



**Politechnika
Śląska**

PRACA MAGISTERSKA

Sekwencje kwaternionowe w głębokim uczeniu

Mateusz Płonka
Nr albumu: 282737

Kierunek: Informatyka
Specjalność: Interaktywna grafika trójwymiarowa

PROWADZĄCY PRACĘ
Dr inż. Damian Pęszor

KATEDRA Grafiki, Wizji Komputerowej i Systemów Cyfrowych
Wydział Automatyki, Elektroniki i Informatyki

Gliwice 2023

Tytuł pracy

Sekwencje kwaternionowe w głębokim uczeniu

Streszczenie

Celem pracy jest weryfikacja hipotezy, w myśl której sieci neuronowe są zdolne do pracy z danymi kwaternionowymi. Główną problematyką tego zagadnienia była kwestia braku powszechnych, komercyjnych rozwiązań, stosujących sztuczną inteligencję bazującą na danych kwaternionowych. Decydujący wpływ na tę sytuację ma innowacyjność rozwiązania, którego popularność w środowiskach naukowych wzrosła dopiero na początku XXI w. Badania nad systemami integrującymi sieci neuronowe z danymi kwaternionowymi są istotne dla dalszego rozwoju dziedziny uczenia maszynowego, umożliwiając wyłonienie aspektów, w których zastosowanie kwaternionów ma wpływ na efektywność oraz precyzyjność sieci. Na potrzeby badań opracowano model kwaternionowej sieci rekurencyjnej QLSTM, którego wszystkie elementy są w postaci danych kwaternionowych, a kluczowe procesy uczenia maszynowego zostały rozbudowane o algebrę kwaternionów. Zaimplementowano również autorską funkcję strat, określającą błąd na podstawie kąta zawartego pomiędzy kwaternionem wynikowym, a oczekiwany. Badania przeprowadzono na zbiorach treningowych będących sekwencjami kwaternionowymi opisującymi rotacje stawów w czasie. Przeprowadzone eksperymenty skupiały się na zestawieniu wyników sieci wyposażonych w algebrę Hamiltona, z podstawowymi sieciami rekurencyjnymi w problemie regresyjnym. Owe zagadnienie polega na przewidywaniu dalszego postępu rotacji na podstawie wprowadzonej sekwencji kwaternionów. Wnioski z badań określają przewagę sieci QLSTM w kontekście pracy z danymi kwaternionowymi, podkreślając również problemy z nią związane.

Słowa kluczowe

głębokie uczenie, sieci rekurencyjne, kwaterniony, sekwencje kwaternionowe, systemy przechwytywania ruchu

Thesis title

Quaternion sequences in deep learning

Abstract

The purpose of this paper is to verify the hypothesis that neural networks are capable of working with quaternion data. The main problem of this issue was the absence of common commercial solutions using artificial intelligence based on quaternion data. The decisive factor in this situation is the innovative nature of the solution, the popularity of which in scientific circles has only increased in the early 21st century. Research on systems integrating neural networks with quaternion data is important for the further development of the field of machine learning, making it possible to identify aspects in which the use of quaternions has an impact on the efficiency and precision of the network. For the purposes of the research, a model of the quaternion recurrent network QLSTM was developed, all of whose elements are in the form of quaternion data, and the key processes of machine learning were extended by the algebra of quaternions. A self-developed loss function was also implemented, determining the error based on the angle included between the resulting quaternion and the expected quaternion. The research was conducted on training sets that are quaternion sequences describing joint rotations over time. The experiments focused on comparing the results of networks equipped with Hamilton algebra, with basic recurrent networks in a regression problem. This problem involves predicting the further progress of rotation based on the input sequence of quaternions. The conclusions of the study define the advantage of the QLSTM network in the context of working with quaternion data, also highlighting the problems associated with it.

Key words

deep learning, recurrent networks, quaternions, quaternion sequences, motion capture

Spis treści

1 Wstęp	1
1.1 Kwanterniony jako reprezentacja rotacji	1
1.2 Wprowadzenie do sekwencji kwaternionów	2
1.3 Wykorzystanie danych kwaternionowych w głębokim uczeniu	2
1.4 Cel pracy	2
1.5 Zakres pracy	3
1.6 Charakterystyka rozdziałów	3
2 Analiza tematu	5
2.1 Wprowadzenie do dziedziny	5
2.2 Wprowadzenie do sztucznej inteligencji	6
2.2.1 Historia sztucznej inteligencji	6
2.2.2 Wstęp do uczenia maszynowego	7
2.2.3 Wstęp do sieci neuronowych	10
2.2.4 Klasy sieci neuronowych	12
2.3 Wprowadzenie do kwaternionów	18
2.3.1 Historia kwaternionów	18
2.3.2 Wstęp do kwaternionów	20
2.3.3 Analiza sekwencji kwaternionowych	25
2.4 Przegląd literatury	27
2.4.1 Quaternion convolutional neural network for color image classification and forensics	27
2.4.2 Quaternion recurrent neural networks	28
2.4.3 Praca doktorancka - Quaternion neural networks	30
2.5 Przegląd istniejących rozwiązań	34
2.5.1 Systemy rozpoznawania mowy	34
2.5.2 Inspekcja Wizualna w Produkcji Elektroniki	35
2.5.3 Analiza i przetwarzanie obrazów	36
2.6 Wnioski z analizy tematu	37

3 Przedmiot badań	39
3.1 Opis wykorzystanych narzędzi	39
3.1.1 Python	39
3.1.2 Pytorch	40
3.1.3 Quaternion Neural Networks Plugin	40
3.1.4 CUDA	40
3.1.5 Tensorboard	40
3.1.6 Autorskie rozwiązania	41
3.2 Opis wykorzystanych sieci neuronowych	42
3.2.1 Struktura modelu LSTM	43
3.2.2 Struktura modelu QLSTM	43
3.2.3 Funkcje straty	44
3.2.4 Parametry treningu	46
4 Badania	47
4.1 Opis stanowiska badawczego	47
4.1.1 Wykorzystane narzędzia i motywacja ich wyboru	47
4.1.2 Wybór i uzasadnienie parametrów modeli	48
4.2 Dane	48
4.3 Rygor badawczy	50
4.3.1 Cel badań	50
4.3.2 Przebieg procesu badawczego	53
4.4 Wyniki eksperymentów	54
4.4.1 Biodro LSTM MSE	54
4.4.2 Biodro LSTM QALE	55
4.4.3 Biodro QLSTM MSE	56
4.4.4 Biodro QLSTM QALE	57
4.4.5 Stopa LSTM MSE	58
4.4.6 Stopa LSTM QALE	59
4.4.7 Stopa QLSTM MSE	60
4.4.8 Stopa QLSTM QALE	61
4.4.9 Szyja LSTM MSE	62
4.4.10 Szyja LSTM QALE	63
4.4.11 Szyja QLSTM MSE	64
4.4.12 Szyja QLSTM QALE	65
4.5 Interpretacja wyników	66
4.5.1 Interpretacja wyników sieci LSTM	67
4.5.2 Interpretacja wyników sieci QLSTM	71
4.6 Wnioski z badań	75

5 Podsumowanie	79
Bibliografia	83
Spis użytych terminów oraz skrótów	87
Lista dodatkowych plików, uzupełniających tekst pracy	95
Spis rysunków	98
Spis tabel	99

Rozdział 1

Wstęp

1.1 Kwateriony jako reprezentacja rotacji

Jednym z powodów rozwoju dziedziny grafiki komputerowej jest potrzeba wiernego odzwierciedlenia rzeczywistego świata. Głównym problemem w tworzeniu realistycznych symulacji jest złożoność środowiska pod względem zjawisk fizycznych oraz jego budowy. W związku z tym wzrasta zapotrzebowanie na narzędzia oraz rozwiązania umożliwiające realistyczne i interaktywne odtworzenie aspektów świata w środowiskach graficznych. Jednym z kluczowych problemów zapewnienia realizmu przedstawianej sceny jest czas reakcji oraz płynność ruchów przedstawionych postaci. Z pozoru banalny problem reprezentacji rotacji w wirtualnym świecie jest tematem badań w środowisku naukowym. Popularność w owym zakresie zyskuje sposób reprezentacji orientacji przy pomocy kwaterionów, które zostały opisane w Podrozdziale 2.3. W odróżnieniu od powszechnego stosowania w tym celu kątów Cardana, użycie kwaterionowej sfery jednostkowej niesie ze sobą liczne korzyści, między innymi: mniejsze zużycie pamięci, szybsze wykonanie operacji oraz zapobieganie występowania zjawiska blokady przegubu Cardana (ang. *gimbal*), która jest powszechnym problemem przy stosowaniu kątów Cardana wyjaśnionym w Sekcji 2.3.2 [27]. Oprócz swojego szerokiego zastosowania w grafice komputerowej, struktury algebraiczne jakimi są kwateriony znajdują również użycie w wielu innych dziedzinach nauki, takich jak fizyka, robotyka czy teoria sterowania.

Kwateriony wykorzystywane w kontekście reprezentowania orientacji obiektu trójwymiarowego określają kąt zawarty względem każdej osi układu współrzędnych [12]. Owa struktura algebraiczna składa się z czterech liczb rzeczywistych, wśród których występuje jedna część realna oraz trzy urojone. W kontekście reprezentowania rotacji, wspomniana część rzeczywista służy do określenia kąta obrotu, a pozostałe części urojone reprezentują oś wokół której następuje rotacja. Pomimo rozbudowanej i skomplikowanej struktury, zastosowanie kwaterionów wpływa znaczco na poprawę wydajności aplikacji [21].

1.2 Wprowadzenie do sekwencji kwaternionów

W oparciu o praktyczne zastosowanie kwaternionów do reprezentowania rotacji, wykorzystywanie pojedynczej wartości nie jest wystarczające gdyż określa ona stan orientacji obiektu trójwymiarowego jedynie w danej chwili czasu. Powyższy przypadek mógłby być wykorzystany w sytuacji określenia stałej orientacji struktur statycznych, przykładowo obrót fasady budynków względem rynku. Częściej jednak wykorzystuje się kwaterniony w celu reprezentowania rotacji w czasie, tworząc w ten sposób animację symulującą ruch. W kontekście tego podejścia wymaganym jest utworzenie sekwencji struktur, reprezentujących stan orientacji obiektu trójwymiarowego w kolejnych chwilach czasu. Tak przygotowane dane są następnie interpretowane jako ruch obrotowy trójwymiarowego elementu poprzez sferyczną interpolację liniową pomiędzy kolejnymi wartościami podanego ciągu.

1.3 Wykorzystanie danych kwaternionowych w głębokim uczeniu

Uwzględniając złożoność struktury kwaternionu oraz informacje o jego zmianie w czasie, zasadnym wydaje się stwierdzenie iż omawiane sekwencje mogą przyczynić się do poprawy efektywności sieci neuronowych głębokiego uczenia, wykonujących operacje na rotacjach. Głębokie uczenie jest dziedziną uczenia maszynowego wykorzystującą algorytmy sztucznej inteligencji (ang. *Artificial Intelligence*, AI) do generowania modeli zbliżonych działaniem do funkcjonowania ludzkiego mózgu. Posiada ona jednak kilka zasadniczych różnic względem podstawowych sieci neuronowych. Dzięki większej liczbie warstw, a co za tym idzie bardziej zaawansowanej strukturze, głębokie sieci neuronowe umożliwiają modelowanie złożonych relacji oraz są w stanie przeprowadzać dokładniejsze przewidywania na danych. Niestety trafność przewidywań związana jest z wymaganiem znacznej ilości zasobów obliczeniowych oraz danych w procesie uczenia. Biorąc pod uwagę powyższą argumentację, istotną wydaje się potrzeba przestudiowania zdolności sieci neuronowych głębokiego uczenia do wykorzystywania danych kwaternionowych.

1.4 Cel pracy

Celem pracy jest zweryfikowanie hipotezy, w myśl której sieci neuronowe są zdolne do pracy z danymi kwaternionowymi. Kluczowym z punktu widzenia przeprowadzanych badań będzie utworzenie sieci neuronowej głębokiego uczenia, wykorzystującej do procesu trenowania obszerny zestaw danych z systemu przechwytywania ruchu. Docelowym wynikiem utworzonej sieci jest przewidywanie dalszego postępu rotacji na podstawie wprowadzonej sekwencji kwaternionowej.

1.5 Zakres pracy

W zakresie pracy zawarte są:

- Zaprojektowanie oraz implementacja sieci neuronowej głębokiego uczenia przyjmującej dane w postaci sekwencji quaternionowych,
- Przygotowanie baz danych treningowych oraz testowych uzyskanych z systemu przechwytywania ruchu (ang. *Motion capture*),
- Analiza otrzymanych wyników z zaimplementowanego systemu.

1.6 Charakterystyka rozdziałów

Praca składa się z pięciu opisanych poniżej rozdziałów:

1. **Wstęp** - wprowadzenie do zagadnień poruszanych w przeprowadzanych badaniach oraz omówienie dziedziny problemu pracy wraz z określeniem jej celu i ram technologicznych. Posiada również zwięzłą charakterystykę rozdziałów oraz zakres pracy.
2. **Analiza tematu** - wprowadzenie do dziedziny tematu. Wprowadzi użytkownika w szczegóły opracowywanego problemu, poprzez zgłębienie jego dziedziny. Zawiera również przegląd literatury naukowej oraz istniejących rozwiązań rynkowych, w konsekwencji prowadząc do wstępnego szkicu modelu badawczego.
3. **Projekt sieci neuronowej głębokiego uczenia** - zawiera szczegółowy opis działania oraz konstrukcji projektu badawczego i bazy danych wejściowych. Zwraca uwagę na techniczną stronę działania sieci oraz określa zasady funkcjonowania składowych elementów.
4. **Badania** - opis badań składających się na analizę danych wyjściowych sieci neuronowej w celu zweryfikowania hipotezy, w myśl której sieci neuronowe są zdolne do pracy z danymi quaternionowymi.
5. **Podsumowanie** - zawiera podsumowanie wykonanych prac oraz syntezę nabytych podczas pracy wniosków. Opisuje możliwe drogi kontynuacji badań oraz możliwości rozwoju projektu badawczego.

Rozdział 2

Analiza tematu

2.1 Wprowadzenie do dziedziny

Niniejsza praca zawiera opis badań przeprowadzonych w celu wykazania możliwości rozwoju zagadnienia interpretacji danych przedstawionych w postaci sekwencji kwaternionowych poprzez zastosowanie metod sztucznej inteligencji. Obydwie wspomniane dziedziny, niewątpliwie przyczyniły się do rozwoju świata naukowego oraz wielu gałęzi przemysłu poprzez usprawnienie i udoskonalenie działania wielu firm, produktów oraz rozwiązań. Potencjalne rozwiązanie oparte na współpracy mechanik związanych z obydwiem, przytoczonymi dziedzinami pozwoliłoby na dalszy rozwój każdej z nich oraz wprowadzenie innowacji w branży przemysłowej oraz rozrywkowej.

Sztuczna inteligencja, będąca jedną z omawianych dziedzin jest dyscypliną umożliwiającą tworzenie maszyn i programów realizujących symulacje wybranych funkcji ludzkiego mózgu oraz rozwiązywanie problemów, których implementacja nie jest możliwa w sposób heurystyczny. Możliwości, które niesie ze sobą AI obejmują takie zagadnienia jak selekcja i ekstrakcja danych, rozpoznawanie oraz analiza mowy, wizji czy pisma bądź symulacja środowisk występujących w świecie rzeczywistym [10]. Zważywszy na powyższe argumenty, słuszny wydaje się stwierdzenie iż sztuczna inteligencja jest powszechnym aspektem codziennego życia, przyczyniając się przy tym do automatyzacji wielu złożonych problemów.

Kwaterniony, które są kolejną przytoczoną dziedziną, są strukturami algebraicznymi rozwijającymi zagadnienie liczb zespolonych, które zostały stworzone do opisu mechaniki w przestrzeni trójwymiarowej. Pomimo swojej złożonej natury, odgrywają one istotną rolę w świecie nauki oraz przemysłu, głównie ze względu na ich użyteczność w opisywaniu orientacji w przestrzeni. Przede wszystkim ich zastosowanie odznacza się w dziedzinach takich jak grafika komputerowa, wizja komputerowa, robotyka, fizyka, bioinformatyka czy mechanika orbitalna [17, 27].

2.2 Wprowadzenie do sztucznej inteligencji

2.2.1 Historia sztucznej inteligencji

Dynamiczne tempo postępu technologicznego w XX wieku pozwoliło na rozwój technologii zdolnej do dokonywania z pozoru świadomych wyborów. Definiującym punktem owej ery było sformułowanie nieznanego wcześniej modelu sieci neuronowej oraz mechanicznego ekwiwalentu ludzkiego mózgu, które jest owocem pracy dwóch badaczy, Warrena McCullocha i Waltera Pitta [2]. Ich innowacyjne podejście, angażujące algorytmy i teorie matematyczne, umożliwiło zrozumienie działania mózgu jako procesora interpretującego i analizującego dane. Wymienieni matematycy dowiedli, łącząc neurony w sieciowe struktury, że omawiany model ma zdolność przeprowadzania dowolnej operacji logicznej lub matematycznej. W niedługim czasie po tym osiągnięciu, Alan Turing, bazując na teoriach McCullocha i Pitta, przewidywał potencjał do stworzenia maszyny posiadającej formę inteligencji. Turing zasugerował również eksperyment, w którym opierając się na odpowiedziach na pytania stawiane przez badacza, miał on zdecydować, czy są one produktem działania maszyny, czy człowieka. Obecnie jest on znany jako *Test Turinga* stanowiąc jedno z fundamentalnych narzędzi badań oraz analizy konceptów sztucznej inteligencji.

Koncept przewyższenia ludzkiego umysłu przez AI, zainspirował badaczy do podjęcia próby określenia tego przełomowego momentu. Za najprostszy wyznacznik można przyjąć pokonanie mistrza w wybranej grze logicznej podczas oficjalnego turnieju. Pierwszym takim przełomem było zwycięstwo w warcabach, by następnie wraz z postępem technologii, sprawdzić zdolności AI poprzez szachy, go, pokera, aż do kostki Rubika, który to turniej sztuczna inteligencja zwyciężyła w 2019 roku.

Na przestrzeni lat nastąpił dynamiczny rozwój wielu dziedzin technologicznych związanych ze sztuczną inteligencją. Jednym z pierwszych ogólnodostępnych modeli AI, poprawiających komfort życia codziennych użytkowników internetu były filtry spamu, które zadebiutowały w latach 90-tych. Wykorzystując zbiór maili oznaczonych jako wiadomości mogące być zagrożeniem dla internauty, utworzony został algorytm pozwalający rozpoznać wzorce wcześniej nienapotkanych treści. Był to zdecydowany krok w przód z racji na problem, którego rozwiążanie nie jest możliwe za pomocą reguł logicznych ze względu na jego złożoność oraz ciągłą ewolucję mailowych oszustw [10].

Filtr spamu jest również powszechnym przykładem praktycznego zastosowania dziedziny **eksploracji danych** (ang. *data mining*). W przypadku tego problemu istnieje możliwość wytrenowania jawnego modelu umożliwiającego przystępna analizę algorytmu w celu weryfikacji jego postępów oraz wyciągniętych w procesie nauki wniosków. W przypadku omawianego filtra spamu, możliwość analizy wytrenowanych słów oraz kombinacji wyrazów uznawanych za najlepsze czynniki prognostyczne, może pozwolić na poznanie nowych, nieznanych dotąd korelacji oraz umożliwić lepsze zrozumienie problemu.

W obecnym świecie sektor uczenia maszynowego jest jednym z najprężniej rozwijających się gałęzi sztucznej inteligencji. Odgrywa on znaczącą rolę w systemach rozpoznawania obrazów, intelligentnych chatbotach takich jak asystenci Siri czy Alexa oraz sektorze przetwarzania języka naturalnego, który przyczynił się do znacznej poprawy jakości życia osób z niepełnosprawnościami czy wadami wzroku lub słuchu. Ciągła potrzeba usprawnienia i automatyzacji czasochłonnych procesów oraz perspektywa możliwości, które owy dział nauki może przynieść w przyszłości sprawia iż rozwój sztucznej inteligencji osiągnął globalną, niespotykaną dotąd skalę. Udowadnia ten fakt to iż liczba opublikowanych artykułów na temat AI w samym 2021 roku przekroczyła łączną sumę publikacji w przedziale od 1980 do 2014 roku [6].

2.2.2 Wstęp do uczenia maszynowego

Uczenie Maszynowe jest praktycznym zastosowaniem systemów adaptacyjnych, charakteryzującym się zdolnością do przetwarzania informacji oraz podejmowaniem decyzji na podstawie zdobytych wcześniej doświadczeń. Celem tej dziedziny jest badanie, tworzenie i ciągłe usprawnianie modeli matematycznych wykorzystując trening bazujący na danych pozyskanych z otoczenia. Utworzony w ten sposób model nie wymaga dostępu do kompletnego zbioru danych, dzięki czemu możliwym jest wyciąganie wniosków oraz podejmowanie racjonalnych decyzji na temat informacji, nieznanych dotąd dla systemu. Technologie wykorzystujące uczenie maszynowe znajdują powszechnie zastosowanie w codziennym życiu w systemach operujących na danych wymagających odpowiedniej obróbki, takich jak filtry spamu czy systemy rozpoznawania twarzy bądź głosu [3].

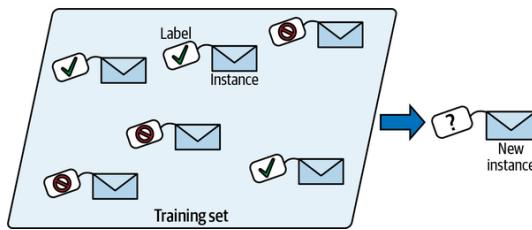
Amerykański informatyk i profesor Tom M. Mitchell opisał uczenie maszynowe w swojej książce, o tytule *Machine Learning*, następującymi słowami:

„Mówimy, że program komputerowy uczy się na podstawie doświadczenia E w odniesieniu do jakiegoś zdarzenia T i pewnej miary wydajności P , jeśli jego wydajność (mierzona przez P) wobec zadania T wzrasta wraz z nabyciem doświadczenia E .“ [20].

Powyzszą definicję wyczerpująco opisuje przykład, gdzie program komputerowy uczy się grać w warcaby. Może on poprawić swoją wydajność **mierzoną jego zdolność do wygrywania** w klasie zadań obejmujących **granie w warcaby**, dzięki doświadczeniu uzyskanemu poprzez **granie w gry przeciwko sobie**. W uproszczeniu, by poprawnie zdefiniować problem uczenia się, należy określić następujące trzy cechy: klasę zadań, miarę wydajności, która ma zostać poprawiona, oraz źródło doświadczenia. Dla omawianego problemu gry w warcaby należy zdefiniować kolejne cechy: gra w warcaby (zadanie T), procent gier wygranych z przeciwnikami (miara wydajności P) oraz rozgrywanie gier treningowych przeciwko sobie (doświadczenie E).

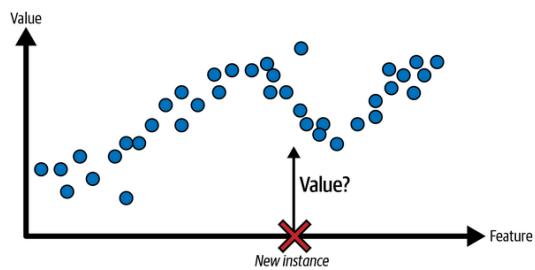
Uczenie nadzorowane - Supervised Learning

W uczeniu nadzorowanym (ang. *supervised learning*) dane uczące przekazywane algorytmowi zawierają dołączone rozwiązania problemu zwane etykietami. Ten rodzaj uczenia cechuje się wykorzystaniem koncepcji systemu nadzorczego, odpowiedzialnego za informowanie o jakości uzyskanych wyników w danym epizodzie treningu, na podstawie porównania wyjścia z etykietą danej na której wykonywana była operacja. Mając wgląd do oczekiwanych wyników, model jest w stanie dostosowywać swoje parametry by zredukować wartość zwracaną przez funkcję straty. Klasycznym zadaniem uczenia nadzorowanego jest **klasyfikacja**. Jest to proces polegający na trenowaniu sieci z wykorzystaniem obszernych zbiorów przykładowych danych należących do jednej klasy. Oczekiwany wynikiem jest model zdolny do rozpoznania nieznanej dotąd danej reprezentującej grupę treningową. Przykładem klasyfikacji jest filtr spamu, w którym model jest trenowany za pomocą dużej liczby wiadomości określonych jako spam, dzięki czemu jest w stanie klasyfikować nowe wiadomości. Trening takiego modelu przedstawiony jest za pomocą Rysunku 2.1.



Rysunek 2.1: Problem klasyfikacji dla filtra spamu [10]

Innym typowym zadaniem dla uczenia nadzorowanego jest problem **regresji**. Polega on na przewidywaniu docelowej wartości numerycznej przy użyciu określonego zbioru cech zwanych **czynnikami prognostycznymi** lub **predykcjami**. Przykładem regresji jest model zwracający cenę samochodu (wartość numeryczną), na podstawie podanych parametrów auta takich jak przebieg, marka czy wiek (predykcje). Przykład relacji wartości do cechy znajduje się na Rysunku 2.2. Niektóre algorytmy regresyjne są powszechnie wykorzystywane w klasyfikowaniu danych i odwrotnie [10]. Jednym z nich jest **regresja logistyczna**, która jest powszechnie stosowana w zadaniach klasyfikacyjnych gdyż umożliwia określenie prawdopodobieństwa przynależności danej grupy cech do określonej klasy.

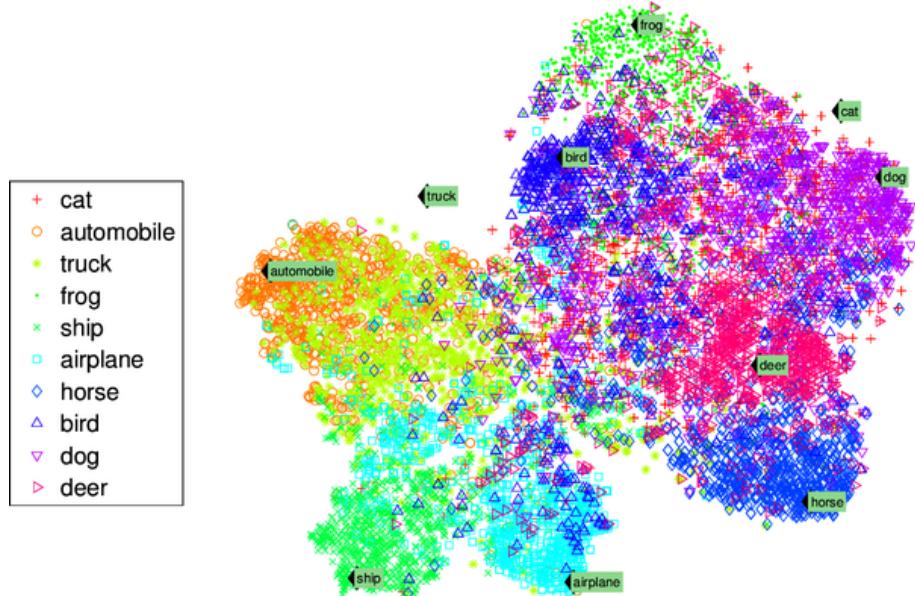


Rysunek 2.2: Problem regresyjny dla przewidywania wartości numerycznej dla cechy. [10]

Jednym z najczęściej spotykanych problemów w uczeniu maszynowym jest zjawisko **nadmiernego dopasowania (ang. Overfitting)**. W tym przypadku model jest zbyt dokładnie dopasowany do danych treningowych, przez co traci zdolność do generalizacji działając prawidłowo jedynie na podanym zbiorze danych. Ograniczenie modelu w celu jego uproszczenia i zmniejszenia ryzyka przetrenowania nosi nazwę **regulacji**.

Uczenie nienadzorowane - Unsupervised Learning

Proces uczenia nienadzorowanego nie zawiera systemu nadzorującego, co przekłada się na brak informacji o poziomie błędu danego epizodu. Uczenie bez nadzoru wykorzystywane jest przede wszystkim, gdy ważne jest segregowanie zbioru elementów według danego schematu. Spośród najważniejszych takich algorytmów wyróżnić można: analizę skupień, algorytmy wizualizujące, wykrywanie anomalii, wizualizację i redukcję wymiarowości oraz uczenie przy użyciu reguł asocjacyjnych. Przykładem problemu jest wykorzystanie algorytmów wizualizujących. Wprowadzony liczny, nieoznaczony zestaw danych jest następnie wyświetlany w postaci punktów na dwuwymiarowym lub trójwymiarowym wykresie. Algorytmy te starają się w maksymalnym stopniu zachować pierwotną strukturę danych rozdzielając poszczególne skupienia w przestrzeni wejściowej, redukując zjawiska nakładania się na siebie poszczególnych klastrów. Umożliwia to łatwiejszą analizę danych i pozwala na odkrycie nieprzewidzianych wzorców. Rysunek 2.3 to przykład wizualizacji.

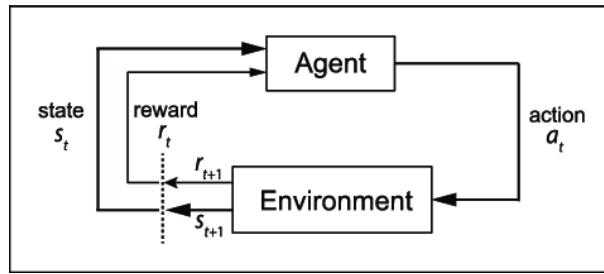


Rysunek 2.3: Przykład wizualizacji t-SNE wskazującej semantyczny podział skupień. [10]

W przypadku, gdy dane wejściowe są częściowo etykietowane, wykorzystywane jest uczenie pół-nadzorowane, które wprowadza drobne ograniczenia algorytmu kategoryzacji. Uczenie nienadzorowane wykorzystywane jest w systemach stosowanych do detekcji podobieństw, podziału obiektów oraz automatycznego etykietowania.

Uczenie ze wzmocnieniem - Reinforcement Learning

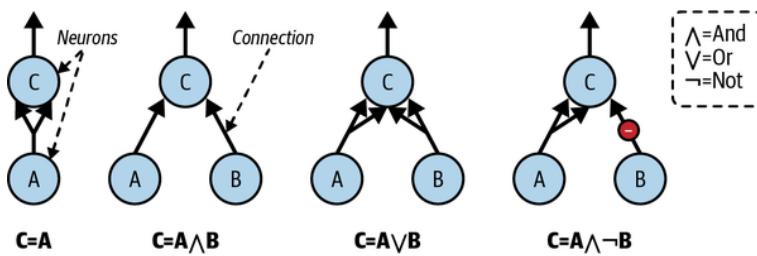
Uczenie ze wzmocnieniem nie posiada systemu nadzoru nad swoim treningiem, ponieważ jego proces opiera się na danych pochodzących z otoczenia. Jego zadaniem jest odkrycie reguły łączącej aktualne obserwacje (bodźce z otoczenia) z reakcją modelu na te bodźce. System uczący się, zwany w tym kontekście agentem, może obserwować środowisko, dobierać i wykonywać czynności, a także odbierać nagrody lub kary będące nagrodami ujemnymi. Rysunek 2.4 przedstawia schemat przebiegu operacji w procesie uczenia ze wzmocnieniem. Z każdą nową iteracją model powinien dążyć do zastosowania optymalnej reguły, aby maksymalizować wartość otrzymanej nagrody. Uczenie ze wzmocnieniem jest przede wszystkim wykorzystywane w sytuacjach o dynamicznym charakterze, gdzie nie da się jednoznacznie określić odchylenia błędu.



Rysunek 2.4: Schemat przebiegu operacji w procesie uczenia ze wzmocnieniem. [8]

2.2.3 Wstęp do sieci neuronowych

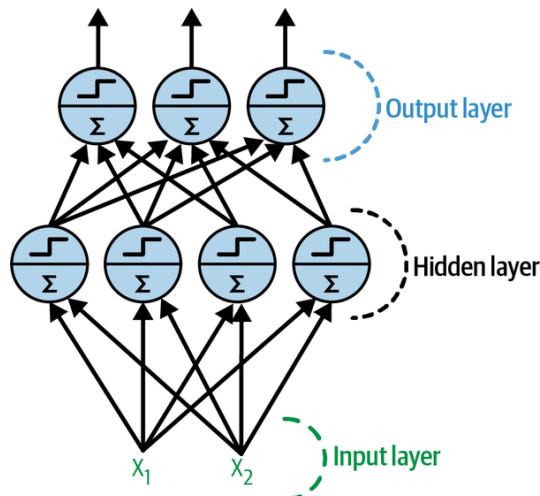
Koncepcja sztucznego neurona zrodziła się z obserwacji funkcjonowania naturalnych komórek nerwowych, zwanych neuronami. Została ona przedstawiona w 1943 roku przez neurofizjologa Warrena McCullocha i matematyka Wiltera Pittsa w epokowym artykule *A Logical Calculus of Ideas Immanent in Nervous Activity* [18]. Zaproponowano wtedy koncept struktury matematycznej mającej co najmniej jedno binarne wejście oraz binarne wyjście. W sztucznym neuronie wyjście jest aktywowane wtedy, gdy jest aktywna określona liczba wejść. Pomimo iż koncept znacznie odbiega od obecnego stanu sztucznych neuronów, autorzy zdołali udowodnić iż przy pomocy sieci przedstawionych modeli istnieje możliwość skonstruowania dowolnej operacji logicznej, co zostało zaprezentowane na Rysunku 2.5.



