f(X)$

Example Tasks: Classification, regression, object detection, semantic segmentation,image captioning,etc.

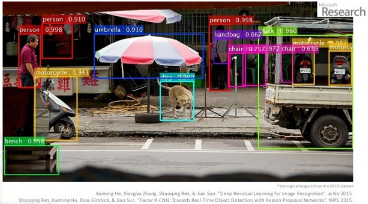


Figure: Object recognition: He: Faster R-CNN

Figure: OCR: Hui Li, Wang, & Shen

Supervised Learning

Data: m training examples (X, y)
X is the data and y is the label.

Goal: Learn a function to map data to the label. $y = f(X)$

Example Tasks: Classification, regression, object detection, semantic segmentation,image captioning,etc.

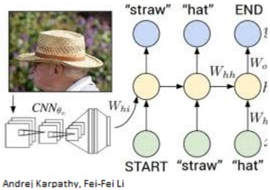


Figure: Image Captioning: Andrej Karpathy, Fei-Fei Li

Figure: Semantic Segmentation

Unsupervised Learning

Data: m training examples (X)
X is the data. No Labels !!!

Goal: Learn some underlying hidden structure of the data.

Example Tasks: Clustering, dimensionality reduction, feature learning, density estimation,etc.

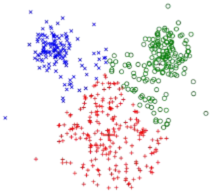


Figure: k-means Clustering

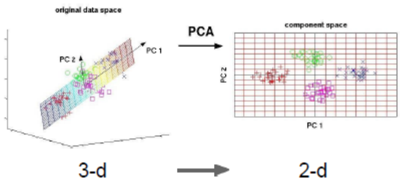


Figure: Dimensionality Reduction

Unsupervised Learning

Data: m training examples (X)
X is the data. No Labels !!!

Goal: Learn some underlying hidden structure of the data.

Example Tasks: Clustering, dimensionality reduction, feature learning, density estimation,etc.

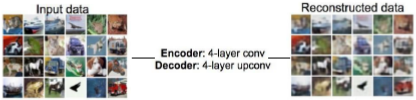
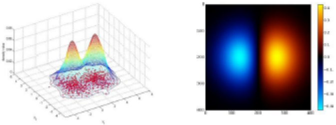


Figure: Feature Learning using Autoencoders

1-d density estimation



2-d density estimation

Figure: Density Estimation

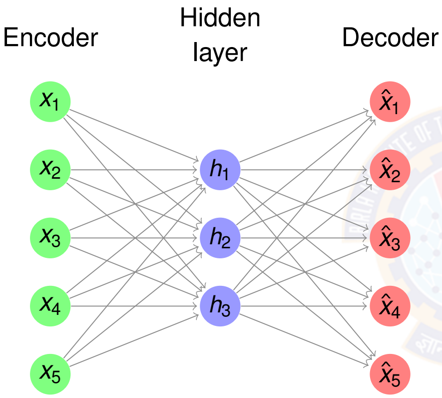
In This Segment

1. Supervised vs Unsupervised Learning Experience
2. **Autoencoders**
3. Undercomplete Autoencoders
4. Regularized Autoencoders
5. Convolutional Autoencoders
6. Denoising Autoencoders
7. Sparse Autoencoders
8. Deep Autoencoders

Autoencoders

- **Unsupervised** experience.
- Learns a lower-dimensional feature representation called **code**; from unlabeled training data.
- The code can be used as features also. Autoencoders are efficient **feature detectors**.
- Autoencoders perform **dimensionality reduction**.
- Trained to attempt to copy its input to its output.

