Nonlinear adaptive method of matrix completion on partial observations

Gorodnitskii Oleg
Moscow Institute of Physics and Technology
gorodnitskii@phystech.edu

Trofimov Mikhaill
Moscow Institute of Physics and Technology
mikhail.trofimov@phystech.edu

Abstract

Matrix completion is a problem of filling in missing entries of a partially observed matrix. Such problems arise in fields like recommendation systems and click-through rate prediction. There are several commonly used approaches to this problem, such as classical low-rank approximation (including Alternating Minimization) or probabilistic matrix factorization (PMF). All classical approaches approximate the entries of the matrix by a fixed representation function and focus only on learning latent (hidden) variables. In this article, we replace this fixed function with the parametrized one. That will allow us to tune both parametrization of the function and latent variables from data. For algorithm validation MovieLens dataset is used.

Index Terms — *Matrix Completion, Low-rank, Neural Networks, Deep Learning, Word Embedding*

1. Introduction

Getting a matrix approximation of a partial matrix of observations is one of the fundamental tasks in machine learning . As it is shown in [1] matrix completion has become one of the leading techniques for recommender systems, where one must handle with large datasets. **Low-rank** is a classical technique for the task of matrix completion. In this approach given matrix $\mathbf{M} \in \mathbf{R}^{n \times m}$ (we will refer to $\mathbf{M}$ as a *target matrix*) is considered as a matrix of a fixed rank k, which means it can be represented as a multiplication of two matrices $\mathbf{U} \in \mathbf{R}^{n \times k}$ and $\mathbf{V} \in \mathbf{R}^{k \times m}$. So each element of $\mathbf{M}$ is calculated as an inner product of a particular row of $\mathbf{U}$ and column of $\mathbf{V}$. The task is to find $\mathbf{U}, \mathbf{V}$, which minimize error over an observed part of a matrix. Low-rank approach has received many modifications after its invention and some of them are listed below.

Low-rank approach allows to reduce task of matrix completion to the optimization task of finding $\mathbf{U}, \mathbf{V}$. But this is a not convex problem, if optimization is carried out simultaneously on both matrices. The [1] describes a techinque of an **Alternating Minimization** (AM). In the AM the target matrix is written as a multiplication of two unknown matrices. Algorithm alternates between optimizing first matrix over fixed second and vise versa. As it is proved in [5] in each case optimization task is convex so it can be effectively solved using different optimization methods.

Because of a good scalability similar techique can be used in biology, where large datasets are common. According to [2] in gene expression analysis the problem of matrix completion is crucial, because many algorithms require a complete matrix of gene array values as an input. Troyanskaya [2] describes an application of an **Iterative SVD** – specially modified singular value decomposition, in a task of completion of DNA microarrays.

In [3] Salakhutdinov suggests different extension of the low-rank model via probabilistic approach. Article represents a **Probabilistic Matrix Factorization** (PMF) technique. In this approach it is assumed that all enteries of the target matrix are independent gaussians with common variance and means given by multiplication of two unknown matrices of fixed rank. PMF can be easly imporved by adding systematic biases for each row and column and one global bias, as it shown in [5].

Another approach, which extends an idea of the low-rank approximation, described in [7].Article represents an idea of **Local Low-rank** approximation. Instead of assuming that the target matrix has a low rank globally it is assumed that it behaves as a low-rank matrix in the locality of certain combinations of rows and columns. Therefore several low-rank approximations are constructed, each being accurate in a particular region of the matrix. According to [8] local low-rank modelling significantly outperformed global low-rank modeling in the context of recommender systems

All mentioned models used matrix factorization through representations of rows and columns. In [6] a significantly different model of **Co-clustering** is proposed. Its main idea can be described in terms of famous Netflix Prize competition. Traditional matrix factorization assumes that a movie preference is based on a weighted sum of preferences for different genres, with the movie properities being represented in a vectorial form. Co-clustering, on the other hand, assumes there exists some 'correct' partition of movies and users on groups. Objects in a same group share similar attributes.

In all low-rank-based approaches a fixed function (also called a "representation function") of two vectors is used to obtain elements of the target matrix. It is selected by an experimenter before an experiment. We consider this funcion as a dependent on some parameters which are determined in a process of learning. Explicit parametrization of the representation function can be expressed as a neural network. So one obtains optimal parameters of the function through neural network learning. This can lead to a better adaptation of the method to a particular task and impove the quality of recovery.

