Kompilera kvantdatorkod med förstärkningsinlärning

Slutrapport: TIFX04-22-20

Chalmers Tekniska Högskola

Institutionen för fysik

Författare:
Carl Strandby
Nils Jakobson Mo
Kári Einarsson
Emrik Lindahl

Handledare:
Mats Granath

12 maj 2022

Kompilera kvantdatorkod med förstärkningsinlärning
Carl Strandby, Nils Jakobson Mo, Kári Einarsson, Emrik Lindahl

E-post:
carstran@student.chalmers.se
nilsjak@student.chalmers.se
karie@student.chalmers.se
emrikli@student.chalmers.se

Handledare: Mats Granath
Examinator: Lena Falk

Kandidatarbete 2022: TIFX04-22-20
Institutionen för fysik
Chalmers tekniska högskola
SE-412 96 Göteborg
Tel: +46 031-772 10 00

Typsatt i LaTeX

Sammandrag

Kvantdatorer förutspås kunna lösa vissa beräkningsintensiva problem som klassiska digitala datorer idag inte kan lösa inom en rimlig tidsram. Ett problem med att exekvera kvantkod är att informationen på en kvantbit har en begränsad livslängd. Det är därför viktigt att exekveringen sker med så få kvantgrindar som möjligt. Kvantdatorns kompilator är också begränsad till att endast exekvera kvantgrindar på kvantbitstillstånd som är länkade på kretsens fysiska arkitektur. Därför behöver kompilatorn tillföra SWAP-grindar för att byta plats på kvantbitstillstånden. Detta tillför exekveringstid och placeringen av dessa SWAP-grindar kan ofta göras på många olika sätt, vilket gör att det är svårt att beräkna en optimal placering. Syftet med detta kandidatarbete är därför att utforska om placering av SWAP-grindar kan utföras med en formulering av djup förstärkningsinlärning som värderar tillstånd istället för handlingar.

Projektet har utförts genom att programmera en miljö för djup förstärkningsinlärning utifrån en matrisrepresentation av kvantkod och kvantkretsar. Ett artificiellt neuralt nätverk av faltningstyp har designats för att läsa in ett framtida möjligt tillstånd för en kvantkrets och värdera detta. Förstärkningsinlärningsagenten använder sedan denna värdering för att välja hur den ska placera SWAP-grindar. Förstärkningsinlärningsmodellen bygger på den öppna programvaran Stable Baseline3 som använder OpenAI Gym för miljön och PyTorch för att konstruera faltningsnätverket.

Modellen användes sedan för att träna två förstärkningsinlärningsagenter för två olika storlekar av kvantkretsar. Agenterna testades sedan och lyckades med att göra en representation av kvantkod exekverbar. Deras resulterande kvantkod uppmättes dessutom endast vara något sämre än kvantbitdirigeringsprogrammet TKET. Detta antyder att värdering av kvantkretstillstånd med neurala nätverk mycket väl skulle kunna användas för att dirigera kvantbitar.

Nyckelord: Djup förstärkningsinlärning, Kvantbitar, Kvantkod, Kvantkrets, Faltningsnätverk, Stable Baselines3, OpenAI Gym, DQN, SWAP-grind, TKET

Abstract

Quantum computers are predicted to be able to solve computationally demanding problems which classical computers are not yet able to solve in a reasonable amount of time. One problem with the execution of quantum code is that the information on a quantum bit has a limited lifespan. Hence it is important that execution of quantum code is done with as few quantum gates as possible. The compiler for a quantum computer is also limited by the fact that it has to execute quantum gates on the qubits that are linked on the physical architecture of the quantum circuit. That is why the compiler needs to add SWAP gates to change the location of the quantum states. The additional execution time and placement of these SWAP gates can be done in multiple ways, which makes it complex to calculate the optimal placement. The scope of this Bachelor project is therefore to explore if the placement of the SWAP gates can be handled by formulating a deep reinforcement learning algorithm that uses a value of the state instead of the action.

The project use a programmed environment for the deep reinforcement learning based

on matrix representations of quantum code and quantum circuits. An artificial neural network that uses convolution has been designed to use a possible future state for a quantum circuit and assign a value to it. The agent then uses this value to decide where to place the SWAP gates. The reinforcement learning model was made using the open source software Stable Baselines3 which uses OpenAI Gym for the environment and PyTorch to construct the convolutional neural network.

The model was then used to train two agents for two different sizes of quantum circuits. The agents were tested and was successful in placing SWAP gates in quantum code so that it could be executed. The resulting quantum code was measured to be only slightly worse that the qubit routing program TKET. This suggest that using the value of a quantum circuit with neural networks is a viable option in quibit routing.

Keywords: Deep Reinforcement Learning, Qubits, Quantum Code, Quantum Circuit, Convolutional Neural Network, Stable Baseilnes3, OpenAI Gym, DQN, SWAP Gate, TKET

Förord

Tack till vår handledare Mats Granath och Ricardo Zamora Solis som hjälpt till att guida oss igenom arbetet. De har varit en otroligt stor hjälp och hållit gruppens motivation och stämning hög genom hela arbetet samt gjort hela arbetet väldigt roligt och intressant. Tack igen!

Ordlista

Kvantgrind = Avser i detta arbete en kvantgrind i kvantkoden som ska utföras mellan två kvantbitar.
SWAP-grind = Avser tillförda CNOT-grindar som byter plats på två noders kvantbitar.
Noder = De fysiska placeringarna av kvantbitar i en kvantkrets
Observation = Från engelskans Observation
Tillplattning = Från engelskans Flatten
Utforskningsfrekvens = Från engelskans Exploration rate
Neuronmatris = Från engelskans Feature map

Innehåll

1 Inledning

Forskare på Chalmers tekniska högskola i Göteborg lyckades år 2020 lösa ett mindre optimeringsproblem med en egenbyggd kvantdator [1]. Forskningslaget verkade inom Wallenberg Centre for Quantum Technology, WACQT, ett tolvårigt forskningsprojekt med en total budget på över 1,3 miljarder kronor [2]. Branschen för tillverkning och användning av kvantdatorer förutspås uppgå till 450-850 miljarder dollar de närmsta 15-30 åren [3]. Orsaken till de stora forskningsanslagen och höga värderingarna är att en kvantdator förutsägs kunna lösa vissa problem som klassiska digitala datorer inte kan lösa inom en rimlig tidsram [4]. Men utvecklingen befinner sig ännu i ett tidigt stadium och det finns många problem kvar att lösa innan kvantdatorer blir kommersiellt tillgängliga.

Ett sådant problem är att en kompilator som exekverar kod på en kvantkrets är begränsad till att endast operera mellan kvantbitar som är länkade till varandra på kretskortet [5]. Om kod ska utföras på kvantbitar som inte befinner sig på länkade noder måste kompilatorn dirigera kvantbitarna. Detta görs med så kallade SWAP-grindar som byter plats på två noders respektive kvantbitar. Problematiken med att dirigera kvantbitar är att exekveringen av SWAP-grindar tar tid, och det är därför viktigt att dirigeringen sker så effektivt som möjligt.

I detta kandidatarbete utvecklar vi en alternativ metod för att dirigera kvantbitar där djup förstärkningsinlärning används för att värdera en kvantkrets exekverbarhet givet en kvantkod. Vår metod bygger vidare på forskning från Pozzi, Sengupta, Herbert och Mullins som också använder djup förstärkningsinlärning, men istället använder den för att värdera vilken placering av SWAP-grindar som är den optimala[5]. Djup förstärkningsinlärning har också använts tidigare för kvantfelskorrektion av Andreasson, Johansson, Liljestrand, Granath [6]; samt av Fitzek, Eliasson, Kockum och Granath [7].

Syftet med detta kandidatarbete är att skapa en djup förstärkningsinlärningsmodell som kan värdera en kvantkrets exekverbarhet i relation till en kvantkod som ska exekveras. Denna värdering ska användas av en förstärkningsinlärningsagent för att dirigera kvantbitar i en miljö som kan simulera en sekventiell kvantkod.

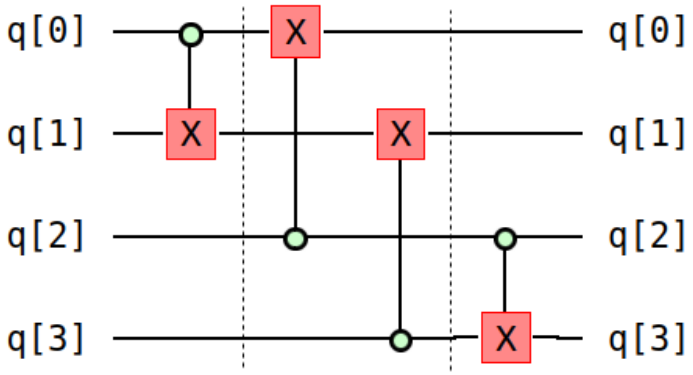
2 Teori

2.1 Kvantdator

En kvantdator kan vid en första anblick te sig relativt lik en klassisk dator. Båda låter en användare ange indata, och designa en funktion som opererar på informationen med datorns grindar. Skillnaden är att en kvantdator behandlar information annorlunda än en klassisk dator. Dels så kan en kvantbit anta värdet av en linjärkombination av baserna 1 och 0 medan en klassisk bit endast kan anta ett av de diskreta värdena 1 och 0. Vidare kan kvantsammanflätning ge interferens som kan användas för att beräkna en fouriertransform [8]. Detta fenomen används i Shors algoritm för primtalsfaktorisering och ökar markant beräkningshastigheten jämfört med algoritmer för klassiska datorer. Något som medför stora praktiska konsekvenser då en stor del av modern kryptering bygger på primtalsfaktorisering.

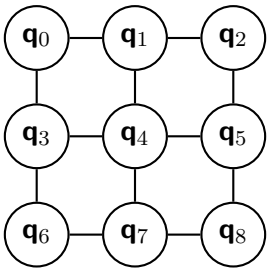
2.2 Kvantkod och kvantkretsar

En sekventiell kvantkod beskriver hur en följd av operationer som kallas kvantgrindar utförs på en eller två kvantbitar. Eftersom att en kvantbit endast kan användas i en kvantgrind i taget delas kvantkod upp i ett antal tidssteg. Varje tidssteg beskriver då ett antal kvantgrindar som kan utföras parallellt utan att en kvantbit används av två olika kvantgrindar samtidigt, se figur 1. Det minsta antal tidssteg en kvantkod kan brytas ned i definierar koddjupet d.



Figur 1: En representation av sekventiell kvantkod som består av fyra kvantgrindar som kan delas upp i tre tidssteg. Kvantkoden har därför djupet tre.

En kompilator för sekventiell kvantkod är dessutom begränsad till att endast exekvera kvantgrindar på noder som är länkade till varandra på kvantkretsens fysiska arkitektur. Därför beskrivs en kvantkrets ofta med ett diagram som visar vilka noder som är länkade till varandra, samt vilka kvantbitar som befinner sig på vilka noder. Ett sådant diagram för en kvantkrets med nio noder ses i figur 2. För rektangulära representationer av kvantkretsar är det brukligt att kretsens storlek anges med notationen (#rader) $\times$ (#kolumner).



Figur 2: Diagram över kvadratisk kvantkrets av storlek 3×3 som visar hur nio kvantbitar är fördelade över nio noder och hur dessa noder är länkade till varandra. Noderna ritas som cirklar och kopplingarna mellan noderna är de vertikala och horisontella streck som ses i figuren. De nio kvantbitarna är numrerade från q_0 till q_8 och inskrivna i cirklarna motsvarande de noder kvantbitarna befinner sig på.

Om en kompilator ska utföra en kvantgrind på två kvantbitar som inte befinner sig på länkade noder benämns kvantkretsen befinna sig i ett icke-exekverbart tillstånd. Kompilatorn måste då dirigera kvantbitarna till noder som är länkade för att kretsen ska befinna

sig i ett exekverbart tillstånd. Detta görs med så kallade SWAP-grindar. En SWAP-grind kan endast utföras på två länkade noder och byter då plats på nodernas kvantbitar. Om det inte går att utföra en SWAP-grind parallellt med en annan kvantgrind behöver SWAP-grinden utföras i ett eget tidssteg. Således ges hur effektiv en kompilator är på att optimera en kvantkrets utifrån hur många tidssteg som adderas när kompilatorn tillför SWAP-grindar. Ett mått på denna effektivitet är koddjupstillägg,

$$CDO = d(c') - d(c), \tag{1}$$

där CDO kommer från engelskans benämning circuit depth overhead, d är koddjupet och den ursprungliga kvantkoden benämns c medan kvantkoden med tillförda SWAP-grindar benämns c'. Ett annat vanligt mått är koddjupsförhållande som ges av ekvationen

$$CDR = \frac{d(c')}{d(c)}, \tag{2}$$

där CDR kommer från engelskans circuit depth ratio.

