LAB 02:
THỰC HÀNH VỚI PYTORCH

Ứng dụng Xử lý ảnh số và video số – 19TGMT

Giáo viên phụ trách:

- PhD. Lý Quốc Ngọc
- MS. Phạm Minh Hoàng
- MS. Nguyễn Mạnh Hùng

Sinh viên:

- Chung Kim Khánh (19127644)

MỤC LỤC

I. Giới thiệu về mạng nơ-ron

Mạng Nơ-ron là một mô hình lý luận dựa trên bộ não con người, bao gồm hàng tỷ tế bào thần kinh và hàng tỷ kết nối giữa chúng.

Bộ não sinh học được coi là một bộ não rất phức tạp, hệ thống xử lý thông tin phi tuyến và song song.

Học tập thông qua kinh nghiệm là một đặc điểm cần thiết.

Tính dẻo (Plasticity): kết nối giữa các tế bào thần kinh dẫn đến "Câu trả lời đúng" được củng cố trong khi những câu trả lời dẫn đến "Câu trả lời sai" bị yếu đi.

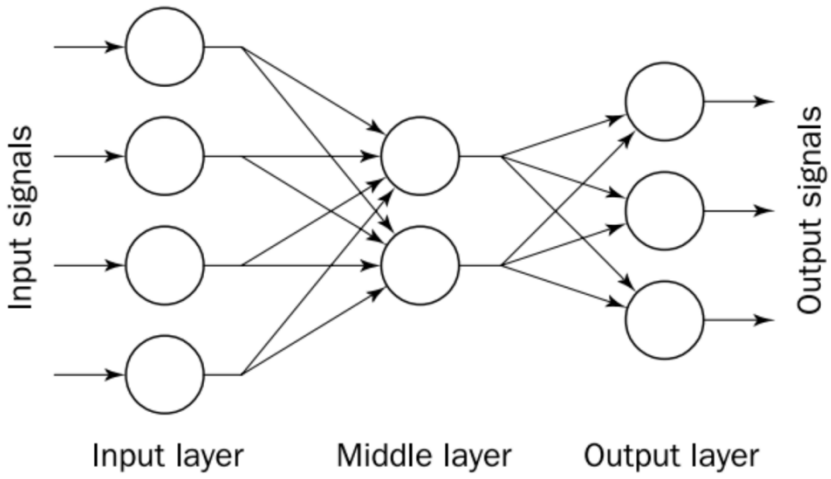
1. Mạng nơ-ron nhân tạo

Mạng nơ-ron nhân tạo (ANN) giống não người xét về cơ chế học tập. Tức là, cải thiện hiệu suất thông qua kinh nghiệm và tổng quát hóa.

Các ứng dụng: Nhận dạng ký tự viết tay, nhận dạng ở người (phát hiện giọng nói), chất nổ ở sân bay, …

2. Cách thiết lập mạng nơ-ron nhân tạo

ANN bao gồm một số tế bào thần kinh, là những bộ xử lý rất đơn giản và có tính liên kết cao sắp xếp theo thứ bậc của các lớp.



Mỗi nơron là một đơn vị xử lý thông tin cơ bản.

Các nơ-ron được kết nối với nhau bằng các liên kết truyền tín hiệu từ nơ-ron này sang nơ-ron khác. Mỗi nơ-ron nhận một số tín hiệu đầu vào thông qua các kết nối của nó và tạo ra nhiều nhất một tín hiệu đầu ra duy nhất.

Mỗi liên kết kết hợp với một trọng số thể hiện sức mạnh của đầu vào nơ-ron → phương tiện cơ bản của trí nhớ dài hạn trong ANN.

ANNs “học hỏi” thông qua các điều chỉnh lặp đi lặp lại của các trọng số này.

