PCA and Autoencoders

Xiaozhong Zhang
xiaozhong@pitt.edu

Table of Contents

• PCA: Example
• PCA: Framework
• PCA: Goal
• PCA: Solution
• Autoencoder: Introduction
• Autoencoder: Types
  • Sparse Autoencoder
  • Denoising Autoencoder
  • Variational Autoencoder
• Autoencoder: Example
• References
PCA: Introduction

• PCA: Principal Component Analysis
• Provides a roadmap for dimension reduction
• Used for:
  • Visualization
  • Preprocessing
  • Compression

PCA: Example

• A ball is attached to a massless frictionless spring
• The ball is released a small distance away from equilibrium
• It oscillates along the x-axis at a set frequency
• We want to find the dynamics of the ball
• We place three cameras to observe the ball’s movement
• Each camera records the ball’s position in a 2-D plane
• Due to our ignorance, we choose three camera axes at some arbitrary angles
PCA: Example

- Goal: Based on the camera records, determine that the dynamics are along the x-axis

![Diagram of camera setup]

PCA: A Naïve Basis

- Each data point is expressed along the x, y axis of the 3 camera planes (6 dimension in total)

\[ \bar{X} = \begin{bmatrix} x_A \\ y_A \\ x_B \\ y_B \\ x_C \\ y_C \end{bmatrix} \]

- If we choose a naïve basis with orthonormal basis vectors \( b_i \)

\[ B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \]

- All data can be trivially expressed as a linear combination of \( \{ b_i \} \)

\[ X = BX \]
PCA: Change of Basis

• PCA is trying to re-express the data as a linear combination of its naïve basis vectors:

\[ Y = PBX = PX \] (1)

with the following quantity definition:
• \( p_i \) are the rows of \( P \)
• \( x_i \) are the columns of \( X \)
• \( y_i \) are the columns of \( Y \)

PCA: Change of Basis

• Equation 1 represents a change of basis and can have many interpretations:
  • \( P \) is a matrix that transforms \( X \) to \( Y \)
  • Geometrically, \( P \) is a rotation and a stretch which again transforms \( X \) to \( Y \)
  • The rows of \( P \), namely \( \{ p_1, \ldots, p_m \} \), are a set of new basis vectors for expressing the columns of \( X \), namely each \( x_i \) of \( X \)
PCA: Change of Basis

- The last interpretation can be seen clearly by writing out the explicit dot products of $PX$:

$$PX = \begin{bmatrix} P_1 \\ \vdots \\ P_m \end{bmatrix} \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix} = \begin{bmatrix} P_1 \cdot x_1 & \cdots & P_1 \cdot x_n \\ \vdots & \ddots & \vdots \\ P_m \cdot x_1 & \cdots & P_m \cdot x_n \end{bmatrix}$$

- Each column of $Y$ is:

$$y_i = \begin{bmatrix} P_1 \cdot x_i \\ \vdots \\ P_m \cdot x_i \end{bmatrix}$$

PCA: Goal

- Goal:
  - Find the best way to “re-express” $X$
  - Find a good choice of basis $P$

- What does “best express” the data mean?

- Three potential confounds that “garbles” the data:
  - Noise
  - Rotation
  - Redundancy
PCA: Noise and Rotation

- Signal-to-noise ratio: \( SNR = \frac{\sigma_{signal}^2}{\sigma_{noise}^2} \)
- Dynamics of interest is along direction with high SNR
- Rotate the naïve basis to lie parallel to \( p^* \)

![Diagram showing signal-to-noise ratio and dynamics](image)

PCA: Redundancy

- More meaningful to record one variable in panel (c)
- Because one can calculate \( r_1 \) from \( r_2 \) using best-fit line
- Removing redundancy is the very idea behind dimension reduction

![Diagram showing redundancy](image)
PCA: Covariance Matrix

• Consider a data set in mean deviation form
  \[ X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \]
• The covariance matrix is \( C_X = \frac{1}{n-1}XX^T \)
  • \( C_X \) is a square symmetric matrix
  • The diagonal terms are variance of measurement
    • Large values correspond to interesting dynamics
  • The off-diagonal terms are covariance between measurement
    • Large values correspond to high redundancy

PCA: Diagonalize Covariance Matrix

• We want to transform \( X \) to \( Y \) using a new basis \( P \) such that the covariance matrix becomes \( C_Y \)
  • \( C_Y \) must be diagonal
  • \( \{p_1, \ldots, p_m\} \) in \( P \) are the principal components
• To find \( P \)
  • Find \( p_1 \) corresponds to the vector parallel to the direction with largest variance in \( X \)
  • Find \( p_2 \) with the second largest variance in the remaining directions orthogonal to previously selected directions
  • Repeat until \( m \) vectors are selected
PCA: Assumptions and Limits

• Linearity
  • Linearity frames the problem as a change of basis
  • Kernel PCA: PCA with nonlinear kernels

• Mean and variance are sufficient statistics
  • Mean and variance fully describe the distribution
  • Gaussian, Exponential distribution, etc.