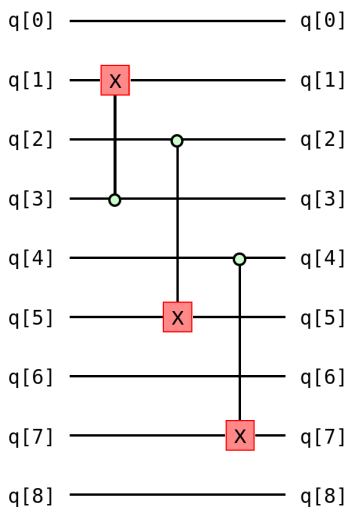
2.3 Matrisrepresentation av kvantkod

En metod för att representera en kvantkrets så att denna kan användas i datorprogram är att representera kvantkretsdiagram och kvantkod med matriser. Till exempel kan ett kvantkretsdiagram, som den i figur 2, beskrivas med matrisen respektive dess tillplattade kolonnvektor

$$\begin{bmatrix} q_0 & q_1 & q_2 \\ q_3 & q_4 & q_5 \\ q_6 & q_7 & q_8 \end{bmatrix} \Longleftrightarrow \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \\ q_7 \\ q_8 \end{bmatrix}. \tag{3}$$

Vidare kan ett tidssteg, där en eller flera kvantgrindar ska exekveras, beskrivas med en observationsmatris. Matriselementen beskriver både antal och de kvantbitar som kvantgrindar ska utföras på. Elementens placering i observationsmatrisen motsvarar kvantbitarnas placering i kvantkretsmatrisen ovan. De båda matriserna har därför samma storlek. Om ingen kvantgrind ska utföras på kvantitet tilldelas observationsmatrisens element värdet 0. Kvantbitar som kvantgrindar ska utföras på indexeras med heltal större än 0 och parvis för kvantgrindar mellan två kvantbitar. Antag till exempel att kvantbitarna befinner sig på noder enligt ekvation 3 och att kvantkoden som ses i figur 3 ska exekveras. Då beskrivs ett tidssteg där tre kvantgrindar ska utföras, en på kvantbitarna q_1 och q_3, en på q_4 och q_7, samt en på q_2 och q_5, med observationsmatrisen

$$\begin{bmatrix} 0 & 1 & 3 \\ 1 & 2 & 3 \\ 0 & 2 & 0 \end{bmatrix}. \tag{4}$$



Figur 3: En representation av den sekventiella kvantkod som beskrivs av observationsmatrisen i ekvation 4.

Notera att denna observationsmatris beskriver ett icke-exekverbart tillstånd eftersom att kvantbitarna q_1 och q_3 inte befinner sig på noder som är grannar. Givet att kvantkretsens noder i detta fall inte är länkade diagonalt går det även att avläsa direkt från observationsmatrisen, att denna beskriver ett icke-exekverbart tillstånd eftersom att de två ettorna inte befinner sig vertikalt eller horisontellt intill varandra.

En SWAP-grind, eller flera parallella SWAP-grindar, som exekveras i ett tidssteg kan beskrivas med en permutationsmatris P. För en kolonnvektor $\mathbf{g}$ som representerar en kvantkrets med n noder har permutationsmatrisen storleken $n \times n$. Exekveringen av SWAP-grindarna representeras då av matrismultiplikationen

$$P\mathbf{g} = \begin{bmatrix} P_{11} & \dots & P_{n1} \\ \vdots & \ddots & \vdots \\ P_{1n} & \dots & P_{nn} \end{bmatrix} \begin{bmatrix} g_1 \\ \vdots \\ g_n \end{bmatrix} = \begin{bmatrix} g_P(1) \\ \vdots \\ g_P(n) \end{bmatrix}. \tag{5}$$

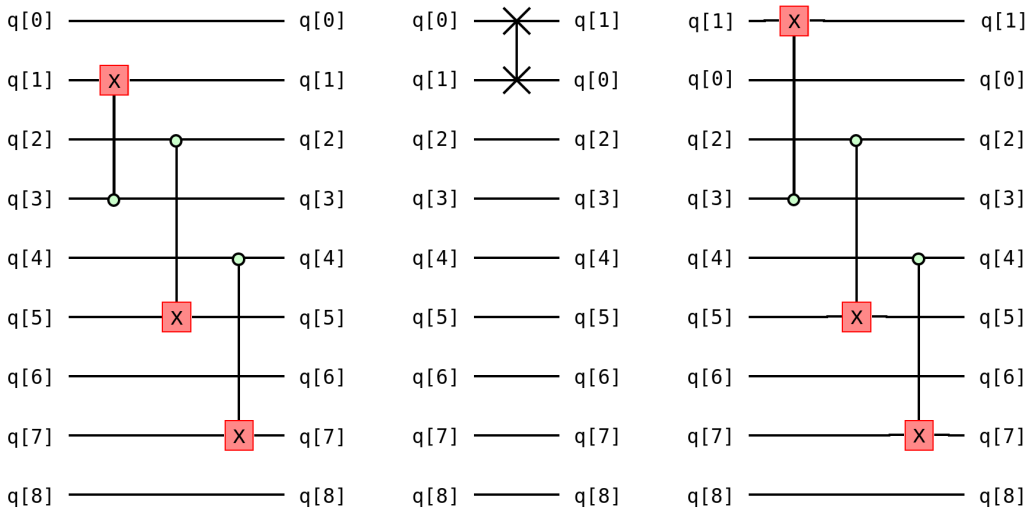
Exempelvis skulle det icke-exekverbara tillståndet beskrivet ovan i ekvation 4, bli exekverbart om en SWAP-grind utförs på kvantbitarna q_0 och q_1. Notera även att ett byte mellan q_0 och q_3 skulle göra tillståndet exekverbart. SWAP-grinden på kvantbitarna q_0 och q_1 ges av matrismultiplikationen

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 3 \\ 1 \\ 2 \\ 3 \\ 0 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 3 \\ 1 \\ 2 \\ 3 \\ 0 \\ 2 \\ 0 \end{bmatrix}. \tag{6}$$

Observationsmatrisen genomgår då transformationen

$$\begin{bmatrix} 0 & 1 & 3 \\ 1 & 2 & 3 \\ 0 & 2 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 3 \\ 1 & 2 & 3 \\ 0 & 2 & 0 \end{bmatrix}, \tag{7}$$

och eftersom att de två ettorna nu befinner sig intill varandra beskriver den nya observationsmatrisen ett exekverbart tillstånd. Hur transformationen påverkar en sekventiell kvantkod ses i figur 4.



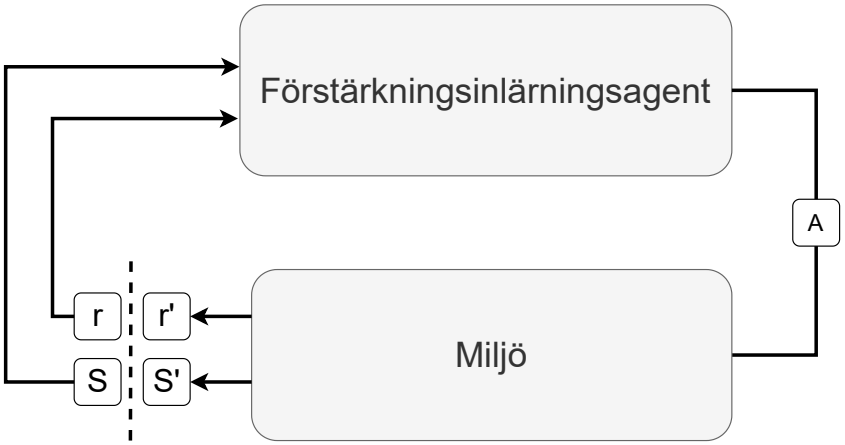
Figur 4: Transformationen som beskrivs av matrsimultiplikationen i ekvation 6 representerad som sekventiell kvantkod. Till vänster ses tillståndet innan exekvering av SWAP-grinden, vars placering ses i mitten. Till höger ses tillståndet efter exekvering där kvantbiten på nod 1 har flyttats till nod 0.

2.4 Djup förstärkningsinlärning

Förstärkningsinlärning är en metod inom maskininlärning där tillstånd s, handlingar a, och belöningar r, används för att träna en agent. Agenten placeras i en miljö som beskriver hur en given handling påverkar ett tillstånd, samt vilken belöning som utdelas för att utföra handlingen, se figur 5. En gynnsam handling ger en positiv belöning och en icke-gynnsam ger en negativ. Utöver detta tilldelas varje möjlig handling ett Q-värde med Bellmanekvationen

$$Q(s, a) = r(s, a) + \gamma Q(s', a'). \tag{8}$$

Detta värde beskriver väntevärdet av belöningen. Den handling med det högsta Q-värdet är således den handling som ger den högsta slutgiltiga belöningen. En optimal beslutspolicy för en förstärkningsinlärningsagent är därför att för varje tillstånd utföra handlingen med högst Q-värde.



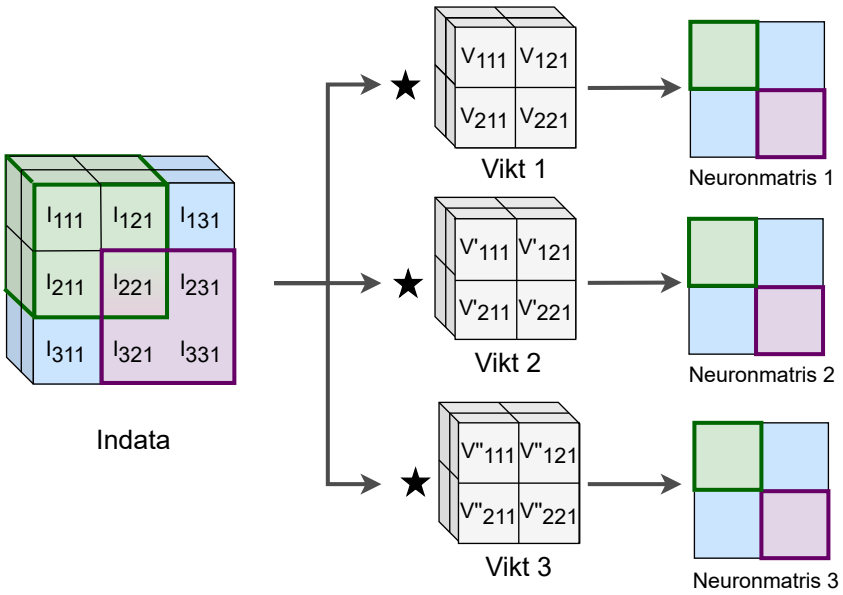
Figur 5: En förstärkningsinlärningsagent som läser in ett tillstånd och en belöning. Agenten beräknar vilken handling som är optimal och som sedan används som indata till miljön.

Förstärkningsinlärningsagenten behöver därmed känna till alla Q-värden för att fatta optimala beslut. Skulle dock antalet möjliga handlingar vara stort så blir det snabbt beräkningsmässigt intensivt att beräkna Q-värdet för varje handling. Då är det istället lämpligt att använda djup förstärkningsinlärning.

Djup förstärkningsinlärning bygger på att varje Q-värde först tilldelas ett preliminärt värde med hjälp av ett neuralt nätverk. Agenten tillåts därefter testa olika handlingar i miljön och uppdatera både Q-värden och det neurala nätverket. Ofta är det neurala nätverket ett faltningsnätverk, se avsnitt 2.5.

