Neuronska mreža na kvantnom računalu

Marević, Lucija

Sveučilište u Splitu
Prirodoslovno-matematički fakultet

Neuronska mreža na kvantnom računalu

Završni rad

Lucija Marević

Split, rujan 2023.

Temeljna dokumentacijska kartica

Sveučilište u Splitu
Prirodoslovno–matematički fakultet
Odjel za fiziku
Ruđera Boškovića 33, 21000 Split, Hrvatska

Završni rad

Neuronska mreža na kvantnom računalu

Lucija Marević

Sveučilišni prijediplomski studij Fizika

Sažetak:

Kvantno strojno učenje još je slabo istraženo područje. U ovom radu nastojimo dati uvid u dijelić tog fascinatnog područja, kojeg zovemo kvantne neuronske mreže. Započinjemo rad uvodeći osnovne pojmove i teoriju potrebnu za razumijevanje obrađenih algoritama. Dajemo kratak teorijski opis principa rada klasičnog višeslojnog perceptrona te teorijski uvodimo njegov kvantni analogon, disipativne kvantne neuronske mreže. Uvodimo i model hibridne klasično-kvantne neuronske mreže te njenu implementaciju u Qiskit-u. Istražujemo mogućnosti takve arhitekture i diskutiramo dobivene rezultate.

Ključne riječi: kvantni krug, kodiranje podataka, prosljeđivanje unaprijed, funkcija cijene, prosljeđivanje unatrag, klasifikacija

Rad sadrži: 34 stranice, 31 sliku, 0 tablica, 12 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: prof. dr. sc. Leandra Vranješ Markić

Ocjenjivači: prof. dr. sc. Leandra Vranješ Markić
Josipa Šćurla, mag. phys.
izv. prof. dr. sc. Larisa Zoranić

Rad prihvaćen: 24. rujna 2023.

Rad je pohranjen u Knjižnici Prirodoslovno–matematičkog fakulteta, Sveučilišta u Splitu.

Basic documentation card

University of Split
Faculty of Science
Department of Physics
Ruđera Boškovića 33, 21000 Split, Croatia

Bachelor thesis

Quantum neural networks

Lucija Marević

University undergraduate study Physics

Abstract:

Quantum machine learning is still quite a bit of a mistery. In this paper we will try to introduce a tiny bit of that fascinating area, called quantum neural networks. We begin by introducing basic concepts and theory neccessary for understanding the analyzed algorithms. We will briefly discuss the theoretical description of the priciples of classical multi-layer perceptron and its quantum analogue, dissipative quantum neural network, DKNN. We will also introduce a hybrid quantum-classical neural network and its implementation in Qiskit. We will explore the possibilities they provide us with and discuss the results.

Keywords: quantum circuit, data encoding, feed forward, cost function, back propagation, classification

Thesis consists of: 34 pages, 31 figures, 0 tables, 12 references. Original language: Croatian.

Supervisor: Prof. Dr. Leandra Vranješ Markić

Reviewers: Prof. Dr. Leandra Vranješ Markić
Josipa Šćurla, MSc. Phys.
Asoc. Prof. Dr. Larisa Zoranić

Thesis accepted: September 24th 2023.

Thesis is deposited in the library of the Faculty of Science, University of Split.

Hvala mentorici prof. dr. sc. Leandri Vranješ Markić na strpljenju i pomoći kako bi ovaj rad bio ostvaren. Hvala i Josipi Šćurli, mag. phys. i izv. prof. dr. sc. Larisi Zoranić što su pristale biti dio povjerenstva. Posebna zahvala ide i svim profesorima koji su mene i moje fakultetske kolege vodili kroz ove tri godine na trudu i volji za prenošenjem znanja. Hvala mojoj obitelji na neprestanoj podršci i strpljenju sa mnom. Bez njih ne bih sada branila prvi veliki rad.

Sadržaj

1 Uvod

U ljudskoj prirodi je pitati se što je sve moguće i na koje načine možemo dalje koristiti i unaprijeđivati do sada stečeno znanje. Tijekom stoljeća razvoja čudesnog svijeta fizike u nekom trenuntku poznati fizičari krenuli su u drugom smjeru i otkrili jedan poseban novi "svemir", kojeg danas nazivamo kvantna mehanika.
Ona nam je otvorila niz novih mogućnosti i otvorila vrata za brojne tehnološke napretke. Jedna od ideja koja se rodila iz toga je spajanje računarstva s principima kvantne mehanike. To uopće nije čudno pogledamo li samo činjenicu da se n klasičnih bitova može nalaziti u ukupno $2n$ stanja, dok nam sustav od n kvantnih bitova omogućuje pristup 2^n stanja. Očigledno je da bi nam razvoj kvantnog računarstva omogućio brže i bolje izvršavanje već postojećih i razvoj novih, dosad neizvedivih algoritama. Prve ideje o kvantnom računarstvu pojavile su se 1980.-ih [1] i od tada je postignut značajan razvoj tog područja. Danas kvantna računala mogu izvoditi gotovo sve što i klasična na nekoj osnovnoj razini. Razvoj kvantnog strojnog učenja započeo je sredinom 1990.-ih [1]. Neuronske mreže posebno su područje unutar strojnog učenja. Prvi spomen kvantnih neuronskih mreža bio je 1995. godine, u odvojenim radovima S. Kak-a i R. Chrisley-a [22]. Od njih do danas imamo više skupova alata za implementaciju kvantnog strojnog učenja, a s njim i neuronskih mreža kao što su Qiskit, Cirq, Nisq i drugi. Oni sadrže niz biblioteka (modula) i klasa u kojima se nalaze brojni algoritmi za primjenu kvantnog računarstva.
U ovom radu dati ćemo kratki uvod u kvantno računarstvo i alate potrebne za kvantno strojno učenje. Opisat ćemo princip rada klasičnih višeslojnih perceptrona te njihovih kvantnih ekvivalenata, disipativnih kvantnih neuronskih mreža, DKNM. Zbog složenosti algoritma i nedostatnih resursa za simulaciju, u ovom radu nije prikazana njihova implementacija u kodu. Zato je kratko opisana hibridna kvantno-klasična neuronska mreža i njena implementacija u Qiskit-u. Prikazat ćemo jedan jednostavan primjer klasifikacije u Qiskit-u te raspraviti dobivene rezultate.

2 Osnovne kvantnog računanja

2.1 Kvantni bit (Q-bit)

Najmanja jedinica koja nosi neku informaciju na klasičnom računalu je bit. Svaki bit može se nalaziti u 2 stanja, njegova vrijednost može biti 0 ili 1. Analogno klasičnom, osnovna jedinica informacije u kvantnom računarstvu je kvantni bit. Vektor stanja kvantnog bita je vektor norme 1 u dvodimenzionalnom unitarnom Hilbertovom vektorskom prostoru nad poljem kompleksnih brojeva s ortonormiranom bazom [6]. Elemente baze je uobičajeno označavati $|0\rangle$ i $|1\rangle$ u Diracovoj notaciji. Njihova matrična reprezentacija je:

$$|0\rangle := \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad |1\rangle := \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{2.1}$$

Kvantni bitovi se osim u ta dva stanja mogu nalaziti i u superpoziciji stanja koju prikazujemo kao linearnu kombinaciju vektora baze. Primjerice:

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle). \tag{2.2}$$

Superpozicija stanja je također vektor stanja u istom Hilbertovom prostoru koji predstavlja neko novo stanje. Prema principima kvantne mehanike, mjerenjem sustav "padne" u jedno od stanja baze. Ako se vektor stanja iz gornjeg primjera zapiše kao:

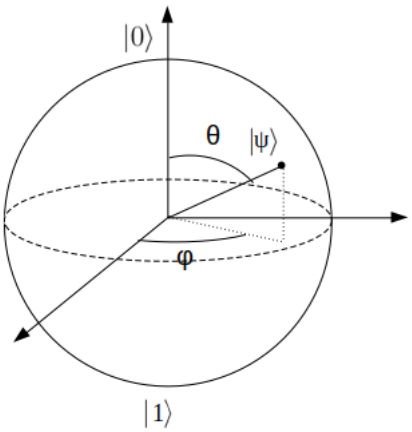
$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{2.3}$$

kada bi netko htio mjeriti njegovo stanje, prema Bornovom pravilu, vjerojatnost da izmjeri $|0\rangle$ iznosila bi $|\alpha|^2$, a $|1\rangle$ $|\beta|^2$. Koeficijenti α i β nazivaju se amplitude vjerojatnosti. Ukupna vjerojatnost mora biti jednaka 1 pa je jasno zašto vektor stanja mora imati normu 1. Činjenica da je ukupna vjerojatnost 1 također nam omogućava da vektor stanja zapišemo kao:

$$|\psi\rangle = e^{i\gamma}(cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}sin\frac{\theta}{2}\,|1\rangle), \tag{2.4}$$

gdje su γ, θ i φ realni brojevi. Faktor $e^{i\gamma}$ nema vidljivih efekata pa ga možemo zanemariti. Kutovi θ i φ definiraju točku na trodimenzionalnoj sferi koju koristimo za grafički prikaz stanja q-bita, a zovemo ju Blochova sfera. Većina operacija koje provodimo na jednom q-bitu mogu se opisati unutar nje. Grafički prikaz nalazi se na Slici 1.

Jedan q-bit sam ne može nositi puno informacija. Što se dogodi ako imamo dva? Dva klasična bita mogu zajedno tvoriti ukupno četiri različita stanja: 00, 01, 10 i 11. Analogno tome bazu



Slika 1: *Grafički prikaz nekog vektora stanja na Blochovoj sferi. Slika izrađena u Libre Office Draw.*

sustava dvaju q-bitova (tenzorski produkt jednodimenzionalnih prostora $V^{\otimes 2} = V \otimes V$) čine četiri vektora stanja: $|00\rangle$, $|01\rangle$, $|10\rangle$ i $|11\rangle$. Sustav se također može nalaziti u superpoziciji ovih stanja.
Općenito se sustav od n q-bitova opisuje normiranim vektorom stanja u prostoru koji je tenzoski produkt drugih vektorskih prostora $V^{\otimes n} = (V \otimes V \otimes ... \otimes V)$. Dimenzija tog prostora je 2^n i ima istaknutu ortonormiranu bazu:

$$\{|00...00\rangle, |00...01\rangle, |00...10\rangle, ..., |11...11\rangle\}. \tag{2.5}$$

Svaki vektor baze $|x\rangle_n$ tenzorski je produkt n dvo-komponentnih vektora $|x_j\rangle$:

$$|x\rangle_n = |x_{n-1}\rangle \otimes |x_{n-2}\rangle \otimes ... \otimes |x_1\rangle, \tag{2.6}$$

navest ćemo jedan primjer:

$$|101\rangle = |1\rangle \otimes |0\rangle \otimes |1\rangle. \tag{2.7}$$

Ovo nas može dovesti do zaključka da se svaki vektor stanja može "faktorizirati" kao tenzorski produkt drugih vektora stanja. Primjerice, ako imamo $|\psi_1\rangle = \alpha|0\rangle + \beta|1\rangle$, $|\psi_2\rangle = \gamma|0\rangle + \delta|1\rangle$, onda je:

$$|\psi_1\rangle \otimes |\psi_2\rangle = \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle. \tag{2.8}$$

Međutim, općenito to nije slučaj, jedan od vektora koji se ne mogu rastaviti na takav način je $\frac{1}{\sqrt{34}}(|00\rangle + 2|01\rangle + 2|10\rangle + 5|11\rangle)$. U ovom slučaju kažemo da su opisani q-biti kvantno spregnuti. To znači da mjerenjem stanja jednoga, mijenjamo stanje drugog. Spregnuta je stanja teško održati, stoga do sada nisu napravljena kvantna računala s više od 127 q-bitova [36].
Zamislimo da imamo vektor stanja sustava s dva q-bita:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle. \tag{2.9}$$

