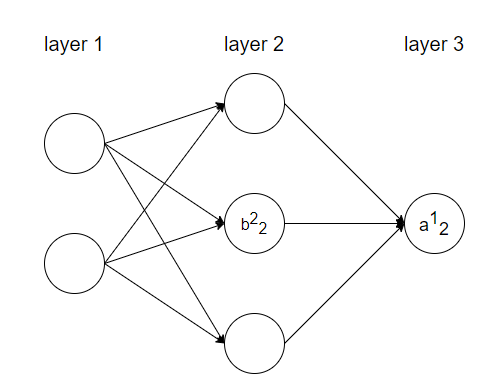
Đây là ký hiệu được sử dụng trong Neural networks. Trong đó w là trọng lượng kết nối của các nút khi quá trọng học tập diễn ra, k là vị trí neurol thuộc(l-1) tầng đến neuron thứ k thuộc l tầng



Nhìn vào ví dụ trên chúng ta đặt bjl trong đó là vị trí của neuron thứ j trong l lớp. Tương tự thì alj

Với những cách ký hiệu trên

**C1:**  Trình bày lý thuyết về thuật toán lan truyền ngược (backpropagation) để học tham số của mạng neural truyền thẳng (feed forward neural network).

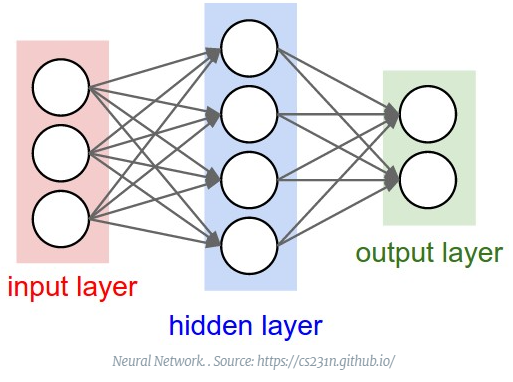
1. **Mạng neuron là gì và các đặc trưng của nó:**

Mạng thần kinh truyền thẳng (hay còn gọi là mạng nơ-ron truyền thẳng, tiếng Anh: feedforward neural network, viết tắt là FNN) là một mạng thần kinh nhân tạo trong đó kết nối giữa các nút không hình thành một chu trình. Mạng neuron truyền thẳng là loại mạng neuron nhân tạo đầu tiên và đơn giản nhất được phát minh ra. Trong mạng này, thông tin di chuyển chỉ một chiều - hướng đến - từ các nút đầu vào, thông qua các nút ẩn (nếu có) và đi đến các nút đầu ra. Không có chu trình (chu kỳ) hoặc vòng lặp trong mạng.

* Các đặc trưng cơ bản của mạng neuron:
  + Liên kết giữa các đơn vị. Xét tổng quát, mỗi liên kết được định nghĩa bởi một trọng số wjk cho ta biết hiệu ứng mà tín hiệu của đơn vị j có trên đơn vị k
  + Gồm một tập các đơn vị xử lý (các noron nhân tạo)
  + Một luật lan truyền quyết định cách tính tín hiệu ra của từng đơn vị từ đầu vào của nó.

1. Kiến trúc mạng Neuron:

Mạng Neuron là sự kết hợp của các tầng bằng percepreon hay còn được gọi là perceptron đa tầng (multilayer perceptron)

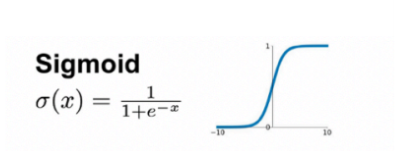


* Một mạng Neuron sẽ có 3 kiểu tầng:
  + Tầng vào (input layer): Là tầng bên trái cùng của mạng thể hiện cho các đầu vào của mạng.
  + Tầng ra (output layer): Là tầng bên phải cùng của mạng thể hiện cho các đầu ra của mạng.
  + Tầng ẩn (hidden layer): Là tầng nằm giữa tầng vào và tầng ra thể hiện cho việc suy luận logic của mạng.

Một mạng Neuron chỉ có 1 tầng vào và 1 tầng ra như có thể có nhiều tầng ẩn

* Mỗi node trong hidden layer và output layer:
  + Liên kết với tất cả các node ở layer trước đó và có hệ số w riêng
  + Mỗi node có 1 hệ số bias riêng
  + Đều diễn ra 2 bước tính: tính tổng linear và áp dụng activation function (ở đây ta dùng hàm sigmoid)

