BÀI TẬP 1

---

1. Phân loại hoa iris (2 lớp):

1.1. Thêm các thư viện cần thiết:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from keras.api.models import Sequential
from tensorflow.keras.layers import Dense
```

Hình. Các thư viện cần thiết

- Thư viện **pandas** được sử dụng để đọc dữ liệu.
- Thư viện **numpy** được sử dụng để xử lý dữ liệu.
- Thư viện **matplotlib** được sử dụng để vẽ biểu đồ, trực quan hóa kết quả.
- Hàm **train_test_split** từ thư viện **scikit_learn** dùng để chia tập dữ liệu thành tập huấn luyện và tập kiểm tra.
- **Sequential** là một lớp trong **Keras** để tạo mô hình theo từng lớp.
- **Dense** là lớp kết nối đầy đủ trong mạng nơ-ron, một thành phần cơ bản và quan trọng của mô hình.

*Phiên bản sử dụng các thư viện:

Thư viện	Phiên bản
pandas	2.1.4
numpy	1.26.2
sklearn	0.0.post10
keras	3.3.3
tensorflow	2.16.1
matplotlib	3.8.0

Bảng. Phiên bản được sử dụng của các thư viện

1.2. Lấy dữ liệu và xử lý nhãn:

```
# Lấy dữ liệu
data = pd.read_csv( filepath_or_buffer: "iris.data", header=None).iloc[:100]
data.iloc[:, 4] = np.where(data.iloc[:, 4] == 'Iris-setosa', 0, 1)
print(data)
```

Hình. Lấy dữ liệu và xử lý cột nhãn

- Đọc dữ liệu bằng hàm **read_csv()** từ thư viện **pandas** để lấy dữ liệu từ file "iris.data", tham số "header" = None là để chỉ định rằng dữ liệu không có dòng tiêu đề. Sau đó, sử dụng **iloc[:100]** để giữ lại 100 hàng đầu tiên của dữ liệu sau khi đọc được.
- Dữ liệu sau khi đọc và giữ lại 100 hàng đầu tiên:

```
     0    1    2    3               4
0   5.1  3.5  1.4  0.2     Iris-setosa
1   4.9  3.0  1.4  0.2     Iris-setosa
2   4.7  3.2  1.3  0.2     Iris-setosa
3   4.6  3.1  1.5  0.2     Iris-setosa
4   5.0  3.6  1.4  0.2     Iris-setosa
..  ...  ...  ...  ...             ...
95  5.7  3.0  4.2  1.2  Iris-versicolor
96  5.7  2.9  4.2  1.3  Iris-versicolor
97  6.2  2.9  4.3  1.3  Iris-versicolor
98  5.1  2.5  3.0  1.1  Iris-versicolor
99  5.7  2.8  4.1  1.3  Iris-versicolor

[100 rows x 5 columns]
```

Hình. Dữ liệu sau khi đọc và giữ lại 100 hàng đầu tiên

- Sau đó, cột nhãn được chuyển thành các giá trị số để đưa vào mô hình, cụ thể: '0' cho "Iris-setosa" và '1' cho "Iris-versicolor". Tại cột nhãn, hàng **data.iloc[:, 4] = np.where(data.iloc[:, 4] == 'Iris-setosa', 0, 1)** nghĩa là: Xét từng mẫu dữ liệu, nếu tại đó đang có giá trị "Iris-setosa" thì sẽ được thay bằng 0, ngược lại, giá trị đó sẽ được thay bằng 1. Kết quả:

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
..	...	...	...	...	..
95	5.7	3.0	4.2	1.2	1
96	5.7	2.9	4.2	1.3	1
97	6.2	2.9	4.3	1.3	1
98	5.1	2.5	3.0	1.1	1
99	5.7	2.8	4.1	1.3	1

Hình. Dữ liệu sau khi xử lý cột nhãn

1.3. Trộn dữ liệu:

