

贝叶斯网络手写数字识别

1. 实验内容

如图，有一张显示手写的屏幕，屏幕上的像素有亮/暗(1/0)两种状态。当字迹覆盖某对应位置的像素时，该像素为亮的状态，否则为暗的状态。本次实验中，你需要使用包含各像素信息和最终数字结果的训练数据搭建一个贝叶斯网络，并根据测试数据的像素亮暗预测对应的数字，与测试数据的真实标签进行比较并计算准确率。

Examples: CPTs

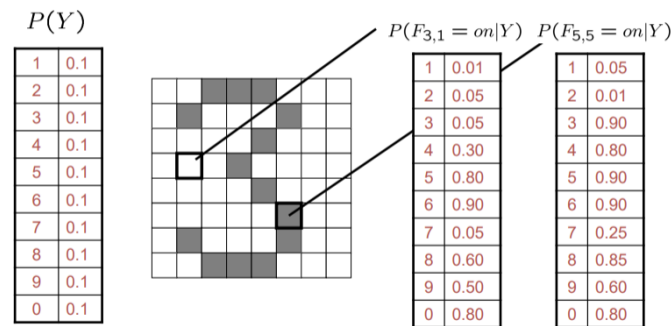


图 1: 示例贝叶斯网络

2. 实验原理

朴素贝叶斯提前假设：条件独立性，即

$$P(A, B|Y) = P(A|Y) \cdot P(B|Y)$$

通过对贝叶斯公式进行变形，得到

$$P(Y = c_k|X = x) = \frac{P(X = x|Y = c_k) \cdot P(Y = c_k)}{\sum_k P(X = x|Y = c_k) \cdot P(Y = c_k)}$$

又根据独立性假设：

$$P(X = x|Y = c_k) = \prod_{i=1}^n P(x_i|Y = c_k)$$

代入上述表达式，可以得到分类：

$$y = \operatorname{argmax}_{c_k} P(Y = c_k) \prod_j P(X = x_j | Y = c_k)$$

3. 具体实现

- 根据频次，分别计算先验概率，条件概率

```

1  # Calculate prior probability
2  for label in labels:
3      self.labels_prior[label] += 1
4
5  # Calculate conditional probability
6  for i in range(n_samples):
7      label = labels[i]
8      for pixel in range(self.n_pixels):
9          pixel_value = pixels[i, pixel]
10         self.pixels_prior[pixel, pixel_value] += 1
11         self.pixels_cond_label[pixel, pixel_value, label] += 1
12
13 # Normalize conditional probability
14 for pixel in range(self.n_pixels):
15     for value in range(self.n_values):
16         self.pixels_prior[pixel, value] /= n_samples
17         for label in range(self.n_labels):
18             self.pixels_cond_label[pixel, value, label] /=
19 self.labels_prior[label]
20 self.labels_prior /= n_samples

```

- 根据朴素贝叶斯算法，进行预测：

```

1  for i in range(n_samples):
2      pixel_values = pixels[i]
3      label_scores = np.zeros(self.n_labels)
4
5      for label in range(self.n_labels):
6          label_scores[label] = self.labels_prior[label]
7          for pixel in range(self.n_pixels):
8              pixel_value = pixel_values[pixel]
9              label_scores[label] *= self.pixels_cond_label[pixel, pixel_value,
10 label]
11
12 labels[i] = np.argmax(label_scores)

