

卷積神經網路 (六) Linear 線性層



山與水你和我

體系結構小白，影像處理小白

30 人同意了該文章

上一篇

[山與水你和我：卷積神經網路 \(五\) 卷積層](#)

完成了最複雜的Conv 卷積層的前向與反向傳播。

我通常將卷積神經網路看成兩個部分：

- 特徵提取層，有一系列的Conv、ReLU、Pool 等網路層串聯或並聯，最終得到特徵圖
- 任務相關層，例如用全連接層對得到的特徵圖做迴歸任務，擬合分佈等

在影像分類中，經常使用全連接層輸出每個類別的機率，但全連接層也有說法是線性變換層+ 激活函數+ 線性變換層+，多層感知機，但本次實驗為了簡單，只有一層Linear 線性層！

通常得到特徵提取部分的最後一個輸出 $B \times C_n \times H_n \times W_n$ ，將每個 $C_n \times H_n \times W_n$ 展平，作為Linear 的輸入，Linear 層有兩個參數，

- W ，線性變換矩陣 $IN \times K$ ，其中 $IN = C_n \times H_n \times W_n$ ， K 是類別數目，
- $bias$ ， K 維向量，每個類別輸出各自有一個可學習的偏移。

如下面的簡單圖示



贊同30



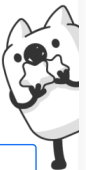
分享



登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。

立即登入/註冊



號▲ 贊同30

號▼

號● 新增評論

號🔗 分享

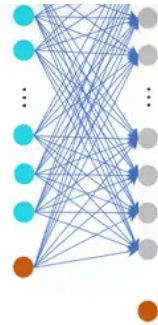
號♥ 喜歡

號★ 收藏

號📄 申請轉載

號...





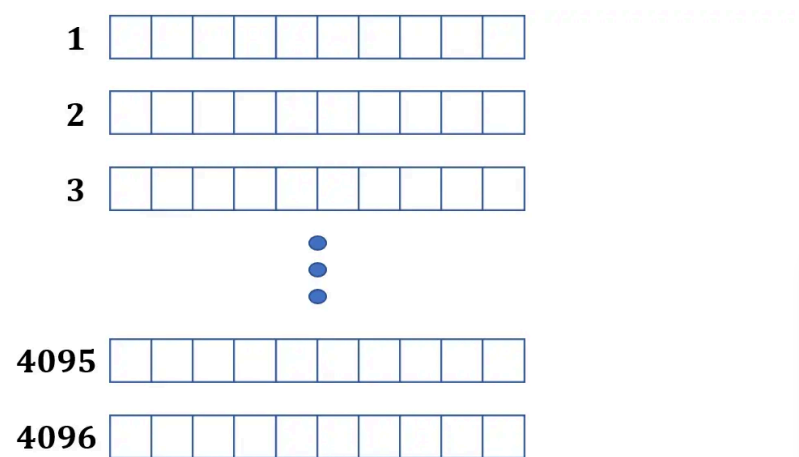
輸入: $B \times IN$

輸出: $B \times K$, K 是類別個數。

向前

前向過程十分簡單, B 個矩陣乘法。

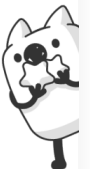
唯一要注意的是, Linear 層的 W 矩陣儲存方式, 假如矩陣大小是 4096×10 , 先儲存第一個10, 緊接著儲存第二個10.....儲存第4096個10

