ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

Tiểu luận

CS331.M21.KHCL - Thị giác máy tính nâng cao

NEURAL NETWORK

Giảng viên hướng dẫn:

TS. Nguyễn Vĩnh Tiệp

Nhóm thực hiện:

Nguyễn Thành Vương	19522542
Lê Thị Thanh Thanh	19520954
Huỳnh Thiện Tùng	19522492

TP. HỒ CHÍ MINH, THÁNG 04/2022

Mục lục

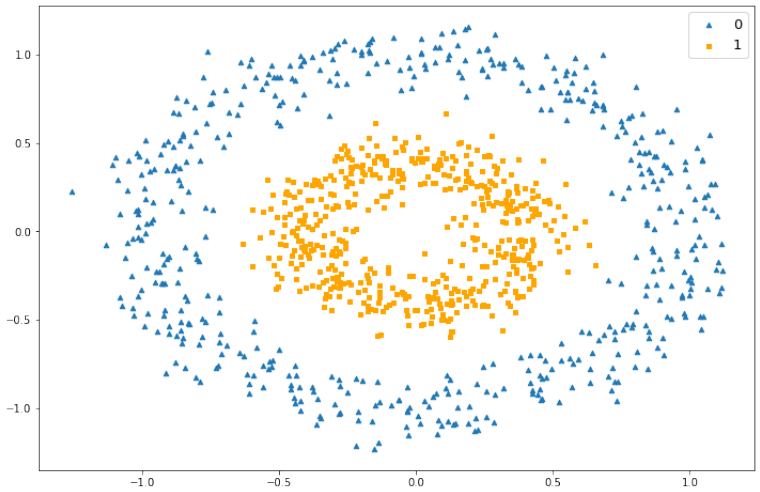
1 Giới thiệu chung

1.1 Giới Thiệu

Mạng nơ-ron nhân tạo - Neural Network (NN) là một trong những mô hình mạng mang lại những kết quả đột phá trong những năm gần đây. Nó được ứng dụng rộng rãi trong mọi lĩnh vực, từ khoa học đến kinh tế – xã hội, như điện, điện tử, tài chính, ngân hàng, dự báo thời tiết, nghiên cứu thị trường, dự đoán giá cổ phiếu,... Không chỉ vậy, nguồn gốc và ý tưởng hình thành mô hình Neural Network cũng rất gũi với con người chúng ta. Trong bài báo cáo này, nhóm em sẽ giới thiệu cũng như phân tích chi tiết về mô hình thú vị này.

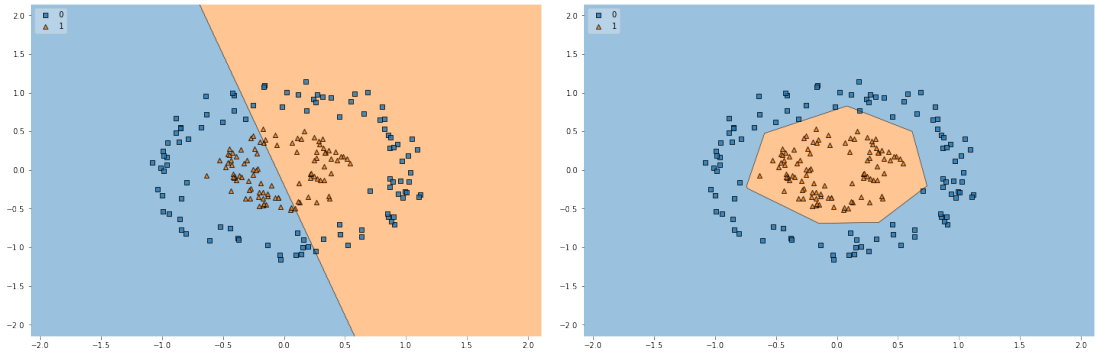
1.2 Đặt vấn đề

Về mặt lý thuyết, khi giải quyết bài toán chúng ta sẽ gặp 2 dạng dữ liệu: phân biệt tuyến tính (*Linearly separable*) và không phân biệt tuyến tính (*Non-linearly separable*), tùy vào mỗi dạng dữ liệu mà chúng ta sẽ đưa ra giải pháp khác nhau để giải quyết bài toán và chắc chắn một điều là bài toán với dữ liệu phân biệt tuyến tính sẽ dễ dàng giải quyết hơn so với dạng còn lại. Chúng ta có thể dùng các thuật toán đơn giản và phổ biến như: *Linear Regression, Logistic Regression, Softmax Regression,...* để giải quyết bài toán. Tuy nhiên, đa số các bài toán trong thực tế đều có dữ liệu không phân biệt tuyến tính, điều này gây khó dễ cho chúng ta trong việc phân loại và dự đoán đối tượng kể cả khi áp dụng các thuật toán đề cập ở trên. Xét ví dụ đơn giản dưới đây:



Hình 1: Phân bố dữ liệu không phân biệt tuyến tính

Có thể nhận thấy ở hình 1, dữ liệu của 2 lớp được phân bố khá đặc biệt và không phân biệt tuyến tính. Tại đây, nhóm sử dụng 2 mô hình: Logistic Regression và Neural Network. Hãy lần lượt xem kết quả của từng mô hình:



(a) Logistic Regresstion (b) Neural Network

Hình 2: Kết quả phân lớp sau khi sử dụng 2 mô hình

Dễ dàng nhận thấy, độ chính xác (*Precision*) và độ phủ (*Recall*) của mô hình sử dụng Neural Network tốt hơn rất nhiều so với mô hình sử dụng Logistic Regression. Rõ ràng khi gặp các dữ liệu không phân biệt tuyến tính thì các mô hình như Logistic Regression, Softmax Regression khó lòng hoạt động tốt vì ranh giới phân chia các lớp (*Decision boundary*) là một đường thẳng. Vậy chúng ta cần làm gì để cải thiện kết quả ? Câu trả lời là chúng ta cần tìm 1 decision boundary linh hoạt về hình dạng như: tròn, elipse, đa giác, ... và Neural Network chính là câu trả lời cho vấn đề ấy.

2 Phương pháp

2.1 Ý tưởng

Neural Network được lấy ý tưởng từ mô hình và cách thức hoạt động cơ bản của hệ thống tế bào thần kinh.

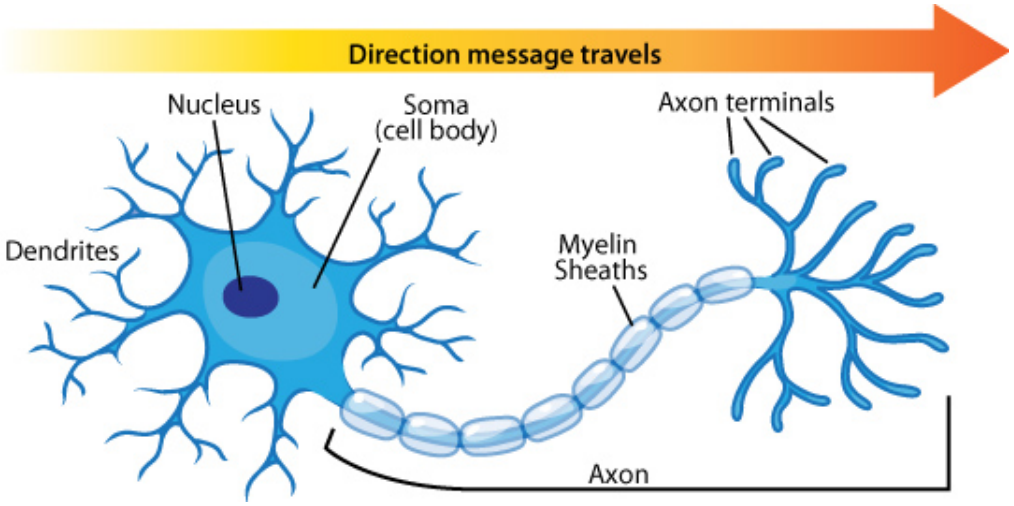
Nơ-ron là đơn vị cơ bản cấu tạo hệ thống thần kinh và là một phần quan trọng nhất của não. Não chúng ta gồm khoảng 100 tỉ nơ-ron và mỗi nơ-ron liên kết với 100 tỉ nơ-ron khác.

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.

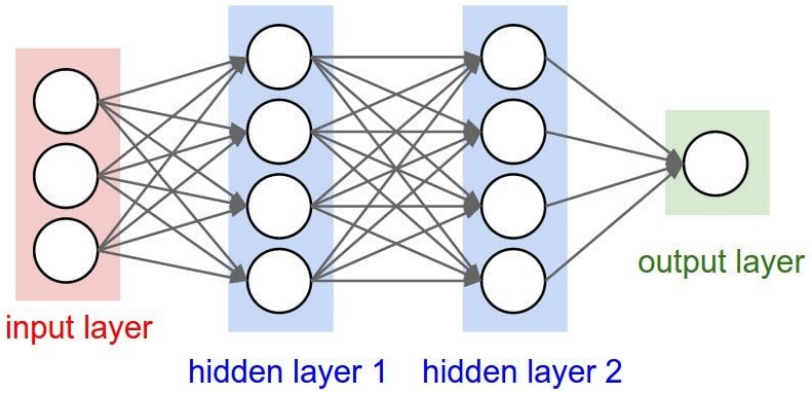
Mỗi nơ-ron nhận xung điện từ các nơ-ron khác qua sợi nhánh. Nếu các xung điện này đủ lớn để kích hoạt nơ-ron, thì tín hiệu này đi qua sợi trục đến các sợi nhánh của các nơ-ron khác. => Ở mỗi nơ-ron cần quyết định có kích hoạt nơ-ron đấy hay không.

