Advanced Machine Learning – summary

Table of Contents

Chapter1intro

- Nonlinear function of x, but linear function of the w_j.
- Sum-of-squares error:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$$

之所以上面要 square，因为这样求导后就是 w 的线性，有唯一解

- Least-Squares regression 结果（之所以有波浪号，其实就是 Xi 加 1，w 加 w_0）：

$$\tilde{\mathbf{w}} = (\widetilde{\mathbf{X}}\widetilde{\mathbf{X}}^T)^{-1}\widetilde{\mathbf{X}}\mathbf{t}$$

- Root mean square error(RMS)（有 2 因为之前除以 2）: E_{RMS}= √ (2E(w*)/N)
  1. Division by N lets us compare different data set sizes;
  2. Square root ensures E_{RMS} is measured on the same scale as the target variable t
- Ridge Regression:

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

- Multi-dimension Gaussian:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}}\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$

- Maximum likelihood systematically underestimates the variance of the distribution(frequentist concept)
- Maximum likelihood of Gaussian:

$$\hat{\mu} = \frac{1}{N}\sum_{n=1}^{N}x_n \quad \hat{\sigma}^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - \hat{\mu})^2$$

L(θ)=p(X| θ)

$$p(x|X) = \int \frac{p(x|\theta)L(\theta)p(\theta)}{p(X)}d\theta = \int \frac{p(x|\theta)L(\theta)p(\theta)}{\int L(\theta)p(\theta)d\theta}d\theta$$

上式给我的理解就是，在已知 X 这个数据集情况下，就算 theta 的 posterior(即 L*p/normalization)，乘上 p(x| θ),就是出现 x 的可能性。（下面也是个积分）

Chapter2linear-regression

- Least-squares regression is equivalent to Maximum Likelihood under the assumption of Gaussian noise.
- $p(\mathbf{w}|\mathbf{X},\mathbf{t},\beta,\alpha) \propto p(\mathbf{t}|\mathbf{X},\mathbf{w},\beta)p(\mathbf{w}|\alpha)$

  感觉左式就是右式/p(t)；
- Maximum-a-posteriori: 个人感觉就是让参数服从一个分布，有个 prior，之间就不会差太多，所以就是对参数的 regularization, λ =α/β（α是 w 参数 Gaussian 分布的 precision）
- L2-regularized regression (Ridge regression) is equivalent to MAP estimation with a Gaussian prior on the parameters w.
- 进一步的 bayesian（注意看 s(x)2，前项在之前的 maximum likelihood 里已经表示，但是这里后项则依赖于 x，表示 uncertainty of w。这样画图时，不同 x 对应的红色可能宽度就会不同）：

$$p(t|x,\mathbf{X},\mathbf{t}) = \mathcal{N}(t|m(x),s^2(x))$$

$$m(x) = \beta\phi(x)^T\mathbf{S}\sum_{n=1}^{N}\phi(\mathbf{x}_n)t_n$$

$$s(x)^2 = \beta^{-1} + \phi(x)^T\mathbf{S}\phi(x)$$

$$\mathbf{S}^{-1} = \alpha\mathbf{I} + \beta\sum_{n=1}^{N}\phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^T$$

Chapter3linear-regression

- Under Squared loss, the optimal regression function is the mean E [t|**x**] of the posterior p(t|**x**)；高斯的情况下 y(x)就求为 mean μ，所以也叫 mean prediction
- Squared loss is poor choice when conditional distribution p(t|x) is multimodal
- 用另一种推导：

$$\begin{aligned}\{y(\mathbf{x})-t\}^2 &= \{y(\mathbf{x})-\mathbb{E}[t|\mathbf{x}]+\mathbb{E}[t|\mathbf{x}]-t\}^2\\ &= \{y(\mathbf{x})-\mathbb{E}[t|\mathbf{x}]\}^2+\{\mathbb{E}[t|\mathbf{x}]-t\}^2\\ &\quad +2\{y(\mathbf{x})-\mathbb{E}[t|\mathbf{x}]\}\{\mathbb{E}[t|\mathbf{x}]-t\}\end{aligned}$$

代入回去：

$$\mathbb{E}[L] = \int \underbrace{\{y(\mathbf{x})-\mathbb{E}[t|\mathbf{x}]\}^2}\, p(\mathbf{x})\,\mathrm{d}\mathbf{x} + \int \underbrace{\mathrm{var}\,[t|\mathbf{x}]}\, p(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

前项体现了 mean，后项是 intrinsic 的 variability of target data，不可消除

- Minkowski loss（是 square loss 的简单拓展）：

$$L(t,y(\mathbf{x})) = |y(\mathbf{x})-t|^q$$

Conditional mean: q=2; conditional median: q=1; conditional mode: q=0

- $y(x,w)=w^T\Phi(x)$

Basis functions:

1. linear :$\Phi_d(x)=x_d$
2. Polynomial: $\Phi_j(x)=x^j$
3. Gaussian: $$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$
4. Sigmoid: $$\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)\sigma(a) = \frac{1}{1+\exp(-a)}$$

$S'(t) = S(t)(1 - S(t))$(这是为啥 sigmoid 这么受欢迎的原因，直接求导即可)

(p is global, g&s are local)

- Multiple outputs: i.e. T becomes matrix

$$\mathbf{W}_{\mathrm{ML}} = \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{T}.$$

但其实计算左侧这个Φ^{+}还是只需要一次，公用的

- Least-mean-squares(LMS) algorithm: stochastic gradient descent（拿 Error 跟 w 求导来算）

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_n$$

- Regularizer：q=1: Lasso; q=2: Ridge; q=0: Best subset selection
- Lagrange multiplier: $\mathcal{L}(x, y, \lambda) = f(x, y) + \lambda \cdot \Big(g(x, y) - c\Big)$

拉格朗日乘数法所得的极点会包含原问题的所有极值点，但并不保证每个极值点都是原问题的极值点。这里后项可以加，也可以减。

- Regularized least square: sparsity for q<=1: minimizations tend to set many coefficients to 0
- Regularizer 实际上使得求解期望不再等于原本的参数期望，所以是 bias-variance 的 trade off
- Lasso 不像 Ridge 那样有 closed form solution。而且还实现困难，要 quadratic programming。但优点是使参数为 0，perform model selection。
- Lasso (q = 1) is the only norm that provides sparsity and convexity
- Non-convexity makes the optimization problem more difficult
- Ridge regression is also the posterior mean, but the lasso and best subset selection are not.
- L2 regulatization with Gaussian prior, and L1 with Laplace centered on zero; http://stats.stackexchange.com/questions/177210/why-is-laplace-prior-producing-sparse-solutions http://stats.stackexchange.com/questions/182098/why-is-lasso-penalty-equivalent-to-the-double-exponential-laplace-prior

Chapter4gaussian-process

- Kernel: $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{x}')$
- Kernel a.k.a. covariance function
- Kernel function is symmetric
- Linear kernel:$k(x,x')=x^{T}x'$
- Kernel Ridge Regression:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{\mathbf{w}^T\phi(\mathbf{x}_n) - t_n\}^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

对 ridge regression 不完全整理，

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^T\mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^T\mathbf{K}\mathbf{a}$$

求出 a，就有如下右部

$$y(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \mathbf{a}^T\Phi\phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}$$

- Kernel

1. pros: 可能有更快的 kernel 算法，比单个算 basis function 再点乘快；更灵活，可以不知道 basis function；可以应对高维
2. Cons：存储空间 NM->N^2; compute matrix inverse of larger, M^2->N^2

- **Take *any* set $\mathcal{X}$ and *any* function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.**
  If k is a positive definite kernel, then we can use k to learn a (soft) maximum-margin classifier for the elements in $\mathcal{X}$!
- To kernelize an algorithm: write the algorithm only in terms of inner products; replace all inner products by kernel function evaluations (not guarantee for better performance, choice of k and model selection are important)
- Linear regression model: $y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\phi(\mathbf{x})$ 然后我们把 w 看成 Gaussian 分布的：

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$$

那么 y 就是 Gaussian 分布的线性，也是 Gaussian。（Gaussian process 视角则是直接给 function y 一个 prior，而非给 w 一个 prior,下面就是把 linear regression 处理成 gaussian process 的例子）

$$\mathbb{E}[\mathbf{y}] = \Phi\mathbb{E}[\mathbf{w}] = \mathbf{0}$$

$$\mathrm{cov}[\mathbf{y}] = \mathbb{E}[\mathbf{y}\mathbf{y}^T] = \Phi\mathbb{E}[\mathbf{w}\mathbf{w}^T]\Phi^T = \frac{1}{\alpha}\Phi\Phi^T = \mathbf{K}$$

- A Gaussian process is a collection of random variables any finite number of which has a joint Gaussian distribution
- Gaussian process

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

is completely defined by:

1. Mean function:

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$$

2. Covariance function:

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]$$

- Consistency Requirement or Marginalization Property: examination of a larger set of variables does not change the distribution of a smaller set.（这是 Gaussian process 的特点）

$(y_1, y_2) \sim \mathcal{N}(\mu, \Sigma)$ $\quad \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$ => $y_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$

- Squared exponential(SE)(又叫 radial basis function(RBF) or Gaussian):

$$\mathrm{cov}[f(\mathbf{x}_p), f(\mathbf{x}_q)] = k(\mathbf{x}_p, \mathbf{x}_q) = \exp\left\{-\frac{1}{2}|\mathbf{x}_p - \mathbf{x}_q|^2\right\}$$

我的理解是，选择了 SE 这种协方差，计算变量 xp，xq 的协方差，画出灰色区域，就是有点就细窄，没点就 variance 为 1 的图。

1. We choose a number of input points X_*
2. We write the corresponding covariance matrix (e.g. using SE) element-wise:

$$K(X_\star, X_\star)$$

3. Then we generate a random Gaussian vector with this covariance matrix:

$$f_\star \sim \mathcal{N}(\mathbf{0}, K(X_\star, X_\star))$$

- Gaussian process with noisy(noise-free 就没有红色部分):

$$\bar{\mathbf{f}}_\star = K(X_\star, X)\left(K(X, X)+\sigma_n^2 I\right)^{-1}\mathbf{t}$$

$$\mathrm{cov}[\mathbf{f}_\star] = K(X_\star, X_\star) - K(X_\star, X)\left(K(X, X)+\sigma_n^2 I\right)^{-1}K(X, X_\star)$$

Explanation: the predictive distribution is a Gaussian whose mean and variance depend on the test points X. and on the kernel k(x,x'), evaluated on the training data X.

