TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN LẬP TRÌNH TÍNH TOÁN
BÀI TOÁN DỰ BÁO DOANH SỐ BÁN HÀNG BẰNG MẠNG NEURAL

Người hướng dẫn**: TS. ĐẶNG THIÊN BÌNH**
Sinh viên thực hiện**:**
NGUYỄN VĂN THƯƠNG **LỚP: 22T_KHDL**
NGUYỄN HỮU HÙNG DŨNG **LỚP: 22T_KHDL**

Đà Nẵng, 06/2023

MỤC LỤC

DANH MỤC HÌNH VẼ

MỞ ĐẦU

Trong thời kì đại khoa học kĩ thuật ngày càng tiến bộ, Công Nghệ Thông Tin đã trở thành một ngành nghề đóng vai trò vô cùng quan trọng trong sự phát triển của thế giới. Vì là sinh viên ngành Công Nghệ Thông Tin, lại còn là chuyên ngành Khoa học dữ liệu và Trí tuệ nhân tạo nên chúng em đặc biệt hứng thú với những đề tài liên quan đến Học máy cũng như Trí tuệ nhân tạo. Vì vậy, để hiểu rõ hơn về lĩnh vực này, chúng em đã chọn "Dự đoán doanh thu bán hàng bằng mạng nơ ron" làm đề tài cho đồ án này.

Qua hai kỳ học tại trường, với các kiến thức nền tảng cơ bản nhưng vô cùng hữu ích đã được học tại các môn như Kỹ thuật lập trình, Cấu trúc dữ liệu, Phương pháp tính hay Toán rời rạc, chúng em đã có thể tìm đọc và hiểu những phương pháp để thực hiện được đề tài. Tuy nhiên, có thể sẽ vẫn không thể tránh khỏi những sai sót trong quá trình thực hiện, kính mong thầy cô thông cảm và góp ý để để tài này được hoàn thiện hơn. Chúng em xin chân thành cảm ơn!

Đồ án này sẽ không thể hoàn thành được nếu thiếu sự hỗ trợ của các thầy cô hướng dẫn cũng như những người đã tư vấn cho chúng em trong quá trình chúng em làm đồ án. Nhóm chúng em mong rằng đây sẽ là minh chứng cho thành quả dạy dỗ của thầy cô cũng như nỗ lực học tập của chúng em trong suốt thời gian vừa qua.

Nhóm chúng em xin chân thành cảm ơn!

1. TỔNG QUAN ĐỀ TÀI

Trong quá trình hoạt động của doanh nghiệp, dự báo doanh thu bán hàng là một nhiệm vụ quan trọng đối với bất kỳ doanh nghiệp nào. Có nhiều lý do khiến dự báo doanh thu bán hàng trở nên quan trọng, trong đó có thể kể đến như giúp các công ty, doanh nghiệp có thể ước tính nhu cầu sản xuất, lập kế hoạch về nguồn nhân lực, vật liệu và các nguồn tài nguyên khác để đáp ứng nhu cầu thị trường; giúp dự đoán doanh thu trong tương lai, từ đó giúp các doanh nghiệp lập kế hoạch tài chính, quản lý dòng tiền và đảm bảo sự ổn định tài chính trong quá trình kinh doanh; giúp các doanh nghiệp ứng phó với các biến động thị trường…

Mục tiêu đề tài là xây dựng một mô hình dự đoán doanh thu đơn giản có thể giúp ích cho doanh nghiệp, công ty trong các vấn đề trên. Để làm được điều đó, chúng em sẽ xây dựng một mạng nơ ron đơn giản được huấn luyện bằng dữ liệu doanh thu trong quá khứ để thay đổi các trọng số của mạng, từ đó có thể dự đoán được doanh thu trong tương lai.

2. CƠ SỞ LÝ THUYẾT

2.1. Ý tưởng

Ý tưởng ban đầu là chúng em sẽ xây dựng một mạng nơ ron đơn giản gồm 3 lớp: lớp input có 4 nơ ron tương ứng với doanh thu 4 ngày liền trước trong quá khứ, lớp ẩn gồm 5 nơ ron và lớp ouput gồm 1 nơ ron là doanh thu dự đoán của ngày tiếp theo. Ban đầu, các trọng số liên kết của mạng nơ ron là giá trị ngẫu nhiên, sau đó chúng em sẽ huấn luyện mạng bằng cách dựa vào dữ liệu doanh thu trong quá khứ để hiệu chỉnh trọng số liên kết giữa các nơ ron đến khi sai số giữa nơ ron đầu ra và dữ liệu mục tiêu được nhỏ nhất. Có nhiều phương pháp để huấn luyện mạng, trong đề tài này, chúng em sẽ sử dụng phương pháp lan truyền ngược, để phân tích sâu hơn về ý tưởng, chúng em kính mời thầy cô đọc tiếp phần tiếp theo.

2.2. Cơ sở lý thuyết

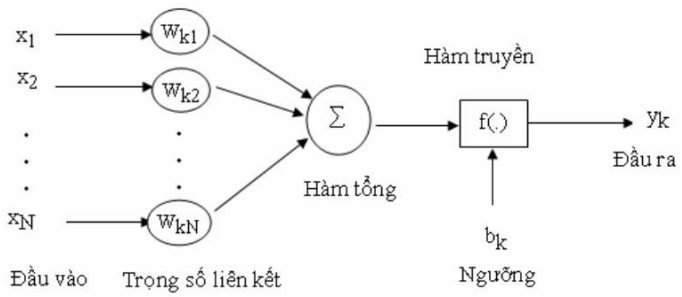
2.2.1 Tổng quan

Đầu tiên, chúng em xin phép nói rõ hơn về mạng nơ ron.

Mạng nơ ron nhân tạo (ANN – Artificial neural network) là một tập hợp các thuật toán phân tích dữ liệu cho phép xác định mối liên hệ toán học phức tạp giữa tập hợp các yếu tố ảnh hưởng (Input) với một biến số hoặc một tập hợp biến số mục tiêu (target). ANN là một ứng dụng trên nền tảng trí tuệ thông minh nhân tạo (Artificial intelligence - AI), cấu trúc của ANN và quá trình học tập, suy luận và tái tạo của nó có

khả năng tự thích ứng linh hoạt (autofit) với nhiều loại dữ liệu khác nhau. ANN xử lý thông tin phỏng theo cách thức xử lý thông tin của các hệ nơ ron sinh học trong não người. Nó được tạo nên từ một số lượng lớn các nơ ron kết nối với nhau thông qua các liên kết có trọng số, tạo thành một thể thống nhất, nhằm xử lý, phân tích một thông tin, một vấn đề. Một mạng nơron nhân tạo được cấu trúc cho một ứng dụng cụ thể (nhận dạng mẫu, phân loại dữ liệu,...) thông qua một quá trình huấn luyện học (training) từ tập các mẫu huấn luyện. Dữ liệu đầu vào (input) sẽ chạy qua toàn bộ mạng nơ ron, sẽ được xử lý, tìm các mối liên hệ và tái tạo lại thành kết quả đầu ra (output). Các output này sẽ được so sánh với với các dữ liệu mục tiêu (target) mà hệ thống đã được học trước đó. Nếu còn có sự sai lệch đáng kể giữa output và target, thì quá trình huấn luyện – học tập (training) lặp lại, các trọng số liên kết giữa các nơ ron lại được hiệu chỉnh để đưa ra output khác cải thiện hơn. Quá trình này liên tục lặp lại nhiều lần cho đến khi mang lại độ chệch bé nhất có thể giữa output và target. Như vậy, về bản chất quá trình huấn luyện - học (training) chính là quá trình hiệu chỉnh trọng số liên kết giữa các nơ ron cho đến khi đạt được một kết quả tối ưu, đó là tái tạo ra output tương tự target và có thể sử dụng để dự báo ngoài mẫu cho target.

Cấu trúc của một nơ ron nhân tạo tiêu biểu có các thành phần cơ bản như mô tả ở hình dưới:



Hình 1: Cấu trúc của nơ ron nhân tạo

Đầu vào cung cấp các tín hiệu vào (input signals) của nơ ron, các tín hiệu này thường được đưa vào dưới dạng một vector N chiều.

Các liên kết: Mỗi liên kết được thể hiện bởi một trọng số (gọi là trọng số liên kết – Synaptic weight). Trọng số liên kết giữa tín hiệu vào thứ j với nơ ron k thường được kí hiệu là w_{kj}. Thông thường, các trọng số này được khởi tạo một cách ngẫu nhiên ở thời điểm khởi tạo mạng và được cập nhật liên tục trong quá trình huấn luyện mạng.

Hàm tổng (Summing function): Thường dùng để tính tổng của tích các đầu vào với trọng số liên kết của nó: $\sum_{i=1}^{m}(w_i x_i + bias)$

Hàm truyền (Transfer function): Hàm này được dùng để giới hạn phạm vi đầu ra của mỗi nơ ron. Nó nhận đầu vào là kết quả của hàm tổng và ngưỡng đã cho. Thông thường, phạm vi đầu ra của mỗi nơ ron được giới hạn trong đoạn [0,1] hoặc [-1, 1]. Các hàm truyền rất đa dạng, thông thường, người ta sẽ dùng hàm kích hoạt phi tuyến tính giúp mô hình mạng nơ ron có khả năng học các mô hình phi tuyến tính phức tạp. Một hàm kích hoạt phi tuyến tính như hàm sigmoid, hàm tanh, hàm ReLU, hay hàm softmax, có khả năng biến đổi đầu vào của mạng thành một đầu ra phi tuyến tính, cho phép mạng nơ ron học và mô hình hóa các quan hệ phi tuyến tính phức tạp trong dữ liệu đầu vào.

