Zastosowanie do optymalizacji nieliniowej

Wstęp

Optymalizacja nieliniowa to obszar badań operacyjnych, a skuteczne algorytmy dla niektórych problemów w tym obszarze są trudne do znalezienia. W tej Części opisujemy problem komiwojażera i omawiamy, jak ten problem jest sformułowany jako problem optymalizacji nieliniowej w celu wykorzystania sieci neuronowych (Hopfielda i Kohonena) do znalezienia optymalnego rozwiązania. Zaczniemy od wyjaśnienia pojęć optymalizacji liniowej, całkowitej liniowej i nieliniowej. Problem optymalizacji ma funkcję celu i zestaw ograniczeń na zmienne. Problem polega na znalezieniu wartości zmiennych, które prowadzą do optymalnej wartości funkcji celu przy spełnieniu wszystkich ograniczeń. Funkcja celu może być funkcją liniową w zmiennych lub może być funkcją nieliniową. Na przykład może to być funkcja wyrażająca całkowity koszt konkretnego planu produkcyjnego lub funkcja dająca zysk netto z grupy produktów, które dzielą dany zbiór zasobów. Celem może być znalezienie minimalnej wartości funkcji celu, jeśli na przykład reprezentuje ona koszt, lub znalezienie maksymalnej wartości funkcji zysku. Zasoby współdzielone przez produkty podczas ich wytwarzania są zwykle ograniczone lub mają inne ograniczenia w ich dostępności. Ta uwaga prowadzi do określenia ograniczeń dla problemu. Każde ograniczenie ma zwykle postać równania lub nierówności. Lewa strona takiego równania lub nierówności jest wyrażeniem w zmiennych dla problemu, a prawa strona jest stałą. Mówi się, że ograniczenia są liniowe lub nieliniowe w zależności od tego, czy wyrażenie po lewej stronie jest funkcją liniową czy nieliniową zmiennych. Problem programowania liniowego to problem optymalizacji z liniową funkcją celu oraz zbiorem ograniczeń liniowych. Problem programowania liniowego całkowitoliczbowego to problem programowania liniowego, w którym zmienne muszą mieć wartości całkowite. Problem optymalizacji nieliniowej ma jedno lub więcej ograniczeń nieliniowych i/lub funkcja celu jest nieliniowa. Oto kilka przykładów instrukcji, które określają funkcje celu i ograniczenia:

• Liniowa funkcja celu: Maksymalizuj $Z = 3X_1 + 4X_2 + 5,7X_3$

• Liniowe ograniczenie równości: $13X_1 - 4,5X_2 + 7X_3 = 22$

• Ograniczenie liniowej nierówności: $3,6X_1 + 8,4X_2 - 1,7X_3 \leq 10,9$

• Nieliniowa funkcja celu: Minimalizuj $Z = 5X^2 + 7XY + Y^2$

• Nieliniowe ograniczenie równości: $4X + 3XY + 7Y + 2Y^2 = 37,6$

• Nieliniowe ograniczenie nierówności: $4,8X + 5,3XY + 6,2Y^2 \geq 34,56$

Przykładem problemu programowania liniowego jest problem mieszania. Przykładem problemu z mieszaniem jest robienie lodów o różnych smakach, polegających na łączeniu różnych składników, takich jak cukier, różne orzechy itd., aby uzyskać różne ilości lodów o wielu smakach. Celem problemu jest znalezienie ilości poszczególnych smaków lodów do wyprodukowania przy danych zapasach wszystkich składników, tak aby łączny zysk był zmaksymalizowany. Przykładem problemu optymalizacji nieliniowej jest problem programowania kwadratowego. Wszystkie ograniczenia są liniowe, ale funkcja celu ma postać kwadratową. Forma kwadratowa jest wyrażeniem dwóch zmiennych z 2 dla sumy wykładników dwóch zmiennych w każdym członie. Przykładem problemu programowania kwadratowego jest prosty problem strategii inwestycyjnej, który można sformułować w następujący sposób. Chcesz zainwestować określoną kwotę w akcje wzrostowe i spekulacyjne, osiągając co najmniej 25% zwrotu. Chcesz ograniczyć swoją inwestycję w akcje spekulacyjne do nie więcej niż 40% całkowitej inwestycji. Zakładasz, że oczekiwany zwrot z akcji wzrostowych wynosi 18%, podczas gdy z akcji spekulacyjnych wynosi 38%. Załóżmy, że G i S reprezentują proporcję Twoich inwestycji

odpowiednio w akcje wzrostowe i spekulacyjne. Do tej pory określiłeś następujące ograniczenia. Są to ograniczenia liniowe:

G + S = 1 To mówi, że proporcje sumują się do 1.

S ≤ 0,4 Oznacza to, że odsetek zainwestowany w akcje spekulacyjne nie jest ponad 40%.

1,18G + 1,38S 1,25 Oznacza to, że oczekiwany zwrot z tych inwestycji powinien wynosić co najmniej 25%.

Teraz należy określić funkcję celu. Określiłeś już oczekiwany zwrot, jaki chcesz osiągnąć. Załóżmy, że jesteś konserwatywnym inwestorem i chcesz zminimalizować wariancję zwrotu. Wariancja jest formą kwadratową. Załóżmy, że jest to określone jako:

$2G^2 + 3S^2 - GS$

Ta kwadratowa forma, która jest funkcją G i S, jest twoją funkcją celu, którą chcesz zminimalizować z uwzględnieniem (liniowych) ograniczeń, które zostały wcześniej określone.

Sieci neuronowe dla problemów z optymalizacją

Możliwe jest zbudowanie sieci neuronowej w celu znalezienia wartości zmiennych, które odpowiadają optymalnej wartości funkcji celu problemu. Na przykład sieci neuronowe korzystające z reguły uczenia Widrow-Hoff znajdują minimalną wartość funkcji błędu przy użyciu najmniejszego błędu średniokwadratowego. Sieci neuronowe, takie jak sieć z propagacją wsteczną ze sprzężeniem do przodu, używają do tego celu metody najbardziej stromego opadania i znajdują lokalne minimum błędu, jeśli nie minimum globalne. Z drugiej strony maszyna Boltzmanna lub maszyna Cauchy'ego wykorzystuje metody statystyczne i prawdopodobieństwa i osiąga sukces w znalezieniu globalnego minimum funkcji błędu. Mamy więc pomysł, jak wykorzystać sieć neuronową, aby znaleźć optymalną wartość funkcji. Pozostaje pytanie, jak potraktować ograniczenia problemu optymalizacji w działaniu sieci neuronowej. Dobrym przykładem odpowiedzi na to pytanie jest problem komiwojażera. Omówmy teraz ten przykład.

Problem z podróżującym sprzedawcą

Problem komiwojażera jest dobrze znany w optymalizacji. Jego matematyczne sformułowanie jest proste, można też podać prostą strategię rozwiązania. Taka strategia jest często niepraktyczna, a jak dotąd nie ma skutecznego algorytmu rozwiązania tego problemu, który konsekwentnie działa we wszystkich przypadkach. Problem komiwojażera to jeden z tak zwanych problemów NP-zupełnych, o którym więcej przeczytasz w dalszej części. Oznacza to, że jakikolwiek algorytm dla tego problemu będzie niepraktyczny na niektórych przykładach. Podejście oparte na sieciach neuronowych ma tendencję do zapewniania rozwiązań o krótszym czasie obliczeniowym niż inne dostępne algorytmy do wykorzystania na komputerze cyfrowym. Problem jest zdefiniowany w następujący sposób. Podróżujący sprzedawca ma do odwiedzenia kilka miast. Sekwencja, w jakiej sprzedawca odwiedza różne miasta, nazywana jest wycieczką. Wycieczka powinna być taka, aby każde miasto na liście było odwiedzane raz i tylko raz, z wyjątkiem tego, że wraca do miasta, z którego startuje. Celem jest znalezienie wycieczki, która minimalizuje całkowitą odległość, jaką pokonuje sprzedawca, spośród wszystkich wycieczek spełniających to kryterium. Prostą strategią rozwiązania tego problemu jest wyliczenie wszystkich możliwych wycieczek – wycieczka jest wykonalna, jeśli spełnia kryterium, że każde miasto jest odwiedzane tylko raz – aby obliczyć całkowitą odległość dla każdej wycieczki i wybrać trasę o najmniejszej całkowitej odległości. Ta prosta strategia staje się niepraktyczna, jeśli liczba miast jest duża. Na przykład, jeśli podróżujący sprzedawca może odwiedzić 10 miast (nie licząc domu), jest

ich 10! = 3628800 możliwych wycieczek, gdzie 10! oznacza silnię 10 — iloczyn wszystkich liczb całkowitych od 1 do 10 - i jest liczbą różnych kombinacji 10 miast. Liczba ta wzrasta do ponad 6,2 miliarda z zaledwie 13 miastami w trasie i do ponad biliona z 15 miastami.

TSP w pigułce

Dla n miast do odwiedzenia, niech X_{ij} będzie zmienną, która ma wartość 1, jeśli sprzedawca jedzie z miasta i do miasta j i wartość 0, jeśli sprzedawca nie jedzie z miasta i do miasta j. Niech d_{ij} będzie odległością od miasta i do miasta j. Problem komiwojażera (TSP) przedstawia się następująco: Zminimalizuj liniową funkcję celu:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij} d_{ij}$$

z zastrzeżeniem:

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} X_{ij} = 1$$ dla każdego j=1, ..., n (ograniczenie liniowe)

$$\sum_{\substack{i=1 \\ j \neq i}}^{n} X_{ij} = 1$$ dla każdego i=1, ..., n (ograniczenie liniowe)

- X_{ij} dla 1 dla wszystkich i oraz j (ograniczenie liczby całkowitej)

Jest to problem programowania liniowego liczb całkowitych 0-1.

Rozwiązanie przez sieć neuronową

Ta sekcja pokazuje, w jaki sposób ograniczenia liniowe i całkowite TSP są absorbowane przez funkcję celu, która jest nieliniowa dla rozwiązania przez sieć neuronową. Pierwszym rozważaniem przy formułowaniu problemu optymalizacji jest identyfikacja podstawowych zmiennych i rodzaju wartości, jakie mogą mieć. W przypadku komiwojażera każde miasto musi być odwiedzone raz i tylko raz, z wyjątkiem miasta, od którego zaczyna się. Załóżmy, że przyjmujesz za pewnik, że ostatni etap wycieczki to podróż między ostatnim odwiedzonym miastem a miastem, z którego rozpoczyna się wycieczka, tak więc ta część wycieczki nie musi być wyraźnie uwzględniona w sformułowaniu. Następnie przy n miastach do odwiedzenia, jedyną potrzebną informacją dla każdego miasta jest pozycja tego miasta w kolejności odwiedzania miast w wycieczce. Sugeruje to, że uporządkowana n-krotka jest powiązana z każdym miastem z jakimś elementem równym 1, a reszta z n-1 elementów równymi 0. W reprezentacji sieci neuronowej wymaga to n neuronów powiązanych z jednym miastem. Tylko jeden z tych n neuronów odpowiadających pozycji miasta, w kolejności miast w wycieczce, uruchamia się lub ma wyjście 1. Ponieważ jest n miast do odwiedzenia, potrzebujesz n^2 neuronów w sieci. Jeśli wszystkie te neurony są ułożone w kwadratową tablicę, potrzebujesz jednej 1 w każdym rzędzie i każdej kolumnie tej tablicy, aby wskazać, że każde miasto jest odwiedzane tylko raz. Niech x_{ij} będzie zmienną oznaczającą, że miasto i jest j-tym miastem odwiedzanym podczas wycieczki. Wtedy x_{ij} jest wyjściem

