Auto-encoder

Nan Ye

School of Mathematics and Physics
The University of Queensland

Recall: More on Dimension Reduction

- Two types of methods
  - Feature selection: find a subset of most important variables.
    - Lasso, LARS, forward/backward selection,...
  - Feature extraction (or feature projection): embed/project the data to a lower dimensional space.
    - PCA, kernel PCA, Isomap, multidimensional scaling, t-SNE, LDA, autoencoder, ...
- We will cover autoencoder in this course (also useful for image compression).

Applications of Autoencoders

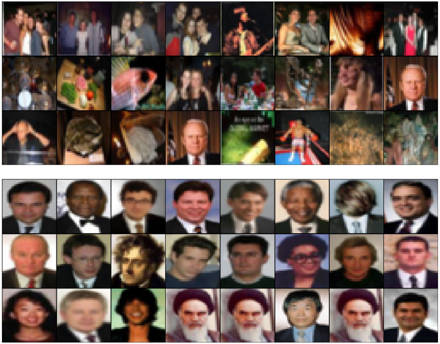
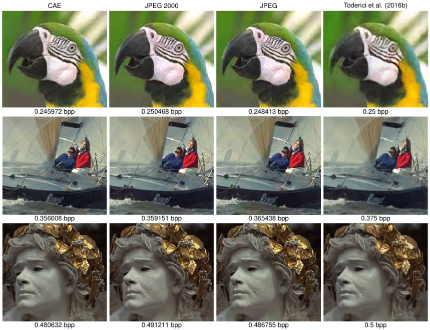


Image retrieval

Krizhevsky and Hinton, Using very deep autoencoders for content-based image retrieval,



Lossy image compression

Theis et al., Lossy image compression with compressive autoencoders, 2017

- Visualization of data by reducing them to 2D vectors
- Using reduced representation as input to train a classifier (in the hope of filtering out noises)
- Denoising speeches

PCA as Neural Nets

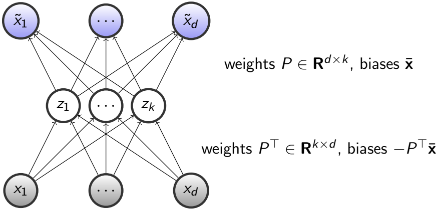
- Suppose the top k principal components for $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbf{R}^d$ are $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \mathbf{R}^d$.
- Let $P = [\mathbf{v}_1, \ldots, \mathbf{v}_k] \in \mathbf{R}^{d \times k}$.
- The k-dimensional representation of $\mathbf{x} \in \mathbf{R}^d$ given by PCA is

$$\mathbf{z} = P^\top(\mathbf{x} - \bar{\mathbf{x}}) \in \mathbf{R}^k,$$

  where $\bar{\mathbf{x}} = \frac{1}{n}\sum_i \mathbf{x}_i$.
- The reconstruction of $\mathbf{x}$ using $\mathbf{z}$ is

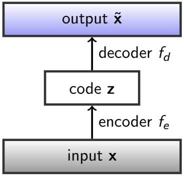
$$\tilde{\mathbf{x}} = P\mathbf{z} + \bar{\mathbf{x}}.$$

- We can represent the mapping from $\mathbf{x}$ to its reconstruction $\tilde{\mathbf{x}}$ using a single hidden layer neural net.



Autoencoders

- Autoencoders generalize the neural net view of PCA to learn a lower dimensional representation of data.
- A basic autoencoder has the following structure



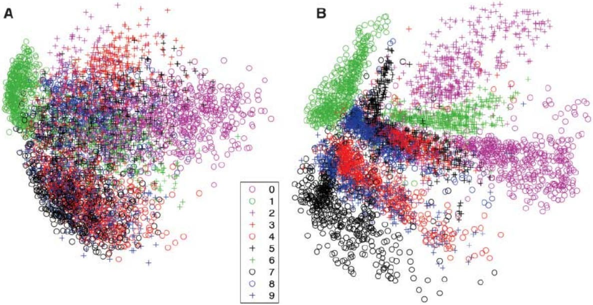
  - The *encoder* f_e is a module that computes the code $\mathbf{z} = f_e(\mathbf{x})$.
  - The *decoder* f_d is a module that computes the reconstruction $\tilde{\mathbf{x}} = f_d(\mathbf{z})$ from the code.
  - The autoencoder optimizes the parameters of f_e and f_d to minimize the reconstruction error $L(\mathbf{x}, \tilde{\mathbf{x}}') = \|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2$.

Undercomplete and overcomplete autoencoders

- *Undercomplete*: dimension of code $<$ dimension of input.
  - an undercomplete code is used for data compression
- *Overcomplete*: dimension of code $\geq$ dimension of input.
  - surprisingly, an overcomplete code can be used to improve classification performance

Linear and nonlinear dimension reduction

- PCA performs linear dimension reduction (code is a linear function of input), and is not suitable for data lying on a nonlinear manifold.
- Autoencoders can be used to perform nonlinear dimension reduction by using nonlinear layers in the encoder and decoder.



PCA and autoencoder codes for MNIST

Hinton and Salakhutdinov, Reducing the dimensionality of data with neural networks, 2006

Sparse Autoencoders

- The basic autoencoder does not pose any constraint on the code, and may not learn a useful representation of data.
- A regularizer is often introduced to encourage sparsity in the code, in the hope that the code will capture only regularity in the data.
  - *sparsity = few non-zero entries*
- Specifically, instead of minimizing $L(\mathbf{x}, \tilde{\mathbf{x}})$, we minimize

$$L(\mathbf{x}, \tilde{\mathbf{x}}) + R(\mathbf{z}),$$

  where $\mathbf{z} = f_e(\mathbf{x})$ is the code for $\mathbf{x}$, and $R(\mathbf{z})$ is a regularizer that favors sparse codes.

