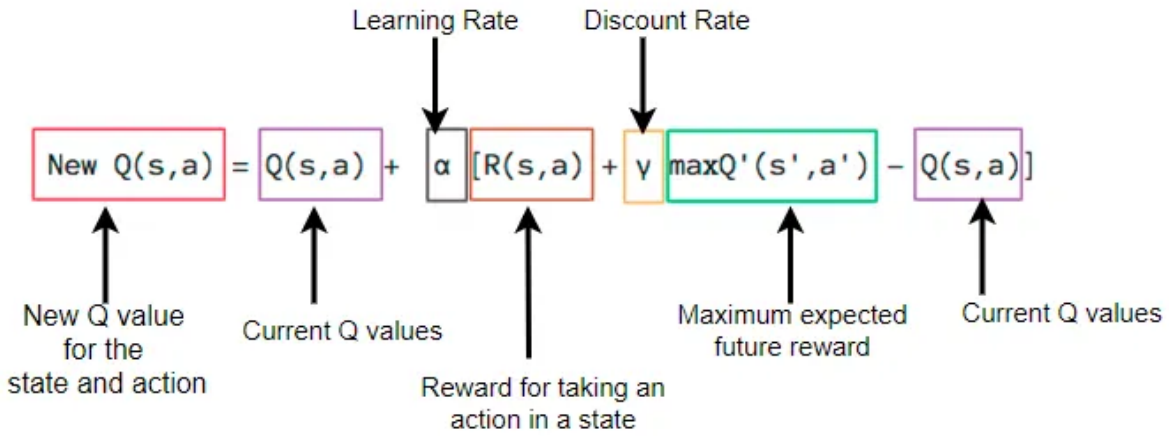
ملوین مختاری ۹۸۳۱۱۴۳ – پروژه دوم درس هوش مصنوعی

الگوریتم کیولرنینگ یک الگوریتم یادگیری تقویتی بدون مدل است که مقدار عمل را در یک وضعیت خاص یاد می‌گیرد. هدف این الگوریتم این است که یک سیاست بهینه برای انتخاب عمل در هر وضعیت پیدا کند. برای این کار، از یک جدول Q استفاده می‌کند که مقادیر Q را برای هر جفت وضعیت و عمل ذخیره می‌کند. مقادیر Qنشان‌دهنده ارزش عمل در یک وضعیت هستند، یعنی انتظار پاداش آینده را با توجه به عمل انجام شده در یک وضعیت نشان می‌دهند.

الگوریتم کار خود را با جدول Q تصادفی شروع می‌کند و سپس با تجربه و تکرار، جدول را به روز می‌کند. فرمول به روزرسانی جدول Q به شکل زیر است:



این فرمول با استفاده از تفاضل بین مقدار فعلی Q و مقدار مورد انتظارQ ، جدول را به سمت حالات بهینه حرکت می‌دهد.

به عنوان مثال، فرض کنید یک ربات داریم که می‌خواهیم به آن یاد بدهیم چطور بازی کند. ربات می‌تواند در هر لحظه چند کار مختلف انجام بدهد، مثلاً بپرد، بدود و ... . هر کاری که ربات انجام می‌دهد، باعث می‌شود وضعیت ربات عوض شود. مثلاً اگر ربات بپرد، وضعیتش از زمین به هوا تغییر می‌کند. همچنین هر کاری که ربات انجام می‌دهد، یک پاداش یا جایزه هم دریافت می‌کند. مثلاً اگر ربات یک سکه جمع کند، پاداش مثبت می‌گیرد، ولی اگر به دشمن برخورد کند، پاداش منفی می‌گیرد.

ملوین مختاری ۹۸۳۱۱۴۳ – پروژه دوم درس هوش مصنوعی

حالا ما می‌خواهیم به ربات یاد بدهیم که چطور بازی کند که بیشترین پاداش را بگیرد. چطور این کار را انجام بدهیم؟

اینجاست که الگوریتم کیولرنینگ به کمک ما می‌آید. این الگوریتم یک جدول دارد که داخل آن نوشته شده هر وضعیت و هر عمل چقدر خوب هست. این جدول را Q می‌گویند. اول این جدول خالی یا تصادفی هست و ربات هر عمل تصادفی رو انجام می‌دهد. ولی بعد از هر عمل، ربات جدول Q رو با توجه به پاداش و وضعیت جدید به روز می‌کند. به این شکل، ربات چندین بار بازی را تکرار می‌کند و جدول Q را بهتر و بهتر می‌کند. در نهایت، جدول Q نشان می‌دهد که در هر وضعیت، کدام عمل بهترین پاداش آینده را دارد. پس ربات با دیدن جدول Qمی‌تواند بهترین عمل را انتخاب کند.این جدول شامل اعداد مختلف است که هر عدد نشان می‌دهد که یک کار در یک وضعیت چقدر خوب است. هر چه عدد بزرگ‌تر باشد، یعنی کار بهتر است و پاداش بزرگتری دارد. هر چه عدد کوچک‌تر باشد، یعنی کار بدتر است و پاداش کوچک‌تر یا منفی دارد.

