Deep Learning Packages and resources

Runpeng Dai

The University of North Carolina at Chapel Hill

Content

1 Introduction to PyTorch basics

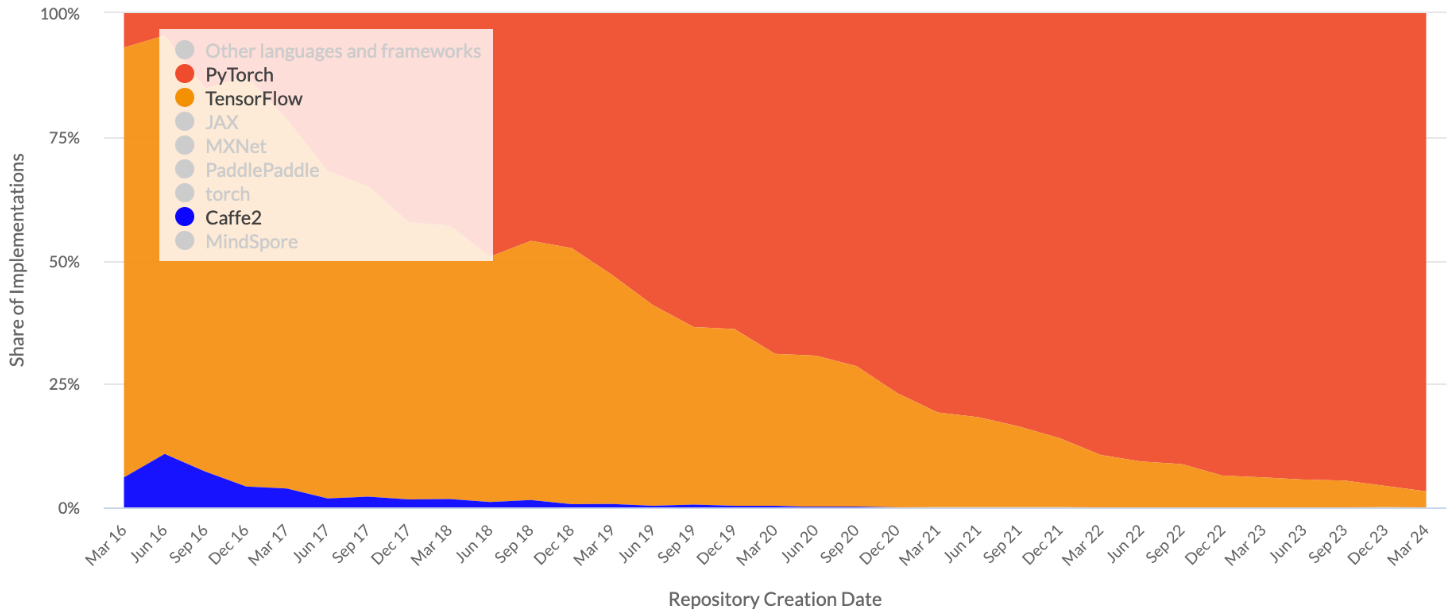
2 An example on Colab environment

3 Resources – Hugging-face, tensorboard, W&B

Why Torch?

Frameworks

Paper Implementations grouped by framework



A simple torch code

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# Define a simple neural network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.dropout(x)
        x = self.fc2(x)
        return F.log_softmax(x)

# Create an instance of the network
net = Net()
```

```
# Data loading
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32,
shuffle=True)

# Define a loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001)

# Training loop
for epoch in range(10):
    for inputs, targets in train_loader:
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

Tensors

- Tensors are the core data structures in PyTorch, used to encode inputs and model weights.
- While many tensor operations resemble those of NumPy arrays, they are specifically optimized for deep learning and can be transferred to a GPU for accelerated processing.

Numpy style operations

```
tensor = torch.ones(4, 4)
print(f"First row: {tensor[0]}")
print(f"First column: {tensor[:, 0]}")
print(f"Last column: {tensor[..., -1]}")
tensor[:,1] = 0
print(tensor)
```

Out:

```
First row: tensor([1., 1., 1., 1.])
First column: tensor([1., 1., 1., 1.])
Last column: tensor([1., 1., 1., 1.])
tensor([[1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.]])
```

Capability of GPU computing

```
# We move our tensor to the GPU if available
if torch.cuda.is_available():
    tensor = tensor.to("cuda")
```

Easy converting from and to numpy array

```
n = np.ones(5)
t = torch.from_numpy(n)
n = t.numpy()
```

Datasets

- Torch can assist in loading and preprocessing datasets (Dataset-Dataloader pipeline). It also provides a number of pre-loaded datasets (e.gMNIST).
- Hugging face also have similar module called Datasets.

