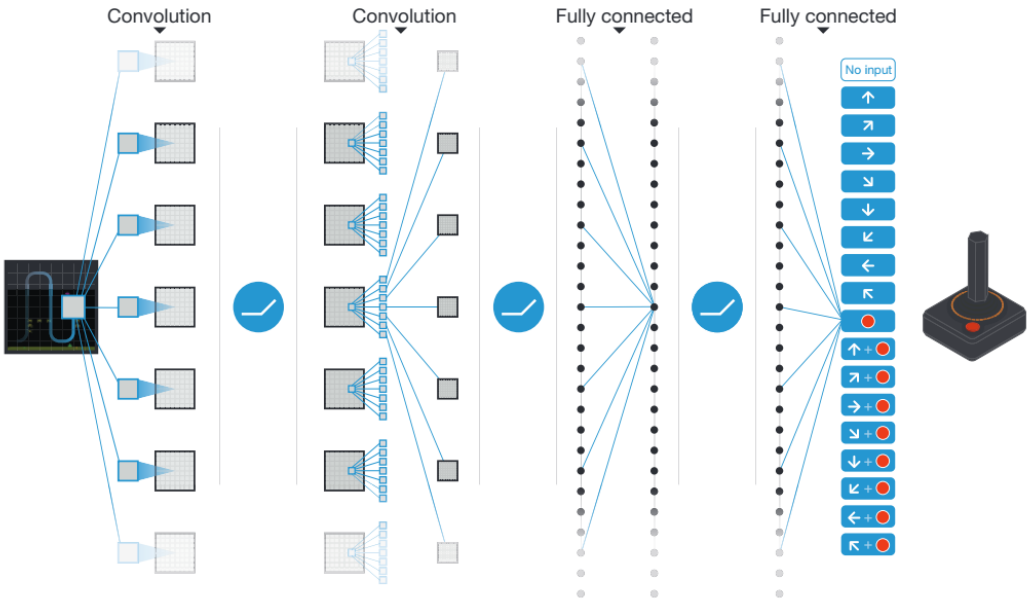
通过深度强化学习达到类人智能

第一章 “Human-level control through deep reinforcement learning”理解

1998 年 Sutton, R. & Barto, A.提出强化学习理论是在心理学和神经科学层面上研究动物如何在复杂环境中优化行为以适应外界变化。然而，如果智能体想要适应纷繁复杂的真实世界，那么必须应对一项非常复杂的任务：智能体必须能够接收高维认知输入，并且要具有突出的泛化能力。显然，人类和其他动物解决这个复杂任务都是通过强化学习和分层信息处理二者结合。大量的神经科学数据已经揭示了多神经元之间通过多巴胺并行地传递信号，这正是时序差分(temporal difference)强化学习算法，同时也证实了高级智慧擅长采用强化学习。应该注意到，在当下的研究中，强化学习智能体在很多方向取得了成功应用，但是，他们还都局限于使用手工提取的特征、或者低维空间。论文中介绍了一种新的人工智能体：深度 Q-网络，使用端到端的强化学习来处理高维输入。该智能体在 Atari 2600 的 49 个游戏中进行实验，只接受画面的像素和得分作为输入，智能体的表现超越了之前的所有算法，达到了专业游戏玩家的水准。这项工作的把高维信息输入和动作响应动作联系到了一起，第一次展示了智能体能够应对复杂的挑战性任务。

研究人员刚开始想为通用人工智能开发一个算法，可以应对多种任务。于是结合强化学习和深度神经网络提出了深度 Q 网络--DQN。其中，深度神经网络通过一些由节点组成的层级结构来逐步地从数据中提取特征，这对于人工神经网络来说，可以直接从原始的传感器信息中学习概念，比如：目标分类。而且，DQN 使用了深度卷积网络 DCNN (见图一)，DCNN 使用分层的卷积滤波器来模仿感受野效应，感受野的概念来自对动物视觉皮层前馈处理的理解。利用图像中局部信息，建立诸如立体视觉中的视角转换等。



图一. CNN 原理图

网络的输入是由映射函数ϕ预处理过的 84x84x4 图像，接着就是三个卷积层(蓝色的蛇形线表示卷积核在图像上的滑动轨迹)和两个全连接层，最后是一个输出层，输出层中每一个神经元表示一个有效的动作。网络中的每一隐层后都紧跟了一个非线性函数(这里是 max(0,x)).

智能体之间通过观察、动作响应、奖励三种机制来相互作用。智能体的目标是选择最佳动作响应以最大化累积奖励。在数学上可用下面的公式表述：

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, a_t = a, \pi]$$

其中，每做一次观察 s，都会有一个动作响应 a，接着选择行为策略$\pi = P(a|s)$。同时在每一时间步 t，奖励的最大值 r_t 都会被乘以 γ 衰减。

强化学习在使用非线性函数（比如一个神经网络）来近似动作值函数(action-value function)比如：Q 函数时，会变得不稳定甚至偏差很大。导致这种不稳定的原因大致如下：

- 观察值序列的相关性；
- 对 Q 很小的更新或许会明显地改变了策略；
- 或者改变了数据分布；
- 或者动作值 Q 和目标值$r + \gamma \max_{a'} Q(s', a')$的相关性。

