Introduction to Convolutional Neural Network

Huimin Ye, 11531016

Abstract

This paper introduces the architecture and inference of convolutional neural network (CNN). Commencing with a description of its multi-layer structure, followed by the derivation of feedforward computation. For the convenience of employing gradient descent method to such an unconstrained minimization, backpropagation which is an efficient algorithm is inferred in detail.

Index Terms

inference, convolution, feedforward, backpropagation, gradient descent.

I. Introduction

Currently, deep learning attracts a great deal of attention especially the researches on Convolutional Neural Network (CNN). Although the initial proposal and application of CNN [1] was in 1995, convolutional neural network was in silence in last decade as if its development was somehow stuck. However, convolutional neural network is prevalent as Artificial Intelligence becomes a hot spot in the research area of Computer Science. This paper discusses the derivation of training approach of CNN concretely.

II. Existing Work

A typical CNN known as LeNet was suggested by LeCun et al. [1] [2] in 1995, which had a great performance on identification of handwritten digits so that it was used by banks to process cheques, and by post offices to recognize addresses. Ciresan et al. [3] exceeded previous results and first achieved near-human performance on recognition digits, in 2012. During the same period, Krizhevsky and Hinton [4] designed a deep convolutional neural network and acquired the top 1 in ImageNet Large Scale Visual

Huimin Ye is a PhD. candidate with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China. e-mail: 1042430841@qq.com.

This report is updated on April 19, 2016.

Recognition Challenge (ILSVRC) 2012, which revealed the power of CNN to other participants. Moreover, a special CNN referred as DeepID [5] was employed to face verification and obtained the state-of-the-art accuracy on LFW dataset. The tasks of object detection and semantic segmentation also gained considerable improvement through the proposals of R-CNN [6] and Fast R-CNN [7] sequentially, in 2014 and 2015 respectively.

III. Architecture

Typically convolutional neural network (see Fig. 1) is composed of pairs of convolution layer and pooling layer with a fully connected layer at the last stage of the architecture. The input of network is a two-dimensional image and its feedforward process contains convolution and pooling (also subsampling) operations that generate the corresponding layers' feature maps. After hierarchical feedforward steps,

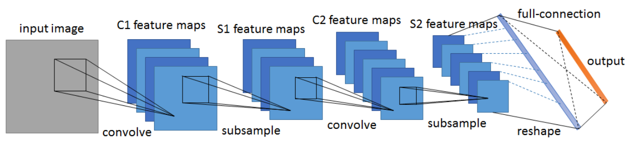

Fig. 1. A convolutional neural network architecture

the feature maps before the fully connected layer which have smaller size than primitive input are all concatenated into a long vector. Considering the classification task, the output of network possesses k units (the class number for discrimination) that are organized as a "one-of-k" code where only one element of output is positive and the rest of them are either zero or negative depending on the choice of output activation function. Like other artificial neural networks, CNN has numerous adjustable parameters, namely weights and biases (notation $\boldsymbol{W}$ and $\boldsymbol{b}$), respectively. Error backpropagation algorithm which is efficient in multi-layer structure is described below to interpret the approach of updating the parameters concretely.

IV. Feedforward Pass

A. Convolution Layer

At a convolution layer ℓ, the previous layer's feature maps (or original input) $\boldsymbol{x}_i^{\ell-1}$ are convolved with adjustable kernels (also weights) $\boldsymbol{k}_{ij}^{\ell}$ and put through the activation function to form the output feature map. In general, we have that

$$\boldsymbol{x}_j^{\ell} = f\left(\sum_{i\in M_j} \boldsymbol{x}_i^{\ell-1} * \boldsymbol{k}_{ij}^{\ell} + b_j^{\ell}\right), \tag{1}$$

where M_j represents a selection of input maps, e.g. pairs or triplets of combinations. Each output map is given an additive bias b, but for a particular output map, the input maps will be computed with distinct kernels, see Fig. 2. The activation function $f(\cdot)$ is commonly chosen to be the sigmoid function

