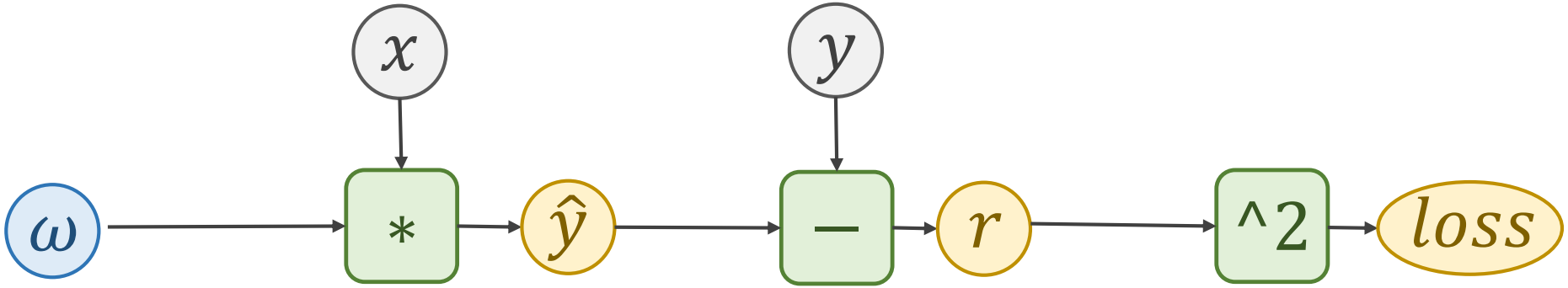
PyTorch Tutorial

05. Linear Regression with PyTorch

Revision

Linear Model	Loss Function
$\hat{y} = x * \omega$	$loss = (\hat{y} - y)^2 = (x \cdot \omega - y)^2$



Revision

```
print("predict (before training)",  4, forward(4).item())

for epoch in range(100):
    for x, y in zip(x_data, y_data):
        l = loss(x, y)
        l.backward()
        print('\tgrad:', x, y, w.grad.item())
        w.data = w.data - 0.01 * w.grad.data

        w.grad.data.zero_()

    print("progress:", epoch, l.item())

print("predict (after training)", 4, forward(4).item())
```

```
predict (before training) 4 4.0
        grad: 1.0 2.0 -2.0
        grad: 2.0 4.0 -7.840000152587891
        grad: 3.0 6.0 -16.228801727294922
progress: 0 7.315943717956543
        grad: 1.0 2.0 -1.478623867034912
        grad: 2.0 4.0 -5.796205520629883
        grad: 3.0 6.0 -11.998146057128906
progress: 1 3.9987640380859375
        grad: 1.0 2.0 -1.0931644439697266
        grad: 2.0 4.0 -4.285204887390137
        grad: 3.0 6.0 -8.870372772216797
progress: 2 2.1856532096862793
        grad: 1.0 2.0 -0.8081896305084229
        grad: 2.0 4.0 -3.1681032180786133
        grad: 3.0 6.0 -6.557973861694336
progress: 3 1.1946394443511963
        grad: 1.0 2.0 -0.5975041389465332
        grad: 2.0 4.0 -2.3422164916992188
        grad: 3.0 6.0 -4.848389625549316
progress: 4 0.6529689431190491
        grad: 1.0 2.0 -0.4417421817779541
        grad: 2.0 4.0 -1.7316293716430664
        grad: 3.0 6.0 -3.58447265625
progress: 5 0.35690122842788696
        grad: 1.0 2.0 -0.3265852928161621
        grad: 2.0 4.0 -1.2802143096923828
        grad: 3.0 6.0 -2.650045394897461
```

PyTorch Fashion

1. Prepare dataset
   we shall talk about this later
2. Design model using Class
   inherit from nn.Module
3. Construct loss and optimizer
   using PyTorch API
4. Training cycle
   forward, backward, update

Linear Regression – 1. Prepare dataset

In PyTorch, the computational graph is in mini-batch fashion, so X and Y are 3×1 Tensors.

