AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa

Implementacja asocjacyjnych grafów neuronowych DASNG i porównanie ich działania z podobnymi rozwiązaniami inteligencji obliczeniowej dla różnych problemów i danych.

Implementation of DASNG associative neural graphs and comparing their performance with similar computational intelligence solutions for various problems and data.

Autor: *Mateusz Badacz*
Kierunek studiów: *Inżynieria Biomedyczna*
Opiekun pracy: *Dr hab. inż. Adrian Horzyk, prof. AGH*

Kraków, *2022 r.*

Spis treści

1.Wprowadzenie

Jedną z najważniejszych ról we współczesnym świecie odgrywają technologie informacyjne i dane związane z każdą wykonywaną przez człowieka czynnością. Można do takich zaliczyć pliki cookie wykorzystywane w każdej przeglądarce internetowej, trend sprzedawania konkretnego towaru, a także profile medyczne i dane zdrowotne. Każdego miesiąca ludzkość generuje około 400 milionów terabajtów informacji [1]. Z coraz bardziej rosnącą popularnością Internetu Rzeczy (ang. Internet of Things, IoT), tempo generowania danych jeszcze dodatkowo wzrośnie. Tylko w latach 2019-2021, wygenerowano 90% danych składowanych na całym świecie [2].

Dane w takiej ilości tworzą nieskończone możliwości z dziedzin obróbki i wykorzystywania np. w systemach inteligentnych, tj. uczeniu maszynowym, systemach rekomendacyjnych czy sztucznej inteligencji. Z drugiej strony, aby pozyskane informacje można było wykorzystać do konkretnych celów, konieczne jest posiadanie odpowiedniego sposobu na wydajne przetrzymywanie ich. Praca łączy oba zagadnienia, przedstawiając propozycję zoptymalizowanej implementacji algorytmu asocjacyjnego, korzystającego z podstawowych konceptów sztucznych sieci neuronowych. Jednocześnie struktura przechowuje zagregowane dane, umożliwia łatwy i szybki dostęp do zdefiniowanych zapytań. Ponadto, analiza procesów pobudzeń stojących za sposobem działania sieci, daje pokłady do przełożenia wyników na próbę zrozumienia funkcjonowania neuronów biologicznych.

1.1.Cel i zakres pracy

Głównym celem pracy magisterskiej było zaimplementowanie zoptymalizowanej wersji asocjacyjnej struktury semantycznych grafów neuronowych DASNG. Jej kluczowym zastosowaniem jest przekształcanie relacyjnych baz danych w strukturę przyspieszającą procesy operowania na dużych zbiorach danych. Sieć wykorzystuje działanie grafu sortującego ASA-graf, w celu agregacji i grupowania wartości podobnych dla poszczególnych atrybutów.

W części teoretycznej omówione zostały podstawowe pojęcia i zagadnienia związane z sieciami neuronowymi, a w szczególności ich wyspecjalizowaną odmianą – sieciach impulsowych. Ponadto, przegląd literaturowy pozwolił na wyszczególnienie rozwiązań podobnych, działających w dziedzinie inteligencji komputerowej.

W części praktycznej zawarto opis procesu budowy algorytmu, dokładne przedstawienie konceptów stojących za strukturą DASNG, a także pojedynczych komponentów, które wzbogacały działanie całego grafu asocjacyjnego.

Po zaimplementowaniu, struktura oraz towarzyszące jej algorytmy zostały przetestowane i porównane z innymi rozwiązaniami stosowanymi w podobnych zagadnieniach inteligencji komputerowej. Do implementacji zostało wykorzystane wyodrębnione środowisko programistyczne Python, a także bazy danych MySQL.

2. Wstęp

2.1. Sztuczne sieci neuronowe – rys historyczny

Historia sztucznych sieci neuronowych bierze swój początek jeszcze w trakcie trwania II Wojny Światowej, kiedy to dwóch uczonych: Warren McCulloch i Walter Pitts, napisali pracę opisującą działanie ludzkiego neuronu. Na potrzeby swoich badań zamodelowali prostą sieć neuronową wykorzystując ówczesne układy elektroniczne [3]. W 1949 roku, psycholog Donald Hebb przeprowadził badania, w których stwierdził, że ścieżki neuronowe wzmacniają się z każdym kolejnym pobudzeniem. Według Hebba, kiedy jedna komórka wielokrotnie pomaga w odpaleniu innej, akson pierwszej komórki rozwija gałki synaptyczne w kontakcie z somą drugiej komórki [4]. Jest to koncepcja zasadniczo wspomagająca ludzki proces uczenia się. Oba wydarzenia podejmują temat od strony biologicznej, próbując w dokładny sposób wyjaśnić działanie niezwykle skomplikowanych struktur jakimi są komórki nerwowe.

Wymienione prace szczególnie przyczyniły się do prób powstawania sztucznych odpowiedników biologicznych komponentów układu nerwowego. Znaczny rozwój technologii komputerowych w latach 50 XX wieku dodatkowo to ułatwił. Na szczególną uwagę zasługuje, stworzony w 1957 roku przez Franka Rosenblatta, algorytm perceptronu. Do dziś wykorzystywany w najprostszych modelach uczenia maszynowego, był pierwszą udaną próba stworzenia modelu obliczeniowego opartego na pracach Hebba, będącego uproszczonym, sztucznym odpowiednikiem biologicznej sieci neuronowej [5]. Perceptron początkowo został zaplanowany nie jako program, ale jako układ elektroniczny. Jego oprogramowanie zostało napisane na maszynie IBM 704 i miało służyć prostemu rozpoznawaniu obrazów [6]. Ogłoszenie wyników eksperymentu pokazało potencjał drzemiący w dopiero odkrywanym zagadnieniu i zaciekawiło wielu ówczesnych naukowców.

Dwa lata później, Bernard Widrow i Marcian Hoff z Uniwersytetu Stanforda opracowali dwa modele o nazwach "ADALINE" i "MADALINE". Pierwszy z nich miał służyć do rozpoznawania wzorów binarnych. Dzięki odczytowi bitów strumieniowych z linii telefonicznej, model był w stanie przewidzieć kolejny pojawiający się bit. MADALINE (ang. *Many ADALINE*) była pierwszą siecią neuronową wykorzystaną do rozwiązania rzeczywistego problemu, wykorzystującą filtr adaptacyjny do eliminowania echa na liniach telefonicznych [7].

Początkowy sukces doprowadził do znacznego wyolbrzymienia wyobrażeń naukowców na temat możliwości sieci neuronowych, a ich rzeczywisty rozwój (w dużej mierze spowodowanym ograniczeniami technologicznymi) nie poruszał się do przodu w wystarczającym tempie i tradycyjne architektury von Neumanna były coraz częściej wykorzystywane w zadaniach obliczeniowych. Dodatkowym czynnikiem hamującym rozwój algorytmów inteligentnych było opublikowanie w 1969 roku słynnej pracy [8], w której autorzy udowadniają, iż standardowy, jednowarstwowy perceptron nie jest w stanie rozwiązać problemu logicznego XOR. Ówczesny stan wiedzy i nieznajomość

modeli wielowarstwowych, uniemożliwiały kontrargumentację, co przyczyniło się do znacznych cięć budżetowych badań nad sieciami neuronowymi.

Stagnacja w pracach nad rozwojem sztucznych sieci neuronowych utrzymywała się przez blisko 15 lat. Pomimo znacznego zastoju, już w 1974 roku została opublikowana rozprawa doktorska opisująca mechanizm propagacji wstecznej. Jest to metoda, w której błędy są przetwarzane na wyjściu struktury, a następnie rozprowadzane wstecz, przechodząc przez kolejne warstwy w celu korygowania wag uczenia. Propagacja wsteczna w przyszłości miała pełnić znaczną rolę w działaniu algorytmów uczenia maszynowego [9]. Kolejnym ciekawym pomysłem, którego realizacji podjął się w 1979 roku japoński uczony Kunihiko Fukushima, była sieć Neocognitron. Jego prace nad widzeniem komputerowym doprowadziły do utworzenia sieci złożonej z hierarchicznie ułożonych warstw, umożliwiającej wykrywanie i zapamiętywanie prostych schematów na obrazach przepuszczanych przez sieć [10]. Jest to jeden z pierwszych algorytmów, które później zostały nazwane sieciami głębokimi [10].

Dyskusja nad algorytmami sztucznej inteligencji ponownie rozgorzała w roku 1986 po publikacji artykułu Davida Rumelharta z Uniwersytetu Stanforda i jego zespołu [11]. Opisali oni dokładnie działanie i możliwości metody propagacji wstecznej. Wykorzystanie tej techniki, umożliwiało projektowanie skutecznie działających sieci wielowarstwowych. Wprowadzone zostało pojęcie warstw ukrytych, czyli takich znajdujących się pomiędzy warstwami wejściowymi i wyjściowymi. Najważniejszy jednak był fakt, iż ze względu na wydajność algorytmu, wsteczna propagacja umożliwiła zastosowanie sztucznych sieci neuronowych do znacznie szerszego zakresu problemów, które wcześniej były poza zasięgiem ze względu na ograniczenia czasowe i finansowe. Trzy lata później, w 1989 roku, Yann LeCun wraz ze swoim zespołem, zaproponowali model wykorzystujący mechanizm propagacji wstecznej, służący do rozpoznawania ręcznie pisanych kodów pocztowych. Algorytm działał z wysoką precyzją, posiadając błąd klasyfikacji jedynie na poziomie 1% [12]. Tak dobre wyniki przekonywały innych naukowców rozpowszechniając ideę uczenia głębokiego. LeCun wykorzystywał również podobne architektury do detekcji nieprawidłowości na obrazach medycznych, a także do detekcji miejsc splatania się intronów i eksonów w sekwencjach DNA [13].

W miarę jak naukowcy tworzyli coraz głębsze sieci neuronowe, wsteczna propagacja stawała się niewystarczającym mechanizmem, obniżającym swoją skuteczność przez występowanie problemu zanikających gradientów. Wspomniana komplikacja wywodzi się z faktu, iż ówcześnie, warstwy korzystały z bardzo prostych funkcji aktywacji: sigmoidalnej i tangensa hiperbolicznego. Z wyjątkiem argumentów z zakresu od -1 do 1, pochodne tych funkcji są bliskie zeru, z czego wywodzi się ich najbardziej niepożądana cecha - z każdą kolejną warstwą przejście gradientowe staje się coraz wolniejsze. Rodzi to problem, w którym rozbudowane architektury borykają się z powolną optymalizacją. Na szczęście, z początkiem lat 2000, z pomocą przyszły algorytmy optymalizacyjne, które wykorzystywane do dzisiaj, pomogły rozwiązać problem powolnego uczenia.

Aktualnie, sieci neuronowe i szeroko pojęta sztuczna inteligencja, wykorzystywane są w wielu gałęziach przemysłu, znacznie ułatwiając pracę człowieka. Przyszłość sieci

neuronowych leży jednak w rozwoju sprzętu i procesorów, które - wyspecjalizowane do konkretnego rodzaju algorytmów – będą w stanie zapewnić szybkie i wydajne przetwarzanie danych. [14]

2.2. Definicja i podstawowe właściwości sieci neuronowych

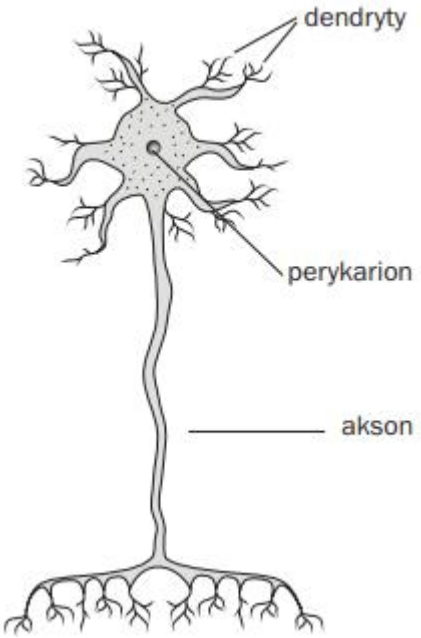
Sztuczne sieci neuronowe (ang. *Artificial Neural Networks, ANNs*) są gałęzią uczenia maszynowego, a także głównym narzędziem tzw. uczenia głębokiego. Ich nazwa, a także struktura, czerpie inspirację z budowy ludzkiego mózgu, sposobu, w jaki biologiczne neurony przesyłają sygnały pomiędzy sobą, powodując proces uczenia i przetwarzania bodźców (danych) z otoczenia [15]. Podstawowa idea stojącą za naturą sieci neuronowych leży w ich zdolności do wydajnego przeprowadzać skomplikowanych obliczeń i reprezentowania hierarchicznej struktury przetwarzanych danych. W naturze, neurony połączone aksonami i dendrytami, tworzące skomplikowane struktury neuronowe są w stanie przesyłać i wymieniać informacje, przechowywać pośrednie rezultaty obliczeń, tworzyć abstrakcyjne reprezentacje, a także rozdzielać proces uczenia na wiele pojedynczych etapów. Efektywnie utworzony model obliczeniowy takiego system również powinien być w stanie przeprowadzać procesy uczenia w sposób podobnie wydajny, jak procesy biologiczne. Do najpopularniejszych zastosowań algorytmów uczenia głębokiego można zaliczyć: tworzenie modeli predykcyjnych dla akcji giełdowych, przetwarzanie mowy ludzkiej na „język” komputerów, interpretowanie zaszumionych i niemożliwych do interpretacji okiem nieuzbrojonym obrazów medycznych, a także analiza obrazów z kamer samochodów autonomicznych.

2.2.1. Neuron biologiczny

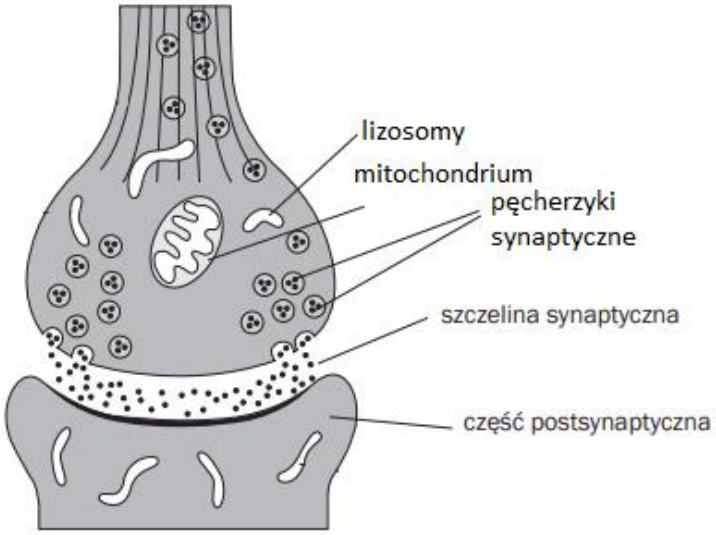
Ludzki mózg jest wysoce wyspecjalizowaną i skomplikowaną strukturą obliczeniową, umożliwiającą wykonywanie procesu logicznego myślenia, zapamiętywania, przeprowadzania obliczeń. Sztuczne sieci neuronowe od lat próbują jak najlepiej odwzorować jego architekturę, aby utworzyć algorytm zdolny do przetwarzania danych na poziomie ludzkiego mózgu. Ludzki mózg składa się z średnio 100 miliardów, gęsto połączonych w sieć komórek nerwowych [15].

Neuron jest podstawową jednostką, która wchodzi w skład układu nerwowego organizmu żywego. Działanie neuronu opiera się na otrzymywaniu i łączeniu sygnałów otrzymywanych z sąsiadujących komórek nerwowych. Impulsy nerwowe przesyłanie są kanałami nazywanymi aksonami, które zakończone są licznie rozgałęzioną strukturą - dendrytami. Każdy neuron tworzy odrębną jednostkę komórkową i impuls z jednego neuronu na drugi lub na inną komórkę efektorową jest przekazywany poprzez połączenie zwane synapsą, czyli węzeł, który kontroluje przepływ sygnału biologicznego poprzez odpowiednie uwalnianie z zakończeń komórkowych neuroprzekaźników [16] – swoistych sygnałów chemicznych wyspecjalizowanych do zadania przesyłu informacji [56].

Przekazywanie sygnałów przez synapsy ma charakter elektro-chemiczny, a siła sygnału zależy od mocy połączenia synaptycznego lub jego konduktancji. Parametry te modyfikowane są w miarę jak mózg uczy się danego bodźca, czyli im częściej takie połączenie jest wykorzystywane [15]. Dokładny schemat omówionych struktur przedstawiają ryciny 1 i 2. Jeśli połączony sygnał ze wszystkich dendrytów jest wystarczająco silny, komórka podejmuje decyzję o wytworzeniu sygnału wyjściowego i przesłaniu go do kolejnej jednostki.



Rys. 1. Schemat budowy komórki nerwowej [19].



Rys. 2. Schemat przedstawiający synapsę chemiczną [16].

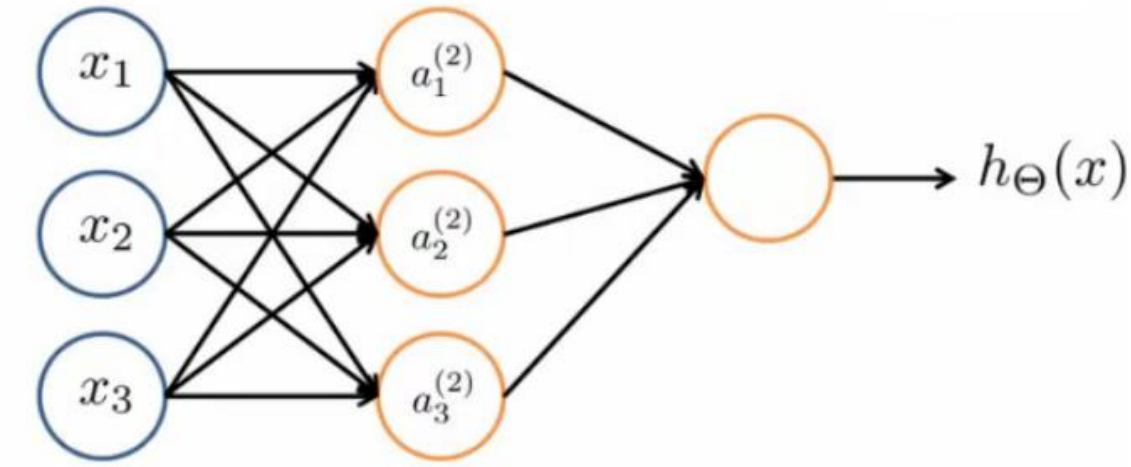
2.2.2. Proces uczenia sztucznej sieci neuronowej

Proces uczenia sieci neuronowej można w skali mikro zobrazować poprzez porównanie do biologicznego odpowiednika – komórki nerwowej. Posługując się tą analogią, neuron jest jednostką obliczeniową, która na wejście przyjmuje pojedynczy, lub wiele sygnałów (poprzez połączenia w biologii nazywane dendrytami), sumuje je i przetwarza, porównując do funkcji aktywacji danego neuronu, a następnie przesyła do kolejnego elementu sieci. Najprostsza możliwa do utworzenia sieć neuronowa składa się właśnie z pojedynczej jednostki, której wyjście odpowiada również za wyjście całej sieci. Oprócz cech wejściowych, jednostka nerwowa może posiadać tzw. bias i wagę przypisaną dla każdego wejścia. Wagi służą zmianie wartości sygnału wejściowego i ocenie jak ważna jest dana cecha. Wagi są regulowane przez sieć w procesie propagacji wstecznej. Bias, czyli dodatkowa stała mająca na celu przesunięcie funkcji aktywacji w przestrzeni, dzięki czemu neuron powinien uczyć się wektorów, które standardowo zostały by odrzucone w procesie porównywania sygnału do funkcji aktywacji. Wynika z tego, ze dodatkowe wejście bias pozwala na zwiększenie zdolności uczenia się neuronu, kosztem ilości wykonywanych obliczeń [17]. Wspomniana wcześniej funkcja aktywacji jest kluczowym krokiem w procesie przetwarzania danych przez sieć neuronową, ponieważ jej wynik decyduje, czy sygnał zostanie przesłany do kolejnej warstwy sieci. W związku z tym każdy węzeł sieci neuronowej można rozpatrywać jak prosty model regresyjny przetwarzający dane w skali mikro. Działanie pojedynczego bloku można opisać ogólnym wzorem:

$$output = f(x) = \begin{cases} 1, & \sum_{i=1}^{m} w_i x_i + bias \geq 0 \\ 0, & \sum_{i=1}^{m} w_i x_i + bias < 0 \end{cases} \quad (1)$$

gdzie w_i to wagi kolejnych wejść, a x_i to kolejne cechy sygnału. W zależności od potrzeb, blok aktywacyjny f(x) może być opisany różnymi funkcjami. Do najbardziej popularnych funkcji aktywacji można zaliczyć: liniową, skoku jednostkowego, a także funkcje nieliniowe. Wśród nich można wymienić: sigmoidalną, tangens hiperboliczny, a także aktualnie najczęściej wykorzystywaną w praktyce – funkcję ReLU (ang. *Rectified Liniear Unit*) oraz jej różne odmiany. Swoją popularność może zawdzięczać bardzo ważnej własności, wedle której nie istnieje możliwość aktywowania wszystkich neuronów warstwy w tym samym czasie. Jest to związane z faktem, iż tylko sygnały wejściowe i wartości powyżej 0 pozwolą na aktywację. W związku z tą cechą, ReLU jest znacznie bardziej wydajna niż np. funkcja sigmoidalna lub tangens hiperboliczny, lecz czasami prowadzi do tworzenia się „martwych” neuronów, które nie reagują na żadne wejścia.

W przypadku, gdy sieć posiada wiele warstw, każda z nich może charakteryzować się osobną funkcją aktywacji. Wśród aktualnie najpopularniejszych architektur, najczęściej wykorzystuje się funkcję ReLU jako aktywację warstw ukrytych i liniową, bądź Softmax dla warstw wyjściowych.



Rys. 3. Schemat prostej, wielowarstwowej sieci neuronowej z warstwami: wejściową, ukrytą i wyjściową [23].

2.3. Rodzaje sztucznych sieci neuronowych

Do tej pory opis działania sztucznych sieci neuronowych został przedstawiony jedynie na najprostszym przykładzie, który w praktyce jest wykorzystywany bardzo rzadko – perceptronie. W historii zagadnień związanych ze sztuczną inteligencją wyróżnia się 3 główne generacje, które zbudowały podłoża do aktualnego stanu wiedzy i zaawansowania algorytmów inteligencji obliczeniowej [18]. Każda z generacji jednakowo stara się naśladować wielowarstwową strukturę ludzkiego mózgu wraz z jej wysoce rozwiniętymi połączeniami, lecz zachowanie neuronów różni się znacząco pomiędzy nimi.

Do pierwszej generacji zalicza się perceptron, o którym pierwsze wzmianki pojawiały się już w 1957 roku. Perceptron charakteryzuje się binarnością sygnału wyjściowego. Oznacza to, iż wartość na wyjściu może być równa 0, bądź 1 i jest otrzymywana poprzez proste progowanie ważonego wejścia synaptycznego.

Drugą generację rozpoczynają wszelkie architektury wielowarstwowe (uczenie głębokie), w których sygnał jest transformowany w sposób nieliniowy za pomocą funkcji takich jak ReLU, sigmoidalna, bądź tangens hiperboliczny. Są to aktualnie najczęściej stosowane modele i sprawdzają się świetnie do procesowania statycznie zgromadzonych danych.

Trzecia generacja zbiera w sobie algorytmy będące dopiero w fazie badań i wyszukiwania najlepszych rozwiązań. Mowa tutaj o sieciach impulsowych, które w sposób jeszcze bardziej dokładny odtwarzają działanie ludzkiego neuronu. Doskonale sprawdzają się do przetwarzania danych czasowych, które wymagają procesowania w

czasie rzeczywistym. Taki rodzaj neuronów koduje dane w formie binarnych zdarzeń, nazywanych inaczej impulsami, w konkretnej chwili czasowej.

Podział ze względu na poziom zaawansowania wykorzystanych algorytmów i sposób reprezentacji sygnału wyjściowego, to tylko jeden z możliwości sklasyfikowania sztucznych sieci neuronowych. Zastosowanie sieci i sposób, w jaki może zostać ona wykorzystana jest kolejną opcją na pogrupowanie algorytmów uczenia maszynowego. Do najbardziej wartych uwagi należy zaliczyć: wielowarstwowy perceptron, sieci konwolucyjne, sieci rekurencyjne, sieci krótkiej pamięci i sieci impulsowe [19].

