
No Coding Farmer

Tao Hu

Department of Computer Science

Peking University

No.5 Yiheyuan Road Haidian District, Beijing, P.R.China

taohu@pku.edu.cn

Abstract

Some Miscellaneous Summary.

Contents

1 Expectation Maximization Introduction	4
1.1 EM Induction	4
1.2 EM convergence proof	4
1.3 Different Writing Style of EM Algorithm	4
2 EM applications	5
2.1 Gaussian Mix Model	5
2.2 Hidden Markov Model	5
2.3 Naive Bayesian	6
2.4 other papers	6
3 VAE	6
4 ADMM	6
5 Key steps you must know when building a DL Framework	6
5.1 Convolution	6
5.2 Loss Function	7
6 R-PCA	8
6.1 Solve RPCA by ADMM	9
6.2 Adaptive Penalty for ADMM	9
7 SFM	9
8 Mainfold Learning	10
8.1 Laplace Matrix	10
8.2 Normalized Cut	10
8.3 Linear Dimension Reduction	10
8.3.1 PCA	10
8.4 NonLinear Dimension Reduction	11
9 DCT	11
10 KL-divergence Application	15
11 GAN	15
11.1 Why is maxD then minG	16
11.2 WGAN	16
12 Reinforcement Learning	18
12.1 Model-based Method	21

12.1.1	Policy Iteration	21
12.1.2	Value Iteration	22
12.2	Model-Free Method	23
12.2.1	on-policy TD	23
12.2.2	on-policy MCMC	23
12.2.3	off-policy method	23
12.2.4	Comparisons: DP, MC, TD	23
13	RNN	26
13.1	BPTT	26
13.2	LSTM	26
13.3	Multidimension RNN	28
14	Algorithm	28
15	Determinantal Point Processes	28
16	Typical Networks	29
16.1	ResNext	29

1 Expectation Maximization Introduction

1.1 EM Induction

$$L(\theta) = \sum_{i=1}^M \log p(X; \theta) = \sum_{i=1}^m \log \sum_z p(X, Z; \theta)$$

let θ_i be some distribution over z's ($\sum_z \theta_i(z) = 1, \theta_i(z) \geq 0$)

$$\begin{aligned} & \sum_i \log p(X^{(i)}; \theta) \\ &= \sum_i \log \sum_{Z^{(i)}} \theta_i(Z^{(i)}) \frac{p(X^{(i)}, Z^{(i)}; \theta)}{\theta_i(Z^{(i)})} \\ &\geq \sum_i \sum_{Z^{(i)}} \theta_i(Z^{(i)}) \log \frac{p(X^{(i)}, Z^{(i)}; \theta)}{\theta_i(Z^{(i)})} (f(x) = \log x \text{ is concave.}) \end{aligned}$$

$$\text{let } \frac{p(X^{(i)}, Z^{(i)}; \theta)}{\theta_i(Z^{(i)})} = C$$

the equality can be only reached when $\frac{p(X^{(i)}, Z^{(i)}; \theta)}{\theta_i(Z^{(i)})}$ is a constant.

$$\text{we can get: } \sum_i \frac{p(X^{(i)}, Z^{(i)}; \theta)}{C} = 1 \text{ namely: } \sum_i p(X^{(i)}, Z^{(i)}; \theta) = C$$

$$\text{further induction: } \theta_i(Z^{(i)}) = \frac{p(X^{(i)}, Z^{(i)}; \theta)}{\sum_i p(X^{(i)}, Z^{(i)}; \theta)} = p(Z^{(i)}|X^{(i)}; \theta)$$

so the procedure of EM algorithm is:

Repeat Until Convergence:

- E-step: for each i, get $Q_i(Z^{(i)}) = p(Z^{(i)}|X^{(i)}; \theta)$
- M-step: $\theta := \operatorname{argmax}_{\theta} \sum_i \sum_{Z^{(i)}} Q_i(Z^{(i)}) \log \frac{p(X^{(i)}, Z^{(i)}; \theta)}{Q_i(Z^{(i)})}$

1.2 EM convergence proof

$$\text{let } l(\theta^{(t)}) = \sum_i \sum_{Z^{(i)}} Q_i^{(t)}(Z^{(i)}) \log \frac{p(X^{(i)}, Z^{(i)}; \theta)}{Q_i^{(t)}(Z^{(i)})}$$

then, we have the following inequality:

$$\begin{aligned} & l(\theta^{(t+1)}) \\ & \geq \sum_i \sum_{Z^{(i)}} Q_i^{(t)}(Z^{(i)}) \log \frac{p(X^{(i)}, Z^{(i)}, \theta^{(t+1)})}{Q_i^{(t)}(Z^{(i)})} \\ & \geq \sum_i \sum_{Z^{(i)}} Q_i^{(t)}(Z^{(i)}) \log \frac{p(X^{(i)}, Z^{(i)}, \theta^{(t)})}{Q_i^{(t)}(Z^{(i)})} \\ & \geq l(\theta^{(t)}) \end{aligned}$$

the first inequality is because: $l(\theta) \geq \sum_i \sum_{Z^{(i)}} Q_i^{(t)}(Z^{(i)}) \log \frac{p(X^{(i)}, Z^{(i)}, \theta)}{Q_i^{(t)}(Z^{(i)})} \forall \theta, Q_i$

the second inequality is because of the maximum of the M-step.

Hence, EM causes the likelihood to converge monotonically.

1.3 Different Writing Style of EM Algorithm

There are many writing style of EM algorithm. here I just mention the book <Statistics Learning Method> by LiHang who is very famous in China.

EM algorithm from LiHang(Li-version):

Algorithm 1 EM from LIHang

Require: observation X,hidden variable Z,joint distribution $P(X, Z|\theta)$,conditional distribution $P(Z|Y, \theta)$

while Not convergence **do**

E-Step: let $\theta^{(i)}$ is the i-th estimate of θ ,

$$Q(\theta, \theta^{(i)}) = E_z[\log P(X, Z|\theta)|X, \theta^{(i)}] = \sum_Z \log P(X, Z|\theta)P(Z|X, \theta^{(i)})$$

M-step: $\theta^{(i+1)} = \operatorname{argmax}_{\theta} Q(\theta, \theta^{(i)})$

end while

output model parameter θ

it seems that Li-version is different from the above version. however, they are the same. because:

- the above version just consider every data, so that it include subscript i. however Li-version only consider one data.
- the above version can be transformed to Li-version.

$$\begin{aligned} & \sum_Z Q(Z) \log \frac{P(X, Z|\theta)}{Q(Z)} \\ &= \sum_Z P(Z|X; \theta^{(t)}) \log \frac{p(X, Z|\theta)}{p(Z|X; \theta^{(t)})} \\ &= \sum_Z P(Z|X; \theta^t) \log P(X, Z; \theta) - \sum_Z P(Z|X; \theta^{(t)}) \log P(Z|X; \theta^{(t)}) \end{aligned}$$

as the variable is θ ,so $\sum_Z P(Z|X; \theta^{(t)}) \log P(Z|X; \theta^{(t)})$ can be removed.

- $Q(\theta, \theta^{(i)}) = \sum_Z \log P(X, Z|\theta)P(Z|X; \theta^{(i)})$ can be also written as $Q(\theta, \theta^{(i)}) = \sum_Z \log P(X, Z|\theta)P(Z, X; \theta^{(i)})$,because X is a observation.

2 EM applications

2.1 Gaussian Mix Model

GMM can be solved by EM. notice here we use the expectation of EM:

$$\begin{aligned} & Q(\theta, \theta^{(i)}) \\ &= E_{\gamma}[\log P(y, \gamma|\theta)|y, \theta^{(i)}] \\ &= E[\sum_{k=1}^K [n_k \log \alpha_k + \sum_{j=1}^N \gamma_{jk} [\log \frac{1}{\sqrt{2\pi}} - \log \sigma_k - \frac{1}{2\sigma_k^2} (y_j - \mu_k)^2]]] \\ &= \sum_{k=1}^K [(E\gamma_{jk}) \log \alpha_k + \sum_{j=1}^N (E\gamma_{jk}) [\log \frac{1}{\sqrt{2\pi}} - \log \sigma_k - \frac{1}{2\sigma_k^2} (y_j - \mu_k)^2]] \end{aligned}$$

here $(E\gamma_{jk})$ can be easily calculated.

$\hat{\mu}_k, \hat{\sigma}_k^2$ can be acquired by derivation.

$\hat{\alpha}_k$ can be acquired by the derivation on the Lagrangian($\sum_i^K \alpha_k = 1$).

2.2 Hidden Markov Model

HMM Learning Method is also called Baum-Welch algorithm.the target is learning $\lambda = (A, B, \pi)$.