通过下面两个思路改进 Q-learning 可以解决这些不稳定：

第一，采用生物启发机制—经验回溯法 (experience replay)，从历史数据中随机回溯，可以消除观察值序列的相关性并能平滑掉数据分布的改变；

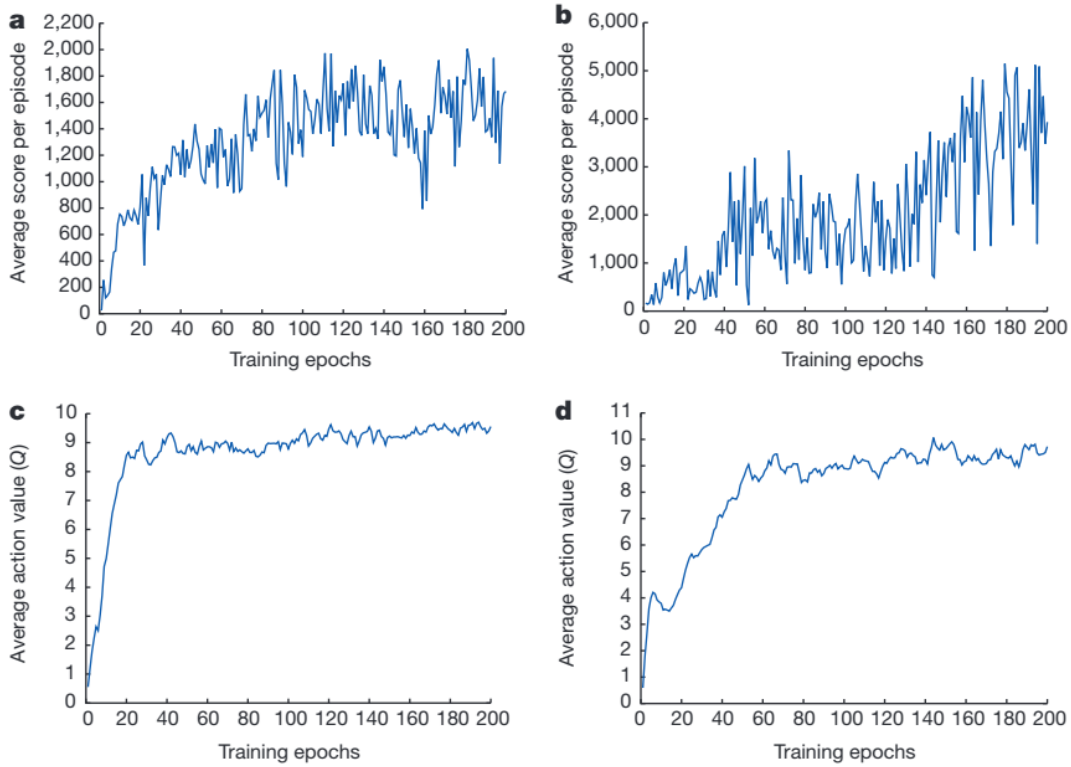
第二，采用迭代更新以调整动作值函数 action-values (Q)，而目标值也周期性更新，因此减少了二者之间的相关性。

在强化学习过程中，还应用到了其他的技巧来稳定地训练神经网络，比如自适应 Q 迭代的神经元(neural fitted Q-iteration)，这些方法中包含了在几百次迭代中从头重复训练网络。所以，对于大规模神经网络显得太低效了，而 DQN 则对于训练大型神经网络则很有效。使用深度卷积网络近似表示值函数 $Q(s,a;\theta_i)$，其中 θ_i 是 Q 网络第 i 次迭代的权重。为了执行经验回溯法，把智能体的每一时间步 t 的经验$e_{t=}(s_t, a_t, r_t, s_{t+1})$保存到集合$D_t = \{e_1, \ldots, e_t\}$。在智能体训练学习的过程中，应用 Q-learning 法更新，在对经验集合抽样的时候要服从均匀分布。每次 Q-learning 更新时，使用下面的损失函数：

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right]$$

式子中 γ 是衰减系数，表明智能体的 horizon，θ_i是 Q 网络第 i 次迭代的参数，θ_i^-是第 i 次迭代中用来计算目标值(target)的网络参数。θ_i^-每隔固定步长 C 就更新一次，而且只随着θ_i更新。

评估 DQN 智能体的时候设计了这样一个实验：在 Atari 2600 游戏平台上约有 49 种不同的游戏，在每次验证时，都使用相同网络架构、相同超参数、相同的学习处理过程，而且采用分辨率为 210x160x60HZ 的彩色视频作为唯一输入信息流（这样更能显示出智能体的鲁棒性），只输入智能体图像数据和允许的动作数，不告诉智能体这二者之间关系。显然，DQN 能够使用强化过的信号来训练大型神经网络，同时也使用了随机梯度下降(stochasticgradient descent) SGD 算法，SGD 通过学习过程的两个参数（智能体每一轮的平均得分和 Q 的预测值）来实现神经网络的时序进化（见图 2）。



