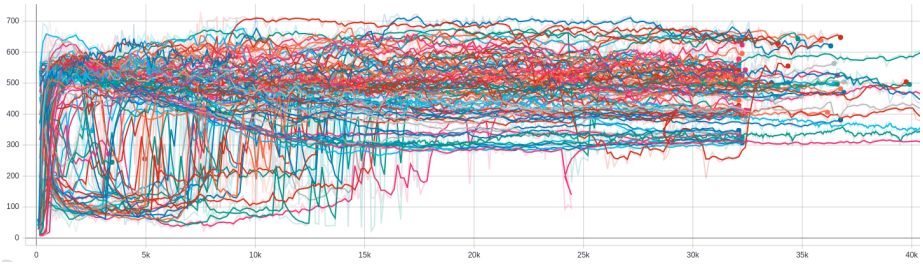
Reinforcement Learning:
Grid-Search på hyperparametere

Prosjektrapport TDAT3025
Anvendt maskinlæring med prosjekt

Bjørn Kristian Punsvik
Johan Martin Skavhaug

Høst 2019



1 Forord

Anvendt maskinlæring er et stort begrep som mange ikke helt har tak på hva dreier seg om. Vi fordyper oss i ett aspekt av et raskt voksende miljø med fremskritt som allerede i dag har stor betydning for hverdagen til alle uansett hvor tekniske de er. Tema er valgt ut av en nysgjerrighet rundt Reinforcement Learning og mystikken om hvordan en maskin kan lære seg selv det den trenger for å utføre de vanskeligste oppgaver. Arbeidet er fordelt mellom Bjørn Kristian og Johan, der Johan er den med mest innsikt i maskinlæringsdelen og kode-utvikling mens Bjørn Kristian er bedre på prosjektstruktur, maskinoppsett, og rapport. Vi bruker hverandres sterke sider samtidig som vi begge researcher og diskuterer sammen. Vi vil takke Jonathan Jørgensen vår veileder og mentor, Ole Christian Eidheim og Donn morrison våre forelesere, NTNU-LAB for lån av treningsmaskin og TIHLDE Drift for kontorplass under dette prosjektet. Vi har ellers ikke motatt noen form for økonomisk støtte, men har en formening i forkant av prosjektstart at maskinlæring er noe utrolig nytting og komplekst som vi ser på som fremtiden innen produktivitet og automasjon.

2 Sammendrag

Hensikten med dette prosjektet var å finne ut hvilke hyperparametre som hadde mest å si på treningen av et enviroment fra openAI's gym ved hjelp av reinforment learning. Vi valgte PyBullet-miljøet AntBulletEnv-v0. Her brukes Tensorflow(Tensorflow 2019) som rammeverk, TF Agents biblioteket(TF-Agents 2018) for reinforcement learning, og algoritmen Proximal Policy Optimization (PPO).

Tidsomfanget var på ca. fem uker (23 arbeidsdager) parallelt med studiet. Treningsmaskinen var 32 kjerner. Det ble trent 103 modeller opp til minst 30 000 iterasjoner på 15 hyperparametre med ulike verdimengder. Treningene ble automatisert ved å permutere relevante hyperparametre og verdier, og loggføre blandt annet *average retrun* for å bestemme om de var gode.

Ingen modeller klarte å gå, figur 8 viser de med høyest poeng. Vi kan fastslå at hyperparametrene vi testet ikke har noen betydelig innvirkning på suksessen i dette tilfellet med gitt miljø, agent og *policy*.

Innhold

3 Introduksjon

I reinforcement learning(RL) har hyperparametre mye å si for utfallet av treningen, uavhengig av hvilket problem og algoritme som er brukt. Å finne den optimale kombinasjonen av verdier for hyperparametrene kan være tidkrevende og vanskelig, da kombinasjonen ikke nødvendigvis trenger å være den man ville resonnert seg fram til, foruten at teorien bak algoritmene også kan være vanskelig å forstå.

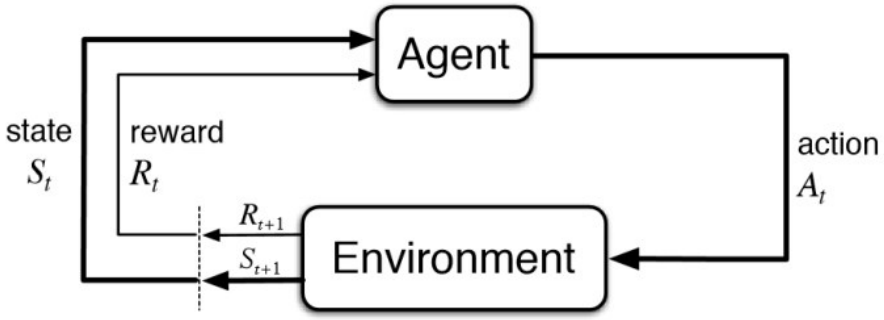
Det vi har gjort i dette prosjektet er å utforske forskjellige kombinasjoner av verdier for hyperparametrene ved hjelp av en blanding av *manuelt* og *grid*-søk. Det er naturlig at ikke alle hyperparametre har like mye å si for treningen i RL, noen vil utgjøre en større forskjell enn andre. Når antall hyperparametre vokser blir mengden kombinasjoner fort for stor til at det er hensiktsmessig å kjøre grid-søk over hele mengden. Ved å kombinere intuisjon og resonnering med grid-søk over mindre deler av mengden er det mulig å avgrense søket til de hyperparametrene som har mest å si.

4 Teori

Alle deltagerene i dette prosjektet har tatt faget "TDAT3025 Anvendt maskinlæring med prosjekt"(NTNU 2019). Her har de lært om sentrale begreper innen maskinlæring blant annet nevrale nettverk, ulike måter å representere, gruppere og klassifisere data, hvilke maskinlæringsmetoder som er hensiktsmessig å bruke ved gitte problemstillinger og begrensninger ved maskinlæring. Det har også blitt holdt en gjesteforelesning om introduksjon til reinforcement learning av Jonathan Jørgensen. Teknologier som blir brukt som studentene har erfaring med er python, bash, tensorflow, tensorboard.

Reinforcement learning er en del av maskinlæring som skiller seg fra de to andre store paradigmene *supervised-* og *unsupervised learning*. Dette fordi den ikke trenger ferdig markerte data A som du vet skal peke til et bestemet utfall B og forteller dette til læringsalgoritmen, x-train og y-train. Et system i RL består av et miljø og en agent som utøver en *policy* på miljøet. Agenten tar observasjoner av tilstanden(*state*) fra miljøet som input og bruker *policyen* til å knytte disse til en handling(*action*). Denne handlingen fører til en ny tilstand og får en belønning(*reward*) basert på hvor bra tilstanden den førte til er. Den nye tilstanden fører til en ny handling med en ny belønning, og slik fortsetter man til man enten gir opp, feiler, eller "vinner" miljøet. Se figur 1. Treningen av systemet består av å finne en policy som fører til høyest mulig akkumulert belønning etter miljøet er terminert.