Q function is:

$$Q(\lambda, \bar{\lambda}) = \sum_I \log P(O, I|\lambda)P(O, I|\bar{\lambda})$$

$$P(O, I, \lambda) = \pi_{i1} b_{i1}(o_1) a_{i1i2} b_{i2}(o_2) \dots a_{iT-1iT} b_{iT}(o_T)$$

so the Q function can also be written as:

$$Q(\lambda, \bar{\lambda}) = \sum_I \log \pi_i P(O, I | \bar{\lambda}) + \sum_I (\sum_{t=1}^{T-1} \log a_{i,i+1}) P(O, I | \bar{\lambda}) + \sum_I (\sum_{t=1}^T \log b_{it}(o_t)) P(O, I | \bar{\lambda})$$

note here: I is not only one state. it includes state length from 1 to T,which all start from i_1

so we can solve the maximum of Q function by derivation on the Lagrangian polynomial (because exists these limitations: $\sum_{i=1}^N \pi_i = 1$, $\sum_{j=1}^N a_{ij} = 1$, $\sum_{i=1}^M b_i = 1$)

2.3 Naive Bayesian

2.4 other papers

We can use softmax to model transition probability, normal distribution to model emission probability.

it's a good example in Car that Knows Before You Do: Anticipating Maneuvers via Learning Temporal Driving Models, the AIO-HMM can be more complicated,which can be enriched by the graphic model by M.I Jordon.

3 VAE

here is a complete VAE tutorial [2]

$$\begin{aligned} & \max \log P(x) \\ \text{lhs} &= \log \int P(x, z) dz \\ &= \log \int P(x/z)p(z) dz \\ &= \log \int \frac{P(x/z)}{q(z/x)} q(z/x)p(z) dz \\ &= \log E_{q(z/x)} \left[\frac{p(x/z)}{q(z/x)} p(z) \right] \end{aligned}$$

jenson's inequality,we can know: $\geq E_{q(z/x)} [\log \frac{p(x/z)}{q(z/x)} p(z)]$

$$\begin{aligned} &= E_{q(z/x)} [\log p(x/z)] + E_{q(z/x)} [\log \frac{p(z)}{q(z/x)}] \\ &= E_{q(z/x)} [\log p(x/z)] - E_{q(z/x)} [\log \frac{q(z/x)}{p(z)}] \\ &= E_{q(z/x)} [\log p(x/z)] - KL(q(z/x) || p(z)) \end{aligned}$$

4 ADMM

5 Key steps you must know when building a DL Framework

5.1 Convolution

convert the convolution to matrix multiplication like the fully-connected network.

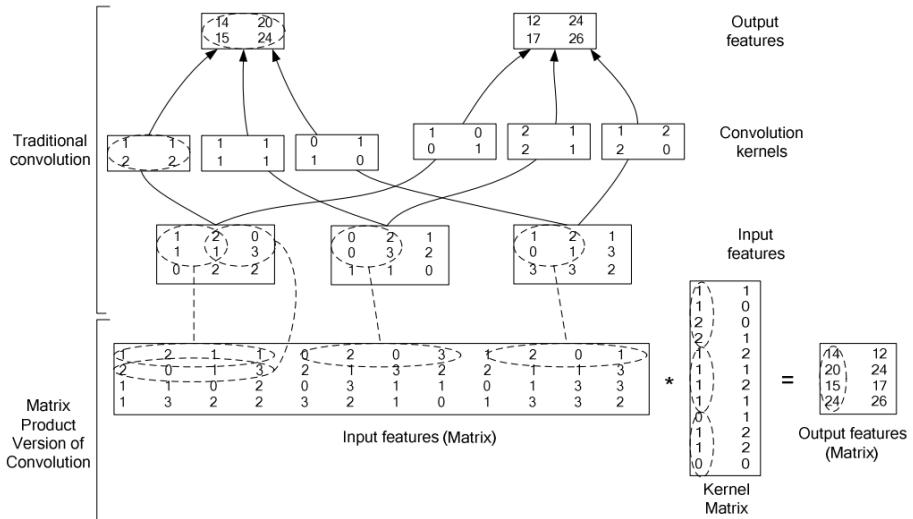


Figure 2. Example convolution operations in a convolutional layer (biases, sub-sampling, and non-linearity omitted). The top figure presents the traditional convolution operations, while the bottom figure presents the matrix version.

figure is from [1].

Let's note:

H:image height

W:image width

in: input image number

out: output image number

K: convolution kernel size

the matrix product version of convolution, the dimension of two matrix is:

$(H*W) * (in*K*K)$

$(in*K*K) * out$

the operation above is like matlab function: img2col
`im2col(A,[m n],block_type)`, where `block_type="sliding"`.

GEMM(General Matrix to Matrix Multiplication) is at the heart of deep learning.<https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/>

5.2 Loss Function

<https://stats.stackexchange.com/questions/222585/what-are-the-impacts-of-choosing-different-loss>

Hinge Loss:

$$f(y, t) = (1 - yt)_+$$

Log Loss:

$$f(y, t) = \ln(1 + e^{-yt})$$

Square Loss:

$$f(y, t) = (1 - yt)^2$$

square loss is sensitive to outliers.

Following plot is coming from Chris Bishop's PRML book. The Hinge Loss is plotted in blue, the Log Loss in red, the Square Loss in green and the 0/1 error in black.

Cross Entropy:

<https://jamesmccaffrey.wordpress.com/2013/11/05/why-you-should-use-cross-entropy-error-instead-of-hinge-loss/>

$$\sum_i^n \sum_k^K -y_{true}^k \log(y_{predict}^{(k)})$$

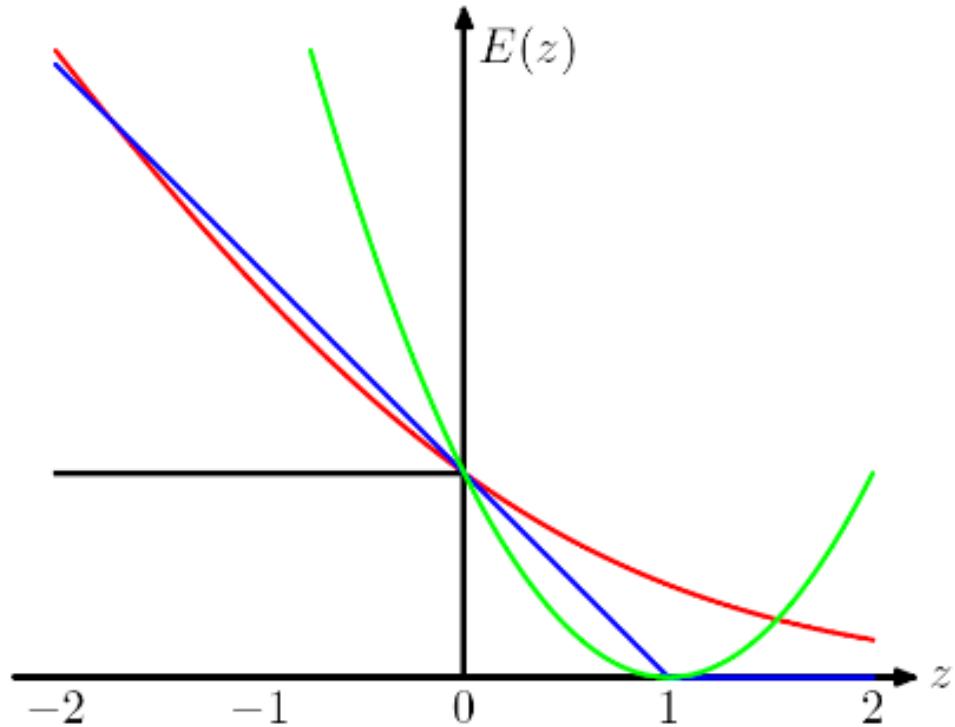
when multi classification, only exists one $y_{true}^i = 1$ with remaining $y_{true}^j = 0, i \neq j$.

Interpretation of Cross Entropy:

(1). encoding length: if a signal exist with probability p. then we only need $\log(\frac{1}{p})$ bits to encode it.