如果看图呢，有 noise，在数据点也是有点宽度的，无 noise，就是一个点。

- Gaussian process vs Basis function model:

Train: O(N^3)(就是上面 kernel 矩阵的求逆); Test: O(N^2)　　Train: O(M^3); Test: O(M^2)

- GP infeasible for large training sets while basis function model is more efficient; however, GP can model covariance functions that can only be expressed by an infinite number of basis functions.
- $$k_y(\mathbf{x}_p, \mathbf{x}_q) = \sigma_f^2 \exp\left\{-\frac{(\mathbf{x}_p - \mathbf{x}_q)^2}{2 \cdot l^2}\right\} + \sigma_n^2 \delta_{pq}$$

SE covariance with hyperparameters is above: signal variance δ^2_f;range of neighbor influence(length scale) l; observation noise δ^2_n ；其中 l 越大，函数越平滑

- SVM 用到了 kernel

Chapter5gaussian-process

- GP regression algorithm（gpml P37, f|X :=N(0,K); y|f:=N(f, $\sigma_n^2 I$)。此外下面式子中 t 改成 y 才对）：

input: X (inputs), $\mathbf{y}$ (targets), k (covariance function), σ_n^2 (noise level), $\mathbf{x}_*$ (test input)

$L := \text{cholesky}(K + \sigma_n^2 I)$

$\boldsymbol{\alpha} := L^\top \backslash (L \backslash \mathbf{y})$

$\bar{f}_* := \mathbf{k}_*^\top \boldsymbol{\alpha}$ } predictive mean eq. (2.25)

$\mathbf{v} := L \backslash \mathbf{k}_*$

$\mathbb{V}[f_*] := k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\top \mathbf{v}$ } predictive variance eq. (2.26)

$\log p(\mathbf{y}|X) := -\frac{1}{2}\mathbf{y}^\top \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2}\log 2\pi$ eq. (2.30)

return: $\bar{f}_*$ (mean), $\mathbb{V}[f_*]$ (variance), $\log p(\mathbf{y}|X)$ (log marginal likelihood)

$$\begin{aligned} \bar{f}_\star &= \mathbf{k}_\star^T \left(K + \sigma_n^2 I\right)^{-1} \mathbf{t} \\ \operatorname{cov}[f_\star] &= k(\mathbf{x}_\star, \mathbf{x}_\star) - \mathbf{k}_\star^T \left(K + \sigma_n^2 I\right)^{-1} \mathbf{k}_\star \\ \log p(\mathbf{t}|X) &= -\frac{1}{2}\mathbf{t}^T \left(K + \sigma_n^2 I\right)^{-1} \mathbf{t} - \frac{1}{2}\log |K + \sigma_n^2 I| - \frac{N}{2}\log 2\pi \end{aligned}$$

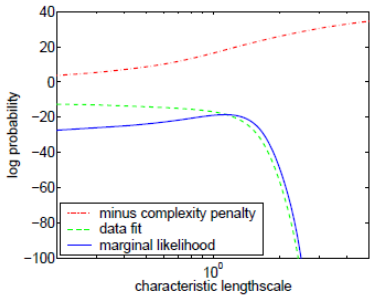
- **Bayesian model selection:**

1. Lowest level: w, e.g. parameters of a linear model;
2. mid-level: θ, e.g. controlling prior distribution of w;
3. top level: typically discrete set of model structures Hi

$$p(\mathbf{w}|\mathbf{y}, X, \boldsymbol{\theta}, \mathcal{H}_i) = \frac{p(\mathbf{y}|X, \mathbf{w}, \mathcal{H}_i) p(\mathbf{w}|\boldsymbol{\theta}, \mathcal{H}_i)}{p(\mathbf{y}|X, \boldsymbol{\theta}, \mathcal{H}_i)} \quad p(\mathbf{y}|X, \boldsymbol{\theta}, \mathcal{H}_i) = \int p(\mathbf{y}|X, \mathbf{w}, \mathcal{H}_i) p(\mathbf{w}|\boldsymbol{\theta}, \mathcal{H}_i)\, d\mathbf{w}$$

上面的 denominator 也叫做 marginal likelihood（每一层都是如此，每一层都 model selection）

- To learn different parameters of Gaussian Process, inference one level at a time.
- It automatically incorporates a trade-off between the model fit and the model complexity
- This model selection require evaluation of several integrals, which may or may not be analytically tractable, depending on details of the models.
- However, GP regression models with Gaussian noise are exception: integrals are tractable and the models are flexible.



Chapter6approximate-inference

- Approximation schemes:

1. Variational methods: never generate exact results, but scalable
2. Sampling methods: can generate exact results, but computational demanding, limited to small-scale(limit time and computational resources)

- Sampling:

$$\mathbb{E}[f] = \int f(\mathbf{z}) p(\mathbf{z}) d\mathbf{z}$$

$$\hat{f} = \frac{1}{L}\sum_{l=1}^{L} f(\mathbf{z}^l)$$ (上个是目标，下面就是 sample 方法)

- Sampling challenges:
  1. Might not be independent
  2. F(z) small in region p(z) large and vice versa=> need large sample sizes
- Transformation method: 通过 pdf 计算 cdf，然后求逆

$$F(x) = \int_{-\infty}^{x} p(z)dz$$

$$u \sim Uniform(0,1) \Rightarrow F^{-1}(u) \sim p(x)$$

- Exponential distribution:

  $$p(y) = \lambda \exp(-\lambda y)$$

  Cdf:

  $$h(y) = 1 - \exp(-\lambda y)$$

  Inverse of integral:

  $$y = h(y)^{-1} = -\lambda^{-1}\ln(1-z)$$

- Cauchy distribution: https://en.wikipedia.org/wiki/Cauchy_distribution

$$p(y) = \frac{1}{\pi}\frac{1}{1+y^2}$$

Cumulative distribution function:

$$F(x;0,1) = \frac{1}{\pi}\arctan(x) + \frac{1}{2}$$

Inverse of integral: tan

- Box-Muller algorithm(Gaussian is not possible by transformation):
  1. Generate pairs of uniformly distributed random numbers z1,z2∈(-1,1)
  2. Discard each pair unless it satisfies $r^2=z1^2+z2^2<=1$
  3. This leads to a uniform distribution of points inside the unit circle with $p(z1,z2)=1/\pi$
  4. For each pair z1,z2, evaluate

$$y_1 = z_1\left(\frac{-2\ln r^2}{r^2}\right)^{1/2} \qquad y_2 = z_2\left(\frac{-2\ln r^2}{r^2}\right)^{1/2}$$

$$\begin{aligned} p(y_1,y_2) &= p(z_1,z_2)\left|\frac{\partial(z_1,z_2)}{\partial(y_1,y_2)}\right| \\ &= \left[\frac{1}{\sqrt{2\pi}}\exp(-y_1^2/2)\right]\left[\frac{1}{\sqrt{2\pi}}\exp(-y_2^2/2)\right] \end{aligned}$$

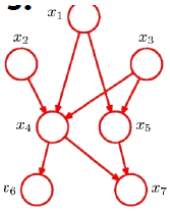
  5. Y1,y2 are independent Gaussian N(0,1)

6. If $y \sim \mathcal{N}(0,1)$, then $\sigma y + \mu \sim \mathcal{N}(\mu, \sigma^2)$

- Ancestral sampling:

  Principle: joint probability factorizes into conditional probabilities

  $$p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k | \mathrm{pa}_k)$$

  1. Start with lowest-numbered node and draw a sample form its distribution
  2. Cycle through each of the nodes in order and draw samples from the conditional distribution (where the parent variable is set to its sampled value

  $$\hat{x}_n \sim p(x_n | \mathrm{pa}_n)$$

- Logic sampling:

  Extension of ancestral sampling with some nodes are instantiated with observed values

  Don't agree, then whole sample is discarded

  Probability of accepting a sample decreases rapidly as the number of observed variables increases=> not practical

- Rejection sampling:
  1. Generate a number z0 from q(z).
  2. Generate a number u0 from the uniform distribution over [0,kq(z0)].
  3. If u0>p~(z0) reject sample, otherwise accept

  Limitation: high-dimensional spaces

  $$k = (\sigma_q / \sigma_p)^D$$

  D 很大则 k 很大，则接受概率很小

- Importance sampling:

  Method approximates expectations directly (but does not enable to draw samples from p(z) directly，这和 rejection sampling 是不一样的)

  感觉就是 p(z)不知道，但是知道 p~(z)，拿一个好取样的相似函数 q~(z)，以此把积分变成有限加法。之所以如此，是为了减轻 sample 量，而 uniform 的 grid sampling 就是特殊情况，因为和原函数不像，所以效果不好。

  $$\mathbb{E}[f] = \int f(\mathbf{z}) p(\mathbf{z}) d\mathbf{z} = \frac{Z_q}{Z_p} \int f(\mathbf{z}) \frac{\tilde{p}(\mathbf{z})}{\tilde{q}(\mathbf{z})} q(\mathbf{z}) d\mathbf{z} \simeq \frac{Z_q}{Z_p} \frac{1}{L} \sum_{l=1}^{L} \tilde{r}_l f(\mathbf{z}^{(l)})$$

  $$\tilde{r}_l = \frac{\tilde{p}(\mathbf{z}^{(l)})}{\tilde{q}(\mathbf{z}^{(l)})}$$

  之后估计常数 ratio

$$\frac{Z_p}{Z_q} = \frac{1}{Z_q}\int \tilde{p}(\mathbf{z})d\mathbf{z} = \int \frac{\tilde{p}(\mathbf{z}^{(l)})}{\tilde{q}(\mathbf{z}^{(l)})}q(\mathbf{z})d\mathbf{z} \simeq \frac{1}{L}\sum_{l=1}^{L}\tilde{r}_l$$

$$\mathbb{E}[f] \simeq \sum_{l=1}^{L} w_l f(\mathbf{z}^{(l)})$$

$$w_l = \frac{\tilde{r}_l}{\sum_m \tilde{r}_m} = \frac{\frac{\tilde{p}(\mathbf{z}^{(l)})}{\tilde{q}(\mathbf{z}^{(l)})}}{\sum_m \frac{\tilde{p}(\mathbf{z}^{(m)})}{\tilde{q}(\mathbf{z}^{(m)})}}$$

Importance sampling depends crucially on how well the two distribution match

If none of the samples falls in regions where p(z)f(z) is large:

1. The results may be arbitrary in error;
2. No diagnostic indication(no large variance in r_l)

So q(z) should not be small or zero in regions where p(z) is significant

Chapter7approximate-inference

- Sampling-Importance-Resampling(SIR)

  Draw L samples $\mathbf{z}^{(1)},\ldots,\mathbf{z}^{(L)}$ from $q(\mathbf{z})$.

