

Report of Project 1

A Review of Convexified Convolutional Neural Network

Xiaodong Jia

June 12, 2017

Contents

| | | |
|----------|--|----------|
| 1 | Introduction | 2 |
| 2 | A Brief Summary of the Main Ideas | 2 |
| 2.1 | Convex DNNs | 2 |
| 2.2 | Convex CNNs | 3 |
| 3 | The Construction of A Three-layer DNN | 3 |
| 4 | The Construction of A Two-layer CCNN | 4 |

1 Introduction

In this review, we analyze a two layer convexified convolutional neural network(CCNN) based on [3], make a review of the related works and use some new methods to solve this problem.

2 A Brief Summary of the Main Ideas

The work in [3] is mainly about convexifying a two-layer convolutional neural network. Before it there is the convexification of a multi-layer deep neural network(DNN) in [2], earlier than which is a more generalized two layer modeling in [1].

There are many advantages of a convex problem, like we won't stack at a local minimum. However, With the increase of layer number, the scale of both problems soon become not affordable, and even a two-layer CNN problem requires much more resource than a classical method.

2.1 Convex DNNs

As a common definition, an n -layer DNN can be written as

$$f(x) = l_n \circ \dots \circ l_2 \circ l_1(x),$$

where $l_i = A_i(w_i^T x + b_i), i \in \{1, \dots, n\}$ with A_i non-linear, often called an activation function. We will omit the bias term b_i for simplicity. We also have a loss function $L(f(x); y)$ to measure the accuracy of the DNN, where y is the label of x . Our aim is then to compute the w such that

$$w = \operatorname{argmin}_w \sum_x L(f(x); y(x)).$$

Now, suppose we have a three-layer DNN. The idea of convexifying it is *relaxation*, which consists of two parts. We first relax the activation function, because it introduces strong non-convex to the model. This is done by change the optimization problem to

$$\min_{W, U, V, \Phi, \Theta} L_1(WX, \Phi) + \frac{1}{2}\|W\|^2 + L_2(U\Phi, \Theta) + \frac{1}{2}\|U\|^2 + L_3(V\Theta, Y) + \frac{1}{2}\|V\|^2.$$

Let's explain the above problem. First, we omit the bias b_i so there's no b_i . Second, W, U, V are the parameters w of the first, second, and third layer function l_1, l_2, l_3 . Third, which is how we relax the activation function, we let Φ and Θ be the outputs of L_1 and L_2 respectively, and put them into the object function, i.e. they are now parameters as U, V, W . There are also new loss functions L_1, L_2, L_3 , which connect the input of output in each layer. These non-linear loss functions replace the activation function. Finally, there are regularization terms $\|U\|^2, \|V\|^2, \|W\|^2$ to control the complexity of the model. They could be weighted but we omit the weights to make it simple.

Now, the problem is how to make the new object function convex on the parameters U, V, W, Φ, Θ . Generally speaking, we relax, or split the parameters so that they or their representatives lie in a larger space.

2.2 Convex CNNs

In the case of CNN, the idea is also relaxation, but this time we keep the activation function, and relax the non-linear part as a whole. A convolutional neural network(CNN) can be written as a function $f(x)$. Particularly, for a two layer convolutional neural network, the following form separates the trainable parameters and others,

$$f^A(x) := (\text{tr}(Z(x)A_1), \dots, \text{tr}(Z(x)A_{d_2})),$$

where A denotes all trainable parameters, and Z only depends on the inputs.

If the loss function $\mathcal{L}(f; y)$ is convex about f , $\mathcal{L}(f^A(Z))$ is convex about A . We can then solve the convex optimization problem

$$\hat{A} \in \text{argmin}_{\|A\|_* \leq R} \tilde{\mathcal{L}}(A)$$

where $\tilde{\mathcal{L}}(A) = \sum_{i=1}^n \mathcal{L}(f^A(x_n); y_n)$, n is the size of mini-batch, $\|\cdot\|_*$ denotes the nuclear norm, and R is a restriction. \hat{A} is then transformed to the corresponding parameters of the CNN. As a result, the original non-convex problem is transformed to a convex one.

