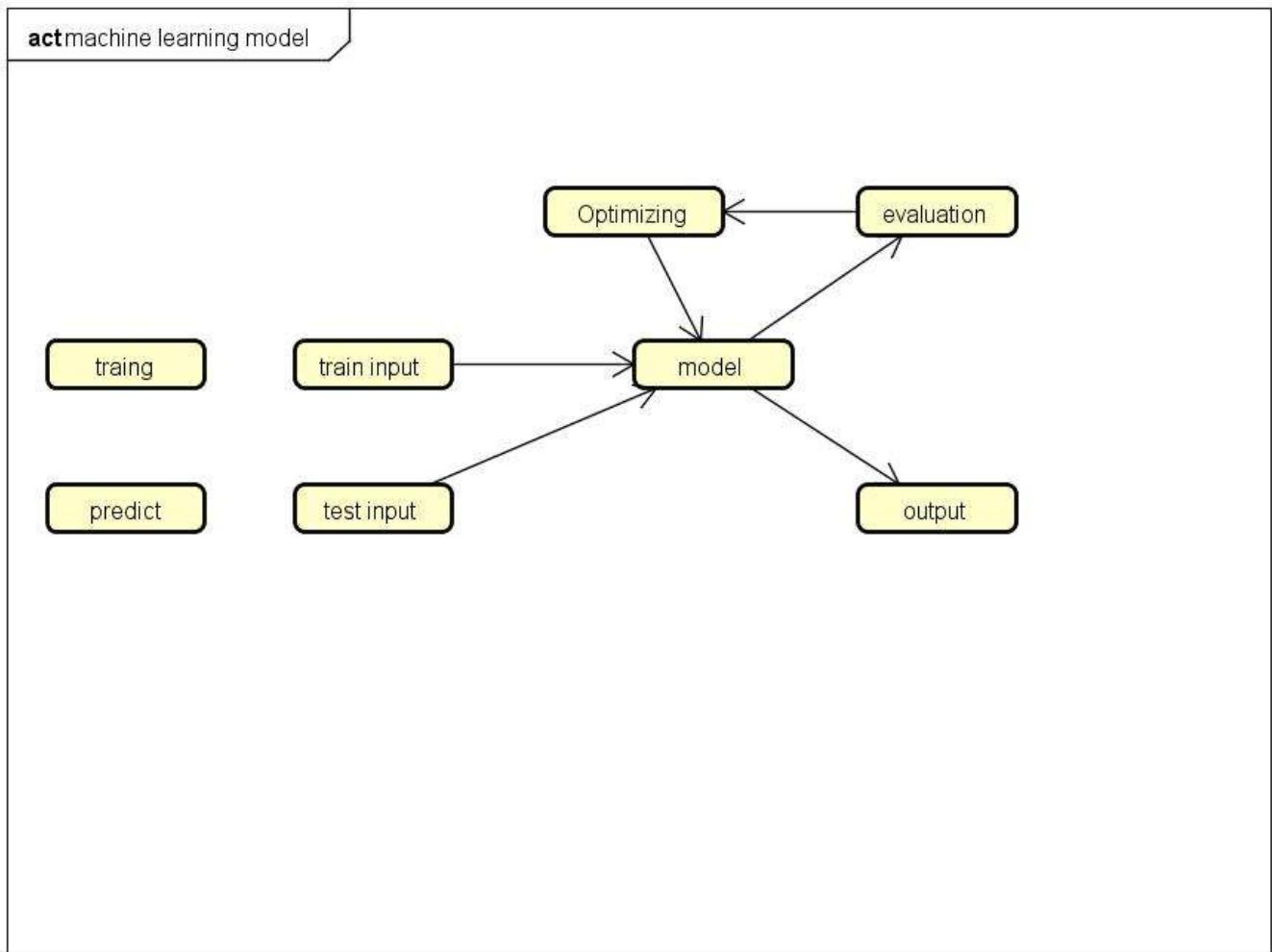
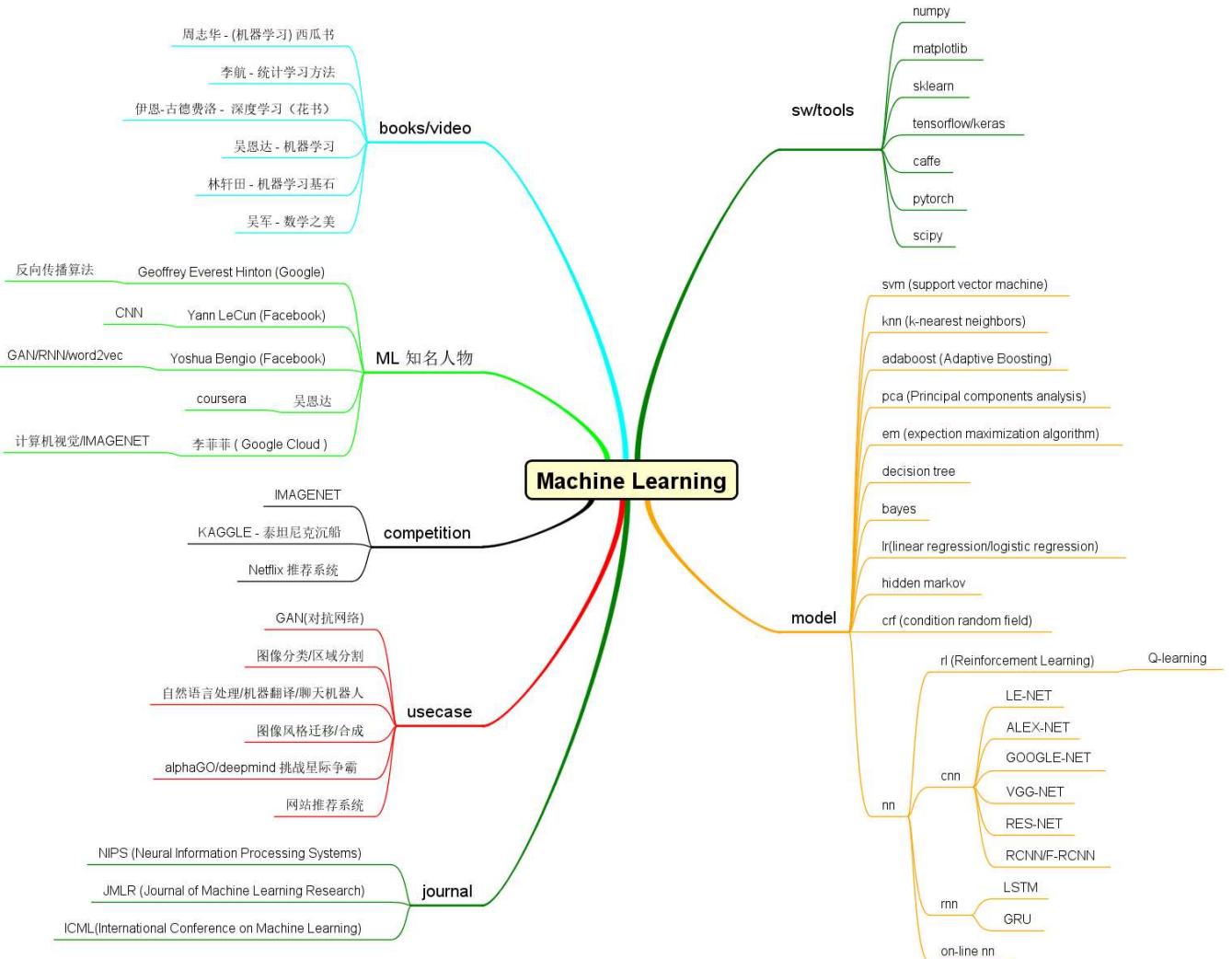


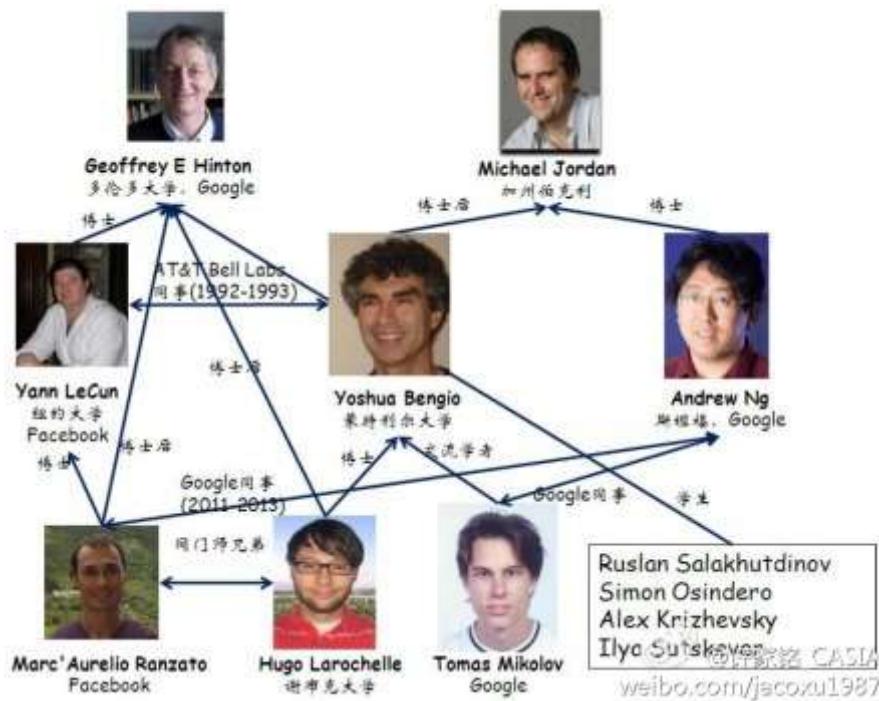
# Machine Learning Base introduce

来自 1998 年 Tom Mitchell 的定义， “对于某类任务T和性能度量P，如果一个计算机程序在T上以P衡量的性能随着经验E而自我完善，那么我们称这个计算机程序在从经验E学习。”



by Cloud.yu 2018-3-10





# Perceptron (感知机)

**任务**：求一条线  $wx + b = 0$  对不同类别的点进行二分类

**类型**：监督学习

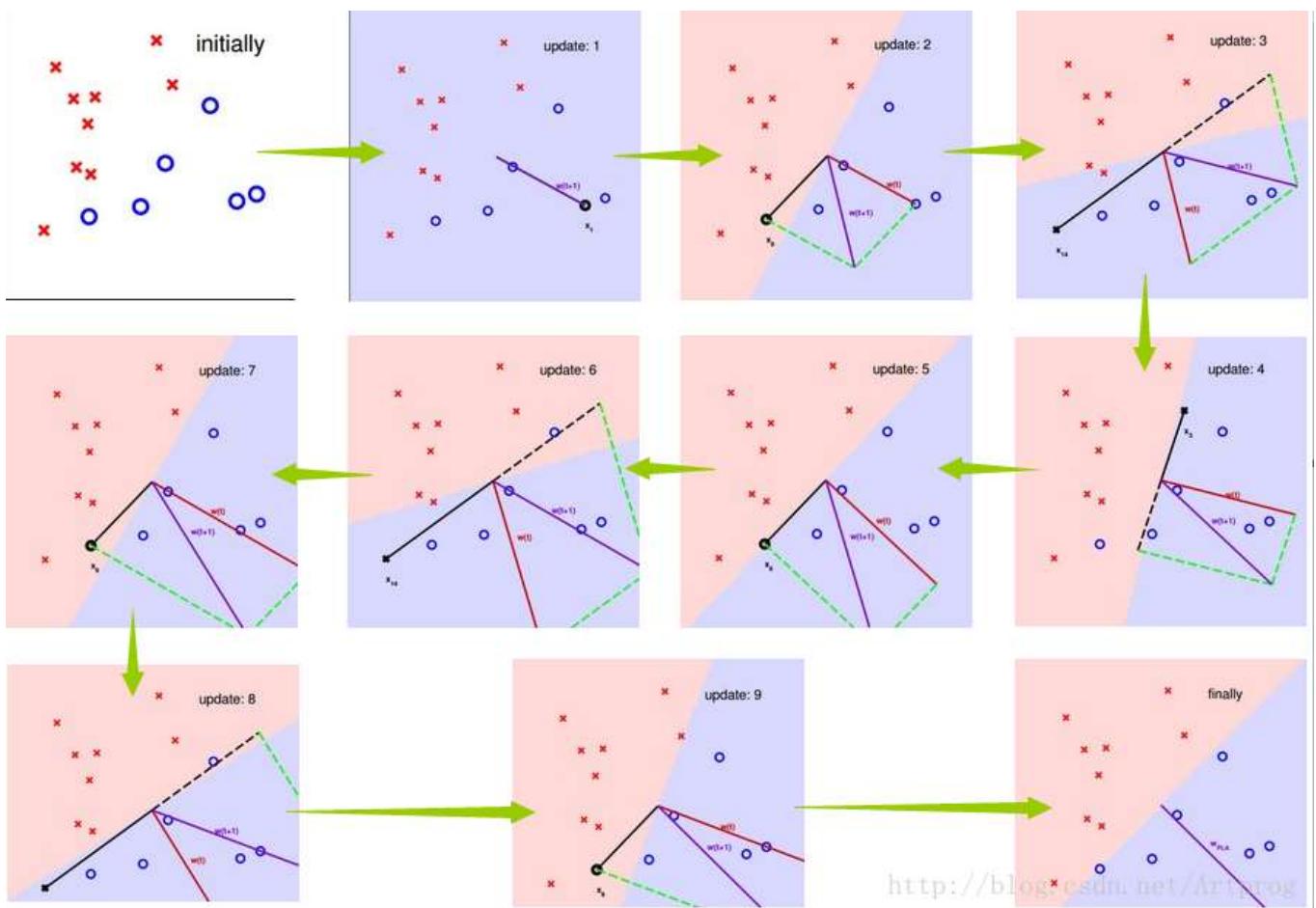
**模型**：感知机  $f(x) = \text{sign}(wx + b)$ , 其中  $w$  叫做权重(weight),  $b$  叫做偏置(bias)

**评价方法**： $\min\{\sum (y(wx + b) < 0)\}$

**优化方法**：PLA/Pocket

## PLA (Perceptron Learning Algorithm)

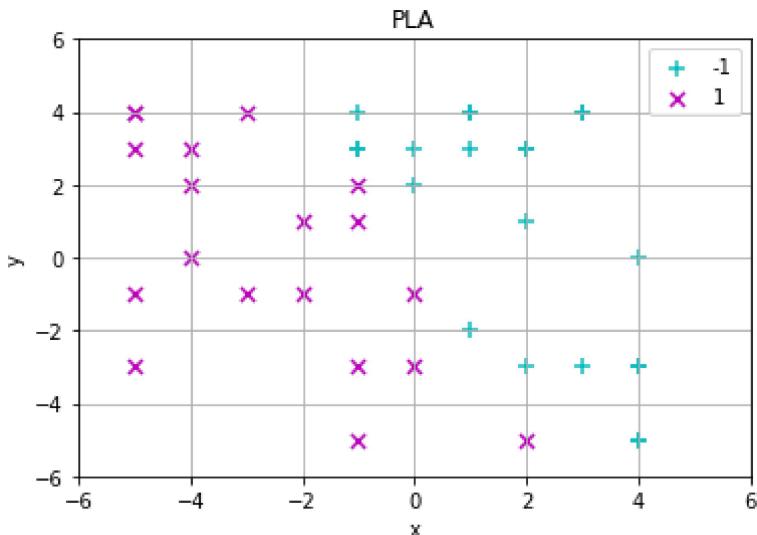
- **Input:** training data ( $T = \{(x_0, y_0), (x_1, y_1), \dots, (x_i, y_i)\}$ )
- **Output:**  $(w, b)$ 
  - init  $(w, b) \rightarrow (0, 0)$ ,  $i \rightarrow 0$
  - while  $i \neq \text{len}(T)$  do
    - if  $y_i(wx_i + b) \leq 0$  then
      - $(w, b) \rightarrow (w, b) + y_i x_i$
      - $i \rightarrow i + 1$
    - else
      - $i = i + 1$
    - end if
  - end while



In [3]:

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

LEN = 40
x = np.random.randint(-5, 5, (LEN, 2))
line = np.random.randint(-5, 5, (2, 1))
#  $2y - 3x$ 
label = np.sign(np.dot(x, line))
label[label == 0] = 1
fig = plt.figure(1)
idx_0 = np.where(label[:, 0] == -1)
p0 = plt.scatter(x[idx_0, 1], x[idx_0, 0], marker = '+', color = 'c', label=' -1', s = 50)
idx_1 = np.where(label[:, 0] == 1)
p1 = plt.scatter(x[idx_1, 1], x[idx_1, 0], marker = 'x', color = 'm', label=' 1', s = 50)
plt.title(' PLA')
plt.xlabel(' x')
plt.ylabel(' y')
plt.xlim((-6, 6))
plt.ylim((-6, 6))
plt.legend()
plt.grid(True)
plt.show()
```



In [4]:

```
cnt = 0
pla = np.zeros((2, 1))
_update = 0

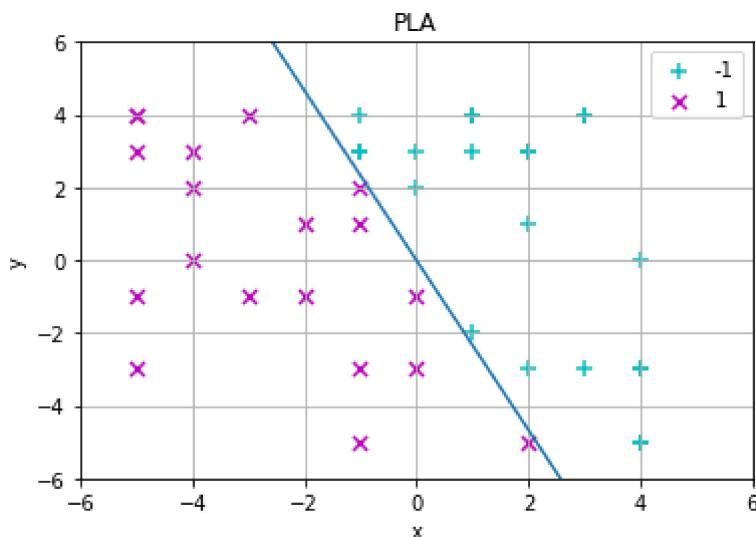
while cnt<LEN:
    _update = np.sign(np.dot(x[cnt, :], pla)) * label[cnt, 0]
    if _update <= 0:
        _x = x[cnt, :]
        _x.shape = (1, 2)
        _x = np.transpose(_x)
        pla = pla + _x*label[cnt, 0]
        cnt = 0
    else:
        cnt = cnt+1
        continue
print(pla)
```

```
[[ -3. ]
 [-7. ]]
```

In [5]:

```
fig = plt.figure(1)
idx_0 = np.where(label[:, 0] == -1)
p0 = plt.scatter(x[idx_0, 1], x[idx_0, 0], marker = '+', color = 'c', label='-1', s = 50)
idx_1 = np.where(label[:, 0] == 1)
p1 = plt.scatter(x[idx_1, 1], x[idx_1, 0], marker = 'x', color = 'm', label='1', s = 50)
plt.title('PLA')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim((-6, 6))
plt.ylim((-6, 6))
plt.legend()
plt.grid(True)

k = -pla[1, :] / pla[0, :]
line_x = np.linspace(-5, 5, 50)
line_y = k*line_x
plt.plot(line_x, line_y)
plt.show()
```



# POCKET Algorithm

---

- **Input:** traing data ( $T = \{(x_0, y_0), (x_1, y_1), \dots, (x_i, y_i)\}$ )
- **Output:**  $(w, b)$ 
  - init  $(w_{best}, b_{best}) \rightarrow random$
  - $loss_{min} \leftarrow sum(sign(w_{best}x + b_{best}) \neq y)$
  - for  $i = 0 \rightarrow 1000$  do
    - init  $(w, b) \rightarrow random$  and  $j \rightarrow 0$
    - Save error label data to  $T_{error} = \{(x_0, y_0), (x_1, y_1), \dots, (x_{err}, y_{err})\}$
    - while  $j \neq len(T_{error})$  do
      - $(\bar{w}, \bar{b}) \rightarrow (w, b) + y_j x_j$
      - $loss \leftarrow sum(sign(\bar{w}x + \bar{b}) \neq y)$
      - if  $loss < loss_{min}$  then
        - break
      - end if
      - $j = j + 1$
    - end while
    - if  $j \neq len(y_{err})$ 
      - $(w_{best}, b_{best}) \leftarrow (\bar{w}, \bar{b})$
      - $loss_{min} \leftarrow loss$
    - end if
    - $j = 0$
  - end for

In [6]:

```
def cal_loss(_pocket, _x, _label):
    _cal = np.sign(np.dot(_x, _pocket))
    _cal.shape = (40, 1)
    idx = np.where(_cal[:, 0] != _label[:, 0])
    loss = np.sum(_cal[:, 0] != _label[:, 0])
    return list(idx[0]), loss

# random init best pocket and min loss value
pocket_best = np.random.randint(-5, 5, (2, 1))
_, min_loss = cal_loss(pocket_best, x, label)

for i in range(15):
    # random pocket and cur loss for each period
    pocket = np.random.randint(-5, 5, (2, 1))
    _pocket = pocket
    idx, cur_loss = cal_loss(pocket, x, label)
    while cnt < len(idx):
        _x = x[idx[cnt], :]
        _x.shape = (2, 1)
        _pocket = pocket + _x * label[idx[cnt], 0]
        _, cur_loss = cal_loss(_pocket, x, label)
        # compare cur loss with min loss
        if cur_loss < min_loss:
            break
        else:
            cnt = cnt + 1
    if cnt != len(idx):
        pocket_best = _pocket
        _, min_loss = cal_loss(pocket_best, x, label)
    cnt = 0

pocket = pocket_best
print(pocket)
```

```
[[ -2]
 [-4]]
```

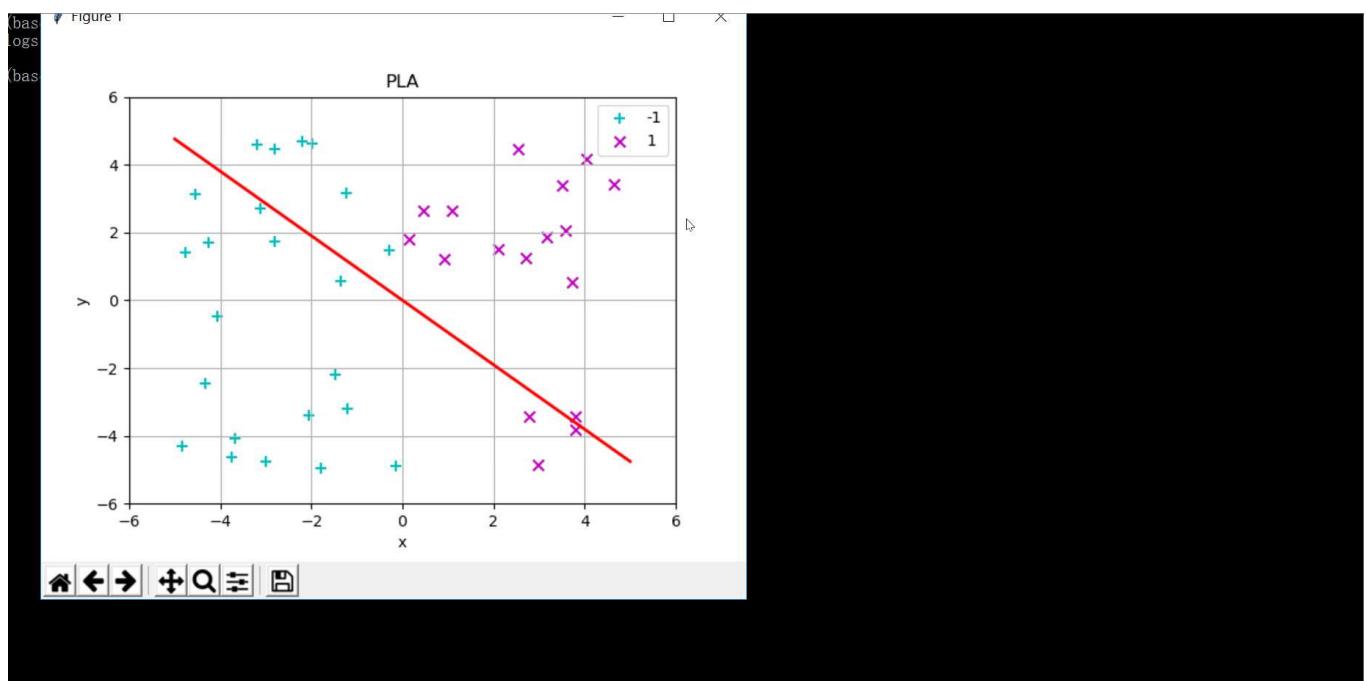
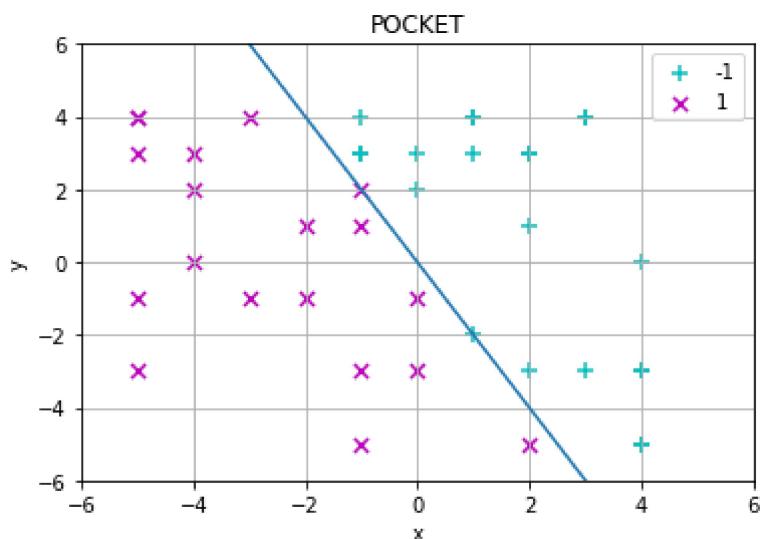
In [7]:

```

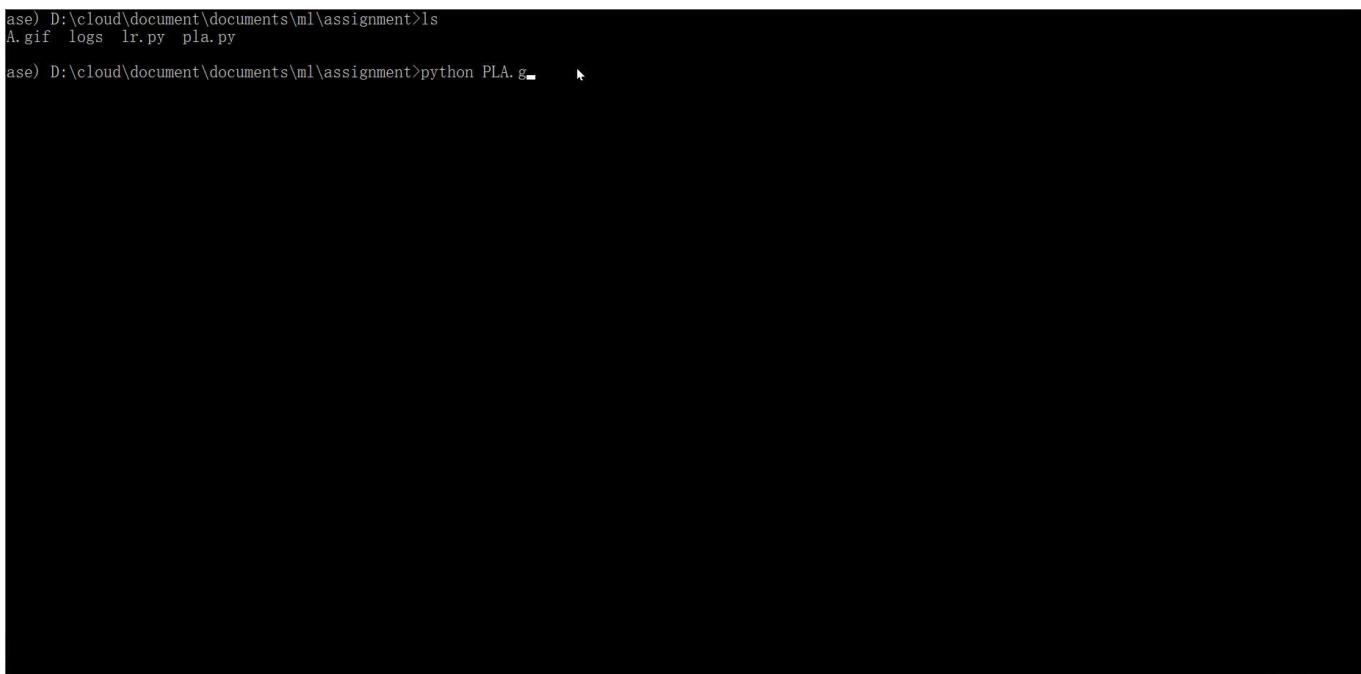
fig = plt.figure(1)
idx_0 = np.where(label[:, 0] == -1)
p0 = plt.scatter(x[idx_0, 1], x[idx_0, 0], marker = '+', color = 'c', label='-1', s = 50)
idx_1 = np.where(label[:, 0] == 1)
p1 = plt.scatter(x[idx_1, 1], x[idx_1, 0], marker = 'x', color = 'm', label='1', s = 50)
plt.title('POCKET')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim((-6, 6))
plt.ylim((-6, 6))
plt.legend()
plt.grid(True)

k = -pocket[1, :] / pocket[0, :]
line_x = np.linspace(-5, 5, 50)
line_y = k*line_x
plt.plot(line_x, line_y)
plt.show()

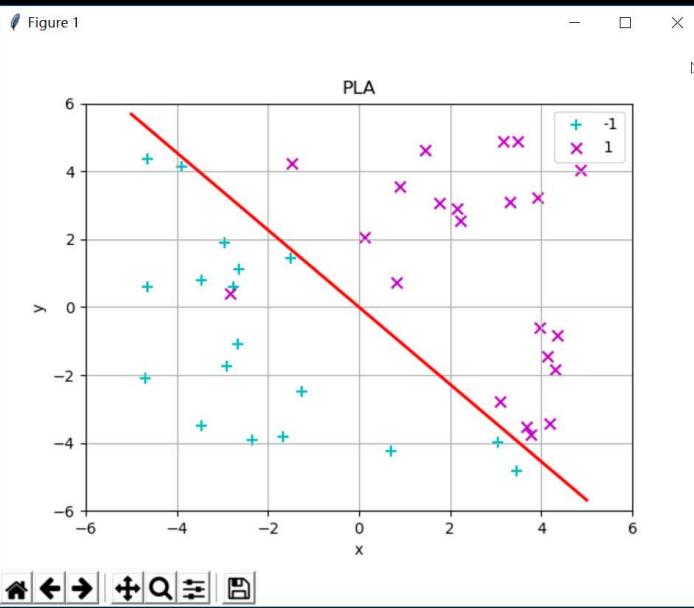
```



```
ase) D:\cloud\document\documents\ml\assignment>ls  
A.gif  logs  lr.py  pla.py  
ase) D:\cloud\document\documents\ml\assignment>python PLA.g_
```



