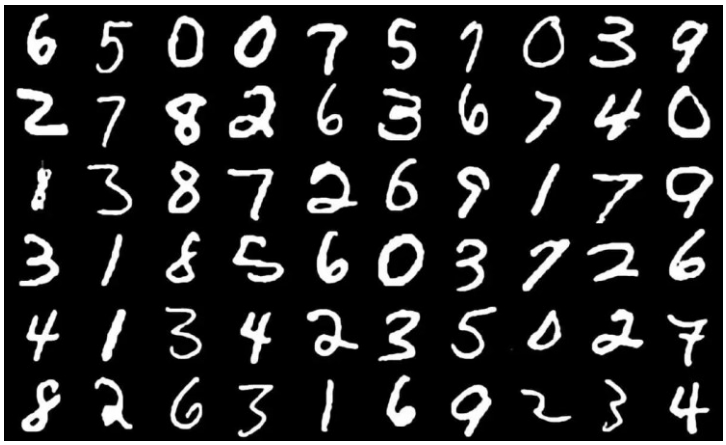
Bài thực hành 8. LÀM QUEN VỚI MẠNG NEURAL CƠ BẢN

1. LOAD DỮ LIỆU

Trong bài này, chúng ta sẽ sử dụng bộ dữ liệu MNIST. Đây là bộ dữ liệu sử dụng cho bài toán nhận dạng chữ số viết tay và được thư viện Keras hỗ trợ.



Minh họa bộ dữ liệu MNIST

Để load bộ dữ liệu MNIST do thư viện Keras cung cấp (bao gồm tập train và tập test), chúng ta sử dụng câu lệnh sau:

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

2. CHUẨN BỊ DỮ LIỆU

(?) Sử dụng thư viện Matplotlib để trực quan hóa ảnh train[0] và ảnh test[0]?

Để xem kích thước của dữ liệu, chúng ta sử dụng thuộc tính shape.

```
X_train.shape
```

(?) Hãy khảo sát bộ dữ liệu MNIST do thư viện Keras cung cấp:

- Xác định số lượng ảnh của tập train và tập test?
- Tính tỉ lệ train : test?
- Xác định kích thước của mỗi ảnh trong tập train và tập test?

(?) Xác định số lượng nhãn và liệt kê các nhãn có trong tập train?

(?) Sử dụng thư viện Matplotlib để vẽ biểu đồ cột thể hiện sự phân bố các nhãn có trong tập train?

Để điều chỉnh kích thước của dữ liệu, chúng ta sử dụng hàm reshape do thư viện Numpy cung cấp: Đưa dữ liệu về kích thước (m, 784), với m là số lượng dữ liệu.

```
X_train_reshaped = X_train.reshape(-1, 784)
```

(?) Sau khi thực hiện thao tác reshape trên tập train, cho biết mỗi điểm dữ liệu là

một vector có số chiều là bao nhiêu?

Tương tự, hãy thực hiện thao tác reshape cho tập test.

Chúng ta cần xây dựng một binary class matrix ứng với số lượng nhãn bằng hàm to_catergorical như sau:

```
from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train, num_classes=10)
```

3. XÂY DỰNG MẠNG NEURAL BẰNG KERAS

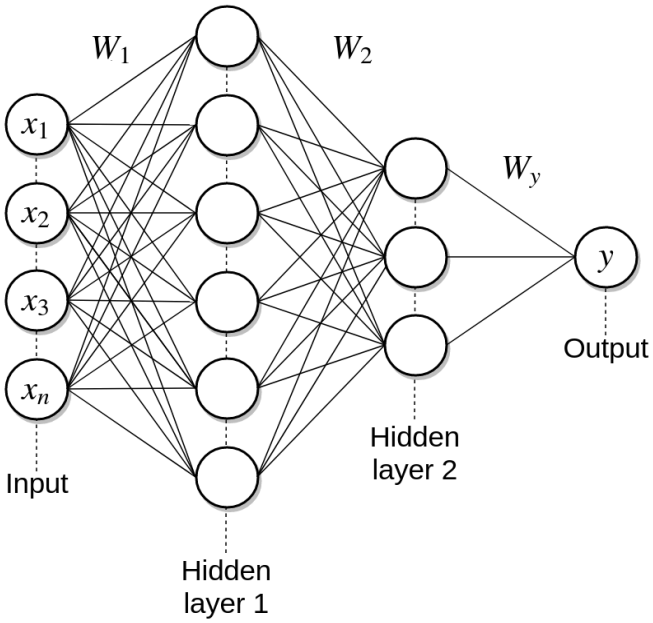
Keras là một thư viện hỗ trợ sẵn các thư viện và các hàm dùng để xây dựng và huấn luyện mạng neural học sâu.

