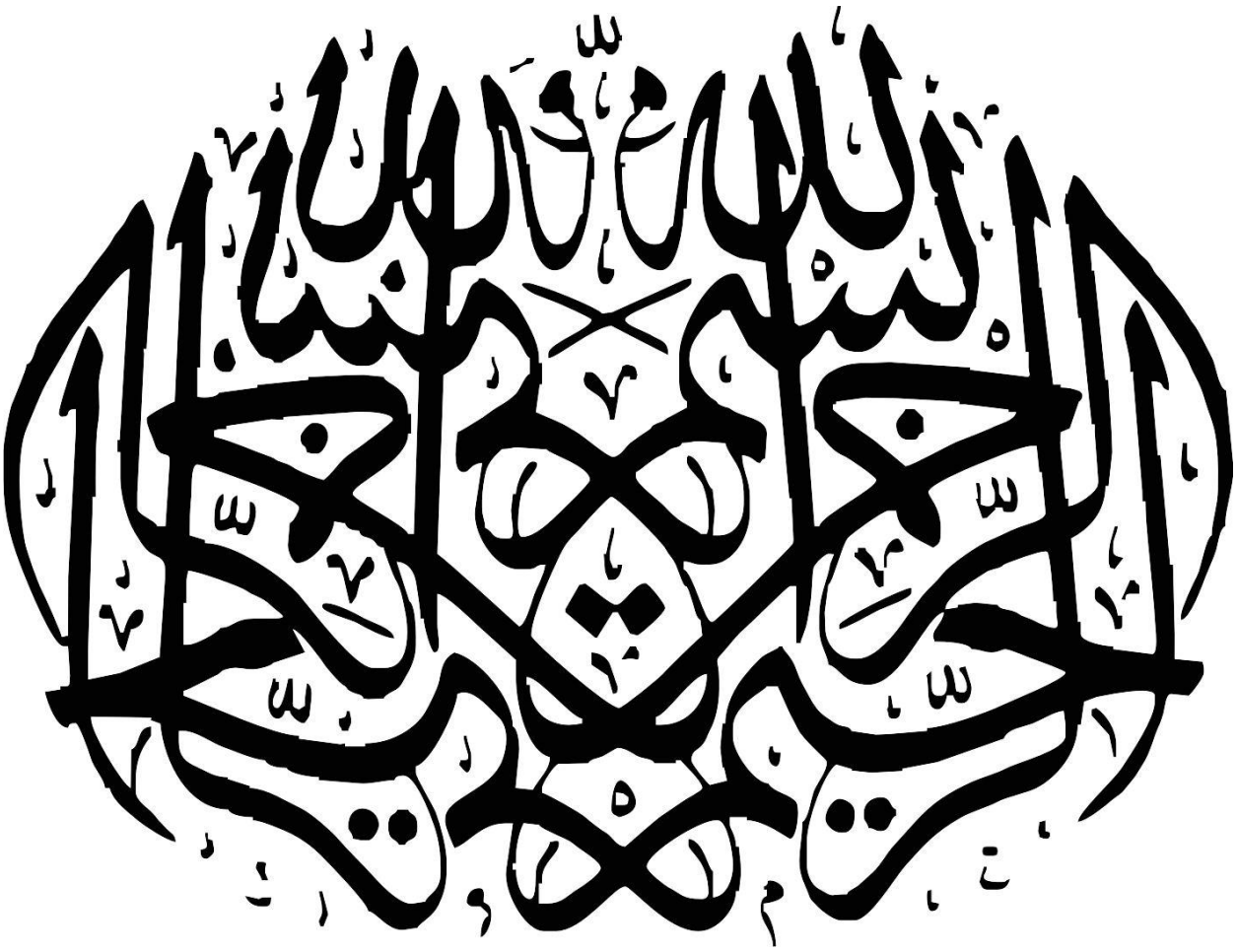
دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

تخصیص کارها به ماشین‌ها در مسئله زمانبندی کارگاهی با استفاده از یادگیری تقویتی عمیق

استاد درس: دکتر مصدق

تهیه کننده: سید نیما محمودیان

شماره دانشجویی: ۴۰۲۱۲۵۰۰۵



فهرست مطالب

فهرست شکل‌ها

فهرست جدول ها

۱-چکیده

در این گزارش به ارائه‌ی کد یک مسئله زمان‌بندی کارگاهی پرداخته می‌شود که با استفاده از شبکه‌ی عصبی گرافی، مسئله را به شکل یک شبکه عصبی مدل می‌کند و با استفاده از یک روش بازیگر-منتقد به نام Proximal Policy Optimization برای آموزش شبکه عصبی استفاده می‌شود. در ادامه به بیان مسئله و نحوه‌ی کد کردن مسئله پرداخته می‌شود.

۲-مقدمه

مسئله زمانبندی کارگاهی (JSSP) یک مسئله بهینه سازی ترکیبی شناخته شده در علوم کامپیوتر و تحقیق در عملیات است و در بسیاری از صنایع مانند تولید و حمل و نقل حاضر است. در JSSP، چندین کار با محدودیت‌های پردازش از پیش تعریف‌شده (مثلاً عملیات‌ها به ترتیب توسط ماشین‌های واجد شرایط پردازش می‌شوند) به مجموعه‌ای از ماشین‌های ناهمگن اختصاص داده می‌شوند تا به هدف مورد نظر مانند به حداقل رساندن زمان ساخت، یا تأخیر دست یابند.

به دلیل NP-hard بودن مسائل JSSP یافتن راه‌حل‌های دقیق برای JSSP اغلب غیرعملی است. بنابراین روش‌های ابتکاری یا روش‌های تقریبی در حل این دسته از مسائل کاراتر هستند. قانون توزیع اولویت (PDR) یک روش ابتکاری است که به طور گسترده در عمل استفاده می‌شود.

در مقایسه با روش‌های بهینه‌سازی پیچیده مانند برنامه‌ریزی ریاضی و الگوریتم‌های فراابتکاری، PDR از نظر محاسباتی سریع است، پیاده‌سازی بصری آن آسان است و به طور طبیعی قادر به مدیریت عدم قطعیت‌هایی است که در عمل همه‌جا وجود دارند. با توجه به این مزایا، تعداد زیادی PDR برای JSSP پیشنهاد شده است. با این حال، معمولاً پذیرفته شده است که طراحی یک PDR مؤثر بسیار پرهزینه و زمان‌بر است و به ویژه برای JSSP پیچیده به دانش قابل توجه و آزمون و خطا، نیاز دارد. علاوه بر این، عملکرد یک PDR اغلب در موارد مختلف به شدت متفاوت است.

بنابراین، سوالی که پیش می‌آید این است که آیا می‌توانیم فرآیند طراحی PDR را خودکار کنیم، به طوری که در گروهی از نمونه‌های JSSP با ویژگی‌های مشترک به خوبی عمل کند؟ در این مقاله، نویسندگان برای حل JSSP از رویکرد زیر استفاده کرده‌اند:

فرمولاسیون MDP: آنها ابتدا مسئله زمانبندی مبتنی بر PDR را به عنوان فرآیند تصمیم گیری مارکوف (MDP) فرموله کردند. این شامل تعریف نمایش حالت بود، که آنها با استفاده از نمایش نمودار منفصل JSSP به آن دست یافتند. این نمایش به طور موثر وابستگی های عملیات و وضعیت ماشین را نشان می دهد و اطلاعات ارزشمندی را برای تصمیم گیری های زمان بندی ارائه می دهد.

شبکه عصبی گراف (GNN) برای نمایش حالت: برای رمزگذاری مؤثر گره‌ها در نمودارهای منفصل، آنها یک طرح مبتنی بر GNN با یک استراتژی محاسباتی کارآمد پیشنهاد کردند. این طرح گره ها را به جاسازی هایی با ابعاد ثابت تبدیل می کند و امکان دستکاری و پردازش کارآمد را فراهم می کند.

