ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN
---------- 🕮🕮🕮 ----------

BÀI TẬP VỀ NHÀ 1
MÔN HỌC: THỊ GIÁC MÁY TÍNH

Họ Và Tên Sinh Viên: *Trần Thuận Phát*
MSSV: *21127666*

GIÁO VIÊN MÔN HỌC:
Thầy Phạm Minh Hoàng
Thầy Nguyễn Trọng Việt
Thầy Võ Hoài Việt

TPHCM – 03/2024

MỤC LỤC

NỘI DUNG

I. Thông tin cá nhân:

Họ và tên	**Trần Thuận Phát**
Mã số sinh viên	21127666

II. Bảng tự đánh giá:

STT	Tiêu chí	Đánh giá
1	Có xây dựng và mô tả các mô hình neural network với số layer ẩn và số node theo đúng yêu cầu và kèm hình ảnh minh họa.	**100%**
2	Có so sánh, đánh giá và trực quan hóa kết quả các mô hình.	**100%**
3	Có giải thích cách thức và đề xuất cách chọn số node và số layer ẩn cho mô hình shallow và deep neural network cho cả 2 tập dữ liệu.	**100%**
4	Có giải thích kết quả và nêu sự khác biệt.	**100%**
5	Cấu trúc thư mục bài nộp theo đúng yêu cầu.	**100%**
6	Báo cáo rõ ràng, đầy đủ, trực quan.	**100%**
7	Kết quả của thuật toán tốt.	**100%**
8	Thời gian chạy nhanh. (trong khoảng 60 giây)	**100%**
9	Dễ bảo trì và mã nguồn có chú thích.	**100%**
10	Có hướng dẫn cách chạy.	**100%**
11	Có kèm danh sách các chức năng (phân đoạn code) và giải thích.	**100%**
12	Nêu đủ tài liệu tham khảo.	**100%**

III. Các bước xây dựng một mô hình NN:

Bước 1: Import các thư viện hỗ trợ cho quá trình xây dựng mô hình, lấy bộ dữ liệu, gán nhãn và trực quan hóa mô hình và kết quả như là numpy, pandas, matplotlib, random, keras.

Bước 2: Load bộ dữ liệu theo nhu cầu và lưu vào các mảng X_train, y_train, X_test và y_test với sự hỗ trợ của thư viện keras và lời gọi hàm

load_data(). Như 2 câu lệnh đã dung trong bài tập dưới đây:

(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

hoặc

(X_train, y_train), (X_test, y_test) = mnist.load_data()

Bước 3: Load tập test từ file csv được chuẩn bị trước và nằm trong thư mục data.

Bước 4: Hiển thị một số hình ảnh từ tập train đã lưu (biến X_train) để đảm bảo rằng dữ liệu đã được lưu đúng và chuẩn bị cho quá trình huấn luyện.

Bước 5: Tiếp theo ta tiền xử lý dữ liệu đầu vào. Đầu tiên dữ liệu ảnh sẽ được trải phẳng từ ma trận 2D (kích thước 28x28 pixels) thành một vectơ 1D có kích thước 784 (28x28). Việc này là cần thiết để có thể đưa dữ liệu vào mạng neural network được thiết kế để nhận dữ liệu đầu vào 1D. Tiếp theo ta chuẩn hóa dữ liệu bằng cách chia X_train cho 255 và X_test chia cho 255. Việc chuẩn hóa dữ liệu bằng cách chia mỗi giá trị pixel cho 255 làm cho các giá trị pixel về khoảng từ 0 đến 1, giúp tăng tốc độ hội tụ của mô hình trong quá trình huấn luyện và giảm khả năng bị ảnh hưởng bởi tỷ lệ giá trị lớn nhỏ khác nhau giữa các thuộc tính.

Bước 6: Tiếp theo, ta thực hiện việc chuyển đổi nhãn y_train và y_test thành one-hot-encoding để phù hợp với mô hình mạng neural network. Vì các mạng neural network thường mong đợi đầu ra ở dạng phân phối xác suất (softmax function), với mỗi lớp đầu ra có một nút tương ứng. Và mỗi nhãn (class) được biểu diễn bằng một vectơ có kích thước bằng số lượng lớp, trong đó chỉ có một phần tử bằng 1 và các phần tử còn lại

bằng 0. Bên cạnh đó nó giúp loại bỏ ảnh hưởng của thứ tự do các nhãn không còn được sắp xếp theo thứ tự nhất định mà được biểu diễn dưới dạng một vectơ vị trí, giúp tránh bất kỳ ảnh hưởng nào của thứ tự trên việc huấn luyện mô hình. Cuối cùng nó giúp dễ dàng so sánh và đánh giá do mã hóa one-hot tạo ra một biểu diễn đồng nhất cho các nhãn, làm cho việc so sánh và đánh giá hiệu quả hơn.

Bước 7: Xây dựng và huấn luyện một mô hình neural network với số layer ẩn và số node theo nhu cầu.

- **Bước 7.1:** Đầu tiên ta sử dụng phương thức Sequential() trong thư viện keras. Đây là một phương thức được sử dụng để tạo ra một mô hình neural network tuần tự. Cụ thể hơn nó là một loại mô hình neural network được sử dụng phổ biến nhất và nó cho phép chúng ta tạo ra một chuỗi tuần tự các lớp neural network và số node theo mong muốn bằng phương thức add() trong đó dữ liệu đi từ lớp này sang lớp khác theo một chiều nhất định, từ đầu vào đến đầu ra và các lớp này sẽ được xếp chồng lên nhau theo thứ tự mà chúng được thêm vào.

```
model = Sequential()
```

- **Bước 7.2:** Add vào mô hình với lớp Dense (Fully Connected Layer) với số node và kích thước đầu vào theo mong muốn.

```
model.add(Dense([số lượng node], input_shape = ([kích thước đầu vào])))
```

- **Bước 7.3:** Add vào mô hình hàm kích hoạt theo ý muốn và lớp này không tạo ra nút mới, mà chỉ định rằng sau lớp

Dense trước đó.

```
model.add(Activation([tên hàm kích hoạt]))
```

- **Bước 7.4:** Add vào mô hình lớp Dropout và lớp này cũng không tạo ra nút mới, mà chỉ là một phần của lớp trước đó. Lớp Dropout giúp loại bỏ ngẫu nhiên và tránh tình trạng overfitting và là một phần của đầu ra của lớp trước đó trong quá trình huấn luyện

```
model.add(Dropout([tỷ lệ Dropout]))
```

- **Bước 7.5:** Sau 3 bước 7.2, 7.3, và 7.4 ta đã tạo được thành công 1 lớp ẩn với số node mong muốn. Nếu ta muốn tạo thêm các lớp ẩn tiếp theo cứ lặp lại 3 bước ở trên theo đúng tuần tự.
- **Bước 7.6:** Cấu hình mô hình trước khi huấn luyện, bao gồm việc chọn hàm mất mát (categorical_crossentropy), thuật toán tối ưu (adam) và các chỉ số đánh giá (accuracy) bằng phương thức compile().

```
model.compile(loss=[Độ lỗi], optimizer=[Thuật toán], metrics=[Metric])
```

- **Bước 7.7:** Ta sẽ huấn luyện mô hình bằng phương thức fit() và cung cấp các trọng số như là epochs (số vòng lặp huấn luyện), verbose (hiển thị thông tin huấn luyện), batch_size (kích thước batch dữ liệu), dữ liệu huấn luyện và dữ liệu test.
- **Bước 7.8:** Đánh giá độ chính xác của mô hình bằng phương thức evaluate với việc cung cấp dữ liệu test và verbose.

