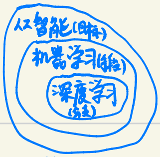
Machine Learning

Lecture 0-1



生物行为 { 先天 / 后天

机器行为 { 先天(人为设定) / 后天(机器学习)

You write the program for learning

Looking for a Function From Data

f(Information) = result

Framework supervised learning

A set of functions (Model) → goodness of function ← Training Data — training

pick the "Best" function → f^* testing

steps:

1、define a set of functions

2、goodness of function

3、pick the best one

Learning Map:

Semi-supervised Learning: 较少 labeled data

Transfer Learning: 少量 Data 和大量无关 Data

Unsupervised Learning: 无 labeled Data

Reinforcement Learning: 不固定输出, 只给输出打分 — learn from critics

Supervised Learning: 大量 Data

Regression 回归预测

↓

Linear Model

↓

Deep Learning | SVM, decision tree, K-NN…

Non-linear Model

↓

structured Learning ← A lot more

Classification

{ Binary Classification: Yes/No

Multi-class classification: class1, class2…

Lecture 1 Regression

Cases:

stock Market Forecast

Self-driving Car → direction

Recommendation

x_i 下标表示输入的某个分量

$f(x_1, x_2 \cdots x_n) = y$

Step 1: Model

$y = b + w \cdot x_{cp}$ → set of function

(b, w: parameter)

Linear Model

$y = b + \sum w_i x_i$ — x_i: an attribute of input x (feature)

b: bias w_i: weight

Step 2: Goodness of function

x^i 用上标表示一个完整的 object 编号

$f(x^i) = \hat{y}^i$ 用 hat 表示真实值

Loss function L: $L(f)$ = how bad 可自定

$L(w,b)$ 一般 $\sum(\hat{y}^n - (b + w x^n_{cp}))^2$

draw a graph of $L(w,b)$

一个点代表一个 f, 可用颜色体现 L(f)

pick the best f — Gradient Descent 可微

$f^* = \arg\min_f L(f)$

Step 3: Gradient Descent: ex. L(w)

Randomly pick w^0

$w^1 = w^0 - \eta \frac{dL}{dw}\Big|_{w=w^0}$ ← $\frac{dL}{dw}$ 越大, 跨度越大

η: learning rate

$w^2 = \cdots$

$w^3 \cdots$

⋮

local optical $\frac{dL}{dw} = 0$ not global optical

多参数情况: $L(w, b)$

Random Pick (w^0, b^0)

$w^1 = w^0 - \eta \frac{\partial L}{\partial w}\Big|_{(w,b)=(w^0,b^0)}$, $b^1 = b^0 - \eta \frac{\partial L}{\partial b}\Big|_{(w,b)=(w^0,b^0)}$

$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w} \\ \frac{\partial L}{\partial b} \end{bmatrix}$

Generalization

care about the error on new data (testing data)

error 过大 → 修改 model (更高次幂)

可能会对 training data 符合很好但与 testing data 偏差较大

$\text{error} = \sum |\hat{y}^i - y^i|$

复杂的model可以降低 training data的error rate

但对testing data都不一定 (Overfitting)

有可能设计的model (set of function)里并没有正确的f

$f(x,y) \notin f(x)$ can't fit testing data → redesign the model

$y = b_1 + \delta(\cdots) w_1 x_{cp}$ (Back to step 1)

$b_2 + \delta(\cdots) w_2 x_{cp}$

$\vdots$

$y = b + \sum w_i x_i$

Back to step 2: Regularization

redefine loss function

$L = \sum_n \left(\hat{y}^n - (b + \sum w_i x_i) \right)^2 + \lambda \sum (w_i)^2$

why? (先暂不考虑bias)

The functions with smaller w_i are better

smooth

使得单一分量对结果产生的影响更小

output对输入的变化更不敏感

受noises的影响较小

Select λ to obtain the best model

Conclusion:

Gradient descent

Overfitting & Regularization

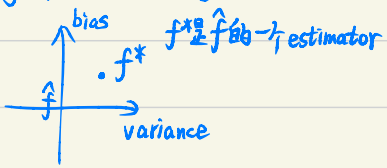
Validation

Lecture 2

error due to { bias / variance

Estimator

$\hat{y} = \hat{f}(x)$ training data → f^*



f^*是$\hat{f}$的一个estimator

例: estimate $\bar{x}$

mean: μ, variance: σ^2

estimate: N sample points: $\{x^1, x^2 \cdots x^N\}$

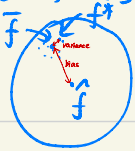
$m = \frac{1}{N}\sum x^i$ $m \neq \mu$ $E(m) = E(\frac{1}{N}\sum x^i) = \frac{1}{N}\sum E(x^i) = \mu$

$E(m) = \mu$: unbiased

$Var[m] = \frac{\sigma^2}{N}$ (depends on the number of samples)

$S^2 = \frac{1}{N}\sum (x^i - m)^2$ $E(S^2) = \frac{N-1}{N}\sigma^2 \neq \sigma^2$ biased estimator

$E(f^*) = \bar{f}$ $\bar{f} \to f$



data → training / testing 每次法选取不同的training data

Same model, different training data ⇒ diff f^*

Simple model → low variance & high bias vise versa

less influenced by the sampled data

但是function set更小, bias偏大

variance大: overfitting	bias大: underfitting
能fit training data	不能fit training data
不能fit testing data	↓
↓	增加模型复杂度
增加data (有效,不实际)	(增加数据量来没有用)
(generate假的training data)	$\hat{f}$不在function set里
自己制作data	
Regularization (smoothen)	
相当于调整function space,可能增大bias	

Model Selection: trade-off between bias and variance

Should NOT do:

Training Set / Testing Set

Model1 → Err 0.9

Model2 → Err 0.7

Model3 → Err 0.5 pick X

Training Set ⇆ Training Set / Validation Set

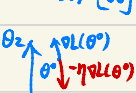
"kaggle"

N-fold Cross Validation 将training set分割为n份, 每次任取一份作为validation

Lecture 3 Gradient Descent

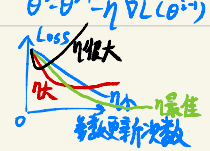
$\theta^* = \arg\min_\theta (L(\theta))$ L: loss function θ: parameter

$\nabla L(\theta) = \begin{bmatrix} \partial L / \partial \theta_1 \\ \partial L / \partial \theta_2 \end{bmatrix}$ $\theta^1 = \theta^0 - \eta \nabla L(\theta^0)$



Tip 1: Tuning your learning rate

$\theta^i = \theta^{i-1} - \eta \nabla L(\theta^{i-1})$



Adaptive Learning Rates

Popular & simple Idea: 减小 η by some factor

开始：η 大，一段时间后适当减小 η

给不同的 parameter 设置不同的 η

Adagrad:

普通 Gradient Descent: $w^{t+1} \leftarrow w^t - \eta^t g^t$

Adagrad: $w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t$ $\eta^t = \frac{\eta}{\sqrt{t+1}}$

$w^1 \leftarrow w^0 - \frac{\eta^0}{\sigma^0} g^0$ $\sigma^0 = \sqrt{(g^0)^2}$

$w^2 \leftarrow w^1 - \frac{\eta^1}{\sigma^1} g^1$ $\sigma^1 = \sqrt{\frac{1}{2}[(g^0)^2 + (g^1)^2]}$

$\vdots$

$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t$ $\sigma^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g^i)^2}$

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t$$

↖不一定沿梯度方向?

adagrad能自动选择最优路径

Adagrad考虑 grad的变化

How surprise it is 反差

$\sqrt{\sum_{i=0}^{t}(g^i)^2}$ ←造成反差的效果

best step: $\frac{|y'|}{|y''|}$ (以 $y=ax^2+bx+c$ 为例)

