Exploring Tooling with Weights and Biases

Similar to tensorboard, weights and biases is an application that tracks all your training metrics, and performs visualizations for you. This tool allows you to cleanly sort, organize, and visualize your experiments. In this notebook, we will go through an example of how to use wandb.ai and have you practice.

1. Make an account at [https://wandb.ai/site (https://wandb.ai/site)](https://wandb.ai/site)
2. pip install wandb
3. wandb login
4. After step 3, please paste your wandb API key

In [ ]:

```
!wget https://raw.githubusercontent.com/Berkeley-CS182/cs182fa23_public/main/q_wand
!pip install wandb
```

In [1]:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import wandb
from architectures import BasicConvNet, ResNet18, MLP
```

First try the example provided by wandb

In [2]:
```python
import wandb
import random

# start a new wandb run to track this script
wandb.init(
    # set the wandb project where this run will be logged
    project="my-awesome-project",

    # track hyperparameters and run metadata
    config={
    "learning_rate": 0.02,
    "architecture": "CNN",
    "dataset": "CIFAR-100",
    "epochs": 10,
    }
)

# simulate training
epochs = 10
offset = random.random() / 5
for epoch in range(2, epochs):
    acc = 1 - 2 ** -epoch - random.random() / epoch - offset
    loss = 2 ** -epoch + random.random() / epoch + offset

    # log metrics to wandb
    wandb.log({"acc": acc, "loss": loss})

# [optional] finish the wandb run, necessary in notebooks
wandb.finish()
```

```
Failed to detect the name of this notebook, you can set it manually with the WAND
B_NOTEBOOK_NAME environment variable to enable code saving.
wandb: Currently logged in as: mingzwhy. Use `wandb login --relogin` to force
relogin
```

Tracking run with wandb version 0.15.12

Run data is saved locally in
f:\new_gitee_code\berkeley_class\Deep_Learning\hw8\wandb\run-20231018_144542-ncz8rcp1

Syncing run **woven-grass-1 (https://wandb.ai/mingzwhy/my-awesome-project/runs/ncz8rcp1)** to Weights & Biases (https://wandb.ai/mingzwhy/my-awesome-project) (docs (https://wandb.me/run))

View project at https://wandb.ai/mingzwhy/my-awesome-project (https://wandb.ai/mingzwhy/my-awesome-project)

View run at https://wandb.ai/mingzwhy/my-awesome-project/runs/ncz8rcp1 (https://wandb.ai/mingzwhy/my-awesome-project/runs/ncz8rcp1)

Waiting for W&B process to finish... **(success).**

```
VBox(children=(Label(value='0.001 MB of 0.012 MB uploaded (0.000 MB deduped)\r'),
FloatProgress(value=0.115196…
```

Run history:

acc 

loss 

Run summary:

acc	0.82327
loss	0.09432

View run **woven-grass-1** at: [https://wandb.ai/mingzwhy/my-awesome-project/runs/ncz8rcp1 (https://wandb.ai/mingzwhy/my-awesome-project/runs/ncz8rcp1)](https://wandb.ai/mingzwhy/my-awesome-project/runs/ncz8rcp1)
Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: `.\wandb\run-20231018_144542-ncz8rcp1\logs`

Organizing wandb Projects

With each run, you will want to have a set of parameters associated with it. For example, I want to be able to log different hyperparameters that I am using, so let's clearly list them below

In [3]:
```
project = 'CS182 WANDB.AI Practice Notebok'
learning_rate = 0.01
epochs = 2
architecture ='CNN'
dataset = 'CIFAR-10'
batch_size = 64
momentum = 0.9
log_freq = 20
print_freq = 200
cuda = torch.cuda.is_available()
device = torch.device("cuda" if cuda else "cpu")
```

Initializing the Run

In [4]:
```
wandb.init(
    # set the wandb project where this run will be logged
    project=project,

