딥러닝과 강화학습을 활용한 주식예측

TEAM MEMBERS

김진영
강민
전종현

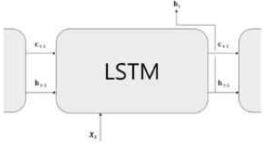
딥 러닝

LSTM

-순환 신경망의 일종으로 이전 상태들을 더 잘 기억할 수 있게 개선된 신경망

-LSTM 유닛에는 망각 게이트, 입력 게이트, 출력게이트가 존재

-sigmoid 함수를 통해 값을 정함



c : 셀 상태
i : 은닉 상태

딥 러닝

LSTM

```python
class LSTMModule(torch.nn.LSTM):
    def __init__(self, *args, use_last_only=False, **kwargs):
        super().__init__(*args, **kwargs)
        self.use_last_only = use_last_only

    def forward(self, x):
        output, (h_n, _) = super().forward(x)
        if self.use_last_only:
            return h_n[-1]
        return output
```

PyTorch의 LSTM 클래스를 상속한 하위 클래스

```python
@staticmethod
def get_network_head(inp, output_dim):
    return torch.nn.Sequential(
        torch.nn.BatchNorm1d(inp[0]),
        LSTMModule(inp[1], 128, batch_first=True, use_last_only=True),
        torch.nn.BatchNorm1d(128),
        torch.nn.Dropout(p=0.1),
        torch.nn.Linear(128, 64),
        torch.nn.BatchNorm1d(64),
        torch.nn.Dropout(p=0.1),
        torch.nn.Linear(64, 32),
        torch.nn.BatchNorm1d(32),
        torch.nn.Dropout(p=0.1),
        torch.nn.Linear(32, output_dim),
    )
```

foward() 메소드의 출력은 LSTM 모든 은닉 상태들을 포함
마지막 은닉 상태만 필요하여
foward를 오버 라이딩

LSTM 신경망 구성

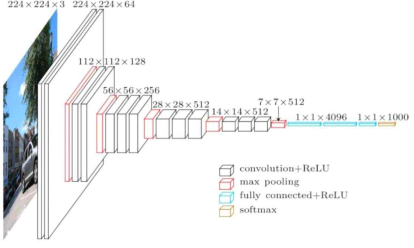
딥 러닝

CNN

- 이미지 데이터 처리에 주로 사용되는 심층 신경망

- 컨볼루션 계층과 풀링 계층을 이용해 입력 데이터의 사이즈를 줄이면서도 복잡한 특징을 추출

- 입력 계층 다음에 컨볼루션, 렐루, 풀링 계층이 처리되고, 이어서 완전 연결 계층이 이어지고 최종적으로 소프트맥스 계층을 거쳐서 결과 출력



딥 러닝

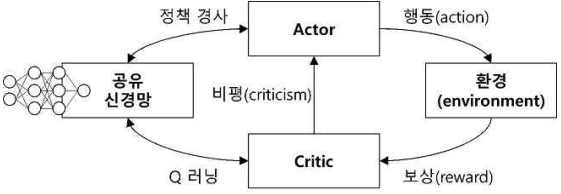
CNN

```python
@staticmethod
def get_network_head(inp, output_dim):
    kernel_size = 2
    return torch.nn.Sequential(
        torch.nn.BatchNorm1d(inp[0]),
        torch.nn.Conv1d(inp[0], 1, kernel_size),
        torch.nn.BatchNorm1d(1),
        torch.nn.Flatten(),
        torch.nn.Dropout(p=0.1),
        torch.nn.Linear(inp[1] - (kernel_size - 1), 128),
        torch.nn.BatchNorm1d(128),
        torch.nn.Dropout(p=0.1),
        torch.nn.Linear(128, 64),
        torch.nn.BatchNorm1d(64),
        torch.nn.Dropout(p=0.1),
        torch.nn.Linear(64, 32),
        torch.nn.BatchNorm1d(32),
        torch.nn.Dropout(p=0.1),
        torch.nn.Linear(32, output_dim),
    )
```

CNN 신경망 구성

강화 학습

Actor-Critic
- Q-러닝 과 정책 경사의 하이브리드 강화학습 모델
- actor는 정책 경사 모델을 사용해 수행할 행동을 결정
- 크리틱은 수행한 행동을 비평하기 위해 Q-러닝 모델을 사용
- 정책 신경망을 학습하면서 동시에 Q-러닝으로 상태,행동 가치 신경망을 학습



