# 6. ЛЕКЦИЯ. Ценностно-ориентированный подход

Рассмотрим второй, Value-based (Ценностно-ориентированный) подход к решению задачи, в котором алгоритм ищет не саму стратегию, а оптимальную Q-функцию. Для этого табличный алгоритм Value Iteration (итерация значения) будет обобщён на более сложные пространства состояний; требование конечности пространства действий $|\mathcal{A}|$ останется ограничением подхода.

***Deep Q-learning (Глубокое Q-обучение)***

*Q-сетка*

В сложных средах пространство состояний может быть непрерывно или конечно, но велико (например, пространство всех экранов видеоигры). В таких средах моделировать функции от состояний, будь то стратегии или оценочные функции, мы можем только приближённо при помощи параметрических семейств.

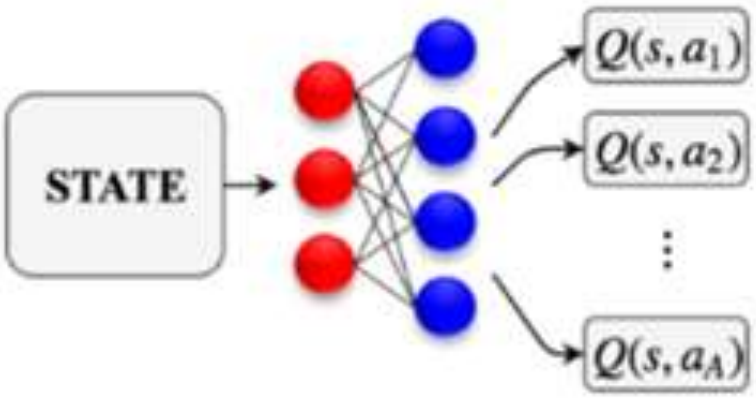



Рис.6.1.

Попробуем промоделировать в сложных средах алгоритм Q-learning. Для этого будем приближать оптимальную Q-функцию $Q^*(s, a)$ при помощи нейронной сети $Q_\theta(s, a)$ с параметрами $\theta$. Заметим, что для дискретных пространств действий сетка может как принимать действия на входе, так и принимать на вход только состояние $s$, а выдавать $|\mathcal{A}|$ чисел $Q_\theta(s, a_1) \dots Q_\theta\theta(s, a_{|\mathcal{A}|})$. В последнем случае мы можем за константу находить жадное действие $\pi(s) = arg\max_a Q_\theta(s, a)$.

В случае, если пространство действий непрерывно, выдать по числу для каждого варианта уже не получится. При этом, если непрерывное действие подаётся на вход вместе с состоянием, то оптимизировать по нему для поиска максимума или аргмаксимума придётся при помощи серии прямых и обратных проходов (для дискретного пространства — за $|\mathcal{A}|$ прямых проходов), что вычислительно нерентабельно. Поэтому такой вариант на практике не встречается, а алгоритм пригоден в таком виде только для дискретных

пространств состояний.

*Переход к параметрической Q-функции*

Как обучать параметры $\theta$ нейронной сети так, чтобы $Q_\theta(s,a) \approx Q^*(s,a)$? Вообще говоря, мы помним, что мы хотим решать уравнения оптимальности Беллмана, и можно было бы, например, оптимизировать невязку (величина ошибки (расхождения) приближённого равенства):

$$\left(Q_\theta(s,a) - r(s,a) - \gamma \mathbb{E}_{s'} \max_{a'} Q_\theta(s',a')\right)^2 \to \min_\theta$$

однако мат.ожидание $\mathbb{E}_{s'}$ в формуле берётся по неизвестному нам распределению (а даже если известному, то почти наверняка в сложных средах аналитически ничего не возьмётся) и никак не выносится (несмещённые оценки градиента нас бы устроили).

В Q-learning-е мы смогли с сохранением теоретических гарантий побороться с этим при помощи стохастической аппроксимации.

$$Q_\theta(s,a) \leftarrow Q_\theta(s,a) + \alpha\left(\mathrm{r(s,a)} + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a)\right)$$

Мы уже отмечали ключевое наблюдение о том, что формула стохастической аппроксимации очень напоминает градиентный спуск, а $\alpha$ играет роль learning rate (скорость обучения).

Пусть у нас есть текущая версия $Q_{\theta_k}(s,a)$, и мы хотим проделать шаг метода простой итерации. Зададим следующую задачу регрессии:

• входом является пара $s,a$

• искомым значением на паре $s,a$ — правая часть уравнения оптимальности Беллмана (3.17), т.е.

$$f(s,a) := r(s,a) + \gamma \mathbb{E}_{s'} \max_{a'} Q_{\theta_\mathrm{k}}(s',a')$$

• наблюдаемым («зашумлённым») значением целевой переменной или таргетом (target)

$$y(s,a) := r(s,a) + \gamma \max_{a'} Q_{\theta_\mathrm{k}}(s',a')$$

где $s'$~p($s'$|s,a)

• функцией потерь MSE: $Loss(y,\ \hat{y}) = \frac{1}{2}(y-\hat{y})^2$

Заметим, что, как и в классической постановке задачи машинного обучения, значение целевой переменной — это её «зашумлённое» значение: по входу $s,a$ генерируется $s'$, затем от $s'$ считается детерминированная функция, и результат $y$ является наблюдаемым значением. Мы можем для данной пары $s,a$ засэмплировать себе таргет, взяв переход $(s,a,r,s')$ и воспользовавшись

сэмплом $s'$, но существенно, что такой таргет — случайная функция от входа. Принципиально: согласно уравнениям Беллмана, мы хотим выучить мат.ожидание такого таргета, а значит, нам нужна квадратичная функция потерь.

