KAIST Include 동아리 스터디

AlphaGo와 AlphaGo Zero를 만들며 익히는 딥러닝 및 강화학습

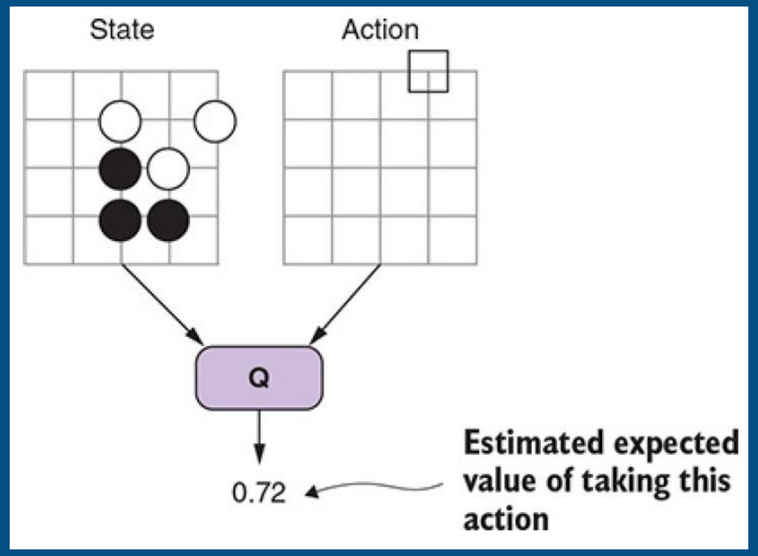
Chris Ohk
utilForever@gmail.com

Q-학습을 사용한 대국

- **행동-가치 함수(Action-Value Function)**
  - 특정 수를 둔 후 종국에 이길 확률이 얼마인지 알려주는 함수
    (즉, 특정 행동의 가치가 얼마나 되는지를 알려주는 함수)
- **Q-학습(Q-Learning)**
  - 강화학습으로 행동-가치 함수를 훈련하는 기법
  - 물론 바둑의 수에 대한 실제 행동-가치 함수가 어떻게 생겼는지는 알 수 없다.
    (이를 이해하려면 무수한 확률값을 사용하는 전체 게임 트리를 알아야 한다.)
  - 하지만 자체 대국을 진행하면서 행동-가치 함수의 **추정치**를 반복적으로 향상시킬 수 있다.
    이 과정을 거치면서 추정치는 더 정확해지고, 추정치를 사용하는 봇은 더 강해질 것이다.