即把样本一次性都放进去 必须是矩阵

$$\begin{bmatrix} y_{pred}^{(1)} \\ y_{pred}^{(2)} \\ y_{pred}^{(3)} \end{bmatrix} = \omega \cdot \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ x^{(3)} \end{bmatrix} + b$$

即 $\hat{y}^{(i)}$

1x3 1x3

```
import torch

x_data = torch.Tensor([[1.0], [2.0], [3.0]])
y_data = torch.Tensor([[2.0], [4.0], [6.0]])
```

Revision: Gradient Descent Algorithm

Derivative

$$\frac{\partial cost(\omega)}{\partial \omega} = \frac{\partial}{\partial \omega}\frac{1}{N}\sum_{n=1}^{N}(x_n \cdot \omega - y_n)^2$$

$$= \frac{1}{N}\sum_{n=1}^{N}\frac{\partial}{\partial \omega}(x_n \cdot \omega - y_n)^2$$

$$= \frac{1}{N}\sum_{n=1}^{N}2 \cdot (x_n \cdot \omega - y_n)\frac{\partial (x_n \cdot \omega - y_n)}{\partial \omega}$$

$$= \frac{1}{N}\sum_{n=1}^{N}2 \cdot x_n \cdot (x_n \cdot \omega - y_n)$$

Gradient

$$\frac{\partial cost}{\partial \omega}$$

Update

$$\omega = \omega - \alpha \frac{\partial cost}{\partial \omega}$$

Update

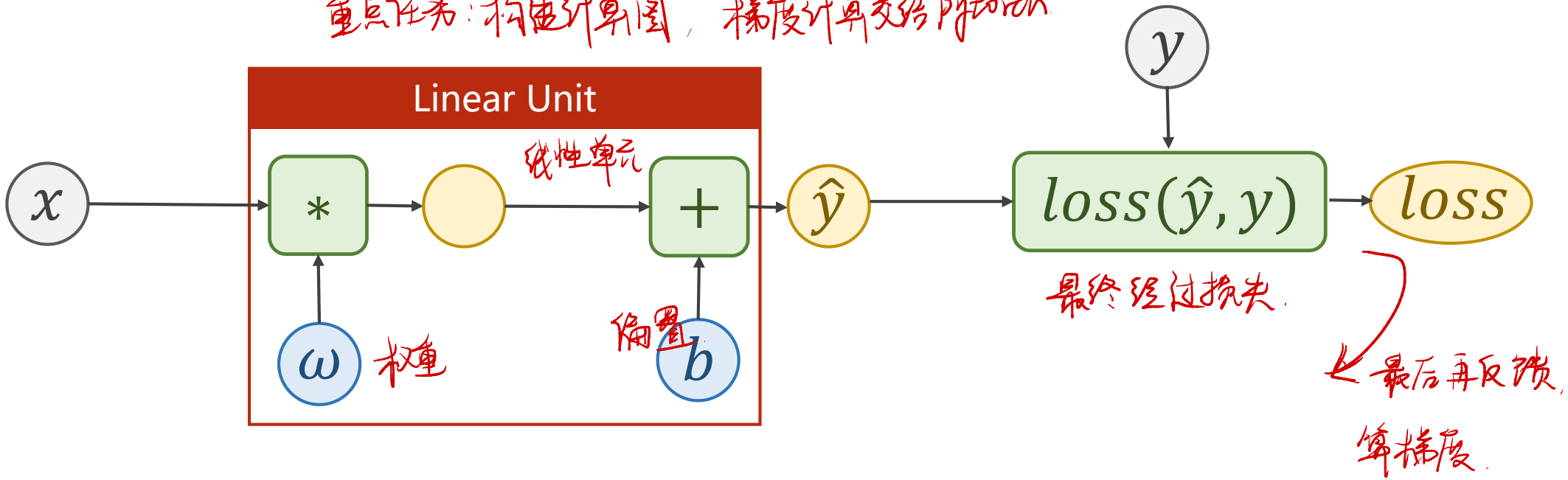
$$\omega = \omega - \alpha \frac{1}{N}\sum_{n=1}^{N}2 \cdot x_n \cdot (x_n \cdot \omega - y_n)$$

Linear Regression – 2. Design Model

最终必须是一个标量，(非向量)

Affine Model	Loss Function
$\hat{y} = x * \omega + b$	$loss = (\hat{y} - y)^2 = (x \cdot \omega - y)^2$

重点任务：构造计算图，梯度计算交给 PyTorch



Linear Regression – 2. Design Model

```python
class LinearModel(torch.nn.Module):
    def __init__(self):
        super(LinearModel, self).__init__()
        self.linear = torch.nn.Linear(1, 1)

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

model = LinearModel()
```

Our model class should be inherit from *nn.Module*, which is Base class for all neural network modules.