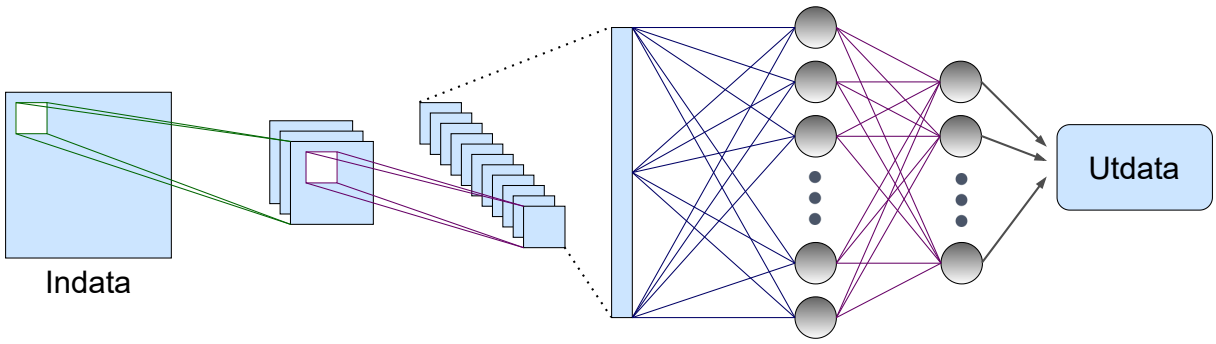
2.5 Faltningsnätverk

Ett faltningsnätverk är ett neuralt nätverk som byggs upp i flera skikt där ett antal filter används för att känna igen olika mönster i indatan. Principen för ett faltningsskikt är att ett filter sveper över indatan i små steg och beräknar, ofta med en korskorrelationsprodukt, ett viktat värde av informationen den finner [9]. Det nya skiktet behöver inte ha samma dimension som indatan, se figur 6, men för att skapa ett faltningsskikt med samma dimension som indatan kan dess representation utvidgas med en eller flera rader eller kolonner med nollor runtom matrisen. Informationen i faltningsskiktets neuronmatriser kan sedan användas som indata i ytterligare ett faltningsskikt och processen upprepas. Notera att varje neuronmatris i det nya faltningsskiktet är anslutet till varje neuronmatris i det föregående skiktet [10].



Figur 6: En tredimensionell faltningsfunktion kan skapa ett godtyckligt antal, i denna figur tre stycken, tvådimensionella neuronmatriser från en tredimensionell indata. Varje kvadrant i respektive neuronmatris innehåller ett viktat värde av indatan motsvarande kvadrant, illustrerat av den gröna cellen i den andra kvadranten och den lila cellen i den fjärde.

I ett faltningsnätverk används ett tillplattningsskikt för att skapa en kolonnvektor efter det sista faltningsskiktet. Sedan läses informationen in av ett neuronskikt där varje neuron läser in och tilldelar de enskilda värdena i kolonnvektorn en vikt. Detta kallas för ett fullt anslutet skikt [10]. Till detta skikt kan ett nytt fyllt anslutet neuronskikt anslutas. Det sista neuronskiktet används ofta för att ge utdata. Figur 7 visar hur ett sådant faltningsnätverk kan se ut.



Figur 7: Faltningsnätverk med två faltningsskikt och två fullt anslutna neuronskikt. Längst till vänster ses en ruta som symboliserar indatan av information. Informationen faltas sedan i två skikt där varje neuronmatris i skiktet har en egen associerad vikt. Efter det andra skiktet transformeras informationen till en kolonnvektor. Ett fullt anslutet neuronskikt hämtar sedan informationen från kolonnvektorn, och är i sin tur anslutet till ytterligare ett fullt anslutet neuronskikt, som används för att ge utdata. Under träning av ett faltninsnätverk justeras vikternas värden kontinuerligt för att förbättra nätverkets prestation.

Ett faltningsnätverk kan användas för att bilda en beslutspolicy för en förstärkningsinlärningsagent. Indata i nätverket kan då vara ett tillstånd och utdata en handling som ges av neuronen med högst värde. Inledningsvis tilldelas nätverkets vikter slumpmässiga värden och det är därför slumpmässigt vilken neuron som väljs. Nätverket kan sedan tränas så att den valda neuronen motsvarar den mest optimala handlingen. Idén bakom träningen är att skapa två till början identiska nätverk. Ett målnätverk, CNN_T, som används av förstärkningsinlärningsagenten för samla in erfarenheter som sparas i en buffert. Det andra är ett kontinuerligt uppdaterat nätverk, CNN_C, som tränas med hjälp av dessa erfarenheter. En förlustfunktion ger skillnaden mellan nätverkens prestation, och används för att justera vikterna i det kontinuerligt uppdaterade nätverket. Dessa justerade vikter används i sin tur för att med jämna intervall uppdatera målnätverkets vikter.

3 Metod

I detta avsnitt definieras först problemformuleringen för vårt kandidatarbete i avsnitt 3.1. Sedan behandlas den mjukvara och de verktyg som använts under arbetets gång i avsnitt 3.2. Det framtagna programmets funktionalitet beskrivs i avsnitt 3.3. Slutligen behandlas verktyg för att kontrollera träningsförlopp och agenters handlingar i avsnitt 3.4.

3.1 Problemformulering

Projektets mål är att använda en modifierad djup förstärkningsinlärning för att träna en agent som kan dirigera kvantbitar med SWAP-grindar på en kvantkrets av storlek 3×3. Den modifikation som undersöks är att istället för att beräkna Q-värden för handlingar enligt

$$Q(s, a) = r(s, a) + \gamma \operatorname{argmax}(Q(s', a')), \tag{9}$$

beräkna värdet för olika observationsmatriser med ekvationen

$$V(s) = \operatorname{argmax}(r(s, a) + \gamma V(s'(a))). \tag{10}$$

En framgångsrikt tränad förstärkningsinlärningsagent definieras av en agent som kan läsa in en representation av sekventiell kvantkod bestående av 40 kvantgrindar mellan två kvantbitar, och placera ut SWAP-grindar som gör koden exekverbar. Med exekverbar kod avses att det finns ett tidssteg då kvantkretsen befinner sig i ett exekverbart tillstånd för varje kvantgrind i följden av den sekventiella representationen av kvantkod. Kvantbitarna ska först befinna sig på de noder som krävs för att första observationsmatrisen ska vara exekverbar, därefter på noder som gör den andra exekverbar och så vidare tills den tionde och sista operationen är exekverbar. Ett gott resultat för kvantkretsar av storlek 3×3 definieras som en agent vars resulterande kod har ett koddjupsförhållande till den ursprungliga koden som är som mest 50 procent större än motsvarande metrik för kod producerat av ett jämförbart kvantbitsdirigeringsprogram.

3.2 Mjukvara och verktyg

Arbetet har utförts med hjälp av programspråket Python. Träningsmiljön baserades på OpenAI Gym och förstärkningsinlärningsagenten och dess faltningsnätverk på Stable Ba-

selines3. Visualisering av agentens handlingar skapades med hjälp av biblioteket Pygame och grafiska beskrivningar av träningsprocessen skapades med TensorBoard. Slutligen gjordes en jämförelse med kvantbitsdirigeringsprogrammet TKET [11]. Det skapade kodförrådet finns i GitHub [12].

3.3 Funktionsbeskrivning

Beskrivningen av programmet som skapats som en del av detta kandidatarbete har delats upp utefter programmets olika delar som återfinns i projektets kodförråd. En del beskriver miljön, se avsnitt 3.3.1, som definierar hur observationen av sekventiell kvantkod delas upp i tidssteg, vad som räknas som ett exekverbart tillstånd, samt hur och när agenten får utföra en handling. Avsnitt 3.3.2 beskriver beslutspolicyn som agenten använder för att välja handlingar. Sedan beskrivs hur programmet tränar och sparar agenter i avsnitt 3.3.3. Slutligen innehåller avsnitt 3.3.4 information för hur en tränad agent kan testas och dess handlingar visualiseras.

3.3.1 Miljö för möjliga handlingar

Om en användare väljer att ange en serie kvantgrindar som ska utföras, läses denna in av miljön. Om inte så genereras en slumpmässig serie observationsmatriser med en storlek angiven av användaren. Matriserna förbehandlas för att minimera antalet tidssteg, så att kvantgrindar som kan utföras parallellt med varandra kombineras till en och samma observationsmatris. Sedan skapas en observation av de tio första observationsmatriserna, vilket sedan ges som inmatning till förstärkningsinlärningsagenten.

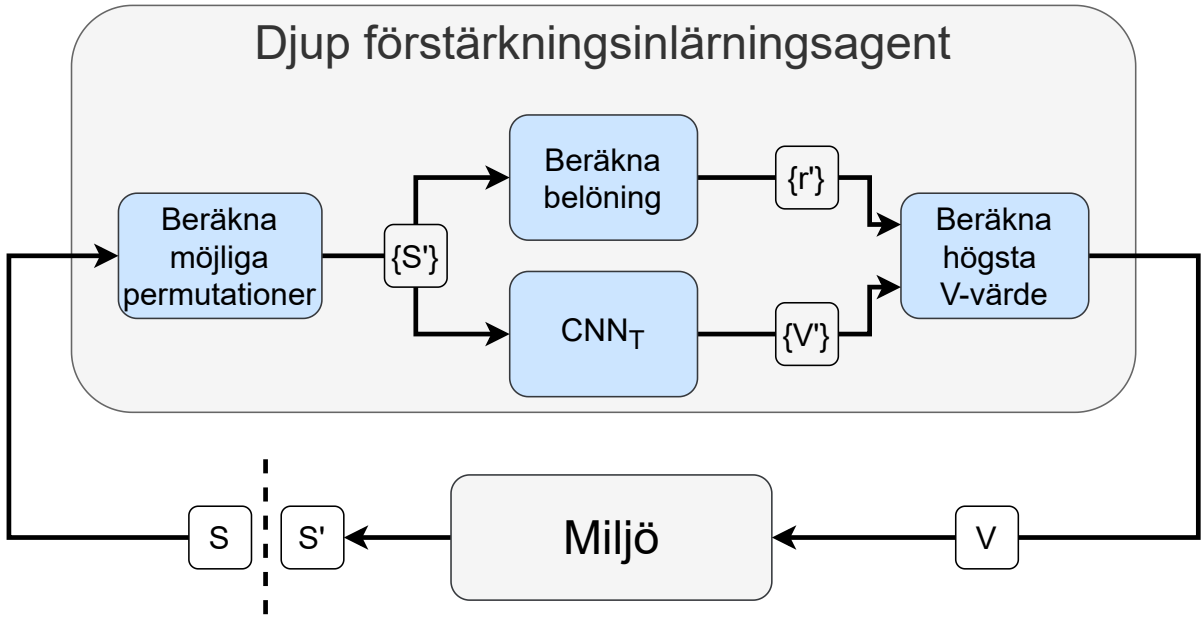
Från agenten läser miljön in ett index som motsvarar den handling agenten har valt att utföra. Den motsvarande permutationsmatrisen identifieras och handlingen utförs. Om handlingen resulterade i ett icke exekverbart tillstånd ges belöningen -2, om tillståndet blev exekverbart ges belöningen -1. Att inte placera ut en SWAP-grind, det vill säga om permutationsmatrisen är identitetsmatrisen, ges belöningen 0. Att utföra en SWAP-grind i samma tidssteg som en annan kvantgrind exekveras ges också belöningen 0.

3.3.2 Policy för beslut version

En schematisk överblick av förstärkningsinlärningsagentens struktur ses i figur 8. Agenten läser in en observation för kvantkretsens nuvarande tillstånd och de framtida grindarna som ska exekveras. Utifrån den första observationsmatrisen genereras en lista av dess möjliga permutationer, som då är de möjliga tillstånden kvantkretsen kan befinna sig i under nästa tidssteg. För varje möjlig permutation i listan beräknas en belöning med hjälp av miljön, samtidigt som observationen ges som inmatning till faltningsnätverket. Nätverket tilldelar sedan varje permutation ett V-värde, och agenten väljer då indexet för det V-värde som tillsammans med sin respektive belöning gav det högsta värdet enligt

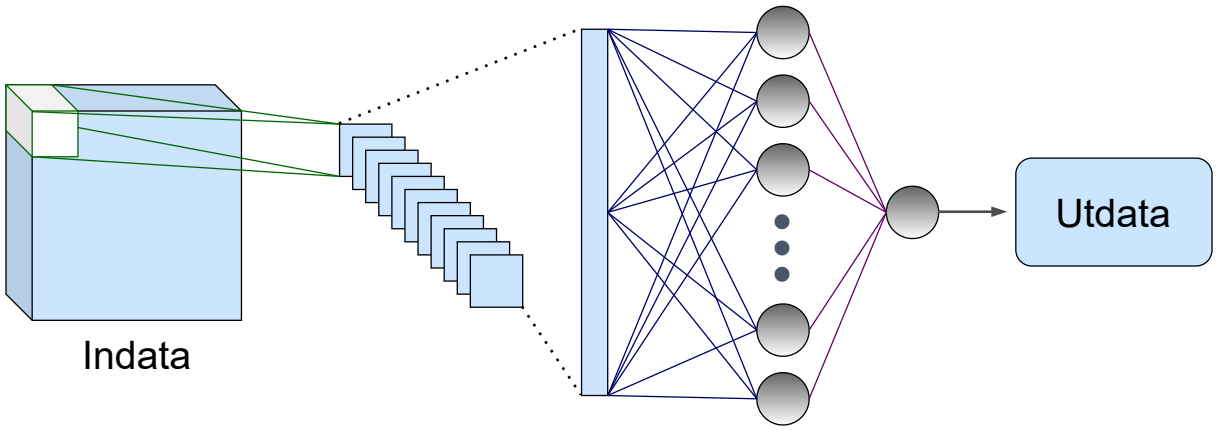
$$V(s) = \text{argmax}(r(\{s\}, \{a\}) + \gamma V(\{s'(a)\})) \tag{11}$$

. Indexet är då den utdata som agenten ger till miljön där permutationen utförs.