```
# Trộn dữ liệu
data = data.iloc[np.random.permutation(len(data))].reset_index(drop=True)
print(data)
```

Hình. Trộn dữ liệu

- Để tránh hiện tượng quá khớp (overfitting), cần trộn ngẫu nhiên các hàng trong dữ liệu. Theo đó, sử dụng hàm **np.random.permutation()** để trộn các hàng dữ liệu trong tập dữ liệu vừa được xử lí. Cụ thể:

+ len(data): Tính số lượng hàng trong ‘data’.

+ np.random.permutation(len(data)): Tạo ra một hoán vị ngẫu nhiên của các chỉ số từ ‘0’ đến ‘len(data) - 1’. Đây cũng là các chỉ số hàng trong ‘data’.

+ data.iloc[np.random.permutation(len(data))]: Sử dụng kết quả từ phía trên để sắp xếp lại các hàng của ‘data’ theo thứ tự ngẫu nhiên.

+ Hàm “reset_index(drop=True)”: Đặt lại chỉ số hàng sau khi hoán vị và loại bỏ chỉ số cũ.

+ Kết quả sau cùng được gán vào biến “data”.

Kết quả:

	0	1	2	3	4
0	6.0	2.9	4.5	1.5	1
1	6.4	2.9	4.3	1.3	1
2	5.8	2.6	4.0	1.2	1
3	4.8	3.0	1.4	0.3	0
4	5.4	3.7	1.5	0.2	0
5	5.7	2.6	3.5	1.0	1
6	5.6	3.0	4.1	1.3	1
7	5.7	2.8	4.5	1.3	1
8	4.8	3.1	1.6	0.2	0
9	5.6	2.7	4.2	1.3	1

Hình. Dữ liệu sau khi trộn (10 hàng đầu tiên)

1.4. Tách dữ liệu:

```
# Tách dữ liệu
X = data.iloc[:, :-1].values.astype('float32')
y = data.iloc[:, -1].values.astype('float32')

X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

Hình. Tách dữ liệu huấn luyện và kiểm tra

- Trước khi tách thành các tập huấn luyện và tập kiểm tra, cần tách các đặc trưng và các nhãn, cụ thể:

+ X = data[iloc:, :-1].values.astype(‘float32’): Chọn từ cột đầu tiên đến trước cột cuối cùng, tương ứng với các cột đặc trưng của mỗi mẫu trong tập dữ liệu. Sau đó, chuyển các mẫu dữ liệu trong dataframe thành một mảng. Tiếp theo, chuyển đổi các giá trị trong mảng sang kiểu dữ liệu ‘float32’, là kiểu dữ liệu số thực.

+ y = data[iloc:, -1].values.astype(‘float32’): Chọn cột cuối cùng của tập dữ liệu, tương ứng với cột nhãn của mỗi mẫu trong tập dữ liệu. Các bước sau được trình bày như trên.

- Sau khi đã tách các đặc trưng và nhãn, tiến hành chia dữ liệu thành các tập huấn luyện (80%) và tập kiểm tra (20%) bằng hàm **train_test_split**. Các tham số đầu vào như sau:

  + X, y lần lượt là các dữ liệu cần chia.

  + test_size = 0.2: Xác định kích thước của tập kiểm tra là 20% của toàn bộ dữ liệu, 80% còn lại đưa vào tập huấn luyện.

  + random_state = 42: Đây là một giá trị cố định được sử dụng để đảm bảo việc chia dữ liệu luôn cho ra cùng một kết quả khi chạy ở lần kế tiếp.

Kết quả:

```
X_train.shape: (80, 4)
X_test.shape: (20, 4)
y_train.shape: (80,)
y_test.shape: (20,)
```

Hình. Kích cỡ của mỗi tập huấn luyện và kiểm tra

1.5. Xây dựng mô hình:

```
# Xây dựng mô hình
model = Sequential()
model.add(Dense( units: 1, input_shape=(4,), activation='sigmoid'))

model.summary()
```

Hình. Xây dựng mô hình để huấn luyện

- **Sequential()**: Khởi tạo mô hình tuần tự.
- Dense(1, input_shape=(4,), activation='sigmoid')):
  + Một tầng (Dense) gồm 1 nơ-ron duy nhất (2 lớp).
  + Có 4 đặc trưng đầu vào.
  + Hàm kích hoạt "sigmoid" được sử dụng vì đây là bài toán phân loại nhị phân.

