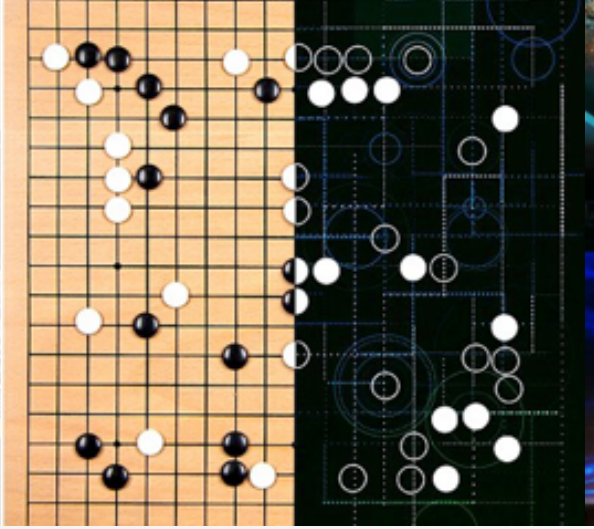
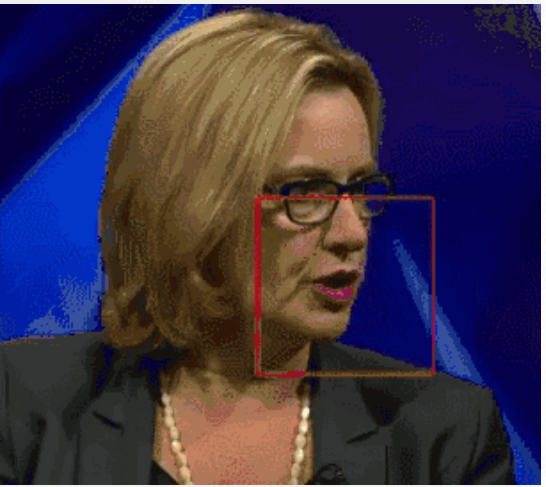
Deep Learning