Loading FashionMNIST

```
from torchvision import datasets
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor())
```

Building a Dataloader

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(training_data, batch_size=64,
shuffle=True)
train_features, train_labels = next(iter(train_dataloader))
```

In practice, many researches opt to write their own data processing and sampling code to gain greater flexibility.

Network structure

- In PyTorch, neural networks are defined by subclassing nn.Module. The network's layers and parameters are initialized within the __init__ method, while the forward method specifies how input data flows through these layers to produce the output.
- After instantiate the network, the network do forward by passing the input data.

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

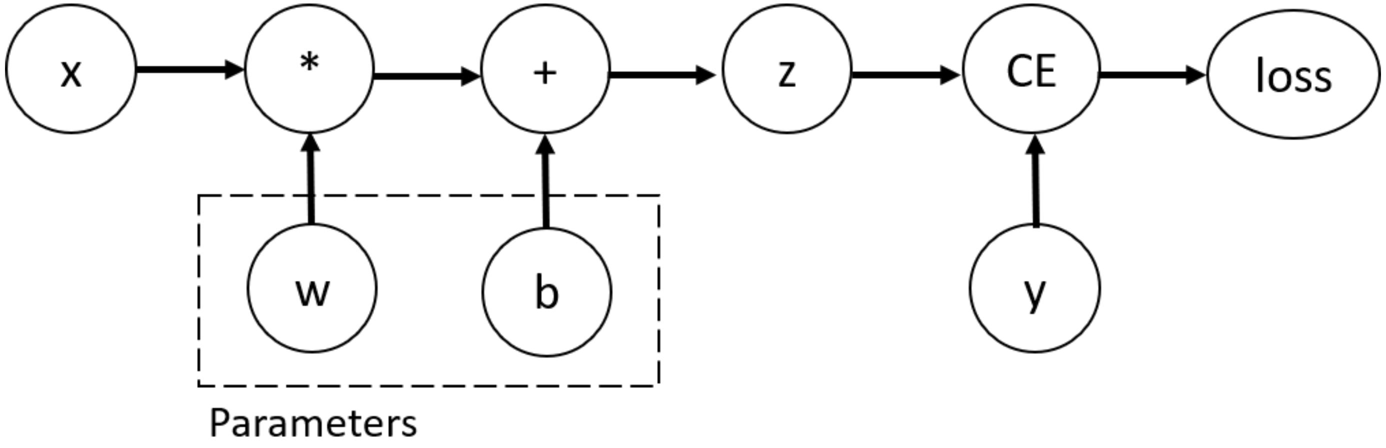
Instantize and send weights to GPU

```
model = NeuralNetwork().to(device)
X = torch.rand(1, 28, 28, device=device)
logits = model(X)
```

It is easy to implement normalizing methods (dropout batchnorm etc)

Auto differentiation

- Pytorch has a built-in engine called autograd to deal with gradients and updates.



- Pytorch have the notion of computation graph.

- Each time we pass input through the network (forward pass), PyTorch dynamically constructs a computational graph that tracks how the final output is related to all tensors requiring gradients.

- When you use backward(backward propagation) to calculate the grad, pytorch calculates the gradient of each component that requires_grad and store it.

```
x = torch.ones(5)  # input tensor
y = torch.zeros(3)  # expected output
w = torch.randn(5, 3, requires_grad=True)
b = torch.randn(3, requires_grad=True)
z = torch.matmul(x, w)+b
loss = torch.nn.functional.binary_cross_entropy_with_logits(z, y)
```

Loss.backward()

Optimization

- Optimization algorithms define how model parameters adjust to reduce model error in each training step.
- Many optimization algorithms (e.g. Adam) are available in torch.optim.

```
# Define the optimizer
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

# Within each training step, first reset the gradient of parameters
# Then calculate the gradients of the loss w.r.t. each parameter.
optimizer.zero_grad()

