Convexified Convolution Neural Networks

Group 16

-K.Sravanthi
-M.Megha
-B.Kaushik
-R Harsha

Introduction:

Convolutional neural network (**CNN**) is a class of deep, feed-forward artificial neural networks that has successfully used in image classification,face recognition,speech recognition,text classification and game playing.

However,training a CNN is solving a non-convex optimization problem which is NP Hard.

Also,training a CNN via back-propagation has 2 drawbacks

1)Rate of convergence

2)Difficulty in understanding statistical properties of CNN

Due to non-convexity,the rate of convergence can be slowed down,which the author has tried to overcome by convexifying the network(which also gives us detailed information of the statistical properties).

These convexified networks are known as Convexified Convolutional Neural Networks(CCNNs).Also,the complexity of these networks is significantly lower than CNN(or fully connected neural network)

In this paper ,the author has 2 main objectives:

1)Formulate CCNNs

2)Show that generalization error of CCNN model is comparable to the best of

CNN models

Previous Works:

-Formulating neural networks as convex optimization method,using infinite parameters(Bengio et al)

-A method for learning multilayer latent variable models in which,for certain activation functions,it is a convex relaxation for learning fully connected neural networks(Aslan et al)

-Convolution Kernel Networks(Mairal et al)

-Scatnet method(Bruna and Mallat)

-Randomly initialized CNN can extract features as powerful as kernel methods(Daniel et al)

Notations used:

$||A||_*$ -nuclear norm-sum of singular values of A

$||A||_2$ -spectral norm -square root of max eigen values of A^HA (A^H -conjugate transpose).

$\langle u,v \rangle := \sum_{i=1}^{\infty} u_i v_i$

The Basic CNN problem setup(2 Layer)

1)Split the input vector x into P patches $\{z_p(x)\}_{p=1}^{P}$.

2)Choose an activation function $\sigma:\mathbb{R} \rightarrow \mathbb{R}$,and collection of weight vectors

$\{w_j\}_{j=1}^{r}$

3)Define a filter function h_j $\{j \in [r]\}$ for each patch z.Same filters are applied to each patch(parameter sharing).

4)For each patch index $p \in [P]$,filter index $j \in [r]$,and output coordinate

$k \in [d_2]$,introduce a coefficient $\alpha_{k,j,p} \in \mathbb{R}$ which governs contribution of the filter h_j on patch $z_p(x)$ to output $f_k(x)$.

Final form of CNN is a vector containing all the individual functions where kth component is given by

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

5)For $f_k(x)$,we do empirical risk minimization by using a loss function (that is convex and L-Lipschitz).This optimization problem is non-convex(as $f_k(x)$ is non-linearly related to α and w.

6)We define some constraints on them

Therefore ,we try to convexify this CNN.

We basically embed the non linear problem into a reproducing kernel hilbert space ,which we reduce to a linear setting.Firstly,we show the convexifying for linear activation function

Reproducing Kernel Hilbert space(RKHS):

An RKHS is a hilbert space of functions in which point evaluation is a continuous linear functional.

An RKHS is associated with a kernel that reproduces every function in the space ,in the sense that for any x in the set,on which the functions are defined ,”evaluation at x” can be performed by taking and inner product with a function determined by the kernel.

Such a reproducing kernel exists if and only if every evaluation functional is continuous.

Case 1:Linear Activation Function

1)Let the activation function be $\sigma(t)=t$.

2)The filter h_j ,when applied to the patch vector $z_p(x)$,outputs a euclidean inner product of the form $\mathbf{h_j(z_p(x))= <z_p(x),w_j>}$ (h_j is already defined before).

3)Define a patch matrix (P * *d1 dimensional*)Z(t),which contains the P input patches.

4)Therefore, the final form can be rewritten as

$$\mathbf{f_k(x) = \sum^{r}_{j=1}\sum^{P}_{p=1}\alpha_{k,j,p}<z_p(x),w_j> = \Sigma^{r}_{j=1}\alpha^{T}_{k,j}\, Z(x)w_j}$$

$$\mathbf{tr(Z(x)(\sum^{r}_{j=1}w_j\alpha^{T}_{k,j}) = tr(Z(x)A_k)}$$

5)We see that every kth component of the function vector depends linearly on matrix parameter A_k .This has a rank of atmost ‘r’ (due to parmeter sharing of CNN)

6)Vector A consists of all the matrix parameters,and we define a function(

$f^A : \mathbb{R}^{d1} \rightarrow \mathbb{R}^{d2}$),which contains all the output components.