$\sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g^i)^2}$ →反映 y'' 的大小

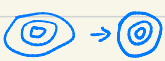
Tip2: Stochastic Gradient Descent

选一个 example x^n

$L^n = (\hat{y}^n - (b+\sum w_i x_i^n))^2$ 仅采用一个数计算 L

Tip3: Feature Scaling

使不同feature的Scale是一样的



对 $x_1^1, x_1^2, x_1^3 \cdots$ 进行类似随机变量标准化的操作：$\frac{x-\mu}{\sigma}$

Limitation of Gradient Descent

1. Local Min
2. settle point (在驻点附近更新维个量)

Lecture 4 Classification

$x \rightarrow f \rightarrow$ class n

Classification as Regression? X

Ideal Alternatives

Function (Model): $x \rightarrow f(x)$

$g(x)>0 \rightarrow$ class1

$g(x)<0 \rightarrow$ class2

Loss function: f 判断错误的次数

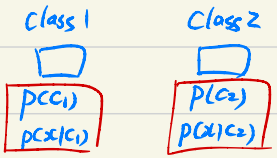
$L(f) = \sum_n \delta(f(x^n) \neq \hat{y}^n)$

Find the best function:

Perceptron, SVM...

贝叶斯公式 (从概率角度思考)

在已知抽到 x 的情况下，x 最可能来自哪个 class



$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1)+P(x|C_2)P(C_2)}$$

$$\Rightarrow P(C_2|x) = \frac{P(x|C_2)P(C_2)}{P(x|C_1)P(C_1)+P(x|C_2)P(C_2)}$$

Generative Model → generate x

$P(x) = P(x|C_1)P(C_1) + P(x|C_2)P(C_2)$

$P(C_1), P(C_2)$: prior 可直接由 training data 计算

$P(x|C_1) = ?$ $P(x|C_2) = ?$ 求 C1, C2 中数据的分布

每个 x 由一个向量描述

Gaussian Distribution

输入 x，输出取到 x 的概率 (密度)

假设所有点是从 Gaussian 分布中随机选出来的 → 求高斯分布

求 μ, Σ：Maximum Likelihood

Likelihood: $L(\mu,\Sigma) = f_{\mu,\Sigma}(x^1) f_{\mu,\Sigma}(x^2) \cdots f_{\mu,\Sigma}(x^n)$

取 $L(\mu,\Sigma)$ 最大的 μ, Σ (μ^*, Σ^*)

$\mu^* = \frac{1}{N}\sum x^i$, $\Sigma^* = \frac{1}{N}\sum (x^i - \mu^*)(x^i - \mu^*)^T$

[ ] [ ] → [ ]

为减少参数的个数，让各 class 共用一个 Σ

$L(\mu^1, \mu^2, \Sigma) = f_{\mu^1,\Sigma}(x^1) f_{\mu^1,\Sigma}(x^2) \cdots f_{\mu^1,\Sigma}(x^{79}) \times$

$f_{\mu^2,\Sigma}(x^{80}) \cdots f_{\mu^2,\Sigma}(x^{140})$

μ^1 和 μ^2 计算方法相同，Σ 根据数据量加权：$\Sigma = \frac{79}{140}\Sigma^1 + \frac{61}{140}\Sigma^2$

3 steps:

Function Set (Model)

$x \rightarrow \boxed{f:\ P(C_1|x)} \rightarrow \begin{matrix} C_1\ (P(C_1|x)>0.5) \\ C_2\ (\text{other}) \end{matrix}$

Goodness of function:

μ, Σ 使得 likelihood 最大

Pick the best function

可任意选择概率分布(例子里是Gaussian)

假设 $x_1, x_2, x_3 \cdots x_k$ 是相互独立的(有的时候不能这么假设)

$P(x|C_1) = P(x_1|C_1)P(x_2|C_1)\cdots P(x_k|C_1)$

Naive Bayes Classifier

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1)+P(x|C_2)P(C_2)} = \frac{1}{1+\frac{P(x|C_2)P(C_2)}{P(x|C_1)P(C_1)}}$$

$$z = \ln\frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)}, \quad P(C_1|x) = \frac{1}{1+e^{-z}} = \sigma(z)$$

$$z = \underbrace{(\mu^1-\mu^2)^T\Sigma^{-1}}_{w^T}x - b \Rightarrow P(C_1|x) = \sigma(w\cdot x+b)$$, 又需找 w 和 b