Теорема: Пусть $Q_{\theta_{k+1}}(s,a)$ — достаточно ёмкая модель (model with enough capacity - модель с достаточной мощностью), выборка неограниченно большая, а оптимизатор идеальный. Тогда решением вышеописанной задачи регрессии будет шаг метода простой итерации для поиска оптимальной Q-функции:

$$Q_{\theta_{k+1}}(s,a) = \mathrm{r(s,a)} + \gamma \mathbb{E}_{s'} \max_{a'} Q_{\theta_{\mathrm{k}}}(s',a')$$

Другими словами, когда мы решаем задачу регрессии с целевой переменной $y(s,a)$ по формуле (наблюдаемым («зашумлённым») значением целевой переменной), мы в среднем сдвигаем нашу аппроксимацию в сторону правой части уравнения оптимальности Беллмана. Полезно наглядно увидеть это в формуле самого градиента. Представим на секунду, что мы решаем задачу регрессии с «идеальной», не зашумлённой целевой переменной $f(s,a) := r(s,a) + \gamma \mathbb{E}_{s'} \max_{a'} Q_{\theta_{\mathrm{k}}}(s',a')$: для одного примера $s, a$ градиент MSE тогда будет равен:

$$\nabla_\theta \frac{1}{2}(f(s,a) - Q_\theta(s,a))^2 = \overbrace{(f(s,a) - Q_\theta(s,a))}^{\text{скаляр}} \underbrace{\nabla_\theta Q_\theta(s,a)}_{\substack{\text{направление увеличения} \\ \text{значения } Q_\theta(s,a)}}$$

И наша «зашумлённая» целевая переменная $y(s,a)$ есть несмещённая оценка f(s,a), поскольку специально построена так, что $\mathbb{E}_{s'} y(s,a) = f(s,a)$. Значит, мы можем несмещённо оценить этот градиент как

$$\big(y(s,a) - Q_\theta(s,a)\big)\nabla_\theta Q_\theta(s,a) = \nabla_\theta \frac{1}{2}\big(y(s,a) - Q_\theta(s,a)\big)^2$$

Мы всегда оптимизируем нейронные сети стохастической градиентной оптимизации, поэтому несмещённая оценка градиента нам подойдёт. В частности, формула Q-learning — это частный случай решения указанной задачи регрессии стохастическим градиентным спуском для специального вида параметрических распределений:

Теорема: Пусть Q-функция задана «табличным параметрическим семейством», то есть табличкой размера $|\mathcal{S}|$ на $|\mathcal{A}|$, где в каждой ячейке записан параметр $\theta_{s,a}$:

$$Q_\theta(s,a) = \theta_{s,a}$$

Тогда формула Q-learning представляет собой градиентный спуск для решения обсуждаемой задачи регрессии.

*Таргет-сеть*

Рассмотренное теоретическое объяснение перехода от табличных методов к нейросетевым, конечно, предполагает, что мы решаем задачу регрессии «полностью», обучая $\theta$ при фиксированных $\theta_{\text{k}}$. «Замороженные» $\theta_{\text{k}}$ соответствуют фиксированию формулы целевой переменной $y(s,a)$, то есть фиксированию задачи регрессии. Так мы моделируем один шаг метода простой итерации, и только после этого объявляем выученные параметры модели $\theta_{k+1} := \theta$. В этот момент задача регрессии изменится (поменяется целевая переменная), и мы перейдём к следующему шагу метода простой итерации.

Но возникает вопрос: сколько шагов градиентного спуска тратить на решение фиксированной задачи регрессии? Возникает естественное желание по аналогии с табличными методами использовать для построения таргета свежую модель, то есть менять целевую переменную в задаче регрессии каждый шаг после каждого градиентного шага:

$$y(s,a) := r(s,a) + \gamma \max_{a'} Q_\theta(s',a')$$

Принципиально важно, что зависимость целевой переменной $y(s,a)$ от параметров текущей модели $\theta$ игнорировалось. Если вдруг в неё протекут градиенты, мы будем не только подстраивать прошлое под будущее, но и будущее под прошлое, что не будет являться корректной процедурой.

Эмпирически легко убедиться, что такой подход нестабилен примерно от слова совсем. Стохастическая оптимизация чревата тем, что после очередного шага модель может стать немножко «сломанной» и некоторое время выдавать неудачные значения на ряде примеров. В обычном обучении с учителем этот эффект сглаживается большим количеством итераций: обучение на последующих мини-батчах (пакетных файлов) «исправляют» предыдущие ошибки, движение идёт в среднем в правильную сторону, но нет гарантий удачности каждого конкретного шага (на то это и стохастическая оптимизация). Здесь же при неудачном шаге сломанная модель может начать портить целевую переменную, на которую она же и обучается. Это приводит к цепной реакции: плохая целевая переменная начнёт портить модель, которая начнёт портить целевую переменную... Этот эффект особенно ярко проявляется ещё и потому, что мы используем одношаговые целевые переменные, которые, сильно смещены (слишком сильно опираются на текущую же аппроксимацию).

Для стабилизации процесса одну задачу регрессии нужно решать более одной итерации градиентного спуска; необходимо сделать хотя бы условно 100-200 шагов. Проблема в том, что если таргет строится по формуле $r +$

$\gamma \max_{a'} Q_\theta(s, a)$, то после первого же градиентного шага $\theta$ поменяется.

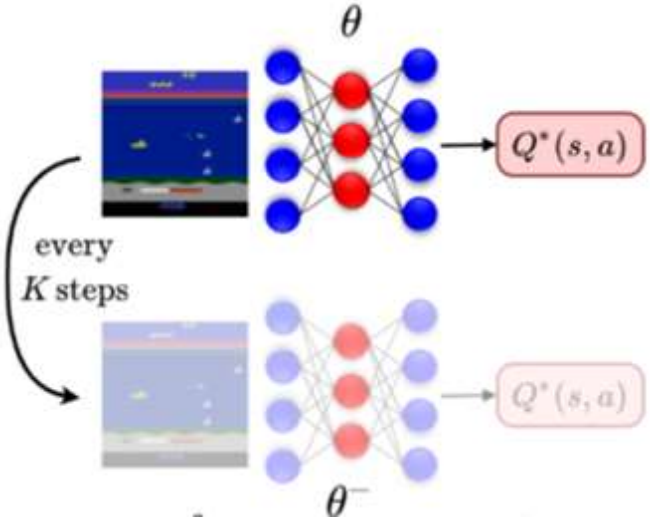



Рис.6.2.