Bước 8: Hiển thị biểu đồ trực quan với trục x là số epoch và trục y là độ chính xác và độ mất mát với sự hỗ trợ của thư viện matplotlib.

Bước 9: Sau khi đã hoàn thành huấn luyện mô hình ta có thể đưa ra dự đoán bằng phương thức predict() với đầu vào tương ứng với đầu vào của việc huấn luyện.

IV. Các mô hình NN đã xây dựng:

1. Mô hình neural network với 1 lớp ẩn và 100 nodes:

```
model_1_layer_100_nodes = Sequential()
model_1_layer_100_nodes.add(Dense(100, input_shape=(784,)))
model_1_layer_100_nodes.add(Activation('relu'))
model_1_layer_100_nodes.add(Dropout(0.2))
model_1_layer_100_nodes.add(Dense(10))
model_1_layer_100_nodes.add(Activation('softmax'))
model_1_layer_100_nodes.summary()
model_1_layer_100_nodes.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history_1_layer_100_nodes = model_1_layer_100_nodes.fit(X_train, Y_train, batch_size=128, epochs=10, verbose=1, validation_data=(X_test, Y_test))
```

2. Mô hình neural network với 1 lớp ẩn và 300 nodes:

```
model_1_layer_300_nodes = Sequential()
model_1_layer_300_nodes.add(Dense(300, input_shape=(784,)))
model_1_layer_300_nodes.add(Activation('relu'))
model_1_layer_300_nodes.add(Dropout(0.2))
model_1_layer_300_nodes.add(Dense(10))
model_1_layer_300_nodes.add(Activation('softmax'))
model_1_layer_300_nodes.summary()
model_1_layer_300_nodes.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history_1_layer_300_nodes = model_1_layer_300_nodes.fit(X_train, Y_train, batch_size=128, epochs=10, verbose=1, validation_data=(X_test, Y_test))
```

3. Mô hình neural network với 1 lớp ẩn và 500 nodes:

```
model_1_layer_500_nodes = Sequential()
model_1_layer_500_nodes.add(Dense(500, input_shape=(784,)))
model_1_layer_500_nodes.add(Activation('relu'))
model_1_layer_500_nodes.add(Dropout(0.2))
model_1_layer_500_nodes.add(Dense(10))
model_1_layer_500_nodes.add(Activation('softmax'))
model_1_layer_500_nodes.summary()
model_1_layer_500_nodes.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history_1_layer_500_nodes = model_1_layer_500_nodes.fit(X_train, Y_train, batch_size=128, epochs=10, verbose=1, validation_data=(X_test, Y_test))
```

4. Mô hình neural network với 2 lớp ẩn và 100 nodes:

```
model_2_layer_100_nodes = Sequential()
model_2_layer_100_nodes.add(Dense(100, input_shape=(784,)))
model_2_layer_100_nodes.add(Activation('relu'))
model_2_layer_100_nodes.add(Dropout(0.2))
model_2_layer_100_nodes.add(Dense(100, input_shape=(784,)))
model_2_layer_100_nodes.add(Activation('relu'))
model_2_layer_100_nodes.add(Dropout(0.2))
model_2_layer_100_nodes.add(Dense(10))
model_2_layer_100_nodes.add(Activation('softmax'))
model_2_layer_100_nodes.summary()
model_2_layer_100_nodes.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history_2_layer_100_nodes = model_2_layer_100_nodes.fit(X_train, Y_train, batch_size=128, epochs=10, verbose=1, validation_data=(X_test, Y_test))
```

5. Mô hình neural network với 2 lớp ẩn và 300 nodes:

```
model_2_layer_300_nodes = Sequential()
model_2_layer_300_nodes.add(Dense(300, input_shape=(784,)))
model_2_layer_300_nodes.add(Activation('relu'))
model_2_layer_300_nodes.add(Dropout(0.2))
model_2_layer_300_nodes.add(Dense(300, input_shape=(784,)))
model_2_layer_300_nodes.add(Activation('relu'))
model_2_layer_300_nodes.add(Dropout(0.2))
model_2_layer_300_nodes.add(Dense(10))
model_2_layer_300_nodes.add(Activation('softmax'))
model_2_layer_300_nodes.summary()
model_2_layer_300_nodes.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history_2_layer_300_nodes = model_2_layer_300_nodes.fit(X_train, Y_train, batch_size=128, epochs=10, verbose=1, validation_data=(X_test, Y_test))
```

6. Mô hình neural network với 2 lớp ẩn và 500 nodes:

```
model_2_layer_500_nodes = Sequential()
model_2_layer_500_nodes.add(Dense(500, input_shape=(784,)))
model_2_layer_500_nodes.add(Activation('relu'))
model_2_layer_500_nodes.add(Dropout(0.2))
model_2_layer_500_nodes.add(Dense(500, input_shape=(784,)))
model_2_layer_500_nodes.add(Activation('relu'))
model_2_layer_500_nodes.add(Dropout(0.2))
model_2_layer_500_nodes.add(Dense(10))
model_2_layer_500_nodes.add(Activation('softmax'))
model_2_layer_500_nodes.summary()
model_2_layer_500_nodes.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history_2_layer_500_nodes = model_2_layer_500_nodes.fit(X_train, Y_train, batch_size=128, epochs=10, verbose=1, validation_data=(X_test, Y_test))
```

V. Mô tả các phân đoạn code:

1. Import the required libraries

Ở phân đoạn này sẽ thực hiện import các thư viện cần thiết hỗ trợ cho quá trình xây dựng mô hình, lấy bộ dữ liệu, gán nhãn và trực quan hóa mô hình và kết quả như là numpy, pandas, matplotlib, random, keras.

2. Load the Dataset:

Ở phần đoạn này sẽ thực hiện chức năng load bộ dữ liệu theo nhu cầu và lưu vào các mảng X_train, y_train, X_test và y_test với sự hỗ trợ của thư viện keras và lời gọi hàm load_data() và hiển thị một số hình ảnh từ tập train đã lưu (biến X_train) để đảm bảo rằng dữ liệu đã được lưu đúng và chuẩn bị cho quá trình huấn luyện.

3. Data Preprocessing:

Ở phân đoạn này sẽ tiền xử lý dữ liệu.

4. Convert to one-hot encoding:

Ở phân đoạn này ta thực hiện việc chuyển đổi nhãn y_train và y_test thành one-hot-encoding để phù hợp với mô hình mạng neural network.