neuronu j w tablicy neuronów odpowiadającej i-temu miastu. Masz n^2 takich zmiennych, a ich wartości są binarne, 0 lub 1. Dodatkowo tylko n z tych zmiennych powinno mieć wartość 1 w rozwiązaniu. Co więcej, dokładnie jeden z x z tym samym pierwszym indeksem dolnym (wartość i) powinien mieć wartość 1. Dzieje się tak, ponieważ dane miasto może zajmować tylko jedną pozycję w kolejności zwiedzania. Podobnie dokładnie jeden z x z tym samym drugim indeksem dolnym (wartość j) powinien mieć wartość 1. Dzieje się tak dlatego, że dana pozycja w wycieczce może być zajęta tylko przez jedno miasto. To są ograniczenia problemu. Jak więc opisujesz trasę? Jako miasto początkowe wycieczki przyjmujemy miasto 1 w tablicy miast. Wycieczkę można określić za pomocą sekwencji 1, a, b, c, …, q, wskazującej, że miasta odwiedzone w wycieczce w kolejności od 1 to a, b, c, …, q i z powrotem do 1. Uwaga że sekwencja indeksów dolnych a, b, …, q jest permutacją 2, 3, … n – 1, x_{a1}=1, x_{b2}=1 itd. Mając zamrożone miasto 1 jako pierwsze miasto wycieczki i zauważ, że odległości są symetryczne, wyraźna liczba wycieczek, które spełniają ograniczenia, nie wynosi n!, gdy w trasie jest n miast, jak podano wcześniej. Jest to znacznie mniej, a mianowicie n!/2n. Tak więc, gdy n wynosi 10, liczba różnych możliwych do wykonania wycieczek wynosi 10!/20, czyli 181 440. Jeśli n wynosi 15, to nadal wynosi ponad 43 miliardy, a liczba ta przekracza bilion z 17 miastami biorącymi udział w trasie. Jednak z praktycznego punktu widzenia nie ma większego pocieszenia, wiedząc, że w przypadku 13 miast 13! to ponad 6,2 miliarda, a 13!/26 to tylko 239,5 miliona - to wciąż trudny problem kombinatoryczny. Koszt wycieczki to całkowity przebyty dystans i należy go zminimalizować. Całkowita odległość przebyta w ramach wycieczki to suma odległości z każdego miasta do następnego. Funkcja celu ma jeden wyraz, który odpowiada całkowitej odległości przebytej podczas wycieczki. Pozostałe terminy, po jednym dla każdego ograniczenia, w funkcji celu są wyrażeniami, z których każde osiąga minimalną wartość wtedy i tylko wtedy, gdy odpowiednie ograniczenie jest spełnione. Funkcja celu przybiera wtedy następującą postać. Hopfield i Tank sformułowali problem jako problem minimalizacji energii. Jest więc zwyczajem odwoływanie się do wartości funkcji celu tego problemu, gdy do jego rozwiązania stosuje się sieć neuronową typu Hopfielda, jako poziom energii E sieci. Celem jest zminimalizowanie tego poziomu energii. Formułując równanie dla E, używa się stałych parametrów A_1, A_2, A_3 i A_4 jako współczynników w różnych terminach wyrażenia po prawej stronie równania. Równanie definiujące E podano w następujący sposób. Zauważ, że zapis w tym równaniu obejmuje d_{ij} dla odległości od miasta i do miasta j.

$$E = A_1 \Sigma_i\Sigma_k \Sigma_{j\neq k}x_{ik}x_{ij} + A_2 \Sigma_i \Sigma_k x_i\Sigma_k \Sigma_{j\neq k}x_{ki}x_{ji} + A_3[(\Sigma_i\Sigma_k x_{ik}) - n]^2 + A_4\Sigma_k \Sigma_k \Sigma_{j\neq k} \Sigma_i d_{kj}x_{ki}(x_{j,i+1} + x_{j,i-1})$$

Nasza pierwsza obserwacja w tym momencie jest taka, że E jest nieliniową funkcją x, ponieważ masz w niej elementy kwadratowe. Tak więc takie sformułowanie problemu komiwojażera czyni go nieliniowym problemem optymalizacji. Wszystkie sumy wskazane przez wystąpienia symbolu sumowania mieszczą się w zakresie od 1 do n dla wartości ich odpowiednich indeksów. Oznacza to, że ta sama suma, taka jak $x_{12}x_{33}$, również jako $x_{33}x_{12}$, pojawia się dwa razy z wymienionymi tylko czynnikami w kolejności ich występowania w sumie. Z tego powodu wielu autorów używa dodatkowego współczynnika 1/2 dla każdego terminu w wyrażeniu na E. Jednak po zminimalizowaniu ilości z przy danym zestawie wartości dla zmiennych w wyrażeniu na z, te same wartości dla te zmienne minimalizują również każdą całkowitą lub ułamkową wielokrotność z. Trzecie sumowanie w pierwszym członie jest nad indeksem j, od 1 do n, ale z wyłączeniem jakiejkolwiek wartości k. Zapobiega to używaniu czegoś takiego jak $x_{12}x_{12}$. Zatem pierwszy wyraz jest skrótem sumy $n^2(n – 1)$ wyrazów bez dwóch czynników w sumie równych. Ten termin jest zawarty, aby odpowiadać ograniczeniu, że nie więcej niż jeden neuron w tym samym wierszu może wypisać 1. Tak więc otrzymujesz 0 dla tego terminu z poprawnym rozwiązaniem. Odnosi się to również do drugiego członu po prawej stronie

równania dla E. Zauważ, że dla dowolnej wartości indeksu i, x_{ii} ma wartość 0, ponieważ nie wykonujesz ruchu z miasta i do tego samego w każdej z wycieczek, które uważasz za rozwiązanie tego problemu. Trzeci wyraz w wyrażeniu na E ma minimalną wartość 0, która jest osiągana wtedy i tylko wtedy, gdy dokładnie n z n^2 x mają wartość 1, a pozostałe 0. Ostatni wyraz wyraża cel, jakim jest znalezienie wycieczki o najmniejszej całkowitej przebytej odległości, wskazując najkrótszą trasę spośród wszystkich możliwych wycieczek dla podróżującego sprzedawcy. Inną ważną kwestią dotyczącą wartości indeksów po prawej stronie równania dla E jest to, co dzieje się z i+1, na przykład, gdy i jest już równe n, a z i-1, gdy i jest równe do 1. Wartości i + 1 oraz i – 1 wydają się być wartościami niemożliwymi, będąc poza dozwolonym zakresem od 1 do n. Sztuczka polega na zastąpieniu tych wartości ich modułami względem n. Oznacza to, że wartość n + 1 jest zastępowana przez 1, a wartość 0 jest zastępowana przez n w opisanych sytuacjach. Wartości modułowe uzyskuje się w następujący sposób. Jeśli chcemy, powiedzmy 13 modulo 5, odejmujemy 5 tyle razy, ile to możliwe, od 13, aż reszta będzie wartością między 0 a 4, gdzie 4 będzie 5 – 1. Ponieważ możemy odjąć 5 dwa razy od 13, aby otrzymać resztę z 3, czyli od 0 do 4, 3 jest wartością 13 modulo 5. Zatem (n + 3) modulo n wynosi 3, jak wspomniano wcześniej. Innym sposobem patrzenia na te wyniki jest to, że 3 to 13 modulo 5, ponieważ jeśli odejmiemy 3 od 13, otrzymamy liczbę podzielną przez 5, lub która ma 5 jako czynnik. Odjęcie 3 od n + 3 daje n, które ma n jako czynnik. Więc 3 to (n + 3) modulo n. W przypadku –1, odejmując od niego (n – 1), otrzymujemy -n, które można podzielić przez n, otrzymując –1. Zatem (n – 1) jest wartością (–1) modulo n.

Przykład problemu z komiwojażerem do obliczeń ręcznych

Załóżmy, że wycieczka obejmuje cztery miasta. Nazwij te miasta C_1, C_2, C_3 i C_4. Niech macierz odległości będzie następującą macierzą D.

$$D = \begin{matrix} 0 & 10 & 14 & 7 \\ 10 & 0 & 6 & 12 \\ 14 & 6 & 0 & 9 \\ 7 & 12 & 9 & 0 \end{matrix}$$

Z naszej wcześniejszej dyskusji na temat liczby ważnych i odrębnych wycieczek wnioskujemy, że są tylko trzy takie wycieczki. Ponieważ jest to tak mała liczba, możemy sobie pozwolić na wyliczenie trzech tras, znalezienie związanych z nimi wartości energii i wybranie trasy, która ma najniższy poziom energii spośród trzech. Trzy wycieczki to:

• Wycieczka 1. 1 - 2 - 3 - 4 - 1 Podczas tej wycieczki najpierw odwiedza się miasto 2, następnie miasto 3, skąd sprzedawca udaje się do miasta 4, a następnie wraca do miasta 1. W przypadku miasta 2 odpowiedni zamówiony tablica to (1, 0, 0, 0), ponieważ miasto 2 jest pierwszym w tej kombinacji miast. Wtedy $x_{21} = 1$, $x_{22} = 0$, $x_{23} = 0$, $x_{24} = 0$. Również (0, 1, 0, 0), (0, 0, 1, 0) i (0, 0, 0, 1) odpowiadają miasta odpowiednio 3, 4 i 1. Całkowita odległość trasy wynosi $d_{12} + d_{23} + d_{34} + d_{41}= 10 + 6 + 9 + 7 = 32$.

• Trasa 2. 1 - 3 - 4 - 2 - 1

• Trasa 3. 1 - 4 - 2 - 3 - 1

Wydaje się, że jest tu jakaś rozbieżność. Jeśli istnieje, potrzebujemy wyjaśnienia. Rozbieżność polega na tym, że możemy znaleźć o wiele więcej wycieczek, które powinny być ważne, ponieważ żadne miasto nie jest odwiedzane więcej niż raz. Można powiedzieć, że różnią się one od trzech wymienionych wcześniej. Niektóre z tych dodatkowych wycieczek to:

• Trasa 4. 1 - 2 - 4 - 3 - 1

• Trasa 5. 3 - 2 - 4 - 1 - 3

• Trasa 6. 2 - 1 - 4 - 3 - 2

Nie ma wątpliwości, że te trzy trasy różnią się od pierwszego zestawu trzech tras. W każdej z tych trzech wycieczek każde miasto jest odwiedzane dokładnie raz, zgodnie z wymaganiami problemu. Więc są to również ważne wycieczki. Dlaczego nasz wzór dał nam 3 dla wartości liczby możliwych ważnych wycieczek, podczas gdy jesteśmy w stanie znaleźć 6?

