deep-learning 笔记

徐世桐

1 Chapter 6

SVM 支持向量机

仍通过 w^Tx+b 得到输出，输出仅表示 identity，正值说明有 identity，负值说明没有

依据：一个平面的公式为 $\beta_0+\beta_1x_1+\beta_2x_2=0$，则当计算 w^Tx+b 得到值后，>0 则为平面上方的数据点，<0 为下方数据点

kernel trick

kernel method 将数据集表示成相近的两个数据点一组的集合 (x_i,x_j)，kernel method 将一对数据变为单一数据点 $x=k(x_i,x_j)=\langle\phi(x_i),\phi(x_j)\rangle$

kernel method 使用 ϕ 转换数据的纬度，而点乘化简后无需先计算 $\phi(x_i),\phi(x_j)$ 即可得到新数据点 x

manifold hypothesis:

当训练数据集合包含大量无规律的数据，则将其中大部分视为无效数据，并只关心落在一个 manifold 上的数据。

例：生成图像文字声音时数据大多很集中，当像素文字随机分布时生成图像大多无意义

deep feedforward network/feedforward neural network/multilayer perceptrons MLP：

找到 θ 使得 $f(x;\theta)$ 最接近数据 y 值。f^* 为最理想的 f，即 $f^*(x)=y$。θ 可为多个参数，如 $f(x;w,b)=x^Tw+b$

$f^*(x)=f^{(3)}(f^{(2)}(f^{(1)}(x)))$，$f^{(1)}$ 为 network 第一层。每一 $f^{(i)}(x)=\phi(x;\theta)^Tw$

神经网络

1. 结构：

   输入层没有 weight，第一 hidden layer 得到所有输入层的值。

   hidden layer 和输出层所有输出都为 0/1，非连续的值

2. 一层 hidden layer 计算方法：$f^{(i)}(x;W,c)=\sigma(W^Tx+c)$

   x 为前一层的输出向量，输入层 x 即为训练参数向量。

   c 为此层常数向量

   z = W^Tx 为一层 hidden layer 对输入取得的中间值向量，称 logit。a = $\sigma(z+c)$ 为对 z + c 每一元素取 σ 的结果向量，a 即此层的输出。

   W 为此层参数矩阵，行数 = 前层节点数，列数 = 当前层节点数

   X 为多个参数点的训练集中前一层的输出矩阵，行数为数据点个数，列数为前一层节点数

   XW 当 W 对参数集矩阵操作时，每行向量 z_i^T 此时为一层 hidden layer 各节点对第 i 参数点的中间值向量。对每行 $+c^T$ 并分别取 σ 得到输出矩阵，a_{ij} 为当使用第 i 个参数点时此层第 j 节点的输出

cross entropy

分部 p 和分部 q 间的 cross entropy $H(p,q) = -E_p(\log(q))$。为 expected value of $log(q)$ with respect to distribution p

cost function

当使用 maximum likelihood 估计参数时，cost function$J(\theta)$ 为训练输入参数的分部和训练结果参数的分部间 cross-entropy：$J(\theta) = -E_{x,y\sim training\_\_dataset}(log(p_{model}(y|x)))$

对于每一在训练集内的 (x, y)，求 $log(p_{model}(y|x))$，并求 expected value。$p_{model}(y|x)$ 即训练得到的 y 关于 x 的分部

例：当 model 为 $y = N(f(x;\theta),1)$ 正则分部时，$J(\theta) = -E_{x,y\sim data}(y - f(x;\theta))^2 + const$

output layer

当输出层的结果和不为 1 时，代表数据没有被准确分到某一类中，使用 exponentiation and normalisation

normalisation 后结果 $p = \frac{\tilde{p}}{\sum_{\tilde{p'}}}$，为 $\tilde{p}$ 在所有结果中占的比例。$\tilde{p}$ 为未 normalise 值

假设输出层结果 $\tilde{P}(y|x)$ 有 $log(\tilde{P}(y|x)) = yz$

$\tilde{P}(y|x) = exp(yz)$

$P(y|x) = \frac{exp(yz)}{\sum_{y'=0}^{1} y'z}$，称 **softmax function**

$P(y|x) = \sigma((2y-1)z)$，y，y' 为训练目标结果，所以 $\sum_{y'=0}^{1}$ 包含所有 y'

对 softmax function 使用 log likelihood 原因：log $softmax(z)_i = z_i - log\sum_j exp(z_j)$。

当 z_i 为 dominant，并对应期望的输出项。log $softmax(z)_i = 0$。则此项不产生高 cost，否则产生 cost。

hidden unit

代表一个 hidden layer 节点的激发函数。

1. rectified linear unit: $g(x) = max(0,x)$

   无法用于 gradient based learning，由于一阶导为 0

   基于 rectified linear unit 的优化：$g(x) = max(0,x) + a * min(0,x)$

   a = -1：absolute value rectifier

   a 为极小值：leaky ReLU

   a 为可学习值：Parametric ReLU, PReLU

2. Maxout units

   将 x 分为多组，每组 h(x) 为组内最高值

backward propagation

一种计算 gradient 的方法，区别于使用 gradient 进行学习的 stochastic gradient descent

算法：

After the forward computation, compute the gradient on the output layer:
$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$
for $k = l, l-1, \ldots, 1$ **do**
  Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation (element-wise multiplication if f is element-wise):
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$
  Compute gradients on weights and biases (including the regularization term, where needed):
  $\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$
  $\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$
  Propagate the gradients w.r.t. the next lower-level hidden layer's activations:
  $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top} \boldsymbol{g}$
end for