使用Kronecker因子近似的深度强化学习可扩展信赖域方法

作者姓名

2024 年 8 月 16 日

1 Reinforcement Learning and Actor-Critic Methods

1.1 强化学习的基本框架

强化学习（Reinforcement Learning, RL）涉及一个智能体（agent）与环境进行交互，以获得最大化累积回报的目标。强化学习问题通常被建模为一个无限时域的、折扣马尔可夫决策过程（discounted Markov Decision Process, MDP）。MDP的关键要素包括：

- 状态空间 $\mathcal{X}$
- 动作空间 $\mathcal{A}$
- 折扣因子 γ
- 转移概率 P
- 奖励函数 r

在每个时间步 t，智能体根据当前状态 s_t 选择一个动作 a_t，根据策略 $\pi_\theta(a_t|s_t)$ 确定动作 a_t。环境则会根据该动作和当前状态，生成下一个状态 s_{t+1}，并给予相应的奖励 $r(s_t, a_t)$。

1.2 智能体的目标

智能体的目标是通过策略 $\pi_\theta(a_t|s_t)$ 最大化期望的折扣累计回报 $J(\theta)$，其定义为：

$$J(\theta) = \mathbb{E}_\pi[R_t] = \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i r(s_{t+i}, a_{t+i})\right]$$

其中，γ 是折扣因子，通常在(0, 1)之间，用于权衡当前奖励和未来奖励的相对重要性。

1.3 策略梯度方法

策略梯度方法是直接对策略 $\pi_\theta(a_t|s_t)$ 进行参数化，并优化参数 θ 以最大化目标函数 $J(\theta)$。其策略梯度的通用形式为：

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \Psi^t \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

在这个公式中：

- Ψ^t 是时间步 t 的加权项，通常选择为优势函数 $A^\pi(s_t, a_t)$，它衡量在给定状态下采取特定动作的相对价值。
- $\log \pi_\theta(a_t|s_t)$ 是策略的对数概率，用于计算策略梯度。

1.4 优势函数的定义

优势函数 $A^\pi(s_t, a_t)$ 通常被定义为 k 步回报减去价值函数 $V_\phi^\pi(s_t)$：

$$A^\pi(s_t, a_t) = \sum_{i=0}^{k-1} \left(\gamma^i r(s_{t+i}, a_{t+i}) + \gamma^k V_\phi^\pi(s_{t+k})\right) - V_\phi^\pi(s_t)$$

- $\gamma^i r(s_{t+i}, a_{t+i})$ 表示未来奖励的折扣和。
- $V_\phi^\pi(s_t)$ 是价值函数，估计从状态 s_t 出发，按照策略 π 所能获得的期望总回报。

1.5 价值网络的训练

为了训练价值网络 $V_\phi^\pi(s_t)$，我们使用了时序差分（Temporal Difference, TD）更新方法，通过最小化引导的 k 步回报 $\hat{R}_t$ 和预测值 $V_\phi^\pi(s_t)$ 之间的平方差：

$$\frac{1}{2}\left\|\hat{R}_t - V_\phi^\pi(s_t)\right\|^2$$

这种方法通过当前的估计值和未来的回报调整价值网络的参数，以提高预测的准确性。

1.6 总结

这一节主要介绍了强化学习中的基本概念以及策略梯度方法的具体实现。它强调了如何通过策略梯度和优势函数来优化策略，以最大化累积回报，并讨论了在actor-critic方法中，如何通过价值网络的更新来进一步提高算法的效率和性能。这些概念和公式是理解强化学习算法（特别是actor-critic方法）的基础。

2 Natural Gradient Using Kronecker-Factored Approximation

2.1 非凸优化中的梯度更新

在优化非凸函数 $J(\theta)$ 时，最速下降法计算的更新量 $\Delta\theta$ 是通过最小化 $J(\theta+\Delta\theta)$ 来得到的，前提是满足约束条件 $\|\Delta\theta\|_B < 1$，其中 $\|\cdot\|_B$ 是一个由正半定矩阵 B 定义的范数：

$$\|\Delta\theta\|_B = \sqrt{\Delta\theta^\top B \Delta\theta}$$

约束优化问题的解为：

$$\Delta\theta = -B^{-1}\nabla_\theta J$$

其中，$\nabla_\theta J$ 是标准梯度。当 $B = I$（即Euclidean范数）时，这就成为常用的梯度下降方法。

然而，Euclidean范数的变化取决于参数化，这并不理想，因为模型的参数化是一个任意选择，不应影响优化路径。因此，自然梯度法通过使用Fisher信息矩阵 F 来构造范数，这种范数是基于KL散度的局部二次近似。它与模型参数化无关，提供了更稳定和更有效的更新。然而，由于现代神经网络可能包含数百万个参数，直接计算和存储精确的Fisher矩阵及其逆是不切实际的，因此需要进行近似。

