Họ và tên: Quách Xuân Phúc

MSV: B20DCCN513

4.1. Trình bày hiểu biết của mình về tensorflow và 5 ví dụ minh họa (không có trong Bài tập 3).

* Khái niệm:

TensorFlow là một thư viện phần mềm mã nguồn mở được phát triển bởi Google, đặc biệt được sử dụng rộng rãi trong lĩnh vực học sâu (deep learning) và máy học (machine learning). Nó cung cấp một nền tảng mạnh mẽ để xây dựng và huấn luyện các mô hình máy học, đặc biệt là mạng thần kinh sâu (neural networks). TensorFlow cho phép xây dựng và huấn luyện các mô hình phức tạp thông qua việc tạo và quản lý các đồ thị tính toán.

Đặc trưng của TensorFlow là khả năng xây dựng các mô hình sử dụng các đồ thị dữ liệu (data flow graphs), trong đó các nút trong đồ thị đại diện cho các phép toán toán học và các cạnh đại diện cho dữ liệu (tensors) được truyền đi giữa các nút. TensorFlow tối ưu hóa việc tính toán trên dữ liệu này, đặc biệt là khi dữ liệu có kích thước lớn và yêu cầu tính toán song song trên nhiều thiết bị như CPU và GPU.

Được thiết kế để linh hoạt và dễ sử dụng, TensorFlow cung cấp nhiều API khác nhau cho các mức độ và mục tiêu sử dụng khác nhau. Người dùng có thể sử dụng TensorFlow để xây dựng các mô hình từ đơn giản đến phức tạp, từ ứng dụng học máy đơn lẻ đến các hệ thống thông minh phức tạp.

* Các API của TensorFlow:

TensorFlow cung cấp nhiều API (Giao diện lập trình ứng dụng). Chúng có thể được phân loại thành 2 nhóm chính:

- API cấp thấp:
  - Điều khiển lập trình toàn bộ
  - Được khuyến nghị cho các nhà nghiên cứu học máy
  - Cung cấp mức kiểm soát tốt đối với các mô hình
  - TensorFlow Core là API cấp thấp của TensorFlow.
- API cấp cao:
  - Xây dựng trên nền TensorFlow Core
  - Dễ học và sử dụng hơn so với TensorFlow Core
  - Làm cho các công việc lặp lại dễ dàng hơn và nhất quán hơn giữa các người dùng khác nhau
  - "tf.contrib.learn" là một ví dụ về API cấp cao.

Một tensor có thể được hiểu đơn giản là một mảng đa chiều. Cụ thể, nó có thể là một scalar (mảng 0 chiều), vector (mảng 1 chiều), ma trận (mảng 2 chiều), hoặc một tensor với số chiều lớn hơn. Chẳng hạn:

- Scalar (0 chiều): Một số duy nhất.
- Vector (1 chiều): Một mảng các số.
- Ma trận (2 chiều): Một bảng số.
- Tensor (nhiều chiều): Một mảng nhiều chiều.

Đây là ví dụ về cách bạn có thể tạo tensors trong TensorFlow sử dụng Python và TensorFlow API:

```python
import tensorflow as tf

# Tạo một scalar (0 chiều)
scalar_tensor = tf.constant(5)  # Đây là một tensor 0 chiều

# Tạo một vector (1 chiều)
vector_tensor = tf.constant([1, 2, 3])  # Đây là một tensor 1 chiều

# Tạo một ma trận (2 chiều)
matrix_tensor = tf.constant([[1, 2], [3, 4]])  # Đây là một tensor 2 chiều

# Tạo một tensor 3 chiều
tensor_3d = tf.constant([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])  # Đây là một tensor 3 chiều
```

Trong TensorFlow, tf.constant là một hàm được sử dụng để tạo một Tensor có giá trị không thay đổi (constant), nghĩa là giá trị của Tensor này không thể được thay đổi sau khi đã được tạo. Đây là một cách phổ biến để đưa dữ liệu vào mô hình hoặc định nghĩa các hằng số trong đồ thị tính toán.

Dưới đây là một ví dụ về cách sử dụng tf.constant:

```python
import tensorflow as tf

# Tạo một Tensor constant với giá trị là 10 và kiểu dữ liệu là int32
constant_tensor = tf.constant(10, dtype=tf.int32)

# Khởi tạo một phiên TensorFlow
with tf.Session() as sess:
    # Chạy đồ thị tính toán để lấy giá trị của Tensor constant
    constant_value = sess.run(constant_tensor)
    print("Giá trị của Tensor constant:", constant_value)  # Kết quả: 10
```

Ở đây, chúng ta tạo một Tensor constant với giá trị là 10 và kiểu dữ liệu là int32 bằng cách sử dụng tf.constant. Sau đó, chúng ta chạy đồ thị tính toán để lấy giá trị của Tensor constant bằng cách sử dụng sess.run.

Lưu ý rằng giá trị của Tensor constant không thể thay đổi sau khi đã được tạo, nó chỉ mang giá trị mà bạn đã gán ban đầu.

