Realizacja modeli pamięci skojarzeniowej przy pomocy sieci neuronowych

Bartłomiej Domański

Kwiecień 2024

Spis treści

1 Wstęp **5**

2 Klasyczna sieć Hopfielda **7**
2.1 Funkcja energetyczna . . . 8
2.1.1 Przesłuch . . . 9
2.2 Analogie do modelu Isinga . . . 11

3 Gęsta pamięć asocjacyjna **15**
3.1 $S > N$ - realizacja alternatywy rozłącznej . . . 16

4 Ciągła sieć Hopfielda i HopfieldLayer **19**
4.1 Ciągła sieć Hopfielda a Transformer . . . 21
4.2 HopfieldLayer . . . 21

5 Implementacja pamięci skojarzeniowej **23**
5.1 Przystosowanie modeli . . . 23
5.1.1 Klasyczna sieć Hopfielda . . . 23
5.1.2 Gęsta pamięć asocjacyjna . . . 23
5.1.3 HopfieldLayer . . . 24
5.2 Liczba wzorców a skuteczność modeli . . . 24
5.3 Eksperymentalne wyznaczenie pojemności klasycznej sieci Hopfielda . . . 25

6 Pamięć skojarzeniowa a problem klasyfikacji **27**
6.1 Proste algorytmy konstrukcji wzorców klasyfikacyjnych . . . 27
6.1.1 Średnia arytmetyczna . . . 27
6.1.2 Litera "matka" . . . 27
6.2 Gradientowa konstrukcja wzorca klasyfikacyjnego . . . 29
6.2.1 Gradient prosty "vanilla" . . . 30
6.2.2 Gradient prosty "Mini-batch" . . . 30
6.2.3 Adagrad . . . 30

7 Pamięć skojarzeniowa i dane eksperymentalne **35**
7.1 Rozpoznanie równania ruchu . . . 35
7.2 Rozpoznawanie wartości argumentów . . . 35

8 Analiza wyników **39**

9 Podsumowanie **41**

10 Poza konkursem **45**
10.0.1 Ciągła klasyczna sieć Hopfielda . . . 45
10.0.2 Proste zwiększanie pojemności sieci - neurony ukryte . . . 45
10.0.3 Zapominanie wzorców przez sieć . . . 46
10.0.4 Maszyny Boltzmanna . . . 46
10.0.5 Gradient bez gradientu . . . 50
10.0.6 Mediana . . . 50

Rozdział 1

Wstęp

Początki rozwoju sztucznej inteligencji sięgają lat 40. XX wieku, kiedy to w 1943 roku Warren McCulloch i Walter Pitts zaproponowali model sztucznej neuronu, nazywany perceptronem. Perceptron był pierwszą próbą matematycznego modelu biologicznego neuronu, który miał za zadanie symulować procesy myślowe człowieka. Model zakładał sumowanie sygnałów docierających do perceptronu i na tej podstawie określenie czy perceptron jest aktywny czy niekatywny. W latach 50. i 60. XX wieku wykazano ograniczenia perceptronu, okazało się, że pojedynczy perceptron nie jest zdolny do rozwiązywania problemów nieliniowych. To odkrycie, w połączeniu z brakiem odpowiednich danych i mocy obliczeniowej, spowodowało spadek zainteresowania sieciami neuronowymi.

W tym okresie większość badań koncentrowała się na innych podejściach do sztucznej inteligencji, dopiero w latach 80. XX wieku, dzięki pracy Johna Hopfielda nad sieciami rekurencyjnymi, zainteresowanie sztucznymi sieciami neuronowymi odrodziło się. Sieci Hopfielda przyniosły nowe spojrzenie na potencjał sztucznych sieci neuronowych, kładąc fundamenty dla dalszych badań nad uczeniem się maszynowym i pamięcią skojarzeniową (adresowaną zawartością). Obecnie sztuczne sieci neuronowe są uważane za najważniejsze narzędzie w dziedzinie sztucznej inteligencji, umożliwiając znaczący postęp w rozpoznawaniu obrazów, przetwarzaniu języka naturalnego, robotyce i wielu innych dziedzinach. Dzięki zaawansowanym algorytmom uczenia maszynowego są one teraz w stanie wykonywać zadania o złożonej naturze, które jeszcze niedawno wydawały się niemożliwe do automatyzacji.

Celem niniejszej pracy jest porównanie modeli pamięci skojarzeniowej - klasycznej sieci Hopfielda, gęstej pamięci asocjacyjnej oraz ciągłej sieci Hopfielda. Podstawą wszystkich trzech jest zaproponowana przez Johna Hopfielda idea opisania sztucznej sieci neuronowej poprzez funkcję energetyczną. Początkowo przeprowadzę teoretyczna analizę modeli i ich właściwości. W części eksperymentalnej przedstawię implementacje oraz wyniki działania konkretnych modeli. Praktyczne aspekty poruszane w pracy obejmują także wykorzystanie modelu pamięci skojarzeniowej w zagadnieniu klasyfikacji danych oraz analizie danych pomiarowych, rozważę potencjalne zastosowania badanych modeli w praktyce oraz perspektyw ich rozwoju w przyszłości.

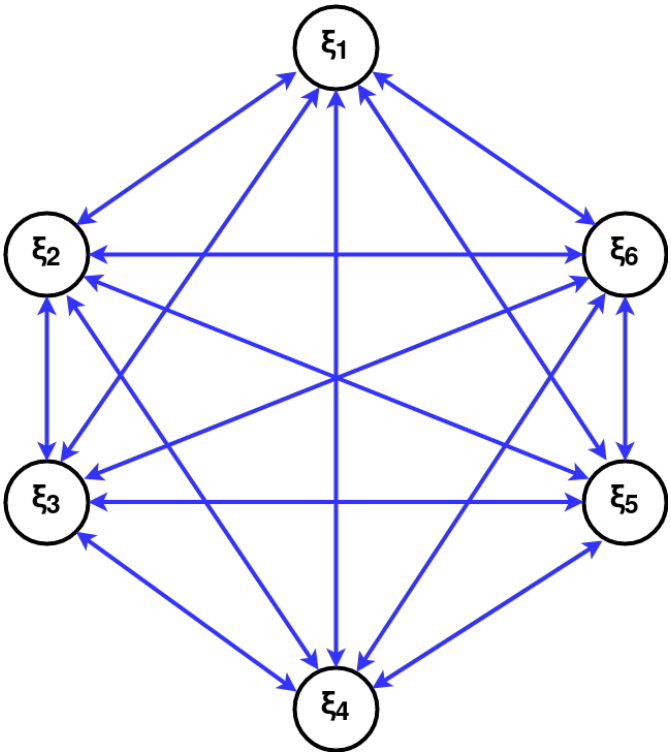
Istotą pamięci skojarzeniowej jest zdolność do odtwarzania danych na podstawie części tych danych lub uszkodzonych fragmentów. W ludzkim mózgu pamięć skojarzeniowa jest związana z możliwością łączenia różnych elementów informacji i tworzenia skojarzeń pomiędzy nimi, proces ten pozwala na przypomnienie sobie pełnych wspomnień albo sekwencji wydarzeń na podstawie jednego elementu lub fragmentu informacji. Kluczową rolę w tym procesie odgrywają połączenia między komórkami nerwowymi.

Podobnie w sztucznych sieciach neuronowych pamięć skojarzeniowa jest symulowana poprzez tworzenie połączeń między neuronami. W procesie uczenia maszynowego, modele oparte na pamięci skojarzeniowej są zdolne do przypisywania wag poszczególnym elementom danych i tworzenia skojarzeń pomiędzy nimi. To pozwala sztucznym systemom na skuteczną rekonstrukcję na podstawie części dostępnych danych.

Rozdział 2

Klasyczna sieć Hopfielda

Klasyczna sieć Hopfielda (sieć asocjacyjna) to model rekurencyjnej sieci neuronowej spopularyzowany w 1982 roku przez Johna Hopfielda. [4] Został opisany po raz pierwszy przez Little'a i Sawa w 1974 roku jako modyfikacja modelu Isinga. Podstawowy model składa się z połączonych ze sobą obustronnie perceptronów.



Każdy z perceptronów ξ jest bipolarny, to znaczy że przyjmuje one wartość ze zbioru {-1, 1}. Każde ξ odbiera sygnał do wszystkich innych neuronów oraz może wysłać sygnał do wszystkich poza sobą samym. Zgodnie z modelem perceptronu [3] neuron jest aktywny (przyjmuje wartość 1) lub jest nieaktywny (przyjmuje wartość -1) na podstawie równania

$$\xi_i = \Theta[\sum_j (W_{ij}\xi_j + b_i)] \tag{2.1}$$

gdzie

$$\Theta(a) = \begin{cases} 1 & a >= 0 \\ -1 & a < 0 \end{cases} \tag{2.2}$$

W równaniu 2.1 rozważamy sumę sygnałów docierających ze wszystich neuronów przemnożonych przez odpowiadającą wagę W_{ij}. Waga określa na ile istotny jest konkretny docierający sygnał, wszystkie wagi zawierają się w macierzy W. Połączenia są symetryczne [4], więc

$$W_{ij} = W_{ji} \tag{2.3}$$

a dodatkowo aby obsłużyć warunek braku połączenia neuronu z samym sobą wprowadza się

$$W_{ii} = 0 \tag{2.4}$$

Dodatkowy wyraz b_i to próg i-tego neuronu, tak zwany bias. Podczas gdy model perceptronu był prosty i abstrakcyjny, Hopfield próbował stworzyć model bliższy biologii ludzkiego mózgu poprzez połączenie perceptronów.

Klasyczna sieć Hopfielda o wielu neuronach zmienia się bardzo dynamicznie ze względu na ilość połączeń modyfikujących stan sieci. Rozważamy dwa sposoby uporządkowania zmian stanu:

- Modyfikacja synchroniczna - wszystkie neurony obliczają swoją aktywację (argument funkcji Θ), a następnie jednocześnie modyfikują swój stan
- Modyfikacja asynchroniczna - w danym momencie pojedynczy neuron oblicza swoją aktywację i modyfikuje swój stan, kolejne neurony do przeprowadzenia tej operacji są wybierane losowo albo z ustaloną wcześniej kolejnością.

W kontekście pamięci skojarzeniowej zależy nam aby sieć potrafiła przechowywać pewne dane (wzorce) a następnie je rozpoznawać i odtwarzać na podstawie danych uszkodzonych lub niepełnych, tak jak robi to ludzki mózg. Wzorzec jest w zasadzie N-elementowym wektorem x o bipolarnych wartościach {-1,1}, gdzie N to liczba neuronów ξ w sieci. Ten wektor reprezentuje konkretny stan sieci, więc liczba jego elementów musi być równa liczbie neuronów. Podstawowa metoda uczenia sieci Hopfielda danego wzorca jest nazywana regułą Hebba. Zgodnie z nią połączenie między dwoma neuronami wzmacnia się jeśli aktywują się one jednocześnie i osłabia się jeśli aktywują się oddzielnie [6], można to określić proporcjonalnością $W_{ij} \sim \xi_i \xi_j$. Reguła uogólniona dla wielu wzorców może zostać przedstawiona jako [3]

$$W_{ij} = \frac{1}{S} \sum_{\mu=1}^{S} \sum_{i,j}^{N} x_i^\mu x_j^\mu = \frac{1}{S} \sum_{\mu=1}^{S} x^\mu (x^\mu)^T \tag{2.5}$$

gdzie S to liczba wzorców, a N to liczba neuronów.

2.1 Funkcja energetyczna

Rozważmy iloczyn stanu neuronu i docierającego do niego sygnału, ξ_i to aktualny stan, a ξ_i^{new} to stan po aktualizacji. Jeżeli znak obu wartości jest ten sam to różnica iloczynów jest równa 0.

$$\xi_i^{new} * \sum_j (W_{ij}\xi_j + b_i) - \xi_i * \sum_j (W_{ij}\xi_j + b_i) = 0 \tag{2.6}$$

Jeżeli znak jest inny:

$$\xi_i^{new} * \sum_j (W_{ij}\xi_j + b_i) - \xi_i * \sum_j (W_{ij}\xi_j + b_i) = 2\xi_i^{new} * \sum_j (W_{ij}\xi_j + b_i) \tag{2.7}$$

W tym przypadku różnica jest dodatnia, biorąc pod uwagę oba równania wnioskiem jest, że wartość iloczynu nigdy nie maleje. Niech funkcja D opisuje te iloczyny globalnie:

$$D(\xi_1, \xi_2, ..., \xi_N) = \sum_i \xi_i * \sum_j (W_{ij}\xi_j + b_i) = \sum_{i,j} W_{ij}\xi_i\xi_j + \sum_i b_i\xi_i \tag{2.8}$$

Jeśli neuron zmieni swój stan, spowoduje to wzrost wartości D. Funkcja D ma górne ograniczenie:

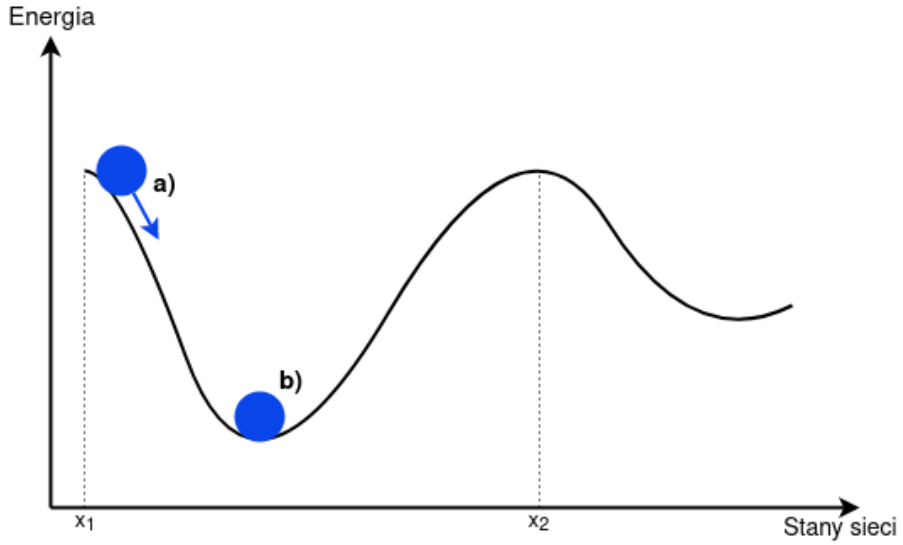
$$D^{max} = \sum_{i,j} |w_{ij}| + \sum_i |b_i| \tag{2.9}$$

Każda zmiana stanu neuronu doprowadza do wzrostu wartości D, więc w skończonej ilości kroków funkcja jest zbieżna. Po przemnożeniu przez -1 i normalizacji pierwszego składnika (ze względu na symetrię wag) uzyskujemy:

$$E = -\frac{1}{2} \sum_{i,j}^{N} W_{ij}\xi_i\xi_j - \sum_i b_i\xi_i = -\frac{1}{2}\xi^T W \xi - \xi^T b \tag{2.10}$$

Powyższa funkcja to zaproponowana przez Hopfielda funkcja energetyczna opisująca sieć neuronową. Zależy ona od stanu wszystkich neuronów oraz wag połączeń między nimi. Opis jest inspirowany hamiltonianem znanego z fizyki statystycznej modelu Isinga.

$$E_{Ising} = -\frac{1}{2} \sum_{<i,j>} J_{ij} S_i S_j - \sum_i h_i S_i \tag{2.11}$$



Rysunek 2.1: Schematyczny wykres energii sieci Hopfielda. Oś pozioma reprezentuje konkretne scenariusze stanu sieci (stanu wszystkich neuronów). Punkty x_1 i x_2 reprezentują zakres basenu atraktora, czyli zakres w którym sieć zbiega do danego lokalnego minimum energii (atraktora). Sieć rozpoczyna działanie w stanie a) i po kolejnych aktualizacjach osiąga stan b) odpowiadający lokalnemu minimum energii.

Oryginalnie model opisuje zachowanie ferromagnetyka poprzez układ dyskretnych spinów ułożonych w węzły sieci. Odpowiednikiem spinów S_i są neurony ξ_i, zaś rolę całki wymiany J_{ij} pełnią wagi W_{ij}. Dodatkowo model Isinga zakłada oddziaływanie z zewnętrznym polem magnetycznym, w tym miejscu sieć Hopfielda wprowadza wyrazy wolne (bias) b_i.

Sieć Hopfielda podczas kolejnych aktualizacji zmniejsza wartość funkcji energetycznej aż w końcu osiąga minimum lokalne i przestaje zmieniać swój stan. Minima lokalne tej funkcji są nazywane atraktorami, a obszary w których funkcja zbiega do danego minimum to baseny atraktora [3], tak jak na rysunku 2.1.

W praktyce idealne nauczenie sieci jest trudne i wzorce na których nam zależy nie są jedynymi istniejącymi atraktorami. Podczas uczenia sieci danego wzorca tak naprawdę "zapamiętuje" ona oprócz wzorca właściwego także jego przeciwną wersję o wartościach odwrotnych. Obrazowo, jeżeli sieć ma atraktor odpowiadający wektorowi (1, 1, 1, -1) to znaczy, że ma też atraktor odpowiadający wektorowi (-1, -1, -1, 1). Jest to bezpośredni skutek symetrii wag w połączeniach między neuronami. Dodatkowo w wyniku błędów w uczeniu może dochodzić do powstawania atraktorów w stanach które nie odpowiadają żadnemu z wybranych wzorców.

