无需写代码能力，手搓最简单BabyGPT模型：前特斯拉AI总监新作

机器之心 2023-04-10 13:34 发表于北京

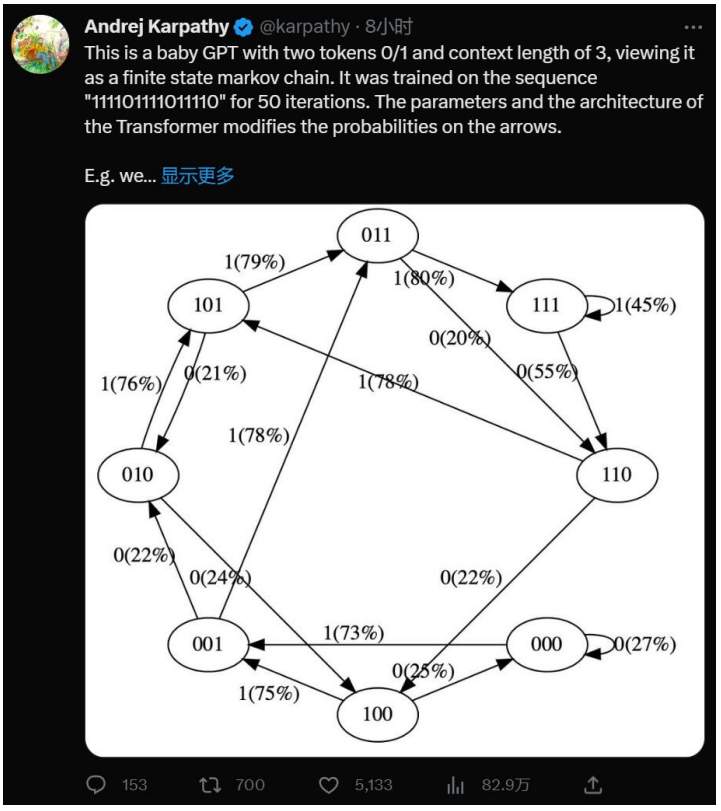
机器之心报道

机器之心编辑部

> GPT 原来这么简单？

我们知道，OpenAI 的 GPT 系列通过大规模和预训练的方式打开了人工智能的新时代，然而对于大多数研究者来说，语言大模型（LLM）因为体量和算力需求而显得高不可攀。在技术向上发展的同时，人们也一直在探索「最简」的 GPT 模式。

近日，特斯拉前 AI 总监，刚刚回归 OpenAI 的 Andrej Karpathy 介绍了一种最简 GPT 的玩法，或许能为更多人了解这种流行 AI 模型背后的技术带来帮助。



是的，这是一个带有两个 token 0/1 和上下文长度为 3 的极简 GPT，将其视为有限状态马尔可夫链。它在序列「111101111011110」上训练了 50 次迭代，Transformer 的参数和架构修改了箭头上的概率。

例如我们可以看到：

- 在训练数据中，状态 101 确定性地转换为 011，因此该转换的概率变得更高 (79%)。但不接近于 100%，因为这里只做了 50 步优化。
- 状态 111 以 50% 的概率分别进入 111 和 110，模型几乎已学会了（45%、55%）。
- 在训练期间从未遇到过像 000 这样的状态，但具有相对尖锐的转换概率，例如 73% 转到 001。这是 Transformer 归纳偏差的结果。你可能会想这是 50%，除了在实际部署中几乎每个输入序列都是唯一的，而不是逐字地出现在训练数据中。

通过简化，Karpathy 已让 GPT 模型变得易于可视化，让你可以直观地了解整个系统。

你可以在这里尝试它：
https://colab.research.google.com/drive/1SiF0KZJp75rUeetKOWqpsA8clmHP6jMg?usp=sharing

实际上，即使是 GPT 的最初版本，模型的体量很相当可观：在 2018 年，OpenAI 发布了第一代 GPT 模型，从论文《Improving Language Understanding by Generative Pre-Training》可以了解到，其采用了 12 层的 Transformer Decoder 结构，使用约 5GB 无监督文本数据进行训练。

但如果将其概念简化，GPT 是一种神经网络，它采用一些离散 token 序列并预测序列中下一个 token 的概率。例如，如果只有两个标记 0 和 1，那么一个很小的二进制 GPT 可以例如告诉我们：

```
[0,1,0] ---> GPT ---> [P (0) = 20%, P (1) = 80%]
```