Rysunek 2.5: Sztuczne sieci neuronowe przeprowadzające proste operacje logiczne. [10]

Kolejnym istotnym krokiem była konstrukcja Perceptronu zaproponowana przez Franka Rosenblatta w 1962 roku [25]. Stanowił on rozszerzenie modelu McCullocha-Pittsa o możliwość uczenia się. Jego podstawą jest zmodyfikowany sztuczny neuron, zwany **progową jednostką logiczną** (ang. *threshold logic unit*, *TLU*), którego wartości wejścia i wyjście są liczbami, w przeciwieństwie do pierwotnego podejścia, w którym są binarne, a każde połączenie ma przyporządkowaną wagę. Jednostka TLU wylicza ważoną sumę sygnałów wejściowych, a następnie wykorzystuje funkcję skokową, najczęściej funkcję Heaviside'a, wobec tej sumy. Perceptron był w stanie samodzielnie dostosowywać wagi połączeń między neuronami na podstawie błędów popełnianych przez model. Mimo początkowych sukcesów, posiadał pewne ograniczenia takie jak problemy z operacjami wymagającymi separacji nieliniowej, jak np. implementacja bramki logicznej XOR (ang. *Exclusive OR*).

W odpowiedzi na ograniczenia, naukowcy wprowadzili koncepcję perceptronu wielowarstwowego (ang. *multilayer perceptron*, *MLP*) [10], który posiadał co najmniej jedną ukrytą warstwę jednostek TLU między warstwą wejściową a wyjściową. W przypadku gdy sztuczna sieć neuronowa posiada wiele warstw ukrytych, nosi nazwę **głębokiej sieci neuronowej**. Prezentacja takiej sieci znajduje się na ilustracji 2.6.



Rysunek 2.6: Architektura perceptronu wielowarstwowego z jedną warstwą ukrytą. [10]

Przez wiele lat naukowcy próbowali bezskutecznie znaleźć sposób uczenia perceptronów wielowarstwowych. Przełom nastąpił w roku 1986 gdy David Rumelhart, Geoffrey Hinton i Ronald Williams opublikowali przełomowy artykuł przedstawiający koncepcję **propagacji wstecznej**[26]. Metoda pozwala na efektywne trenowanie sieci przy użyciu techniki automatycznego obliczania gradientu. Algorytm propagacji wstecznej wykonuje obliczenia błędu sieci w odniesieniu do każdego parametru modelu w zaledwie dwóch przebiegach. Umożliwia ona wówczas modelowanie złożonych wzorców nieliniowych. W odniesieniu do powyższego akapitu, zasadnym wydaje się stwierdzenie iż wynalezienie propagacji wstecznej było przełomowym odkryciem i jest obecnie kluczowym krokiem w procesie trenowania sieci głębokiego uczenia.

Sieci głębokiego uczenia mogą być stosowane zarówno do rozwiązywania **problemów regresyjnych jak i klasyfikacyjnych**. Typ architektury sieci, funkcje aktywacji oraz funkcje straty są wybierane zależnie od specyfiki problemu. W przypadku problemów regresji typowo używa się jednego neuronu wyjściowego bez funkcji aktywacji oraz funkcji straty typu błąd średniokwadratowy (ang. *mean squared error*, MSE). Dla problemów klasyfikacji binarnej używany jest jeden neuron wyjściowy z sigmoidalną funkcją aktywacji i funkcją entropii krzyżowej jako funkcję straty. Natomiast, dla problemów klasyfikacji wieloklasowej, dla każdej klasy przyjmuje się jeden neuron wyjściowy z funkcją aktywacji softmax i ponownie funkcję straty entropii krzyżowej. Należy również zwrócić uwagę na fakt, iż złożona struktura tych modeli umożliwia uczenie ich bez konieczności ręcznego projektowania cech [10].

2.2.4 Klasy sieci neuronowych

Sieci splotowe

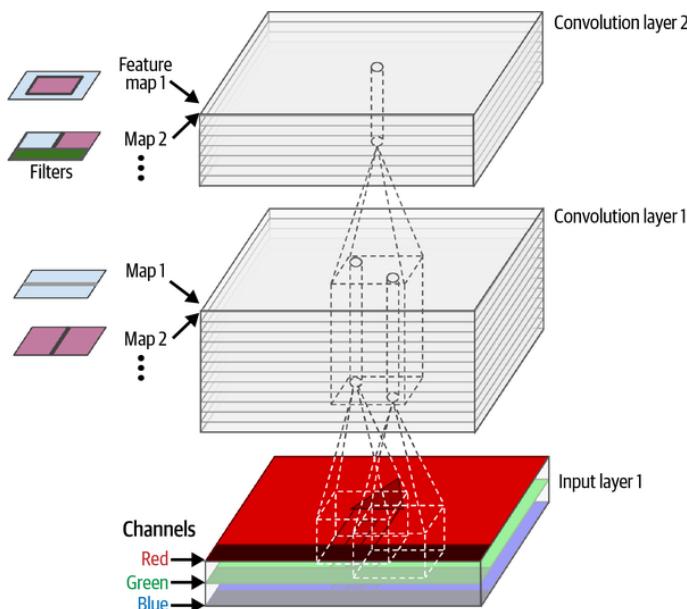
Na przełomie lat 50 i 60 XX wieku, dwóch naukowców, David Hubel i Torsten Wiesel przeprowadzili szereg eksperymentów na kotach oraz małpach badając strukturę ich kory wzrokowej [14]. Udowodnili oni wówczas, że neurony w tym obszarze mózgu stwarzają struktury splotowe, reagujące jedynie na bodźce wzrokowe mieszczące się w określonym polu nazwanym później **lokalnym polem recepcyjnym**. Zauważono również iż pewne neurony reagują wyłącznie na obrazy składające się z linii poziomych a inne z pionowych oraz że różne sploty posiadają różne wielkości pól recepcyjnych, pozwalając im na rozpoznanie bardziej złożonych wzorców. Panowie Hubel oraz Wiesel otrzymali za swoje odkrycie nagrodę nobla w dziedzinie fizjologii lub medycyny. Badania kory wzrokowej stanowiły inspirację do budowy pierwszych sieci splotowych.

Yann LeCun zaprezentował w 1998 roku rewolucyjną sieć **LeNet-5** [16], która była powszechnie używana do rozpoznawania odręcznie pisanych cyfr. Model posiadał znane z sieci neuronowych elementy takie jak połączone warstwy oraz sigmoidalną funkcję aktywacji, jednakże posiadał również nowe konstrukty którymi były warstwy splotowe i warstwy łączące. Wraz z powstaniem tego modelu rozpoczęł się okres rozwoju sieci splotowych (konwolucyjnych) (ang. *convolutional neural networks, CNN*).

Sieci splotowe bazują swoją architekturę na wcześniej opisanym przykładzie części rzeczywistego mózgu odpowiedzialnej za wizję. Najistotniejszym składnikiem sieci CNN jest **warstwa splotowa**. Neurony z pierwszej warstwy splotowej nie są połączone z każdym pikselem obrazu, jak miałyby to miejsce w przypadku implementacji pospolitej, głębszej sieci neuronowej. Komunikują się one jedynie z pikselami znajdującymi się w ich polu recepcyjnym, z kolei każdy neuron w drugiej warstwie splotowej łączy się wyłącznie z neuronami znajdującymi się w niewielkim obszarze pierwszej warstwy. Dzięki takiej konstrukcji sieć ma możliwość koncentrować się na rozpoznawaniu prostych kształtów na

pierwszych poziomach abstrakcji, by z każdą kolejną warstwą znajdować bardziej złożone wzorce. Owa hierarchiczna struktura sprawdza się przy problemach związanych z rozpoznawaniem i klasyfikacją obiektów na zdjęciach, przez to też sieci CNN są głównie wykorzystywane do tych zadań.

Jednym z kluczowych aspektów warstw splotowych jest użycie **filtrów** (ang. *kernels*), które reprezentują wagi neuronów. Każdy filtr odpowiada za wykrywanie określonej cechy w obrazie, na przykład konkretnej krawędzi czy linii. Warstwa składająca się z neuronów używających tego samego filtra generuje **mapę cech** (ang. *feature map*), która określa obszary obrazu, które najbardziej aktywują filtr. W praktyce warstwa splotowa tworzy wiele map cech jednocześnie, stosując liczne filtry, co pozwala na detekcję wielu cech jednocześnie. Schemat takiej sieci splotowej znajduje się na Rysunku 2.7. W efekcie końcowym, CNN są zdolne do uczenia się i rozpoznawania złożonych wzorców w obrazach.



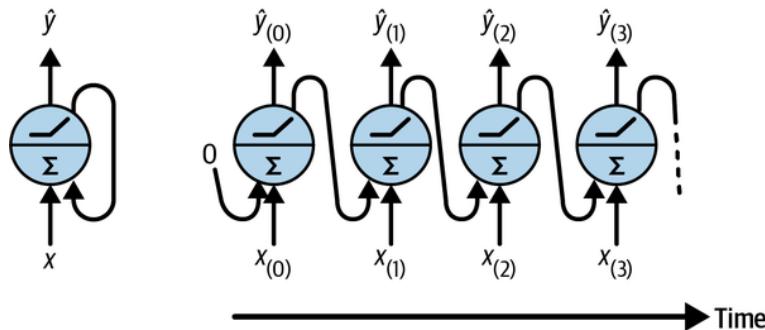
Rysunek 2.7: Warstwy splotowe zawierające wiele map cech, a także zdjęcie z trzema kanałami barw. [10]

Cechą wyróżniającą sieci splotowe jest fakt, iż wszystkie neurony w mapie cech dzielą te same parametry, co pozwala na znaczące ograniczenie liczby parametrów w modelu. Gdy CNN nauczy się rozpoznawać wzorzec w jednym miejscu, będzie w stanie to robić również w innych lokacjach. Dla porównania, generatywne sieci stochastyczne (ang. *Generative Stochastic Network*, GSN) po nauczeniu się rozpoznawania wzorca w jednym miejscu nie potrafiły przełożyć tej informacji na inne obszary. Sieci splotowe stanowią efektywne narzędzie w zadaniach związanych z przetwarzaniem obrazów i innych rodzajów danych, które można reprezentować w formie wielowymiarowych macierzy. Wykorzystanie filtrów oraz tożsama parametryzacja na poziomie map cech umożliwiają CNN wykrywanie i naukę złożonych wzorców przestrzennych w danych, co przekłada się na ich wysoką skuteczność.

Sieci rekurencyjne

Sieci rekurencyjne (ang. *Recurrent Neural Networks, RNN*) są zdolne do analizy danych w formie szeregów czasowych, takich jak liczba aktywnych użytkowników strony internetowej na przestrzeni dnia, godzinowa temperatura w danym mieście, codzienne zużycie energii w domu, trajektorie sąsiednich samochodów i wiele więcej. Po wyuczeniu się określonych wzorców danych, wykorzystują one otrzymane informacje do prognozowania wydarzeń, przy założeniu, że środowisko, w którym nastąpiła predykcja nie uległo zmianom. Sieci RNN umożliwiają pracę na sekwencjach danych, w przeciwieństwie do stałej liczby parametrów w przypadku podstawowych sieci neuronowych. Taka architektura szczególnie sprawdza się w zadaniach związanych z przetwarzaniem tekstu, dźwięku lub sekwencji zmiany danych w czasie. Czyni to sieci RNN przydatnymi w zastosowaniach przetwarzania języka naturalnego, takich jak automatyczne tłumaczenie czy przekształcanie mowy w tekst.

Dotychczas przeprowadzona analiza skupiała się na sieciach neuronowych jednokierunkowych (ang. *Feed-forward*), w których aktywacje przepływają w jednym kierunku, od warstwy wejściowej do warstwy wyjściowej. Rekurencyjna sieć neuronowa bardzo przypomina sieci jednokierunkowe, z tą różnicą iż umożliwia tworzenie połączeń wstecz. Przykład takiego połączenia został zobrazowany na Rysunku 2.8.



Rysunek 2.8: Neuron rekurencyjny rozwijany w czasie [10]

Najprostszą implementację opisywanej sieci można opisać jako pojedynczy neuron, który otrzymuje sygnały wejściowe oraz generuje sygnały wyjściowe, następnie przesyłając je ponownie na swoje wejście. W każdym taktie czasowym t , rekurencyjny neuron odbiera dane wejściowe $x_{(t)}$ oraz swój własny wynik z poprzedniego kroku czasowego, $\hat{y}_{(t-1)}$. Ponieważ w pierwszym kroku czasowym nie występuje poprzedni wynik, wówczas domyślnie podaje się na wejściu 0. Przekazywanie logiki z poprzednich taktów na wejście można reprezentować w formie zmiany stanu określonego neuronu w czasie. Proces ten nazywany jest rozwinięciem sieci w czasie. Należy również zauważyć, że każdy neuron rekurencyjny posiada dwa zestawy wag: jeden dla danych wejściowych $x_{(t)}$ i drugi dla wyników z poprzedniego kroku czasowego, $\hat{y}_{(t-1)}$.

Analogicznie do jednokierunkowych sieci neuronowych, rekurencyjne neurony można łączyć w warstwy. W każdym kroku czasowym t , neurony składające się na daną warstwę, odbierają zarówno wektor wejściowy $x_{(t)}$, jak i wektor wyjściowy z poprzedniego kroku czasowego $\hat{y}_{(t-1)}$. Biorąc pod uwagę całą warstwę rekurencyjną, a nie tylko jeden neuron, możliwym jest umieszczenie wszystkich wektorów wag w dwóch macierzach W_x i W_y . Wówczas wynik warstwy rekurencyjnej dla jednego przypadku można reprezentować następującym Równaniem 2.1.

$$y_{(t)} = \phi(\mathbf{W}_x^T \mathbf{x}_{(t)} + \mathbf{W}_y^T \mathbf{y}_{(t-1)} + \mathbf{b}) \quad (2.1)$$

W tym równaniu:

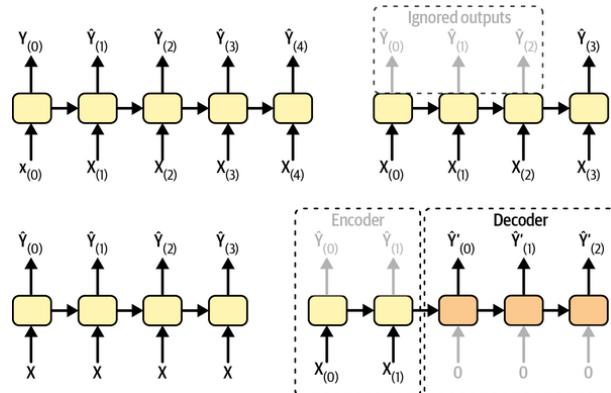
- $Y_{(t)}$ - macierz zawierająca dane wyjściowe warstwy w taktie t dla każdego elementu minigrupy.
- $X_{(t)}$ - macierz zawierająca dane wejściowe dla każdego neuronu.
- W_x - macierz wag połączeń dla wejść w bieżącym taktie.
- W_y - macierz wag połączeń dla wyjść z poprzedniego taktu.
- b - wektor członów obciążenia każdego neuronu.
- ϕ - funkcja aktywacji.

Wyjście neuronu rekurencyjnego w taktie t stanowi funkcję wszystkich danych wejściowych z poprzednich ramek czasowych, dlatego można stwierdzić iż sieci RNN posiadają strukturę na kształt pamięci. Fragment sieci neuronowej przechowujący informacje o stanie poprzednich iteracji nazywany jest **komórką pamięci** (ang. ***memory cell***). Właśnieność neuronów rekurencyjnych sprawia, że są niezbędne w sieciach zdolnych do uczenia się krótkoterminowych wzorców, co jest szczególnie przydatne w zastosowaniach przewidywania szeregów czasowych. Stan komórki w kroku czasowym t , oznaczany jako $h_{(t)}$, jest funkcją wejść w tym kroku czasowym oraz stanu w poprzednim kroku czasowym został opisany za pomocą Równania 2.2.

$$h_{(t)} = f(x_{(t)}, h_{(t-1)}) \quad (2.2)$$

Wyjście w kroku czasowym t , oznaczane jako $\hat{y}_{(t)}$, jest również funkcją poprzedniego stanu oraz bieżących wejść. W przypadku podstawowych komórek, wyjście jest tożsame ze stanem, ale w bardziej zaawansowanych typach komórek nie zawsze ta zasada nie zawsze jest spełniona.

Sieci neuronowe rekurencyjne wykorzystują różne architektury przepływu danych, których schematy zostały zaprezentowane na Rysunku 2.9. Przykładowo, umożliwiają jednocześnie przyjmowanie sekwencji danych na wejściu oraz generowanie za ich pomocą wyników na wyjściu. Jednym z przykładów jest **siec sekwencyjna**, przydatna do prognozowania danych szeregow czasowych, takich jak ceny akcji na giełdzie ze względu na swoją architekturę, w której przyjmuje oraz zwraca pełną sekwencję danych. Istnieją również inne struktury, które pozwalają przyjmować sekwencję wejść i ignorować wszystkie wyjścia z wyjątkiem ostatniego. Są to tak zwane **sieci sekwencyjno-wektorowe**. Przykładem ich wykorzystania jest system przyjmujący recenzję filmu w formie słów na wejściu, zwracając na wyjściu ocenę nacechowania autora do recenzowanego materiału w skali $<-1, 1>$. Istnieją również sieci, które przyjmują ten sam wektor wejściowy na każdym kroku czasowym i zwracają sekwencję, co jest typowe dla **sieci wektorowo-sekwencyjnych**. Wykorzystuje się je w generatorach opisów do obrazu, które umożliwiają kilkukrotne wprowadzenie tego samego źródła, oczekując różnych opisów. Ostatecznie, można uzyskać strukturę **sieci sekwencyjno-wektorowej**, zwanej **koderem**, a następnie sieci **wektorowo-sekwencyjnej**, zwanej **dekoderem**, które w połączeniu sprawdzają się w zadaniach związanych z tłumaczeniem tekstów.

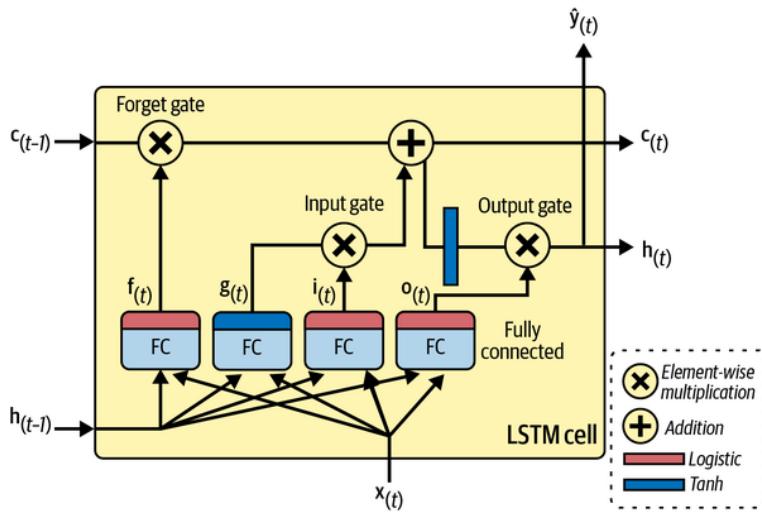


Rysunek 2.9: Architektury sieci RNN: (kolejno) sekwencyjna, sekwencyjno-wektorowa, wektorowo-sekwencyjna, koder-dekoder. [10]

Do wytrenowania sieci rekurencyjnej na długich sekwencjach, należy przetworzyć wiele taktów przez co rozwinięta sieć RNN staje się bardzo głęboka. Podobnie jak w przypadku innych sieci neuronowych jest ona wówczas narażona na **problem niestabilnych gradientów**. Odnosi się on do zjawiska, w którym gradienty funkcji straty w głębokich sieciach neuronowych osiągają ekstremalnie duże lub małe wartości w trakcie procesu uczenia, co prowadzi do trudności w optymalizacji parametrów sieci. Dodatkowo w miarę przetwarzania długich sekwencji danych sieć rekurencyjna, stopniowo zapomina o pierwszych wejściach w sekwencji. Problem ten nazywa się **zanikaniem gradientu**. W obliczu tych trudności, naukowcy zaproponowali różne typy komórek z długotrwałą pamięcią, które mają na celu poprawę wydajności sieci RNN.

Najpopularniejszą komórką z długotrwałą pamięcią jest komórka LSTM (ang. *Long Short-Term Memory*) zaproponowana w 1997 roku przez Seppa Hochreitera i Jürgena Schmidhubera [13]. Mechanizm działania opiera się na zdolności sieci do nauki, które informacje przechowywać w stanie długoterminowym, oraz kiedy należy je odczytać. Głównym elementem, wyróżniającym LSTM, jest podział stanu na dwa wektory: $h_{(t)}$, będący stanem krótkoterminowym, oraz $c_{(t)}$, czyli stan długoterminowy.

Działanie komórki LSTM opiera się na trzech istotnych elementach, zwanych bramkami: bramką zapominania (ang. *forget gate*), bramką wejściową (ang. *input gate*) oraz bramką wyjściową (ang. *output gate*). Kontrolowane są one przez funkcje aktywacji, pozwalając sieci decydować, które informacje powinny być przechowywane, odrzucone oraz odczytywane. Schemat struktury komórki LSTM został przedstawiony na Rysunku 2.10.



Rysunek 2.10: Schemat komórki LSTM [10]

Opis poszczególnych bramek wchodzących w skład komórki LSTM:

- **Bramka zapominania** - sterowana przez $f_{(t)}$, kontroluje, które części stanu długoterminowego powinny zostać usunięte.
 - **Bramka wejściowa** - sterowana przez $i_{(t)}$, decyduje, które części analizy obecnych wejść i poprzedniego stanu krótkoterminowego powinny zostać dodane do stanu długoterminowego.
 - **Bramka wyjściowa** - sterowana przez $o_{(t)}$, kontroluje, które części stanu długoterminowego powinny być odczytane i wyjściowo zwrócone w danym kroku czasowym.

W rezultacie, komórka LSTM jest w stanie rozpoznawać istotne dane na wejściu, przechowywać je w stanie długoterminowym, uczyć się przechowywać je dopóty, dopóki są potrzebne, a także wydobywać je wtedy, gdy nadaje się taka potrzeba. To sprawia, że komórki te są niezwykle skuteczne w przechwytywaniu długoterminowych wzorców w szeregach czasowych, długich tekstach, nagraniach audio i innych sekwencjach danych.

2.3 Wprowadzenie do kwaternionów

2.3.1 Historia kwaternionów

Początek XIX wieku był bardzo eksyktującym okresem dla rozwoju analizy zespolonej. Pomimo tego iż zagadnienie było znane już od XV wieku, badanie jej było często odrzucone, uważając zagadnienie za bezużyteczne. Dopiero koniec XVIII oraz pierwsza połowa XIX wieku były okresem rozkwitu tej dziedziny, głównie za sprawą takich naukowców jak Leonhard Euler, Augustin Louis Cauchy oraz Bernhard Riemann.

Należy również zwrócić uwagę na inną zasłużoną postać tamtych czasów. Sir William Rowan Hamilton, matematyk, astronom, profesor uniwersytetu Trinity Collage w Dublinie. Zasłużył się między innymi badaniami w dziedzinie algebry, rachunku wariacyjnego, mechaniki teoretycznej oraz optyki, jednakże to właśnie odkrycie kwaternionów i późniejsze prace nad nimi były jego głównym wkładem w rozwój nauki.

Przed odkryciem kwaternionów, Hamilton zajmował się liczbami zespolonymi formułując w 1833 roku **Teorię par** (ang. *Theory of Couplets*), która wówczas była uważana za nową algebraiczną reprezentację liczb zespolonych. Przedstawiając dwie liczby rzeczywiste jako parę (a, b) , zdefiniował dla nich operacje dodawania oraz mnożenia, opisanych poprzez Równanie 2.3, tworząc w ten sposób algebraiczną definicję liczb zespolonych [4].

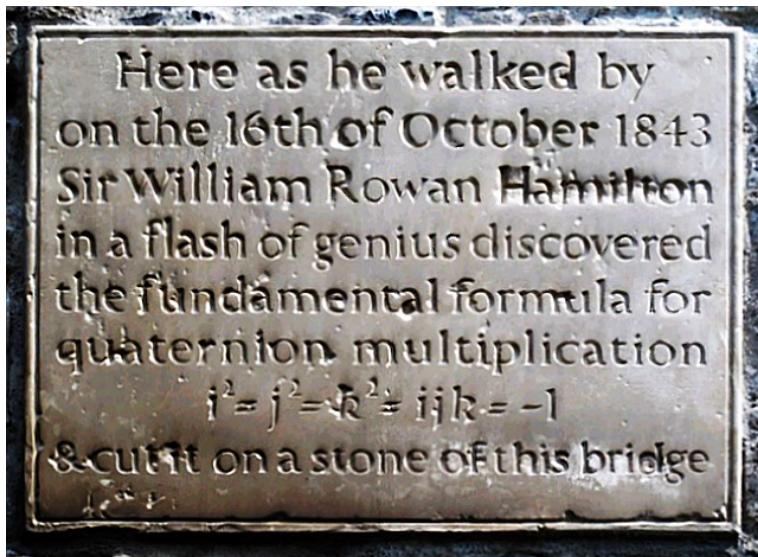
$$\begin{aligned}(a, b) + (c, d) &= (a + c, b + d) \\ (a, b) \cdot (c, d) &= (ac - bd, ad + bc)\end{aligned}\tag{2.3}$$

Kolejnym etapem badań Hamiltona było rozszerzenie opracowanej dotychczas algebry o kolejny wymiar, tworząc w ten sposób nową strukturę posiadającą jedną część rzeczywistą i dwie urojone. System miał nosić nazwę **Teoria trójek** (ang. *Theory of Triplets*). Istnieje wiele matematycznych powodów, dla których określenie takowych zasad byłoby istotne, jednakże Hamiltonowi zależało w głównej mierze na wykorzystaniu trójek do reprezentowania obrotu w przestrzeni trójwymiarowej, tak jak liczby zespolone mogą być używane do reprezentowania obrotu na płaszczyźnie dwuwymiarowej. Prace nad teorią trójek trwały ponad dziesięć lat, przynosząc ze sobą kolejne logiczne błędy oraz dodatkowo zwiększać w autorze chęć rozwiązania tego problemu. Do przełomowego momentu doszło 16 października 1843 roku, gdy podczas spaceru ze swoją żoną przez Kanał Królewski w Dublinie, Hamilton zrozumiał jak zbudowaćowy system. Zrozumiał on wówczas że jego nowa algebra będzie potrzebowała trzech części urojonych i , j oraz k , spełniających następujące warunki opisane za pomocą Równania 2.4.

$$i^2 = j^2 = k^2 = ijk = -1\tag{2.4}$$

Struktury nazwane później **kwaternionami** są szczególnie z kilku powodów. Przede wszystkim, nie przestrzegają one zasady przemienności, co oznacza, że $i \cdot j$ nie jest równe $j \cdot i$. W przypadku kwaternionów $i \cdot j = k$, jednakże $j \cdot i = -k$. To odróżnia kwaterniony od liczb rzeczywistych i zespolonych, które przestrzegają zasady przemienności.

Hamilton w przypływie emocji wyrzeźbił równanie na moście Broom, który następnie został nazwany mostem Hamiltona. Owy rysunek nie został zachowany, a widnieje na jego miejscu tablica upamiętniająca to wydarzenie, przedstawiona na Rysunku 2.11. Matematycy z Wydziału Matematyki Narodowego Uniwersytetu Irlandii w Maynooth co rok organizują spacer w to miejsce właśnie 16 października, nazywając to wydarzenie spacerem Hamiltona (ang. *Hamilton Walk*).



Rysunek 2.11: Tablica upamiętniająca odkrycie kwaternionów na moście Broom w Dublinie [32]

Sir Wiliam Hamilton poświęcił resztę swojego życia na pracę nad rozwojem i popularyzacją kwaternionów. Wydał wiele prac na ich temat, w tym książkę zatytuowaną *Lectures on Quaternions* (1853). Hamilton rozwijał dalej teorię kwaternionów, która okazała się niezwykle użyteczna w analizie matematycznej i algebrze liniowej. Należy również zaznaczyć iż rachunek macierzowy został wprowadzony dopiero w drugiej połowie XIX wieku, dlatego też wiele ówczesnych praw fizycznych zostało zdefiniowanych w zapisie kwaternionowym właśnie.

Odkrycie kwaternionów przez Hamiltona wpłynęło na wiele różnych dziedzin nauki. W matematyce, kwaterniony stały się podstawą do rozwoju algebry nieprzemiennej. W fizyce, są one używane w mechanice kwantowej i teorii względności. W informatyce, kwaterniony są używane do opisania obrotów w grafice komputerowej i robotyce.

2.3.2 Wstęp do kwaternionów

Algebra Kwaternionów

Kwaterniony są bytem matematycznym składającym się z czterech liczb rzeczywistych, w tym trzech określonych jako jednostki urojone. Każda ze składowych struktury posiada parametr skalarny nazywany często parametrem Eulera. Nie należy tej nazwy mylić z kątami Eulera. Obiekty te można dodawać i mnożyć jako pojedynczą jednostkę w podobny sposób, jak w zwykłej algebrze liczb. Istnieje jednak zasadnicza różnica. Kwaterniony są komutatywne co oznacza iż kolejność wykonywania działań w przypadku operacji mnożenia ma znaczenie i może zwracać różne wyniki w zależności od układu czynników. Oznacza to że z matematycznego punktu widzenia mnożenie kwaternionów nie jest przemienne [28]. Powszechny zapis kwaternionu q prezentuje Równanie 2.5.

$$q = a + bi + cj + dk \quad (2.5)$$

Parametry a, b, c, d są wspomnianymi wyżej wartościami skalarnymi w dziedzinie liczb rzeczywistych, natomiast i, j, k są jednostkowymi kwaternionami, spełniającymi następujące warunki:

- $ii = jj = kk = -1$,
- $ij = k, ji = -k$,
- $jk = i, kj = -i$,
- $ki = j, ik = -j$.

Uproszczenie powyższych zasad prowadzi bezpośrednio do określenia skróconej wersji warunków oraz podstawowego wzoru teorii trójkę przedstawionego Równaniem 2.4. Kwaterniony są również często postrzegane jako rozszerzenie zbioru liczb zespolonych. Pomimo tego iż nie jest to jedyna interpretacja tego bytu matematycznego, owe porównanie posłuży do przedstawienia podobieństw oraz różnic w przypadku dwóch podstawowych operacji arytmetycznych. W przypadku operacji dodawania kwaternionów, zachowują one wszystkie podstawowe prawa algebry, podobnie do zbioru liczb zespolonych które rozszerzają. Określając dwie jednostki q_1, q_2 za pomocą Równania 2.5, można przedstawić operacje ich dodawania w postaci przedstawionej przez Równanie 2.6.

$$\begin{aligned} R_1 &= (a_1 + b_1i + c_1j + d_1k) + (a_2 + b_2i + c_2j + d_2k) \\ R_2 &= (a_1 + a_2) + (b_1 + b_2)i + (c_1 + c_2)j + (d_1 + d_2)k \\ R_1 &= R_2 \end{aligned} \quad (2.6)$$

Operacja mnożenia kwaterionów jest bardziej złożona. Zasady mnożenia parametrów Eulera pozostają bez zmian jednakże w przypadku jednostek urojonych należy zastosować wspomniane wcześniej warunki wystarczające dla określenia liczby kwaterionem. W celu uproszczenia tej operacji, powstała Tabela 2.1 obrazująca wzajemne relacje parametrów kwaterionu oraz iloczyny każdego ze składowej tej struktury matematycznej.

Tabela 2.1: Tabela mnożenia jednostek kwaterionów

	1	i	j	k
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

Cyfra 1 oznacza w niej wartość części rzeczywistej, a kolejno i, j, k są jednostkami urojonymi. Mnożenie kwaterionów spełnia zatem prawo łączności oraz prawo dystryбуacyjne, jednakże nie spełnia prawa przemienności. Co za tym idzie kolejność składników tej operacji jest istotna.