Figur 1: The reinforcement learning loop



Hvis dimensjonene til observasjonene og handlingene er små, og handlingsrommet er diskret, vil man kunne lage en *policy* $\pi(s) \rightarrow (a)$ hvor hver tilstand har en bestemt handling som fører til største belønning. Er derimot dimensjonene store vil *policyen* ikke bare ha en bestemt handling som fører til høy belønning, den vil ha flere handlinger som fører til høy belønning, og det er vanskelig å forutsi hvilken handling som vil føre til høyest belønning senere. Dermed vil *policyen* $\pi(s)$ føre til den stokastiske variabelen $[a]$ som er en sannsynlighetsfordeling som oppgir sannsynligheten for at $s \rightarrow a$. Dette representeres som en *Markov Decision Process*.

For å forbedre en *policy* er det naturlig å ta steg i retningen som gir den høyeste estimerte verdien av *policyen*. Tar man korte steg vil det ta lang tid å komme til toppen av *reward-fjellet*, men tar man lange steg er sjansen stor for at man tråkker feil og faller ned fra fjellet. Trust Region Policy Optimization(TRPO) er en algoritme som optimaliserer *policyen* ved å ta steg i den retningen som forbedrer *policyen* mest, men uten at den nye *policyen* er for forskjellig fra den forrige. Dette hindrer at den nye *policyen* som blir valgt er en som har falt ned fra fjellet.

KL-divergens(D_{kl}) er et mål på hvor forskjellig en sannsynlighetsfordeling er fra en annen. Dette bruker TRPO til å sette en begrensning på hvor mye *policyen* kan endre seg for hvert steg.(Schulman, Wolski mfl. 2017) Algoritmen velger deretter den retningen som oppnår den største forbedringen i policy uten at D_{kl} blir større enn en satt grense.

Proximal Policy Optimization(PPO) er en implementasjon av TRPO som legger til D_{kl} som et ledd i tapsfunksjonen i treningen. Dette i motsetning til TRPO som i stedet for å legge til D_{kl} i tapsfunksjonen bruker tilnærminger av andregrads deriverte for å minske kompleksiteten.(Schulman, Levine mfl. 2015) PPO tvinger den førstegrads deriverte løsningen nærmere den andregrads deriverte løsningen ved å legge til D_{kl} som et ledd i tapsfunksjonen.

Teori om *Generalized Advantage Estimation* kan finnes i Schulman, Moritz mfl. 2015, mens *importance ratio clipping* kan leses i Schulman, Wolski mfl. 2017 kapittel 3.

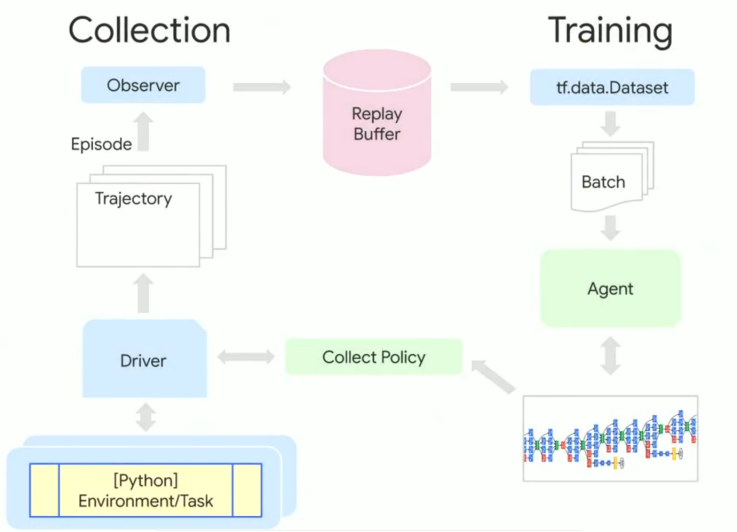
5 Tidligere relevant arbeid

I arbeidet til Young mfl. 2015 ble det eksperimentert med genetiske algoritmer for å unngå noe av de unødvendige iterasjonene med hyperparametre som har lite å si. Hyperparametrene det ble forsket på var nettverkskonfigurasjoner i konvolusjonelle nevrale nettverk(CNN), da spesiellt *kernel-size* og *output units* i konvulasjonslagene.

6 Metode

TF-agents og Tensorflow ble henholdsvis brukt til å modellere reinforcement learning agenten og til å trene agenten. Det oppretter en agent med *fully connected* nevrale nettverk for policy og verdifunksjonen. En mer detaljert loop mellom agenten og miljøet finner du i figur 2. Figuren beskriver TF-Agents implimentasjon av en RL-loop, som er det kildekoden vår er modellert etter (se vedlegg). Der setter vi opp hyperparametrene, logging, miljøet, nettverket, agenten, *metrics* for visualiseringen, *policy*, *replay buffer*, *checkpoints*, driver, og til slutt selve treningen.

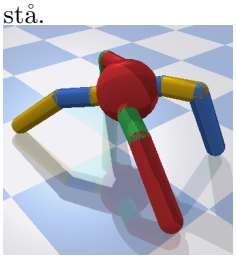
Figur 2: TF Agents



OPEN AI(Brockman mfl. 2016) Har en rekke miljøer klare til å trenes, men bruker ikke disse siden de krever en lisens. PyBullet var da et gratis alternativ. PyBullets miljøer kan være vanskeligere, men lisensen var et for stort hinder. PyBullet er en samling av RL gym miljøer som inkluderer versioner av Open AI miljøene *ant, hopper, humanoid* og *walker*. Samlingen av miljøer kan trenes med algoritmer som DQN(Deep Q-network Mnih mfl. 2013), PPO(Proximal Policy OptimizationSchulman, Wolski mfl. 2017) og SAC(Soft Actor-Critic Haarnoja mfl. 2018).

Miljøet valgt er *AntBulletEnv-v0*, avbildet i figur 3. Ant er et reinforcement learning miljø med et kontinuerlig handlingsrom, noe som gjør miljøet vanskeligere å løse enn miljø med diskrete handlingsrom. Beskrevet av PyBullet med *"Ant is heavier, encouraging it to typically have two or more legs on the ground"*. Målet i miljøet er at mauren lærer seg å gå.

Figur 3: Vår maur som har lært seg å stå.



Det agenten har å jobbe med ser du i figur 4. Agenten får 28 forskjellige input på form *float32* som til sammen utgjør en *state*. Agenten kan da utføre en handling ved å bestemme 8 verdier fra -1 til 1 som den tror på sikt vil gi en større *average return*. og gir de tilbake til miljøet.