Đầu ra (output) Là tín hiệu đầu ra của một nơ ron, với mỗi nơ ron sẽ có tối đa là một đầu ra.

Như vậy tương tự như nơ ron sinh học, nơ ron nhân tạo cũng nhận các tín hiệu đầu vào, xử lý (nhân các tín hiệu này với trọng số liên kết, tính tổng các tích thu được rồi gửi kết quả tới hàm truyền), và tái tạo tín hiệu đầu ra (là kết quả của hàm truyền).

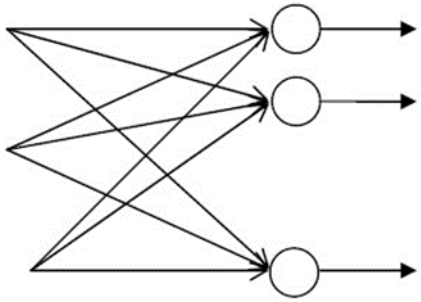
2.2.2 Mô hình mạng nơ ron nhân tạo:

Mặc dù mỗi nơron đơn lẻ có thể thực hiện những chức năng xử lý thông tin nhất định, sức mạnh của tính toán nơron chủ yếu có được nhờ sự kết hợp các nơron trong một kiến trúc thống nhất. Một mạng nơron là một mô hình tính toán được xác định qua các tham số: kiểu nơron (như là các nút nếu ta coi cả mạng nơron là một đồ thị), kiến trúc kết nối (sự tổ chức kết nối giữa các nơron) và thuật toán học (thuật toán dùng để học cho mạng).

Về bản chất một mạng nơron có chức năng như là một hàm ánh xạ F: X → Y, trong đó X là không gian trạng thái đầu vào (input state space) và Y là không gian trạng thái đầu ra (output state space) của mạng. Các mạng chỉ đơn giản là làm nhiệm vụ ánh xạ các vec-tơ đầu vào x ∈ X sang các vec-tơ đầu ra y ∈ Y thông qua "bộ lọc" (filter) các trọng số. Tức là y = F(x) = s(W, x), trong đó W là ma trận trọng số liên kết. Hoạt động của mạng thường là các tính toán số thực trên các ma trận.

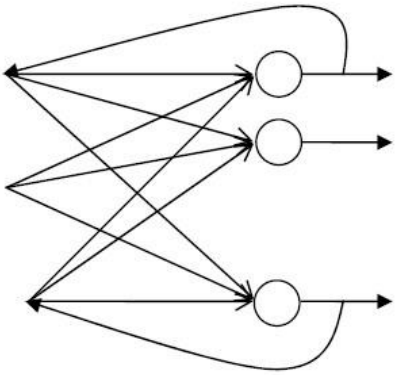
Trong bộ não của con người, các tế bào nơron liên kết với nhau chằng chịt và tạo nên một mạng lưới vô cùng phức tạp, tuy nhiên mạng nơron nhân tạo được chia thành các loại chính sau:

Mạng nơron truyền thẳng một lớp (perceptron) là loại mạng chỉ có lớp nơron đầu vào và một lớp nơron đầu ra (thực chất lớp nơron đầu vào không có vai trò xử lý, do đó ta nói mạng chỉ có một lớp). Loại mạng này còn được gọi là mạng perceptron một lớp. Mỗi nơron đầu ra có thể nhận tín hiệu từ các đầu vào x1, x2, …, xm để tạo ra tín hiệu đầu ra tương ứng.



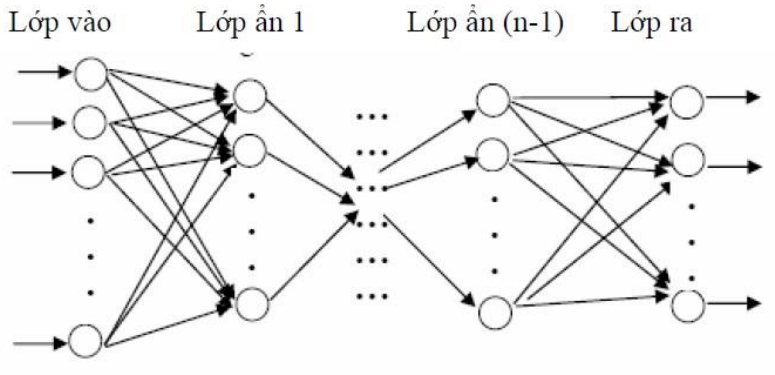
Hình 2: Mạng nơron truyền thẳng 1 lớp (Single-layer feedforward network)

Mạng có phản hồi (feedback network) là mạng mà đầu ra của một nơron có thể trở thành đầu vào của nơron trên cùng một lớp hoặc của lớp trước đó. Mạng feedback có chu trình khép kín gọi là mạng quy hồi (recurrent network).



Hình 3: Mạng nơron hồi quy 1 lớp

Mô hình mạng truyền thẳng nhiều lớp: Mô hình mạng nơron được sử dụng rộng rãi nhất là mô hình mạng nhiều tầng truyền thẳng (MLP: Multi Layer Perceptron). Một mạng MLP tổng quát là mạng có n ($n \geq 2$) lớp (thông thường lớp đầu vào không được tính đến): trong đó gồm một lớp đầu ra (lớp thứ n) và (n-1) lớp ẩn.



Hình 4: Mạng MLP tổng quát

Một số kết quả đã được chứng minh với mạng MLP:

- Bất kì một hàm Boolean nào cũng có thể biểu diễn được bởi một mạng MLP 2 lớp trong đó các nơron sử dụng hàm truyền sigmoid.
- Tất cả các hàm liên tục đều có thể xấp xỉ bởi một mạng MLP 2 lớp sử dụng hàm truyền sigmoid cho các nơron lớp ẩn và hàm truyền tuyến tính cho các nơron lớp ra với sai số nhỏ tùy ý.
- Mọi hàm bất kỳ đều có thể xấp xỉ bởi một mạng MLP 3 lớp sử dụng hàm truyền sigmoid cho các nơron lớp ẩn và hàm truyền tuyến tính cho các nơron lớp ra.

Mô hình mạng kết hợp : Trong thực tế, người ta thường kết hợp nhiều loại mạng để giải quyết các bài toán. Có thể tham khảo mô hình ANN hệ thống chơi cờ vây AlphaGo của Google bao gồm kết hợp 3 mạng ANN. Mạng 1 SL có cấu trúc xoắn bao gồm 13 lớp ẩn với tập dữ liệu học có giám sát là hàng triệu nước cờ đã có thu thập từ các ván cờ các kỳ thủ đã chơi. Mạng 2 RL có cấu trúc giống mạng SL nhưng được tối ưu hóa bộ trọng số qua bộ huấn luyện là các ván cờ tự chơi với nhau giữa các máy. Mạng 3 Value dự đoán kết quả trò chơi dựa trên RL, giá trị của vị trí p được định nghĩa như 1 kỳ vọng phân phối các kết quả phát từ vị trí p đến cuối ván cờ. Cuối cùng, AlphaGo kết hợp mạng trên với thuật toán Monte Carlo Tree Search để đưa ra đánh giá vị trí cuối cùng cho nước đi tiếp theo.

2.2.3 Quá trình học của mạng nơron:

Học là quá trình cập nhật trọng số sao cho giá trị hàm lỗi là nhỏ nhất. Một mạng nơron được huấn luyện sao cho với một tập các vec-tơ đầu vào X, mạng có khả năng tạo ra tập các vec-tơ đầu ra mong muốn Y của nó. Tập X được sử dụng cho huấn luyện mạng được gọi là tập huấn luyện (training set). Các phần tử x thuộc X được gọi là các mẫu huấn luyện (training example). Quá trình huấn luyện bản chất là sự thay đổi các

trọng số liên kết của mạng. Trong quá trình này, các trọng số của mạng sẽ hội tụ dần tới các giá trị sao cho với mỗi vec-tơ đầu vào x từ tập huấn luyện, mạng sẽ cho ra vec-tơ đầu ra y như mong muốn.

Có ba phương pháp học phổ biến là học có giám sát (supervised learning), học không giám sát (unsupervised learning) và học tăng cường (Reinforcement learning).

Học có giám sát trong các mạng nơron:

Học có giám sát có thể được xem như việc xấp xỉ một ánh xạ: X→ Y, trong đó X là tập các vấn đề và Y là tập các lời giải tương ứng cho vấn đề đó. Các mẫu (x, y) với x = (x1, x2, ..., xn) ∈ X, y = (yl, y2, ..., ym) ∈ Y được cho trước. Học có giám sát trong các mạng nơron thường được thực hiện theo các bước sau:

- Bước 1: Xây dựng cấu trúc thích hợp cho mạng nơron, chẳng hạn có n nơron vào, m nơron đầu ra, và khởi tạo các trọng số liên kết của mạng.
- Bước 2: Đưa một vec-tơ x trong tập mẫu huấn luyện X vào mạng
- Bước 3: Tính vec-tơ đầu ra z của mạng
- Bước 4: So sánh vec-tơ đầu ra mong muốn t (là kết quả được cho trong tập huấn luyện) với vec-tơ đầu ra z do mạng tạo ra; nếu có thể thì đánh giá lỗi.
- Bước 5: Hiệu chỉnh các trọng số liên kết theo một cách nào đó sao cho ở lần tiếp theo khi đưa vec-tơ x vào mạng, vec-tơ đầu ra z sẽ giống với t hơn.
- Bước 6: Nếu cần, lặp lại các bước từ 2 đến 5 cho tới khi mạng đạt tới trạng thái hội tụ. Việc đánh giá lỗi có thể thực hiện theo nhiều cách, cách dùng nhiều nhất là sử dụng lỗi tức thời: Err = (z - t), hoặc Err = |z - t| ; lỗi trung bình bình phương (MSE: mean-square error): Err = $(z- t)^2/2$.