图二. 智能体平均得分和平均动作预测值的训练曲线

A 图中，每一个点代表智能体在对 520k 帧 Space Invader 游戏图像上运行ε-贪婪策略(ε = 0.05)获得的平均值

B 图是显示了 Seaquest 游戏中，每轮获得平均分

C 图是在 Space Invaders 游戏中，留存状态集中的平均预测动作值。曲线上的每个点都是动作值函数 Q 在留存状态集上计算得到的。

D 图是 Seaquest 游戏中平均预测动作值

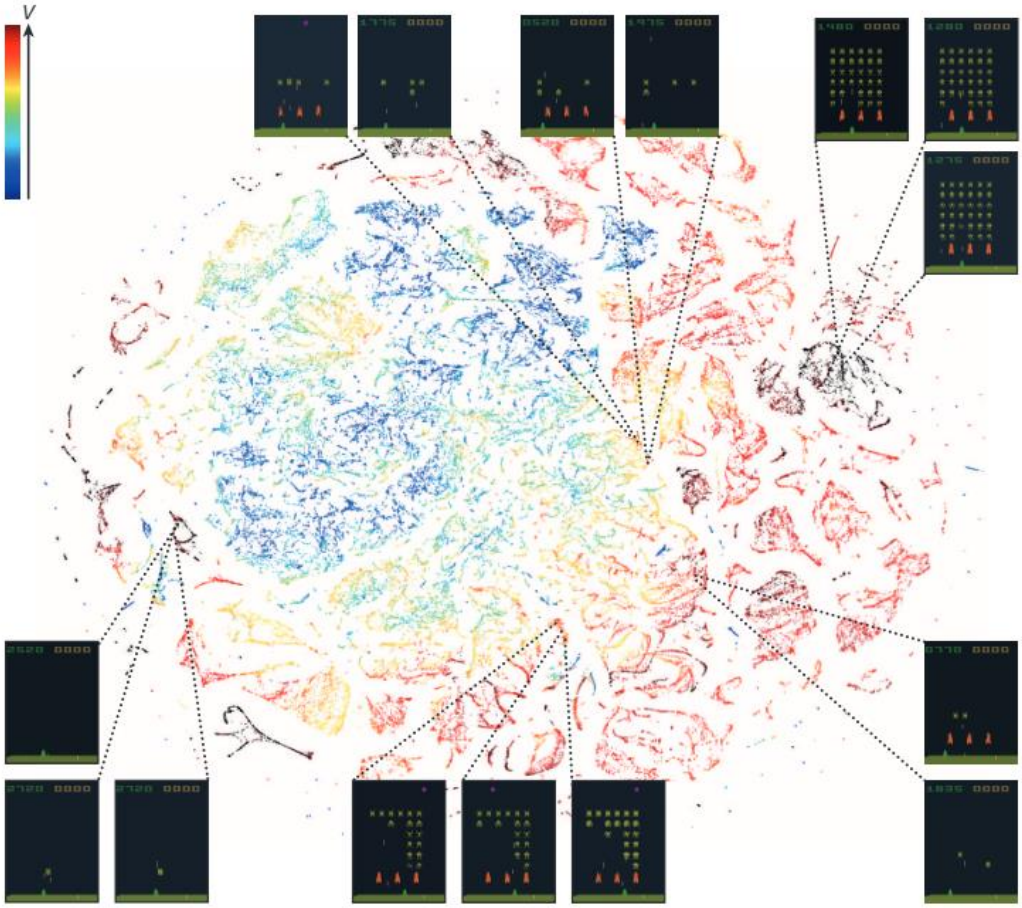
对比了 DQN 和现有最好的强化学习在这 49 个游戏中表现，DQN 在其中 43 个游戏中表现最好。另外，还对比了专业人类玩家与学习好的智能体之间在游戏策略上的差异，DQN 算法在 29 个游戏项目中达到了人类分数的 75%。

在仿真实验中，通过使能各个核心模块，验证了这些模块和技术对智能体的影响，比如：记忆回溯，分离 Q 网络和 DCNN 等。

如果使用处理高维数据可视化技术“t-SNE”（见图四），还可以继续增强 DQN 在游戏中的表现。t-SNE 算法试图将 DQN 中看起来的相似状态聚类到一个点上。但是我们把这些看着相似的状态映射到他们各自的期望奖励上时，又发现这些奖励看起来差异很大。DQN 网络的确能够从高维输入中自适应学习到相关描述(representation)。把人类和智能体在游戏中的表现作为网络中游戏的经验状态，记录在最后一个隐层的描述中，将 t-SNE 算法后的结果可视化。扩展数据图 2 展示了 DQN 如何精确地预测状态和动作值。

PS: t-SNE(t-distributed stochastic neighbor embedding)是用于降维的一种机器学习算法，是由 Laurens van der Maaten 和 Geoffrey Hinton 在 2008 年提出。此外，t-SNE 是一种非线性降维算法，非常适用于高维数据降维到 2 维或者 3 维，进行可视化。对称 SNE 实际上在高维度下 另外一种减轻”拥挤问题”的方法：在高维空间下，在高维空间下我们使用高斯分布将距离转换为概率分布，在低维空间下，我们使用更加偏重长尾分布的方式来将距离转换为概率分布，使得高维度下中低等的距离在映射后能够有一个较大的距离。我们对比一下高

