《神经网络理论与应用》第三讲

Neural Network Theory and Applications

主讲教师：郑伟龙
助教：尹昊龙、史涵雯

上海交通大学计算机科学与工程系

weilong@sjtu.edu.cn
http://bcmi.sjtu.edu.cn

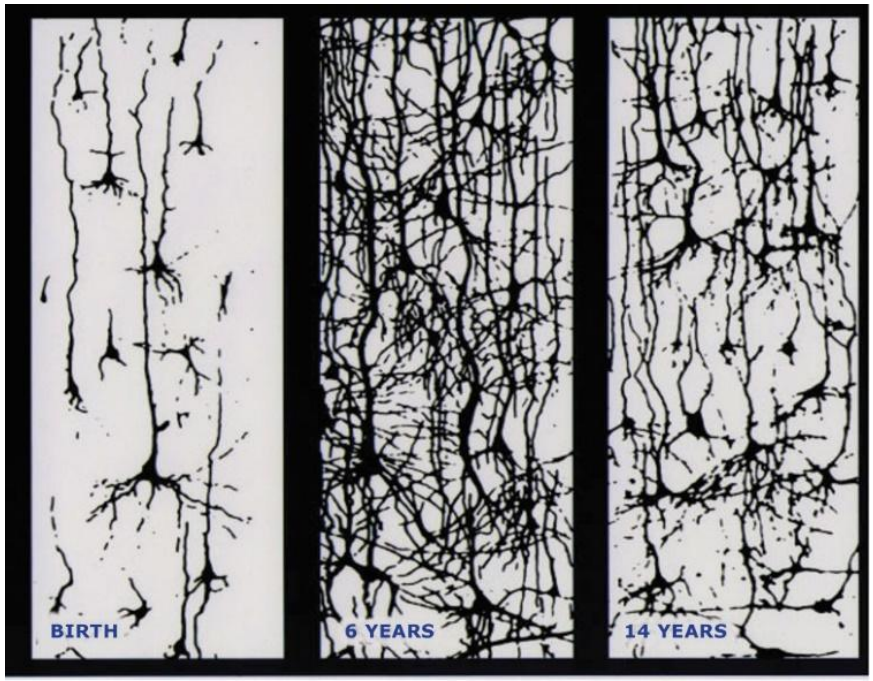
Summary of MLP and BP Algorithm

- Solve linearly non-separable problem
- Two modes of training
  - sequential; on-line; stochastic
  - batch; mini-batch
- Three factors affect generalization:
  - The size and representativeness of the training set
  - The architecture of the neural network
  - The physical complexity of the problem at hand
- How to select the number of hidden units?
- BP algorithm converges very slowly!

Two ways of determining network size

- Pruning

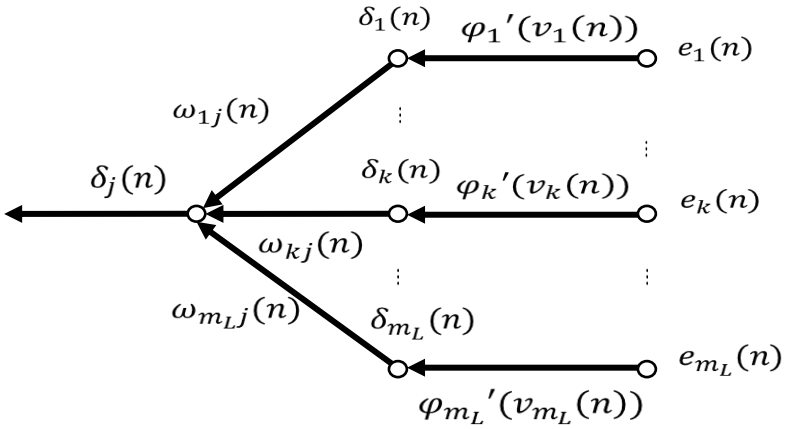
- Growing



梯度消失和梯度爆炸

- 在训练深度神经网络的时候，我们会使用梯度下降和反向传播的算法。具体来说，我们通过从最后一层向第一层遍历网络，计算偏导。利用链式法则，更加深层的梯度值会经过更加多次的矩阵乘法。
- 梯度爆炸
  - 如果导数较大，那么梯度在层层反向传播的过程中就会指数级上升，最终导致梯度过大
- 梯度消失
  - 如果导数较小，那么在反向传播的过程中，梯度就会指数级减少

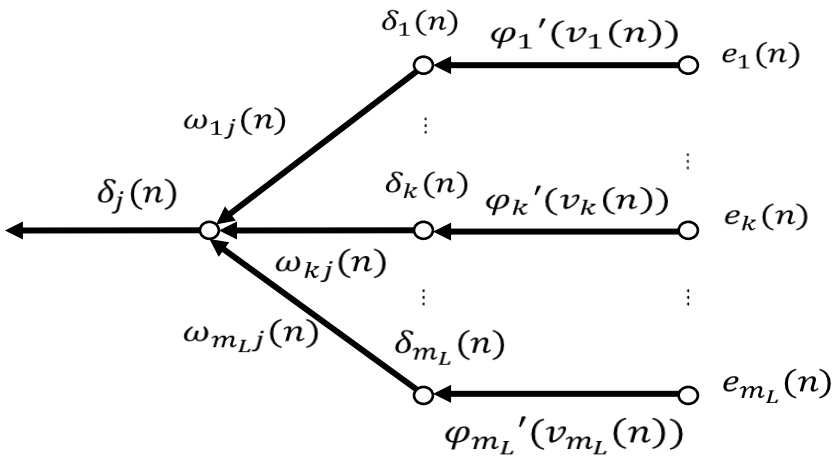
$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$



梯度消失和梯度爆炸：层数越多问题越严重

- 梯度爆炸会导致模型权重参数变化很大，导致整个网络十分不稳定，最坏情况会直接导致权重值溢出（NaN）。
- 梯度消失会导致模型失去学习的能力，无法有效地更新权重，最坏情况会直接导致梯度归零，模型停滞。

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$



激活函数的性质

- 连续并可导（允许少数点上不可导）的非线性函数。
  - 可导的激活函数可以直接利用数值优化的方法来学习网络参数。
- 激活函数及其导函数要尽可能的简单
  - 有利于提高网络计算效率。
- 激活函数的导函数的值域要在一个合适的区间内
  - 不能太大也不能太小，否则会影响训练的效率和稳定性。
- 单调递增

Outline of Lecture Three

- Deep Learning Framework
- Basic Concepts in Machine Learning
- Support Vector Machine
- Performance Evaluation Index

人工神经网络

- 人工神经网络主要由大量的神经元以及它们之间的有向连接构成。因此考虑三方面：

- 神经元的激活规则
  - 主要是指神经元输入到输出之间的映射关系，一般为非线性函数。
- 网络的拓扑结构
  - 不同神经元之间的连接关系。
- 学习算法
  - 通过训练数据来学习神经网络的参数。

损失函数

- 对于多分类问题
  - 如果使用**Softmax**回归分类器，相当于网络最后一层设置 **C** 个神经元，其输出经过**Softmax**函数进行归一化后可以作为每个类的条件概率。

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z}^{(L)}) \qquad \text{Softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

  - 采用交叉熵损失函数，对于样本**(x,y)**，其损失函数为

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbf{y}^{\mathrm{T}} \log \hat{\mathbf{y}}$$

参数学习

- 给定训练集为$D = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$，将每个样本$x^{(n)}$输入给前馈神经网络，得到网络输出为$\hat{y}^{(n)}$，其在数据集D上的结构化风险函数为：

$$\mathcal{R}(W, \mathbf{b}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)}) + \frac{1}{2}\lambda\|W\|_F^2$$

- 梯度下降

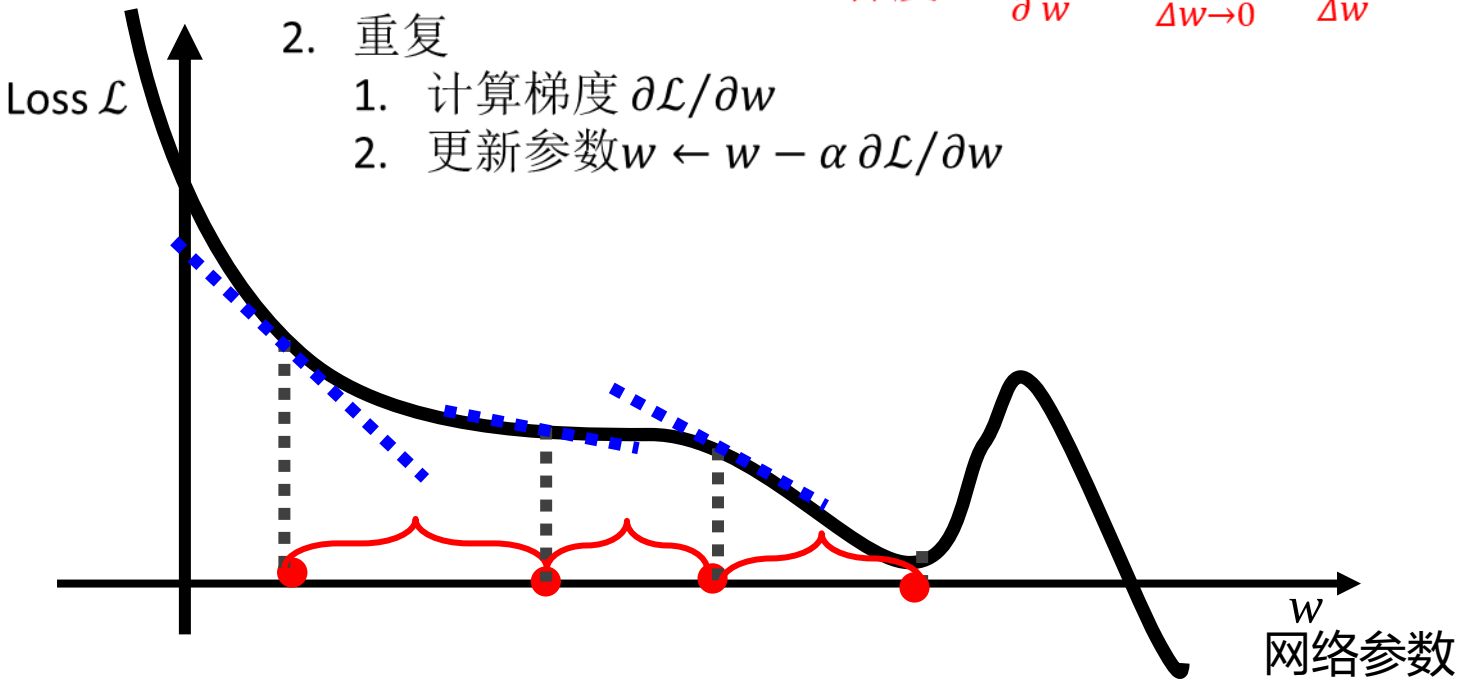
$$W^{(l)} \leftarrow W^{(l)} - \alpha\frac{\partial\mathcal{R}(W, \mathbf{b})}{\partial W^{(l)}}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha\frac{\partial\mathcal{R}(W, \mathbf{b})}{\partial \mathbf{b}^{(l)}}$$

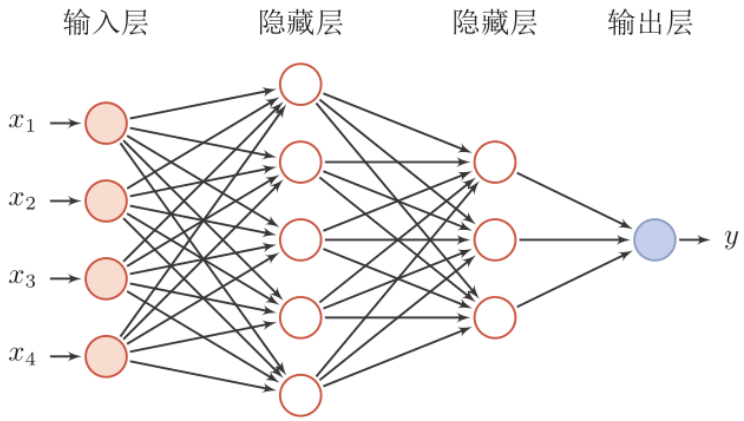
梯度下降

1. 初始化w
2. 重复
   1. 计算梯度 $\partial \mathcal{L}/\partial w$
   2. 更新参数 $w \leftarrow w - \alpha\, \partial \mathcal{L}/\partial w$

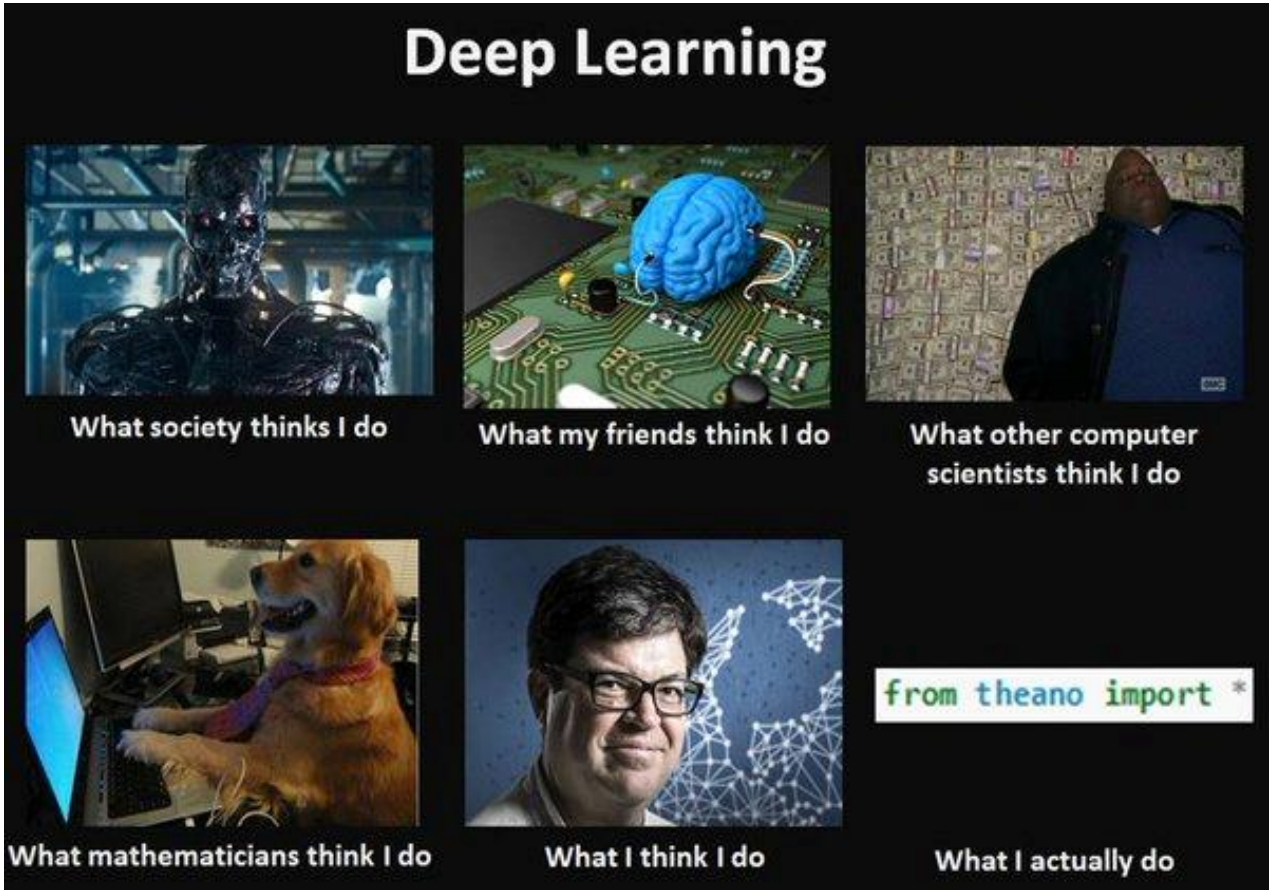
梯度：$\frac{\partial f(w)}{\partial w} = \lim_{\Delta w \to 0} \frac{f(w+\Delta w)}{\Delta w}$



构建神经网络模型的三个步骤



如何实现？


Deep Learning
What society thinks I do
What my friends think I do
What other computer scientists think I do
What mathematicians think I do
What I think I do
from theano import *
What I actually do

开源框架概述

- 在这个图像中有不同的分类：**猫，骆驼，鹿，大象**等
- **卷积神经网络（CNNs）**对于这类图像分类任务十分有效。
- **从头开始实现一个卷积神经网络模型?**
- 我们可能需要几天（甚至几周）才能得到一个有效的模型，时间成本太高!

深度学习框架让一切变得可能!

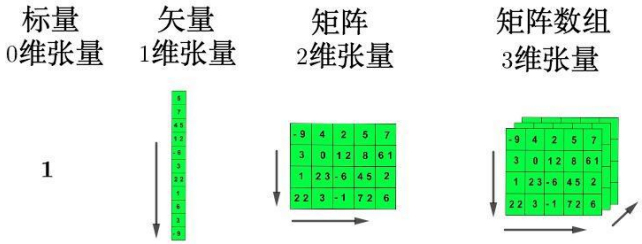
开源框架概述

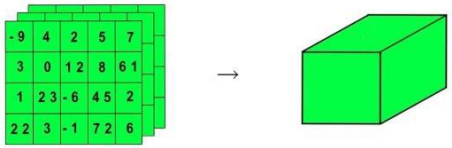
- **什么是深度学习框架?**
  - 深度学习框架是一种接口、库或工具， 利用预先构建和优化好的组件集合定义模型。
- **一个良好的深度学习框架应具备的关键特性:**
  - 性能优化
  - 易于理解和编码
  - 良好的社区支持
  - 并行处理以减少计算

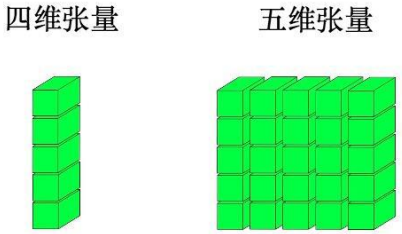
核心组件-张量

- 张量是深度学习框架中最核心的组件，**Tensor 实际上就是一个多维数组。**

- **Tensor对象具有3个属性：**
  - rank： number of dimensions
  - shape: number of rows and columns
  - type: data type of tensor's elements



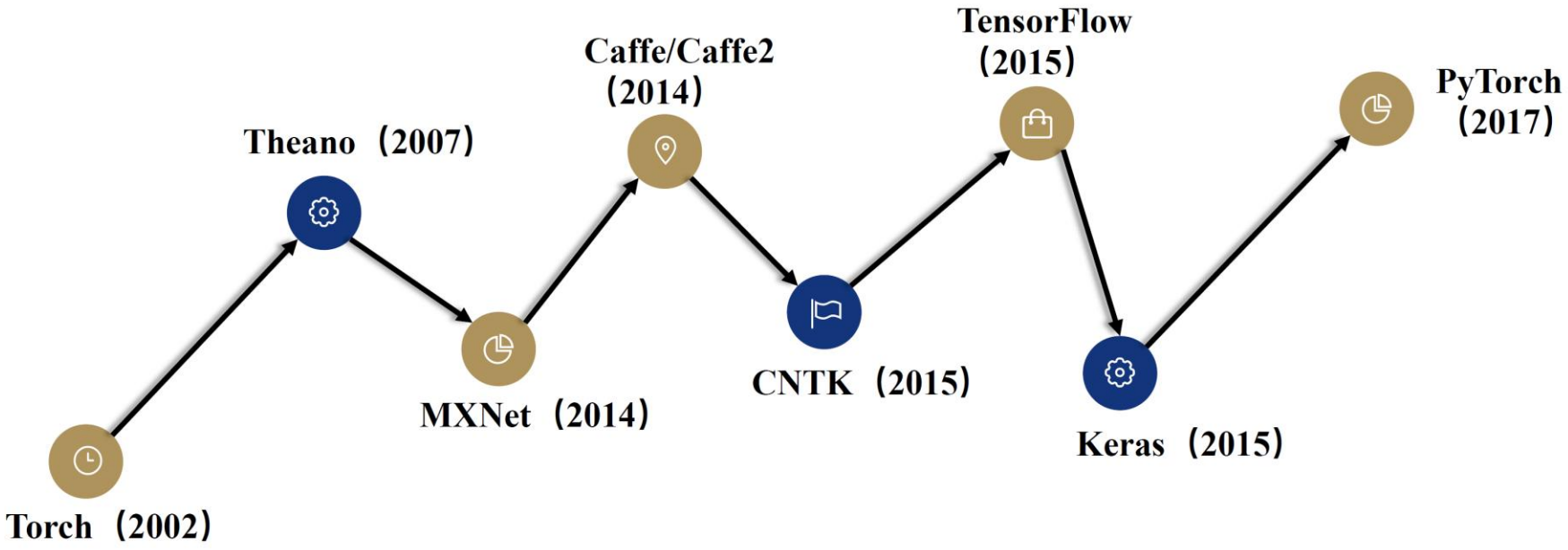




核心组件-基于张量的操作

- **张量的相关操作：**
  - 类型转换：字符串转为数字、转为 64（32）位浮点类型（整型）等。
  - 数值操作：按指定类型与形状生成张量、正态（均匀）分布随机数、设置随机数种子。
  - 形状变换：将数据的 shape 按照指定形状变化、插入维度、将指定维度去掉等。
  - 数据操作：切片操作、连接操作等。
  - 算术操作：求和、减法、取模、三角函数等。
  - 矩阵相关的操作：返回给定对角值的对角tensor 、对输入进行反转、矩阵相乘、求行列式...