강화 학습

ActorCriticLearner

```
class ActorCriticLearner(ReinforcementLearner):
    def __init__(self, *args, shared_network=None,
                 value_network_path=None, policy_network_path=None, **kwargs):
        super().__init__(*args, **kwargs)
        if shared_network is None:
            self.shared_network = Network.get_shared_network(
                net=self.net, num_steps=self.num_steps,
                input_dim=self.num_features,
                output_dim=self.agent.NUM_ACTIONS)
        else:
            self.shared_network = shared_network
        self.value_network_path = value_network_path
        self.policy_network_path = policy_network_path
        if self.value_network is None:
            self.init_value_network(shared_network=self.shared_network)
        if self.policy_network is None:
            self.init_policy_network(shared_network=self.shared_network)
```

A2C의 부모 클래스 ActorCritic클래스 생성
가치 신경망과 정책 신경망 모두 생성,

강화 학습

ActorCriticLearner

```
def get_batch(self):
    memory = zip(
        reversed(self.memory_sample),
        reversed(self.memory_action),
        reversed(self.memory_value),
        reversed(self.memory_policy),
        reversed(self.memory_reward),
    )
    x = np.zeros((len(self.memory_sample), self.num_steps, self.num_features))
    y_value = np.zeros((len(self.memory_sample), self.agent.NUM_ACTIONS))
    y_policy = np.zeros((len(self.memory_sample), self.agent.NUM_ACTIONS))
    value_max_next = 0
    for i, (sample, action, value, policy, reward) in enumerate(memory):
        x[i] = sample
        r = self.memory_reward[-1] - reward
        y_value[i, :] = value
        y_value[i, action] = r + self.discount_factor * value_max_next
        y_policy[i, :] = policy
        y_policy[i, action] = utils.sigmoid(r)
        value_max_next = value.max()
    return x, y_value, y_policy
```

y_value: 가치 신경망 학습 레이블
y_policy: 정책 신경망 학습 레이블

get_batch() 메소드는 샘플 배열, 가치 신경망 학습 레이블 배열, 정책 신경망 학습 레이블 배열을 반환

강화 학습

A2C(advantage actor-critic)

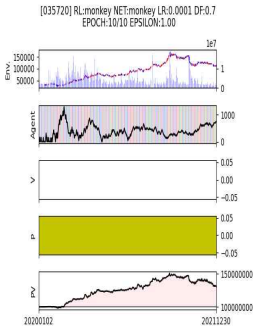
```
class A2CLearner(ActorCriticLearner):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    def get_batch(self):
        memory = zip(
            reversed(self.memory_sample),
            reversed(self.memory_action),
            reversed(self.memory_value),
            reversed(self.memory_policy),
            reversed(self.memory_reward),
        )
        x = np.zeros((len(self.memory_sample), self.num_steps, self.num_features))
        y_value = np.zeros((len(self.memory_sample), self.agent.NUM_ACTIONS))
        y_policy = np.zeros((len(self.memory_sample), self.agent.NUM_ACTIONS))
        value_max_next = 0
        reward_next = self.memory_reward[-1]
        for i, (sample, action, value, policy, reward) in enumerate(memory):
            x[i] = sample
            r = reward_next + self.memory_reward[-1] - reward * 2
            reward_next = reward
            y_value[i, :] = value
            y_value[i, action] = np.tanh(r + self.discount_factor * value_max_next)
            advantage = y_value[i, action] - y_value[i].mean()
            y_policy[i, :] = policy
            y_policy[i, action] = utils.sigmoid(advantage)
            value_max_next = value.max()
        return x, y_value, y_policy
```

actor-critic 방법에서 기대 출력을
Advantaage 로 사용하는 방법

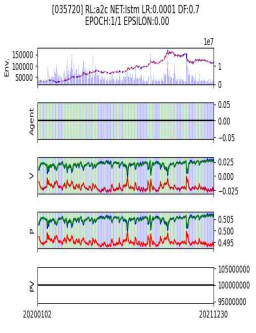
critic을 advantage로 학습
-> 어떠한 상태에서 행동이 얼마나 좋은지뿐만
아니라 얼마나 더 좋아지는지학습가능
Advantage=(상태,행동가치) - (상태 가치)
sigmodi 함수에 적용해 정책 신경망의
학습 레이블로 적용

테스트 결과

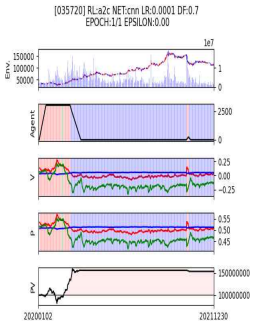
< 원숭이 투자 >



< LSTM >



< CNN >



같은 날짜 데이터로 학습후 테스트 한 결과
랜덤하게 매매하는 원숭이 투자에 비해 CNN이 더 좋은 결과를 보이고 있고
LSTM은 좋지 못한 결과를 보이고 있다