Figur 4: Handlingsrom i miljøet *AntBulletEnv-v0*

```
Observation Spec:
BoundedTensorSpec(shape=(28,), dtype=tf.float32, name='observation',
                  minimum=array(-3.4028235e+38, dtype=float32),
                  maximum=array(3.4028235e+38, dtype=float32))
Action Spec:
BoundedTensorSpec(shape=(8,), dtype=tf.float32, name='action',
                  minimum=array(-1., dtype=float32),
                  maximum=array(1., dtype=float32))
```

Miljøet har en resetfunksjon som blir trigget om en av to *states* er sanne. Om mauren enten har kroppen sin i kontakt med bakken, funnet ut ved Y-verdi på et punkt i kulen. Eller at *replay_buffer_capacity* er nådd, altså lengden på episoden har nådd max antall *timestep*. Enkelt forklart; mauren detter eller tiden går ut.

Poengene blir gitt ut fra hvor stor fart kroppen har og øker etter som distansen til målet minker. En test på miljøet hviser at på første *timestep* der mauren ikke beveger på seg blir det gitt 0.54 poeng. Største manuelt observerte gitte poeng på ett enkelt timestep er: $1.5 < poeng < 2$. Dette ble observert i det en landing fikk mauren til å velte og kroppen hadde fart før den aktiverte en reset på miljøet.

6.1 Grid Search

Grid search er prosessen å utføre hyperparametrendringer for å finne den optimale samlingen for en gitt modell. Dette kan gjøres ved å kjøre læringen med alle permutasjonene av utvalgte hyperparametre (se kode i figur 5 side 8) og sammenligne resultatet. Slik finner du ut hvilke hyperparametre som er interesante og hvilke som ikke påvirker modellen betydelig.

Figur 5: eksempelkode på permutasjonsscript i bash

```
# En gjennomgang for hver permutasjon av variablene a,b,c
# og verdiene de kan ha
for X in {a=1,a=2}{b=True,b=False}{c=0,c=0.1};do
    # Lager en unik suffix
    sfx=`echo $X | sed 's/--//g' | tr ' ' '_' | tr '=' '-'| \
        sed 's/true/t/g' | sed 's/false/f/g'`

    # Kjører trening på hyperparameterene
    python train_eval.py $X --sfx=$sfx
done
```

6.2 hyperparametre

Ved å bruke TF-Agens er det mulig å endre på over 20 hyperparametre. Det blir for mange permutasjoner til å finne meningsfulle data innen tidsbegrensningene. Tabell 1 side 9 viser de utvalgte parameterene som antas å ha størst innvirking på treningen og hvilke forskjellige verdier som er testet. For å bestemme om en permutasjon er mer interesant enn en annen ser man hovedsaklig på hvor stor *average return* modellen samler opp etter hvert som modellen trenes. Får å sette en stopper der modellen tilsynelatende ikke vil bli stort bedre ble det gjort tester og funnet ut at ingen betydelige endringer skjer etter ca 30 000 iterasjoner. Dette ble avluttningen for de fleste treningene.

Tabell 1: De relevante hyperparametrene

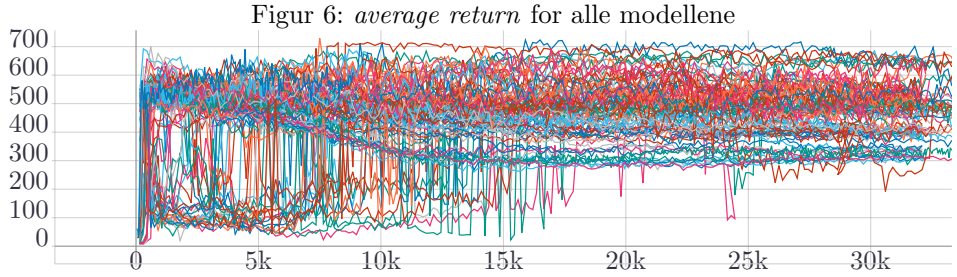
Navn	**Syntax**	**Verdimengde**	**Beskrivelse**
Tanh	tanh	True/False	Bruk tanh aktivering for Nevrale Nettverk, hvis ikke relu
Læringsrate	learning_rate	3e-{1,2,3,4}	Learningsrate, steglengden til *optimizer*-en
Læringsrateforfall	lr_decay	.8, .9, .95	Bruk tilpassende LR? Starter LR på LR*10
Value nettverk	vfc	(200,100), (280,38,5), (555,53,5), (1100,75,5)	Fullt koblet lag for *value network*
Policy nettverk	pfc	(200.100), (280,150,80), (550,149,40), (1100,300,80)	Fullt koblet lag for *policy network*
Klipprate	clip	0, .2	*Importance ratio clipping*
GAE	use_gae	True/False	Bruk *Generalized Advantage Estimation*?
Entropi	entropy	0, .01	Entropiregulering
calue prediction loss	value_pred_loss_coef	.5, .0001	Multiplikator for *value prediction loss* for å balansere *policy gradient loss*
KL *cutoff factor*	kl_cutoff_factor	2	Hvis *policy* KL endres mer enn dette per *timestep*, legg til en kubet KL straff til *loss*-funksjonen
KL *cutoff coef*	kl_cutoff_coef	1000	*Loss coefficient* for KL *KL cutoff*
Startverdi ad. KL beta	init_kl_beta	1	Startverdi for beta koeffisient av adaptiv KL straff
Ønsket KL mål	kl_target	.01, .02, .03	Ønsket KL mål for *policy* oppdateringer
KL beta toleranse	kl_tolerance	.3, .5, 1	Toleranse for *kl_beta*
*E*pisoder per iterasjon	collect_episodes_per_iteration	30, 1020	Antall steg i ta i miljøet før hver oppdatering av *policyen*, fordelt på alle paralelle miljø

Treningsmaskin

Til prosjektet fikk studentene tilgang til en VM i NTNUs VM-park. Denne hadde 32 kjærner klokket til 2.1 GHz, 32 GB minne og kjørte Ubuntu 18.04 64-bit. Denne ble brukt til å trene modellene. De fleste modellene tok 2.5 til 4.5 timer å trene per permutasjon, avhengig av hvilke permutasjoner som ble testet.

