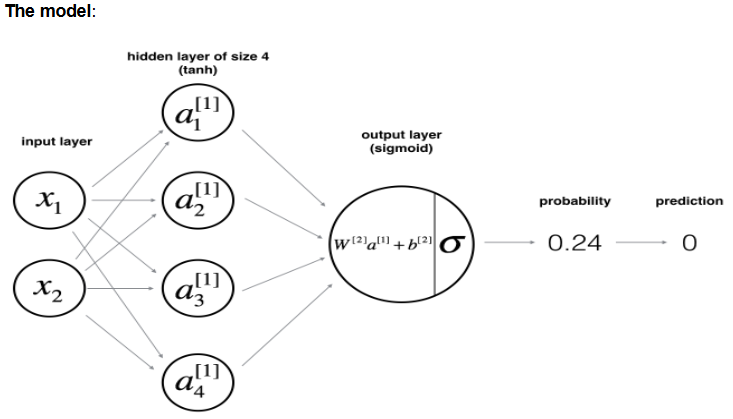
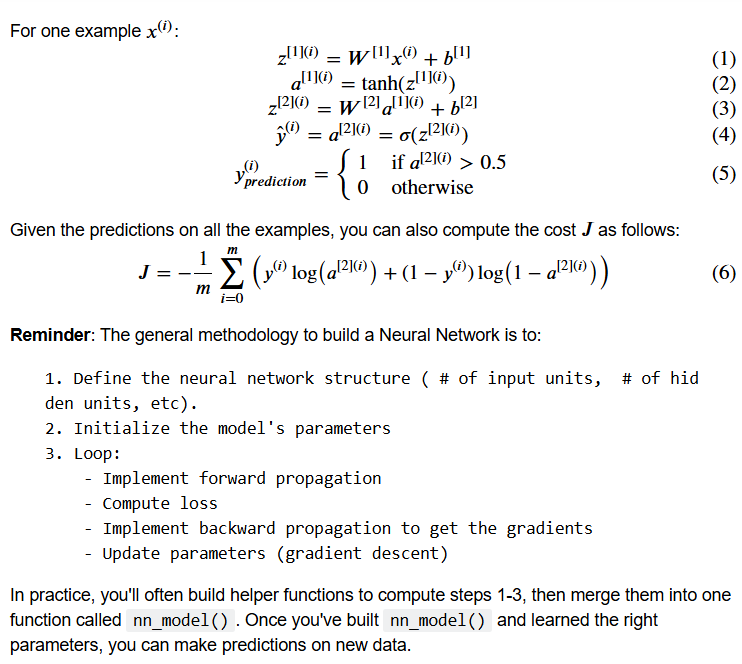
**REPORT EXERCISE COURSERA 1**

**Planar data classification with one hidden layer (Optional [assignment](https://www.coursera.org/learn/neural-networks-deep-learning/programming/e6FsA/planar-data-classification-with-one-hidden-layer))**

1. **Exercise and knowledge:**
2. **Using Simple Logistic Regression:**
   1. **If trust use model Simple Logistic Regression, the predict doesn’t perform well.**
3. **Using Neural Network model:**





* 1. **Layer\_sizes:** trong bài toán này thì khai báo để biết được kích thước của layer trong mạng neural.

N\_h: số lượng đặc trưng cho hidden layer

Nếu muốn tạo nhiều hidden layer: thì khi khai báo hàm ta sẽ thêm 1 giá trị là hidden layer và khai báo nó số lượng để tạo mạng neuron

* 1. **Initialize the model parameters:**

Khởi tạo W, b

Ta thực hiện random các giá trị để np.random.randn(n\_h, n\_x) \* 0.01

Việc \*0.01 giúp cho việc khới tạo giá trị không lớn, nó nhỏ nhưng không quá nhỏ để giúp quá trình học và train trở nên hiệu quả.

* Nếu khởi tạo w, b quá lớn thì dẫn đến “exploding gradients’ và khiến cho model trở nên không ổn định
* Nếu khởi tạo w, b quá nhỏ thì dẫn đến “vanishing gradients” và kiến cho model học chậm.