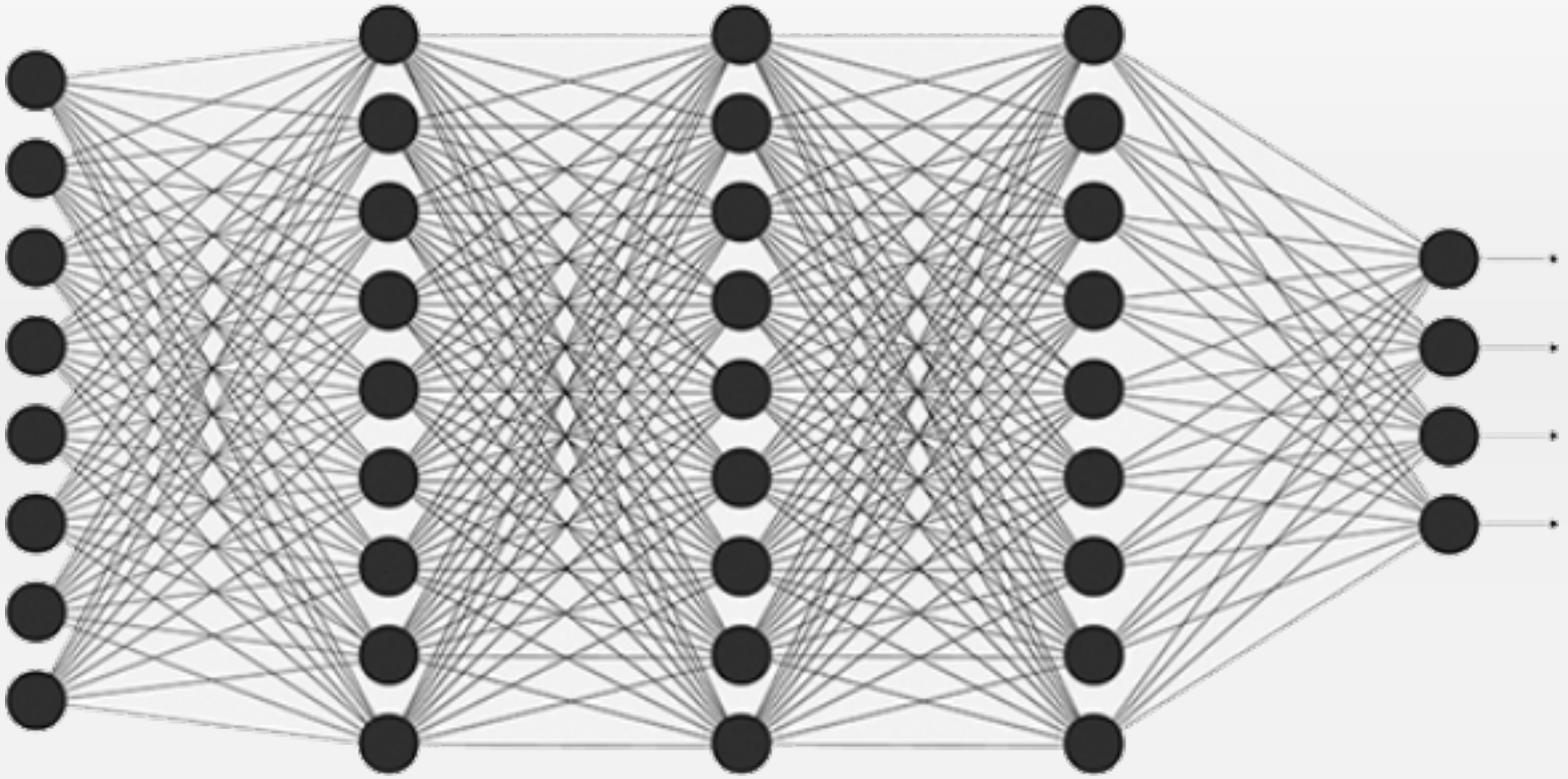
张江

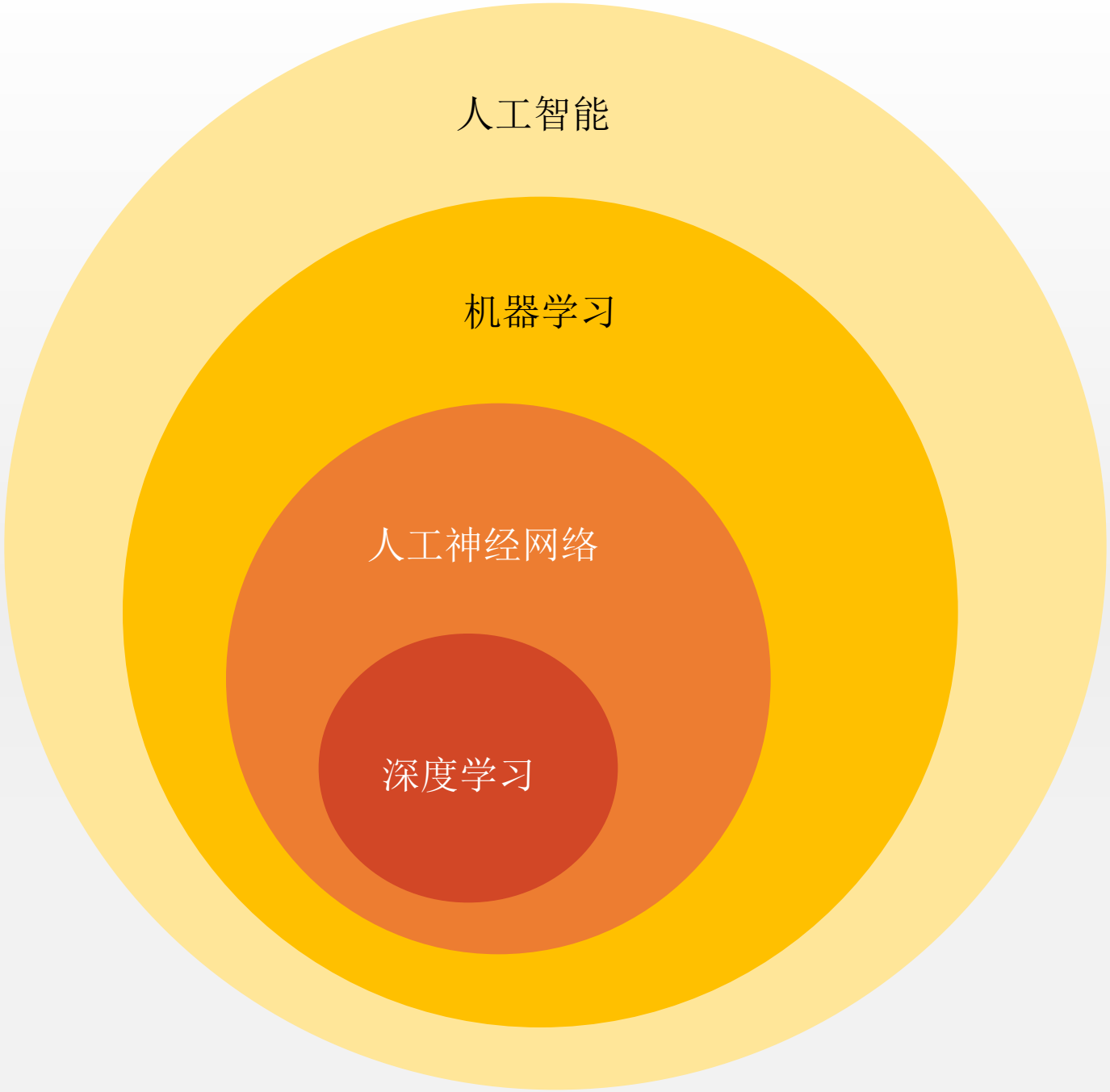


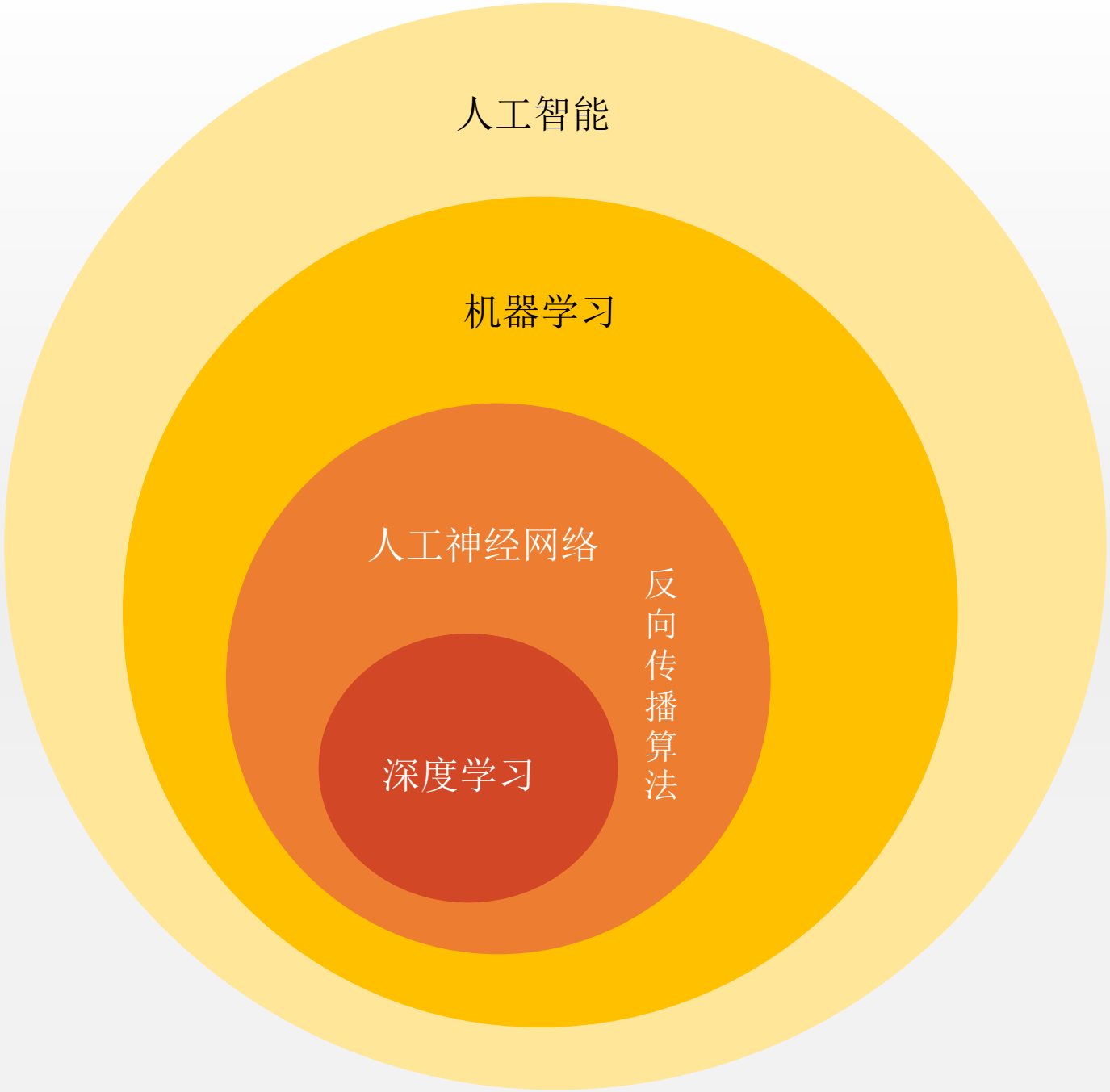
Deep Learning



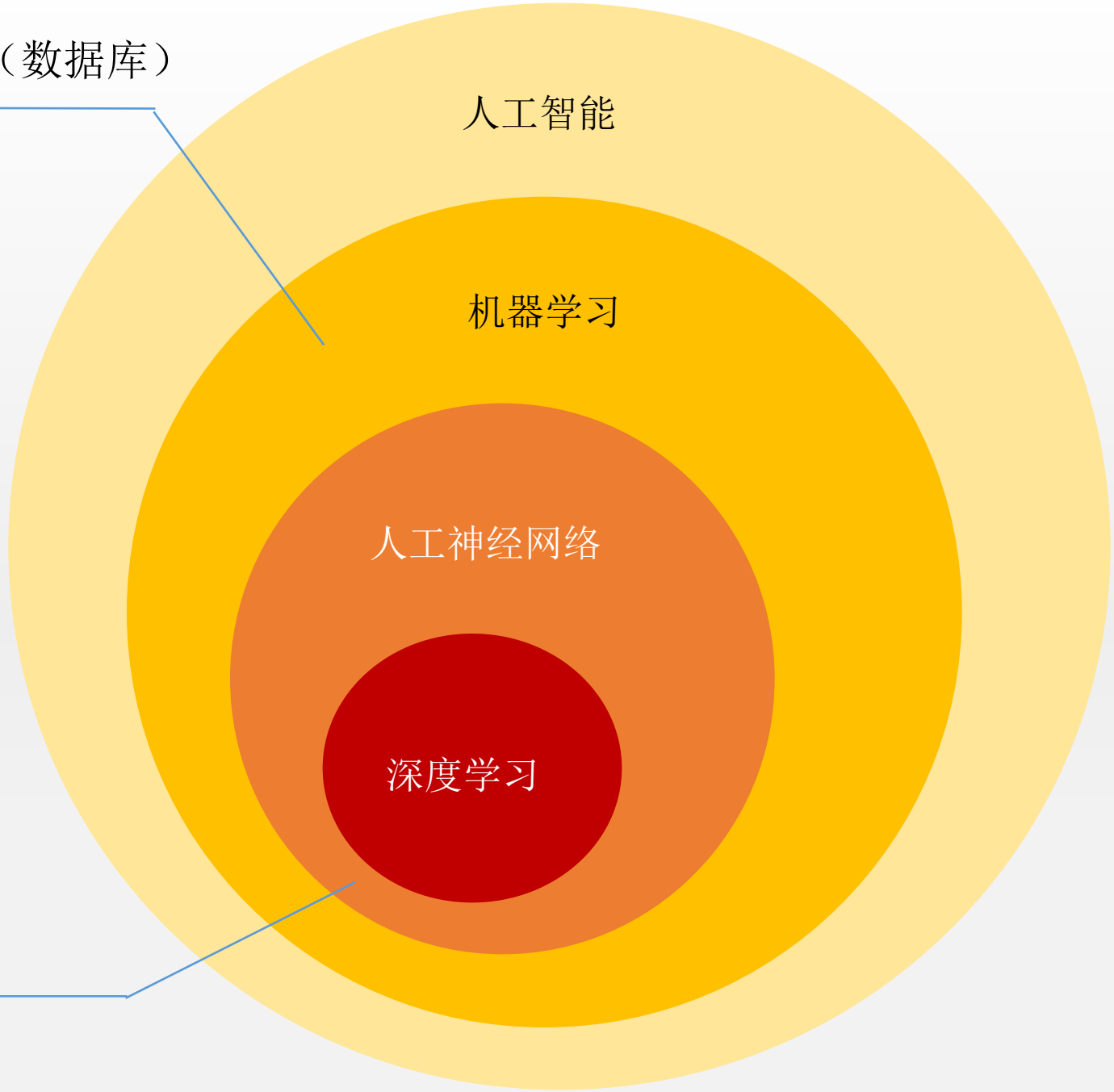
什么是深度学习？



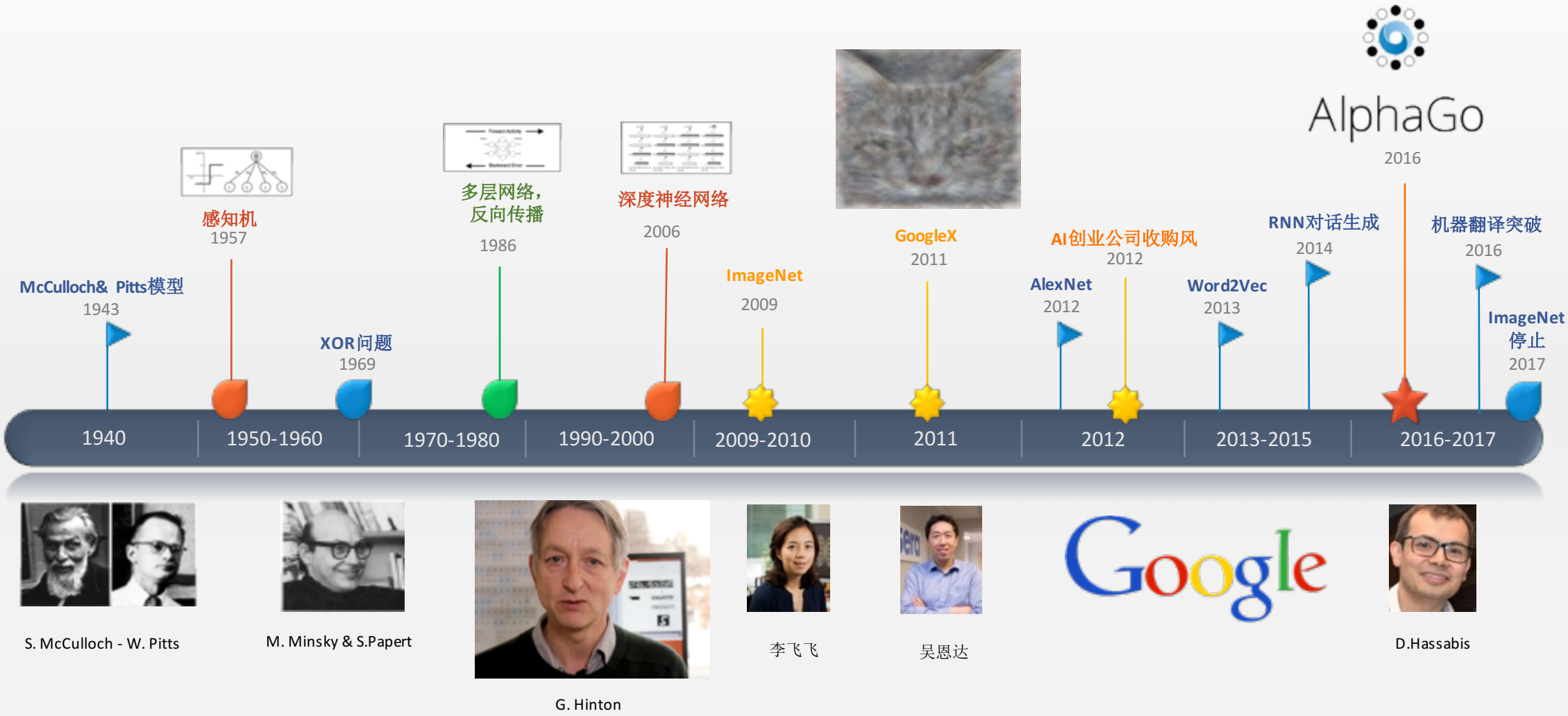

人工智能
机器学习
人工神经网络
深度学习


人工智能
机器学习
人工神经网络
反向传播算法
深度学习

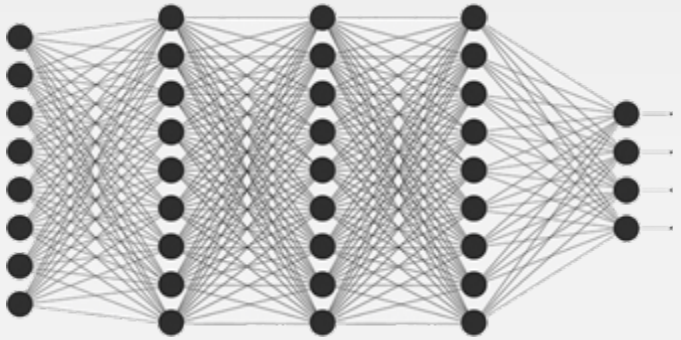
结构化数据（数据库）

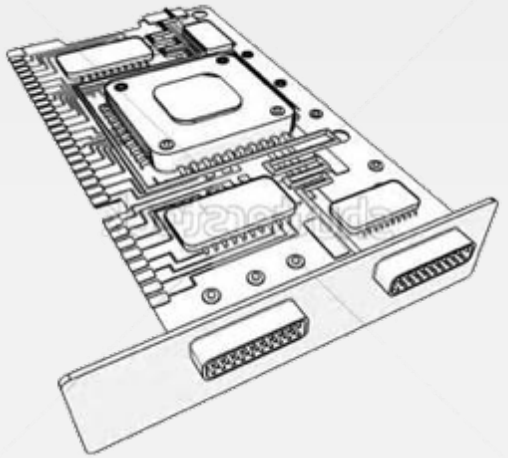


非结构化数据：
图像及符号序列


AlphaGo
2016
感知机
1957
多层网络，
反向传播
1986
深度神经网络
2006
GoogleX
2011
AI创业公司收购风
2012
RNN对话生成
2014
机器翻译突破
2016
McCulloch& Pitts模型
1943
XOR问题
1969
ImageNet
2009
AlexNet
2012
Word2Vec
2013
ImageNet
停止
2017
1940
1950-1960
1970-1980
1990-2000
2009-2010
2011
2012
2013-2015
2016-2017
S. McCulloch - W. Pitts
M. Minsky & S.Papert
G. Hinton
李飞飞
吴恩达
Google
D.Hassabis

Deep Learning

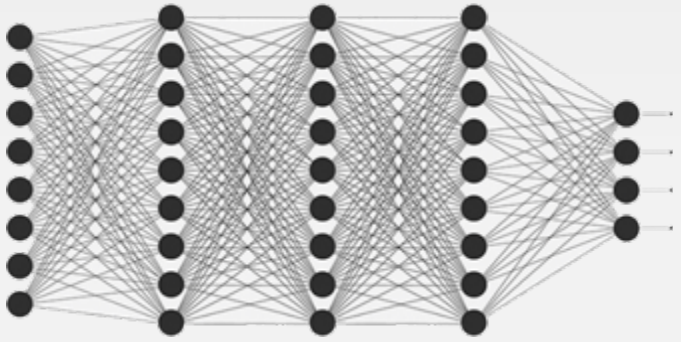




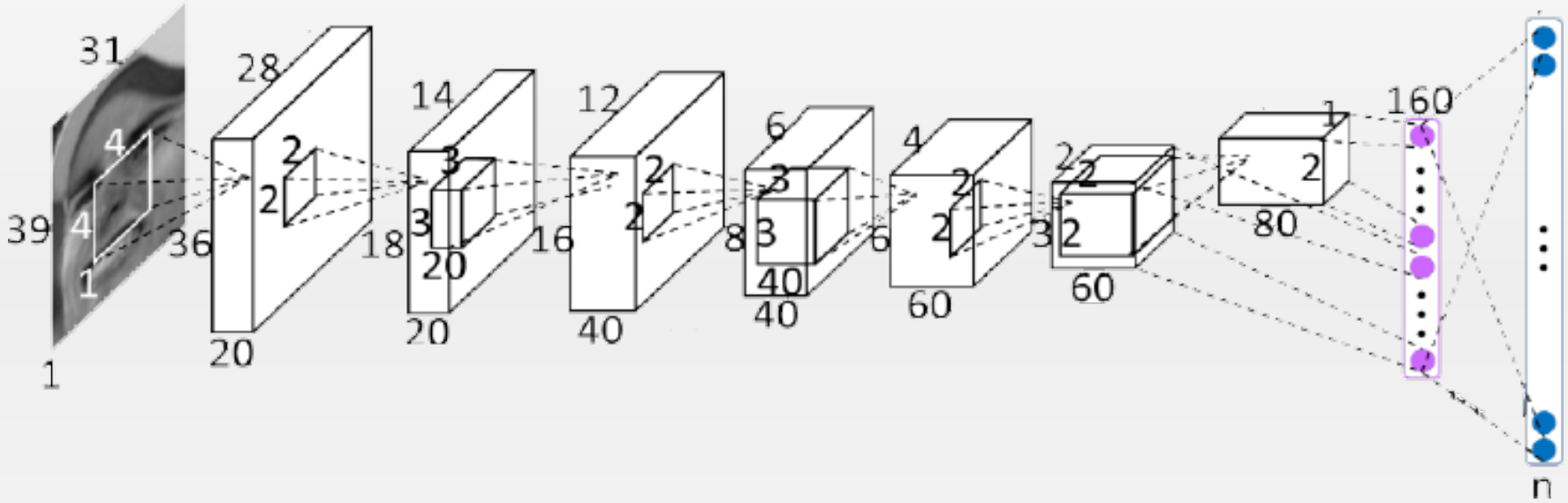
超参数

架构

训练方法

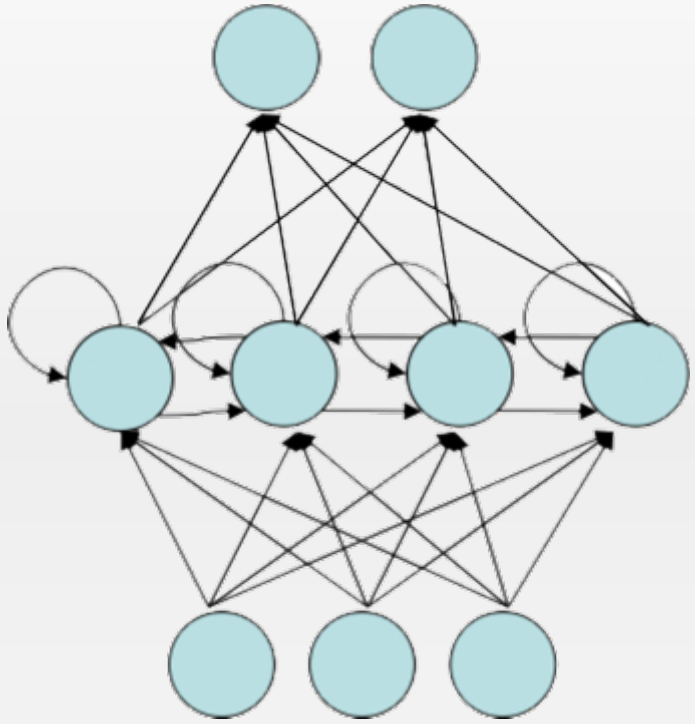


卷积神经网络(CNN)：处理图像数据



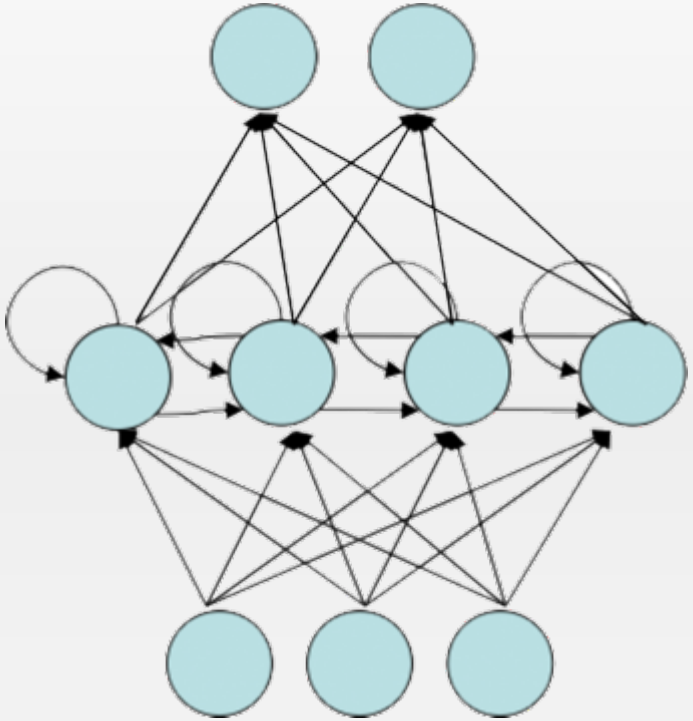
CNN可以应付图像数据中的平移、旋转等空间不变性

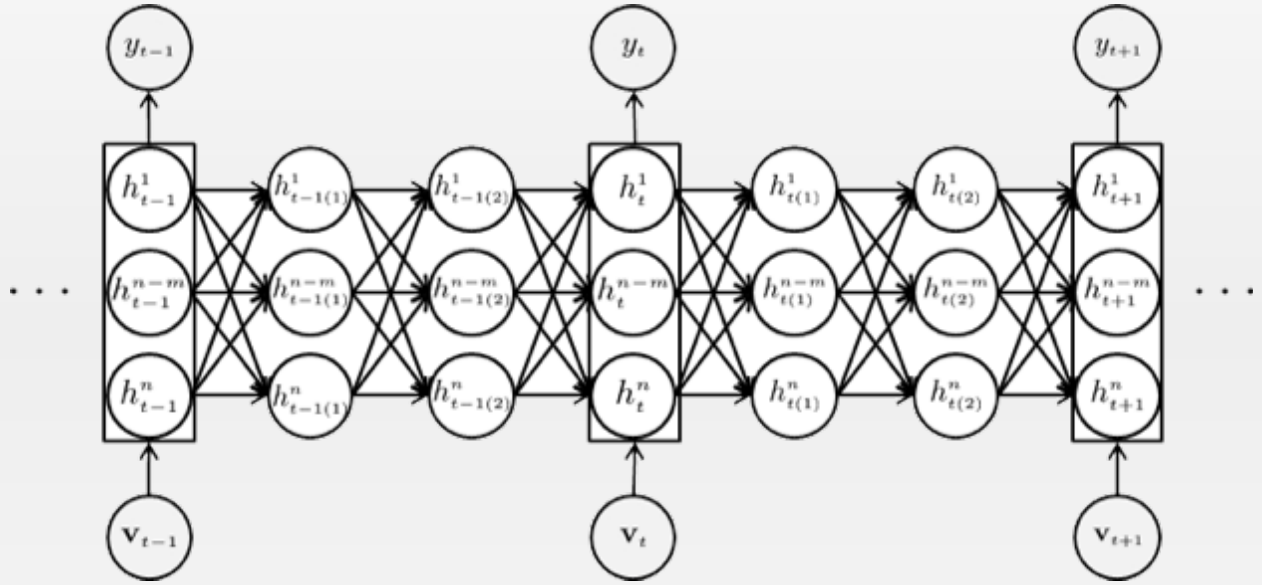
循环神经网络(RNN)：处理序列数据



RNN可以应付序列中的长程依赖性，周期性

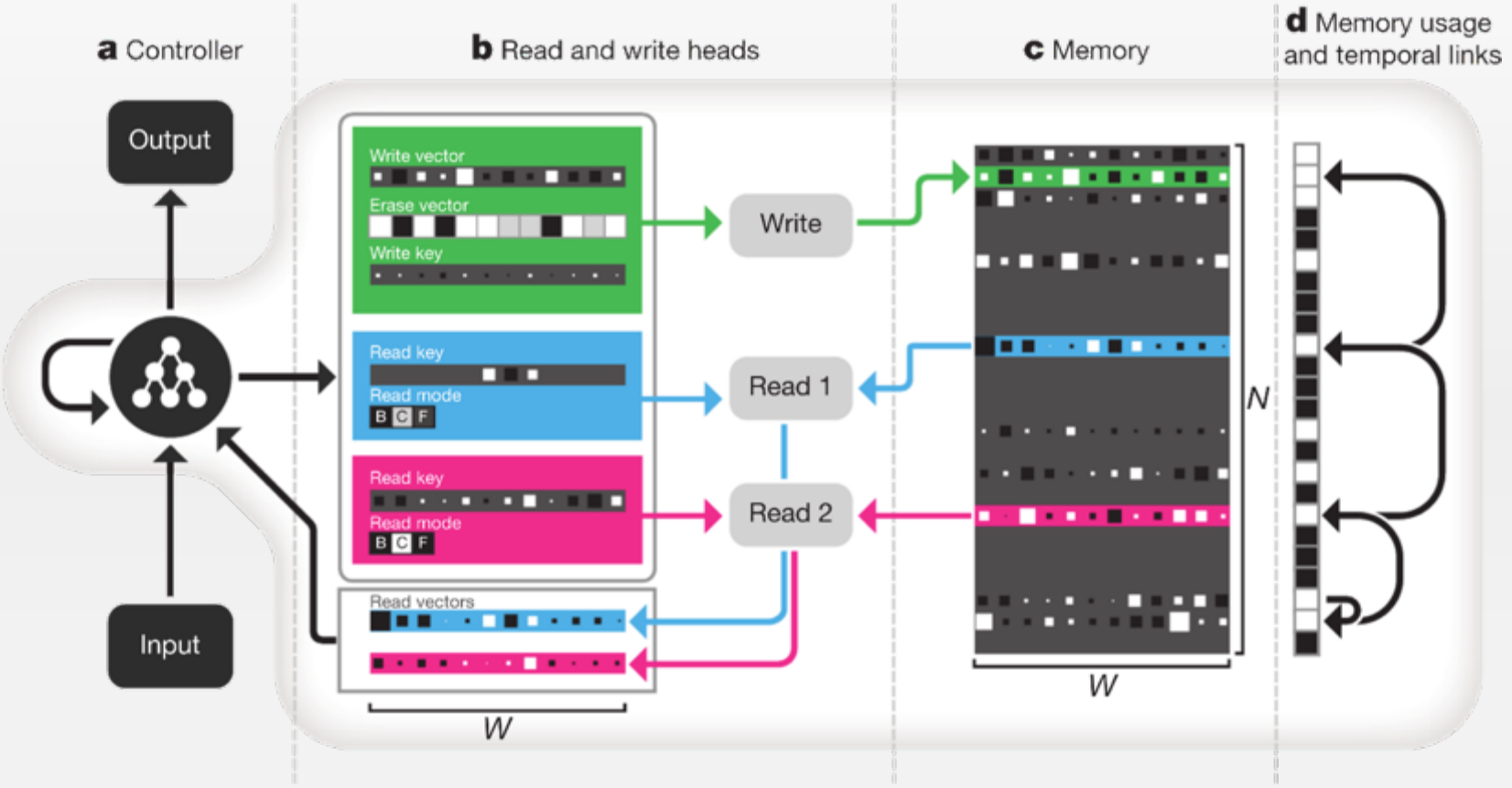
循环神经网络(RNN)：处理序列数据





RNN可以应付序列中的长程依赖性，周期性

神经计算机（神经图灵机）



神经计算机可以用于复杂的推理任务

训练方式的影响

Curriculum Learning

Yoshua Bengio[1] YOSHUA.BENGIO@UMONTREAL.CA
Jérôme Louradour[1,2] JEROMELOURADOUR@GMAIL.COM
Ronan Collobert[3] RONAN@COLLOBERT.COM
Jason Weston[3] JASONW@NEC-LABS.COM
(1) U. MONTREAL, P.O. BOX 6128, MONTREAL, CANADA (2) A2IA SA, 40BIS FABERT, PARIS, FRANCE
(3) NEC LABORATORIES AMERICA, 4 INDEPENDENCE WAY, PRINCETON, NJ, USA