Odpowiedź tkwi w tym, że jeśli dwie ważne trasy są symetryczne i mają ten sam poziom energii, ponieważ mają taką samą wartość dla całkowitej odległości przebytej w trasie, to jedna z nich jest w pewnym sensie zbędna lub jedna z nich może być uważane za zdegenerowane, używając terminologii wspólnej dla tego kontekstu. Dopóki są ważne i dają taką samą całkowitą odległość, obie wycieczki nie są indywidualnie interesujące i wystarczy jedna z nich. Poprzez prostą inspekcję można znaleźć łączne odległości dla sześciu wymienionych tras. Są to 32 dla trasy 1, 32 również dla trasy 6, 45 dla każdej z tras 2 i 4 oraz 39 dla każdej z tras 3 i 5. Zauważ również, że trasa 6 nie różni się zbytnio od trasy 1. Zamiast zaczynać od miasta 1, tak jak w przypadku trasy 1, jeśli zaczniesz w mieście 2, a stamtąd podążasz za trasą 1 w odwrotnej kolejności do zwiedzania miast, otrzymasz trasę 6. Dlatego pokonana odległość jest taka sama dla obu tych tras. Podobne zależności można znaleźć dla innych par wycieczek, które mają taką samą całkowitą odległość dla wycieczki. Odwracając kolejność miast w wycieczce lub wykonując permutację kołową kolejności miast w wycieczce, otrzymujesz kolejną wycieczkę o tej samej całkowitej odległości. W ten sposób możesz znaleźć wycieczki. W ten sposób możliwe są tylko trzy różne odległości całkowite, a spośród nich 32 jest najniższą. Wycieczka 1 - 2 - 3 - 4 - 1 jest najkrótsza i stanowi optymalne rozwiązanie tego problemu komiwojażera. Istnieje alternatywne optymalne rozwiązanie, a mianowicie trasa 6, z 32 na całkowitą długość trasy. Problemem jest znalezienie optymalnej trasy, a nie znalezienie wszystkich optymalnych tras, jeśli istnieje więcej niż jedna. To jest powód, dla którego formuła na liczbę odrębnych ważnych tras sugeruje w tym przypadku tylko trzy różne objazdy. Wzór, którego użyliśmy, aby uzyskać liczbę prawidłowych i odrębnych tras równych 3, opiera się na wyeliminowaniu takiej symetrii. Aby wyjaśnić całą tę dyskusję, powinieneś określić poziomy energii wszystkich sześciu zidentyfikowanych wcześniej tras, mając nadzieję na znalezienie par o identycznych poziomach energii. Zauważ, że pierwszy składnik po prawej stronie równania daje 0 dla prawidłowej trasy, ponieważ ten składnik ma zapewnić, że w każdym rzędzie nie będzie więcej niż jedna 1. Oznacza to, że w dowolnym podsumowaniu w pierwszym semestrze co najmniej jeden z czynników, x_{ik} lub x_{ij}, gdzie k j musi wynosić 0, aby trasa była ważna. Tak więc wszystkie te sumy wynoszą 0, co powoduje, że pierwszy składnik sam w sobie wynosi 0. Podobnie, drugi składnik to 0 dla ważnej trasy, ponieważ w każdej sumie co najmniej jeden z czynników, x_{ki} lub x_{ji}, gdzie k_j musi wynosić 0, aby wycieczka. W sumie dokładnie 4 z 16 x to 1, co daje w sumie 4 x. To powoduje, że trzeci wyraz jest równy 0 dla prawidłowej trasy. Obserwacje te jasno pokazują, że dla ważnych tras nie ma znaczenia, jakie wartości są przypisane do parametrów A_1, A_2 i A_3. Przypisanie dużych wartości tym parametrom spowodowałoby, że poziomy energii dla nieważnych tras byłyby znacznie większe niż poziomy energii dla ważnych tras. Tym samym wycieczki te stają się nieatrakcyjne dla rozwiązania problemu komiwojażera. Użyjmy wartości 1/2 dla parametru A_4. Zademonstrujmy obliczenie wartości dla ostatniego członu w równaniu na E, w przypadku trasy 1. Przypomnijmy, że potrzebne równanie to

$$E = A_1 \Sigma_i\Sigma_k \Sigma_{j\neq k} x_{ik}x_{ij} + A_2 \Sigma_i\Sigma_k \Sigma_{j\neq k}x_{ki}x_{ji} + A_3[(\Sigma_i\Sigma_k x_{jk}) - n]^2 + A_4\Sigma_k \Sigma_{j\neq k}\Sigma_i d_{kj}x_{ki}(x_{j,i+1} + x_{j,i-1})$$

Ostatni termin rozszerza się zgodnie z następującym obliczeniem:

$$A_4\{\ d_{12}[x_{12}(x_{23}+x_{21})+x_{13}(x_{24}+x_{22})+x_{14}(x_{21}+x_{23})]+$$
$$d_{13}[x_{12}(x_{33}+x_{31})+x_{13}(x_{34}+x_{32})+x_{14}(x_{31}+x_{33})]+$$
$$d_{14}[x_{12}(x_{43}+x_{41})+x_{13}(x_{44}+x_{42})+x_{14}(x_{41}+x_{43})]+$$
$$d_{21}[x_{21}(x_{12}+x_{14})+x_{23}(x_{14}+x_{12})+x_{24}(x_{11}+x_{13})]+$$
$$d_{23}[x_{21}(x_{32}+x_{34})+x_{23}(x_{34}+x_{32})+x_{24}(x_{31}+x_{33})]+$$
$$d_{24}[x_{21}(x_{42}+x_{44})+x_{23}(x_{44}+x_{42})+x_{24}(x_{41}+x_{43})]+$$
$$d_{31}[x_{31}(x_{12}+x_{14})+x_{32}(x_{13}+x_{11})+x_{34}(x_{11}+x_{13})]+$$
$$d_{32}[x_{31}(x_{22}+x_{24})+x_{32}(x_{23}+x_{21})+x_{34}(x_{21}+x_{23})]+$$
$$d_{34}[x_{31}(x_{42}+x_{44})+x_{32}(x_{43}+x_{41})+x_{34}(x_{41}+x_{43})]+$$
$$d_{41}[x_{41}(x_{12}+x_{14})+x_{42}(x_{13}+x_{11})+x_{43}(x_{14}+x_{12})]+$$
$$d_{42}[x_{41}(x_{22}+x_{24})+x_{42}(x_{23}+x_{21})+x_{43}(x_{24}+x_{22})]+$$
$$d_{43}[x_{41}(x_{32}+x_{34})+x_{42}(x_{33}+x_{31})+x_{43}(x_{34}+x_{32})]\ \}$$

Gdy odpowiednie wartości zostaną zastąpione dla trasy 1 - 2 - 3 - 4 - 1, poprzednia kalkulacja staje się:

$$1/2\{10[0(0+1)+0(0+0)+1(1+0)]+14[0(0+0)+0(0+1)+1(0+0)]+$$
$$7[0(1+0)+0(0+0)+1(0+1)]+10[1(0+1)+0(1+0)+0(0+0)]+$$
$$6[1(1+0)+0(0+1)+0(0+0)]+12[1(0+0)+0(0+0)+0(0+1)]+$$
$$14[0(0+1)+1(0+0)+0(0+0)]+6[0(0+0)+1(0+1)+0(1+0)]+$$
$$9[0(0+0)+1(1+0)+0(0+1)]+7[0(0+1)+0(0+0)+1(1+0)]+$$
$$12[0(0+0)+0(0+1)+1(0+0)]+9[0(1+0)+0(0+0)+1(0+1)]\}$$
$$= 1/2(\ 10+0+7+10+6+0+0+6+9+7+0+9)$$
$$= 1/2(64)$$
$$= 32$$

Tabela zawiera wartości, które otrzymujemy dla czwartego członu po prawej stronie równania oraz dla E, z sześcioma wymienionymi trasami.

Tour #	Non-Zero x's	Value for the Last Term	Energy Level	Comment
1	$x_{14}, x_{21}, x_{32}, x_{43}$	32	32	1 - 2 - 3 - 4 - 1 tour
2	$x_{14}, x_{23}, x_{31}, x_{42}$	45	45	1 - 3 - 4 - 2 - 1 tour
3	$x_{14}, x_{22}, x_{33}, x_{41}$	39	39	1 - 4 - 2 - 3 - 1 tour
4	$x_{14}, x_{21}, x_{33}, x_{42}$	45	45	1 - 2 - 4 - 3 - 1 tour
5	$x_{13}, x_{21}, x_{34}, x_{42}$	39	39	3 - 2 - 4 - 1 - 3 tour
6	$x_{11}, x_{24}, x_{33}, x_{42}$	32	32	2 - 1 - 4 - 3 - 2 tour

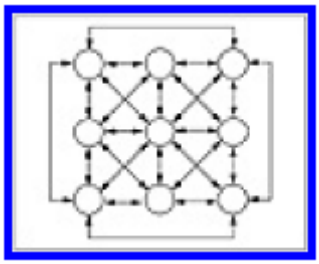
Problem z siecią neuronową dla podróżujących sprzedawców

Hopfield i Tank wykorzystali sieć neuronową do rozwiązania problemu komiwojażera. Otrzymane rozwiązanie może w niektórych przypadkach nie być optymalne. Ale ogólnie rzecz biorąc, możesz

uzyskać rozwiązanie zbliżone do rozwiązania optymalnego. Dla problemu komiwojażera nie można znaleźć konsekwentnie efektywnej metody rozwiązania, gdyż udowodniono, że należy on do zespołu problemów zwanych problemami NP-zupełnymi. Problemy NP-zupełne wyróżniają się tym, że nie ma znanego algorytmu, który byłby wydajny i praktyczny, i jest małe prawdopodobieństwo, że taki algorytm zostanie opracowany w przyszłości. Jest to zastrzeżenie, o którym należy pamiętać podczas korzystania z sieci neuronowej w celu rozwiązania problemu komiwojażera.

Wybór i układ sieci

Opiszemy wykorzystanie sieci Hopfield do próby rozwiązania problemu komiwojażera. W sieci znajduje się n^2 neuronów ułożonych w dwuwymiarową tablicę n neuronów na wiersz i n na kolumnę. Sieć jest w pełni podłączona. Połączenia w sieci w każdym rzędzie iw każdej kolumnie są połączeniami bocznymi. Dla ilustracji rozmieszczenie neuronów w sieci wraz z ich połączeniami pokazano na rysunku dla przypadku trzech miast. Aby uniknąć bałaganu, nie pokazano połączeń między neuronami po przekątnej.



Najważniejszym zadaniem jest wtedy znalezienie odpowiedniej macierzy wag połączeń. Jest ona skonstruowana z uwzględnieniem tego, że nieważne wycieczki należy zapobiegać i preferować ważne wycieczki. Należy wziąć pod uwagę, że np. żadne dwa neurony w tym samym rzędzie (kolumnie) nie powinny odpalać w tym samym cyklu działania sieci. W konsekwencji połączenia boczne powinny służyć do hamowania, a nie do wzbudzania. W tym kontekście funkcja delta Kroneckera służy do ułatwienia prostego zapisu. Funkcja delta Kroneckera ma dwa argumenty, które zwykle podawane są jako indeksy symbolu . Z definicji δ_{ik} ma wartość 1, jeśli i = k, oraz 0, jeśli i ≠ k. Oznacza to, że dwa indeksy powinny zgadzać się, aby delta Kroneckera miała wartość 1. W przeciwnym razie jej wartość wynosi 0.Odnosimy się do neuronów za pomocą dwóch indeksów, jeden dla miasta, do którego się odnosi, a drugi dla kolejności tego miasta w wycieczce. Dlatego element macierzy wag dla połączenia między dwoma neuronami musi mieć cztery indeksy, z przecinkiem po dwóch indeksach. Przykładem jest $w_{ik,lj}$ odnoszący się do wagi połączenia między neuronem (ik) a neuronem (lj). Wartość tej masy ustalana jest w następujący sposób:

$$W_{ik,lj} = -A_1\delta_{il}(1-\delta_{kj})-A_2\delta_{kj}(1-\delta_{kj}(1-\delta_{il})-A_3-A_4\, d_{il}(\delta_{j,\,k+1} + \delta_{j,k-1})$$

Tutaj negatywne znaki wskazują na hamowanie przez boczne połączenia w rzędzie lub kolumnie. $-A_3$ to określenie globalnego zahamowania.