5. One Layer And 100 Nodes

Ở phân đoạn này ta xây dựng mô hình NN với 1-layer ẩn và 100 nodes.

6. One Layer And 300 Nodes

Ở phân đoạn này ta xây dựng mô hình NN với 1-layer ẩn và 300 nodes.

7. One Layer And 500 Nodes

Ở phân đoạn này ta xây dựng mô hình NN với 1-layer ẩn và 500 nodes.

8. Two Layer And 100 Nodes

Ở phân đoạn này ta xây dựng mô hình NN với 2 layers ẩn và 100 nodes.

9. Two Layer And 300 Nodes

Ở phân đoạn này ta xây dựng mô hình NN với 2 layers ẩn và 300 nodes.

10. Two Layer And 500 Nodes

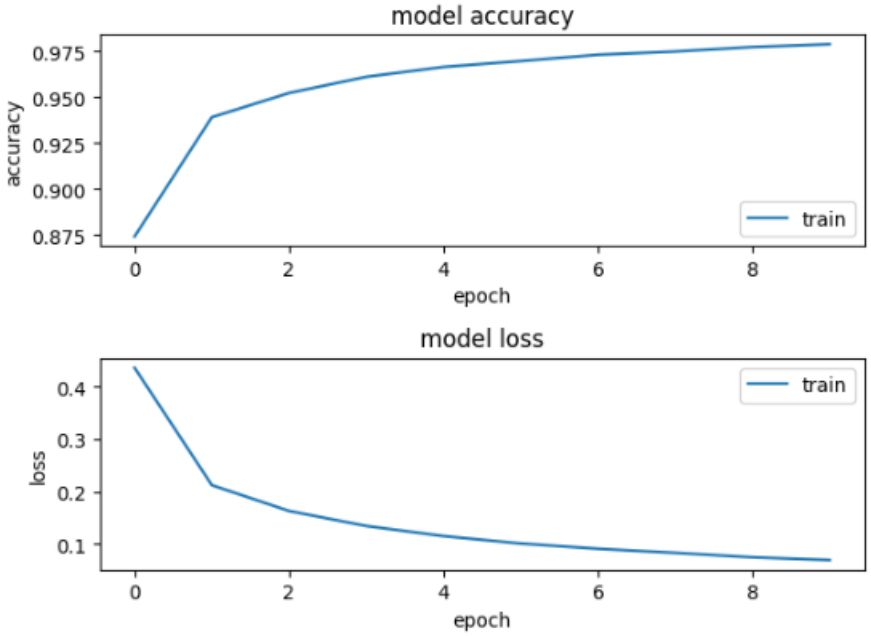
Ở phân đoạn này ta xây dựng mô hình NN với 2 layers ẩn và 500 nodes.

11. Evaluate the model:

Ở phân đoạn này giúp đánh giá độ chính xác của mô hình và trực quan độ mất mát và độ chính xác của tập train và test qua từng epoch. Cuối cùng là hiển thị một số hình ảnh dự đoán đúng và sai từ mô hình đã đào tạo.

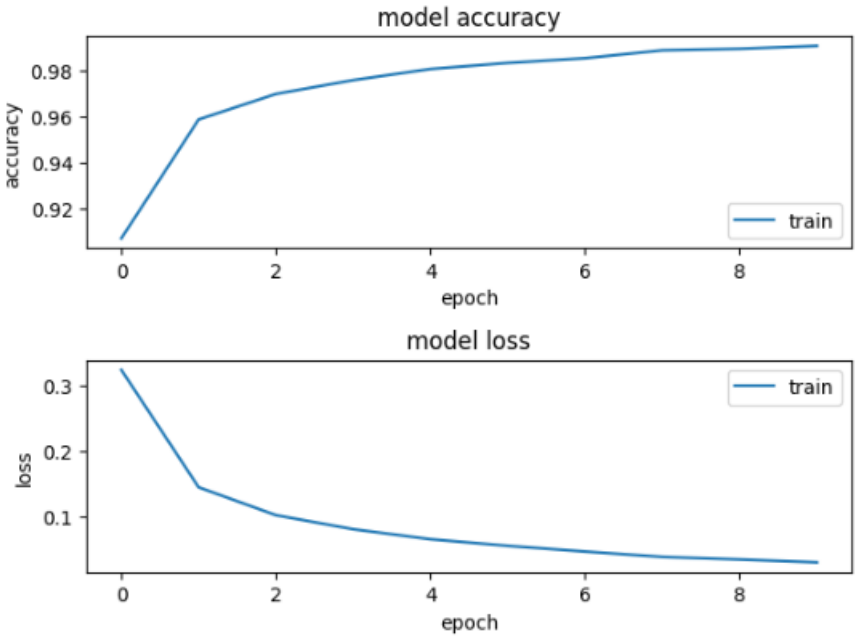
VI. Phân tích đánh giá kết quả với tập MNIST:

1. Mô hình neural network với 1 lớp ẩn và 100 nodes:



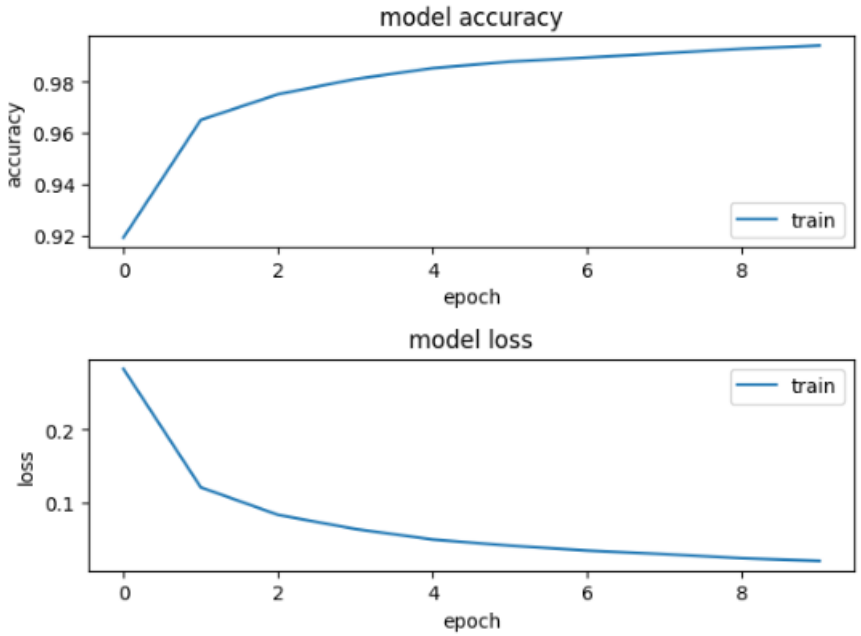
- **Độ chính xác:** 0.976
- **Đánh giá:** Đầu tiên xét về độ chính xác từ tập test, ta thấy ở đây khá cao không bị overfitting và rất tốt cho thấy mô hình được xây dựng đưa ra dự đoán tốt. Thứ hai, thời gian huấn luyện khá nhanh trong tầm 10 giây và tổng bộ nhớ mô hình chỉ chiếm 310.59KB với 79510 tham số. Cuối cùng là đường đi của 2 đường acc và loss trong mỗi epoch ở đồ thị trên rất ổn định và tăng hoặc giảm rất đều đặn cho thấy mô hình phù hợp dữ liệu đang huấn luyện.

