Machine Learning

Practical Basis

Teaching Assistant： Shuwei Yan

Today’s Topics

- Deepseek
- Python tutorial
  - Numpy
  - Matplotlib
  - Pytorch

Today's Topics

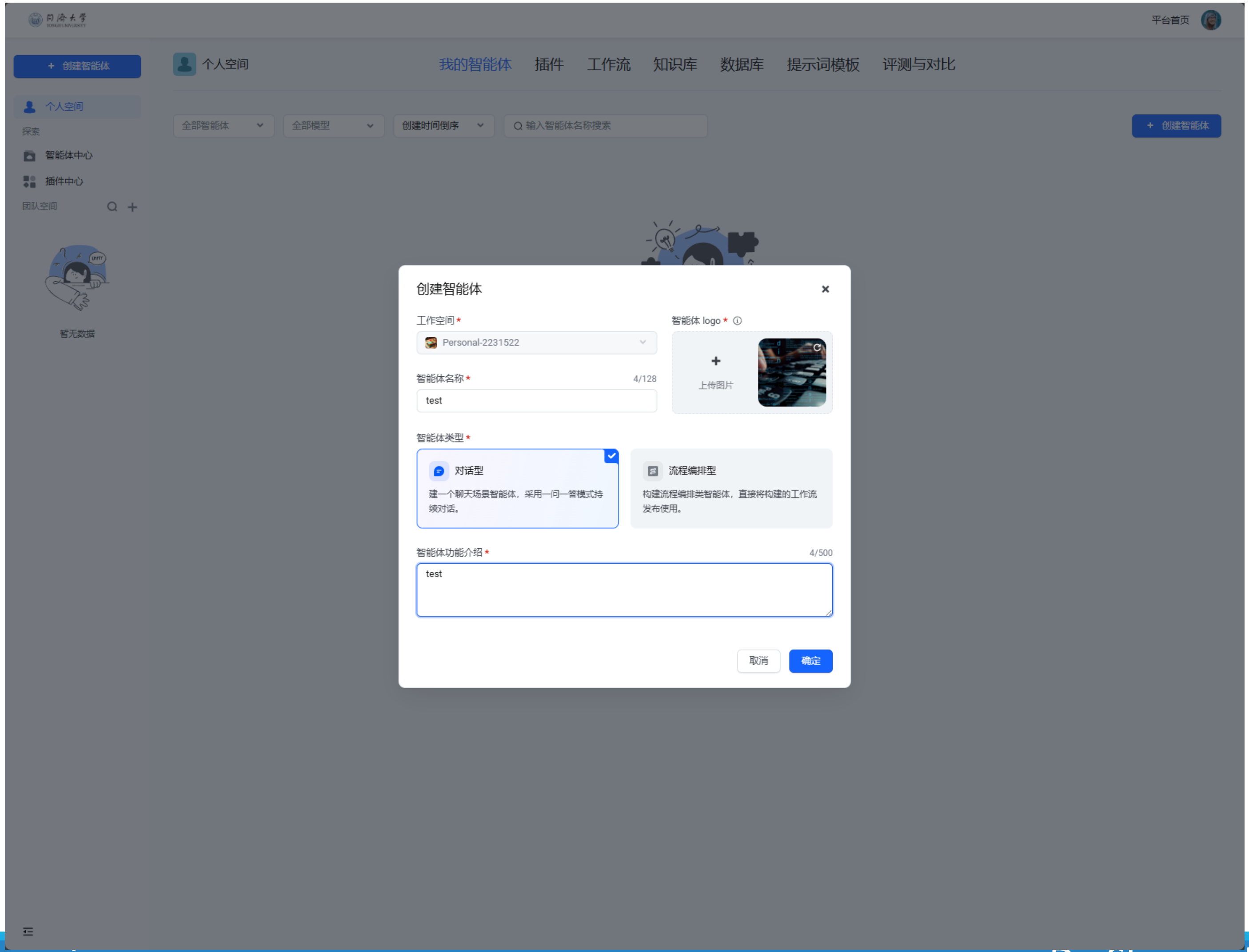
- ***Deepseek***

- Python tutorial
  - Numpy
  - Matplotlib
  - Pytorch

Deepseek

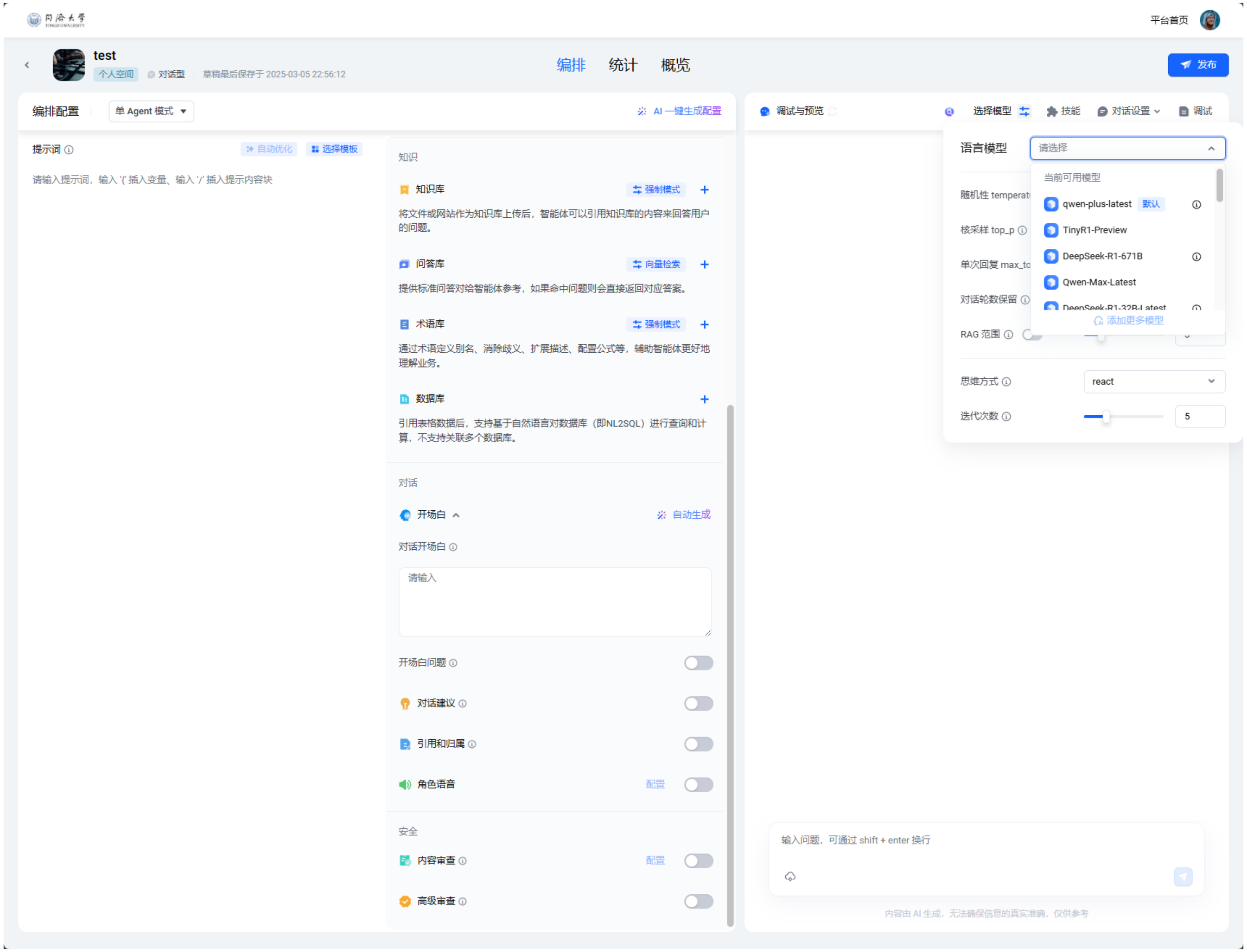
Tongji AI Agent

agent.tongji.edu.cn



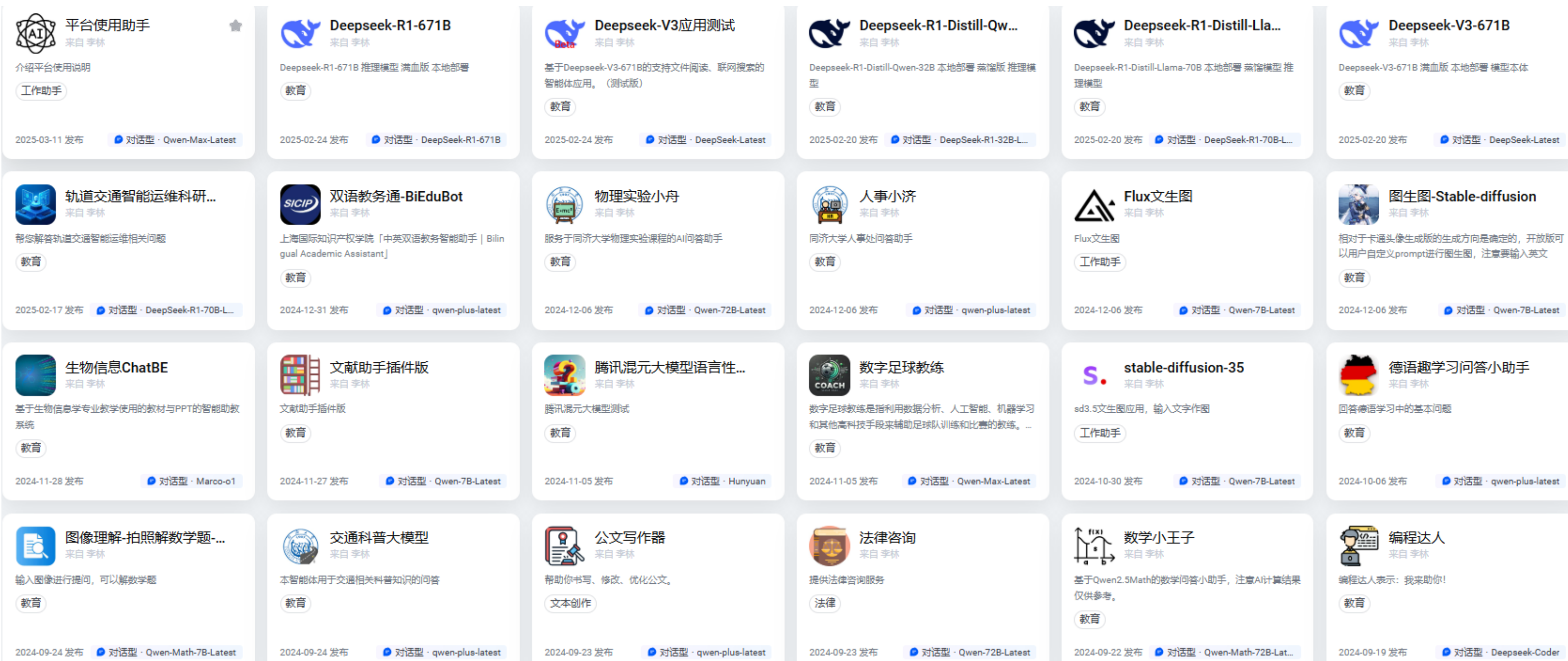
Deepseek

Tongji AI Agent https://dev.tongji.edu.cn/agent-document/#/



Deepseek

Tongji AI Agent



Today's Topics

- Deepseek
- Python tutorial
  - ***Numpy***
  - Matplotlib
  - Pytorch

Numpy

Numpy is the core library for scientific computing in Python.

[Useful for processing data in practice of machine learning]

```
import numpy as np
```

Arrays

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers.

The number of dimensions is the rank of the array;

the shape of an array is a tuple of integers giving the size of the array along each dimension.

Arrays Initialization

Vector

```
a = np.array([1, 2, 3])
