

Intership - Sparse Coding

Thomas Rolland

Abstract

Little document to summarize sparse coding. Mainly based on the Hugo Larochelle courses.

This document is a draft

Contents

1	Sparse coding	1
1.1	Introduction	1
1.2	Inderence of Sparse code	2
1.2.1	Compute h	2
1.2.2	Compute D	3
1.3	Other algorithms	5
1.4	Application for MNIST dataset	5
1.4.1	Prototype	5
1.4.2	SPAMS	5
2	Sparse Coding for speech recognition	6

1 Sparse coding

1.1 Introduction

Idea Sparse dictionary learning is a representation learning method which aims at finding a sparse representation of the input data (in form of a linear combination of basic elements (called Atoms). The idea of using learned dictionary instead of a predefined one is based on wavelets. This sparse learned models has recently led to state-of-the-art result for denoising, classification,...

Unsupervised learning Only use the inputs $x^{(t)}$ ($X = [x_1, \dots, x_n]$ in $\mathbb{R}^{m \times n}$) for learning. Automatically extract meaningful features of our data, leverage the availability of unlabeled data and add a data-dependent regularize to trainings.

Sparse coding is one of the neural networks used for unsupervised learning (like restricted boltzmann machines and autoencoders).

The idea behind sparse coding is: For each x^t find a latent representation h^t such that:

- It is sparse: the vector h^t has many zeros (only few nonzero elements)
- We can reconstruct the original input $x^{(t)}$ as well as possible.

That mean, more formally:

$$\min_D \frac{1}{T} \sum_{t=1}^T \min_{h^{(t)}} \frac{1}{2} \|x^{(t)} - D h^{(t)}\|_2^2 + \lambda \|h^{(t)}\|_1$$

- D is a matrix of weights, usually refer to that matrix as a dictionary matrix (containt atoms) with $D \in \mathbb{R}^{m \times k}$ (k the number of atoms)
- $\|x^{(t)} - D h^{(t)}\|_2^2$ is the reconstruction error

- $D h^{(t)}$ is the reconstruction of $\hat{x}^{(t)}$
- $\|h^{(t)}\|_1$ is the sparsity penalty (more 0 in h we have, better it is)

This two objectives fight each other. But it still a optimization problem (cf min), and we'll try to optimize it for each training example $x^{(t)}$. This is why we have a sum over all the training examples.

We also constrain the columns of D to be of norm 1 (otherwise, D could grow big while $h^{(t)}$ becomes small to satisfy the prior). And sometimes the columns are constrained to be no greater than 1.

However, $h^{(t)}$ is now a complicated function of $x^{(t)}$:
Encoder is the minimization $h(x^{(t)}) = \arg \min_{h^{(t)}} = \frac{1}{2} \|x^{(t)} - D h^{(t)}\|_2^2 + \lambda \|h^{(t)}\|_1$, so the optimization problem is more complicated than a simple non linear problem.
The idea to solve this minimization problem is [Lee et al., 2007] :

while D not_converged :

Fix D

Minimize h (1)

Fix h

Minimize D (2)

Dictionary We can also write $\hat{x}^{(t)} = D h(x^{(t)}) = \sum_{\substack{k.s.t. \\ h(x^{(t)})_k \neq 0}} D_{:,k} h(x^{(t)})_k$

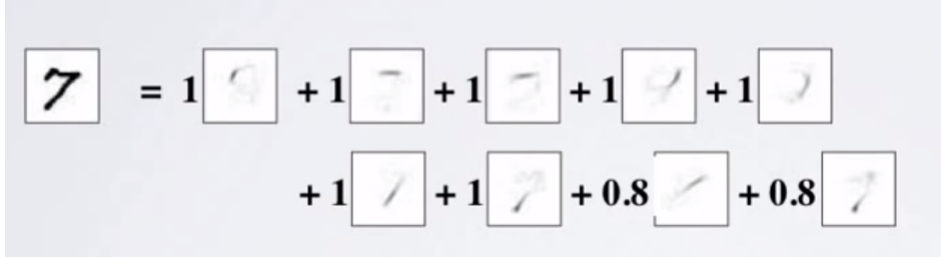


Figure 1: Example of reconstruction using sparse coding

The images refer to $D_{:,k}$ (columns of D wich are not equals to 0) and the factor (1 or 0.8 in this case) refer to $h(x^{(t)})_k$

We also refer to D as the dictionary:

- in certain applications, we know what dictionary matrix to use
- often however, we have to learn it

In general we have $k \ll n$. But we can use an overcomplete dictionary with $k > m$.

