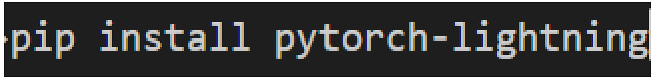
PyTorch Lightning

PyTorch Lightning

Introduction

- Python package
- framework built on top of PyTorch
- claims „making coding complex networks simple“
- nice features
  - logging of training / validation metrics
  - creating checkpoints
  - early stopping
  - training on multiple GPUs, TPUs, and CPUs

- Installation:



```
pip install pytorch-lightning
```

PyTorch Lightning

Introduction

- requires a class with four functions:
  - __init__()
  - forward()
  - configure_optimizers()
  - training_step()
- optional functions
  - prepare_data()
  - validation_step()
  - test_step()
  - predict_step()

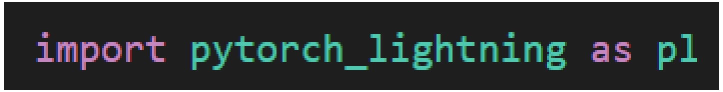
PyTorch Lightning

Comparison PyTorch vs. PyTorch Lightning

your package imports

PyTorch

your package imports

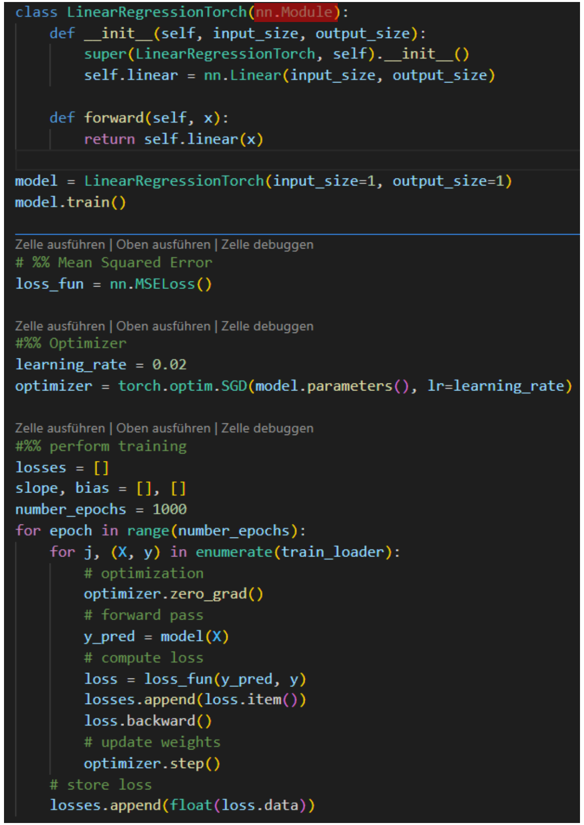


```
import pytorch_lightning as pl
```