发展历程


Torch（2002）
Theano（2007）
MXNet（2014）
Caffe/Caffe2
（2014）
CNTK（2015）
TensorFlow
（2015）
Keras（2015）
PyTorch
（2017）

发展历程

下图是各个主流开源框架在Github上的数据统计（2020年5月）。

框架	机构	支持语言	Stars	Forks	Contributors
Torch	Facebook	Lua	8.5K	2.4K	130
Theano	U.Montreal	Python	9.1K	2.5K	332
MXNet	DMLC	Python/C++/…	18.7K	6.6K	794
Caffe	BVLC	C++/Python	30.3K	18.3K	265
CNTK	Microsoft	C++	16.8K	4.4K	199
TensorFlow	Google	Python/C++/…	145K	81.3K	2496
Keras	fchollet	Python	48.3K	18.3K	817
PyTorch	Facebook	Python/C++/…	38.9K	10K	1406

主流框架介绍

- **深度学习的主流框架：**
  - Caffe
  - Theano
  - Torch & PyTorch
  - TensorFlow
  - Keras
  - MXNet

主流框架介绍

Caffe：

- Caffe 的全称是 Convolutional Architecture for Fast Feature Embedding 它是一个 清晰 、高效 的深度学习框架于2013 年底由加州大学伯克利分校开发核心语言是C++ 。 它支持命令行 、 Python和MATLAB接口 。 Caffe的一个重要特色是可以在不编写代码的情况下训练和部署模型 。
- https://github.com/BVLC/caffe
- Caffe2是由 Facebook组织开发的一个**轻量级**的深度学习框架，具有**模块化和可扩展性**等特点 。它在原来的 Caffe 的基础上进行改进提高了它的表达性，速度和模块化现在被并入Pytorch项目 。
- Caffe 曾经名噪一时，但由于使用不灵活 、代码冗长 、安装困难 、不适用构建循环网络等问题，已经很少被使用 。
- https://github.com/caffe2

主流框架介绍

Theano：

- Theano 是深度学习框架的鼻祖 由 Yoshua Bengio 和蒙特利尔大学的研究小组于 2007 年创建 是率先广泛使用的深度学习框架 。 Theano 是一个 Python 库，速度更快，功能强大，可以高效的进行数值表达和计算，可以说是从NumPy 矩阵表达向tensor表达的一次跨越，为后来的深度学习框架提供了样板 。 遗憾的是 Theano 团队 2017 年已经停止了该项目的更新深度学习应用框架的发展进入到了背靠工业界大规模应用的阶段 。
- 原始的 Theano 只有比较低级的 API；大型模型可能需要 很长的编译时间不支持多GPU ； 有的错误信息的提示没有帮助。

主流框架介绍

Torch&PyTorch：

- Torch 是一个有大量机器学习算法支持的科学计算框架其诞生已经有十余年，但真正起势得益于Facebook开源了大量Torch的深度学习模块和扩展 。Torch的一个特殊之处是采用了Lua编程语言。

- PyTorch 于 2016 年 10 月发布 是一款专注于**直接处理数组表达式**的低级API 。前身是Torch 。Facebook 人工智能研究院对 PyTorch 提供了强力支持 。PyTorch **支持动态计算图，** 为更具数学倾向的用户提供了**更低层次**的方法和更多的灵活性，目前许多新发表的论文都采用 PyTorch 作为论文实现的工具，**成为学术研究的首选解决方案** 。

主流框架介绍

- **PyTorch优点：**
  - 简洁易用
  - 可以为使用者提供更多关于深度学习实现的细节，如反向传播和其他训练过程
  - 活跃的社区：提供完整的文档和指南
  - 代码很Pythonic（简洁、优雅）
- **PyTorch缺点：**
  - 无可视化接口和工具
  - 导出模型不可移植，工业部署不成熟
  - 代码冗余量较大

主流框架介绍

TensorFlow：

- TensorFlow最初由Google Brain团队针对机器学习和深度神经网络进行研究所开发的，目前开源之后可以在几乎各种领域适用。它灵活的架构可以部署在一个或多个CPU、GPU的台式及服务器中，或者使用单一的API应用在移动设备中。TensorFlow可以说是当今十分流行的深度学习框架，Airbnb、DeepMind、Intel、Nvidia、Twitter以及许多其他著名公司都在使用它。**TensorFlow提供全面的服务，构建了活跃的社区，完善的文档体系，大大降低了我们的学习成本，另外，TensorFlow有很直观的计算图可视化呈现。模型能够快速的部署在各种硬件机器上，从高性能的计算机到移动设备，再到更小的更轻量的智能终端。**
- 但是，TensorFlow相比Pytorch，Caffe等框架，计算速度很慢。而且通过它构建一个深度学习框架需要更复杂的代码，还要忍受重复的多次构建静态图。

主流框架介绍

Tensorflow 优点：

- 自带 tensorboard 可视化工具，能够让用户**实时监控观察训练过程**
- 拥有大量的开发者，有详细的说明文档、可查询资料多
- 支持**多GPU 、分布式训练，跨平台运行能力强**
- 具备不局限于深度学习的多种用途， **还有支持强化学习和其他算法的工具**

Tensorflow 缺点：

- 频繁变动的接口
- 接口设计过于晦涩难懂

主流框架介绍

Keras：

- Keras于2015年3月首次发布，拥有“为人类而不是机器设计的API”，由Google的Francis Chollet创建与维护，它是一个用于快速构建深度学习原型的**高层神经网络库**，由纯Python编写而成，以TensorFlow，CNTK，Theano和MXNet为底层引擎，提供简单易用的API接口，能够极大地减少一般应用下用户的工作量。能够和TensorFlow，CNTK或Theano配合使用。
- 通过Keras的API可以仅使用数行代码就构建一个网络模型，Keras+Theano，Keras+CNTK的模式曾经深得开发者喜爱。目前Keras整套架构已经封装进了TensorFlow，在TF.keras可以完成Keras的所有事情。
- https://keras.io/

主流框架介绍

Keras 优点：

- 更简洁，更简单的API
- 丰富的教程 和可重复使用的代码
- 更多的部署选项（直接并且通过 TensorFlow 后端），更简单的模型导出
- 支持多GPU训练

Keras 缺点：

- 过度封装导致丧失灵活性，导致用户在新增操作或是获取底层的数据信息时过于困难
- 许多BUG都隐藏于封装之中，无法调试细节
- 初学者容易依赖于Keras的易使用性而忽略底层原理

主流框架介绍

MXNet:

- MXNet 于 2014 年由上海交大校友陈天奇与李沐组建团队开发,2017 年 1 月 MXNet 项目进入Apache 基金会,成为 Apache 的孵化器项目 。 MXNet 主要用 C++ 编写，强调提高内存使用的效率，甚至能在智能手机上运行诸如图像识别等任务 。
- 它拥有类似于 Theano 和 TensorFlow 的数据流图，为多 GPU 配置提供了良好的配置，还有着类似于 Blocks 等更高级别的模型构建块，并且可以在任何硬件上运行（包括手机）。
- 同时，MXNet 是一个旨在提高效率和灵活性的深度学习框架 提供了强大的工具来帮助开发人员利用 GPU 和云计算的全部功能 。
- https://mxnet.apache.org/

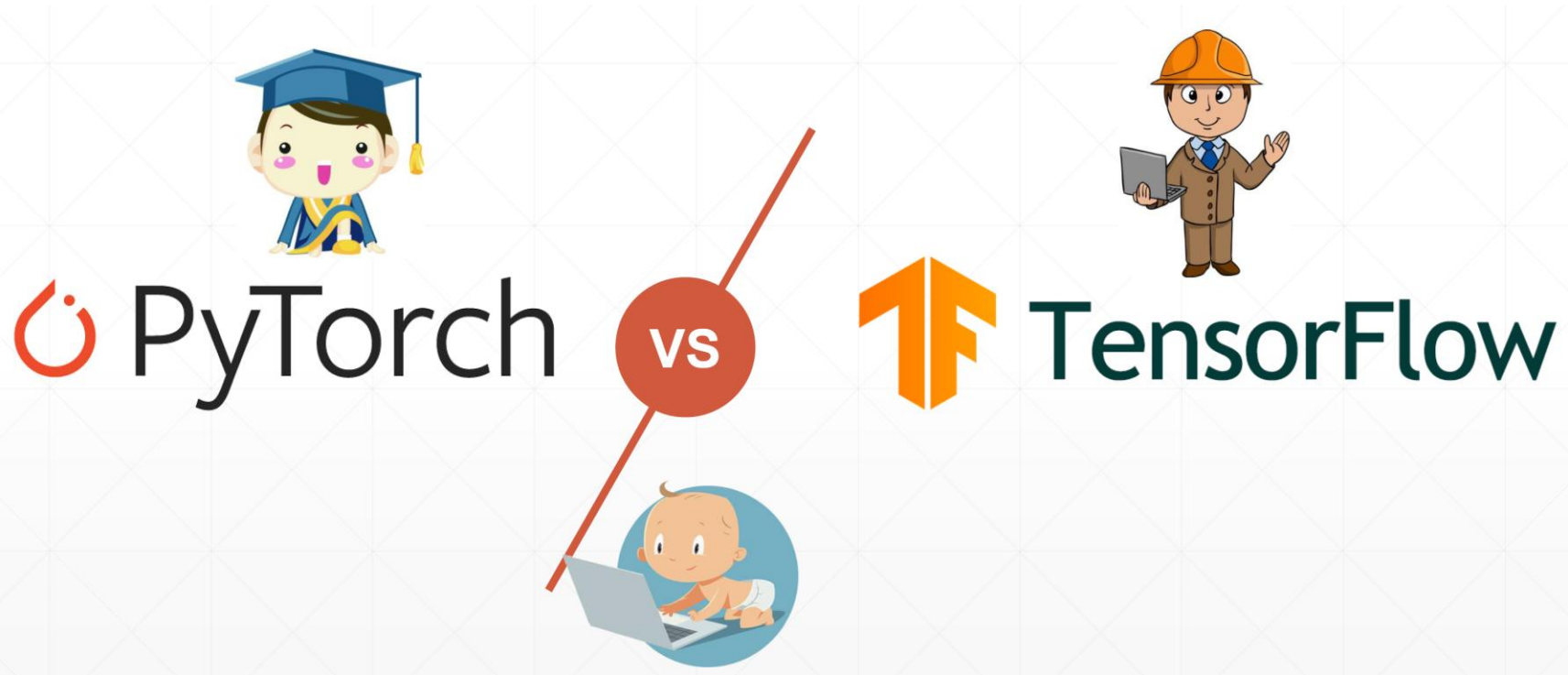
主流框架介绍

- **MXNet 优点：**
  - 灵活的编程模型
  - 从云端到客户端可移植
  - 多语言支持
  - 本地分布式训练
  - 性能优化
- **MXNet 缺点：**
  - 社区相对 PyTorch 和 TensorFlow 来说相对小众

Tensorflow VS Pytorch

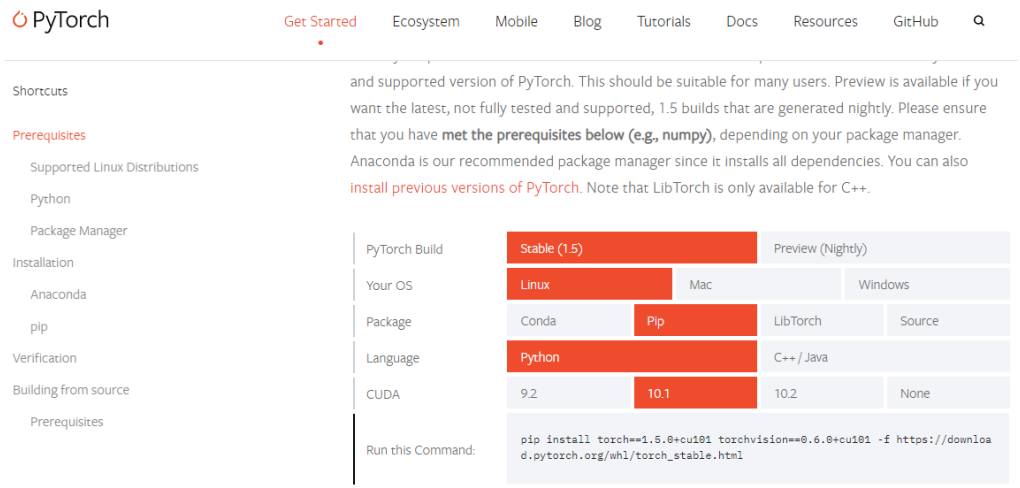
- Pytorch和numpy的转换方便，动态图的特性导致其语法更加自然流畅，适合初学者学习。
- Tensorflow虽然在2.0版本后支持了动态图，并且利用keras可以极大简化代码，但是语法自然程度弱于pytorch。
- Github中早期经典代码大部分为tensorflow版本，或者theano等语言版本，pytorch的相对较少。
- 近期的工作中，使用pytorch撰写代码的研究者逐渐增多。
- Tensorflow更方便于多gpu部署，嵌入式设备的部署等， 并且拥有tensorboard等工具，便于研究者的开发调试。

Tensorflow VS Pytorch



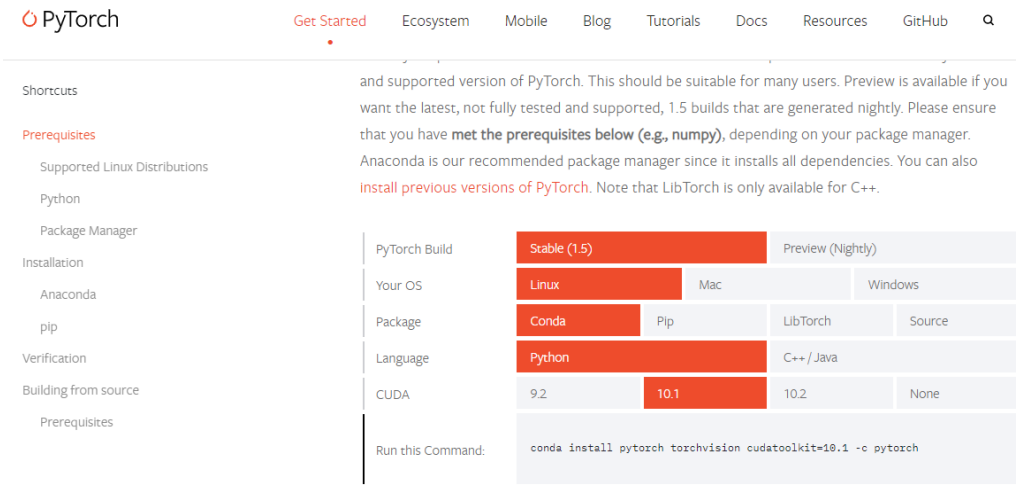
PyTorch入门-安装

- **使用 pip 安装：**
  - 目前使用 pip 安装 PyTorch 二进制包是最简单 、最不容易出错，同时也是最适合新手的安装方式 。 从 PyTorch 官网选择**操作系统 、包管理器 pip 、Python版本及CUDA 版本**，会对应不同的安装命令，如图所示：https ://pytorch.org/get-started/locally/



PyTorch入门-安装

- **使用 conda 安装：**
  - conda是 Anaconda 自带的包管理器 。 如果使用 Anaconda 作为 Python 环境，则除了使用 pip 安装还可以使用 conda 进行安装 。 同样，在 PyTorch 官网中选择**操作系统 、 包管理器 conda 、 Python 版本及 CUDA 版本**对应不同的安装命令 。 如图所示：



Pytorch基本数据类型

- 和其他深度学习框架类似，Pytorch通过tensor构建出计算图，常见的tensor类型如右图所示
- 一般的，用于训练模型的数据是float类型，而作为label的数据则是long类型
- 利用dtype属性可以进行数据的类型转换

Data type	dtype	CPU tensor	GPU tensor
32-bit floating point	`torch.float32` or `torch.float`	`torch.FloatTensor`	`torch.cuda.FloatTensor`
64-bit floating point	`torch.float64` or `torch.double`	`torch.DoubleTensor`	`torch.cuda.DoubleTensor`
16-bit floating point	`torch.float16` or `torch.half`	`torch.HalfTensor`	`torch.cuda.HalfTensor`
8-bit integer (unsigned)	`torch.uint8`	`torch.ByteTensor`	`torch.cuda.ByteTensor`
8-bit integer (signed)	`torch.int8`	`torch.CharTensor`	`torch.cuda.CharTensor`
16-bit integer (signed)	`torch.int16` or `torch.short`	`torch.ShortTensor`	`torch.cuda.ShortTensor`
32-bit integer (signed)	`torch.int32` or `torch.int`	`torch.IntTensor`	`torch.cuda.IntTensor`
64-bit integer (signed)	`torch.int64` or `torch.long`	`torch.LongTensor`	`torch.cuda.LongTensor`
Boolean	`torch.bool`	`torch.BoolTensor`	`torch.cuda.BoolTensor`

Pytorch tensor的使用

- 利用torch.tensor()进行创建
- 利用内置的函数例如 torch.ones() 进行创建

```
# In[47]:
double_points = torch.ones(10, 2, dtype=torch.double)
short_points = torch.tensor([[1, 2], [3, 4]], dtype=torch.short)
```

- 将numpy数组转换成 tensor 或者将tensor 转换成numpy数组

```
# In[55]:
points = torch.ones(3, 4)
points_np = points.numpy()
points_np

# Out[55]:
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]], dtype=float32)

# In[56]:
points = torch.from_numpy(points_np)
```

- 转换

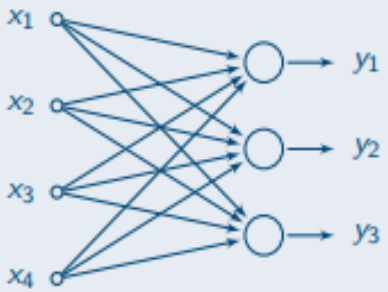
```
# In[49]:
double_points = torch.zeros(10, 2).double()
short_points = torch.ones(10, 2).short()

# In[50]:
double_points = torch.zeros(10, 2).to(torch.double)
short_points = torch.ones(10, 2).to(dtype=torch.short)
```

-

```
# In[64]:
points_gpu = torch.tensor([[1.0, 4.0], [2.0, 1.0], [3.0, 4.0]],
    device='cuda')

# In[65]:
points_gpu = points.to(device='cuda')
```

Fully Connected

Layer



$$\mathbf{y} = f(\mathbf{W}^T \cdot \mathbf{x} + \mathbf{b})$$

CLASS `torch.nn.Linear(in_features, out_features, bias=True)` [SOURCE]

Applies a linear transformation to the incoming data: $y = xA^T + b$

Parameters:

- **in_features** – size of each input sample
- **out_features** – size of each output sample
- **bias** – If set to False, the layer will not learn an additive bias. Default: `True`

Shape:

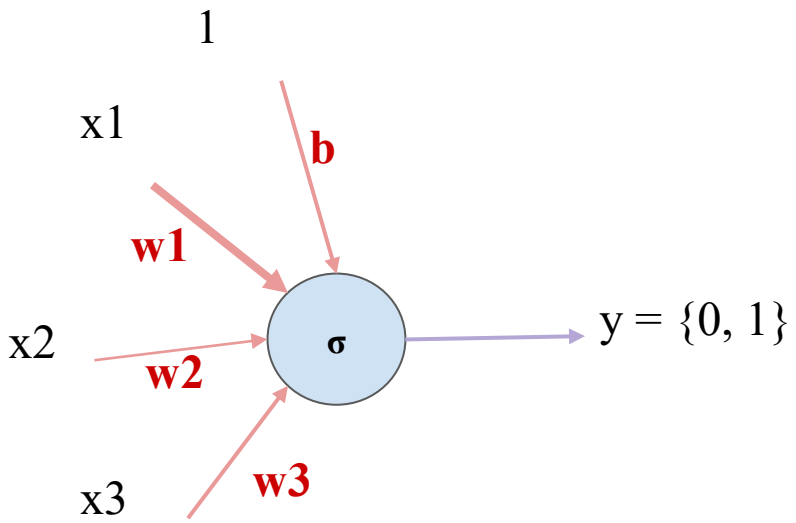
- Input: $(N, *, \text{in\_features})$ where $*$ means any number of additional dimensions
- Output: $(N, *, \text{out\_features})$ where all but the last dimension are the same shape as the input.

Variables:

- **weight** – the learnable weights of the module of shape $(\text{out\_features}, \text{in\_features})$. The values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{\text{in\_features}}$
- **bias** – the learnable bias of the module of shape (out_features). If `bias` is `True`, the values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{1}{\text{in\_features}}$

Fully Connected: A perceptron

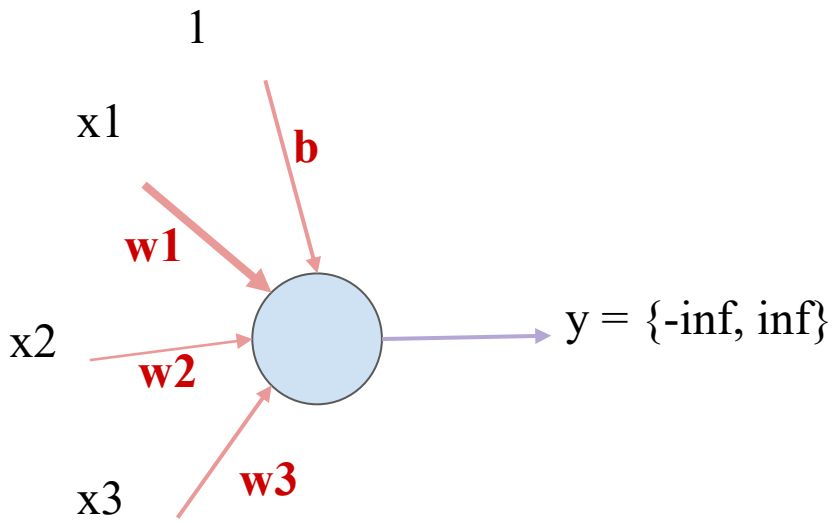
- Fully connected layer with one unit. A sigmoid activation makes it a **logistic regression** (binary linear classifier):




```
lor = nn.Sequential(
    nn.Linear(NUM_INPUTS, 1),
    nn.Sigmoid()
)
```

Fully Connected: A perceptron

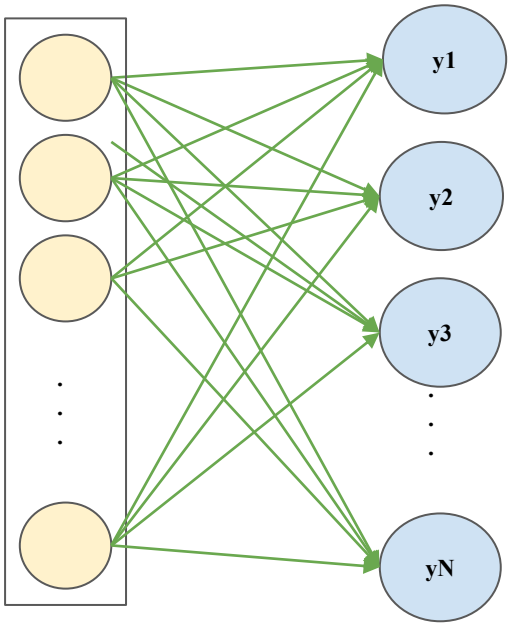
- Fully connected layer with one unit. No activation makes it a **linear regression**:




```
lir = nn.Sequential(
    nn.Linear(NUM_INPUTS, 1)
)
```

Fully Connected: Multiclass classifier

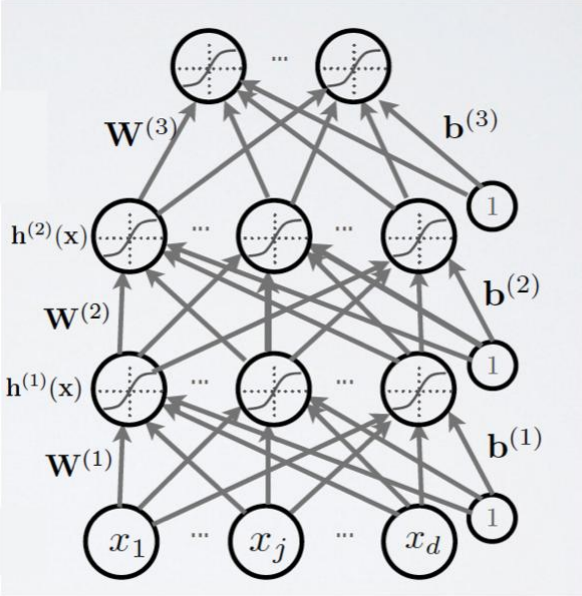
- Fully connected layer with many units. Softmax activation makes it a "softmax classifier" .




```
smx = nn.Sequential(
    nn.Linear(NUM_INPUTS, NUM_OUTPUTS),
    nn.LogSoftmax(dim=1)
)
```

Fully Connected: MultiLayer Perceptron

- Many fully connected layers with many units.