# Create a rank 1 array
print(type(a), a.shape, a[0], a[1], a[2])

a[0] = 5  # Change an element of the array
print(a)
```

```
<class 'numpy.ndarray'> (3,) 1 2 3

[5 2 3]
```

Arrays Initialization

Matrix

```
b = np.array([[1,2,3],[4,5,6]])
# Create a rank 2 array
print(b)
```

```
[[1 2 3]
[4 5 6]]
```

```
print(b.shape)
print(b[0, 0], b[0, 1], b[1, 0])
```

```
(2, 3)
1 2 4
```

Arrays Initialization

```
a = np.zeros((2,2))  # Create an array of all zeros
print(a)
```

```
[[0. 0.]
 [0. 0.]]
```

```
b = np.ones((1,2))  # Create an array of all ones
print(b)
```

```
[[1. 1.]]
```

```
c = np.full((2,2), 7)  # Create a constant array
print(c)
```

```
[[7 7]
 [7 7]]
```

Arrays Initialization

```
d = np.eye(2)  # Create a 2x2 identity matrix
print(d)
```

```
[[1. 0.]
 [0. 1.]]
```

```
e = np.random.random((2,2))  # Create an array filled with random values
print(e)
```

```
[[0.02711854 0.96345501]
[0.46516113 0.97936752]]
```

```
f = np.arange(3)  # Create an array filled with integers from 0 to n-1.
print(f)
```

```
[0 1 2]]
```

Arrays Initialization

Element Type

http://docs.scipy.org/doc/numpy/reference/arrays.dtypes.html

```
x = np.array([1, 2])
# Let numpy choose the datatype
y = np.array([1.0, 2.0])
# Let numpy choose the datatype
z = np.array([1, 2], dtype=np.int64)
# Force a particular datatype
print(x.dtype, y.dtype, z.dtype)
```

```
int64 float64 int64
```

Array Operation

Array Indexing: Starts from 0

Array Slicing: Must specify the corresponding slice for each dimension of the array

```python
import numpy as np
# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
b = a[:2, 1:3]
print(b)
```

```
[[2 3]
 [6 7]]
```

Array Operation

```
# Create the following rank 2 array with shape (3, 4)
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print(a)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

```
row_r1 = a[[1], :]  # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)
```

```
[[5 6 7 8]] (1, 4)
```

```
# An example of integer array indexing .
b = np.array([0, 1, 0])
print(a[[0, 1, 2],b])
```