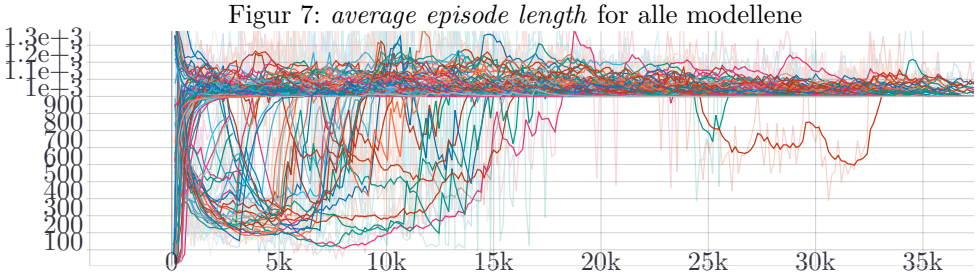
7 Resultat

Figur 6: *average return* for alle modellene



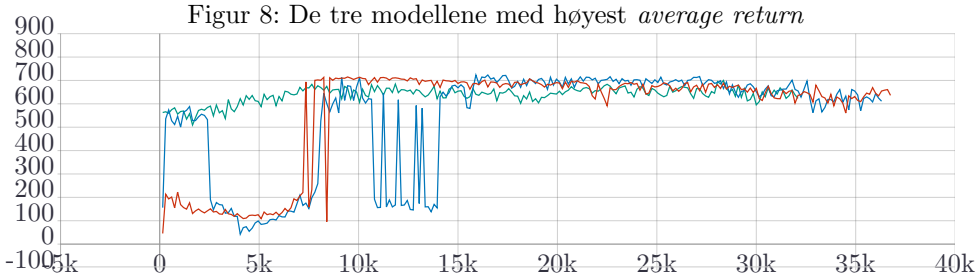
Figur 6 viser *average return* fra alle modellene i en graf for å kunne se hva som omhandler alle modellene.

Figur 7: *average episode length* for alle modellene



Figur 7 viser *average episode lenght* fra alle modellene i en graf for å kunne se hva som omhandler alle modellene. Lagt til 0.8 i smoothing for å klarere se forskjellene på modellene.

Figur 8: De tre modellene med høyest *average return*

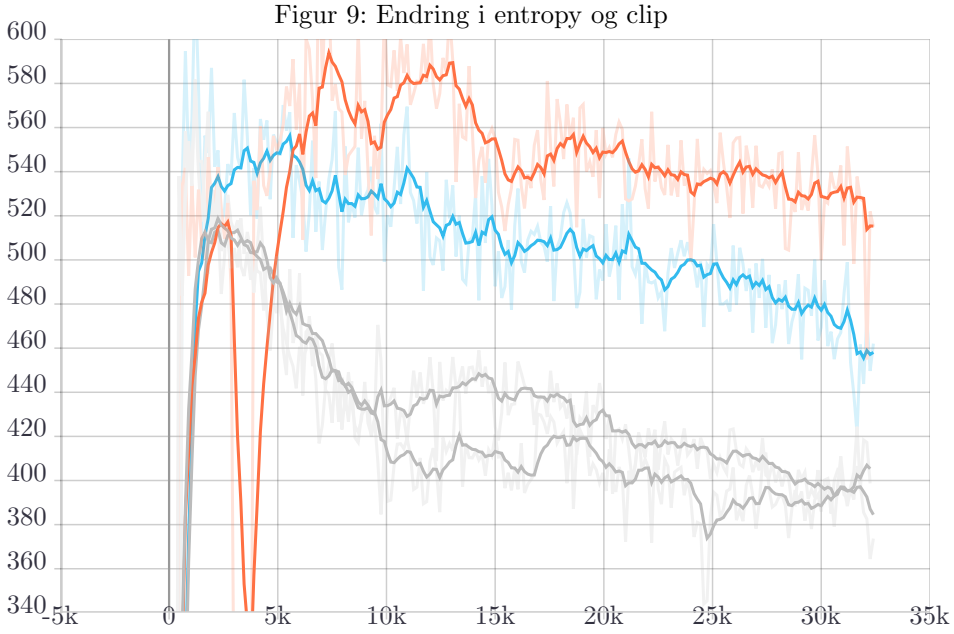


I figur 8 er det plottet de med mest poeng av alle permutasjonene. Den røde ligger på 700 etter bare 10 000 iterasjoner, den blå konkurerer om ledelsen etter 16 000, og den grønne er den modellen med gjennomsnittlig mest poeng tilsammen på alle stegene. Rød, blå, grønn har henholdsvis kl-toleranse og kl-target på (0.3, 0.02), (0.5, 0.03) og (1.0, 0.03).

Hvis ikke annet er oppgitt har de neste modellene disse hyperparametrene:

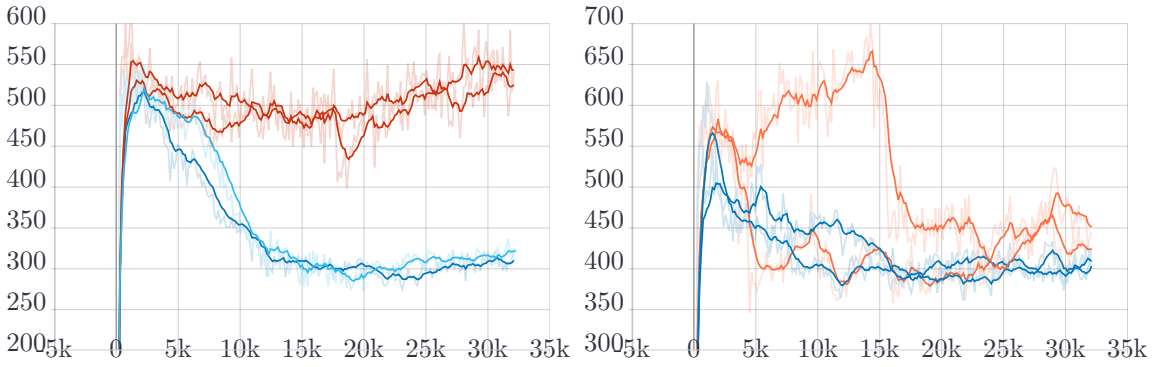
- Bruk GAE
- Bruk aktiveringsfunksjon Tanh
- Læringsrate: 0.0003
- Læringsratedecay: 0.98
- Oppsett av nevrale nettverk: 2 fullt koblet lag med hhv. 200 og 100 enheter.
- *Importance Ratio Clipping*: 0.0
- Entropi regularization loss beta (heretter bare **entropy**): 0.0
- *Value prediction loss coefficient*: 0.5
- Ønsket KL mål: 0.01
- KL beta toleranse: 0.3
- Episoder per iterasjon: 30