Slide Credit: Hugo Laroche NN course

```
NUM_INPUTS=100
HIDDEN_SIZE=1024
NUM_OUTPUTS=20

mlp = nn.Sequential(
    nn.Linear(NUM_INPUTS, HIDDEN_SIZE),
    nn.Tanh(),
    nn.Linear(HIDDEN_SIZE, HIDDEN_SIZE),
    nn.Tanh(),
    nn.Linear(HIDDEN_SIZE, NUM_OUTPUTS),
    nn.LogSoftmax(dim=1)
)
```

Pytorch 神经网络模型接口

- **激活函数：**
  - sigmoid, relu, leaky_relu, elu, selu, glu, gelu, logsigmoid, softsign, softplus, softmin, softmax, tanh等
  - 一般利用torch.nn.functinoal中的函数进行非线性变换
- **神经网络层：**
  - ConvolutionLayers, PoolingLayers, NormalizationLayers, RecurrentLayers, LinearLayers, DropoutLayers, PaddingLayers等常见的网络连接方式
  - 一般通过torch.nn接口，利用已有的模型进行网络搭建
- **损失函数：**
  - L1Loss, MSELoss, CrossEntropyLoss, NLLLoss, KLDivLoss, BCELoss等
- **优化算法：**
  - Adadelta, Adagrad, Adam, SGD, RMSProp等
- **Pytorch提供了从底层到顶层的接口，可以方便验证自己的想法**

Pytorch 神经网络模型接口

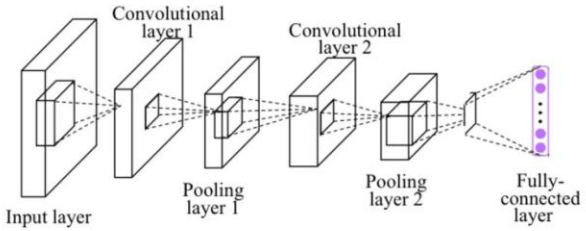
- **Torch.nn的核心数据结构是Module，它既可以表示神经网络中的某个层也可以表示一个包含很多层的神经网络。 最常见的做法就是继承nn.Module编写自己的网络 。**
  - 能够自动检测到自己的 parameter 并将其作为学习参数 。
  - 主 Module 能够递归查找子 Module 中的 parameter 。
- 为了方便用户使用，PyTorch 实现了神经网络中绝大多数的 layer 这些layer都继承于nn.Module 封装了可学习参数parameter并实现了forward函数， 且专门针对GPU运算进行了CuDNN优化，其速度和性能都十分优异。

可学习的参数，不可学习的参数（超参数）？

Pytorch 神经网络模型接口

- **实现一个自定义层大致分以下几个主要的步骤：**
- 1）自定义一个类，继承Module类，并且一定要实现两个基本函数
  - 构造函数_init_
  - 前向计算函数 forward 函数
- 2) 在构造函数 init 中实现层的参数定义 。
  - 例如：Linear层的权重和偏置
- 3）在前向传播forward函数里面实现前向运算。
  - 通过torch.nn.functional 函数来实现
  - 自定义自己的运算方式。 如果该层含有权重，那么权重必须是 nn.Parameter 类型
- 4）补充：一般情况下，我们定义的参数是可以求导的，但是自定义操作如不可导，需要实现 backward 函数。

Pytorch 神经网络模型接口

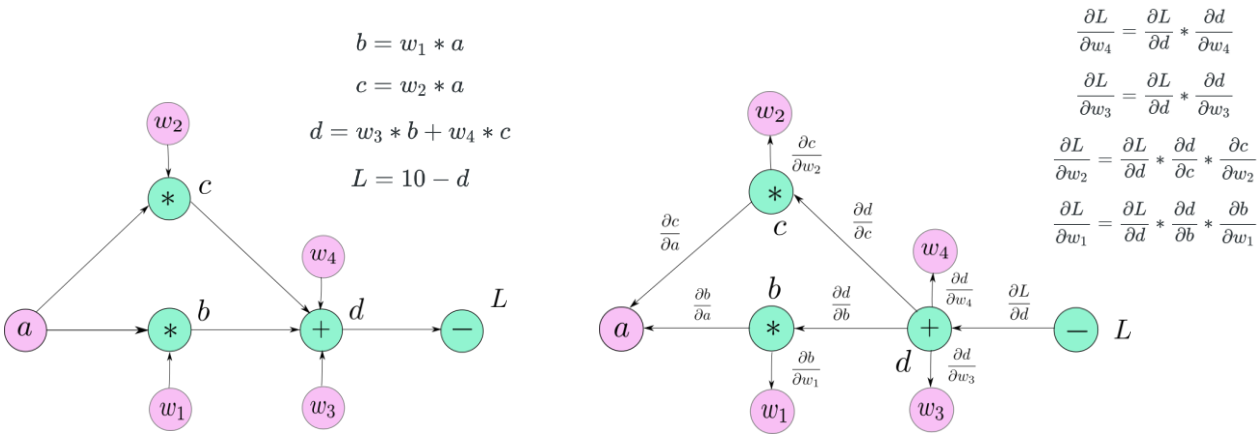



```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(320, 10) # 320 -> 10

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = F.relu(self.mp(self.conv2(x)))
        x = x.view(in_size, -1) # flatten the tensor
        x = self.fc(x)
        return F.log_softmax(x)
```

Pytorch, zero to all. HKUST

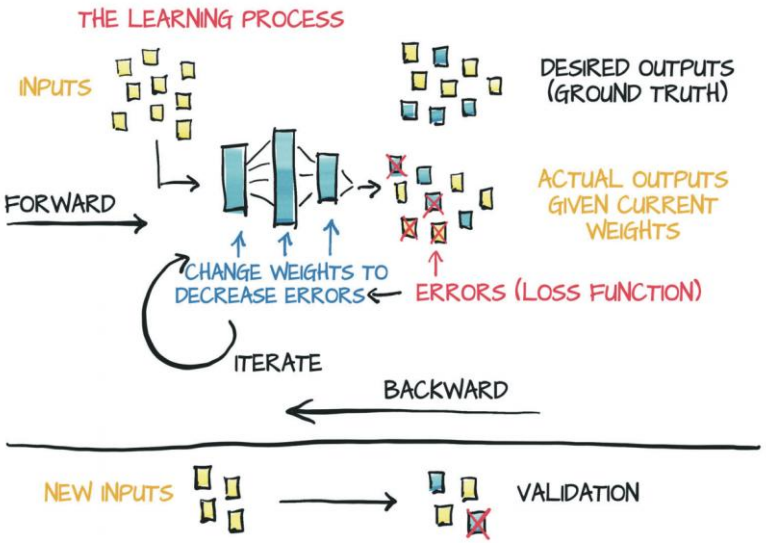
Pytorch自动求导简介

- **深度学习库的核心：自动求导（autograd）**
  - 在没有深度学习库时，既要实现前向过程，又要实现反向传播。层数越多，越容易出现错误。
  - torch.autograd 是为了方便用户使用,专门开发的一套**自动求导引擎**，它能够根据**输入**和**前向传播**过程自动构建计算图，并执行**反向传播**。
- **利用计算图实现自动求导：将每一个变量节点的导数进行保存，利用链式求导法则进行导数的计算。**

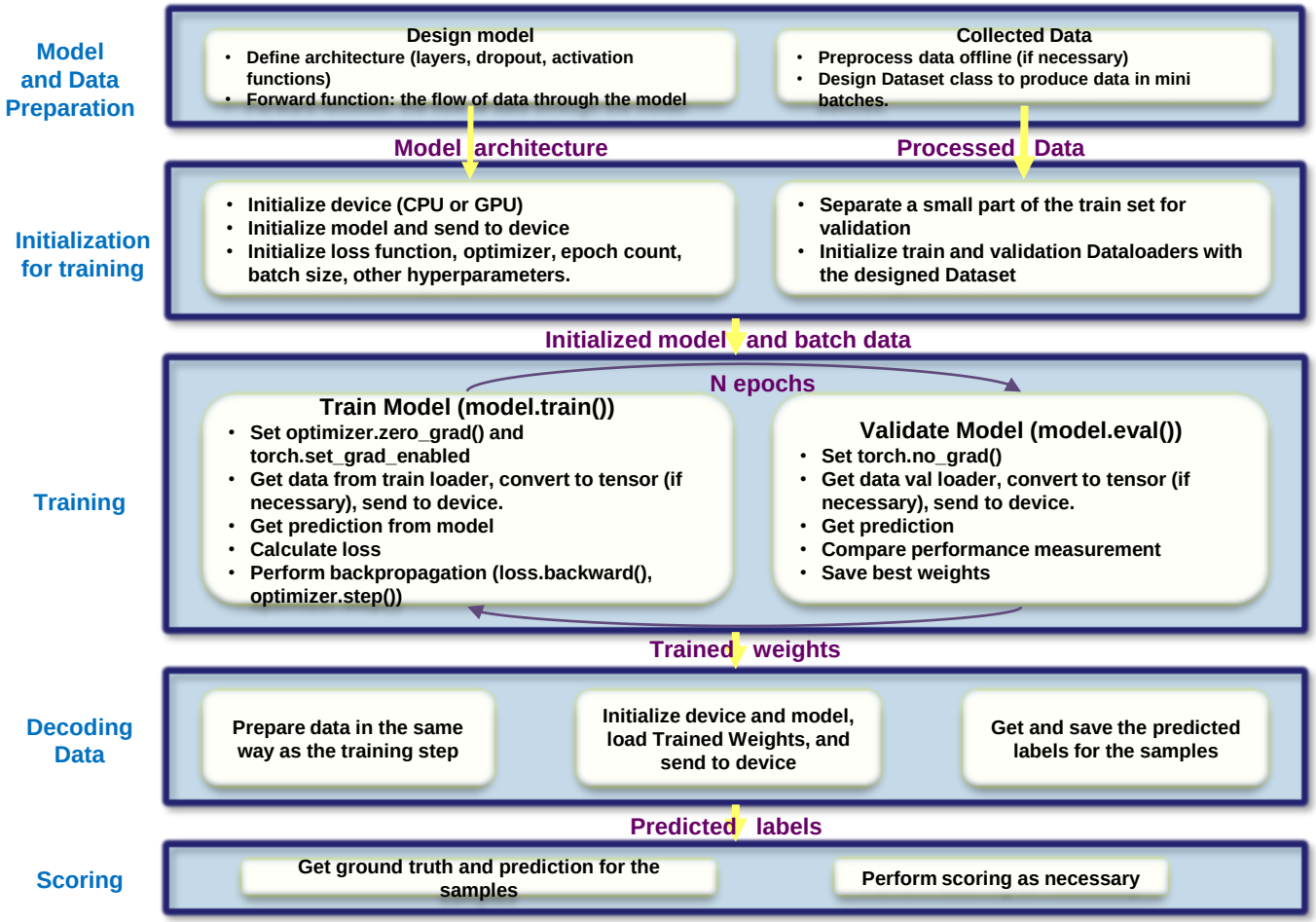


Pytorch模型构建整体流程

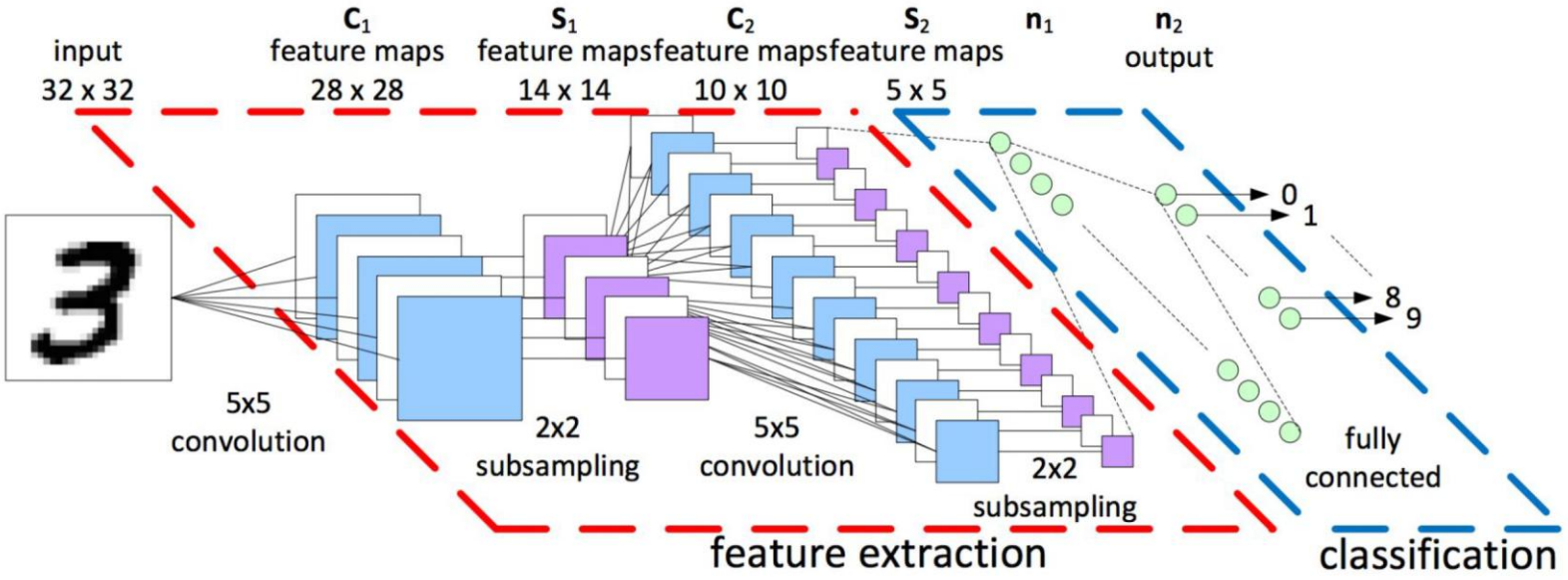
- **数据导入：**
  - data, target
  - device
- **模型构建：**
  - 模型结构设计
  - Layers
- **模型训练：**
  - 损失函数
  - 优化算法
  - epoch
  - 自动求导
- **模型验证：**
  - 泛化能力



Pytorch 深度学习流程



训练一个简单的神经网络



1. Forward: compute output of each layer
2. Backward: compute gradient
3. Update: update the parameters with computed gradient

训练一个简单的神经网络

PyTorch: nn

Define our model as a sequence of layers

nn also defines common loss functions

```
import torch
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10
x = Variable(torch.randn(N, D_in))
y = Variable(torch.randn(N, D_out), requires_grad=False)

model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out))
loss_fn = torch.nn.MSELoss(size_average=False)

learning_rate = 1e-4
for t in range(500):
    y_pred = model(x)
    loss = loss_fn(y_pred, y)

    model.zero_grad()
    loss.backward()

    for param in model.parameters():
        param.data -= learning_rate * param.grad.data
```

Stanford cs231n.

训练一个简单的神经网络

PyTorch: nn

Define our model as a sequence of layers

nn also defines common loss functions

```
import torch
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10
x = Variable(torch.randn(N, D_in))
y = Variable(torch.randn(N, D_out), requires_grad=False)

model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out))
loss_fn = torch.nn.MSELoss(size_average=False)

learning_rate = 1e-4
for t in range(500):
    y_pred = model(x)
    loss = loss_fn(y_pred, y)

    model.zero_grad()
    loss.backward()

    for param in model.parameters():
        param.data -= learning_rate * param.grad.data
```

Stanford cs231n.

训练一个简单的神经网络

PyTorch: nn

Forward pass: feed data to model, and prediction to loss function

```
import torch
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10
x = Variable(torch.randn(N, D_in))
y = Variable(torch.randn(N, D_out), requires_grad=False

model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out))
loss_fn = torch.nn.MSELoss(size_average=False)

learning_rate = 1e-4
for t in range(500):
    y_pred = model(x)
    loss = loss_fn(y_pred, y)

    model.zero_grad()
    loss.backward()

    for param in model.parameters():
        param.data -= learning_rate * param.grad.data
```

Stanford cs231n.

训练一个简单的神经网络

PyTorch: nn

```
import torch
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10
x = Variable(torch.randn(N, D_in))
y = Variable(torch.randn(N, D_out), requires_grad=False)

model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out))
loss_fn = torch.nn.MSELoss(size_average=False)

learning_rate = 1e-4
for t in range(500):
    y_pred = model(x)
    loss = loss_fn(y_pred, y)

    model.zero_grad()
    loss.backward()

    for param in model.parameters():
        param.data -= learning_rate * param.grad.data
```

Backward pass:
compute all gradients

Stanford cs231n.

训练一个简单的神经网络

PyTorch: nn

```
import torch
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10
x = Variable(torch.randn(N, D_in))
y = Variable(torch.randn(N, D_out), requires_grad=False)

model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out))
loss_fn = torch.nn.MSELoss(size_average=False)

learning_rate = 1e-4
for t in range(500):
    y_pred = model(x)
    loss = loss_fn(y_pred, y)

    model.zero_grad()
    loss.backward()

    for param in model.parameters():
        param.data -= learning_rate * param.grad.data
```

Make gradient step on each model parameter

Stanford cs231n.

训练一个简单的神经网络

PyTorch: optim

Use an **optimizer** for different update rules

```
import torch
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10

x = Variable(torch.randn(N, D_in))
y = Variable(torch.randn(N, D_out), requires_grad=False)

model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out))
loss_fn = torch.nn.MSELoss(size_average=False)

learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(),
                             lr=learning_rate)
for t in range(500):
    y_pred = model(x)
    loss = loss_fn(y_pred, y)

    optimizer.zero_grad()
    loss.backward()

    optimizer.step()
```

Stanford cs231n.

训练一个简单的神经网络

PyTorch: optim

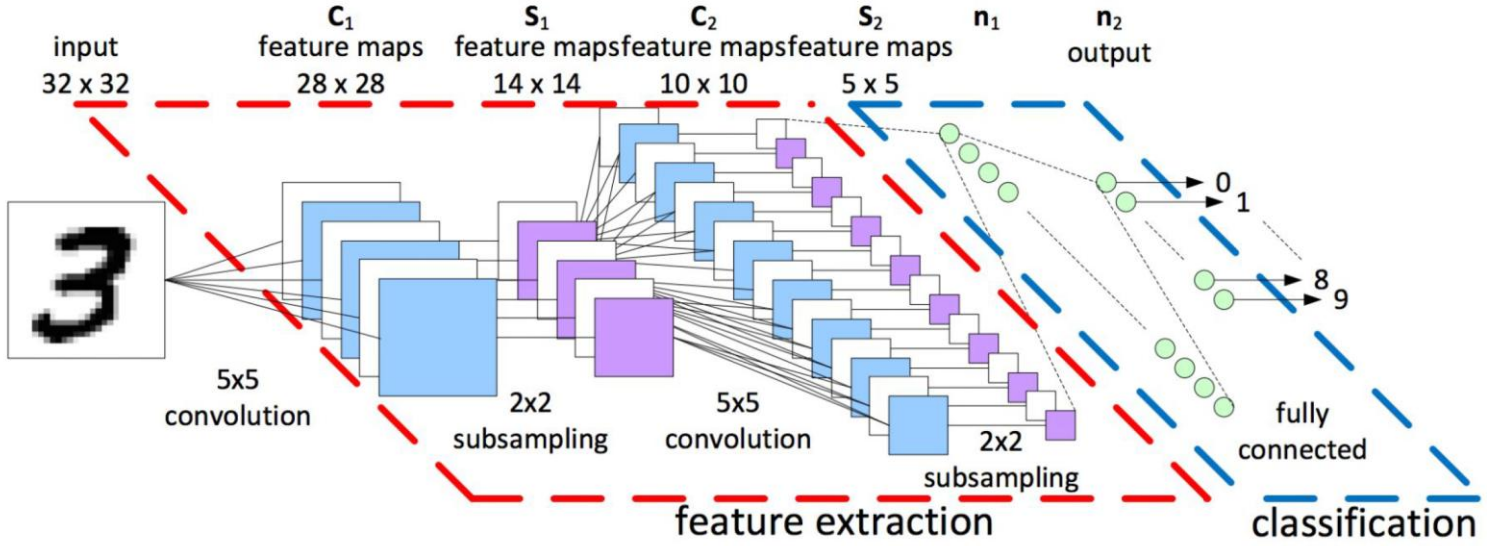
```
import torch
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10

x = Variable(torch.randn(N, D_in))
y = Variable(torch.randn(N, D_out), requires_grad=False)

model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out))
loss_fn = torch.nn.MSELoss(size_average=False)

learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(),
                             lr=learning_rate)
for t in range(500):
    y_pred = model(x)
    loss = loss_fn(y_pred, y)

    optimizer.zero_grad()
    loss.backward()

    optimizer.step()
```

Update all parameters after computing gradients

Stanford cs231n.

训练一个简单的神经网络

MNIST example



神经网络超参数调节

- **超参数类别：**
  - 网络设计相关的参数：网络层数、不同层的类别和搭建顺序、隐藏层神经元的参数设置、LOSS 的选择、正则化参数。
  - 训练过程相关的参数：网络权重初始化方法、学习率、迭代次数、小批量数据的规模。
- **超参数搜寻方法：网格搜索法，即根据主要需要调节的参数，一般为学习率、网络层数以及隐层节点等超参，设计网格，并逐一进行搜寻。**

神经网络超参数调节

Random Search vs. Grid Search

Random Search for Hyper-Parameter Optimization
Bergstra and Bengio, 2012

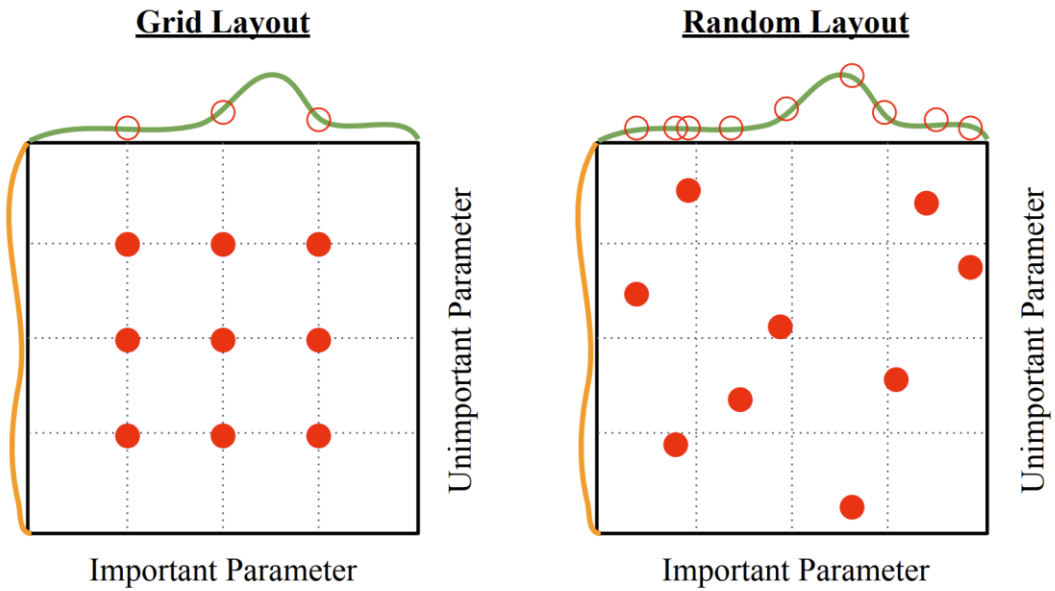


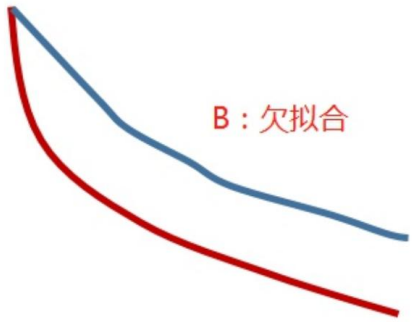
Illustration of Bergstra et al., 2012 by Shayne Longpre, copyright CS231n 2017

神经网络超参数调节

Step 1: Check initial loss
Step 2: Overfit a small sample
Step 3: Find LR that makes loss go down
Step 4: Coarse grid, train for ~1-5 epochs
Step 5: Refine grid, train longer
Step 6: Look at loss and accuracy curves
Step 7: GOTO step 5

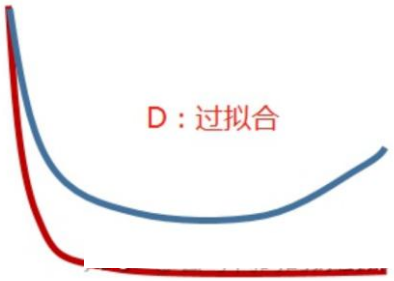
神经网络超参数调节-欠拟合

- 欠拟合：从LOSS曲线上看，训练集和验证集从趋势上还没有收敛。
- 可以考虑调节以下参数：
  - 加大训练迭代次数，有可能是网络还没训练完
  - 进一步衰减调小学习率
  - 添加更多的层，也有可能是网络容量不够
  - 去掉正则化约束的部分，l1、l2正则（正则主要是为了防止过拟合）
  - 加入BN层加快收敛速度
  - 增加网络的非线性度（ReLu）