Figur 8: En schematisk bild av programmets djupa förstärkningsinlärningsmodell. En agent läser in ett tillstånd och beräknar dess möjliga permutationer. Från listan av möjliga permutationer beräknas respektive permutations belöning med hjälp av en funktion från miljön och ett V-värde beräknas med ett faltningsnätverk. Från dessa värden kan tillståndet med högst V-värde identifieras.

En representation av faltningsnätverket som används ses i figur 9. Nätverket läser in en observation av längd l från miljön. Observationen faltas sedan med ett filter av storlek $2 \times 2 \times l$ för att bilda ett antal neuronmatriser. Storleken valdes för att kombinationer av värden i neuronmatriserna ska korrelera till hur nära placeringen av elementen för ett givet heltalspar i observationsmatrisen befinner sig. I ett tillplattningsskikt skapas en kolonnvektor av neuronmatriserna som i sin tur ansluts till ett fullt anslutet neuronskikt med ett hundratal neuroner. Detta skikt ansluts i sin tur till ett fullt anslutet neuronskikt med en neuron vars värde ges som utmatningsvärde.



Figur 9: Representation av det faltningsnätverk som används i den djupa förstärkningsinlärningsmodellen. Observationen av ett möjligt tillstånd anges som indata. Denna faltas med ett filter av storleken $2 \times 2 \times l$, där l är längden av observationen. De resulterande neuronmatriserna görs om till en kolonnvektor som kopplas till ett fullt anslutet neuronskikt, som i sin tur kopplas till en enskild neuron vars värde ger utdatan.

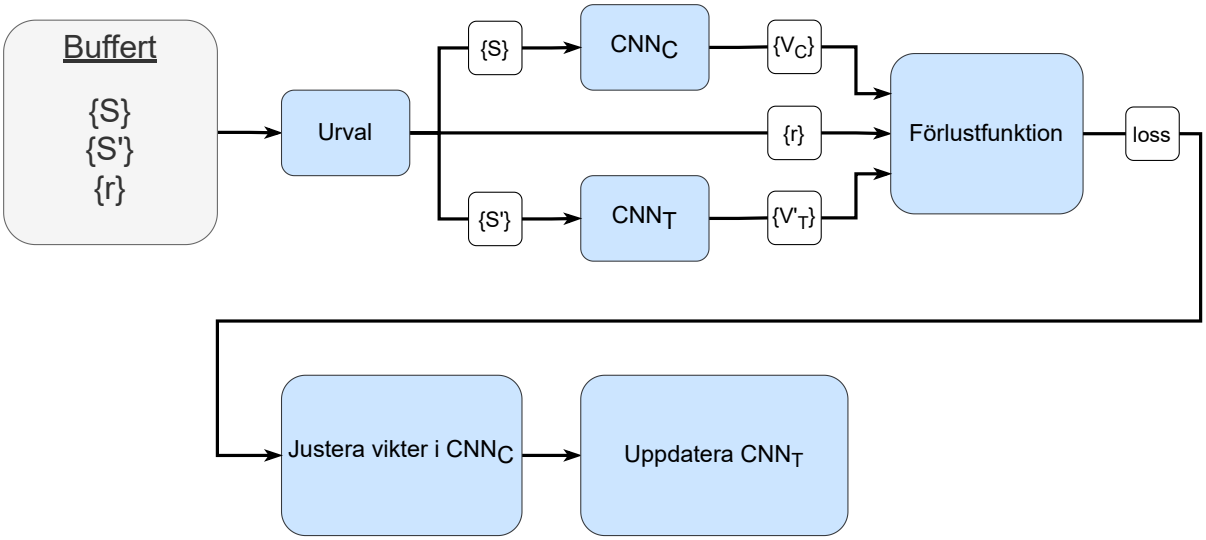
3.3.3 Träning av agent

När en ny förstärkningsinlärningsagent skapas tilldelas dess målnätverk, CNN_T, och det kontinuerligt uppdaterade nätverket, CNN_C, slumpmässiga vikter. Träningen för att justera dessa vikter bestäms av ett antal inlärningsparametrar. Bland annat anger dessa under hur många tidssteg träningen ska fortgå och sannolikheten för att agenten utför en slumpmässig handling. Det vill säga handlingar som görs för att utforska och upptäcka nya kombinationer av gynnsamma och icke-gynnsamma handlingar.

Träningen inleds med en uppvärmningsperiod där inga av de båda faltningsnätverkens vikter uppdateras, vilket resulterar i att slumpmässiga handlingar utförs. Därefter inleds justeringen av vikterna som beskrivs schematiskt i figur 10. Från de slumpmässiga handlingarna skapas en buffert som innehåller tre listor. Den första listan är en serie observationsmatriser, $\{S\}$, som agenten läst in. Den andra listan,$\{S'\}$, innehåller observationsmatrisen för det tillstånd faltningsnätverket bedömde ha högst Q-värde givet motsvarande observationsmatris i den första listan. Den tredje listan, $\{r\}$, innehåller belöningen för handlingen som tog observationsmatrisen i den första listan till observationsmatrisen i den andra. Ur dessa listor görs ett urval där den första listan ges som indata till det kontinuerligt uppdaterade faltningsnätverket, den andra ges som inmatning till målnätverket. Faltningsnätverken beräknar sedan V-värden elementvis för respektive lista. Utifrån dessa beräknas förlustfunktionen, en Mean-Squared-Error-funktion, enligt

$$\mathrm{MSE}(S, S', r) = \frac{1}{n} \sum_{i=1}^{n} [V_C(S_i) - (r_i + \gamma V_T(S'_i))]^2, \tag{12}$$

där n är antalet element i listan. Värdet från förlustfunktionen används sedan för att bakåtpropagera och justera vikterna i det kontinuerligt uppdaterade nätverket, CNN_C, med en stokastisk gradientnedgångsfunktion. De nya vikterna i det kontinuerligt uppdaterade nätverket används sedan för att med jämna intervall uppdatera målnätverkets vikter med en faktor τ.



Figur 10: En schematisk skiss för träningen av en djup förstärkningsinlärningsagent. Från en buffert med erfarenheter hämtas ett urval. Det kontinuerligt uppdaterade nätverket beräknar V-värden av listan $\{S\}$ och målnätverket beräknar V-värden för listan $\{S'\}$. Dessa används för att beräkna förlustfunktionen enligt ekvation 12 som används för att ge ett mått på hur mycket vikterna i det kontinuerligt uppdaterade nätverket uppdateras. Slutligen uppdateras målnätverkets vikter utifrån vikterna i det kontinuerligt uppdaterade nätverket med jämna intervall.

3.3.4 Testning av agent

När en agent har tränats och sparats kan den användas för att göra en observation av sekventiell kvantkod exekverbar med hjälp av ett program för att testa en agent. Detta program läser in agentens nätverk och buffert tillsammans med observationen. Testningen är i själva verket mycket lik träningsprocessen beskriven i avsnitt 3.3.3, med skillnaden att agentens nätverk och buffert inte uppdateras samt att den alltid väljer handlingen med högst V-värde. Testningen har dessutom ett inbyggt visualiseringsprogram som kan användas för att undersöka exakt vilka handlingar agenten utför på en observation vilket beskrivs mer i avsnitt 3.4.

3.4 Visualisering av träning och handlingar

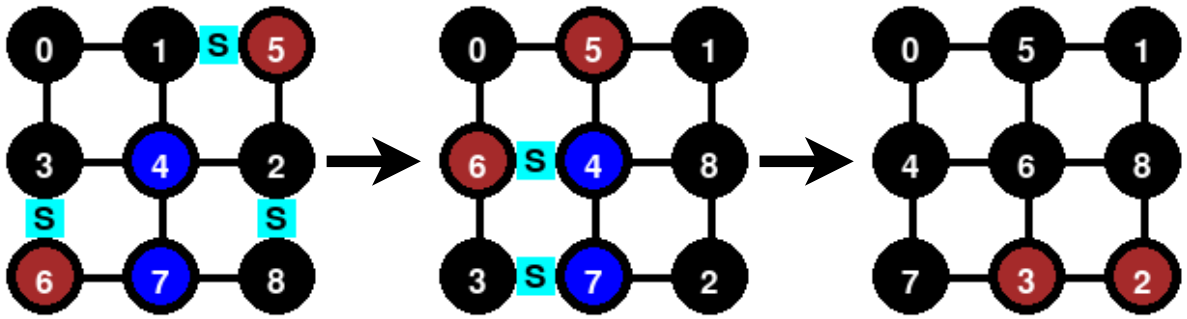
Det är viktigt för en användare att kunna följa hur en agent presterar i ett tränings- respektive testningsförlopp. Därför sparas under dessa förlopp ett antal parametrar i en fil som kan öppnas med programmet TensorBoard. Bland annat sparas antalet tidssteg som agenten har exekverat, förlustfunktionen och den genomsnittliga belöningen per episod, vilket är genomsnittet av belöningarna för varje observationsmatris i observationen.

Det är också viktigt för en användare att kunna kontrollera att de handlingar som en agent utför faktiskt leder till att en observation sätts i ett exekverbart tillstånd. Därför har ett visualiseringsprogram skapats. Med hjälp av detta kan en användare se en observation med tillförda SWAP-grindar och se tidssteg för tidssteg vilka handlingar agenten valde att utföra. Noder har ritats ut som cirklar, och kvantbitarna numreras med heltal från 0 och uppåt. SWAP-grindar mellan kvantbitar har markerats med ett S. Tre exempel på

visualiseringen av tre tidssteg ses i figur 11. Agenten har här läst in observationsmatrisen

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 2 & 0 \end{bmatrix}, \tag{13}$$

som beskriver den observationsmatris som associeras med tidssteget längst till vänster i figur 11 där två kvantgrindar ska exekveras. En mellan kvantbitarna q_4 och q_7 som tilldelats blå färg, och en mellan de bruna kvantbitarna q_5 och q_6. Svart färg beskriver kvantbitar som ingen kvantgrind ska exekveras på, vilka indexeras med värdet 0 i observationsmatrisen. Eftersom att de bruna kvantbitarna q_5 och q_6 inte befinner sig på noder som är grannar kan kvantgrindarna inte exekveras i detta tidssteg. Agenten har därför placerat SWAP-grindar, markerade med ett S på turkos bakgrund, för att byta plats på kvantbitarna q_3 och q_6 samt kvantbitarna q_1 och q_5. Efter ytterligare ett tidssteg där agenten placerat ut SWAP-grindar görs kretsen exekverbar och en ny observationsmatris läses in, vilket markeras med att kvantbitarna tilldelas nya färger.



Figur 11: En bild från visualiseringsprogrammet som visar hur en sekvens av kvantkod för en kvantkrets av storlek 3×3 görs exekverbar. Agenten har läst in observationsmatrisen som ses i ekvation 13 där kvantgrindar ska exekveras mellan kvantbitarna q_5 och q_6 samt q_4 och q_7, och valt att placera ut tre SWAP-grindar. I nästa tidssteg placerar agenten ut två SWAP-grindar och kvantkretsen blir då exekverbar. Därmed har agenten i det tredje tidssteget läst in en ny observationsmatris som visar att en kvantgrind ska utföras mellan kvantbitarna q_5 och q_6. Notera hur agentens placering av SWAP-grindar gör att tillståndet i det tredje tidssteget är direkt exekverbart.