2. Problem Statement

Let us give some definitions:

Representation $\mathbf{w} \in \mathbf{R}^k$ *– is a vector of variables (sometimes called "latent" variables) that describes one of the dimensions of the matrix. Each row and column has its own unqiue representation, associated with it. All representations, that correspond to the same dimension, have same length, value of which is determined by a researcher before the experiment. Representations that correspond to different dimension can have different lengths.*

Representation function $\mathbf{F} : \mathbf{R}^{k+l} \mapsto \mathbf{Y}$ *– is a function of two representations which provides a mapping from the space of representations to* $\mathbf{Y} \subset \mathbf{R}$

Epoch *– is a one full training cycle of a neural network on a training set. Epoch is finished when all samples from the training set were used to update weights of the neural network.*

Dense layer *– is a fully connected layer of a neural network, all neurons of which have connections with all neurons of a previous layer.*

Hidden layer *– is a dense layer, located between input and output layers.*

Depth *of a neural network – number of hidden layers in a neural network (excluding input and output layers)*

Batch *– is a subset of the training set. All batches share same size.*

Batch learning *– is a learning model in which a neural netowork is trained on a sequence of batches. During a single batch processing all parameters of the network stay constant and are updated only after whole batch is processed.*

Dropout *[13] – is a technique in the neural network learning which is in blocking outputs from some randomly chosen subset of neurons from input and hidden layers. Relative size of the blocked subset is determined by a dropout parameter p. In the case of batch learning, during processing of a single batch the blocked subset stays constant. As shown in [13] this technique helps to make a neural network more robust to the damage of the input and reduces overfitting.*

Flatten layer *– is a special layer of a neural network that flattens all its input data into one-dimensional vectors and concatenates them into a single vector. Flatten layer always outputs a single one-dimensional vector.*

Per-dimensional optimization method *– is an optimization method that uses different parameters for operations with different dimensions of a parameter vector. As an example, a per-dimensional optimization method can use different learning rates for different dimensions.*

An unknown matrix $\mathbf{M} \in R^{m \times n}$ and a set of its known elements (observations): $\mathbf{O} = \{(x_i, y_i)\}_{i=1}^{l}$, where $x_i \in [1,...,m] \times [1,...,n], y_i \in \mathbf{Y} \subset \mathbf{R}$ are given. It is required to find an approximation $\mathbf{A}$ of the matrix $\mathbf{M}$, that minimizes quadratic element-wise deviation on some $\mathbf{D}_{test} = \{(x_i, y_i)\}_{i=1}^{h}$:

$$A = \underset{A}{arg\,min} \sum_{(x_i,y_i) \in \mathbf{D}_{test}} (A_{x_i} - y_i)^2 \qquad (1)$$

As a main dataset we use MovieLens 100K dataset [10]. It is given in a form of set of (user-movie, rank) pairs: $\mathbf{D} = \{(x_i, y_i)\}_{i=1}^{l}$, where $x_i \in [1,...,m] \times [1,...,n]$, $y_i \in \{1,2,3,4,5\}$ correspond to user-movie pair and rating respectively. Dataset is divided into L train-test sections $\{\mathbf{D}_{train}^i, \mathbf{D}_{test}^i\}_{i=1}^{L}$. A loss of model is estimated as an averaged quadratic loss over all train-test sections

$$Q(A) = \frac{1}{L} \sum_{i=1}^{L} \sum_{(x_j,y_j) \in D_{test}^i} \frac{1}{|\mathbf{D}|_{test}} (A_{x_j} - y_j)^2 \qquad (2)$$

Each train-test section is obtained by splitting dataset randomly onto two parts, so that $|\mathbf{D}|_{train} = (L-1)|\mathbf{D}|_{test}$ and sets of users and movies of $\mathbf{D}_{test}$ were nested into corresponding sets of $\mathbf{D}_{train}$. Choice of the quadratic loss as the loss function is due to the fact that we predict rating, in other words, we solve the problem of regression and use a classical loss function for the regression problems. For the experiment with the MovieLens 100K dataset L is set to 5.