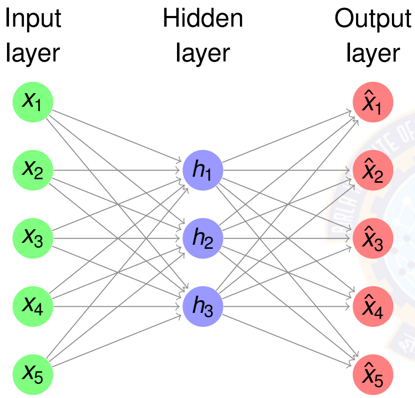
Autoencoders Architecture



$$h = f(W_e x_i + b)$$
$$\hat{x}_i = g(W_d h + c)$$

- An autoencoder is a special type of feed forward neural network which does the following.
- **Encodes** its input x_i into a hidden representation h.
- **Decodes** the input again from this hidden representation.
- The model is trained to minimize a certain loss function which will ensure that $\hat{x}_i$ is close to x_i.

Autoencoders Architecture



$$h = f(W_e x_i + b)$$
$$\hat{x}_i = g(W_d h + c)$$

- Feedforward deep neural network
- Contains input layer, hidden layer and output layers.
- The number of output neurons is exactly same as the number of input neurons
- The hidden layer describes the code used to represent the input.
- All techniques and optimizations of DNN are applicable.
- Train the autoencoder such that features can be used to reconstruct original data.

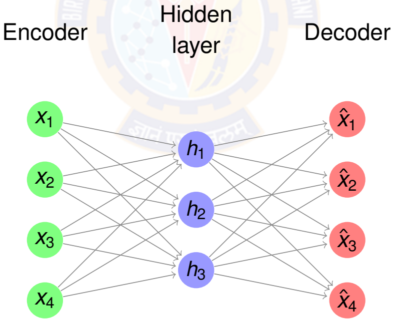
In This Segment

Undercomplete Autoencoders

- One hidden layer which has lesser units than input layer.

$$dim(h) < dim(x)$$

- Input and output layers have equal number of neurons.
- h is a loss-free encoding of x_i . It captures all important characteristics of x_i.



Undercomplete Autoencoders

- Components of Autoencoders
  - Encoder (recognition network)
    - converts input to code.

$$h = f(x) = f(W_e x + b)$$

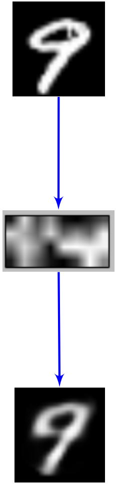
    - Code h has a smaller representation than input x.
  - Decoder (generative network)
    - produces a reconstruction.

$$\hat{x} = g(h) = g(W_d h + c) = g(f(x))$$

  - Loss function
    - reconstruction loss which penalizes when outputs are different from input.
    - Mean squared error

$$\mathcal{L}(x, \hat{x}) = ||x - \hat{x}||^2$$

Undercomplete Autoencoders



Input data

Codes

Reconstructed input data

x

$h = f(x)$ Encoder

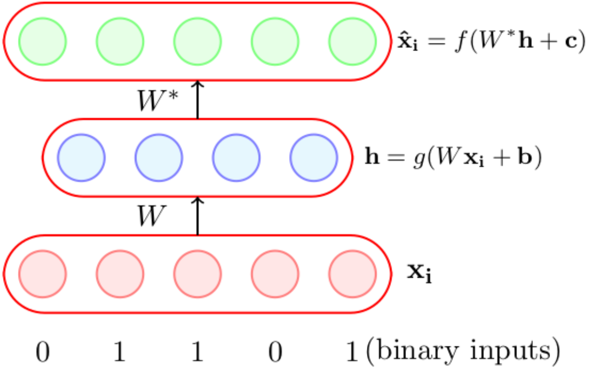
h

$\hat{x} = g(h)$ Decoder

$\hat{x}$

Loss function $\rightarrow ||x - \hat{x}||^2$

Autoencoders – Binary Input



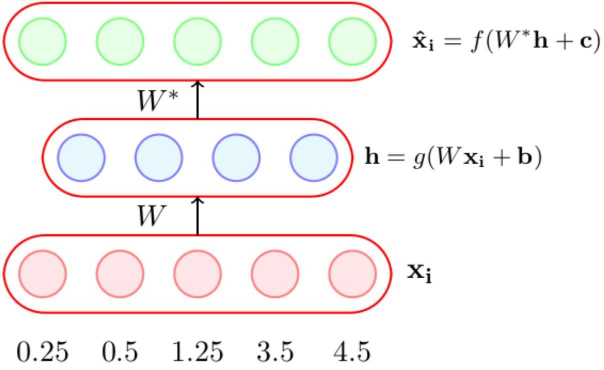
g is typically chosen as the sigmoid function

