Report of Project 1
A Review of Convexified Neural Network

Xiaodong Jia 1601110031
Jianhao Shen 1400010662

June 13, 2017

Contents

1 Introduction

In this article, we review the effort people made to convexify a neural network(NN), in both deep NNs(DNNs) and convolutional NNs(CNNs).

We know that a classical NN as a function is not convex in general, which is known to be NP-hard to solve by the standard approach like stochastic gradient method(SGD). What's more, it has two drawbacks. First, the rate of convergence of SGD can be slow due to the nonconvexity. Second, its statistical properties are very difficult to understandd, as the actual performance is determined by some combination of the CNN architecture along with the optimization algorithm.

We thus find it really interesting to make a CNN convex and address the two drawbacks above. However, as we will see, the convexification of NNs uses relaxation methods, which tranform the original trainable parameters to other relaxed variables in a much larger domain. As a result, the space consuming will soon become unacceptable with the increase of the number of layers, and even a two-layer CNN problem requires much more resource than a classical method.

A brief summary of the related works are as follows. The first work can be traced back to [**?**] in 2006, which gave theotical properties of convexification. Then, a constructional reformulation of a general two-layer model was put forward in [**?**] in 2013. After one year, the convexification of a multi-layer DNN was shown in [**?**]. In 2016, [**?**] gave a two-layer convexified CNN(called CCNN in their article).

The work in [**?**] 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 [**?**], earlier than which is a more generalized two layer modeling in [**?**].

2 A Brief Summary of the Main Ideas

2.1 Convex Neural Networks

The work in [**?**] shows that training an NN can be viewed as a convex optimization problem, but with infinite hidden units. By choosing a regularizer that promotes sparse solutions(e.g. L^1 regularization), the solution will have a finite number of *active* hidden units, but the problem still consists of a large number of variables. An effective approach to this problem is to add one hidden unit at a time and stop once the global optimum is reached so that not all possible hidden units are visited.

2.2 Convex DNNs

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

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

where $l_i = A_i(w_i^T x + b_i), i \in \{1, \cdots, n\}$ 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

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

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.

2.3 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,

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

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

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

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. $\widehat{A}$ 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 Convex NN

Neural Network(NN) is a function of the form $\hat{y}(x) = \sum_{i=1}^{m} w_i h_i(x)$ where x is an input vector, $h_i(x)$ are hidden units and w_i are weight values. Hidden units are obtained from a linear discriminant function: $h_i(x) = s(v_i x + b_i)$ where $s(a)$ is an nonlinear function, e.g. $s(a) = tanh(a)$ or $s(a) = \frac{1}{1+e^{-a}}$. A loss function $Q(y, \hat{y})$ is introduced to measure the mismatches between the prediction $\hat{y}(x_i)$ and the observed y_i. A regularizer $\Omega(w)$ is introduced to favor smaller parameters, e.g. a L^1 regularizer promotes sparse solutions. With notations above, we want to solve the optimization problem:

$$\min_{m,w,v} \Omega(w) + \sum_{i=1}^{n} Q(y_i, \hat{y}_i(x_i)) \tag{1}$$

where $\{(x_i, y_i) : 1 \leq i \leq n\}$ is the observed data set.

This problem is not convex, even if $\Omega(w)$ and $Q(y, \hat{y})$ are convex, since the hidden units $h_i(x) = s(v_i x + b)$ is not convex w.r.t.v_i because of the nonlinear function s(a). Notice that the non-convexity only lies in v_i which determine the chosen hidden units, the problem can be converted into a convex one by including all hidden units in our NN and only tuning w_i to determine both chosen hidden units and weight values(i.e., a hidden unit is chosen iff the corresponding weight value is nonzero). More specifically, we define the "infinite" NN:

Definition 3.1 *Let $\mathcal{H}$ be a set of functions from an input space $\mathcal{X}$ to $\mathbb{R}$. Elements of $\mathcal{H}$ can be understood as "hidden units" in a NN. Let $\mathcal{W}$ be the Hilbert space of functions from $\mathcal{H}$ to $\mathbb{R}$, with an inner product denoted by $a \cdot b$ for a, b $\in \mathcal{W}$. An element of $\mathcal{W}$ can be understood as the output weights vector in a NN. Let $h(x) : \mathcal{H} \to \mathbb{R}$ the function that maps any element h_i of $\mathcal{H}$ to $h_i(x)$. $h(x)$ can be understood as the vector of activations of hidden units when input x is observed. Let $w \in \mathcal{W}$ represent a parameter(the output weights). The NN prediction is denoted $\hat{y}(x) = w \cdot h(x)$. Let $Q : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be a cost function convex in its second argument that takes a scalar target value y and a scalar prediction $\hat{y}(x)$ and returns a scalar cost. This is the cost to be minimized on example pair (x, y). Let $D = \{(x_i, y_i) : 1 \leq i \leq n\}$ a training set.Let $\Omega : \mathcal{W} \to \mathbb{R}$ be a convex regularization function that penalizes for the choice of more "complex" parameters. We define the convex NN criterion $C(\mathcal{H}, Q, \Omega, D, w)$ with parameter w as follows:*

$$C(\mathcal{H}, Q, \Omega, D, w) = \Omega(w) + \sum_{t=1}^{n} Q(y_t, w \cdot h(x_t)) \tag{2}$$