1.6. Biên dịch mô hình:

```
# Biên dịch mô hình
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Hình. Biên dịch mô hình trước khi huấn luyện

- Hàm mất mát được chọn là "binary_crossentropy" vì đây là bài toán phân loại nhị phân, khi đầu ra chỉ có 2 lớp.
- Thuật toán tối ưu "Adam" là thuật toán tối ưu dựa trên gradient, nó tự động điều chỉnh tốc độ học trong quá trình huấn luyện, giúp mô hình học nhanh hơn và hiệu quả hơn.
- Chỉ số đánh giá "accuracy" là chỉ số đánh giá phổ biến trong bài toán phân loại, được tính bằng tỷ lệ phần trăm số dự đoán đúng so với tổng số dự đoán.

1.7. Huấn luyện mô hình:

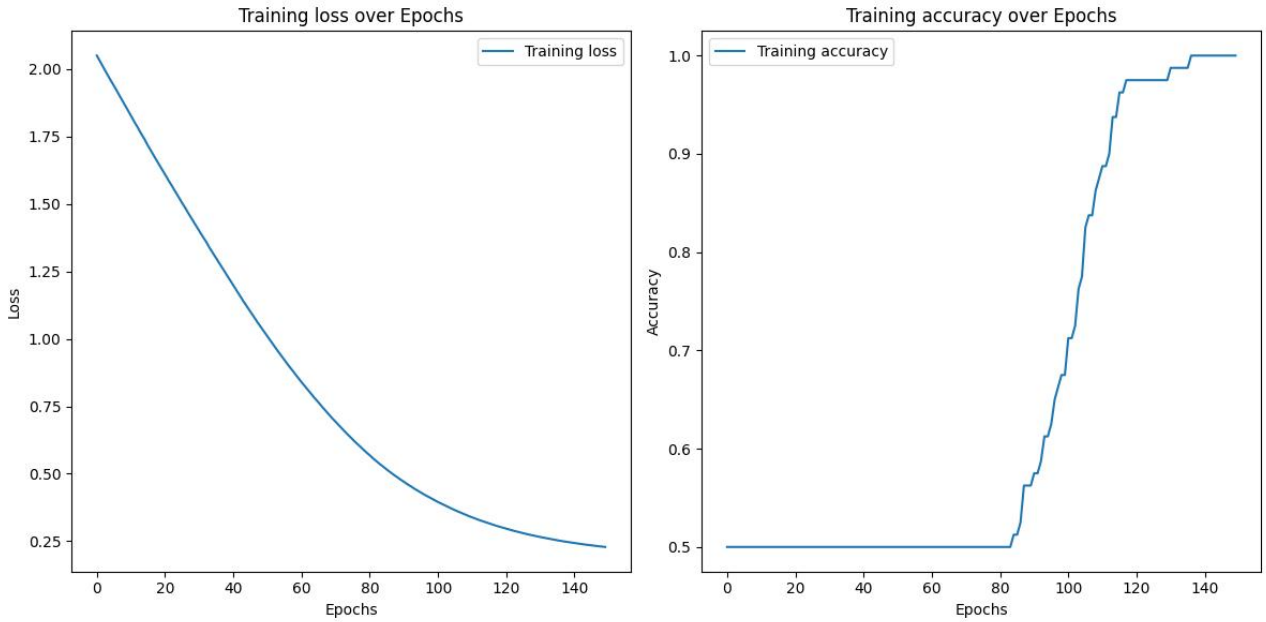
```
# Huấn luyện mô hình
model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=1)
```

Hình. Huấn luyện mô hình

Sử dụng hàm **fit()** để huấn luyện mô hình trên dữ liệu huấn luyện, qua đó, mô hình sẽ thực hiện một số lần lặp trên toàn bộ dữ liệu huấn luyện để học cách dự đoán chính xác hơn. Cụ thể:

- X_train, y_train là dữ liệu huấn luyện.
- epochs=150: Epoch là số lần mà toàn bộ tập dữ liệu huấn luyện được đưa vào mô hình. Trong trường hợp này, mô hình sẽ học 150 lần.
- batch_size=32: Batch_size là số lượng mẫu dữ liệu được sử dụng trong mỗi lần cập nhật trọng số. Trong bài này, mô hình sẽ cập nhật trọng số sau mỗi batch gồm 32 mẫu dữ liệu.
- verbose=1: Verbose thể hiện mức độ chi tiết của thông tin hiển thị trong quá trình huấn luyện. "verbose=1" sẽ hiển thị tiến trình huấn luyện sau mỗi epoch, gồm thông tin về mất mát và chỉ số đánh giá trên tập huấn luyện.

Kết quả:



Hình. Biểu đồ thể hiện độ mất mát (bên trái)
và độ chính xác (bên phải) sau khi huấn luyện

1.8. Đánh giá mô hình:

```
# Đánh giá mô hình
score = model.evaluate(X_test, y_test, verbose=0)
print(f"Test loss: {score[0]}")
print(f"Test accuracy: {score[1]}")
```

Hình. Đánh giá mô hình

Sử dụng hàm **evaluate()** để đánh giá mô hình trên tập dữ liệu kiểm tra. Kết quả đầu ra của hàm này là một danh sách có hai phần tử. Trong đó, phần tử đầu tiên là giá trị mất mát và phần tử thứ hai thể hiện độ chính xác trên tập kiểm tra. Kết quả:

```
Test loss: 0.2322521209716797
Test accuracy: 1.0
```

Hình. Kết quả đánh giá mô hình

2. Phân loại hoa iris (3 lớp):

2.1. Thêm các thư viện cần thiết:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from keras.api.models import Sequential
from tensorflow.keras.layers import Dense
```

Hình. Các thư viện cần thiết

- Thư viện **pandas** được sử dụng để đọc dữ liệu.
- Thư viện **numpy** được sử dụng để xử lý dữ liệu.
- Thư viện **matplotlib** được sử dụng để vẽ biểu đồ, trực quan hóa kết quả.
- Hàm **train_test_split** từ thư viện **scikit_learn** dùng để chia tập dữ liệu thành tập huấn luyện và tập kiểm tra.
- **Sequential** là một lớp trong **Keras** để tạo mô hình theo từng lớp.
- **Dense** là lớp kết nối đầy đủ trong mạng nơ-ron, một thành phần cơ bản và quan trọng của mô hình.

*Phiên bản sử dụng các thư viện:

Thư viện	Phiên bản
pandas	2.1.4
numpy	1.26.2
sklearn	0.0.post10
keras	3.3.3
tensorflow	2.16.1
matplotlib	3.8.0

Bảng. Phiên bản được sử dụng của các thư viện

2.2. Lấy dữ liệu và xử lí nhãn:

```
# Lấy dữ liệu
data = pd.read_csv( filepath_or_buffer: "iris.data", header=None)
data.iloc[:, -1] = np.where(data.iloc[:, -1] == 'Iris-setosa', 0,
                  np.where(data.iloc[:, -1] == 'Iris-versicolor', 1, 2))
print(data)
```

