

目录

01.概述、线性代数和NDArray	19
1 A17ACoTF_ND13O4_0	19
1.1 Slide 1	19
1.2 概要	20
1.3 深度学习	21
1.4 图像分类	22
1.5 图像分类	23
1.6 目标检测和分割	24
1.7 图片风格变换	25
1.8 人脸合成	26
1.9 自然语言向量类比	27
1.10 机器翻译	28
1.11 自然语言文本合成	29
1.12 自然语言自动问答	30
1.13 图像描述	31
1.14 线性代数	32
1.15 元素	33
1.16 向量	34
1.17 向量	35
1.18 向量	36
1.19 矩阵	37
1.20 矩阵	38
1.21 矩阵	39
1.22 矩阵	40
1.23 矩阵	41
1.24 矩阵	42
1.25 特殊矩阵	43
1.26 Special Matrices	44
1.27 GPUs 十分喜爱矩阵和向量 (它们有许多处理器)	45
1.28 NDArray	46
1.29 NDArray	47
1.30 NDArray	48
1.31 创建NDArray	49
1.32 元素索引	50
1.33 总结	51
02.概率与统计	52
1 CE1ACoTF_C8hWyK_0	52
1.1 Slide 1	52
1.2 概要	53
1.3 基础概率	54
1.4 概率	55
1.5 韦恩图	56
1.6 独立性与相关性	57
1.7 识别猫和狗本应该很容易.....	58
1.8 识别猫和狗本应该很容易.....	59
1.9 识别猫和狗本应该很容易.....	60

1.10 识别猫和狗本应该很容易.....	61
1.11 识别猫和狗本应该很容易.....	62
1.12 识别猫和狗本应该很容易.....	63
1.13 发生了什么？	64
1.14 不确定性和条件作用	65
1.15 贝叶斯法则	66
1.16 艾滋病检测实例	67
1.17 朴素贝叶斯	68
1.18 朴素贝叶斯 - 垃圾邮件 (spam) 过滤器	69
1.19 图形模型	70
1.20 朴素贝叶斯	71
1.21 Slide 21	72
1.22 朴素贝叶斯 - 垃圾邮件 (spam) 过滤器	73
1.23 朴素贝叶斯 - 垃圾邮件 (spam) 过滤器	74
1.24 简易模型	75
1.25 Slide 25	76
1.26 对数空间中的概率乘积	77
1.27 GPUs 的浮点数字	78
1.28 采样	79
1.29 均匀分布	80
1.30 离散分布	81
1.31 掷骰子	82
1.32 正态分布	83
1.33 正态分布	84
1.34 中心极限定理	85
1.35 总结	86
03. 导数、逆向传播和复杂度	87
1 F21FCoTF_6YG75r_0	87
1.1 Slide 1	87
1.2 概要	88
1.3 矩阵微积分	89
1.4 标量求导回顾	90
1.5 次导数	91
1.6 梯度	92
1.7 Slide 7	93
1.8 例子	94
1.9 Slide 9	95
1.10 Slide 10	96
1.11 例子	97
1.12 推广到矩阵	98
1.13 链式法则	99
1.14 链式法则	100
1.15 Slide 15	101
1.16 Slide 16	102
1.17 Slide 17	103
1.18 Slide 18	104
1.19 自动微分法	105
1.20 自动微分 (AD)	106

1.21 计算图	107
1.22 计算图	108
1.23 计算图	109
1.24 两种模式	110
1.25 反向传播	111
1.26 反向传播	112
1.27 反向传播	113
1.28 反向传播	114
1.29 反向传播 总结	115
1.30 复杂度	116
1.31 [拓展] 再具体化 (re-materialization)	117
1.32 再具体化 (re-materialization)	118
1.33 复杂度	119
1.34 总结	120
04.线性方法、基础优化和层序回归	121
1 1633CoTF_7S8pMp_0	121
1.1 Slide 1	121
1.2 概要	122
1.3 线性方法	123
1.4 房价预测 101	124
1.5 房价预测	125
1.6 一个简易模型	126
1.7 线性方法	127
1.8 线性方法是一个单层神经网络	128
1.9 神经科学的灵感	129
1.10 测量估计质量	130
1.11 训练数据集	131
1.12 学习参数	132
1.13 封闭解	133
1.14 基础优化	134
1.15 梯度下降	135
1.16 选择学习率	136
1.17 小批量随机梯度下降 (SGD)	137
1.18 选择批量值	138
1.19 总结	139
1.20 softmax 回归	140
1.21 回归与分类	141
1.22 Kaggle上的各种分类任务	142
1.23 Kaggle上的各种分类任务	143
1.24 Kaggle上的各种分类任务	144
1.25 从回归到多类分类	145
1.26 softmax 函数	146
1.27 softmax 梯度	147
1.28 是否用平方损失 ?	148
1.29 交叉熵	149
1.30 线性回归与softmax回归	150
1.31 总结	151
05.最大似然估计 和 逻辑回归	152

1 3A38CoTF_ded2Y4_0	152
1.1 Slide 1	152
1.2 概要	153
1.3 最大似然估计	154
1.4 正态分布	155
1.5 估计正态分布的参数	156
1.6 Slide 6	157
1.7 最大似然估计	158
1.8 Slide 8	159
1.9 Slide 9	160
1.10 Slide 10	161
1.11 Slide 11	162
1.12 Slide 12	163
1.13 最大后验估计	164
1.14 这与回归有什么关系？	165
1.15 回归	166
1.16 Slide 16	167
1.17 损失函数	168
1.18 平方损失 (L2 损失)	169
1.19 平方损失 (L2 损失)	170
1.20 L1 损失	171
1.21 Slide 21	172
1.22 Slide 22	173
1.23 Huber 损失	174
1.24 总结	175
06.多层感知机	176
1 5A55CoTF_Kk71nk_0	176
1.1 Slide 1	176
1.2 概要	177
1.3 感知机	178
1.4 感知机	179
1.5 感知机	180
1.6 感知机	181
1.7 训练感知机	182
1.8 Slide 8	183
1.9 Slide 9	184
1.10 Slide 10	185
1.11 Slide 11	186
1.12 收敛定理	187
1.13 结果	188
1.14 XOR 问题 (Minsky & Papert, 1969)	189
1.15 多层感知机	190
1.16 学习 XOR	191
1.17 单隐藏层	192
1.18 单隐藏层	193
1.19 单隐藏层	194
1.20 单隐藏层	195

1.21 单隐藏层	196
1.22 S型 (sigmoid) 激活函数	197
1.23 双曲正切 (tanh) 激活函数	198
1.24 线性 (ReLU) 修正函数	199
1.25 多类别分类	200
1.26 Slide 26	201
1.27 Slide 27	202
1.28 总结	203
07.模型选择，权重衰减 和 丢弃法	204
1 7A91CoTF_K8zy92_0	204
1.1 Slide 1	204
1.2 概要	205
1.3 模型选择	206
1.4 训练误差和泛化误差	207
1.5 验证数据集和测试数据集	208
1.6 K 折交叉验证	209
1.7 欠拟合 过拟合	210
1.8 欠拟合 和 过拟合	211
1.9 模型复杂度	212
1.10 模型复杂度的影响	213
1.11 估计模型复杂度	214
1.12 VC维度	215
1.13 线性分类器的VC维度	216
1.14 VC维度的效用	217
1.15 数据复杂度	218
1.16 权重衰减	219
1.17 平方正则化作为“硬”约束	220
1.18 平方正则化作为“软”约束	221
1.19 图解说明最优解	222
1.20 更新规则	223
1.21 丢弃法 (dropout)	224
1.22 动机	225
1.23 添加没有偏差的噪音	226
1.24 丢弃法 – 训练	227
1.25 丢弃法 – 推断	228
1.26 总结	229
08.数值稳定性，激活函数 和 硬件	230
1 9ACDCoTF_5L56Of_0	230
1.1 Slide 1	230
1.2 概要	231
1.3 数值稳定性	232
1.4 神经网络的梯度	233
1.5 深度神经网络的两个问题	234
1.6 例子： MLP	235
1.7 梯度爆炸	236
1.8 梯度爆炸的问题	237
1.9 梯度消失	238

1.10 梯度消失	239
1.11 梯度消失的问题	240
1.12 稳定模型训练	241
1.13 稳定模型训练	242
1.14 权重初始化	243
1.15 每层神经网络的常数方差	244
1.16 例子：MLP	245
1.17 Slide 17	246
1.18 反向均值和方差	247
1.19 Xavier 初始化	248
1.20 激活函数	249
1.21 简单的线性激活函数	250
1.22 反向	251
1.23 回顾激活函数	252
1.24 深度学习之硬件	253
1.25 GPU	254
1.26 Intel i7-6700K	255
1.27 提高CPU利用率 -- 一	256
1.28 案例分析	257
1.29 提高CPU利用率 -- 二	258
1.30 案例分析	259
1.31 Nvidia Titan X (Pascal)	260
1.32 提高GPU利用率	261
1.33 如何购买 GPUs	262
1.34 CPU 与 GPU	263
1.35 CPU/GPU 带宽	264
1.36 CPUs 与 GPUs	265
1.37 CPU/GPU 上编程	266
1.38 更有前景的硬件	267
1.39 Slide 39	268
1.40 数字信号处理器	269
1.41 现场可编程门阵列 (FPGA)	270
1.42 AI ASIC	271
1.43 收缩阵列	272
1.44 具有脉动阵列的矩阵乘法	273
1.45 具有脉动阵列的矩阵乘法	274
1.46 具有脉动阵列的矩阵乘法	275
1.47 具有脉动阵列的矩阵乘法	276
1.48 具有脉动阵列的矩阵乘法	277
1.49 具有脉动阵列的矩阵乘法	278
1.50 具有脉动阵列的矩阵乘法	279
1.51 具有脉动阵列的矩阵乘法	280
1.52 脉动阵列 (Systolic Array)	281
1.53 Slide 53	282
1.54 总结	283
09.泛化表现，协变量偏移 和 对抗性数据	284
1 C2C9CoTF_tCXZnd_0	284

1.1 Slide 1	284
1.2 概要	285
1.3 训练 ≠ 测试	286
1.4 泛化表现	287
1.5 泛化表现	288
1.6 泛化表现	289
1.7 泛化表现	290
1.8 为什么呢?	291
1.9 为什么呢?	292
1.10 为什么呢?	293
1.11 为什么呢?	294
1.12 怎么修复	295
1.13 怎么修复	296
1.14 怎么修复	297
1.15 Slide 15	298
1.16 训练集	299
1.17 测试时...	300
1.18 协变量转变 (Covariate Shift)	301
1.19 Slide 19	302
1.20 协变量转变 (Covariate Shift)	303
1.21 为什么呢?	304
1.22 协变量偏移校正	305
1.23 训练集	306
1.24 测试集	307
1.25 分类器在测试期间可能会执行得更差	308
1.26 简单的回归问题	309
1.27 简单的回归问题	310
1.28 简单的回归问题	311
1.29 简单的回归问题	312
1.30 训练误差可能会误导 (例如面孔)	313
1.31 没有对偏差的预防	314
1.32 TL; DR 在我们没有足够数量的训练数据时测试 , 可能会产生奇怪的结果。	315
1.33 协变量偏移校正	316
1.34 协变量偏移校正	317
1.35 协变量偏移校正	318
1.36 协变量偏移校正	319
1.37 标签偏移	320
1.38 训练集	321
1.39 测试集	322
1.40 标签转移	323
1.41 标签转移	324
1.42 标签转移	325
1.43 标签转移	326
1.44 CIFAR10上的黑盒协变量偏移校正	327
1.45 对抗性数据	328
1.46 Slide 46	329
1.47 对抗性数据 – 图像生成(e.g. Sharif et al. 2017)	330
1.48 对抗性数据 – 语言生成 (e.g. Carlini & Wagner, 2018)	331

1.49 '不自然'的数据	332
1.50 垃圾邮件防御	333
1.51 不变性	334
1.52 不变性	335
1.53 不变性和耐久的损失	336
1.54 非平稳环境	337
1.55 与环境的互动	338
1.56 批量	339
1.57 在线学习算法	340
1.58 有状态系统	341
1.59 强化学习	342
1.60 强化学习	343
1.61 训练 ≠ 测试	344
1.62 概要	345
10.深度学习框架	346
1 EAC5CoTF_n36K3c_0	346
1.1 Slide 1	346
1.2 Slide 2	347
1.3 Caffe	348
1.4 Tensorflow	349
1.5 Keras	350
1.6 Pytorch	351
1.7 MXNet	352
1.8 MXNet + Gluon	353
1.9 GluonCV	354
1.10 GluonNLP	355
1.11 DGL	356
1.12 2019 展望	357
1.13 Slide 13	358
1.14 Gluon - 命令式神经网络API	359
11.卷积和池化层	360
1 AF2CoTF_FVm9Yi_0	360
1.1 Slide 1	360
1.2 概要	361
1.3 Slide 3	362
1.4 分类图像中的狗和猫	363
1.5 单隐含层网络	364
1.6 Slide 6	365
1.7 两个原则	366
1.8 重新思考 - 稠密层	367
1.9 原则 #1 - 平移不变性	368
1.10 原则 #2 - 局部性	369
1.11 卷积层	370
1.12 二维交叉相关	371
1.13 二维卷积层	372
1.14 例子	373
1.15 例子	374

1.16 互相关与卷积	375
1.17 1-D 和 3-D 交叉相关	376
1.18 填充和步幅	377
1.19 填充	378
1.20 填充	379
1.21 填充	380
1.22 步幅	381
1.23 步幅	382
1.24 步幅	383
1.25 多个输入和输出通道	384
1.26 多个输入通道	385
1.27 多个输入通道	386
1.28 多个输入通道	387
1.29 多个输入通道	388
1.30 多个输出通道	389
1.31 多个输入和输出通道	390
1.32 1x1 卷积层	391
1.33 Slide 33	392
1.34 池化层	393
1.35 池化层	394
1.36 2-D 最大池化	395
1.37 2-D 最大池化	396
1.38 池化层 - 填充，步幅和多个通道	397
1.39 平均池化层	398
1.40 总结	399
12.LeNet, AlexNet, VGG 和 NiN	400
1 32EECoTF_HmmWxh_0	400
1.1 Slide 1	400
1.2 概要	401
1.3 LeNet 架构	402
1.4 手写的数字识别	403
1.5 MNIST	404
1.6 Slide 6	405
1.7 Slide 7	406
1.8 MXNet 中的 LeNet	407
1.9 AlexNet	408
1.10 机器学习	409
1.11 几何学	410
1.12 特征工程	411
1.13 硬件	412
1.14 ImageNet 数据集 (2010)	413
1.15 AlexNet	414
1.16 AlexNet 架构	415
1.17 AlexNet 架构	416
1.18 AlexNet 架构	417
1.19 更多技巧	418
1.20 复杂度	419

1.21 VGG	420
1.22 VGG	421
1.23 VGG 块	422
1.24 VGG 结构	423
1.25 进程	424
1.26 Slide 26	425
1.27 网络中的网络	426
1.28 最后一层的诅咒	427
1.29 最后一层的诅咒	428
1.30 VGG 参数	429
1.31 打破最后一层诅咒	430
1.32 为什么用 1x1 卷积?	431
1.33 NiN 块	432
1.34 NiN 网络	433
1.35 NiN 最后一层	434
1.36 NiN 总结	435
1.37 总结	436
13.Inception, 批量归一化 和 残差网络 (ResNet)	437
1 5702CoTF_1gKpb9_0	437
1.1 Slide 1	437
1.2 概要	438
1.3 Inception	439
1.4 选最合适的卷积 ...	440
1.5 干嘛选呢? 都用就是了。	441
1.6 Inception 块	442
1.7 Inception 块	443
1.8 Inception 块	444
1.9 GoogLeNet	445
1.10 阶段 1 & 2	446
1.11 阶段 3	447
1.12 阶段 4 & 5	448
1.13 许多种类的 Inception 网络	449
1.14 Inception V3 块 - 阶段 3	450
1.15 Inception V3 块 - 阶段 4	451
1.16 Inception V3 块 - 阶段 5	452
1.17 Slide 17	453
1.18 Batch Normalization	454
1.19 批量归一化	455
1.20 批量归一化	456
1.21 这是最初的动机.....	457
1.22 批量归一化	458
1.23 细节	459
1.24 残差网络 (ResNet)	460
1.25 添加层会提高准确性吗?	461
1.26 残差网络 (ResNet)	462
1.27 残差块	463

1.28 残差块	464
1.29 不同的残差块	465
1.30 残差块	466
1.31 残差网络 (ResNet)	467
1.32 Slide 32	468
1.33 残差网络 (ResNet)	469
1.34 降低卷积的成本	470
1.35 降低卷积的成本	471
1.36 RexNeXt 成本	472
1.37 更多模型	473
1.38 稠密连接网络 (DenseNet)	474
1.39 Squeeze-Excite Net	475
1.40 ShuffleNet (Zhang et al., 2018)	476
1.41 可分离的卷积层 - 所有通道分开	477
1.42 总结	478
14.混合编程, 异步计算, 多GPU训练 - hf	479
1 7F0ECoTF_1xKa29_0	479
1.1 Slide 1	479
1.2 概要	480
1.3 命令式与符号式编程的混合	481
1.4 命令式编程	482
1.5 符号式编程	483
1.6 Gluon 中的混合编程	484
1.7 异步计算	485
1.8 异步计算	486
1.9 自动并行	487
1.10 编写并行程序很痛苦...	488
1.11 自动并行	489
1.12 多GPU训练	490
1.13 数据并行	491
1.14 分布训练	492
1.15 分布计算	493
1.16 GPU机器层次结构	494
1.17 一次迭代	495
1.18 一次迭代	496
1.19 一次迭代	497
1.20 一次迭代	498
1.21 一次迭代	499
1.22 一次迭代	500
1.23 一次迭代	501
1.24 一次迭代	502
1.25 同步 SGD	503
1.26 性能	504
1.27 性能权衡	505
1.28 实际建议	506
1.29 总结	507
15.图像增广，微调 和 样式迁移	508
1 9F5ACoTF_sr2AMS_0	508

1.1 Slide 1	508
1.2 概要	509
1.3 图像增广	510
1.4 CES'19 真实故事	511
1.5 数据增广	512
1.6 图像增广	513
1.7 反转	514
1.8 裁剪	515
1.9 变色	516
1.10 更多方法	517
1.11 微调	518
1.12 标记数据集非常昂贵	519
1.13 网络架构	520
1.14 微调	521
1.15 微调中的权重初始化	522
1.16 微调训练	523
1.17 重用已有分类器的参数	524
1.18 修复一些层	525
1.19 样式迁移	526
1.20 Slide 20	527
1.21 Slide 21	528
1.22 Slide 22	529
1.23 Slide 23	530
1.24 Slide 24	531
1.25 Slide 25	532
1.26 Slide 26	533
1.27 神经元风格	534
1.28 内容损失	535
1.29 样式损失	536
1.30 噪声损失	537
1.31 总损失函数	538
1.32 Slide 32	539
1.33 总结	540
16. 目标检测，计算机视觉训练技巧	541
1 C36ECoTF_k0H9xy_0	541
1.1 Slide 1	541
1.2 概要	542
1.3 Slide 3	543
1.4 Slide 4	544
1.5 Slide 5	545
1.6 边界框和锚框	546
1.7 边界框	547
1.8 目标检测数据集	548
1.9 锚框	549
1.10 交并比	550
1.11 将标签分配给锚框	551
1.12 将标签分配给锚框	552
1.13 将标签分配给锚框	553

1.14 将标签分配给锚框	554
1.15 将标签分配给锚框	555
1.16 输出预测边界框	556
1.17 区域卷积神经网络	557
1.18 区域卷积神经网络 (R-CNN)	558
1.19 RoI 池化层	559
1.20 Fast RCNN	560
1.21 Faster R-CNN	561
1.22 Slide 22	562
1.23 Mask R-CNN	563
1.24 单发多框检测 (SSD)	564
1.25 生成锚框	565
1.26 单发多框检测 (SSD) 模型	566
1.27 Slide 27	567
1.28 你只需看一次(YOLO)	568
1.29 YOLO	569
1.30 Slide 30	570
1.31 计算机视觉 训练技巧	571
1.32 Slide 32	572
1.33 Slide 33	573
1.34 混合训练数据	574
1.35 混合训练数据	575
1.36 标签平滑化	576
1.37 学习率预热	577
1.38 余弦衰减	578
1.39 同步批量归一化	579
1.40 随机批量调整形状	580
1.41 图像分类	581
1.42 YOLO v3 结果	582
1.43 总结	583
18.序列模型	584
1 EB6BCoTF_M8yM4x_0	584
1.1 Slide 1	584
1.2 概要	585
1.3 相关的随机变量	586
1.4 数据	587
1.5 训练 ≠ 测试	588
1.6 时间重要性 (Koren, 2009)	589
1.7 时间重要性 (Koren, 2009)	590
1.8 Slide 8	591
1.9 Slide 9	592
1.10 Slide 10	593
1.11 空间重要性 – 地震预测	594
1.12 FTSE 100	595
1.13 TL;DR – 数据通常不是完全相同且相互独立	596
1.14 序列模型	597
1.15 序列模型	598

1.16 序列模型	599
1.17 序列模型	600
1.18 计划 A - 马尔可夫 (Markov) 假设	601
1.19 计划 A - 马尔可夫 (Markov) 假设	602
1.20 计划 B – 隐马尔可夫模型	603
1.21 计划 B – 隐马尔可夫模型	604
1.22 隐变量模型 (经典)	605
1.23 时间聚类 - 隐马尔可夫模型	606
1.24 时间PCA - 卡尔曼滤波器	607
1.25 序列模型	608
1.26 序列模型	609
1.27 计划 A - 马尔可夫 (Markov) 假设	610
1.28 马尔可夫 (Markov) 假设	611
1.29 语言模型	612
1.30 语言模型 101	613
1.31 N-grams (更长的标记序列)	614
1.32 我们来看看实际的语言统计数据	615
1.33 文本预处理	616
1.34 符号化	617
1.35 小批量生成	618
1.36 小批量生成	619
1.37 小批量生成	620
1.38 代码 ...	621
1.39 总结	622
19.循环神经网络	623
1 1367CoTF_G7xXOv_0	623
1.1 Slide 1	623
1.2 概要	624
1.3 循环神经网络	625
1.4 隐变量模型	626
1.5 循环神经网络	627
1.6 循环神经网络	628
1.7 代码 ...	629
1.8 实现RNN语言模型	630
1.9 输入编码	631
1.10 输入编码	632
1.11 具有隐含状态机制的RNN	633
1.12 输出编码	634
1.13 梯度	635
1.14 困惑度	636
1.15 代码 ...	637
1.16 截断通过时间反向传播	638
1.17 循环神经网络	639
1.18 目标函数	640
1.19 隐含状态梯度	641
1.20 隐含状态梯度	642
1.21 隐含状态梯度	643
1.22 隐含状态梯度	644

1.23 截断通过时间反向传播	645
1.24 截断通过时间反向传播	646
1.25 截断通过时间反向传播	647
1.26 计算图	648
1.27 详细示例	649
1.28 详细示例	650
1.29 详细示例	651
1.30 Truncation in practice	652
1.31 门控循环单元 (GRU)	653
1.32 在一个序列的注意力	654
1.33 门控循环单元	655
1.34 候选隐含状态	656
1.35 隐含状态	657
1.36 门控循环单元 (GRU) 总结	658
1.37 代码 ...	659
1.38 长短期记忆 (LSTM)	660
1.39 电门联想	661
1.40 长短期记忆 (LSTM)	662
1.41 输入门、遗忘门和输出门	663
1.42 候选记忆细胞	664
1.43 记忆细胞	665
1.44 隐含状态	666
1.45 长短期记忆 (LSTM) 总结	667
1.46 代码 ...	668
1.47 总结	669
20.高级循环神经网络	670
1 377CCoTF_l4xjNu_0	670
1.1 Slide 1	670
1.2 概要	671
1.3 深度循环 神经网络	672
1.4 使用循环神经网络	673
1.5 使用循环神经网络	674
1.6 回顾 - 循环神经网络	675
1.7 计划 A - 单元的非线性	676
1.8 计划 A - 单元的非线性	677
1.9 计划 B - 深度循环神经网络	678
1.10 Slide 10	679
1.11 代码 ...	680
1.12 双向循环神经网络	681
1.13 “未来”的重要性	682
1.14 “未来”的重要性	683
1.15 “未来”的重要性	684
1.16 回顾 - 图模型	685
1.17 动态编程	686
1.18 动态编程	687
1.19 动态编程	688
1.20 Slide 20	689

1.21 动态编程	690
1.22 应用到 RNN 上 ?	691
1.23 双向循环神经网络	692
1.24 这不适用于序列生成...	693
1.25 为什么呢?	694
1.26 原因	695
1.27 循环神经网络 的残差网络	696
1.28 回顾 - 添加层是否可以提高准确性 ?	697
1.29 回顾 - 残差网络 (ResNet)	698
1.30 回顾 – 深度循环神经网络	699
1.31 残差网络 (ResNet)	700
1.32 循环神经网络 的稠密连接网络	701
1.33 循环神经网络的稠密连接网络 (DenseNet)	702
1.34 循环神经网络的正则化	703
1.35 过拟合	704
1.36 回顾 – 丢弃法	705
1.37 变分丢弃法 (Gal & Ghahramani, 2015)	706
1.38 Zoneout (Krueger et al., 2016)	707
1.39 更多技巧	708
22. 嵌入向量, 词嵌入, 子词嵌入, 全局向量的词嵌入	709
1 5B90CoTF_8EVB0a_0	709
1.1 Slide 1	709
1.2 概要	710
1.3 词嵌入 (Word2vec)	711
1.4 动机	712
1.5 词嵌入 (Word2vec)	713
1.6 Skip-Gram 模型	714
1.7 似然函数	715
1.8 负采样	716
1.9 负抽样	717
1.10 CBOW 模型	718
1.11 似然函数	719
1.12 代码 ...	720
1.13 更多嵌入模型	721
1.14 子词嵌入 (fastText)	722
1.15 子词嵌入 (fastText)	723
1.16 Slide 16	724
1.17 全局向量的词嵌入 (GloVe)	725
1.18 全局向量的词嵌入 (GloVe)	726
1.19 代码...	727
1.20 总结	728
23. 编码器解码器 , Seq2seq模型 , 束搜索	729
1 7BCCCoTF_pkTrW3_0	729
1.1 Slide 1	729
1.2 概要	730
1.3 编码器 - 解码器 (Encoder - Decoder)	731
1.4 重新思考 CNN	732

1.5 重新思考 RNN	733
1.6 编码器 - 解码器架构	734
1.7 编码器 Class	735
1.8 解码器 Class	736
1.9 编码器-解码器 Class	737
1.10 Seq2seq 模型	738
1.11 机器翻译	739
1.12 Seq2seq 模型	740
1.13 Seq2seq 模型	741
1.14 机器翻译训练	742
1.15 代码...	743
1.16 束搜索 Beam Search	744
1.17 贪婪搜索	745
1.18 穷举搜索 (exhaustive search)	746
1.19 束搜索	747
1.20 束搜索	748
1.21 总结	749
24. 注意力机制	750
1 9BF8CoTF_Xz9HLd_0	750
1.1 Slide 1	750
1.2 概要	751
1.3 注意力机制	752
1.4 动机	753
1.5 注意力层	754
1.6 注意力层	755
1.7 点乘注意力	756
1.8 多层感知注意力	757
1.9 代码...	758
1.10 Seq2seq 与注意力机制	759
1.11 模型架构	760
1.12 编码器—解码器上的注意力机制	761
1.13 代码...	762
1.14 变换器模型 (Transformer)	763
1.15 自注意力机制	764
1.16 Transformer模型架构	765
1.17 多头注意力机制 (Multi-head Attention)	766
1.18 位置前馈网络	767
1.19 添加与归一化	768
1.20 位置编码	769
1.21 预测	770
1.22 代码 ...	771
1.23 BERT	772
1.24 NLP中的迁移学习 (Transfer Learning)	773
1.25 BERT 的动机	774
1.26 BERT 架构	775
1.27 输入	776
1.28 预训练任务1：掩码语言模型	777

1.29 预训练任务2：下一句话预测	778
1.30 用 Bert 微调	779
1.31 语句分类	780
1.32 命名实体识别	781
1.33 自动问答	782
1.34 GluonNLP 有更多资源/代码/模型... https://gluon-nlp.mxnet.io/	783
25.优化问题	784
1 C00DCoTF_Zq8skb_0	784
1.1 Slide 1	784
1.2 概要	785
1.3 优化问题	786
1.4 优化问题	787
1.5 局部最小值和全局最小值	788
1.6 凸集	789
1.7 凸函数	790
1.8 一阶特征条件	791
1.9 二阶特征条件	792
1.10 常见凸集和非凸集	793
1.11 凸优化	794
1.12 凸优化证明	795
1.13 梯度下降	796
1.14 算法	797
1.15 学习率的选择	798
1.16 收敛率	799
1.17 收敛率证明	800
1.18 收敛率证明 II	801
1.19 收敛率证明 III	802
1.20 深度学习应用	803
1.21 随机梯度下降 (SGD)	804
1.22 算法	805
1.23 例子	806
1.24 收敛率	807
1.25 实际应用	808
1.26 代码...	809
1.27 小批量随机梯度下降 (mini-batch SGD)	810
1.28 算法	811
1.29 代码...	812
1.30 总结	813