2.2 Kronecker因子近似（K-FAC）

最近提出的一种技术称为Kronecker因子近似曲率（K-FAC），它对Fisher矩阵进行近似，以实现有效的自然梯度更新。我们令 $p(y|x)$ 表示神经网络的输出分布，$L = \log p(y|x)$ 表示对数似然。设 $\mathbf{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}}$ 是第 ℓ 层的权重矩阵，其中 C_{out} 和 C_{in} 分别是该层的输出/输入神经元的数量。设输入激活向量为 $\mathbf{a} \in \mathbb{R}^{C_{\text{in}}}$，下一层的预激活向量为 $\mathbf{s} = \mathbf{W}\mathbf{a}$。注意权重梯度为 $\nabla_{\mathbf{W}} L = (\nabla_{\mathbf{s}} L)\mathbf{a}^\top$。K-FAC利用这一事实，进一步近似第 ℓ 层对应的Fisher矩阵块 F_ℓ 为：

$$\begin{aligned} F_\ell &= \mathbb{E}[\text{vec}(\nabla_{\mathbf{W}} L)\text{vec}(\nabla_{\mathbf{W}} L)^\top] \\ &= \mathbb{E}[\mathbf{a}\mathbf{a}^\top \otimes \nabla_{\mathbf{s}} L (\nabla_{\mathbf{s}} L)^\top] \\ &\approx \mathbb{E}[\mathbf{a}\mathbf{a}^\top] \otimes \mathbb{E}[\nabla_{\mathbf{s}} L (\nabla_{\mathbf{s}} L)^\top] := A \otimes S = \hat{F}_\ell \end{aligned}$$

其中：

- $A = \mathbb{E}[\mathbf{a}\mathbf{a}^\top]$ 表示输入激活的二阶统计量。
- $S = \mathbb{E}[\nabla_{\mathbf{s}} L (\nabla_{\mathbf{s}} L)^\top]$ 表示反向传播梯度的二阶统计量。

这个近似可以解释为假设激活和反向传播梯度的二阶统计量是非相关的。利用这个近似，自然梯度更新可以通过以下基本恒等式有效计算：

$$(\mathbf{P}\otimes\mathbf{Q})^{-1} = \mathbf{P}^{-1}\otimes\mathbf{Q}^{-1} \quad \text{和} \quad (\mathbf{P}\otimes\mathbf{Q})\text{vec}(\mathbf{T}) = \mathbf{P}\text{vec}(\mathbf{T})\mathbf{Q}^\top$$

因此，Kronecker因子近似自然梯度更新为：

$$\begin{aligned} \text{vec}(\Delta\mathbf{W}) &= \hat{F}_\ell^{-1}\text{vec}(\nabla_{\mathbf{W}} J) \\ &= \text{vec}\left(A^{-1}\nabla_{\mathbf{W}} J S^{-1}\right) \end{aligned}$$

从上面的公式可以看出，K-FAC近似自然梯度更新只需要对与 $\mathbf{W}$ 大小相当的矩阵进行计算。Grosse和Martens最近将K-FAC算法扩展到卷积网络中。Ba等人进一步开发了该方法的分布式版本，其中大部分计算开销通过异步计算来减轻。分布式K-FAC在训练大规模现代分类卷积网络时，实现了2到3倍的速度提升。

2.3 总结

这一节主要讨论了如何在非凸优化中使用Kronecker因子近似来有效计算自然梯度更新。自然梯度方法通过使用Fisher信息矩阵来定义更新方向，从而减少了梯度更新中的参数依赖性。K-FAC方法通过将Fisher矩阵分解为Kronecker积的形式，大大降低了计算复杂度，使得在大型神经网络中的应用成为可能。

