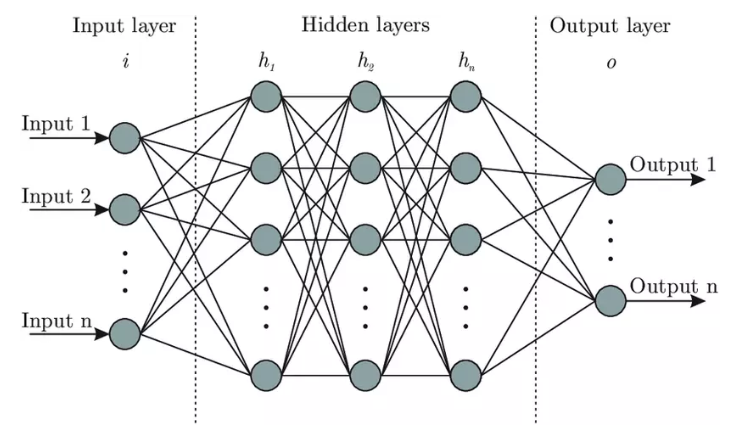
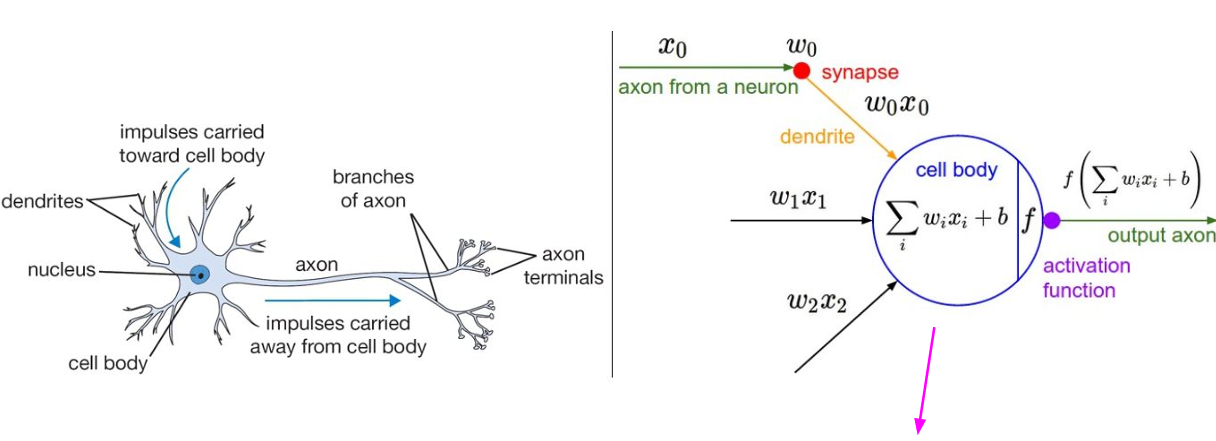
**Tìm hiểu Neural Networks**

**1. Tổng quan về mô hình Deep Learning**



* Layer đầu tiên là input layer, các layer ở giauwx gọi là hidden layer, layer cuối gọi alf output layer. Một mạng Neural Network bắt buộc phải có input layer và output layer, các layer ẩn có thể có hoặc không.
* Các hình tròn trong layer gọi là node (hay còn gọi là neural). Các nốt này liên kết với tất cả các node trước đó và trọng số w riêng. Mỗi node đều có 1 hệ số bias b riêng và sau đó sẽ tính tổng các w với input trước đó rồi áp dụng activation function để truyền tới các node của layer tiếp theo.

**2. Biological Neuron vs Artificial Neuron**



* Ý tưởng của Neural Network là mô phỏng lại quá trình mà não bộ của chúng ta tiếp nhận, xử lý và phân tích dữ liệu. Từ đó giúp máy móc có thể làm được những công việc thay thế được cho con người.
* Nơ-ron là đơn vị cơ bản cấu tạo hệ thống thần kinh và là một phần quan trọng nhất của não. Não chúng ta gồm khoảng 10 triệu nơ-ron và mỗi nơ-ron liên kết với 10.000 nơ-ron khác. Ở mỗi nơ-ron có phần thân (soma) chứa nhân, các tín hiệu đầu vào qua sợi nhánh (dendrites) và các tín hiệu đầu ra qua sợi trục (axon) kết nối với các nơ-ron khác. Hiểu đơn giản mỗi nơ-ron nhận dữ liệu đầu vào qua sợi nhánh và truyền dữ liệu đầu ra qua sợi trục, đến các sợi nhánh của các nơ-ron khác. Mỗi nơ-ron nhận xung điện từ các nơ-ron khác qua sợi nhánh. Nếu các xung điện này đủ lớn để kích hoạt nơ-ron, thì tín hiệu này đi qua sợi trục đến các sợi nhánh của các nơ-ron khác. => Ở mỗi nơ-ron cần quyết định có kích hoạt nơ-ron đấy hay không.
* Trong deep learning chỉ là lấy cảm hứng từ não bộ và cách nó hoạt động, chứ không phải bắt chước toàn bộ các chức năng của nó. Việc chính của chúng ta là dùng mô hình đấy đi giải quyết các bài toán chúng ta cần.