Có hai loại lỗi trong đánh giá một mạng nơron. Thứ nhất, gọi là lỗi rõ ràng (apparent error), đánh giá khả năng xấp xỉ các mẫu huấn luyện của một mạng đã được huấn luyện. Thứ hai, gọi là lỗi kiểm tra (test error), đánh giá khả năng tổng quát hóa của một mạng đã được huấn luyện, tức khả năng phản ứng với các vec-tơ đầu vào mới. Để đánh giá lỗi kiểm tra chúng ta phải biết đầu ra mong muốn cho các mẫu kiểm tra.

Thuật toán tổng quát ở trên cho học có giám sát trong các mạng nơron có nhiều cài đặt khác nhau, sự khác nhau chủ yếu là cách các trọng số liên kết được thay đổi trong suốt thời gian học. Trong đó tiêu biểu nhất là thuật toán lan truyền ngược sai số.

2.2.4 Khả năng ứng dụng của mạng nơron nhân tạo

Đặc trưng của mạng nơron nhân tạo là khả năng học. Nó có thể gần đúng mối quan hệ tương quan phức tạp giữa các yếu tố đầu vào và đầu ra của các quá trình cần

nghiên cứu và khi đã học được thì việc kiểm tra độc lập thường cho kết quả tốt. Sau khi đã học xong, mạng nơron nhân tạo có thể tính toán kết quả đầu ra tương ứng với bộ số liệu đầu vào mới.

Về mặt cấu trúc, mạng nơron nhân tạo là một hệ thống gồm nhiều phần tử xử lý đơn giản cùng hoạt động song song. Tính năng này của ANN cho phép nó có thể được áp dụng để giải các bài toán lớn.

Về khía cạnh toán học, theo định lý Kolmogorov, một hàm liên tục bất kỳ f(x1, x2,…, xn) xác định trên khoảng In (với I =[0,1]) có thể được biểu diễn dưới dạng:

$$f(x) = \sum_{j=1}^{2n+1} \chi_j \left(\sum_{i=1}^{n} \Psi_{ij}(x_i) \right)$$

trong đó: χ_j , Ψ_{ij} là các hàm liên tục một biến. Ψ_{ij} là hàm đơn điệu, không phụ thuộc vào hàm f. Mặt khác, mô hình mạng nơron nhân tạo cho phép liên kết có trọng số các phần tử phi tuyến (các nơron đơn lẻ) tạo nên dạng hàm tổng hợp từ các hàm thành phần. Do vậy, sau một quá trình điều chỉnh sự liên kết cho phù hợp (quá trình học), các phần tử phi tuyến đó sẽ tạo nên một hàm phi tuyến phức tạp có khả năng xấp xỉ hàm biểu diễn quá trình cần nghiên cứu. Kết quả là đầu ra của nó sẽ tương tự với kết quả đầu ra của tập dữ liệu dùng để luyện mạng. Khi đó ta nói mạng nơron nhân tạo đã học được mối quan hệ tương quan đầu vào - đầu ra của quá trình và lưu lại mối quan hệ tương quan này thông qua bộ trọng số liên kết giữa các nơron. Do đó, mạng nơron nhân tạo có thể tính toán trên bộ số liệu đầu vào mới để đưa ra kết quả đầu ra tương ứng.

Với những đặc điểm đó, mạng nơron nhân tạo đã được sử dụng để giải quyết nhiều bài toán thuộc nhiều lĩnh vực của các ngành khác nhau. Các nhóm ứng dụng mà mạng nơron nhân tạo đã được áp dụng rất có hiệu quả là:

- Bài toán phân lớp: Loại bài toán này đòi hỏi giải quyết vấn đề phân loại các đối tượng quan sát được thành các nhóm dựa trên các đặc điểm của các nhóm đối tượng đó. Đây là dạng bài toán cơ sở của rất nhiều bài toán trong thực tế: nhận dạng chữ viết, tiếng nói, phân loại gen, phân loại chất lượng sản phẩm, …
- Bài toán dự báo: Mạng nơron nhân tạo đã được ứng dụng thành công trong việc xây dựng các mô hình dự báo sử dụng tập dữ liệu trong quá

khứ để dự đoán số liệu trong tương lai. Đây là nhóm bài toán khó và rất quan trọng trong nhiều ngành khoa học.

- Bài toán điều khiển và tối ưu hoá: Nhờ khả năng học và xấp xỉ hàm mà mạng nơron nhân tạo đã được sử dụng trong nhiều hệ thống điều khiển tự động cũng như góp phần giải quyết những bài toán tối ưu trong thực tế.

Tóm lại, mạng nơron nhân tạo được xem như là một cách tiếp cận đầy tiềm năng để giải quyết các bài toán có tính phi tuyến, phức tạp và đặc biệt là trong tình huống mối quan hệ bản chất vật lý của quá trình cần nghiên cứu không dễ thiết lập tường minh.

2.2.5 Mạng nơron truyền thẳng nhiều lớp

Mạng perceptron một lớp do F.Rosenblatt đề xuất năm 1960 là mạng truyền thẳng chỉ một lớp vào và một lớp ra không có lớp ẩn. Trên mỗi lớp này có thể có một hoặc nhiều nơron. Rosenblatt đã chứng minh rằng quá trình học của mạng Perceptron sẽ hội tụ tới bộ trọng số W, biểu diễn đúng các mẫu học với điều kiện là các mẫu này biểu thị các điểm rời rạc của một hàm khả tách tuyến tính nào đó (f: R n →R được gọi là khả tách tuyến tính nếu các tập {F -1 (xk)}, với xk thuộc miền trị của f, có thể tách được với nhau bởi các siêu phẳng trong không gian Rn).

Năm 1969, Minsky và Papert đã chứng minh một cách chặt chẽ rằng lớp hàm thể hiện sự phụ thuộc giữa đầu vào và đầu ra có thể học bởi mạng Perceptron một lớp là lớp hàm khả tách tuyến tính. Khả tách tuyến tính là trường hợp tồn tại một mặt siêu phẳng để phân cách tất cả các đối tượng của một lớp này với một lớp khác, ví dụ một mặt phẳng sẽ phân chia không gian ba chiều thành hai vùng riêng biệt. Mở rộng ra, nếu có n đầu vào, n>2 thì công thức $\Sigma_{j=1}^{n} w_{ij} x_j = \theta_i$ tạo nên một siêu phẳng có n-1 chiều trong không gian n chiều, nó chia không gian đó thành hai nửa. Trong nhiều bài toán thực tế đòi hỏi chia các vùng của các điểm trong một siêu không gian thành các lớp riêng biệt. Loại bài toán này gọi là bài toán phân lớp. Bài toán phân lớp có thể giải quyết bằng cách tìm các tham số thích hợp cho một siêu phẳng để nó có thể chia không gian n chiều thành các vùng riêng biệt.

Với tính chất của như đã nêu trên, mạng perceptron một lớp có thể mô tả các hàm logic như AND, OR và NOT. Tuy nhiên nó không thể hiện được hàm XOR. Như vậy chứng tỏ mô hình perceptron một lớp không thể giải quyết bài toán này. Vấn đề này sẽ được giải quyết bằng mô hình mạng nơron perceptron nhiều lớp (Multi Layer Perceptron - MLP)

Mạng perceptron nhiều lớp (Multilayer Perceptron – MLP) còn được gọi là mạng truyền thẳng nhiều lớp là sự mở rộng của mô hình mạng perceptron với sự bổ sung thêm những lớp ẩn và các nơron trong các lớp ẩn này có hàm truyền (hàm kích hoạt) dạng phi tuyến. Mạng MLP có một lớp ẩn là mạng nơron nhân tạo được sử dụng phổ biến nhất, nó có thể xấp xỉ các hàm liên tục được định nghĩa trên một miền có giới hạn cũng như những hàm là tập hợp hữu hạn của các điểm rời rạc.

