> 2022-04-16 — 2022-04-21
>
> 计算机视觉导论 (北京大学 2022 春)
>
> 第二次课程作业笔记 by PkuCuipy

> ⚠️ 请独立完成课程作业!
>
> 报告及代码仅供参考, 欢迎 issue 讨论代码中的问题和改进办法, 提前表示感谢!
>
> 另附我的完成时间如下, 供任务量参考:
>
> 第一题: 5h (误入歧途浪费了很多时间, 如果直接找博客直奔主题, 2h 应该能搞定)
>
> 第二题: 3h (外加很久 (3h+) 的训练和调参的尝试, 虽然后来发现是自己想复杂了..)
>
> 第三题: 1.5h (把 Slides 上的结论实现成代码就行)

---

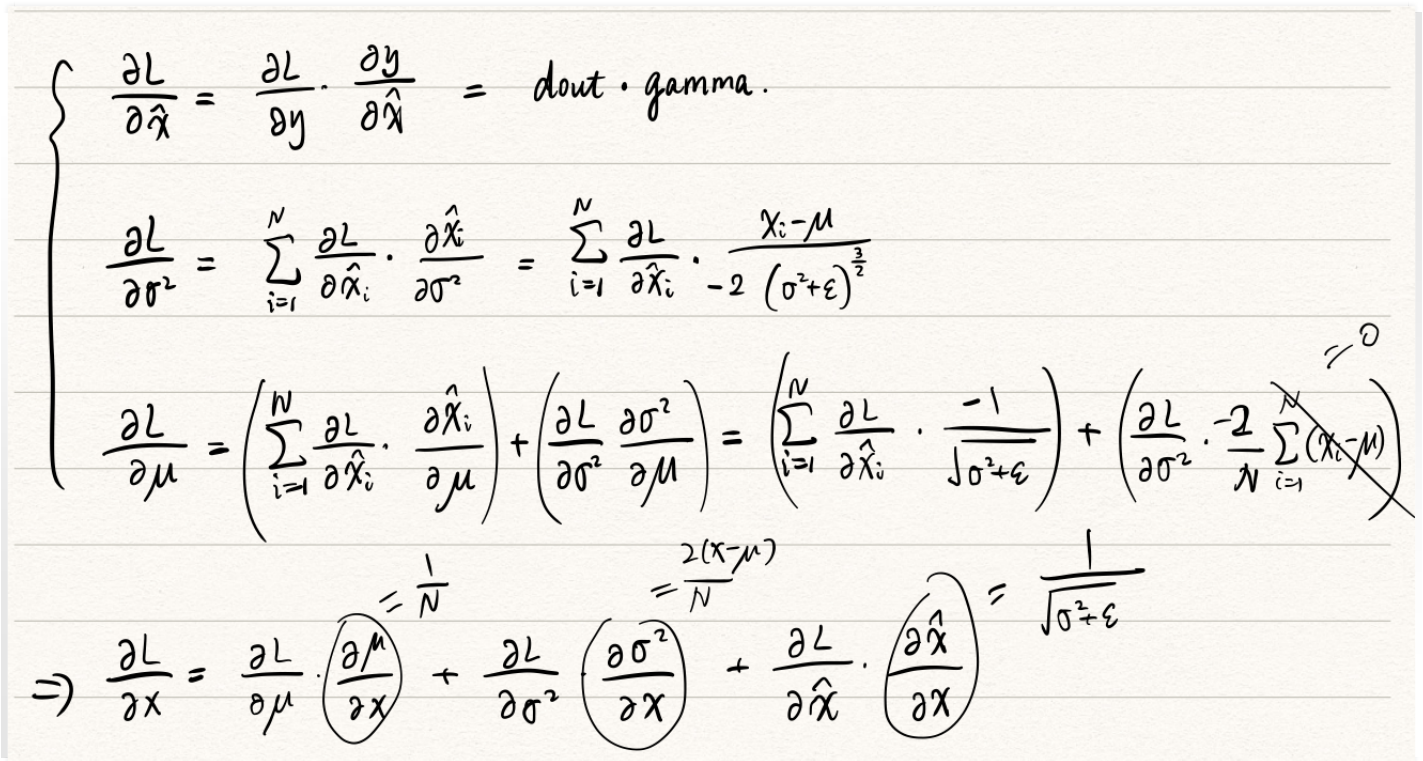
1. Batch Normalization (30')