Analogno kao za jedan q-bit, kompleksni koeficijenti α_{ij} su amplitude vjerojatnosti. Kvadrat modula amplitude vjerojatnosti predstavlja vjerojatnost da sustav nakon mjerenja bude u stanju baze kojem ona pripada. Ukupna vjerojatnost mora biti jednaka 1 pa imamo uvjet normalizacije $\sum_{i,j=0}^{1} |\alpha_{ij}| = 1$. Tako dobijemo normirani vektor stanja:

$$|\psi'\rangle = \frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2}}. \tag{2.10}$$

2.2 Čista i miješana stanja

U kvantnoj mehanici sustave možemo opisati matricama gustoće, koje ćemo ovdje definirati. Takozvano čisto stanje sustava zapisujemo kao vektor u Hilbertovom prostoru $H = \mathbb{C}^n$. Općenito:

$$|\psi\rangle = \sum_{q_1 \ldots q_n \in \{0,1\}} \alpha_{q_1 \ldots q_n} |q_1 \ldots q_n\rangle . \tag{2.11}$$

Miješana stanja prikazujemo matricom gustoće ρ, koja je element Hilbertovog prostora $H = \mathbb{C}^n \times \mathbb{C}^n$. Ako je stanje ρ miješavina m čistih stanja $|\psi_i\rangle$, definiramo matricu gustoće:

$$\rho = \sum_{i=1}^{m} p_i |\psi_i\rangle \langle\psi_i| , \tag{2.12}$$

gdje je $\sum_i p_i = 1$, $p_i \geq 0$ su statističke težine čistih stanja. Matrica gustoće za čisto stanje je poseban slučaj izraza za miješano stanje:

$$\rho = |\psi\rangle \langle\psi| . \tag{2.13}$$

Neka svojstva matrica gustoće:

$$Tr(\rho) \leq 1, \tag{2.14}$$

$$\rho \geq 0. \tag{2.15}$$

Za čista stanja vrijedi $\rho^2 = \rho \rightarrow Tr(\rho^2) = 1$.
Matrice gustoće korisne su jer omogućuju povezivanje svog traga s mjerenjima. Trag umnoška matrice gustoće čistog stanja i nekog hermitskog operatora u nekoj bazi $\{|O_j\rangle\}$, koja sadrži m elemenata izgleda ovako:

$$Tr(\rho\hat{O}) = \sum_j \langle O_j| \rho\hat{O} |O_j\rangle = \sum_j \langle O_j| |\psi\rangle \langle\psi| \hat{O} |O_j\rangle = \langle\psi| \hat{O} |\psi\rangle , \tag{2.16}$$

gdje smo uzeli u obzir da je $\sum_j |O_j\rangle \langle O_j| = 1$.
Sustav s n q-bita ima matricu gustoće koja je tenzorski produkt matrica gustoće sustava s 1

q-bitom:

$$\rho = \sum_i p_i \rho_1^i \otimes ... \otimes \rho_n^i. \tag{2.17}$$

Ako iz ovoga želimo dobiti samo određe podsustave, to činimo uzimajući parcijalni trag matrice gustoće. Primjerice, ako želimo ukloniti i-tu komponentu sustava imamo:

$$Tr_i(\rho_1 \otimes ... \otimes \rho_n) = \rho_1 \otimes ... \otimes \rho_{i-1} \otimes \rho_{i+1} \otimes ... \otimes \rho_n. \tag{2.18}$$

Na primjer, ako imamo dva stanja matrica gustoće je:

$$\rho_{AB} = \rho_A \otimes \rho_B. \tag{2.19}$$

Za stanje sustava imamo:

$$|\psi_{AB}\rangle = \sum_j c_j \left|j\right\rangle_A \otimes \left|j\right\rangle_B, \tag{2.20}$$

pa matricu gustoće možemo zapisati i kao:

$$\rho_{AB} = \left|\psi\right\rangle \left\langle\psi\right|_{AB} = \sum_{jk} c_j c_k^* \left|j\right\rangle \left\langle k\right|_A \otimes \left|j\right\rangle \left\langle k\right|_B. \tag{2.21}$$

Ako želimo izdvojiti npr. stanje B:

$$\begin{aligned}
\rho_B =& Tr_A \rho_{AB} = \sum_{i \in A} \left\langle i\right| \rho_{AB} \left|i\right\rangle \\
=& \sum_{i \in A} \sum_{jk} c_j c_k^* \left\langle j|k\right\rangle \left\langle k|i\right\rangle \otimes \left|i\right\rangle \left\langle k\right|_B \\
=& \{\langle i|j\rangle = \delta_{ij}, \langle k|i\rangle = \delta_{jk}\}
\end{aligned}$$

$$\Rightarrow \rho_B = \sum_i |c_i|^2 \left|i\right\rangle \left\langle i\right|. \tag{2.22}$$

S druge strane, ako mjerimo stanja koja pripadaju sustavu A dobijemo $\left|j\right\rangle_A \otimes \left|j\right\rangle_B$ s vjerojatnošću $|c_j|^2$. Ako ne gledamo što smo dobili nego to samo odbacimo dobijemo ansambl čistih stanja $\left|j\right\rangle_B$ od kojih svako ima vjerojatnost $p_j = |c_j|^2$:

$$\rho_B = \sum_j |c_j|^2 \left|j\right\rangle \left\langle j\right|. \tag{2.23}$$

Vidimo da smo dobili isti rezultat kao u (2.22).

2.3 Kvantna vrata i kvantni krugovi

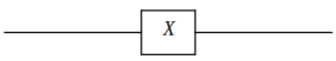
Na klasičnim računalima svaka operacija može se opisati logičkim krugovima. Započinjemo postavljanjem početne vrijednosti svakog bita. Na njih primjenjumo niz logičkih vrata (AND, OR, NOR...) kako bismo ostvarili željenu operaciju. Na kraju očitajemo vrijednosti bitova da bismo znali rezultat operacije.
Kvantna računala rade na sličan princip. U kvantnim krugovima koristimo kvantna vrata kako bismo manipulirali početnim stanjima q-bitova i dobili željene rezultate. Osnovna razlika u načinu rada je da klasična vrata mijenjaju stanje bita jednostavnom promjenom nule u jedinicu, a kvantna vrata su unitarni operatori koje primjenjujemo na vektore stanja q-bitova u Hilbertovom prostoru. Unitarni operatori čuvaju normu, što znači da preslikavaju vektor stanja u vektor stanja [6].
Unitarnost također znači da vrijedi relacija

$$\hat{U}^\dagger \hat{U} = \hat{U}\hat{U}^\dagger = 1, \tag{2.24}$$

koja nas upućuje na to da $\hat{U}$ ima inverzni operator, tj. da je kvantni krug reverzibilan. Ovo je jako važno jer kad bismo proveli neku ireverzibilnu operaciju, više ne bi bilo načina za dobiti informaciju o prijašnjim stanjima q-bitova, a to bi značilo da je izvršeno mjerenje i krug bi bio završen. Kvantni krugovi sastoje se od 3 dijela: pripreme početnih stanja q-bitova, izvršavanja kvantnih vrata (operatora) i mjerenja konačnog stanja.
Crtanje dijagrama kvantnih krugova započinjemo ravnom linijom. Početno stanje predstavljamo s lijeve strane linije. n q-bita u tom stanju označavamo s kosom crtom i n preko ravne linije. Preko te linije crtamo i simbole koji predstavljaju kvantna vrata koja provodimo nad njima.
Proći ćemo kroz osnovna vrata koja ćemo koristiti prilikom izrade našeg modela kvantne neuronske mreže. Počet ćemo s onima koja djeluju na jedan q-bit. Prvo imamo X ili NOT vrata. Kao što se da zaključiti iz naziva, ona mijenjaju stanje $|0\rangle$ u stanje $|1\rangle$ i obratno. Predstavljaju rotaciju oko osi x za kut π. Matrično ih prikazujemo na sljedeći način:

$$X := \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{2.25}$$

Postoje 2 načina prikaza u dijagramu kvantnog kruga, jedan je sa kružićem koji ima plus unutar sebe, a drugi je vidljiv na Slici 2.

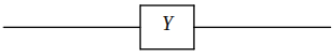


Slika 2: *Prikaz X ili NOT vrata u dijagramu kvantnog kruga. Slika izrađena u Libre Office Draw.*

Slična X vratima imamo Y vrata. Ona su rotacija oko osi y za kut π. Matrični prikaz je:

$$Y := \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}. \tag{2.26}$$

Njihov prikaz u dijagramu kvantnog kruga nalazi se na Slici 3.



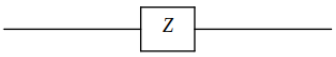
Slika 3: *Prikaz Y vrata u dijagramu kvantnog kruga Slika izrađena u Libre Office Draw..*

Naravno, imamo i Z vrata. Ona su rotacija oko osi z za kut π. Matrični prikaz je:

$$Z := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{2.27}$$

Prikaz na dijagramu kvantnog kruga nalazi se na Slici 4.

Što ako želimo zarotirati vektor stanja za proizvoljan kut oko neke osi? Napisat ćemo primjer



Slika 4: *Prikaz Z vrata u dijagramu kvantnog kruga. Slika izrađena u Libre Office Draw.*

za operator rotacije faze, R_φ ili P. Matrični prikaz je:

$$R_\varphi := \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix}. \tag{2.28}$$

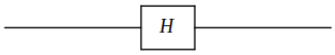
Jedna manje poznata, ali nama bitna vrata su rotacija oko osi y za proizvoljni kut φ, $R_y(\varphi)$. Njihov matrični prikaz je:

$$R_y(\varphi) := \begin{bmatrix} cos\left(\frac{\varphi}{2}\right) & -sin\left(\frac{\varphi}{2}\right) \\ sin\left(\frac{\varphi}{2}\right) & cos\left(\frac{\varphi}{2}\right) \end{bmatrix}. \tag{2.29}$$

Sada pogledajmo Hadamardova ili H vrata. Ona su jedan od najbitnijih elemenata kvantnih krugova jer prebacuju stanja baze $|0\rangle$ ili $|1\rangle$ u superpoziciju tih dvaju stanja. Njihov matrični prikaz je:

$$H := \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{2.30}$$

Prikaz na dijagramu kvantnog kruga se nalazi na Slici 5. Za bolju predodžbu djelovanja ovog



Slika 5: *Prikaz H vrata u dijagramu kvantnog kruga [1]. Slika izrađena u Libre Office Draw.*

operatora kao primjer ćemo raspisati njegovo djelovanje na stanja $|0\rangle$ i $|1\rangle$:

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \tag{2.31}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}}, \tag{2.32}$$

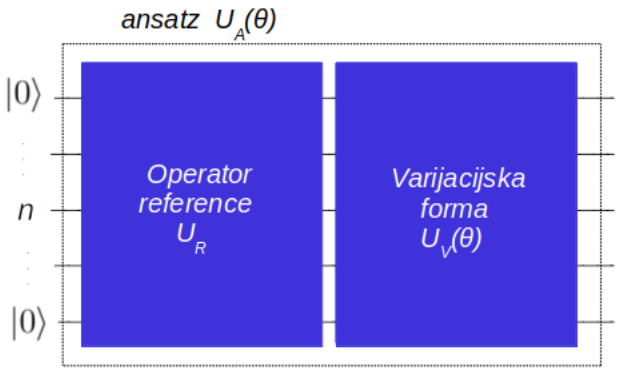
Od vrata koja djeluju na jedan q-bit treba još spomenuti vrata identiteta I. Matrični prikaz je:

$$I := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{2.33}$$

U dijagramu kvantnog kruga prikazuje se analogno ostalim kvantnim vratima, tj. kao kućica sa slovom I u sredini.
Od vrata koja djeluju na dva q-bita nama su bitna $CNOT$ ili kontrolirana NOT vrata. Kod njih imamo kontrolni q-bit i q-bit koji je meta. Ako je kontrolni q-bit u stanju $|0\rangle$ ništa se ne događa, a ako je u stanju $|1\rangle$ primjenjujemo NOT vrata na drugi q-bit. Matrični prikaz je:

$$CNOT := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2.34}$$

Ova vrata koristimo kako bismo spregnuli dva q-bita.