1.2 Indereence of Sparse code

1.2.1 Compute h

Idea Here we develop the (1) computation from the initial idea.

Assume we are given a dictionary matrix D, how do we compute $h(x^{(t)})$. We have to optimize:

$$l(x^{(t)}) = \frac{1}{2} \|x^{(t)} - D h^{(t)}\|_2^2 + \lambda \|h^{(t)}\|_1 \text{ w.r.t. } h^{(t)}$$

We could use a gradient descent method:

$$\Delta_{h^{(t)}} l(x^{(t)}) = D^T (D h^{(t)} - x^{(t)}) + \lambda \text{sign}(h^{(t)})$$

The issue is L1 norm is not differentiable at 0. The solution is : if $h^{(t)}$ changes sign because of L1 norm gradient then clamp to 0. That mean :

$$\begin{aligned} h_k^{(t)} &= h_k^{(t)} - \alpha (D_{:,k})^T (Dh^{(t)} - x^{(t)}) \\ \text{if } \text{sign}(h_k^{(t)}) &\neq \text{sign}(h_k^{(t)} - \alpha \lambda \text{sign}(h_k^{(t)})) \text{ then: } h_k^{(t)} = 0 \\ \text{else } h_k^{(t)} &= h_k^{(t)} - \alpha \lambda \text{sign}(h_k^{(t)}) \end{aligned}$$

ISTA (iterative Shrinkage and Thresholding Algorithm) :

```

initialize h
while h not_converged:
    for each h_k in h:
        h_k = h_k - alpha * transpose(D[:,k]) * (D*h - x)
        h_k = shrink(h_k, alpha*lambda_coef)
return h

```

Here **shrink(a,b)** = [..., sign(a_i) max(| a_i | - b_i , 0), ...]

1.2.2 Compute D

There are three algorithms used for dictionary update.

Algorithm 1: A gradient descent method Our original problem is:

$$\min_D \frac{1}{T} \sum_{t=1}^T \min_{h^{(t)}} \frac{1}{2} \|x^{(t)} - D h^{(t)}\|_2^2 + \lambda \|h^{(t)}\|_1$$

But here we assume $h(x^{(t)})$ doesn't depend on D. So we must minimize:

$$\min_D \frac{1}{T} \sum_{t=1}^T \min_{h^{(t)}} \frac{1}{2} \|x^{(t)} - D h^{(t)}\|_2^2$$

```

while D not_converged:
    # Perform gradient update of D
    D = D - alpha * (1/T) * sum((x - D h) * tranpose(h))
    # Renormalize the columns of D
    for each column D[:,j]:
        D[:,j] = (D[:,j] / norm(D[:,j]))
return D

```

Algorithm 2: Block-coordinate descent We must minimize:

$$\min_D \frac{1}{T} \sum_{t=1}^T \min_{h^{(t)}} \frac{1}{2} \|x^{(t)} - D h^{(t)}\|_2^2$$

The idea is to solve for each column $D_{:,j}$ in cycle (that mean to optimize in one direction at time). For that we must set the gradient for $D_{:,j}$ to zero.