Tuy nhiên Neural Network 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.



Hình 3: Cấu trúc tế bào thần kinh ở người

2.2 Kiến trúc mạng



Hình 4: Mô hình tổng quát của Neural Network

Nói một cách đơn giản, Neural Network được cấu tạo từ nhiều tầng (layer) lại với nhau. Trong mỗi layer lại bao gồm nhiều nơ-ron (node). Chúng ta có thể chia làm ***03*** tầng chính :

1. **Input layer**: nằm bên trái của sơ đồ mạng, mỗi 1 Input sẽ tương ứng với một đầu vào của mạng.

2. **Hidden layer**: là tầng trung gian giữa Input layer và Output layer, nó thể hiện quá trình suy luận logic của mạng. Tầng này nhận thông tin từ tầng liền kề trước nó, sau đó xử lý rồi chuyển thông tin đã được xử lý đến tầng sau nó. Một mạng Neural Network có thể có 1 hoặc nhiều Hidden layer, thậm chí là không có Hidden layer.

3. **Output layer**: nằm bên phải ngoài cùng của sơ đồ mạng, mỗi một Output tượng trưng cho một đầu ra hay kết quả của mạng dữ liệu. Khi xuất hiện Output nghĩa là vấn đề cần giải quyết đã được Neural Network xử lý để cho ra giải pháp.

Quá trình suy luận từ Input layer tới Output layer của mạng Neural Network là quá trình lan truyền tiến (feedforward), tức là đầu vào các nơ-ron tại 1 tầng đều lấy từ kết quả các nơ-ron tầng trước đó mà không có quá trình suy luận ngược lại.

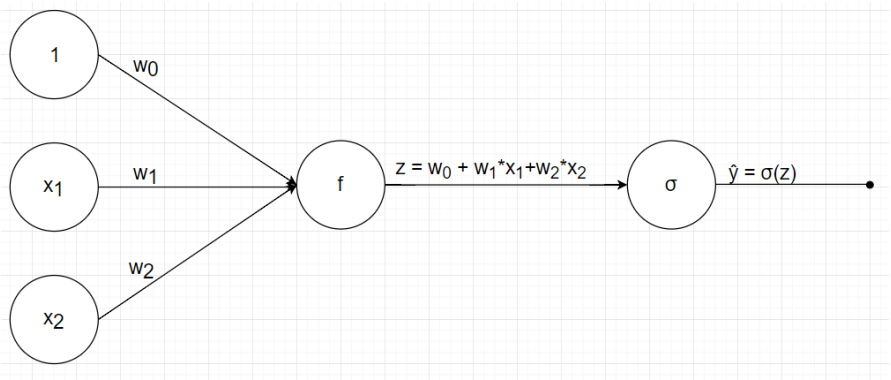
Mỗi node trong Hidden layer và Output layer:

1. Liên kết với tất cả các node ở layer trước đó với các hệ số w riêng.

2. Mỗi node có 1 hệ số *bias* **b** riêng.

3. Diễn ra 2 bước: tính tổng Linear và áp dụng hàm kích hoạt (*Activation function*)

Sau khi phân tích kiến trúc mạng Neural Network, có thể thấy Logistic Regression là trường hợp đơn giản nhất của Neural Network chỉ với Input layer và Output layer như hình bên dưới:



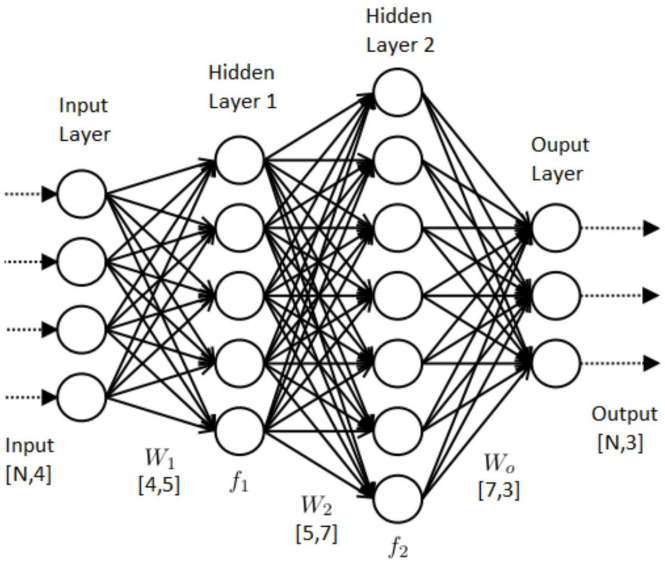
Hình 5: Mô hình Logistic Regression

Cũng tương tự như mô hình Neural Network, Logistic Regression cũng trải qua 2 bước:

1. Tính tổng linear: $z = w_0 + w_1 * x_1 + w_2 * x_2$ (w_0 là bias)
2. Áp dụng hàm kích hoạt Sigmoid: $\hat{y} = \sigma(z)$

Điểm tạo ra sự khác biệt giữa 2 mô hình là Logistic Regression không có Hidden layer. Điều này làm cho Neural Network có kiến trúc phức tạp và "nhạy cảm" với overfitting hơn so với Logistic Regression. Tuy nhiên, chúng ta có thể giới hạn kích thước của mô hình mạng bằng cách giảm số lượng nơ-ron ở lớp ẩn, giảm số lượng biến,...

2.3 Các đặc trưng của Neural Network



Hình 6: Ví dụ đơn giản về các đặc trưng của Neural Network

1. Gồm một tập các đơn vị xử lý được gọi là unit (các noron nhân tạo)
2. Đầu ra của đơn vị xử lý.
3. Liên kết giữa các unit. Xét tổng quát, mỗi liên kết được định nghĩa bởi một trọng số W_{jk} cho ta biết hiệu ứng mà tín hiệu của unit j có trên lớp k
4. Một giải thuật lan truyền quyết định cách tính tín hiệu ra của từng unit trong lớp hiện tại từ đầu ra của lớp phía trước.

5. Một hàm kích hoạt (activation function)
6. Một đơn vị điều chỉnh (độ lệch) (bias, offset) của mỗi unit
7. Môi trường để hệ thống có thể hoạt động.

3 Cài đặt và sử dụng Neural Network

3.1 Bộ dữ liệu MNIST

MNIST (Modified National Institute of Standards and Technology database), là một bộ dữ liệu nổi tiếng về các chữ số viết tay thường được sử dụng để đào tạo các hệ thống xử lý ảnh cho học máy. Các ảnh trong bộ dữ liệu MNIST đều là ảnh xám (gray scale) và có kích thước 28x28 pixels. MNIST nổi tiếng vì tần suất mà bộ dữ liệu được sử dụng. Nó phổ biến vì hai lý do:

1. Người mới bắt đầu sử dụng nó vì nó dễ dàng.
2. Các nhà nghiên cứu sử dụng nó để chuẩn (so sánh) các mô hình khác nhau.
3. Kích thước của bộ dữ liệu này phù hợp cho quá trình huấn luyện.

Với những lí do trên, nhóm quyết định sử dụng bộ dữ liệu MNIST.

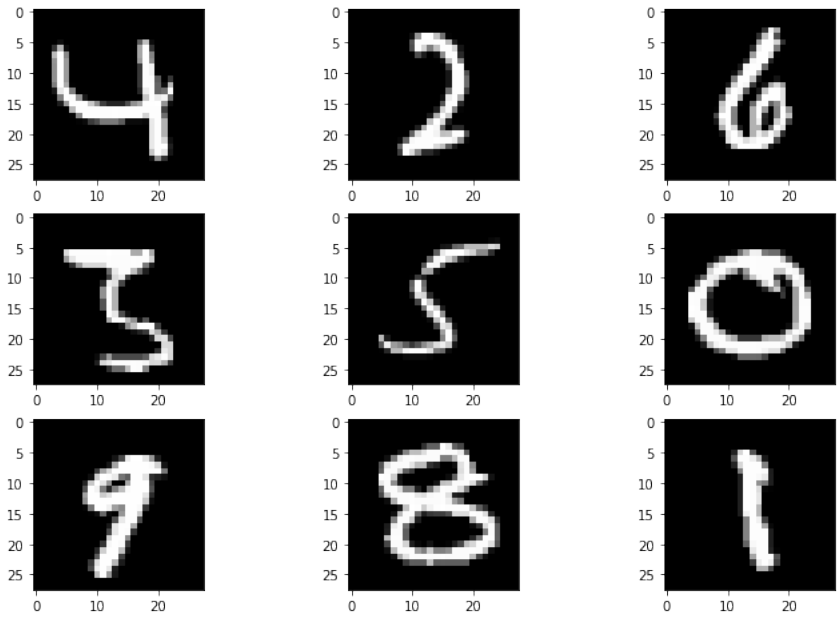
3.2 Đọc và trực quan dữ liệu

Chúng ta sẽ sử dụng module **tf.keras.datasets.mnist** để load bộ dữ liệu MNIST:

```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
11501568/11490434 [==============================] - 1s 0us/step
```