Linear Regression – 2. Design Model

```
class LinearModel(torch.nn.Module):
    def __init__(self):
        super(LinearModel, self).__init__()
        self.linear = torch.nn.Linear(1, 1)

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

model = LinearModel()
```

Member methods *__init__()* and *forward()* have to be implemented.

Linear Regression – 2. Design Model

```
class LinearModel(torch.nn.Module):
    def __init__(self):
        super(LinearModel, self).__init__()
        self.linear = torch.nn.Linear(1, 1)

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

model = LinearModel()
```

Just do it. :)

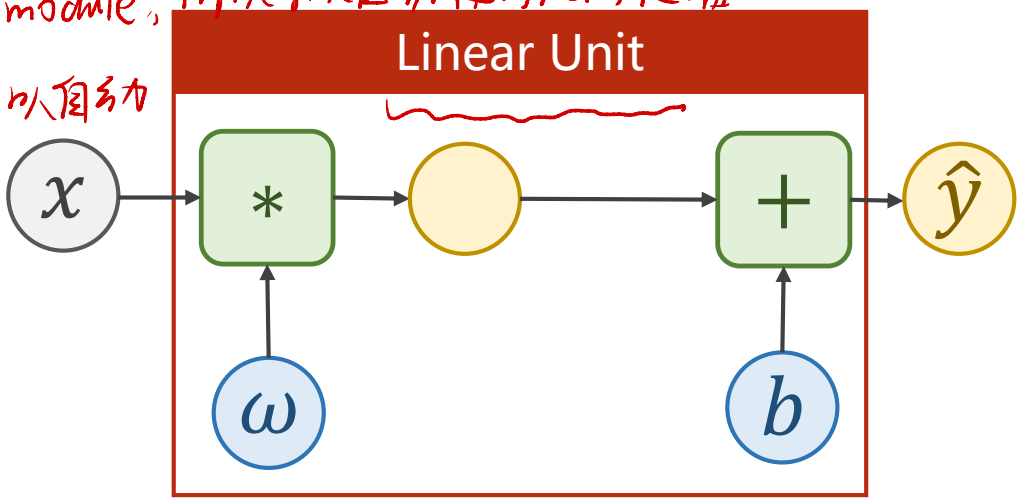
Linear Regression – 2. Design Model

```
class LinearModel(torch.nn.Module):
    def __init__(self):
        super(LinearModel, self).__init__()
        self.linear = torch.nn.Linear(1, 1)

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

model = LinearModel()
```

Class *nn.Linear* contain two member **Tensors**: **weight** and **bias**.



Linear Regression – 2. Design Model

class **torch.nn.Linear**(*in_features, out_features, bias=True*) [source]

Applies a linear transformation to the incoming data: $y = Ax + b$

Linear 所作的运算

Parameters:

- **in_features** – size of each input sample (即维度)
- **out_features** – size of each output sample (样本)
- **bias** – If set to False, the layer will not learn an additive bias. Default: `True` (是否有偏置 b)

$$\underset{y_{n\times 2}}{[\ ]} = \underset{X_{n\times 3}}{[\ ]}\ \underset{W_{3\times 2}}{[\ ]}$$

但实际一般作转置

$$\underset{y_{2\times n}}{[\ ]} = \underset{2\times 3}{W^T}\ \underset{X_{3\times n}}{[\ ]} + \underset{b}{[\ ]}$$

行数为特征数量，列数为样本数量

Shape:

- Input: $(N, *, in\_features)$ where $*$ means any number of additional dimensions
- Output: $(N, *, out\_features)$ where all but the last dimension are the same shape as the input.