(2). KL-divergence. $E_{x \sim p}[-\log(Q(x))] - E_{x \sim p}[-\log P(x)] = E_{x \sim p}[\log \frac{P(x)}{Q(x)}] = KL(P||Q)$

Here the empirical distribution for each data point simply assigns probability 1 to the class of that data point, and 0 to all other classes. namely p only equals 1 once. **notice here P is empirical distribution from network, it's known. what we only optimize is Q!**



6 R-PCA

RPCA problem:

$$\min_{A,E} \|A\|_* + \lambda \|E\|_1 \\ \text{S.t } D = A + E$$

RPCA dual problem:

Augmented Lagrangian is :

$$A_t(A, E; \Lambda) = \min_{A,E} L(A, E; \Lambda) \\ = \min_{A,E} \|A\|_* + \lambda \|E\|_1 + \langle \Lambda, D - A - E \rangle \\ = \min_A \|A\|_* - \langle \Lambda, A \rangle + \\ \min_E \lambda \|E\|_1 - \langle \Lambda, E \rangle + \langle \Lambda, D \rangle$$

both of the sub-problem is conjugate function, according to the property of conjugate function :

$$A_t(A, E; \Lambda) = \langle \Lambda, D \rangle \\ \text{S.t } \|\Lambda\|_2 \leq 1, \|\Lambda\|_\infty \leq \lambda$$

so the dual problem is:

$$\begin{aligned} & \max_{\Lambda} \quad \langle \Lambda, D \rangle \\ S.t \quad & \|\Lambda\|_2 \leq 1, \|\Lambda\|_\infty \leq \lambda \end{aligned}$$

6.1 Solve RPCA by ADMM

the ADMM sub-problem is:

A-sub-problem:

$$A_{k+1} = \operatorname{argmin}_A \|A\|_* + \frac{\beta}{2} \|D - A - E_k + \Lambda_k/\beta\|_F^2$$

E-sub-problem:

$$E_{k+1} = \operatorname{argmin}_E \lambda \|E\|_1 + \frac{\beta}{2} \|D - A_{k+1} - E + \Lambda_k\|_F^2$$

E-sub-problem has closed-form solution as follows:

$$E_{k+1} = S_{\lambda\beta^{-1}}(D - A_k + \Lambda_k/\beta).$$

$S_\epsilon = \operatorname{sgn}(x) \max(|x| - \epsilon, 0)$, which is the same form as shrinkage.

A-sub-problem has a closed-form solution offered by Singular Value Thresholding(SVT): suppose that the SVD of $W = D - E_k + \Lambda_k/\beta$ is $W = U\Sigma V^T$, then the optimal solution is $A = US_{\beta^{-1}}(\Sigma)V^T$.

6.2 Adaptive Penalty for ADMM

Lin et al.[5] suggest updating the penalty parameter β as follows:

$$\beta_{k+1} = \min(\beta_{\max}, \rho\beta_k)$$

where ρ_{\max} is an upper bound of $\{\beta_k\}$. the value of ρ is defined as:

$$\rho = \begin{cases} \rho_0 & \text{if } \frac{\beta_k \max(\sqrt{\eta_A} \|x_{k+1} - x_k\|_2, \sqrt{\eta_B} \|y_{k+1} - y_k\|_2)}{\|c\|_2} < \epsilon_2 \\ 1 & \text{otherwise} \end{cases}$$

where η_A, η_B is linearized Taylor second-order factor.

7 SFM

<https://www.robots.ox.ac.uk/~vgg/hzbook/hzbook2/HZepipolar.pdf>

- F is a rank 2 homogeneous matrix with 7 degrees of freedom.
- **Point correspondence:** If \mathbf{x} and \mathbf{x}' are corresponding image points, then $\mathbf{x}'^T F \mathbf{x} = 0$.
- **Epipolar lines:**
 - ◊ $\mathbf{l}' = F\mathbf{x}$ is the epipolar line corresponding to \mathbf{x} .
 - ◊ $\mathbf{l} = F^T \mathbf{x}'$ is the epipolar line corresponding to \mathbf{x}' .
- **Epipoles:**
 - ◊ $F\mathbf{e} = \mathbf{0}$.
 - ◊ $F^T \mathbf{e}' = \mathbf{0}$.
- **Computation from camera matrices P, P' :**
 - ◊ General cameras, $F = [\mathbf{e}']_{\times} P' P^+$, where P^+ is the pseudo-inverse of P , and $\mathbf{e}' = P' \mathbf{C}$, with $P \mathbf{C} = \mathbf{0}$.
 - ◊ Canonical cameras, $P = [\mathbf{I} \mid \mathbf{0}]$, $P' = [\mathbf{M} \mid \mathbf{m}]$, $F = [\mathbf{e}']_{\times} \mathbf{M} = \mathbf{M}^{-T} [\mathbf{e}]_{\times}$, where $\mathbf{e}' = \mathbf{m}$ and $\mathbf{e} = \mathbf{M}^{-1} \mathbf{m}$.
 - ◊ Cameras not at infinity $P = K[\mathbf{I} \mid \mathbf{0}]$, $P' = K'[\mathbf{R} \mid \mathbf{t}]$, $F = K'^{-T} [\mathbf{t}]_{\times} R K^{-1} = [K' \mathbf{t}]_{\times} K' R K^{-1} = K'^{-T} R K^T [K R^T \mathbf{t}]_{\times}$.

Figure 1: Summary of Fundamental matrix properties

8 Mainfold Learning

<http://www.cad.zju.edu.cn/reports/%C1%F7%D0%CE%D1%A7%CF%B0.pdf>

8.1 Laplace Matrix

8.2 Normalized Cut

8.3 Linear Dimension Reduction

PCA,CDMS,RP

8.3.1 PCA

let $X = (X_1, X_2, X_3, \dots, X_p)^T$, the covariance matrix is:

$$\Sigma = (\sigma_{ij})_{p \times p} = E[(X - E(X))(X - E(X))^T]$$

which is a p-order semi-positive definite matrix. let $\mathbf{l}_i = (l_{i1}, l_{i2}, l_{i3}, \dots, l_{ip})^T$ be a column vector. consider the linear transformation:

$$Y = L^T X$$

$Y = (Y_1, Y_2, \dots, Y_p)^T$, $L = (l_1, l_2, \dots, l_p)$. we have:

$$\begin{aligned} Var(Y_i) &= Var(l_i^T X) = l_i^T \Sigma l_i \\ Cov(Y_i, Y_j) &= Cov(l_i X, l_j X) = l_i^T \Sigma l_j \end{aligned}$$

the above formulation is easy to understand, where Y_i, X is variable! l_i is a constant!
just recall the one-dimension circumstance: variance(3X) = 9 variance(X), covariance(2X,3X) = 6 variance(X).

the final optimization problem is:

$$\begin{aligned} \max_{l_1} \text{Var}(Y_1) &= l_1^T \Sigma l_i \\ \text{S.t } l_1^T l_1 &= 1 \end{aligned}$$

the optimization of l_2 :

$$\begin{aligned} \max_{l_2} \text{Var}(Y_2) &= l_2^T \Sigma l_i \\ \text{S.t } l_2^T l_2 &= 1 \\ l_1^T l_2 &= 0 \end{aligned}$$

the optimization of l_3 :

$$\begin{aligned} \max_{l_3} \text{Var}(Y_3) &= l_3^T \Sigma l_i \\ \text{S.t } l_3^T l_3 &= 1 \\ l_1^T l_3 &= 0 \\ l_2^T l_3 &= 0 \end{aligned}$$

8.4 NonLinear Dimension Reduction

KPCA,ISOMAP,LLE,LSTA,LPCA, LE(Laplacian EigenMaps) , Diffusion Maps, MVU

9 DCT

http://eeweb.poly.edu/~yao/EE3414/ImageCoding_DCT.pdf

1D Unitary Transform

Consider the N -point signal $s(n)$ as an N -dimensional vector

$$\mathbf{s} = \begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_{N-1} \end{bmatrix}$$

The inverse transform says that \mathbf{s} can be represented as the sum of N basis vectors

$$\mathbf{s} = t_0 \mathbf{u}_0 + t_1 \mathbf{u}_1 + \dots + t_{N-1} \mathbf{u}_{N-1}$$

where \mathbf{u}_k corresponds to the k -th transform kernel :

$$\mathbf{u}_k = \begin{bmatrix} u_{k,0} \\ u_{k,1} \\ \vdots \\ u_{k,N-1} \end{bmatrix}$$

The forward transform says that the expansion coefficient t_k can be determined by the inner product of \mathbf{s} with \mathbf{u}_k :

$$t_k = (\mathbf{u}_k, \mathbf{s}) = \sum_{n=0}^{N-1} u_{k,n}^* s_n$$

1D Discrete Cosine Transform

- Can be considered “real” version of DFT
 - Basis vectors contain only co-sinusoidal patterns

DFT

$$u_{k,n} = \frac{1}{\sqrt{N}} \exp\left(j \frac{2\pi k}{N} n\right) = \frac{1}{\sqrt{N}} \left(\cos\left(\frac{2\pi k}{N} n\right) + j \sin\left(\frac{2\pi k}{N} n\right)\right)$$

$$\mathbf{u}_k = \frac{1}{\sqrt{N}} \begin{bmatrix} \cos\left(\frac{2\pi k}{N} 0\right) \\ \cos\left(\frac{2\pi k}{N} 1\right) \\ \vdots \\ \cos\left(\frac{2\pi k}{N} (N-1)\right) \end{bmatrix} + j \frac{1}{\sqrt{N}} \begin{bmatrix} \sin\left(\frac{2\pi k}{N} 0\right) \\ \sin\left(\frac{2\pi k}{N} 1\right) \\ \vdots \\ \sin\left(\frac{2\pi k}{N} (N-1)\right) \end{bmatrix}$$