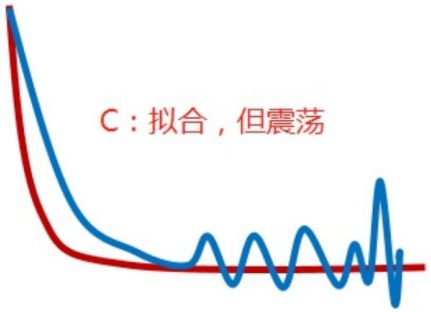
神经网络超参数调节-过拟合

- 过拟合：从LOSS曲线上看，都趋向收敛，但是验证集上的LOSS很高，甚至出现回升现象。说明模型泛化能力很差，有可能训练集和验证集数据分布不一致，更加可能的是模型太复杂。
- 可以考虑调节以下参数：
  - 增加样本数量，训练样本太少或者说小样本问题
  - 数据增强
  - 早停法（Early stopping），从LOSS不在下降的地方拿到模型，作为训练好的模型
  - 增加网络的稀疏度
  - 降低网络的复杂度（深度）
  - L1、 L2 regularization、Dropout
  - 适当降低Learning rate
  - 适当减少epoch的次数



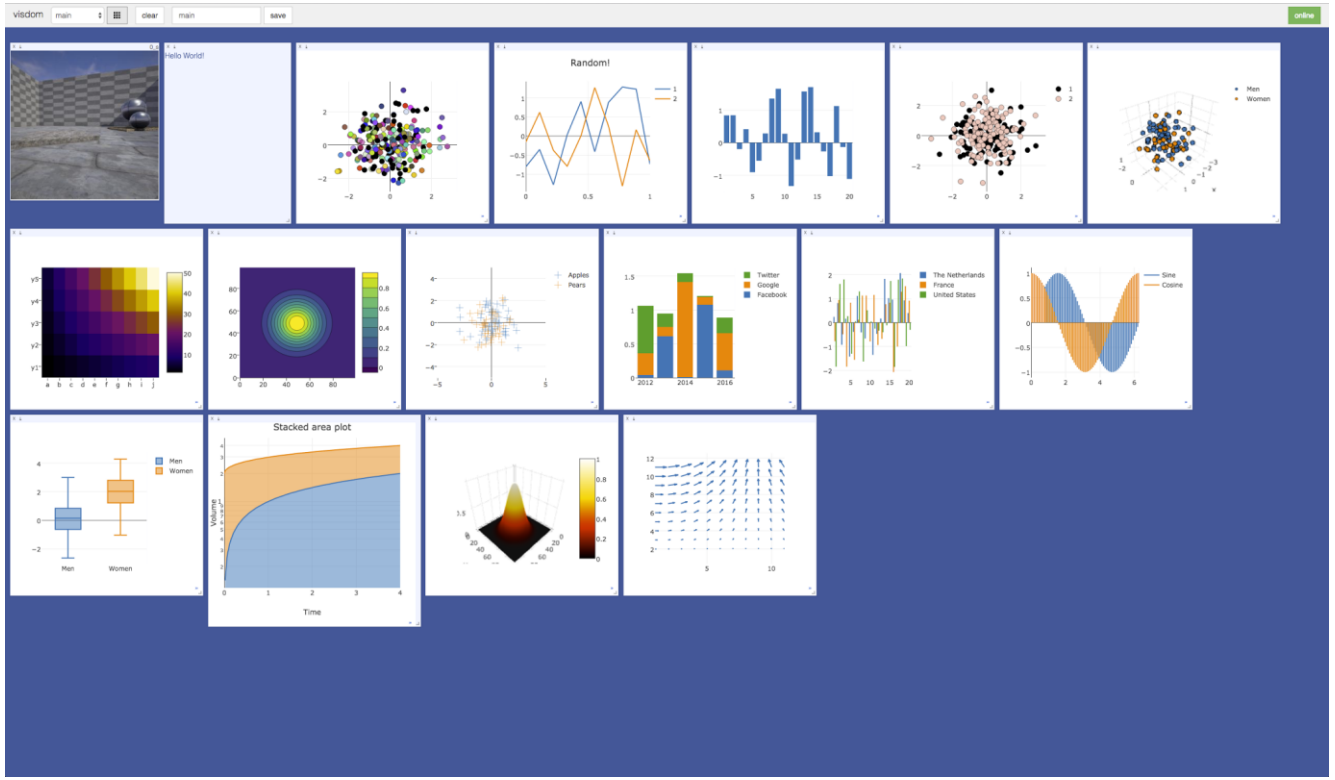
神经网络超参数调节-震荡

- 震荡：网络在训练集上已经趋于收敛，但是在验证集上存在很严重的LOSS震荡的情况。
- 可以考虑调节以下参数：
  - 训练集和验证集分布是否存在较大的差异
  - 是否由于数据增强做的太过分了
  - 学习率是不是还很高？如果是，降低学习率
  - 测试集LOSS的计算是基于单个batch还是整个验证集？（一定要基于整个验证集来看）
  - 网络是否存在欠拟合的可能，如果欠拟合参考欠拟合的方法。



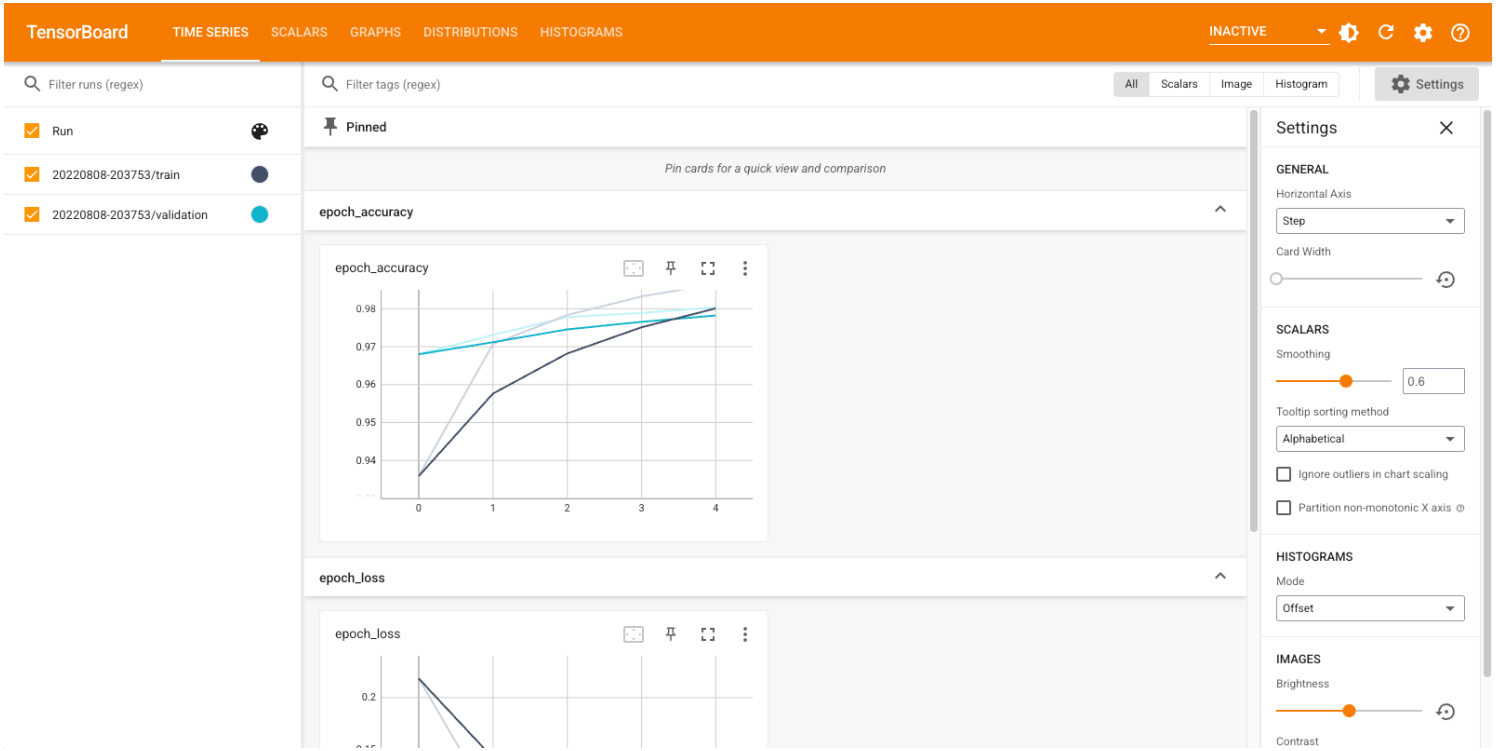
可视化-Pytorch Visdom

- https://github.com/facebookresearch/visdom

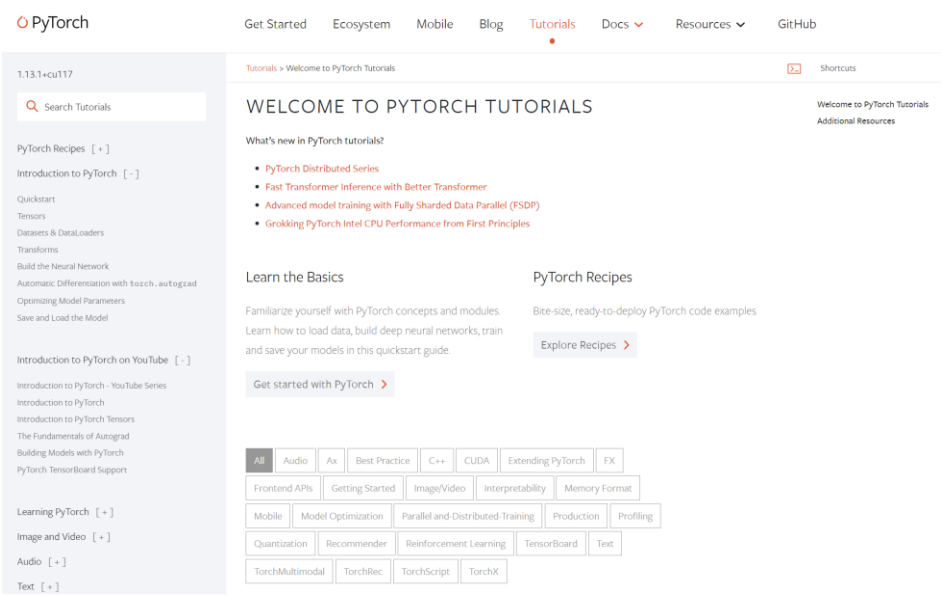


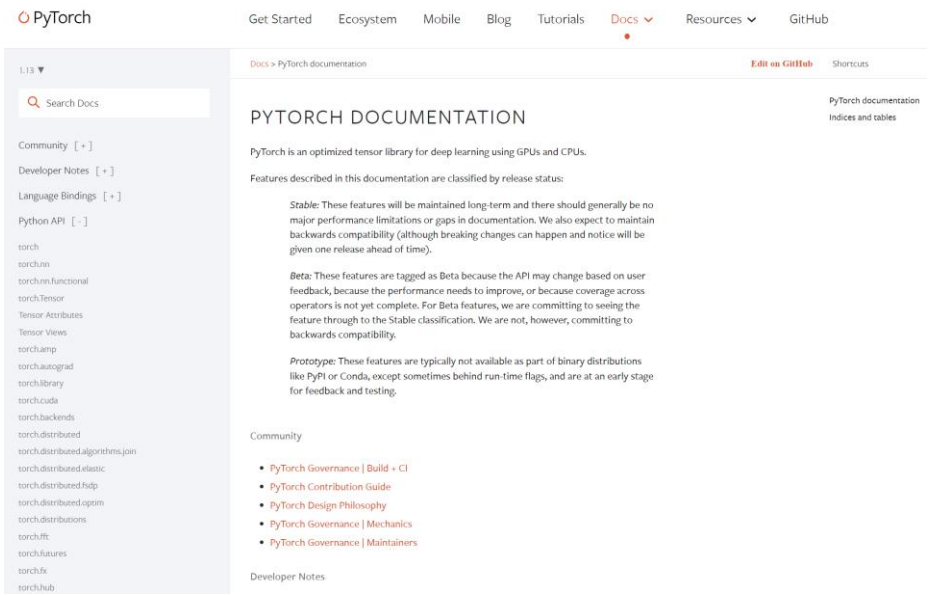
可视化-TensorBoard

- https://www.tensorflow.org/get_started/summaries_and_tensorboard



Pytorch官方资料



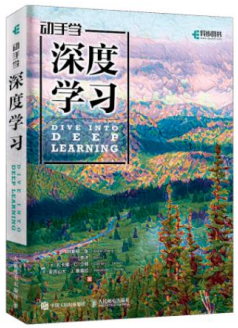


Google colab简介

- colab简介
  - 免费的GPU云服务器，内置Jupyter Notebook环境，可以帮助你快速开发深度学习应用
  - 限制：实例单次运行时间不能超过12小时
  - 支持PyTorch, Keras, TensorFlow, and OpenCV
  - 可以挂载你的Google Drive(网盘)，使用里面的文件
  - 你可以上传本机的notebooks、GitHub中的notebooks等

Pytorch 学习资料

- **官网教程**
  https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
- **动手学深度学习**
  https://github.com/ShusenTang/Dive-into-DL-PyTorch



+ PyTorch

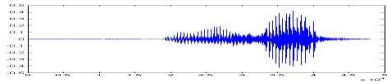
视频资料

https://space.bilibili.com/1567748478?spm_id_from=333.337.0.0

Outline of Lecture Three

- Deep Learning Framework
- **Basic Concepts in Machine Learning**
- Support Vector Machine
- Performance Evaluation Index

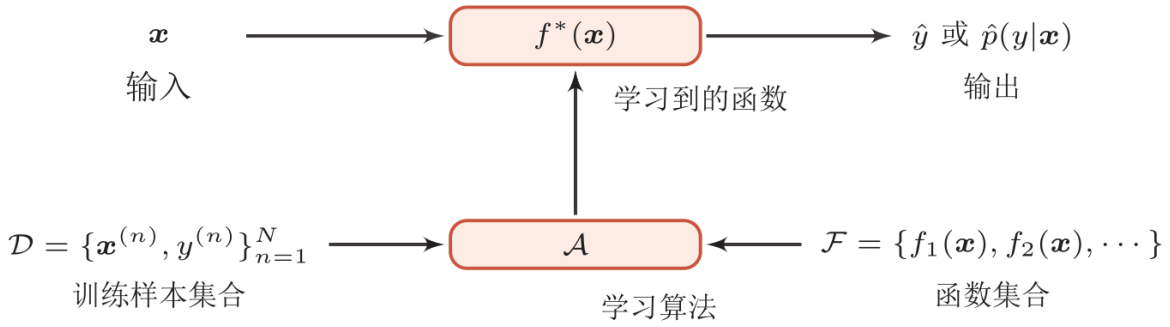
机器学习≈构建一个映射函数

- 语音识别 $f($  $)=$ “你好”
- 图像识别 $f($  $)=$ “猫”
- 围棋 $f($  $)=$ “5-5”（落子位置）
- 对话系统 $f($ “你好” $)=$ “今天天气真不错”

用户输入　　机器

什么是机器学习？

- 机器学习：通过算法使得机器能从大量数据中学习规律从而对新的样本做决策。
  - 规律：决策（预测）函数



独立同分布 p(x, y)

机器学习的三要素

- **模型**
  - **线性方法：** $f(\mathbf{x}, \theta) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b$
  - **广义线性方法：** $f(\mathbf{x}, \theta) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) + b$
    - **如果$\phi(x)$为可学习的非线性基函数，$f(x,\theta)$就等价于神经网络。**
- **学习准则**
  - **期望风险** $\mathcal{R}(f) = \mathbb{E}_{(\mathbf{x},y)\sim p(\mathbf{x},y)}[\mathcal{L}(f(\mathbf{x}), y)],$
- **优化**
  - **梯度下降**

学习准则

- **期望风险未知，通过经验风险近似**
  - **训练数据：** $\mathcal{D} = \{\mathrm{x}^{(\mathrm{n})}, y^{(\mathrm{n})}\}, i \in [1, N]$

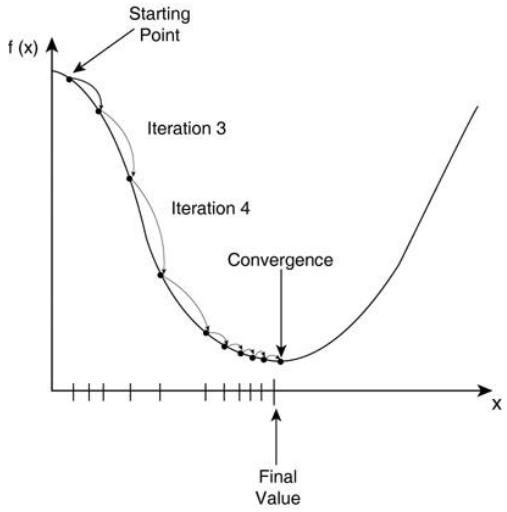
$$\mathcal{R}_{\mathcal{D}}^{emp}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(y^{(n)}, f(x^{(n)}, \theta))$$

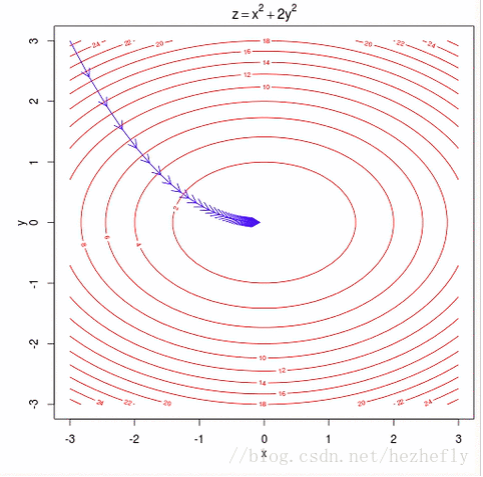
- **经验风险最小化**
  - **在选择合适的风险函数后，我们寻找一个参数θ∗，使得经验风险函数最小化。**

$$\theta^* = \arg\min_{\theta} \mathcal{R}_{\mathcal{D}}^{emp}(\theta)$$

- **机器学习问题转化成为一个最优化问题**

梯度下降法（Gradient Descent）



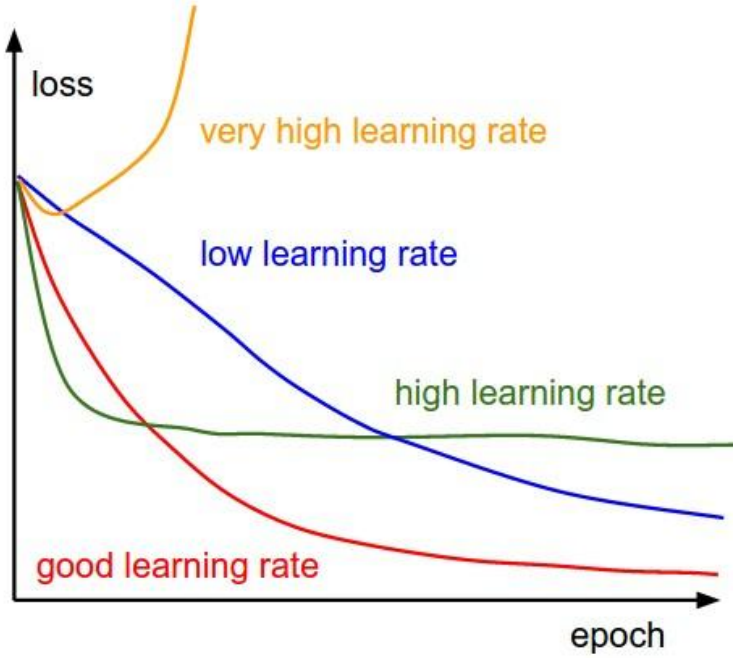


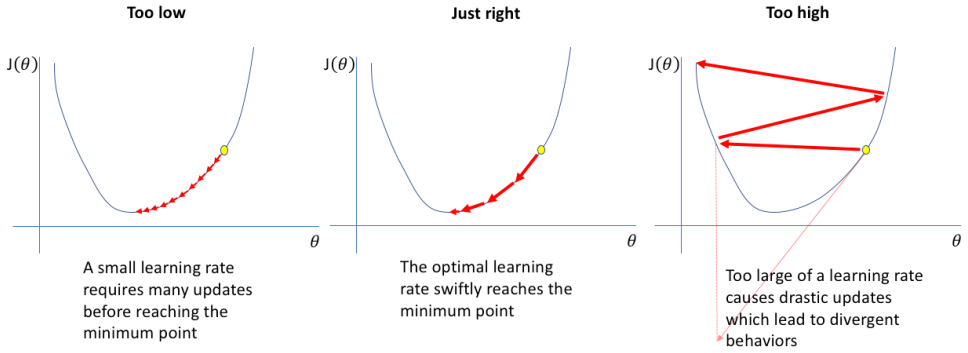
$$\theta_{t+1} = \theta_t - \alpha \frac{\partial \mathcal{R}(\theta)}{\partial \theta_t}$$

$$= \theta_t - \alpha \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathcal{L}\left(\theta_t; x^{(i)}, y^{(i)}\right)}{\partial \theta}.$$

搜索步长 α 中也叫作学习率（Learning Rate）

学习率是十分重要的超参数！





随机梯度下降法

- 随机梯度下降法（Stochastic Gradient Descent，SGD）也叫增量梯度下降，每个样本都进行更新

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial \mathcal{L}\left(\theta_t; x^{(t)}, y^{(t)}\right)}{\partial \theta},$$

- 小批量（Mini-Batch）随机梯度下降法

随机梯度下降法

算法 2.1: 随机梯度下降法

输入: 训练集 $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$，验证集 $\mathcal{V}$，学习率 α

随机初始化 θ;

repeat

  对训练集 $\mathcal{D}$ 中的样本随机重排序;

  for $n = 1 \cdots N$ **do**

    从训练集 $\mathcal{D}$ 中选取样本 $(\mathbf{x}^{(n)}, y^{(n)})$;

    // 更新参数

    $\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}(\theta; x^{(n)}, y^{(n)})}{\partial \theta}$;

  end

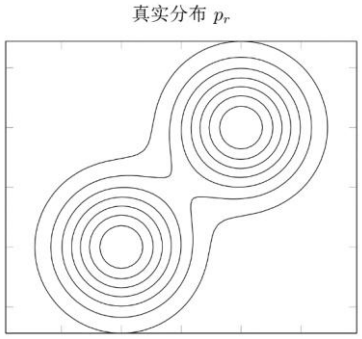
until 模型 $f(\mathbf{x}; \theta)$ 在验证集 $\mathcal{V}$ 上的错误率不再下降;

输出: θ

泛化错误

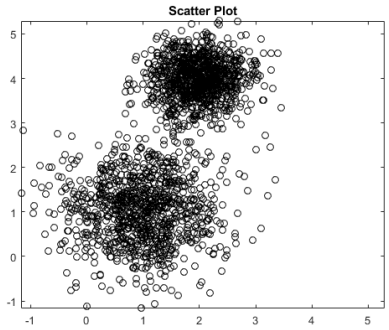
期望风险

$$\mathcal{R}(f) = \mathbb{E}_{(\mathbf{x},y)\sim p(\mathbf{x},y)}[\mathcal{L}(f(\mathbf{x}), y)],$$