Sau khi đọc bộ dữ liệu thành công, chúng ta sẽ sử dụng module **matplotlib** để trực quan bộ dữ liệu bằng cách tạo và nối các subplot lại với nhau:



Hình 7: Một số ảnh trong bộ dữ liệu MNIST

3.3 Tiền xử lý dữ liệu

Như đã giới thiệu ở trên, MNIST gồm các ảnh xám và có kích thước 28x28 pixels. Chúng ta sẽ xây dựng một mô hình mạng xử lý vector có độ dài là 784. Mỗi pixel có giá trị từ 0 đến 255, với 0 là pixel có giá trị màu đen và 255 là pixel có giá trị màu trắng. Vì chúng ta đang làm việc với ảnh xám và để hạn chế tính toán với các trọng số lớn, chúng ta sẽ chuẩn hóa dữ liệu về khoảng [0;1] bằng cách chia cho 255.

```
X = mnist.data.astype('float32')
y = mnist.target.astype('int64')
X /= 255.0
```

Kết quả sau khi xử lý dữ liệu:

```
print(f'Min value: {X.min()}')
print(f'Max value: {X.max()}')
```

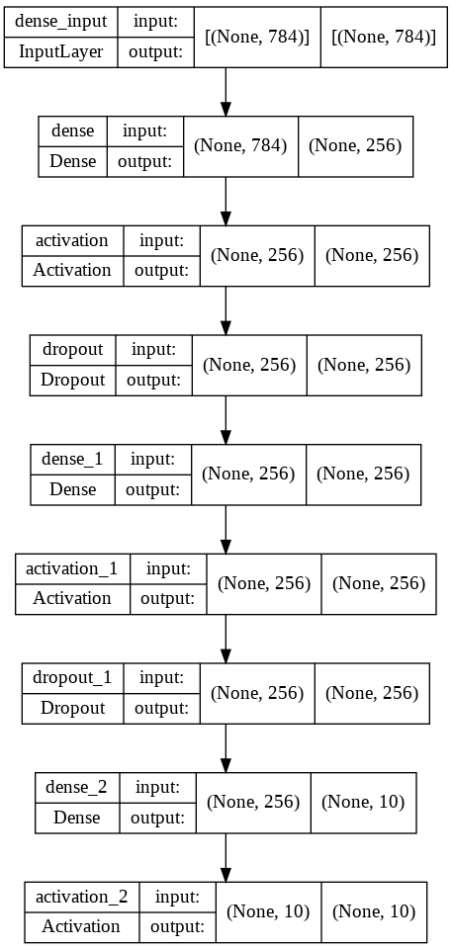
```
Min value: 0.0
Max value: 1.0
```

Hiện tại, nhãn (label) của tập dữ liệu vẫn ở dạng giá trị vô hướng rời rạc (sparse scalar) như [1], [2], [3],... Nó không phù hợp với tầng đưa ra dự đoán của Neural Network vì output của nó là xác suất mỗi lớp. Vì vậy, bước kế tiếp chúng ta sẽ tiến hành one-hot label để được các one-hot vector bằng cách sử dụng module **to_categorical** của **tensorflow**:

```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

3.4 Xây dựng mô hình Neural Network

3.4.1 Thiết kế mô hình



Hình 8: Kiến trúc mô hình Neural Network

Ý tưởng kiến trúc mạng Neural Network như sau:

1. Kiến trúc Neural Network nhóm xây dựng sẽ gồm 03 mạng nơ-ron truyền thẳng nhiều lớp (Multilayer perceptron - MLP). Các lớp này được hiện thực thông qua lớp **Dense** của **Keras**.

2. MLP layer thứ 1 và thứ 2 sẽ đều có số node là 256, theo sau chúng sẽ là hàm kích hoạt **ReLU** và thực hiện **Dropout** để hạn chế sự phức tạp của mô hình.

3. Output layer sẽ dùng hàm kích hoạt **Softmax**.

Sở dĩ chúng em chọn **256** node ở MLP layer 1 và layer 2 vì khi chọn các giá trị như 128, 512, 1024 cho kết quả tệ hơn. Với giá trị 128, mô hình hội tụ nhanh hơn nhưng độ chính xác trên tập test lại thấp hơn. Đối với 512 và 1024, mô hình sẽ trở nên phức tạp hơn và dễ trở nên overfitting hơn.

3.4.2 Khởi tạo và huấn luyện mô hình

Để khởi tạo mô hình ở Hình 8, ta chạy đoạn mã dưới đây:

```
hidden_units = 256 # Your choice
input_size = 784 # Vector 784 dimensions
num_class = 10 # 0 -> 9 => 10 classes

model = Sequential()
model.add(Dense(hidden_units, input_dim=input_size))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(hidden_units))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(Dense(num_class))
model.add(Activation('softmax'))
```

Sau khi khởi tạo, chúng ta có thể xem tổng quan model bằng phương thức **summary()**:

```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 256)               200960

 activation_15 (Activation)  (None, 256)               0

 dropout_10 (Dropout)        (None, 256)               0

 dense_16 (Dense)            (None, 256)               65792

 activation_16 (Activation)  (None, 256)               0

 dropout_11 (Dropout)        (None, 256)               0

 dense_17 (Dense)            (None, 10)                2570

 activation_17 (Activation)  (None, 10)                0

=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
```

Hình 9: Tổng quan mô hình

Dựa vào Hình 9, mô hình chúng ta đã xây dựng cần *269.322* tham số, đây là một con số đáng kể khi chúng ta chỉ thực hiện nhiệm vụ đơn giản là phân loại các chữ số trong tập MNIST. Tổng số tham số yêu cầu có thể được tính như sau:

- Từ Input layer đến Dense layer 1: $784 \times 256 + 256 = 200{,}960$.
- Từ Dense layer 1 đến Dense layer 2: $256 \times 256 + 256 = 65{,}792$.
- Từ Dense layer 2 đến Output layer: $10 \times 256 + 10 = 2{,}570$.

$\rightarrow$ Tổng tham số yêu cầu là: $200{,}690 + 65{,}972 + 2{,}570 = 269{,}322$.

Sau khi đã nắm được tổng quan mô hình, chúng ta sẽ tiến hành compile mô hình:

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

- Vì đây là bài toán phân loại đa lớp (Multiclass classification) nên chúng ta sẽ sử dụng hàm loss **categorical_crossentropy**.

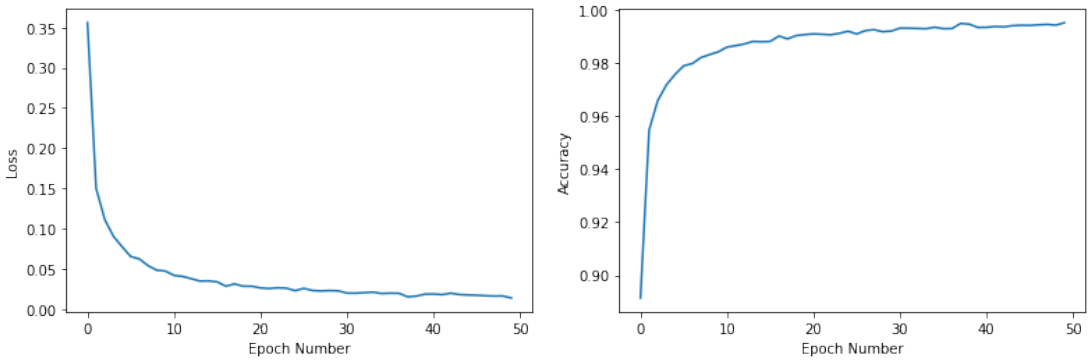
- Thuật toán tối ưu sử dụng là **adam**. Nó sẽ tiến hành tối ưu các hyperparameter như: *learning rate, momentum, decay,...*
- Phương thức đánh giá hiệu năng mô hình sẽ là độ chính xác *(Accuracy)*. Độ chính xác sẽ được tính bằng số dự đoán chính xác trên tổng số dự đoán đưa ra.

Nào, chúng ta cùng đến bước cực kỳ quan trọng đó là huấn luyện mô hình:

```
history = model.fit(X_train, y_train, epochs=20, batch_size=128)
```

- Vì huấn luyện trên tập train nên chúng ta sẽ truyền vào X_train, y_train.
- Với mô hình mạng đã đề xuất, nhóm chọn số **epoch** là 20 vì trong quá trình quan sát, giá loss đã bắt đầu bão hòa tại giai đoạn này.
- Giá trị **batch_size** nhóm chọn là 128. Con số này có thể thay đổi thành 32, 64,... tùy vào khả năng phần cứng của máy.

3.5 Kết quả



(a) Biểu đồ loss (b) Biểu đồ accuracy

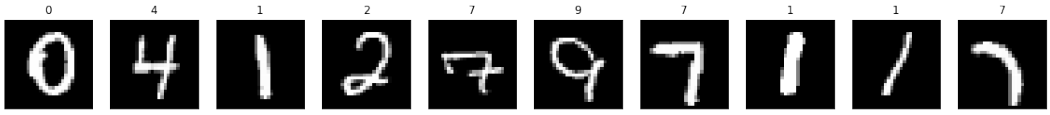
Hình 10: Biểu đồ kết quả quá trình huấn luyện

Nhìn vào 2 biểu đồ, ta có thấy mô hình Neural Network đề xuất cho kết quả khá tốt trên tập train. Với biểu loss, loss giảm từ 0.35 xuống còn 0.02 qua mỗi epoch. Còn về phía accuracy, accuracy tăng từ 0.89 lên đến 0.99 qua mỗi epoch. Vậy còn đối với tập test thì sao ?

