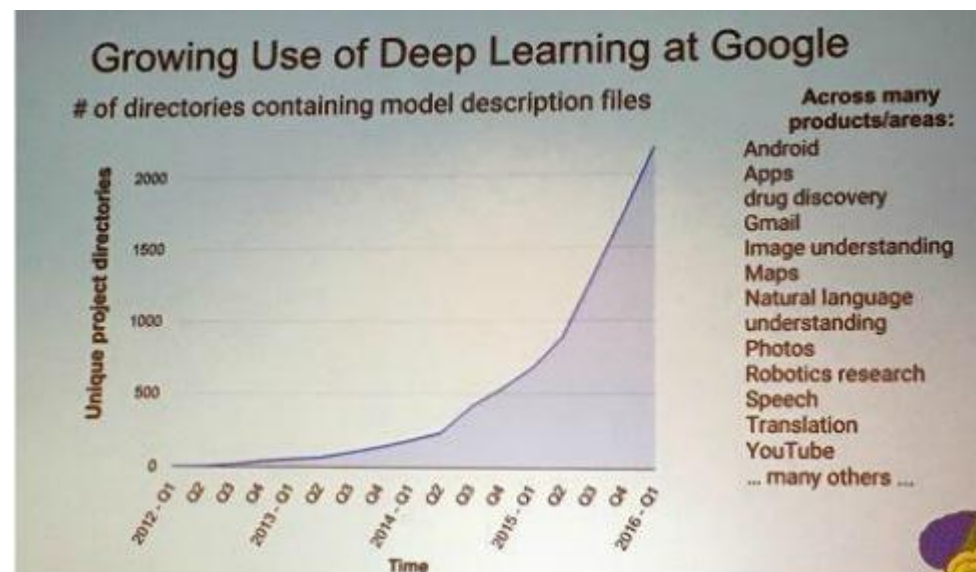


Deep Learning Tutorial

人工智能最火的领域：深度学习



Deep learning trends
at Google. Source:
SIGMOD/Jeff Dean

目录

1. 深度学习简介
2. 深度学习的训练
3. 深度学习常用的几种模型
4. 强化学习 (RL)

1 深度学习简介

深度学习 “三步走”

(建模-误差-优化)

设计神经网络模型

$$f(f(f(\bullet)))$$

评价网络的性能

Loss

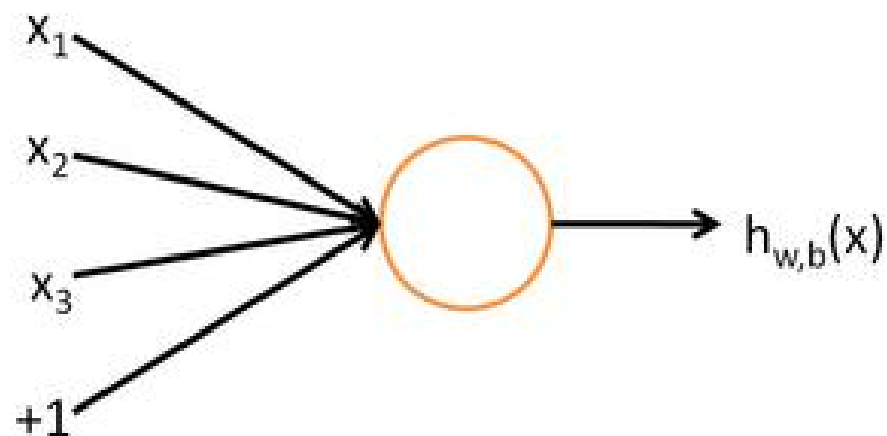
选择最好的模型

BP

神经元

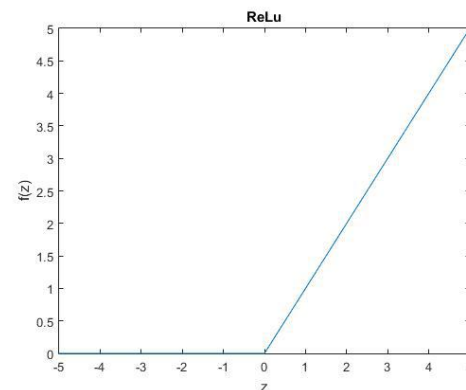
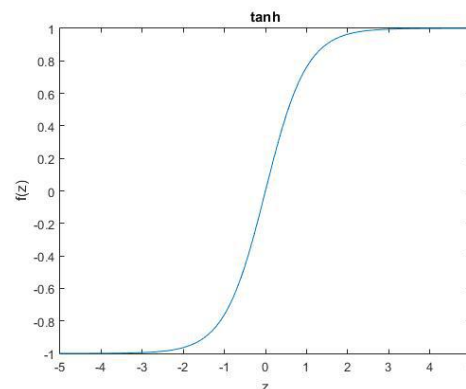
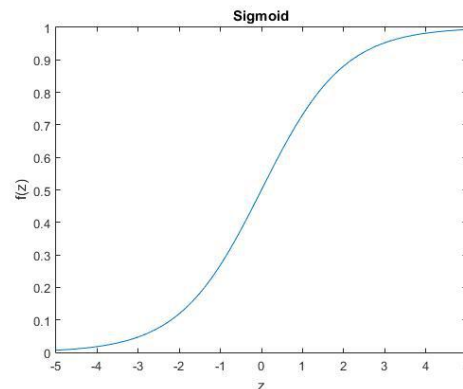
- 假设我们有训练样本集，神经网络算法能够提供一种复杂且非线性的假设模型 $h_{W,b}(x)$ ，参数是 W, b
- 非线性激活函数： $f: \mathbb{R} \rightarrow \mathbb{R}$

$$h_{W,b}(x) = f(Wx + b)$$



常用的激活函数

- Sigmoid : $f(z) = \frac{1}{1 + e^{-z}}$
- Tanh : $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- ReLu : $f(z) = \begin{cases} z & z > 0 \\ 0 & z \leq 0 \end{cases}$

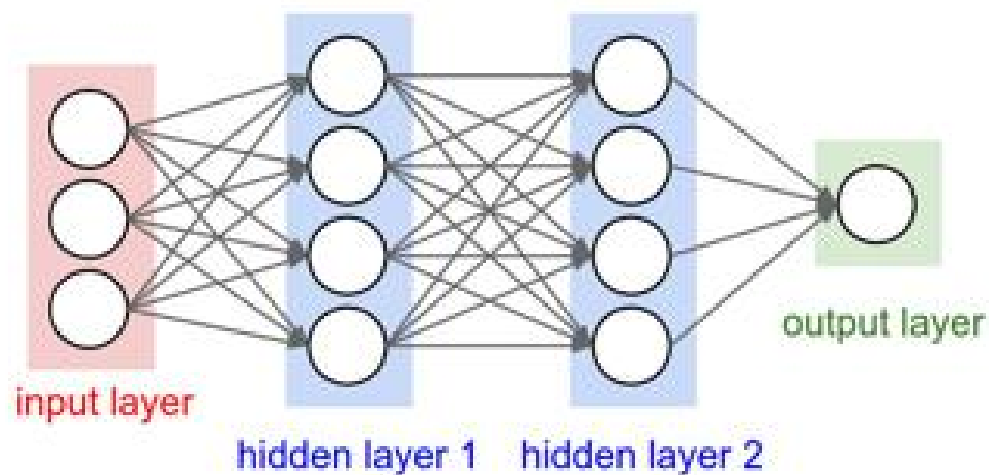


神经网络模型

- 神经网络就是将许多单一的神经元**连接**在一起，一个神经元的输出就可以作为另一个神经元的输入

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$$

$$a^{(l+1)} = f(z^{(l+1)})$$



- 以上计算步骤叫做**前向传播**

神经网络模型

- 多层神经网络模型可以理解为
多个非线性函数“嵌套”

构成足够复杂的模型：

$$f(\cdots f(f(f(z, w_1, b_1), w_2, b_2) \cdots), w_n, b_n)$$

- 由于层数理论上可以无限叠加，
所以神经网络具有**无限建模**能力，
可以**拟合**任意函数。

- “机器学习 \approx 寻找一个函数”
(李宏毅教授，台湾)

- Speech Recognition

$$f(\text{[audio waveform]}) = \text{"How are you"}$$

- Image Recognition

$$f(\text{[cat image]}) = \text{"Cat"}$$

- Playing Go

$$f(\text{[Go board image]}) = \text{"5-5"}_{\text{(next move)}}$$

- Dialogue System

$$f(\text{"Hi"}_{\text{(what the user said)}}) = \text{"Hello"}_{\text{(system response)}}$$

深度学习 “三步走”

(建模-误差-优化)

设计神经网络模型

$$f(f(f(\bullet)))$$

评价网络的性能





Loss



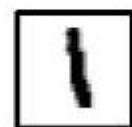

选择最好的模型

BP

训练集




- 训练集包括两部分：样本 (sample) 以及 标签 (label)
 - 注释：回归任务中样本本身就是标签

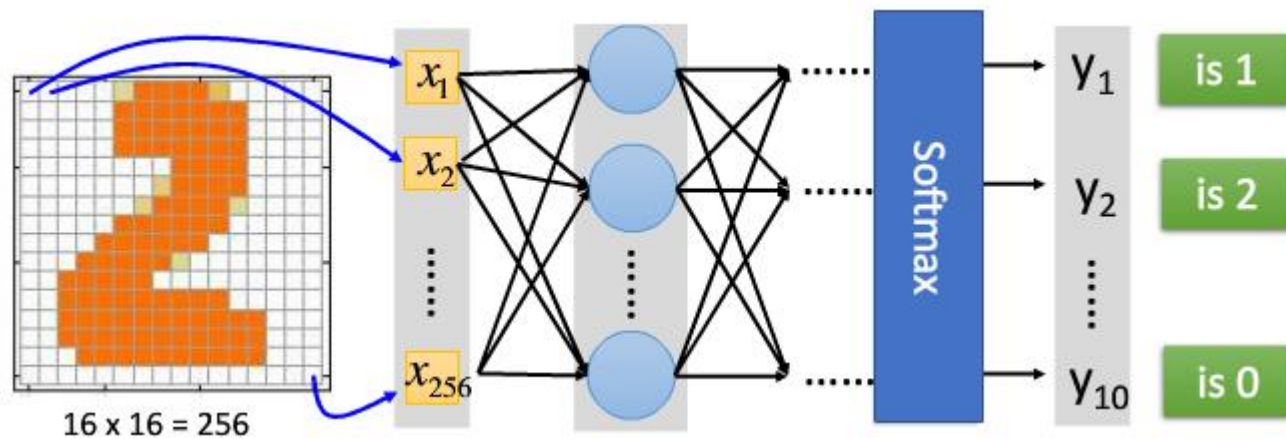
 "5"  "0"  "4"  "1"

 "9"  "2"  "1"  "3"

- 独热编码 (One-Hot Encoding)
 - 1:(1,0,0,0,0,0,0,0,0,0); 2:(0,1,0,0,0,0,0,0,0,0);..... ;
 - 0:(0,0,0,0,0,0,0,0,0,1)

学习目标

- 输入  , y_1 最大
- 输入  , y_3 最大
- 输入  , y_{10} 最大



损失（ Loss ）函数

- Loss：网络输出与目标（期望输出）的距离函数
- 常见的损失函数
 - 回归问题：均方误差函数 MSE $loss = (y - a)^2$
 - 分类问题：交叉熵 Cross Entropy $loss = y \ln a + (1 - a) \ln y$
 - log-likelihood cost: 深度学习中更普遍的做法是将 softmax 作为最后一层，此时常用代价函数是 log-likelihood cost.
- 一个好的模型应该使所有样本的误差尽可能小

Total loss:
$$L = \sum_{i=1}^N loss_i$$

深度学习 “三步走”

(建模-误差-优化)

设计神经网络模型

$$f(f(f(\bullet)))$$

评价网络的性能

Loss

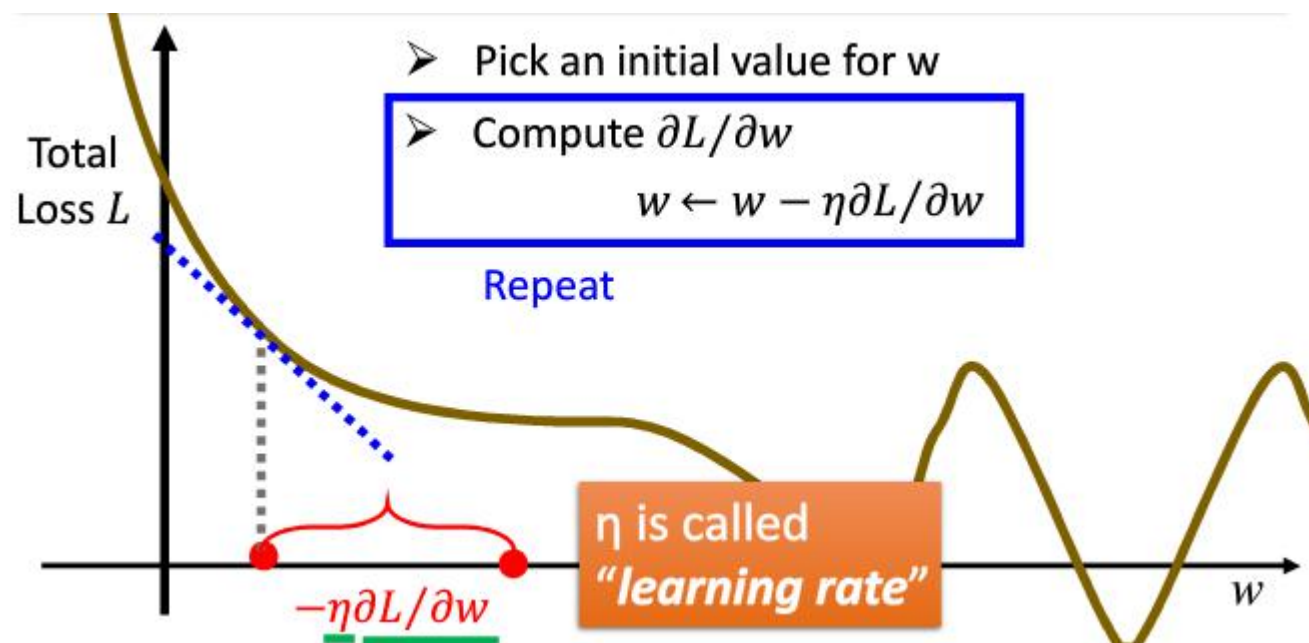
选择最好的模型

BP

梯度下降法

- 通过遍历的方法选取网络最优的权值显然非常愚蠢
- 往往采用最基本的优化方法：梯度下降 (Gradient Descent)

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$



图片来自
于李宏毅

反向传播 (BP)

- 反向传播：一种计算 $\frac{\partial L}{\partial w}$ 的有效方式

参考UFLDL：http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm

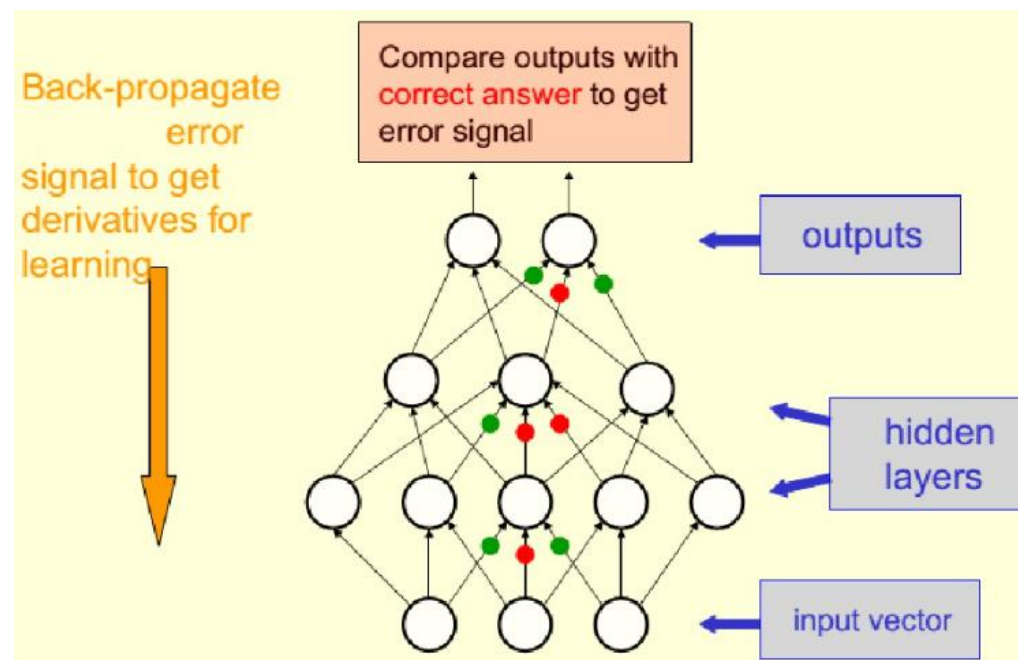
- 手动计算 $\frac{\partial L}{\partial w}$ 是相当麻烦的

(有兴趣的同学可以推导 CNN和RNN的梯度)

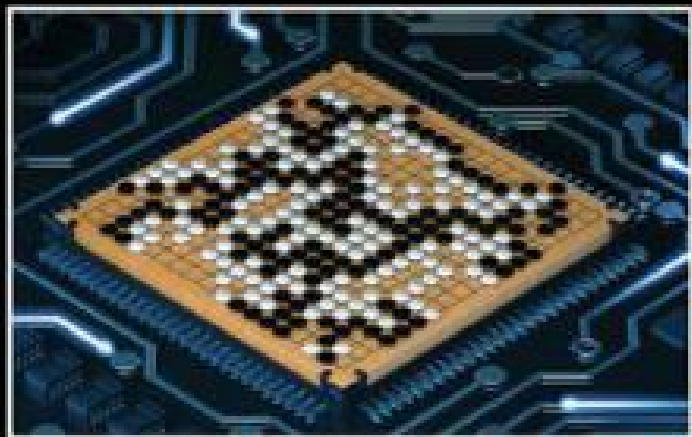
以下开源工具解决了这个麻烦：

- Tensorflow , Pytorch, Theano, Caffe.....
- Keras, TensorLayer, TFLearn.....

- So.....



Deep Learning研究生



朋友覺得我在



我媽覺得我在



大眾覺得我在



指導教授覺得我在



我以為我在



事實上我在

图片来自网络

From NN to Deep Learning

- Why Deep?

- 更多的参数 → 更复杂的模型 → 更强的拟合能力；

- 类比逻辑电路：

- 两层逻辑闸可以表示任何布尔函数，
实际应用中不会这么做
因为使用很多层更高效；

- Fat + short or thin + tall?

- 并不是参数越多表现越好(过拟合)

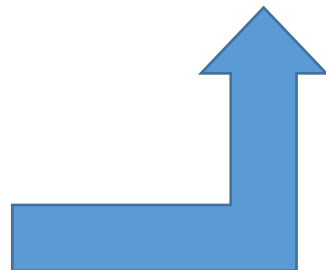
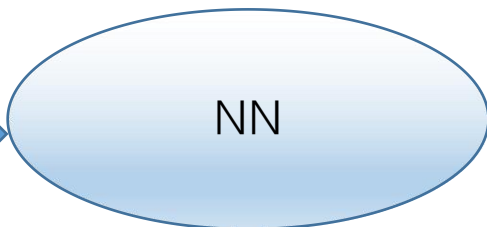
Layer X Size	Word Error Rate (%)
1 X 2k	24.2
2 X 2k	20.4
3 X 2k	18.4
4 X 2k	17.8
5 X 2k	17.2
7 X 2k	17.1

Layer X Size	Word Error Rate (%)	Layer X Size	Word Error Rate (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2		
7 X 2k	17.1		
		1 X 3772	22.5
		1 X 4634	22.6
		1 X 16k	22.1

Why?

2 深度学习的训练

一般流程



足够小的训练误差？

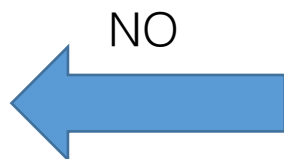


YES

足够小的测试误差？

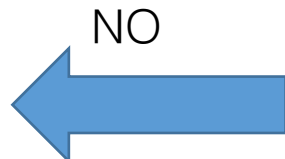


YES

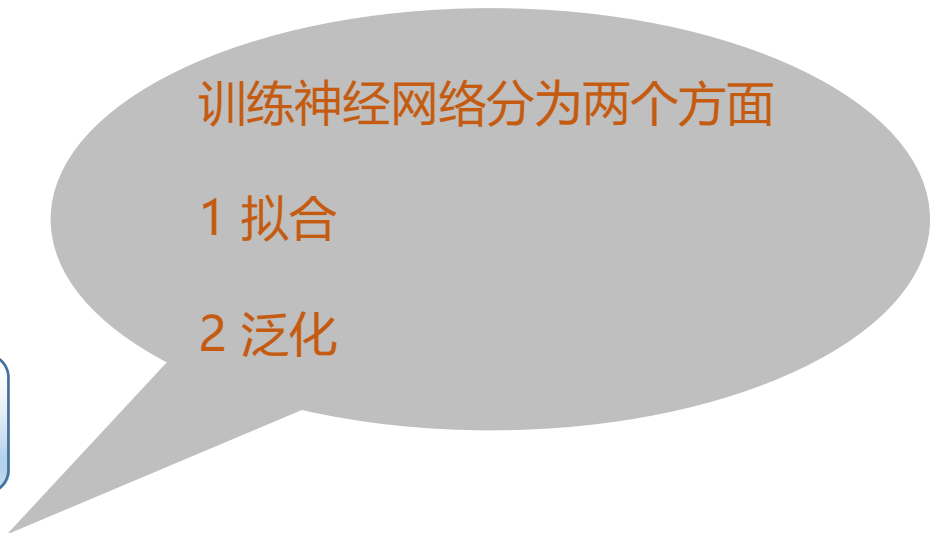


NO

过拟合！！！！



NO



拟合

- 合适的损失函数
 - 分类：交叉熵
 - 回归：均方误差
- 合适的初始权重
 - 用小的随机数初始化权重

拟合

- Mini-batch

- 普通的梯度下降算法在更新回归系数时要遍历整个数据集，是一种批 (batch) 处理方法，这样训练数据特别庞大时，可能出现如下问题：
 1. 收敛过程可能非常慢；
 2. 如果误差曲面上有多个局极小值，那么不能保证这个过程会找到全局最小值。
- 为了解决上面的问题，实际中我们应用的是梯度下降的一种变体被称为随机梯度下降 (SGD) 每次迭代随机取所有数据的一部分 (Mini-batch) 进行梯度下降训练
- 理论和实践证明：Mini-batch is Faster and Better

拟合

- 合适的激活函数

- Sigmoid 的问题：梯度消失

$$Grad = Error \cdot Sigmoid'(x) \cdot x$$

1. $Sigmoid'(x) \in (0,1)$ 导数缩放
2. $x \in (0,1)$ 或 $x \in (-1,1)$ 饱和值缩放

这样，经过每一层时，Error都是成倍的衰减，一旦进行递推式的多层的反向传播，梯度就会不停的衰减，消失，使得网络学习变慢。

- Relu :

- 单侧抑制
 - 相对宽阔的兴奋边界
 - 稀疏激活性
 - 与Sigmoid不同，Relu导数为1，避免了梯度消失

拟合

- 合适的损失函数
 - 分类：交叉熵
 - 回归：均方误差
- 合适的初始权重
 - 用小的随机数初始化权重

拟合

- 随机梯度下降算法中，学习率是深度网络难以设置的超参数之一，因为它对模型的性能有显著的影响。
- 动量 (Momentum)算法可以缓解这些问题，但这样做的代价是引入了另一个超参，而且同样对所有的参数都适用相同的调整策略

$$\begin{aligned}\hat{g} &\leftarrow + \frac{1}{m} \nabla_{\theta} \sum_i L(f(x_i; \theta), y_i) \\ v &\leftarrow \alpha v - \epsilon \hat{g} \\ \theta &\leftarrow \theta + v\end{aligned}$$

- 自适应的学习速率

AdaGrad	$\epsilon_n = \frac{\epsilon}{\delta + \sqrt{\sum_{i=1}^{n-1} g_i \odot g_i}}$
RMSProp	$\begin{aligned}\hat{g} &\leftarrow + \frac{1}{m} \nabla_{\theta} \sum_i L(f(x_i; \theta), y_i) \\ r &\leftarrow \rho r + (1 - \rho) \hat{g} \odot \hat{g} \\ \Delta \theta &= - \frac{\epsilon}{\delta + \sqrt{r}} \odot \hat{g} \\ \theta &\leftarrow \theta + \Delta \theta\end{aligned}$
Adam	$\begin{aligned}g &\leftarrow + \frac{1}{m} \nabla_{\theta} \sum_i L(f(x_i; \theta), y_i) \\ s &\leftarrow \rho_1 s + (1 - \rho_1) g \\ r &\leftarrow \rho_2 r + (1 - \rho_2) g \odot g \\ \hat{s} &\leftarrow \frac{s}{1 - \rho_1} \\ \hat{r} &\leftarrow \frac{r}{1 - \rho_2} \\ \Delta \theta &= -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta} \\ \theta &\leftarrow \theta + \Delta \theta\end{aligned}$

泛化

- 正则化 (Regularization)

- L2 regularization (权重衰减 **weight decay**)

- L2正则化就是在代价函数后面再加上一个L2正则化项 (对权重的罚):

$$L = L_0 + \lambda \|w\|_2$$

- 参考岭回归 (**ridge regression**)

- L1正则化就是原始的代价函数后面加上一个L1正则化项, 即所有权重 w 的绝对值的和, 乘以 λ/n

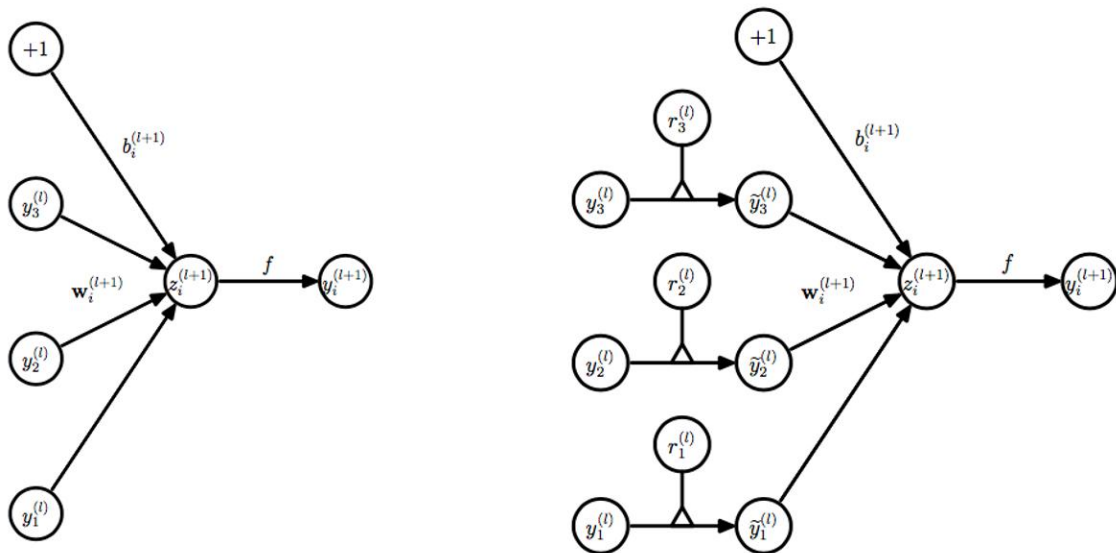
$$L = L_0 + \lambda \|w\|_1$$

- 参考套索 (lasso)

泛化

- Dropout

- 每次迭代，**随机**地以一定概率“删除”的隐层单元,保持输入输出层不变，按照BP算法更新神经网络中的权值，直至训练结束；
- Dropout 本质是一种广义上的 **bagging** 集成方法



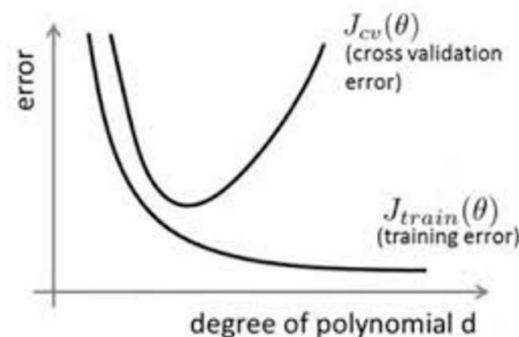
泛化

- 早停 (early stopping)

- 方差-偏倚权衡

- 深度网络训练有足够的表示能力甚至出现过拟合时，我们经常观察到，训练误差会随着时间的推移但验证集的误差会再次上升，这种现象几乎在深度网络的训练中必然出现

- 将原始数据集分为三部分：training data、validation data，testing data
 - validation data是用来避免过拟合的，在训练过程中，我们通常用它来确定一些超参数
 - 如果在testing data进行早停，那么随着训练的进行，我们的网络实际上就是在一点一点地overfitting我们的testing data



3 深度学习常用的几种模型

卷积神经网络CNN

- 卷积神经网络和标准神经网络非常相似：
 - 它们都是由神经元组成，神经元参数有权重 w 和偏差 b
 - 不同的是 权重 w 与 输入 x 进行的是卷积运算*
$$y = f(w * x + b)$$
- 卷积层的每一个特征map是不同卷积在前一层所有map上作卷积并将对应元素累加后加一个偏置
- 假设输入大小是 $n*n$, 卷积核大小是 $k*k$, 则该层输出为 $(n-k+1) * (n-k+1)$

1 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	0	0
0 <small>x0</small>	1 <small>x1</small>	1 <small>x0</small>	1	0
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

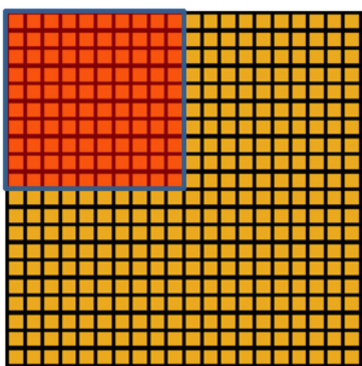
4		

Convolved
Feature

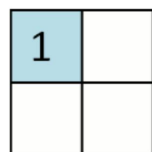
$n=5, k=3$

CNN的Pooling过程

- 对某一个区域上的特征取平均或最大值得操作叫做池化 (pooling)
- 池化单元具有平移不变性 (translation invariant)



Convolved
feature



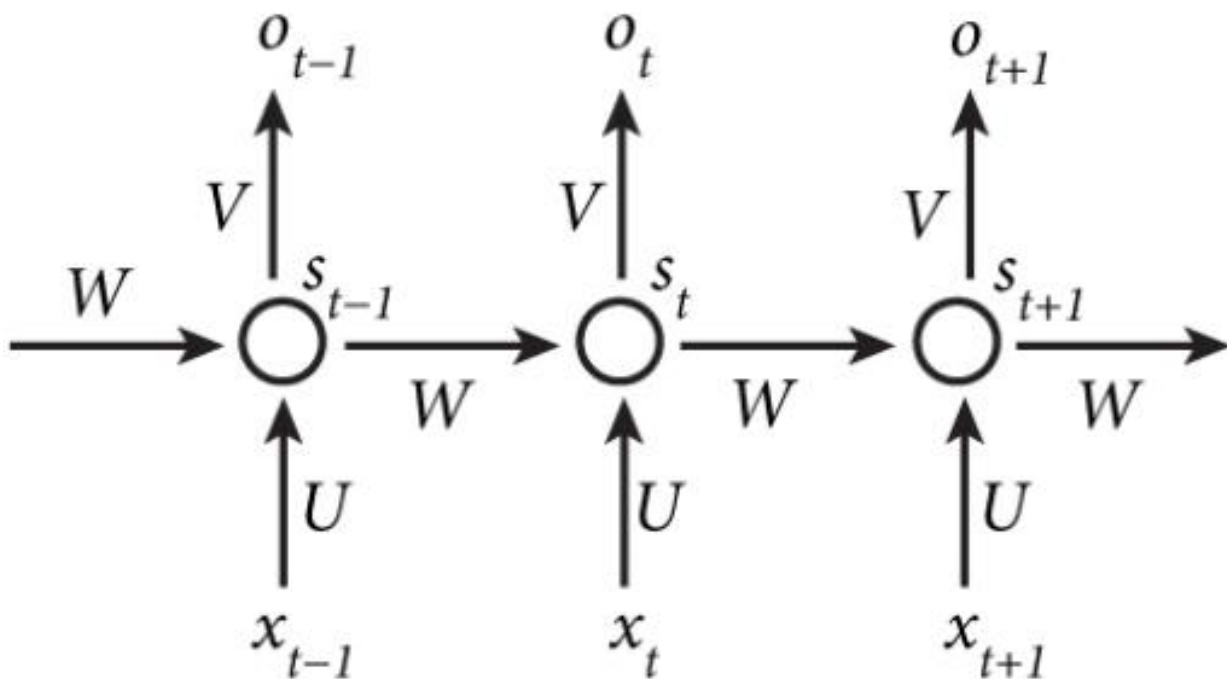
Pooled
feature

$n \times n = 10 \times 10$

Why CNN works?

- 以图像处理为例
 - 在图像处理中，往往把图像表示为像素的向量，比如一个 1000×1000 的图像，可以表示为一个1000000的向量。
 - 如果使用全连接神经网络中，如果隐含层数目与输入层一样，即也是1000000时，那么输入层到隐含层的参数数据为 $1000000 \times 1000000 = 10^{12}$ ，这是很难训练的
 - 假如每个神经元只和 10×10 个像素值相连，那么权值数据为 1000000×100 个参数，减少为原来的千分之一。而那 10×10 个像素值对应的 10×10 个参数，其实就相当于卷积操作。——权值共享

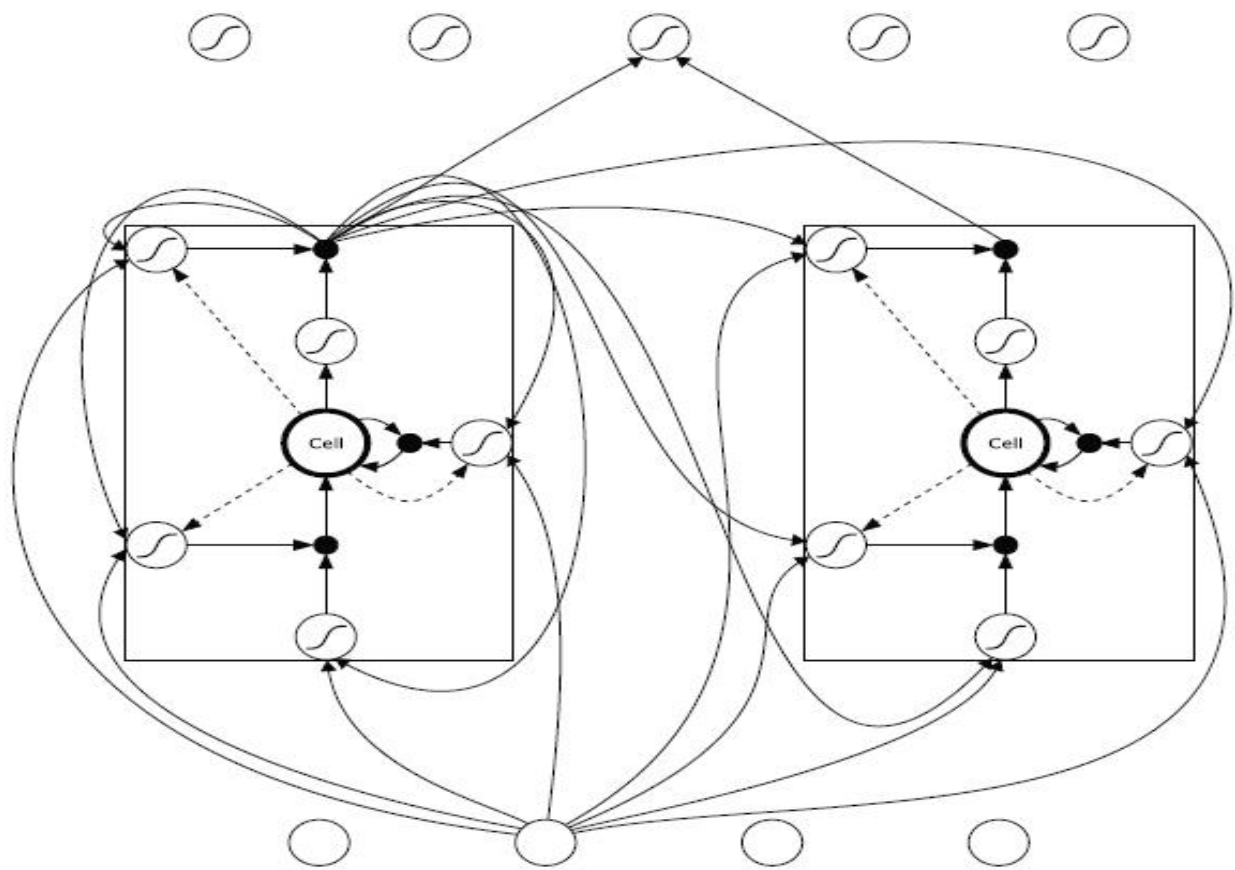
Recurrent neural network



x_t 可以看成是第 t 层的输入；
 s_t 是第 t 层的隐藏状态，负责整个神经网络的记忆功能；
 $s_t = f(Ux_t + Ws_{t-1})$, f 通常是个非线性函数；
 o_t 是第 t 层输出；
 $o_t = \text{softmax}(Vs_t)$;
值得注意的是没一层的参数 U , W , V 都是共享的；参数空间相对于 CNN 等网络要小很多；
在理论上 s_t 可以逼近前面每一层发生的事情，但是实际中依赖的长度很难训练

Three kinds of gates of LSTM

1、选择记忆



An LSTM network

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$



2、状态更新

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$



3、生存新状态

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

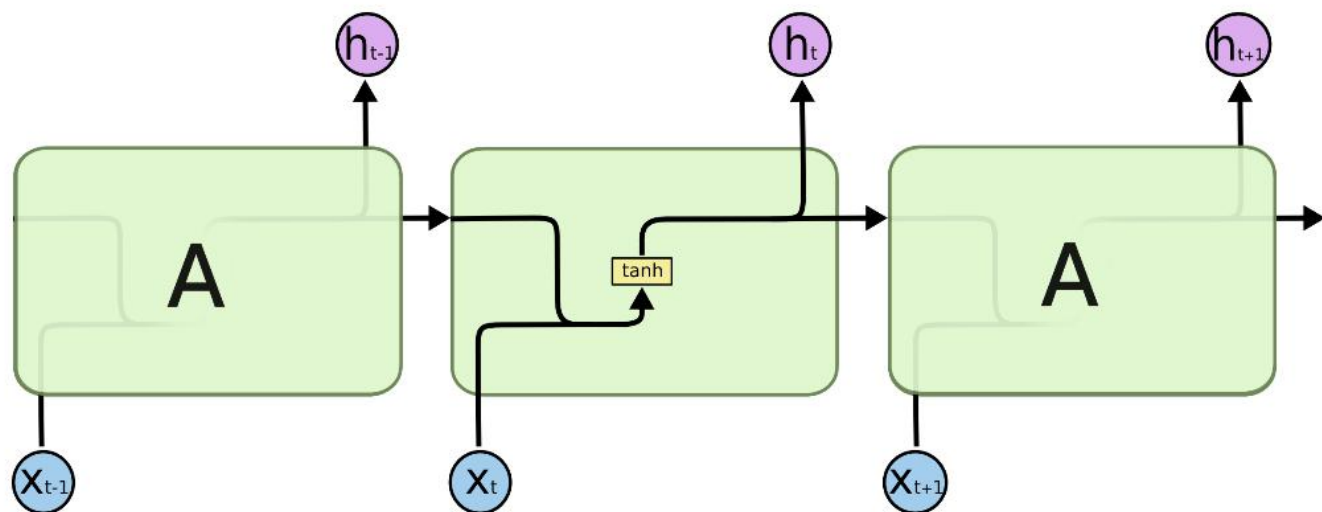


4、模块循环输出

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

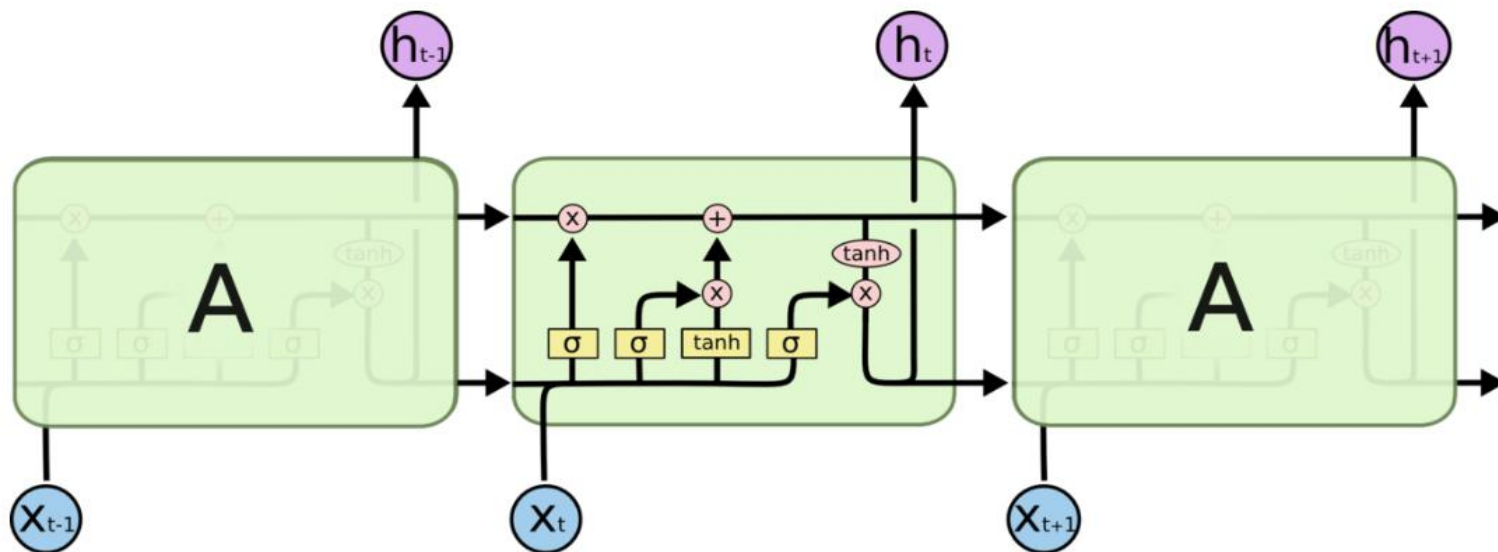
Long Short-Term Memory



RNN *v.s.* LSTM

传统RNN每一步的隐藏单元只是执行一个简单的tanh或ReLU操作；

LSTM每个循环的模块内又有4层结构:3个sigmoid层, 1个tanh层



4 强化学习

Reinforcement Learning

- 定义和符号

Environment : $E = \langle X, A, P, R \rangle$

Action : a

Action space: $A, a \in A$

State: x

State space: $X, x \in X$

Reward: $R: X \times A \times X \rightarrow \mathbb{R}$

Policy: $\pi, a = \pi(x)$

Transition probability : $P: X \times A \times X \rightarrow \mathbb{R}$

State value function: $V(?)$

State-action value function: $Q(?)$

$Q^\pi(x, a)$ 表示从状态 x 出发，执行动作 a 后再使用策略 π 带来的累计奖赏

Reinforcement Learning

- Exploration-Exploitation dilemma（尝试次数是有限的）
- Exploration-only: 将所有尝试机会平均分配给每个动作，可以很好地估计每个动作的奖赏，但失去很多采取最优动作的机会；
- Exploitation-only：按目前最优（截止目前为止平均奖赏最大）的动作，没有很好地估计奖赏，很可能错过最优动作。

Q Learning

- 算法的核心步骤

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{r_{t+1} + \underbrace{\gamma \cdot \max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

where r_{t+1} is the reward observed after performing a_t in s_t , and where $\alpha_t(s, a)$ ($0 < \alpha \leq 1$) is the learning rate (may be the same for all pairs).

Q Learning

- 简单解释一下:用 $Q(s,a)$ 平均值来估计 $Q(s,a)$.

The diagram illustrates the derivation of the Q-learning update formula. It starts with the definition of the average u_k as the sum of k values divided by k . This is then expressed as a weighted sum of the current value x_k and the previous average u_{k-1} . The learning rate α is introduced as the weight for the current value. The final formula for the Q-value update is shown at the bottom, with arrows indicating the correspondence between its terms and the intermediate variables.

$$u_k = \frac{1}{k} \sum_{j=1}^k x_j = \frac{1}{k} \left(x_k + \sum_{j=1}^{k-1} x_j \right) = \frac{1}{k} (x_k + (k-1)u_{k-1}) = u_{k-1} + \frac{1}{k} (x_k - x_{k-1})$$

$x_1 + x_2 + \dots + x_k$ 平均值 (新平均值)

$$u_k = u_{k-1} + \alpha (x_k - x_{k-1})$$

Q-learning更新公式

$$u_k \leftarrow (1 - \alpha)u_{k-1} + \alpha \cdot x_{k-1}$$

新平均值 旧平均值 学习速率

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha) \left[r + \gamma \max_{a'} Q(s', a') \right]$$

Q Learning

- Example : flappy birds

1. Initialize Q arbitrarily //随机初始化Q值
2. Repeat (for each episode): //每一次游戏, 从小鸟出生到死亡是一个episode
3. Initialize S //小鸟刚开始飞, S为初始位置的状态
4. Repeat (for each step of episode):
5. 根据当前Q和位置S, 使用一种策略, 得到动作A //这个策略可以是 ϵ -greedy等
6. 做了动作A, 小鸟到达新的位置S', 并获得奖励R //奖励可以是1, 50或者-1000
7. $Q(S,A) \leftarrow (1-\alpha)*Q(S,A) + \alpha*[R + \gamma*\max_a Q(S',a)]$ //更新之前位置的Q
8. $S \leftarrow S'$
9. until S is terminal //即到小鸟死亡为止

Deep Q Learning

- 算法

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right] \quad (3) \quad \text{deep}$$

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{D} to capacity N
Initialize action-value function Q with random weights
for episode = 1, M **do**
 Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
 for $t = 1, T$ **do**
 With probability ϵ select a random action a_t
 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
 Execute action a_t in emulator and observe reward r_t and image x_{t+1}
 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}
 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}
 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
 end for
end for