经验风险

$$\mathcal{R}_{\mathcal{D}}^{emp}(\theta) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(y^{(n)}, f(x^{(n)}, \theta))$$



$\neq$

$$\mathcal{G}_{\mathcal{D}}(f) = \mathcal{R}(f) - \mathcal{R}_{\mathcal{D}}^{emp}(f)$$

泛化错误

如何减少泛化错误？

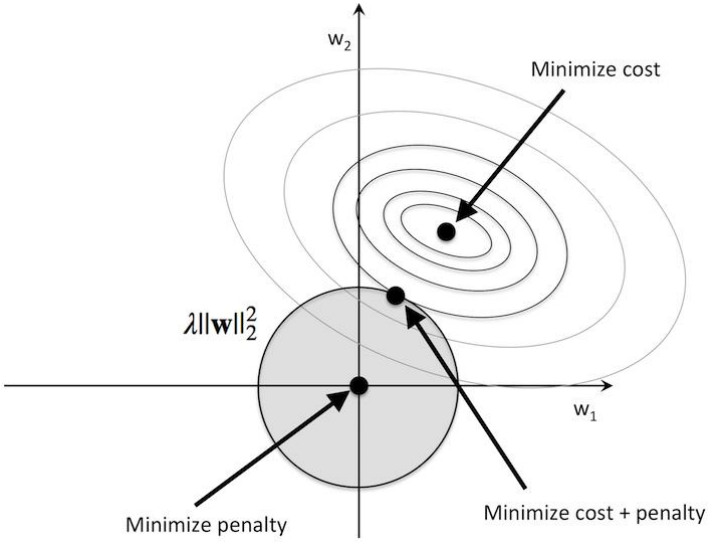
优化
经验风险最小

正则化
降低模型复杂度

正则化（regularization）

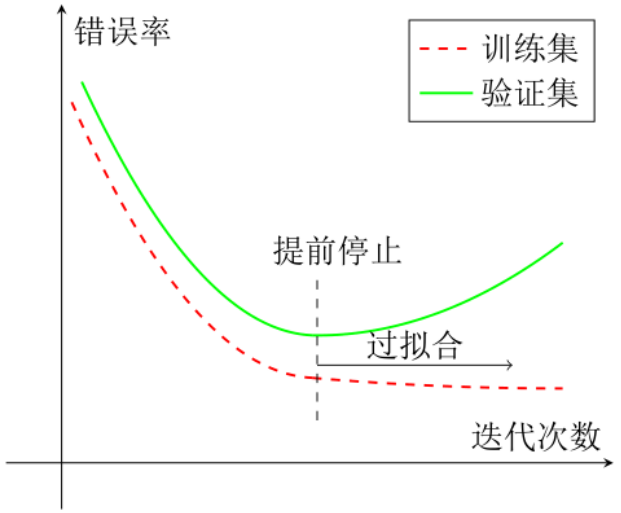
所有损害优化的方法都是正则化

增加优化约束



L1/L2约束、数据增强

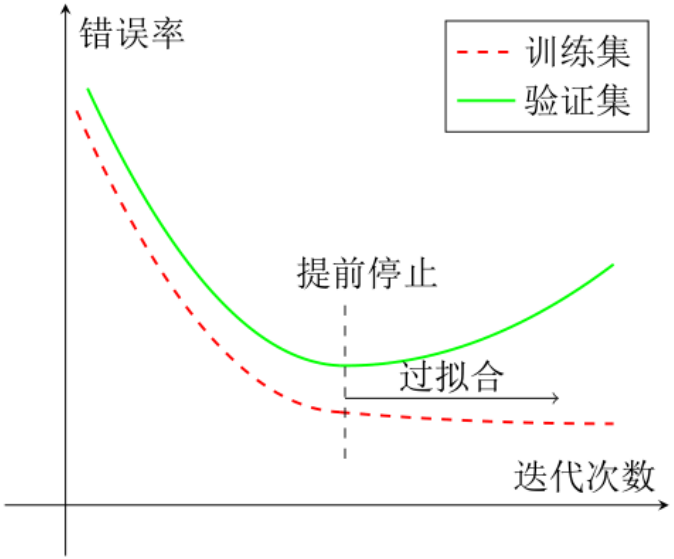
干扰优化过程



权重衰减、随机梯度下降、提前停止

提前停止

- 我们使用一个验证集（Validation Dataset）来测试每一次迭代的参数在验证集上是否最优。如果在验证集上的错误率不再下降，就停止迭代。



如何选择一个合适的模型？

- 模型选择
  - 拟合能力强的模型一般复杂度会比较高，容易过拟合。
  - 如果限制模型复杂度，降低拟合能力，可能会欠拟合。
- 偏差与方差分解
  - 期望误差可以分解为

$$\mathcal{R}(f) = (\text{bias})^2 + \text{variance} + \varepsilon.$$

$$\mathbb{E}_{\mathbf{x}}\left[\left(\mathbb{E}_{\mathcal{D}}\left[f_{\mathcal{D}}(\mathbf{x})\right] - f^*(\mathbf{x})\right)^2\right]$$

$$\mathbb{E}_{\mathbf{x}}\left[\mathbb{E}_{\mathcal{D}}\left[\left(f_{\mathcal{D}}(\mathbf{x}) - \mathbb{E}_{\mathcal{D}}\left[f_{\mathcal{D}}(\mathbf{x})\right]\right)^2\right]\right]$$

$$\mathbb{E}_{(\mathbf{x},y)\sim p_r(\mathbf{x},y)}\left[\left(y - f^*(\mathbf{x})\right)^2\right]$$

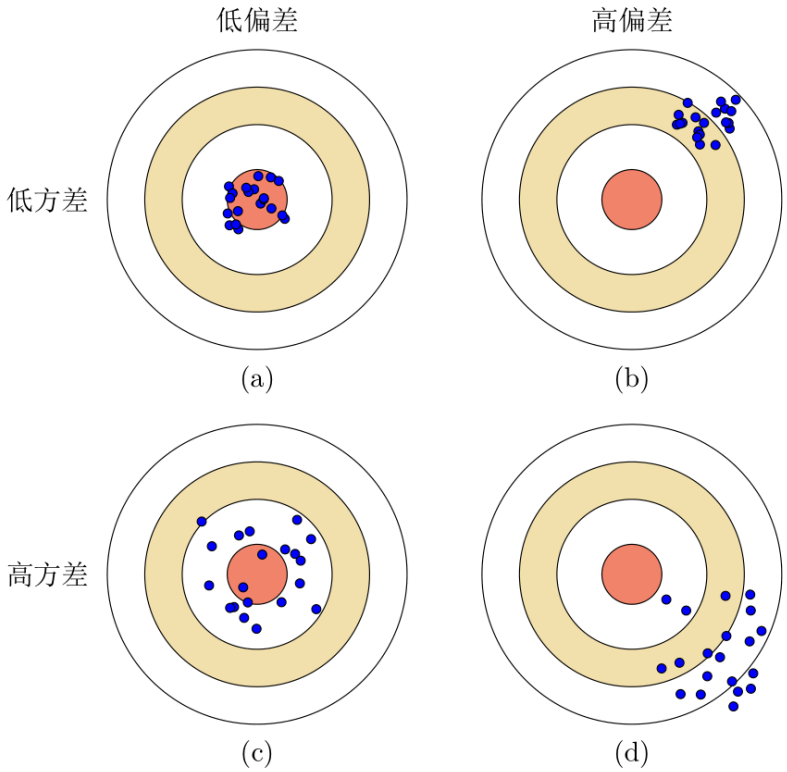
Mathematical Derivation for Total Error

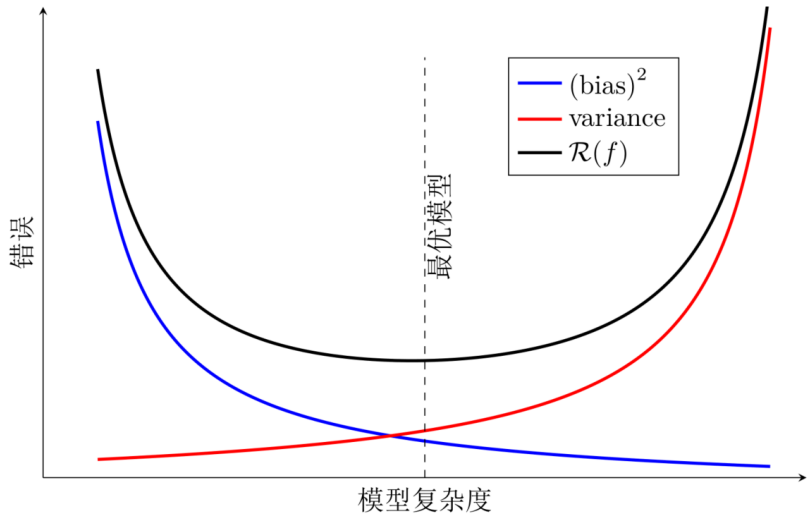
$$\begin{aligned}\text{MSE} &= (Y - \hat{Y})^2 \\ &= (Y - E(\hat{Y}) + E(\hat{Y}) - \hat{Y})^2 \\ &= (Y - E(\hat{Y}))^2 + (E(\hat{Y}) - \hat{Y})^2 + 2(Y - E(\hat{Y}))(E(\hat{Y}) - \hat{Y})\end{aligned}$$

Applying the Expectations on both sides.

$$\begin{aligned}E[(Y - \hat{Y})^2] &= E[(Y - E(\hat{Y}))^2 + (E(\hat{Y}) - \hat{Y})^2 + 2(Y - E(\hat{Y}))(E(\hat{Y}) - \hat{Y})] \\ &= E[(Y - E(\hat{Y}))^2] + E[(E(\hat{Y}) - \hat{Y})^2] + 2E[(Y - E(\hat{Y}))(E(\hat{Y}) - \hat{Y})]] \\ &= [(Y - E(\hat{Y}))^2] + E[(E(\hat{Y}) - \hat{Y})^2] + 2(Y - E(\hat{Y}))E[(E(\hat{Y}) - \hat{Y})]] \\ &= [(Y - E(\hat{Y}))^2] + E[(E(\hat{Y}) - \hat{Y})^2] + 2(Y - E(\hat{Y}))[E[E(\hat{Y})] - E[\hat{Y}]] \\ &= [(Y - E(\hat{Y}))^2] + E[(E(\hat{Y}) - \hat{Y})^2] + 2(Y - E(\hat{Y}))[E(\hat{Y})] - E[\hat{Y}]] \\ &= [(Y - E(\hat{Y}))^2] + E[(E(\hat{Y}) - \hat{Y})^2] + 2(Y - E(\hat{Y}))[0] \\ &= [(Y - E(\hat{Y}))^2] + E[(E(\hat{Y}) - \hat{Y})^2] + 0 \\ &= [\text{Bias}^2] + \text{Variance}\end{aligned}$$

模型选择：偏差与方差


低偏差
高偏差
低方差
高方差
(a)
(b)
(c)
(d)


$(\text{bias})^2$
variance
$\mathcal{R}(f)$
错误
最优模型
模型复杂度

分类问题

- 将分类问题看作条件概率估计问题
  - 引入非线性函数g 来预测类别标签的条件概率 $p(y = c|x)$。
  - 以二分类为例，

$$p(y = 1|\mathbf{x}) = g(f(\mathbf{x}; \mathbf{w}))$$

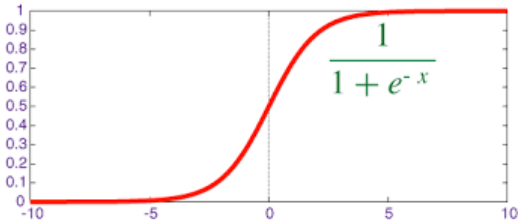
如何构建函数g？

  - 函数f：线性函数
  - 函数g：把线性函数的值域从实数区间“挤压”到了(0,1)之间，可以用来表示概率。

Logistic函数与回归

Logistic函数

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$



Logistic回归

$$p(y = 1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x})$$
$$\triangleq \frac{1}{1 + \exp(-\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x})}$$

学习准则

- **模型预测条件概率**$p_\theta(y|\boldsymbol{x})$

$$p_\theta(y=1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x})$$

- **真实条件概率**$p_r(y|\boldsymbol{x})$
  - **对于一个样本**(x, y^*)**,其真实条件概率为**

$$p_r(y=1|\boldsymbol{x}) = y^*$$
$$p_r(y=0|\boldsymbol{x}) = 1 - y^*$$

如何衡量两个条件分布的差异？

熵（Entropy）

- 在信息论中，熵用来衡量一个随机事件的不确定性。
  - 自信息（Self Information）
  - 熵

$$I(x) = -\log(p(x))$$

$$\begin{aligned} H(X) &= \mathbb{E}_X[I(x)] \\ &= \mathbb{E}_X[-\log p(x)] \\ &= -\sum_{x \in X} p(x) \log p(x) \end{aligned}$$

  - 熵越高，则随机变量的信息越多；
  - 熵越低，则随机变量的信息越少 .

  - 在对分布 $q(y)$ 的符号进行编码时，熵 $I(q)$ 也是理论上最优的平均编码长度，这种编码方式称为熵编码（ Entropy Encoding ）

交叉熵（Cross Entropy）

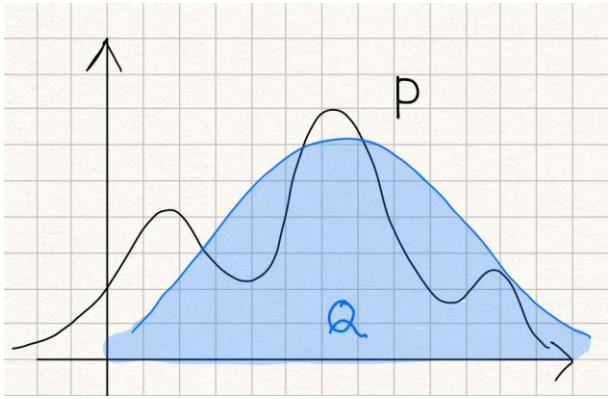
- 交叉熵是按照概率分布q的最优编码对真实分布为p的信息进行编码的长度。

$$
\begin{aligned}
H(p,q) &= \mathbb{E}_p[-\log q(x)] \\
&= -\sum_x p(x)\,\log q(x)
\end{aligned}
$$

  - 在给定 q 的情况下，如果 p和 q 越接近，交叉熵越小；
  - 如果 p 和 q 越远，交叉熵就越大 .

KL散度（Kullback-Leibler Divergence）

- KL散度是用概率分布q来近似p时所造成的信息损失量。
  - KL散度是按照概率分布q的最优编码对真实分布为p的信息进行编码，其平均编码长度（即交叉熵）H(p,q)和p的最优平均编码长度（即熵）H(p)之间的差异。



$$\mathrm{KL}(p,q) = H(p,q) - H(p)$$

$$= \sum_x p(x) \log \frac{p(x)}{q(x)}$$

交叉熵损失

$$\mathrm{D}_{KL}(p_r(y|x)||p_\theta(y|x)) = \sum_{y=0}^{1} p_r(y|x)\log\frac{p_r(y|x)}{p_\theta(y|x)}$$ KL散度

$$\propto -\sum_{y=0}^{1} p_r(y|x)\log p_\theta(y|x)$$ 交叉熵损失

$$= -\ I(y^*=1)\log p_\theta(\mathrm{y}=1|x) - I(y^*=0)\log p_\theta(\mathrm{y}=0|x)$$

y^*为x的真实标签

$$= -\ y^*\log p_\theta(\mathrm{y}=1|x) - (1-y^*)\log p_\theta(\mathrm{y}=0|x)$$

$$= -\ \log p_\theta(y^*|x)$$ 负对数似然

交叉熵损失函数

- 负对数似然损失函数

$$\mathcal{L}(\mathbf{y}, f(\mathbf{x}, \theta)) = -\sum_{c=1}^{C} y_c \log f_c(\mathbf{x}, \theta)$$

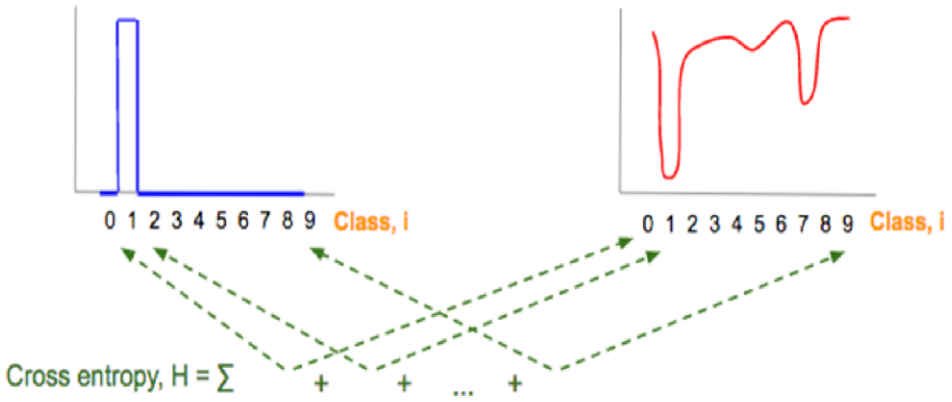
- 对于一个三类分类问题，类别为[0,0,1]，预测类别概率为[0.3,0.3,0.4]，则

$$\begin{aligned}\mathcal{L}(\theta) &= -\big(0 \times \log(0.3) + 0 \times \log(0.3) + 1 \times \log(0.4)\big) \\ &= -\log(0.4).\end{aligned}$$

交叉熵损失

$$-\sum_{y=1}^{C} p_r(y|x) \log p_\theta(y|x)$$

真实概率$p_r(y|x)$　　预测概率的负对数$-\log p_\theta(y|x)$



梯度下降

- 交叉熵损失函数，模型在训练集的风险函数为

$$\mathcal{R}(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}\log\left(\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})\right) + (1-y^{(i)})\log\left(1-\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})\right)\right).$$

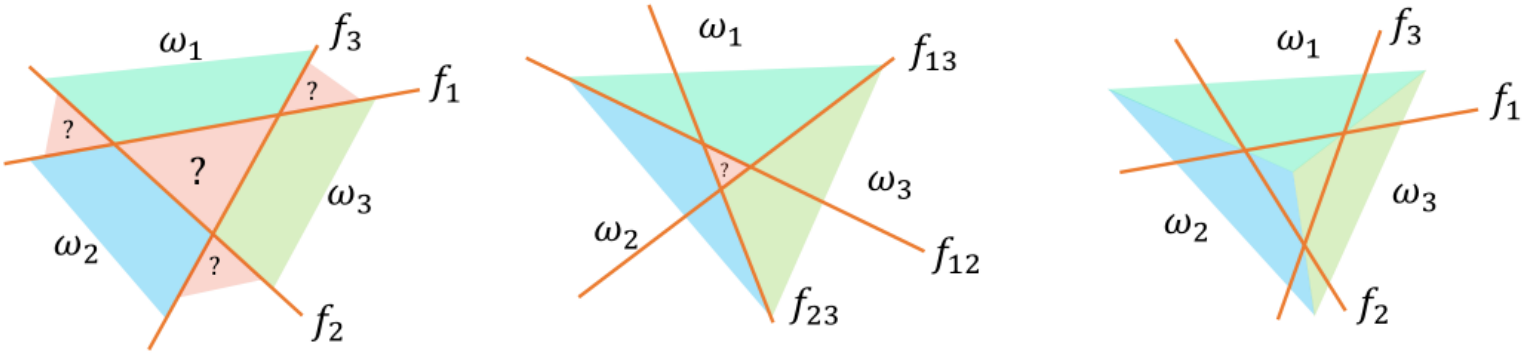
- 梯度为

$$\frac{\partial \mathcal{R}(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{N}\sum_{i=1}^{N}\left(\mathbf{x}^{(i)}\cdot\left(\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}) - y^{(i)}\right)\right)$$

推导过程

$$
\begin{aligned}
\frac{\partial \mathcal{R}(\mathbf{w})}{\partial \mathbf{w}} &= -\frac{1}{N}\sum_{n=1}^{N}\left(y^{(n)}\frac{\hat{y}^{(n)}(1-\hat{y}^{(n)})}{\hat{y}^{(n)}}\mathbf{x}^{(n)} - (1-y^{(n)})\frac{\hat{y}^{(n)}(1-\hat{y}^{(n)})}{1-\hat{y}^{(n)}}\mathbf{x}^{(n)}\right) \\
&= -\frac{1}{N}\sum_{n=1}^{N}\left(y^{(n)}(1-\hat{y}^{(n)})\mathbf{x}^{(n)} - (1-y^{(n)})\hat{y}^{(n)}\mathbf{x}^{(n)}\right) \\
&= -\frac{1}{N}\sum_{n=1}^{N}\mathbf{x}^{(n)}\left(y^{(n)} - \hat{y}^{(n)}\right).
\end{aligned}
$$

多分类（Multi-class Classification）



(a) “一对其余”方式　(b) “一对一”方式　(c) “argmax”方式

“*argmax*”方式：这是一种改进的“一对其余”方式，共需要C个判别函数

$$f_c(\mathbf{x};\mathbf{w}_c)=\mathbf{w}_c^{\mathrm{T}}\mathbf{x}+b_c,\qquad c=[1,\cdots,C] \tag{3.10}$$

如果存在类别c,对于所有的其他类别$\tilde{c}(\tilde{c}\neq c)$都满足$f_c(\mathbf{x};\mathbf{w}_c)>f_{\tilde{c}}(\mathbf{x},\mathbf{w}_{\tilde{c}})$，那么$\mathbf{x}$属于类别$c$。即

$$y=\arg\max_{c=1}^{C} f_c(\mathbf{x};\mathbf{w}_c). \tag{3.11}$$

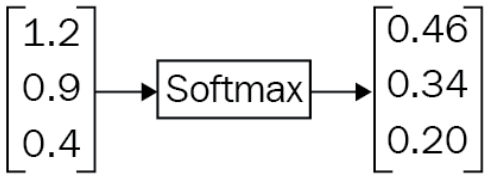
Softmax回归

- 多分类问题

$$y = \arg\max_{c=1}^{C} f_c(\boldsymbol{x}; \boldsymbol{w}_c)$$

- Softmax函数

$$\text{softmax}(x_k) = \frac{\exp(x_k)}{\sum_{i=1}^{K} \exp(x_i)}$$



Softmax回归

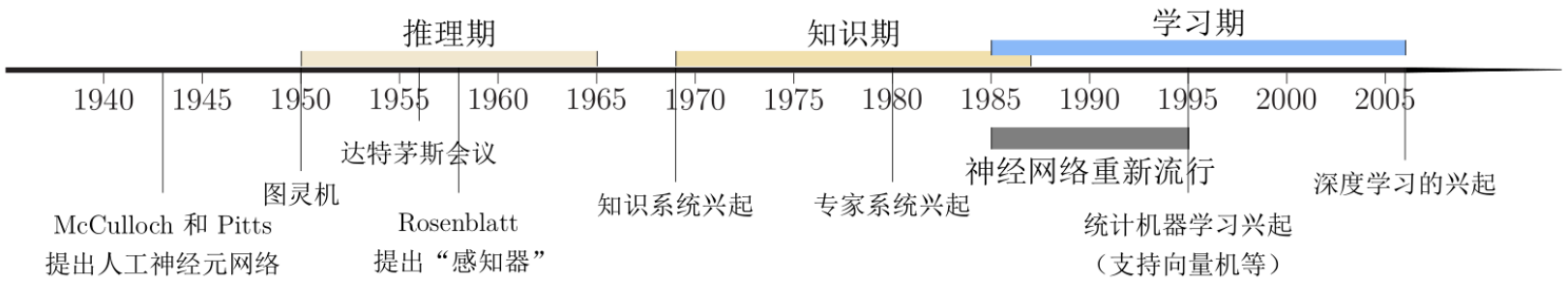
- 利用softmax函数，目标类别y = c的条件概率为：

$$P(y = c|\mathbf{x}) = \text{softmax}(\mathbf{w}_c^{\mathrm{T}}\mathbf{x})$$
$$= \frac{\exp(\mathbf{w}_c^{\top}\mathbf{x})}{\sum_{i=1}^{C}\exp(\mathbf{w}_i^{\top}\mathbf{x})}.$$