2.3.1. Perceptron

Działanie perceptronu wielowarstwowego, najczęściej określanego mianem sieci głębokich, zostało już dokładnie omówione powyżej, jednak konieczne jest zwrócenie uwagi, iż pomimo nazwy, w rzeczywistości zbudowane są z neuronów sigmoidalnych (lub zdefiniowanych inną funkcją nieliniową). Dane do takich modeli są zwykle wprowadzane w celu ich wytrenowania. Najlepiej sprawdzają się w zadaniach widzenia komputerowego, przetwarzania mowy ludzkiej lub w zadaniach regresyjnych i klasyfikacyjnych.

2.3.2. Konwolucyjne sieci neuronowe

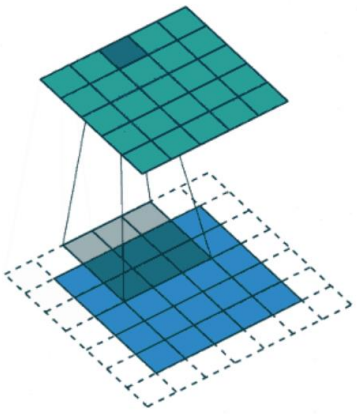
Sieci konwolucyjne, czyli jedne z pierwszych modyfikacji standardowego perceptronu, pierwszy raz przedstawione już w 1980. Za pioniera tego typu architektury można uznać opisaną już w rozdziale 2.1. sieć Neocognitron. Sieci konwolucyjne są podobne do klasycznych sieci neuronowych, pod tym względem że tak samo zbudowane są z neuronów dążących do samodzielnej optymalizacji poprzez proces uczenia. Swoją nazwę bierze z matematycznej operacji splotu (konwolucji) dwóch funkcji lub macierzy, dających na wyjściu zmodyfikowaną wersję swoich poprzedników [20, 21]. Zastosowanie znajduje najczęściej w zadaniach z dziedziny wizji komputerowej: segmentacja obrazu, wykrywanie schematów, a także klasyfikacja i wykrywanie obiektów. Oprócz wymienionych przykładów, z algorytmami konwolucyjnymi można się również spotkać w problemach przetwarzania mowy ludzkiej. Jedną z głównych cech charakterystycznych sieci konwolucyjnych, jest otrzymywanie abstrakcyjnych cech obrazów, wraz z przechodzeniem sygnału wejściowego przez kolejne warstwy. Cechy ekstrahowane są od ogólnych, takich jak krawędzie, do najbardziej szczegółowych (znaki szczególne na twarzy) [20]. Dzięki tej funkcjonalności, poniekąd naturalnie, sieć jest w stanie segmentować obraz i wykrywać poszczególne obiekty na nim przedstawione.

Nazwa nadana takiej architekturze nie jest przypadkowa, ponieważ konwolucje rzeczywiście są podstawą działania tego rodzaju sieci. W zagadnieniach analizy obrazu, konwolucje odnoszą się do mechanizmów filtrowania, czyli operacji iloczynu skalarnego pomiędzy macierzą filtra, a pikselami obrazu źródłowego, w wyniku której uzyskiwany

jest nowy, przekształcony obraz. Wartość każdego piksela obrazu wyjściowego jest obliczana na podstawie wartości pikseli w sąsiedztwie odpowiadającego mu piksela w obrazie wejściowym [22]. Analogiczne operacje zachodzą w procesie uczenia sieci konwolucyjnej. Dane wejściowe są filtrowane za pomocą macierzy o określonym rozmiarze. W tym przypadku, wartości filtra zachowują się jak waga w klasycznych sieciach głębokich, natomiast struktura dąży do ich zoptymalizowania wraz z każdą kolejną iteracją procesu uczenia. Rozmiar filtra lub inaczej rozmiar jądra jest jednym z hiperparametrów określanych przez projektanta przy budowie struktury, zanim przystąpi do procesu uczenia. Wybór tego parametru ma ogromny wpływ na zadanie klasyfikacji obrazu. Filtry o małych rozmiarach są w stanie wydobyć z danych wejściowych znacznie większą ilość informacji, pozwalając na tworzenie głębszych, bardziej dokładnych architektur. Z drugiej strony, im większe filtry tym sieć będzie się szybciej uczyć i lepiej generalizować [24]. W większości sytuacji jest mniejszy od wymiarów początkowych obrazu wejściowego, co rodzi problem redukcji wymiarowości. Aby temu przeciwdziałać, wprowadzono mechanizm wypełnienia (ang. *padding*), kolejnego hiperparametru, na który wpływ ma architekt sieci. Wypełnienie definiuje rozmiar sztucznego obramowania próbki. Oznacza to, że na krawędziach badanej macierzy dopisywane są cechy o stałej wartości (najczęściej 0), które po operacji przetwarzania w neuronie mają niewielki wpływ na wynik, ale pozwalają na utrzymanie takiego samego rozmiaru na wyjściu z bloku konwolucyjnego. Rozmiar obrazu wyjściowego można obliczyć posługując się wzorem (2):

$$O = 1 + \frac{N - F}{s} \tag{2}$$

, gdzie *N* to rozmiar obrazu wejściowego, *F* – rozmiar filtra, a *s* to krok (ang. *stride*) z jakim porusza się okno filtru wzdłuż macierzy z cechami wejściowymi. Posługując się wzorem (2) można z łatwością określić jakiego wypełnienia należy użyć. Krok s, również jest jednym z głównych parametrów przypisywanym konkretnej warstwie. Rysunek 4 przedstawia wizualizację mechanizmu konwolucji, wraz z wypełnieniem o wartości jeden.



Rys. 4. Schemat działania mechanizmu konwolucji [23].

Opisane dokładnie mechanizmy konwolucyjne wykorzystywane są tylko w jednym z rodzajów warstw budujących sieć CNN. Oprócz warstw konwolucyjnych, sieci składają się również z warstw: wejściowej, łączącej (ang. *pooling layer*), porzucania (ang. *dropout*) i spłaszczającej (wyjściowej).

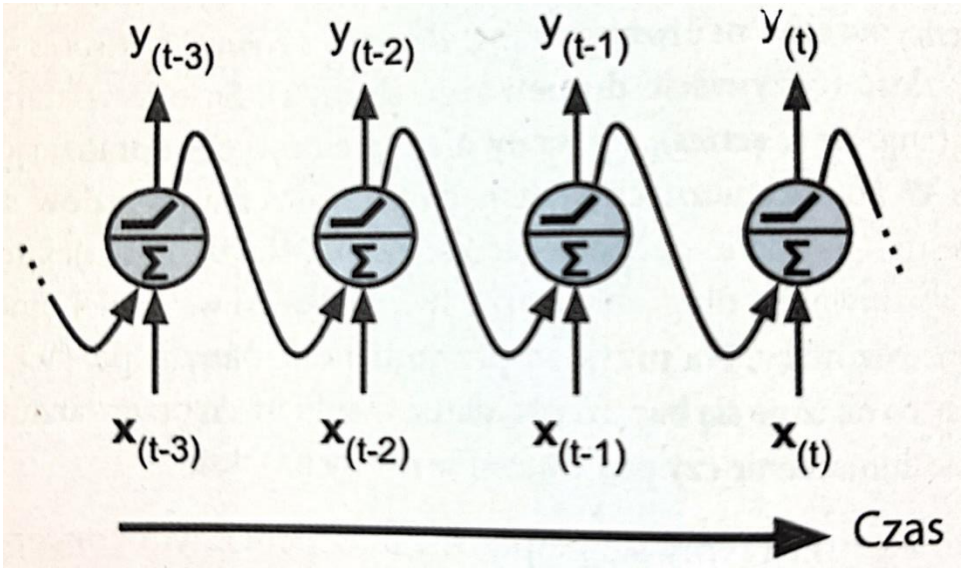
- Wejście sieci najczęściej reprezentowane jest przez klasyczne neurony, które mają na celu jedynie przechowywać dane początkowe w formie wektora cech.
- Idea warstw łączących jest bardzo podobna do mechanizmu filtrowania obecnego w segmencie konwolucyjnym. Poprzez dokonywanie lokalnej konwersji danych, zmniejsza się wymiarowości wyjścia. W oknie o określonym rozmiarze dokonuje się wybranej operacji, a następnie przesuwa się je do kolejnej ramki. W warstwie łączącej krok jest równy wielkości okna. Najczęściej wykorzystuje się łączenie wybierające z przetwarzanego obszaru wartość maksymalną, lub średnią.
- Porzucanie polega na losowym ustawieniu wychodzących krawędzi ukrytych jednostek (neuronów tworzących ukryte warstwy) na 0 przy każdej aktualizacji fazy treningu. Dzięki temu co każdą iterację procesu uczenia, zostaje wyłączone losowe połączenie. Mechanizm ten jest najczęściej wykorzystywany do pozbycia się problemu przetrenowania sieci.
- Warstwa spłaszczająca jest ostatnim, jednak bardzo istotnym krokiem, przez który muszą przejść dane pokonując sieć. Blok ten przekształca wielowymiarowe macierze do pojedynczego wektora, umożliwiając algorytmowi dokonanie procesu klasyfikacji.

Podobnie jak w przypadku klasycznych głębokich sieci neuronowych, architektury konwolucyjne wykorzystują nieliniowość do osiągania dobrych wyników uczenia [20]. Do najczęściej wykorzystywanych funkcji aktywacji wykorzystuje się więc bloki ReLU, przeciekające ReLU i wykładniczo-liniową (ang. *ELU*) [24]. Funkcja ReLU została już opisania, jako jedna z popularniejszych i najwydajniejszych ze względu na brak możliwości otrzymania wartości niedodatnich w wyniku procesowania przez nią. Rodzi to jednak problem, w którym raz nieaktywowane neurony, już nigdy się nie aktywują, ponieważ optymalizacja gradientowa może nigdy nie zredefiniować wag tego neuronu. Najbardziej popularne modyfikacje starają się wyeliminować ten problem poprzez pochylenie części stałej (dziedziny, w której każdy podany argument zostanie wyzerowany), tak aby umożliwić otrzymywanie skompresowanych, negatywnych wartości. W ten sposób, przeciekająca ReLU wprowadza liniowe nachylenie, którego kąt nachylenia jest jednym z hiperparametrów bloku aktywującego, natomiast funkcja ELU eliminuje stały fragment na rzecz funkcji wykładniczej. Takie udoskonalenia pozwalają na osiągnięcie szybszego stopnia uczenia i lepszej precyzji w zadaniach klasyfikacyjnych [24, 23].

2.3.3. Rekurencyjne sieci neuronowe

Do tej pory omawiane typy sieci neuronowych służyły głównie do radzenia sobie z zadaniami klasyfikacyjnymi i regresyjnymi. Rekurencyjne sieci neuronowe poniekąd łączą te dwa zagadnienia, operując na ogromnych zbiorach danych czasowych i sekwencyjnych, jednocześnie umożliwiając uczenie się cech i zależności długoterminowych [26]. W ogólnym rozrachunku są w stanie pracować na sekwencjach danych wejściowych o dowolnej długości, w przeciwieństwie do omawianych wcześniej sieci, w których wejście ma określony rozmiar. Najczęściej zastosowanie znajdują np. w systemach predykcyjnych, oprogramowaniu do pojazdów autonomicznych, czy algorytmach przetwarzania mowy ludzkiej. Ideę tę umożliwia specyficzna budowa pojedynczych komórek sieci, które łączą wiele bloków nieliniowych, pomiędzy którymi występuje przynajmniej jeden kanał ze sprzężeniem zwrotnym [28]. W odróżnieniu do klasycznych, jednokierunkowych sieci neuronowych, w których pobudzenia przepływają tylko w kierunku od wejścia do wyjścia, w RNN wykorzystywany jest mechanizm połączeń wstecznych.

Najprostszy model sieci RNN może składać się nawet z pojedynczego neuronu, który otrzymuje sygnały wejściowe, przetwarza je, a następnie sygnał wyjściowy przesyła z powrotem na początek komórki kanałem wstecznym. Proces generowania sprzężenia zwrotnego nazywany jest taktem *t*, bądź ramką czasową, i w trakcie jego trwania, neuron rekurencyjny otrzymuje sygnały wejściowe $x_{(t)}$ oraz wynik generowany przez siebie w poprzednim takcie $y_{(t-1)}$[26]. Wizualizację tego procesu przedstawiono na rysunku 5. Jednak, podobnie jak w przypadku klasycznego perceptronu, sieć zbudowana jedynie z pojedynczego neuronu nie będzie sprawować się dobrze w rozwiązywaniu rzeczywistych problemów. Analogicznie, możliwe jest utworzenie warstwy złożonej z wielu neuronów, w której mechanizm sprzężenia zwrotnego w żaden sposób nie utrudnia procesu uczenia. W każdym takcie *t*, każdy neuron otrzymuje zarówno sygnał wejściowy $x_{(t)}$, jak i wynik otrzymany z poprzedniego taktu $y_{(t-1)}$. Z racji otrzymywania dwóch różnych sygnałów, neuron musi być do tego przygotowany poprzez posiadanie zestawu dwóch niezależnych wag; każdej przeznaczonej dla odpowiadającemu jej sygnałowi. Komunikacja zwrotna służy do odkrywania cech z przeszłości, które lokalnie mogą nie mieć znaczącego wpływu na wagę połączenia, lecz posiadają duże powinowactwo z cechami oddalonymi od nich na osi czasu [29]. Jednak, takie przedstawienie działania neuronów w RNN, nie wyjaśnia jak konkretnie komórka rekurencyjna uczy się zapamiętywania sekwencji i przewidywania przyszłych zdarzeń.



Rys. 5. Schemat przetwarzania sygnału przez neuron rekurencyjny w kolejnych ramkach czasowych [27]

Wspomniane wcześniej zapamiętywanie zdarzeń sprzed kilku taktów, jest najważniejszym aspektem odróżniającym architektury RNN od klasycznych ANN. Fragment sieci neuronowej zachowujący informację z poprzednich stanów nazywany jest komórką pamięci [29]. Stan $h_{(t)}$ komórki w takcie t wyliczany jest jako funkcja danych wejściowych $x_{(t)}$ w tej ramce czasowej i jej stanu $h_{(t-1)}$ z poprzedniego taktu. Jako funkcje aktywacyjne w jednostkach pamięci stosuje się przeważnie funkcje sigmoidalne i tangens hiperboliczny [26, 32]. Podstawowe komórki rekurencyjne umożliwiają jedynie przechowywanie sekwencji długich na około 10 taktów wstecz [27, 28], co w rzeczywistych zadaniach z reguły nie jest wystarczające.

Analogicznie jak w przypadku klasycznych sieci neuronowych, rozwiązywanie bardziej skomplikowanych zadań wymaga od projektanta sieci tworzenia coraz głębszych struktur. Podejście takie można uznać za prawidłowe, jednak nieuniknione jest wystąpienie problemu zanikających gradientów. Jak wykazano w [26], im dłuższe sekwencje (więcej taktów), tym wagi połączeń w sposób naturalny będą maleć z czasem, co przekłada się na wręcz wykładnicze zanikanie gradientów. To zjawisko powoduje, że sieć ignoruje długoterminowe zależności i z trudem uczy się korelacji między odległymi w czasie zdarzeniami. Rozwiązaniem problemu zanikających gradientów w sieciach typu RNN są przykładowo: zaproponowana w 2016 r. przez Lei Ba [31] normalizacja warstwowa lub znana z sieci konwolucyjnych - metoda porzucania [27].

W wyniku przekształceń danych zachodzących podczas ich przepływu przez sieć, w każdym takcie następuje utrata niewielkiej części informacji. Po przetworzeniu odpowiedniej ilości ramek czasowych, sieć nie przechowuje już informacji nagromadzonych na początku procesu uczenia. Rozwiązaniem na ten problem są zaproponowane na późniejszych etapach badań nad sieciami rekurencyjnymi, specjalne komórki pamięci długotrwałej. Okazały się one na tyle skuteczne, że praktycznie całkowicie wykluczyły z użytku podstawowe komórki RNN.

Jako przykład mogą posłużyć komórki długiej pamięci krótkotrwałej (ang. *Long Short-Term Memory, LSTM*) zaproponowane w 1997r. przez Seppa Hochreitera i Jurgena Schmidhuberta [30]. Architektura komórki LSTM jest już znacznie bardziej skomplikowana niż podstawowe jednostki RNN lub NN. Na najwyższym poziomie abstrakcji rozdziela ona wektor stanu na dwa wektory reprezentujące zależności długo $h_{(t)}$ i krótkotrwałe $c_{(t)}$. Podstawowa idea polega na tym, że sieć poprzez analizę stanu krótkotrwałego jest w stanie przefiltrować dane pod kątem tego, co należy zapisać w stanie długotrwałym, odrzucić, bądź co z niego odczytywać. Taki mechanizm jest możliwy dzięki zastosowaniu dodatkowych warstw służących jako kontrolery bramek, które korzystając z logistycznej funkcji aktywacji oceniają istotność kolejnych cech sygnału. W skład komórki wchodzą trzy bramki: zapominająca, wejściowa i wyjściowa. Sygnał stanu długotrwałego przechodząc przez komórkę LSTM poddawany jest operacjom sumowania lub splatania z sygnałem stanu krótkotrwałego, zmodyfikowanego przez kolejne bramki. Dzięki temu $h_{(t)}$ przenosi najważniejsze dane (rola bramki wejściowej), przechowuje je tak długo jak są potrzebne (bramka zapominająca), a także wydobywa je kiedy jest to wymagane (bramka wyjściowa) [27].

2.3.4. Impulsowe sieci neuronowe

Opisanie w poprzednim rozdziale sieci rekurencyjne, opierały swoje działanie o koncept zapamiętywania zależności i sekwencji czasowych, tzn. dane wyjściowe bezpośrednio wynikały ze stanów, które na osi czasu wydarzyły się w przeszłości. Jest to znaczne poruszenie badań nad sieciami neuronowymi bliżej rzeczywistego modelu działania mózgu, który opiera się na pamięci i wyciąganiu zależności z przeszłych zdarzeń. Sieci impulsowe (ang. *Spiking Neural Networks, SNN*) posuwają technologię jeszcze o krok do przodu, wprowadzając do swojej architektury model komórki impulsowej (ang. *Spiking neuron*) wykorzystujących paradygmat czasu oraz zmian stanów neuronów w czasie. Przekazywanie informacji pomiędzy neuronami naśladuje proces przesyłania informacji w biologicznych komórkach nerwowych, tj. poprzez precyzyjne śledzenie czasu i częstotliwości występowania impulsów lub ich całych sekwencji. Liczne badania kory mózgowej wykazały, że chwila, w której komórka nerwowa generuje impuls jest postrzega jako sposób na kodowanie informacji. Przez to jest bardzo ważnym parametrem, na którym opiera się działanie wielu struktur nerwowych [40, 41]. Między innymi z tego też powodu algorytmy próbujące odwzorować takie zachowanie zostały określone mianem trzeciej generacji sieci neuronowych.

Na najwyższym poziomie abstrakcji proces uczenia sieci neuronowej, niezależnie której generacji, polega na dostosowywaniu i optymalizacji wag pomiędzy kolejnymi warstwami sieci. Neurony impulsowe nieznacznie odbiegają od tej reguły, ponieważ sposób ich działania umożliwia występowanie swego rodzaju wiarygodności biologicznej, czerpiącej ze schematów i częstotliwości przetwarzania i przekazywania impulsów. Zdolność ta umożliwia tworzenie połączeń i asocjacji pomiędzy cechami, niemożliwych do osiągnięcia w klasycznych sieciach pierwszej i drugiej generacji [44]. Mówiąc dokładniej, wagi dostosowywane są w oparciu o lokalne pobudzenia sąsiednich

neuronów w konkretnym oknie czasowym. Niekonwencjonalne obliczanie wag synaptycznych jest również głównym czynnikiem utrudniającym budowanie wielowarstwowych, głębokich struktur, ponieważ klasyczny mechanizm propagacji wstecznej nie radzi sobie z nieróżniczkowalnymi wartościami [39].

Sposób, w jaki impuls jest uwalniany, a także jak skoki potencjału i ich częstotliwość są obsługiwane, zależy w dużym stopniu od architektury neuronu. Można wyróżnić kilka najistotniejszych przykładów.

2.3.4.1. Model Huxleya

Model neuronu stworzony w 1952 roku przez Huxleya i Hodgkina [42] jest jednocześnie jednym z najlepiej opisującym mechanizmy zachodzące w komórkach biologicznych. Wyraźnie opisuje zachowania na poziomie subkomórkowym, membranowe generowanie prądów biologicznych i propagację skoków neuronowych. Jednak, właśnie ta dokładność wyklucza go z rozważań nad projektowaniem struktur głębokiej sieci i próby komputerowej symulacji w makro skali.

2.3.4.2. Model Izhikevicha

Znacznie uproszczona względem modelu Hodgkina-Huxleya, przy jednoczesnym zachowaniu najważniejszych funkcjonalności, architektura Izhikevicha umożliwiła implementowanie neuronów impulsowych w sposób obliczeniowo wydajny. Neuron Izhikevicha jest w stanie odtworzyć biologiczny mechanizm skoku i wysłania impulsu w sposób analogiczny do znanych typów komórek nerwowych w korze mózgowej [43].

Tym, co odróżnia modele III generacji, a w szczególności neuron Izhikevicha, od standardowych neuronów sieci głębokich, jest brak różniczkowalnej funkcji aktywacji. Każdy neuron posiada próg, po przekroczeniu którego impuls zostaje rozprowadzony. W zależności od częstotliwości, siły sygnałów docierających do neuronu, próg aktywacji zmienia się.

Popularność algorytmu, motywowała wielu naukowców do wprowadzania usprawnień i badania zachowań symulowanego neuronu w określonych warunkach. W szczególności prace badawcze skupiały się na mechanizmie wytwarzania impulsu i wykrywaniu schematów według, których sygnały są przekazywane [44, 45].

2.3.4.3. Przeciekający neuron „całkuj i strzelaj” / Neuron LiF

Model LIF (ang. *Leaky Integrate-and-Fire*) jest zaklasyfikowana jako modyfikacja najprostszego algorytmu symulującego działanie generowania skoków sygnału (iglic) – modelu Lapicque’a [46]. Ze względu na swoją prostotę i wysoką wydajność, model LIF cieszy się największą popularnością wśród algorytmów symulujących działanie neuronów biologicznych. Opisuje on potencjał membranowy neuronu pod kątem wejść

synaptycznych i impulsów, które otrzymuje. Potencjał czynnościowy - przekazanie impulsu - jest generowany, gdy potencjał błonowy osiąga określony próg, jednak rzeczywiste zmiany związane z napięciem błonowym i przewodnictwem napędzającym potencjał czynnościowy nie stanowią części modelu. Stąd określenie modelu jako znacznie uproszczona wersja architektury Hodgkina i Huxleya. Model wykorzystuje fakt, że potencjały czynnościowe neuronu biologicznego mają zawsze w przybliżeniu ten sam kształt. Jeśli kształt potencjału czynnościowego jest zawsze taki sam, to nie może być wykorzystany do przekazywania informacji; informacja zawarta jest raczej w obecności lub braku pobudzenia. Z tego powodu, potencjały czynnościowe są zredukowane do "zdarzeń", które mają miejsce w ściśle określonym momencie. Modele LIF mają dwa oddzielne parametry, które są niezbędne do zdefiniowania ich dynamiki: po pierwsze, równanie opisujące ewolucję potencjału membranowego; i po drugie, mechanizm generowania zdarzeń [47]. W przypadku omawianego modelu do opisu tych dwóch własności wykorzystuje się, odpowiednio: liniowe równanie różnicowe korzystające z podstawowych praw teorii elektryczności i stałą wartość progu [47].

Proces przetwarzania sygnału polega na ładowaniu neuronu kolejnymi sygnałami; impulsami, które pobudzają komórkę i w zależności od wagi danego połączenia, dodają odpowiednią wartość do potencjału membranowego. Jeśli nie poddawany działaniu kolejnych impulsów, potencjał maleje w czasie, aktywując zjawisko relaksacji; stąd w nazwie modelu pojawia się słowo „przeciekający” (ang. *leaky*) [50]. Kiedy suma sygnałów przekroczy ustalony próg, dochodzi do wydzielenia impulsu wyjściowego przez neuron (ang. *integrate and fire*). Moment ten jest jednym z najważniejszych parametrów uzyskiwanych z sieci SNN i statystyka kolejnych takich zdarzeń i interwałów pomiędzy nimi, daje wiele informacji o sposobie działania neuronów impulsowych [50]. Natychmiast po wystrzale sygnału, wewnętrzny stan neuronu zostaje zresetowany i przechodzi w fazę refrakcji, czyli uśpienia. W tym okresie przyjmowanie nowych sygnałów jest niemożliwe, a potencjał powoli zwiększa się dążąc do osiągnięcia stanu spoczynkowego. Opisane stadia potencjału komórki, wspólnie z momentami wystąpienia zdarzenia przekazania impulsu wyjściowego, są odpowiedzialne za mechanizm kodowania informacji obsługiwany przez neurony impulsowe [33].