It is easy to see that the convex NN cost $C(\mathcal{H}, Q, \Omega, D, w)$ is a convex function of w, but has infinite hidden units, which cannot be trained. However, with regularizer that promotes sparse solutions like L^1 regularizer, it can be proved that the solution has finite hidden units. We show the special case with $Q(y, \hat{y}) = max(0, 1 - y\hat{y})$ and L^1 regularization, and the training criterion is

$$C(w) = K\|w\|_1 + \sum_{t=1}^{n} max(0, 1 - y_t w \cdot h(x_t)) \tag{3}$$

We can rewrite this cost function as:

$$\min_{w,\xi} K\|w\|_1 + \sum_{t=1}^{n} \xi_t \tag{4}$$

such that

$$y_t[w \cdot h(x_t)] \geq 1 - \xi_t \tag{5}$$

$$\xi_t \geq 0, t = 1, ..., n \tag{6}$$

Then we derive the dual problem (P):

$$\max_{\lambda} \sum_{t=1}^{n} \lambda_t \tag{7}$$

such that

$$\lambda \cdot Z_i - K \leq 0, i \in I \tag{8}$$

$$\lambda_t \leq 1, t = 1, ..., n \tag{9}$$

where $(Z_i)_t = y_t h_i(x_t)$,and I is the index set of hidden units. Such a problem follows Theorem 4.2 from (Hettich and Kortanek, 1993), and the following theorem holds:

Theorem 3.1 *The solution of (P) can be attained with constraints $C_2^{'}$ and only $n+1$ constraints $C_1^{'}$(i.e., there exists a subset of n+1 constraints $C_1^{'}$ giving rise to the same maximum as when using the whole set of constraints). Therefore, the primal problem associated is the minimization of the cost function of a NN with n + 1 hidden neurons.*

This convex NN can be optimized by a stepwise algorithm, which is shown in Algorithm 1:

And it has been proved that Alogrithm convex NN stops when it reaches the global optimum of $C(w)$. However, finding a linear classifier that minimize the weighted sum of classfication errors is an NP-hard problem, which can only be solved efficiently when the input dimension is low. For high dimension input, we can implement approximate minimization, which turns out to perform well in practice.

In another paper [**?**], the author gives a detailed analysis of generalization bound of different prediction functions, which is listed below in Table 1:

4 The Construction of A Three-layer DNN

As mentioned, the problem we want to solve is

$$\min_{W,U,V,\Phi,\Theta} L_1(WX, \Phi) + \frac{1}{2}\|W\|^2 + L_2(U\Phi, \Theta) + \frac{1}{2}\|U\|^2 + L_3(V\Theta, Y) + \frac{1}{2}\|V\|^2. \tag{10}$$

Algorithm 1 Convex(NN)

Input: training set $D = \{(x_1, y_1), ..., (x_n, y_n)\}$, convex loss function Q, and scalar regularization penalty λ.s is either the *sign* function or the *tanh* function.

Set $v_1 = (0, 0, ..., 1)$ and select $w_1 = argmin_{w_1} \sum_t Q(y_t, w_1 s(1)) + \lambda |w_1|$
Set $i = 2$
while True **do**
  Let $q_t = Q^{'}(y_t, \sum_{j=1}^{i-1} w_j h_j(x_t))$
  if s=*sign* **then**
    train linear classifier $h_i(x) = sign(v_i \cdot \tilde{x})$ with examples $\{(x_t, sign(q_t))\}$ and errors weighted by $|q_t|, t = 1, ..., n$.(i.e., $maximize \sum_t q_t h_i(x_t)$)
  else if s=*tanh* **then**
    train linear classifier $h_i(x) = tanh(v_i \cdot \tilde{x})$ to $maximize \sum_t q_t h_i(x_t)$
  end if
  if $\sum_t q_t h_i(x_t) \leq \lambda$ **then**
    stop
  end if
  select $w_1, ..., w_i$(and optimally $v_2, ..., v_i$) minimizing (exactly or approximally) $C = \sum_t Q(y_t, \sum_{j=1}^{i} w_j h_j(x_t)) + \lambda \sum_j |w_j|$ such that $\frac{\partial C}{\partial w_j} = 0$ for $j = 1, ..., i$
  Return the predictor $\hat{y}(x) = \sum_{j=1}^{i} w_j h_j(x)$
end while

