

人工智能基础实验 2

傅申 PB20000051

1. 实验环境

本次实验的代码在我的笔记本上运行, 配置如下:

CPU Intel i5-1035G1

GPU NVIDIA GeForce MX350

内存 16 GB 双通道 DDR4

相应的软件环境如下, 其中 Python 环境及其相关包由 micromamba 包管理器提供:

操作系统 GNU/Linux 6.1.31-2-MANJARO x86_64

Micromamba 1.4.7

Python 3.10.12

NumPy 1.24.2

Matplotlib 3.6.3

OpenCV (OpenCV-Python/cv2) 4.7.0

PyTorch 2.0.1, build py3.10_cuda11.8_cudnn8.7.0_0

tqdm 4.65.0

2. 传统机器学习

2.1. 贝叶斯网络手写数字识别

2.1.1. 实验原理与实现

记输入 (取值为 1/0 的像素矩阵/向量) 为 \mathbf{X} .

在本次实验中, 给定一个数据样本 \mathbf{x} , 模型要计算样本对应数字的概率分布 $P(Y|\mathbf{X} = \mathbf{x})$, 并取概率最大的数字作为输出 (预测的数字).

由 Bayes 定理, 上面的概率分布可以写为

$$P(Y|\mathbf{X} = \mathbf{x}) = \frac{P(Y)P(\mathbf{X} = \mathbf{x} | Y)}{P(\mathbf{X} = \mathbf{x})} \quad (1)$$

由于我们只关心概率分布的相对大小 (只选择最大值作为输出), 因此分母 $P(\mathbf{X} = \mathbf{x})$ 并不会影响模型, 因此有

$$\text{Predict}(\mathbf{x}) = \underset{y}{\operatorname{argmax}} P(Y = y)P(\mathbf{X} = \mathbf{x} | Y = y) \quad (2)$$

这意味着, 我们只需要知道数字的先验概率分布和像素的条件概率分布, 就可以构建出相应的模型 (Bayesian 网络) 来识别手写数字.

给定数据集 $D = \{\mathbf{x}_i, y_i\}_{i=1}^N$, 其中 \mathbf{x}_i 是一个屏幕像素信息 (长度为 M 的 1/0 向量), y_i 是对应的数字, 我们可以估计所需的概率分布, 具体来说, 有

$$P(Y = y) = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(y_i = y) \quad (3)$$

$$P(\mathbf{X} = \mathbf{x} \mid Y = y) = \prod_{j=1}^M P(\mathbf{X}_j = \mathbf{x}_j \mid Y = y)$$

其中

$$\mathbb{I}(x = y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

$$P(\mathbf{X}_j = \mathbf{x} \mid Y = y) = \frac{\sum_{i=1}^N \mathbb{I}(\mathbf{X}_j = \mathbf{x}, y_i = y)}{\sum_{i=1}^N \mathbb{I}(y_i = y)} \quad (4)$$

综上所述, 我们可以通过统计数据集中的信息, 估计出所需的概率分布, 并据此进行预测. 其中, 估计的过程可以通过以下代码实现:

```
def fit(self, pixels, labels):
    n_samples = len(labels)
    self.labels_prior[np.unique(labels)] = np.bincount(labels) / n_samples
    # conditional probability
    for i in range(self.n_labels):
        self.pixels_cond_label[:, 1, i] = \
            np.sum(pixels[labels == i], axis=0) / np.sum(labels == i)
        self.pixels_cond_label[:, 0, i] = \
            1 - self.pixels_cond_label[:, 1, i]
```

这里的 `labels_prior[y]` 对应上面的先验概率 $P(Y = y)$, `pixels_cond_label[j, x, y]` 对应上面的条件概率 $P(X_j = x \mid Y = y)$.

相应地, 预测的过程可以通过以下代码实现:

```
def predict(self, pixels):
    n_samples = len(pixels)
    labels = np.zeros(n_samples)
    pixels_cond_label_one = self.pixels_cond_label[:, 1, :]
    pixels_cond_label_zero = self.pixels_cond_label[:, 0, :]
    for i in range(n_samples):
        sample = np.transpose([pixels[i]])
        prob_one = np.multiply(sample, pixels_cond_label_one)
        prob_zero = np.multiply(1 - sample, pixels_cond_label_zero)
        prob = np.prod(prob_one + prob_zero, axis=0) * self.labels_prior
        labels[i] = np.argmax(prob)

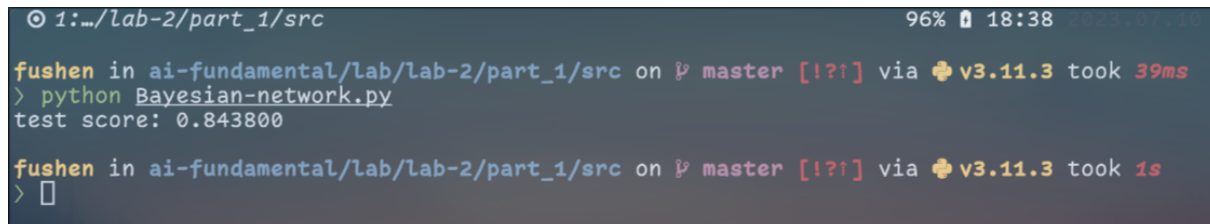
    return labels
```

考虑 `for` 循环中处理单个新样本 `sample` (记为 \mathbf{x}) 的过程, `pixels_cond_label_one[j][y]` 对应先验条件概率 $P(\mathbf{X}_j = 1 \mid Y = y)$, 而 `pixels_cond_label_zero[j][y]` 对应先验条件概率 $P(\mathbf{X}_j = 0 \mid Y = y)$. 分别处理 \mathbf{x} 中取值为 1 的部分 `prob_one` 和取值为 0 的部分 `prob_zero`, 将

其相加后得到 $(\text{prob_one} + \text{prob_zero})[j][y]$ 就对应先验条件概率 $P(\mathbf{X}_j = x_j | Y = y)$, 每列连乘就得到了 $P(\mathbf{X} = \mathbf{x} | Y)$. 最后按照等式 2 计算预测结果.

2.1.2. 实验结果

运行程序, 得到准确率为 0.843800, 如下截图



```

1:~/lab-2/part_1/src 96% 18:38
fushen in ai-fundamental/lab/lab-2/part_1/src on master [!?] via v3.11.3 took 39ms
> python Bayesian-network.py
test score: 0.843800

fushen in ai-fundamental/lab/lab-2/part_1/src on master [!?] via v3.11.3 took 1s
> 

```

图 1: Bayesian 网络运行结果

2.2. 利用 K-means 实现图片压缩

2.2.1. 实验原理与实现

在本次实验中, 输入为一张图片, 模型将对图片上的像素点使用 K-means 算法进行聚类, 并将所有数据点的值替换成其聚类中心的值, 以达到压缩图片的目的.

具体而言, 我们将图片的每个像素点映射到 RGB 三维空间的数据点 (R, G, B) , 对这些数据点进行聚类. 本次实验中定义数据点的距离为欧氏距离 (向量的 2-范数).

K-means 算法的伪代码如下:

K-means(k, D):

- 1 随机生成 k 个点作为初始聚类中心 $\mu = \{\mu_1, \dots, \mu_k\}$
- 2 **repeat**
- 3 将各个数据点 x 分配到最近的聚类中心 μ_j // assign_points()
- 4 更新聚类中心 $\mu_j = \frac{1}{|C_j|} \sum_{x \in C_j} x$ // update_centers()
- 5 **until** μ 不再变化或达到最大迭代次数
- 6 **return** 聚类中心 μ

对应的 Python 代码如下:

```

# k-means clustering
def fit(self, points):
    centers = self.initialize_centers(points)
    last_centers = centers.copy()
    for _ in range(self.max_iter):
        labels = self.assign_points(centers, points)
        centers = self.update_centers(centers, labels, points)
        diff = centers - last_centers
        if np.all(diff < 1e-5):
            break
        last_centers = centers.copy()
    return centers

```

其中

```
# Assign each point to the closest center
def assign_points(self, centers, points):
    n_samples, n_dims = points.shape
    labels = np.zeros(n_samples)
    dists = np.zeros((self.k, n_samples))
    for k in range(self.k):
        dists[k] = np.linalg.norm(points - centers[k], axis=1)
    labels = dists.argmin(axis=0)
    return labels

# Update the centers based on the new assignment of points
def update_centers(self, centers, labels, points):
    for k in range(self.k):
        centers[k] = points[labels == k].mean(axis=0)

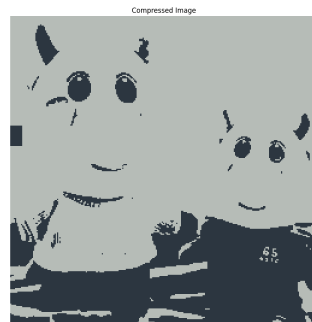
    return centers
```

2.2.2. 实验结果

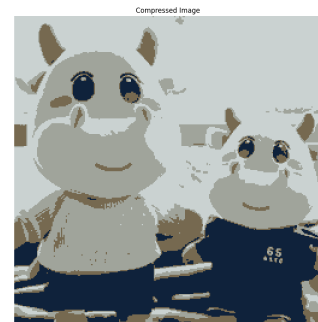
设定 k 分别为 2, 4, 8, 16, 32, 压缩后的图片如下所示:



a) 原始图片



b) $k = 2$



c) $k = 4$



d) $k = 8$



e) $k = 16$



f) $k = 32$

图 2: 压缩后的图片

3. 深度学习

3.1. 实验实现

基于提供的实验框架, 我对其中的代码进行了部分修改:

- (1) 直接实现 MultiHeadAttention 类, 而不是先实现 Head 再利用其实现 MultiHeadAttention.
- (2) 修改了 get_batch() 函数的实现以使其更高效;
- (3) 使用 tqdm 库显示训练进度.

下面将针对前两点进行说明.

3.1.1. MultiHeadAttention

MultiHeadAttention 类的实现如下:

```
class MultiHeadAttention(nn.Module):
    """multi-head self-attention"""

    def __init__(self, n_embd, n_heads):
        super().__init__()
        # parameters
        if (n_embd % n_heads != 0):
            raise ValueError("n_embd must be divisible by n_heads")

        self.n_heads = n_heads
        self.d_k = n_embd // n_heads

        self.qkv_proj = nn.Linear(n_embd, 3 * n_embd)
        self.out_proj = nn.Linear(n_embd, n_embd)

        # mask for attention
        self.register_buffer("tril",
                             torch.tril(torch.ones(block_size, block_size)))

    def forward(self, inputs):
        batch, time, _ = inputs.shape

        qkv = self.qkv_proj(inputs)
        qkv = qkv.reshape(batch, time, self.n_heads, -1)
        qkv = qkv.permute(0, 2, 1, 3)
        q, k, v = qkv.chunk(3, dim=-1)

        out = q @ k.transpose(-2, -1) / math.sqrt(self.d_k)
        out = torch.masked_fill(out,
                                self.tril[:time, :time] == 0,
                                float("-inf"))
        out = F.softmax(out, dim=-1) @ v

        out = out.permute(0, 2, 1, 3).reshape(batch, time, -1)
        out = self.out_proj(out)

        return out
```

在实现中, 我使用了一个线性层 `qkv_proj` 来一次性计算所有的 Q, K, V , 并以此来直接计算 `concat` 后的 `out` (即 `out_proj` 的输入).

3.1.2. `get_batch()`

如下代码是修改前后的代码对比:

```
# separate the dataset into train and validation
train_data = torch.tensor(encode(text[: -len(text) // 10]), dtype=torch.long)
val_data = torch.tensor(encode(text[-len(text) // 10 :]), dtype=torch.long)

def get_batch(split):
    data = train_data if split == "train" else val_data
    ix = torch.randint(len(data) - block_size, (batch_size,))
    x = torch.stack([data[i : i + block_size] for i in ix])
    y = torch.stack([data[i + 1 : i + block_size + 1] for i in ix])
    x, y = x.to(device), y.to(device)
    return x, y
```

```
# separate the dataset into train and validation
data = torch.tensor(encode(text), dtype=torch.long)
data_len = data.shape[0]
train_data = data[:-data_len // 10]
val_data = data[-data_len // 10:]

# Unfold into blocks
train_blocks = train_data.unfold(0, block_size, 1)
train_x = train_blocks[:-1]
train_y = train_blocks[1:]

val_blocks = val_data.unfold(0, block_size, 1)
val_x = val_blocks[:-1]
val_y = val_blocks[1:]

def get_batch(split: str):
    "Get batched data"
    if split == "train":
        x = train_x
        y = train_y
    elif split == "val":
        x = val_x
        y = val_y
    else:
        raise ValueError("split must be either 'train' or 'val'")
    idx = torch.randint(x.shape[0], (batch_size,))
    return x[idx].to(device), y[idx].to(device)
```

经过测试, 两个实现是等价的, 并且后者的速度会更快.

3.1.3. Transformer 生成文本

在 `Transformer.generate()` 中, 每轮迭代生成一个字符 (token) 添加到 `inputs` 末尾, 重复 `max_new_tokens` 次迭代. 其中每次生成都只会取 `inputs` 末尾的至多 `block_size` 个字符作为输入.

```
class Transformer(nn.Module):
    def generate(self, inputs, max_new_tokens):
        for _ in range(max_new_tokens):
            logits, _ = self.forward(inputs[:, -block_size:])
            probs = self.softmax(logits)
            next_token = torch.multinomial(probs[:, -1], num_samples=1)
            inputs = torch.cat([inputs, next_token], dim=1)
        return inputs

def generate(model, text = ""):
    if text == "":
        context = torch.zeros((1, 1), dtype=torch.long, device=device)
    else:
        context = torch.tensor([encode(text)], dtype=torch.long, device=device)
    print(decode(model.generate(context, max_new_tokens=500)[0].tolist()))
```

3.2. 实验结果

训练过程中的 Loss 变化如下图所示:

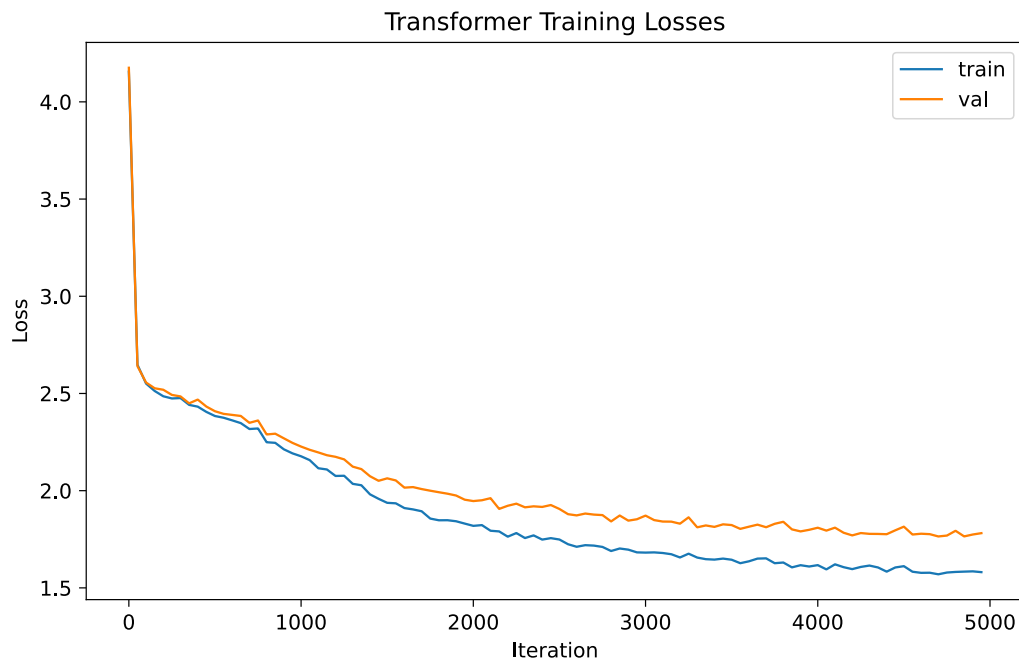


图 3: Transformer 训练过程中的 Loss 变化

`generate()` 输入的文本为 The meaning of life is, 模型输出如下截图所示:

```

① 1:~/lab-2/part_2/src 95% 21:52
fushen in ai-fundamental/lab/lab-2/part_2/src on  master [!?!] via  v3.10.12 via  ai-lab took 41ms
> python train.py
100%|██████████| 5000/5000 [18:15<00:00, 4.56it/s, last_train_loss=1.580911, last_val_loss=1.781459]
qt.qpa.wayland: Wayland does not support QWindow::requestActivate()
The meaning of life is naturea.
Ah, sir, for to clie onjure's daly dison; if singly.

Puty:
You home, for trans of dather's nothies fried.'

PETERO:
Welcome pertaly enop, who, I know have your lilo,
We should's being adm by boke lulinge
With with up to all refiencer touchf,
Privatul, Loyard'd joy? 'C, I
HORSTABRENG:
Nighter her, who sir, the his parporm twasan?
A woull, cretttation, thou hand slain a shine adks; I canno,
I have purposer o' a Oufier again,
Teache stin?

ESCALUS:
Oncest shall doubth them ime a Vol!

S

fushen in ai-fundamental/lab/lab-2/part_2/src on  master [!?!] via  v3.10.12 via  ai-lab took 19m22s
> 

```

The meaning of life is naturea.
 Ah, sir, for to clie onjure's daly dison; if singly.

Puty:
 You home, for trans of dather's nothies fried.'

PETERO:
 Welcome pertaly enop, who, I know have your lilo,
 We should's being adm by boke lulinge
 With with up to all refiencer touchf,
 Privatul, Loyard'd joy? 'C, I

HORSTABRENG:
 Nighter her, who sir, the his parporm twasan?
 A woull, cretttation, thou hand slain a shine adks; I canno,
 I have purposer o' a Oufier again,
 Teache stin?

ESCALUS:
 Oncest shall doubth them ime a Vol!

S

可以看到, 预测的字符能够组成类似英语单词的结构, 可以认为模型是学习到了一定的效果。

4. 总结与意见

在本次实验中, 我实现了两个传统机器学习算法 Bayesian 网络和 K-means 聚类, 并且实现了 Transformer 的 Decoder 模块。

意见: 希望设计 Transformer 实验的助教能注意一下自己的编码习惯 (不论是为了以后设计实验还是其他目的), 因为给出的实验框架中存在代码风格不一致的问题, 比如

- (1) class char_tokenizer 和 class MultiHeadAttention 的命名风格不一致。
- (2) class char_tokenizer 中的函数参数中有 Type Annotation, 其他函数均没有。