Online Non-negative Matrix Factorization

Introduction

Suppose we are given data vectors x that always have the form $x = Dy$, where D is a fixed non-negative *decoder matrix* and y is a *code vector* that is also non-negative but otherwise arbitrary. This is the same as saying the data can be explained as arbitrary non-negative combinations of a set of non-negative *feature vectors* — the columns of D. Given a sufficiently large set of data vectors x, can we recover the set of features, that is, some column-permutation of D?

The usual approach to this problem is to arrange a large number of the data vectors, by columns, into a rectangular matrix X. If we can express X in the form $X = DY$, where both D and Y are non-negative matrices, then D is the sought decoder matrix and the columns of Y give the non-negative feature-compositions of the data. Performing such a non-negative factorization of a non-negative matrix X is always possible: just set $D = X$ and use the identity matrix for Y (each data vector is assigned its own feature vector). A more interesting question is finding factorizations, or approximate factorizations, where D has few columns. When we don't insist on non-negativity of the factors this is an easy problem and is solved by singular value decomposition. Non-negative factorization is in general harder. In fact, just establishing the *non-negative rank* — the fewest possible number of feature vectors — is an NP-hard problem.

Online algorithms for non-negative matrix factorization (NMF) have a natural machine learning architecture called an *autoencoder*. This is a three-layer feed-forward network. The nodes in the first layer hold the values of the data vector x. In the hidden layer we have the code vector y, computed as $y = Ex$ where E is the *encoding matrix*. Finally, y is multiplied by D to reproduce the original data vector x in the third layer. The autoencoder is successful when it is able to recover any x (in the data) from x, for fixed E and D, and where the hidden layer nodes always have non-negative values. Note that only D is required to be non-negative, while E has no such constraint.

In an online, neural network based NMF algorithm, the edge-weights corresponding to the matrices E and D would be initialized in some way and modified in training so that for sampled data x the agreement between x and DEx improves over time as does the non-negativity of the hidden layer values Ex. Training an autoencoder is an example of *unsupervised learning*. The usual approach, called *stochastic gradient descent*, averages the gradient of some measure of the autoencoder fidelity with respect to the edge-weights over many random data, and then uses the gradient information to increment the edge-weights. These notes describe an alternative training regimen called *conservative learning*.

In conservative learning the network is updated upon seeing each data item x. The idea is to make the minimal change to the edge-weights (the E and D matrices) so that x is correctly recovered from itself and of course the hidden layer values y are non-negative. By minimizing the changes to E and D we minimize the corruption of the reconstruction of previously seen data. One can prove that this strategy converges for *perceptrons* — linear two-layer networks. Not much can be proved for the case at hand, where the output of the network depends nonlinearly on the parameters (elements of E and D). However, most of our algorithm is based on a linearization of the equations, so we should still expect good convergence once the parameters enter domains where the linear approximation is good. When this is not the case, the updates can behave chaotically, like Newton-iterations for solving equations when the initial point is far from a root. This is not necessarily bad given the hardness of NMF in general, where exhaustive searching of some kind cannot be avoided.

Equations for the conservative learning update

We currently have encoder and decoder matrices E and D and are given a new data item x. Our goal is to find updated matrices $E_* = E + dE$, $D_* = D + dD$ and code vector $y_* = E_* x$ that satisfy

(a) $y_* \geq 0$

(b) $D_* y_* = x$

(c) $D_* \geq 0$

and also minimize

$$\mathcal{L}_0 = \frac{1}{2}\,\mathrm{Tr}\, dE^T dE + \frac{1}{2w}\,\mathrm{Tr}\, dD^T dD.$$

The function $\mathcal{L}_0$ is the L^2 norm of the matrix changes with a relative scale factor w included as a possibly useful hyper-parameter. In the optimization literature it is called the Lagrangian function, mostly because it is often augmented (see below) with Lagrange-multiplier terms to implement constraints. There is no connection with Lagrangian mechanics. To simplify the presentation we also assume the data are normalized as $\|x\|^2 = x^T x = 1$.