Figur 9: Endring i entropy og clip



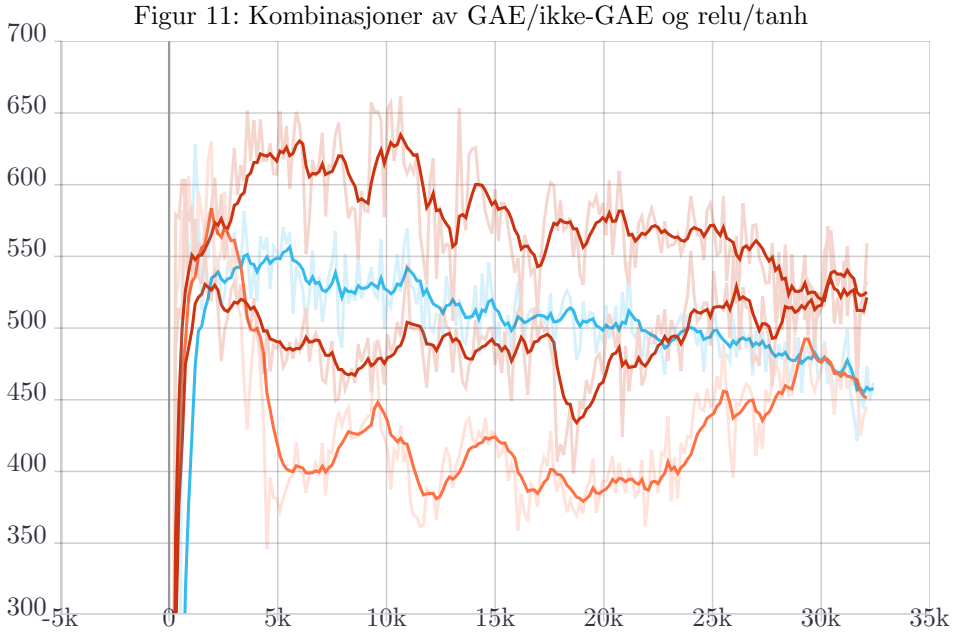
Her er de øverste linjene med entropy regularization = 0, og henholdsvis clip = 0.2 og 0. De nederste linjene har entropy regularization = 0.01 og linjen med clip = 0 er den eneste som økte i reward i løpet av de siste treningsstegene.

Figur 10: Endring i entropy og clip
Venstre: ikke GAE, høyre: aktiveringsfunksjon relu



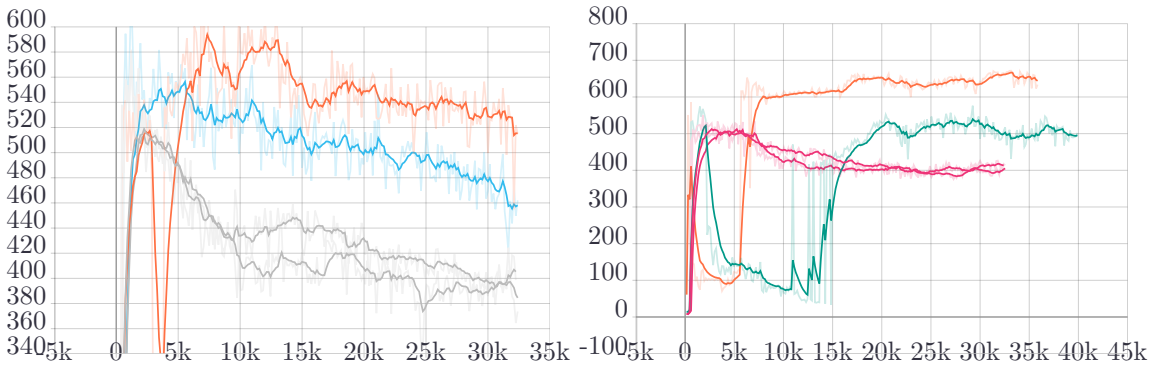
Slik som på forrige plot er også de beste modellene med entropy = 0, med den beste av de med clip = 0.2

Figur 11: Kombinasjoner av GAE/ikke-GAE og relu/tanh



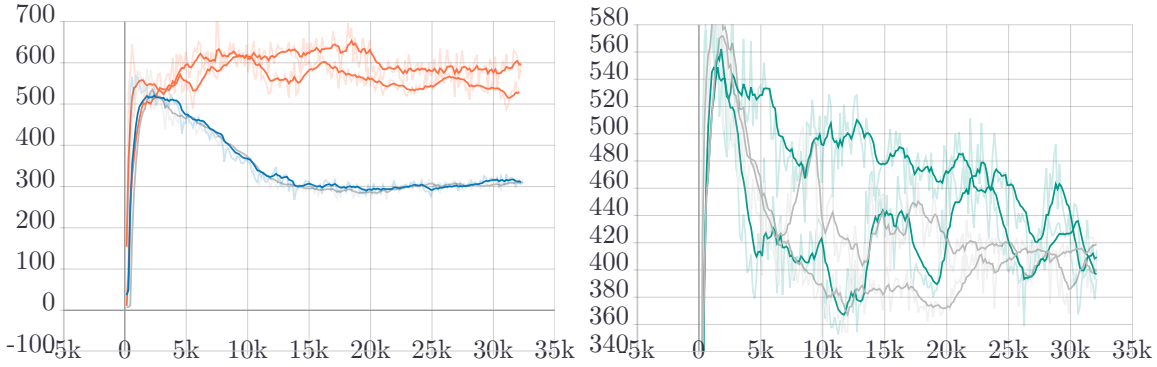
De røde linjene er det brukt GAE, blå og oransje ikke. Det er den blå linja, og den nederste røde som har tanh som aktiveringsfunksjon. Det er den røde linja med tanh som ender opp øverst til slutt.

Figur 12: Endring i entropy og clip.
Venstre: value coefficient = 0.5, høyre: valuecoefficient = 0.0001



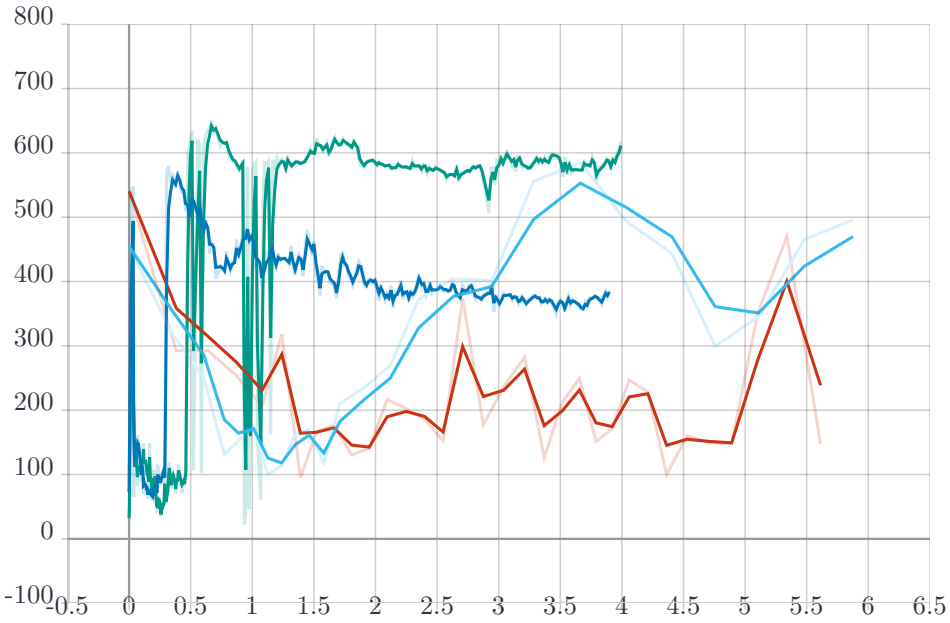
I begge plottene er de øverste linjene med entropy regularization = 0, og henholdsvis clip = 0.2 og 0. De nederste linjene har entropy regularization = 0.01.