Trong TensorFlow, một Operation (hoặc Op) đại diện cho một phép toán hoặc một hành động được thực hiện trên dữ liệu để tạo ra một kết quả. Các Operation là các nút trong đồ thị tính toán của TensorFlow, và chúng thực hiện các phép toán số học, logic, ma trận, hoặc bất kỳ phép toán nào khác cần thiết trong quá trình tính toán.

Một số thông tin quan trọng về Operations trong TensorFlow:

- Các Loại Operation: TensorFlow cung cấp nhiều loại Operation khác nhau, bao gồm phép toán số học (cộng, trừ, nhân, chia), phép toán logic (AND, OR, NOT), phép toán ma trận, activation functions, loss functions, optimizer operations, và nhiều loại khác.
- Phép Toán (Operation) trong Đồ Thị Tính Toán: Mỗi Operation là một nút trong đồ thị tính toán của TensorFlow. Đầu vào của một Operation là các Tensor đại diện cho dữ liệu vào, và đầu ra là một Tensor mới chứa kết quả của phép toán.
- Khởi tạo và Thực Thi Operation: Để thực thi một Operation và lấy kết quả, bạn cần tạo một phiên TensorFlow và sử dụng phương thức run. Khi chạy một Operation, nó có thể cần dữ liệu đầu vào từ các Tensor khác trong đồ thị, và TensorFlow tự động xác định thứ tự thực thi dựa trên các phụ thuộc.

Dưới đây là một ví dụ về cách tạo và sử dụng một Operation trong TensorFlow:

```
import tensorflow as tf

# Tạo các Tensor đầu vào
a = tf.constant(5)
b = tf.constant(3)

# Tạo một Operation thực hiện phép cộng
addition = tf.add(a, b)

# Khởi tạo một phiên TensorFlow
with tf.Session() as sess:
    # Chạy đồ thị tính toán để lấy kết quả của Operation
    result = sess.run(addition)
    print("Kết quả của phép cộng:", result)  # Kết quả: 8
```

Trong ví dụ này, chúng ta tạo hai Tensor constant a và b, sau đó tạo một Operation (addition) để thực hiện phép cộng giữa a và b. Khi chúng ta chạy đồ thị tính toán và thực thi Operation, kết quả của phép cộng được in ra.

Khi huấn luyện một mô hình trong TensorFlow, ta sử dụng các biến (variables) để lưu trữ và cập nhật các tham số của mô hình. Các biến là các bộ đệm trong bộ nhớ (in-memory buffers) chứa các tensors, và chúng được thiết kế để chứa các giá trị có thể thay đổi trong quá trình huấn luyện.

Trước đó, chúng ta đã làm việc với các tensors constant (hằng số) trong TensorFlow, chúng là các tensors có giá trị không thay đổi sau khi được khởi tạo. Các tensors constant được sử dụng để đại diện cho dữ liệu không thay đổi hoặc các giá trị mà chúng ta không muốn thay đổi trong quá trình tính toán.

Trong khi đó, biến (variables) trong TensorFlow được sử dụng để đại diện cho các tham số mô hình như trọng số (weights) và bias. Các giá trị của các biến có thể thay đổi theo thời gian, đặc biệt là trong quá trình huấn luyện mô hình, khi chúng được cập nhật để tối ưu hóa hiệu suất của mô hình.

Tóm lại, biến (variables) là một khái niệm quan trọng trong TensorFlow, cho phép lưu trữ và cập nhật các tham số mô hình trong quá trình huấn luyện, khác với các tensors constant mà chúng ta đã làm việc trước đó.

```python
import tensorflow as tf

# Khởi tạo một biến với giá trị ban đầu là 0 và kiểu dữ liệu là float32
my_variable = tf.Variable(0.0, dtype=tf.float32)

# Tạo một phép cộng để tăng giá trị của biến lên 1
add_operation = tf.add(my_variable, 1)

# Tạo một phép gán để cập nhật giá trị của biến
update_operation = tf.assign(my_variable, add_operation)

# Khởi tạo biến trước khi sử dụng
init = tf.global_variables_initializer()

# Bắt đầu một phiên tính toán TensorFlow
with tf.Session() as sess:
    # Khởi tạo biến
    sess.run(init)

    # Hiển thị giá trị ban đầu của biến
    print("Giá trị ban đầu của biến: {}".format(sess.run(my_variable)))

    # Thực hiện 5 lần cập nhật biến
    for _ in range(5):
        sess.run(update_operation)
        print("Giá trị biến sau {} lần cập nhật: {}".format(_, sess.run(my_variable)))
```

Placeholder (Nơi chứa)