Abstract

Humans and animals learn much better when the examples are not randomly presented but organized in a meaningful order which illustrates gradually more concepts, and gradually more complex ones. Here, we formalize such training strategies in the context of machine learning, and call them "curriculum learning". In the context of recent research studying the difficulty of training in the presence of non-convex training criteria (for deep deterministic and stochastic neural networks), we explore curriculum learning in various set-ups. The experiments show that significant improvements in generalization can be achieved. We hypothesize that curriculum learning has both an effect on the speed of convergence of the training process to a minimum and, in the case of non-convex criteria, on the quality of the local minima obtained: curriculum learning can be seen as a particular form of continuation method (a general strategy for global optimization of non-convex functions).

training and remarkably increase the speed at which learning can occur. This idea is routinely exploited in *animal training* where it is called **shaping** (Skinner, 1958; Peterson, 2004; Krueger & Dayan, 2009).

Previous research (Elman, 1993; Rohde & Plaut, 1999; Krueger & Dayan, 2009) at the intersection of cognitive science and machine learning has raised the following question: can machine learning algorithms benefit from a similar training strategy? The idea of training a learning machine with a curriculum can be traced back at least to Elman (1993). The basic idea is to *start small*, learn easier aspects of the task or easier subtasks, and then gradually increase the difficulty level. The experimental results, based on learning a simple grammar with a recurrent network (Elman, 1993), suggested that successful learning of grammatical structure depends, not on innate knowledge of grammar, but on starting with a limited architecture that is at first quite restricted in complexity, but then expands its resources gradually as it learns. Such conclusions are important for developmental psychology, because they illustrate the adaptive value of starting, as human infants do, with a simpler initial state, and then building on that to develop more and more sophisticated representations of structure. Elman (1993)

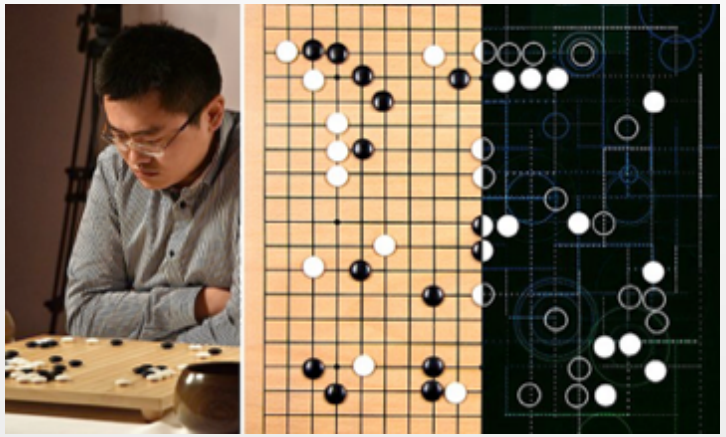
How transferable are features in deep neural networks?

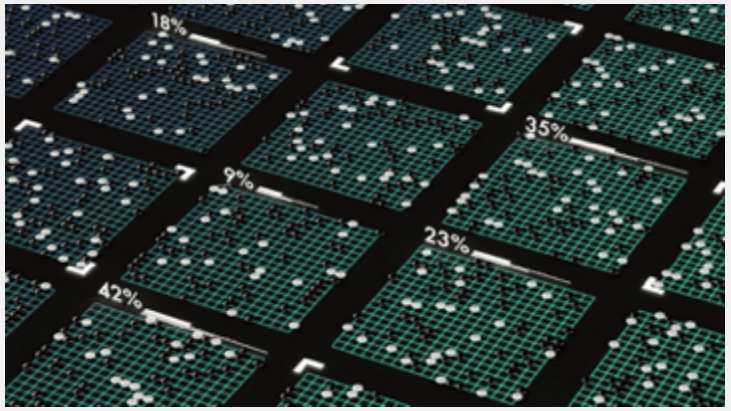
Jason Yosinski,[1] **Jeff Clune,**[2] **Yoshua Bengio,**[3] **and Hod Lipson**[4]
[1] Dept. Computer Science, Cornell University
[2] Dept. Computer Science, University of Wyoming
[3] Dept. Computer Science & Operations Research, University of Montreal
[4] Dept. Mechanical & Aerospace Engineering, Cornell University

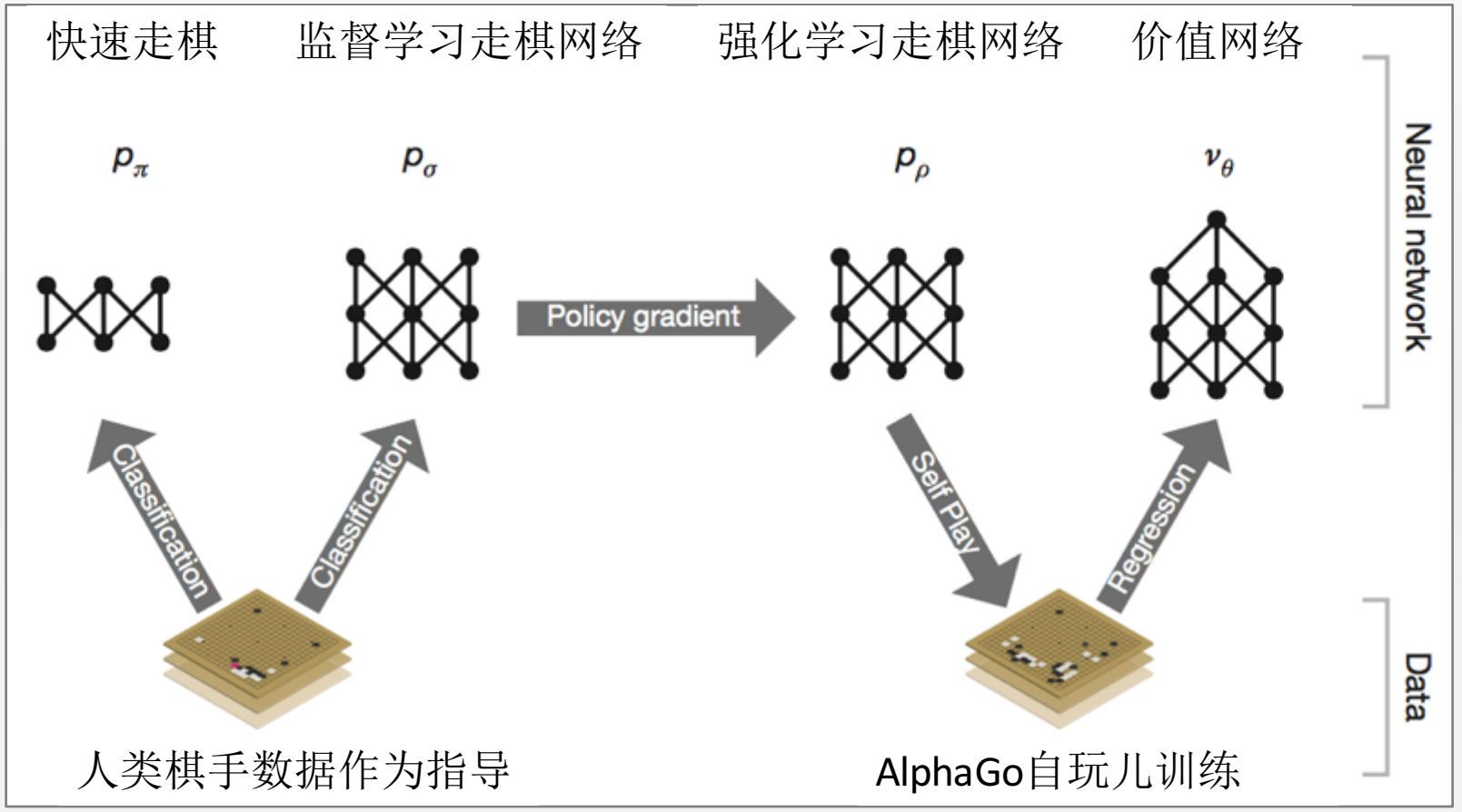
Abstract

Many deep neural networks trained on natural images exhibit a curious phenomenon in common: on the first layer they learn features similar to Gabor filters and color blobs. Such first-layer features appear not to be *specific* to a particular dataset or task, but *general* in that they are applicable to many datasets and tasks. Features must eventually transition from general to specific by the last layer of the network, but this transition has not been studied extensively. In this paper we experimentally quantify the generality versus specificity of neurons in each layer of a deep convolutional neural network and report a few surprising results. Transferability is negatively affected by two distinct issues: (1) the specialization of higher layer neurons to their original task at the expense of performance on the target task, which was expected, and (2) optimization difficulties related to splitting networks between co-adapted neurons, which was not expected. In an example network trained on ImageNet, we demonstrate that either of these two issues may dominate, depending on whether features are transferred from the bottom, middle, or top of the network. We also document that the transferability of features decreases as the distance between the base task and target task increases, but that transferring features even from distant tasks can be better than using random features. A final surprising result is that initializing a network with transferred features from almost any number of layers can produce a boost to generalization that lingers even after fine-tuning to the target dataset.

复杂的训练流程




18%
35%
9%
23%
42%


快速走棋
监督学习走棋网络
强化学习走棋网络
价值网络
p_π
p_σ
p_ρ
v_θ
Policy gradient
Classification
Classification
Self Play
Regression
Neural network
Data
人类棋手数据作为指导
AlphaGo自玩儿训练

什么是深度学习的本质？

什么是深度学习的本质？

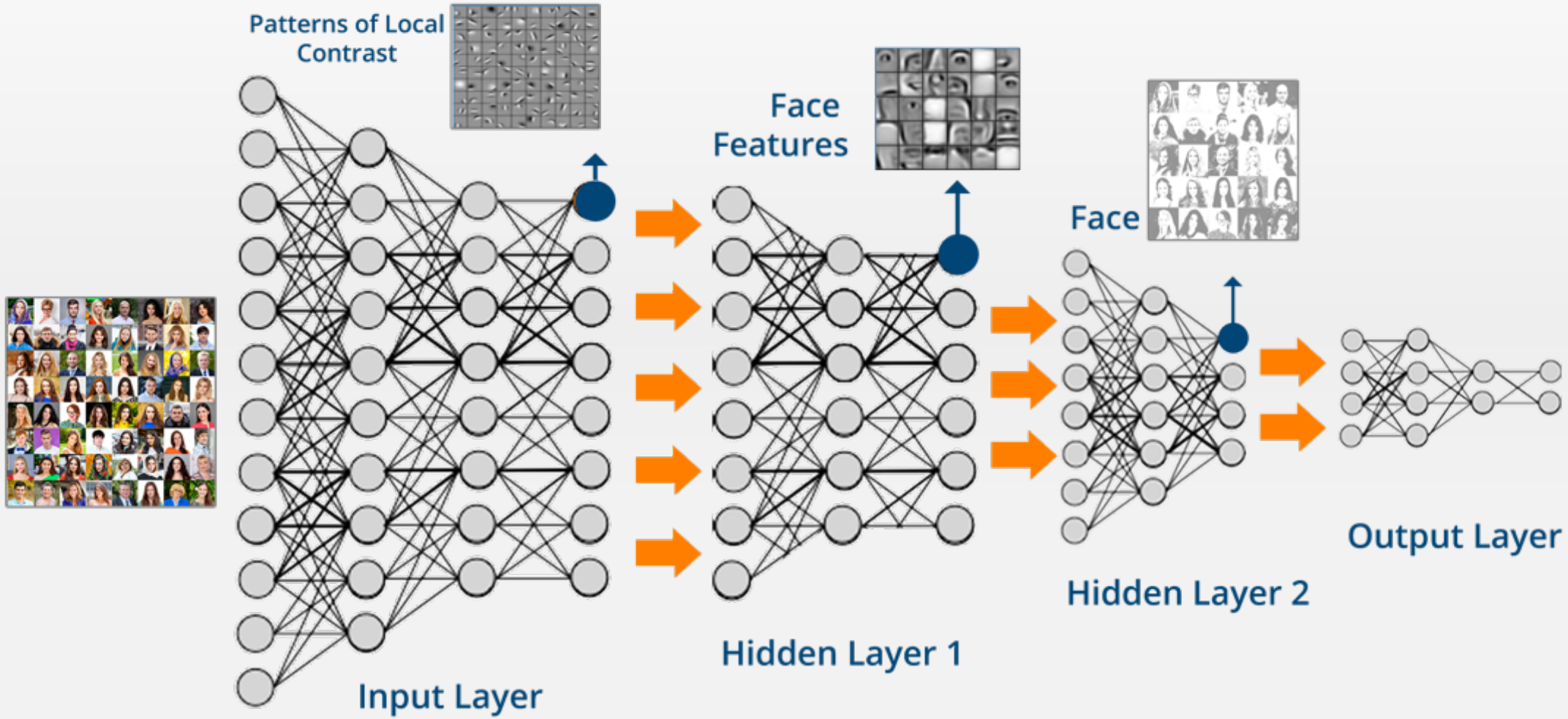
Deep?

什么是深度学习的本质？

Deep?

