# deep-learning 笔记

徐世桐

## 1 基本定义

**label 标签**：输出结果，$\hat{y}$ 为计算得到的结果，$y$ 为实际测量结果
**feature 特征**：用于预测标签的输入变量，$x_j^{(i)}$ 为第 i 组 sample 第 j 号特征
**sample 样本**：一组特征的取值和对应的标签输出
**batch**：batch size 个 sample 被分为一组，进行向量化的计算，称 $B$
**hyperparameter 超参数**：人为设定的参数。如样本个数（批量大小 batch size）$|B|$，学习率 $\eta$。少数情况下通过学习得到
**全连接层 fully-connected layer/稠密层 dense layer**：此层所有节点都分别和前一层所有节点连接
**softmax 函数**：$softmax(Y) = \frac{exp(y)}{\sum_{y' \in Y} exp(y')}$，将数值输出转化为概率值，1. 值为正 2. 值总和为 1
**cross entropy 交叉熵**

定义：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

公式：$H(y^{(i)}, \hat{y}^{(i)}) = -\sum_{j \in B} y^{(i)} log(\hat{y}^{(i)})$

使用：联系两个值概率分部间的差异，即可将数值输出 $\hat{y}$ 和分类结果 $y$ 直接做对比

仍可和 softmax 同时使用，softmax 将可能性先转换为正数并和为 1，随后使用 cross entropy

## 2 linear regression 线性回归

**平方代价函数**：$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} J^{(i)}(\theta)$，为所有样本误差的平均值
**迭代**：$\theta_i = \theta_i - \frac{\eta}{|B|} \sum_{i \in B} \frac{dJ^{(i)}(\theta)}{d\theta_i}$，即对所有 sample 训练一次，得到 label 差值，对每一参数减斜率 * 学习率的平均值

当使用平方代价函数：

$\theta_i = \theta_i - \frac{\eta}{|B|} \sum_{i \in B} x_i^{(j)} (x_1^{(j)} \theta_1 + x_2^{(j)} \theta_2 + ... + const - y^{(j)}) = \theta_i - \frac{\eta}{|B|} \sum_{i \in B} x^{(i)} (\hat{y}^{(j)} - y^{(j)})$
$const = const - \frac{\eta}{|B|} \sum_{i \in B} (x_1^{(j)} \theta_1 + x_2^{(j)} \theta_2 + ... + const - y^{(i)}) = const - \frac{\eta}{|B|} \sum_{i \in B} (\hat{y}^{(j)} - y^{(j)})$

对样本 i 的偏导数向量为 $\nabla_\theta J^{(i)}(\theta) = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ ... \\ 1 \end{bmatrix} (\hat{y}^{(i)} - y^{(i)})$

**交叉熵代价函数**：$J(\theta) = \frac{1}{|B|} \sum_{i \in B} H(y^{(i)}, \hat{y}^{(i)})$
**softmax 线性回归**：单层神经网络，使用 softmax 代价函数
**过拟合问题**

**1. 权重衰减**：在代价函数中惩罚高权重的值，尽可能使所有权重值减小

新代价函数 $= J(\theta) + \frac{\lambda}{2}\sum_{w\in W} w^2$，即 $J(\theta) + \frac{\lambda}{2}*$ 所有权重的平方和。$\lambda$ 为超参数，决定权重衰减的程度

**2. 丢弃法**

每一权重（不包括 const）有 p 的几率 $\theta' = 0$，有 1-p 的几率 $\theta' = \frac{\theta}{1-p}$

为了得到确切的值，在测试模型时较少使用

**初始化参数**

**1.MXNet 默认随机初始化**：所有权重 $\sim N(0,1)$ 的 normal distribution，所有 const 取 0

**2.Xavier 随机初始化**：对一全连接层，输入个数 a，输出个数 b，则所有参数 $\sim U(-\sqrt{\frac{6}{a+b}}, \sqrt{\frac{6}{a+b}})$

**预处理数据集**

**1. 特征标准化**：$x' = \frac{x-\mu}{\sigma}$，即统计中 z 值

**2. 离散值转换成指示特征**：对于一个可取值为 A, B, C 的离散输入值，转换成 3 个数值输入。即如果原输入为 A，转换后 3 个数值输入为 1，0，0。原离散值为 B 则转换后为 0，1，0

**结构**

- 将训练集分组，每组 batch_size 个 sample。
- 对这个 batch 的数据进行向量化计算，计算 loss，斜率，调用优化函数。
- 即每一 batch 使用相同的权重偏差。一次训练一共只遍历一次所有 sample，共 $\frac{sample\_size}{|B|}$ 次向量化计算

# 3 convolutional neural network 卷积神经网络

**互相关运算**：

输入一个二维数组，和**二维核 kernel** 进行互相关运算，得到二维数组

**二维核/卷积核/filter 过滤器**：在输入数组上滑动，每次和二维数组矩阵一部分按元素相乘，相乘结果求和作为输出矩阵的元素

**二维卷积层**：

将输入和卷积核做互相关运算，结果加上 const 作为输出

# 4 Deep Feedforward Network (Deep Learning 第 6 章笔记)

**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 $\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$，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 $\phi$ 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 $\theta$ 使得 $f(x;\theta)$ 最接近数据 y 值。$f^*$ 为最理想的 f, 即 $f^*(x) = y$。$\theta$ 可为多个参数，如 $f(x;w,b) = x^T w + b$

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

**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：$f^{(i)}(x;W,c) = \sigma(W^T x + c)$

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 = $W^T x$ 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 $\sigma$ 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 $\sigma$ 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**

---