Một đồ thị có thể được tham số hóa để chấp nhận đầu vào từ bên ngoài, được gọi là placeholders. Một placeholder là một lời hứa để cung cấp một giá trị sau này. Trong quá trình đánh giá đồ thị mà liên quan đến các nút placeholder, một tham số feed_dict được truyền vào phương thức run của phiên để xác định các Tensor cung cấp giá trị cụ thể cho những nút placeholder này. Xem xét ví dụ dưới đây:

```python
import tensorflow as tf

# Tạo một placeholder với kiểu dữ liệu float32 và shape None (kích thước có thể thay đổi)
input_placeholder = tf.placeholder(tf.float32, shape=(None,))

# Tạo một phép toán sử dụng placeholder
output = input_placeholder * 2

# Tạo một phiên TensorFlow
with tf.Session() as sess:
    # Tạo một feed dictionary để cung cấp dữ liệu đầu vào cho placeholder
    input_data = np.array([1.0, 2.0, 3.0])
    feed_dict = {input_placeholder: input_data}

    # Chạy đồ thị tính toán với dữ liệu được cung cấp
    result = sess.run(output, feed_dict=feed_dict)
    print("Kết quả:", result)  # Kết quả: [2.0, 4.0, 6.0]
```

Trong ví dụ này:

Chúng ta định nghĩa một placeholder với kiểu dữ liệu float32 và shape None, cho phép đầu vào có thể thay đổi kích thước theo mong muốn.

Sau đó, chúng ta định nghĩa một phép toán sử dụng placeholder, ở đây là nhân đầu vào với

Trong phiên TensorFlow, chúng ta tạo một feed dictionary để cung cấp dữ liệu đầu vào cho placeholder.

Khi chúng ta chạy đồ thị tính toán, chúng ta truyền feed dictionary để cung cấp giá trị cho placeholder và tính toán output.

Placeholder cho phép ta linh hoạt đưa dữ liệu vào trong quá trình tính toán và rất hữu ích khi xây dựng mô hình học máy.

4.2

importing the dependencies: import các thư viện cần thiết (dependencies)

import tensorflow.compat.v1 as tf

import numpy as np

import matplotlib.pyplot as plt

TensorFlow (tf): Thư viện học sâu mạnh mẽ được sử dụng để xây dựng, huấn luyện và triển khai các mô hình học máy. Import thư viện TensorFlow. tensorflow.compat.v1 cho biết đây là phiên bản tương thích (compatibility version 1.x) của TensorFlow. Tên ngắn gọn tf là tên gọi được sử dụng trong mã để gọi các hàm và lớp từ thư viện TensorFlow.

NumPy (np): Thư viện cực kỳ quan trọng trong Python cho tính toán khoa học và toán học trên mảng nhiều chiều.

Matplotlib.pyplot (plt): Thư viện được sử dụng để vẽ đồ thị và biểu đồ.

Model Parameters: Khai báo thông số mô hình:

learning_rate = 0.01

training_epochs = 2000

display_step = 200

Tốc độ học (learning_rate): Đây là một siêu tham số quan trọng trong quá trình huấn luyện. Nó quy định tốc độ mà mô hình sẽ cập nhật các trọng số của nó.

Số lượng epochs (training_epochs): Số lượng lần mô hình sẽ duyệt qua toàn bộ dữ liệu huấn luyện.

Bước hiển thị thông tin (display_step): Số lượng epochs mà sau đó thông tin về quá trình huấn luyện sẽ được hiển thị.

Training Data: Dữ liệu huấn luyện và kiểm tra:

train_X = np.asarray([3.3,4.4,5.5,...])

train_y = np.asarray([1.7,2.76,2.09,...])

test_X = np.asarray([6.83, 4.668, 8.9,...])

test_y = np.asarray([1.84, 2.273, 3.2,...])

Dữ liệu huấn luyện (train_X và train_y) và dữ liệu kiểm tra (test_X và test_y): Đây là các tập dữ liệu mà mô hình sẽ được huấn luyện và kiểm tra sau khi huấn luyện xong.

Test Data: Dòng này khai báo dữ liệu kiểm tra (test_X). Đây là các đặc trưng đầu vào được sử dụng để kiểm tra mô hình sau khi đã được huấn luyện.

Set placeholders for feature and target vectors: Khai báo placeholders và biến mô hình:

X = tf.placeholder(tf.float32)

y = tf.placeholder(tf.float32)

W = tf.Variable(np.random.randn(), name="weight")

b = tf.Variable(np.random.randn(), name="bias")

Placeholders (X và y): Đối tượng trong TensorFlow để cung cấp dữ liệu đầu vào và đầu ra khi huấn luyện và kiểm tra mô hình.

Biến (W và b): Các tham số mô hình sẽ được tối ưu hoá trong quá trình huấn luyện.

Set model weights and bias: Tạo biến (W và b) để lưu trữ trọng số và sai số (bias) của mô hình.

Construct a linear model: xây dựng mô hình tuyến tính (linear_model) dựa trên đặc trưng đầu vào và các trọng số.

linear_model = W*X + b

Mô hình tuyến tính (linear_model): Đây là mô hình đơn giản với một đầu vào (X), một trọng số (W), và một sai số (b).

Mean squared error: Hàm mất mát

cost = tf.reduce_sum(tf.square(linear_model - y)) / (2*n_samples)

Mean Squared Error (cost): Hàm mất mát sẽ đánh giá sự khác biệt giữa đầu ra dự đoán và đầu ra thực tế, và mục tiêu là giảm hàm mất mát này.