Поэтому хранится копия Q-сетки, называемая таргет сетью (target network), единственная цель которой — генерировать таргеты текущей задачи регрессии для транзишнов (эффект плавного анимированного перехода от одного состояния к другому) из засэмплированных мини-батчей. Традиционно её параметры обозначаются $\theta^-$. Итак, целевая переменная в таких обозначениях генерится по формуле

$$y(\mathbb{T}) := r + \gamma \max_{a'} Q_{\theta^-}(s', a')$$

а раз в $K$ шагов веса $\theta^-$ просто копируются из текущей модели с весами $\theta$ для «задания» новой задачи регрессии.

*Декорреляция сэмплов*

Q-learning после одного шага в среде делал апдейт (обновления) одного значения таблицы $Q(s, a) \approx Q^*(s, a)$. Сейчас нас такой вариант, очевидно, не устраивает, потому что обучать нейросетки на мини-батчах размера 1 (особенно с уровнем доверия к нашим таргетам) — это явно плохая идея, но главное, сэмплы $s, a, y$ сильно скоррелированы: в сложных средах последовательности состояний обычно очень сильно похожи. Нейросетки на скоррелированных данных обучаются очень плохо (чаще — не обучаются вовсе). Поэтому сбор мини-батча подряд с одной траектории здесь не сгодится.

Есть два доступных на практике варианта декорреляции сэмплов (sample decorrelation). Первый — запуск параллельных агентов, то есть сбор данных сразу в нескольких параллельных средах. Этот вариант доступен всегда, по крайней мере, если среда виртуальна; иначе эта опция может быть дороговатой... Второй вариант — реплей буфер, который, как мы помним, является прерогативой исключительно off-policy (вне политики) алгоритмов. При наличии

реплей буфера агент может решать задачи регрессии, сэмплируя мини-батчи переходов $\mathbb{T} = (s, a, r', s')$ из буфера, затем делая для каждого перехода расчёт таргета $y(\mathbb{T}) := r + \gamma \max_{a'} Q_{\theta^-}(s', a')$, игнорируя зависимость таргета от параметров, и проводя шаг оптимизации по такому мини-батчу. Такой батч уже будет декоррелирован.

*DQN*

Собираем алгоритм целиком.

Алгоритм: Deep Q-learning (DQN)

**Гиперпараметры:** $B$ — размер мини-батчей, $K$ — периодичность апдейта таргет-сети, $\varepsilon(t)$ — стратегия исследования, $Q$ — нейросетка с параметрами $\theta$, SGD-оптимизатор

Инициализировать $\theta$ произвольно

Положить $\theta^- := \theta$

Пронаблюдать $s_0$

**На очередном шаге $t$:**

1. выбрать $a_t$ случайно с вероятностью $\varepsilon(t)$, иначе $a_t := \operatorname{argmax}_{a_t} Q_\theta(s_t, a_t)$
2. пронаблюдать $r_t$, $s_{t+1}$, $\text{done}_{t+1}$
3. добавить пятёрку $(s_t, a_t, r_t, s_{t+1}, \text{done}_{t+1})$ в реплей буфер
4. засэмплировать мини-батч размера $B$ из буфера
5. для каждого перехода $\mathbb{T} = (s, a, r, s', \text{done})$ посчитать таргет:
$$y(\mathbb{T}) := r + \gamma(1 - \text{done}) \max_{a'} Q_{\theta^-}(s', a')$$
6. посчитать лосс:
$$\text{Loss}(\theta) := \frac{1}{B} \sum_{\mathbb{T}} (Q_\theta(s, a) - y(\mathbb{T}))^2$$
7. сделать шаг градиентного спуска по $\theta$, используя $\nabla_\theta \text{Loss}(\theta)$
8. если $t \bmod K = 0$: $\theta^- \leftarrow \theta$

Все алгоритмы, которые относят к Value-based (Ценностно-ориентированному)подходу в RL, будут основаны на DQN: само название «valuebased» обозначает, что мы учим только оценочную функцию, а такое возможно только если мы учим модель Q-функции и полагаем, что policy improvement (улучшение политики) проводится на каждом шаге жадно. Неявно в DQN, конечно же, присутствует текущая политика, «целевая политика», которую мы оцениваем — $arg \max_a Q_\theta(s, a)$. Тем не менее ключевое свойство алгоритма, которое стоит помнить — это то, что он работает в off-policy режиме, и потому потенциально является достаточно sample efficient (эффективная выборка).

Есть, однако, много причин, почему алгоритм может не раскрыть этот потенциал и «плохо» заработать на той или иной среде: либо совсем не

обучиться, либо обучаться очень медленно. Со многими из этих недостатков можно пытаться вполне успешно бороться, что и пытаются делать модификации этого алгоритма,

***Модификации DQN***

*Overestimation Bias (Ошибка переоценки)*

Отмечалось, что без таргет-сетки (при обновлении задачи регрессии каждый шаг) можно наблюдать, как Q начинает неограниченно расти. Хотя таргет-сетка более-менее справляется с тем, чтобыстабилизироватьпроцесс, предотвратить этот эффект полностью у неё не получается: сравнение обучающейся Q с Монте-Карло оценками и зачастую просто со здравым смыслом выдаёт присутствие в алгоритме заметного смещения в сторону переоценки (overestimation bias). Почему так происходит? Очевидно, что источник проблемы — оператор максимума в формуле построения таргета:

$$y(\mathbb{T}) := r + \gamma \max_{a'} Q_{\theta^-}(s', a')$$

При построении таргета есть два источника ошибок: 1) ошибка аппроксимации 2) внешняя стохастика. Максимум здесь «выбирает» то действие, для которого из-за ошибки нейросети или из-за везения в прошлых играх $Q(s', a')$ больше правильного значения.

