Report of Project 1
A Review of Convexified Convolutional Neural Network

Xiaodong Jia

June 12, 2017

Contents

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 is the convexification of a two-layer deep neural network work in [1] and [2].

2.1 Convex Two Layer and Deep Learning

A convolutional neural network(CNN) can be written as a function $f(x)$,

$$f : \mathbb{R}^{d_0} \to \mathbb{R}^{d_2},$$

where d_0 is the dimension of input vectors x, d_2 is the number of classes.

Particularly, for a two layer convolutional neural network, the following form separates the trainable parameters and others,

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

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

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

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. $\widehat{A}$ is then transformed to the corresponding parameters of the CNN. As a result, the original non-convex problem is tranformed to a convex one.

3 The Construction of A Two-layer CCNN

A two-layer CCNN can be written as a function $f : \mathbb{R}^{d_0} \to \mathbb{R}^{d_2}$, 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 kth 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 $w_j \in \mathbb{R}^{d_1}$, $\sigma : \mathbb{R} \to \mathbb{R}$ 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 parameters α, w with other constants. Rewritten f_k as

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

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

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

$$f^A := (\mathrm{tr}(ZA_1), \cdots, \mathrm{tr}(ZA_{d_2})),$$

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

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

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

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

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

Thus, we can define a more general CCNN class by

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

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

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

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

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

$$\begin{aligned} f_k(x) &= \sum_{j=1}^{r}\sum_{p=1}^{P} \alpha_{k,j,p} h_j(z_p(x)) \\ &= \mathrm{tr}(QA_k) \\ &= \mathrm{tr}(ZA_k). \end{aligned}$$

Suppose we have computed a result $A \in \mathbb{R}^{m\times Pd_2}$. At test time, given a new input $x \in \mathbb{R}^{d_0}$, we can compute the matrix Z as follows.

Notice that the 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

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

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 $Q \in \mathcal{R}^{nP\times m}$.

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 $\widehat{A} = (\widehat{A}_1, \cdots, \widehat{A}_{d_2})$:

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

5. Compute a rank-r approximation $\tilde{A} \approx \widehat{U}\widehat{V}^T$ where $\widehat{U} \in \mathbb{R}^{m\times r}$ and $\widehat{V} \in \mathbb{R}^{Pd_2\times r}$.

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

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.