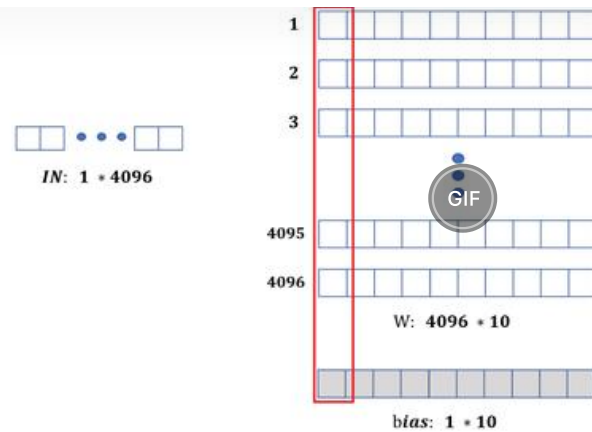


每次矩陣乘法, 都是 1×4096 與 4096×10 , 需要找到 W 矩陣所有行的某一行

登入即可查看超5億專業優質內容

超5千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



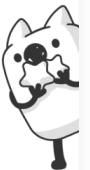


具體代碼如下：

```
std::vector<tensor> LinearLayer::forward(const std::vector<tensor>& input) {
    // 获取输入信息
    const int batch_size = input.size();
    this->delta_shape = input[0]->get_shape();
    // 清空之前的结果, 重新开始
    std::vector<tensor>().swap(this->output);
    // 分配空间
    for(int b = 0; b < batch_size; ++b)
        this->output.emplace_back(new Tensor3D(out_channels));
    // 如果有 backward 记录输入
    if(not no_grad) this->__input = input;
    // batch 每个图像分开算
    for(int b = 0; b < batch_size; ++b) {
        // 矩阵相乘, dot
        data_type* src_ptr = input[b]->data; // 1 * 4096
        data_type* res_ptr = this->output[b]->data; // 4096 * 10
        for(int i = 0; i < out_channels; ++i) {
            data_type sum_value = 0;
            for(int j = 0; j < in_channels; ++j)
                sum_value += src_ptr[j] * this->weights[j * out_channels + i];
            res_ptr[i] = sum_value + bias[i];
        }
    }
}
```

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



與卷積層初始化類似，本次實驗所採用的最簡單的常態分佈初始化

```
LinearLayer::LinearLayer(std::string _name, const int _in_channels, const int _out_channels)
    : Layer(_name),
      in_channels(_in_channels),
      out_channels(_out_channels),
      weights(_in_channels * _out_channels, 0),
      bias(_out_channels) {
    // 随机种子初始化
    std::default_random_engine e(1998);
    std::normal_distribution<float> engine(0.0, 1.0);
    for(int i = 0; i < _out_channels; ++i) bias[i] = engine(e) / random_times;
    const int length = _in_channels * _out_channels;
    for(int i = 0; i < length; ++i) weights[i] = engine(e) / random_times;
}
```

落後

Linear 層的backward，和先前的Conv卷積層類似，給定從下一層傳回來的梯度 δ^l ，要求

- 對本線性層參數 $W, bias$ 的梯度 $\delta W, \delta b$
- 傳給上一層的梯度 δ^{l-1}

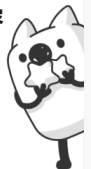
先求 $\delta W, \delta b$,

W, bias 的梯度

老規矩，求 W 中每一個參數，都要找這個參數參與了哪些計算，每個計算分別對哪些輸出有貢獻，下圖是一個範例

登入即可查看超5億專業優質內容

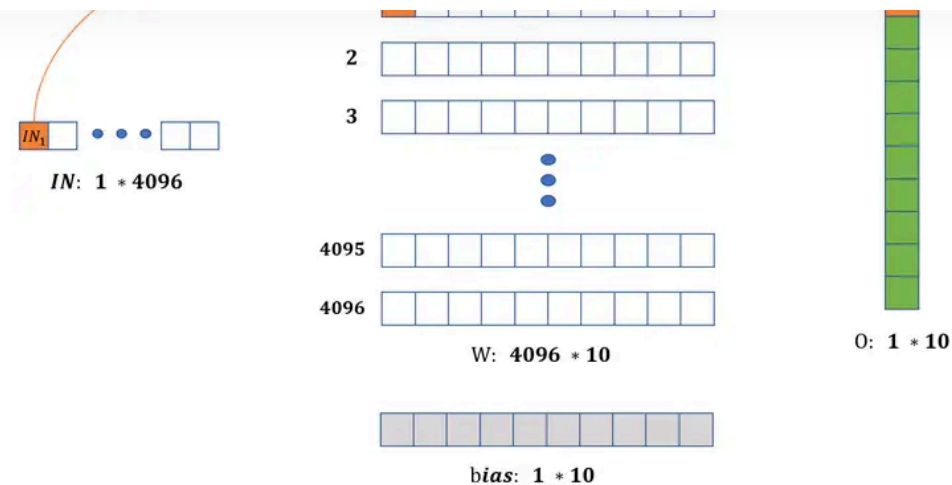
超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