Утверждение: Рассмотрим одно $s'$. Пусть $Q^*(s', a')$ — истинные значения, и для каждого действия $a'$ есть модель $Q(s', a') \approx Q^*(s', a')$, которая оценивает это действие с некоторой погрешностью $\varepsilon(a')$:

$$Q(s', a') = Q^*(s', a') + \varepsilon(a')$$

Пусть эти погрешности $\varepsilon(a')$ независимы по действиям, а завышение и занижение оценки равеновероятны:

$$P(\varepsilon(a') > 0) = P(\varepsilon(a') < 0) = 0.5$$

Тогда оценка максимума скорее завышена, чем занижена:

$$P\left(\max_{a'} Q(s', a')\right) > \max_{a'} Q(s', a') > 0.5$$

Одно из хороших решений проблемы заключается в разделении (decoupling) двух этапов подсчёта максимума: выбор действия (action selection) и оценка действия (action evaluation):

$$\max_{a'} Q(s', a') = \overbrace{Q(s', \underbrace{\operatorname{argmax}_{a'} Q(s', a')}_{\text{выбор действия}})}^{\text{оценка действия}}$$

Действительно: мы словно дважды используем одну и ту же погрешность при выборе действия и оценке действия. Из-за скоррелированности ошибки на

этих двух этапах и возникает эффект завышения. Основная идея борьбы с этой проблемой заключается в следующем: предлагается3 обучать два приближения Q-функции параллельно и использовать аппроксимацию Q-функции «независимого близнеца» для этапа оценивания действия:

$$y_1(\mathbb{T}) \coloneqq r + \gamma Q_{\theta_2}(s', \operatorname*{argmax}_{a'} Q_{\theta_1}(s', a'))$$

$$y_2(\mathbb{T}) \coloneqq r + \gamma Q_{\theta_1}(s', \operatorname*{argmax}_{a'} Q_{\theta_2}(s', a'))$$

Если обе аппроксимации Q-функции идеальны, то, понятное дело, мы всё равно получим честный максимум. Однако, если оба DQN честно запущены параллельно и даже собирают каждый свой опыт (что в реальности, конечно, дороговато), их ошибки аппроксимации и везения будут в «разных местах». В итоге, Q-функция близнеца выступает в роли более пессимистичного критика действия, выбираемого текущей Q-функцией.

*Twin (близнец) DQN*

Разовьём интуицию дальше. Вот мы строим таргет для $Q_{\theta_1}$. Мы готовы выбрать при помощи неё же действия, но не готовы оценить их ею же самой в силу потенциальной переоценки. Для этого мы и берём «независимого близнеца» $Q_{\theta_2}$. Но что, если он выдаёт ещё больше? Что, если его оценка выбранного действия, так случилась, потенциально ещё более завышенная? Давайте уж в таких ситуациях всё-таки брать то значение, которое поменьше! Получаем следующую интересную формулу, которую называют twin-оценкой:

$$y_1(\mathbb{T}) \coloneqq r + \gamma \min_{i=1,2} Q_{\theta_i}(s', \operatorname*{argmax}_{a'} Q_{\theta_2}(s', a'))$$

$$y_2(\mathbb{T}) \coloneqq r + \gamma \min_{i=1,2} Q_{\theta_i}(s', \operatorname*{argmax}_{a'} Q_{\theta_1}(s', a'))$$

Это очень забавная формула, поскольку она говорит бороться с проблемой «клин клином»: зная, что взятие максимума по аппроксимациям приводит к завышению, мы в водим в оценку искусственное занижение, добавляя в формулу минимум по аппроксимациям! По сути, это формула «ансамблирования» Q-функций: только вместо интуитивного среднего берём минимум, «для борьбы с максимумом».

*Double (Двойной) DQN*

Запускать параллельно обучение двух сеток дороговато, а при общем реплей буфере корреляция между ними всё равно будет. Поэтому предлагается простая идея: запускать лишь один DQN, а в формуле таргета для оценивания в место «близнеца» использовать таргет-сеть. То есть: пусть $\theta$ —текущие веса, $\theta^-$ — веса таргет-сети, раз в $K$ шагов копирующиеся из $\theta$. Тогда таргет вычисляется

по формуле:

$$y(\mathbb{T}) := r + \gamma \mathrm{Q}_{\theta} - \left(s', \max_{a'} Q_{\theta}(s', a')\right)$$

Хотя понятно, что таргет-сеть и текущая сетка очень похожи, такое изменение формулы целевой переменной всё равно «избавляет» нас от взятия оператора максимума; эмпирически оказывается, что такая декорреляция действительно помогает стабилизации процесса. Приэтом, в отличие от предыдущих вариантов, такое изменение бесплатно: не требует обучения второй нейросети.

*Dueling (дуэль) DQN*

В сложных MDP ситуация зачастую такова, что получение негативной награды в некоторой области пространства состояний означает в целом, что попадание в эту область нежелательно. Верно, что с точки зрения теории остальные действия должны быть «исследованы», но неудачный опыт должен учитываться внутри оценки самого состояния; иначе агент будет возвращаться в плохие состояния с целью перепробовать все действия без понимания, что удача здесь маловероятна, вместо того, чтобы исследовать те ветки, где негативного опыта «не было». Понятно, что проблема тем серьёзнее, чем больше размерность пространства действий $|\mathcal{A}|$.

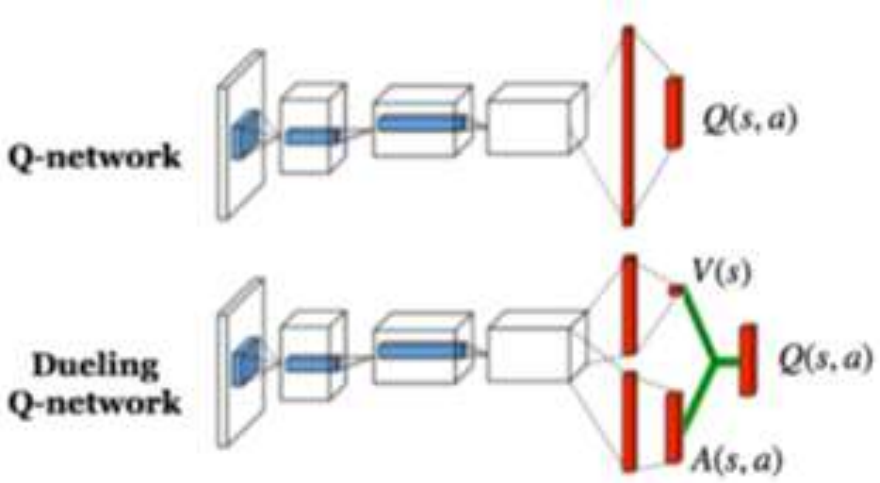



Рис.6.3.