• Large variances have important dynamics

• The principal components are orthogonal
  • Soluble with linear algebra decomposition techniques

PCA: Eigenvectors of Covariance

• Find some orthonormal matrix $P$ where $Y = PX$ such that
  \[ C_Y = \frac{1}{n-1} YY^T \]
  is diagonalized.

• The rows of $P$ are the principal components of $X$. 
PCA: Eigenvectors of Covariance

• Rewrite $C_Y$ in terms of our variable of choice $P$

$$C_Y = \frac{1}{n-1}YY^T$$
$$= \frac{1}{n-1}(PX)(PX)^T$$
$$= \frac{1}{n-1}PXX^TP^T$$
$$= \frac{1}{n-1}P(XX^T)P^T$$
$$C_Y = \frac{1}{n-1}PAP^T$$

PCA: Eigenvectors of Covariance

• With $A = EDE^T$ ($E$ is a matrix of eigenvectors of $A$) and $P \equiv E^T$ and $P^{-1} = P^T$, we have:

$$C_Y = \frac{1}{n-1}PAP^T$$
$$= \frac{1}{n-1}P(P^TDPP^T)P^T$$
$$= \frac{1}{n-1}(PP^T)D(PP^T)$$
$$= \frac{1}{n-1}(PP^{-1})D(PP^{-1})$$
$$C_Y = \frac{1}{n-1}D$$
PCA: Eigenvectors of Covariance

• Results of PCA in the matrices $P$ and $C_Y$
  • The principal components of $X$ are the eigenvectors of $XX^T$; or the rows of $P$.
  • The $i^{th}$ diagonal value of $C_Y$ is the variance of $X$ along $p_i$.

Table of Contents

• PCA: Example
• PCA: Framework
• PCA: Goal
• PCA: Solution
• Autoencoder: Introduction
• Autoencoder: Types
  • Sparse Autoencoder
  • Denoising Autoencoder
  • Variational Autoencoder
• Autoencoder: Example
• References
Introduction to Autoencoders

• An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner.

• The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction.

• Recently, the autoencoder concept has become more widely used for learning generative models of data.
Introduction to Autoencoders

- Modern autoencoders have generalized the idea of an encoder and a decoder beyond deterministic functions to stochastic mappings $p_{\text{encoder}}(h \mid x)$ and $p_{\text{decoder}}(x \mid h)$.

![Diagram of autoencoder](attachment:diagram.png)

Introduction to Autoencoders

- The learning process is minimizing a loss function

\[ L \left( x, g(f(x)) \right) \]

where $L$ is a loss function penalizing $g(f(x))$ for being dissimilar from $x$, such as the mean squared error.

![Diagram of autoencoder](attachment:diagram2.png)
Undercomplete Autoencoders

- If the code dimension is larger than the input dimension, an autoencoder tends to learn $g \circ f$ as an identity function.
- An autoencoder whose code dimension is smaller than the input dimension is called undercomplete.
- When the encoder and decoder are linear and $L$ is the mean squared error, an undercomplete autoencoder learns to span the same subspace as PCA.

Undercomplete Autoencoders

- Define $h = f(Wx)$; $r = g(Vh)$

- Goal

$$
\min_{W,V} \frac{1}{2N} \sum_{n=1}^{N} \|x^{(n)} - r^{(n)}\|^2
$$

- If $f$ and $g$ are linear

$$
\min_{W,V} \frac{1}{2N} \sum_{n=1}^{N} \|x^{(n)} - VWy^{(n)}\|^2
$$

- In other words, the optimal solution is PCA
Regularized Autoencoders

- Undercomplete autoencoders can also fail to learn anything useful if the encoder and decoder are given too much capacity e.g. nonlinearity.

- Regularized autoencoders use a loss function that encourages the model to have other properties besides the ability to copy its input to its output:
  - Sparsity of the representation
  - Robustness to noise or to missing inputs
  - Smallness of the derivative of the representation

Sparse Autoencoders

- A sparse autoencoder involves a sparsity penalty $\Omega(h)$ on the code layer $h$, in addition to the reconstruction error:

$$L \left( x, g( f(x)) \right) + \Omega(h)$$

where $g(h)$ is the decoder output and typically we have $h = f(x)$, the encoder output.

