LC4

## Q Learning

Hand-in: verslag en R-code  
Teams: Tweetallen binnen de eigen practicumgroep

Het doel van deze practicum opdracht is dat je jezelf bekend maakt met de werking van Q-Learning en het exploration-exploitation dilemma. Het eindresultaat is een verslag waarin je het model uitlegt en illustreert aan de hand van een aantal simulaties. Voor deze opdracht zullen we ook echte data fitten. Beantwoord de onderstaande hoofdvragen en verwerk deze tot een lopend verhaal in het verslag. De onderstaande sub-vragen dienen daarin verwerkt te worden.

# 

# Opdracht

Schrijf een rapport met de volgende onderdelen:

1. Leg uit wat het Q-learning model is en wat de variabelen betekenen, en hoe die impact hebben op het leergedrag van het model (3 punten)
2. Laat zien wat de impact is van verschillende exploration-exploitation regels. ( 4 punten)
3. Fit het Q-learning model op de data van een proefpersoon. (3 punten)

Geef bij al je illustraties ook aan hoe je een en ander in R uitgerekend of geprogrammeerd hebt. Maak gebruik van duidelijke functies en (for-)loops. Als het meerwaarde heeft voor het beantwoorden van de vragen, dan kan je de code toevoegen aan het verslag (gebruik dan de {r , eval = F} blocks).

De onderstaande uitleg, (programmeer) vragen en opdrachten zijn bedoeld om je op weg te helpen bij het schrijven van het rapport. Het is noodzakelijk om alle vragen te beantwoorden voor een goed verslag en het is de bedoeling dat je het beschreven Markov Decision Proces (MDP) als model neemt. Zorg dat bij figuren altijd duidelijk aangegeven is welke model parameters horen bij de geplotte data. Vergeet niet de random seed op 123 te zetten (set.seed(123)). Elke vraag (Q#) is een punt waard.

# Q-Learning

Q-learning algorithmes zijn gebaseerd op de TD-learning modellen die we eerder hebben besproken. Q-learning leert direct de associatie tussen states, actions en outcomes.

Details over Q-learning zijn terug te vinden in de college slides en het hoofdstuk van Love & Gureckis.

Het leren in deze opdracht speelt zich af in een simpel **Markov Decsion Process** met de volgende structuur:



In deze taak moet de robot telkens uit een van de schatkisten iets pakken. In sommige schatkisten zit meer geld dan in anderen, maar de robot weet in het begin nog niks over de schatkisten, en verwacht er maar weinig van. In elke ronde wordt uitkomst van een schatkist bepaald door een trekking van een waarde uit een normaalverdeling ( rnorm())

De schatkisten verschillen in gemiddelde (mean) maar niet in variatie (standard deviatie) van uitkomsten.

**Let op.** in dit simpele experiment is er maar een state, waarin de robot telkens terugkeert na het maken van een keuze. Dit heeft als gevolg dat bij het leren geen rekening gehouden hoeft te worden met de actie in de volgende state gemaakt wordt. De standaard prediction-error:

 = *rt*+1 +  max *Q* (*st*+1 , *a*) − *Q* (*st* , *at* )

verandert dus simpelweg in:

 = *rt*+1 − *Q* (*st* , *at* )

In het begin van het experiment heeft de robot geen enkele kennis van de wereld en geen enkele verwachtingen voor van het krijgen van beloningen. Voor elke schatkist geldt :

Q(1)=Q(2)=Q(3)=Q(4)=0.

**Q1.** Schrijf de Q-learning functie op die de nieuwe waarde Q uitrekent nadat de robot een schatkist heeft uitgekozen. Welke vrije variabelen heeft deze functie?

**Q2.** Stel de robot selecteert schatkist #1 en vindt twee munten. Wat is hierna de waarde van Q(1) rapporteer dit voor 2 verschillende waardes van de vrije variabelen. 

**Q3** Schrijf een functie Q\_LEARN die als input, alpha (), epsilon() en rondes (trials) accepteert. De output van deze functie moet een lijst met (Q) waarden zijn voor elke schatkist (1 t/m 4) in de wereld van de robot na het leren van een aantal rondes. We gaan er nu van uit dat de robot de -greedy keuze regel toepast.