Podobnie jak w przypadku modelu Izhikevicha, neurony LIF wykorzystywane są w głównie w badaniach nad schematami i zachowaniami impulsów przemieszczających się po sieci połączeń neuronowych [48, 49]. Badania nad sieciami impulsowymi prowadzone są w wielu dziedzinach, w których aktualny prym wiodą poprzednie generacje architektur neuronowych: konwolucyjne i rekurencyjne. Przede wszystkim z SNN wykorzystywane są w pracach z zakresu wizji komputerowej (konkretnie wykrywania schematów), rozpoznawania mowy, a także w diagnostyce medycznej [36, 37, 38, 39].

Jako praktyczny przykład architektury wykorzystującej mechanizm neuronów impulsowych, można zaproponować sieć APNN opisaną w między innymi w [35, 51].

2.4. Asocjacyjna impulsowa sieć neuronowa

Sieci APNN (Asocjacyjne Pulsujące Sieci Neuronowe) rozszerzają działanie klasycznych sieci impulsowych, wprowadzając implementację mechanizmu bio plastyczności do wykrywania asocjacji pomiędzy danymi uczącymi [53]. Asocjacja, jest metodą eksploracji danych wykorzystywaną głównie w dziedzinach uczenia maszynowego i sztucznej inteligencji. Polega na analizie zbioru cech w celu znalezienia powtarzających się w nim zależności [52]. Najbardziej znanym przykładem wykorzystania technik asocjacyjnych jest tzw. problem analizy koszyka zakupów, której celem jest znalezienie naturalnych wzorców zachowań konsumenckich klientów poprzez analizę najczęściej kupowanych razem produktów. W modelu biologicznym, tworzenie asocjacji jest zazwyczaj rezultatem odczuwania wrażeń, będących wynikiem pobudzenia receptorów zmysłowych. Symulacja takiego mechanizmu daje znaczną przewagę sieci APNN nad siostrzanymi algorytmami III generacji, a także stanowi kolejny krok w kierunku lepszego, komputerowego oddawania działania rzeczywistych neuronów [53].

2.4.1. Asocjacyjny neuron impulsowy

Implementacja architektury APNN wymaga wprowadzenia nowatorskiego rodzaju neuronu, opierającego swoje działanie na koncepcie czasu rzeczywistego i obsłudze kolejnych zdarzeń z uwzględnieniem linii czasowej. Asocjacyjny neuron impulsowy (ang. *Associative Pulsing Neuron, APN*) stanowi próbę bardziej trafnej symulacji biologicznych jednostek nerwowych, których rdzeniem działania jest wytwarzanie asocjacji i relacji w oparciu o długość pobudzenia i chwile, w których nastąpiły [55]. Pomimo swojej próby zbliżenia się do realnych neuronów, APN rezygnuje z implementacji wielu właściwości i procesów zachodzących w neuronach; nawet takich, które można było znaleźć w innych architekturach wpisujących się w ramy algorytmów III generacji, np. potencjał błonowy w modelu Izhikevicha.

W modelu asocjacyjnego neuronu, czas jest najważniejszym parametrem, na podstawie którego sieć uczy się relacji i opracowywane są wyniki z badań architektury. Aby było to możliwe, neurony zapisują informację o lokalnych różnicach pomiędzy chwilami, w których nastąpiła aktywacja i przekazanie impulsu nerwowego, a także jaki czas minął od tego zdarzenia do aktywacji sąsiednich neuronów. Właśnie te pozornie proste dane służą ocenie siły połączenia synaptycznego, a co za tym idzie wagi korygującej siłę sygnału. W procesie tworzenia sieci i jej uczenia, neurony nie tylko stymulują połączenia pomiędzy sobą, ale również są w stanie tworzyć krawędzie z wartościami podobnymi lub sąsiednimi. Taki mechanizm można określić mianem plastyczności sieci, występującej również w rzeczywistych strukturach nerwowych.

Kumulacja wartości pobudzenia w czasie bezpośrednio przekłada się na stan, w którym w danej chwili znajduje się analizowany neuron. Wyróżnia się 4 główne stadia:

- **Spoczynek**

  Stan, w którym neuron oczekuje na przyjęcie sygnału wejściowego. Wartość pobudzenia znajduje się na stałym, bazowym poziomie.

- **Ładowanie**

  Po otrzymaniu sygnału wejściowego neuron przechodzi w stan ładowania, w trakcie którego stopniowo buduje wartość pobudzenia. Długość, z jaką komórka jest pobudzana, zależy od siły połączenia synaptycznego i jest równa odwrotności wagi przypisanej do neuronu.

- **Relaksacja**

  Po ustąpieniu procesu ładowania, jeśli żaden zewnętrzny bodziec nie pobudzi komórki, neuron przechodzi w stan relaksacji, czyli samoistnego osłabiania wartości pobudzenia, aż do osiągnięcia poziomu spoczynku. Długość występowania tego stanu można określić jako wielokrotność aktualnej wartości pobudzenia komórki, przy założeniu, że mieści się ono w zakresie <0; 1>.

- **Refrakcja**

  Refrakcja następuje zaraz po wygenerowaniu przez komórkę sygnału wyjściowego. W jej trakcie wartość pobudzenia momentalnie spada poniżej poziomu bazowego. Następnie, neuron przechodzi w etap relatywnej refrakcji, czyli samoistnego ładowania, aż do osiągnięcia fazy spoczynku. Na żadnym z etapów refrakcji neuron nie może przyjmować impulsów wejściowych.

Wszystkie omówione stany, dokładne czasy rozpoczęcia i trwania są koordynowane lokalnie, poprzez mechanizm wewnętrznej kolejki procesów (*ang. Internal Process Queue, IPQ*). IPQ sortuje i przypisuje kolejnym stanom chwile czasowe, w których dane zdarzenia powinny mieć miejsce. Opisany mechanizm jest implementowany przez każdy z neuronów osobno i jest bardzo ważnym elementem struktur APN. [51, 52, 53]

2.4.2. Działanie sieci APN

Asocjacyjne sieci impulsowe są dynamicznie tworzonymi sieciami neuronowymi, składającymi się z sieci jednostek neuronów APN, a także połączeń pomiędzy nimi. Połączenia i przypisane do nich wagi reprezentują asocjacje pomiędzy kolejnymi atrybutami całej struktury. Oprócz neuronów, w skład architektury wchodzą również elementy nazwane receptorami [51]. Receptory są odpowiedzialne za przekształcenie danych uczących sieć. W [51] do tej operacji wykorzystano opisane w [54] AVB drzewa, jednak można je zastąpić innymi strukturami umożliwiającymi szybkie przeszukiwanie i sortowanie danych. Ponadto, receptory tworzą wewnętrzne połączenia z neuronami obiektowymi przekazując początkowy sygnał, wytworzony po wprowadzeniu nowej

wartości wejściowej. Ostatnią cechą receptorów jest ich zdolność do reprezentowania wartości zbliżonych, podobnych, zwłaszcza w przypadku danych liczbowych [51].

Każdy neuron reprezentujący cechę danych wejściowych, w zależności od stopnia i długości pobudzenia odbieranego z receptorów, w danej chwili czasowej może się znajdować w jednym z czterech stanów: oczekiwania na pobudzenie, ładowania, relaksacji i refrakcji. Jak zostało opisane w rozdziale 2.4.1. neuron samodzielnie nadzoruje i planuje czas wystąpienia zmiany stanu. Jednak, pomimo zdolności do planowania zdarzeń lokalnie, neuron nie ma możliwości komunikowania się z innymi obiektami, celem koordynacji stanów. Rozwiązaniem tego problemu okazało się wprowadzenie globalnej kolejki (ang. *Global Event Queue, GEQ*) nadzorującej przebieg zdarzeń, definiowanych przez procesy wewnętrzne każdego neuronu z osobna. Mechanizm został dokładnie opisany w rozdziale 4.2.1., ponieważ podobne rozwiązanie wykorzystano w omawianej, w pracy, strukturze DASNG. GEQ sortuje zdarzenia w czasie, więc neurony są zawsze aktualizowane w odpowiednich momentach, gdy kończą się ich wewnętrzne procesy [53]. Takie podejście umożliwia poprawne i szybkie działanie, minimalną liczbę aktualizacji i stosunkowo dokładne zasymulowanie konceptu czasu, mającego kluczowe znaczenie w realnych strukturach nerwowych.

2.4.3. Przykładowe zastosowania sieci APNN

Architektura złożona z asocjacyjnych neuronów impulsowych świetnie wykorzystuje swoje cechy do rozwiązywania zadań klasyfikacyjnych. Odnajdywanie relacji pomiędzy poszczególnymi cechami danych wejściowych pozwala w dokładny sposób grupować i określać przynależność nowych danych do konkretnych klas przedstawionych w zbiorze uczącym. Testy opisane w [34] wskazują, że sieć APNN jest w stanie osiągać precyzję zbliżoną lub nawet przekraczającą najlepsze i najczęściej wykorzystywane klasyfikatory tj.: drzewa decyzyjne, maszyny wektorowe czy naiwne klasyfikatory Bayesa.

Ponadto, w [53] przedstawiono skuteczną próbę zamodelowania pamięci semantycznej. Model był w stanie przypisywać konkretne sekwencje i skojarzenia do pojedynczych słów, tworząc swego rodzaju symulację długotrwałej pamięci deklaratywnej.

Oprócz klasycznych zastosowań, w których wyspecjalizowane są również inne, znacznie prostsze w implementacji algorytmy, sieci APNN i ich główna składowa – asocjacyjne neurony impulsowe – w dużej mierze przyczyniają się do rozwoju wiedzy na temat działania ich biologicznego, rzeczywistego odpowiednika. Sposób działania, komunikacja pomiędzy neuronami, schematy według których dane wejściowe są procesowane, to główne mechanizmy, na których swoje funkcjonowanie opierają komórki nerwowe, a o których zawiłościach można wysnuwać wnioski dzięki analizie sieci impulsowych.

3. Struktury danych

Kiedy aplikacja korzysta z danych, nawet w najmniejszej ilości, potrzebuje sposobu na przechowywanie i manipulowanie nimi, z pomocą przychodzą struktury danych. Struktury danych są wyspecjalizowaną formą algorytmów podchodzącą do problemu procesowania danych od strony, z jakiej mogą być reprezentowane i przetwarzane.

Termin struktura jest używany w wielu różnych dziedzinach do oznaczenia obiektów, które są zbudowane ze swoich składników w regularny i charakterystyczny sposób. W prostszych słowach, struktura danych to architektura, której składnikami są obiekty danych. W informatyce termin ten jest używany bardziej precyzyjnie, gdyż struktura danych jest zbiorem wartości danych, relacji między nimi oraz funkcji lub operacji, które można zastosować do danych. Jeśli brakuje którejkolwiek z tych trzech cech lub nie jest ona dokładnie określona, badana struktura nie kwalifikuje się jako struktura danych.

Klasyczne podejście do budowania struktur danych zakłada tworzenie połączeń pomiędzy kolejnymi komórkami odpowiedzialnymi za przetrzymywanie kolejnych obiektów danych. Inaczej pojedynczą komórkę można nazwać węzłem (ang. *node*), z których każda posiada cechy opisujące przetrzymywaną wartość dowolnego typu, t.j. bit, liczba całkowita, wskaźnik przetrzymujący referencję do innego obiektu, itd. W większości przypadków dane przetrzymywane w pojedynczej strukturze powinny być tego samego typu, jednak istnieją wyjątki od tej reguły. Dostęp do struktury i jej elementów odbywa się na różne sposoby, np. poprzez indeksowanie, lub specjalne węzły wejściowe. O strukturach, w których dane są połączone między sobą, można myśleć jak o grafie skierowanym, którym pierwszy element posiada wskaźnik do drugiego elementu, drugi element posiada wskaźnik do trzeciego itd. Przykładem opisanej struktury może być np. lista łączona.

Do najbardziej popularnych operacji implementowanych przez najprostsze struktury danych można zaliczyć: dostęp do wskazanego elementu i aktualizacja, na którą składa się dodawanie i usuwanie [58]. Operacje wyszukiwania przetrzymywanego obiektu w większości implementacji wymagają przeszukania przynajmniej części struktury, aby finalnie dotrzeć do pożądanego celu. Za wyjątek można uznać mapy, które umożliwiają szybki dostęp do każdego elementu, o ile znany jest klucz, do którego dana cecha została przyporządkowana. Jeśli algorytm obsługuje kolejność wstawiania lub nawet sortowania, operacja dodawania jest już nieco bardziej zaawansowana, ponieważ nierzadko wymaga przeprowadzania na zmianę procesów wyszukiwania i przebudowywania struktury. W niektórych implementacjach, t.j. kolejki lub stosy, dane są dodawane na końcu struktury, co wymaga jedynie przetrzymywania informacji o pierwszym i ostatnim elemencie. Metody wchodzące w skład pojęcia aktualizacji mogą się więc różnić w zależności od rodzaju struktury; od najbardziej trywialnych jak utworzenie pojedynczego, nowego obiektu, do usuwania, które wymaga przepinania wskaźników i rebalansowania struktury, aby utrzymać status posortowania danych.

Jako najczęściej wykorzystywane struktury danych można zaliczyć: tablice, listy, mapy, kolejki, drzewa i stosy.

3.1. Tablice

Tablica to liniowa struktura danych, która zbiera elementy tego samego typu i przechowuje je w przyległych i sąsiadujących ze sobą miejscach pamięci. Tablice działają na systemie indeksów od 0 do (*n*-1), gdzie *n* jest rozmiarem tablicy. Najczęściej podczas inicjalizowania tablicy, konieczne jest zadeklarowanie jej rozmiaru, a także typu danych jakie będzie przechowywać. Struktura może przyjmować różną wymiarowość, dzięki czemu możliwe jest przechowywanie nawet najbardziej skomplikowanych wartości z różnymi zależnościami pomiędzy nimi. Operacje jakie można przeprowadzać na tablicach to: przejście wzdłuż, aktualizacja, wyszukiwanie i sortowanie. Każda z operacji korzysta z mechanizmu iterowania, czyli przechodzenia po kolejnych elementach za pomocą przypisanych do nich indeksów. Procesy dodawania i odejmowania, wymagają aktualizacji indeksów i przesunięciu istniejących już wskaźników w przód bądź w tył. Mechanizmy wyszukiwania i sortowania zależą tylko i wyłącznie od woli projektanta i algorytmów, których ma zamiar użyć do wykonywania tych operacji. Jednak, niezależnie od wyboru aktualizacja struktury jest oparta, ponownie, o posługiwanie się indeksami i odpowiednie modulowanie nimi [58].

Tablice są najprostszą strukturą danych wykorzystywaną w każdej aplikacji i algorytmie. Przez konieczność zdefiniowania rozmiaru tablicy, w łatwy sposób można zoptymalizować przepływ pamięci i zapobiegać wyciekom pamięciowym. Ich prostota pozwala w wydajny i efektywny sposób przetrzymywać dane w formie tabelarycznej. Do największych minusów tablic można zaliczyć ich statyczną naturę, co uniemożliwia modyfikowanie uprzednio zdefiniowanego rozmiaru. Dodatkowo, wstawienie w dowolnym miejscu innym niż sam koniec tablicy jest kosztowną operacją, ponieważ wymaga przesunięcia wszystkich wartości od punktu wstawienia do końca tablicy. Analogicznie sytuacja wygląda w przypadku usuwania wartości ze środka struktury, ponieważ wymaga przesunięcia wszystkich elementów z punktu usunięcia na sam koniec listy. Większą swobodę w tym zakresie umożliwiają listy. [58, 66]

3.2. Listy

Lista łączona jest najczęściej wykorzystywaną strukturą danych, do obsługi dynamicznie zmiennych elementów danych. Każda lista składa się z pojedynczego elementu zwanego inaczej węzłem. W skład każdego takiego obiektu wchodzą dwa główne atrybuty: pierwszy przetrzymuje informację o danych, natomiast drugi przechowuje referencję do adresu kolejnego węzła [58]. W przeciwieństwie do opisanej wyżej tablicy, lista połączona w bardzo wygodny sposób może się powiększać i kurczyć. Magazynowane wartości nie są przechowywane w kolejnych miejscach pamięci, więc duży blok pamięci ciągłej nie jest wymagany nawet do przechowywania znacznych ilości danych. Każdy węzeł struktury wymaga przechowywania dodatkowego wskaźnika. Lista połączona nie może być przeszukiwana za pomocą wyszukiwania binarnego, ponieważ nie ma bezpośredniego dostępu do węzłów. Jednak zarówno wstawianie jak i usuwanie

dowolnego elementu listy (przy założeniu, że pozycja wstawienia/usunięcia jest znana) jest bardzo wydajne i przebiega w stałym czasie.

Pobieranie konkretnego elementu listy bez znajomości wartości poprzedzającej, sprowadza się do iterowania po każdym węźle za pomocą wskaźników, aż do znalezienia pożądanej cechy. W jednostronnie łączonej liście (ang. *single-linked linear list, SLLL*) każdy element posiada swojego unikatowego następcę za wyjątkiem ostatniego węzła, którego wskaźnik jest oznaczany wartością 'NIL'. Inną wariacją opisywanej struktury może być dwustronna lista połączona (ang. *doubly linked list, DLL*), której elementy posiadają dwa wskaźniki do referencji węzła poprzedzającego i następującego. Możliwe jest również wprowadzenie modyfikacji zarówno listy pojedynczej, jak i podwójnej, poprzez połączenie elementu ostatniego z pierwszym, tworząc swego rodzaju pętlę, która może usprawniać niektóre zadania. Operacje dodawania i usuwania sprowadzają się do utworzenia nowej komórki listy, a następnie aktualizacji wskaźników istniejących elementów o wskaźnik do świeżo powstałego węzła.

Listy łączone są wykorzystywane jako pojedyncze struktury do przetrzymywania danych w trakcie wykonywania algorytmu, które świetnie radzą sobie z dynamicznym lokowaniem pamięci. Oprócz tego są również bardzo ważnym elementem, na którego podstawie budowane są nieco bardziej skomplikowane struktury, takie jak: kolejki, stosy, mapy. [58, 66]

3.3. Stosy

Stos (ang. *stack*) jest prostą modyfikacją listy łączonej działającej na zasadach LIFO (ang. *Last In First Out*) – ostatni dodany element zostanie z niej wyciągnięty jako pierwszy. Aby dostać się do konkretnie zamierzonego elementu, konieczne jest usunięcie wszystkich węzłów wprowadzonych do struktury później niż pożądana dana. W związku z tą cechą, tylko wierzchołek struktury jest dostępny z zewnętrznego wywołania – nie można usunąć elementu znajdującego się w środku stosu. Do najważniejszych operacji obsługiwanych przez stos należą: umieszczenie nowej informacji na szczycie struktury, a także zdjęcie istniejącego już elementu ze szczytu.

Opisywany kontener można zbudować na bazie dwóch podstawowych algorytmów struktur danych: tablicy i listy łączonej. Bardziej naturalna implementacja wykorzystuje jako bufor listę łączoną z racji na jej dynamiczną charakterystykę, jednak wykorzystanie prostej tablicy jest równie naturalne. W takim przypadku należy pamiętać, aby zainicjalizowany bufor miał odpowiednio duży rozmiar, a także aby struktura przetrzymywała informację o ostatnim wykorzystywanym indeksie. Operacja pobierania elementu jest wtedy równoznaczna z jego dekrementacją. [58, 66]

Do przykładów wykorzystania stosów można zaliczyć: algorytmy RegEx, algorytmy propagacji wstecznej, a także jako narzędzie do rozwiązywania zadania wież Hanoi [66].

3.4. Kolejki

Kolejki (ang. *queue*) należą do kolejnego rodzaju modyfikacji list, które obsługują zapytania w kolejności FIFO (ang. *First In First Out*). Oznacza to, że nowe elementy dodawane są na końcu kontenera, jednak usuwanie następuje po drugiej stronie struktury – przeciwnie jak w przypadku stosu, gdzie zarówno dodawanie jak i odejmowanie było wykonywane na końcu listy. Do prostego zobrazowania omawianej struktury można sobie wyobrazić prostą kolejkę sklepową ze świata rzeczywistego – jej cyfrowy odpowiednik spełnia te same założenia. Aby struktura mogła prawidłowo działać i spełniać swoje właściwości, konieczne jest aby zapisywała informację o wskaźniku do pierwszego i ostatniego przechowywanego elementu. Podobnie jak w przypadku stosu, kolejkę można zaimplementować zarówno z pomocą tablicy, jak i listy łączonej. Aby złożoność operacji dodawania i odejmowania w notacji O była stała konieczne jest śledzenie informacji o wskaźniku lub indeksie pierwszego, jak i ostatniego elementu.

Kolejki znalazły zastosowanie w algorytmach przeszukiwania grafów, optymalizacji sieci, a także planowaniu rozłożenia przepływu danych w procesorach [57].

Działanie kolejek i stosów można łatwo zgeneralizować do zachowania struktury deque (ang. *Double Ended Queue*). Jej implementacja umożliwia przeprowadzanie operacji dodawania i usuwania elementów zarówno z początku, jak i końca listy. [58, 66]

3.5. Mapy

Mapy, inaczej słowniki są nieuporządkowanym zbiorem rekordów, z których każdy jest zdefiniowany za pomocą pary klucz-wartość. Wewnątrz pojedynczej instancji struktury, wszystkie klucze muszą być unikatowe. Przetrzymywane wartości nie muszą przestrzegać tego warunku. Jak wskazuje sama nazwa, struktura ta przyporządkowuje (mapuje) wprowadzane wartości do unikatowego klucza. Mapy jako bufor danych wykorzystują tablicę, których pojedynczy element składa się z opisanej wcześniej pary, z czego klucz odpowiada indeksowi do którego został przypisany taki węzeł [66]. Szczególnym przypadkiem słownika jest hash mapa, która wykorzystuje algorytmy hashujące do rzutowania prymitywnych typów danych na indeksy obsługiwane przez tablicę będącą fundamentem opisywanej struktury danych. Funkcje hashujące muszą być skonstruowane w taki sposób, aby podając na jej wejście konkretną wartość zawsze zwracały taką samą reprezentację numeryczną. Dzięki temu, posiadając klucz do poszukiwanej wartości, dostęp do niej staje się bardzo szybki, niezależnie od ilości przetrzymywanych przez strukturę danych. Głównym problemem takiego sposobu składowania danych jest ograniczenie rozmiarem tablicy służącej jako bufor dla mapy. Oznacza to, że w niektórych przypadkach różniącym się od siebie wartością mogą zostać przypisane takie same indeksy, pomimo iż ich hashe są inne. Jednym z rozwiązań wykorzystywanych w popularnych implementacjach hash map jest wykorzystanie listy połączonej do składowania wszystkich elementów o tym samym

indeksie [65]. Dla małych zbiorów danych taka sytuacja zachodzi na tyle rzadko, że nie wpływa znacznie na ogólną wydajność opisywanego kontenera danych.

Aby struktura mogła być nazywana mapą musi implementować trzy podstawowe operacje: dodawanie / aktualizowanie wartości, usuwanie, wyszukiwanie. Analogicznie jak w przypadku tablic, wszystkie mechanizmy opierają swoje działanie na obsłudze indeksowania i wyszukiwania elementu za jego pomocą. Aby znaleźć element przetrzymywany przez hash mapę, pożądany klucz musi być najpierw przeprocesowany przez funkcję haszującą, a następnie tak przygotowany indeks jest gotowy do wyszukania. Dzięki temu operacje odczytywania i dodawania są wykonywane w czasie stałym notacji O. W zależności od tego jak dużo elementów jest przetrzymywanych w strukturze czas potrzebny do znalezienia elementu może się wydłużyć do liniowego, w przypadku gdy konieczne jest przeszukanie całej listy przypiętej w danym indeksie.