Gradient descent:Tối ưu hóa.

optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

Optimizer (optimizer): Sử dụng thuật toán Gradient Descent để tối ưu hóa hàm mất mát và cập nhật các tham số mô hình.

Initializing the variables: Khởi tạo biến mô hình trước khi bắt đầu quá trình huấn luyện.

init = tf.global_variables_initializer()

Khởi tạo biến (init): Đây là bước quan trọng để khởi tạo tất cả các biến mô hình.

Launch the graph with tf.Session() as sess: Bắt đầu phiên TensorFlow và huấn luyện mô hình:

with tf.Session() as sess:

Phiên TensorFlow (sess): Đây là phiên làm việc với TensorFlow, nơi mà ta thực hiện các phép tính và huấn luyện mô hình.

Fit all training data: Dòng này mô tả việc huấn luyện mô hình trên dữ liệu huấn luyện (train_X và train_y).

perform gradient descent step: Huấn luyện mô hình

sess.run(optimizer, feed_dict={X: train_X, y: train_y})

Huấn luyện mô hình (sess.run(optimizer, ...)) : Chạy quá trình huấn luyện mô hình bằng cách thực hiện một bước tối ưu hóa trên dữ liệu huấn luyện.

Display logs per epoch step: Hhiển thị thông tin sau mỗi epoch để theo dõi tiến trình của quá trình huấn luyện.

if (epoch+1) % display_step == 0:

    c = sess.run(cost, feed_dict={X: train_X, y: train_y})

    print("Epoch:{0:6} \t Cost:{1:10.4} \t W:{2:6.4} \t b:{3:6.4}".format(epoch+1, c, sess.run(W), sess.run(b)))

In ra thông tin sau mỗi epoch: Sau mỗi số epoch được chỉ định bởi display_step, in ra mất mát và các thông số mô hình.

Print final parameter values: in ra giá trị cuối cùng của trọng số và sai số sau khi huấn luyện kết thúc.

print("Optimization Finished!")

training_cost = sess.run(cost, feed_dict={X: train_X, y: train_y})

print("Final training cost:", training_cost, "W:", sess.run(W), "b:", sess.run(b), '\n')

Graphic display: Dòng này mô tả việc hiển thị biểu đồ để trực quan hóa dữ liệu huấn luyện và mô hình đã học.

plt.plot(train_X, train_y, 'ro', label='Original data')

plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')

plt.legend()

plt.show()

Hiển thị biểu đồ: Vẽ biểu đồ để so sánh dữ liệu huấn luyện và đường tuyến tính mà mô hình đã học.

Testing the model: kiểm tra mô hình trên dữ liệu kiểm tra (test_X).

testing_cost = sess.run(tf.reduce_sum(tf.square(linear_model - y)) / (2 * test_X.shape[0]), feed_dict={X: test_X, y: test_y})

print("Final testing cost:", testing_cost)

print("Absolute mean square loss difference:", abs(training_cost - testing_cost))

Kiểm tra mô hình trên dữ liệu kiểm tra (testing_cost) : Đánh giá hiệu suất của mô hình bằng cách tính toán mất mát trên dữ liệu kiểm tra.

In ra kết quả kiểm tra: In ra mất mát trên dữ liệu kiểm tra và sự khác biệt giữa mất mát trên dữ liệu huấn luyện và kiểm tra.

Display fitted line on test data: Hiển thị đường tuyến tính đã học được trên dữ liệu kiểm tra để đánh giá mô hình.

plt.plot(test_X, test_y, 'bo', label='Testing data')

plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')

plt.legend()

plt.show()

Hiển thị đường tuyến tính (plt.plot(...)) trên dữ liệu kiểm tra: Đây là bước để đánh giá mô hình bằng cách so sánh đầu ra dự đoán trên dữ liệu kiểm tra với đường tuyến tính đã học được.

4.3

Keras là một giao diện lập trình ứng dụng (API) dành cho Python được sử dụng để xây dựng và đào tạo các mô hình học máy và học sâu một cách dễ dàng và hiệu quả. Keras được phát triển bởi François Chollet và ban đầu là một dự án độc lập, sau đó đã được tích hợp chặt chẽ vào TensorFlow, một thư viện học máy và học sâu phổ biến.

Keras giúp các nhà phát triển xây dựng mô hình neural network một cách nhanh chóng và dễ dàng thông qua các lớp và hàm API trực quan. Nó được thiết kế để làm cho việc định nghĩa, đào tạo, và đánh giá các mô hình học máy trở nên đơn giản hơn, giúp tiết kiệm thời gian và công sức cho các dự án học máy và học sâu.