Формализуя идею, мы хотели бы в модели учить не $Q^*(s, a)$ напрямую, а получать их с учётом $V^*(s)$. Иными словами, модель должна знать ценность самих состояний и с её учётом выдавать ценности действий. При этом при получении, скажем, негативной информации о ценности одного из действий, должна понижаться в том числе и оценка V-функции. Дуэльная (dueling) архитектура — это модификация вычислительного графа нашей модели с параметрами $\theta$, в которой на выходе предлагается иметь две головы, V-функцию

$V(s) \approx V^*(s)$ и Advantage (преимущество)-функцию $A(s,a) \approx A^*(s,a)$:

$$Q_\theta(s,a) := V_\theta(s) + A_\theta(s,a)$$

Это интересная идея решить проблему просто сменой архитектуры вычислительного графа: при апдейте значения для одной пары $s, a$ неизбежно поменяется значение $V_\theta(s)$ ценности всего состояния, которое общее для всех действий. Мы таким образом вводим следующий «прайор»: ценности действий в одном состоянии всё таки связаны между собой, и если одно действие в состоянии плохое, то вероятно, что и остальные тоже плохие.

Вообще-то, Advantage-функция не является произвольной функцией и обязана подчиняться. Для оптимальных стратегий в предположении жадности нашей стратегии это утверждение вырождается в следующее свойство:

Утверждение:

$$\forall s: \max_a A^*(s,a) = 0$$

Это условие можно легко учесть, вычтя максимум в формуле $Q_\theta(s,a)$

$$Q_\theta(s,a) := V_\theta(s) + A_\theta(s,a) - \max_{\hat{a}} A_\theta(s,\hat{a})$$

Таким образом мы гарантируем, что максимум по действиям последних двух слагаемых равен нулю, и они корректно моделируют Advantage (преимущество)-функцию.

Заметим, что $V$ и $A$ не учатся по отдельности (для $V^*$ уравнение оптимальности Беллмана не сводится к регрессии, для $A^*$ уравнения Беллмана не существует вообще); вместо этого минимизируется лосс (потеря) для Q функции точно так же, как и в обычном DQN.

*Шумные сети (Noisy Nets)*

По дефолту, алгоритмы на основе DQN решают дилемму исследования-использования при помощи примитивной $\varepsilon$-жадной стратегии взаимодействия со средой. Этот бэйзлайн-подход плох примерно всем, в первую очередь тем, что крайне чувствителен к гиперпараметрам: для $\varepsilon$ обязательно нужно составлять какое-нибудь расписание, чтобы в начале обучения он был побольше, а потом постепенно затухал, и откуда брать это расписание — непонятно. При этом слишком большие значения шума существенно замедляют обучение, заставляя агента вести себя случайно, а раннее затухание приведёт к застреванию алгоритма в каком-нибудь локальном оптимуме (агент будет биться головой об стенку, не пробуя её обойти).

Ключевая причина, почему ε-жадная стратегия примитивна, заключается в независимости добавляемого шума от текущего состояния. Мы выдаём оценки Q-функции и в зависимости только от времени принимаем решение,

использовать ли эти знания или эксплорить. Интуитивно, правильнее было бы принимать это решение в зависимости от текущего состояния: если состояние исследовано, чаще принимать решение в пользу использования знаний, если ново — в пользу исследования. Открытие новой области пространства состояний скорее всего означает, что в ней стоит поделать разные действия, когда двигаться к ней нужно за счёт использования уже накопленных знаний.

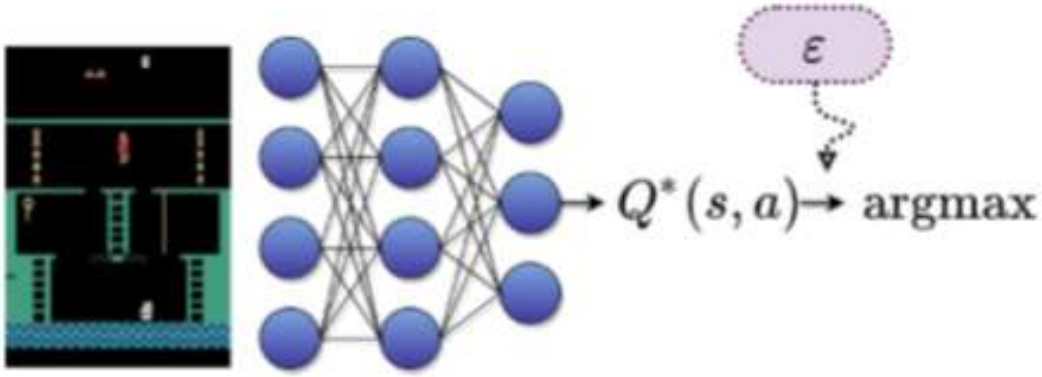



Рис.6.4.

Шумные сети (noisy nets) — добавление шума с обучаемой и, главное, зависимой от состояния (входа в модель) дисперсией. Хак (обходное техническое решение в программировании) чисто инженерный: давайте каждый параметр в модели заменим на

$$\theta_i := w_i + \sigma_i \varepsilon_i, \qquad \varepsilon_i \sim N(0,1),$$

или, другими словами, заменим веса сети на сэмплы из $N(w_i, \sigma_i^2)$, где $w, \sigma \in \mathbb{R}^h$ — параметры модели, обучаемые градиентным спуском. Очевидно, что выход сети становится случайной величиной, и, в зависимости от шума $\varepsilon$, будет меняться выбор действия $a = arg\max_a Q_\theta(s, a, \varepsilon)$. При этом влияние шума на принятое решение зависит от поданного в модель входного состояния.