斯分布和 t 分布,t 分布受异常值影响更小，拟合结果更为合理，较好的捕获了数据的整体特征。此外，t 分布是无限多个高斯分布的叠加，计算上不是指数的，会方便很多。对于较大相似度的点，t 分布在低维空间中的距离需要稍小一点；而对于低相似度的点，t 分布在低维空间中的距离需要更远。即同一簇内的点(距离较近)聚合的更紧密，不同簇之间的点(距离较远)更加疏远。t-SNE 的梯度更新有两大优势：对于不相似的点，用一个较小的距离会产生较大的梯度来让这些点排斥开来;这种排斥又不会无限大(梯度中分母)，避免不相似的点距离太远。



图四. DQN 在 Space Invaders 游戏中，对最后一个隐层对游戏状态的描述进行 2 维的 t-SNE 聚类

这个图是让 DQN 智能体玩两个小时的游戏，然后对最后一个隐层所记录的游戏状态进行 t-SNE 聚类算法生成的。颜色混合的区域是 DQN 根据游戏状态所做的预测状态值(V，状态的期望奖励最大值)，黑红色是 V 的最大值，黑色是 V 的最小值。图中可以看出：DQN 智能体对右上角的状态预测了高状态值，而对左下角的状态预测了很低，这是因为智能体已经学习到了如果完全清除掉屏幕中的敌人，那么就会导致游戏屏幕重新刷满敌人。如果只清除掉部分敌人(图中的底部状态)那么会得到较低的状态值，因为可被击杀的敌人少了，所以可获得奖励也少了。图中的左上角和右下角还有中间的部分虽然看着没有上面两个例子那么相似，但是也被映射到了相近的位置，这是因为橙色的区域在同一等级下不能突出更明显的强调了。

DQN 在简单的特定的游戏中可以玩得非常好，比如：射击或者赛车游戏，这样考验反应力正是程序非常擅长的。

但是，DQN 也能在一些长期策略的游戏中学到点什么。比如 Breakout(逃脱类游戏)，智能体能够学习到这样一个优化策略：直接在挖一条隧道到墙的后面并把球送到掩体后面，然

后破坏掉许多障碍。还有一个视频演示了 DQN 在训练过程中如何进化的。但是呢，对于更复杂的策略类游戏，DQN 网络就胜任不了。

总结一下，这篇论文的主要贡献就是设计了一种只利用一点点先验知识就能够从复杂环境中学习控制策略的算法架构，这个架构只需要接收像素和游戏得分作为输入，在每一把游戏中使用相同的网络架构和超参数，就像人类玩家玩游戏的过程一样。与前人的工作相比，这种包含 DQN 的架构实现了“端到端”的强化学习，并使用奖励模型和深度卷积网络提出环境特征以优化预测价值。文中的方法也借鉴了神经学中研究：对于灵长类动物，感知学习中的奖励信号或许会影响大脑视觉皮层的特征表现（大脑总是能够快速匹配到已经见过的事物）。最重要的是，经验回溯算法是强化学习与深度网络架构结合的关键，同时涉及了经验策略的存储与近期经验转换。现有的证据也表明哺乳类动物大脑中的海马体或许就是完成上述过程的物质基础，针对海马体在离线周期内（比如：醒着休息）对近期经验的重激活的过程可以做一个这样的假设：值函数或许是通过与基底神经节的交互来进行有效地更新的。

在以后的研究中还有以下的研究方向：

探索如何挖掘对特殊事件的经验回溯内容；

海马体回溯历史经验具有一定倾向性的现象；

强化学习中“优先擦除(prioritized sweeping)”的概念；

总结起来一句话，利用最先进的机器学习算法和生物启发机制创建的智能体具有强大的学习能力，足以胜任许多有挑战性的任务。

强化学习概念的理解：

在训练的过程中，不断的去尝试，错了就扣分，对了就奖励，由此训练得到在各个状态环境当中最好的决策。强化学习当中有几个重要的组成元素，包括前面说到的奖励 reward，可以认为是学习过程中的一个反馈；另外一个就是智能体 agent，是一个被抽象出来感知周围环境的单元，可以想想为一个小的机器人，在实际的应用当中可能是一个游戏玩家，一个棋手，一辆自动驾驶的汽车等等。智能体 agent 感知到的环境被称作的状态 state，智能体 Agent 试图通过一种策略决策来最大化 奖励，通过策略便会引起 agent 的行动动作 action。以上构成了 强化学习的基本要素。

值函数（value function）：强化学习是通过奖励或惩罚来学习怎样选择能产生最大积累奖励的行动的算法。为了找到最好的行动，非常有效的方式是，找到那些奖励最大的状态就好了，即在我们目前的环境（environment）中首先找到最有价值的状态 states。例如，在赛车跑道上最有价值的是终点线（这里好像就是你冲刺要达到 deadline 的前一步，这个状态肯定最有价值），这也是奖励最多的状态，因此在跑道之上的状态也比在跑道之外的状态更有价值。最有价值的 state 可能不只一个，一旦我们确定了哪些状态是最有价值的，我们就可以给这些状态赋奖励值。我们可以将这些状态和奖励当成一个离散函数，这些状态中，有些中间状态并不一定有奖励（奖励为 0），但是这些状态是通向奖励状态的必经之路。需要注意的一点是值函数求解的是一个 累积奖励，通过找到最大的值的状态，然后朝着这个方向前进，但是想想是有问题的：在一场的跑步比赛当中距离终点很远的一个状态只要朝着目标前进，它的奖励值也会很大。但是这显然不合理。因此，在值函数当中会引入一个折扣因子 discount Factor，由此使得离他较远的状态给他的奖励贡献是比较小的。因此，折扣因子是一个 0～1 之间的数，，并且随着距离的增加，这个值会越来越小。当折扣因子为 0.5 时，仅经过 3 次状态改变，奖励值就会变成初始值的八分之一，所以 agent 更倾向于搜索临近状态的奖励值。