Lecture 5 Logistic Regression

找 $P_{w,b}(C_1|x) = \sigma(z)$ $\quad z = w\cdot x+b = \sum_i w_i x_i + b$

$f_{w,b}(x) = P_{w,b}(C_1|x)$

Step 1: $f_{w,b}(x) = \sigma\left(\sum_i w_i x_i + b\right) \in (0,1)$

Step 2: 假设数据是由 $f_{w,b}(x)$ 产生的

对于一组 w, b, 计算样本是由 $f_{w,b}$ 产生的可能性

$L(w,b) = f_{w,b}(x^1)\, f_{w,b}(x^2)\,(1-f_{w,b}(x^3))\cdots$

$w^*, b^* = \arg\max_{w,b} L(w,b)$

$= w^*, b^* = \arg\min_{w,b} -\ln L(w,b)$

$\ln L(w,b) = \ln f_{w,b}(x^1) + \ln f_{w,b}(x^2) + \ln(1-f_{w,b}(x^3)) + \cdots$

$x^1 \rightarrow \hat{y}^1 = 1 \quad x^2 \rightarrow \hat{y}^2 = 1 \quad x^3 \rightarrow \hat{y}^3 = 0$

$\ln f_{w,b}(x^1) = \hat{y}^1 \ln f(x^1) + (1-\hat{y}^1)\ln(1-f(x^1))$

$-\ln L(w,b) = \sum_n -[\hat{y}^n \ln f(x^n) + (1-\hat{y}^n)\ln(1-f(x^n))]$

cross entropy 表示两变量有多接近

$L(f) = \sum_n C(f(x^n), \hat{y}^n)$

Loss, 与上面的 L 不同

$C(f(x^n), \hat{y}^n) = -[\hat{y}^n \ln f(x^n) + (1-\hat{y}^n)\ln(1-f(x^n))]$

使 function 的 output 尽可能与 $\hat{y}$ 接近

不能用线性回归里的方差

Step 3: find the best function

$w_i \leftarrow w_i - \eta \sum_n -(\hat{y}^n - f_{w,b}(x^n))x_i^n$

(与线性回归相同) $\hat{y} = 0$ 或 1

Logistic Regression + Square Error?

会出现离最优解很远但 $L(f)\approx 0$ 的情况

Discriminative 和 Generative

相同的 model，不同的假设

贝叶斯 潜在问题：$P(C|x)$ 最大的 Class 并不一定是正确的 Class（可能是 training data 中该 class 占比很大）

Generative Model 作了更多的假设（针对少量 data）

受 data 影响较小（针对带 noise 的 data）

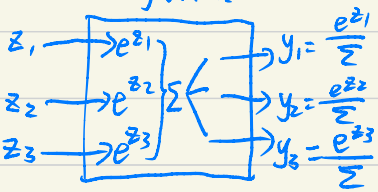
Multi-class Classification

$C_1: w^1, b_1 \quad z_1 = w^1 \cdot x + b_1$

$C_2: w^2, b_2 \quad z_2 = w^2 \cdot x + b_2$

$C_3: w^3, b_3 \quad z_3 = w^3 \cdot x + b_3$

Softmax



normalization

对最大值做强化 $y_i = P(C_i|x)$

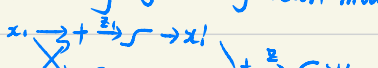
计算 y 与 $\hat{y}$ 的 cross entropy $\sum -\hat{y}_i \ln y_i$

$\hat{y} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$

Logistic Regression 的局限性（分界线是线性的）

feature transformation（很难找）

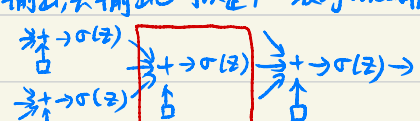
Cascading logistic regression models



将 feature transformation 当作特性变换

通过 Logistic Regression 寻找线性变换

某一个 Logistic 的多个输入可以是其它 logistic 的输出，其输出也可以是下一级 logistic 的输入



neuron

Neural Network

Lecture 6 Deep Learning Introduction

Perceptron (Linear)

Perceptron Limits

Multi-layer perceptron（与现在的 DNN 差不多）

Backpropagation

通常3片以上 hidden layer 没有太大用处

1片 hidden layer "good enough"

