# Weights and Biases

## Contents



Weights & Biases is a powerful tool designed to help machine learning practitioners track their experiments, visualize data, and optimize models more effectively.

## Reasons to use Weights & Biases

- **Easy integration** ➡️ Easy integration with PyTorch, TensorFlow, and Keras, as well as with other tools like Jupyter Notebooks (Minimal code changing).
- **Comprehensive Experiment Tracking** ➡️ Keep detailed logs of every experiment including code version, metrics, hyperparameters, output files, and

automatically organizes your experiment history, making it easy to compare and reproduce results.

- **Rich Visualizations** ➡️ Generate rich visual reports, including plots, images, audios, etc.
- **Real-time Monitoring** ➡️ View live updates of your training and validation metrics.
- **Artifact tracking** ➡️ Version and track models and other files as part of your pipeline.

# Tutorial: Using Weights & Biases with PyTorch

In this tutorial, we will cover how to integrate Weights & Biases (W&B) into a PyTorch project to track experiment metrics, visualize data, and log hyperparameters.

## 1. Setting Up Weights & Biases

First, install Weights & Biases in your environment:

```
pip install wandb
```

Log in to your W&B account using:

```
wandb login
```

This will prompt you to provide an API key that you can obtain from your W&B account page.

# 2. Integrating W&B in a PyTorch Project

Let's say we have a simple PyTorch project for training a neural network on the MNIST dataset. We'll integrate W&B step by step.

1. **Import wandb and Initialize**:

   ```
   import wandb
   import torch
   import torch.nn as nn
   import torch.optim as optim
   from torchvision import datasets, tr

   # Initialize W&B
   wandb.init(project="project-name")
   ```

2. **Define the Model and Training Loop**:

   Create a simple neural network and integrate W&B to log important metrics.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28 * 28
        self.fc2 = nn.Linear(128, 10

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = Net()
optimizer = optim.SGD(model.paramete
criterion = nn.CrossEntropyLoss()
```

3. **Add W&B Logging to the Training Loop**:

```
def train(model, device, train_loade
    model.train()
    for batch_idx, (data, target) in
        data, target = data.to(devic
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, tar
        loss.backward()
        optimizer.step()

        # Log metrics to W&B
        wandb.log({"loss": loss.item

        if batch_idx % 100 == 0:
            print(f'Train Epoch: {ep

# Run the training
wandb.watch(model, log="all")
device = torch.device("cuda" if torc
train_loader = torch.utils.data.Data
    datasets.MNIST('./data', train=T
                   transform=transfo
    batch_size=64, shuffle=True)

for epoch in range(1, 6):
    train(model, device, train_loade
``
```

4. **Saving Artifacts and Results**:

   You can also save model checkpoints:

```
torch.save(model.state_dict(), "mode
wandb.save("model.pth")
```

# Tutorial: Using Weights & Biases with PyTorch

# Lightning

PyTorch Lightning is a high-level framework for PyTorch that abstracts away much of the boilerplate code. Integrating W&B with PyTorch Lightning is even more straightforward.

## 1. Install PyTorch Lightning and W&B

```
pip install lightning
pip install wandb
```

## 2. Integrating W&B in a PyTorch Lightning Project

1. **Import Required Libraries**:

```
import lightning as L
from lightning.pytorch.loggers impor
from lightning.pytorch.callbacks imp
import torch
import torch.nn.functional as F
from torch.utils.data import DataLoa
from torchvision import datasets, tr
```

2. **Define the Model**:

```
class LitModel(pl.LightningModule):
    def __init__(self):
        super(LitModel, self).__init
        self.fc1 = torch.nn.Linear(2
        self.fc2 = torch.nn.Linear(1

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

    def training_step(self, batch, b
        x, y = batch
        y_hat = self(x)
        loss = F.cross_entropy(y_hat
        self.log('train_loss', loss)
        return loss

    def configure_optimizers(self):
        return torch.optim.SGD(self.
```

3. **Set Up Data Loaders**:

```
transform = transforms.ToTensor()
mnist_train = datasets.MNIST('', tra
train_loader = DataLoader(mnist_trai
```

4. **Initialize the W&B Logger**:

```
wandb_logger = WandbLogger(project='
```

5. **Train the Model with W&B Integration**:

```
model = LitModel()
trainer = pl.Trainer(max_epochs=5, l
trainer.fit(model, train_loader)
```

- **Logging Artifacts**: If you want to log the model weights into W&B, you can do so by adding the following callback to the trainer:

```
from lightning.pytorch.callbacks

checkpoint_callback = ModelCheck
    dirpath='checkpoints/',
    filename='model-{epoch:02d}-
    monitor='val_loss',
    save_top_k=1,
    mode='min'
)
```

  Then, pass the callback to the trainer.

The PyTorch Lightning integration with W&B allows for even more automation. Metrics will automatically be logged, and the training process will be visualized in the W&B dashboard.