DCT

$$u_{k,n} = \alpha(k) \cos\left(\frac{\pi k}{2N} (2n+1)\right)$$

$$\alpha(0) = \sqrt{\frac{1}{N}}, \alpha(k) = \sqrt{\frac{2}{N}}, k = 1, 2, \dots, N-1$$

$$\mathbf{u}_k = \alpha(k) \begin{bmatrix} \cos\left(\frac{\pi k}{2N} 1\right) \\ \cos\left(\frac{\pi k}{2N} 3\right) \\ \vdots \\ \cos\left(\frac{\pi k}{2N} (2N+1)\right) \end{bmatrix}$$

Example: 4-point DCT

$$\text{Using } u_{k,n} = \alpha(k) \cos\left(\frac{k\pi}{2} (2n+1)\right), \alpha(0) = \sqrt{\frac{1}{4}} = \frac{1}{2}, \alpha(k) = \sqrt{\frac{2}{4}} = \sqrt{\frac{1}{2}}, k \neq 0,$$

$$\text{1D DCT basis are: } \mathbf{u}_0 = \frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}; \mathbf{u}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} \cos\left(\frac{\pi}{8}\right) \\ \cos\left(\frac{3\pi}{8}\right) \\ \cos\left(\frac{5\pi}{8}\right) \\ \cos\left(\frac{7\pi}{8}\right) \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.9239 \\ 0.3827 \\ -0.3827 \\ -0.9239 \end{bmatrix}; \mathbf{u}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} \cos\left(\frac{\pi}{4}\right) \\ \cos\left(\frac{3\pi}{4}\right) \\ \cos\left(\frac{5\pi}{4}\right) \\ \cos\left(\frac{7\pi}{4}\right) \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}; \mathbf{u}_3 = \frac{1}{\sqrt{2}} \begin{bmatrix} \cos\left(\frac{3\pi}{8}\right) \\ \cos\left(\frac{9\pi}{8}\right) \\ \cos\left(\frac{15\pi}{8}\right) \\ \cos\left(\frac{21\pi}{8}\right) \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.3827 \\ -0.9239 \\ 0.9239 \\ -0.3827 \end{bmatrix}$$

For $\mathbf{s} = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 3 \end{bmatrix}$, determine the transform coefficients t_k . Also determine the reconstructed vector from all coefficients and two largest coefficients.

2D Separable Transform

Consider the $M \times N$ -point image \mathbf{S} as a $M \times N$ -dimensional array (matrix)

$$\mathbf{S} = \begin{bmatrix} S_{0,0} & S_{0,1} & \dots & S_{0,N-1} \\ S_{1,0} & S_{1,1} & \dots & S_{1,N-1} \\ \dots & \dots & \dots & \dots \\ S_{M-1,0} & S_{M-1,1} & \dots & S_{M-1,N-1} \end{bmatrix}$$

The inverse transform says that \mathbf{s} can be represented as the sum of $M \times N$ basis images

$$\mathbf{S} = T_{0,0}\mathbf{U}_{0,0} + T_{0,1}\mathbf{U}_{0,1} + \dots + T_{M-1,N-1}\mathbf{U}_{M-1,N-1}$$

where $\mathbf{U}_{k,l}$ corresponds to the (k, l) -th transform kernel :

$$\mathbf{U}_{k,l} = \mathbf{u}_k(\mathbf{u}_l)^T = \begin{bmatrix} u_{k,0} \\ u_{k,1} \\ \dots \\ u_{k,N-1} \end{bmatrix} \begin{bmatrix} u_{l,0}^* & u_{l,1}^* & \dots & u_{l,N-1}^* \end{bmatrix} = \begin{bmatrix} u_{k,0}u_{l,0}^* & u_{k,0}u_{l,1}^* & \dots & u_{k,0}u_{l,N-1}^* \\ u_{k,1}u_{l,0}^* & u_{k,1}u_{l,1}^* & \dots & u_{k,1}u_{l,N-1}^* \\ \dots & \dots & \dots & \dots \\ u_{k,N-1}u_{l,0}^* & u_{k,N-1}u_{l,1}^* & \dots & u_{k,N-1}u_{l,N-1}^* \end{bmatrix}$$

The forward transform says that the expansion coefficient S_k can be determined by the inner product of \mathbf{S} and $\mathbf{U}_{k,l}$:

$$T_{k,l} = (\mathbf{U}_{k,l}, \mathbf{S}) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} U_{k,l;m,n}^* S_{m,n}$$

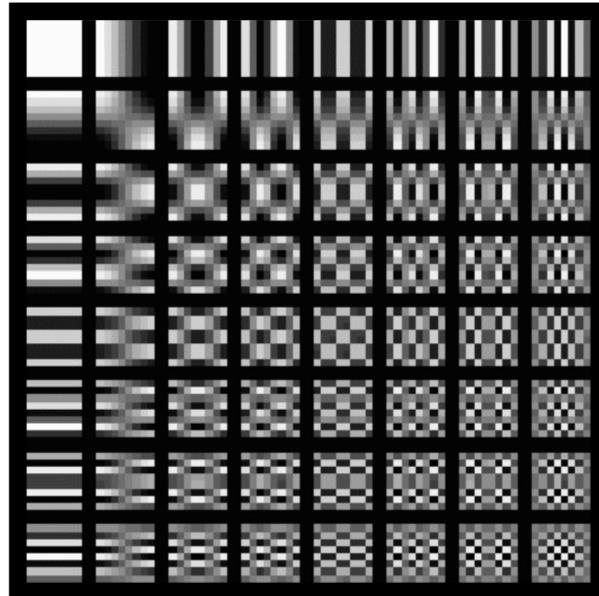
- Basis image = outer product of 1D DCT basis vector

$$\mathbf{u}_{k;N} = \alpha(k) \begin{bmatrix} \cos\left(\frac{\pi k}{2N} 1\right) \\ \cos\left(\frac{\pi k}{2N} 3\right) \\ \dots \\ \cos\left(\frac{\pi k}{2N} (2N+1)\right) \end{bmatrix}, \quad \alpha(0) = \sqrt{\frac{1}{N}}, \alpha(k) = \sqrt{\frac{2}{N}}, k = 1, 2, \dots, N-1$$

$$\begin{aligned} \mathbf{U}_{k,l;M,N} &= \mathbf{u}_{k;M} (\mathbf{u}_{l;N})^T \\ &= \alpha(k) \alpha(l) \begin{bmatrix} \cos\left(\frac{k\pi}{2M} 1\right) \cos\left(\frac{l\pi}{2N} 1\right) & \cos\left(\frac{k\pi}{2M} 1\right) \cos\left(\frac{l\pi}{2N} 3\right) & \dots & \cos\left(\frac{k\pi}{2M} 1\right) \cos\left(\frac{l\pi}{2N} (2N+1)\right) \\ \cos\left(\frac{k\pi}{2M} 3\right) \cos\left(\frac{l\pi}{2N} 1\right) & \cos\left(\frac{k\pi}{2M} 3\right) \cos\left(\frac{l\pi}{2N} 3\right) & \dots & \cos\left(\frac{k\pi}{2M} 3\right) \cos\left(\frac{l\pi}{2N} (2N+1)\right) \\ \dots & \dots & \dots & \dots \\ \cos\left(\frac{k\pi}{2M} (2M+1)\right) \cos\left(\frac{l\pi}{2N} 1\right) & \cos\left(\frac{k\pi}{2M} (2M+1)\right) \cos\left(\frac{l\pi}{2N} 3\right) & \dots & \cos\left(\frac{k\pi}{2M} (2M+1)\right) \cos\left(\frac{l\pi}{2N} (2N+1)\right) \end{bmatrix} \end{aligned}$$