```

4. 实验结果

```

1  $ python .\Baysian-network.py
2  test score: 0.843800

```

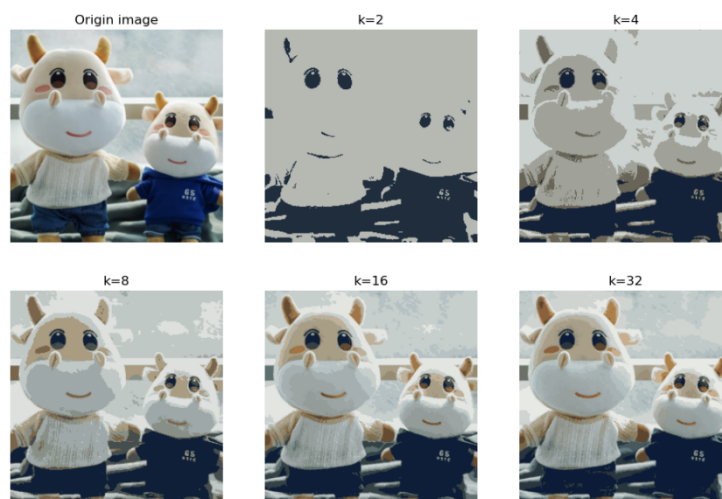
可以看到，基本完成了对数字的识别

利用 K-means 实现图片压缩

1. 实验内容

本次实验，你需要实现KMeans算法，并使用KMeans算法来压缩一张图像。使用KMeans压缩图像的原理为，将图像上的每一个像素(R,G,B)视作一个数据点，并对所有点进行聚类。完成聚类之后，将所有数据点的值替换成其聚类中心的值，这样仅需保留几个聚类中心的数据点值以及其他点的类别索引，从而达到压缩图片的目的。

本次实验中你需要分别尝试 $k=2, 4, 8, 16, 32$ 时的压缩结果。



2. 实验原理

- 随机选取 k 个样本作为初始的聚类中心
- 计算每个样本与各个聚类中心之间的距离，把每个样本分配给距离其最近的聚类中心
- 对于每一个新的聚类，重新距离每个样本之和最小的计算聚类中心
- 重复上述两步，直到中心不再移动

3. 具体实现

- 初始化（已在框架中实现）

```
1 n, d = points.shape
2 centers = np.zeros((self.k, d))
3 for k in range(self.k):
4     random_index = np.random.choice(n, size=10, replace=False)
5     centers[k] = points[random_index].mean(axis=0)
```

- 重新分配聚类

```
1 for i in range(n_samples):
2     distances = np.linalg.norm(points[i] - centers, axis=1)
3     labels[i] = np.argmin(distances)
```

- 计算新的聚类中心

```
1 new_centers = np.zeros_like(centers)
2 for k in range(self.k):
3     cluster_points = points[labels == k]
4     if len(cluster_points) > 0:
5         new_centers[k] = np.mean(cluster_points, axis=0)
6     else:
7         new_centers[k] = centers[k]
```




- 循环

```
1 for _ in range(self.max_iter):
2     labels = self.assign_points(centers, points)
3     new_centers = self.update_centers(centers, labels, points)
4     if np.all(centers == new_centers):
5         break
6     centers = new_centers
```

4. 实验结果



上方从左到右依次为原图， $k = 2$ ， $k = 4$ ， $k = 8$ ， $k = 16$ ， $k = 32$ 时候的结果，下方为压缩后文件的体积，可以看到K-means算法，顺利完成图像的压缩任务

 ustc-cow.png	2023/7/1 14:04	PNG 文件	104 KB
 compressed_image2.png	2023/7/1 17:25	PNG 文件	15 KB
 compressed_image4.png	2023/7/1 17:25	PNG 文件	20 KB
 compressed_image8.png	2023/7/1 17:25	PNG 文件	29 KB
 compressed_image16.png	2023/7/1 17:25	PNG 文件	39 KB
 compressed_image32.png	2023/7/1 17:25	PNG 文件	53 KB

深度学习

感谢gpt对本实验的大力支持

1. 实验内容

本次实验通过实现 **transformer decoder** 的架构，完成对莎士比亚文集的补全。实验的输入为 `input.txt`，其中是莎士比亚文集的语句，需要模型根据这些语句根据 **n-gram** 作为语言模型进行自监督学习。**n-gram** 的具体形式如下：

$$P(x_i | x_{i-1}, x_{i-2}, \dots, x_{i-(n-2)}, x_{i-(n-1)})$$

其中待预测的词 x_i ，是由其前面的 $n - 1$ 个词 $x_{i-(n-1)}, x_{i-(n-2)}, \dots, x_{i-1}$ 决定的。换言之，给定前 $n - 1$ 个词，来预测下一个词是什么。