2.1.1 Przesłuch

Rozważmy stabilność konkretnego wzorca x^ν, w ramach stabilności oczekujemy, że:

$$\Theta(h_i) = x_i^\nu \quad \forall i \tag{2.12}$$

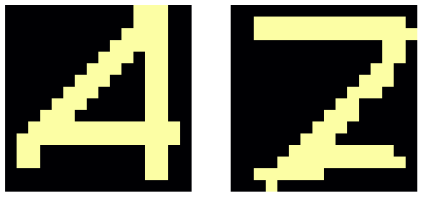
Gdzie h_i jest wkładem sieci do neuronu i. W oparciu o 2.1 i 2.5 można zapisać:

$$h_i = \sum_j^N W_{ij} x_j^\mu = \frac{1}{N} \sum_j^N \sum_m^S x_i^m x_j^m x_j^\mu \tag{2.13}$$

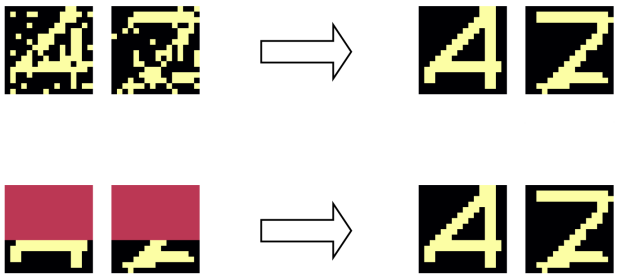
Powyższa forma to podejście od innej strony do obliczenia sygnału docierającego do neuronu i. Rozważamy wpływ wszystkich wzorców m na odtwarzanie konkretnego wzorca μ. 2.13 może zostać rodzielone na dwa wyrazy, część pochodzącą od poszukiwanego wzorca oraz od wszystkich innych:

$$h_i = x_i^\mu + \frac{1}{N} \sum_j^N \sum_{m \neq \mu}^S x_i^m x_j^m x_j^\mu \tag{2.14}$$

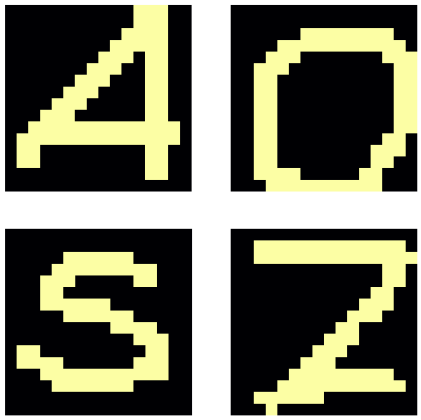
Drugi składnik powyższej sumy to tak zwany przesłuch czyli składnik pochodzący od innych wzorców, który utrudnia odtworzenie wzorca x^μ. Jeśli przesłuch jest zaniedbywalnie mały w stosunku do pierwszego składnika to nie zmienia on znaku h_i. Wartością przy której przesłuch odwraca około 1 procent aktywacji jest liczba wzorców $S_{max} = 0.138N$, powyżej może dochodzić do lawinowego wzrostu liczby błędów i permanentnej niestabilności sieci [3]. Uzyskanie tego wyniku wymaga szczegółowej analizy związku klasycznej sieci Hopfielda z modelem Isinga.



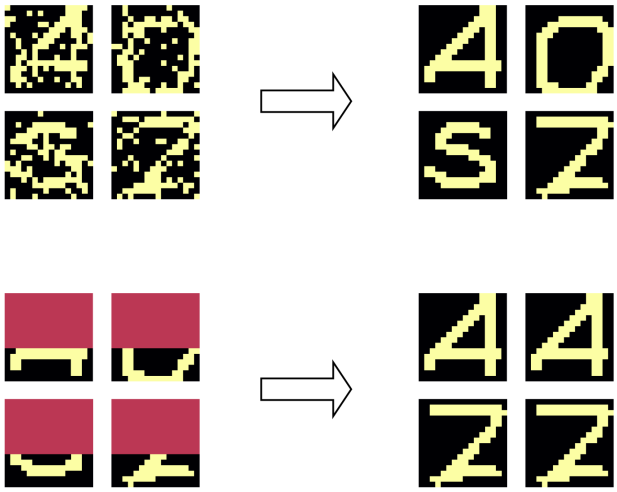
(a) Dane treningowe pierwszej sieci



(b) Rekonstrukcja danych w pierwszej sieci



(c) Dane treningowe drugiej sieci



(d) Rekonstrukcja danych w drugiej sieci

Rysunek 2.2: Przykład działania klasycznej sieci Hopfielda. Jako dane treningowe, czyli wzorce, posłużyły obrazki 16x16 pikseli przedstawiające litery. Każdy piksel odpowiada jednemu neuronowi sieci, tak więc sieć ma 256 neuronów. Kiedy zadaniem jest odtworzenie niewielkiej ilości wzorców (tutaj 2) pierwsza sieć bezproblemowo radzi sobie zarówno z danymi zaszumionymi jak i częściowymi. Przy zwiększaniu liczby wzorców zaprezentowanych drugiej sieci nie udało się zrekonstruować danych, dwukrotnie sieć zatrzymała się w złym minimum energetycznym.

2.2 Analogie do modelu Isinga

Rozważmy model namagnesowanego materiału z oznaczeniami sieci Hopfielda, na każdy spin wpływa zewnętrzne pole magnetyczne.

$$b_i = \sum_j W_{ij}\xi_j + b^{ext} \tag{2.15}$$

Współczynnik W_{ij} określa wpływ spinu ξ_j w polu magnetycznym na spin ξ_i, te wpływy są symetryczne. W niskiej temperaturze spiny ustawiają się zgodnie z równaniem $\xi_i = \Theta(h_i)$, te zmiany następują asynchronicznie w losowej kolejności. Hamiltonian określający powyższe założenia:

$$H = -\frac{1}{2}\sum_{ij} W_{ij}\xi_i\xi_j - b^{ext}\sum_i \xi_i \tag{2.16}$$

W przypadku gdy temperatura nie jest niska dochodzi do fluktuacji zamieniających wartości spinów, co zaburza ich ustawianie się zgodnie z polem magnetycznym. Im wyższa temperatura tym większy wpływ fluktuacji na zachowanie spinów. [3]

Dynamika Glaubera opisuje matematycznie efekt fluktuacji na model Isinga, prawdopodobieństwo ustalenia spinu w konkretnym stanie wynosi:

$$P(\xi_i := 1) = g(b_i) \tag{2.17}$$

$$P(\xi_i := -1) = 1 - g(b_i) \tag{2.18}$$

$$g(h) = \frac{1}{1 + exp(-2\beta b)} \tag{2.19}$$

gdzie

$$\beta = \frac{1}{k_B T} \tag{2.20}$$

Funkcja $g(b)$ (w tym przypadku sigmoida) zależy od temperatury T panującej w układzie. Jako że $1 - g(h) = g(-h)$ dynamika zmiany spinów może zostać opisana jako:

$$P(\xi_i := \pm 1) = g(\pm b_i) = \frac{1}{1 + exp(\mp 2\beta b_i)} \tag{2.21}$$

Temperatura bezpośrednio wpływa na kształt sigmoidy $g(b)$, przy temperaturze bliskiej zeru sigmoida zbliża się do funkcji Θ wykorzystywanej w klasycznej sieci Hopfielda.

W celach demonstracyjnych można zastosować powyższą dynamikę do opisu pojedynczego spinu, w tym przypadku uzyskamy średnią magnetyzację (średnia z jednego spinu równoważna z jednym spinem):

$$\xi_i = <\xi_i> = P(\xi_i = 1) * 1 + P(\xi_i = -1) * (-1) = \tag{2.22}$$

$$= \frac{1}{1 + exp(-2\beta b_i)} - \frac{1}{1 + exp(2\beta b_i)} = \frac{exp(\beta b_i) - exp(-\beta b_i)}{exp(\beta b_i) - exp(-\beta b_i)} = tanh(\beta b_i) \tag{2.23}$$

Ten wynik można zastosować także dla układu N spinów jeśli wpływa na nich to samo zewnętrzne pole magnetyczne a spiny nie wpływają na siebie, taki układ to paramagnetyk z magnetyzacją $M = N < S >$.

W przypadku kiedy każdy spin jest źródłem lokalnego pola magnetycznego należy zastosować przybliżenie, które okazuje się przydatne w analizie modelu Hopfielda. Teoria pola średniego zakłada zastąpienie "fluktuującego" b_i poprzez: [3]

$$< b_i >= \sum_j W_{ij} < \xi_j > + b^{ext} \tag{2.24}$$

wówczas równanie średniej magnetyzacji 2.23 przyjmuje postać:

$$< \xi_i >= tanh(\beta < b_i >) = tanh(\beta \sum_j W_{ij} < \xi_j > + \beta b^{ext}) \tag{2.25}$$

Ideą teorii jest wybranie pojedynczego spiniu i zastąpienie wszystkich innych poprzez średnie zewnętrzne pole magnetyczne. Fluktuacje pozostałych spinów nie są brane pod uwagę, także po zmianie wybranego spinu.

Rzeczywista temperatury oczywiście nie dotyczy sztucznych sieci neuronowych, w ich przypadku można zdefiniować pseudo-temperaturę wyrażaną jako $\beta = \frac{1}{T}$, tak aby wciąż móc opisać zmianę kształtu sigmoidy i tangensa

hiperbolicznego. Zakładając niewielką ilość wspomnień w sieci oraz łącząc teorię pola średniego z równaniem 2.5 uczenia Hebba można opisać stochastyczną sieć neuronową:

$$< \xi_i >= tanh(\frac{\beta}{N} \sum_{j,\mu} x_i^\mu x_j^\mu < \xi_j >) \tag{2.26}$$

Powyższe równanie nie ma prostego rozwiązania, dla uproszczenia można przyjąć, że $< \xi_i >$ jest proporcjonalne do jednego ze wspomnień w sieci:

$$< \xi_i >= m x_i^\nu \tag{2.27}$$

$$m x_i^\nu = tanh(\frac{\beta}{N} \sum_{j,\mu} x_i^\mu x_j^\mu m x_j^\nu) \tag{2.28}$$

Podobnie jak w klasycznej sieci Hopfielda argument tangensa hiperbolicznego może zostać rozbity na wyraz proporcjonalny do wspomnienia oraz przesłuch. Przy aktualnym założeniu małej ilości wspomnień, przesłuch jest pomijalny, w związku z tym:

$$m x_i^\nu = tanh(\beta m x_j^\nu) \tag{2.29}$$

a jako że $tanh(-a) = -tanh(a)$:

$$m = tanh(\beta m) \tag{2.30}$$

Dla takiej postaci równania stany sieci odpowiadające wspomnieniom są stabilne dla temperatury mniejszej lub równej 1. Przyjmując nazewnictwo z opisu ferromagnetyków, temperatura krytyczna dla stochastycznej sieci z małą ilością wspomnień wynosi 1. Podejmując równanie średniej magnetyzacji 2.27 można zapisać:

$$m = \frac{< \xi_i >}{x_i^\nu} = P(bit\ i\ jest\ poprawny) - P(bit\ i\ jest\ niepoprawny) \tag{2.31}$$

Dodatkowo średnia ilość poprawnych spinów w otrzymanej odpowiedzi sieci wynosi [3]

$$< N_{poprawnych} >= \frac{1}{2} N(1+m) \tag{2.32}$$

Powyżej temperatury krytycznej $< N_{poprawnych} >$ wynosi $\frac{N}{2}$ co jest po prostu liczbą poprawnych jednostek, której można się spodziewać przy losowej odpowiedzi, $< N_{poprawnych} >$ zmierza do N (wszystkie poprawne) dla niskich temperatur.

Powyższa analiza jest słuszna dla małej liczby wspomnień ($S \ll N$), w celu analizy rzeczywistej pojemności klasycznej sieci Hopfielda wprowadzam parametr obciążenia:

$$\gamma = \frac{S}{N} \tag{2.33}$$

Punktem wyjścia jest równanie łączące teorię pola średniego z teorią Hebba 2.26, z tą różnicą że tym razem przesłuch nie może zostać pominięty, teraz skupiamy się na wszystkich wspomnieniach, także tych mieszanych czyli nakładających się na siebie.

$$m_\nu = \frac{1}{N} \sum_i x_i^\nu < \xi_i > \tag{2.34}$$

Zakładamy, że badamy odzyskiwanie wzorca numer 1. Wtedy m_1 ma rząd jedności, podczas gdy każde z m_ν dla $\nu \neq 1$ jest małe, rzędu $\frac{1}{\sqrt{N}}$ dla losowych wspomnień. Wielkość r, która jest średnim kwadratem nakładania się konfiguracji systemu z nieodzyskanymi wspomnieniami, ma rząd jedności.

$$r = \frac{1}{\gamma} \sum_{\nu \neq 1} m_\nu^2 \tag{2.35}$$

Równość $\frac{1}{\gamma} = \frac{N}{S}$ sprawia, że r jest średnią z S (lub $S-1$) kwadratów nakładania się i anuluje oczekiwaną zależność m_ν rzędu $\frac{1}{\sqrt{N}}$. Równanie pola średniego 2.28 przy większej ilość wspomnień przyjmuje formę

$$m_\nu = \frac{1}{N} \sum_i x_i^\nu tanh(\beta \sum_\mu x_i^\mu m_\mu) \tag{2.36}$$

a po podzieleniu na czynniki z $\mu = 1$ i $\mu = \nu$

$$m_\nu = \frac{1}{N} \sum_i x_i^\nu x_i^1 tanh(\beta(m_1 + x_i^\nu x_i^1 m_\nu + \sum_{\mu \neq 1,\nu} x_i^\mu x_i^1 m_\mu)) \tag{2.37}$$

Pierwszy argument tangensa hiperbolicznego ma rząd 1, trzeci argument też jest znaczący ze względu na większą ilość S wspomnień, ale drugi argument jest pomijalny (rząd $\frac{1}{\sqrt{N}}$) dlatego wykorzystując pochodną tangensu hiperbolicznego można zapisać [3]

$$m_\nu = \frac{1}{N}\sum_i x_i^\nu x_i^1 tanh(\beta(m_1 + \sum_{\mu\neq 1,\nu} x_i^\mu x_i^1 m_\mu)) + \frac{\beta}{N}\sum_i (1 - tanh^2(\beta(m_1 + \sum_{\mu\neq 1,\nu} x_i^\mu x_i^1 m_\mu)))m_\nu \tag{2.38}$$

Niewielkie nakładanie się m_μ, $\mu \neq 1$ może przyjąć małe ujemne lub małe dodatnie wartości, stąd można przybliżyć je jako zmienne losowe o średniej równej zero i wariancji $\frac{\gamma r}{S}$. Składnik $x_i^\mu x_i^1$ jest losowy i niezależny od m_μ więc zgodnie z centralnym twierdzeniem granicznym cała suma jest średnią z Gaussowskiego szumu, to pozwala zredukować drugą połowę równania 2.38 i całe powyższe równanie do postaci:

$$m_\nu = \frac{1}{N}\sum_i x_i^\nu x_i^1 tanh(\beta(m_1 + \sum_{\mu\neq 1,\nu} x_i^\mu x_i^1 m_\mu)) + \beta m_\nu - \beta q m_\nu \tag{2.39}$$

lub

$$m_\nu = \frac{N^{-1}\sum_i x_i^\nu x_i^1 tanh[\beta(m_1 + \sum_{\mu\neq 1,\nu} x_i^\mu x_i^1 m_\mu)]}{1 - \beta(1-q)} \tag{2.40}$$

gdzie

$$q = \int \frac{dz}{\sqrt{2\pi}} exp(-\frac{z^2}{2}) tanh^2[\beta(m_1 + \sqrt{\gamma r}z)] \tag{2.41}$$

Aby teraz policzyć r, bierzemy kwadrat równania 2.40:

$$m_\nu^2 = [\frac{1}{1-\beta(1-q)}]^2 \frac{1}{N^2}\sum_{ij} x_i^\nu x_i^1 x_j^\nu x_j^1 * tanh[\beta(m_1 + \sum_{\mu\neq 1,\nu} x_i^\mu x_i^1 m_\mu)] * tanh[\beta(m_1 + \sum_{\mu\neq 1,\nu} x_j^\mu x_j^1 m_\mu)] \tag{2.42}$$

i liczymy średnią po wszystkich wzorcach tak jak w 2.35. Jako że wzorzec ν nie pojawia się w argumentach tangensa hiperbolicznego, średnie z wyrazów $x_i^\nu x_i^1 x_j^\nu x_j^1$ mogą zostać policzone niezależnie i przetrwa tylko czynnik $i = j$. Pozostała średnia tangensów hiperbolicznych jest niezależna od ν więc:

$$r = \frac{q}{[1-\beta(1-q)]^2} \tag{2.43}$$

W podobny sposób można uzyskać równanie na m_1:

$$m_1 = \int \frac{dz}{\sqrt{2\pi}} exp(-\frac{z^2}{2}) tanh[\beta(m_1 + \sqrt{\gamma r}z)] \tag{2.44}$$