```
[1 6 9]
```

Array Operation

Boolean array

```
import numpy as np
a = np.array([[1,2], [3, 4], [5, 6]])
bool_idx = (a > 2)
# Find the elements of a that are bigger than 2;
# this returns a numpy array of Booleans of the same
# shape as a, where each slot of bool_idx tells
# whether that element of a is > 2.
print(bool_idx)
```

```
[[False False]
[ True True]
[ True True]]
```

Array Operation

Boolean array

```
# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values of bool_idx
print(a[bool_idx])
# We can do all of the above in a single concise statement:
print(a[a > 2])
```

```
[3 4 5 6]
[3 4 5 6]
```

Array math

Basic Mathematical Functions Perform Element-wise Operations on Arrays

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
# Elementwise sum; both produce the array
print(x + y)
print(np.add(x, y))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
```

```
# Elementwise difference; both produce the array
print(x - y)
print(np.subtract(x, y))
```

Lists

Concatenate

```
[1,2,3] + [5,4,7]
```

```
[1, 2, 3, 5, 4, 7]
```

Array math

Basic Mathematical Functions Perform Element-wise Operations on Arrays

```
# Elementwise product; both produce the array
print(x * y)
print(np.multiply(x, y))
```

```
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
```

```
# Elementwise division; both produce the array
# [[ 0.2        0.33333333]
#  [ 0.42857143 0.5 ]]
print(x / y)
print(np.divide(x, y))
```

Array math

Matrix Multiplication Requires Dimension Alignment

```
x = np.array([[1,2],[3,4]])
v = np.array([[9], [10]])  # shape = (2,1)
w = np.array([[9, 10]])  # shape = (1,2)
```

```
# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(x.dot(w))
```

```
[[29]
 [67]]
ValueError: shapes (2,2) and (1,2) not aligned
```

Array math

Vector operations have broadcasting properties (broadcasting: using a smaller array multiple times to perform operations on a larger array).

```
x = np.array([[1,2],[3,4]])
v = np.array([9,10])
```

```
# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))
```

```
[29 67]
[29 67]
```

Array math

Vector operations have broadcasting properties (broadcasting: using a smaller array multiple times to perform operations on a larger array).

```
import numpy as np
# We will add the vector v to each row of the matrix x, storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v # Add v to each row of x using broadcasting
print(y)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

Array math

NumPy provides many useful functions for performing calculations on arrays.

```
x = np.array([[1,2],[3,4]])
print(np.sum(x))
print(np.sum(x, axis=0))
print(np.sum(x, axis=1))
print(x.T)
```

```
10
[4 6]
[3 7]
[[1 3]
 [2 4]]
```

Exercise

```
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

B = A[0:2, :]

C = B + np.array([1, 2]).reshape(2, 1)

D = C[C > 3]

result = np.sum(D)

print("Array B:\n", B)
print("Array C:\n", C)
print("Array D:\n", D)
print("Final result (sum of D):", result)
```

Exercise

```
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

B = A[0:2, :]

C = B + np.array([1, 2]).reshape(2, 1)

D = C[C > 3]

result = np.sum(D)

print("Array B:\n", B)
print("Array C:\n", C)
print("Array D:\n", D)
print("Final result (sum of D):", result)
```

1.Array B:

[[1 2 3] [4 5 6]]

2.Array C:

[[2 3 4] [6 7 8]]

3.Array D:

[4 6 7 8]

4.Final Result:

25

Today’s Topics

- Deepseek
- Python tutorial
  - Numpy
  - ***Matplotlib***
  - Pytorch

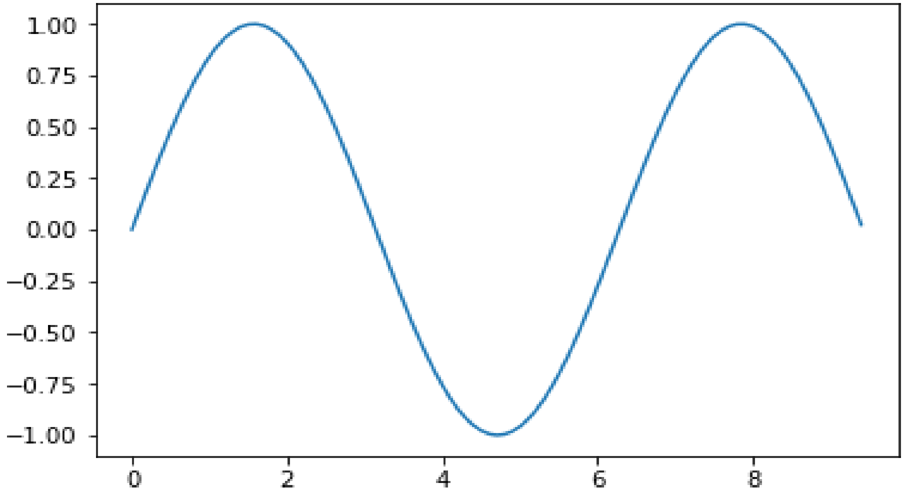
Matplotlib

Similar to MATLAB's plotting system.