PyTorch Lightning

PyTorch Lightning

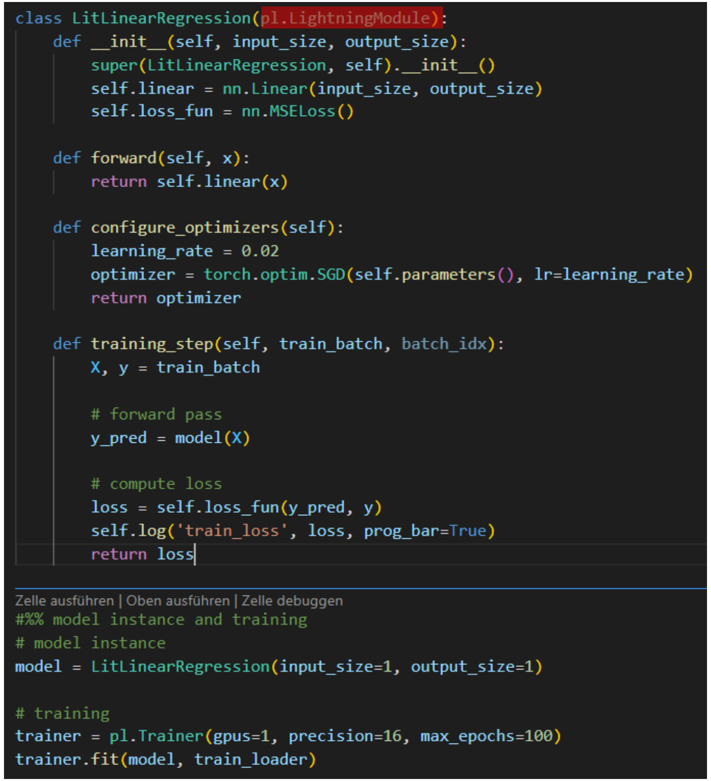
Comparison PyTorch vs. PyTorch Lightning



```
class LinearRegressionTorch(nn.Module):
    def __init__(self, input_size, output_size):
        super(LinearRegressionTorch, self).__init__()
        self.linear = nn.Linear(input_size, output_size)

    def forward(self, x):
        return self.linear(x)

model = LinearRegressionTorch(input_size=1, output_size=1)
model.train()

Zelle ausführen | Oben ausführen | Zelle debuggen
# %% Mean Squared Error
loss_fun = nn.MSELoss()

Zelle ausführen | Oben ausführen | Zelle debuggen
#%% Optimizer
learning_rate = 0.02
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

Zelle ausführen | Oben ausführen | Zelle debuggen
#%% perform training
losses = []
slope, bias = [], []
number_epochs = 1000
for epoch in range(number_epochs):
    for j, (X, y) in enumerate(train_loader):
        # optimization
        optimizer.zero_grad()
        # forward pass
        y_pred = model(X)
        # compute loss
        loss = loss_fun(y_pred, y)
        losses.append(loss.item())
        loss.backward()
        # update weights
        optimizer.step()
    # store loss
    losses.append(float(loss.data))
```

PyTorch

Class Inheritance



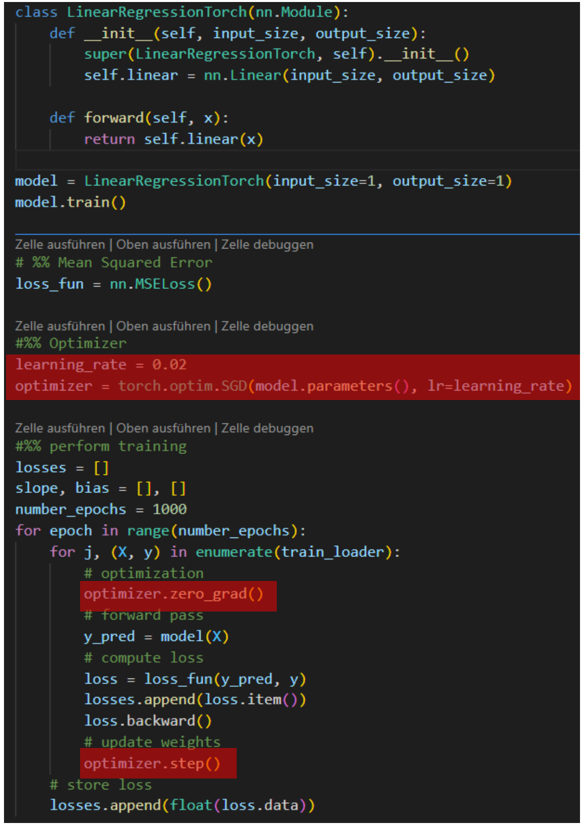
```
class LitLinearRegression(pl.LightningModule):
    def __init__(self, input_size, output_size):
        super(LitLinearRegression, self).__init__()
        self.linear = nn.Linear(input_size, output_size)
        self.loss_fun = nn.MSELoss()

    def forward(self, x):
        return self.linear(x)

    def configure_optimizers(self):
        learning_rate = 0.02
        optimizer = torch.optim.SGD(self.parameters(), lr=learning_rate)
        return optimizer

    def training_step(self, train_batch, batch_idx):
        X, y = train_batch

        # forward pass
        y_pred = model(X)

        # compute loss
        loss = self.loss_fun(y_pred, y)
        self.log('train_loss', loss, prog_bar=True)
        return loss

Zelle ausführen | Oben ausführen | Zelle debuggen
#%% model instance and training
# model instance
model = LitLinearRegression(input_size=1, output_size=1)

# training
trainer = pl.Trainer(gpus=1, precision=16, max_epochs=100)
trainer.fit(model, train_loader)
```