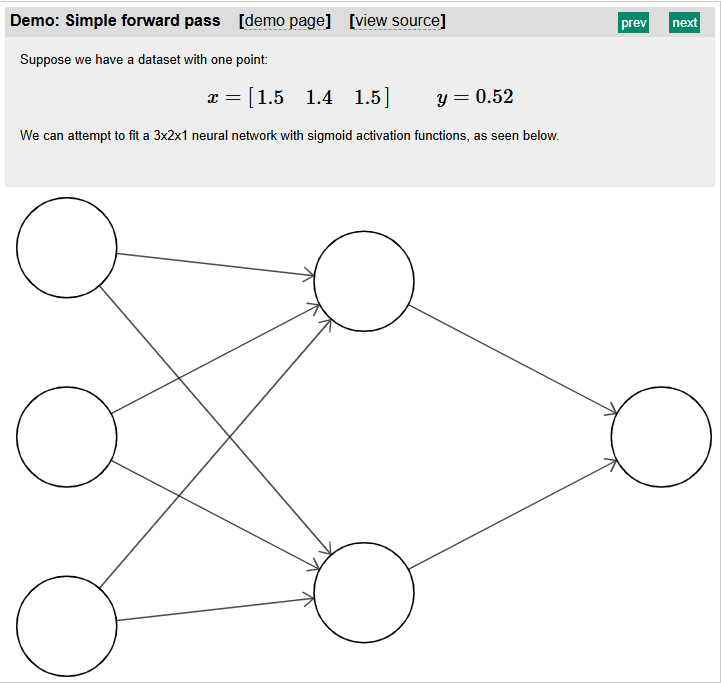
**3. Activation functions**

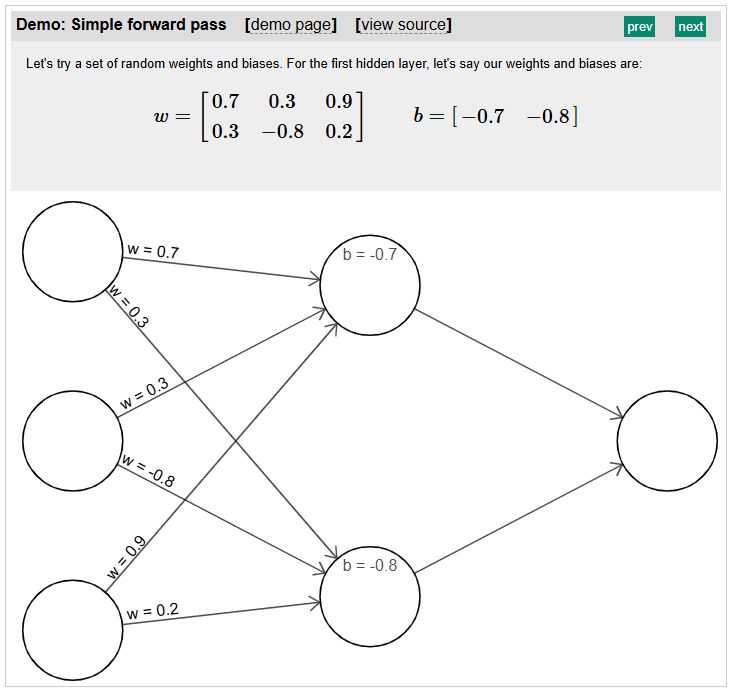
* Hàm kích hoạt (activation function) mô phỏng tỷ lệ truyền xung qua axon của một neuron thần kinh, là những hàm phi tuyến được áp dụng vào đầu ra của các nơ-ron trong tầng ẩn của một mô hình mạng ( có nhiệm vụ là chuẩn hoá output của neura) và được sử dụng làm input data cho tầng tiếp theo
* Một số hàm kích hoạt phổ biến:

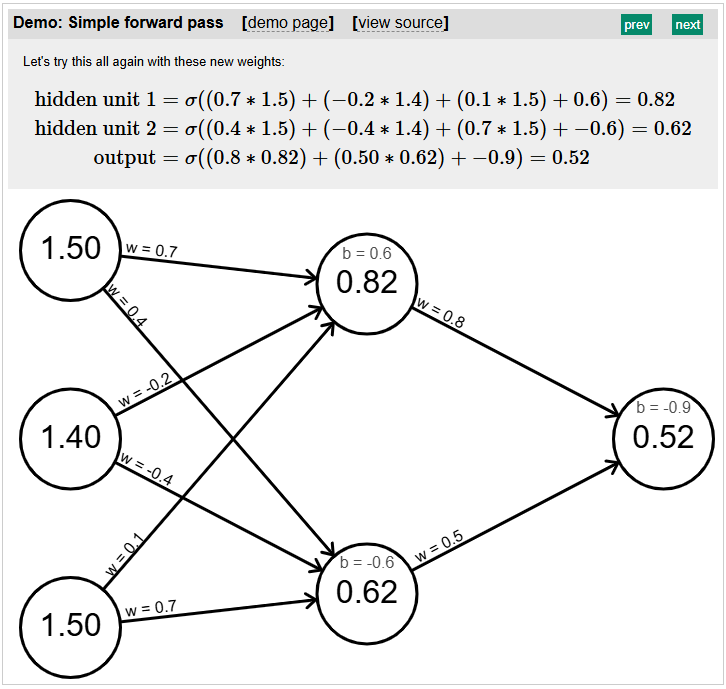
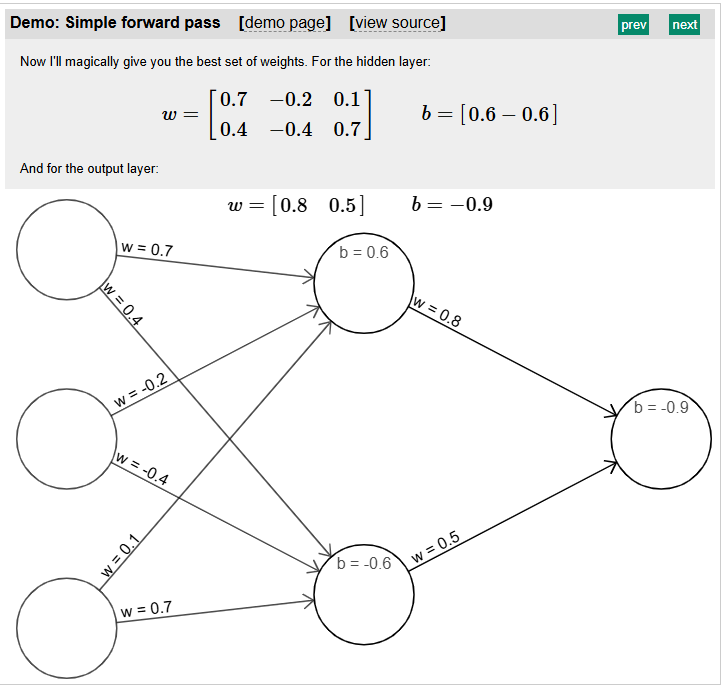
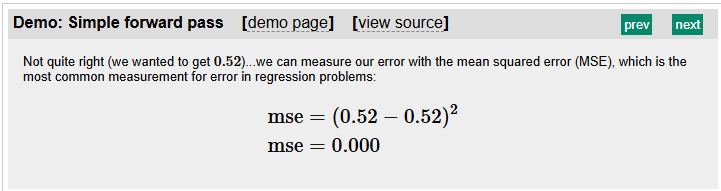
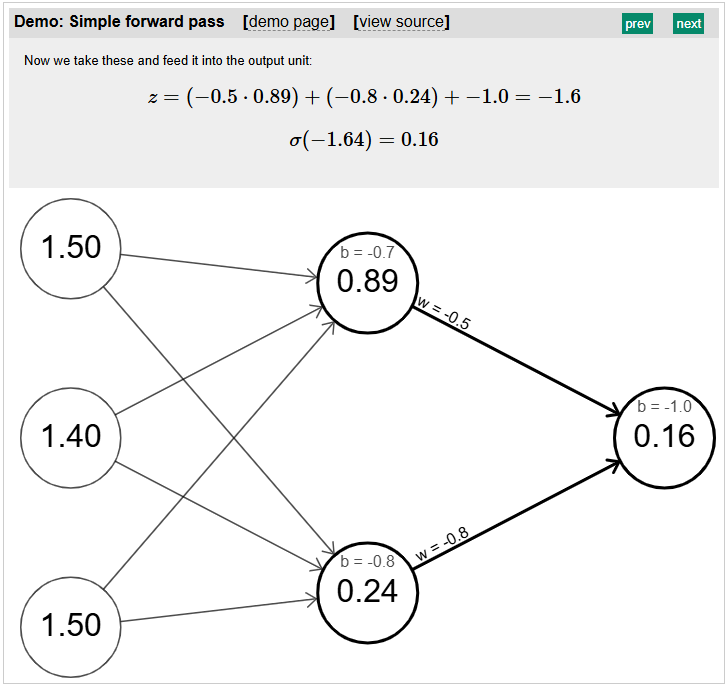
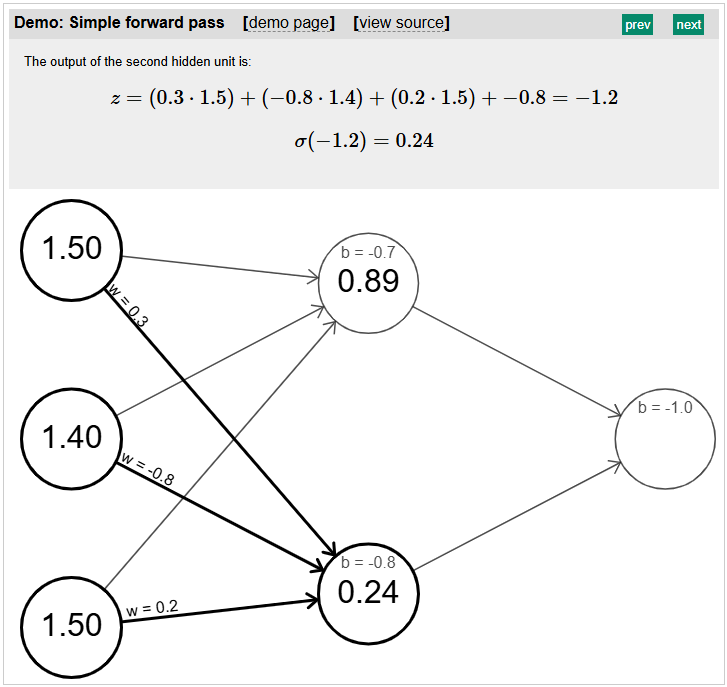
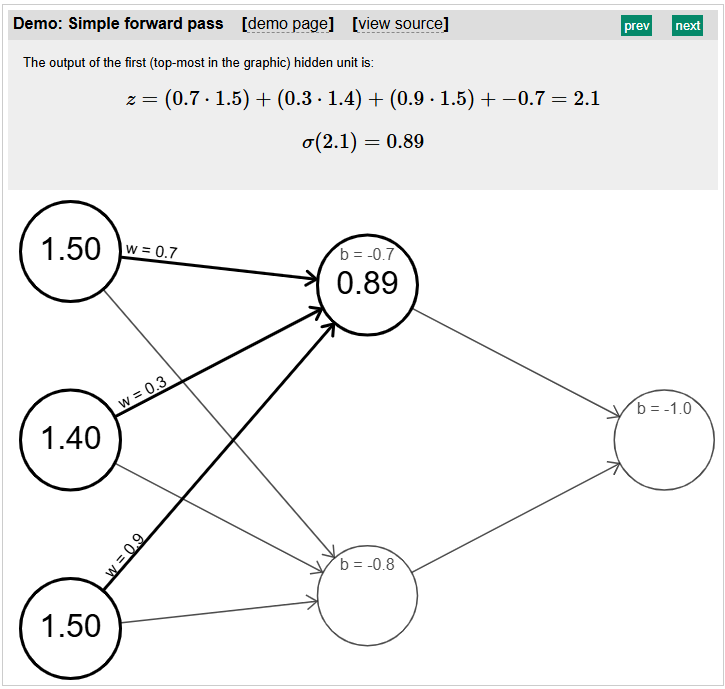
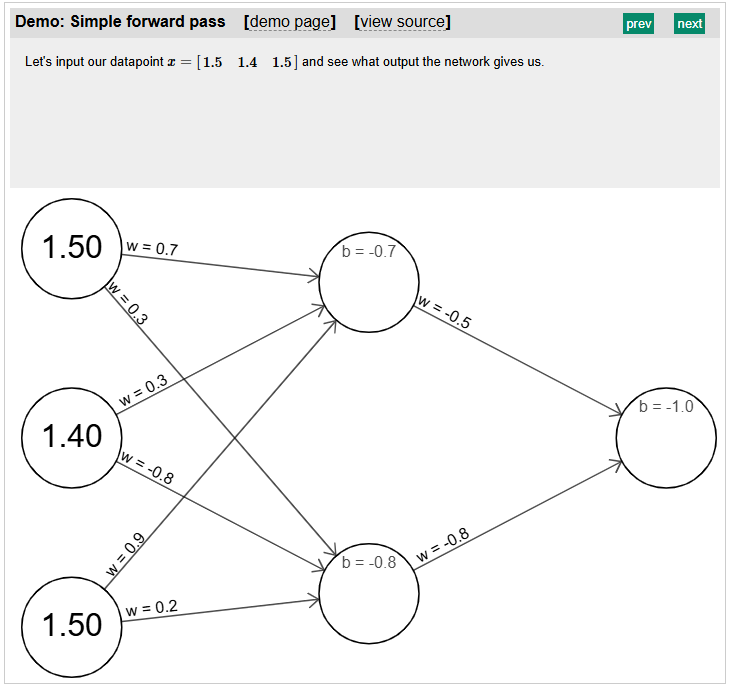
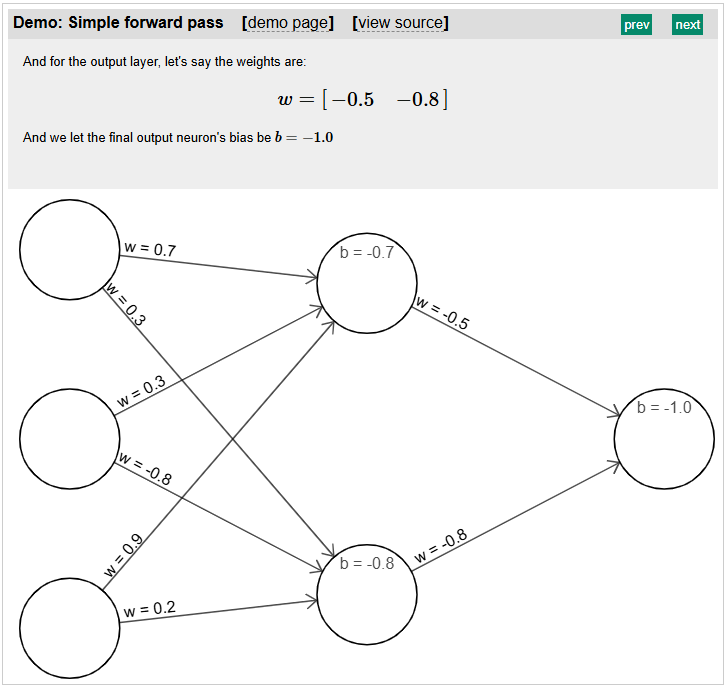
|  |  |  |  |
| --- | --- | --- | --- |
| Sigmoid | Tanh | ReLU | Leaky ReLU |
|  |  |  |  |
| Là hàm lớn tuổi nhất, và được dùng nhiều trong nhiều năm trước. Tuy nhiên gần đây không được dùng nhiều do gây ra hiện tượng vanishing gradient | Không được dùng nhiều | Gần đây được dùng nhiều, giải quyết được vấn đề vanishing gradient nhưng lại có nhược điểm không khả vị tại điểm 0 (khắc phục bằng cách nếu input bằng 0 thì gán cho nó 1 giá trị khác 0) | Là hàm biến thể của ReLU được sử dụng rộng rãi bây giờ |