- Suppose all our inputs are binar (each $x_{ij} \in \{0, 1\}$)
- Which of the following functio 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 a outputs to be between 0 and 1

Autoencoders – Real Input



(real valued inputs)

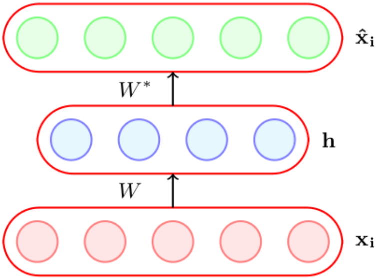
Again, g is typically chosen as the sigmoid function

- Suppose all our inputs are real (eac $x_{ij} \in \mathbb{R}$)
- Which of the following functio 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 reconstru ted $\hat{\mathbf{x}}_\mathbf{i}$ to lie between [0,1] or [-1, whereas we want $\hat{\mathbf{x}}_\mathbf{i} \in \mathbb{R}^n$

Autoencoders – Learning



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

- 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:

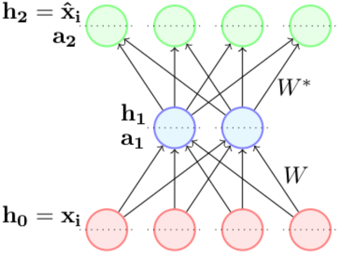
$$\min_{W,W^*,\mathbf{c},\mathbf{b}} \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{x}_{ij} - x_{ij})^2$$

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

- 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

Autoencoders – Learning

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



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

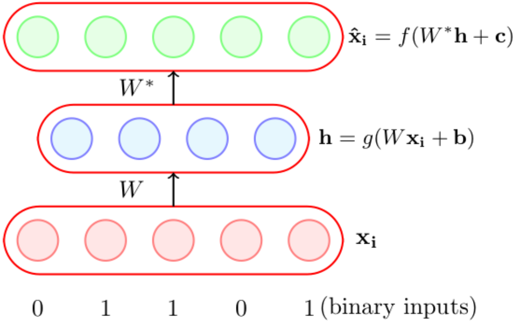
- $$\frac{\partial \mathscr{L}(\theta)}{\partial W^*} = \frac{\partial \mathscr{L}(\theta)}{\partial \mathbf{h_2}} \boxed{\frac{\partial \mathbf{h_2}}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial W^*}}$$

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

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

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

Autoencoders – Learning



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

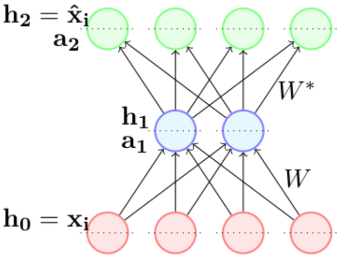
- Consider the case when the inputs are binary
- We use a sigmoid decoder which will produce outputs between 0 and 1, and can be interpreted as probabilities.
- For a single n-dimensional i^{th} input we can use the following loss function

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

- 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

Autoencoders – Learning

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



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

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

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

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

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
  - normalize the inputs to