```
ase) D:\cloud\document\documents\ml\assignment>ls  
A.gif  PLA_ERR.gif  logs  lr.py  pla.py  pocket.py  
ase) D:\cloud\document\documents\ml\assignment>
```



# SVM (Support Vector Machine)

**任务**：对线性/非线性可分不同类别的点进行二分类

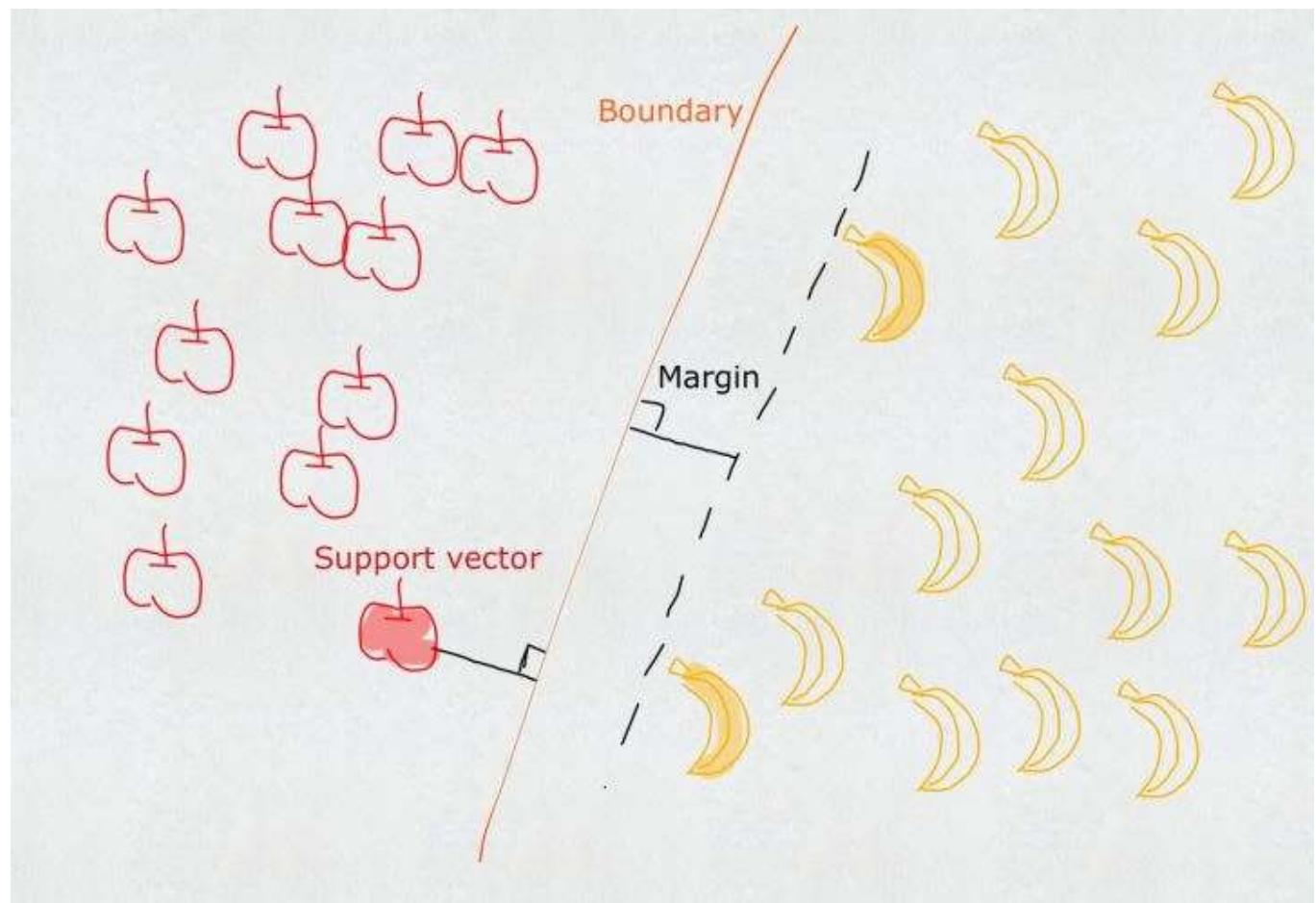
**类型**：监督学习

**模型**： $f(x) = \text{sign}(wx + b)$

**评价方法**： $\min_{w,b} \left( \frac{1}{2} \|w\|^2 \right) + C \sum_{i=1}^m \max(0, 1 - y_i(w^T x_i + b))$

$$\text{s.t. } y_i(wx_i + b) - 1 \geq 0, i = 1, 2, \dots, N$$

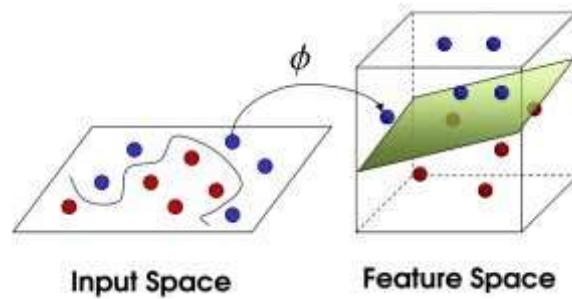
**优化方法**：核技巧/SMO



# 核技巧

核函数是二元函数，输入是变换之前的两个向量，其输出与两个向量变换之后的内积相等。  
 $k(x_1, x_2) = f(x_1^T x_2)$

- 定义直线  $y(x) = w^T x + b$
- 任意点  $x_0$  到直线的距离为  $\frac{1}{\|w\|} (w^T x_0 + b)$
- 求在误分类最少的情况下最大间距:  $\arg \max_{w,b} \left\{ \frac{1}{\|w\|} \min_n [y_i (w^T x_i + b)] \right\}$
- 最大间距对偶形式:  $\arg \max_{\alpha} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle$



- linear:  $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$ .
- polynomial:  $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + r)^d$ ,  $\gamma > 0$ .
- radial basis function (RBF):  $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$ ,  $\gamma > 0$ .
- sigmoid:  $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + r)$ .

Here,  $\gamma$ ,  $r$ , and  $d$  are kernel parameters.

In [5]:

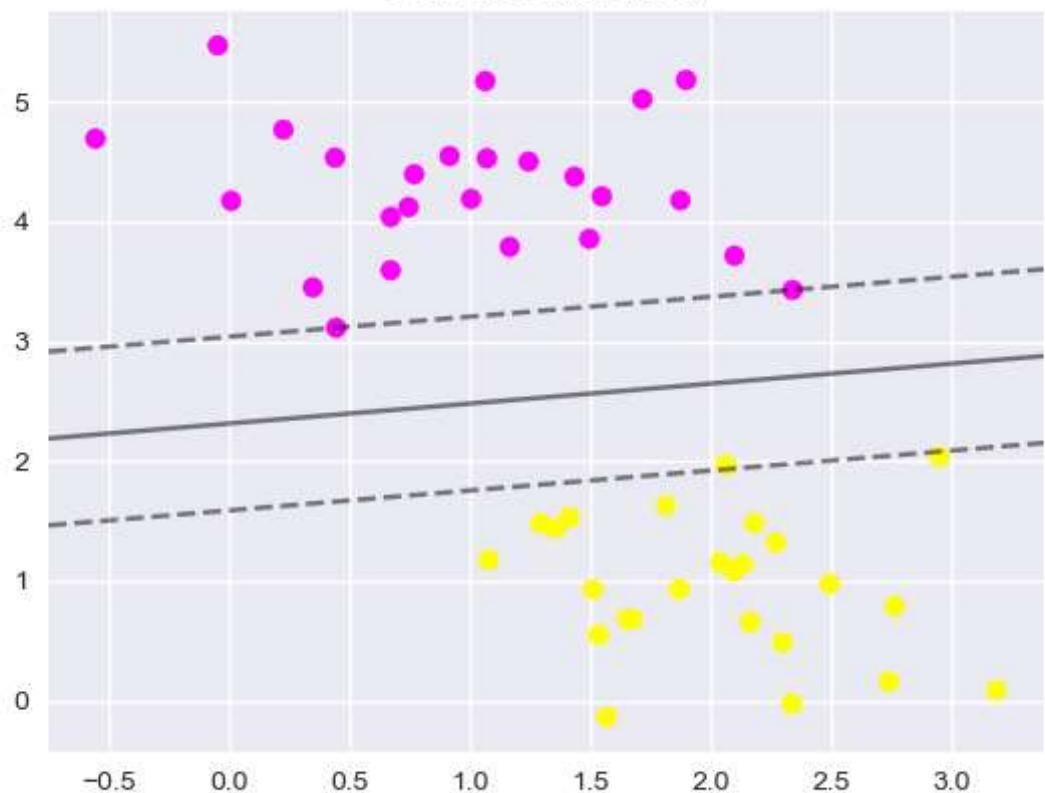
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib import style
style.use("ggplot")
from sklearn import svm
from sklearn.datasets.samples_generator import make_blobs

## 测试数据生成
_x = np.random.randint(-10, 10, (400, 2))
_= np.linalg.norm(_x, axis=1)
circle_in = _x[_<5]
circle_out = _x[_>8]
x = np.vstack((circle_in, circle_out))
y = np.zeros(x.shape[0])
y[0:circle_in.shape[0]] = 1

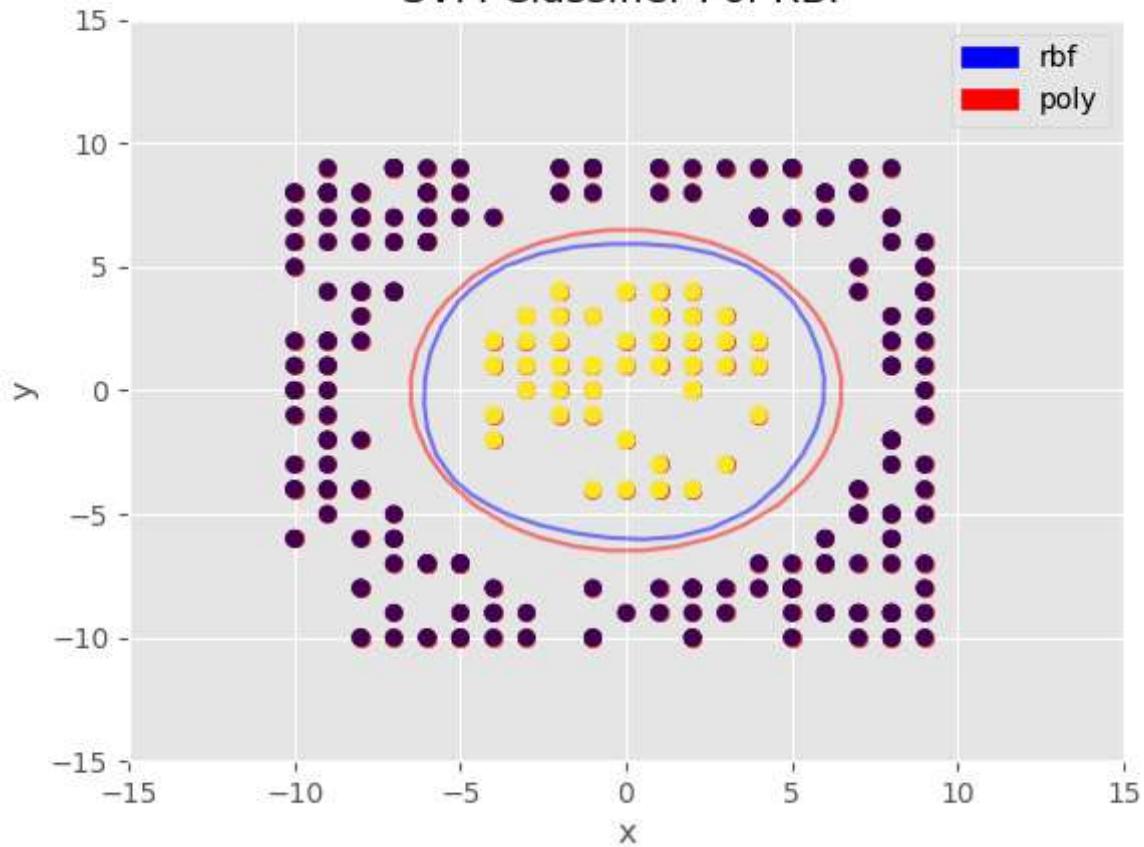
## 运行SVC核
svc_rbf = svm.SVC(kernel='rbf', C=1e3, gamma=0.1)
svc_poly = svm.SVC(kernel='poly', C=1e3, degree=2)
print(svc_rbf)
print(svc_poly)

SVC(C=1000.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
SVC(C=1000.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=2, gamma='auto', kernel='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

SVM Classifier Linear



SVM Classifier Pol-RBF



# LR (Logistic Regression)

**任务**：对线性/非线性可分不同类别的点进行多分类

**类型**：监督学习

**模型**：3层fc nn, output layer(softmax) :  $y_i = \frac{e^{wx_i+b}}{\sum_{j=1}^C e^{wx_j+b}}$

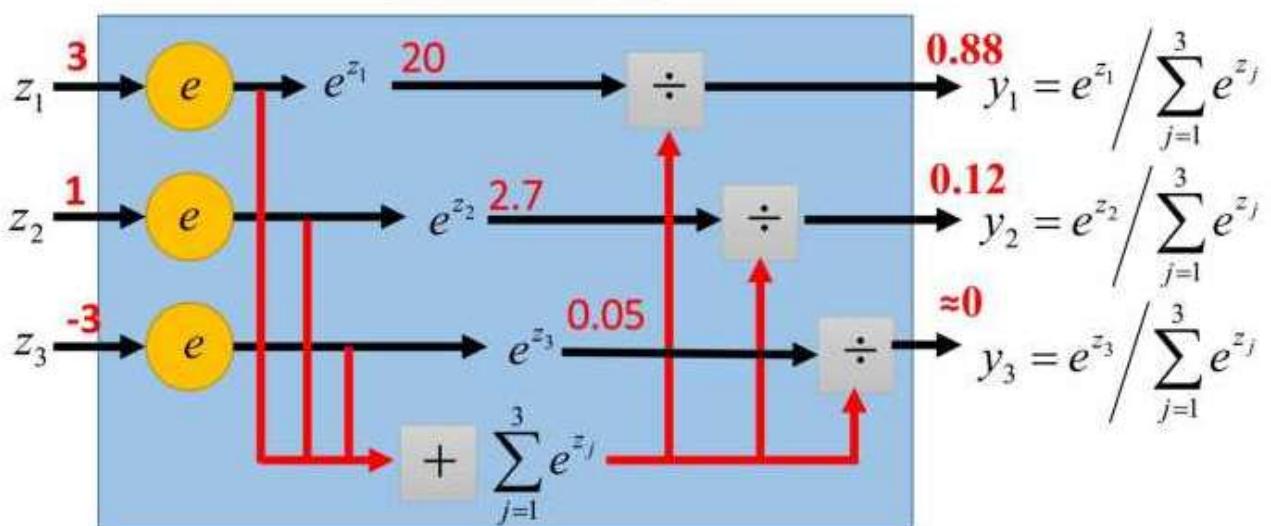
**评价方法**： $-\sum_{i=1}^N y_i \log \frac{e^{wx_i+b}}{\sum_{j=1}^C e^{wx_j+b}}$

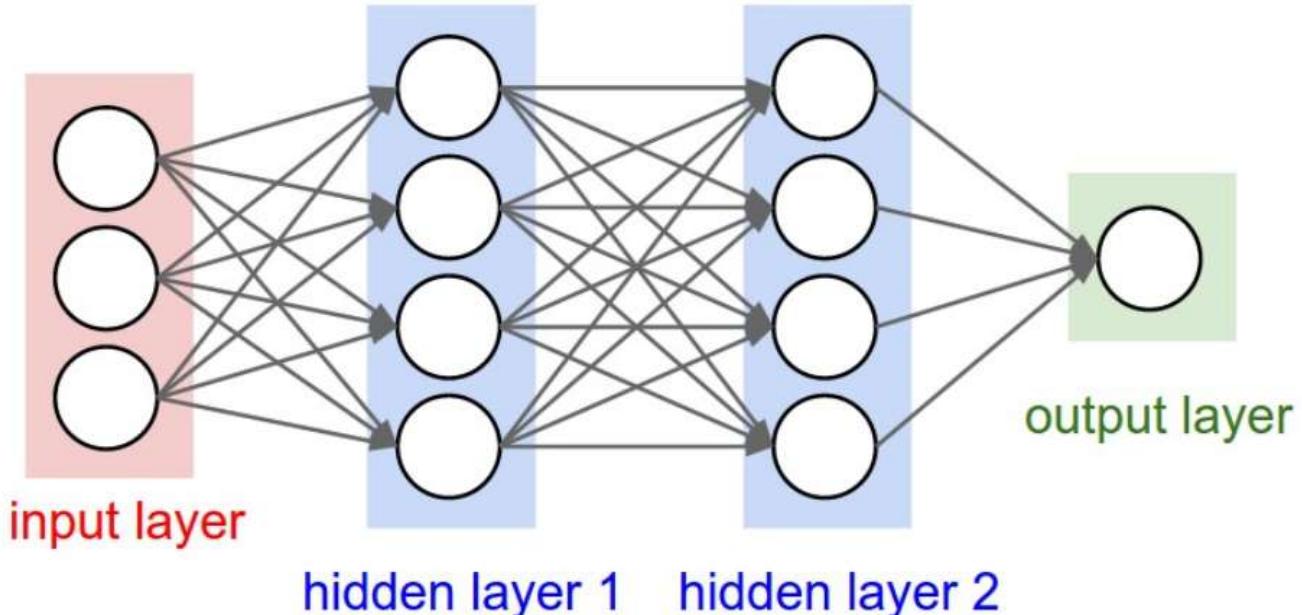
**优化方法**：Gradient Descent/SGD/ADAM/Adagrad/RMSprop

**Probability:**

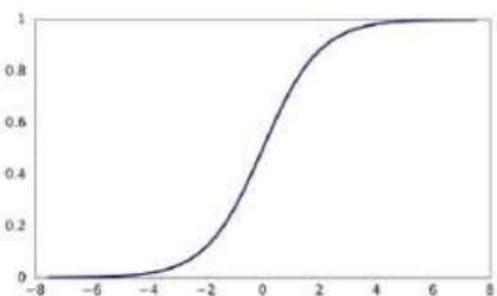
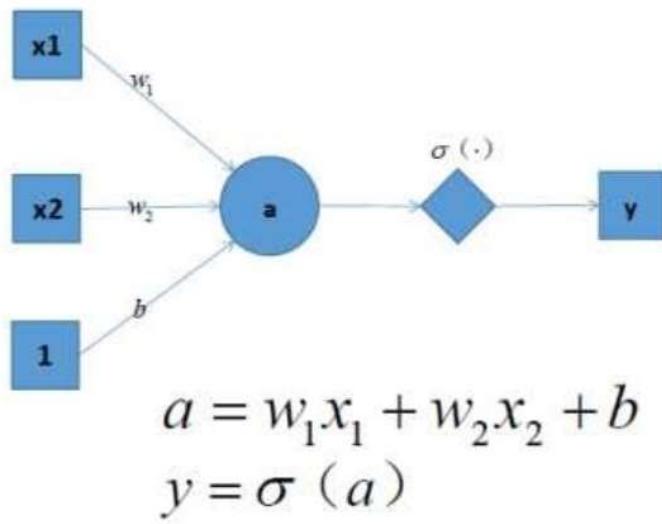
- $1 > y_i > 0$
- $\sum_i y_i = 1$

## Softmax Layer





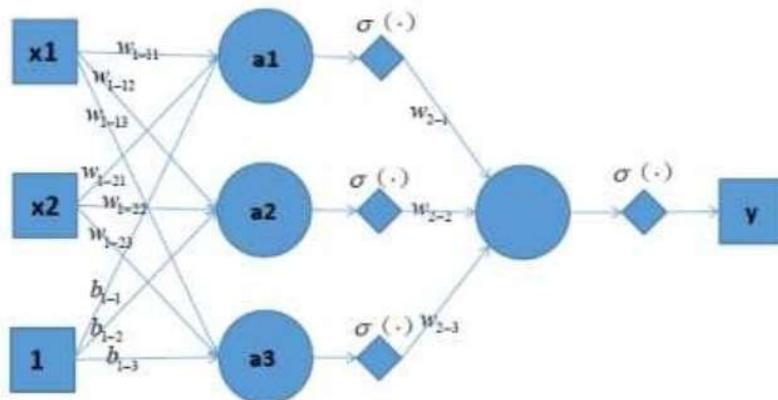
## Perceptron with non-linear activation function



$\sigma(\cdot)$  is a non-linear activation function, sigmoid was the most popular one,

$$\sigma(y) = \frac{1}{1+e^{-y}}$$

# Perceptron with non-linear activation function



$$a1 = w_{1-11}x_1 + w_{1-21}x_2 + b_{1-1}$$

$$a2 = w_{1-12}x_1 + w_{1-22}x_2 + b_{1-2}$$

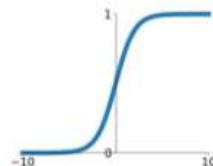
$$a3 = w_{1-13}x_1 + w_{1-23}x_2 + b_{1-3}$$

$$y = \sigma(w_{2-1}\sigma(a1) + w_{2-2}\sigma(a2) + w_{2-3}\sigma(a3))$$

## Activation functions

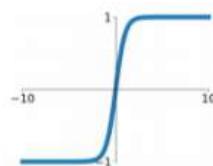
### Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



### tanh

$$\tanh(x)$$



### ReLU

$$\max(0, x)$$



### Leaky ReLU

$$\max(0.1x, x)$$



### Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

### ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# 请问人工神经网络中的activation function的作用具体是什么？为...

第一个问题：为什么引入非线性激励函数？

如果不使用激励函数（其实相当于激励函数是 $f(x) = x$ ），在这种情况下你每一层输出都是上层输入的线性函数，很容易验证，无论你神经网络有多少层，输出都是输入的线性组合，与没有隐藏层效果相当，这种情况就是最原始的感知机（Perceptron）了。

正因为上面的原因，我们决定引入非线性函数作为激励函数，这样深层神经网络就有意义了（不再是输入的线性组合，可以逼近任意函数）。最早的想法是sigmoid函数或者tanh函数，输出有界，很容易充当下一层输入（以及一些人的生物解释balabala）。

第二个问题：为什么引入Relu呢？

第一，采用sigmoid等函数，算激活函数时（指数运算），计算量大，反向传播求误差梯度时，求导涉及除法，计算量相对大，而采用Relu激活函数，整个过程的计算量节省很多。

第二，对于深层网络，sigmoid函数反向传播时，很容易就会出现梯度消失的情况（在sigmoid接近饱和区时，变换太缓慢，导数趋于0，这种情况会造成信息丢失，参见 @Haofeng Li 答案的第三点），从而无法完成深层网络的训练。

第三，Relu会使一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生（以及一些人的生物解释balabala）。

## Softmax Classifier (Multinomial Logistic Regression)



$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

cat  
car  
frog

3.2
5.1
-1.7

exp

24.5
164.0
0.18

normalize

0.13
0.87
0.00

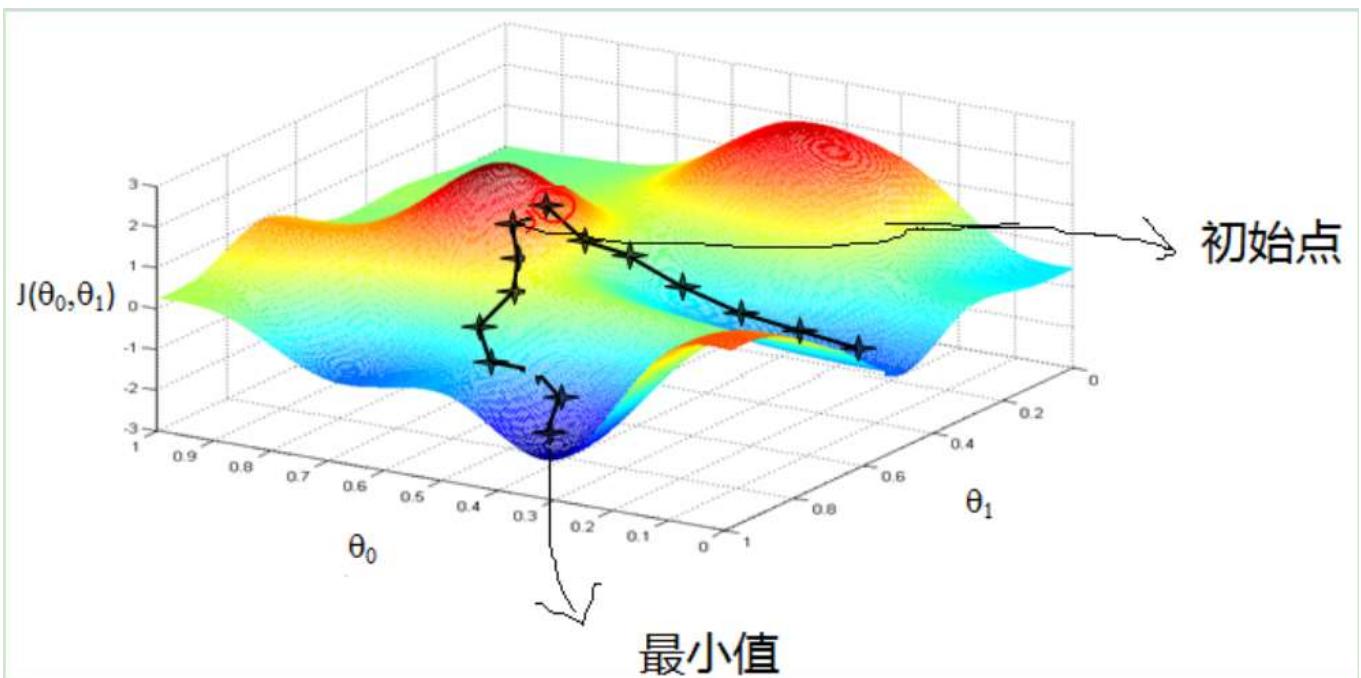
$$\rightarrow L_i = -\log(0.13) = 0.89$$

unnormalized log probabilities

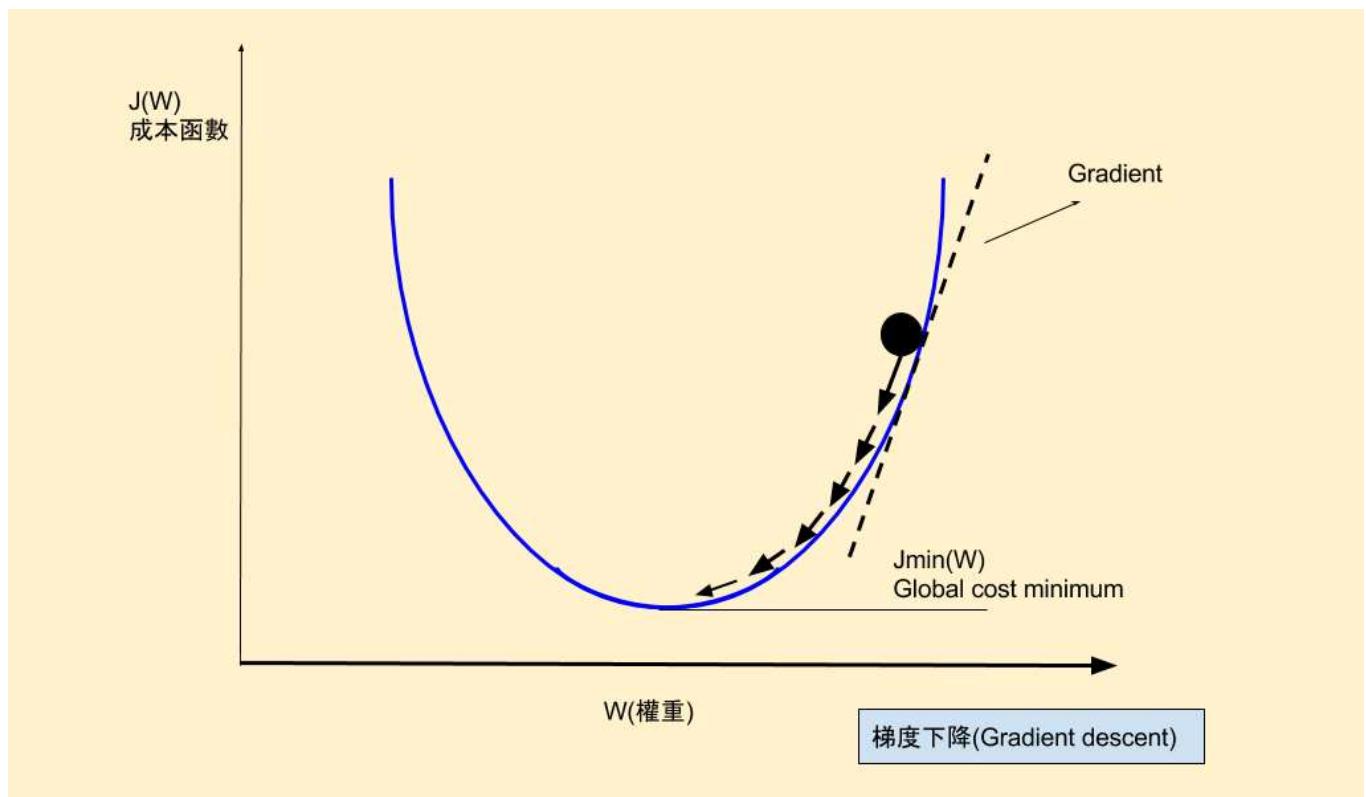
probabilities

Q: What is the min/max possible loss  $L_i$ ?

# Gradient Descent



$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$



# Backpropagation

Backpropagation: a simple example

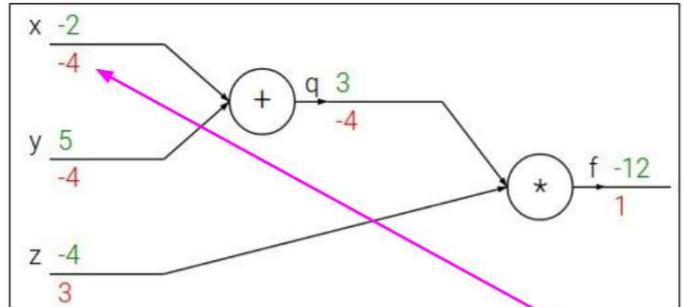
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2$ ,  $y = 5$ ,  $z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want:  $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

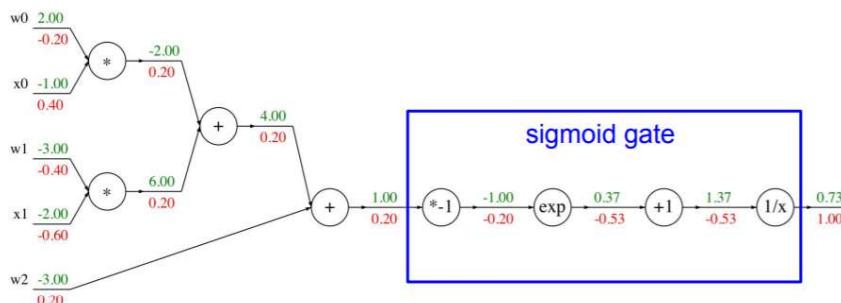
$$\frac{\partial f}{\partial x}$$

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

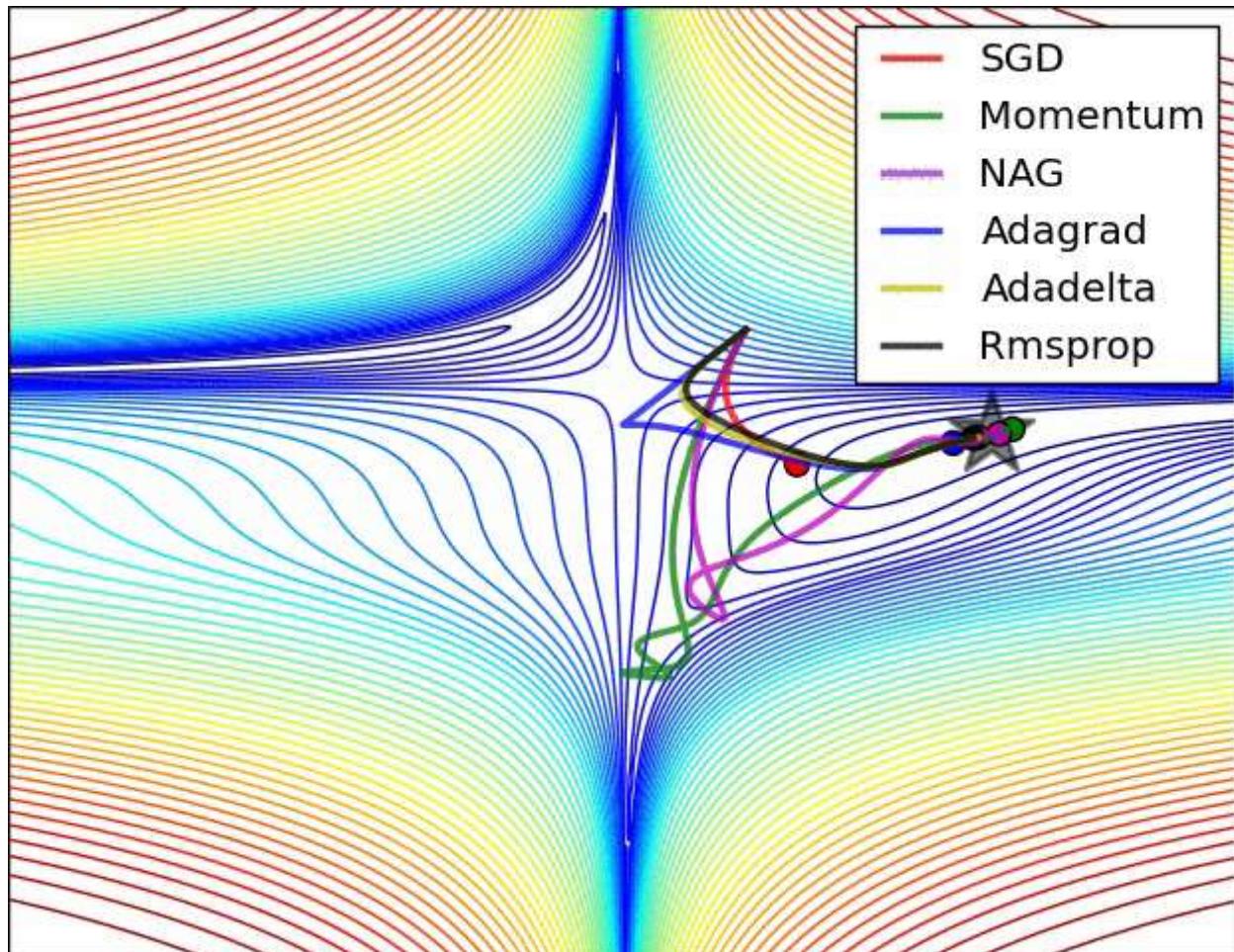
sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x)$$



# Gradient Descent各算法对比

<https://zhuanlan.zhihu.com/p/32230623> (<https://zhuanlan.zhihu.com/p/32230623>)



In [1]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs
import time
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

DIMENSION = 2
KNUM = 4
SAMPLE_CNT = 240

def init_variable(name, shape):
    return(tf.get_variable(name, shape=shape, initializer=tf.contrib.layers.xavier_initializer()))

# 测试数据生成
x_data, y_data = make_blobs(n_samples=SAMPLE_CNT, n_features=DIMENSION, centers=KNUM, center_box=(-20, 20), cluster_std=1.5)
x = x_data.tolist()
label = np.zeros((SAMPLE_CNT, KNUM))

label[y_data==0] = [0, 0, 0, 1]
label[y_data==1] = [0, 0, 1, 0]
label[y_data==2] = [0, 1, 0, 0]
label[y_data==3] = [1, 0, 0, 0]

y = label.tolist()
keep_prob = tf.placeholder(tf.float32)
prediction_value = y.copy()

# 生成学习网络
# layer0: input
xs = tf.placeholder("float", [None, 2])
ys = tf.placeholder("float", [None, 4])
keep_prob = tf.placeholder(tf.float32)

# layer1:
W1 = init_variable('W1', [2, 20])
bias1 = init_variable('b1', [1, 20])
o1 = tf.nn.relu(tf.matmul(xs, W1) + bias1)
o1_drop = tf.nn.dropout(o1, keep_prob)

# layer2:
W2 = init_variable('W2', [20, 20])
bias2 = init_variable('b2', [1, 20])
o2 = tf.nn.relu(tf.matmul(o1_drop, W2) + bias2)
o2_drop = tf.nn.dropout(o2, keep_prob)

# layer3: output
W3 = init_variable('W3', [20, 4])
bias3 = init_variable('b3', [1, 4])
#predict = tf.nn.softmax(tf.matmul(o2_drop, W3) + bias3)
o3 = tf.matmul(o2_drop, W3) + bias3
predict = tf.nn.softmax(o3)
```

```
D:\cloud\anaconda\lib\site-packages\h5py\__init__.py:34: FutureWarning: Conversion
of the second argument of issubdtype from `float` to `np.floating` is deprecated.
In future, it will be treated as `np.float64 == np.dtype(float).type`.
from ._conv import register_converters as _register_converters
```