**4. Forward pass/ Forward propagation**

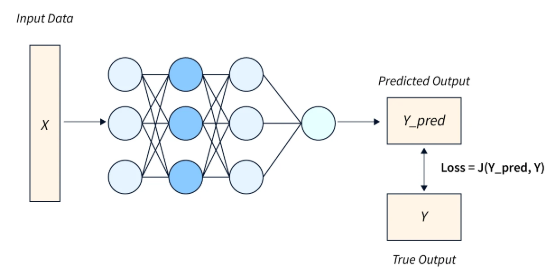
* Forward pass/ Forward propagation là quá trình đi từ input layer -> hidden layer -> output layer
* Mô phỏng quá trình Forward pass/ Forward propagation (Link: [tại đây](https://ml4a.github.io/demos/simple_forward_pass/))



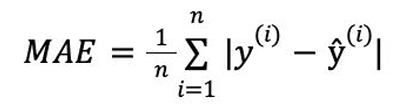




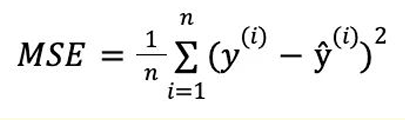
**5. Loss function**



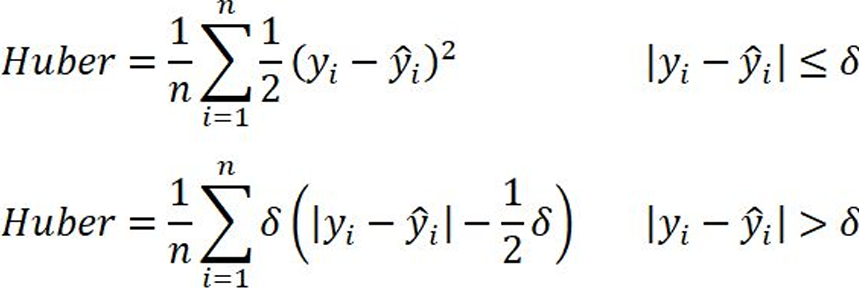
* Loss function cho ta biết sai lệch giữa giá tri thực tế và giá trị dự đoán => trong quá trình huấn luyện mô hình mục tiêu làm cho loss càng bé càng tốt
* Một vài loss function:
  + Mean Absolute Error Loss



