Un mdp è un framework per definire l’ambiente (stati, azioni, funzioni,ecc…) con il quale il nostro agente (IA o programmazione dinamica(algoritmo VI)) andra ad interagire per apprendere e risolvere il task.

E’ quadrupla S,A,P,R,gamma dove S è l’insieme dei possibili stati, ed ogni stato ha delle variabili che lo caratterizzano, e queste variabili possono assumere un valore all’interno di un certo dominio (anche semplicemente booleane) e altre quindi un vettore di tutti gli attacchi classificabili dall’intrucion detection system (ho gli attacchi nello stato tanti quanti ne conosce l’IDS e per ognuno dia ha una probabilità che indica se sono sotto attacco da quella tipologia di attacco).

A sono le azioni che possono essere intraprese all’istante t per andare su un nuovo stato t+1 con una certa probabilità P, con cui un’’azione venga scelta ed è data da una matrice StxAtxSt+1 e questa stessa matrice mi da o la ricompensa o il costo (asseconda del segno) che mi rappresenta la bonta di quell’azione in quello stato che mi portato sul nuovo (se l’azione ha senso reward alta, se non centra nulla bassa, quindi costo per segno negativo). Nel caso dei sistemi stazionari probabilità e reward non cambiano anche se cambia l’istante temporale, perche se mi trovo in St e vado in St+1 con sempre le stesse azioni, avro sempre stessa bonta. Cambia nel caso di sistemi non stazionari ovvero St puo variare al variare di t (non solo aggiungo nuova azione, rimuovo un’attacco, aggiungo variabile,ecc…)

L’obiettivo è di massimizzare la ricompensa, ed il comportamento dell’agente è descritto da una policy pigreco: SxA ovvero la probabilità con cui l’agente scelga l’azione At in St.

Infine in S ovvero gli stati trovo grazie a T, una funzione target, un sottoinsieme di S che sono gli stati di non attacco, stati target sicuri dove il mio agente si arresta, così riduco la dimensione. Infine ho anche una funzione PC ovvero pre condizioni che mi determina un sottoinsieme delle azioni applicabile in un determinato stato (se ho un certo attacco alcune contromisure sono inutili quindi non faccio determinate azioni se non mi aiutano a patchare; sto subendo sql injection ma vado a petchare cross site scripting, inutile).

DQN: deep Q network, uno dei primi se non il primo algoritmo per la risoluzione, ovvero l'approssimazione del comportamento di Q\*, ovvero la policy dell'agente, con reti deep. E' basato su convolutional neural network,multilayer perceptron il tutto addestrato con l'algoritmo Q learning con una variante ovvero lo stocastic gradient discent.

APEX-DQN: un'evoluzione del DQN in quanto, risolve alcuni problemi di efficacia e instabilità dell'algoritmo dqn. Basato sull'architettura apex per ridurre i tempi di convergenza nella fase di training ed il delta del valore di approssimazione della policy.

Fa uso di un'architettura distribuita per scalare DQN, distribuendo la generazione e la selezione dei dati piuttosto che usare un approccio standard di parallelizzazione della risoluzione del gradient (le tecniche possono essere combinate).

La raccolta delle esperienze può essere estesa a centinaia di lavoratori della CPU grazie ad un valore di priorità dell'esperienza che viene assegnata prima dell'archiviazione nei buffer di riproduzione.

C'è un buffer di riproduzione che viene utilizzato per le esperienze ovvero coppie di stato-azione-reward (interazioni passate dell'agente con l'ambiente) che vengono utilizzati per il training.

ciò permette di eliminare la correlazione temporale dell'azione, aumenta il riuso dell'esperienza con priorità maggiore dato che ha un carico informativo maggiore, campionandoli più volte ed evitare di rigenerare sempre nuovi dati dall'interazione dell'agente con l'ambiente.

IMPALA: basato sullo stocasthic gradient discent progettato per ambienti paralleli e distribuiti che fa uso di V-TrACE ovvero un algoritmo di apprendimento.

ovvero ci sono più attori che generano esperienze (azione-observation-reward) e che le mandano ad un learner che apprende, quindi c'è un disaccoppiamento tra l'apprendimento mediante i dati e la generazione dei dati, con la possibilità di una generazione parallela con più attori.

C'è anche un lag tra la policy usata per la generazione dei dati e quella per l'apprendimento del learner(la stima del gradiente), e qui entra in gioco l'algoritmo v-trace per ridurre questo lag.

PG: Policy gradient, basato su SGD per massimizzare la cumuative reward.

PPO: Proximal Policy Optimization Basato sul PG con l'aggiunta di tecniche di clipping per evitare aggiornamenti di policy troppo grandi.

Proximal Policy Optimization Basato sul PG con l'aggiunta di tecniche di clipping per evitare aggiornamenti di policy troppo grandi (+ stabilita, + convergenza)

COSE FATTE FIN’ORA:

creato un ambiente custom con pettingzoo con 2 agenti, un attaccante ed un difensore.

Ho definito un ambiente relativo al modello iniziale dove:

7 attacchi (pscan,pvsftpd,psmbd,pphpcgi,pircd,pdistccd,prmi) hanno una probabilità con tui il difensore lo valuta

# 0 < T1 < T2 < 1 e p < T1 rumore, T1 < p < T2 possibile attacco (prevenzione), p > T2 attacco by IDS (contromisure),

# p=1 attacco noto e strategia da attuare

# Lo stato del difensore è composto da: 7 p (una per ogni attacco) + 14 variabili di sistema,

# presumibilmente l'attaccante ora vede tutto per un taining più efficace

# il difensore ha 18 azioni, 21 componenti nello stato

# l'attaccante ne ha 7, lo stato esamina quello del difensore

# considererei lo stato clean quello di partenza e lo stato target una configurazione delle anomalie innocua (DA DEFINIRE BENE)

# per l'attaccante qual'è lo stato target di vittoria??

# STATO

# [ firewall([True/False])(0), blockedip([])(1), flowlimit\_ips([])(2), alert([True/False])(3), honeypot\_ips([])(4),

# log\_verb([0-5])(5),

# active([True/False])(6), quarantined([True/False])(7), rebooted([True/False])(8), backup([True/False])(9),

# updated([True/False])(10),

# manuallySolved([True/False])(11), everQuarantined([True/False])(12), everShutDown([True/False])(13),

# +

# pscan([0-1])(14), pvsftpd([0-1])(15), psmbd([0-1])(16), pphpcgi([0-1])(17), pircd([0-1])(18), pdistccd([0-1])(19), prmi([0-1])(20),]

# DIFENSORE: 18 azioni= [GenerateAlert(0), FirewallActivation(1), BlockSourceIp(2), UnblockSourceIp(3),

# FlowRateLimit(4), UnlimitFlowRate(5), RedirectToHoneypot(6), UnHoneypot(7), IncreaseLog(8),

# DecreaseLog(9), QuarantineHost(10), UnQuarantineHost(11), ManualResolution(12), SystemReboot(13),

# SystemShutdown(14), SystemStart(15), BackupHost(16), SoftwareUpdate(17), noOp(18)]

# ATTACCANTE: attacchi=[Pscan(0), Pvsftpd(1), Psmbd(2), Pphpcgi(3), Pircd(4), Pdistccd(5), Prmi(6), noOp(7)]

spiegare tutte le pre e post condizioni del 1 modello !