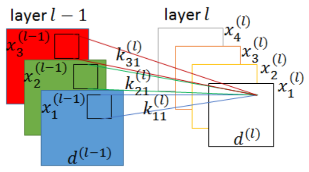

Fig. 2. Convolution operation

$f(x) = a/(1+e^{-bx})$ or the hyperbolic tangent function $f(x) = a\tanh(bx)$. In addition, it is convenient that the derivatives of both may be represented by themselves, i.e.,

$$\text{sigmoid} \quad f'(x) = f(x)(1-f(x)), \tag{2}$$

$$\text{tanh} \quad f'(x) = 1 - f^2(x). \tag{3}$$

B. Pooling Layer

A pooling layer produces a downsampled version of the input maps. The subsampling operation does not alter the number of feature maps but the sizes of them. That is, N input maps will obtain exactly N output maps with a smaller dimension. More formally,

$$\boldsymbol{x}_j^{\ell} = f(\text{down}(\boldsymbol{x}_j^{\ell-1}) + b_j^{\ell}), \tag{4}$$

where down$(\cdot)$ is a pooling (or subsampling) operation. Specifically this function will extract the average (mean pooling) or the maximum (max pooling) over each distinct n-by-n block in an input map so that the corresponding output map is n-times smaller along both spatial dimensions but preserves the characteristics simultaneously. Each output map also plus an additional bias b. Fig. 3 illustrates this procedure.

In the derivation that follows, we will consider the squared-error loss function. For a multi-class problem with c class and N training examples, this error is given by

$$E^N = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}(t_k^n - y_k^n)^2. \tag{5}$$

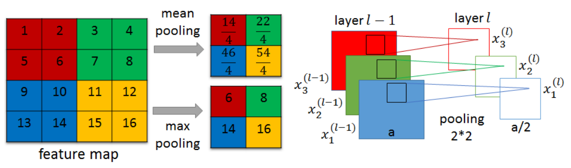

Fig. 3. Pooling operation

Here t_k^n is the k-th dimension of the n-th pattern's corresponding label, and y_k^n is similarly the value of the k-th output layer unit in response to the n-th input pattern. Because this error over the entire dataset is just a sum over the individual errors on each pattern, we will consider it with respect to a single pattern, i.e. the n-th one,

$$E^n = \frac{1}{2}\sum_{k=1}^{c}(t_k^n - y_k^n)^2 = \frac{1}{2} \parallel \boldsymbol{t}^n - \boldsymbol{y}^n \parallel_2^2 . \tag{6}$$

Then we formulate our problem to the standard form as follows,

$$\text{minimize} \quad \sum_n \frac{1}{2} \parallel \boldsymbol{t}^n - \boldsymbol{y}^n \parallel_2^2 . \tag{7}$$

Note that there is no constraint though $\boldsymbol{y}^n$, which is the output vector of last layer, can be represented as a complicated function with respect to variables W, b. Because of the hierarchical convolution and pooling, we are not able to understand the relationship between $\boldsymbol{y}^n$ and W, b, directly. In convolutional neural network, tiny changes of W somehow may lead to obvious promotion or deterioration on its performance, but gigantic variations of W may not work in the same way. In brief, it is difficult to comprehend or prove whether the problem proposed above is convex or not.

Though the convexity of our problem remains unknown (i.e., we can not find the global optimal solutions W, b straightforwardly), it is still possible to employ the gradient descent method, one of the effective approaches for unconstrained minimization, to solve the problem.

With a variety of W and b, we do not insure that W and b converge eventually to their global optimal solutions instead of local optimal solutions, respectively. In practice, it is quite probable to get to the latter case.

V. Backpropagation Pass

In short, we update the parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ in the light of rules given by

$$\boldsymbol{W}_{\tau+1} = \boldsymbol{W}_\tau - \eta\frac{\partial E}{\partial \boldsymbol{W}_\tau}, \tag{8}$$

$$\boldsymbol{b}_{\tau+1} = \boldsymbol{b}_\tau - \eta\frac{\partial E}{\partial \boldsymbol{b}_\tau}. \tag{9}$$

Here τ denotes the number of iterations and learning rate (notation η) is usually a small value. Since multi-layer architecture results in the expressions of partial derivatives with respect to $\boldsymbol{W}, \boldsymbol{b}$, in former layer, being complex considerably through the chain rule, we use error backpropagation algorithm to simplify them.