动手学深度学习

1. 概述、线性代数和NDArray

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/introduction.html>

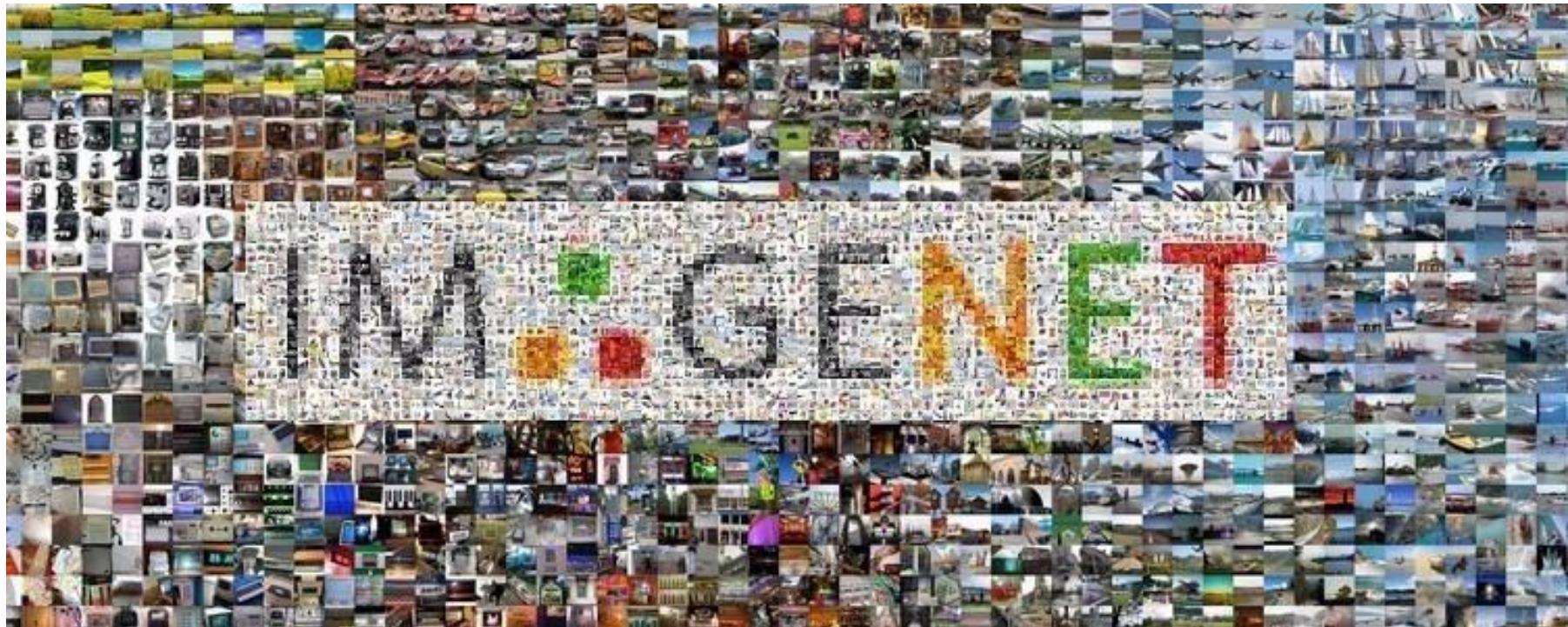
概要

- 深度学习简要概述
- 线性代数
- NDArray



深度学习

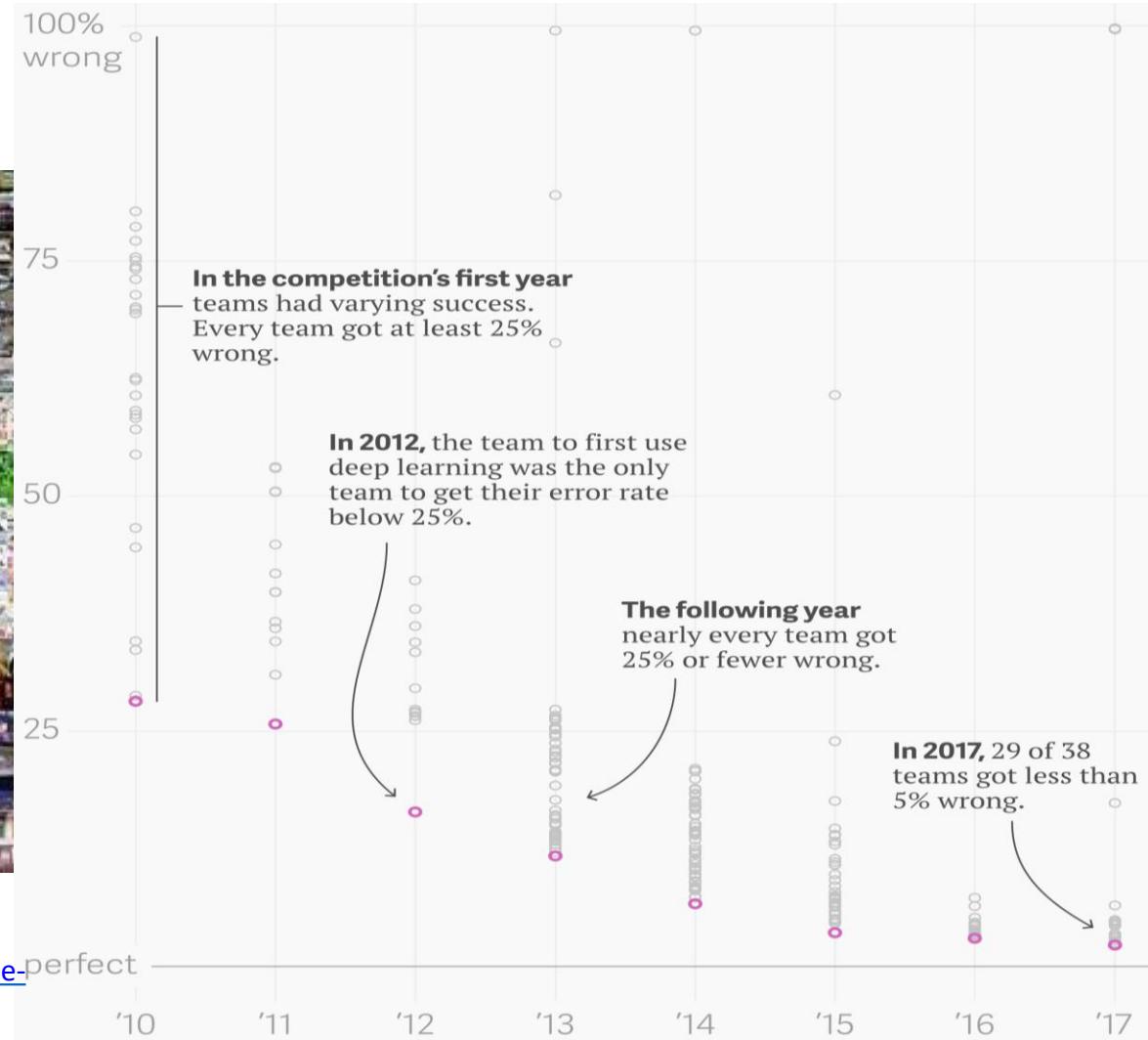
图像分类



<http://www.image-net.org/>

D2L.ai

图像分类



Yanofsky, Quartz

<https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/>

目标检测和分割



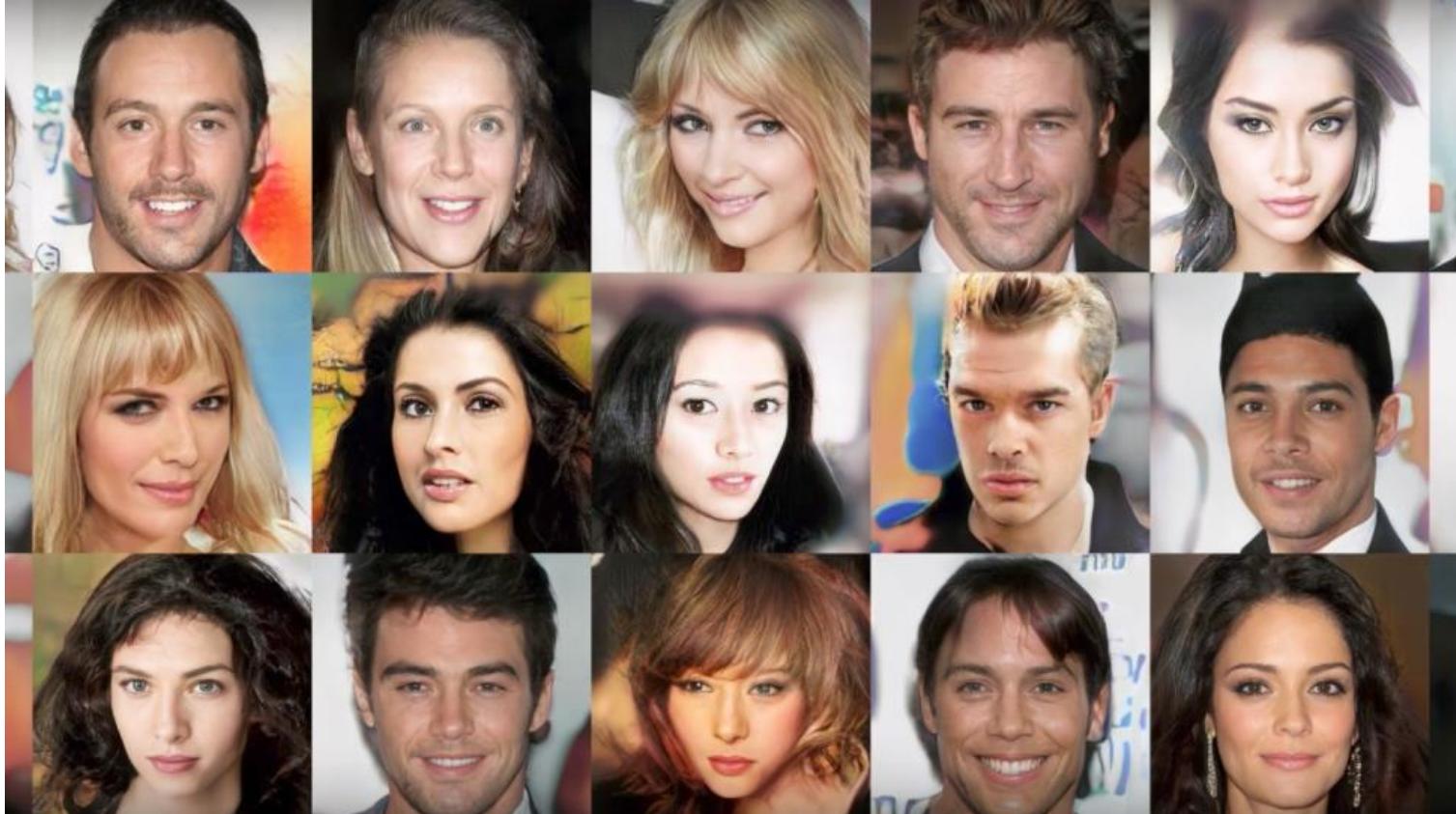
https://github.com/matterport/Mask_RCNN

图片风格变换



<https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/>

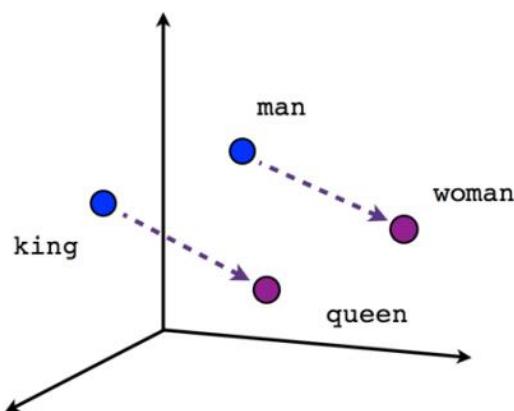
人脸合成



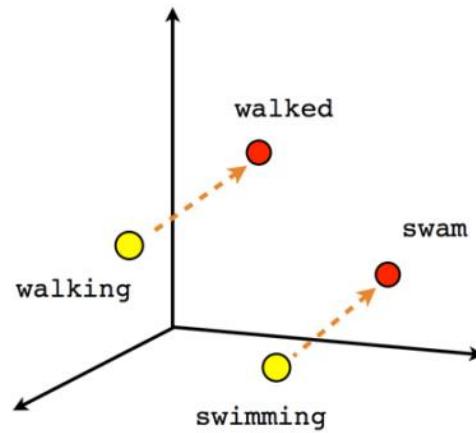
Karras et al, ICLR 2018

D2L.ai

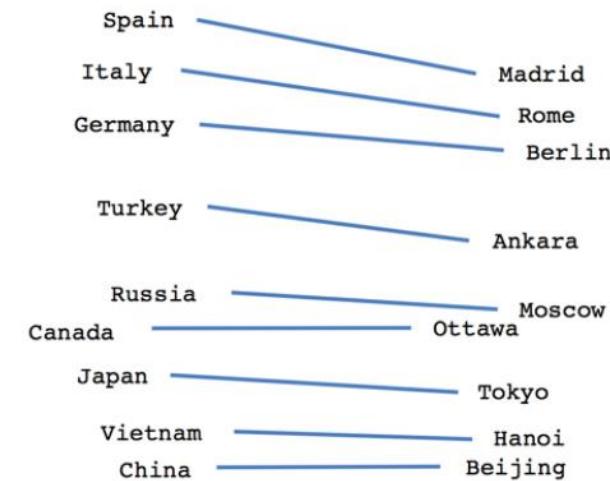
自然语言向量类比



Male-Female



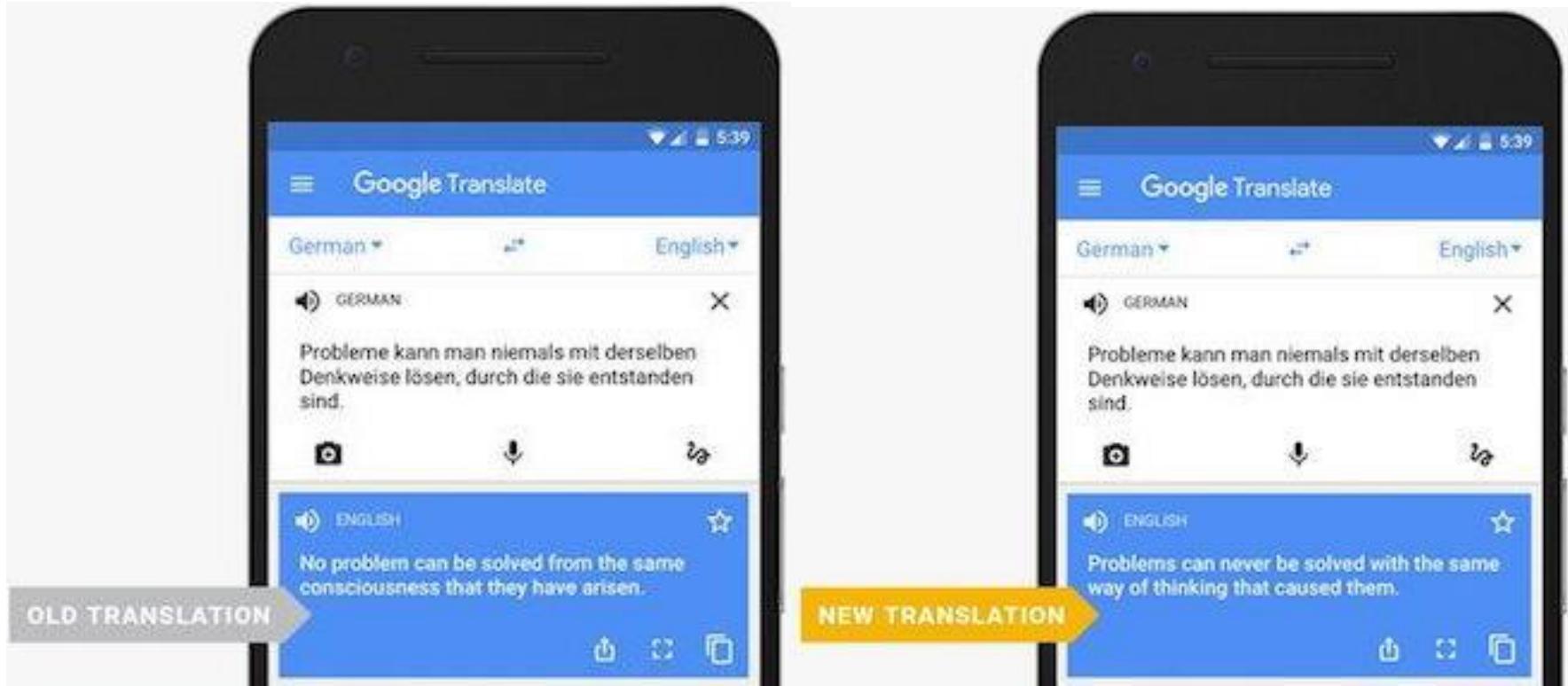
Verb tense



Country-Capital

<https://www.tensorflow.org/tutorials/word2vec>

机器翻译



<https://www.pc当地.com/news/349610/google-expands-neural-networks-for-language-translation>

自然语言文本合成

Content: Two dogs play by a tree.

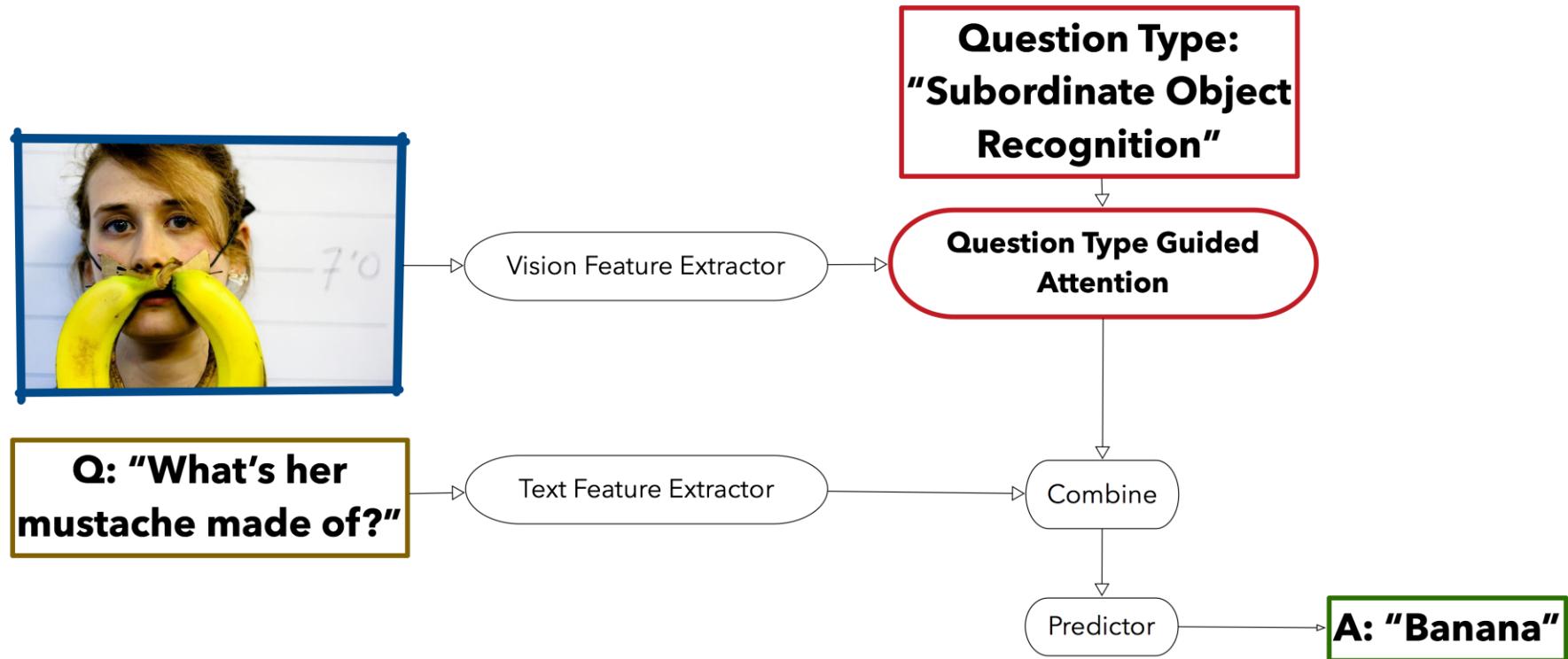
Style: **happily, love**



Two dogs **in love** play **happily** by a tree.

Li et al, NACCL, 2018

自然语言自动问答



图像描述

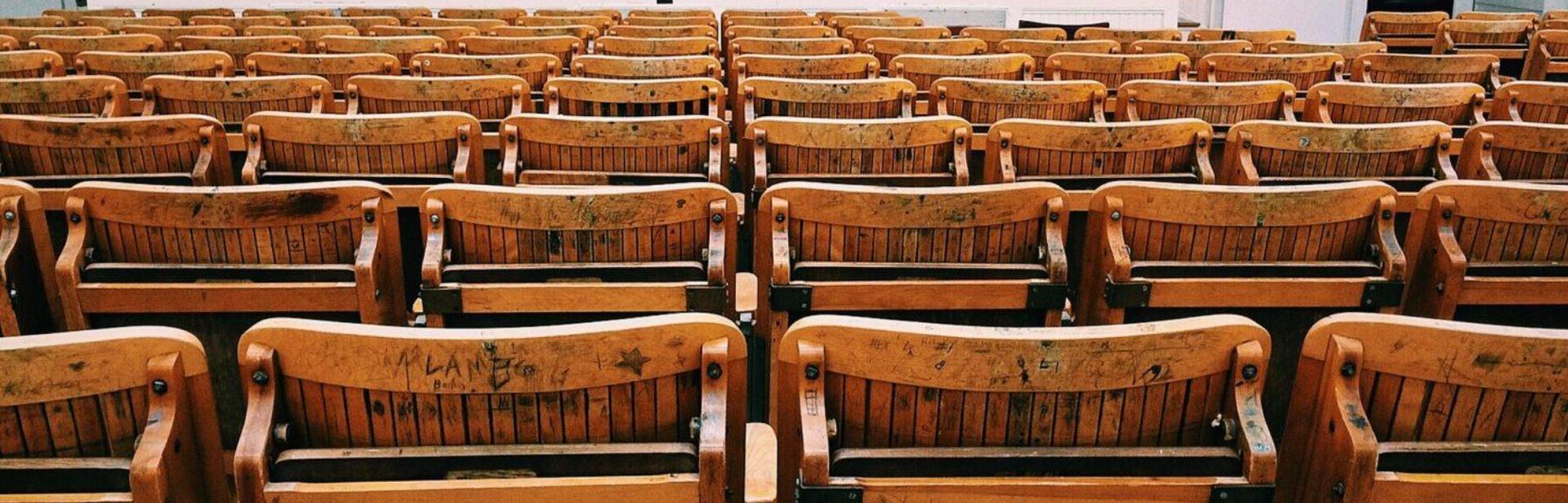
Human captions from the training set



Shallue et al, 2016

<https://ai.googleblog.com/2016/09/show-and-tell-image-captioning-open.html>

线性代数



元素



- 简易运算

$$c = a + b$$

$$c = a \cdot b$$

$$c = \sin a$$

- 范数

$$|a| = \begin{cases} a & \text{if } a > 0 \\ -a & \text{otherwise} \end{cases}$$

$$|a + b| \leq |a| + |b|$$

$$|a \cdot b| = |a| \cdot |b|$$

向量



- 简易运算

$$c = a + b \quad \text{where } c_i = a_i + b_i$$

$$c = \alpha \cdot b \quad \text{where } c_i = \alpha b_i$$

$$c = \sin a \quad \text{where } c_i = \sin a_i$$

- 范数

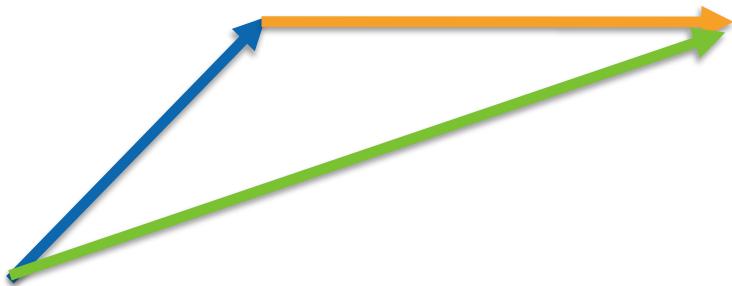
$$\|a\|_2 = \left[\sum_{i=1}^m a_i^2 \right]^{\frac{1}{2}}$$

$$\|a\| \geq 0 \text{ for all } a$$

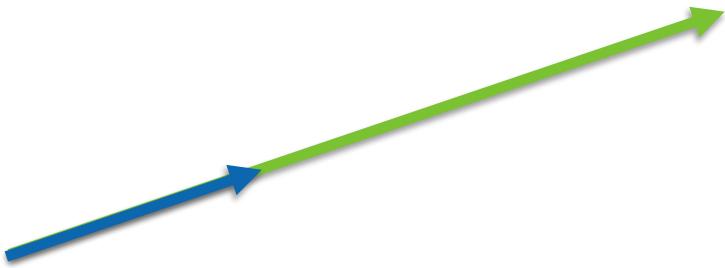
$$\|a + b\| \leq \|a\| + \|b\|$$

$$\|a \cdot b\| = |a| \cdot \|b\|$$

向量



$$c = a + b$$



$$c = \alpha \cdot b$$

Mathematician's 'parallel for all do'

向量

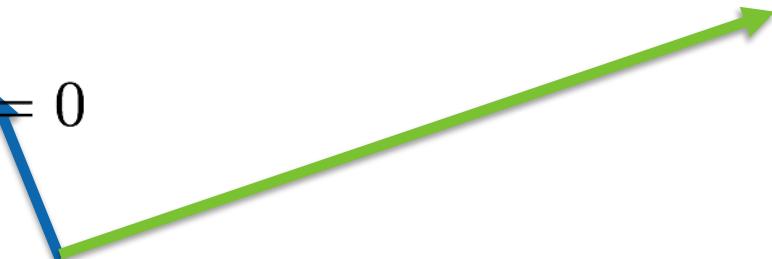


- 点积

$$a^\top b = \sum_i a_i b_i$$

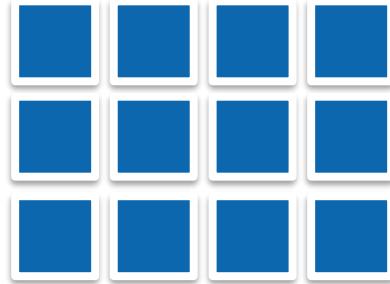
- 正交性

$$a^\top b = \sum_i a_i b_i = 0$$



如果我们有两个向量与第三个正交，它们的线性组合向量也正交

矩阵



• 简易运算

$$C = A + B$$

where $C_{ij} = A_{ij} + B_{ij}$

$$C = \alpha \cdot B$$

where $C_{ij} = \alpha B_{ij}$

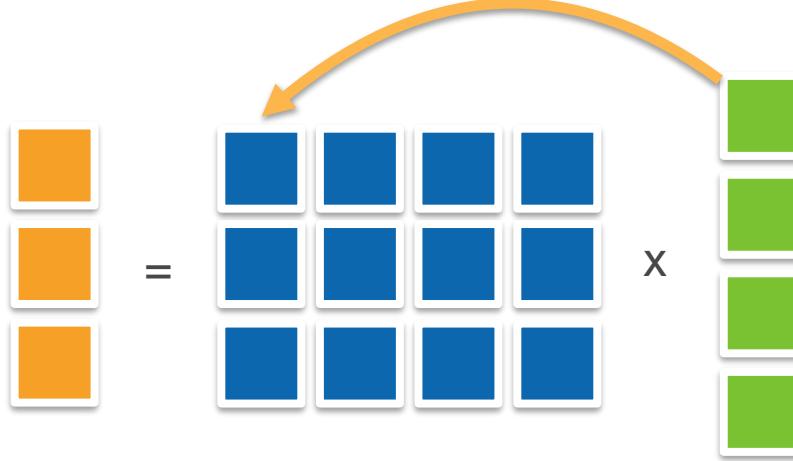
$$C = \sin A$$

where $C_{ij} = \sin A_{ij}$

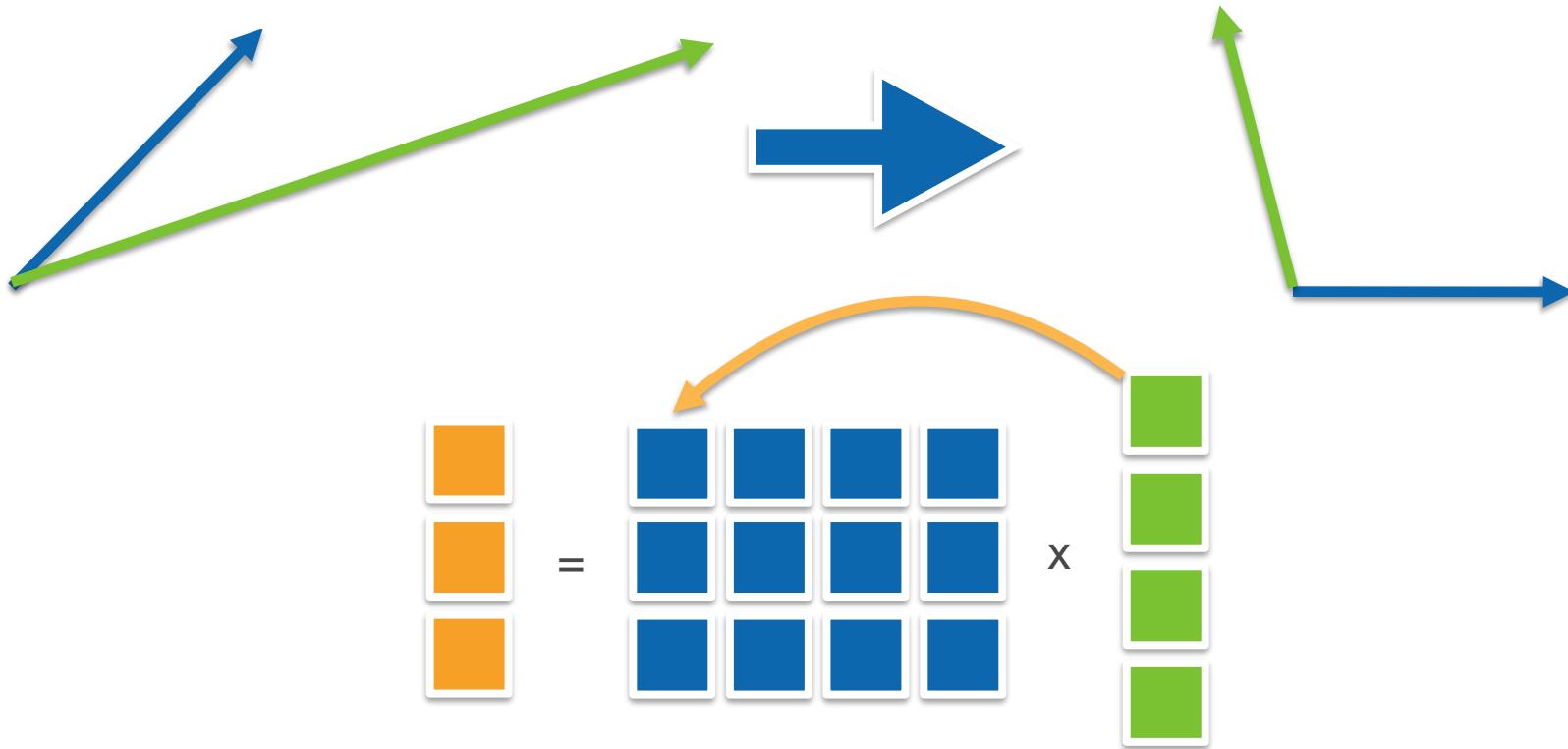
矩阵

- 矩阵相乘 (矩阵 \times 向量)

$$c = Ab \text{ where } c_i = \sum_j A_{ij} b_j$$



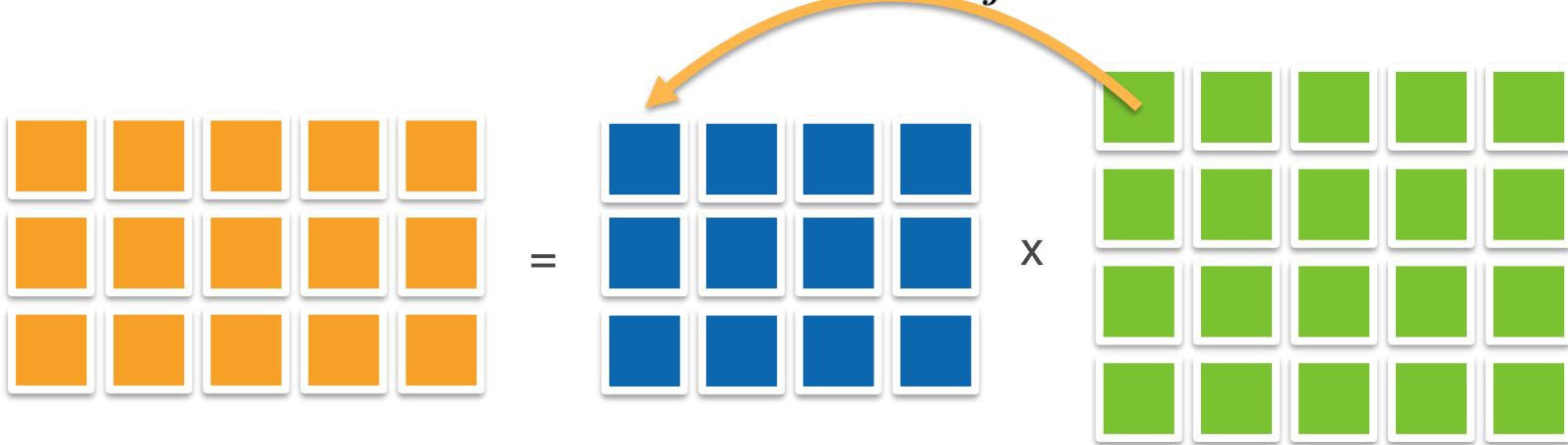
矩阵



矩阵

- 矩阵相乘 (矩阵 \times 矩阵)

$$C = AB \text{ where } C_{ik} = \sum_j A_{ij}B_{jk}$$



矩阵

- 范数

$$c = A \cdot b \text{ hence } \|c\| \leq \|A\| \cdot \|b\|$$

- 常见范数

- Frobenius 范数

$$\|A\|_{\text{Frob}} = \left[\sum_{ij} A_{ij}^2 \right]^{\frac{1}{2}}$$

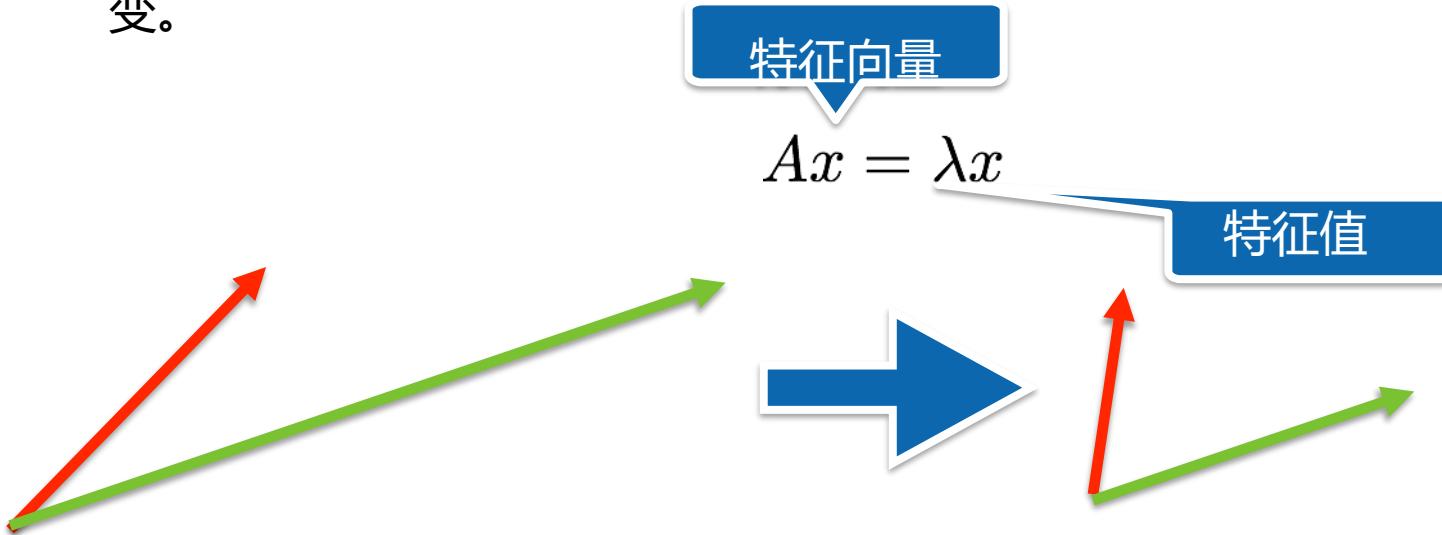
- H-infinity 范数

$$\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2} = \sqrt{\text{trace}(A^* A)} = \sqrt{\sum_{i=1}^{\min\{m, n\}} \sigma_i^2}$$