Size-Agnostic Policy Network: آنها یک شبکه عصبی مستقل از اندازه طراحی کردند که می تواند نمونه های JSSP با اندازه دلخواه را مدیریت کند. این شبکه را می توان در نمونه های کوچکتر آموزش داد و سپس به مسائل در مقیاس بزرگتر تعمیم داد و یادگیری کارآمد را بدون محدود شدن به اندازه های خاص مشکل ممکن می سازد.

الگوریتم policy gradient برای آموزش: آنها از یک الگوریتم policy gradient برای آموزش شبکه و به دست آوردن PDR های با کیفیت بالا استفاده کردند. این الگوریتم با تشویق اقداماتی که منجر به نتایج زمان‌بندی بهتر می‌شود، سیاست‌های اتخاذ شده توسط شبکه عصبی را بهینه می‌کند.

۳-شرح مسئله

برای استفاده از یادگیری تقویتی، احتیاج است که چند مورد را تعریف و بر اساس آن مسئله را مدل کرد. این سه مورد شامل تعریف مسئله بر اساس فرایند تصمیم گیری مارکوف، تعریف سیاست لازم برای تصمیم گیری و الگوریتم یادگیری است.

۳-۱-فرایند تصمیم گیری مارکوف

نویسندگان از فرمول‌بندی فرآیند تصمیم‌گیری مارکوف (MDP) برای حل مسئله JSSP استفاده کرده‌اند . MDP یک نمایش حالت (state)، فضای عمل (action)، تابع انتقال حالت (state transition function) و تابع پاداش (reward function) را تعریف می‌کند. در ادامه به توضیح هر یک از این موارد می‌پردازیم.

۳-۱-۱-نمایش حالت

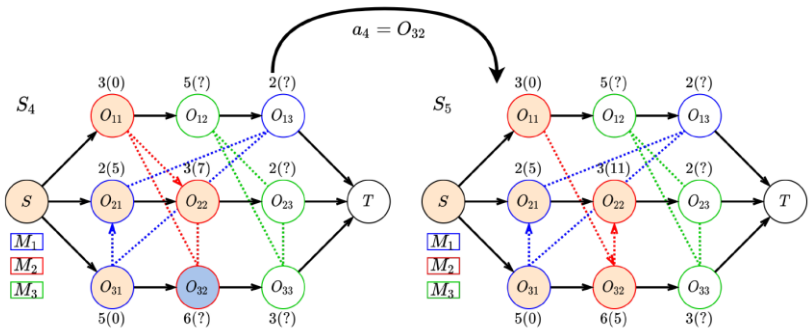
حالت MDP یک گراف منفصل است که وضعیت فعلی راه حل را نشان می‌دهد. این گراف شامل گره‌هایی برای کارها و ماشین‌ها و کمان‌هایی برای محدودیت‌های اولویت است. حالت، همچنین شامل دو ویژگی برای هر گره است: $I(O;s_t)$ یک نشانگر باینری که اگر انجام کار O در s_t برنامه ریزی شده ۱ است و در غیر این صورت ۰ است. و $C_{LB}(O;s_t)$ حد پایینی زمان تخمینی تکمیل (ETC) گره مذکور.

۳-۱-۲-فضای عمل

فضای عمل $a_t \in A_t$ در مرحله تصمیم t مجموعه ای از عملیات واجد شرایط است که می تواند در مرحله بعدی برنامه ریزی شود. حداکثر اندازه فضای عمل برابر با تعداد کارها است. همچنین توجه شود که برای هر کار حداکثر یک عملیات را می‌توان برای انجام در نظر گرفت.

۳-۱-۳-تابع انتقال حالت

هنگامی که یک عملیات در برای برنامه ریزی انتخاب می شود، اقدامات زیر انجام می شود: اولین دوره زمانی ممکن را برای تخصیص در ماشین مورد نیاز پیدا می‌شود. جهت کمان های منفصل آن ماشین بر اساس روابط زمانی فعلی به روز می‌شود. یک مجموعه گراف منفصل جدید به عنوان حالت s_{t+1} ایجاد خواهد شد. شکل ۱ انتخاب یک تصمیم و انتقال حالت از s_4 به s_5 را نمایش می‌دهد.



شکل ۳: انتخاب یک تصمیم و انتقال حالت از s_4 به s_5

۳-۱-۴-تابع پاداش

تابع پاداش $R(s_t;a_t)$ تفاوت کیفیت بین راه حل های جزئی مربوط به دو حالت s_t و s_{t+1} است. پاداش به صورت مقدار منفی کران پایین

C_{max} خواهد بود.

۱-۳-۵-سیاست

برای حالت s_t یک سیاست $\pi(a_t|s_T)$ نشان دهنده یک توزیع احتمالی بر روی فضای عمل A_t است. اگر از سیاست‌های سنتی PDR استفاده کنیم، کار با بالاترین اولویت دارای احتمال ۱ و سایر کارها دارای اولویت صفر خواهند بود. اما در این مسئله، انتخاب سیاست مناسب در هر s_t به عهده GNN که بر روی ده هزار نمونه آموزش دیده شده است گذاشته می‌شود. در ادامه درمورد پارامتری کردن سیاست توضیحات بیشتری ارائه خواهد شد.

۳-۲-پارامتری کردن سیاست

۳-۲-۱-استفاده از شبکه عصبی مبتنی بر گراف

نویسندگان از شبکه‌های عصبی گراف (GNN) برای جاسازی (embedding) گراف منفصل مسئله زمان‌بندی کارگاه (JSSP) در یک بردار با ابعاد ثابت استفاده می‌کنند. GNN ها نوعی شبکه عصبی هستند که می توانند نمایش داده های ساختاریافته با استفاده از گراف را بیاموزند، که برای JSSP مناسب است زیرا گراف جداکننده وابستگی های بین عملیات و ماشین ها را نشان می دهد.

نویسندگان از شبکه ایزومورفیسم گراف (GIN) به عنوان معماری GNN خود استفاده می کنند. GIN با در نظر گرفتن ویژگی‌های گره و همسایه‌های آن، مجموعه‌ای از به‌روزرسانی‌ها را روی جاسازی‌های گره در نمودار انجام می‌دهد. این فرآیند به GIN اجازه می دهد تا نمایشی از گراف را بیاموزد که روابط مهم بین گره ها را به تصویر می کشد.

نویسندگان از دو استراتژی مختلف برای جاسازی گراف منفصل در یک GNN استفاده می کنند:

استراتژی حذف کمان: در این استراتژی، نویسندگان هر کمان بدون جهت در نمودار را با دو کمان جهت دار، یکی در هر جهت، جایگزین می‌کنند. این کار منجر به یک نمودار کاملاً جهت دار می شود که می تواند توسط یک GNN پردازش شود. با این حال، این استراتژی می‌تواند منجر به نمودارهای بسیار متراکم شود که پردازش آنها از نظر محاسباتی هزینه بر است.

استراتژی افزودن کمان: در این استراتژی، نویسندگان فقط کمان‌های جهت‌دار را اضافه می‌کنند که در وضعیت فعلی زمان‌بندی، جهتی به آنها اختصاص داده شده است. این کار منجر به نمودارهای بسیار پراکنده‌تر می‌شود که می‌توانند با کارایی بیشتری توسط GNN پردازش شوند.

نویسندگان دریافتند که این رویکرد تعبیه گراف برای یادگیری SPT برای JSSP موثر است. PDR هایی که با استفاده از این رویکرد آموخته شدند، توانستند از PDR های طراحی شده دستی موجود بهتر عمل کنند و به خوبی به نمونه های بزرگتر JSSP تعمیم داده شوند.