RBM initialization (breakthrough) 实际上没什么用

GPU 加速

Speech recognition

ILSVRC image competition

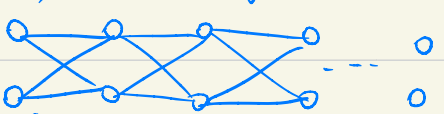
Step 1：Define function set (Neutral Network)

θ：所有的 weight 和 bias

连接方式决定 Neural Network 的结构

连接方式：

1. Fully Connect Feedforward Network



当 function 看，输入向量，输出向量

由 Neural Network 的结构确定一个 function set

Deep：Many hidden layers

用矩阵操作来表示 现在较多会用 sigmoid

$\sigma\left(\begin{bmatrix} \quad \end{bmatrix}_{w}\begin{bmatrix} \quad \end{bmatrix} + \begin{bmatrix} \quad \end{bmatrix}_{b}\right) = \begin{bmatrix} \quad \end{bmatrix}$ σ 不一定是 sigmoid

写成矩阵运算的形式，由 GPU 计算

可进化神经网络？GA 来确定神经网络结构？用的不多

2. Convolutional Neural Network (CNN)

一种神经网络的连接方式

total loss $L = \sum_{n=1}^{N} C^n$ $C(y, \hat{y}) = -\sum_i \hat{y}_i \ln y_i$

使 L 最小

Gradient Descent

Backpropagation：在NN中有效地计算 $\frac{\partial L}{\partial w}$

Universality Theorem

连续函数 $f: R^N \to R^M$ (任意)

可用带1个hidden layer的NN实现

(在hidden layer中用足够多的neuron)

Lecture 7 Backpropagation

为了高效计算Gradient (很多参数)

本质上是Chain Rule

$L(\theta) = \sum_{n=1}^{N} C^n(\theta)$ $\quad \frac{\partial L(\theta)}{\partial w} = \sum_{n=1}^{N} \frac{\partial C^n(\theta)}{\partial w}$

$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial z}{\partial w}\frac{\partial C}{\partial z}$

$\frac{\partial z}{\partial w}$：forward pass 好求

$\frac{\partial C}{\partial z}$：backward pass 不好求

forward pass：$\frac{\partial z}{\partial w} = x$ $(z = \sum w_i x_i + b)$

backward pass：$\frac{\partial C}{\partial z}$

$\frac{\partial C}{\partial z} = \frac{\partial a}{\partial z}\frac{\partial C}{\partial a}$, $\frac{\partial C}{\partial a} = \frac{\partial z'}{\partial a}\frac{\partial C}{\partial z'} + \frac{\partial z''}{\partial a}\frac{\partial C}{\partial z''} \cdots$

$= \sum w \frac{\partial C}{\partial z}$ 再向下一层…

实际上是从Output层开始算，反向传播

建立反向的NN用于backward pass

Lecture 8 "Hello World"

Keras MNIST

```
model = Sequential()                                   # units
model.add(Dense(input_dim=28*28, output_dim=500))
model.add(Activation('sigmoid'))

model.add(Dense(output_dim=500))
model.add(Activation('sigmoid'))

model.add(Dense(output_dim=10))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

numpy array

[10000, (784维)]　　[10000, (0001000…)]

```
score = model.evaluate(x_test, y_test)
```

score[0]: total loss　score[1]: accuracy