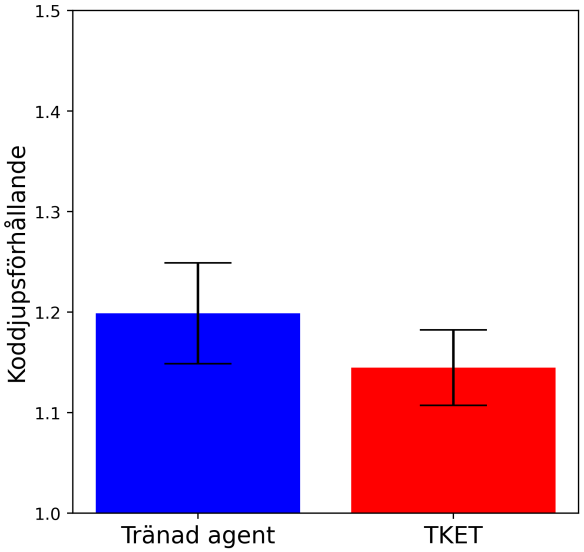
4 Resultat

Nedan redovisas resultat för tränade agenter på kvantkretsar av storlek 2×2 respektive 3×3. I avsnitt 4.1 presenteras och jämförs resultaten med kvantdirigeringsprogrammet TKET. Agenternas träningsprocess presenteras sedan i avsnitt 4.2.

4.1 Resultat för tränade agenter och jämförelse med TKET

Förstärkningsinlärningsagenten som tränades på en kvantkrets av storlek 2×2, testades med en sekventiell kvantkod bestående av 4 000 episoder uppdelat på fyra kodlängder

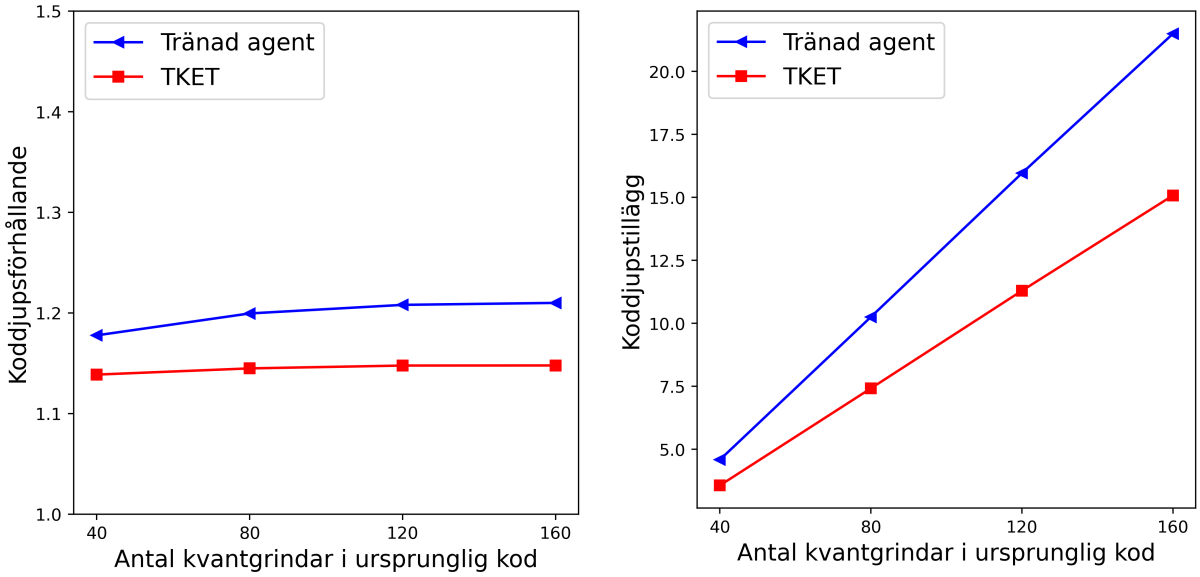
om 1 000 episoder med 40, 80, 120 respektive 160 kvantgrindar. Agenten gjorde koden exekverbar och kodsjupsförhållandet uppmättes vara $1,20$ med en standardavvikelse på $0,05$. Detta kan jämföras med TKET som testades på samma kvantkod och då uppmättes prestera något bättre, dess koddjupsförhållande uppmättes vara $1,15$ med en standardavvikelse på $0,04$. En grafisk jämförelse av dessa resultat ses i figur 12. Samtliga mätvärden för kvantkretsen med storlek 2×2 ses även i tabell 1.



Figur 12: Jämförelse av medelvärde för koddjupsförhållande hos resulterande kod från agenten som tränats på kvantkretsar av storleken 2×2 med kvantdirigeringsprogrammet TKET. Den ursprungliga koden bestod av $4 \cdot 10^5$ kvantgrindar. Det uppmätta koddjupsförhållandet för den tränade agenten var $1,20 \pm 0,05$ och för TKET $1,15 \pm 0,04$.

Resultaten för koddjupsförhållandet delades upp i de olika kodlängderna, vilket ses i figur 13a. Skillnaden i koddjupsförhållande mellan den tränade förstärkningsinlärningsagenten och TKET var ungefär $0,04$ på koden med 40 kvantgrindar. När antalet kvantgrindar ökades till 80, steg också skillnaden i koddjupsförhållande till ungefär $0,06$. Däremot så ökade differensen inte lika mycket när antalet kvantgrindar ökades till 120 och 160, vars skillnader båda avrundas till $0,06$ respektive $0,06$.

För att tydliggöra skillnaden mellan de två kompilatorerna har deras resulterande kod kretsdjupstillägg beräknats, vilka ses i figur 13b. För att göra kvantkod med 40 kvantgrindar exekverbar tillförde förstärkningsinlärningsagenten i snitt $4,6$ tidssteg, medan TKET tillförde i snitt $3,6$ tidssteg. Skillnaden ökade relativt linjärt med antalet kvantgrindar i den ursprungliga kvantkoden. För kvantkod med 160 kvantgrindar lade förstärkningsinlärningsagenten i snitt till $6,4$ tidssteg mer än TKET.



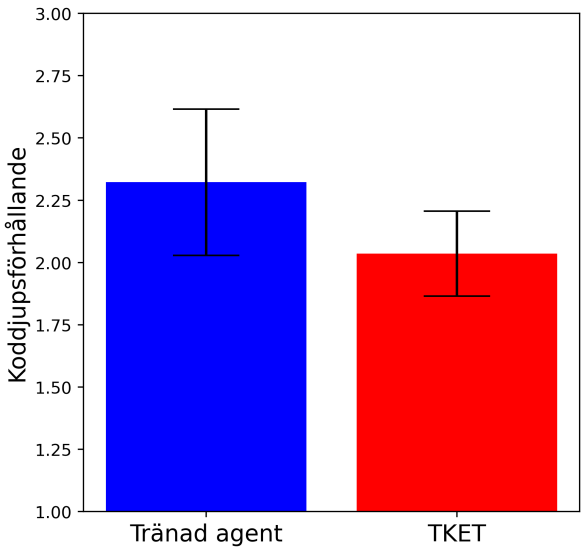
(a) (b)

Figur 13: Uppmätt koddjupsförhållande och koddjupstillägg för agenten som tränades på kvantkretsar av storlek 2×2 som innehöll totalt $4 \cdot 10^5$ kvantgrindar uppdelat på fyra olika kodlängder. Varje värde i bilden är ett medelvärde för 1 000 mätvärden. Samtliga mätvärden och standardavvikelser ses i tabell 1.

Tabell 1: Uppmätt koddjup av ursprunglig respektive resulterande kod samt koddjupstillägg och koddjupsförhållande för agenten som tränades på kvantkretsar av storlek 2×2 som innehöll totalt $4 \cdot 10^5$ kvantgrindar uppdelat på fyra olika kodlängder. För varje kodlängd togs ett medelvärde för 1 000 mätvärden. Den tränade förstärkningsinlärningsagenten förkortas här TA för Tränad Agent.

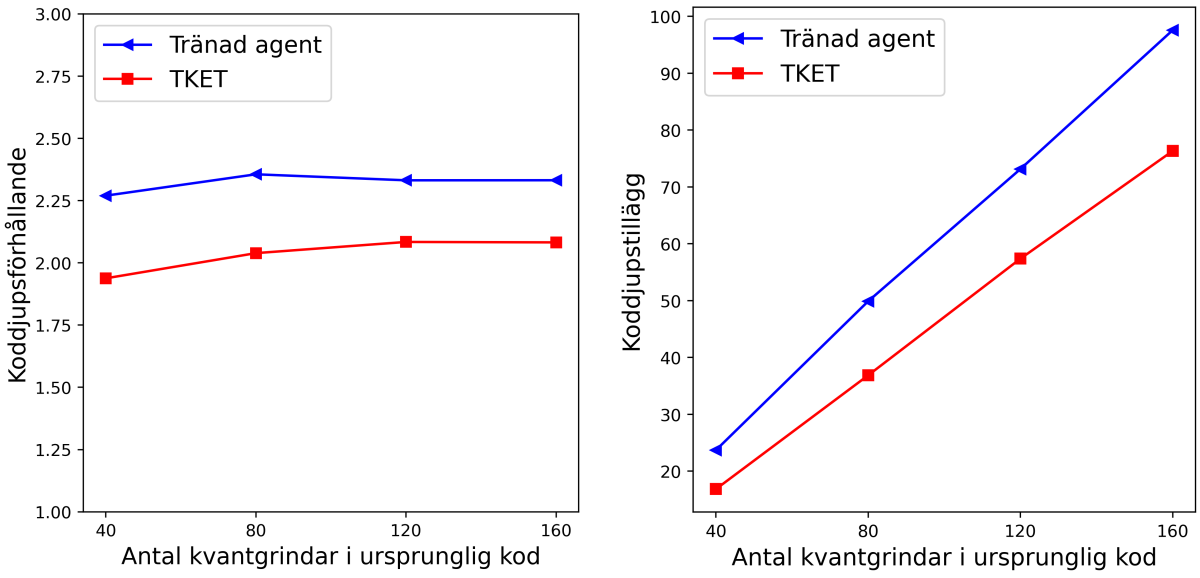
Antal kvantgrindar	Djup av ursprunglingkod		Djup av resulterandekod		Koddjupstillägg		Koddjupsförhållande			
	TA	TKET	TA	TKET	TA	TKET	TA	TKET	SDA TA	SDA TKET
40	25,75	25,65	30,32	29,21	4,58	3,56	1,18	1,14	0,07	0,05
80	51,37	51,18	61,62	58,59	10,25	7,41	1,20	1,14	0,05	0,04
120	76,73	76,46	92,69	87,75	15,96	11,29	1,21	1,15	0,04	0,03
160	102,41	102,03	123,91	117,11	21,50	15,08	1,21	1,15	0,04	0,03
Totalt medelvärde							1,20	1,14	0,05	0,04

För att testa förstärkningsinlärningsagenten på en kvantkrets av storlek 3×3 användes en liknande sekventiell kvantkod som för kvantkretsen av storlek 2×2. Fyra kodlängder med 40, 80, 120 respektive 160 kvantgrindar genererades med 1 000 episoder för varje kodlängd. Både förstärkningsinlärningsagenten och TKET gjorde kvantkoden exekverbar. Kvantkoden för den förstnämnda uppmättes ha ett genomsnittligt koddjupsförhållande på $2,32 \pm 0,24$. Motsvarande värde för TKET var $2,04 \pm 0,17$, vilket är något bättre. Figur 14 visar en grafisk jämförelse av dessa värden och samtliga mätvärden ses i tabell 2.



Figur 14: Jämförelse av medelvärde för koddjupsförhållande hos resulterande kod från agenten som tränats på kvantkretsar av storleken 3×3 med kvantdirigeringsprogrammet TKET. Den ursprungliga koden bestod av $4 \cdot 10^5$ kvantgrindar. Det uppmätta koddjupsförhållandet för den tränade agenten var $2,32 \pm 0,29$ och för TKET $2,04 \pm 0,17$.