1주차 (5/23 ~ 5/27)

- [ ]

프로젝트 계획 및 수정

2주차 (5/30 ~ 6/5)

[화면흐름도]

1. Home
2. 회원가입 / 로그인
3. 자동거래 + 차트
4. 매매현황
5. 주문현황
6. 수익현황
7. 차트 (매매내역 누르면)

[업무 분담]

<강민>

- [x] ~~홈 / Next Routing / Footer 컴포넌트를 작성한다.~~
- [x] ~~매매현황 컴포넌트를 작성한다.~~
- [ ] 주문현황 컴포넌트를 작성한다.
- [ ] 수익현황 컴포넌트를 작성한다.
- [x] ~~비트코인 / 도지코인 / 마확정 코인을 확정한다.~~

<김진영>

- [ ] 사이트 홈 화면의 인트로 부분을 작성한다.
- [ ] 비트코인 Chart API 를 가져오기를 구현한다.
- [ ] 자동매매 설정환경을 분석한다.

<전종현>

- [x] ~~로고를 만든다.~~
- [x] ~~로그인 / 회원가입 컴포넌트를 작성한다.~~
- [x] ~~자동매매폼 컴포넌트를 작성한다.~~

3주차 (6/6 ~ 6/10)

<강민>

- [x] ~~모든화면 TS변환~~
- [x] ~~홈화면 배경이미지 구현~~
- [x] ~~홈화면 이미지클릭시 해당route가기 적용~~
- [x] ~~공지사항 페이지 작성~~
- [x] ~~FAQ페이지 작성~~
- [x] ~~소개 페이지 작성~~
- [x] ~~주문현황 컴포넌트를 작성한다.~~
- [x] ~~수익현황 컴포넌트를 작성한다.~~

<김진영>

<전종현>

- [x] ~~ERD 테이블 구상~~
- [x] ~~ERD 용어 정리~~
- [x] ~~ERD 관계 연결~~
- [x] ~~ERD 속성값 정리~~
- [x] ~~ERD 기반 도메인 작성~~
- [x] ~~더미데이터 연결~~

4주차 (6/13 ~ 6/17)

전종현

- [x] ~~프로젝트 주간보고서 작성~~
- [x] ~~알고리즘 문제 풀이(hash, greedy, graph)~~
- [x] ~~TS&SpringBoot 연결~~

김진영

- [ ] **TensorFlow** 예제를 통한 모델 구성 연습
- [ ] titanic
- [ ] iris

5주차 (6/20 ~ 6/24)

전종현

- [x] ~~알고리즘 문제 풀이(hash, greedy, graph)~~
- [x] ~~프로젝트 주간보고서 작성~~
- [x] ~~TS&SpringBoot 연결~~
- [x] ~~깃클론~~

김진영

- [ ] **GPU** 사용을 위한 환경설정
- [ ] 문제 확인후 에러 해결

6주차 (6/27 ~ 7/1)

전종현

- [x] ~~알고리즘 문제 풀이(hash, greedy, graph)~~
- [x] ~~프로젝트 주간보고서 작성~~
- [x] ~~token생성~~
- [x] ~~User기본쿼리 메소드 작성~~

김진영

- [ ] **UpbitAPI** 활용하여 불러온 데이터 전처리
- [ ] **fbProphet** 분석 및 사용

7주차 (7/4 ~ 7/8)

전종현

- [x] ~~알고리즘 문제 풀이(hash, greedy, graph)~~
- [x] ~~프로젝트 주간보고서 작성~~
- [x] ~~User기본쿼리 메소드 작성, 테스트~~
- [x] ~~User domain 회원정보 추가~~

김진영

- [ ] **UpbitAPI** 활용하여 불러온 데이터 전처리

- [ ] **fbProphet** 분석 및 사용

8주차 (7/11 ~ 7/15)
김진영
- [ ] LSTM 분석 및 사용

9주차 (7/18 ~ 7/22)
김진영
- [ ] Candle Chart 이미지 변환 및 CNN을 이용한 코인 예측
- [ ] 객체지향 프로그래밍

10주차 (7/25 ~ 7/29)
김진영
- [ ] LSTM 을 활용한 코인 예측 객체지향프로그래밍

11주차 (8/1 ~ 8/5)
김진영
- [ ] 사용할 모델 정하기
- [ ] 발표 자료 준비하기