2. Mô hình neural network với 1 lớp ẩn và 300 nodes:



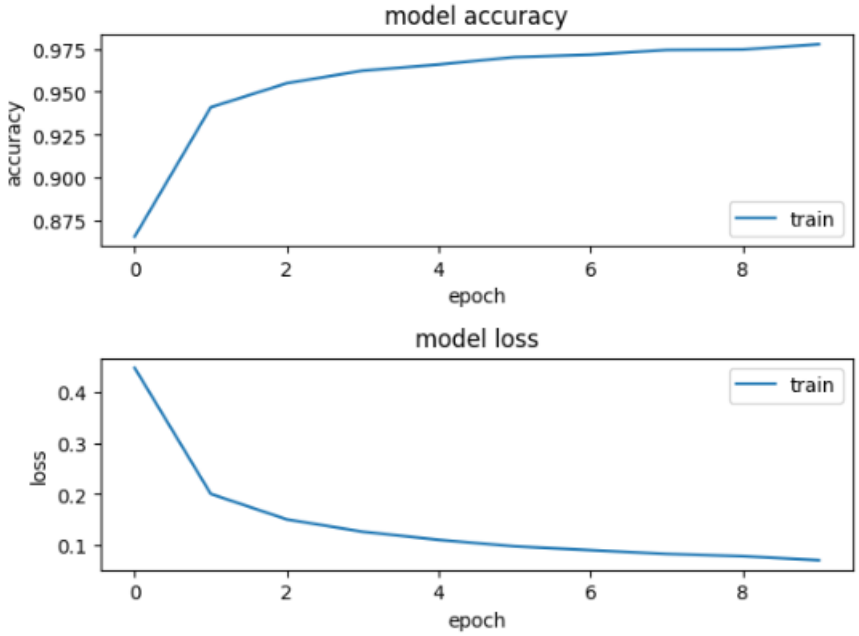
- **Độ chính xác:** 0.982
- **Đánh giá:** Tương tư như đánh giá ở trên và đầu tiên xét về độ chính xác từ tập test, ta thấy ở đây khá cao và cao hơn cả mô hình trên cho thấy mô hình không bị overfitting và đưa ra dự đoán tốt. Thứ hai, thời gian huấn luyện khá nhanh trong tầm 31.7 giây và tổng bộ nhớ mô hình chỉ chiếm 931.68KB với 238510 tham số. Cuối cùng là đường đi của 2 đường acc và loss trong mỗi epoch ở đồ thị trên rất ổn định và tăng hoặc giảm rất đều đặn cho thấy mô hình phù hợp dữ liệu đang huấn luyện. Tóm lại, việc tăng số node ở đây có ý nghĩa.

3. Mô hình neural network với 1 lớp ẩn và 500 nodes:



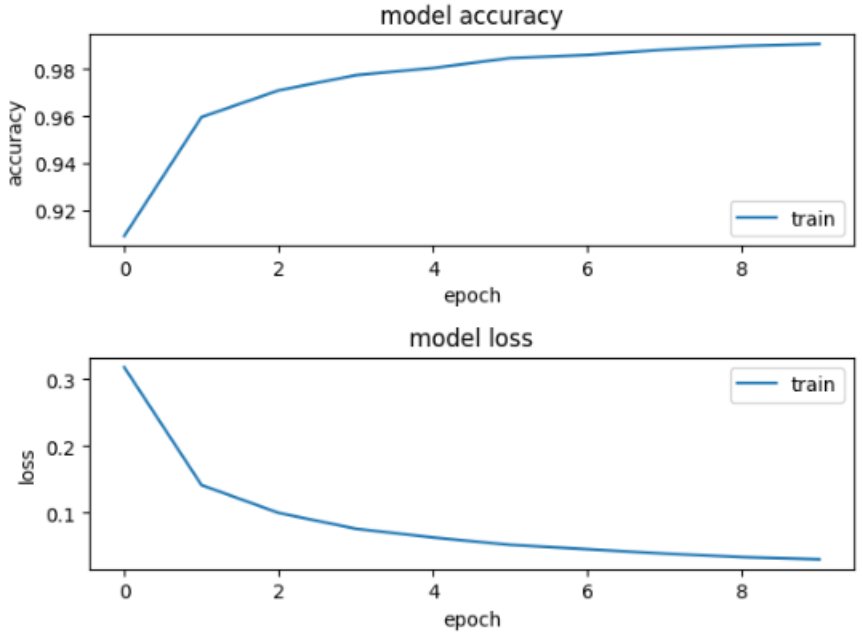
- **Độ chính xác:** 0.980
- **Đánh giá:** Tương tư như đánh giá ở trên và đầu tiên xét về độ chính xác từ tập test, ta thấy ở đây khá cao và nhưng không cao hơn mô hình trên nhưng vẫn cho thấy mô hình không bị overfitting và đưa ra dự đoán tốt. Thứ hai, thời gian huấn luyện cũng khá nhanh trong tầm 38.4 giây và tổng bộ nhớ mô hình chỉ chiếm 1.52MB với 397510 tham số. Cuối cùng là đường đi của 2 đường acc và loss trong mỗi epoch ở đồ thị trên rất ổn định và tăng hoặc giảm rất đều đặn cho thấy mô hình phù hợp dữ liệu đang huấn luyện. Tóm lại, việc tăng node ở đây không thực sự có ý nghĩa vì không tăng độ chính xác mà làm tăng bộ nhớ lưu trữ và thời gian xử lý.