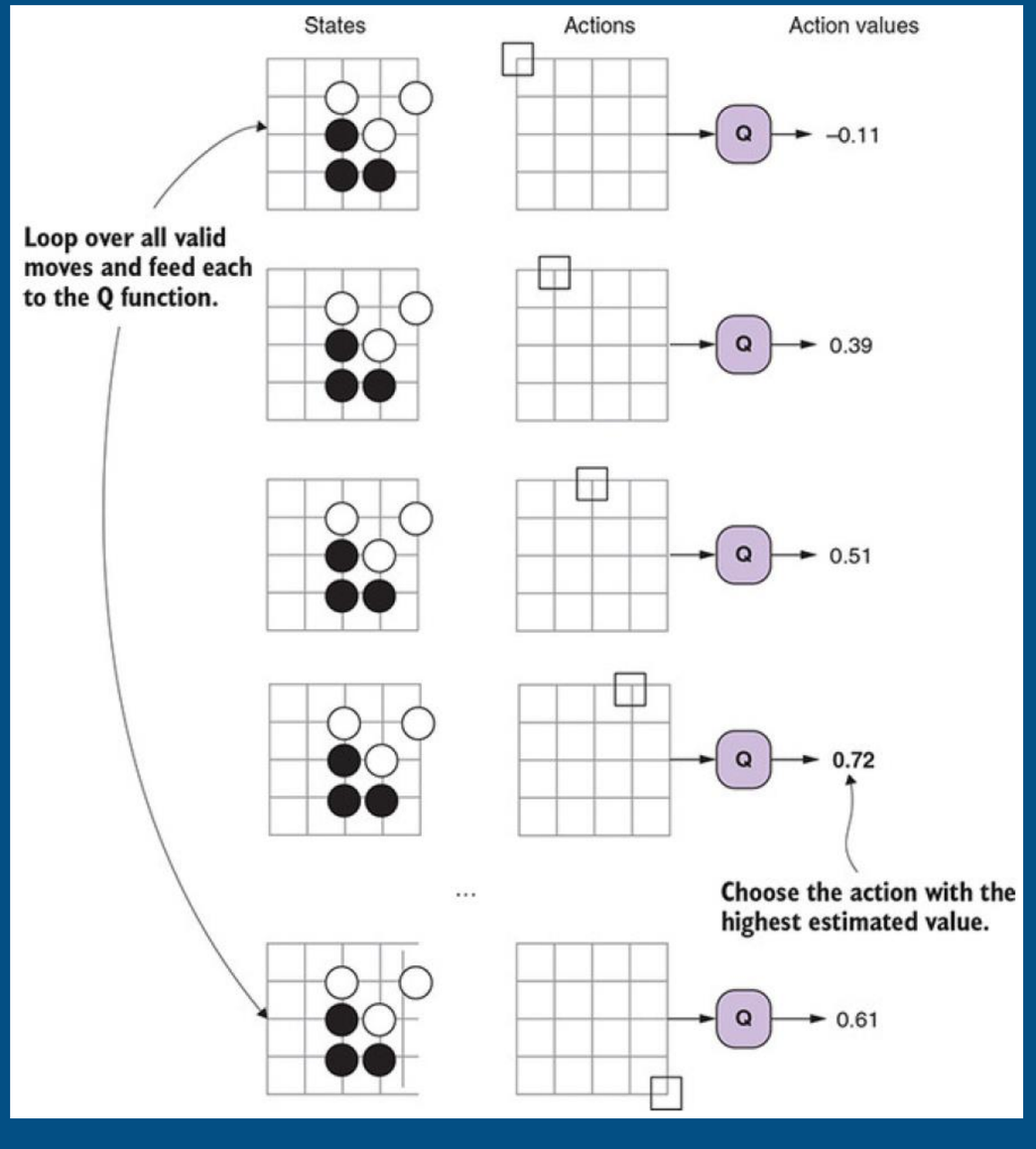
Q-학습을 사용한 대국

- **Q-학습(Q-Learning)**
  - 행동-가치 함수는 전통적으로 $Q(s, a)$라는 기호를 사용한다.
    - s는 에이전트가 접하는 상태(바둑판의 상태 등)를 말한다.
    - a는 에이전트가 고려하는 행동(다음에 둘만한 수)를 말한다.
  - 신경망을 사용해서 Q 함수를 추정하는 걸 **심층 Q-학습(Deep Q-Learning)**이라 한다.