  Construct weights using importance weighting

  $$w_l = \frac{\tilde{r}_l}{\sum_m \tilde{r}_m} = \frac{\frac{\tilde{p}(\mathbf{z}^{(l)})}{\tilde{q}(\mathbf{z}^{(l)})}}{\sum_m \frac{\tilde{p}(\mathbf{z}^{(m)})}{\tilde{q}(\mathbf{z}^{(m)})}}$$

  and draw a second set of samples $\mathbf{z}^{(1)},\ldots,\mathbf{z}^{(L)}$ with probabilities given by the weights $w^{(1)},\ldots,w^{(L)}$.

  The resulting L samples are only approximately distributed according to p(z), but the distribution becomes correct in the limit L->∞

- Both rejection sampling and importance sampling scale badly with high dimensionality

  $$p(\mathbf{z}) \sim \mathcal{N}(0, I), \qquad q(\mathbf{z}) \sim \mathcal{N}(0, \sigma^2 I)$$

  Rejection sampling: fraction of proposals accepted: σ^{-D}

  Importance sampling: variance of importance weights: $\left(\frac{\sigma^2}{2-1/\sigma^2}\right)^{D/2} - 1$
- Rejection sampling, importance sampling and SIR: based on independent samples/ sucks in high-dimension
- Metropolis algorithm:

  Proposal distribution is symmetric: $q(\mathbf{z}_A|\mathbf{z}_B) = q(\mathbf{z}_B|\mathbf{z}_A)$

$$A(\mathbf{z}^\star, \mathbf{z}^{(\tau)}) = \min\left(1, \frac{\tilde{p}(\mathbf{z}^\star)}{\tilde{p}(\mathbf{z}^{(\tau)})}\right)$$

From z^γ -> z*，实现就是 random u uniformly from (0,1), accept sample if $A(z^*,z^\gamma)>u$,否则保持原样（不舍弃），就会同一个 sample 出现很多次

Property: if $q(z_A|z_B)>0$ for all z, the distribution of z^γ tends to p(z) as $\gamma \to \infty$

The samples are highly correlated, we can obtain (largely) independent samples by retaining every M^{th} sample

- Markov chains properties:

  First-order(应该是指前一项有影响，此外注意这里是 p，即 $z^{(m)}$->$z^{(m+1)}$的概率，和 q 不一样) Markov chain:

$$p\left(\mathbf{z}^{(m+1)}|\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\right) = p\left(\mathbf{z}^{(m+1)}|\mathbf{z}^{(m)}\right)$$

  Marginal probability:

$$p\left(\mathbf{z}^{(m+1)}\right) = \sum_{\mathbf{z}^{(m)}} p\left(\mathbf{z}^{(m+1)}|\mathbf{z}^{(m)}\right) p\left(\mathbf{z}^{(m)}\right)$$

A Markov chain is called ***homogeneous*** if the transition probabilities $p(z^{(m+1)} \mid z^{(m)})$ are the same for all m.

A distribution is said to be ***invariant*** (or ***stationary***) w.r.t. a Markov chain if each step in the chain leaves that distribution invariant

Transition probabilities(定义):

$$T\left(\mathbf{z}^{(m)}, \mathbf{z}^{(m+1)}\right) = p\left(\mathbf{z}^{(m+1)}|\mathbf{z}^{(m)}\right)$$

For homogeneous Markov chain, distribution p*(z) is invariant if:

$$p^\star(\mathbf{z}) = \sum_{\mathbf{z}'} T\left(\mathbf{z}', \mathbf{z}\right) p^\star(\mathbf{z}')$$

a given Markov chain may have more than one invariant distribution(如果 transition probabilities are identity transformation,那么任何 distribution 都是 invariant)

Detailed balance is a sufficient (but not necessary) condition to ensure that a distribution is invariant: $p^\star(\mathbf{z})T\left(\mathbf{z}, \mathbf{z}'\right) = p^\star(\mathbf{z}')T\left(\mathbf{z}', \mathbf{z}\right)$

充分性证明：$$\sum_{\mathbf{z}'} p^\star(\mathbf{z}')T(\mathbf{z}', \mathbf{z}) = \sum_{\mathbf{z}'} p^\star(\mathbf{z})T(\mathbf{z}, \mathbf{z}') = p^\star(\mathbf{z}) \sum_{\mathbf{z}'} p(\mathbf{z}'|\mathbf{z}) = p^\star(\mathbf{z}).$$

(感觉这一部分得这么想，invariant 就是最后的收敛的分布，而 detailed balance 就是此分布的一个小特点，即互相转换相等；感觉 homogeneous 指每一次乘转换矩阵 T，转换矩阵本身保持不变)

Ergodicity(之前是最后结果的特点，这个是保证收敛出结果的 condition)

Mixture distributions:

$$T(\mathbf{z}', \mathbf{z}) = \sum_{k=1}^{K} \alpha_k B_k(\mathbf{z}', \mathbf{z})$$

$$\alpha_k \geq 0 \text{ and } \sum_k \alpha_k = 1$$

这个感觉就是把转换概率拆分一下，比如 each base transition only change a subset of variables

特点是 base 都 invariant 或都 detailed balance，那么合起来仍然是

- Metropolis-Hastings algorithm:

  A generalization to Metropolis: proposal distribution not required to be symmetric

$$A(\mathbf{z}^\star, \mathbf{z}^{(\tau)}) = \min\left(1, \frac{\tilde{p}(\mathbf{z}^\star) q_k(\mathbf{z}^{(\tau)}|\mathbf{z}^\star)}{\tilde{p}(\mathbf{z}^{(\tau)}) q_k(\mathbf{z}^\star|\mathbf{z}^{(\tau)})}\right)$$

K labels the members of the set of possible transitions considered;就是把之前的扩展，增加了 qk 的乘项，看 Gibbs 里面 $p(z_k|z_{\setminus k})$

证明 invariant：

$$A(\mathbf{z}', \mathbf{z}) = \min\left\{1, \frac{\tilde{p}(\mathbf{z}') q_k(\mathbf{z}|\mathbf{z}')}{\tilde{p}(\mathbf{z}) q_k(\mathbf{z}'|\mathbf{z})}\right\}$$

$$\begin{aligned}
\tilde{p}(\mathbf{z}) q_k(\mathbf{z}'|\mathbf{z}) A_k(\mathbf{z}', \mathbf{z}) &= \min\{\tilde{p}(\mathbf{z}) q_k(\mathbf{z}'|\mathbf{z}), \tilde{p}(\mathbf{z}') q_k(\mathbf{z}|\mathbf{z}')\} \\
&= \min\{\tilde{p}(\mathbf{z}') q_k(\mathbf{z}|\mathbf{z}'), \tilde{p}(\mathbf{z}) q_k(\mathbf{z}'|\mathbf{z})\} \\
\tilde{p}(\mathbf{z}) q_k(\mathbf{z}'|\mathbf{z}) A_k(\mathbf{z}', \mathbf{z}) &= \tilde{p}(\mathbf{z}') q_k(\mathbf{z}|\mathbf{z}') A_k(\mathbf{z}, \mathbf{z}') \\
\tilde{p}(\mathbf{z}) T(\mathbf{z}', \mathbf{z}) &= \tilde{p}(\mathbf{z}') T(\mathbf{z}, \mathbf{z}')
\end{aligned}$$

- Random walk: after N steps, distance on average proportional to √N; central goal in MCMC is to avoid this property
- Example: a common choice of proposal distribution is a Gaussian centered on the current state, and the variance of the proposal should be the same order as the smallest length scale σ_{min}(larger rejection high, smaller walk long).
  1. Number of steps to arrive at state independent of origin is $O((\sigma_{max}/\sigma_{min})^2)$
  2. Strong correlations(max/min 差别就会很大) can slow down the Metropolis(-Hasting) algorithm
- Gibbs sampling

  A special case of Metropolis-Hasting

  Idea: update one coordinate at a time by replacing z_i by a value drawn from $p(z_i|\mathbf{z}_{\setminus i})$; cycling through all variables or choosing the next(下面就是 1,2,3 顺序)

$$z_2^{(\tau+1)} \sim p(z_2|z_1^{(\tau+1)}, z_3^{(\tau)})$$

  The algorithm always accept!

$$A(\mathbf{z}^\star, \mathbf{z}) = \frac{p(\mathbf{z}^\star) q_k(\mathbf{z}|\mathbf{z}^\star)}{p(\mathbf{z}) q_k(\mathbf{z}^\star|\mathbf{z})} = \frac{p(z_k^\star|\mathbf{z}_{\setminus k}^\star) p(\mathbf{z}_{\setminus k}^\star) p(z_k|\mathbf{z}_{\setminus k}^\star)}{p(z_k|\mathbf{z}_{\setminus k}) p(\mathbf{z}_{\setminus k}) p(z_k^\star|\mathbf{z}_{\setminus k})} = 1$$

$$\mathbf{z}^*_{\setminus k} = \mathbf{z}_{\setminus k}$$

$$q_k(\mathbf{z}^*|\mathbf{z}) = p(z^*_k|\mathbf{z}_{\setminus k}) \text{ and } p(\mathbf{z}) = p(z_k|\mathbf{z}_{\setminus k})\, p(\mathbf{z}_{\setminus k})$$

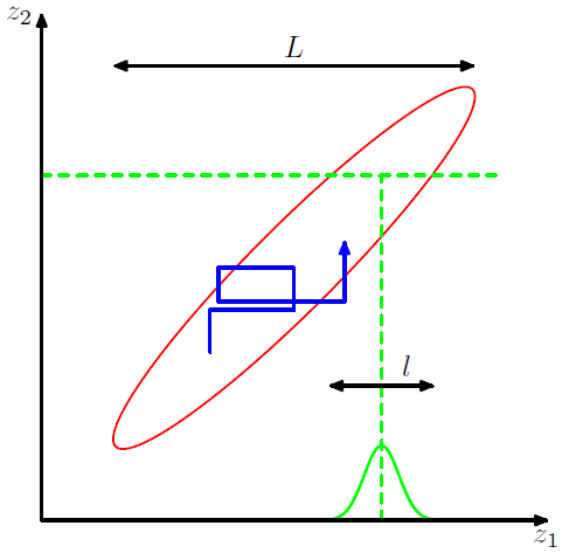
Benefits 是变量少，好算:

$$p(x_i|\mathbf{x}_{j\neq i}) = \frac{p(x_i, \mathbf{x}_{j\neq i})}{\sum_{x_i'} p(x_i', \mathbf{x}_{j\neq i})}$$