Basis Images of 8x8 DCT

Low-Low



High-Low

High-High

Example: 4x4 DCT

Using $u_{k,n} = \alpha(k) \cos\left(\frac{k\pi}{2*4}(2n+1)\right)$, $\alpha(0) = \sqrt{\frac{1}{4}} = \frac{1}{2}$, $\alpha(k) = \sqrt{\frac{2}{4}} = \sqrt{\frac{1}{2}}$, $k \neq 0$,

$$\text{1D DCT basis are: } \mathbf{u}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}; \mathbf{u}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} \cos\left(\frac{\pi}{8}\right) \\ \cos\left(\frac{3\pi}{8}\right) \\ \cos\left(\frac{5\pi}{8}\right) \\ \cos\left(\frac{7\pi}{8}\right) \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.9239 \\ 0.3827 \\ -0.3827 \\ -0.9239 \end{bmatrix}; \mathbf{u}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} \cos\left(\frac{\pi}{4}\right) \\ \cos\left(\frac{3\pi}{4}\right) \\ \cos\left(\frac{5\pi}{4}\right) \\ \cos\left(\frac{7\pi}{4}\right) \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}; \mathbf{u}_3 = \frac{1}{\sqrt{2}} \begin{bmatrix} \cos\left(\frac{3\pi}{8}\right) \\ \cos\left(\frac{9\pi}{8}\right) \\ \cos\left(\frac{15\pi}{8}\right) \\ \cos\left(\frac{21\pi}{8}\right) \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.3827 \\ -0.9239 \\ 0.9239 \\ -0.3827 \end{bmatrix}$$

using $\mathbf{U}_{k,l} = \mathbf{u}_k (\mathbf{u}_l)^T$ yields:

$$\mathbf{U}_{0,0} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad \mathbf{U}_{0,2} = \frac{1}{4} \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix}, \quad \mathbf{U}_{2,0} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad \mathbf{U}_{2,2} = \frac{1}{4} \begin{bmatrix} 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}, \quad \dots$$

What Should You Know

- How to perform 2D DCT: forward and inverse transform
 - Manual calculation for small sizes, using inner product notation
 - Using Matlab: dct2, idct2
- Why DCT is good for image coding
 - Real transform, easier than DFT
 - Most high frequency coefficients are nearly zero and can be ignored
 - Different coefficients can be quantized with different accuracy based on human sensitivity
- How to quantize DCT coefficients
 - Varying stepsizes for different DCT coefficients based on visual sensitivity to different frequencies
 - A quantization matrix specifies the default quantization stepsize for each coefficient
 - The matrix can be scaled using a user chosen parameter (QP) to obtain different trade-offs between quality and size

10 KL-divergence Application

LINE: Large-scale Information Network Embedding <https://arxiv.org/pdf/1503.03578.pdf>

11 GAN

optimization problem:

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

notation:

p_g : generator's distribution

p_{data} : training data's distribution

$p_g(x)$: the probability that x comes from generator

$p_{data}(x)$: the probability that x comes from data

$p_z(z)$: a input noise variable

$G(z; \theta_g)$: generator network, input is z , parameter is θ_g

$D(x; \theta_d)$: discriminator network, input is x , parameter is θ_d

$D_G^*(x)$: optimal discriminator for any given generator G

K-L divergence:

$$KL(p_1 || p_2) = E_{x \sim p_1} \log \frac{p_1}{p_2} = \int_x p_1(x) \log \left(\frac{p_1(x)}{p_2(x)} \right) dx$$

JS divergence:

$$JS(p_1 || p_2) = \frac{1}{2} KL(p_1 || \frac{p_1+p_2}{2}) + \frac{1}{2} KL(p_2 || \frac{p_1+p_2}{2})$$

a important property of JS divergence:

the Jensen–Shannon divergence between two distributions is always non-negative and zero only when they are equal.

Firstly, let's image the generator is given!!!
just let $G(z) = x$, $z \sim p_z(Z)$, x is the G function operating on z , so $x \sim p_g$
therefore, $E_{z \sim p_z(z)}[\log(1 - D(G(z)))]$ can be converted to $E_{x \sim p_g(x)}[\log(1 - D(x))]$

given any generator G , we should maximize the quantity $V(G, D)$:
 $V(G, D) = \int_x [p_{data}(x)\log D(x) + p_g(x)(1 - D(x))]dx$

according to the definition calculus, we can calculate the optimum inner the calculus, which induces the problem to a $a \log(y) + b \log(1-y)$, the optimal value is : $\frac{a}{a+b}$. thus, the optimal discriminator is
 $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$

when $D(x)$ get optimal, we can convert $C(G)$ as:
 $C(G) = -\log(4) + \log \frac{p_{data}}{\frac{p_{data}^2}{p_{data} + p_d}} + \log \frac{p_d}{\frac{p_{data}^2}{p_{data} + p_d}}$
 $= -\log(4) + KL(p_{data}||p_d) + KL(p_d||p_{data})$
 $= -\log(4) + JS(p_{data}||p_d)$

Since the Jensen–Shannon divergence between two distributions is always non-negative and zero only when they are equal. we can show that $C(G) = -\log(4)$ is the global minimum of $C(G)$ and that the only solution is $p_g = p_{data}$, i.e., the generative model perfectly replicating the data generating process.

11.1 Why is maxD then minG

reference in Boyed's Book 5.4.3

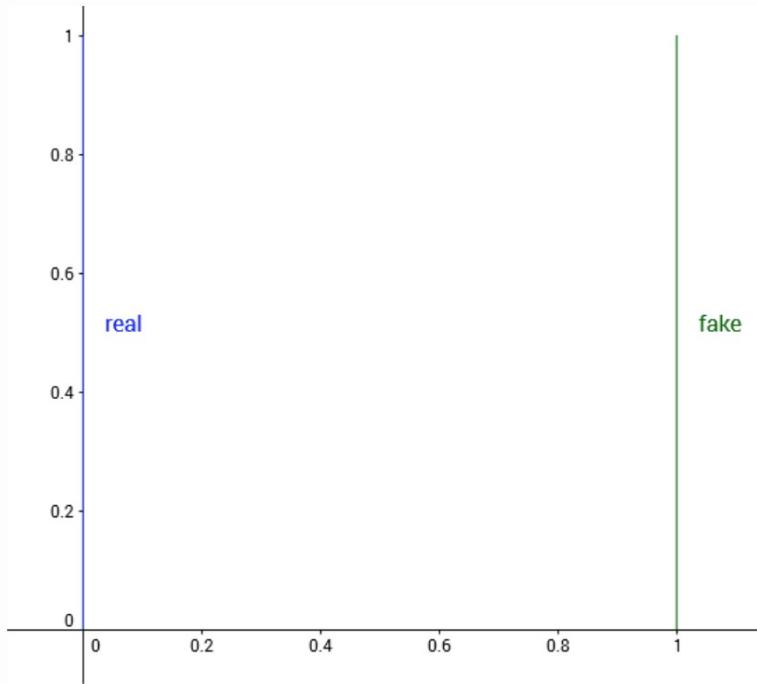
11.2 WGAN

<http://www.alexirpan.com/2017/02/22/wasserstein-gan.html>

Earth Mover (EM) or Wasserstein distance:
 $W(P_r, P_g) = \inf_{r \in \Pi(P_r, P_g)} E_{(x,y) \sim r}[||x - y||]$

Wasserstein distance advantage:
even two distribution have no overlap, Wasserstein distance can also describe the mutual distance .

Consider probability distributions defined over \mathbb{R}^2 . Let the true data distribution be $(0, y)$, with y sampled uniformly from $U[0, 1]$. Consider the family of distributions P_θ , where $P_\theta = (\theta, y)$, with y also sampled from $U[0, 1]$.



Real and fake distribution when $\theta = 1$

We'd like our optimization algorithm to learn to move θ to 0. As $\theta \rightarrow 0$, the distance $d(P_0, P_\theta)$ should decrease. But for many common distance functions, this doesn't happen.

- Total variation: For any $\theta \neq 0$, let $A = \{(0, y) : y \in [0, 1]\}$. This gives

$$\delta(P_0, P_\theta) = \begin{cases} 1 & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0. \end{cases}$$

- KL divergence and reverse KL divergence: Recall that the KL divergence $KL(P\|Q)$ is $+\infty$ if there is any point (x, y) where $P(x, y) > 0$ and $Q(x, y) = 0$. For $KL(P_0\|P_\theta)$, this is true at $(\theta, 0.5)$. For $KL(P_\theta\|P_0)$, this is true at $(0, 0.5)$.

$$KL(P_0\|P_\theta) = KL(P_\theta\|P_0) = \begin{cases} +\infty & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0, \end{cases}$$

- Jenson-Shannon divergence: Consider the mixture $M = P_0/2 + P_\theta/2$, and now look at just one of the KL terms.

$$KL(P_0\|M) = \int_{(x,y)} P_0(x, y) \log \frac{P_0(x, y)}{M(x, y)} dy dx$$

For any x, y where $P_0(x, y) \neq 0$, $M(x, y) = \frac{1}{2}P_0(x, y)$, so this integral works out to $\log 2$. The same is true of $KL(P_\theta\|M)$, so the JS divergence is

$$JS(P_0, P_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0, \end{cases}$$

- Earth Mover distance: Because the two distributions are just translations of one another, the best way transport plan moves mass in a straight line from $(0, y)$ to (θ, y) . This gives $W(P_0, P_\theta) = |\theta|$

This example shows that there exist sequences of distributions that don't converge under the JS, KL, reverse KL, or TV divergence, but which do converge under the EM distance.