PyTorch Lightning

PyTorch Lightning

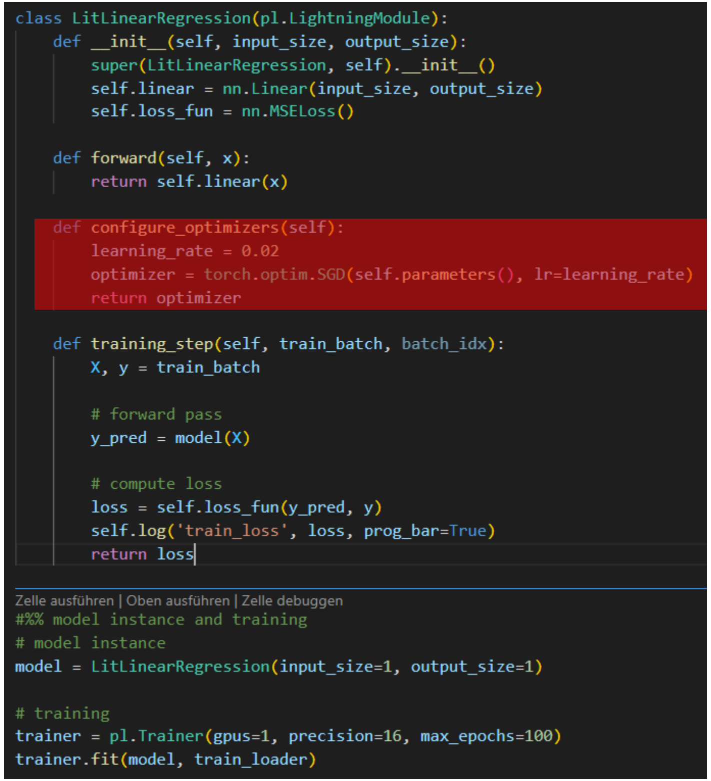
Comparison PyTorch vs. PyTorch Lightning

Optimizer



```
class LinearRegressionTorch(nn.Module):
    def __init__(self, input_size, output_size):
        super(LinearRegressionTorch, self).__init__()
        self.linear = nn.Linear(input_size, output_size)

    def forward(self, x):
        return self.linear(x)

model = LinearRegressionTorch(input_size=1, output_size=1)
model.train()

Zelle ausführen | Oben ausführen | Zelle debuggen
# %% Mean Squared Error
loss_fun = nn.MSELoss()

Zelle ausführen | Oben ausführen | Zelle debuggen
#%% Optimizer
learning_rate = 0.02
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

Zelle ausführen | Oben ausführen | Zelle debuggen
#%% perform training
losses = []
slope, bias = [], []
number_epochs = 1000
for epoch in range(number_epochs):
    for j, (X, y) in enumerate(train_loader):
        # optimization
        optimizer.zero_grad()
        # forward pass
        y_pred = model(X)
        # compute loss
        loss = loss_fun(y_pred, y)
        losses.append(loss.item())
        loss.backward()
        # update weights
        optimizer.step()
    # store loss
    losses.append(float(loss.data))
```

PyTorch



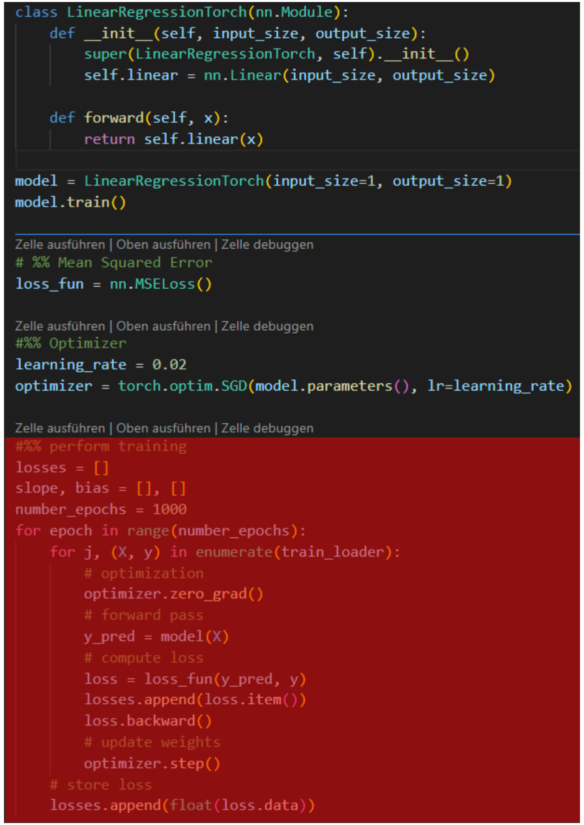
```
class LitLinearRegression(pl.LightningModule):
    def __init__(self, input_size, output_size):
        super(LitLinearRegression, self).__init__()
        self.linear = nn.Linear(input_size, output_size)
        self.loss_fun = nn.MSELoss()

    def forward(self, x):
        return self.linear(x)

    def configure_optimizers(self):
        learning_rate = 0.02
        optimizer = torch.optim.SGD(self.parameters(), lr=learning_rate)
        return optimizer

    def training_step(self, train_batch, batch_idx):
        X, y = train_batch

        # forward pass
        y_pred = model(X)

        # compute loss
        loss = self.loss_fun(y_pred, y)
        self.log('train_loss', loss, prog_bar=True)
        return loss

Zelle ausführen | Oben ausführen | Zelle debuggen
#%% model instance and training
# model instance
model = LitLinearRegression(input_size=1, output_size=1)

# training
trainer = pl.Trainer(gpus=1, precision=16, max_epochs=100)
trainer.fit(model, train_loader)
```