Ở đây ta thực hiện random cho các thông số cho w, b, tùy thuộc vào bài toán mà ta có thể khởi tạo

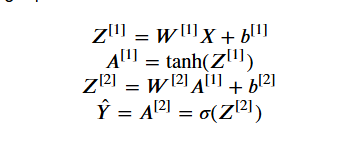
Khởi tạo w, b random:

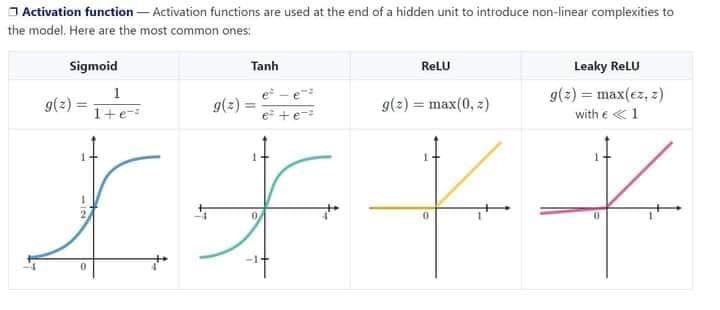
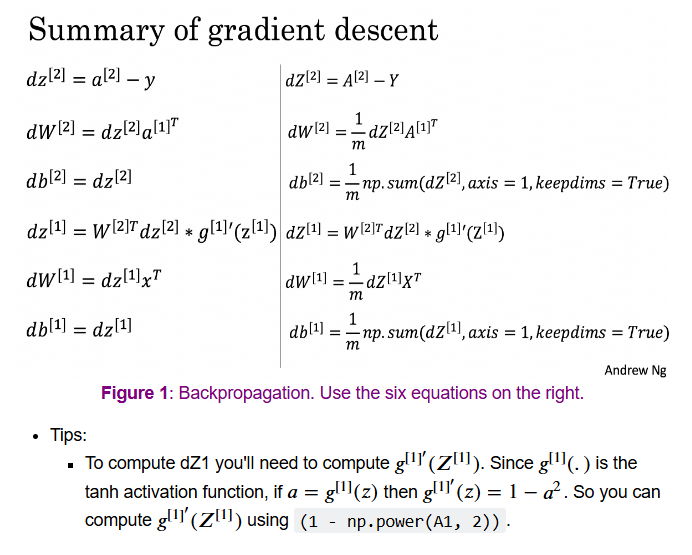
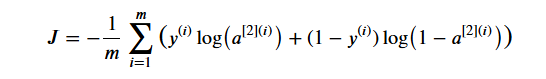
* Khi mạng neural có nhiều lớp và nhiều parameter thì việc khởi tạo các giá trị random để khi tính toán lan truyền ngược cập nhật trọng số của các neuron trong lớp sẽ khác nhau giúp model đạt hiệu quả tốt trong quá trình học.
* Khi khởi tạo w, b là 0 thì các neuron trong 1 lớp sẽ như nhau và khi lan truyền ngược thì cập nhật trọng số sẽ giống nhau sẽ khiến model mất khả năng đa dạng và làm giảm khả năng học của model

Khởi tạo w, b là 0:

* Thường được sử dụng trong mô hình hồi quy logistic or các mô hình đơn giản khác nơi chỉ có một đầu vào và một đầu ra

* 1. **Forward\_backward propagation:**
* **Forward propagation**



* **Backward propagation:**
* Np.sum(dz, axis = 1, keepdims = True): axis thực hiện tính tổng theo trục dọc của dz ( 0: tính toán theo cột( hàng); 1: tính toán theo hàng( cột)
* Keepdims: giữ lại số chiều của mảng ban đầu trong kết quả, giúp thuận tiện cho các phép toán.
  1. **Cost:**
* logprobs = np.multiply(np.log(A2), Y) + np.multiply(np.log(1 - A2), 1 - Y)
* cost = - np.sum(logprobs) / m

Or có thể tính theo cách này Cost = -1 / m \* np.sum(Y \* np.log(A) + (1 - Y) \* np.log(1 - A)) : Y: nhãn thực tế, A giá trị predict

* Nếu dùng np.multiply và np.sum thì kết quả sẽ là type Float
* Nếu dùng np.dot kết quả sẽ là numpy array 2D
* **np.squeeze():** it remove dimension have size = 1

Ex: 1-dimensional array: [1, 2, 3] ; have size (3,)

1. dimensional array: [[1, 2, 3], [4, 5, 6]]: have size(2, 3)
2. dimensional array: [[1, 2, 3, 4]]: size (1, 4)

3-dimensional array: [[1], [2], [3]]: size(1, 3, 1)

Or size (1, 3, 1, 4)

When use np.squeeze() --> size(1, 3, 1, 4) = (3, 4)

Size(1, 4) = (4, )

Size(1, 3, 1) = (3)

* Vd nếu kết quả [[17]] thì np.squezze([[17]]) ---> 17
* Và float() để đám bảo kết quả cost là một số chấm động để quá trình tính toán và thực hiện đo đạc dễ dàng so sánh hơn.
  1. **Update parameters**

Thực hiện update lại thông số W, b để kết quả trở nên tốt hơn:

* + 1. Num\_iterations: số lần lặp của thuật toán gradient descent
    2. Learning\_rate: tốc độ học thuật, điều khiển mức độ thay đổi của w, b trong mỗi lần cập nhật.
    3. Print\_cost: cờ boolean để bật tắt in giá trị hàm mất mát sau mỗi lần lặp.
    4. np.deepcopy(): tạo bản sao của w, b ban đầu
    5. w = w - learning\_rate \* dw

b = b - learning\_rate \* db

Cập nhật hệ số, theo quy ước di chuyển ngược lại so với w, b ban đầu và nhân với learning\_rate để giảm thiểu hàm chi phí sinh ra cho mô hình học.