知乎

首發於
電腦視覺基礎

切換模式



可見， W 的這個參數 w_1 對只參與了一次計算 $w_1 * IN_1$ ，只對第一個輸出有貢獻，因此計算 w_1 的梯度如下：

$$\delta W_{11}^{l-1} = \delta_1^l$$

其餘 w 同理；求偏置 $bias$ 的梯度，對每個輸出來說，每個位置的 $bias$ 都恰好只參與了一次計算，因此， $bias$ 只要加上對應位置返回來的梯度即可，

$$\delta b_i^{l-1} = \delta_i^l$$

實作很簡單，如下

```
// 第一次回傳，給緩衝區的梯度 w, b 分配空間
if(this->weights_gradients.empty()) {
    this->weights_gradients.assign(in_channels * out_channels, 0);
    this->bias_gradients.assign(out_channels, 0);
}
// 計算 W 的梯度
for(int i = 0; i < in_channels; ++i) {
    data_type* w_ptr = this->weights_gradients.data() + i * out_channels;
    for(int j = 0; j < out_channels; ++j) {
        data_type sum_value = 0;
        for(int b = 0; b < batch_size; ++b) {
            // 計算 w_ptr[j] 的梯度
            sum_value += IN[b][i] * delta[b][j];
        }
        w_ptr[j] = sum_value;
    }
}
```

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。

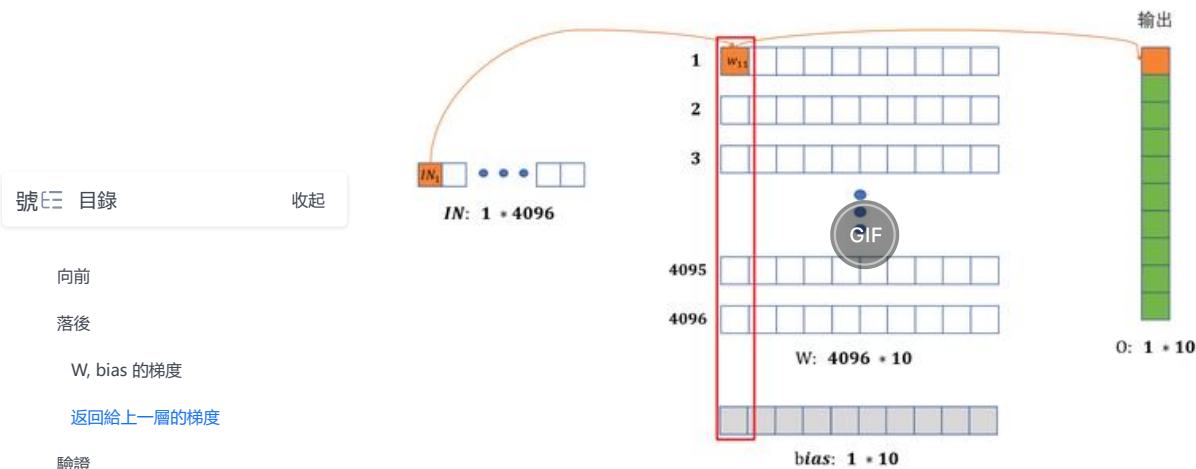


```
// 計算 bias 的梯度
for(int i = 0; i < out_channels; ++i) {
    data_type sum_value = 0;
    for(int b = 0; b < batch_size; ++b)
        sum_value += delta[b]->data[i];
    this->bias_gradients[i] = sum_value / batch_size;
}
```