- Regularized maximum likelihood corresponds to maximizing $p(\theta \mid x)$, which is equivalent to maximizing

$$\log p(x \mid \theta) + \log p(\theta)$$. The log $p(x \mid \theta)$ term is the usual data log-likelihood term and the log $p(\theta)$ term, the log-prior over parameters, incorporates the preference over particular values of $\theta$. 
Sparse Autoencoders

• Regularized autoencoders defy such an interpretation because the regularizer depends on the data

• But, we still can think of the entire sparse autoencoder framework as approximating maximum likelihood training of a generative model that has latent variables.

• Suppose we have a model with visible variables x and latent variables h, with an explicit joint distribution:

\[ p_{model}(x,h) = p_{model}(h) \cdot p_{model}(x \mid h) \]

Sparse Autoencoders

• We refer to \( p_{model}(h) \) as the model’s prior distribution over the latent variables, representing the model’s beliefs prior to seeing x.

• Then the likelihood can be decomposed as:

\[ \log p_{model}(x) = \log \sum_h p_{model}(h, x) \]

• We can think of the autoencoder as approximating this sum with a point estimate for just one highly likely value for h.
Sparse Autoencoders

- From this point of view, with this chosen $h$, we maximize
  $$\log p_{\text{model}}(h, x) = \log p_{\text{model}}(h) + \log p_{\text{model}}(x|h)$$
  - The $\log p_{\text{model}}(h)$ term can be sparsity-inducing. For example, the Laplace prior,
    $$p_{\text{model}}(h_i) = \frac{\lambda}{2} e^{-\lambda|h_i|}$$
  corresponds to an absolute value sparsity penalty. Expressing the log-prior as an absolute value penalty, we obtain
    $$\Omega(h) = \lambda \sum_i |h_i|$$

Sparse Autoencoders

- This view provides a different motivation for training an autoencoder: it is a way of approximately training a generative model.
  - It also provides a different reason for why the features learned by the autoencoder are useful: they describe the latent variables that explain the input.
Denoising Autoencoders

- The denoising autoencoder (DAE) is an autoencoder that receives a corrupted data point as input and is trained to predict the original, uncorrupted data point as its output.
- A denoising autoencoder or DAE minimizes

\[ L(x, g(f(\tilde{x}))) \]

where \( \tilde{x} \) is a copy of \( x \) that has been corrupted by some form of noise.

Denoising Autoencoders

- \( C(\tilde{x}|x) \) represents a conditional distribution over corrupted samples \( \tilde{x} \), given a data sample \( x \).
Denoising Autoencoders

• So long as the encoder is deterministic, the denoising autoencoder is a feedforward network minimizing the loss

\[ L = - \log p_{\text{decoder}}(x|h = f(\tilde{x})) \]

• We can view the DAE as performing stochastic gradient descent on the following expectation:

\[ - \mathbb{E}_{\tilde{x} \sim \hat{p}_{\text{data}}(x)} \mathbb{E}_{z \sim \mathcal{C}(\tilde{x}|x)} \log p_{\text{decoder}}(x|h = f(\tilde{x})) \]

where \( \hat{p}_{\text{data}}(x) \) is the training distribution.

Contractive Autoencoders

• The contractive autoencoder introduces an explicit regularizer on the code \( h = f(x) \), encouraging the derivatives of \( f \) to be as small as possible:

\[ L \left( x, g(f(x)) \right) + \Omega(h, x) \]

\[ \Omega(h, x) = \lambda \sum_i ||\nabla_x h_i||^2 \]

• The penalty \( \Omega(h) \) is the squared Frobenius norm (sum of squared elements) of the Jacobian matrix of partial derivatives associated with the encoder function.
**Contractive Autoencoders**

- The name *contractive* arises from the way that the CAE warps space. Specifically, because the CAE is trained to resist perturbations of its input, it is encouraged to map a neighborhood of input points to a smaller neighborhood of output points.

**Variational Autoencoders**

- Variational autoencoders are *generative models*.
- An common way of describing a neural network is an approximation of some function. However, they can also be thought of as a data structure that *holds information*.
- Assume a network with a few *deconvolution layers*.
- Set the input to be a vector of ones.
**Variational Autoencoders**

- Train the network to reduce the mean squared error between the deconvoluted image and the target image.
- The "data" for that image is now contained within the network's parameters.

![Diagram](image1.png)

**Variational Autoencoders**

- Use real vector (latent variable) to remember more images
- Choosing the latent variables randomly is a bad idea.
- In an autoencoder, we add in another component that takes in the original images and encodes them into vectors for us.

![Diagram](image2.png)
Variational Autoencoders

• To generate images, we add a constraint on the encoding network, that forces it to generate latent vectors that roughly follow some distribution $q(z \mid x)$, e.g. unit Gaussian.