1. Discrete:
2. Continuous: often univariate

Graphical model 和 ancestral sampling 类似

Strong correlation also slow down Gibbs sampling($O(L/l)^2$)：



Burn-in: 扔掉开头还没进入 equilibrium 状态的 n 个 sample

但是我们不知道啥时候才是 long enough

解决 dependent: 1.thinning, keep only Mth;2.use Monte Carlo estimator on all samples

- Summary:

MCMC:

a) simple & effective
b) typically computational expensive
c) scales well with dimension

Gibbs:

a) Used in practice
b) Parameter free
c) Requires sampling conditional distributions

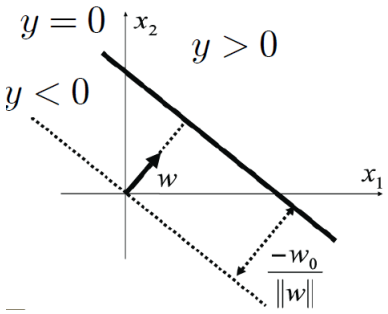
Chapter8linear-discriminants

- Linear discriminant function:

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0$$

w is called a weight vector, and w0 is a bias. The negative of the bias is called a threshold

注意，二维平面，则 w 有两项（未包括 w0）



- (w,w0)是 D 维超平面，而且过 D+1 维 expanded input space 原点
- 2-class 应用 sum-of-squares error 来求解：

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(\mathbf{w}^\top \mathbf{x}_n - t_n\right)^2$$

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \sum_{n=1}^{N}\left(\mathbf{w}^\top \mathbf{x}_n - t_n\right)\mathbf{x}_n = \mathbf{X}\mathbf{X}^\top\mathbf{w} - \mathbf{X}\mathbf{t} \stackrel{!}{=} 0$$

$$\mathbf{w} = \left(\mathbf{X}\mathbf{X}^\top\right)^{-1}\mathbf{X}\mathbf{t}$$

好处是 exact, closed-form solution

- K-class(output use 1-of-K notation):

$y_k(\mathbf{x}) = \mathbf{w}_k^{\mathrm{T}}\mathbf{x} + w_{k0}, \qquad k = 1, \ldots, K$ (然后 y 写成向量粗体，w 因为有转置，所以本身 w=(w1,w2,…wk))

$$E(\widetilde{\mathbf{W}}) = \sum_{n=1}^{N}\sum_{k=1}^{K}\left(y(\mathbf{x}_n;\mathbf{w}_k) - t_{kn}\right)^2$$
$$= \frac{1}{2}\mathrm{Tr}\left\{(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \mathbf{T})^\top(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \mathbf{T})\right\}$$

Tr is trace,就是方阵斜对角的和

$$\widetilde{\mathbf{W}} = \widetilde{\mathbf{X}}^\dagger\mathbf{T} = (\widetilde{\mathbf{X}}^\top\widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^\top\mathbf{T}$$

可以看出结果又是 exact, closed-form。结果转置后乘以 x 就是 y,即 discriminant function

- 上面 sum-of-square 对应 least-squares 方法，缺点是 sensitive to outliers & penalizes predictions that are “too correct”
- Generalization to linear model:
  1. Activation function:

$$y(\mathbf{x}) = g\left(\mathbf{w}^\top\mathbf{x} + w_0\right)$$

     a) If g is monotonous, the resulting decision boundaries are still linear functions of x
     b) Can bound the influence of outliers and “too correct” data points
     c) If using sigmoid for g(), we can interpret y(x) as posterior probabilities

  2. Basis function:

$$y_k(\mathbf{x}) = \sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}) + w_{k0}$$

Basis functions allow non-linear decision boundaries

But minimization no longer in closed form

- 因为上面，没有 closed-form，所以用 gradient descent 来 minimize iteratively towards ***local*** minimum(下面这种根据错误学习就是 delta rule)

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

分为两类:

Batch learning & sequential updating 区别就是 E(w)一个是所有的和，一个是单 sample

继续用 sum-of-squares error:

$$y_k(\mathbf{x}) = g(a_k) = g\left(\sum_{j=0}^{M} w_{ki}\phi_j(\mathbf{x}_n)\right)$$

$$\frac{\partial E_n(\mathbf{w})}{\partial w_{kj}} = \frac{\partial g(a_k)}{\partial w_{kj}} \left(y_k(\mathbf{x}_n; \mathbf{w}) - t_{kn}\right) \phi_j(\mathbf{x}_n)$$

- Generalized linear discriminants:

  Pros:

  a) g & fi allow us to address linearly non-separable problems

  b) simple sequential learning approach using gradient descent

  Cons:

  a) limited by curse of dimensionality(g & fi often introduce additional parameters)

  b) linearly separable case often overfitting

- Logistic regression: this is a model for classification rather than regression

$$p(\mathcal{C}_1|\boldsymbol{\phi}) = y(\boldsymbol{\phi}) = \sigma(\mathrm{w}^T\boldsymbol{\phi})$$

$$p(\mathcal{C}_2|\boldsymbol{\phi}) = 1 - p(\mathcal{C}_1|\boldsymbol{\phi})$$

Properties:

a) Focus on decision hyperplane

b) Advantageous for high-dimensional spaces, requires less parameters than explicitly modeling p(Φ|C_k) and p(C_k)

- Logistic sigmoid:

Definition: $$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

Logit function (inverse): $$a = \ln\left(\frac{\sigma}{1-\sigma}\right)$$

Symmetry property: $$\sigma(-a) = 1 - \sigma(a)$$

Derivative: $$\frac{d\sigma}{da} = \sigma(1-\sigma)$$

- Cross-entropy error function:

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n} \{1 - y_n\}^{1-t_n}$$

对上式求 negative log-likelihood(t_n 属于{0,1},而 y_n 属于(0,1))就得到了

$$\begin{aligned} E(\mathbf{w}) &= -\ln p(\mathbf{t}|\mathbf{w}) \\ &= -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \end{aligned}$$

- Gradient of cross-entropy error function（上式对 w 求偏导）:

$$\begin{aligned} y_n &= \sigma(\mathbf{w}^T \boldsymbol{\phi}_n) \\ \frac{dy_n}{d\mathbf{w}} &= y_n(1 - y_n)\boldsymbol{\phi}_n \end{aligned}$$

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\boldsymbol{\phi}_n$$

(求导时应用上面两式)

注意到 sigmoid with cross-entropy 和 linear regression with sum-of-squares 的 gradient 一样，都是如下：

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta(y_k(\mathbf{x}_n; \mathbf{w}) - t_{kn})\phi_j(\mathbf{x}_n)$$

不过用这种方法 sequential estimation is slow

- Newton-Paphson： $\mathbf{w}^{(\text{new})} = \mathbf{w}^{(\text{old})} - \mathbf{H}^{-1}\nabla E(\mathbf{w})$(H 是 E 对 w 的两次求导)

$$\begin{aligned} \mathbf{w}^{(\text{new})} &= \mathbf{w}^{(\text{old})} - (\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{R}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{y} - \mathbf{t}) \\ &= (\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{R}\boldsymbol{\Phi})^{-1}\{\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{R}\boldsymbol{\Phi}\mathbf{w}^{(\text{old})} - \boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{y} - \mathbf{t})\} \\ &= (\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{R}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{R}\mathbf{z} \end{aligned}$$

（这是对 sigmoid with cross-entropy）

- Newton-Raphson gives exact solution in one step for linear regression model(这里上面用的是 sum-of-squares error); but IRLS to logistic regression model with cross-entropy error（因为 R depends on w）; also IRLS to multiclass logistic regression with cross-entropy error
- Logistic regression property:
  a) Directly represent posterior distribution
  b) Requires fewer parameters than modeling likelihood + prior
  c) Cross-entropy error is concave: unique minimum, but no closed-form solution, iterative optimization(IRLS)
  d) Both online and batch optimization exist
  e) Tends to systematically overestimate odds ratios when sample size is small
- Softmax regression: (就是把 target{0,1}变成 1-of-K 的 K-class 扩展)

$$\frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

Error function:

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\left\{\mathbb{I}(t_n = k)\ln\frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{j=1}^{K}\exp(\mathbf{w}_j^\top \mathbf{x})}\right\}$$

$$\nabla_{\mathbf{w}_j} E(\mathbf{w}_1,\ldots,\mathbf{w}_K) = \sum_{n=1}^{N}(y_{nj} - t_{nj})\,\boldsymbol{\phi}_n$$

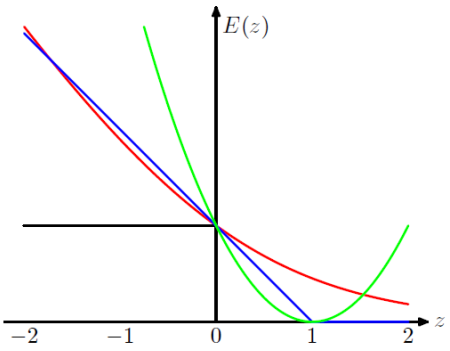
- Comparison of error functions:

a) Ideal misclassification error: ideal but not differentiable, and gradient is 0 for misclassified points

b) Squared error: closed-form solution; but sensitive to outliers and penalizes “too correct”

c) Cross-entropy error: concave, unique minimum exists, robust to outliers; but no closed-form solution, requires iterative estimation(rescaled by 1/ln(2)才过(0,1))

d) Hinge error: leads to sparse solutions, not sensitive to outliers; not differentiable around z=1(cannot be optimized directly)



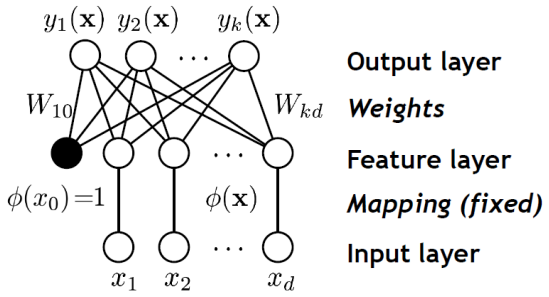
Chapter9neural-networks

- SVM:

$$\min_{\mathbf{w}\in\mathbb{R}^D} \underbrace{\frac{1}{2}\|\mathbf{w}\|^2}_{\text{L}_2\text{ regularizer}} + \underbrace{C\sum_{n=1}^{N}[1 - t_n y(\mathbf{x}_n)]_+}_{\text{“Hinge loss”}}$$

$[x]_+ := \max\{0,x\}.$

- Perceptron:



Linear outputs:

$$y_k(\mathbf{x}) = \sum_{i=0}^{d} W_{ki}\phi(x_i)$$

Logistic output:

$$y_k(\mathbf{x}) = \sigma\left(\sum_{i=0}^{d} W_{ki}\phi(x_i)\right)$$

Remarks:

1. Perceptrons are “generalized linear discriminants”
2. Feature functions Φ(x) are fixed, not learned

Error(tn 是{-1,1}，根据正负判断 2class):

$$E_{\mathrm{P}}(\mathbf{w}) = -\sum_{n\in\mathcal{M}} \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}_n t_n$$

Learning(学习参数可以设为 1，没错的 sample 不 update):

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_{\mathrm{P}}(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta\boldsymbol{\phi}_n t_n$$

Perceptron learning corresponds to 1st-order gradient descent of a quadratic error function

- Loss functions(注意 hinge loss 有个加号，就是和 0 比，取 max):

L2 loss ⇒ Least-squares regression

$$L(t, y(\mathbf{x})) = \sum_n (y(\mathbf{x}_n) - t_n)^2$$

L1 loss: ⇒ Median regression

$$L(t, y(\mathbf{x})) = \sum_n |y(\mathbf{x}_n) - t_n|$$

Cross-entropy loss ⇒ Logistic regression

$$L(t, y(\mathbf{x})) = -\sum_n \{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\}$$

Hinge loss ⇒ SVM classification

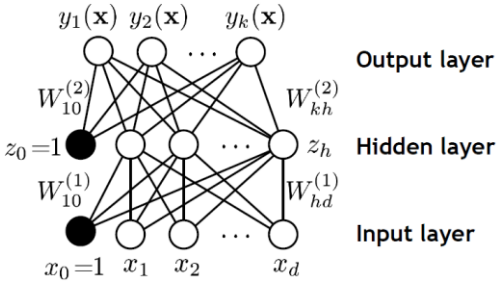
$$L(t, y(\mathbf{x})) = \sum_n [1 - t_n y(\mathbf{x}_n)]_+$$

Softmax loss ⇒ Multi-class probabilistic classification

$$L(t, y(\mathbf{x})) = -\sum_n \sum_k \left\{\mathbb{I}(t_n = k)\ln\frac{\exp(y_k(\mathbf{x}))}{\sum_j \exp(y_j(\mathbf{x}))}\right\}$$

39

- Can apply different regularization(and different error function) to perceptron: L1 for sparsity;L2(weight decay)
- Limitations of perceptron:
  Fixed, hand-coded input features; not flexible
- Multi-layer perceptrons(来解决上面的问题，学习 features):



$$y_k(\mathbf{x}) = g^{(2)}\left(\sum_{i=0}^{h} W_{ki}^{(2)} g^{(1)}\left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j\right)\right)$$

Activation functions $g^{(k)}$ (just for examples):

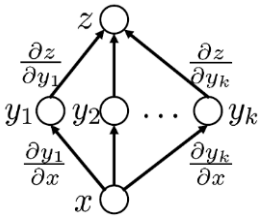
$$g^{(2)}(a) = \sigma(a),\ g^{(1)}(a) = a$$

(注意，虽然叫 multilayer perceptron，但用的是 sigmoid 这种,differentiable,而不是 step)

- Hidden layer can have arbitrary number of nodes, can be multiple layers
- 2-layer network (1 hidden layer, enough nodes) can approximate any continuous function of a compact domain arbitrarily well
- More linear units no help, fixed output non-linear not enough, so need multiple layers of adaptive non-linear hidden units

Chapter10backpropagation

- Gradient descent(多层神经网络，用此来更新 w):
  1. Naive analytical differentiation(就是对每个 w，展开所有路径硬算):



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x} + \cdots = \sum_{i=1}^{k} \frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial x}$$

Cons: Increasing depth, exponential paths, infeasible to compute

  2. Numerical differentiation

Make small changes to W and accept those improve E(W)

Cons: inefficient, several forward pass(run entire dataset) for each weight

  3. Backpropagation

Core steps:

a) Apply an input vector x_n to the network and forward propagate through the network using (5.48) and (5.49) to find the activations of all the hidden and output units

$$a_j = \sum_i w_{ji} z_i \quad \text{and} \quad z_j = h(a_j)$$

b) Evaluate the δ_k for all the output units using (5.54).

$$\delta_k = y_k - t_k$$ (k 层（输出层）的错误，通过直接和目标值 target 对比得到；可以换成别的，比如加个 sigmoid 啥的，就再多乘 y(1-y)即可)

c) Backpropagate the δ's using (5.56) to obtain δ_j for each hidden unit in the network(注意下式求和是对 k).

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$

d) Use (5.53) to evaluate the required derivatives(z_i是forward时已知的,dynamic programming).

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

(说明：i->j->k; aj 是输入的和，zj 是 h 后的输出)

Algorithm:

Forward Pass

$\mathbf{y}^{(0)} = \mathbf{x}$

for $k = 1, ..., l$ do

$\mathbf{z}^{(k)} = \mathbf{W}^{(k)}\mathbf{y}^{(k-1)}$

$\mathbf{y}^{(k)} = g_k(\mathbf{z}^{(k)})$

endfor

$\mathbf{y} = \mathbf{y}^{(l)}$

$E = L(\mathbf{t}, \mathbf{y}) + \lambda\Omega(\mathbf{W})$

Backward Pass

$\mathbf{h} \leftarrow \frac{\partial E}{\partial \mathbf{y}} = \frac{\partial}{\partial \mathbf{y}} L(\mathbf{t}, \mathbf{y}) + \lambda \frac{\partial}{\partial \mathbf{y}} \Omega$

for $k = l, l\text{-}1, ..., 1$ do

$\mathbf{h} \leftarrow \frac{\partial E}{\partial \mathbf{z}^{(k)}} = \mathbf{h} \odot g'(\mathbf{y}^{(k)})$

$\frac{\partial E}{\partial \mathbf{W}^{(k)}} = \mathbf{h}\mathbf{y}^{(k-1)\top} + \lambda \frac{\partial \Omega}{\partial \mathbf{W}^{(k)}}$

$\mathbf{h} \leftarrow \frac{\partial E}{\partial \mathbf{y}^{(k-1)}} = \mathbf{W}^{(k)\top}\mathbf{h}$

endfor

(圈点表示 element-wise product)

4. Automatic differentiation

a) Convert the network into a computational graph

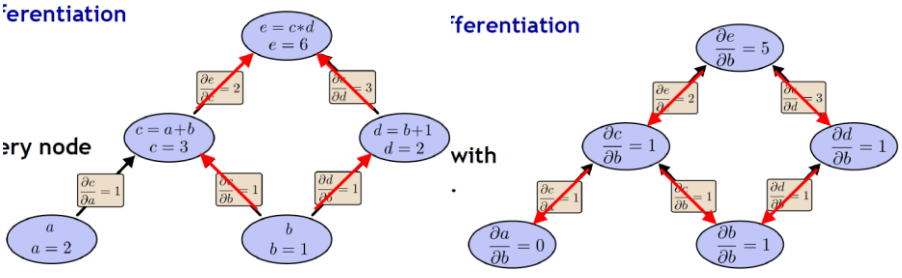
b) Each layer/module specify how it affects the forward and backward passes

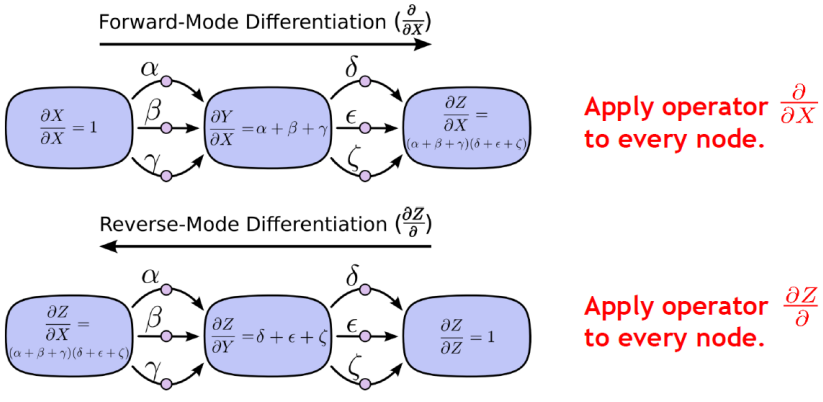
c) Apply reverse-mode differentiation

$$\mathbf{y} = \text{module.}\mathbf{fprop}(\mathbf{x})$$

$$\frac{\partial E}{\partial \mathbf{x}} = \text{module.}\mathbf{bprop}(\frac{\partial E}{\partial \mathbf{y}})$$

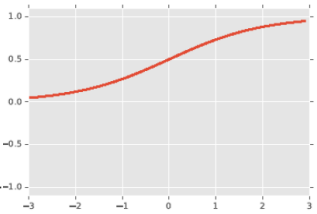
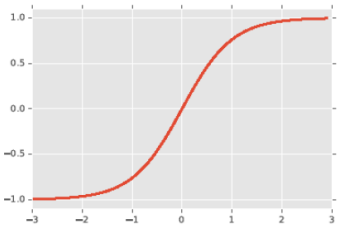
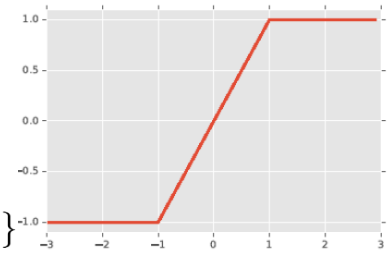
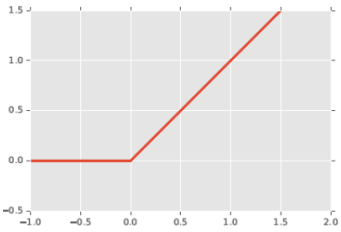
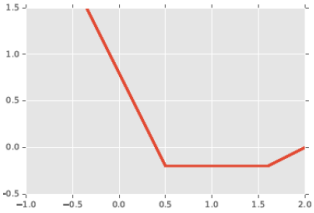
Computational graph(一个向上催债，一个向下发钱，都是O(edge)，但右边快O(input)):





(这是 deep learning libraries 的示例，x, y, and intermediate results are stored in the module)

- Common non-linearities

1. Sigmoid: $g(a) = \sigma(a) = \frac{1}{1+\exp\{-a\}}$ 
2. Hyperbolic tangent (tanh): $g(a) = tanh(a) = 2\sigma(2a) - 1$ 
3. Softmax: $g(\mathbf{a}) = \frac{\exp\{-a_i\}}{\sum_j \exp\{-a_j\}}$
4. Hard tanh: $g(a) = \max\{-1, \min\{1, a\}\}$ 
5. Rectified linear unit (ReLU): $g(a) = \max\{0, a\}$ 
6. Maxout: $g(\mathbf{a}) = \max_i \{\mathbf{w}_i^\top \mathbf{a} + b_i\}$ 