Istotną operacją w kontekście reprezentacji rotacji przez kwateriony jest **sprzężenie kwaterionów**. Przejmuje ona jedną jednostkę, a jej wynikiem jest kwaterion o tych samych wielkościach, ale ze odwróconym znakiem części urojonych. Sprzężenie kwaterionów można opisać Równaniem 2.7.

$$\text{conj}(a + bi + cj + dk) = a - bi - cj - dk \quad (2.7)$$

Normalizacja kwaterionów jest często wykorzystywana operacją, w szczególności w przypadku ograniczania parametrów kwaterionu do sfery jednostkowej w celu reprezentowania rotacji. Operację normalizacji wyraża się wzorem przedstawionym za pomocą Równania 2.8.

$$Q = \frac{Q}{\sqrt{a^2 + b^2 + c^2 + d^2}} \quad (2.8)$$

Reprezentacja rotacji

Rozważając problematykę zastosowania kwaternionów w reprezentacji rotacji w trójwymiarowym środowisku, kluczowe jest zrozumienie ich czterowymiarowej struktury. Pomimo swego nieintuicyjnego charakteru, są one niezastąpione w kontekście rotacji, oferując efektywność oraz precyzję wykonywanych operacji. Istotnym jest fakt, iż rotacje reprezentowane za pomocą kwaternionów eliminują pewne problemy, które mogą wystąpić podczas korzystania z innych metod, takich jak macierze obrotu czy kąty obrotu Cardana, na przykład problem blokowania osi obrotu (ang. *gimbal lock*).

W kontekście reprezentowania rotacji, kwaternion można reprezentować wzorem $q = w + xi + yj + zk$ gdzie w, x, y, z są liczbami rzeczywistymi oraz i, j, k są jednostkami urojonymi. Elementy kwaternionu (w, x, y, z) są związane z rotacją w następujący sposób - skalar w reprezentuje cosinus połowy kąta rotacji, natomiast wektor (x, y, z) reprezentuje osь rotacji pomnożoną przez sinus połowy kąta rotacji. Można tę relację opisać również poniższym wzorem 2.9, gdzie α to kąt obrotu a (x, y, z) to osie obrotu.

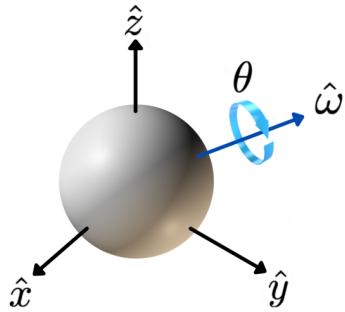
$$q = \cos(\alpha/2) + i(x \cdot \sin(\alpha/2)) + j(y \cdot \sin(\alpha/2)) + k(z \cdot \sin(\alpha/2)) \quad (2.9)$$

Kwaternion reprezentujący rotację można w przystępny sposób przedstawić w postaci punktów na krawędzi okręgu (lub na powierzchni hipersfery w przypadku rotacji wokół więcej niż jednej osi), gdzie poszczególne punkty odpowiadają konkretnym kątom obrotu. Wizualizacja takiej hipersfery została przedstawiona na Rysunku 2.12 [12]. Kluczową cechą kwaternionów jest to, że kąt obrotu wynosi dwukrotność wartości kąta reprezentowanego przez punkt na okręgu czy na sferze. Owa cecha odnosi się bezpośrednio do zjawiska o nazwie podwójne pokrycie (ang. *double cover*). Oznacza to, że dwie różne wartości kwaternionów mogą reprezentować ten sam obrót w przestrzeni trójwymiarowej.

Ilustrując opisaną cechę, jeśli do składników W i X kwaternionu wprowadzimy wartości $(0.707, 0.707)$, odpowiadające 45° na okręgu, otrzymamy w efekcie obrót o 90° wokół osi X . Wszystkie kwaterniony reprezentujące rotację są kwaternionami jednostkowymi, co oznacza, że ich długość wynosi 1. Wówczas długość kwaternionu reprezentującego rotację można wyrazić jako Równanie 2.10.

$$\sqrt{w^2 + x^2 + y^2 + z^2} = 1 \quad (2.10)$$

Warunek normalizacji kwaternionu w kontekście tego problemu przekłada się bezpośrednio na nazwę modelu wizualizacji gdyż wspomnianą wcześniej hipersferę nazywa się powszechnie **hipersferą jednostkową**.



Rysunek 2.12: Rotacja reprezentowana przez kwaternion [19]

Wartym przytoczenia jest również zjawisko związane bezpośrednio z podwójnym pokryciem. **Kwaterniony bliźniacze** cechują się tym, że każdy kwaternion posiada parę, reprezentującą tę samą końcową rotację, ale z przeciwnym zwrotem. Ten fakt ma kluczowe znaczenie dla interpolacji sferycznej (ang. *spherical interpolation, SLERP*), gdzie wybór pomiędzy bliźniaczymi kwaternionami pozwala na kontrolę nad kierunkiem obrotu.

Dokonując rotacji punktu w trójwymiarowej przestrzeni, używa się operacji mnożenia kwaternionów. Wektor opisujący punkt przekształca się w kwaternion z zerową częścią rzeczywistą. Następnie wykorzystuje się specyficzny dla kwaternionów schemat mnożenia. Schemat ten polega na mnożeniu kwaternionu reprezentującego rotację przez kwaternion reprezentujący punkt oraz przez sprzężenie kwaternionu reprezentującego rotację [28]. Efektem końcowym jest kwaternion, którego część urojona opisuje nowe położenie punktu po rotacji. Po ponownym przekształceniu wyniku do postaci wektorowej, otrzymujemy nowe współrzędne wprowadzonego punktu. Przedstawioną operację opisuje Równanie 2.11.

$$P_{out} = q \cdot P_{in} \cdot conj(q) \quad (2.11)$$

Parametry powyższego wzoru odpowiadają:

- P_{out} i P_{in} - punkty w przestrzeni 3D reprezentowane przez części i , j oraz k kwaternionu.
- $conj()$ - funkcja sprzężająca kwaternion.
- q - kwaternion reprezentujący obrót za pomocą Równania 2.9.

Należy podkreślić, że mnożenie kwaternionów nie jest przemienne, co ma istotne znaczenie przy łączeniu rotacji. Kolejność mnożenia kwaternionów odpowiada kolejności wykonywania rotacji, a wynikowe mnożenie kwaternionów reprezentuje skumulowany obrót. W przypadku łączenia dwóch rotacji, wynikowy kwaternion jest równy iloczynowi kwaternionów reprezentujących te rotacje.

Przewaga kwaternionów nad macierzami rotacji

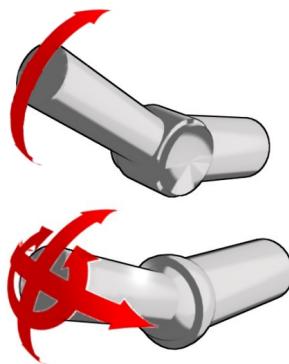
Zastosowanie kwaternionów w reprezentacji rotacji ma wiele zalet w porównaniu do kątów Cardana oraz macierzy rotacji zarówno na poziomie efektywności jak i precyzji wykonywanych obliczeń. Poniższa lista przedstawia część z zalet stosowania kwaternionów:

1. **Unikanie problemu blokowania osi obrotu** - utrata stopnia swobody w rotacji 3D, prowadzący do niezdolności reprezentowania pewnych obrotów. Zjawisko to jest spowodowane wykorzystaniem kątów Cardana do reprezentacji rotacji. Problem zachodzi gdy dwa z trzech stopni swobody zostają zbieżna względem siebie. Dwa osie obrotu ustawione równolegle powodują, że obrót wokół jednej generuje automatyczny obrót wokół drugiej zbieżnej do niej. Konsekwencją tego problemu jest zmniejszenie stopni swobody z trzech do dwóch co jest szczególnie problematyczne w dziedzinach takich jak animacja komputerowa. Kwaterniony, posiadając cztery elementy, zapewniają cztery stopnie swobody, co skutecznie eliminuje ten problem.
2. **Płynna interpolacja rotacji** - kwaterniony umożliwiają łatwą i płynną interpolację między dwoma orientacjami, co jest kluczowe w wielu zastosowaniach, takich jak animacje w grach komputerowych. Technika ta, zwana sferyczną interpolacją liniową, jest prosta do zaimplementowania i efektywna obliczeniowo w kontekście kwaternionów. W praktyce oznacza to, że posiadając dwie różne orientacje reprezentowane przez kwaterniony, można przystępnie obliczyć dowolny stan pośredni.
3. **Efektywność obliczeniowa** - kwaterniony są efektywniejsze obliczeniowo w porównaniu do kątów Cardana. W przypadku kątów Cardana, każda pojedyncza operacja rotacji wymaga przeliczenia i zastosowania macierzy obrotu. Mając do czynienia z wielokrotnymi rotacjami, na przykład obrotem wokół osi x , a następnie wokół osi y , należy przemnożyć odpowiednie macierze rotacji, abytrzymać jedną, końcową macierz, reprezentującą łączny obrót. Ta operacja wymaga wykonania szeregu operacji matematycznych i może prowadzić do błędów numerycznych. Dla kwaternionów, rotacje mogą być łatwo łączone poprzez mnożenie kwaternionów, bez konieczności przeliczania macierzy. To sprawia, że kwaterniony są zdecydowanie bardziej efektywne dla aplikacji, które wymagają dużej liczby operacji rotacji.
4. **Stabilność numeryczna** - kwaterniony są odporniejsze na błędy numeryczne, mogące gromadzić się podczas wielokrotnych operacji rotacji. Podlegają łatwej normalizacji, co oznacza, że mogą ciągle reprezentować prawidłowy obrót eliminując błąd akumulacji.
5. **Zwiększona precyzja** - czterowymiarowa natura kwaternionów pozwala na reprezentowanie obrotu z większą precyzją niż kąty Cardana. W przypadku wystąpienia małych kątów kwaterniony utrzymują wysoką precyzję dla dowolnego obrotu.

2.3.3 Analiza sekwencji kwaternionowych

Sekwencja kwaternionowa jako reprezentacja rotacji w czasie

W kontekście animacji i symulacji w czasie rzeczywistym, kluczowym jest przedstawianie zmiany orientacji obiektów w czasie. Kwaternion, pomimo swojej precyzji oraz efektywności, określa jedynie stan orientacji w danej chwili. Aby umożliwić opis rotacji przy pomocy kwaternionów, należy przedstawić ich zmianę w czasie, czyli stworzyć sekwencję kwaternionów. Przez sekwencję kwaternionów należy rozumieć ciąg, w którym kolejne elementy są stanami orientacji określonego obiektu bądź stawu w danym czasie. Pozwala to na reprezentację zmiany orientacji w czasie a co za tym idzie przedstawienie dynamiki obiektów w przestrzeni trójwymiarowej. Przykład rotacji stawu w symulacji 3D został przedstawiony na Rysunku 2.13.



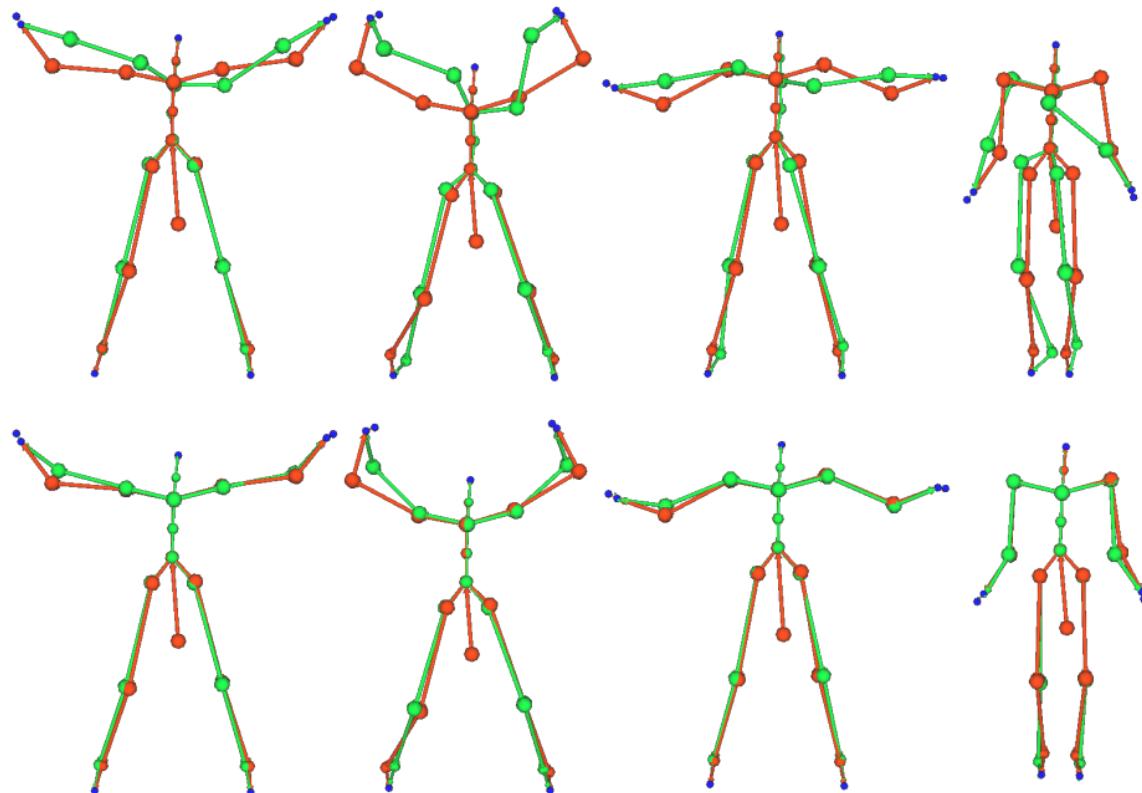
Rysunek 2.13: Rotacja stawu w grafice 3D [7]

Z sekwencjami kwaternionów wiąże się bezpośrednio zagadnienie dyskretyzacji. Odnosi się ono do procesu przekształcenia ciągłej funkcji rotacji, uzyskanej przykładowo z systemu przechwytywania ruchu, w serię dyskretnych migawek kwaternionów w określonych punktach czasu. Każda taka migawka to pojedynczy kwaternion reprezentujący orientację obiektu w danym momencie. Proces ten jest niezbędny do praktycznej realizacji animacji, ponieważ komputer musi operować na skończonym zestawie danych. Skutkuje to jednak późniejszą potrzebą przeprowadzenia procesu interpolacji pomiędzy kolejnymi elementami w sekwencji w celu uzyskania płynnej rotacji punktów w wizualizacji trójwymiarowej.

Interpolacja pomiędzy klatkami jest techniką, pozwalającą na tworzenie płynnej animacji pomimo dyskretnej natury danych. Najpopularniejszą metodą interpolacji kwaternionów jest SLERP, która pozwala na obliczenie kwaternionu reprezentującego orientację obiektu w dowolnej chwili czasu pomiędzy dwiema klatkami. SLERP jest preferowany względem podstawowych, liniowych interpolacji, ponieważ zapewnia jednolitą prędkość rotacji pomiędzy dwoma kwaternionami, a także zachowuje ich długość, co jest kluczowe dla utrzymania prawidłowej orientacji obiektu. W praktyce metoda służy do wygładzania przejścia pomiędzy dyskretnymi migawkami kwaternionów w procesie animacji.

Uzyskanie sekwencji kwaternionowej

Istnieje wiele źródeł danych pozwalających na uzyskanie sekwencji kwaternionów jako zmiany orientacji obiektów w czasie. Najczęszszym z nich są formaty danych odpowiadające za przechowywanie animacji szkieletowych. Owe pliki animacji można pozyskać zarówno z systemów przechwytywania ruchu (ang. *motion capture*) jak i z ręcznie wykonanych animacji w programach graficznych do modelowania 3D. Charakteryzują się one reprezentacją stawów w postaci nieskończenie małych punktów powiązanych ze sobą tworząc w ten sposób szkielet przeznaczony do animacji. Przykład animacji szkieletowej został przedstawiony na Rysunku 2.14.



Rysunek 2.14: Animacja szkieletowa [7]

Stan pozycji i orientacji odpowiednich stawów względem tych, z którymi ma połączenie jest następnie reprezentowany w postaci wektorowej lub kwaternionowej w przestrzeni dyskretnej. Zmiana stanu stawów następuje przy zmianie klatki, dlatego istotnym jest wykorzystanie interpolacji w celu uzyskania płynnego ruchu przy implementacji danej animacji w grach bądź filmach.

2.4 Przegląd literatury

2.4.1 Quaternion convolutional neural network for color image classification and forensics

Problem użycia kwaternionów w sieciach głębokiego uczenia jest tematem wielu książek oraz publikacji naukowych. Jednym z przykładów jest artykuł omawiający wykorzystanie kwaternionów w celu zwiększenia możliwości oraz wydajności sieci splotowych o tytule *Quaternion convolutional neural network for color image classification and forensics* [24]. Autorzy rozpoczynają swoją publikację od przedstawienia problemu, podkreślając iż sieci konwolucyjne mają znaczący wpływ w obecny rozwój w dziedzinie klasyfikacji obrazów oraz ekstrakcji cech znajdujących się na nich. Zwracają jednak uwagę na zasadnicze ograniczenia, którymi cechuje się architektura sieci CNN. Tradycyjne sieci splotowe postrzegają trzy kanały, opisujące kolejno kolor czerwony, zielony i niebieski, (ang. *Red, Green, Blue, RGB*) jako niepowiązane mapy cech, co prowadzi do utraty informacji o relacjach między kanałami. W odpowiedzi na to ograniczenie, autorzy proponują model **kwaterionowej, konwolucyjnej sieci neuronowej** (ang. *quaternion convolutional neural network, QCNN*). W tym podejściu, kolorowy obraz reprezentowany jest jako macierz kwaternionów, umożliwiając przetwarzanie koloru jako liczby hiperzespolonej, zamiast trzech oddzielnych kanałów, co z kolei pozwala zachować informacje o powiązaniach między kanałami. QCNN, w przeciwieństwie do tradycyjnego CNN, utrzymuje nie tylko pionowe zależności między cechami, ale również poziome powiązania kanałów kolorów.

W przeciwieństwie do tradycyjnych rzeczywistych sieci neuronowych, wykorzystujących macierze liczb rzeczywistych jako jednostki pomiarowe mapy cech, omawiana publikacja opisuje projekt, w którym każda mapa cech jest macierzą kwaternionową. Zastosowanie owej poprawki zmusiło autorów do wykonania modyfikacji poszczególnych kroków procesu konwolucji, takich jak normalizacja, aktywacja i podpróbkowanie (ang. *pooling*), aby dostosować je do formatu danych kwaternionowych. Do opisanych poprawek należą:

- **Normalizacja wsadowa kwaternionów** - innowacyjna technika normalizacji wsadowej, zaprojektowana specjalnie pod dane kwaternionowe. Pozwala na utrzymanie korelacji między kanałami kolorów, co jest korzystne w przetwarzaniu obrazów.
- **Kwaterionowa funkcja aktywacji** - rektyfikowana jednostka liniowa w sieciach QCNN pozwala zachować nieliniowość modelu, niezbędną do uczenia się cech.
- **Kwaterionowa warstwa w pełni połączona** - ostatnia warstwa sieci, łącząca wszystkie wyuczone cechy z poprzednich warstw w celu dokonania klasyfikacji. Zarazem adaptując ten proces do pracy z kwaternionami.
- **Moduł uwagi kwaternionów** - innowacyjny moduł uwagi, pozwalający na ekstrakcję najważniejszych cech, redukując jednocześnie wpływ zbędnych informacji. Dzięki temu, sieć neuronowa jest w stanie lepiej optymalizować proces uczenia, szczególnie w przypadku dużej ilości cech.

Ostateczny projekt architektury sieci QCNN składa się z dwóch bloków konwolucyjnych, dwóch warstw maksymalnego podpróbkowania (ang. *max-pooling*) oraz dwóch warstw w pełni połączonych, opartych na kwaternionach. Eksperymenty autorów wykazały, iż zastosowanie normalizacji wsadowej kwaternionów oraz modułu uwagi kwaternionów wpływa korzystnie na dokładność sieci. Zaproponowany model QCNN, przetestowany w kontekście klasyfikacji obrazów kolorowych i forenzyki, wykazał wyższą skuteczność w porównaniu z tradycyjnymi sieciami CNN. W przypadku badania w kontekście klasyfikacji obrazów QCNN uzyskał wyższy wynik dokładności sieci, bo **0.842** w porównaniu do podstawowej sieci splotowej, która uzyskała **0.755**. Autorzy zapowiedzieli również pracę nad rozwojem architektury sieci QCNN.

2.4.2 Quaternion recurrent neural networks

Sieci splotowe nie są jedynym działem głębokiego uczenia wykorzystującym kwaterniony. Istnieje również wiele publikacji naukowych przedstawiających użycie Trójkę Hamiltona w sieciach rekurencyjnych, zestawiając je w sekwencje powiązanych logicznie danych, zmieniających się w jednostce czasu. Jednym z przykładów takich prac jest publikacja pod tytułem *Quaternion recurrent neural networks*, zaprezentowana w 2018 roku przez grupę siedmiu naukowców z Francji oraz Kanady [23].

Publikacja skupia się na rekurencyjnych sieciach neuronowych (RNN), skutecznych w modelowaniu danych sekwencyjnych dzięki zdolności uczenia się zależności krótkoterminowych oraz długoterminowych relacji między elementami sekwencji. Autorzy zwracają uwagę iż popularne zadania, takie jak rozpoznawanie mowy czy obrazów, zawierają wielowymiarowe wejścia z silnymi zależnościami między sobą. Zawracają oni uwagę na problem pomijania wspominanych powiązań w przypadku uczenia z wykorzystaniem standardowych architektur sieci RNN. Autorzy proponują nowy typ sieci, będący kwaterionową sieć neuronową (ang. *Quaternion Recurrent Neural Networks*, QRNN) oraz kwaterionową sieć o pamięci długotrwałej (ang. *Quaternion Long Short-Term Memory*, QLSTM). Nowe podejście wykorzystuje algebra kwaternionów w celu uwzględnienia zewnętrznych relacji między elementami pojedynczej składowej sekwencji oraz wewnętrznych zależności strukturalnych. Kwaternioni pozwalają QRNN kodować wewnętrzne zależności, traktując i przetwarzając wielowymiarowe cechy jako pojedyncze jednostki. Zawarte w publikacji badania dowodzą iż kwaterionowe wersje znanych sieci posiadają większą wydajność oraz wymagają mniej parametrów od standardowej architektury sieci rekurencyjnej do wykonania tego samego zadania. Według opisanych badań, konkretne przypadki zastosowania QRNN pozwalają trzykrotnie zredukować liczbę potrzebnych parametrów.

Bazując na sukcesie wcześniejszych głębokich kwaterionowych sieci konwolucyjnych, autorzy proponują dostosowanie reprezentacji hiperzłożonych liczb do zdolności sieci rekurencyjnych, tworząc naturalny i efektywny model do zadań sekwencyjnych, takich jak roz-

poznawanie mowy. Zwracają również uwagę na efektywność zastosowania kwaterionów w przypadku przestrzeni wielowymiarowych. Przedstawiają oni nową architekturę bazującą w pełni na danych kwaterionowych. Dane wejściowe, wyjściowe, wagi oraz obciążenia sieci zostały zmienione z wartości skalarnych na Trójkę Hamiltona zmniejszając w ten sposób liczbę parametrów modelu oraz umożliwiając tworzenie wewnętrznych relacji pomiędzy nimi. Przykładem przytoczonym przez autorów było zastosowanie współczesnych systemów rozpoznawania mowy wykorzystujących sekwencje wejściowe składające się z wielowymiarowych cech akustycznych, które są często wzbogacane o ich pierwsze, drugie i trzecie pochodne czasowe. W przypadku standardowych sieci rekurencyjnych, traktujących wspomniany zestaw czterech danych jako tensor, wartości każdej z pochodnych traktowane są bez wewnętrznej zależności między sobą, co rozwiązuje zastosowanie kwaterionów traktujących cztery skalary jako jeden byt. Implementacja sieci w całości opartej na danych kwaterionowych wymusiła na autorach utworzenie nowych algorytmów pozwalających na pracę z tym typem struktur. Podstawowa operacja iloczynu skalarnego została zastąpiona złożonym mnożeniem kwaterionów a znany z sieci rekurencyjnych algorytm propagacji wstecznej przez czas został urozmaicony o operacje kwaterionowe, gdzie znajdywanie minimum gradientu odbywa się z uwzględnieniem każdej ze składowych danej kwaterionowej. Autorzy wspominają również iż zastosowanie kwaterionów pozwala na lepszą generalizację danych ze względu na możliwość bezpośredniego osadzenia wartości w wielowymiarowej przestrzeni. Jako przykład funkcji straty została użyta funkcja błędu średniokwadratowego.

Przedstawiony tekst zawiera szczegóły eksperymentów dotyczących sieci neuronowych QRNN i QLSTM do zadań rozpoznawania mowy. Autorzy przeprowadzają eksperyment na surowym dźwięku przetwarzanym na 40-wymiarowe współczynniki. Reprezentacja kwaterionowa dla cech akustycznych obejmuje natężenie dźwięku oraz jej pierwszą, drugą i trzecią pochodną. Dane treningowe obejmują 3696 zdań od 462 mówców, dane testowe składają się ze 192 fraz od 24 osób i używany jest również zestaw walidacyjny składający się z 400 próbek od 50 rozmówców. Modele są porównywane w oparciu o stałą liczbę warstw i różne liczby neuronów. Należy zwrócić uwagę iż sieci kwaterionowe ze względu na swoją strukturę pozwalają na generowanie czterokrotnie więcej informacji z jednego neuronu niż w przypadku standardowych sieci. Wnioski z badań są następujące.

- QRNN wymagają do 2,25 raza mniej parametrów od RNN.
- QRNN zapewniają lepszy współczynnik błędu fonemów (ang. *phoneme error rate*, PER) z 18,5% w porównaniu do RNN, z wynikiem 19,0% na zestawie testowym.
- Najlepszy zaobserwowany współczynnik PER wyniósł 15,1% dla QLSTM i 15,3% dla LSTM na zbiorze testowym.
- QLSTM osiągnął wynik przy 3,3 razy mniejszej liczbie parametrów niż LSTM.

Autorzy podsumowują swoje badania twierdzeniem iż modele oparte na kwaternionach (QRNN i QLSTM) konsekwentnie przewyższają swoje odpowiedniki z liczbami rzeczywistymi pod względem wydajności modelu, ponieważ wymagają mniejszej liczby parametrów. Wspominają również iż wykazują one obiecującą zdolność do rozwiązywania złożonych zadań, takich jak rozpoznawanie mowy, przy mniejszej ilości zasobów obliczeniowych, co czyni je potencjalnie idealnymi dla urządzeń o ograniczonej mocy obliczeniowej, takich jak smartfony.

2.4.3 Praca doktorancka - Quaternion neural networks

Szczegółowy opis techniczny implementacji sieci kwaternionowych oraz zasadność ich użycia została zawarta w pracy doktoranckiej pod tytułem *Quaternion neural networks* autorstwa Titouana Parcolleta [22]. Szczególną uwagę należy zwrócić na temat sieci rekurencyjnych, któremu autor poświęcił cały rozdział. Przedstawia on w nim koncept działania sieci rekurencyjnych jako podstawowe komórki RNN oraz ich rozwinięcia o pamięć długotrwałą w architekturze LSTM oraz GRU (ang. *gated recurrent unit*) będącej bramkową jednostką rekurencyjną. Zaznacza jednak problem gdyż tradycyjne sieci rekurencyjne traktują wielowymiarowe dane jako wiele niepowiązanych komponentów. Aby rozwiązać ten problem, wprowadzono **kwaternionowe rekurencyjne sieci neuronowe** (ang. *Quaternion Recurrent Neural Networks (QRNN)*). Omawiany rozdział przedstawia definicję QRNN oraz przedstawia mechanizm uczenia dla tych sieci. QRNN jest następnie rozszerzany o Quaternion Long-Short Term Memory Neural Network (QLSTM), wyposażony w bramki, eliminujące główne wady prostych QRNN.

Autor rozpoczyna szczegółową analizę nowej architektury QRNN poprzez opis zmian, które musiały zostać wprowadzone do procesu trenowania modelu. Zważywszy na fakt iż w modelu QRNN wszystkie wartości są kwaternionami, wliczając w to dane wejściowe, wyjściowe, wagi oraz obciążenia, podstawową zmianą była modyfikacja funkcji obliczającej stan ukryty sieci. Pod względem struktury algorytmu przebiegu w przód sieć QRNN nie różni się znaczco od swojego poprzednika. Owy przebieg polega na obliczeniu ukrytego stanu $h_{t,l}^n$ neuronu n w kroku czasowym t i warstwie l . Jest to kombinacja liniowa cech wejściowych i poprzedniego kroku czasowego ($t - 1$). Jedyną istotną zmianą jest zastąpienie wszystkich operacji iloczynu skalarnego, produktem Hamiltona, czyli mnożeniem kwaternionów. Następnie wyjście sieci w neuronie n , warstwie l , i kroku czasowym t jest obliczane zgodnie z kwaternionową nieliniową transformacją. Dokładny wzór procesu obliczania wartości stanu ukrytego został zaprezentowany w Równaniu 2.12.

$$h_n^{t,l} = \alpha \left(\sum_{m=0}^H w_{nm,hh}^l \otimes h_m^{t-1,l} + \sum_{m=0}^{N_{l-1}} w_{nm,h\gamma}^l \otimes \gamma_m^{t,l-1} + b_n^l \right) \quad (2.12)$$

Analogicznie, przebieg w tył, oraz bezpośrednio z nim związany algorytm propagacji wstecznej również musiał otrzymać odpowiednie poprawki. Kwaterionowa propagacja wsteczna przez czas (ang. *Quaternion Backpropagation Through Time, QBPTT*) to rozszerzenie standardowej wstecznej propagacji kwaterionów. Gradient względem funkcji straty E_t dla każdej macierzy wag wyrażonej jako Równanie 2.13

$$\Delta_{hy}^t = \frac{\partial E^t}{\partial W_{hy}}, \Delta_{hh}^t = \frac{\partial E^t}{\partial W_{hh}}, \Delta_{hx}^t = \frac{\partial E^t}{\partial W_{hx}} \quad (2.13)$$

oraz wektora biasu opisanego poprzez Równanie 2.14

$$\Delta_b^t = \frac{\partial E^t}{\partial B_h} \quad (2.14)$$

jest generowany do postaci przedstawionej w Równaniu 2.15

$$\Delta^t = \frac{\partial E^t}{\partial W} \quad (2.15)$$

przy pomocy wzoru zobrazowanego w Równaniu 2.16.

$$\Delta^t = \frac{\partial E^t}{\partial W} = \frac{\partial E^t}{\partial W^r} + \mathbf{i} \frac{\partial E^t}{\partial W^i} + \mathbf{j} \frac{\partial E^t}{\partial W^j} + \mathbf{k} \frac{\partial E^t}{\partial W^k} \quad (2.16)$$

Każdy człon powyższej relacji jest następnie obliczany przez zastosowanie reguły łańcuchowej. W przeciwieństwie do propagacji wstecznej o wartości rzeczywistej, QBPTT definiuje dynamikę strat w odniesieniu do każdego składnika kwaterionowych parametrów neuronowych. Funkcja straty jest kwaterionowy, średni kwadrat błędu.

Według autora, określenie odpowiedniej funkcji straty dla sieci kwaterionowych nie jest zadaniem trywialnym. Do refleksji może skłonić również wybór funkcji MSE, która jest stosunkowo prostą strukturą w porównaniu do złożonych sieci wykorzystujących trójkę Hamiltona. Autor pracy zwraca uwagę, że definicja funkcji straty dla kwaterionów pozostaje nieroziązanym problemem. Nie da się stwierdzić, czy kwateriony naturalnie pasują do zadań klasyfikacyjnych. Ich rozkład prawdopodobieństwa nie jest trywialny, a tradycyjne etykiety są reprezentowane jako wektory jednokierunkowe lub wektory binarne. Zatem reprezentacja kwaterionów nie jest prosta do rozważenia, według autora, w przypadku modeli kwaterionowych często stosuje się standardowe funkcje straty, indywidualnie dobrane do rozwiązywanego problemu. W takim przypadku warto rozważyć zastąpienie warstwy wyjściowej tradycyjną warstwą o wartościach rzeczywistych ze względu na spójność. Niemniej jednak w publikacji *Neural networks for quaternion-valued function approximation* [1] wprowadzono rozszerzenie MSE na kwateriony do zadań aproksymacji, zastępując wyłącznie liczby rzeczywiste liczbami hiper-zespolonymi jak w Równaniu 2.17.

$$E = \frac{1}{N} \sum_{n=1}^N [(t_{rpn} - S_{rn}^M)^2 + (t_{ipn} - S_{in}^M)^2 + (t_{jpn} - S_{jn}^M)^2 + (t_{kpn} - S_{kn}^M)^2] \quad (2.17)$$

Kwaternionowy średni błąd kwadratowy (ang. *Quaternion mean squared error, QMSE*) jest funkcją podzielonej straty opartą na standardowym MSE. Brak funkcji kosztów uwzględniających algebrę kwaternionów jest kluczowym punktem, czyniąc modele kwaternionowe, modelami hybrydowymi. Należy również zwrócić uwagę na strukturę wzoru 2.17 gdyż pod względem implementacji nie posiada on różnic w porównaniu do standardowego MSE w odniesieniu do czterowymiarowego wektora. Prowadzi to do częstego wykorzystywania podstawowej funkcji MSE w zagadnieniach związanych z sieciami kwaternionowymi.