在这里，GPT 采用位序列 [0,1,0]，并根据当前的参数设置，预测下一个为 1 的可能性为 80%。重要的是，默认情况下 GPT 的上下文长度是有限的。如果上下文长度为 3，那么它们在输入时最多只能使用 3 个 token。在上面的例子中，如果我们抛出一枚有偏差的硬币并采样 1 确实应该是下一个，那么我们将从原始状态 [0,1,0] 转换到新状态 [1,0,1]。我们在右侧添加了新位 (1)，并通过丢弃最左边的位 (0) 将序列截断为上下文长度 3，然后可以一遍又一遍地重复这个过程以在状态之间转换。

很明显，GPT 是一个有限状态马尔可夫链：有一组有限的状态和它们之间的概率转移箭头。每个状态都由 GPT 输入处 token 的特定设置定义（例如 [0,1,0]）。我们可以以一定的概率将其转换到新状态，如 [1,0,1]。让我们详细看看它是如何工作的：

```
# hyperparameters for our GPT
# vocab size is 2, so we only have two possible tokens: 0,1
vocab_size = 2
# context length is 3, so we take 3 bits to predict the next bit probability
context_length = 3
```

GPT 神经网络的输入是长度为 context_length 的 token 序列。这些 token 是离散的，因此状态空间很简单：

```
print ('state space (for this exercise) = ', vocab_size ** context_length)
# state space (for this exercise) = 8
```

细节：准确来说，GPT 可以采用从 1 到 context_length 的任意数量的 token。因此如果上下文长度为 3，原则上我们可以在尝试预测下一个 token 时输入 1 个、2 个或 3 个 token。这里我们忽略这一点并假设上下文长度已「最大化」，只是为了简化下面的一些代码，但这一点值得牢记。

```
print ('actual state space (in reality) = ', sum (vocab_size ** i for i in ran
# actual state space (in reality) = 14
```

我们现在要在 PyTorch 中定义一个 GPT。出于本笔记本的目的，你无需理解任何此代码。

现在让我们构建 GPT 吧：

```
config = GPTConfig (
    block_size = context_length,
    vocab_size = vocab_size,
    n_layer = 4,
    n_head = 4,
    n_embd = 16,
    bias = False,
)
gpt = GPT (config)
```

对于这个笔记本你不必担心 n_layer、n_head、n_embd、bias，这些只是实现 GPT 的 Transformer 神经网络的一些超参数。

GPT 的参数（12656 个）是随机初始化的，它们参数化了状态之间的转移概率。如果你平滑地更改这些参数，就会平滑地影响状态之间的转换概率。

现在让我们试一试随机初始化的 GPT。让我们获取上下文长度为 3 的小型二进制 GPT 的所有可能输入：

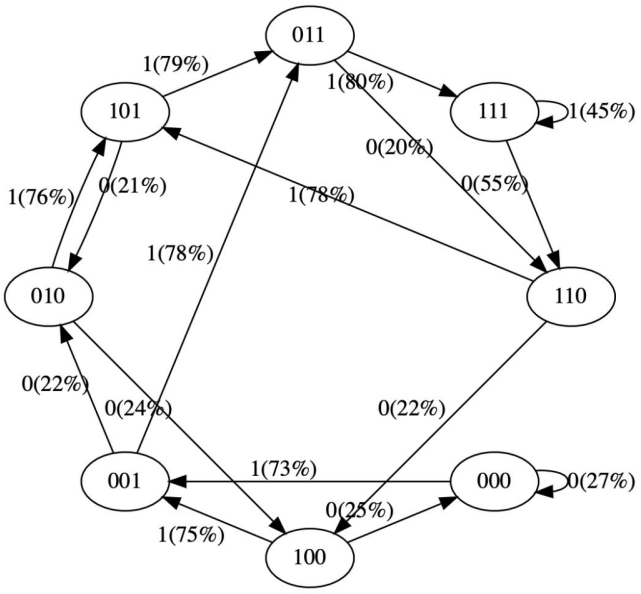
```
def all_possible (n, k):
    # return all possible lists of k elements, each in range of [0,n)
    if k == 0:
        yield []
    else:
        for i in range (n):
            for c in all_possible (n, k - 1):
                yield [i] + c
list (all_possible (vocab_size, context_length))
```

```
[[0, 0, 0],
 [0, 0, 1],
 [0, 1, 0],
 [0, 1, 1],
 [1, 0, 0],
 [1, 0, 1],
 [1, 1, 0],
 [1, 1, 1]]
```

这是 GPT 可能处于的 8 种可能状态。让我们对这些可能的标记序列中的每一个运行 GPT，并获取序列中下一个标记的概率，并绘制为可视化程度比较高的图形：

```
# we'll use graphviz for pretty plotting the current state of the GPT
from graphviz import Digraph


def plot_model ():
    dot = Digraph (comment='Baby GPT', engine='circo')


    for xi in all_possible (gpt.config.vocab_size, gpt.config.block_size):

        # forward the GPT and get probabilities for next token
        x = torch.tensor (xi, dtype=torch.long)[None, ...] # turn the list ir
        logits = gpt (x) # forward the gpt neural net
        probs = nn.functional.softmax (logits, dim=-1) # get the probabilitie
        y = probs [0].tolist () # remove the batch dimension and unpack the t
        print (f"input {xi} ---> {y}")


        # also build up the transition graph for plotting later
        current_node_signature = "".join (str (d) for d in xi)
        dot.node (current_node_signature)
        for t in range (gpt.config.vocab_size):
            next_node = xi [1:] + [t] # crop the context and append the next
            next_node_signature = "".join (str (d) for d in next_node)
            p = y [t]
            label=f"{t}({p*100:.0f}%)"
            dot.edge (current_node_signature, next_node_signature, label=labe

    return dot


plot_model ()
```

```
input [0, 0, 0] ---> [0.4963349997997284, 0.5036649107933044]
input [0, 0, 1] ---> [0.4515703618526459, 0.5484296679496765]
input [0, 1, 0] ---> [0.49648362398147583, 0.5035163760185242]
input [0, 1, 1] ---> [0.45181113481521606, 0.5481888651847839]
input [1, 0, 0] ---> [0.4961162209510803, 0.5038837194442749]
input [1, 0, 1] ---> [0.4517717957496643, 0.5482282042503357]
input [1, 1, 0] ---> [0.4962802827358246, 0.5037197470664978]
input [1, 1, 1] ---> [0.4520467519760132, 0.5479532480239868]
```



我们看到了 8 个状态，以及连接它们的概率箭头。因为有 2 个可能的标记，所以每个节点有 2 个可能的箭头。请注意，在初始化时，这些概率中的大多数都是统一的（在本例中为 50%），这很好而且很理想，因为我们甚至根本没有训练模型。

下面开始训练：

```
# let's train our baby GPT on this sequence
seq = list (map (int, "111101111011110"))
seq
```

```
[1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0]
```

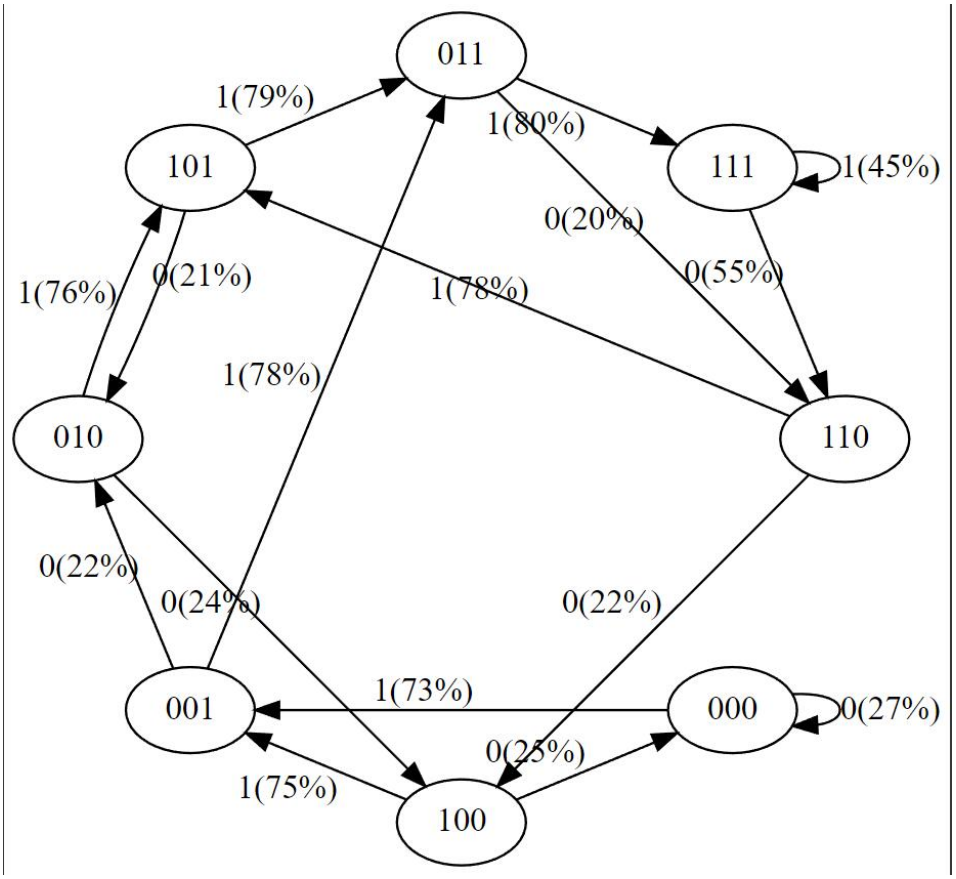
```
# convert the sequence to a tensor holding all the individual examples in tha
X, Y = [], []
# iterate over the sequence and grab every consecutive 3 bits
# the correct label for what's next is the next bit at each position
for i in range (len (seq) - context_length):
    X.append (seq [i:i+context_length])
    Y.append (seq [i+context_length])
    print (f"example {i+1:2d}: {X [-1]} --> {Y [-1]}")
X = torch.tensor (X, dtype=torch.long)
Y = torch.tensor (Y, dtype=torch.long)
print (X.shape, Y.shape)
```