1(a) Implement the Forward Pass for Training

- 注意计算 μ 和 σ 时分母是 N, 而不是 N-1. 调包的时候可能要注意.
- β 和 γ 是 C 维向量, 而不是一个实数, 之前一直搞错了hhh

1(b) Implement the Backward Pass for Training

- 用计算图的方法来计算可以参考这篇知乎文章. 其思路就是严格地按照计算图一步一步来.
- 一个 Tip 是推导这个的时候可以不考虑样本的 channel 数 C, 而是只需要先考虑假设 x 是 Batch Size 个 Scalar 来推即可.
- 对于这里, ∂L/∂γ 和 ∂L/∂β 都是比较简单的. 而 ∂L/∂x 则要用计算图的技巧——如果不这样, 硬推的话要注意 x_i 影响 L 可不仅是**通过** y_i, 而是通过全部的 $y_1, \ldots, y_n$! 这里一开始我搞错了, 浪费了不少时间还没检查出错误, 多亏课友帮忙.
- 注: 计算图的核心思路就是: 假设节点 A 的**下游**节点是 B 和 C, 那么 ∂L/∂A 就拆解为 ∂L/∂B · ∂B/∂A + ∂L/∂C · ∂C/∂A .



1(c) Implement the Forward Pass for Inference

- 和 1(a) 的区别在于这里 μ 和 σ 使用 Streaming Algorithm 给出, 而非通过 batch 算出来 (test 的时候 B 恒为 1 也没法算 σ).

---

2. Train a CNN on CIFAR-10 (40')

- CIFAR-10 数据集: 60000 32x32 colour images in 10 classes, 6000 images per class. 50000 train and 10000 test.

2(a) Implement a CNN

- CNN 网络结构的建议见 PDF 最后一页, 保险起见我正是按照这个实现的.

  其中每个结构的参数 (channel 数、padding、kernel_size 等) 可以通过图中箭头上的数字推断得到.
- **不要**擅自主张在最后加 `nn.Softmax()`! 因为给的代码中用的 Loss 是 CrossEntropyLoss, 而根据 PyTorch 的文档, 这个东西接收的 input 应该是 "raw" 的, 即不是 Softmax 过的! 本质的原因也很简单, 因为 `nn.CrossEntropyLoss()` 是 `nn.LogSoftmax()` 和 `nn.NLLLoss()` 组合起来的简写而已hhh.

```python
self.model = nn.Sequential(
    # PART 1
    nn.Conv2d(in_channels=3,  out_channels=32, kernel_size=3), nn.ReLU(),
    nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3), nn.ReLU(),
    nn.MaxPool2d(kernel_size=2),
    # PART 2
    nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3), nn.ReLU(),
    nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3), nn.ReLU(),
    nn.MaxPool2d(kernel_size=2),
    # PART 3
    nn.Flatten(start_dim=1, end_dim=-1),
    nn.Linear(in_features=1600, out_features=512), nn.ReLU(),
    nn.Linear(in_features=512,  out_features=10),
    # nn.Softmax()  # 不要擅作主张!
)
```