3 Natural Gradient in Actor-Critic

3.1 自然梯度的背景

自然梯度（Natural Gradient）最早由Kakade在十多年前提出，用于策略梯度方法。自然梯度的一个关键优

势是它可以提供比传统梯度下降更稳定的优化路径。然而，尽管自然梯度法的理论基础已经确立，但目前仍然缺乏一个可扩展、样本效率高且通用的自然策略梯度实例。

在本节中，我们引入了第一个适用于actor-critic方法的可扩展和样本高效的自然梯度算法，即基于Kronecker因子近似的信赖域方法（ACKTR）。这个方法通过Kronecker因子近似来计算自然梯度更新，并将自然梯度更新同时应用于actor（策略）和critic（价值）。

3.2 Fisher信息矩阵的定义

为了定义用于强化学习目标的Fisher度量，自然的选择是使用定义在当前状态上的策略函数，它定义了一个在当前状态下动作的分布，并对整个轨迹分布进行期望计算：

$$F = \mathbb{E}_{p(\tau)} \left[\nabla_\theta \log \pi(a_t|s_t) (\nabla_\theta \log \pi(a_t|s_t))^\top \right]$$

其中：
- $p(\tau)$ 是轨迹的分布，给定为 $p(s_0) \prod_{t=0}^{T} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t)$。
- $\pi(a_t|s_t)$ 是在状态 s_t 下采取动作 a_t 的策略。
- $\nabla_\theta \log \pi(a_t|s_t)$ 是策略的对数概率对参数 θ 的梯度。

在实践中，这种期望通常是通过在训练过程中收集的轨迹上进行近似计算的。

3.3 应用自然梯度优化critic

我们接着讨论如何应用自然梯度来优化critic。优化critic可以看作是一个最小二乘函数逼近问题，只不过目标是在不断变化的。在最小二乘函数逼近中，常用的二阶优化算法是高斯-牛顿（Gauss-Newton）法，它使用雅可比矩阵的近似曲率进行更新。

高斯-牛顿矩阵 G 定义为：

$$G := \mathbb{E}[J^\top J]$$

其中，J 是从参数到输出的映射的雅可比矩阵。

在这种设置下，高斯-牛顿矩阵等价于假设高斯观测模型下的Fisher矩阵。因此，可以将K-FAC应用于critic的优化。具体来说，假设critic的输出 v 定义为高斯分布 $p(v|s_t) \sim \mathcal{N}(v; V(s_t), \sigma^2)$，那么Fisher矩阵相对于这个高斯输出分布来定义。在实践中，我们可以简单地将 σ 设为1，这相当于传统的高斯-牛顿法。

3.4 共享网络架构下的自然梯度优化

如果actor和critic是独立的，则可以分别对两者应用K-FAC更新。但为了避免训练中的不稳定性，通常采用共享底层表示但具有不同输出层的网络架构。在这种情况下，我们可以通过假设两个输出分布相互独立来定义联合分布 $p(a, v|s) = \pi(a|s)p(v|s)$，并以此构建Fisher度量。

Fisher度量为：

$$F = \mathbb{E}_{p(\tau)} \left[\nabla \log p(a, v|s) \nabla \log p(a, v|s)^\top \right]$$

然后，可以使用K-FAC来近似这个Fisher矩阵，并同时对actor和critic进行更新。

3.5 其他优化细节

此外，我们还使用了Tikhonov正则化的因子分解阻尼方法，并通过异步计算的方式减少计算时间。

3.6 总结

这一节介绍了如何在actor-critic框架中应用自然梯度方法，特别是在共享网络结构下通过Kronecker因子近似来优化策略（actor）和价值函数（critic）。我们讨论了Fisher信息矩阵的构建，以及如何通过高斯-牛顿法的等价性将自然梯度应用于critic的优化。这些方法的核心是通过简化和加速计算来提高强化学习算法的效率和稳定性。

4 Step-Size Selection and Trust-Region Optimization

4.1 传统自然梯度更新与问题

传统上，自然梯度更新通常使用类似于随机梯度下降（SGD）的更新规则：

$$\theta \leftarrow \theta - \eta F^{-1} \nabla_\theta L$$

其中，η 是学习率，F^{-1} 是Fisher信息矩阵的逆，$\nabla_\theta L$ 是损失函数 L 对参数 θ 的梯度。

然而，在深度强化学习（Deep RL）中，Schulman等人发现，这种更新规则可能导致策略的大幅更新，从而使得算法过早地收敛到接近确定性的策略。这种情况不利于探索，可能导致较差的策略性能。