* initieer de verwachtingen van de robot voor de 4 keuzes:
  + Q(1)=Q(2)=Q(3)=Q(4)=0.
  + Bijvoorbeeld: Q\_list<-rep(0,4)
  + Intieer total\_score <- 0
* initieer de beloningen voor de 4 schatkisten:
  + K(1): mean= 20, SD= 4
  + K(2)= mean= 30, SD= 4
  + K(3)= mean= 50, SD= 4
  + K(4)= mean= 70, SD= 4
* Creëer een for-loop over alle rondes:
  + Elke ronde selecteert de robot een kist op basis van -greedy. Denk er aan bij explore een random kist, bij exploit de kist met hoogste Q value (als er meerdere de hoogste waarde hebben, daar weer random uit kiezen).
  + Kijk wat de beloning is na maken van een keuze, en update dan Q

Laat me behulp van deze functie de robot 200 rondes leren over deze wereld (1 leer episode bestaat dus uit 200 rondes). Hoe zien zijn verwachtingen (Q-values) voor de schatkisten aan het eind van het experiment er uit gegeven α = .1 en  .1En voor α = .5 en .Wat is er dichter bij de waarheid?

**Q5** Laten we nu kijken hoe succesvol de verschillende strategieën (lees combinatie van parameter waardes) zijn. Zorg ervoor dat de Q\_LEARN ook het totaal van verzamelde punten opslaat.

Schrijf nu een loop die Q\_LEARN 500 keer aanroept met een bepaalde parameter setting (en altijd 200 leerrondes) en sla telkens het totaal aantal punten op, zodat je aan het eind een lijst hebt van 200 totaal scores.

Vergelijk het gemiddelde van die 500 totaalscores voor [(α = .1 en  .1),( α = .5 en .), (α = .8 en  .1)], en leg uit hoe de verschillen tot stand komen.

**Exploration-Exploitation**

We gaan nu dieper in op het exploration vs. exploitation dilemma. Laten we eerst kijken hoe verschillende e parameters uitwerking hebben op het aantal punten dat gewonnen wordt. Gebruik hier de functie van Q\_LEARN van de vorige vraag. Verander de  van .05 , .2 en .6 (en een alpha () van .3).

**Q6** Kijk voor elke parameter setting weer naar de gemiddelde totaal score van 500 leer episodes. Waar ligt ongeveer het optimale niveau van exploratie?

**Q7** Pas nu de Q\_LEARN zo aan dat de parameter  gedurende een leer episode steeds kleiner wordt. Dit kan bijvoorbeeld door elke ronde met een vast percentage te verkleinen (denk aan iets tussen 0 en 10%), maar andere manieren zijn ook mogelijk. Sla dit model op als Q\_LEARN\_decay

**Q8** Kijk nu naar een aantal begin waarden voor parameter , en kijk of dit model meer punten kan verdienen in de taak (verken hier ook hoge waarden van ). Gebruik weer het gemiddelde aantal punten over 500 leer episodes (en nog steeds 200 rondes per episode en alpha () van .3).

**SOFTMAX**

Een andere zeer populaire methode om exploratie te implementeren is de softmax regel. Deze kan gebruikt worden om de waarschijnlijkheid uit te rekenen dat de robot een specifieke kist kiest. Bijvoorbeeld de waarschijnlijkheid dat de robot op een bepaald moment kist 1 kiest is:

*P(Q(1)) = e*(Q(1)\*theta) / ( *e*(Q(1)\*theta) + *e*(Q(2)\*theta) + *e*(Q(3)\*theta) +*e*(Q(4)\*theta) )

Voor kist 2:

*P(Q(2)) = e*(Q(2)\*theta) / ( *e*(Q(1)\*theta) + *e*(Q(2)\*theta) + *e*(Q(3)\*theta) +*e*(Q(4)\*theta) )

En natuurlijk:

*P(Q(1))+ P(Q(2)) +P(Q(3))+ P(Q(2)) = 1 (100%)*

want de robot kiest altijd een van de 4 opties, dus samen moeten dat 100% kans zijn.

Implementeer nu de softmax regel in de Q\_LEARN functie en geef deze de naam Q\_LEARN\_SMAX. Zorg dat deze elke ronde de waarschijnlijkheid P(Q) van het kiezen van elke kist wordt opgeslagen zodat we hier later weer naar kunnen kijken. Gebruik ook elke ronde de P(Q) informatie om de robot een kist te laten kiezen.

Gebruik dit model om verschillende waardes van theta ( tussen .01 en 1) met elkaar te vergelijken (op zn minst 5), gebruik makende van de methode die we hier boven ontwikkeld hebben.