7)This model class is based on some constraints.

-If B_1 and B_2 are positive radii, then,$F_{CNN}(B_1,B_2)$ to be a set of functions f^A which satisfies constraints

(C1)$\max_{j \in [r]} ||w||_2 <= B_1$

(C2) rank(A)=r

Any matrix A satisfying the constraint C_1 and C_2 must have nuclear norm bounded as $||A||_* <= B_1 B_2 r\sqrt{d_2}$.

Therefore we define the function class F_{CCNN} as :

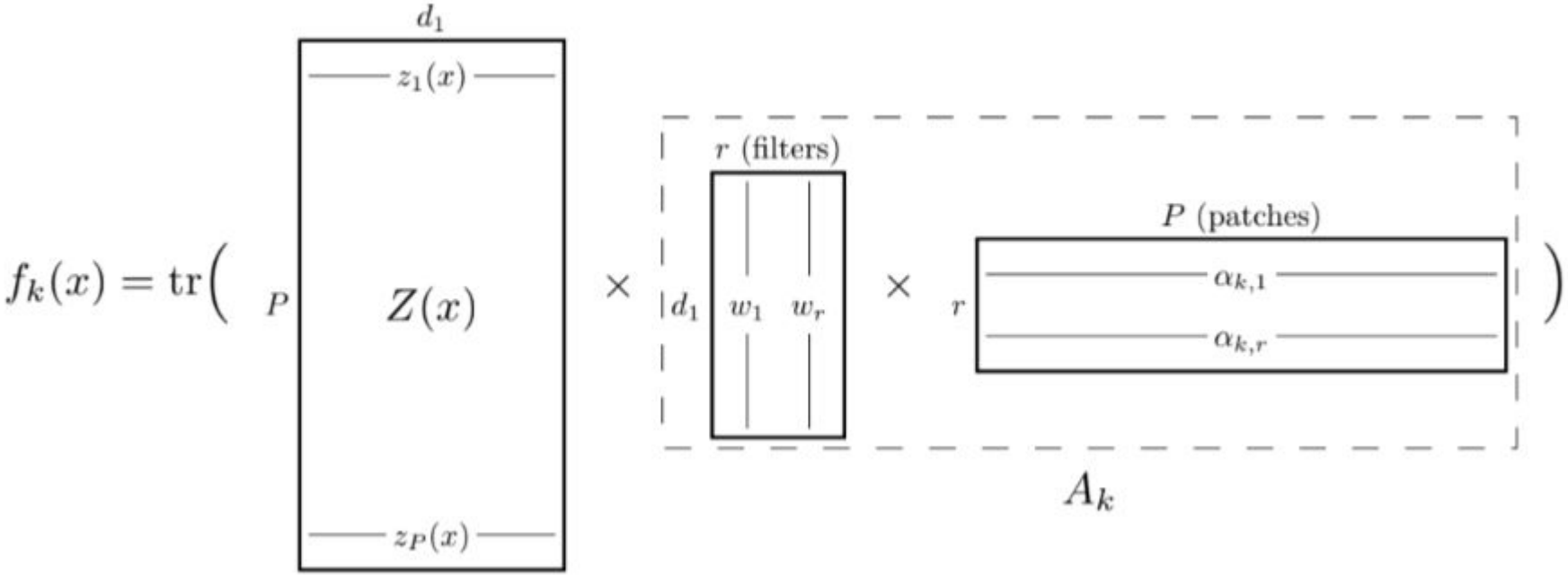
$$\mathbf{F_{CCNN} := \{ f^A : ||A||_* <= B_1 B_2 r\sqrt{d_2} \}}$$

which guarantees $F_{CNN} \subseteqq F_{CCNN}$

7)We then try to minimize empirical risk minimization over F_{CCNN}

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

Output expressed as matrix product



Case 2: Non-linear activations:RKHS filters

For non-linear activation function ,we relax the class of the CNN filters to an RKHS.