矩阵

• 特征值和特征向量

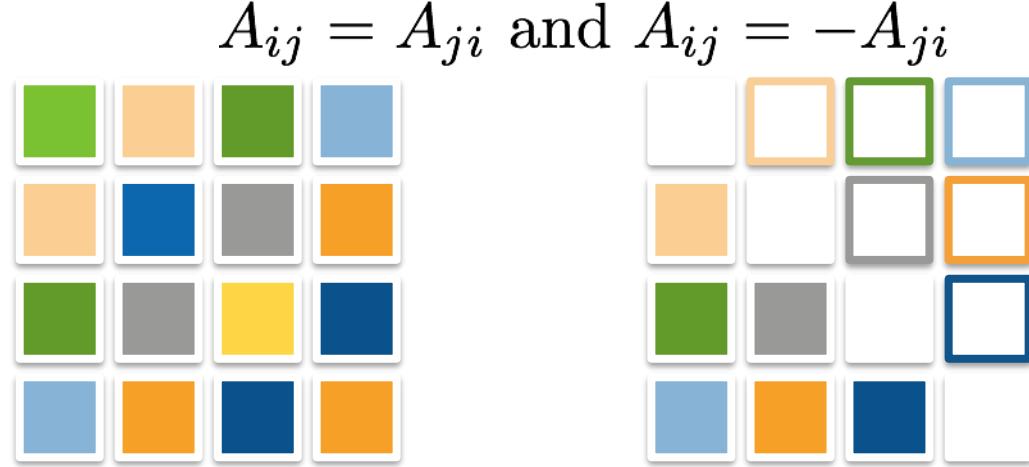
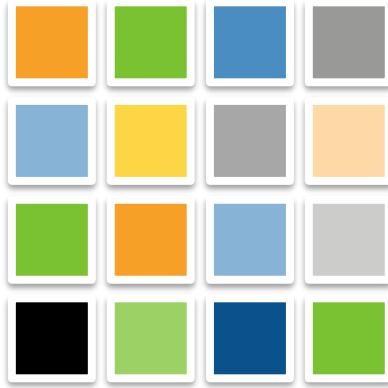
- 对于一个给定的线性变换A，它的特征向量x，经过这个线性变换之后，得到的新向量仍然与原来的v保持在同一条直线上，但其长度或方向也许会改变。



- 对称矩阵总会有相应的特征向量和特征值

特殊矩阵

- 对称性 & 非对称性



- 正定性

$$\|x\|^2 = x^\top x \geq 0 \text{ generalizes to } x^\top Ax \geq 0$$

(特征值均为非负数)

Special Matrices

- 正交矩阵

- 所有的列向量都是单位正交向量
- 所有的行向量都是单位正交向量
- 可以写为：

$$UU^\top = \mathbf{1}$$

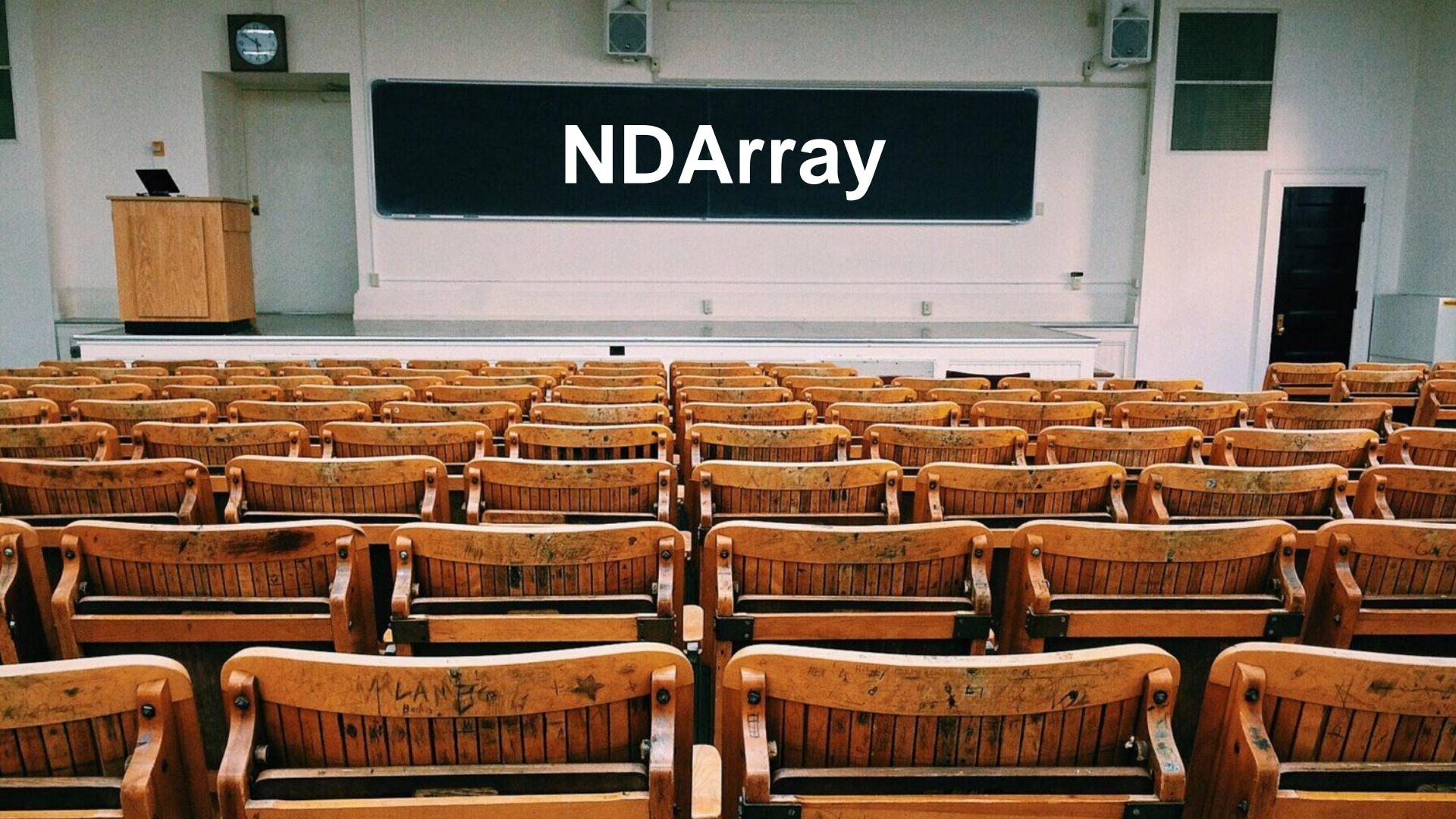
- 置换矩阵

矩阵的每一行和每一列的元素中只有一个1,其余元素都为0。

P where $P_{ij} = 1$ if and only if $j = \pi(i)$



GPUs 十分喜爱矩阵和向量
(它们有许多处理器)

A photograph of a large lecture hall. In the foreground, there are rows of wooden student desks and chairs, many of which have graffiti on them. In the middle ground, there is a large chalkboard. On the left side, there is a wooden podium with a laptop on it. A clock is mounted on the wall above the door on the left. The room has white walls and a door on the right.

NDArray

NDArray

- NDArray是存储和变换数据的主要工具
- NDArray提供GPU计算和自动求梯度等更多功能，这些使NDArray更加适合深度学习

0-d (元素)



1.0

一个类标签

1-d (向量)



[1.0, 2.7, 3.4]

一个特征向量

2-d (矩阵)

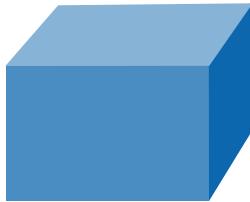


[[1.0, 2.7, 3.4]
[5.0, 0.2, 4.6]
[4.3, 8.5, 0.2]]

一个特征矩阵

NDArray

3-d



```
[[[0.1, 2.7, 3.4]  
[5.0, 0.2, 4.6]  
[4.3, 8.5, 0.2]]  
[[3.2, 5.7, 3.4]  
[5.4, 6.2, 3.2]  
[4.1, 3.5, 6.2]]]
```

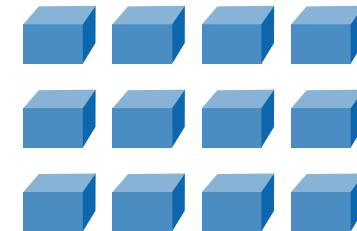
一张RGB图像
(长 x 宽 x 高)

4-d



```
[[[[...  
...  
...]]]]
```

5-d



```
[[[[...  
...  
...]]]]
```

一组RGB图像
(批量大小 x 长
x 宽 x 高)

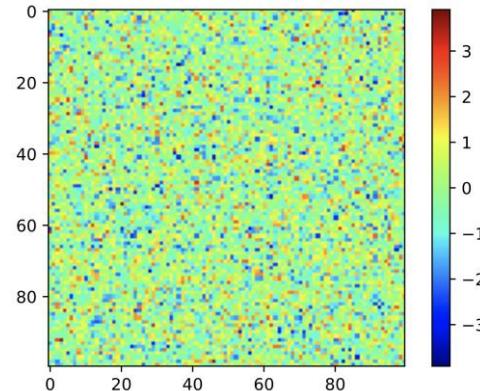
一组RGB视频
(批量大小 x 时间
x 长 x 宽 x 高)

创建NDArray

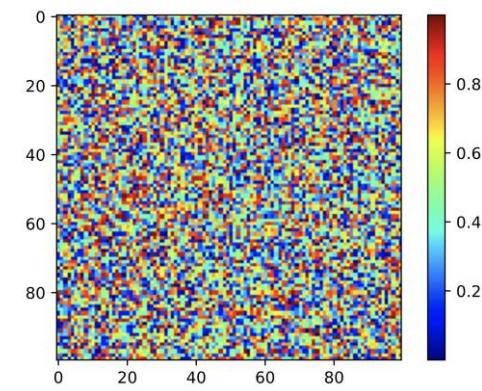
- 按如下标准创建NDArray
 - 形状(shape), e.g. 3×4
 - 元素值, e.g 全部为0, 或者随机值

100 x100 矩阵

正态分布



均匀分布



元素索引

一个元素: [1, 2]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

一行: [1, :]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

一列: [1, :]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

0 1 2 3

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

0 1 2 3

	0	1	2	3
0		1	2	3
1	5	6	7	8
2		9	10	11
3	13	14	15	16

总结

- 深度学习简要概述
- 线性代数
- NDArray

动手学深度学习

2. 概率与统计

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/probability.html>

概要

- **基础概率**

- 随机变量, 条件概率, 贝叶斯法则

- **朴素贝叶斯**

- 多项测试

- **采样**

- 中心极限定理

- 分布 (分类变量, 正态分布, 均匀分布)

基础概率

I USED TO THINK
CORRELATION IMPLIED
CAUSATION.



THEN I TOOK A
STATISTICS CLASS.
NOW I DON'T.



SOUNDS LIKE THE
CLASS HELPED.
WELL, MAYBE.



xkcd.com

D2L.ai

概率

事件空间 X

- 服务器正常工作; 服务器反应迟缓; 服务器停滞
- 用户收入 (如 \$95,000)
- 查询搜索文本 (如“统计教程”)

柯尔莫哥洛夫公理 (Kolmogorov)

$$\Pr(X) \in [0, 1], \Pr(\mathcal{X}) = 1$$

$$\Pr(\cup_i X_i) = \sum_i \Pr(X_i) \text{ if } X_i \cap X_j = \emptyset$$

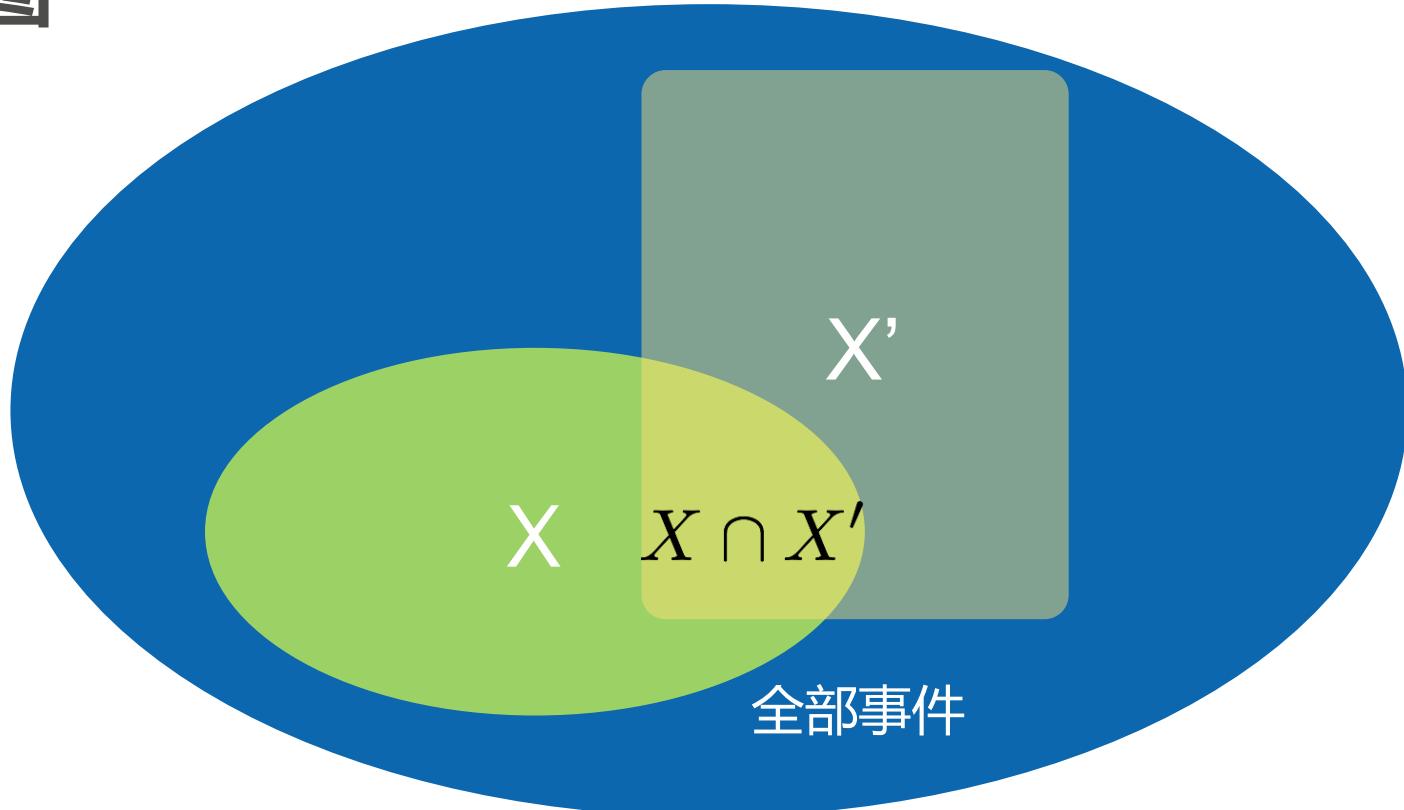
范例

- $P(\text{服务器正常工作}) = 0.999$
- $P(90,000 < \text{收入} < 100,000) = 0.1$

离散

连续

韦恩图



$$\Pr(X \cup X') = \Pr(X) + \Pr(X') - \Pr(X \cap X')$$

独立性与相关性

独立事件

- 两个用户（概率的）登录行为
- 磁盘（概率）以不同的颜色崩溃

$$\Pr(x, y) = \Pr(x) \cdot \Pr(y)$$

相关事件

- 邮件
- 搜索
- 新闻流
- 即时通讯

识别猫和狗本应该很容易.....

10px



识别猫和狗本应该很容易.....

10px

- $P(\text{cat}) = 0.4$

- $P(\text{cat}) = 0.3$

识别猫和狗本应该很容易.....

10px

20px



识别猫和狗本应该很容易.....

10px



20px



40px



识别猫和狗本应该很容易.....

10px



20px



40px



80px



识别猫和狗本应该很容易.....

10px



20px



40px



80px



160px



发生了什么？

不确定性和条件作用

- 不确定性
 - 扔硬币 (正, 反, 边)
 - 彩票
- 条件作用
 - (如果信息相关,) 更多信息使事情更加确定。
 - $p(y|x)$ 而不是 $p(y)$
 - 我们可以建立分类器, 回归量等等。 (本书主要涉及的内容)

贝叶斯法则

- 联合概率

$$\Pr(X, Y) = \Pr(X|Y) \Pr(Y) = \Pr(Y|X) \Pr(X)$$

- 贝叶斯法则

$$\Pr(X|Y) = \frac{\Pr(Y|X) \cdot \Pr(X)}{\Pr(Y)}$$

- 假设检验
- 逆向假设

艾滋病检测实例

- 测试数据
 - 所有人群中，约 0.1% 感染病例（阳性）
 - 测试成功检测到所有感染病例（阳性）
 - 测试误判 1% 健康人群为感染病例（阳性）
- 如果检测为阳性，患艾滋病的概率

$$\begin{aligned}\Pr(a = 1|t) &= \frac{\Pr(t|a = 1) \cdot \Pr(a = 1)}{\Pr(t)} \\ &= \frac{\Pr(t|a = 1) \cdot \Pr(a = 1)}{\Pr(t|a = 1) \cdot \Pr(a = 1) + \Pr(t|a = 0) \cdot \Pr(a = 0)} \\ &= \frac{1 \cdot 0.001}{1 \cdot 0.001 + 0.01 \cdot 0.999} = 0.091\end{aligned}$$



UserFriendly.org

IMPAIRING PRODUCTIVITY SINCE 1997

Copyright © 2007 UserFriendly.org. All Rights Reserved.

朴素贝叶斯

D2L.ai

朴素贝叶斯 - 垃圾邮件 (spam) 过滤器

主要假设

- 在给定文档标签的情况下，单词彼此独立

$$p(w_1, \dots, w_n | \text{spam}) = \prod_{i=1}^n p(w_i | \text{spam})$$

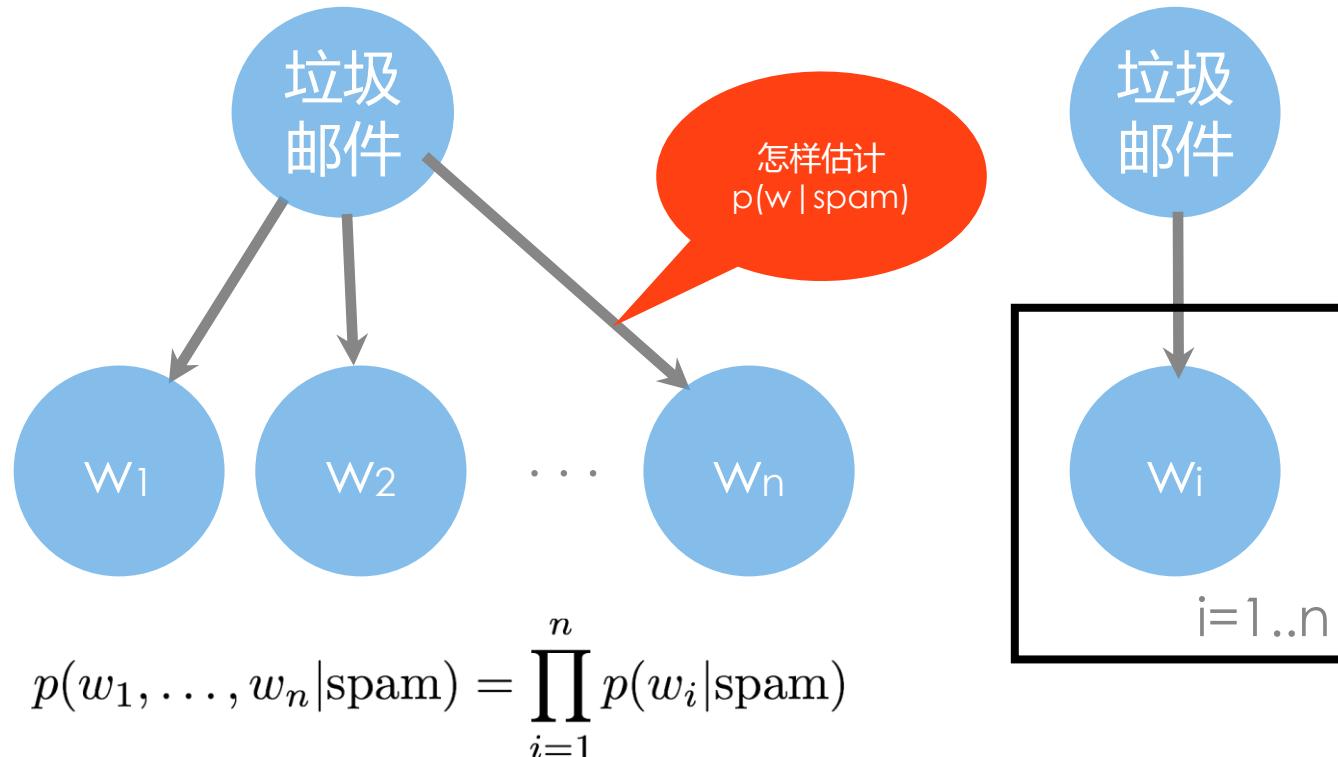
- 通过贝叶斯规则进行垃圾邮件分类

$$p(\text{spam} | w_1, \dots, w_n) \propto p(\text{spam}) \prod_{i=1}^n p(w_i | \text{spam})$$

- **参数估计**

计算垃圾邮件和火腿的垃圾邮件概率和单词分布

图形模型



朴素贝叶斯

- 数据
 - 邮件 (标题, 正文, 元数据)
 - 标注 (垃圾邮件 y/n)
假设用户标记了所有邮件
- 需要估量 $p(y)$, $p(x_i|y)$
 - 对于每一个标注 y , 计算 x_i 分布
 - 计算 y 的分布

简化模型

- 日期
- 时间
- 收信人
- IP
- 发信人
- 编码
-

Delivered-To: alex.smola@gmail.com
Received: by 10.216.47.73 with SMTP id s51cs361171web;
Tue, 3 Jan 2012 14:17:53 -0800 (PST)
Received: by 10.213.17.145 with SMTP id s17mr2519891eba.147.1325629071725;
Tue, 03 Jan 2012 14:17:51 -0800 (PST)
Return-Path: <alex+caf_alex.smola@gmail.com@smola.org>
Received: from mail-ey0-f175.google.com (mail-ey0-f175.google.com [209.85.215.175])
by mx.google.com with ESMTPS id n4si2926423eef.57.2012.01.14.17.51
(version=TLSv1/SSLv3 cipher=OTHER);
Tue, 03 Jan 2012 14:17:51 -0800 (PST)
Received-SPF: neutral (google.com: 209.85.215.175 is neither permitted nor denied by best guess record for domain of alex+caf_alex.smola@gmail.com) client-ip=209.85.215.175;
Authentication-Results: mx.google.com: 209.85.215.175 is neither permitted nor denied by best guess record for domain of alex+caf_alex.smola@gmail.com; smtp.mail=alex+caf_alex.smola@gmail.com; dkim-pass (test mode)
header.i=@googlemail.com
Received: by eaaf1 with SMTP id l1so15092746eaa.6
for <alex.smola@gmail.com>; Tue, 03 Jan 2012 14:17:51 -0800 (PST)
Received: by 10.205.135.18 with SMTP id ie18mr5325064bkc.72.1325629071362;
Tue, 03 Jan 2012 14:17:51 -0800 (PST)
X-Forwarded-To: alex.smola@gmail.com
X-Forwarded-For: alex@smola.org alex.smola@gmail.com
Delivered-To: alex@smola.org
Received: by 10.204.65.198 with SMTP id k6cs206093bk1;
Tue, 3 Jan 2012 14:17:50 -0800 (PST)
Received: by 10.52.88.179 with SMTP id bh19mr1072940vdb.38.1325629068795;
Tue, 03 Jan 2012 14:17:48 -0800 (PST)
Return-Path: <althoff.tim@googlemail.com>
Received: from mail-vx0-f179.google.com (mail-vx0-f179.google.com [209.85.220.179])
by mx.google.com with ESMTPS id dt4si1767074vdb.93.2012.01.03.14.17.48
(version=TLSv1/SSLv3 cipher=OTHER);
Tue, 03 Jan 2012 14:17:48 -0800 (PST)
Received-SPF: pass (google.com: domain of althoff.tim@googlemail.com designates 209.85.220.179 as permitted sender) client-ip=209.85.220.179;
Received: by vcbf13 with SMTP id f13so11295098vcb.10
for <alex@smola.org>; Tue, 03 Jan 2012 14:17:48 -0800 (PST)
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;
d=googlemail.com; s=gamma;
h=mime-version:sender:date:x-google-sender-auth:message-id:subject
:from:to:content-type;
bh=WCbz5xa2c5dpH02xCryDDots993hKwsAVpGrH0w=;
b=WK2B2+ExWnf/gvTk6wUjkU4XeoKnJg3USYTmRARk8dSFjyOQsiHeAP9Yssxp6O
7ngGoT2YqdizsyifvQclAWp1PCjh8AMcnqWkvONMeovVlp2HQooZwxSOCx5ZRgY+7qX
uibbddna4IUQ6uFe16SpLDcptd8OZ3gr7+o=
MIME-Version: 1.0
Received: by 10.220.108.81 with SMTP id e17mr24104004vcp.67.1325629067787;
Tue, 03 Jan 2012 14:17:47 -0800 (PST)
Sender: althoff.tim@googlemail.com
Received: by 10.220.17.129 with HTTP; Tue, 3 Jan 2012 14:17:47 -0800 (PST)
Date: Tue, 3 Jan 2012 14:17:47 -0800
X-Google-Sender-Auth: 6bwI6D17HJlkxOEol38NZzyeHs
Message-ID: <CAFJHDGPBW-Sdz0MdAABiAkyd0ktpewMoDjYGjoGO-WC7osg@mail.gmail.com>
Subject: CS 281B: Advanced Topics in Learning and Decision Making
From: Tim Althoff <althoff@eecs.berkeley.edu>
To: alex@smola.org
Content-Type: multipart/alternative; boundary=f46d043c7af4b07e8d04b5a7113a
--f46d043c7af4b07e8d04b5a7113a
Content-Type: text/plain; charset=ISO-8859-1



朴素贝叶斯 - 垃圾邮件 (spam) 过滤器

- 两类 (spam/ham)
- 特征 (如出现\$\$\$)
- 最简模型
 - 计算垃圾/非垃圾邮件特征的出现次数
 - 计算垃圾/非垃圾邮件

特征概率

$$p(x_i = \text{TRUE}|y) = \frac{n(i, y)}{n(y)} \text{ and } p(y) = \frac{n(y)}{n}$$

垃圾邮件概率

$$p(y|x) \propto \frac{n(y)}{n} \prod_{i:x_i=\text{TRUE}} \frac{n(i, y)}{n(y)} \prod_{i:x_i=\text{FALSE}} \frac{n(y) - n(i, y)}{n(y)}$$

朴素贝叶斯 - 垃圾邮件 (spam) 过滤器



$$p(y|x) \propto \frac{n(y)}{n} \prod_{i:x_i=\text{TRUE}} \frac{n(i,y)}{n(y)} \prod_{i:x_i=\text{FALSE}} \frac{n(y) - n(i,y)}{n(y)}$$

简易模型

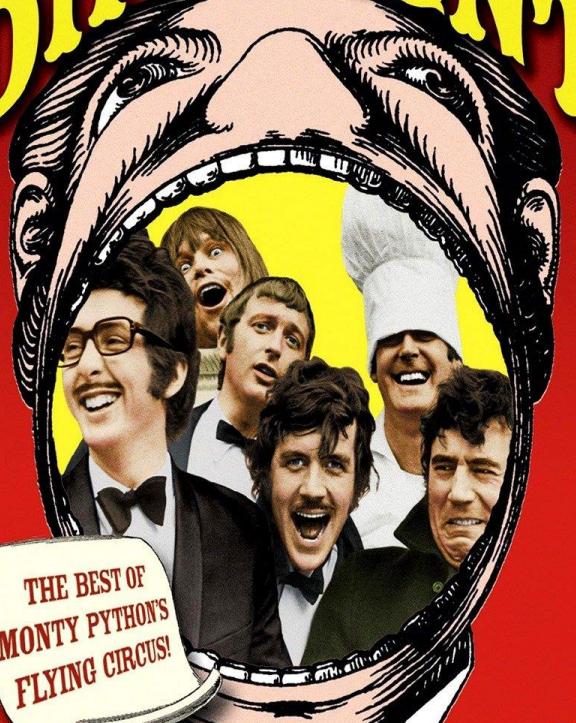
- 对于每一个文档 (x, y) ：
 - 对于每个 y , 标签计数
 - 对于每个 x_i
 - 汇总统计 (x_i, y)
- 对于每个 y , 估量概率 $p(y)$
- 对于每对 (x_i, y) ：
估量概率 $p(x_i|y)$, e.g. Parzen窗估计, 指数族 (高斯, Laplace, 泊松分布, ...), 混合分布
- 对于每个新的变量, 计算

$$p(y|x) \propto p(y) \prod_j p(x_j|y)$$

GRAHAM CHAPMAN JOHN CLEEESE TERRY GILLIAM
ERIC IDLE TERRY JONES MICHAEL PALIN

MONTY PYTHON'S

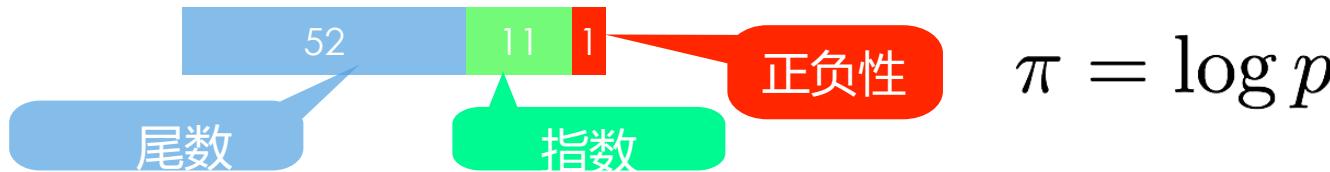
AND NOW FOR SOMETHING COMPLETELY
DIFFERENT



THE BEST OF
MONTY PYTHON'S
FLYING CIRCUS!