Wejścia

Wejścia do sieci są wybierane arbitralnie. Jeśli w wyniku wyboru danych wejściowych aktywacje wyjdą na wynik, który sumuje się do liczby miast, powstanie wstępne rozwiązanie problemu, czyli legalna wycieczka. Może również pojawić się problem polegający na tym, że sieć utknie na lokalnym minimum. Aby uniknąć takiego zdarzenia, można dodać losowy szum. Zazwyczaj jako dane wejściowe w każdym

neuronie przyjmuje się stałą pomnożoną przez liczbę miast i dostosowujemy ją dodając liczbę losową, która może być różna dla różnych neuronów.

Aktywacje, wyjścia i ich aktualizacja

Aktywację neuronu w i-tym wierszu i j-tej kolumnie oznaczamy przez a_{ij}, a wyjście oznaczamy przez x_{ij}. Stosowane są również stała czasowa i wzmocnienie. Innym używanym parametrem jest stała m. Ponadto t oznacza przyrost w czasie od jednego cyklu do następnego. Należy pamiętać, że indeks sumowania waha się od 1 do n, czyli liczby miast. Wykluczone wartości indeksu są oznaczone symbolem ≠ Zmiana aktywacji jest wtedy podawana przez Δa_{ij}, gdzie:

$$\Delta a_{ij} = \Delta t\,(Term_1 + Term_2 + Term_3 + Term_4 + Term_5)$$
$$Term_1 = -\,a_{ij}/\tau$$
$$Term_2 = -\,A_1 \sum_{k \neq j} x_{ik}$$
$$Term_3 = -\,A_2 \sum_{k \neq i} x_{kj}$$
$$Term_4 = -\,A_3(\Sigma_i \Sigma_k x_{ik} - m)$$
$$Term_5 = -\,A_4 \sum_{k \neq i} d_{ik}(x_{k,j+1} + x_{k,j-1})$$

Aby zaktualizować aktywację neuronu w i-tym wierszu i j-tej kolumnie, bierzesz:

$a_{ijnew} = a_{ijold} + \Delta a_{ij}$.

Wyjście neuronu w i-tym wierszu i j-tej kolumnie jest obliczane z:

$$x_{in} = (1 + \tanh(\lambda a_{ij}))/2$$

Użytą tutaj funkcją jest hiperboliczna funkcja tangensa. Wspomniany wcześniej parametr wzmocnienia to λ. Wyjście każdego neuronu jest obliczane po aktualizacji aktywacji. Idealnie byłoby, gdyby dane wyjściowe były zerami i jedynkami, najlepiej pojedynczą dla każdego wiersza i każdej kolumny, aby reprezentować trasę, która spełnia warunki problemu. Ale funkcja tangensa hiperbolicznego daje liczbę rzeczywistą i musisz zadowolić się wartością wystarczająco bliską 1 lub 0. Możesz otrzymać na przykład 0,96 zamiast 1 lub 0,07 zamiast 0. Rozwiązanie należy uzyskać z takie wartości, zaokrąglając w górę lub w dół, tak, że zostanie użyte 1 lub 0, w zależności od przypadku.

Wydajność sieci Hopfield

Przyjrzyjmy się teraz podejściu Hopfielda i Tanka do rozwiązania TSP.

Przykład Hopfielda i Tanka

Wykorzystanie sieci Hopfielda w rozwiązaniu problemu komiwojażera jest pionierskim przedsięwzięciem w wykorzystaniu podejścia sieci neuronowych do tego problemu. Przykład Hopfielda i Tanka dotyczy problemu z 10 miastami. Zastosowano następujące parametry: A_1=500, A_2=500, A_3=200, A_4=500, τ = 1, λ = 50 i m=15. Dobre rozwiązanie odpowiadające lokalnemu minimum dla E jest oczekiwanym, jeśli nie najlepszym , rozwiązanie (minimum globalne). Można rozważyć proces wyżarzania, aby wyjść z jakiegokolwiek lokalnego minimum. Jak wspomniano wcześniej, problem komiwojażera jest jednym z tych problemów, dla których nie można znaleźć jednego podejścia, które będzie skuteczne we wszystkich przypadkach. Nie ma zbyt wielu wskazówek, jak ogólnie dobrać parametry do wykorzystania sieci Hopfield do rozwiązania problemu komiwojażera.

Implementacja C++ sieci Hopfield dla problemu komiwojażera

Przedstawiamy program C++ do obsługi sieci Hopfield dla problemu komiwojażera. Plik nagłówkowy znajduje się na listingu 1, a plik źródłowy na listingu 2. Klasa tsneuron jest zadeklarowana dla neuronu, a klasa sieci dla sieci. Klasa sieci jest zadeklarowana jako przyjaciel w klasie tsneuron. Program postępuje zgodnie z opisaną procedurą ustawiania wejść, wag połączeń i aktualizacji.

Szczegóły programu

Poniżej znajduje się lista cech programu C++ wraz z definicjami i/lub funkcjami.

• Liczba miast, liczba iteracji i odległości między miastami są pobierane od użytkownika.

• Odległości są traktowane jako wartości całkowite. Jeśli chcesz używać liczb rzeczywistych jako odległości, typ macierzy odległości musi zostać zmieniony na pływający, a odpowiednie zmiany są potrzebne dla funkcji calcdist ( ) itp.

• Tablice tourcity i tourorder śledzą miasta, które mają być objęte, i kolejność, w jakiej należy to zrobić.

• Neuron odpowiada każdej kombinacji miasta i jego kolejności w wycieczce. I-te odwiedzane miasto w kolejności j , to neuron odpowiadający elementowi j + i*n w tablicy dla neuronów. Tutaj n to liczba miast. i oraz j wahają się od 0 do n – 1. Jest n^2 neuronów.

• mtrx to macierz podająca wagi połączeń między neuronami. Jest to macierz kwadratowa rzędu n^2.

• Wektor wejściowy jest generowany losowo w funkcji main ( ) i jest później określany jako ip

• funkcja asgninpt ( ) prezentuje wektor wejściowy ip do sieci i określa początkowe aktywacje neuronów.

• Funkcja getacts ( ) aktualizuje aktywacje neuronów po każdej iteracji.

• funkcja getouts ( ) oblicza wyjścia po każdej iteracji. la jest używany jako skrót od lambda w swoim argumencie.

• funkcja iteracji ( ) wykonuje żądaną liczbę iteracji.

• Funkcja findtour ( ) określa kolejność zwiedzania miast, które mają zostać zwiedzone, korzystając z wyjść neuronów. Użyty na końcu iteracji daje rozwiązanie problemu komiwojażera.

• funkcja calcdist ( ) oblicza odległość trasy w rozwiązaniu.

Listing 1 Plik nagłówkowy programu C++ dla sieci Hopfield dla problemu komiwojażera

```
//trvslsmn.h V. Rao, H. Rao

#include <iostream.h>

#include <stdlib.h>

#include <math.h>

#include <stdio.h>

#define MXSIZ 11

class tsneuron

{
```

```
protected:
int cit,ord;
float output;
float activation;
friend class network;
public:
tsneuron() { };
void getnrn(int,int);
};
class network
{
public:
int citnbr;
float pra,prb,prc,prd,totout,distnce;
tsneuron (tnrn)[MXSIZ][MXSIZ];
int dist[MXSIZ][MXSIZ];
int tourcity[MXSIZ];
int tourorder[MXSIZ];
float outs[MXSIZ][MXSIZ];
float acts[MXSIZ][MXSIZ];
float mtrx[MXSIZ][MXSIZ];
float citouts[MXSIZ];
float ordouts[MXSIZ];
network() { };
void getnwk(int,float,float,float,float);
void getdist(int);
void findtour();
void asgninpt(float *);
void calcdist();
void iterate(int,int,float,float,float);
void getacts(int,float,float);
```

```
void getouts(float);

//print functions

void prdist();

void prmtrx(int);

void prtour();

void practs();

void prouts();

};
```

Plik źródłowy dla Hopfield Network dla problemu podróżującego sprzedawcy

Poniższy listing podaje kod źródłowy programu C++ dla sieci Hopfield dla problemu komiwojażera. Użytkownik jest proszony o wprowadzenie liczby miast i maksymalnej liczby iteracji dla działania sieci. Parametry a, b, c, d zadeklarowane w funkcji main odpowiadają odpowiednio A_1, A_2, A_3 i A_4. Te i parametry tau, lambda i nprm są wartościami podanymi w instrukcjach deklaracji w funkcji main. Jeśli zmienisz te wartości parametrów lub zmienisz liczbę miast w problemie komiwojażera, program się skompiluje, ale może nie działać tak, jak chcesz.