3. Basic Experiment

Main purpose of the basic experiment is to obtain results of the traditional (baseline) methods on the main dataset. Further, these results will be compared with the results of the method, presented in this artcle. Classical low-rank and Iterative SVD were chosen as traditional methods.

In the low-rank method target matrix $\mathbf{M}$ is considered as a matrix of a fixed rank k, i.e. it can be represented as a multiplication of two matrices $\mathbf{U} \in R^{n\times k}$ and $\mathbf{V} \in \mathbf{R}^{m\times k}$: $\mathbf{M} = \mathbf{U}\mathbf{V}^T$. In another words, each element of the resulting matrix is calculated as inner product of particular representations, related to the corresponding row and column of the $\mathbf{M}$. This leads to the optimization task of finding

$$\mathbf{U},\mathbf{V} = \underset{\mathbf{U},\mathbf{V}}{arg\,min} \sum_{(x_i,y_i)\in\mathbf{D}_{test}} (< u_{x_i^1}, v_{x_i^2} > - y_i)^2 + \\ l_1(\|u_{x_i^1}\|_1 + \|v_{x_i^2}\|_1) + l_2(\|u_{x_i^1}\|_2 + \|v_{x_i^2}\|_2) \quad (3)$$

where $u_{x_i^1}, v_{x_i^2}$ are rows of the $\mathbf{U}$ and $\mathbf{V}$ respectively. and l_1, l_2 are l_1, l_2 norms regularization parameters. For the optimization task solving "Adam" [11] algorithm from Downhill 0.32 package is used.

The Iterative SVD method can be described in a the following way(Algorithm 1): because SVD can only be performed over complete matrices, in the first step all missing values of $\mathbf{M}$ are replaced with the row averages. In such way a matrix $\mathbf{A^0}$ is obtained. After that, a truncated SVD [8] is performed over $\mathbf{A^0}$ and all values of $\mathbf{A^0}$, that were missed in $\mathbf{M}$, are replaced with the values obtained from the truncated SVD of $\mathbf{A^0}$. In each further step a procedure of truncated SVD and replacement of missing values is performed on a matrix, obtained in a previous step.

Main datatable [10] consists of 943 users with at least 20 rated movies among of 1682 total from Movielens 100K dataset. Users were selected at random for inclusion. Ratings are made on a 5-star scale. Zero rating denotes absence of vote.
Results of the low-rank method and Iterative SVD on first 20 ranks are presented in a table 1 and can be seen on graphics 4, 2. Vertical blue bars are representing standard deviation of the error over 5 folds.

4. Solution

In this section we provide a description for the new method (we will call it Neural Network Matrix Factorization or NNMF). As a low-rank method NNMF associates representations with each row and column of the target matrix. Representations for different dimensions can have different lengths. Each element of the matrix is considered as a value of representation function $\mathbf{F}$ at a corresponding representations $\mathbf{w_i}, \mathbf{w_j}$:

$$a_{i,j} = \mathbf{F}(\mathbf{w_i}, \mathbf{w_j}) \quad (4)$$

But, unlike in low-rank, function $\mathbf{F}$ is not fixed (for low-rank $\mathbf{F}(\mathbf{x},\mathbf{y}) =< \mathbf{x},\mathbf{y} >$). In NNMF approach $\mathbf{F}$ is considered as a nonlinear composition of linear functions. Arbitrary continuous function can be uniformly approximated (with any degree of accuracy) by superposition of add operation and nonlinear function of a single argument [9]. Therefore $\mathbf{F}$ can uniformly approximate arbitrary continious function. Techincally neural networks with nonlinear activation function are exactly a superposition of addition and nonlinear function of a single argument. We replace fixed $\mathbf{F}$ with a neural network. Because the neural network has a fixed structure, by choosing it we

Algoritm 1 Iterative SVD

```
Iterative SVD
In: $M$;
Out: $A$;
    $rank = k$; {Initialize rank}
    for $i = 1,\dots,n$
        for $j = 1,\dots,m$
            if $((i,j),M_{i,j}) \in \mathbf{O}$ then
                $A^0_{i,j} = M_{i,j}$
            else
                $A^0_{i,j} = \frac{1}{m}\sum_{k:((i,j),M_{i,j})\in\mathbf{O}} M_{i,k}$
    i = 0
    while $Q(A^i) > eps$
        $A^{i+1} = TruncatedSVD(A^i, rank = k)$
```