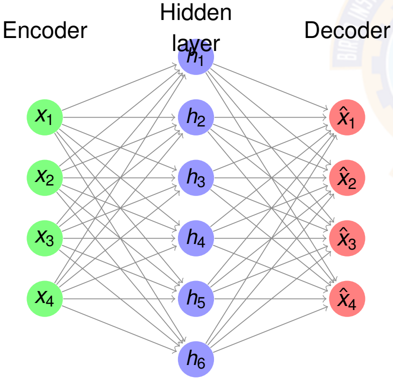
$$r_{ij} = \frac{1}{\sqrt{m}} \left(x_{ij} - \frac{1}{m} \sum_{k=1}^{m} x_{kj} \right)$$

Overcomplete Autoencoders

- One hidden layer which has more units than input layer.

$$dim(h) \geq dim(x)$$

- Learn a trivial encoding by simply copying x_i into h and then copying h into $\hat{x}_i$.



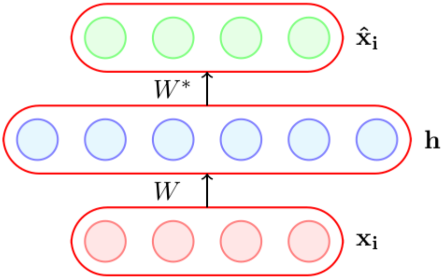
$$h = g(W_e x_i + b)$$
$$\hat{x}_i = f(W_d h + c)$$

In This Segment

Regularized Autoencoders

- Autoencoders fail to learn anything useful if the
  - the encoder and decoder are given too much capacity.
  - the hidden code is allowed to have dimension equal to the input.
  - the hidden code has dimension greater than the input. (overcomplete case)
- Use Regularized autoencoders to resolve the above problems.
- To avoid poor generalization, introduce regularization.
- Introduces properties
  - sparsity of the representation
  - smallness of the derivative of the representation
  - robustness to noise or to missing inputs.

Regularized Autoencoders – L2

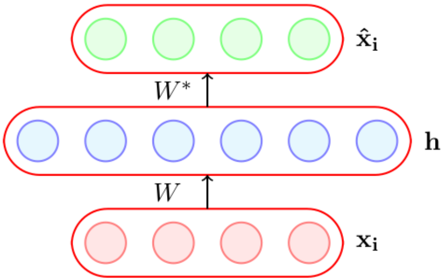


- The simplest solution is to add a L_2-regularization term to the objective function

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

- 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)

Regularized Autoencoders – Weight Sharin



- 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

In This Segment

1. Supervised vs Unsupervised Learning Experience
2. Autoencoders
3. Undercomplete Autoencoders
4. Regularized Autoencoders
5. Convolutional Autoencoders
6. Denoising Autoencoders
7. Sparse Autoencoders
8. Deep Autoencoders

Convolutional Autoencoders

- Allows us to increase the size of the output feature map compared to the input feature map.
- Fractionally strided convolution is another term.
- **Strides** In transposed convolutions, we stride over the output; hence, larger strides will result in larger outputs (opposite to regular convolutions).

Convolutional Autoencoders

Regular Convolution:

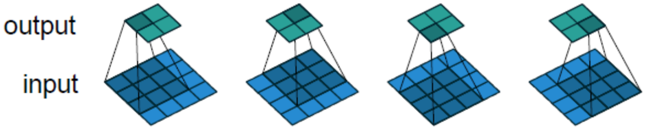
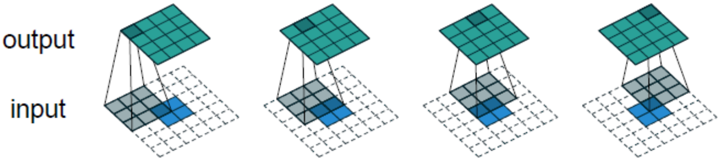


Figure 2.1: (No padding, unit strides) Convolving a $3 * 3$ kernel over a $4 * 4$ input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

Transposed Convolution (emulated with direct convolution):



Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).

Convolutional Autoencoders

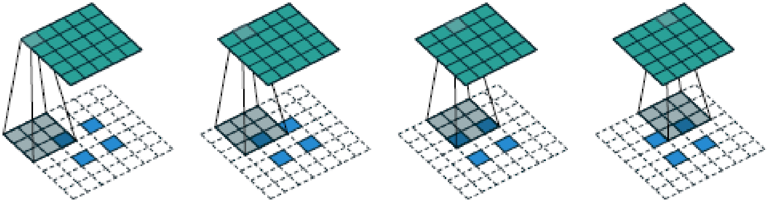


Figure 4.5: The transpose of convolving a 3 * 3 kernel over a 5 * 5 input using 2 * 2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 0$). It is equivalent to convolving a 3 * 3 kernel over a 2 * 2 input (with 1 zero inserted between inputs) padded with a 2 * 2 border of zeros using unit strides

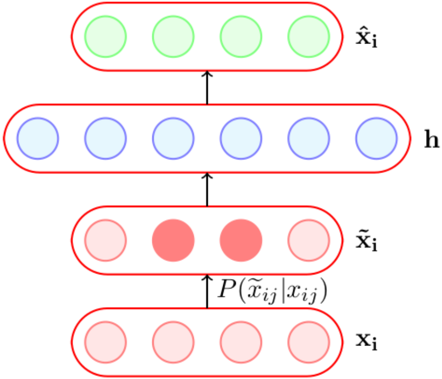
Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).

In This Segment

Denoising Autoencoders (DAE)

- Receives a corrupted data point as input and is trained to predict the original, uncorrupted data point as its output.
- $\tilde{x}$ is a copy of x corrupted by some form of noise.
- The corruption process adds some noise to x according to the conditional distribution $C(\tilde{x}, x)$.
- Denoising autoencoders removes this corruption.

Denoising Autoencoder

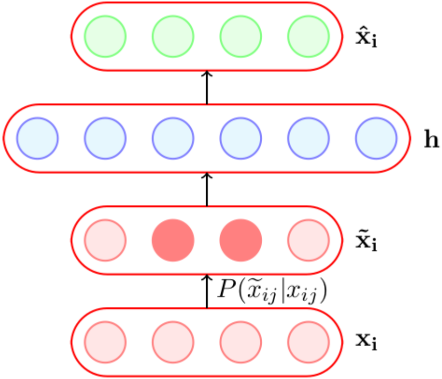


- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\widetilde{x}_{ij}|x_{ij})$) before feeding it to the network
- A simple $P(\widetilde{x}_{ij}|x_{ij})$ used in practice is the following

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

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

Denoising Autoencoder



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 $\widetilde{\mathbf{x}}_i$ 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.

Denoising Autoencoder

Task: Hand-written digit recognition

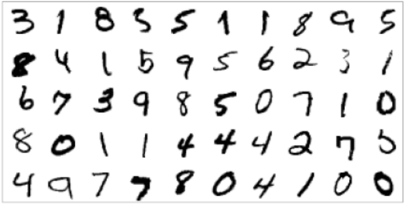


Figure: MNIST Data

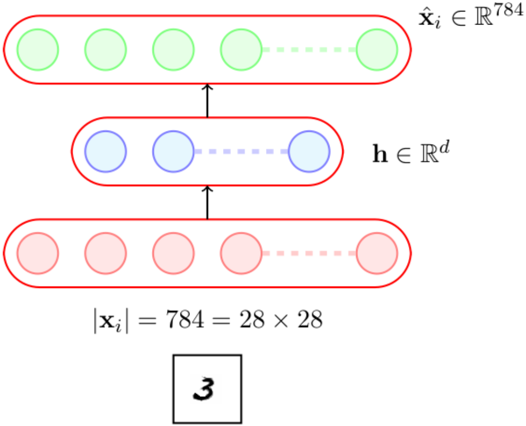


Figure: AE approach (first learn important characteristics of data)