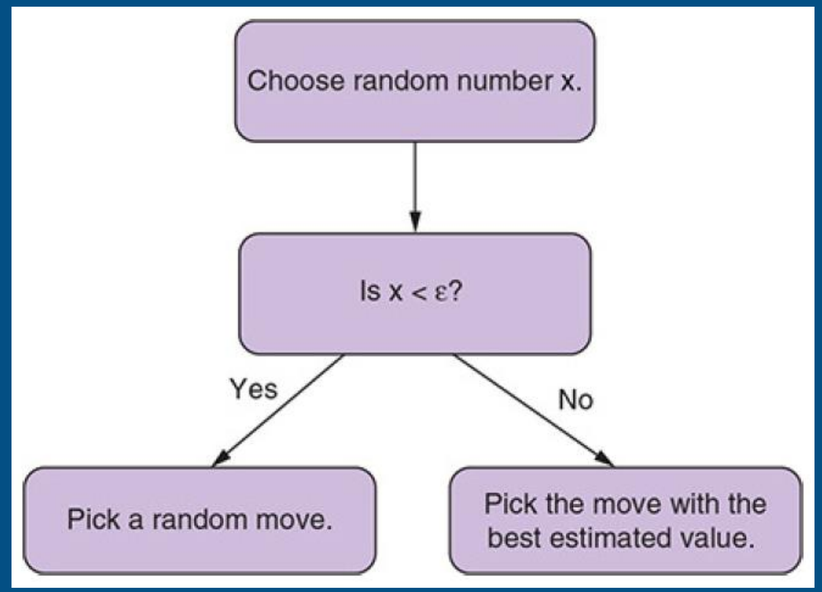
Q-학습을 사용한 대국

- Q-학습의 구조
  - 우선 자체 대국을 두는 에이전트를 만들고, 모든 의사 결정과 경기 결과를 저장한다.
  - 경기 결과를 보면 의사 결정이 잘 되었는지를 알 수 있고, 에이전트의 행동을 갱신할 수 있다.
  - Q 함수를 사용하는 바둑 대국 에이전트를 만들려면 정책에 Q 함수를 적용해야 한다.
  - 모든 가능한 수를 Q 함수에 연결한 후 **탐욕(Greedy)** 정책을 통해 가장 높은 기대 수익으로 수를 선택한다.



Q-학습을 사용한 대국

- 탐욕 정책
  - 행동값 추정이 믿을만하다면 탐욕 정책만한 것이 없다. 하지만 추정을 향상시켜야 한다면 봇이 간헐적으로 미지의 값을 탐색하도록 해야 한다. 이를 **ε-탐욕 정책**이라고 한다.
  - 일부 시간 동안 정책이 수를 완전히 임의로 선택하도록 하고,
    나머지 시간에는 일반 탐욕 정책을 실행한다.