We define a positive semi-definite kernel function $\mathcal{K}$ .With this ,and a sufficiently smooth activation function we can show that the filter ‘h’ is contained in RKHS induced by $\mathcal{K}$.

Let $S := \{ z_p (x_i) : p \in [P] , i \in [n] \}$ be a set of patches in the training data set.

According to the representer theorem,function value for any patch $z_p (x_i) \in S$ can be represented as:

--Let $\mathbf{K} \in \mathbb{R}^{np \times np}$ be a symmetric kernel matrix.To optimize the number of calculations, we factorize $K = QQ^T$,where $Q \in \mathbb{R}^{np \times m.}$

We interpret $Q_{(i,p)}$ as a feature vector in place of original patch vector $z_p(x_i)$,and replace in the filter equation from earlier as:

$$h(z_p(x_i)) = \langle Q_{(i,p)}, w \rangle \quad \text{where} \quad w := \sum_{(i',p')} c_{i',p'} Q_{(i',p')}.$$

It suffices to know about the vector “w” to learn the fiter “h”.

--We compute a patch matrix for the training data,and then test data.If input given is x,

-The p-th row of this matrix is the feature vector for patch p, which is equal to $Q^{\dagger} v(z_p(x)) \in R\ m$, where for any patch z, the vector v(z) is defined as a nP-dimensional vector whose (i, p)-th coordinate is equal to $K(z, z_p(x_i))$.

-Then compute the predictor.

Algorithm 1: Learning two-layer CCNNs

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) and row (i', p') is equal to $\mathcal{K}(z_p(x_i), z_{p'}(x_{i'}))$. Compute a factorization $K = QQ^\top$ or an approximation $K \approx QQ^\top$, where $Q \in \mathbb{R}^{nP \times m}$.

2. For each x_i, construct patch matrix $Z(x_i) \in \mathbb{R}^{P \times m}$ whose p-th row is the (i, p)-th row of Q, where $Z(\cdot)$ is defined in Section 3.2.

3. Solve the following optimization problem to obtain a matrix $\widehat{A} = (\widehat{A}_1, \ldots, \widehat{A}_{d_2})$:

$$\widehat{A} \in \operatorname*{argmin}_{\|A\|_* \le R} \widetilde{\mathcal{L}}(A) \quad \text{where} \quad \widetilde{\mathcal{L}}(A) := \sum_{i=1}^n \mathcal{L}\Big(\big(\operatorname{tr}(Z(x_i)A_1), \ldots, \operatorname{tr}(Z(x_i)A_{d_2})\big); y_i\Big). \tag{12}$$

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

Output: Return the predictor $\widehat{f}_{\mathrm{ccnn}}(x) := \big(\operatorname{tr}(Z(x)\widehat{A}_1), \ldots, \operatorname{tr}(Z(x)\widehat{A}_{d_2})\big)$ and the convolutional layer output $H(x) := \widehat{U}^\top (Z(x))^\top$.

Activation function and kernel functions used:

Kernel functions used - Gaussian kernel and inverse polynomial

Activation functions used -we have to choose such that the generalization error of F_{CCNN} is comparable to that of best CNN models.

We consider following types of activation functions:

(a) arbitrary polynomial functions
(b) sinusoid activation function σ(t) := sin(t)
(c) erf function σ erf (t) := $2/\sqrt{\pi} \int_0^t e^{-z^2} dz$ (approximation to sigmoid)
(d) a smoothed hinge loss σ sh (t) := $\int_{-\infty}^{t} 0.5 (\sigma_{erf} (z) +1) dz$

Inverse polynomial kernel captures all the 4 categories (a,b,c,d,).

Gaussian kernel captures only the first 2 categories(a,b).