2(b) Adjust Learning Rate and Visualize Curves

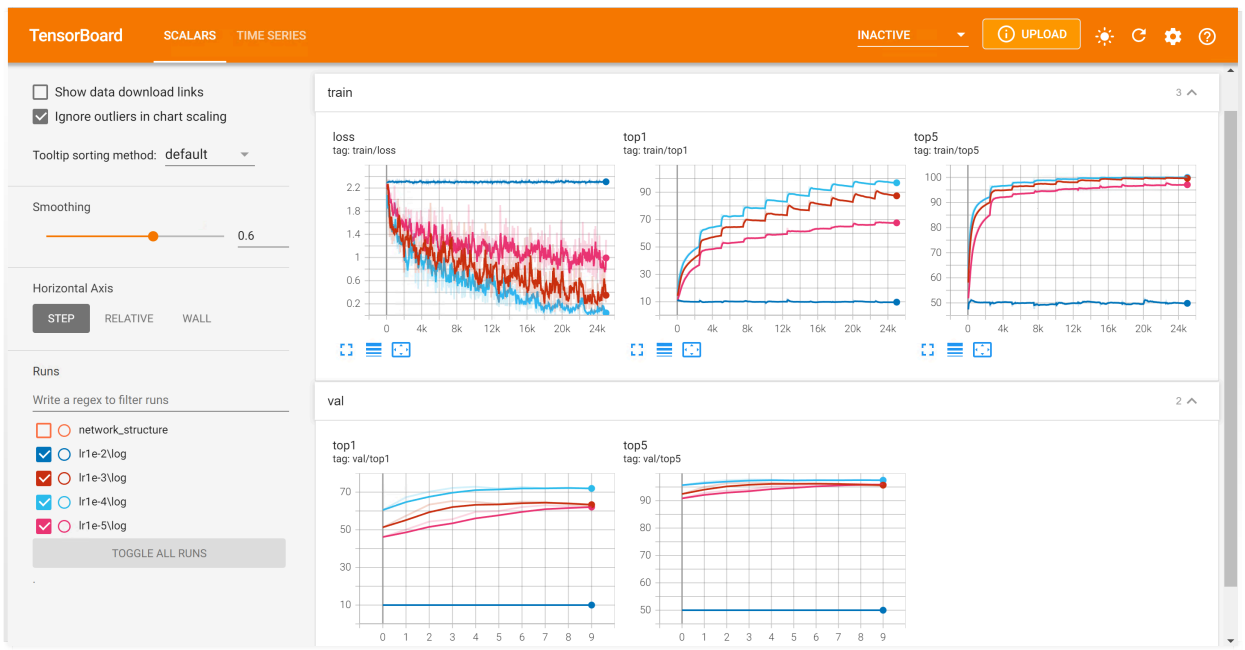
- 利用 Tensorboard 绘制用到的关键代码如下. `add_scalar` 的参数依次为: 图表名称、y 轴值、x 轴值.

```python
writer.add_scalar('train/loss', loss, iteration)
```

- 实现完绘图指令后, 在 Shell 运行如下命令得到不同学习率下的图表📈 (`&` 是依次运行的小技巧):

```shell
python train.py -e lr1e-2 -l 1e-2 &
python train.py -e lr1e-3 -l 1e-3 &
python train.py -e lr1e-4 -l 1e-4 &
python train.py -e lr1e-5 -l 1e-5 &
echo "All Done!"
```

- 关于这里统计的 top-1 和 top-5:
  - top-1: 模型预测概率最高**那个**类**是否是**真实类?
  - top-5: 模型预测概率最高的**那 5 个**类中**是否包含**真实类?
- 运行结果:



- 为什么 Accuracy 曲线 (第一层中间) 呈阶梯状???

2(c) Data Augmentation and Visualization

- 根据代码框架, 需要补全的是 `CIFAR10` 类的 `__getitem__` 方法来实现 Data Augmentation.

  这里我疑惑了一会——因为感觉这样会让原数据被替换, 而非增加新数据.

  后来看到这个帖子才恍然大悟——因为具有随机性, 所以对 dataset 的同一个 index 的多次索取取出的图片是具有随机变化的!

2(d) Further Improve your Network

- 根据 2(b) 的运行结果, 可知 1e-4 的学习率可以达到最好的效果, 此时 Validation Accuracy 是 72%,

  而这道题只要 ⩾75% 就能拿到满分, 所以可以直接基于这个进行调整——事实上只要加上 Horizontal Flip 的 Data Augment, 就可以达到 76% 的 Accuracy, 从而拿到满分. 以下是踩坑:
- 最开始发现只要一加 ColorJitter 就学不出来, 非常诡异; 后来过了几天查出来问题了: `torch.transforms` 接收的图片是 float32 的 [0, 1] 的 `torch.tensor`! 然而代码框架里原本的 `img` 是 [0, 255] 的 float32 `numpy.ndarray`, 这实在令人迷惑! 我一开始看到 float32 想当然地以为是 [0, 1] 的了.. 所以解决办法就是: 喂给 transform 前 `/255`, 变换完再乘回去 (只为了训练其实也不用乘回去, 乘回去只是因为之前代码框架中的 output 是 0~255 的 float.. 这样可以兼容作业框架中「可视化 Data Augment 后的 CIFAR data」的那部分代码.

---

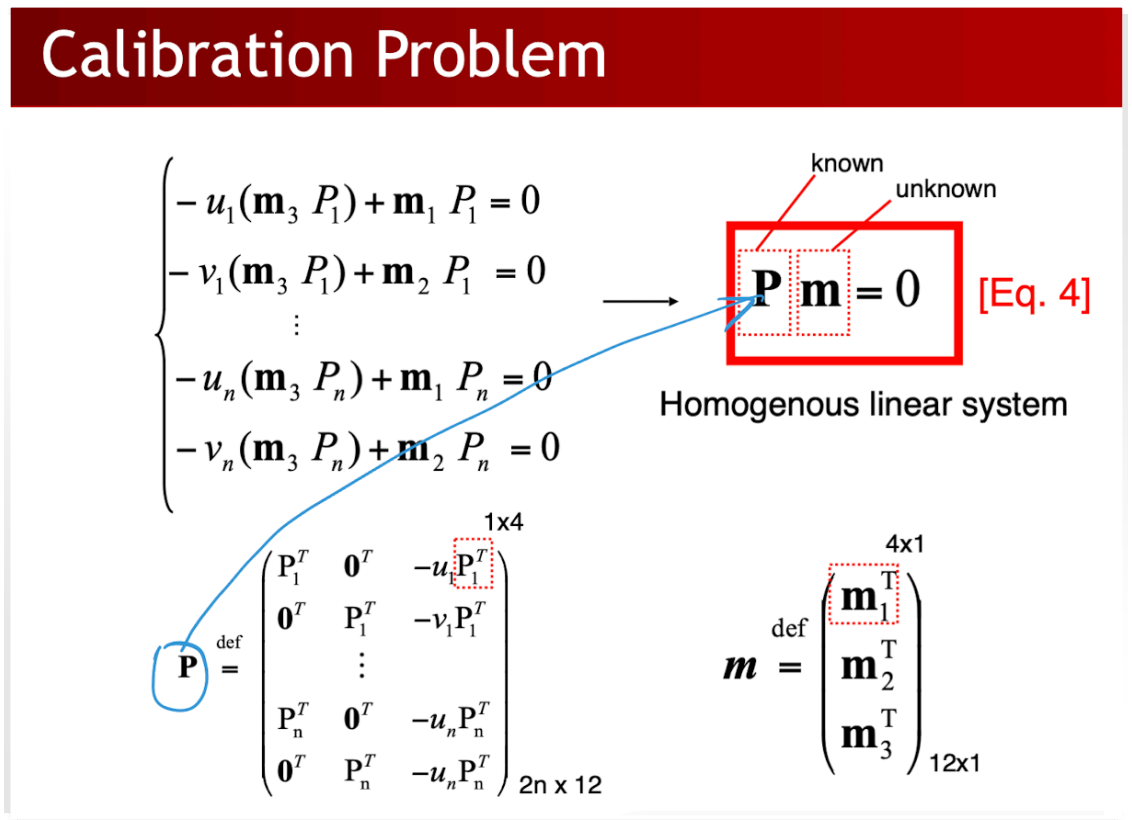
3. Camera Calibration (30')

3(a) Compute the Corresponding 3D coordinates

- 不知道什么叫 Compute, 我反正是人工一个一个写的..