In [2]:

```
# loss sum( y*log(predict) )
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=o3, labels=ys))
#loss = tf.reduce_mean(-tf.reduce_sum(ys*tf.log(predict), reduction_indices=[1]))

# train method
#train = tf.train.GradientDescentOptimizer(0.1).minimize(loss)
train = tf.train.AdamOptimizer(1e-3).minimize(loss)

#init all variables, it will be activated by use sess.run(init)
init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    for i in range(200):
        sess.run(train, feed_dict={xs:x, ys:y, keep_prob: 0.5})
        if i % 10 == 0:
            prediction_value = sess.run(predict, feed_dict={xs:x, keep_prob: 1})
            print(sess.run(loss, feed_dict={xs:x, ys:y, keep_prob: 1}))
```

WARNING:tensorflow:From <ipython-input-2-cd5c0946c093>:2: softmax\_cross\_entropy\_with\_logits (from tensorflow.python.ops.nn\_ops) is deprecated and will be removed in a future version.

Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow into the labels input on backprop by default.

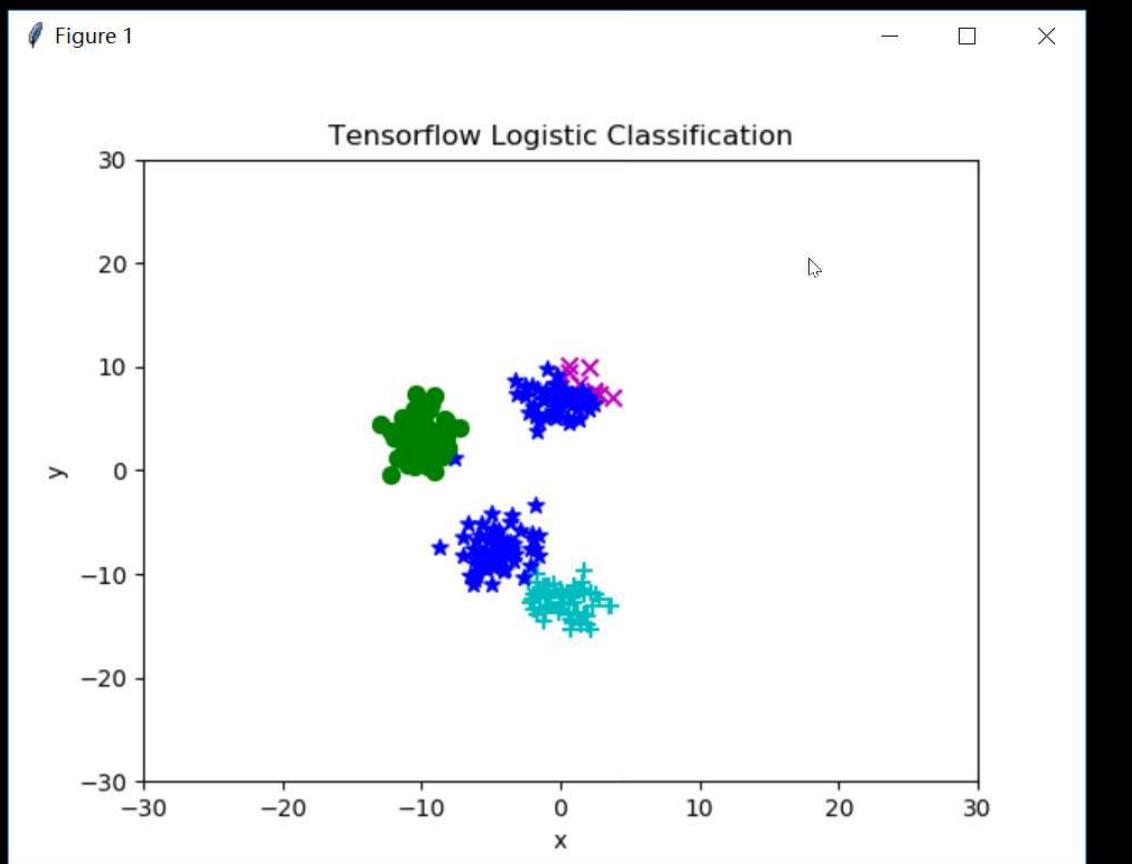
See `tf.nn.softmax_cross_entropy_with_logits_v2`.

```
2.0771701
1.4298545
1.2080476
1.062615
0.9569904
0.8710387
0.8117963
0.7671517
0.7361572
0.71022105
0.6842088
0.6632282
0.6432349
0.63005894
0.6273728
0.6227281
0.6168578
0.61016667
0.60824126
0.60349876
```

```

l. 4142144
l. 3800199
l. 3467587
l. 3144107
l. 2833151
l. 2533123
l. 2239541
l. 1952354
l. 1673808
l. 1404877
l. 1145782
l. 0892626
l. 0648935
l. 0410236
l. 0177578
l. 9955776
l. 9741459
l. 9535567
l. 933569
l. 9142443
l. 8958012
l. 8782816
l. 8614809
l. 8453185
l. 82987195
l. 815012
l. 8008142
l. 78693503
l. 77341944
l. 7603458
l. 7478824
l. 735613
l. 72364223
l. 7122387
l. 7010354
l. 6900834
l. 67940044
l. 6691880

```



## LR (Linear Regression)

**任务**：对一组数据进行曲线拟合

**类型**：监督学习

**模型**：2层fc nn, output layer :  $y_i = wx_i + b$

**评价方法**： $\sum_{i=1}^N (y_i - (wx_i + b))^2$

**优化方法**：Gradient Descent/SGD/ADAM/Adagrad/RMSprop

In [6]:

```

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import time
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# 测试数据生成
x_data = np.linspace(-1, 1, 300)[:, np.newaxis]
noise = np.random.normal(0, 0.05, x_data.shape)
y_data = np.square(x_data) - 0.5 + noise

# 生成学习网络
# layer0: input
xs = tf.placeholder(tf.float32, [None, 1])
ys = tf.placeholder(tf.float32, [None, 1])

# layer1:
W1 = tf.Variable(tf.random_normal([1, 10]))
bias1 = tf.Variable(tf.zeros([1, 10]) + 0.1)
o1 = tf.nn.relu(tf.matmul(xs, W1) + bias1)

# layer2: output
W2 = tf.Variable(tf.random_normal([10, 1]))
bias2 = tf.Variable(0.1)
predict = tf.matmul(o1, W2) + bias2

# loss
loss = tf.reduce_mean(tf.reduce_sum(tf.square(ys - predict),
                                    reduction_indices=[1]))

# train method
train = tf.train.GradientDescentOptimizer(0.1).minimize(loss)

#init all variables, it will be activated by use sess.run(init)
init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    for i in range(1000):
        sess.run(train, feed_dict={xs:x_data, ys:y_data})
        if i % 100 == 0:
            prediction_value = sess.run(predict, feed_dict={xs:x_data})
            print(sess.run(loss, feed_dict={xs:x_data, ys:y_data}))

```

```

0.21007633
0.009326225
0.0065052332
0.0041997177
0.0037967209
0.003648324
0.0035490533
0.0034687535
0.003410836
0.0033446297

```

```
ase) D:\cloud\document\documents\ml\assignment>ls  
A.gif PLA.ERR.gif POCKET.gif logs lr.py pla.py pocket.py  
ase) D:\cloud\document\documents\ml\assignment>p_
```

## KMEANS

---

**任务** : 对未标记的点进行多分类

---

**类型** : 非监督学习

**模型** :  $y_j - > \min \sum_{i=1}^K (x_j - k_i)^2$

**评价方法** :  $\sum_{i=1}^K \sum_{j=1}^N (x_j - k_i)^2$

**优化方法** :  $k_j - > \frac{1}{N} \sum_{i=1}^N x_i \bullet (y_i == j)$

# KMEANS Algorithm

---

- **Input:** data ( $T = \{x_0, x_1, \dots, x_i\}$ )
- **Output:**  $\{y_0, y_1, \dots, y_i\}$ 
  - init  $\{k_0, k_1, \dots, k_K\} \rightarrow x_{random}$  (s.t. kmeans cluster cnt: K)
  - update  $y_i \rightarrow minarg\{(x_i - k)^2\}$
  - $loss_{min}, loss_{cur}, loss_{pre} \leftarrow \sum_i^K \sum_j^{y==i} (x_j - k_i)^2$
  - for  $i = 0 \rightarrow 1000$  do
    - init  $(w, b) \rightarrow random$  and  $j \rightarrow 0$
    - Save error label data to  $T_{error} = \{(x_0, y_0), (x_1, y_1), \dots, (x_{err}, y_{err})\}$
    - while  $j! = len(T_{error})$  do
      - $(\bar{w}, \bar{b}) \rightarrow (w, b) + y_j x_j$
      - $loss \leftarrow sum(sign(\bar{w}x + \bar{b})! = y)$
      - if  $loss < loss_{min}$  then
        - break
      - end if
      - $j = j + 1$
    - end while
    - if  $j! = len(y_{err})$ 
      - $(w_{best}, b_{best}) \leftarrow (\bar{w}, \bar{b})$
      - $loss_{min} \leftarrow loss$
    - end if
    - $j = 0$
  - end for

In [12]:

```

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_blobs

DIMENSION = 2
KNUM = 4
SAMPLE_CNT = 240
train_step = 100

def label_reset(x, kpoint):
    _label = np.zeros((x.shape[0], 1))
    for i in range(0, x.shape[0]):
        _ = np.linalg.norm(x[i, :] - kpoint, axis=1)
        _label[i] = np.argmin(_, 0)

    return _label

def kpoint_reset(x, label):
    _kpoint = np.zeros((KNUM, DIMENSION))
    idx = 0
    for i in range(0, KNUM):
        idx = np.where(label[:, 0] == i)
        _kpoint[i] = np.mean(x[idx], axis=0)
    return _kpoint

def cal_total(x, kpoint, label):
    _sum = 0
    for i in range(0, KNUM):
        idx = np.where(label[:, 0] == i)
        _linalg = np.linalg.norm(x[idx] - kpoint[i], axis=1)
        _sum += np.sum(_linalg)
    _sum /= SAMPLE_CNT
    return _sum

"""

# x, y, label
x = np.zeros((SAMPLE_CNT, DIMENSION))
x = np.random.randint(-20, 20, (SAMPLE_CNT, DIMENSION))
"""

x, _ = make_blobs(n_samples=SAMPLE_CNT, n_features=DIMENSION, centers=KNUM, center_box=(-20, 20), cluster_std=1.5)

label = np.zeros((SAMPLE_CNT, 1))

#kmeans init
kpoint = np.random.randint(-20, 20, (KNUM, DIMENSION))
kpoint_min = kpoint.copy()

label = label_reset(x, kpoint)
label_min = label.copy()

sum = cal_total(x, kpoint, label)
sum_min = sum
delta = sum

```

In [13]:

```
for i in range(train_step):
    kpoint = np.random.randint(-20, 20, (KNUM, DIMENSION))
    label = label_reset(x, kpoint)
    sum = cal_total(x, kpoint, label)
    delta = sum
    while delta > 0.001:
        kpoint = kpoint_reset(x, label)
        label = label_reset(x, kpoint)
        _sum = cal_total(x, kpoint, label)
        delta = sum - _sum
        sum = _sum
        #print("sum: %f" %sum)
    if sum < sum_min:
        print("Update sum_min: %f" %sum)
        sum_min = sum
        kpoint_min = kpoint.copy()
        label_min = label.copy()
print(sum)
```

Update sum\_min: 1.833920

Update sum\_min: 1.833920

D:\cloud\anaconda\lib\site-packages\numpy\core\fromnumeric.py:2957: RuntimeWarning: Mean of empty slice.

