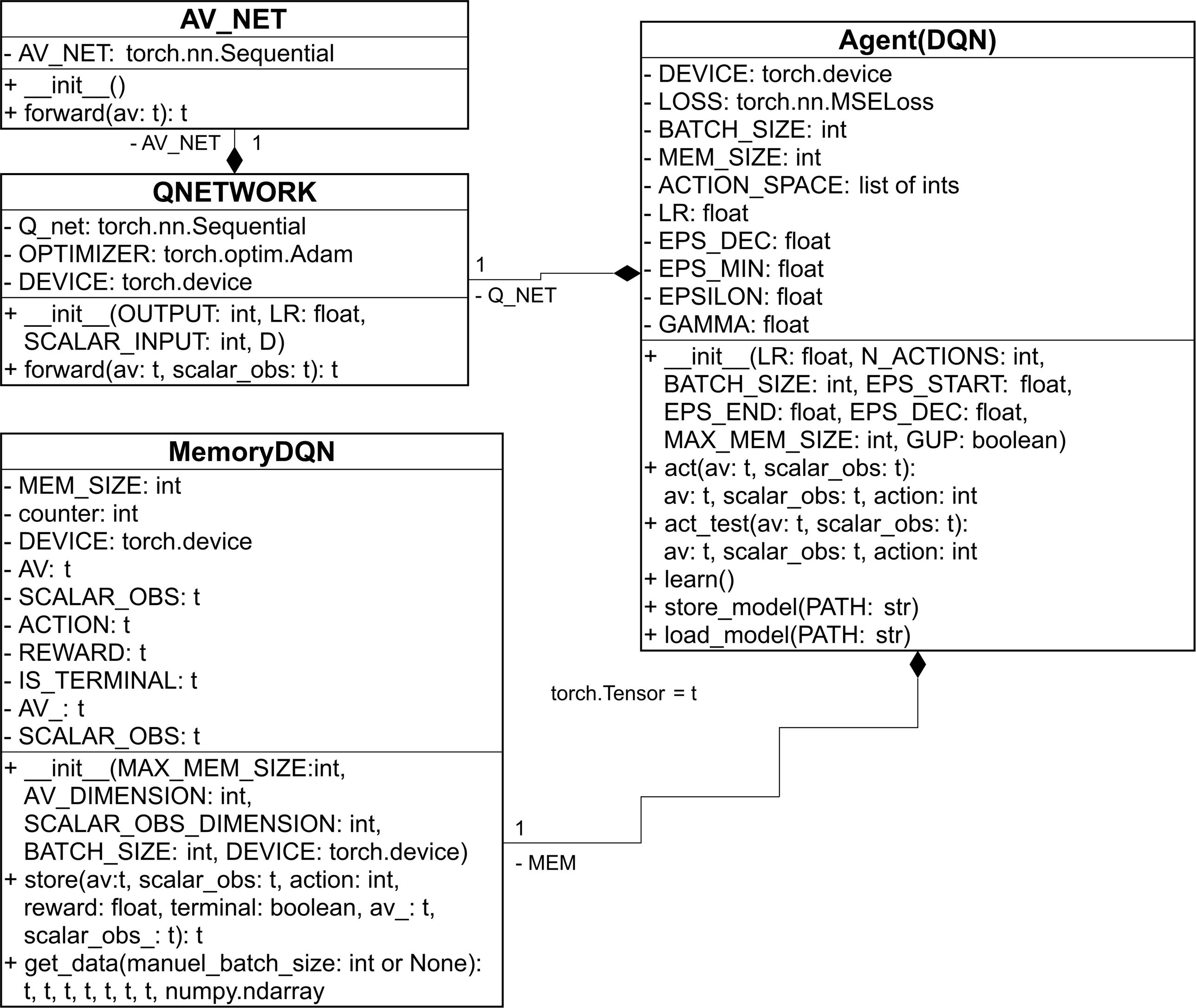
AV_NET
- AV_NET: torch.nn.Sequential
+ __init__()
+ forward(av: t): t
- AV_NET
1
QNETWORK
- Q_net: torch.nn.Sequential
- OPTIMIZER: torch.optim.Adam
- DEVICE: torch.device
+ __init__(OUTPUT: int, LR: float,
SCALAR_INPUT: int, D)
+ forward(av: t, scalar_obs: t): t
1
- Q_NET
Agent(DQN)
- DEVICE: torch.device
- LOSS: torch.nn.MSELoss
- BATCH_SIZE: int
- MEM_SIZE: int
- ACTION_SPACE: list of ints
- LR: float
- EPS_DEC: float
- EPS_MIN: float
- EPSILON: float
- GAMMA: float
+ __init__(LR: float, N_ACTIONS: int,
BATCH_SIZE: int, EPS_START: float,
EPS_END: float, EPS_DEC: float,
MAX_MEM_SIZE: int, GUP: boolean)
+ act(av: t, scalar_obs: t):
av: t, scalar_obs: t, action: int
+ act_test(av: t, scalar_obs: t):
av: t, scalar_obs: t, action: int
+ learn()
+ store_model(PATH: str)
+ load_model(PATH: str)
torch.Tensor = t
MemoryDQN
- MEM_SIZE: int
- counter: int
- DEVICE: torch.device
- AV: t
- SCALAR_OBS: t
- ACTION: t
- REWARD: t
- IS_TERMINAL: t
- AV_: t
- SCALAR_OBS: t
+ __init__(MAX_MEM_SIZE:int,
AV_DIMENSION: int,
SCALAR_OBS_DIMENSION: int,
BATCH_SIZE: int, DEVICE: torch.device)
+ store(av:t, scalar_obs: t, action: int,
reward: float, terminal: boolean, av_: t,
scalar_obs_: t): t
+ get_data(manuel_batch_size: int or None):
t, t, t, t, t, t, t, numpy.ndarray
1
- MEM