```
loss, acc = model.evaluate(X_test, y_test, batch_size=128)
print("\nTest accuracy: %.1f%%" % (100.0 * acc))
```

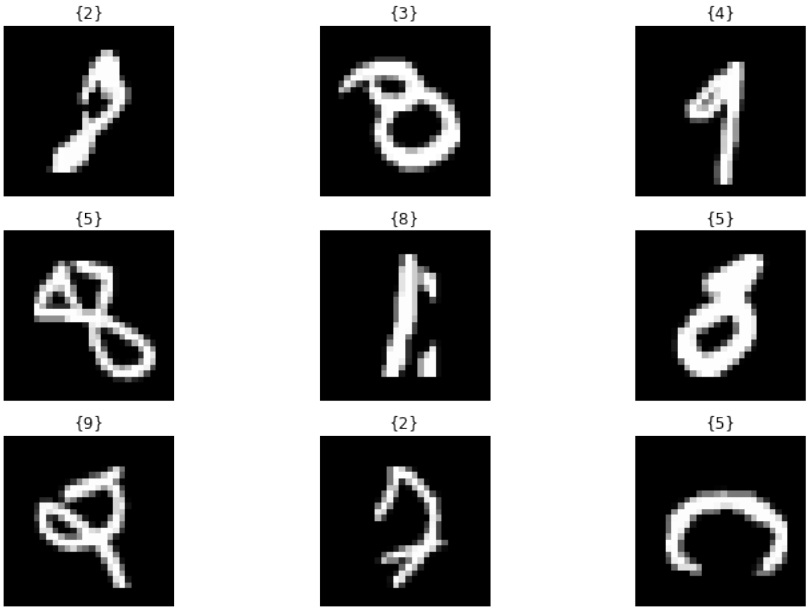
```
110/110 [==============================] - 1s 3ms/step - loss: 0.0792 - accuracy: 0.9804

Test accuracy: 98.0%
```



Hình 11: Kết quả dự đoán đúng trên tập test

Có thể thấy kết quả dự đoán trên tập test của mô hình rất tốt. Tuy nhiên, mô hình vẫn dự đoán sai **175/10000** (khoảng 2% trên tập test). Trong đó số dự đoán sai nhiều nhất là số **8** với **30** lần dự đoán sai.



Hình 12: Kết quả dự đoán sai trên tập test

Xét tổng quan, mô hình Neural Network đã xây dựng cho kết quả rất tốt, thời gian huấn luyện cũng tương đối nhanh. Để thể hiện sức mạnh của mô hình Neural Network, nhóm sẽ so sánh kết quả với mô hình Softmax Regression.