Listing 2 Plik źródłowy programu C++ dla sieci Hopfield dla problemu komiwojażera

```
//trvslsmn.cpp V. Rao, H. Rao

#include “trvslsmn.h”

#include <stdlib.h>

#include <time.h>

//generate random noise

int randomnum(int maxval)

{

// random number generator

// will return an integer up to maxval

return rand() % maxval;

}

//Kronecker delta function

int krondelt(int i,int j)

{

int k;

k= ((i == j) ? (1):(0));

return k;
```

```
};
void tsneuron::getnrn(int i,int j)
{
cit = i;
ord = j;
output = 0.0;
activation = 0.0;
};
//distances between cities
void network::getdist(int k)
{
citnbr = k;
int i,j;
cout<<"\n";
for(i=0;i<citnbr;++i)
{
dist[i][i]=0;
for(j=i+1;j<citnbr;++j)
{
cout<<"\ntype distance (integer) from city "<<
i<<" to city "<<j<<"\n";
cin>>dist[i][j];
}
cout<<"\n";
}
for(i=0;i<citnbr;++i)
{
for(j=0;j<i;++j)
{
dist[i][j] = dist[j][i];
}
```

```
}
prdist();
cout<<"\n";
}
//print distance matrix
void network::prdist()
{
int i,j;
cout<<"\n Distance Matrix\n";
for(i=0;i<citnbr;++i)
{
for(j=0;j<citnbr;++j)
{
cout<<dist[i][j]<<" ";
}
cout<<"\n";
}
}
//set up network
void network::getnwk(int citynum,float a,float b,float c,float d)
{
int i,j,k,l,t1,t2,t3,t4,t5,t6;
int p,q;
citnbr = citynum;
pra = a;
prb = b;
prc = c;
prd = d;
getdist(citnbr);
for(i=0;i<citnbr;++i)
{
```

```
for(j=0;j<citnbr;++j)
{
tnrn[i][j].getnrn(i,j);
}
}
//find weight matrix
for(i=0;i<citnbr;++i)
{
for(j=0;j<citnbr;++j)
{
p = ((j == citnbr-1) ? (0) : (j+1));
q = ((j == 0) ? (citnbr-1) : (j-1));
t1 = j + i*citnbr;
for(k=0;k<citnbr;++k)
{
for(l=0;l<citnbr;++l)
{
t2 = l + k*citnbr;
t3 = krondelt(i,k);
t4 = krondelt(j,l);
t5 = krondelt(l,p);
t6 = krondelt(l,q);
mtrx[t1][t2] =
-a*t3*(1-t4) -b*t4*(1-t3)
-c -d*dist[i][k]*(t5+t6);
}
}
}
}
prmtrx(citnbr);
}
```

```
//print weight matrix
void network::prmtrx(int k)
{
int i,j,nbrsq;
nbrsq = k*k;
cout<<"\nWeight Matrix\n";
for(i=0;i<nbrsq;++i)
{
for(j=0;j<nbrsq;++j)
{
if(j%k == 0)
{
cout<<"\n";
}
cout<<mtrx[i][j]<<" ";
}
cout<<"\n";
}
}
//present input to network
void network::asgninpt(float *ip)
{
int i,j,k,l,t1,t2;
for(i=0;i<citnbr;++i)
{
for(j =0;j<citnbr;++j)
{
acts[i][j] = 0.0;
}
}
//find initial activations
```

```
for(i=0;i<citnbr;++i)
{
for(j =0;j<citnbr;++j)
{
t1 = j + i*citnbr;
for(k=0;k<citnbr;++k)
{
for(l=0;l<citnbr;++l)
{
t2 = l + k*citnbr;
acts[i][j] +=
mtrx[t1][t2]*ip[t1];
}
}
}
}
//print activations
cout<<"\ninitial activations\n";
practs();
}
//find activations
void network::getacts(int nprm,float dlt,float tau)
{
int i,j,k,p,q;
float r1, r2, r3, r4,r5;
r3 = totout - nprm ;
for(i=0;i<citnbr;++i)
{
r4 = 0.0;
p = ((i == citnbr-1) ? (0) : (i+1));
q = ((i == 0) ? (citnbr-1) : (i-1));
```

```
for(j=0;j<citnbr;++j)
{
r1 = citouts[i] - outs[i][j];
r2 = ordouts[i] - outs[i][j];
for(k=0;k<citnbr;++k)
{
r4 += dist[i][k] *
(outs[k][p] + outs[k][q]);
}
r5 = dlt*(-acts[i][j]/tau -
pra*r1 -prb*r2 -prc*r3 -prd*r4);
acts[i][j] += r5;
}
}
}
//find outputs and totals for rows and columns
void network::getouts(float la)
{
float b1,b2,b3,b4;
int i,j;
totout = 0.0;
for(i=0;i<citnbr;++i)
{
citouts[i] = 0.0;
for(j=0;j<citnbr;++j)
{
b1 = la*acts[i][j];
b4 = b1/500.0;
b2 = exp(b4);
b3 = exp(-b4);
outs[i][j] = (1.0+(b2-b3)/(b2+b3))/2.0;
```

```
citouts[i] += outs[i][j];};
totout += citouts[i];
}
for(j=0;j<citnbr;++j)
{
ordouts[j] = 0.0;
for(i=0;i<citnbr;++i)
{
ordouts[j] += outs[i][j];
}
}
}
//find tour
void network::findtour()
{
int i,j,k,tag[MXSIZ][MXSIZ];
float tmp;
for (i=0;i<citnbr;++i)
{
for(j=0;j<citnbr;++j)
{
tag[i][j] = 0;
}
}
for (i=0;i<citnbr;++i)
{
tmp = -10.0;
for(j=0;j<citnbr;++j)
{
for(k=0;k<citnbr;++k)
{
```

```
if((outs[i][k] >=tmp)&&
(tag[i][k] ==0))
tmp = outs[i][k];
}
if((outs[i][j] ==tmp)&&
(tag[i][j]==0))
{
tourcity[i] =j;
tourorder[j] = i;
cout<<"\ntourcity "<<i
<<" tour order "<<j<<"\n";
for(k=0;k<citnbr;++k)
{
tag[i][k] = 1;
tag[k][j] = 1;
}
}
}
}
}
//print outputs
void network::prouts()
{
int i,j;
cout<<"\nthe outputs\n";
for(i=0;i<citnbr;++i)
{
for(j=0;j<citnbr;++j)
{
cout<<outs[i][j]<<" ";
}
```

```
cout<<"\n";
}
}
//calculate total distance for tour
void network::calcdist()
{
int i, k, l;
distnce = 0.0;
for(i=0;i<citnbr;++i)
{
k = tourorder[i];
l = ((i == citnbr-1 ) ? (tourorder[0]):(tourorder[i+1]));
distnce += dist[k][l];
}
cout<<"\n distance of tour is : "<<distnce<<"\n";
}
// print tour
void network::prtour()
{
int i;
cout<<"\n the tour :\n";
for(i=0;i<citnbr;++i)
{
cout<<tourorder[i]<<" ";
cout<<"\n";
}
}
//print activations
void network::practs()
{
int i,j;
```

```
cout<<"\n the activations:\n";
for(i=0;i<citnbr;++i)
{
for(j=0;j<citnbr;++j)
{
cout<<acts[i][j]<<" ";
}
cout<<"\n";
}
}
//iterate specified number of times
void network::iterate(int nit,int nprm,float dlt,float tau,float la)
{
int k;
for(k=1;k<=nit;++k)
{
getacts(nprm,dlt,tau);
getouts(la);
}
cout<<"\n" <<nit<<" iterations completed\n";
practs();
cout<<"\n";
prouts();
cout<<"\n";
}
void main()
{
//numit = #iterations; n = #cities; u=intial input; nprm - parameter n'
//dt = delta t;
// ——————————————————-
// parameters to be tuned are here:
```

```
int u=1;
int nprm=10;
float a=40.0;
float b=40.0;
float c=30.0;
float d=60.0;
float dt=0.01;
float tau=1.0;
float lambda=3.0;
//————————————————————-
int i,n2;
int numit=100;
int n=4;
float input_vector[MXSIZ*MXSIZ];
srand ((unsigned)time(NULL));
cout<<"\nPlease type number of cities, number of iterations\n";
cin>>n>>numit;
cout<<"\n";
if (n>MXSIZ)
{
cout << "choose a smaller n value\n";
exit(1);
}
n2 = n*n;
for(i=0;i<n2;++i)
{
if(i%n == 0)cout<<"\n";
input_vector[i] =(u + (float)(randomnum(100)/100.0))/20.0;
cout<<input_vector[i]<<" ";
}
//create network and operate
```

```
network *tntwk = new network;

if (tntwk==0)

{

cout << “not enough memory\n”;

exit(1);

}

tntwk->getnwk(n,a,b,c,d);

tntwk->asgninpt(input_vector);

tntwk->getouts(lambda);

tntwk->prouts();

tntwk->iterate(numit,nprm,dt,tau,lambda);

tntwk->findtour();

tntwk->prtour();

tntwk->calcdist();

cout<<”\n”;
```

Dane wyjściowe z Twojego programu w języku C++ dotyczące problemu z podróżującym sprzedawcą

Problem z wycieczką po trzech miastach jest trywialny, ponieważ istnieje tylko jedna wartość dla całkowitej odległości, bez względu na to, jak ustawisz miasta w kolejności zwiedzania. W tym przypadku naturalny porządek jest sam w sobie optymalnym rozwiązaniem. Dla ilustracji program jest uruchamiany w dwóch przypadkach. Pierwszy bieg dotyczy problemu z czterema miastami. Drugi dotyczy problemu pięciu miast. Nawiasem mówiąc, miasta są ponumerowane od 0 do n – 1. Te same wartości parametrów są używane w dwóch przejazdach. Różna była liczba miast, a co za tym idzie macierz odległości. W pierwszym przebiegu żądana liczba iteracji wynosi 30, aw drugim — 40. Rozwiązanie problemu czterech miast nie jest takie w naturalnym porządku. Całkowita odległość trasy wynosi 32. Trasa w rozwiązaniu to 1 - 0 - 3 - 2 - 1. Ta trasa jest odpowiednikiem trasy w naturalnym porządku, jak widać, zaczynając od 0 i czytając cyklicznie od prawej do lewej w poprzedniej sekwencji miast. Rozwiązaniem problemu pięciu miast jest trasa 1 - 2 - 0 - 4 - 3 - 1. To brzmi, zaczynając od 0 jako 0 - 4 - 3 - 1 - 2 - 0 lub jako 0 - 2 - 1 - 3 - 4 - 0. Różni się od trasy w naturalnym porządku. Ma całkowitą odległość 73, w porównaniu do odległości 84 z porządkiem naturalnym. Nie jest to jednak optymalne, ponieważ trasa o najkrótszym dystansie wynosi 0 - 4 - 2 - 1 - 3 - 0, a całkowity dystans wynosi 50. Czy program może zaoferować krótszą trasę? Tak, rozwiązanie może być inne, jeśli uruchomisz program ponownie, ponieważ za każdym razem, gdy uruchamiasz program, wektor wejściowy jest inny, ponieważ jest wybierany losowo. Wartości parametrów podane w programie są zgadywane. Zauważ, że ziarno generatora liczb losowych jest podane w instrukcji

```
srand ((unsigned)time(NULL));
```

Program podaje porządek w wycieczce dla każdego miasta. Na przykład, jeśli jest napisane tourcity 1 kolejność zwiedzania 2, oznacza to, że drugie miasto (turystyka) jest miastem odwiedzanym jako trzecie (w kolejności zwiedzania). Twoje zamówienia na wycieczki mają również wartości od 0 do n - 1,

podobnie jak miasta. Dane wejściowe użytkownika są napisane kursywą, a dane wyjściowe komputera są normalne, jak widzieliście wcześniej.

Dane wyjściowe dla problemu czterech miast

Please type number of cities, number of iterations

4 30

0.097 0.0585 0.053 0.078 //input vector—there

are 16 neurons in the network

0.0725 0.0535 0.0585 0.0985

0.0505 0.061 0.0735 0.057

0.0555 0.075 0.0885 0.0925

type distance (integer) from city 0 to city 1

10

type distance (integer) from city 0 to city 2

14

type distance (integer) from city 0 to city 3

7

type distance (integer) from city 1 to city 2

6

type distance (integer) from city 1 to city 3

12

type distance (integer) from city 2 to city 3

9

Distance Matrix

0 10 14 7

10 0 6 12

14 6 0 9

7 12 9 0

Weight Matrix //16x16 matrix of weights. There are

16 neurons in the network.