Slika 6: *Grafički prikaz ansatza. Slika izrađena u Libre Office Draw.*

3 Alati potrebni za kvantno strojno učenje

3.1 Parametrizirani kvantni krugovi

Varijacijski ili parametrizirani algoritmi provode se u nekoliko koraka. Prvo moramo pripremiti sustav za provedbu algoritma, zatim pripremamo parametrizirani kvantni krug, nakon toga izračunavamo funkciju cijene te optimiziramo parametre. Cilj je birajući optimalne parametre za dane ulazne podatke dobiti željeni ishod. Početno stanje označavamo $|\psi_0\rangle$, a ono je obično takvo da su svi q-biti u stanju $|0\rangle$, tj. $|\psi_0\rangle = |0\rangle^{\otimes n}$. Prvi korak u krugu je ovo stanje transformirati u željeno fiksno stanje koje nazivamo referentno [26]. Ako već znamo neke podatke o željenim ishodima algoritma postavljanje njih za referentno stanje može uvelike olakšati put do rješenja. Referentno stanje postavljamo koristeći operator reference:

$$U_R |0\rangle = |\rho\rangle \,. \tag{3.1}$$

Parametrizirani kvantni krugovi sastoje se od kvantnih vrata s parametrima koje je moguće podešavati. Da bismo optimizirali parametre potrebno je definirati varijacijsku formu $U_V(\vec{\theta})$ koja reprezentira skup parametriziranih stanja koje naš parametrizirani algoritam istražuje [27]. Kombinacija referentnog stanja i varijacijske forme naziva se *ansatz* i označava sa $U_A(\vec{\theta})$.

$$U_A(\vec{\theta}) |0\rangle = U_V(\vec{\theta}) |\rho\rangle = U_V(\vec{\theta}) U_R |0\rangle \,. \tag{3.2}$$

Na Slici 6 nalazi se prikaz ovog dijela algoritma na dijagramu kvantnog kruga.
Nakon izvršenog parametriziranog dijela algoritma provodi se mjerenje te se računa funkcija cijene. Ona nam govori o tome koliko je dobiveni ishod blizu željenom. S obzirom na njenu vrijednost optimiziraju se parametri.
Kako bi parametrizirani kvantni krug bio dobar i koristan potrebno je da može doći do značajnog broja stanja unutar Hilbertovog prostora u kojem dobivamo rezultate, a potrebno je i

da može sprezati q-bitove. U protivnom bi bilo vrlo lako simulirati njegov rad na klasičnom računalu. Pokrivenost rezultirajućeg Hilbertovog prostora zove se ekspresibilnost i što je veća krug je bolji. Na primjer, zamislimo parametrizirani kvantni krug koji se sastoji od Hadamardovih vrata i rotacije oko osi z za proizvoljan kut θ. Hadardovim vratima možemo doći do nekog stanja na kružnici radijusa 1 u xy ravnini, a $R_z(\theta)$ će dobiveno stanje zarotirati na način da će se novo stanja opet nalaziti na istoj kružnici. Ako ovome krugu dodamo, primjerice, rotaciju oko osi y, dobit ćemo puno veći skup stanja do kojih možemo doći provodeći sustav kroz taj krug.
U kvantnom strojnom učenju parametrizirani kvantni krugovi nailaze na dvije primjene. Prva je kodiranje podataka gdje se parametri podešavaju prema klasičnim podacima koje kodiramo. Druga je da ih koristimo kao kvantne modele, gdje se parametri namještaju u procesu optimizacije.

3.2 Kodiranje podataka

Kako bi se izvršavali algoritmi kvantnog strojnog učenja potrebno je uspješno ubaciti klasične podatke u kvantni sustav. Proces prebacivanja klasičnih podataka u kvantna stanja naziva se *kodiranje podataka*. Loše kodiranje podataka može ozbiljno pogoršati kvalitetu rezultata. Zamislimo da imamo klasični skup podataka $\mathscr{K}$, koji se sastoji od M primjeraka, od kojih svaki ima N svojstava:

$$\mathscr{K} = \{x^{(1)}, x^{(2)}, ..., x^{(M)}\}, \tag{3.3}$$

svaki $x^{(m)}$ je N-dimenzionalni vektor. Kodiranje možemo promatrati kao parametrizirani kvantni krug koji djeluje na stanje $|0^n\rangle$ (n q-bitova u stanju $|0\rangle$), u kojem su parametri određeni klasičnim podacima. Postoje brojni načini kodiranja podataka, a u ovom radu proći ćemo kroz neke od popularnijih.
Najintuitivniji način za prebaciti klasične podatke u kvantna stanja je kodiranje bazom. Ono prebacuje n-bitni binarni string x u n-q-bitno kvantno stanje $|x\rangle$ [35]. Dakle, podatak prebaci u binarni zapis i zatim ga preslika u odgovarajuće kvantno stanje u zadanoj bazi. Npr. za N-bitni string $x = (b_1, b_2, ..., b_N)$ dobili bismo $|x\rangle = |b_1, b_2, ..., b_N\rangle$, gdje je $b_n \in \{0, 1\}$, $n = 1, 2, ..., N$. Za navedeni klasični skup podataka $\mathscr{K}$ svaki $x^{(m)}$ preslikat će se u odgovarajući $|x^m\rangle$. Cijeli skup možemo prikazati kao superpoziciju stanja baze:

$$|\mathscr{K}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} |x^{(m)}\rangle \,. \tag{3.4}$$

Prethodni se postupak uglavnom koristi za kodiranje realnih brojeva.
Kompleksniji način kodiranja, pogodniji za vektore je kodiranje amplitudom. Provodi se tako da preslikavamo N-dimenzionalni **x** na n q-bita, koji imaju $2^n = N$ stanja baze i 2^n pripadajućih

amplituda.

$$|\psi_x\rangle = \sum_{i=1}^{N} x_i \, |i\rangle \, , \tag{3.5}$$

gdje je x_i i-ta komponenta vektora **x**, a $|i\rangle$ i-to stanje baze Hilbertovog prostora. Da bismo gornji klasični skup podataka $\mathscr{K}$ kodirali potreban nam je normirani vektor s $N \times M$ elemenata [33]:

$$\alpha = A_{norm.}(x_1^{(1)}, ..., x_N^{(1)}, ..., x_1^{(m)}, ..., x_N^{(m)}, ..., x_1^{(M)}, ...x_N^{(M)}), \tag{3.6}$$

gdje je $A_{norm.}$ konstanta normalizacije koja osigurava da je ukupna duljina α jednaka jedan, osnosno da je normiran. Za prikazati cijeli skup podataka sada imamo izraz:

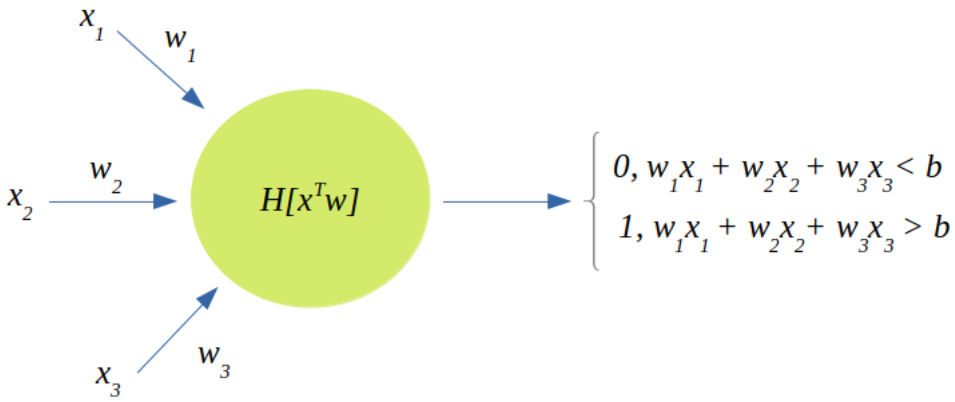
$$|\mathscr{K}\rangle = \sum_{i=1}^{N \times M} \alpha_i \, |i\rangle \, . \tag{3.7}$$

Mnogo algoritama kvantnog strojnog učenja traži ovaj način kodiranja podataka.
Kodiranje kutom koristi se rotacijskim kvantnim vratima na način da prevodi klasične podatke u kuteve rotacije:

$$|x\rangle = \bigotimes_{i=1}^{N} R(x_i) \, |0^n\rangle \, . \tag{3.8}$$

Rotacija može biti oko bilo koje osi, a potreban broj q-bita obično je jednak dimenziji podatka kojeg kodiramo. U ovom načinu kodiranja ne možemo odjednom kodirati cijeli skup podataka. Postoje i brojni drugi načini kodiranja podataka koji prebacuju N svojstava u rotacije na N parametriziranih kvantnih vrata na n q-bita, $n \leq N$.

4 Klasične neuronske mreže



Slika 7: *Grafički prikaz perceptrona, s x_i označeni su ulazni podaci, a sa w_i pripadne težine s kojima ih taj perceptron množi. Njegova pristranost označena je s b, a izlaz (rezultat Heaviside funkcije) označen je s H. Slika izrađena u Libre Office Draw.*

Strojno učenje je područje unutar umjetne inteligencije koje se bavi oblikovanjem algoritama koji poboljšavaju učinkovitost na temelju empirijskih podataka. Obično se koriste skupovi podataka koji se sastoje od podataka za koje treba odrediti nešto i točne vrijednosti onoga što treba odrediti. Algoritmi u sebi sadrže parametre koji se u svakom koraku optimiziraju nakon provjere ispravnosti na danom skupu podataka. Neuronske mreže posebno su područje unutar strojnog učenja. Za cilj imaju simuliranje rada ljudskog mozga, gradimo ih od umjetnih neurona koji provode razne operacije s dobivenim skupom podataka.
Umjetni neuron je vektor (**x**, **w**, φ, y), gdje je $\mathbf{x}^T = (x_0, x_1, ..., x_N) \in \mathbb{R}^{n+1}, x_0 = -1$ vektor podataka koje obrađujemo, $\mathbf{w}^T = (w_0, w_1, ..., w_N)$ vektor težina, φ je aktivacijska funkcija, a rezultat neurona je definiran s $y = \varphi(x^T w)$ [9]. Prvi element vektora težina naziva se i pristranost (eng. *"bias"*) i označava s b. Neuron funkcionira tako da primljene ulazne podatke množi s pripadnim težinama, sumira sve umnoške i sumu ubacuje kao argument u aktivacijsku funkciju. S obzirom na to da je $x_0 b = -b$, ovo se može lijepo zapisati formulom:

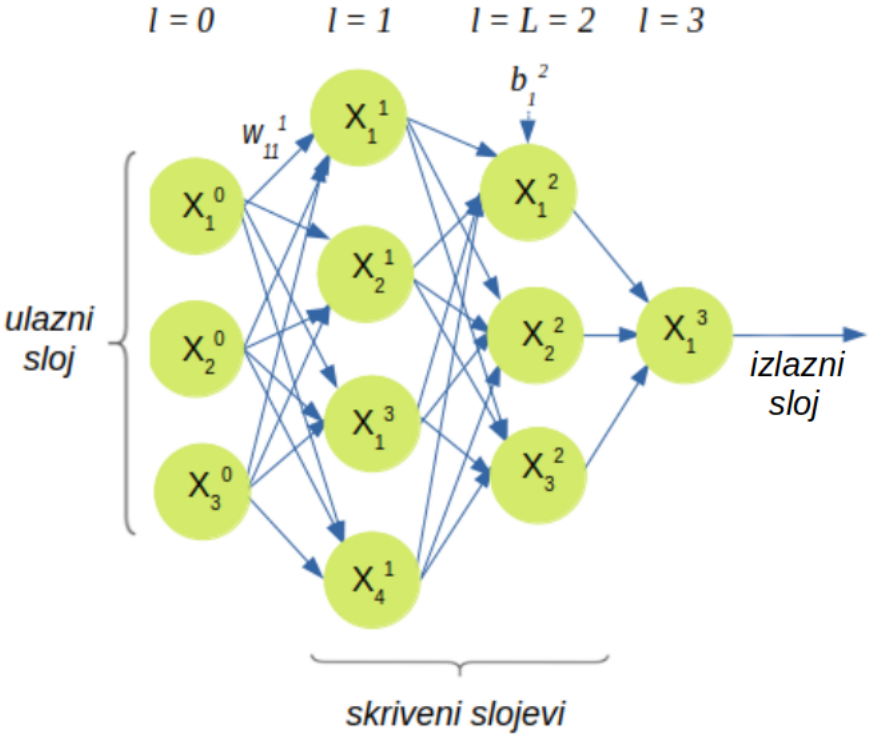
$$y = \varphi\left(\sum_{i=1}^{N}(x_i w_i - b)\right). \tag{4.1}$$

Perceptron je umjetni neuron sa svojstvom da svi ulazni podaci imaju vrijednosti između 0 i 1 ($x_i \in \{0, 1\}, i = 1, 2, ...N$), a aktivacijska funkcija je Heaviside funkcija:

$$H(x) = \begin{cases} 0, x < 0 \\ 1, x \geq 0 \end{cases} \tag{4.2}$$

Ovisno o vrijednosti pristranosti, funkcija koja daje rezultat također poprima vrijednosti 0 ili 1. Grafički prikaz nalazi se na Slici 7. Perceptron na slici prima podatke x_i od prethodnog

neurona, oni se množe s težinama w_i, dodaje se pristranost i primjenjuje se aktivacijska funkcija H.