۳-۲-۲-انتخاب کردن تصمیم در هر s_t

نویسندگان بیان می‌کنند که با استفاده از شبکه پرسپترون چندلایه (MLP)، برای هر کاری که امکان انجام شدن دارد، یک امتیاز محاسبه می‌شود. سپس با استفاده از یک تابع softmax، این امتیاز ها به احتمالاتی برای هر a_t در s_t تبدیل می‌شود. در نهایت از بین کارها، کاری که بیشتری میزان احتمال را دارد برای انجام شدن انتخاب می‌شود.

۳-۳-الگوریتم یادگیری

نویسندگان از Proximal Policy Optimization (PPO) برای آموزش شبکه خط مشی خود استفاده می کنند. PPO یک الگوریتم بازیگر منتقد (actor-critic) است که برای مسائل یادگیری تقویتی با فضاهای تصمیم پیوسته مناسب است. در PPO، شبکه بازیگر مسئول انتخاب

تصمیم‌ها و شبکه منتقد مسئول ارزیابی ارزش هر حالت (state) است. هردوی این شبکه‌ها از GIN با معماری یکسانی استفاده خواهند کرد. نویسندگان از یک فرآیند دو مرحله ای برای آموزش شبکه خط مشی خود استفاده می کنند:

به‌روزرسانی خط‌مشی: نویسندگان مجموعه‌ای از تاپل‌های تجربه را به شکل $(s_t, a_t, s_{t+1}, r_{t+1})$ تولید می‌کنند. سپس از این تاپل‌های تجربه برای به روز رسانی شبکه سیاست با استفاده از الگوریتم PPO استفاده می کنند. الگوریتم PPO شبکه خط مشی را با به حداکثر رساندن مقدار مورد انتظار یک تابع پاداش جایگزین، که تابعی از پاداش های تجمعی و احتمالات سیاست است، به روز می کند.

به روزرسانی شبکه منتقد: نویسندگان همچنین شبکه منتقد را با استفاده از همان دسته از تاپل های تجربه به روز می‌کنند. شبکه انتقادی با به حداقل رساندن میانگین مربعات خطا بین مقدار تخمینی حالت و مقدار واقعی حالت به روز می شود.

نویسندگان از تعدادی تکنیک برای بهبود آموزش شبکه خط مشی خود استفاده می کنند، از جمله clipped surrogate loss function و یک trust region constraint. این تکنیک‌ها به جلوگیری از تطبیق بیش از حد شبکه سیاست با داده‌های آموزشی کمک می‌کنند و اطمینان می‌دهند که شبکه سیاست در برابر تغییرات در محیط مقاوم است.

۳-۵-تنظیم ابرپارامترها

نویسندگان از انواع ابرپارامترها برای آموزش شبکه خط مشی خود استفاده کردند. این ابرپارامترها شامل تعداد تکرارها، تعداد مسیرهای مستقل در هر تکرار، تعداد نمونه‌های اعتبارسنجی، مقیاس نرمال‌سازی، تعداد تکرارهای GIN، مقدار، تعداد لایه‌های پنهان در MLPها، نرخ یادگیری، و ضریب تنزیل. آنها دریافتند که بهترین عملکرد با تنظیمات زیر به دست آمده است:

تعداد تکرار: ۱۰۰۰۰

تعداد مسیرهای مستقل در هر تکرار: ۴

تعداد نمونه های اعتبارسنجی: ۱۰۰.

تعداد تکرارهای GIN: ۲

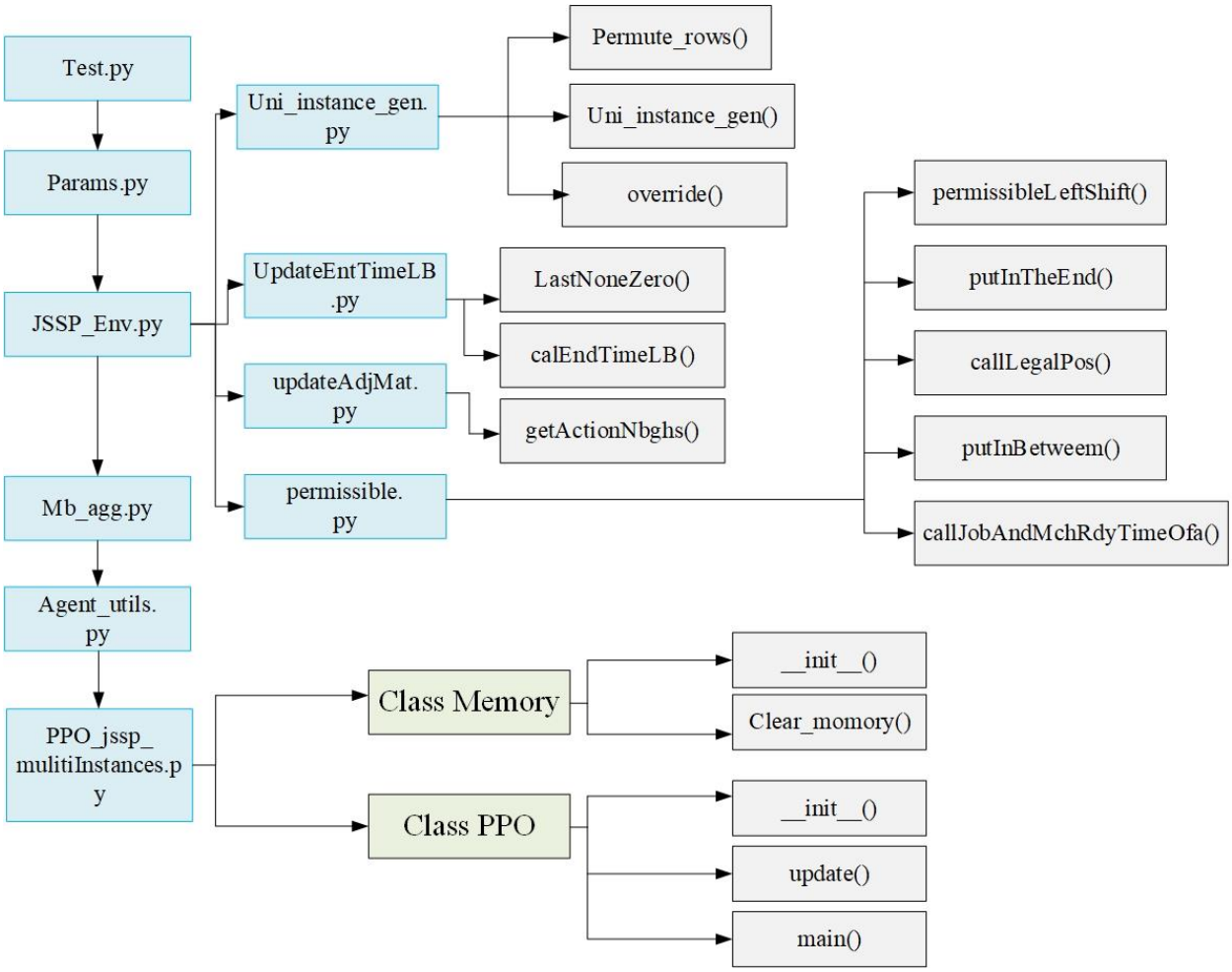
تعداد لایه های پنهان در MLP ها: ۲ لایه پنهان با بعد پنهان ۶۴ برای لایه های GIN و ۲ لایه پنهان با بعد پنهان ۳۲ برای هر دو شبکه بازیگر و منتقد.

ضریب تنزیل: ۱

نویسندگان دریافتند که این فراپارامترها به شبکه خط مشی آنها اجازه می دهد تا بهترین عملکرد را در انواع نمونه های JSSP به دست آورد. آنها همچنین دریافتند که شبکه خط مشی آنها می تواند به خوبی به نمونه های بزرگتر بدون آموزش مجدد تعمیم دهد.