Tradycyjne sieci RNN mają trudności z obsługą długoterminowych zależności w sekwencjach danych. Mogą też napotykać na problemy z zanikającym lub eksplodującym gradientem, co utrudnia lub uniemożliwia ich trening. Te problemy zostały zmniejszone przez wprowadzenie LSTM (ang. *Long Short-Term Memory*) z mechanizmami bramek, które pozwalają modelowi aktualizować lub zapominać informacje w komórkach pamięci. Sieci LSTM osiągnęły najlepsze wyniki w wielu benchmarkach opartych na złożonych zależnościach czasowych. Autorzy zainspirowani tymi wynikami, zaproponowali rozszerzenie sieci LSTM do dziedziny kwaternionów, tworząc QLSTM. Posiadają one dodatkowe komponenty takie jak wagi i bramki, przykładowo zapomnienia, wejścia i wyjścia. Każda z tych bramek kontroluje inny przepływ informacji, które są następnie łączone, aby uzyskać pamięć QLSTM, nazywaną też stanem komórki. Następnie definiowany jest ukryty stan.

Aby zdefiniować QLSTM, należy wprowadzić kolejne komponenty: $w_{f,hh}$, $w_{f,h\gamma}$, $w_{i,hh}$, $w_{i,h\gamma}$, $w_{o,hh}$, $w_{o,h\gamma}$ jako równoważne wagi QRNN, dla bramek zapomnienia $f_n^{t,l}$, wejścia $i_n^{t,l}$ i wyjścia $o_n^{t,l}$ odpowiednio w krokach czasowych t , warstwie l i neuronie n . Wówczas wartość kolejnych bramek można wyznaczyć przy pomocy wzorów zaprezentowanych w Równaniach 2.18, 2.19 oraz 2.20 gdzie α oznacza kwaternionową funkcję aktywacji.

$$f_n^{t,l} = \alpha \left(\sum_{m=0}^H w_{f,nm,hh}^l \otimes h_m^{t-1,l} + \sum_{m=0}^{N_{l-1}} w_{f,nm,h\gamma}^l \otimes \gamma_m^{t,l-1} + b_{f,n}^l \right) \quad (2.18)$$

$$i_n^{t,l} = \alpha \left(\sum_{m=0}^H w_{i,nm,hh}^l \otimes h_m^{t-1,l} + \sum_{m=0}^{N_{l-1}} w_{i,nm,h\gamma}^l \otimes \gamma_m^{t,l-1} + b_{i,n}^l \right) \quad (2.19)$$

$$o_n^{t,l} = \alpha \left(\sum_{m=0}^H w_{o,nm,hh}^l \otimes h_m^{t-1,l} + \sum_{m=0}^{N_{l-1}} w_{o,nm,h\gamma}^l \otimes \gamma_m^{t,l-1} + b_{o,n}^l \right) \quad (2.20)$$

Każda bramka kontroluje inny przepływ informacji, który jest następnie łączony w celu uzyskania pierwszej części pamięci QLSTM, zwanej również stanem komórki $c_n^{t,l}$, zdefiniowaną wzorem 2.21.

$$c_n^{t,l} = f_n^{t,l} c_n^{t-1,l} + i_n^{t,l} \tanh \left(\sum_{m=0}^H w_{c,nm,hh}^l \otimes h_m^{t-1,l} + \sum_{m=0}^{N_{l-1}} w_{c,nm,h\gamma}^l \otimes \gamma_m^{t,l-1} + b_{c,n}^l \right) \quad (2.21)$$

Gdzie $w_{c,hh}$, $w_{c,h\gamma}$ są macierzami wag stanów komórek, a wszystkie iloczyny z wyjątkiem \otimes są iloczyńami składowych. Należy zauważać, że tanh jest kwaternionową funkcją aktywacji. Ostatecznie, stan ukryty $h_n^{t,l}$ jest zdefiniowany jak w Równaniu 2.22.

$$h_n^{t,l} = o_n^{t,l} \tanh(c_n^{t,l}) \quad (2.22)$$

Gdzie $w_{h,hh}$ oraz $w_{h,h\gamma}$ są macierzami wag stanów ukrytych.

Należy zwrócić uwagę na działanie bramek sieci QLSTM gdyż ich akcje stosowane są poprzez iloczyn skalarny zamiast iloczyn kwaternionów. Mnożenie wartości kwaternionowych nie odbywa się poprzez wzór sformułowany w algebrze kwaternionów. W zamian wykorzystywany jest standardowy iloczyn skalarny wymnażający wartości pomiędzy odpowiednimi parametrami. Potencjały bramek reprezentują ilość informacji, która ma być eksponowana, zapomniana lub aktualizowana. W związku z tym ważne jest, aby zapewnić pełną kontrolę nad tymi wartościami w komponentach sygnału kwaternionowego za pomocą iloczynu skalarnego. Mniej złożone operacje mogą również zapobiec przeuczeniu oraz przyspieszyć proces trenowania sieci.

Po zaprojektowaniu oraz konstrukcji komórek QLSTM autor przystąpił do przeprowadzenia testu, który miał dowieść że wprowadzenie kwaternionów w sieć LSTM nie zaburzyło zdolności uczenia się zależności długoterminowych i krótkoterminowych. Wykorzystał w tym celu zadanie kopiowania, które jest syntetycznym testem przedstawiającym, jak modele oparte na RNN radzą sobie z długoterminową pamięcią. Wnioskiem z badania jest brak negatywnych konsekwencji z wprowadzenia kwaternionów do komórek LSTM. Kwaternionowa odmiana poradziła sobie w odpowiednich przypadkach lepiej od swojego poprzednika, posiadając przy tym mniej parametrów co skutkuje wyższą zdolnością do generalizacji.

2.5 Przegląd istniejących rozwiązań

Opisywane w poprzednim rozdziale publikacje zostały opublikowane w latach 2018-2019. Fakt ten świadczy o tym iż kwaterionowe sieci głębokiego uczenia są nowym, jednakże przejnie rozwijającym się zagadnieniem w świecie nauki. Pomimo wielu słusznych przykładów wdrożenia sieci kwaterionowych, występujących w pracach naukowych, istnieje niewiele informacji odnośnie ich wykorzystania w przemyśle. W związku z tym niniejszy rozdział jest poświęcony rzeczywistym rozwiązańom istniejącym na rynku oraz możliwością ich rozbudowy o sieci kwaterionowe na podstawie konkretnych publikacji.

2.5.1 Systemy rozpoznawania mowy

Współczesne systemy rozpoznawania mowy stanowią produkt zaawansowanego rozwoju w dziedzinie sztucznej inteligencji i uczenia maszynowego. Jednym z najbardziej popularnych zastosowań technologii rozpoznawania mowy są asystenci wirtualni, tacy jak Siri (Apple), Google Assistant (Google) czy Cortana (Microsoft) [5]. Bazując przeważnie na głębokich sieciach neuronowych, umożliwiają konwersję akustycznego sygnału mowy na tekst z wysoką precyzją. Jednak, mimo osiągniętych sukcesów, istnieją pewne ograniczenia i wyzwania związane z przetwarzaniem wielowymiarowych cech sygnału akustycznego. Logotypy popularnych asystentów głosowych zostały przedstawione na Rysunku 2.15.



Rysunek 2.15: Logotypy asystentów: Alexa, Siri, Google Assistant, Cortana [5]

Według ostatnich badań naukowych [22], kwateriony oferują obiecujące możliwości w kontekście przetwarzania wielowymiarowych danych. Dzięki swojej unikalnej właściwości reprezentowania rotacji w przestrzeni trójwymiarowej, zdają się być doskonałym narzędziem do opisu złożonych zależności w danych akustycznych. Propozycja wprowadzenia sieci kwaterionowych do systemów rozpoznawania mowy wynika z chęci poszukiwania bardziej efektywnych metod przetwarzania wielowymiarowych cech akustycznych. Badania przeprowadzone na zadaniu rozpoznawania fonemów w zbiorze nagrani głosów TIMIT wykazały, że QRNN oraz QLSTM przewyższają tradycyjne RNN oraz LSTM pod względem efektywności, przy jednoczesnym zmniejszeniu liczby parametrów od 2 do 3 razy [23]. Algebra kwaterionowa oferuje lepszą i bardziej zwartą reprezentację dla wielowymiarowych cech a sieci kwaterionowe wykazują lepszą zdolność do uczenia się wewnętrznych zależności w danych.

Pomimo iż wspomniani wcześniej asystenci głosowi są już systemami zaawansowanymi, wciąż występują sytuacje, w których mają trudności z poprawnym rozpoznaniem mowy, zwłaszcza w hałaśliwym otoczeniu. Wprowadzenie sieci kwaternionowych do architektury takiego asystenta mogłoby przynieść szereg korzyści:

- **Lepsze przetwarzanie wielowymiarowej mowy:** Dzięki zdolności kwaternionów do reprezentowania złożonych zależności, asystent mógłby lepiej radzić sobie z złożonymi frazami lub akcentami.
- **Optymalizacja wydajności:** Sieci kwaternionowe potrzebują mniej parametrów do osiągnięcia porównywalnych lub lepszych wyników niż tradycyjne sieci. Oznacza to potencjalnie szybsze odpowiedzi i mniejsze zużycie zasobów urządzenia [23].
- **Poprawa jakości w trudnych warunkach:** W hałaśliwym otoczeniu, gdzie mowa może być zakłócona różnymi dźwiękami, sieci kwaternionowe mogą oferować lepszą zdolność do wychwytywania i rozróżniania informacji z cichych dźwięków.

Choć rynkowe systemy rozpoznawania mowy osiągnęły już imponujący poziom zaawansowania, ciągłe dążenie do innowacji i doskonałości jest kluczem do przyszłych sukcesów w tej dziedzinie. Użycie algebry kwaternionowej w formie QRNN i QLSTM stanowi obiecujący kierunek badań, oferując potencjał do tworzenia jeszcze bardziej precyzyjnych i efektywnych systemów przetwarzania mowy.

2.5.2 Inspekcja Wizualna w Produkcji Elektroniki

W przemyśle produkcji elektroniki, dokładna inspekcja wizualna jest kluczowa dla utrzymania wysokiej jakości produkowanych komponentów i urządzeń. Automatyczne systemy inspekcji wizualnej (ang. *Automated Optical Inspection, AOI*) skanują płytki drukowane (ang. *Printed Circuit Board, PCB*) w poszukiwaniu wad, takich jak niewłaściwie zamontowane elementy, źle przylutowane połączenia czy braki w komponentach.

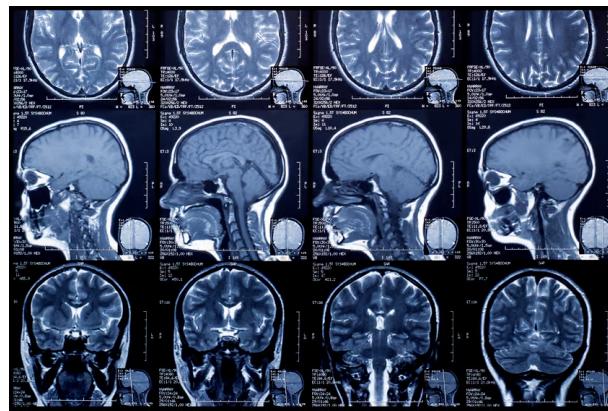
Tradycyjne systemy AOI korzystają z konwolucyjnych sieci neuronowych do rozpoznawania i klasyfikacji różnorodnych defektów. Jednakże ze względu na bogactwo kolorowych informacji oraz szczegółowość wymaganą w tej dziedzinie, zastosowanie kwaternionowych sieci konwolucyjnych może być wartościowe. Zaproponowana w literaturze architektura QCNN osiągnęła obiecujące wyniki w klasyfikacji obrazów kolorowych, co wskazuje na jej potencjał w złożonych zadaniach, takich jak detekcja czy segmentacja obiektów [24]. Zarazem, uwzględnienie trzech kanałów jednocześnie pozwala na bogatszą reprezentację informacji zawartej w obrazie, przekładającą się na większą precyzję oraz zdolność do radzenia sobie z skomplikowanymi scenami. Jest to istotne, w przypadku dokładnej inspekcji kolorowych obrazów płyt PCB, zawierających różne warstwy materiałów i składników. Takie podejście może umożliwić AOI dokładniejsze wykrywanie subtelnych wad, które są niezauważalne dla tradycyjnych systemów.

Podsumowując, zastosowanie QCNN w automatycznych systemach inspekcji wizualnej w przemyśle elektronicznym może przynieść znaczącą poprawę jakości wykrywania wad, co przekłada się na wyższą jakość końcowych produktów oraz niższe koszty związane z wadliwymi komponentami.

2.5.3 Analiza i przetwarzanie obrazów

Kwaternionowe sieci rekurencyjne i kwaternionowe sieci LSTM posiadają potencjał do przetwarzania danych wielowymiarowych w efektywniejszy sposób niż tradycyjne sieci. W dziedzinie analizy i przetwarzania obrazów, powszechnym podejściem jest wykorzystywanie obrazów kodowanych w standardzie RGB, gdzie każdy kolor reprezentowany jest jako oddzielnny kanał. Każdy z owych kanałów można traktować jako trzy wymiary w przestrzeni kwaternionowej. Chociaż tradycyjne kwaterniony posiadają cztery parametry, pozostały można wykorzystać jako dana pomocnicza bądź dodatkowy kanał przechowujący dane o przeźroczystości.

Wykorzystanie QRNN lub QLSTM umożliwiły przetwarzanie trzech kanałów jednocześnie, co mogłoby umożliwić bardziej skuteczne wykrywanie cech i związków między kolorami w obrazach. Przykładowo, podczas rozpoznawania wzorców na obrazie, kwaternionowe sieci rekurencyjne mogłyby lepiej uwzględnić subtelne różnice w odcieniach i kontrastach, które są trudne do wykrycia przy użyciu standardowych sieci. Jedno z konkretnych zastosowań może dotyczyć medycyny. W medycynie obrazowej, takiej jak rezonans magnetyczny czy tomografia komputerowa, często analizuje się obrazy wielowymiarowe. Przykład skanu rezonansu magnetycznego, rozłożonego na kilka płaszczyzn, został przedstawiony na Rysunku 2.16.



Rysunek 2.16: Obraz skanu rezonansu magnetycznego głowy [11]

Kwaternionowe sieci neuronowe mogą oferować bardziej efektywne analizy obrazów, pomagając w wykrywaniu i klasyfikacji chorób oraz patologii. Takie podejście może prowadzić do lepszego wykorzystania informacji zawartych w danych, skutkując precyzyjnymi i wiarygodnymi wynikami w stosunku do tradycyjnych technik analizy obrazów.

2.6 Wnioski z analizy tematu

Analizując obszerną historię oraz liczne przykłady zastosowania kwaternionów oraz uczenia maszynowego, naturalnym wydaje się stwierdzenie iż są to dziedziny informatyki posiadające duży wpływ na rozwój obecnej nauki oraz posiadają szerokie spektrum zastosowań. Przegląd literatury przedstawił publikacje dowodzące zasadności połączenia tych dwóch dziedzin. Kwaternionowe sieci głębokiego uczenia zyskują na popularności, głównie za sprawą zwiększonej wydajności modeli przy mniejszej liczbie parametrów oraz zdolności kwaternionów do reprezentowania złożonych zależności pomiędzy danymi. Ta informacja pozwala wstępnie potwierdzić zasadność hipotezy przedstawionej we wstępie, w myśl której sieci neuronowe głębokiego uczenia są zdolne do pracy z danymi kwaternionowymi. Przeprowadzone w latach 2018-2019 badania z pewnością zwiększają prawdopodobieństwo powodzenia, jednakże bez implementacji sieci oraz przeprowadzenia badań nie można ostatecznie określić słuszności tej hipotezy, w szczególności gdy problem ekstrapolacji orientacji z sekwencji kwaternionów nie był istotą omawianych prac naukowych.

Rozdział poświęcony analizie tematu zwrócił szczególną uwagę na dwa typy kwaternionowych sieci głębokiego uczenia. Pierwszym z nich są QCNN czyli kwaternionowa odmiana sieci splotowych, specjalizująca się w analizie obrazów z uwzględnieniem powiązań pomiędzy wszystkimi kanałami barw jednocześnie. Drugim typem są kwaternionowe sieci rekurencyjne, dokładniej QRNN oraz jego ewolucja QLSTM, wyposażona w bramki pamięci krótkotrwałej i długotrwałej. Rekurencyjne wersje sieci kwaternionowych pozwalają zatem na przetwarzanie sekwencji danych zakodowanych w formie Trójkę Hamiltona, pozwalając dodatkowo na tworzenie relacji nie tylko pomiędzy kolejnymi elementami sekwencji, ale również pomiędzy danymi wewnętrz jednej składowej.

W kontekście przytoczonego powyżej problemu, jakim jest przewidywanie dalszego postępu rotacji na podstawie wprowadzonej sekwencji kwaternionowej, zasadnym wydaje się wybór kwaternionowych sieci rekurencyjnych, zwłaszcza ze względu na sekwencje, które są głównym przedmiotem badań. Należy również zwrócić uwagę iż dane pozyskane z systemów przechwytywania ruchu mogą posiadać długie zależności czasowe, wprost proporcjonalne do występujących klatek w pliku. W tym przypadku słusznym będzie zastosowanie szczególnego typu kwaternionowych sieci rekurencyjnych czyli QLSTM, ze względu na ich zdolności do przechowywania relacji w długich sekwencjach danych.

W dalszych rozdziałach zawarty zostanie opis prac nad stworzeniem wybranej na podstawie analizy sieci QLSTM oraz przeprowadzonych badań wykonanych przy jej użyciu.

Rozdział 3

Przedmiot badań

3.1 Opis wykorzystanych narzędzi

3.1.1 Python

Interpretowany, wieloplatformowy język programowania wysokiego poziomu, charakteryzujący się dynamicznym typowaniem oraz zarządzaniem pamięcią. Jego cechą jest wsparcie dla wielu paradygmatów programowania, w tym proceduralnego, obiektowego i funkcyjnego, co przekłada się na jego wysoką popularność. Według danych udostępnionych przez serwis Github, w samym 2022 zainteresowanie Pythonem wzrosło o 22.5% [9]. Ze względu na bogatą bibliotekę standardową oraz szeroki wybór dostępnych pakietów zewnętrznych, Python znajduje zastosowanie w wielu dziedzinach naukowych, takich jak tworzenie aplikacji webowych, przetwarzanie grafiki komputerowej, analiza danych czy uczenie maszynowe. W ramach projektu wykorzystane zostały również jego biblioteki.

Numpy - Numeric Python

Biblioteka przeznaczona dla języka Python, której głównym celem jest wsparcie dla wielowymiarowych tablic oraz oferowanie szerokiej gamy funkcji matematycznych do operacji na tych tablicach. Umożliwia realizację złożonych operacji algebraicznych oraz statystycznych. Efektywność numpy wynika z wewnętrznych implementacji funkcji w językach C i Fortran, co przekłada się na wysoką wydajność obliczeniową.

Scipy - Scientific Python

Biblioteka rozszerzająca funkcjonalność oferowaną przez numpy o bardziej zaawansowane narzędzia niezbędne w naukowych obliczeniach. Scipy dostarcza moduły do optymalizacji, interpolacji, całkowania, algebraicznych równań różniczkowych i innych zadań typowo matematycznych czy inżynierskich. Biblioteka korzysta z tablic numpy, dostarczając narzędzi do bardziej specjalistycznych zastosowań naukowych.

3.1.2 Pytorch

Otwarta platforma do uczenia maszynowego rozwijana przez Facebook's AI Research lab. Wyróżnia się dynamicznymi obliczeniami graficznymi, umożliwiając modyfikację modeli podczas ich działania. Dzięki integracji z biblioteką NumPy oraz bogatemu interfejsu graficznemu (ang. *Application Programming Interface*, API), stała się cennym narzędziem dla naukowców oraz analityków danych. Dodatkowym atutem PyTorch jest wsparcie przetwarzania na procesor graficzny (ang. *graphics processing unit*, GPU) z wykorzystaniem systemu CUDA (ang. *Compute Unified Device Architecture*), znacznie przyspieszając obliczenia na dużych zbiorach danych. Platforma pozwoliła na implementację, trening oraz użycie modeli omawianych w tej pracy.

3.1.3 Quaternion Neural Networks Plugin

Wtyczka dla środowiska Pytorch autorstwa Titouana Parcolleta, będącego również twórcą pracy doktoranckiej *Quaternion neural network* [22, 31]. Opracowane przez Pana Parcolleta repozytorium oferuje aktualne implementacje wielu kwaternionowych sieci neuronowych w środowisku PyTorch. Zawiera również rozbudowane API, dostarczające zestaw narzędzi matematycznych w dziedzinie algebry Hamiltona oraz algorytmy umożliwiające przystępna pracę z kwaternionami w sieciach neuronowych.

3.1.4 CUDA

Platforma obliczeniowa i model programowania stworzona przez firmę NVIDIA. Zaprojektowana, by wykorzystywać możliwości obliczeniowe kart graficznych w celu przyspieszenia obliczeń wykonywanych poza obszarem renderowania grafiki. CUDA pozwala na bezpośrednie programowanie GPU poprzez specjalnie zaprojektowany język C/C++ rozszerzony o dyrektywy obsługujące równoległość. Jej zastosowanie przekłada się na znaczne przyspieszenie wielu operacji, szczególnie tych, które nadają się do równoległego przetwarzania, takich jak symulacje, analiza danych czy obliczenia z dziedziny uczenia maszynowego. Wsparcie CUDA przez platformę Pytorch pozwoliło na przeprowadzenie treningu omawianych modeli na karcie graficznej, znacznie przyspieszającowy proces.

3.1.5 Tensorboard

Narzędzie wizualizacyjne, zaprojektowane w celu monitorowania procesów związanych z uczeniem maszynowym. Pozwala na obserwację postępów w treningu modelu, analizę metryk oraz wizualizację architektur sieci neuronowych i przepływu danych przez grafy obliczeniowe. Ułatwia śledzenie ewolucji różnych metryk uczenia oraz parametrów modelu w trakcie treningu. Współpraca z wieloma bibliotekami do uczenia maszynowego, w tym z PyTorch, jest możliwa dzięki dedykowanym rozszerzeniom.

3.1.6 Autorskie rozwiązania

Na potrzeby przeprowadzenia badań utworzone zostały trzy programy których zadaniem jest w konsekwencji utworzenie zbioru treningowego i walidacyjnego dla procesu uczenia i ewaluacji omawianych modeli. Każda z tych aplikacji została zrealizowana w języku programowanie Python oraz jest obsługiwana z wiersza poleceń.

Interpreter plików C3D

W przypadku omawianej pracy wymagana jest ekstrakcja danych z plików pochodzących z systemu przechwytywania ruchu o rozszerzeniu *.c3d*. Format *C3D* zawiera specyficzną reprezentację danych w postaci trzech segmentów: nagłówka, parametry, dane. W celu uzyskania określonych wartości i ich zmianę w czasie, należy zlokalizować dane w sekcji danych na podstawie etykiet oraz informacji o częstotliwości klatek z sekcji parametrów oraz nagłówka [29]. Wykonywanie operacji na plikach *C3D* umożliwia program Mokka, jednakże nie dysponuje on bezpośrednią opcją eksportu danych zawartych w stawach do plików o znany formacie, przykładowo *.csv*.

Aby rozwiązać problem interpretacji danych C3D, utworzona została aplikacja umożliwiająca odczyt informacji zawartych w plikach *.c3d* oraz zapis do formatu *.csv*, który to następnie można otworzyć przy użyciu specjalistycznych programów, na przykład Microsoft Excel. Interpreter posiada również opcję wyboru informacji, które mają zostać zapisane. Spośród siedmiu dostępnych sekcji, użytkownik ma możliwość wybrać informacje dotyczące: markerów, wirtualnych markerów, modelowanych markerów, kątów, sił, momentów oraz mocy. W przypadku omawianej pracy, wykorzystywana będzie jedynie opcja zapisu kątów, która generuje zmianę kątów 27 stawów w nagraniu ruchu w postaci kątów Cardana. Ostateczny plik wyjściowy jest tabelą relacji klatki nagrania do wartości orientacji danych stawów w danej chwili czasu.

Konwerter kątów Cardana do kwaternionów

Zważywszy na fakt iż niniejsza praca opiera się na sekwencjach kwaternionów a pliki *.c3d* reprezentują orientacje stawów w postaci kątów Cardana, wymaganym było opracowanie programu konwertującego. Głównym założeniem aplikacji jest odczytywanie plików o rozszerzeniu *.csv* zwracanych przez interpreter *C3D* a następnie przekształcenie zawartych tam informacji do postaci kwaternionowej. Dane odczytywane są niezależnie dla każdego stawu oraz klatki po czym są grupowane w struktury reprezentujące kąty Cardana. Obiekty są następnie poddawane metodom wtyczki **Scipy**, dysponującej szerokim zestawem narzędzi naukowych, w tym mechanizmów konwertowania rotacji. Ostatecznie generowany jest plik o identycznej strukturze co wynik interpretera *.c3d*, jednakże zawierający po cztery wartości dla każdego stawu w danej chwili czasu, gdzie każdy z nich odpowiada pojedynczemu kwaternionowi.

Generator zbiorów treningowych

Program umożliwiający generowanie plików zawierających zestaw treningowy oraz odpowiadający mu zestaw przewidywanych wartości. Odczytuje zarówno dane zwracane przez interpreter plików *.c3d* oraz konwerter kwaternionów, przy wcześniejszym określeniu źródła danych. Aplikacja wymaga również określenia długości sekwencji **n**, przewidywanych dla sieci rekurencyjnej, na której będzie odbywał się trening. Po otrzymaniu pliku wejściowego, generuje zestaw treningowy poprzez wycinanie z niego danych przy użyciu okna o długości **n**. Zbiór walidacyjny jest wówczas określany poprzez zapis wartości na pozycji **n+1**. Program kończy swoje działanie generując dwa pliki o formacie *.csv*, będące gotowymi od czytania przez system trenowania i walidacji sieci.

3.2 Opis wykorzystanych sieci neuronowych

Istotą owej pracy jest zweryfikowanie hipotezy, w myśl której zastosowanie kwaternionów w sieciach neuronowych pozwoli na usprawnienie ich wydajności. Kluczowym z punktu widzenia przeprowadzanych badań będzie utworzenie sieci neuronowej głębokiego uczenia, wyposażonej w algorytmy stosujące kwaternionową algebrę Hamiltona. Model wykorzystywał będzie do procesu trenowania obszerny zestaw danych z systemu przechwytywania ruchu a docelowym wynikiem utworzonej sieci jest przewidywanie dalszego postępu rotacji na podstawie wprowadzonej sekwencji kwaternionowej.

Opierając się na wcześniej przeprowadzonej analizie literatury, w kontekście pracy nad sekwencjami czasowymi orientacji, wykorzystane zostaną komórki sieci rekurencyjnej QLSTM (ang. *Quaternion Long Short-Term Memory*), będącej kwaternionową ewolucją standardowej komórki LSTM (ang. *Long Short-Term Memory*), czyli długiej pamięci krótkotrwałej. Wybór owego modelu jest szczególny nie tylko ze względu na charakterystykę danych treningowych, ale również zauważalny na oczekiwany wynik. Sztuczne sieci neuronowe wykorzystujące komórki LSTM są szeroko stosowanym narzędziem wśród problemów związanych z predykcją danych.

Planowane badania będą miały charakter porównawczy, zestawiając ze sobą wyniki sieci LSTM oraz QLSTM. Istotą eksperymentów jest znalezienie przypadków, w których sieci rekurencyjne oparte na kwaternionach są wydajniejsze od ich standardowych implementacji, biorąc pod uwagę możliwość niewystąpienia takich przypadków. Dla potrzeb przeprowadzonych w owej pracy badań skonstruowano dwa modele sieci neuronowych. Pierwszy z nich opiera się na tradycyjnej architekturze LSTM, drugi to jego kwaternionowa ewolucja QLSTM. Pomimo tego iż oba modele posiadają identyczne warstwy i tę samą liczbę neuronów, różnią się strukturalnie w zakresie zastosowania kwaternionów. Celem implementacji dwóch sieci jest przeprowadzenie badań porównawczych między tradycyjnym modelem a nowatorskimi implementacjami bazującymi na kwaternionach.

3.2.1 Struktura modelu LSTM

Model LSTM jest rodzajem sieci neuronowej rekurencyjnej (ang. *Recurrent Neural Network*, RNN) pozwalającym na efektywne uczenie sekwencji danych. Sieci tego typu są szczególnie użyteczne w przypadkach, w których długozakresowe zależności w danych.

- **Warstwy LSTM** - oby dwie warstwy zawierają 128 jednostek z funkcją aktywacji **tanh** dla bramek oraz **sigmoid** dla resetowania i aktualizacji wartości stanu. Jedyną różnicą jest kwestia zwracanych danych, gdzie wynikiem pierwszej są pełne sekwencje, pozwalające na podłączenie kolejnej warstwy LSTM, która zwraca tylko ostatnią wartość z sekwencji pozwalającą na przewidywanie następnej wartości.
- **Warstwa wyjściowa** - warstwa w pełni połączona z liniową funkcją aktywacji. Zawierająca 4 jednostki odpowiadające kolejnym elementom wynikowego kwaternionu.
- **Operacja normalizacji kwaternionów** - proces normalizacji wyniku przy użyciu wzoru na normalizację kwaternionów, opisanego za pomocą Równania ?? w celu utrzymania rozwiązania na jednostkowej hipersferze.

3.2.2 Struktura modelu QLSTM

Realizacja sieci QLSTM opierała się na wykorzystaniu wtyczki *PyTorch Quaternion Neural Networks* [31], która dostarczała wstępna implementację, zawierającą algorytmy kwaternionowej propagacji wstecznej przez czas oraz kwaternionowe adaptacje procesów przebiegu w przód. Realizacja kompleksowej implementacji modelu QLSTM odbyła się z wykorzystaniem warstw *QuaternionLinearAutograd* oraz interfejsu, obejmującego metody algebry Hamiltona. Należy podkreślić, że zmodyfikowano pierwotną implementację o integrację wsparcia dla wielowarstwowych modeli QLSTM oraz optymalizowano ją w zakresie formatowania tensorów reprezentujących rotacje w formie kwaternionowej oraz normalizację kwaternionową końcowego wyniku.

- **Warstwy QLSTM** - w tym przypadku również oby dwie warstwy zawierają 32 jednostki z funkcją aktywacji **tanh** dla bramek oraz **sigmoid** dla resetowania i aktualizacji wartości stanu. Jednak pierwsza warstwa wraca pełne sekwencje, aby móc podłączyć kolejną warstwę QLSTM, której wynikiem jest tylko ostatnią wartość z sekwencji, służącą do przewidywania następnej wartości.
- **Warstwa wyjściowa** - podstawowa warstwa w pełni połączona, operująca na wartościach rzeczywistych z liniową funkcją aktywacji. Zawierająca 4 jednostki odpowiadające kolejnym elementom wynikowego kwaternionu.
- **Operacja normalizacji kwaternionów** - proces normalizacji wyniku przy użyciu wzoru na normalizację kwaternionów, opisanego za pomocą Równania ?? w celu utrzymania rozwiązania w jednostkowej hipersferze.

Należy zwrócić uwagę że sieci kwaternionowe ze względu na swoją strukturę pozwalają na generowanie czterokrotnie więcej informacji z jednego neuronu niż w przypadku standardowych sieci. Główną zaletą kwaternionów w kontekście sieci neuronowych jest ich zdolność do kodowania zależności wielowymiarowych w kompaktowej formie. W przypadku przetwarzania sekwencji wielowymiarowych danych, kwaterniony mogą skutecznie zastąpić oddzielne analizy dla każdego wymiaru, oferując bardziej zbiorczą analizę.

W kontekście QLSTM:

1. **Kompaktowa reprezentacja** - dzięki kwaternionom, jedna aktualizacja wag w QLSTM może jednocześnie oddziaływać na wszystkie cztery składowe kwaternionu. W tradycyjnym LSTM każdy wymiar musiałby być aktualizowany oddziennie.
2. **Mnożenie kwaternionów** - podczas mnożenia kwaternionów uwzględniane są interakcje między składowymi urojonymi. Ta zdolność do uwzględniania wzajemnych interakcji między wieloma wymiarami w jednym kroku sprawia, że QLSTM jest efektywniejszy niż LSTM w przetwarzaniu danych wielowymiarowych.
3. **Zintegrowane funkcje aktywacji** - funkcje aktywacji w QLSTM mogą być zastosowane bezpośrednio do całych kwaternionów, a nie do każdego wymiaru osobno, przyspieszając proces uczenia.

W związku z powyższym, QLSTM może potrzebować znacznie mniej neuronów, aby osiągnąć tę samą pojemność reprezentacyjną co tradycyjne sieci LSTM. Jednak należy zaznaczyć, że mniej neuronów niekoniecznie oznacza mniej parametrów. W QLSTM, chociaż jest mniej neuronów, każdy z nich jest bardziej złożony ze względu na strukturę kwaternionu. Niemniej jednak, efektywność i zdolność do kompaktowego kodowania informacji wielowymiarowej czyni QLSTM atrakcyjną alternatywą dla tradycyjnych sieci LSTM w wielu zastosowaniach, co udowodniła publikacja [23].