3. Cách xây dựng một mô hình ANN

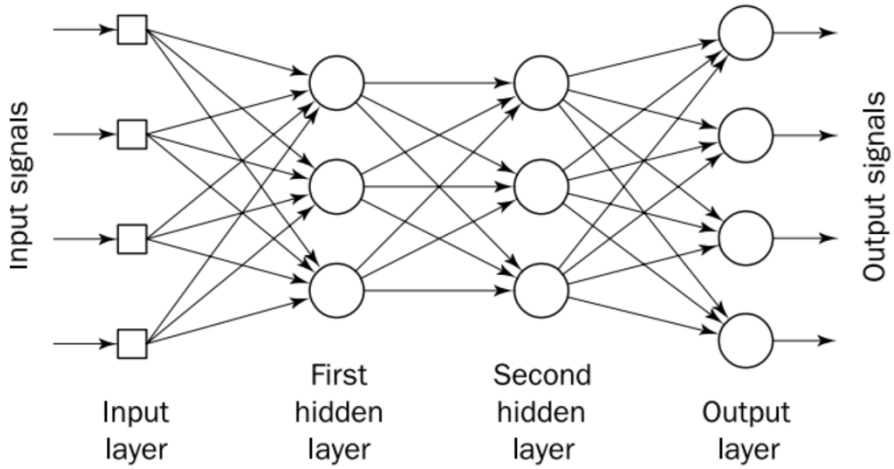
Kiến trúc mạng phải được quyết định trước tiên: Có bao nhiêu nơ-ron sẽ được sử dụng? Làm thế nào các tế bào thần kinh được kết nối để tạo thành một mạng lưới?

Sau đó, xác định thuật toán học tập nào sẽ sử dụng: Học có giám sát / bán giám sát / không giám sát / tăng cường

Cuối cùng huấn luyện mạng nơ-ron: Làm thế nào để khởi tạo các trọng số của mạng? Cách cập nhật chúng từ một tập hợp các ví dụ đào tạo.

4. Mạng nơron nhiều lớp

Một mạng chuyển tiếp với một hoặc nhiều lớp ẩn. Các tín hiệu đầu vào được truyền chuyển tiếp trên cơ sở từng lớp.



5. Thuật toán học truyền ngược (Back-propagation learning algorithm)

Được tìm ra bởi Bryson và Ho năm 1969, phổ biến nhất trong số hơn một trăm thuật toán học tập khác nhau có sẵn.

Bước 1: Khởi tạo

- Các trọng số và ngưỡng ban đầu được gán cho các số ngẫu nhiên.
- Số ngẫu nhiên có thể phân bố đồng đều trong một phạm vi nhỏ $\left(-\frac{2.4}{F_i}, +\frac{2.4}{F_i}\right)$ – Haykin, 1999 (trong đó F_i là tổng số đầu vào của nơron)
- Việc khởi tạo trọng số được thực hiện trên cơ sở từng nơron.

Bước 2: Kích hoạt

- Kích hoạt mạng bằng cách áp dụng các đầu vào $x_1(p), x_2(p), \dots, x_n(p)$ và các đầu ra mong muốn $y_{d,1}(p), y_{d,2}(p), \dots, y_{d,l}(p)$.

- Tính toán đầu ra thực tế của các nơ-ron trong lớp ẩn $y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p)w_{ij}(p) - \theta_j\right]$ trong đó n là số đầu vào của nơ-ron j trong lớp ẩn.
- Tính toán đầu ra thực tế của các nơ-ron trong lớp đầu ra $y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_j(p)w_{jk}(p) - \theta_j\right]$ trong đó m là số đầu ra của nơ-ron k trong lớp ẩn.
- $sigmoid(x) = \frac{1}{1+e^{-x}}$

Bước 3: Training trọng số

- Cập nhật các trọng số trong mạng lan truyền ngược và truyền ngược các lỗi liên quan đến các nơron đầu ra.
- Tính toán **gradient lỗi** cho các nơ-ron trong lớp đầu ra $\delta_k(p) = y_k(p) \times [1 - y_k(p)] \times e_k(p)$ trong đó $e_k(p) = y_{d,k}(p) - y_k(p)$
- Tính toán hiệu chỉnh trọng số $\Delta w_{jk}(p) = \alpha \times y_j(p) \times \delta_k(p)$
- Cập nhật trọng số tại các nơ-ron đầu ra $w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$
- Tính toán gradient lỗi cho các nơ-ron trong lớp ẩn $\delta_j(p) = y_j(p) \times \left[1 - y_j(p)\right] \times \sum_{k=1}^{l} \delta_k(p)w_{jk}(p)$
- Tính toán hiệu chỉnh trọng số $\Delta w_{ij}(p) = \alpha \times x_i(p) \times \delta_j(p)$
- Cập nhật trọng số tại các nơ-ron ẩn: $w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$

Bước 4: Sự lặp lại

- Tăng số lần lặp p một lần, quay lại Bước 2 và lặp lại quy trình cho đến khi tiêu chí lỗi đã chọn được thỏa mãn.