Cấu trúc của một mạng MLP tổng quát có thể mô tả như sau:

- Đầu vào là các vec-tơ (x1, x2, ..., xp) trong không gian p chiều, đầu ra là các vec- tơ (y1, y2, ..., yq) trong không gian q chiều. Đối với các bài toán phân loại, p chính là kích thước của mẫu đầu vào, q chính là số lớp cần phân loại.
- Mỗi nơron thuộc lớp sau liên kết với tất cả các nơron thuộc lớp liền trước nó.
- Đầu ra của nơron lớp trước là đầu vào của nơron thuộc lớp liền sau nó

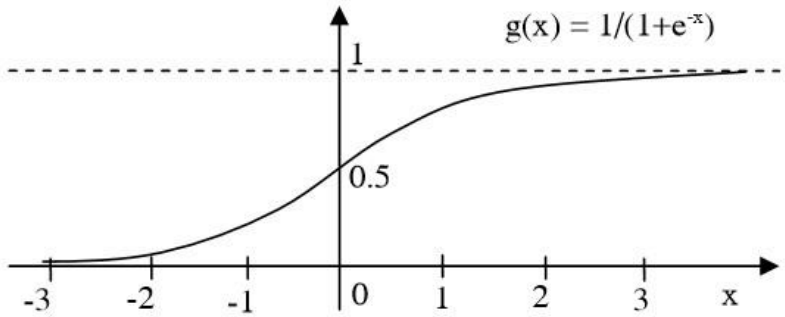
Hoạt động của mạng MLP như sau: tại lớp đầu vào các nơron nhận tín hiệu vào xử lý (tính tổng trọng số, gửi tới hàm truyền) rồi cho ra kết quả (là kết quả của hàm truyền); kết quả này sẽ được truyền tới các nơron thuộc lớp ẩn thứ nhất; các nơron tại đây tiếp nhận như là tín hiệu đầu vào, xử lý và gửi kết quả đến lớp ẩn thứ 2 ; … ; quá trình tiếp tục cho đến khi các nơron thuộc lớp ra cho kết quả

2.2.6 Một số vấn đề cần chú ý khi sử dụng mạng MLP

Mạng nơron perceptron nhiều lớp là loại mạng nơron được sử dụng trong nhiều ứng dụng thực tế. Tuy nhiên, để mạng có thể đưa ra kết quả tốt, chúng ta cần quan tâm đến một số vấn đề có ảnh hưởng khá quan trọng đến hiệu quả làm việc của nó bao gồm: vấn đề chuẩn hoá số liệu đầu vào, vấn đề học chưa đủ và học quá của mạng, vấn đề lựa chọn một cấu trúc mạng phù hợp với bài toán.

2.2.6.1. Vấn đề chuẩn hóa số liệu đầu vào

Mạng MLP thường sử dụng hàm truyền là hàm sigmoid có dạng như sau:



Hình 5: Hàm sigmoid g(x) = 1/(1+e -x)

Với dạng hàm này, giá trị ở đầu ra của mỗi nơron nằm trong phạm vi khoảng (0,1) và nó đạt các giá trị bão hoà (xấp xỉ 0 hay 1) khi $|x|$ lớn. Do đó, khi đầu vào của mạng có giá trị tuyệt đối lớn thì ta cần chuẩn hoá nó về khoảng có giá trị nhỏ, nếu không thì các nơron tại các lớp ẩn ngay ban đầu đã có thể đạt giá trị bão hoà và quá trình học của mạng không đạt kết quả mong muốn. Với dạng hàm như trên thì giá trị đầu vào của mạng thường được chuẩn hoá về khoảng thuộc đoạn [-3, 3]. Mặt khác, do tín hiệu đầu ra của nơron nằm trong khoảng giá trị (0,1) nên các giá trị đầu ra thực tế trong các mẫu học cũng cần chuẩn hoá về khoảng giá trị này để có thể dùng cho quá trình luyện mạng. Do vậy trong quá trình tính toán, để có các giá trị thực tế ở đầu ra của mạng chúng ta cần phải chuyển các giá trị trong khoảng (0,1) về miền các giá trị thực tế.

2.2.6.2. Vấn đề học chưa đủ và học quá thuộc của mạng

Vấn đề mấu chốt khi xây dựng một mạng nơron nhân tạo là làm thế nào mạng có khả năng tổng quát hoá cao để đưa ra kết quả tốt cả với những trường hợp đầu vào của mạng không nằm trong tập mẫu đã dùng để luyện mạng. Giống như các mô hình hồi quy phi tuyến khác, đối với mạng nơron nhân tạo ta cũng phải giải quyết hai vấn đề là ANN học chưa đủ (underfitting) và học quá (overfitting). Khi mạng có cấu trúc (số nút ẩn và liên kết) cũng như số lần học chưa đủ so với nhu cầu của bài toán thì sẽ dẫn tới tình trạng mạng không đủ khả năng mô tả gần đúng mối quan hệ tương quan giữa đầu vào và đầu ra của quá trình cần dự báo và dẫn tới học chưa đủ. Trái lại, nếu mạng quá phức tạp (quá nhiều nút ẩn và quá nhiều tham số) và được học "quá khít" đối với các mẫu dùng để luyện mạng thì có thể dẫn tới tình trạng mạng học cả thành phần nhiễu lẫn trong các mẫu đó, đây là tình trạng "học quá thuộc" của mạng. Vấn đề nêu trên có thể làm cho nhiều loại mạng nơron, đặc biệt là mạng MLP có thể có những trường hợp cho kết quả dự đoán rất sai lệch với thực tế.

Một số giải pháp cho vấn đề học quá của mạng:

- Sử dụng tập số liệu có tính đại diện tốt để luyện mạng: Đây được xem là một cách khá tốt để tránh hiện tượng overfitting. Khi tập mẫu dùng để luyện mạng thể hiện được nhiều trạng thái có thể xảy ra của quá trình cần nghiên cứu thì sau khi học mạng sẽ có khả năng tổng quát hoá tương đối tốt từ tập dữ liệu đó và sẽ không chịu ảnh hưởng nhiều của hiện tượng overfitting. Ngoài ra một số biện pháp dưới đây cũng có thể góp phần quan trọng giúp khắc phục hiện tượng overfitting của mạng.

- Lựa chọn cấu trúc mô hình phù hợp: Việc lựa chọn mô hình của mạng (số lớp ẩn, số nơron trên mỗi lớp ẩn) có ảnh hưởng quan trọng đến hiện tượng học chưa đủ (underfitting) và học quá (overfitting) của mạng. Các mạng có độ phức tạp hơn tuy nó có thể học khá chính xác các mẫu được sử dụng nhưng chính điều này lại làm cho nó học quá nhiều cả thành phần nhiễu nên khả năng tổng quát hoá giảm và dẫn tới hiện tượng học quá (overfitting).
- Dừng học đúng lúc: giải pháp dừng học đúng lúc để tránh hiện tượng học quá của mạng như sau :

+ Tập mẫu được chia làm hai phần: một phần dùng để luyện mạng và phần còn lại để kiểm thử.

+ Sử dụng các giá trị khởi tạo nhỏ.

+ Sử dụng hằng số tốc độ học có giá trị thấp

+ Tính toán sự thay đổi lỗi kiểm thử trong quá trình luyện mạng

+ Dừng học khi thấy lỗi kiểm thử bắt đầu tăng.

2.2.6.3. Lựa chọn kích thước mạng:

Các công trình dựa trên định lý của Kolmogorov dự kiến rằng toàn bộ các ánh xạ liên tục từ $[0,1]^p$ đến $[0,1]^n$ đều có thể được xấp xỉ bằng một mạng perceptron ba lớp có lớp vào gồm p nơron, lớp ra gồm n nơron và lớp ẩn gồm (2p+1) nơron. Tuy nhiên không thể chỉ ra được chính xác số lượng nơron tối ưu trong mạng, tính chất của các nơron, tức là dạng phi tuyến cụ thể thực hiện phép xấp xỉ này. Một số công trình nghiên cứu về chủ đề này cho rằng số nơron tối ưu ở lớp ẩn thường nhỏ hơn (2p+1). Ngoài ra cũng cần phải nói cơ sở dữ liệu học phải có kích thước phù hợp với kiến trúc mạng. Theo Vapnik và Chervonenkis, cơ sở dữ liệu học phải có số mẫu thoả mãn: $N \approx 10.N_w$, ở đó N_w là số trọng số của mạng. Gọi số nơron thuộc lớp ẩn là L, số nơron ở lớp vào là p thì trọng số của các kết nối giữa lớp vào và lớp ẩn thứ nhất (kể cả ngưỡng) là: D=(p+1).L. Theo một số kết quả nghiên cứu, số mẫu của cơ sở dữ liệu học cần phải thoả mãn: $N \approx 4.D$. Khi số lượng mẫu của cơ sở dữ liệu học chưa đạt đến giới hạn cần thiết thì ta nên làm giảm số lượng các kết nối để tránh hiện tượng học thuộc lòng.

3 TỔ CHỨC CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

3.1 Phát biểu bài toán

Mô tả đầu vào (Input) và đầu ra (Output)

- INPUT:
  File *.txt, trong file gồm 3 dòng, bao gồm:
  ➢ Dòng 1 là số n: số dữ liệu doanh thu đã thu thập trong quá khứ.
  ➢ Dòng 2 chứa n dữ liệu doanh thu đã thu thập.
- OUTPUT:
  Chương trình cung cấp cho người dùng 3 thao tác:
  1. Nhập tên file dữ liệu dùng để huấn luyện và tiến hành huấn luyện.
  2. Kiểm tra khả năng học của mạng.
  3. Dự đoán doanh thu ngày tiếp theo.