对数空间中的概率乘积

- 浮点数字 (FP64)



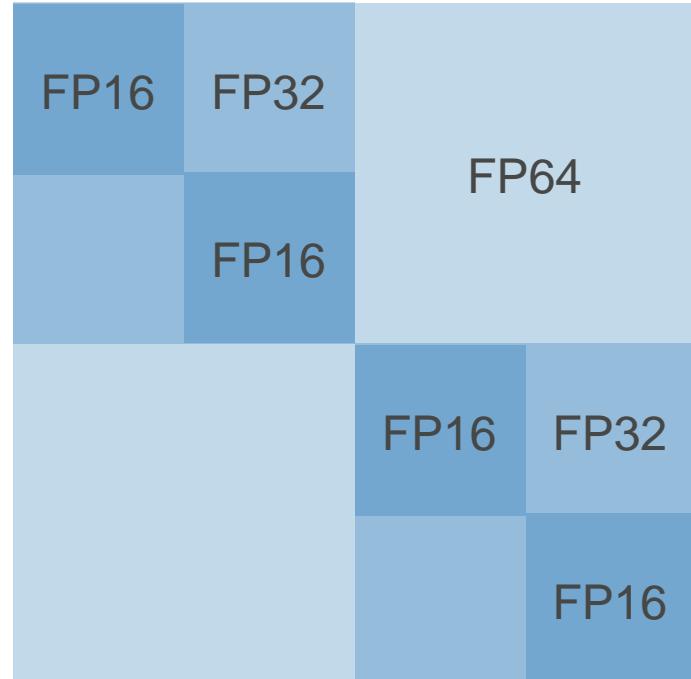
- 概率可能非常小。
特别是许多较小概率相乘。 下溢!
- 数据存储在尾数中，而不是指数

$$\prod_i p_i \rightarrow \sum_i \pi_i \quad \sum_i p_i \rightarrow \max \pi + \log \sum_i \exp [\pi_i - \max \pi]$$

GPUs 的浮点数字

NVIDIA Titan RTX

- 400 GFlops FP64
- 13 TFlops FP32
- 27 TFlops FP16
- 107 TFlops FP16 Tensor cores
- 215 TFlops INT8 Tensor cores
- 430 TFlops INT4 Tensor cores



对于固定带宽，操作次数的两倍
上溢 / 下溢 很危险



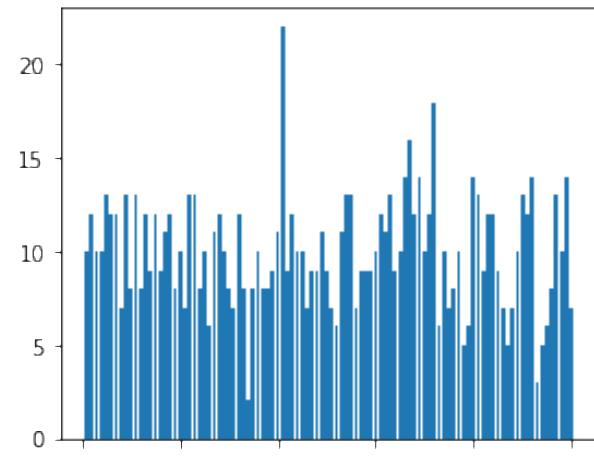
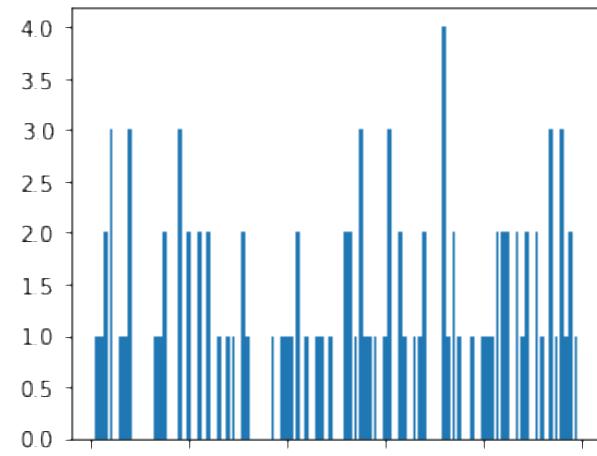
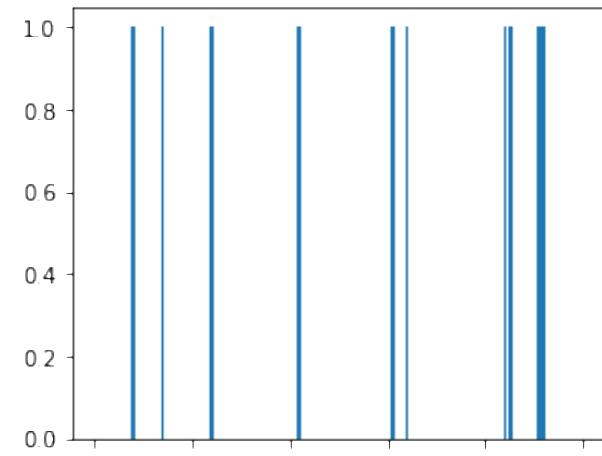
采样

均匀分布

- 在一个区间内恒定，在区间外为零

$$p(x) = \frac{1}{U - L} \text{ if } L \leq x \leq U$$

- 用于初始化参数或负载分配



离散分布

- 离散的一组结果
例如 单词分发, 课堂分发, 垃圾邮件

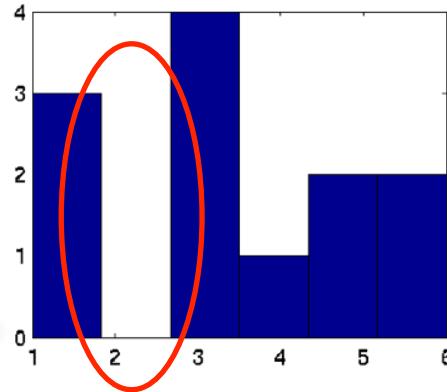
the	and	house	see	is	white	a
0.1	0.05	0.05	0.1	0.2	0.2	0.3

- 采样
 - 暴力法: $O(n)$ 时间
 - 构建堆: $O(\log n)$ 时间, 准备时间 $O(n)$
- 估计 如通过 softmax

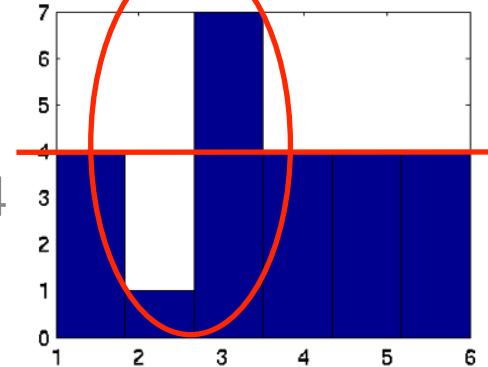
擲骰子



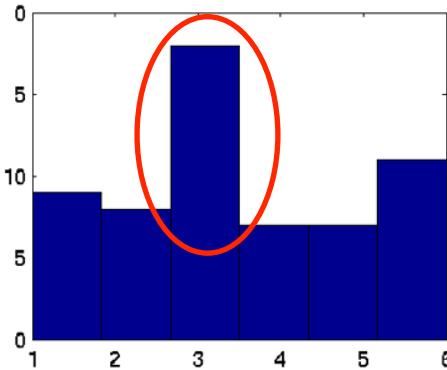
12



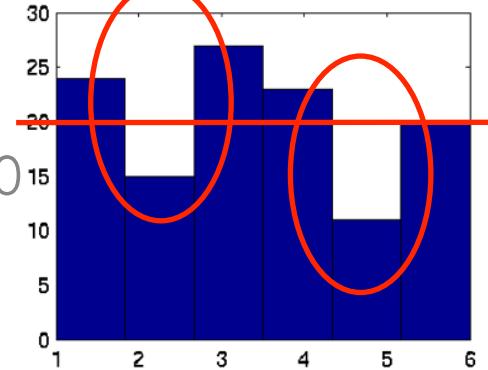
24



60

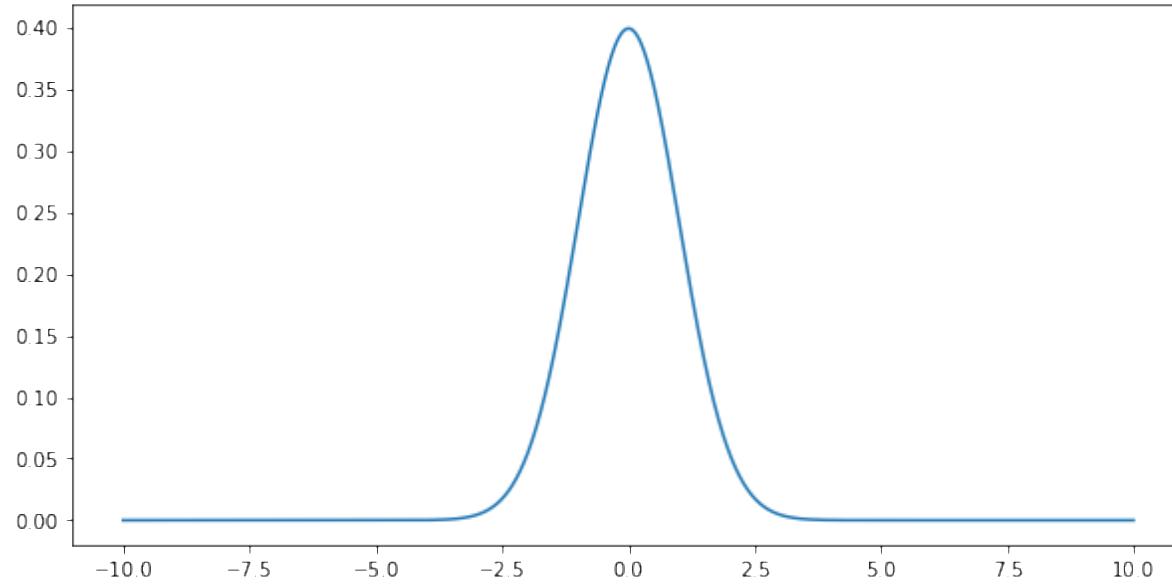


120



正态分布

- 概率密度函数 (PDF) $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$



正态分布

- 概率密度函数 (PDF) $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$

- 平均值

$$\mathbf{E}[x] = \int x dp(x) = \mu$$

- 方差

$$\mathbf{E}[(x-\mu)^2] = \int (x-\mu)^2 dp(x) = \mathbf{E}[x^2] - [\mathbf{E}[x]]^2 = \sigma^2$$

中心极限定理

如下变量趋近高斯分布.

$$z_n := \left[\sum_{i=1}^n \sigma_i^2 \right]^{-\frac{1}{2}} \left[\sum_{i=1}^n x_i - \mu_i \right]$$

实际应用：

独立随机变量的和将收敛到具有联合方差的正态分布。

总结

- **基础概率**

- 随机变量, 条件概率, 贝叶斯法则

- **朴素贝叶斯**

- 多项测试

- **采样**

- 中心极限定理

- 分布 (分类变量, 正态分布, 均匀分布)

动手学深度学习

3. 导数、反向传播和复杂度

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/arrays.html>

概要

- 矩阵微积分
- 链式法则
- 自动微分法
- 反向传播

矩阵微积分

Calculus

$\frac{d}{dx} \left[\frac{f(x)}{g(x)} \right] = \frac{g(x)f'(x) - f(x)g'(x)}{g(x)^2}$

$F = mg = ma = m \frac{d^2x}{dt^2} = -kx$

$\frac{dA}{dt} = \frac{dB}{dt} = \frac{dC}{dt} = \frac{dD}{dt} = (c_1)AB - (c_2)CD$

$\frac{du}{dx} = \frac{du}{dy} = \frac{dy}{dx}$

$y = mx + b$

Gottfried Wilhelm Leibniz

Maria Gaetana Agnesi

$(\ln x)' = \frac{1}{x}$

$\int \frac{1}{x} dx = \ln|x| + C$

$f(x) = x^2$

$\int \sin x dx = -\cos x + C$

$\int_a^b f'(x) dx = f(b) - f(a)$

$m \frac{d^2x}{dt^2} = -kx$

$\frac{df(x)}{dz}$

$x^2 - 3x - 4 = 0$

$4x^2 - 3x - 1 = 0$

$\int f(x) dx$

$\frac{dA}{dt} = \frac{dB}{dt} = -\frac{dC}{dt} = -\frac{dD}{dt} = (d_1)T^{\frac{1}{2}}AB - (d_2)T^{\frac{1}{2}}CD$

$x^2 = A$

$\frac{dT}{dt} = (c_3) \frac{dA}{dt} - (c_4)(T_0 - T)$

$\left[x + \frac{b}{2a} \right]^2 = \frac{b^2 - 4ac}{4a^2}$

$x + \frac{b}{2a} = \frac{\sqrt{b^2 - 4ac}}{2a}$ or $x + \frac{b}{2a} = -\frac{\sqrt{b^2 - 4ac}}{2a}$

$\frac{d}{dx} \int_a^x f(t) dt = f(x)$

v_z

$(x+h, f(x+h))$

$m \frac{d^2x}{dt^2} = -kx - f \frac{dx}{dt} + A \sin(\omega t)$

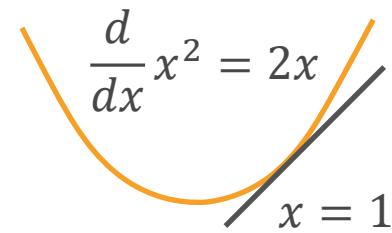
$y' = v$, and $v' = -kv - fv + A \sin(\omega t)$

$f(x-h) - f(x)$

标量求导回顾

y	a	x^n	$\exp(x)$	$\log(x)$	$\sin(x)$
$\frac{dy}{dx}$	0	nx^{n-1}	$\exp(x)$	$\frac{1}{x}$	$\cos(x)$

导数是切线的斜率



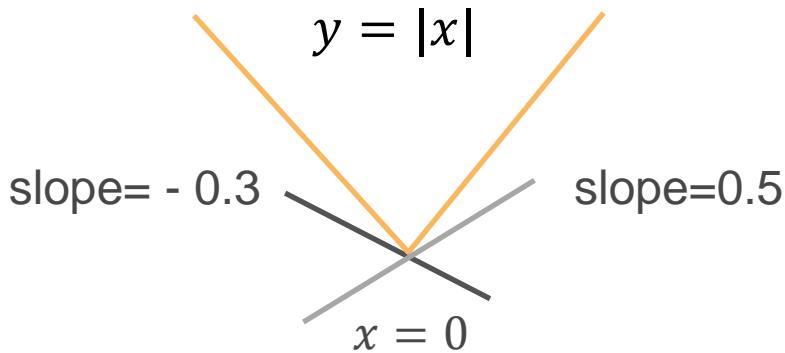
y	$u + v$	uv	$y = f(u), u = g(x)$	
$\frac{dy}{dx}$	$\frac{du}{dx} + \frac{dv}{dx}$	$\frac{du}{dx}v + \frac{dv}{dx}u$	$\frac{dy}{du}\frac{du}{dx}$	

切线的斜率为2

次导数

- 不可求导情况下的导数

Example 1:



$$\frac{\partial|x|}{\partial x} = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ a & \text{if } x = 0, a \in [-1, 1] \end{cases}$$

Example 2:

$$\frac{\partial}{\partial x} \max(x, 0) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ a & \text{if } x = 0, a \in [0, 1] \end{cases}$$

梯度

- 矢量求导推广

标量 矢量

	x	\mathbf{x}
标量	y	$\frac{\partial y}{\partial x}$
矢量	\mathbf{y}	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$

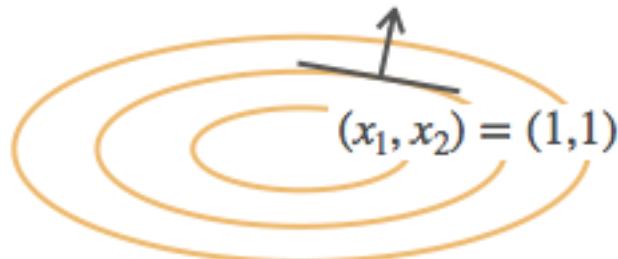
$\partial y / \partial \mathbf{x}$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \frac{\partial y}{\partial \mathbf{x}} = \left[\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \dots, \frac{\partial y}{\partial x_n} \right]$$

x	y	\mathbf{x}
$\frac{\partial y}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$
	\mathbf{y}	

$$\frac{\partial}{\partial \mathbf{x}} x_1^2 + 2x_2^2 = [2x_1, 4x_2]$$

Direction (2, 4), perpendicular to
the contour lines



例子

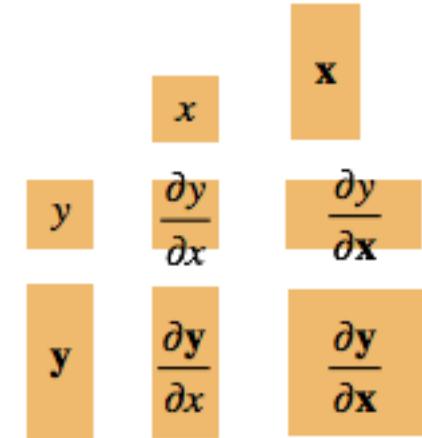
y	a	au	$sum(\mathbf{x})$	$\ \mathbf{x} \ ^2$
$\frac{\partial y}{\partial \mathbf{x}}$	$\mathbf{0}^T$	$a \frac{\partial u}{\partial \mathbf{x}}$	$\mathbf{1}^T$	$2\mathbf{x}^T$

y	$u + v$	uv	$\langle \mathbf{u}, \mathbf{v} \rangle$
$\frac{\partial y}{\partial \mathbf{x}}$	$\frac{\partial u}{\partial \mathbf{x}} + \frac{\partial v}{\partial \mathbf{x}}$	$\frac{\partial u}{\partial \mathbf{x}}v + \frac{\partial v}{\partial \mathbf{x}}u$	$\mathbf{u}^T \frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \mathbf{v}^T \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$

$\partial \mathbf{y} / \partial x$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \vdots \\ \frac{\partial y_m}{\partial x} \end{bmatrix}$$

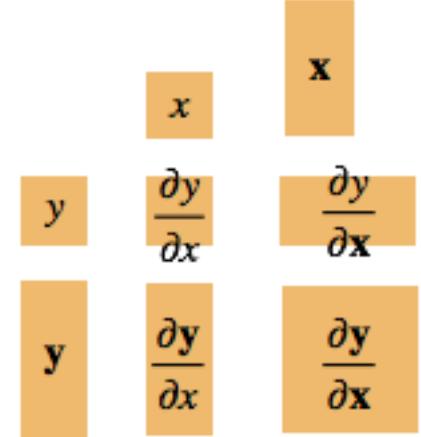


$\partial \mathbf{y} / \partial x$ 叫做分子布局 (numerator-layout 或 Jacobian formulation) , 是行矢量

$\partial \mathbf{y} / \partial \mathbf{x}$ 叫做分母布局 (denominator-layout 或 Hessian formulation) , 是列矢量

$\partial \mathbf{y} / \partial \mathbf{x}$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \quad \mathbf{x} \in \mathbb{R}^n, \quad \mathbf{y} \in \mathbb{R}^m, \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathbb{R}^{m \times n}$$



$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial \mathbf{x}} \\ \frac{\partial y_2}{\partial \mathbf{x}} \\ \vdots \\ \frac{\partial y_m}{\partial \mathbf{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1}, \frac{\partial y_1}{\partial x_2}, \dots, \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1}, \frac{\partial y_2}{\partial x_2}, \dots, \frac{\partial y_2}{\partial x_n} \\ \vdots \\ \frac{\partial y_m}{\partial x_1}, \frac{\partial y_m}{\partial x_2}, \dots, \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

例子

y	a	x	Ax	$x^T A$
$\frac{\partial y}{\partial x}$	0	I	A	A^T

$$x \in \mathbb{R}^n, y \in \mathbb{R}^m, \frac{\partial y}{\partial x} \in \mathbb{R}^{m \times n}$$

a, a 和 A 不是关于 x 的函数

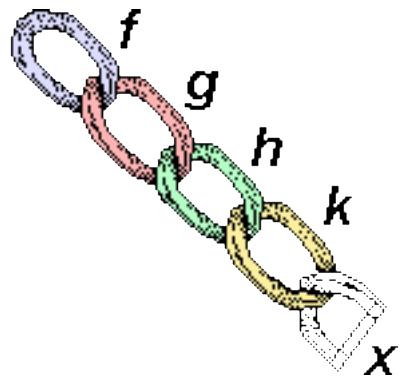
0 和 I 为矩阵

y	$a\mathbf{u}$	$A\mathbf{u}$	$\mathbf{u} + \mathbf{v}$
$\frac{\partial y}{\partial x}$	$a \frac{\partial \mathbf{u}}{\partial x}$	$A \frac{\partial \mathbf{u}}{\partial x}$	$\frac{\partial \mathbf{u}}{\partial x} + \frac{\partial \mathbf{v}}{\partial x}$

推广到矩阵

	标量	矢量	矩阵
标量	x $(1,)$	\mathbf{x} $(n, 1)$	\mathbf{X} (n, k)
矢量	y $(1,)$	$\frac{\partial y}{\partial \mathbf{x}}$ $(1,)$	$\frac{\partial y}{\partial \mathbf{X}}$ (k, n)
矩阵	\mathbf{y} $(m, 1)$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ $(m, 1)$	$\frac{\partial \mathbf{y}}{\partial \mathbf{X}}$ (m, k, n)
	\mathbf{Y} (m, l)	$\frac{\partial \mathbf{Y}}{\partial \mathbf{x}}$ (m, l)	$\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ (m, l, n)

链式法则



链式法则

- 链式法则 – 标量:

$$y = f(u), u = g(x) \quad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \frac{\partial u}{\partial x}$$

- 链式法则 – 矢量:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

$(1, n)$ $(1,)$ $(1, n)$

$(1, n)$ $(1, k)$ (k, n)

(m, n) (m, k) (k, n)

例1

假设 $\mathbf{x}, \mathbf{w} \in \mathbb{R}^n, y \in \mathbb{R}$ $z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$

计算 $\frac{\partial z}{\partial \mathbf{w}}$

例1

假设 $\mathbf{x}, \mathbf{w} \in \mathbb{R}^n, y \in \mathbb{R}$ $z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$

计算 $\frac{\partial z}{\partial \mathbf{w}}$

$$\text{分解 } a = \langle \mathbf{x}, \mathbf{w} \rangle$$

$$b = a - y$$

$$z = b^2$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

$$\begin{aligned}\frac{\partial z}{\partial \mathbf{w}} &= \frac{\partial z}{\partial b} \frac{\partial b}{\partial a} \frac{\partial a}{\partial \mathbf{w}} \\ &= \frac{\partial b^2}{\partial b} \frac{\partial a - y}{\partial a} \frac{\partial \langle \mathbf{x}, \mathbf{w} \rangle}{\partial \mathbf{w}} \\ &= 2b \cdot 1 \cdot \mathbf{x}^T \\ &= 2(\langle \mathbf{x}, \mathbf{w} \rangle - y) \mathbf{x}^T\end{aligned}$$

例2

假设 $\mathbf{X} \in \mathbb{R}^{m \times n}$, $\mathbf{w} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^m$ $z = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$

计算 $\frac{\partial z}{\partial \mathbf{w}}$

例2

假设 $\mathbf{X} \in \mathbb{R}^{m \times n}$, $\mathbf{w} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^m$ $z = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$

计算 $\frac{\partial z}{\partial \mathbf{w}}$

$$\text{分解 } \mathbf{a} = \mathbf{X}\mathbf{w}$$

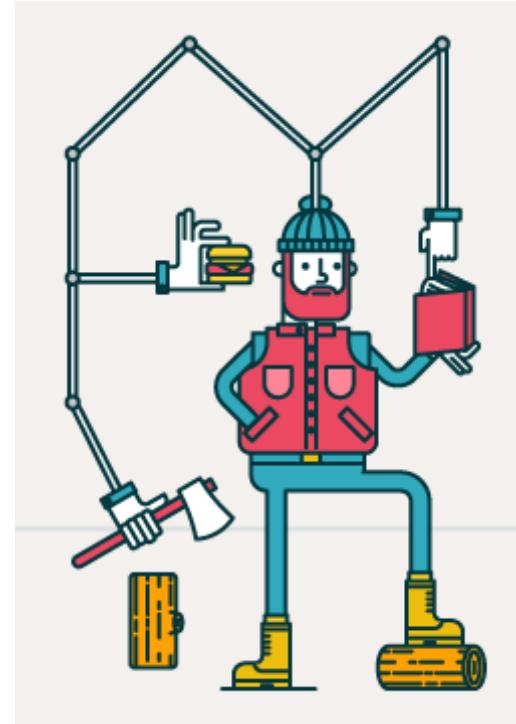
$$\mathbf{b} = \mathbf{a} - \mathbf{y}$$

$$z = \|\mathbf{b}\|^2$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

$$\begin{aligned}\frac{\partial z}{\partial \mathbf{w}} &= \frac{\partial z}{\partial \mathbf{b}} \frac{\partial \mathbf{b}}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \\ &= \frac{\partial \|\mathbf{b}\|^2}{\partial \mathbf{b}} \frac{\partial \mathbf{a} - \mathbf{y}}{\partial \mathbf{a}} \frac{\partial \mathbf{X}\mathbf{w}}{\partial \mathbf{w}} \\ &= 2\mathbf{b}^T \times \mathbf{I} \times \mathbf{X} \\ &= 2(\mathbf{X}\mathbf{w} - \mathbf{y})^T \mathbf{X}\end{aligned}$$

自动微分法



自动微分 (AD)

- 自动微分 (AD) 将符号微分法应用于最基本的算子，然后代入数值，应用于整个函数
- 其它常见微分法
 - 符号微分法

```
In[1]:= D[4 x3 + x2 + 3, x]
```

```
Out[1]= 2 x + 12 x2
```

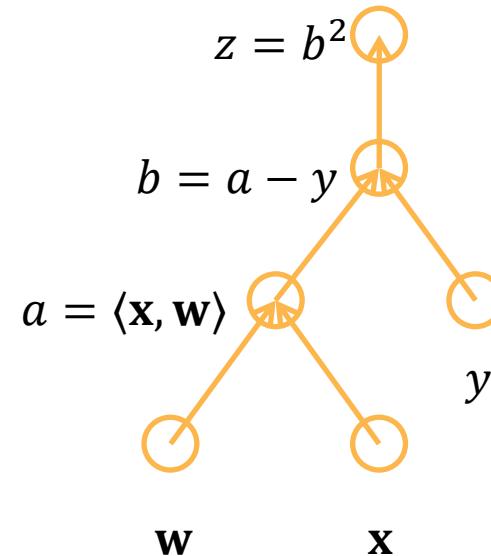
- 数值微分法

$$\frac{\partial f(x)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

计算图

- 分解成最基本的方程
- 构造有向无环图来表示运算

$$\text{假设 } z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$$



计算图

- 分解成最基本的方程
- 构造有向无环图来表示运算
- 显性构造
Tensorflow/Theano/MXNet

```
from mxnet import sym
```

```
a = sym.var()
```

```
b = sym.var()
```

```
c = 2 * a + b
```

稍后将数据绑定到a和b中

计算图

- 分解成最基本的方程
- 构造有向无环图来表示运算
- 显性构造
Tensorflow/Theano/MXNet
- 隐性构造
PyTorch/MXNet

```
from mxnet import autograd, nd
```

```
with autograd.record():
```

```
a = nd.ones((2,1))
```

```
b = nd.ones((2,1))
```

```
c = 2 * a + b
```

两种模式

- 通过链式法则 $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_n} \frac{\partial u_n}{\partial u_{n-1}} \cdots \frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x}$

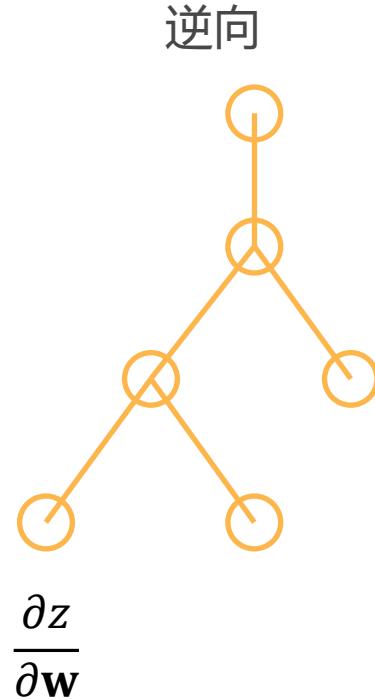
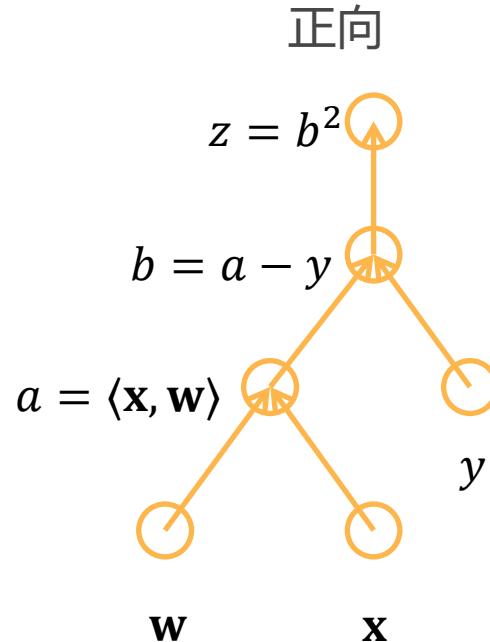
- 正向传播 $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_n} \left(\frac{\partial u_n}{\partial u_{n-1}} \left(\cdots \left(\frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x} \right) \right) \right)$