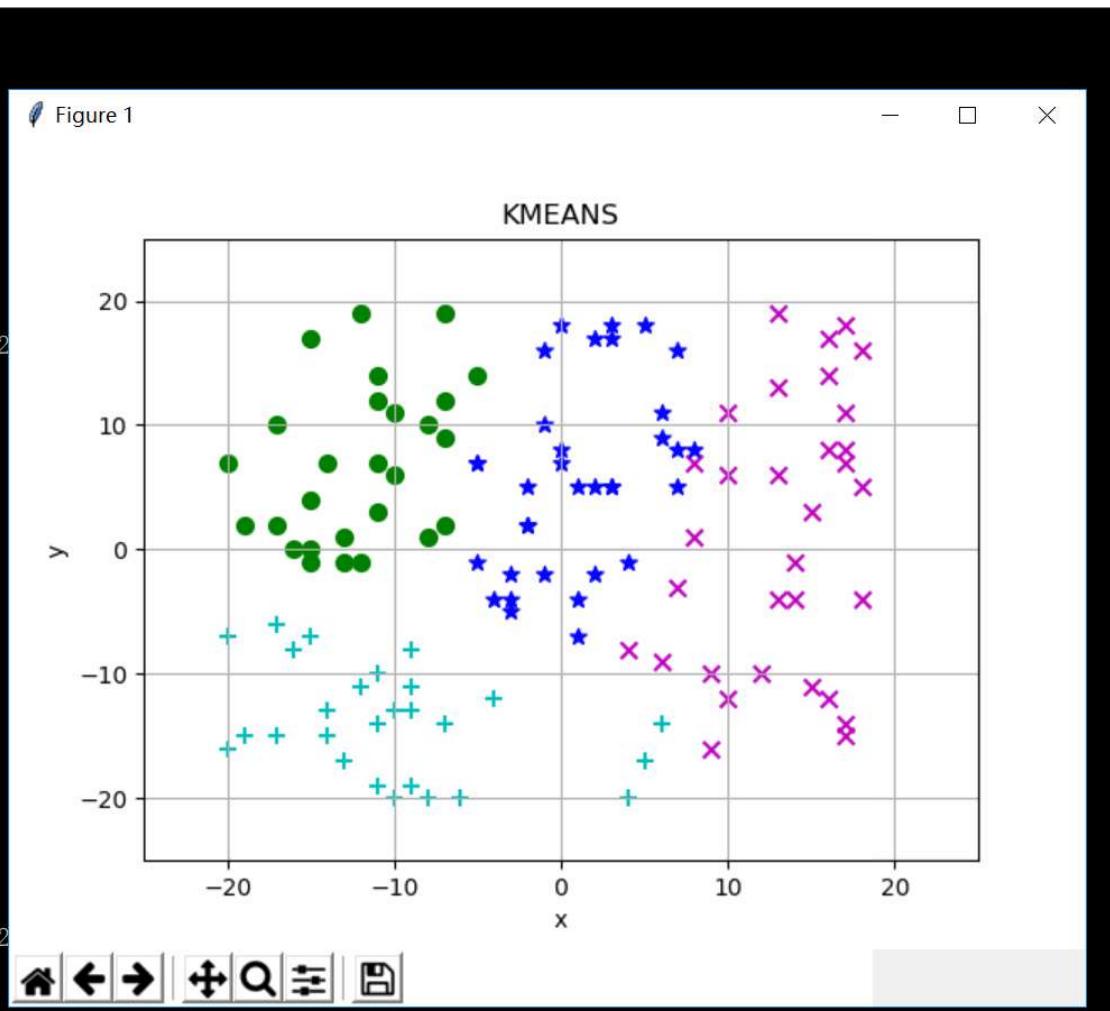
out=out, \*\*kwargs)

D:\cloud\anaconda\lib\site-packages\numpy\core\\_methods.py:73: RuntimeWarning: invalid value encountered in true\_divide

ret, rcount, out=ret, casting='unsafe', subok=False)

1.8339197144463897

```
sum: 7. 818776
sum: 7. 626627
sum: 7. 498779
sum: 7. 459135
sum: 7. 418426
sum: 7. 378499
sum: 7. 351362
sum: 7. 337422
sum: 7. 317943
sum: 7. 229703
sum: 7. 207922
sum: 7. 207518
Update sum_min: 7. 2
sum: 7. 840566
sum: 7. 625659
sum: 7. 505159
sum: 7. 465197
sum: 7. 419460
sum: 7. 385262
sum: 7. 351362
sum: 7. 337422
sum: 7. 317943
sum: 7. 229703
sum: 7. 207922
sum: 7. 207518
sum: 8. 211126
sum: 7. 903738
sum: 7. 589811
sum: 7. 369796
sum: 7. 343427
sum: 7. 314871
sum: 7. 243016
sum: 7. 207922
sum: 7. 207518
Update sum_min: 7. 2
sum: 8. 235258
sum: 8. 071350
sum: 7. 078012
```



# CNN

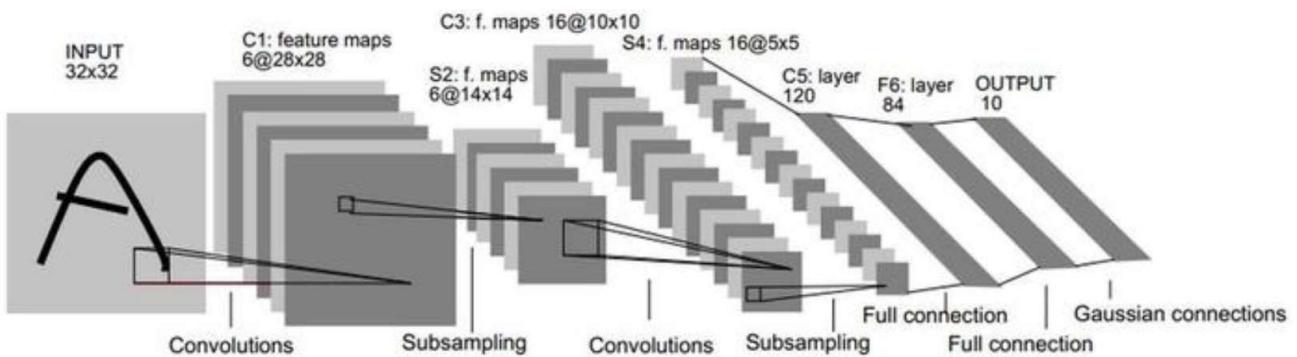
**任务** : OCR end-to-end 识别

**类型** : 监督学习

**模型** : 3层cnn + 3层relu + 2层fc, output layer(softmax) :  $y_i = \frac{e^{wx_i+b}}{\sum_{i=1}^C e^{wx_i+b}}$

**评价方法** :  $-\sum_{i=1}^N y_i \log \frac{e^{wx_i+b}}{\sum_{j=1}^C e^{wx_j+b}}$

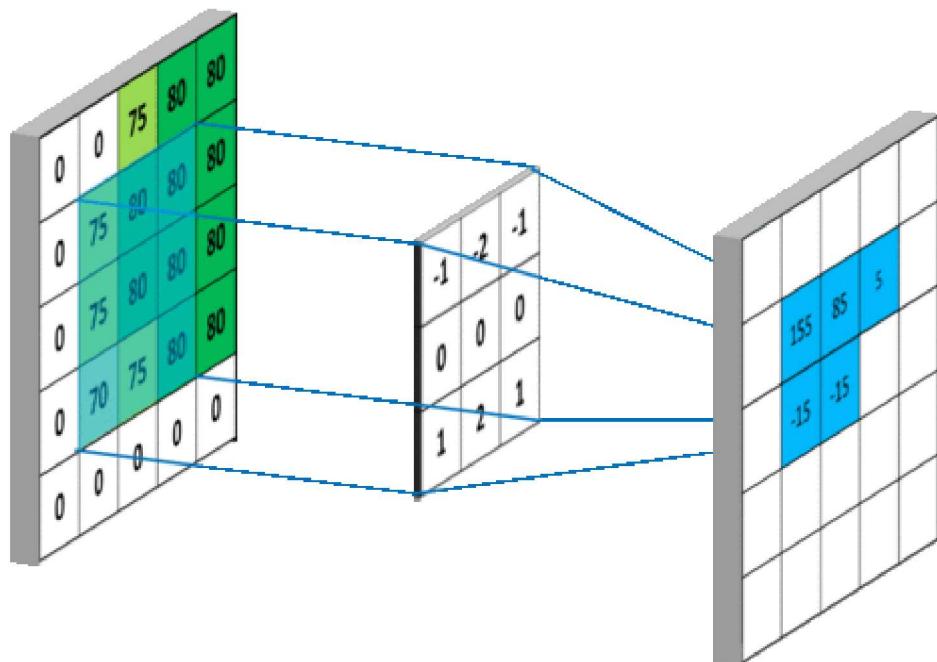
**优化方法** : Gradient Descent/SGD/ADAM/Adagrad/RMSprop



# Convolution

翻转->移动->乘积->求和

$$f[x, y] * g[x, y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \bullet g[x - n_1, y - n_2]$$



In [22]:

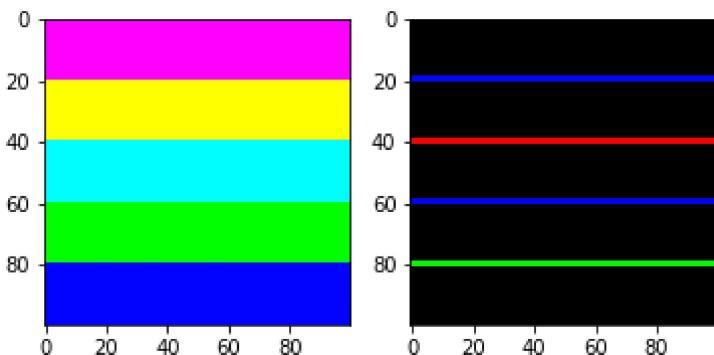
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import sys
%matplotlib inline

img = np.ones((100, 100, 3))
img[0:20, :, 1] = 255
img[20:40, :, 2] = 255
img[40:100, :, 0] = 255
img[60:80, :, 2] = 255
img[80:100, :, 1] = 255

prewitt = np.array([[-1, -1, -1],
                   [0, 0, 0],
                   [1, 1, 1]])
res = cv2.filter2D(img, -1, prewitt)
regular = np.uint8(res)
print(img[19:22, 19:22, 1])
print(res[20, 20, 1])

plt.figure()
plt.subplot(1, 2, 1)
plt.imshow(img)
plt.subplot(1, 2, 2)
plt.imshow(regular)
plt.show()
```