3.2 Cấu trúc dữ liệu

3.2.1 Kiểu dữ liệu cơ bản

```
int INPUT_COUNT=4, HIDDEN_COUNT=5, OUTPUT_COUNT=1, TRAINING_SET, DATASET_SIZE;
double* dataset;
double inputWeights[INPUT_COUNT][HIDDEN_COUNT], hiddenWeights[HIDDEN_COUNT], hiddenBias[HIDDEN_COUNT],
    outputBias, learnRate, datamax=-INT_MAX;
```

Hình 6: Kiểu dữ liệu cơ bản trong chương trình

3.2.2 Khai báo các hàm

```
double sigmoid(double x); // ham sigmoid
double dSigmoid(double x); // dao ham cua ham sigmoid
double predict(double input[], double inputWeights[][HIDDEN_COUNT], double hiddenWeights[],
               double hiddenBias[], double *outputBias); // ham du doan
void learn(double input[][INPUT_COUNT], double outputReal[], double learnRate,
           double inputWeights[][HIDDEN_COUNT], double hiddenWeights[], double hiddenBias[],
           double *outputBias); // ham huan luyen mang
void giveRandWeights(double inputWeights[][HIDDEN_COUNT], double hiddenWeights[], double hiddenBias[],
                     double *outputBias); // ham tao trong so ngau nhien
void getData(); // ham lay du lieu
void printMenu(); // ham in menu
```

Hình 7: Các hàm trong chương trình

3.3 Thuật toán

Đầu tiên, vì đầu ra của hàm sigmoid nằm trong khoảng (0;1) nên chúng em sẽ chuyển tập dữ liệu về trong khoảng đó bằng cách chia mỗi dữ liệu cho giá trị lớn nhất của tập dữ liệu có trong file, tiếp theo sẽ đến quá trình học.

Thuật toán học theo phương pháp lan truyền ngược sai số là một trong số những kết quả nghiên cứu quan trọng nhất đối với sự phát triển của mạng nơron nhân tạo.

Thuật toán này được áp dụng cho mạng truyền thẳng nhiều lớp trong đó các nơron có thể sử dụng các hàm truyền là các hàm liên tục có các dạng khác nhau.

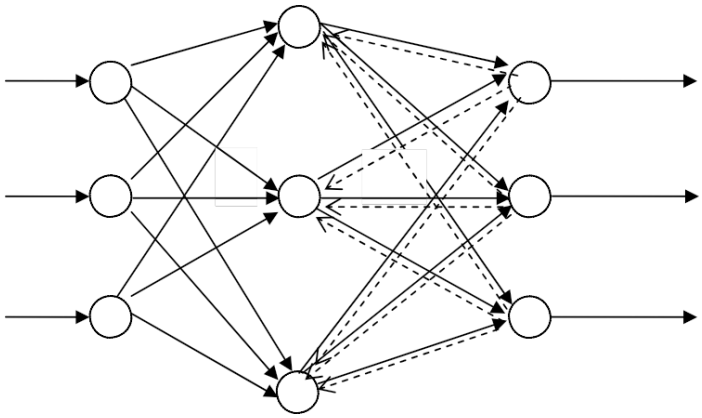
3.3.1 Mô hình của thuật toán lan truyền ngược sai số

Thuật toán sử dụng một tập các mẫu gồm các cặp đầu vào - đầu ra để luyện mạng. Với mỗi cặp đầu vào - đầu ra ($x^{(k)}$, $d^{(k)}$) thuật toán lan truyền ngược sai số thực hiện hai giai đoạn sau:

- Giai đoạn thứ nhất, mẫu đầu vào $x(^k)$ được truyền từ lớp vào tới lớp ra, và ta có kết quả đầu ra tính toán được là $y(^k)$.
- Giai đoạn tiếp theo, tín hiệu lỗi được tính toán từ sự khác nhau giữa đầu ra quan sát được $d(^k)$ với đầu ra tính toán $y(^k)$ sẽ được lan truyền ngược lại từ lớp ra đến các lớp trước để điều chỉnh các trọng số của mạng.

Để làm ví dụ ta xét mạng truyền thẳng có một lớp ẩn dưới đây, đối với các mạng có kích thước lớn hơn thì thao tác cũng tương tự.

Mạng nơron được xét có m nơron ở lớp vào, l nơron trong lớp ẩn và n nơron ở lớp ra. Đường kẻ liền thể hiện luồng tín hiệu được truyền từ đầu vào tới đầu ra còn các đường kẻ nét đứt thể hiện luồng tín hiệu lỗi được truyền ngược trở lại từ đầu ra.



Hình 8: Lan truyền tín hiệu trong quá trình học theo pp lan truyền ngược sai số

Chúng ta xét một cặp đầu vào - đầu ra để luyện mạng (x, d), để đơn giản chúng ta bỏ ký hiệu mũ k thể hiện số thứ tự của cặp mẫu này trong bộ mẫu dùng để luyện mạng. Khi đưa vào đầu vào x, nơron thứ q trong lớp ẩn sẽ nhận tín hiệu vào của mạng là :

$$net_q = \sum_{j=1}^{m} v_{qj} x_j$$

Nơron q ở lớp ẩn sẽ tính toán và tạo kết quả ở đầu ra của nó là :

$$z_q = g(net_q) = g\left(\sum_{j=1}^{m} v_{qj}x_j\right)$$

Do đó tín hiệu vào của nơron thứ i trên lớp ra sẽ là :

$$net_i = \sum_{q=1}^{l} w_{qi}z_j = \sum_{q=1}^{l} w_{qi}g\left(\sum_{j=1}^{m} v_{qj}x_j\right)$$

Và cuối cùng, đầu ra của nơron i trên lớp ra sẽ là :

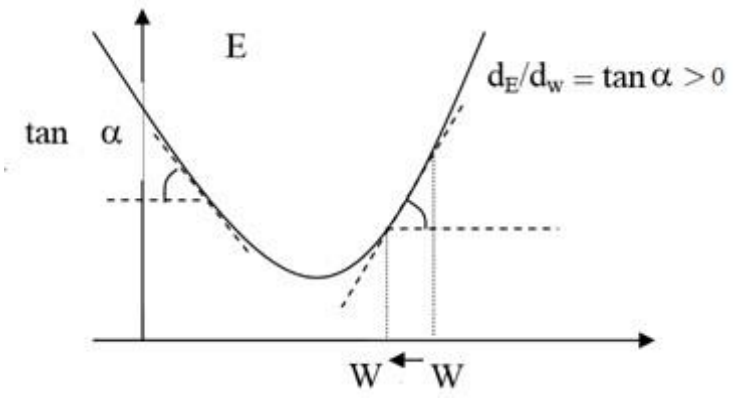
$$y_i = g(net_i) = g\left(\sum_{q=1}^{l} w_{qi}z_j\right) = g\left(\sum_{q=1}^{l} w_{qi}g\left(\sum_{j=1}^{m} v_{qj}x_j\right)\right)$$

Công thức trên cho biết quá trình lan truyền tín hiệu từ đầu vào qua lớp ẩn tới đầu ra. Tiếp theo chúng ta xét tín hiệu lỗi được lan truyền ngược lại từ lớp ra. Trước hết, đối với mỗi cặp giá trị vào – ra chúng ta xây dựng một hàm giá như sau:

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(d_i - y_i)^2 = \frac{1}{2}\sum_{i=1}^{n}[d_i - g(net_i)]^2 = \frac{1}{2}\sum_{i=1}^{n}\left[d_i - g(\sum_{q=1}^{l} w_{qi}z_j)\right]^2$$

Như vậy với một tập gồm p mẫu học, chúng ta lần lượt xây dựng được p hàm giá như vậy. Việc học của mạng hay nhiệm vụ của giải thuật thực chất là tìm kiếm tập trọng số W trong không gian R^M (M là số trọng số có trong mạng) để lần lượt tối thiểu hoá các hàm giá như vậy. Điều đáng chú ý là việc tối thiểu hoá được tiến hành liên tiếp nhau và theo chu kỳ đối với các hàm giá.

Để tối thiểu hoá các hàm giá như vậy, giải thuật Lan truyền ngược sai số sử dụng phương pháp giảm gradient để điều chỉnh các trọng số liên kết giữa các nơron. Bản chất của phương pháp này là khi sai số E được vẽ như hàm của tham số gây ra sai số sẽ phải có một cực tiểu tại bộ giá trị nào đó của tham số. Khi quan sát độ dốc của đường cong, chúng ta quyết định phải thay đổi tham số thế nào để có thể tiến gần đến cực tiểu cần tìm kiếm hơn. Trong hình vẽ dưới đây, giá trị của trọng số phải giảm nếu đạo hàm dE/dw là dương.



Hình 9: Sai số E được xét là hàm của trọng số

Bằng biểu thức, chúng ta có thể biểu diễn phương pháp giảm gradient như sau :

$\Delta w = w^{(new)} - w^{(old)} = -\eta.\partial E/\partial w$

Ở đây η là hằng số dương xác định tốc độ giảm giá trị của w, còn dấu âm chỉ chiều giảm gradient.

Áp dụng phương pháp giảm gradient đối với các trọng số liên kết giữa các nơron trong lớp ẩn tới các nơron của lớp ra ta có:

$$\Delta w_{iq} = -\eta \frac{\partial E}{\partial w_{iq}}$$

Do hàm sai số E là một hàm phức tạp và là hàm gián tiếp của trọng số w_{iq}, sử dụng nguyên tắc tính đạo hàm của hàm gián tiếp cho $\frac{\partial E}{\partial w_{iq}}$ ta có:

$$\Delta w_{iq} = -\eta \left[\frac{\partial E}{\partial y_i}\right]\left[\frac{\partial y_i}{\partial net_i}\right]\left[\frac{\partial net_i}{\partial w_{iq}}\right] = -\eta[d_i - y_i][g'(net_i)][z_q] \triangleq \eta\, \delta_{oi}\, z_q$$

Trong đó δ_{oi} là tín hiệu sai số và chỉ số oi có nghĩa là nút thứ i trong lớp ra.

