Deep Convolutional NMF Net
A Neural Network with Non-negative Matrix Factorization (NMF) Layers

Hangyan Jiang∗1, Zexi Jin†1, Zhihao Zheng‡1, and Pengyu Hong§1

1Department of Computer Science, Brandeis University

Abstract

In this paper, we explored the possible buildup of a neural network based on Non-negative Matrix Factorization (NMF). We constructed a neural network with NMF layers that process input data in a convolutional manner. This ensures those novel layers first focus features locally and gradually expand their scope as they stack on each other. After testing we found that NMF preserves the original signal very well and has the potential to accelerate network.

1. Introduction

Non-negative Matrix Factorization (NMF) is a useful tool for high dimensional data feature extraction. Its restricting both factor matrices to be non-negative produces features that are relatively meaningful and mutual exclusive. However, the discussions about NMF so far has been focusing on the improvement of NMF itself while NMF is potentially a nice complement of existent deep learning algorithms. This paper aims to explore the possible neural network construction with NMF algorithm, propose solutions to the possible problems when shaping the architecture, and analyze the performance of the novel neural network.

2. Non-negative Matrix Factorization

The Non-negative Matrix Factorization applications proposed by Lee et al. [4] in 1999 has brought much attention. NMF has become a widely used tool in various tasks such as image feature extraction, documents clustering, audio signal processing, etc.

For an input matrix \( V \) with dimension \( m \times n \), NMF finds two non-negative matrix factors \( W \) and \( H \) whose dimensions are \( m \times k \) and \( k \times n \) such that

\[
V = WH
\]

Conventionally, \( W \) is named “features” and \( H \) ”coefficients”. To put both terms in a more intuitive way, suppose the input \( m \) by \( n \) matrix \( V \) is a set of images with each column as one vectorized image, then the \( m \) by \( k \) matrix \( W \) would be a set of \( k \) features that NMF extracts and the \( k \) by \( n \) matrix \( H \) would be the corresponding coefficients that is necessary for image recovery.

![Figure 1](image)

For a set of images \( V \), nmf extracts 2 sets of features \( (W) \) and together with corresponding coefficient matrix \( H \). With these two factor matrices, we can perfectly recover original images matrix \( V \).

2.1. NMF Algorithm

To measure the goodness of the factorization, it is necessary to define the cost function first. There are a few ways to formulate the cost function. Here we introduce two of them

\[
\min_{W,H \geq 0} \| V - WH \|^2 \tag{2}
\]

and

\[
\min_{W,H \geq 0} D(V||WH) \tag{3}
\]

where

\[
\| V - WH \|^2 = \sum_{i,j} (V_{ij} - WH_{ij})^2 \tag{4}
\]

and

\[
D(V||WH) = \sum_{i,j} \left( V_{ij} \log \frac{V_{ij}}{WH_{ij}} - V_{ij} + WH_{ij} \right) \tag{5}
\]
Lee et al. [5] made a detailed analysis on both of the formulas. Here, equation (4) is named Euclidean Distance and equation (5) is named Kullback-Leibler Divergence. Distance and divergence are both valid and the choice of cost function may depend on the specific task that NMF is working on.

To minimize the cost function, NMF performs gradient descent, updating \( W \) and \( H \), from randomized matrices, until cost function finds its global minimum point. So far, there has been a vast amount of literature discussing the optimization of the converging algorithm [1]. But in this paper, we are only interested in the validity of NMF neural network and the performance is not our prior concern.

Note, to ensure meaningful outputs, that is to make sure matrix \( W \) contains relatively local and sparse features, the parameter \( k \) should always be a positive integer that satisfies \( k(m+n) < mn \)

![Figure 2](image)

NMF applying on MNIST images extracts features into \( W \) and the picture reconstruction is close to the original image.

### 2.2. NMF Analysis

First, Non-negative Matrix Factorization is an unsupervised matrix decomposition algorithm. Therefore, it can always finds meaningful patterns from a given input once the parameter \( k \) is determined. This makes the algorithm intuitively understandable. We will talk about that in detail in the following section.

Second, both \( W \) and \( H \) contain information necessary for reconstructing original input \( V \). But the difference is that if we consider NMF in a mini-batch style (NMF actually does not support that, but we can reform the algorithm. A discussion about this is in the latter section) entire \( m \) by \( k \) matrix \( W \) is updated when processing each input batch. However, only corresponding columns of \( k \) by \( n \) matrix \( H \) will receive a new value. Now if we transform the NMF equation to be

\[
H \approx W^T \times V
\]

we will obtain a formula similar to linear layer equation in neural network where \( W^T \) is the set of parameters with \( V \) and \( H \) being input and output respectively of the current layer. This is a starting point of the NMF layer formulation.