This example also shows that for the JS, KL, reverse KL, and TV divergence, there are cases where the gradient is always 0. This is especially damning from an optimization perspective - any approach that works by taking the gradient $\nabla_\theta d(P_0, P_\theta)$ will fail in these cases.

Admittedly, this is a contrived example because the supports are disjoint, but the paper points out that when the supports are low dimensional manifolds in high dimensional space, it's very easy for the intersection to be measure zero, which is enough to give similarly bad results.

The calculation of Original Wasserstein distance is very difficult, A result from Kantorovich-Rubinstein duality shows Wasserstein is equivalent to:
 $W(P_r, P_\theta) = \sup_{||f||_L \leq K} E_{x \sim P_r}[f(x)] - E_{x \sim P_\theta}[f(x)]$
which format is very similar to the original GAN.

12 Reinforcement Learning

Markov Decision Process (**MDP**) is tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$:

r is a reward function, $r(s, a, s')$

γ is a discount factor

P is the transition probability distribution:

probability from state s with action a to state s' : $P(s'|s, a)$

S is a finite set of states.

A is a finite set of actions.

Markov Property:

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, \dots, s_t)$$

Stochastic Policy:

$\pi(a|s) = P(a_s = a | s_t = s)$, $\pi(a|s)$ here is a probability, we can often see $\pi(s)$, which returns a, means under policy π and state s , you should better take action a .

Value function:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

State-value function:

$$V_{\pi}(s) = E_{\pi}(G_t | s_t = s)$$

Action-value function:

$$Q_{\pi}(s, a) = E_{\pi}(G_t | s_t = s, a_t = a)$$

Bellman Equation:

$$\begin{aligned} V_{\pi}(s) &= E_{\pi}(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots | s_t = s) \\ &= E_{\pi}(r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \dots) | s_t = s) \\ &= E_{\pi}(r_{t+1} + \gamma G_{t+1} | s_t = s) \\ &= E_{\pi}(r_{t+1} + \gamma V_{\pi}(s_{t+1}) | s_t = s) \end{aligned}$$

For state-value function, Bellman Equation can be written as:

$$\begin{aligned} V_{\pi}(s) &= E_{\pi}(r_{t+1} + \gamma V_{\pi}(s_{t+1}) | s_t = s) \\ &= \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P(s'|s, a) [r(s, a, s') + \gamma V_{\pi}(s')] \\ r(s, a, s') &\text{ is same with } r_{t+1} \text{ to some extent.} \end{aligned}$$

For action-value function, Bellman Equation can be written as:

$$\begin{aligned} Q_{\pi}(s, a) &= E_{\pi}(r(s, a, s') + \gamma Q_{\pi}(s', a') | s, a) \\ &= \sum_{s' \in S} P(s'|s, a) [r(s, a, s') + \gamma \sum_{a' \in A} \pi(a', s') Q_{\pi}(s', a')] \end{aligned}$$

Normally, we just assume the $\pi(a|s), p(s'|s, a), r(s, a, s')$ are known (namely the MDP is known). so we can solve the linear equation above. however, when data become huge, it is not feasible to solve the Bellman Equation directly.

optimal value function:

the optimal state-value function $V_*(s)$ is :

$$V_*(s) = \max_{\pi} V_{\pi}(s)$$

the optimal action-value function $Q_*(s, a)$ is:

$$Q_*(s, a) = \max_{\pi} Q_{\pi}(s, a)$$

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$$

following important properties:

$$Q^*(s, a) = E[r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a]$$

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a)$$

$$Q^*(s, a) = \sum_{s' \in \mathcal{S}} p(s' | s, a) [r(s, a, s') + \gamma V^*(s')]$$

The goal for any MDP is finding the optimal value function.

Or equivalent an optimal policy π^* for any policy π , $V_{\pi^*}(s) \geq V_\pi(s), \forall s \in S$

how do we take action after we obtain the optimal $V^*(s)$? if we known the $p(s' | s, a)$, $r(s, a, s')$, then we can get: $Q^*(s, a) = \sum_{s' \in \mathcal{S}} p(s' | s, a) [R(s, a, s') + \gamma V^*(s')]$. after $Q^*(s, a)$ is acquired,it's more trivial to take action a when it's at state s.

Relation between Q and V Functions

reference: <http://www.cs.cmu.edu/~mgormley/courses/10601-s17/slides/lecture26-ri.pdf>

Q from V:

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} P(s' | s, a) [R(s, a, s') + \gamma V^\pi(s')]$$

V from Q:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a | s) Q^\pi(s, a)$$

V and Q

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a | s) \sum_{s' \in \mathcal{S}} p(s' | s, a) [r(s, a, s') + \gamma V^\pi(s')]$$

Q and V

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} p(s' | s, a) [r(s, a, s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a' | s') Q^\pi(s', a')]$$

more complicated..

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a | s) \sum_{s' \in \mathcal{S}} p(s' | s, a) [r(s, a, s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a' | s') Q^\pi(s', a')]$$

mainly can classified into two main approaches:

Model based approaches:

First we will discuss methods that need to know the model:

$$P(s'|s, a) \text{ and } R(s, a, s').$$

- Policy Iteration**
- Value Iteration**

Model-free approaches:

Then we will discuss “model-free” methods that do NOT need to know the model: $P(s'|s, a)$ and $R(s, a, s')$.

- Monte Carlo Method**
- TD Learning**

34

Figure 2: Structure of RL

12.1 Model-based Method

12.1.1 Policy Iteration

One drawback of policy iteration is that each iteration involves policy evaluation.

1. Initialization

- $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$.
- $\pi(s)$ is a deterministic policy.
- $\delta > 0$ is a small threshold parameter.

2. Policy Evaluation

repeat

$\Delta \leftarrow 0$

for all $s \in \mathcal{S}$ **do**:

$v \leftarrow V(s)$

$a \leftarrow \pi(s)$

$V(s) \leftarrow \sum_{s' \in \mathcal{S}} P(s'|s, a)[R(s, a, s') + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

end for

until $\Delta < \delta$

3. Policy Improvement

```

 $policyStable \leftarrow true$ 
for all  $s \in \mathcal{S}$  do:
     $b \leftarrow \pi(s)$ 
     $\pi(s) \leftarrow \arg \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s'|s, a)[R(s, a, s') + \gamma V(s')]$ 
    if  $b \neq \pi(s)$  then
         $policyStable \leftarrow false$ 
    end if
end for
if  $policyStable$  then
    STOP
else
    Go to 2 (Policy Evaluation)
end if

```

Policy Improvement just improve the policy $\pi(s)$, then when we back to policy evaluation, the next action a is determined by the policy $\pi(s)$ which was updated in policy improvement. thus the relationship of policy improvement and policy evaluation is founded.

12.1.2 Value Iteration

Value Iteration

Main idea:

Use the Bellman equation of V^* instead of V^π

The greedy operator:

$$[T^*V](s) := \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s'|s, a)[R(s, a, s') + \gamma V(s')]$$

V^* is the solution of $V = T^*(V)$ fixpoint iteration.

The value iteration update:

$k = 0$ and $V_0(s) \in \mathbb{R}$ for all $s \in \mathcal{S}$

repeat

for all $s \in \mathcal{S}$ **do**:

$$V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s'|s, a)[R(s, a, s') + \gamma V_k(s')]$$

end for

$k \leftarrow k + 1$

until $V_k(\cdot)$ converged

37

- 1, value iteration converges to the true solution of Bellman optimal equations
- 2, Learn optimal value function directly, unlike policy iteration, there is no explicit policy.