Tín hiệu sai số được tính như sau:

$$\delta_{oi} \triangleq -\left[\frac{\partial E}{\partial net_i}\right] = -\left[\frac{\partial E}{\partial y_i}\right]\left[\frac{\partial y_i}{\partial net_i}\right] = [d_i - y_i][g'(net_i)]$$

Trong đó net_i là tín hiệu vào của nơron thứ i trên lớp ra và $g'(net_i) = \partial g(net_i)/\partial net_i$.

Để điều chỉnh trọng số của các liên kết giữa lớp vào tới lớp ẩn ta cũng sử dụng phương pháp giảm gradient và lấy đạo hàm theo các biến trung gian như đã áp dụng ở trên. Xét liên kết giữa nơron thứ j ở lớp vào và nơron thứ q trên lớp ra:

$$\Delta v_{qj} = -\eta \frac{\partial E}{\partial v_{qj}} = -\eta \left[\frac{\partial E}{\partial net_q}\right]\left[\frac{\partial net_q}{\partial v_{qj}}\right] = -\eta \left[\frac{\partial E}{\partial z_q}\right]\left[\frac{\partial z_q}{\partial net_q}\right]\left[\frac{\partial net_q}{\partial v_{qj}}\right]$$

$$\Delta v_{qj} = \eta \sum_{i=1}^{n} [(d_i - y_i) g'(net_i) w_{iq}]\, g'(net_q) x_j$$

$$\Delta v_{qj} = \eta \sum_{i=1}^{n} [\delta_{oi} w_{iq}]\, g'(net_q) x_j = \mu \delta_{hq} x_j$$

Trong đó δ_{hq} là tín hiệu lỗi của nơron thứ q trong lớp ẩn và được định nghĩa như sau:

$$\delta_{hq} \triangleq -\left[\frac{\partial E}{\partial net_q}\right] = -\left[\frac{\partial E}{\partial z_q}\right]\left[\frac{\partial z_q}{\partial net_q}\right] = g'(net_i) \sum_{i=1}^{n} [\delta_{oi} w_{iq}]$$

Với net_q là tín hiệu vào của nơron thứ q, như vậy tín hiệu lỗi của nơron trên lớp ẩn khác với tín hiệu lỗi của nơron trên lớp ra. Vì sự khác nhau này, thủ tục điều chỉnh trọng số được gọi là luật học delta mở rộng. Nhìn lại công thức tín hiệu lỗi δ_{hq} của nơron thứ q trong lớp ẩn được xác định từ các tín hiệu lỗi δ_{oi} của các nơron trên lớp ra.

Tổng quát đối với lớp bất kỳ, luật lan truyền ngược có dạng:

$\Delta w_{ij} = \eta\, \delta_i\, x_j = \eta\, \delta_{\text{output\_i}}\, x_{\text{input\_j}}$ trong đó "output_i" là đầu ra của nơron i và "input_j" là đầu vào của nơron j, δ_i là tín hiệu học được định nghĩa trong công thức trên.

Thuật toán lan truyền ngược sai số được xây dựng như sau:

Xét một mạng nơron truyền thẳng có Q lớp, q = 1, 2, …, Q, và gọi net_i và y_i là tín hiệu vào và ra của nơron thứ i trong lớp q. Mạng này có m đầu vào và n đầu ra. Đặt w_{ij} là trọng số của liên kết từ nơron thứ j trong lớp q-1 tới nơron thứ i trong lớp q.

Đầu vào: Một tập các cặp mẫu học $\{(x^{(k)}, d^{(k)}) \mid k= 1, 2, \ldots, p\}$

- Bước 1 (khởi tạo): Chọn một hằng số $\eta > 0$ và Emax (dung sai cho phép). Khởi tạo ngẫu nhiên các trọng số w_{ij} trong khoảng giá trị nhỏ. Đặt E=0 và k=1.
- Bước 2 (thực hiện một quá trình lặp cho việc huấn luyện mạng) :
  + Sử dụng mẫu học thứ k ;

+ Tại lớp vào (q =1), với mọi i ta có: ${}^{q}y_i = {}^{1}y_i = x^{(k)}_i$

- Bước 3 (lan truyền tín hiệu từ lớp vào tới lớp ra)

$$ {}^{q}y_i = g({}^{q}net_i) = g\left(\sum_j {}^{q}w_{ij}\,{}^{q-1}y_i\right) $$

- Bước 4 (xác định tín hiệu lỗi ${}^{Q}\delta_i$ tại lớp ra)

$$ E = \frac{1}{2}\sum_{i=1}^{n}\left(d_i^{(k)} - {}^{Q}y_i\right)^2 + E $$

$$ {}^{Q}\delta_i = \left(d_i^{(k)} - {}^{Q}y_i\right)g'({}^{Q}net_i) $$

- Bước 5 (lan truyền ngược sai số)

Lan truyền ngược sai số để điều chỉnh các trọng số và tính toán tín hiệu lỗi ${}^{q-1}\delta_i$ cho các lớp trước :

$$ \Delta_q w_{ij} = \eta .\, {}_q\delta_i .\, {}_{q-1}y_i $$

$$ {}_q w_{ijnew} = {}_q w_{ijold} + \Delta_q w_{ij} $$

$$ {}_{q-1}\delta_i = g'({}_{q-1}net_i)\sum_j {}_q w_{ij}\,{}_q\delta_i \quad \text{với q=Q,Q-1,…2} $$

- Bước 6 (kiểm tra điều kiện lặp)

If (k<p) then

k=k+1 ;

goto 2 ;

End if

- Bước 7 (kiểm tra lỗi tổng cộng hiện thời đã chấp nhận được chưa)

If (E > E_{max}) then

E =0;

k = 1;

Else

{kết thúc quá trình học và đưa ra bộ trọng số cuối cùng}

End if

Mỗi lần toàn bộ tập mẫu học được lan truyền qua mạng được gọi là một epoch. Số epoch phụ thuộc vào từng trường hợp cụ thể và sự khởi tạo ban đầu. Có trường hợp

thuật toán phải sau hàng chục nghìn epoch mới hội tụ tới lời giải. Nếu tham số khởi tạo không phù hợp có thể làm cho quá trình học không đạt kết quả mong muốn.

Đối với mỗi epoch ta tính sai số trung bình của mạng theo công thức sau:

$$RMS = \sqrt{\frac{\sum_{k=1}^{p}\sum_{i=1}^{n}(d_i - y_i)^2}{p.n}}$$

Trong đó p là số mẫu được dùng để luyện mạng, n là số biến của véc-tơ đầu ra. Sai số RMS thường được dùng để đánh giá kết quả học của mạng nơ ron

Một số yếu tố ảnh hưởng đến quá trình học theo phương pháp lan truyền ngược c của mạng nơron.

Khởi tạo các trọng số

Các giá trị được khởi tạo ban đầu cho các trọng số trong mạng lan truyền ngược sai số ảnh hưởng rất lớn đến kết quả học cuối cùng của mạng. Các giá trị này thường được khởi tạo ngẫu nhiên trong phạm vi giá trị tương đối nhỏ. Thông thường hàm truyền sử dụng cho mạng MLP là hàm sigmoid, do vậy nếu ta chọn các giá trị trọng số khởi tạo lớn thì các hàm này có thể bão hoà ngay từ đầu và dẫn tới hệ thống có thể bị tắc ngay tại một cực tiểu địa phương hoặc tại một vùng bằng phẳng nào đó gần điểm xuất phát.

Hằng số học

Hằng số học η cũng là một yếu tố quan trọng ảnh hưởng đến hiệu quả và độ hội tụ của giải thuật lan truyền ngược sai số. Không có hằng số η phù hợp cho tất cả các bài toán khác nhau. Hằng số học này thường được chọn bằng thực nghiệm cho mỗi bài toán ứng dụng cụ thể bằng phương pháp thử sai.

Trong nhiều ứng dụng thực tế cho thấy một hằng số học có thể phù hợp ở thời điểm bắt đầu của quá trình học nhưng lại không phù hợp với giai đoạn sau của quá trình học. Do đó, có một phương pháp hiệu quả hơn đó là sử dụng hằng số học thích nghi. Một cách xử lý đơn giản cho vấn đề này đó là kiểm tra xem các trọng số mới có làm giảm hàm giá hay không, nếu không thì có thể các trọng số đã vượt quá xa vùng cực tiểu và như vậy hằng số η cần phải giảm. Trái lại, nếu sau vài vòng lặp, hàm giá

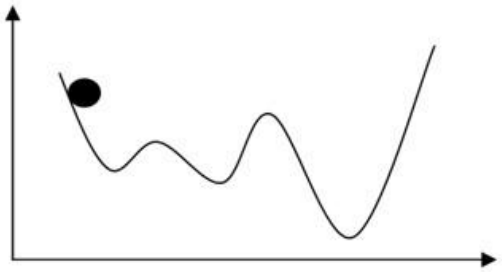
liên tục giảm thì ta có thể thử tăng hằng số η để đẩy nhanh hơn tốc độ hội tụ đến giá trị cực tiểu

Hằng số quán tính

Tốc độ học của giải thuật Lan truyền ngược sai số có thể rất chậm nếu hằng số học nhỏ, nhưng nếu hằng số học lớn thì nó lại có thể gây ra sự dao động lớn trong quá trình tìm giá trị cực tiểu theo phương pháp giảm gradient. Để giải quyết vấn đề này người ta thường thêm thành phần quán tính vào các phương trình hiệu chỉnh trọng số như sau:

$\Delta w(t) = -\eta \nabla E(t) + \alpha \Delta w(t-1)$ với α là hằng số quán tính, $\alpha \in [0, 1]$

Nhờ thành phần này, quá trình học có thể vượt qua điểm cực tiểu địa phương để tìm đến điểm cực tiểu toàn cục, đồng thời thành phần quán tính cũng ngăn cản sự thay đổi đột ngột của các trọng số theo hướng khác với hướng mà lời giải đang di chuyển đến.



Hình 10: Minh họa về ý nghĩa của quán tính trong thực tế

Hàm giá

Trong phần nghiên cứu trên hàm giá được chọn là hàm bình phương sai số.

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(d_i - y_i)^2$$

Tuy nhiên, nó có thể được thay thế bằng một hàm F(y,d) bất kỳ có đạo hàm và đạt cực tiểu khi hai đối số d_i và y_i bằng nhau. Thông thường hàm giá được chọn có dạng:

$$E(w) = \frac{1}{p}\sum_{i=1}^{n}(d_i - y_i)^p$$

Với $1 \leq p \leq \infty$, như vậy khi chọn p = 2 ta có hàm giá là hàm bình phương sai số như đã xét ở trên.

Trong đồ án lần này, để đề phòng trường hợp lặp quá nhiều vòng, chúng em đã giới hạn lại thành 100000 vòng lặp thay vì lặp đến khi hàm giá đạt giá trị mong muốn, Tuy nhiên, điều này cũng sẽ không ảnh hưởng nhiều đến kết quả học của mạng vì 100000 vòng cũng đã làm cho hàm giá đạt giá trị đủ bé với tập dữ liệu nhỏ.

3.3.2 Chi tiết thuật toán lan truyền ngược sai số

```
void learn(double input[][INPUT_COUNT], double outputReal[], double learnRate,
           double inputWeights[][HIDDEN_COUNT], double hiddenWeights[], double hiddenBias[],
           double *outputBias){
    double inputWeightsIn[INPUT_COUNT][HIDDEN_COUNT], hiddenWeightsIn[HIDDEN_COUNT],
           hiddenBiasIn[HIDDEN_COUNT], outputBiasIn;
    for (int i=0; i<HIDDEN_COUNT; i++){
        for (int j=0; j<INPUT_COUNT; j++) inputWeightsIn[j][i] = inputWeights[j][i];
        hiddenWeightsIn[i] = hiddenWeights[i];
        hiddenBiasIn[i]=hiddenBias[i];
    }
    outputBiasIn=*outputBias;
    double error, hiddenError[HIDDEN_COUNT];
    int epoch=0;
    while (epoch++<100000){
        for (int t=0; t<TRAINING_SET; t++){
            double hiddenLayer[HIDDEN_COUNT], output=0;
            for (int i=0; i<HIDDEN_COUNT; i++) hiddenLayer[i]=0;

            for (int j=0; j<HIDDEN_COUNT; j++){
                for (int i=0; i<INPUT_COUNT; i++)
                    hiddenLayer[j]+=inputWeights[i][j]*input[t][i];   // Tinh mot node cua hidden layer
                hiddenLayer[j]+=hiddenBias[j];
                hiddenLayer[j]=sigmoid(hiddenLayer[j]);     // Dua qua ham sigmoid
                output+=hiddenLayer[j]*hiddenWeights[j];
            }
            output+=(*outputBias);
            output=sigmoid(output);    // Tinh output hien tai

            error = (outputReal[t] - output)*dSigmoid(output);    // Tinh sai so
             // Lan truyen tin hieu loi
            for (int i=0; i<HIDDEN_COUNT; i++)
                hiddenError[i] = error*hiddenWeights[i]*dSigmoid(hiddenLayer[i]);
            // Tinh lai weight cua input layer va hidden layer
            for (int j=0; j<HIDDEN_COUNT; j++){
                for (int i=0; i<INPUT_COUNT; i++)
                    inputWeights[i][j] += learnRate*hiddenError[j]*input[t][i]/TRAINING_SET;
                hiddenWeights[j] += learnRate*error*hiddenLayer[j]/TRAINING_SET;
            }

            (*outputBias) += learnRate*error/TRAINING_SET;

            for (int i=0; i<HIDDEN_COUNT; i++) hiddenBias[i] += learnRate*hiddenError[i]/TRAINING_SET;
        }
    }
}
```

Hình 11: Chi tiết thuật toán lan truyền ngược sai số

4. CHƯƠNG TRÌNH VÀ KẾT QUẢ

4.1 Tổ chức chương trình

- Các thư viện có sẵn trong Dev-C++.
- Định nghĩa và xây dựng các hàm
- File dữ liệu đầu vào được xây dựng sẵn
- Chương trình chính

4.2 Ngôn ngữ cài đặt

- Đồ án được nhóm xây dựng bằng ngôn ngữ C.

4.3 Kết quả thực hiện

4.3.1 Giao diện chính của chương trình

```
->>>Input the name of the file you want to open: input
File read successfully.

~~~~/~~~/~~/~/// MENU ///~/~~/~~~/~~~~
1. Train neural network.
2. Validate neural network.
3. Predict using neural network.
4. Exit.
~~~~/~~~/~~/~///~~~~~~~///~/~~/~~~/~~~~
Enter your selection:
```

Hình 12: Giao diện chính của chương trình

Đầu tiên, chương trình sẽ yêu cầu nhập tên file chứa data mà chúng ta cần, sau khi nhập file thành công, chương trình sẽ in ra menu cho người dùng lựa chọn.

4.3.2 Các kết quả thực thi của chương trình

```
Enter your selection: 1
Input weights before:
0.442427   0.00259407   0.114322   0.755364   0.344615
0.216926   0.64803   0.805933   0.693014   0.306192
0.0724815   0.168401   0.559069   0.263588   0.285745
0.0166021   0.318369   0.771294   0.313913   0.553697

Input weights after:
11.1165   1.33582   1.20449   1.44326   1.21485
-1.89813   0.680927   0.491549   0.948677   0.109451
-0.704263   1.95835   1.59604   0.26749   -0.00861438
-0.350881   2.95826   2.55998   0.806593   0.422388


Hidden bias after:
-0.0389944   -0.619697   -0.244071   -0.457566   -0.11371

Output bias before:
0.20304

Output bias after:
-1.39768

Press any button to continue.
```

Hình 13: Khi thực hiện chức năng 1

Khi thực hiện chức năng 1, chương trình sẽ lấy dữ liệu từ file đã chọn và thực hiện quá trình huấn luyện mạng. Sau khi huấn luyện xong, chương trình sẽ trả về các thông số để chúng ta có thể nhận thấy được sự thay đổi.

```
Enter your selection: 2
Predicted sale: 0.0616542
Actual sale: 0.701068

Predicted sale: 0.358851
Actual sale: 0.343082

Predicted sale: 0.304472
Actual sale: 0.290387

Press any button to continue.
```

Hình 14: Khi thực hiện chức năng 2

Khi thực hiện chức năng 2, chương trình trả về kết quả dự đoán và kết quả thực tế của doanh thu 3 ngày cuối cùng trong tập dữ liệu đầu vào, từ đó chúng ta có thể đánh giá được kết quả học của mạng nơ ron.

```
Enter your selection: 3
Predicted sale of the 31st day: 0.13501

Press any button to continue.
```

Hình 15: Khi thực hiện chức năng 3

Khi thực hiện chức năng 3, chương trình trả về kết quả dự đoán của ngày tiếp theo.

4.3.3 Nhận xét đánh giá kết quả

Dự trên kết quả thực thi chương trình, chúng ta có thể thấy rằng mạng nơ ron sau khi huấn luyện đã có thể dự đoán gần chính xác một số ngày, tuy nhiên vẫn có sự sai lệch nhất định. Điều này xảy ra có thể do tập dữ liệu dùng để huấn luyện ít hoặc do yếu tố đặc biệt nào đó ảnh hưởng đến doanh thu của ngày hôm đó gây sai lệch cho việc dự đoán.

5 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 Kết luận

Nhìn chung chúng em đã xây dựng thành công mô hình mạng nơ ron để dựng báo doanh thu bán hàng. Tuy nhiên, mô hình vẫn còn khá đơn giản với 1 lớp đầu vào, 1 lớp ẩn và 1 lớp đầu ra, và kết quả dự đoán vẫn còn sai lệch ít nhiều vì đó là điều tất yếu của bất cứ quá trình dự đoán nào.

5.2 Hướng phát triển

Ở đồ án lần này, chúng em chỉ xây dựng một mô hình đơn giản và chỉ mới thử nghiệm trên tập dữ liệu nhỏ. Trong tương lai khi tiếp tục tìm hiểu kiến thức và thu thập được tập dữ liệu lớn hơn, chúng em sẽ cải tiến và phát triển thêm cho mô hình để đưa ra dự đoán chính xác nhất.