Imposing code vector non-negativity

It is straightforward to find the optimal update when constraints (b) and (c) above are ignored. We take this approach confident that equation (b) can be satisfied by adjusting D_* and also E_*, through its effect on just the positive elements of y_* (both matrices should be changed to minimize $\mathcal{L}_0$). Finally, it is valid to completely neglect constraint (c) when the training data includes items where the code vector is sparse. In particular, each 1-sparse y_* imposes non-negativity of the corresponding column of D_* through constraint (b).

To satisfy (a) while minimizing $\mathcal{L}_0$ we need the minimal change dE_1 that sets all the negative values in

$$y_0 = Ex$$

to zero, leaving the rest unchanged:

$$y_1 = \max(0, y_0).$$

This is accomplished by the rank-1 update

$$dE_1 = (y_1 - y_0)x^T,$$

and we write the updated encoding matrix after this stage as

$$E_1 = E + dE_1.$$

Imposing zero reconstruction error

The next step is to make changes $E_* = E_1 + dE_2$ and $D_* = D + dD$ that give zero reconstruction error on x (constraint (b)). We are able to do this in the limit where both dE_2 and dD are small.

The Lagrangian function for this step includes two constraint multiplier terms:

$$\begin{aligned}\mathcal{L}_1 = &\frac{1}{2}\operatorname{Tr} dE_2^T dE_2 + \frac{1}{2w}\operatorname{Tr} dD^T dD \\ &+ \xi^T\left(y_* - (E_1 + dE_2)x\right) + \eta^T\left(x - (D + dD)y_*\right).\end{aligned}$$

Our assumption that small dE_2 and dD can give zero reconstruction error implies that the *forward pass* values

$$y_1 = E_1 x \tag{1}$$

$$x_1 = D y_1 \tag{2}$$

are respectively close to the optimal y_* and x. We therefore treat $y_* - y_1$ and $x - x_1$ also as small quantities.

Asserting stationarity (at the optimum) of $\mathcal{L}_1$ with respect to dE_2 we obtain

$$dE_2 = \xi x^T$$

and using the normalization of x

$$dE_2\, x = \xi. \tag{3}$$

Since x has a fixed norm, this shows that the Lagrange multiplier ξ has the same order of smallness as dE_2. Stationarity of $\mathcal{L}_1$ with respect to ξ (the constraint imposed by this multiplier) gives the equation

$$y_* = E_1 x + dE_2 x,$$

or, using (1) and (3),

$$y_* - y_1 = \xi. \tag{4}$$

We need also assert stationarity with respect to dD. This yields the following equation

$$\begin{aligned} dD &= w\eta\, y_*^T \\ &\approx w\eta\, y_1^T, \end{aligned}$$

where in the approximation we retain only the leading order terms. Multiplying on the right by y_1 we obtain

$$dD\, y_1 \approx w\|y_1\|^2\eta. \tag{5}$$

As in the case of the other multiplier, we see that η also has the same order of smallness as the matrix changes.

Stationarity of $\mathcal{L}_1$with respect to y_* gives an equation that relates the two multipliers (again retaining only leading order terms in the approximation):

$$\begin{aligned} \xi &= (D^T + dD^T)\eta \\ \xi &\approx D^T\eta. \end{aligned} \tag{6}$$

We are now in a position to study the main equation, the one that imposes zero reconstruction error. Most directly this follows from stationarity of $\mathcal{L}_1$with respect to the remaining parameter, η:

$$\begin{aligned} x &= (D + dD)y_* \\ &= (D + dD)(y_1 + y_* - y_1) \\ &\approx Dy_1 + D(y_* - y_1) + dDy_1 \\ &\approx x_1 + D\xi + w\|y_1\|^2\eta. \end{aligned}$$