值迭代算法： 首先把所有状态的奖励值初始化为 0 或者一个特定的值，然后搜索所有

状态的可能的下一个状态（从当前状态转移到其中的任意一个状态都有一个概率），并估计下一个状态 agent 可能得到的奖励，通过这种方式学习每一个状态特有的局部奖励值。如果 agent 在下一个状态得到奖励，那么这个奖励就会累计到当前的状态中。重复这个过程，直到每个状态的局部奖励值不再改变（其实这就是迭代的终止条件），意味着每次变换状态采取的可能转向以及每个状态的奖励值都被考虑在内。

策略函数（policy function）：值函数相当于是将所有的策略都计算在内求出来的一个期望额奖励值，而策略函数则是对不同的策略求解值函数（相当于值函数在某一策略的条件概率下求解），这样的做的目标很明确，就是我最终的结果是要求解到一个最好的策略，那为什么不在进行值函数计算的时候就分不同的策略求解勒？实际上上文最后的例子也是采用了贪婪的策略，就是每一个的策略都是朝着奖励值最大的方向去，但实际上这并不一定好。策略函数（policy function）是根据价值函数选择产生最大（长期）奖励的行动的一组策略。在所有可能的下一步行动中，通常没有明确的优胜者。例如，agent 面临选择下一步进入 4 个状态 A, B, C, D 中的一个，它们的奖励分别为 A=10, B=10, C=5, D=5。A 与 B 都是好的即刻选择，但是随着时间的推移，A 状态之后的路径得到的奖励可能比 B 状态好得多，或者进入 C 状态的行动甚至是最好的选择。所以在训练的过程中，探索所有的选择是值得的，但同时，如果只看到即刻奖励，就可能会导致非最优的选择。值得注意的是，策略函数和价值函数相互依赖。

Q-函数（Q-function）：我们已经看到策略和价值函数是高度相依的：我们的策略大多取决于我们看重什么，而我们看重什么决定了我们的行动。因此我们或许可以把策略和价值函数结合起来，这个结合就叫 Q-函数（Q-function）。其实就是状态－动作值函数。

同样用条件概率来理解它： 策略迭代的思想就是 在每一个状态下面分各个 策略来计算值函数，但是其实每一个策略同样对应了多种的动作 Action。Q 函数就是，那我直接计算采取某种动作的值函数就好了啊。也就是说：在某种的状态 s 下，计算在一种动作 a 的条件概率的情况下，该值函数的值。Q-函数考虑了当前的状态（如价值函数）和下一步行动（如策略函数），然后针对状态-行动组合，返回局部奖励值。在更复杂的情况下，Q-函数可能会结合更多状态来预测下一步状态。

Q-学习（Q-Learning）：为了训练 Q-函数我们将所有状态-行动组合的 Q-值初始化为零，并将状态奖励值设定为给定的值，作为状态的初始化值。由于 agent 只能看到下一个状态的 Q-值，因为初始时都为 0，因此起初 agent 并不知道如何获得奖励，agent 可能会探索很多状态直到发现一个奖励，学习 Q-函数是从结果（奖励）到开始（起始状态）进行的。

深度 Q-学习（Deep Q-Learning）：我们可以将当前驾驶员看到的视野——一张图像——输入到卷积神经网络（CNN），训练它预测下一个可能行动的奖励值。因为相似状态的图像也是相似的（许多左转弯看起来相似），它们也会导致相似的行动。例如。神经网络会生成许多左转弯，并且甚至在没有遇到过的左转弯做出适当的行动。正如一个通过很多物品的图像训练的卷积神经网络能够准确识别这些物品一样，一个通过很多相似左转弯训练的网络也能够对不同左转弯做出速度和位置的微调。

然而要成功地使用深度 Q 学习，我们不能简单地应用规则来训练之前所描述的 Q 函数。如果我们盲目地应用 Q 学习规则，那么网络将在进行左转弯时学习做好左转弯，但同时将开始忘记如何做好右转弯。这是因为所有神经网络的动作都使用相同的权重；调整左转弯的权重会使它们在其他情况中表现得更糟糕。解决方案是将所有输入图像和输出动作存储为「经验（experiences）」：即将状态、动作和奖励三者一起存储。运行了一段时间训练算法后，我们从迄今为止的所有经验中随机选择一个，并为神经网络权重创造一个均值更新，它能为每一个发生在那些经验期间的动作最大化 Q 值（奖励）。这样我们可以在同一时间教我们的神经网络左转和右转。由于在跑道上的较早期驾驶经验并不重要——因为它们源于我

们代理的经验不足、甚至是初学者的一个时期——我们只跟踪固定数量的过去经验并忽略休息期。这个过程被称为**经验回放（experience replay）**。

第二章 主要方法

主要有以下几个阶段：

Preprocessing：第一步预处理，原始输入是 210x160 分辨率的 128 色图像，这需要大量的计算资源，所以需要一些降维预处理。

首先，为了对每一帧编码，取每个像素前后帧的最大值。同时，有必要消除闪烁，特别是一些只出现在偶数帧或者只出现在奇数帧的目标；

第二，从 RGB 图像中提取 Y 通道也就是亮度值，并重新调节图像为 84x84。下面提到的算法 1 中ϕ函数就是处理 m 帧图像中然后送入到 Q-函数，这里 m=4，也可以选择 3 或 5.