3.2.3 Funkcje straty

W opracowanych badaniach wykorzystano dwie funkcje straty, które stanowią podstawę analizy. Pierwsza z nich to funkcja **Błędu średniokwadratowego** (ang. *Mean squared error, MSE*), dostępna w bibliotece PyTorch. Warto podkreślić, że w kontekście uczenia i ewaluacji kwaternionowych sieci neuronowych zalecane jest stosowanie funkcji QMSE zaprezentowanej w Równaniu 3.1, opisanej w pracy *Quaternion neural network* [22]. Elementy funkcji QMSE odpowiadają odpowiednim wartościom danej kwaternionowej, jednakże pod względem implementacji algorytmu od strony technicznej nie różni się ona od operacji MSE na wektorze czterowymiarowym. Z tego powodu postanowiono pozostać przy użyciu podstawowej, wbudowanej funkcji MSE.

$$E = \frac{1}{N} \sum_{n=1}^N [(t_{rpn} - S_{rn}^M)^2 + (t_{ipn} - S_{in}^M)^2 + (t_{jpn} - S_{jn}^M)^2 + (t_{kpn} - S_{kn}^M)^2] \quad (3.1)$$

Wykorzystanie MSE w kontekście oceny poprawności predykcji danych kwaternionowych niesie ze sobą pewne komplikacje. Funkcja błędu średniokwadratowego porównuje odpowiednie wartości na podstawie ich odległości w przestrzeni euklidesowej. Takie podejście może generować błędy w przypadku porównywania danych quaternionowych zważywszy na podwójne pokrycie, którym się cechują. Podwójne pokrycie (ang. *double cover*) w kontekście kwaternionów odnosi się do faktu, iż każda rotacja w trójwymiarowej przestrzeni może być reprezentowana przez dwa przeciwnie skierowane kwaterniony jednostkowe. Dla danej rotacji w przestrzeni 3D, istnieją dwa kwaterniony q oraz $-q$, które reprezentują tę samą rotację. Podwójne pokrycie jest ważnym i użytecznym atrybutem kwaternionów, zwłaszcza w kontekście sferycznej interpolacji liniowej kwaternionów, dlatego należy uwzględnić jej wystąpienie w przypadku generowanych przez sieć danych.

Proponowanych rozwiązaniem tego problemu jest zastosowanie drugiej, autorskiej funkcji straty o nazwie **Funkcja kwaternionowego błędu długości kąta** (ang. *Quaternion angle length error, QALE*), opisanej Równaniem 3.2. Wykonuje ona porównanie danych quaternionowych na podstawie długości kąta zawartego pomiędzy nimi.

$$E = \frac{1}{N} \sum_{n=1}^N [(2 * \arccos(w(q1 \otimes \text{conj}(q2))))^2] \quad (3.2)$$

Kąt zawarty pomiędzy dwoma kwaternionami obliczany jest poprzez mnożenie kwaternionów pomiędzy jednym składnikiem oraz sprzężeniem drugiego, otrzymując wówczas nowy quaternion reprezentujący różnicę pomiędzy dwoma rotacjami. Następnie obliczana jest wartość funkcji \arccos dla części rzeczywistej otrzymanego quaternionu, generując w ten sposób kąt zawarty między dwoma rotacjami na hipersferze jednostkowej. Wynik jest dodatkowo podnoszony do kwadratu by pozbyć się wartości ujemnych oraz zwiększyć wzrost wartości straty dla większych różnic między wartością przewidywaną a rzeczywistą. Ostatecznie wszystkie wyniki poddawane są podstawowej średniej arytmetycznej.

W badaniach obie funkcje są stosowane zarówno w procesie treningu, jak i walidacji, w celu porównania ich efektywności oraz wydajności. Głównym zadaniem jest określenie, czy stosowanie funkcji straty specyficznej dla kwaternionów jest uzasadnione, czy też tradycyjne podejście bazujące na funkcji MSE, często spotykane w literaturze, jest również efektywne, bądź nawet bardziej korzystne.

3.2.4 Parametry treningu

W tym rozdziale omówione zostaną kluczowe parametry oraz techniki zastosowane podczas procesu treningu modelu LSTM oraz QLSTM. Precyzyjna definicja tych parametrów jest niezbędna nie tylko dla głębokiego zrozumienia metodyki badań, ale także dla ewentualnej reprodukcji eksperymentu przez innych badaczy. Parametry te wpływają na efektywność i szybkość uczenia modelu, a także na ostateczną jakość i zdolność generalizacji na nowych danych. Poniżej przedstawione są szczegółowe informacje dotyczące stałych oraz zmiennych hiperparametrów modeli wykorzystanych w przedstawionych badaniach.

Hiperparametry bazowe (stałe):

- **Optymalizator** - wykorzystano algorytm Adam z domyślnymi parametrami, czyli spadkiem wagi równym 0 oraz współczynnikiem beta 0,9.
- **Współczynnik uczenia** - jego wartość wynosi 0,001.
- **Liczba epok** - proces treningowy trwał 25 epok.
- **Rozmiar wsadu (ang. *batch size*)** - modele były trenowane z użyciem sekwencji o długości 100.
- **Techniki augmentacji danych** - dane zostały losowo podzielone na zbiór treningowy oraz walidacyjny, w proporcji 80% : 20%.
- **Warunki sprzętowe** - trening odbywał się na karcie graficznej NVIDIA RTX 3060.

Hiperparametry badawcze (zmienne):

- **Funkcja straty** - wykorzystano błąd średniokwadratowy (MSE) oraz błąd kwaternionowej długości kata (QALE).
- **Źródło danych** - modele były trenowane na zbiorach rotacji pochodzących z nagrani różnych stawów.

Rozdział 4

Badania

4.1 Opis stanowiska badawczego

4.1.1 Wykorzystane narzędzia i motywacja ich wyboru

PyTorch

Wpływ na wybór środowiska PyTorch miała w głównej mierze jego obiektowa składnia oraz fakt, że posiada ona najwyższy poziom abstrakcji spośród popularnych bibliotek do uczenia maszynowego w języku Python. Zdecydował także o nim artykuł *Keras vs Tensorflow vs Pytorch: Key Differences Among Deep Learning* [30], w którym porównywane są różne narzędzia do głębokiego uczenia. Autorzy zaznaczają zalety stosowania Pytorch takie jak: efektywne wykorzystanie pamięci, wsparcie dla równoległości danych, szybkość wykonywania obliczeń ze względu na swoją niskopoziomowość oraz duże wsparcie społeczności. Wstępna implementacja sieci kwaternionowych Titouana Parcolleta, została opracowana w środowisku Pytorch, co również zadecydowało o wyborze.

Wtyczka *Quaternion Neural Networks Plugin*

Wtyczka będąca bezpośrednią implementacją pracy doktoranckiej pt. *Quaternion neural network* [22], której autorem jest Titouan Parcollet. Repozytorium zawiera aktualne implementacje wielu kwaternionowych sieci neuronowych w środowisku PyTorch. Skorzystanie z tego źródła pozwoliło na głębsze zrozumienie działania sieci kwaternionowych oraz przyspieszyło implementację modelu QLSTM.

CUDA

Zastosowanie technologii CUDA umożliwiło znaczące przyspieszenie procesu trenowania modeli. Wykorzystanie GPU, w postaci karty NVIDIA RTX 3060, pozwoliło na efektywny trening sieci na sprzęcie domowym a wsparcie PyTorch dla CUDA uczyniło proces implementacji bardziej płynnym i prostym.

4.1.2 Wybór i uzasadnienie parametrów modeli

Wybór hiperparametrów bazowych

Wartości hiperparametrów bazowych zostały przyjęte z pracy oraz repozytorium autorstwa Titouan Parcollet [22], które przedstawiają je jako optymalne dla większości przypadków. Obejmują one współczynnik uczenia równy 0.001, optymalizator Adam oraz długość sekwencji 100. Stosowane są one zarówno dla LSTM, jak i QLSTM.

Wybór hiperparametrów badawczych

Zarówno tradycyjna metoda uczenia rekurencyjnego, oparta na modelu LSTM, jak i jej kwaterionowa odmiana, QLSTM, są poddawane procesowi treningowemu i ewaluacji przy użyciu dwóch funkcji straty: MSE oraz QALE. Druga z wybranych funkcji jest autorskim rozwiązaniem opisanym za pomocą Równania ???. Służy ona do określania straty między wartościami kwaterionowymi, opierając się na różnicy kątów między nimi. Dodatkowo, zastosowanie QALE eliminuje również błąd wynikający z nieuwzględnienia podwójnego pokrycia kwaterionów przez MSE. Wykorzystanie obu algorytmów, opisanych w poprzednim rozdziale, pozwala na ocenę ich skuteczności i wydajności w kontekście zastosowania kwaterionów w głębokim uczeniu. W analizie uwzględniono kwestię, czy zastosowanie funkcji straty specyficznej dla kwaterionów jest uzasadnione, czy też klasyczne podejście oparte na funkcji MSE, często cytowane w literaturze, oferuje równą, jeśli nie większą efektywność.

4.2 Dane

Źródło danych

Przy wyborze zbioru danych treningowych kluczowym kryterium była konieczność dysponowania zróżnicowanym zestawem danych. Dane te nie mogły stanowić zbioru niezależnych kwaterionów, lecz musiały być sekwencją wartości reprezentujących zmianę orientacji w czasie. Kolejnym aspektem była uniwersalność danych, umożliwiając przy tym zachowanie pewnego wspólnego schematu w całym zbiorze. W kontekście tych wymagań, naturalnym źródłem danych okazał się system przechwytywania ruchu, dostarczający długie sekwencje informacji na temat położenia stawów i ich wzajemnej orientacji w czasie. Decyzję tę umocniła również współpraca z Centrum Badawczo-Rozwojowym w Bytomiu [15], umożliwiając wykorzystanie nagrań z ich bazy danych. Badania koncentrują się na złożoności ruchu różnych stawów, z subtelnych ruchów bioder po bardziej dynamiczne ruchy w stawie kolanowym, celem sprawdzenia przewagi sieci kwaterionowych w zależności od złożoności toru rotacji.

Surowe dane, dostarczone przez PJATK, są przechowywane w formacie C3D. Za pomocą trzech programów opisanych w poprzednim rozdziale następuje proces konwersji tych danych do postaci, która jest możliwa do interpretacji przez sieć. Schemat konwersji jest następujący:

1. **Interpreter plików c3d** - narzędzie to umożliwia odczyt surowych danych z pliku .c3d oraz ich zapis do formatu CSV. Posiada opcję wyboru sekcji danych do zapisu spośród siedmiu dostępnych. W ramach niniejszej pracy wykorzystywany jest jedynie zapis kątów, odpowiadający za reprezentację zmian kątów 27 stawów w formie kątów Cardana. Końcowy plik zawiera tabelę relacji między klatkami nagrania a wartościami orientacji konkretnych stawów w danym momencie.
2. **Konwerter kątów Cardana do kwaternionów** - Mając na uwadze, że praca skupia się na sekwencjach kwaternionów, a pliki .c3d reprezentują rotacje w formie kątów Cardana, konieczne było zastosowanie konwertera. Narzędzie to przetwarza dane zwrócone przez interpreter plików C3D, przekształcając kąty Cardana w kwaterniony. Końcowy plik ma identyczną strukturę jak wynikowy plik interpretera .c3d, lecz dla każdego stawu w danym momencie czasu posiada cztery wartości reprezentujące pojedynczy kwaternion.
3. **Generator zbiorów treningowych** - Aplikacja ta przetwarza plik .csv wygenerowany przez konwerter kwaternionów, przyjmując dodatkowo parametr określający długość sekwencji. Jest on istotny ponieważ dzięki niemu istnieje możliwość generowania kilku różnych zbiorów o różnych długościach sekwencji z tego samego źródła. Umożliwia to generowanie różnych zbiorów z tej samej bazy. Aplikacja wyrina dane używając okna o długości **n**, przy czym zbiór walidacyjny jest określany na podstawie wartości w pozycji **n+1**. Końcowy wynik to dwa pliki .csv gotowe do przetwarzania przez system treningowy i walidacyjny.

Zestawy danych przygotowane według powyższego schematu są następnie przetwarzane przez skrypt odpowiedzialny za trening i validację. Po odczycie plików .csv dane są konwertowane do klasy *Database* w środowisku Pytorch. Następnie dokonuje się ich podziału na zbiory treningowe oraz walidacyjne. Choć rozdzielenie odbywa się w sposób losowy, stosuje się stałe ziarno randomizacji, gwarantujące powtarzalność procesu. Dzięki temu podejściu sekwencje o podobnych charakterystykach mogą zostać rozdzielone, co sprzyja lepszej generalizacji modeli neuronowych, jednocześnie zapewniając spójność zbiorów przy każdorazowym podziale. Następnie zapisywane są w klasie *DataLoader*, która umożliwia podział danych na serie oraz iterację po nich w trakcie treningu.

4.3 Rygor badawczy

4.3.1 Cel badań

Istotą owej pracy jest zweryfikowanie hipotezy, w myśl której zastosowanie kwaternionów w sieciach neuronowych pozwoli na usprawnienie ich wydajności. Badania mają charakter porównawczy, zestawiając ze sobą wyniki sieci LSTM oraz QLSTM. Kryteriami oceny będą miedzy innymi: dokładność przewidywanych danych, średnia wartość straty oraz czas uczenia modelu. Istotą testów jest znalezienie przypadków, w których sieci rekurencyjne oparte na kwaternionach są wydajniejsze od ich standardowych implementacji, biorąc pod uwagę możliwość niewystąpienia takich przypadków. W ramach rygoru badawczego niniejszego projektu wyznaczono pewne konieczne etapy oraz normy postępowania, które miały na celu skuteczne i wiarygodne przetestowanie postawionej hipotezy.

Funkcje straty

Każdy z modeli zostanie wytrenowany za pomocą dwóch funkcji straty: standardowego MSE (ang. *Mean squared error*) oraz autorskiego QALE (ang. *Quaternion angle length error*). Druga z omówionych funkcji posłuży do oceny różnic między wartościami kwaternionowymi, kierując się różnicą kątów między nimi. Dodatkową zaletą jest eliminacja błędów wynikających z nieuwzględnienia podwójnego pokrycia kwaternionów przez MSE. Wykorzystanie obu funkcji, pozwala na ocenę ich skuteczności oraz wydajności w kontekście zastosowania kwaternionów w głębokim uczeniu. Zarazem umożliwiając sprawdzenie hipotezy, w myśl której zastosowanie funkcji straty specyficznej dla kwaternionów zwiększy efektywność bądź zrówna ją do klasycznego podejścia opartego na funkcji MSE.

Zbiory danych

Istotną kwestią było wyłonienie sekwencji zróżnicowanych pod kątem złożoności rotacji kwaternionów w czasie. W tym celu zastosowano dwie metody pozwalające na ocenę ich zróżnicowania. **Współczynnik zmienności zbioru** (ang. *coefficient of variation*, CV) jest miarą relatywnej zmienności zbioru wyrażonej jako procent średniej wartości zestawu danych. Obliczany jest poprzez iloraz odchylenia standardowego σ oraz średniej wartości danych w zestawie μ . Równanie 4.1 jest jej ogólnym wzorem.

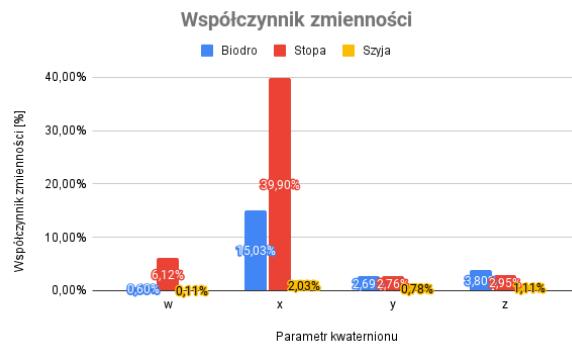
$$CV = \frac{\sigma}{\mu} * 100\% \quad (4.1)$$

Wartości kwaternionów jednostkowych zostały dodatkowo podniesione o wartość 1 w celu przeniesienia wykresu ze zbioru wartości $< -1, 1 >$ do $< 0, 2 >$. Umożliwiło to pozbycie się wartości ujemnych, które uniemożliwiają wykorzystanie współczynnika CV.

Rozstęp interkwartylowy (ang. *interquartile range*, IQR) jest miarą rozproszenia danych jako rozbieżność względem mediany zbioru poprzez różnicę między trzecim Q_3 , a pierwszym kwartylem Q_1 zestawu. Wzór ogólny tej miary opisuje Równanie 4.2.

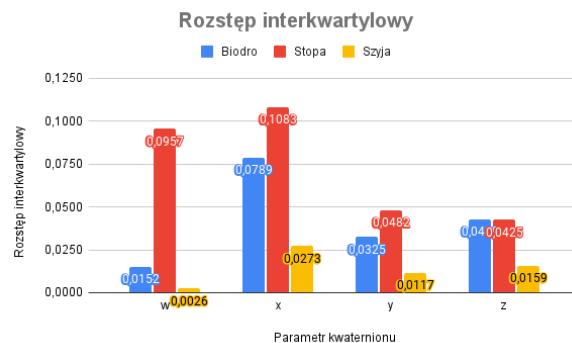
$$IQR = Q_3 - Q_1 \quad (4.2)$$

Bazując na opisanych miarach możliwym było porównanie sekwencji pod względem ich relatywnej zmienności oraz wyłonienie trzech najbardziej zróżnicowanych zbiorów. Współczynnik miary zmienności wybranych zbiorów treningowych pozwolił na zauważenie, że rotacja **stopa** posiada największą wartość CV wynoszącą 12,94%, co czyni ją w tym przypadku zborem o największym zróżnicowaniu danych. Kolejnymi są następująco **szyja** (5,53%) oraz **biodro** (1,01%). Różnice w zmienności poszczególnych parametrów kwaternionu wybranych sekwencji, zostały ukazane za pomocą Rysunku 4.1.



Rysunek 4.1: Współczynnik zmienności zbiorów treningowych

Rozstęp interkwartylowy umożliwił określenie rozbieżność danych względem mediany zbioru. Wykonane obliczenia wykazały, że rotacja **stopa** posiada największą średnią wartość IQR, wynoszącą 0,073, co czyni ją zbiorem o największym rozproszeniu. Kolejnymi zbiorami są **biodro** (0,042) oraz **szyja** (0,014). Rozproszenie poszczególnych parametrów kwaternionu wybranych sekwencji, zostały ukazane za pomocą Rysunku 4.2.

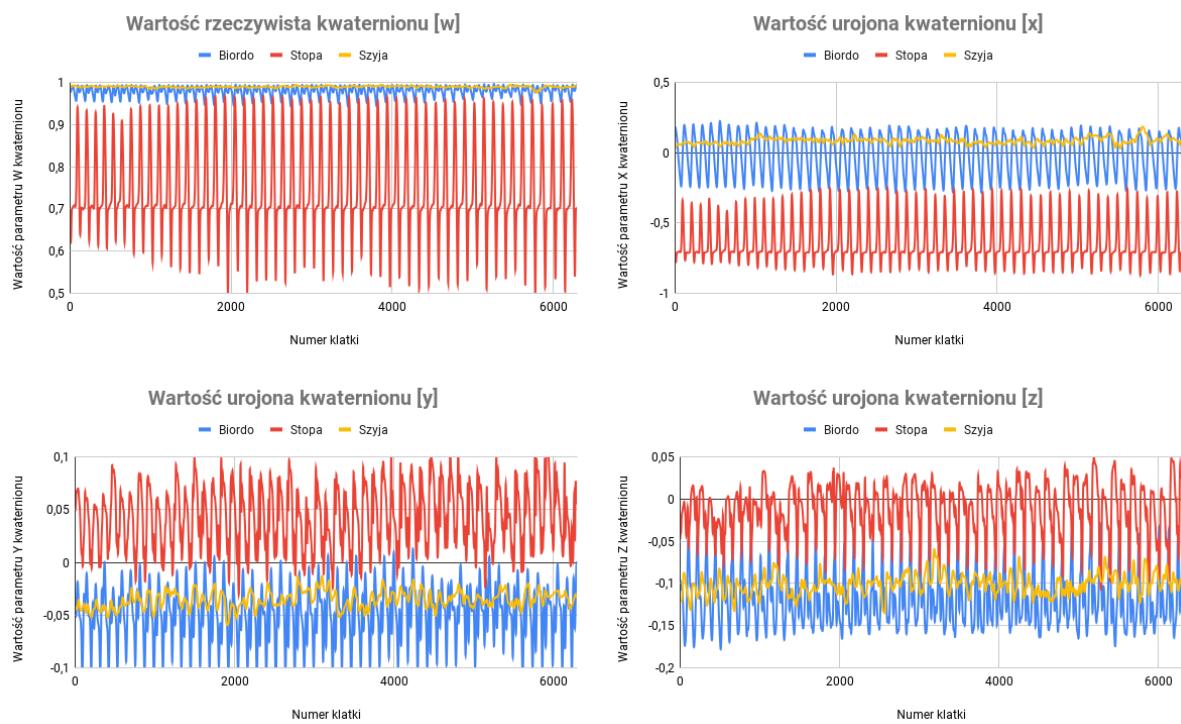


Rysunek 4.2: Rozstęp interkwantylowy zbiorów treningowych

W celu sklasyfikowania wybranych zbiorów oraz wyodrębnienia ich cech, ostatecznie przyjęto zależności wynikające z obu miar. Owe cechy prezentują się następująco:

1. **rotacja biodra** - zbiór o średniej złożoności, ze względu na niską zmienność oraz średnią rozbieżność, zbiór charakteryzuje dynamiczny ruch, o średnim zakresie, wykazujący zarazem cechy okresowości.
2. **rotacja szyi** - najmniej złożony oraz rozproszony zbiór, charakteryzujący się krótkim zakresem ruchu i brakiem gwałtownych zmian kierunku.
3. **rotacja stopy** - najbardziej złożona zestaw, wyróżniający się najwyższą zmiennością oraz rozbieżnością. Najtrudniejszy w analizie poprzez duży zakres ruchu oraz gwałtowne zmiany rotacji.

Zróżnicowanie parametrów kwaterionów poszczególnych rotacji w czasie, z jednego wybranego pliku, został przedstawiony na Rysunku 4.3. Wybór tych zbiorów pozwolił na ocenę wydajności sieci w różnych warunkach. Przy użyciu programów do interpretacji plików c3d utworzone zostały ostateczne, trzy zbiory treningowe.



Rysunek 4.3: Zbiór wykresów opisujących wartości parametrów kwaternionów w czasie

4.3.2 Przebieg procesu badawczego

Całość procesu badawczego obejmowała wytrenowanie łącznie 12 różnych modeli, uwzględniających wszystkie kombinacje funkcji straty oraz zbiorów danych. Każdy z wytrenowanych modeli podlegał szczegółowej ewaluacji opartej na czterech kluczowych kryteriach. Te kryteria zostały wybrane w celu dokładnego i wszechstronnego porównania wydajności modeli LSTM oraz QLSTM:

1. **Wartość średniego błędu funkcji MSE** - dla każdego modelu obliczono wartość średniego błędu, korzystając z funkcji MSE. Pozwoliło to ocenić ogólną skuteczność modelu w przewidywaniu danych.
2. **Wartość średniego błędu funkcji QALE** - wartość średniego błędu była obliczana za pomocą autorskiej funkcji QALE. Umożliwiło to dogłębną analizę różnic w działaniu obu funkcji strat oraz ocenę skuteczności QLSTM, jako metody określającej strategię na podstawie różnicy kątów, w kontekście danych quaternionowych.
3. **Długość procesu uczenia** - czas uczenia był rejestrowany dla każdego modelu, dając możliwość oceny efektywności i wydajności różnych podejść. Krótszy czas uczenia mógłby wskazywać większą optymalizację i skuteczność danego podejścia.

W procesie oceny eksperymentalnej uwzględniono możliwość wystąpienia anomalii lub odstępstw w danych. Analizując uzyskane wyniki, poszukiwano sytuacji, w których sieci oparte na quaternionach miałyby przewagę nad tradycyjnymi podejściami. Jednocześnie brano pod uwagę, że takie przypadki mogą nie wystąpić, co również przyniosłoby istotne informacje na temat skuteczności omawianych metod.

4.4 Wyniki eksperymentów

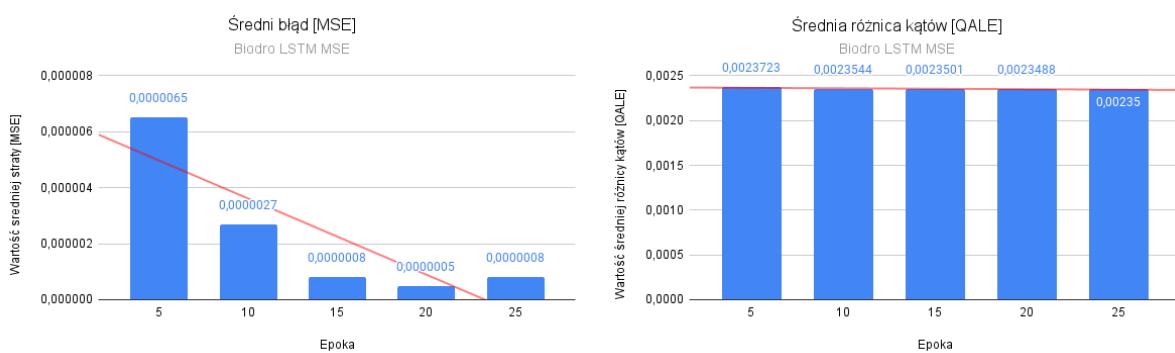
Wyniki eksperymentów zostały podzielone na dwanaście części opisując kolejno każdy z wytrenowanych modeli na podstawie przedstawionych w poprzednim podrozdziale kryteriów. Opracowane wyniki przedstawione są w postaci tabel zawierających stan każdego z kryterium w co piątej epoce.

4.4.1 Biodro LSTM MSE

Tabela 4.1 przedstawia wyniki modelu opartego na architekturze LSTM, trenowanego na zbiorze *biodro* z użyciem funkcji straty MSE. W miarę postępu treningu, średni błąd MSE znaczco malał, z minimalnym wzrostem w dwudziestej piątej epoce, podczas gdy średnia różnica kątów QALE pozostawała stosunkowo stabilna przez cały czas treningu. Wykresy zmian wartości średniego błędu oraz średniej różnicy kątów dla modelu Biodro LSTM MSE zostały przedstawione na Rysunku 4.4.

Tabela 4.1: Wyniki modelu LSTM dla funkcji MSE oraz zbioru *biodro*

Epoka	Średni błąd [MSE]	Średnia różnica kątów [QALE]	Czas Treningu	Czas Treningu [s]
5	0,0000065	0,0023723	5min 38s	338
10	0,0000027	0,0023544	10min 25s	625
15	0,0000008	0,0023501	13min 15s	795
20	0,0000005	0,0023488	16min 03s	963
25	0,0000008	0,0023500	19min 02s	1 142



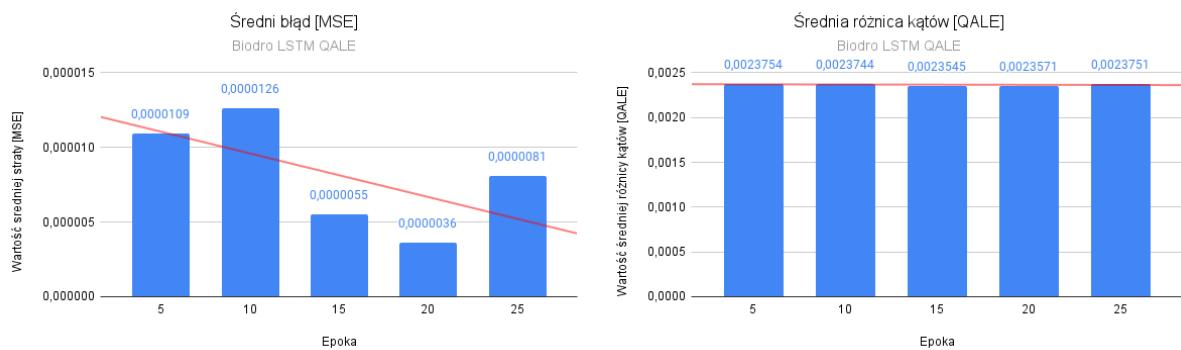
Rysunek 4.4: Wykresy opisujące zmianę MSE i QALE dla modelu Biodro LSTM MSE

4.4.2 Biodro LSTM QALE

Tabela 4.2 prezentuje wyniki dla modelu opartego na architekturze LSTM, trenowanego z użyciem funkcji straty QALE na zbiorze treningowym *biodro*. Mimo początkowego wzrostu wartości, średni błąd MSE wykazywała tendencję malejącą w trakcie procesu treningowego, z wyjątkiem wzrostu obserwowanego w epoce dwudziejstej piątej. Natomiast średnia różnica kątów QALE wskazywała stabilność w trakcie całego procesu, z drobnymi fluktuacjami. Wykresy zmian wartości średniego błędu oraz średniej różnicy kątów dla modelu Biodro LSTM QALE zostały przedstawione na Rysunku 4.5.

Tabela 4.2: Wyniki modelu LSTM, wytrenowanego przy użyciu funkcji błędu średniokwadratowego MSE oraz zbioru *biodro*

Epoka	Średni błąd [MSE]	Średnia różnica kątów [QALE]	Czas Treningu	Czas Treningu [s]
5	0,0000109	0,0023754	06min 35s	395
10	0,0000126	0,0023744	12min 03s	723
15	0,0000055	0,0023545	16min 03s	963
20	0,0000036	0,0023571	19min 49s	1 189
25	0,0000081	0,0023751	23min 47s	1 427



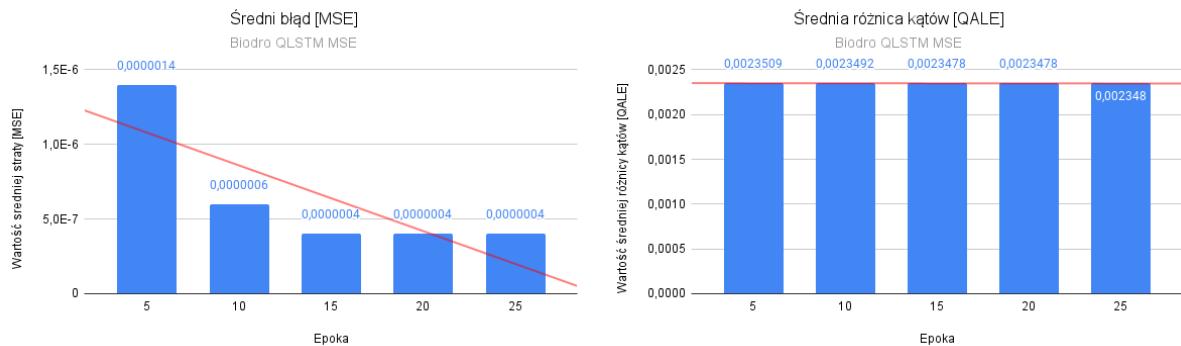
Rysunek 4.5: Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Biodro LSTM QALE

4.4.3 Biodro QLSTM MSE

Tabela 4.3 prezentuje wyniki dla modelu opartego na architekturze QLSTM, trenowanego z użyciem funkcji straty MSE na zbiorze *biodro*. Zaobserwować na niej można znaczący spadek średniego błędu MSE w początkowych epokach treningowych, który ustabilizował się na poziomie 0,0000004 od piętnastej epoki. Równocześnie, średnia różnica kątów QALE pozostawała stabilna na przestrzeni całego procesu treningowego, z nieznacznymi fluktuacjami. Czas treningu znaczaco wzrastał z każdą epoką, osiągając niemal pięćdziesiąt godzin w dwudziestej piątej epoce. Wykresy zmian wartości średniego błędu oraz średniej różnicy kątów dla modelu Biodro QLSTM MSE zostały przedstawione na Rysunku 4.6.

Tabela 4.3: Wyniki modelu QLSTM, wytrenowanego przy użyciu funkcji błędu średniokwadratowego MSE oraz zbioru *biodro*

Epoka	Średni błąd [MSE]	Średnia różnica kątów [QALE]	Czas Treningu	Czas Treningu [s]
5	0,0000014	0,0023509	09h 46min 41s	35 201
10	0,0000006	0,0023492	20h 09min 20s	72 560
15	0,0000004	0,0023478	29h 26min 52s	109 919
20	0,0000004	0,0023478	39h 23min 50s	145 737
25	0,0000004	0,002348	49h 32min 14s	182 241



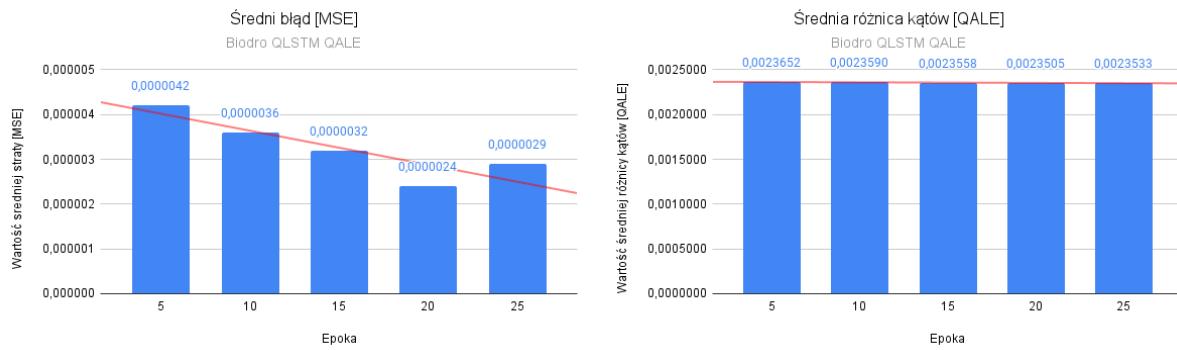
Rysunek 4.6: Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Biodro QLSTM MSE

4.4.4 Biodro QLSTM QALE

Tabela 4.4 ilustruje wyniki dla modelu opartego na architekturze QLSTM, trenowanego z użyciem funkcji straty QALE na zbiorze treningowym *biodro*. Możliwym do zaobserwowania jest generalna tendencja spadkowa średniego błędu MSE w trakcie trwania procesu treningowego, pomimo niewielkiego wzrostu w ostatniej epoce. Analogicznie, średnia różnica kątów QALE wykazała niewielki spadek, stabilizując się w końcowych epokach. Czas treningu rósł w sposób znaczący z każdą epoką, osiągając ponad pięćdziesiąt trzy godziny w dwudziestej piątej epoce. Wykresy zmian wartości średniego błędu oraz średniej różnicy kątów dla modelu Biodro QLSTM QALE zostały przedstawione na Rysunku 4.7.

