## 1. Neural Network là gì?

Linear Classifier: f = Wx

2 - layers Neural Network: $f = W_2 \max(0, W_1 x)$

3 - layers Neural Network: $f = W_3 \max(0, W_2(\max(0, W_1 x))$

$$x \in R^D, W_1 \in R^{H_1 \times D}, W_1 \in R^{H_2 \times H_1}, W_1 \in R^{C \times H_2}$$

Trong đó D là số tham số đầu vào, C là output đầu ra ứng với các class, H là số neurons của mỗi layer

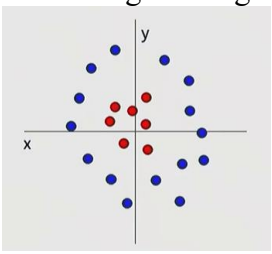
-> Neural Network = Linear + Non-Linear

Neural Network còn được gọi là Fully Connected vì các neuron layer trước kết nối hoàn toàn với các neuron layers sau

### 1.1. Tại sao lại cần Non-Linear mới thành Neural Network?

Nếu không có các Non-Linear thì hàm sẽ như này: $f = W_1 W_2 x = W_3 x$ -> nó sẽ trở lại thành Linear Classifier.

Bên cạnh đó, không phải problem nào cũng có thể giải quyết bằng Linear ví dụ:




Trong ví dụ này không thể có một đường Linear nào phù hợp để có thể phân loại giữa xanh và đỏ.

Vì vậy chúng ta cần có các hàm Non-Linear hay còn được gọi là Activation function.

### 1.2. Các hàm Activation

Các hàm cơ bản thường thấy trong Neural Network

| Tên hàm | Công thức |
|---|---|
| ReLU | $\max(0, x)$ |
| Leaky ReLU | $\max(0.1x, x)$ |
| ELU | x nếu x >= 0 và $\alpha(e^x - 1)$ nếu x < 0 |
| GELU | $x\Phi(x)$ |
| Sigmoid | $\sigma(x) = \frac{1}{1 - e^{-x}}$ |
| Tanh | $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ |
| SiLU | $f(x = x.\sigma(x)$ |

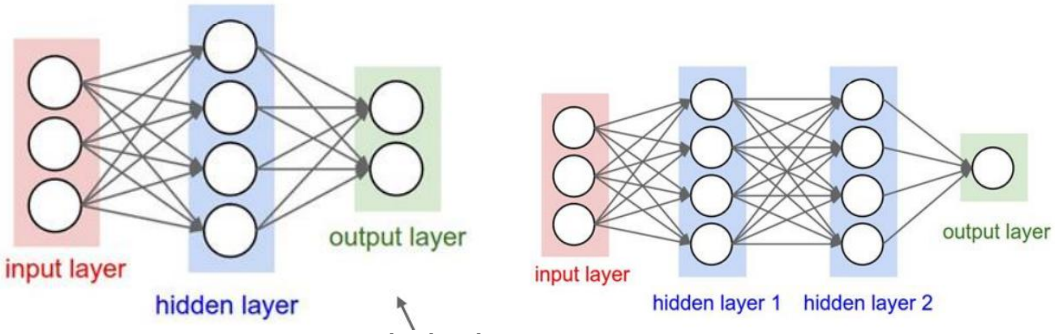
Trong đó, ReLU thường là lựa chọn phổ biến nhất

Tại sao có nhiều Non-Linear function bởi vì nó liên quan tới các vấn đề như có đạo hàm được không hoặc là nó có thể bị Vanishing problem(đạo hàm biến mất) hay không

### 1.3. Kiến trúc Neural Network

Có 1 hidden layer gọi là 1 hidden - layer Neural Net hoặc là 2 layer Neural Net

Có 2 hidden layer gọi là 2 hidden - layer Neural Net hoặc là 3 layer Neural Net




Các w (weight) sẽ là các trọng số kết nối (các mũi tên)
-> Điểm khó: làm sao để chọn lượng layers, số neuron mỗi layer và activation function của neuron đó

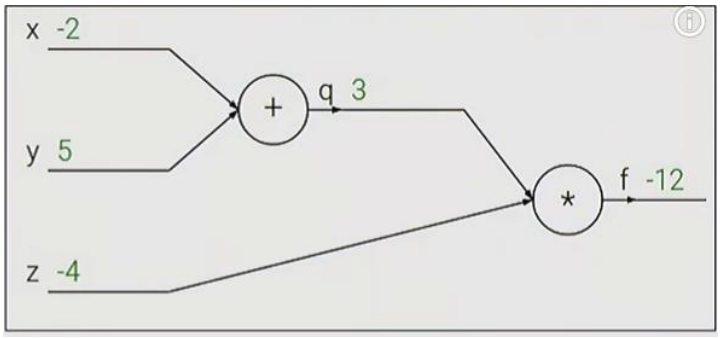
## 2. Backpropagation

### 2.1. Vấn đề

Neural Network có rất nhiều tham số và hàm loss L(W) phụ thuộc vào từng weight -> Không thể viết công thức đạo hàm cho từng layer được -> kiến trúc khác là phải làm lại -> lãng phí
Ý tưởng mới: Backpropagation = áp dụng Chain Rule một cách có hệ thống trên Computational Graph

### 2.2. Computational Graph

Mỗi node là 1 phép toán và mỗi cạnh là 1 dòng dữ liệu
Vd: f = z(x+y)




### 2.3. Chain rule

Chain rule là cách để đạo hàm thông qua các local gradient. Vd trong ví dụ ở 2.2 để đạo hàm biến y:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

Trong đó $\frac{\partial f}{\partial q}$ là upstream gradient (gradient từ phía loss) và $\frac{\partial q}{\partial y}$ là local gradient (đạo hàm của node)