In [1]:

```
%matplotlib inline
import torch
from d2l import torch as d2l
```

In [3]:

```
class LinearRegressionScratch(d2l.Module):  #@save
    """The linear regression model implemented from scratch."""
    def __init__(self, num_inputs, lr, sigma=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.w = torch.normal(0, sigma, (num_inputs, 1), requires_grad=True)
        self.b = torch.zeros(1, requires_grad=True)
```

In [4]:

```
@d2l.add_to_class(LinearRegressionScratch)  #@save
def forward(self, X):
    return torch.matmul(X, self.w) + self.b
```

In [5]:

```
@d2l.add_to_class(LinearRegressionScratch)  #@save
def loss(self, y_hat, y):
    l = (y_hat - y) ** 2 / 2
    return l.mean()
```

In [6]:

```
class SGD(d2l.HyperParameters):  #@save
    """Minibatch SGD."""
    def __init__(self, params, lr):
        self.save_hyperparameters()

    def step(self):
        for param in self.params:
            param -= self.lr * param.grad

    def zero_grad(self):
        for param in self.params:
            if param.grad is not None:
                param.grad.zero_()
```

In [7]:

```
@d2l.add_to_class(LinearRegressionScratch)  #@save
def configure_optimizers(self):
    return SGD([self.w, self.b], self.lr)
```

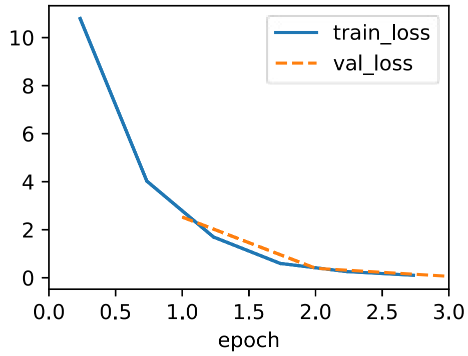
In [8]:

```
@d2l.add_to_class(d2l.Trainer)  #@save
def prepare_batch(self, batch):
    return batch

@d2l.add_to_class(d2l.Trainer)  #@save
def fit_epoch(self):
    self.model.train()
    for batch in self.train_dataloader:
        loss = self.model.training_step(self.prepare_batch(batch))
        self.optim.zero_grad()
        with torch.no_grad():
            loss.backward()
            if self.gradient_clip_val > 0: #나중
                self.clip_gradients(self.gradient_clip_val, self.model)
            self.optim.step()
        self.train_batch_idx += 1
    if self.val_dataloader is None:
        return
    self.model.eval()
    for batch in self.val_dataloader:
        with torch.no_grad():
            self.model.validation_step(self.prepare_batch(batch))
        self.val_batch_idx += 1
```

In [9]:

```
model = LinearRegressionScratch(2, lr=0.03)
data = d2l.SyntheticRegressionData(w=torch.tensor([2, -3.4]), b=4.2)
trainer = d2l.Trainer(max_epochs=3)
trainer.fit(model, data)
```



In [10]:

```
with torch.no_grad():
    print(f'error in estimating w: {data.w - model.w.reshape(data.w.shape)}')
    print(f'error in estimating b: {data.b - model.b}')
```

```
error in estimating w: tensor([ 0.1526, -0.2056])
error in estimating b: tensor([0.2406])
```

Discussion: Before diving into the main content, I had maintained the stance of not typing out comments, which is why I didn't include #@save. However, since it kept appearing repeatedly, I decided to look it up and learned that it indicates sections of code that can be reused multiple times. Thus, unlike other comments, I decided to include it. This section focused on implementing a linear regression model and training the model's weights and bias using SGD. I'm curious how the learning rate (lr) in the SGD class will affect the final result, as this hasn't been covered yet. The loss function used is MSE, which I was happy to see because it's a function I learned about in computational mathematics before. During the training process, I came across the new concept of an "epoch." After the data was trained, the estimated parameters and the actual parameters were compared by printing out the error between them.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js