Figur 13: Endring i entropy og clip med *valuecoefficient* = 0.0001
Venstre: ikke GAE, høyre: aktiveringsfunksjon relu



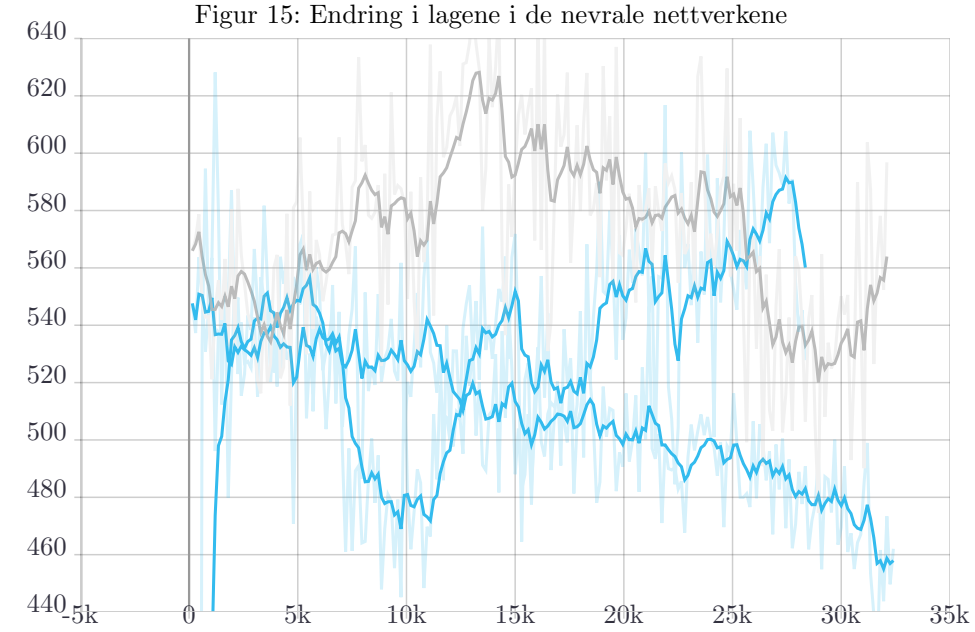
Til venstre er beste modellene med entropy = 0, med den beste av de med clip = 0.2. Til høyre er de grønne med entropy = 0.

Figur 14: Endring i entropy og episoder



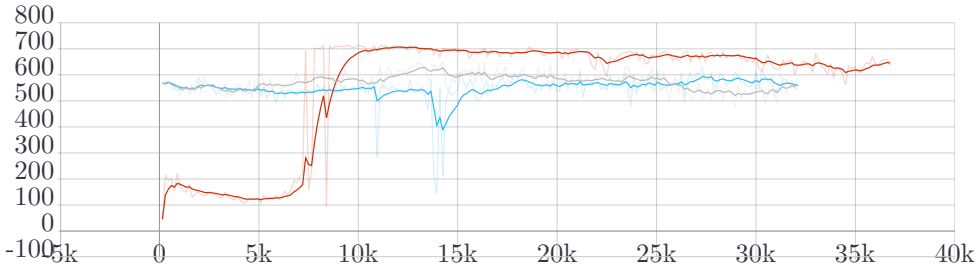
De blå linjene har entropy = 0.01. Mørkeblå og grønn har 30 episoder, rød og lyseblå har 1020. Y-aksen er beregningstid.

Figur 15: Endring i lagene i de nevrale nettverkene



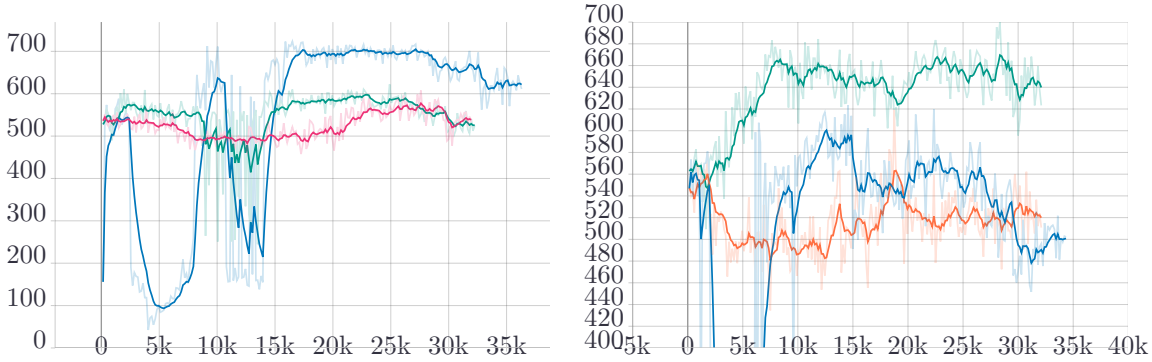
Den øverste blå linjen har en nettverkskonfigurasjon på (1110,75,5) og (1110,300,80) for hhv. verdi og policy nettverket i agenten. Den grå linjen er på (555,53,5) og (555,149,40). Den nederste blå linjen er (200,100) i begge. Verdi- og *policy*nettverk på hhv. (555,53,5) og (555,149,40) er nettverkene som ble brukt i de neste kombinasjonene.

Figur 16: Endring i kl_target, nettverk (555, ...), kl_tol = 0.3(standard)



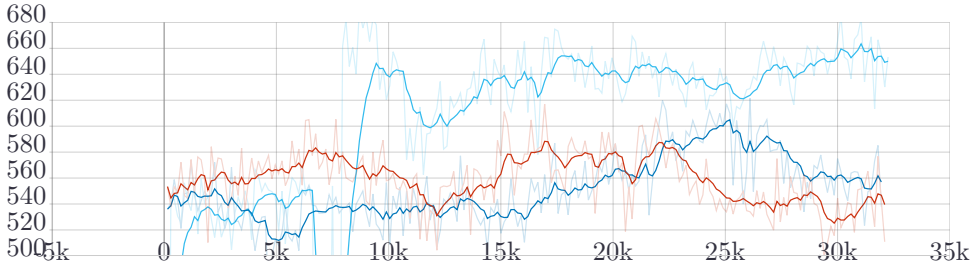
Grå, rød, blå med henholdsvis kl_target på 0.01(standard), 0.02 og 0.03.