4. Mô hình neural network với 2 lớp ẩn và 100 nodes:



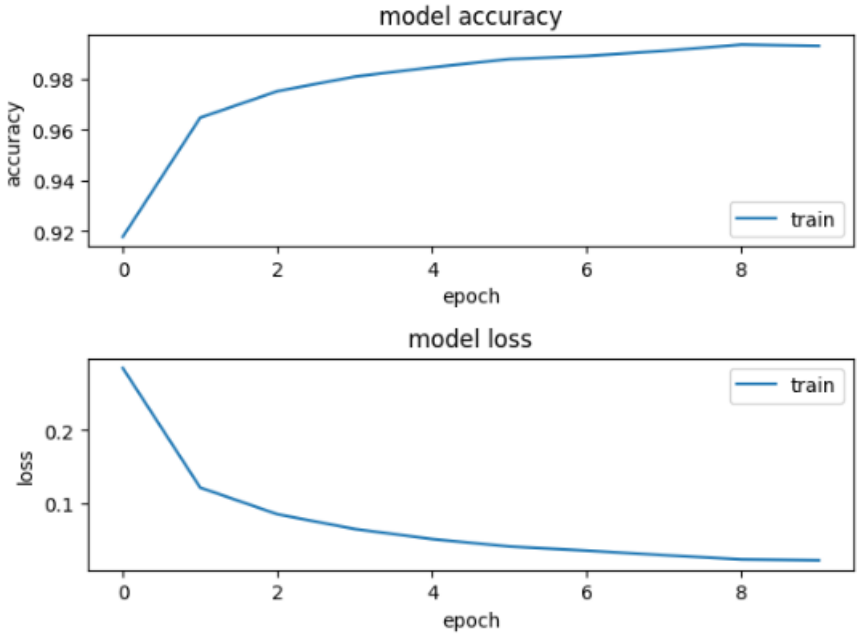
- **Độ chính xác:** 0.977
- **Đánh giá:** Đầu tiên xét về độ chính xác từ tập test, ta thấy ở đây khá cao không bị overfitting và rất tốt cho thấy mô hình được xây dựng đưa ra dự đoán tốt và độ chính xác cao hơn cả mô hình 1 layer với 100 nodes. Thứ hai, thời gian huấn luyện khá nhanh trong tầm 26.5 giây và tổng bộ nhớ mô hình chỉ chiếm 350.04KB với 89610 tham số. Cuối cùng là đường đi của 2 đường acc và loss trong mỗi epoch ở đồ thị trên rất ổn định và tăng hoặc giảm rất đều đặn cho thấy mô hình phù hợp dữ liệu đang huấn luyện. Tóm lại, việc tăng layer ở đây có ý nghĩa.

5. Mô hình neural network với 2 lớp ẩn và 300 nodes:



- **Độ chính xác:** 0.981
- **Đánh giá:** Đầu tiên xét về độ chính xác từ tập test, ta thấy ở đây khá cao không bị overfitting và rất tốt cho thấy mô hình được xây dựng đưa ra dự đoán tốt và độ chính xác cao hơn cả mô hình trên nhưng không cao hơn mô hình 1 layer với 300 nodes. Thứ hai, thời gian huấn luyện khá nhanh trong tầm 43 giây và tổng bộ nhớ mô hình chỉ chiếm 1.25MB với 328810 tham số. Cuối cùng là đường đi của 2 đường acc và loss trong mỗi epoch ở đồ thị trên rất ổn định và tăng hoặc giảm rất đều đặn cho thấy mô hình phù hợp dữ liệu đang huấn luyện. Tóm lại, việc tăng node có ý nghĩa nhưng tăng layer thì không vì làm tăng bộ nhớ và thời gian chạy nhưng độ chính xác không tăng.

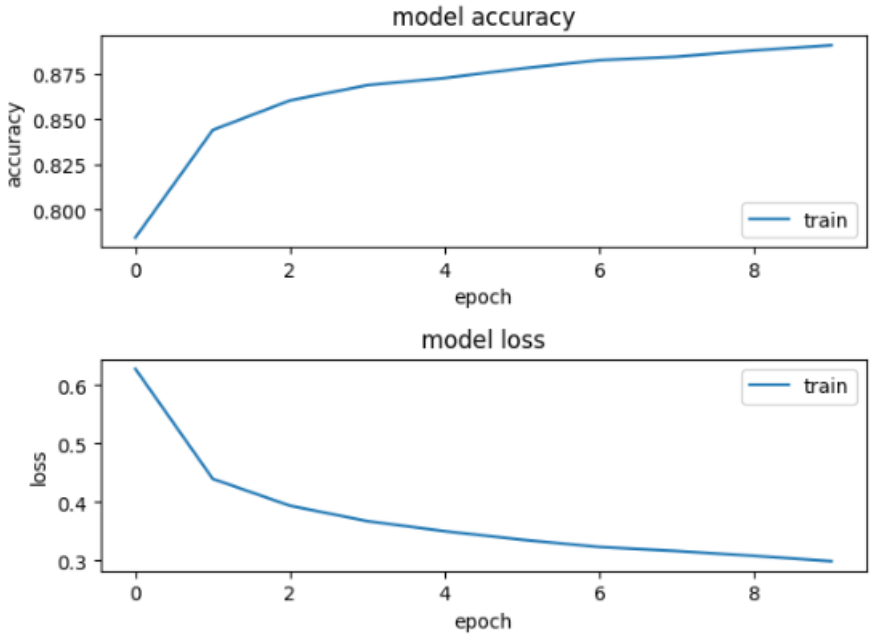
6. Mô hình neural network với 2 lớp ẩn và 500 nodes:



- **Độ chính xác:** 0.979
- **Đánh giá:** Đầu tiên xét về độ chính xác từ tập test, ta thấy ở đây khá cao không bị overfitting và rất tốt cho thấy mô hình được xây dựng đưa ra dự đoán tốt nhưng độ chính xác thấp hơn cả mô hình trên và cả mô hình 1 layer với 500 nodes. Thứ hai, thời gian huấn luyện chậm nhất trong tầm 83 giây và tổng bộ nhớ mô hình chỉ chiếm 2.47MB với 648010 tham số. Cuối cùng là đường đi của 2 đường acc và loss trong mỗi epoch ở đồ thị trên rất ổn định và tăng hoặc giảm rất đều đặn cho thấy mô hình phù hợp dữ liệu đang huấn luyện. Tóm lại, việc tăng layer và tăng node ở đây chưa thực sự có ý nghĩa vì làm tăng bộ nhớ và thời gian chạy nhưng độ chính xác không tăng.

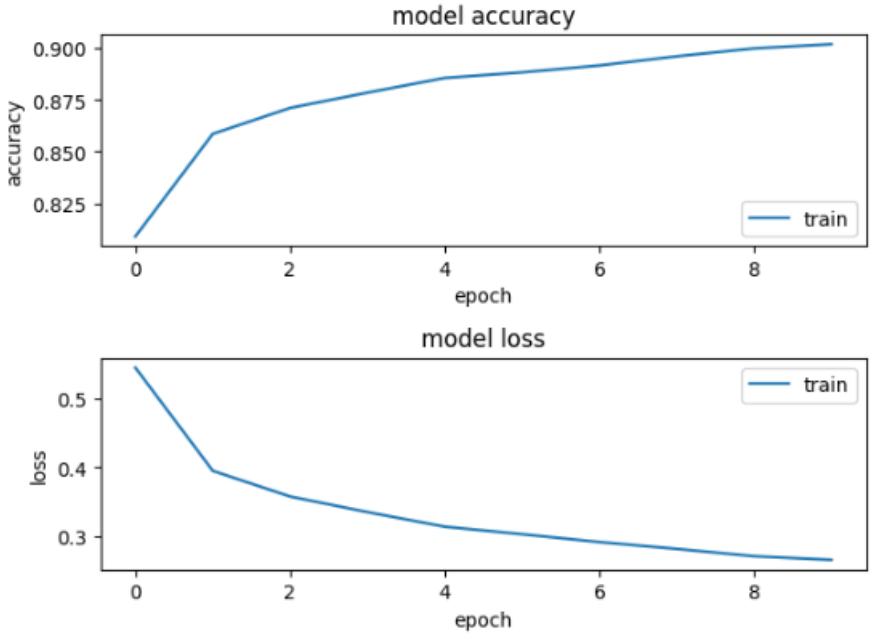
VII. Phân tích đánh giá kết quả với tập MNIST-FASHION:

1. Mô hình neural network với 1 lớp ẩn và 100 nodes:



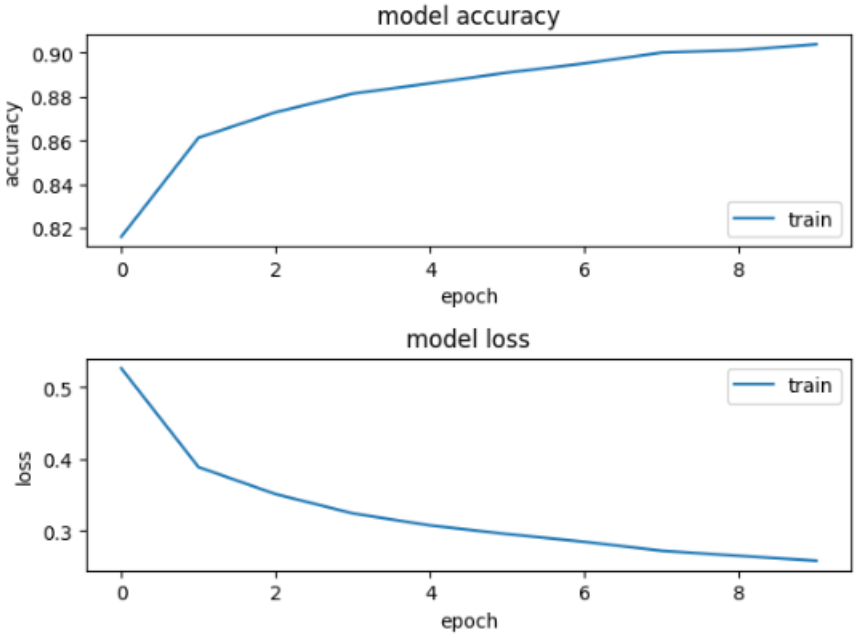
- **Độ chính xác:** 0.879
- **Đánh giá:** Đầu tiên xét về độ chính xác từ tập test, ta thấy ở đây khá cao không bị overfitting và rất tốt cho thấy mô hình được xây dựng đưa ra dự đoán tốt. Thứ hai, thời gian huấn luyện khá nhanh trong tầm 10 giây và tổng bộ nhớ mô hình chỉ chiếm 310.59KB với 79510 tham số. Cuối cùng là đường đi của 2 đường acc và loss trong mỗi epoch ở đồ thị trên rất ổn định và tăng hoặc giảm rất đều đặn cho thấy mô hình phù hợp dữ liệu đang huấn luyện.

2. Mô hình neural network với 1 lớp ẩn và 300 nodes:



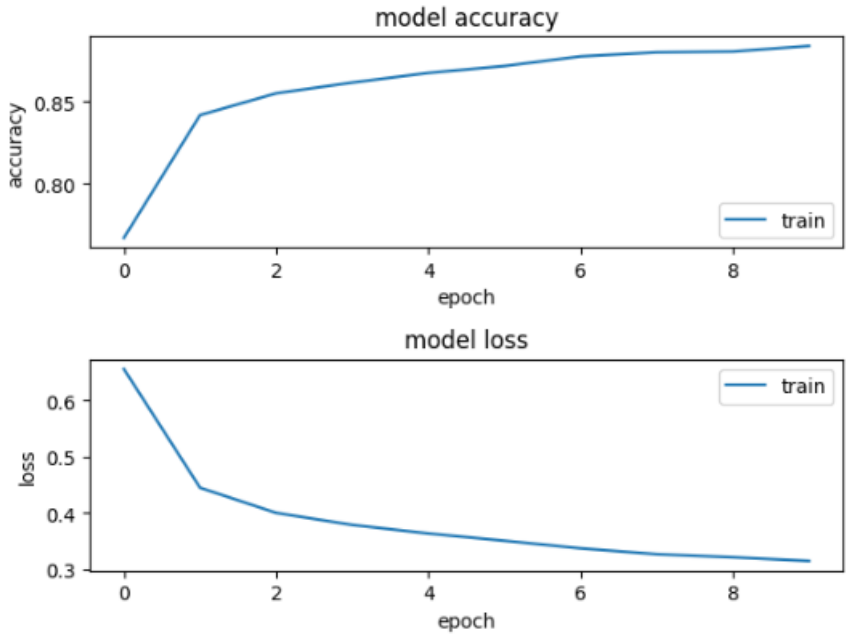
- **Độ chính xác:** 0.885
- **Đánh giá:** Tương tư như đánh giá ở trên và đầu tiên xét về độ chính xác từ tập test, ta thấy ở đây khá cao và cao hơn cả mô hình trên cho thấy mô hình không bị overfitting và đưa ra dự đoán tốt. Thứ hai, thời gian huấn luyện khá nhanh trong tầm 28.1 giây và tổng bộ nhớ mô hình chỉ chiếm 931.68KB với 238510 tham số. Cuối cùng là đường đi của 2 đường acc và loss trong mỗi epoch ở đồ thị trên rất ổn định và tăng hoặc giảm rất đều đặn cho thấy mô hình phù hợp dữ liệu đang huấn luyện. Tóm lại, việc tăng số node ở đây có ý nghĩa.

3. Mô hình neural network với 1 lớp ẩn và 500 nodes:



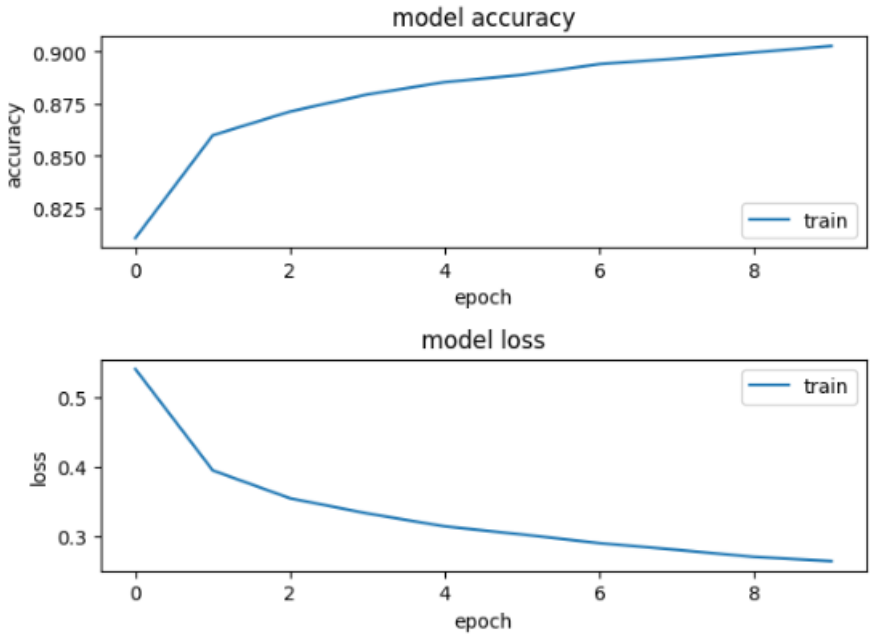
- **Độ chính xác:** 0.882
- **Đánh giá:** Tương tư như đánh giá ở trên và đầu tiên xét về độ chính xác từ tập test, ta thấy ở đây khá cao và nhưng không cao hơn mô hình trên nhưng vẫn cho thấy mô hình không bị overfitting và đưa ra dự đoán tốt. Thứ hai, thời gian huấn luyện cũng khá nhanh trong tầm 36.8 giây và tổng bộ nhớ mô hình chỉ chiếm 1.52MB với 397510 tham số. Cuối cùng là đường đi của 2 đường acc và loss trong mỗi epoch ở đồ thị trên rất ổn định và tăng hoặc giảm rất đều đặn cho thấy mô hình phù hợp dữ liệu đang huấn luyện. Tóm lại, việc tăng node ở đây không thực sự có ý nghĩa vì không tăng độ chính xác mà làm tăng bộ nhớ lưu trữ và thời gian xử lý.