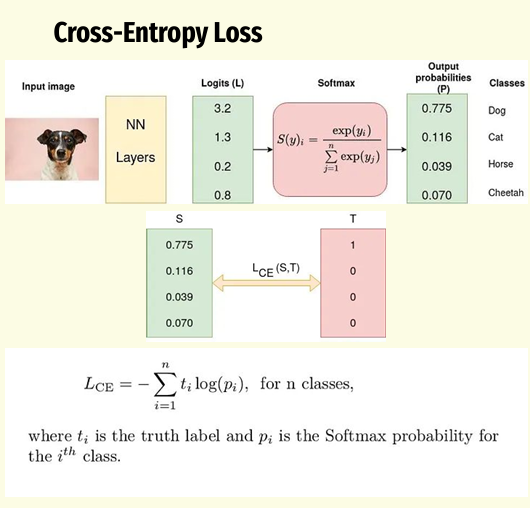
* + Mean Squared Error Loss



* + Huber Loss



* + Cross-Entropy Loss



* Output của mô hình Neeural Network sẽ cho đi qua hàm Softmax. Khi đi qua hàm Softmax thì output chỉ có giá trị từ 0 -> 1 và tổng của chúng bằng 1.

=> Do có tính chất như vậy chúng được xem như xác suất.

(Trong cá bài toán Classification chủ yếu đung hàm loss này)

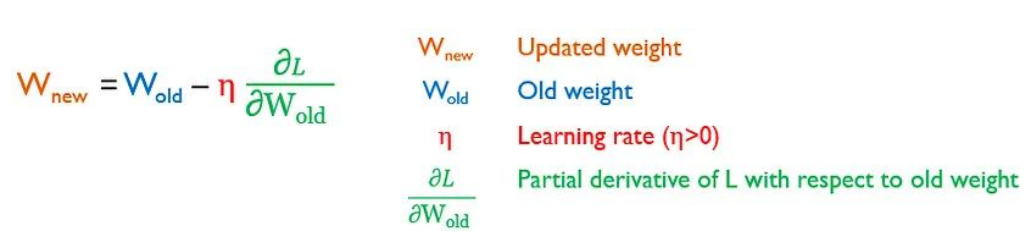
**6. Gradient (độ dốc)**



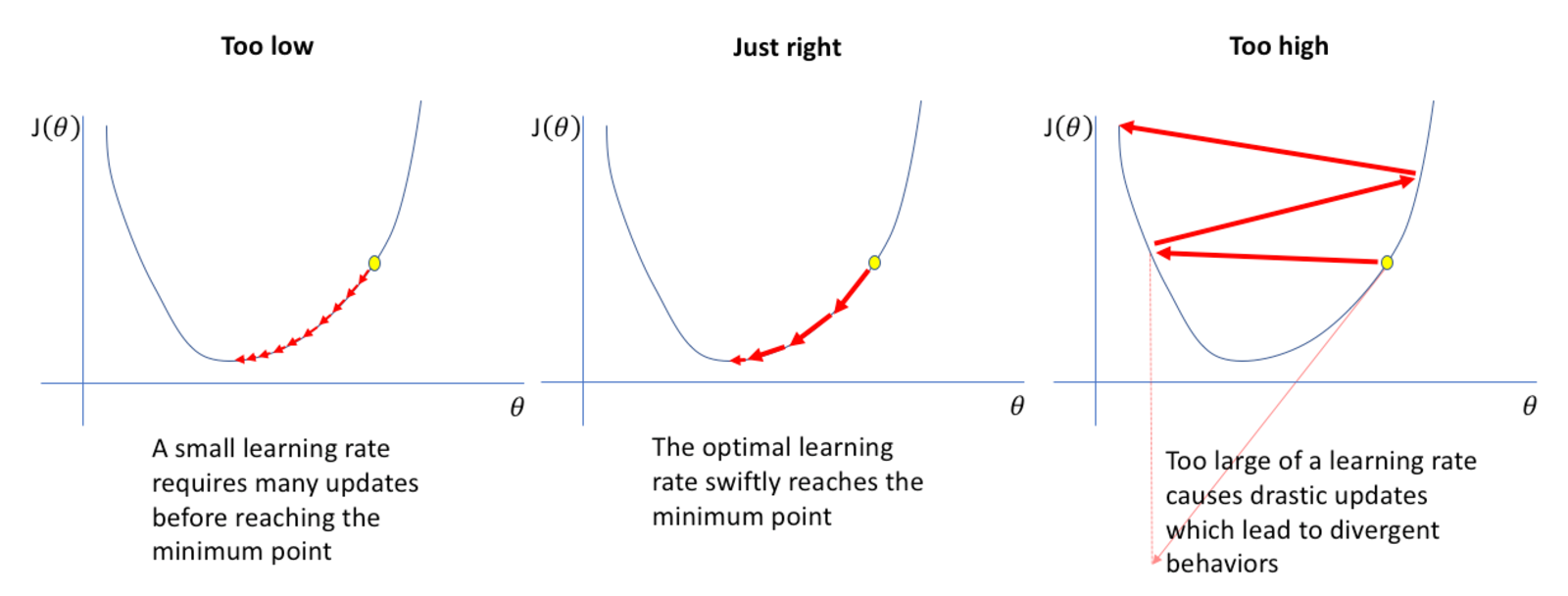
* Gradient là một vectơ trong khi đạo hàm là giá trị vô hướng (hiểu nôm na là các giá trị số). Véc tơ này chỉ ra hướng mà tại đó giá trị của hàm thay đổi nhiều nhất và trở thành véc tơ 0 khi hàm đạt giá trị cực đại hoặc cực tiểu địa phương.
* Một cách diễn đạt trực quan, tưởng tượng như ta đang ở trên một quả núi. Tại chân núi, các chỗ trũng của thung lũng và tại đỉnh núi, độ dốc bằng không. Tùy vào hình dạng của quả núi và vị trí ta quan sát, độ dốc sẽ có những giá trị khác nhau. Khi đang ở đỉnh núi thì đi về hướng nào cũng là “đường xuống núi” cả
* Mục tiêu của chúng ta là đưa Loss nhỏ nhất có thể vì vậy với ý tưởng dùng gradient descent ta có thể tìm được Loss phù hợp nhất, trong thực tế ta không thể tìm được Loss nhỏ nhất mà chỉ cố gắng tìm được các Loss gần với Loss nhỏ nhất.
* Một số Gradient Descent:

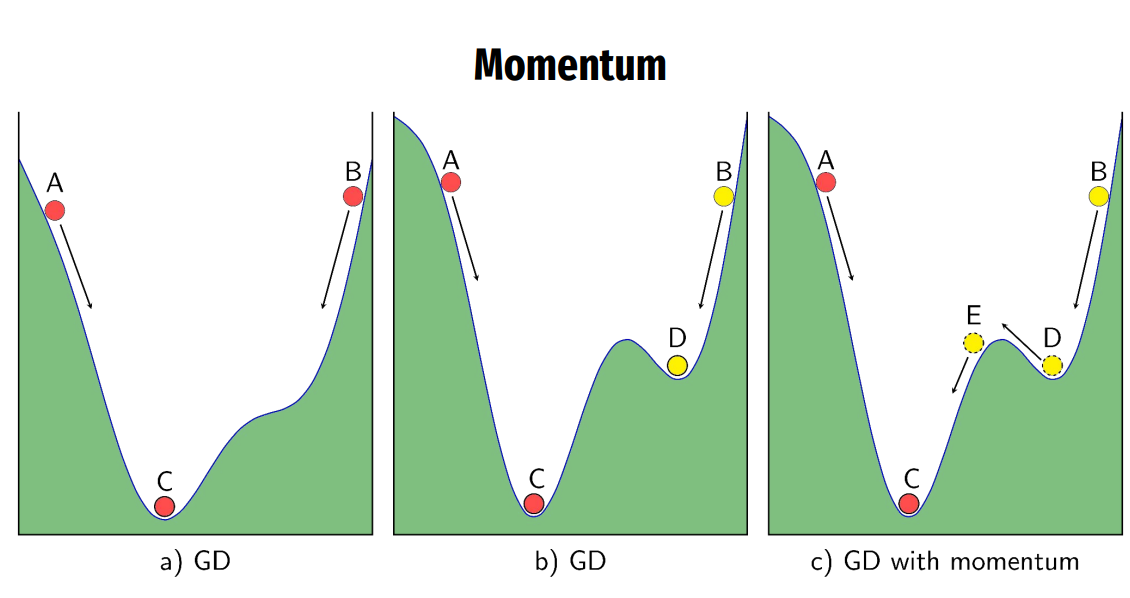
|  |  |  |
| --- | --- | --- |
| **Batch Gradient Descent** | **Stochastic Gradient Descent** | **Mini-batch Gradient Descent** |
| Tất cả datapoints được đưa vào mô hình cùng 1 lúc để tính gradient | Từng datapoint một sẽ được đưa vào mô hình để tính gradient | N datapoint sẽ được đưa vào mô hình cùng lúc để tính gradient |
| => Train tất cả thì đỏi hỏi nhiều tài nguyên tính toán, tính toán nhiều -> chậm (chỉ phù hợp với bộ dữ liệu ít) | => Train từng datapoint thì rất là chậm | => Cân bằng 2 phương pháp trước và là cách hay dùng nhất |

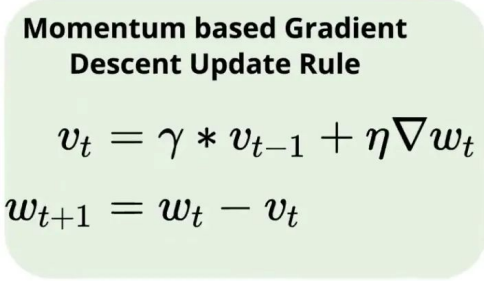
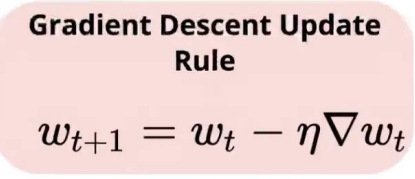
* Update parameters:



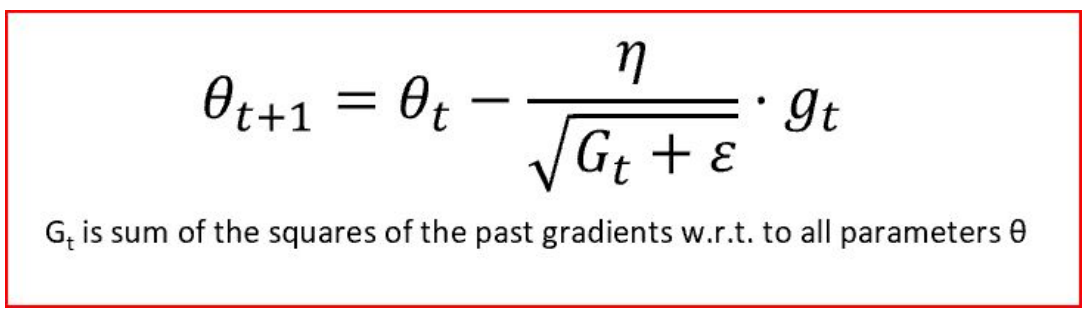
* Learning rate: giống như việc ta đi bước dài hay bước ngắn, nếu không có một Learning rate phù hợp cũng sẽ dẫn đến việc hội tụ sẽ chậm hoặc khó hội tụ:



* Để khắc phục việc bị dứng lại Loss không tốt hoặc khó hội tụ thì có sự kết hợp Momemtum với Gradient Descent => Giúp loại bỏ được các Loss không tốt, mặc dù vậy nhưng nó cũng có nhược điểm hội tụ lâu hơn 

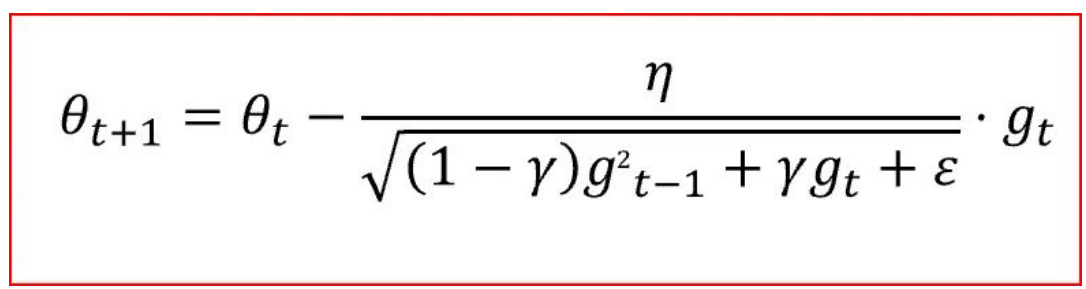


* Với thuật toán Gradient Descent bình thường thì laerning rate là một hằng số, để có thể tăng hiệu suất thì trong quá trình train thì nên cập nhập cả learning rate => Một số Optimizer:
  + Adaptive Gradient Descent:

Với mô hình này thì learning rate sẽ giảm dần khi đi gần xuống chân núi, nhưng không phải lúc nào phần gần đỉnh cũng sẽ dốc hơn, có 1 số trường hợp phần gần đỉnh nó sẽ thoải thì việc dùng Optimizer này có thể làm không hội tụ được. Ngoài ra việc learning rate khi ở gần chân núi càng ngày càng bé -> hội tụ chậm

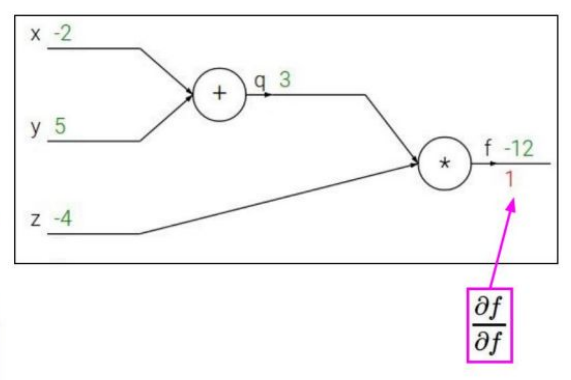
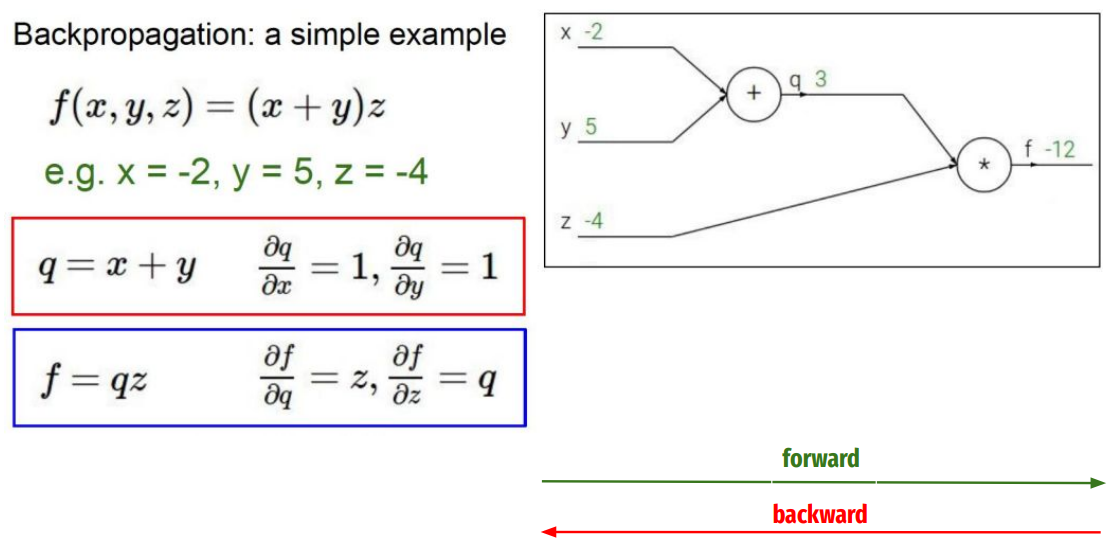
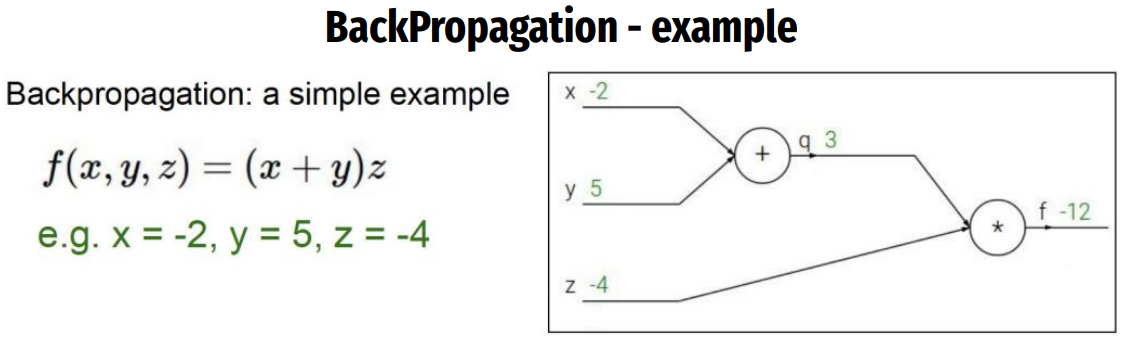
=> Để khắc phục nhược điểm Adaptive Gradient Descent thì cần có laerning rate vừa có tể răng, vừa giảm tùy trường hợp nên người ta đề xuất Optimizer mới Root Mean Square Propagation