Hình. Lấy dữ liệu và xử lý nhãn

- Đọc dữ liệu bằng hàm **read_csv()** từ thư viện **pandas** để lấy dữ liệu từ file "iris.data", tham số "header" = None là để chỉ định rằng dữ liệu không có dòng tiêu đề. Dữ liệu sau khi đọc:

```
       0    1    2    3               4
0    5.1  3.5  1.4  0.2     Iris-setosa
1    4.9  3.0  1.4  0.2     Iris-setosa
2    4.7  3.2  1.3  0.2     Iris-setosa
3    4.6  3.1  1.5  0.2     Iris-setosa
4    5.0  3.6  1.4  0.2     Iris-setosa
..   ...  ...  ...  ...             ...
145  6.7  3.0  5.2  2.3  Iris-virginica
146  6.3  2.5  5.0  1.9  Iris-virginica
147  6.5  3.0  5.2  2.0  Iris-virginica
148  6.2  3.4  5.4  2.3  Iris-virginica
149  5.9  3.0  5.1  1.8  Iris-virginica

[150 rows x 5 columns]
```

Hình. Dữ liệu sau khi đọc

- Sau đó, cột nhãn được chuyển thành các giá trị số để đưa vào mô hình, cụ thể: '0' cho "Iris-setosa", '1' cho "Iris-versicolor" và '2' cho "Iris-virginica". Tại cột nhãn, hàng **data.iloc[:, -1] = np.where(data.iloc[:, -1] == 'Iris-setosa', 0, np.where(data.iloc[:, -1] == 'Iris-versicolor', 1, 2))** nghĩa là: Xét từng mẫu dữ liệu, nếu tại đó đang có giá trị "Iris-setosa" thì sẽ được thay bằng 0, ngược lại, tiếp tục xét mẫu dữ liệu đó, nếu nó đang mang giá trị "Iris-versicolor" thì sẽ được thay bằng 1, còn lại được thay bằng 2. Kết quả:

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
..	...	...	...	...	..
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

Hình. Dữ liệu sau khi xử lý cột nhãn

2.3. Trộn dữ liệu:

```
# Trộn dữ liệu
data = data.iloc[np.random.permutation(len(data))].reset_index(drop=True)
print(data)
```

Hình. Trộn dữ liệu

- Để tránh hiện tượng quá khớp (overfitting), cần trộn ngẫu nhiên các hàng trong dữ liệu. Theo đó, sử dụng hàm **np.random.permutation()** để trộn các hàng dữ liệu trong tập dữ liệu vừa được xử lí. Cụ thể:

+ len(data): Tính số lượng hàng trong ‘data’.

+ np.random.permutation(len(data)): Tạo ra một hoán vị ngẫu nhiên của các chỉ số từ ‘0’ đến ‘len(data) - 1’. Đây cũng là các chỉ số hàng trong ‘data’.

+ data.iloc[np.random.permutation(len(data))]: Sử dụng kết quả từ phía trên để sắp xếp lại các hàng của ‘data’ theo thứ tự ngẫu nhiên.

+ Hàm “reset_index(drop=True)”: Đặt lại chỉ số hàng sau khi hoán vị và loại bỏ chỉ số cũ.

+ Kết quả sau cùng được gán vào biến “data”.

Kết quả:

	0	1	2	3	4
0	5.4	3.4	1.7	0.2	0
1	5.8	2.7	5.1	1.9	2
2	6.7	3.3	5.7	2.5	2
3	4.9	3.0	1.4	0.2	0
4	5.1	3.8	1.9	0.4	0
..	...	...	...	...	..
145	5.3	3.7	1.5	0.2	0
146	6.3	3.3	4.7	1.6	1
147	7.7	2.6	6.9	2.3	2
148	5.0	3.6	1.4	0.2	0
149	5.7	2.8	4.1	1.3	1

Hình. Dữ liệu sau khi trộn

2.4. Tách dữ liệu:

```
# Tách dữ liệu
X = data.iloc[:, :-1].values.astype('float32')
y = data.iloc[:, -1].values.astype('float32')

X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)
print(f"X_train.shape: {X_train.shape}")
print(f"X_test.shape: {X_test.shape}")
print(f"y_train.shape: {y_train.shape}")
print(f"y_test.shape: {y_test.shape}")
```

Hình. Tách dữ liệu huấn luyện và kiểm tra

- Trước khi tách thành các tập huấn luyện và tập kiểm tra, cần tách các đặc trưng và các nhãn, cụ thể:

+ X = data[iloc:, :-1].values.astype(‘float32’): Chọn từ cột đầu tiên đến trước cột cuối cùng, tương ứng với các cột đặc trưng của mỗi mẫu trong tập dữ liệu. Sau đó, chuyển các mẫu dữ liệu trong dataframe thành một mảng. Tiếp theo, chuyển đổi các giá trị trong mảng sang kiểu dữ liệu ‘float32’, là kiểu dữ liệu số thực.