4. Mô hình neural network với 2 lớp ẩn và 100 nodes:



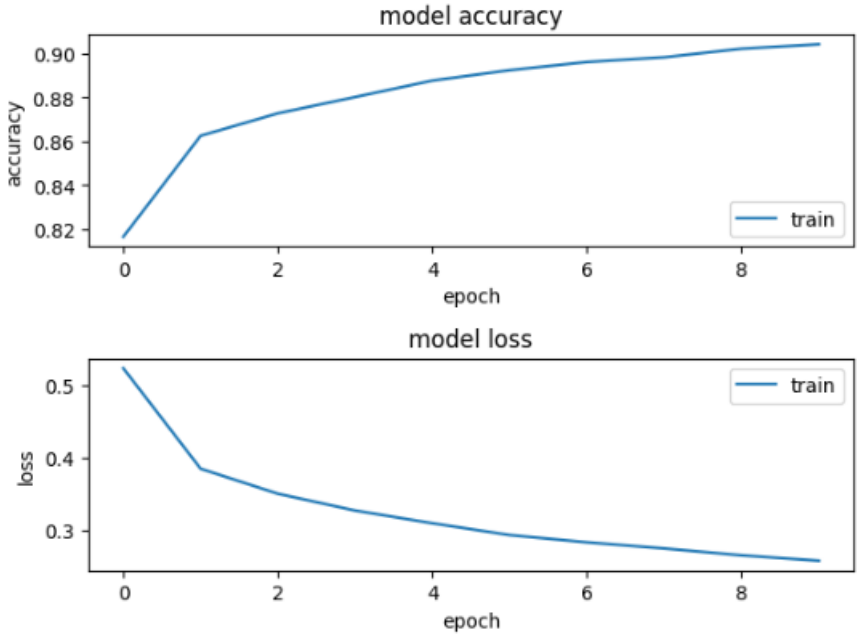
- **Độ chính xác**: 0.880
- **Đánh giá:** Đầu tiên xét về độ chính xác từ tập test, ta thấy ở đây khá cao không bị overfitting và rất tốt cho thấy mô hình được xây dựng đưa ra dự đoán tốt và độ chính xác cao hơn cả mô hình 1 layer với 100 nodes. Thứ hai, thời gian huấn luyện khá nhanh trong tầm 26.5 giây và tổng bộ nhớ mô hình chỉ chiếm 350.04KB với 89610 tham số. Cuối cùng là đường đi của 2 đường acc và loss trong mỗi epoch ở đồ thị trên rất ổn định và tăng hoặc giảm rất đều đặn cho thấy mô hình phù hợp dữ liệu đang huấn luyện. Tóm lại, việc tăng layer ở đây có ý nghĩa.

5. Mô hình neural network với 2 lớp ẩn và 300 nodes:



- **Độ chính xác:** 0.885
- **Đánh giá:** Đầu tiên xét về độ chính xác từ tập test, ta thấy ở đây khá cao không bị overfitting và rất tốt cho thấy mô hình được xây dựng đưa ra dự đoán tốt và độ chính xác cao hơn cả mô hình trên và không thấp hơn mô hình 1 layer với 300 nodes. Thứ hai, thời gian huấn luyện khá nhanh trong tầm 51.9 giây và tổng bộ nhớ mô hình chỉ chiếm 1.25MB với 328810 tham số. Cuối cùng là đường đi của 2 đường acc và loss trong mỗi epoch ở đồ thị trên rất ổn định và tăng hoặc giảm rất đều đặn cho thấy mô hình phù hợp dữ liệu đang huấn luyện. Tóm lại, việc tăng node và layer ở đây có ý nghĩa.

6. Mô hình neural network với 2 lớp ẩn và 500 nodes:



- **Độ chính xác**: 0.885
- **Đánh giá:** Đầu tiên xét về độ chính xác từ tập test, ta thấy ở đây khá cao không bị overfitting và rất tốt cho thấy mô hình được xây dựng đưa ra dự đoán tốt nhưng độ chính xác thấp hơn cả mô hình trên nhưng cao hơn mô hình 1 layer với 500 nodes. Thứ hai, thời gian huấn luyện chậm nhất trong tầm 80 giây và tổng bộ nhớ mô hình chỉ chiếm 2.47MB với 648010 tham số. Cuối cùng là đường đi của 2 đường acc và loss trong mỗi epoch ở đồ thị trên rất ổn định và tăng hoặc giảm rất đều đặn cho thấy mô hình phù hợp dữ liệu đang huấn luyện. Tóm lại, việc tăng layer ở đây có ý nghĩa nhưng việc tăng node thì không vì làm tăng bộ nhớ và thời gian chạy nhưng độ chính xác không tăng.

VIII. Đề xuất cách chọn số layer ẩn và số node:

1. Một số quy luật:

- Đối với cách chọn số lớp ẩn thì nếu dữ liệu có thể phân tách tuyến tính thì không cần bất kỳ lớp ẩn nào cả và cũng không cần NN để phân giải dữ liệu của mình, nhưng nó vẫn thực hiện được công việc. Nhưng nếu dữ liệu ít phức tạp hơn và có ít kích thước hoặc tính năng hơn thì mạng thần kinh có 1 đến 2 lớp ẩn sẽ hoạt động. Cuối cùng, nếu dữ liệu có kích thước hoặc tính năng lớn thì để có được giải pháp tối ưu, có thể sử dụng 3 đến 5 lớp ẩn. Tuy nhiên, cần lưu ý rằng việc tăng các lớp ẩn cũng sẽ làm tăng độ phức tạp của mô hình và việc chọn các lớp ẩn như 8, 9 hoặc hai chữ số đôi khi có thể dẫn đến việc overfitting.