# Adjust the parameters using gradients
optimizer.step()
```

A simple torch code

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

# Define a simple neural network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.dropout(x)
        x = self.fc2(x)
        return F.log_softmax(x)

# Create an instance of the network
net = Net()
```

```
# Data loading
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32,
shuffle=True)

# Define a loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001)

# Training loop
for epoch in range(10):
    for inputs, targets in train_loader:
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

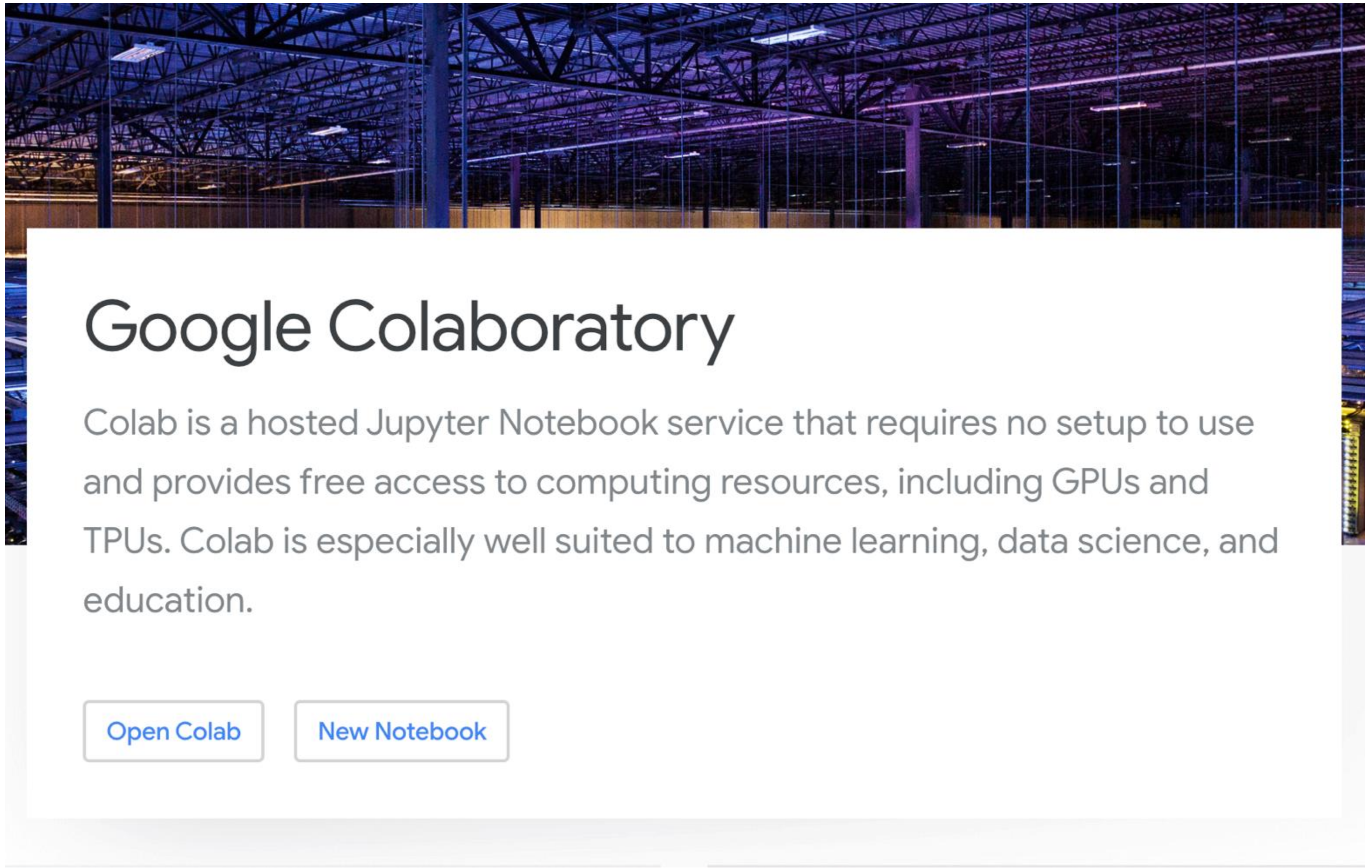
Content

1 Introduction to PyTorch basics

2 Setup the colab environment

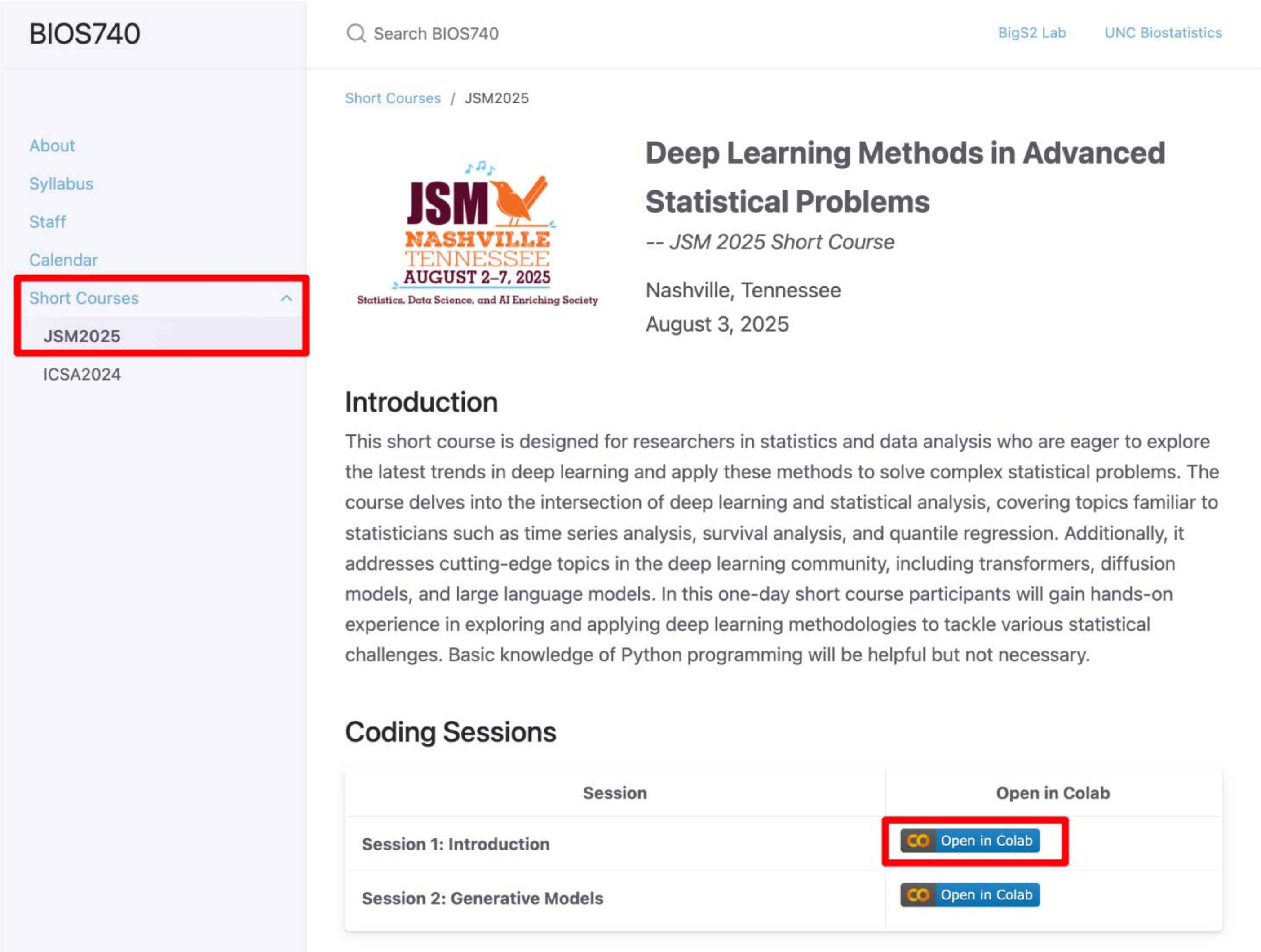
3 Resources – Hugging-face, Paperwithcode, tensorboard, W&B

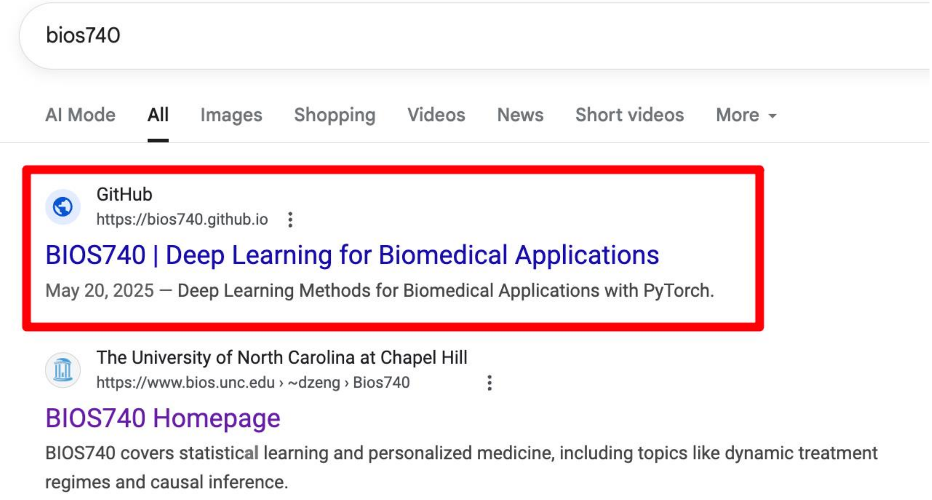
Google Colab



- Google Colab, short for Colab, is a free, cloud-based platform provided by Google Research. It allows users to write and execute Python code through a web browser.
- One of Colab's key features is the use of Jupyter Notebooks