Một số đặc điểm quan trọng của Keras bao gồm:

- Dễ sử dụng: Keras cung cấp cú pháp đơn giản và dễ hiểu cho việc xây dựng mô hình neural network, điều này làm giảm ngưỡng cho người mới học về học máy.
- Tích hợp: Keras được tích hợp chặt chẽ với TensorFlow, cho phép sử dụng toàn bộ khả năng của TensorFlow trong các mô hình Keras.
- Hỗ trợ nhiều backend: Keras hỗ trợ nhiều backend như TensorFlow, Theano, và CNTK, cho phép bạn lựa chọn backend tùy theo nhu cầu của dự án.
- Học chuyển tiếp: Keras cung cấp sẵn các mô hình được đào tạo trước (pre-trained models) cho nhiều nhiệm vụ, như phân loại ảnh hoặc xử lý ngôn ngữ tự nhiên, giúp bạn áp dụng học chuyển tiếp (transfer learning) cho dự án của mình.

Tóm lại, Keras là một công cụ mạnh mẽ và linh hoạt giúp bạn xây dựng các mô hình học máy và học sâu một cách dễ dàng và nhanh chóng, đặc biệt là khi bạn làm việc với dự án liên quan đến học máy và sử dụng TensorFlow như backend.

Keras so với TensorFlow:

TensorFlow là một thư viện mã nguồn mở toàn diện để tạo và làm việc với mạng neural, chẳng hạn như những mô hình được sử dụng trong dự án Học máy (ML) và Học sâu.

Keras là một API cấp cao chạy trên nền tảng của TensorFlow. Keras giúp đơn giản hóa việc triển khai các mạng neural phức tạp với khung làm việc dễ sử dụng.

- Mức độ trừu tượng hóa:

Keras: Keras là một giao diện lập trình ứng dụng (API) cao cấp, nơi bạn có thể định nghĩa mô hình neural network bằng cách xếp các lớp lại với nhau một cách dễ dàng. Nó cung cấp một cách trừu tượng hóa để định nghĩa và đào tạo các mô hình học máy và học sâu.

TensorFlow: TensorFlow là một thư viện mạnh mẽ cho việc tính toán số học và học máy, chứ không phải là một API trừu tượng. Bạn cần định nghĩa mô hình neural network bằng cách xây dựng và tùy chỉnh các phép tính trên biểu đồ tính toán.

- Tích hợp:

Keras: Keras ban đầu là một thư viện độc lập, nhưng sau đó đã được tích hợp chặt chẽ vào TensorFlow. Hiện nay, Keras là một phần của TensorFlow và được sử dụng như một lớp trừu tượng cao cấp giúp đơn giản hóa việc xây dựng mô hình trên nền tảng TensorFlow.

TensorFlow: TensorFlow là một thư viện riêng biệt và không phụ thuộc vào bất kỳ API cụ thể nào. Nó cung cấp cho bạn sự linh hoạt tuyệt vời trong việc xây dựng và tùy chỉnh các mô hình học máy và học sâu.

- Độ linh hoạt:

Keras: Keras thích hợp cho người mới bắt đầu với học máy và học sâu, cũng như cho các dự án đòi hỏi sự nhanh chóng trong việc xây dựng mô hình. Nó giới hạn tính linh hoạt một chút, nhưng làm cho quá trình đào tạo mô hình đơn giản hơn.

TensorFlow: TensorFlow cung cấp cho bạn sự linh hoạt tối đa để tạo và tùy chỉnh mô hình của mình. Điều này thích hợp cho các dự án phức tạp và đòi hỏi tùy chỉnh cao.

- Học chuyển tiếp (Transfer Learning):

Keras: Keras cung cấp nhiều mô hình được đào tạo trước (pre-trained models) cho các nhiệm vụ phổ biến như phân loại ảnh hoặc xử lý ngôn ngữ tự nhiên. Điều này giúp bạn áp dụng học chuyển tiếp (transfer learning) một cách dễ dàng.

TensorFlow: TensorFlow cũng hỗ trợ học chuyển tiếp, nhưng việc triển khai có thể phức tạp hơn và đòi hỏi kiến thức sâu về cách hoạt động của mô hình.

Tóm lại, Keras là một lớp trừu tượng cao cấp giúp đơn giản hóa việc xây dựng mô hình trên TensorFlow và thích hợp cho những người mới bắt đầu hoặc các dự án đòi hỏi tính nhanh chóng. TensorFlow là một thư viện mạnh mẽ và linh hoạt hơn, thích hợp cho các dự án phức tạp và yêu cầu tùy chỉnh cao.

Sử dụng Keras khi:

- Bạn mới bắt đầu với học máy hoặc học sâu và muốn học một cách dễ dàng và nhanh chóng.

- Bạn muốn xây dựng mô hình nhanh chóng và tập trung vào cấu trúc của mô hình hơn là chi tiết triển khai.

- Dự án của bạn đủ phức tạp, nhưng bạn muốn giữ mã của mình ngắn gọn và dễ hiểu.

- Bạn muốn sử dụng các mô hình được đào tạo trước (pre-trained models) và thực hiện học chuyển tiếp một cách dễ dàng.

Sử dụng TensorFlow khi:

- Bạn muốn tùy chỉnh một cách chi tiết mô hình học máy hoặc học sâu của mình.