### 3. Neural Network Construction

Equation (6) is not enough for a neural network. Since as mentioned above NMF is unsupervised, it is impossible to control what to extract. However, we may set up a scope constraints so that NMF has to search for features in the designed frames. After scanning through small-sized windows, we merge those windows so that the frames are expanded. This way, NMF is able to focus on local features in the beginning while a feature with parts distributed far from each other will less likely to appear. We name such algorithm “Merge-local-feature”

Another thing to notice is that unlike linear layers, activation functions such as ReLU and Sigmoid do not function well on equation (6) and there is no need to worry about a linearity trap will may face in linear layers. For ReLU, since NMF algorithm forces \( H \) to be non-negative, ReLU has already performed and reapplying it is simply unnecessary. Processing \( H \) with Sigmoid, however, hurts the contrast of coefficients and may loss the features of signals as neural network goes deeper.

### 3.1. Architecture

The architecture of NMF layered neural network is shown in figure 3. After processing inputs through multiple NMF layers, the network outputs a prediction by feeding the signals to a Fully Connected Layer (FC layer).

![Figure 3](image)

However, each NMF layer works slightly different from traditional layers in a few ways. First, while NMF layers have the same forward-propagation procedure, they do not require back-propagation to update parameters \( W^T \). That is even though NMF updates \( W \) with gradient descent, NMF layer will not be able to produce a gradient to feed backwards. Second, since for each window of input segment does not have many features, \( k \) should be reasonably small. This is also true when comparing to the size of hidden layer of traditional Neural Networks or the number of filters of Convolutional Neural Networks (CNN) [2]. For example, in MNIST [3] dataset, while each convolutional layer of CNN usually has a few dozens filters, NMF layers only needs \( k = 20 \) to produce good results. Besides, a convolutional layer
applies filters on the entire image but a NMF layer only requires $W$ on image fragments.

![Image 55x607 to 281x683](image)

Figure 4

We can also connects NMF layers with convolutional layers as Figure 4 illustrates. If flatten all the byproducts of Merge local feature algorithm, which we will introduce in the next subsection, into a single column, we can feed the result of the last NMF layer into the first convolutional layer and we will still receive a pretty reasonable prediction.

3.2. Merge Local Feature

Suppose the input data $V$ has dimension $q \times q \times n$ where $q^2 = m$ and $n$ is sample size. Merge local feature first cuts out the each sample by a $p$ by $p$ window in a convolutional manner where $p < q$, then flattens the matrix window into one dimensional vector. Now we have new data $V_{\text{new}}$ with dimension $p^2 \times (q - p + 1) \times (q - p + 1) \times n$. Then if we consider first dimension as a single dot, and once again apply the $p \times p$ window-cutting convolutionally followed by flattening, we have finished merge local feature. At this point, we will have a data with dimension $p^4 \times (q - 2p + 2) \times (q - 2p + 2) \times n$. Now if we reshape the data to be a $p^4 \times n(q - 2p + 2)^2$ matrix, we could factorize the matrix using NMF with a parameter $k$ which produces a $p^3$ by $k$ matrix $W_1$ and a $k$ by $n(q - 2p + 2)^2$ matrix $H_1$.

3.3. Connection Between NMF Layers

As Algorithm 1 and 2 illustrates, an $i^{th}$ NMF layer accepts a $k_i$ by $n[q - (i + 1)(p - 1)]^2$ matrix $H_i$, reshapes it to dimension $[q - (i + 1)(p - 1)] \times [q - (i + 1)(p - 1)] \times k_i n$, performs merge local feature and NMF algorithms, producing a $k_{i+1}$ by $n[q - (i + 2)(p - 1)]^2$ matrix $H_{i+1}$ fed to next layer. At the same time, $W_i$ is generated which stores all the features found in a $(ip - 1)$ by $(ip - 1)$ window scope.

As for forward-propagation, a NMF layer reshapes and merges local features as described above. Then it outputs

$$H_{i+1} = W_i^T \times H_i$$

4. Mini-Batch Problem

One of the important techniques Neural Networks use is Mini-Batch. This is essentially useful when input data has large sample size. Instead of passing entire data set into the memory, Mini-Batch defined a batch size $b$ and Neural Network trains itself with data sample with size $b$.

However, we have mentioned previously that original NMF does not support Mini-Batching. That means a NMF Neural Network requires NMF layers to process entire data set. This lacks efficiency and usually is not practical. For example, for MNIST[3] data set, there are 60,000 images in training set which most machines can not process at once.

Online Non-negative Matrix Factorization (ONMF) proposed by R. Zhao et al. [7] can be a possible solution to this problem.