Mapy są najczęściej wykorzystywane na średniej wielkości zbiorach danych, kiedy nie jest wymagany aspekt posortowania składowanych wartości. Dodatkowo, najlepiej jeśli znany jest rozmiar zbioru, tak aby możliwe było określenie rozmiaru tablicy buforowej, celem uniknięcia kolizji indeksów powstałych w skutek procesowania funkcją hashujące. [65, 66]

3.6. Drzewa

Wszystkie opisane wcześniej struktury danych, t.j.: tablice, listy, kolejki i stosy przechowują dane w sposób liniowy, a ich elementy są połączone ze sobą sekwencyjnie. Wraz z rosnącą ilością węzłów, rośnie również złożoność takiej struktury, kiedy konieczne jest przejście po wszystkich jej elementach. W aktualnym świecie generowane dane są na tyle rozbudowane, że taki sposób ich przetrzymywanie jest zwyczajnie niemożliwy, bądź przynajmniej nieoptymalny wydajnościowo. Na potrzeby rozwiązania tego problemu, powszechnie stosowanymi algorytmami są nieliniowe struktury danych.

Drzewa są idealnym przykładem będącym odpowiedzią na opisaną komplikację; jest nieliniową, hierarchiczną strukturą danych, składająca się ze zbioru jednostek nazywanych inaczej węzłami. Wszystkie węzły w drzewiastej strukturze są połączone ze sobą za pomocą krawędzi, które mogą być zarówno skierowane, jak i nieskierowane. Opis struktury drzewiastej wymaga wprowadzenia kilku pojęć, które pozwolą dokładnie zrozumieć działanie tak zbudowanego kontenera danych.

Węzeł jest najbardziej podstawową jednostką drzewa i oprócz przetrzymywania wartości, śledzi również wskaźniki do tzw. węzłów dzieci – prawego i lewego. Korzeń drzewa jest szczególnym rodzajem węzła, od którego zawsze rozpoczyna się mechanizmy przeszukiwania drzewa. Korzeń również posiada wskaźniki do swoich dzieci, jednak jest jedynym węzłem w całej strukturze, który nie posiada rodzica. Rodzicem, lub inaczej węzłem nadrzędnym, nazywana jest każda jednostka będąca

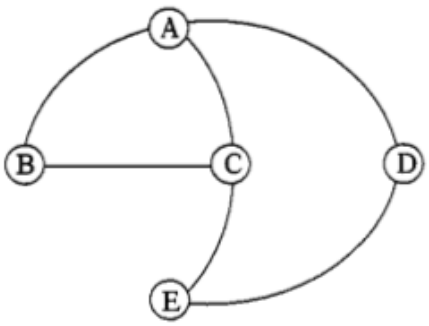
poprzednikiem dowolnego innego węzła. Wszelkie połączenia pomiędzy pojedynczymi jednostkami określane są mianem krawędzi. Ilość krawędzi standardowego drzewa binarnego można wyliczyć jako N-1, gdzie N – ilość węzłów w strukturze. Ostatnim pojęciem powszechnie wykorzystywanym do opisu struktury drzewiastej jest liść, czyli węzeł od którego nie odchodzą już żadne krawędzie – znajduje się na najniższym poziomie całej struktury.

Podstawowe operacje, które powinny być implementowane przez drzewa to: wyszukiwanie, dodawanie, usuwanie. Analogicznie jak w przypadku struktur liniowych, przechodzenie po elementach, z których składa się cały kontener, wiąże się z odwiedzaniem pojedynczych jednostek, a następnie przemieszczanie się w dół po krawędziach, aż do osiągnięcia wyszukiwanej wartości, lub ostatniego poziomu drzewa – liści. Najbardziej klasyczny mechanizm przeszukiwania struktury rozpoczyna swoje działanie od korzenia, a następnie z pomocą wywołań rekurencyjnych sprawdza lewe poddrzewo, aby na końcu przejść do prawego poddrzewa. Istnieje wiele wariacji tego mechanizmu optymalizujących proces przeszukiwania pod konkretne zastosowania.

Drzewiaste, hierarchiczne struktury znalazły zastosowanie w wielu aplikacjach, służąc jako bardzo dobre algorytmy sortujące, kontenery przechowujące znaczą ilość danych, lub mechanizmy sprawdzające składnię w kompilatorach języków programowania. Szeroki wachlarz zastosowań jest możliwy dzięki wysokiej plastyczność tych struktur i możliwości odpowiedniej modyfikacji pod kątem zadań, do których mają być przeznaczone. [58, 66]

3.7. Grafy

Grafy, podobnie jak drzewa należą do nieliniowych struktur danych składających się ze skończonej liczby węzłów i wierzchołków i łączących je krawędzi. Jeśli krawędzie są skierowane, graf jest czasem nazywany digrafem. Grafy mogą być używane do modelowania danych, gdy interesują nas połączenia i relacje między danymi. Elementy budujące tę strukturę danych, a także sposób ich łączenia nie odbiega znacznie od drzew, jednak w odróżnieniu do nich, sieć połączeń w grafie może przybierać charakter cykliczny. Klasyczną wizualizację można zaobserwować na rysunku 6.



Rys. 6. Przykładowy schemat przedstawiający graf [58].

Podobnie jak w przypadku drzew, istnieje wiele rodzajów grafów, jednak na potrzeby niniejszej pracy nie jest konieczne wprowadzanie i dokładny opis każdego z nich. Na uwagę zasługują pojęcia m.in. grafów skierowanych i ważonych. Pierwszy z nich oznacza sytuację, w której krawędzie jasno wskazują na kierunek, którego należy przestrzegać poruszając się po strukturze. Jeśli połączenia pomiędzy konkretnymi węzłami mają przypisaną jakąś wagę – koszt, który musi zostać poniesiony w trakcie przechodzenia przez tę krawędź, to taką architekturę nazywamy ważoną.

W technologii informacji grafy reprezentowane są dwojako: poprzez macierz sąsiedztwa lub listę sąsiedztwa. Macierz służy do pokazania, które węzły leżą obok siebie, tzn. czy istnieje jakieś połączenie pomiędzy konkretnymi węzłami. Dla takiej reprezentacji należy utworzyć macierz *N x N* M, gdzie *N* – ilość wierzchołków. Jeśli między węzłami A i B istnieje krawędź, to odpowiadający jej element macierzy M_{AB} będzie równy 1 (lub wartość wagi, dla grafu ważonego). W przypadku braku krawędzi M_{AB} wyniesie 0. Lista sąsiedztwa podchodzi do zadania reprezentowania grafu w zgoła inny sposób, ponieważ do każdego wierzchołka przypisana jest lista podłączonych do niego sąsiadów. Cała reprezentacja zbudowana jest na tablicy indeksowanej przez numery wierzchołków, w której każdy element x wskazuje na pojedynczo połączoną listę sąsiadów x.

Analogicznie jak w implementacji drzew, podstawowymi operacjami obsługiwanymi przez tę strukturę są: dodawanie, usuwanie i wyszukiwanie węzła. Jednak, w grafach niezwykle ważną rolę spełniają krawędzie, tak więc ich modyfikacje również muszą być możliwe. Z racji na potencjalność występowania wielu ścieżek pomiędzy węzłami, algorytmy przeszukiwania grafów są nieco bardziej skomplikowane i zaawansowane niż klasyczne przechodzenie przez listy, czy nawet drzewa. Najbardziej popularnymi mechanizmami służącymi do rozwiązywania takiego problemu są BFS (ang. *Breadth-First Search*) i DFS (ang. *Depth-First Search*). Pierwszy z nich wykonuje proces przechodzenia przez graf poprzez sprawdzanie sąsiadów kolejnych poszukiwanych węzłów. Zaczynając od korzenia grafu, mechanizm przeskakuje na każdego z sąsiadów, sprawdza czy spełniają one określone przez projektanta warunki, a następnie dla każdego prawidłowego węzła powtarza wymienione kroki. BFS wykorzystuje strukturę kolejkową do przetrzymywania węzłów grafu, które mają być sprawdzone jako następne w kolejności. Operacja przeszukiwania działa tak długo, aż w kolejce znajduje się przynajmniej jeden element, lub zostanie osiągnięty pożądany element. Dzięki takiemu sposobowi procesowania algorytm zawdzięcza swoją nazwę – przeszukiwanie wszerz. Zgoła innym podejściem charakteryzuje się operacja DFS, czyli przeszukiwanie w głąb. Algorytm rekurencyjnie przemieszcza się po krawędziach grafu aż do osiągnięcia elementu niespełniającego założonych warunków; wtedy rozpoczyna się proces propagacji wstecznej do ostatniego prawidłowego węzła i wybranie innej dostępnej ścieżki. Algorytmy BFS i DFS, wraz z ich modyfikacjami cieszą się bardzo dużą popularnością w zagadnieniach algorytmiki, kiedy wymagany jest wysoki poziom optymalizacji i efektywności wykorzystywanych narzędzi.

Grafy są wykorzystywane m.in. do rozwiązywania problemów świata rzeczywistego, w których informacje reprezentowane są jako sieć, jak sieci telefoniczne, sieci obwodowe i sieci społeczne. Przykładowa reprezentacja oznacza pojedynczego użytkownika jako węzły lub wierzchołki w sieci telefonicznej, podczas gdy ich połączenia telefoniczne reprezentują krawędzie. [58, 66]

3.8. ASA-grafy

Opisane w rozdziale 3.6. drzewa zostały przedstawione jako struktury, będące stosunkowo łatwe w modyfikacji, kiedy wymagane jest dostosowanie do obsługi warunków brzegowych konkretnego zadania. Za przykład można przytoczyć ASA-grafy (AG), wykorzystywane jako kluczowy element w opisywanej w pracy strukturze - DASNG. Głównym problemem niniejszego badania jest znaczna ilość danych, do których dostęp musi być swobodny, efektywny i bez znacznych opóźnień. Struktura DASNG ma za zadanie symulowanie działania neuronów ludzkich, gdzie każdy zdarzenie, impuls musi odbywać się w dokładnie wyznaczonym czasie, stąd wszelkie spowolnienia wynikające z wydawać by się mogło, trywialnego zadania pobrania danych z przechowującej jej struktury, mogą mieć wpływ na ostateczny wynik. Rozwiązaniem tego problemu jest wykorzystanie struktur ASA-graf, czyli agregująco-sortujących grafów asocjacyjnych. Przechowują one dane w posortowanej kolejności, pozwalając na bardzo szybki, zoptymalizowany dostęp do przechowywanych danych, wyszukiwanie, wstawianie, usuwanie i aktualizację. Dzięki zdolnościom agregującym, redukowana jest wymiarowość struktury, poprzez reprezentację duplikatów w formie pojedynczego węzła. Ponadto, wiadomość o ilości duplikatów może służyć do wyszukiwania najczęściej powtarzających się zależności. Połączenie tych dwóch głównych cech ASA-grafów ułatwia również zbieranie wartości statystycznych z reprezentowanych danych takich jak: sumy, średnie i mediany. Kolejną istotną cechą implementowaną przez AG jest łączenie kolejnych węzłów w rosnącej kolejności, już na etapie tworzenia całego kontenera. Opisywana struktura, swoją charakterystyką i sposobem działania bardzo przypomina samobalansujące drzewa poszukiwania binarnego w połączeniu z posortowaną listą łączoną. Dzięki połączeniu wszystkich przedstawionych operacji i cech, ASA-grafy oferują znaczną uniwersalność przy jednoczesnym zachowaniu wysokiej wydajności z maksymalnie logarytmicznym czasie przetwarzania w odniesieniu do unikatowych kluczy [59].

3.8.1. Budowa

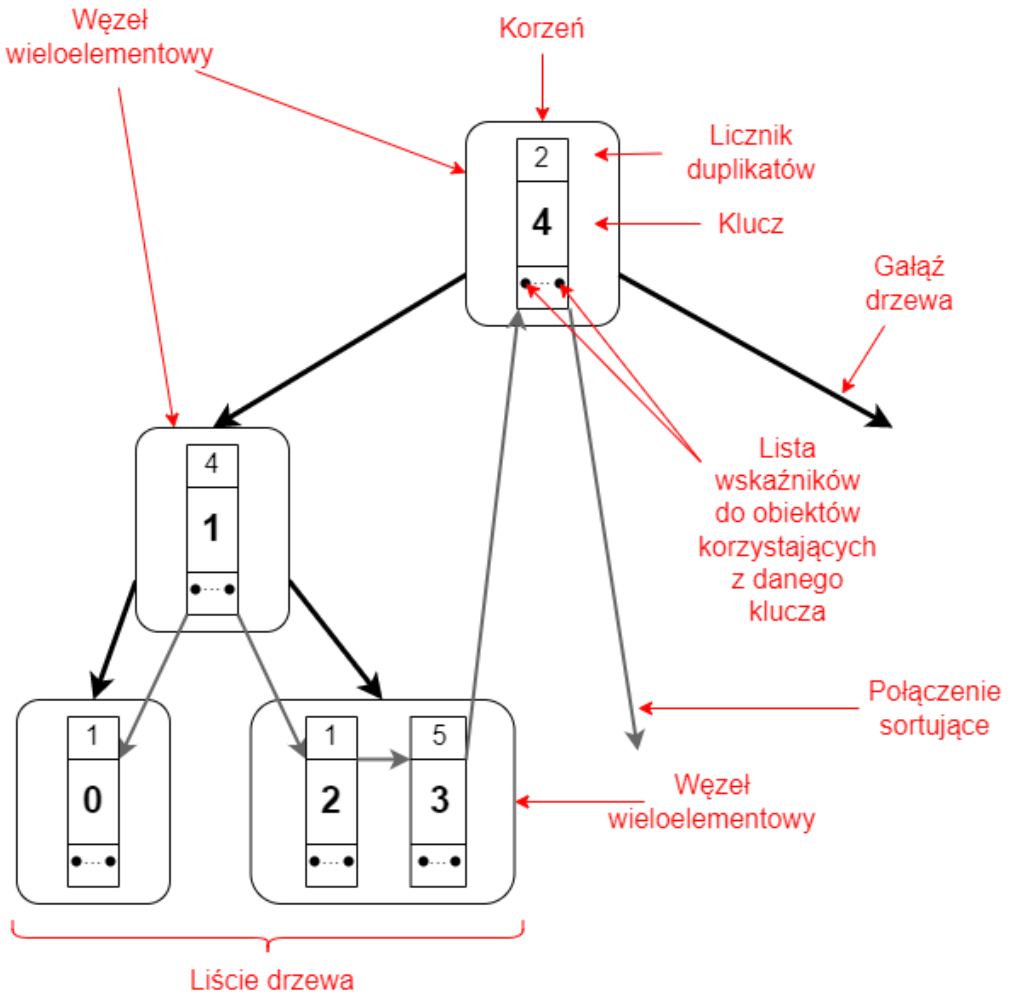
Aby zdefiniować pełną budowę ASA-grafu należy przedstawić sposoby wytwarzania lokalnych kompartmentów struktury. Węzły są najważniejszymi elementami bardziej zaawansowanych struktur danych, ponieważ to one przetrzymują informację o wszelkich zależnościach, t.j.: wskaźniki do sąsiadów lub zduplikowane wartości. W związku z tym

każdy węzeł AG musi posiadać: unikatowy klucz reprezentowanych danych, licznik zduplikowanych wartości dla danego klucza, dwa wskaźniki do poprzedniego i następnego elementu, listę wskaźników do obiektów wykorzystujących przechowywany klucz. W ten sposób utworzone elementy budują strukturę listy i drzewa przeszukiwania binarnego, a zachowanie wszystkich cech AG łączy się z koniecznością przestrzegania przez węzły wymienionych warunków. Rysunek 7 w dokładny sposób przedstawia schemat wizualizacji wszystkich elementów struktury ASA-graf.

Wspomniana lista, jest klasycznym przykładem podwójnie połączonej, posortowanej listy, w której każdy unikatowy element połączony jest za pomocą wskaźnika do referencji swojego poprzednika i następcy. Nieco bardziej skomplikowana jest struktura samobalansującego drzewa trzeciego stopnia, której wymogiem jest spełnianie następujących warunków:

- Każdy węzeł przechowuje jedną lub dwie porcje danych.
- Każdy węzeł niebędący liściem zapisuje listę wskaźników do dzieci, których liczba może być maksymalnie o 1 większa niż ilość przetrzymywanych kluczy.
- Wszystkie liście znajdują się tylko na ostatnim poziomie drzewa.
- Klucze przechowywane w lewym poddrzewie danego węzła muszą być mniejsze od kluczy znajdujących się w tym elemencie.
- Klucze przechowywane w prawym poddrzewie danego węzła muszą być większe od kluczy znajdujących się w tym elemencie.
- Klucze przechowywane w środkowym poddrzewie danego węzła – o ile takie istnieje – muszą być większe od najmniejszego i mniejsze od największego klucza znajdującego się w tym elemencie.

Mając na względzie powyższe definicje, możliwe jest określnie ASA-grafu jako hybrydowej jednostki złożonej z siatki specjalnie zaprogramowanych węzłów, umożliwiającej wykonywanie operacji charakterystycznych zarówno dla list łączonych, jak i drzew przeszukiwania binarnego.



Rys. 7. Schemat struktury danych ASA-graf [59, opracowanie własne].

3.8.2. Działanie

ASA-graf jest strukturą łączącą właściwości listy podwójnie łączonej z samobalansującym drzewem binarnym trzeciego stopnia. W związku z tym dostęp do każdego z węzłów może być zapewniony na dwa sposoby: poprzez przechodzenie przez elementy łączonej listy lub przesuwając się krawędziami drzewa, aż do osiągniecia żądanego elementu. Umiejętność agregacji danych i redukcja zduplikowanych elementów pozwala na nieznaczne przyspieszenie operacji przechodzenia względem klasycznych B-drzew, wykonując ją w czasie logarytmicznym od unikatowych kluczy zamiast wszystkich. Sposób, w jaki są zaprojektowane AG, umożliwia również reprezentację asocjacji pomiędzy węzłami. Dzięki temu, że każdy element posiada licznik powtórzeń i listę wystąpień danego węzła w innych obiektach, możliwe jest wyszukiwanie ważonych powiązań i relacji pomiędzy przedstawianymi przez strukturę obiektami. ASA-grafy zostały zaprojektowane specjalnie, żeby w zoptymalizowany sposób współdziałać z asocjacyjnymi strukturami neuronowymi.

3.8.3. Operacje

Analogicznie jak w przypadku klasycznych drzew i innych podstawowych struktur danych, ASA-grafy udostępniają metody wyszukiwania, dodawania i usuwania elementów. Operacje aktualizowania struktury implementują również mechanizm ponownego balansowania architektury, tak aby zawsze spełniała warunki opisane w 3.8.1.

Operacja przeszukiwania grafu wykorzystuje implementację algorytmu przeszukiwania w głąb DFS. Zaczynając od korzenia struktury, algorytm rekurencyjnie sprawdza kolejne dzieci węzła przy okazji sprawdzają czy pożądany element jest mniejszy, bądź większy od skrajnych kluczy przetrzymywanych przez węzeł. Wynik tej operacji logicznej określa jakie poddrzewa ma być następnie sprawdzane. Obecność wskaźników do referencji posortowanych elementów poprzedzających i sukcesorów, pozwala również na szybkie zwrócenie tych sąsiadów.

Mechanizm dodawania nowo utworzonego elementu w pierwszej kolejności korzysta z zdefiniowanej operacji przeszukiwania drzewa, kierując się porównywaniem wartości z obecnymi kluczami. Jeśli klucz dodawanego elementu już istnieje, należy zwiększyć licznik agregacji. W przeciwnym przypadku algorytm musi przestrzegać kilku kluczowych warunków. Kiedy iteracja drzewa dotrze do liścia możliwe jest dodanie nowego klucza w posortowanej kolejności. Jeśli liczba przetrzymywanych w węźle kluczy nie przekracza dwóch, to operacja kończy swoje działanie. W przeciwnym wypadku algorytm dzieli przepełniony węzeł i aktualizuje wszystkie połączenia aż do ponownego osiągnięcia równowagi struktury (wszystkie liście są na jednakowym, ostatnim poziomie). Podczas operacji dodawania nowego elementu, połączenia do poprzedniego i następnego węzła są odpowiednio aktualizowane, przez co lista posortowanych danych jest ciągle aktualna.

Najbardziej skomplikowaną metodą dostępną w obiekcie ASA-grafu jest operacja usuwania. Proces rozpoczyna się od wykorzystania mechanizmu przeszukiwania drzewa celem znalezienia pożądanej wartości. Jeśli licznik duplikatów jest większy niż 1, operacja kończy się na zdekrementowaniu go. W przypadku, gdy poszukiwany klucz reprezentował pojedynczą wartość, element zostaje usunięty, a wskaźniki jego sąsiadów zostają zaktualizowane, tak aby powstały połączenia pomiędzy nimi. Wraz z tym momentem rozpoczyna się proces ponownego przebudowywania drzewa do stanu równowagi. Przebudowa odbywa się według zmodyfikowanej wersji algorytmu zygzakowania, wykorzystywanego naprzemiennie z mechanizmem RB (ang. *red-black*) w innych strukturach samobalansujących. Jeśli usuwany klucz nie był jedynym elementem w węźle, a przetwarzany węzeł jednocześnie był liściem struktury, to operacja kończy swoje działanie. W przypadku, gdy cały liść zostaje pusty, algorytm może zdecydować o wypełnieniu liścia, lub usunięciu go. W pierwszym przypadku sprawdzane jest czy jeden z jego rodzeństwa posiada wiele elementów. Jeśli tak, jeden z kluczy rodzica zostaje zrzucony do pustego drzewa, a na zastępstwo pobierany jest element z wyszukanego rodzeństwa. W przeciwnym wypadku najbliższy element rodzica zostaje przeniesiony do węzła rodzeństwa. W sytuacji, kiedy zarówno rodzic jak

i rodzeństwo posiadają pojedynczy klucz dochodzi do zredukowania przetwarzanego poddrzewa jedynie do węzła rodzica, który wraz z tym momentem staje się liściem struktury. Zgodnie z zasadą równowagi poziomu, na którym znajdują się wszystkie liście struktury, konieczne jest wywołanie metody ponownego równoważenia drzewa. W ostatnim rozpatrywanym przypadku, usuwany element znajduje się w węźle niebędącym liściem. Algorytm uruchamia wtedy metodę rekurencyjnego przechodzenia po poddrzewach w dół, w trakcie której uzupełniany jest węzeł rodzica kluczem z jednego z węzłów podrzędnych. Mechanizm postępuje aż do natknięcia się na liść. Następnie, algorytm zaczyna postępować analogicznie jak w przypadku pozbycia się wartości z liścia, przechodząc do metod redukcji lub wypełnienia poddrzewa.

Dokładny opis wraz z pseudokodem implementacji poszczególnych operacji został przedstawiony w [59].

3.8.4. Złożoność obliczeniowa

Złożoność obliczeniowa wszystkich podstawowych operacji mieszczą się w czasie logarytmu od unikatowych kluczy wyekstrahowanych z kolekcji danych. Swoją szybkość zawdzięczają przede wszystkim redukcji wymiarowości wynikającej z ograniczenia liczby kluczy, tylko do unikatowych kluczy. W porównaniu do klasycznych B+drzew, AG przechodzą mniejszą lub taką samą liczbę kroków dla takiego samego zbioru danych. Dodatkowo, B+drzewa przechowują odniesienia do kluczy tylko w liściach, przez co operacja wyszukiwania będzie w większości przypadków wolniejsza niż w ASA-grafach z racji konieczności przechodzenia po całej wysokości struktury. Warty zauważenia jest również fakt, iż jeśli przechowywane dane charakteryzują się bardzo dużą liczbą duplikatów, ASA-grafy znacznie zyskują na wydajności względem klasycznych, nieagregujących struktur. Dokładne porównanie do najbardziej popularnych struktur danych zostało przedstawione w tabeli 1.

Tabela 1. Porównanie czasów wykonania operacji poszczególnych struktur danych. Ň reprezentuje liczbę unikatowych kluczy przechowywanych w strukturze, stąd $\check{N} \leq N$. [59]

Struktura Danych	Wyszukiwania	Dodawanie	Usuwanie	Min/Max	Mediana/ Średnia/ Suma
Hash mapa	O(1)	O(1)	O(1)	O(1)	O(N logN)
B-drzewo	O(log N)	O(log N)	O(log N)	O(log N)	O(N)
B+drzewo	O(log N)	O(log N)	O(log N)	O(log N)	O(N)
RB-drzewo	O(log N)	O(log N)	O(log N)	O(log N)	O(N)
AVL-drzewo	O(log N)	O(log N)	O(log N)	O(log N)	O(N)
WAVL-drzewo	O(log N)	O(log N)	O(log N)	O(log N)	O(N)
ASA-graf	O(log Ň)	O(log Ň)	O(log Ň)	O(log Ň)	O(Ň)

ASA-grafy przechowują więcej relacji pomiędzy danymi niż inne podstawowe struktury danych opisywane w powyższych rozdziałach, przy jednoczesnym zachowania wysokiego poziomu wydajności dla typowych operacji. W przypadku zadań o charakterze statystycznym, takich jak znajdowanie średnich, median, odchyleń standardowych, sum, minimów, maksimów, podobieństwa, sąsiedztwo itp., jest w stanie osiągać nawet lepszą szybkość obliczeniową.