Mạng neural học sâu sẽ gồm nhiều lớp, các lớp sẽ được kết nối với nhau để truyền thông tin. Để kết nối các lớp với nhau, Keras hỗ trợ thư viện Sequential để "stack" các lớp lại với nhau.

Chúng ta sẽ tiến hành xây dựng một mô hình mạng neural đơn giản gồm hai lớp. Trước tiên, chúng ta khởi tạo một Sequential model:

```
from keras.models import Sequential
model = Sequential()
```

Tiếp đến, chúng ta sẽ lần lượt thêm các lớp (layers) vào mô hình:
Fully-connected layer là hidden layer có mỗi node được kết nối với tất cả các node trong layer trước.



Minh họa mô hình mạng neural có Fully-connected layer

Với dữ liệu trước đó, mỗi điểm dữ liệu sẽ có số lượng thuộc tính là 784 tương ứng

với 784 điểm ảnh. Do đó, chúng ta sử dụng một lớp Fully-connected gồm 784 units như sau:

```
from keras.layers import Dense
Dense(784, input_shape=(784, ))
```

Trong đó, input_shape là kích thước (shape) của dữ liệu đầu vào.
Ghi chú: Lớp Dense có tham số activation để xác định hàm kích hoạt cần dùng cho mỗi lớp, có giá trị mặc định là None - tương ứng với hàm tuyến tính (Linear).

Để thêm một lớp vào mô hình, chúng ta sử dụng hàm add của Sequential:

```
model.add(Dense(784, input_shape=(784, )))
```

(?) Xác định kích thước đầu ra (output_shape) của lớp được thêm vào mô hình?

Lớp đầu ra của mô hình sẽ là xác suất ứng với mỗi nhãn (xem thêm về bài toán phân loại đối với MNIST). Do đó, lớp này sẽ có 10 units:

```
model.add(Dense(10, input_shape=(10, )))
```

(?) Tại sao số units ở lớp này lại là 10?

Để xem cấu trúc của mô hình mạng neural đã xây dựng, chúng ta sử dụng lệnh summary:

```
model.summary()
```

4. HUẤN LUYỆN MÔ HÌNH

Sử dụng hàm tối ưu hóa (optimizer): Adam

```
from tensorflow.keras.optimizers import Adam
optimizer = Adam(learning_rate=0.01)
```

(?) Vì sao cần set giá trị learning_rate nhỏ?

Hàm mất mát (loss): BinaryCrossEntropy

```
from tensorflow.keras.losses import BinaryCrossentropy
loss = BinaryCrossentropy()
```

Độ đo đánh giá (metrics): Accuracy
Compile model:

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

Huấn luyện mô hình với các thông số { batch_size=128, epoch=10 }:

```
model.fit(X_train_reshaped, y_train, batch_size=128, epochs=10)
```

Lưu ý: Bật GPU của Google Colab để thực hiện huấn luyện nhanh hơn.

5. ĐÁNH GIÁ MÔ HÌNH

Chúng ta tiến hành đánh giá mô hình đã xây dựng trên tập test.

Dự đoán dữ liệu mới (dữ liệu trong tập test):

```
y_pred = model.predict(X_test_reshaped)
```

(?) Lớp được chọn sẽ có xác suất cao nhất hay thấp nhất?

Để tìm nhãn có xác suất cao nhất, chúng ta sử dụng hàm argmax trong thư viện Numpy.

```
y_pred_label = np.argmax(y_pred, axis=-1)
```

Để tính độ chính xác dự đoán của mô hình, chúng ta sử dụng hàm accuracy_score trong thư viện Sklearn:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)*100
```

(?) Độ chính xác của mô hình là bao nhiêu?

6. LƯU MÔ HÌNH

Thư viện Keras hỗ trợ việc lưu lại mô hình ở định dạng file .keras.

Để lưu lại mô hình, chúng ta sử dụng hàm save.