Teraz wszystkie trzy równania na q, m_1 i r mogą zostać policzone numerycznie, najlepiej przeanalizować przypadek temperatury dążącej do 0. W tej granicy q dąży do 1, ale równanie $\beta(1-q)$ pozostaje skończone. W tym miejscu przydatne są przybliżenia: [3]

$$\int \frac{dz}{\sqrt{2\pi}} exp(-\frac{z^2}{2})[1 - tanh^2[\beta(az+b)]] \approx \frac{1}{\sqrt{2\pi}} exp(-\frac{z^2}{2})|_{tanh^2\beta[az+b]=0} * \int dz(1 - tanh^2\beta[az+b]) \tag{2.45}$$

$$= \frac{1}{\sqrt{2\pi}} exp(-\frac{b^2}{2a^2}) \frac{1}{a\beta} \int dz \frac{\partial}{\partial z} tanh[\beta(az+b)] = \sqrt{\frac{2}{\pi}} \frac{1}{a\beta} exp(-\frac{b^2}{2a^2}) \tag{2.46}$$

oraz

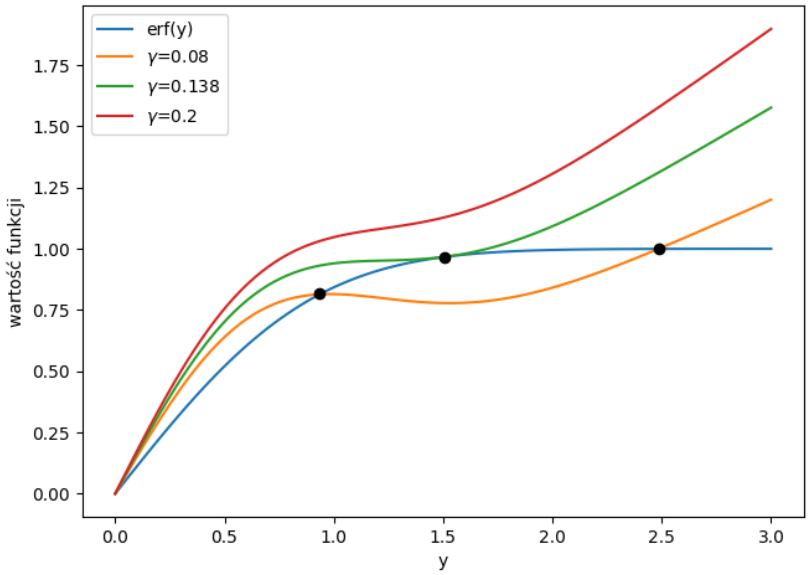
$$\int \frac{dz}{\sqrt{2\pi}} exp(-\frac{z^2}{2}) tanh[\beta(az+b)] \overset{T\to 0}{\to} \int \frac{dz}{\sqrt{2\pi}} exp(-\frac{z^2}{2}) sgn[\beta(az+b)] \tag{2.47}$$

$$= 2\int_{-b/a}^{\infty} \frac{dz}{\sqrt{2\pi}} exp(-\frac{z^2}{2}) - 1 = erf(\frac{b}{\sqrt{2}a}) \tag{2.48}$$

gdzie erf to funkcja błędu. Po użyciu takich przybliżeń można zapisać trzy równania:

$$C \equiv \beta(1-q) = \sqrt{\frac{2}{\pi\gamma r}} exp(-\frac{m^2}{2\gamma r}) \tag{2.49}$$

$$r = \frac{1}{(1-C)^2} \tag{2.50}$$



Rysunek 2.3: Graficzne rozwiązanie równania 2.52 dla trzech wybranych wartości γ. Rozwiązania nietrywialne z $m > 0$ są dane przez przecięcia (oprócz tego w $y = 0$). Istnieje wartość krytyczna γ, powyżej której rozwiązania nietrywialne ($m = 0$) zanikają.

$$m = erf(\frac{m}{\sqrt{2\gamma r}}) \tag{2.51}$$

Definiując $y = \frac{m}{\sqrt{2\gamma r}}$, można zapisać:

$$y(\sqrt{2\gamma} + \frac{2}{\sqrt{\pi}} exp(-y^2)) = erf(y) \tag{2.52}$$

Rozwiązując równanie 2.52 można znaleźć wartość krytyczną γ, przy której rozwiązania nietrywialne ($m = 0$) zanikają. Rozwiązanie numeryczne daje $\gamma \approx 0.138$ [3], oznacza to teoretyczną granicę liczby wzorców $S_{max} \approx 0.138N$.

Rozdział 3

Gęsta pamięć asocjacyjna

Jak wynika z numerycznego rozwiązania, klasyczny model Hopfielda jest skuteczny gdy liczba przechowywanych wzorców jest znacznie mniejsza od liczby neuronów w sieci. Przy zbyt dużej ilości wzorców, niektóre zaczną nakładać się na siebie czyli generować wkład do sieci o tym samym rzędzie. Rozwiązaniem, które zwiększa pojemność sieci jest modyfikacja standardowej funkcji energii do postaci:

$$E = -\sum_{\mu=1}^{S} F[\sum_{i} x_i^\mu \xi_i] \tag{3.1}$$

Funkcja F (nazywana dalej funkcją interakcji) przyjmuje jedną z trzech postaci

$$F(\alpha) = \alpha^n \tag{3.2}$$

$$F(\alpha) = \begin{cases} \alpha^n & \text{dla } \alpha \geq 0 \\ 0 & \text{dla } \alpha < 0 \end{cases} \tag{3.3}$$

lub

$$F(\alpha) = exp(\alpha) \tag{3.4}$$

gdzie n to dodatnia liczba całkowita [5] [2]. W przypadku $n = 2$ sieć redukuje się do klasycznego modelu Hopfielda. Przy $n > 2$ w tej samej przestrzeni konfiguracyjnej można umieścić więcej wzorców zanim zaczną one nakładać się na siebie [5]. Ideą nowej funkcji aktualizacji jest znalezienie różnicy energii między stanem sieci z i-tym neuronem w stanie aktywnym a i-tym neuronem w stanie nieaktywnym.

$$\xi_i = \Theta[E(\xi_i = 1) - E(\xi_i = -1)] \tag{3.5}$$

po bezpośrednim wprowadzeniu równania energii funkcja aktualizacji przyjmuje postać

$$\xi_i = \Theta[\sum_{\mu=1}^{S}(F(1 * x_i^\mu + \sum_{j\neq i} x_j^\mu \xi_j) - F(-1 * x_i^\mu + \sum_{j\neq i} x_j^\mu \xi_j))] \tag{3.6}$$

$$\xi_i = \Theta[\sum_{\mu=1}^{S}(F(x_i^\mu + \sum_{j\neq i} x_j^\mu \xi_j) - F(-x_i^\mu + \sum_{j\neq i} x_j^\mu \xi_j))] \tag{3.7}$$

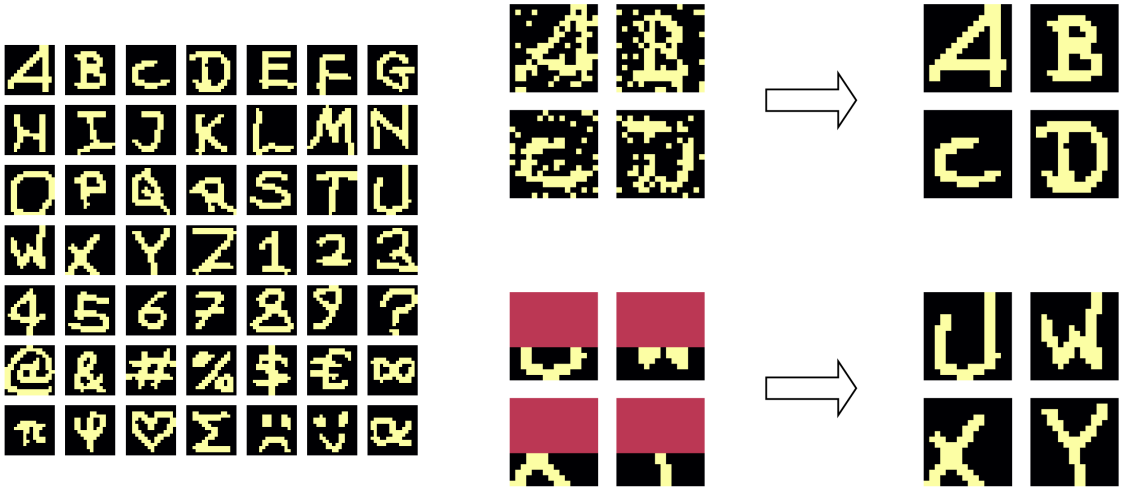
Innymi słowy, wartość neuronu jest ustalona na 1 lub -1 w zależności od tego co jest korzystniejsze przy zmniejszaniu energii całego układu. Jak widać w tym przypadku zmiana stanu neuronu zależy wprost od funkcji energetycznej, odmiennie od podstawowego modelu gdzie energia była tylko pojęciem używanym do abstrakcyjnego opisu algorytmu. Nowy model opisany powyżej nosi nazwę Gęstej pamięci asocjacyjnej.

Jak powyższa zmiana wpływa na pojemność sieci? Załóżmy zestaw wzorców o losowych wartościach ze zbioru {-1,1} oraz sieć zainicjowaną w stanie $x = \xi^\mu$, czyli w stanie odpowiadającym jednemu z wzorców. Różnicę energii między stanem początkowym a stanem w którym i-ty neuron jest odwrócony przy funkcji aktywacji 3.2 można zapisać jako:

$$\Delta E = \sum_{\nu=1}^{S}(x_i^\nu x_i^\mu + \sum_{j\neq i} x_j^\nu x^\mu)^n - \sum_{\nu=1}^{S}(-x_i^\nu x_i^\mu + \sum_{j\neq i} x_j^\nu x^\mu)^n \tag{3.8}$$

Średnia tego równania wynosi [5]:

$$< \Delta E >= N^n - (N-2)^2 \approx 2nN^{n-1} \tag{3.9}$$



(a) Dane treningowe (b) Rekonstrukcja danych

Rysunek 3.1: Przykład działania modelu gęstej pamięci asocjacyjnej. Jako dane treningowe posłużyło 49 obrazków 16x16 pikseli przedstawiających różne symbole. Nowy model bez większych problemów radzi sobie z większą ilością danych odtwarzając zarówno dane zniekształcone jak i niepełne. Zwiekszona pojemność pozwala umożliwia teraz lepsze rozróżnienie skorelowanych wzorców co jest niemożliwe przy tradycyjnej sieci Hopfielda. Wykorzystana funkcja interakcji to $F(\alpha) = \alpha^8$. Przy zaszumionych danych pełne odtworzenie zachodziło zawsze po 1 aktualizacji, dla danych z uciętymi 176 pikselami pełne odtworzenie zachodziło między 1 a 3 aktualizacjami.

wynika to z członu $\nu = \mu$. Wariancja w granicy dużego N wynosi [5]:

$$\sigma^2 = 4n^2(2n-3)!!(S-1)N^{n-1} \tag{3.10}$$

Jeśli wielkość fluktuacji przekracza lukę energetyczną ΔE i jej znak jest przeciwny do znaku luki energetycznej to i-ta jednostka staje się niestabilna. Zatem przy N i K wystarczająco dużych, aby szum uznać za zgodny z rozkładem Gaussa, prawdopodobieństwo, że stan pojedynczego neuronu jest niestabilny jest równe [5]

$$P_{error} = \int_{<\Delta E>}^{\infty} \frac{1}{\sqrt{2\pi Var}} exp(-\frac{y^2}{2\sigma^2})\, dy \approx \sqrt{\frac{(2n-3)!!S}{2\pi N^{n-1}}} exp(-\frac{N^{n-1}}{2S(2n-3)!!}) \tag{3.11}$$

Aby osiągnąć prawdopodobieństwo mniejsze od konkretnej zadanej wartości, należy przyjąć górne ograniczenie liczby wzorców które sieć jest w stanie przechowywać

$$S_{max} = \omega_n N^{n-1} \tag{3.12}$$

gdzie ω to numeryczna stała, zależna od przyjętego progu prawdopodobieństwa. Przypadek $n = 2$ odpowiada wcześniej wyznaczonej wartości $S_{max} \approx 0.138N$, dla idealnego odtwarzania pamięci (czyli $P_{error} < 1/N$) wynik wynosi [5]

$$S_{max}^{noerrors} \approx \frac{N^{n-1}}{2(2n-3)!!ln(N)} \tag{3.13}$$

Przy zwiększaniu potęgi n pojemność rośnie w sposób nieliniowy, umożliwiając sieci skuteczne przechowywanie i odtwarzanie znacznie większej liczby wzorców niż liczba neuronów. Przy małym n wiele wyrażeń nakłada się na siebie równomiernie i utrudnia odtworzenie μ w równaniu 3.7. W granicy $n \to \infty$ dominujący wkład pochodzi z pojedynczego wzorca, który jest najbardziej podobny do stanu początkowego. Optymalne obliczeniowo jest przyjęcie $2 < n \ll \infty$.

3.1 $S > N$ - realizacja alternatywy rozłącznej

Za pomocą analizy prostego przykładu alternatywy rozłącznej (XOR) można zademonstrować, że wraz ze wzrostem n, rosną możliwości obliczeniowe sieci. Dodatkowo liczba wzorców przekroczy liczbę neuronów, przy klasycznej sieci

Hopfielda choćby zbliżenie się do takiego wyniku nie byłoby możliwe. Rozważamy przypadek gdzie prezentacja danych wejściowych x i y zwraca z zgodnie z:

$$z = \begin{cases} -1 & \text{dla } x = -1 \text{ i } y = -1 \\ 1 & \text{dla } x = -1 \text{ i } y = 1 \\ 1 & \text{dla } x = 1 \text{ i } y = -1 \\ -1 & \text{dla } x = 1 \text{ i } y = 1 \end{cases} \tag{3.14}$$

Rolę wzorców przyjmą zestawy $[x, y, z]$ zgodne z powyższymi zasadami, uzyskamy w ten sposób $S = 4$ wzorce przy $N = 3$ neuronach. Równanie energii 3.1 przyjmuje postać:

$$E_n(x, y, z) = -(-x - y - z)^n - (-x + y + z)^n - (x - y + z)^n - (x + y - z)^n \tag{3.15}$$

Dla kilku pierwszych całkowitych n energia jest równa:

$$E_1(x, y, z) = 0 \tag{3.16}$$

$$E_2(x, y, z) = -4(x^2 + y^2 + z^2) \tag{3.17}$$

$$E_3(x, y, z) = 24xyz \tag{3.18}$$

$$E_4(x, y, z) = -4(x^4 + y^4 + 6y^2z^2 + z^4 + 6x^2 + z^2) \tag{3.19}$$

$$E_5(x, y, z) = 80xyz(x^2 + y^2 + z^2) \tag{3.20}$$

W przypadku $n = 1$ funkcja jest równa 0. Dla parzystych n energia jest funkcją parzystą, zmiana wartości któregokolwiek z argumentów na przeciwną nie zmienia wartości energii (przez parzystą potęgę), a więc nie można odczytać pożądanej zmiany energii. Jeżeli n jest nieparzyste i większe od 1 mamy do czynienia z funkcją nieparzystą $E_n(x, y, -z) = E_n(x, y, z)$, generalizując można zapisać:

$$E_n(x, y, z) = \begin{cases} 0 & \text{dla } n = 1 \\ A_n & \text{dla } n = 2, 4, ... \\ A_n xyz & \text{dla } n = 3, 5, ... \end{cases} \tag{3.21}$$

Gdzie A_n jest numeryczną stałą zależną od n. Przy odpowiednim n (3, 5, ...) gęsta pamięć asocjacyjna jest w stanie przyjąć jako dane wejściowe x i y a następnie odtworzyć z poprzez minimalizację wartości energii. Przykładowo dla $n = 3$ reguła aktualizacji 3.5 przyjmuje postać:

$$z = \Theta[E_3(x, y, -1) - E_3(x, y, +1)] \tag{3.22}$$

$$= \Theta[(-x-y-1)^3-(-x-y+1)^3-(x-y-1)^3+(x-y+1)^3-(-x+y-1)^3+(-x+y+1)^3+(x+y-1)^3-(x+y+1)^3] \tag{3.23}$$

$$= \Theta[-48xy] = \Theta[A_3xy] \tag{3.24}$$

Z tak zadaną regułą aktualizacji sieć jest w stanie odtworzyć element z mimo tego, że liczba wzorców przekracza liczbę neuronów. Dla wielomianów rektyfikowanych (funkcja interakcji 3.3) podobny problem jest możliwy do rozwiązania dla dowolnego całkowitego $n > 2$.

Rozdział 4

Ciągła sieć Hopfielda i HopfieldLayer

Klasyczna sieć Hopfielda i Gęsta pamięć asocjacyjna stanowią pewne rozwiązania zagadnienia pamięci asocjacyjnej, jednak ich głównym problemem jest operowanie jedynie na bipolarnych wartościach {-1,1} więc także operowanie na nieciągłej funkcji energii. We współczesnych architekturach sztucznych sieci neuronowych powszechnie wykorzystywane jest obliczanie gradientu funkcji kosztu w procesie uczenia, do czego konieczna jest ciągła postać takiej funkcji (tu funkcja energii).