۴-کد کردن مسئله

برای کد کردن مسئله، ابتدا هر بخش مسئله به چند فایل پایتون شکسته شد. هر فایل شامل توابع و کلاس‌هایی است که در حل مسئله کمک می‌کنند. شکل ۲ نحوه‌ی ارتباط فایل‌ها و اینکه هرکدام شامل چه توابع و کلاس‌هایی می‌شوند را نشان می‌دهد. در ادامه، به توضیح فایل‌های مهم و توابع موجود در هرکدام پرداخته می‌شود.



شکل ۴: نحوه ارتباط فایل‌ها با یکدیگر

حال به توضیح فایل‌های مهم و نحوه کارکرد آنها می‌پردازیم.

با توجه به اینکه وظیفه توابع __init__() در هر کلاس، این است که برای هر شی ساخته شده از آن کلاس، گروهی از مقادیر اولیه را ایجاد کند از ارائه توضیح اضافه درمورد توابع __init__() در هر فایل خودداری می‌کنیم. همچنین درمورد عبارت if name==”main” می‌توان گفت که هنگامی که هر فایل فراخوانی می‌شود، به جای شروع پردازش فایل از خط اول، از خط بعد از این عبارت، پردازش فایل آغاز می‌شود.

۴-۱-فایل Params.py

از آنجایی که هر بخش از فایل‌ها مسئول انجام بخشی از وظایف است، ممکن است تنظیم پارامترها در فایل‌های جداگانه به سادگی صورت نگیرد. یک راه حل این است که هنگام اجرا کردن برنامه از طریق خط فرمان (command line) پارامترها را از طریق خط فرمان به برنامه داده شوند. مزیت این کار این است که همه‌ی پارامترها در یک مکان قابل دسترسی هستند. شکل ۳، کد مربوط به این فایل را نمایش می‌دهد.

```python
import argparse

parser = argparse.ArgumentParser(description='Arguments for ppo_jssp')
# args for device
parser.add_argument('--device', type=str, default="cuda", help='Number of jobs of instances')
# args for env
parser.add_argument('--n_j', type=int, default=15, help='Number of jobs of instance')
parser.add_argument('--n_m', type=int, default=15, help='Number of machines instance')
parser.add_argument('--rewardscale', type=float, default=0., help='Reward scale for positive rewards')
parser.add_argument('--init_quality_flag', type=bool, default=False, help='Flag of whether init state quality is 0, True for 0')
parser.add_argument('--low', type=int, default=1, help='LB of duration')
parser.add_argument('--high', type=int, default=99, help='UB of duration')
parser.add_argument('--np_seed_train', type=int, default=200, help='Seed for numpy for training')
parser.add_argument('--np_seed_validation', type=int, default=200, help='Seed for numpy for validation')
parser.add_argument('--torch_seed', type=int, default=600, help='Seed for torch')
parser.add_argument('--et_normalize_coef', type=int, default=1000, help='Normalizing constant for feature LBs (end time), normalization way: fea/constant')
parser.add_argument('--wkr_normalize_coef', type=int, default=100, help='Normalizing constant for wkr, normalization way: fea/constant')
# args for network
parser.add_argument('--num_layers', type=int, default=3, help='No. of layers of feature extraction GNN including input layer')
parser.add_argument('--neighbor_pooling_type', type=str, default='sum', help='neighbour pooling type')
parser.add_argument('--graph_pool_type', type=str, default='average', help='graph pooling type')
parser.add_argument('--input_dim', type=int, default=2, help='number of dimension of raw node features')
parser.add_argument('--hidden_dim', type=int, default=64, help='hidden dim of MLP in fea extract GNN')
parser.add_argument('--num_mlp_layers_feature_extract', type=int, default=2, help='No. of layers of MLP in fea extract GNN')
parser.add_argument('--num_mlp_layers_actor', type=int, default=2, help='No. of layers in actor MLP')
parser.add_argument('--hidden_dim_actor', type=int, default=32, help='hidden dim of MLP in actor')
parser.add_argument('--num_mlp_layers_critic', type=int, default=2, help='No. of layers in critic MLP')
parser.add_argument('--hidden_dim_critic', type=int, default=32, help='hidden dim of MLP in critic')
# args for PPO
parser.add_argument('--num_envs', type=int, default=4, help='No. of envs for training')
parser.add_argument('--max_updates', type=int, default=10000, help='No. of episodes of each env for training')
parser.add_argument('--lr', type=float, default=2e-5, help='lr')
parser.add_argument('--decayflag', type=bool, default=False, help='lr decayflag')
parser.add_argument('--decay_step_size', type=int, default=2000, help='decay_step_size')
parser.add_argument('--decay_ratio', type=float, default=0.9, help='decay_ratio, e.g. 0.9, 0.95')
parser.add_argument('--gamma', type=float, default=1, help='discount factor')
parser.add_argument('--k_epochs', type=int, default=1, help='update policy for K epochs')
parser.add_argument('--eps_clip', type=float, default=0.2, help='clip parameter for PPO')
parser.add_argument('--vloss_coef', type=float, default=1, help='critic loss coefficient')
parser.add_argument('--ploss_coef', type=float, default=2, help='policy loss coefficient')
parser.add_argument('--entloss_coef', type=float, default=0.01, help='entropy loss coefficient')

configs = parser.parse_args()
```

شکل ۵: فایل پایتون params

برای اینکه آرگومان‌ها را تعریف کنیم، از ماژول argparse استفاده می‌کنیم. ابتدا یک شی از کلاس ArgumentParser ساخته می‌شود و سپس با استفاده از متد add_argument() آرگومان‌ها را به شی اضافه می‌کنیم. البته احتیاجی به تنظیم همه‌ی آرگومان‌ها نیست چون اکثر آرگومان‌ها دارای مقدار پیشفرض هستند. سپس همه مقادیر آرگومان‌ها در متغیر configs ذخیره می‌شوند. سپس هرکجا به پارامترها احتیاج بود، با وارد کردن متغیر config می‌توان به پارامترها دسترسی داشت.

۴-۲-فایل JSSP_Env.py

در این فایل یک کلاس به اسم SJSSP برنامه نویسی شده است که این کلاس شامل چند تابع است. وظیفه اصلی این کلاس و متدهای آن، این است که با توجه به فرایند تصمیم گیری مارکوف، فضای حالت را برای مسئله JSS کد کند و این فضا را در یک شی ذخیره کند. حال به بررسی توابع حاضر در این فایل می‌پردازیم.

۴-۲-۱-متد done()

متد done() یک مقدار صفر یا یک را برمی‌گرداند که نشان می‌دهد آیا راه حل جزئی فعلی کامل است یا خیر. اگر راه‌حل جزئی شامل تمام کارها باشد، True را برمی‌گرداند و در غیر این صورت False را برمی‌گرداند.