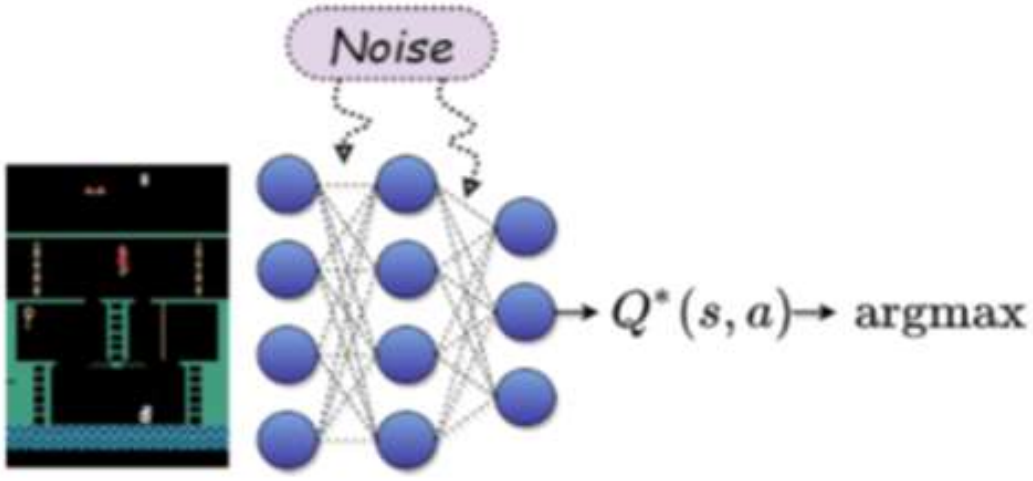



Рис.6.5.

Формально, коли наша модель стала стохастичной, мы поменяли оптимизируемый функционал: мы хотим минимизировать функцию потерь в среднем по шуму:

$$\mathbb{E}_{\varepsilon}\,\mathbf{Loss}(\theta,\varepsilon) \to \min_{\theta}$$

Видно, что градиент такого функционала можно несмещённо оценивать по Монте-Карло:

$$\nabla_\theta \mathbb{E}_\varepsilon Loss(\theta,\varepsilon) = \mathbb{E}_\varepsilon \nabla_\theta\, Loss(\theta,\varepsilon) \approx \nabla_\theta Loss(\theta,\varepsilon), \quad \varepsilon \sim N(0,I)$$

Гарантий, что магнитуда шума в среднем будет падать для исследованных состояний, вообще говоря, нет. Надежда этой идеи в том, что магнитуда будет подстраиваться в зависимости от текущих в модель градиентов: если модель часто видит какое-то $s$ и функция потерь говорит, что на этом состоянии нужно выдавать, скажем, низкое значение, модель будет учиться при любых сэмплах $\varepsilon$ выдавать указанное низкое значение. Для этого модели будет удобно уменьшать дисперсию впрыскиваемого шума и больше опираться на те нейронные связи, которые мало зашумлены. Если же в сеть поступают противоречивые сигналы о паре s,a, или это какое-то новое s, которого модель ещё не видела, выходное значение модели будет, интуитивно, сильно зашумлено, и часто аргмаксимум будет достигаться именно на нём.

Ещё одно ключевое преимущество идеи в том, что в этом подходе отсутствуют гиперпараметры.

Заметим, что таргет $y(\mathbb{T})$, который мы генерируем для каждого перехода $\mathbb{T}$ из батча, формально теперь тоже должен вычисляться как мат.ожидание по шуму:

$$y(\mathbb{T}) := \mathbb{E}_\varepsilon \left[ r \; + \gamma \max_{a'} Q_\theta(s', a', \varepsilon) \right]$$

Опять же, мат.ожидание несмещённо оценивается по Монте-Карло, однако с целью декорреляции полезно использовать в качестве ε другие сэмплы, нежели используемые при вычислении лосса. Считается, что подобное «зашумление» целевой переменной в DQN может даже пойти на пользу.

*Приоритизированный реплей (Prioritized DQN)*

Off-policy (вне политики) алгоритмы позволяют хранить и переиспользовать весь накопленный опыт. Однако, интуитивно ясно, что встречавшиеся переходы существенно различаются по важности. Зачастую большая часть буфера, особенно поначалу обучения, состоит из записей изучения агентом ближайшей стенки, а переходы, включавшие, например, получение ещё не выученной внутри аппроксимации Q-функции награды,

встречаются в буфере сильно реже и при равномерном сэмплировании редко оказываются в мини-батчах. Important one

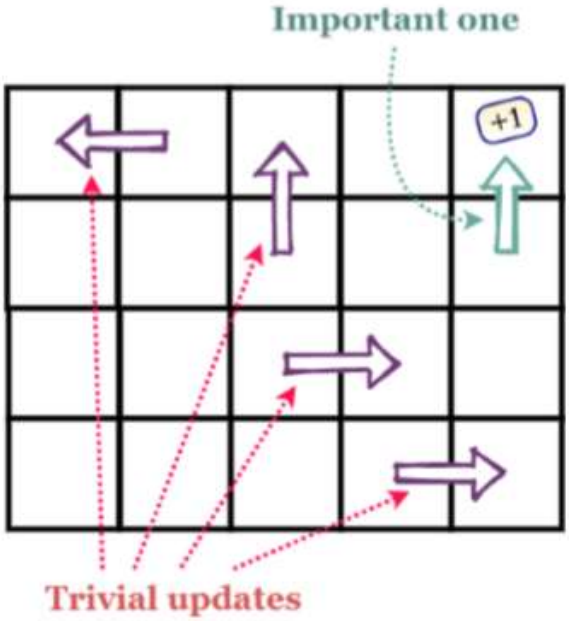



Рис.6.6. Trival updates (Тривиальные обновления), Important one (Важный)

Важно, что при обучении оценочных функций информация о награде распространяется от последних состояний к первым. Например, на первых итерациях довольно бессмысленно обновлять те состояний, где сигнала награды не было ($r(s,a)=0$), а Q-функция для следующего состояния примернослучайна(аименнотакиепереходычащевсегоипопадаютсяалгоритму). Такие обновления лишь схлопывают выход аппроксимации к константе (которая ещё и имеет тенденцию к росту из-за оператора максимума). Ценной информацией поначалу являются терминальные состояния, где целевая переменная по определению равна $y(\mathbb{T})=r(s,a)$ и является абсолютно точным значением $Q^*(s,a)$. Типично, что на таких переходах значения временной разницы (лосса DQN (потери)) довольно высоко. Аналогичная ситуация в принципе справедлива для любых наград, которые для агента новы и ещё не распространились в аппроксимацию через уравнение Беллмана.