Slika 8: *Grafički prikaz neuronske mreže koja se sastoji od četiri sloja. Prvo imamo ulazni sloj, zatim dva skrivena pa izlazni. Broj skrivenih slojeva označen je sa L. Koristimo w^l_{jk} kako bismo označili pripadnu težinu koju koristi k-ti neuron u (l-1)-om sloju kada šalje podatke j-tom neuronu u l-tom sloju. Pristranost j-tog neurona u l-tom sloju označava se b^l_j, a na ovoj silci izlaz tog istog neurona označena je s x^l_j. Ovaj izlaz se prosljeđuje kao ulaz sljedećem sloju. Slika izrađena u Libre Office Draw.*

Skup svih perceptrona koji primaju iste ulazne podatke naziva se sloj. U ovom radu opisat ćemo model višeslojih perceptrona. Oni se sastoje od ulaznog, izlaznog i skrivenih slojeva. Grafički prikaz slojeva nalazi se na Slici 8. Koristimo w^l_{jk} kako bismo označili pripadnu težinu koju koristi k-ti neuron u (l-1)-om sloju kada šalje podatke j-tom neuronu u l-tom sloju. Pristranost j-tog neurona u l-tom sloju označava se b^l_j, a na ovoj silci izlazna funkcija tog istog neurona označena je s x^l_j. Izlazi svih neurona nekog sloja se prosljeđuju svim neuronima sljedećeg kao ulazi.

Paradigma učenja koja određuje odnos neuronskih mreža u ovom radu prema okolini je učenje s nadzorom [12]. Ideja je da je dan skup od konačno mnogo ulaza i njima odgovarajućih izlaza na temelju kojih se optimiziraju parametri mreže.

4.1 Algoritam treniranja

Učenje u višeslojnim perceptronima odvija se preko algoritma "širenja unatrag" [11]. U mreži širenja unatrag podaci se šire od ulaznog sloja preko skrivenih do izlaznog, a zatim se određuje greška koja se širi unatrag do ulaznog sloja, gdje se ugrađuje u formulu za učenje. Ovo se događa u nekoliko koraka. Najprije ulazni sloj učitava podatke, primjenjuje aktivacijsku funkciju i šalje ih dalje u prvi skriveni sloj. Opet se primjenjuje aktivacijska funkcija te se podaci šalju u sljedeći skriveni ili izlazni sloj. Kada podaci stignu do izlaznog sloja u njemu se za svaki podatak funkcijom cijene $C(w, b)$ (još se naziva i funkcija greške) računa razlika između željenog i dobivenog izlaza. Koristeći algoritam prosljeđivanja unatrag računa se gradijent funkcije cijene te se na kraju gradijentnim spustom korigiraju vrijednosti težina i pristranosti.
Uzmimo da je dan skup podataka za treniranje $s = (x_j^0, y^j)_{j=1}$, gdje su $x_j^0 \in X$ ulazni parametri mreže s pripadnim oznakama $y^j \in Y$. Osim skupa podataka za treniranje možemo odvojiti još i skup za validaciju i skup za testiranje. Točnost modela definira se kao postotak točno određenih oznaka. Točnost na skupu za treniranje pomaže u namještanju parametara mreže, a točnost na skupu za testiranje nam govori koliko je dobro mreža istrenirana [9].
Cilj nam je da razlika između željene i dobivene oznake bude minimalna, stoga se proces učenja neuronske mreže svodi na tražnje izlazne funkcije $f : X \to Y$, za koju će funkcija cijene biti najmanja.
Podatke prvo provlačimo unaprijed kroz neuronsku mrežu. Za j-ti neuron u l-tom sloju izlaz će biti:

$$x_j^l = \varphi\left(\sum_{k=1}^{m_{l-1}} w_j^l x_k^{l-1} - b_j^l\right), \tag{4.3}$$

gdje je m_{l-1} ukupan broj neurona u (l-1) sloju. Ulazne podatke, pripadne težine i pristranosti u jednom sloju možemo prikazati vektorima:

$$X^l = (x_1^l, ..., x_{m_l}^l)^T, \quad W^l = (w_{jk}^l)_{j,k}, \quad B^l = (b_1^l, ..., b_{m_l}^l)^T. \tag{4.4}$$

Ovaj zapis možemo iskoristiti za matrični zapis izlaza svih neurona l-tog sloja:

$$X^l = \varphi\left(W^l X^{l-1} - B^l\right). \tag{4.5}$$

4.2 Prosljeđivanje unatrag

Tražimo minimum funkcije cijene algoritmom s povratnim prosljeđivanjem pogreške uz pomoć metode gradijentnog spusta. Funckija cijene ima težine i pristranost za varijable, što omogućava zapis njenog gradijenta kao $\nabla C = (\nabla_w C, \nabla_b C)$. Derivaciju funkcije cijene po težinama pišemo kao:

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial s_j^l}\frac{\partial s_j^l}{\partial w_{ij}}, \tag{4.6}$$

gdje je $s_j^l = \sum_{k=1}^{m_{l-1}} w_j^l x_k^{l-1} - b_j^l$ signal koji j-ti neuron u l-tom sloju prosljeđuje dalje. Derivaciju funkcije cijene po pristranostima pišemo kao:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial s_j^l} \frac{\partial s_j^l}{\partial b_j^l}, \tag{4.7}$$

Uzevši u obzir da je $\frac{\partial s_j^l}{\partial w_{ij}} = x_i^{l-1}$ i $\frac{\partial s_j^l}{\partial b_j^l} = -1$ te definirajući $\delta_j^l = \frac{\partial C}{\partial s_j^l}$ dobijemo gradijent funkcije cijene:

$$\nabla C = \delta_j^l (x_i^{l-1}, -1). \tag{4.8}$$

Da bismo našli vrijednost δ_j^l krećemo od zadnjeg sloja i idemo unatrag. U većini neuronskih mreža uzimamo da se nalazimo u euklidskom prostoru. Kako funkcija cijene treba provjeriti sličnost između dva podatka, želimo naći njihovu udaljenost i učiniti ju minimalnom. Iz tog razloga funkcija cijene uglavnom je kvadratna:

$$C(w, b) = \frac{1}{2m_{L+1}} \sum_{j=1}^{m_{L+1}} (x_j^{L+1} - y_i)^2, \tag{4.9}$$

L označava zadnji skriveni sloj pa je m_{L+1} broj neurona u izlaznom sloju. Za zadnji sloj imamo:

$$\delta_j^{L+1} = \frac{\partial C}{\partial s_j^{L+1}} = \frac{1}{m_{L+1}} (x_j^{L+1} - y^i) \varphi'(s_j^{L+1}), \tag{4.10}$$

gdje smo uzeli u obzir da je $x_j^l = \varphi(s_j^l)$. Sada možemo δ_j^{l-1} zapisati preko δ_j^l:

$$\delta_k^{l-1} = \frac{\partial C}{\partial s_k^{l-1}} = \sum_{j=1}^{m_l} \frac{\partial C}{\partial s_j^l} \frac{\partial s_l^j}{\partial s_k^{l-1}} = \sum_{j=1}^{m_l} \delta_j^l \frac{\partial s_l^j}{\partial s_k^{l-1}}, \tag{4.11}$$

Iskoristivši da je:

$$\frac{\partial s_l^j}{\partial s_k^{l-1}} = \frac{\partial (\sum_{k=1}^{m_l} w_{jk}^l \varphi(s_k^{l-1}) - b_j^l)}{\partial s_k^{l-1}} = w_{ij}^l \varphi'(s_k^{l-1}) \tag{4.12}$$

Dobije se formula za povratno prosljeđivanje pogreške:

$$\delta_k^{l-1} = \varphi'(s_k^{l-1}) \sum_{j=1}^{m_l} \delta_j^l w_{ij}^l. \tag{4.13}$$

Gradijent funkcije definira smjer njenog najbržeg rasta, dakle smjer najbržeg pada pokazuje negativni gradijent. Minimum funkcije cijene onda tražimo u smjeru negativnog gradijenta. Kako bismo pronašli neki novi x, moramo se od sadašnjega pomaknuti u smjeru negativnog gradijenta funkcije cijene [12]:

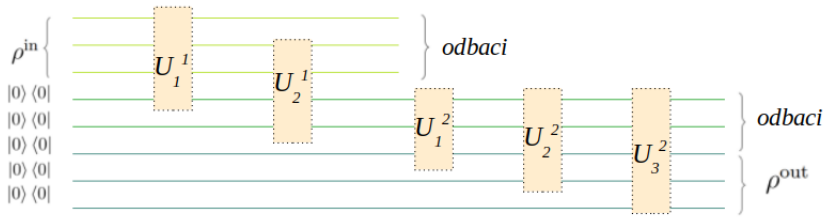
$$x \to x' = x - \eta \nabla C. \tag{4.14}$$

Primjenivši ovaj izraz na neuronsku mrežu dobijemo:

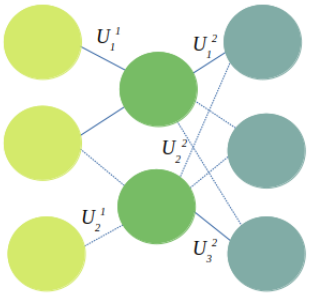
$$w_{ij}^l(n+1) = w_{ij}^l(n) - \eta \delta_j^l(n) x_i^{l-1}(n). \tag{4.15}$$

$$b_j^l(n+1) = b_j^l(n) + \eta \delta_j^l(n), \tag{4.16}$$

gdje su $x_i^l(n)$ i $\delta_j^l(n)$ funkcije težina $w_{ij}^l(n)$ i $b_j^l(n)$ dobivenih u n-tom koraku. Algoritam ponavljamo dok se ne izvrši unaprijed određen broj koraka. Postoje i drugi kriteriji zaustavljanja algoritma, no oni su nevažni za potrebe ovog rada.



Slika 9: *Prikaz kvantne neuronske mreže u dijagramu kvantnog kruga. U_j^l označava j-ti perceptron koji djeluje između slojeva l i $l-1$. Slika izrađena u Libre Office Draw.*



Slika 10: *Shematski prikaz radi usporedbe s klasičnom arhitekturom. U_j^l označava j-ti perceptron koji djeluje između slojeva l i $l-1$. Slika izrađena u Libre Office Draw.*

5 Kvantne neuronske mreže

Motivacija za kvantno strojno učenje je što bi mogućnosti kvantnog računanja u spoju s metodama klasičnog strojnog učenja otvorile vrata novim, naprednijim algoritmima učenja. Kvantne neuronske mreže kombiniraju principe klasičnih neuronskih mreža i parametrizirane kvantne krugove, stoga ih se može promatrati na dva načina. Prvi je iz perspektive klasičnog strojnog učenja, one su algoritmi koje učimo opažati pravilnosti u skupovima podatka. One kodiraju klasične podatke u kvantne i provlače ih kroz parametrizirane kvantne krugove koje možemo trenirati. Nakon obrade podataka mjeri se stanje i rezultat se može poslati funkciji cijene koja namješta težine kroz prosljeđivanje unatrag. Drugi način je iz perspektive kvantnog računarstva, one su kvantni algoritmi bazirani na parametriziranim kvantnim krugovima [31]. Kvantni perceptron definiran je parametriziranom unitarnom matricom U, dimenzije $2^{n+m} \times 2^{n+m}$, gdje je m broj ulaznih, a n izlaznih q-bitova [9]. U ovom radu teorijski ćemo uvesti ekvivalent višeslojnog perceptrona u kvantnom računarstvu, disipativne kvantne neuronske mreže. Ovaj model uveden je u [10], a obrađen je i u [9] te ćemo se u ovom radu osloniti na izvode u njima. Grafički prikaz nalazi se na Slici 9. Ovdje U_j^l označava j-ti perceptron koji djeluje između slojeva l i $l-1$. Obično se grafički prikaz gradi tako da prvo nacrtamo perceptron koji prvi djeluje, zatim redom ostale ispod njega.