We have:

$$0 = \frac{1}{T} \sum_{t=1}^T (x^{(t)} - Dh(x^{(t)})) h_j^{(t)}$$

We separe $D_{:,j}$ from the rest of D:

$$0 = \frac{1}{T} \sum_{t=1}^T (x^{(t)} - (\sum_{i \neq j} D_{:,i} h_i^{(t)}) - (D_{:,j} h_j^{(t)})) h_j^{(t)}$$

Our aim is to find the value of $D_{:,j}$, we must isolate $D_{:,j}$:

$$0 = \frac{1}{T} \sum_{t=1}^T (x^{(t)} h_j^{(t)} - (\sum_{i \neq j} D_{:,i} h_i^{(t)} h_j^{(t)}) - (D_{:,j} h_j^{(t)2}))$$

$$0 = (\sum_{t=1}^T (x^{(t)} h_j^{(t)} - (\sum_{i \neq j} D_{:,i} h_i^{(t)} h_j^{(t)}) - (\sum_{t=1}^T (D_{:,j} h_j^{(t)2}))))$$

$$\sum_{t=1}^T (D_{:,j} h_j^{(t)2}) = \sum_{t=1}^T (x^{(t)} h_j^{(t)} - (\sum_{i \neq j} D_{:,i} h_i^{(t)} h_j^{(t)}))$$

$$\begin{aligned}
D_{:,j} \sum_{t=1}^T h_j^{(t)2} &= \sum_{t=1}^T (x^{(t)} h_j^{(t)} - (\sum_{i \neq j} D_{:,i} h_i^{(t)} h_j^{(t)})) \\
D_{:,j} &= \frac{1}{\sum_{t=1}^T h_j^{(t)2}} \sum_{t=1}^T (x^{(t)} h_j^{(t)} - (\sum_{i \neq j} D_{:,i} h_i^{(t)} h_j^{(t)})) \\
D_{:,j} &= \frac{1}{\underbrace{\sum_{t=1}^T h_j^{(t)2}}_{A_{j,j}}} \underbrace{\sum_{t=1}^T (x^{(t)} h_j^{(t)})}_{B_{:,j}} - \sum_{i \neq j} D_{:,i} \underbrace{(\sum_{t=1}^T h_i^{(t)} h_j^{(t)})}_{A_{i,j}} \\
D_{:,j} &= \frac{1}{A_{j,j}} (B_{:,j} - D A_{:,j} + D_{:,j} A_{j,j})
\end{aligned}$$

```

while D not_converged:
    # For each column D[:, j] perform updates
    for each column D[:, j]:
        D[:, j] = (1/A[j, j])*(B[:, j] - D A[:, j] + D[:, j] A[j, j])
        # Normalization
        D[:, j] = D[:, j]/norm(D[:, j])

return D

```

Algorithm 3: Online learning algorithm For large datasets we want to update D after visiting each $x^{(t)}$. The solution is for each $x^{(t)}$ [Mairal et al., 2009] :

- Perform inference of $h(x^{(t)})$ after visiting each $x^{(t)}$
- Update running averages of the quantities required to update D:
 - $B = \beta B + (1 - \beta)x^{(t)}h(x^{(t)})^T$
 - $A = \beta A + (1 - \beta)h(x^{(t)})h(x^{(t)})^T$
- Use current value of D as "warm start" to block-coordinate descent (warm start \iff With the previous value of D)

(We have to specify β like a learning rate α in the gradient descent)

```

Initialize D # Not to 0 ! (To respect the constraint we define before)
while D not_converged:
    for each x:
        Infer code h
        #Update dictionary
        A = A + h * transpose(h)
        B = B + x * transpose(h)
        #Batch upgrade
        #A = beta * A + ( 1 - beta ) * h * transpose(h)
        #B = beta * B + ( 1 - beta ) * x * transpose(h)
        while D not_converged:
            for each column D[:, j]:
                D[:, j] = (1/A[j, j])*(B[:, j] - D A[:, j] + D[:, j] A[j, j])
                # Normalization
                D[:, j] = D[:, j]/norm(D[:, j])

```

Optimizing the Algorithm In practice, it's possible to improve the convergence speed of this algorithm by using a Mini-batch extension: By drawing $\eta > 1$ signals at each iteration instead of a single one.