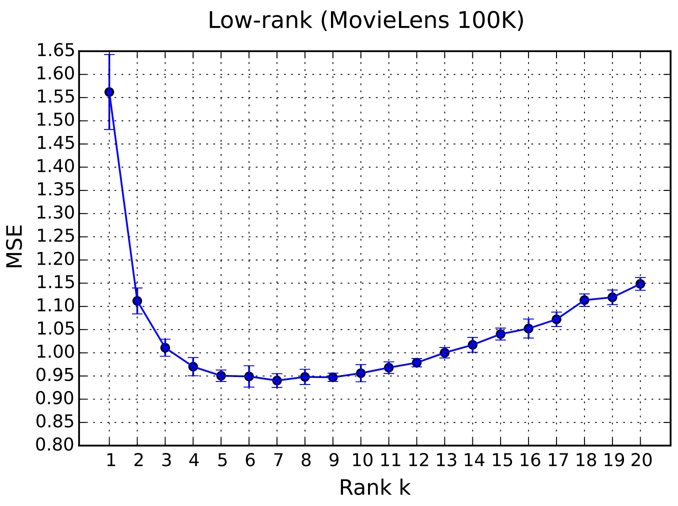

Fig. 1. Low-rank

Table 1. Dependence of MSE from rank

Rank	Matrix factorization	Iterative SVD
1	**1.562**	**0.910**
2	**1.111**	**0.895**
3	**1.010**	**0.931**
4	**0.970**	**0.996**
5	**0.950**	**1.059**
6	**0.949**	**1.146**
7	**0.940**	**1.258**
8	**0.948**	**1.369**
9	**0.947**	**1.559**
10	**0.956**	**1.675**
11	**0.968**	**1.740**
12	**0.978**	**1.886**
13	**1.000**	**1.996**
14	**1.017**	**2.107**
15	**1.040**	**2.216**
16	**1.052**	**2.321**
17	**1.072**	**2.490**
18	**1.113**	**2.573**
19	**1.119**	**2.693**
20	**1.148**	**2.773**

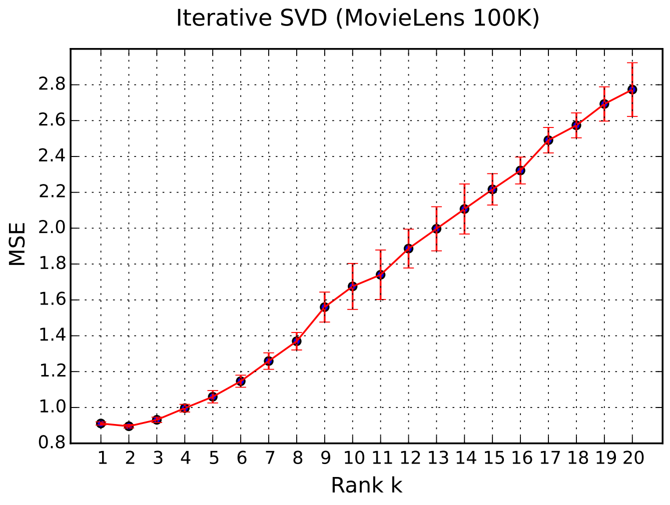

Fig. 2. ISVD

$$g(x_1, x_2, x_3) = f(w^2_{1,1} \times f(w^1_{1,1}x_1 + w^1_{2,1}x_2) + w^2_{2,1} \times f(w^1_{3,2}x_3))$$

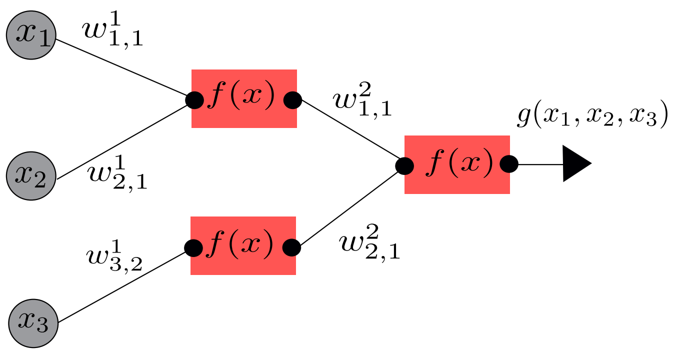

Fig. 3. Example of parametrization

choose some particular class of functions. During neural network training we are, actually, looking for a function, in the fixed class, that fits best to a particular task. At the same time representations for each row and column are learned. Simultaneous learning of the representations and the neural network allows to consider them as a single structure, which will be reflected at the technical implementation.