- Dự án của bạn đòi hỏi kiểm soát tối đa và linh hoạt trong việc thiết kế và triển khai mô hình.

- Bạn có kỹ năng kỹ thuật sâu và muốn sâu vào chi tiết triển khai mô hình.

- Bạn muốn tận dụng tối đa các tính năng và tính linh hoạt của TensorFlow, chẳng hạn như TPU (Tensor Processing Unit) hoặc tối ưu hóa linh hoạt cho mô hình của bạn.

Thực tế, một cách phổ biến là sử dụng Keras như một giao diện trừu tượng cho TensorFlow. Bạn có thể xây dựng mô hình của mình bằng Keras, nhưng dưới lớp trừu tượng này, Keras sẽ sử dụng TensorFlow để thực hiện các phép tính. Điều này kết hợp sự dễ dàng sử dụng của Keras với tính linh hoạt và hiệu suất của TensorFlow.

5 ví dụ:

Phân loại ảnh với mạng neural đơn giản:

```python
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.datasets import mnist
from keras.utils import to_categorical

# Tải dữ liệu từ MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Chuẩn hóa dữ liệu và mã hóa one-hot
X_train = X_train / 255.0
X_test = X_test / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Xây dựng mô hình
model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Biên dịch và đào tạo mô hình
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

```
Epoch 1/10
1500/1500 [==============================] - 4s 2ms/step - loss: 0.2929 - accuracy: 0.9158 - val_loss: 0.1523 - val_accuracy: 0.9558
Epoch 2/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.1322 - accuracy: 0.9610 - val_loss: 0.1299 - val_accuracy: 0.9615
Epoch 3/10
1500/1500 [==============================] - 4s 2ms/step - loss: 0.0910 - accuracy: 0.9735 - val_loss: 0.1007 - val_accuracy: 0.9685
Epoch 4/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0676 - accuracy: 0.9800 - val_loss: 0.0941 - val_accuracy: 0.9715
Epoch 5/10
1500/1500 [==============================] - 4s 2ms/step - loss: 0.0516 - accuracy: 0.9841 - val_loss: 0.0896 - val_accuracy: 0.9740
Epoch 6/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0407 - accuracy: 0.9872 - val_loss: 0.0885 - val_accuracy: 0.9746
Epoch 7/10
1500/1500 [==============================] - 4s 2ms/step - loss: 0.0319 - accuracy: 0.9905 - val_loss: 0.0977 - val_accuracy: 0.9730
Epoch 8/10
1500/1500 [==============================] - 4s 2ms/step - loss: 0.0242 - accuracy: 0.9928 - val_loss: 0.0860 - val_accuracy: 0.9764
Epoch 9/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0202 - accuracy: 0.9945 - val_loss: 0.1036 - val_accuracy: 0.9729
Epoch 10/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0172 - accuracy: 0.9950 - val_loss: 0.0961 - val_accuracy: 0.9755
<keras.src.callbacks.History at 0x232e0e05e10>
```

Giải thích: Ví dụ này mô tả cách xây dựng và đào tạo một mạng neural để phân loại chữ số viết tay từ bộ dữ liệu MNIST.

Dự đoán giá nhà với mạng neural hồi quy:

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
import pandas as pd

# Đọc dữ liệu từ tệp CSV
data = pd.read_csv('house_prices.csv')
X = data.drop('price', axis=1)
y = data['price']

# Chia dữ liệu thành tập huấn luyện và tập kiểm tra
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Xây dựng mô hình hồi quy
model = Sequential()
model.add(Dense(64, input_dim=3, activation='relu'))
model.add(Dense(1))

# Biên dịch và đào tạo mô hình
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)

# Đánh giá mô hình
mse = model.evaluate(X_test, y_test)
```

Ví dụ này minh họa việc xây dựng một mạng neural hồi quy để dự đoán giá nhà dựa trên các đặc trưng như diện tích, số phòng, và vị trí.

Phân loại văn bản với LSTM:

```
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np

# Dữ liệu văn bản và nhãn
texts = ["This is a positive sentence.", "That is a negative statement.", "I feel great today."]
labels = [1, 0, 1]

# Chuyển đổi văn bản thành dãy số và đệm chuỗi
tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
padded_sequences = pad_sequences(sequences)

# Xây dựng mô hình LSTM
model = Sequential()
model.add(Embedding(input_dim=1000, output_dim=64, input_length=padded_sequences.shape[1]))
model.add(LSTM(128))
model.add(Dense(1, activation='sigmoid'))

# Biên dịch và đào tạo mô hình
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(padded_sequences, np.array(labels), epochs=5)
```

Ví dụ này minh họa việc sử dụng mạng LSTM để phân loại văn bản thành hai lớp: tích cực (1) và tiêu cực (0).