4.1. Online Non-negative Matrix Factorization

To mini-batchalize NMF algorithm, ONMF has a different updating schedule[6]. While NMF updates $W$ and $H$ simultaneously in each descending circle, ONMF treats $W$ and $H$ differently. This makes sense since in NMF layers, $W$ represents a global view of data features and $H$ is closely related to each sample and local features, hence updating them in distinct manners should not cause the uneven distribution problem.

---

**Algorithm 1 NMF layer**

1. Input: $H_i, W_i$
2. $H_i \leftarrow$ merge-local-feature($H_i$)
3. $H_i \leftarrow$ reshape($H_i$)
4. if forward-propogation then
5. $H_{i+1} \leftarrow W_i^T H_i$
6. Output: $H_{i+1}$
7. End if
8. $W_i, H_{i+1} \leftarrow$ NMF($H_i$)
9. Output: $H_{i+1}$

**Algorithm 2 merge-local-feature**

1. Input: $H_i$
2. patchSize, $\sim$, dataLen, featLen $\leftarrow$ size($H_i$)
3. $\text{Result} \leftarrow$ zeros(dataLen(patchSize $- winSize + 1)+ (patchSize - winSize + 1), featLen(winSize)^2$
4. for idx in range dataLen do
5. for x in range patchSize-winSize+1 do
6. for y in range patchSize-winSize+1 do
7. $\text{Result}(1 + \text{idx}(patchSize $- winSize + 1)^2 + x(patchSize - winSize + 1)) \leftarrow$ reshape($H_i(x \sim x + \text{winSize}, y \sim y + \text{winSize}, \text{idx})$)
8. End for
9. End for
10. End for
11. Output: Result
Figure 5 is an illustration of ONMF algorithm. Each time ONMF seeks to update \(W_{t-1}\) to \(W_t\), it first calculates \(h_t\) first from randomized matrix \(h_0\), then using \(h_t\) and data batch the algorithm is able to update \(W_t\).

Algorithm 3 ONMF

1. **Input**: data batch \(v_t\), feature matrix \(W_{t-1}\)
2. \(h_t \leftarrow learnH(v_t, W_{t-1})\)
3. Update \(W_{t-1}\) to \(W_t\) by NMF updating rule such that it minimizes cost function

\[
\min_{W_{t-1}} \| v_t - W_{t-1} h_t \|^2 \quad (7)
\]

4: or

\[
\min_{W_{t-1}} D(v_t | | W_{t-1} h_t) \quad (8)
\]

5: given \(h_t\) and \(v_t\)
6: End for
7: **Output**: \(W_t\)

Algorithm 4 learnH

1. **Input**: data batch \(v_t\), feature matrix \(W_{t-1}\)
2. Randomize \(h_0\)
3. for \(idx\) in range \(s\) where \(s\) is a hyper-parameter do
4. Update \(h_{idx}^{t}\) from \(h_{idx}^{t-1}\) by NMF updating rule such that it minimizes cost function

\[
\min_{h_{idx}^{t-1}} \| v_t - W_{t-1} h_{idx}^{t-1} \|^2 \quad (9)
\]

5: or

\[
\min_{h_{idx}^{t-1}} D(v_t | | W_{t-1} h_{idx}^{t-1}) \quad (10)
\]

6: given \(W_{t-1}\) and \(v_t\)
7: End for
8: **Output**: \(h_{s}\)

4.2. ONMF Algorithm

Algorithm 3 and 4 shows the pseudocode of ONMF. Note each \(W_t\) update require a new update of \(h_t\) from scratch, but \(W_t\) should not be initialized since it stores features extracted from other batches. Initializing a new \(W_t\) will discard previous features and result in a loss of information. We do not need to worry about \(h_t\) initialization since each \(h_t\) are independent with each other and only stores the information of current batch. In practice, each \(h_t\) updates in algorithm 4 requires \(s\) to be more than 100 times but for \(W_t\) updates one update is enough.

**Step size** of the NMF updating algorithm is hard to choose. R. Zhao et al. [7] chose their step size \(\eta\) satisfying

\[
\sum \eta = \infty
\]

and

\[
\sum \eta^2 < \infty
\]

However, in this paper we bypassed step size choose by using the updating rule proposed by Lee et al. [5]. According to Lee and Seung, instead of using a traditional additive updating algorithm we can set step size dynamically and transform it updating rule to be multiplicative. This preserves the convergence of gradient descent.