```
import matplotlib.pyplot as plt
```

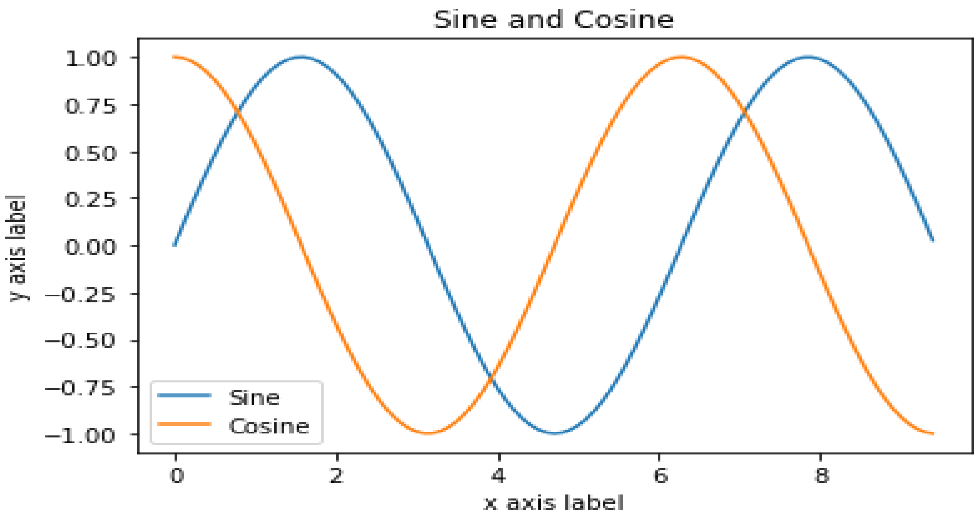
Matplotlib