As before, we retained only the leading order (beyond $x \approx x_1$) and used (2), (4) and (5). Using (6) we can write this equation just in terms of the multiplier η:

$$\Delta = x - x_1 = (DD^T + w\|y_1\|^2)\eta. \tag{7}$$

Solving the symmetric, positive definite system (7) is the most expensive step in the update. For large networks one would want to use an iterative method, such as conjugate gradient (CG). In general CG needs as many iterations as the size of η — the number of input nodes. Each iteration involves multiplication by DD^T, a backward-forward propagation pair in the network. A complete solution would therefore be prohibitive for very large networks. In order to preserve good scaling behavior we instead make only a single CG iteration. A weak defense of this strategy is that the algorithm will receive similar training data in the future, for which it will then perform the second, third, etc. CG iterations (this neglects interference from non-similar data).

The CG iterates are uniquely determined by the choice of the first point, which we take to be $\eta = 0$ since this corresponds to the zero reconstruction error point ($\Delta = 0$) we have been linearizing about. For this initial point the first CG iteration corresponds to

$$\eta = c\Delta$$

for some constant c whose value is determined by projecting (7) onto Δ:

$$\|\Delta\|^2 = \Delta^T(DD^T + w\|y_1\|^2)c\Delta.$$

Solving for c we obtain the single-CG-iteration approximate solution for the η multiplier:

$$\eta = \frac{1}{\frac{\|D^T\Delta\|^2}{\|\Delta\|^2} + w\|y_1\|^2}\Delta.$$

This completes the approximate calculation for conservatively bringing the reconstruction error to zero.

Summary of the conservative update rule

1. Normalize, if necessary, the data vector x.
2. Compute the code vector by encoding the data:
$$y_0 = Ex.$$
3. Compute the nearest non-negative code vector:
$$y_1 = \max(0, y_0).$$
4. Make the first rank-1 update to the encoding matrix $E \to E + dE_1$ where
$$dE_1 = (y_1 - y_0)x^T.$$
5. Compute the autoencoder output by decoding the non-negative code vector:
$$x_1 = Dy_1.$$
6. Compute the reconstruction error vector:
$$\Delta = x - x_1.$$
7. Compute the multiplier for the reconstruction fidelity constraint:
$$\eta = \frac{1}{\frac{\|D^T\Delta\|^2}{\|\Delta\|^2} + w\|y_1\|^2}\Delta.$$
8. Update the decoder matrix $D \to D + dD$ where
$$dD = w\eta\, y_1^T.$$
9. Compute the code vector multiplier:
$$\xi = D^T\eta.$$
10. Make the second rank-1 update to the encoding matrix $E \to E + dE_2$ where
$$dE_2 = \xi x^T.$$

Although it should not be necessary if the data includes samples generated by sparse code vectors, one may include

11. Make decoding matrix non-negative by hand:
$$D \to \max(0, D).$$

Implementation by the program `nmf.c`

Online non-negative matrix factorization, based on the conservative learning rule, is implemented by the C program `nmf`. In addition to the data to be factorized, the size of the code vector, and the hyper-parameter w, `nmf` is told how many data to process and the size of batch for recording results. When the number of data to process exceeds the number of vectors in the data file, `nmf` makes repeated passes through the data file (always in the same order). Results are appended to a log-file with suffix `.log` at the completion of each batch.

The elements of the encoding matrix E are initialized to uniform random numbers in the range -1 to 1. This particular choice simply sets the scale, which is arbitrary because the decoding matrix D is initially set to zero. As the data are processed, the reconstruction error $\|\Delta\|$ is computed and averaged for the current batch. Recall that learning occurs with each data item and is independent of the choice of batch size. When a batch of data is processed the average $\|\Delta\|$ is written to the log-file and the current E and D are written to a model-file with suffix `.model` if the average $\|\Delta\|$ is the best achieved so far.