- 反向传播

$$\frac{\partial y}{\partial x} = \left(\left(\left(\frac{\partial y}{\partial u_n} \frac{\partial u_n}{\partial u_{n-1}} \right) \cdots \right) \frac{\partial u_2}{\partial u_1} \right) \frac{\partial u_1}{\partial x}$$

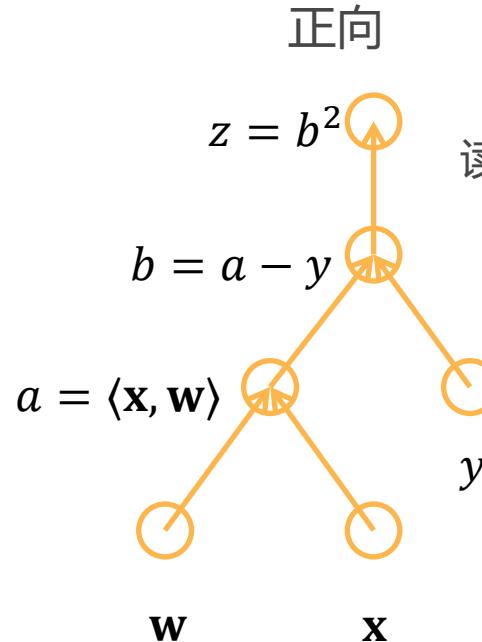
反向传播

假设 $z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$

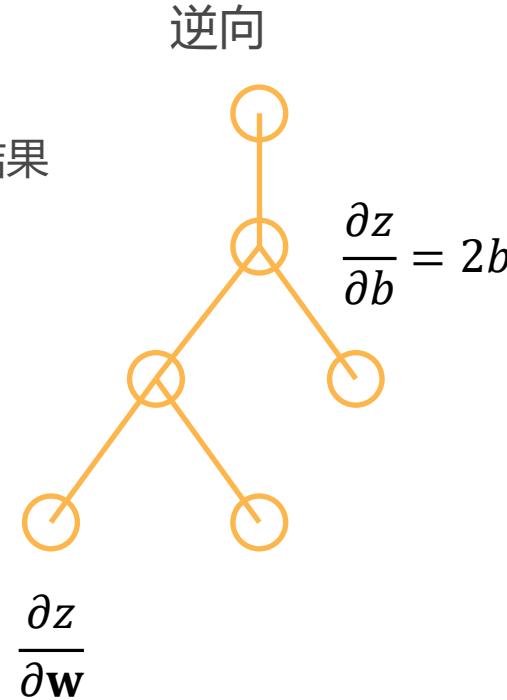


反向传播

假设 $z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$

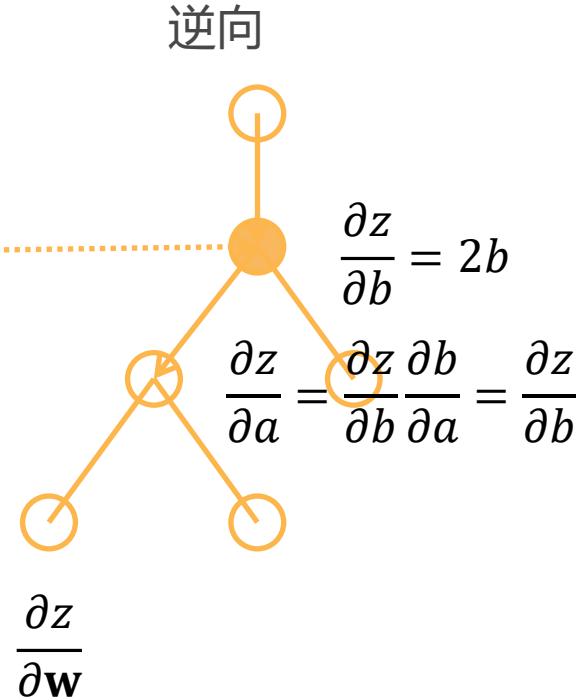
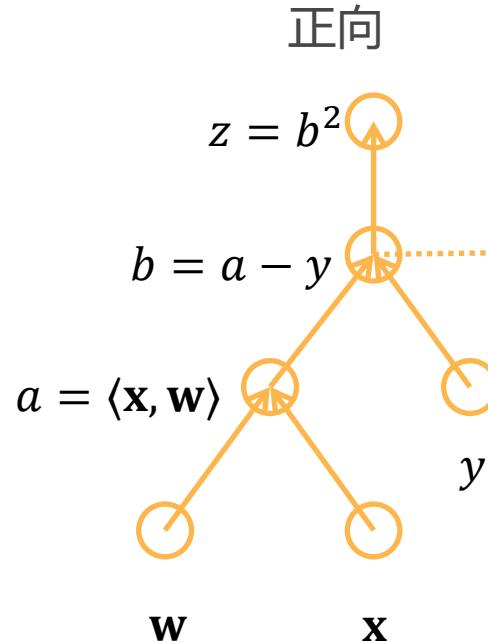


读取上一步结果



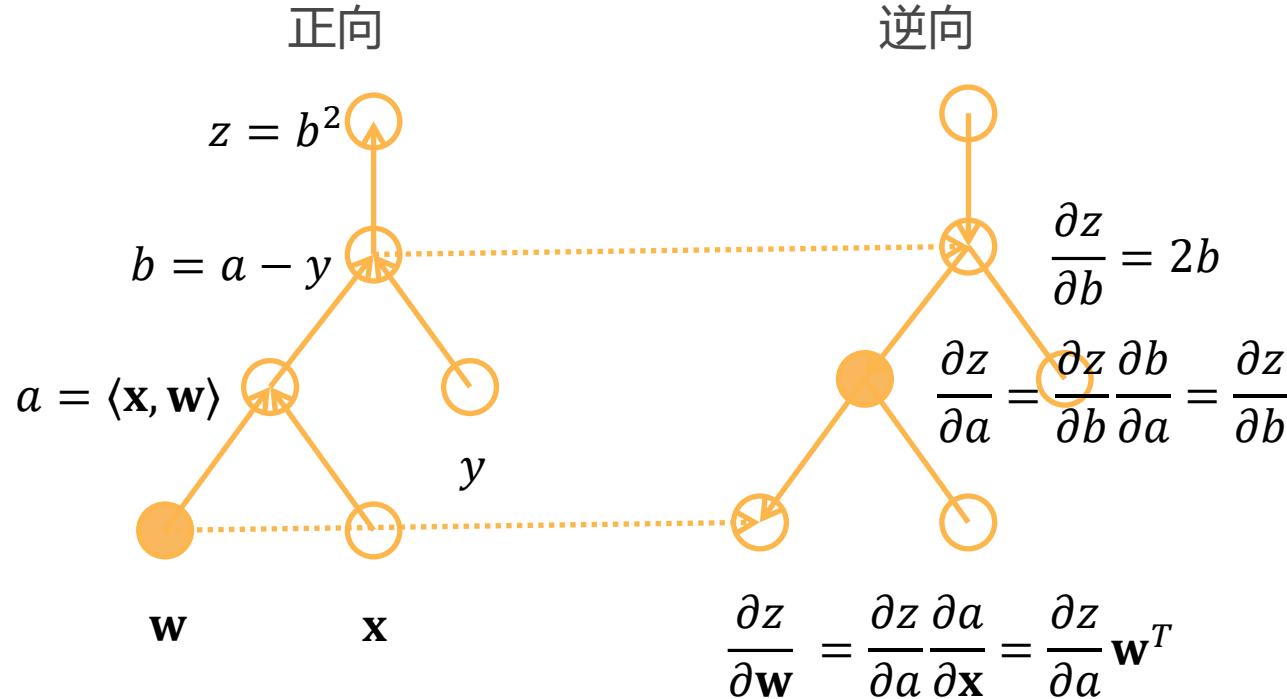
反向传播

假设 $z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$



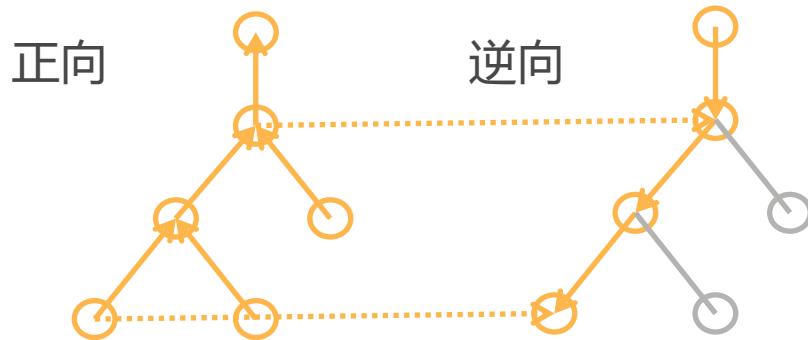
反向传播

假设 $z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$



反向传播 总结

- 创建一个计算图
- 正向：计算有向无环图，储存中间值
- 反向：逆向计算有向无环图
 - 减少不需要的图



复杂度

$O(n)$, n 为计算次数

- 反向传播复杂度：
 - 时间复杂度： $O(n)$, 计算所有导数，基本上与正向复杂度一致
 - 内存复杂度： $O(n)$, 需要储存所有正向计算的中间值
- 对比正向传播：
 - 时间复杂度： $O(n)$, 计算 k 个变量的导数为 $O(n*k)$
 - 内存复杂度： $O(1)$

[拓展] 再具体化 (re-materialization)

- 内存是逆向传播的瓶颈
 - 随着层数和批量大小线性增长
 - 有限 GPU 内存 (最多32GB)
- 用算力换内存
 - 只保存一部分中间计算值
 - 当需要时重新计算未保存中间值

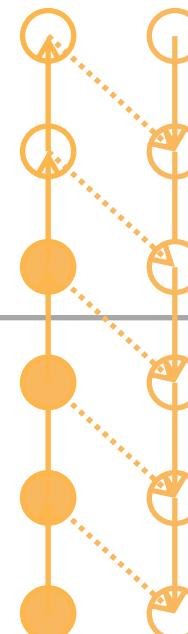
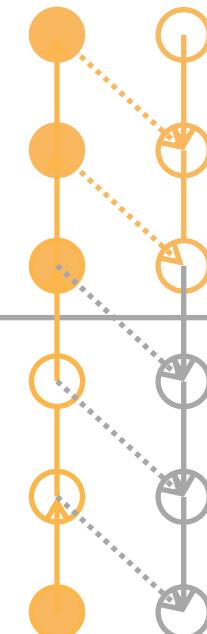
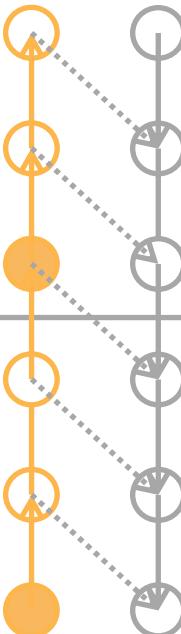
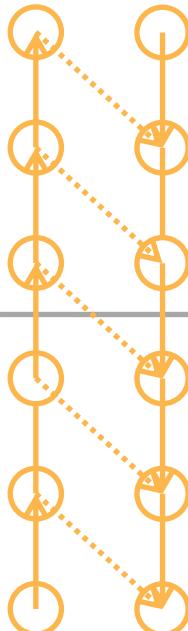
再具体化 (re-materialization)

正向 逆向

只保留每部分的
头部结果

重新计算第2部分
未保留的中间值

重新计算第1部分
未保留的中间值



复杂度

- 多一步正向传播
- 假设共有 m 部分, 则有 $O(m)$ 头部结果, 每个部分需内存 $O(n/m)$
 - 令 $m = \sqrt{n}$, 则内存复杂度为 $O(\sqrt{n})$
- 运用到深度学习网络
 - 只丢弃简单层, 如激活函数层, 常见 $<30\%$
 - 训练 10 倍大的网络, 或者 10 倍大的批量大小

总结

- 矩阵微积分
- 链式法则
- 自动微分法
- 反向传播

动手学深度学习

4. 线性方法、基础优化和 softmax 回归

中文教材: zh.d2l.ai

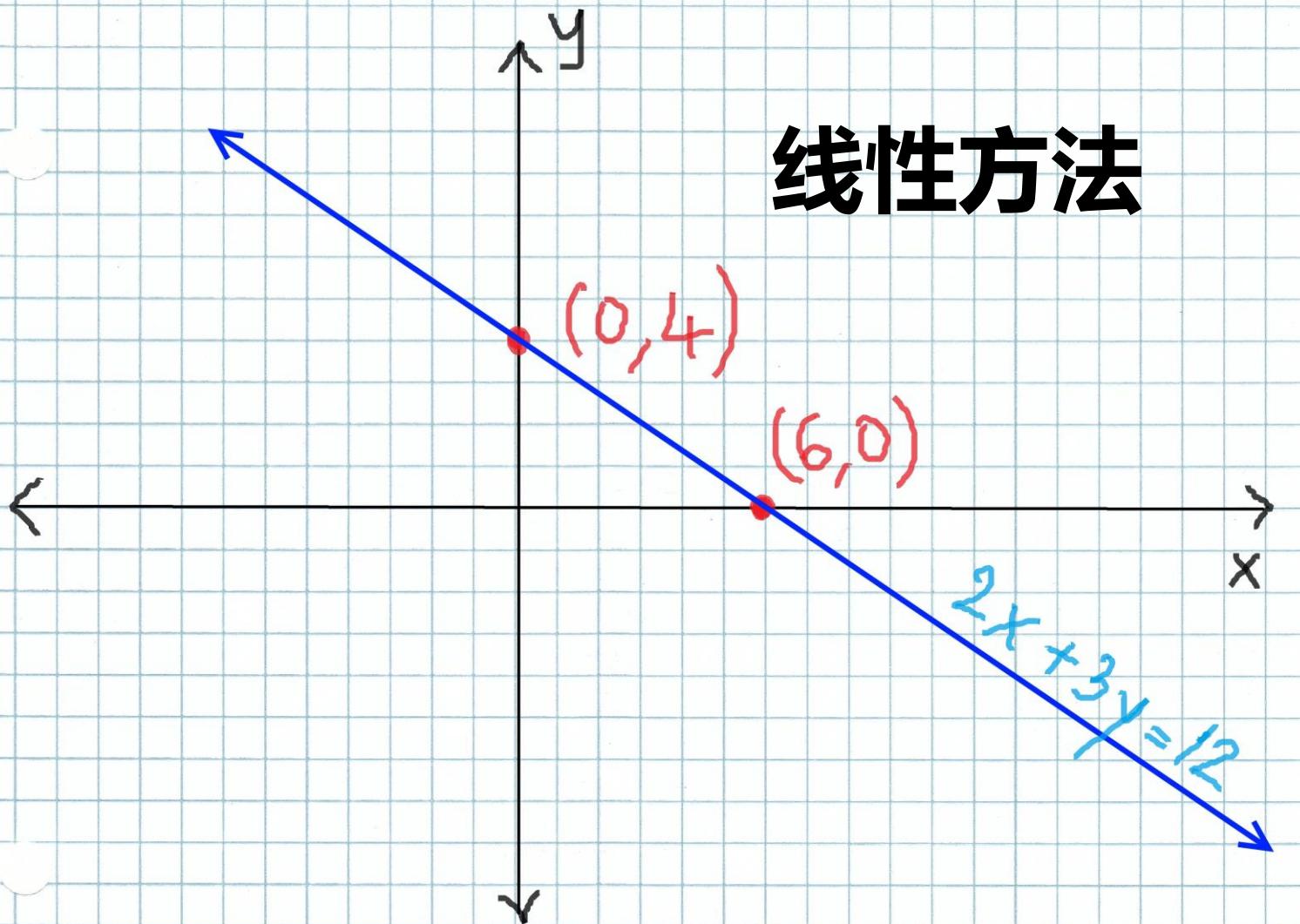
英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/linear.html>

概要

- 线性方法
 - 神经科学的灵感
- 基础优化
- softmax 回归
 - 回归与分类
 - Kaggle上的分类任务

线性方法



房价预测 101

- 挑选一间别墅，逛一逛，了解卖点
- 估计竞拍价

代理商的
市场价格

\$5,498,000 | 7 | 5 | 4,865 Sq. Ft.
Price | Beds | Baths | \$1130 / Sq. Ft.
Redfin Estimate: \$5,390,037 On Redfin: 15 days

估计
销售价格



Virtual Tour

- Branded Virtual Tour
- [Virtual Tour \(External Link\)](#)

Parking Information

- Garage (Minimum): 2
- Garage (Maximum): 2
- Parking Description: Attached Garage, On Street
- Garage Spaces: 2

Multi-Unit Information

- # of Stories: 2

School Information

- Elementary School: El Carmelo El
- Elementary School District: Palo A
- Middle School: Jane Lathrop Stan
- High School: Palo Alto High
- High School District: Palo Alto Un

Interior Features

Bedroom Information

- # of Bedrooms (Minimum): 7
- # of Bedrooms (Maximum): 7

- Kitchen Description: Countertop Dishwasher, Garbage Disposal, Ho Island with Sink, Microwave, Over

房价预测

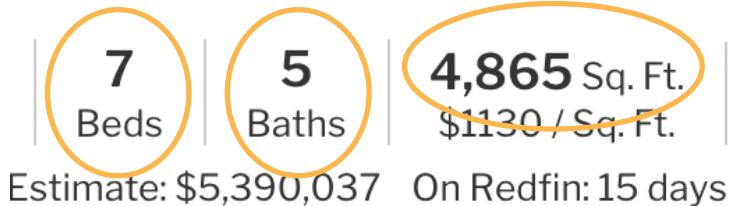
非常重要，因为这是真钱.....

\$100K+ 差距



Redfin 高估房价，B 相信它

一个简易模型



- 假设 1

影响房价的关键因素：

卧室数目，卫浴数目和房子大小，分别用 x_1, x_2, x_3 表示

- 假设 2

销售价格是关键因素的加权总和：

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

权重和偏差稍后确定。

线性方法

- 给予 n 维输入, $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$
- 线性方法有 n 个权重和偏差:

$$\mathbf{w} = [w_1, w_2, \dots, w_n]^T, b$$

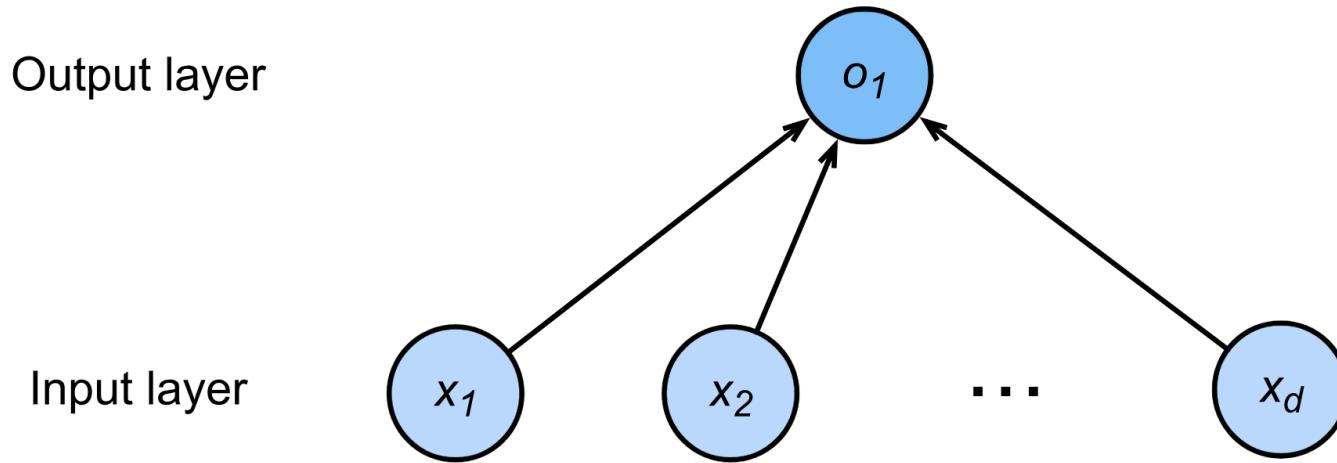
- 输出是输入的加权总和:

$$y = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$$

矢量化版本:

$$y = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

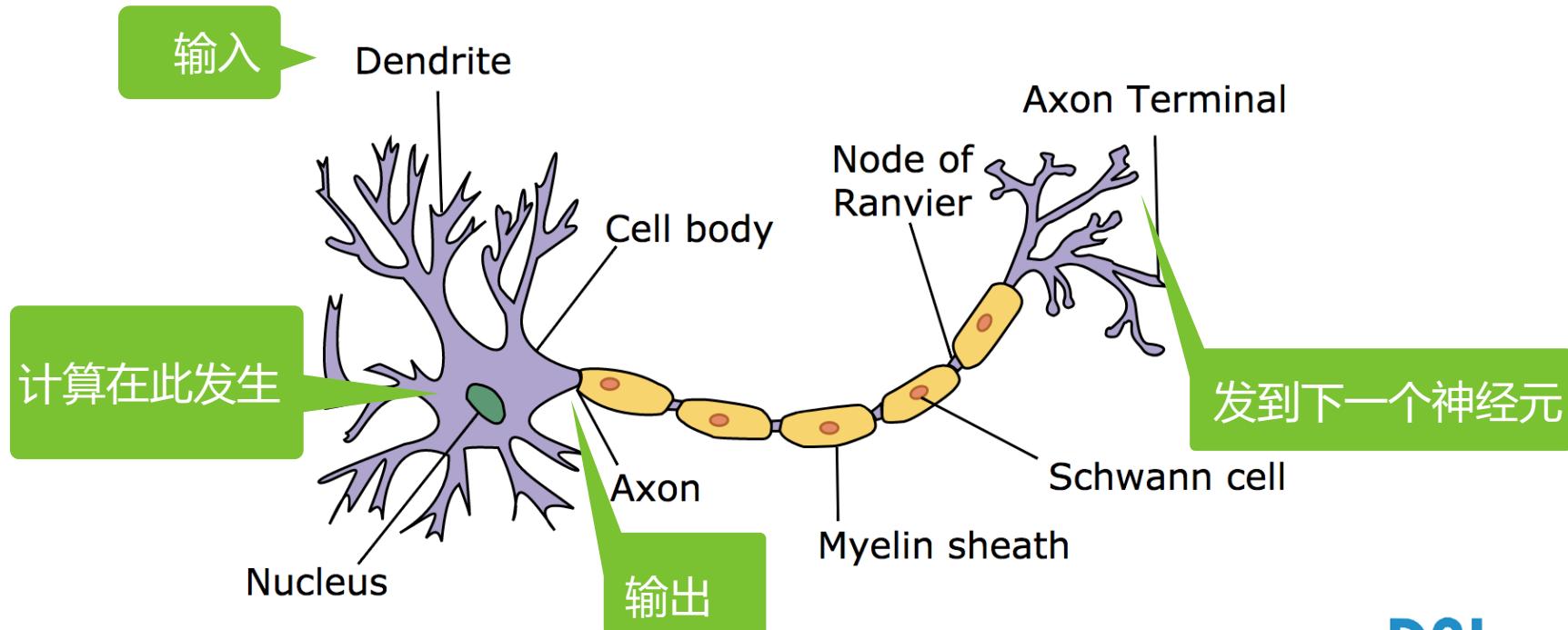
线性方法是一个单层神经网络



我们可以堆叠多个层来获得深层神经网络。

神经科学的灵感

真实的神经元



测量估计质量

- 比较真实值与估计值 (实际销售价格与估计的房价)
- 以 y 作为真实值, \hat{y} 作为估计值, 我们可以比较损失

平方损失:

$$\ell(y, \hat{y}) = (\hat{y} - y)^2$$

训练数据集

- 收集多个数据点以训练参数 (如 在过去6个月内出售的房屋)
- 这个叫做训练数据集
- 训练数据集 越大越好
- 假设有 n 个房屋

$$\mathbf{X} = [\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_n]^T \quad \mathbf{y} = [y_0, y_1, \dots, y_n]^T$$

学习参数

- 训练损失

$$\ell(\mathbf{X}, \mathbf{y}, \mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^n (y_i - \langle \mathbf{x}_i, \mathbf{w} \rangle - b)^2 = \frac{1}{n} \| \mathbf{y} - \mathbf{X}\mathbf{w} - b \|_2^2$$

- 最小化学习参数的损失

$$\mathbf{w}^*, \mathbf{b}^* = \underset{\mathbf{w}, b}{\operatorname{argmin}} \ell(\mathbf{X}, \mathbf{y}, \mathbf{w}, b)$$

封闭解

- 将偏差添加到权重中

$$\mathbf{X} \leftarrow [\mathbf{X}, \mathbf{1}] \mathbf{w} \leftarrow \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}$$

$$\ell(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{1}{n} \| \mathbf{y} - \mathbf{X}\mathbf{w} \|^2 \quad \frac{\partial}{\partial \mathbf{w}} \ell(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{2}{n} (\mathbf{y} - \mathbf{X}\mathbf{w})^T \mathbf{X}$$

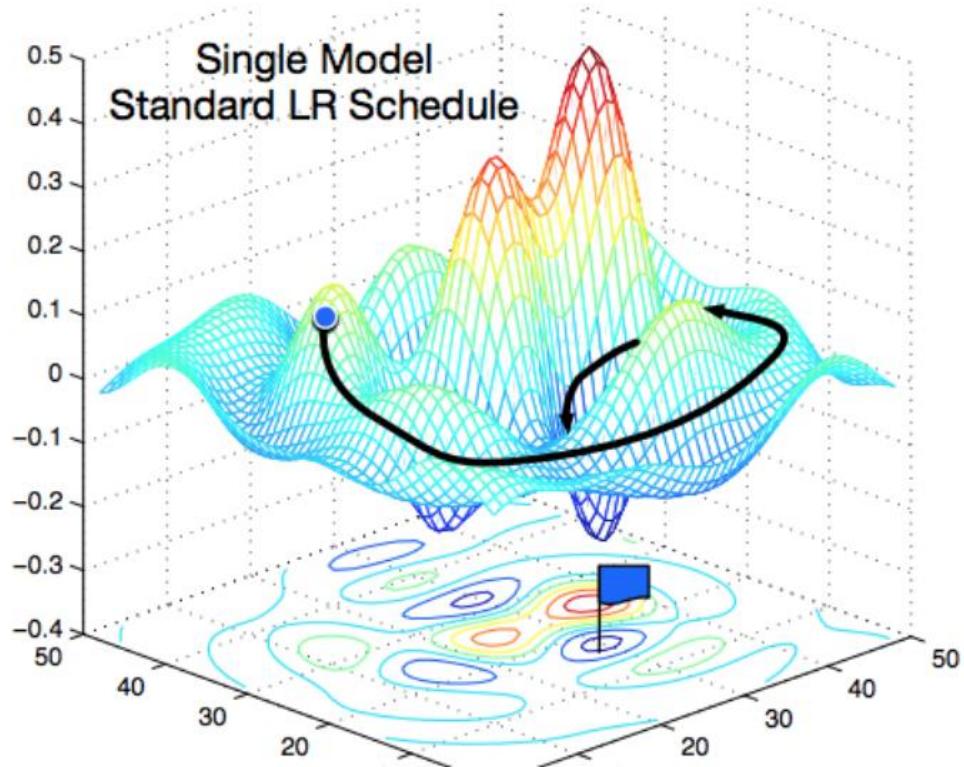
- 损失是凸性的，因此最优解满足：

$$\frac{\partial}{\partial \mathbf{w}} \ell(\mathbf{X}, \mathbf{y}, \mathbf{w}) = 0$$

$$\Leftrightarrow \frac{2}{n} (\mathbf{y} - \mathbf{X}\mathbf{w})^T \mathbf{X} = 0$$

$$\Leftrightarrow \mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X} \mathbf{y}$$

基础优化

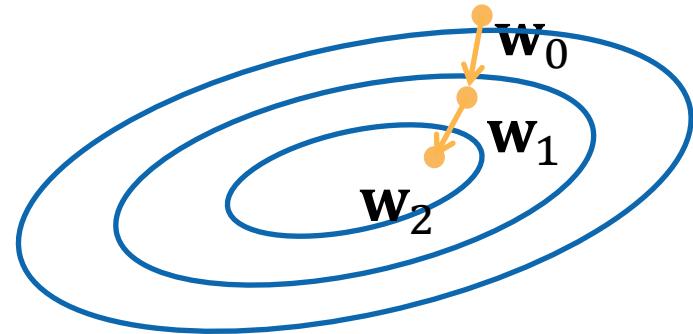


梯度下降

- 选择一个起点 w_0
- 重复更新权重 $t = 1, 2, 3$

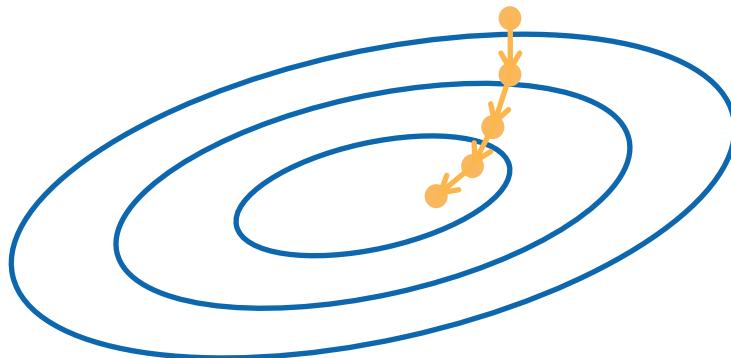
$$w_t = w_{t-1} - \eta \frac{\partial \ell}{\partial w_{t-1}}$$

- 梯度： 更新权重的方向
- 学习率： 一个超参数指定 每梯度的步长

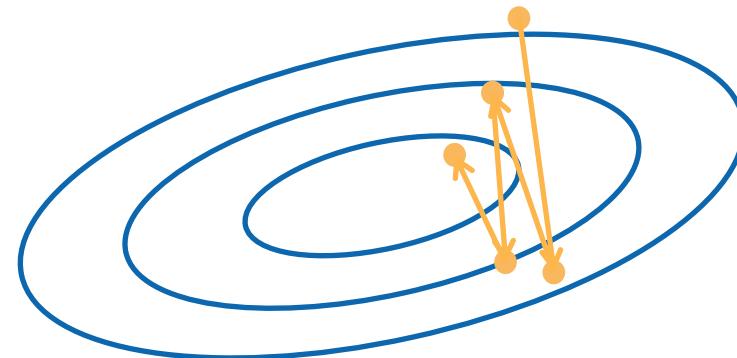


选择学习率

不要太小



不要太大



小批量随机梯度下降 (SGD)