例子

给定文本：

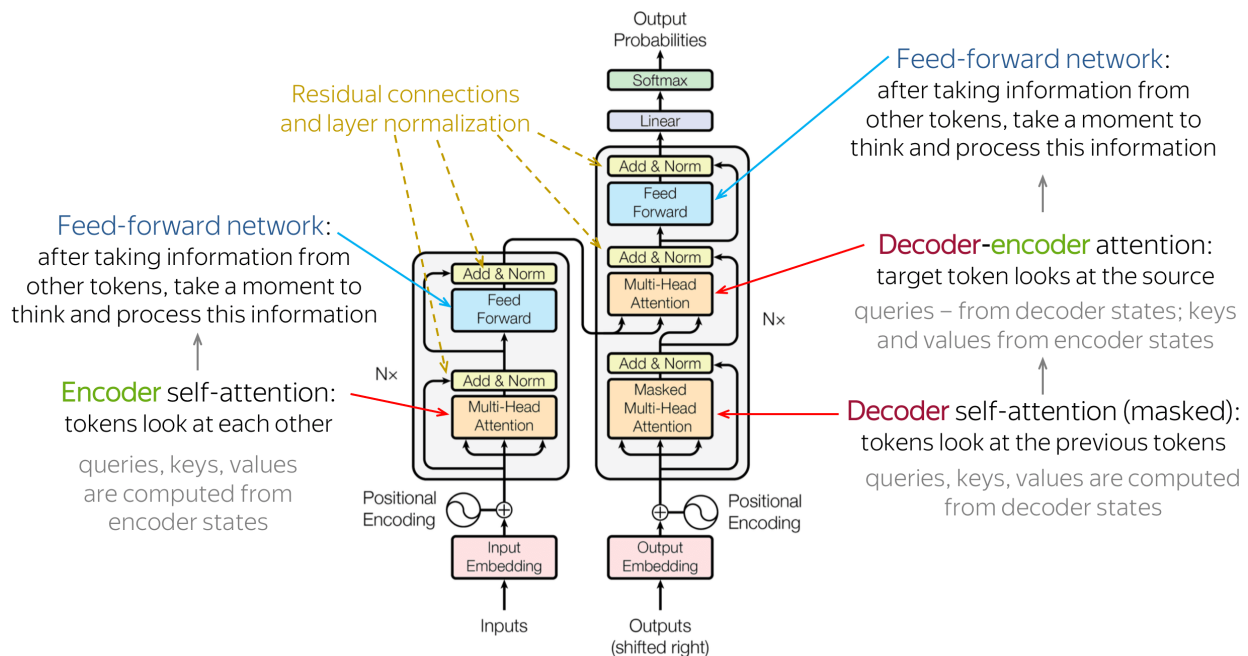
To be, or not to be: that is the

让模型去预测下一个词 **question**

To be, or not to be: that is the question

在 **transformer decoder** 中，模型会根据当前的输入去预测下一个词，之后，模型会将之前的输入与预测的词串联起来作为新的输入，去预测下下个词，这个过程不断迭代，直到预测得到的词为特殊的结束词 **EOS**（End-of-Sequence token）或者达到预测词数上限时终止预测（本实验中只需要给定预测词数上限即可）。

2. 实验原理



3. 具体实现

- `char_tokenizer`

实现对数据的编码操作，使用最简单的方式，按照序号编码

```
1 class char_tokenizer:
2     def __init__(self, corpus: List[str]):
3         self.corpus = corpus
4         self.n_vocab = len(corpus)
5         self.char2int = {char: i for i, char in enumerate(corpus)}
6     def encode(self, string: str):
7         return [self.char2int[char] for char in string]
8     def decode(self, codes: List[int]):
9         return ''.join([self.corpus[code] for code in codes])
10
```

- `Head`

表示一个注意力头，注意力头接收输入张量，并根据输入计算注意力权重，并将其应用于输入张量的不同位置

```
1 class Head(nn.Module):
2     def __init__(self, head_size):
3         super().__init__()
4         self.Key = nn.Linear(head_size, head_size)
5         self.Query = nn.Linear(head_size, head_size)
6         self.Value = nn.Linear(head_size, head_size)
7         self.register_buffer("tril", torch.tril(
8             torch.ones(block_size, block_size)))
9     def forward(self, inputs):
10        batch, time, n_embd = inputs.shape
11        keys = self.Key(inputs)
12        queries = self.Query(inputs)
13        values = self.Value(inputs)
14        scores = torch.matmul(queries, keys.transpose(1, 2))
15        scores = scores.masked_fill(
16            self.tril[:time, :time] == 0, float('-inf'))
17        attention_weights = F.softmax(scores, dim=-1)
18        out = torch.matmul(attention_weights, values)
19        return out
```