12.2 Model-Free Method

12.2.1 on-policy TD

TD0

TD(n)

TD λ

12.2.2 on-policy MCMC

12.2.3 off-policy method

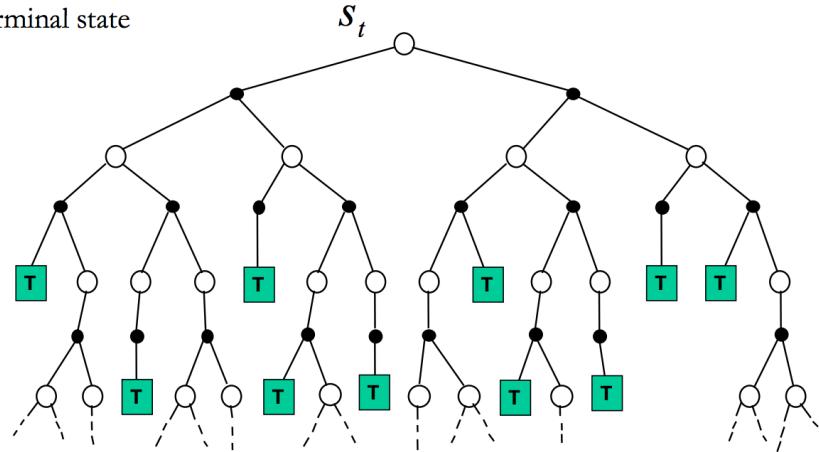
12.2.4 Comparisons: DP, MC, TD

Comparisons: DP, MC, TD

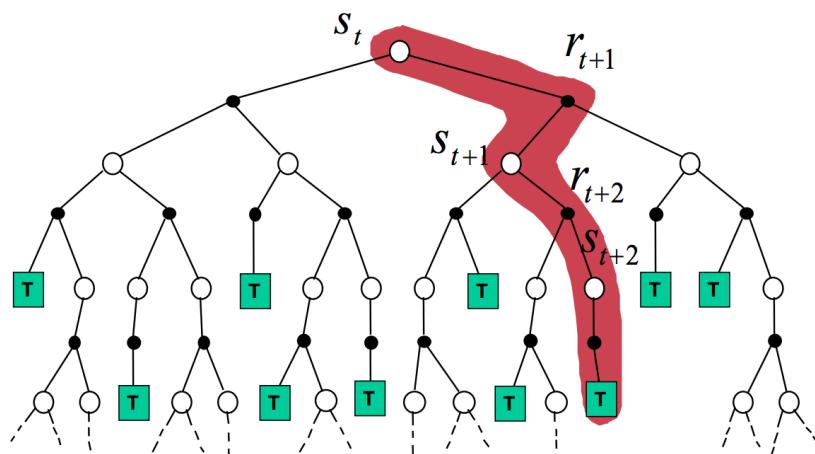
- They all estimate V^π
- DP: $V_k(s_t) \approx E_\pi(r_{t+1} + \gamma V_{k-1}(s_{t+1}) \mid s_t)$
 - Estimate comes from the Bellman equation
 - It needs to know the model
- TD: $V_k(s_t) \approx (r_{t+1} + \gamma V_{k-1}(s_{t+1}))$
 - Expectation is approximated with random samples
 - Doesn't need to wait for the end of the episodes.
- MC: $V_k(s_t) \approx R_t(s_t)$
 - Expectation is approximated with random samples
 - It needs to wait for the end of the episodes

MDP Backup Diagrams

- White circle: state
- Black circle: action
- T: terminal state

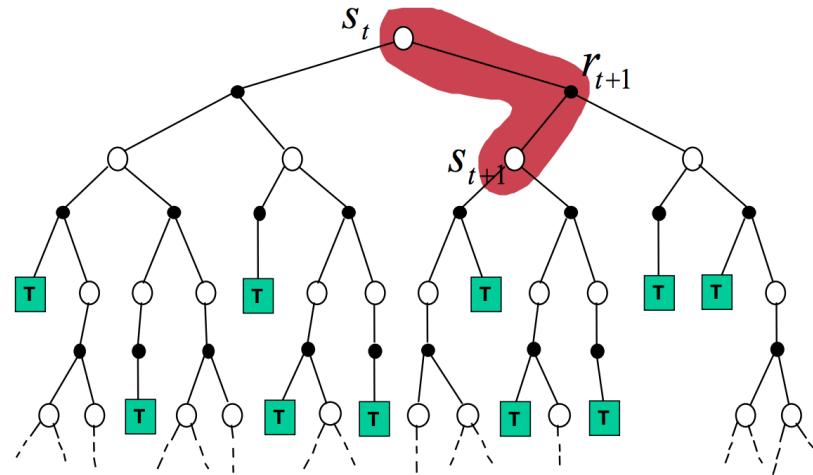


Monte Carlo Backup Diagram



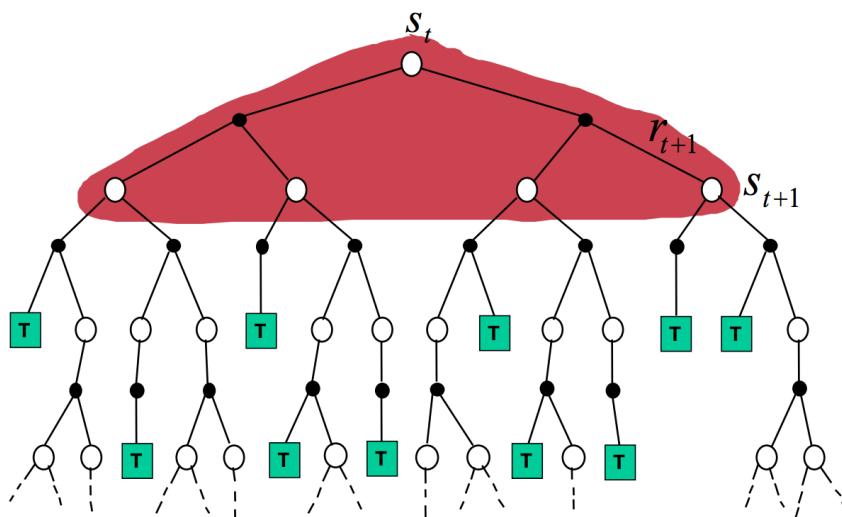
$$\text{MC estimate: } V_k(s_t) := V_{k-1}(s_t) + \alpha_k \cdot (R_k(s_t) - V_{k-1}(s_t))$$

Temporal Differences Backup Diagram



$$\text{TD estimate: } V_k(s_t) := V_{k-1}(s_t) + \alpha_k \cdot ((r_{t+1} + \gamma V_{k-1}(s_{t+1})) - V_{k-1}(s_t))$$

Dynamic Programming Backup Diagram



$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s, a) [R(s, a, s') + \gamma V^\pi(s')]$$

13 RNN

standard RNN:

$$\begin{aligned} a^{(t)} &= b + Wh^{(t-1)} + Ux^{(t)} \\ h^{(t)} &= \tanh(a^{(t)}) \\ o^{(t)} &= c + Vh^{(t)} \\ y^{(t)} &= \text{softmax}(o^{(t)}) \end{aligned}$$

13.1 BPTT

13.2 LSTM

<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

standard LSTM:

$$\begin{aligned} f_t &= \sigma_g(W_fx_t + U_fh_{t-1} + b_f) \\ i_t &= \sigma_g(W_ix_t + U_ih_{t-1} + b_i) \\ o_t &= \sigma_g(W_ox_t + U_oh_{t-1} + b_o) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_cx_t + U_ch_{t-1} + b_c) \\ h_t &= o_t \circ \sigma_h(c_t) \end{aligned}$$

f means forget gate, i means input gate, o means output gate h means hidden.

another equivalent writing style of LSTM is as follows, this writing style is equivalent to the original version,because $[x_t^T, h_{t-1}^T]^T$ means concat in row, and the W means W and U of original version, then utilize the Block Matrix Multiplication. most importantly, this style is more convenient to implement in the mainstream deep learning framework such as tensorflow,caffe etc.

$$\begin{aligned} i_t &= \text{sigmoid}(W_i[x_t^T, h_{t-1}^T]^T) \\ f_t &= \text{sigmoid}(W_f[x_t^T, h_{t-1}^T]^T) \\ o_t &= \text{sigmoid}(W_o[x_t^T, h_{t-1}^T]^T) \\ c_t &= i_t \circ \tanh(W_c[x_t^T, h_{t-1}^T]^T) + f_t \circ c_{t-1} \\ h_t &= o_t \circ \tanh(c_t) \end{aligned}$$

LSTM variants:

PeepHole:

$$\begin{aligned} f_t &= \sigma(W_f[c_{t-1}, h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i[c_{t-1}, h_{t-1}, x_t] + b_i) \\ o_t &= \sigma(W_o[c_t, h_{t-1}, x_t] + b_o) \end{aligned}$$