• Generating new images is now easy:
  • Get the encoded feature
  • Based on it, sample a latent vector from the unit Gaussian
  • Pass it to the decoder

$L(q) = \mathbb{E}_{z \sim q(z|x)} \log p_{model}(z, x) + H(q(z|x))$

$= \mathbb{E}_{z \sim q(z|x)} \log p_{model}(x \mid z) + D_{KL}(q(z \mid x) \mid \mid p_{model}(z))$

$\leq \log p_{model}(x)$
Variational Autoencoders

• In first line, the first term is the joint log-likelihood of the visible and hidden variables. The second term is entropy of the approximate posterior, which encourages the variational posterior to place high probability mass on many z values that could have generated x, rather than collapsing to a single point estimate of the most likely value.

• In second, the first term is the reconstruction log-likelihood. The second term tries to make the approximate posterior distribution \( q(z \mid x) \) and the model prior \( p_{\text{model}}(z) \) approach each other.

Variational Autoencoders

• In order to optimize the KL divergence, we need to apply a simple reparameterization trick:
  • Instead of the encoder generating a vector of real values
  • It will generate a vector of means and a vector of standard deviations.
Training of Autoencoders

• Depth can exponentially reduce the computational cost of representing some functions.
• Depth can also exponentially decrease the amount of training data needed to learn some functions.
• A common strategy for training a deep autoencoder is to greedily pretrain the deep architecture by training a stack of shallow autoencoders.

Software for Autoencoders

- Tensorflow
- Caffe
- Torch
- MXNet
- Keras
- Theano
- CNTK
- Chainer
Example for Autoencoders

• MNIST dataset overview
  • 60,000 examples for training
  • 10,000 examples for testing
  • Size-normalized digits
  • Centered in a fixed-size image (28x28 pixels)
  • Values from 0 to 1

```
# tf Graph input (only pictures)
X = tf.placeholder("float", [None, num_input])

weights = {
  'encoder_h1': tf.Variable(tf.random_normal([num_input, num_hidden_1])),
  'encoder_h2': tf.Variable(tf.random_normal([num_hidden_1, num_hidden_2])),
  'decoder_h1': tf.Variable(tf.random_normal([num_hidden_2, num_hidden_1])),
  'decoder_h2': tf.Variable(tf.random_normal([num_hidden_1, num_input])),
}

biases = {
  'encoder_b1': tf.Variable(tf.random_normal([num_hidden_1])),
  'encoder_b2': tf.Variable(tf.random_normal([num_hidden_2])),
  'decoder_b1': tf.Variable(tf.random_normal([num_hidden_1])),
  'decoder_b2': tf.Variable(tf.random_normal([num_input])),
}
```
Example for Autoencoders

• Define network structure

```python
# Building the encoder
def encoder(X):
    # Encoder Hidden layer with sigmoid activation #1
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['encoder_h1']),
                                  biases['encoder_b1']))
    # Encoder Hidden layer with sigmoid activation #2
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['encoder_h2']),
                                    biases['encoder_b2']))
    return layer_2

# Building the decoder
def decoder(X):
    # Decoder Hidden layer with sigmoid activation #1
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['decoder_h1']),
                                    biases['decoder_b1']))
    # Decoder Hidden layer with sigmoid activation #2
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['decoder_h2']),
                                    biases['decoder_b2']))
    return layer_2

# Construct model
encoder_op = encoder(X)
decoder_op = decoder(encoder_op)
```

Example for Autoencoders

• Define loss and optimizer

```python
# Prediction
y_pred = decoder_op
# Targets (Labels) are the input data.
y_true = X

# Define loss and optimizer, minimize the squared error
loss = tf.reduce_mean(tf.pow(y_true - y_pred, 2))
optimizer = tf.train.RMSPropOptimizer(learning_rate).minimize(loss)

# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
```
Example for Autoencoders

• Training

```python
# Start Training
# Start a new TF session
sess = tf.Session()

# Run the initializer
sess.run(init)

# Training
for i in range(1, num_steps+1):
    # Prepare Data
    # Get the next batch of MNIST data (only images are needed, not labels)
    batch_x, _ = mnist.train.next_batch(batch_size)

    # Run optimization op (backprop) and cost op (to get loss value)
    _, l = sess.run([optimizer, loss], feed_dict={X: batch_x})

    # Display logs per step
    if i % display_step == 0 or i == 1:
        print('Step %s Minibatch Loss: %f' % (i, l))
```

Example for Autoencoders

• Result

![Original Images](image1.png)

![Reconstructed Images](image2.png)
References

- CS2750 Spring 2015 Lecture 20 slides
- CS3750 Fall 2014 Lecture 9 slides
- A Tutorial on PCA by Jonathon Shlens
- A Tutorial on PCA by Lindsay I Smith
- http://kvfrans.com/variational-autoencoders-explained/
- https://github.com/aymericdamien/TensorFlow-Examples/