+ y = data[iloc:, -1].values.astype(‘float32’): Chọn cột cuối cùng của tập dữ liệu, tương ứng với cột nhãn của mỗi mẫu trong tập dữ liệu. Các bước sau được trình bày như trên.

- Sau khi đã tách các đặc trưng và nhãn, tiến hành chia dữ liệu thành các tập huấn luyện (80%) và tập kiểm tra (20%) bằng hàm **train_test_split**. Các tham số đầu vào như sau:

+ X, y lần lượt là các dữ liệu cần chia.

+ test_size = 0.2: Xác định kích thước của tập kiểm tra là 20% của toàn bộ dữ liệu, 80% còn lại đưa vào tập huấn luyện.

+ random_state = 42: Đây là một giá trị cố định được sử dụng để đảm bảo việc chia dữ liệu luôn cho ra cùng một kết quả khi chạy ở lần kế tiếp.

Kết quả:

```
X_train.shape: (120, 4)
X_test.shape: (30, 4)
y_train.shape: (120,)
y_test.shape: (30,)
```

Hình. Kích cỡ của mỗi tập huấn luyện và kiểm tra

2.5. Xây dựng mô hình:

```
# Xây dựng mô hình
model = Sequential()
model.add(Dense( units: 3, input_shape=(4,), activation='softmax'))
```

Hình. Xây dựng mô hình để huấn luyện

- **Sequential()**: Khởi tạo mô hình tuần tự.
- Dense(3, input_shape=(4,), activation='softmax')):

+ Một tầng (Dense) gồm 3 nơ-ron, mỗi nơ-ron sẽ tính toán một đầu ra khác nhau.

+ Có 4 đặc trưng đầu vào.

+ Hàm kích hoạt "softmax" được sử dụng vì đây là bài toán phân loại đa lớp. Hàm này sẽ biến đổi đầu ra của 3 nơ-ron thành các xác suất của từng lớp, đảm bảo tổng các xác suất bằng 1.