Variables:

- **weight** – the learnable weights of the module of shape *(out_features x in_features)*
- **bias** – the learnable bias of the module of shape *(out_features)*

Linear Regression – 2. Design Model

class **torch.nn.Linear**(*in_features, out_features, bias=True*) [source]

Applies a linear transformation to the incoming data: $y = Ax + b$

Parameters:

$$\begin{bmatrix} y_{pred}^{(1)} \\ y_{pred}^{(2)} \\ y_{pred}^{(3)} \end{bmatrix} = \omega \cdot \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ x^{(3)} \end{bmatrix} + b$$

Output

Input

tive bias. Default: `True`

Shape:

- Input: $(N, *, in\_features)$ where $*$ means any number of additional dimensions
- Output: $(N, *, out\_features)$ where all but the last dimension are the same shape as the input.

Variables:

- **weight** – the learnable weights of the module of shape *(out_features x in_features)*
- **bias** – the learnable bias of the module of shape *(out_features)*

Linear Regression – 2. Design Model

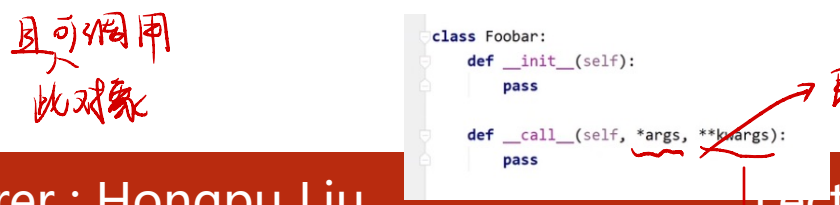
```
class LinearModel(torch.nn.Module):
    def __init__(self):
        super(LinearModel, self).__init__()
        self.linear = torch.nn.Linear(1, 1)

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

model = LinearModel()
```

Class *nn.Linear* has implemented the magic method ***__call__()***, which enable the instance of the class can be called just like a function. Normally the ***forward()*** will be called.

Pythonic!!!



Linear Regression – 2. Design Model

**kwargs 传递归类的参数如 x=1, y=2, 并将其转为字典

```
class LinearModel(torch.nn.Module):
    def __init__(self):
        super(LinearModel, self).__init__()
        self.linear = torch.nn.Linear(1, 1)

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

model = LinearModel()
```

Create a instance of class **LinearModel**.

Linear Regression – 3. Construct Loss and Optimizer

```
criterion = torch.nn.MSELoss(size_average=False)

optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

class **torch.nn.MSELoss**(*size_average=True, reduce=True*) [source]

Creates a criterion that measures the mean squared error betwee
target *y*.

The loss can be described as:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = (x_n - y_n)^2,$$

where N is the batch size.

Also inherit from **nn.Module**.

Linear Regression – 3. Construct Loss and Optimizer

```
criterion = torch.nn.MSELoss(size_average=False)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

class **torch.optim.SGD**(*params, lr=<object object>, momentum=0, dampening=0, weight_decay=0, nesterov=False*) [source]

Implements stochastic gradient descent (optionally with momentum).

Linear Regression – 3. Construct Loss and Optimizer

```
criterion = torch.nn.MSELoss(size_average=False)

optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

Parameters:

- **params** (*iterable*) – iterable of parameters to optimize or dicts defining parameter groups
- **lr** (*float*) – learning rate

Linear Regression – 4. Training Cycle

训练过程

```
for epoch in range(100):
    y_pred = model(x_data)
    loss = criterion(y_pred, y_data)
    print(epoch, loss)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

前馈

Forward: Predict

Linear Regression – 4. Training Cycle

```
for epoch in range(100):
    y_pred = model(x_data)
    loss = criterion(y_pred, y_data)
    print(epoch, loss)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Forward: Loss

Linear Regression – 4. Training Cycle

```
for epoch in range(100):
    y_pred = model(x_data)
    loss = criterion(y_pred, y_data)
    print(epoch, loss)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

梯度清零

NOTICE:

The grad computed by *.backward()* will be **accumulated**.