Tabela 4.4: Wyniki modelu QLSTM, wytrenowanego przy użyciu funkcji QALE oraz zbioru *biodro*

Epoka	Średni błąd [MSE]	Średnia różnica kątów [QALE]	Czas Treningu	Czas Treningu [s]
5	0,0000042	0,0023652	10h 51min 47s	39 107
10	0,0000036	0,0023590	20h 28min 57s	73 737
15	0,0000032	0,0023558	28h 54min 08s	104 048
20	0,0000024	0,0023505	43h 03min 49s	155 029
25	0,0000029	0,0023533	53h 38min 14s	193 094



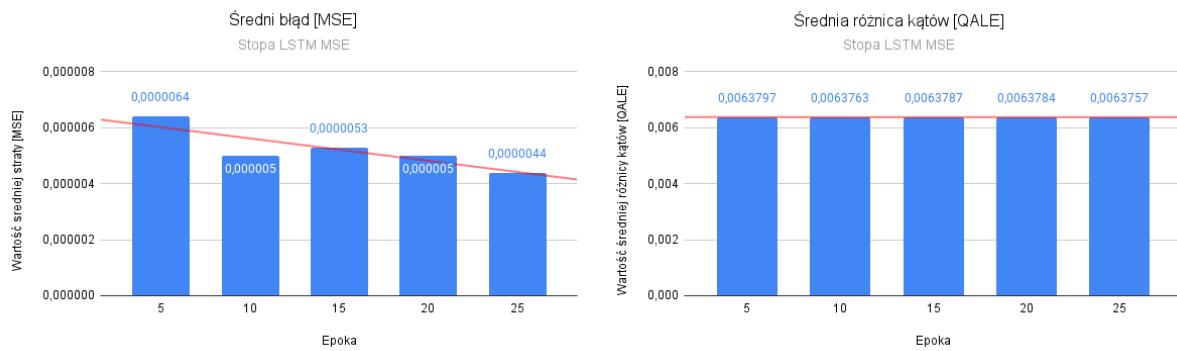
Rysunek 4.7: Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Biodro QLSTM QALE

4.4.5 Stopa LSTM MSE

Tabela 4.5 przedstawia wyniki modelu opartego na sieci LSTM, trenowanego z użyciem funkcji straty MSE na zbiorze treningowym *stopa*. Możliwym do zaobserwowania był niewielki, ale systematyczny spadek średniego błędu MSE w kolejnych epokach, z 0,0000064 w piątej epoce do 0,0000044 w dwudziestej piątej epoce. Jednocześnie, średnia różnica kątów QALE wykazywała niewielkie fluktuacje, zarazem pozostając na podobnym poziomie przez cały czas treningu. On sam zwiększał się liniowo wraz z kolejnymi epokami. Wykresy zmian wartości średniego błędu oraz średniej różnicy kątów dla modelu Stopa LSTM MSE zostały przedstawione na Rysunku 4.8.

Tabela 4.5: Wyniki modelu LSTM, wytrenowanego przy użyciu funkcji MSE oraz zbioru *stopa*

Epoka	Średni błąd [MSE]	Średnia różnica kątów [QALE]	Czas Treningu	Czas Treningu [s]
5	0,0000064	0,0063797	04min 54s	294
10	0,000005	0,0063763	09min 50s	590
15	0,0000053	0,0063787	14min 44s	884
20	0,000005	0,0063784	17min 40s	1 060
25	0,0000044	0,0063757	22min 36s	1 356



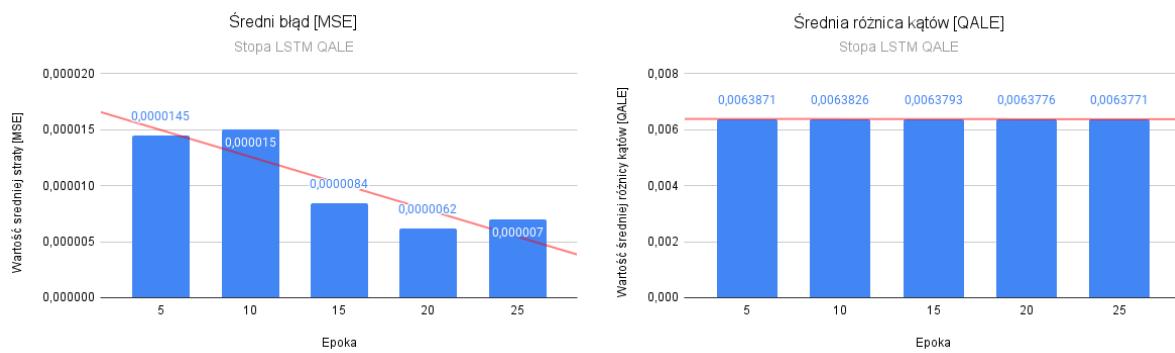
Rysunek 4.8: Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Stopa LSTM MSE

4.4.6 Stopa LSTM QALE

Tabela 4.6 przedstawia wyniki modelu LSTM wytrenowanego z użyciem funkcji QALE na zestawie treningowym *stopa*. Zauważalne były fluktuacje w średnim błędzie MSE w kolejnych epokach, z najwyższą wartością 0,000015 w dziesiątej epoce i najniższą 0,0000062 w dwudziestej epoce. Średnia różnica kątów QALE wykazywała niewielkie różnice w całym procesie treningu, oscylując wokół wartości 0,00638. Czas treningu, podobnie jak w przypadku użycia MSE, zwiększał się liniowo wraz z kolejnymi epokami. Wykresy zmian wartości średniego błędu oraz średniej różnicy kątów dla modelu Stopa LSTM QALE zostały przedstawione na Rysunku 4.9.

Tabela 4.6: Wyniki modelu LSTM, wytrenowanego przy użyciu funkcji QALE oraz zbioru *stopa*

Epoka	Średni błąd [MSE]	Średnia różnica kątów [QALE]	Czas Treningu	Czas Treningu [s]
5	0,0000145	0,0063871	05min 34s	334
10	0,000015	0,0063826	10min 08s	608
15	0,0000084	0,0063793	15min 16s	916
20	0,0000062	0,0063776	20min 04s	1 160
25	0,000007	0,0063771	25min 16s	1 472



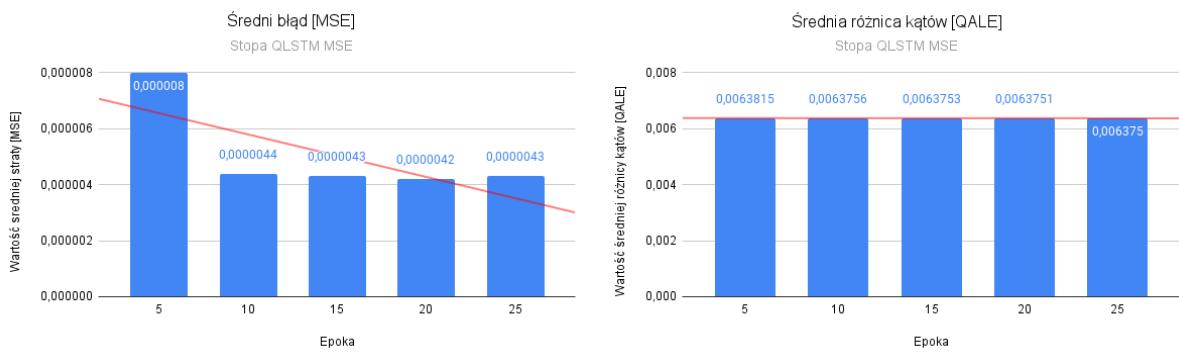
Rysunek 4.9: Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Stopa LSTM QALE

4.4.7 Stopa QLSTM MSE

Tabela 4.7 przedstawia wyniki uzyskane przez model QLSTM wytrenowany przy użyciu funkcji straty MSE na zbiorze danych *stopa*. Średni błąd (MSE) zaczynał się od wartości 0,000008 w epoce piątej i stabilizował się na poziomie około 0,0000043 w epokach od piętnastej do dwudziestej piątej. Średnia różnica kątów (QALE) również wykazywała tendencję do stabilizacji, zaczynając się od wartości 0,0063815 w epoce piątej i osiągając 0,006375 w epoce dwudziestej piątej. Czas treningu rósł liniowo z każdą epoką, zaczynając się od 35326 sekund (9 godzin 48 minut 46 sekund) w epoce piątej i osiągając 178840 sekund (49 godzin 40 minut 40 sekund) w epoce dwudziestej piątej. Wykresy zmian wartości średniego błędu oraz średniej różnicy kątów dla modelu Stopa QLSTM MSE zostały przedstawione na Rysunku 4.10.

Tabela 4.7: Wyniki modelu QLSTM, wytrenowanego przy użyciu funkcji MSE oraz zbioru *stopa*

Epoka	Średni błąd [MSE]	Średnia różnica kątów [QALE]	Czas Treningu	Czas Treningu [s]
5	0,000008	0,0063815	9h 48min 46s	35 326
10	0,0000044	0,0063756	19h 12min 31s	69 151
15	0,0000043	0,0063753	30h 13min 25s	108 805
20	0,0000042	0,0063751	39h 56min 53s	143 813
25	0,0000043	0,006375	49h 40min 40s	178 840



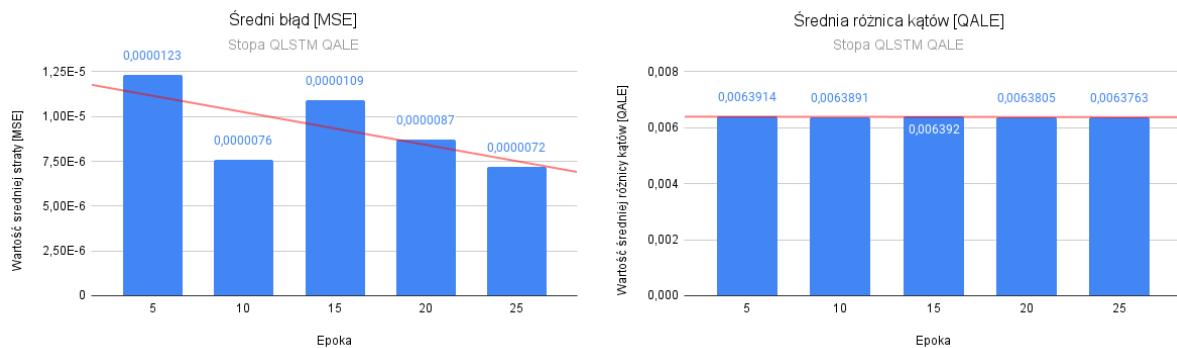
Rysunek 4.10: Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Stopa QLSTM MSE

4.4.8 Stopa QLSTM QALE

Tabela 4.8 przedstawia wyniki modelu QLSTM wytrenowanego przy użyciu funkcji QALE na zbiorze treningowym *stopa*. Zarówno wartości MSE jak i QALE wykazywały pewne fluktuacje w procesie treningu. Średni błąd (MSE) zaczynał się od wartości 0,0000123 w epoce piątej, osiągając minimum 0,0000072 w epoce dwudziestej piątej. Średnia różnica kątów (QALE) wynosiła początkowo wartość 0,0063914 w epoce piątej, by osiągając finalnie 0,0063763 w epoce dwudziestej piątej. Oba kryteria wykazywały nieliniowość zwracanych średnich wartości pomiędzy epokami. Czas treningu rosł liniowo z każdą epoką, zaczynając się od 37957 sekund (10 godzin 32 minuty 37 sekund) w epoce piątej i osiągając 183307 sekund (50 godzin 54 minuty 45 sekund) w epoce dwudziestej piątej. Wykresy zmian wartości średniego błędu oraz średniej różnicy kątów dla modelu Stopa QLSTM QALE zostały przedstawione na Rysunku 4.11.

Tabela 4.8: Wyniki modelu QLSTM, wytrenowanego przy użyciu funkcji QALE oraz zbioru *stopa*

Epoka	Średni błąd [MSE]	Średnia różnica kątów [QALE]	Czas Treningu	Czas Treningu [s]
5	0,0000123	0,0063914	10h 32min 37s	37 957
10	0,0000076	0,0063891	20h 16min 34s	72 994
15	0,0000109	0,0063920	30h 40min 07s	110 407
20	0,0000087	0,0063805	40h 42min 13s	146 555
25	0,0000072	0,0063763	50h 54min 45s	183 307



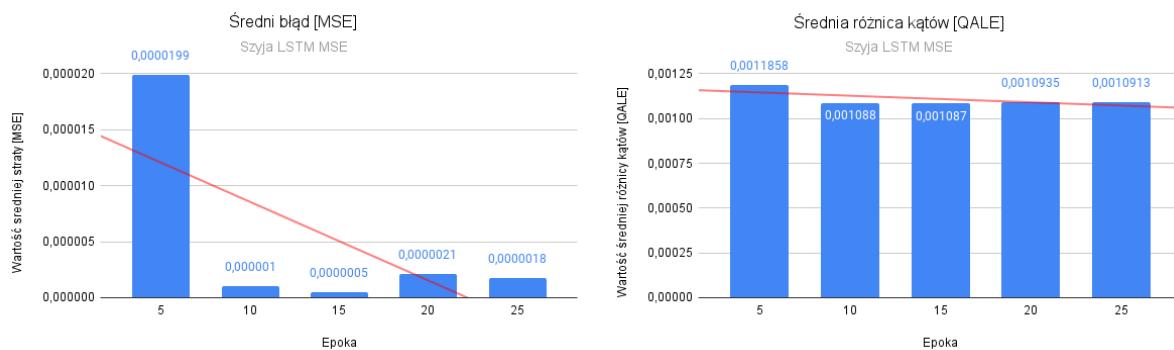
Rysunek 4.11: Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Stopa QLSTM QALE

4.4.9 Szyja LSTM MSE

Tabela 4.9 przedstawia wyniki modelu LSTM wytrenowanego przy użyciu funkcji MSE na zbiorze treningowym *szyja*. Średni błąd MSE osiągnął najniższą wartość 0,0000005 w epoce piętnastej, by następnie wykazywać tendencję wzrostową. Podobna tendencja zaobserwowana była dla średniej różnicy kątów (QALE), która także malała, osiągając najniższą wartość 0,001087 w epoce piętnastej, a następnie nieznacznie wzrasta w kolejnych epokach. Uwagę zwracał również drastyczny spadek średniego błędu pomiędzy piątą a dziesiątą epoką. Wykresy zmian wartości średniego błędu oraz średniej różnicy kątów dla modelu Szyja LSTM MSE zostały przedstawione na Rysunku 4.12.

Tabela 4.9: Wyniki modelu LSTM, wytrenowanego przy użyciu funkcji MSE oraz zbioru *szyja*

Epoka	Średni błąd [MSE]	Średnia różnica kątów [QALE]	Czas Treningu	Czas Treningu [s]
5	0,0000199	0,0011858	04min 32s	272
10	0,0000001	0,001088	07min 06s	426
15	0,0000005	0,001087	09min 48s	588
20	0,0000021	0,0010935	14min 03s	843
25	0,0000018	0,0010913	26min 49s	1 609



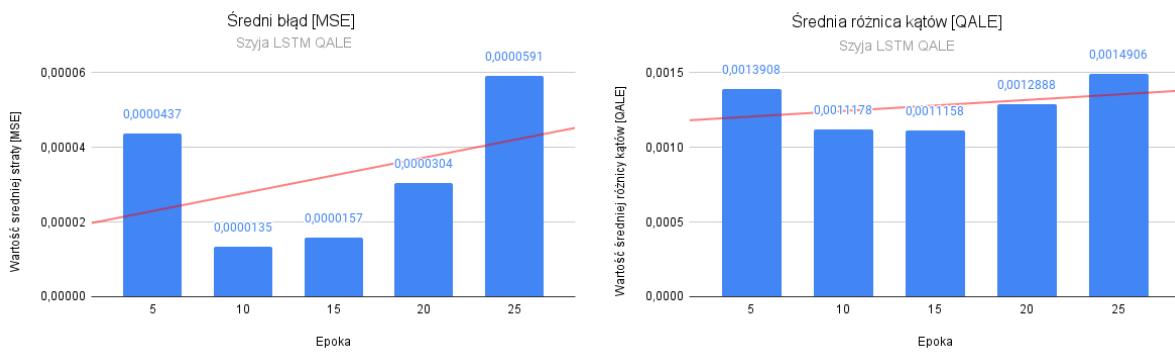
Rysunek 4.12: Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Szyja LSTM MSE

4.4.10 Szyja LSTM QALE

Tabela 4.10 przedstawia wyniki modelu LSTM, wytrenowanego przy użyciu funkcji QALE na zbiorze treningowym *szyja*. Analiza danych wykazuje, że wartość błędu MSE malała do piętnastej epoki, by następnie osiągnąć wzrost tej wartości. Podobna tendencja została zaobserwowana dla średniej różnicy kątów (QALE), gdzie najniższa wartość została osiągnięta w piętnastej epoce, by następnie zaobserwować jej wzrost. Zauważalny był również skok pomiędzy piątą a dziesiątą epoką. Zachowanie sieci LSTM QALE dla zbioru *szyja* posiadało było zbliżone do sieci LSTM MSE, co mogło sugerować iż było ono bezpośrednio powiązane ze specyfiką zestawu treningowego. Wykresy zmian wartości średniego błędu oraz średniej różnicy kątów dla modelu Szyja LSTM QALE zostały przedstawione na Rysunku 4.13.

Tabela 4.10: Wyniki modelu LSTM, wytrenowanego przy użyciu funkcji QALE oraz zbioru *szyja*

Epoka	Średni błąd [MSE]	Średnia różnica kątów [QALE]	Czas Treningu	Czas Treningu [s]
5	0,0000437	0,0013908	05min 48s	348
10	0,0000135	0,0011178	09min 34s	574
15	0,0000157	0,0011158	13min 35s	815
20	0,0000304	0,0012888	20min 50s	1 250
25	0,0000591	0,0014906	34min 42s	2 082



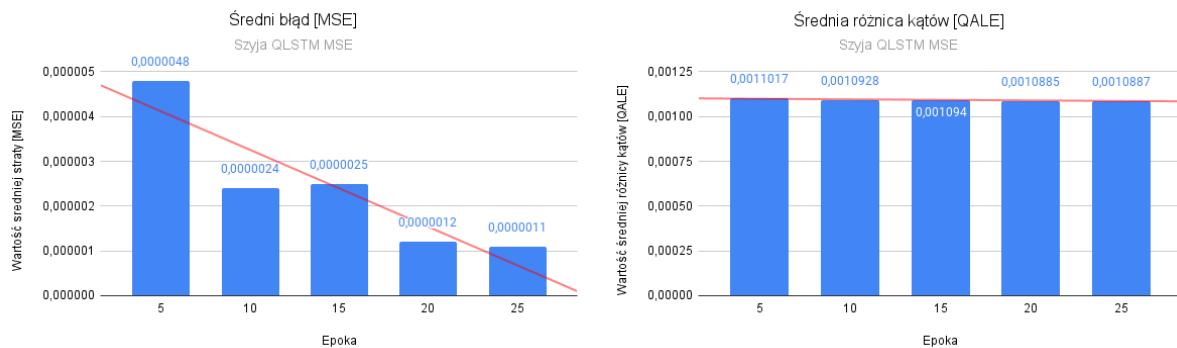
Rysunek 4.13: Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Szyja LSTM QALE

4.4.11 Szyja QLSTM MSE

Tabela 4.11 przedstawia wyniki modelu QLSTM wytrenowanego przy użyciu funkcji MSE na zbiorze treningowym *szyja*. Zarówno wartości MSE jak i QALE wykazywały pewne fluktuacje w procesie treningu. Średni błąd (MSE) zaczynał się od wartości 0,0000048 w epoce piątej, osiągając minimum 0,0000011 w epoce dwudziestej piątej. Średnia różnica kątów (QALE) zaczynała się od wartości 0,0011017 w epoce piątej i osiągała wynik 0,0010887 w epoce dwudziestej piątej. Oba kryteria wykazywały się nieliniowością zwracanych średnich wartości pomiędzy epokami. Wykresy zmian wartości średniego błędu oraz średniej różnicy kątów dla modelu Szyja QLSTM MSE zostały przedstawione na Rysunku 4.14.

Tabela 4.11: Wyniki modelu QLSTM, wytrenowanego przy użyciu funkcji MSE oraz zbioru *szyja*

Epoka	Średni błąd [MSE]	Średnia różnica kątów [QALE]	Czas Treningu	Czas Treningu [s]
5	0,0000048	0,0011017	09h 32min 54s	34 374
10	0,0000024	0,0010928	20h 16min 23s	72 983
15	0,0000025	0,001094	30h 07min 32s	108 452
20	0,0000012	0,0010885	40h 57min 06s	147 426
25	0,0000011	0,0010887	50h 22min 10s	181 330



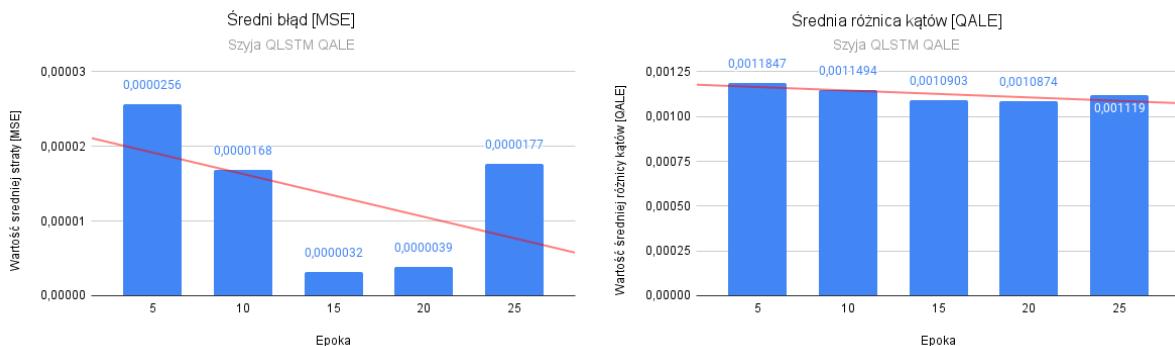
Rysunek 4.14: Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Szyja QLSTM MSE

4.4.12 Szyja QLSTM QALE

Tabela 4.12 przedstawia wyniki modelu QLSTM, wytrenowanego przy użyciu funkcji QALE na zbiorze treningowym *szyja*. Zarówno wartości MSE jak i QALE wykazywały niestabilne zachowanie w procesie treningu. Średni błąd (MSE) rozpoczynał się od wartości 0,0000256 w epoce piątej, osiągając minimum o wartości 0,0000032 w epoce piętnastej, by ponownie odnotować wzrost do 0,0000177 w epoce dwudziestej piątej. Średnia różnica kątów (QALE) malała stopniowo z 0,0011847 w epoce piątej do wartości 0,0010874 w epoce dwudziestej, a ostatecznie osiągnęła wynik 0,001119 w epoce dwudziestej piątej. Oba kryteria wykazywały nieliniowość zwracanych średnich wartości pomiędzy epokami. Czas treningu rósł liniowo z każdą epoką, zaczynając od 36730 sekund (10 godzin 12 minut 10 sekund) w epoce piątej, by osiągnąć 186898 sekund (51 godzin 54 minuty 58 sekund) w epoce dwudziestej piątej. Wykresy zmian wartości średniego błędu oraz średniej różnicy kątów dla modelu Szyja QLSTM QALE zostały przedstawione na Rysunku 4.15.

Tabela 4.12: Wyniki modelu QLSTM, wytrenowanego przy użyciu funkcji QALE oraz zbioru *szyja*

Epoka	Średni błąd [MSE]	Średnia różnica kątów [QALE]	Czas Treningu	Czas Treningu [s]
5	0,0000256	0,0011847	10h 12min 10s	36 730
10	0,0000168	0,0011494	20h 35min 03s	74 103
15	0,0000032	0,0010903	30h 47min 41s	110 861
20	0,0000039	0,0010874	41h 22min 38s	148 958
25	0,0000177	0,001119	51h 54min 58s	186 898



Rysunek 4.15: Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Szyja QLSTM QALE

4.5 Interpretacja wyników

Proces interpretacji wyników uzyskanych w trakcie badań odbywał się etapami, skupiając się na analizie porównawczej modeli z punktu widzenia precyzji wygenerowanych wyników. Etap egzaminacyjny obejmował ocenę modeli w ramach jednego typu sieci, gdzie wyniki były zestawiane przy zastosowaniu różnych funkcji straty podczas uczenia, stopniowo zbliżając się do wniosków o charakterze ogólnym. Ostatecznie porównano precyzję odpowiednich modeli, co pozwoliło zdefiniować ostateczny wniosek. Kryterium umożliwiającym ocenę precyzji danego modelu była średnia różnica kątów pomiędzy kwaternionem wynikowym modelu, a przewidywanym wyjściem, zwracanym przez funkcję QALE. Wyniki funkcji MSE były wykorzystane jedynie w celu przedstawienia procesu treningu sieci. Dodatkowo, podczas oceny wyników, brany pod uwagę był sam czas uczenia danego modelu, co przedstawia Tabela 4.13.

Tabela 4.13: Zestawienie czasu treningu wszystkich modeli poddanych badaniu

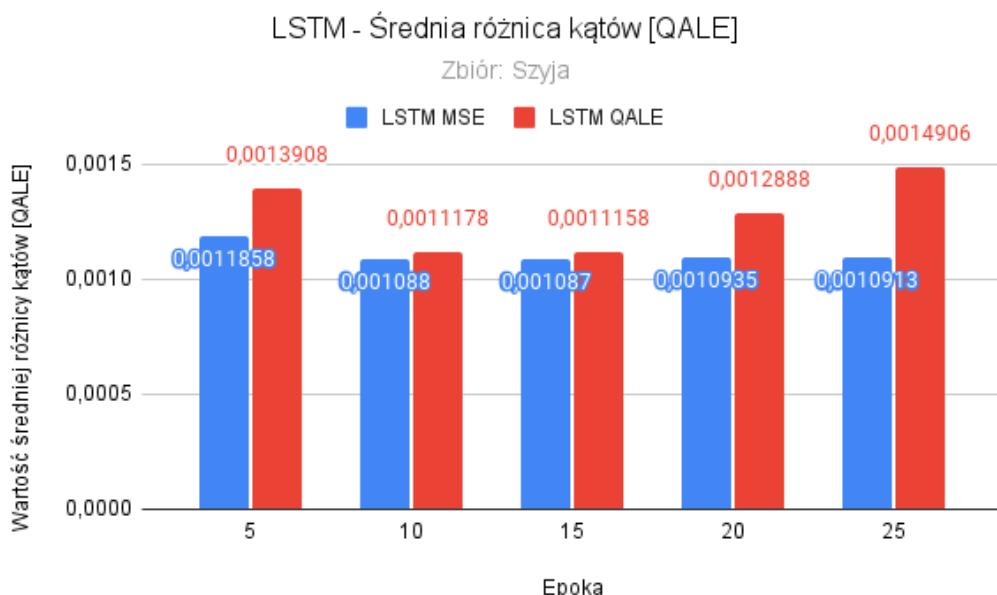
Zestaw danych	Typ sieci	Funkcja straty	Czas uczenia	Czas uczenia [s]
Szyja	LSTM	MSE	26min 49s	1 609
		QALE	34min 42s	2 082
	QLSTM	MSE	50h 22min 10s	181 330
		QALE	51h 54min 58s	186 898
Biodro	LSTM	MSE	19min 02s	1 142
		QALE	23min 47s	1 427
	QLSTM	MSE	49h 32min 14s	182 241
		QALE	53h 38min 14s	193 094
Stopa	LSTM	MSE	22min 36s	1 356
		QALE	25min 16s	1 472
	QLSTM	MSE	49h 40min 40s	178 840
		QALE	50h 54min 45s	183 307

4.5.1 Interpretacja wyników sieci LSTM

Zbiór treningowy Szyja

Porównanie średniej różnicy kątów między modelami LSTM wykorzystujących funkcje MSE bądź QALE dla zbioru treningowego *Szyja* zostało przedstawione za pomocą Rysunku 4.16. W obu przypadkach zauważalne było niestabilne zachowanie, w którym wartość średniej różnicy kątów początkowo malała, by w okolicy piętnastej epoki odnotować wzrost. Model LSTM MSE osiągnął w pierwszej epoce wartość 0,0011858, utrzymując stabilność poprzez odnotowywanie zbliżonych wyników w kolejnych krokach. Minimum wyniosło zaledwie 0,001087. W dwudziestej epoce nastąpił niewielki wzrost do 0,0010935, osiągając maksimum, by w dwudziestej piątej epoce wartość ta nieznacznie spadała do 0,0010913. W przypadku modelu LSTM QALE średnia różnica kątów wyniosła początkowo wartość 0,0013908 w piątej epoce, by odnotować spadek do 0,0011178 w dziesiątej epoce i utrzymywać się na tym poziomie do piętnastej epoki. Następnie, w dwudziestej epoce, wartość ta wzrosła do 0,0012888, osiągając 0,0014906 w dwudziestej piątej.

Należy podkreślić, że mimo podobnego trendu, końcowy skok wartości w przypadku LSTM MSE jest znacznie mniejszy niż w przypadku LSTM QALE. Dla MSE wartość w ostatniej epoce jest większa o 0,0000043 od wartości minimalnej, podczas gdy dla QALE różnica ta wynosi 0,0003748. Dodatkowo, wartość końcowa QALE jest większa o 0,0000998 od wartości początkowej, co może wskazywać na przeuczenie sieci. Podsumowując, dla zestawu treningowego *Szyja*, sieć LSTM MSE osiągnęła lepsze wyniki końcowe i wykazała stabilniejszy trend uczenia w porównaniu z siecią LSTM QALE.

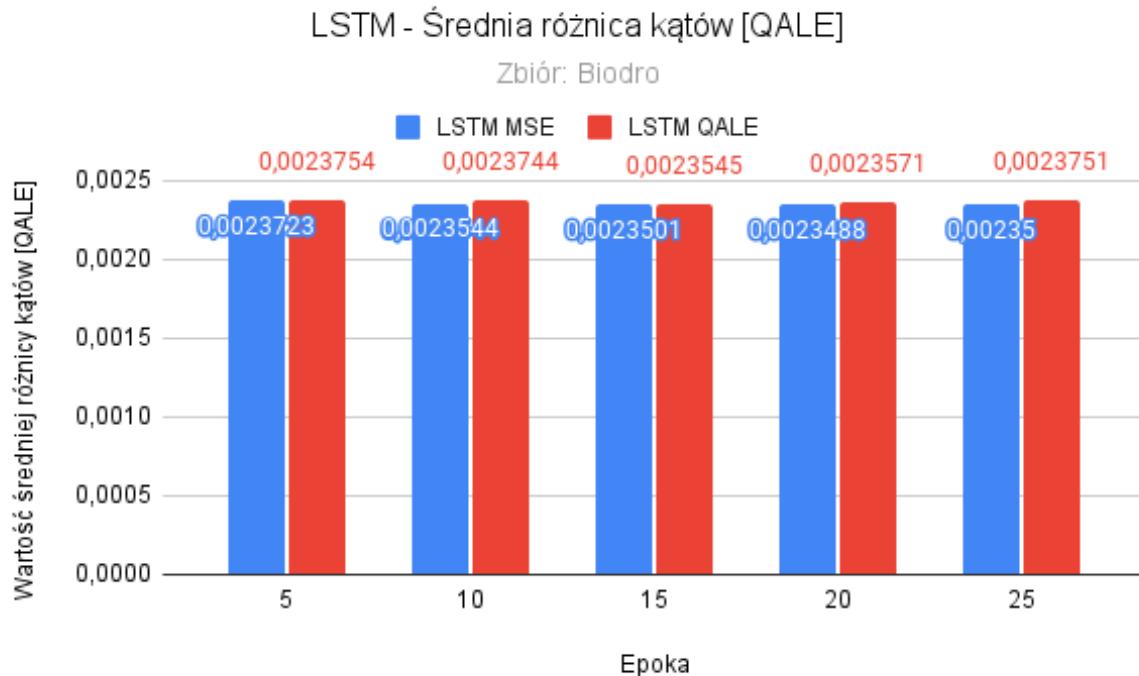


Rysunek 4.16: Porównanie zmian średniej różnicy kątów dla modelu Szyja LSTM MSE oraz Szyja LSTM QALE

Zbiór treningowy Biodro

Analiza wykresu porównawczego przedstawionego na Rysunku 4.17 wykazuje, że wartości średniej różnicy kątów dla modeli LSTM MSE i LSTM QALE cechują się niewielkimi różnicami w trakcie kolejnych epok. Dla modelu LSTM MSE wartość początkowa wynosiła 0,0023723, by następnie marginalnie zmaleć do 0,0023544 w dziesiątej epoce. Ta wartość kontynuowała swój niewielki spadek, osiągając wartość minimalną 0,0023488 w dwudziestej piątej epoce. Ostatecznie, w dwudziestej piątej epoce, wartość ta nieznacznie wzrosła do 0,0023500. Z kolei dla modelu LSTM QALE wartość początkowa wyniosła 0,0023754, nieznacznie malejąc do wartości minimalnej 0,0023545 w piętnastej epoce. Wartość ta wzrosła do 0,0023751 w dwudziestej piątej epoce.

