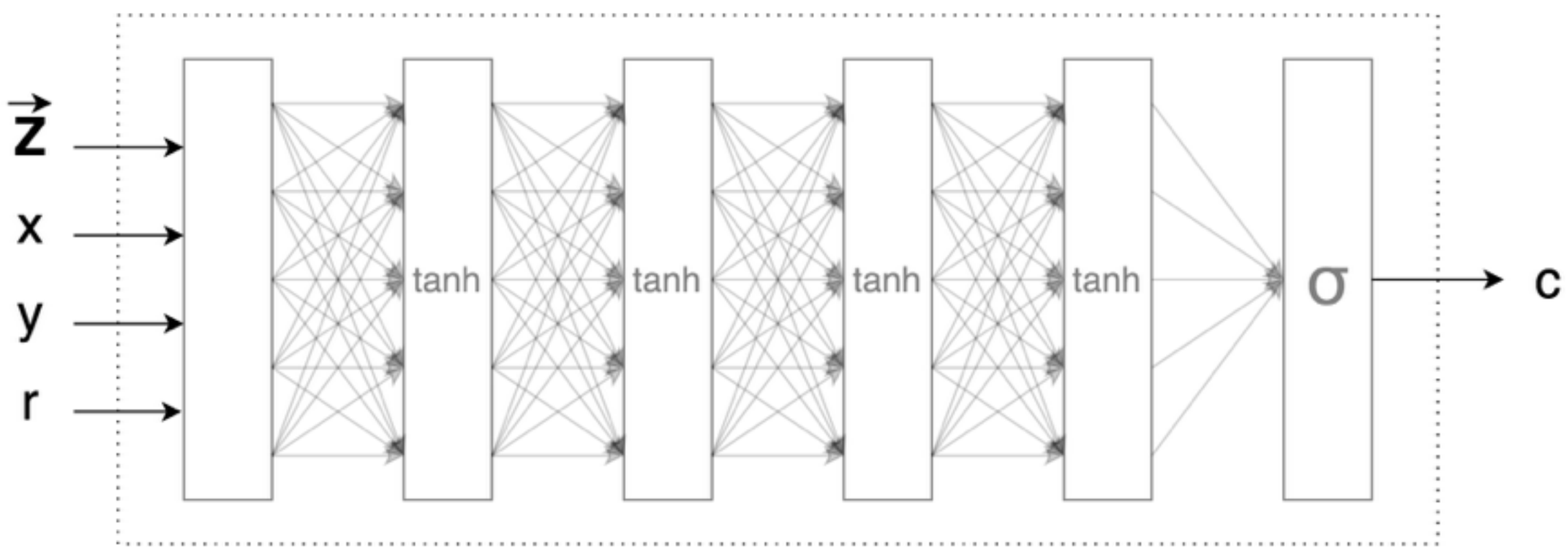
9. 更多的网络类型

丰富的神经网络类型

- 深度学习除了经典的卷积网络、循环网络之外还有广泛的网络类型，应用在不同的问题。

一个简单但是有趣的例子

- CPPN (Compositional Pattern Producing Network)



http://zhouchang.info/blog/2016-04-08/simple-cppn.html

- 网络输入是像素的坐标值（x,y）
- 网络输出是像素的RGB值
- 把[0,0] - [100,100]坐标逐个输入，将输出的RGB值组成完整图像，将会是什么样子?

怎么实现的?

```
require 'torch'
require 'image'
require 'nn'


local ConvLayer = function(in_size, out_size)
    local conv = nn.SpatialConvolutionMM(in_size, out_size, 1, 1)
    conv.weight:normal(0, 1.0)

    return conv
end

local buildNet = function(input_size, output_size, net_number, layer_number)
    local net = nn.Sequential()

    net:add( ConvLayer(input_size, net_number) )
    net:add( nn.Tanh() )
    for i = 1, layer_number - 1 do
        net:add( ConvLayer(net_number, net_number) )
        net:add( nn.Tanh() )
    end
    net:add( ConvLayer(net_number, output_size) )
    net:add( nn.Sigmoid() )

    return net
end
```

```lua
local buildXYRInput = function(width, height)
    local xyr_in = torch.Tensor(3, height, width)

    for i = 1, width do
        xyr_in[{1, 1, i}] = (i - width/2)
    end
    for i = 2, height do
        xyr_in[{1, i, {}}]:copy( xyr_in[{1, 1, {}}] )
    end

    for i = 1, height do
        xyr_in[{2, i, 1}] = (i - height/2)
    end
    for i = 2, width do
        xyr_in[{2, {}, i}]:copy( xyr_in[{2, {}, 1}] )
    end

    xyr_in[3] = torch.cmul(xyr_in[1], xyr_in[1] ) + torch.cmul(xyr_in[2], xyr_in[2])
    xyr_in[3]:sqrt()

    return xyr_in
end
local net = buildNet(3, 3, 32, 5)
local xyr = buildXYRInput(640, 640)
xyr = xyr / 500.0
local out = net:forward(xyr)
image.savePNG('./output.png',out)
```

http://blog.otoro.net/2016/03/25/generating-abstract-patterns-with-tensorflow/

孪生网络(Siamese)

- 孪生网络(Siamese network)是一种网络结构，通过一个NN将样本的维度降低到某个较低的维度。

- 在低维空间，任意两个样本：

  - 如果它们是相同类别，空间距离尽量接近0

  - 如果它们是不同类别，空间距离大于某个间隔

https://github.com/Teaonly/easyLearning.io/tree/master/siamese_network

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
HingeEmbeddingCriterion

 criterion = nn.HingeEmbeddingCriterion([margin])

Creates a criterion that measures the loss given an input x which is
a 1-dimensional vector and a label y ( 1 or -1 ).
This is usually used for measuring whether two inputs are similar or dissimilar,
e.g. using the L1 pairwise distance, and is typically used for learning
nonlinear embeddings or semi-supervised learning.
```

$$\text{loss}(x, y) = 1/n \begin{cases} x_i, & \text{if } y_i == 1 \\ \max(0, \text{margin} - x_i), & \text{if } y_i == -1 \end{cases}$$

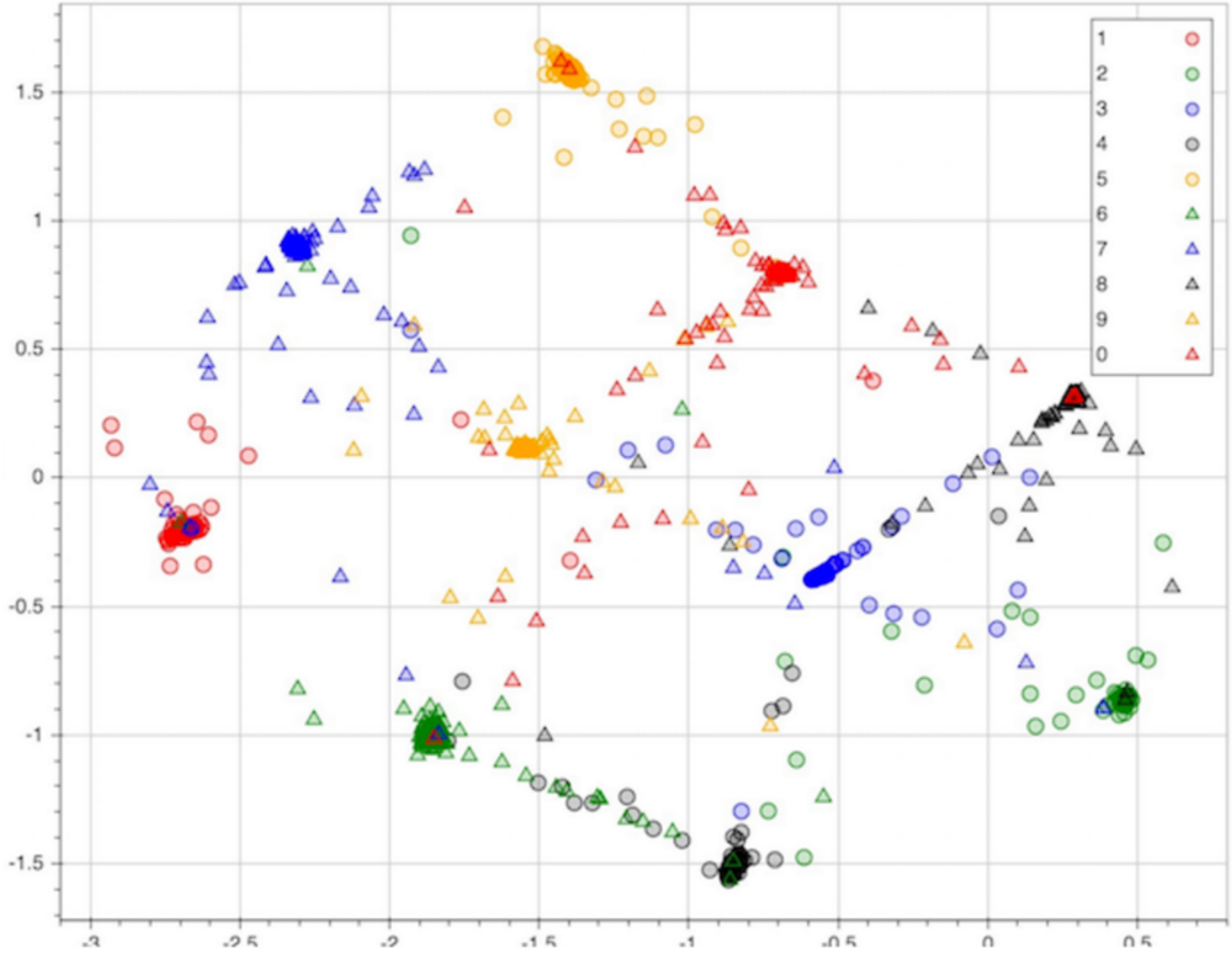
```
If x and y are n -dimensional Tensor s, the sum operation still
operates over all the elements, and divides by n (this can be avoided
if one sets the internal variable sizeAverage to false ). The margin
has a default value of 1 , or can be set in the constructor.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```lua
require 'nn'

function build_model()

    local cnn = nn.Sequential()
    cnn:add(nn.SpatialConvolution(1, 8, 3, 3, 1, 1, 1))
    cnn:add(nn.ReLU())
    cnn:add(nn.SpatialMaxPooling(2, 2))
    cnn:add(nn.SpatialConvolution(8, 16, 3, 3, 1, 1, 1))
    cnn:add(nn.ReLU())
    cnn:add(nn.SpatialMaxPooling(2, 2))
    cnn:add(nn.SpatialConvolution(16, 16, 3, 3, 1, 1, 1))
    cnn:add(nn.ReLU())
    cnn:add(nn.SpatialMaxPooling(2, 2))
    cnn:add(nn.SpatialConvolution(16, 32, 3, 3, 1, 1, 1))
    cnn:add(nn.ReLU())

    cnn:add(nn.Reshape(32*4*4))
    cnn:add(nn.Linear(32*4*4, 512))
    cnn:add(nn.ReLU())
    cnn:add(nn.Linear(512, 2))

    --The siamese model
    --clone the encoder and share the weight, bias. Must also share the gradWeight and gradBias
    siamese_encoder = nn.ParallelTable()
    siamese_encoder:add(cnn)
    siamese_encoder:add(cnn:clone('weight','bias', 'gradWeight','gradBias'))

    --The siamese model (inputs will be Tensors of shape (2, channel, height, width))
    model = nn.Sequential()
    model:add(nn.SplitTable(2)) -- split input tensor along the rows (1st dimension) to table for input to ParallelTable
    model:add(siamese_encoder)
    model:add(nn.PairwiseDistance(2)) --L2 pariwise distance

    return model
end
```

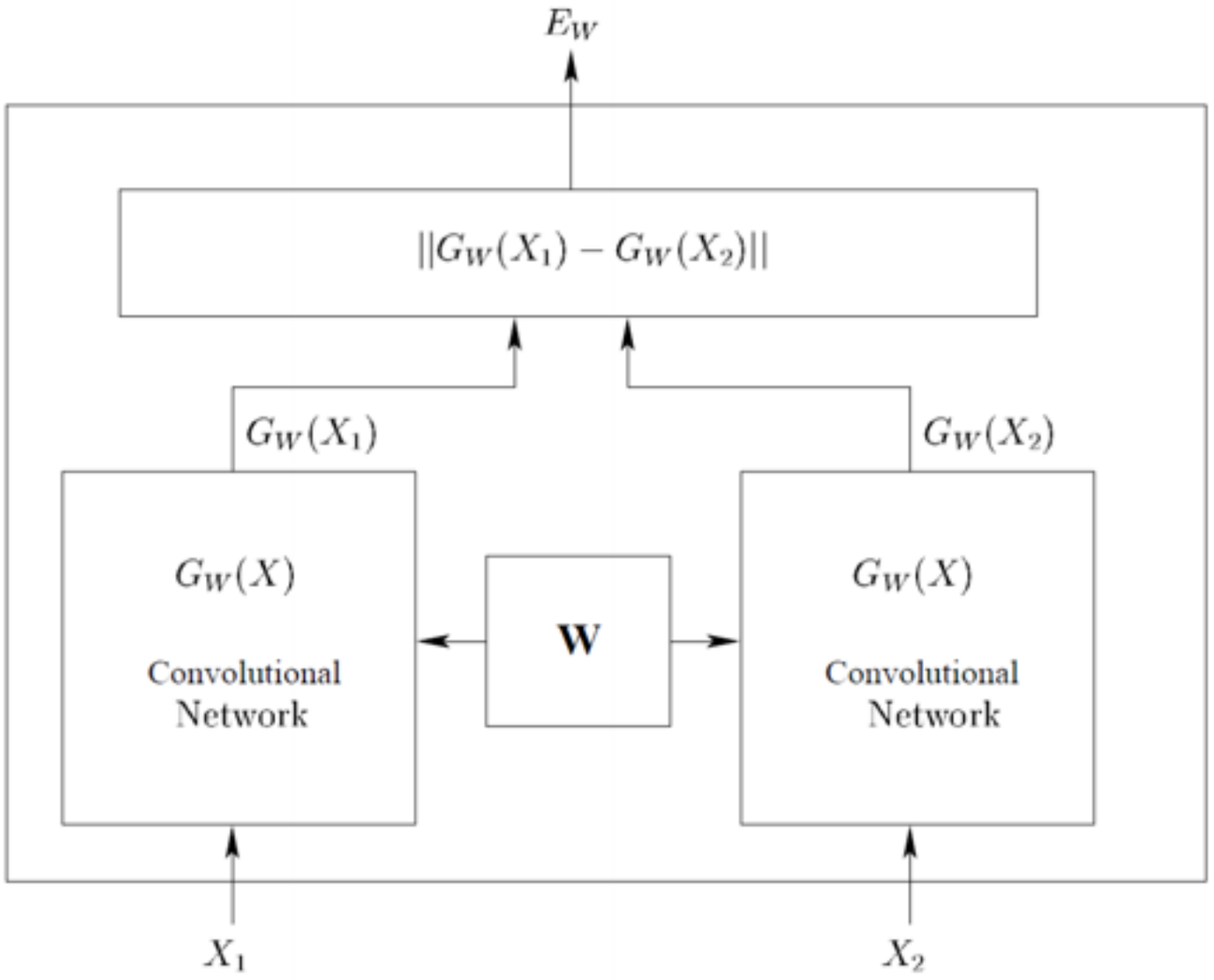
```lua
local loadBatch = function(index, isTest)
    local ii = index or 1

    local data = g.allTrainSamples.data
    local labels = g.allTrainSamples.labels
    if ( isTest ) then
        data = g.allTestSamples.data
        labels = g.allTestSamples.labels
    end

    local number = data:size(1)

    local batch = {}
    batch.data = torch.Tensor(g.batchSize, 2, 1, 32, 32)
    batch.label = torch.zeros(g.batchSize) - 1
    for i = 1, g.batchSize do
        batch.data[i][1]:copy ( data[ii] )
        local iii = g.siamese[ii]
        batch.data[i][2]:copy ( data[iii] )

        if ( labels[ii] == labels[iii] ) then
            batch.label[i] = 1
        end

        ii = ii + 1
        if ( ii > number ) then
            ii = 1
        end
    end

    if _CUDA_ then
        batch.data = batch.data:cuda()
        batch.label = batch.label:cuda()
    end

    return batch, ii
end
```

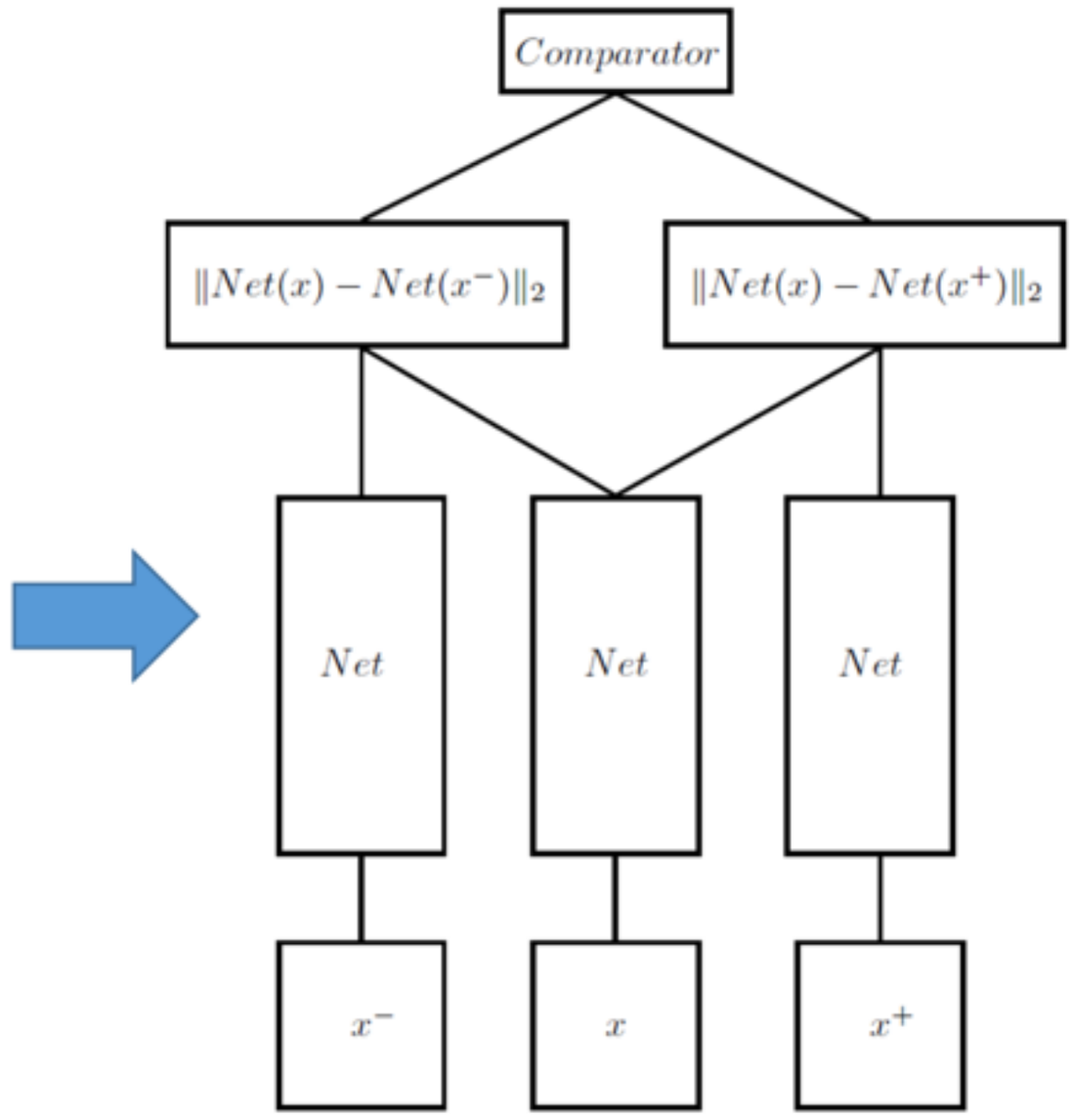
```lua
local doTrain = function()
    print(">>>>>>>>>>>>>>TRAINING>>>>>>>>>>>>>>");

    if _CUDA_ then
        g.model:cuda()
        g.criterion:cuda()
    end
    g.model:training()

    local parameters,gradParameters = model:getParameters()
    local batch = nil
    local feval = function(x)
        -- get new parameters
        if x ~= parameters then
            parameters:copy(x)
        end

        -- reset gradients
        gradParameters:zero()

        local distance = g.model:forward(batch.data)

        local f = g.criterion:forward(distance, batch.label)
        local df = g.criterion:backward(distance, batch.label)

        g.model:backward(batch.data, df)

        return f, gradParameters
    end

    local index = 1
    local maxIterate = torch.floor( g.allTrainSamples.data:size(1) / g.batchSize )
    for i = 1, maxIterate do
        batch, index = loadBatch(index)

        g.optim(feval, parameters, g.optimState)

        collectgarbage();
        xlua.progress(i, maxIterate)
    end
end
```

样本降维到2维时的分布



Triplet Network





From Siamese to Triplet Network

Elad Hoffer, etc. DEEP METRIC LEARNING USING TRIPLET NETWORK. Under review as a conference paper at ICLR 2015
http://arxiv.org/abs/1412.6622

- 应用类型
  - Image ranking

  - Face verification

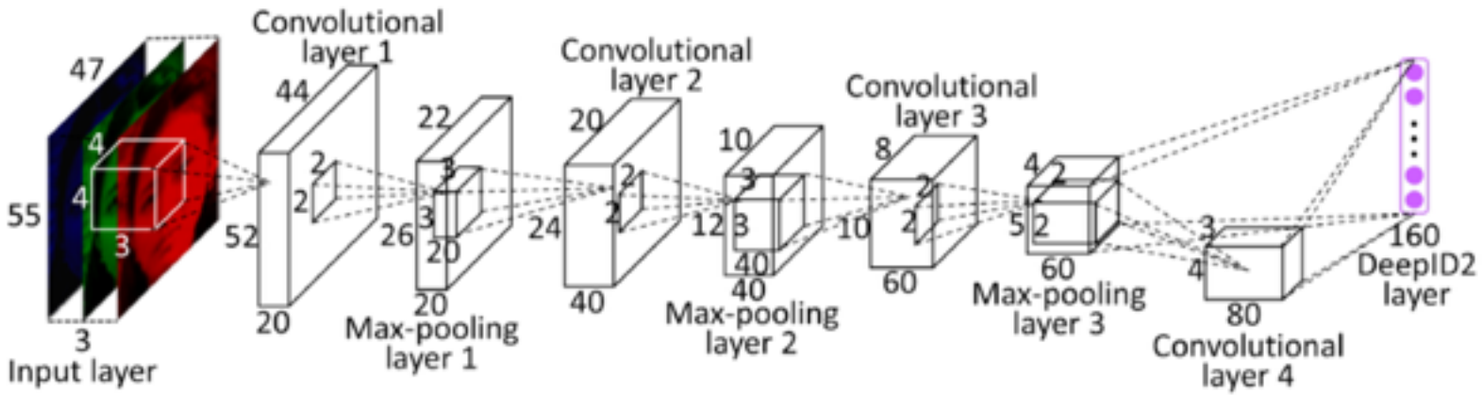
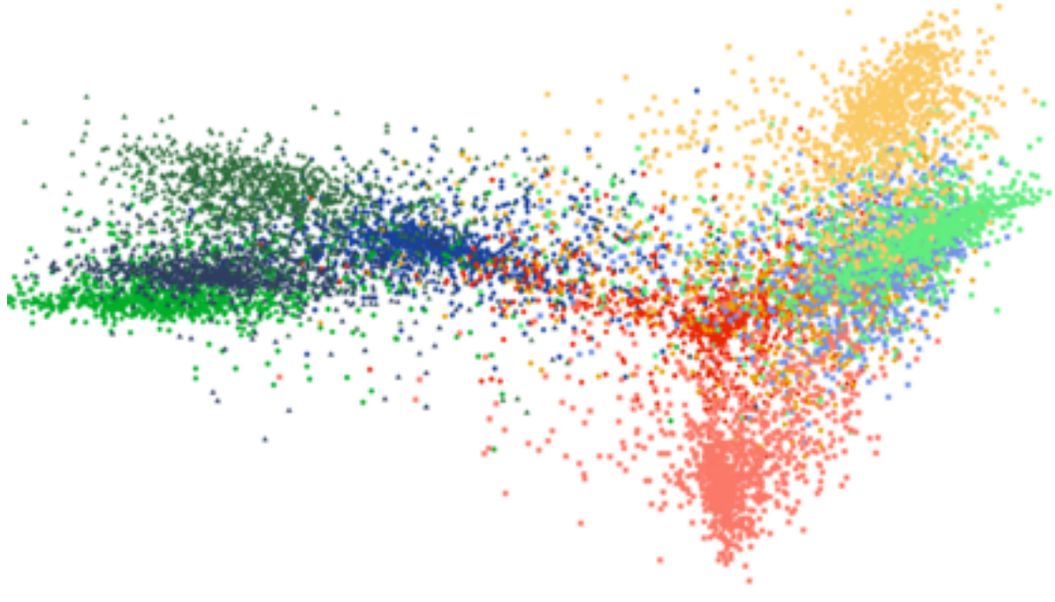


Figure 1: The ConvNet structure for DeepID2 extraction.

  - Metric learning



Variational Auto-encoder

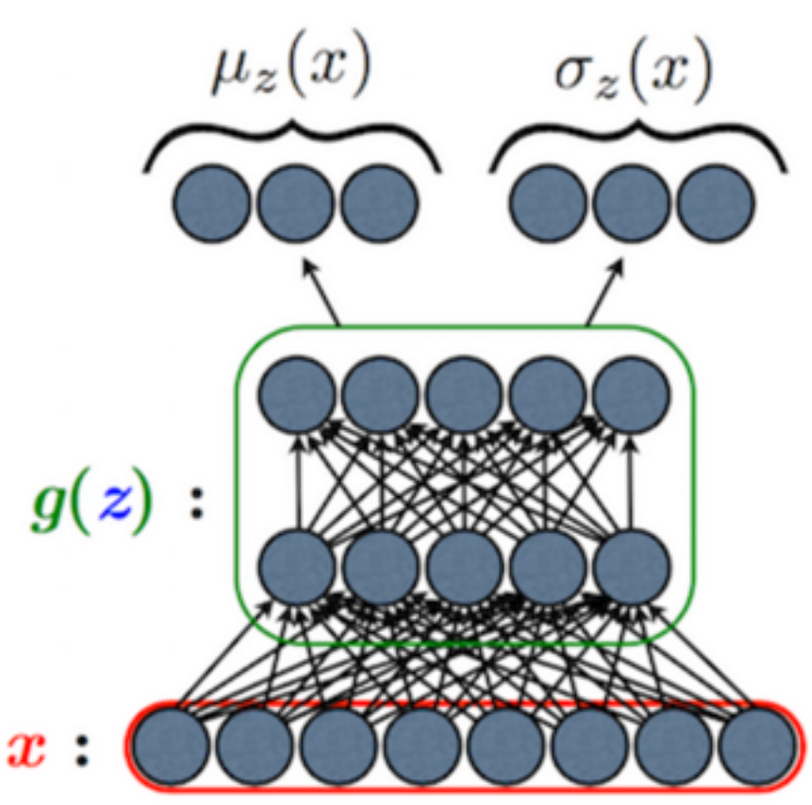
- 将Variational Bayes引入到神经网络中
- 网络和经典的Auto Encoder类似
  - 编码网络：将输入编码到一个参数（隐变量）分布
  - 解码网络：对分布的参数进行采样，拟合输入数据

首先引入隐变量z, 利用“编码网络”拟合参数化的后验概率 $q_\phi(z|x)$

这里的ϕ就是神经网络的权重，网络输出为隐变量z的条件分布参数（即后验概率），这里采用的是高斯分布：

$$q_\phi(z|x) = N(z; \mu_z(x), \sigma_z(x))$$

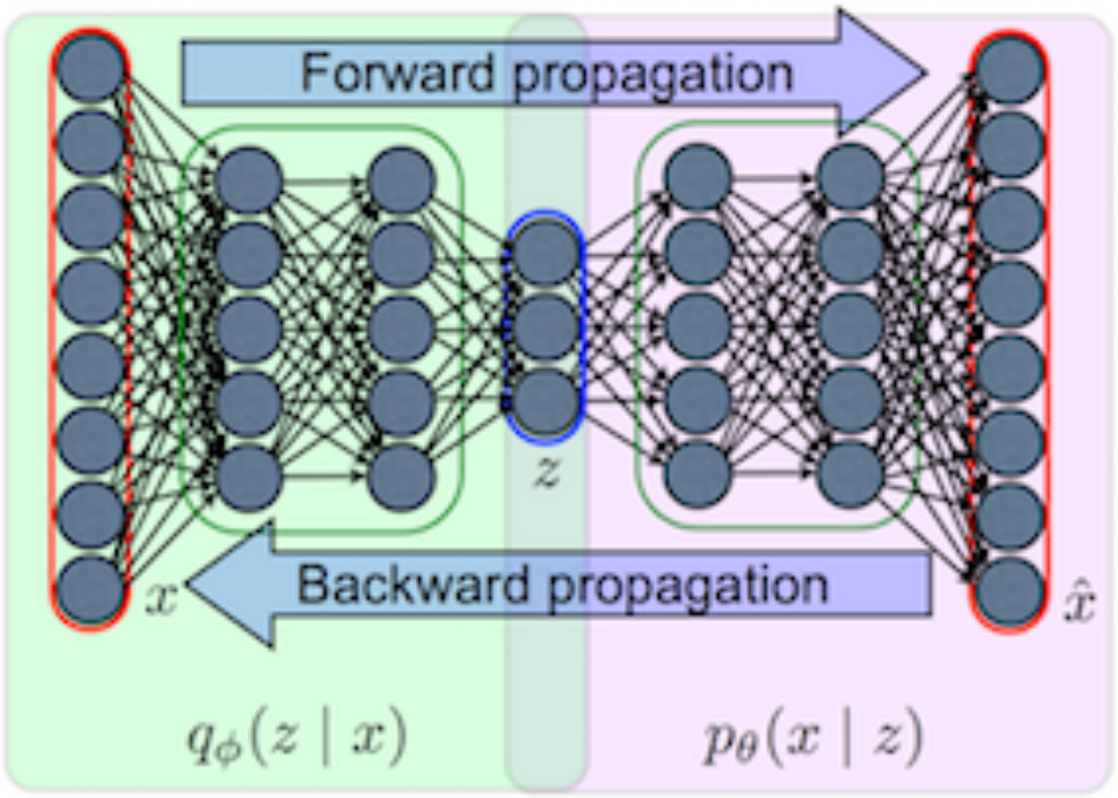
其中$\mu_z(x)$, $\sigma_z(x)$，就是Encoder网络的输出



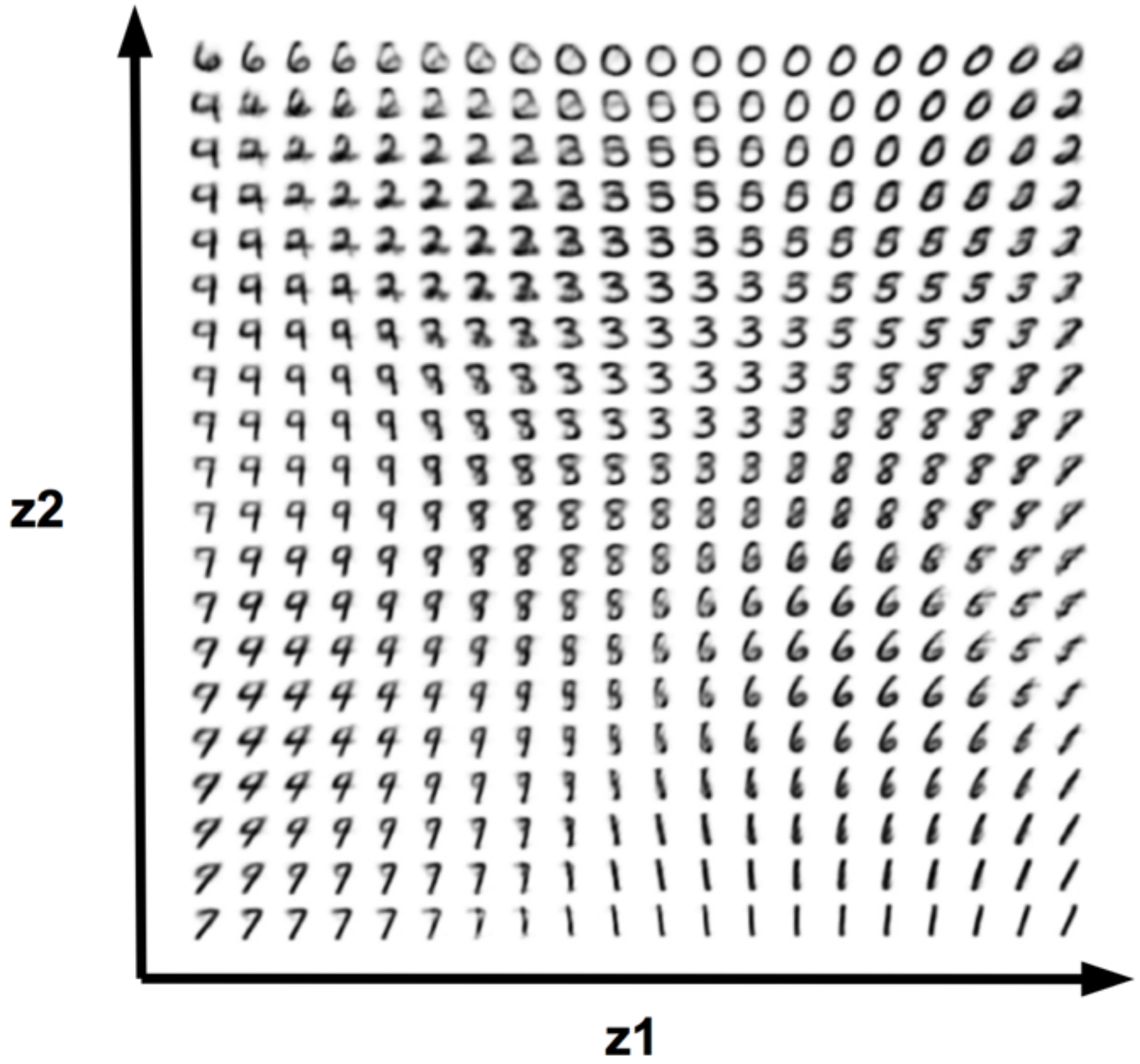
2. 解码网络和变分推断

解码网络是已知隐变量z，计算样本x的条件概率，这个概率是似然概率，参数为θ，即计算： $p_\theta(x|z)$

Objective function: $\mathcal{L}(\theta, \phi, x) = -D_{\mathrm{KL}}\left(q_\phi(z \mid x) \| p_\theta(z)\right) + \mathrm{E}_{q_\phi(z|x)}\left[\log p_\theta(x \mid z)\right]$



2D Latent space: MNIST



强化学习中的Policy network

- Markov decision processes
- Bellman equations
- Policy network & Policy Gradient Methods

Markov decision processes

- S：系统所处状态的集合
- A：系统可以做出的动作的集合
- P_{sa} 状态转移的概率，即在状态s下采取动作a之后，新状态的概率
- $\gamma \in [0, 1)$ 折现系数
- 奖励函数 $R : S \times A \mapsto \mathbb{R}$ $R : S \mapsto \mathbb{R}$

- 状态序列

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

- 某个过程的价值函数定义（重视当前的奖励，遥远未来的奖励全中逐步降低）

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots .$$ 与a有关

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots .$$ 与a无关

- 系统目标，找到一组动作，过程奖励期望最大

$$\mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots\right]$$

- 决定动作输出的策略（policy）定义为一个函数：

$$\pi : S \mapsto A$$

- 在某个策略下，过程的价值函数

$$V^{\pi}(s) = \mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \,\middle|\, s_0 = s, \pi\right].$$

- Bellman 公式：

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s').$$

$$\pi : S \mapsto A$$

- 最优价值函数可以定义为：

$$V^*(s) = \max_{\pi} V^{\pi}(s).$$

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s').$$

- 最优的决策函数，可以定义为：

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

注意最优决策函数，和初始状态s无关

$$V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s).$$

- 价值迭代，求解 $V^*(s)$

1. For each state s, initialize $V(s) := 0$.
2. Repeat until convergence {

   For every state, update $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s')V(s')$

   }

- 决策迭代，直接求解 $\pi^*(s)$

1. Initialize π randomly.
2. Repeat until convergence {

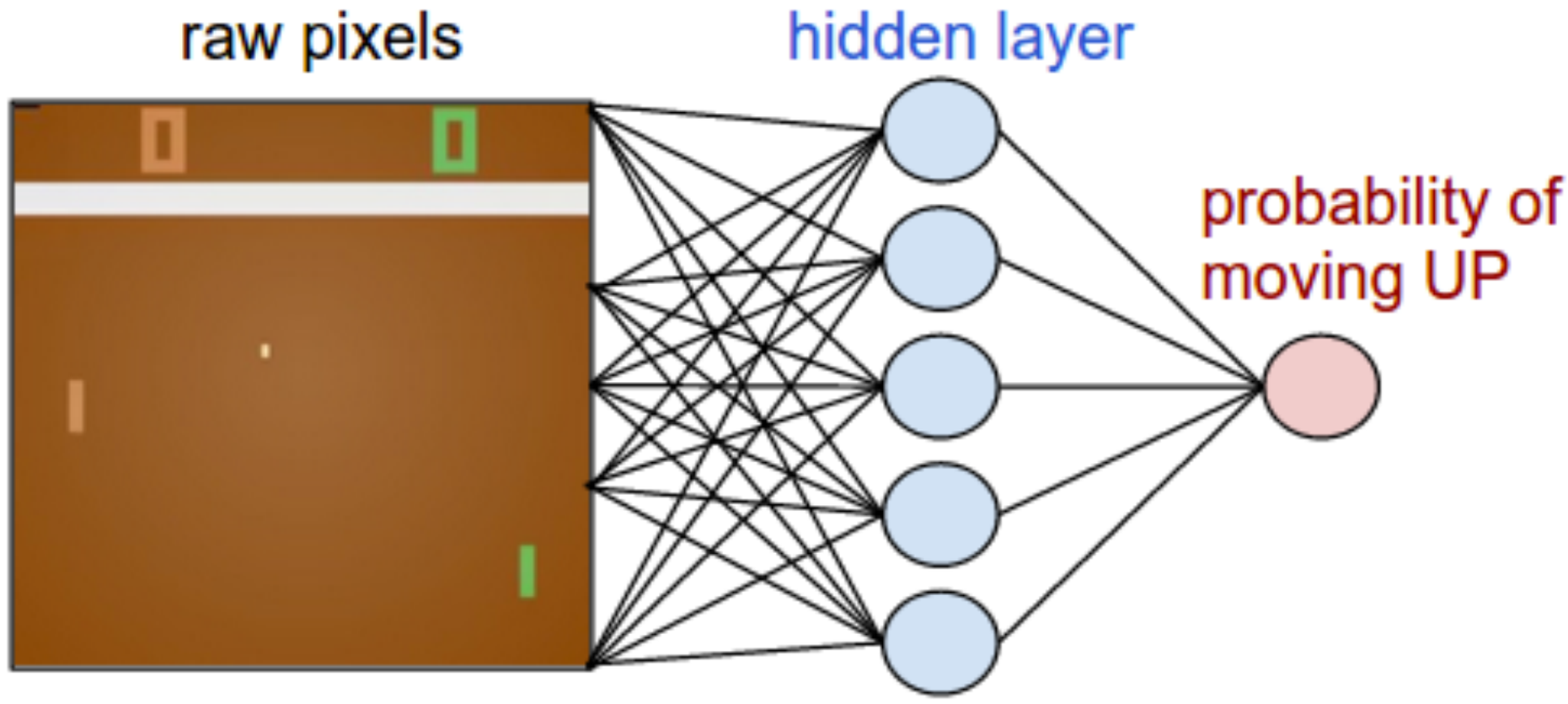
   (a) Let $V := V^{\pi}$.

   (b) For each state s, let $\pi(s) := \arg\max_{a \in A} \sum_{s'} P_{sa}(s')V(s')$

   }

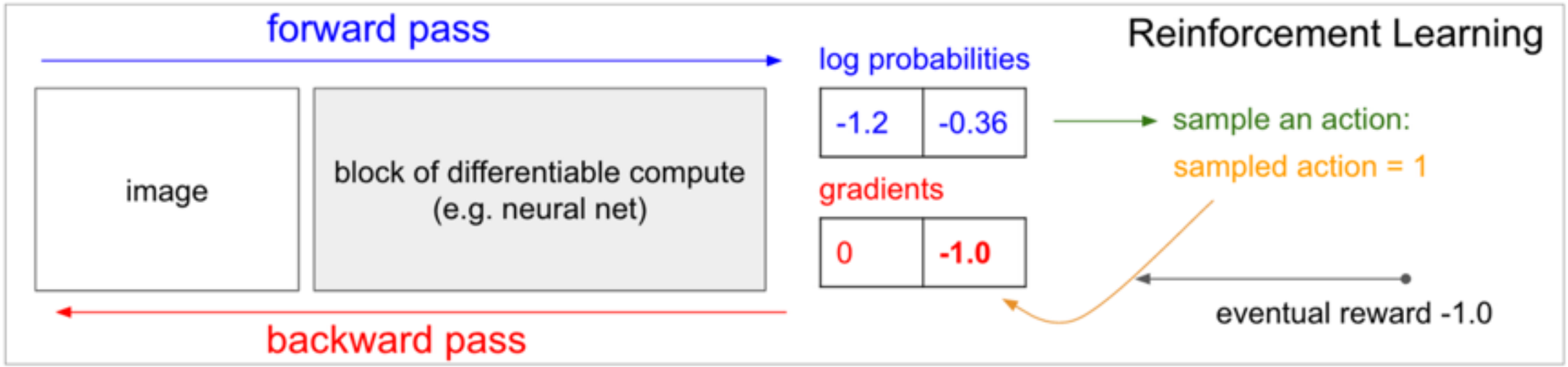
$|S| < \infty$, $|A| < \infty$

Policy network

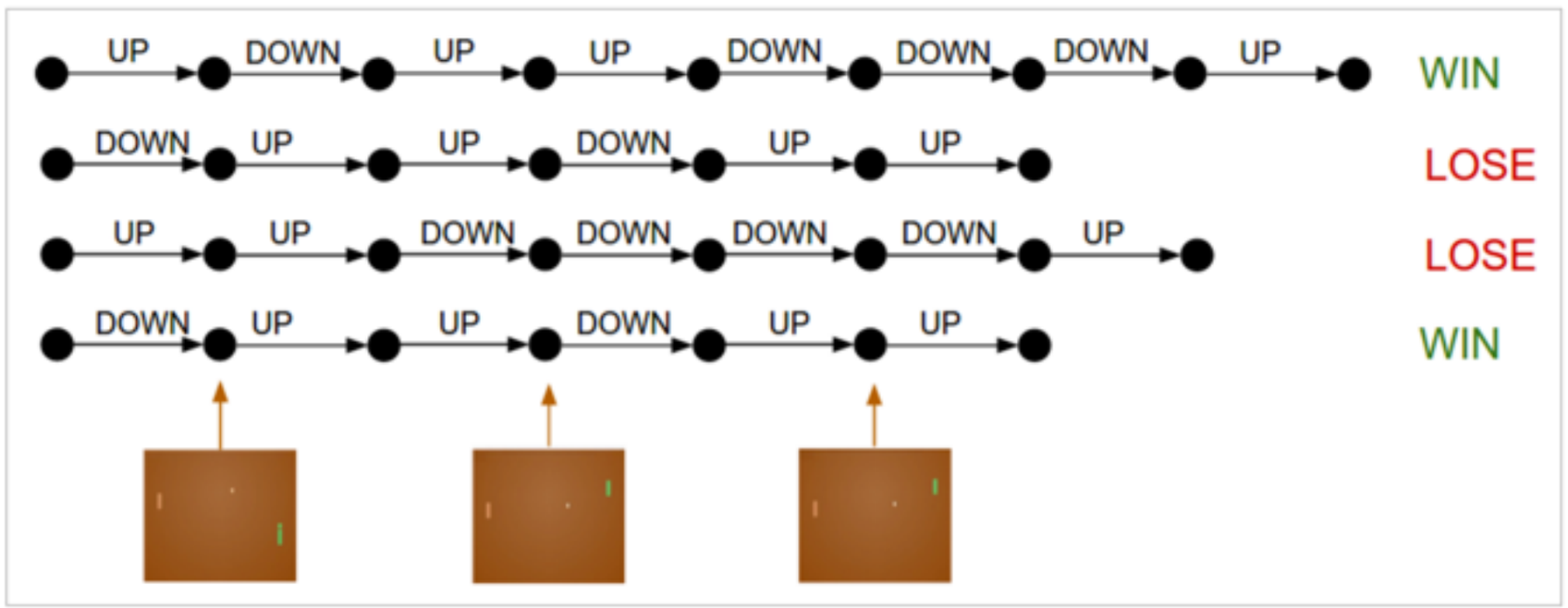


- 通过采样Policy Network，取得当前状态的输出

- Policy Gradient Methods



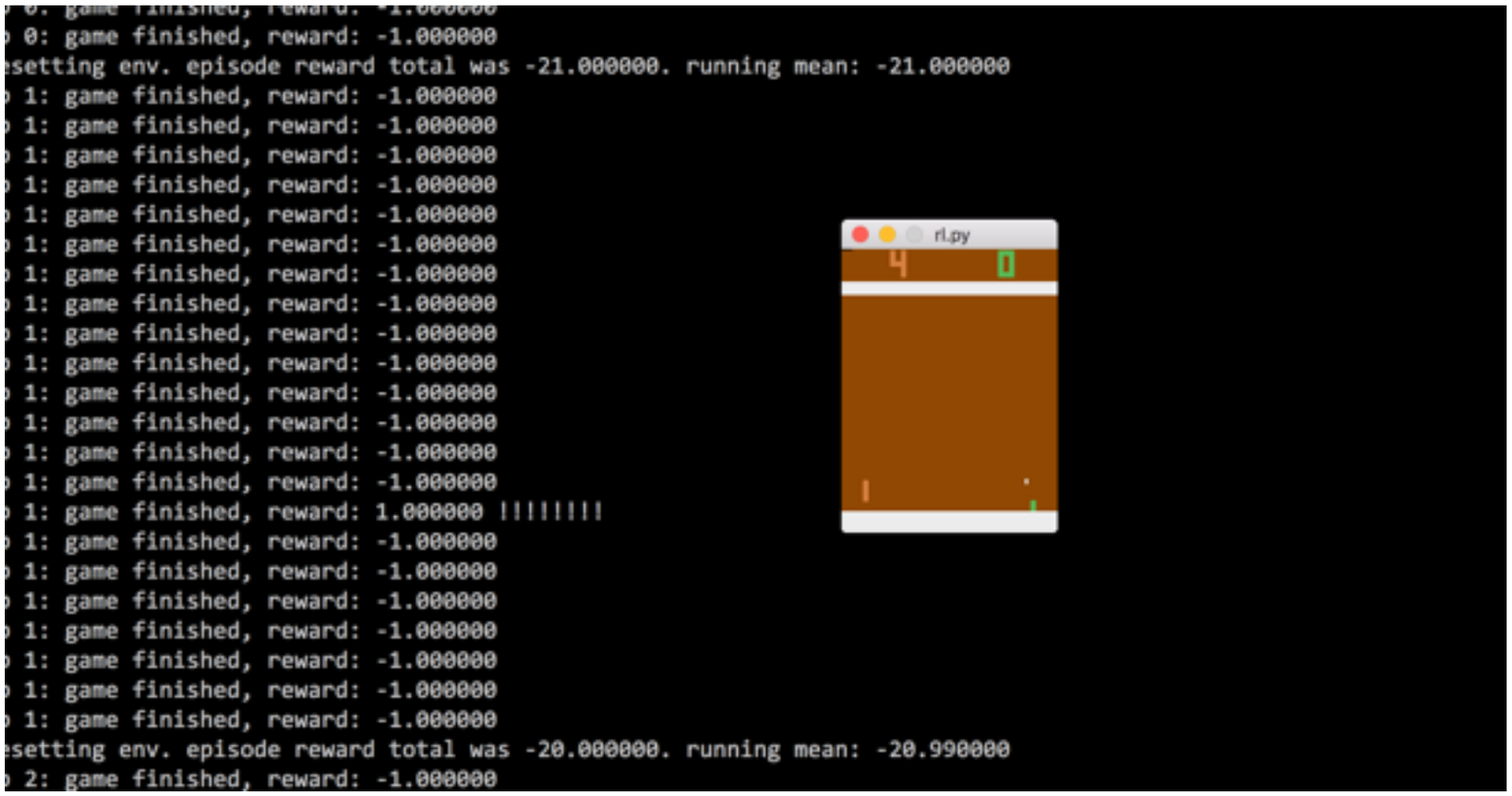
Policy Gradients: Run a policy for a while. See what actions led to high rewards. Increase their probability.



- 1. 随机初始化Policy network
- 2. 进行大量迭代 {
    - 固定network，执行多次网络，得到一系列结果。
    - 通过BP算法，增大好的结果概率，减少差的结果概率
- }

原理：通过多次采样平均梯度方向，逐步得需要的期望

- 实现一个简单的Policy Network
  - 使用OpenAI Gym工具



https://gym.openai.com/docs

http://karpathy.github.io/2016/05/31/rl/

1.准备模型，定义参数

```
""" Trains an agent with (stochastic) Policy Gradients on Pong. Uses OpenAI Gym. """
import numpy as np
import cPickle as pickle
import gym

# hyperparameters
H = 200 # number of hidden layer neurons
batch_size = 10 # every how many episodes to do a param update?
learning_rate = 1e-4
gamma = 0.99 # discount factor for reward
decay_rate = 0.99 # decay factor for RMSProp leaky sum of grad^2
resume = False # resume from previous checkpoint?
render = False

# model initialization
D = 80 * 80 # input dimensionality: 80x80 grid
if resume:
  model = pickle.load(open('save.p', 'rb'))
else:
  model = {}
  model['W1'] = np.random.randn(H,D) / np.sqrt(D) # "Xavier" initialization
  model['W2'] = np.random.randn(H) / np.sqrt(H)

grad_buffer = { k : np.zeros_like(v) for k,v in model.iteritems() } # update buffers that add up gradients over a batch
rmsprop_cache = { k : np.zeros_like(v) for k,v in model.iteritems() } # rmsprop memory
```

2.辅助函数

```
27  def sigmoid(x):
28    return 1.0 / (1.0 + np.exp(-x)) # sigmoid "squashing" function to interval [0,1]
29
30  def prepro(I):
31    """ prepro 210x160x3 uint8 frame into 6400 (80x80) 1D float vector """
32    I = I[35:195] # crop
33    I = I[::2,::2,0] # downsample by factor of 2
34    I[I == 144] = 0 # erase background (background type 1)
35    I[I == 109] = 0 # erase background (background type 2)
36    I[I != 0] = 1 # everything else (paddles, ball) just set to 1
37    return I.astype(np.float).ravel()
38
39  def discount_rewards(r):
40    """ take 1D float array of rewards and compute discounted reward """
41    discounted_r = np.zeros_like(r)
42    running_add = 0
43    for t in reversed(xrange(0, r.size)):
44      if r[t] != 0: running_add = 0 # reset the sum, since this was a game boundary (pong specific!)
45      running_add = running_add * gamma + r[t]
46      discounted_r[t] = running_add
47    return discounted_r
48
```

3.Forward/Backward函数

```
49  def policy_forward(x):
50    h = np.dot(model['W1'], x)
51    h[h<0] = 0 # ReLU nonlinearity
52    logp = np.dot(model['W2'], h)
53    p = sigmoid(logp)
54    return p, h # return probability of taking action 2, and hidden state
55
```

```
56  def policy_backward(eph, epdlogp):
57    """ backward pass. (eph is array of intermediate hidden states) """
58    dW2 = np.dot(eph.T, epdlogp).ravel()
59    dh = np.outer(epdlogp, model['W2'])
60    dh[eph <= 0] = 0 # backpro prelu
61    dW1 = np.dot(dh.T, epx)
62    return {'W1':dW1, 'W2':dW2}
```

4.训练（1）

```
env = gym.make("Pong-v0")
observation = env.reset()
prev_x = None # used in computing the difference frame
xs,hs,dlogps,drs = [],[],[],[]
running_reward = None
reward_sum = 0
episode_number = 0
while True:
  if render: env.render()

  # preprocess the observation, set input to network to be difference image
  cur_x = prepro(observation)
  x = cur_x - prev_x if prev_x is not None else np.zeros(D)
  prev_x = cur_x

  # forward the policy network and sample an action from the returned probability
  aprob, h = policy_forward(x)
  action = 2 if np.random.uniform() < aprob else 3 # roll the dice!
```

4.训练（2）

```
83    # record various intermediates (needed later for backprop)
84    xs.append(x) # observation
85    hs.append(h) # hidden state
86    y = 1 if action == 2 else 0 # a "fake label"
87    dlogps.append(y - aprob) # grad that encourages the action that was taken to be taken (see http://cs231n.gi
88
89    # step the environment and get new measurements
90    observation, reward, done, info = env.step(action)
91    reward_sum += reward
92
93    drs.append(reward) # record reward (has to be done after we call step() to get reward for previous action)
```

注意我们保存了每一步的x, h,以及reward，为bp计算准备

4.训练（3）

```
98      # stack together all inputs, hidden states, action gradients, and rewards for this episode
99      epx = np.vstack(xs)
100     eph = np.vstack(hs)
101     epdlogp = np.vstack(dlogps)
102     epr = np.vstack(drs)
103     xs,hs,dlogps,drs = [],[],[],[] # reset array memory
104
105     # compute the discounted reward backwards through time
106     discounted_epr = discount_rewards(epr)
107     # standardize the rewards to be unit normal (helps control the gradient estimator variance)
108     discounted_epr -= np.mean(discounted_epr)
109     discounted_epr /= np.std(discounted_epr)
110
111     epdlogp *= discounted_epr # modulate the gradient with advantage (PG magic happens right here.)
112     grad = policy_backward(eph, epdlogp)
113     for k in model: grad_buffer[k] += grad[k] # accumulate grad over batch
```

注意一次游戏，有许多轮，每一轮才是一个完整的序列

4.训练（4）

```
114
115     # perform rmsprop parameter update every batch_size episodes
116     if episode_number % batch_size == 0:
117       for k,v in model.iteritems():
118         g = grad_buffer[k] # gradient
119         rmsprop_cache[k] = decay_rate * rmsprop_cache[k] + (1 - decay_rate) * g**2
120         model[k] += learning_rate * g / (np.sqrt(rmsprop_cache[k]) + 1e-5)
121         grad_buffer[k] = np.zeros_like(v) # reset batch gradient buffer
122
123     # boring book-keeping
124       running_reward = reward_sum if running_reward is None else running_reward * 0.99 + reward_sum * 0.01
125       print 'resetting env. episode reward total was %f. running mean: %f' % (reward_sum, running_reward)
126       if episode_number % 100 == 0: pickle.dump(model, open('save.p', 'wb'))
127       reward_sum = 0
128       observation = env.reset() # reset env
129       prev_x = None
130
131   if reward != 0: # Pong has either +1 or -1 reward exactly when game ends.
132     print ('ep %d: game finished, reward: %f' % (episode_number, reward)) + ('' if reward == -1 else ' !!!!!!!!')
```

- 实际Demo（视频演示）