Energia Gęstej pamięci asocjacyjnej 3.1 ograniczona do funkcji interakcji 3.4 może zostać zapisana w postaci:

$$E_{gpa} = -\sum_{\mu=1}^{S} exp(\sum_{i} x_i^{\mu}\xi_i) = -\sum_{\mu=1}^{S} exp[(x^{\mu})^T\xi] = -exp(lse(1, X^T\xi)) \tag{4.1}$$

gdzie lse jest wypukłą funkcją zdefiniowaną jako:

$$lse(\beta, a) = \beta^{-1}log(\sum_{i=1}^{N} exp(\beta a_i)) \tag{4.2}$$

a $X = (x^1, ..., x^S)$ to macierz przechowująca wszystkie wzorce. Taka postać nadal odtwarza jedynie bipolarne stany ξ. Energia nowego modelu Ciągłej sieci Hopfielda oparta o 4.1 jest zdefiniowana jako: [7]

$$E = ln(E_{gpa}) + \frac{1}{2}\xi^T\xi + \beta log(S) + \frac{1}{2}M^2 = -lse(\beta, X^T\xi) + \frac{1}{2}\xi^T\xi + \beta log(S) + \frac{1}{2}M^2 \tag{4.3}$$

gdzie M to największa z norma wzorców $M = max_{\mu}||x^{\mu}||$. Nowe równanie energii składa się z logarytmu energii Gęstej pamięci asocjacyjnej i wyrażenia kwadratowego bieżącego stanu. Wyrażenie kwadratowe zapewnia, że norma wektora stanu ξ pozostaje skończona, a energia jest ograniczona [7]. Poprzednie modele nie wymagają tego ograniczenia, ponieważ wektor stanu jest bipolarny a więc także skończony. Dwa ostatnie wyrazy, czyli $\beta log(S)$ i $\frac{1}{2}M^2$ pełnią rolę normalizacyjną i gwarantują, że energia ma ograniczony przedział. Wprowadzając wzorzec najbardziej podobny do aktualnego stanu ξ, czyli $x^{\xi} = x^k$ gdzie $k = arg\ max_i \xi^T x^i$, rozważmy 4.3 w postaci:

$$E = -lse(\beta, X^T\xi) + \frac{1}{2}\xi^T\xi + \beta log(S) + \frac{1}{2}M^2 = -\beta^{-1}ln(\sum_{i=1}^{S} exp(\beta(x^i)^T\xi)) + \beta^{-1}ln(S) + \frac{1}{2}\xi^T\xi + \frac{1}{2}M^2 = \tag{4.4}$$

$$= -\beta^{-1}ln(\frac{1}{S}\sum_{i=1}^{S} exp(\beta(x^i)^T\xi)) + \frac{1}{2}\xi^T\xi + \frac{1}{2}M^2 \geq -\beta^{-1}ln(\frac{1}{S}\sum_{i=1}^{S} exp(\beta(x^i)^T\xi)) + \frac{1}{2}\xi^T\xi + \frac{1}{2}(x^{\xi})^T x^{\xi} \geq \tag{4.5}$$

$$\geq -\beta^{-1}ln(exp(\beta(x^{\xi})^T\xi)) + \frac{1}{2}\xi^T\xi + \frac{1}{2}(x^{\xi})^T x^{\xi} = \beta^{-1}\beta(x^{\xi})^T\xi + \frac{1}{2}\xi^T\xi + \frac{1}{2}(x^{\xi})^T x^{\xi} = \tag{4.6}$$

$$= (x^{\xi})^T\xi + \frac{1}{2}\xi^T\xi + \frac{1}{2}(x^{\xi})^T x^{\xi} = \frac{1}{2}(\xi - x^{\xi})^T(\xi - x^{\xi}) = \frac{1}{2}||\xi - x^{\xi}||^2 \geq 0 \tag{4.7}$$

Zawsze których z wzorców będzie najbliższy do aktualnego stanu, stąd ograniczenie jest globalne. Z drugiej strony E jest ograniczone przez $2M^2$ [7], stąd można zapisać:

$$0 \leq E \leq 2M^2 \tag{4.8}$$

Nowa reguła aktualizacji przyjmuje postać

$$\xi = Xp = Xsoftmax(\beta X^T\xi) \tag{4.9}$$

gdzie softmax to znormalizowana funkcja eksponencjalna:

$$p_i = [softmax(a)]_i = \frac{exp(a_i)}{\sum_k exp(a_k)} \tag{4.10}$$

Argument β odpowiada za dynamikę uczenia, przy wysokich wartościach baseny atraktorów konkretnych wzorców są dobrze oddzielone od innych co eliminuje powstawanie niestabilnych stanów sieci. Reguła aktualizacji klasycznej sieci Hopfielda 2.1 bez biasu i z oznaczeniami ciągłej sieci Hopfielda przyjęłaby postać:

$$\xi = \Theta(WW^T\xi) = \Theta(XX^T\xi) \tag{4.11}$$

W równaniu 4.9 funkcja Θ jest zbędna - ciągła sieć Hopfielda, jak nazwa wskazuje, operuje na ciągłych wartościach zamiast bipolarnych. W miejsce wektora $X^T\xi$ wykorzystano $softmax(\beta X^T\xi)$, można więc określić regułę aktualizacji ciągłej sieci Hopfielda jako ciągłą i znormalizowaną regułę aktualizacji klasycznej sieci Hopfielda.

Z czego wynika wykorzystanie znormalizowanej funkcji eksponencjalnej? Równanie 4.3 można zapisać jako:

$$E(\xi) = E_1(\xi) + E_2(\xi) \tag{4.12}$$

$$E_1(\xi) = \frac{1}{2}\xi^T\xi + \beta^{-1}ln(S) + \frac{1}{2}M^2 = \frac{1}{2}\xi^T\xi + C_1 \tag{4.13}$$

$$E_2(\xi) = -lse(\beta, X^T\xi) \tag{4.14}$$

E_1 jest funkcją wypukłą a E_2 jest funkcją wklęsłą, C_1 to stała, która nie zależy od stanu sieci. Obliczając:

$$\nabla_\xi E_1(\xi^{t+1}) = -\nabla_\xi E_2(\xi^t) \tag{4.15}$$

$$\nabla_\xi(\frac{1}{2}\xi^T\xi + C_1)(\xi^{t+1}) = \nabla_\xi lse(\beta, X^T\xi)(\xi^t) \tag{4.16}$$

uzyskujemy regułę aktualizacji stanu: [7]

$$\xi^{t+1} = Xp^t = Xsoftmax(\beta X^T\xi^t) \tag{4.17}$$

Równanie 4.15 to tak zwana procedura wklęsło-wypukła (Concave-Convex Procedure [9]). Autorzy procedury udowodnili, że ewolucje funkcji opisujących układy, dających się opisać jako suma funkcji wklęsłej i wypukłej mają gwarancję na monotoniczną minimalizację przez odpowiednio określoną regułę aktualizacji.

Co więcej, dla odpowiednio rozseparowanych wzorców reguła 4.17 gwarantuje osiągnięcie stanu odpowiadającego wzorcowi (z pewnym błędem ϵ) już po jednej aktualizacji. Definicja separacji wzorca x^i od pozostałych wzorców z macierzy X:

$$\Delta^i = min_{j,j\neq i}\frac{1}{2}(||x^i||^2 - ||x^j||^2 + ||x^i - x^j||^2) = \frac{1}{2}||x^i||^2 - min_{j,j\neq i}(\frac{1}{2}(||x^j||^2 - ||x^i - x^j||^2) \tag{4.18}$$

Innymi słowy jest to separacja wzorca x^i od najbardziej podobnego wzorca x^j. Odtworzenie wzorca z zadanym błędem ϵ można określić jako:

$$||Xsoftmax(\beta X^T\xi) - x^i|| = ||f(\xi) - x^i|| < \epsilon \tag{4.19}$$

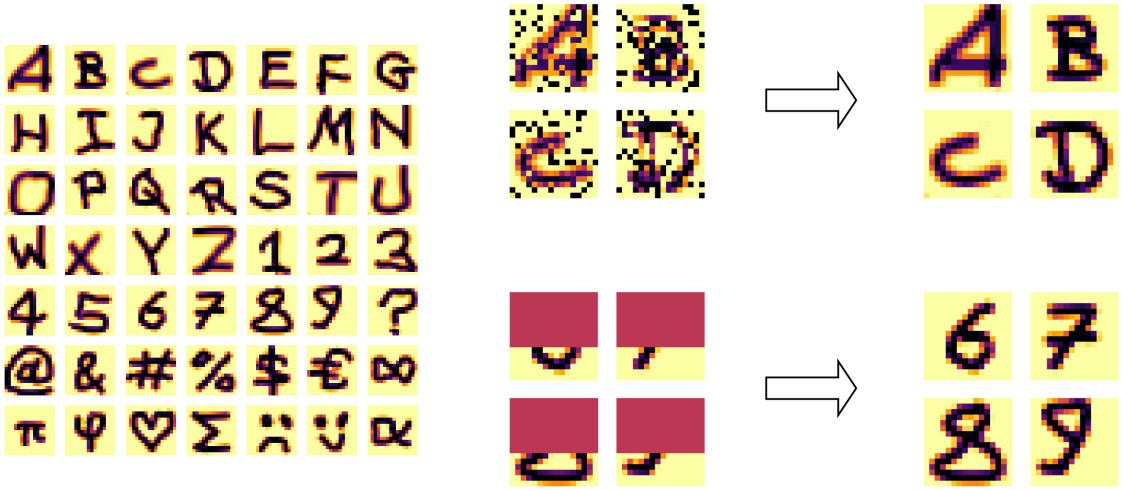
Błąd ϵ spada eksponencjalnie do zera wraz ze wzrostem separacji. [7] Podobnie jak przy gęstej pamięci asocjacyjnej pojemność nowego modelu wynosi $S_{max} = 2^{\frac{N}{2}}$, jednakże przy zwiększaniu ilości wzorców liniowo zmniejsza się średnia separacja co ostatecznie dla dużych wartości S będzie powodować zwiększanie błędu ϵ do zauważalnych wartości. Jednowymiarowy stan sieci ξ może zostać uogólniony do dwuwymiarowego stanu S jednowymiarowych "podstanów" $\Xi = (\xi^1, ..., \xi^S)$ nazywanego przestrzenią asocjacyjną. Wówczas uogólniona reguła aktualizacji przyjmuje postać:

$$\Xi^{new} = Xsoftmax(\beta X^T\Xi) \tag{4.20}$$

Ciągła sieć Hopfielda została zaprojektowane w celu integracji jej w większych strukturach uczenia głębokiego. W tym kontekście istotne jest aby jedno wywołanie funkcji aktualizacji było wystarczające do osiągnięcia pożądanego stanu $\xi = x^\mu \pm \epsilon$, gdzie ϵ to ustalona ilość dopuszczalnych błędów. Dla wzorca x^i można zdefiniować separację Δ_i jako

$$\Delta_i = min_{j\neq i}[(x^i)^Tx^i - (x^i)^Tx^j] = (x^i)^Tx^i - max_{j\neq i}[(x^i)^Tx^j] \tag{4.21}$$

Teoretycznie udowodniono, że nowa reguła aktualizacji typowo odzyskuje wzorce po jednej iteracji, oprócz tego błąd ϵ maleje eksponencjalnie wraz ze wzrostem separacji Δ_i [7].



(a) Dane treningowe (b) Rekonstrukcja danych

Rysunek 4.1: Przykład działania modelu HopfieldLayer. Nowy model pozwala na roszerzenie poprzedniego przykładu o wektory o wartościach ciągłych, zamiast jedynie tych bipolarnych (nadal tylko jeden kanał RGB). Po jednej aktualizacji stanu zarówno dane zaszumione jak i częściowe zostały w pełni odtworzone.

4.1 Ciągła sieć Hopfielda a Transformer

W pierwszej kolejności rozważmy X^T jako N surowych wektorów wzorców $Y = (y^1, ..., y^N)^T$, które są mapowane na przestrzeń asocjacyjną przez W_K i Ξ^T jako S surowych wektorów stanu $R = (\xi^1, ..., \xi^S)^T$, które są mapowane na przestrzeń asocjacyjną przez W_Q. Definiując:

$$Q = \Xi^T = RW_Q \tag{4.22}$$

$$K = X^T = YW_K \tag{4.23}$$

$$\beta = \frac{1}{\sqrt{d_k}} \tag{4.24}$$

reguła aktualizacji przyjmuje postać

$$(Q^{new})^T = K^T softmax(\frac{1}{\sqrt{d_k}} KQ^T) \tag{4.25}$$

W_Q i W_K to macierze które mapują poszczególne wzorce na przestrzeń asocjacyjną. Po przeprowadzeniu transpozycji uzyskujemy

$$Q^{new} = softmax(\frac{1}{\sqrt{d_k}} QK^T)K \tag{4.26}$$

a następnie można zmapować Q^{new} na przestrzeń asocjacyjną poprzez kolejną macierz W_V

$$Z = Q^{new}W_V = softmax(\frac{1}{\sqrt{d_k}} QK^T)KW_V \tag{4.27}$$

Powyższe równanie spełnia założenia uwagi transformera (transformer attention). W sieciach typu transformer mamy do czynienia z dzieleniem tekstu na tokeny, a następnie na wektory słów. Mimo, że Ciągła sieć Hopfielda operuje na innym typie danych to wykorzystanie podobnych mechanizmów okazuje się zasadne. Macierze W_K (key matrix), W_Q (question matrix) i W_V (value matrix) są uczone poprzez mechanizm propagacji wstecznej (backpropagation).

4.2 HopfieldLayer

W kontekście pamięci asocjacyjnej Ciagła sieć Hopfielda nie musi być nauczona żadnych wag, w takim przypadku równanie 4.27 przyjmuje postać

$$Z = softmax(\frac{1}{\sqrt{d_k}} RY^T)Y \tag{4.28}$$

i powracając do oryginalnych oznaczeń

$$\xi^{new} = softmax(\beta \xi X^T) X \tag{4.29}$$

Powyższa reguła aktualizacji, z wagami lub bez, jest podstawą architektury HopfieldLayer. "Warstwy Hopfielda" mogą zostać wykorzystane jako elementy składowe zaawansowanych architektur sztucznych sieci neuronowych, takich jak Convolutional Neural Networks i Generative Neural Networks. Dla stosunkowo łatwych do spełnienia warunków wystarcza jedna aktualizacja stanu aby HopfieldLayer zbiegło do minimum energetycznego z niewielkim błędem.

Rozdział 5

Implementacja pamięci skojarzeniowej

5.1 Przystosowanie modeli

Teoretyczne założenia zaimplementowane jako program.

5.1.1 Klasyczna sieć Hopfielda

W wykorzystywanym modelu wagi połączeń między neuronami są uczone za pomocą reguły Hebba 2.5, zaś stan neuronów będzie modyfikowany za pomocą metody asynchronicznej i równania 2.1 z $b_i = 0,\ \forall i$. Algorytm prezentuje się następująco:

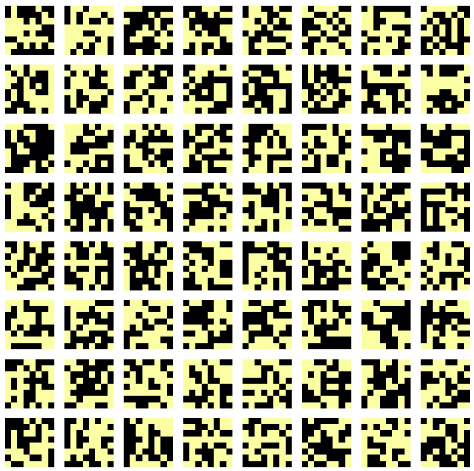
```
for i in range(max):
    new_state = copy(state)
    indexes = random_shuffle(neuron_indexes)

    for index in indexes:
        activation = dot(network.weights[index], new_state)
        new_state[index] = 1 if activation >= 0 else -1

    if state = new_state:
        return new_state

    state = new_state

return state
```

Proces jest powtarzany zadaną ilość razy lub aż zostanie osiągnięty stabilny stan sieci (miniumum energetyczne).

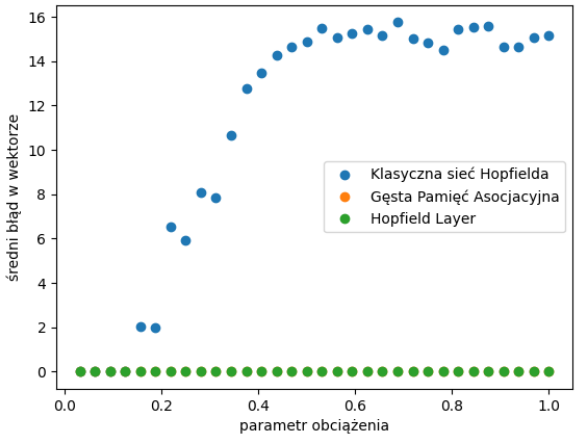
5.1.2 Gęsta pamięć asocjacyjna

Algorytm aktualizacji sieci zgodny z równaniem 3.7:

```
for i in range(max):
    new_state = copy(state)
    indexes = random_shuffle(indexes)

    for index in indexes:
        activation = 0
        for mu in range(network.patterns):
            sum_value = 0
            for j in range(length(indexes)):
                sum_value += network.patterns[mu][j] * state[j]
            sum_value -= network.patterns[mu][index] * state[index]
            activation += F(network.patterns[mu][index] + sum_value, n)
                        - F(-network.patterns[mu][index] + sum_value, n)

            new_state[index] = 1 if activation >= 0 else -1
```



(a) Dane treningowe: 64 wektory z losowo wybranymi wartościami 1 lub -1.