PyTorch Lightning

PyTorch Lightning

Comparison PyTorch vs. PyTorch Lightning

Training



```
class LinearRegressionTorch(nn.Module):
    def __init__(self, input_size, output_size):
        super(LinearRegressionTorch, self).__init__()
        self.linear = nn.Linear(input_size, output_size)

    def forward(self, x):
        return self.linear(x)

model = LinearRegressionTorch(input_size=1, output_size=1)
model.train()
```

Zelle ausführen | Oben ausführen | Zelle debuggen

```
# %% Mean Squared Error
loss_fun = nn.MSELoss()
```

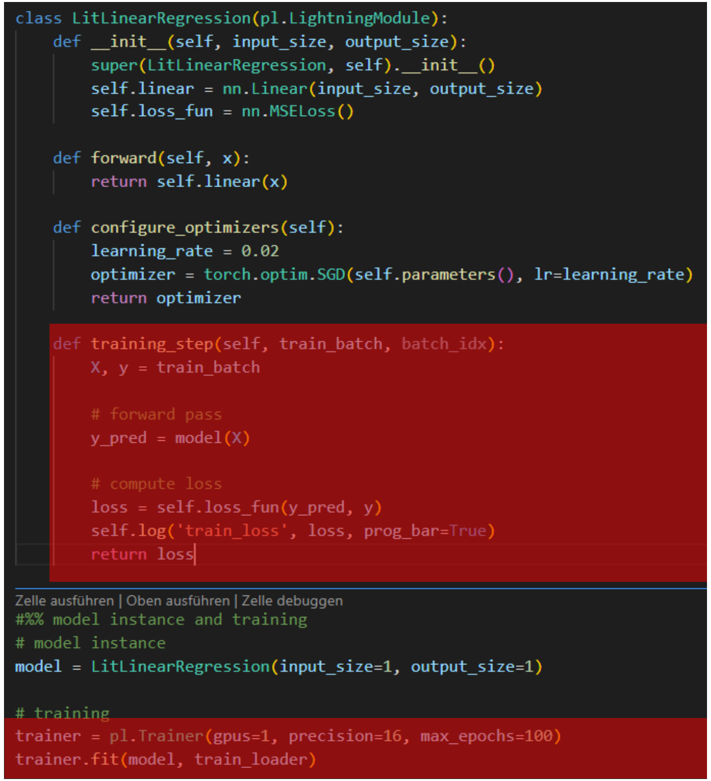
Zelle ausführen | Oben ausführen | Zelle debuggen

```
#%% Optimizer
learning_rate = 0.02
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

Zelle ausführen | Oben ausführen | Zelle debuggen

```
#%% perform training
losses = []
slope, bias = [], []
number_epochs = 1000
for epoch in range(number_epochs):
    for j, (X, y) in enumerate(train_loader):
        # optimization
        optimizer.zero_grad()
        # forward pass
        y_pred = model(X)
        # compute loss
        loss = loss_fun(y_pred, y)
        losses.append(loss.item())
        loss.backward()
        # update weights
        optimizer.step()
    # store loss
    losses.append(float(loss.data))
```

PyTorch



