Machine Learning Mastery
Making Developers Awesome at Machine Learning

A Gentle Introduction to Matrix Factorization for Machine Learning

by **Jason Brownlee** on August 9, 2019 in **Linear Algebra**

Many complex matrix operations cannot be solved efficiently or with stability using the limited precision of computers.

Matrix decompositions are methods that reduce a matrix into constituent parts that make it easier to calculate more complex matrix operations. Matrix decomposition methods, also called matrix factorization methods, are a foundation of linear algebra in computers, even for basic operations such as solving systems of linear equations, calculating the inverse, and calculating the determinant of a matrix.

In this tutorial, you will discover matrix decompositions and how to calculate them in Python.

After completing this tutorial, you will know:

- What a matrix decomposition is and why these types of operations are important.
- How to calculate an LU andQR matrix decompositions in Python.
- How to calculate a Cholesky matrix decomposition in Python.

Kick-start your project with my new book Linear Algebra for Machine Learning, including *step-by-step tutorials* and the *Python source code* files for all examples.

Let's get started.

- **Update Mar/2018**: Fixed small typo in the description of QR Decomposition.
- **Update Jul/2019**: Fixed a small typo when describing positive definite matrices.

A Gentle Introduction to Matrix Decompositions for Machine Learning
Photo by mickey, some rights reserved.

Tutorial Overview

This tutorial is divided into 4 parts; they are:

1. What is a Matrix Decomposition?
2. LU Matrix Decomposition
3. QR Matrix Decomposition
4. Cholesky Decomposition

What is a Matrix Decomposition?

A matrix decomposition is a way of reducing a matrix into its constituent parts.

It is an approach that can simplify more complex matrix operations that can be performed on the decomposed matrix rather than on the original matrix itself.

A common analogy for matrix decomposition is the factoring of numbers, such as the factoring of 10 into 2 x 5. For this reason, matrix decomposition is also called matrix factorization. Like factoring real values, there are many ways to decompose a matrix, hence there are a range of different matrix decomposition techniques.

Two simple and widely used matrix decomposition methods are the LU matrix decomposition and the QR matrix decomposition.

Next, we will take a closer look at each of these methods.

LU Matrix Decomposition

The LU decomposition is for square matrices and decomposes a matrix into L and U components.

```
A = L . U
```

Or, without the dot notation.

```
A = LU
```

Where A is the square matrix that we wish to decompose, L is the lower triangle matrix and U is the upper triangle matrix.

> *The factors L and U are triangular matrices. The factorization that comes from elimination is A = LU.*

— Page 97, Introduction to Linear Algebra, Fifth Edition, 2016.

The LU decomposition is found using an iterative numerical process and can fail for those matrices that cannot be decomposed or decomposed easily.

A variation of this decomposition that is numerically more stable to solve in practice is called the LUP decomposition, or the LU decomposition with partial pivoting.

```
A = P . L . U
```

The rows of the parent matrix are re-ordered to simplify the decomposition process and the additional P matrix specifies a way to permute the result or return the result to the original order. There are also other variations of the LU.

The LU decomposition is often used to simplify the solving of systems of linear equations, such as finding the coefficients in a linear regression, as well as in calculating the determinant and inverse of a matrix.

The LU decomposition can be implemented in Python with the lu() function. More specifically, this function calculates an LPU decomposition.

The example below first defines a 3×3 square matrix. The LU decomposition is calculated, then the original matrix is reconstructed from the components.

```
# LU decomposition
from numpy import array
from scipy.linalg import lu
# define a square matrix
A = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(A)
# LU decomposition
P, L, U = lu(A)
print(P)
print(L)
print(U)
# reconstruct
B = P.dot(L).dot(U)
print(B)
```

Running the example first prints the defined 3×3 matrix, then the P, L, and U components of the decomposition, then finally the original matrix is reconstructed.