5.1 Algoritam treniranja

Prije početka treniranja kodiramo klasičan skup podataka u kvantna stanja. Cilj algoritma je da neuronska mreža nauči koji skup unitarnih matrica primjeniti kako bi svakom ulaznom stanju pridužila odgovarajuće izlazno stanje. Parove podataka definiramo kao $|\phi_x^{in}\rangle, |\phi_x^{des}\rangle$. Ulazno stanje je $\rho_x^{in} = |\phi_x^{in}\rangle \langle\phi_x^{in}|$, a $|\phi_x^{des}\rangle$ željeni rezultat. Izlazno stanje je $\rho_x^{out} = |\phi_x^{out}\rangle \langle\phi_x^{out}|$. Za optimizaciju mreže potrebno je usporediti željena i dobivena stanja te prilagoditi parametre unitarnih operatora.
Sličnost para kvantnih stanja mjeri se pomoću vjernosti (eng. *"fidelity"*). Neka su ρ_1 i ρ_2 čista stanja u Hilbertovom prostoru. Vjernost tih dvaju stanja definira se kao:

$$F(\rho_1, \rho_2) = F(|\psi_1\rangle \langle\psi_1|, |\psi_2\rangle \langle\psi_2|) = |\langle\psi_1|\psi_2\rangle|^2. \tag{5.1}$$

Ako uzmemo da je ρ čisto stanje i želimo provjeriti vjernost s nekim miješanim stanjem σ koristimo definiciju Uhlmann-Josza vjernosti [20]:

$$F(\rho, \sigma) = \left(Tr\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right)^2. \tag{5.2}$$

Za slučaj dva čista stanja:

$$F(\rho, \rho) = \left(Tr\sqrt{\sqrt{\rho}\rho\sqrt{\rho}}\right)^2 = (Tr(\rho))^2 = 1. \tag{5.3}$$

Jedno od važnijih svojstava vjernosti je sljedeće:

$$F(U\rho U^\dagger, U\sigma U^\dagger) = F(\rho, \sigma), \tag{5.4}$$

za unitarnu matricu U. Vjernost ćemo koristiti za usporedbu dobivenog izlaznog stanja sa željenim. Kao i kod klasičnih neuronskih mreža, dijelimo podatke na skupove za treniranje, validaciju i testiranje.
Izraz kojim opisujemo djelovanje prvog perceptrona na početno stanje $|\rho^{in}\rangle$:

$$U_1^1 \otimes \mathbb{I}_1^1(\rho^{in} \otimes |0...0\rangle_{hid,out} \langle 0...0|)U_1^{1\dagger} ... \otimes \mathbb{I}_1^1 \tag{5.5}$$

gdje je $\mathbb{I}_1^1$ matrica identiteta koja djeluje na one q-bite koje U_1^1 ne obuhvaća, a $|0...0\rangle_{hid,out} \langle 0...0|$ predstavlja stanje nultog produkta koje obuhvaća neurone u skrivenim slojevima i izlaznom sloju. Radi jednostavnosti pišemo $U_j^l \otimes \mathbb{I}_j^l \equiv U_j^l$. Perceptroni se primjenjuju jedan za drugim, što znači da uzastopno primjenjujemo unitarne transformacije. Umnožak unitarnih operatora je također unitarni operator stoga ukupno djelovanje svih perceptrona u prvom sloju možemo zapisati kao:

$$U_{m_1}^1 ... U_1^1(\rho^{in} \otimes |0...0\rangle_{hid,out} \langle 0...0|)U_1^{1\dagger} ... U_{m_1}^{1\dagger}, \tag{5.6}$$

gdje je m_1 broj perceptrona u prvom sloju. Nakon niza perceptrona za prvi sloj primjenjivali bi niz za drugi sloj itd. sve do izlaznog sloja. Taj niz primjenjenih perceptrona možemo prikazati jedinstvenom unitarnom matricom U pa se tako dobije:

$$U = (U_1^1...U_{m_1}^1)...(U_1^l...U_{m_l}^l)...(U_1^{out}...U_{m_{out}}^{out}) = U^1...U^l...U^{out}. \tag{5.7}$$

$$U(\rho^{in} \otimes |0...0\rangle_{hid,out} \langle 0...0|)U^\dagger. \tag{5.8}$$

Sada konačno stanje ρ^{out} možemo dobiti kao:

$$\rho^{out} = Tr_{in,hid}(U(\rho^{in} \otimes |0...0\rangle_{hid,out} \langle 0...0|)U^\dagger), \tag{5.9}$$

gdje je $Tr_{in,hid}$ parcijalni trag sustava q-bita u ulaznom i skrivenim slojevima. Iz definicije parcijalnog traga jasno je da uzimanjem parcijalnog traga po ulaznom i skrivenim slojevima ostavljamo samo izlazni sloj.
Bitna osobina ovog algoritma je da se izlazno stanje može zapisati kao kompozicija niza potpuno pozitivnih preslikavanja niza u niz [21].

$$\rho^{out} = \mathscr{E}(\rho^{in}), \tag{5.10}$$

$$\rho^{out} = \mathscr{E}^l(\mathscr{E}^{l-1}(...\mathscr{E}^2(\mathscr{E}^1(\rho^{in}))...)) \tag{5.11}$$

$$\mathscr{E}^l(X^{l-1}) = Tr_{l-1}\left(\Pi_{j=m_l}^1 U^l(X^{l-1} \otimes |0...0\rangle_l \langle 0...0|)\Pi_{j=1}^{m_l} U^{l\dagger}\right). \tag{5.12}$$

Ovo nam pokazuje važnu karakteristiku algoritma, a to je da informacije putuju od ulaza do izlaza te tako prirodno primjenjuju prosljeđivanje unaprijed. Ovo je baza za kvantni analogon algoritma prosljeđivanja unatrag.

5.2 Prosljeđivanje unatrag

Trebamo izvesti izraz za prosljeđivanje kvantnog stanja unatrag kroz mrežu. Kako je inverz unitarnim matricama jednak njihovoj adjungiranoj matrici ($U^{-1} = U^\dagger$), to znači da ustvari tražimo adjunkt potpunog preslikavanja $\mathscr{E}$. Za naše preslikavanje

$$\mathscr{E}^l(X^{l-1}) = Tr_{l-1}(U^l(X^{l-1} \otimes |0...0\rangle_l \langle 0...0|)U^{l\dagger}). \tag{5.13}$$

taj adjunkt se dobije kao:

$$\mathscr{F}_t^l = Tr_l((\mathbb{I}_{l-1} \otimes |0...0\rangle_l \langle 0...0|)U^{l\dagger}(t)(\mathbb{I}_{l-1} \otimes X^l)U^l(t)), \tag{5.14}$$

gdje je $\mathbb{I}_{l-1}$ jedinična matrica koja djeluje na (l-1) sloj.
Definiramo funkciju cijene preko vjernosti kako bismo usporedili dobiveni izlaz ρ_x^{out} sa željenim stanjem $|\varphi^{des}\rangle$. ρ_x^{out} definirali smo kao $|\varphi_x^{out}\rangle \langle\varphi_x^{out}|$, vjernost smo definirali kao skalarni umnožak, dakle za neki x imamo:

$$F(\rho_x^{out}, |\varphi_x^{des}\rangle \langle\varphi_x^{des}|) = |\langle\varphi_x^{out}|\varphi_x^{des}\rangle|^2, \tag{5.15}$$

ovo možemo zapisati kao:

$$\langle\varphi_x^{des}|\varphi_x^{out}\rangle \langle\varphi_x^{out}|\varphi_x^{des}\rangle = \langle\varphi_x^{des}| \rho_x^{out} |\varphi_x^{des}\rangle . \tag{5.16}$$

Dakle, usrednjavamo dobiveni izlaz s obzirom na željeno stanje preko ukupnog broja korištenih podataka. Ako ukupno imamo N podataka od kojih S koristimo u procesu treniranja, tada ćemo na skupu podataka za treniranje imati:

$$C_{tr} = \frac{1}{S}\sum_{x=1}^{S} F(|\varphi_x^{tr}\rangle \langle\varphi_x^{tr}| , \rho_x^{out}) = \frac{1}{S}\sum_{x=1}^{S} \langle\varphi_x^{tr}| \rho_x^{out} |\varphi_x^{tr}\rangle . \tag{5.17}$$

Analogno se može napisati izraz za funkciju cijene na skupu za validaciju. Kako funkcija vjernosti postiže svoj maksimum kada su stanja na koja ju primjenjujemo jednaka, želimo pronaći maksimum funkcije cijene. Za njen gradijent imamo izraz:

$$\frac{\partial C_{tr}(t)}{\partial t} = \frac{i}{S}\sum_{x=1}^{S} Tr(M_{m_{L+1}}^{L+1}(t)K_{m_{L+1}}^{L+1}(t) + ... + M_1^1(t)K_1^1(t)), \tag{5.18}$$

gdje je:

$$K_j^l(t) = \frac{\eta 2^{m_{l-1}} i}{S}\sum_x Tr_{ostatak}(M_j^l(x,t)), \tag{5.19}$$

gdje je η stopa učenja, trag djeluje na q-bite čije stanje ostaje nepromijenjeno operatorom U_j^l i

$$M_j^l(x,t) = [U_j^l(t)...U_1^1(t)(\rho_x^{in} \otimes |0...0\rangle \langle 0...0|)U_1^{1\dagger}(t)...U_j^l(t), \tag{5.20}$$

$$U_{j+1}^{l\dagger}(t)...U_{m_{L+1}}^{L+1}(t)(\mathbb{I}_{in,hid} \otimes |\varphi_x^{SV}\rangle \langle\varphi_x^{SV}|)U_{m_{L+1}}^{L+1}(t)...U_{j+1}^l(t)],$$

gdje je $\mathbb{I}_{in,hid}$ jedinična matrica koja djeluje na q-bite u ulaznom i skrivenim slojevima. Iz ovih izraza se dobije formula koju koristimo za ažuriranje matrica koje predstavljaju perceptrone:

$$U_j^l(t+\epsilon) = e^{i\epsilon K_j^l(t)} U_j^l(t), \tag{5.21}$$

gdje je $\epsilon \in \mathbb{R}$.
Da sažmemo algoritam, prvo inicijaliziramo stanje, zatim provodimo prosljeđivanje podataka unaprijed pa prosljeđivanje unatrag. Na kraju ažuriramo parametre mreže i ponavljamo proces

dok se ne ispuni određeni kriterij zaustavljanja. Najčešći kriterij je unaprijed određen broj koraka.