-30 -70 -70 -70

-70 -630 -30 -630

-70 -870 -30 -70

-30 -70 -70 -630

-70 -30 -70 -70

-630 -70 -630 -30

-870 -70 -870 -70

-70 -30 -70 -30

-70 -70 -30 -70

-30 -630 -70 -630

-30 -870 -70 -70

-70 -70 -30 -630

-70 -70 -70 -30

-630 -30 -630 -70

-870 -30 -870 -70

-630 -30 -630 -30

-70 -630 -30 -630

-30 -70 -70 -70

-70 -390 -30 -630

-70 -630 -30 -70

-630 -70 -630 -30

-70 -30 -70 -70

-390 -70 -390 -30

-630 -70 -630 -70

-30 -630 -70 -630

-70 -70 -30 -70

-30 -390 -70 -630

-30 -630 -70 -70

-630 -30 -630 -70

-70 -70 -70 -30

-390 -30 -390 -70

-870 -30 -870 -70

-70 -870 -30 -870

-70 -390 -30 -390

-30 -70 -70 -870

-70 -870 -30 -390

-870 -70 -870 -30

-390 -70 -390 -30

-70 -30 -70 -30

-870 -70 -870 -30

-30 -870 -70 -870

-30 -390 -70 -390

-70 -70 -30 -870

-30 -870 -70 -390

-870 -30 -870 -70

-390 -30 -390 -70

-70 -70 -70 -70

-450 -30 -450 -70

-70 -450 -30 -450

-70 -750 -30 -750

-70 -570 -30 -450

-70 -450 -30 -750

-450 -70 -450 -30

-750 -70 -750 -30

-570 -70 -570 -30

-450 -70 -450 -30

-30 -450 -70 -450

-30 -750 -70 -750

-30 -570 -70 -450

-30 -450 -70 -750

-450 -30 -450 -70

-750 -30 -750 -70

-570 -30 -570 -70

-70 -70 -70 -30

initial activations

the activations:

-333.680054 -215.280014 -182.320023 -371.280029

-255.199997 -207.580002 -205.920013 -425.519989

-258.560028 -290.360016 -376.320007 -228.000031

-278.609985 -363 -444.27005 -377.400024

the outputs

0.017913 0.070217 0.100848 0.011483

0.044685 0.076494 0.077913 0.006022

0.042995 0.029762 0.010816 0.060882

0.034115 0.012667 0.004815 0.010678

30 iterations completed

the activations:

-222.586884 -176.979172 -195.530823 -380.166107

-164.0271 -171.654053 -214.053177 -421.249023

-158.297867 -218.833755 -319.384827 -245.097473

-194.550751 -317.505554 -437.527283 -447.651581

the outputs

0.064704 0.10681 0.087355 0.010333

0.122569 0.113061 0.071184 0.006337

0.130157 0.067483 0.021194 0.050156

0.088297 0.021667 0.005218 0.004624

tourcity 0 tour order 1

tourcity 1 tour order 0

tourcity 2 tour order 3

tourcity 3 tour order 2

the tour :

1032

distance of tour is : 32

Dane wyjściowe dla problemu pięciu miast

Please type number of cities, number of iterations

5 40

0.0645 0.069 0.0595 0.0615 0.0825 //input vector—there are 25

neurons in the network

0.074 0.0865 0.056 0.095 0.06

0.0625 0.0685 0.099 0.0645 0.0615

0.056 0.099 0.065 0.093 0.051

0.0675 0.094 0.0595 0.0635 0.0515

type distance (integer) from city 0 to city 1

10

type distance (integer) from city 0 to city 2

14

type distance (integer) from city 0 to city 3

7

type distance (integer) from city 0 to city 4

6

type distance (integer) from city 1 to city 2

12

type distance (integer) from city 1 to city 3

9

type distance (integer) from city 1 to city 4

18

type distance (integer) from city 2 to city 3

24

type distance (integer) from city 2 to city 4

16

type distance (integer) from city 3 to city 4

32

Distance Matrix

0 10 14 7 6

10 0 12 9 18

14 12 0 24 16

7 9 24 0 32

6 18 16 32 0

Weight Matrix //25x25 matrix of weights. There are 25 neurons

in the network.

-30 -70 -70 -70 -70

-70 -630 -30 -30 -630

-70 -70 -30 -70 -70

-70 -630 -70 -630 -30

-30 -870 -70 -70 -30

-70 -30 -70 -70 -70

-630 -70 -630 -30 -30

-870 -70 -70 -30 -70

-70 -30 -630 -70 -630

-30 -30 -70 -70 -70

-70 -70 -30 -70 -70

-30 -630 -70 -630 -30

-30 -70 -70 -70 -30

-70 -30 -30 -630 -70

-630 -30 -70 -70 -70

-70 -70 -70 -30 -70

-30 -30 -630 -70 -630

-30 -70 -70 -70 -70

-30 -630 -30 -30 -630

-70 -870 -70 -630 -30

-70 -70 -70 -70 -30

-630 -30 -30 -630 -70

-870 -70 -630 -30 -30

-630 -30 -70 -70 -70

-70 -70 -630 -70 -630

-70 -630 -30 -30 -630

-30 -70 -70 -70 -70

-70 -630 -70 -630 -30

-30 -70 -30 -70 -70

-70 -750 -30 -630 -70

-630 -70 -630 -30 -30

-70 -30 -70 -70 -70

-750 -30 -630 -70 -630

-30 -70 -70 -30 -70

-70 -30 -30 -30 -630

-30 -630 -70 -630 -30

-70 -70 -30 -70 -70

-30 -30 -30 -630 -70

-630 -70 -70 -70 -30

-70 -30 -630 -30 -30

-30 -30 -630 -70 -630

-70 -70 -70 -30 -70

-30 -630 -30 -30 -630

-70 -70 -70 -70 -70

-30 -750 -70 -870 -30

-630 -30 -30 -630 -70

-70 -70 -70 -70 -30

-750 -70 -870 -30 -30

-870 -70 -750 -30 -30

-750 -30 -870 -70 -870

-70 -870 -30 -30 -870

-70 -750 -30 -30 -750

-30 -870 -70 -870 -30

-30 -750 -70 -750 -30

-30 -70 -30 -870 -70

-870 -70 -870 -30 -30

-750 -70 -750 -30 -30

-70 -30 -870 -70 -870

-30 -30 -750 -70 -750

-30 -70 -30 -30 -870

-30 -870 -70 -870 -30

-30 -750 -70 -750 -30

-70 -30 -30 -870 -70

-870 -30 -30 -750 -70

-750 -70 -870 -30 -30

-30 -30 -870 -70 -870

-30 -30 -750 -70 -750

-70 -870 -30 -30 -870

-70 -750 -30 -30 -750

-70 -70 -70 -450 -30

-870 -30 -30 -870 -70

-750 -30 -30 -750 -70

-70 -70 -450 -30 -30

-450 -70 -570 -30 -30

-570 -70 -450 -70 -450

-70 -450 -30 -30 -450

-70 -570 -30 -30 -570

-70 -450 -70 -450 -30

-30 -570 -70 -570 -30

-30 -1470 -30 -450 -70

-450 -70 -450 -30 -30

-570 -70 -570 -30 -30

-1470 -30 -450 -70 -450

-30 -30 -570 -70 -570

-30 -30 -30 -30 -450

-30 -450 -70 -450 -30

-30 -570 -70 -570 -30

-30 -30 -30 -450 -70

-450 -30 -30 -570 -70

-570 -30 -450 -30 -30

-30 -30 -450 -70 -450

-30 -30 -570 -70 -570

-30 -450 -30 -30 -450

-70 -570 -30 -30 -570

-70 -1470 -70 -390 -30

-450 -30 -30 -450 -70

-570 -30 -30 -570 -70

-1470 -70 -390 -30 -30

-390 -70 -1110 -30 -30

-1110 -70 -390 -70 -390

-70 -390 -30 -30 -390

-70 -1110 -30 -30 -1110

-70 -390 -70 -390 -30

-30 -1110 -70 -1110 -30

-30 -990 -30 -390 -70

-390 -70 -390 -30 -30

-1110 -70 -1110 -30 -30

-990 -30 -390 -70 -390

-30 -30 -1110 -70 -1110

-30 -30 -30 -30 -390

-30 -390 -70 -390 -30

-30 -1110 -70 -1110 -30

-30 -30 -30 -390 -70

-390 -30 -30 -1110 -70

-1110 -30 -390 -30 -30

-30 -30 -390 -70 -390

-30 -30 -1110 -70 -1110

-30 -390 -30 -30 -390

-70 -1110 -30 -30 -1110

-70 -990 -30 -30 -990

-390 -30 -30 -390 -70

-1110 -30 -30 -1110 -70

-990 -30 -30 -990 -70

-1950 -30 -30 -1950 -70

-70 -70 -70 -70 -30

initial activations

the activations:

-290.894989 -311.190002 -218.365005 -309.344971 -467.774994

-366.299957 -421.254944 -232.399963 -489.249969 -467.399994

-504.375 -552.794983 -798.929871 -496.005005 -424.964935

-374.639984 -654.389832 -336.049988 -612.870056 -405.450012

-544.724976 -751.060059 -418.285034 -545.465027 -500.065063

the outputs

0.029577 0.023333 0.067838 0.023843 0.003636

0.012181 0.006337 0.057932 0.002812 0.003652

0.002346 0.001314 6.859939e-05 0.002594 0.006062

0.011034 0.000389 0.017419 0.000639 0.00765

0.001447 0.000122 0.006565 0.001434 0.002471

40 iterations completed

the activations:

-117.115494 -140.58519 -85.636215 -158.240143 -275.021301

-229.135956 -341.123871 -288.208496 -536.142212 -596.154297

-297.832794 -379.722595 -593.842102 -440.377625 -442.091064

-209.226883 -447.291016 -283.609589 -519.441101 -430.469696

-338.93219 -543.509766 -386.950531 -538.633606 -574.604492

the outputs

0.196963 0.156168 0.263543 0.130235 0.035562

0.060107 0.016407 0.030516 0.001604 0.000781

0.027279 0.010388 0.000803 0.005044 0.004942

0.07511 0.004644 0.032192 0.001959 0.005677

0.016837 0.001468 0.009533 0.001557 0.001012

tourcity 0 tour order 2

tourcity 1 tour order 0

tourcity 2 tour order 1

tourcity 3 tour order 4

tourcity 4 tour order 3

the tour :

12043

distance of tour is : 73

Inne podejścia do rozwiązania problemu komiwojażera

Poniżej opisano kilka innych metod rozwiązania problemu komiwojażera.

Prezentacja Anzai

Yuichiro Anzai w nieco inny sposób opisuje sieć Hopfield dotyczącą problemu komiwojażera. Po pierwsze, nie jest używany termin globalnego hamowania. Wartość progowa jest powiązana z każdym neuronem, dodawana do aktywacji i przyjmowana jako średnia z A_1 i A_2 według naszego wcześniejszego zapisu. Funkcja energii jest sformułowana nieco inaczej, w następujący sposób:

$$E = A_1 \Sigma_l(\Sigma_k x_{lk}-1)^2 + A_2 \Sigma_i(\Sigma_k x_{ki}-1)^2 + A_4 \Sigma_k \Sigma_{j\neq k} \Sigma_{i\neq k} d_{kj} x_{ki}(x_{j,i+1} + x_{j,i-1})$$

Pierwszy termin to 0, jeśli suma wyników wynosi 1 w każdej kolumnie. To samo dotyczy drugiego terminu w odniesieniu do wierszy. Wyjście jest obliczane za pomocą parametru zwanego tutaj referencyjnym poziomem aktywacji, jako:

$$x_{ij} = (1 + \tan \tanh(a_{ij}/\lambda))/2$$

Użyte parametry to $A_1 = 1/2$, $A_2 = 1/2$, $A_4 = 1/2$, $\Delta t = 1$, $\tau = 1$ i $\lambda = 1$. Podjęto próbę rozwiązania problemu dla wycieczki po 10 miastach. Otrzymane rozwiązanie nie jest wyraźne, w tym sensie, że dokładnie jeden 1 występuje w każdym rzędzie i każdej kolumnie, istnieją wartości o różnej wielkości z jedną dominującą wartością w każdym rzędzie i kolumnie. Wybitna wartość jest uważana za część rozwiązania.