Kvantkodjupsförhållandet för förstärkningsinlärningsagenten och TKET för de olika koddjupen jämförs i figur 15a. För koden med 40 kvantgrindar uppmättes agenten och TKET ha ett koddjupsförhållande på $2,27$ respektive $1,94$, en skillnad på $0,33$. Denna skillnad minskar sedan något, till $0,32$ för koden med 80 kvantgrindar och till $0,25$ för både 120 och 160 kvantgrindar. Kretsdjupstillägget som visar antalet tillförda tidssteg för de två agenterna visas i figur 15b. På kvantkoden med 40 grindar tillförde förstärkningsinlärningsagenten i genomsnitt $23,7$ tidssteg, $6,8$ fler än TKET som tillför $16,9$. Skillnaden ökade till $13,0$ på kvantkod med 80 kvantgrindar och till $15,7$ respektive $21,3$ för 120 och 160 kvantgrindar.



(a) (b)

Figur 15: Uppmätt koddjupsförhållande och koddjupstillägg för agenten som tränades på kvantkretsar av storlek 3×3 som innehöll totalt $4 \cdot 10^5$ kvantgrindar uppdelat på fyra olika kodlängder. Varje värde i bilden är ett medelvärde för 1 000 mätvärden. Samtliga mätvärden och standardavvikelser ses i tabell 2.

Tabell 2: Uppmätt koddjup av ursprunglig respektive resulterande kod samt koddjupstillägg och koddjupsförhållande för agenten som tränades på kvantkretsar av storlek 3×3 som innehöll totalt $4 \cdot 10^5$ kvantgrindar uppdelat på fyra olika kodlängder. För varje kodlängd togs ett medelvärde för 1 000 mätvärden. Den tränade förstärkningsinlärningsagenten förkortas här TA för Tränad Agent.

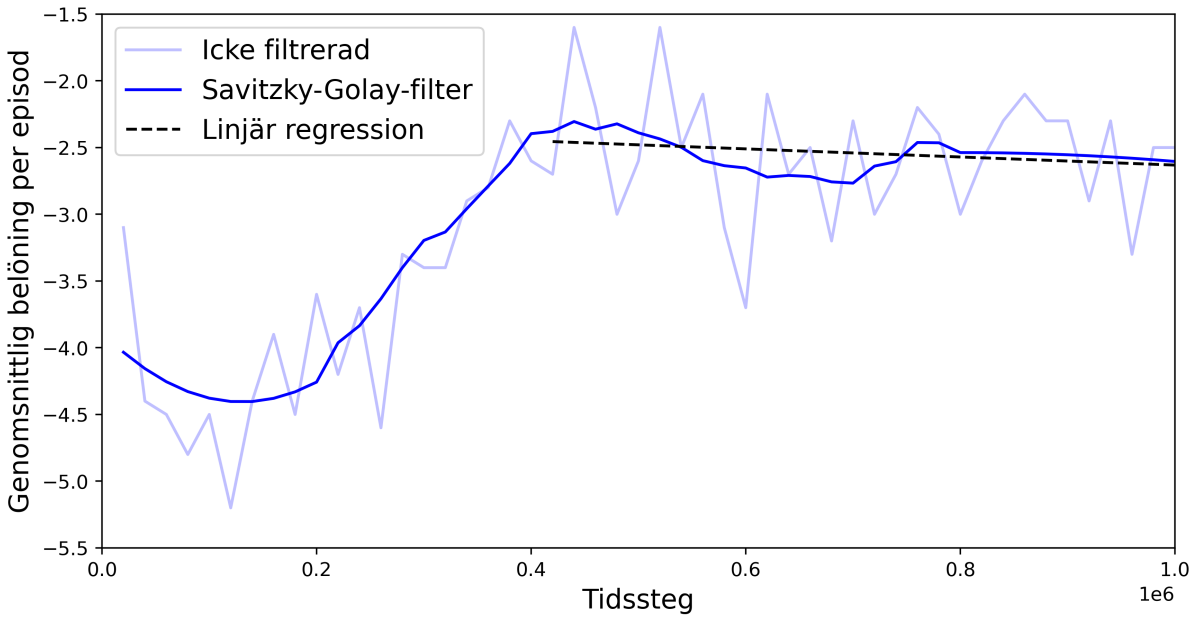
Antal kvantgrindar	Djup av ursprungling kod		Djup av resulterande kod		Koddjupstillägg		Koddjupsförhållande			
	TA	TKET	TA	TKET	TA	TKET	TA	TKET	SDA TA	SDA TKET
40	18,67	17,96	42,37	34,81	23,70	16,85	2,27	1,94	0,39	0,24
80	36,80	35,48	86,69	72,34	49,89	36,86	2,36	2,04	0,43	0,17
120	54,93	52,93	128,07	110,30	73,14	57,37	2,33	2,08	0,21	0,15
160	73,25	70,52	170,81	146,83	97,56	76,32	2,33	2,08	0,15	0,13
Totalt medelvärde							2,32	2,04	0,29	0,17

4.2 Träning av förstärkningsinlärningsagenter

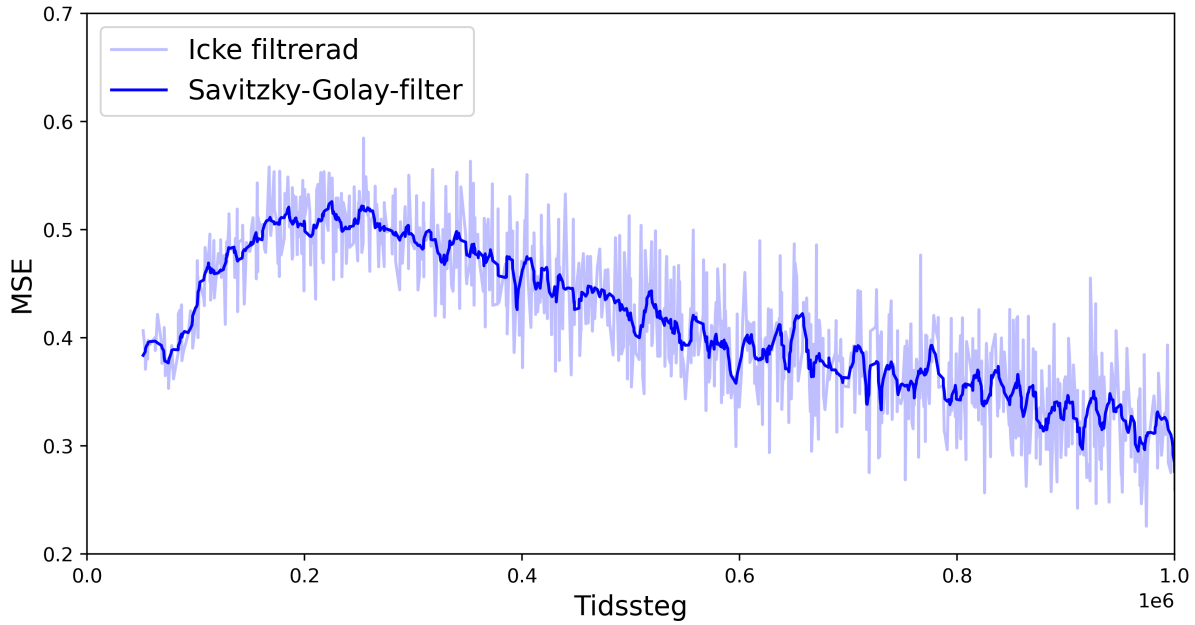
Förstärkningsinlärningsagenten för kvantkretsar av storleken 2×2 tränades på 10^6 tidssteg. Tidsstegen delades upp i episoder där varje episod representerade en serie av 40 kvantgrindar. Hur den genomsnittliga belöningen per episod förändrades under tidsstegen ses i figur 16. Efter uppvärmningsperioden på de första $0,05 \cdot 10^6$ tidsstegen är den genomsnittliga belöningen per episod ungefär -4. Efter att först sjunka något börjar den vid cirka $0,2 \cdot 10^6$ tidssteg öka mot ungefär $-2,3$. Slutligen ses den sjunka något och närma sig $-2,5$.

Förlustfunktionen för träningen av agenten beräknades med en Mean-Squared-Error-funktion, se ekvation 12, mellan det kontinuerligt uppdaterade nätverket, CNN_C, och målnätverket, CNN_T, och denna ses i figur 17. Efter uppvärmningsperioden har nätverken ett MSE på ungefär $0,4$ för att sedan öka till något över $0,5$ vid $0,2 \cdot 10^6$ tidssteg. Detta indikerar att vikterna i de två faltningsnätverken har börjat förändras och att nätverken blir mer olika varandra. Efter detta börjar förlustfunktionen att sjunka för att i slutet av träningen närma sig ett MSE något under $0,3$. Nätverken blir alltså mer lika varandra, vilket antyder att vikterna inte längre justeras slumpmässigt. Vid en jämförelse med grafen för genomsnittlig belöning per episod, figur 16, ses att den genomsnittliga belöningen börjar öka vid ungefär samma tidssteg som förlustfunktionen börjar sjunka, båda runt $0,2 \cdot 10^6$ tidssteg. Vikterna justeras därmed inte bara slumpmässigt, utan också på ett sådant sätt att den genomsnittliga belöningen ökar.

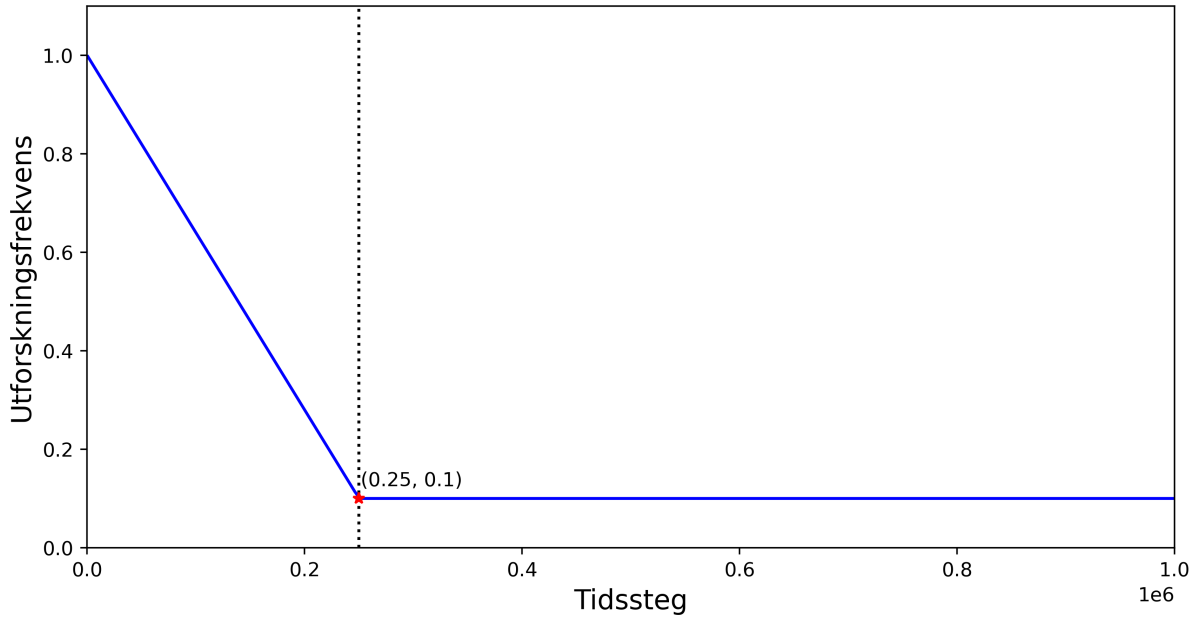
I figur 18 visas utforskningsfrekvensen, en inlärningsparameter, för agenten som tränades på kvantkrets av storlek 2×2. Utforskningsfrekvensen börjar på 1 och minskar sedan linjärt till $0,1$ under de första $0,25 \cdot 10^6$ för att sedan vara konstant.



Figur 16: Genomsnittlig belöningsgraf för en agent under träning på kvantkod av storlek 2×2 under 10^6 tidssteg där varje episod utgörs av 40 kvantgrindar. Det sista mätvärdet är $-2,5$. Den linjära regressionens lutning är $-3,04 \cdot 10^{-7}$.



Figur 17: Förlustfunktion för en agent under träning på kvantkod av storlek 2×2 under 10^6 tidssteg där varje episod utgörs av 40 kvantgrindar. Det sista mätvärdet är $0,26$.