(Maxout 应该是什么凸函数都能模拟出来，比如 ReLU)

Comparison:

Output nodes:

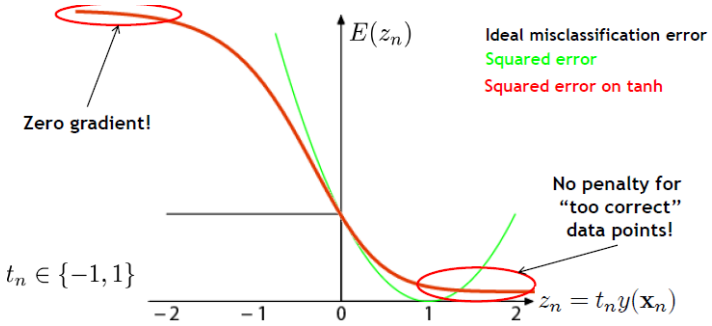
1. Sigmoid for probabilistic interpretation(range [0,1])
2. Tanh for regression task

Internal nodes:

1. Tanh better than sigmoid since it is centered

2. Tanh often implemented as piecewise linear function(similar to hard tanh and maxout)
3. Historically most use tanh, recently often use ReLU for classification tasks

- Do not use L2 loss with sigmoid/tanh output(use crosss-entropy instead)(原因是下面红色线条，在特别自信的错误处 gradient 是 0，没法改正)



- Implementing Softmax(standard for multi-class outputs):

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\left\{\mathbb{I}(t_n = k)\ln\frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{j=1}^{K}\exp(\mathbf{w}_j^\top \mathbf{x})}\right\}$$

1. Do not compute first softmax, then log. But instead directly evaluate log-exp in the denominator(这里指拆成 ln Exp – ln (Sum Exp)，用我下方的方法，可以进一步简化，就是 $e^{1001}=e^{1000}*e$)

$$\log\sum_{n=1}^{N}\exp\{x_n\} = a + \log\sum_{n=1}^{N}\exp\{x_n - a\}$$

3. softmax has the property that for a fixed vector b:
softmax(a+b)=softmax(a)
也就是说 subtract the largest weight vector wj from the others(感觉下面这个方法在上面的之前使用).

$$\begin{aligned} p(y^{(i)} = j|x^{(i)};\theta) &= \frac{e^{(\theta_j-\psi)^T x^{(i)}}}{\sum_{l=1}^{k} e^{(\theta_l-\psi)^T x^{(i)}}} \\ &= \frac{e^{\theta_j^T x^{(i)}} e^{-\psi^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}} e^{-\psi^T x^{(i)}}} \\ &= \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}. \end{aligned}$$

Chapter11optimization

- stochastic learning: choose single; estimate is noisy;
  pros:
  a) usually much faster;
  b) often better solution;
  c) can be used for tracking changes
- batch learning: process full;
  pros:
  a) conditions of convergence well understood;

b) many acceleration techniques(e.g. conjugate gradients) only in batch learning;

c) theoretical analysis of weight dynamics and convergence rates are simpler

- minibatches: process only a small batch of training examples together; **start with a small batch size and increase it as training proceeds**;

  pros:

  a) gradients more stable than stochastic, and faster to compute than batch;

  b) take advantages of redundancies in the training set

  c) matrix operations are more efficient than vector operations

  caveat:

  error function should be normalized by the minibatch size, s.t. we can use same learning rate between minibatches

$$E(\mathbf{W}) = \frac{1}{N}\sum_n L(t_n, y(\mathbf{x}_n; \mathbf{W})) + \frac{\lambda}{N}\Omega(\mathbf{W})$$

- shuffling:

  a) choose sample most unfamiliar(class A, B, A, B)

  b) present large-relative-error input more frequently(be careful with outliers)

  (stochastic and monibatch 用,batch 无所谓 shuffling)

- Data augmentation(apply all):

  a) Cropping(切割，图像一部分，等大)

  b) zooming

  c) flipping

  d) color PCA

  Pros:

  a) larger training set

  b) robustness against expected variations(reduce overfitting)

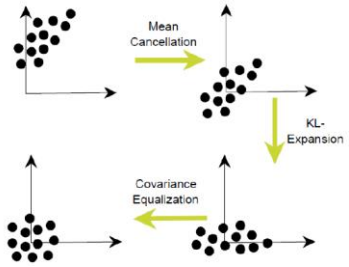
  When testing:

  a) Need to apply cropping again

  b) Beneficial to apply flipping

  c) Applying several ColorPCA get improvement, but increase runtime

- Normalizing the inputs(这条每一层都适用)



  a) Normalize all inputs units to 0-mean, unit covariance

  b) Decoorelate them using PCA(KL-transform)

- Choosing sigmoid:

  Symmetric sigmoids(e.g. tanh), converge faster than logistic sigmoid(0,1)

$$f(x) = 1.7159\tanh\left(\frac{2}{3}x\right)$$

  (variance of the outputs will be close to 1 because the effective gain of the sigmoid is roughly

1 over its useful range)

- Initializing weights

  Assuming: training set has been normalized; recommended sigmoid(above1.716) is used

  Initial weights should be drawn(uniform of normal) with 0-mean and variance

  $$\sigma_w^2 = \frac{1}{n_{in}}$$ (n_{in} is the fan-in, incoming connections of the node)

  variance of uniform distribution:

  $$\sigma^2 = \frac{1}{12}(b-a)^2$$

  Glorot(一种改进):

  $$\mathrm{Var}(Y) = \mathrm{Var}(W_1X_1 + W_2X_2 + \cdots + W_nX_n) = n\mathrm{Var}(W_i)\mathrm{Var}(X_i)$$

  令 Var(Y)==Var(X), n 既可以是 in，也可以是 out，所以妥协如下：

  $$\mathrm{Var}(W) = \frac{2}{n_{\mathrm{in}} + n_{\mathrm{out}}}$$

  $$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right]$$

  Extension to ReLU(max{0,a},上面是针对 tanh 这种对称的):

  $$\mathrm{Var}(W) = \frac{2}{n_{\mathrm{in}}}$$

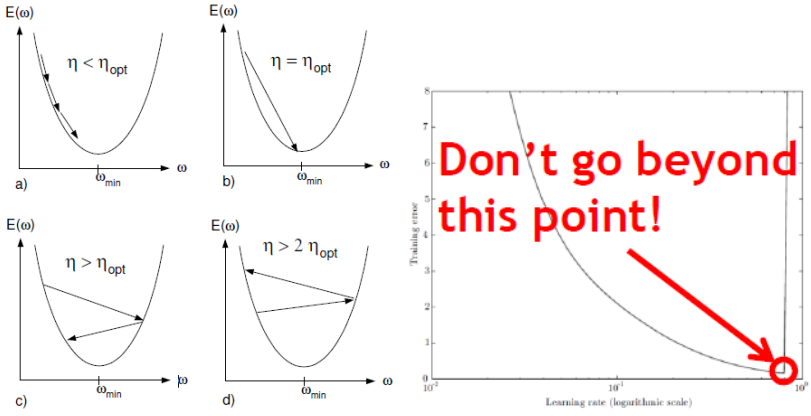
- 之前都是数据的预处理，初始化，怎么选择，下面是 gradient descent 怎么做能更好的 converge
- Choosing learning rate(一种分析):

  The learning rate of lower layer should be larger than that in higher layer(输出层)

  In one dimension and E is quadratic, optimal learning rate is inverse of Hessian:

  $$\eta_{\mathrm{opt}} = \left(\frac{\mathrm{d}^2 E(W^{(\tau)})}{\mathrm{d}W^2}\right)^{-1}$$

  (底下图是根据上面 1-d，quadratic Error 来的，前三能 converge)



- Momentum:

a) Dampen oscillations;

b) Build up speed(velocity) in directions with consistent gradient

(decay mu <1, dx 就是 x 处的 gradient descent)

```
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

mu 就是下面的 alpha

$$\mathbf{v}(\infty) = \frac{1}{1-\alpha}\left(-\varepsilon\frac{\partial E}{\partial \mathbf{w}}\right)$$

上面看到 alpha 接近 1 则我们速度可以非常大，converge 更快

开始小 alpha=0.5，之后增大到 0.9,0.99。

- Nesterov Momentum(就是看到哪，看那的 gradient；先 make mistake 再 correct):