```
[[255. 255. 255.]
 [ 1.   1.   1.]
 [ 1.   1.   1.]]
-762.0
```



# LeNet、AlexNet、GoogLeNet、VGG、ResNet

模型名	AlexNet	VGG	GoogLeNet	ResNet
初入江湖	2012	2014	2014	2015
层数	8	19	22	152
Top-5错误	16.4%	7.3%	6.7%	3.57%
Data Augmentation	+	+	+	+
Inception(NIN)	-	-	+	-
卷积层数	5	16	21	151
卷积核大小	11,5,3	3	7,1,3,5	7,1,3,5
全连接层数	3	3	1	1
全连接层大小	4096,4096,1000	4096,4096,1000	1000	1000
Dropout	+	+	+	+
Local Response Normalization	+	-	+	-
Batch Normalization	-	-	-	+

## Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

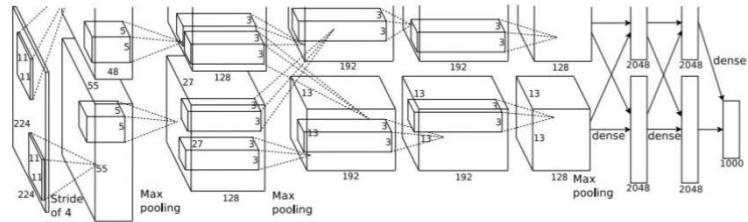
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

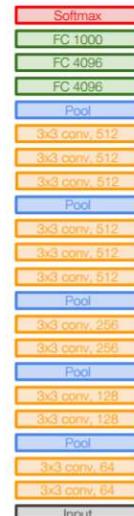


### Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% > 15.4%

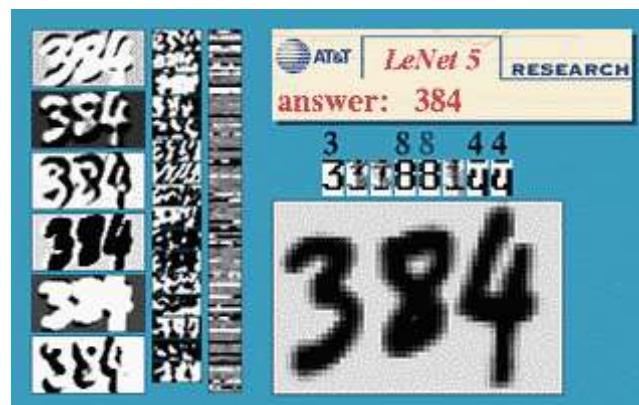
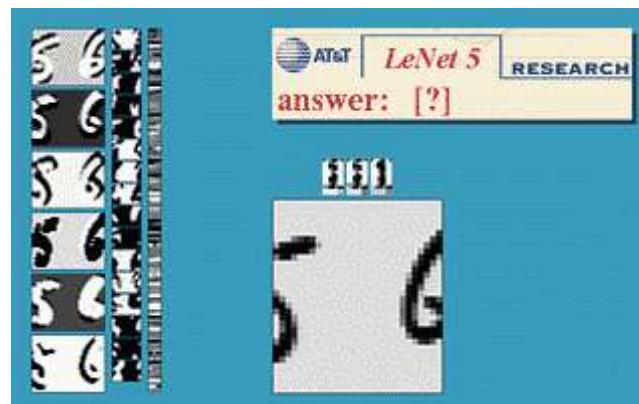
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

INPUT: [224x224x3] memory:  $224 \times 224 \times 3 = 150K$  params: 0 (not counting biases)  
 CONV3-64: [224x224x64] memory:  $224 \times 224 \times 64 = 3.2M$  params:  $(3 \times 3 \times 3) \times 64 = 1,728$   
 CONV3-64: [224x224x64] memory:  $224 \times 224 \times 64 = 3.2M$  params:  $(3 \times 3 \times 64) \times 64 = 36,864$   
 POOL2: [112x112x64] memory:  $112 \times 112 \times 64 = 800K$  params: 0  
 CONV3-128: [112x112x128] memory:  $112 \times 112 \times 128 = 1.6M$  params:  $(3 \times 3 \times 64) \times 128 = 73,728$   
 CONV3-128: [112x112x128] memory:  $112 \times 112 \times 128 = 1.6M$  params:  $(3 \times 3 \times 128) \times 128 = 147,456$   
 POOL2: [56x56x128] memory:  $56 \times 56 \times 128 = 400K$  params: 0  
 CONV3-256: [56x56x256] memory:  $56 \times 56 \times 256 = 800K$  params:  $(3 \times 3 \times 128) \times 256 = 294,912$   
 CONV3-256: [56x56x256] memory:  $56 \times 56 \times 256 = 800K$  params:  $(3 \times 3 \times 256) \times 256 = 589,824$   
 CONV3-256: [56x56x256] memory:  $56 \times 56 \times 256 = 800K$  params:  $(3 \times 3 \times 256) \times 256 = 589,824$   
 POOL2: [28x28x256] memory:  $28 \times 28 \times 256 = 200K$  params: 0  
 CONV3-512: [28x28x512] memory:  $28 \times 28 \times 512 = 400K$  params:  $(3 \times 3 \times 256) \times 512 = 1,179,648$   
 CONV3-512: [28x28x512] memory:  $28 \times 28 \times 512 = 400K$  params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$   
 CONV3-512: [28x28x512] memory:  $28 \times 28 \times 512 = 400K$  params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$   
 POOL2: [14x14x512] memory:  $14 \times 14 \times 512 = 100K$  params: 0  
 CONV3-512: [14x14x512] memory:  $14 \times 14 \times 512 = 100K$  params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$   
 CONV3-512: [14x14x512] memory:  $14 \times 14 \times 512 = 100K$  params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$   
 CONV3-512: [14x14x512] memory:  $14 \times 14 \times 512 = 100K$  params:  $(3 \times 3 \times 512) \times 512 = 2,359,296$   
 POOL2: [7x7x512] memory:  $7 \times 7 \times 512 = 25K$  params: 0  
 FC: [1x1x4096] memory:  $4096$  params:  $7 \times 7 \times 512 \times 4096 = 102,760,448$   
 FC: [1x1x4096] memory:  $4096$  params:  $4096 \times 4096 = 16,777,216$   
 FC: [1x1x1000] memory:  $1000$  params:  $4096 \times 1000 = 4,096,000$



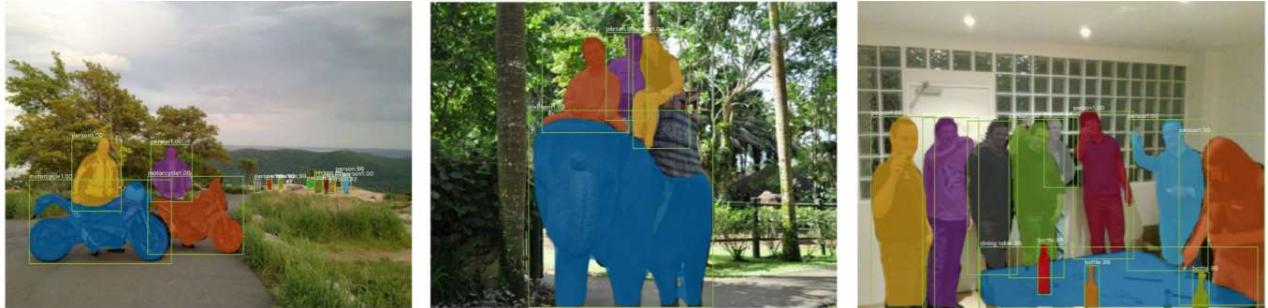
VGG16

TOTAL memory:  $24M * 4$  bytes  $\approx 96MB$  / image (only forward!  $\sim 2$  for bwd)  
 TOTAL params: 138M parameters



## Mask R-CNN Segment

Mask R-CNN: Very Good Results!



He et al, "Mask R-CNN", arXiv 2017  
 Figures copyright Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick, 2017.  
 Reproduced with permission.

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 11 - 91 May 10, 2017

## Mask R-CNN Post

Mask R-CNN  
 Also does pose

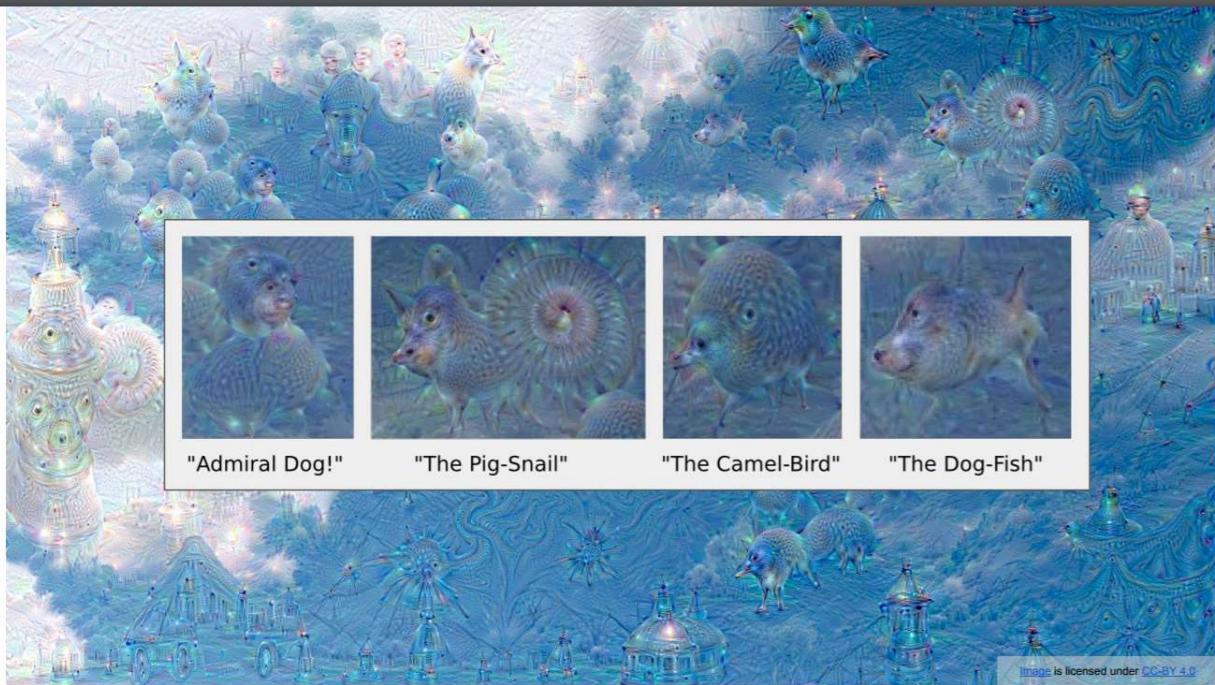


He et al, "Mask R-CNN", arXiv 2017  
 Figures copyright Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick, 2017.  
 Reproduced with permission.

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 11 - 93 May 10, 2017

# DeepDream

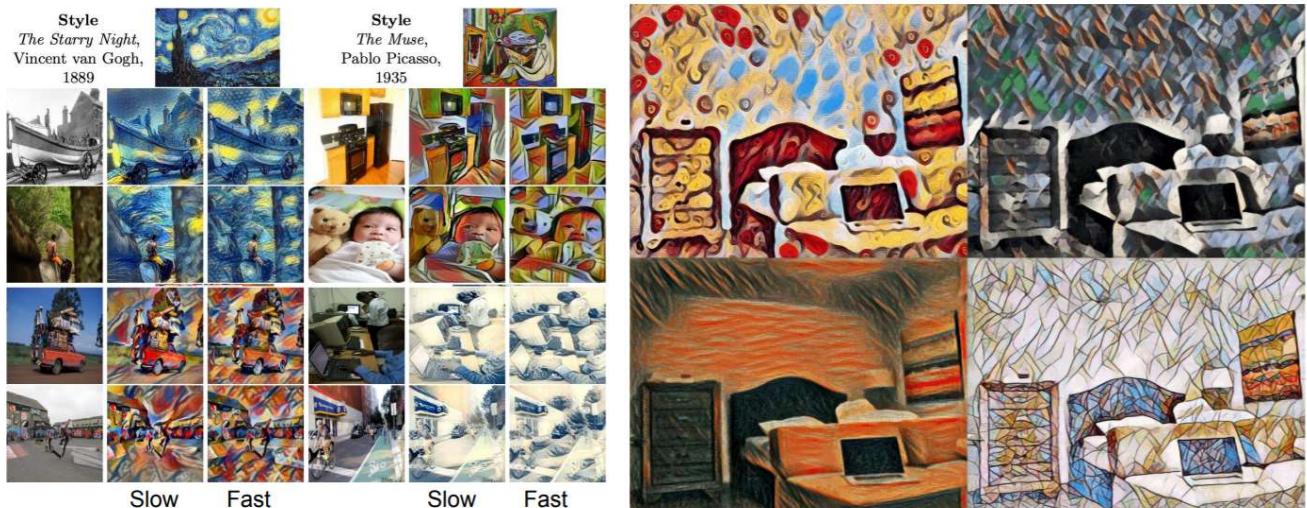


Fei-Fei Li &amp; Justin Johnson &amp; Serena Yeung

Lecture 11 - 45 May 10, 2017

## Style-transfer

### Fast Style Transfer



Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016  
Figure copyright Springer, 2016. Reproduced for educational purposes.

<https://github.com/jcjohnson/fast-neural-style>

Fei-Fei Li &amp; Justin Johnson &amp; Serena Yeung

Lecture 11 - 79 May 10, 2017

# Gramian 矩陣

設  $A$  為一個  $m \times n$  階實矩陣， $n$  階方陣  $G = [g_{ij}] = A^T A$  稱為 Gramian 或 Gram 矩陣，也有人稱之為交互乘積 (cross-product) 矩陣。考慮  $A$  的行向量表達式  $A = [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \cdots \quad \mathbf{a}_n]$ ,  $\mathbf{a}_i \in \mathbb{R}^m$ ，則：

$$G = A^T A = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_n^T \end{bmatrix} [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \cdots \quad \mathbf{a}_n] = \begin{bmatrix} \mathbf{a}_1^T \mathbf{a}_1 & \mathbf{a}_1^T \mathbf{a}_2 & \cdots & \mathbf{a}_1^T \mathbf{a}_n \\ \mathbf{a}_2^T \mathbf{a}_1 & \mathbf{a}_2^T \mathbf{a}_2 & \cdots & \mathbf{a}_2^T \mathbf{a}_n \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_n^T \mathbf{a}_1 & \mathbf{a}_n^T \mathbf{a}_2 & \cdots & \mathbf{a}_n^T \mathbf{a}_n \end{bmatrix}$$

In [23]:

```
import numpy as np

A = np.random.randint(0, 5, (2, 2))

B = np.dot(A.T, A)
print(A)
print(B)
```

```
[[1 1]
 [3 4]]
[[10 13]
 [13 17]]
```

In [2]:

```

        parser.add_argument('--content_layers', nargs='+', type=str, default=['conv4_2'], help=
'VGG19 layers used for content loss')
        parser.add_argument('--style_layers', nargs='+', type=str, default=['relu1_1', 'relu2_1',
, 'relu3_1', 'relu4_1', 'relu5_1'], help='VGG19 layers used for style loss')

y"""
    """https://github.com/hwalsuklee/tensorflow-style-transfer/blob/master/style_transfer.p
y"""
    """ compute loss """
L_content = 0
L_style = 0
for id in self.Fs:
    if id in self.CONTENT_LAYERS:
        ## content loss ##

        F = self.Fs[id]           # content feature of x
        P = self.Ps[id]           # content feature of p

        _, h, w, d = F.get_shape() # first return value is batch size (must be one)
        N = h.value*w.value       # product of width and height
        M = d.value               # number of filters

        w = self.CONTENT_LAYERS[id]# weight for this layer

        # You may choose different normalization constant
        if self.content_loss_norm_type==1:
            L_content += w * tf.reduce_sum(tf.pow((F-P), 2)) / 2 # original paper
        elif self.content_loss_norm_type == 2:
            L_content += w * tf.reduce_sum(tf.pow((F-P), 2)) / (N*M) #artistic style tra
nsfer for videos
        elif self.content_loss_norm_type == 3: # this is from https://github.com/cysmit
h/neural-style-tf/blob/master/neural_style.py
            L_content += w * (1. / (2. * np.sqrt(M) * np.sqrt(N))) * tf.reduce_sum(tf.po
w((F - P), 2))

    elif id in self.STYLE_LAYERS:
        ## style loss ##

        F = self.Fs[id]

        _, h, w, d = F.get_shape() # first return value is batch size (must be one)
        N = h.value * w.value     # product of width and height
        M = d.value               # number of filters

        w = self.STYLE_LAYERS[id] # weight for this layer

        G = self._gram_matrix(F)   # style feature of x
        A = self.As[id]           # style feature of a