ASA-grafy, dzięki swojej uniwersalności, poziomie wyspecjalizowania w przedstawianiu relacji, a także wysokiej wydajności zostały wybrane jako główna struktura przechowująca dane uczące sieci asocjacyjnej DASNG.

4. Charakterystyka problemu

Przetwarzanie, wydobywanie i eksploracja dużych danych to jedne najważniejszych zadań współczesnej informatyki, zwłaszcza po uwzględnieniu coraz bardziej popularnego sposobu chmurowego przetwarzania procesów komputerowych. Aby czerpać korzyści z zbiorów big data, konieczne jest stosowanie inteligentnych i wydajnych metod wyszukiwania danych i eksploracji wiedzy. Nie jest to łatwe zadanie, ponieważ dane są zazwyczaj przechowywane w relacyjnych bazach danych, które reprezentują relacje i związki pomiędzy danymi w sposób horyzontalny, nie pozwalając na przedstawianie niektórych asocjacji. Dane muszą być posortowane, poindeksowane lub połączone, a relacje pionowe muszą być często wyszukiwane i przetwarzane w wielu czasochłonnych, zagnieżdżonych pętlach.

W niniejszej pracy zaprezentowano implementację głębokich, asocjacyjnych, semantycznych grafów neuronowych pozwalających na przechowywanie danych, które są automatycznie poziomo i pionowo powiązane i uporządkowane według wszystkich atrybutów, bez ponoszenia znacznych kosztów obliczeniowych i pamięciowych.

4.1. Modelowanie danych

Modelowanie danych jest definiowane jako proces tworzenia wizualnej reprezentacji lub schematu, w celu przekazania połączeń pomiędzy punktami i strukturami danych. Taki plan lub model danych pomaga różnym podmiotom planowanego projektu, takim jak analitycy danych, naukowcy i inżynierowie, stworzyć jednolity widok danych organizacji. Celem jest zilustrowanie typów danych używanych i przechowywanych w systemie, relacji między tymi typami danych, sposobów grupowania i organizowania danych oraz ich formatów i atrybutów [67]. Modelowanie danych można rozpocząć na różnych poziomach abstrakcji, od początkowych badań i zbierania danych, do przekształcania baz danych i tworzenia modeli uczenia maszynowego. Pod generyczne pojęcie modelowania podchodzi wiele dziedzin inżynierii danych, jednak cały koncept jest ustandaryzowany odpowiednimi schematami i formalnymi technikami. Rozróżnia się trzy główne typy modeli, których obecność i proces powstawania jest zależny od reszty z nich [68].

4.1.1. Typy modeli danych

Proces modelowania danych zwykle rozpoczyna się od opracowania koncepcyjnego modelu danych. Na tym etapie należy wiedzieć, jakie założenia projektowe są ujmowane w wymaganiach dotyczących danych, co one oznaczają, gdzie znajdują się dane i jakie są niektóre relacje w tych danych. Taki rodzaj modelu najczęściej jest tworzony przez analityków i osoby odpowiedzialne za określenie wymagań projektowych.

Następnym krokiem jest utworzenie logicznego modelu danych, stanowiącego znaczną większość pracy związanej z modelowaniem danych. Obejmuje on wykorzystanie założeń, danych i informacji zwrotnych otrzymanych z procesu modelowania koncepcyjnego. Logiczne modele mapują koncepcyjne klasy danych na techniczne struktury. Dzięki temu możliwe jest określenie większej ilości szczegółów dotyczących pojęć danych i złożonych relacji, które zostały zidentyfikowane na etapie koncepcyjnym. Na tej podstawie tworzy się konkretne specyfikacje aplikacji.

Ostatnim etapem w złożonym procesie modelowania danych jest określenie wymagań dotyczących aspekty fizycznego przechowywania danych. Modele fizyczne zapewniają schemat, instrukcję, w jaki sposób dane będą fizycznie przechowywane w bazie danych. Z technicznego punktu widzenia są najmniej abstrakcyjne ze wszystkich opisanych. Oferują ostateczny projekt, który może być zaimplementowany jako relacyjna baza danych, z uwzględnieniem tablic asocjacyjnych, ilustrujących relacje między podmiotami, jak również klucze podstawowe i klucze obce, które będą używane do utrzymania tych relacji. [69]

4.1.2. Techniki modelowania danych

Przedstawiony w rozdziale 4.1.1. proces jest ogólnym schematem, według którego należy podążać wdrażając koncept modelowania danych do swojego projektu. Jednakże, techniki i podejścia, którymi finalnie będą się kierować projektanci przy tworzeniu baz danych mogą być zgoła różne i zależeć od tego, w jaki sposób informacje mają być przedstawiane. Poniżej przedstawiono najbardziej popularne wzorce modelowania danych:

- **Hierarchiczne**

  W modelowaniu hierarchicznym, relacje pomiędzy poszczególnymi elementami danych reprezentowane są w formie drzewa. Takie modele są w stanie przedstawiać relacje jeden do wielu, z rodzicami lub głównymi klasami danych mapującymi do kilku dzieci.

- **Grafowe**

  Naturalne rozszerzenie modeli hierarchicznych. Modele grafowe reprezentują relacje, traktujące encje danych jednakowo. Węzły z elementami danych mogą łączyć się ze sobą w relacjach jeden do wielu lub wiele do wielu bez podziału na rodzica i dzieci.

- **Relacyjne**

  Najbardziej popularny model przedstawiania danych, wykorzystywany w większości baz danych, w którym klasy danych przedstawiane są w formie tabel. Poszczególne tabele są ze sobą połączone aby móc przedstawiać skomplikowane zależności zachodzące pomiędzy danymi w rzeczywistym świecie. Jest to możliwe dzięki wprowadzeniu kluczy głównych i obcych, które

współpracują ze sobą ukazując relacje. Relacyjne modele danych mogą być wykorzystywane do reprezentowania m.in. danych strukturalnych.

- **Obiektowe**

  Wywodzące się z konceptu programowania obiektowego, modelowanie o tej samej nazwie, wykorzystuje struktury danych zwane obiektami do przechowywania porcji informacji. Każdy taki element jest abstrakcyjną reprezentacją rzeczywistej encji danych. Modelowanie obiektowe w znacznie łatwiejszy sposób jest w stanie przedstawiać relacje i zależności pomiędzy konkretnymi klasami danych. Jest to jeden z głównych powodów, dla których modelowanie obiektowe może rozwiązywać problemy klasycznych modeli relacyjnych.

- **Wielowymiarowe**

  Zaprojektowane głównie do optymalizacji prędkości pobierania i celów analitycznych w magazynach danych. Podczas gdy modele relacyjne kładą nacisk na wydajne przechowywanie, modele wymiarowe zwiększają redundancję, aby ułatwić lokalizację informacji na potrzeby raportowania i wyszukiwania.

Modelowanie danych jest niezwykle przydatnym narzędziem ułatwiającym programistom, architektom danych i analitykom biznesowym przeglądanie i zrozumienie relacji między danymi w bazach i magazynach danych. Wiele z wymienionych modeli jest na co dzień wykorzystywana w przemyśle i badaniach naukowych, jednak nie wszystkie techniki można określić mianem bezproblemowych. [67, 70]

4.2. Problemy relacyjnych baz danych

Opisany zwięźle w rozdziale 4.1.2. relacyjny model danych jest najczęściej wykorzystywaną techniką wytwarzania baz danych w świecie teorii informacji. Relacyjne bazy danych składają się z typów encji, które reprezentują klasę danych i określają różne poziome relacje, które mogą istnieć pomiędzy instancjami tych typów encji. W przełożeniu na rzeczywisty scenariusz, każdy wiersz tabeli reprezentuje jedną instancję danej encji, a każde pole reprezentuje typ atrybutu. Przedstawianie relacji pomiędzy encjami różnych typów jest możliwe dzięki zastosowaniu kluczy głównych i obcych. Pierwszy z wymienionych reprezentuje obiekt danego typu jako klucz obcy w encjach innych tabel.

Do najważniejszych korzyści relacyjnych baz danych należą: prostota modelu, łatwość użycia, prędkość, bezpieczeństwo przechowywanych danych. Pomimo wielu niezaprzeczalnych benefitów korzystania z takiego modelu reprezentacji danych, istnieją wady i punkty krytyczne, w których omawiane podejście jest niewystarczające i niewydajne. Mowa przede wszystkim o słabej skalowalności takich baz i znacznym

spadku wydajności wraz ze wzrostem ilości reprezentowanych danych. Wiąże się to przede wszystkim z tabelarycznym charakterem modelu relacyjnego i systemem reprezentowania relacji poprzez indeksowanie. Podobnie jak w strukturach słownikowych opisanych w rozdziale 3.5., przy stosunkowo niskiej ilości informacji dostęp do nich jest szybki i bezproblemowy, to niestety, rozbudowywanie struktury i jej złożoności ma bardzo duży wpływ na ogólną prędkość wykonywania najbardziej podstawowych operacji wyszukiwania, dodawania i usuwania. Ponadto, pomimo bardzo dobrej obsługi relacji poprzecznych - czyli takich pomiędzy różnymi typami encji - relacyjne bazy danych nie są przystosowane do opisywania relacji poziomych, wewnątrz tej samej tabeli. Brak optymalnej obsługi takich zależności wymusza na systemach zarządzania bazą danych do poszukiwania pionowych relacji z pomocą wielu pętli wywołań SQL (ang. *Structured Query Language*). Operacje wyszukiwania w relacyjnych bazach danych są zazwyczaj najczęściej wykonywanymi zapytaniami, więc nieefektywność ich wykonywania może być bardzo kosztowna przy zarządzaniu dużymi zbiorami danych. Finalnie, sortowanie danych po wielu atrybutach wymaga od bazy danych wprowadzania indeksowania, aby odpowiednie algorytmy sortujące (np. B+drzewa) mogły sobie poradzić z takimi zapytaniami. Indeksowanie spowalnia wykonywanie operacji dodawania i usuwania, z racji iż muszą one być aktualizowane przy każdym takim wywołaniu. Dodatkowo, jest to kolejna informacja, która musi być przetrzymywana w pamięci urządzenia.

Proponowany w niniejszej pracy algorytm, próbuje rozwiązywać opisane problemy, oferując sposób na reprezentację relacji zarówno poprzecznych, jak i poziomych. Dzięki takiemu podejściu, znacznie zwiększa się ilość asocjacji, które można odkryć pomiędzy pozornie nie połączonymi ze sobą danymi. Dodatkowo, symulowanie operowania rzeczywistych struktur biologicznych pozwala na wgląd w sposób, w jaki działa ludzki umysł, tworząc relacje i połączenia na podstawie bodźców z otoczenia.

4.3. DASNG – proponowane rozwiązanie

Ludzki mózg jako ciało składa się z gęstej i bardzo złożonej sieci komórek nerwowych różnego typu [16]. Taką sieć można zobrazować w formie grafu, bądź drzewa wykorzystujących tysiące swoich połączeń celem reprezentowania wiedzy i danych zbieranych przez cały okres życia jednostki. Dzięki temu, ludzka pamięć jest w stanie działać szybko, odpowiadać na złożone bodźce i znajdować odpowiedzi na skomplikowane problemy [60]. Powyższy sposób działania struktury mózgowej został zasymulowany w proponowanym w niniejszej pracy rozwiązaniu – Głębokich Asocjacyjnych Semantycznych Grafach Neuronowych (ang. *Deep Associative Semantic Neuronal Graphs, DASNG*).

Graf DASNG jest rodzajem sieci neuronowej, opartej na asocjacyjnym podejściu do modelowania danych. Oznacza to, że jego działanie bazuje na wyszukiwaniu, przetwarzaniu i opisywaniu związków pomiędzy obiektami reprezentującymi konkretne

cechy danych. Ponadto, DASNG można rozpatrywać w dziedzinach kognitywnego systemu neuronowego, w związku z czym jego działanie jest podobne do sieci semantycznych przedstawiających związki pomiędzy połączonymi ze sobą klasami cech [58]. W wielu praktycznych sytuacjach, takie rozwiązanie pozwala na bezpośrednie pobieranie powiązanych ze sobą informacji, zamiast wielokrotnego iterowania w pętlach zapytań SQL.

Grafy DASNG można opisać jako hybrydową strukturę wykorzystującą opisane w rozdziale 3.8. ASA-grafy do przetrzymywania danych konkretnych klas cech (co również pozwala na szybki dostęp do pionowych zależności) i zmodyfikowane neurony impulsowe przetrzymujące informację o pojedynczych instancjach encji wraz z wskaźnikami do innych neuronów. Asocjacyjne neurony impulsowe opisane w 2.4.1. zastępują klasyczne wierzchołki grafu, tworząc swego rodzaju obiektowy model danych. Aby adresować przedstawiony problem, opisywana sieć powinna symulować działanie biologicznej struktury neuronowej. W związku z tym, neurony, podobnie jak ich biologiczny odpowiednik, powinny być w stanie rejestrować zewnętrzne pobudzenia i na ich podstawie propagować ważony sygnał w głąb grafu. Neurony APN ze względu na zakodowaną możliwość obsługiwania mechanizmu pobudzania i reagowania na zewnętrzne bodźce, zostały wykorzystane w strukturze DASNG. Takie podejście znacznie wzbogaca klasyczne sposoby reprezentowania relacji przez modele danych i daje wolną przestrzeń do wykonywania badań i wyszukiwania zależności, które nie byłyby możliwe do osiągnięcia zwyczajnymi sposobami.

Przedstawianie relacji i powiązań pomiędzy danymi jest kluczowe w procesach modelowania danych. Takie połączenia mogą przedstawiać wiele zależności m.in.: stopnień podobieństwa, sąsiedztwa w przestrzeni, czy sekwencyjność pomiędzy następującymi po sobie cechami. Dzięki dokładnemu określaniu relacji możliwe jest grupowanie obiektów, przypisywanie ich do konkretnych klas cech, itp. Aby to było możliwe, algorytmy stosują różne warunki, ograniczenia, procedury wyszukiwania i operacje, które porównują lub grupują obiekty, celem spełnienia zdefiniowanych wymagań lub osiągnięcia zadanych celów [59]. Prezentowana architektura w sposób naturalny jest w stanie określać te parametry, posiadając jako punkt zaczepienia jedynie relacje opisane kluczami głównymi i obcymi z przekształcanej bazy danych.

Podobieństwo pomiędzy obiektami może być określane jako każdy podzbiór atrybutów blisko ze sobą korelujących wewnątrz tej samej klasy encji. Są one w naturalny sposób określane dzięki mechanizmowi pobudzania. Sygnał pobudzenia będzie stopniowo rozszerzał się na kolejne sensory, których relacje są blisko ze sobą skorelowane.

Klasy obiektów mogą być definiowane z pomocą parametru sąsiedztwa, a także sposobu pogrupowania danych powstałego w skutek przekształcania pierwotnej bazy. W strukturze DASNG, wszystkie takie same wartości są zagregowane, podczas gdy podobne cechy są do nich podłączone; zarówno w sposób bezpośredni, jak i pośredni. Pośrednie połączenia przebiega wtedy przez inne neurony obiektowe korzystające z

danej wartości. Można się spodziewać, ze wraz z dostarczaniem sygnału wejściowego, klasy obiektów najbliższych do tego pobudzanego, będą pulsować najszybciej.

Ostatnim parametrem, który można wyznaczać z wykorzystaniem architektury DASNG jest sąsiedztwo w czasie i przestrzeni. Nie jest to jednak parametr, który można utożsamiać z podobieństwem i klasami obiektów. O sąsiedztwie można wspominać, kiedy dwa porównywane ze sobą obiekty będą pulsować w podobnym punkcie w czasie, bądź kiedy ich współrzędne w wielowymiarowej przestrzeni są do siebie zbliżone. Jest to parametr niezależny od podobieństwa i różnic pomiędzy dwoma cechami, więc wprowadza dodatkową informację o wspólnej relacji dwóch pozornie rożnych cech danych. Dzięki temu algorytm jest w stanie wyszukiwać związki pojawiające się dopiero po dłuższej stymulacji, wśród obiektów z różnych klas, niebędących połączonymi w widoczny sposób. Ta cecha jest bardzo ważna z punktu widzenia przechowywania złożonych sekwencji, które mogą być stosowane tylko w pewnych specyficznych sytuacjach i okolicznościach, w jakich nasz mózg zmusza nas do podjęcia określonej decyzji.

Wszystkie opisane parametry wzbogacają informacje zawarte w bazie danych w szereg relacji, których nie da się w dogodny sposób opisać w klasycznym modelu relacyjnym. Podobieństwo w działaniu do struktury rzeczywistego mózgu rzuca nowe światło na sposoby przetwarzania danych w zagadnieniach inteligencji obliczeniowej. Analogicznie jak w swoim biologicznym odpowiedniku [61], model DASNG większość swojego działania opiera na jednostkach nazywanych sensorami odpowiedzialnymi za zbieranie zewnętrznych bodźców, a także sieć połączonych między sobą neuronów, która w bezpośredni sposób przechowuje zakodowane w niej porcje informacji.

4.2.1. Sensory i neurony obiektowe

W trakcie przekształcania bazy danych do struktury grafu neuronowego, algorytm przepisuje relacje pomiędzy danymi, a następnie wykorzystuje je do tworzenia wag i oceny podobieństwa pomiędzy cechami wejściowymi. Taki mechanizm można przyrównać do procesu uczenia klasycznej sieci neuronowej i tak też powinien być traktowany. Sieć DASNG ucząc się korzysta z dwóch fundamentalnych rodzajów jednostek, które zostaną opisane poniżej: sensory i asocjacyjne neurony impulsowe. Taka konstrukcja architektury pozwala na szybkie procesowanie z wykorzystaniem różnych kombinacji impulsów i pobudzania elementów sieci.

Sensory są najważniejszymi elementami sieci pod kątem odbierania sygnałów wejściowych. Ich rolę można przyrównać do komórek nerwowych, których zadaniem jest rejestrowanie bodźców z otoczenia. Każdy atrybut A_k przekształcanej bazy danych rzutowany jest na tzw. pole sensorowe F^{A_k}, za których reprezentację odpowiadają opisane w rozdziale 3.8. ASA-grafy. Każdy węzeł takiego pola, po nieznacznej modyfikacji, będzie dalej nazywany sensorem S^{A_k}. Każdy sensor $S_i^{A_k}$ posiada wszystkie parametry węzła AG tzn. przechowuje informację o ilości zduplikowanych wartości $v_i^{A_k}$

reprezentowanych przez dany klucz, a także posiada wskaźniki do referencji posortowanych kluczy: poprzedzającego i następującego po danym obiekcie. Modyfikacja sensora względem węzła struktury sortującej wynika z konieczności wprowadzenia kilku dodatkowych informacji opisanych poniżej. Po pierwsze, sensor musi posiadać informację o sile, z jaką będzie pobudzany w chwili wystąpienia impulsu wejściowego. Pobudzenie $x_{v_i}^{A_k}$ wyliczane jest następującym wzorem:

$$x_{v_i}^{A_k} = \begin{cases} 1 - \dfrac{\left|v_i^{A_k} - v^{A_k}\right|}{r^{A_k}}, & r^{A_k} > 0 \\ \dfrac{\left|v_i^{A_k}\right|}{\left|v_i^{A_k}\right| + \left|v_i^{A_k} - v^{A_k}\right|}, & r^{A_k} = 0 \end{cases} \tag{3}$$

gdzie v^{A_k} – średnia wartości sensorów w drzewie danego atrybutu, r^{A_k} – różnica między ekstremami w drzewie danego atrybutu. Dzięki własciwości automatycznego równoważenia ASA-grafu, ekstrema są na bieżąco aktualizowane jeśli do struktury jest dodawany, lub usuwany dowolny element. W przypadku obsługi sensora typu *string,* pobudzenie zawsze następuje z watością 1.0. Jeśli przetwarzane atrybuty posiadają wartości liczbowe, możliwe jest rownież wyliczenie wagi $w_{i,j}^{A_k}$ podobieństwa pomiędzy dwoma kolejnymi elementami $v_i^{A_k}$ i $v_j^{A_k}$, według wzoru (4).

$$w_{i,j}^{A_k} = w_{j,i}^{A_k} = 1 - \frac{\left|v_i^{A_k} - v^{A_k}\right|}{r^{A_k}} \tag{4}$$

Owe wagi mają kluczowe znaczenie na późniejszych etapach działania algorytmu, kiedy sygnał przepływający przez sieć musi również zostać rozpropagowany wszerz pola sensorowego.

Pobudzenie sensorów i przesłanie impulsu następuje, jeśli przez odpowiednio długi czas, konkretne pole sensorowe jest stymulowane stałym sygnałem. Okres $t_{v_i}^{A_k}$, przez jaki niniejszy stan musi się utrzymywać, uznawany jest jako kolejny parametr przechowywany w obiekcie sensora. Zgodnie ze wzorem (5), przy założeniu ciągłego pobudzania, po upłynięciu chwili $t_{v_i}^{A_k}$, sensor jest gotowy do przekazania sygnału do kolejnego węzła sieci.

$$t_{v_i}^{A_k} = \begin{cases} \dfrac{r^{A_k}}{\left(r^{A_k} - \left|v_i^{A_k} - v^{A_k}\right|\right)}, & r^{A_k} > \left|v_i^{A_k} - v^{A_k}\right| \\ 1 + \left|\dfrac{v_i^{A_k} - v^{A_k}}{v_i^{A_k}}\right|, & r^{A_k} = 0 \\ \infty, & r^{A_k} = \left|v_i^{A_k} - v^{A_k}\right| \end{cases} \tag{5}$$

W kolejnym kroku przekształcania bazy danych do struktury DASNG, algorytm tworzy neurony obiektowe O^{T_n}, które reprezentują kolejne rekordy tabel T. W związku z tym, element $O_i^{T_n}$ przetrzymuje informacje o wszystkich wartościach danego rekordu. Z racji, iż każda kolejna wartość posiada swojego reprezentanta wśród pól sensorowych, neuron obiektowy musi zapisywać informację o wskaźnikach do odpowiadających sensorów, tak aby możliwe było przekazywanie impulsu po pobudzeniu węzła pola sensorowego. Analogicznie, jak na poziomie pól F, sygnał musi być przesyłany z odpowiednią siłą, wynikającą ze wzoru (6). W tym przypadku, waga sygnału jest dostrajana pod kątem ilości połączeń, jakie tworzą węzły; im mniej obiektów jest podłączonych do danego węzła, tym sygnał będzie mocniejszy. System jest opracowany w sposób, aby jak najdokładniej odwzorowywać rzeczywisty układ nerwowy. Można sobie wyobrazić sytuację, w której posiadając pewne unikatowe wspomnienia, niektóre bodźce aktywują je natychmiastowo, natomiast bardziej regularne pobudzenia, jak np. czynność picia kawy, nie doprowadzą do szybkiego przypomnienia sobie żadnej szczegółowej sytuacji. Należy również wspomnieć, iż wyliczanie wag nieco różni się dla tabel posiadających jedynie klucze główne, a tych, w których obecne również są relacje reprezentowane kluczami obcymi. W pierwszym przypadku, waga jest wyliczana jedynie jako odwrotność ilości neuronów podpiętych do danego sensora, natomiast w drugiej sytuacji, liczyć się również będą wskaźniki do innych neuronów. W implementacji algorytmu nie ma potrzeby wprowadzać takiego rozgraniczenia, ponieważ wszystkie relacje przetrzymywane są w jednej strukturze, której rozmiar l może posłużyć do określania wag.

$$w_{i,T_n}^{A_k} = \frac{1}{l} \tag{6}$$

Wraz z dodaniem lub usunięciem jakiegoś elementu, wagi, na które mogło to mieć wpływ, są automatycznie aktualizowane.