4.2 信赖域方法的引入

为了解决上述问题，我们提出使用信赖域方法，即通过将更新的幅度限制在一个指定的范围内（以KL散度度量），从而对策略分布的更新进行缩放。这种方法的目标是限制策略分布的变化，使其不超过指定的范围，从而避免过早收敛。

因此，我们采用了Schulman等人提出的基于K-FAC的信赖域方法，选择有效的步长η为：

$$\eta = \min\left(\eta_{\max}, \sqrt{\frac{2\delta}{\Delta\theta^\top \hat{F} \Delta\theta}}\right)$$

其中：- $\eta_{\max}$ 是最大学习率。- δ 是信赖域的半径。- $\hat{F}$ 是近似的Fisher信息矩阵。

步长 η 的选择不仅要考虑学习率 $\eta_{\max}$ 的上限，还要保证更新量 $\Delta\theta$ 不会导致策略分布的剧烈变化，即限制在信赖域半径 δ 内。

4.3 Actor和critic的分离与联合优化

如果actor和critic是独立的，那么需要为它们分别调整一组 $\eta_{\max}$ 和 δ 超参数。而如果它们共享表示（如共享底层网络结构），则需要为它们共同调整一组 $\eta_{\max}$ 和 δ，同时也要调整critic训练损失的加权参数，使其相对于actor的损失保持适当的比例。

此外，critic输出分布的方差参数可以被吸收到学习率参数中，这等同于传统的高斯-牛顿方法。如果actor和critic共享表示，还需要调整 $\eta_{\max}$、δ 以及critic损失的加权参数，以确保适当的优化效果。

4.4 总结

这部分内容探讨了在自然梯度方法中如何选择合适的步长，以避免策略的过度更新。信赖域方法通过限制策略分布的变化来防止算法过早收敛，同时通过调整步长参数来保证优化的稳定性。对于共享表示的actor-critic结构，我们建议通过联合调整超参数和损失加权参数来优化两者的表现。

5 Related Work

5.1 自然梯度的早期应用

自然梯度最早由Kakade提出，用于策略梯度方法。Bagnell和Schneider进一步证明了Kakade提出的度量实际上是由路径分布流形引出的协变度量。Peters和Schaal后来将自然梯度应用于actor-critic算法中，提出使用自然策略梯度算法更新actor，并使用最小二乘时序差分（LSTD）方法更新critic。

5.2 计算挑战与早期解决方案

自然梯度方法的一个主要挑战是如何高效地存储Fisher信息矩阵并计算其逆矩阵。为了解决这个问题，早期的研究限制了可兼容函数逼近器的使用（例如线性函数逼近器），以避免计算开销。信赖域策略优化（TRPO）提出了近似解决这个问题的方法，即通过共轭梯度法与快速Fisher矩阵-向量乘积来求解线性系统，这与Martens的工作类似。

然而，TRPO方法存在两个主要缺点：

1. 它需要反复计算Fisher向量积，这使得其难以扩展到通常用于从图像观测中学习的更大架构。
2. 它需要在训练期间通过大批量的rollouts来准确估计曲率。

5.3 K-FAC的优势

K-FAC方法通过两种方式克服了上述问题：

1. 它利用可跟踪的Fisher矩阵近似，通过在训练期间保持曲率统计的平均值来避免反复计算Fisher向量积。
2. 它将这些计算保持在可扩展的范围内，从而在更大规模的架构中也能有效应用。

尽管TRPO在每次迭代的进展上比基于一阶优化器（如Adam）的策略梯度方法表现得更好，但在样本效率上通常不如K-FAC方法。

5.4 其他改进TRPO的方法

Wang等人提出的方法试图改善TRPO的计算效率。为避免重复计算Fisher向量积，Wang等人将优化问题约

束在策略网络和当前策略网络之间的KL散度的线性近似上。与之相对，Heess等人和Schulman等人添加了一个KL约束作为目标函数的软约束。这两篇论文在连续和离散控制任务的样本效率上都比原始的策略梯度方法有所改善。

5.5 经验回放和辅助目标

还有一些研究引入了经验回放和辅助目标，以提高actor-critic模型的样本效率。这些方法与本文的工作是正交的，可以与ACKTR结合使用，以进一步增强样本效率。