II. Thực nghiệm

1. Thay đổi tốc độ học

- **Giá trị đầu vào và đầu ra mẫu cho training:**

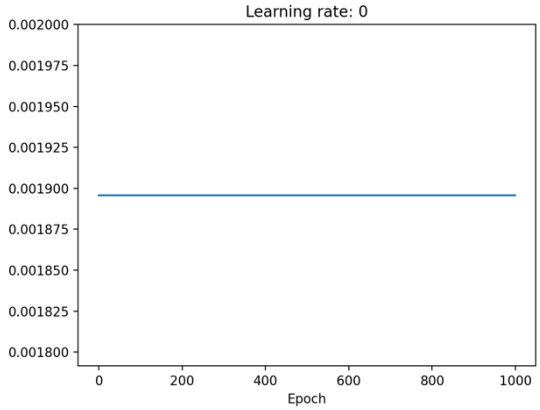
```
X = torch.tensor(([2, 9, 0], [1, 5, 1], [3, 6, 2]), dtype=torch.float)  # 3 X 3 tensor
Y = torch.tensor(([90], [100], [88]), dtype=torch.float)  # 3 X 1 tensor
```

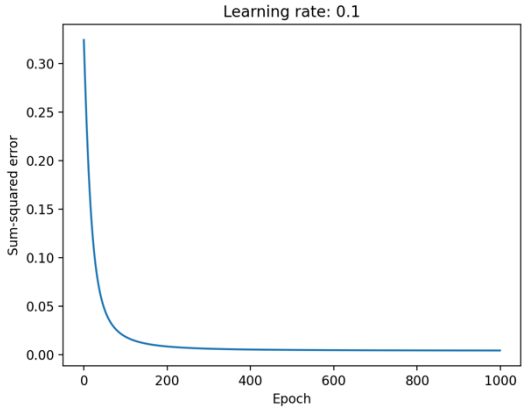
- **Đầu vào mẫu x để dự đoán**

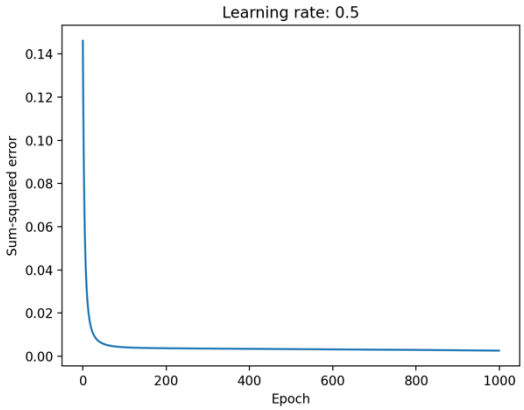
```
x_predict = torch.tensor(([3, 8, 4]), dtype=torch.float)  # 1 X 3 tensor
```