6 Hibridna kvantno-klasična neuronska mreža

Za stvaranje kvantnih računarskih programa potrebni su nam alati za razvoj kvantnih softwera. Jedan od kompleta alata, pisan u Pythonu, otvorenog koda je Qiskit. Njegove je algoritme najlakše implementirati u Jupyter bilježnicama preko IBM Quantum Learning-a. Qiskit sadrži razne biblioteke i module s algortmima korisnim za određeno područje. Za spoj kvantnog računarstva i klasičnog strojnog strojnog učenja Qiskit je razvio biblioteku *qiskit-machine-learning*. Unutar nje nalaze se i brojni drugi moduli s klasama korisnim za razna područja unutar strojnog učenja popout modula *neural_networks*, koji sadrži klase za implementaciju hibridnih neuronskih mreža poput *EstimatorQNN* i *SamplerQNN*. Još neki od primjera su *algorithms.classifiers* i *algorithms.regressors*, koji sadrže razne algoritame za riješavanje problema klasifikacije i regresije, uključujući i neuronske mreže.
Spomenute klase za implementaciju mreža bazirane su na apstraktnoj klasi *NeuralNetwork* iz biblioteke *qiskit.primitives*. U istoj biblioteci nalaze se i dvije klase, *Sampler* i *Estimator* koje se mogu poslati kao argument klasama iz modula *qiskit-machine-learning*. One sadrže osnovne gradivne jedinice od kojih možemo izgraditi nešto koristno. *Sampler* funkcionira tako da za dobiveno stanje $|\psi\rangle$ računa vjerojatnost mjerenja svakog stanja baze. *Estimator* za dobivenu varijablu $\hat{H}$ i stanje $|\psi\rangle$ računa srednju vrijednost.
Kažemo da su ove implementacije hibridne jer one kodiraju podatke koristeći kvantne krugove, ali nakon mjerenja optimiziraju parametre klasičnim algoritmom.

6.1 Priprema podataka

Korišteni kod inspiriran je stranicom [32] te su dijelovi koda preuzeti s te stranice. Prije svega u algoritam trebamo uključiti potrebne module i klase. To činimo pozivanjem funkcije import i navođenjem imena modula i klasa, kao što je prikazano na Slici 11. Svrhu svakog od njih objasniti ćemo u daljenjem tekstu. Skup podataka koje ćemo koristiti pripremamo generirajući

```
import matplotlib.pyplot as plt
import numpy as np
from IPython.display import clear_output
from qiskit import QuantumCircuit
from qiskit.algorithms.optimizers import COBYLA
from qiskit.circuit import Parameter
from qiskit.circuit.library import RealAmplitudes, ZZFeatureMap
from qiskit.utils import algorithm_globals

from qiskit_machine_learning.algorithms.classifiers import NeuralNetworkClassifier
from qiskit_machine_learning.neural_networks import SamplerQNN, EstimatorQNN
```

Slika 11: *Pozivanje funkcije import da bi se uključili potrebni moduli i klase.*

nasumične točke u 2D prostoru. Kod kojim smo izgenerirali skup podataka nalazi se na Slici 12. Prvo definiramo num_inputs = 2, što znači da će svaki ulazni podatak imati dvije komponente. Zatim definiramo num_samples = 50, što znači da generiramo 50 podataka. U sljedećoj liniji

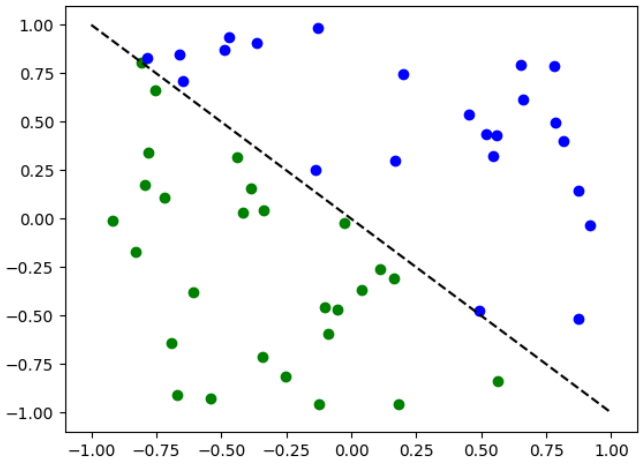
```
num_inputs = 2
num_samples = 50
X = 2 * algorithm_globals.random.random([num_samples, num_inputs]) - 1
y01 = 1 * (np.sum(X, axis=1) >= 0)  # u { 0,  1}, 0 za ispod pravca, 1 za iznad
y = 2 * y01 - 1  # u {-1, +1}, -1 za ispod pravca, 1 za iznad

for x, y_target in zip(X, y):
    if y_target == 1:                #ako je iznad pravca
        plt.plot(x[0], x[1], "bo")
    else:                            #ako nije
        plt.plot(x[0], x[1], "go")

plt.plot([-1, 1], [1, -1], "--", color="black")
plt.savefig("dataset")
plt.show()
```

Slika 12: *Kod kojim generiramo skup podataka.*

na nasumičan način generiramo ulazne podatke, oni imaju oblik:

$$x_i = (x_i^{(1)}, x_i^{(2)}), \qquad i = 1, ..., \text{num\_samples}. \tag{6.1}$$

U sljedećoj liniji provjeravamo jesu li podaci ispod ili iznad pravca $x^{(2)} = -x^{(1)}$ te dodijeljujemo oznaku 0 ako je podatak ispod, a 1 ako je iznad danog pravca. To spremamo u listu oznaka $y01$. Budući da će *EstimatorQNN* klasama pridijeliti oznake -1 i 1, preformuliramo našu listu oznaka da se poklapaju s tim, spremamo ju u varijablu y. Sljedećih nekoliko linija služe za grafički prikaz generiranog skupa podataka, koji se nalazi na Slici 13. U kodu ćemo mijenjati broj generiranih podataka kako bismo procjenili ovisnost točnosti o broju podataka. Očekivano je da će točnost rasti s brojem podataka. Skup koji smo generirali koristimo i za treniranje i za testiranje. U ovom kodu radimo bez podataka za validaciju.



Slika 13: *Grafički prikaz generiranog skupa podataka. Zelene točke ispod pravca pripadaju klasi koju smo označili sa -1, a plave točke iznad pravca klasi s oznakom 1. Slika izrađena u IBM Quantum Learning-u u Jupyter bilježnici.*

Podaci koje smo generirali su klasični, moramo ih kodirati kako bismo mogli nastaviti dalje. Koristimo klase *ZZFeatureMap* i *RealAmplitudes* iz modula *qiskit.circuit.library* za

pripremanje varijacijske forme.

Konstruktor klase *ZZFeatureMap* kao argumente uzima dimenziju dobivenih podataka, ona odgovara broju q-bita i broj ponavljanja, odnosno koliko puta treba primijeniti kvantna vrata za sprezanje q-bita i rotaciju. Krug započinje primjenom Hadamardovih vrata i rotacije pojedinačnog q-bia te se nastavlja sprezanjem q-bita i ponovnom rotacijom. Q-biti se sprežu kontroliranim NOT vratima opisanim u poglavlju Kvantna vrata. Ova klasa bazira se na klasi *PauliFeatureMap*, koja kodira podatke provodeći nad njima Paulijeve matrice rotacije. *ZZFeatureMap* je poseban slučaj u kojem se primjenjuju rotacije oko osi z. Izaz kojim ju opisujemo je [29]:

$$U_{\varphi(\vec{x})} = \left(\exp\left(i \sum_{jk} \varphi_{\{jk\}}(\vec{x}) Z_j \otimes Z_k \right) \exp\left(i \sum_{j} \varphi_{\{j\}}(\vec{x}) Z_j \right) H^{\otimes n} \right)^d, \tag{6.2}$$

gdje je d dubina, a φ funkcija koju koristimo za mijenjanje parametara s obzirom na klasične podatke, a izraz joj je [29]:

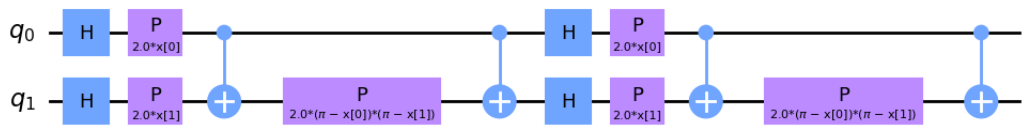
$$\varphi_S : \vec{x} \to \begin{cases} x_i, & S = i \\ (\pi - x_i)(\pi - x_j), & S = i, j \end{cases} \tag{6.3}$$

Kod za dobiti referentno stanje nalazi se na Slici 14, a prikaz u dijagramu kvantnog kruga na Slici 15.

Konstruktor klase *RealAmplitudes* kao argumente prima iste argumente kao i *ZZFeatrueMap*,

```
feature_map = ZZFeatureMap(num_inputs)
feature_map.decompose().draw(output="mpl", fold=20)
```

Slika 14: *Kod korišten za pripremu prvog dijela varijacijske forme (ZZFeatureMap).*



Slika 15: *Prikaz kvantnog kruga za izradu prvog dijela varijacijske forme (ZZFeatureMap). Slika izrađena u IBM Quantum Learning-u u Jupyter bilježnici.*

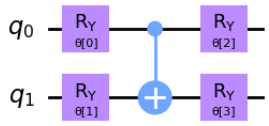
a sastoji se od izmjeničnih rotacija oko osi y i sprezanja q-bita CNOT vratima. Kod korišten za izradu varijacijske forme nalazi se na Slici 16, a prikaz u dijagramu kvantnog kruga na Slici 17.

Podaci se kodiraju uzastopnom primjenom objekata ovih dviju klasa u parametriziranom

```
ansatz = RealAmplitudes(num_inputs, reps=1)
ansatz.decompose().draw(output="mpl", fold=20)
```

Slika 16: *Kod korišten za izradu drugog dijela varijacijske forme (RealAmplitudes).*

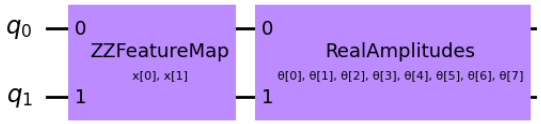
kvantnom krugu.



Slika 17: *Prikaz kvantnog kruga za izradu drugog dijela varijacijske forme (RealAmplitudes). Slika izrađena u IBM Quantum Learning-u u Jupyter bilježnici.*

```
qc = QuantumCircuit(2)
feature_map = ZZFeatureMap(2)
ansatz = RealAmplitudes(2)
qc.compose(feature_map, inplace=True)
qc.compose(ansatz, inplace=True)
qc.draw(output="mpl")
```

Slika 18: *Kod korišten za kodiranje klasičnih podataka u kvantna stanja (ansatza).*



Slika 19: *Prikaz kvantnog kruga korištenog za kodiranje klasičnih podataka u kvantna stanja (ansatza). Slika izrađena u IBM Quantum Learning-u u Jupyter bilježnici.*

6.2 Obrada podataka

Prilikom obrade podataka kreiramo objekt tipa *EstimatorQNN* (klasa iz *qiskit_machine_learning.neural_networks*). On može kao parametar primiti objekt tipa *Estimator* (klasa iz *qiskit.primitives*). Ako ga ne dobije kao argument, *EstimatorQNN* sama stvara objekt tipa *Estimator*. Kao parametar može primiti i opservablu za koju će računati očekivanu vrijednost. Također, aku ju ne primi sama ju stvara po defaultu i ona je $Z^{\otimes n}$. Princip rada opisan je u [28], ovdje donosimo kratki opis.
Očekivanu vrijednost opservable (npr. $\hat{H}$) računamo kao sredinu svih mogućih ishoda λ (svojstvena vrijednost) mjerenja stanja $|\psi\rangle$, pomnožene s odgovarajućom vjerojatnosti:

$$\langle \hat{H} \rangle_\psi = \sum_\lambda p_\lambda \lambda = \langle \psi | \hat{H} | \psi \rangle . \tag{6.4}$$

Ako je opservabla kompliciranа za mjeriti, *Estimator* ju razbije na mjerljive Paulijeve opservable ($\hat{P}$). One odgovaraju jediničnoj i Paulijevim matricama (I, X, Y, Z) te njihovim direktnim produktima. Tako imamo:

$$\hat{H} = \sum_{k=0}^{4^n-1} w_k \hat{P}_k, \tag{6.5}$$

gdje je n broj q-bita, a w_k neka težina. Nakon ove dekompozicije *Estimator* stvara novi kvantni krug $V_k |\psi\rangle$ kojim dijagonalizira Paulijeve opservable $\hat{P}_k$ i mjeri ih. Za svaku opservablu mjeri više puta i računa vjerojatnost p_{kj} za svaki izlaz j. Uzevši svojstvenu vrijednost λ_{kj} i težinu w_k računa očekivanu vrijednost po formuli:

$$\langle \hat{H} \rangle_\psi = \sum_{k=0}^{4^n-1} w_k \sum_{k=0}^{2^n-1} p_{kj} \lambda_{kj}. \tag{6.6}$$