3 The Construction of A Three-layer DNN

As mentioned, the problem we want to solve is

$$\min_{W, U, V, \Phi, \Theta} L_1(WX, \Phi) + \frac{1}{2}\|W\|^2 + L_2(U\Phi, \Theta) + \frac{1}{2}\|U\|^2 + L_3(V\Theta, Y) + \frac{1}{2}\|V\|^2. \quad (1)$$

(1) is convex in U, V, W given Φ, Θ , but not jointly convex in all of them, due to the interaction between U, V, W and Φ, Θ . To tackle this problem, consider the part of (1) corresponding to the second layer,

$$\min_U L(U\Phi, \Theta) + \frac{1}{2}\|U\|^2. \quad (2)$$

What we want is to split U and Φ . By the *representer theorem*, we know there is a matrix A such that $U = A\Phi'$. Denote $Z = U\Phi = A\Phi'\Phi = AK$ where $K = \Phi'\Phi$. Notice that Φ is the input of layer 2, and K is called the input kernel matrix. Then, we have

$$\|U\|^2 = \text{tr}(UU') = \text{tr}(AKA') = \text{tr}(AKK^\dagger KA') = \text{tr}(ZK^\dagger Z'),$$

where K^\dagger is the Moore-Penrose pseudo-inverse(recall $KK^\dagger K = K$ and $K^\dagger K K^\dagger = K^\dagger$). (2) can then be rewritten as

$$\min_Z L(Z, \Theta) + \frac{1}{2} \text{tr}(ZK^\dagger Z'). \quad (3)$$

We thus split U and Φ by represent them to Z and K .

The another problem is that Φ and Θ serve as the input and output for different layers simultaneously. We must make (2) convex in U, Φ, Θ but (3) is not convex in Θ . To solve this problem, we assume the loss function satisfies an extra postulate, so that Θ is boolean valued, and thus we can get a reformulation.

Postulate 1. $L(Z, \Theta)$ can be written as $L^u(\Theta'Z, \Theta'\Theta)$ for L^u jointly convex in both arguments.

The postulate may seem strange but it has backgrounds. The term $\Theta'Z$ is called a *propensity matrix* and $\Theta'\Theta$ is called an unnormalized *output kernel*, so now the loss function depends only on them.

Using Postulate 1 and $Z = U\Phi$, (2) can be rewritten as $L^u(\Phi'U\Phi, \Theta'\Theta) + \frac{1}{2}\|U\|^2$. Now, denote $N := \Theta'\Theta$ and $S := \Theta'Z = \Theta'U\Phi$ (hence $S \in \Theta'\mathbb{R}\Phi = N\mathbb{R}K$), (2) can be rewritten as

$$\min_S L^u(S, N) + \frac{1}{2} \text{tr}(K^\dagger S' N^\dagger S).$$

This objective is jointly convex in the propensity matrix S and output kernel N . Notice that N is positive semidefinite. Thus, to make (2) convex, the final postulate should require the domain of N to be convex.

Postulate 2. The domain of $N = \Phi'\Phi$ can be relaxed to a convex set preserving sufficient structure.

There is an relaxation approach in [1], which is only applicable for a single hidden layer network. A more generalized approach in [2] is as follows.

4 The Construction of A Two-layer CCNN

A two-layer CCNN can be written as a function $f : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_2}$, which takes in a vector x that is often the vector-representation of a picture. In the context of this review, the output $f(x)$ is a discrete distribution vector, i.e. the k th element $f_k(x) \in [0, 1]$ denotes the probability of x belonging to class k .

In a common explanation, the construction of f can be written as follows.

The input vector, or picture, x , is first separated to P patches, which can be written as a function $z_p(x) \in \mathbb{R}^{d_1}, 1 \leq p \leq P$.

Then, each patch is transformed to r scalars, which can be written as $h_j(z_p) = \sigma(w_j^T z_p), 1 \leq j \leq r$, where $w_j \in \mathbb{R}^{d_1}, \sigma : \mathbb{R} \rightarrow \mathbb{R}$ is in general a non-linear function. Each h_j is known as a *filter*.