```
# Compute the x and y coordinates for # points on a
sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
# Plot the points using matplotlib
plt.plot(x, y)
```



Matplotlib

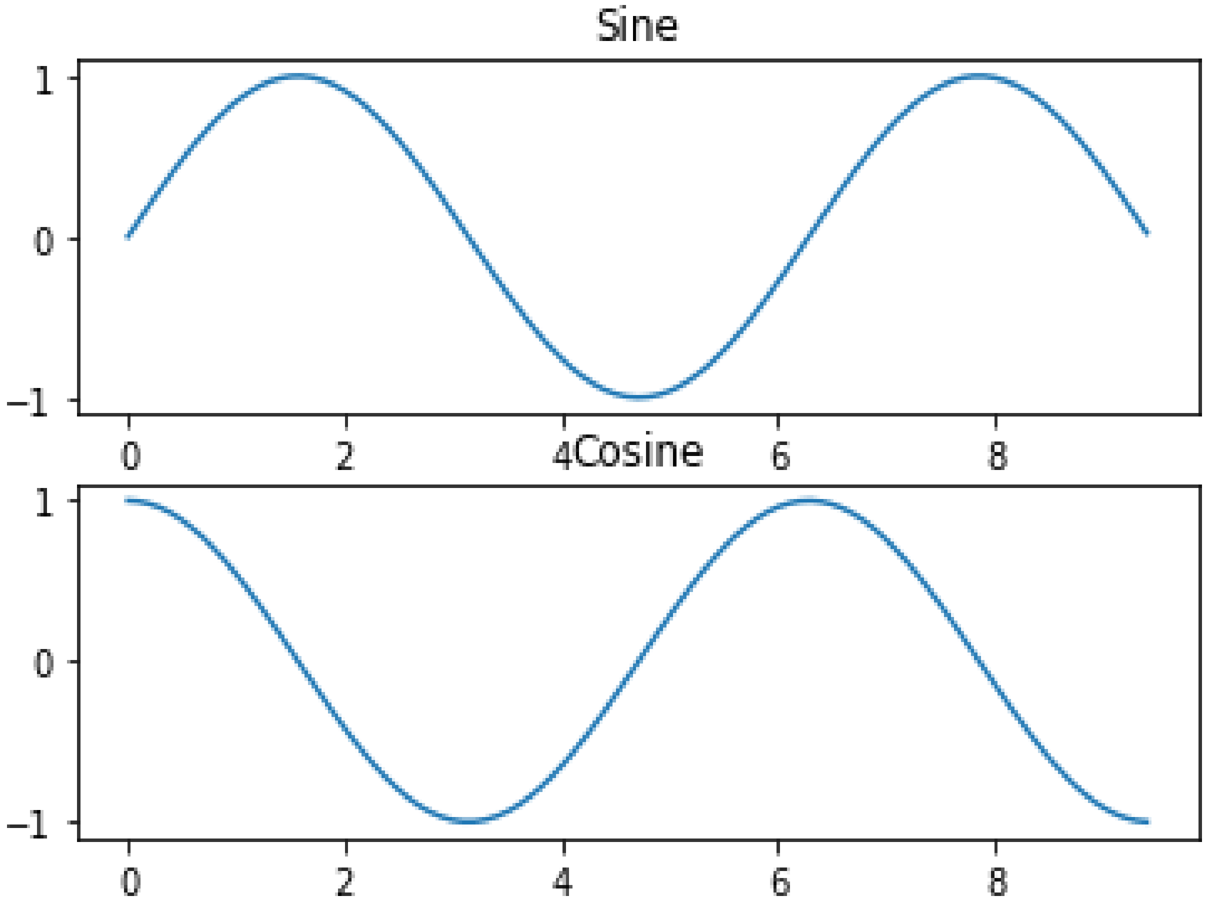
```
y_sin = np.sin(x)
y_cos = np.cos(x)
# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
```



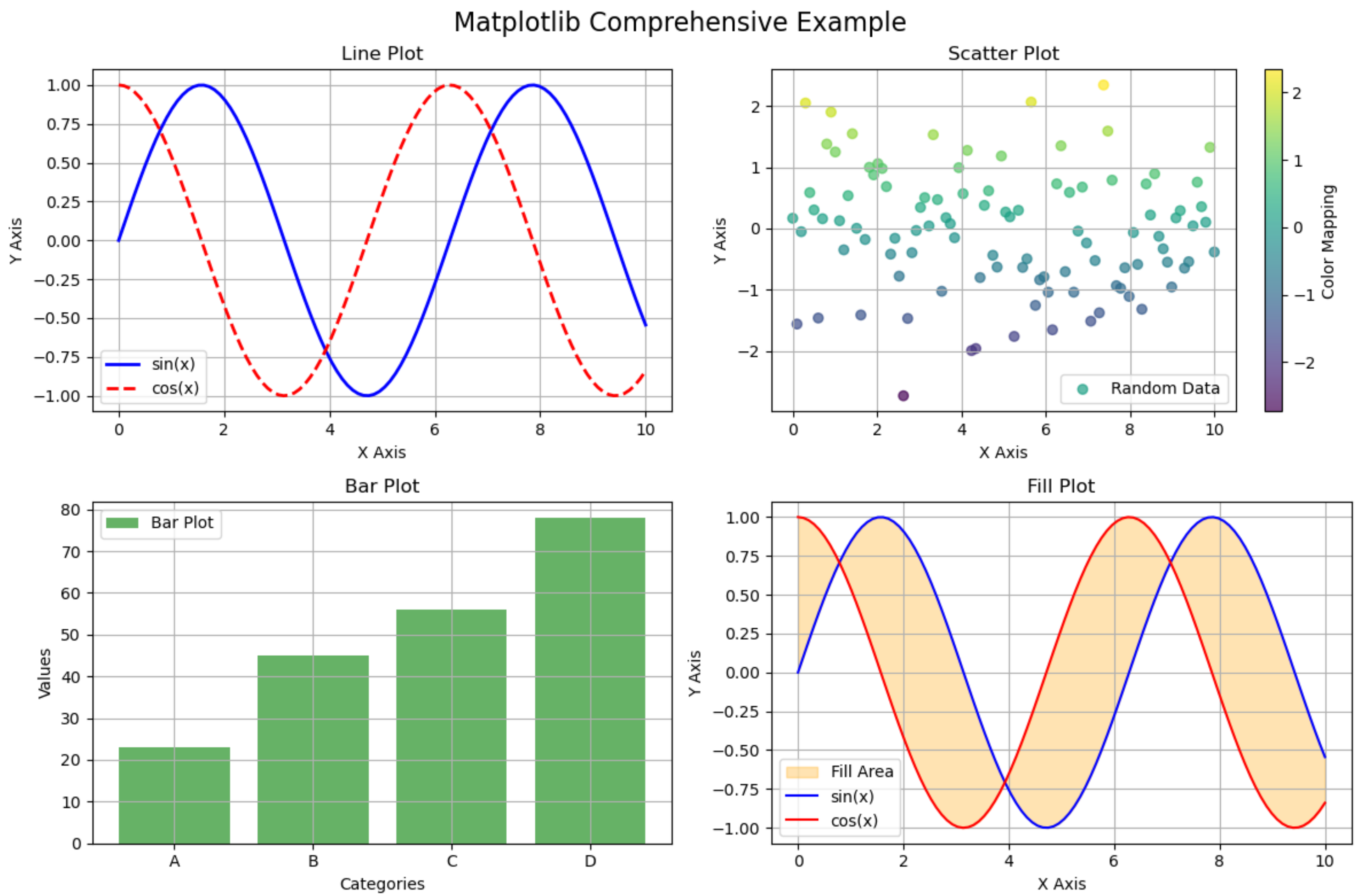
Matplotlib

