Convotional Neural Networks

Khanh Nguyen

June 22, 2015

Convolutional neural networks (CNN) is one of the major successes that initiate the resurrection of neural networks in the last decade. The model revolutionizes the traditional framework in computer vision, which heavily depended on manually feature-engineering. CNNs automate feature-engineering and only takes raw image pixels as input but still outperforms traditional kernel-based models (e.g. SVM) by large margins. Particularly, CNNs participated in the ImageNet object recognition contest in 20XX and surprised the community by reducing the error rate a incredible XX%.

1 Motivation

Images are extremely high-dimension objects if we consider each pixel to be a feature. Using a fully-connected neural networks for an image would significantly increase the amount of computation. More seriously, such a model would have a gigantic set of paramters and thus is very susceptible to overfitting. To address this problem, CNNs are designed as sparsely connected neural networks. Each network layer is partitioned into a set of feature dectectors, often called *filters*. The input image can be considered as a filter itself. Practically, for RBG images, we have three filters per image, corresponding to values for red, green, and blue. The numbers of filters of the hidden layers are determined by the model. A feature dectector computes features on every possible position of the images and produce a *feature maps*. To compute the feature values, CNNs employs a special operation called *convolution*. For each position, the filter is convolved with the corresponding region of the image. Convolution operation is the main difference between CNNs and regular neural networks that operate on matrix multiplication.

Using a small set of filters is not enough to reduce the complexity of the model. A special technique, called *pooling*, is required to downsample the feature maps to make the network even sparser. Pooling is simply a mean of aggregating features in a local neighborhood.

Both derivation of the convolution and pooling operations can be derived easily. Therefore, CNNs can be train efficiently with backpropgation.

2 Convolution

Convolution is originally a mathemetical term that denotes this operation:

$$f * g = \int_{-\infty}^{\infty} f(t)g(x-t) \tag{1}$$

where t is size of the convolution window.

Note that convolution is commutative, i.e. $f * g = g * f$. For 2D functions, convolution is defined as follows:

$$f * g = \int_{-\infty}^{\infty} f(x,y)g(m-x,n-y) \tag{2}$$

where m, n are dimensions of the convolution window.

In the context of CNNs, we can imagine f as the filter, g as the image and (m, n) as a position on the image's 2D grid. The filter serves as a sliding window that goes across the image (in both dimensions) and convolves with corresponding subimages. Let $k \times k$ be the size of the filter, $w \times h$ be the size the of the image. The convolution of the filter with the (larger) image produces a feature map of size $(w - k + 1) \times (h - k + 1)$.

3 Pooling

As discussed above, pooling is a mean of combining statistics in a local region of a feature map. There are two common pooling mechanisms: take the average over a region and take the maximum value. We will analyze mean pooling to illustrate how this operation can be representation as a series of one convolution and two matrix multiplication operations. First, we create the mean feature map by convolving the original feature map with a filter whose elements are 1/K (K is the number of entries in the convolved region). Then we use matrix multiplications to take a subset of rows and columns from the mean feature map [1].

TODO: Why pooling?

4 Backpropagration review

Consider a generic feed-forward neural network consisting of L layers. Let $h^{(0)}$ be the input vector for the network. The forward pass is conducted as follows:

For $l = 1 \ldots L$:

- $a^{(l)} = W^{(l)} h^{(l-1)} + b^{(l)}$
- $h^{(l)} = f(a^{(l)})$

where $W^{(l)}$, $b^{(l)}$ are the parameters of the model and $f(.)$ is the activation function.

The cost function J is calculated from the final layer output $h^{(L)}$.

The backpropagation algorithm works as follows:

[1] Hint: you will need one left and one right matrix multiplication.

Set $\delta^{(L)} = \nabla_{h^{(L)}} J \bullet f'(a^{(L)})$

For $l = L \ldots 1$:

- $\nabla_{W^{(l)}} = \delta^{(l)} h^{(l-1)\top}$
- $\nabla_{b^{(l)}} = \delta^{(l)}$
- $\delta^{(l-1)} = W^{(l)\top} \delta^{(l)} \bullet f'(a^{(l)})$

where $\bullet$ is the element-wise multiplication operation.