- Colab provides free access to powerful hardware accelerators, including GPUs and TPUs
- Colab can mount google drive.

Setup

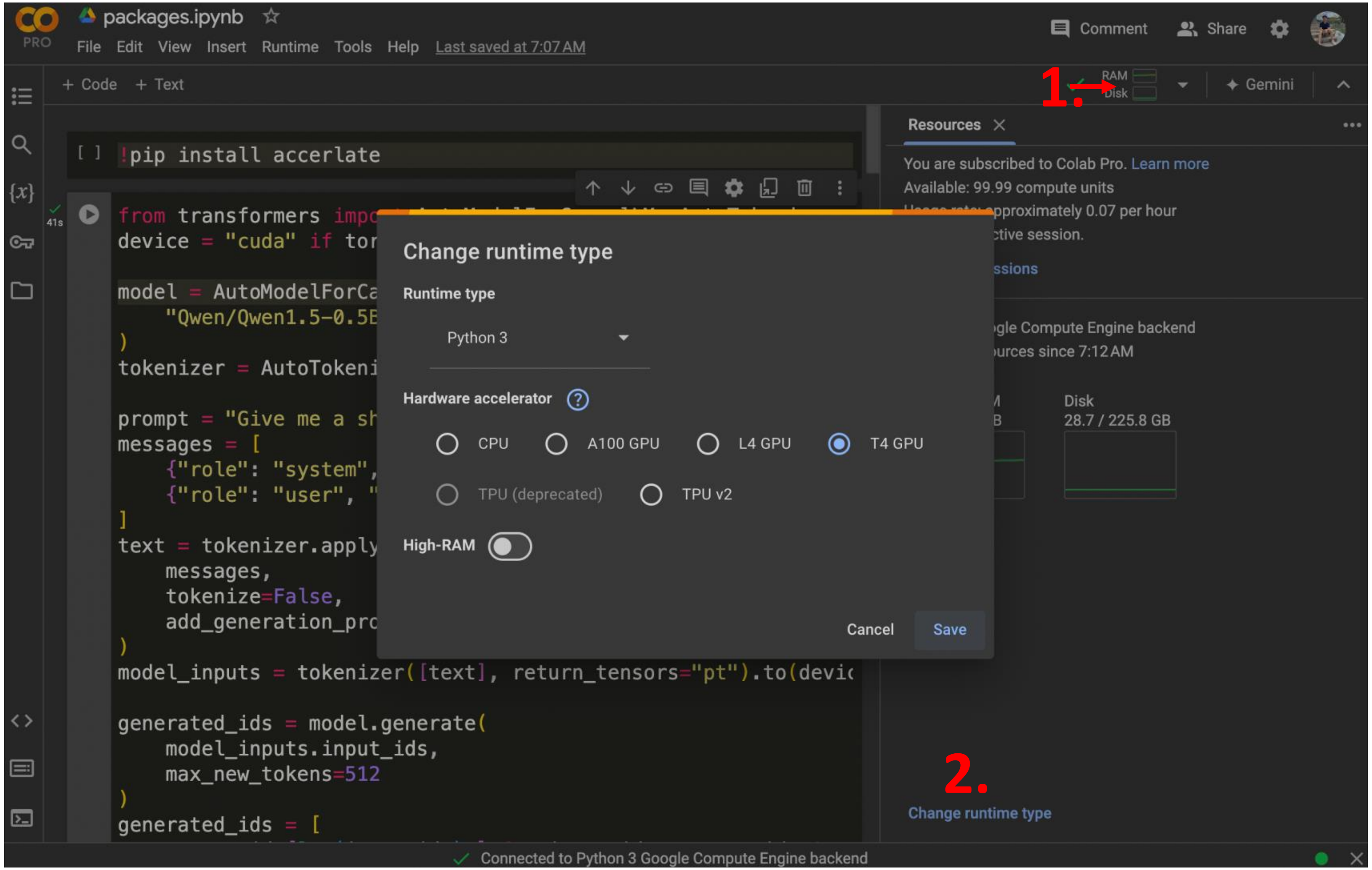




- We offer two small coding sessions in the short course.
- First, open the course page https://bios740.github.io/ and go to JSM2025
- Then open the first colab program by clicking

Code session – Pytorch Basics

- Change the runtime type to T4 GPU to have access to GPU computing resource.



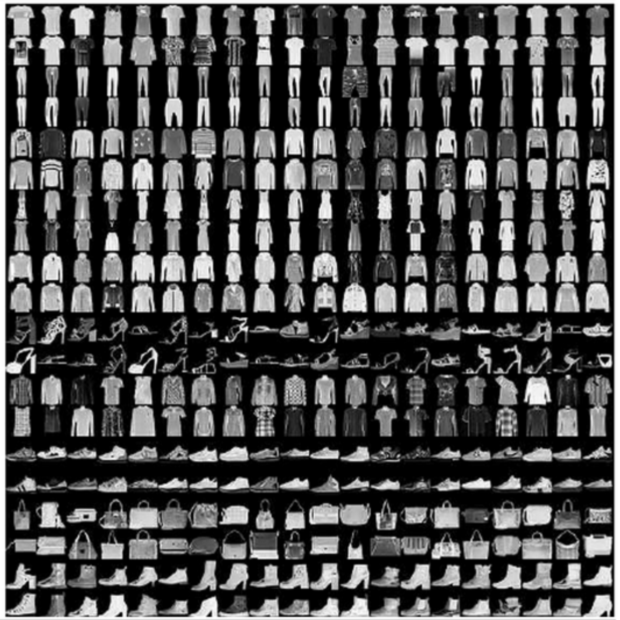
Code session – Pytorch basics