Model was implemented in Python language, using Theano framework [14], [15] and Lasagne library [18] – library for neural networks building and training. As an input, network gets indexes of a row and a column of an element to be calculated. Then indices are passed to an embedding layer. Word embedding is a technique, widely used in natural language processing (NLP) and a main part of Google's Word2Vec technology [17], [16]. This technique provides a mapping from a finite set of objects (words) **S** to continious space of vectors $\mathbf{f} : \mathbf{S} \to \mathbf{R}^n$. Technically it is implemented by matching between some finite set of numbers and a set of vectors, both sets sharing the same cardinality. Consider this cardinality as a **m** and a dimensionality of the vectors' space as a **n**. Then a patricular embedding can be written in a from of a matrix $\mathbf{W} \in \mathbf{R}^{m \times n}$, where to each number a particulr vector (row in **W**) is matched. We are using embedding layer's weights matrix as **W**, storing representations in it. For the two provided indexes of the row and column embedding layer will output two vectors - representations of the corresponding row and column. Weights of **W** are just a part of the neural network. That allows us to combine representations and neural network into a single structure and learn them from the data simultaneously.
Vectors from the embedding layer then pass into a flatten layer, which concatenates them and passes them as an input to a sequence of the dense layers. Exacly this sequence of the dense layers form main part of the neural network and its structure determines the class of functions in which the representation function **F** will be built. Last dense layer of the sequence outputs the answer – element of the matrix, that corresponds to the provided pair of indexes.

5. Primary experiment (Test on MovieLens 100K Data Set)

The main purpose of the primary experiment is to compare results of the NNMF method with the results

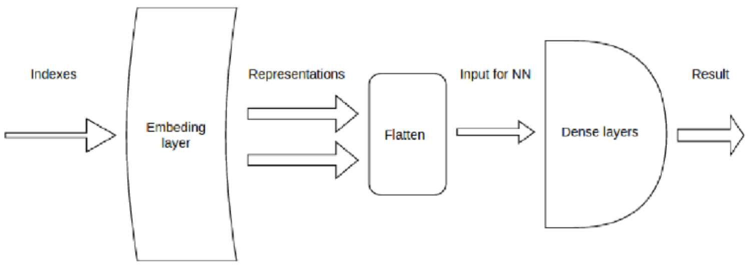

Fig. 4. Scheme of NN

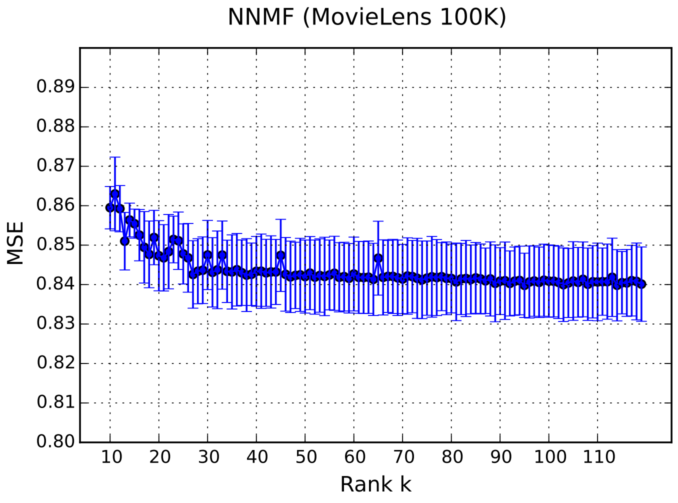

Fig. 5. NNMF

of the basic methods.
For the MovieLens 100K dataset, parameters of NNMF (parameters of network: depth, number of neurons in hidden layers, dropout, learning rate, regularization parameters, activation function and optimization method) were selected by brute force during cross-validation on 5 folds.