模型架构：使用一个神经网络可以有好几种方式来参数化 Q。因为 Q 历史动作映射与他们的 Q 值是对应关系，在之前的方法中，历史和动作都曾被作为神经网络的输入。这种方法的主要缺点就是每一个前向传播中的都需要计算每一个动作的 Q 值，所需要的计算资源是随着动作的数目呈线性增长。改进之处就是为每一个可能的动作指定一个独立的输出单元，同时只把状态描述作为神经网络的输入。这样，输出就对应着输入状态中每个独立动作的 Q 预测值。这样做的好处是只在一次神经网路的前向传播中，计算出了所有可能动作的 Q 预测值。

图一显示的结构接收被ϕ函数预处理过的 84x84x4 图像作为输入，第一个隐层包含了 32 个 8x8 的卷积核以步长为 4 在输入图像上滑动，并紧跟着一个非线性函数来矫正输出。第二个隐层包含了 64 个 4x4 的卷积核，步长为 2，同样紧跟着一个非线性函数。第三层隐层 64 个 3x3 的卷积核，步长为 1，也跟着一个非线性函数。最后一个隐层是一个全连接层，紧跟着 512 个矫正单元。输出层是一个线性的全连接层，每一个单元后都表示一个有效的动作。动作的数量根据游戏的不同在 4-18 之间调节。

训练细节：在 Atari 2600 的 49 个游戏平台上对比个多个强化学习算法。针对所有游戏，使用相同的网络架构、学习算法、超参数设置和最少的先验知识，以证明算法的鲁棒性和泛化性能。在训练的过程中，如果想在相同游戏上评估智能体，我们只改动奖励机制。对于游戏之间分数的差异过大，我们规定积极奖励为 1，反之为-1,0 表示奖励不变。使用这种方法可以减轻惩罚的力度以在多种游戏中应用相同的学习率。但是同时，由于智能体不能区分不同惩罚机制的惩罚力度，这也会对智能体的学习有影响。对于那些带有生命计数器的游戏，Atari 2600 模拟器也会发送剩余的生命次数，以用来判断每轮循环是否结束。

在这些实验中，使用的是均方根反向传播(RMSProp)（*一种权值更新算法，类似于 SGD 算法，其中，RMSProp 是 RProp 算法的改良版。1. rmsprop 算法给每一个权值一个变量 MeanSquare(w,t) 用来记录第 t 次更新步长时前 t 次的梯度平方的平均值。2. 然后再用第 t 次的梯度除上前 t 次的梯度的平方的平均值，得到学习步长的更新比例。3. 根据此比例去得到新的学习步长。如果当前得到的梯度为负，那么学习步长就会减小一点点；如果当前得到的梯度为正，那么学习步长就会增大一点点。rmsprop 算法步长的更新更加缓和。这些算法并不能完全解决局部最小值问题，只是使得参数收敛的速度更快。*）最小批处理大小为 32。训练过程中，行为策略选择的是ε贪心算法，在前 1000 000 帧时，ε的大小从 1.0 到 0.1 线性衰减，在此之后就不再变化了。总共训练了 5 千万帧(总的花费了约 38 天)，而回放记忆的选择范围为最近的 1 百万帧。

针对帧数过大的问题，使用一个简单的帧跳跃技术(frame-skipping technique)，智能体每个 k 帧读取一次帧图像，而不是每一帧都读取。在跳过帧的时候，它的最后一个动作是重复

的。因为仿真器每运行一步的时间比智能体做出动作的时间少，帧跳跃技术可以让智能体在总时间没有增加的前提下多玩了 k 倍次数的游戏，实验中的 k=4。

在游戏 Pong, Breakout, Seaquest, Space Invaders and Beam Rider.中，所有超参数和优化参数的值都可以通过一个随机搜索的方法的得到。没有使用网格搜索是因为花费计算太大。这些参数在其他的游戏中也不会得到更改。

实验中使用最少的先验知识：游戏视频，游戏得分，动作的总数，和剩余生命值。

评估过程：对训练好的智能体进行评估，玩 30 次相同的游戏，每次五分钟，每把游戏的开局都是随机的。ε-贪心策略的系数为 0.05。这个过程是为了最小化评估过程产生过拟合。设置了一个基础对照组，每 6 帧产生 10HZ 的动作，再两帧之间重复所选择的动作。10Hz 是人类摁下“射击”按钮最快的速度。另外设置随机智能体作为对照也是防止人类操作游戏时分数造假。同时也评估了随机智能体以 60Hz 的速度相应动作。高速度的作用如下：在 Boxing, Breakout, Crazy Climber, Demon Attack, Krull and Robotank 六个游戏中，相比较 10Hz 速度的 DQN 只有 5%的提高，但是，在所有的游戏中，DQN 都远远超过了最好的人类选手。