Biorąc pod uwagę konstrukcję wzoru (6), można zauważyć, iż waga – siła, z jaką będzie pobudzany obiekt, nigdy nie przekroczy wartości 1. Co za tym idzie, maksymalny próg θ jaki element może osiągnąć, aby doszło do pobudzenia, również przyjmuje taką wartość. Jest to dosyć intuicyjne, ponieważ jeśli ma zostać wywołana konkretna, unikatowa wartość, to po podaniu jej na wejściu struktury, powinna zostać zwracana natychmiast. Proces ten jest analogiczny do wywołania metody SEARCH w zapytaniu SQL.

Zarówno neurony obiektowe, jak i sensory w swojej implementacji bazują na strukturze asocjacyjnych neuronów impulsowych (rozdział 2.4.1.), których mechanizmy idealnie nadają się do obsługi operacji przetwarzanych przez sieć DASNG. Architektura obsługuje koncept czasu i zdarzeń odpowiedzialnych za zmiany stanów, w których neuron znajduje się w danej chwili. Parametr opisany wzorem (5) jest przypisywany jednemu z czterech obsługiwanych stanów – ładowaniu. Oprócz tego, dowolny węzeł sieci może również znajdować się w fazach spoczynku, relaksacji i refrakcji

bezwzględnej i względnej. Pierwszy z nich to tzw. stan czuwania, w którym element jedynie oczekuje na pobudzenie. Wraz z pojawieniem się sygnału, rozpoczyna się ładowanie, którego trwanie jest zdefiniowane z pomocą (5). Jeśli w międzyczasie dojdzie do ponownego pobudzenia, dochodzi do nałożenia się sygnałów, a co za tym idzie czas pozostały do pełnego naładowania skraca się dwukrotnie. Wraz z przekroczeniem progu θ, sensor lub neuron wydzielają impuls przekazując sygnał dalej, a same przechodzą w stan refrakcji bezwzględnej *RF*. W biologicznych strukturach, typowy czas trwania tego stanu waha się od 1 do 2 ms [62], jednak na potrzeby badań DASNG, został zdefiniowany jako odwrotność wielokrotności m_{RF} (tabela 3) wartości pobudzenia *s* w chwili przejścia w ten stan (7). Warty dodania jest fakt, iż w trakcie refrakcji neuron nie może pobierać żadnych nowych sygnałów, a po jej minięciu węzłowi zostaje przypisany stan spoczynku. Z drugiej strony, jeśli po zakończeniu procesu ładowania siła pobudzenia nie będzie wystarczająca do przekroczenia progu θ, neuron znajdzie się w stanie relaksacji *RL*, w którym będzie trwał aż wyzerowania jego wartości pobudzenia. Podobnie jak w przypadku refrakcji, okres ten został zdefiniowany na potrzeby badań jako wielokrotność m_{RL} poziomu pobudzenia *s* w danej chwili (8). Wielkość tego parametru była również sprawdzana w trakcie testowania algorytmu; w rzeczywistych strukturach biologicznych długość relaksacji zależy od rodzaju komórki nerwowej, w której zachodzi proces pobudzania i może dochodzić aż do 50 ms [78]. W implementowanym algorytmie, parametr relaksacji ma znaczny wpływ na to, jak daleko symulowany sygnał może się rozprzestrzeniać. Tabela 2 pokazuje związek pomiędzy wydłużeniem czasu, z jakim siła sygnału maleje, a ilością unikatowych neuronów i sensorów, które zostały pobudzone w trakcie procesowania. Można zaobserwować, że optymalna wartość współczynnika wielokrotności *m*, to 50. Po przekroczeniu jej, liczba odkrywanych neuronów i sensorów zaczyna maleć. Może to być spowodowane za częstym wchodzeniem wewnętrznej kolejki w stan refrakcji, przez co sygnał jest blokowany i nie ma możliwości dalej się rozprzestrzeniać. Każdy z testów był wykonywany przy zdefiniowaniu takich samych warunków początkowych: czas symulacji – 15s, częstotliwość pobudzania – 50/s, liczba stymulowanych jednocześnie receptorów – 3. Podobnej analizie została poddana wartość współczynnika m_{RF}, przypisanego refrakcji. Tabela 3 pokazuje wyniki badania tego parametru, w warunkach identycznych, jak te zdefiniowane dla testów m_{RL}. Ponownie, trend wskazuje na wzrost ilości unikatowych pobudzeni, aż do osiągnięcia pewnej wartości granicznej, po czym zaczyna spadać. Obie cechy powinny być traktowane jako hiperparametry sieci, które należy poddawać procesowi dostrajania, tak aby dostosować je do przetwarzanego zbioru danych.

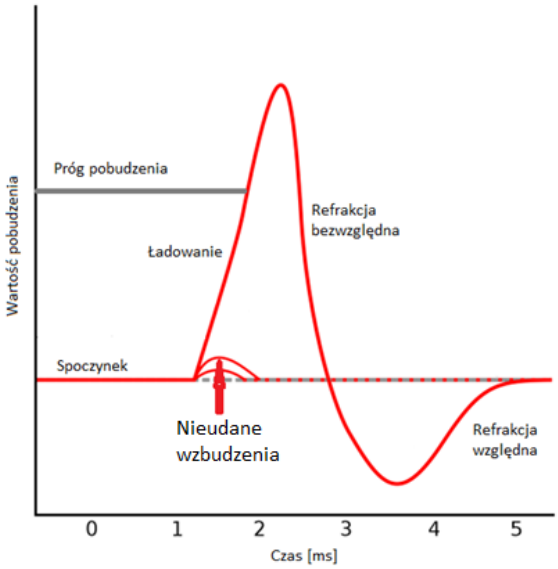
Tabela 2. Związek pomiędzy wartością współczynnika wielokrotności m_{RF}, a ilością unikatowych pobudzeń obiektów reprezentujących sensory i neurony w sieci DASNG.

Współczynnik m_{RF}	L. unikatowych, pobudzonych obiektów	L. unikatowych, pobudzonych sensorów
5	7	14
10	11	21
25	20	37
50	40	92
100	24	59
200	21	44

Tabela 3. Związek pomiędzy wartością współczynnika wielokrotności m_{RL}, a ilością unikatowych pobudzeń obiektów reprezentujących sensory i neurony w sieci DASNG.

Współczynnik m_{RL}	L. unikatowych, pobudzonych obiektów	L. unikatowych, pobudzonych sensorów
0,5	16	33
1	24	51
1,5	31	78
2	56	89
3	36	90
5	22	49

Obsługa wymienionych stanów jest możliwa dzięki mechanizmowi wewnętrznej kolejki zdarzeń IPQ, która została zaimplementowana wewnątrz każdego, pojedynczego obiektu sieci DASNG. Mechanizm ten odpowiada za wyliczanie czasów trwania konkretnych stanów i zmienia je, w trakcie dowolnego wywołania metody pobudzenia danego neuronu. Wszystkie zmiany czasów potrzebnych do zakończenia danego stanu, jak również wartości aktualnego poziomu pobudzenia, są modelowane liniowo, dzięki czemu znacznie ułatwione jest ich dodawania, odejmowanie, a także nakładanie się procesów ładowania i relaksacji.



Rys. 8. Schemat procesu pobudzania występującego w każdym neuronie.

$$t_{RF_{v_i}}^{A_k} = \frac{1}{m_{RF} * s} \tag{7}$$

$$t_{RL_{v_i}}^{A_k} = m_{RL} * s \tag{8}$$

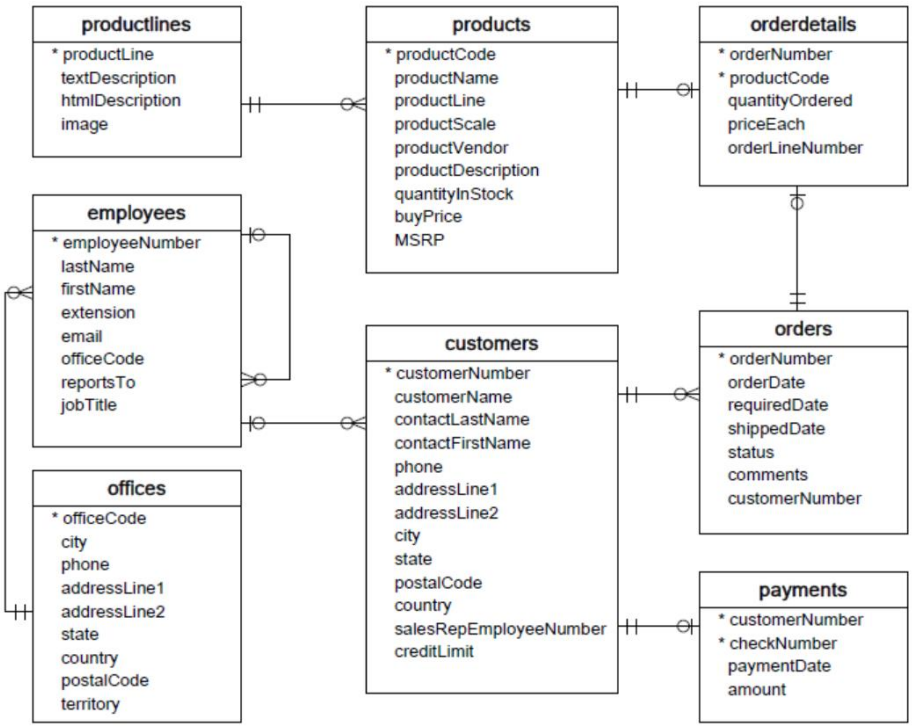
Mechanizm wewnętrznej kolejki procesów świetnie sprawdza się lokalnie, celem określania stanów danego neuronu, a także w jakim punkcie czasowym mają się zmienić. Jednak, pomimo sieci połączeń pomiędzy elementami struktury, informacja o stanach nie jest przekazywana wraz z sygnałem. Koordynacją zdarzeń pochodzących z lokalnych zmian stanów zajmuje się globalna kolejka zdarzeń (ang. *Global Event Queue, GEQ*). W przedstawionej implementacji rolę kolejki pełni prosta struktura słownika, z czasami wykonywania kolejnych zmian stanów jako kluczami i obiektami, których dotyczą. Struktura GEQ jest niezależna od reszty sieci DASNG i cały czas aktualizuje się działając w tle, na osobnym wątku. Kiedy przychodzi pora na wykonanie konkretnego zdarzenia, obiekt zostaje wyciągnięty z kolejki i wywoływana jest metoda aktualizacji statusu. W tym momencie, ponownie uruchamia się działanie kolejki wewnętrznej neuronu. Taki sposób komunikacji jest możliwy dzięki zdolności IPQ do wyliczania czasów zmiany stanów zawczasu, z wykorzystaniem wzorów podanych wyżej (6, 7, 8). Oprócz tego, GEQ posiada zdolność do aktualizowania czasów zdarzeń, wynikających z przekształceń dokonanych przez neuron lokalnie. Rozwiązanie takie jest wymagane w przypadku np. ponownego doładowania neuronu, które skutkuje skróceniem czasu potrzebnego do osiągnięcia progu θ i przekazania impulsu.

Wyniki zwracane przez sieć DASNG nieco różnią się od danych, które można dostać na wyjściu klasycznych sieci neuronowych II generacji. W omawianej strukturze, odpowiedzi są produkowane na podstawie częstotliwości występowania impulsów oraz

czasu, jaki upłynął od momentu stymulacji zewnętrznej do momentu, w którym neurony te zaczynają pulsowanie. W związku z tym, neurony impulsowe, które osiągają poziom pełnego pobudzenia, reprezentują wynik przetwarzania sygnału przez sieć. Jest to zdecydowanie przeciwne podejście, do tego stosowanego np. w głębokich sieciach neuronowych, gdzie wynikiem są konkretne wartości otrzymywane z ostatniej warstwy architektury.

4.2.2. Proces uczenia sieci DASNG

Na potrzeby tego rozdziału wykorzystano bazę danych zaprezentowaną na rysunku 9. Baza danych *classicmodels* [71] jest prostą architekturą wykorzystywaną do nauki języka SQL i testowania skryptów obsługujących taki rodzaj modeli danych. Model zestawia typowe dane z zakresu biznesu, czyli informacje o klientach, produktach, zamówieniach i zależnościach przedstawiających ścieżkę procesowania zamówienia. Dzięki temu, jak i przez swoją nieskomplikowaną budowę, w wygodny i prosty sposób możliwe było przedstawienie opisywanego zagadnienia.



Rys. 9. Schemat struktury bazy danych *classicmodels* [71].

Proces przekształcana bazy danych do struktury grafu asocjacyjnego rozpoczyna się od wyodrębnienia tabel zawierających jedynie klucze główne. Na zaprezentowanym przykładzie są to tabele *offices* i *productlines*. W pierwszym etapie, kolumny z wybranych tabel są przekształcane do postaci ASA-grafów reprezentowanych w formie pól sensorowych. Wszystkie wartości przetworzonego atrybutu są przechowywane w

węzłach grafu (dalej nazywanych receptorami). Następnie, kiedy algorytm zbuduje odpowiednie pole, rozpoczyna proces wypełniania go nowopowstałymi elementami – sensorami. Dla każdego receptora zostaje utworzony odpowiadający mu sensor, po czym oba elementy zostają połączone wskaźnikami, aby ułatwić proces komunikacji wymaganej do obsługi mechanizmu pobudzania. Jeśli przekształcany atrybut jest zbudowany z wartości liczbowych, pomiędzy kolejnymi sensorami są tworzone wagi odpowiadające parametrowi podobieństwa (4). Na rysunku 10, szare linie odpowiadają takiemu połączeniu. Na przykładzie tabeli *offices* zostanie utworzonych 9 różnych pól sensorowych, w których receptory i sensory będą odpowiadać kolejnym wartościom kolejnych encji. Kiedy cała tabela zostanie przetransformowana, wszystkie jej rekordy zostają przekształcone do postaci neuronów obiektowych, tzn. elementów przechowujących wskaźniki do wszystkich sensorów reprezentujących ich cechy. Takie połączenie musi być dwustronne, ze względu na charakter operacji pobudzania, dzięki której sygnał może być wysyłany zarówno od sensora do obiektu, jak i na odwrót. Zostały one przedstawione jako czarne strzałki na schemacie z rysunku 10. Obsługa tego mechanizmu wymaga od węzłów posiadania zdolności do przechodzenia pomiędzy kolejnymi stanami pobudzenia neuronowego. Takie właściwości oferują neurony asocjacyjne (rozdział 2.4.1.), tak więc zarówno sensory, jak i neurony obiektowe rozszerzają ich implementację wykorzystując najważniejsze metody i atrybuty.

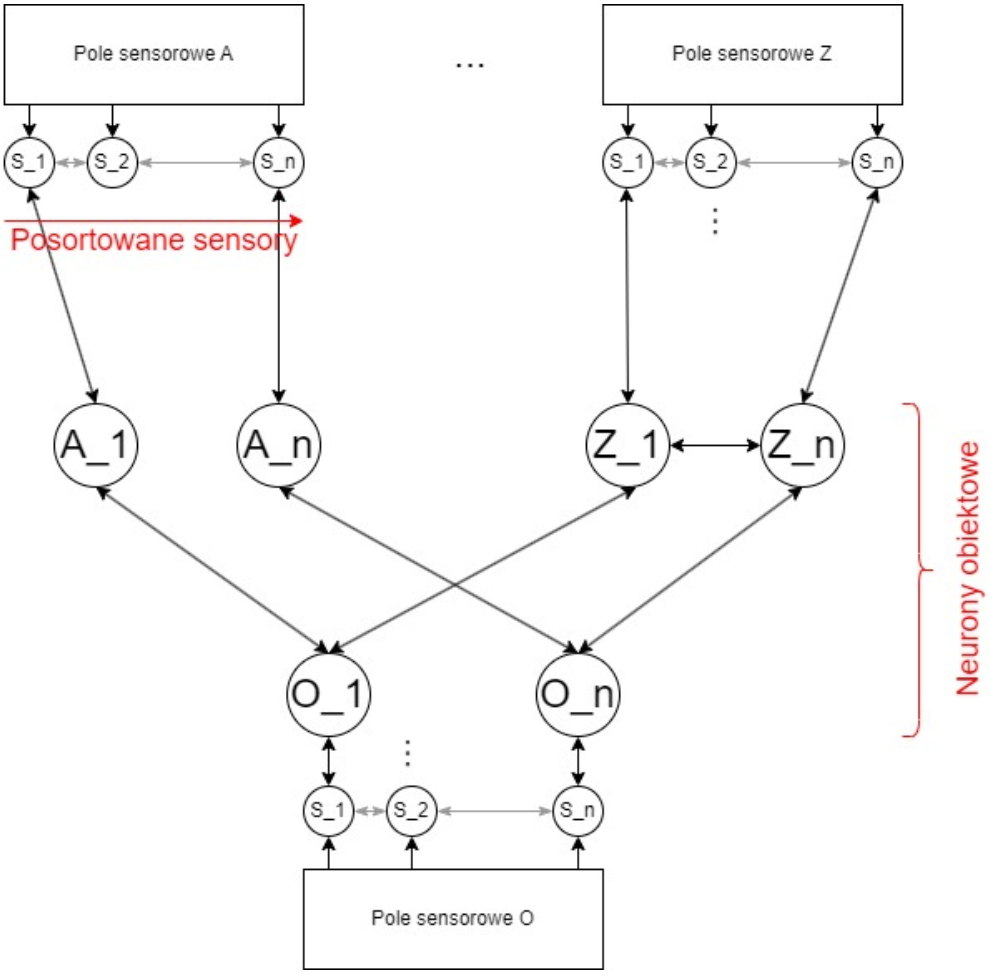
W następnym kroku przekształcane są tabele przechowujące atrybuty reprezentujące zarówno klucze główne, jak i klucze obce. Ponownie odnosząc się do przykładowej bazy danych z rysunku 9, będą to wszystkie pozostałe po wykluczeniu *offices* i *productlines* przekształconych w poprzednim etapie. Warte wspomnienia jest, iż ASA-grafy przechowujące dane konkretnego atrybutu są unikatowe, tzn. jeśli dwie różne tabele posiadają kolumny o takiej samej nazwie, to ich wartości będą przechowywane przez jedno pole sensorowe, zamiast dwóch różnych. Takie podejście znacznie upraszcza architekturę modelu DASNG i pozwala na szybszą inferencję, grupując podobne dane i sortując je. Wszystkie kroki przekształcania tabeli, tworzenia sensorów i finalnie neuronów obiektowych przebiegają niemal tak samo. Wyjątkiem jest jedynie powstawanie dodatkowych połączeń reprezentujących relację klucz główny – klucz obcy. Jest ono implementowane w formie dwustronnego wskaźnika pomiędzy węzłami reprezentującymi dwa oddzielne neurony obiektowe. Wszystkie koneksje są przetrzymywane wewnątrz struktury słownika zapisywanej jako atrybut każdego z obiektów.

Na samym końcu przepisywania bazy danych, pod uwagę brane są tzw. tabele łącznikowe, czyli takie, które nie przechowują żadnych konkretnych danych, a jedynie posiadają klucze obce przedstawiające relacje pomiędzy innymi atrybutami. DASNG redukuje tę informację jedynie do wskaźnika pomiędzy obiektami, dzięki czemu nie ma potrzeby rzeczywistego przekształcania encji takich tabel i zapisywania ich w sieci. Jest to kolejna cecha opisywanej architektury, która pozwala na zmniejszenie złożoności struktury i wykluczenie powstawania wielu pętli zapytań SQL celem wykonania niektórych zapytań. Pomimo tego, wszystkie kluczowe informacje są przechowywane w węzłach grafu asocjacyjnego, a dodatkowe cechy ASA-grafów, z których zbudowane są

pola sensorowe, pozwalają na wykonywanie operacji sortowania, wyszukiwania ekstremów niemal w czasie rzeczywistym.

Kolejność, w której przekształcane są tabele na każdym z poziomów, nie ma większego znaczenia, ponieważ ostatecznie struktura i tak osiągnie takie samo rozłożenie. Dodatkowo, dzięki zachowaniu wszystkich kluczowych informacji, możliwe jest ponowne odbudowanie oryginalnej bazy danych poprzez proces odwrotnego przekształcania.

Każda tabela bazy danych, która reprezentuje tylko jeden atrybut jest przekształcana w pojedyncze pole sensorowe, sensory i neurony wartości, podczas gdy tabele zawierające więcej atrybutów, wraz z kluczami obcymi, są reprezentowane jako oddzielne warstwy neuronów obiektowych. W związku z tym taki graf asocjacyjny może mieć wiele poziomów połączonych warstw w zależności od wielkości i złożoności przekształcanej bazy danych, przypominając struktury głębokie sieci neuronowych.



Rys. 10. Schemat struktury DASNG powstałej po operacji przekształcania bazy danych [opracowanie własne].

4.2.3. Pobudzanie sieci

Po przekształceniu tabel baz danych w sieć neuronową DASNG, powstała struktura może być wykorzystana do pobudzania reprezentowanych obiektów, calem szybkiego wyszukiwania podobnych obiektów, klas cech, identyfikacji wspólnych atrybutów, filtrowania i sortowania obiektów, a także wyciągania pożytecznych informacji na temat relacji i asocjacji występujących w zbiorze danych uczących.

Najprostszą interakcją, którą można wykonać z siecią DASNG jest przefiltrowanie danych pod kątem określonej wartości. Operacja jest równoznaczna z pobudzeniem sensora, bądź sensorów przedstawiających konkretne cechy. Po wydzieleniu impulsu przez sensor, struktura bezpośrednio zwróci odpowiadające zapytaniu obiekty. Warto zaznaczyć, że im więcej obiektów jest reprezentowanych przez pojedynczy sensor, tym dłuższe musi być pobudzenie, aby możliwe było zwrócenie neuronów obiektowych. Wiąże się to z faktem, iż bardziej unikatowym informacjom wejściowym są przypisane wyższe wagi połączeń. Kontynuowanie pobudzenia uruchomi rekurencyjną operację przekazywania sygnału na kolejne poziomy sieci, wskazując również dalsze połączenia i udostępniając informację o relacjach zwróconych węzłów. Mechanizm działa w podobny sposób do asocjacyjnych procesów pamięciowych zachodzących w mózgu, gdzie wywołanie pojedynczego wspomnienia potrafi uruchomić całą reakcję łańcuchową kolejnych momentów w naszej pamięci, przypominając je sobie po kolei [63]. Dalsze prace nad algorytmem mogą pozwolić na wykonywanie kolejnych procesów jednocześnie, na różnych wątkach procesora, co będzie oznaczało zwracanie odpowiedzi niemal w czasie stałym [59].

Procesy wnioskowania wykonywane przez sieć neuronową DASNG bazują na zliczaniu impulsów wydzielanych przez neurony, wraz z interwałami czasowymi, w jakich dochodziło do pobudzenia. Neurony, które są najbliżej połączone z sensorem reprezentującym dane wejściowe (czyli są bezpośrednią odpowiedzą na zadane zapytanie) zaczną pulsować najwcześniej i najczęściej. Rzadsze i bardziej opóźnione przekazywanie sygnału wskazuje na obiekty będące jedynie pośrednio, bądź częściowo powiązane z początkowymi wartościami pobudzającymi. Dzięki temu, chronologia aktywowania się pojedynczych elementów, automatycznie sortuje je względem stopnia powiązania i relacji z wprowadzonym zapytaniem. Wprowadza to dodatkowy poziom informacji, niedostępnych w klasycznych modelach reprezentacji danych.

Dane zwracane na wyjściu sieci opierają się na interwałach i częstotliwości pulsowania sensorów i neuronów. Na tej podstawie możliwe jest wyliczanie szeregu statystyk, które następnie można wykorzystywać do analizowania przetwarzanych informacji. Jedną z głównych funkcjonalności jest wyznaczanie klas obiektów, do których dochodzi sygnał od momentu rozpoczęcia symulacji. Tym sposobem pobudzając nawet jeden receptor, sieć będzie w stanie przyporządkować go do różnych klas, jednocześnie zwracając informację o powinowactwie sensora do danej etykiety. Wewnątrz każdej klasy, możliwe jest również zwrócenie neuronów, które były pobudzane i informacji o tym, kiedy i jak często miało to miejsce. Opisywana cecha może się bardzo dobrze sprawdzać w zadaniach klasyfikacyjnych na danych niepoznanych do tej pory przez sieć.

Należy jednak podkreślić, że na tę chwilę taka funkcjonalność działa jedynie na liczbowych danych wejściowych.