(b) Błędy w działaniu sieci a parametr obciążenia

Rysunek 5.1: Eksperyment porównujący pojemność klasycznej sieci Hopfielda z nowszymi modelami. (a) to graficzna prezentacja wektorów wykorzystanych jako dane treningowe, w różnych ilościach dla różnych wariantów sieci. (b) przedstawia wykres zależności średniej liczby błędnych pikseli w końcowym stanie sieci od parametru obciążenia (liczba wspomnień / liczba neuronów). W przypadku klasycznej sieci Hopfielda wyraźnie widać przeskok przy przejściu parametru obciążenia z 0.125 na 0.156, jest to zbieżne z wyznaczoną numerycznie granicą równą 0.138. Dla modelu Gęstej Pamięci Asocjacyjnej (z funkcją wielomianową o stopniu 5) i Hopfield Layer (z parametrem beta równym 5) bezbłędne odtworzenie wszystkich wzorców nie stanowiło problemu. Granica pojemności tych dwóch modeli leży dalej niż $\alpha = 1$.

```
        if state == new_state:
            return new_state

        state = new_state

return state
```

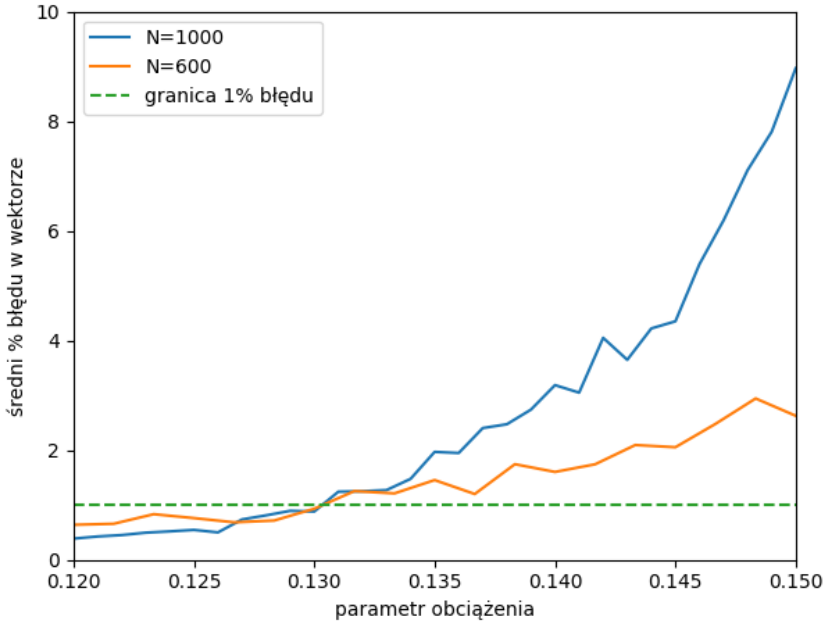
5.1.3 HopfieldLayer

Algorytm aktualizacji stanu sieci zgodnie z równaniem 4.29:

```
X_T = transpose(network.patterns)

exp_values = exp(matrix_multiplication(beta * state, X_T))
softmax_result = exp_values / sum(exp_values)

new_state = matrix_multiplication(softmax_result, network.patterns)

return new_state
```

5.2 Liczba wzorców a skuteczność modeli

Celem eksperymentu jest porównanie działania trzech omówionych modeli pamięci asocjacyjnej. W tym celu przygotowuję zestaw 64 obrazków w formacie 8 na 8 pikseli o losowych wartościach 1 lub -1, które posłużą za dane



Rysunek 5.2: Eksperymentalne wyznaczenie pojemności klasycznej sieci Hopfielda. Na wykresie wyraźnie widać jak wraz ze wzrostem liczby wspomnień dla zadanej liczby neuronów sieć traci swoje zdolności do odtwarzania. Przyjmując akceptowany błąd jako 1% błędnych pikseli graniczną wartością parametru obciążenia jest 0.131 i 0.132 co odpowiada 131 wspomnieniom w tej 1000-neuronowej sieci i 79 wspomnieniom w 600-neuronowej sieci. Nie jest to teoretyczne 0.138 ale warto wziąć pod uwagę że w eksperymencie nie mamy do czynienia z warunkami idealnymi, to znaczy początkowy stan sieci nie odpowiada wspomnieniom, a ich częściowo zaszumionym wariantom. Wraz z wyborem jak najbardziej różniących się od siebie wspomnień i zmniejszaniem liczby wyzerowanych pikseli w danych testowych graniczna wartość powinna zbliżać się do tej teoretycznej. W mojej implementacji dla parametru równego 0.138 błędnie odtworzone zostało 2.5% pikseli dla N = 1000 i 1.8% dla N = 600..

treningowe. Generalnie dla każdego modelu przygotowuję przygotowuję 32 warianty sieci, różniące się liczbą zapamiętanych wzorców (2, 4, ..., 64) z zestawu widocznego na 6.4a, badane będzie działanie tych wariantów po zaprezentowaniu im wspomnień z wyzerowanymi pikselami (w liczbie 4, 8, ..., 32). Ostatecznie każdy model zostanie przetestowany na 8448 obrazkach, które są pochodnymi zestawu treningowego. Wyniki eksperymentu przedstawia 5.1b. Warto zauważyć, że kody odpowiedzialne za każdy z modeli miały różny czas działania. Działanie klasycznej sieci Hopfielda to 17s 1ms, dla gęstej pamięci asocjacyjnej było to 17min 35s 918ms, a dla Hopfield Layer 6s 452ms. Duże odchylenie dla drugiego modelu wynika z jego asynchronicznej natury, przy aktualizacji każdego pojedynczego piksela model musi przetworzyć dużą ilość danych. Biorąc pod uwagę postać równań opisujących trzeci model najkrótszy czas działania może być zaskakujący. Przyczyną jest to, że wszystkie jednostki aktualizują się synchronicznie poprzez operacje na macierzach, dodatkowo aktualizacja następuje tylko raz.

5.3 Eksperymentalne wyznaczenie pojemności klasycznej sieci Hopfielda

W pierwszym eksperymencie tego rozdziału udało mi się oszacować, że graniczna wartość parametru obciążenia leży w przedziale od 0.125 do 0.156. W tym eksperymencie spróbuję dokładniej wyznaczyć ten przedział ale bez podejścia numerycznego. Przygotowuję zestaw danych treningowych tak, aby zbadać przedział parametru obciążenia między 0.12 a 0.15. Dla N-neuronowej klasycznej sieci Hopfielda potrzebne będzie 0.15N wektorów o N losowych wartościach z pary {-1,1}, które posłużą za dane treningowe. Dodatkowo danymi testowymi będą dwie kopie danych treningowych, w pierwszej wyzerowane 0.05N pikseli, a w drugiej 0.1N.

Rozdział 6

Pamięć skojarzeniowa a problem klasyfikacji

Baza danych MNIST to obszerna baza danych ręcznie rysowanych cyfr, zawiera kilkadziesiąt tysięcy przykładów w formacie 28 na 28 pikseli. Zestaw jest często wykorzystywanych w zagadnieniach uczenia maszynowego, konkretnie w problemie klasyfikacji cyfr z zestawu testowego.

Dotychczas pamięć skojarzeniowa była wykorzystywana w kontekście odtwarzania wspomnień, gdzie model starał się rekonstruować uszkodzone lub niepełne dane na podstawie dostępnych informacji. Jednakże, poprzez odpowiednie modyfikacje, możliwe jest wykorzystanie tych modeli do nowych zadań, takich jak klasyfikacja. W przypadku konstrukcji modelu Hopfield Layer do problemu klasyfikacji cyfr, najważniejszym krokiem jest stworzenie 10 obrazków, które będą jak najbardziej reprezentatywne dla każdej z cyfr. W ten sposób, teoretycznie, model powinien zbiegać do odpowiedniego wzorca klasyfikacyjnego dla danej cyfry wejściowej.

6.1 Proste algorytmy konstrukcji wzorców klasyfikacyjnych

W pierwszej kolejności zaprezentuję wyniki wykorzystania trzech prostych algorytmów do konstrukcji wzorców klasyfikacyjnych. Baza danych MNIST została podzielona na 75% danych treningowych i 25% danych testowych.

6.1.1 Średnia arytmetyczna

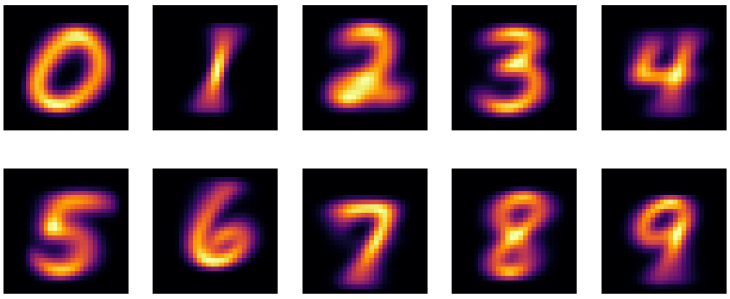
Pierwszy eksperyment zakłada wygenerowanie wzorców klasyfikacyjnych w formie średniej arytmetycznej. Będzie to średnia z wartości odpowiadających pikseli w całym zestawie treningowym. Tak skonstruowany zestaw wzorców pozwolił modelowi HopfieldLayer poprawnie zaklasyfikować 58.4% cyfr z zestawu testowego, szczegóły na grafice 6.2.

6.1.2 Litera "matka"

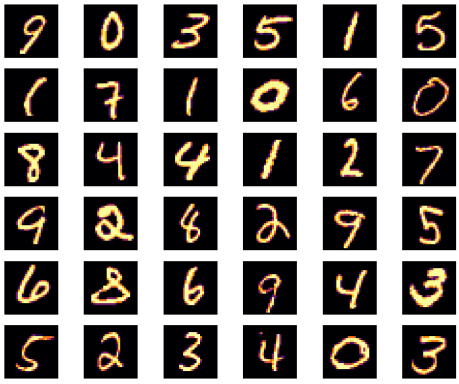
Założeniem tego eksperymentu jest znalezienie pojedynczego zestawu cyfr, które są najbardziej podobne do wszystkich pozostałych. Miarą podobieństwa będzie skuteczność klasyfikacji zestawu testowego w modelu HopfieldLayer. Przebieg iteracji "poszukiwania":

- Program losuje 2000 cyfr z zestawu danych testowych.
- Model przyjmuje każdy dostępny zestaw z danych treningowych jako wzorce i oblicza skuteczność klasyfikacji zestawu z punktu pierwszego.
- Do kolejnej iteracji przechodzą zestawy danych treningowych, które przekroczyły zadany próg skuteczności.

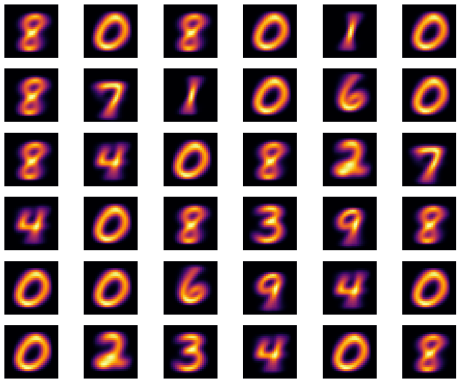
Początkowo akceptowany próg skuteczności wynosi 30%, i w każdej kolejnej iteracji jest zwiększany o 1 punkt procentowy. Całą koncepcję "iteracji poszukiwań" przyjmuję ze względu na losowość, pojedyncze przyjęcie wysokiego progu mogłoby spowodować, że zadowalająca skuteczność wynikałaby ze szczęśliwego wylosowania danych testowych z pełnego zestawu. Przebieg poszukiwania:



(a) Skonstruowane wzorce klasyfikacyjne



(b) Część danych wejściowych



(c) Część danych wyjściowych

Rysunek 6.1: (a) przedstawia zestaw wzorców klasyfikacyjnych wygenerowanych jako średnia arytmetyczna danych treningowych, czyli ponad 40 tysięcy obrazków cyfr. Model HopfieldLayer z takimi wzorcami był w stanie poprawnie zaklasyfikować 58.4% cyfr z zestawu treningowego. Na (b) oraz (c) widoczny jest wynik dla 36 przykładowych cyfr z zestawu testowego, w tym konkretnym przypadku poprawnie odtworzonych zostało 20 z nich.

iteracja	próg skuteczności	liczba kandydatów
1	30%	1932
2	31%	1712
3	32%	1591
4	33%	1264
5	34%	1064
6	35%	944
7	36%	720
8	37%	545
9	38%	373
10	39%	271
11	40%	186
12	41%	147
13	42%	122
14	43%	82
15	44%	53
16	45%	29
17	46%	23
18	47%	13
19	48%	12
20	49%	6
21	50%	3

Gdzie liczba kandydatów to liczba zestawów wzorców cyfr które spełniły zakładany próg i przeszły do kolejnej iteracji. Po 21 iteracjach pozostały tylko 3 zestawy, ostatecznie przejdą one test skuteczności klasyfikacji wszystkich danych treningowych i testowych.

indeks zestawu	testowe	treningowe
1	49.18%	48.96%
2	50.52%	50.15%
3	49.01%	49.30%

Ostatecznie najlepszy "ręcznie" wybrany zestaw wzorców poprawnie dopasował nieco ponad połowę zarówno cyfr z danych testowych jak i treningowych. Graficzna postać zestawu:

6.2 Gradientowa konstrukcja wzorca klasyfikacyjnego

Metoda gradientu prostego to algorytm numeryczny służacy do poszukiwania lokalnego minimum zadanej funkcji celu, powszechnie wykorzystywane do optymalizacji sztucznych sieci neuronowych. Algorytm wykorzystuje gradient liczony ze względu na argumenty funkcji celu, rozmiar kroków (prędkość zbiegania do minimum) jest modyfikowana przez współczynnik uczenia η (learning rate). [8]

Aby wykorzystać metodę gradientu prostego do klasyfikowania cyfr definiuję funkcję celu, jej minimalizacja oznacza odnalezienie zestawu 784 pikseli najbardziej podobnych do wszystkich cyfr z zestawu treningowego.

$$J(\theta) = \sum_{przykady} \sum_{piksele} (piksel\, przykadu - piksel\, wzorca)^2 \tag{6.1}$$

$$\nabla_\theta J(\theta) = \sum_{przykady} \sum_{piksele} 2(piksel\, przykadu - piksel\, wzorca) \tag{6.2}$$

Założeniem eksperymentu jest skonstruowanie 10 wzorców klasyfikacyjnych na podstawie minimalizacji funkcji celu. Następnie takie wzorce wykorzystam w modelu HopfieldLayer.

6.2.1 Gradient prosty "vanilla"

Gradient jest obliczany dla wszystkich parametrów, aby przeprowadzić jedną aktualizację argumentów θ wykorzystywany jest cały zestaw treningowy.

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{6.3}$$

Minimalizacja jest realizowana przez prosty kod:

```
target_image = random_integer(0, 128, size=784)

for _ in range(iterations):
    gradient = zeros_like(target_image)

    for image in training_images:
        gradient += 2 * (array(image) - target_image)

    target_image = clip(target_image - learning_rate * gradient, 0, 255)
```

Obrazki MNIST dzielę na zestaw treningowy (75%) i zestaw testowy (25%). Początkowo wzorzec klasyfikacyjny to losowy szum o wartościach od 0 do 128, później w każdej iteracji wzorzec jest aktualizowany zgodnie z powyższym algorytmem w oparciu o zestaw treningowy. Taki proces odbywa się dla wszystkich 10 cyfr, konstruując w ten sposób 10 wzorców klasyfikacyjnych. Istotnym krokiem jest odpowiedni dobór parametrów uczenia tj. współczynnik uczenia i liczba epok. Wyniki działania przedstawia 6.2

6.2.2 Gradient prosty "Mini-batch"

Modyfikacja algorytmu zakłada podzielenie zestawu treningowego na mniejsze części tak, aby przyspieszyć działanie. W każdej epoce wybierane zostaje 10% dostępnych przykładów i to na ich podstawie obliczana jest aktualizacja wartości argumentów θ.

```
target_image = random_integer(0, 128, size=784)

for i in range(iterations):
    subset_images = random_choice(images, size=subset_size, replace=False)
    gradient = zeros_like(target_image)

    for image in subset_images:
        gradient += 2 * (array(image) - target_image)

    target_image = clip(target_image - learning_rate * gradient, 0, 255)
```

Przyjmuję leraning rate $5e-6$ i badam jak zmienia się wartość minimalizowanej funkcji celu w zależności od epok uczenia algorytmu. Wyniki działania przedstawia 6.3

6.2.3 Adagrad

Algorytm Adagrad dostosowuje współczynnik uczenia η do zmian wartości parametrów, poprzednio współczynnik był stały dla każdego argumentu. Wprowadzam gradient w stosunku do argumentu θ_i w punkcie czasu t:

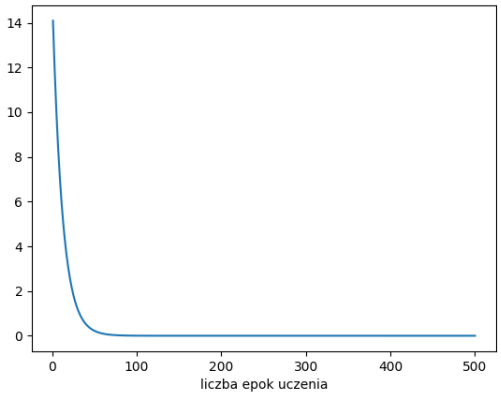
$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i}) \tag{6.4}$$

Przy takim oznaczeniu algorytm vanilla może zostać zapisana jako:

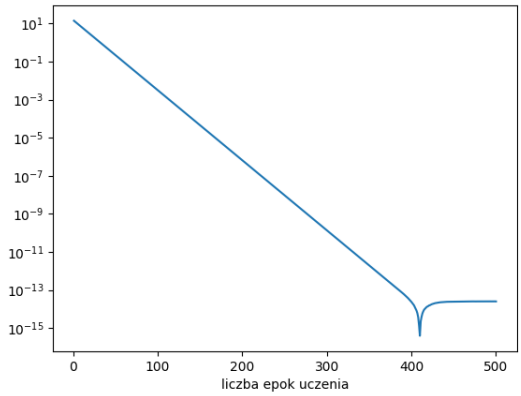
$$\theta_{t+1,i} = \theta_{t,i} - \eta g_{t,i} \tag{6.5}$$

