



Linear Regression Implementation from Scratch

Eng Teong Cheah

Contents

Linear Regression Implementation from Scratch

Linear Regression Implementation from Scratch



Linear Regression Implementation from Scratch



```
%matplotlib inline  
import d2l  
from mxnet import autograd, nd  
import random
```

Generating Data Sets

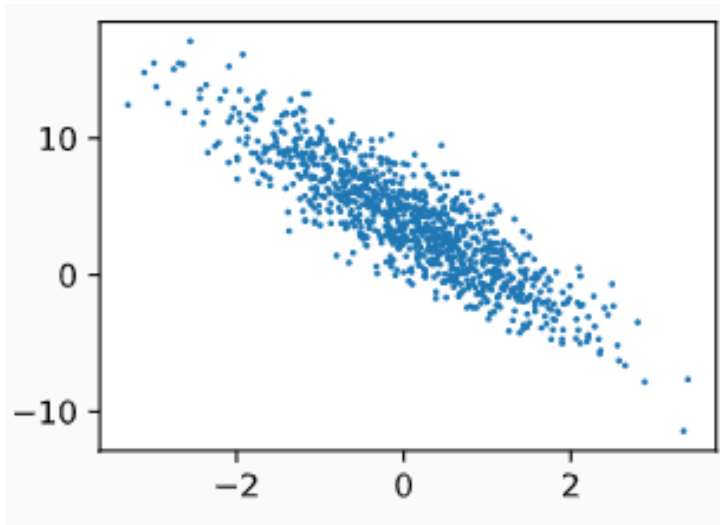


```
num_inputs = 2
num_examples = 1000
true_w = [2, -3.4]
true_b = 4.2
features = nd.random.normal(scale=1, shape=(num_examples, num_inputs))
labels = true_w[0] * features[:, 0] + true_w[1] * features[:, 1] +
true_b
labels += nd.random.normal(scale=0.01, shape=labels.shape)
```


Visualize the Second Feature and Label



```
display.set_matplotlib_formats('svg')  
plt.figure(figsize=(6, 3))  
plt.scatter(features[:, 1].asnumpy(), labels.asnumpy(),  
1);
```



Reading Data



```
def data_iter(batch_size, features, labels):
    num_examples = len(features)
    indices = list(range(num_examples))
    # The examples are read at random, in no particular order
    random.shuffle(indices)
    for i in range(0, num_examples, batch_size):
        j = nd.array(indices[i: min(i + batch_size, num_examples)])
        yield features.take(j), labels.take(j)
        # The "take" function will then return the corresponding element
        # based on the indices
```

Print a Small Data Batch

```
batch_size = 10

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

```
[[ 1.7130351 2.0944266 ] [-1.95122 -1.0635319 ]
 [ 0.22546557 0.8499169 ] [-1.6903632 -0.15907401]
 [ 0.85316896 1.0389048 ] [-0.17798574 1.410267
 [-2.152955 -1.3841926 ] [ 0.8605494 -0.22276537]
 [-0.33177793 -0.99060804] [ 2.584889 -1.1416703 ]]
<NDArray 10x2 @cpu(0)>
[ 0.5036066 3.9271483 1.7579273 1.353747 2.3738737 -
 4.593405 6.6843915 6.8792953 13.258079 ]
NDArray 10 @cpu(0)>
```


Initial Model Parameters



```
w = nd.random.normal(scale=0.01, shape=(2, 1))  
b = nd.zeros(shape=(1,))
```

Define the Linear Model



```
# Save to the d2l package.  
def linreg(X, w, b):  
    return nd.dot(X, w) + b
```

Define the Loss function



```
# Save to the d2l package.  
def squared_loss(y_hat, y):  
    return (y_hat - y.reshape(y_hat.shape)) ** 2 /  
2
```

Define the Optimization Algorithm



```
# Save to the d2l package.  
def sgd(params, lr, batch_size):  
    for param in params:  
        param[:] = param - lr * param.grad / batch_size
```

Training

```
lr = 0.03 # Learning rate
num_epochs = 3 # Number of iterations
net = linreg # Our fancy linear model
loss = squared_loss #  $0.5 (y-y')^2$ 

for epoch in range(num_epochs):
    # Assuming the number of examples can be divided by the batch size, all
    # the examples in the training data set are used once in one epoch
    # iteration. The features and tags of mini-batch examples are given by X
    # and y respectively
    for X, y in data_iter(batch_size, features, labels):
        with autograd.record():
            l = loss(net(X, w, b), y) # Minibatch loss in X and y
            l.backward() # Compute gradient on l with respect to [w,b]
            sgd([w, b], lr, batch_size) # Update parameters using their gradient
        train_l = loss(net(features, w, b), labels)
    print('epoch %d, loss %f' % (epoch + 1, train_l.mean().asnumpy()))
```

Training

```
lr = 0.03 # Learning rate
num_epochs = 3 # Number of iterations
net = linreg # Our fancy linear model
loss = squared_loss #  $0.5 (y - y')^2$ 

for epoch in range(num_epochs):
    # Assuming the number of examples can be divided by the batch size, all
    # the examples in the training data set are used once in one epoch
    # iteration. The features and tags of mini-batch examples are given by X
    # and y respectively
    for X, y in data_iter(batch_size, features, labels):
        with autograd.record():
            l = loss(net(X, w, b), y) # Minibatch loss in X and y
            l.backward() # Compute gradient on l with respect to [w,b]
            sgd([w, b], lr, batch_size) # Update parameters using their gradient
        train_l = loss(net(features, w, b), labels)
    print('epoch %d, loss %f' % (epoch + 1, train_l.mean().asnumpy()))
```

Thanks!

Does anyone have any questions?

Twitter: @walkercet

Blog: <https://ceteongvanness.wordpress.com>

Resources

Dive into Deep Learning