1. Depth : 3
2. Number of neurons in hidden layers: 256
3. Dropout parameter p: 0.3 [13]
4. Activation function: Rectifier
5. Optimization method: Adadelta [12]
6. Learning rate η: $8 * 10^{-3}$

Because the shape of the datatable in MovieLens 100K is close to square, all representations share same lenght. After the parameters were chosen, NNMF was tested on the dataset with the length of representations (equal to the rank k in low-rank interpretation) in a range from 80 to 120. As a stop criterion an early stopping has been chosen: it stops learning process after the epoch with number [***Stop epoch***] if an error after the epoch with the number [***Stop epoch - early stopping parameter***] was lower. The error after [***Stop epoch - early stopping parameter***] epoch is considered as an error of NNMF for the particular rank.

	Low Rank	Iterative SVD	NNMF
Best MSE	0.94	0.895	0.841
Rank	7	2	95
SD	0.02	0.01	0.01

6. Conclusion

This paper represented a Neural Network Matrix Factorization - a new method for solving the problem of Matrix Completion. The method showed good quality of solution within frames of MSE metrics and outperformed both baseline algorithms. Besides, method easily generalizes on arbitrary dimensionality.

Further researches are possible with usage of additional technics as compositions, convolution neural networks (CNN) and different quality metrics as nDCG, MAP, etc.

All materials needed for conducting an experiment can be found at `https://sourceforge.net/p/mlalgorithms/code/HEAD/tree/Group374/Gorodnitskii2016AdaptiveApproximation/`.

References

[1] Hsiang-Fu Yu, Cho-Jui Hsieh, Inderjit S. Dhillon, *Parallel Matrix Factorization for Recommender System*, 2013.
[2] Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein, Russ B. Altman, *Missing value estimation methods for DNA microarrays*, 2001.
[3] Salakhutdinov and A. Mnih, *Probabilistic matrix factorization*, 2008.
[4] Prateek J., Praneeth N., Sujay S., *Low-rank matrix completion using alternating minimization* 2012.
[5] Koren, R. Bell, and C. Volinsky., *Matrix factorization techniques for recommender systems* 2009.
[6] Beutel A., Ahmed A., Alexander J. Smola., *ACCAMS: Additive Co-Clustering to Approximate Matrices Succinctly* 2013.
[7] L. Joonseok, K. Seungyeon, L. Guy., *Local Low-Rank Matrix Approximation* 2013.
[8] Pet Christian Hansen, *The truncated SVD as a method for regularization* 1986.
[9] Gorban A.N., *Neuroinformatics* 1998.
[10] F. Maxwell Harper and Joseph A. Konstan, *The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19, 19 pages.* 2015.

[11] Diederik Kingma, Jimmy Ba, *Adam: A Method for Stochastic Optimization* 2014.

[12] Matthew D. Zeiler, *ADADELTA: An Adaptive Learning Rate Method* 2012.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, 2014.

[14] Bastien, Frédéric and Lamblin, Pascal and Pascanu, Razvan and Bergstra, James and Goodfellow, Ian J. and Bergeron, Arnaud and Bouchard, Nicolas and Bengio, Yoshua, *Theano: new features and speed improvements*, 2012.

[15] Bergstra, James and Breuleux, Olivier and Bastien, Frédéric and Lamblin, Pascal and Pascanu, Razvan and Desjardins, Guillaume and Turian, Joseph and Warde-Farley, David and Bengio, Yoshua, *Theano: a CPU and GPU Math Expression Compiler*, 2010.

[16] Omer Levy, Yoav Goldberg, *Neural Word Embedding as Implicit Matrix Factorization*, 2010.

[17] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, *Efficient Estimation of Word Representations in Vector Space*, 2010.

[18] Sander Dieleman and Jan Schlüter and Colin Raffel and Eben Olson and Søren Kaae Sønderby and Daniel Nouri and Daniel Maturana and Martin Thoma and Eric Battenberg and Jack Kelly and Jeffrey De Fauw and Michael Heilman and diogo149 and Brian McFee and Hendrik Weideman and takacsg84 and peterderivaz and Jon and instagibbs and Dr. Kashif Rasul and CongLiu and Britefury and Jonas Degrave, *Lasagne: First release.*, 2015.