Concise Implementation of Linear Regression

Generating Data Sets

In [1]:

```
from mxnet import autograd, nd

num_inputs = 2
num_examples = 1000
true_w = nd.array([2, -3.4])
true_b = 4.2
features = nd.random.normal(scale=1, shape=(num_examples, num_inputs))
labels = nd.dot(features, true_w) + true_b
labels += nd.random.normal(scale=0.01, shape=labels.shape)
```

Reading Data

In [2]:

```
from mxnet.gluon import data as gdata

batch_size = 10
# Combine the features and labels of the training data
dataset = gdata.ArrayDataset(features, labels)
# Randomly reading mini-batches
data_iter = gdata.DataLoader(dataset, batch_size, shuffle=True)
```

Read a Data Batch

In [3]:

```
for X, y in data_iter:
    print(X, y)
    break
```

```
[[ 0.91097313 -1.1046128 ]
 [-0.23816614 -0.3334923 ]
 [-2.7786708   0.01066511]
 [ 0.05081175 -0.55327344]
 [-1.3000388   1.5999995 ]
 [-0.02302935  1.7881038 ]
 [-1.0592172  -0.47244707]
 [-1.4682877   0.5651783 ]
 [-0.70997626 -1.0089529 ]
 [ 1.1747298   1.1240152 ]]
<NDArray 10x2 @cpu(0)>
[ 9.7651005   4.8740754 -1.3962101   6.187774   -3.8292062 -1.9272817
  3.681892   -0.6382029   6.213706    2.720426 ]
<NDArray 10 @cpu(0)>
```

Define the Model

In [4]:

```
from mxnet.gluon import nn
net = nn.Sequential()
net.add(nn.Dense(1))
```

Initialize Model Parameters

- Initialize weight parameter by a normal distribution with a mean of 0 and standard deviation of 0.01.
- The bias parameter is initialized to zero by default.

In [5]:

```
from mxnet import init
net.initialize(init.Normal(sigma=0.01))
```

Define the Loss Function

In [6]:

```
from mxnet.gluon import loss as gloss
loss = gloss.L2Loss()  # The squared loss is also known as the L2 norm loss
```

Define the Optimization Algorithm

In [7]:

```
from mxnet import gluon
trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rate': 0.03})
```

Training

In [8]:

```
num_epochs = 3
for epoch in range(1, num_epochs + 1):
    for X, y in data_iter:
        with autograd.record():
            l = loss(net(X), y)
        l.backward()
        trainer.step(batch_size)
    l = loss(net(features), labels)
    print('epoch %d, loss: %f' % (epoch, l.mean().asnumpy()))
```

```
epoch 1, loss: 0.035035
epoch 2, loss: 0.000130
epoch 3, loss: 0.000049
```

Evaluate

In [9]:

```
w = net[0].weight.data()
print('Error in estimating w', true_w.reshape(w.shape) - w)
b = net[0].bias.data()
print('Error in estimating b', true_b - b)
```

```
Error in estimating w
[[ 0.00043857 -0.00051284]]
<NDArray 1x2 @cpu(0)>
Error in estimating b
[-4.1007996e-05]
<NDArray 1 @cpu(0)>
```