برای مثال، فرض کنید ربات ما در یک بازی ماشین سواری است. ربات می‌تواند در هر لحظه سه تا کار انجام بدهد: راست برود، چپ برود یا مستقیم بماند. این سه تا کار را با اعداد ۰، ۱ و ۲ نشان می‌دهیم. همچنین ربات در هر لحظه در یک وضعیت است. وضعیت ربات نشان می‌دهد که ماشین ربات در کجای جاده قرار دارد. این وضعیت‌ها را با حروفA ، B و C نشان می‌دهیم. جدول Q ربات به شکل زیر است:

State	Action 0	Action 1	Action 2
A	-10	-5	5
B	-5	-10	5
C	-10	-10	-10

این جدول به ربات می‌گوید که در هر وضعیت، کدام عمل بهترین است. برای مثال، در وضعیتA ، عمل ۲ (مستقیم ماندن) بهترین عمل است و عمل ۰ (به راست رفتن) بدترین عمل است. عمل ۲ بهترین عمل است چون بیشترین عدد را در جدول Q دارد. این یعنی اگر ربات این عمل را انجام دهد، بیشترین پاداش آینده را می‌گیرد. پاداش آینده ممکن است شامل پاداش فوری (مثلاً گرفتن سکه) یا پاداش غیرفوری (مثلاً رسیدن به خط پایان) باشد. ربات همیشه سعی می‌کند که بیشترین پاداش آینده را بگیرد، چون هدف اصلی ربات این است.

ملوین مختاری ۹۸۳۱۱۴۳ – پروژه دوم درس هوش مصنوعی

deep q learningیک روش یادگیری تقویتی است که از یک شبکه عصبی عمیق برای تخمین مقادیر Q استفاده می‌کند. این روش در سال ۲۰۱۳ توسط تیم DeepMind معرفی شد و توانست در چندین بازی آتاری عملکرد بهتری از روش‌های قبلی داشته باشد.

deep q learning از q learning الهام گرفته شده است، اما به جای استفاده از یک جدول Q برای ذخیره مقادیرQ ، از یک شبکه عصبی عمیق به نام Q-network استفاده می‌کند Q-network یک تابع تخمین‌گر است که ورودی آن وضعیت فعلی ربات و خروجی آن مقادیر Q برای هر عمل ممکن است.

deep q learning چندین مزیت نسبت به q learning دارد. اول این که نیاز به حافظه کمتری دارد، چون نمی‌خواهد همه وضعیت‌ها و عمل‌ها را در یک جدول ذخیره کند. دوم این که می‌تواند با وضعیت‌های پیچیده و پیوسته کار کند، چون شبکه عصبی می‌تواند وضعیت‌های مشابه را تشخیص دهد و از تجربه قبلی استفاده کند. سوم این که می‌تواند از چندین تکنیک بهبودبخشیدن به الگوریتم استفاده کند، مثلexperience replay، target network ، double q learning و dueling q network .

الگوریتم Epsilon Decay یک روش برای کاهش مقدار Epsilon در الگوریتم Deep Q-Networks (DQN) است. این الگوریتم در ابتدا یک مقدار ثابت برای Epsilon تعیین می‌کند که به احتمال Epsilon عملیات انتخاب عمل تصادفی را انجام می‌دهد (یا به‌اصطلاح Explore می‌کند). سپس با گذشت زمان، مقدار Epsilon به صورت نمایی کاهش پیدا می‌کند (یا به اصطلاح Exploit می‌کند) تا در نهایت به یک حداقل مشخص برسد. این روش به دلیل کاهش تصادفی بودن عمل‌های انجام شده، باعث بهبود عملکرد الگوریتم DQNمی‌شود.