TÀI LIỆU THAM KHẢO

[1] Sales Prediction through Neural Networks for a Small Dataset - https://www.researchgate.net/publication/324576461_Sales_Prediction_through_Neural_Networks_for_a_Small_Dataset
[2] Ứng dụng mạng nơ ron nhân tạo vào bài toán dự báo thủy văn – https://imech.ac.vn/upload/NewsImage/2021/1/12/ung-dung-mang-noron-nhan-tao-vao-bai-toan-du-bao-thuy-van.pdf
[3] Multi-layer Perceptron và Backpropagation - https://machinelearningcoban.com/2017/02/24/mlp/

PHỤ LỤC

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include <string.h>
int INPUT_COUNT=4, HIDDEN_COUNT=5, OUTPUT_COUNT=1,
TRAINING_SET, DATASET_SIZE, VALIDATION_SET;
double* dataset;
double sigmoid(double x){
   return 1.0/(1+exp(-x));
}
double dSigmoid(double x){
   return (x)*(1-x);
}
void getData(){
   FILE* f;
   char name[100];
   printf("\n->>>Input the name of the file you want to open: ");
   scanf("%s",name);
   strcat(name,".txt");
   f=fopen(name, "r");
   if (NULL == f) {
      printf("File can't be opened.\n");
      exit(1);
   }
```

```
        else printf("File read successfully.\n\n");
        fscanf(f, "%d", &DATASET_SIZE);
        VALIDATION_SET=DATASET_SIZE/10;
        TRAINING_SET=DATASET_SIZE-VALIDATION_SET-INPUT_COUNT;
        dataset = (double*)calloc(DATASET_SIZE,sizeof(double));
        fscanf(f, "%lf", &dataset[0]);
        for (int i=1; i<DATASET_SIZE; i++){
           fscanf(f, ",%lf", &dataset[i]);
        }
        fclose(f);
    }

    double predict(double input[], double inputWeights[][HIDDEN_COUNT],
double hiddenWeights[],
                double hiddenBias[], double *outputBias){
        double hiddenLayer[HIDDEN_COUNT], output=0;
        for (int i=0; i<HIDDEN_COUNT; i++) hiddenLayer[i]=0;

        for (int j=0; j<HIDDEN_COUNT; j++){
           for (int i=0; i<INPUT_COUNT; i++)
hiddenLayer[j]+=inputWeights[i][j]*input[i];   // Tinh mot node cua hidden layer
           hiddenLayer[j]+=hiddenBias[j];
           hiddenLayer[j]=sigmoid(hiddenLayer[j]);     // Dua qua ham sigmoid
           output+=hiddenLayer[j]*hiddenWeights[j];
        }
        output+=(*outputBias);
        output=sigmoid(output);
        return output;
```

```
    }

    void learn(double input[][INPUT_COUNT], double outputReal[], double
learnRate, double inputWeights[][HIDDEN_COUNT],
            double hiddenWeights[], double hiddenBias[], double *outputBias){
        double inputWeightsIn[INPUT_COUNT][HIDDEN_COUNT],
hiddenWeightsIn[HIDDEN_COUNT], hiddenBiasIn[HIDDEN_COUNT],
outputBiasIn;
        for (int i=0; i<HIDDEN_COUNT; i++){
            for (int j=0; j<INPUT_COUNT; j++) inputWeightsIn[j][i] =
inputWeights[j][i];
            hiddenWeightsIn[i] = hiddenWeights[i];
            hiddenBiasIn[i]=hiddenBias[i];
        }
        outputBiasIn=*outputBias;

        double error, hiddenError[HIDDEN_COUNT];
        int epoch=0;
        while (epoch++<100000){
            for (int t=0; t<TRAINING_SET; t++){

                double hiddenLayer[HIDDEN_COUNT], output=0;
                for (int i=0; i<HIDDEN_COUNT; i++) hiddenLayer[i]=0;

                for (int j=0; j<HIDDEN_COUNT; j++){
                    for (int i=0; i<INPUT_COUNT; i++)
hiddenLayer[j]+=inputWeights[i][j]*input[t][i];   // Tinh mot node cua hidden layer
                    hiddenLayer[j]+=hiddenBias[j];
```

```
                hiddenLayer[j]=sigmoid(hiddenLayer[j]);     // Dua qua ham sigmoid
                output+=hiddenLayer[j]*hiddenWeights[j];
            }
            output+=(*outputBias);
            output=sigmoid(output);    // Tinh output hien tai

            error = (outputReal[t] - output)*dSigmoid(output);    // Tinh sai so

            for (int i=0; i<HIDDEN_COUNT; i++) hiddenError[i] =
error*hiddenWeights[i]*dSigmoid(hiddenLayer[i]);    // Lan truyen tin hieu loi

            for (int j=0; j<HIDDEN_COUNT; j++){
                for (int i=0; i<INPUT_COUNT; i++) inputWeights[i][j] +=
learnRate*hiddenError[j]*input[t][i]/TRAINING_SET;   // Tinh lai weight cua input
layer
                hiddenWeights[j] +=
learnRate*error*hiddenLayer[j]/TRAINING_SET;     // Tinh lai weight cua hidden
layer
            }

            (*outputBias) += learnRate*error/TRAINING_SET;

            for (int i=0; i<HIDDEN_COUNT; i++) hiddenBias[i] +=
learnRate*hiddenError[i]/TRAINING_SET;
        }
    }

    printf("Input weights before:\n");
    for (int i=0; i<INPUT_COUNT; i++){
```

```
            for (int j=0; j<HIDDEN_COUNT; j++) printf("%g   ",
inputWeightsIn[i][j]);
            printf("\n");
        }
        printf("\n");
        printf("Input weights after:\n");
        for (int i=0; i<INPUT_COUNT; i++){
            for (int j=0; j<HIDDEN_COUNT; j++) printf("%g   ", inputWeights[i][j]);
            printf("\n");
        }
        printf("\n");

        printf("Hidden weights before:\n");
        for (int i=0; i<HIDDEN_COUNT; i++) printf("%g   ", hiddenWeightsIn[i]);
        printf("\n\n");
        printf("Hidden weights after:\n");
        for (int i=0; i<HIDDEN_COUNT; i++) printf("%g   ", hiddenWeights[i]);
        printf("\n\n");

        printf("Hidden bias before:\n");
        for (int i=0; i<HIDDEN_COUNT; i++) printf("%g   ", hiddenBiasIn[i]);
        printf("\n\n");
        printf("Hidden bias after:\n");
        for (int i=0; i<HIDDEN_COUNT; i++) printf("%g   ", hiddenBias[i]);
        printf("\n\n");

        printf("Output bias before:\n%g\n\nOutput bias after:\n%g\n\n", outputBiasIn,
*outputBias);
```

```
}

void giveRandWeights(double inputWeights[][HIDDEN_COUNT], double hiddenWeights[], double hiddenBias[], double *outputBias){
    srand(time(0));
    for (int i=0; i<INPUT_COUNT; i++){
        for (int j=0; j<HIDDEN_COUNT; j++) inputWeights[i][j] = (double)rand()/(double)RAND_MAX;
    }
    for (int i=0; i<HIDDEN_COUNT; i++){
        hiddenWeights[i] = (double)rand()/(double)RAND_MAX;
        hiddenBias[i] = (double)rand()/(double)RAND_MAX;
    }
    (*outputBias) = (double)rand()/(double)RAND_MAX;
}

void printMenu(){
    printf("~~~~/~~~/~~/~/// MENU ///~/~~/~~~/~~~~\n");
    printf("1. Train neural network.\n2. Validate neural network.\n3. Predict using neural network.\n4. Exit.\n");
    printf("~~~~/~~~/~~/~///~~~~~~///~/~~/~~~/~~~~\n");
    printf("Enter your selection: ");
}

int main(){
    system("COLOR 70");
    getData();
```

```
        double inputWeights[INPUT_COUNT][HIDDEN_COUNT],
hiddenWeights[HIDDEN_COUNT], hiddenBias[HIDDEN_COUNT], outputBias,
learnRate=0.2,
           datamax=-INT_MAX;
        for (int i=0; i<DATASET_SIZE; i++)
           datamax=(datamax>dataset[i])?datamax:dataset[i];


        giveRandWeights(inputWeights,hiddenWeights,hiddenBias,&outputBias);


        int index=0;
        double input[TRAINING_SET][INPUT_COUNT],
outputReal[TRAINING_SET];
        for (int i=0; i<TRAINING_SET; i++){
           for (int j=0; j<INPUT_COUNT; j++)
              input[i][j]=dataset[index++]/datamax;
           index-=3;
        }
        index=INPUT_COUNT;
        for (int i=0; i<TRAINING_SET; i++){
           outputReal[i]=dataset[index++]/datamax;
        }
        char temp;
        double ans, input0[INPUT_COUNT];
        do{
           fflush(stdin);
           printMenu();
           scanf("%c", &temp);
           switch(temp){
```

```
            case '1':
learn(input,outputReal,learnRate,inputWeights,hiddenWeights,hiddenBias,&outputBia
s);
               break;
            case '2':
               index=TRAINING_SET;
               for (int i=1; i<=VALIDATION_SET; i++){
                  for (int i=0; i<INPUT_COUNT; i++)
input0[i]=dataset[index++]/datamax;

ans=predict(input0,inputWeights,hiddenWeights,hiddenBias,&outputBias);
                  printf("Predicted sale: %g\n", ans);
                  printf("Actual sale: %g\n\n", dataset[index]/datamax);
                  index-=INPUT_COUNT-1;
               }
               break;
            case '3':
               index=DATASET_SIZE-INPUT_COUNT;
               for (int i=0; i<INPUT_COUNT; i++)
input0[i]=dataset[index]/datamax;

ans=predict(input0,inputWeights,hiddenWeights,hiddenBias,&outputBias);
               printf("Predicted sale of the 31st day: %g\n\n", ans);
               break;
            case '4': system("COLOR 07");
               return 0;
            default: printf("Please choose a number from 1-4.\n");
               break;
         }
```

```
            printf("Press any button to continue.\n\n");
            getch();
        } while (temp!='4');
    }
```