```
# Compute the x and y coordinates for points on sine
# and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)
# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)
# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')
# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')
# Show the figure.
plt.show()
```

Matplotlib



Exercise



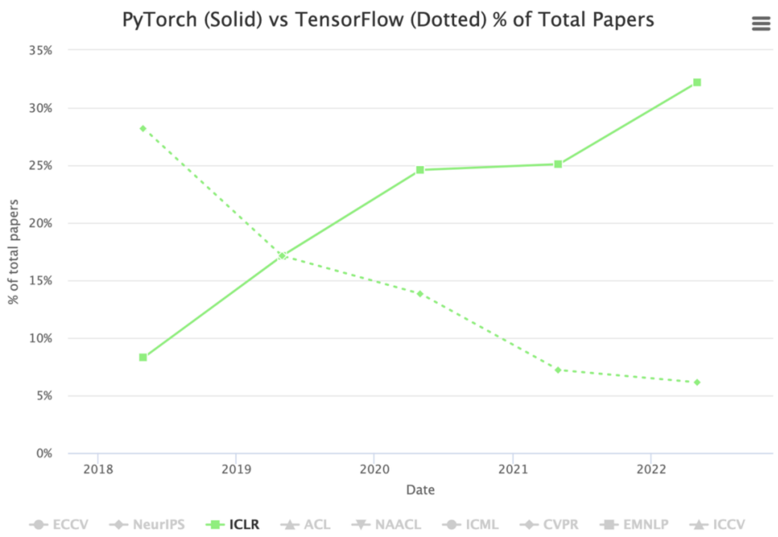
Matplotlib Comprehensive Example
Line Plot
Scatter Plot
Bar Plot
Fill Plot
X Axis
Y Axis
sin(x)
cos(x)
Random Data
Color Mapping
Bar Plot
Values
Categories
A
B
C
D
Fill Area

Today's Topics

- Deepseek
- Python tutorial
  - Numpy
  - Matplotlib
  - ***Pytorch***

Pytorch

Pytorch vs TensorFlow



Tensor

Common Data Structures in PyTorch, Similar to Arrays

Advantages: Supports GPU-accelerated computation, automatic differentiation, and dynamic computation graphs.

Tensor Initialization

```
import torch
t = torch.empty(1,2)
print(t)
t = torch.zeros(1, 2)
print(t)

t = torch.rand(1,2)
print(t)
t = torch.randn(1,2)
print(t)
```

```
tensor([[0., 0.]])
tensor([[0., 0.]])
tensor([[0.5755, 0.2029]])
tensor([[-0.2145,  0.1351]])
```

Tensor Initialization

```python
import torch
t = torch.tensor([1,2])
# torch.int64
print(t)
t = torch.zeros((1, 2))
print(t)

import numpy as np
n = np.array([1, 2])
t = torch.tensor(n)
print(t)
```

```
tensor([1, 2])
tensor([[0., 0.]])
tensor([1, 2], dtype=torch.int32)
```

Tensor Operations

```
t1 = torch.tensor([[1, 2, 3], [4, 5, 6]])

print(t1.flatten())

print(t1.reshape(2, 1, 3))

print(t1.unsqueeze(1))

print(t1.unsqueeze(1).squeeze(1))

……
```

```
tensor([1, 2, 3, 4, 5, 6])
tensor([[[1, 2, 3]],[[4, 5, 6]]])
tensor([[[1, 2, 3]], [[4, 5, 6]]])
tensor([[1, 2, 3], [4, 5, 6]])
```

Tensor Reading

```
t1 = torch.arrange((1, 11))
print(t1)
print(t1[0])
print(t1[0].item())
```

```
tensor([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
tensor(1)
1
```

Tensor Gradient Computation

```
x = torch.randn(3,4,requires_grad = True)
print(x)
b = torch.randn(3,4,requires_grad = True)
t = x + b
y = t.sum()
y.backward()
print(b.grad)
```

```
tensor([[-1.1978, -0.2716, -0.8612,  0.7631],
        [ 1.1333, -1.1994,  1.2061, -0.5115],
        [-1.0289,  0.1047, -1.5174,  0.4490]],
requires_grad=True)
tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]])
```

Tensor Gradient Computation

```
x = torch.randn(1,2,requires_grad = True)
print(x)
b = torch.randn(1,2,requires_grad = True)
print(b)
t = x * b
y = t.sum()
y.backward()
print(x.grad)
print(b.grad)
```

```
tensor([[-0.9592, -0.3329]], requires_grad=True)
tensor([[-0.3429, -0.4745]], requires_grad=True)
tensor([[-0.3429, -0.4745]])
tensor([[-0.9592, -0.3329]])
```

Machine Learning Training Framework

1.Import Required Libraries

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
```

- **torch**: Core PyTorch library for tensors and autograd.
- **torch.nn**: Modules for building neural networks.
- **torch.optim**: Optimization algorithms.
- **torchvision**: Datasets and image processing tools.