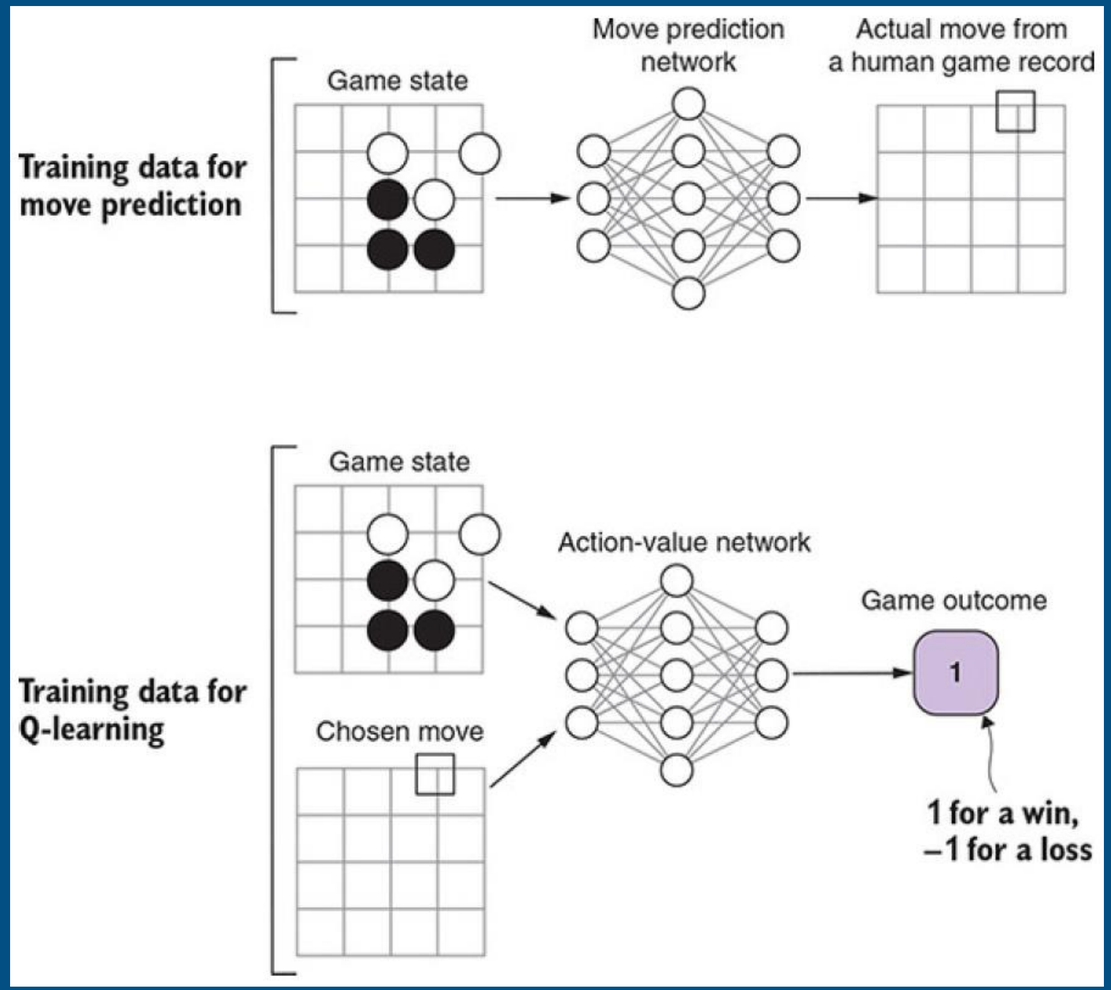
Q-학습을 사용한 대국

- ε-탐욕 정책 의사 코드

```
def select_action(state, epsilon):
  possible_actions = get_possible_actions(state)
  if random.random() < epsilon:
    return random.choice(possible_actions)
  best_action = None
  best_value = MIN_VALUE
  for action in get_possible_actions(state):
    action_value = self.estimate_action_value(state, action)
    if action_value > best_value:
      best_action = action
      best_value = action_value
  return best_action
```

Q-학습을 사용한 대국

- 심층 Q-학습용 훈련 데이터 만들기