和之前Conv 卷積層一樣，一個batch 的梯度累加取平均值。

返回給上一層的梯度

同理，求返回給上一層梯度 δ^{l-1} ，也是要找出輸入的每個值，都和哪些Linear層的參數做了運算，得到了哪些輸出，如下圖示輸入 IN_1 ：



號 目錄

收起

向前

落後

W, bias 的梯度

返回給上一層的梯度

驗證

程式碼

🔍 🔍 🔍 🔍 🔍 🔍 🔍 🔍 🔍 🔍

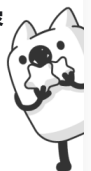
因此，可以寫出 IN_1 這個位置上的梯度

$$\delta_1^{l-1} = \sum_{i=1}^{10} w_{1i} * \delta_i^l$$

同理，規律為 $\delta_k^{l-1} = \sum_{i=1}^{10} w_{ki} * \delta_i^l$ 。

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



```
for(int i = 0; i < in_channels; ++i) { // 每个输入神经元
    data_type sum_value = 0;
    data_type* w_ptr = this->weights.data() + i * out_channels;
    for(int j = 0; j < out_channels; ++j) // 每个输出都由第 i 个神经元与对应的 w_ij 相乘
        sum_value += src_ptr[j] * w_ptr[j];
    res_ptr[i] = sum_value;
}
// 返回到上一层给的梯度
return this->delta_output;
```


至此，返回給上一層的梯度 δ^{l-1} ，在每一個位置都可以求解。

驗證

至此，卷積神經網路最基本的層都寫好了，下一步開始建立一個最簡單的捲積神經網絡

卷積神經網路 (七) 建構CNN 網路結構
5 贊同 4 評論 文章

程式碼

<https://github.com/hermosayhl/CNN>
號  github.com/hermosayhl/CNN

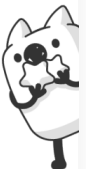
編輯於2022-02-20 11:26

C++ 卷積神經網路 (CNN) 反向傳播

寫下你的評論...

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。



知乎

首發於
電腦視覺基礎

切換模式



還沒有評論，發表第一個評論吧

文章被以下專欄收錄



電腦視覺基礎

最簡單的捲積神經網路、ReLU、BN、目標偵測等

推薦閱讀

卷積神經網路 (基礎篇)

1. 卷積層: import torch
in_channels, out_channels = 5,
10 # 輸入通道大小和輸出通道大小
width, height = 100, 100 # 圖片
大小kernel_size = 3 # 卷積核大小
batch_size = 1 input = t...

菱角

時間分塊 - 倍增有限差分運算速度的關鍵

摘要在科學與工程計算中，為克服有限差分等模板計算中的記憶體頻寬瓶頸問題，研究者先後提出了平行四邊形分塊、梯形分塊、三角形分塊、菱形分塊（鑽石分塊）以及六邊形分塊。本文介紹了這些分...

尼科科尼

[基礎知識補全計畫] 詳解可變卷積: Deformable...

Deformable Convolutional NetworksICCV 2017 截至3.29 引用數1611 tips: 1. 這篇文章主要講了可變卷積和可變roi池化的兩個模組的構建，實驗和消融部分有興趣可以跳轉至原文Deformable...

風一樣的男... 發表於機器學習論...

$$\text{Time} \sim O\left(\sum_{l=1}^D M_l^2 \cdot K_l^2 \cdot C_{l-1} \cdot C_l\right)$$

$$\text{Space} \sim O\left(\sum_{l=1}^D K_l^2 \cdot C_{l-1} \cdot C_l + \sum_{l=1}^D M_l^2 \cdot C_l\right)$$

卷積神經網路的複雜度分析

麥可·袁

登入即可查看超5億專業優質內容

超5 千萬創作者的優質提問、專業回答、深度文章和精彩影片盡在知乎。