* 1. **Model:**

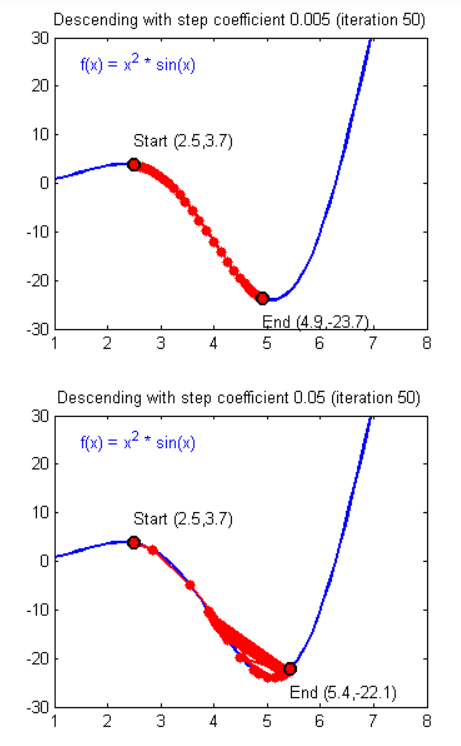
Thực hiện train model theo các thông số hàm đã tạo ở trên và tính toán cost sau 1000 interation.

Việc in ra sau 1000 lần sẽ giúp theo dõi được mô hình hoạt động ntn nếu in ra quá ít sẽ không nhận ra được sự thay đổi trong hàm chi phí và sẽ không biết được mô hình có vấn đề ở đâu. Nếu nhiều quá có thể khiến tăng tgian chạy và sẽ khó có thể theo dỗi được.

* 1. **Test the model and predicts**

Dùng kết quả A[2] được tính toán thông qua hàm forward\_propagation từ dữ liệu đầu vào “X”

Và so sánh kết quả đó với ngưỡng “0.5” vd kết quả dự đoán y\_predict > 0.5 thì sẽ là True còn < 0.5 sẽ là False

* **What happens when you change the tanh activation for a sigmoid activation or a ReLU activation?**
* Nếu chuyển từ hàm Tanh sang hàm sigmoid các giá trị trong neuron sẽ thay đổi, nó sẽ làm giảm giá trị đầu ra sau khi dùng hàm sigmoid về khoảng [0, 1] sẽ ảnh hưởng đến giá trị đầu vào “vanishing gradient” và nó có thể làm mất các giá trị âm điều này sẽ làm cho sự hội tụ chậm hơn trong quá trình huấn luyện
* Nếu chuyển từ hàm Tanh sang hàm ReLU, có thể giúp hàm tính toán và học nhanh do tính tuyến tính của hàm nó sẽ trả về giá trị đầu vào như vậy cho những giá trị dương và 0 cho các giá trị âm. Nó hiệu quả tính toán và tránh được vấn đề mất gradient”vanishing” cho các giá trị đầu vào dương. Nhưng nếu đầu vào là âm thì sẽ coi như mất và bị chết neuron.
* Hàm Tanh sẽ tính toán và giảm giá trị đầu vào trong phạm vi [-1, 1] được canh giữa ở 0 và đầu ra của Tanh xung quanh 0. nhưng cũng có nhược điểm nếu đầu vào là giá trị rất lớn hàm sẽ bị mất gradient or giá trị quá nhỏ.
* **Play with the learning\_rate. What happens?**
* 
* Learning Rate cao: mô hình có thể vượt quá giá trị nhỏ nhất của hàm cost và không thể hội tụ được. Điều đó dẫn đến phân kì or dao động xung quanh giá trị nhỏ nhất.
* Learning rate thấp: mô hình mất nhiều thời gian để hội tụ về giá trị nhỏ nhất của hàm. Và cần nhiều lần lặp(epoch) để hội tụ. Phù hợp trong trường hợp khi dữ liệu phức tạp và nhiều noise, khi model bị overfitting
* **What if we change the dataset? (See part 7 below!)**
* Thay đổi tập dữ liệu có thể ảnh hưởng lớn đến hiệu suất của mô hình. Nếu tập dữ liệu có nhiều mẫu khác nhau or phức tạp, mô hình cần thay đổi kiến trúc, learning rate or các hàm activation để phù hợp với dữ liệu. Nếu tập dữ liệu lớn có thể cần nhiều lần lặp để huấn luyện tốt hơn