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]

[[ 0.  1.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]]

[[ 1.          0.          0.        ]
 [ 0.14285714  1.          0.        ]
 [ 0.57142857  0.5         1.        ]]

[[  7.00000000e+00   8.00000000e+00   9.00000000e+00]
 [  0.00000000e+00   8.57142857e-01   1.71428571e+00]
 [  0.00000000e+00   0.00000000e+00  -1.58603289e-16]]

[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
```

QR Matrix Decomposition

The QR decomposition is for m x n matrices (not limited to square matrices) and decomposes a matrix into Q and R components.

```
A = Q . R
```

Or, without the dot notation.

```
A = QR
```

Where A is the matrix that we wish to decompose, Q a matrix with the size m x m, and R is an upper triangle matrix with the size m x n.

The QR decomposition is found using an iterative numerical method that can fail for those matrices that cannot be decomposed, or decomposed easily.

Like the LU decomposition, the QR decomposition is often used to solve systems of linear equations, although is not limited to square matrices.

The QR decomposition can be implemented in NumPy using the qr() function. By default, the function returns the Q and R matrices with smaller or ‘reduced’ dimensions that is more economical. We can change this to return the expected sizes of m x m for Q and m x n for R by specifying the mode argument as ‘complete’, although this is not required for most applications.

The example below defines a 3×2 matrix, calculates the QR decomposition, then reconstructs the original matrix from the decomposed elements.

```
# QR decomposition
from numpy import array
from numpy.linalg import qr
# define a 3x2 matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
# QR decomposition
Q, R = qr(A, 'complete')
print(Q)
print(R)
# reconstruct
B = Q.dot(R)
```

```
print(B)
```

Running the example first prints the defined 3×2 matrix, then the Q and R elements, then finally the reconstructed matrix that matches what we started with.

```
[[1 2]
 [3 4]
 [5 6]]

[[-0.16903085  0.89708523  0.40824829]
 [-0.50709255  0.27602622 -0.81649658]
 [-0.84515425 -0.34503278  0.40824829]]

[[-5.91607978 -7.43735744]
 [ 0.          0.82807867]
 [ 0.          0.        ]]

[[ 1.  2.]
 [ 3.  4.]
 [ 5.  6.]]
```

Cholesky Decomposition

The Cholesky decomposition is for square symmetric matrices where all eigenvalues are greater than zero, so-called positive definite matrices.

For our interests in machine learning, we will focus on the Cholesky decomposition for real-valued matrices and ignore the cases when working with complex numbers.

The decomposition is defined as follows:

```
A = L . L^T
```

Or without the dot notation:

```
A = LL^T
```

Where A is the matrix being decomposed, L is the lower triangular matrix and L^T is the transpose of L.

The decompose can also be written as the product of the upper triangular matrix, for example:

```
A = U^T . U
```

Where U is the upper triangular matrix.

The Cholesky decomposition is used for solving linear least squares for linear regression, as well as simulation and optimization methods.

When decomposing symmetric matrices, the Cholesky decomposition is nearly twice as efficient as the LU decomposition and should be preferred in these cases.

> *While symmetric, positive definite matrices are rather special, they occur quite frequently in some applications, so their special factorization, called Cholesky decomposition, is good to know about. When you can use it, Cholesky decomposition is about a factor of two faster than alternative methods for solving linear equations.*

— Page 100, Numerical Recipes: The Art of Scientific Computing, Third Edition, 2007.

The Cholesky decomposition can be implemented in NumPy by calling the cholesky() function. The function only returns L as we can easily access the L transpose as needed.

The example below defines a 3×3 symmetric and positive definite matrix and calculates the Cholesky decomposition, then the original matrix is reconstructed.

```
# Cholesky decomposition
from numpy import array
from numpy.linalg import cholesky
# define a 3x3 matrix
A = array([[2, 1, 1], [1, 2, 1], [1, 1, 2]])
print(A)
# Cholesky decomposition
L = cholesky(A)
print(L)
# reconstruct
B = L.dot(L.T)
print(B)
```

Running the example first prints the symmetric matrix, then the lower triangular matrix from the decomposition followed by the reconstructed matrix.