3(b) Construct the Equation Pm = 0 and Solve m

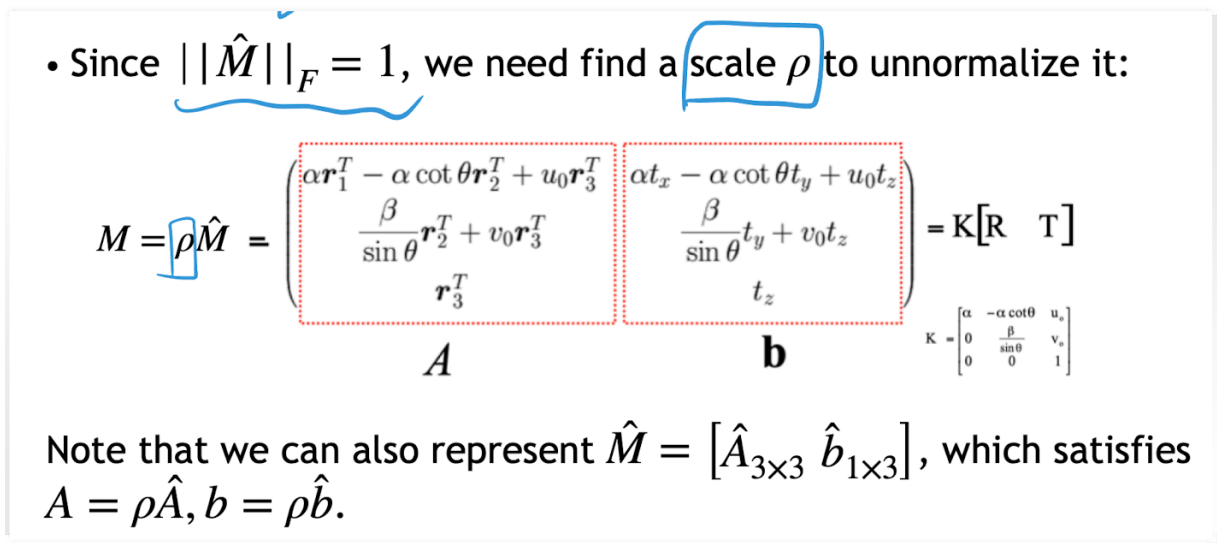
- 使用 Slides 上的公式来构造系数矩阵 P:



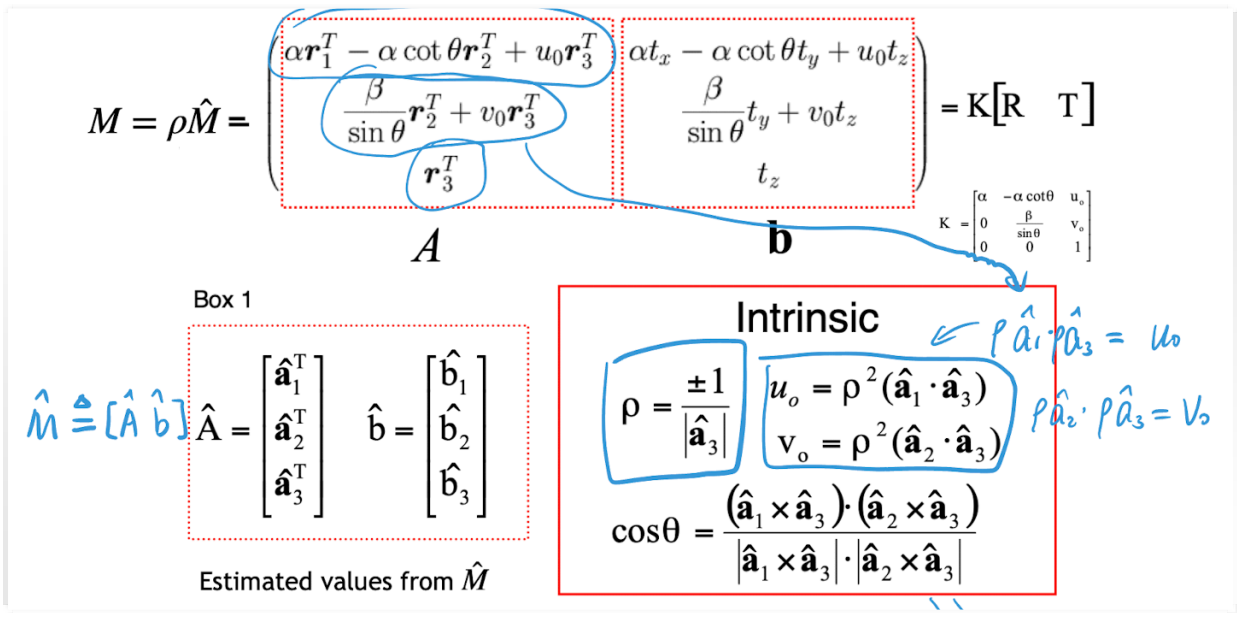
- 使用 SVD 的技巧求解方程 Pm=0 中的 m:
  - 为了避免问题 $Pm = 0$ 的平凡解 $m = 0$.
  - 问题转换为在 $|m| = 1$ 的约束下最小化 $|Pm|$.
  - 而对 P 进行 SVD 分解, 即 $P = USV^T$, 则其中的 v_n 刚好符合 m 的定义. 即 $m = v_n$, 其中 v_n 是 V 的最后一列.