TensorFlow 2로 Q-학습 만들기

- 입력값이 둘인 신경망 만들기
  - TensorFlow 2 순차형 API를 사용한 모델 정의

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(32, input_shape=(19, 19)))
model.add(Dense(24))
```

TensorFlow 2로 Q-학습 만들기

- 입력값이 둘인 신경망 만들기
  - TensorFlow 2 함수형 API를 동일한 모델 정의하기

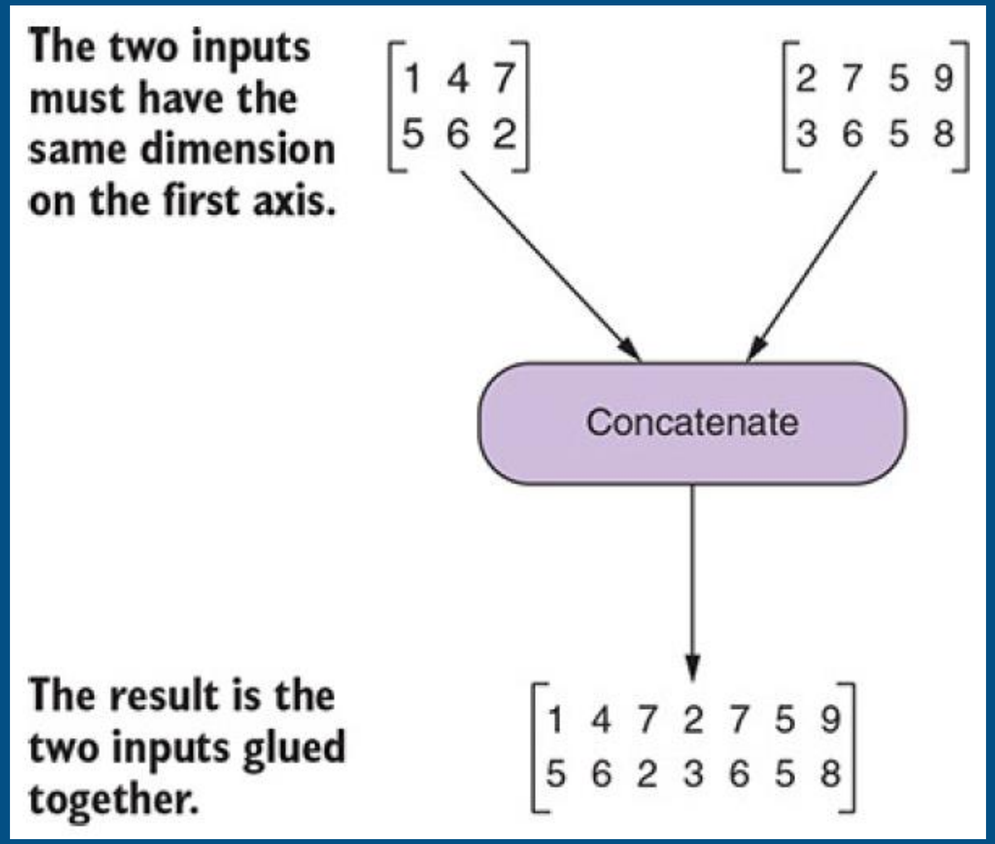
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Input

model_input = Input(shape=(19, 19))
hidden_layer = Dense(32)(model_input)
output_layer = Dense(24)(hidden_layer)

model = Model(inputs=[model_input], outputs=[output_layer])
```