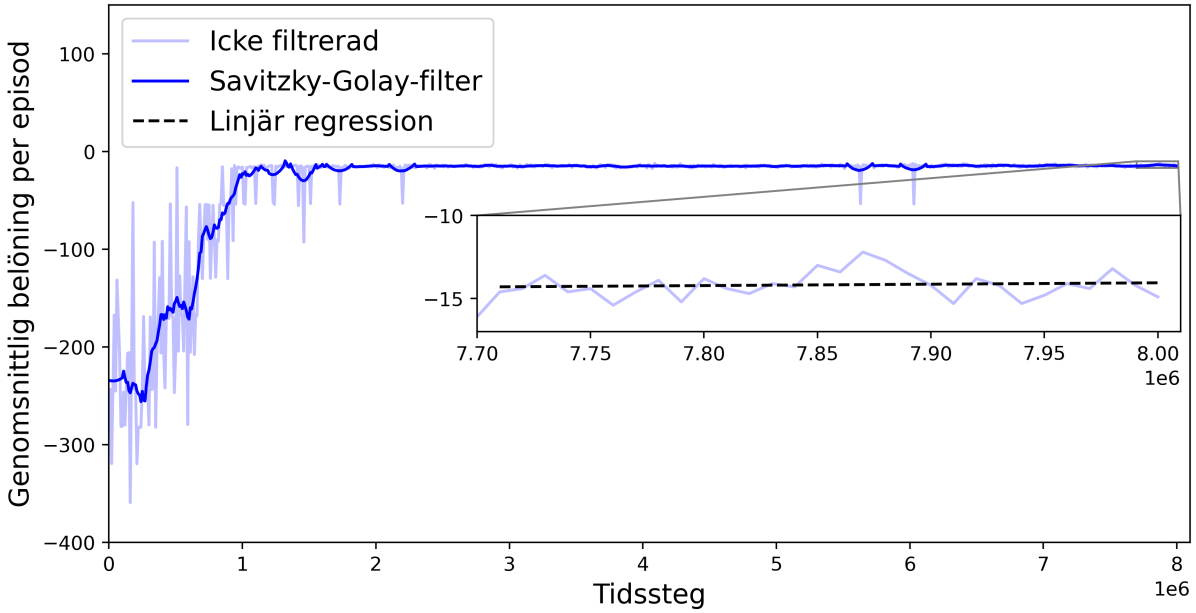


Figur 18: Utforskningsfrekvens för en agent under träning på kvantkod av storlek 2×2 under 10^6 tidssteg. Utforskningsfrekvensen är en inlärningsparameter som valts så att den går mot $0,1$ och når detta värde vid tidssteget $0,25 \cdot 10^6$.

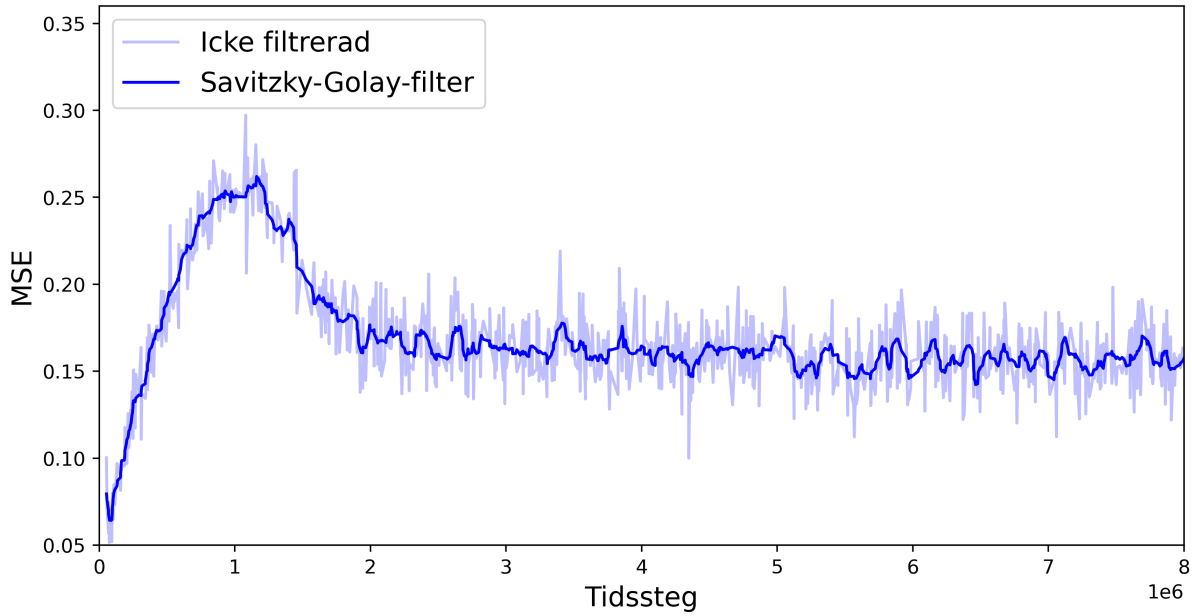
Förstärkningsinlärningsagenten som tränades på kvantkretsar av storleken 3×3, tränades under $8 \cdot 10^6$ tidssteg med episoder om 40 kvantgrindar vardera. Dess genomsnittliga belöning per episod ses i figur 19. Likt den motsvarande grafen för agenten som tränades på kvantkretsar med storleken 2×2, jämför med figur 16, börjar den genomsnittliga

belöningen efter uppvärmningsperioden med att sjunka något. Här från cirka -230 till -250. Sedan börjar den att öka och går mot en genomsnittlig belöning på ungefär -20 runt ungefär $1 \cdot 10^6$ tidssteg. Mot slutet av träningen, mellan 7 och $8 \cdot 10^6$ tidssteg har den genomsnittliga belöningen närmat sig ett värde något över -15.

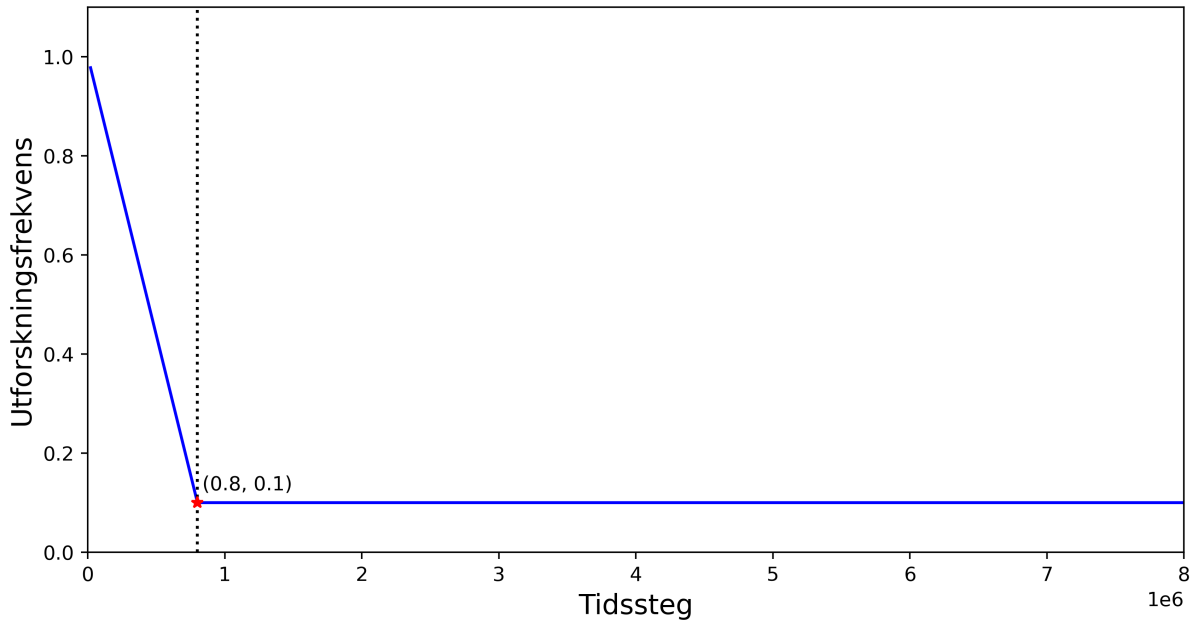
Den stora förbättringen av nätverkets genomsnittliga belöning från början av träningen till $1 \cdot 10^6$ kan jämföras med grafen för förlustfunktionen under träningen, som ses i figur 20. Under denna del av träningen går förlustfunktionen från ett MSE på $0,07$ till $0,25$, vilket betyder att det kontinuerligt uppdaterade nätverket och målnätverket blir mer olika varandra. Den ökande belöningen antyder i belöningsgrafen att nätverken blir bättre på att värdera tillstånd trots att nätverken enligt förlustfunktionsgrafen blir mer olika varandra. När den genomsnittliga belöningen stabiliserat sig strax efter $1 \cdot 10^6$ tidssteg, ses i figuren för förlustfunktionen hur MSE börjar sjunka och nätverken blir mer lika varandra. Utforskningsfrekvensen för agenten presenteras som referens och visas i figur 21. Vid tidssteget $0,8 \cdot 10^6$ antar frekvensen värdet 0,1 och förblir sedan konstant.



Figur 19: Genomsnittlig belöningsgraf för en agent under träning på kvantkod av storlek 3×3 under $8 \cdot 10^6$ tidssteg där varje episod utgörs av 40 kvantgrindar. Det sista mätvärdet är $-14,9$ och den linjära regressionens lutning är $8,32 \cdot 10^{-7}$.



Figur 20: Förlustfunktion för en agent under träning på kvantkod av storlek 3×3 under 10^6 tidssteg där varje episod utgörs av 40 kvantgrindar. Det sista mätvärdet är $0,15$.

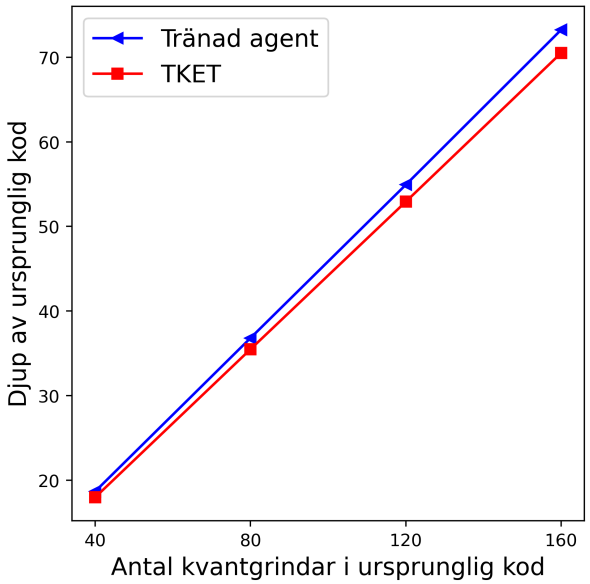


Figur 21: Utforskningsfrekvens för en agent under träning på kvantkod av storlek 3×3 under $8 \cdot 10^6$ tidssteg. Utforskningsfrekvensen är en inlärningsparameter som valts så att den går mot $0,1$ och når detta värde vid tidssteget $0,8 \cdot 10^6$.

5 Diskussion

Resultaten för de djupa förstärkningsinlärningsagenterna kan anses visa att det fungerar relativt väl att låta ett faltningsnätverk beräkna V-värden för tillstånd. Agenternas resulterande kod har dessutom koddjupsförhållanden respektive koddjupstillägg som är något sämre, men i samma storleksordning som motsvarande värden för TKET. Dessutom så antas inlärningsparametrarna för träningen av förstärkningsinlärningsagenter inte vara optimerade, och eventuellt skulle en justering av dessa förbättra resultaten för framtida tränade agenter.