Figur 17: Endring i kl_target, nettverk (555, ...)
Venstre: kl_tol = 0.5 høyre: kl_tol = 1.0



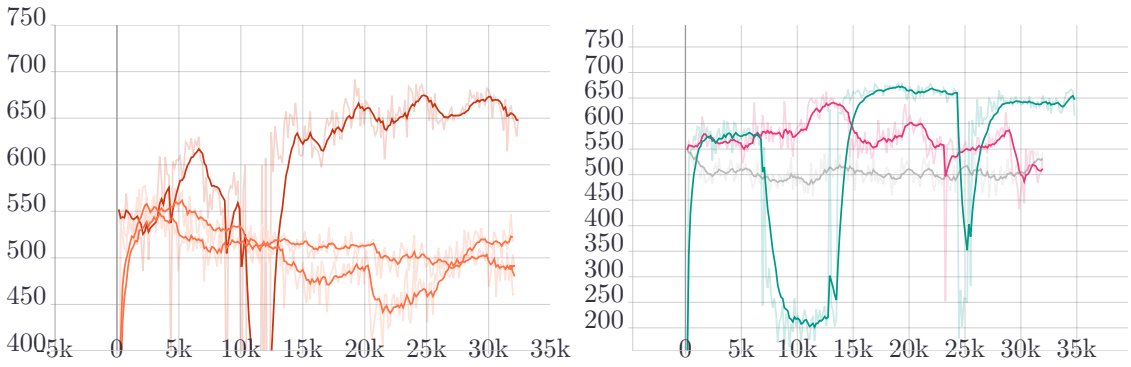
Rosa, grønn, blå med hhv. 0.01, 0.02 og 0.03 i kl_target. Blå, oransje, grønn med hhv. 0.01, 0.02 og 0.03 i kl_target.

Figur 18: Endring i learning rate, nettverk (555,...), lr_decay = 0.95



Lyseblå, blå, rød har hhv. learning rate på 0.3, 0.03 og 0.003.

Figur 19: Endring i learning rate, nettverk (555,...)
Venstre: lr_decay = 0.9 høyre: lr_decay = 0.8



Rød har learning rate 0.3. Den nederste oransje har learning rate 0.03 og tar igjen learning rate 0.003 på slutten. Grønn, rosa, grå har hhv. learning rate på 0.003, 0.03 og 0.3.

8 Diskusjon

Vi vet at en maur som holder seg i livet ut tiden men står i ro på stedet vil generere rundt 0.6 poeng per *timestep*. På vår grense satt på 1000 timesteps kan vi anta at en slik modell vil ha gjennomsnittlig 600 poeng. Om kroppen har fart store deler av episoden kan vi med trygghet si at gjennomsnittspoeng per timestep vil være over 1, særlig om mauren kommer seg nærmere målet og dermed får en økt mengde poeng per steg. Fra figur 8 ser vi at de to beste modellene aldri får mer enn rundt 700 poeng. Ut ifra antagelsen at en maur med fart ville opptjent over 1000 poeng, som ville vært gjennomsnittlig 1 poeng per timestep, kan vi dermed fastslå at ingen av permutasjonene fikk til å gå.

Generelt ut ifra figur 6 og 7 kan vi si at uansett hvilken permutasjon av hyperparametre vi har testet vil alle kunne stå uten å falle etter rundt 18 000 iterasjoner siden dette er punktet der ingen modeller har en *average return* på mindre enn rundt 300 og lengden på episoden er alle 1000 stegene.

Figur 7 kan vi fra iterasjon 22 000 se at noen modeller ikke overlever sine fulle 1000 tidssteg. Dette er trolig grunnet tilfeldigheten i utforskningen som får modellen til å ha "uflaks". Vi ser den kommer seg tilbake til 1000 tidssteg raskt. Dessuten ar antallet modeller som gjør dette 3 av 103, så de kan ansees som unormalheter som ikke er typisk for resten av de trente modellene. Noe uforklarlig som ikke er funnet svar på er hvorfor modellene viser at gjennomsnittlig episodelengde kan gå over 1000 tidsstag når dette er et hardt tak satt i kildekoden.

I figur 9 til figur 19 er det plottet forskjellige kombinasjoner opp mot hverandre. Disse er plottet hver for seg for å kunne få ett inntrykk av hver hyperparameter. I figur 9 ser vi at entropy regularization = 0.01 er vesentlig dårligere

enn 0.0. Den beste modellen her har clip = 0.2, men forskjellen ser ikke ut til å være konsistent. Dette støttes av resultatene i figur 10 hvor entropy = 0.0 gjorde det best, og clip ser ikke ut til å hverken ha stor negativ eller positiv effekt.

I tillegg ser vi at entropy = 0.01 i kombinasjon med å ikke bruke GAE gir dårligere resultat, mens å ikke bruke GAE ikke ser ut til å forverre utfallet når entropy = 0.0. Vi ser også at det å bruke relu som aktiveringsfunksjon i de nevrale nettverkene ser ut til å gjøre treningen særs ustabil. Figur 11 bekrefter at å bruke GAE gir bedre resultat, men det ser ikke ut til at relu som aktiveringsfunksjon er like destruktiv her, faktisk er det kombinasjonen relu/GAE som har høyest score gjennom store deler av treningen.

Figur 12 viser hvordan koeffisienten til verdiprediksjonstapet påvirker treningen. Her er det valgt å bruke GAE, og tanh som aktiveringsfunksjon siden dette førte til de beste og mest stabile resultatene i de forrige plottene. Vi ser at modellene med entropy = 0 får en veldig dårlig policy like etter treningen starter, og deretter kommer seg kraftig tilbake lenger ut i treningen. Hvorfor akkurat denne kombinasjonen fører dukkert i ytelsen er usikkert, men det er uansett klart at entropy = 0 og clip = 0.2 får best resultat.

Slik som i figur 10 viser også figur 13 at aktiveringsfunksjonen relu virker dårligere enn tanh, og at at entropy = 0 er best. Siden det er vist at relu i kombinasjon med GAE fungerer dårlig droppes relu, og GAE og tanh brukes i resten av kombinasjonene.

