Backpropagation in a Two-Layer Convolutional Neural Network

Abstract

This document provides a detailed derivation of backpropagation through a two-layer convolutional neural network (CNN) consisting of a sequence of operations: Convolution, ReLU, Max-Pooling, Fully Connected layer, and Softmax output. We compute gradients layer-by-layer with respect to the cross-entropy loss and explain the mathematical significance of each term.

1 Network Architecture Overview

We consider the following forward pipeline for a single training sample:

$$X \in \mathbb{R}^{C\times H\times W} \xrightarrow{\text{Conv}(W,b)} Z \xrightarrow{\text{ReLU}} A \xrightarrow{\text{MaxPool}} P \xrightarrow{\text{Flatten}} p \xrightarrow{\text{FC}(U,c)} s \xrightarrow{\text{Softmax}} \hat{y}$$

$$L = -\sum_{i=1}^{M} y_i \log \hat{y}_i$$

2 Forward Pass Details

2.1 Input and Convolution

- $X_{c,u,v}$: Input feature map.
- $W_{f,c,i,j}$, b_f: Convolutional filter weights and bias.

$$Z_{f,u,v} = \sum_{c=1}^{C}\sum_{i=1}^{K}\sum_{j=1}^{K} W_{f,c,i,j} X_{c,u+i-1,v+j-1} + b_f$$

2.2 ReLU Activation

$$A_{f,u,v} = \max(0, Z_{f,u,v})$$

2.3 Max-Pooling

Using non-overlapping 2×2 blocks:

$$P_{f,u',v'} = \max_{(i,j)\in\{0,1\}^2} A_{f,2u'+i,2v'+j}$$

2.4 Fully Connected Layer and Softmax

Flatten P into a vector p. Then compute:

$$s_i = \sum_k U_{i,k} p_k + c_i, \quad \hat{y}_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

$$L = -\sum_{i=1}^{M} y_i \log \hat{y}_i$$

3 Backpropagation Derivation

3.1 Gradient Through Softmax and Cross-Entropy

$$\delta_i^s = \frac{\partial L}{\partial s_i} = \hat{y}_i - y_i$$

3.2 Gradients in the Fully Connected Layer

$$\frac{\partial L}{\partial U_{i,k}} = \delta_i^s p_k, \quad \frac{\partial L}{\partial c_i} = \delta_i^s$$

$$\frac{\partial L}{\partial p_k} = \sum_{i=1}^{M} \delta_i^s U_{i,k} \Rightarrow \delta^P_{f,u',v'} = \frac{\partial L}{\partial P_{f,u',v'}}$$

3.3 Gradient Through Max-Pooling

Let (i^*, j^*) be the max index in block (f, u', v'), then:

$$\delta^A_{f,2u'+i,2v'+j} = \begin{cases} \delta^P_{f,u',v'}, & \text{if } (i,j) = (i^*, j^*) \\ 0, & \text{otherwise} \end{cases}$$

3.4 Gradient Through ReLU

$$\delta^Z_{f,u,v} = \delta^A_{f,u,v} \cdot \mathbf{1}_{\{Z_{f,u,v}>0\}}$$

3.5 Gradient Through Convolution

$$\frac{\partial L}{\partial b_f} = \sum_{u,v} \delta^Z_{f,u,v} \tag{1}$$

$$\frac{\partial L}{\partial W_{f,c,i,j}} = \sum_{u,v} \delta^Z_{f,u,v} \cdot X_{c,u+i-1,v+j-1} \tag{2}$$

$$\frac{\partial L}{\partial X_{c,x,y}} = \sum_{f=1}^{F} \sum_{i,j} \delta^Z_{f,x-i+1,y-j+1} \cdot W_{f,c,i,j} \tag{3}$$

4 Explanation of Key Terms

- $X_{c,u,v}$: Input at channel c, position (u, v).
- $W_{f,c,i,j}$: Filter weight for feature f, input channel c, offset (i, j).
- b_f: Bias term for filter f.
- $Z_{f,u,v}$: Pre-activation output of convolution.
- $A_{f,u,v}$: ReLU activation; retains positive components.
- $P_{f,u',v'}$: Max-pooled output from $A_{f,u,v}$.
- p_k: Flattened vectorized version of pooled features.
- $U_{i,k}, c_i$: Weights and bias in the fully connected layer.
- s_i: Logit score for class i.
- $\hat{y}_i$: Softmax probability for class i.
- δ^s_i: Gradient of loss with respect to logit s_i.
- **Weight Sharing**: Each convolutional filter is applied across spatial locations; hence, its gradients are aggregated over all positions.

5 Final Note

- The backpropagation steps follow from systematic application of the chain rule through all layers.
- Weight sharing in convolution layers means gradients accumulate over all spatial locations, a critical feature enabling translation invariance.
- This layer-by-layer breakdown enables implementation of a minimal backpropagation routine using libraries like NumPy.