در الگوریتمDQN ، یک حافظه به نام "experience replay" برای ذخیره کردن تجربیاتِ جمع آوری شده توسط عامل استفاده می‌شود. این حافظه به عنوان یک مکانیزم ذخیره‌سازی برای تجربیاتِ جمع آوری شده عمل می‌کند و با استفاده از نمونه‌برداری تصادفی، تجربیاتِ جمع آوری شده را مجدداً استفاده می‌کند. این کار باعث بهبود و پایدار شدن فرآیند آموزش مدل می‌شود.

ملوین مختاری ۹۸۳۱۱۴۳ – پروژه دوم درس هوش مصنوعی

بسیار خوب، بپردازیم به توضیحات خود کد:

```
!pip install stable-baselines3[extra]
!pip install gymnasium
```

این بلوک کد در کولب یا کگل دو کتابخانه stable baselines۳ و gymnasium را نصب می‌کند.

کتابخانه اول شامل الگوریتم‌های یادگیری تقویتی و کتابخانه دوم شامل شبیه سازی محیط بازی هستند.

```
import os
from stable_baselines3.common.callbacks import BaseCallback
from stable_baselines3.common import env_checker
from stable_baselines3.common.vec_env import VecFrameStack, DummyVecEnv
from stable_baselines3.common.evaluation import evaluate_policy
from stable_baselines3.common.env_util import make_atari_env
from stable_baselines3.common.monitor import Monitor
from stable_baselines3 import DQN
import gymnasium as gym
```

در خطوط بالا توابع و کلاس‌های مورد نیاز در کد افزوده شدند که در ادامه از آنها استفاده شده و هرکدام توضیح داده خواهد شد.

```
class TrainAndLoggingCallback(BaseCallback):
    def __init__(self, check_freq, save_path, verbose=1):
        super(TrainAndLoggingCallback, self).__init__(verbose)
        self.check_freq = check_freq
        self.save_path = save_path

    def _init_callback(self):

        if self.save_path is not None:
            os.makedirs(self.save_path, exist_ok=True)

    def _on_step(self):
        if self.n_calls % self.check_freq == 0:
            model_path = os.path.join(self.save_path, f'best_model_{self.n_calls}')
            self.model.save(model_path)
        return True
```

در کد بالا کلاسی تعریف کردیم که از کلاس BaseCallback ارث بری می‌کند و از آن برای ذخیره کردن مدل در بازه‌های مختلف زمانی برحسب n_calls که همان شماره step یا frame بازی هست، استفاده کردیم، یعنی اگر check_frame را برابر ۱۰۰ قرار دهیم در فریم‌های ۱۰۰، ۲۰۰، ۳۰۰ و ... مدل را در آدرس save_path با فرمت best_model_XXXX.zip ذخیره می‌کند که XXXX شماره step در زمان ذخیره مدل می‌باشد. (این کار را تابع _on_step انجام می‌دهد و تابع _init_callback درصورت نبودن دایرکتوری save_path آن را می‌سازد)

```
CHECKPOINT_DIR = './train/'
LOG_DIR = './logs/'
window = 4
env = make_atari_env("ALE/Breakout-v5", n_envs=1, monitor_dir=LOG_DIR)
# Frame-stacking with 4 frames
vec_env = VecFrameStack(env, n_stack=window)
```

در کد بالا در CHECKPOINT_DIR مسیر ذخیره مدل‌ها و در LOGS_DIR مسیر ذخیره شدن لاگ‌ها نوشته شده است.

متغیر window میزان تکرار تصاویر برای تشخیص مسیر حرکت نشان داده شده است.

تابع make_atari_env یک پیش پردازش خوب برای بازی‌های این چنینی می‌باشد که سایز محیط بازی را به 84x84 تغییر داده و ان را برای ساده شدن محاسبات سیاه سفید یا همان gray می‌کند.

تابع VecFrameStack برای تشخیص مسیر حرکت به تعداد window فریم‌های پشت سر هم را کنار هم می‌نشاند.

```
callback = TrainAndLoggingCallback(check_freq=100000, save_path=CHECKPOINT_DIR)
newmodel = DQN('CnnPolicy', vec_env, tensorboard_log=LOG_DIR, verbose=1,
            buffer_size=50000, batch_size=64, learning_starts=10000, gamma=0.95,
            exploration_fraction = 0.5, exploration_final_eps=0.1)
```

از کلاس TrainAndLogginCallback که پیش‌تر توضیح دادیم استفاده کردیم و یک callback با دوره ۱۰۰۰۰۰و مسیر CHECKPOINT_DIR ساختیم.