Ovaj algoritam je efektivan jedino ako je značajan broj w_k jednak nuli. Broj w_k koji nije nula ne smije rasti brže od polinomijalnog rasta s obzirom na broj q-bita. Kada *Estimator* dobije očekivanu vrijednost šalje ju optimizatoru kojemu je cilj pronaći njezin minimum.
EstimatorQNN osim opcionalne opservable i objekta tipa *Estimator* mora primiti parametrizirani kvantni krug, ulazne podatke i težine. Ona nam služi za prosljeđivanje unaprijed podataka koje primi u konstruktoru preko primljenog kruga i objekta tipa *Estimator*. *NeuralNetworkClassifier* u konstruktoru prima neku instancu klasa za implementaciju neuronskih mreža, optimizator i povratnu funkciju (eng. *"callback function"*). Objekt ove klase provodi treniranje. Optimizator je algoritam koji služi za minimizaciju funkcije cijene, čije vrijednosti dobije od *EstimatorQNN* koju također šaljemo kao argument. U našem kodu koristimo optimizator *COBYLA*, koji funkcionira tako da radi linearne aproksimacije funkcije koju želimo minimizirati, a koju on naziva *"objective function"* [34]. On mora kao parametar primiti maksimalan broj evaluacija funkcije. Povratnu funkciju definira korisnik, a preko nje može doći do podataka unutar procesa treniranja kao što su težine, vrijednosti funkcije koju minimiziramo itd... Mi smo ju definirali tako da dohvaća vrijednosti funkcije i crta ih na graf u ovisnosti o broju iteracije. Kod korišten prilikom inicijalizacije potrebnih objekata i treniranja mreže nalazi se na Slikama 20 - 23.

```
# callback function that draws a live plot when the .fit() method is called
def callback_graph(weights, obj_func_eval):
    clear_output(wait=True)
    objective_func_vals.append(obj_func_eval)
    plt.title("Objective function value against iteration")
    plt.xlabel("Iteration")
    plt.ylabel("Objective function value")
    plt.plot(range(len(objective_func_vals)), objective_func_vals)
    plt.show()
```

Slika 20: *Vrijednosti funkcije koju želimo minimizirati u ovisnosti o broju iteracije.*

```
estimator_qnn = EstimatorQNN(
    circuit=qc, input_params=feature_map.parameters, weight_params=ansatz.parameters
)
estimator_qnn.forward(X[0, :], algorithm_globals.random.random(estimator_qnn.num_weights))
```

Slika 21: *Inicijalizacija objekta tipa EstimatorQNN.*

```
estimator_classifier = NeuralNetworkClassifier(
    estimator_qnn, optimizer=COBYLA(maxiter=60), callback=callback_graph
)
```

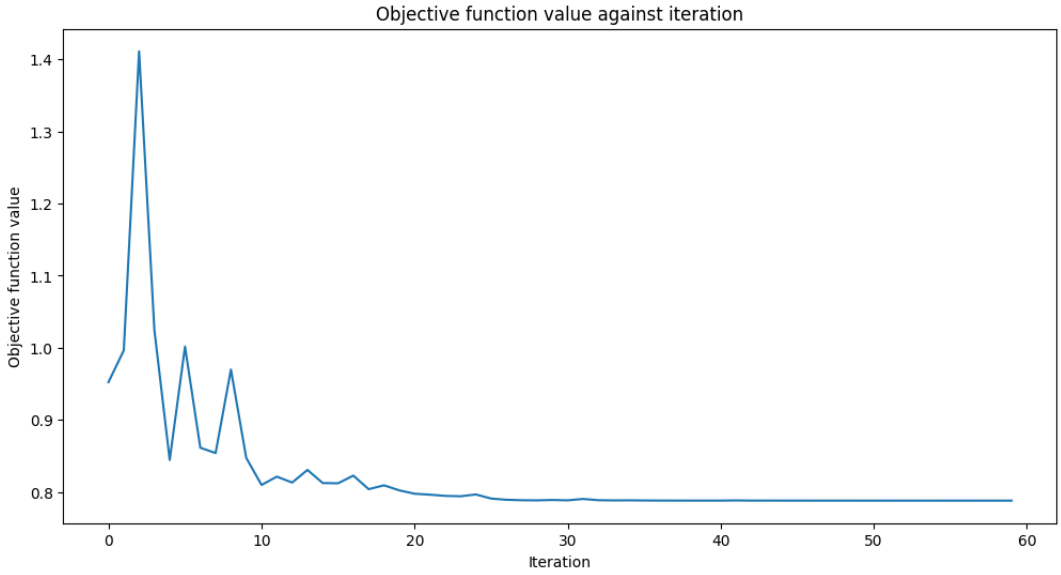
Slika 22: *Inicijalizacija objekta tipa NeuralNetworskClassifier.*

```
objective_func_vals = []
plt.rcParams["figure.figsize"] = (12, 6)

# fit classifier to data
estimator_classifier.fit(X, y)

# return to default figsize
plt.rcParams["figure.figsize"] = (6, 4)

# score classifier
estimator_classifier.score(X, y)
```

Slika 23: *Kod korišten za treniranje mreže.*

6.3 Rezultati

U prethodnoj cjelini opisali smo kako u Qiskit-u izgraditi hibridnu neuronsku mrežu, sada ćemo opisati dobivene rezultate. Graf koji daje povratna funkcija nalazi se na Slici 24. Točnost

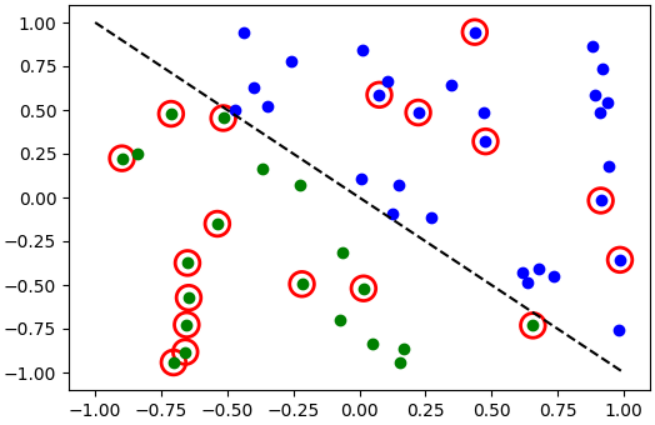


Slika 24: *Graf ovisnosti vrijednosti koju minimiziramo o broju iteracije. Slika izrađena u IBM Quantum Learning-u u Jupyter bilježnici.*

provjerava pozivajući metodu *score* klase *NeuralNetworkClassifier* te ona iznosi 0.64, odnosno 64%. Točnost mreže možemo provjeriti i na još dva načina, predviđanjem na skupu za testiranje i matricom konfuzije. Kod za provedbu predviđanja na skupu za testiranje nalazi se na Slici 25, a graf na kojem su zaokružene pogrešno klasificirane točke nalazi se na Slici 26. Matrica konfuzije matrica je još jedna od mjera točnosti u strojnom učenju. Ona je kvadratna i za problem s n klasa red joj je (n,n). Objasnit ćemo princip na primjeru s dvije klase. Označimo klase s i i j i matricu konfuzije sa M. Na dijagonalnim elementima matrice M_{ii} i M_{jj} će biti broj točno predviđenih elemenata klase i, odnosno j, a van dijagonale ćemo imati na mijestu M_{ij} broj elemenata kojima je predviđena klasa i, a pripadaju klasi j, a na mjestu M_{ji} broj elemenata kojima je predviđena klasa j, a pripadaju klasi i. Za njenu izradu u kodu možemo koristiti već gotove klase i metode iz biblioteka *sklearn.metrics* i *seaborn*. Kod se

```
y_predict = estimator_classifier.predict(X)

# plot results
# red == wrongly classified
for x, y_target, y_p in zip(X, y, y_predict):
    if y_target == 1:
        plt.plot(x[0], x[1], "bo")
    else:
        plt.plot(x[0], x[1], "go")
    if y_target != y_p:
        plt.scatter(x[0], x[1], s=200, facecolors="none", edgecolors="r", linewidths=2)
plt.plot([-1, 1], [1, -1], "--", color="black")
plt.show()
```

Slika 25: *Kod korišten za predviđanje na skupu za testiranje.*



Slika 26: *Grafički prikaz skupa podataka za testiranje. Točke koje su pogrešno predviđene zaokružene su crvenom linijom. Slika izrađena u IBM Quantum Learning-u u Jupyter bilježnici.*

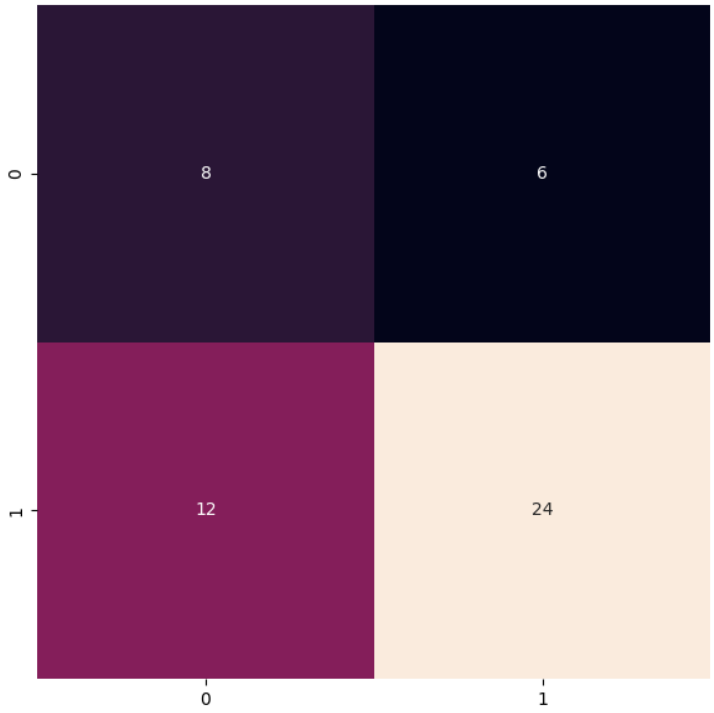
nalazi na Slici 27, a dobivena matrica na Slici 28. Ne možemo biti sigurni da su parametri s