Mô hình Học sâu đơn giản với một lớp ẩn:

```
from keras.models import Sequential
from keras.layers import Dense

# Khởi tạo mô hình
model = Sequential()

# Thêm lớp ẩn
model.add(Dense(units=10, activation='relu', input_dim=20))

# Lớp output
model.add(Dense(units=1, activation='sigmoid'))

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

Transfer Learning với mô hình đã được đào tạo trước

```
from keras.applications import VGG16
from keras.models import Model
from keras.layers import Flatten, Dense

# Load mô hình VGG16 đã được đào tạo trước
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Thêm các lớp mới
x = base_model.output
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# Xây dựng mô hình mới
model = Model(inputs=base_model.input, outputs=predictions)
```

4.4

* Code 1:

```
import tensorflow as tf
import numpy as np
#print("TensorFlow version:", tf.__version__)

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
tf.keras.layers.Flatten(input_shape=(28, 28)),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(10)])

predictions = model(x_train[:1])
predictions
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
loss_fn(y_train[:1], predictions)
model.compile(optimizer='adam',
loss=loss_fn,
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test, verbose=2)
```

```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3033 - accuracy: 0.9113
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1465 - accuracy: 0.9561
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1097 - accuracy: 0.9673
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0888 - accuracy: 0.9728
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0746 - accuracy: 0.9767
313/313 - 0s - loss: 0.0737 - accuracy: 0.9775 - 437ms/epoch - 1ms/step
[0.0737399086356163, 0.9775000214576721]
```

* Giải thích:

- Import các thư viện TensorFlow và NumPy:

```
import tensorflow as tf
import numpy as np
```

- Tải bộ dữ liệu MNIST và chia thành dữ liệu huấn luyện và kiểm tra:

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

- Chuẩn hóa giá trị pixel của hình ảnh về khoảng [0, 1]:

```
x_train, x_test = x_train / 255.0, x_test / 255.0
```

- Xây dựng mô hình mạng neural. Mô hình này bao gồm lớp Flatten để chuyển hình ảnh từ 2D thành 1D, một lớp fully connected với 128 đơn vị và hàm kích hoạt ReLU, một lớp Dropout để ngăn overfitting, và lớp fully connected cuối cùng với 10 đơn vị (tương ứng với số lớp dự đoán):

```
model = tf.keras.models.Sequential([
tf.keras.layers.Flatten(input_shape=(28, 28)),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(10)])
```

- Tạo dự đoán cho ví dụ đầu tiên trong tập dữ liệu huấn luyện:

```
predictions = model(x_train[:1])
```

- Định nghĩa hàm mất mát là Sparse Categorical Crossentropy:

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

- Tính mất mát cho ví dụ đầu tiên trong tập dữ liệu huấn luyện:

```
loss_fn(y_train[:1], predictions)
```

- Biên dịch mô hình với tối ưu hóa 'adam', hàm mất mát, và metric 'accuracy':

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

- Huấn luyện mô hình trên dữ liệu huấn luyện trong 5 epochs:

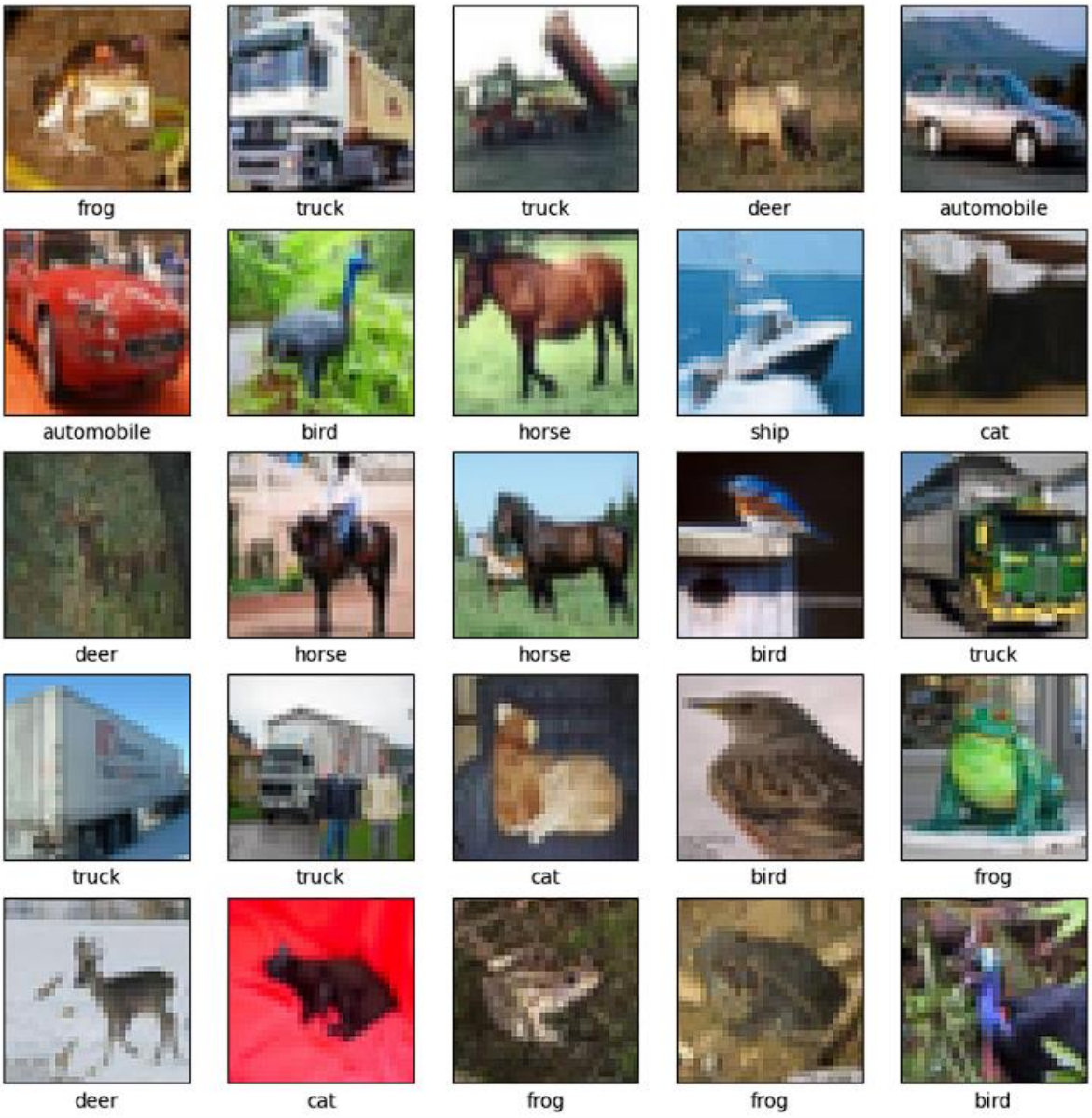
```
model.fit(x_train, y_train, epochs=5)
```

- Đánh giá mô hình trên dữ liệu kiểm tra và in ra kết quả:

```
model.evaluate(x_test, y_test, verbose=2)
```

* Code 2:

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

# Load CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    plt.xlabel(class_names[train_labels[i][0]])

plt.show()
```

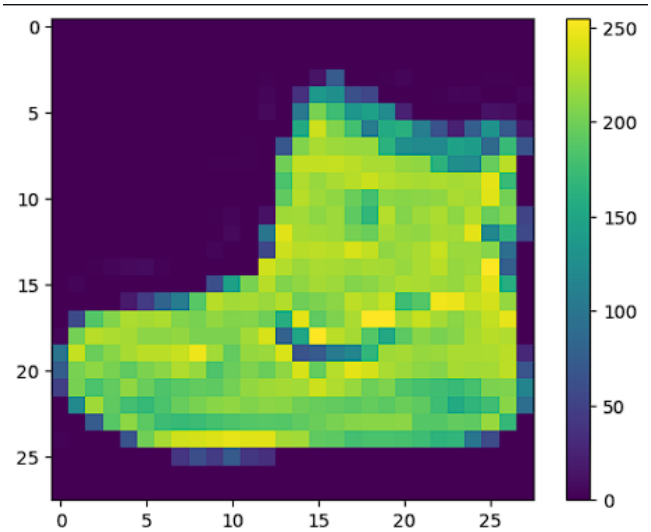

frog
truck
truck
deer
automobile
automobile
bird
horse
ship
cat
deer
horse
horse
bird
truck
truck
truck
cat
bird
frog
deer
cat
frog
frog
bird

* Giải thích:

```python
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt


# Tải dữ liệu CIFAR-10 và chia thành tập huấn luyện và tập kiểm tra
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()


# Chuẩn hóa giá trị pixel để nằm trong khoảng từ 0 đến 1
train_images, test_images = train_images / 255.0, test_images / 255.0


# Định nghĩa tên các lớp trong CIFAR-10
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
 'dog', 'frog', 'horse', 'ship', 'truck']


# Trực quan hóa 25 ảnh từ tập huấn luyện
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i]) # Hàm hiển thị ảnh


# Nhãn trong bộ dữ liệu CIFAR-10 được biểu diễn dưới dạng mảng, đó là lý do tại sao bạn cần chỉ số bổ sung.
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

* Code 3:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Kiểm tra phiên bản TensorFlow
print(tf.__version__)

# Tải dữ liệu Fashion MNIST
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# Danh sách tên lớp
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Kiểm tra kích thước tập huấn luyện
train_images.shape

# Hiển thị một hình ảnh từ tập huấn luyện
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()

# Chuẩn hóa giá trị pixel về khoảng [0, 1]
train_images = train_images / 255.0
test_images = test_images / 255.0

# Hiển thị 25 hình ảnh từ tập huấn luyện
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```




Ankle boot
T-shirt/top
T-shirt/top
Dress
T-shirt/top
Pullover
Sneaker
Pullover
Sandal
Sandal
T-shirt/top
Ankle boot
Sandal
Sandal
Sneaker
Ankle boot
Trouser
T-shirt/top
Shirt
Coat
Dress
Trouser
Coat
Bag
Coat