ملوین مختاری ۹۸۳۱۱۴۳ – پروژه دوم درس هوش مصنوعی

با کلاس DQN یک مدل با ساختار CNN و محیط vec_env و مسیر لاگ LOGS_DIR و سایز حافظه ۵۰۰۰۰و سایز بچ 64 و ضریب گاما ۰.۹۵ و حداقل ضریب explore ۰.۱ با نرخ نزول ۰.۵ ساختیم که از فریم ۱۰۰۰۰ به بعد شروع به یاد گیری می‌کند و تا ۱۰۰۰۰ کاملاً رندوم حرکت می‌کند (به اصطلاح گرم می‌کند).

```
newmodel.learn(total_timesteps=1000000, callback=callback, log_interval=1000)
```

با کد فوق مدل شروع به یاد گیری می‌کند و ۱۰۰۰۰۰۰ فریم یادگیری را ادامه می‌دهد و هر ۱۰۰۰ episode لاگ می‌اندازد.

```
!pip install gdown
!gdown --id 1JeVp1J1pasmlmgVrHI2uyK_6sUZT5Gdc
```

این کد کتابخانه gdown را نصب و با استفاده از آن کد آپلود شده در google drive را دانلود می‌کند.

```
model = DQN.load("/kaggle/working/best_model_1000000.zip", print_system_info=True)
window = 4
```

کد بالا مدل را از فایل دانلود شده در کگل بارگیری می‌کند.

```
import imageio
import numpy as np

env = make_atari_env("ALE/Breakout-v5", n_envs=1)
window=4
# Frame-stacking with 4 frames
vec_env = VecFrameStack(env, n_stack=window)
model.set_env(vec_env)
imgs = []
episodes = 1
for episode in range(1, episodes+1):
    obs = vec_env.reset()
    dones = [False]
    score = 0

    while True:
        action, _states = model.predict(obs, deterministic=False)
        obs, rewards, dones, info = vec_env.step(action)
#         print(info[0]['lives'])
        score += rewards[0]
        vec_env.render("rgb_array")
        imgs.append(model.env.render(mode='rgb_array'))
        if info[0]['lives'] == 0 and dones[0]:
            break

    print('Episode:{} Score:{}'.format(episode, score))
vec_env.close()
del vec_env

imageio.mimsave('breakout.gif', [np.array(img) for i, img in enumerate(imgs)])
```

در کد بالا پس از تعریف دوباره محیط برای یک episode (بازی کامل تا باخت) با استفاده از مدل بازی می‌کند.

در خط ۱۲ محیط را reset می‌کند تا بازی از اول شروع شود.

در خط ۱۶ با دادن obs یا همان تصویر محیط بازی که همان state است به مدل action یا عمل مناسب را می‌گیریم و در خط بعدی بعد محیط داده و نتایج مانند برد و باخت یا امتیاز کسب شده را به ما می‌دهد.

امتیاز در خط ۱۸ در متغیر score جمع شده است و این نکته هم دقت داشته باشید که این امتیاز درج شده در بالای تصویر بازی نیست چون آجرهای ردیف‌های بالا از دید بازی، امتیاز بیشتری دارند و score فقط تعداد آجری هست در بازی که با ضربه توپ، خراب شده است.

در خط ۲۰ تصویری از محیط بازی می‌گیریم و در لیست imgs ذخیره می‌کنیم تا در خط آخر با استفاده از آن‌ها انیمیشن بازی انجام شده را در فایل breakout.gif ذخیره کنیم در خط ۲۱ هم چک می‌شود که آیا بازی تمام شده یا خیر.

در خط ۲۴ میزان امتیاز نهایی چاپ می‌شود و در دو خط ۲۵ و ۲۶ هم محیط بسته می‌شود تا با کد دیگری تداخل نکند.

```
from IPython.display import Image, display
display(Image(data=open('/kaggle/working/breakout.gif','rb').read(), format='gif'))
```

در این کد فایل گیف نمایش داده می‌شود.

```
env = make_atari_env("ALE/Breakout-v5", n_envs=1)
window=4
# Frame-stacking with 4 frames
vec_env = VecFrameStack(env, n_stack=window)
rew_mean, rew_std = evaluate_policy(model, vec_env, n_eval_episodes=100, render=True)
```

در خط آخر این کد با تابع evaluate_policy ۱۰۰ بار بازی می‌کند و میانگین و انحراف معیار امتیازهای کسب شده را می‌دهد که به ترتیب در rew_mean و rew_std ذخیره می‌شود که تقریباً ۲۹.۹۲ و ۱۰.۰۲۸۶ بودند.