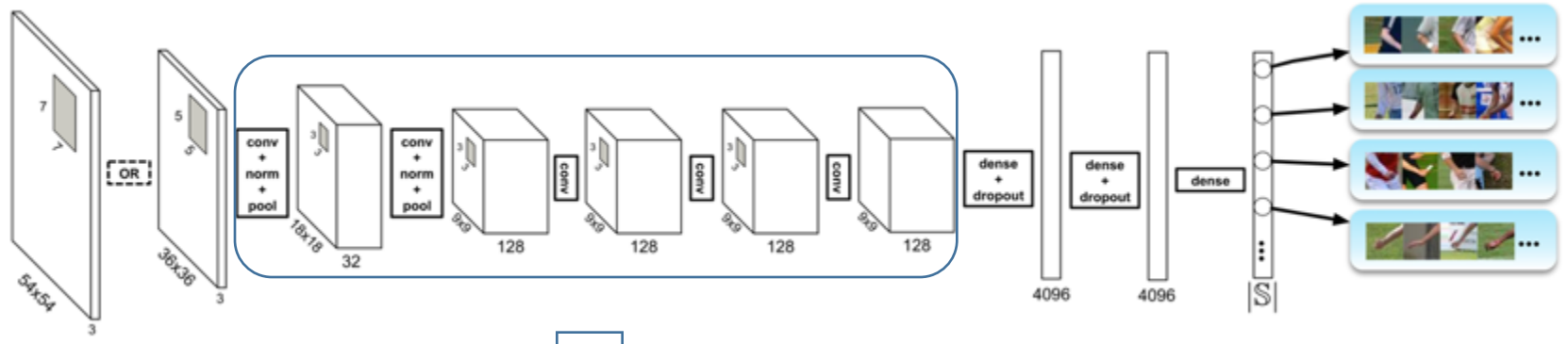
特征学习

深度可以从多个尺度把握数据中的特征



特征抽取之后还可以迁移

图像识别网络

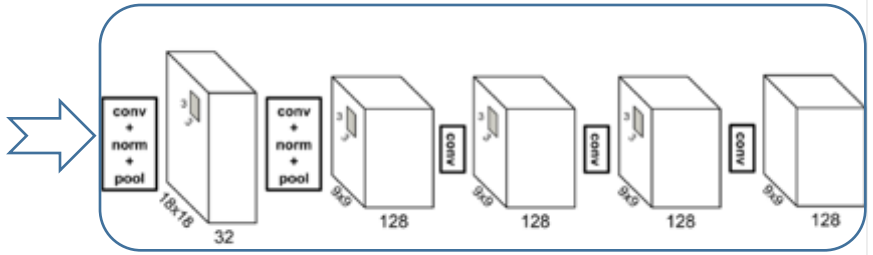


CNN网络

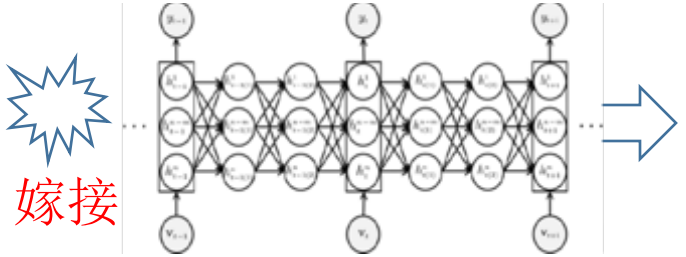
迁移

RNN网络

看图说话网络

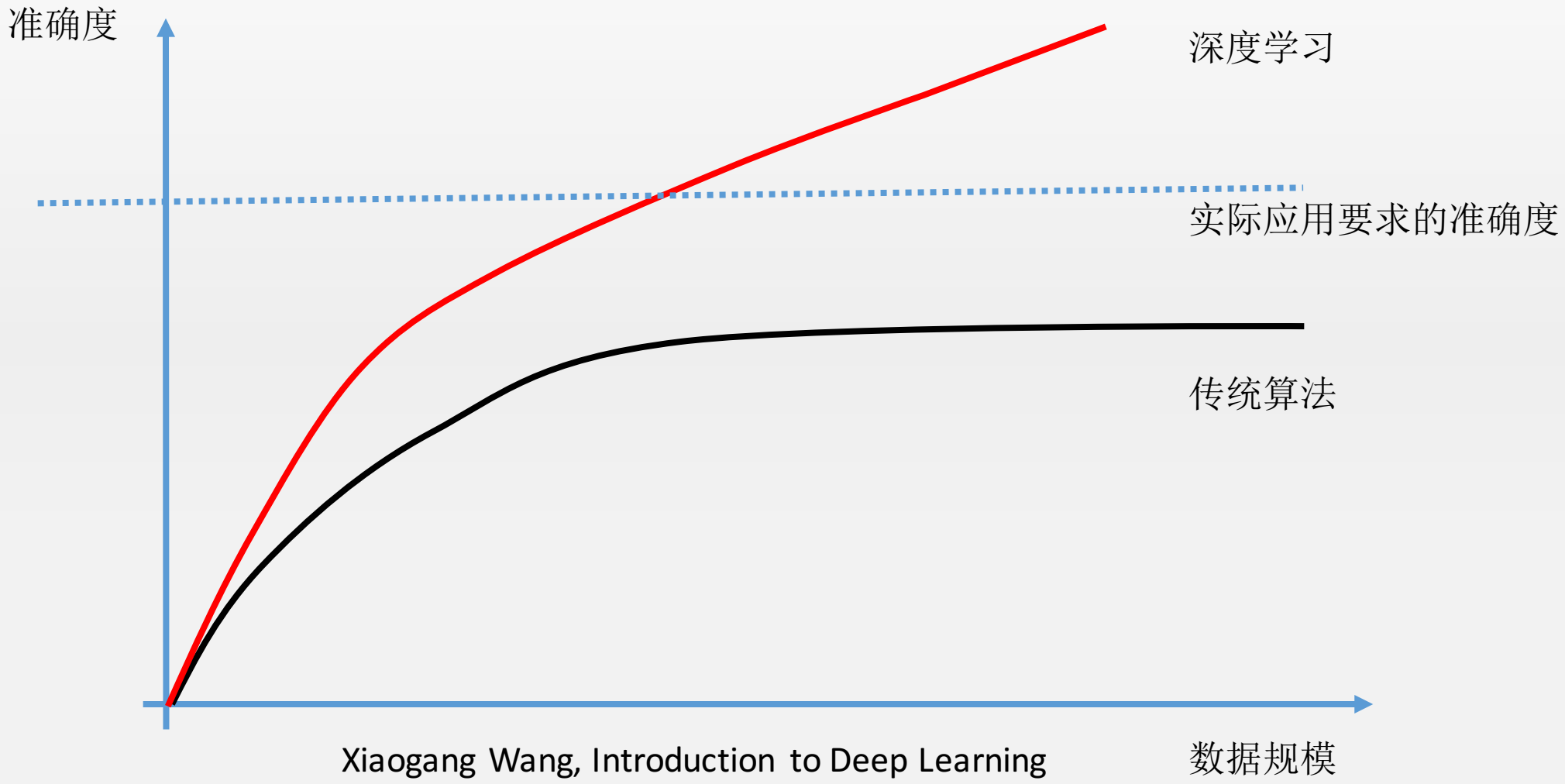


嫁接



一群人在菜市场卖菜

为什么如此受欢迎?



Xiaogang Wang, Introduction to Deep Learning


The Incredible
PYTORCH

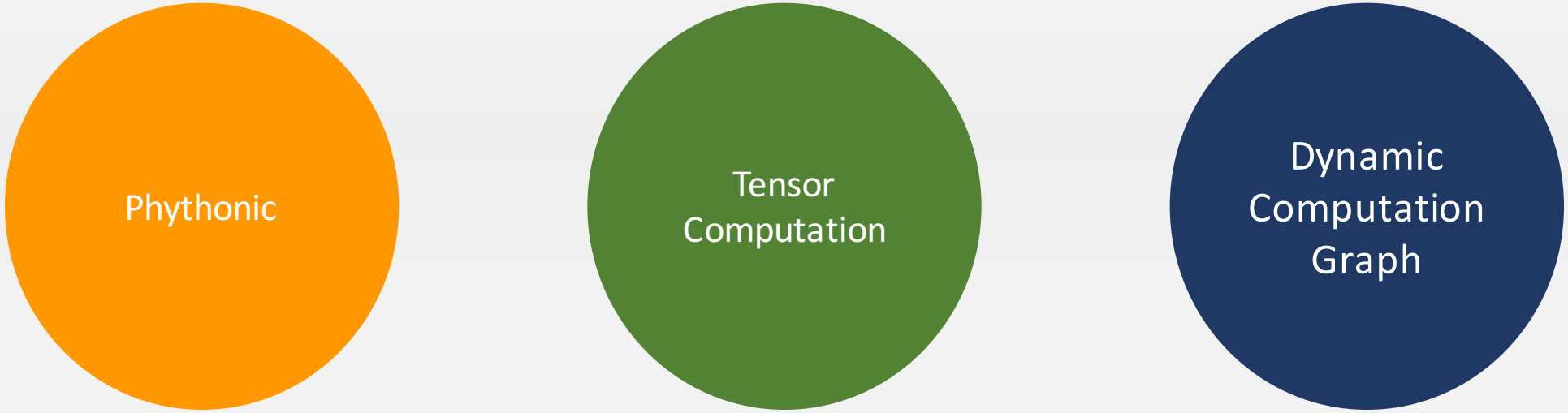
一个深度学习
开源框架

The Incredible PYTORCH

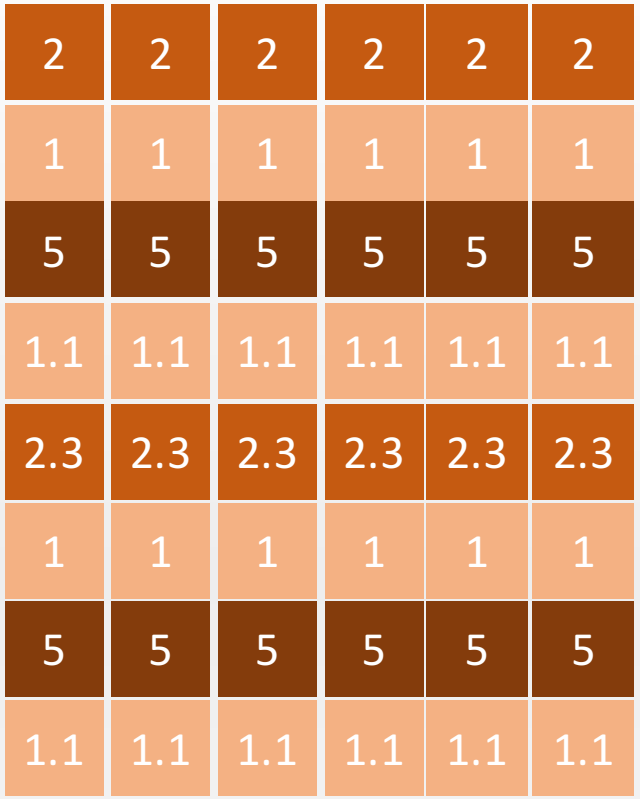
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)

model = Net()
if args.cuda:
    model.cuda()
```

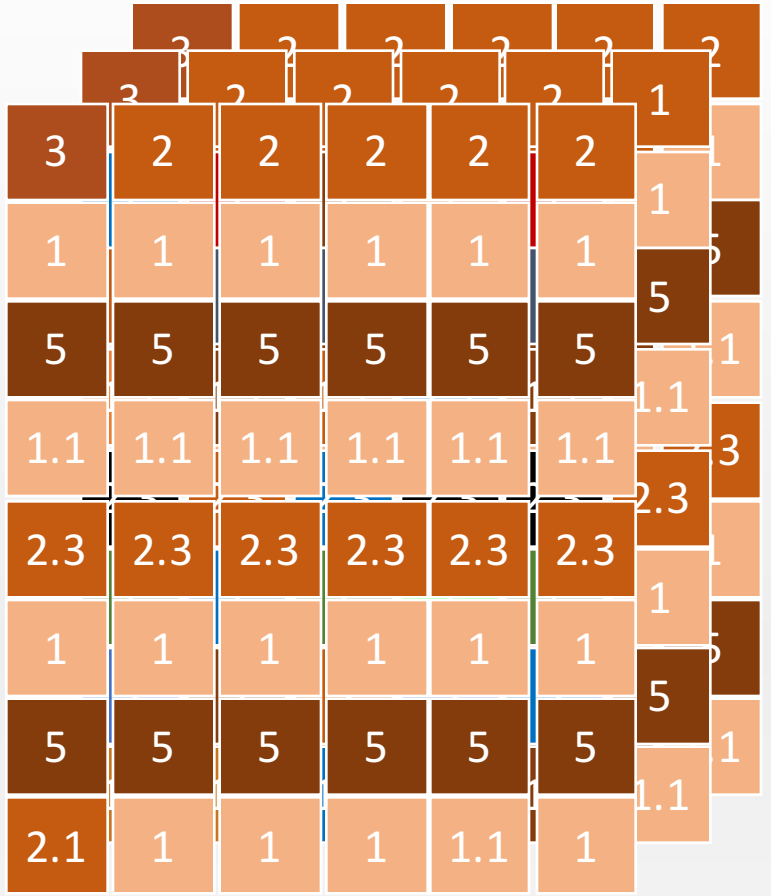
```
optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=args.momentum)

def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        if args.cuda:
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.data[0]))
```

The Incredible PYTORCH



什么是Tensor?





a= [2,1,5,1.1,2.3,1,5,1.1]

a.size()= 8

$$\begin{bmatrix} 2 & \cdots & 2 \\ \vdots & \ddots & \vdots \\ 1.1 & \cdots & 1.1 \end{bmatrix}$$

(8,6)

$$\begin{bmatrix} 3 & \cdots & 2 \\ \vdots & \ddots & \vdots \\ 2.1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} 3 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 2.1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} 3 & \cdots & 2 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

(3,8,6)

Tensor的创建

In [1]:
```
import torch
```

首先，要导入PyTorch包，输入完后按Shift+Enter

Tensor的创建

In [1]:
```
import torch
```

In [2]:
```
x = torch.rand(5, 3)
x
```

创建一个5*3的随机矩阵
并显示它（Shift+Enter）

Tensor的创建

In [1]:
```
import torch
```

In [2]:
```
x = torch.rand(5, 3)
x
```

创建一个5*3的随机矩阵
并显示它（Shift+Enter）

Out[2]:
```
0.5311 0.7043 0.1347
0.3073 0.2343 0.4198
0.0552 0.8724 0.9396
0.9530 0.6955 0.1635
0.9159 0.4498 0.5068
[torch.FloatTensor of size 5x3]
```

Tensor的运算

In [3]:

```
y = torch.ones (5, 3)
y
```

创建一个5*3的全1矩阵
并显示它（Shift+Enter）

Tensor的运算

In [3]:

```
y = torch.ones (5, 3)
y
```

Out[3]:

```
1 1 1
1 1 1
1 1 1
1 1 1
1 1 1
[torch.FloatTensor of size 5x3]
```

创建一个5*3的全1矩阵
并显示它（Shift+Enter）

Tensor的运算

In [3]:

```
y = torch.ones (5, 3)
y
```

Out[3]:

```
1 1 1
1 1 1
1 1 1
1 1 1
1 1 1
[torch.FloatTensor of size 5x3]
```

创建一个5*3的全1矩阵
并显示它（Shift+Enter）

In [4]:

```
z = x + y
z
```

计算两个矩阵相加

Tensor的运算

In [3]:
```
y = torch.ones (5, 3)
y
```

Out[3]:
```
1 1 1
1 1 1
1 1 1
1 1 1
1 1 1
[torch.FloatTensor of size 5x3]
```

创建一个5*3的全1矩阵
并显示它（Shift+Enter）

In [4]:
```
z = x + y
z
```

Out[4]:
```
1.0377 1.0584 1.8994
1.0033 1.7044 1.6078
1.7675 1.0331 1.1202
1.2346 1.3340 1.6460
1.6617 1.9530 1.6127
[torch.FloatTensor of size 5x3]
```

Tensor的运算

In [5]:
```
q = x.mm(y.t)
```

矩阵乘法　　矩阵转置

x =

0.5311 0.7043 0.1347
0.3073 0.2343 0.4198
0.0552 0.8724 0.9396
0.9530 0.6955 0.1635
0.9159 0.4498 0.5068

y =

1 1 1
1 1 1
1 1 1
1 1 1
1 1 1

y^T =

1 1 1 1 1
1 1 1 1 1
1 1 1 1 1

=

1.3701 1.3701 1.3701 1.3701 1.3701
0.9614 0.9614 0.9614 0.9614 0.9614
1.8672 1.8672 1.8672 1.8672 1.8672
1.8120 1.8120 1.8120 1.8120 1.8120
1.8725 1.8725 1.8725 1.8725 1.8725

关于Tensor

- 所有Numpy上面关于ndarray的运算全部可以应用于Tensor
  - 有关Tensor的全部运算，参见:
  - http://pytorch.org/docs/master/tensors.html

关于Tensor

- 所有Numpy上面关于ndarray的运算全部可以应用于Tensor
- 从Numpy到Tensor的转换： torch.from_numpy(a)
- 从Tensor到Numpy的转换： a.numpy()

关于Tensor

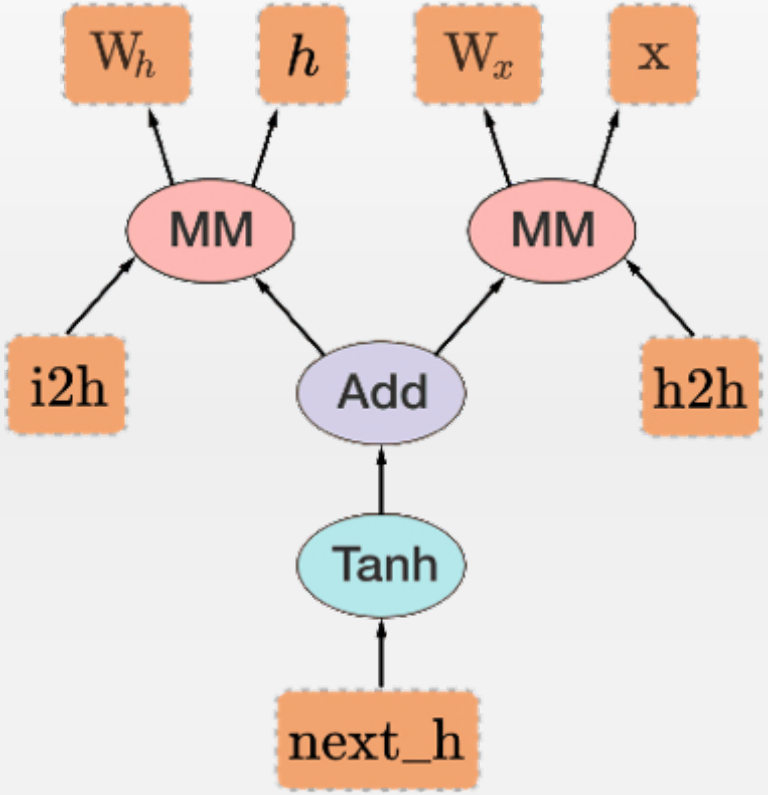
- 所有Numpy上面关于ndarray的运算全部可以应用于Tensor
- 从Numpy到Tensor的转换： torch.from_numpy(a)
- 从Tensor到Numpy的转换： a.numpy()
- Tensor与Numpy的最大不同： Tensor可以在GPU上运算

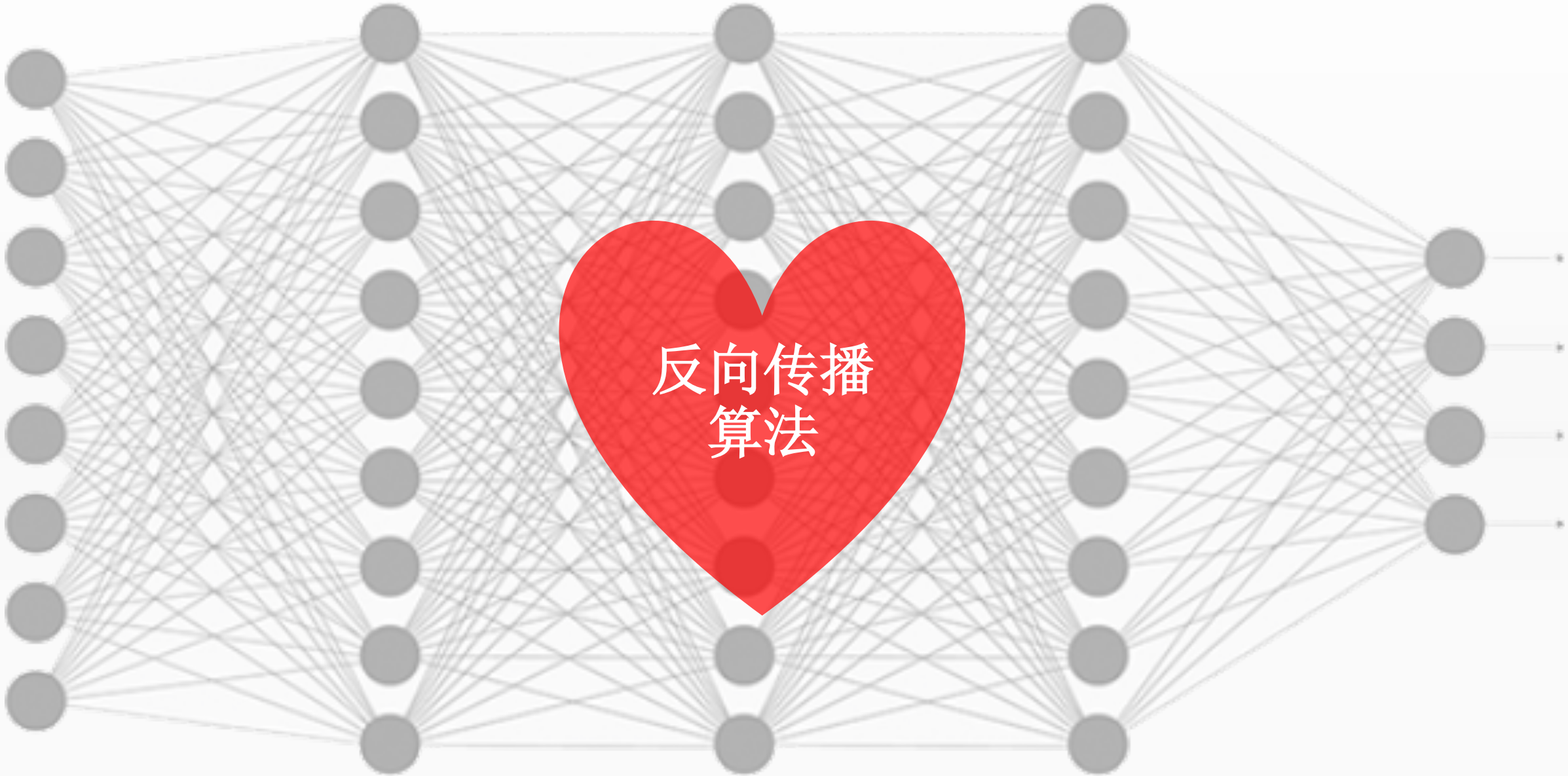
In [8]:

```
if torch.cuda.is_available():
    x = x.cuda()
    y = y.cuda()
    print(x + y)
```

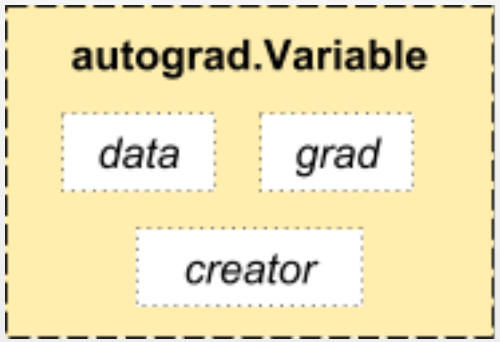
动态计算图

- 动态运算图（Dynamic Computation Graph）是PyTorch的最主要特性
- 它可以让我们的计算模型更灵活、复杂
- 它可以让反向传播算法随时进行




反向传播
算法

自动微分变量



定义一个自动微分变量

In [9]:
```
from torch.autograd import Variable
```

In [10]:
```
x = Variable(torch.ones(2, 2), requires_grad=True)
x
```

Out[10]:
```
Variable containing:
 1 1
 1 1
[torch.FloatTensor of size 2x2]
```

x

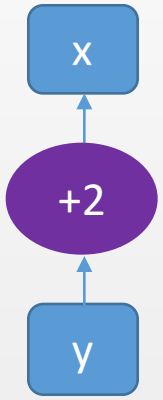
自动微分变量

In [9]:
```
from torch.autograd import Variable
```

In [10]:
```
x = Variable(torch.ones(2, 2), requires_grad=True)
x
```

In [11]:
```
y = x + 2
y.creator
```

Out[11]:
```
<torch.autograd._functions.basic_ops.AddConstant at
0x1126bef28>
```



自动微分变量

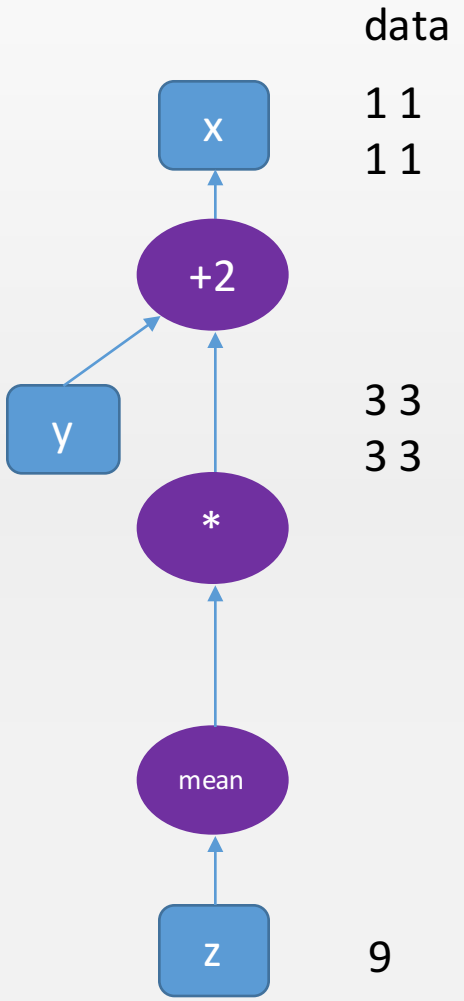
In [9]:
```
from torch.autograd import Variable
```

In [10]:
```
x = Variable(torch.ones(2, 2), requires_grad=True)
x
```

In [11]:
```
y = x + 2
y.creator
```

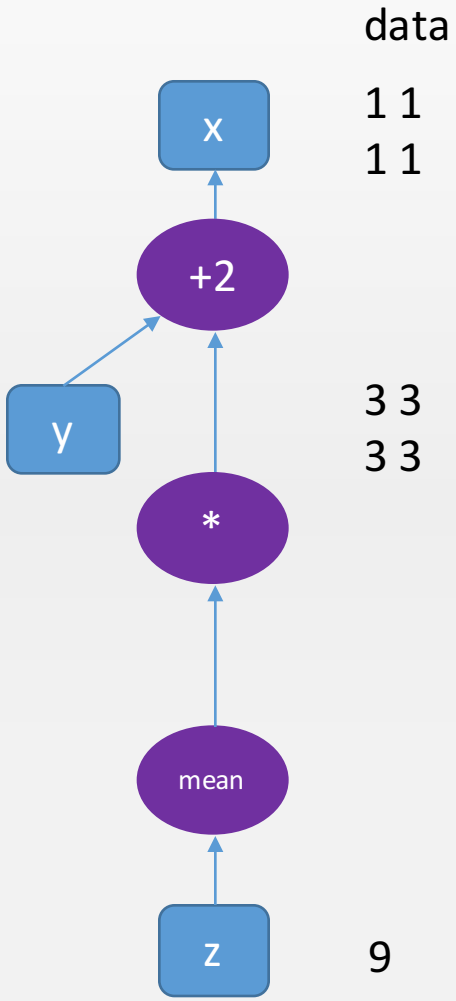
In [13]:
```
z = torch.mean(y * y)
z.data
```

Out[13]:
```
9
[torch.FloatTensor of size 1x1]
```



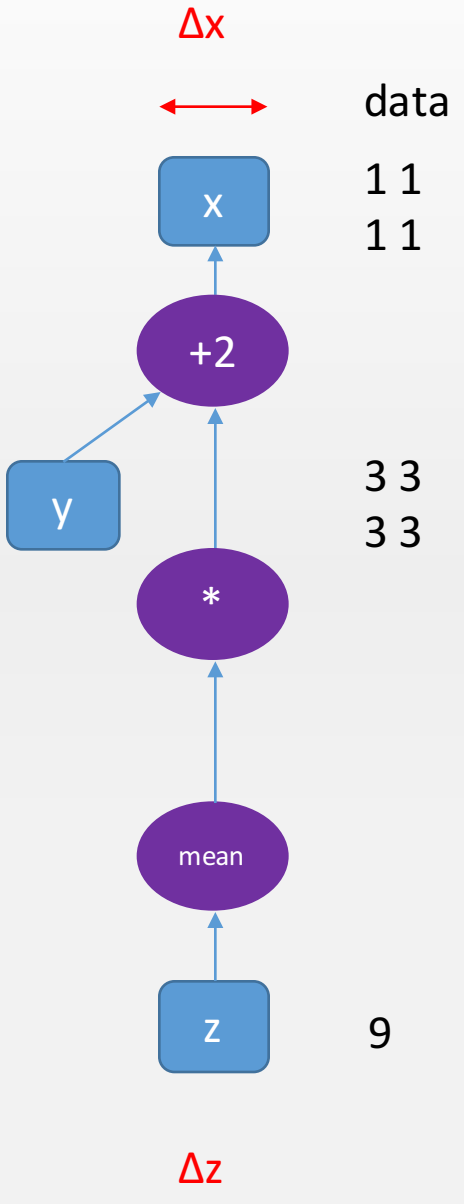
多层运算

- z=m((x+2)*(x+2))
- z=m(f(g(x))), $m(x)=\frac{\sum x_i}{N}$ $f(x)=x\cdot x$ $g(x)=x+2$
- 一个多层神经网络



梯度信息

- z=m((x+2)*(x+2))
- z=m(f(g(x))), $m(x)=\frac{\sum x_i}{N}$ $f(x)=x\cdot x$ $g(x)=x+2$
- 一个多层神经网络
- 当x发生小变化，会引起z端多大变化？
- 观察到z端的变化，推测x端发生了多大变化？



计算梯度与反向传播

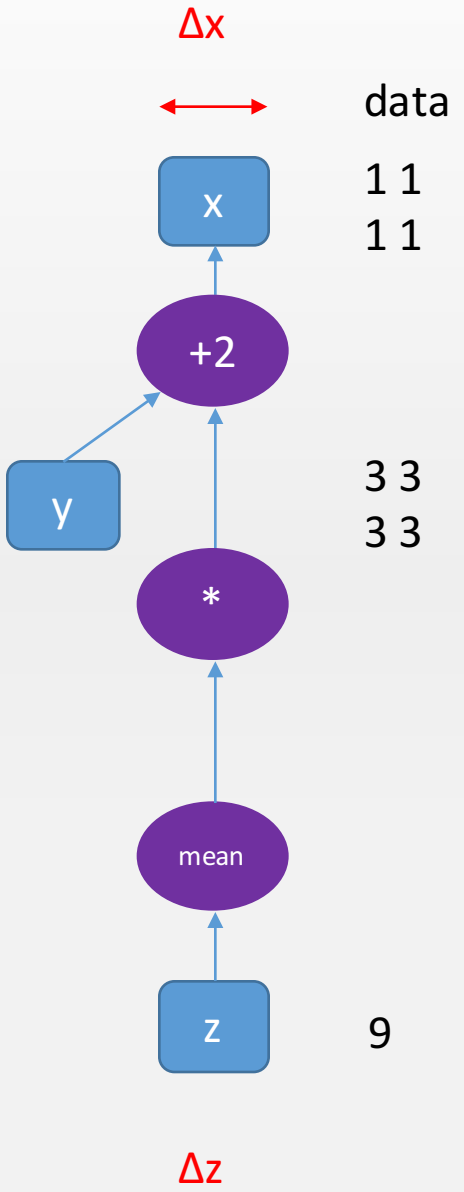
- z=m((x+2)*(x+2))
- z=m(f(g(x))), $m(x)=\frac{\sum x_i}{N}$ $f(x)=x\cdot x$ $g(x)=x+2$

计算梯度：

$$\left(\frac{\partial z}{\partial x}\right)\Big|_{x=\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix}} = \left(\frac{\partial m}{\partial f}\cdot\frac{\partial f}{\partial g}\cdot\frac{\partial g}{\partial x}\right)\Big|_{x=\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix}}$$

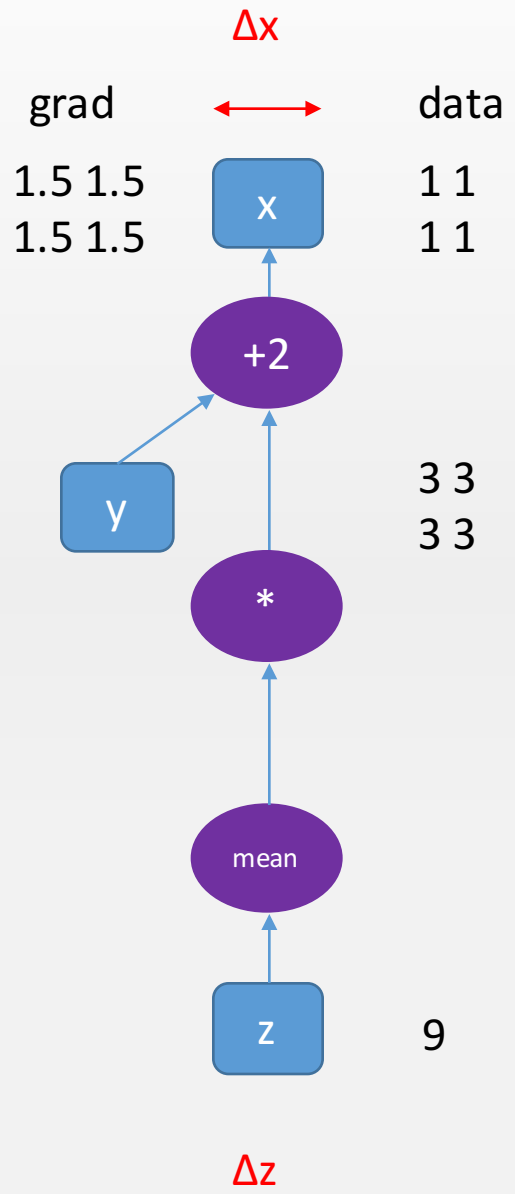
$$= \left(\frac{1}{N}\cdot 2g\cdot 1\right)\Big|_{x=\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix}}$$

$$= \begin{pmatrix}1.5 & 1.5\\ 1.5 & 1.5\end{pmatrix}$$



求导运算与反向传播

- z=m((x+2)*(x+2))
- z=m(f(g(x))), $m(x)=\frac{\sum x_i}{N}$ $f(x)=x\cdot x$ $g(x)=x+2$



In [14]:

```
z.backward()
print(z.grad)
print(y.grad)
print(x.grad)
```

Out[14]:

```
None
None
Variable containing:
1.5000 1.5000
1.5000 1.5000
[torch.FloatTensor of size 2x2]
```

更疯狂的函数依赖……

In [15]:

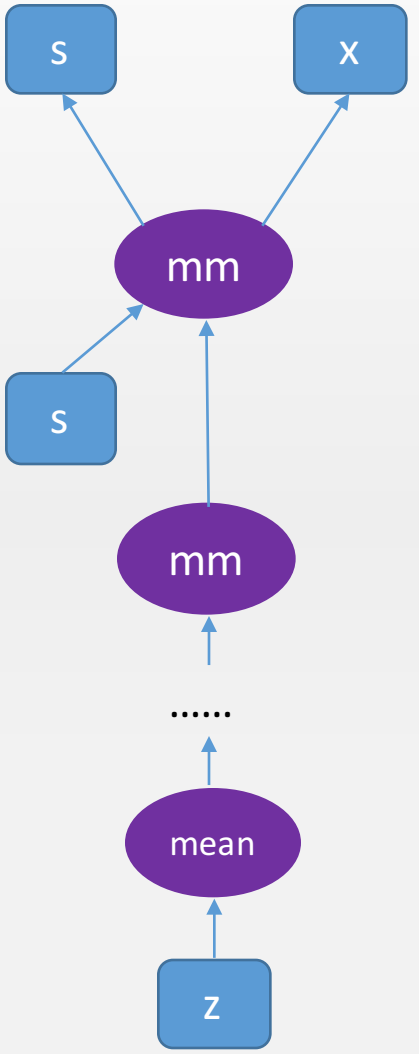
```
s = Variable(torch.FloatTensor([[0.01, 0.02]]),
                                        requires_grad = True)
x = Variable(torch.ones(2, 2), requires_grad = True)
for i in range(10):
    s = s.mm(x)
z = torch.mean(s)
```

In [16]:

```
z.backward()
print(x.grad)
print(s.grad)
```

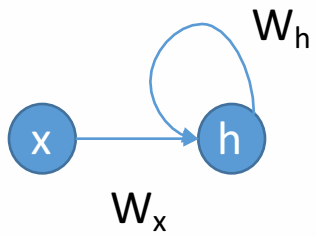
Out[16]:

```
Variable containing:
 37.1200 37.1200
 39.6800 39.6800
[torch.FloatTensor of size 2x2]

None
```



一个动态生成的神经网络

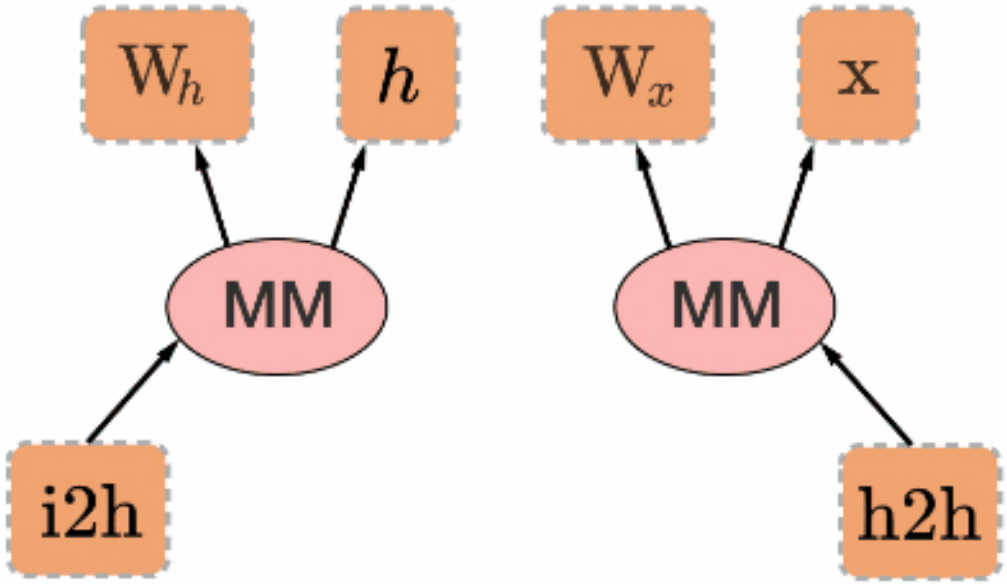

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```

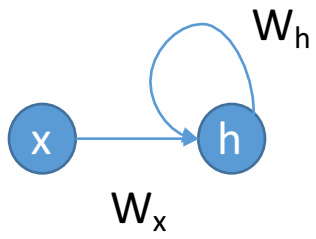


$$h_{t+1} = Tanh(W_h \cdot h + W_x \cdot x)$$

一个动态生成的神经网络

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))

i2h = torch.mm(W_x, x.t())
h2h = torch.mm(W_h, prev_h.t())
```

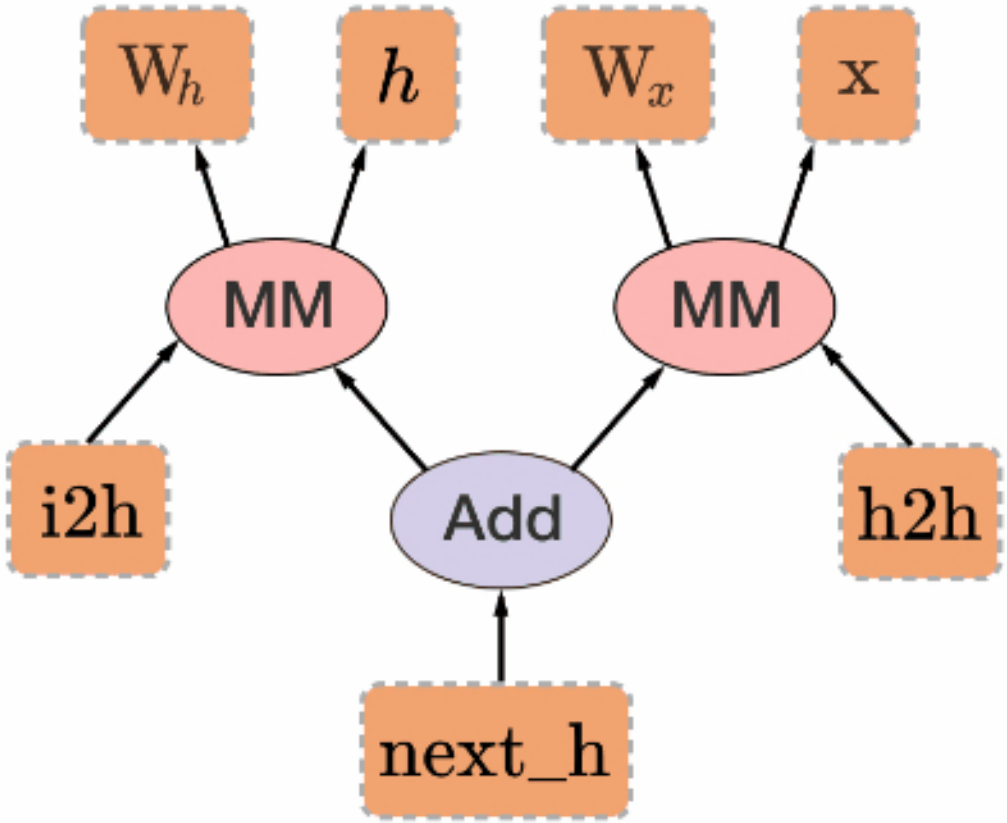


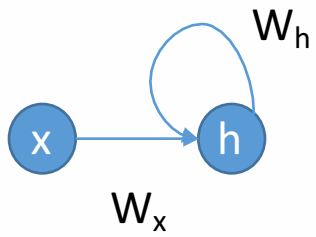


$$h_{t+1} = Tanh(W_h \cdot h + W_x \cdot x)$$

一个动态生成的神经网络

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))

i2h = torch.mm(W_x, x.t())
h2h = torch.mm(W_h, prev_h.t())
next_h = i2h + h2h
```

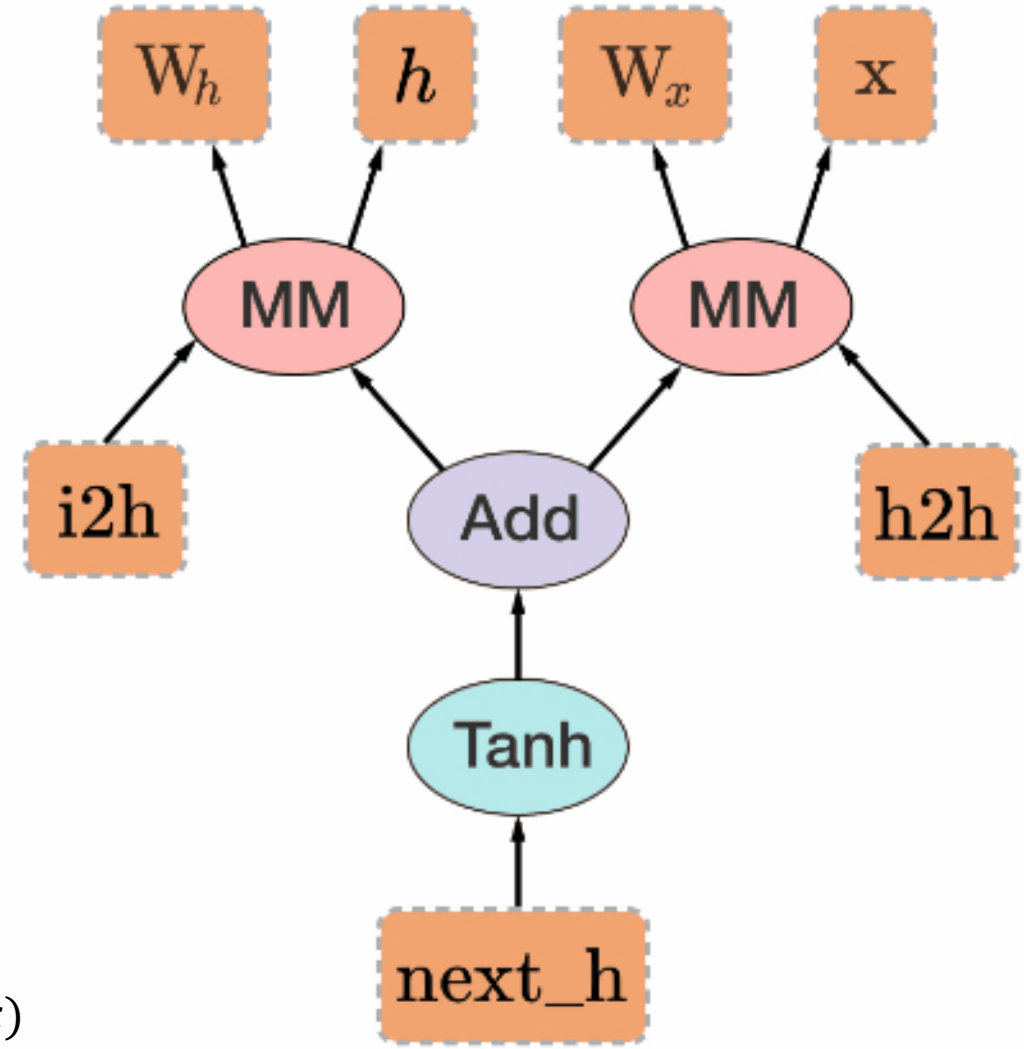


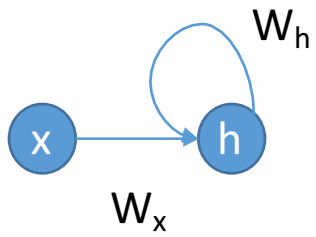


$$h_{t+1} = Tanh(W_h \cdot h + W_x \cdot x)$$

一个动态生成的神经网络

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))

i2h = torch.mm(W_x, x.t())
h2h = torch.mm(W_h, prev_h.t())
next_h = i2h + h2h
next_h = next_h.tanh()
```

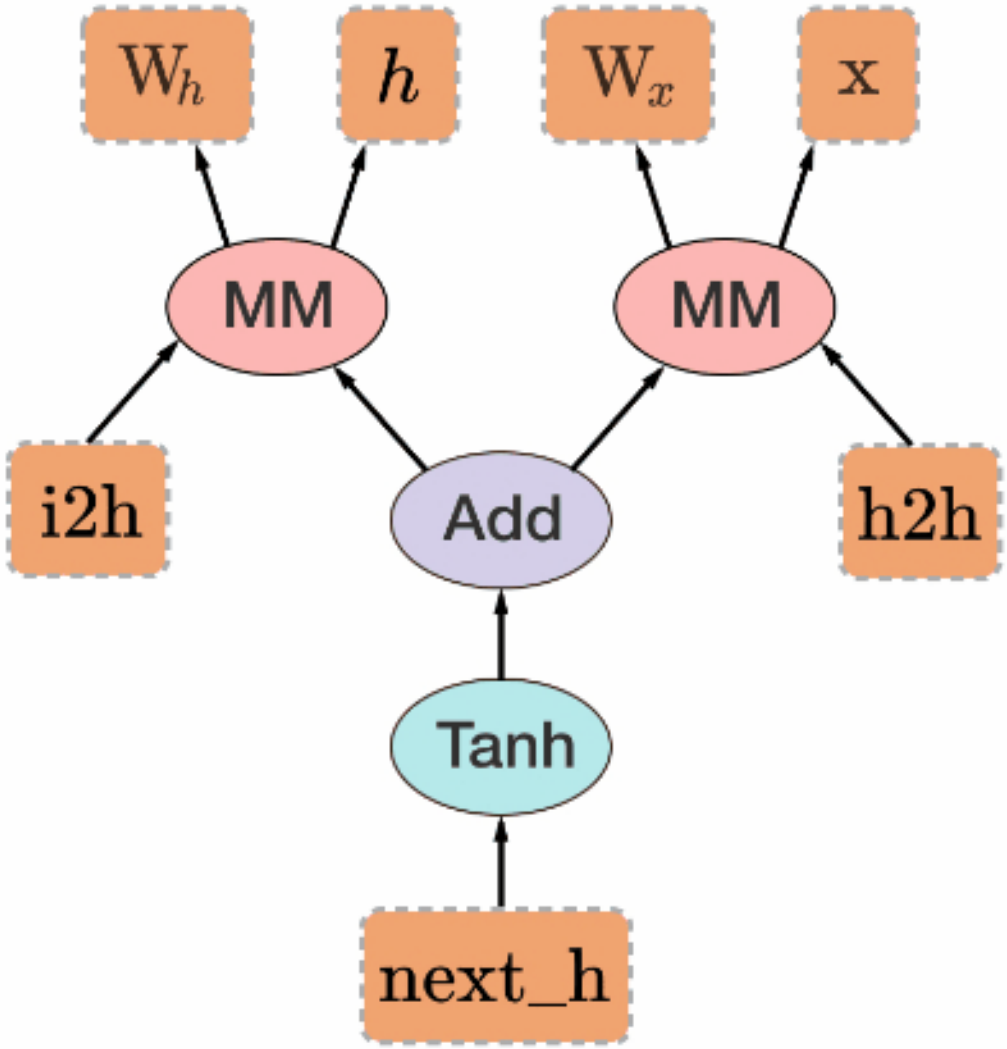




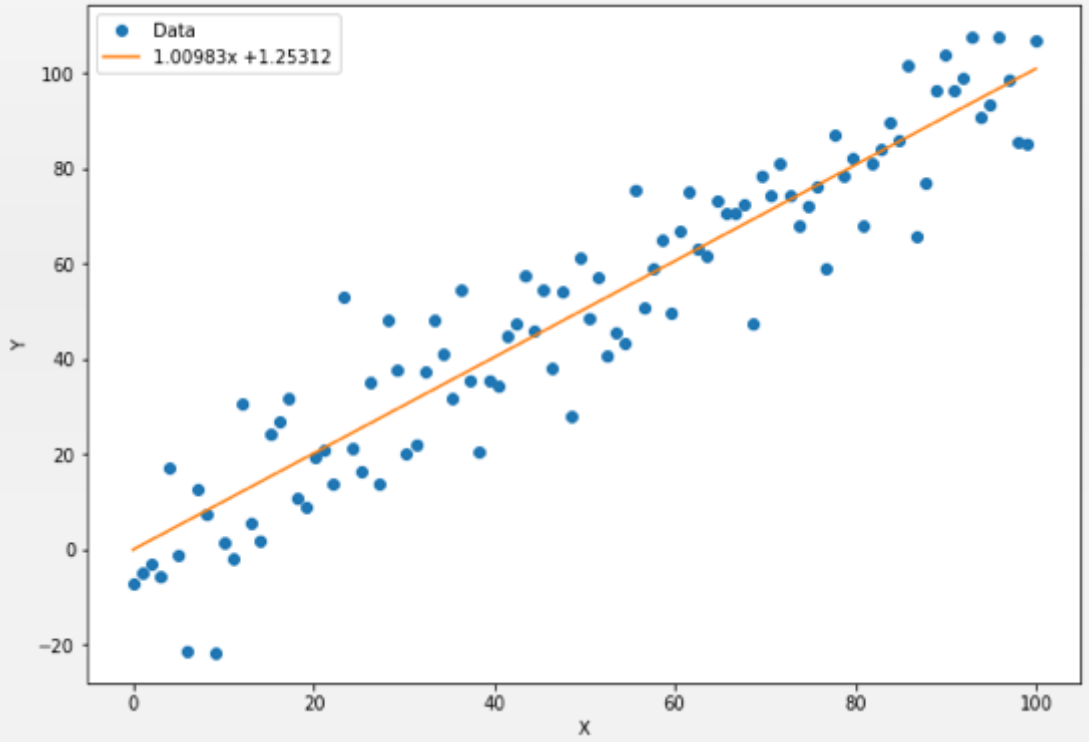
$$h_{t+1} = Tanh(W_h \cdot h + W_x \cdot x)$$