        L_style += w * (1. / (4 * N ** 2 * M ** 2)) * tf.reduce_sum(tf.pow((G-A), 2))

```

File "&lt;ipython-input-2-e8223f7ccc10&gt;", line 2

L\_content = 0

IndentationError: unexpected indent

# CNN GAN

## Generative Adversarial Nets: Interpretable Vector Math

Glasses man



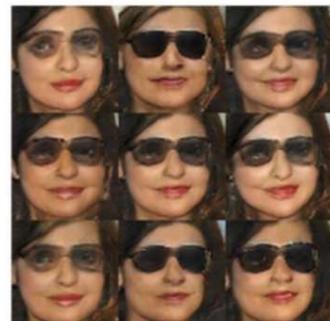
No glasses man



No glasses woman

Radford et al,  
ICLR 2016

Woman with glasses



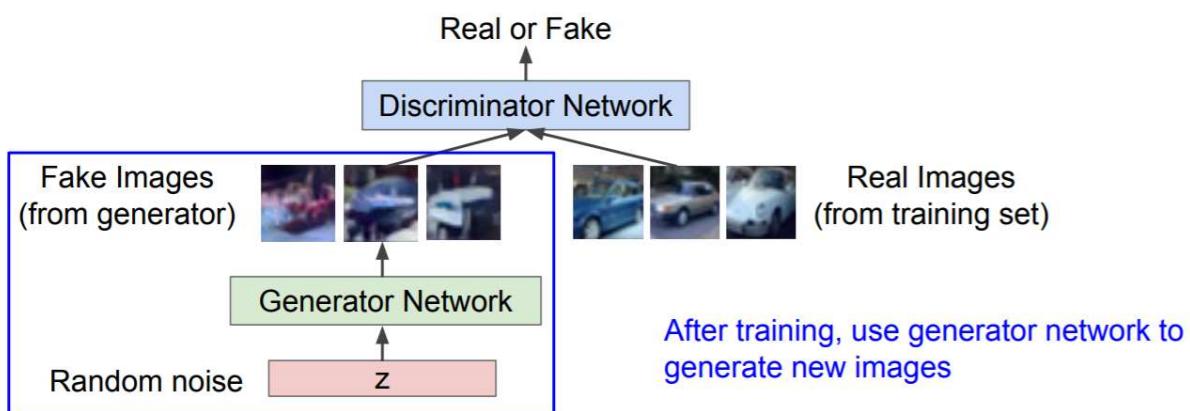
Fei-Fei Li &amp; Justin Johnson &amp; Serena Yeung

Lecture 13 -  $\frac{12}{6}$  May 18, 2017

## GAN Training

### Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

**Generator network:** try to fool the discriminator by generating real-looking images**Discriminator network:** try to distinguish between real and fake images

Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

Fei-Fei Li &amp; Justin Johnson &amp; Serena Yeung

Lecture 13 -  $\frac{11}{5}$  May 18, 2017

# Sequential Model

- Sequence of words in an English sentence
- Acoustic features at successive time frames in speech recognition
- Successive frames in video classification
- Rainfall measurements on successive days in Hong Kong
- Daily values of current exchange rate
- Nucleotide base pairs in a strand of DNA
- **Instead of making independent predictions on samples, assume the dependency among samples and make a sequence of decisions for sequential samples**

## Modeling sequential data

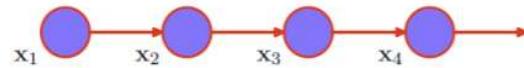
- Use the chain rule to express the joint distribution for a sequence of observations
$$p(\mathbf{x}_1, \dots, \mathbf{x}_T) = \prod_{t=1}^T p(\mathbf{x}_t | \mathbf{x}_1, \dots, \mathbf{x}_{t-1})$$
- Impractical to consider general dependence of future dependence on all previous observations  $p(\mathbf{x}_t | \mathbf{x}_{t-1}, \dots, \mathbf{x}_0)$ 
  - Complexity would grow without limit as the number of observations increases
- It is expected that recent observations are more informative than more historical observations in predicting future values

# Markov model

## Markov models

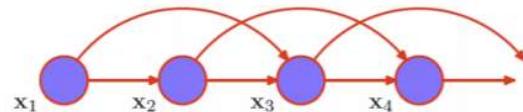
- Markov models assume dependence on most recent observations
- First-order Markov model

$$p(\mathbf{x}_1, \dots, \mathbf{x}_T) = \prod_{t=1}^T p(\mathbf{x}_t | \mathbf{x}_{t-1})$$

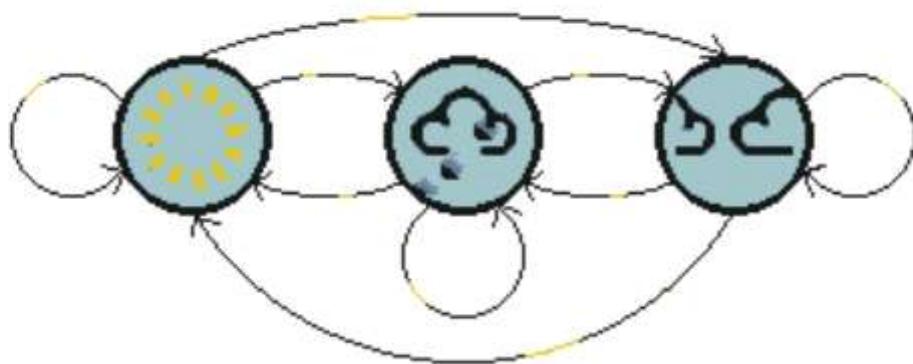


- Second-order Markov model

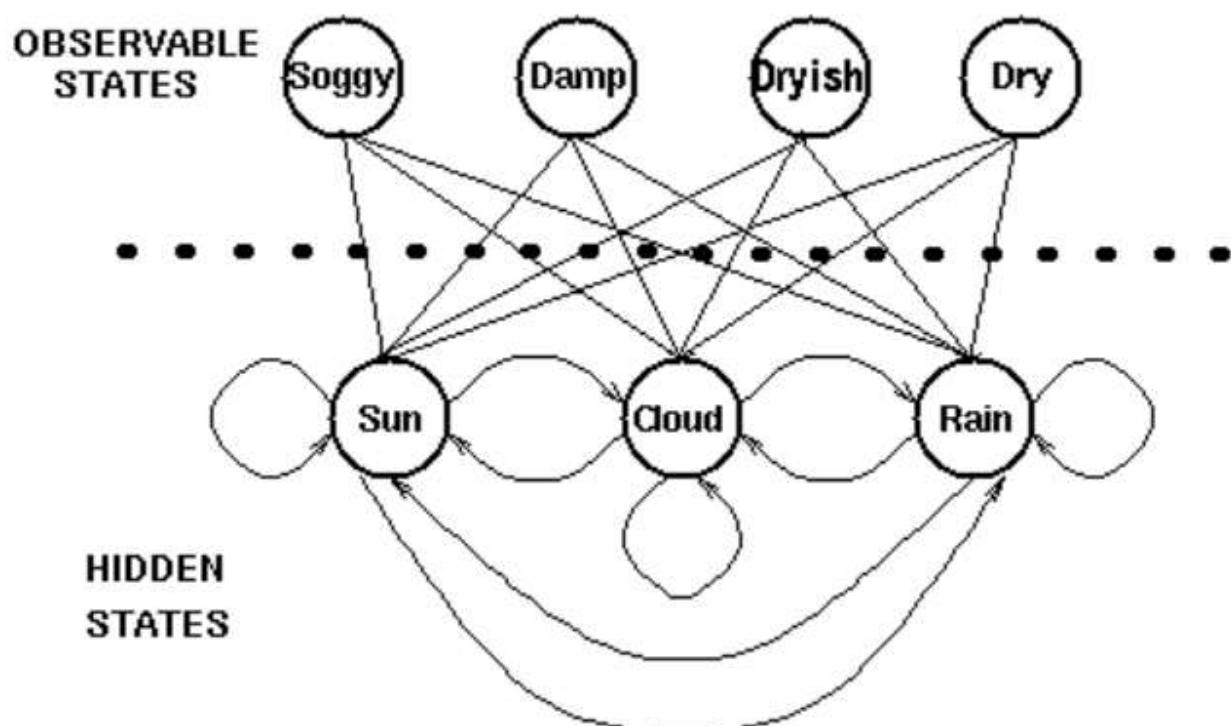
$$p(\mathbf{x}_1, \dots, \mathbf{x}_T) = \prod_{t=1}^T p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_{t-2})$$



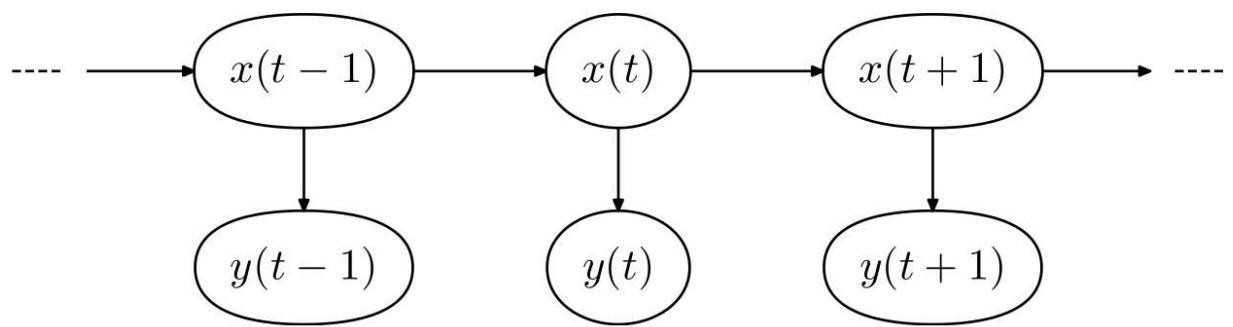
## Markov model



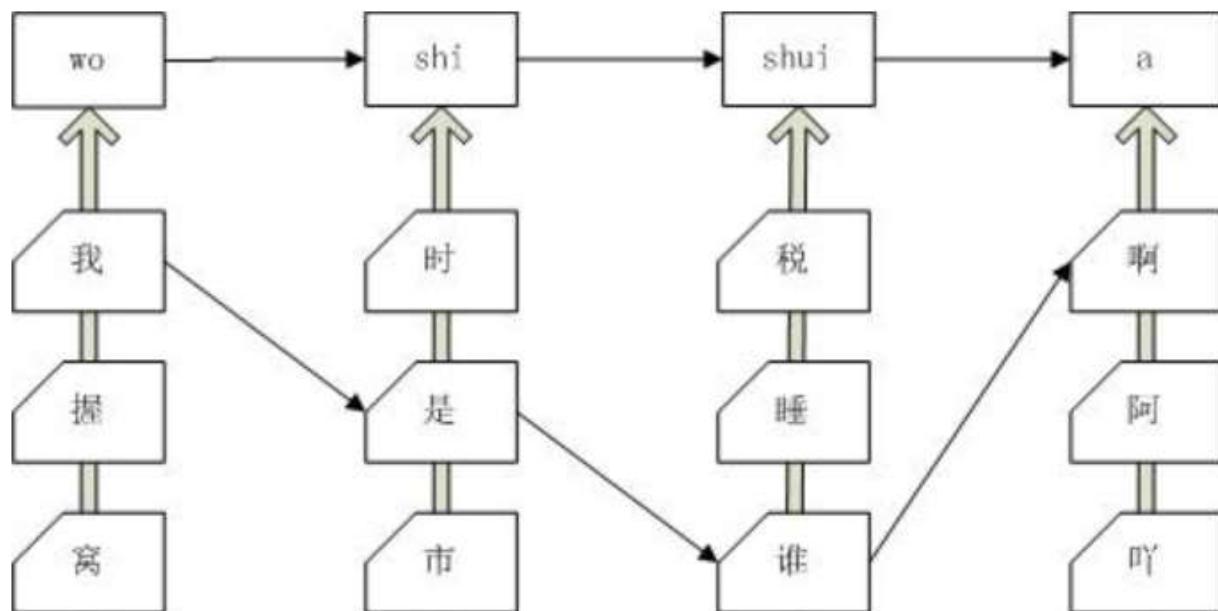
## Hidden Markov model



## Hidden Markov model



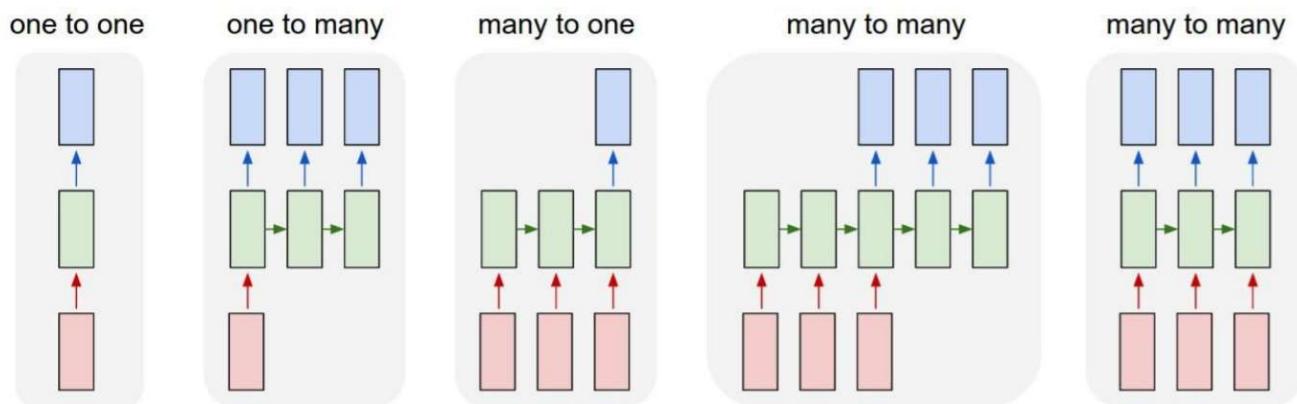
## HMM 输入法应用



# RNN

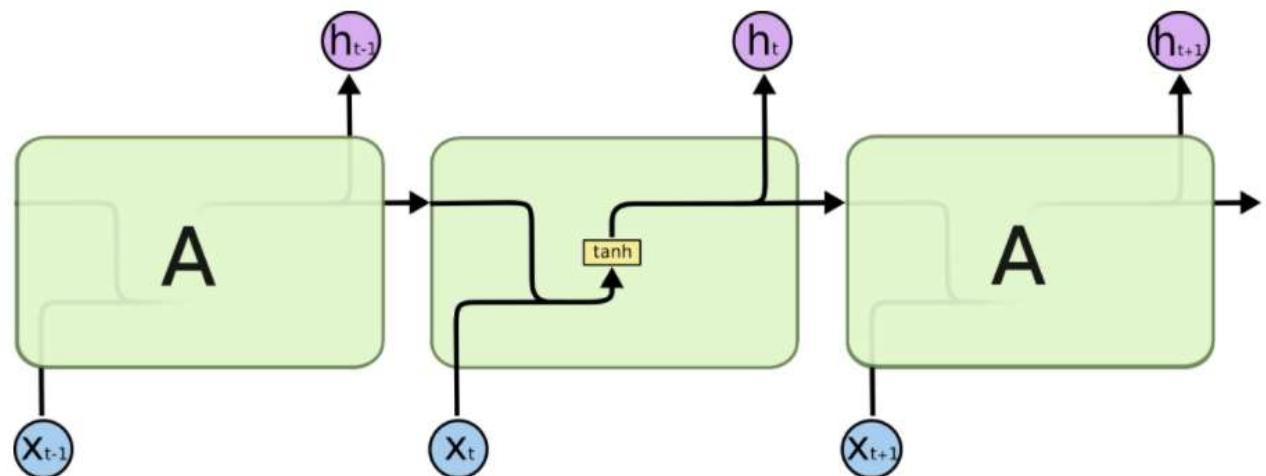
- image captioning
- sentiment classification
- machine translation
- video classification on frame level

## Recurrent Neural Networks: Process Sequences



# RNN

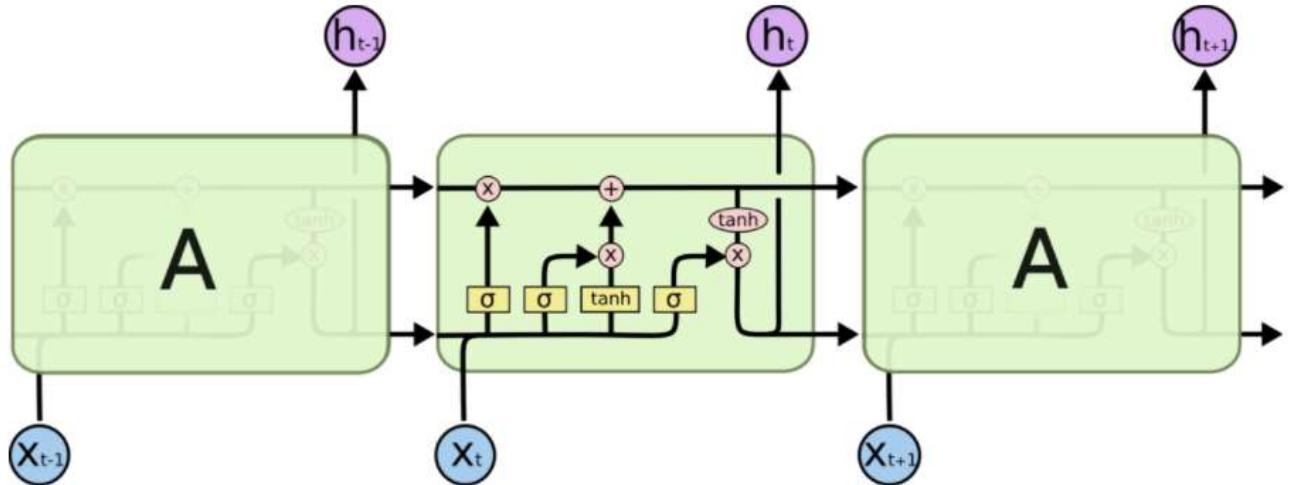
$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b)$$



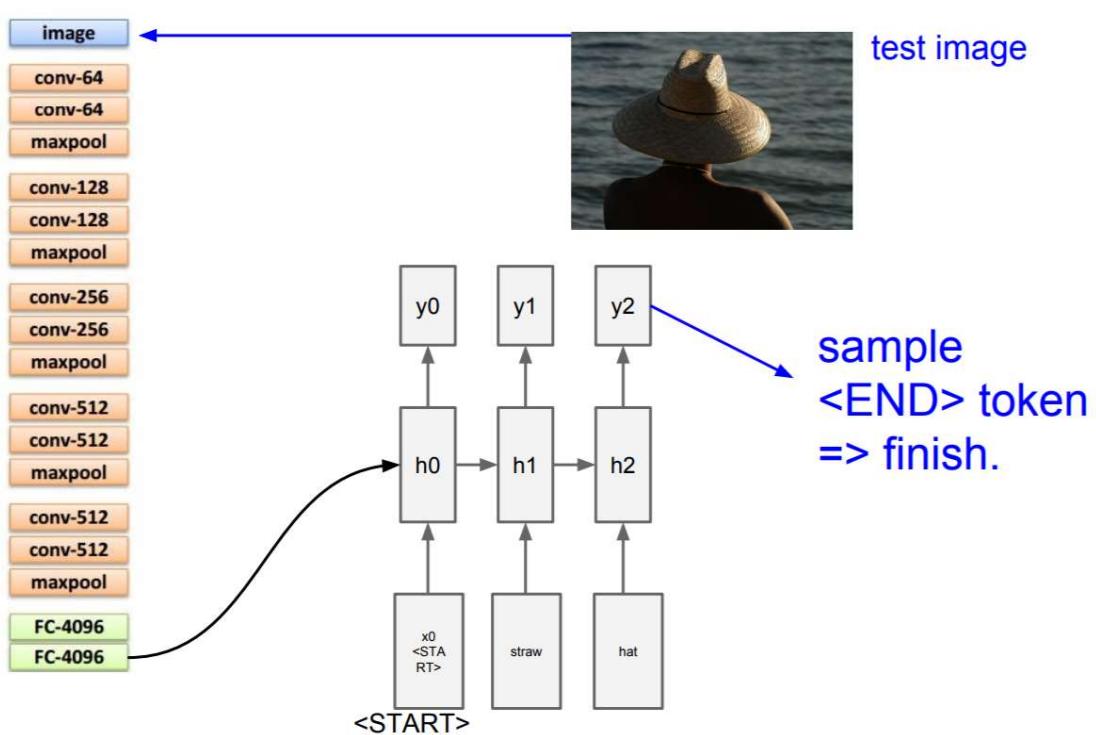
# LSTM

Colah LSTM : <http://colah.github.io/posts/2015-08-Understanding-LSTMs/> (<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>)

Colah中文版: <https://www.yunaitong.cn/understanding-lstm-networks.html>  
[\(https://www.yunaitong.cn/understanding-lstm-networks.html\)](https://www.yunaitong.cn/understanding-lstm-networks.html)



## Image Captioning



# Image Captioning: Example Results

Captions generated using neuraltalk2  
All images are CC0 Public domain:  
cat suitcase, cat tree, dog, bear,  
surfers, tennis, giraffe, motorcycle



A cat sitting on a suitcase on the floor



A cat is sitting on a tree branch



A dog is running in the grass with a frisbee



A white teddy bear sitting in the grass



Two people walking on the beach with surfboards



A tennis player in action on the court



Two giraffes standing in a grassy field



A man riding a dirt bike on a dirt track

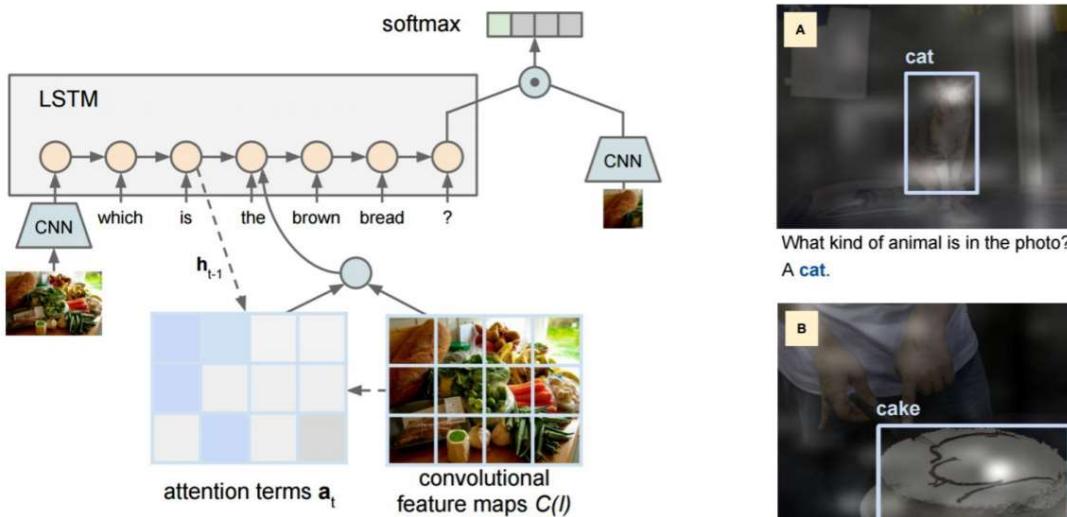
Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 10 - 75 May 4, 2017

```
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000fffff8) & 0x000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}
```

Generated  
C code

# Visual Question Answering: RNNs with Attention



Zhu et al., "Visual 7W: Grounded Question Answering in Images", CVPR 2016  
Figures from Zhu et al, copyright IEEE 2016. Reproduced for educational purposes.



What kind of animal is in the photo?

A cat.



Why is the person holding a knife?

To cut the cake with.

## RNN中为什么要采用tanh而不是ReLU作为激活函数？

明确的提到的：

Recent research on deep feedforward networks has also produced some impressive results [19, 3] and there is now a consensus that for deep networks, rectified linear units (ReLUs) are easier to train than the logistic or tanh units that were used for many years [27, 40]. At first sight, ReLUs seem inappropriate for RNNs because they can have very large outputs so they might be expected to be far more likely to explode than units that have bounded values. A second aim of this paper is to explore whether ReLUs can be made to work well in RNNs and whether the ease of optimizing them in feedforward nets transfers to RNNs.

也就是说在RNN中直接把激活函数换成ReLU会导致非常大的输出值。为了讲清楚这一点，我们先用同上面相似的符号把原始的RNN表示出来：

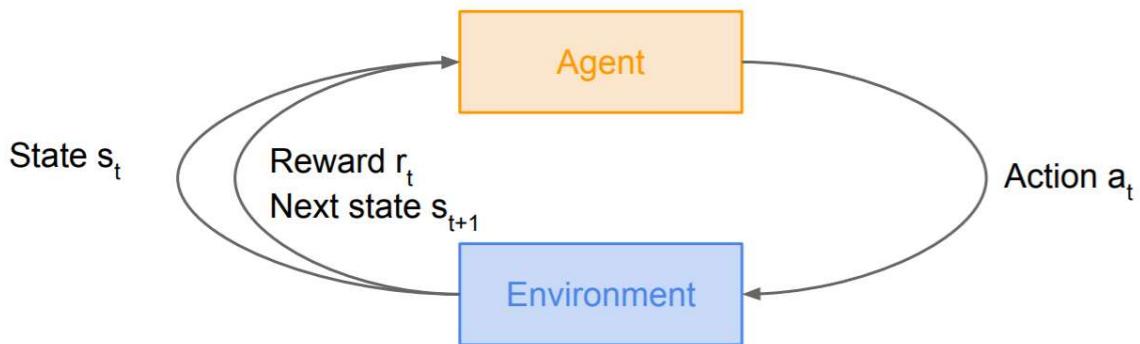
$$a_i = Wf_{i-1} + Ux_i + b_i ,$$

$$f_i = f[a_i]$$

在这个表示中，RNN每个阶段的输入是  $x_i$ ，和CNN每一层使用独立的参数  $W_i$  不同，原始的 RNN 在每个阶段都共享一个参数  $W$ 。如果我们假设从某一层开始输入  $x_i$  和偏置  $b_i$  都为0，那么最后得到的输出就是  $f[W \dots [Wf[Wf[Wf_i]]]]$ ，这在某种程度上相当于对参数矩阵  $W$  作连乘，很显然，只要  $W$  有一个大于1的特征值，在经过若干次连乘后都会导致结果是一个数值非常庞大的矩阵。

# Reinforcement Learning

## Reinforcement Learning

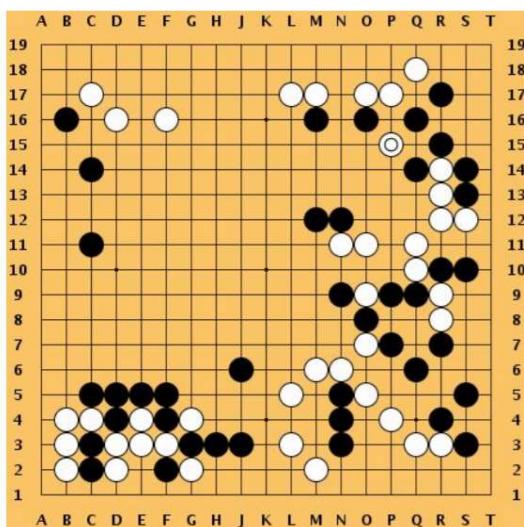


Fei-Fei Li &amp; Justin Johnson &amp; Serena Yeung

Lecture 14 - 13

May 23, 2017

## Go



**Objective:** Win the game!

**State:** Position of all pieces

**Action:** Where to put the next piece down

**Reward:** 1 if win at the end of the game, 0 otherwise

This image is CC0 public domain

Fei-Fei Li &amp; Justin Johnson &amp; Serena Yeung

Lecture 14 - 17

May 23, 2017

# Q-learning

Q-learning demo 英文版 : [\(http://mnemstudio.org/path-finding-q-learning-tutorial.htm\)](http://mnemstudio.org/path-finding-q-learning-tutorial.htm)

Q-learning demo 中文版: [\(http://www.cnblogs.com/stevenbush/p/3359603.html\)](http://www.cnblogs.com/stevenbush/p/3359603.html)

## 算法 1.1 (*Q-learning 算法*)

**Step 1** 给定参数  $\gamma$  和 reward 矩阵  $R$ .

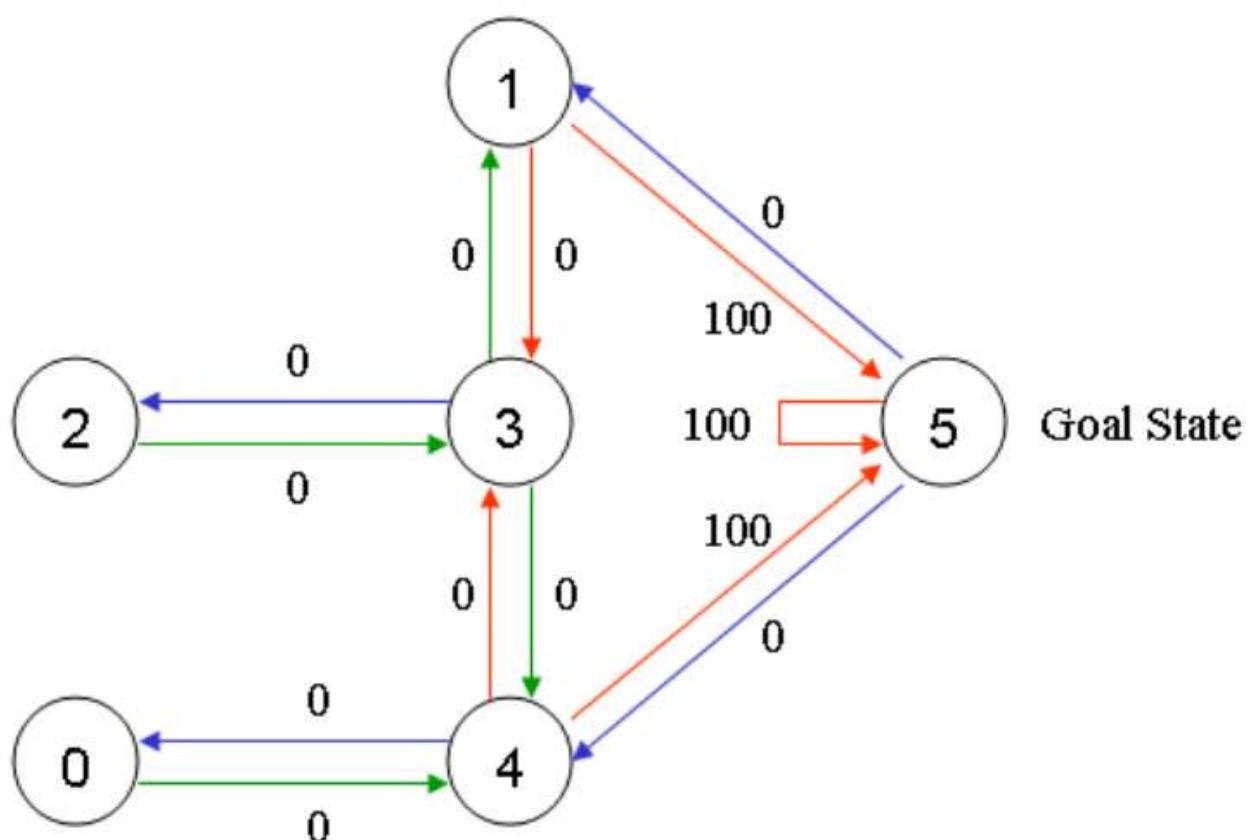
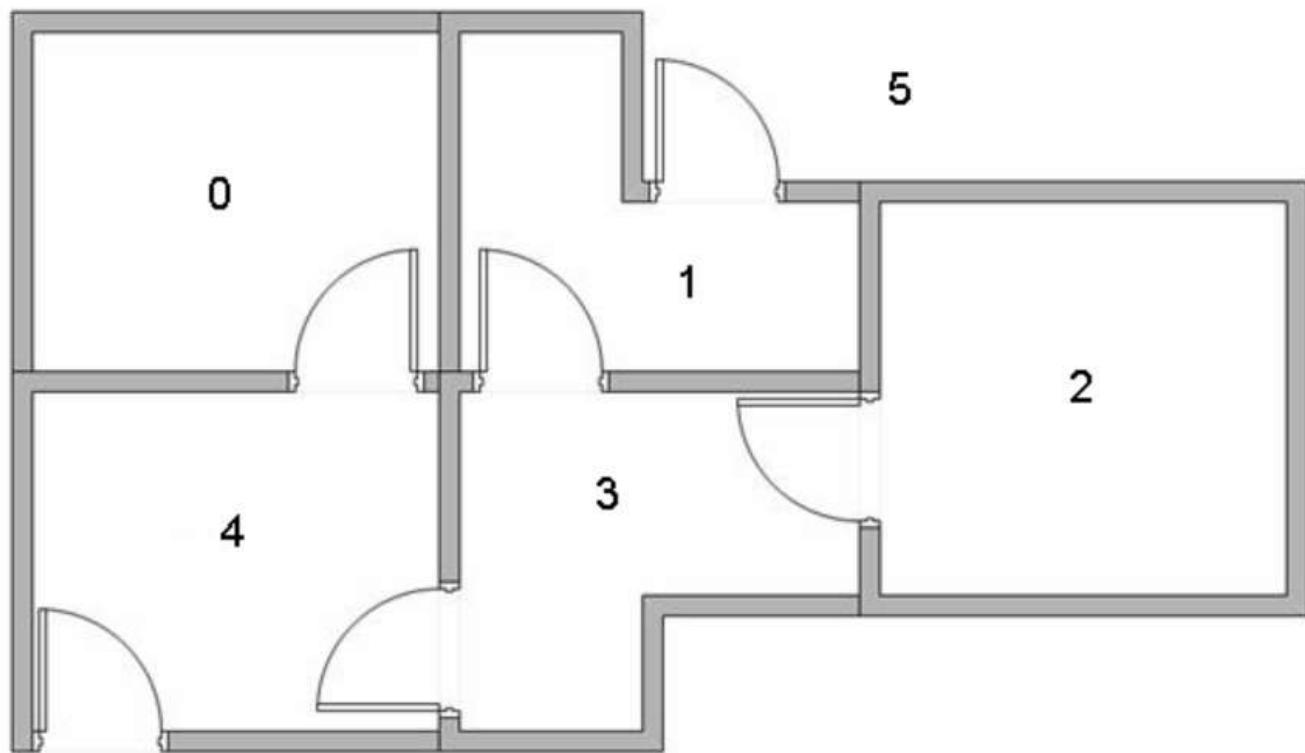
**Step 2** 令  $Q := 0$ .

**Step 3** For each episode:

    3.1 随机选择一个初始的状态  $s$ .

    3.2 若未达到目标状态, 则执行以下几步

- (1) 在当前状态  $s$  的所有可能行为中选取一个行为  $a$ .
- (2) 利用选定的行为  $a$ , 得到下一个状态  $\tilde{s}$ .
- (3) 按照 (1.1) 计算  $Q(s, a)$ .
- (4) 令  $s := \tilde{s}$ .

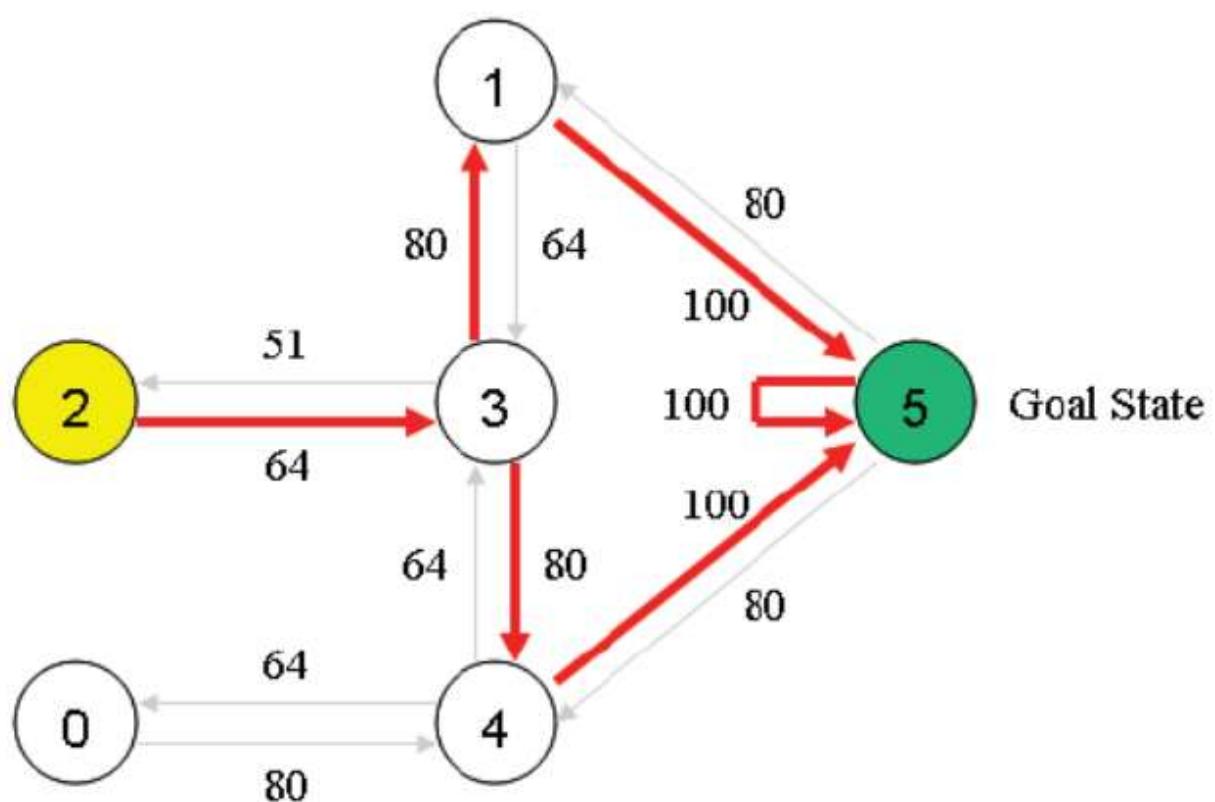


State	Action					
	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

$$Q(1, 5) = R(1, 5) + 0.8 * \max\{Q(5, 1), Q(5, 4), Q(5, 5)\} = 100 + 0.8 * \max\{0, 0, 0\} = 100$$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$$Q = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \left[ \begin{matrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{matrix} \right] \end{matrix}$$



$$2. Q(S, A) \leftarrow (1 - \alpha) * Q(S, A) + \alpha * [R + \gamma * \max_a Q(S', a)]$$

这个就是Q-learning的训练公式了。其中 $\alpha$ 为**学习速率** ( learning rate ) ,  $\gamma$ 为**折扣因子** ( discount factor )。根据公式可以看出，**学习速率** $\alpha$ 越大，保留之前训练的效果就越少。折扣因子 $\gamma$ 越大， $\max_a Q(S', a)$  所起到的作用就越大。但 $\max_a Q(S', a)$  指什么呢？

小鸟在对状态进行更新时，会考虑到**眼前利益** ( R )，和**记忆中的利益** (  $\max_a Q(S', a)$  )。

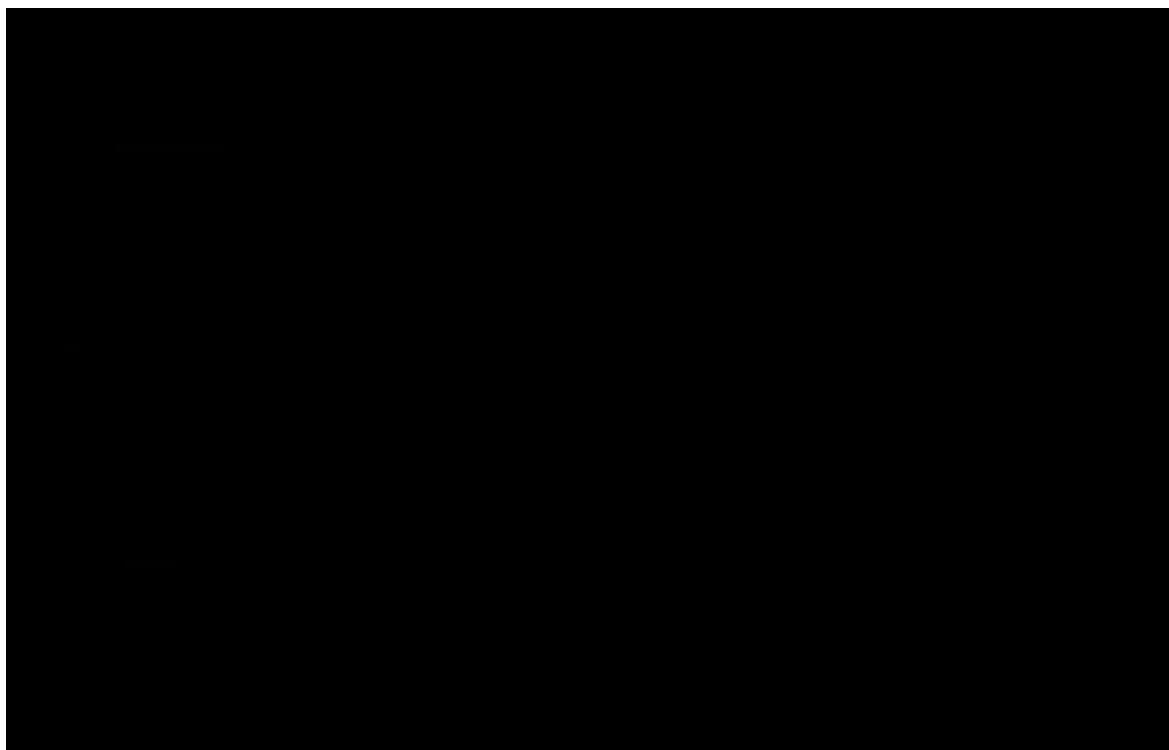
$\max_a Q(S', a)$  指的便是**记忆中的利益**。它是指小鸟记忆里下一个状态  $S'$  的动作中效用值的最大值。如果小鸟之前在下一个状态  $S'$  的某个动作上吃过甜头（选择了某个动作之后获得了50的奖赏），那么它就更希望提早地得知这个消息，以便下回在状态  $S$  可以通过选择正确的动作继续进入这个吃甜头的状态  $S'$ 。

可以看出， $\gamma$ 越大，小鸟就会越重视以往经验，越小，小鸟只重视眼前利益 ( R )。

In [1]:

```
from IPython.display import HTML  
  
# Youtube  
HTML('<iframe width="720" height="480" src="sources/Bidirectionally-Coordinated Nets (BiCNet) fo  
r Playing StarCraft Combats.mp4?rel=0&controls=0&showinfo=0" frameborder="0" allowfullsc  
reen></iframe>')
```

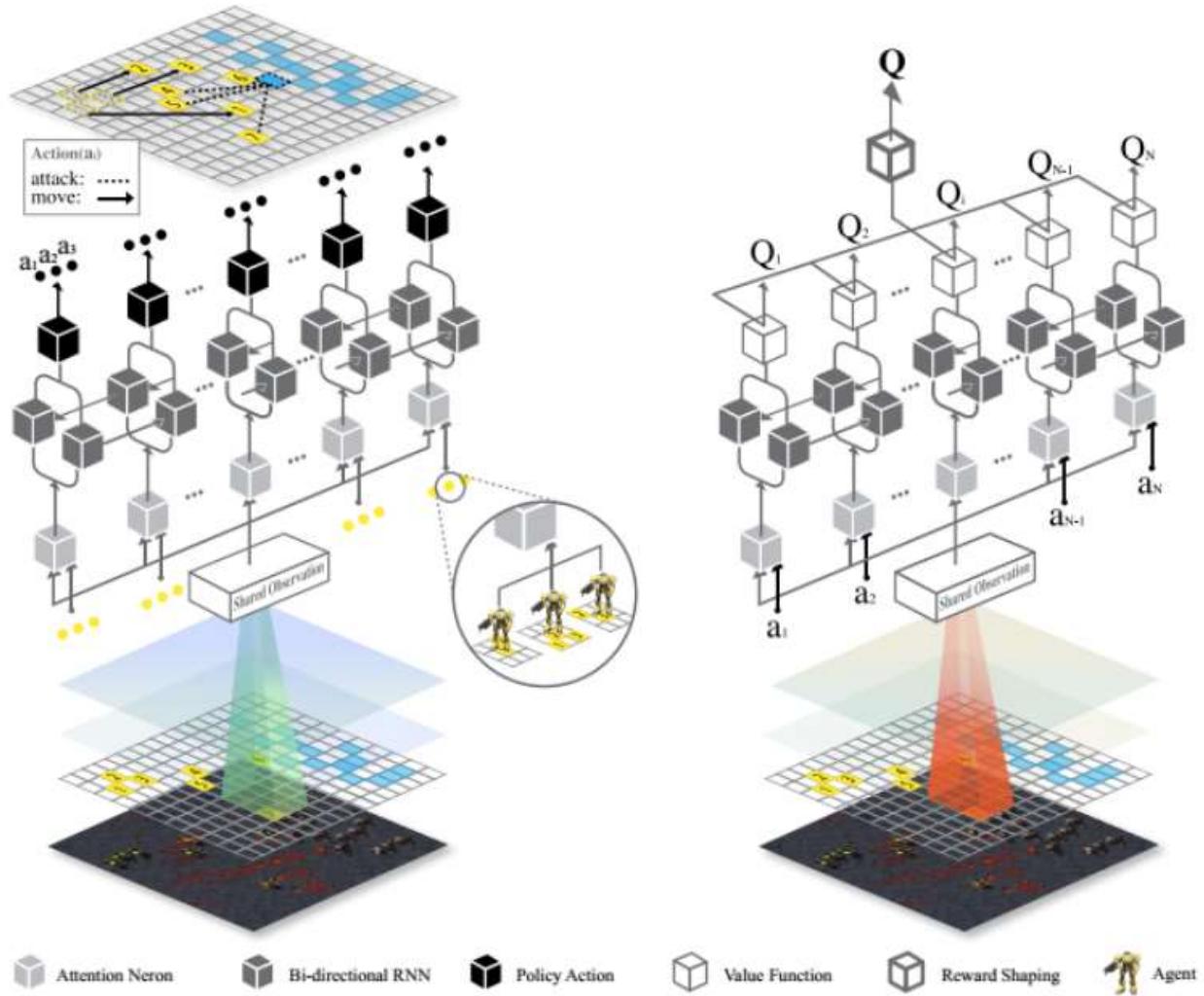
Out[1]:



2:17 / 2:17

# Mastering StarCraft with AI

Researchers from Alibaba and University College London developed a deep learning-based system that learned how to execute a number of strategies for the popular real-time strategy game StarCraft.



(a) Multiagent policy networks with grouping      (b) Multiagent Q networks with reward shaping  
Figure 1: Bidirectionally-Coordinated Net (BiCNet).

## ML问题点

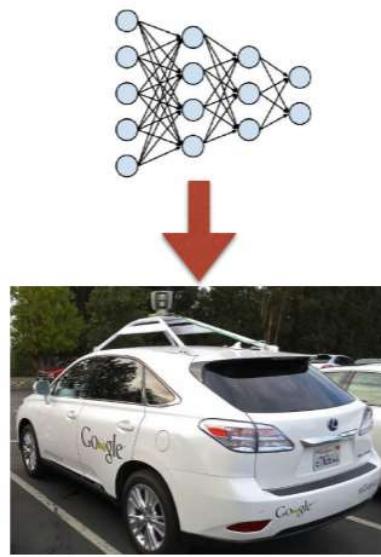
- size
- time
- energy

## The first Challenge: Model Size

Hard to distribute large models through over-the-air update



App icon is in the public domain  
Phone image is licensed under CC-BY 2.0



This image is licensed under CC-BY 2.0

Stanford University

5

## The Second Challenge: Speed

	Error rate	Training time
ResNet18:	10.76%	2.5 days
ResNet50:	7.02%	5 days
ResNet101:	6.21%	1 week
ResNet152:	6.16%	1.5 weeks

Such long training time limits ML researcher's productivity

# The Third Challenge: Energy Efficiency



[This image](#) is in the public domain

AlphaGo: 1920 CPUs and 280 GPUs,  
**\$3000 electric bill per game**



[This image](#) is in the public domain



[Phone image](#) is licensed under CC-BY 2.0

on mobile: **drains battery**  
on data-center: **increases TCO**



[This image](#) is licensed under CC-BY 2.0