**Q9** Wat is ongeveer de optimale waarde voor theta, hoe verhoud dit model zich tot de simpele versie van -greedy?

**Model fitting**

We gaan nu kijken naar de resultaten van een echt experiment. We hebben de data van een proefpersoon die het bovenstaande experiment heeft gespeeld. In de data file kunnen we terugvinden welke van de 4 opties zij gekozen hebben en hoeveel punten vervolgens kregen. We gaan kijken hoe goed Q-learning met s choice rules het gedrag van de proefpersonen kan voorspellen, en wat voor een parameter waardes dit oplevert.

Lees de data in:

data<-read.delim("L4\_data\_1.txt")

Pas de Q\_LEARN\_SMAX functie aan zodat je deze op de data van de proefpersonen kan fitten. (Zie LC1 voor details over hoe functies gefit moeten worden).

Ga er van uit dat de proefpersonen enige ervaring hebben met dit type experiment en verwachten dat ze gemiddeld wel 30 punten per ronde gaan verdienen (alle Q’s starten op 30 ipv 0).

We gaan voor het fitten van het model gebruik maken van **optim()** en we gaan proberen Log Likelihood te maximaliseren.

Wat we op elke trial willen weten is wat de waarschijnlijkheid is dat het model dezelfde keuze maakte als de proefpersoon. Hoe waarschijnlijker dat is hoe beter het mode fit.

In het databestand van de proefpersoon kunnen zien welke van de 4 kisten de proefpersoon koos. Dit kunnen we dan op elke ronde vergelijken met de corresponderende P(Q). In de eerste ronde zijn alle Q values nog gelijk dus zijn alle P(Q)s ook .25.

Voor de eerste ronde geld daarom automatisch dan waarschijnlijkheid (likelihood) van de keuze van de proefpersoon ook .25 is, maar dat gaat veranderen naarmate er geleerd wordt.

In het data bestand staat ook voor elke ronde wat de uitkomst van een keuze was, deze moet gebruikt worden om vervolgens de Q-values up te daten, net als we eerder gedaan hebben.

De output van deze functie moet de som van alle log(P(Q(chosen))) zijn. Let op vermenigvuldig deze som met -1. Dat doen we omdat **optim()** de functie probeert te minimaliseren we opzoek zijn naar de max LL.

**Q10 Welke parameterwaarden fitten de data van de proefpersoon het beste? Probeer eens een andere start waarde voor Q values (niet 30) en hoe beinvloed dit de model fit?**

**Q11 Herschrijf de functie nogmaals en zorg er voor dat de initialisatie van de Q waarde (de start Q waarden) ook een vrije parameter wordt. Als je dit doet kan je de start waarde vinden die het best bij het gedrag van de proefpersoon past. Rapporteer deze waarde.**

**Bonus -> Q11 Herschrijf de functie nogmaals en zorg er voor dat er twee learning rates zijn, een voor uitkomsten die slechter zijn dan verwacht (negative prediction error) en een voor uitkomsten die beter zijn dan verwacht (positive prediction errors). Fit dit model beter of slechter?**

Pseudo code for fitting model to help you along ( add lines if necessary and fill in code next to “ <- “ :

data<-read.delim("L4\_data\_1.txt")

Q\_LEARN\_SMAX\_FIT<-function(par){

nRounds<-nrow(data)

alpha<- par[1] #learning rate from wins

theta<- par[2] #exploratoin

init<-30

# int weights

Q\_list<-rep(init,nBandits)

likes<-c() # empty list for the LL

for (i in 1:nRounds) {

## Use softmax to calculate probability of chosing each option:

Prob1 <-

Prob2 <-

Prob3 <-

Prob4 <-

# find choice

choice <- data$choice[i]

outcome <- data$outcome[i]

## store the probability of only chosen option in the likes() list you made before

switch(choice,likes[i]<-Prob1,likes[i]<-Prob2,likes[i]<-Prob3,likes[i]<-Prob4)

# now update Q value of chosen option based on feedback!

Q\_list[choice]<-

}

## determine summed log like

LL<-sum(log(likes))

#transform to G2 (we minimize function so it should return a negative number)

G2=-2\*LL

return(G2)

}

# to fit

optim(c(.5,.5), fn = Q\_LEARN\_SMAX\_FIT, method = 'L-BFGS-B',lower = c(0,0),upper = c(1,10))