    # track hyperparameters and run metadata
    config={
    "learning_rate": learning_rate,
    "architecture": architecture,
    "dataset": dataset,
    "epochs": epochs,
    "batch_size": batch_size,
    "momentum": momentum
    }
)
```

Tracking run with wandb version 0.15.12

Run data is saved locally in
`f:\new_gitee_code\berkeley_class\Deep_Learning\hw8\wandb\run-20231018_145146-erzbsuxc`

Syncing run **pretty-universe-1 (https://wandb.ai/mingzwhy/CS182%20WANDB.AI%20Practice%20Notebok/runs/erzbsu** to Weights & Biases (https://wandb.ai/mingzwhy/CS182%20WANDB.AI%20Practice%20Notebok) (docs (https://wandb.me/run))

View project at https://wandb.ai/mingzwhy/CS182%20WANDB.AI%20Practice%20Notebok (https://wandb.ai/mingzwhy/CS182%20WANDB.AI%20Practice%20Notebok)

View run at https://wandb.ai/mingzwhy/CS182%20WANDB.AI%20Practice%20Notebok/runs/erzbsuxc (https://wandb.ai/mingzwhy/CS182%20WANDB.AI%20Practice%20Notebok/runs/erzbsuxc)

Out[4]: Display W&B run

From here on, we have some standard CIFAR training definitions.

In [6]:
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./../cifar-10/', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./../cifar-10/', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [7]:
```
class Net(nn.Module):
    def __init__(self):
        super().__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)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

In [8]:
```
net = Net()
```

In [9]:
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=learning_rate, momentum=momentum)
```

Training with wandb

As you can see, similar to tensorboard, each gradient step we will want to log the accuracy and loss. See below for an example.

In [10]:
```
for epoch in range(epochs):  # loop over the dataset multiple times
    running_loss = 0.0
    running_acc = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        accuracy = torch.mean((torch.argmax(outputs, dim=1) == labels).float()).iter

        # print statistics
        running_acc += accuracy
        running_loss += loss.item()
        if i % log_freq == log_freq - 1:
            wandb.log({'accuracy': accuracy, 'loss': loss.item()})

        if i % print_freq == print_freq - 1:    # print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / print_freq:.5f}
            running_loss = 0.0
            running_acc = 0.0
```

```
[1,   200] loss: 2.25610 accuracy: 15.78125
[1,   400] loss: 1.90479 accuracy: 29.90625
[1,   600] loss: 1.69474 accuracy: 37.71875
[2,   200] loss: 1.48649 accuracy: 45.19531
[2,   400] loss: 1.43179 accuracy: 48.33594
[2,   600] loss: 1.38526 accuracy: 50.20312
```

After we are done with this run, we will want to call `wandb.finish()`

In [11]:
```python
wandb.finish()
```

Waiting for W&B process to finish... **(success).**

```
VBox(children=(Label(value='0.001 MB of 0.001 MB uploaded (0.000 MB deduped)\r'),
FloatProgress(value=1.0, max…
```

Run history:

accuracy	
loss	

Run summary:

accuracy	56.25
loss	1.31502

View run **pretty-universe-1** at:
[https://wandb.ai/mingzwhy/CS182%20WANDB.AI%20Practice%20Notebok/runs/erzbsuxc (https://wandb.ai/mingzwhy/CS182%20WANDB.AI%20Practice%20Notebok/runs/erzbsuxc)](https://wandb.ai/mingzwhy/CS182%20WANDB.AI%20Practice%20Notebok/runs/erzbsuxc)
Synced 6 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: `.\wandb\run-20231018_145146-erzbsuxc\logs`

Your Task

We will be once again building classifiers for the CIFAR-10. There are various architectures set up for you to use in the architectures.py file. Using wandb, please search through 10 different hyperparameter configurations. Examples of choices include: learning rate, batch size, architecture, optimization algorithm, etc. Please submit the hyperparameters that result in the highest accuracies for this classification task. Please then explore wandb for all the visualization that you may need. In addition, feel free to run as many epochs as you like.

In [ ]:
```python
def run(params):
    raise NotImplementedError
```

This software/tutorial is based on PyTorch, an open-source project available at
https://github.com/pytorch/tutorials/ (https://github.com/pytorch/tutorials/)

In [13]:
```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import wandb
from architectures import BasicConvNet, ResNet18, MLP
from torch.utils.tensorboard import SummaryWriter
from tqdm import tqdm
from torch.utils.data import DataLoader
```

In [12]:
```python
device = torch.device("cuda" if cuda else "cpu")

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

trainset = torchvision.datasets.CIFAR10(root='./../cifar-10/', train=True,
                                        download=True, transform=transform)
testset = torchvision.datasets.CIFAR10(root='./../cifar-10/', train=False,
                                       download=True, transform=transform)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [18]:
```
def get_optimizer(params, optim_type, lr):
    if optim_type == "sgd":
        optimizer = optim.SGD(params, lr=lr)
    elif optim_type == "adam":
        optimizer = optim.Adam(params, lr=lr)
    else:
        raise ValueError(optim_type)

    return optimizer

def get_model(model_type):
    if model_type == "basicconvnet":
        model = BasicConvNet()
    elif model_type == "resnet18":
        model = ResNet18()
    elif model_type == "mlp":
        model = MLP()
    else:
        raise ValueError(model_type)

    return model

def get_criterion(loss_type):
    if(loss_type == "mse"):
        criterion = nn.MSELoss()
    elif(loss_type == "cross"):
        criterion = nn.CrossEntropyLoss()
    else:
        raise ValueError(loss_type)

    return criterion
```