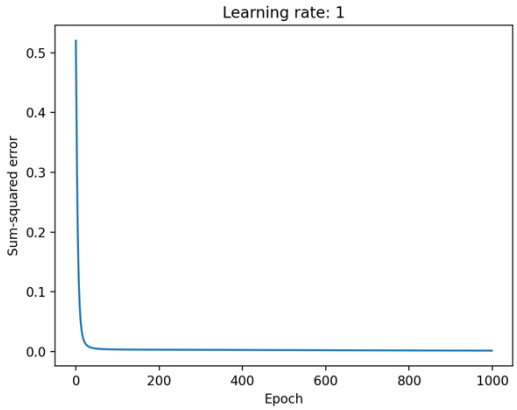
Bảng 1. Kết quả thay đổi Training Rate

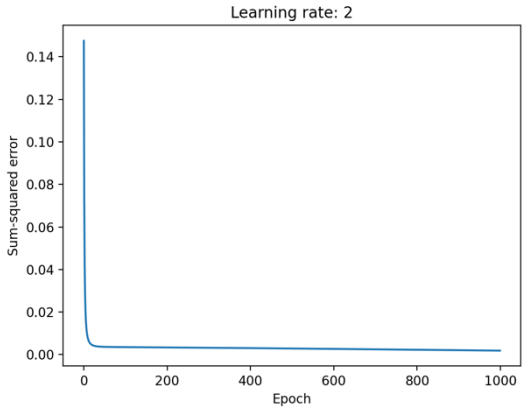
Tốc độ	Dự đoán dữ liệu dựa trên trọng số được đào tạo	
	Input	**Output**
0	tensor([0.3750, 1.0000, 0.5000])	tensor([0.3944])
0.1		tensor([0.9129])
0.5		tensor([0.9441])
1		tensor([0.9391])
2		tensor([0.9612])
7		tensor([0.9675])

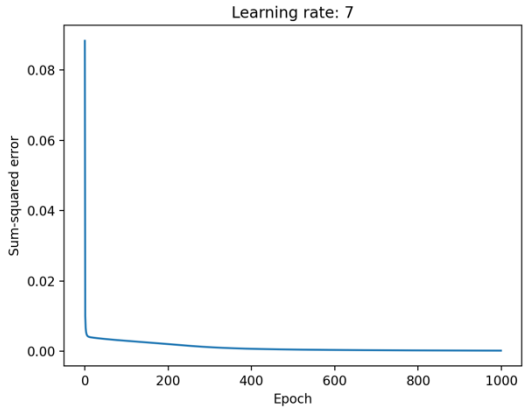












Nhận xét:

Tốc độ học tập thích ứng: Điều chỉnh tham số tốc độ học tập α trong quá trình đào tạo

- α nhỏ → trọng lượng thay đổi nhỏ qua các lần lặp lại → đường cong học tập trơn tru
- α lớn → tăng tốc quá trình luyện tập với khối lượng thay đổi lớn hơn → có thể mất ổn định và dao động.

Các phương pháp tiếp cận giống như Heuristic để điều chỉnh α

- Dấu đại số của sự thay đổi SSE vẫn duy trì trong một số hệ quả của Epoch → tăng α.
- Dấu đại số của sự thay đổi SSE xen kẽ trong một số hệ quả của Epoch → giảm α.

2. Thay đổi kích thước lớp ẩn

- **Giá trị đầu vào và đầu ra mẫu cho training:**

X = torch.tensor(([2, 9, 0], [1, 5, 1], [3, 6, 2]), dtype=torch.float) # 3 X 3 tensor

Y = torch.tensor(([90], [100], [88]), dtype=torch.float) # 3 X 1 tensor

- **Đầu vào mẫu x để dự đoán**

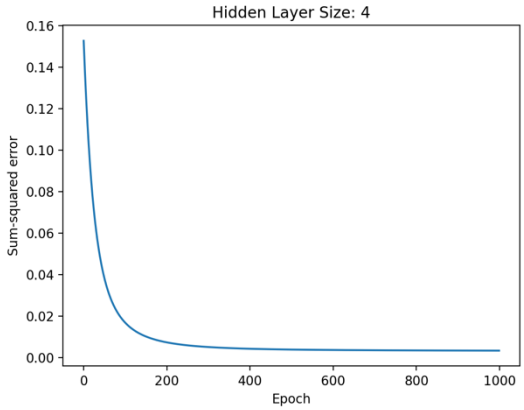
x_predict = torch.tensor(([3, 8, 4]), dtype=torch.float) # 1 X 3 tensor