- 计算整个训练数据的梯度太昂贵了
 - DNN模型需要几分钟到几小时
- 解决方案： 随机抽样 b 个样本 i_1, i_2, \dots, i_b 来估算损失

$$\frac{1}{b} \sum_{i \in I_b} \ell(\mathbf{x}_i, y_i, \mathbf{w})$$

- b 是批量大小，另一个重要的超参数

选择批量值

不要太小

批量值太小，难以充分
利用计算资源

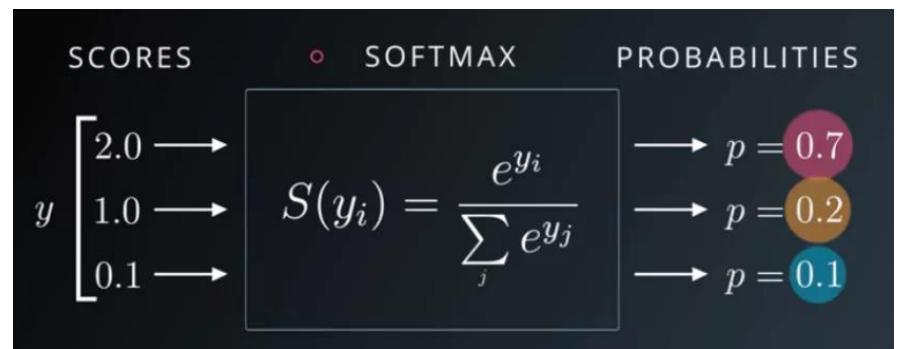
不要太大

批量值太大，浪费计算资
源；
例如当 x_i 都相同时

总结

- 问题：估计一个真正的值
- 模型： $y = \langle \mathbf{w}, \mathbf{x} \rangle + b$
- 损失：平方损失 $\ell(y, \hat{y}) = (\hat{y} - y)^2$
- 小批量随机梯度 (mini-batch SGD) 学习
 - 选择一个起点
 - 重复
 - 计算梯度
 - 更新参数

softmax 回归



回归与分类

- 回归估计连续值
- 分类预测离散类别

MNIST：对手写数字进行分类
(10类)

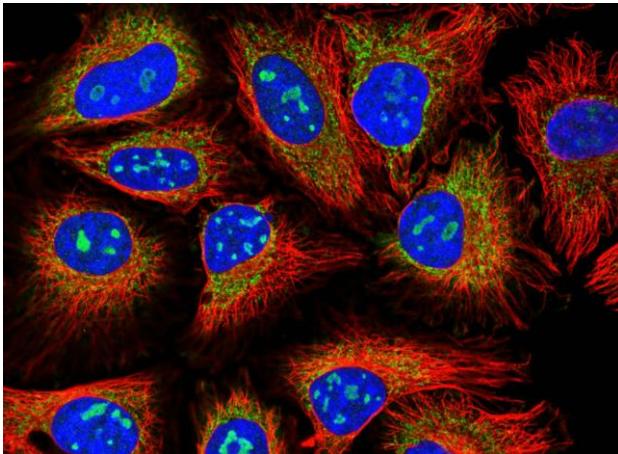
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

ImageNet：对自然对象进行分类
(1000类)



Kaggle上的各种分类任务

- 将人类蛋白质显微镜图像分为28类



0. Nucleoplasm
1. Nuclear membrane
2. Nucleoli
3. Nucleoli fibrillar
4. Nuclear speckles
5. Nuclear bodies
6. Endoplasmic reticu
7. Golgi apparatus
8. Peroxisomes
9. Endosomes
10. Lysosomes
11. Intermediate fila
12. Actin filaments
13. Focal adhesion si
14. Microtubules
15. Microtubule ends
16. Cytokinetic bridge

<https://www.kaggle.com/c/human-protein-atlas-image-classification>

Kaggle上的各种分类任务

- 将恶意软件分为9类



<https://www.kaggle.com/c/malware-classification>

Kaggle上的各种分类任务

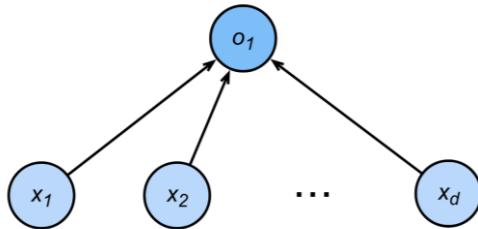
- 将维基百科上的恶语评论分为7类

comment_text	toxic	severe_toxic	obscene
Explanation\nWhy the edits made under my user...	0	0	0
D'aww! He matches this background colour I'm s...	0	0	0
Hey man, I'm really not trying to edit war. It...	0	0	0
"\nMore\nI can't make any real suggestions on ...	0	0	0
You, sir, are my hero. Any chance you remember...	0	0	0

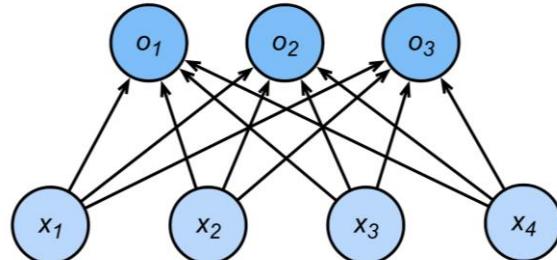
<https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge>

从回归到多类分类

- 单个连续数值输出



- 输出每个类别的置信度分数



- 预测类别

$$\operatorname{argmax}_i(o_1, o_2, o_3)$$

- \max 不可求导
- 定义一个损失函数

softmax 函数

$$\text{softmax}([x_1, x_2, \dots, x_n]^T) = \left[\frac{e^{x_1}}{\sum_{i=1}^n e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^n e^{x_i}}, \dots, \frac{e^{x_n}}{\sum_{i=1}^n e^{x_i}} \right]$$

- 用 \exp 获得大于0的值
- 除以总和以获得概率分布

例: [1, -1, 2] 的 softmax 为 [0.26, 0.04, 0.7]

softmax 梯度

$$\text{softmax}([x_1, x_2, \dots, x_n]^T) = \left[\frac{e^{x_1}}{\sum_{i=1}^n e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^n e^{x_i}}, \dots, \frac{e^{x_n}}{\sum_{i=1}^n e^{x_i}} \right]$$

$$\frac{\partial}{\partial \mathbf{x}} \text{softmax}(\mathbf{x}) = \mathbf{x}\mathbf{x}$$

是否用平方损失?

- 对 y 进行一位有效编码 (One-hot encoding)

$$\mathbf{y} = [y_1, y_2, \dots, y_n]^T, y_i = \begin{cases} 1 & \text{if } i = y \\ 0 & \text{otherwise} \end{cases}$$

- 我们需要 softmax 结果接近 y
- 用平方损失: $y = [0, 0, 1]$

softmax 结果	损失
[0.3, 0, 0.7]	0.18
[0.17, 0.17, 0.66]	0.173

交叉熵

- 一位有效编码与 softmax 运算结果都为概率分布
- 交叉熵通常用于比较概率分布

• 如果 $y = i$, 那么

$$H(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^n y_i \log(\hat{y}_i)$$

$$H(\mathbf{y}, \hat{\mathbf{y}}) = -\log(\hat{y}_i)$$

线性回归与softmax回归

问题	线性回归 (回归问题)	softmax回归 (分类问题)
模型	$\langle \mathbf{w}, \mathbf{x} \rangle + b$ $\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}$	$softmax(\mathbf{Wx} + \mathbf{b})$ $\mathbf{W} \in \mathbb{R}^{k \times n}, \mathbf{b} \in \mathbb{R}^k$
损失	平方损失	交叉熵

总结

- 线性方法
 - 神经科学的灵感
- 基础优化
- softmax 回归
 - 回归与分类
 - Kaggle上的分类任务

动手学深度学习

5. 最大似然估计 和 逻辑回归

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/loss.html>

概要

- 最大似然估计
- 损失函数
 - l_2 失损
 - l_1 失损
 - Huber 失损



MAGIC Etch A Sketch[®] SCREEN

最大似然估计



Horizontal
Dial



Vertical
Dial

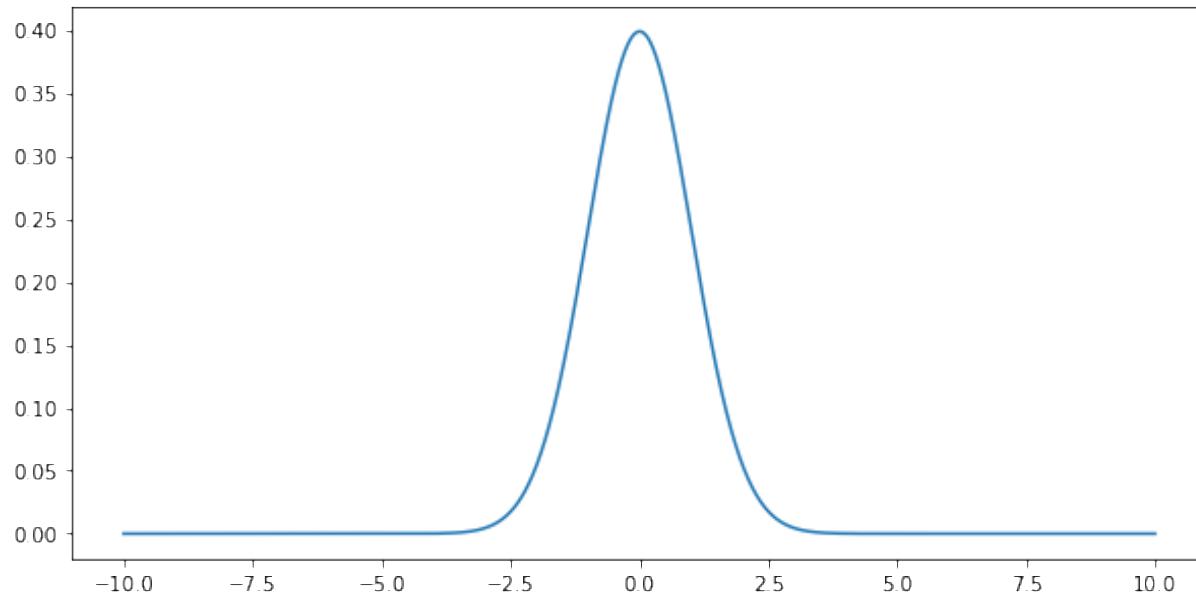
OHIO ART TM The World of Toys[®]

MAGIC SCREEN IS GLASS SET IN STURDY PLASTIC FRAME
USE WITH CARE

D2L.ai

正态分布

概率密度函数 $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$



估计正态分布的参数

- 均值

$$\mu = \mathbf{E}[x], \text{ hence } \hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i$$

- 方差

$$\sigma^2 = \mathbf{E}[(x - \mu)^2], \text{ hence } \hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu})^2$$

为什么呢？

可能性

- 观测记录数据 $X = \{x_1, \dots, x_n\}$

- 假设数据是从高斯分布生成的

$$p(X; \mu, \sigma^2) = \prod_{i=1}^n p(x_i; \mu, \sigma^2) = \prod_{i=1}^n \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right)$$

- 拟合参数相当于使 $p(X; \mu, \sigma^2)$ 关于 μ, σ^2 最大化

- 实用简化版本：

$$\underset{\mu, \sigma^2}{\text{maximize}} p(X; \mu, \sigma^2) \iff \underset{\mu, \sigma^2}{\text{minimize}} -\log p(X; \mu, \sigma^2)$$

最大似然估计

- 通过可解释数据来估算参数

$$\underset{\mu, \sigma^2}{\text{minimize}} -\log p(X; \mu, \sigma^2)$$

- 展开式子

$$\begin{aligned}-\log p(X; \mu, \sigma^2) &= \sum_{i=1}^n \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} (x_i - \mu)^2 \\ &= \frac{n}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2\end{aligned}$$

当 $\mu = \frac{1}{n} \sum_{i=1}^n x_i$ 时，可最小化

最大似然估计

- 估计方差

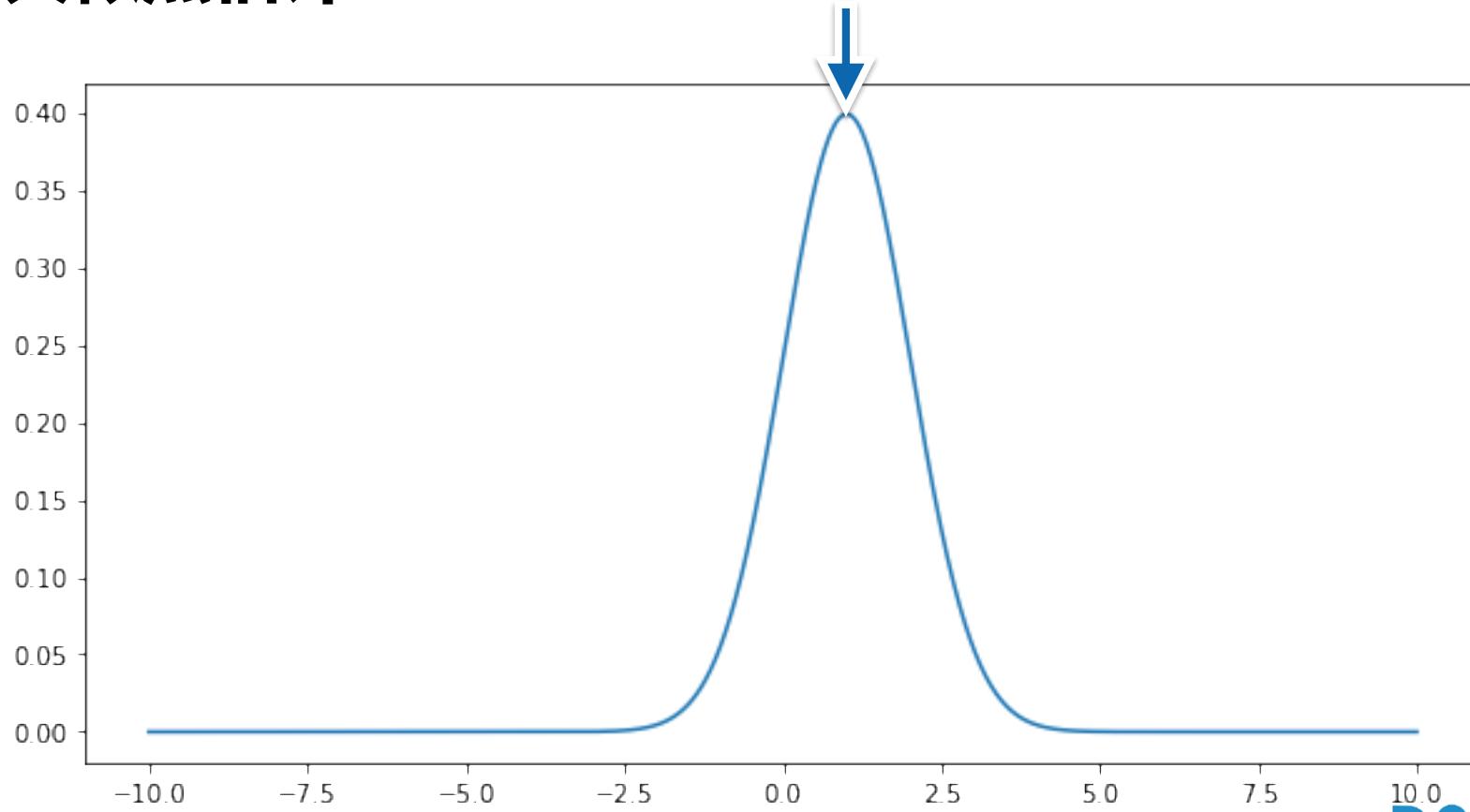
$$\frac{n}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2$$

- 对其求导

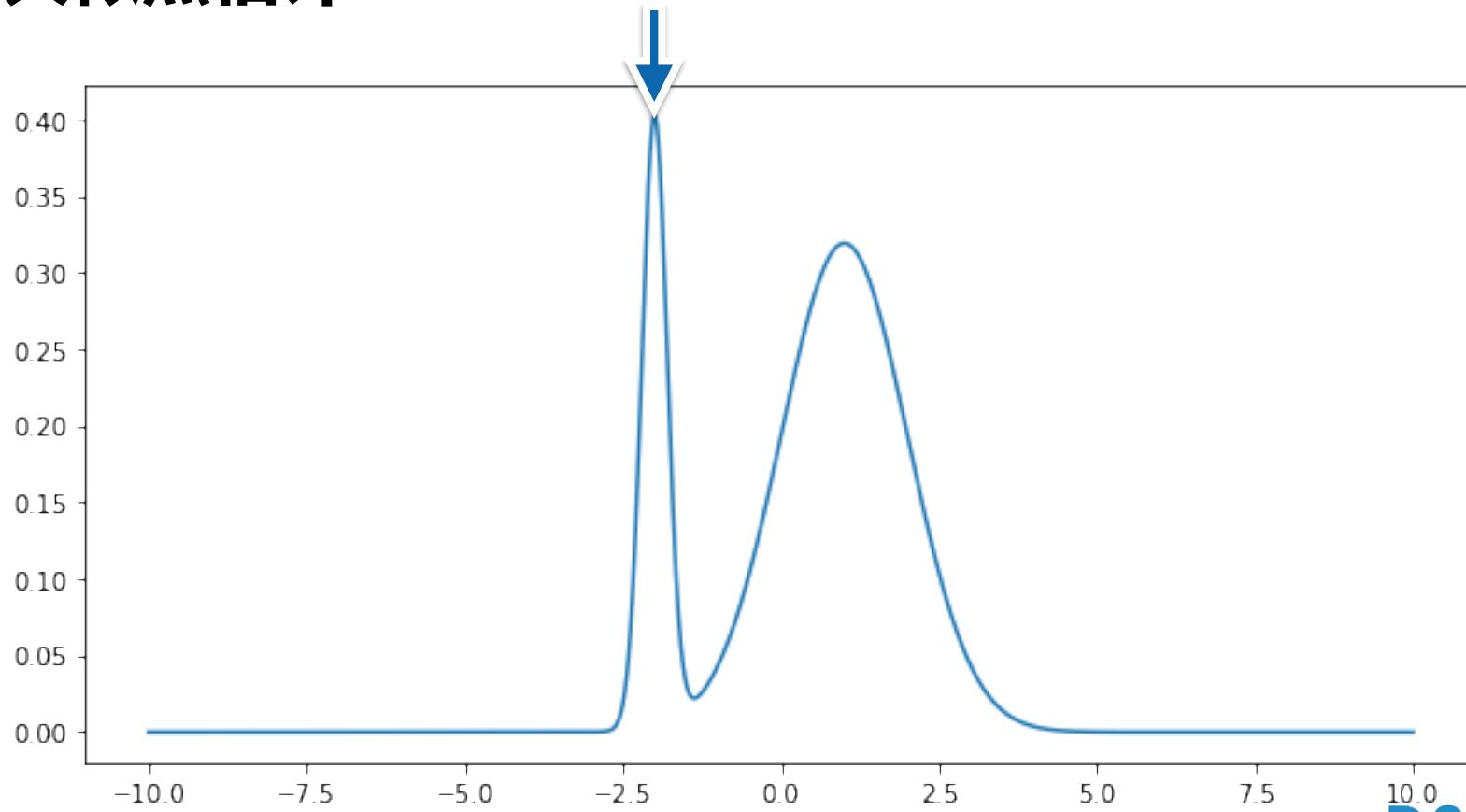
$$\partial_{\sigma^2} [\cdot] = \frac{n}{2\sigma^2} - \frac{1}{2\sigma^4} \sum_{i=1}^n (x_i - \mu)^2 = 0$$

$$\Rightarrow \sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2$$

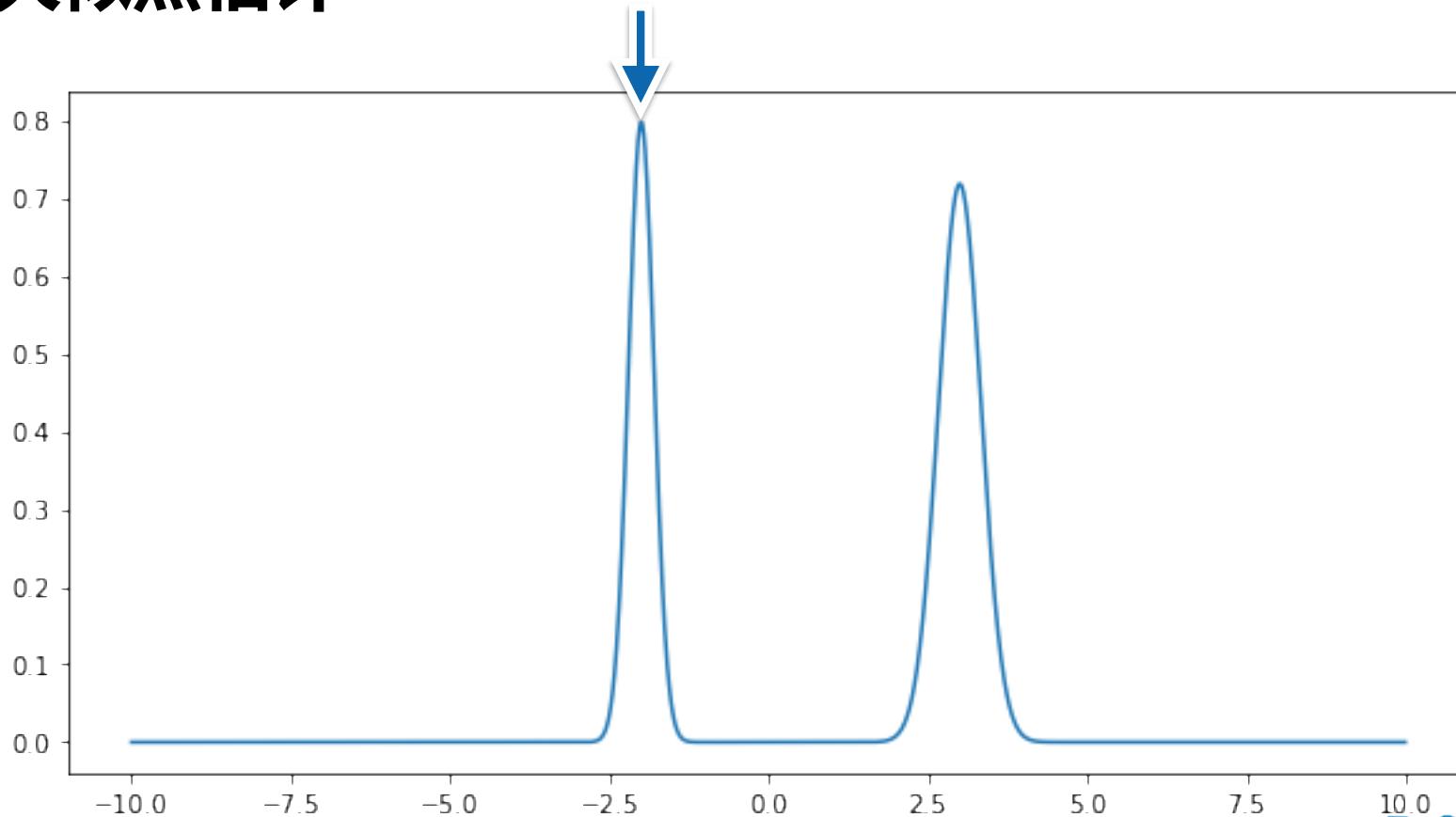
最大似然估计



最大似然估计



最大似然估计



最大似然估计

- 已有数据 - ‘学生未交作业’
- 可能的参数
 - ‘狗吃了作业’
 - ‘外星人拿走了作业’
 - ‘学生太懒’
 - ‘祖母生病’
- 所有参数可解释数据

我们应该相信全部嘛？

最大后验估计

- 后验概率

$$p(w | X) \propto p(X | w)p(w)$$

惩罚

$$\text{hence } -\log p(w | X) = -\log p(X | w) - \log p(w) + c$$

- 最大后验估计

$$\underset{w}{\text{minimize}} -\log p(X; w) - \log p(w)$$

- 例子

$$P(\text{未完成作业} | \text{合理解释}) = P(\text{合理解释} | \text{未完成作业}) / P(\text{未完成作业})$$

lazy student	grandma sick	dog ate it	alien abduction
0.8	0.19	0.0099	0.0001

这与回归有什么关系？

回归

- 回顾 - 优化问题

$-\log p(w)$

$$\underset{w}{\text{minimize}} \sum_{i=1}^n (y_i - f(x_i, w))^2 + \text{penalty}(w)$$

该模型有效吗？

添加高斯噪声

- 数据生成模型

$$y_i = f(x_i, w) + \epsilon_i \text{ where } \epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

- 高斯先验函数 $p(w) = \frac{1}{2\bar{\sigma}^2} \|w\|^2 + \text{const.}$

回归

- 最大后验估计

$$\underset{w}{\text{minimize}} -\log p(w | X, Y)$$

$$\iff \underset{w}{\text{minimize}} \frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - f(x_i, w))^2 + \frac{1}{2\bar{\sigma}^2} \|w\|^2 + \text{const.}$$

$$\iff \underset{w}{\text{minimize}} \frac{1}{2n} \sum_{i=1}^n (y_i - f(x_i, w))^2 + \frac{\lambda}{2} \|w\|^2$$



MAGIC Etch A Sketch[®] SCREEN

损失函数



Horizontal
Dial



Vertical
Dial

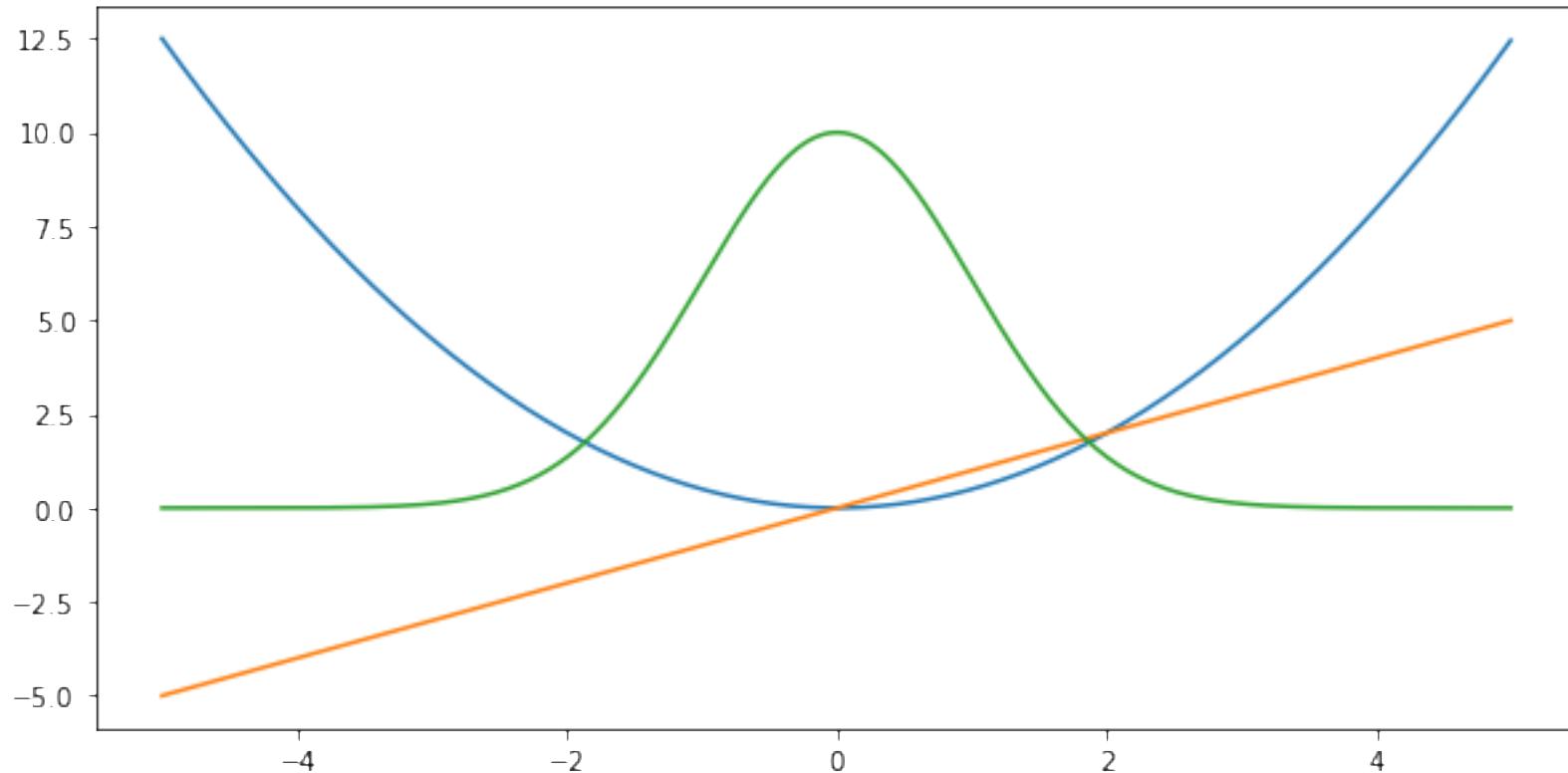
OHIO ART *The World of Toys*[®]

