rl_results

June 9, 2023

[ ]:
```
!pip install statsmodels plotly --quiet
```

```
WARNING: You are using pip version 22.0.4; however, version 23.1.2 is
available.
You should consider upgrading via the
'/Users/danielmiller/.pyenv/versions/3.9.15/envs/torch_rl/bin/python3.9 -m pip
install --upgrade pip' command.
```

[ ]:
```
%jupyter labextension install jupyterlab-plotly@4.14.3
```

```
UsageError: Line magic function `%jupyter` not found.
```

[ ]:
```
USE_GPU = False
from typing import List, Tuple, Dict, Any, Optional
import seaborn as sns
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score,␣
 ↪recall_score, f1_score
import statsmodels.api as sm
import re
import plotly.express as px
import tensorboard
import pandas as pd
import numpy as np
import seaborn as sns
from tensorboard.backend.event_processing.event_accumulator import␣
 ↪EventAccumulator
import matplotlib.pyplot as plt
import os
import glob
from typing import Dict
from statsmodels.tsa.filters.hp_filter import hpfilter
import re
from datetime import datetime
import plotly.offline as pyo
```

```
import plotly.io as pio
```

[ ]:
```
pio.renderers.default = 'notebook+pdf'
# pyo.init_notebook_mode()
```

[ ]:
```
TIME_STATS = [
    'sess_time/ended_time',
    'sess_time/session_minutes',
    'sess_time/time_cutoff',
    'sess_time/time_large',
    'sess_time/time_medium',
    'sess_time/time_small',
]

TIME_STATS_GRANULAR = [
    'ended_time',
    'session_minutes',
    'time_cutoff',
    'time_large',
    'time_medium',
    'time_small'
]

SIZE_STATS_GRANULAR = [
    'ended_event',
    'session_size',
    'size_cutoff',
    'inc_small',
    'inc_medium',
    'inc_large'
]
```

[ ]:
```
def tensorboard_results(log_matrix, scalars):

    log_df = []
    for model, log_dir in log_matrix.items():
        print(f'Getting {model} results')
        events = EventAccumulator(log_dir)
        events.Reload()
        stats_summary_matrix = {}
        for scalar_key in scalars:
            stats = events.Scalars(scalar_key)
            stats_summary = pd.DataFrame({
                'wall_time': [x.wall_time for x in stats],
                scalar_key: [x.value for x in stats],
            })
```

```
            stats_summary[scalar_key] = stats_summary[scalar_key].clip(lower=-1)

            stats_summary['wall_time'] = pd.
 ↪to_datetime(stats_summary['wall_time'], unit='s')
            stats_summary = stats_summary.set_index('wall_time') \
                .resample('1T') \
                .mean() \
                .reset_index() \
                .drop(columns=['wall_time'])

            cycle, trend = sm.tsa.filters.hpfilter(stats_summary[scalar_key],␣
 ↪lamb=100)
            stats_summary_matrix[scalar_key] = trend
        df = pd.DataFrame(stats_summary_matrix)
        df['model'] = model
        df['step'] = df.index.values

        log_df.append(df)
    final_df = pd.concat(log_df, axis=0).reset_index(drop=True)

    return final_df



def plot_vectors(df, y, title):

    fig = px.line(
        df,
        x='step',
        y=y,
        color='model',
        title=f'Training {title}',
    )

    fig.show()

def plot_vectors_multiple(df, y, title):
    fig = px.line(
        df,
        x='step',
        y=y,
        title=f'Training {title}',
    )

    fig.show()
```

```
def df_by_time_window(df, model_name):
    df = df.drop(columns=['Unnamed: 0'])
    df = df.rename(columns={'exp_runs': 'exp_count'})
    df = df.groupby(['exp_count']).mean().reset_index()

    df['model'] = model_name
    df['step'] = df['exp_count']

    print(df.columns)
    df['model'] = model_name
    for col in TIME_STATS_GRANULAR:
        cycle, trend = sm.tsa.filters.hpfilter(df[col], lamb=100)
        df[col] = trend


    return df

def time_stats(df, model):
    df = df.groupby(['exp_runs']).mean().reset_index()
    df['time_stamp'] = pd.date_range(start='2023 06 07 00:00:00', end='2023 06␣
 ↪07 06:00:00', periods=len(df))
    df = df.set_index('time_stamp') \
        .resample('1T') \
        .mean() \
        .reset_index() \
        .drop(columns=['time_stamp'])
    df['model'] =  model
    df['step'] = df.index.values
    return df
```