2.6. Biên dịch mô hình:

```
# Biên dịch mô hình
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Hình. Biên dịch mô hình trước khi huấn luyện

- Hàm mất mát được chọn là “sparse_categorical_crossentropy” vì đây là bài toán phân loại đa lớp, trong đó nhãn được biểu diễn dưới dạng số nguyên.
- Thuật toán tối ưu “Adam” là thuật toán tối ưu dựa trên gradient, nó tự động điều chỉnh tốc độ học trong quá trình huấn luyện, giúp mô hình học nhanh hơn và hiệu quả hơn.
- Chỉ số đánh giá “accuracy” là chỉ số đánh giá phổ biến trong bài toán phân loại, được tính bằng tỷ lệ phần trăm số dự đoán đúng so với tổng số dự đoán.

2.7. Huấn luyện mô hình:

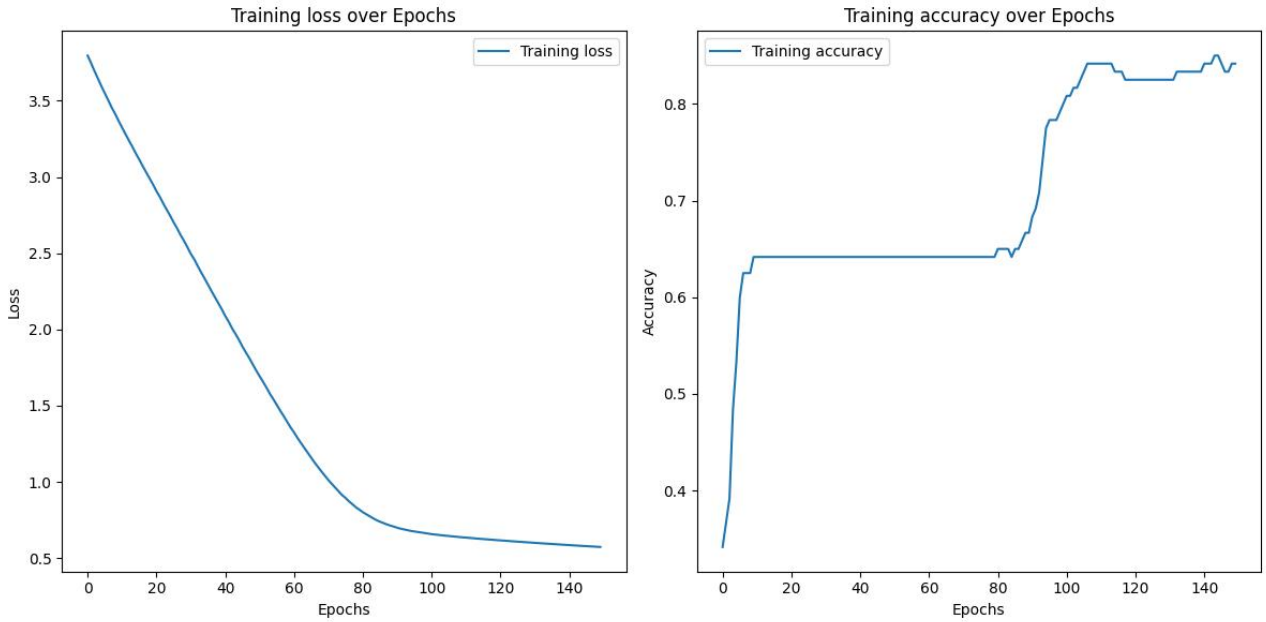
```
# Huấn luyện mô hình
model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=1)
```

Hình. Huấn luyện mô hình

Sử dụng hàm **fit()** để huấn luyện mô hình trên dữ liệu huấn luyện, qua đó, mô hình sẽ thực hiện một số lần lặp trên toàn bộ dữ liệu huấn luyện để học cách dự đoán chính xác hơn. Cụ thể:

- X_train, y_train là dữ liệu huấn luyện.
- epochs=150: Epoch là số lần mà toàn bộ tập dữ liệu huấn luyện được đưa vào mô hình. Trong trường hợp này, mô hình sẽ học 150 lần.
- batch_size=32: Batch_size là số lượng mẫu dữ liệu được sử dụng trong mỗi lần cập nhật trọng số. Trong bài này, mô hình sẽ cập nhật trọng số sau mỗi batch gồm 32 mẫu dữ liệu.
- verbose=1: Verbose thể hiện mức độ chi tiết của thông tin hiển thị trong quá trình huấn luyện. “verbose=1” sẽ hiển thị tiến trình huấn luyện sau mỗi epoch, gồm thông tin về mất mát và chỉ số đánh giá trên tập huấn luyện.

Kết quả:



Hình. Biểu đồ thể hiện độ mất mát (bên trái)
và độ chính xác (bên phải) sau khi huấn luyện

2.8. Đánh giá mô hình

```
# Đánh giá mô hình
score = model.evaluate(X_test, y_test, verbose=0)
print(f"Test loss: {score[0]}")
print(f"Test accuracy: {score[1]}")
```

Hình. Đánh giá mô hình

Sử dụng hàm **evaluate()** để đánh giá mô hình trên tập dữ liệu kiểm tra. Kết quả đầu ra của hàm này là một danh sách có hai phần tử. Trong đó, phần tử đầu tiên là giá trị mất mát và phần tử thứ hai thể hiện độ chính xác trên tập kiểm tra. Kết quả:

```
Test loss: 0.4972517788410187
Test accuracy: 0.9333333373069763
```

Hình. Kết quả đánh giá mô hình

3. Code đầy đủ:

*Phân loại hoa iris (2 lớp):

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from keras.api.models import Sequential
from tensorflow.keras.layers import Dense

# Lấy dữ liệu và xử lí
data = pd.read_csv( filepath_or_buffer: "iris.data", header=None).iloc[:100]
# print(data)

data.iloc[:, 4] = np.where(data.iloc[:, 4] == 'Iris-setosa', 0, 1)
# print(data)

# Trộn dữ liệu
data = data.iloc[np.random.permutation(len(data))].reset_index(drop=True)
print(data)

# Tách dữ liệu
X = data.iloc[:, :-1].values.astype('float32')
y = data.iloc[:, -1].values.astype('float32')

X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=None)
print(f"X_train.shape: {X_train.shape}")
print(f"X_test.shape: {X_test.shape}")
print(f"y_train.shape: {y_train.shape}")
print(f"y_test.shape: {y_test.shape}")

# Xây dựng mô hình
model = Sequential()
model.add(Dense( units: 1, input_shape=(4,), activation='sigmoid'))

model.summary()
```

```
# Biên dịch mô hình
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Huấn luyện mô hình
epochs = 150
history = model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=1)

plt.figure(figsize=(12, 6))

plt.subplot( *args: 1, 2, 1)
plt.plot( *args: range(epochs), history.history['loss'], label='Training loss')
plt.title("Training loss over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

plt.subplot( *args: 1, 2, 2)
plt.plot( *args: range(epochs), history.history['accuracy'], label='Training accuracy')
plt.title("Training accuracy over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()

plt.tight_layout()
plt.show()


# Đánh giá mô hình
score = model.evaluate(X_test, y_test, verbose=0)
print(f"Test loss: {score[0]}")
print(f"Test accuracy: {score[1]}")
```

Hình. Code đầy đủ phân loại iris 2 lớp

*Phân loại hoa iris (3 lớp):

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from keras.api.models import Sequential
from tensorflow.keras.layers import Dense

# Lấy dữ liệu và xử lí
data = pd.read_csv( filepath_or_buffer: "iris.data", header=None)
# print(data)

data.iloc[:, -1] = np.where(data.iloc[:, -1] == 'Iris-setosa', 0,
                  np.where(data.iloc[:, -1] == 'Iris-versicolor', 1, 2))
# print(data)

# Trộn dữ liệu
data = data.iloc[np.random.permutation(len(data))].reset_index(drop=True)
print(data)

# Tách dữ liệu
X = data.iloc[:, :-1].values.astype('float32')
y = data.iloc[:, -1].values.astype('float32')

X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)
print(f"X_train.shape: {X_train.shape}")
print(f"X_test.shape: {X_test.shape}")
print(f"y_train.shape: {y_train.shape}")
print(f"y_test.shape: {y_test.shape}")

# Xây dựng mô hình
model = Sequential()
model.add(Dense( units: 3, input_shape=(4,), activation='softmax'))
```

```
model.summary()

# Biên dịch mô hình
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Huấn luyện mô hình
epochs = 150
history = model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=1)

plt.figure(figsize=(12, 6))

plt.subplot( *args: 1, 2, 1)
plt.plot( *args: range(epochs), history.history['loss'], label='Training loss')
plt.title("Training loss over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

plt.subplot( *args: 1, 2, 2)
plt.plot( *args: range(epochs), history.history['accuracy'], label='Training accuracy')
plt.title("Training accuracy over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()

plt.tight_layout()
plt.show()

# Đánh giá mô hình
score = model.evaluate(X_test, y_test, verbose=0)
print(f"Test loss: {score[0]}")
print(f"Test accuracy: {score[1]}")
```

Hình. Code đầy đủ phân loại iris 3 lớp

-- HẾT --