The lines of the model file have the same size as the data vectors for both E and D, so the latter needs to be transposed to correspond to the conventions above. If `nmf` is processing the data with the input choice of f features (code vector size), then (non-blank) line $k + f$ is the non-negative feature vector k while line k is the vector that acts as the *detector* or *receptive field* for that feature. Note that each feature vector may be rescaled if the corresponding detector vector is rescaled by the inverse factor. The scales that arise in the model-file are purely the result of the conservative learning rule; the program does not apply any normalization.

Demonstration with synthetic data

As a simple test of the online NMF algorithm we took the 64 MNIST images shown in Figure 1 as the columns of a $28^2 \times 64$ decoder D, and generated synthetic data by combining random subsets of eight columns with uniformly random and non-negative coefficients. A sampling of the 10^4 data generated in this way is shown in Figure 2.

The algorithm readily discovers the 64 generating digit images from the data: the decoding vectors in the model-file are indistinguishable from a permutation of the images in Figure 1. Empirically the best results are obtained for w of order unity. The evolution of the average

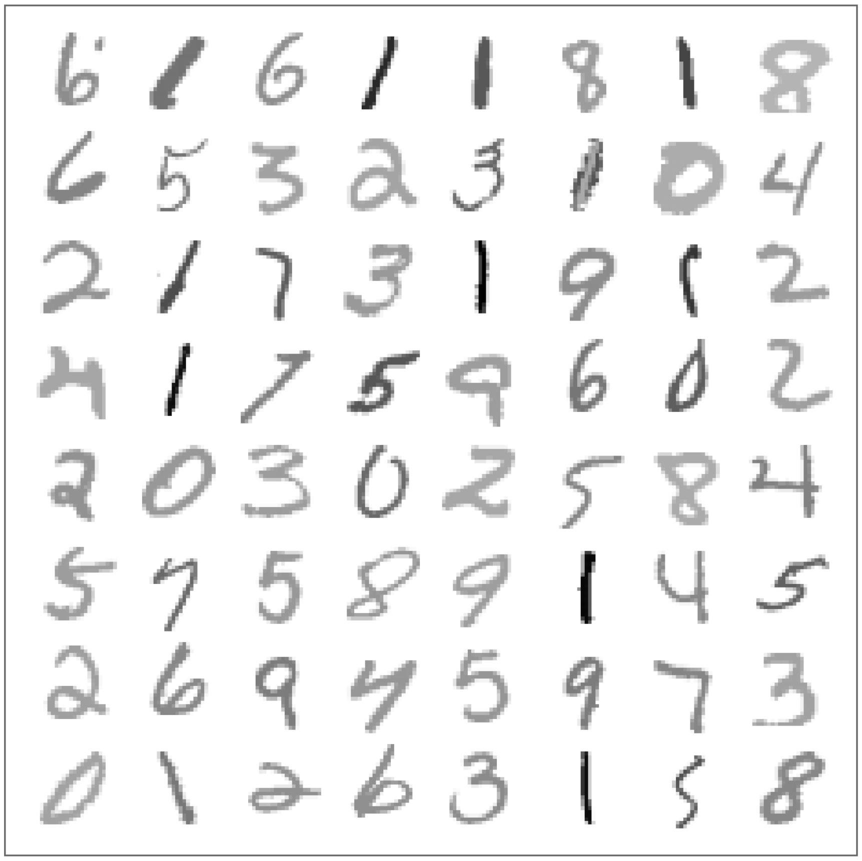

Figure 1: The 64 MNIST images used to generate the synthetic data. Since each image is a vector of size 28^2, the decoder matrix D is 784×64.