- Đối với số node có một số quy tắc kinh nghiệm rút ra từ thực nghiệm thì điều thường được dựa vào nhất là ' kích thước tối ưu của lớp ẩn thường nằm giữa kích thước của lớp đầu vào và kích thước của lớp đầu ra '. Jeff Heaton, tác giả cuốn Giới thiệu về Mạng thần kinh trong Java. Số lượng node thích hợp sẽ bằng √(nút lớp đầu vào * nút lớp đầu ra). Cuối cùng, số lượng node nên tiếp tục giảm ở các lớp tiếp theo để ngày càng tiến gần hơn đến việc trích xuất mẫu và tính năng cũng như để xác định lớp mục tiêu.

- Tóm lại, đối với hầu hết các vấn đề để có thể đạt được hiệu suất tốt bằng cách thiết lập cấu hình lớp ẩn chỉ bằng hai quy tắc: số lớp ẩn bằng một và số lượng nơ-ron trong lớp đó là giá trị trung bình của các node ở lớp đầu vào và đầu ra.

2. Đối với tập dữ liệu MNIST:

- Nếu dựa theo thực nghiệm thì sau khi đã huấn luyện và đánh giá độ chính xác của các mô hình, ta thấy được mô hình lý tưởng nhất cho tập dữ liệu này là mô hình 1 layer với 300 nodes vì đây là mô hình có độ chính xác cao nhất với bộ nhớ sử dụng và thời gian chạy lý tưởng.

- Theo lý thuyết thì số node lý tưởng sẽ là √(nút lớp đầu vào * nút lớp đầu ra) = √(784 * 10) ≈ 88 và gần bằng 100 nên ta chọn 100 node. Trong đó, 784 tương đương với số pixel và 10 tương ứng với 10 lớp nhãn. Về số layer thì với bộ dữ liệu phân biệt chữ số không quá phức

tạp thì ta có thể chọn 1 layer ở đây.

3. Đối với tập dữ liệu MNIST-FASHION:

- Tương tự, nếu dựa theo thực nghiệm thì sau khi đã huấn luyện và đánh giá độ chính xác của các mô hình, ta thấy được mô hình lý tưởng nhất cho tập dữ liệu này là mô hình 2 layers với 300 nodes vì đây là mô hình có độ chính xác cao nhất với bộ nhớ sử dụng và thời gian chạy lý tưởng.

- Và cũng theo lý thuyết thì số node lý tưởng sẽ là √(nút lớp đầu vào * nút lớp đầu ra) = √(784 * 10) ≈ 88 và gần bằng 100 nên ta chọn 100 node. Trong đó, 784 tương đương với số pixel và 10 tương ứng với 10 lớp nhãn. Về số layer thì phân biệt quần áo sẽ khá phức tạp hơn chữ số thì ta có thể chọn 2 layers ở đây.

IX. Giải thích kết quả và sự khác biệt của chúng:

- Đầu tiên, đối với MNIST, dữ liệu đơn giản hơn so với MNIST-Fashion, nên mô hình cần ít hơn về số layer để học các đặc trưng cơ bản.
- Cả hai đều có cùng số đặc trưng pixel đầu vào nên đều có thể có cùng số node.
- Với cùng một mô hình và huấn luyện trên 2 tập dữ liệu này thì độ chính xác của tập MNIST luôn cao hơn tập MNIST-FASHION với lý do là nhận diện chữ số sẽ ít chi tiết hơn trong việc phát hiện các chi tiết trong trang phục. Bên cạnh đó, các đường nét trong chữ số sẽ dễ dàng dự đoán hơn trang phục.
- Thời gian chạy và bộ nhớ lưu trữ của 2 tập dữ liệu trên cùng mô hình gần tương đồng nhau do có cùng kích thước đầu vào.
- Với cùng một tập dữ liệu MNIST thì các mô hình với 2 layers có độ chính xác luôn thấp hơn các mô hình 1-layer với cùng số node. Trong khi đó, với cùng tập dữ liệu MNIST-FASHION thì ngược lại các mô hình với 2 layers có độ chính xác luôn cao hơn mô hình 1-layer với cùng số node. Vì tập dữ liệu MNIST-FASHION có chi tiết xử lý phức tạp hơn nên cần nhiều layers để mô hình nhận biết tổng quát hơn và nhận biết tốt các chi tiết nhỏ. Còn tập dữ liệu MNIST thì đơn giản hơn khá nhiều chỉ cần nhận biết các đường nét nên không cần có quá nhiều layers nó sẽ gây nhiễu và mất thông tin, giảm hiệu suất mô hình.

X. Hướng dẫn cách chạy:

1. Đối với tập dữ liệu MNIST:

Các bước chạy

Bước 1: Mở VSCode.

Bước 2: Chọn File → Open Folder → Chọn thư mục Source chứa source code và data.

Bước 3: Nhấp chọn file juypter notebook với tên MNIST.ipynb.

Bước 4: Ấn vào nút Run All để khởi chạy chương trình.

Bước 5: Chương trình sẽ yêu cầu chọn ngôn ngữ lập trình thì ở đây chúng ta sẽ chọn ngôn ngữ Python 3.6 trở lên.

Bước 6: Trong quá trình chạy đảm bảo rằng các thư viện được sử dụng trong chương trình đã được tải về hết với lệnh pip install [tên thư viện].

Bước 7: Đợi chương trình chạy hoàn thành và lướt xuống để xem kết quả.

2. Đối với tập dữ liệu MNIST-FASHION:

Các bước chạy

Bước 1: Mở VSCode.

Bước 2: Chọn File → Open Folder → Chọn thư mục Source chứa source code và data.

Bước 3: Nhấp chọn file juypter notebook với tên MNIST-FASHION.ipynb.

Bước 4: Ấn vào nút Run All để khởi chạy chương trình.

Bước 5: Chương trình sẽ yêu cầu chọn ngôn ngữ lập trình thì ở

đây chúng ta sẽ chọn ngôn ngữ Python 3.6 trở lên.

Bước 6: Trong quá trình chạy đảm bảo rằng các thư viện được sử dụng trong chương trình đã được tải về hết với lệnh pip install [tên thư viện].

Bước 7: Đợi chương trình chạy hoàn thành và lướt xuống để xem kết quả.

XI. Tài liệu tham khảo:

[1] **Slides – Google Drive. (n.d.-b).** **https://drive.google.com/drive/folders/1DIE9e6TLL8Iz8BacyBnmzZu-8O2MN70I**

[2] **How to choose the number of hidden layers and nodes in a feedforward neural network? (n.d.). Cross Validated.** **https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw**

[3] **Sachdev, H. S. (2021, April 20). Choosing number of Hidden Layers and number of hidden neurons in Neural Networks.** **https://www.linkedin.com/pulse/choosing-number-hidden-layers-neurons-neural-networks-sachdev/**

[4] **Heeraldedhia. (2020, September 19). MNIST Classifier - first Deep Learning project. Kaggle.** **https://www.kaggle.com/code/heeraldedhia/mnist-classifier-first-deep-learning-project**

[5] **What are the differences between a shallow network and a deep network? | MLExpert - Crush Your Machine Learning interview. (n.d.). MLExpert.**

https://www.mlexpert.io/machine-learning/interview-questions/shallow-and-deep-networks