Średnie wartości różnicy kątów dla modelu LSTM MSE wykazywały bardzo niewielkie fluktuacje przez cały okres treningowy, z minimalnym spadkiem do dwudziestej epoki i nieznacznym wzrostem w dwudziestej piątej epoce. Dla modelu LSTM QALE wartości były nieco bardziej zróżnicowane, z niewielkim spadkiem od piątej do piętnastej epoki, a następnie niewielkim wzrostem w dwudziestej i dwudziestej piątej epoce. Podsumowując, oba modele wykazywały stabilność, w obliczu kryterium QALE, przez cały proces treningowy. Różnice w wartościach między oboma modelami były minimalne, co sugerowało, że wybór funkcji straty nie miał znaczącego wpływu na wyniki w kontekście tego eksperymentu.

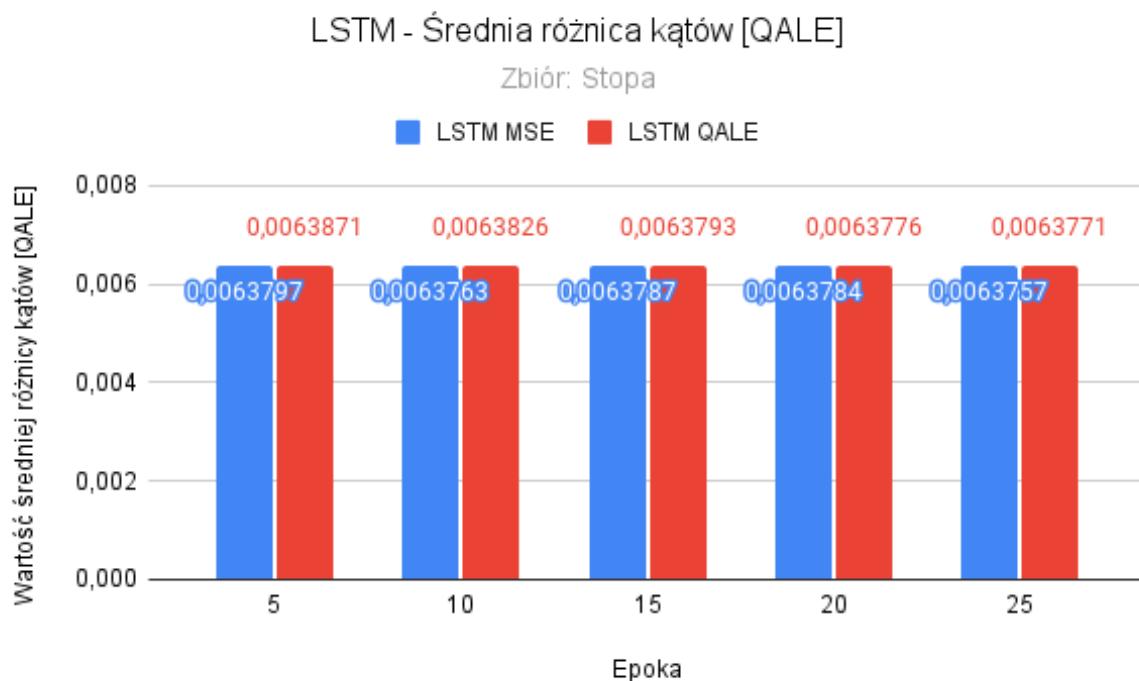


Rysunek 4.17: Porównanie zmian średniej różnicy kątów dla modelu Biodro LSTM MSE oraz Biodro LSTM QALE

Zbiór treningowy Stopa

Rysunek 4.18 przedstawiał porównanie średniej różnicy kątów dla modeli LSTM MSE i LSTM QALE, trenowanych na zestawie danych *Stopa*. Zauważalnym było, że wartości te różniły się marginalnie w poszczególnych epokach, jednak całościowo prezentowały bardzo zbliżone wartości. Dla modelu LSTM MSE wartość początkowa wynosiła 0,0063797 w piątej epoce, a następnie konsekwentnie malała, osiągając minimalną wartość 0,0063757 w dwudziestej piątej epoce. Z kolei dla modelu LSTM QALE, wartość początkowa w piątej epoce wyniosła 0,0063871, która następnie nieznacznie malała, osiągając wartość 0,0063771 w dwudziestej piątej epoce.

Wartości średniej różnicy kątów danych dla obu modeli, LSTM MSE i LSTM QALE, charakteryzowały się minimalnymi fluktuacjami oraz ogólnym trendem spadkowym w trakcie całego procesu treningowego. Różnice w wartościach między modelami były nieistotne, co wskazywało na to, że funkcje straty nie miały znaczącego wpływu na wyniki w kontekście tego kryterium. Podsumowując, oba modele wykazują stabilność w kontekście analizowanego kryterium przez cały proces treningowy.

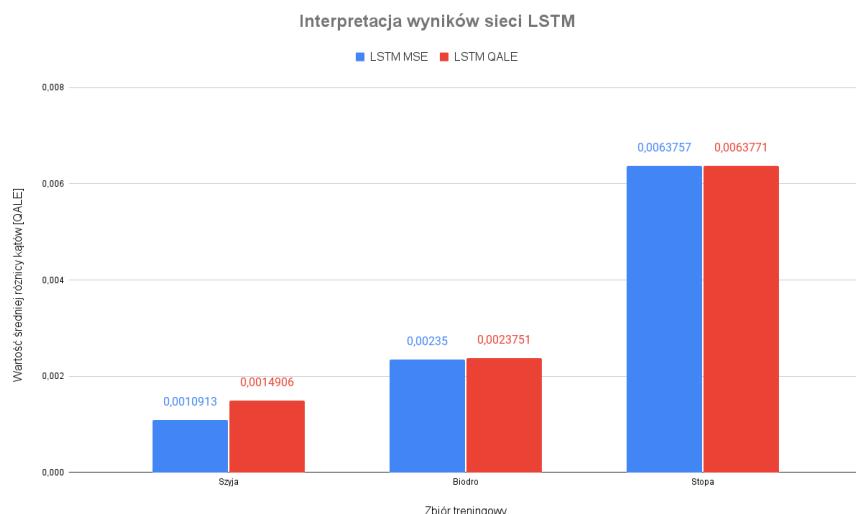


Rysunek 4.18: Porównanie zmian średniej różnicy kątów dla modelu Stopa LSTM MSE oraz Stopa LSTM QALE

Wnioski

Biorąc pod uwagę powyższą analizę, możliwym jest wysunięcie wniosku, że sieci LSTM, niezależnie od zastosowanej funkcji straty w trakcie procesu treningowego, wykazują minimalne fluktuacje i stabilny trend spadkowy w przypadku problemów regresyjnych z wykorzystaniem kwaterionów, co zostało przedstawione na Rysunku 4.19. Wyjątek stanowił zbiór danych *Szyja*, charakteryzujący się większymi niestabilnościami oraz zauważalnym skokiem wartości pod koniec procesu treningowego. Istnieje duże prawdopodobieństwo, że obserwowane zachowanie sieci było wywołane specyficznymi cechami zestawu, takimi jak ograniczony zakres rotacji. Obie sieci wykazywały bardzo podobne zachowania, a obserwowane nieregularności nie wystąpiły w innych przypadkach zbiorów. Pomimo problemów z końcowym skokiem wartości w przypadku tego zestawu, model LSTM MSE uzyskał wyższą precyzję. Ponadto, zaobserwowano, że wyniki średniej straty były tym niższe, im łatwiejszy był analizowany zbiór, co było rezultatem przewidywanym i oczekiwany.

Oba modele wykazywały stabilność w kontekście kryterium QALE przez cały proces treningowy. Różnice w wartościach między modelami były minimalne, wykazując, że wybór funkcji straty nie miał znaczącego wpływu na wyniki tego eksperymentu. Jednakże, zaobserwowane problemy w przypadku zbioru *Szyja* sugerują, że funkcja straty MSE jest mniej podatna na specyficzne cechy zestawu danych, co potwierdza znacznie większa stabilność modelu LSTM MSE w porównaniu z LSTM QALE w analizowanym eksperymencie. W konsekwencji, w oparciu o przedstawione wyniki i argumenty, można wnioskować, że MSE jest lepszą funkcją straty dla sieci LSTM dla problemów regresyjnych operujących na kwaterionach. Należy także wspomnieć, iż proces trenowania sieci LSTM do 25 epoki był stosunkowo krótki i trwał nie więcej niż 35 minut, z minimalnym zaobserwowanym czasem 19 minut.



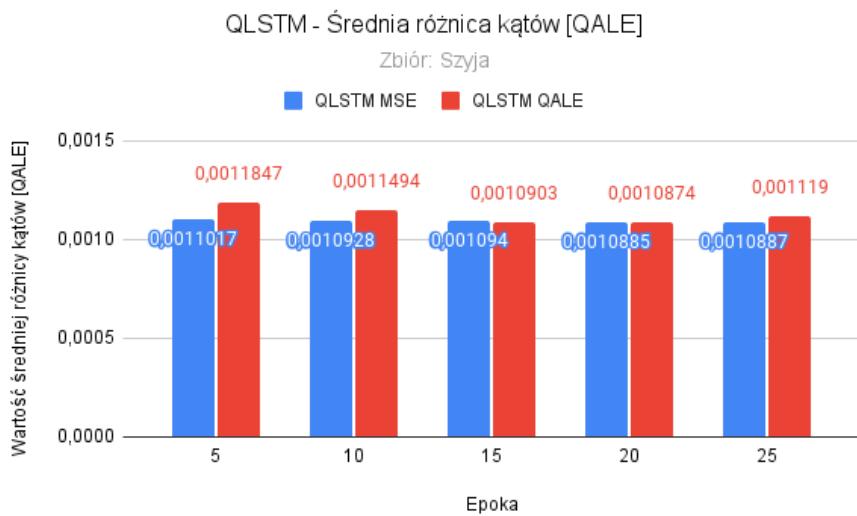
Rysunek 4.19: Porównanie zmian średniej różnicy kątów dla modeli LSTM MSE oraz LSTM QALE

4.5.2 Interpretacja wyników sieci QLSTM

Zbiór treningowy Szyja

Analiza zaprezentowana na Rysunku 4.21 wykazała, że modele QLSTM MSE i QLSTM QALE, trenowane na zbiorze *Szyja*, charakteryzują się zbliżonymi wartościami średniej różnicy kątów w trakcie kolejnych epok procesu treningu sieci. Dla modelu QLSTM MSE, początkowa wartość średniej różnicy wynosiła 0,0011017 i ulega stopniowej minimalizacji do wartości 0,0010885 w epoce dwudziestej. Następnie utrzymała stały trend do końca treningu. Podobne zachowanie zaobserwowano dla modelu QLSTM QALE, którego średnia różnica kątów w epoce piątej wynosła 0,0011847. Wartość ta następnie zmalała do 0,0010874 w epoce dwudziestej gdzie osiągnęła minimum, by ostatecznie nieznacznie wzrosła do 0,001119 w epoce dwudziestej piątej.

Oba modele wykazywały ogólny trend spadkowy wartości średniej różnicy kątów na przestrzeni kolejnych epok treningowych. Zaobserwowana niestabilność była minimalna, co sugerowało, że oba modele charakteryzowały się stałością w kontekście analizowanego kryterium. Ponownie zauważono fluktuacje oraz skok wartości pod koniec procesu treningowego dla zestawu danych *Szyja*, co dodatkowo potwierdziło hipotezę, że obserwowane zjawisko było związane z indywidualnymi cechami tego konkretnego zestawu danych. Różnice w wartościach między analizowanymi modelami były niewielkie, jednak model QLSTM MSE wykazywał większą stabilność w porównaniu z modelem QLSTM QALE, który charakteryzował się większymi wartości między kolejnymi epokami. Pomimo tego, iż oba modele wykazywały stabilny spadek wartości średniego błędu oraz zbliżone różnice pomiędzy kolejnymi epokami dla zestawu *Szyja*, to ostatecznie sieć QLSTM MSE osiągnęła lepsze wyniki końcowe wykazując korzystniejszy trend uczenia.

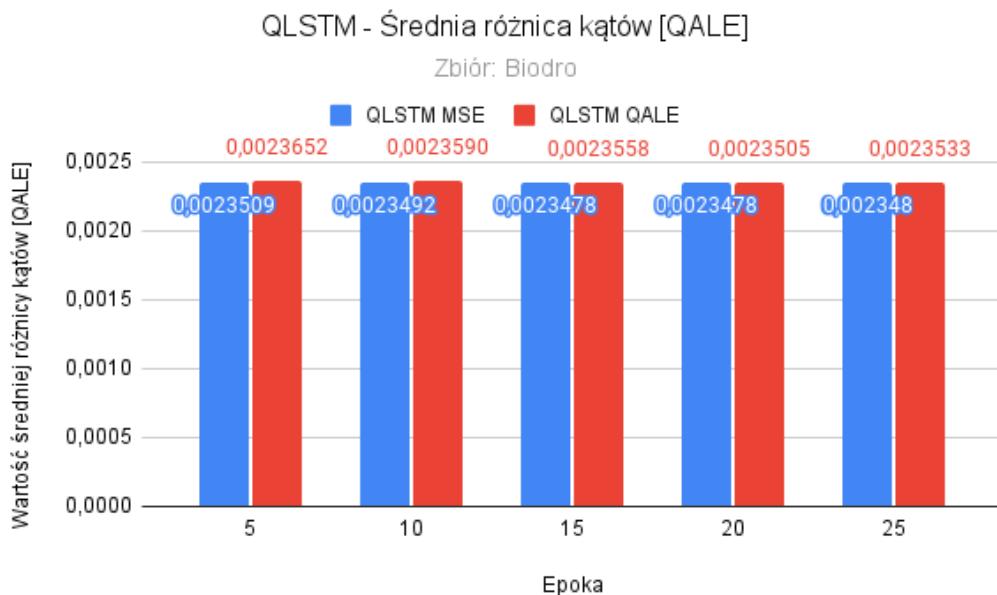


Rysunek 4.20: Porównanie zmian średniej różnicy kątów dla modelu Szyja QLSTM MSE oraz Szyja QLSTM QALE

Zbiór treningowy Biodro

Przedstawiona na Rysunku 4.21 analiza porównawcza wykazała, że modele QLSTM MSE oraz QLSTM QALE, wytrenowane na zbiorze *Biodro*, cechowały się zbliżonym trendem zmian wartości średniej wartości różnicy kątów. Dla QLSTM MSE, wartość ta początkowo wynosiła 0,0023509, stopniowo malejąc do wartości minimalnej 0,0023478, którą utrzymała w przedziale epoki piętnastej oraz dwudziestej. Na końcu treningu wartość ta nieznacznie wzrosła do 0,002348. Z kolei dla modelu QLSTM QALE, wartość początkowa wynosiła 0,0023652, systematycznie malejąc, by osiągnąć minimum 0,0023505 w epoce dwudziestej. Ostatecznie wzrosła do 0,0023533 w dwudziestej piątej epoce.

Oba modele, QLSTM MSE i QLSTM QALE, prezentują ogólny trend spadkowy wartości średniej różnicy kątów danych w trakcie kolejnych epok, z kilkoma niewielkimi fluktuacjami. Są to jednak różnice minimalne, co sugeruje stabilność obu modeli w odniesieniu do tego kryterium. Pomimo niewielkich różnic w wartościach pomiędzy oboma modelami, ostatecznie QLSTM MSE wykazuje marginalnie większą stabilność w porównaniu z QLSTM QALE, który cechuje się większymi niestabilnościami między kolejnymi epokami i wyższą wartością końcową straty. W świetle przedstawionej argumentacji, można wysunąć wniosek, że różnice w funkcjach straty MSE i QALE nie miały istotnego wpływu na różnice w wynikach obu modeli. Ostatecznie, oba modele wydają się być bardzo efektywne i stabilne, z niewielkimi różnicami w wartościach średniej różnicy kątów danych na przestrzeni epok, z nieznaczco korzystniejszym wynikiem dla sieci QLSTM MSE.

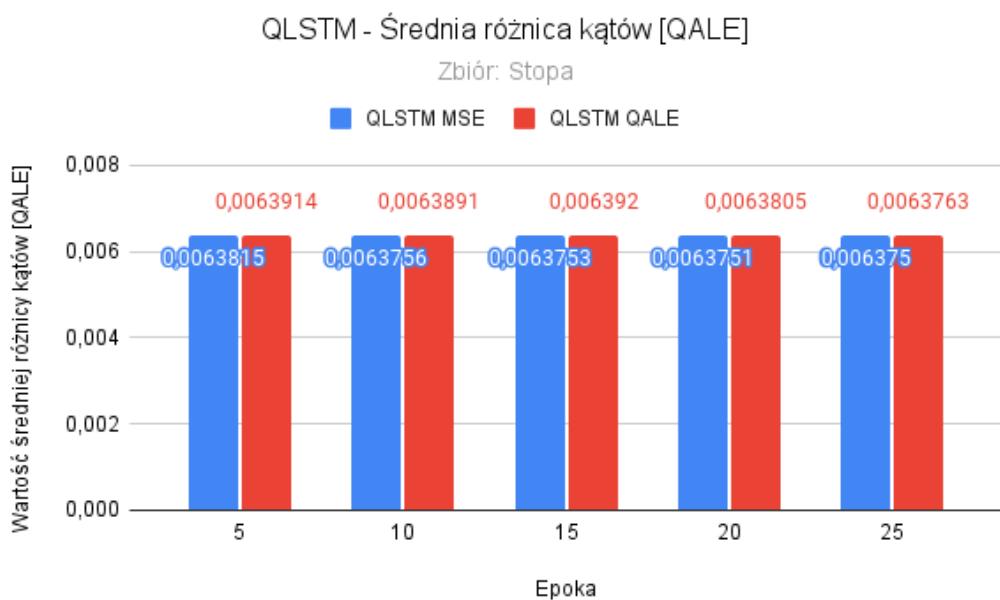


Rysunek 4.21: Porównanie zmian średniej różnicy kątów dla modelu Biodro QLSTM MSE oraz Biodro QLSTM QALE

Zbiór treningowy Stopa

Analiza przedstawiona na Rysunku 4.22 pozwala na zaobserwowanie, że modele QLSTM MSE oraz QLSTM QALE, trenowane na zbiorze *Stopa*, wykazują podobny trend zmian wartości średniej różnicy kątów. Dla modelu QLSTM MSE, wartość ta zaczynała się od 0,0063815 i doznawała niewielkich zmian w kolejnych epokach, stabilizując się na poziomie około 0,006375 w epokach dwudziestej i dwudziestej piątej. Z kolei dla modelu QLSTM QALE, początkowa wartość w epoce piątej wyniosła 0,0063914. Malejąc następnie do wartości 0,006376 w epoce dwudziestej piątej, co wykazuje marginalną nie-stabilność podczas treningu.

Oba modele, QLSTM MSE i QLSTM QALE, charakteryzowały się ogólnym trendem spadkowym wartości średniej różnicy kątów na przestrzeni kolejnych epok, z kilkoma drobnymi fluktuacjami. Te niewielkie różnice sugerowały stabilność obu modeli w kontekście tego kryterium. Choć różnice w wartościach między oboma modelami były niewielkie, model QLSTM QALE wykazywał nieco większe skoki między kolejnymi epokami w porównaniu z QLSTM MSE. Bazując na tych wnioskach, da się zauważyc, że różnice w funkcjach straty MSE oraz QALE nie miały istotnego wpływu na różnice w wynikach obu modeli. Ostatecznie, oba modele zdawały się być wysoce efektywne i stabilne, z minimalnymi różnicami w wartościach średniej różnicy kątów danych na przestrzeni epok.

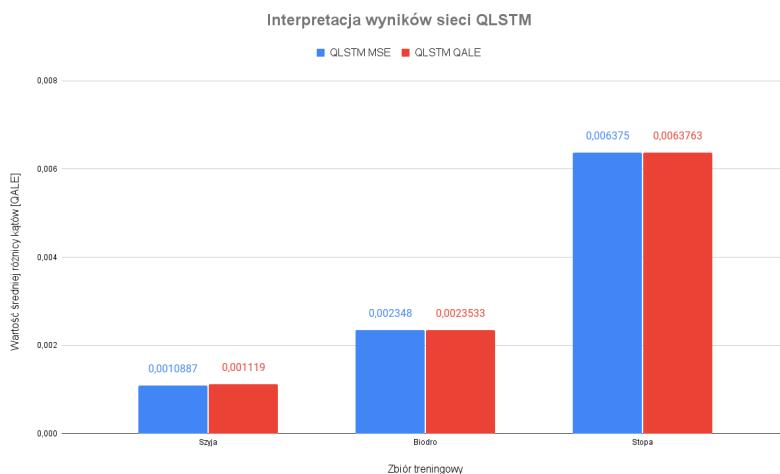


Rysunek 4.22: Porównanie zmian średniej różnicy kątów dla modelu Stopa QLSTM MSE oraz Stopa QLSTM QALE

Wnioski

Z analizy wynika, że sieci QLSTM, niezależnie od zastosowanej funkcji straty w trakcie procesu treningowego, wykazywały niewielkie fluktuacje i generalny trend spadkowy w przypadku problemów regresyjnych z wykorzystaniem kwaterionów, co zostało zilustrowane na Rysunku 4.23. Wyjątkiem był ponownie zbiór danych *Szyja*, który wykazywał większe niestabilności oraz zauważalny skok wartości na końcu procesu treningowego. Pomimo tego, że fluktuacje modeli QLSTM trenowanych na zbiorze *Szyja* były zdecydowanie mniejsze od tych zaobserwowanych dla LSTM, umacnia to hipotezę o specyfice tego właśnie zbioru. Wskazuje na to również fakt, że obie sieci wykazywały bardzo podobne zachowania wyłącznie dla tego zestawu danych, podczas gdy obserwowane nieregularności nie wystąpiły w innych przypadkach. Pomimo problemów z końcowym skokiem wartości dla zbioru *Szyja*, model QLSTM MSE osiągnął nieznacznie wyższą precyzję, różniącą się od QALE o 0,000083. Dodatkowo zauważono, że wyniki średniej straty były tym niższe, im prostszy był analizowany zbiór danych, co było rezultatem przewidywanym i oczekiwany. W pozostałych zbiorach treningowych, różnice pomiędzy wynikami odpowiednich funkcji były marginalne, sięgając przyblizonego błędu o wartości 0,000053.

Oba modele wykazały stabilność w kontekście kryterium QALE przez większość procesu treningowego. Różnice w wartościach między modelami były minimalne, co sugeruje, że wybór funkcji straty nie miał znaczącego wpływu na wyniki tego eksperymentu. Obserwowane problemy w przypadku zbioru *Szyja* wykazują, że funkcja straty MSE jest mniej wrażliwa na specyficzne właściwości zestawu danych, jednak różnica błędu jest w tym przypadku marginalna. Nie wykluczone, że błąd ten mógłby się pogłębiać w kolejnych epokach. W rezultacie, na podstawie przedstawionych wyników i argumentów, można wnioskować, że wybór funkcji straty nie miał znaczącego wpływu na wyniki w kontekście użycia modeli QLSTM dla problemów regresyjnych operujących na kwaterionach.

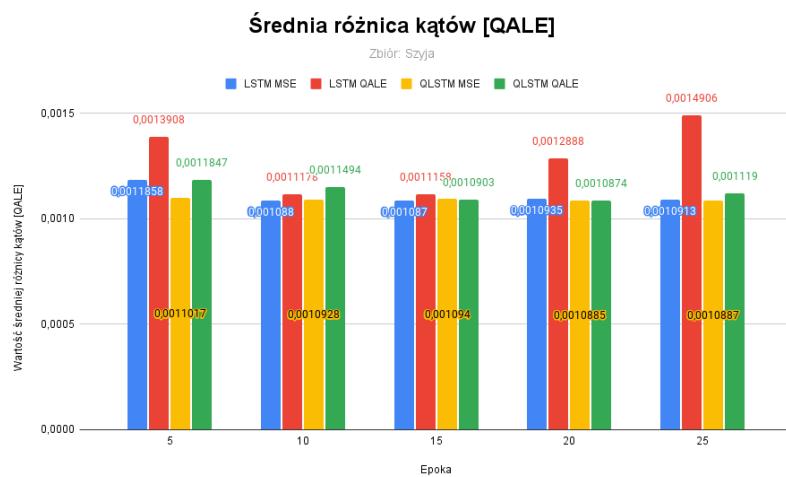


Rysunek 4.23: Porównanie zmian średniej różnicy kątów dla modeli QLSTM MSE oraz QLSTM QALE

Należy również podkreślić czas treningu analizowanych sieci QLSTM, który wahały się między 49 a 53 godzinami. Jest to wynik prawie stukrotnie gorszy niż w przypadku modeli LSTM, jednak istnieje wysokie prawdopodobieństwo, że jest to spowodowane implementacją sieci bezpośrednio w języku Python. Zoptymalizowane sieci LSTM zostały opracowane w języku Lua, dostępnym poprzez platformę Pytorch.

4.6 Wnioski z badań

Na podstawie analizy oraz interpretacji wyników badań, można wywnioskować, że zarówno dla modeli LSTM, jak i QLSTM, użycie funkcji straty MSE w procesie uczenia prowadziło do osiągnięcia lepszych rezultatów. Dla modeli QLSTM różnice te były marginalne, z największym zaobserwowanym błędem wynoszącym 0,000083 dla zbioru *Szyja*. Tymczasem, analiza modeli LSTM wykazała większe różnice, z najgorszym przypadkiem, w którym błąd wynosił 0,0003993, co jest prawie pięciokrotnie większą wartością błędu w porównaniu z modelami QLSTM. Równie ważnym jest zauważenie, że omawiana anomalia wystąpiła w obu przypadkach podczas eksperymentu na zbiorze *Szyja*. Rozkład wyników funkcji straty na tym zbiorze dla wszystkich modeli jest przedstawiony na Rysunku 4.24.

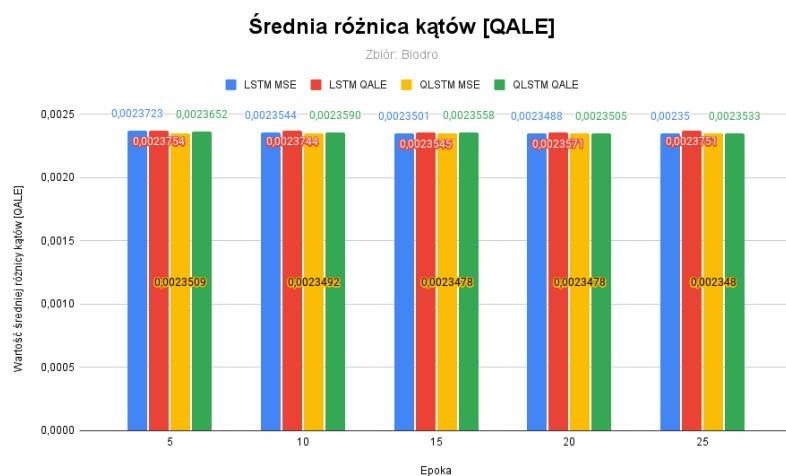


Rysunek 4.24: Porównanie zmian średniej różnicy kątów dla zbioru Szyja

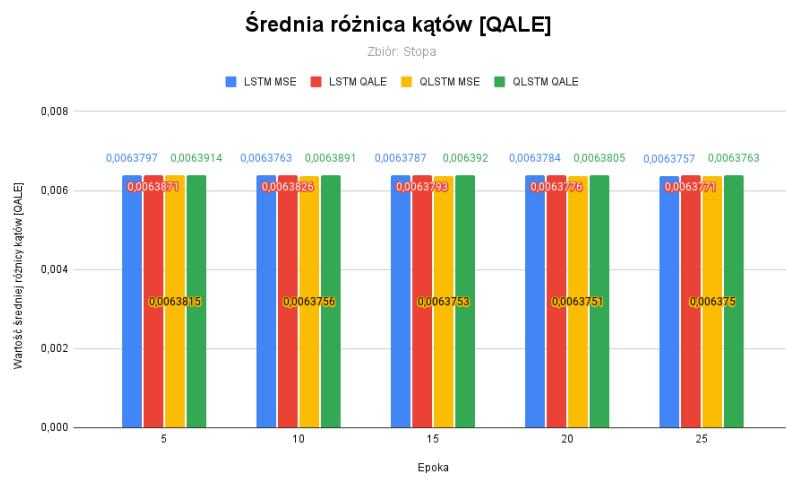
Sieci QLSTM były mniej podatne na anomalię w przypadku operowania na danych kwaternionowych. Analizując modele trenowane na zbiorze *Szyja*, który wykazywał większą niestabilność niż pozostałe zbiory, modele QLSTM cechowały się prawie trzykrotną różnicą w ostatecznym skoku pomiędzy funkcjami MSE (0,0000303) i QALE (0,0003993). Dodatkowo, wartość LSTM QALE dla omawianego skoku osiągnęła pod koniec uczenia wartość wyższą niż na początku, co nie zdarzyło się w żadnym innym przypadku. Można zatem wywnioskować, że sieci QLSTM są znacznie mniej podatne na anomalię podczas operowania na danych kwaternionowych.

Zaobserwowane anomalie wykazały również przewagę zastosowania funkcji straty MSE nad QALE, gdyż wartości skoków były za każdym razem mniejsze dla funkcji MSE. Jednak nie można wykluczyć, że przy dłuższym treningu ten trend mógłby się odwrócić. Ograniczenia sprzętowe, na których przeprowadzane były eksperymenty, wpłynęły na ograniczenie liczby przetrenowanych epok.

Eksperymenty na zbiorach *Biodro* i *Stopa* wykazały stabilny trend spadkowy dla każdego z badanych modeli, potwierdzając oczekiwania, że błąd będzie tym większy, im bardziej jest zestaw treningowy. Miary współczynnika zmienności zbioru oraz rozstęp interkwartylowego zostały użyte do stopniowania trudności zestawów treningowych. Sieci trenowane na funkcji MSE wykazały nieco niższy błąd średniej różnicy kątów, co zostało przedstawione dla zestawu treningowego *Biodro* na Rysunku 4.25 oraz dla zestawu *Stopa* na Rysunku 4.26.