5.6 总结

本节回顾了自然梯度法在策略梯度和actor-critic方法中的应用，讨论了当前方法的局限性，以及通过引入K-FAC等技术来改善这些不足的策略。我们总结了在不同任务中提高样本效率的相关方法，并强调了本文方法在提高计算效率和扩展性方面的贡献。

6 ACKTR算法代码解析与论文内容匹配

本节将详细解读ACKTR算法的代码实现，并将其与论文内容进行匹配。

6.1 代码结构概述

代码实现了ACKTR算法的关键部分，主要包括策略梯度（Policy Gradient）和价值函数（Value Function）的损失计算，以及K-FAC（Kronecker-Factored Approximate Curvature）优化器的应用。

6.2 代码具体部分与论文的匹配

6.2.1 策略梯度损失计算 `Pg_loss`

```
pg_loss = tf.reduce_mean(ADV*neglogpac)
```

匹配论文: 这部分代码计算了策略梯度损失，基于ADV（优势函数）和动作的负对数概率。这与论文中的策略梯度计算公式相匹配，即通过计算策略的log概率梯度更新策略参数。

6.2.2 熵正则化项 `Entropy`

```
entropy = tf.reduce_mean(train_model.pd.entropy())
pg_loss = pg_loss - ent_coef * entropy
```

匹配论文: 熵正则化用于防止策略过度确定性，保持探索能力。论文中提到为了平衡探索与利用，会使用熵作为正则化项。

6.2.3 价值函数损失 `Vf_loss`

```
vf_loss = tf.losses.mean_squared_error(tf.squeeze(
    train_model.vf), R)
```

匹配论文: 价值函数损失使用均方误差（MSE）来衡量预测的价值和实际收益之间的差异。论文中提到的Critic的优化与此部分相对应，强调了使用最小二乘法来优化价值函数。

6.2.4 Fisher信息矩阵的构造 `Pg_fisher`, `Vf_fisher`, `Joint_fisher`

```
self.pg_fisher = pg_fisher_loss = -tf.reduce_mean(
    neglogpac)
...
self.vf_fisher = vf_fisher_loss = - vf_fisher_coef*tf.
    reduce_mean(tf.pow(train_model.vf - tf.stop_gradient(
    sample_net), 2))
self.joint_fisher = joint_fisher_loss = pg_fisher_loss +
    vf_fisher_loss
```

匹配论文: Fisher信息矩阵的构造用于自然梯度的计算，尤其是联合策略和价值函数的Fisher损失。这部分实现了论文中提到的自然梯度计算，采用Kronecker分解来近似计算Fisher信息矩阵。

6.2.5 K-FAC优化器的应用

```
self.optim = optim = kfac.KfacOptimizer(learning_rate=
    PG_LR, clip_kl=kfac_clip, momentum=0.9, kfac_update
    =1, epsilon=0.01, stats_decay=0.99, is_async=is_async
    , cold_iter=10, max_grad_norm=max_grad_norm)
```

匹配论文: 这里的K-FAC优化器用于高效计算自然梯度更新，代码中的参数设置与论文中的描述一致。ACKTR算法通过K-FAC优化器来实现信赖域优化，确保算法的稳定性和计算效率。

6.2.6 梯度计算与应用 `Grads_check, Train_op`

```
self.grads_check = grads = tf.gradients(train_loss,params)
...
train_op, q_runner = optim.apply_gradients(list(zip(grads,
    params)))
```

匹配论文: 计算损失函数相对于模型参数的梯度，并应用这些梯度更新参数。该部分与论文中对K-FAC的应用一致，论文中提到使用自然梯度对actor和critic进行优化。

6.2.7 训练函数 `Train`

```
def train(obs, states, rewards, masks, actions, values):
    ...
    policy_loss, value_loss, policy_entropy, _ = sess.run
     ([pg_loss, vf_loss, entropy, train_op], td_map)
    return policy_loss, value_loss, policy_entropy
```

匹配论文: `train`函数执行了整个训练过程，包括策略更新和价值函数更新。它将观察到的状态、动作和奖励输入模型，计算损失并更新参数，这与论文中的整体训练流程相符。

6.3 总结

代码实现了ACKTR算法的核心步骤，从策略梯度和价值函数的损失计算到自然梯度的K-FAC优化。这些步骤在论文中均有详细阐述，代码与论文中的算法设计紧密对应。通过K-FAC的使用，ACKTR在计算效率和样本效率上实现了显著提升。