Intership - Sparse Coding

Thomas Rolland

Abstract

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

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. Thie sparse learned models has recently led to state-of-the-art result for denoising, classification,...

Unsupervised learning Only use the inputs $x^{(t)}$ 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 sparese: 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)
- $\|x^{(t)} - D\ h^{(t)}\|_2^2$ 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)

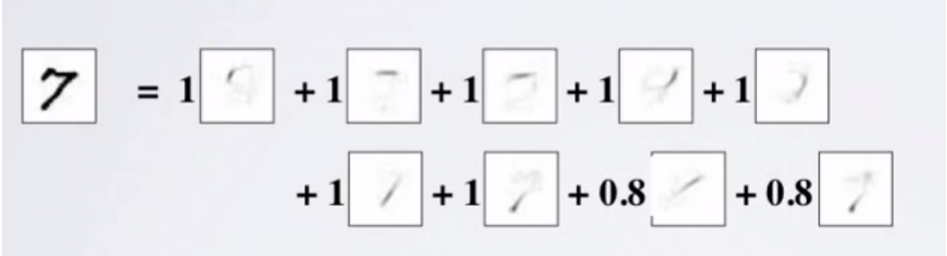
This two objectives fight each other. But it still a optimization problem (cf min), and we'll try to optimize it for each training example $x^{(t)}$. This is why we have a sum over all the training examples.

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 greather 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
    while h not_converged:
        Minimize h                (1)
    Fix h
    Minimize D                    (2)
```

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



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

1.2 Inderence 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:

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

We could use a gradient descent method:

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

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

ISTA (iterative Shrinkage and Thresholding Algorithm :

```
initialize h
while h not_converged:
    h = h -alpha * transpose(D) * (D*h - x)
    h = shrink(h, alpha*lambda_coef)
return h
```

Here **shrink(a,b)** = [..., $\text{sign}(a_i)\ \max(|a_i| - b_i, 0)$, ...]

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:

$$0 = \frac{1}{T} \sum_{t=1}^{T} (x^{(t)} - (\sum_{i \neq j} D_{.,i}\; h(x^{(t)})_i\;) - (D_{.,j}\; h(x^{(t)})_j)\;)\; h_j^{(t)}$$

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

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

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

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

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

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

$$D_{.,j} = \underbrace{\frac{1}{\sum_{t=1}^{T} h_j^{(t)2}}}_{A_{j,j}} \underbrace{\sum_{t=1}^{T} (x^{(t)} h_j^{(t)})}_{B_{.,j}} - \sum_{i \neq j} D_{.,i} (\underbrace{\sum_{t=1}^{T} h_i^{(t)} h_j^t}_{A_{i,j}})$$

$$D_{.,j} = \frac{1}{A_{j,j}} (B_{.,j} - DA_{.,j} + D_{.,j} A_{j,j})$$

```
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 specifie 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 used Shift-Invariant dictionary learning.

2 Sparse Coding for speech recognition

In this part of the paper we will see novel feature exraction technique based on the principles of sparse coding. Sparse codigin 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).

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.

[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.