```
[[2 1 1]
 [1 2 1]
 [1 1 2]]

[[ 1.41421356  0.          0.        ]
 [ 0.70710678  1.22474487  0.        ]
 [ 0.70710678  0.40824829  1.15470054]]

[[ 2.  1.  1.]
 [ 1.  2.  1.]
 [ 1.  1.  2.]]
```

Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Create 5 examples using each operation with your own data.
- Search machine learning papers and find 1 example of each operation being used.

If you explore any of these extensions, I'd love to know.

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

Books

- Section 6.6 Matrix decompositions. No Bullshit Guide To Linear Algebra, 2017.
- Lecture 7 QR Factorization, Numerical Linear Algebra, 1997.
- Section 2.3 LU Decomposition and Its Applications, Numerical Recipes: The Art of Scientific Computing, Third Edition, 2007.
- Section 2.10 QR Decomposition, Numerical Recipes: The Art of Scientific Computing, Third Edition, 2007.
- Section 2.9 Cholesky Decomposition, Numerical Recipes: The Art of Scientific Computing, Third Edition, 2007.
- Lecture 23, Cholesky Decomposition, Numerical Linear Algebra, 1997.

API

- scipy.linalg.lu() API
- numpy.linalg.qr() API
- numpy.linalg.cholesky() API

Articles

- Matrix decomposition on Wikipedia
- LU decomposition on Wikipedia
- QR Decomposition on Wikipedia
- Cholesky decomposition on Wikipedia

Summary

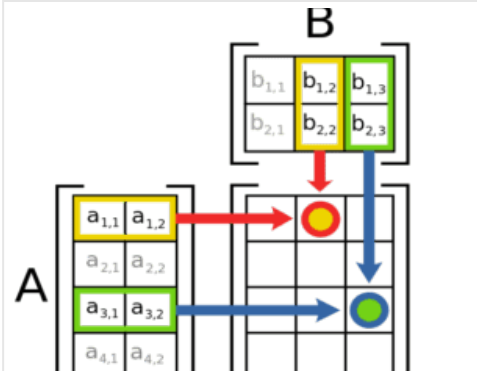
In this tutorial, you discovered matrix decompositions and how to calculate them in Python.

Specifically, you learned:

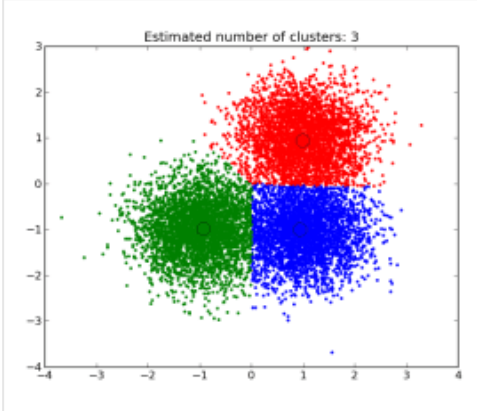
- What a matrix decomposition is and why these types of operations are important.
- How to calculate an LU andQR matrix decompositions in Python.
- How to calculate a Cholesky matrix decomposition in Python.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

More On This Topic



Matrix Types in Linear Algebra for Machine Learning



A Gentle Introduction to Scikit-Learn: A Python…

A Gentle Introduction to XGBoost for Applied Machine…

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.

View all posts by Jason Brownlee →

‹ A Gentle Introduction to Tensors for Machine Learning with NumPy

Gentle Introduction to Eigenvalues and Eigenvectors for Machine Learning ›

28 Responses to *A Gentle Introduction to Matrix Factorization for Machine Learning*

Stephen February 16, 2018 at 6:07 am #

REPLY

Hi Jason, is there a typo for the Cholesky decomposition with upper triangular matrices?

I think “A = U^T . T” should be “A = U^T . U”

Jason Brownlee February 16, 2018 at 8:36 am #

REPLY

Thanks, fixed.

Vladimir February 16, 2018 at 10:24 pm #

REPLY

Hi, Jason, good point! Could you call Py API for Nonnegative Matrix Factorization (NMF) more effective and flexible as sklearn.decomposition.NMF()?

Jason Brownlee February 17, 2018 at 8:45 am #

REPLY

Nice, thanks.

Janez March 2, 2018 at 6:44 am #

REPLY

Hi, there is a mistake in the definition of positive definite matrices in the section about Cholesky decomposition:

“[…] square symmetric matrices where all values are greater than zero, so-called positive definite matrices.”

This is not exactly true: a matrix M will be positive definite if all eigenvalues are positive; or by definition, if for all vectors z: z’ * M * z > 0. Since z can be expressed as a linear combination of the eigenvectors of M, the two statements are equivalent. It’s also worth mentioning that Cholesky decomposition will exist iff M is a positive definite matrix.

An counterexample to the statement from the article is a matrix A=[1 2; 2 1]. Since det(A) = -3 and the determinant is a product of the eigenvalues of A, there must exist a eigenvalue with a value < 0.

Jason Brownlee March 2, 2018 at 3:16 pm #

REPLY

Thanks for the clarification Janez, much appreciated!

Joe March 2, 2018 at 6:34 pm #

REPLY

Jason,