ℓ_p **regularizer**

- We can choose
  - $R(\mathbf{z}) = \lambda\|\mathbf{z}\|_1$ (ℓ_1 regularization) or,
  - $R(\mathbf{z}) = \lambda\|\mathbf{z}\|_2^2$ (ℓ_2 regularization).
- ℓ_1 regularization can encourage the code values to be exactly 0.
- ℓ_2 regularization shrinks the code values towards 0.

k-sparse regularizer

- A regularization effect can be obtained by zeroing all but the top k activation values.
- This directly achieve sparsity of a given level, and forces the decoder to reconstruct using a sparse representation.
- The encoder thus need to supply a good code as well.

KL-divergence regularizer

- If each element in the code lies in the range [0, 1], we can use the KL-divergence to encourage the code values to be close to 0 (inactive).
- Assume there are n examples $\mathbf{x}_1, \ldots, \mathbf{x}_n$, and $\mathbf{h} = \frac{1}{n} \sum_i f_e(\mathbf{x}_i)$.
- Given a desired sparsity level $\rho \in [0, 1]$, we train the autoencoder to minimize

$$\sum_i L(\mathbf{x}_i, \tilde{\mathbf{x}}_i) + \sum_j KL(\rho, h_j),$$

  where $KL(\rho, h) = \rho \ln \frac{\rho}{h} + (1 - \rho) \ln \frac{1-\rho}{1-h}$ is the KL-divergence between Bernoulli(p) and Bernoulli($1 - h$).
- $KL(\rho, h)$ is small when h is close to ρ, thus the regularizer encourage the average activation level to to be close to ρ.

Denoising Autoencoders

- Denoising autoencoders aim to reconstruct the input using its perturbed versions.
- Specifically, during training, each input is randomly perturbed, but the reconstruction error is still measured using the original input.
- This can be seen as minimizing the objective

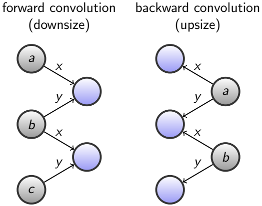
$$\mathbb{E}_{\mathbf{x}^{(noise)} \sim q(\cdot|\mathbf{x})} L(\mathbf{x}, f_d(f_e(\mathbf{x}^{(noise)}))).$$

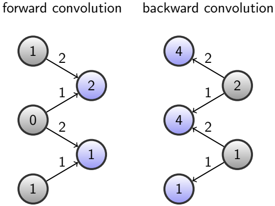
- We can choose the distribution q to propose noisy examples that mimick actual corruptions on the input.
- The resulting denoising autoencoder can then be used to reconstruct original inputs using corrupted inputs.

original noisy reconstructed

Transposed convolution (Optional)

- The encoder requires downsizing an input, and convolution provides a natural way to do this.
- The decoder requires upsizing an input, and transposed convolution provides a way to do this based on convolution.
- We can simply use backward convolution to do this!



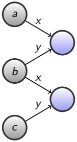


- Clearly $BackwardConv(ForwardConv(x)) \neq x$, i.e., backward convolution is not the inverse of its corresponding forward convolution.

  Note. The discussion here applies to convolution general convolutions.

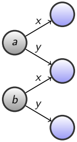
- Backward convolution is often called transposed convolution
  - this is derived from the matrix representation of convolution
  - forward = multiply input by C; backward = multipy input by $C^\top$

convolution
(downsize)



$$\overbrace{\begin{pmatrix} x & y & 0 \\ 0 & x & y \end{pmatrix}}^{C} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} xa + yb \\ xb + yc \end{pmatrix}$$

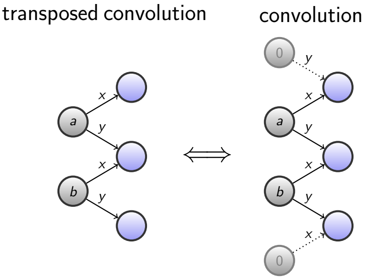
transposed convolution
(upsize)



$$\overbrace{\begin{pmatrix} x & 0 \\ y & x \\ 0 & y \end{pmatrix}}^{C^\top} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} xa \\ ya + xb \\ yb \end{pmatrix}$$

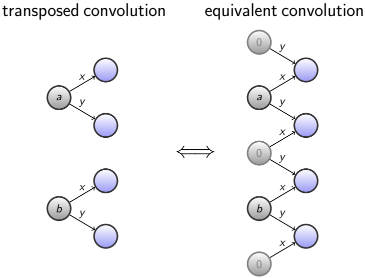
- aka deconvolution, which is an unfortunate misnomer as deconvolution refers to the inverse of convolution in maths

- A transposed convolution is in fact also a convolution



- Why? In backward convolution, each output has the same kind of connectivity patterns to the inputs (when the input tensor is padded with zeros).

- A transposed convolution is also called a fractionally strided convolution
  - Why? For a transposed convolution with $S > 1$, its equivalent convolution is applied to the input tensor with 0 added between the entries
  - $\Rightarrow$ effective stride of the equivalent convolution on the original input tensor is < 1.



What You Need to Know

- From PCA to autoencoders
- Sparse autoencoders
- Denoising autoencoders