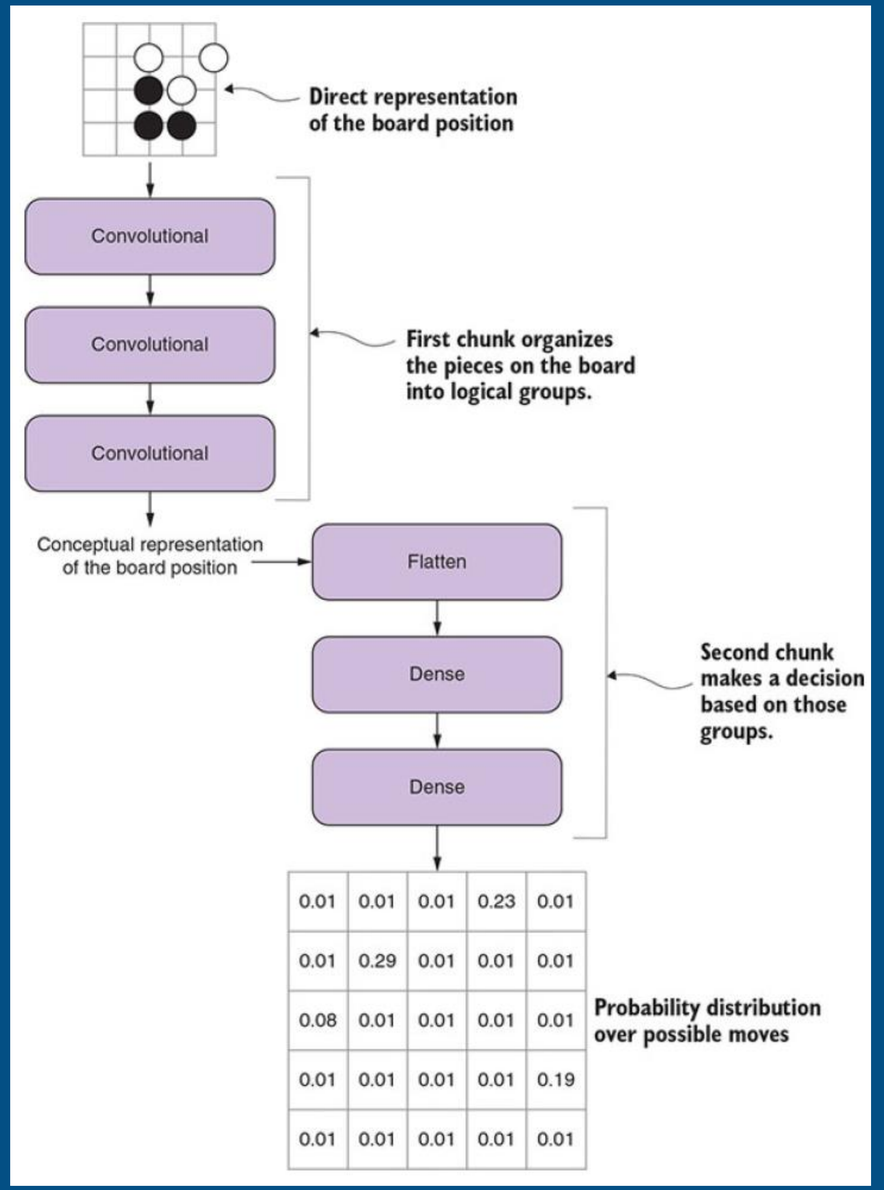
TensorFlow 2로 Q-학습 만들기

- 입력값이 둘인 신경망 만들기
  - TensorFlow 2의 `Concatenate` 층은 두 텐서를 하나로 이어 붙인다.



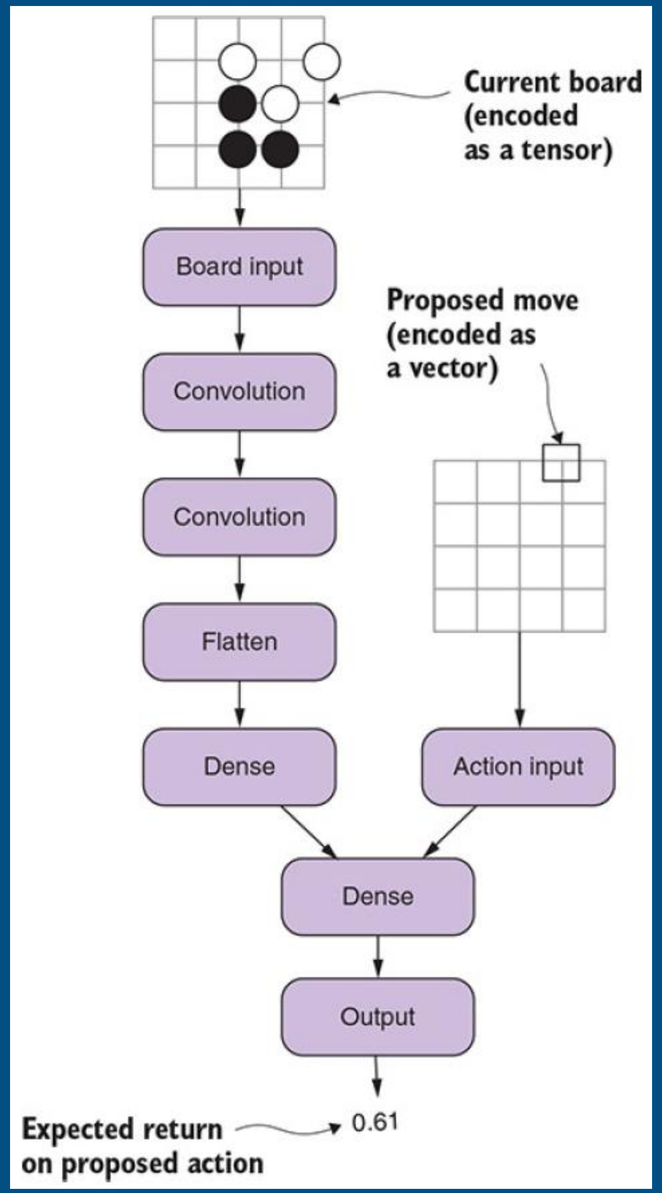
TensorFlow 2로 Q-학습 만들기

- 입력값이 둘인 신경망 만들기
  - 이전에 다뤘던 수 예측 신경망
    - 우선 합성곱층에서 바둑판의 주요 수 모양을 정의한다.
    - 그 후 밀집층이 이 모양을 기반으로 판단을 내린다.