۴-۲-۲- متد reset()

متد reset() محیط را به حالت اولیه بازنشانی می کند. کارهای زیر را انجام می دهد: ماتریس adj را که گراف همسایگی MDP را نشان می‌دهد، مقداردهی اولیه می کند. ماتریس LBs را صفر می‌کند، که کران‌های پایین‌تر زمان پایان را نشان می‌دهد. بردارهای omega و mask

را که نشان دهنده بردارهای omega و mask شدنی برای جواب جزئی فعلی هستند را ایجاد می کند. بردار finished_mark را راه‌اندازی می‌کند، که نشان می‌دهد کدام کارها در حل جزئی تکمیل شده‌اند.متغیر posRewards را راه‌اندازی می‌کند، که پاداش تجمعی را برای پاداش‌های مثبت دنبال می‌کند. شکل ۴ کد پایتون متد reset() را نشان می‌دهد.

```python
@override
def reset(self, data):

    self.step_count = 0
    self.m = data[-1]
    self.dur = data[0].astype(np.single)
    self.dur_cp = np.copy(self.dur)
    # record action history
    self.partial_sol_sequeence = []
    self.flags = []
    self.posRewards = 0

    # initialize adj matrix
    conj_nei_up_stream = np.eye(self.number_of_tasks, k=-1, dtype=np.single)
    conj_nei_low_stream = np.eye(self.number_of_tasks, k=1, dtype=np.single)
    # first column does not have upper stream conj_nei
    conj_nei_up_stream[self.first_col] = 0
    # last column does not have lower stream conj_nei
    conj_nei_low_stream[self.last_col] = 0
    self_as_nei = np.eye(self.number_of_tasks, dtype=np.single)
    self.adj = self_as_nei + conj_nei_up_stream

    # initialize features
    self.LBs = np.cumsum(self.dur, axis=1, dtype=np.single)
    self.initQuality = self.LBs.max() if not configs.init_quality_flag else 0
    self.max_endTime = self.initQuality
    self.finished_mark = np.zeros_like(self.m, dtype=np.single)

    fea = np.concatenate((self.LBs.reshape(-1, 1)/configs.et_normalize_coef,
                          # self.dur.reshape(-1, 1)/configs.high,
                          # wkr.reshape(-1, 1)/configs.wkr_normalize_coef,
                          self.finished_mark.reshape(-1, 1)), axis=1)
    # initialize feasible omega
    self.omega = self.first_col.astype(np.int64)

    # initialize mask
    self.mask = np.full(shape=self.number_of_jobs, fill_value=0, dtype=bool)

    # start time of operations on machines
    self.mchsStartTimes = -configs.high * np.ones_like(self.dur.transpose(), dtype=np.int32)
    # Ops ID on machines
    self.opIDsOnMchs = -self.number_of_jobs * np.ones_like(self.dur.transpose(), dtype=np.int32)

    self.temp1 = np.zeros_like(self.dur, dtype=np.single)

    return self.adj, fea, self.omega, self.mask
```

شکل ۶: کد پایتون متد reset()

۴-۲-۳-متد step()

متد step() یک تصمیم را به عنوان ورودی دریافت می‌کند، و حالت، پاداش و متغیر صفر و یک done را برمی‌گرداند. این متد، مراحل زیر را انجام می‌دهد: بررسی می‌کند که آیا عملکرد داده شده معتبر است (یعنی قبلاً در راه حل جزئی نبوده). اگر عمل معتبر است: اطلاعات اولیه (step_count ، finished_mark و dur_a) را به روز می کند. برای به‌روزرسانی زمان شروع عملیات جدید، تابع permissibleLeftShift() را فراخوانی می‌کند. omega یا mask را بر اساس موقعیت عملیات جدید به روز می کند. temp1 و LBs را به روزرسانی می‌کند. ماتریس adj را

به روز می کند. بردار ویژگی را محاسبه می کند. مقدار تابع پاداش را بر اساس تفاوت بین حداکثر زمان پایان کران پایین و کیفیت اولیه (یا پاداش قبلی اگر اولین اقدام باشد) محاسبه می کند.. شکل ۴ کد پایتون این متد را نمایش می‌دهد

```python
@override
def step(self, action):
    # action is a int 0 - 224 for 15x15 for example
    # redundant action makes no effect
    if action not in self.partial_sol_sequeence:

        # UPDATE BASIC INFO:
        row = action // self.number_of_machines
        col = action % self.number_of_machines
        self.step_count += 1
        self.finished_mark[row, col] = 1
        dur_a = self.dur[row, col]
        self.partial_sol_sequeence.append(action)

        # UPDATE STATE:
        # permissible left shift
        startTime_a, flag = permissibleLeftShift(a=action, durMat=self.dur, mchMat=self.m, mchsStartTimes=self.mchsStartTimes, opIDsOnMchs=self.opIDsOnMchs)
        self.flags.append(flag)
        # update omega or mask
        if action not in self.last_col:
            self.omega[action // self.number_of_machines] += 1
        else:
            self.mask[action // self.number_of_machines] = 1

        self.temp1[row, col] = startTime_a + dur_a

        self.LBs = calEndTimeLB(self.temp1, self.dur_cp)

        # adj matrix
        precd, succd = self.getNghbs(action, self.opIDsOnMchs)
        self.adj[action] = 0
        self.adj[action, action] = 1
        if action not in self.first_col:
            self.adj[action, action - 1] = 1
        self.adj[action, precd] = 1
        self.adj[succd, action] = 1
        if flag and precd != action and succd != action:  # Remove the old arc when a new operation inserts between two operations
            self.adj[succd, precd] = 0

    # prepare for return
    fea = np.concatenate((self.LBs.reshape(-1, 1)/configs.et_normalize_coef,
                          self.finished_mark.reshape(-1, 1)), axis=1)
    reward = - (self.LBs.max() - self.max_endTime)
    if reward == 0:
        reward = configs.rewardscale
        self.posRewards += reward
    self.max_endTime = self.LBs.max()

    return self.adj, fea, reward, self.done(), self.omega, self.mask
```

شکل ۷: کد پایتون متد step()

حال به بررسی فایل‌هایی پرداخته می‌شود که ایجاد این محیط را تسهیل می‌کنند.

۴-۳-فایل updateAdjMat.py

این کد تابعی به نام getActionNbghs() را تعریف می‌کند که پیشنیازها و پسنیازهای یک کار را در ماتریس opIDsOnMchs شناسایی این تابع مشخص می‌کند که براساس تصمیمی که در هر زمان اتخاذ می‌شود، چه کارهایی قابل انجام خواهند بود. تابع getActionNbghs دو پارامتر دارد: action که تصمیم فعلی را نشان می‌دهد و opIDsOnMchs که ماتریسی حاوی شناسه‌های عملیات برای هر کار و ماشین است.

این تابع از np.where() برای یافتن مختصات تصمیم اتخاذ شده، در ماتریس opIDsOnMchs استفاده می کند. اولین عنصر از تاپل برگشت داده شده، نماینده کار و عنصر دوم نماینده ماشین است. شکل ۶ کد پایتون این فایل را نمایش می‌دهد:

```python
from Params import configs
import numpy as np


def getActionNbghs(action, opIDsOnMchs):
    coordAction = np.where(opIDsOnMchs == action)
    precd = opIDsOnMchs[coordAction[0], coordAction[1] - 1 if coordAction[1].item() > 0 else coordAction[1]].item()
    succdTemp = opIDsOnMchs[coordAction[0], coordAction[1] + 1 if coordAction[1].item() + 1 < opIDsOnMchs.shape[-1] else coordAction[1]].it
    succd = action if succdTemp < 0 else succdTemp
    # precedX = coordAction[0]
    # precedY = coordAction[1] - 1 if coordAction[1].item() > 0 else coordAction[1]
    # succdX = coordAction[0]
    # succdY = coordAction[1] + 1 if coordAction[1].item()+1 < opIDsOnMchs.shape[-1] else coordAction[1]
    return precd, succd


if __name__ == '__main__':
    opIDsOnMchs = np.array([[7, 29, 33, 16, -6, -6],
                            [6, 18, 28, 34, 2, -6],
                            [26, 31, 14, 21, 11, 1],
                            [30, 19, 27, 13, 10, -6],
                            [25, 20, 9, 15, -6, -6],
                            [24, 12, 8, 32, 0, -6]])

    action = 16
    precd, succd = getActionNbghs(action, opIDsOnMchs)
    print(precd, succd)
```