(10) 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 (10) corresponding to the second layer,

$$\min_U L(U\Phi, \Theta) + \frac{1}{2}\|U\|^2. \tag{11}$$

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

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

where $K^{\dagger}$ is the Moore-Penrose pseudo-inverse(recall $KK^{\dagger}K = K$ and $K^{\dagger}KK^{\dagger} = K^{\dagger}$). (11) can then be rewritten as

$$\min_Z L(Z, \Theta) + \frac{1}{2}\,\mathrm{tr}(ZK^{\dagger}Z'). \tag{12}$$

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 (11) convex in U, Φ, Θ but (12)

Model	Function form	Generalization bound
No assumption		$n^{-1/(d+3)}logn$
Affine function	$w^\top x + b$	$d^{1/2} \cdot n^{-1/2}$
Generalized additive model	$\sum_{j=1}^k f_j(w_j^\top x), w_j \in \mathbb{R}^d$	$kd^{1/2} \cdot n^{-1/4}logn$
Single-layer neural network	$\sum_{j=1}^k \mu_j(w_j^\top x + b_j)_+$	$kd^{1/2} \cdot n^{-1/2}$
Projection pursuit	$\sum_{j=1}^k f_j(w_j^\top x), w_j \in \mathbb{R}^d$	$kd^{1/2} \cdot n^{-1/4}logn$
Dependence on subspace	$f(W^\top x), W \in \mathbb{R}^{d\times s}$	$d^{1/2} \cdot n^{-1/(s+3)}logn$

Table 1: Summary of generalization bounds for various models.

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 $Z = U\Phi$, (11) 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 $S \in \Theta'\mathbb{R}\Phi = N\mathbb{R}K$), (11) can be rewritten as

$$\min_S L^u(S, N) + \frac{1}{2}\operatorname{tr}(K^\dagger S' N^\dagger S). \tag{13}$$

This objective is jointly convex in the propensity matrix S and output kernel N. Notice that N is positive semidefinite. Thus, to make (11) 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 [**?**], which is only applicable for a single hidden layer network. A more generalized approach in [**?**] is as follows.

Let's reconsider (11). This time we change the form of it to

$$\min_U L(U\Phi, \Theta) + \frac{1}{2}\|\Theta' U\|^2. \tag{14}$$

One can see we only change the regularization term $\|U\|^2$ to $\|Theta' U\|$. A key fact is that U still satisfy the representer theorem under this change, so we have $U = (\Theta\Theta')^\dagger A\Phi'$ for some A. Generally speaking, Θ has full row rank, so we can use a change of variables $A = \Theta B$, and our regularization becomes

$$\|\Theta' U\|^2 = \|\Theta'(\Theta\Theta')^\dagger A\Phi'\|^2 = \|\Theta'(\Theta\Theta')^\dagger \Theta B\Phi'\|^2.$$

Now, we define $M := \Theta'(\Theta\Theta')^\dagger\Theta$ to be the *normalized output kernel*. The term $(\Theta\Theta')^\dagger$ essentially normalizes the spectrum of the kernel $\Theta'\Theta$, and, since

$M^2 = M$, we must have the eigenvalues of M, $\lambda_M = \{0, 1\}$. Finally, the regularization term can be rewritten as

$$\|MB\Phi'\|^2 = \mathrm{tr}(MBKB'M) = \mathrm{tr}(MBKK^\dagger KB'M) = \mathrm{tr}(SK^\dagger S'),$$

where $S = MBK$. Another important fact is that we have

$$S = MBK = \Theta'(\Theta\Theta')^\dagger \Theta BK = \Theta'(\Theta\Theta')^\dagger AK = \Theta'(\Theta\Theta')^\dagger A\Phi'\Phi = \Theta' U\Phi.$$

Now, to make the form convex, we need other structures on L. We rewrite Postulate 1 so that it holds for *normalized* kernel matrix.

Postulate 3. The loss $L(Z, \Theta)$ can be written as $L^n(\Theta' Z, \Theta'(\Theta\Theta')^\dagger)$, where L^n is jointly convex in both arguments. Here the n of L^n emphasizes the use of normalized kernels.

By Postulate 3, (14) can be rewritten as

$$L^n(S, M) + \frac{1}{2}\mathrm{tr}(SK^\dagger S'), \quad \text{where } S \in M\mathbb{R}K. \tag{15}$$

which is jointly convex in S, M, K. Compared to (13), one can see that the output kernel N has been removed from the regularization. What's more, the feasible region $\{(S, M, K) : M \succeq 0, K \succeq 0, S \in M\mathbb{R}K\}$ is also convex.