Adagrad modyfikuje to równanie do postaci:

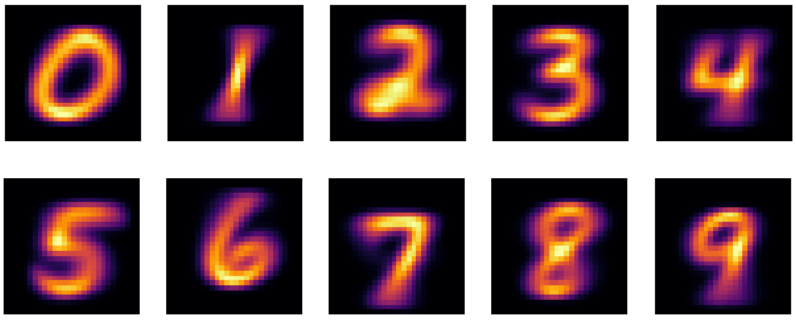
$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}} + \epsilon} g_{t,i} \tag{6.6}$$



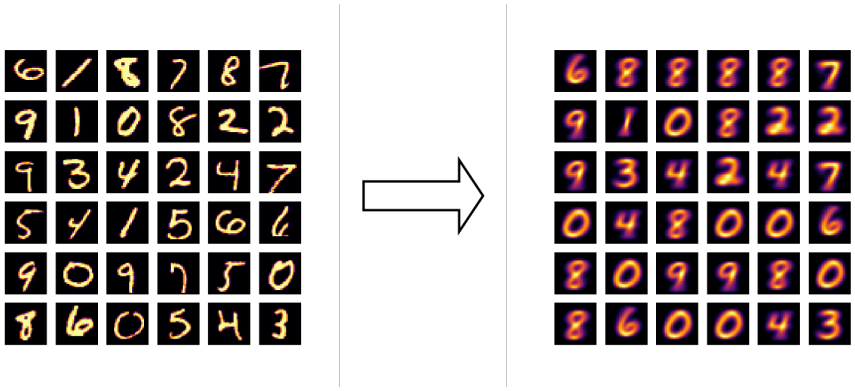
(a) Przebieg uczenia A



(b) Przebieg uczenia B

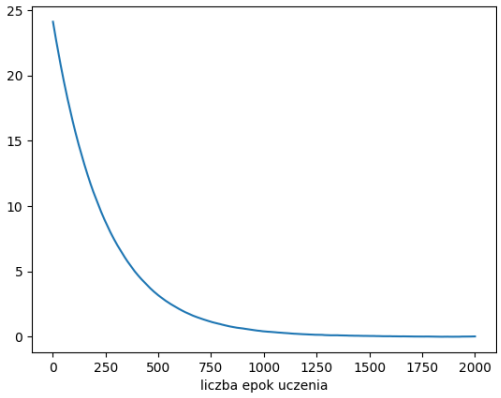


(c) Skonstruowane wzorce klasyfikacyjne

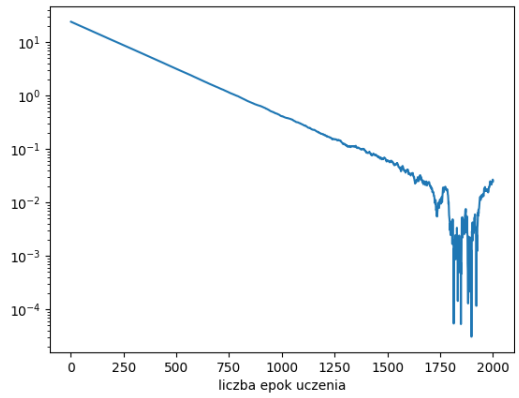


(d) Przykładowe wyniki

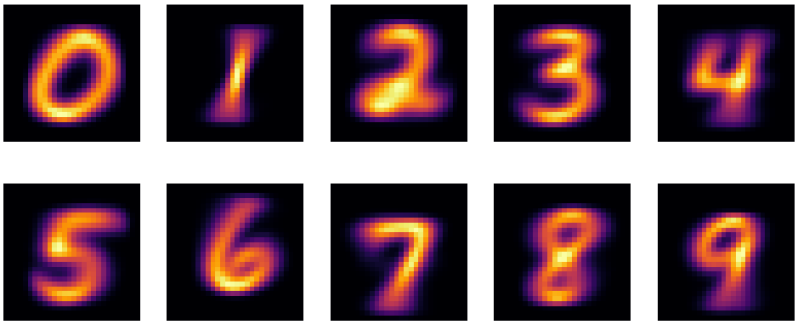
Rysunek 6.2: Wykresy (a) oraz (b) prezentują proces konstrukcji wzorca klasyfikacyjnego dla cyfry 0, wykorzystując gradient prosty. Oś x przedstawia liczbę kolejnych aktualizacji stanu wzorca na podstawie obliczonego gradientu, natomiast oś y to znormalizowana wartość funkcji celu (pierwiastek kwadratowy wartości per piksel). Na skali logarytmicznej zauważalne jest wyraźne załamanie algorytmu w pewnym momencie, gdzie dalsze aktualizacje nie przynoszą dalszego zmniejszenia wartości funkcji. Załamanie to występuje w różnym momencie dla każdej z cyfr (od 408 do 427 epoki). W dalszych testach uwzględniono każdy wzorzec klasyfikacyjny, dla którego wartość funkcji przestała spadać, co zostało przedstawione na wykresie (c). Dla pełnego zestawu testowego model osiągnął 58,39% poprawnych klasyfikacji, co częściowo ilustruje wykres (d) zawierający 36 wylosowanych przykładów przebiegu klasyfikacji.



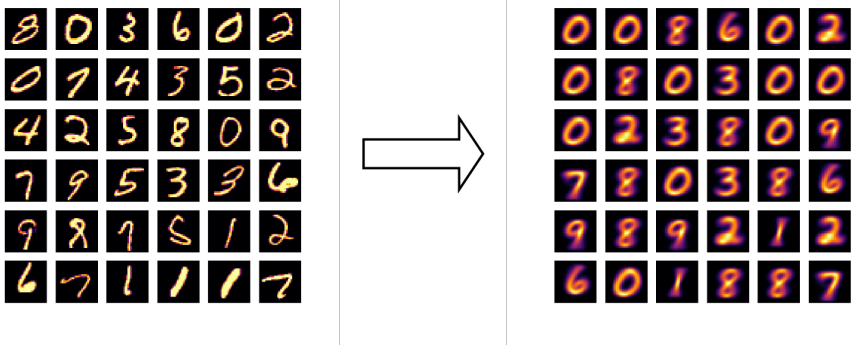
(a) Przebieg uczenia A



(b) Przebieg uczenia B



(c) Skonstruowane wzorce klasyfikacyjne



(d) Przykładowe wyniki

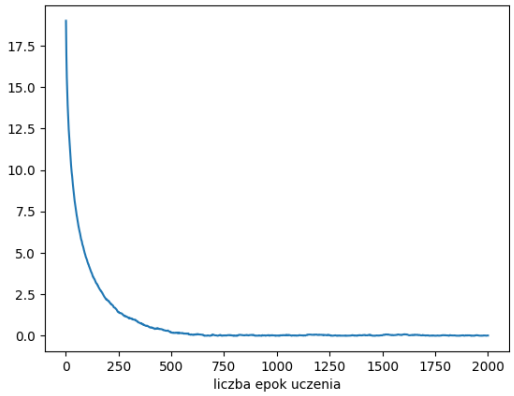
Rysunek 6.3: Wykresy (a) oraz (b) prezentują proces konstrukcji wzorca klasyfikacyjnego dla cyfry 4, wykorzystując gradient prosty "Mini-batch". W tym przypadku spadek wartości funkcji celu następuje wolnej ale podobnie jak w poprzednim przypadku widoczne jest załamanie algorytmu, od 1620 do 1966 epoki dla różnych cyfr. Wzorce klasyfikacyjne, dla którego wartość funkcji przestała spadać ilustruje (c). Dla pełnego zestawu testowego model osiągnął 58,46% poprawnych klasyfikacji, co częściowo ilustruje wykres (d) zawierający 36 wylosowanych przykładów przebiegu klasyfikacji.

Macierz G jest obliczana jako suma kwadratów gradientów, argument ϵ pozwala na uniknięcie dzielenia przez 0. Równanie w postaci macierzowej:

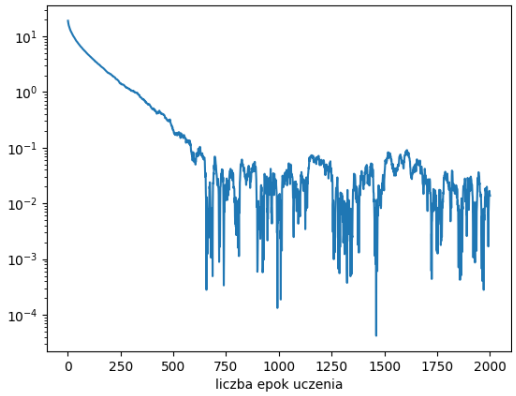
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t} + \epsilon} g_t \tag{6.7}$$

```
target_image = np.random.randint(0, 128, size=784)

for i in range(iterations):
    subset_images = random_choice(images, size=subset_size, replace=False)
    gradient = zeros_like(target_image)

    for image in subset_images:
        gradient += 2 * (array(image) - target_image)

    G = gradient^2
    target_image = target_image + (learning_rate / (sqrt(G) + epsilon)) * gradient
    target_image = clip(target_image, 0, 255)
```

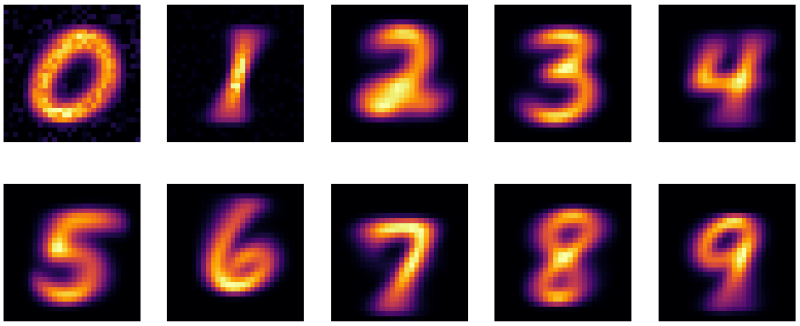
Wyniki działania przedstawia 6.4



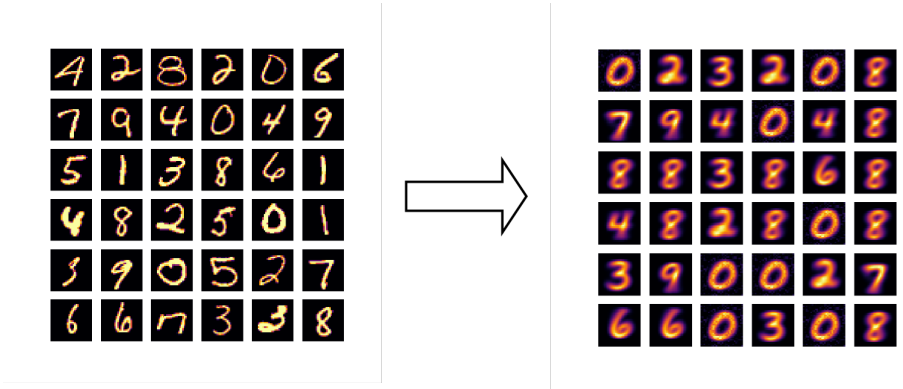
(a) Przebieg uczenia A



(b) Przebieg uczenia B



(c) Skonstruowane wzorce klasyfikacyjne



(d) Przykładowe wyniki

Rysunek 6.4: Wykresy (a) oraz (b) prezentują proces konstrukcji wzorca klasyfikacyjnego dla cyfry 7, wykorzystując gradient Adagrad. Spadek wartości funkcji celu przebiega szybciej niż przy mini-batch, załamanie algorytmu następuje między 311 a 1948 epoką dla różnych cyfr. Wzorce klasyfikacyjne, dla którego wartość funkcji przestała spadać ilustruje (c). Dla pełnego zestawu testowego model osiągnął 59,24% poprawnych klasyfikacji, co częściowo ilustruje wykres (d) zawierający 36 wylosowanych przykładów przebiegu klasyfikacji.

Rozdział 7

Pamięć skojarzeniowa i dane eksperymentalne

W poprzednich implementacjach stosowałem sieci Hopfielda tylko w roli modeli pamięci asocjacyjnej operującej na grafice. Jako że sieć operuje na wektorach danych, które jedynie dla prezentacji przekształcam na obrazek, wektor nie musi koniecznie reprezentować grafiki. Mechanizm osiągania minimum energetycznego jest aktualny dopóki format danych jest poprawny.

W następnych eksperymentach wykorzystam model HopfieldLayer do operowania na danych eksperymentalnych. Celem jest aby po zaprezentowaniu danych sieci potrafiła ona rozpoznać co one prezentują i dopasować postać przewidywaną przez teoretyczne równanie.

7.1 Rozpoznanie równania ruchu

W pierwszej implementacji generuję trzy zestawy danych teoretycznych, będą to wartości przewidziane przez teoretyczne równania w 100 momentach czasu, co 0.1s:

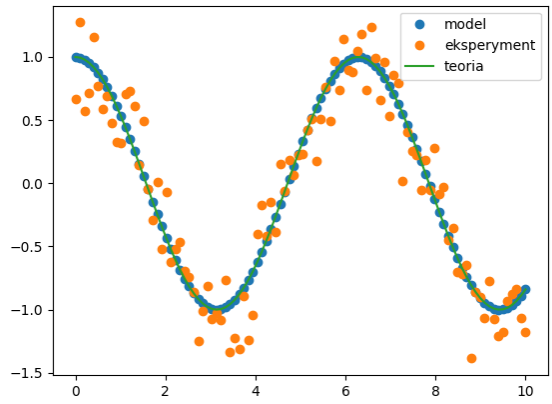
- Oscylator harmoniczny $x(t) = Acos(\omega t)$, $(A = 1, \omega = 1)$
- Oscylator tłumiony $x(t) = exp(-\frac{C}{2m}t)[cos(\sqrt{\frac{k}{m}}t) + sin(\sqrt{\frac{k}{m}}t)\frac{C}{2m}\sqrt{\frac{m}{k}}]$, $(m = 1, C = 0.5, k = 2)$
- Rzut pionowy w górę $x(t) = v_0 t - at^2$, $(v_0 = 1, a = 0.1)$

Takie dane teoretyczne zadziałają jak dane treningowe (wzorce), zostaną one zaprezentowane modelowi. Aby przetestować skuteczność modelu symuluję dane eksperymentalne zaszumiając dane teoretyczne zgodnie z rozkładem Gaussa (numpy.random.normal). Symuluję sytuację, w której mamy do czynienia z zestawem danych eksperymentalnych, ale nie wiadomo z pomiarów jakiej trajektorii one pochodzą, zadanie rozpoznania należy do modelu. Model jest nauczony trzech konkretnych przykładów, otrzymuje dane eksperymentalne i po jednej aktualizacji zwraca dopasowane dane, jest to jeden z zapamiętanych wzorców. Na wykresie znajdują się dane zwrócone przez model, dane "eksperymentalne" oraz postać przewidywana przez teorię.

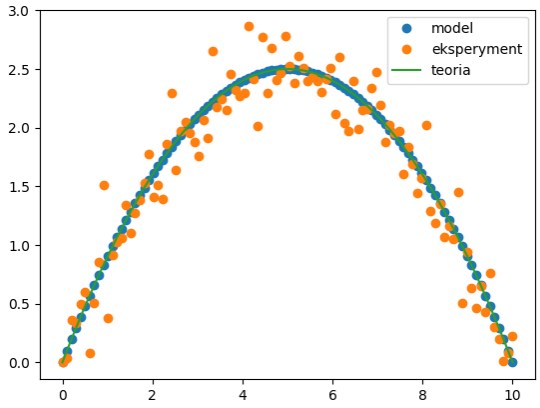
7.2 Rozpoznawanie wartości argumentów

Problemem powyższej implementacji jest przywiązanie do jednego konkretnego przypadku z każdego uczonego prawa fizycznego. W drugiej implementacji jako dane treningowe wykorzystam różne przypadki oscylatora harmonicznego, dla 50 różnych wartości argumentu ω od 0 do 5, w ten sposób uzyskuję 50 wzorców. Symuluję sytuację w której wiemy z pomiaru jakiego zjawiska fizycznego pochodzą dane, ale poszukujemy wartości parametrów, całość przedstawia 7.2a. Mimo wszystko proces jest dość życzeniowy, wybieramy stały przedział czasowy i stałe wartości wszystkich pozostałych argumentów. Dodatkowo jeżeli chcielibyśmy badać większy przedział badanego argumentu lub uzyskać dokładniejszy wynik, za dane treningowe musiałby służyć duży zestaw teoretycznych rozwiązań. Model sam w sobie nie przewiduje żadnych przesunięć.