Kolejnym ciekawym zastosowaniem sieci jest określanie stopnia zależności pomiędzy różnymi danymi wejściowymi. Analiza wyników pozwala na wyznaczenia części wspólnej poddrzew pobudzeń pochodzących od pozornie różnych cech, którymi sieć jest stymulowana. Ponadto, informacje płynące z grafu, umożliwiają określenie kiedy po raz pierwszy doszło do napotkania się sygnałów i jak często wyznaczone części wspólne pulsowały. Takie podejście doskonale może się sprawdzać w zadaniach eksploracji danych, kiedy konieczne jest wyznaczenie zależności niewidocznych i niewykrywalnych z wykorzystaniem najprostszych zapytań SQL.

Oprócz klasycznych aspektów analizy danych, sieć DASNG świetnie sprawdza się również do przeprowadzania badań nad biologicznymi strukturami nerwowymi. Neurony wykorzystywane w architekturze operują na konceptach znanych z rzeczywistych komórek nerwowych, dzięki czemu symulują i przybliżają proces ich działania. Analiza wyników może w łatwy i przystępny sposób tłumaczyć związki zachodzące w mechanizmach kognitywnych ludzkiego mózgu. Wiele parametrów, na których opiera się przetwarzanie sygnału, jest modyfikowalna, tak aby dobrze móc się dostosować do danych uczących. Wprowadza to kolejne pole do manewru, dla badaczy chcących dowiedzieć się jak najwięcej o pobudzaniu nerwowym i regułach według których impulsy nerwowe są przetwarzane.

Ponadto, w skład architektury DASNG wchodzą ASA-grafy, które same w sobie oferują szereg możliwości manipulowania danymi. Każde pole sensorowe składające się z takiej struktury udostępnia metody pozwalające na zwracanie informacji o ekstremach w obrębie przypisanego im zakresu danych, automatyczne sortowanie danych, a także w przypadku danych liczbowych, podstawowe statystyki takie jak: średnia, mediana, itp.

Sieć DASNG udostępnia interfejs obsługujący wiele zagadnień z pogranicza analizy danych, uczenia maszynowego, a także badań nad układem nerwowym. Opisane metody pozwalają na wykonywanie zadań klasyfikacji, grupowania wartości podobnych, filtrowania i sortowania pod kątem wprowadzonej wartości, wyszukiwania zależności bezpośrednich i co ważniejsze – pośrednich. Dodatkowo, DASNG w przeciwieństwie do klasycznych algorytmów uczenia maszynowego działa w sposób transparentny i wyjaśnialny wpisując się w ogólną ideę wyjaśnialnej sztucznej inteligencji (ang. *explainable AI*). Najważniejszą cechą sieci jest jej zdolność do określania relacji pomiędzy kolejnymi obiektami, co daje świetny pogląd na to, w jaki sposób sygnał jest przez nią przetwarzany. Plastyczność parametrów grafu pozwala na dostosowanie go do różnorodnych sytuacji i zbiorów danych, oferując wiele możliwości analitycznych i badawczych.

5. Wyniki

Wszystkie podstawowe testy były wykonywane z wykorzystaniem bazy danych *classicmodels* przedstawionej w poprzednim rozdziale (rys. 9), natomiast działanie sieci w scenariuszu rzeczywistym zostało sprawdzone na modelu *Mutagenesis* (rys. 14). Celem testów było określenie możliwości oferowanych przez strukturę głębokiego grafu asocjacyjnego DASNG, przedstawienie procesu dostrajania parametrów sieci, zaprezentowanie sposobu interpretacji zwracanych wyników, a także porównanie z istniejącymi już rozwiązaniami.

5.1. Podstawowe testy architektury

W pierwszym etapie badań sprawdzano ogólne możliwości i charakterystykę sieci. Opracowano sposób reprezentowania wyników, które następnie poddano podstawowej analizie i oceniono sposób, w jaki architektura rozprowadza sygnał w głąb swojej struktury, określono hiperparametry i zbadano ich wpływ na zwracane wyniki. Opisano również problemy, z którymi napotkano się podczas testów. Rys. 11 przedstawia przykładowy raport formatujący informacje wyjściowe w przystępny sposób. Z poziomu danych wyjściowych, użytkownik zyskuje dostęp do posortowanych i pogrupowanych danych. Sekcja „interval_sorted” listuje wszystkie pobudzone sensory i neurony obiektowe w kolejności, w jakiej były wywoływane, wraz z informacją kiedy dokładnie miało to miejsce i jak często. Analogicznie, w zakładce „counter_sorted” elementy są posortowane pod kątem ilości wywołań, malejąco. Struktura „classes” opisuje klasy (odpowiadają nazwą tabel z przekształcanej bazy danych), których rekordy zostały wywołane. Dodatkowo, każda klasa przechowuje informację o maksymalnie trzech najczęściej wywoływanych obiektach przez nią opisywanych. Pozostałe sekcje raportu informują również o obiektach i sensorach, które zostały wywołane jako pierwsze i ostatnie w całej sieci, a także o ilości unikatowo napotkanych węzłów.

```
{
  "test_params": {
    "time_interval": 0.1,
    "period": 15,
    "value": [[60.86]],
    "table_name": ["buyPrice"]
  },
  "interval_sorted": {
    "sensors": {
      "60.86": {
        "counter": 2,
        "stim_interval": [1.54, 1.99]
      },
      "60.78": {···
      },
      "61.34": {···
      }
    },
    "objects": {
      "products_57": {···
      },
      "products_34": {···
      },
      "products_64": {···
      }
    }
  },
  "counter_sorted": {···
  },
  "classes": [
    [
      "products",
      {
        "counter": 3,
        "entities": [···
        ]
      }
    ]
  ],
  "first_stimulated": {
    "sensor": 60.86,
    "object": "products_57"
  },
  "last_stimulated": {
    "sensor": 61.34,
    "object": "products_64"
  },
  "no_sensors": 3,
  "no_objects": 3,
```


Rys. 11. Przykładowe raport zwracany po zakończeniu działania algorytmu [opracowanie własne].

A)

customerNumber	customerName	contactLastName	contactFirstName	phone	addressLine1	addressLine2	city	state	postalCode	country	salesRepEmployeeNumber	creditLimit
103	Atelier graphique	Schmitt	Carine	40.32.2555	54, rue Royale	NULL	Nantes	NULL	44000	France	1370	21000.00
112	Signal Gift Stores	King	Jean	7025551838	8489 Strong St.	NULL	Las Vegas	NV	83030	USA	1166	71800.00
114	Australian Collectors, Co.	Ferguson	Peter	03 9520 4555	636 St Kilda Road	Level 3	Melbourne	Victoria	3004	Australia	1611	117300.00
119	La Rochelle Gifts	Labrune	Janine	40.67.8555	67, rue des Cinquante Otages	NULL	Nantes	NULL	44000	France	1370	118200.00
121	Baane Mini Imports	Bergulfsen	Jonas	07-98 9555	Erling Skakkes gate 78	NULL	Stavern	NULL	4110	Norway	1504	81700.00
124	Mini Gifts Distributors Ltd.	Nelson	Susan	4155551450	5677 Strong St.	NULL	San Rafael	CA	97562	USA	1165	210500.00

B)

productCode	productName	productLine	productScale	productVendor	productDescription	quantityInStock	buyPrice	MSRP
S10_1678	1969 Harley Davidson Ultimate Chopper	Motorcycles	1:10	Min Lin Diecast	This replica features working kickstand, front su...	7933	48.81	95.70
S10_1949	1952 Alpine Renault 1300	Classic Cars	1:10	Classic Metal Creations	Turnable front wheels; steering function; detail...	7305	98.58	214.30
S10_2016	1996 Moto Guzzi 1100i	Motorcycles	1:10	Highway 66 Mini Classics	Official Moto Guzzi logos and insignias, saddle b...	6625	68.99	118.94
S10_4698	2003 Harley-Davidson Eagle Drag Bike	Motorcycles	1:10	Red Start Diecast	Model features, official Harley Davidson logos a...	5582	91.02	193.66
S10_4757	1972 Alfa Romeo GTA	Classic Cars	1:10	Motor City Art Classics	Features include: Turnable front wheels; steeri...	3252	85.68	136.00
S10_4962	1962 LanciaA Delta 16V	Classic Cars	1:10	Second Gear Diecast	Features include: Turnable front wheels; steeri...	6791	103.42	147.74

C)

officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA
2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA
4	Paris	+33 14 723 4404	43 Rue Jouffroy D'abbans	NULL	NULL	France	75017	EMEA
5	Tokyo	+81 33 224 5000	4-1 Kioicho	NULL	Chiyoda-Ku	Japan	102-8578	Japan
6	Sydney	+61 2 9264 2451	5-11 Wentworth Avenue	Floor #2	NULL	Australia	NSW 2010	APAC

Rys. 12. Przedstawienie przykładowych wypisów z basy danych *clasicmodels*, tabele: A) *customers*, B) *products*, C) *offices* [71].

Na samym początku sieć DASNG poddano ogólnym testom mającym na celu zapoznanie się podstawowymi pryncypiami działania architektury. Badania oparto na modelu bazy danych *classicmodels*. Strukturę pobudzano danymi wejściowymi reprezentującymi tabele *products, customers* i *offices.* Na rysunku 12 przedstawiono przykładowe wypisy z niniejszych tabel. Strukturę DASNG poddano serii 5 testów, gdzie w każdym z nich pobudzanie następowało z wykorzystaniem innej konfiguracji danych wejściowych. Dodatkowo, każdy test był przeprowadzany 3-krotnie, przez zmienną długość działania symulacji: 5 sekund, 15 sekund i 30 sekund. Dane wejściowe dla każdej próby zostały podane poniżej:

- Test 1: {"buyPrice": [60.86]}
- Test 2: {"buyPrice": [60.86, 38.58, 77.90]}
- Test 3: {

  "buyPrice": [60.86],

  "creditLimit": [53800],

  "city": ["Paris"],

  }
- Test 4: {

  "buyPrice": [60.86, 38.58, 77.90],

  "creditLimit": [53800, 72600, 34800],

  "city": ["Paris", "Los Angeles", "Charleroi"],

  }
- Test 5: {"customers": 36}

Tabela 4. Przedstawienie wyników przeprowadzonej analizy rozprzestrzeniania sygnału pobudzania w głąb struktury DASNG w zależności od długości pobudzania [źródło: opracowanie własne]

	Unikatowe sensory			Unikatowe obiekty			Pobudzone klasy		
Czas symulacji [s] \ Przypadek testowy	5	15	30	5	15	30	5	15	30
1	3	3	3	3	3	3	1	1	1
2	18	19	17	13	11	15	2	2	2
3	20	33	62	16	15	39	4	4	4
4	100	102	111	45	64	69	4	4	4
5	35	57	31	16	21	11	3	3	2

Tabela 5. Przedstawienie wyników przeprowadzonej analizy rozprzestrzeniania sygnału pobudzania w głąb struktury DASNG w zależności od częstotliwości pobudzania [źródło: opracowanie własne]

	Unikatowe sensory			Unikatowe obiekty			Pobudzone klasy		
Częstotliwość pobudzania [n/s] \ Przypadek testowy	50	20	10	50	20	10	50	20	10
1	3	2	3	3	2	3	1	1	1
2	19	11	5	11	8	4	2	1	1
3	33	18	17	15	11	13	4	4	4
4	102	63	43	64	22	23	4	3	4
5	35	14	22	16	7	11	3	2	3

W powyższej tabeli 4 można zaobserwować wyniki badania przeprowadzonego na sieci DASNG z wykorzystaniem różnych prób parametrów wejściowych, jak również przy zmiennym czasie wykonywania symulacji. Pierwsza próba polegała na pobudzaniu pojedynczego receptora wybranego pola sensorowego reprezentującego atrybut buyPrice z tabeli *products.* Jak można zaobserwować, wydłużanie czasu symulacji nie wpływało w żaden sposób na to, na jak odległe poziomy sieci sygnał był w stanie się rozprzestrzenić. Wpływa na to liczba pobudzanych sensorów, która w tym przypadku jest równa jeden. Poprzez stymulowanie pojedynczego sensora, nie ma możliwości wejść w interakcję z całą siecią, ze względu na mechanizmy relaksacji i refrakcji. Niezależnie od tego, jak długo struktura działa, sygnał osiąga pewną granicę, której przekroczenie nie jest możliwe, ponieważ zanim nowy impuls zdąży dotrzeć do tego miejsca, poprzedni wygaśnie. Dodatkowo, w teście dochodzi do pobudzenia jedynie neuronów z jednej, unikatowej klasy – *products.* Podobne wnioski można wysnuć z testu drugiego, gdzie liczba wykrywanych sensorów i obiektów reprezentujących rekordy bazy danych jest stosunkowo podobna, niezależnie od długości symulacji. Różnica względem pierwszej próby pojawia się przy liczbie klas, do których należały pobudzane węzły. Można zaobserwować, iż w wynikach pojawiają się elementy nienależące do klasy, z której wywodziły się parametry początkowe.

Analizując próby 3 i 4, można zaobserwować, iż wydłużenie czasu aplikowania stymulacji wpływa pozytywnie na proces odkrywania nowych poziomów sieci. Wzrasta zarówno liczba pulsujących sensorów, jak i neuronów obiektowych. Ponadto, odkrywana jest nowa klasa, która nie posiadała swojego reprezentanta w parametrach wejściowych. W przypadku omawianych wyników jest to klasa *employees,* co ma głęboki sens biorąc pod uwagę, że tabela oznakowana taką etykietą jest połączona relacjami zarówno z tabelami *customers*, jak i *offices* [rys. 9]. Warta zaobserwowania jest również różnica w wynikach pomiędzy próbą 2, a próbą 3. W obu przypadkach pobudzanie wejściowe dotyczyło tylko 3 receptorów, jednak w pierwszej sytuacji było to trzech przedstawicieli jednego atrybutu, natomiast w teście 3 – po jednym przedstawicielu trzech różnych pól sensorowych. Można z tego wnioskować, że większa różnorodność danych wejściowych

przekłada się na bardziej rozległe zbadanie całej sieci. Jednocześnie zwracane wyniki są bardziej ogólne, brak im wyspecjalizowania. W sytuacjach, w których konieczne jest znalezienie odpowiedzi na zadane sieci pytanie lub sklasyfikowanie nieznanych parametrów wejściowych, lepsze może okazać się podawanie bardziej sprecyzowanych zapytań.

W piątej próbie przetestowano możliwości innego podejścia do pobudzania sieci; stymulacji dokonywano od strony neuronów obiektowych. Analizując ostatni rząd tabeli 5 można wnioskować, iż wykonywanie symulacji w taki sposób znacząco zwiększa liczbę zwracanych wyników w porównaniu, np. do pierwszego testu. Dzieje się tak, ponieważ pojedynczy neuron przechowuje wskaźniki do wielu sensorów reprezentujących różne atrybuty, dlatego pomimo pobudzania tylko jednego elementu sieci, zwracana jest duża przetwarzana jest znaczna jej część. Wymaga to jednak, aby osoba wykonująca eksperymenty posiadała dostęp do wszystkich neuronów sieci, co nie zawsze może być możliwe.

Tabela 5 przedstawia odmienne podejście do dostrajania parametrów wejściowych sieci, ponieważ uwzględnia częstotliwość, z jaką wywoływano metodę pobudzania receptorów. Jak można zaobserwować na podstawie niemal wszystkich przypadków testowych, mniejsza ilość stymulacji na sekundę przekłada się na znaczne pogorszenie rozprzestrzeniania sygnału, a co za tym idzie spadek ilości odkrywanych węzłów sieci. Im bardziej rozległe dane wejściowe, a co za tym idzie więcej przypadków do obsłużenia, tym większe spadki odkrywanych obiektów można zaobserwować. Kluczowe z wyznaczaniu hiperparametrów sieci jest znalezienie złotego środka pomiędzy długością stymulacji, a częstotliwością przykładania pobudzenia. Możliwe jest jednak określenie głównej reguły, wedle której częstotliwość stymulowania powinna być możliwie niska, natomiast długość, z jaką powinien działać algorytm, można dostosowywać od potrzeb projektanta i tego, jak rozległa jest przetwarzana baza danych.

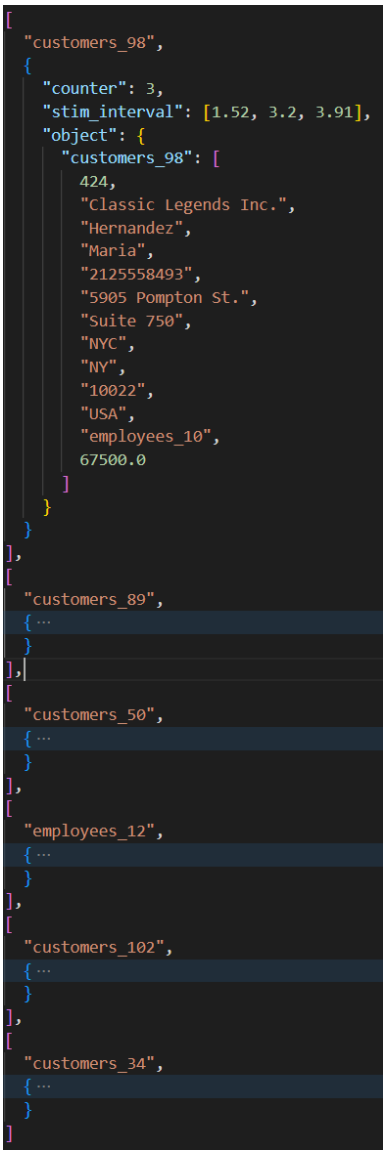
Analiza raportów zwracanych przez sieć pozwoliła dodatkowo określić kilka ogólnych wniosków, którymi można opisać charakter działania struktury asocjacyjnego grafu DASNG. Po pierwsze, czas, z jakim pobudza się sieć, ma znaczenie tylko do momentu osiągnięcia pewnej granicy, której początkowy sygnał nie jest w stanie przekroczyć niezależnie od tego, jak długo jest wydzielany. Kluczem do optymalnego działania algorytmu jest znalezienie tego czasu, tak więc powinien być on uznawany za jeden z hiperparametrów. Można również zaznaczyć, że rozpoczynanie pobudzania sieci od obiektów, najczęściej i najszybciej będzie prowadziło integrację z przypisanymi do nich sensorami. Analogicznie, stymulując sensory, sygnał najszybciej dotrze do odpowiadającym im neuronom obiektowym. Ważne jest jednak wspomnieć, iż sensory przechowujące wartość liczbową mogą również rozprzestrzeniać sygnał wszerz, tj. do swoich sąsiadów. Oznacza to, że w przypadkach, w których znaczna ilość neuronów korzysta z jednego sensora waga, z jaką sygnał będzie uwalniany będzie wyższa dla sąsiadów w polu sensorowym, przez co sygnał dotrze do nich szybciej.

5.2. Testy szczegółowe

W następnym etapie testowania struktury DASNG skupiono się na sprawdzeniu jak sieć poradzi sobie w bardziej szczegółowych, rzeczywistych przypadkach wnioskowania na zbiorze danych. Pierwszy przypadek opierał się na wyszukiwaniu pośrednich relacji łączących dwa zbiory parametrów wejściowych. Sieć miała za zadanie zwrócić podzbiór obiektów, które były współdzielone między wprowadzonymi przypadkami testowymi. Ponownie sprawdzono, jaka jest optymalna częstotliwość i długość stymulowania receptorów, a następnie zestawiono wyniki pomiędzy kolejnymi próbami.

- Test 1: {"creditLimit": [53800, 72600, 34800]}
- Test 2: {"city": ["Paris", "Los Angeles", "Charleroi"]}

Próba wykorzystuje cechę przypisaną do klienta, a także miasta, w których znajdują się biura sklepów obsługujących zamówienia. Ważne jest zaznaczenie, iż dane wejściowe wcale nie muszą być ze sobą w bezpośredni sposób powiązane, tzn. pobudzony klient nie musiał być obsługiwany sklep znajdujący się w danym mieście. Sieć i tak powinna być w stanie znaleźć nawet najmniej znaczące powiązanie pomiędzy tymi parametrami. Przykładowy raport z wykonanego testu, w którym zwrócone obiekty zostały posortowane względem pierwszeństwa w wywołaniu można zaobserwować na rysunku 13. Ze zwróconych danych wynika, że pierwszym węzłem sieci, który był wspólny dla dwóch różnych zestawów parametrów wejściowych, był obiekt reprezentujący krotkę z tabeli *customers*. Warto zauważyć, iż żadna z cech tego obiektu nie pokrywa się z danymi wejściowymi, co oznacza, że w czasie 1,52 s została wykryta pierwsza relacja pośrednia pomiędzy wartościami, którymi sieć była stymulowana.



Rys. 13. Przykładowy raport zwracany w wyniku wykonania operacji wyznaczenia części wspólnej ścieżki sygnałów pochodzących od różnych parametrów wejściowych [opracowanie własne].

Podobnie jak w testach ogólnego działania sieci, w tym etapie czas stymulowania sieci był zmienny. Częstotliwość pobudzania była stała i wynosiła 50/s. W poniższej tabeli zapisano wyniki otrzymane w trakcie tej próby.

Tabela 6. Statystyka prowadzona przy poszukiwaniu pośrednich relacji w zależności od długości stymulacji [źródło: opracowanie własne].

Czas symulacji [s]	Odkryte korelacje	Pierwsze wystąpienie [s]	Odkryte klasy
15	6	1,52	customers, employees
45	12	0,77	customers, employees
90	23	0,73	customers, employees, offices

Wyniki zawarte w tabeli 6 sugerują, że dłuższy czas przeprowadzanej stymulacji ma znaczny wpływ na ilość odkrywanych relacji. W analizowanym przypadku, ma to również bezpośrednie przełożenie na różnorodność klas, z których pochodzą pobudzane obiekty. Jednym z pobudzanych obiektów jest reprezentant klasy *employees*. Na schemacie bazy danych *classimodels* (rys. 9), pobudzona klasa łączy tabele *customers* i *offices*, których reprezentanci znajdowali się w parametrach wejściowych do sieci. Jednym z czynników wpływających na działanie architektury DASNG jest obsługa mechanizmu pulsowania elementów sieci, przez co priorytetyzowane są najczęściej pobudzane neurony. Również w przypadku tego testu częstotliwość z jaką obiekty są pobudzane, będzie wskazywała na siłę relacji zachodzącej pomiędzy zapytaniami wejściowymi. Na uwagę zasługuje również czas, po jakim algorytm zwraca pierwsze powiązania. Jest on równoznaczny z 3-4-krotnym pobudzeniem sensorów reprezentujących dane wejściowe.

Na podstawie wszystkich opisanych do tej pory aspektów działania algorytmu grafu asocjacyjnego DASNG możliwe było utworzenie mapy relacji, które występującą w bazie danych *classicmodels* na przykładzie 6 wybranych atrybutów (tabela 7). Tabela zawiera najczęściej wywoływane połączenia, czyli takie, które występują pomiędzy najbardziej skorelowanymi atrybutami. Na potrzeby opisu wybrano po 3 relacje, jednak raport z działania algorytmu zawiera wszystkie, które wystąpiły w czasie trwania symulacji. Wszystkie testy były przeprowadzane na 5 losowo wybranych receptorach reprezentujących dany atrybut. Możliwe jest oczywiście utworzenie takiej macierzy dla każdego dostępnego atrybutu, jednak wraz z rosnącą ilością cech do sprawdzenia, wydłuża się czas potrzebny do przeprowadzenia tej operacji. Jedną z możliwości optymalizacji algorytmu w przyszłości, mogłoby być zaimplementowanie mechanizmu zapisywania dotychczas wykrytych relacji, wraz z ich siłą w postaci nowych połączeń i neuronów sieci, co również jest przewidziane w sposobie działania sieci DASNG. Zaoszczędziłoby to czasu potrzebnego na ponowne wyliczanie korelacji, które pojawiły się przy wcześniejszy inferencjach.

Tabela 7. Najczęściej występujące relacje pomiędzy wybranymi atrybutami [źródło: opracowanie własne].

Atrybut	Najczęściej występujące relacje	Siła relacji
creditLimit	customerNumber	0,140
	addressLine1	0,120
	phone	0,100
city	creditLimit	0,200
	phone	0,09
	customerNumber	0,09
buyPrice	productScale	0,140
	productVendor	0,140
	productName	0,140
lastName	employeeNumber	0,262
	firstName	0,120
	email	0,120
productScale	quantityInStock	0,179
	buyPrice	0,179
	productVendor	0,154
phone	customerNumber	0,137
	city	0,118
	postalCode	0,118

Tabela 8. Porównanie precyzji wybranych algorytmów eksploracji danych do zadania klasyfikacyjnego na zbiorze danych *Mutagenesis* [źródło: opracowanie własne].