For example, for Euclidean Distance function shown in equation (2) and equation (4), we have additive updating rule for \(H\)

\[
H_{ij} = H_{ij} + \eta_{ij} \left[ (W^T V)_{ij} - (W^T W H)_{ij} \right] \quad (11)
\]

If we set

\[
\eta_{ij} = \frac{H_{ij}}{(W^T W H)_{ij}} \quad (12)
\]

then we have

\[
H_{ij} = H_{ij} \left( \frac{(W^T V)_{ij}}{(W^T W H)_{ij}} \right) \quad (13)
\]

Similarly for \(W\), we have multiplicative updating rule

\[
W_{ip} = W_{ip} \left( \frac{V H^T)_{ip}}{(W H H^T)^{ip}} \right) \quad (14)
\]

Figure 6 illustrates a test of NMF and ONMF on MNIST\(^3\) images with \(k = 15, m = 784, n = 2000, batchSize = 50\). Here, we used MATLAB incorporated NMF package nnmf and as for ONMF we used multiplicative updating rule on Euclidean Distance cost function.
One thing to note is that ONMF the recovery tends to have a better recovery image comparing to traditional NMF. This may because the special $W$ updating mechanism of ONMF does not overfit certain classes of images due to a one update per batch pattern. However, traditional NMF emphasizes every class of images which leads final feature matrix $W$ overfits all the classes and there would be no way to avoid blur when reconstructing original data due to unclear information stored in $W$.

### 4.3. Mini-Batch Deep NMF Neural Network

Now we are ready to make NMF Neural Network to support mini-batch by replacing all the NMF with ONMF algorithm. The Deep NMF Neural Network is then fully built.

For each batch of input, the network updates $W$s in NMF layers with ONMF while applies back-propagation when updating parameters in other layers. Note, since we do not want other layers to connect NMF layers during back-propagation, NMF layers should always comes before other layers to avoid a broke in the back-propagating chain. As a matter of fact, it only makes sense if NMF layers comes first in that after linear or convolutional layers, NMF layers' sparse feature distraction becomes meaningless.

### 5. Test on MNIST

We tested the the image classification of NMF Neural Networks with architectures illustrated in figure 3 and figure 4. Neural Networks are tested on MNIST\[3\] data set with classes 10, training set size 60,000, testing set size 10,000, batch size 128, and epoch 10. Each image has dimension $28 \times 28$ with 1 color channel. $W$ of each NMF layers are updated by 1,000 images.

#### 5.1. Test Result

We tested a pure NMF Neural Network architecture with 9 NMF layers followed by a fully connected layer and a hybrid Neural Network with 9 NMF layers followed by 2 layers of convolutional layers and a fully connected layer. For comparison, we run Convolutional Neural Network with the same setting i.e. 2 conv layers and 1 FC layer.

The testing result is shown in figure 7. According to average accuracy each Neural Network produced, NMF Neural Networks performs generally worse than CNN and a hybrid network tends to significantly improve the accuracy.

However, an average accuracy of 84% indicates that NMF layer does preserve the information necessary for image classification. Considering $W$ only contains features from 1,000 images, feeding $W$ with more training images will dramatically improve the result of the Deep NMF Neural Networks.

To testing on the running time of Neural Networks above, we ran them with Pytorch framework on a machine with Ubuntu 16.04 LTS Operating System, Nvidia 970m 3GB GPU, 8GB RAM, Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz. Figure 8 shows the average time each Network finish training on MNIST data set.

![Figure 8](image)

Deep Neural Network tends to take less time training the model while CNN took longest time finish training. However, as mentioned above, since only 1,000 images are used to train NMF layers, to enhance the accuracy of the NMF Neural Network, running time will increase in a large scale.

#### 5.2. Test Analysis

One of the most exciting result we found is that NMF layers boost the training efficiency of a Neural Network. However, this comes with a deduced accuracy as a price. In this section, we try to dissect the NMF layer and search for possible explanation for test results.

As mentioned before, NMF layers generates fewer parameters $W$ (as for NMF algorithm it is called features). This reduces the the number of convolutional calculating circle. It is true that merge-local-feature algorithm produces more image fragments for NMF calculation. However, window-cutting also reduces the image size for convolution algorithm to perform. Conclude from practice, image size reducing affects the performance of a layer more significantly than new samples producing.

From the result in figure 7, convolutional layer did a much better job processing input signals for classification than NMF layers did. This might because NMF layers only feed partial data to the next layer while a conv layer leaves no information behind when passing signla to its next layer. In other words, NMF layers did a relatively poor job preserving all the information from input data. This seems inevitable since coefficient matrix $H$, in theory, does not contain any information about specific features but only the pattern certain features appear in the original data. Passing part of the $W$ or accessing $W$ from next layer may be
a promising approach to enhance the connectivity between NMF layers.

6. Conclusion

Based on the test result presented above, we can conclude that Deep NMF Neural Network is a valid model for classification tasks. NMF layers while deducing the training time of a Neural Network, also compromises its accuracy. NMF layers can be a very potential way to enhance the performance of the existant Neural Networks. However, there are a few aspects of the current NMF layer setting-up that needs closer examination and may leads to accuracy breaking through. We will leave them to future works.

References