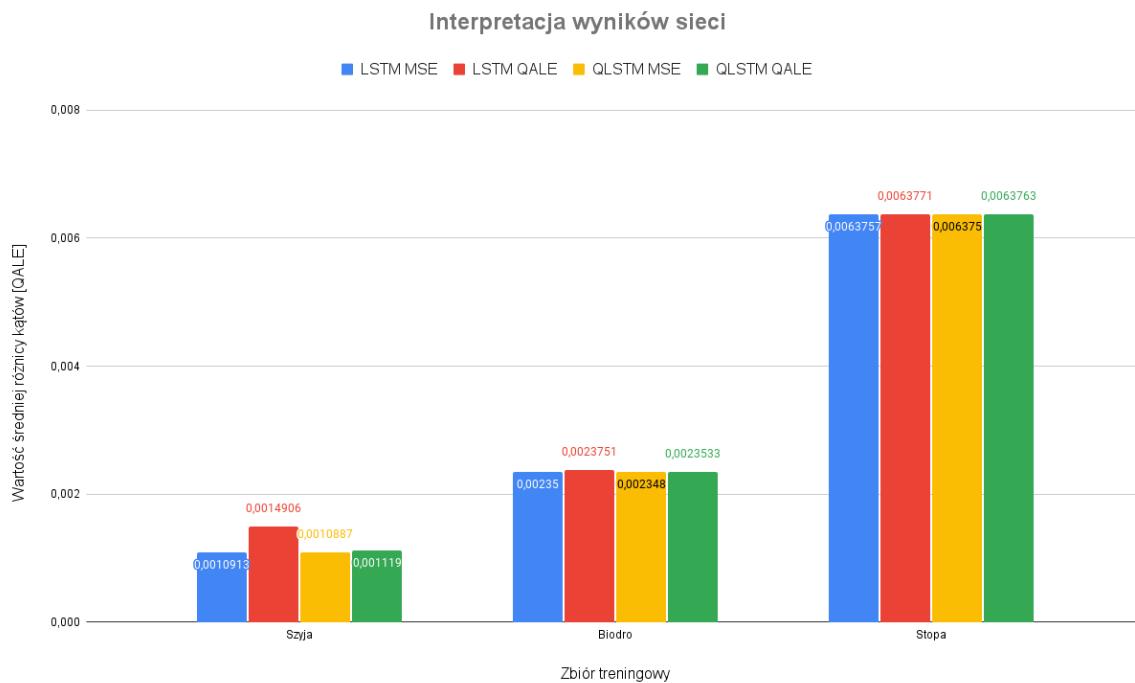
Rysunek 4.25: Wykres porównawczy zmian średniej różnicy kątów dla zbioru Biodro



Rysunek 4.26: Porównanie zmian średniej różnicy kątów dla zbioru Stopa

Warto również zwrócić uwagę na czasy treningu modeli LSTM oraz QLSTM. Pomimo marginalnie lepszych wyników sieci QLSTM, ich czasy treningu oscylowały w okolicy 50 godzin na model, co prawie stukrotnie przekracza średnią wartość czasu trenowania sieci LSTM, które zakończyły trening w około 30 minut na model. Jest to istotny wniosek z punktu widzenia praktycznego zastosowania, ponieważ różnica w wynikach pomiędzy typami modeli nie jest adekwatna do czasu oczekiwania na gotowy model. Istnieje prawdopodobieństwo, że problem z długim czasem uczenia sieci kwaternionowych wynika z ich implementacji na wysokim poziomie, co stanowi obszar do dalszych badań.

Podsumowując, w kontekście problemów regresyjnych operujących na danych kwaternionowych, model QLSTM trenowany przy użyciu funkcji MSE wykazał najlepszą precyzję, osiągając marginalnie lepsze wyniki niż pozostałe modele. W przypadkach, gdzie trend był stabilny i wartość błędu systematycznie malała z każdą epoką, wybór funkcji straty nie miał znaczącego wpływu na wynik. Funkcja MSE wykazała lepszą precyzję niż QALE w eksperymentach, gdzie występowały anomalie, co czyni ją lepszym wyborem w kontekście analizowanego problemu. Porównanie ostatecznych wyników każdego z badanych modeli zostało przedstawione na Rysunku 4.27.



Rysunek 4.27: Porównanie zmian średniej różnicy katów dla modeli LSTM MSE, LSTM QALE, QLSTM MSE oraz QLSTM QALE

Rozdział 5

Podsumowanie

Celem pracy było zweryfikowanie hipotezy, w myśl której sieci neuronowe są zdolne do pracy z danymi kwaternionowymi. Główną problematyką tego zagadnienia była kwestia braku powszechnych, komercyjnych rozwiązań, stosujących sztuczną inteligencję bazującą na danych kwaternionowych. Prawdopodobnie, decydujący wpływ na tę sytuację ma innowacyjność rozwiązania, którego popularność w środowiskach naukowych wzrosła dopiero na początku XXI w. Badania nad systemami integrującymi sieci neuronowe z danymi kwaternionowymi są istotne dla dalszego rozwoju dziedziny uczenia maszynowego, umożliwiając wyłonienie aspektów, w których zastosowanie kwaternionów ma wpływ na efektywność oraz precyzyjność sieci.

Na potrzeby badań opracowano model kwaternionowej sieci rekurencyjnej QLSTM, którego wszystkie elementy były w postaci danych kwaternionowych, a kluczowe znane procesy uczenia maszynowego, takie jak propagacja wsteczna czy przebieg w przód, zostały rozbudowane o algebrę kwaternionów. Zaimplementowano również autorską funkcję strat, określającą błąd na podstawie kąta zawartego pomiędzy kwaternionem wynikowym, a oczekiwany. Badania przeprowadzono na zbiorach treningowych będących sekwencjami kwaternionowymi opisującymi rotacje stawów w czasie, które zostały pozyskane z systemu przechwytywania ruchu. Przeprowadzone eksperymenty skupiały się na zestawieniu wyników sieci wyposażonych w algebrę Hamiltona, z podstawowymi sieciami rekurencyjnymi w problemie regresyjnym. Owe zagadnienie polega na przewidywaniu dalszego postępu rotacji na podstawie wprowadzonej sekwencji kwaternionów.

Badanie wykazało że sieć QLSTM wytrenowana na funkcji straty MSE uzyskuje najlepsze efekty ze wszystkich badanych, z uwzględnieniem, że różnice pomiędzy wartościami średniego błędu były minimalne. Sieci kwaternionowe utrzymywały również zauważalnie większą stabilność w eksperimentach, na danych zawierających krótki zakres zmian orientacji, w przeciwieństwie do standardowych sieci rekurencyjnych gdzie zaobserwowano wyraźne skoki. Dodatkowo wyniki średniej straty były tym niższe, im prostszy był analizowany zbiór danych, co było rezultatem przewidywanym i oczekiwany.

Pomimo lepszych wyników sieci QLSTM, w większości przeprowadzonych badań nie były one znaczące w porównaniu do sieci LSTM. W szczególności istotnym jest, iż dane sieci kwaterionowe uczyły się prawie sto razy dłużej, co w kontekście minimalnej poprawy, stawia QLSTM w gorszym świetle. Ważna jest również sama obserwacja zachowań sieci wytrenowanych na autorskiej funkcji straty QALE. Pomimo, że błąd kątów pomiędzy kwaterionami był głównym kryterium oceny precyzji sieci, modele wytrenowane na tej funkcji generowały gorsze wyniki od sieci wyuczonych na podstawowej funkcji MSE.

Badania z pewnością przyczyniły się do dogłębniego zrozumienia założeń sieci kwaterionowych oraz wykazały ich potencjał wraz z problemami wiążącymi się z nimi. Kwestia długiego czasu treningu pozostawia obszerne pole do dalszych badań, gdyż istnieje duże prawdopodobieństwo iż problem wynika z kwestii czysto technicznych. Niemniej jednak eksperyment wykazał wąski zakres zadań, do których QLSTM mógłby zostać wykorzystany, sprawdzając jedynie jego podstawowe możliwości. Przeprowadzenie dalszych badań nad bardziej złożonymi problemami oraz chęć wykorzystywania QLSTM w komercyjnych rozwiązańach pozwoliłoby na zauważenie pełnego potencjału tego systemu.

Bibliografia

- [1] Paolo Arena, Luigi Fortuna, Luigi G. Occhipinti oraz Maria G. Xibilia. „Neural networks for quaternion-valued function approximation”. W: *Proceedings of IEEE international symposium on circuits and systems-ISCAS'94*. Tom 6. IEEE. 1994, strony 307–310.
- [2] Dr. Bobbe Baggio oraz Nov Omana. „AI and the Agile Workplace”. W: *Journal of Systemics, Cybernetics and Informatics* 17.2 (2019), strony 84–91.
- [3] Giuseppe Bonacorso. *Machine Learning Algorithms*. Packt Publishing Ltd, 2017. ISBN: 9781785884511.
- [4] Amy Buchmann. „A Brief History of Quaternions and the Theory of Holomorphic Functions of Quaternionic Variables”. W: *Department of Mathematics and Computer Sciences Schmid College of Science Chapman University* (2011).
- [5] Varun Cheedalla. *Google vs. Siri vs. Alexa vs. Cortana: Which Reigns Supreme?* 2020. URL: <https://medium.com/techtalkers/google-vs-siri-vs-alexa-vs-cortana-which-reigns-supreme-faf7143ccbca> (opublikowano 23/06/2020).
- [6] Yogesh K. Dwivedi, Anuj Sharma oraz Nripendra P. Rana. „Evolution of artificial intelligence research in Technological Forecasting and Social Change: Research topics, trends, and future directions”. W: *Technological Forecasting and Social Change* 192 (2023), strona 122579.
- [7] Morten Engell-Nørregård oraz Kenny Erleben. „Estimation of Joint types and Joint Limits from Motion capture data”. W: *Václav Skala-UNION Agency* (2009).
- [8] Isaac Galatzer-Levy, Kelly V. Ruggles oraz Zhe Chen. „Relevance of Machine Learning to the Study of Stress Pathology, Recovery, and Resilience”. W: *Chronic Stress* 2 (2018).
- [9] GitHub. *The top programming languages*. 2023. URL: <https://octoverse.github.com/2022/top-programming-languages>.
- [10] Aurélien Géron. *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow*. O'Reilly, 2020. ISBN: 978-83-283-6002-0.

- [11] Rich Haridy. *Single MRI scan promises to diagnose early- and late-stage Alzheimer's*. 2022. URL: <https://newatlas.com/medical/mri-scan-machine-learning-diagnose-alzheimers-dementia/> (opublikowano 20/06/2022).
- [12] Scripting helpers. *How to think about Quaternions*. 2018. URL: <https://scriptinghelpers.org/blog/how-to-think-about-quaternions> (opublikowano 27/06/2018).
- [13] Sepp Hochreiter oraz Juergen Schmidhuber. „Long Short-Term Memory”. W: *Neural Computation* 9.8 (1997), strony 1735–1780.
- [14] David H. Hubel oraz Torsten N. Wiesel. „Receptive fields and functional architecture of monkey striate cortex”. W: *The Journal of Physiology* 195.1 (1968), strony 215–243.
- [15] Polsko-Japońska Akademia Technik Komputerowych. *Centrum Badawczo-Rozwojowe w Bytomiu*. 2015. URL: <http://bytom.pja.edu.pl/>.
- [16] Yann LeCun. „Gradient-based learning applied to document recognition”. W: *Proceedings of the IEEE* 86.11 86.11 (1998), strony 2278–2324.
- [17] J. M. McCarthy. *An Introduction to Theoretical Kinematics*. MIT Press, 1990. ISBN: 9780262132527.
- [18] Warren S. McCulloch oraz Walter Pitts. „A logical calculus of the ideas immanent in nervous activity”. W: *The bulletin of mathematical biophysics* 5 (1943), strony 115–133.
- [19] Mecharithm. *Unit Quaternions to Express Orientations in Robotics*. 2019. URL: <https://www.mecharithm.com/unit-quaternions-to-express-orientations-in-robotics/> (opublikowano 25/08/2019).
- [20] Tom Michael Mitchell. *Machine learning*. McGraw-hill, 2007. ISBN: 9783662124055.
- [21] Hendrik D. Mouton. „Comparison of body rotations using Euler angles and quaternions”. W: *Faculty of Engineering and the Built Environment* (2021).
- [22] Titouan Parcollet. „Quaternion neural networks”. phdthesis. Avignon, Francja: Académie d'Aix-Marseille, Avignon Université, 2019.
- [23] Titouan Parcollet, Mirco Ravanelli, Mohamed Morchid, Georges Linalès, Chiheb Trabelsi, Renato De Mori oraz Yoshua Bengio. „Quaternion recurrent neural networks”. W: *arXiv preprint arXiv:1806.04418* (2018).
- [24] Yin Qilin. „Quaternion convolutional neural network for color image classification and forensics”. W: *IEEE Access* 7 7 (2019), strony 20293–20301.
- [25] Frank Rosenblatt. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Tom 55. Spartan books Washington, DC, 1962.

- [26] David E. Rumelhart, Geoffrey E. Hinton oraz Ronald J. Williams. „Learning internal representations by error propagation”. W: *Institute for Cognitive Science, University of California* (1985).
- [27] Ken Shoemake. „Animating rotation with quaternion curves”. W: *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*. 1985, strony 245–254.
- [28] Euclidean Space. *Quaternion Notations*. 1998. URL: <https://www.euclideanspace.com/math/algebra/realNormedAlgebra/quaternions/notations>.
- [29] Motion Lab Systems. *The C3D File Format - A Technical User Guide*. Motion Lab Systems, Inc. Baton Rouge LA, USA, 2021.
- [30] John Terra. *Keras vs Tensorflow vs Pytorch: Key Differences Among Deep Learning*. 2023. URL: <https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article> (opublikowano 23/08/2023).
- [31] TParcollet. *PyTorch-Quaternion-Neural-Networks*. 2019. URL: <https://github.com/Orkis-Research/Pytorch-Quaternion-Neural-Networks>.
- [32] Nie od razu naukę zbudowano. *William Rowan Hamilton: kwaterniony – odkrycie i obsesja*. 2020. URL: <https://kierul.wordpress.com/2020/09/15/william-rowan-hamilton-kwaterniony-odkrycie-i-obsesja-16-pazdziernika-1843/> (opublikowano 15/09/2020).

Dodatki

Spis użytych terminów oraz skrótów

Terminologia

Agent - jednostka zdolna do wykonywania działań w środowisku, często używana w kontekście sztucznej inteligencji i uczenia ze wzmacnieniem (ang. *Agent*)

Algebra Hamiltona - system matematyczny oparty na kwaterionach, wprowadzony przez Sir Williama Rowana Hamiltona. Jest to rozszerzenie liczb zespolonych na cztery wymiary (ang. *Hamilton's algebra*)

Analiza skupień - technika statystyczna i uczenia nienadzorowanego, służąca do identyfikowania grup podobnych obiektów w danych (ang. *Cluster analysis*)

Batch size - liczba próbek danych przetwarzanych jednocześnie w jednej iteracji algorytmu uczenia maszynowego. Wybór rozmiaru wsadu ma wpływ na szybkość uczenia się modelu oraz na jego ostateczną wydajność

Bramka logiczna - podstawowy element cyfrowy używany do wykonywania operacji logicznych, takich jak AND, OR, NOT (ang. *Logic gate*)

Bramka wejściowa - element LSTM kontrolujący, które informacje z aktualnego kroku czasowego powinny zostać zapisane w komórce pamięci (ang. *Input gate*)

Bramka wyjściowa - element LSTM kontrolujący, które informacje z komórki pamięci powinny być przekazane do następnego kroku czasowego (ang. *Output gate*)

Bramka zapomnienia - element LSTM kontrolujący, jakie informacje z poprzedniego kroku czasowego powinny zostać zapomniane (ang. *Forget gate*)

Cecha - w zależności od kontekstu ma kilka różnych znaczeń, jednakże najczęściej jest używana jako atrybut wraz jej wartością (ang. *feature*)

Eksploracja danych - proces analizowania dużych zestawów bądź zbiorów danych w celu identyfikacji wzorców, korelacji, trendów czy anomalii, które mogą dostarczyć wartościowych informacji lub przewidywań (ang. *Data Mining*)

Entropia krzyżowa - miara błędu stosowana w uczeniu maszynowym dla problemów klasyfikacyjnych, która mierzy różnicę między prawdziwym a przewidywanym rozkładem prawdopodobieństwa (ang. *Cross-entropy*)

Filtry (kernele) - małe matryce używane do przetwarzania danych w sieciach konwolucyjnych, które przesuwają się przez obraz wejściowy, wykrywając różne cechy (ang. *Filters (kernels)*)

Gimbal lock - zjawisko utraty jednego stopnia swobody w układzie trzech obrotów, co może prowadzić do niemożności jednoznacznego określenia orientacji. Często występuje w systemach opartych na kątach Eulera.

Kąty Eulera - układ trzech kątów, za pomocą których można jednoznacznie określić orientację ciała w przestrzeni trójwymiarowej (ang. *Euler angles*)

Klasyfikacja - proces przypisywania kategorii lub etykiet do obiektów na podstawie ich cech, będący typem zadania nadzorowanego (ang. *Classification*)

Komórka pamięci - element rekurencyjnej sieci neuronowej, który pozwala na przechowywanie i manipulację informacją przez kilka kroków czasowych (ang. *Memory cell*)

Konwolucja - operacja matematyczna, która łączy dwie funkcje w jedną trzecią, będącą funkcją reprezentującą, jak kształt jednej funkcji jest modyfikowany przez drugą. W kontekście sieci neuronowych, konwolucja jest często używana w przetwarzaniu obrazu, gdzie filtr jest przesuwany po obrazie wejściowym, aby wyprodukować nowy obraz (ang. *Convolution*)

Kwaternion - struktura algebraiczna rozszerzająca ideę liczb rzeczywistych, zespolonych i układu kartezjańskiego Posiada jedną liczbę rzeczywistą oraz trzy ujęte w postaci wektora (ang. *Quaternions*)

LeNet-5 - jedna z pierwszych konwolucyjnych sieci neuronowych, stworzona przez Yanna LeCuna w 1998 roku, często stosowana do rozpoznawania pisma ręcznego (ang. *LeNet-5*)

Liczby zespolone - liczby będące elementami rozszerzenia ciała liczb rzeczywistych o jednostkę urojoną i (ang. *Complex numbers*)

Macierz obrotu - kwadratowa macierz używana do wykonania liniowego przekształcenia obrotu w układzie euklidesowym (ang. *Rotation matrix*)

Mapa cech - wyjście warstwy konwolucyjnej w sieci neuronowej, reprezentujące cechy wykryte w danych wejściowych (ang. *Feature map*)

Miara wydajności - wskaźnik używany do oceny, poprawności działania jednostki lub procesu (ang. *Performance measure*)

Model - matematyczna reprezentacja danych lub zjawiska, ulepszana na podstawie analizy danych przy użyciu algorytmów uczenia maszynowego

Moduł uwagi - składnik sieci neuronowej, pozwalający modelowi na ważenie różnych części wejścia podczas podejmowania decyzji, symulując zarazem ludzką uwagę

Motion Capture - technika pozwalająca na przechwytywanie ruchu aktora i jego zapis na komputerze w postaci pozycji i rotacji punktów charakterystycznych

MYCIN - system ekspercki stworzony w Stanford University w latach 70, którego zadaniem było wspomaganie medyków w diagnostyce zakażeń bakteryjnych sugerując przy tym odpowiednie leki.

Nadmierne dopasowanie - zjawisko występujące w uczeniu maszynowym, gdy model zbyt dokładnie dopasowuje się do danych treningowych kosztem swojej zdolności do generalizacji na nowych danych (ang. *Overfitting*)

Neuron - podstawowy typ komórki w układzie nerwowym, spełniający rolę przetwarzania i przekazywania informacji poprzez elektryczne i chemiczne sygnały. Sieci neuronowe w informatyce symulują działanie rzeczywistych neuronów rozszerzając ich możliwości o wartości zmienoprzecinkowe z przedziału $<0, 1>$ (ang. *Neuron*)

Orientacja - pojęcie odnoszące się do położenia przestrzennego obiektu względem układu odniesienia. Może odnosić się do kierunku, w którym obiekt jest skierowany, lub do kątów obrotu obiektu wokół określonych osi (ang. *Orientation*)

Perceptron - prosty algorytm uczenia maszynowego, będący rodzajem liniowego klasyfikatora, który korzysta z funkcji progowej jako funkcji aktywacji

Podpróbkowanie - operacja w przetwarzaniu obrazu i analizie, agregująca dane w określonych regionach, zazwyczaj w celu zmniejszenia ich wymiarowości (ang. *Pooling*)

Pole recepcyjne - obszar w sieciach neuronowych, z którego każda jednostka otrzymuje część informacji wejściowych. Jest to kluczowy element w konwolucyjnych sieciach neuronowych (ang. *Receptive field*)

Problemy regresyjne - typ zadań uczenia maszynowego, w których celem jest przewidywanie ciągłej zmiennej wynikowej na podstawie jednej lub więcej zmiennych wejściowych (ang. *Regression problems*)

Propagacja wsteczna - metoda uczenia sieci neuronowych polegająca na aktualizacji wag na podstawie błędu obliczanego od końca sieci do początku (ang. *Backpropagation*)

Redukcja wymiarowości - proces zmniejszania liczby zmiennych wejściowych, stosowany w celu uproszczenia modeli danych (ang. *Dimensionality reduction*)

Regresja - technika statystyczna i uczenia maszynowego, służąca do przewidywania ciągłej zmiennej wynikowej na podstawie jednej lub więcej zmiennych wejściowych (ang. *Regression*)

Rotacja - operacja geometryczna, która obraca obiekt wokół punktu (w 2D) lub osi (w 3D). W matematyce rotacja jest zazwyczaj opisywana za pomocą macierzy rotacji lub kwaternionów (ang. *Rotation*)

Rozproszenie zbioru - miara tego, jak daleko elementy zbioru danych są rozproszone w przestrzeni na podstawie wybranych kryteriów, takich jak odległość Euklidesowa lub Manhattan (ang. *Set dispersion*)

Scraper - narzędzie lub program komputerowy służący do ekstrakcji danych z witryn internetowych lub obcych formatów plików

Sieć neuronowa - rodzaj modelu sztucznej inteligencji przeznaczony do przetwarzania informacji, którego budowa i zasada działania są wzorowane na funkcjonowaniu fragmentów rzeczywistego systemu nerwowego (ang. *Neural network*)

Softmax - funkcja stosowana w uczeniu maszynowym, konwertująca wektor liczb rzeczywistych na wektor prawdopodobieństw, które sumują się do wartości 1. Używana zazwyczaj w warstwach wyjściowych sieci neuronowych klasyfikujących (ang. *Softmax function*)

System adaptacyjny - system zdolny do zmiany swojego zachowania lub struktury w odpowiedzi na zmieniające się warunki środowiskowe (ang. *Adaptive system*)

Test Turinga - metoda oceny zdolności maszyny do wykazania inteligencji polegająca na prowadzeniu konwersacji tekstowej między człowiekiem a maszyną. Jeżeli po pewnym czasie rozmówcy nie są w stanie jednoznacznie stwierdzić, czy rozmawiają z człowiekiem, czy maszyną, to uznaje się, że maszyna przeszła test Turinga - to znaczy, wykazała inteligencję na poziomie nieroóżnialnym od ludzkiego (ang. *Turing Test*)

Uczenie głębokie - poddziedzina uczenia maszynowego koncentrująca się na algorytmach inspirowanych strukturą i funkcją mózgu, zwanych sztucznymi sieciami neuronowymi. Głębokie w uczeniu głębokim odnosi się do liczby warstw przez które przechodzą dane w sieci neuronowej (ang. *Deep learning*)

Uczenie maszynowe - poddziedzina sztucznej inteligencji, odpowiedzialna za rozwój algorytmów, które umożliwiają komputerom samodzielne uczenie się z doświadczenia, zazwyczaj poprzez analizę dostępnych danych (ang. *Machine learning*)

Uczenie nadzorowane - rodzaj uczenia maszynowego, w którym model trenowany na danych wejściowych oraz odpowiadających im etykietach (ang. *Supervised learning*)

Uczenie nienadzorowane - rodzaj uczenia maszynowego, w którym model uczy się wykrywać wzorce w danych. Jednak nie ma on dostępu do informacji o konkretnych etykietach wyjściowych (ang. *Unsupervised learning*)

Uczenie ze wzmacnieniem - rodzaj uczenia maszynowego, w którym model uczy się poprzez interakcje ze środowiskiem, otrzymując pozytywne i negatywne nagrody za swoje działania (ang. *Reinforcement learning*)

Zmienność zbioru - miara zróżnicowania elementów w zbiorze danych. W kontekście statystyki, zmienność może być mierzona za pomocą różnych wskaźników, takich jak wariancja, odchylenie standardowe lub zakres (ang. *Set variability*)

Spis skrótów:

AI - sztuczna inteligencja (ang. *Artificial Intelligence*)

AOI - metoda kontroli wizualnej, której celem jest wykrycie wad na elementach, takich jak płytki drukowane (ang. *Automated Optical Inspection*)

API - interfejs programistyczny aplikacji będący zestawem definicji, protokołów i narzędzi używanych do tworzenia oprogramowania (ang. *Application Programming Interface*)

C3D - format pliku przechowujący informacje wyjściowe nagrań z systemów przechwytywania ruchu

CNN - splotowa (konwolucyjna) sieć neuronowa (ang. *Convolutional Neural Network*)

CUDA - platforma programistyczna i model programowania opracowany przez NVIDIA, który pozwala deweloperom wykorzystać moc procesorów graficznych (ang. *GPU*) do obliczeń ogólnego przeznaczenia (ang. *Compute Unified Device Architecture*)

CSV - format pliku używany do przechowywania i wymiany danych, w którym wartości są oddzielane przecinkami (ang. *Comma-Separated Values*)

CV - współczynnik zmienności zbioru (ang. *Coefficient of Variation*)

GPU - procesor graficzny (ang. *Graphics Processing Unit*)

GRU - bramkowa jednostka cykliczna (ang. *Gated Recurrent Unit*)

GSN - generatywna sieć stochastyczna (ang. *Generative Stochastic Network*)

IT - skrótowa nazwa dziedziny Informatyki (ang. *Informatics*)

IQR - rozstęp międzykwartylowy (ang. *Interquartile Range*)

LSTM - typ rekurencyjnej sieci neuronowej zaprojektowanej do zapamiętywania długoterminowych zależności w sekwencjach danych (ang. *Long Short-Term Memory*)

ML - uczenie maszynowe (ang. *Machine Learning*)

MLP - perceptron wielowarstwowy (ang. *Multilayer Perceptron*)

MSE - błąd średniokwadratowy (ang. *Mean Squared Error*)

PCB - płytka drukowana (ang. *Printed Circuit Board*)

PER - współczynnik błędu fonemów (ang. *Phoneme Error Rate*)

PJATK - Polsko-Japońska Akademia Technik Komputerowych

ReLU - funkcja aktywacji używana w sieciach neuronowych, która zwraca zero dla wartości ujemnych i dla wartości dodatnich zwraca samą wartość (ang. *Rectified Linear Unit*)

RGB - model koloru oparty na trzech kanałach: czerwonym, zielonym i niebieskim, używany w technologii wyświetlania i przetwarzania obrazów (ang. *Red, Green, Blue*)

RNN - rekurencyjna sieć neuronowa (ang. *Recurrent Neural Network*)

SLERP - metoda interpolacji sferycznej, która zapewnia płynne przejścia między dwoma punktami na powierzchni sfery (ang. *Spherical Linear Interpolation*)

t-SNE - technika do wizualizacji wysokowymiarowych danych przez redukcję wymiarów, z zachowaniem możliwie największej ilości informacji o pierwotnych dystansach między punktami (ang. *t-Distributed Stochastic Neighbor Embedding*)

TLU - progowa jednostka logiczna (ang. *Threshold logic unit*)

QALE - kwaternionowy błąd długości kąta (ang. *Quaternion Angle Length Error*)

QBPTT - kwaternionowa propagacja wsteczna przez czas (ang. *Quaternion Backpropagation Through Time*)

QCNN - kwaternionowa sieć CNN (ang. *Quaternion Convolutional Neural Network*)

QLSTM - kwaternionowa sieć LSTM (ang. *Quaternion Long Short-Term Memory*)

QMSE - kwaternionowy błąd średniokwadratowego (ang. *Quaternion Mean Squared Error*)

QRNN - kwaternionowa sieć RNN (ang. *Quaternion Recurrent Neural Network*)

XOR - operacja logiczna, która zwraca prawdę tylko wtedy, gdy dokładnie jedno z wejść jest prawdziwe (ang. *Exclusive OR*)

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie do pracy dołączono dodatkowe pliki zawierające:

- dostęp do repozytorium zawierającego źródła projektu,
- zbiory danych użyte w eksperymentach,
- film pokazujący działanie opracowanego oprogramowania.

Spis rysunków

2.1	Problem klasyfikacji dla filtra spamu [10]	8
2.2	Problem regresyjny dla przewidywania wartości numerycznej dla cechy. [10]	8
2.3	Przykład wizualizacji t-SNE wskazującej semantyczny podział skupień. [10]	9
2.4	Schemat przebiegu operacji w procesie uczenia ze wzmacnieniem. [8]	10
2.5	Sztuczne sieci neuronowe przeprowadzające proste operacje logiczne. [10] . .	10
2.6	Architektura perceptronu wielowarstwowego z jedną warstwą ukrytą. [10] . .	11
2.7	Warstwy splotowe zawierające wiele map cech, a także zdjęcie z trzema kanałami barw. [10]	13
2.8	Neuron rekurencyjny rozwijany w czasie [10]	14
2.9	Architektury sieci RNN: (kolejno) sekwencyjna, sekwencyjno-wektorowa, wektorowo-sekwencyjna, koder-dekoder. [10]	16
2.10	Schemat komórki LSTM [10]	17
2.11	Tablica upamiętniająca odkrycie kwaternionów na moście Broom w Dublinie [32]	19
2.12	Rotacja reprezentowana przez kwaternion [19]	23
2.13	Rotacja stawu w grafice 3D [7]	25
2.14	Animacja szkieletowa [7]	26
2.15	Logotypy asystentów: Alexa, Siri, Google Assistant, Cortana [5]	34
2.16	Obraz skanu rezonansu magnetycznego głowy [11]	36
4.1	Współczynnik zmienności zbiorów treningowych	51
4.2	Rozstęp interkwantylowy zbiorów treningowych	51
4.3	Zbiór wykresów opisujących wartości parametrów kwaternionów w czasie . .	52
4.4	Wykresy opisujące zmianę MSE i QALE dla modelu Biodro LSTM MSE .	54
4.5	Wykresy opisujące zmianę średniego błędu oraz średniej różnicy katów dla modelu Biodro LSTM QALE	55
4.6	Wykresy opisujące zmianę średniego błędu oraz średniej różnicy katów dla modelu Biodro QLSTM MSE	56
4.7	Wykresy opisujące zmianę średniego błędu oraz średniej różnicy katów dla modelu Biodro QLSTM QALE	57

4.8 Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Stopa LSTM MSE	58
4.9 Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Stopa LSTM QALE	59
4.10 Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Stopa QLSTM MSE	60
4.11 Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Stopa QLSTM QALE	61
4.12 Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Szyja LSTM MSE	62
4.13 Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Szyja LSTM QALE	63
4.14 Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Szyja QLSTM MSE	64
4.15 Wykresy opisujące zmianę średniego błędu oraz średniej różnicy kątów dla modelu Szyja QLSTM QALE	65
4.16 Porównanie zmian średniej różnicy kątów dla modelu Szyja LSTM MSE oraz Szyja LSTM QALE	67
4.17 Porównanie zmian średniej różnicy kątów dla modelu Biodro LSTM MSE oraz Biodro LSTM QALE	68
4.18 Porównanie zmian średniej różnicy kątów dla modelu Stopa LSTM MSE oraz Stopa LSTM QALE	69
4.19 Porównanie zmian średniej różnicy kątów dla modeli LSTM MSE oraz LSTM QALE	70
4.20 Porównanie zmian średniej różnicy kątów dla modelu Szyja QLSTM MSE oraz Szyja QLSTM QALE	71
4.21 Porównanie zmian średniej różnicy kątów dla modelu Biodro QLSTM MSE oraz Biodro QLSTM QALE	72
4.22 Porównanie zmian średniej różnicy kątów dla modelu Stopa QLSTM MSE oraz Stopa QLSTM QALE	73
4.23 Porównanie zmian średniej różnicy kątów dla modeli QLSTM MSE oraz QLSTM QALE	74
4.24 Porównanie zmian średniej różnicy kątów dla zbioru Szyja	75
4.25 Wykres porównawczy zmian średniej różnicy kątów dla zbioru Biodro	76
4.26 Porównanie zmian średniej różnicy kątów dla zbioru Stopa	76
4.27 Porównanie zmian średniej różnicy kątów dla modeli LSTM MSE, LSTM QALE, QLSTM MSE oraz QLSTM QALE	77

Spis tabel

2.1 Tabela mnożenia jednostek kwaternionów	21
4.1 Wyniki modelu LSTM dla funkcji MSE oraz zbioru <i>biodro</i>	54
4.2 Wyniki modelu LSTM, wytrenowanego przy użyciu funkcji błędu średnio-kwadratowego MSE oraz zbioru <i>biodro</i>	55
4.3 Wyniki modelu QLSTM, wytrenowanego przy użyciu funkcji błędu średniookwadratowego MSE oraz zbioru <i>biodro</i>	56
4.4 Wyniki modelu QLSTM, wytrenowanego przy użyciu funkcji QALE oraz zbioru <i>biodro</i>	57
4.5 Wyniki modelu LSTM, wytrenowanego przy użyciu funkcji MSE oraz zbioru <i>stopa</i>	58
4.6 Wyniki modelu LSTM, wytrenowanego przy użyciu funkcji QALE oraz zbioru <i>stopa</i>	59
4.7 Wyniki modelu QLSTM, wytrenowanego przy użyciu funkcji MSE oraz zbioru <i>stopa</i>	60
4.8 Wyniki modelu QLSTM, wytrenowanego przy użyciu funkcji QALE oraz zbioru <i>stopa</i>	61
4.9 Wyniki modelu LSTM, wytrenowanego przy użyciu funkcji MSE oraz zbioru <i>szyja</i>	62
4.10 Wyniki modelu LSTM, wytrenowanego przy użyciu funkcji QALE oraz zbioru <i>szyja</i>	63
4.11 Wyniki modelu QLSTM, wytrenowanego przy użyciu funkcji MSE oraz zbioru <i>szyja</i>	64
4.12 Wyniki modelu QLSTM, wytrenowanego przy użyciu funkcji QALE oraz zbioru <i>szyja</i>	65
4.13 Zestawienie czasu treningu wszystkich modeli poddanych badaniu	66