Machine Learning Training Framework

2. Data Preparation

We use the CIFAR-10 dataset as an example:

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True,
                                        transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4, shuffle=True)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True,
                                       transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4, shuffle=False)
```

- Convert images to tensors.
- Normalize pixel values.
- Use DataLoader for efficient batch loading.

Machine Learning Training Framework

2. Data Preparation

```
class CustomDataset(Dataset):
    def __init__(self, data_dir, transform=None):
        self.data_dir = data_dir
        self.transform = transform
        self.image_files = os.listdir(data_dir)

    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, idx):
        img_path = os.path.join(self.data_dir, self.image_files[idx])
        image = Image.open(img_path).convert('RGB')

        if self.transform:
            image = self.transform(image)

        label = int(self.image_files[idx].split('_')[0])
        return image, label
```

- Create dataset with torch.utils.data.Dataset
- Implement __len__() and __getitem__()
- Use transformations to preprocess images

Machine Learning Training Framework

2. Data Preparation

```
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
```

Data Preprocessing

- Use transforms for data augmentation
- Horizontal flip, rotation, and cropping
- Normalize images for stable training

Machine Learning Training Framework

3. Define a Neural Network

A simple CNN with:

- Two Conv2d layers for feature extraction.
- A MaxPool2d layer for downsampling.
- Three fully connected layers for classification.
- ReLU activations.
- The forward function defines the computation flow.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()
```

Machine Learning Training Framework

4. Define Loss Function and Optimizer

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

- **Loss Function**: CrossEntropyLoss, suitable for multi-class classification.
- **Optimizer**: SGD with a learning rate of 0.001 and momentum 0.9 for faster convergence.

Machine Learning Training Framework

5. Train the Model

```
for epoch in range(2):  # Loop over dataset multiple times
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data

        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 2000 == 1999:  # Print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0
print('Finished Training')
```

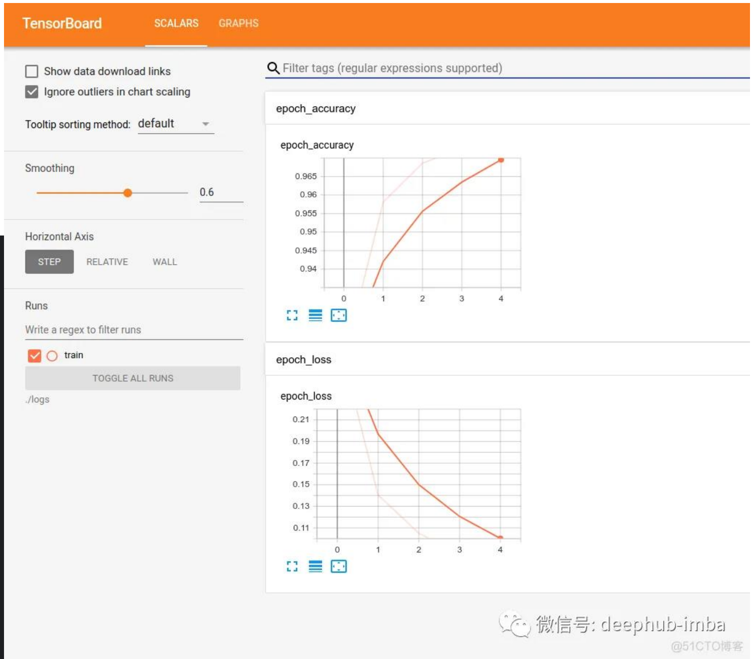
Training loop steps:

1. Iterate over epochs.
2. Load mini-batches.
3. Forward pass to compute predictions.
4. Compute loss.
5. Backpropagate gradients.
6. Update model parameters.
7. Print loss every 2000 batches for monitoring.

Machine Learning Training Framework

5. Train the Model

Using TensorBoard to track training loss




```python
from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter('runs/experiment1')

for epoch in range(2):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        ......
        if i % 100 == 99:
            writer.add_scalar('Training Loss', running_loss / 100, epoch * len(trainloader) + i)
            running_loss = 0.0

writer.close()
```

1.Create a TensorBoard writer

2.Log loss during training

3.Launch TensorBoard

- tensorboard --logdir=runs
- Open http://localhost:6006/ to view graphs

Machine Learning Training Framework

6. Evaluate the Model

```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy: {100 * correct / total:.2f}%')
```

- Disable gradient computation for efficiency.
- Perform inference on the test set.
- Compare predicted labels with ground truth.
- Compute accuracy as the performance metric.

Machine Learning Training Framework

7. Save and Load Model

- Save only model parameters (Recommended)
- Load weights into an existing model architecture

```
torch.save(net.state_dict(), 'model.pth')

net = Net()
net.load_state_dict(torch.load('model.pth'))
```

- Save the entire model
- Includes architecture and parameters, can be used directly

```
torch.save(net, 'model_complete.pth’)

net = Net()
net = torch.load('model_complete.pth')
```