In this section, we'll work through a hands-on task together to get a practical introduction to deep learning and PyTorch coding.

FashionMNIST Image Classification

In this task, we'll classify clothing images from the FashionMNIST dataset.
We'll start with a basic fully connected neural network and then try a CNN to see how different architectures impact performance.

Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	



Content

1 Introduction to PyTorch basics

2 Setup the colab environment

3 Resources – Hugging-face, tensorboard, W&B

Hugging Face

Hugging face is something you need to know in the era of transformers.

1. Transformers: A widely known python package providing state-of-the-art implementations of popular transformer based models such as VIT, BERT, GPT (Yes LLM!), and many others.
2. Model Hub: A place where users share pretrained models. Especially pretrained large language models.
3. Datasets: A library providing a wide range of datasets for different machine learning tasks.

Hub

Host Git-based models, datasets and Spaces on the Hugging Face Hub.

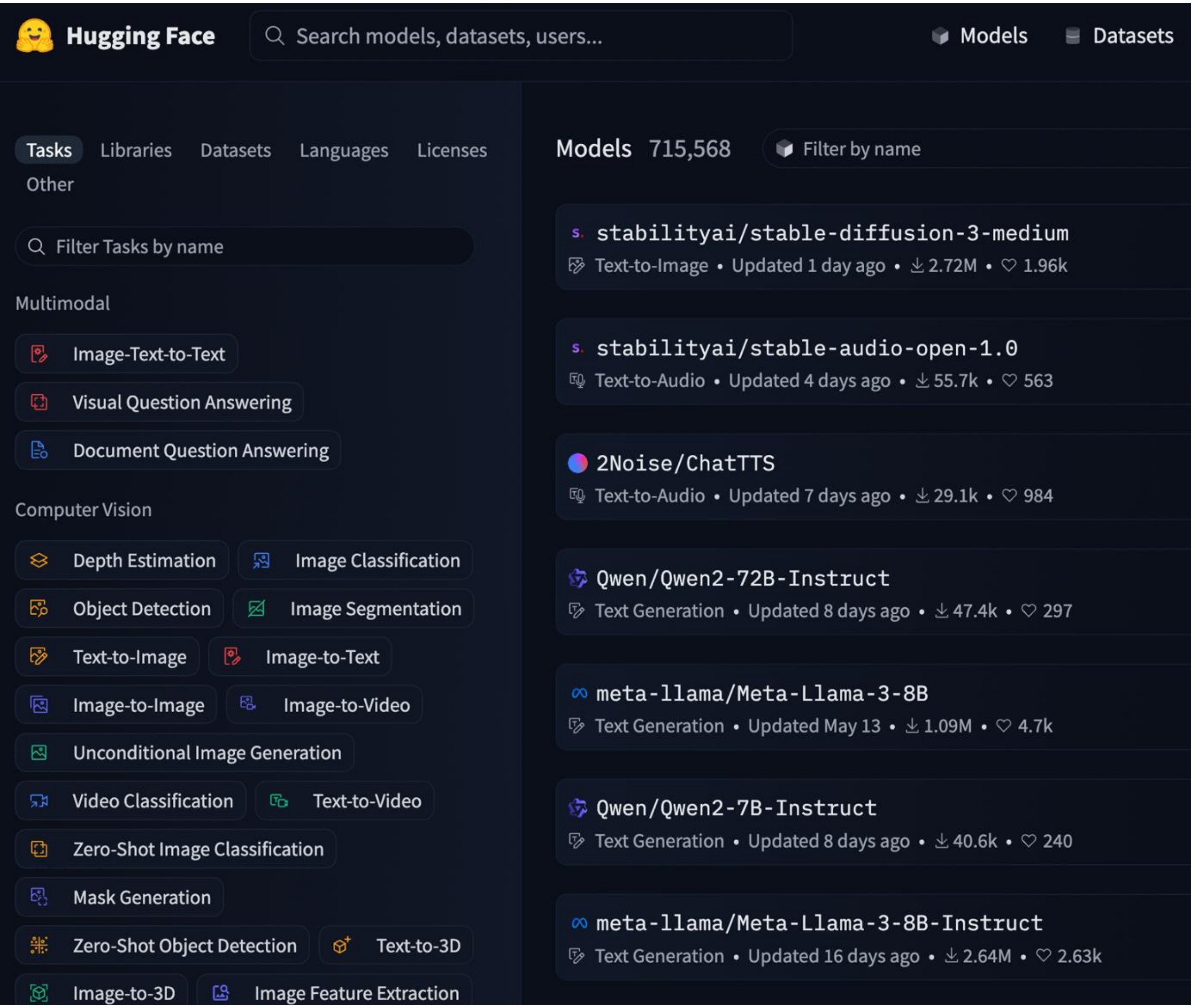