```
x_ahead = x + mu * v
# evaluate dx_ahead (the gradient at x_ahead instead of at x)
v = mu * v - learning_rate * dx_ahead
x += v
```

a) first jump in the direction of previous accumulated gradient

b) then measure the gradient where you end up and make a correction

- 之前的 learning rate 也都是不变的，也是全局的，接下来看如何改 learning rate，上面的 momentum 啥的可以和下面 RMSprop 结合起来一起用
- Separate adaptive learning rates:

$$\Delta w_{ij} = -\varepsilon g_{ij}\frac{\partial E}{\partial w_{ij}}$$

$$\text{if } \left(\frac{\partial E}{\partial w_{ij}}(t)\frac{\partial E}{\partial w_{ij}}(t-1)\right) > 0$$

$$\text{then } g_{ij}(t) = g_{ij}(t-1) + 0.05$$

$$\text{else } \ g_{ij}(t) = g_{ij}(t-1) * 0.95$$

g_{ij} 就是 local gain, start with 1。一出现 oscillation, local gain decay rapidly

- RMS(root mean square):

$$x_{\text{rms}} = \sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \cdots + x_n^2\right)}.$$

- RMSProp(better than 之前的 separate adaptive learning rate):

$$MeanSq(w_{ij}, t) = 0.9MeanSq(w_{ij}, t-1) + 0.1\left(\frac{\partial E}{\partial w_{ij}}(t)\right)^2$$

Divide gradient by ***sqrt***(MeanSq(w_{ij},t))

Motivation:

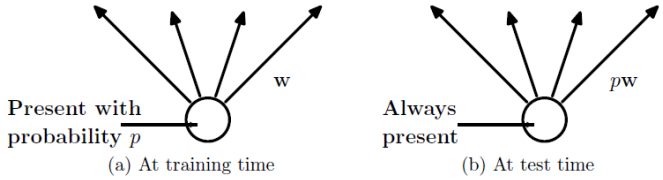
a) Magnitude of gradient can be very different for different weights and may change during learning

b) 之前那个只用了 sign of 梯度，适用于 batch。但是拓展到 minibatch，我们就得用 RMSProp，更好的应对上面一条。

- AdaGrad, AdaDelta, Adam(这个是目前主流)
- Saddle points dominate in high-dimensional spaces, so be patient
- Reducing learning rate(这是在之前那些弄完，最后训练收官 convergence 阶段):
  a) Reduce learning rate by factor of 10
  b) Continue training for a few epoch
  c) Do this 1-3 times, then stop
- Epoch vs iteration: if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch
- Batch normalization(improves convergence, more important for large dataset than dropout):

batch normalization happens to introduce some noise into the network, so it can regularize the model a little bit(所以 eliminates the need of dropout in some cases)
idea: introduce intermediate layer that center the activations of the previous layer per minibatch
i.e perform transformation on all activations and undo those transformations when backprop gradients

- Dropout(a regularization method):



(training 阶段 randomly 按 p 概率 switch off，backprop 也只针对 sub-network，但是 test 时所有都上，不过乘上 p 参数)

Chapter12optimization

The same

Chapter13cnn

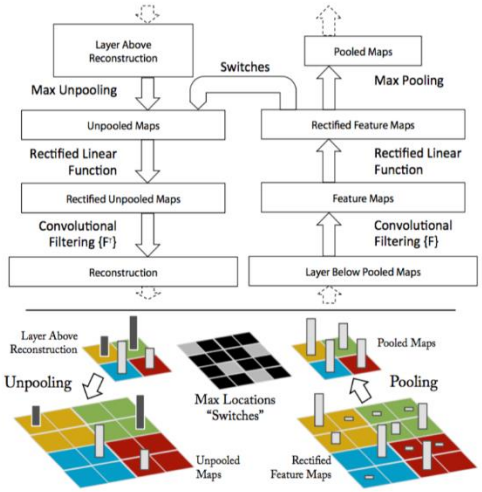
Cnn:

- CNN learns the values of these filters on its own during the training process (although we still need to specify parameters such as number of filters, filter size, architecture of the network etc. before the training process)
- All neural net activations arranged in 3 dimensions(width, height, depth)
- Convolution 之后要引入 non-linear. tanh or sigmoid can also be used instead of ReLU, but ReLU has been found to perform better in most situations
- Pooling 时没有重复的，convolution 时可以 stride 1,2,3..
- 可以 0-padding,或者没有 padding，就是 filter 不涉及方框外
- 最后 Fully connected layer is multi layer perceptron. Use softmax(好处是 sum of output probabilities==1)(or SVM) in the output layer
- Filter weights are shared between locations=> gradients are added for each filter location

- we can have multiple Convolution + ReLU operations in succession before a having a Pooling operation
- input 32*32*3, convolution filter 5*5*3,12 个，输出可以是 32*32*12（意思就是 filter 一定管所有 depth，但是输出仍然是 1 个）
- Prefer a stack of small filter CONV to one large receptive field CONV layer, 3 个 3*3 的 conv layer 叠在一起，第三层就有 7*7 的视野，但用 3*3*3 的参数，而不是 7*7。而且还有层间 non-linearities，more expressive。当然实际运用时也有缺点，就是 need more memory to hold all the intermediate CONV layer results

chapter14cnn

- visualizing ConvNet:



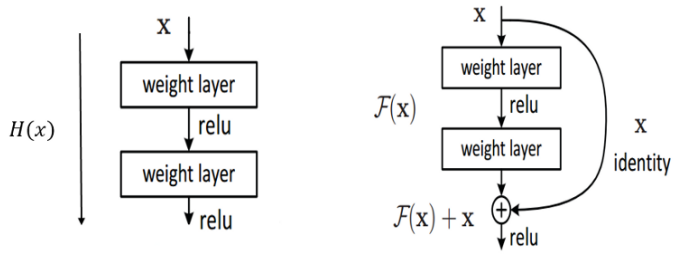
和下面的一样，把除了一个外的都设为 0，然后根据图反向传播到 pixel
左边叫 DeconvNet
Convolutional layer 的反：regular convolutional layer with its filters transposed
Max-pool 的反：record where maximums originated from in forward propagation, placed there(虽然把 2*2 都设成 max 也成)
ReLU 的反:直接过一遍 ReLU 即可
此外为了看是否正确获取特征，还是错误的提取了背景
Occlusion experiment:
Mask part of the image with an occluding square, monitor the output

- Inceptionism:

就是将输出类别设成 1，别的是 0，梯度反向传播，通过梯度上升法，不断修正图片。
生成某类：Start with an image full of random noise, then gradually tweak the image towards what net considers a banana. Impose a prior constraint on the image, e.g. neighboring pixels need to be correlated
自由脑洞：simply feed the network an arbitrary image or photo and let the network analyze the picture. We then pick a layer and ask the network to enhance whatever it detected. Each layer of the network deals with features at a different level of abstraction, so the complexity of features we generate depends on which layer we choose to enhance

chapter15cnn

- residual network(ResNet):



2 weight layers used to fit H(x), now fit F(x)

a) If identity is optimal, it is easy to set weights as 0;

b) If optimal mapping closer to identity, easier to find small fluctuations

c) Direct path for gradient to flow to the previous

Property:

a) Almost all 3*3 convolutions

b) Spatial size/2=>filters*2(same complexity per layer)

c) Batch normalization

- Applications of cnn

Transfer learning with cnn:

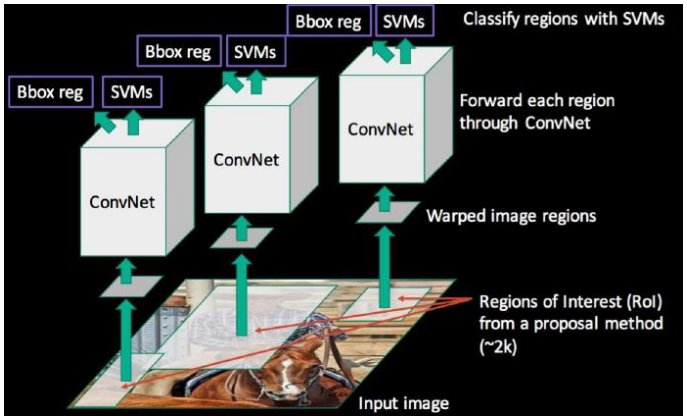
If small dataset: fix all weights, retrain classifier(最后的 FC 和 softmax)

If medium: use old weights as initialization, train full or some of higher layers(后几个 conv 以及 fc 以及 softmax)

R-cnn:

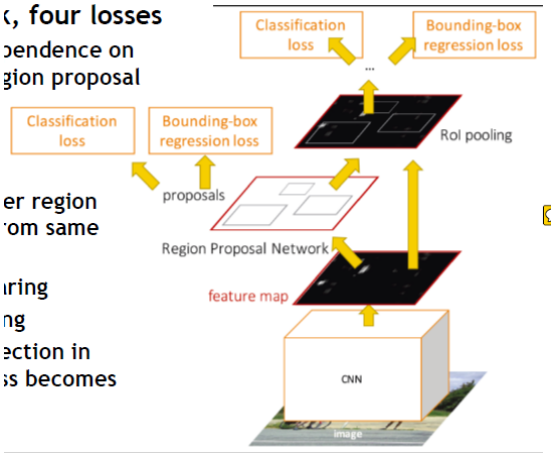
a) extract region proposals(selective search 这是单独的一个算法)

b) Use pre-trained&fine-tuned classification network as feature extractor on those regions(bounding box regression and SVMs)



faster r-cnn:

infer region proposals from same CNN, feature sharing, joint training, object detection in a single pass



四个 loss

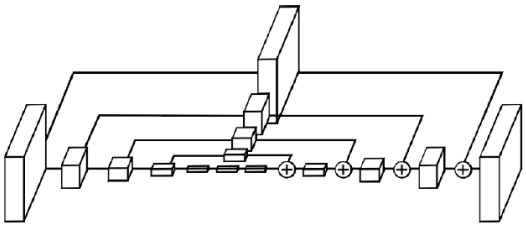
- Fully convolutional networks for semantic segmentation

All operations formulated as convolutions

Can process arbitrarily sized images

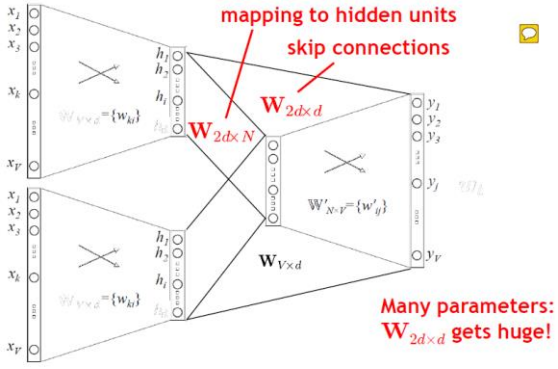
FCN can be think of a sliding-window classification, producing a heatmap of output scores for each class

FCN output has low resolution; perform upsampling to get back to desired resolution and use skip connections to preserve higher-resolution information（底下是 encoder-decoder architecture）



Chapter16word-embeddings

- N-gram method: count as probability;
  Problems: scalability; partial observability(并不是观察到 0，概率就是 0)
- Word embeddings are a successful applications of unsupervised learning. They don't require expensive annotation, but can be derived from large unannotated corpora that are readily available（问题是参数太多）



- word2vec and GloVe are geared towards producing word embeddings that encode general semantic relationships, which are beneficial to many downstream tasks(所以 word2vec 可以用 CBOW，用前几项，后几项，而且不关心顺序)
- word2vec

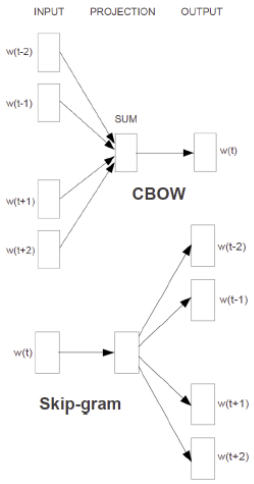
  CBOW:

  projection layer is shared for all words (not just the projection matrix); thus, all words get projected into the same position (their vectors are averaged).

  Skip gram:

  the output layer of the neural network is replicated multiple times; the error vectors from all output layers are summed up to adjust the weights via backpropagation

  for each training word we will select randomly a number R in range < 1;C >, and then use R words from history and R words from the future of the current word as correct labels.



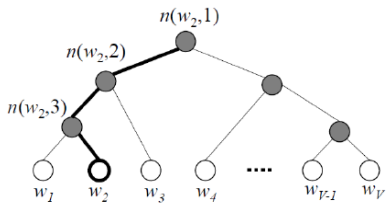
  Hierarchical softmax:

  The V words must be leaf units of the binary tree.