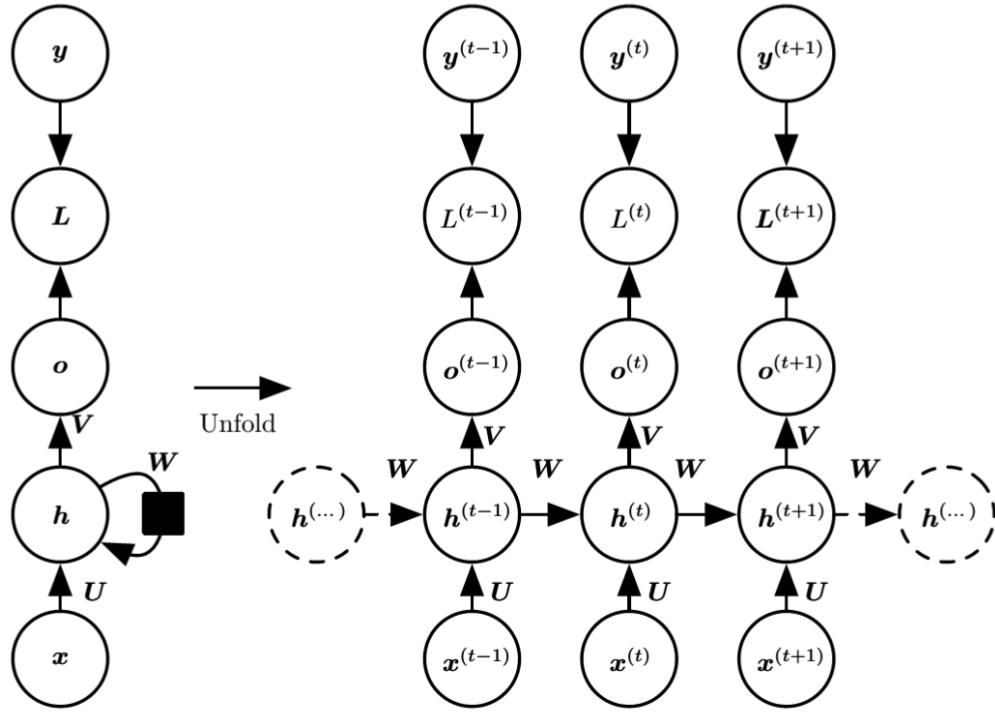
Gated Recurrent Unit (GRU):

$$\begin{aligned} z_t &= \sigma_g(W_zx_t + U_zh_{t-1} + b_z) \\ r_t &= \sigma_g(W_rx_t + U_rh_{t-1} + b_r) \\ h_t &= z_t \circ h_{t-1} + (1 - z_t) \circ \sigma_h(W_hx_t + U_h(r_t \circ h_{t-1}) + b_h) \end{aligned}$$

z_t : update gate

r_t : reset gate

h_t : output gate



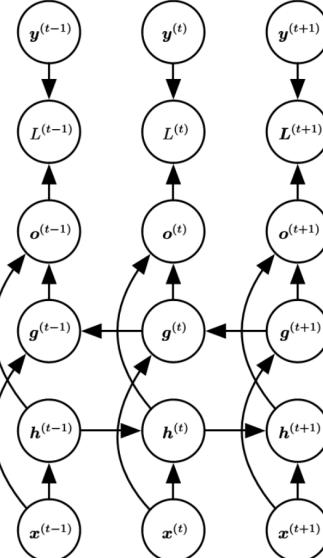
https://cs224d.stanford.edu/lecture_notes/LectureNotes4.pdf, the two RNNs can be imagined parallelized. the following is the structure of Bidirectional LSTM.

- In many applications, however, we want to output a prediction that may depend on the whole input sequence.

For example, in speech recognition, the correct interpretation of the current sound as a phoneme may depend on the next few phonemes because of co-articulation and may even depend on the next few words because of the linguistic dependencies between nearby words:

if there are two interpretations of the current word that are both acoustically plausible, we may have to look far into the future (and the past) to disambiguate them.

This is also true of handwriting recognition and many other sequence-to-sequence learning tasks



and several BiLSTM can also concatenated into a big LSTM network. (stacked LSTM)

13.3 Multidimension RNN

MDRNN[3] the most fundamental structure used for multi dimension data(image, video, fMRI). in this paper Multi direction MDRNN also was proposed by extending the conception of BiRNN. the forward pass MDRNN is as follows:

```

for  $x_1 = 0$  to  $X_1 - 1$  do
  for  $x_2 = 0$  to  $X_2 - 1$  do
    ...
    for  $x_n = 0$  to  $X_n - 1$  do
      initialize  $a \leftarrow \sum_j in_j^x w_{kj}$ 
      for  $i = 1$  to  $n$  do
        if  $x_i > 0$  then
           $a \leftarrow a + \sum_j h_j^{(x_1, \dots, x_{i-1}, \dots, x_n)} w_{kj}$ 
         $h_k^x \leftarrow \tanh(a)$ 

```

Algorithm 1: MDRNN Forward Pass

14 Algorithm

Longest Ordered Subsequence:

$h(i) = \max(h(j)) + 1$, where $j < i$ and $h(j) < h(i)$.
this is $O(n)$ solution.

we can reach a $O(n \log(n))$ solution via setting up a array to save previous information.(this method is very tricky.).

Bipartite graph: Hungarian algorithm: <http://blog.csdn.net/hurmishine/article/details/52749670>

15 Determinantal Point Processes

the following introduction is copied from [6].

Determinantal Point Processes (DPPs) are discrete probability models over the subsets of a ground set of N items. They provide an elegant model to assign probabilities to an exponentially large sample, while permitting tractable (polynomial time) sampling and marginalization. They are often used to provide models that balance “diversity” and quality, characteristics valuable to numerous problems in machine learning and related areas.

suppose we have a ground set of N items $\mathcal{Y} = \{1, \dots, N\}$, A discrete DPP over \mathcal{Y} is a probability measure \mathcal{P} on $2^{\mathcal{Y}}$ parametrized by a positive definite matrix K (the marginal kernel) such that $0 \leq K \leq I$, so that for any $Y \in \mathcal{Y}$ drawn from \mathcal{P} , the measure satisfies:

$$\forall A \subseteq \mathcal{Y}, P(A \subseteq Y) = \det(K_A)$$

where K_A is the submatrix of K indexed by elements in A . if a DPP with marginal kernel K assigns nonzero probability to the empty set, the DPP can alternatively be parametrized by a positive definite matrix L (the DPP kernel) so that:

$$P(Y) = \frac{\det(L_Y)}{L+I}$$

a brief manipulation([4] Eq.15) shows that when the inverse exist, $L = K(I - K)^{-1}$.

a detailed introduction pdf can be found here <https://jmhlldotorg.files.wordpress.com/2014/02/slidesrcc-dpps.pdf>.

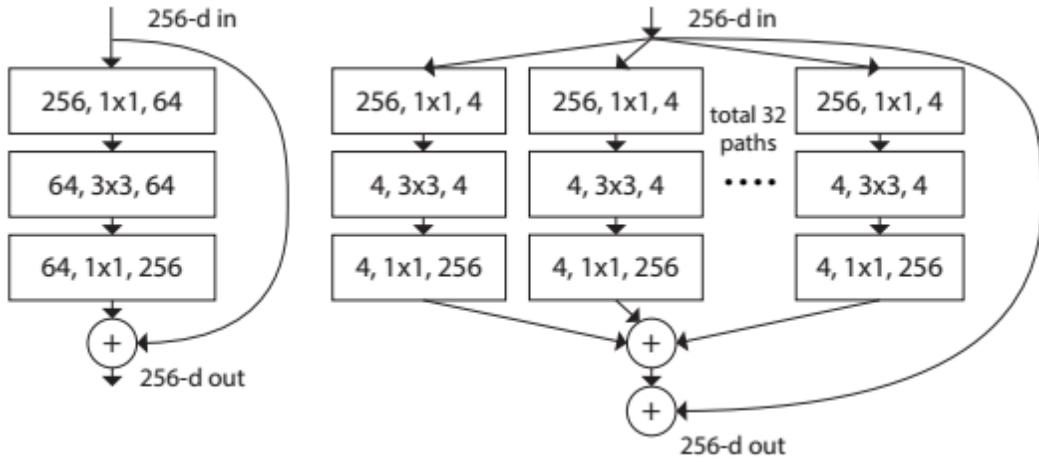
a application of DPP is video summarization:[7].

16 Typical Networks

16.1 ResNext

<http://www.cnblogs.com/lillylin/p/6799173.html>

- introduce a concept of cardinality.
- same parameter, but better effect!
- use a trick like the resnet: first 1*1 convolution to decrease the feature map number, then apply 3*3 convolution to do normal conv operation, finally use 1*1 convolution again to recover the feature map number.



Acknowledgments

References

- [1] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.
- [2] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.
- [3] Alex Graves, Santiago Fernandez, and Juergen Schmidhuber. Multi-dimensional recurrent neural networks, 2007.
- [4] Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. 2012.
- [5] Zhouchen Lin, Risheng Liu, and Zhixun Su. Linearized alternating direction method with adaptive penalty for low-rank representation. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 612–620. Curran Associates, Inc., 2011.
- [6] Zelda Mariet and Suvrit Sra. Kronecker determinantal point processes, 2016.
- [7] Ke Zhang, Wei-Lun Chao, Fei Sha, and Kristen Grauman. Video summarization with long short-term memory, 2016.