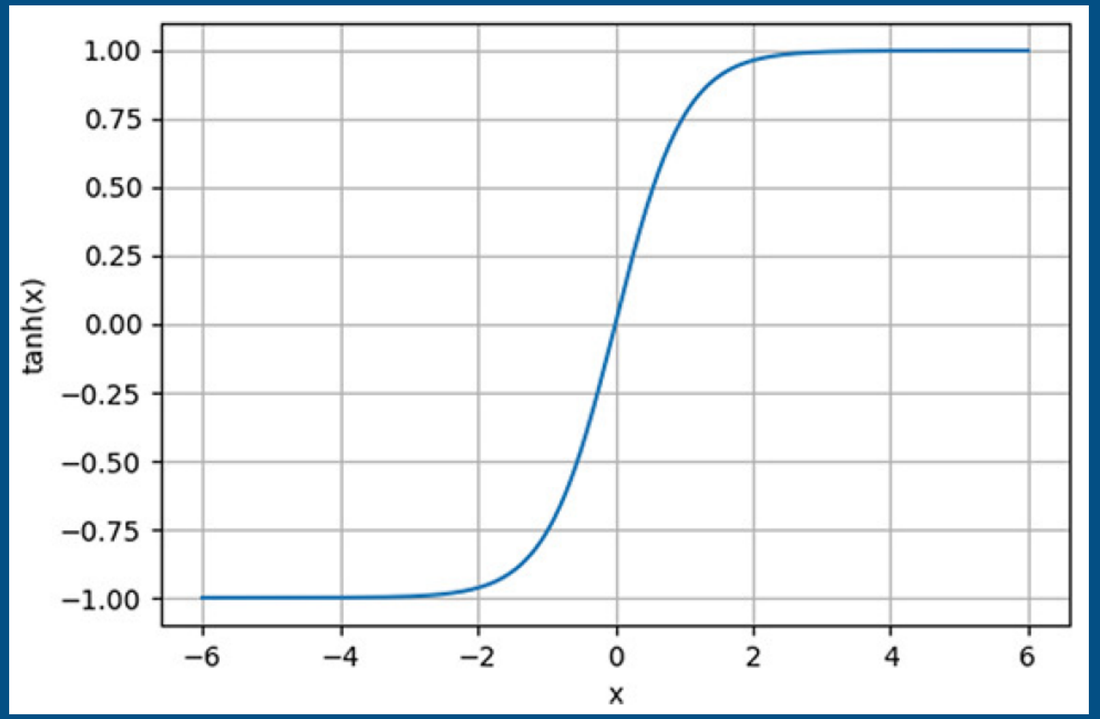
TensorFlow 2로 Q-학습 만들기

- 입력값이 둘인 신경망 만들기
  - 두 입력값을 사용하는 신경망
    - 정의된 돌의 집 기반으로 판단하는 대신 바둑판과 제안된 행동을 기반으로 가치를 추정한다. 그러므로 합성곱층 후 제안된 수 벡터를 갖고 와야 한다.



TensorFlow 2로 Q-학습 만들기

- 입력값이 둘인 신경망 만들기
  - tanh 함수
    - 지면 -1을, 이기면 1을 사용하므로 행동-가치는 -1과 1 사이의 단일값이다.
    - 이 작업을 수행하려면 크기가 1인 Dense층을 만들고 tanh 활성화 함수를 넣는다.