[ ]:
```
TB_LOGS = 'dqn_tb'
CSV_LOGS = 'rl_results/dqn_csv'
```

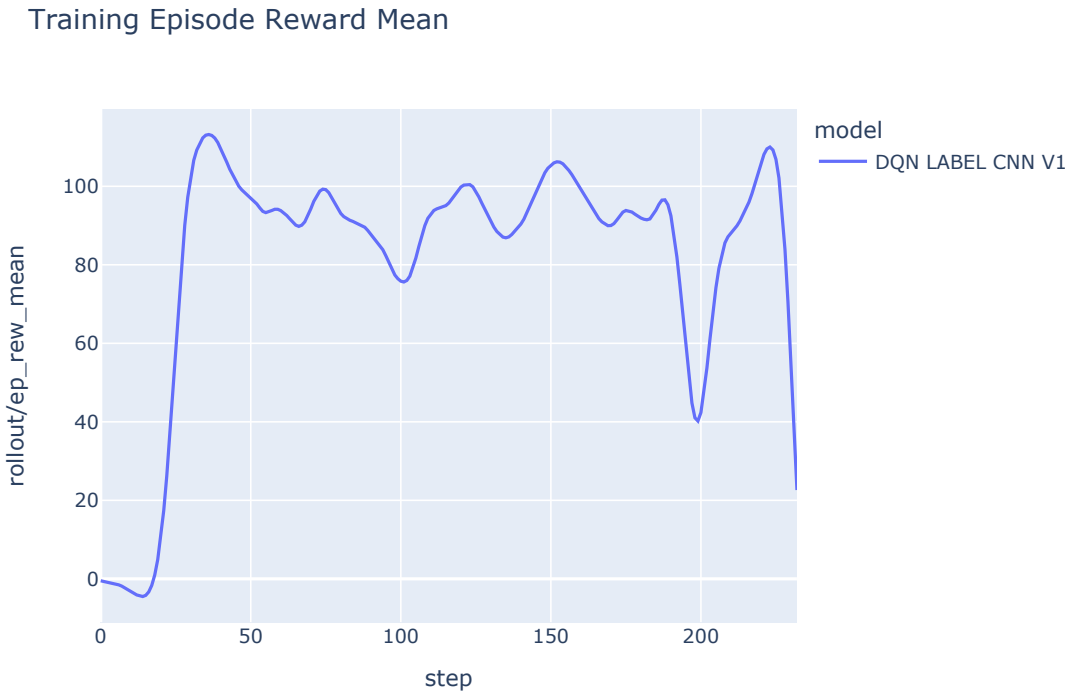
[ ]:
```
log_dirs_q1_dqn = {
    'DQN LABEL CNN V1': 'events.out.tfevents.1686255549.nz7qcy6hwh.395.0'
}

log_dirs_q1 = log_dirs_q1_dqn.copy()
log_dirs_q1 = {
    k: os.path.join(TB_LOGS, v) for k, v in log_dirs_q1.items()
}


results_dir = os.path.join('rl_stats', 'question_1')

if not os.path.exists(results_dir):
    os.makedirs(results_dir)
```

```
for k, v in log_dirs_q1.items():
    print(k, v)
```

```
DQN LABEL CNN V1 dqn_tb/events.out.tfevents.1686255549.nz7qcy6hwh.395.0
```

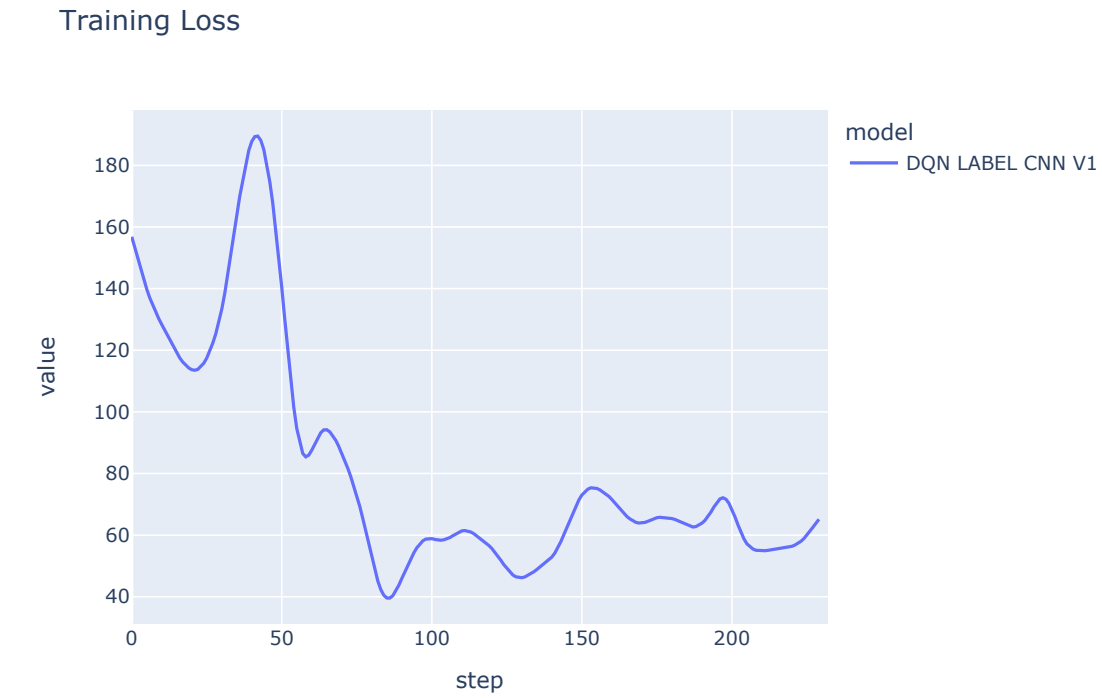
[ ]:
```
training_stats = tensorboard_results(log_dirs_q1, ['rollout/ep_rew_mean',␣
↪'train/loss'])
plot_vectors(training_stats.copy(), 'rollout/ep_rew_mean', 'Episode Reward␣
↪Mean')
```

```
Getting DQN LABEL CNN V1 results
```



