Game (Pygame)

-play\_step(action) führt GameLoop aus, Spiel immer wieder neu starten

-> reward, game\_over, score

Agent führt Game und Model zusammen

Implementiert Training Loop

-state = get\_state(game)

-action = get\_move(state)

-model.predict()

-reward, game\_over, score = game.play\_step(action)

-new\_state = get\_state(game)

-remember

-model.train()

Model (PyTorch)

Linear\_QNet (DQN)

-model.predict(state)

-> action

Reward

-eat food: +10

-game over: -10

-else: 0

Action

-[1,0,0] -> straight

-[0,1,0] -> right turn

-[0,0,1] -> left turn

-nur die 3 Steps, weil bei z.B. turn right gefolgt von turn left -> Game Over (180 grad turn)

-immer in „laufrichtung“

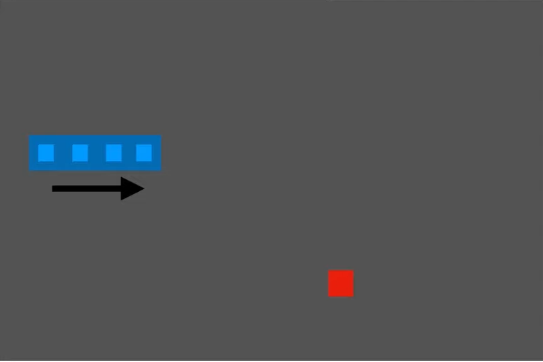
State (11 Values)

Tells game the state of the Game

[danger straight, danger right, danger left

Direction left, direction right, direction up, direction down

Food left, food right, food up, food down]

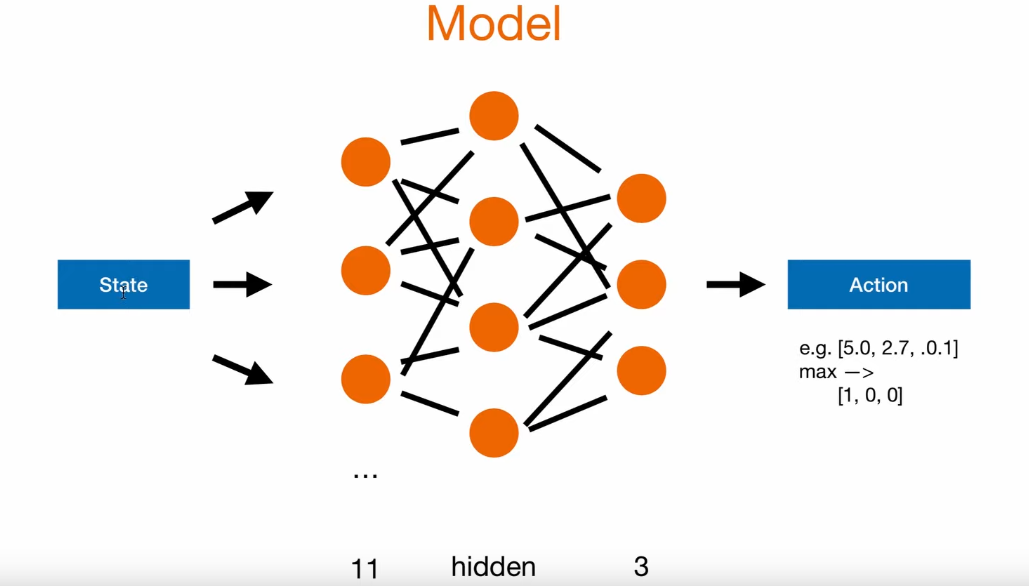


Food ist in diesem Beispiel „right turn“

[0,0,0 -> danger straight = false, danger right = false, danger left = false

0,1,0,0 -> direction left = false, direction right = true, direction up = false, direction down = false

0,1,0,1] -> food left = false, food right = true, food up = false, food down = true



Model kalkuliert nächste Action über ein „forward neural net“?

Input Layer (11 States)

Hidden Layer

Output Layer (3 Outputs)

* Predict Action über die Wahrscheinlichkeiten Maximum bestimmen

(Deep) Q Learning

Q Value = Quality of action

1. Init Q Value (= init model)
2. Choose action (model.predict(state)) (oder random move)
3. Perform action
4. Measure reward
5. Update Q value (+train model)

1-4 wiederholen

Bellman Equation wird verwendet um nächste action zu bestimmen