Очень хочется сэмплировать переходы из буфера не равномерно, а приоритизировано. Приоритет установим, например, следующим образом:

$$\rho(\mathbb{T}) := (y(\mathbb{T}) - Q_\theta(s,a))^2 = Loss(y(\mathbb{T}), Q_\theta(s,a))$$

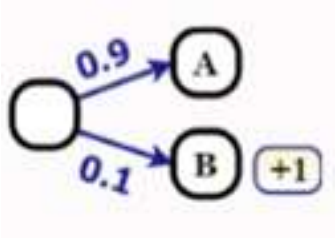
Сэмплирование переходов из буфера происходит по следующему правилу:

$$P(\mathbb{T}) \propto \rho(\mathbb{T})^\alpha$$

где гиперпараметр $\alpha > 0$ контролирует масштаб приоритетов (в частности, $\alpha = 0$ соответствует равномерному сэмплированию, когда $\alpha \to +\infty$ соответствовало бы жадному сэмплированию самых «важных» переходов).

Техническая проблема идеи заключается в том, что после каждого обновления весов сети $\theta$ приоритеты переходов меняются для всего буфера (состоящего обычно из миллионов переходов). Пересчитывать все приоритеты, конечно же, непрактично, и необходимо ввести некоторые упрощения. Например, можно обновлять приоритеты только у переходов из текущего батча, для которых значение лосса так и так считается. Это, вообще говоря, означает, что если у перехода был низкий приоритет, и до него дошла, условно, «волна распространения» награды, алгоритм не узнает об этом, пока не засэмплирует переход с тем приоритетом, который у него был.

Пример: Пусть для данной пары $s, a$ с вероятностью 0.9 мы попадаем в $s' = A$, а с вероятностью 0.1 мы попадаем в $s' = B$. В условно бесконечном буфере для этой пары $s, a$ среди каждых 10 сэмплов будет 1 сэмпл с $s' = B$ и 9 сэмплов с $s' = A$, и равномерное сэмплирование давало бы переходы, удовлетворяющие $s' \sim p(s'|s, a)$




Для приоритизированного реплея, веса у переходов с $s' = A$ могут отличаться от весов для переходов с $s' = B$. Например, если мы оцениваем $V(A) = 0, V(B) = 1$, и уже даже правильновыучилисреднеезначение $Q(s, a) = 0.1$, то $Loss(s, a, s' = A)$ будет равен $0.1^2$, а для $Loss(s, a, s' = B) = 0.9^2$. Значит, $s' = B$ будет появляться в засэмплированных переходах чаще, чем с вероятностью 0.1, и это выбьет $Q(s, a)$ с её правильного значения.

Иными словами, приоритизированное сэмплирование приводит к смещению (bias). Этот эффект не так страшен поначалу обучения, когда распределение, из которого приходят состояния, всё равно скорее всего не сильно разнообразно. Более существенно нивелировать этот эффект по ходу обучения, в противном случае процесс обучения может полностью дестабилизироваться или где-нибудь застрять. Заметим, что равномерное сэмплирование не является единственным «корректным» способом, но основным доступным. Мы не очень хотим «возвращаться» к нему постепенно с ходом обучения, но можем сделать похожую вещь: раз мы хотим подменить распределение, то можем при помощи importance sampling (выборка по важности) сохранить тот же оптимизируемый функционал:

Теорема: При сэмплировании с приоритетами $P(\mathbb{T})$ использование весов

$w(\mathbb{T}) := \frac{1}{P(\mathbb{T})}$ позволит избежать эффекта смещения.

Importance sampling (выборка по важности) подразумевает, что мы берём «интересные» переходы, но делаем по ним меньшие шаги (вес меньше именно для «приоритетных» переходов). Цена за такую корректировку, конечно, в том, что полезность приоритизированного сэмплирования понижается. Раз поначалу смещение нас не так беспокоит, предлагается вводить веса постепенно: а именно, использовать веса

$$w(\mathbb{T}) := \frac{1}{P(\mathbb{T})^{\beta(t)}}$$

где $\beta(t)$ — гиперпараметр, зависящий от итерации алгоритма $t$. Изначально $\beta(t = 0) = 0$, что делает веса равномерными (корректировки не производится), но постепенно $\beta(t)$ растёт к 1 и полностью избавляет алгоритм от эффекта смещения.

*Multi-step DQN (Многошаговый)*

Уже упоминалось, что DQN из-за одношаговых целевых переменных страдает от проблемы отложенного сигнала и сопряжённой с ней в контексте нейросетевой аппроксимации проблемы накапливающейся ошибки. Эта проблема фундаментальна для off-policy подхода: про bias-variance trade-off (компромисс между смещением и дисперсией) упоминалось, что разрешать дилемму смещения-разброса (то есть «проводить умный credit assignment (уступка кредита)») мы можем только в on-policy (с политикой) режиме. Многошаговый (multi-step) DQN — теоретически некорректная эвристика для занижения этого эффекта. Грубо говоря, нам очень хочется распространять за одну итерацию награду сразу на несколько шагов вперёд, то есть решать многошаговые уравнения Беллмана. Мы как бы и можем уравнение оптимальности многошаговое выписать

Утверждение — N-шаговое уравнение оптимальности Беллмана:

$$Q^*(s_0, a_0) = \mathbb{E}_{\mathcal{T}:_{N\sim\pi^*|s_0,a_0}} \left[ \sum_t^{N-1} \gamma^t r_t \; + \; \gamma^N \mathbb{E}_{s_N} \max_{a_N^*} Q^*(s_N, a_N^*) \right]$$