Jämförelsen mellan de båda förstärkningsinlärningsagenterna och kvantbitdirigeringsprogrammet TKET, som presenteras i avsnitt 4.1, antyder att de två agenterna är något sämre på att dirigera kvantbitar jämfört med TKET. Denna jämförelsen tar dock inte hänsyn till vissa skillnader mellan hur de tränade agenterna och TKET fungerar. Till exempel använder TKET sig, förutom SWAP-grindar, också av BRIDGE-grindar, samt en annorlunda funktion för att komprimera kvantkodsdjup. Denna effekt illustreras av figur 22 som visar att antalet tidssteg som TKET komprimerar den ursprungliga koden till, genomgående är mindre än den tränade agenten. Skillnaden antyds dessutom öka när antalet kvantgrindar i den ursprungliga koden ökas. Det är troligt att detta påverkar jämförelsen mellan de tränade agenterna och TKET i den senares fördel. En annan möjlig faktor som gör att resultaten för TKET skiljer sig från resultaten för tränade agenterna, är att de har tränats med annorlunda träningsdata. Det är således osäkert om skillnaden mellan de två kompilatorernas resultat beror på att de komprimerar ursprunglig och resulterande kod annorlunda, om det beror på TKET:s användande av BRIDGE-grindar, eller om skillnaden är ett resultat av att träningsdatan är annorlunda.



Figur 22: Koddjup av ursprunglig kod för kvantkod på kvantkretsar av storleken 3×3 uppdelat över fyra kodlängder. Bilden illustrerar skillnaden mellan antalet tidssteg som den tränade agenten och TKET komprimerar den ursprungliga koden i. Varje mätvärde är ett medelvärde över 1 000 episoder.

I avsnitt 4.2 presenteras resultatet för träningen av agenterna som användes.

Förlustfunktionen för träningen av agenten på kvantkretsar av storleken 2×2 ses i slutet av träningen, mellan $0,8 \cdot 10^6$ och $1 \cdot 10^6$ tidssteg, ha en nedåtgående trend och ännu inte ha konvergerat till ett stabilt värde. Detta antyder att förstärkningsinlärningsagentens genomsnittliga belöning per episod hade kunnat förbättras ytterligare om träningen hade förlängts med fler tidssteg. En liknande trend kan ses för agenten som tränades på kvantkretsarna av storleken 3×3 då den anpassade linjärregressionens lutning inte är noll. I graferna för genomsnittlig belöning, figur 16 och 19, ses också de icke-filtrerade värdena fluktuera runt linjen för de Savitzky-Golay-filtrerade värdena. Detta kan bero på att kvantkoderna som genererades kräver att olika många SWAP-grindar tillförs för att koden ska bli exekverbar. Det kan också betyda att det fortfarande genereras mönster i kvantkoden som faltningsnätverket inte känner igen. Om det senare fallet är sant så indikerar detta att agenterna hade kunnat tränas på ytterligare episoder för att minska denna fluktuation.

Det är även intressant att förlustfunktionen för förstärkningsinlärningsagenten som tränades på kvantkretsar av storleken 3×3, se figur 20, inte ser ut att gå mot noll. Detta skulle kunna bero på att listan av tillstånden $\{S'\}$ i bufferten skiljer sig från listan av tillstånden $\{S\}$. Skillnaden är att det andra elementet i listan $\{S\}$ är det första i $\{S'\}$, och det sista i elementet i $\{S\}$ är det näst sista i $\{S'\}$. I träningsmodellen för agenter så fylls denna buffert kontinuerligt på med nya element allteftersom att det föregående gjorts exekverbart. Detta gör att funktionernas argument i ekvation 12 uppdateras löpande under träningens gång, vilket skulle kunna göra att det är naturligt att förlustfunktionen inte går mot noll.

5.1 Vidare utveckling

En framtida utforskning som skulle vara intressant är att låta en förstärkningsinlärningsagent och TKET tränas med samma träningsdata för att se om skillnaden i deras resultat kvarstår. Detta skulle bidra till att tydliggöra skillnaden mellan deras faltningsnätverk och att de baserar sina valda handlingar på olika ekvationer för Q-värden, se ekvationerna 9 och 10. Vidare skulle även inlärningsparametrarna för träning av förstärkningsinlärningsagenter kunna optimeras. Det skulle till exempel vara intressant att göra detta med optimeringsvara som Zoo, Optuna eller Scikit-Optimize. Under arbetet så gjordes ett försök till att implementera Zoo, förkortat från Rl-baselines3-zoo, vilket är ett ramverk som kan användas för att ändra på variabler och algoritmer. Efter en extensiv felsökning togs beslutet att inte använda ramverket i vårt färdiga program. Anledningen till detta var problem med att göra programvaran kompatibel med vår förstärkningsinlärningsmodell. Förslagsvis skulle ett nytt försök kunna göras där en förstärkningsinlärningsmodell skapas från grunden med kompatibilitet till optimeringsvara i åtanke.

En annan fortsättning av arbetet skulle också kunna vara att utveckla förstärkningsinlärningsmodellen så att den fungerar på större matriser med fler kvantbitar. Huvudproblemet med detta är att antalet möjliga tillstånd som behöver beräknas ökar snabbt med antalet kvantbitar. Detta ökar kraftigt träningstiden för agenter och minskar sannolikheten för att förlustfunktionen ska konvergera. Vilket betyder att faltningsnätverket fluktuerar och att det blir närmast slumpmässigt vilket av de möjliga tillstånden som tilldelas det högsta V-värdet. För att hantera detta skulle en heuristik kunna

implementeras. Införandet av en väl vald tumregel som på förhand effektivt kan sortera bort möjliga tillstånd som oavsett skulle ha tilldelats ett dåligt V-värde, skulle då minska antalet tillstånd faltningsnätverket behöver undersöka. Detta skulle öka sannolikheten för att kunna träna agenter på större kvantkretsar.

6 Slutsats

Kandidatarbetets syfte kan sammanfattas som att utforska om en djup förstärkningsinlärningsmodell där tillstånd som värderas kan användas av en kompilator för att placera ut SWAP-grindar för att en kvantkod ska bli exekverbar. Förstärkningsinlärningsagenterna som tränades applicerades för att göra kvantkod exekverbar på kvantkretsar av storlekarna 2×2 och 3×3. För kvantkoden på kvantkretsar av storleken 2×2 uppmättes ett koddjupsförhållande på $1,2 \pm 0,1$ och för kvantkretsar av storleken 3×3 uppmättes koddjupsförhållandet $2,3 \pm 0,65$. Dessa koddjupsförhållanden var dessutom endast något större än motsvarande uppmätta värden för kvantbitdirigeringsprogrammet TKET, vilka var $1,15 \pm 0,8$ respektive $2,0 \pm 0,4$. Det är dock osäkert om skillnaden i denna jämförelse beror på faltningsnätverkens effektivitet eller på skillnader för hur ursprunglig och resulterande kvantkod komprimeras. Dessutom antas inlärningsparametrarna för förstärkningsinlärningsagenternas träning gå att optimera för att förbättra resultat för framtida tränade agenter. Sammanfattningsvis antyder detta att metodiken för att beräkna V-värden för möjliga tillstånd mycket väl skulle kunna användas för att dirigera kvantbitar.

Referenser

[1] Bengtsson A, Vikstål P, Warren C, Svensson M, Gu X, Kockum AF, et al. Improved Success Probability with Greater Circuit Depth for the Quantum Approximate Optimization Algorithm. Phys Rev Applied. 2020 Sep;14:034010. Available from: https://link.aps.org/doi/10.1103/PhysRevApplied.14.034010.

[2] Sweden's quantum computer project shifts up a gear. Chalmers University of Technology; 2021. Available from: https://news.cision.com/chalmers/r/sweden-s-quantum-computer-project-shifts-up-a-gear,c3305315.

[3] Bobier JF, Langione M, Tao E, Gourévitch A. What happens when 'if' turns to 'when' in quantum computing?. BCG Global;. Available from: https://www.bcg.com/publications/2021/building-quantum-advantage.

[4] Casati G, Benenti G. Quantum Computation and Chaos. In: Bassani F, Liedl GL, Wyder P, editors. Encyclopedia of Condensed Matter Physics. Oxford: Elsevier; 2005. p. 9-17. Available from: https://www.sciencedirect.com/science/article/pii/B0123694019011463.

[5] Pozzi MG, Herbert SJ, Sengupta A, Mullins RD. Using Reinforcement Learning to Perform Qubit Routing in Quantum Compilers; 2020. Available from: https://arxiv.org/abs/2007.15957.

[6] Andreasson P, Johansson J, Liljestrand S, Granath M. Quantum error correction for the toric code using deep reinforcement learning. Quantum. 2019 sep;3:183. Available from: https://doi.org/10.22331%2Fq-2019-09-02-183.

[7] Fitzek D, Eliasson M, Kockum AF, Granath M. Deep Q-learning decoder for depolarizing noise on the toric code. Phys Rev Research. 2020 May;2:023230. Available from: https://link.aps.org/doi/10.1103/PhysRevResearch.2.023230.

[8] Bub J. Quantum Information and Computation. In: Butterfield J, Earman J, editors. Philosophy of Physics. Handbook of the Philosophy of Science. Amsterdam: North-Holland; 2007. p. 555-660. Available from: https://www.sciencedirect.com/science/article/pii/B9780444515605500099.

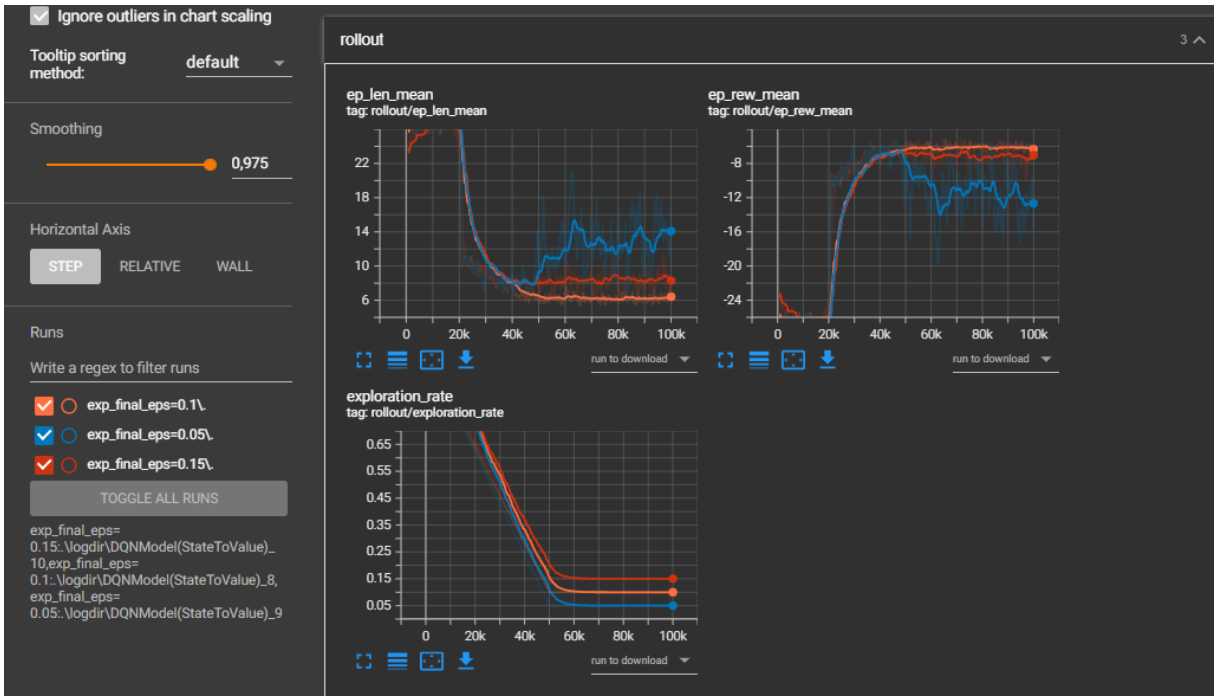
[9] Conv3D;. Accessed: 2022-05-12. https://pytorch.org/docs/stable/generated/torch.nn.Conv3d.html.

[10] O'Shea K, Nash R. An Introduction to Convolutional Neural Networks. CoRR. 2015;abs/1511.08458. Available from: http://arxiv.org/abs/1511.08458.

[11] CQCL. TKET. Cambridge Quantum; Sep 13, 2021 (hämtad Maj 10, 2022). Available from: https://github.com/CQCL/tket.

[12] Einarsson JMS Lindahl. DQNSWAPModel; Maj 12, 2022 (hämtad Maj 12, 2022). Available from: https://github.com/karieinarsson/TIFX04-Kompilering-av-kvantdatorkod.

A Utforskningssfrekvens



Figur 23: Ett tidigt försök av olika utforskningsfrekvenser som låg till grund för beslutet att anta att en utforskningsfrekvens på $0,1$ skulle ge tillräckligt goda resultat för förstärkningsinlärningsmodellen.