Denoising Autoencoder

Task: Hand-written digit recognition

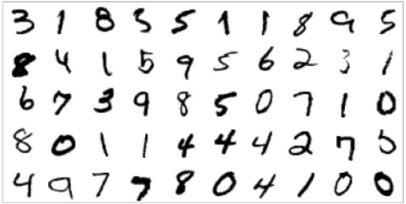


Figure: MNIST Data

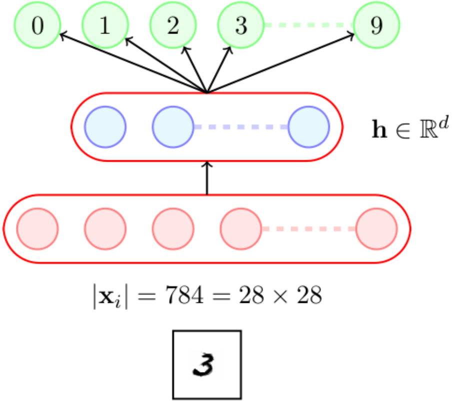


Figure: AE approach (and then train a classifier on top of this hidden representation)

Denoising Autoencoder

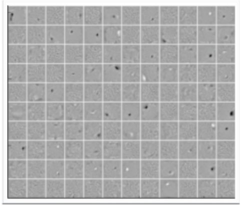


Figure: Vanilla AE (No noise)

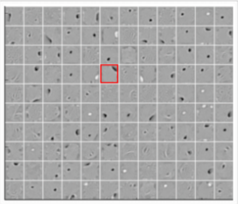


Figure: 25% Denoising AE (q=0.25)

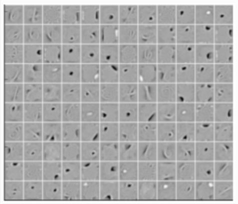
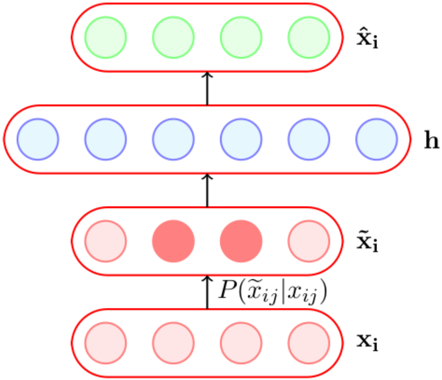


Figure: 50% Denoising AE (q=0.5)

- 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

Denoising Autoencoder



- 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

Denoising Autoencoder

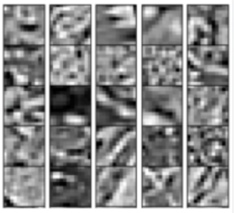


Figure: Data

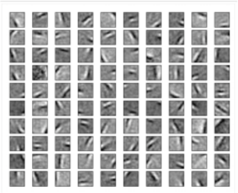


Figure: AE filters

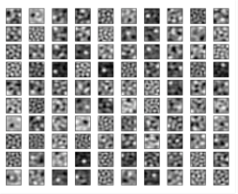
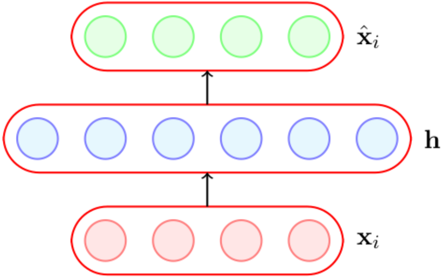


Figure: Weight decay filters

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

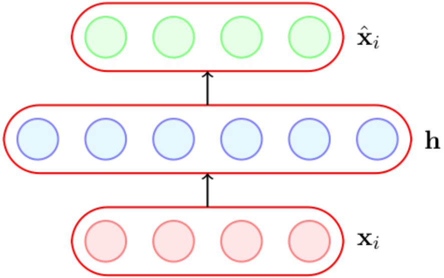
In This Segment

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 Autoencoder



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

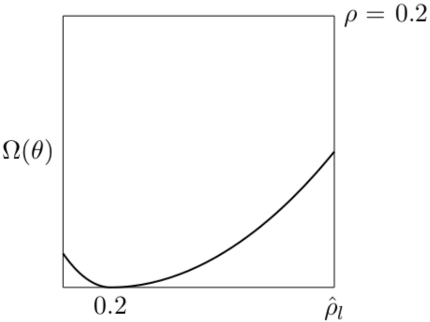
$$\hat{\rho}_l = \frac{1}{m}\sum_{i=1}^{m} h(\mathbf{x}_i)_l$$