我们可以看到在那个序列中有 12 个示例。现在让我们训练它：

```
# init a GPT and the optimizer
torch.manual_seed (1337)
gpt = GPT (config)
optimizer = torch.optim.AdamW (gpt.parameters (), lr=1e-3, weight_decay=1e-1)
```

```
# train the GPT for some number of iterations
for i in range (50):
    logits = gpt (X)
    loss = F.cross_entropy (logits, Y)
    loss.backward ()
    optimizer.step ()
    optimizer.zero_grad ()
    print (i, loss.item ())
```

```
print ("Training data sequence, as a reminder:", seq)
plot_model ()
```



我们没有得到这些箭头的准确 100% 或 50% 的概率，因为网络没有经过充分训练，但如果继续训练，你会期望接近。

请注意一些其他有趣的事情：一些从未出现在训练数据中的状态（例如 000 或 100）对于接下来应该出现的 token 有很大的概率。如果在训练期间从未遇到过这些状态，它们的出站箭头不应该是 50% 左右吗？这看起来是个错误，但实际上是可取的，因为在部署期间的真实应用场景中，几乎每个 GPT 的测试输入都是训练期间从未见过的输入。我们依靠 GPT 的内部结构（及其「归纳偏差」）来适当地执行泛化。

大小比较：

- GPT-2 有 50257 个 token 和 2048 个 token 的上下文长度。所以 `log2 (50,257) * 2048 = 每个状态 31,984 位 = 3,998 kB。这足以实现量变。
- GPT-3 的上下文长度为 4096，因此需要 8kB 的内存；大约相当于 Atari 800。
- GPT-4 最多 32K 个 token，所以大约 64kB，即 Commodore64。
- I/O 设备：一旦开始包含连接到外部世界的输入设备，所有有限状态机分析就会崩溃。在 GPT 领域，这将是任何一种外部工具的使用，例如必应搜索能够运行检索查询以获取外部信息并将其合并为输入。

Andrej Karpathy 是 OpenAI 的创始成员和研究科学家。但在 OpenAI 成立一年多后，Karpathy 便接受了马斯克的邀请，加入了特斯拉。在特斯拉工作的五年里，他一手促成了 Autopilot 的开发。这项技术对于特斯拉的完全自动驾驶系统 FSD 至关重要，也是马斯克针对 Model S、Cybertruck 等车型的卖点之一。

今年 2 月，在 ChatGPT 火热的背景下，Karpathy 回归 OpenAI，立志构建现实世界的 JARVIS 系统。

最近一段时间，Karpathy 给大家贡献了很多学习材料，包括详解反向传播的课程 、重写的 minGPT 库、从零开始构建 GPT 模型的完整教程等。

参考内容

https://twitter.com/karpathy/status/1645115622517542913

https://news.ycombinator.com/item?id=35506069

https://twitter.com/DrJimFan/status/1645121358471495680

畅聊「AIGC 技术探索与应用创新」，字节跳动 NLP 算法工程师陈家泽、英特尔 AI 软件工程师杨亦诚、Google Cloud 机器学习专家王顺、清华大学 KEG 知识工程实验室研究助理郑勤锴、九合创投 COO 张少宇、稀土掘金江昇等多位业界专家已集结完毕。

4月13日下午，北京大钟寺地铁站方恒时尚中心，邀你线下参会，更有多款稀土掘金原创周边等你来！

扫描下方二维码，抢线下免费参会票。

喜欢此内容的人还喜欢

开发者笑疯了！ LLaMa惊天泄露引爆ChatGPT平替狂潮，开源LLM领域变

天

新智元

GPT-4震撼发布：多模态大模型，直接升级ChatGPT、必应，开放API，游戏终结了？

机器之心

吴 军：ChatGPT不算新技术革命，带不来什么新机会

机器学习算法与自然语言处理