人类测试者使用与智能体相同的游戏模拟器，在相同的控制条件下测试，也不允许使用暂停、保存、加载等功能。在原始的 Atari 2600 环境下，模拟器视频刷新率 60Hz，没有声音：同样的，人类与智能体使用相同是显示器。人类测试者每个游戏练习两个小时，测试的时候每把最多玩 5 分钟，每个游戏玩 20 次，取平均得分。

算法：在 Atari 模拟器的环境中，智能体通过动作序列、奖励、观测与环境进行交互。在每一个时间步里，智能体从合法的游戏动作集合中选取一个a_t. 这个动作被传输到模拟器中改变游戏状态并获取得分。通常情况下，这种选择都是随机的。智能体观测不到模拟器的游戏状态，只能观测到模拟器的游戏画面--像素矩阵。智能体还会收到奖励信号r_t来表示游戏得分的变化。需要指出的是，正常的游戏收益有时候取决于之前的动作和观察序列；关于某个动作的反馈或许会在几千步后才能表现出来。

由于智能体只能观察当前的屏幕显示，所以它并不能完全理解屏幕显示的内容。因此，动作和观察序列是算法的输入，智能体从这些输入中学习游戏策略。所以的序列都是有限的，这种做法提升了效果但是限制了马尔科夫决策过程（MDP），每一个序列都是一个清晰的状态。所以，我们利用时间 t 时刻的状态描述序列 s_t 对 MDPs 使用了标准的强化学习方法。

智能体的目标是通过与模拟器的交互并选择相应的动作使得预期奖励最大化。我们对于预期奖励做了一个衰减假设，衰减系数γ恒为 0.99，定义时间 t 时刻的预期奖励为$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ 其中 T 表示游戏中时间步，我们定义优化动作值函数 Q*(s,a)作为任意一个策略能达到的最大期望。最优动作值函数满足 Bellman 方程，这是基于以下的直觉：如果序列s'在下一个时间步的最优值 Q*(s,a)是针对所有可能的动作a'，那么最优策略的选择遵循如下公式：

$$Q*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q*(s',a') D_{s,a}\right]$$

很多强化学习算法的都是用 bellman 方程来更新动作值函数的预测值。实际上，这个基础的算法并不可行，因为动作值函数是被分开估计的，不能通用。所以，需要一个近似函数来化简动作值函数$Q^*(s,a;\theta) \approx Q^*(s,a)$，在之前，经常使用线性近似的方法，有时候也使用非线性函数近似。使用神经网络作为 Q 网络会更好，因为神经网络可以自动调整权重以减少 Bellman 方程的均方误差。将损失转化成一个可优化的目标函数，这个目标值取决于网络的权重，相比较监督学习，监督学习在训练开始之前目标已经固定好了。在优化的每一个阶段，当优化第 i 次的损失 $L_i(\theta_i)$时，固定上一迭代的参数值不变，这样就转化成了一些有解决方法的优化问题。如果对损失进行微分，那么可以使用带有梯度优化的策略。特别是使用随机梯度下降算法是非常有效的，而常见的 Q-learning 算法也可以应用到这个框架中来更新

每一步的权重。

需要指出的是上面提到的算法不受模型限制：它直接使用模拟器采样解决了强化学习任务，并没有直接估计奖励和动态规划。算法通过贪心策略$a = argmax_{a'}Q(s,a';\theta)$，以确保能够探索到足够状态空间。实际上，行为分布常常用ε贪心策略来选择，表示贪心策略的概率是 1-ε，和以ε的概率随机选择一个动作。

深度 Q 网络的训练算法：智能体使用基于 Q 的贪心策略选择动作，这是因为很难使用任意长度的历史数据作为神经网络的输入，所以 Q 函数就固定了这个长度。算法通过两种方法改进了标准的 Q-learning 是的训练神经网络收敛。

第一，使用经验回溯法，之前已经将智能体的每一个时间步的经验存储到数据集中了，在算法开始迭代的时候，采用 Q-learning 或者 minibatch 更新，对存储的历史经验进行随机采样。这个方法相比标准的在线 Q-learning 有很多好处。第一，每一步的经验都有可能被用到很多；次来更新权重，更高效的利用了数据；第二，由于采样信号之间的耦合性很大，所以从连续的采样信号中直接学习是低效的，随机采样可以打破这种耦合性从而降低了更新的方差；第三，训练当前参数时可以确定下一步训练要使用的数据。比如：如果最大化奖励的动作是让角色左移，那么训练采样将会倾向从左手边采样，同理，右边亦然。显然，这里的反馈循环是有害的，或许卡在局部最小值甚至发散不收敛。采用了经验回溯的方法之后，可以平均掉许多之前的状态，从而平滑训练过程已达到收敛。事实上，论文中的算法只是存储了最近 N 步的历史经验用来回溯，数据集采样服从均匀分布。当然这样做也有一些弊端：内存不是无限的，所以新的经验会覆盖掉旧的，但是智能体不能区别那些经验是重要的，那些是不重要的。同样的，均分分布采样的时候，每一个历史经验被采样的概率都是相等的，没有重要性的差别。可以使用 prioritized sweeping(优先擦除)的方法改进采样策略。