Loading [MathJax]/extensions/MathMenu.js

[ ]:
```
plot_vectors(training_stats.copy(), ['train/loss'], 'Loss')
```

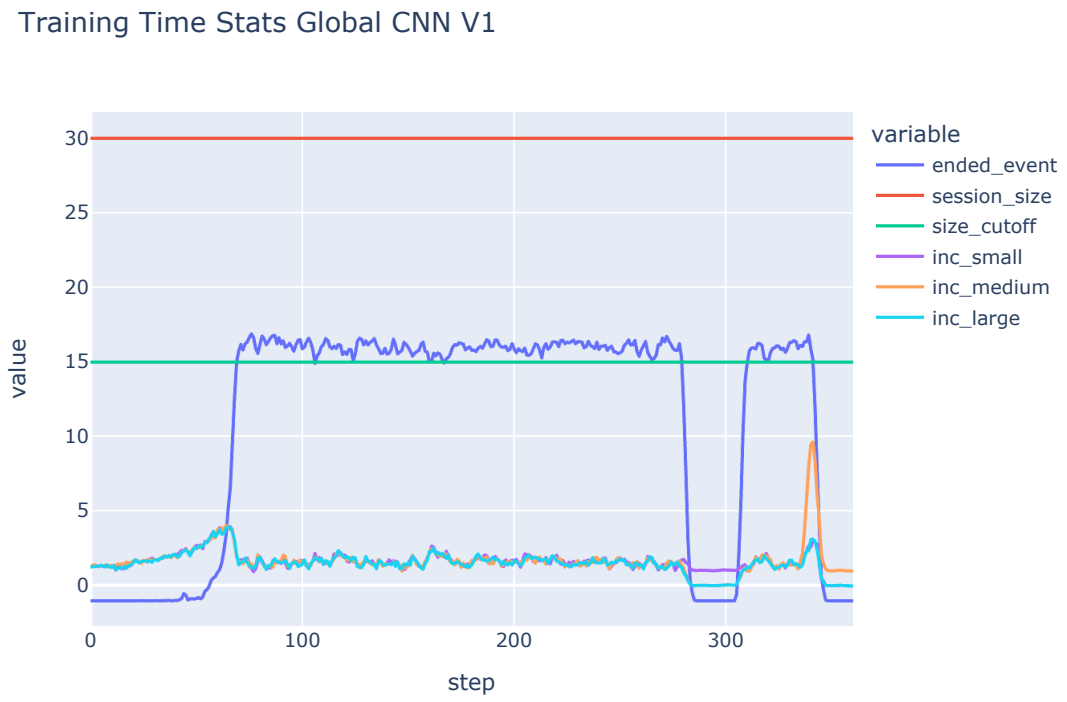


Loading [MathJax]/extensions/MathMenu.js

[ ]:
```
df = pd.read_csv('dqn_csv/exp_dqn_label_cnn_1.csv')
df['session_size'] = 30
df.head()
global_stats = time_stats(df.copy(), 'DQN LABEL CNN V1')

global_stats.head()
global_stats.shape
```

[ ]:
```
(361, 17)
```

[ ]:
```
plot_vectors_multiple(global_stats, SIZE_STATS_GRANULAR, 'Time Stats Global CNN␣
↪V1')
```



[ ]:
```

```

[ ]:
```
df['legal_move'] = df['ended_event'] > 0
df['is_finished'] = df['ended_event'] == 30
```

[ ]:
```
legal_perc = df.groupby(['exp_runs'])['legal_move'].sum().reset_index().
 ↪rename(columns={'legal_move': 'legal_move_count'})
size_runs = df.groupby(['exp_runs']).size().reset_index().rename(columns={0:␣
 ↪'exp_count'})
is_finished = df.groupby(['exp_runs'])['is_finished'].sum().reset_index().
 ↪rename(columns={'is_finished': 'is_finished_count'})
```

[ ]:
```
joined_stats = legal_perc.merge(size_runs, on='exp_runs').merge(is_finished,␣
 ↪on='exp_runs')
```

[ ]:
```
joined_stats['legal_move_perc'] = joined_stats['legal_move_count'] /␣
 ↪joined_stats['exp_count']
joined_stats['is_finished_perc'] = joined_stats['is_finished_count'] /␣
 ↪joined_stats['exp_count']
```

[ ]:
```
fig = px.line(
    joined_stats,
    x='exp_runs',
    y=['legal_move_perc', 'is_finished_perc'],
    title='Legal Move and Finished Percentage',
)

fig.show()
```