شکل ۸: کد پایتون فایل updatAdjMat.py

۴-۴-فایل updateEntTimeLB.py

این فایل به برنامه این امکان را می‌دهد تا حد پایین C_{max} را برای هر جواب جزئی محاسبه کند. شکل ۷ کد پایتون مربوط به این فایل را نمایش می‌دهد.

```python
import numpy as np


def lastNonZero(arr, axis, invalid_val=-1):
    mask = arr != 0
    val = arr.shape[axis] - np.flip(mask, axis=axis).argmax(axis=axis) - 1
    yAxis = np.where(mask.any(axis=axis), val, invalid_val)
    xAxis = np.arange(arr.shape[0], dtype=np.int64)
    xRet = xAxis[yAxis >= 0]
    yRet = yAxis[yAxis >= 0]
    return xRet, yRet


def calEndTimeLB(temp1, dur_cp):
    x, y = lastNonZero(temp1, 1, invalid_val=-1)
    dur_cp[np.where(temp1 != 0)] = 0
    dur_cp[x, y] = temp1[x, y]
    temp2 = np.cumsum(dur_cp, axis=1)
    temp2[np.where(temp1 != 0)] = 0
    ret = temp1+temp2
    return ret


if __name__ == '__main__':
    dur = np.array([[1, 2], [3, 4]])
    temp1 = np.zeros_like(dur)

    temp1[0, 0] = 1
    temp1[1, 0] = 3
    temp1[1, 1] = 5
    print(temp1)

    ret = calEndTimeLB(temp1, dur)
```

شکل ۹: کد پایتون فایل updateEntTimeLB.py

۴-۵-فایل uniform_instance_gen.py

این فایل با استفاده از پارامترهای داده شده، به تعداد ماشین‌ها و کارها، اعدادی رندوم بین حد بالا و پایین تعیین شده تولید می‌کند. همچنین یک تابع آن، وظیفه‌ی ایجاد جایگشت بین کارها و ماشینها را دارد. تابع override یک تابع decorator است که هر تابعی که به آن پاس داده می‌شود را بدون تغییر برمی‌گرداند. این تابع باعث می‌شود که درصورت داشتن دو تابع همنام بین یک کلاس والد و یک کلاس فرزند، دستورات تابعی که decorator دارد اجرا شود. شکل ۸ کد پایتون uniform_instance_gen.py را نمایش می‌دهد.

```python
import numpy as np


def permute_rows(x):
    '''
    x is a np array
    '''
    ix_i = np.tile(np.arange(x.shape[0]), (x.shape[1], 1)).T
    ix_j = np.random.sample(x.shape).argsort(axis=1)
    return x[ix_i, ix_j]


def uni_instance_gen(n_j, n_m, low, high):
    times = np.random.randint(low=low, high=high, size=(n_j, n_m))
    machines = np.expand_dims(np.arange(1, n_m+1), axis=0).repeat(repeats=n_j, axis=0)
    machines = permute_rows(machines)
    return times, machines


def override(fn):
    """
    override decorator
    """
    return fn
```

شکل ۱۰: کد پایتون uniform_instance_gen.py

۴-۶-فایل permissibleLS

تابع permissibleLeftShift بررسی می‌کند که آیا می‌توان یک کار را زودتر از زمانی که برنامه‌ریزی شده انجام داد، و در صورت امکان آن را برای زمانی زودتر برنامه ریزی می‌کند (به اصلاح، آن کار را Left shift می‌کند.) سایر توابع حاضر در این فایل، توابع کمکی هستند و بخشی از الگوریتم انجام عمل شیفت دادن به چپ را به عهده می‌گیرند.

۴-۷-فایل agent_utils.py

یک تابع در این فایل ابتدا توزیع احتمال برای هر یک از تصمیم‌ها را به وسیله یک تابع softmax محاسبه می‌کند. سه تابع دیگر، عمل اتخاذ تصمیم را به روش‌های متفاوت انجام می‌دهند. شکل ۹، کد پایتون این فایل را نمایش می‌دهد.

۴-۷-۱-تابع eval_actions()

این تابع مجموعه ای از تصمیمات را با استفاده از توزیع softmax بر روی توزیع احتمال p ارزیابی می کند و لگاریتم احتمالات اعمال (ret) و میانگین آنتروپی توزیع را برمی‌گرداند.

۴-۷-۲- توابع select_action() و greedy_select_action() و sample_select_action()

توابع select_action() و sample_select_action() تصمیمات را بر اساس توزیع احتمالی p ، اتخاذ می‌کنند و تابع greedy_select_action() تصمیمات را براساس کاری که بیشترین مقدار خروجی از تابع softmax دارد انتخاب می‌کند.

```python
from torch.distributions.categorical import Categorical


def select_action(p, cadidate, memory):
    dist = Categorical(p.squeeze())
    s = dist.sample()
    if memory is not None: memory.logprobs.append(dist.log_prob(s))
    return cadidate[s], s


# evaluate the actions
def eval_actions(p, actions):
    softmax_dist = Categorical(p)
    ret = softmax_dist.log_prob(actions).reshape(-1)
    entropy = softmax_dist.entropy().mean()
    return ret, entropy


# select action method for test
def greedy_select_action(p, candidate):
    _, index = p.squeeze().max(0)
    action = candidate[index]
    return action


# select action method for test
def sample_select_action(p, candidate):
    dist = Categorical(p.squeeze())
    s = dist.sample()
    return candidate[s]
```

شکل ۱۱: کد پایتون فایل agent_utils.py

۴-۸-فایل mb_agg.py

دو تابع حاضر در این فایل، وظیفه‌ی تبدیل مدل JSSP به یک شبکه عصبی مبتنی بر گراف را دارند. شکل ۱۰ نشان دهنده‌ی کد پایتون این دو تابع است.

```python
import torch


def aggr_obs(obs_mb, n_node):
    # obs_mb is [m, n_nodes_each_state, fea_dim], m is number of nodes in batch
    idxs = obs_mb.coalesce().indices()
    vals = obs_mb.coalesce().values()
    new_idx_row = idxs[1] + idxs[0] * n_node
    new_idx_col = idxs[2] + idxs[0] * n_node
    idx_mb = torch.stack((new_idx_row, new_idx_col))
    # print(idx_mb)
    # print(obs_mb.shape[0])
    adj_batch = torch.sparse.FloatTensor(indices=idx_mb,
                                         values=vals,
                                         size=torch.Size([obs_mb.shape[0] * n_node,
                                                          obs_mb.shape[0] * n_node]),
                                         ).to(obs_mb.device)
    return adj_batch


def g_pool_cal(graph_pool_type, batch_size, n_nodes, device):
    # batch_size is the shape of batch
    # for graph pool sparse matrix
    if graph_pool_type == 'average':
        elem = torch.full(size=(batch_size[0]*n_nodes, 1),
                          fill_value=1 / n_nodes,
                          dtype=torch.float32,
                          device=device).view(-1)
    else:
        elem = torch.full(size=(batch_size[0] * n_nodes, 1),
                          fill_value=1,
                          dtype=torch.float32,
                          device=device).view(-1)
    idx_0 = torch.arange(start=0, end=batch_size[0],
                         device=device,
                         dtype=torch.long)
    # print(idx_0)
    idx_0 = idx_0.repeat(n_nodes, 1).t().reshape((batch_size[0]*n_nodes, 1)).squeeze()

    idx_1 = torch.arange(start=0, end=n_nodes*batch_size[0],
                         device=device,
                         dtype=torch.long)
    idx = torch.stack((idx_0, idx_1))
    graph_pool = torch.sparse.FloatTensor(idx, elem,
                                          torch.Size([batch_size[0],
                                                      n_nodes*batch_size[0]])
                                          ).to(device)

    return graph_pool
```

شکل ۱۲:کد پایتون فایل mb_agg.py

۴-۹-فایل PPO_jssp_multiInstances

این فایل شامل دو کلاس است. کلاس اول، کلاس Memory است. وظیفه اصلی این کلاس، ذخیره اطلاعات مورد نیاز برای انجام محاسبات مربوط به یادگیری تقویتی عمیق است.