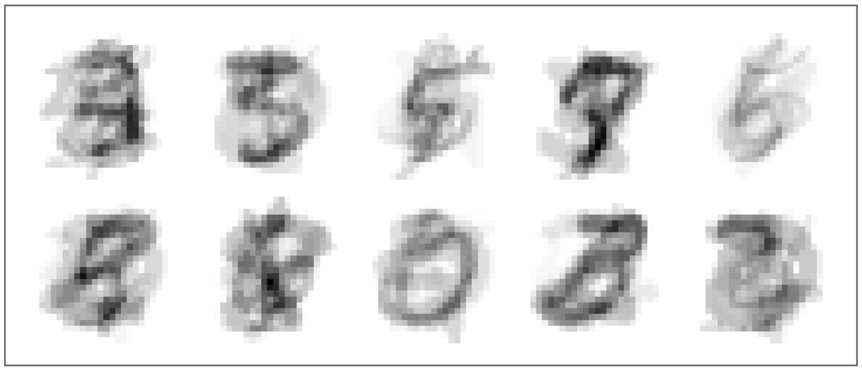

Figure 2: Ten samples of the synthetic data set, each one a random non-negative combination of eight of the images in Figure 1.

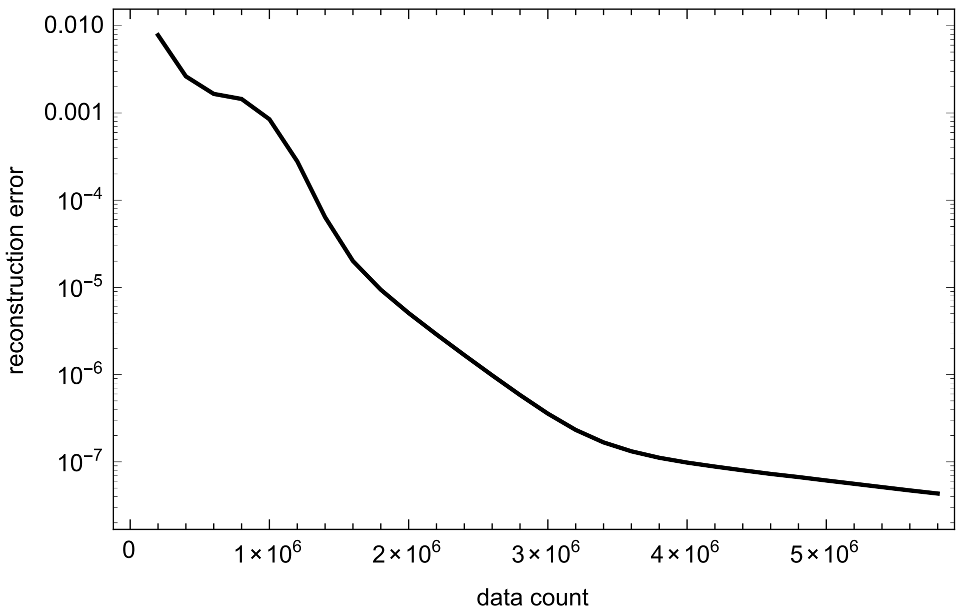

Figure 3: Decrease of the average reconstruction error with the number of data processed, for the synthetic data set.

reconstruction error with $w = 1$ is plotted in Figure 3.

The rows of E (not shown) corresponding to the detectors for the 64 features are not very interesting because they are mostly random. This is expected for our synthetic data set, where the data perfectly spans only a 64 dimensional space leaving $784 - 64$ orthogonal dimensions that can be added to E without effect. In an exact factorization E is some pseudo-inverse of D, but not necessarily a nice one like the Moore-Penrose pseudo-inverse.

Demonstration with natural data

The 60,000 MNIST (training) images are non-negative vectors generated by humans with a limited repertoire of writing motions. Is it possible to represent these as non-negative combinations of a significantly smaller number of non-negative feature vectors? In this case we do not expect the reconstruction error to reach zero, but instead are interested in obtaining the smallest possible error for particular choices of the number of feature vectors, f.

We now find that the algorithm converges more quickly when the parameter w is small.