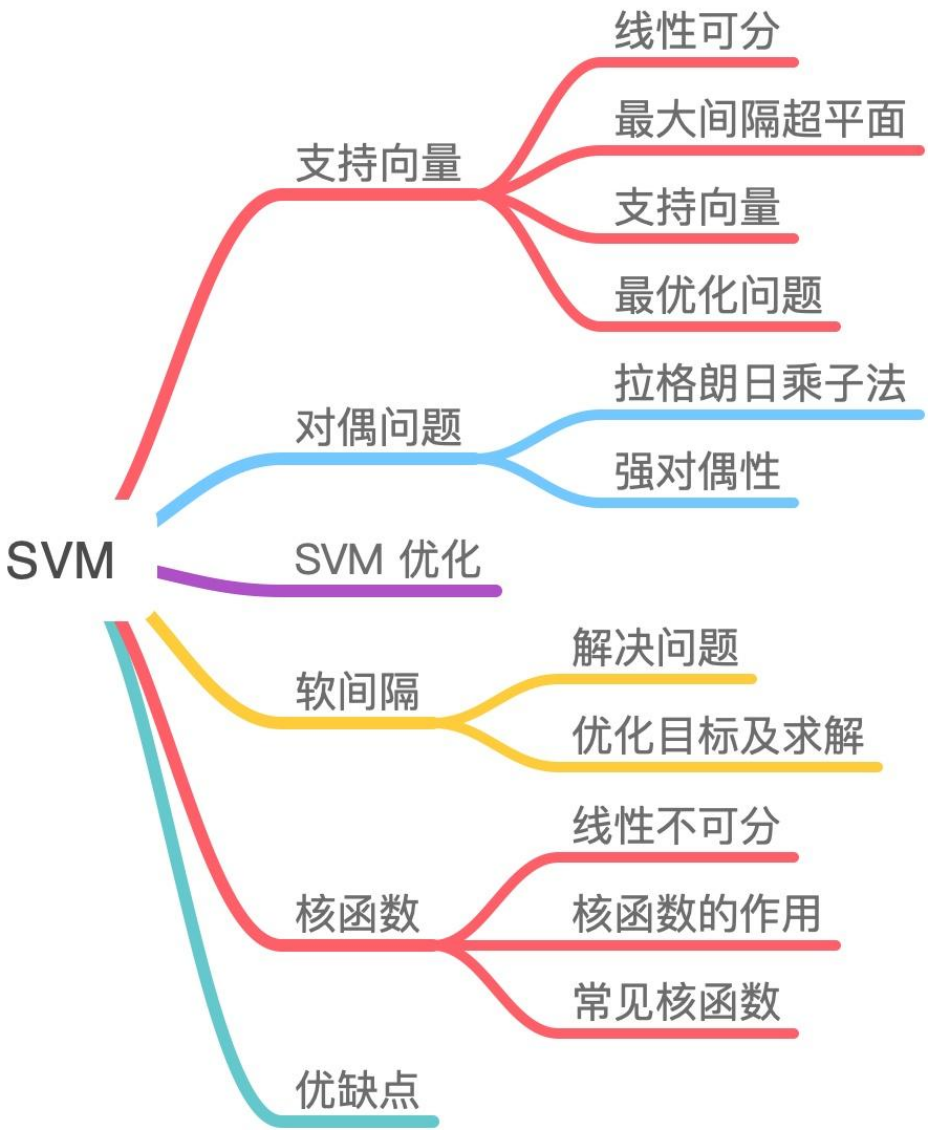
Outline of Lecture Three

- Deep Learning Framework
- Basic Concepts in Machine Learning
- **Support Vector Machine**
- Performance Evaluation Index

发展历史


推理期
知识期
学习期
1940
1945
1950
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
图灵机
达特茅斯会议
神经网络重新流行
深度学习的兴起
McCulloch 和 Pitts
提出人工神经元网络
Rosenblatt
提出“感知器”
知识系统兴起
专家系统兴起
统计机器学习兴起
（支持向量机等）

Support Vector Machine



Introduction

- What are benefits SV learning?
  - Based on simple idea
  - High performance in practical applications
- Characteristics of SV method
  - Can dealing with complex nonlinear problems
    (pattern recognition, regression, feature extraction)
  - But working with a simple linear algorithm
    (by the use of kernels)

Empirical Risk

- **We want to estimate a function using training data**

$$T = \{(X_i, d_i)\}_{i=1}^{N} \rightarrow F(X, W)$$

- **Loss between desired response and actual response**

$$L(d, F(X, W)) = (d - F(X, W))^2$$

- **Expected risk （风险泛函）**

$$R(W) = \frac{1}{2} \int L(d, F(X; W))\, dF_{X,D}(X, d)$$

- **Empirical risk （经验风险泛函）**

$$R_{emp}(W) = \frac{1}{N} \sum_{i=1}^{N} L(d_i, F(X_i, W)$$

Empirical risk minimization principle

- The true expected risk is approximated by empirical risk

$$R_{emp}(W) = \frac{1}{N}\sum_{i=1}^{N} L(d_i, F(X_i, W))$$

- The learning based on the empirical minimization principle is defined as

$$W^* = \arg\min_{W} \; R_{\text{emp}}(W)$$

Examples of algorithms: Perceptron, Back-propagation, etc.

PAC学习

- PAC： Probably Approximately Correct
- 概率近似正确
- 根据大数定律，当训练集大小|D|趋向无穷大时，泛化错误趋向于0，即经验风险趋近于期望风险。

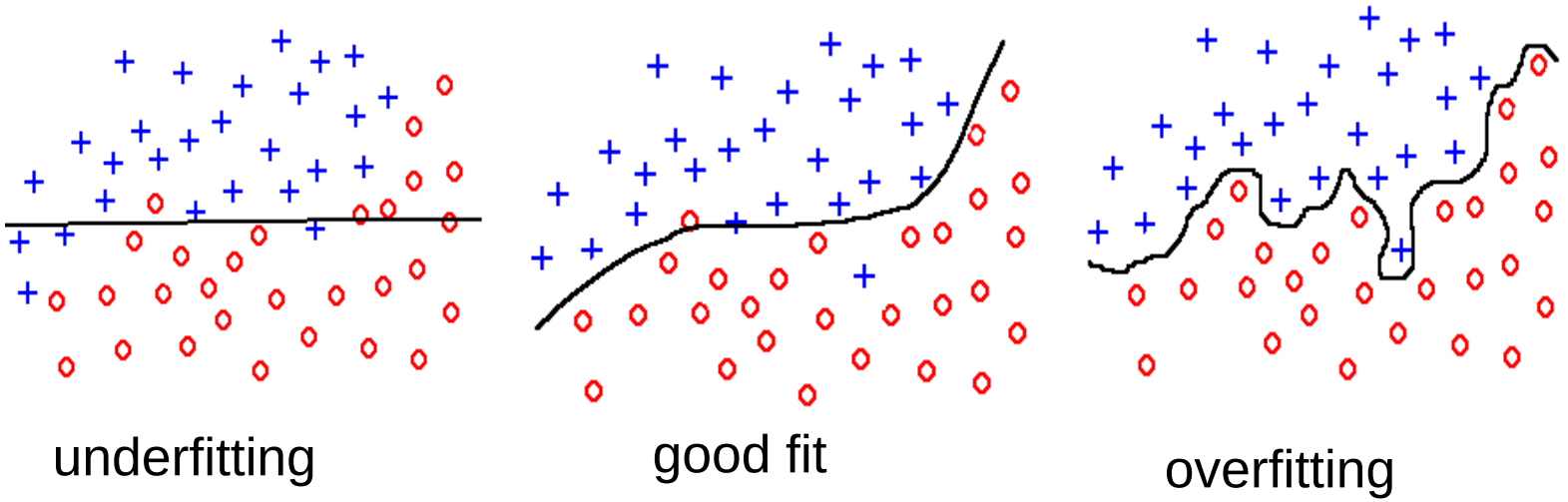
$$\lim_{|\mathcal{D}|\to\infty} \mathcal{R}(f) - \mathcal{R}_{\mathcal{D}}^{emp}(f) = 0$$

- PAC学习

$$P\Big(\underbrace{(\mathcal{R}(f) - \mathcal{R}_{\mathcal{D}}^{emp}(f)) \leq \epsilon}_{\text{近似正确}}\Big) \geq \underbrace{1-\delta}_{\text{可能}}$$

Overfitting and underfitting

- **Problem: How rich class of classifications *F(X,W)* *to use***



- **Problem of generalization: A small empirical risk $R_{emp}(W)$ does not imply small true expected risk $R(W)$**

Structural Risk Minimization

- Statistical learning theory : Vapnik & Chervonenkis
- An upper bound on the expected risk of a classification rule

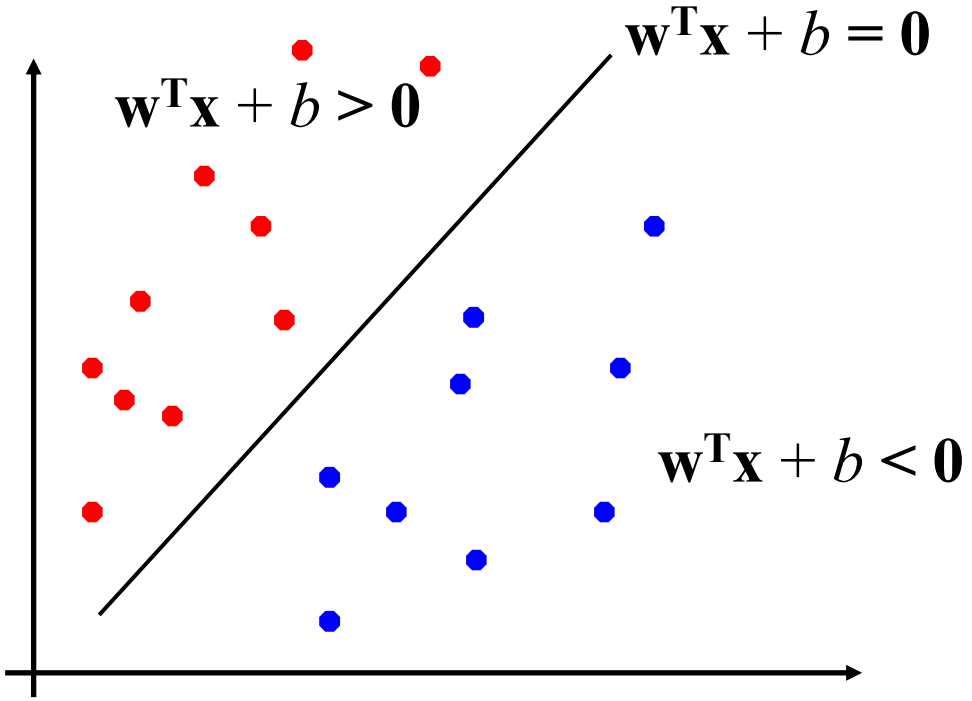
$$R(W) \le R_{emp}(W) + \sqrt{\frac{h[\log(2N/h)+1]-\log(\alpha)}{N}}$$

where N is the number of training data, h is VC-dimension of class of functions.

- SRM Principle: to find a network structure such that decreasing the VC dimension occurs at the expense of the smallest possible increase in training error

Perceptron Revisited: Linear Separators

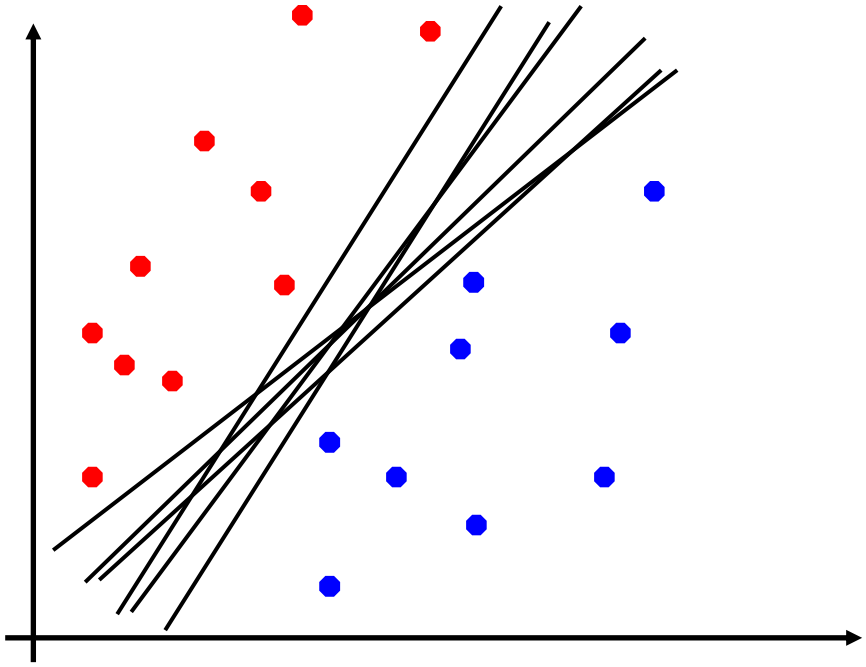
- Binary classification can be viewed as the task of separating classes in feature space:



$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$$

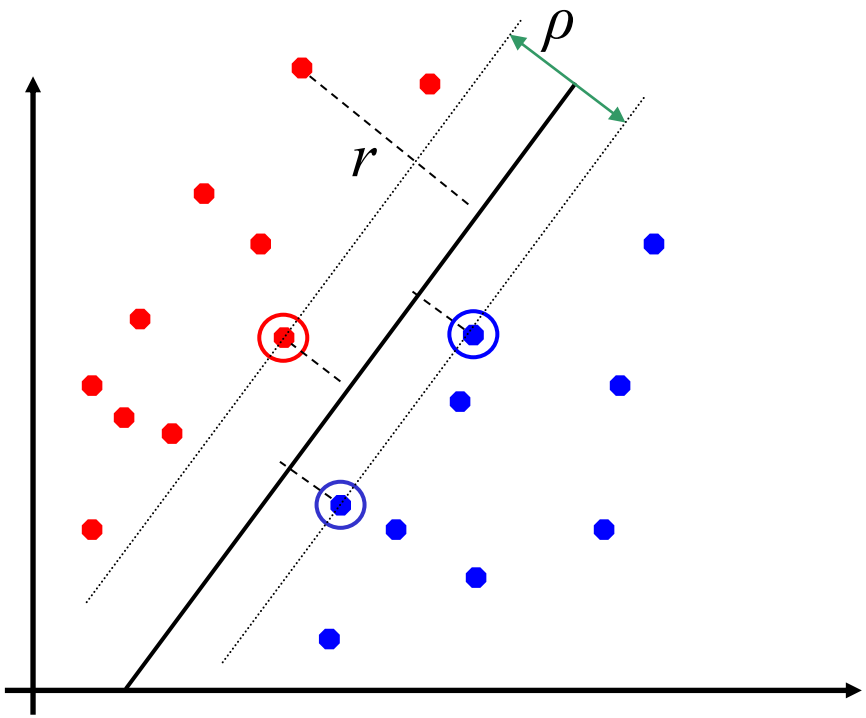
Linear Separators

- Which of the linear separators is optimal?



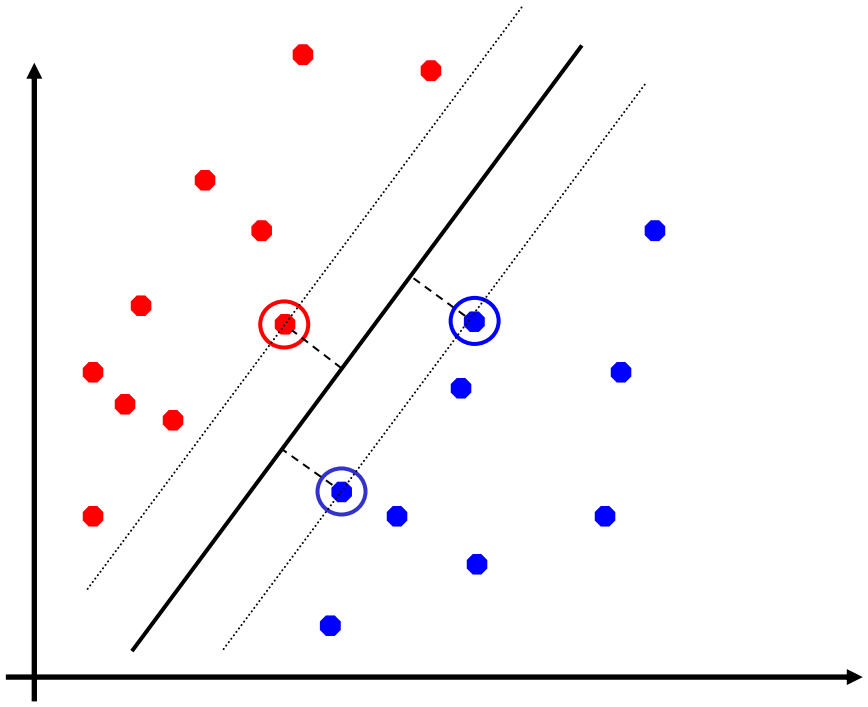
Classification Margin

- Distance from example $\mathbf{x}_i$ to the separator is $r = \dfrac{\mathbf{w}^T\mathbf{x}_i + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are *support vectors*.
- *Margin* ρ of the separator is the distance between support vectors.



Maximum Margin Classification

- Maximizing the margin is good according to intuition and PAC theory.
- Implies that only support vectors matter; other training examples are ignorable.



Linear SVM Mathematically

- Let training set $\{(x_i, y_i)\}_{i=1..n}$, $x_i \in R^d$, $y_i \in \{-1, 1\}$ be separated by a hyperplane with margin ρ. Then for each training example (x_i, y_i):

$$\begin{aligned} \mathbf{w}^T\mathbf{x}_i + b &\leq -\rho/2 \quad \text{if } y_i = -1 \\ \mathbf{w}^T\mathbf{x}_i + b &\geq \rho/2 \quad \text{if } y_i = 1 \end{aligned} \quad \Leftrightarrow \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq \rho/2$$

- For every support vector x_s the above inequality is an equality. After rescaling w and *b* by $\rho/2$ in the equality, we obtain that distance between each x_s and the hyperplane is

$$r = \frac{y_s(\mathbf{w}^T\mathbf{x}_s + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

- Then the margin can be expressed through (rescaled) w and b as:

$$\rho = 2r = \frac{2}{\|\mathbf{w}\|}$$

Linear SVMs Mathematically (cont.)

- **Then we can formulate the *quadratic optimization problem:***

Find $\mathbf{w}$ and b such that

$$\rho = \frac{2}{\|\mathbf{w}\|} \text{ is maximized}$$

and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Which can be reformulated as:

Find $\mathbf{w}$ and b such that

$\mathbf{\Phi}(\mathbf{w}) = \|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{w}$ is minimized

and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Solving the Optimization Problem

Find $\mathbf{w}$ and b such that
$\mathbf{\Phi(w)} = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i\,(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Need to optimize a quadratic function subject to linear constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist.
- The solution involves constructing a dual problem where a Lagrange multiplier α_i is associated with every inequality constraint in the primal (original) problem:

Find $\alpha_1 \ldots \alpha_n$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

The Optimization Problem Solution

- Given a solution $\alpha_1 \ldots \alpha_n$ to the dual problem, solution to the primal is:

$$\mathbf{w} = \Sigma \alpha_i y_i \mathbf{x}_i \qquad b = y_k - \Sigma \alpha_i y_i \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_k \quad \text{for any } \alpha_k > 0$$

- Each non-zero α_i indicates that corresponding $\mathbf{x}_i$ is a **support vector**.

Decision function:

$$f(x) = \omega^T x + b = \sum_{i=1}^{m} \alpha_i y_i x_i^T x + b$$

KKT condition:

$$\begin{cases} \alpha_i \geq 0 \\ y_i f(x_i) - 1 \geq 0 \\ \alpha_i (y_i f(x_i) - 1) = 0 \end{cases}$$

$$\alpha_i = 0 \quad \text{or} \quad y_i f(x_i) = 1$$

Support Vectors

The Optimization Problem Solution

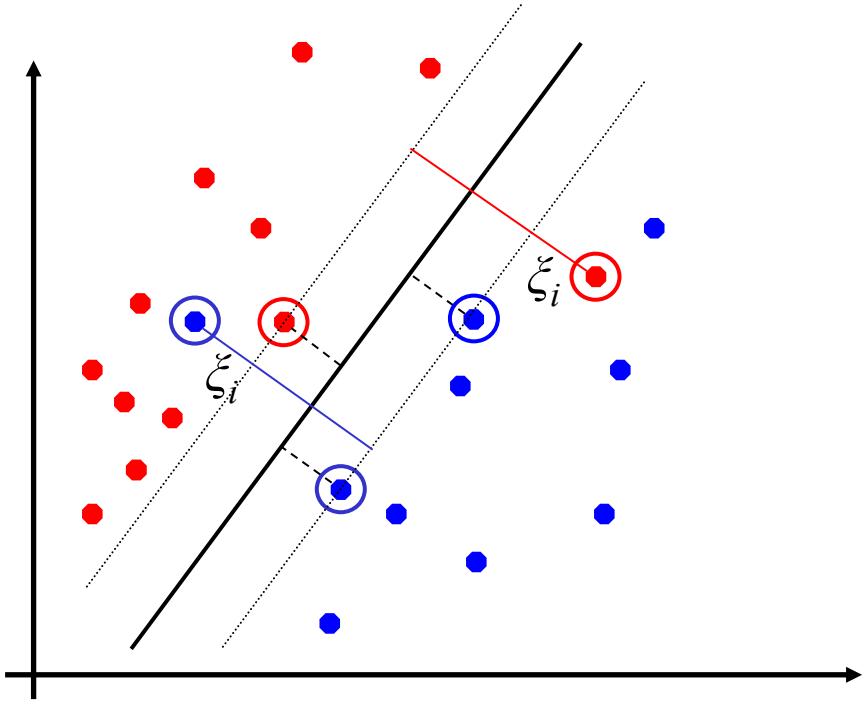
- Then the classifying function is (note that we don't need w explicitly):

$$f(\mathbf{x}) = \Sigma \alpha_i y_i \mathbf{x}_i^{\mathrm{T}} \mathbf{x} + b$$

- Notice that it relies on an inner product between the test point x and the support vectors x_i
- Also keep in mind that solving the optimization problem involved computing the inner products $x_i^T x_j$ between all training points.

Soft Margin Classification

- What if the training set is not linearly separable?
- Slack variables ξ_i can be added to allow misclassification of difficult or noisy examples, resulting margin called soft.