- **Learning rate = 0.1**

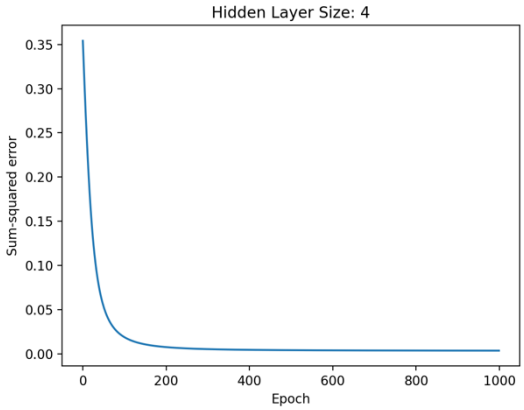
Không thể thay đổi InputSize và OutputSize vì nó phụ thuộc vào giá trị ban đầu. Vậy nên ta sẽ thay đổi kích thước của lớp ẩn.

Bảng 2. Kết quả thay đổi HiddenSize

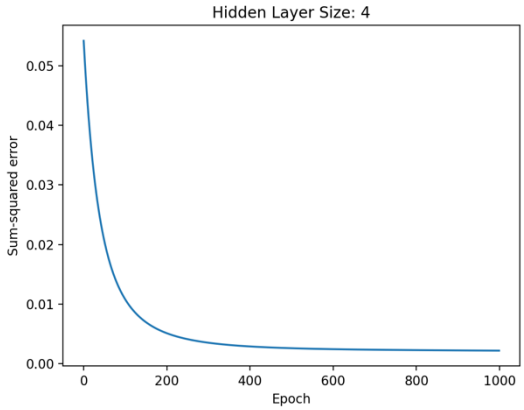
Size	Lần	Dự đoán dữ liệu dựa trên trọng số được đào tạo	
		Input	**Output**
4	1	tensor([0.3750, 1.0000, 0.5000])	tensor([0.9294])
	2		tensor([0.9170])
	3		tensor([0.9314])
	4		tensor([0.9314])
100	1		tensor([0.9909])
	2		tensor([0.9999])
	3		tensor([0.9708])
	4		tensor([0.0002])
	5		tensor([0.9423])
	6		tensor([0.9127])