Datasets

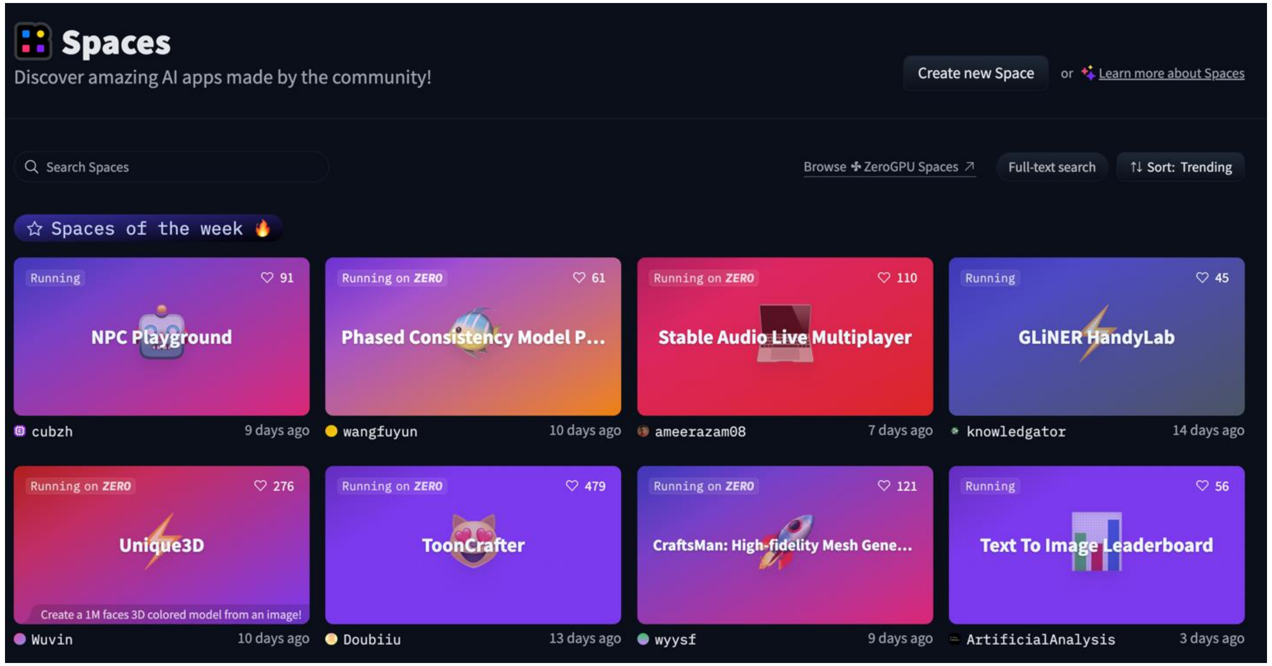
Access and share datasets for computer vision, audio, and NLP tasks.

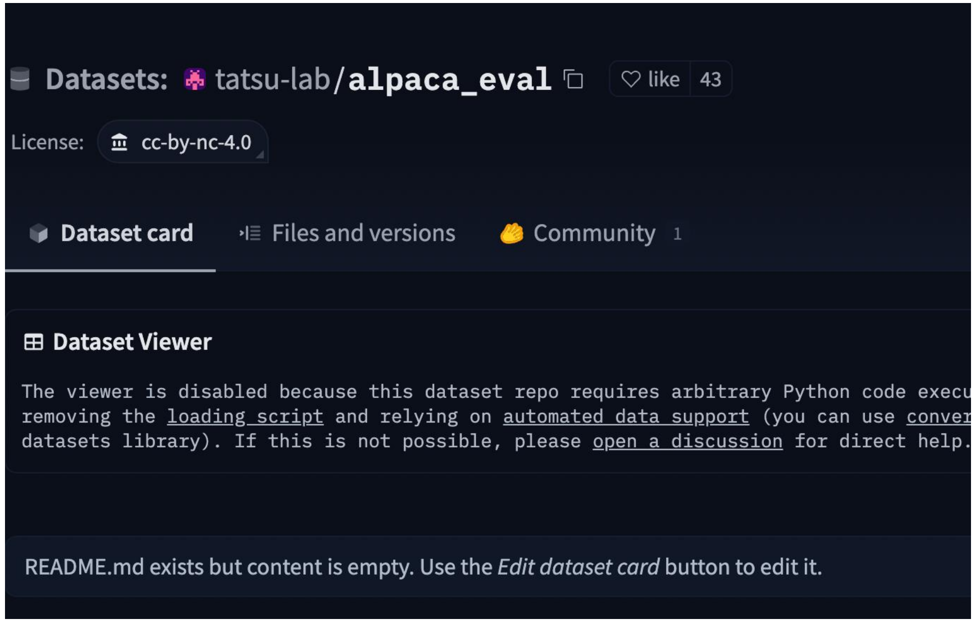
Transformers

State-of-the-art ML for Pytorch, TensorFlow, and JAX.

Hugging Face

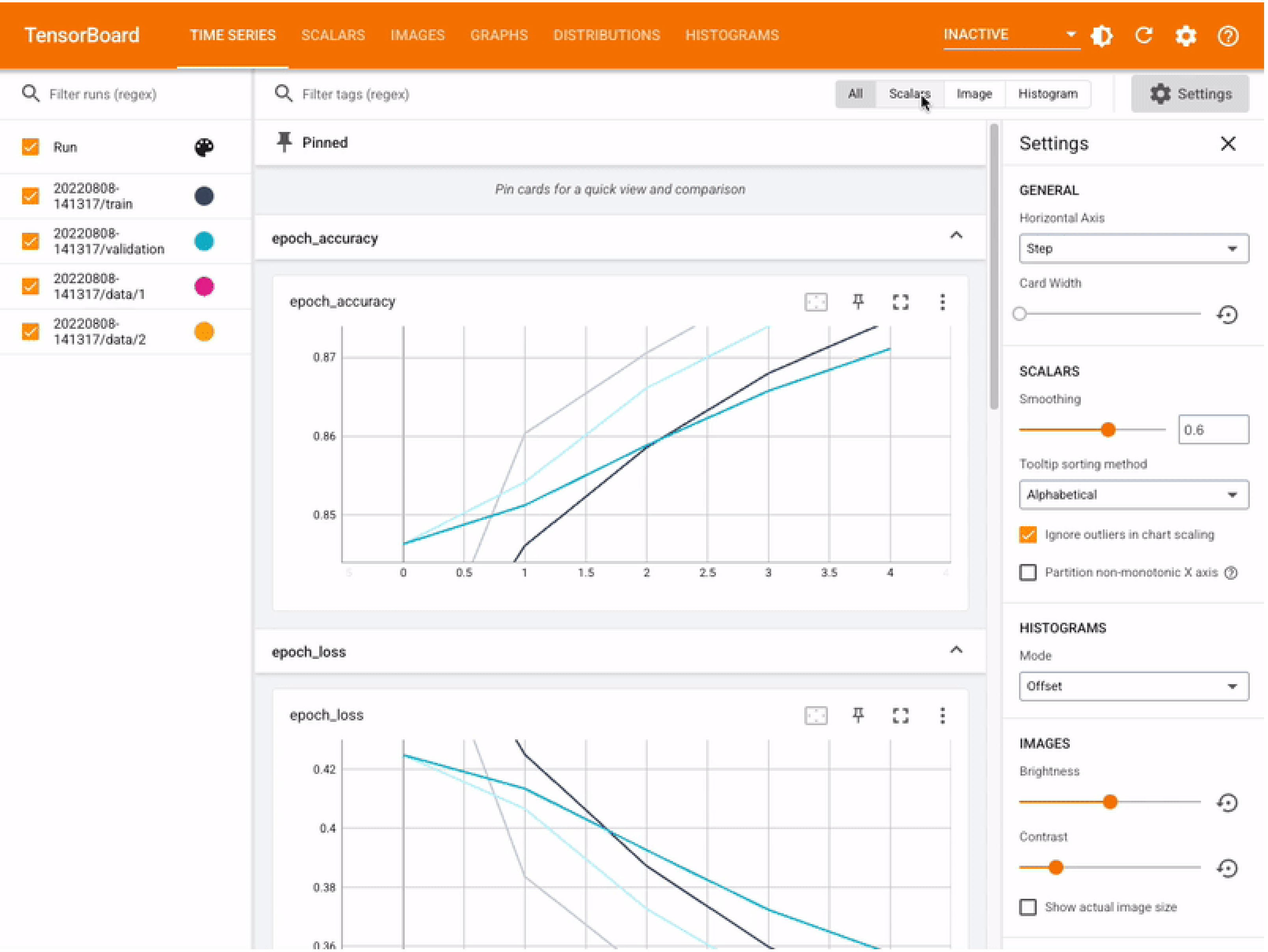






Code session – Hugging face and LLM

TensorBoard



- TensorBoard is an interactive visualization tool. It is used to monitoring the training and testing process and select hyper parameters.
- It offers more flexibility than tqdm.

Weights and bias