FlyingBird

Tobias Hanssen, Jørgen Troland, Andreas Falkmo 24.10.2025

1: BESKRIV PROBLEMET

OMFANG / *SCOPE*

Vi skal bruke maskinlære for å trene opp en agent til å kjøre gjennom spillet på best mulig måte og oppnå høyest mulig score. Spillet er en kopi av det berømte spillet Flappy Bird fra 2013. Vi mener at vi kan lære opp en maskin til å komme lengre enn hva ett menneske ville klart på mye kortere tid.
Produktet er ikke rettet til å bli solgt ut i markedet, men heller en gøyal løsning av prosjektet hvor vi samtidig får utfordret oss i maskinlæring. Det finnes allerede lignende løsninger som for eksempel "Flappybird-AI". Flappybird-AI benytter seg ikke av RL og OPP som vi kommer til å gjøre, men N.E.A.T.

METRIKKER

Dette prosjektet benytter Reinforcement Learning (RL) med algoritmen Proximal Policy Optimization (PPO). Dette gjør at klassiske klassifikasjonsmetrikker som accuracy eller precision ikke passer. I stedet evaluerer vi ytelsen basert på belønningssignalene i miljøet:

1. *Gjennomsnittlig total belønning (Inkluderer all form for poeng som er tildelt.)*
2. *Gjennomsnittlig poengsum (1 poeng per rør passert)*
3. *Overlevelseslengde (Antall frames før agenten koliderer.)*
4. *Forklart varians- En intern metrikk fra PPO som måler hvor godt verdifunksjonen forutsier belønning. Verdier nær 1.0 betyr at modellen har lært en god intern representasjon av miløet. Brukes til å overleve treningskvalitet.*

Minimum business metric-ytelse for at prosjektet skal være en suksess:

1. *Læringsforberdring:*
    a. *Den trente agenten skal opppnå minst 3 ganger høyere gjennomsnittlig poengsum enn baseline-agenten med enkel heuristikk.*
2. *Stabil oppførsel:*
    a. *Agenten skal kunne passere minst 3 rør på rad i flertallet av testkjøringer uten å treffe et hinder.*

2: DATA

Prosjektet vårt bruker ikke tradisjonell treningsdata i form av statiske datasett. Det genereres data kontinuerlig gjennom interaksjon mellom miljøet og spilleren. Hver interaksjon består av fire deler:
State – informasjon om spillsituasjonen.
Action – hva spilleren gjør.
Reward – Spilleren får tilbakemelding etter hvor bra handlingen var.
Next state – er neste observasjon.
Dette er veldig vanlig Reinforcement Learning-data. I denne implementasjonen er observasjonen en vektor med fire forskjellige verdier.
Den første verdien er fuglen sin vertikale posisjon, imellom 0 og 1. Den andre er fuglen sin vertikale hastighet, ca mellom –1 og 1. Tredje verdien er den horisontale avstanden til neste røråpning, imellom 0 og 1. Den fjerde og siste verdien er den vertikale avstanden til neste røråpning, også mellom 0 og 1. Alle fire verdiene får vi ut av miljøet i hvert tids-steg av spillet.
På grunn av at vi bruker Reinforcement Learning og ikke Supervised Learning har vi ikke ferdige labels. Vår agent lærer med å bli belønnet enten med positiv eller negativt. Så belønningsfunksjonen fungerer som en slags label.
Når vi har gjort det slik vil vi unngå problemer med label-støy, dette er fordi belønningene er deterministiske og konsistente ved at de er hardkodet inn i miljøet vårt.
Dataen samles da inn av vår egen simulering etter at agenten gjør handlinger i miljøet. Hvor mye data vi trenger er ikke veldig spesifikt for vår del, siden vi i teorien kan oppnå å fly uendelig med fuglen og få en “uendelig” score. Dette betyr at vi har tilgang til så mye data vi måtte ønske. I forhold til Supervised Learning er dette en stor fordel, for vi får ingen personvernproblemer, ingen lisensproblemer, og som sagt kan vi samle inn uendelig mye data om ønskelig i spillet vårt.

Vedlagt bilde viser resultatet på 2 av våre trente agenter og baseline-modellen.

Forsøk	800k Timesteps	200k Timesteps	Baseline
1	22	0	0
2	38	0	0
3	78	2	1
4	68	0	0
5	14	1	1
6	1	2	1
7	29	1	0
8	39	2	0
9	9	0	0
10	33	0	0
Gjennomsnitt Score:	33.1	0.8	0.3

Vi kan her med godt bevis si at agenten våres er en suksess. Gjennomsnittscoren til den best trente modellen 800k, er 110x bedre enn baseline modellen.

3: MODELLERING

Vi bruker som nevnt tidligere Reinforcement Learning (RL). RL baserer seg på å gi AI-en poeng/-poeng basert på utførelsen. Dersom den passerer et hinder får den +1 poeng. For hvert steg den tar uten å kolidere får den +0.1 poeng. Den blir også belønnet +0.1 poeng ved å holde seg nært midten av neste pipene vertikalt.
Kolliderer den i hinderet får den –1 poeng.

4: DEPLOYMENT

> *Hvordan skal modellen(e) settes i drift? Hvordan skal prediksjonene brukes? Hva er dine planer for monitorering og vedlikehold av maskinlæringssystemet? Hvis relevant, hvilke planer har du for å forbedre systemet etter at det er satt i drift?*

Modellen vår settes i drift ved å integreres direkte i en webapplikasjon som vi hoster hos Streamlit. Webapplikasjonen kjører spillet og henter handlinger fra en av de trente RL-modellene våre i sann tid.
Prediksjonene foregår for hver frame og i følgende ordning:

1. Henter observasjon fra miljøet; fuglehøyde, fart, avstand til neste rør.
2. Sender observasjonen inn i den trente PPO-modellen.
3. Modellen returnerer en handling; flapp / ikke flapp.
4. Handlingen sendes tilbake og simulerer neste steg.

Prediksjonene brukes ikke bare i bakgrunn, men visualiseres i spillet. Målet er å vise at vår trente PPO-agent har lært seg å spille spillet bedre enn baseline modellen vår.
Potensielle forbedringer vi kan gjøre er å trene agenter enda flere timestamps. Eventuelt trene agenter der pipene har smalere mellomrom vertikalt. Alt i alt, så er vi veldig fornøyd med utfallet av modellene våre og hvor god den mest trente modellen er.

5: REFERANSER

https://medium.com/@danushidk507/ppo-algorithm-3b33195de14a
https://flappybird-ai.netlify.app/
https://en.wikipedia.org/wiki/Flappy_Bird
https://chatgpt.com/
https://github.com/DLR-RM/stable-baselines3

Starte app:

python -m venv .venv

source .venv/Scripts/activate

streamlit run app/app.py