```
model.save('my_model.keras')
```

(?) Việc lưu lại mô hình có tác dụng gì?

Để load mô hình đã có, chúng ta sử dụng hàm load_model.

```
from keras.models import load_model
model = load_model('my_model.keras')
```

7. MỘT SỐ THAO TÁC XỬ LÝ KHÁC VỚI MẠNG NEURAL

Trước hết, chúng ta tiến hành load và xem cấu trúc của mô hình mạng neural đơn giản được lưu trong file my_model.keras.

```
from keras.models import load_model
model_new = load_model('.../my_model.keras')
```

(?) Sử dụng lệnh summary để xem cấu trúc của mô hình và cho biết kết quả?

Để truy xuất tập các lớp của mô hình, chúng ta có thể sử dụng thuộc tính layers như sau:

```
model.layers
```

Để truy xuất lớp có chỉ số id trong mô hình (tính từ hidden layer về sau), chúng ta có thể sử dụng thuộc tính layers[id] hoặc hàm get_layer như sau:

```
model.layers[id]
```

```
model.get_layer(index=id)
```

Để xem bộ tham số của mô hình hoặc một lớp có chỉ số id trong mô hình, chúng ta có thể sử dụng thuộc tính weights hoặc hàm get_weights như sau:

- Xem tham số của mô hình:
  model.weights
  model.get_weights()
- Xem tham số của một lớp:
  model.layers[id].weights
  model.layers[id].get_weights()

Để xóa lớp cuối cùng của mô hình, chúng ta sử dụng hàm pop() như sau:

```
model.pop()
```

(?) Sử dụng lệnh summary để xem cấu trúc của mô hình và cho biết kết quả?

Để xóa cả mô hình, chúng ta sử dụng lệnh del như sau:

```
del model
```

8. BÀI TẬP

Bài tập 1

Xây dựng lại mô hình mạng neural, trong đó sử dụng hàm kích hoạt Sigmoid cho cả hai lớp, và xem kết quả.

Gợi ý: Thêm tham số activation='sigmoid' để xác định hàm kích hoạt Sigmoid khi khởi tạo lớp Dense.

```
Dense(784, input_shape=(784, ), activation='sigmoid')
```

Bài tập 2

Xây dựng lại mô hình mạng neural, trong đó sử dụng hàm kích hoạt ReLU cho lớp thứ nhất và Sigmoid cho lớp đầu ra, và xem kết quả.

Bài tập 3*

Để chuẩn hóa giá trị cho dữ liệu (normalizing data), chúng ta thực hiện như sau:

```
X_train_new = X_train/255.0
X_test_new = X_test/255.0
```

- Hãy thực hiện lại mô hình ở Bài tập 2 trên dữ liệu X_train và X_test đã được chuẩn hoá.
- So sánh độ chính xác của mô hình trong hai trường hợp: không chuẩn hóa và có chuẩn hóa ?

Bài tập 4*

In ra confusion matrix của mô hình ở Bài tập 3 và nêu nhận xét.

Gợi ý:

- Sử dụng hàm confusion_matrix để xác định ma trận nhầm lẫn:
  from sklearn.metrics import confusion_matrix
  cm = confusion_matrix(y_test, y_pred)
- Code in ma trận nhầm lẫn:
  df_cm = pd.DataFrame(cm, index=[i for i in range(0,10)], columns=[i for i in range(0,10)])
  plt.figure(figsize=(10,7))
  sn.heatmap(df_cm, annot=True, fmt='.5g')

Bài tập 5*

Tìm hiểu bộ dữ liệu small CIFAR10 và xây dựng mô hình mạng neural đơn giản gồm hai lớp cho bộ dữ liệu này.

Gợi ý: Load bộ dữ liệu small CIFAR10 như sau:

```
from keras.datasets.cifar10 import load_data
(X_train, y_train), (X_test, y_test) = load_data()
```