Moreover, we observe that the reconstruction error, up to small residual fluctuations, settles on a value that depends on f but seems insensitive to the random initial encoder E. For $w = 10^{-5}$, we obtain $\|\Delta x\| = 0.0177$ for $f = 50$, 0.0122 for $f = 100$, and 0.00770 for $f = 200$.

A more critical test is reproducibility of the feature vectors for different random starts. A quick survey of the results supports this conclusion, but a quantitative comparison has not been performed. Figure 4 shows the features obtained for $f = 100$; Figure 5 shows the corresponding detectors. Many interesting observations can be made about the structure of these images, as well as the distribution of the code vector y over the MNIST data. For example, we find that almost all the components of y have a strongly bimodal distribution, with the interpretation that features are either absent or present with a fixed magnitude.

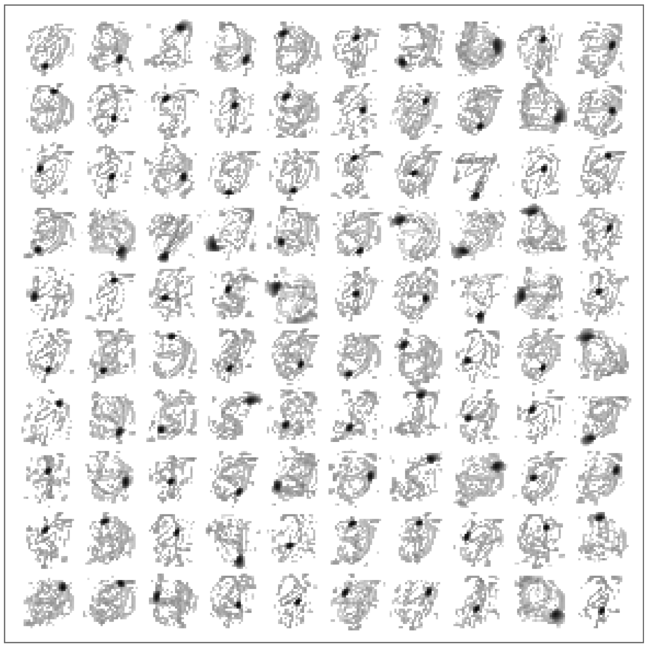

Figure 4: The set of 100 feature vectors discovered by `nmf` when applied to the 60,000 MNIST images; contrast was exponentiated to the power 0.2 to reduce the dynamic range.

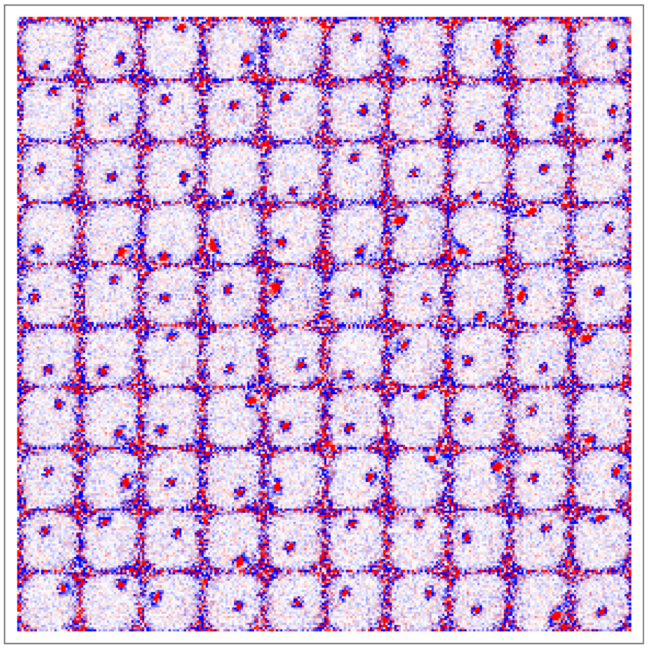

Figure 5: Detector vectors corresponding to the features in Figure 4. Positive value are rendered in red, negative values in blue.