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

y_t = []

for x, y_target, y_p in zip(X, y, y_predict):
    y_t.append(y_target)

plt.figure(figsize=(7,7))
mat = confusion_matrix(y_t, y_predict)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
plt.savefig("matrica konfuzije")
plt.xlabel('točno')
plt.ylabel('predviđeno');
```

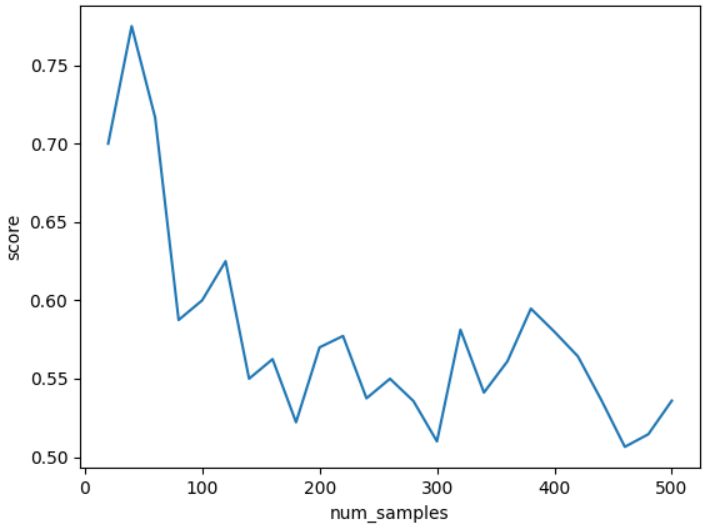
Slika 27: *Kod korišten za izradu matrice konfuzije.*

kojima smo radili do sada optimalni pa smo istražili kako točnost modela ovisi o raznim parametrima. Prvo smo gledali ovisnost o broju podataka. Grafički prikaz ovisnosti točnosti o broju podataka nalazi se na Slici 29. Suprotno očekivanjima dobili smo pad točnosti s povećanjem broja točaka. Najbolje rezultate dobili smo za 40 točaka i ona iznosi 75%.

Drugi parametar koji smo mijenjali je broj ponavljanja rotacija u *ZZFeatureMap*. Kako smo



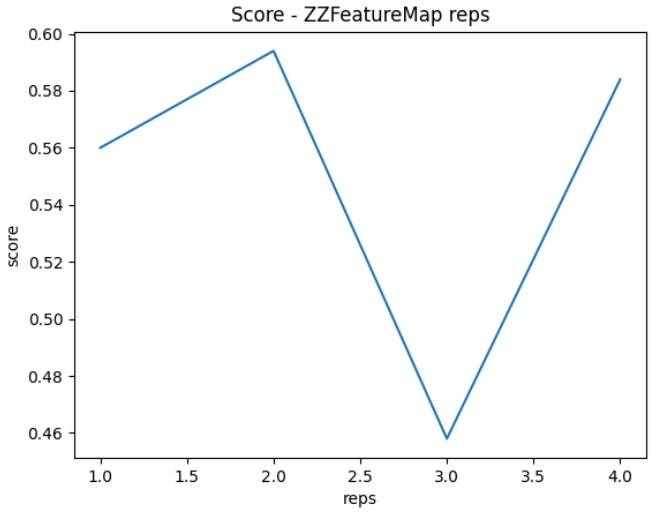
Slika 28: *Grafički prikaz matrice konfuzije. Vidimo da je predviđanje bilo uspješnije za točke koje su iznad pravca. Slika izrađena u IBM Quantum Learning-u u Jupyter bilježnici.*



Slika 29: *Grafički prikaz ovsnosti točnosti o broju podataka za testiranje. Slika izrađena u IBM Quantum Learning-u u Jupyter bilježnici.*

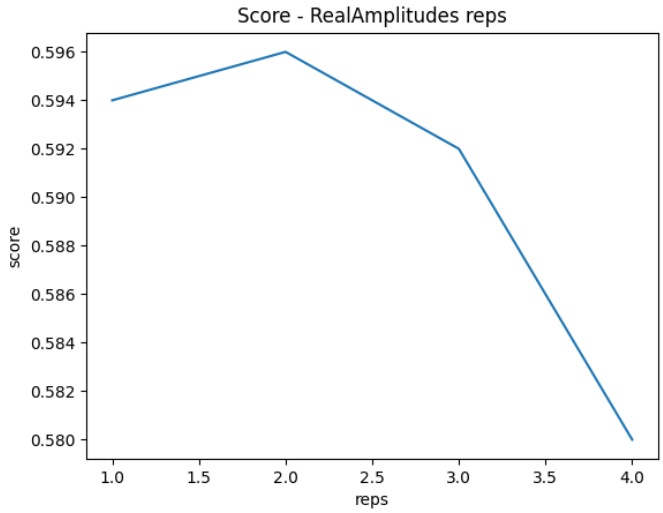
kod provjere ovisnosti točnosti o broju podataka dobili najveću točnost za skup podataka od 40 točaka, koristili smo taj skup za testiranje i ovdje. *RealAmplitudes* smo napravili s jednim

ponavljanjem rotacija. Radili smo provjeru točnosti za 1 do 4 ponavljanja rotacija u *ZZFeatureMap*. Graf ovisnosti nalazi se na Slici 30. Najveća točnost dobivena je za 2 ponavljanja i ona iznosi 59%.



Slika 30: *Grafički prikaz ovsnosti točnosti o broju rotacija u ZZFeatureMap. Korišten je skup podataka za testiranje od 40 točaka, a u RealAmplitudes je jedno ponavljanje rotacija. Slika izrađena u IBM Quantum Learning-u u Jupyter bilježnici.*

Provjeru smo napravili još za broj ponavljanja rotacija u *RealAmplitudes*, također za 1-4 ponavljanja. Korišten je skup podataka za testiranje od 40 podataka, a kako smo dobili najveću točnost za 2 ponavljanja rotacija u *ZZFeatureMap*, koristili smo 2 ponavljanja i ovdje. Dobiveni graf nalazi se na Slici 31. Najveća dobivena točnost je opet za 2 ponavljanja, a iznosi također 59%.



Slika 31: *Grafički prikaz ovsnosti točnosti o broju rotacija u RealAmplitudes. Korišten je skup podataka za testiranje od 40 podataka i 2 ponavljanja rotacija u ZZFeatureMap. Slika izrađena u IBM Quantum Learning-u u Jupyter bilježnici.*

7 Zaključak

U ovom radu uveli smo matematički model disipativne kvantne neuronske mreže i demonstrirali smo implementaciju hibridne kvantno-klasične neuronske mreže u Qiskit-u na vrlo jednostavnom skupu podataka. Maksimalna točnost koju smo dobili iznosi 75%. Kada bismo isti skup podataka ubacili u klasičnu neuronsku mrežu, dobivena točnost bila bi preko 90% i algoritam bi se izvršio u znatno kraćem vremenu. Točnost ovisi o nizu parametara, od kojih smo ovdje testirali samo nekolicinu. Moglo bi se još testirati i neke druge optimizatore, kao i ovisnost o broju mjerenja opservabli (shots). Postav koji smo ovdje testirali nije imao mogućnost kontrole broja mjerenja, no navedeno bi se moglo postići pristupom drugim kvantnim simulatorima u oblaku, što je izašlo iz opsega ovog rada. Uz sve navedeno, vjerojatno bismo mogli postići i veću točnost. Naši, kao i rezultati iz literature [23] ukazuju na to da se ne može reći da trenutne mogućnosti izvedbe hibridnih neuronskih mreža nadmašuju mogućnosti klasičnih arhitektura.

Moramo uzeti u obzir da se algoritam izvršavao u simulatoru, a ne na stvarnom kvantnom hardveru, što sigurno utječe na stvarne mogućnosti kvantnih neuronskih mreža. S razvojem tehnologije, odnosno dostupnosti kvantnih računala s većim brojem q-bita, a posebno kodova za ispravljanje grešaka ovaj problem će postati zanemariv.

Kvantno strojno učenje je aktivno područje istraživanja. Jednostavne algoritme moguće je razvijati i testirati s malim brojem podataka na osobnim računalima, što omogućava velikom broju istraživača da se aktivno uključi u razvoj algoritama. Uzevši to u obzir, možemo reći da će stvarne mogućnosti biti vidljive tek za nekoliko godina.

8 Literatura

[1] J. D. Hidary: *Quantum Computing: An Applied Aproach*, Springer Nature Switzerland, Palo Alto, 2021.

[2] E. R. Johnston, N. Harrigan i M. Gimeno-Segovia: *Programming Quantum Computers: Essential Algorithms and Code Samples*, O'Reilley Media, grad?, 2019.

[3] M. A. Nielsen i I. L. Chuang: *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, 2010.

[4] N. D. Mermin: *Quantum Computer Science: An Introduction*, Cambridge University Press, New York, 2007.

[5] R. Loredo: *Learn Quantum Computing with Python and IBM Quantum Experience: A hands-on introduction to quantum computing and writing your own quantum programs with Python*, Packt Publishing, Birmingham, 2020.

[6] M. Kazalicki: *Kvatntno računanje*, interna skripta Sveučilišta u Zagrebu

[7] W. Kasirajan: *Fundamentals of Quantum Computing: Theory and Practice*, Springer Nature Switzerland, Colorado, 2021.

[8] Y. Du, M.-H. Hsieh, T. Liu, D. Tao: *Expressive power of parametrized quantum circuits*, Sydney, 2020.

[9] N. U. Olle: *A mathematical framework for quantum neural networks*, Barcelona, 2023.

[10] K. Beer, D. List, G. Muller, T. J. Osborne, C. Struckmann: *Training Quantum Neural Networks on NISQ Devices*, Hannover, 2021.

[11] S. Dumančić: *Neuronske mreže*, Osijek, 2014.

[12] M. Kalinić: *Matematičke osnove neuronskih mreža*, Split, 2017.

[13] I. Goodfellow, Y. Bengio, A. Courville: *Deep learning*, MIT Press, 2016. http://www.deeplearningbook.org

[14] N. Innan, M. Al-Zafar Khan, M. Bennai: *Financial Fraud Detection: A Comparative Study of Quantum Machine Learning Models*, 2023.

[15] M.A. Nielsen, I. Chuang: *Quantum computation and quantum information*, 2002.

[16] K. Beer: *Quantum neural networks*, Hannover, 2002.

[17] R. LaRose: *Quantum States and Partial Trace*

[18] J. Hammoud: *Kvantni aspekti crnih rupa*, Zagreb, 2020.

[19] R. P. Frynman: *Statistical Mechanics: A Set of Lectures*, California, 1998.

[20] Y.-C. Liang, Y.-H. Yeh, P. E. M. F. Mendonca, R. Y. Teh, M. D. Reid, P. D. Drummond: *Quantum fidelity measures for mixed states*, 2018.

[21] K. Beer, D. Bondarenko, T. Farrelly, T. J. Osborne, R. Salzmann, D. Scheiermann, R. Wolf: *Training deep quantum neural networks*, 2020.

[22] U. Priyam: *Quantum Neural Networks: A History of QNNs and Google's Quantum AI*, 2020. https://www.linkedin.com/pulse/quantum-neural-networks-history-qnns-googles-ai-utkarsh-priyam

[23] S. Raubitzek, K. Mallinger: *On the Applicability of Quantum Machine Learning*, Beč, 2023.

[24] F. Shah: *Quantum Encoding: An Overview* https://quantumzeitgeist.com/quantum-encoding-an-overview/ [citirano 6.9.2023.]

[25] B. Roy: *All about Data Encoding for Quantum Machine Learning* https://medium.datadriveninvestor.com/all-about-data-encoding-for-quantum-machine-learning-2a7344b1dfef [citirano 6.9.2023.]

[26] IBM Quantum Learning: *Variational Algorithms* https://learning.quantum-computing.ibm.com/course/variational-algorithm-design/variational-algorithms#simplified-hybrid-workflow [citirano 22.9.2023.]

[27] IBM Quantum Learning: *Ansatze and Variational Forms* https://learning.quantum-computing.ibm.com/course/variational-algorithm-design/ansatze-and-variational-forms#variational-form-and-ansatz [citirano 23.9.2023.]

[28] IBM Quantum Learning: *Cost Functions* https://learning.quantum-computing.ibm.com/course/variational-algorithm-design/cost-functions [citirano 23.9.2023.]

[29] Qiskit: *Quantum Feature Maps and Kernels* https://learn.qiskit.org/course/machine-learning/quantum-feature-maps-kernels [citirano 14.9.2023.]

[30] Qiskit: *Parameterized quantum circuits* https://learn.qiskit.org/course/machine-learning/parameterized-quantum-circuits [citirano 3.9.2023.]

[31] Qiskit: *Quantum Neural Networks* https://qiskit.org/ecosystem/machine-learning/tutorials/01_neural_networks.html [citirano 19.9.2023.]

[32] Qiskit: *Neural Network Classifier & Regressor* https://qiskit.org/ecosystem/machine-learning/tutorials/02_neural_network_classifier_and_regressor.html [citirano 22.9.2023.]

[33] Qiskit: *Data encoding* https://learn.qiskit.org/course/machine-learning/data-encoding [citirano 7.9.2023.]

[34] Scipy: *scipy.optimize.minimize* https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html [citirano 23.9.2023.]

[35] Quantum: *Encoding Classical Data into Quantum States* https://qml.baidu.com/tutorials/machine-learning/encoding-classical-data-into-quantum-states.html [citirano 7.9.2023.]

[36] IBM Newsroom: *IBM Unveils Breakthrough 127-Qubit Quantum Processor* https://newsroom.ibm.com/2021-11-16-IBM-Unveils-Breakthrough-127-Qubit-Quantum-Processor [citirano 22.9.2023.]