- `MultiHeadAttention`

在不同的子空间中计算多个注意力头，并将它们的输出进行连接和线性映射

```

1  class MultiHeadAttention(nn.Module):
2      def __init__(self, n_heads, head_size):
3          super().__init__()
4          self.heads = nn.ModuleList([Head(head_size)
5                                     for _ in range(n_heads)])
6          self.projection = nn.Linear(n_heads * head_size, head_size)
7      def forward(self, inputs):
8          head_outputs = [head(inputs) for head in self.heads]
9          out = torch.cat(head_outputs, dim=-1)
10         return self.projection(out)

```

- **FeedForward**

对输入的特征进行非线性变换和映射

```

1  class FeedForward(nn.Module):
2      def __init__(self, n_embd):
3          super().__init__()
4          self.net = nn.Sequential(
5              nn.Linear(n_embd, 4 * n_embd),
6              nn.ReLU(),
7              nn.Linear(4 * n_embd, n_embd)
8          )
9      def forward(self, inputs):
10         return self.net(inputs)

```

- **Block**

在输入序列上执行自注意力操作和前馈神经网络操作

```

1  class Block(nn.Module):
2      def __init__(self, n_embd, n_heads):
3          super().__init__()
4          self.attention = MultiHeadAttention(n_heads, n_embd)
5          self.norm1 = nn.LayerNorm(n_embd)
6          self.ffn = FeedForward(n_embd)
7          self.norm2 = nn.LayerNorm(n_embd)
8      def forward(self, inputs):
9          attention_output = self.attention(inputs)
10         attention_output = self.norm1(inputs + attention_output)
11         ffn_output = self.ffn(attention_output)
12         inputs = self.norm2(attention_output + ffn_output)
13         return inputs

```

- **Transformer**

主体结构，完成整体模型的组织

- 初始化

```

1 self.embedding = nn.Embedding(tokenizer.n_vocab, n_embd)
2 self.blocks = nn.ModuleList(
3     [Block(n_embd, n_heads) for _ in range(n_layers)])
4 self.norm = nn.LayerNorm(n_embd)
5 self.linear = nn.Linear(n_embd, tokenizer.n_vocab)

```

- 前向传播

```

1 def forward(self, inputs, labels=None):
2     batch, context = inputs.shape
3     embedding_output = self.embedding(inputs)
4     attens = embedding_output
5     for block in self.blocks:
6         attens = block(attens)
7     logits = self.linear(self.norm(attens))
8     if labels is None:
9         loss = None
10    else:
11        batch, time, channel = logits.shape
12        logits = logits.view(batch * time, channel)
13        labels = labels.view(batch * time)
14        loss = F.cross_entropy(logits, labels)
15    return logits, loss

```

- 预测 (用block_size-1个字符预测新的字符)

```

1 def generate(self, inputs, max_new_tokens):
2     for _ in range(max_new_tokens):
3         output, _ = self(inputs[:, -block_size + 1:])
4         predicted_token = output.argmax(dim=-1)[:, -1]
5         inputs = torch.cat([inputs, predicted_token[:, None]], dim=-1)
6     return inputs

```

4. 实验结果

- 实验环境

```

1 >>> import torch
2 >>> print(torch.__version__)
3 2.0.1+cu118
4 >>> print(torch.cuda.get_device_name(0))
5 NVIDIA GeForce RTX 4060 Laptop GPU

```

- 模型训练的超参数


```

1  batch_size = 16
2  block_size = 512
3  max_iters = 10000
4  eval_interval = 50
5  learning_rate = 1e-3
6  device = "cuda"
7  eval_iters = 200
8  n_embd = 64
9  n_heads = 8
10 n_layers = 8