The “error” which propagates backwards through the network can be thought of as “sensitivities” with respect to the current total input of each unit, yields

$$\boldsymbol{\delta}^{\ell} = \frac{\partial E}{\partial \boldsymbol{u}^{\ell}}, \quad \boldsymbol{u}^{\ell} = \boldsymbol{W}^{\ell}\boldsymbol{x}^{\ell-1} + \boldsymbol{b}^{\ell}, \quad \boldsymbol{x}^{\ell} = f(\boldsymbol{u}^{\ell}). \tag{10}$$

A. *Fully connected Layer*

We obtain the error and corresponding gradients of the last layer without much effort,

$$\boldsymbol{\delta}^{L} = (\boldsymbol{y}^{n} - \boldsymbol{t}^{n}) \circ f'(\boldsymbol{u}^{L}), \tag{11}$$

$$\frac{\partial E}{\partial \boldsymbol{W}^{L}} = \boldsymbol{\delta}^{L}(\boldsymbol{x}^{L-1})^{T}, \quad \frac{\partial E}{\partial \boldsymbol{b}^{L}} = \boldsymbol{\delta}^{L}. \tag{12}$$

In (11), “$\circ$” denotes the Hadamard product (i.e. element-wise multiplication). From that, the partial derivative with respect to $\boldsymbol{W}$ is computed as an outer product between the vector of input and sensitivities, and more compact for $\boldsymbol{b}$. Then this error is passed to previous layer by means of the recurrence relation below,

$$\boldsymbol{\delta}^{L-1} = (\boldsymbol{W}^{L})^{T}\boldsymbol{\delta}^{L} \circ f'(\boldsymbol{u}^{L-1}). \tag{13}$$

B. *Pooling Layer*

Having acquired the sensitivities for each map of the pooling layer ℓ, we immediately compute the gradient for $\boldsymbol{b}$ by summing over all elements simply in a sensitivities map,

$$\frac{\partial E}{\partial b_j} = \sum_{u,v} (\boldsymbol{\delta}_j^{\ell})_{u,v}. \tag{14}$$

Assuming that it is a convolution layer $\ell - 1$ that is distributed before a pooling layer ℓ. Hence a block of pixels in convolution layer’s map is associated with one pixel in present layer’s map. In case of mean pooling operation, to propagate sensitivities back efficiently, we extend the sensitivity map’s dimension (and average its value) in layer ℓ to make it exactly same size as prior layer $\ell - 1$, then multiply this quantity element-wise by the derivative of activation function evaluated at $\ell - 1$ layer’s pre-activation input, $\boldsymbol{u}$. We can repeat the same computation for each map j between two layers, see Fig. 4 for details.

$$\boldsymbol{\delta}_j^{\ell-1} = up(\boldsymbol{\delta}_j^{\ell}) \circ f'(\boldsymbol{u}_j^{\ell-1}), \tag{15}$$

$$up(\boldsymbol{x}) = \boldsymbol{x} \otimes \mathbf{1}_{n\times n}/n^2. \tag{16}$$

Note that "$\otimes$" represents the Kronecker product, and $up(\cdot)$ denotes an upsampling operation that duplicates each pixel in the input horizontally and vertically n times to the output (i.e. to be a $n \times n$ matrix) when the factor of pooling is n.

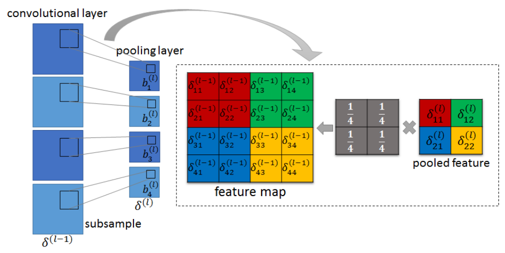

Fig. 4. Backpropagation in pooling layer

C. Convolution Layer

In case of obtained the sensitivities for each map of the convolution layer ℓ, we compute the gradients for kernel weights $\boldsymbol{k}_{mn}^{\ell}$ and $\boldsymbol{b}$ using backpropagation where weights are shared across many connections. Thus, we sum the gradients, for a given weight $\boldsymbol{k}_{mn}^{\ell}$, over all connections that are related to this weight. We do not repeat interpreting the gradients for $\boldsymbol{b}$ due to the same method of calculation showed above.