Now we have $P \times r$ scalars. These scalars are finally summed together with weights, denoting as $\alpha_{k,j,p}$. The two-layer CNN can then be written as

$$f_k(x) := \sum_{j=1}^r \sum_{p=1}^P \alpha_{k,j,p} h_j(z_p(x)).$$

When σ is identity, i.e. $\sigma(x) = x, x \in \mathbb{R}$, we can separate the trainable param-

ters α, w with other constants. Rewritten f_k as

$$\begin{aligned}
f_k(x) &= \sum_{j=1}^r \sum_{p=1}^P \alpha_{k,j,p} h_j(z_p(x)) \\
&= \sum_{j=1}^r \sum_{p=1}^P \alpha_{k,j,p} w_j^T z_p(x) \\
&= \sum_{j=1}^r \alpha_{k,j}^T Z w_j \\
&= \sum_{j=1}^r \text{tr}(\alpha_{k,j}^T Z w_j) \\
&= \sum_{j=1}^r \text{tr}(Z w_j \alpha_{k,j}^T) \\
&= \text{tr}(Z \sum_{j=1}^r w_j \alpha_{k,j}^T) \\
&= \text{tr}(Z A_k)
\end{aligned}$$

where in the second equation $Z = (z_1(x), \dots, z_P(x))^T$, in the third equation $\alpha_{k,j} = (\alpha_{k,j,1}, \dots, \alpha_{k,j,P})^T$ and in the final equation $A_k = \sum_{j=1}^r w_j \alpha_{k,j}^T$.

Thus, let $A := (A_1(x), \dots, A_{d_2}(x))$ denoting all A_k , so A is in fact all the trainable parameters. We can then define a function

$$f^A := (\text{tr}(Z A_1), \dots, \text{tr}(Z A_{d_2})),$$

which is a linear function corresponding to A .

The CNN model class is a collection of such functions with constraints on A . In particular, we define

$$\mathcal{F}_{\text{CNN}}(B_1, B_2) := \{f^A \mid \max_{j \in [r]} \|w_j\|_2 \leq B_1, \max_{k \in [d_2], j \in [r]} \|\alpha_{k,j}\|_2 \leq B_2, \text{rank}(A) = r\}$$

where $[x] = \{i \mid 1 \leq i \leq x\}$.

So $\mathcal{F}(B_1, B_2)$ includes all such functions with a limited trainable parameters, and the rank constraint inherits from the formulation of A_k . Now, the matrix A can be decomposed as $A = UV^T$, where both U and V have r columns. The column space of A contains the convolution parameters $\{w_j\}$, and the row space of A contains the output parameters $\{\alpha_{k,j}\}$.

Now, the matrices A satisfying the constraints in \mathcal{F} in fact form a non-convex set. To make it convex, a standard relaxation is based on the nuclear norm $\|A\|_*$

which is the sum of the singular values of A . By the triangle inequality we have

$$\begin{aligned}
\|A\|_* &= \|(A_1, \dots, A_P)\|_* \\
&\leq \sum_{j=1}^r \|w_j(\alpha_{1,j}^T, \dots, \alpha_{d_2,j}^T)\|_* \\
&\leq \sum_{j=1}^r \|w_j\| \|(\alpha_{1,j}^T, \dots, \alpha_{d_2,j}^T)\|_* \\
&\leq r B_1 \sqrt{d_2 B_2^2} \\
&= r \sqrt{d_2} B_1 B_2.
\end{aligned}$$

Thus, we can define a more general CCNN class by

$$\mathcal{F}_{\text{CCNN}} := \{f^A \mid \|A\|_* \leq r \sqrt{d_2} B_1 B_2\},$$

which is convex and we have $\mathcal{F}_{\text{CCNN}} \supseteq \mathcal{F}_{\text{CNN}}$.

Now, let $\mathcal{L}(f(x), y)$ denote the loss function, where y is the label of x . We assume \mathcal{L} is convex and L -Lipschitz in the first argument. Our aim is then compute

$$\hat{f}_{\text{CCNN}} := \operatorname{argmin}_{f^A \in \mathcal{F}_{\text{CCNN}}} \sum_{i=1}^n \mathcal{L}(f^A(x_i); y_i).$$

When σ is a non-linear function, we transform the filter $h_{\sigma,w}(z_p(x))$ to a linear form by

$$h(z_p(x_i)) = \sum_{(i', p') \in [n] \times [P]} c_{i', p'} k(z_p(x_i), z_{p'}(x_{i'})),$$

where k is a positive semi-definite kernel function. Viewing

$$k(*, z_{p'}(x_{i'})), (i', p') \in [n] \times [P]$$

as a basis, the filter h is represented by $c_{i', p'}$. Now, let $K \in \mathbb{R}^{nP \times nP}$,

$$K_{(i,p),(i',p')} = k(z_p(x_i), z_{p'}(x_{i'})).$$

Consider a factorization $K = QQ^T$, where $Q \in \mathbb{R}^{nP \times m}$, we can rewrite h by

$$h(z_p(x_i)) = c^T (Q_{(i,p)} Q^T)^T = \langle Q_{(i,p)}, c^T Q \rangle.$$

Let $w = c^T Q$, we have $h(z_p(x_i)) = \langle Q_{(i,p)}, w \rangle$. In order to learn the filter h , it suffices to learn w . The non-linear h is thus transformed to a linear function (of w).