Soft Margin Classification Mathematically

- **The old formulation:**

Find $\mathbf{w}$ and b such that
$\Phi(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- **Modified formulation incorporates slack variables:**

Find $\mathbf{w}$ and b such that
$\Phi(\mathbf{w}) = \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$, $\xi_i \geq 0$

- **Parameter C can be viewed as a way to control overfitting: it "trades off" the relative importance of maximizing the margin and fitting the training data.**

Soft Margin Classification – Solution

- Dual problem is identical to separable case (would not be identical if the 2-norm penalty for slack variables $C\Sigma\xi_i^2$ was used in primal objective, we would need additional Lagrange multipliers for slack variables):

Find $\alpha_1 \ldots \alpha_N$ such that

$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and

(1) $\Sigma\alpha_i y_i = 0$

(2) $0 \leq \alpha_i \leq C$ for all α_i

- Again, xi with non-zero αi will be support vectors.
- Solution to the dual problem is:

$\mathbf{w} = \Sigma\alpha_i y_i \mathbf{x}_i$

$b = y_k(1 - \xi_k) - \Sigma\alpha_i y_i \mathbf{x}_i^T \mathbf{x}_k$ for any k s.t. $\alpha_k > 0$

Again, we don't need to compute **w** explicitly for classification:

$f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$

SVM Boundaries with different *C*

Find **w** and b such that

$\mathbf{\Phi(w)} = \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized

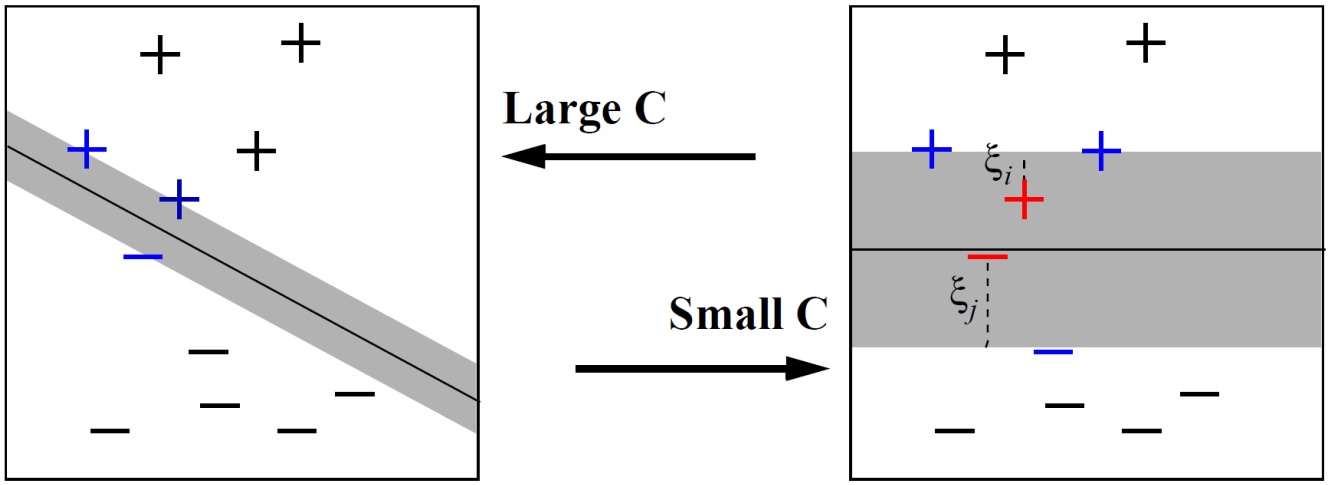
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$, , $\xi_i \geq 0$

Find $\alpha_1 \ldots \alpha_N$ such that

$\mathbf{Q(\alpha)} = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and

(1) $\Sigma\alpha_i y_i = 0$

(2) $0 \leq \alpha_i \leq C$ for all α_i



Theoretical Justification for Maximum Margins

- **Vapnik has proved the following:**

  The class of optimal linear separators has VC dimension h bounded from above as

$$h \leq \min\left\{\left\lceil \frac{D^2}{\rho^2} \right\rceil, m_0\right\} + 1$$

  where ρ is the margin, D is the diameter of the smallest sphere that can enclose all of the training examples, and m_0 is the dimensionality.

- **Intuitively, this implies that regardless of dimensionality m_0 we can minimize the VC dimension by maximizing the margin ρ.**
- **Thus, complexity of the classifier is kept small regardless of dimensionality.**

Linear SVMs: Overview

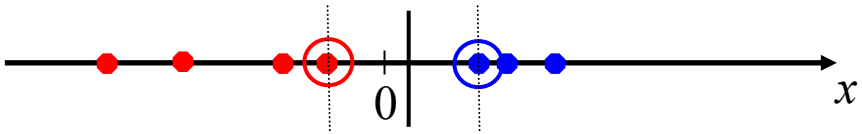
- The classifier is a separating hyperplane.
- Most "important" training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points xi are support vectors with non-zero Lagrangian multipliers αi.
- Both in the dual formulation of the problem and in the solution training points appear only inside inner products:

Find $\alpha_1 \ldots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \le \alpha_i \le C$ for all α_i

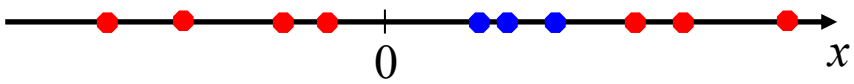
$$f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x}_i^T\mathbf{x} + b$$

Non-linear SVMs

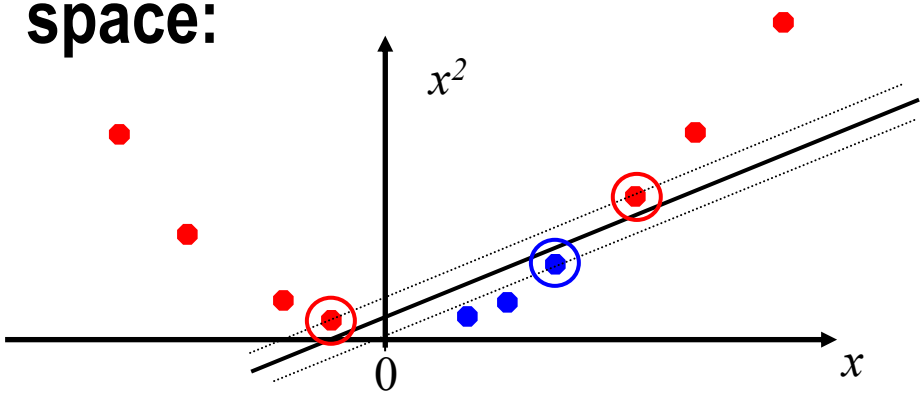
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?

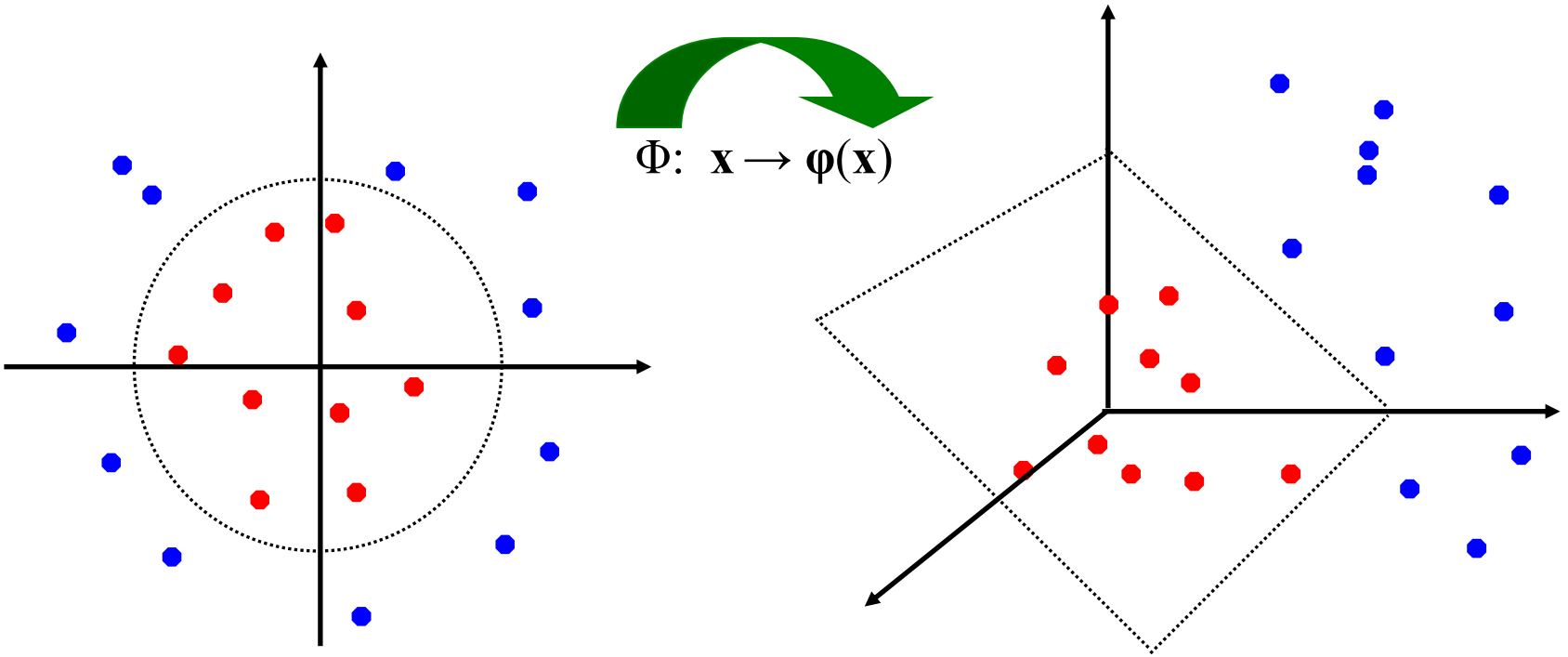


- How about… mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



The "Kernel Trick"

- The linear classifier relies on inner product between vectors $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j$
- If every datapoint is mapped into high-dimensional space via some transformation $\Phi: \boldsymbol{x} \rightarrow \varphi(\boldsymbol{x})$, the inner product becomes:
  $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \varphi(\boldsymbol{x}_i)^T \varphi(\boldsymbol{x}_j)$
- A kernel function is a function that is eqiuvalent to an inner product in some feature space.

Example:

2-dimensional vectors $\boldsymbol{x} = [x_1 \;\; x_2]$; let $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (1 + \boldsymbol{x}_i^T \boldsymbol{x}_j)^2$,

Need to show that $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \varphi(\boldsymbol{x}_i)^T \varphi(\boldsymbol{x}_j)$:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (1 + \boldsymbol{x}_i^T \boldsymbol{x}_j)^2 = 1 + x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{j1}x_{i2}x_{j2} + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$$

$$= [1 \;\; x_{i1}^2 \;\; \sqrt{2x_{i1}x_{i2}} \;\; x_{i2}^2 \;\; \sqrt{2x_{i1}} \;\; \sqrt{2x_{i2}}]^{\mathrm{T}} \; [1 \;\; x_{j1}^2 \;\; \sqrt{2x_{j1}x_{j2}} \;\; x_{j2}^2 \;\; \sqrt{2x_{j1}} \;\; \sqrt{2x_{j2}}]$$

$$= \varphi(\boldsymbol{x}_i)^T \varphi(\boldsymbol{x}_j),$$

where $\varphi(\boldsymbol{x}) = [1 \;\; x_1^2 \;\; \sqrt{2x_1x_2} \;\; x_2^2 \;\; \sqrt{2x_1} \;\; \sqrt{2x_2}]$

Thus, a kernel function implicitly maps data to a high-dimensional space (without the need to compute each $\varphi(\boldsymbol{x})$ explicitly).

What Functions are Kernels?

- For some functions $K(x_i, x_j)$ checking that $K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$ can be cumbersome.
- Mercer's theorem:

  Every semi-positive definite symmetric function is a kernel
- Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

K=

$K(\mathbf{x}_1,\mathbf{x}_1)$	$K(\mathbf{x}_1,\mathbf{x}_2)$	$K(\mathbf{x}_1,\mathbf{x}_3)$	…	$K(\mathbf{x}_1,\mathbf{x}_n)$
$K(\mathbf{x}_2,\mathbf{x}_1)$	$K(\mathbf{x}_2,\mathbf{x}_2)$	$K(\mathbf{x}_2,\mathbf{x}_3)$		$K(\mathbf{x}_2,\mathbf{x}_n)$
…	…	…	…	…
$K(\mathbf{x}_n,\mathbf{x}_1)$	$K(\mathbf{x}_n,\mathbf{x}_2)$	$K(\mathbf{x}_n,\mathbf{x}_3)$	…	$K(\mathbf{x}_n,\mathbf{x}_n)$

Examples of Kernel Functions

- **Linear kernel**

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')$$

- **Polynomial kernel**

$$K(\mathbf{x}, \mathbf{x}') = \left(1 + (\mathbf{x}^T \mathbf{x}')\right)^p$$

where $p = 2, 3, \ldots$. To get the feature vectors we concatenate all p^{th} order polynomial terms of the components of $\mathbf{x}$ (weighted appropriately)

- **Radial basis kernel**

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\|\mathbf{x} - \mathbf{x}'\|^2\right)$$

In this case the feature space consists of functions and results in a *non-parametric* classifier.

Non-linear SVMs Mathematically

- **Dual problem formulation:**

Find $\alpha_1 \ldots \alpha_n$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

- **The solution is:**

$$f(\mathbf{x}) = \Sigma\alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_j) + b$$

- **Optimization techniques for finding α_i's remain the same!**

Examples of Kernel Functions

- **Linear kernel**

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')$$

- **Polynomial kernel**

$$K(\mathbf{x}, \mathbf{x}') = \left(1 + (\mathbf{x}^T \mathbf{x}')\right)^p$$

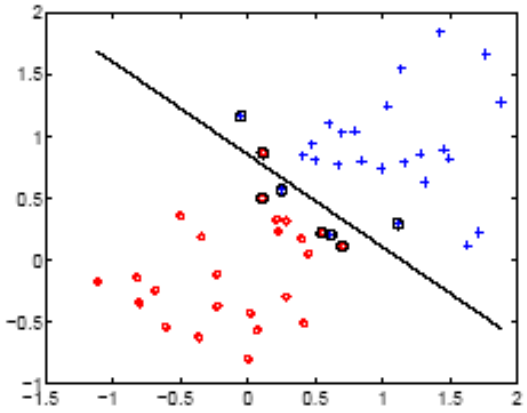
where $p = 2, 3, \ldots$. To get the feature vectors we concatenate all p^{th} order polynomial terms of the components of $\mathbf{x}$ (weighted appropriately)

- **Radial basis kernel**

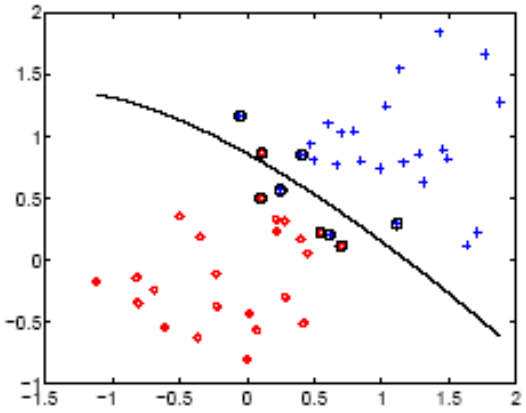
$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\|\mathbf{x} - \mathbf{x}'\|^2\right)$$

In this case the feature space consists of functions and results in a *non-parametric* classifier.

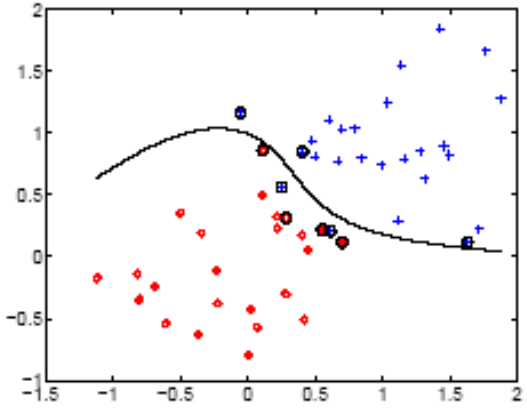
SVM Examples



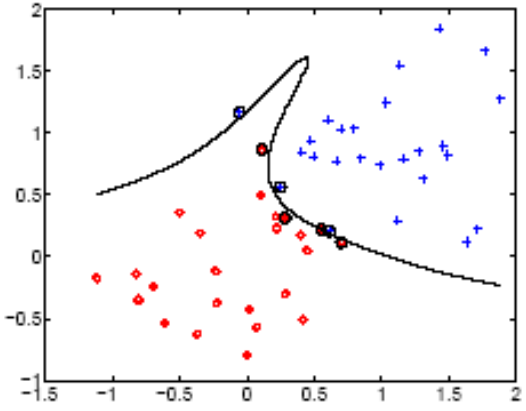
linear



2^{nd} order polynomial



4^{th} order polynomial



8^{th} order polynomial

Key Points

- Learning depends only on dot products of sample pairs.
- Exclusive reliance on dot products enables approach to non-linearly separable problems.
- The classifier depends only on the support vectors, not on all the training points.
- Max margin lowers hypothesis variance.
- The optimal classifier is defined uniquely-there are no "local maxima" in the search space
- Polynomial in number of data points and dimensionality

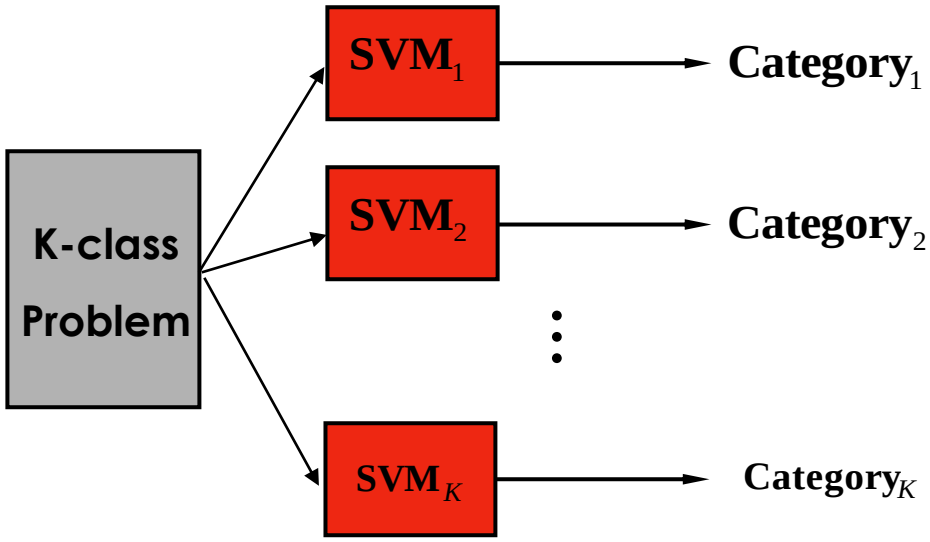
SVMs for Multi-class Classification Problems

Two task decomposition methods:

- One-versus-rest
- One-versus-one

One-Versus-Rest method

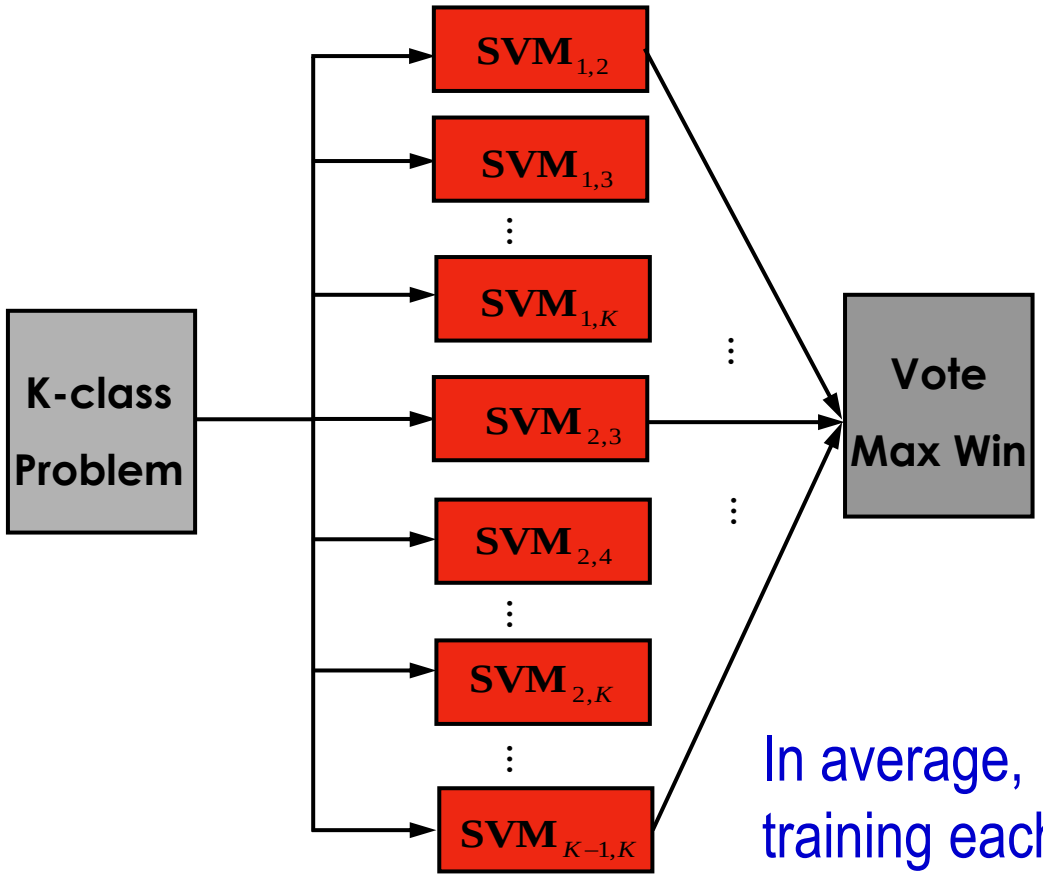
- This method requires one classifier per category. The i th SVM will be trained with all of the examples in the ith class with positive labels, and all other examples with negative labels.



The Number of training data for each classifier is ***N***

One-Versus-One Method

- This method constructs K(K-1)/2 classifiers where each one is trained on data from two out of K classes.



In average, number of data for training each classifier is *2N/K*

Limitations of traditional Methods

- Some of the two-class problems may fall into a load imbalance situation for the size of each class may be very imbalance in some problems.
- Using the one-versus-one, some of the two-class problems may still be too large to learn.