$$\frac{\partial E}{\partial \boldsymbol{k}_{mn}^{\ell}} = \sum_{u,v} (\boldsymbol{\delta}_j^{\ell})_{uv} (\boldsymbol{p}_{mn}^{\ell-1})_{uv}, \quad \frac{\partial E}{\partial b_j} = \sum_{u,v} (\boldsymbol{\delta}_j^{\ell})_{u,v}. \tag{17}$$

In (17), the *patch* denoted by $\boldsymbol{p}_{mn}^{\ell-1}$ is a part matrix of $\boldsymbol{x}_{\boldsymbol{i}}^{\ell-1}$ that was multiplied element-wise by $\boldsymbol{k}_{mn}^{\ell}$ during convolution in feedforward pass in order to compute the corresponding units in the output map $\boldsymbol{x}_j^{\ell}$. Fig. 5 demonstrates this process concretely.

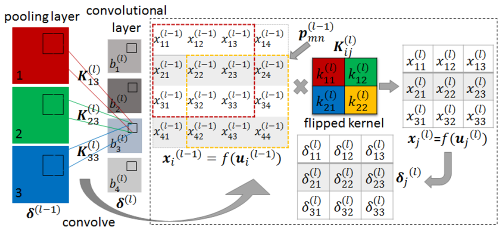

Fig. 5. Backpropagation in convolution layer

Supposing that a pooling layer $\ell - 1$ is followed by a convolution layer ℓ. The difficulty lies in computing the sensitivity maps that propagates back to the previous layer $\ell - 1$. Here, we figure out how a given unit in the previous layer's sensitivity map $\boldsymbol{\delta}_i^{\ell-1}$ participates in generating the patch in the current layer's sensitivity map $\boldsymbol{\delta}_j^{\ell}$ through kernels mentioned. Therefore, we apply a recursion rule that looks something like equation (13). Observe that the weights multiplying the connections between two layers are exactly the rotated convolution kernels. The operation goes below

$$\boldsymbol{\delta}_i^{\ell-1} = (\mathrm{rot}180(\boldsymbol{K}_{ij}^{\ell}) \circledast \boldsymbol{\delta}_j^{\ell}) \circ f'(\boldsymbol{u}_i^{\ell-1}), \tag{18}$$

where "$\circledast$" denotes a full convolution operation that will automatically pad the missing inputs with zeros. Notice that the kernel $\boldsymbol{K}_{ij}^{\ell}$ is rotated before to make the convolution perform cross-correlation actually. Fig. 6 yields the detailed procedure.

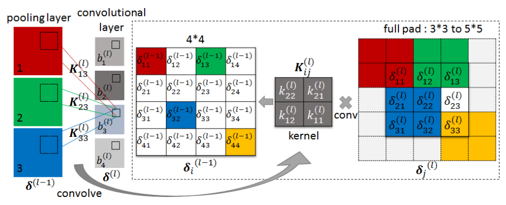

Fig. 6. Sensitivities backpropagation process

VI. Conclusion

After analyzing sequential steps of feedforward and backpropagation separately, we conclude that the training approach of convolutional neural network through gradient descent method is summarized as follows

```
Training procedure of Convolutional Neural Network
Initialize weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ in each layer with stochastic values.
Repeat
    1. Compute activation forwards through every element in each layer.
    2. Evaluate Loss function $E$ of the last layer's output.
    3. Compute backwards the gradients for weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ in each layer, respectively.
    4. Update parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ through gradient descent method.
Until
    Loss function $E$ is smaller than threshold pre-determined, or the maximum number of iterations reaches.
```

References

[1] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger *et al.*, "Comparison of learning algorithms for handwritten digit recognition," in *International conference on artificial neural networks*, vol. 60, 1995, pp. 53–60.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3642–3649.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[5] Y. Sun, X. Wang, and X. Tang, "Deep learning face representation from predicting 10,000 classes," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[7] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1440–1448.