```
result = model.predict(x_test)
```

Lecture 9 Tips for Deep Learning

recipe: 1. define a set of function
2. goodness of function
3. pick the best function
↓
Neural Network

good on training set? NO ↑
↓ Yes
good on testing set? NO (over fitting) ↑

一般第一个问题不是over fitting

training结果不好:

1. New activation function
2. Adaptive Learning Rate

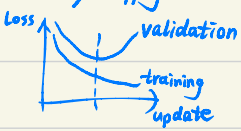
testing结果不好:

1. Early Stopping
2. Regularization
3. Dropout

Vanishing Gradient Problem:
在靠近output的地方更新快，靠近input更新慢
ΔW的变化在由Sigmoid处理后变得很小→改用ReLU
Leaky ReLU Parametric ReLU
Maxout (自动学习Activation)
RMSProp
$w^1 \leftarrow w^0 - \frac{\eta}{\sigma^0} g^0 \quad \sigma^0 = g^0$
$w^2 \leftarrow w^1 - \frac{\eta}{\sigma^1} g^1 \quad \sigma^1 = \sqrt{\alpha(\sigma^0)^2 + (1-\alpha)(g^1)^2}$
$w^3 \leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 \quad \sigma^2 = \sqrt{\alpha(\sigma^1)^2 + (1-\alpha)(g^2)^2}$
$w^{t+1} \leftarrow w^t - \frac{\eta}{\sigma^t} g^t \quad \sigma^t = \sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^t)^2}$
(更倾向于相信新的g，前面的g的影响会变弱)
Momentum：加上前一次移动的方向
$v^0 = 0, v^1 = \lambda v^0 - \eta \nabla L(\theta^0)$
$v^2 = \lambda v^1 - \eta \nabla L(\theta^1) \cdots$
adam → momentum + RMSprop

在testing上结果不好:

1. Early Stopping



2. Regularization: Redefine the loss function
Weight Decay (不常用)
3. dropout
有次参数update之前:
每个neuron有p%的几率被丢掉
↳ network结构改变
用新的network训练
testing时不dropout:
如果training的dropout rate是p%，则所有weight乘上(1-p)%

Keras Demo
model.add(Dropout(0.7))

Lecture 10 CNN

CNN：人为简化后的NN

大部分的pattern比整个image小，不同区域会有相同图像 (Conv)

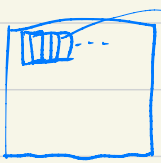
Connecting to small region with less parameters

subsampling：拿掉image的奇数行和偶数列 (MaxPooling)

↳使image变小，减少参数

CNN：图像 → convolution → max pooling（重复数次）→ flatten → DNN

用filter来识别图象的一部分



filter →

一个filter生成一个更小的pattern

用Maxpooling进行subsampling

多个filter →

feature map

彩色：RGB 对应的filter也有3层

feature map → MaxPooling → 取每块最大值 得到一个新的小的image

weight sharing

Flatten：将feature map拉直

输入三维(RGB)的vector(tensor)

keras：RGB:3