***sigmoid and ReLU functions are not used because they fail to converge quickly**

Generally,by choosing a hyperparameter R in the given algorithm,expected error is at most where R= $C_\sigma(B_1)B_2r$

$$\mathbb{E}_{X,Y}[\mathcal{L}(\widehat{f}_{ccnn}(X);Y)] \leq \inf_{f\in\mathcal{F}_{cnn}} \mathbb{E}_{X,Y}[\mathcal{L}(f(X);Y)] + \frac{c\,LC_\sigma(B_1)B_2r\sqrt{\log(nP)\,\mathbb{E}_X[\|K(X)\|_2]}}{\sqrt{n}},$$

where $c > 0$ is a universal constant.

Learning multi-layer CCNNs:

For models with more than 1 hidden layer,the CCNN theory doesn't apply .

We need to compute the filters explicitly unlike in the 2-layer case.

We approximate the parametric matrix A as a rank r approximation $A \doteq \hat{U} \hat{V}^T$ and set the jth filter h_j to the mapping

$$z \mapsto \langle \hat{U}_j, Q^{\dagger} v(z) \rangle$$

We apply all the r filters to all patches of the input.The resulting output is :

$$H(x) := \hat{U}^T Z(x)^T \quad \{r * P \text{ vector}\}$$

For multilayer CCNNs , two important tasks are average pooling and processing of multi channel inputs.

Average pooling reduces the output dimension of convolution layer.Because of the linearity of CCNN, pooling can be done before convolution.

Processing multi channel inputs: a multi channel patch vector is defined as a concatenation of patch vector for each channel.we form the feature matrix Z(x) using these concatenated patch vectors and continue the process as in the previously given algorithm.

Algorithm 2: Learning multi-layer CCNNs

Input: Data $\{(x_i, y_i)\}_{i=1}^n$, kernel function $\mathcal{K}$, number of layers m, regularization parameters $R_1, \ldots, R_m$, number of filters $r_1, \ldots, r_m$.

Define $H_1(x) = x$.

For each layer $s = 2, \ldots, m$:

- Train a two-layer network by Algorithm 1, taking $\{(H_{s-1}(x_i), y_i)\}_{i=1}^n$ as training examples and R_s, r_s as parameters. Let H_s be the output of the convolutional layer and $\widehat{f}_s$ be the predictor.

Output: Predictor $\widehat{f}_m$ and the top convolutional layer output H_m.

Cifar-10 dataset results

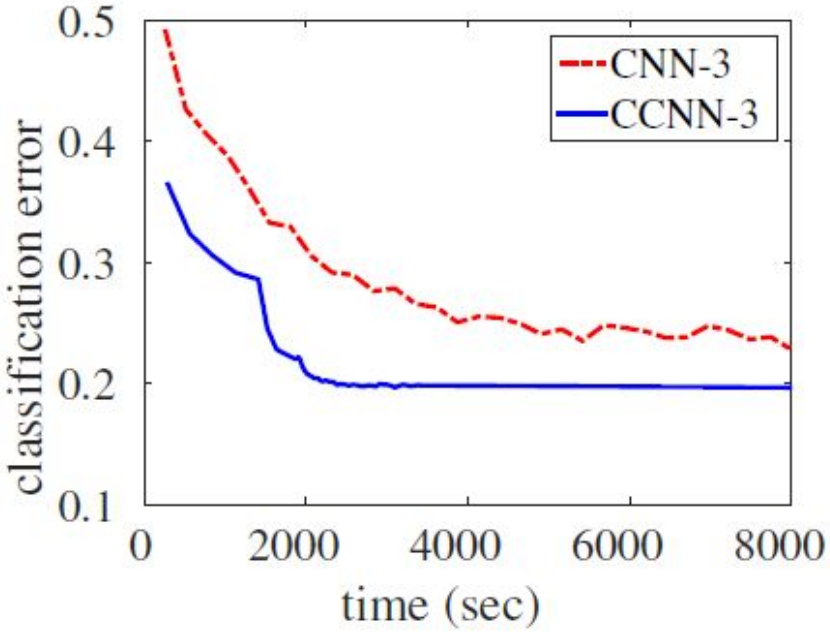


Fig : The convergence of CNN-3 and CCNN-3 on the CIFAR-10 dataset.

CNN-3 represents convolution neural networks with 3 hidden layers.

THANK YOU