MAGIC SCREEN IS GLASS SET IN STURDY PLASTIC FRAME
USE WITH CARE

2L.ai

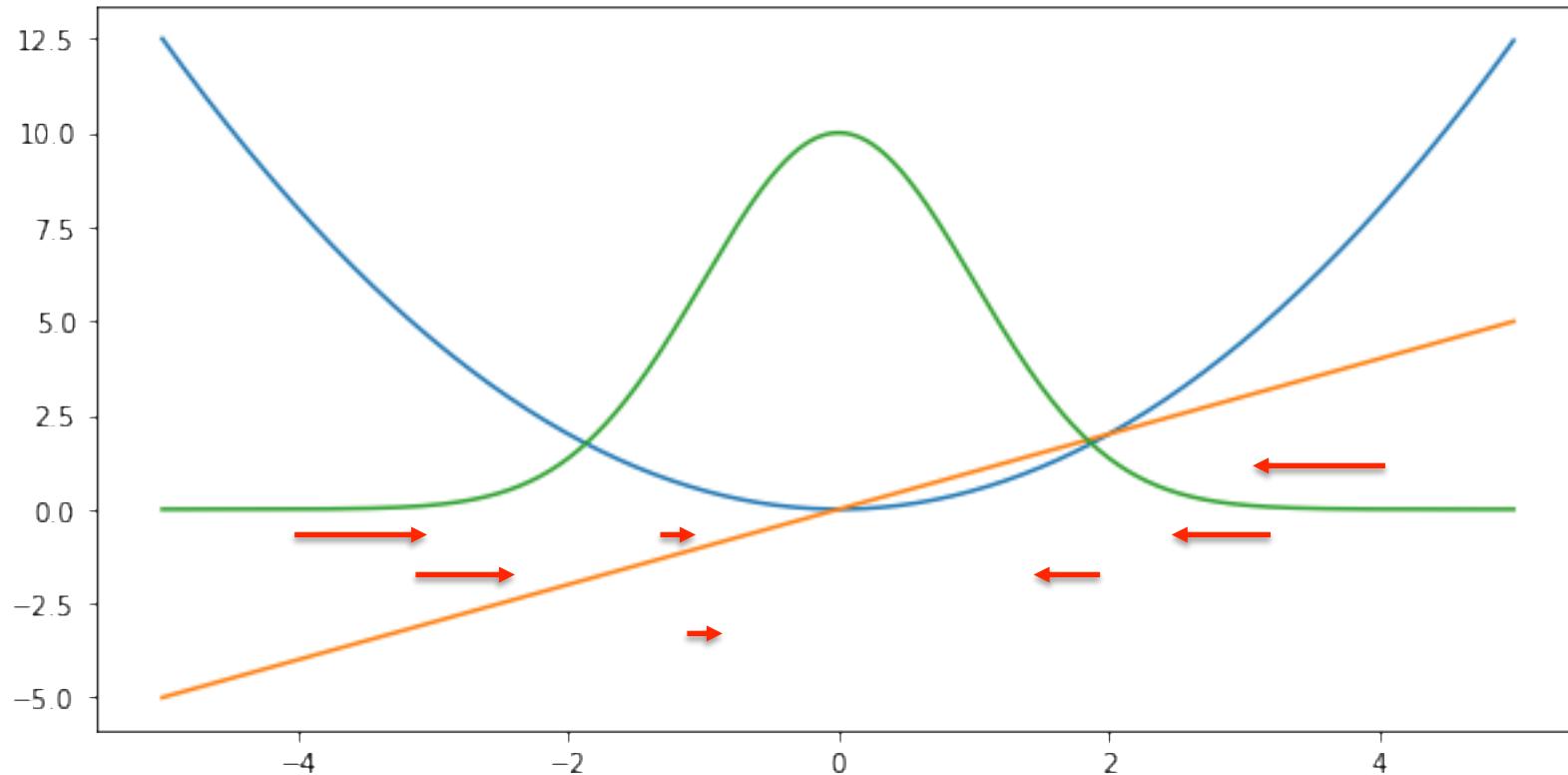
平方损失 (L2 损失)

$$l(y, y') = \frac{1}{2} (y - y')^2$$



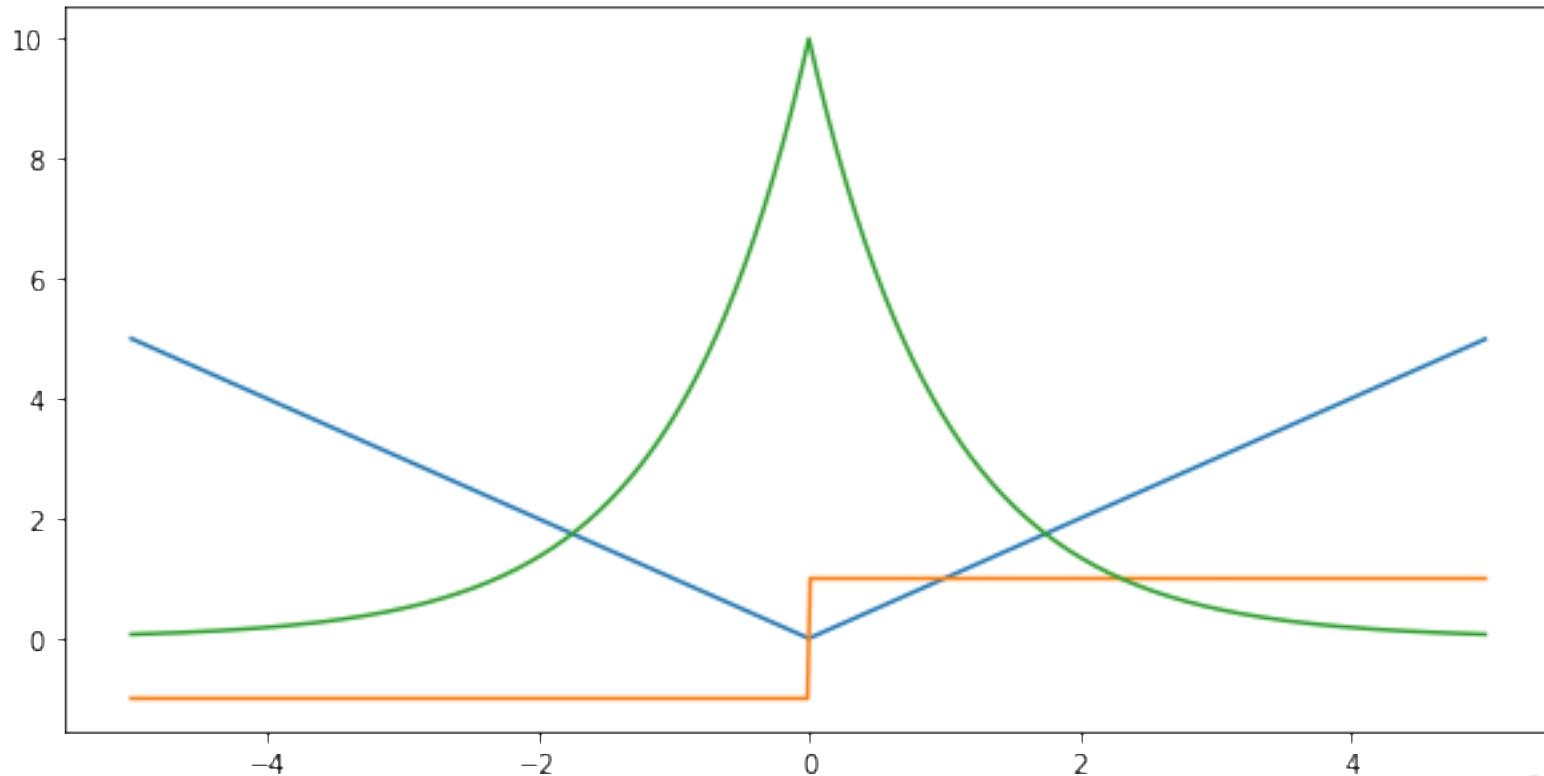
平方损失 (L2 损失)

$$l(y, y') = \frac{1}{2}(y - y')^2$$



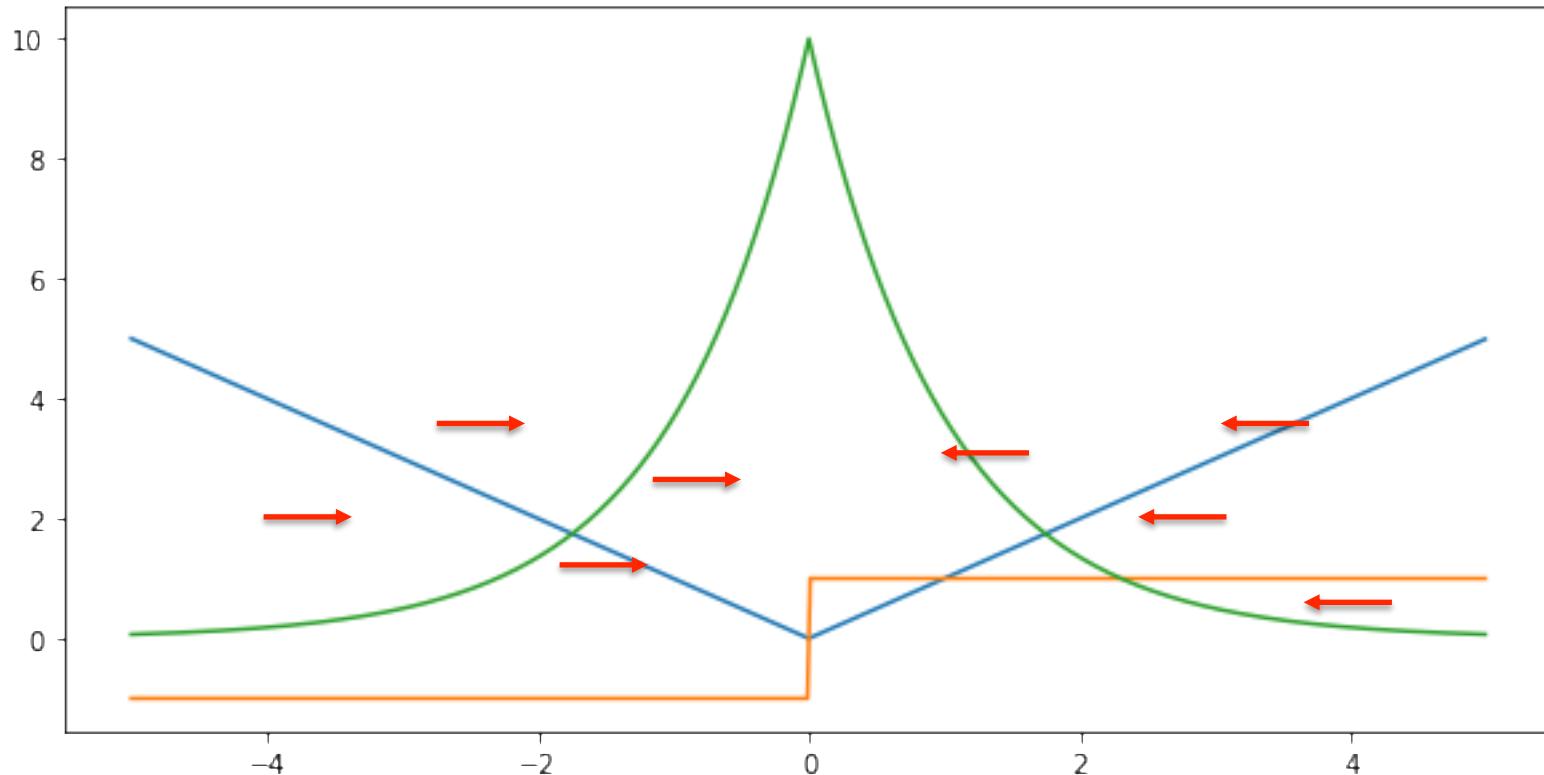
L1 损失

$$l(y, y') = |y - y'|$$

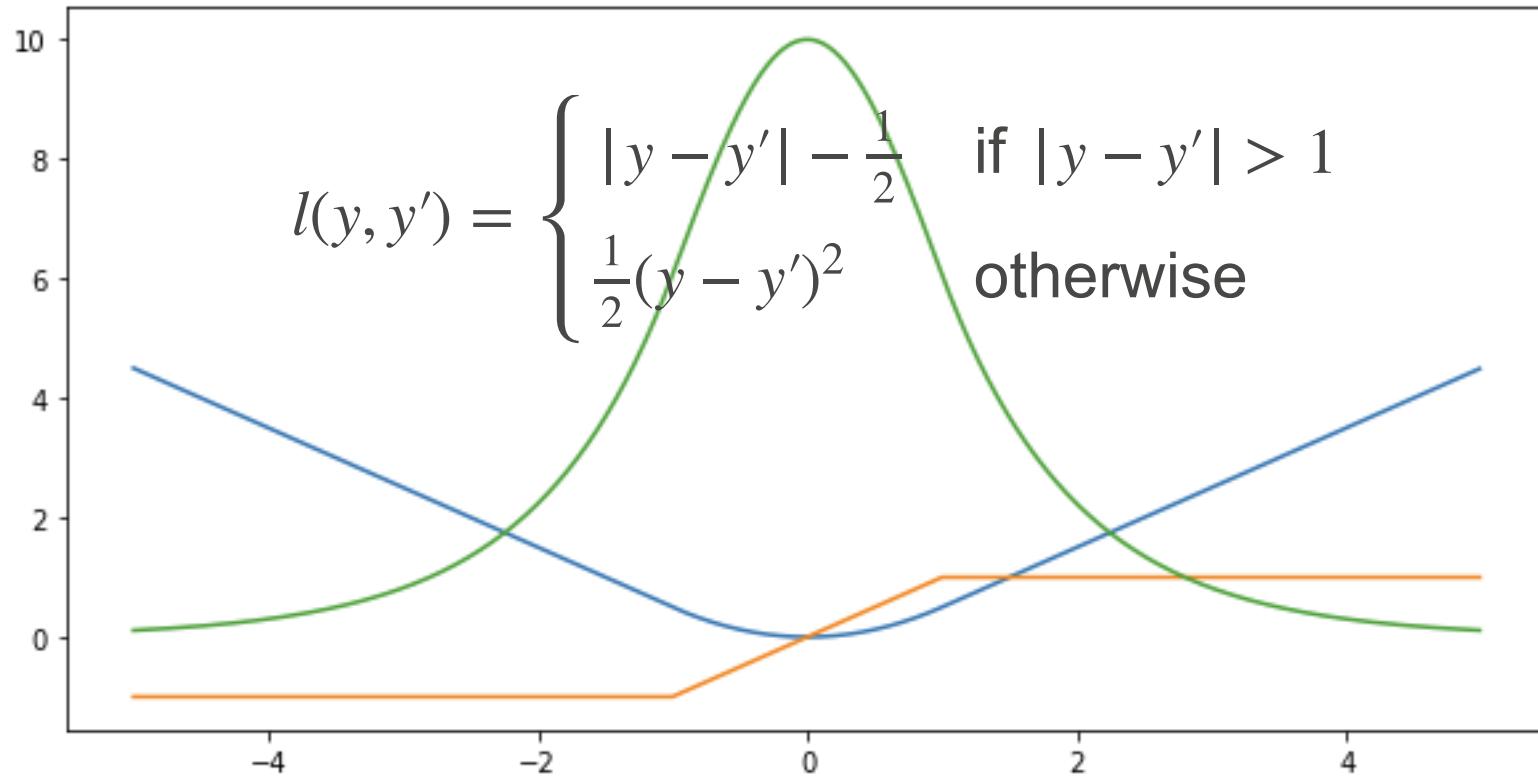


L1 损失

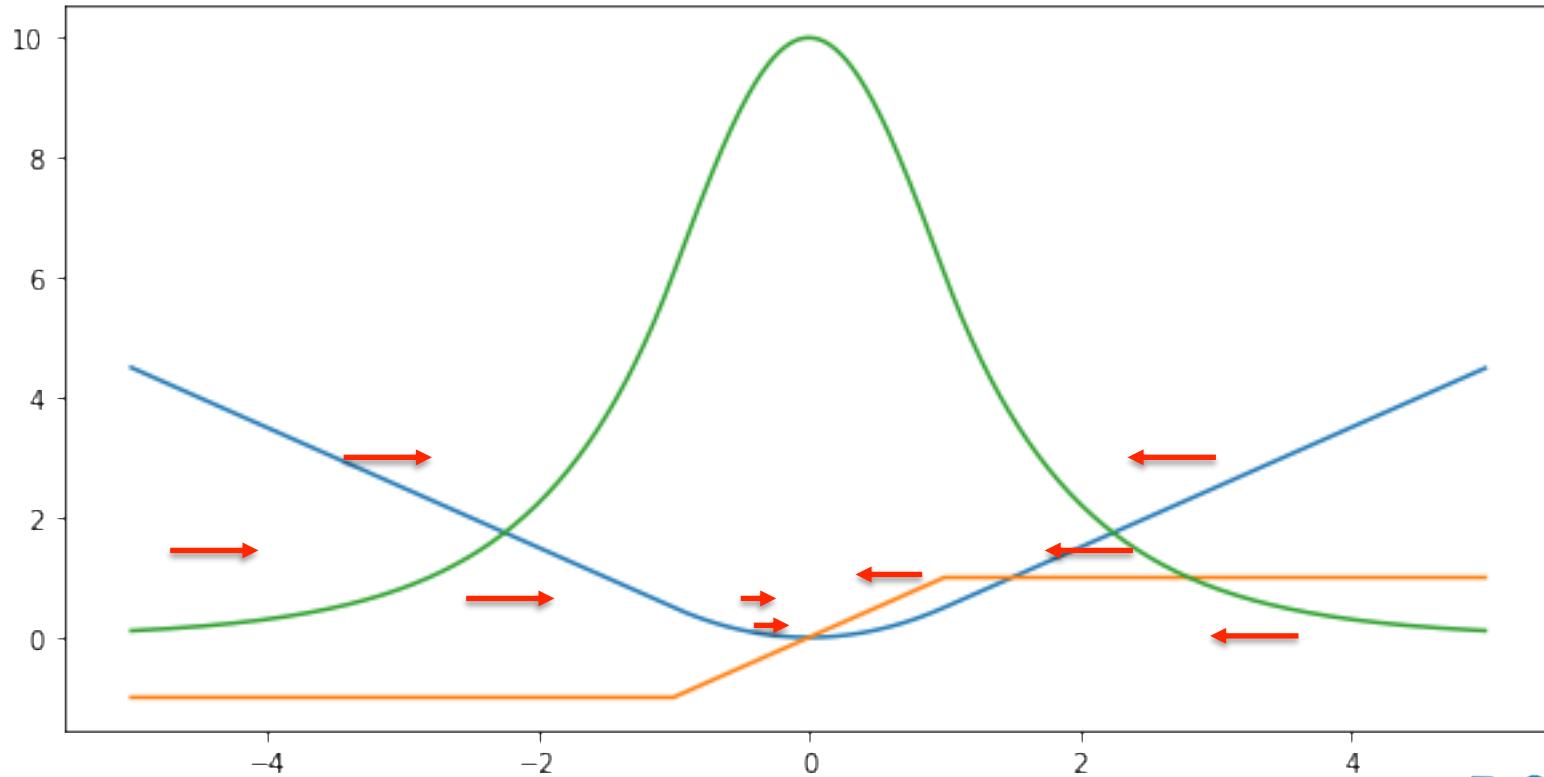
$$l(y, y') = |y - y'|$$



Huber 损失



Huber 损失



总结

- 最大似然估计
- 损失函数
 - l_2 失损
 - l_1 失损
 - Huber 失损

动手学深度学习

6.多层感知机

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

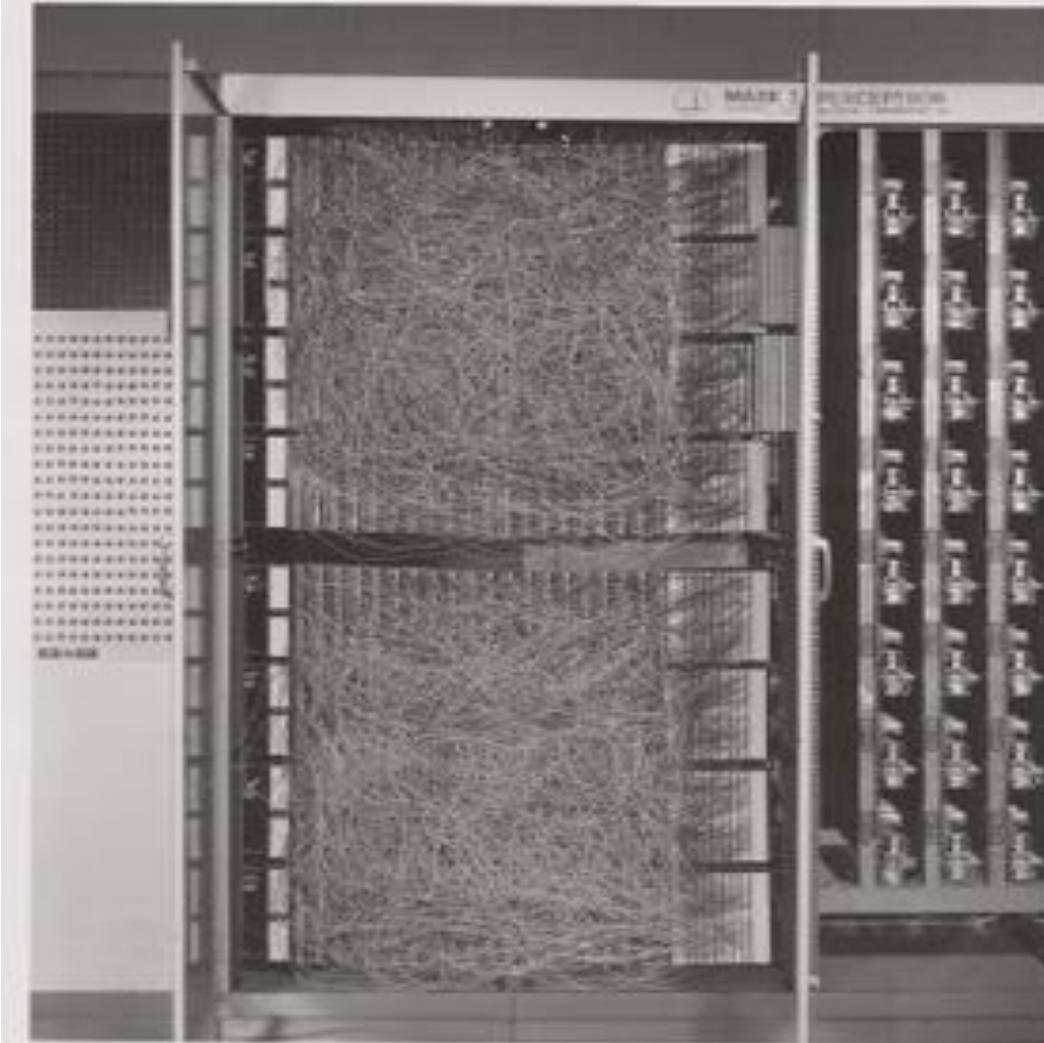
教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/mlp.html>

概要

- **单层感知机**
 - 决策边界
 - XOR
- **多层感知机**
 - 层数
 - 非线性
 - 计算成本

感知机

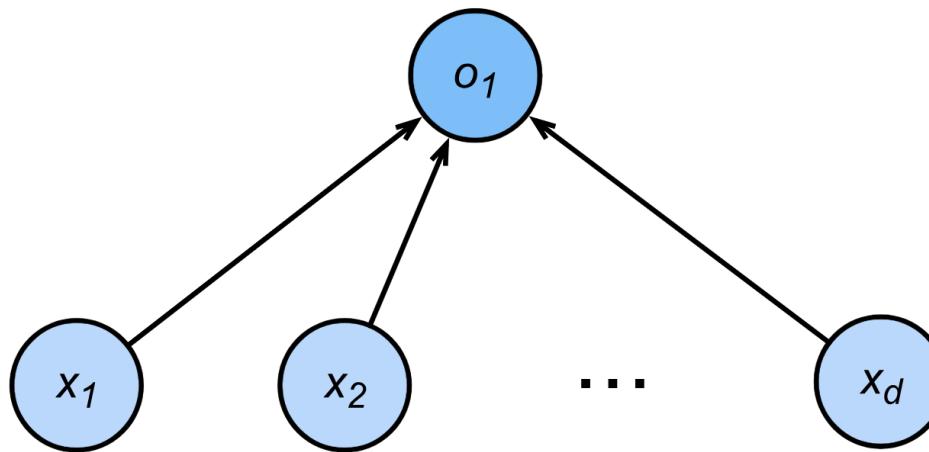
Mark I Perceptron, 1960
([wikipedia.org](https://en.wikipedia.org))



感知机

- 给予输入 \mathbf{x} , 权重 \mathbf{w} 和偏差 b ; 感知机输出:

$$o = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b) \quad \sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

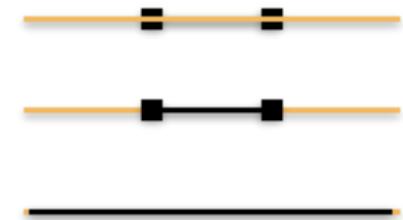


感知机

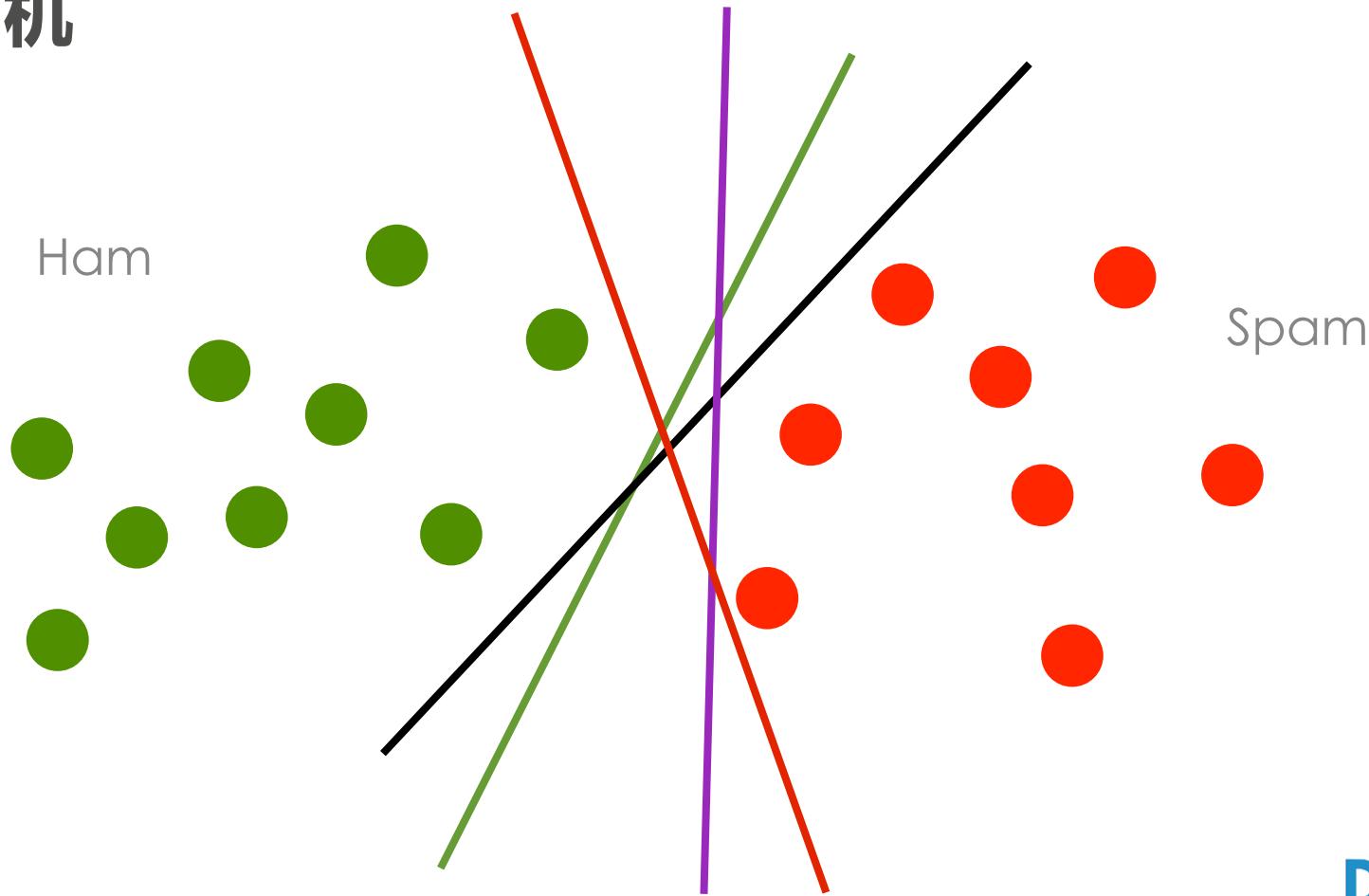
- 给予输入 x , 权重 w 和偏差 b ; 感知机输出:

$$o = \sigma(\langle w, x \rangle + b) \quad \sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

- 二分类 (0 或 1)
 - Vs. 回归的标量实际值
 - Vs. 逻辑回归的概率



感知机

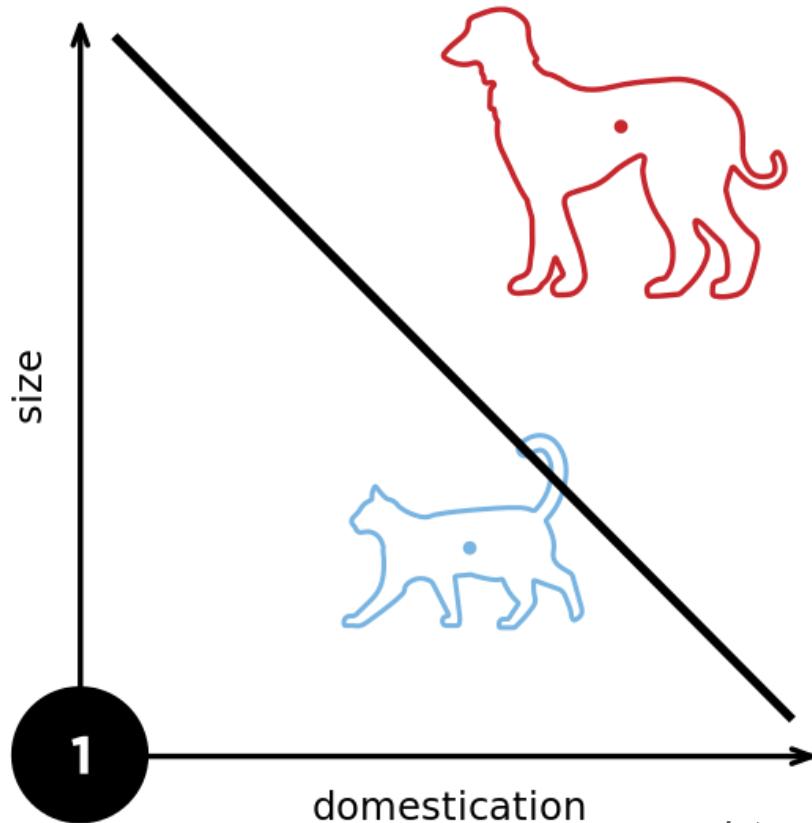


训练感知机

代码：
 initialize $w = 0$ and $b = 0$
 repeat
 if $y_i [\langle w, x_i \rangle + b] \leq 0$ **then**
 $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$
 end if
 until all classified correctly

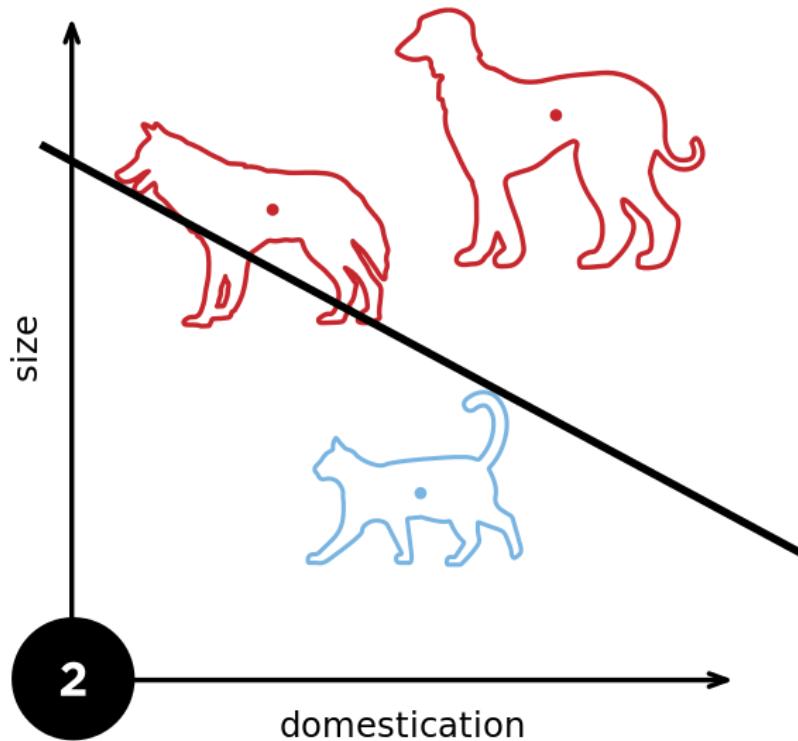
相当于批量为 1 的随机梯度下降 (SGD) 用于一下损失：

$$\ell(y, \mathbf{x}, \mathbf{w}) = \max(0, -y \langle \mathbf{w}, \mathbf{x} \rangle)$$