```
Epoch: 0001, Loss: 2.745756889
Epoch: 0002, Loss: 1.084883546
Epoch: 0003, Loss: 0.856760169
Epoch: 0004, Loss: 0.749665617
Epoch: 0005, Loss: 0.683298426
Epoch: 0006, Loss: 0.636348472
Epoch: 0007, Loss: 0.600593829
Epoch: 0008, Loss: 0.572091515
Epoch: 0009, Loss: 0.548642030
Epoch: 0010, Loss: 0.528886259
Epoch: 0011, Loss: 0.511927748
Epoch: 0012, Loss: 0.497148226
Epoch: 0013, Loss: 0.484105674
Epoch: 0014, Loss: 0.472474599
Epoch: 0015, Loss: 0.462009432
Epoch: 0016, Loss: 0.452521032
Epoch: 0017, Loss: 0.443861151
Epoch: 0018, Loss: 0.435911763
Epoch: 0019, Loss: 0.428577637
Epoch: 0020, Loss: 0.421781017
Learning finished
Accuracy:  0.8981
```

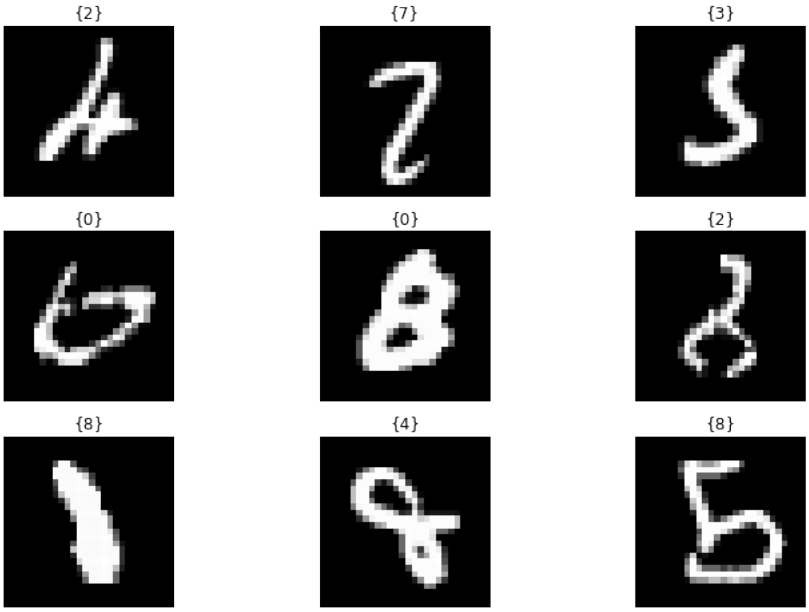
Hình 13: Kết quả trên tập test của mô hình Softmax Regression

Về mô hình Softmax Regression, nhóm cũng đã thực hiện những bước như: đọc dữ liệu, tiền xử lý dữ liệu,... trước khi huấn luyện mô hình. Như Hình 13, độ chính xác trên tập test MNIST của Softmax Regression là *89%* (8957/10000 ảnh), con số khá ổn khi mô hình này còn khá đơn giản. Để tiếp tục thể hiện sức mạnh của mô hình Neural Network, nhóm đã lấy các kết quả dự đoán sai của mô hình Softmax Regression để dự đoán trên mô hình Neural Network. Kết quả thu được như sau:

```
Number most wrong predicted: 8
Number of times wrong: 21
Wrong Image predicted: 128
```

Hình 14: Thử nghiệm so sánh 2 mô hình

- Dự đoán đúng **915/1043** ảnh mà Softmax Regression đã dự đoán sai. Chính xác hơn khoảng 9%.
- Số **8** vẫn là số bị dự đoán sai nhiều nhất (21 lần).



Hình 15: Một số ảnh mà cả 2 mô hình đều dự đoán sai

Tuy mô hình Neural Network cho kết quả tốt hơn mô hình Softmax Regression nhưng vẫn còn nhiều ảnh dự đoán sai tương tự với mồ hình khác. Chúng ta có thể cải thiện mô hình bằng cách: thêm hidden layer, thay đổi số node ở các layer, thay đổi kiến trúc mô hình, thay đổi hàm kích hoạt phi tuyến khác,... Có rất nhiều cách để tunning mô hình. Tuy nhiên, không phải lúc nào cái gì càng nhiều thì cũng càng tốt (số lượng hidden layer, số node, ...), chúng ta phải biết linh hoạt điều chỉnh các giá trị này thì mô hình mới có thể tránh overfitting và đạt kết quả như mong muốn.

4 Kết luận

Mô hình Neural Network đã và đang trở nên phổ biến trong xu hướng phát triển của lĩnh vực Computer Science. Với sự đóng góp từ phía cộng đồng về mặt dữ liệu và sự phát triển vượt bậc của phần cứng tính toán đã giúp mô hình Neural Network càng dễ hiện thực vào những bài toán phức tạp. Việc nắm rõ những kiến thức nền tảng của mô hình Neural Network sẽ dễ dàng giúp chúng ta có thể tiếp cận được các bài toán thực tế hiện nay.

5 Tài liệu tham khảo

Tài liệu

[1] Neural Network on MNIST dataset: https://colab.research.google.com/drive/1j53QZLyQB2j5PvYdtyAnVeW_SxMRCBUz?usp=sharing

[2] Softmax Regression on MNIST dataset: https://colab.research.google.com/drive/1LVg3-i0UxzMZ81YbvfF9vC2p2b9DJi8W?usp=sharing

[3] Regression vs Neural Network, *Siddhartha Sharma:* https://nsiddharthasharma.medium.com/regression-vs-neural-networks-f8e854b8bef5

[4] [NN] Mạng nơ-ron nhân tạo - Neural Networks, *Nguyen Duy Sim:* https://viblo.asia/p/nn-mang-no-ron-nhan-tao-neural-networks-bWrZn6dwZxw

[5] MNIST - Deep Neural Network with Keras, *PRASHANT BANERJEE:* https://www.kaggle.com/code/prashant111/mnist-deep-neural-network-with-keras/notebook