

8.23-9.5 周报

赵晓辉

2021 年 9 月 3 日

1 写在前面

这个双周主要完成了 cs231n 课程的 Lecture 1 至 Lecture 6, 主要内容包括距离函数、KNN、SVM、损失函数及优化、BP 算法、CNN 架构、非线性激活函数以及神经网络的参数优化等, 并完成 cs231n assignment1。课程概要笔记及 assignment 将在<https://github.com/zxh991103/cs231NOTE>持续跟踪。

2 Lec 1-6 课程概要

2.1 距离函数

L_1 Distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L_2 Distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

2.2 KNN

计算测试样本与所有训练集样本之间的距离值, 并根据 K 值投票选举出最相似的标签。

2.3 SVM

计算能够划分训练集样本且距离最大的超平面。

$$w \cdot x + b = 0$$

2.4 损失函数

损失函数评估模型预测值与模型真实值之间的差异性，我们要将其最小化。对于给定的训练集 $(x_i, y_i)_{i=1}^N$ ，我们有损失函数：

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

对于 Multi-SVM，我们有损失函数，即 hinge loss：

$$s = f(x_i, W)$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

对于 softmax loss：

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

2.5 正则化

根据奥卡姆剃刀原则，模型越简单越符合实际，所以我们将正则惩罚项加在损失函数上。

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i) + R(W)$$

L1

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

L2

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

Elastic

$$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$$

2.6 BP 算法

链式法则：

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

故在计算损失函数对于参数的梯度值时，我们应当将本地梯度值与上游回传梯度值相乘。

此时，我们也可以发现 Relu 函数，即 max gate 中只有前向传播计算中的正值能影响下游。

A vectorized example:

$$f(x, W) = \|Wx\|^2 = \sum_{i=1}^n (Wx)_i^2$$

$$q = W \cdot x$$

$$\nabla_W f = 2q \cdot x^T$$

2.7 NN

假设我们将神经网络的计算图表示为:

$$f = \text{Softmax}(W_2 \text{Relu}(W_1 x))$$

神经元 A 拥有 W_1 , 能够具有识别出来 100 种特征的功能, 比如识别出马的左脸或者右脸、车头或者车尾。而神经元 B 拥有 W_2 , 其功能就在于将马的左脸或右脸合并为马的特征, 将车头或车尾合并成车的特征, 从而进行识别。

2.8 CNN

卷积层:

假设有 $32 \times 32 \times 3$ 的图片, 卷积核 $w \ 5 \times 5 \times 3$, 以及偏置 b , 卷积后我们获得 $28 \times 28 \times 1$ 的矩阵, 其中 1 是卷积核的数量。卷积公式为 (每位相乘再求和):

$$f[x, y] * g[x, y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[x, y] * g[x - n_1, y - n_2]$$

步长 (stride):

假设我们有 7×7 的输入, 3×3 的卷积核, 2 的步长, 最后的输出为 3×3 。

此时 $\text{outputsize} = \frac{(N-F)}{\text{stride}} + 1$

填充 (Pad):

图像四周补充 0, 来防止在深层卷积时张量过小。此时, $\text{outputsize} = \frac{(N-F+2P)}{\text{stride}} + 1$ 。

Example:

input volume $32 \times 32 \times 3$, 10 5×5 filters (include 3 depth), stride 1, pad 2, we have 760 parameters ($10 * (5 * 5 * 3 + 1 \text{ bias}) = 760$)

pooling layer: 相当于下采样。

maxpooling:

一般, 每一个池化 filter 具有和步长相同的大小以避免 overlap.

例如,

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

我们使用 2*2 的 filter 和 2 的 stride,maxpooling 后变为:

6	8
3	4

2.9 激活函数

若全部线性连接则等同于一个线性连接, 所以网络中需要非线性的激活函数变换。

sigmod

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

tanh

$$\tanh(x)$$

Relu

$$\max(0, x)$$

LeakyRelu

$$\max(0.1x, x)$$

Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

Elu

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

SELU

$$f(x) = \begin{cases} \lambda x & \text{if } x > 0 \\ \lambda \alpha(e^x - 1) & \text{otherwise} \end{cases}$$

softmax 的问题:

- x 过大或过小，本地梯度接近 0，使得与上游梯度乘积也接近于 0，更新缓慢。
- x 的值恒正或恒负，本地梯度总是大于 0 的，造成 w 的移动时锯齿状的，接近最优值放缓。

relu 的问题:

- 若 $w \cdot x + b$ 总是负的，则本地梯度为 0，造成参数不更新。

2.10 数据处理

```
1 # ZERO-CENTER
2 X -= np.mean(X,axis = 0)
3 # normalize
4 X /= np.std(X,axis = 0)
```

Listing 1: normalization

2.11 参数初始化

Naive: 为参数初始化小随机数。但是随着网络深度的增加，本地梯度与上游梯度相乘之后接近零，学习十分缓慢。

Xavier:

```
1 W = np.random.randn(dim_in,dim_out)/np.sqrt(dim_in)
```

Listing 2: Xavier

原因:

we want $\text{Var}(y) = \text{Var}(x_i)$, and we have

$$y = \sum_{i=1}^{D_{in}} x_i w_i$$

and we assume that every x has same var. so we have

$$\text{var}(y) = D_{in} \times \text{var}(x) \times \text{var}(w_i)$$

and obviously initial $w_i \sim N(0,1)$, we make $\frac{w_i}{\sqrt{D_{in}}}$ to achieve the var is $\frac{1}{D_{in}}$

Kaiming/MSRA:

```
1 W = np.random.randn(dim_in,dim_out)*np.sqrt(2/dim_in)
```

Listing 3: MSRA

2.12 Batch Normalization

we have the input x like $N \times D$

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

so that we have:

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j}$$

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

the net is supposed to learn $\gamma \in R^D$ and $\beta \in R^D$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

BN 层经常用于全连接或卷积层后。

2.13 Norm For Conv.

batch norm , 对于每一个 batch 中的每个 channel 取平均得到 $(1 \times 1 \times C)$

layer norm , 对于所有的 batch 我们取一个平均的图片 $(H \times W \times C)$

instance norm , 对于所有的 batch 我们将取一个平均的单通道图片 $(H \times W \times 1)$

group norm , 对于所有 batch, 我们将 channel 分为 k 个组, 在每个组上取平均值得到 $(H \times W \times k)$

3 Assignment1

3.1 KNN

双循环实现距离矩阵:

```
1 num_test = X.shape[0]
2 num_train = self.X_train.shape[0]
3 dists = np.zeros((num_test, num_train))
4 for i in range(num_test):
5     for j in range(num_train):
6         t1 = X[i]
```

```

7         t2 = self.X_train[j]
8         t = t1 - t2
9         t = t*t
10        t = t.sum()
11        t = t**0.5
12        dists[i][j] = t

```

Listing 4: KNN 双循环实现距离矩阵

单循环实现距离矩阵：

```

1    num_test = X.shape[0]
2    num_train = self.X_train.shape[0]
3    dists = np.zeros((num_test, num_train))
4    for i in range(num_test):
5        dists[i,:] = np.sum((X[i,:]-self.X_train)**2,axis = 1)

```

Listing 5: KNN 单循环实现距离矩阵

向量操作实现距离矩阵（矩阵下的完全平方公式）：

```

1    M = np.dot(X, self.X_train.T)
2    nrow=M.shape[0]
3    ncol=M.shape[1]
4    te = np.diag(np.dot(X,X.T))
5    tr = np.diag(np.dot(self.X_train,self.X_train.T))
6    te= np.reshape(np.repeat(te,ncol),M.shape)
7    tr = np.reshape(np.repeat(tr, nrow), M.T.shape)
8    sq=-2 * M +te+tr.T
9    dists = np.sqrt(sq)

```

Listing 6: KNN 向量操作实现距离矩阵

KNN 预测：

```

1    num_test = dists.shape[0]
2    y_pred = np.zeros(num_test)
3    labelnum = len(np.unique(self.y_train))
4    for i in range(num_test):
5        # A list of length k storing the labels of the k nearest neighbors
6        # to
7        # the ith test point.
8        closest_y = []
9        closest_y = np.argsort(dists[i,:])[:k]
10       t = np.zeros(labelnum,dtype=int)
11       for j in closest_y:
12           t[self.y_train[j]]+=1
13       y_pred[i] = t.argmax()

```

Listing 7: KNN 预测

3.2 SVM

Listing 8: Python example

计算 SVM loss、dW(naive):

```
1  dW = np.zeros(W.shape) # initialize the gradient as zero
2
3  # compute the loss and the gradient
4  num_classes = W.shape[1]
5  num_train = X.shape[0]
6  loss = 0.0
7  for i in range(num_train):
8      scores = X[i].dot(W)
9      correct_class_score = scores[y[i]]
10     for j in range(num_classes):
11         if j == y[i]:
12             continue
13         margin = scores[j] - correct_class_score + 1 # note delta = 1
14         if margin > 0:
15             loss += margin
16 loss /= num_train
17
18 # Add regularization to the loss.
19 loss += reg * np.sum(W * W)
20
21 for i in range(num_train):
22     scores = X[i].dot(W)
23     correct_class_score = scores[y[i]]
24     for j in range(num_classes):
25         if j == y[i]:
26             continue
27         margin = scores[j] - correct_class_score + 1 # note delta = 1
28         if margin > 0:
29             dW[:,j] += X[i]
30             dW[:,y[i]] -= X[i]
31 dW /= num_train
32 dW += reg * W * 2
```

Listing 9: SVM(naive)