Økes antall steg som skal tas i miljøet for hver oppdatering av *policyen* vil hver oppdatering ta proporsjonalt mere tid, og antall steg per iterasjon vil gå opp. Derfor ble modellene med 1020 steg per oppdatering trent i dobbelt så mange steg som de vanlige på 30, og enkleste måte å sammenligne de blir i relativ beregningstid. I figur 14 ser vi den samme dyppen som i 12, men entropy = 0.01 ender opp øverst. Den signifikante økningen i steg per oppdatering ser ikke ut til å være verdt å utforske nærmere, da den tar svært lengre tid å trene modellene for lite til ingen gevinst.

Når sammensetningen til de nevrale nettverkene endres i figur 15 er det klart at det tredje laget i de nevrale nettverkene fører til en merkbar økning i *average return*. Størrelsen på de nevrale nettverkene ble valgt ut med tanke på dimensjonene til observasjons- og handlingsrommet. Det vises også at størst nødvendigvis ikke er best, og dermed ble det også forsøkt med henholdsvis (280, 38, 5) og (280, 150, 80) for verdi- og *policy*nettverkene. Disse nettverksstørrelsene ble brukt til å trene modeller på et modifisert Ant-miljø slik at resultatene ikke kan sammenlignes.

Når kl_target økes forventes en økning av modellens utforskning av handlingsrommet, og dermed en mindre stabil trening av modellen. I figur 16 ser vi at en liten økning fra 0.01 til 0.02 gjør at modellen utforsker mer av handlingsrommet og henger seg opp på en dårlig verdi før den etterhvert finner fram til bedre handlinger. Øker vi kl_target enda mer ser vi at modellen ikke ender opp med å bli enda bedre, og at denne modellen ga ok resultat fra starten av. Det er tenkelig at denne modellen rett og slett var heldig, og vi ser klart at den utforsket en mindre heldig del av handligsrommet senere.

Hvis toleransen fra kl_target økes forventes enda større utforskning, og også veldig ustabil trening. Figur 17 bekrefter dette. Den økte utforskningen fører også til at vi finner prosjektets høyeste *average return* verdier her. Selv om de høyeste verdiene er her bør ikke de økte kl verdiene brukes videre, da det er en viss tilfeldighet over modellene.

Videre ser vi også i figurene 18 og 18 at en økning i læringsraten også øker ustabiliteten i modellene, men at også dette kan lede til høye verdier for *average return*.

Gjennomgående for alle modellene er at de bruker minimal tid på å lære seg at å falle er en dårlig ting. Sett fra figur 7 på *average episode length* er det størst samling de første 2000 iterasjonene. Deretter er det bare noen modeller som strever med å så lengden opp til 1000. Dette forteller oss at de fleste modellene vil lære seg å holde seg stående og at det er tidsbegrensningene som setter stopper for poengene og ikke at de detter og resettes på at kroppen er i kontakt med bakken. Dette vil være en fin måte å eliminere modeller som ikke gjør det bra i starten hvis målet var å finne modeller som er raske å trene. Om man bare trengte å trene de 3-5000 første iterasjonene ville man kunne utforske mange flere hyperparametre og gjort et bredere grid-search. Det er likevell ikke alltid en god idé å terminere dårlige starter om målet er å finne en god modell og tid ikke er en prioritering, som vi kan se på figur 8 der 2 av 3 modeller ikke har lært seg å stå på 5000 itterasjonspunktet. Dette er noe man må overveie om man kan få en modell som senere vil gjøre det bra eller om man vil gå for mange raskere modeller og mulig finne en like god eller bedre på kortere tid. For begrensningene og hensikten med dette prosjektet kunne vi valgt å gå for en kortere itterasjon og dermed få trent flere modeller og sjekket flere permutasjoner av hyperparametre.

Den største feilen som ble gjort underveis i prosjektet var å bruke lang tid på å få en modell til å gå. Det ble for stort fokus på å løse moljøet og ikke på å kjøre grid-search på flest mulig parametere. I tillegg var tiden satt av til analyse av dataene vi genererte for kort. Noe man kunne fokusert på var å lage python-script som analyserte rådata og gav konkrete hyperparametre som var verdt å satse på. Nåværende arbeidsmetode krevde manuel analyse av grafer fra tensorboard. Det er 3.6 GB rådata som et grid-search-script ville kunne finne gode analyser fra.

Prosjektet har funnet ut hvilke modeller disse settene med hyperparametre og verdier trenes til. Dette er over 300 timer med komputert data som videre trening kan bygge på. Det kan fastslås at ingen av disse vil trene opp en modell som går innen iterasjonsrammen.

9 Konklusjon

I dette prosjektet startet vi med verdier på hyperparametrene tatt fra TF-agents sin eksempelkode. Disse hyperparametrene endret vi ettersom vi så hvilke verdier andre hadde prøvd, og våre egne ideer på hvilke som ville gi interessante resultat. Vi har testet forskjellige verdier for alle hyperparametrene vi mener har en signifikant betydning for utfallet. Vi kan ikke si at noen av hyperparametrene vi testet har en betydelig innvirkning på suksessen til treningen i implementasjonen vi har brukt. Vi identifiserer at enkelte verdier av hyperparametre er til hjelp og andre til hindring, men ingen av kombinasjonene fikk en average return høy nok til at modellen ble en suksess.

Referanser

Tensorflow (2019). *An end-to-end open source machine learning platform*. URL: https://www.tensorflow.org/ (sjekket 12.11.19).

TF-Agents (2018). *TF-Agents: A library for Reinforcement Learning in TensorFlow*. URL: https://github.com/tensorflow/agents (sjekket 12.11.19).

NTNU, Trondheim (2019). *Anvendt maskinlæring med prosjekt*. URL: https://www.ntnu.no/studier/emner/TDAT3025/ (sjekket 12.11.19).

Schulman, John, Filip Wolski mfl. (2017). *Proximal Policy Optimization Algorithms*. arXiv: 1707.06347 [cs.LG].

Schulman, John, Sergey Levine mfl. (2015). *Trust Region Policy Optimization*. arXiv: 1502.05477 [cs.LG].

Schulman, John, Philipp Moritz mfl. (2015). *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. arXiv: 1506.02438 [cs.LG].

Young, Steven R. mfl. (2015). "Optimizing Deep Learning Hyper-parameters Through an Evolutionary Algorithm". I: *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*. MLHPC '15. Austin, Texas: ACM, 4:1–4:5. ISBN: 978-1-4503-4006-9. DOI: 10.1145/2834892.2834896. URL: http://doi.acm.org/10.1145/2834892.2834896.

Brockman, Greg mfl. (2016). *OpenAI Gym*. eprint: arXiv:1606.01540.

Mnih, Volodymyr mfl. (2013). "Playing Atari With Deep Reinforcement Learning". I: *NIPS Deep Learning Workshop*.

Haarnoja, Tuomas mfl. (2018). *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. arXiv: 1801.01290 [cs.LG].