  $p(n, \text{left}) = \sigma\left(\mathbf{v}'^{T}_{n} \cdot \mathbf{h}\right)$ (向右就是 1-这个，这样的树最后的概率就是 normalized，和为 1)

  Output embedding for all internal nodes, |V|-1。和 vocabulary 差不多

  we are only able to obtain speed-up(log V 更新一条线) during training, when we know the word we want to predict (and consequently its path) in advance. During testing, when we need to find the most likely prediction, we still need to calculate the probability of all words



- Word2vec uses a single hidden layer, fully connected neural network as shown below. The neurons in the hidden layer are all linear neurons
- 就是 input 乘以矩阵到 hidden layer，然后乘以矩阵到 output，用 softmax 处理成和为 1，再 backprop
- Word2vec embedding can answer analogy questions; CBOW better for syntactic; skip-gram

better for semantic

- Siamese network

  contain two or more identical subnetworks; used to finding similarity or a relationship between two comparable things; shared part of parameters

  learn embedding network that preserves semantic similarity between inputs

- Triplet loss network

$$\|f(x_i^a) - f(x_i^p)\|_2^2 < \|f(x_i^a) - f(x_i^n)\|_2^2$$

  Use for face recognition; negative, anchor, positive

Chapter17rnn

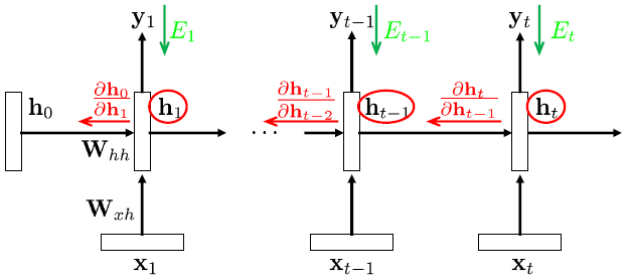
- convNet vs RNN: conv-nets take a fixed size input and generate fixed-size outputs. RNN, on the other hand, can handle arbitrary input/output lengths, but would typically require much more data because it is a more complex model.
- Rnn:

  x0(就是 hidden state0) is provided by the user, set to zero or learned

  we may not need inputs/outputs at each time step（即 1-many，many-1，many-many）

  LSTM and vanilla RNN 都是 RNN，vanilla 用 backpropagation through time，但 long-term dependencies 学不好,due to vanishing/exploding gradient。所以 LSTM，GRU 应运而生

  Unroll rnn, but remember weights for hidden layer are shared between temporal layers



Backpropagation through time:

$$\mathbf{h}_t = \sigma\left(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + b\right)$$

$$\hat{\mathbf{y}}_t = \operatorname{softmax}\left(\mathbf{W}_{hy}\mathbf{h}_t\right)$$

求 y 对 Why（V）的偏导（只涉及 E3,y3; z3=V*s3）：

$$\begin{aligned}\frac{\partial E_3}{\partial V} &= \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} \\ &= \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial V} \\ &= (\hat{y}_3 - y_3) \otimes s_3\end{aligned}$$

底下这个 w 是 h 之间的：

$$E = \sum_{1 \le t \le T} E_t$$

$$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \le k \le t} \left(\frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial w_{ij}} \right)$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{t \ge i > k} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{t \ge i > k} \mathbf{W}_{hh}^\top diag\left(\sigma'(\mathbf{h}_{i-1})\right)$$

Error 就是各时间都加一起

二式的加号意思是 immediate derivative，with h_{k-1} as constant

三式 diag converts a vector into a diagonal matrix, and σ' computes element-wise the derivative of σ

- Gradient vanishing/exploding:

  Exploding less notorious: exploding gradients are obvious. Your gradients will become NaN (not a number) and your program will crash. Secondly, clipping the gradients at a pre-defined threshold is a very simple and effective solution to exploding gradients.

  If t->infinity, and l=t-k

$$\begin{aligned} \frac{\partial h_t}{\partial h_k} &= \prod_{t \ge i > k} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \\ &= \left(\mathbf{W}_{hh}^\top\right)^l \end{aligned}$$

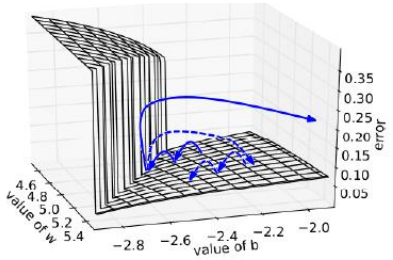
  Largest eigenvalue>1, may explode;<1,will vanish

  Gradient clipping:

Algorithm 1 Pseudo-code for nc
dients whenever they explode

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \ge threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

Gradient vanishing: proper initialization, regularization;可以把 sigmoid/tanh 改成 ReLU



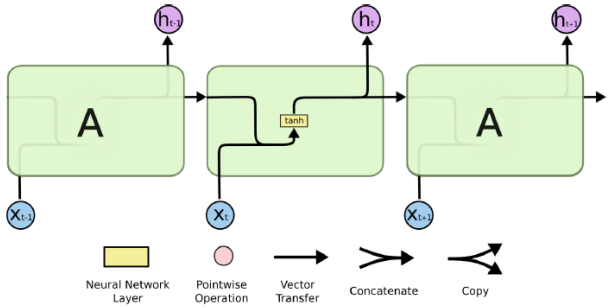
Chapter18rnn

- LSTM(long short-term memory):

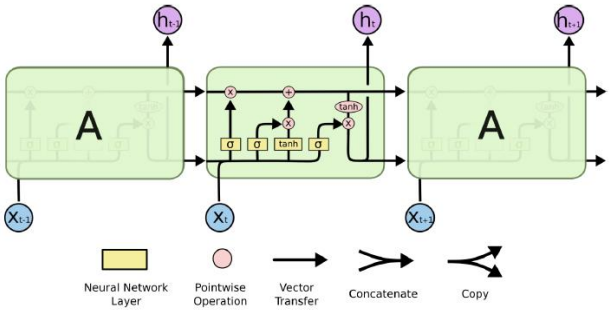
目标：want to achieve constant error flow through a single unit

Want the unit to be able to pick up long-term connections or focus on short-term ones(depend on

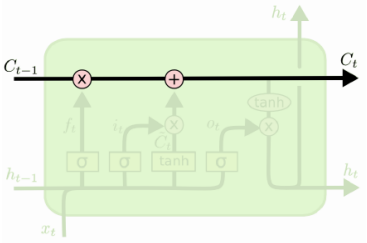
problems)



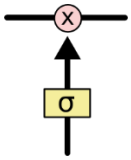
(standard rnn, simple)



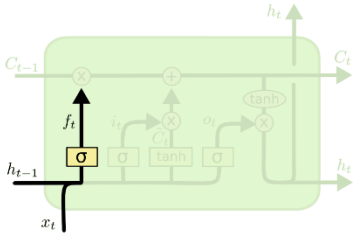
(LSTM,4 layers)



(cell state, act as a conveyor belt)

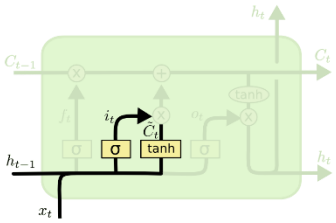


(gate, sigmoid net layers + pointwise multiplication; 0:nothing through,1: everything through; gate layers are learned with other parameters; there are 3 gates in LSTM)



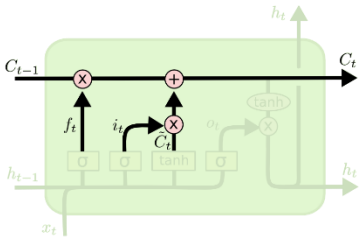
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

(forget gate layer: 0:completely delete; 1: completely keep)



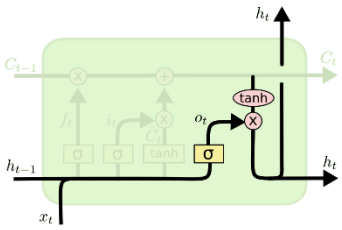
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

(update gate layer: first, a sigmoid layer called the “input gate layer” decides which values we’ll update. Next, a tanh layer creates a vector of new candidate values, C~t, that could be added to the state.)



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

(updating: first multiply old state by ft(forget), then add new information; it 也就是 sigmoid 的输出控制更新 scale)



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

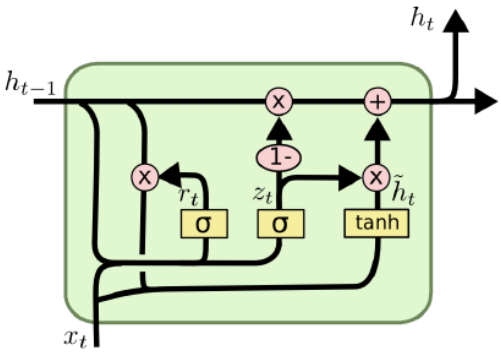
(output gate layer: First, we run a sigmoid layer which decides what parts of the cell state we’re going to output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.)

LSTM better than RNN:

a) More expressive multiplicative interactions
b) Gradients flow nicer
c) The network can explicitly decide to reset the hidden state

- GRU:

a) Combines forget and input gates into update gate
b) Similar definition for reset gate
c) Merge cell state and hidden state



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

GRU similar performance, but fewer parameters(2 gates vs 3gates of LSTM)
Reset->0, ignore previous
Update->0, copy through many steps
Short-term: active reset gate
Long-term: inactive update gate

Chapter19reinforcement-learning

- Reinforcement learning:

a) Agent interact with environment
b) Action influence future state
c) Success measured by reward
d) Select actions to maximize future rewards

- Formalized as a partially observable markov decision process(POMDP)
- Thus the discounting factor is a weight that controls whether the value function favors prudent or greedy actions(就是越 future 的 reward 越小)
- use both the policy and value functions to guide the agent to learn good strategies to achieve that outcome
- expected discounted return:

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

- value of state under policy

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|\, S_t = s]$$

Value of taking action a in state s under policy

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|\, S_t = s, A_t = a]$$

V function=>q function

- Bellman optimality equations:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a)$$

$$= \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]$$

(both unique solution to this systems)

- exploration(non-greedy)-exploitation(greedy) trade-off

Questions:

1. 3, P44,yellow remark
2. 5, P34,例子是简单的步长的表示？
3. 6, P22,如何从 y 和 z 的关系式，推出 jacobian 的式子？
4. 6, P27,为啥 p 要 unnormalize 成 p~?只有 q 调整 k 不就成了？
5. 7, P15,variance 是怎么算的
6. 7, P22,这个图能怎么出题
7. 7, P25,invariant, homogeneous,以及 yellow remark
8. 8, P41,yellow remark
9. 9, P18, SVM 没仔细看
10. 10,P15，如果再多一层，式子怎么写？多一个求和号？
11. 11，P15,为啥 normalization 要连着 w 的惩罚一起，这个应该不变吧
12. 11, P50,图片没看懂为啥减少 learning rate 会突然往下掉