کلاس دوم، کلاس PPO است که وظیفه ایجاد الگوریتم Proximal Policy Optimization را دارد. در ادامه به بررسی متد مربوط به این کلاس می‌پردازیم. درواقع، تمامی فایل‌هایی که تا کنون درمورد آنها توضیح داده شد، در فایل ()main حاضر هستند و وظیفه آموزش شبکه عصبی اصلی و ذخیره آن را به عهده دارند.

۴-۹-۱-متد ()update

این متد، اطلاعات ذخیره شده در حافظه، تعداد وظایف و گراف تولید شده از مسئله را دریافت می‌کند و سیاست پیش گرفته شده را به روز می‌کند. به روز رسانی سیاست به این شکل اتفاق می‌افتد: مشاهدات، پاداش‌های جمع‌آوری شده، تصمیمات نامزد، متغیر mask و لگاریتم احتمال‌های قبلی جمع‌آوری می‌شوند. با استفاده از ضریب تنزیل، پاداش محاسبه می‌شود.

۴-۹-۲-متد ()main

کد ارائه شده در این متد، یک حلقه آموزش جامع برای الگوریتم PPO را در زمینه حل مسئله JSSP با استفاده از یادگیری تقویتی نشان می‌دهد. این متد، شامل تولید داده، مدیریت حافظه، ارزیابی سیاست، بهبود سیاست، اعتبارسنجی و ذخیره مدل برای آموزش موثر عامل PPO است که پیش از این، به معرفی فایل‌های مربوط به آنها پرداخته شده.

حلقه اصلی آموزش شامل این مراحل است: راه اندازی محیط: فهرستی از محیط های SJSSP (envs) و یک شی uniform_instance_gen را برای تولید نمونه های تصادفی JSSP راه اندازی می‌کند.

Initialize Memory Buffer: فهرستی از اشیاء حافظه (حافظه ها)، یکی برای هر محیط ایجاد می‌کند. این بافرهای حافظه، تاپل‌های تجربه را در طول تمرین ذخیره می‌کنند. سپس یک مدل برای بازیگر و یک مدل برای منتقد تولید می‌شود.

در ادامه، با ایجاد دسته‌ای از تانسورهای مشاهده، فضای حالت تولید می‌شود. همچنین یک تانسور که نماینده تصمیم‌های قابل اخذ در حالت s_t ایجاد می‌شود. سپس با استفاده از تابع softmax به یک توزیع احتمال برای تانسور مربوط به تصمیمات قابل اخذ تولید می‌شود که سیاست در پیش گرفته شده با توجه به این احتمالات تصیم‌گیری را انجام می‌دهد. در نهایت، سیاست با توجه به تابع پاداش و میانگین پاداش از دست رفته (v_loss) اصلاح می‌شود.

در ادامه مدل با استفاده از صد دیتاست تصادفی اعتبارسنجی می‌شود. و اگر نتایج مدل به دست آمده از مدل قبلی بهتر بود، پارامترهای مربوط به مدل بهتر ذخیره خواهند شد.

۵- استفاده از کد نوشته شده

در این بخش به بررسی نحوه استفاده از این کد، با استفاده از خط فرمان و فضای گوگل کولب پرداخته می‌شود. **با توجه به حجم بالای کتابخانه pytorch و نیازمندی عملیات‌ها به توان محاسباتی بالا، اکیداً توصیه می‌شود کدها در فضای گوگل کولب اجرا شود.**

۵-۱-تمرین و تست یک شبکه عصبی جدید

برای استفاده از محیط کولب، ابتدا فولدر ارائه شده را در گوگل درایو خود آپلود نمایید. توجه فرمایید برای استفاده از آدرس دهی پیشفرض موجود در فایل ژوپیتر، فولدر باید مستقیماً در خود درایو آپلود شود و نه در یکی از فولدرهای موجود در درایو شما. در غیر اینصورت آدرس‌دهی موجود در اولین بلوک را با توجه به مسیری که فولدر را در آن آپلود می‌کنید، تغییر دهید. شکل ۱۱، آدرس دهی پیشفرض را نمایش می‌دهد.

```
[2]  %cd /content/drive/MyDrive/L2D-main

/content/drive/MyDrive/L2D-main
```

شکل ۱۳: آدرس دهی پیشفرض فولدر

حال فرض کنید می‌خواهیم یک شبکه عصبی برای پیدا کردن یک make span مناسب برای ۱۷ کار و ۱۲ ماشین پیدا کنیم. ابتدا باید برای تمرین دادن شبکه عصبی یک دیتاست ایجاد کنیم. برای این منظور از فایل generate_data.py استفاده می‌کنیم که کد مربوطه در بلوک بعدی قرار گرفته است. شکل ۱۲ بلوکی که وظیفه تولید داده‌ها را به عهده دارد نشان می‌دهد.

```
import numpy as np
from uniform_instance_gen import uni_instance_gen

j = 17
m = 12
l = 1
h = 99
batch_size = 100
seed = 150

np.random.seed(seed)

data = np.array([uni_instance_gen(n_j=j, n_m=m, low=l, high=h) for _ in range(batch_size)])
print(data.shape)
np.save('./DataGen/generatedData{}_{}_Seed{}.npy'.format(j, m, seed), data)
```

```
(100, 2, 17, 12)
```

شکل ۱۴:تولید داده‌ها

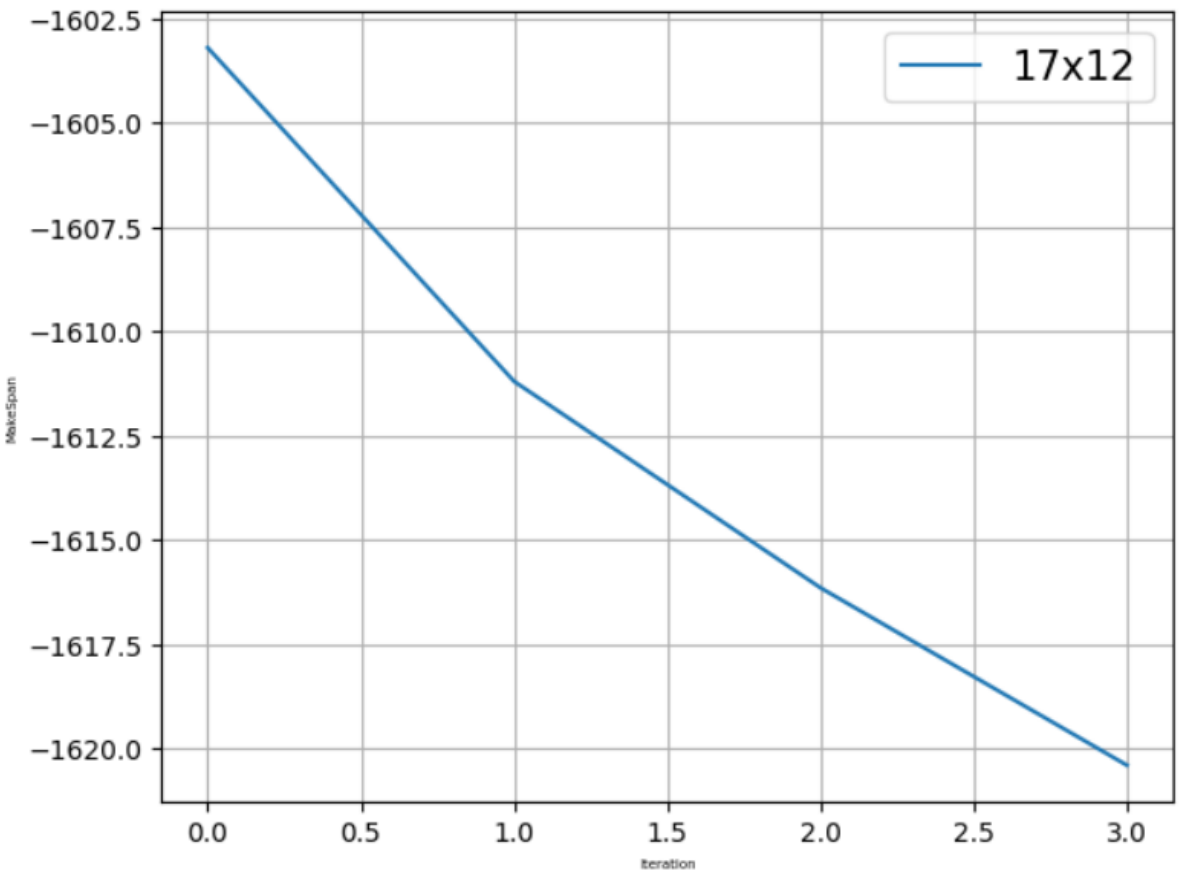
حال فایل PPO_jssp_multiInstances.py را با استفاده از خط فرمان فراخوانی می‌کنیم و این پارامترها را تنظیم می‌کنیم: --n_j