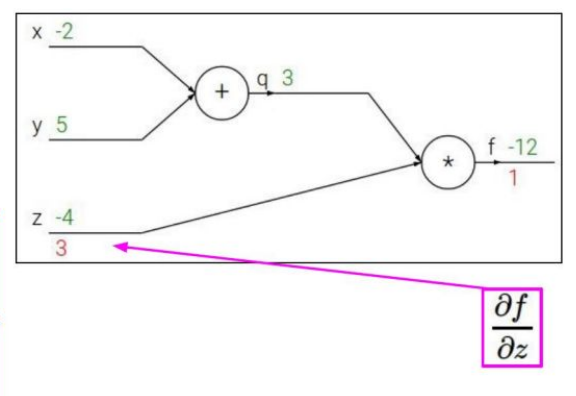
* + Root Mean Square Propagation:

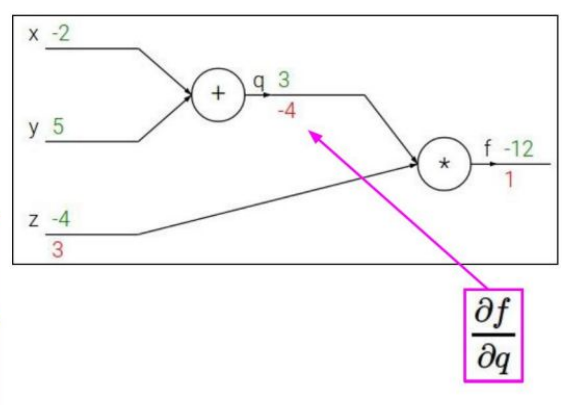


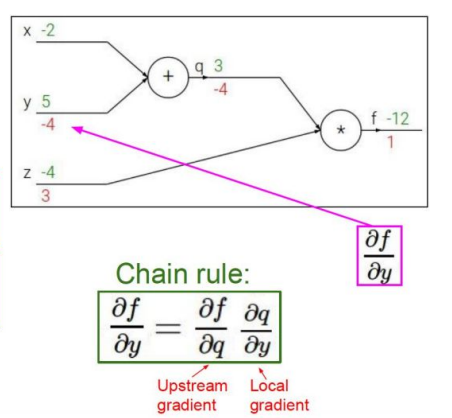
Với mô hình này dựa vào bước trước mà đang đi với vận tốc lớn thì đó là đoạn dốc -> laerning rate lớn; khi mà vận tốc bước trước nhỏ thì đó là đoạn thoải -> learning nhỏ

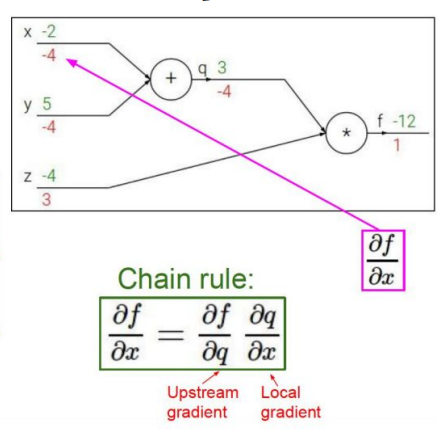
**7. BackPropagation**

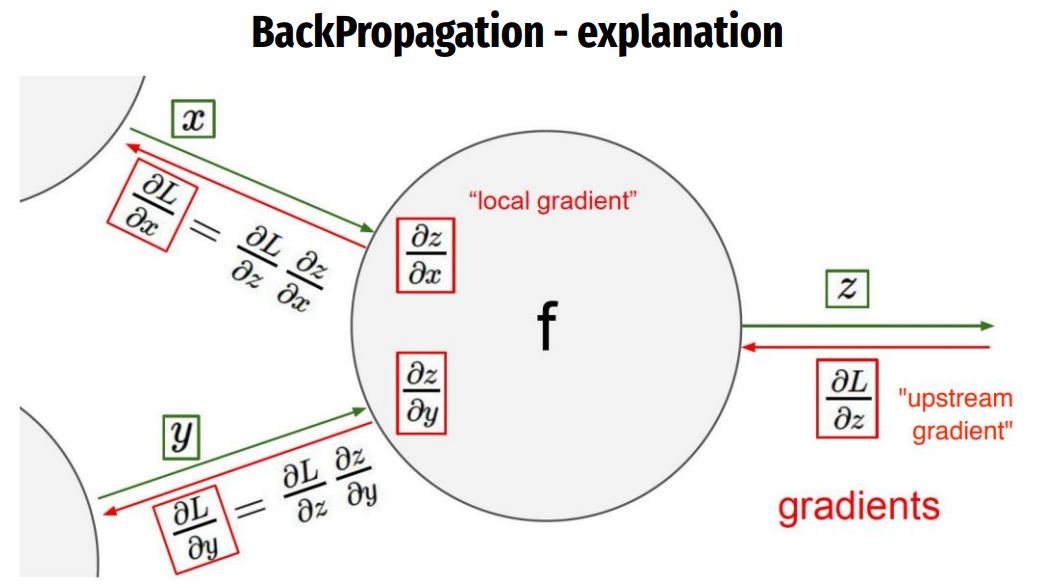












**8. Neural network’s learning process**