第二个修改的地方是分离 Q-learning 过程中目标y_j的生成，每 C 步更新的时候，克隆一个$\hat{Q}$来生成目标y_j，这样，对于所有的 a 增大，$Q(s_t,a)$ $Q(s_{t+1},a)$也会增大，这个改进使得算法更稳定。同时，使用旧的参数生成目标y_j增大了更新 Q 的延迟从而影响了目标y_j的更新，进一步降低了算法发散的可能性。

实现细节：

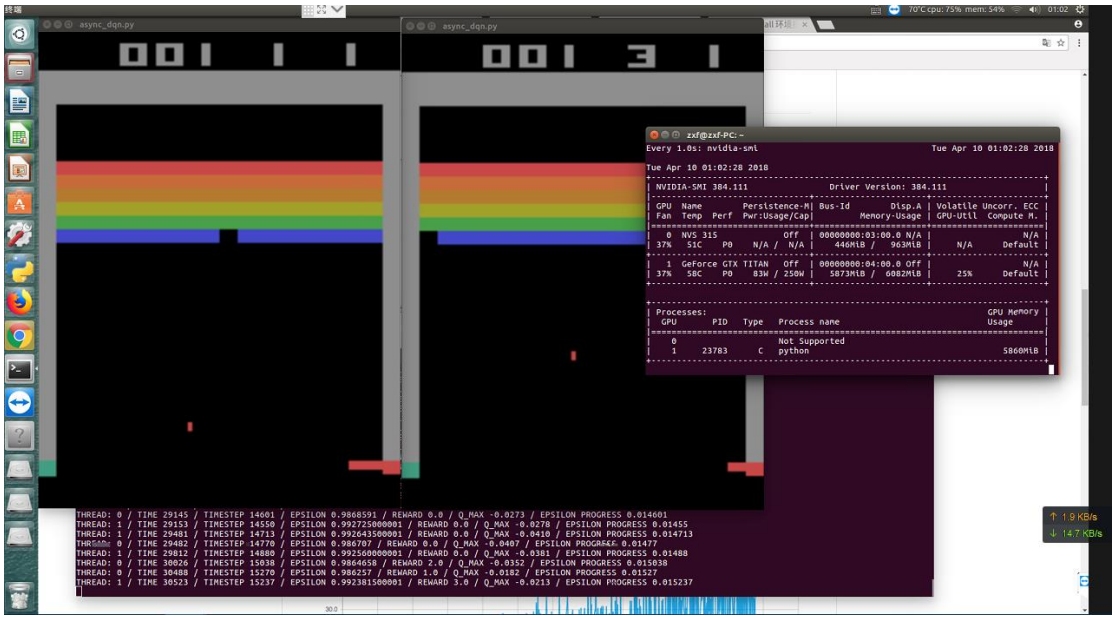
首先，每一个 episode 从随机数量（0 到 30 之间）的「No-op」低级别 Atari 动作开始（相对于将智能体的动作（action）重复 4 个帧），以抵消智能体所看见的帧，这是因为智能体每次只能看到 4 个 Atari 帧。类似地，用作 CNN 输入的 m 个帧历史是智能体最后看见的 m 个帧，而不是最后的 m 个 Atari 帧。此外，在使用梯度下降迭代之前，我们会执行 50000 步的随机策略作为补充经验以避免对早期经验的过拟合。

另一个值得注意的参数是网络更新频率（network update frequency）。原始的 DQN 实现仅在算法的每 4 个环境步骤后执行一个梯度下降步骤，这和算法 1 截然不同（每一个环境步骤执行一个梯度下降步骤）。这不仅仅大大加快了训练速度（由于网络学习步骤的计算量比前向传播大得多），还使得经验内存更加相似于当前策略的状态分布（由于训练步骤之间需要添加 4 个新的帧到内存中，这和添加 1 个帧是截然不同的），可能有防止过拟合的作用。

绝大多数 Atari 游戏中，玩家都有几条「命」，对应游戏结束之前玩家可以失败的次数。为了提升表现，Mnih et al. [2015] 选择在训练中把生命数的损失（在涉及生命数的游戏中）作为 MDP 的最终状态。这一终止条件在 DQN 论文中没有提及太多，但却对提升性能至关重要,MDP 中的最终状态意味着智能体无法再获得更多奖励。

简单的 demo 演示

我在论文中看到作者给出的源码，但是使用 torch 和 lua 语言编写的，手中的电脑没有这样的运行环境，所以找了一个 python 的例子运行了一下 demo，初次运行界面如图五。在训练一个晚上时间之后，智能体可以获得十几得分（运行演示见附件的 GIF）。



图五. 初次运行 async-rl

参考资料

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529.

[2] https://github.com/mgbellemare/Arcade-Learning-Environment

[3] http://www.datakit.cn/blog/2017/02/05/t_sne_full.html

[4] https://blog.csdn.net/ikerpeng/article/details/52604506

[5] https://www.jiqizhixin.com/articles/2017-11-24-4

[6] https://github.com/coreylynch/async-rl