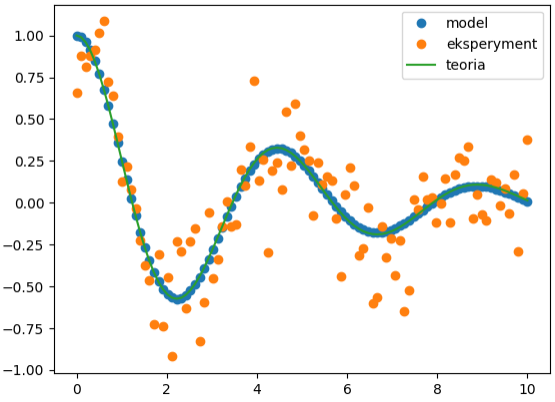
Co jeżeli dane wejściowe nie mają żadnego związku z nauczonymi wzorcami? W trzeciej implementacji wykorzystuję ten sam zestaw treningowy co w poprzednim teście, różnica jest taka, że tym razem dane treningowe to losowe liczby całkowite z zakresu od -3 do 3. Tak jak pokazuje 7.2b i 7.2c model "na siłę" dopasowuje dane do jednego z wzorców nawet jeśli są to dane czysto losowe. Największa zaleta, czyli szybkie zbieganie do minimum



(a) Dopasowanie trajektorii oscylatora harmonicznego



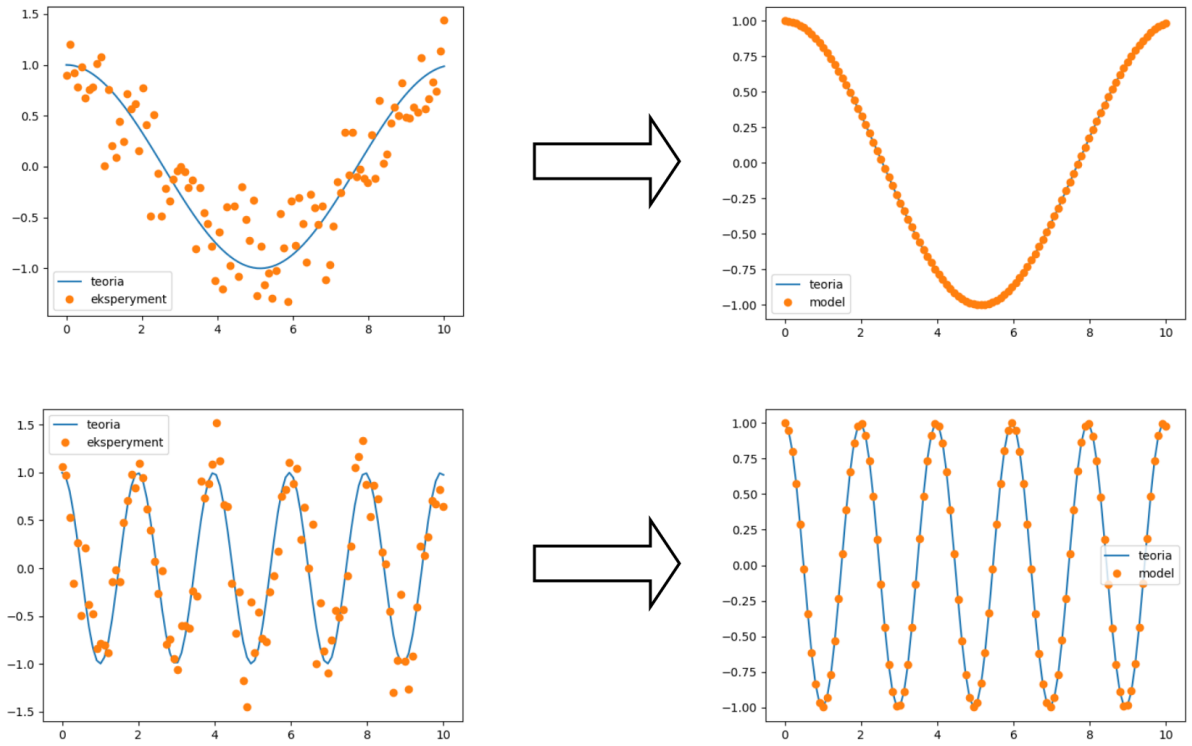
(b) Dopasowanie trajektorii rzutu ukośnego



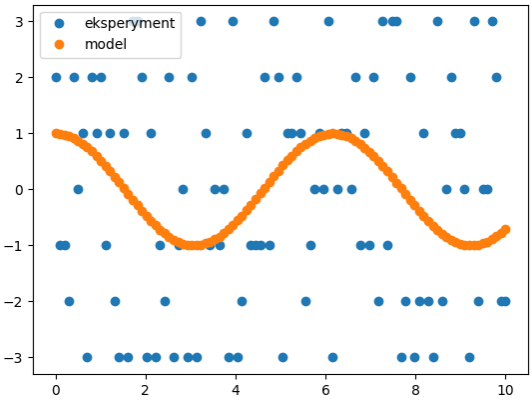
(c) Dopasowanie trajektorii oscylatora harmonicznego

Rysunek 7.1: Powyższe wykresy przedstawiają przebieg dopasowania danych eksperymentalnych przez model HopfieldLayer. Już po jednej aktualizacji minimum atraktor został osiągnięty w każdym z tzech przypadków, dane zwrócone przez model pokrywają się z postacią przewidzianą w teoretycznym równaniu.

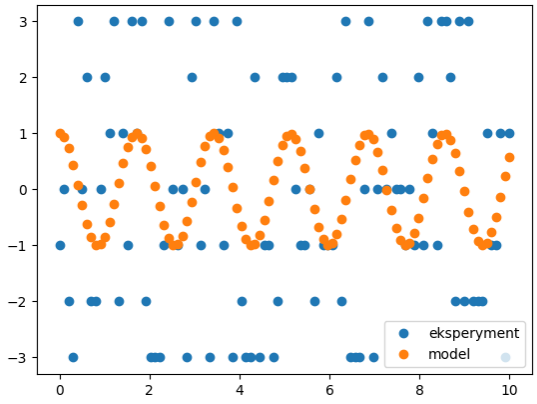
energetycznego, w tym przypadku okazała się największym źródłem błędu. Model HopfieldLayer nie jest w stanie rozpoznać czy dane testowe mają związek z danymi treningowymi, po prostu zwróci te zapamiętane dane, które są najbardziej podobne.



(a) Przebieg dopasowania



(b) Przebieg dopasowania



(c) Przebieg dopasowania

Rysunek 7.2: Dwa wyniki działania tego samego modelu nauczonego 50 przebiegami oscylatora harmonicznego. Mimo, że dane wejściowe to losowe liczby całkowite to model dopasował je do jednego ze wspomnień; w przypadku (a) jest to $\omega = 1.9$ a w (b) jest to $\omega = 4.9$. Mimo braku bliższego związku z danymi treningowymi, model HopfieldLayer zawsze dopasuje "jakiś" wzorzec.

Rozdział 8

Analiza wyników

Klasyczna sieć Hopfielda jest interesującym modelem z punktu widzenia historii rozwoju sztucznych sieci neuronowych oraz jej powiązań z fizyką statystyczną. Jeżeli chodzi o współczesne wykorzystanie implementacji tego modelu, jest on szybki w działaniu ale jego skuteczność jest wątpliwa. Eksperyment przeprowadzony w celu porównania pojemności modeli pamięci asocjacyjnej wykazały różnice w skuteczności odwzorowania wzorców w zależności od liczby zapamiętanych wzorców oraz obciążenia. Wyniki sugerują, że modele oparte na Gęstej Pamięci Asocjacyjnej i Hopfield Layer mogą utrzymać większą pojemność niż klasyczna sieć Hopfielda. Ta ostatnia wyraźnie odstaje od porównywanych modeli już przy prostym zadaniu. Przystosowanie modelu do bardziej złożonych zadań wymagałoby mocnego zwiększenia liczby neuronów co oczywiście wydłużyłoby czas działania i zwiększyło ilość wykorzystywanej pamięci do przechowywania macierzy wag połączeń między neuronami.

Przy Gęstej Pamięci Asocjacyjnej widać wyraźny postęp jeżeli chodzi o skuteczność w modelu opartym na tej samej koncepcji minimalizacji energii. Największym problemem tego modelu jest jego asynchroniczna natura, która sprawia, że działa on wyraźnie wolniej od dwóch pozostałych.

Model HopfieldLayer oparty o Ciągłą sieć Hopfielda wydaje się rozwiązywać problemy starszych implementacji pamięci asocjacyjnej. Działa on najszybciej, z dużym prawdopodobieństwem osiąga minimum energetyczne po jednej aktualizacji stanu i przetwarza różne typy danych, także tych w postaci ciągłej a nie tylko bipolarnej. Najlepiej widać to przy rozwiązaniu problemu klasyfikacji z rozdziału 4 oraz danych eksperymentalnych z rozdziału 5. HopfieldLayer otworzył wiele możliwości na eksperymenty, których implementacja była prosta z programistycznego punktu widzenia. W mojej opinii praca na tym modelu jest najbardziej przyjemna, łatwiej zaimplementować rozwiązanie swojego problemu i przewidzieć wyniki.

Badania wykorzystujące bazę danych MNIST miały na celu zastosowanie modeli pamięci asocjacyjnej do problemu klasyfikacji cyfr. Wyniki sugerują, że modyfikacja algorytmu konstrukcji wzorców klasyfikacyjnych może wpłynąć na skuteczność klasyfikacji, ale mimo wszystko najlepsza uzyskana skuteczność wynosi 59%. Jest to niewątpliwie lepszy wynik niż "rzut monetą" ale odbiega on od rozwiązań innymi metodami uczenia maszynowego (np 99.8% dla konwolucyjnej sieci neuronowej [1]). Sugeruje to, że pamięć asocjacyjna nie jest w stanie rozwiązać tego typu problemu bez bardziej zaawansowanych technik pre-processingu danych.

Eksperymenty przeprowadzone w rozdziale 5. pokazują, że model HopfieldLayer może być skutecznie wykorzystany do rozpoznawania trajektorii ruchu na podstawie danych eksperymentalnych oraz do identyfikacji różnych wartości parametrów na podstawie danych treningowych. Jednakże model ten może być podatny na błędne dopasowania, szczególnie gdy dane testowe nie mają bezpośredniego związku z danymi treningowymi.

Rozdział 9

Podsumowanie

model	teoretyczna pojemność S_{max}	złożoność pamięciowa
klasyczna sieć Hopfielda	$0.138N$	$N^2 + N$, typ int
gęsta pamięć asocjacyjna	$2^{N/2}$	$N(S+1)$, typ int
ciągła sieć Hopfielda	$2^{N/2}$	$N(S+1)$, typ float

Bibliografia

[1] Sanghyeon An, Minjun Lee, Sanglee Park, Heerin Yang, and Jungmin So. An ensemble of simple convolutional neural network models for mnist digit recognition, 2020.

[2] Mete Demircigil, Judith Heusel, Matthias Löwe, Sven Upgang, and Franck Vermet. On a model of associative memory with huge storage capacity. *Journal of Statistical Physics*, 168(2):288–299, May 2017.

[3] John Hertz, Anders Krogh, and Richard G. Palmer. Iintroduction to the theory of neural computation, 1991.

[4] John Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79:2554–8, 05 1982.

[5] Dmitry Krotov and John J Hopfield. Dense associative memory for pattern recognition, 2016.

[6] David J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. nvm, 2003.

[7] Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, Victor Greiff, David Kreil, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield networks is all you need, 2021.

[8] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[9] Alan L Yuille and Anand Rangarajan. The concave-convex procedure (cccp). In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

Rozdział 10

Poza konkursem

Kilka rzeczy, które ruszałem w ramach pracy nad tekstem, istotność raczej wątpliwa.

10.0.1 Ciągła klasyczna sieć Hopfielda

Zmiana funkcji aktywacyjnej

W poprzednich przykładach posługiwałem się sieciami w których funkcja aktywacji przyjmowała formę funkcji progowej zwracającej tylko i wyłącznie wartości ze zbioru {-1, 1}. Aby uogólnić model do wartości ciągłych, można postawić na funkcję postaci tangensa hiperbolicznego.

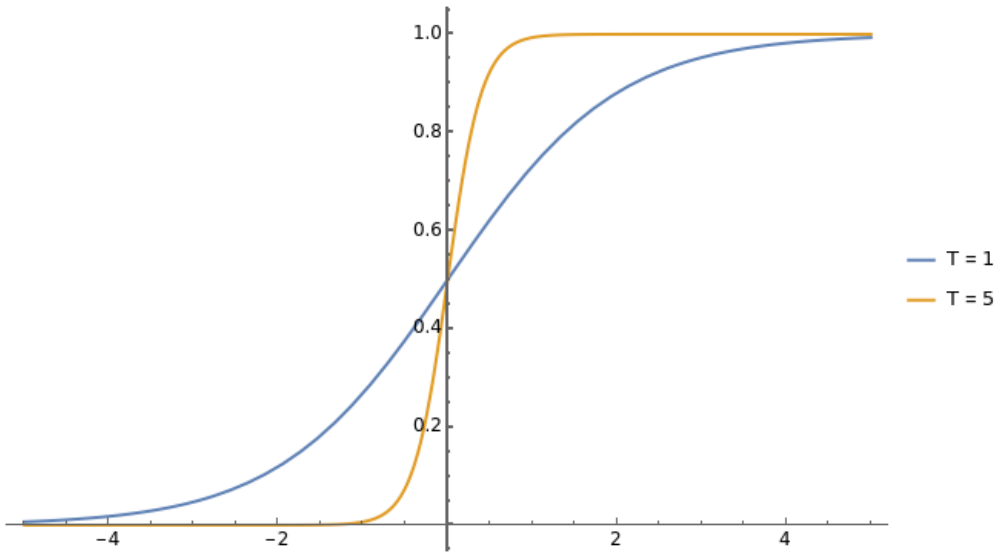
$$\xi_i = tanh[T \sum_{j}^{N} (W_{ij}\xi_j + b_i)] \tag{10.1}$$

Argument T wprowadzam aby móc modyfikować kształt funkcji. Dzięki wykorzystaniu ciągłej funkcji jesteśmy w stanie odczytać więcej informacji na temat wartości przekazywanych w aktywacji neuronu. Konkretny przykład - dwie sieci Hopfielda wykorzystujące tangens hiperboliczny (T = 8) jako funkcję aktywacji. Pierwsza sieć jest uczona metodą gradientową indywidualnie iterując każdy wzorzec 50-krotnie (learning rate: R - 0.9, O - 0.2, X - 0.5, Z - 0.04, T - 0.03, A - 0.005), w drugiej sieci podczas jednej iteracji każdy wzorzec jest wprowadzany jednokrotnie (z learning rate: R - 0.2, O - 0.1, X - 0.1, Z - 0.02, T - 0.02, A - 0.03), a cały proces powtarza się 20-krotnie. Zarówno przy danych zwróconych przez pierwszą sieć jak i drugą widać, że część neuronów ostatecznie nie osiągnęło ani wartości -1 ani 1, to bezpośredni skutek wykorzystania ciągłej funkcji. Może to oznaczać, że sieć jest "niedouczona" i należy dalej szukać odpowiednich parametrów uczenia. Dodatkowo sieć operująca na wartościach ciągłych wydaje się wymagać większej mocy obliczeniowej; dla pierwszej sieci liczba iteracji - od 1 do 17, liczba błędów - od 0 do 4, dla drugiej sieci liczba iteracji - od 1 do 54, liczba błędów od 0 do 6.

10.0.2 Proste zwiększanie pojemności sieci - neurony ukryte

Maksymalna "rozsądna" pojemność sieci Hopfielda jest odwrotnie proporcjonalna do ilości neuronów sieci, dodatkowo przy większej ilośći wzorców uczenie sieci jest trudniejsze - baseny atraktora obejmują mniejsze obszary co wymaga starannego doboru parametrów uczenia. Biorąc pod uwagę oba aspekty można ułatwić sobie pracę z siecią Hopfielda poprzez "sztuczne" rozszerzenie jej o dodatkowe neurony, których bezpośrednio nie obserwujemy. Te dodatkowe (dalej ukryte) neurony są połączone z pozostałymi w taki sam sposób, ale nie będą bezpośrednio reprezentować wzorca który wprowadzimy do sieci. Z perspektywy technicznej, wzorzec o postaci n-elementowego wektora zostanie umieszczony w k-elementowym wektorze, gdzie k-n to ilość neuronów ukrytych. Najprościej będzie ustawić wartości elementów wektora [n+1, n+2, ... , k-1, k] jako losowy wybór wartości z pary {-1,1}. Takie podejście zwiększa koszty obliczeniowe procesu uczenia ale ułatwia dobór parametrów które generują zadowalające wyniki.

Aby przetestować powyższy mechanizm implementuję 512-neuronową sieć Hopfielda, do której wprowadzę 256-pikselowe wzorce rozszerzone o 256 losowo ułożonych pikseli [Rysunek, można przedstawić to graficznie jako obrazek 16 na 32 piksele gdzie górna połowa reprezentuje widoczną warstwę. Łącznie do sieci wprowadzam 25 wzorców liter alfabetu, w pojedynczej iteracji każdy wzorzec był uczony jednokrotnie, program wykonał 20 iteracji uczenia (learning rate: A - 0.035, B - 0.015, C - 0.035, D - 0.02, E - 0.025, F - 0.035, G - 0.035, H - 0.025, I - 0.01, J - 0.025, K - 0.025, L - 0.02, M - 0.03, N - 0.025, O - 0.04, P - 0.03, Q - 0.02, R - 0.02, S - 0.03, T - 0.015, U - 0.025, W - 0.02, X - 0.025, Y - 0.02, Y - 0.02, Z - 0.03). Po wprowadzeniu danych testowych w postaci zaszumionych wzorców sieć zwracała dane w liczbie iteracji od 2 do 300 (wartość graniczna ustalona w kodzie), liczba błędnych pikseli wyniosła



Rysunek 10.1: Kształt sigmoidy zależnie od dobranego parametru T.

od 2 do 22. Warto zauważyć że wartość błędu odnosi się także do warstwy ukrytej, więc można bezpiecznie przyjąć, że błąd warstwy widocznej to 50% tych wartości tj. od 1 do 11 błędnych pikseli. Mimo wprowadzenia 25 wzorców proces doboru parametrów uczenia był wymagający w podobnym stopniu co przy 6 wzorcach w poprzedniej symulacji.