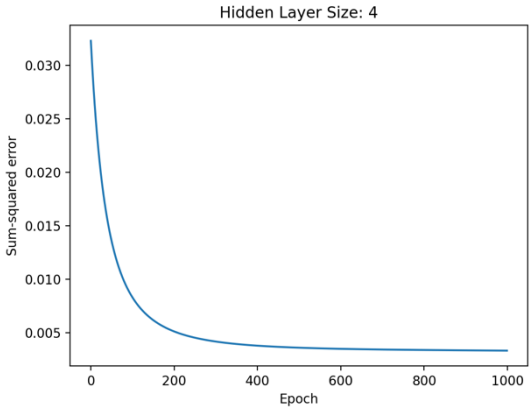
Hidden Layer Size 4_ 1



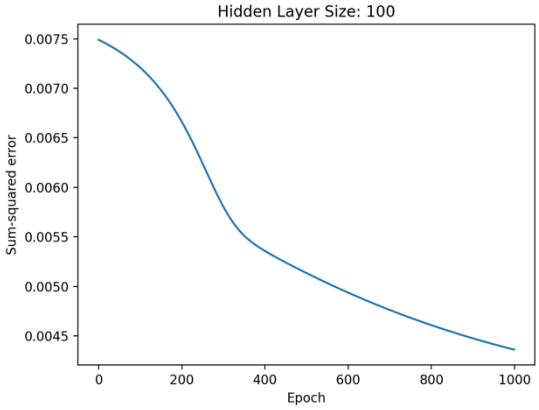
Hidden Layer Size 4_ 2



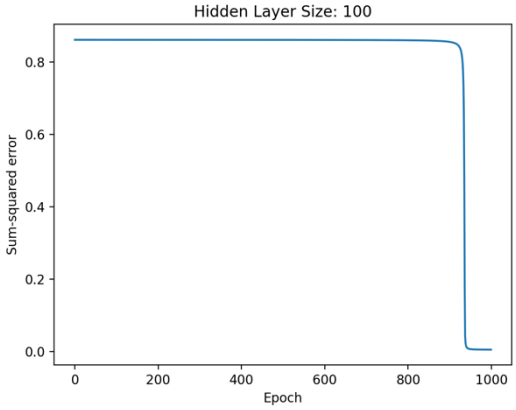
Hidden Layer Size 4_ 3



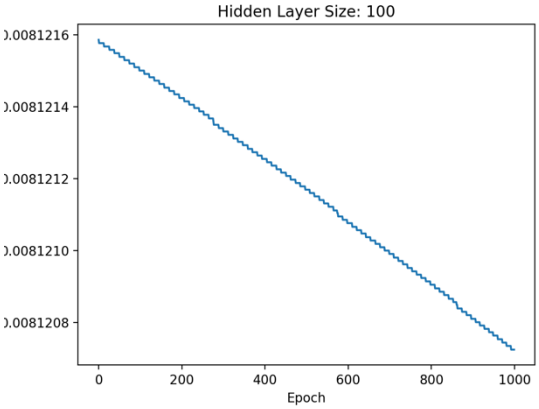
Hidden Layer Size 4_ 4



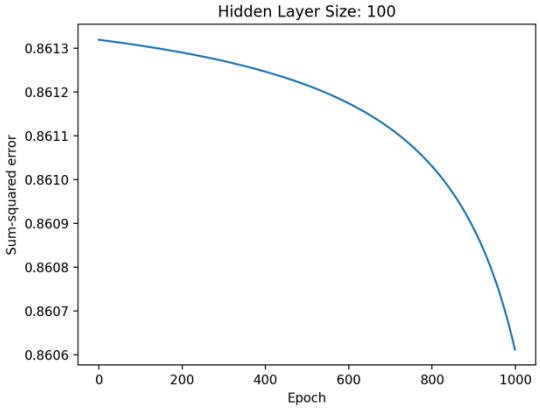
Hidden Layer Size 100_ 1



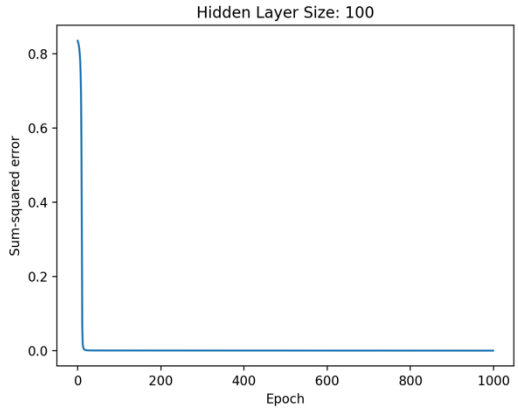
Hidden Layer Size 100_ 3



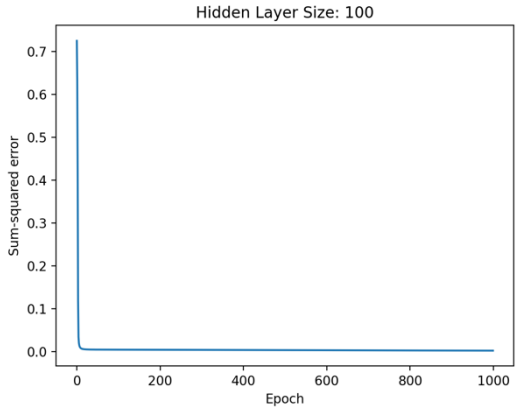
Hidden Layer Size 100_ 2



Hidden Layer Size 100_ 4



Hidden Layer Size 100_ 5



Hidden Layer Size 100_ 6

Nhận xét: Kích thước của Hidden Layer càng tăng thì ***sai số*** càng lớn.

3. Thay đổi số lớp ẩn

Nhận xét: Số lớp ẩn (Hidden Layer) càng nhiều thì ***sai số*** càng giảm. Tuy nhiên độ chính xác cao trong trường hợp thực tế lại phản tác dụng. Nó sẽ chỉ nhận diện được chính xác đúng đối tượng đã được học và không nhận diện thêm những trường hợp đối tượng đó ở những hoàn cảnh khác.

III. Nguồn tham khảo

[1] Neuron Networks, Cơ sở trí tuệ nhân tạo, ĐH KHTN