In [19]:
```
def train(dataloader, model, loss_fn, optimizer, epoch):
    size = len(dataloader.dataset)
    num_batch = len(dataloader)
    model.train()

    total_loss = 0
    correct = 0

    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        pred = model(X)
        loss = loss_fn(pred, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        correct += (pred.argmax(1) == y).type(torch.float).sum().item()

        if(batch % 100 == 0):
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f} [{current:>5d} / {size:>5d}]")

    avg_loss = total_loss / num_batch
    correct /= size

    # write into wandb
    wandb.log({'train accuracy': correct, 'train loss': avg_loss})

    print(f"Train Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {avg_loss:>8


def test(dataloader, model, loss_fn, epoch):

    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()

    test_loss = 0
    correct = 0.1
    with torch.no_grad():
        for batch, (X, y) in enumerate(dataloader):
            X, y = X.cuda(), y.cuda()
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    test_loss /= num_batches
    correct /= size

    # write into wandb
    wandb.log({'test accuracy': correct, 'test loss': test_loss})

    print(f"Evaluation Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_l
```

In [20]:
```
def run_training(trainset, testset, hyperparameters, log_dir = "logs"):

    print("---------------------------config------------------------------")
    print(hyperparameters)
    print("---------------------------------------------------------------")

    name = ""
    for i, key in enumerate(hyperparameters.keys()):
        value = hyperparameters[key]
        if i != (len(hyperparameters.keys()) - 1):
            item = key + "_" + str(value) + "_"
        else:
            item = key + "_" + str(value)
        name = name + item

    model_type = hyperparameters['model']
    model = get_model(model_type)
    loss_type = hyperparameters['loss_fn']
    criterion = get_criterion(loss_type)
    learning_rate = hyperparameters['lr']
    optim_type = hyperparameters['optimizer']
    optimizer = get_optimizer(model.parameters(), optim_type, lr=learning_rate)
    batch_size = hyperparameters['batch_size']
    num_epochs = hyperparameters['epochs']

    # build train data loader
    trainloader = DataLoader(trainset, batch_size=batch_size, shuffle=True)
    # build test data loader
    testloader = DataLoader(testset, batch_size=batch_size, shuffle=False)

    # create a wandb project

    wandb.init(
        # set the wandb project where this run will be logged
        project = name,

        # track hyperparameters and run metadata
        config={
        "learning_rate": learning_rate,
        "architecture": model_type,
        "dataset": 'CIFAR-10',
        "epochs": num_epochs,
        "batch_size": batch_size,
        }
    )

    print(f"log will be written to project {name}")

    model.cuda()

    for t in range(num_epochs):
        print(f"Epoch {t+1}\n-------------------------------")
        train(trainloader, model, criterion, optimizer, t+1)
        test(testloader, model, criterion, t+1)

    wandb.finish()
```

Type *Markdown* and LaTeX: α^2

In [22]:
```
hyperparameters1 = {
    "model" : "basicconvnet",
    "lr" : 0.0001,
    "loss_fn" : "cross",
    "optimizer" : "adam",
    "epochs" : 3,
    "batch_size" : 16
}
```

In [23]:
```
run_training(trainset, testset, hyperparameters1)
```

```
--------------------------config----------------------------
{'model': 'basicconvnet', 'lr': 0.0001, 'loss_fn': 'cross', 'optimizer': 'ada
m', 'epochs': 3, 'batch_size': 16}
------------------------------------------------------------
```

Tracking run with wandb version 0.15.12

Run data is saved locally in
`f:\new_gitee_code\berkeley_class\Deep_Learning\hw8\wandb\run-20231018_150815-781t4xy1`

Syncing run **swept-cloud-1**
(https://wandb.ai/mingzwhy/model_basicconvnet_lr_0.0001_loss_fn_cross_optimizer
to Weights & Biases
(https://wandb.ai/mingzwhy/model_basicconvnet_lr_0.0001_loss_fn_cross_optimizer_adam
(docs (https://wandb.me/run))

View project at

In [ ]: