prototype

November 23, 2025

1 Import and config

Using this cell to import all the necessary libraries

[1]:
```
# %pip install matplotlib
```

[2]:
```
import yaml
import pandas as pd
import matplotlib.pyplot as plt

from pipeline.build_dataset import make_state_frame
from features.state_assembler import StateAssembler
from envs.buy_env import BuyEnv
from agents.buy_agent_ddqn import BuyAgentTrainer
```

```
Gym has been unmaintained since 2022 and does not support NumPy 2.0 amongst
other critical functionality.
Please upgrade to Gymnasium, the maintained drop-in replacement of Gym, or
contact the authors of your software and request that they upgrade.
Users of this version of Gym should be able to simply replace 'import gym' with
'import gymnasium as gym' in the vast majority of cases.
See the migration guide at
https://gymnasium.farama.org/introduction/migration_guide/ for additional
information.
```

2 Load config and data

[3]:
```
cfg = yaml.safe_load(open("config/data_config.yaml"))

ticker = "AAPL"
dataset = make_state_frame(ticker, cfg)
print("Raw dataset shape:", dataset.shape)
print(dataset.tail())
```

```
[**********************100%**********************]  1 of 1 completed

Raw dataset shape: (1224, 10)
            return_1d     rsi14  macd_diff      bb_b     atr14     roc10  \
Date
```

```
2023-12-22  -0.182773  0.089437  -0.627491 -0.178014 -1.820035 -0.282916
2023-12-26  -0.538524 -0.175613  -0.849465 -0.533543 -1.984075 -0.573011
2023-12-27  -0.341273 -0.314185  -1.030830 -0.759738 -2.209548 -0.357041
2023-12-28  -0.084725 -0.299801  -1.123309 -0.790062 -2.209548 -0.531208
2023-12-29   0.041502 -0.208808  -1.127636 -0.746857 -2.209548 -0.867669

                obv     mfi14   willr14       price
Date
2023-12-22  1.349133 -0.061185 -0.080590  191.788773
2023-12-26  1.259801 -0.044919 -0.988173  191.243912
2023-12-27  1.190502 -0.392735 -1.621949  191.342987
2023-12-28  1.300912 -0.723739 -1.453421  191.768951
2023-12-29  1.382170 -0.785871 -1.292257  190.728745
```

3 Building rolling states

[4]:
```
feature_cols = [c for c in dataset.columns if c != "price"]
assembler = StateAssembler(feature_cols=feature_cols, window_size=30)

state_df = assembler.assemble(dataset)
prices = dataset["price"].iloc[30:]  # align with state_df

print("state_df shape:", state_df.shape)
print("prices shape:", prices.shape)
print("NaNs in state_df:", state_df.isna().sum().sum())
print("NaNs in prices:", prices.isna().sum())
```

```
state_df shape: (1194, 270)
prices shape: (1194,)
NaNs in state_df: 0
NaNs in prices: 0
```

4 Train DDQN Buy Agent

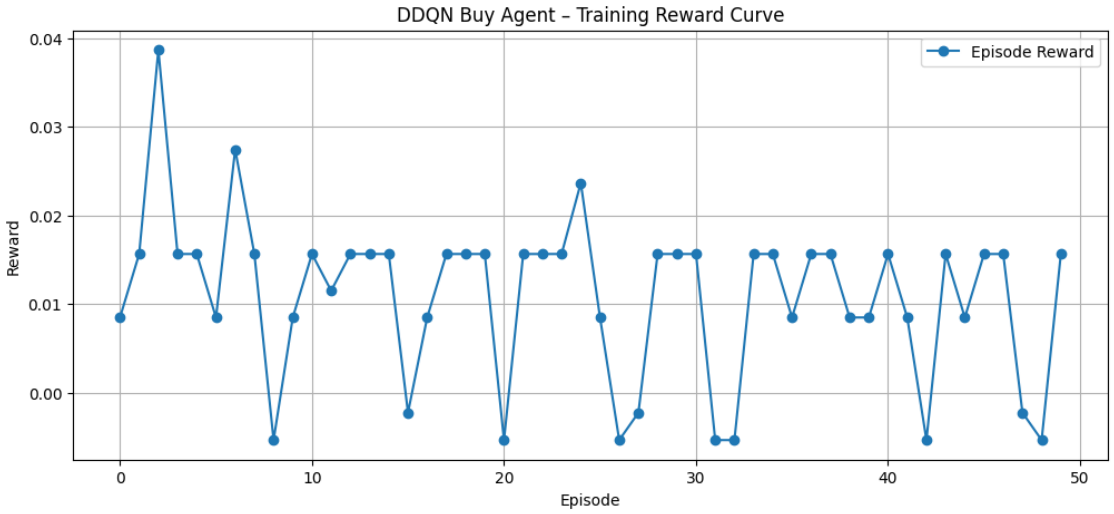
[5]:
```
trainer = BuyAgentTrainer(
        cfg_path="config/data_config.yaml",
        ticker="AAPL",
        window_size=30,
        horizon=5,
        transaction_cost=0.001,
    )

rewards = trainer.train(
    n_episodes=50,
    warmup_steps=500,
```

```
    max_steps_per_episode=None,
)

print("Last 10 episode rewards:", rewards[-10:])

greedy_policy = trainer.make_greedy_policy()
total_reward, steps = greedy_policy()
print(f"Greedy run: total_reward={total_reward:.4f}, steps={steps}")
```

```
[**********************100%**********************]  1 of 1 completed

[BuyTrainer] Raw dataset shape for AAPL: (1224, 10)
[BuyTrainer] After dropna: (1224, 10)
[BuyTrainer] Rolling state_df shape: (1194, 270)
[BuyTrainer] state_dim=270, n_actions=2
[Episode 1/50] Reward=0.0085, Epsilon=1.000
[Episode 10/50] Reward=0.0085, Epsilon=0.994
[Episode 20/50] Reward=0.0157, Epsilon=0.991
[Episode 30/50] Reward=0.0157, Epsilon=0.987
[Episode 40/50] Reward=0.0085, Epsilon=0.983
[Episode 50/50] Reward=0.0157, Epsilon=0.980
Training complete.
Final rewards: [0.015659015625715256, 0.008492113091051579,
-0.0053479536436498165, 0.015659015625715256, 0.008492113091051579,
0.015659015625715256, 0.015659015625715256, -0.0022531943395733833,
-0.0053479536436498165, 0.015659015625715256]
Last 10 episode rewards: [0.015659015625715256, 0.008492113091051579,
-0.0053479536436498165, 0.015659015625715256, 0.008492113091051579,
0.015659015625715256, 0.015659015625715256, -0.0022531943395733833,
-0.0053479536436498165, 0.015659015625715256]
Greedy run: total_reward=0.0157, steps=1
```

5 Plot training reward curve

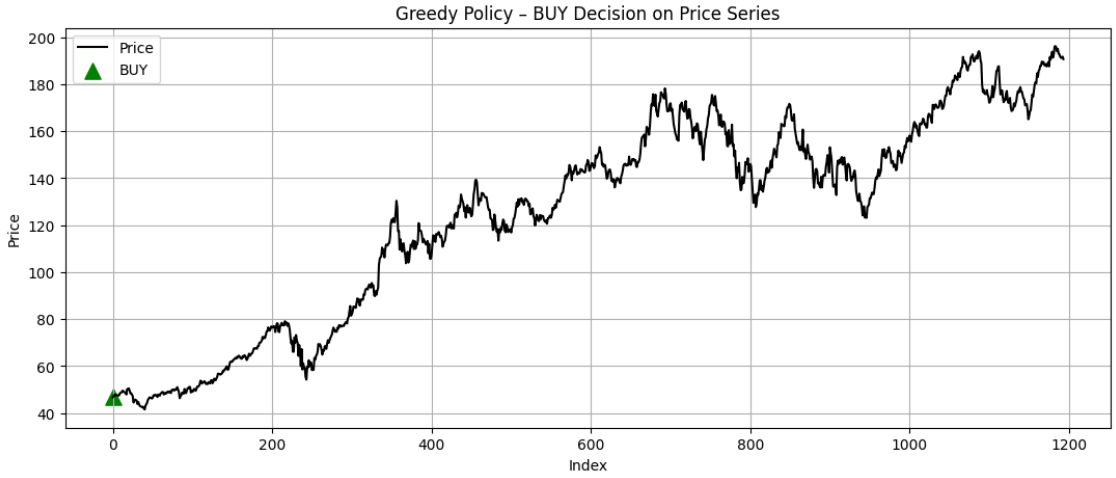
[6]:
```
plt.figure(figsize=(12,5))
plt.plot(rewards, label="Episode Reward", marker="o")
plt.title("DDQN Buy Agent - Training Reward Curve")
plt.xlabel("Episode")
plt.ylabel("Reward")
plt.grid(True)
plt.legend()
plt.show()
```



6 Buy signal

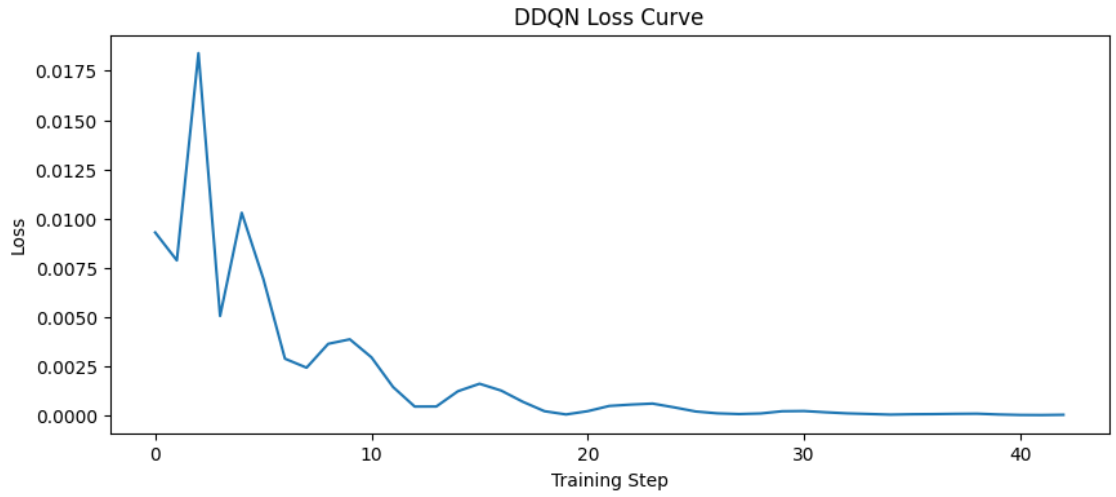
[7]:
```
# Get environment price series
prices = trainer.prices.copy()
index = prices.index

# Re-run greedy to get signal points
env = trainer.env
state = env.reset()
done = False
buy_index = None

while not done:
    a = trainer.agent.select_action(state, greedy=True)
    if a == 1:   # BUY
        buy_index = env.idx
    state, r, done, info = env.step(a)

plt.figure(figsize=(13,5))
plt.plot(prices.values, label="Price", color="black")
if buy_index is not None:
    plt.scatter([buy_index], [prices.iloc[buy_index]], s=120, color="green",␣
 ↪label="BUY", marker="^")
plt.title("Greedy Policy - BUY Decision on Price Series")
plt.xlabel("Index")
plt.ylabel("Price")
plt.legend()
plt.grid(True)
```

```
plt.show()
```



7 Loss history

[8]:
```
plt.figure(figsize=(10,4))
plt.plot(trainer.agent.loss_history)
plt.title("DDQN Loss Curve")
plt.xlabel("Training Step")
plt.ylabel("Loss")
plt.show()
```



8 Inspecting one simple state

[9]:
```
sample = trainer.state_df.iloc[0]
print("Example state vector:", sample.values[:20], "... (total length:",␣
 ↪len(sample), ")")
```

```
Example state vector: [ 0.32262402  0.66738896  0.30020807  0.49006653
-1.91650102 -0.20492171
  0.01857323  1.41689004  0.19677356 -0.30519106  0.49233849  0.15419442
  0.33882595 -1.98984126 -0.29584722 -0.07254095  1.0053385  -0.36132825
  0.56566652  0.70682841] … (total length: 270 )
```