10.0.3 Zapominanie wzorców przez sieć

Badanie zachowania sieci przy podmienianiu wartości macierzy wag, sieć o 256 neuronach jest nauczona czterech wzorców (100 iteracji; learning rate: A - 0.3, E - 0.25, O - 0.2, Z - 0.2). Po standardowym przekształceniu danych wejściowych przez sieć podmieniam 500 losowo wybranych punktów macierzy wag na wartość -1 i testuję działanie sieci. Cały proces powtarzam 10-krotnie do osiągając ostatecznie 6500 podmienionych punktów, cała macierz wag zawiera $256^2 = 65536$ wartości. Jak można było się spodziewać, w miarę wzrostu liczby podmian wartości spada zdolność sieci do poprawnego ostwarzania wzorców, tracąc jakąkolwiek skuteczność przy 6000 podmian. (ewolucja sumarycznego błędu: 1, 3, 5, 8, 11, 14, 18, 22, 27, 44, 54, 66, 159, 162)

10.0.4 Maszyny Boltzmanna

a.k.a stochastyczna sieć Hopfielda Czyli sieć Hopfielda jako rozkład prawdopodobieństwa powiązany z rozkładem Boltzmanna gdzie:

$$a_i = \frac{1}{T}\sum_j w_{ij}x_j \tag{10.2}$$

$$P(x_i = 1) = \frac{1}{1 + e^{-a_i}} \tag{10.3}$$

Rozkład prawodopodobieństwa przyjęcia przez sieć danego stanu $\mathbf{x}$:

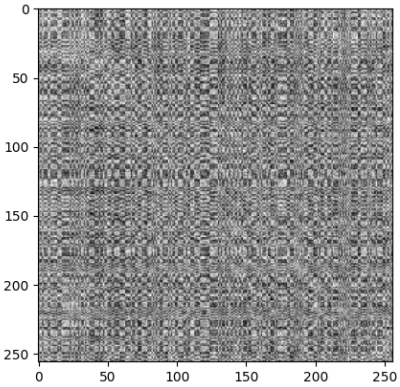
$$E(\mathbf{x}) = -\frac{1}{2}\mathbf{x}^T\mathbf{W}\mathbf{x} \tag{10.4}$$

$$P(\mathbf{x}) = C * exp(-\frac{E(\mathbf{x})}{T}) \tag{10.5}$$

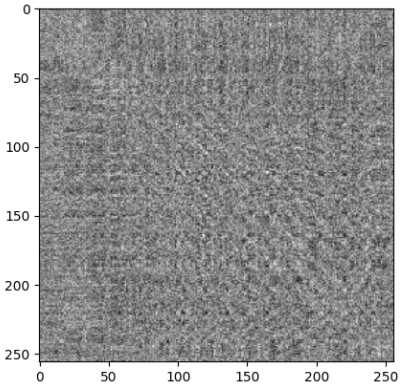
Przyjmujemy T jako parametr "pseudotemperatury" dla całej sieci. Uczenie:

$$w_{ij} = w_{ij} + \eta\frac{d(ln(P(\mathbf{x}_p)))}{dw_{ij}} \tag{10.6}$$

Do sieci Hopfielda o N neuronach dodajemy "ukrytą warstwę" o np. K neuronach (np. K = 10*N). Podczas uczenia wzorca blokujemy "widoczną warstwę" w stanie odpowiadającym wzorcowi i rozpoczynamy ewolucję sieci (w całości widoczne + ukryte). Pewnym uproszczeniem struktury jest Ograniczona(?) Maszyna Boltzmanna (Restricted Boltzmann Machine) gdzie każdy neuron z warstwy widocznej jest połączony z każdym neuronem z warstwy ukrytej i na odwrót, ale nie łączymy między sobą neuronów warstwy widocznej i nie łączymy między sobą neuronów warsztwy ukrytej. Deep Boltzmann Machines rozszesza ideę, istnieją Deep Boltzmann Machines i Helmholtz Machine

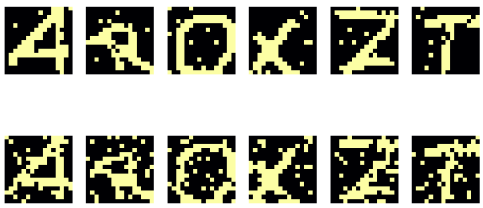


(a) Macierz wag pierwszej sieci

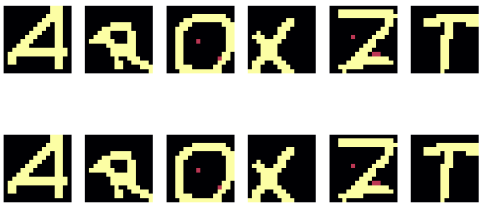


(b) Macierz wag drugiej sieci

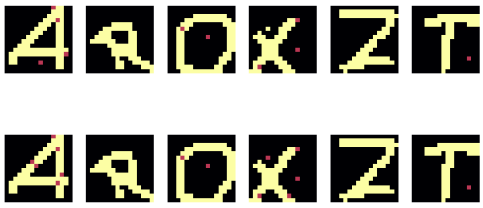
(c) Wzorce i wzorce odwrotne



(d) Dane wejściowe

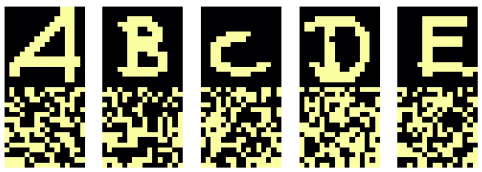


(e) Dane wyjściowe pierwszej sieci



(f) Dane wyjściowe drugiej sieci

Rysunek 10.2: Wyniki sieci Hopfielda z tangensem hiperbolicznym



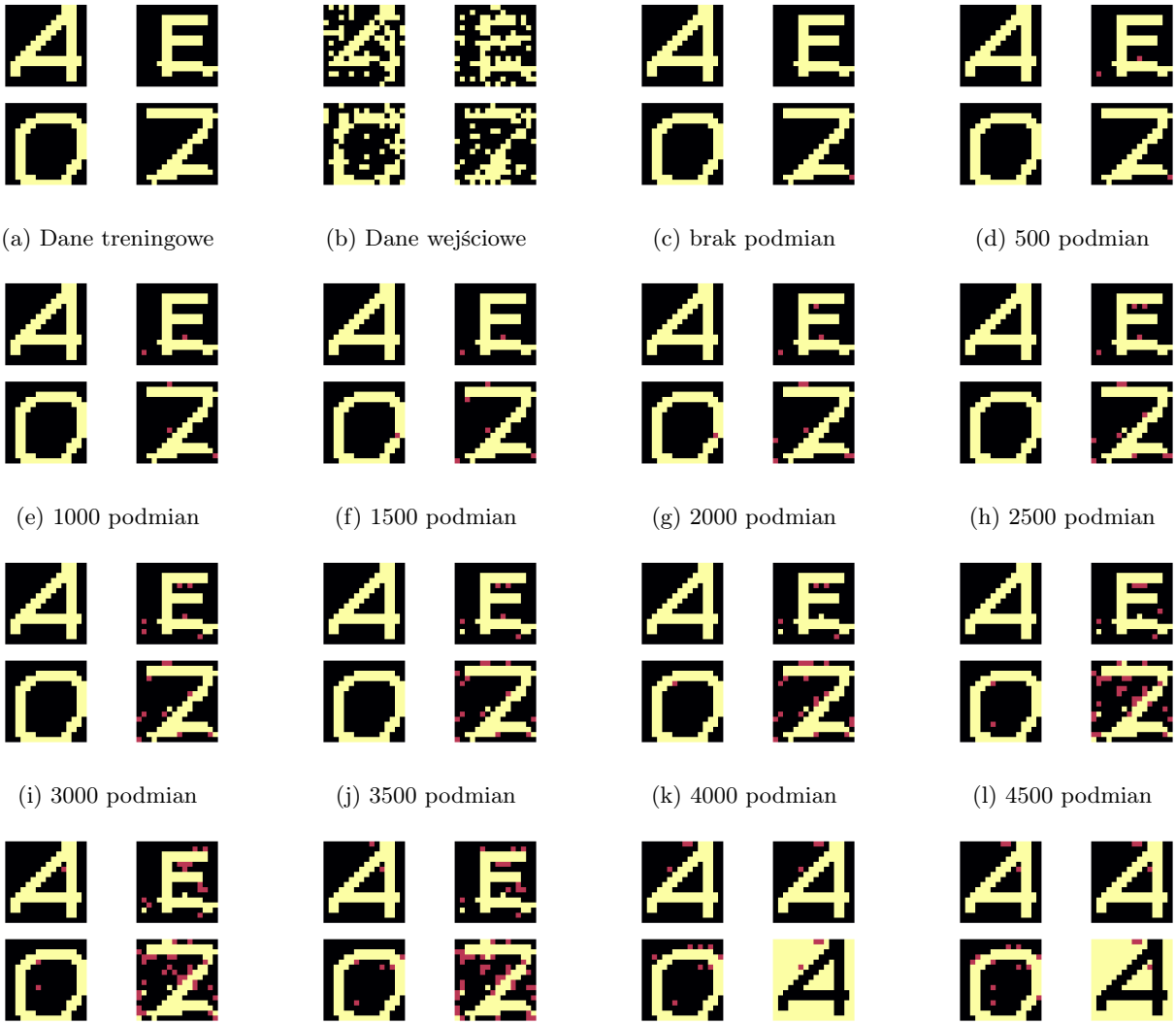
(a) Rzeczywista postać części danych treningowych

(b) Warstwa widoczna danych treningowych

(c) Warstwa widoczna danych wejściowych

(d) Warstwa widoczna danych wyjściowych

Rysunek 10.3: Wyniki działania sieci z 256 neuroanmi ukrytymi



(a) Dane treningowe (b) Dane wejściowe (c) brak podmian (d) 500 podmian

(e) 1000 podmian (f) 1500 podmian (g) 2000 podmian (h) 2500 podmian

(i) 3000 podmian (j) 3500 podmian (k) 4000 podmian (l) 4500 podmian

(m) 5000 podmian (n) 5500 podmian (o) 6000 podmian (p) 6500 podmian

Rysunek 10.4: Zakłócanie macierzy wag

10.0.5 Gradient bez gradientu

Do ucznenia sieci Hopfielda można podejść od innej strony. Rozważmy równanie:

$$W = W + \eta_i(\sum_{\xi \in wzorce} \xi\xi^T - \sum_{\xi \notin wzorce} \xi\xi^T) \tag{10.7}$$

Próbujemy zminimalizować energię sieci w stanach odpowiadających wzorcom i zmaksymalizować energię w stanach które nie odpowiadają wzorcom. W powyższym równaniu zwiększamy wpływ wzorców na macierz wag i zmniejszamy wpływ "niewzorców".

- Macierz wag jest inicjalizowana losowo.
- Do sieci wprowadzamy wzorzec x^μ
- Sieć ewoluuje i osiąga jakiś stabilny stan ξ^s
- Aktualizujemy wagi

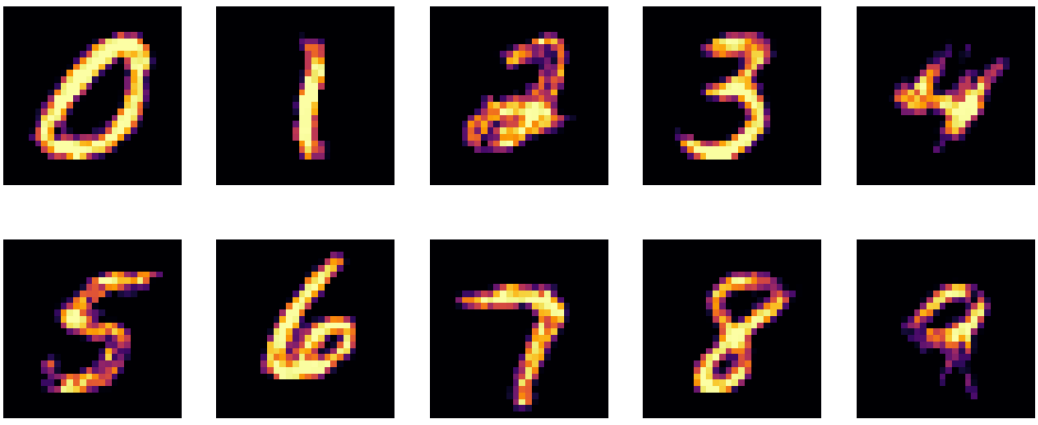
$$\mathbf{W} = \mathbf{W} + \eta(x^\mu(x^\mu)^T - \xi^s(\xi^s)^T) \tag{10.8}$$

Kroki 2-4 powtarzamy odpowiednio wybierając parametry (ilość iteracji i learning rate η) dla wszystkich wzorców i wielokrotnie aż uda się osiągnąć zadowalające efekty. Dla wzorców które są ważniejsze lub trudniejsze do odtworzenia powinniśmy wybrać większą ilość iteracji i/lub wyższą wartość learning rate tak aby cały proces miał większy wpływ na ostateczną macierz wag.

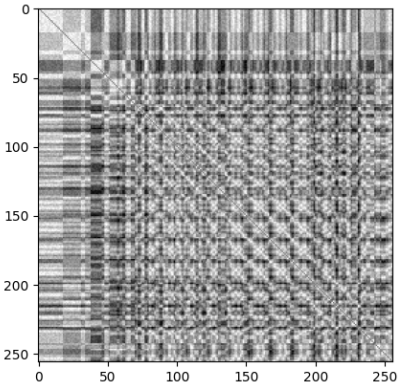
Gradientowe obliczanie macierzy wag jest niezbędne przy zwiększaniu liczby wzorców w stosunku do liczby neuronów, wówczas metoda Hebba przestaje być skuteczna produkując błędnie położone atraktory. Już przy pięciu wzorcach dla 256 neuronów sieć uczona metodą Hebba ma tendecję do zwracania błędnych wyników. Liczba iteracji potrzebnych do stabilizacji wyniosła od 1 do 4, zaś liczba błędnych pikseli od 0 do 37. Dla porównania sieć o tych samych wymiarach uczona metodą gradientową zwróciła wyniki o wiele bliższe do wzorców, liczba iteracji - od 1 do 7, liczba błędów - od 0 do 3. Druga metoda wymaga odpowiedniego doboru parametrów uczenia, w tym przypadku każdy wzorzec oddzielnie aktualizował macierz wag 100-krotnie z odpowiednimi wartościami learning rate (R - 1.5, O - 0.4, X - 0.4, Z - 0.1, T - 0.03). Jak widać metoda gradientowa pozwala na osiągnięcie lepszych wyników od metody Hebba, ale wymaga większych mocy obliczeniowych i starannie dobranych parametrów uczenia co wiąże się z potrzebą wytrenowania wielu sieci zanim uda się osiągnąć wymaganą skuteczność.

10.0.6 Mediana

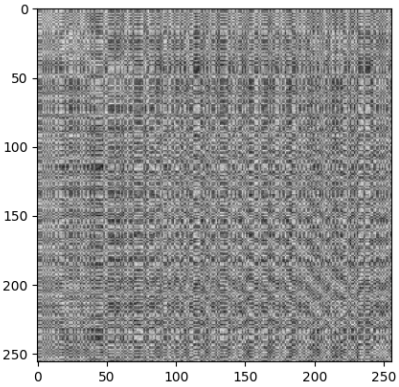
Tym razem wzorzec klasyfikacyjny to mediana odpowiadających sobie pikseli, pozostałe założenia pozostają bez zmian. Przykładowy zestaw wzorców klasyfikacyjnych dla jednej z iteracji:



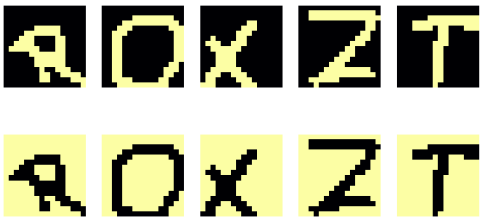
W całości eksperymentu model oparty na medianie osiągnął 40,3% poprawnych klasyfikacji, czyli gorzej niż przy średniej.



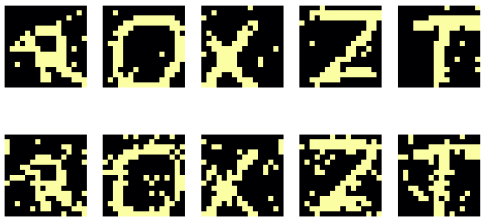
(a) Macierz wag dla metody Hebba



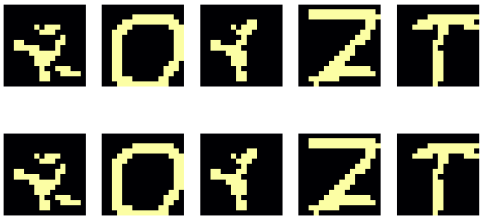
(b) Macierz wag dla metody gradientowej



(c) Wzorce i wzorce odwrotne



(d) Dane wejściowe



(e) Dane wyjściowe sieci uczonej metodą Hebba

(f) Dane wyjściowe sieci uczonej metodą gradientową

Rysunek 10.5: Porównanie metod uczenia sieci Hopfielda.