```
model.add(Convolution2D(25,3,3,input_shape=(1,28,28)))
model.add(MaxPooling2D((2,2)))
model.add(Convolution2D(50,3,3))
model.add(MaxPooling2D((2,2)))
model.add(Flatten())
```

分析CNN：第k层filter被激活的程度：$\sum a_i^k$（第k层）

找一个image使第k层的activation最大

$x^* = \arg\max_x a^k$ (gradient ascent)

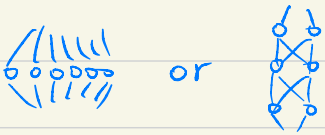
让classifier最active的可能完全不像那个class

Deep Dream Deep Style

什么时候使用CNN：

1. image中的pattern很小
2. pattern在image的不同区域出现
3. subsampling后不会改变object

Lecture 11 Why deep?



Deep → Modularization

module：detect attribute

Speech

Phoneme 发音受前后的影响

Tri-phone

State：Tied-state

End-to-end Learning

Production line

Lecture 12 Semi-Supervised Learning

training data 有部分 unlabeled (一般有很多)

unlabeled data 的分布能帮助学习(需有一定假设)

Generative Model

Low-density Separation: 不同的class间有明显的分界

Self-training

labeled data → f^* → 给未标记的数据标记

→ Pseudo-label (选一些label加入training)

Entropy-based Regularization

假设x的分布集中在某些区域

Cluster and then label

Graph-based smoothness

$S=\frac{1}{2}\sum w_{ij}(y^i-y^j)^2=y^TLy$ L: Graph Laplacian

$L=D-W$

Better Representation

Lecture 13 Unsupervised Learning

Reduction 化繁为简

Generation 无中生有

Clustering

k-means

$X=\{x^1,x^2\cdots x^N\}\rightarrow k$ cluster

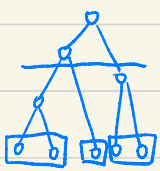
找 k 个 center

→ 求其它数据与k个center的哪个最近 → 归类

update center

Hierarchical Agglomerative Clustering (HAC)

建树: 两两计算相似度, 取最相似的对



Distributed Representation

用向量描述 attribute

分量

Dimension Reduction

Feature selection

Principle component analysis (PCA)

SVD autoencoder deep autoencoder

PCA弱点 {unsupervised, Linear}

non-negative matrix factorization

LSA, PLSA, LDA

Lecture 15 Neighbor Embedding

LLE - Locally Linear Embedding

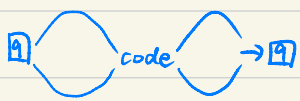
Laplacian Eigenmaps

t-SNE → 常用于 visualization

Lecture 16 Auto-encoder

9 → NN encoder → code

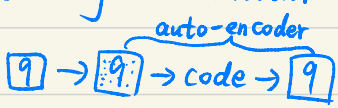
→ NN decoder → 9



Vector Space Model

pre-training DNN fine tune

De-noising auto-encoder



Contractive auto-encoder

auto-encoder for CNN

depooling de convolution (实际上就是 convolution)

可以利用 decoder 再从 code 生成 image

Lecture 17 Deep Generative Model 1

PixelRNN

Variational Autoencoder (VAE)

Generative Adversarial Network (GAN)

Pixel Recurrent Neural Network

每次画一个 pixel

$c_i = e^{\sigma_i} \cdot e_i + m_i$

VAE: input → NN encoder → m_1, m_2, m_3 and $\sigma_1, \sigma_2, \sigma_3$; exp → $e^{\sigma_1}, e^{\sigma_2}, e^{\sigma_3}$; 随机 → e_1, e_2, e_3; c_1, c_2, c_3 → NN Decoder → output

minimize: $\sum (1 + \sigma_i - (m_i)^2 - e^{\sigma_i})$ 和 reconstruction error

Lecture 18 Deep generative model 2

VAE: 加上 noise m_i: 原来的 code

c_i: 加上 noise 后的 code

Gaussian Mixture Model

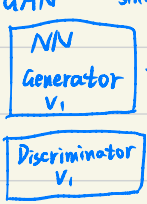
KL divergence

Conditional VAE

VAE 的问题: VAE 又产生与 traing data 相近的图像

而不是以假乱真的图像

GAN since sliced bread 有史以来



generator 不看 real image，由 discriminator 看 real image 来评价 generator 的好坏

image → discriminator → 1 real / 0 fake

Vectors from a distribution → generator → image

用 real image 和 fake image 加 label 训练 discriminator

调 generator 让 discriminator 认为是 real image

将generator和discriminator当作一个NN训练

固定discriminator的参数

difficult to optimize

LSTM: 4 input 1 output

Lecture 19 Transfer Learning

存在一些不相干的data

source data target data

用target data fine-tune model

用source data得到的model作为初始值

(容易 over fitting)

Conservative training

新旧model差距不要太大

Layer Transfer

Image:通常copy前面几层,只train后面几层

Multitask Learning

Progressive Neural Networks

Lecture 20 SVM

Hinge loss + kernel Method → Support Vector (SVM) Machine

convex

kernel Trick

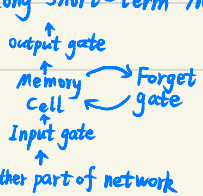
Lecture 21-1 RNN 1

Slot Filling

Elman Network & Jordan Network

Bidirectional RNN

Long Short-term Memory (LSTM)



Lecture 21-2 RNN 2

BPTT

LSTM可以解决gradient vanishing

Gate Recurrent Unit (GRU)

Attention-based Model

Neural Turing Machine

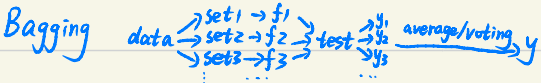
HMM

deep and structured

Lecture 22 Ensemble

A mixture of all approaches

Bagging



Model很复杂,可能over fitting时用(降低variance)

Decision Tree —bagging→ random forest

Boosting: improve week classifiers

$f_1 \to f_2 \to f_3 \cdots$

Adaboost: training $f_2(x)$ on the new training set that fails $f_1(x)$

Gradient Boosting: $g_t \in g_{t-1} - \eta \frac{\partial L(g)}{\partial g}\Big|_{g=g_{t-1}}$ $g(x)$是函数

Stacking