Podejście Kohonena do problemu komiwojażera

Samoorganizujące się mapy Kohonena można wykorzystać do rozwiązania problemu komiwojażera. Podsumowujemy omówienie tego podejścia opisanego w pracy Erica Davalo i Patricka Naima. Do każdego miasta branego pod uwagę podczas wycieczki odwołują się jego współrzędne x i y. Każdemu miastu odpowiada neuron. Neurony są umieszczone w jednej tablicy, w przeciwieństwie do tablicy dwuwymiarowej stosowanej w podejściu Hopfielda. Neurony pierwszy i ostatni w tablicy są uważane za sąsiadów. Dla każdego neuronu istnieje wektor wagowy, który również składa się z dwóch składników. Wektor wagowy neuronu to obraz neuronu na mapie, którego celem jest samoorganizacja. Jest tyle wektorów wejściowych, ile jest miast, a para współrzędnych miasta stanowi wektor

wejściowy. Wybierany jest neuron o wektorze wagowym najbliższym wektorowi wejściowemu. Wagi neuronów w sąsiedztwie wybranego neuronu są modyfikowane, inne nie. Stopniowo zmniejszający się współczynnik skali jest również używany do modyfikacji odważników. Jeden neuron jest tworzony jako pierwszy, a jego wektor wagowy ma 0 dla jego składowych. Inne neurony są tworzone pojedynczo, w każdej iteracji uczenia się. Neurony również mogą zostać zniszczone. Utworzenie neuronu i zniszczenie neuronu sugeruje tymczasowe dodanie miasta do końcowej listy w wycieczce i tymczasowe usunięcie miasta z tej listy. W ten sposób zapobiega się możliwości przypisania dowolnego neuronu do dwóch wejść lub dwóch miast. To samo dotyczy przypisania dwóch neuronów do tego samego wejścia. Ponieważ wektory wejściowe są prezentowane sieci, jeśli niewybrany neuron dwukrotnie znajdzie się w sąsiedztwie najbliższego, zostanie utworzony. Jeżeli utworzony neuron nie zostanie wybrany w trzech kolejnych iteracjach do modyfikacji wag, wraz z modyfikacją innych, zostanie zniszczony. To, że trasa o najkrótszej odległości wynika z działania tej sieci, wynika z faktu, że wybierane są najbliższe neurony. Poinformowano, że wyniki eksperymentów są bardzo obiecujące. Czas obliczeń jest mały i uzyskuje się rozwiązania nieco zbliżone do optymalnego, jeśli nie samo rozwiązanie optymalne. Tak jak wcześniej w przypadku problemu komiwojażera, jest to problem NP-zupełny i należy zaakceptować prawie efektywne (prowadzące do nieoptymalnych rozwiązań, ale szybsze) podejście do niego.

Algorytm dla podejścia Kohonena

Parametr wzmocnienia i współczynnik skali q są używane podczas modyfikowania wag. W poprzednich przykładach próbowano uzyskać wartość między 0,02 a 0,2 dla q. Odległość neuronu od wybranego neuronu jest definiowana jako liczba całkowita z zakresu od 0 do n – 1, gdzie n to liczba miast objętych wycieczką. Oznacza to, że odległości te niekoniecznie są rzeczywistymi odległościami między miastami. Mogłyby być w jakiś sposób reprezentatywne dla rzeczywistych odległości. Jedna z takich prób jest opisana w poniższej dyskusji na temat implementacji C++. Odległość ta jest oznaczona przez dj dla neuronu j. Wykorzystywana jest również funkcja zgniatania podobna do funkcji gęstości Gaussa. Szczegóły algorytmu znajdują się w artykule cytowanym w Davalo. Kroki algorytmu w zakresie podanym przez Davalo to:

• Znajdź wektor wag, dla którego odległość od wektora wejściowego jest najmniejsza

• Zmodyfikuj wagi za pomocą

$wj_{new} = w_{jold} + (l_{new} - wj_{old})g(\lambda, d_j)$, gdzie $g(\lambda, d_j) = \exp(- dj^2/\lambda) / \sqrt{}$

• Zresetuj λ jako λ (1 - q)

Implementacja C++ podejścia Kohonena

Nasza implementacja C++ tego algorytmu (opisana powyżej) jest z niewielkimi modyfikacjami. Tworzymy, ale nie niszczymy wyraźnie neuronów. Oznacza to, że nie liczymy liczby kolejnych iteracji, w których neuron nie został wybrany do modyfikacji wag. To konsekwencja tego, że nie zdefiniowaliśmy sąsiedztwa neuronu. Nasz przykład dotyczy problemu z pięcioma neuronami, ilustracją, a ze względu na małą liczbę zaangażowanych neuronów, cały zestaw jest uważany za sąsiedztwo każdego neuronu. Po utworzeniu wszystkich neuronów z wyjątkiem jednego, pozostały neuron jest tworzony bez dalszej pracy z algorytmem i przypisywany do wejścia, które nie jest jeszcze przypisane do neuronu. Po utworzeniu n - 1 neuronów powinno pozostać tylko jedno nieprzypisane wejście. W naszej implementacji C++ macierz odległości dla odległości między neuronami, w naszym przykładzie, jest podana w następujący sposób, zgodnie z założeniem algorytmu, że wartości te powinny być liczbami całkowitymi z zakresu od 0 do n – 1.

```
d =
     0  1  2  3  4
     1  0  1  2  3
     2  1  0  1  2
     3  2  1  0  1
     4  3  2  1  0
```

Uruchomiliśmy również program, zastępując poprzednią macierz następną macierzą i uzyskaliśmy to samo rozwiązanie. Rzeczywiste odległości między miastami są mniej więcej cztery razy większe od odpowiadających im wartości w tej macierzy. Nie uwzględniliśmy danych wyjściowych z drugiego uruchomienia programu.

```
d =
     0  1  3  3  2
     1  0  3  2  1
     3  3  0  4  2
     3  2  4  0  1
     2  1  2  1  0
```

W naszej implementacji wybraliśmy funkcję podobną do funkcji gęstości Gaussa jako funkcję zgniatania. Stosowana funkcja zgniatania to:

$$f(d,\lambda) = \exp(-d^2/\lambda) / \sqrt{(2)}$$

Plik nagłówkowy programu C++ dla podejścia Kohonena

Listing 3 zawiera plik nagłówkowy tego programu, a listing 4 zawiera odpowiedni plik źródłowy:

Listing 3 Plik nagłówkowy programu C++ dla podejścia Kohonena

```
//tsp_kohn.h V.Rao, H.Rao

#include<iostream.h>

#include<math.h>

#define MXSIZ 10

#define pi 3.141592654

class city_neuron

{

protected:

double x,y;

int mark,order,count;

double weight[2];

friend class tspnetwork;

public:

city_neuron(){};
```

```
void get_neuron(double,double);
};
class tspnetwork
{
protected:
int chosen_city,order[MXSIZ];
double gain,input[MXSIZ][2];
int citycount,index,d[MXSIZ][MXSIZ];
double gain_factor,diffsq[MXSIZ];
city_neuron (cnrn)[MXSIZ];
public:
tspnetwork(int,double,double,double,double*,double*);
void get_input(double*,double*);
void get_d();
void find_tour();
void associate_city();
void modify_weights(int,int);
double wtchange(int,int,double,double);
void print_d();
void print_input();
void print_weights();
void print_tour();
};
```

Listing 4 Plik źródłowy programu C++ dla podejścia Kohonena

```
//tsp_kohn.cpp V.Rao, H.Rao
#include “tsp_kohn.h”
void city_neuron::get_neuron(double a,double b)
{
x = a;
y = b;
mark = 0;
```

```
count = 0;
weight[0] = 0.0;
weight[1] = 0.0;
};
tspnetwork::tspnetwork(int k,double f,double q,double h,
double *ip0,double *ip1)
{
int i;
gain = h;
gain_factor = f;
citycount = k;
// distances between neurons as integers between 0 and n-1
get_d();
print_d();
cout<<"\n";
// input vectors
get_input(ip0,ip1);
print_input();
// neurons in the network
for(i=0;i<citycount;++i)
{
order[i] = citycount+1;
diffsq[i] = q;
cnrn[i].get_neuron(ip0[i],ip1[i]);
cnrn[i].order = citycount +1;
}
}
void tspnetwork::associate_city()
{
int i,k,j,u;
double r,s;
```

```
for(u=0;u<citycount;u++)
{
//start a new iteration with the input vectors
for(j=0;j<citycount;j++)
{
for(i=0;i<citycount;++i)
{
if(cnrn[i].mark==0)
{
k = i;
i =citycount;
}
}
//find the closest neuron
for(i=0;i<citycount;++i)
{
r = input[j][0] - cnrn[i].weight[0];
s = input[j][1] - cnrn[i].weight[1];
diffsq[i] = r*r +s*s;
if(diffsq[i]<diffsq[k]) k=i;
}
chosen_city = k;
cnrn[k].count++;
if((cnrn[k].mark<1)&&(cnrn[k].count==2))
{
//associate a neuron with a position
cnrn[k].mark = 1;
cnrn[k].order = u;
order[u] = chosen_city;
index = j;
gain *= gain_factor;
```

```
//modify weights
modify_weights(k,index);
print_weights();
j = citycount;
}
}
}
}
void tspnetwork::find_tour()
{
int i;
for(i=0;i<citycount;++i)
{
associate_city();
}
//associate the last neuron with remaining position in
// tour
for(i=0;i<citycount;++i)
{
if( cnrn[i].mark ==0)
{
cnrn[i].order = citycount-1;
order[citycount-1] = i;
cnrn[i].mark = 1;
}
}
//print out the tour.
//First the neurons in the tour order
//Next cities in the tour
//order with their x,y coordinates
print_tour();
```

```
}
void tspnetwork::get_input(double *p,double *q)
{
int i;
for(i=0;i<citycount;++i)
{
input[i][0] = p[i];
input[i][1] = q[i];
}
}
//function to compute distances (between 0 and n-1) between
//neurons
void tspnetwork::get_d()
{
int i,j;
for(i=0;i<citycount;++i)
{
for(j=0;j<citycount;++j)
{
d[i][j] = (j-i);
if(d[i][j]<0) d[i][j] = d[j][i];
}
}
}
//function to find the change in weight component
double tspnetwork::wtchange(int m,int l,double g,double h)
{
double r;
r = exp(-d[m][l]*d[m][l]/gain);
r *= (g-h)/sqrt(2*pi);
return r;
```

```
}
//function to determine new weights
void tspnetwork::modify_weights(int jj,int j)
{
int i;
double t;
double w[2];
for(i=0;i<citycount;++i)
{
w[0] = cnrn[i].weight[0];
w[1] = cnrn[i].weight[1];
//determine new first component of weight
t = wtchange(jj,i,input[j][0],w[0]);
w[0] = cnrn[i].weight[0] +t;
cnrn[i].weight[0] = w[0];
//determine new second component of weight
t = wtchange(jj,i,input[j][1],w[1]);
w[1] = cnrn[i].weight[1] +t;
cnrn[i].weight[1] = w[1];
}
}
//different print routines
void tspnetwork::print_d()
{
int i,j;
cout<<"\n";
for(i=0;i<citycount;i++)
{
cout<<" d: ";
for(j=0;j<citycount;j++)
{
```

```
cout<<d[i][j]<<" ";
}
cout<<"\n";
}
}
void tspnetwork::print_input()
{
int i,j;
for(i=0;i<citycount;i++)
{
cout<<"input : ";
for(j=0;j<2;j++)
{
cout<<input [i][j]<<" ";
}
cout<<"\n";
}
}
void tspnetwork::print_weights()
{
int i,j;
cout<<"\n";
for(i=0;i<citycount;i++)
{
cout<<" weight: ";
for(j=0;j<2;j++)
{
cout<<cnrn[i].weight[j]<<" ";
}
cout<<"\n";
}
```

```
}
void tspnetwork::print_tour()
{
int i,j;
cout<<"\n tour : ";
for(i=0;i<citycount;++i)
{
cout<<order[i]<<" —> ";
}
cout<<order[0]<<"\n\n";
for(i=0;i<citycount;++i)
{
j = order[i];
cout<<"("<<cnrn[j].x<<", "<<cnrn[j].y<<") —> ";
}
j= order[0];
cout<<"("<<cnrn[j].x<<", "<<cnrn[j].y<<")\n\n";
}
void main()
{
int nc= 5;//nc = number of cities
double q= 0.05,h= 1.0,p= 1000.0;
double input2[][5]= {7.0,4.0,14.0,0.0,5.0,3.0,6.0,13.0,12.0,10.0};
tspnetwork tspn2(nc,q,p,h,input2[0],input2[1]);
tspn2.find_tour();
}
```

Wyjście z przykładowego przebiegu programu

Program, jak wspomniano, jest stworzony z myślą o podejściu Kohonena do problemu komiwojażera dla pięciu miast. Brak danych wejściowych użytkownika z klawiatury. Wszystkie wartości parametrów są podawane do programu z odpowiednimi poleceniami w funkcji main. Podano współczynnik skali 0,05 do zastosowania do parametru wzmocnienia, który jest podany jako 1. Początkowo odległość każdego wektora wagi neuronu od wektora wejściowego jest ustawiona na 1000, aby ułatwić znalezienie najbliższego za pierwszym razem. Dla wektorów wejściowych określone są miasta o

współrzędnych (7,3), (4,6), (14,13), (0,12), (5,10). Znaleziona trasa nie jest trasą w naturalnym porządku, czyli 0 1 2 3 4 0, z odległością 43,16. Znaleziona trasa ma kolejność 0 3 1 4 2 0, która obejmuje dystans 44,43, czyli nieco wyższy, jak pokazano na rysunku 15.2. Najlepsza trasa 0 2 4 3 1 0 ma łączny dystans 38,54.