Good article. Thanks for sharing. One suggestion… I think this article would be more helpful if you gave a little more insight to the WHY for these factoizations. For example, talking about the orthogonality of Q in the QR factorization is an important property that can be exploited (e.g. projections on to constraint manifolds and the like).

Jason Brownlee March 3, 2018 at 8:08 am #

REPLY

Great suggestion Joe, thanks.

Salman March 14, 2018 at 1:45 am #

REPLY

In the first line of the QR decomposition section, I think it should be for “m X n” matrices since Q is “m X m” and R is “m X n”

REPLY

Jason Brownlee March 14, 2018 at 6:29 am #

Thanks, I’ll add it to Trello and look into it.

REPLY

Ednydecomp April 25, 2018 at 9:10 am #

There’s plenty more to this than meet your eyes… Dataless data, for example…see gov that stole my work… City, state, Feds, military…

REPLY

Jason Brownlee April 26, 2018 at 6:18 am #

What is dataless data?

REPLY

Geoffrey Anderson July 26, 2018 at 3:09 am #

I made a tensorflow matrix factorization model for a recommender on big dataset from reddit.com. The gradient descent via the adam optimizer worked pretty well. The problem is the code does a sparse to dense conversion and of course that’s going to limit the capacity. Soon I will find out what that capacity actually is. Maybe the dense conversion can be completely avoided and remain all sparse, not sure. One great thing is that it converges fast, and correctly handles missing data which is a big deal in recommenders especially since 99.99 percent sparsity is evident in my application. Garbage in = garbage out. It solves missing data problem by using a cost function which ignores the missing data. In some recommenders there is no way to preimpute except by making huge misleading assumptions so you’d definitely need to handle missing data correctly without assumptions in certain recommenders. The model is pq + user_bias + item_bias + regularization (L2). pq are the factor matrices. The bias terms are learned not constant or precomputed. TF is a pretty great solver for factorization models becuase it uses any number of GPU cards you have.

Anyway I am moving to a deep neural network model for better capacity. Its based on the semantic hashing design described by Hinton, where there is a binary latent vector in the middle. It is seemingly near impossible however to get Keras to make me a true binary vector, which is a big problem, a showstopper in this case. I experimented with Keras for days and nothing works. I might have to move to all tensorflow which would be a bummer but maybe TF has the power to make a binary latent vector.

REPLY

Jason Brownlee July 26, 2018 at 7:46 am #

Great.

REPLY

Karan Verma January 9, 2019 at 4:10 am #

Hey, I'm also working along the same lines for building a recommender system using MovieLens dataset. I'm trying to build a naive recommender system using latent factor model for MovieLens dataset. From the observed set of ratings I'm trying to build a model which will decompose the sparse matrix to N * K and K * M, where N is the number of users, M is the number of Movies and K is the number of dimensions in the latent space that I'm trying to learn. But I'm having trouble deciding the regularization term in the loss function. Here is the precise problem statement that I posted on stack exchange: https://datascience.stackexchange.com/questions/43497/regularization-term-in-matrix-factorization

REPLY

Jason Brownlee January 9, 2019 at 8:48 am #

Sorry, I don't have tutorials on recommender systems. Perhaps in the future.

REPLY

Geoffrey Anderson July 26, 2018 at 3:18 am #