TensorFlow 2로 Q-학습 만들기

- 입력값이 둘인 신경망 만들기
  - 두 입력값을 갖는 행동-가치 신경망

```
from tensorflow.keras.layers import Conv2D, Dense, Flatten, Input
from tensorflow.keras.layers import ZeroPadding2D, concatenate
from tensorflow.keras.models import Model

board_input = Input(shape=encoder.shape(), name='board_input')
action_input = Input(shape=(encoder.num_points(),), name='action_input')

conv1a = ZeroPadding2D((2, 2))(board_input)
conv1b = Conv2D(64, (5, 5), activation='relu')(conv1a)

conv2a = ZeroPadding2D((1, 1))(conv1b)
conv2b = Conv2D(64, (3, 3), activation='relu')(conv2a)

flat = Flatten()(conv2b)
processed_board = Dense(512)(flat)

board_plus_action = concatenate([action_input, processed_board])
hidden_layer = Dense(256, activation='relu')(board_plus_action)
value_output = Dense(1, activation='tanh')(hidden_layer)

model = Model(inputs=[board_input, action_input], outputs=value_output)
```

TensorFlow 2로 Q-학습 만들기

- ε-탐욕 정책 구현하기
  - Q-학습 에이전트 생성자 및 유틸리티 메소드

```
class QAgent(Agent):
  def __init__(self, model, encoder):
    self.model = model
    self.encoder = encoder
    self.collector = None
    self.temperature = 0.0

  def set_temperature(self, temperature):
    self.temperature = temperature

  def set_collector(self, collector):
    self.collector = collector
```

TensorFlow 2로 Q-학습 만들기

- ε-탐욕 정책 구현하기
  - Q-학습 에이전트에서 수 선택하기

```
class QAgent(Agent):
  def select_move(self, game_state):
    board_tensor = self.encoder.encode(game_state)

    moves = []
    board_tensors = []
    for move in game_state.legal_moves():
      if not move.is_play:
        continue
      moves.append(self.encoder.encode_point(move.point))
      board_tensors.append(board_tensor)
    if not moves:
      return goboard.Move.pass_turn()

    num_moves = len(moves)
    board_tensors = np.array(board_tensors)
    move_vectors = np.zeros(
      (num_moves, self.encoder.num_points()))
    for i, move in enumerate(moves):
      move_vectors[i][move] = 1
```

TensorFlow 2로 Q-학습 만들기

- ε-탐욕 정책 구현하기
  - Q-학습 에이전트에서 수 선택하기

```
    values = self.model.predict(
      [board_tensors, move_vectors])
    values = values.reshape(len(moves))

    ranked_moves = self.rank_moves_eps_greedy(values)

    for move_idx in ranked_moves:
      point = self.encoder.decode_point_index(
        moves[move_idx])
        if not is_point_an_eye(game_state.board,
                               point,
                               game_state.next_player):
        if self.collector is not None:
          self.collector.record_decision(
            state=board_tensor,
            action=moves[move_idx],
          )
          self.last_move_value = float(values[move_idx])
          return goboard.Move.play(point)
    return goboard.Move.pass_turn()
```

TensorFlow 2로 Q-학습 만들기

- ε-탐욕 정책 구현하기
  - Q-학습 에이전트의 수 선택

```
class QAgent(Agent):
  def rank_moves_eps_greedy(self, values):
    if np.random.random() < self.temperature:
      values = np.random.random(values.shape)
    ranked_moves = np.argsort(values)
    return ranked_moves[::-1]
```

TensorFlow 2로 Q-학습 만들기

- 행동-가치 함수 훈련
  - 경험 데이터로 Q-학습 에이전트 훈련하기

```python
class QAgent(Agent):
  def train(self, experience, lr=0.1, batch_size=128):
    opt = SGD(lr=lr)
    self.model.compile(loss='mse', optimizer=opt)

    n = experience.states.shape[0]
    num_moves = self.encoder.num_points()
    y = np.zeros((n,))
    actions = np.zeros((n, num_moves))
    for i in range(n):
      action = experience.actions[i]
      reward = experience.rewards[i]
      actions[i][action] = 1
      y[i] = reward

    self.model.fit(
      [experience.states, actions], y,
      batch_size=batch_size,
      epochs=1)
```

감사합니다!

스터디 듣느라 고생 많았습니다.