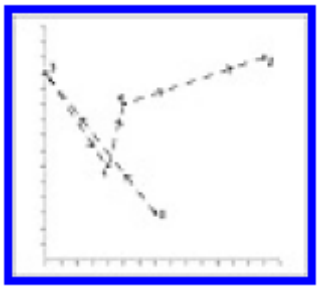


Tabela podaje dla przykładu z pięcioma miastami 12 (5!/10) odrębnych odległości objazdów i odpowiadające im objazdy reprezentatywne. Nie są one generowane przez program, ale przez ręczne wyliczanie i obliczanie. Ta tabela jest dostępna tutaj, aby zobaczyć różne rozwiązania tego przykładu problemu sprzedawcy w pięciu miastach.

Odległości i reprezentatywne trasy dla przykładu pięciu miast

Distance	Tour	Comment
49.05	0-3-2-1-4-0	worst case
47.59	0-3-1-2-4-0	
45.33	0-2-1-4-3-0	
44.86	0-2-3-1-4-0	
44.43	0-3-1-4-2-0	tour given by the program
44.30	0-2-1-3-4-0	
43.29	0-1-4-2-3-0	
43.16	0-1-2-3-4-0	
42.73	0-1-2-4-3-0	
42.26	0-1-3-2-4-0	
40.00	0-1-4-3-2-0	
38.54	0-2-4-3-1-0	optimal tour

Istnieje 12 różnych odległości, które można uzyskać na wycieczki z tymi miastami, obliczając je ręcznie, a cztery z nich są wyższe, a siedem niższych niż ta, którą znajdziesz w tym programie. W najgorszym przypadku trasa (0 [rarr] 3 [rarr] 2 [rarr] 1 [rarr] 4 [rarr] 0) daje dystans 49,05, a najlepszy, jak widzieliście powyżej, 38,54. Rozwiązanie z programu znajduje się mniej więcej w połowie najlepszych i najgorszych pod względem całkowitej przebytej odległości. Wynik programu, który jest w całości wygenerowany przez komputer, jest podany poniżej w następujący sposób:

```
d: 0   1   2   3   4
d: 1   0   1   2   3
d: 2   1   0   1   2
d: 3   2   1   0   1
d: 4   3   2   1   0

input : 7   3
input : 4   6
input : 14  13
input : 0   12
input : 5   10

weight: 1.595769   2.393654


weight: 3.289125e-09   4.933688e-09
weight: 2.880126e-35   4.320189e-35
weight: 1.071429e-78   1.607143e-78
weight: 1.693308e-139  2.539961e-139

weight: 1.595769   2.393654
weight: 5.585192   5.18625
weight: 2.880126e-35   4.320189e-35
weight: 1.071429e-78   1.607143e-78
weight: 1.693308e-139  2.539961e-139

weight: 1.595769   2.393654
weight: 5.585192   5.18625
weight: 5.585192   5.18625
weight: 1.071429e-78   1.607143e-78
weight: 1.693308e-139  2.539961e-139

weight: 1.595769   2.393654
weight: 5.585192   5.18625
weight: 5.585192   5.18625
weight: 5.585192   5.18625
weight: 1.693308e-139   2.539961e-139

weight: 1.595769   2.393654
weight: 5.585192   5.18625
weight: 5.585192   5.18625
weight: 5.585192   5.18625
weight: 5.585192   5.18625

tour : 0 -> 3-> 1 -> 4 -> 2-> 0

(7, 3) -> (0, 12) -> (4, 6) -> (5, 10) -> (14, 13) -> (7, 3)
```

Optymalizacja portfela akcji

Rozwój sieci neuronowej w procesie selekcji akcji w obrocie papierami wartościowymi jest podobny do zastosowania sieci neuronowych do nieliniowych problemów optymalizacji. Podstawą takiego rozwoju jest przełomowa praca Markowitza w matematycznym sformułowaniu funkcji celu w kontekście doboru portfolio. Istnieje ryzyko, które należy zminimalizować lub nałożyć pułap, a zyski należy zmaksymalizować. Kapitał inwestycyjny jest naturalnie ograniczonym zasobem.

Funkcja celu jest sformułowana w taki sposób, aby optymalny portfel minimalizował funkcję celu. W funkcji celu byłby termin obejmujący iloczyn każdej pary cen akcji. Kowariancja tej pary cen jest również wykorzystywana w funkcji celu. Produkt spełnia funkcję celu kwadratowego. Oczywiście istniałyby również pewne terminy liniowe i reprezentują one indywidualne ceny akcji ze średnim zwrotem akcji jako współczynnikiem w każdym takim okresie. Już wiesz, że ten problem optymalizacji należy do kategorii problemów programowania kwadratowego, które dają w wyniku wartości liczb rzeczywistych dla zmiennych w optymalnym rozwiązaniu. Niektóre inne terminy byłyby również zawarte w funkcji celu, aby upewnić się, że ograniczenia problemu są spełnione. Praktycznym rozwiązaniem jest to, że rzeczywista wartość ilości akcji może być nierealistyczna, ponieważ ułamkowe liczby akcji nie mogą być kupowane. Bardziej sensowne jest zadanie, aby zmienne przyjmowały tylko 0 lub 1. Implikacją jest więc to, że albo kupujesz akcje, w tym przypadku włączasz je do portfela, albo w ogóle nie kupujesz. To jest zwykle nazywane problemem programowania zero-jedynkowego. Określasz to również jako problem kombinatoryczny. Widziałeś już problem optymalizacji kombinatorycznej w zagadnieniu komiwojażera. Ograniczenia zostały włączone do specjalnych terminów w funkcji celu, tak że jedyną funkcją do obliczenia jest funkcja celu. Uznając funkcję celu za podanie energii sieci w danym stanie, do rozwiązania problemu można wykorzystać paradygmat symulowanego wyżarzania i sieć Hopfielda. Masz wtedy sieć neuronową, w której każdy neuron reprezentuje akcje, a rozmiar warstwy jest określany przez liczbę akcji w puli, z której chcesz zbudować swój portfel akcji. Zaproponowany tutaj paradygmat dąży do minimalizacji energii maszyny. W związku z tym należy określić funkcję celu w celu minimalizacji, aby uzyskać najlepszy możliwy portfel.

Sieć neuronowa Tabu

Wyszukiwanie tabu, spopularyzowane przez Freda Glovera z jego wkładem, jest paradygmatem, który z powodzeniem został wykorzystany w wielu problemach optymalizacyjnych. Jest to metoda, która może sterować procedurą wyszukiwania z domeny ograniczonej do domeny rozszerzonej, tak aby znaleźć rozwiązanie lepsze niż lokalne minimum lub lokalne maksimum. Wyszukiwanie tabu (TS) sugeruje, że pamięć adaptacyjna i responsywna eksploracja muszą być częścią algorytmu. Eksploracja responsywna wykorzystuje informacje pochodzące z wybranej strategii. Taka informacja może być bardziej merytoryczna, nawet jeśli wybrana strategia jest w pewnym sensie złą strategią, niż to, co można uzyskać nawet w dobrej strategii opartej na losowości. Dzieje się tak, ponieważ takie informacje dają możliwość inteligentnego modyfikowania strategii. Możesz uzyskać wskazówki, jak zmodyfikować strategię. Kiedy masz paradygmat, który obejmuje pamięć adaptacyjną, dostrzegasz znaczenie powiązania sieci neuronowej:. TANN to sieć neuronowa Tabu. Wyszukiwanie tabu i samoorganizująca się mapa Kohonena mają wspólne podejście, ponieważ działają z „sąsiedztwem”. W miarę określania nowej dzielnicy TS zakazuje niektórych wcześniejszych rozwiązań, ponieważ klasyfikuje je jako tabu. Takie rozwiązania zawierają atrybuty identyfikowane jako aktywne tabu. Wyszukiwarka Tabu zawiera również komponenty STM i LTM. Pamięć krótkotrwała jest czasami nazywana pamięcią opartą na ostatnich czasach. Chociaż może to wystarczyć do znalezienia dobrych rozwiązań, uwzględnienie pamięci długotrwałej sprawia, że metoda wyszukiwania jest znacznie silniejsza. Nie wymaga również dłuższych przebiegów procesu wyszukiwania. Niektóre przykłady aplikacji korzystających z wyszukiwania Tabu to:

• Trening sieci neuronowych z reaktywnym wyszukiwaniem Tabu

• Tabu Learning: metoda wyszukiwania sieci neuronowych do rozwiązywania niewypukłych problemów optymalizacji

• Masowo równoległe wyszukiwanie Tabu dla problemu przypisania kwadratowego

• Implementacja automatu łączącego algorytmu wyszukiwania Tabu dla problemu komiwojażera

• Procedura wyszukiwania Tabu dla lokalizacji/alokacji wielu towarów z wymogami bilansowania

Podsumowanie

Problem komiwojażera został przedstawiony jako przykład optymalizacji nieliniowej z wykorzystaniem sieci neuronowych. Podano szczegóły formułowania funkcji energii i jej oceny. Przedstawiono podejścia do rozwiązania problemu komiwojażera z wykorzystaniem sieci Hopfield oraz mapy samoorganizującej się Kohonena. Programy C++ są dołączone do obu podejść. Wyniki z programu C++ dla sieci Hopfield odnoszą się do przykładów cztero- i pięciomiejskich wycieczek. Dane wyjściowe z programu C++ dla podejścia Kohonena są podane dla wycieczki po pięciu miastach, dla ilustracji. Otrzymane rozwiązanie jest dobre, jeśli nie optymalne. Problem z podejściem Hopfielda polega na doborze odpowiednich wartości parametrów. Wybory Hopfielda są podane dla jego problemu z wycieczką po 10 miastach. Te same wartości parametrów mogą nie działać w przypadku różnej liczby miast. Pokrótce omówiono również wersję tego podejścia podaną przez Anzai. W tym rozdziale przedstawiono również wykorzystanie sieci neuronowych do optymalizacji nieliniowej w zastosowaniu do selekcji portfela. Jesteś wprowadzony do wyszukiwania Tabu i jego wykorzystania w optymalizacji z wykorzystaniem obliczeń neuronowych.