So before backward, remember set the grad to **ZERO**!!!

Linear Regression – 4. Training Cycle

```
for epoch in range(100):
    y_pred = model(x_data)
    loss = criterion(y_pred, y_data)
    print(epoch, loss)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

前 <

反 <

更新 →

计算梯度（反向传播）

进行一次更新

根据lr，和包含的权重

Backward: Autograd

Linear Regression – 4. Training Cycle

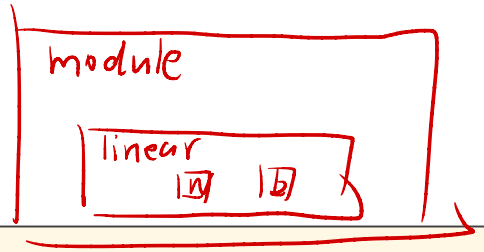
```
for epoch in range(100):
    y_pred = model(x_data)
    loss = criterion(y_pred, y_data)
    print(epoch, loss)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
for x, y in zip(x_data, y_data):
    ......
    w.data = w.data - 0.01 * w.grad.data
```

Update

Linear Regression – Test Model




```
# Output weight and bias
print('w = ', model.linear.weight.item())
print('b = ', model.linear.bias.item())

# Test Model
x_test = torch.Tensor([[4.0]])
y_test = model(x_test)
print('y_pred = ', y_test.data)
```

```
86 0.3036523759365082
87 0.2992883026599884
88 0.29498720169067383
89 0.2907477021217346
90 0.28656935691833496
91 0.28245046734809875
92 0.27839142084121704
93 0.27439042925834656
94 0.2704470157623291
95 0.2665606141090393
96 0.262729674577713
97 0.25895369052886963
98 0.2552322745323181
99 0.2515641450881958
w =  1.666100263595581
b =  0.7590328454971313
y_pred =  tensor([[ 7.4234]])
```

100 Iterations

```
986 3.594939812501252e-07
987 3.5411068211033125e-07
988 3.4917979974125046e-07
989 3.4428359185767476e-07
990 3.392528924450744e-07
991 3.3442694302721065e-07
992 3.294019847999152e-07
993 3.247135396122758e-07
994 3.199925231456291e-07
995 3.1540417921860353e-07
996 3.109785779997646e-07
997 3.0668098816022393e-07
998 3.020934400410624e-07
999 2.977626536448952e-07
w =  1.9996366500854492
b =  0.0008257834706455469
y_pred =  tensor([[ 7.9994]])
```

1000 Iterations

Linear Regression

```python
import torch

x_data = torch.Tensor([[1.0], [2.0], [3.0]])
y_data = torch.Tensor([[2.0], [4.0], [6.0]])

class LinearModel(torch.nn.Module):
    def __init__(self):
        super(LinearModel, self).__init__()
        self.linear = torch.nn.Linear(1, 1)

    def forward(self, x):
        y_pred = self.linear(x)
        return y_pred
model = LinearModel()

criterion = torch.nn.MSELoss(size_average=False)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

for epoch in range(1000):
    y_pred = model(x_data)
    loss = criterion(y_pred, y_data)
    print(epoch, loss.item())

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

print('w = ', model.linear.weight.item())
print('b = ', model.linear.bias.item())

x_test = torch.Tensor([[4.0]])
y_test = model(x_test) //利用模型
print('y_pred = ', y_test.data)
```

1. Prepare dataset
   we shall talk about this later
2. Design model using Class
   inherit from nn.Module
3. Construct loss and optimizer
   using PyTorch API
4. Training cycle
   forward, backward, update

Exercise 5-1: Try Different Optimizer in Linear Regression

- torch.optim.Adagrad
- torch.optim.Adam
- torch.optim.Adamax
- torch.optim.ASGD
- torch.optim.LBFGS
- torch.optim.RMSprop
- torch.optim.Rprop
- torch.optim.SGD

pytorch 的优化器

Exercise 5-2: Read more example from official tutorial

Table of Contents

https://pytorch.org/tutorials/beginner/pytorch_with_examples.html

PyTorch Tutorial

05. Linear Regression with PyTorch