3(c) Solve K and [R T] from m

- 套 Slides 上的公式即可:



记刚才得到的 m 为 M_hat



ρ, u0, v0 和 θ 的公式. (这里±号我取的+)

$$M = \rho\hat{M} = \begin{pmatrix} \alpha\boldsymbol{r}_1^T - \alpha\cot\theta\boldsymbol{r}_2^T + u_0\boldsymbol{r}_3^T & \alpha t_x - \alpha\cot\theta t_y + u_0 t_z \\ \frac{\beta}{\sin\theta}\boldsymbol{r}_2^T + v_0\boldsymbol{r}_3^T & \frac{\beta}{\sin\theta}t_y + v_0 t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix} = \mathrm{K}[\mathrm{R} \quad \mathrm{T}]$$

A b

Box 1

$$\hat{\mathrm{A}} = \begin{bmatrix} \hat{\mathbf{a}}_1^{\mathrm{T}} \\ \hat{\mathbf{a}}_2^{\mathrm{T}} \\ \hat{\mathbf{a}}_3^{\mathrm{T}} \end{bmatrix} \qquad \hat{\mathrm{b}} = \begin{bmatrix} \hat{\mathrm{b}}_1 \\ \hat{\mathrm{b}}_2 \\ \hat{\mathrm{b}}_3 \end{bmatrix}$$

Estimated values from $\hat{M}$

Intrinsic

$$\alpha = \rho^2 |\hat{\mathbf{a}}_1 \times \hat{\mathbf{a}}_3| \sin\theta$$

$$\beta = \rho^2 |\hat{\mathbf{a}}_2 \times \hat{\mathbf{a}}_3| \sin\theta$$

α, β 的公式

$$M = \rho\hat{M} = \begin{pmatrix} \alpha\boldsymbol{r}_1^T - \alpha\cot\theta\boldsymbol{r}_2^T + u_0\boldsymbol{r}_3^T & \alpha t_x - \alpha\cot\theta t_y + u_0 t_z \\ \frac{\beta}{\sin\theta}\boldsymbol{r}_2^T + v_0\boldsymbol{r}_3^T & \frac{\beta}{\sin\theta}t_y + v_0 t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix} = \mathrm{K}[\mathrm{R} \quad \mathrm{T}]$$

A b

Box 1

$$\hat{\mathrm{A}} = \begin{bmatrix} \hat{\mathbf{a}}_1^{\mathrm{T}} \\ \hat{\mathbf{a}}_2^{\mathrm{T}} \\ \hat{\mathbf{a}}_3^{\mathrm{T}} \end{bmatrix} \qquad \hat{\mathrm{b}} = \begin{bmatrix} \hat{\mathrm{b}}_1 \\ \hat{\mathrm{b}}_2 \\ \hat{\mathrm{b}}_3 \end{bmatrix}$$

Estimated values from $\hat{M}$

Extrinsic

$$\mathbf{r}_1 = \frac{(\hat{\mathbf{a}}_2 \times \hat{\mathbf{a}}_3)}{|\hat{\mathbf{a}}_2 \times \hat{\mathbf{a}}_3|} \qquad \mathbf{r}_3 = \frac{\pm\hat{\mathbf{a}}_3}{|\hat{\mathbf{a}}_3|}$$

$$\mathbf{r}_2 = \mathbf{r}_3 \times \mathbf{r}_1 \qquad \mathrm{T} = \rho\ \mathrm{K}^{-1}\hat{\mathbf{b}}$$

R = [r1, r2, r3] 和 T 的公式. (这里±号我取的+)