Now, as in the linear case but denoting Q as Z , we have

$$\begin{aligned}
f_k(x) &= \sum_{j=1}^r \sum_{p=1}^P \alpha_{k,j,p} h_j(z_p(x)) \\
&= \operatorname{tr}(Q A_k) \\
&= \operatorname{tr}(Z A_k).
\end{aligned}$$

Suppose we have computed a result $A \in \mathbb{R}^{m \times P d_2}$. At test time, given a new input $x \in \mathbb{R}^{d_0}$, we can compute the matrix Z as follows.

Notice that the (i, p) th row of K corresponding to $z_p(x_i)$, denoted as $K_{(i,p)}$, is $K_{(i,p)} = Q_{(i,p)} Q^T$. Let $v(z_p(x)) = (k(z_p(x), z_{p'}(x_{i'})))_{(i',p')} \in \mathbb{R}^{nP}$, we want to find the representation c_x so that

$$c_x = \operatorname{argmin}_{c_x} \|v(z_p(x))^T - c_x^T Q^T\|_2^2 = \|v(z_p(x)) - Q c_x\|_2^2.$$

The answer is $c_x = Q^\dagger v_p(x)$, where $v_p(x) = v(z_p(x))$. So we have $Z(x) = (Q^\dagger v(x))^T$.

Algorithm 1 Learning Two-layer Convexified Convolutional Neural Networks

Input: Data $\{(x_i, y_i)\}_{i=1}^n$, kernel function \mathcal{K} , regularization parameter $R > 0$, number of filters r .

1. Construct a kernel matrix $K \in \mathbb{R}^{nP \times nP}$ such that the entry at column (i, p) is and row (i', p') is equal to $\mathcal{K}(z_p(x_i), z_{p'}(x_{i'}))$.
2. Compute a factorization $K = QQ^T$ or an approximation $K \approx QQ^T$, where $Q \in \mathbb{R}^{nP \times m}$.
3. For each x_i , construct patch matrix $Z(x_i) = (Q^\dagger v(x_i))^T \in \mathbb{R}^{P \times m}$.
4. Solve the following optimization problem to obtain a matrix $\hat{A} = (\hat{A}_1, \dots, \hat{A}_{d_2})$:

$$\hat{A} \in \operatorname{argmin}_{\|A\|_* \leq R} \tilde{\mathcal{L}}(A) := \sum_{i=1}^m \mathcal{L}(\operatorname{tr}(Z(x_i)A), \dots, \operatorname{tr}(Z(x_i)A_{d_2}); y_i).$$

5. Compute a rank- r approximation $\tilde{A} \approx \hat{U} \hat{V}^T$ where $\hat{U} \in \mathbb{R}^{m \times r}$ and $\hat{V} \in \mathbb{R}^{P d_2 \times r}$.

Output: Return the predictor $\hat{f}_{\text{CCNN}}(x) := (\operatorname{tr}(Z(x)\hat{A}_1), \dots, \operatorname{tr}(Z(x)\hat{A}_{d_2}))$ and the convolutional layer output $H(x) := \hat{U}^T (Z(x))^T$.

References

- [1] Özlem Aslan, Hao Cheng, Xinhua Zhang, and Dale Schuurmans. Convex two-layer modeling. In *Advances in Neural Information Processing Systems*, pages 2985–2993, 2013.
- [2] Özlem Aslan, Xinhua Zhang, and Dale Schuurmans. Convex deep learning via normalized kernels. In *Advances in Neural Information Processing Systems*, pages 3275–3283, 2014.
- [3] Yuchen Zhang, Percy Liang, and Martin J Wainwright. Convexified convolutional neural networks. *arXiv preprint arXiv:1609.01000*, 2016.