在动态图上反向传播

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))

i2h = torch.mm(W_x, x.t())
h2h = torch.mm(W_h, prev_h.t())
next_h = i2h + h2h
next_h = next_h.tanh()

next_h.backward(torch.ones(1, 20))
```

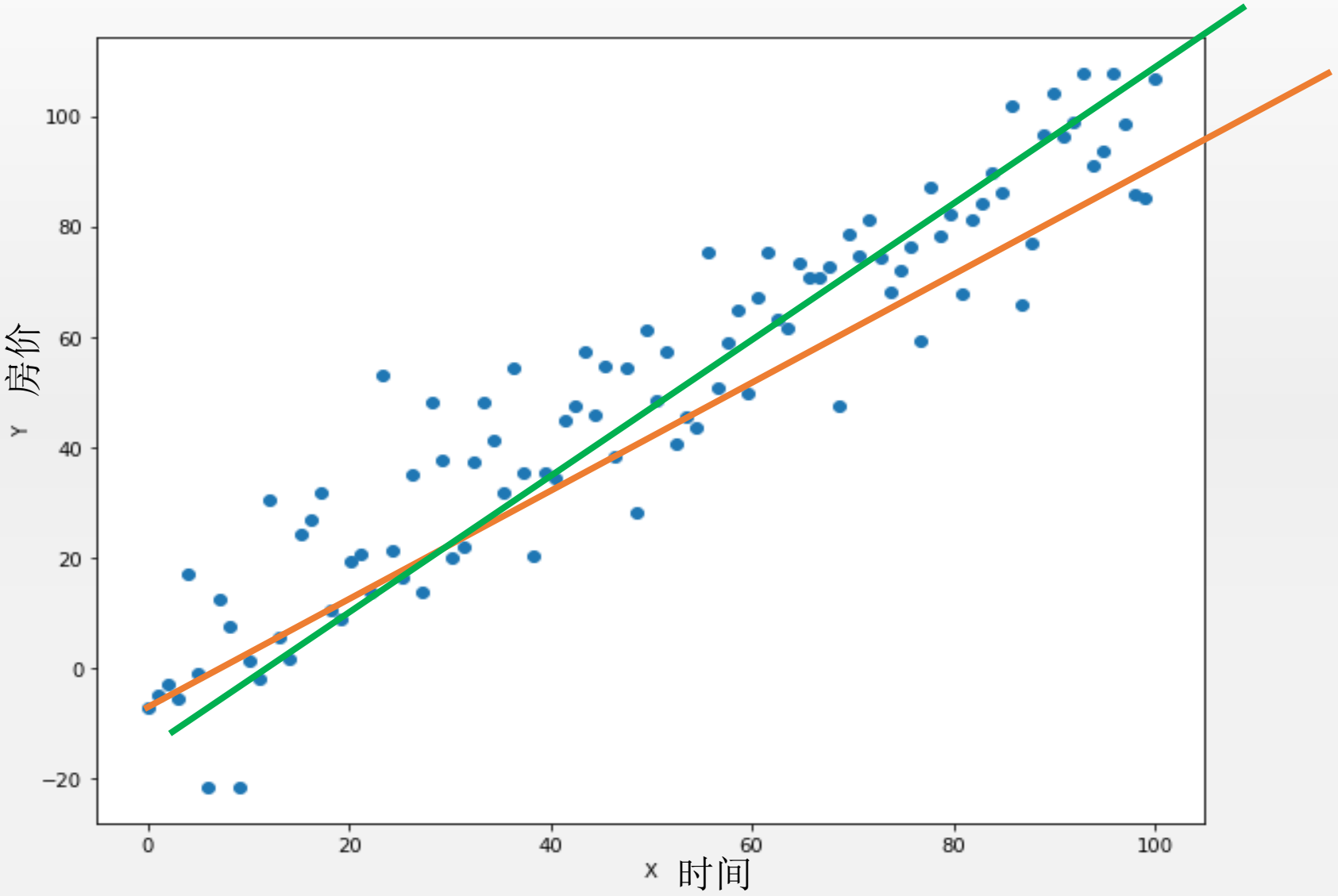


The Incredible PYTORCH



第一个机器学习程序：线性回归

问题描述



每一个点就是一个样本

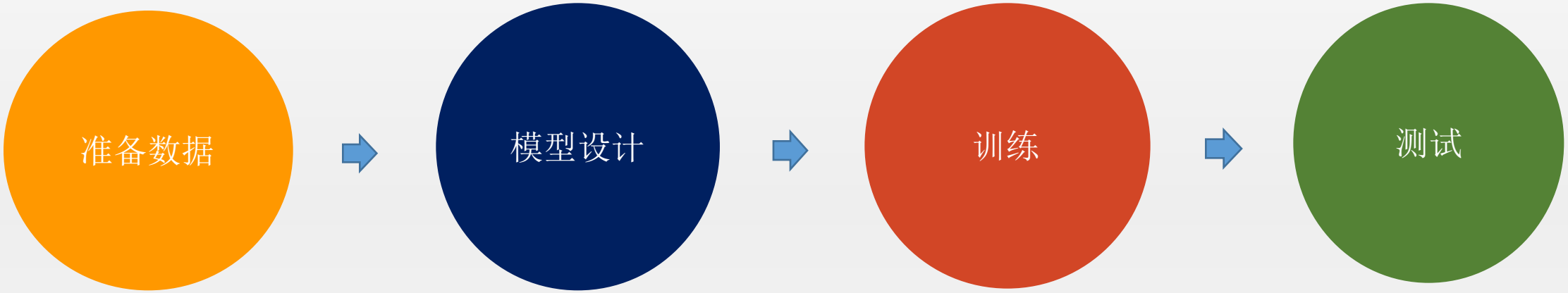
X是特征，Y是标签

直线是模型Y = a X + b, a,b是参数

训练阶段：根据样本得到最优模型（参数a,b）

测试（预测）阶段：使用模型计算新特征点（如110）的数值

步骤



准备数据：人为生成数据

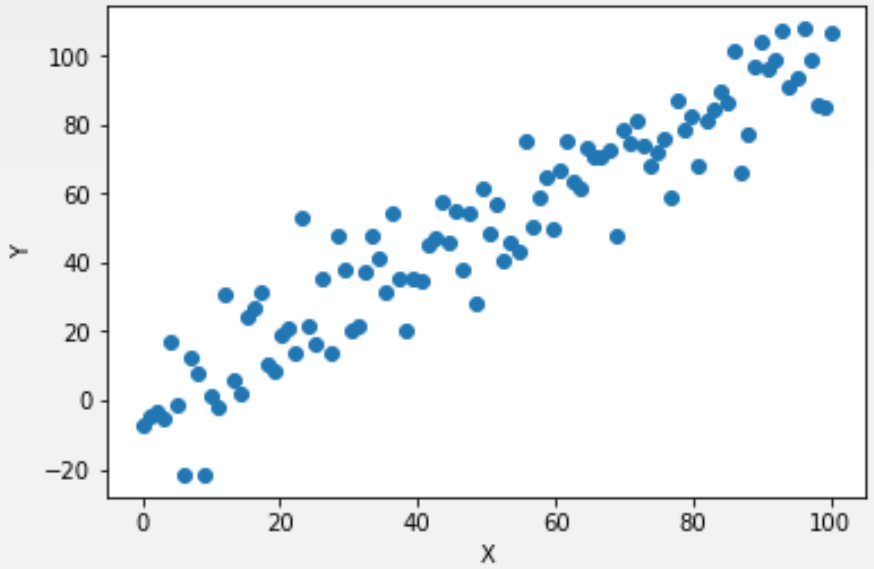
In [15]:
```
x = Variable(torch.linspace(0, 100).type(torch.FloatTensor))
rand = Variable(torch.randn(100)) * 10
y = x + rand
```

生成100个0-100间的数

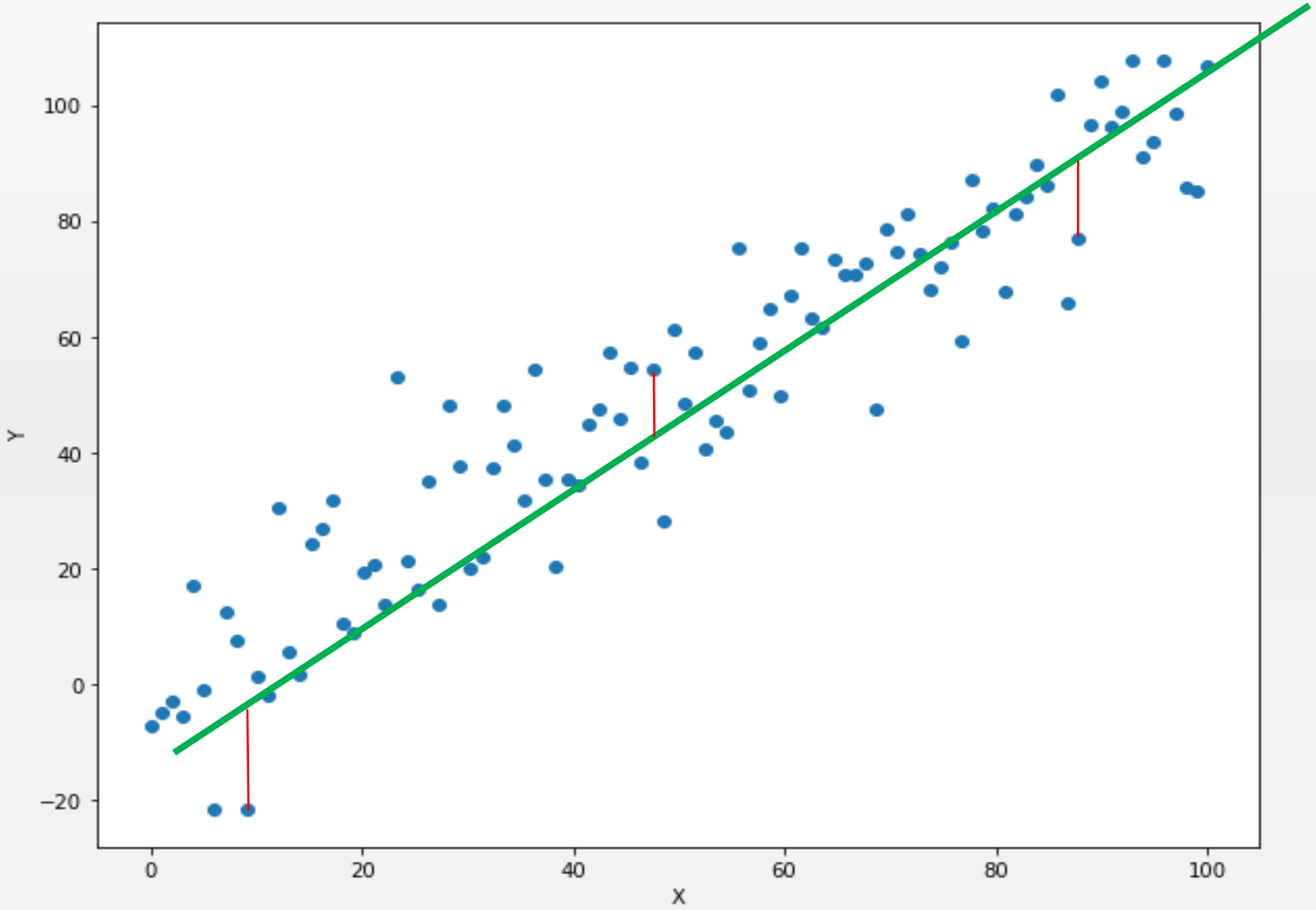
生成100个正态分布随机数，均值为0，方差为10

In [16]:
```
import matplotlib.pyplot as plt
plt.plot(x.data.numpy(), y.data.numpy(),'o')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