SVM software packages

- LibSVM
  - Http://www.csie.ntu.edu.tw/~cjlin/libsvm/
  - Chih-Chung Chang and Chin-Jen Lin
- SVM^{light}
  - http://svmlight.joachims.org/
  - Thorsten Joachims

LibSVM

- Various language versions
  - C++, C#, java, MatLab, etc.
  - Recommend C++ version
- The source code is readable
- The interface is clear

LibSVM

- Two executable files
  - Train.exe
    - Compiled by svm.cpp, svm.h and svm-train.c
  - Test.exe
    - Complied by svm.cpp, svm.h and svm-predict.c

LibSVM

- Description of svmtrain.exe
  - "one versus one" is implemented a solution to multi-class problem
  - Several frequently used parameters
    - -s : svm type (0 for classification)
    - -t : kernel type (2 for RBF kernel)
    - -g : gamma value
    - -c : panelized cost
    - e. g.,

    svmtrain -s 0 -t 2 -g 0.5 –c 2 train_file model_file

LibSVM

- **Description of svmpredict.exe**
  - **e. g.,**

    svmpredict test_file model_file result_file

LibSVM

- If you want to directly modify the source code and do your homework…
  - The source code has several interface functions. You can write codes to call these functions.
    - svm_train(), svm_predict_values(), svm_save_model(),…
  - Not recommended unless you have strong understanding to SVMs

Performance Evaluation Index

混淆矩阵(Confusion Matrix)

	预测正类	预测负类
正类	TP	FN
负类	FP	TN

TP (True Positive)：样本属于正类，预测结果为正类

FP (False Positive)：样本属于负类，预测结果为正类

FN (False Negative)：样本属于正类，预测结果为负类

TN (True Negative)：样本属于负类，预测结果为负类

常用的评价指标：精度

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

使用精度度量分类器的局限性

- 考虑一个2类问题：
  - 第一类有9990个测试样本
  - 第二类只有10个测试样本
- 如果分类器把全部测试样本都分为第一类，其精度为9990/10000=99%！
- 显然，这里凸显出精度的局限性。因为，它未能全面地衡量分类器对第二类的分类性能。

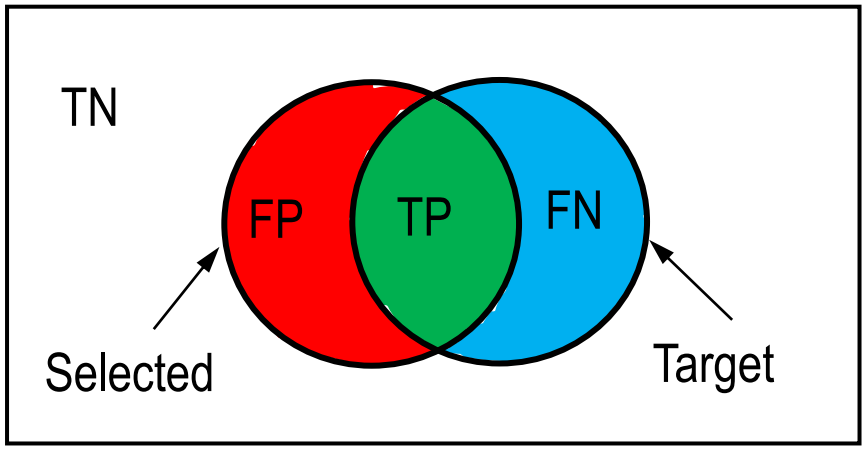
$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

An example: Mammography Data Set

- A collection of images acquired from mammography exams
- Positive and negative samples represent cancerous patient and healthy patient, respectively
- The data set contains 10923 negative samples and 260 positive samples
- Classifiers have accuracy near to 100 percent on the majority class and accuracies of 0-10 percent on minority class
- Suppose a classifier achieves 10 percent accuracy on minority class
- As a result, 234 cancerous patients are classified into noncancerous patients

精确率、召回率、F1度量

- 真正（True positive, TP); 假负（False negative, FN）
- 假正（False positive, FP）; 真负（True negative, TN）
- 真正率（True positive rate, TPR):TPR=TP/(TP+FN)
- 假正率（False positive rate, FPR)：FPR=FP/(FP+TN)
- 精确率（Precision)：p=TP/(TP+FP)
- 召回率（Recall)：r=TP/(TP+FN)
- F1度量：F1=2r*p/(r+p)



p: 精确率率
r: 召回率

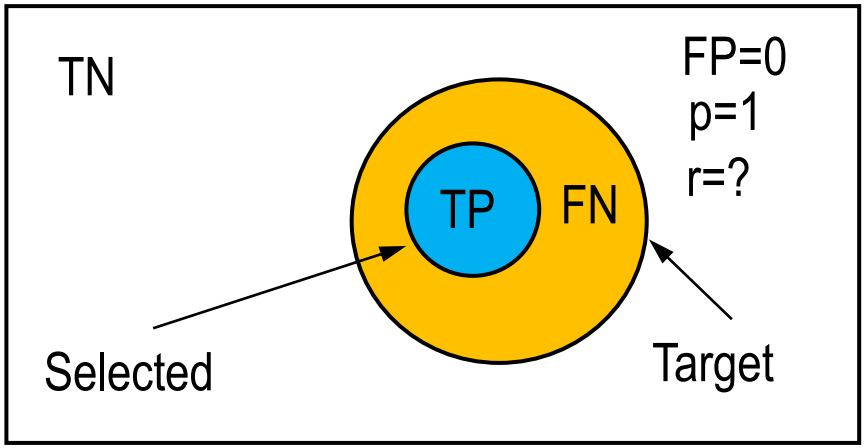
Macro-p
Macro-r
Macro-F1
先计算p、r再求平均

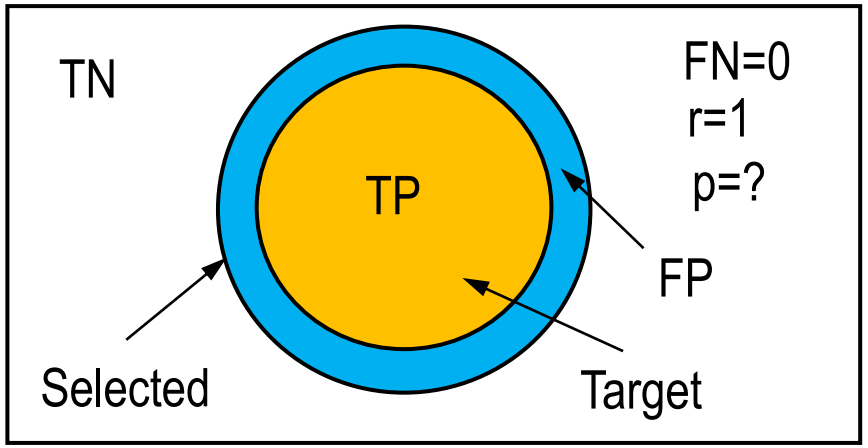
Micro-p
Micro-r
Micro-F1
先求TP、FP、TN、FN平均，再求p、r、F1

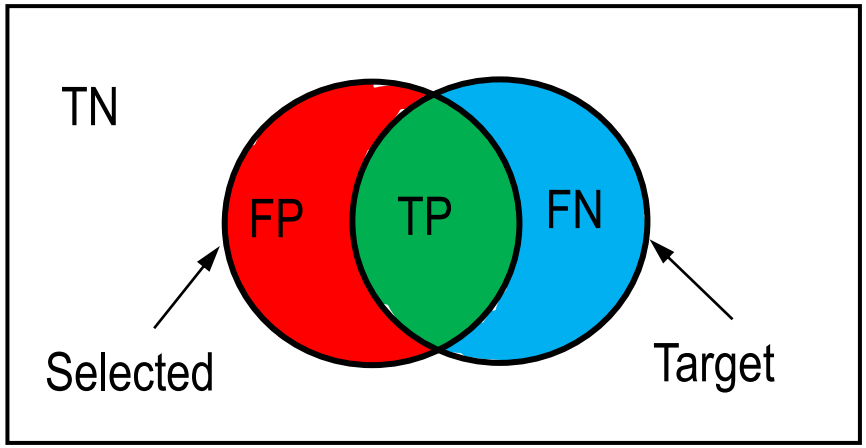
精确率、召回率、F1度量

- 精确率（Precision）：p=TP/(TP+FP)
- 召回率（Recall）：r=TP/(TP+FN)
- F1度量：F1=2*r*p/(r+p)

	预测正类	预测负类
正类	TP	FN
负类	FP	TN







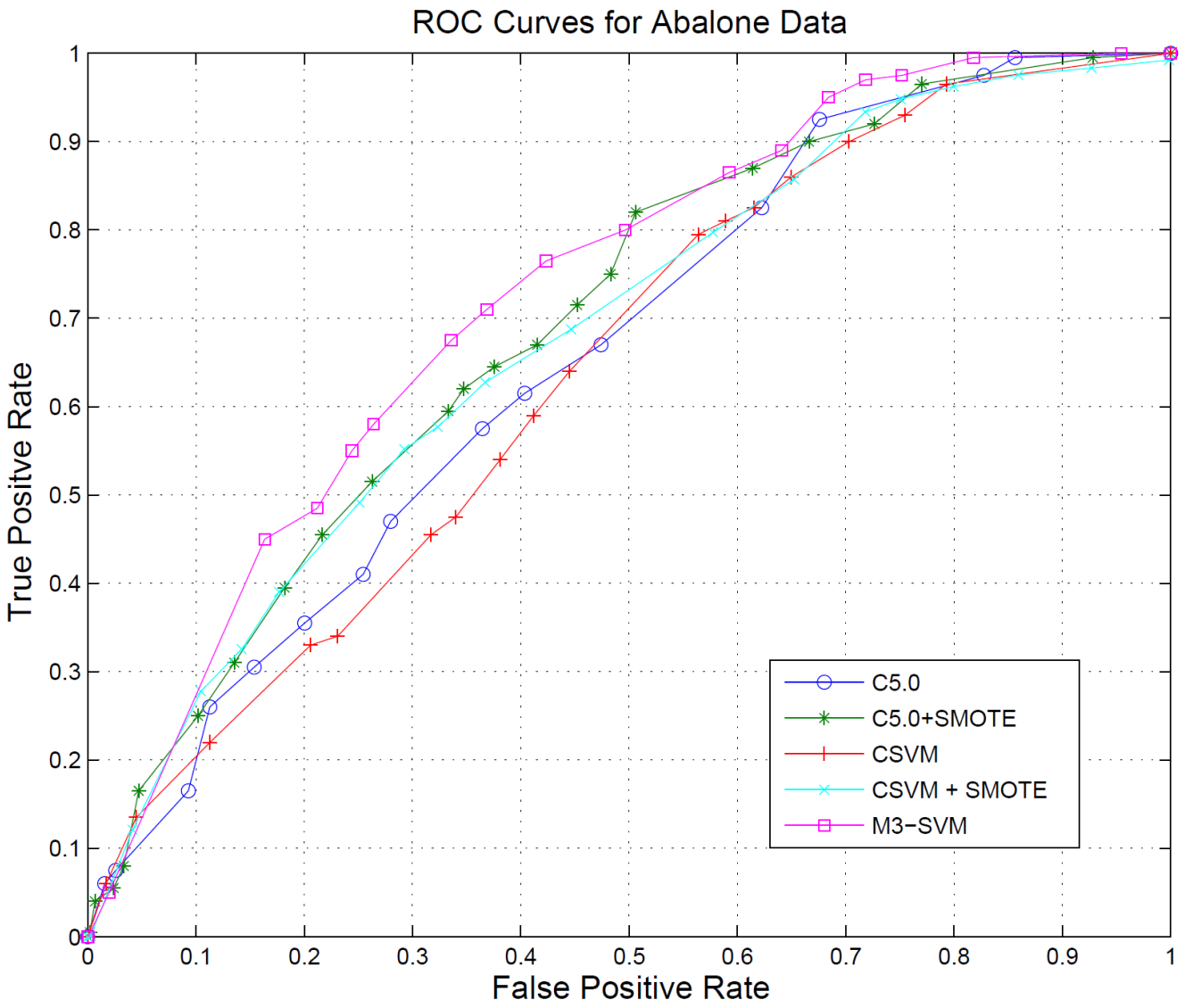
ROC (Receiver Operating Characteristic)曲线

ROC (Receiver Operating Characteristic)曲线

- Developed in 1950s for signal detection theory to analyze noisy signals
  - Characterize the trade-off between positive hits and false alarms
- ROC curve plots TPR (on the y-axis) against FPR (on the x-axis)
- Performance of each classifier represented as a point on the ROC curve
  - Changing the threshold of algorithm, sample distribution or cost matrix changes the location of the point
- AUC： Area under ROC Curve

- 真正率（True positive rate, TPR)： TPR=TP/(TP+FN)
- 假正率（False positive rate, FPR)： FPR=FP/(FP+TN)

使用ROC曲线比较 分类算法

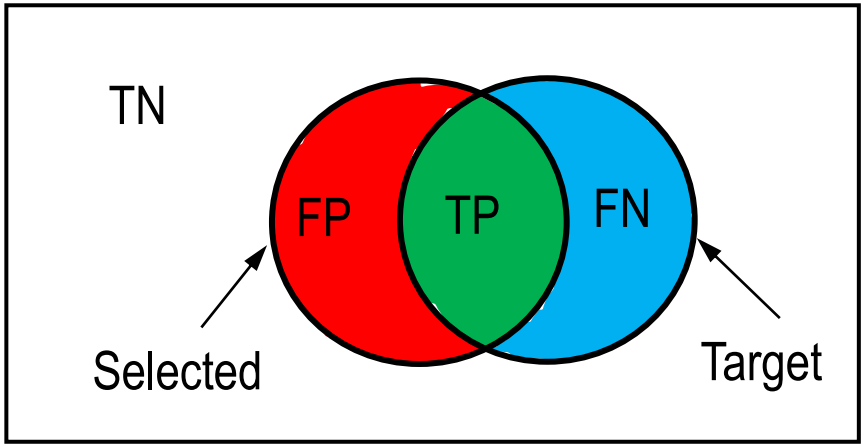


评估指标

- 真正类率（召回率）

TPR (True Positive Rate)：

$$TPR = \frac{TP}{TP + FN}$$



- 假正类率

FPR (False Positive Rate)：

$$FPR = \frac{FP}{FP + TN}$$

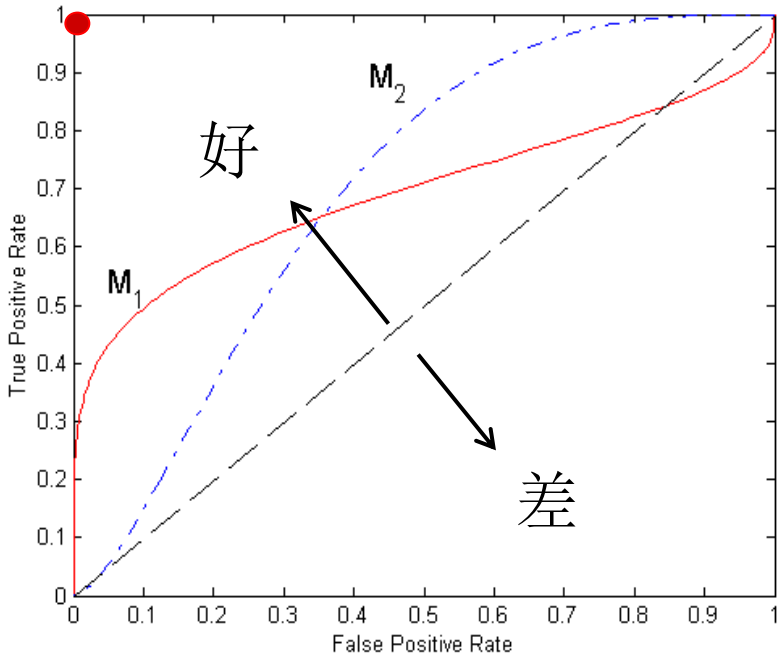
- **TPR越大，分类效果越好。而FPR越大，分类效果越差。**

ROC 曲线的几个关键点

- (TPR=0, FPR=0): 把每个输入都预测为负类；
- (TPR=1, FPR=1): 把每个输入都预测为正类
- (TPR=1, FPR=0): 理想模型
- 主对角线: 随机猜测

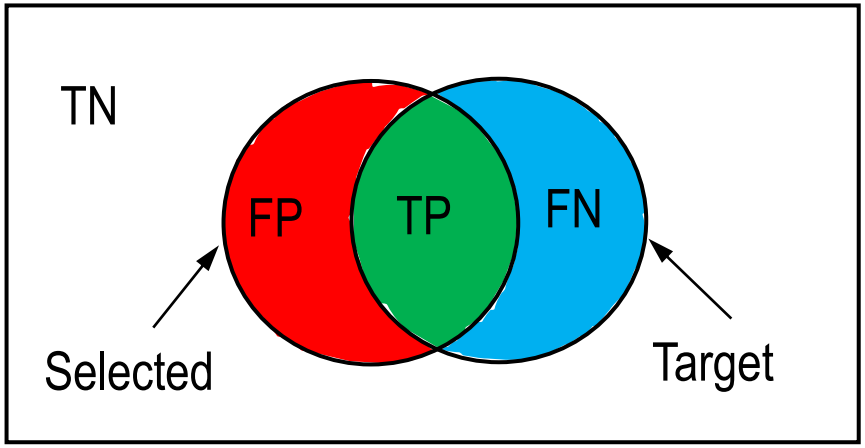
（随机猜测是指以固定的概率*p* 把输入分为正类）

	预测正类	预测负类
正类	TP	FN
负类	FP	TN



TPR=TP/(TP+FN)

FPR=FP/(FP+TN)



使用AUC指标比较分类算法

表 6.2　5种方法在三个数据上的结果

数据	方法	TP%	TN%	B-ACC%	AUC
Rooftop	C5.0	78.5	80.2	79.9	87.43
	CSVM	80.3	**81.8**	81.1	87.98
	C5.0 + SMOTE	79.9	80.1	80.0	88.22
	CSVM + SMOTE	81.3	80.4	80.9	87.87
	M3-SVM	**81.6**	81.4	**81.5**	**89.28**
Park	C5.0	82.6	85.8	84.2	90.39
	CSVM	84.9	85.5	85.2	93.93
	C5.0 + SMOTE	84.3	83.8	84.2	90.96
	CSVM + SMOTE	85.4	85.1	85.3	94.10
	M3-SVM	**87.2**	**87.7**	**87.5**	**94.54**
Abalone	C5.0	61.5	59.6	60.6	66.84
	CSVM	59.0	58.8	58.9	64.25
	C5.0 + SMOTE	64.5	62.4	63.5	69.53
	CSVM +SMOTE	62.7	63.3	63.0	68.00
	M3-SVM	**67.5**	**66.4**	**67.0**	**72.67**

二类分类器预测过程

- 一般二类分类器在预测时会计算一个评估函数

  $f(x;w)$

  其中，**x**为待预测样本的特征向量，**w**为已训练的参数。对于线性分类器有$f(x;w)=x\cdot w$。

- 预测结果如下输出：
  - 若 $f(x;w)>0$ ，**分类器输出正类**
  - 若 $f(x;w)<0$ ，**分类器输出负类**

二类分类器预测过程

- 通过引入阈值 t，改变分类器预测时对正负类的倾向：
  - 若 $f(x;w) > t$，分类器输出正类。
  - 若 $f(x;w) < t$，分类器输出负类。
- 阈值 t 增大，分类器预测结果偏向负类。
- 阈值 t 减小，分类器预测结果偏向正类。

ROC曲线的绘制

- ROC曲线以FPR为横轴，TPR为纵轴。
- 设置不同的阈值t，分类器预测结果有不同的FPR值和TPR值。

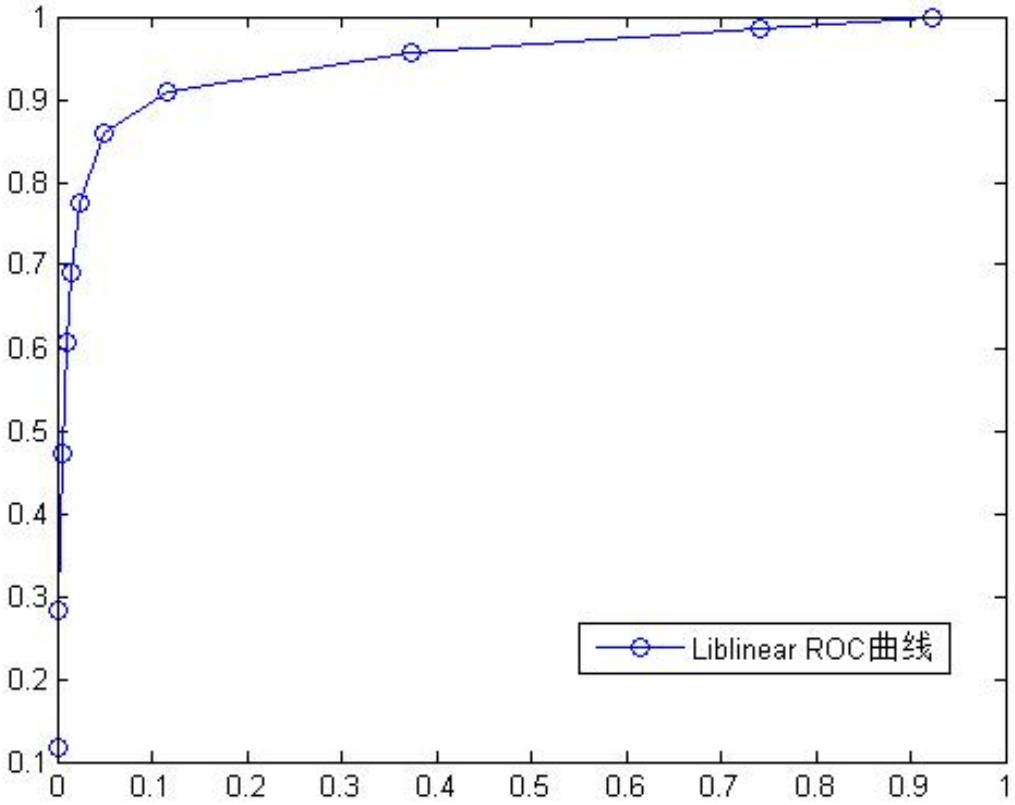
例：Liblinear的ROC曲线

- 分别以-8,-4,-2,-1,-0.5,0,0.5,1,2,4,8为阈值，使用Liblinear进行预测，结果如下表：

t	-8	-4	-2	-1	-0.5	0	0.5	1	2	4	8
FPR	0.001	0.002	0.005	0.010	0.014	0.025	0.049	0.115	0.373	0.741	0.924
TPR	0.117	0.282	0.472	0.606	0.691	0.774	0.859	0.910	0.956	0.986	0.998

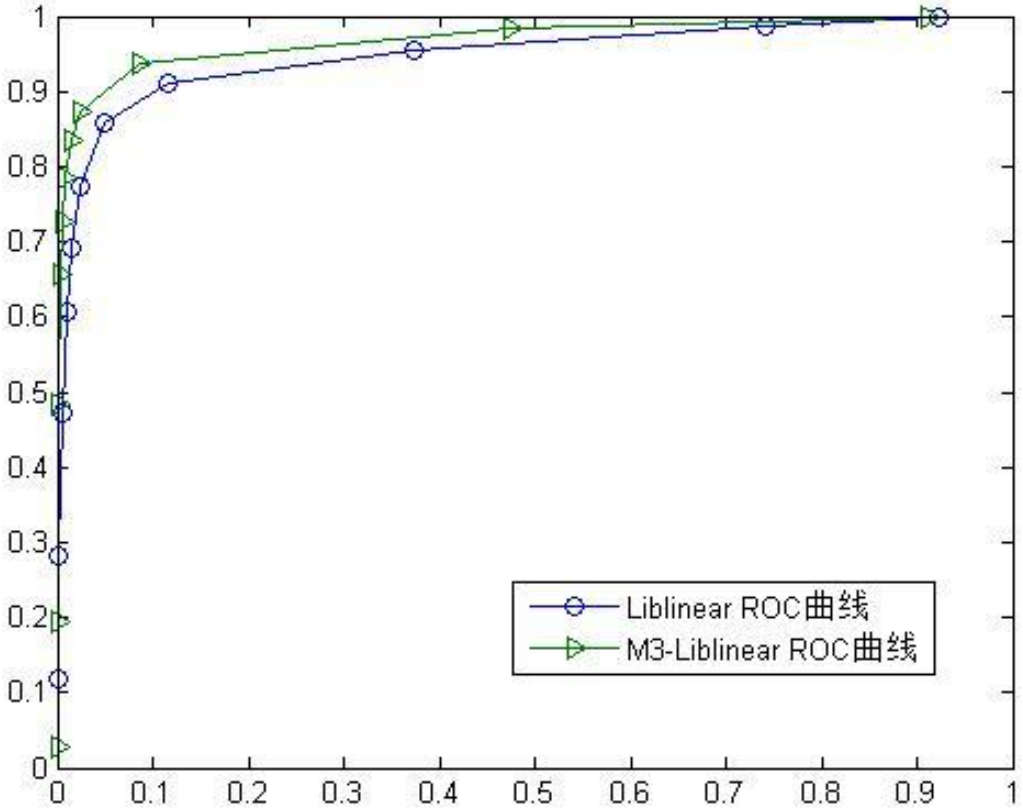
例：Liblinear的ROC曲线

- 使用画图工具将表格中的数据绘成图片。如下图：



ROC曲线与分类效果

- ROC曲线下方面积越大，分类效果越好。下图是Liblinear的ROC曲线与M3-Liblinear的ROC曲线。



谢谢！ 下周见！