	DASNG	Aleph [72]	CILP++ [72]	Random Forest [74]	kFOIL [75]	TILDE [73]
Precyzja	0,722	0.809	0.892	0.729	0.813	0.894

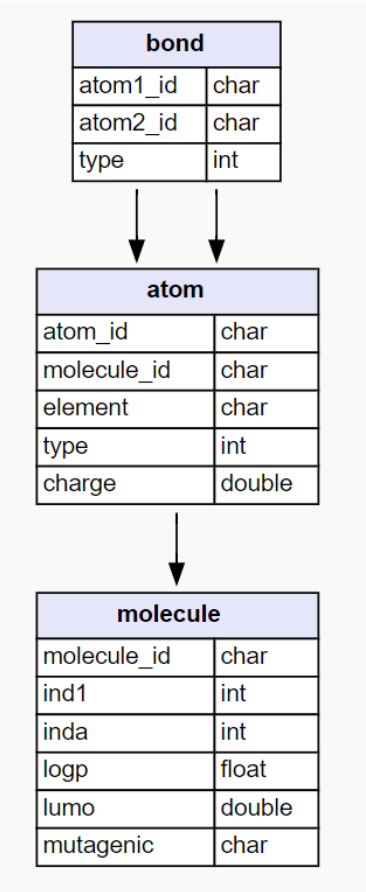
W ostatniej serii testów algorytm został sprawdzony pod kątem zadań klasyfikowania rekordów nieznanych przez sieć. Wykorzystano bazę danych *Mutagenesis* [76] przechowującą informacje na temat molekuł badanych pod kątem mutagenności na Salmonella typhimurium [77]. Baza danych składała się z trzech tabel tworzących charakterystyki molekuł, atomów z których były tworzone, a także wiązania występującego pomiędzy konkretnymi atomami. Schemat modelu przedstawia rysunek 14. Przed przekształceniem bazy do struktury DASNG, wydzielono z niej sekcję złożoną z 18 rekordów, która miała odpowiadać za testowanie wyuczonej sieci. Następnie, dane testowe były przekształcone do postaci odpowiedniej dla sieci, w formie parametrów wejściowych. Celem symulacji było określenie czy molekuła utworzona z dwóch atomów reprezentowane przez konkretne cechy, klasyfikuje się jako zmutowana. Z raportu wynikowego wyciągana była informacja o najczęściej pulsującym obiekcie z klasy *molecule*. Jeśli parametr mutagenności odpowiadał temu ze zbioru walidacyjnego, test uznawany był za pozytywny. Po wykonaniu wszystkich testów celność algorytmu

spisano w formie współczynnika precyzji. Otrzymany wynik zestawiono z innymi rozwiązaniami, co zostało przedstawione w tabeli 8. Analiza powyższej tabeli, jak również raportów z poszczególnych symulacji wykazała, że algorytm DASNG stosunkowo dobrze sprawdza się z w zadaniach klasyfikacyjnych. Potencjalną poprawę wyników mogło by zapewnić lepsze dostrojenie hiperparametrów symulacji. Dodatkowo, zaobserwowano, iż w przypadku przeprowadzonych testów klasyfikacji pierwszy pobudzony neuron obiektowy nie zawsze pokrywał się z obiektami pulsującymi najczęściej. W tabeli 9 przedstawiono 6 losowo wybranych prób z opisanego wyżej testu.

Tabela 9. Szczegółowe wyniki dotyczące pobudzania neuronów obiektowych w próbach klasyfikacji nieznanych przez strukturę DASNG danych [źródło: opracowanie własne].

Oczekiwany wynik (*molecule_id*)	Najwcześniej zwrócony obiekt	Najczęściej pulsujący obiekt	Unikatowe sensory	Unikatowe obiekty	Liczba atrybutów wejściowych	Pobudzone atrybuty
d101	d20	d101	39	1107	4	4
d110	d110	d110	93	583	4	4
d118	d118	d118	261	621	4	4
d123	d123	d36	77	473	4	5
d131	d87	d87	14	3	4	3
d178	d178	d178	101	395	4	5

Precyzję klasyfikowania wyliczano w oparciu o najczęściej pulsujący neuron i wynosiła ona 72,2%. Opracowanie wyników zostało oparte na cesze grafu neuronowego DASNG, według której największe powinowactwo względem receptorów startowych osiągnie właśnie obiekt, którego próg pobudzenia jest przekraczany z najwyższą częstotliwością. Dla kontrastu sprawdzono również, jaka byłaby miara precyzji, jeśli sprawdzany by był zawsze pierwszy występujący neuron poszukiwanej klasy; wyniosła ona niewiele mniej – 66,6%. Warto mieć jednak na uwadze, że takie podejście znacznie przyspiesza proces klasyfikowania, ponieważ algorytm symulacji można przerwać w momencie wykrycia pierwszego obiektu-kandydata. Dalsza analiza tabeli 9 pozwala zaobserwować, że liczba odkrywanych sensorów i obiektów nie jest spójna i powtarzalna. Może to być spowodowane stopniem unikatowości stymulowanych na samym początku sensorów. Ponadto, niektóre próby docierają również do sensorów należących do innych atrybutów niż te wejściowe, co pokrywa się z poprzednim wnioskiem. Bardziej precyzyjna ścieżka rozchodzenia się sygnału pozwala dotrzeć do bardziej oddalonych w przestrzeni wartości.



Rys. 14. Schemat bazy danych *Mutagenesis* [76].

6. Podsumowanie

Niniejsza praca połączyła zagadnienia inteligencji obliczeniowej i problematyki big data, przedstawiając propozycję algorytmu asocjacyjnego, korzystającego z podstawowych konceptów sztucznych sieci neuronowych. Jednocześnie, zdolności struktury do przechowywania zagregowanych danych, umożliwiały łatwy i szybki dostęp do zdefiniowanych zapytań. W części teoretycznej skupiono się na przeglądzie najczęściej wykorzystywanych rozwiązań w zakresie sieci neuronowych, przedstawiono rysy historyczne, a także przykładowe zastosowania. Ponadto, w dokładny sposób zostały opracowane informacje o strukturach danych, zarówno tych podstawowych, jak i nieco bardziej skomplikowanych. Przegląd teoretyczny pozwolił na wybór najbardziej kluczowych dla sieci DASNG cech, dokładne zrozumienie i usprawnienie implementacji.

Całość części praktycznej została zrealizowana z wykorzystaniem środowiska programistycznego Python. Dzięki swojej obiektowej naturze był odpowiednim narzędziem służącym do opisania całej architektury. Zakres pracy pokrywał utworzenie od podstawi architektury DASNG, ale również jej części pośrednich tj. ASA-grafy. W implementacji używano jedynie bibliotek znajdujących się w pakiecie standardowym języka – nie korzystano z żadnych zewnętrznych modułów. Dane uczące pobierane były z baz danych MySQL dostępnych w ramach otwartych licencji.

W pracy przedstawiono nowe podejście do zagadnienia eksploracji danych poprzez implementację i przedstawienie teoretycznych zagadnień opisujących działanie asocjacyjnych grafów neuronowych DASNG, po raz pierwszy przedstawionych przez w [59]. Algorytm korzysta z nowatorskiego mechanizmu neuronów impulsowych, których wkład w badania nad sieciami neuronowymi, z roku na rok jest coraz większy. Wykorzystanie takiego rodzaju obiektów pozwoliło na komputerowe zasymulowanie biologicznego mechanizmu potencjału czynnościowego, dostarczając wskazówek jak rzeczywiste komórki nerwowe procesują sygnał nerwowy. W stosunku do klasycznych modeli uczenia maszynowego, wykorzystanie mechanizmów zdarzeń, pulsowania, częstotliwości pobudzania, wymagało zmiany podejścia do sposobu reprezentowania danych wyjściowych. DASNG produkuje odpowiedzi w oparciu o szybkość i częstotliwości wydzielanych przez neurony impulsów, które reprezentują najbardziej powiązane wartości lub obiekty wewnątrz struktury sieci.

Struktura DASNG operuje na zasadach naśladujących działanie rzeczywistych neuronów. Dzięki temu bardzo dobrze służy jako narzędzie służące do badania biologii naszego układu nerwowego, a w szczególności mechanizmów potencjału czynnościowego. Początkowe testy oprogramowania pozwoliły na zapoznanie się z podstawowymi pryncypiami działania architektury. Na podstawie zmiennej długości czasu i częstotliwości pobudzania za pomocą konkretnego bodźca, możliwa była ocena sposobu w jaki sygnał rozchodzi się wewnątrz sieci, a także gdzie leży granica rozprzestrzeniania się impulsu. Taki sposób działania algorytmu pozwala na automatyczne określanie stopnia powinowactwa jednych obiektów względem drugich. Dzięki tej cesze sieci, możliwe jest utworzenie mapy korelacji zachodzących pomiędzy wszystkimi atrybutami przetwarzanych danych. Ostatnim przydatnym mechanizmem

DASNG jest zdolność do określania części wspólnej podgrafów ścieżek pobudzeń dwóch różnych zestawów parametrów wejściowych. Jest to kolejna właściwość mówiąca o sposobie działania sieci, a także pozwalająca na wzbogacanie zestawu danych o nowe relacje. Warto zaznaczyć, że większość opisanych operacji i wyniki przez nie zwracane, w dużej mierze zależą od stopnia zróżnicowania danych wejściowych, długości i częstotliwości pobudzania sieci, a także długości czasu trwania stanów relaksacji i refrakcji. Dzięki temu struktura jest bardzo plastyczna i w łatwy sposób można ją dostosowywać do wymagań projektantów badań.

DASNG jako struktura neuronowa rozwinięta w trakcie badań, która pozwala na sortowanie, filtrowanie i grupowanie danych ze względu stopień relacji pomiędzy obiektami znajdującymi się w sieci sprawdziła się bardzo dobrze. Nie jest to jednak architektura pozbawiona wad. Ze względu na swoją charakterystykę, bardzo duże zbiory danych mogą znacznie wydłużać proces wyszukiwania pożądanych relacji i cech, zwłaszcza przy standardowej implementacji sekwencyjnej, bez wykorzystania wielowątkowości i możliwości procesorów graficznych. Dodatkowo, badania mechanizmu klasyfikacji wykazały, że struktura sprawdza się w tego typu zadaniach stosunkowo dobrze, lecz nie posiada aż takiej skuteczności jak niektóre algorytmy wykorzystywane w dziedzinach uczenia maszynowego. Ostatnim zauważonym problemem było „gubienie" niektórych zdarzeń odpowiadającym zmianom stanów neuronów w przypadku długich symulacji o wysokiej częstotliwości pobudzania. Mogło to być związane ze sposobem implementacji kolejki GEQ jako struktury mapującej zdarzenie do odpowiedniej chwili w przyszłości. Przeszukiwanie kolejki w poszukiwaniu konkretnego punktu w czasie mogło go pomijać, jeśli operacje związane z poprzednimi zdarzeniami zajmowały zbyt dużo czasu. Problem mógł być również nasilony przez ograniczenia sprzętowe. Opisane utrudnienie nie wpływało jednak w znacznym stopniu na wyniki i procesy wnioskowania sieci.

Implementacja algorytmu pozwoliła na znaczne rozszerzenie wiedzy z zakresu budowy struktur danych, programowania obiektowego oraz realizacji programistycznego projektu badawczego, czerpiącego w znacznym stopniu z zagadnień biologicznych. Ponadto, praca nad projektem wymagała rozwinięcia umiejętności projektowych, oraz nabycia zdolności do stosowania ustalonych praktyk programistycznych – niezwykle istotnych w przemyśle.

Przedstawiona implementacja nie jest wolna od błędów i istnieje wiele możliwości na usprawnienie jej działania. Do głównych zagadnień, które z powodzeniem mogą ulepszyć niniejszy algorytm należy wymienić:

- Skorzystanie z biblioteki numpy celem zwiększenia wydajności algorytmu, zwłaszcza w przypadkach wykonywania wielu intensywnych pętli,
- Zrównoleglenie mechanizmu pobudzania poprzez wykorzystanie wielowątkowości procesora,
- Utworzenie interfejsu graficznego dla lepszego zobrazowania procesów zachodzących wewnątrz struktury w trakcie mechanizmu pobudzania,

- Mechanizm automatycznego aktualizowania sieci o odkrywane połączenia, tak aby ponowne uruchamianie algorytmu było inteligentniejsze o odkryte wcześniej ścieżki.

7. Bibliografia

1. **Cisco Visual Networking Index: Forecast and Trends**, 2017–2022
2. https://bernardmarr.com/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#:~:text=The%20amount%20of%20data%20we,in%20the%20world%20was%20generated [dostęp 26.11.2022 r.]
3. McCulloch, Warren S., and Walter Pitts. **A Logical Calculus of the Ideas Immanent in Nervous Activity.** *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, Dec. 1943, pp. 115–33. *Springer Link*
4. D. O. Hebb. **The Organization Of Behavior A Neuropsychological Theory.** 1949. *Internet Archive*
5. Rosenblatt, F. The Perceptron: **A Probabilistic Model for Information Storage and Organization in The Brain**. *Psychol. Rev.* 1958, 65–386,
6. Foote, Keith D. **A Brief History of Neural Networks**. *DATAVERSITY*, 9 Nov. 2021
7. Widrow B., Lehr M. A.. **Artificial Neural Networks of the Perceptron, Madaline, and Backpropagation Family.** 1993. *Elsevier Science Publishers*
8. Minsky M., Papert S. A., **Perceptrons**. 15 Jan. 1969. *MIT Press*
9. Werbos, P. J. **The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting**. *Wiley*, 1994.
10. Fukushima K. ,**Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.** *Biological Cybernetics*, **36** (1980) 93-202.
11. Rumelhart, D., Hinton, G. & Williams, R. **Learning representations by back-propagating errors**. *Nature* **323**, 533–536 (1986).
12. LeCun, Y., et al. **Handwritten Digit Recognition with a Back-Propagation Network**. *Advances in Neural Information Processing Systems*, vol. 2, Morgan-Kaufmann, 1989.
13. http://yann.lecun.com/ex/research/index.html [dostęp 26.11.2022 r.]
14. https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html [dostęp 26.11. 2022 r.]
15. Uhrig, R. E. (n.d.). **Introduction to artificial neural networks**. *Proceedings of IECON* '95 - 21st Annual Conference on IEEE Industrial Electronics.
16. Tadeusiewicz, R., **Neurocybernetyka Teoretyczna**. Warsaw University Press, 11-16, 2009.

17. http://galaxy.agh.edu.pl/~vlsi/AI/bias/bias.html [dostęp 26.11.2022 r.]

18. Nandakumar, S. R., Kulkarni, S. R., Babu, A. V., & Rajendran, B. (2018). **Building Brain-Inspired Computing Systems: Examining the Role of Nanoscale Devices**. *IEEE Nanotechnology Magazine*, 12(3), 19–35.

19. **Neural Networks**, https://www.ibm.com/cloud/learn/neural-networks [dostęp 26.11.2022 r.]

20. Albawi S., Mohammed T. A. and Al-Zawi S., **Understanding of a convolutional neural network,** *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1-6

21. Katok, A, and Thouvenot J. P. **Spectral Properties and Combinatorial Constructions in Ergodic Theory**. *Handbook of Dynamical Systems*, vol. 1, Elsevier, 2006, pp. 649–743.

22. https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html [dostęp 26.11.2022 r.]

23. https://miroslawmamczur.pl/jak-dzialaja-konwolucyjne-sieci-neuronowe-cnn/ [dostęp 26.11.2022 r.]

24. Gu, J., et al. **Recent Advances in Convolutional Neural Networks**. *Pattern Recognition*, vol. 77, May 2018, pp. 354–77.

25. O'Shea K., Nash R. **An Introduction to Convolution Neural Networks.** *arXiv*, 2 Dec. 2015

26. Salehinejad, H., et al. **Recent Advances in Recurrent Neural Networks**. *arXiv*, 2018.

27. Géron, A. **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.** 2nd edition, *Helion*, 2020.

28. Mikolov, T., Joulin, T., Chopra, S., Mathieu, M., and Ranzato, M., **Learning longer memory in recurrent neural networks.** *arXiv*, 2014.

29. Elsayed, E. **Recurrent Neural Networks.** Jan. 2020.

30. Hochreiter, S., Schmidhuber, J., **Long Short-Term Memory.** *Neural Computation* 9(8):1735-1780, 1997.

31. Ba, J. L., et al. **Layer Normalization**. *arXiv*, 21 Jul. 2016.

32. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks [dostęp 26.11.2022 r.]

33. Ghosh-Dastidar, S. & Adeli, H.. (2009). **Spiking Neural Networks**.. *Int. J. Neural Syst..* 19. 295-308. 10.1142/S0129065709002002.

34. Horzyk, A. & Starzyk, J.. (2018). **Multi-Class and Multi-Label Classification Using Associative Pulsing Neural Networks**. 10.1109/IJCNN.2018.8489176.

35. Horzyk, A. & Starzyk, J.. (2017). **Fast Neural Network Adaptation with Associative Pulsing Neurons**. 10.1109/SSCI.2017.8285369.

36. Wysoski, S. G., et al. **Fast and Adaptive Network of Spiking Neurons for Multi-View Visual Pattern Recognition**. *Neurocomputing*, vol. 71, no. 13–15, Aug. 2008, pp. 2563–75.

37. Tavanaei, A., and Maida, A. **Bio-Inspired Multi-Layer Spiking Neural Network Extracts Discriminative Features from Speech Signals**. *Neural Information Processing*, edited by Derong Liu et al., Springer International Publishing, 2017, pp. 899–908

38. Kasabov, N., et al. **Evolving Spiking Neural Networks for Personalised Modelling, Classification and Prediction of Spatio-Temporal Patterns with a Case Study on Stroke**. *Neurocomputing*, vol. 134, June 2014, pp. 269–79.

39. Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., & Maida, A. (2019**). Deep learning in spiking neural networks**. *Neural Networks*, 111, 47–63.

40. Sejnowski, T. J., **Open questions about computation in the cerebral cortex**, in D. E. Rumelhart and J. L. McClelland (eds.), *Parallel Distributed Processing*, MIT Press, Cambridge, MA, 2, 1986, pp. 372–389
41. Maass, W., **Lower bounds for the computational power of spiking neural networks**, *Neural Computation* 8(1) (1996) 1–40.

42. Hodgkin, Al., Huxley, Af.. **A quantitative description of membrane current and its application to conduction and excitation in nerve**. *J Physiol*. 1952 Aug;117(4):500-44.

43. Izhikevich, E. M., **Simple model of spiking neurons**, in *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569-1572, Nov. 2003

44. Hojjatinia S, Aliyari Shoorehdeli M, Fatahi Z, Hojjatinia Z, Haghparast A. **Improving the Izhikevich Model Based on Rat Basolateral Amygdala and Hippocampus Neurons, and Recognizing Their Possible Firing Patterns.** *Basic Clin Neurosci*. 2020 Jan-Feb;11(1):79-90.

45. Fang, X., et al. **Memristive Izhikevich Spiking Neuron Model and Its Application in Oscillatory Associative Memory.** *Frontiers in Neuroscience*, vol. 16, 2022.

46. Abbott LF. **Lapicque's introduction of the integrate-and-fire model neuron (1907)**. *Brain Research Bulletin*. 1999 Nov-Dec;50(5-6):303-4

47. **Integrate-And-Fire Models**. *Neuronal Dynamics Online Book*. https://neuronaldynamics.epfl.ch/online/Ch1.S3.html [dostęp 26.11.2022 r.]

48. Teeter, C., et al. **Generalized Leaky Integrate-and-Fire Models Classify Multiple Neuron Types**. *Nature Communications*, vol. 9, no. 1, Feb. 2018, p. 709.

49. Nordlie, E., et al. **Rate Dynamics of Leaky Integrate-and-Fire Neurons with Strong Synapses**. *Frontiers in Computational Neuroscience*, vol. 4, 2010.

50. Burkitt, A. (2006). **A Review of the Integrate-and-fire Neuron Model: I. Homogeneous Synaptic Input**. *Biological cybernetics*. 95. 1-19. 10.1007/s00422-006-0068-6.

51. Horzyk, A. & Starzyk, J. (2017). **Fast Neural Network Adaptation with Associative Pulsing Neurons**. 10.1109/SSCI.2017.8285369.

52. https://www.ibm.com/docs/en/imdm/12.0?topic=associations-data [dostęp 26.11.2022 r.]

53. Orzechowski, P. & Horzyk, A. & Starzyk, J. & Moore, J. (2018). **Retrieving Impressions from Semantic Memory Modeled with Associative Pulsing Neural Networks**. 10.1109/SSCI.2018.8628780.

54. Horzyk, A. (2017). **Deep Associative Semantic Neural Graphs for Knowledge Representation and Fast Data Exploration**. 10.5220/0006504100670079.

55. Wang JH, Cui S. **Associative memory cells: Formation, function and perspective**. *F1000Res*. 2017 Mar 17;6:283.

56. Vetulani, J. **Jak usprawnić pamięć.** *Platan*. 52-53. 2003

57. **The Ultimate Guide to Understand the Differences Between Stack and Queue.** https://www.simplilearn.com/tutorials/data-structure-tutorial/stacks-and-queues [dostęp 26.11.2022 r.]

58. Ralston, A., Reilly, E. D., Hemmendinger D., **Data Structures.** *Encyclopedia of Computer Science.* 4th Edition. *Wiley.* 507-512. 2003

59. Horzyk, A. & Bulanda, D. & Starzyk, J. (2020). **ASA-graphs for efficient data representation and processing**. *International Journal of Applied Mathematics and Computer Science*. 30. 717-731. 10.34768/amcs-2020-0053.

60. Kalat, J.W., **Biological Psychology**, Belmont, CA: *Wadsworth Publishing*. 2012

61. **How Are Memories Formed**. 2 Dec. 2016. https://qbi.uq.edu.au/brain-basics/memory/how-are-memories-formed [dostęp 26.11.2022 r.]

62. Feher, J. **The Action Potential**. *Quantitative Human Physiology*, Elsevier, 2017, pp. 265–79.

63. Caviezel, M. P., et al. **The Neural Mechanisms of Associative Memory Revisited: FMRI Evidence from Implicit Contingency Learning**. *Frontiers in Psychiatry*, vol. 10, 2020.

64. Dai, X., et al. **Convolutional Embedding for Edit Distance**. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 599–608.

65. https://www.linuxjournal.com/content/hash-tables—theory-and-practice [dostęp 26.11.2022 r.]

66. **Data Structures and Algorithms**. https://cathyatseneca.gitbooks.io/data-structures-and-algorithms/content/ [dostep 26.11.2022 r.]

67. **Data Modelling**. https://www.ibm.com/cloud/learn/data-modeling [dostęp 26.11.2022 r.]

68. Batra, D., & Davis, J. G. (1992). **Conceptual data modelling in database design: similarities and differences between expert and novice designers.** *International Journal of Man-Machine Studies, 37(1), 83–101.*

69. Sherman, R. **Foundational Data Modeling**. *Business Intelligence Guidebook*, Elsevier, 2015, pp. 173–95.

70. Data Modelling. https://aws.amazon.com/what-is/data-modeling/ [dostęp 26.11.2022 r.]

71. https://relational.fit.cvut.cz/dataset/ClassicModels [dostęp 26.11.2022 r.]

72. Manoel, V. M., França, Zaverucha G., Garcez A. **Fast Relational Learning using Bottom Clause Propositionalization with Artificial Neural Networks.** 2013

73. Dinh, Q. T., et al. ***A Link-Based Method for Propositionalization***. 2012, p. à paraître

74. Vens, C., Van Assche, A., Blockeel, H., Džeroski, S. (2004). **First Order Random Forests with Complex Aggregates**. In: Camacho, R., King, R., Srinivasan, A. (eds) *Inductive Logic Programming*. ILP 2004. Lecture Notes in Computer Science(), vol 3194. Springer, Berlin, Heidelberg

75. Landwehr, N., Passerini, A., De Raedt, L., Frasconi, P. **kFOIL: Learning Simple Relational Kernels.** *American Association for Artificial Intelligence.* 2006

76. http://www.cs.ox.ac.uk/research/ai_ml/index.html [dostęp 26.11.2022 r.]

77. Debnath, A. K., et al. **Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with Molecular Orbital Energies and Hydrophobicity**. *Journal of Medicinal Chemistry*, vol. 34, no. 2, Feb. 1991, pp. 786–97.

78. Casale, A. E., and D. A. McCormick. **Active Action Potential Propagation But Not Initiation in Thalamic Interneuron Dendrites**. *Journal of Neuroscience*, vol. 31, no. 50, Dec. 2011, pp. 18289–302.