Now, we can apply (12) to the first two layers and (15) to the output layer, so that the objective in (10) becomes

$$\begin{aligned} &L_1^n(S_1, M_1) + \frac{1}{2}\mathrm{tr}(S_1 K^\dagger S_1') \\ +&L_2^n(S_2, M_2) + \frac{1}{2}\mathrm{tr}(S_2 M_1^\dagger S_2') \\ +&L_3(Z_3, Y) + \frac{1}{2}\mathrm{tr}(Z_3 M_2^\dagger Z_3'), \end{aligned} \tag{16}$$

where $S_1 \in M_1\mathbb{R}K$, $S_2 \in M_2\mathbb{R}M_1$, and $Z_3 \in \mathbb{R}M_2$.

Recall that we should make the domain of M convex and the loss functions L^n satisfy postulate 3.

The output kernels $M := \Theta'(\Theta\Theta')^\dagger\Theta$ has no convex domain in general. We want to relax M, such that they preserve the nice properties including

$$M \succeq 0, M \preceq I, \mathrm{tr}(M) = \mathrm{tr}((\Theta\Theta')^\dagger(\Theta\Theta')) = \mathrm{rank}(\Theta\Theta') = \mathrm{rank}(\Theta),$$

for any Θ. Here, $M \succeq 0$ denotes that M is a symmetric semi-positive matrix. So, given a number of hidden nodes, h, we will enforce $\mathrm{tr}(M) = h$. We further relax the property $\lambda_M = \{0, 1\}$ to $\forall\lambda_i \in \lambda_M$, $0 \le \lambda_i \le 1$. A final relaxation would be $M\mathbf{1} = \mathbf{1}$, which holds when Θ expresses target values for a multiclass classification, i.e. $\Theta_{ij} \in \{0, 1\}$, $\Theta'\mathbf{1} = 1$, representing each index of the cluster.

We then relax M to the followings domain,

$$\mathcal{M} := \{M | 0 \preceq M \preceq I, M\mathbf{1} = \mathbf{1}, \mathrm{tr}(M) = h\}. \tag{17}$$

The last thing is to give a appropriate loss function. An example was put in [**?**], which use a large margin, multi-label loss

$$\tilde{L}(z, y) = \max(1 - y + kz - \mathbf{1}(y'z))$$

where z is a linear response and $y \in \{0, 1\}^h$ is the target vector. This example can be adapted to a normalized case as follows. We first generalize the notion of margin to consider a *normalized label* $(YY')^\dagger y$:

$$L(z, y) = \max(1 - (YY')^\dagger y + kz - \mathbf{1}(y'z)).$$

Then, given t input-output pairs (Z, Y), we can rewrite the loss to a compact form as

$$L(Z, Y) = \sum_j L(z_j, y_j) = \tau(kZ(YY')^\dagger Y) + t\,\mathrm{tr}(Y'Z),$$

where $\tau(X) := \sum_j \max_i X_{ij}$. As a result, the loss below can be shown to satisfy Postulate 3,

$$L^n(S, M) = \tau(S - \frac{1}{k}M) + t - tr(S), \quad \text{where } S = Y'Z, M = Y'(YY')^\dagger Y. \tag{18}$$

Thus, by (16), (17) and (18), we reformulate the original three-layer NN problem to a convex form.

5 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 $z_p(x) \in \mathbb{R}^{d_1}, 1 \le p \le P$.

Then, each patch is transformed to r scalars, which can be written as $h_j(z_p) = \sigma(w_j^T z_p), 1 \le j \le r$, 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 parame-

ters α, w with other constants. Rewritten f_k as

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6 Numerical Results

The source code of CCNN is available at CCNN on GitHub, which implements the CCNN applied to MNIST and Cifar-10 dataset.

As noted in the homepage, the current CCNN implementation caches all feature vectors in the memory, which results in a high memory requirement (about 10GB for the mnist datasets, about 50GB for the cifar10 dataset). What's more, the whole process for Cifar-10 takes about 1.5 hours but it doesn't make use of GPU, which will hopefully accelerate the process.

The Cifar-10 program reaches an accuracy of 94.92% on the training dataset, while MNIST 96.75%.

The Cifar-10 program reaches an accuracy of 79.25% on the testing dataset, while MNIST 69.62%.

One can see that althogh the training data reaches a good point, CCNN shows poor robust on the testing dataset.

7 Summary

In this report, we reviewed several important convexification methods in recent years, which require tricky reformularization of the original problems, and introduce large space requirement. In particular, the performance of CCNNs is not as well as a traditional method in both time consuming and robust. Further research could be conduct on how to increase the robust of CCNNs and how to reduce the space cost.