Что мы можем сделать? Мы можем прикинуться, будто решаем многошаговые уравнения Беллмана, задав целевую переменную следующим образом:

$$y(s_0, a_0) := \sum_t^{N-1} \gamma^t r_t \; + \gamma^N \; \max_{a_N} Q_\theta(s_N, a_N)$$

где $s_1, a_1 \ldots a_{N-1}, s_N$ взяты из буфера. Для этого в буфере вместо

одношаговых переходов $\mathbb{T}:=(s,a,r,s',done)$ достаточно просто хранить другую пятёрку:

$$\mathbb{T}:=\left(s,a,\sum_{n=0}^{N-1}\gamma^n r^{(n)},s^{(N)},done\right)$$

где $r^{(n)}$ —награда, полученная через n шагов после посещения рассматриваемого состояния $s$, $s^{(N)}$ —состояние, посещённое через $N$ шагов, и, что важно, флаг $done$ (сделанный) указывает на то, завершился ли эпизод в течение этого N-шагового роллаута (внедрения). Все остальные элементы алгоритма не изменяются, в частности, можно видеть, что случай $N=1$ соответствует обычному DQN.

Видно, что теперь награда, полученная за один шаг, распространяется на N состояний в прошлое, и мы таким образом не только ускоряем обучение оценочной функции стартовых состояний, но и нивелируем проблему накапливающейся ошибки.

*Retrace (Восстановление)*

Как мы обсуждали, теоретически корректным способом обучаться в off-policy с многошаговых оценок является использование Retrace оценки. Конечно, она может на практике схлопываться в одношаговые обновления, но по крайней мере гарантирует, что алгоритм не ломается; и важно, что если записанные в засэмплированном из буфера действия достаточно вероятны для оцениваемой политики, то оценка получается достаточно длинной.

Конечно, сложно говорить про «достаточно вероятны», когда оцениваемая политика детерминирована. Поэтому в практическом алгоритме Retrace предлагается перейти от моделирования Q-learning к моделированию SARSA то есть, считать целевой политикой $\pi(a|s)$ $\varepsilon$-жадную стратегию по отношению к текущей модели Q-функции. Преимущество в том, что это делает стратегию стохастичной, и любые действия в буфере не приведут к занулению следа и полному схлопыванию в одношаговую оценку.

В буфере так же нужно сохранять вероятности выбора сохранённых действий $\mu(a|s)$ в момент сбораданных (для $\varepsilon$-жадных стратегий эти значения всё время будут или $\frac{\varepsilon}{|A|}$, или $1-\varepsilon+\frac{\varepsilon}{|A|}$ , где $\varepsilon$ — параметр эксплорейшна на момент сбора перехода). Вместо отдельных переходов теперь хранятся роллауты — фрагменты траекторий некоторой длины $N$.

Далее для каждого засэмплированного из буфера роллаута $s_0,a_0,r_0,s_1,a_1,r_1,\dots s_N$ мы сначала для каждой пары $s_t,a_t$ считаем, используя формулы из теории Retrace, следующие вспомогательные величины: значение

коэффициента затухания следа $c_t$ (коэффициент $\lambda$ обычно полагают равным единице) и значение одношаговой ошибки $\Psi_{(1)}(s_t, a_t)$:

$$c_i := min\left(1, \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}\right)$$

$$\Psi_{(1)}(s_t, a_t) = r_t + \mathbb{E}_{\hat{a}_{t+1}\sim\pi} Q_{\theta^-}(s_{t+1}, \hat{a}_{t+1}) - Q_{\theta^-}(s_t, a_t)$$

Всюду, где используется $\pi$, используется $\varepsilon-$жадная стратегии по отношению к таргет-сети. В частности, мат.ожидание по $\pi$ можно посчитать явно:

$$\mathbb{E}_{\hat{a}\sim\pi} Q_{\theta^-}(s, \hat{a}) = (1-\varepsilon)\max_{\hat{a}} Q_{\theta^-}(s_{t+1}, \hat{a}_{t+1}) - Q_{\theta^-}(s_t, a_t)$$

После этого в Retrace для одной пары $s_t, a_t$ все будущие одношаговые ошибки нужно просуммировать (воспользуемся индексом $\hat{t}$ для обозначения этого перебора), но заглядывание на каждый следующий $i$-ый шаг в будущее обязывает нас потушить след в $c_i$ раз:

$$\Psi^{retrace}(s_t, a_t) := \sum_{\hat{t}\geq t}^{N} \gamma^{\hat{t}-1}\left(\prod_{i=t+1}^{i=\hat{t}} c_i\right)\Psi_{(1)}(s_{\hat{t}}, a_{\hat{t}})$$

Заметим, что в этой формуле внешняя сумма по $\hat{t}$ идёт не до бесконечности, как в теории Retrace, а до $N$, до конца роллаута. После этого считаем, что след зануляется: это корректно, хотя иногда можно и потерять возможность получить более длинную оценку. Далее в табличном методе мы бы провели обновление по формуле

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Psi^{retrace}(s,a)$$

то есть воспользовались бы $\Psi^{retrace}(s,a)$ как градиентом. Другими словами, оценка указывает, нужно ли увеличивать выход модели для рассматриваемой пары $s, a$ или уменьшать. Чтобы получить задачу регрессии, целевая переменная строится по формуле

$$y(s_t, a_t) := \Psi^{retrace}(s_t, a_t) + Q_\theta(s_t, a_t)$$

и дальше оптимизируется MSE, игнорируя зависимость $y$ от $\theta$:

$$(y(s_t, a_t) - Q_\theta(s_t, a_t))^2 \to \min_\theta$$

Тогда градиент функции потерь по $\theta$ для одного примера равен:

$$\nabla_\theta \frac{1}{2}(y(s,a) - Q_\theta(s,a))^2 = (y(s,a) - Q_\theta(s,a))\nabla_\theta Q_\theta(s,a) = \Psi^{\text{retrace}}(s,a)\nabla_\theta Q_\theta(s,a)$$