که نشان دهنده‌ی تعداد کار هاست، --n_m که نشان‌دهنده‌ی تعداد ماشین‌هاست، --max_updates نشان‌هنده‌ی تعداد دفعاتی است که شبکه عصبی تا رسیدن به پایان مسئله، بر روی یک محیط تمرین داده می‌شود و --num_envs تعداد محیط‌هایی است که برای تمرین دادن شبکه عصبی ساخته می‌شود. توجه کنید که اجرای این مرحله می‌تواند بسیار طولانی باشد و تاثیر زیادی بر دقت مدل داشته باشد. شکل ۱۳ بلوک مربوط به این خط فرمان را نمایش می‌دهد.

```
[ ]  !python3 PPO_jssp_multiInstances.py --n_j=17 --n_m=12 --max_updates=50 --num_envs=2
```

شکل ۱۵: بلوک مربوط به خط فرمان

بلوک بعدی یک نمودار رسم می‌کند که makespan را در هر اپیزود نمایش می‌دهد. شکل ۱۴ نشان‌دهنده‌ی نمودار رسم شده از طریق اجرای این بلوک است.



شکل ۱۶: نمودار آموزش شبکه عصبی

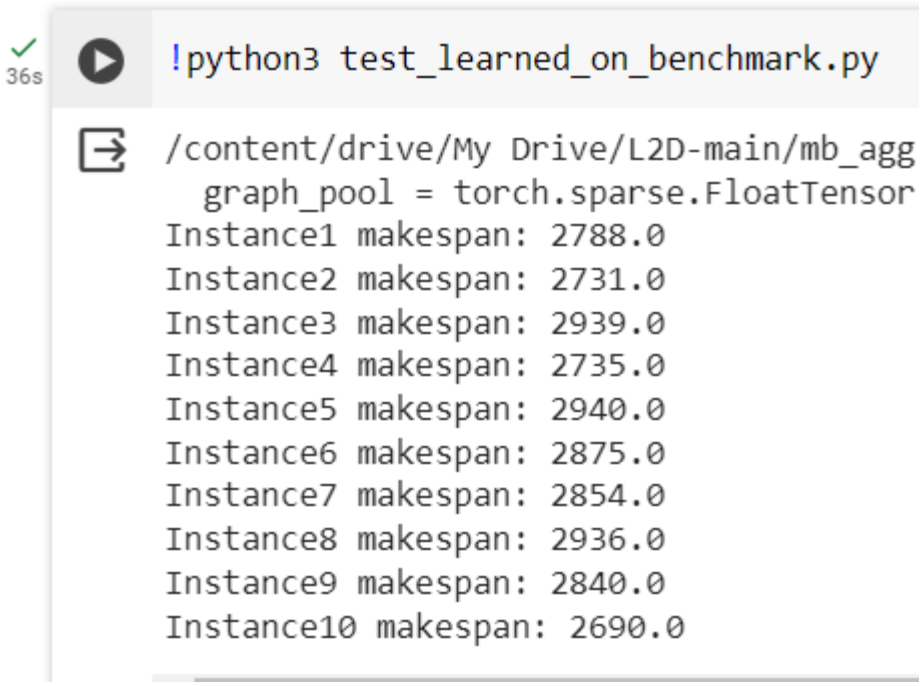
و بلوک آخر شبکه عصبی تمرین داده شده را بر روی داده‌های بنچمارک تست می‌کند. متاسفانه تا زمان نوشتن گزارش، باگ مربوط به دریافت پارامترهای مربوط به شبکه عصبی برطرف نشد و باید دو پارامتر Nn_j-- که نشان دهنده‌ی تعداد کارهاییست که شبکه روی آن آموزش دیده، و Nn_m-- که تعداد ماشین هاییست که شبکه روی آن آموزش دیده را به صورت دستی و در فایل validation.py تغییر داد و مجدد در فولدر مربوطه در گوگل کولب آپلود کرد.

برای این منظور، مقدار default مربوط به این دو پارامتر را به صورت دستی تغییر می‌دهیم. شکل ۱۵ این تغییر را نمایش می‌دهد.

```
                    help='Number of machines instances to test')
parser.add_argument('--Nn_j', type=int, default=17,
                    help='Number of jobs on which to be loaded net are trained')
parser.add_argument('--Nn_m', type=int, default=12,
                    help='Number of machines on which to be loaded net are trained')
```

شکل ۱۷: تغییر دستی پارامترها

با اجرای این بلوک، شبکه‌ی ایجاد شده در معرض تست قرار می‌گیرد. شکل ۱۶ این بلوک را نمایش می‌دهد.



شکل ۱۸: بلوک مربوط به تست شبکه عصبی

همچنین برای راحتی در استفاده، در برخی از ابعاد، یکسری دیتاست و دیتای بنچمارک و شبکه عصبی مربوط به هر کدام، از قبل آماده شده و به ترتیب در فولدرهای DataGen، BenchDataNmpy، و SavedNetwork، قرار گرفته است. برای استفاده از این داده‌ها و شبکه عصبی مربوطه، احتیاجی به تمرین شبکه عصبی و اجرای بلوک یک، دو و سه نیست و می‌توان به صورت مستقیم به سراغ بلوک آخر و تنظیم پارامترهای مربوط به آن رفت.

همچنین با توجه به موجود بودن دیتاست‌ها می‌توان برای تمرین دادن شبکه‌های عصبی جدید، از داده‌های موجود استفاده کرد. جدول ۱ ابعاد از پیش آماده شده را نمایش می‌دهد.

جدول ۱: اطلاعات مربوط به بخش‌های از پیش آماده شده

تعداد ماشین * تعداد کار	شبکه عصبی	داده‌های آموزش	داده‌های بنچمارک
۶*۶	موجود	موجود	
۱۰*۱۰	موجود	موجود	
۱۰*۱۵		موجود	
۱۵*۱۵	موجود	موجود	
۲۰*۱۰		موجود	
۲۰*۱۵	موجود	موجود	موجود
۲۰*۲۰	موجود	موجود	موجود
۳۰*۱۵	موجود	موجود	موجود
۳۰*۲۰	موجود	موجود	موجود
۴۰*۱۵			موجود
۴۰*۲۰			موجود

۵۰*۱۵			موجود
۵۰*۲۰		موجود	موجود
۱۰۰*۲۰		موجود	موجود
۲۰۰*۵۰		موجود	

۶-نتیجه‌گیری

در این گزارش تلاش شد کد پایتونی بر اساس مقاله‌ی Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning ارائه شود که در آن با استفاده از یادگیری عمیق تقویتی به آموزش و تست شبکه‌های عصبی پرداخته می‌شود که برای یافتن یک حالت بهینه برای مسئله زمانبندی کارگاهی استفاده می‌شود.