$$\begin{cases} A_t = \beta A_{t-1} + \sum_{i=1}^{\eta} \alpha_{t,i} \alpha_{t,i}^T \\ B_t = \beta B_{t-1} + \sum_{i=1}^{\eta} x \alpha_{t,i}^T \end{cases}$$

Then $\beta = \frac{\theta+1-\eta}{\theta+1}$, where $\theta = t\eta$ if $t < \eta$ and $\eta^2 + t - \eta$ if $t \geq \eta$

1.3 Other algorithms

There are other algorithms like Efficient Shift-Invariant Dictionary learning which refers to the problem of discovering a set of latent basis vectors that capture informative *local patterns* at different locations of the input sequences and not in all input sequences [Zheng et al., 2016]. But in our problem we used dictionary learning on windows of 10ms, with approach we already capture informative local patterns, we don't need to use Shift-Invariant dictionary learning.

1.4 Application for MNIST dataset

The MNIST database of handwritten digits, available from Yann Lecun's website. MNIST has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.



Figure 2: Example of MNIST's handwritten digits

1.4.1 Prototype

My first task is to realise a Sparse Coding prototype to compute Sparse Coding on this dataset, using Python. The aim here, is to understand the underlying principles of this method, you can find this prototype in `Code directory` of this repository as `SparseCoding.py`.

There are some results of this prototype: For time saving I used only 100 digits as input.

1.4.2 SPAMS

SPAMS (SPArse Modeling Software) is an optimization toolbox for solving various sparse estimation problems.

- Dictionary learning and matrix factorization (NMF, sparse PCA, ...)
- Solving sparse decomposition problems with LARS, coordinate descent, OMP, SOMP, proximal methods
- Solving structured sparse decomposition problems (ℓ_1/ℓ_2 , ℓ_1/ℓ_{inf} , sparse group lasso, tree-structured regularization, structured sparsity with overlapping groups,...).

It is developed and maintained by Julien Mairal (Inria), and contains sparse estimation methods resulting from collaborations with various people: notably, Francis Bach, Jean Ponce, Guillermo Sapiro, Rodolphe Jenatton and Guillaume Obozinski.

2 Sparse Coding for speech recognition

In this part of the paper we will see novel feature extraction technique based on the principles of sparse coding [S. V. S. Sivaram et al., 2010]. Sparse coding deals with the problem of how represent a given audio input as a linear combination of a minimum number of basis function. The weights of the linear combination are used as feature for speech recognition (acoustic modeling). Note the input dimensionality is typically **much** less than the number of atoms in the dictionary *i.e.* we use overcomplete dictionary. We use Sparse Coding algorithm as describe before and we get the dictionary D and the matrix of sparse coefficients h .

Reflection path In [S. V. S. Sivaram et al., 2010] they used spectro-temporal speech domain which is obtained by performing a short time Fourier transform (STFT) with an analysis window of length 25 ms and a frameshift of 10 ms on the input signal. Log critical band energies are subsequently obtained by projecting the magnitude square values of the STFT output on a set of frequency weights, which are equally spaced on the Bark frequency scale, and then applying a logarithm on the output projections.

References

- [Lee et al., 2007] Lee, H., Battle, A., Raina, R., and Ng, A. Y. (2007). Efficient sparse coding algorithms. In Schölkopf, B., Platt, J. C., and Hoffman, T., editors, *Advances in Neural Information Processing Systems 19*, pages 801–808. MIT Press.
- [Mairal et al., 2009] Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2009). Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09*, pages 689–696, New York, NY, USA. ACM.
- [S. V. S. Sivaram et al., 2010] S. V. S. Sivaram, G., Krishna Nemala, S., Elhilali, M., D. Tran, T., and Hermansky, H. (2010). Sparse coding for speech recognition.
- [Zheng et al., 2016] Zheng, G., Yang, Y., and Carbonell, J. (2016). Efficient shift-invariant dictionary learning. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16*, pages 2095–2104, New York, NY, USA. ACM.