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



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

Sparse Autoencoder

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

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)

- 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,

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

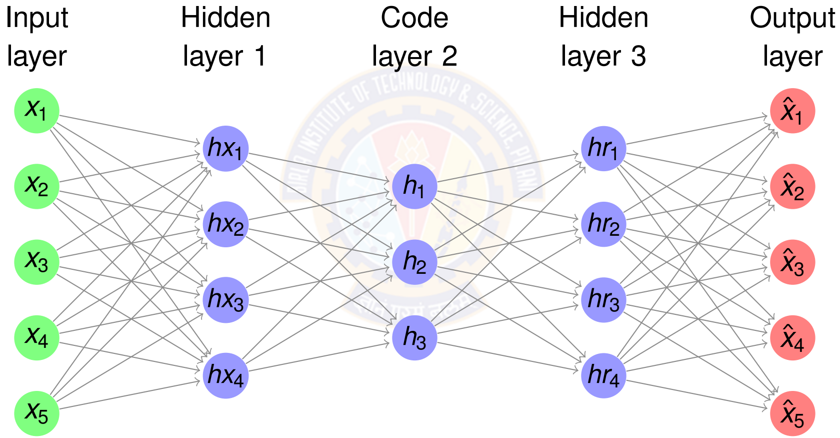
(and we know how to calculate both terms on R.H.S)

In This Segment

Deep Autoencoders

- Input and output layers have hidden units.
- Code layer has lesser units that input layer.
- Depth can exponentially reduce the computational cost of representing some functions.
- Depth can exponentially decrease the amount of training data needed to learn some functions.
- Deep autoencoders yield much better compression than corresponding shallow or linear autoencoders .

Deep Autoencoders



Autoencoder Applications

- Dimensionality reduction (representation learning)
- Information retrieval/semantic hashing tasks - we can store all database entries in a hash table that maps binary code vectors to entries
- Classification
- Denoising autoencoders
- Useful for segmentation and deep-feature
- Neural inpainting

Autoencoder Applications

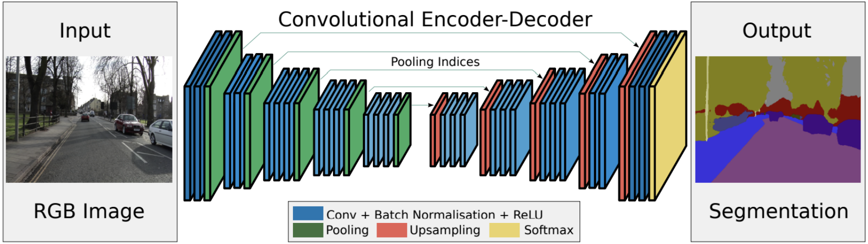


Fig. 2. An illustration of the SegNet architecture. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification.

Autoencoder Applications

A comparison of methods for compressing digit images to 30 real numbers.

real data

30-D deep auto

30-D logistic PCA

30-D PCA

Autoencoder – Summary

- An autoencoder network is actually a pair of two connected networks, an encoder and a decoder.
- An encoder network takes in an input, and converts it into a smaller, dense representation.
- A decoder network can use to convert the dense representation back to the original input.

Input → $\boxed{\text{Encoder}}$ → Code → $\boxed{\text{Decoder}}$ → Output

References

1. Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville
   https://www.deeplearningbook.org/
2. Deep Learning with Python by Francois Chollet.
   https://livebook.manning.com/book/deep-learning-with-python/

Thank You!