将Variable转换为数组，绘图



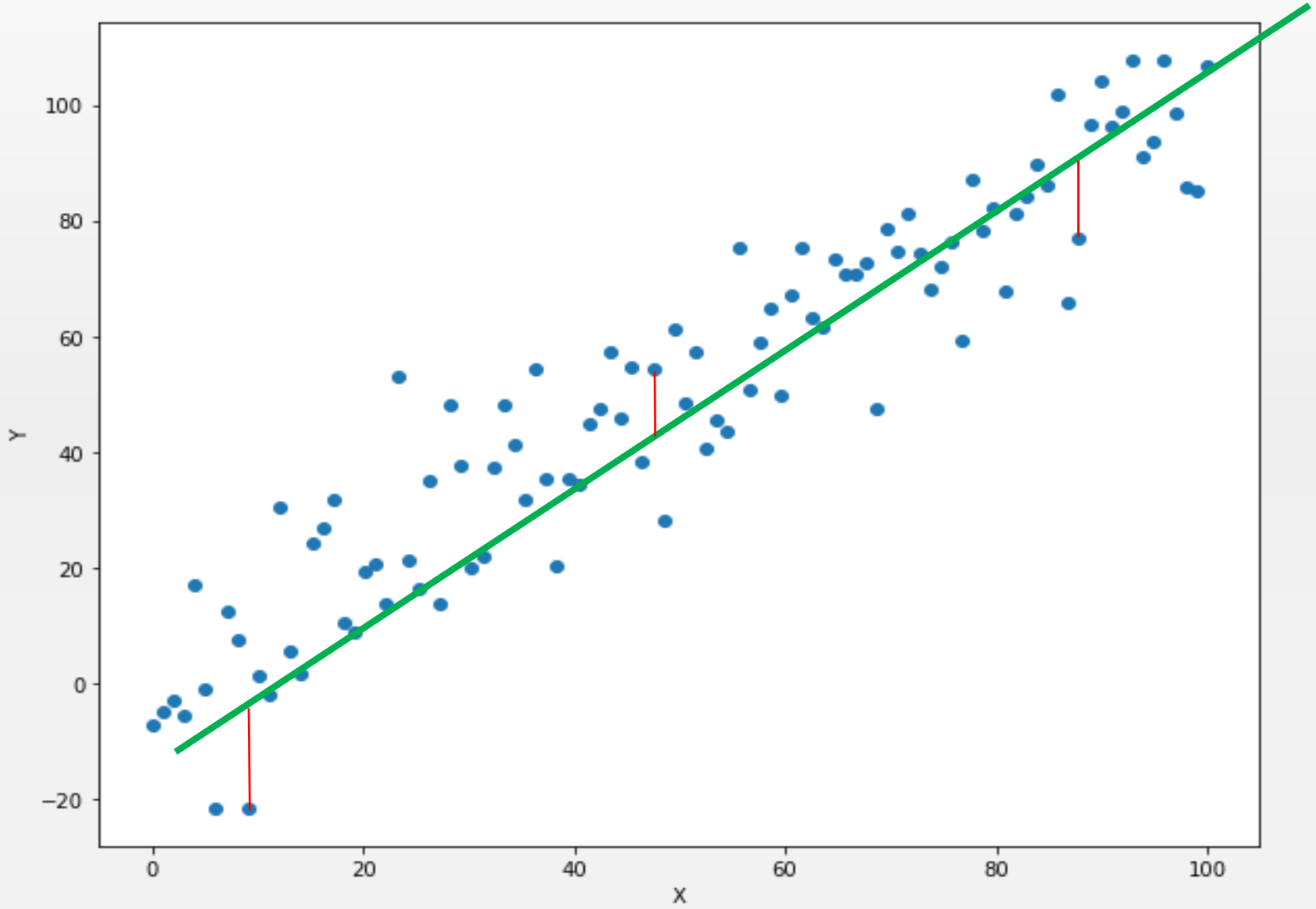
线性回归



找到一条直线，使得它到所有点的距离都很小

$$L = \frac{1}{N}\sum_{i=1}^{N}(aX_i + b - Y_i)^2$$

线性回归

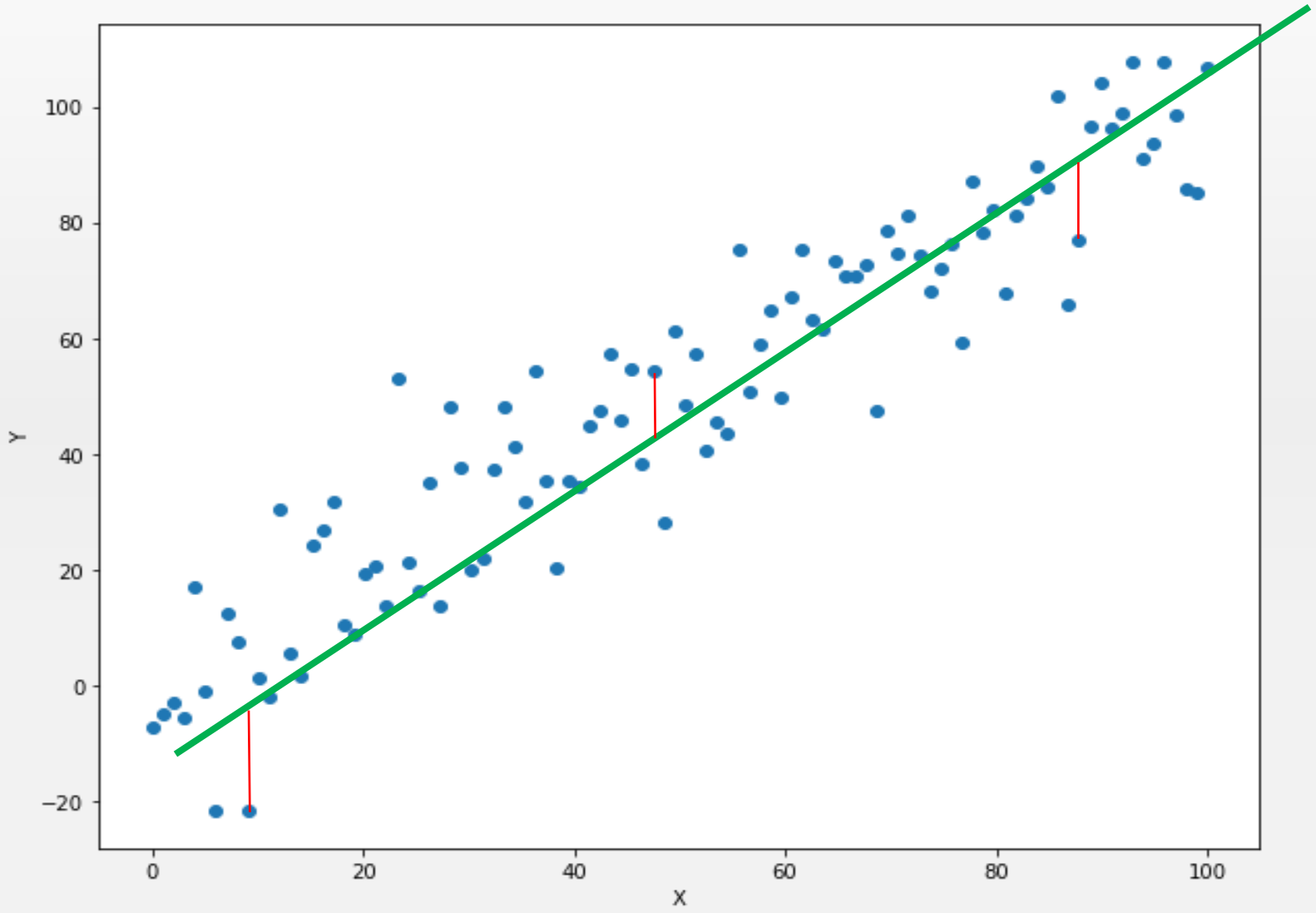


找到一条直线，使得它到所有点的距离都很小

$$L = \frac{1}{N}\sum_{i=1}^{N}(aX_i + b - Y_i)^2$$

$$\min_{a,b} L(a,b)$$

线性回归



找到一条直线，使得它到所有点的距离都很小

$$L = \frac{1}{N}\sum_{i=1}^{N}(aX_i + b - Y_i)^2$$

计算梯度：$\frac{\partial L}{\partial a}, \frac{\partial L}{\partial b}$

梯度下降算法：

$$a_{t+1} = a_t - \alpha \cdot \frac{\partial L}{\partial a}$$

$$b_{t+1} = b_t - \alpha \cdot \frac{\partial L}{\partial b}$$

训练模型代码

In [17]:

```
a = Variable(torch.rand(1), requires_grad = True)
b = Variable(torch.rand(1), requires_grad = True)
print('Initial parameters:', [a, b])
learning_rate = 0.0001
for i in range(1000):
    if (a.grad is not None) and (b.grad is not None):
        a.grad.data.zero_()
        b.grad.data.zero_()
    predictions = a.expand_as(x) * x + b.expand_as(x)
    loss = torch.mean((predictions - y) ** 2)
    print('loss:', loss)
    loss.backward()
    a.data.add_(- learning_rate * a.grad.data)
    b.data.add_(- learning_rate * b.grad.data)
```

设置拟合参数变量

设置学习率

清空变量中的梯度数据

计算出预测数值

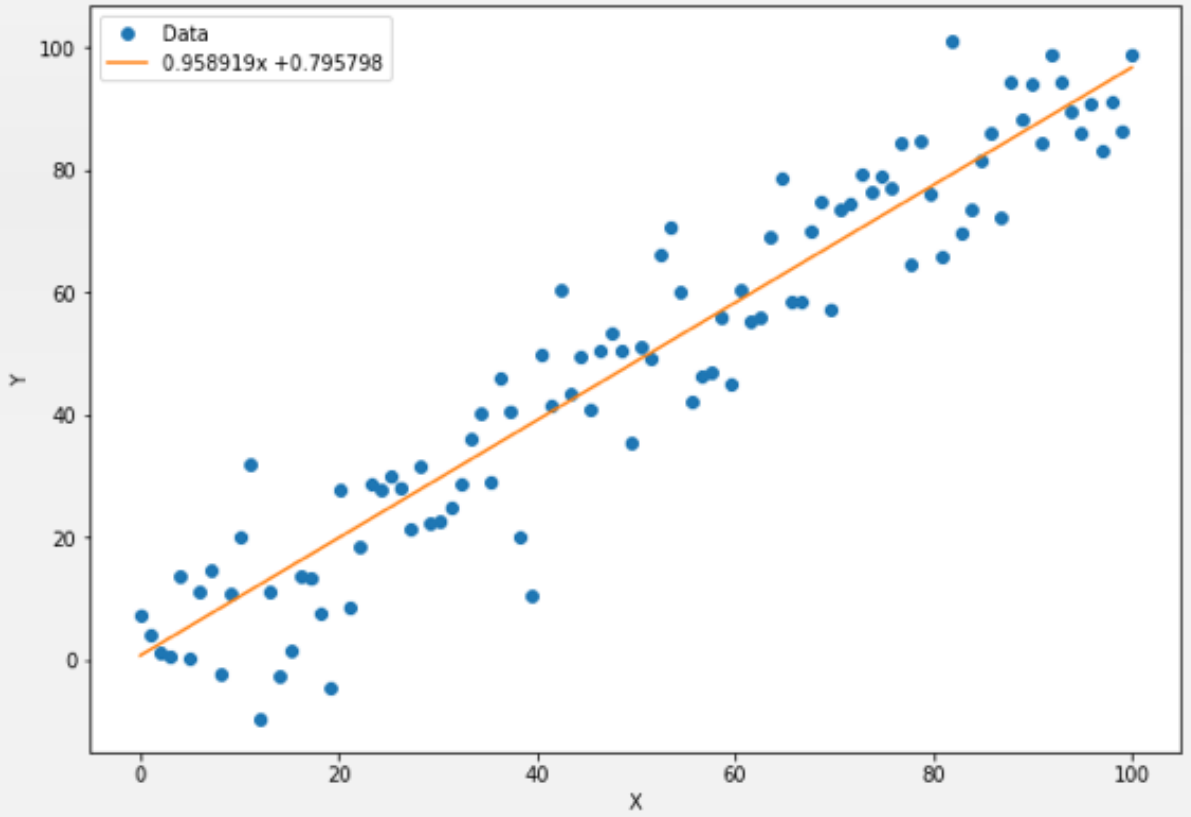
计算误差

反向传播

参数的更新

In [18]:

```
x_data = x.data.numpy()
plt.figure(figsize  = (10, 7))
xplot,= plt.plot(x_data, y.data.numpy(), 'o')
yplot, = plt.plot(x_data, a.data.numpy() * x_data + b.data.numpy())
plt.xlabel('X')
plt.ylabel('Y')
str1 = str(a.data.numpy()[0]) + 'x +' + str(b.data.numpy()[0])
plt.legend([xplot, yplot],['Data', str1])
plt.show()
```



绘制图形

测试模型代码

In [16]:

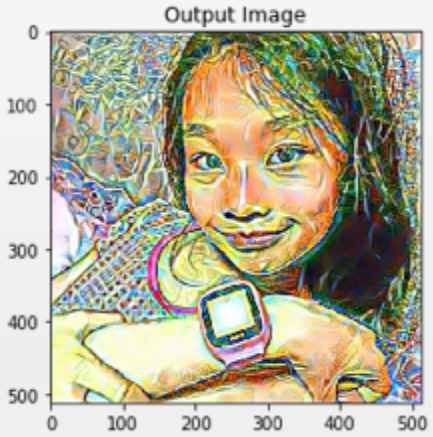
```
x_test = Variable(torch.FloatTensor([1, 2, 10, 100, 1000]))
predictions = a.expand_as(x_test) * x_test + b.expand_as(x_test)
predictions
```

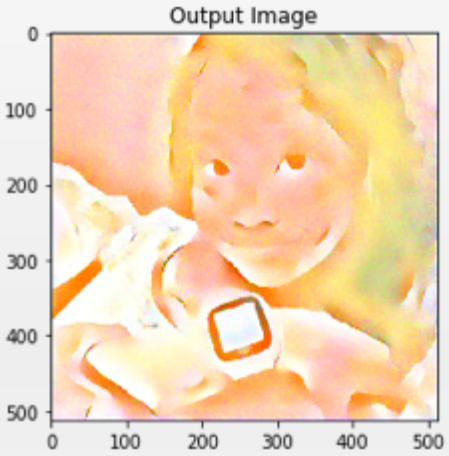
Out[16]:

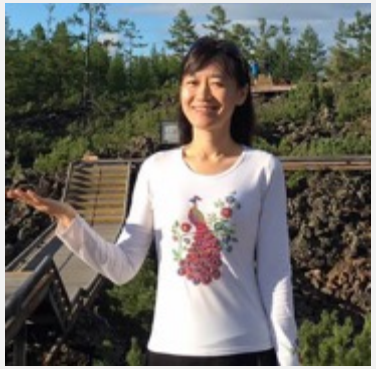
```
Variable containing:
1.7547
2.7136
10.3850
96.6877
959.7147
[torch.FloatTensor of size 100]
```

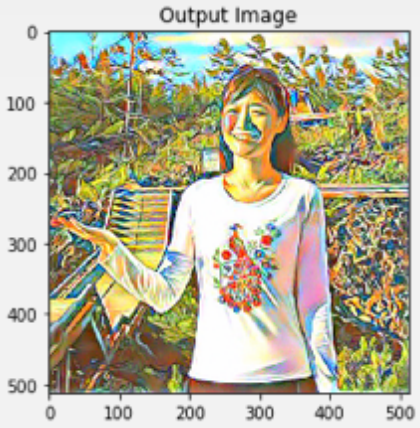
风格

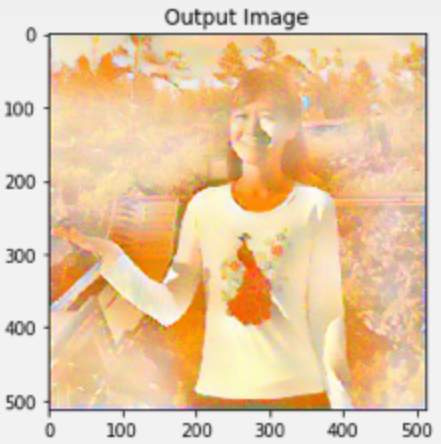
内容

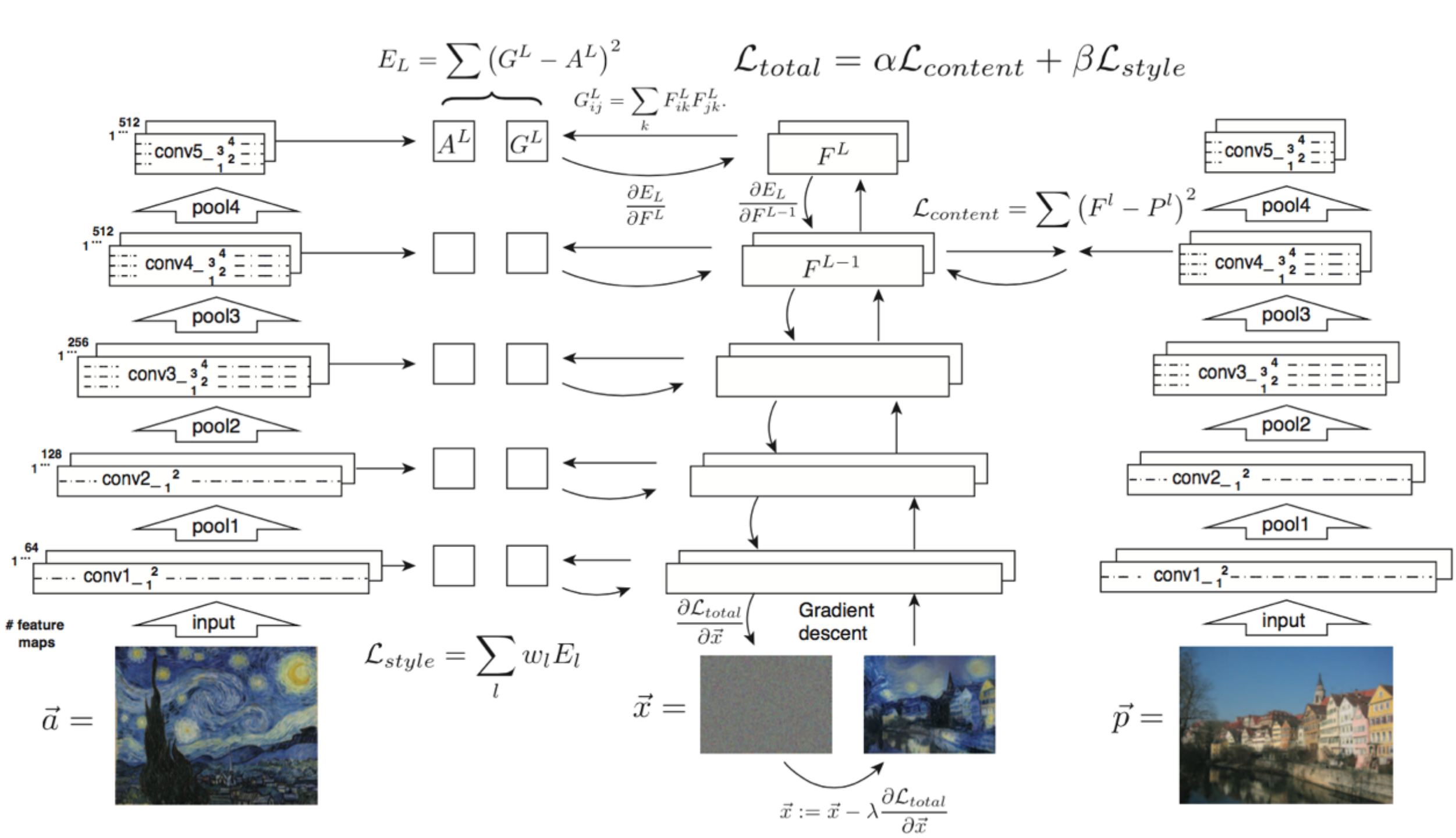

Output Image
0
100
200
300
400
500
0
100
200
300
400
500


Output Image
0
100
200
300
400
500
0
100
200
300
400
500




Output Image
0
100
200
300
400
500
0
100
200
300
400
500


Output Image
0
100
200
300
400
500
0
100
200
300
400
500


$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik} F^L_{jk}$.
A^L
G^L
F^L
F^{L-1}
$\frac{\partial E_L}{\partial F^L}$
$\frac{\partial E_L}{\partial F^{L-1}}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
$\vec{a} =$
$\vec{x} =$
$\vec{p} =$
feature maps
512
conv5_1
pool4
512
conv4_1
pool3
256
conv3_1
pool2
128
conv2_1
pool1
64
conv1_1
input

```python
class ContentLoss(nn.Module):

    def __init__(self, target, weight):
        super(ContentLoss, self).__init__()
        # we 'detach' the target content from the tree used
        self.target = target.detach() * weight
        # to dynamically compute the gradient: this is a stated value,
        # not a variable. Otherwise the forward method of the criterio
        # will throw an error.
        self.weight = weight
        self.criterion = nn.MSELoss()

    def forward(self, input):
        self.loss = self.criterion(input * self.weight, self.target)
        self.output = input
        return self.output

    def backward(self, retain_variables=True):
        self.loss.backward(retain_variables=retain_variables)
        return self.loss

class GramMatrix(nn.Module):

    def forward(self, input):
        a, b, c, d = input.size()  # a=batch size(=1)
        # b=number of feature maps
        # (c,d)=dimensions of a f. map (N=c*d)

        features = input.view(a * b, c * d)  # resise F_XL into \hat F_

        G = torch.mm(features, features.t())  # compute the gram produc

        # we 'normalize' the values of the gram matrix
        # by dividing by the number of element in each feature maps.
        return G.div(a * b * c * d)
```

```python
def run_style_transfer(cnn, content_img, style_img, input_img, num_steps=300,
                       style_weight=1000, content_weight=1):
    """Run the style transfer."""
    print('Building the style transfer model..')
    model, style_losses, content_losses = get_style_model_and_losses(cnn,
        style_img, content_img, style_weight, content_weight)
    input_param, optimizer = get_input_param_optimizer(input_img)

    print('Optimizing..')
    run = [0]
    while run[0] <= num_steps:

        def closure():
            # correct the values of updated input image
            input_param.data.clamp_(0, 1)

            optimizer.zero_grad()
            model(input_param)
            style_score = 0
            content_score = 0

            for sl in style_losses:
                style_score += sl.backward()
            for cl in content_losses:
                content_score += cl.backward()

            run[0] += 1
            if run[0] % 50 == 0:
                print("run {}:".format(run))
                print('Style Loss : {:4f} Content Loss: {:4f}'.format(
                    style_score.data[0], content_score.data[0]))
                print()

            return style_score + content_score

        optimizer.step(closure)
```

StyleTransfer.ipynb

今日回顾

- 深度学习、机器学习
- 反向传播算法在神经网络中的作用
- 深度学习算法中的架构与训练方法
- 两种最重要的深度网络：CNN、RNN
- 深度学习的本质

- Pytorch的特点
- Tensor与numpy数组有何不同？
- 什么是动态计算图？
- 如何完成自动微分？

今日PyTorch主要指令

- 更多资料: 编程环境搭建视频
- Pytorch的官方教程: http://pytorch.org/tutorials/
- 吴莫烦的教程: https://morvanzhou.github.io/tutorials/machine-learning/torch/

火炬上的深度学习（上）

摩拜需要我

我卷卷卷

“移情别恋”

“镜像网络”与“猫鼠游戏”

Swarmer舞台