```
class LitLinearRegression(pl.LightningModule):
    def __init__(self, input_size, output_size):
        super(LitLinearRegression, self).__init__()
        self.linear = nn.Linear(input_size, output_size)
        self.loss_fun = nn.MSELoss()

    def forward(self, x):
        return self.linear(x)

    def configure_optimizers(self):
        learning_rate = 0.02
        optimizer = torch.optim.SGD(self.parameters(), lr=learning_rate)
        return optimizer

    def training_step(self, train_batch, batch_idx):
        X, y = train_batch

        # forward pass
        y_pred = model(X)

        # compute loss
        loss = self.loss_fun(y_pred, y)
        self.log('train_loss', loss, prog_bar=True)
        return loss
```

Zelle ausführen | Oben ausführen | Zelle debuggen

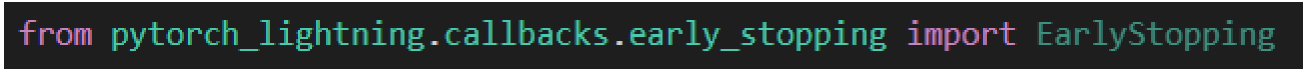
```
#%% model instance and training
# model instance
model = LitLinearRegression(input_size=1, output_size=1)

# training
trainer = pl.Trainer(gpus=1, precision=16, max_epochs=100)
trainer.fit(model, train_loader)
```

PyTorch Lightning

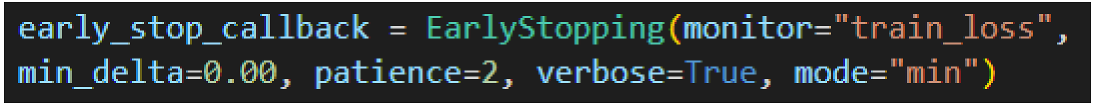
Early Stopping

Early Stopping

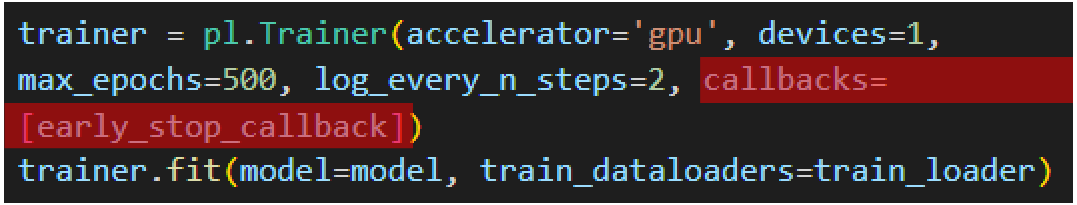
Additional Features: Early Stopping



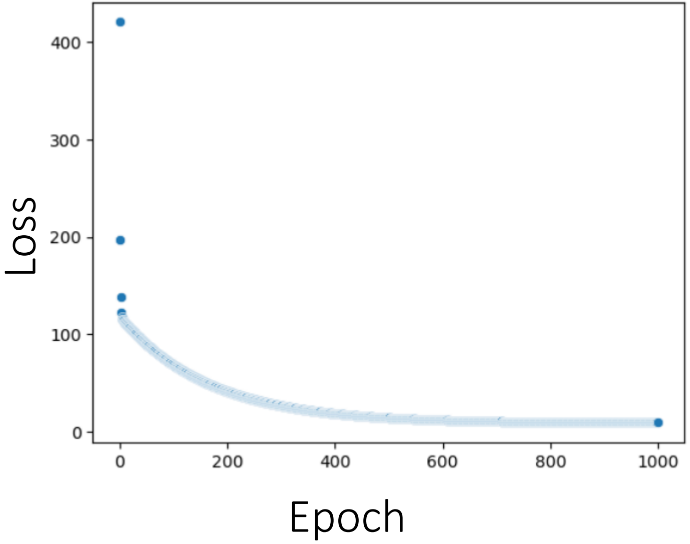
```
from pytorch_lightning.callbacks.early_stopping import EarlyStopping
```



```
early_stop_callback = EarlyStopping(monitor="train_loss",
min_delta=0.00, patience=2, verbose=True, mode="min")
```



```
trainer = pl.Trainer(accelerator='gpu', devices=1,
max_epochs=500, log_every_n_steps=2, callbacks=
[early_stop_callback])
trainer.fit(model=model, train_dataloaders=train_loader)
```


Loss
Epoch