```

- 训练过程截图

```

# define the model
model = Transformer().to(device)
train(model)
✓ 92m 49.2s

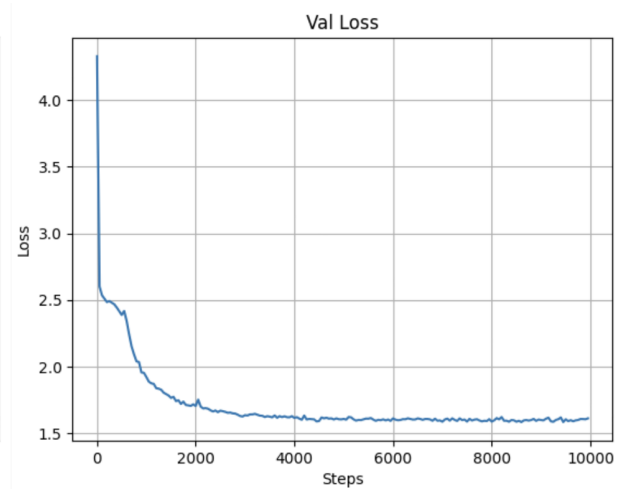
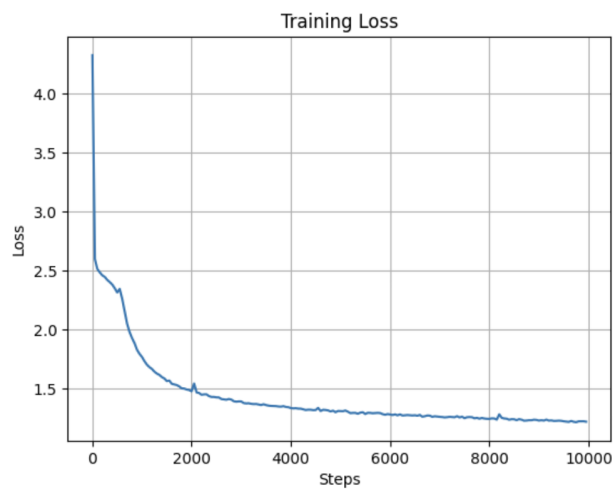
```

```

step 9250: train loss 1.2299, val loss 1.5850
step 9300: train loss 1.2254, val loss 1.5987
step 9350: train loss 1.2261, val loss 1.6042
step 9400: train loss 1.2278, val loss 1.6174
step 9450: train loss 1.2263, val loss 1.5829
step 9500: train loss 1.2221, val loss 1.6037
step 9550: train loss 1.2200, val loss 1.5896
step 9600: train loss 1.2170, val loss 1.5970
step 9650: train loss 1.2249, val loss 1.5887
step 9700: train loss 1.2180, val loss 1.5955
step 9750: train loss 1.2146, val loss 1.5982
step 9800: train loss 1.2218, val loss 1.6058
step 9850: train loss 1.2218, val loss 1.6054
step 9900: train loss 1.2223, val loss 1.6036
step 9950: train loss 1.2188, val loss 1.6106

```

- 训练误差和测试误差



- 测试预测功能

```
def generate(model):
    start_text = "GONZALO:\nI do well believe your highness; and"
    start_tokens = tokenizer.encode(start_text)
    context = torch.tensor([start_tokens], device=device, dtype=torch.long)
    # context = torch.zeros((1, 1), device=device, dtype=torch.long)
    print(decode(model.generate(context, max_new_tokens=3000)[0].tolist()))

generate(model)
```

✓ 1m 48.8s

GONZALO:

I do well believe your highness; and there is the sea,
And therefore the countern to the courting of him,
And there is no more to the people to him.

LEONTES:

The stand the statutes of his head and the course,
And then the sea through the course of his head,
And the sea the sea of his soul soul death,
And then the state of his soul souls and strike.

LEONTES:

The stand the state of his souls and the sea true,
And then the state of his soul souls and the sea,
And the state of his soul souls and the sea trutous,

可以看到，模型能够实现简单的预测功能

实验收获

- 对朴素贝叶斯模型，聚类算法，transformer有了更深的认识
- 对深度学习的过程有了初步的了解
- 虽然以后的方向和ai没有太大关系，但在实验过程中对ChatGPT的使用，让我深刻体会了人工智能的极大魅力