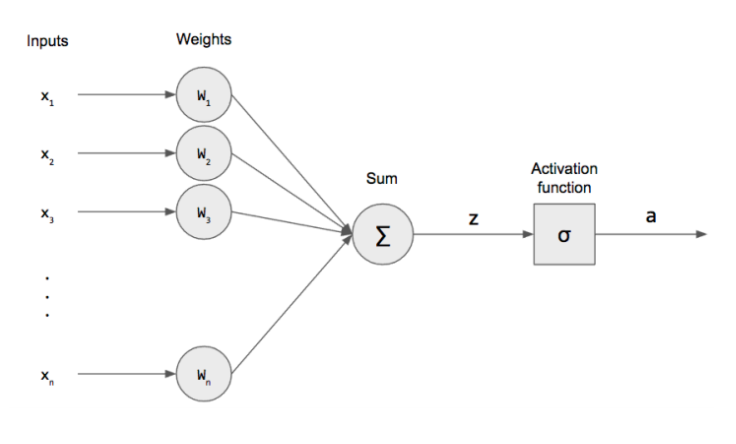
1. Hàm sigmoid:



Hàm sigmoid là hàm số nếu đầu vào lớn hàm số sẽ cho đầu ra gần với giá trị 1. Còn đầu vào nhỏ (rất âm), hàm số sẽ cho đầu ra gần với giá trị 0

1. Logistic regression:

Logistic regression là mô hình Neural network đơn giản nhất chỉ với input layer và output layer.



Mô hình của logistic regression có dạng *y*^​=*σ*(*w*0​+*w*1​∗*x*1​+*w*2​∗*x*2​). Ở đây ta có thể nhìn thấy 2 bước:

Tính tổng linear: *z*=1∗*w*0​+*x*1​∗*w*1​+*x*2​∗*w*2​

Áp dụng hamg sigmoid function: *y*^​=*σ(z)*

1. Mean Square Error:

Mean Square Error (Sai số bình phương) là hàm mất mát được sử dụng phổ biến nhất để hồi quy. Sự mất mát là dữ liệu giám sát trung bình của sự khác biệt bình phương giữa giá trị thực và giá trị dự đoán, hoặc viết nó dưới dạng công thức.



Trong đó Y^ là một vector của n dự báo

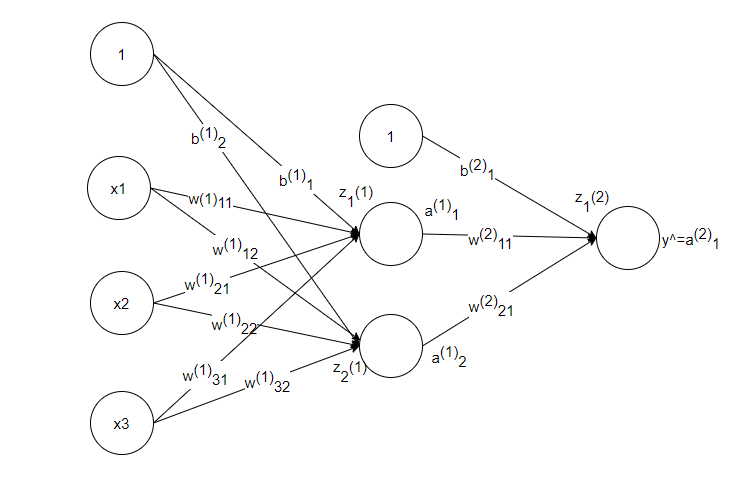
Y là vector của các giá trị quan sát được

1. Ký hiệu:



Ta ký hiệu:

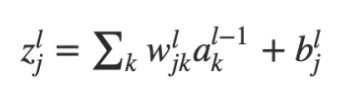
* + Số node trong hidden layer thứ j là l(j)
  + Ma trận W(l) kích thức l(l-1) \* ll là ma trận hệ số giữa layer (l-1) và layer l trong đó wjkl là hệ số kết nối từ node thứ j thuộc layer l-1 đến node thứ k của layer l.
  + blj là hệ số bias của các node j trong layer thứ l
  + zlj là giá trị các node trong layer l sau bước tính tổng linear
  + akl-1 là giá trị của node trong layer l sau khi áp dụng activation function



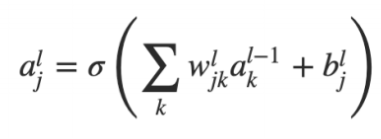
* Nhìn vào mô hình trên ta thấy:
  + Mô hình trên có 2 layer (số lượng layer không tính input layer)
  + Mô hình: 3-2-1, nghĩa là 3 node trong input layer, 2 node trong hidden layer, và output layer có 1 node
  + Input layer và hidden layer luôn thêm 1 node để tính bias cho layer sau. Nhưng không tính vào số lượng node trong layer
  + Ở mỗi node trong hidden layer và output layer đề thực hiện 2 bước: tính tổng linear và áp dụng activation function
  + Các hệ số bias ký hiệu trong hình

1. Các bước tính toán forward computation

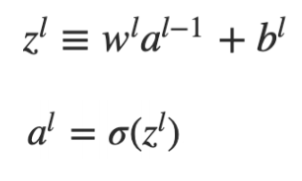
* Đầu tiên ta cần tính tổng linear của từng node:



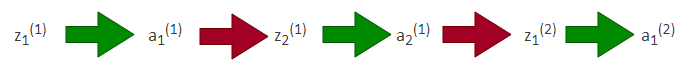
* Tiếp đến ta áp dụng activation function (ở đây là hàm sigmoid) với tham số là tổng linear



* Sau đó ta tính output

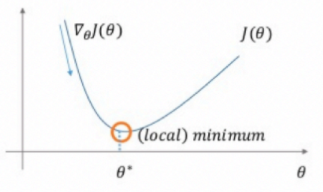


* Ví dụ Forward Computation dựa theo hình vẽ ở trên:
  + z1(1) = b1(1) + x1 \* w11(1) + x2 \* w21(1) + x3 \* w31(1)
  + a1(1) = *σ*(z1(1) ​)
  + z2(1) = b2(1) + x1 \* w12(1) + x2 \* w22(1) + x3 \* w32(1)
  + a2(1) = *σ*(z2(1) ​)
  + z1(2) = b1(2) + a1(1) \* w11(2) + a2(1) \* w21(2)
  + a1(2) = *σ*(z1(2) ​) = y^

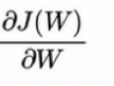


1. Các bước tính toán trong gradient descent

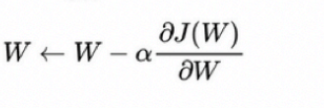
* Thuật toán gradient descent (lan truyền ngược) ban đầu được giới thiệu vào những năm 1970, nhưng tầm quan trọng không được đánh giá đầy đủ cho đến khi một bài báo nổi tiếng năm 1986 của David Rumelhart, Geoffrey Hinton và Ronald Williams:
  + Bài báo mô tả một số mạng nơ-ron nơi hoạt động của quá trình lan truyền ngược nhanh hơn nhiều so với các phương pháp tiếp cận học tập trước đây, giúp bạn có thể sử dụng mạng thần kinh để giải quyết các vấn đề mà trước đây không thể giải quyết được.
  + Ngày nay, thuật toán lan truyền ngược là công cụ học tập trong lĩnh vực thần kinh mạng lưới.
* Gradient descent giúp chúng ta tìm các điểm cực tiểu cục bộ một cách xấp xỉ sau một số vòng lặp



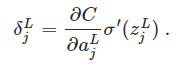
* Các bước tính toán của Gradient descent:
  + Bước khởi tạo: Bắt đầu bằng các hệ số nhỏ, hoặc hệ số ngẫu nhiên hoặc bằng 0: Quá trình này gọi là thiết lập vị trí ban đầu (starting point – điểm khởi đầu, hay initial weight – hệ số khởi tạo).
  + Bước 1: Tính đạo hàm từng phần của Loss function ở vị trí hiện tại đối với từng biến



* + Bước 2: Cập nhật hệ số mới theo hàm cập nhật sau :



* Bốn phương trình cơ bản của lan truyền ngược:
  + Phương trình cho lỗi trong lớp đầu ra δL:



Trong đó ∂C/ ∂aLj, chỉ đo lường mức độ thay đổi nhanh chóng của như một hàm của vị trí node j kích hoạt đầu ra. σ′(zLj)σ đo lường mức độ nhanh chóng của chức năng kích hoạt σ đang thay đổi của zLj

* + Phương trình lỗi δl về lỗi trong lớp tiếp theo δl + 1



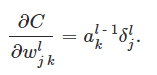
Ta có (wl+1)T là chuyển vị ma trận trọng lượng wl+1 cho (l+1) lớp. Giả sử chúng ta biết lỗi δl+1 tại l +1 lớp. Khi chúng ta áp dụng ma trận trọng lượng chuyển vị, (wl+1)T , chúng ta có thể nghĩ về điều này một cách trực quan khi chuyển lỗi về phía sau qua mạng, cho chúng ta một số loại thước đo lỗi ở đầu ra của l lớp. Sau đó chúng ta lấy Hadamard (Tích nguyên tố của hai vector) ⊙σ′(zl). Điều này di chuyển lỗi trở lại thông qua chức năng kích hoạt trong lớp l, cho chúng ta lỗi δl trong đầu vào có trọng số cho lớp l.

* + Phương trình cho tốc độ thay đổi đối với bất kỳ sai lệch nào trong mạng



Lỗi δlk là chính xác bằng với tốc độ thay đổi ∂C/ ∂blj

* + Phương trình cho tốc độ thay đổi với bất kỳ trọng số nào trong mạng



* Thuật toán trong Gradient descent:
  + Đầu vào x: Đặt bias tương ứng a1 cho lớp đầu vào
  + Feedforward: Với mỗi l=2,3, … L tính toán zl= wlal-1 + bl và al=σ(zl)
  + Lỗi ở tầng ra δL= Tính toán vectơ δL=∇aC⊙σ′(zL)
  + Sao chép lại lỗi :  Đối với mỗil = L - 1 , L - 2 , … , 2 tính toán δl= ( (wl+1)T δl+1) ⊙σ′(zl)
  + Kết quả là độ dốc của hàm được tính bởi