In some applciations there is no missing data. In other applications there is just a bit of missing data. Yet other applciations the missing data is the majority such as in implicit feedback recommenders. Great care must be used in selecting a factorization that handles missing data appropriately for the application or else garbage can dominate your model and it's worthless really. For example singular value decomposition is perfectly fine when there are no missing values. However SVD cannot run correctly on a ratings recommender where missing values are the majority of the available ratings matrix at a point in time — it will run but the results are nonsense if you assume star ratings are zero when missing. In this case a great design solution I worked out is a gradient based recommender where missing data are simply contributing zero loss to the optimizer. I made such a system in tensorflow and it works great.

The bottom line is, don't put garbage data into your model and expect it to learn something useful, because it won't. SVD or QR and so on can be perfectly fine but it is strongly application depedent whether it's sensible to use the matrix factorization algorithms that cannot handle missing data as such.

REPLY

Jason Brownlee July 26, 2018 at 7:47 am #

Great tip Geoffrey.

REPLY

ENOCH MOGENDI April 10, 2019 at 6:55 pm #

This is a good one,
Would you do another one on Low Rank Matrix Decomposition

REPLY

Jason Brownlee April 11, 2019 at 6:32 am #

Great suggestion, thanks!

REPLY

Sean O'Connor April 11, 2020 at 6:19 pm #

ReLU is just a switch.
On=identity function=connect.
Off=zero=disconnect.
For a particular input vector to a ReLU net all the switch states are decided.
The dot product of a number of dot products is still a dot product.
Therefore the network output for that particular input vector can be condensed to a single matrix operating on the input vector.
I suppose you can factorized that matrix and get some metrics out of it.
Anyway conventional neural networks seem to calculate that matrix in a painful way, maybe they should be called Convoluted Neural Networks.

REPLY

Jason Brownlee April 12, 2020 at 6:16 am #

Thanks for sharing your thoughts Sean.

REPLY

rose March 5, 2021 at 6:47 pm #

hello dear
Solve a linear system of equations in the form Mx = b, find vector X

, using QR factorization.

by a linear algebra the rule.

X=R^-1 *Q^T *b
by this rule you find vector X

but I wont by python

REPLY

Jason Brownlee March 6, 2021 at 5:15 am #

You can use the QR factorization above and combine it with the example in this post:
https://machinelearningmastery.com/solve-linear-regression-using-linear-algebra/

Sorry, I don't have the capacity to implement it for you.

REPLY

rose March 5, 2021 at 8:07 pm #

Diagonalize a matrix M, such that it can be written in the form D = P −1
MP,

where D is diagonal.

REPLY

Jason Brownlee March 6, 2021 at 5:16 am #

This can help you calculate the diagonal:
https://machinelearningmastery.com/introduction-to-types-of-matrices-in-linear-algebra/

david oluyole ajekigbe May 5, 2021 at 8:34 am #

REPLY

good work but is it possible you use R instead of python. i am a student of R.
thank you.

Jason Brownlee May 6, 2021 at 5:41 am #

REPLY

Thanks for the suggestion.

Leave a Reply

Name (required)

Email (will not be published) (required)

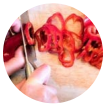
SUBMIT COMMENT

Welcome!
I'm *Jason Brownlee* PhD
and I **help developers** get results with **machine learning**.
Read more

Never miss a tutorial:

Picked for you:


Linear Algebra for Machine Learning (7-Day Mini-Course)


How to Index, Slice and Reshape NumPy Arrays for Machine Learning


How to Calculate Principal Component Analysis (PCA) from Scratch in Python


A Gentle Introduction to Sparse Matrices for Machine Learning


How to Calculate the SVD from Scratch with Python

Loving the Tutorials?

The Linear Algebra for Machine Learning EBook is where you'll find the ***Really Good*** stuff.

>> SEE WHAT'S INSIDE