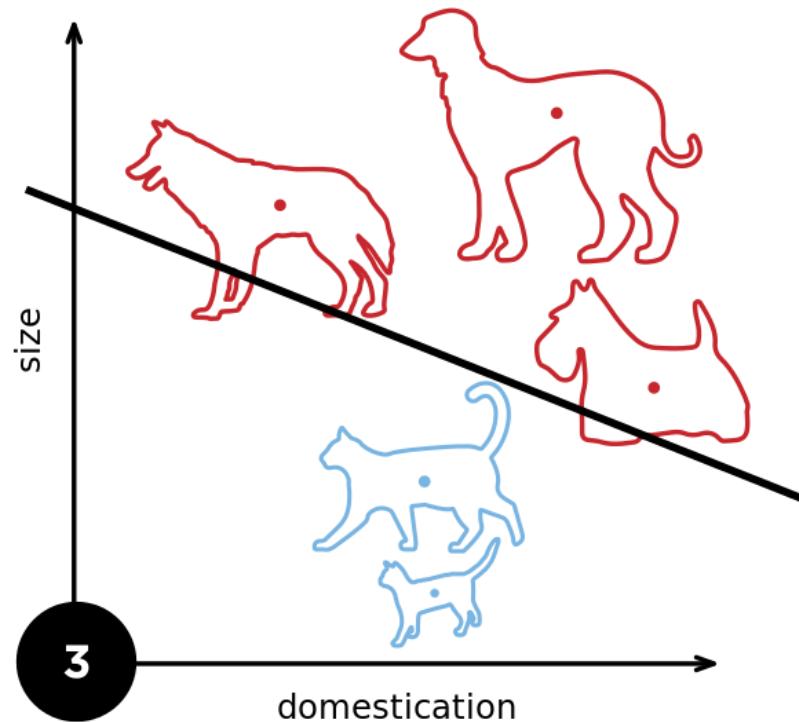
来源于维基百科

D2L.ai



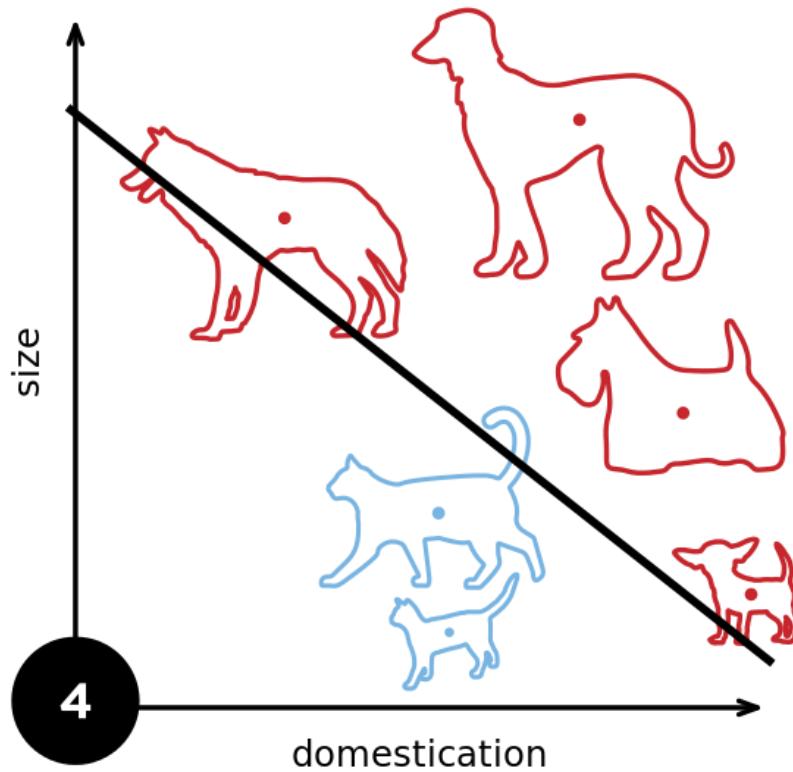
来源于维基百科

D2L.ai



来源于维基百科

D2L.ai



来源于维基百科

D2L.ai

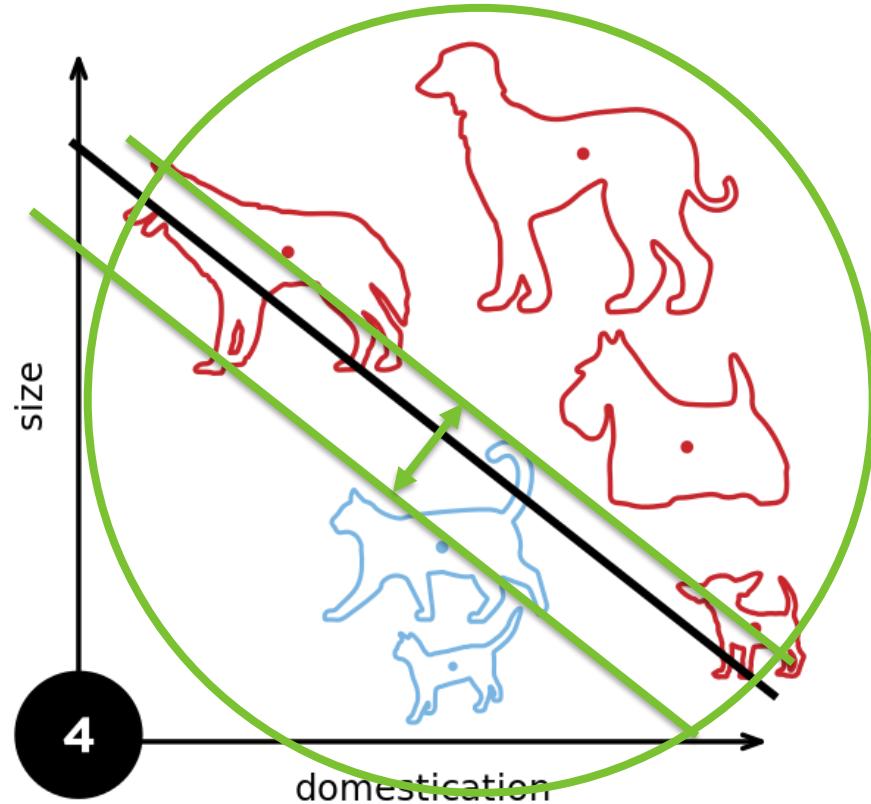
收敛定理

- 半径 r 包含数据
- 间隔 ρ 区分类别

$$y(\mathbf{x}^\top \mathbf{w} + b) \geq \rho$$

$$\text{for } \|\mathbf{w}\|^2 + b^2 \leq 1$$

- 保证感知机会在 $\frac{r^2+1}{\rho^2}$ 步之后收敛



结果

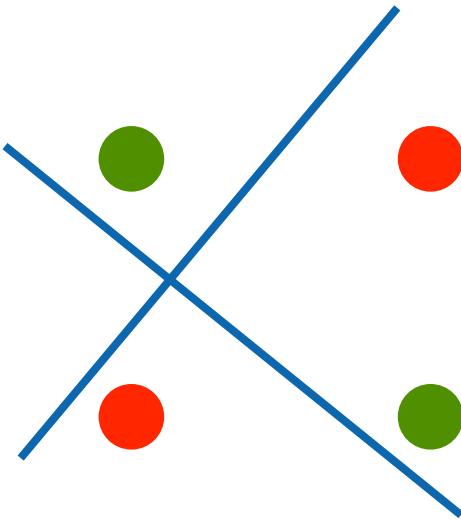
- 只需要存储错误
- 这给出了感知机的压缩界限
- **由于嘈杂的数据而失败**

不要用感知机来训练你的头像

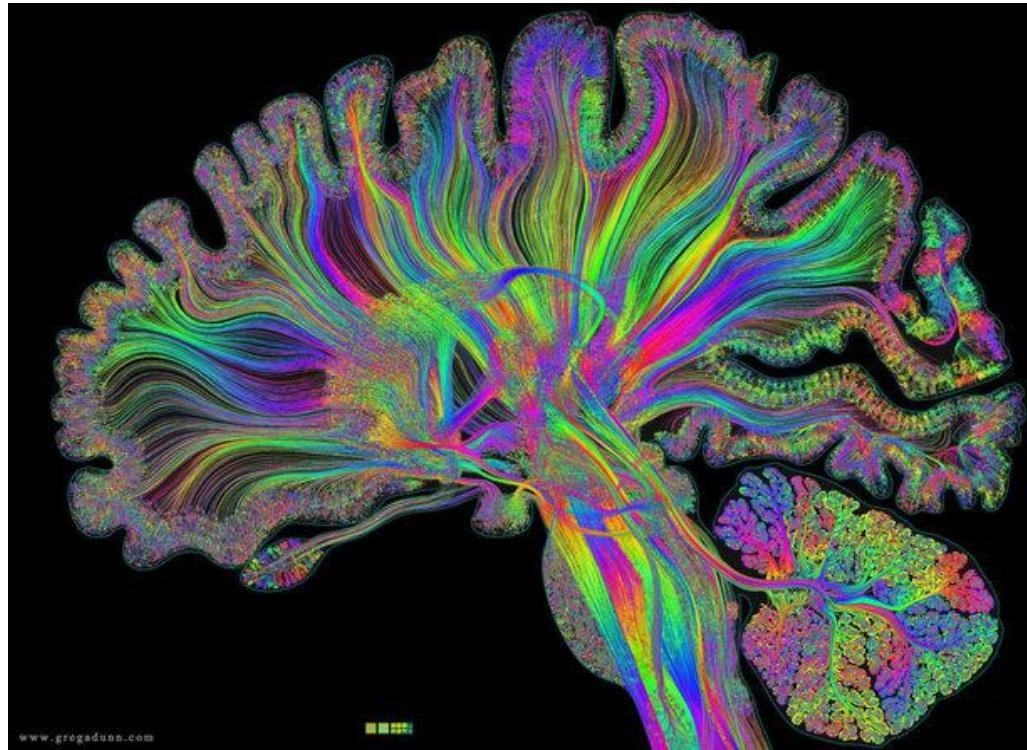


XOR 问题 (Minsky & Papert, 1969)

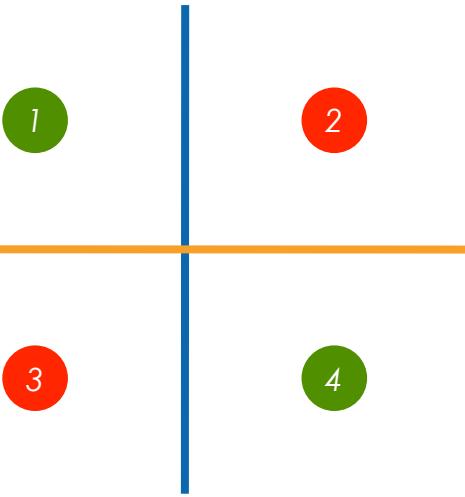
感知机无法学习 XOR 问题 (神经元只能生成线性分离器)



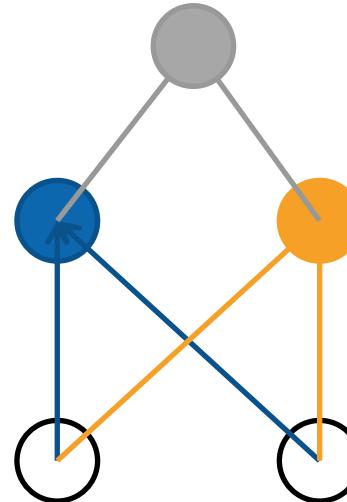
多层感知机



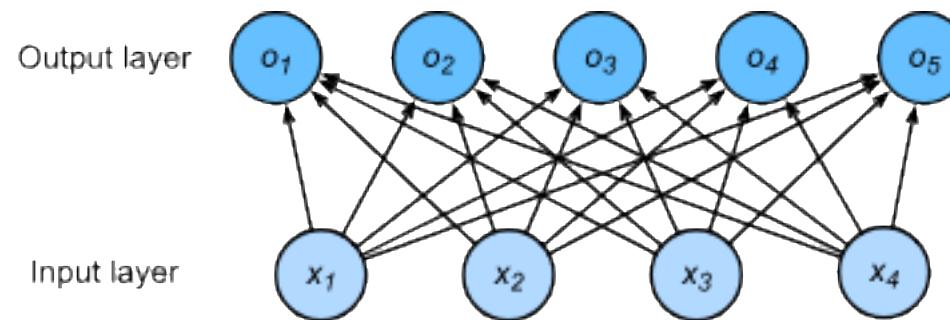
学习 XOR



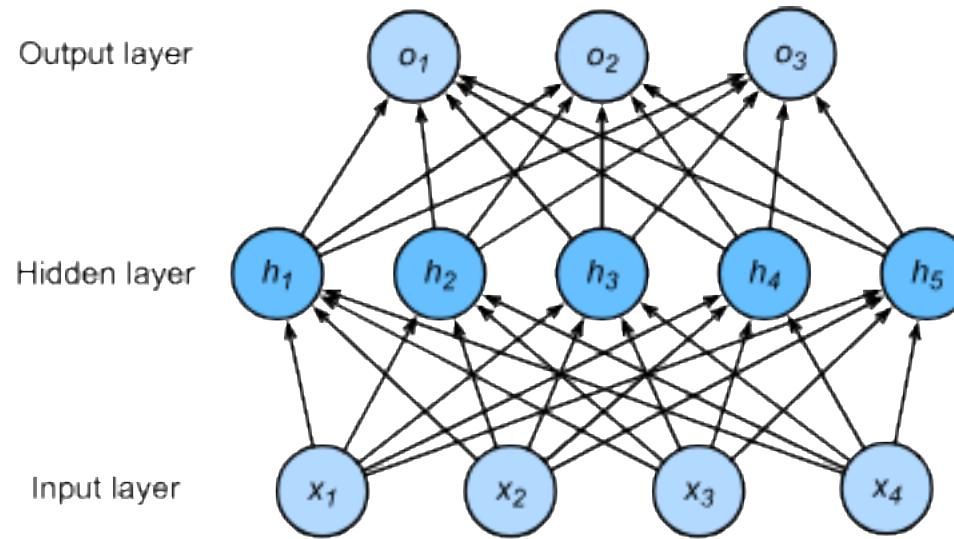
	1	2	3	4
1	+	-	+	-
2	+	+	-	-
乘积	+	-	-	+



单隐藏层



单隐藏层



超参数 - 隐藏层的大小为m

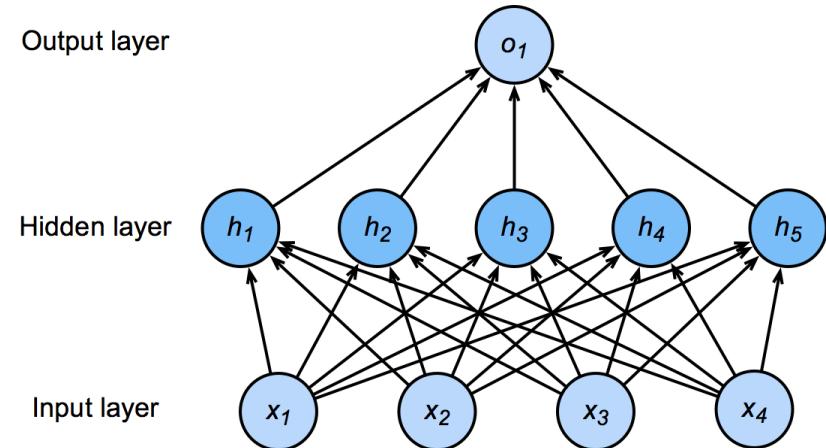
单隐藏层

- 输入: $\mathbf{x} \in \mathbb{R}^n$
- 隐藏层: $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$
- 输出: $\mathbf{w}_2 \in \mathbb{R}^m, b_2 \in \mathbb{R}$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T \mathbf{h} + b_2$$

激活函数



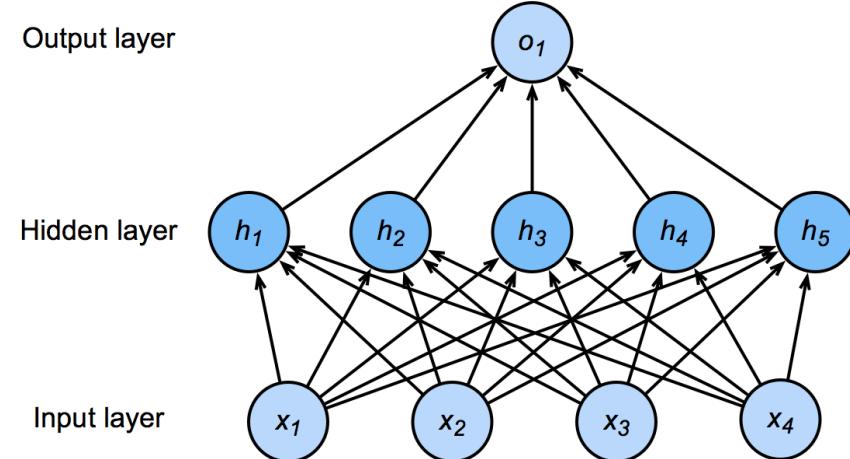
单隐藏层

为什么我们需要非线性激活函数呢？

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T \mathbf{h} + b_2$$

一个逐元素激活函数



单隐藏层

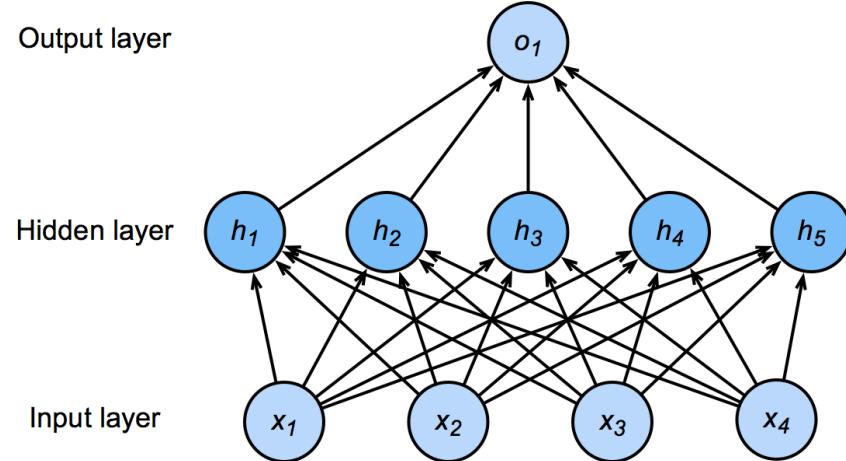
为什么我们需要非线性激活函数呢？

$$\mathbf{h} = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1$$

$$\mathbf{o} = \mathbf{w}_2^T \mathbf{h} + b_2$$

$$\text{hence } o = \mathbf{w}_2^T \mathbf{W}_1 \mathbf{x} + b'$$

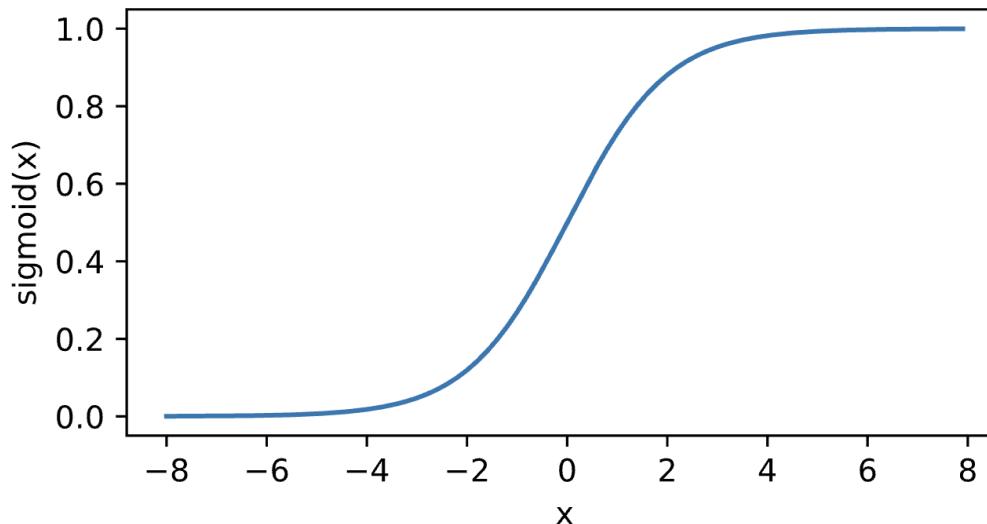
因为它是线性的...



S型 (sigmoid) 激活函数

将输入映射到 $(0, 1)$, 一个平滑的版本 $\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$

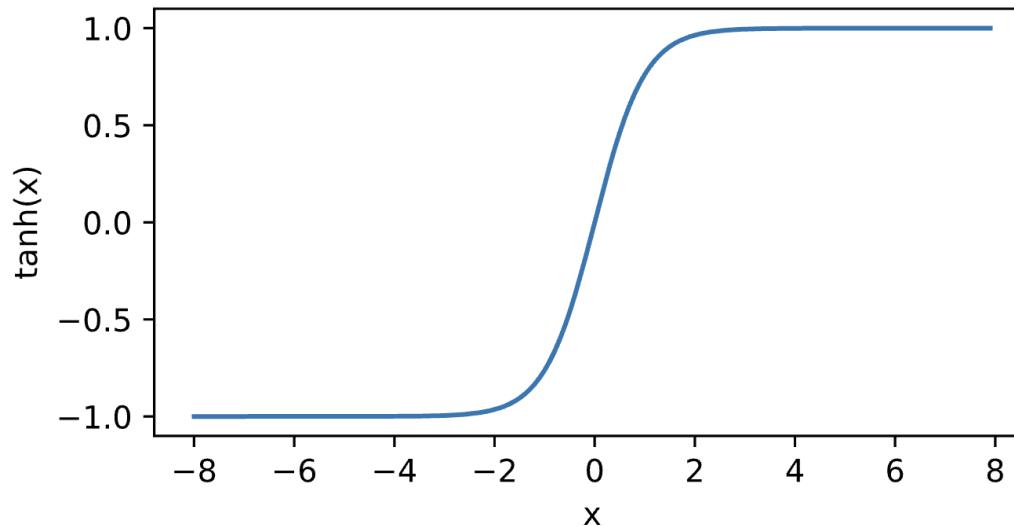
$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



双曲正切 (tanh) 激活函数

将输入映射到 (-1, 1)

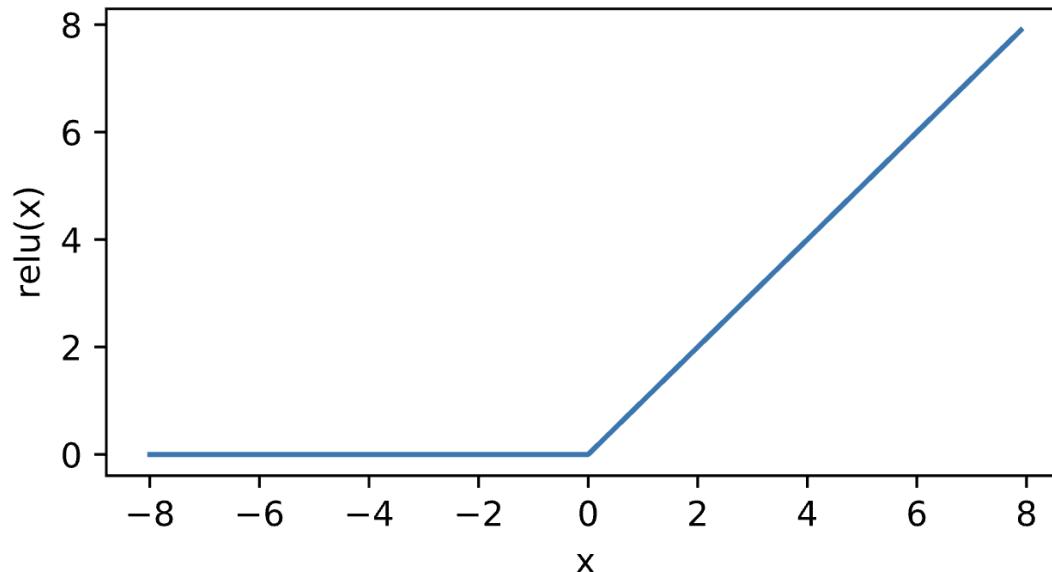
$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$



线性 (ReLU) 修正函数

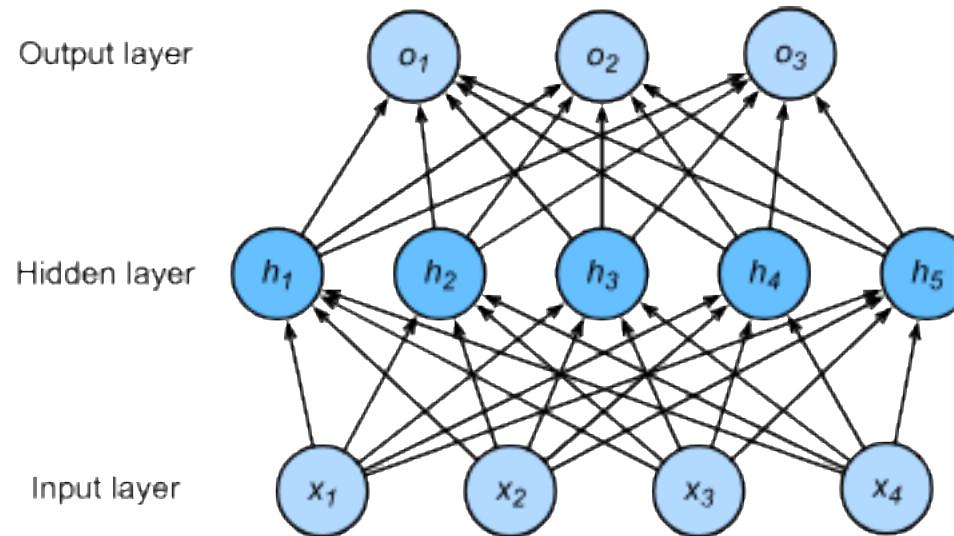
ReLU: 线性修正单元

$$\text{ReLU}(x) = \max(x, 0)$$



多类别分类

$$y_1, y_2, \dots, y_k = \text{softmax}(o_1, o_2, \dots, o_k)$$



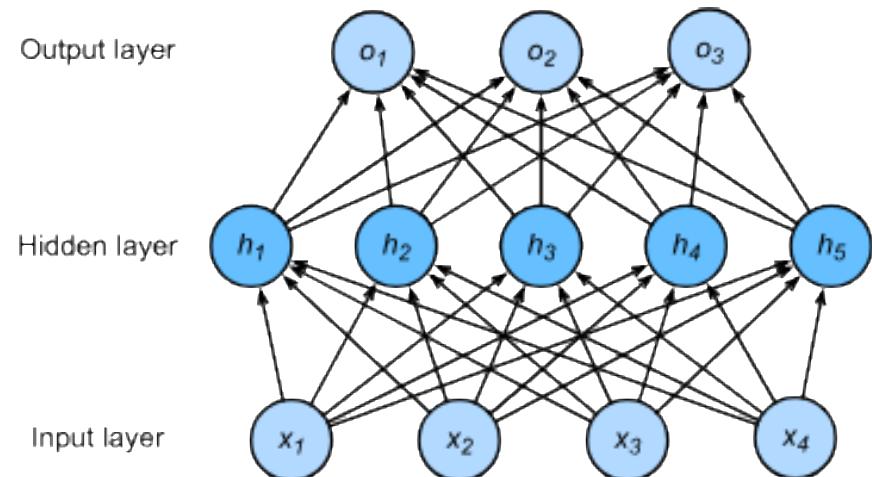
多类分类

- 输入: $\mathbf{x} \in \mathbb{R}^n$
- 隐藏层: $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$
- 输出: $\mathbf{w}_2 \in \mathbb{R}^m, b_2 \in \mathbb{R}$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T \mathbf{h} + b_2$$

一个按元素激活函数



多个隐藏层多类分类

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

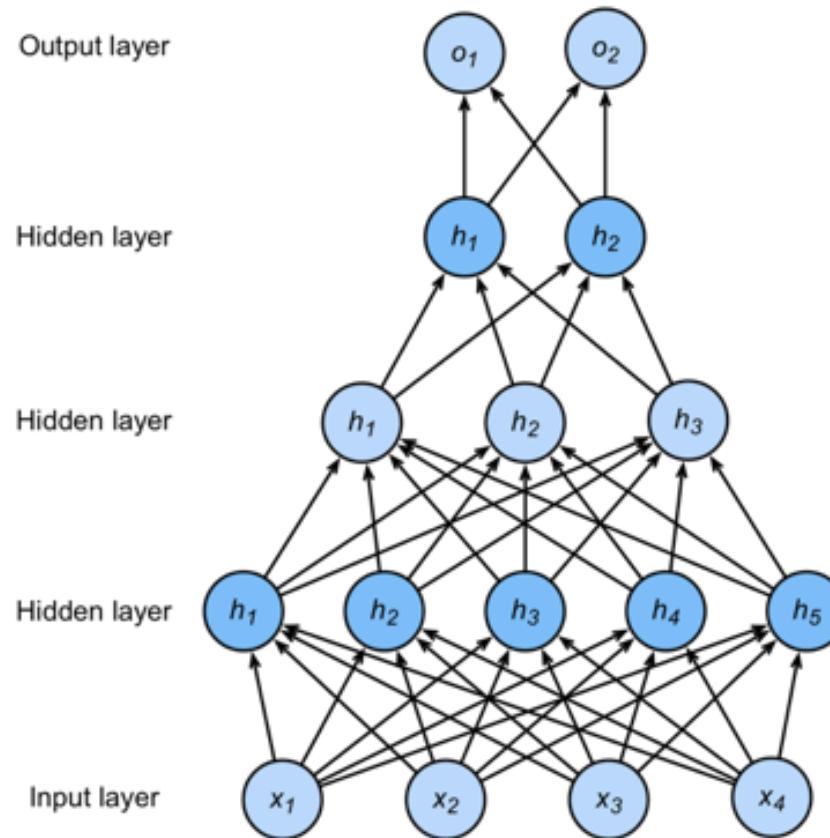
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2)$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}_3 \mathbf{h}_2 + \mathbf{b}_3)$$

$$\mathbf{o} = \mathbf{W}_4 \mathbf{h}_3 + \mathbf{b}_4$$

超参数

- 隐藏层数量
- 每层的隐藏单元数目



总结

- **单层感知机**
 - 决策边界
 - XOR
- **多层感知机**
 - 层数
 - 非线性
 - 计算成本

动手学深度学习

7. 模型选择，欠拟合与过拟合，权重衰减 和 丢弃法

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/capacity.html>

概要

- 模型选择
 - 训练误差和泛化误差
 - K 折交叉验证
- 欠拟合与过拟合
 - 模型复杂度
 - VC 维度
- 权重衰减
- 丢弃法

模型选择



训练误差和泛化误差

- 训练误差：出自于训练数据
- 泛化误差：出自于新数据
- 示例：使用历年考试真题准备将来的考试
 - 在历年考试真题取得好成绩（训练误差）并不能保证未来考试成绩好（泛化误差）
 - 学生 A 通过死记硬背学习在历年考试真题取得0错误
 - 学生 B 理解并给出答案的解释

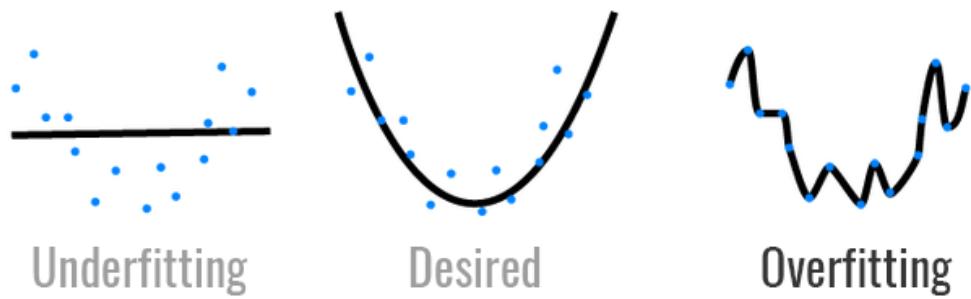
验证数据集和测试数据集

- 验证数据集：用于评估模型的数据集
 - 例如：取出50%的训练数据
 - 不应与训练数据混在一起（# 1 常见错误）
- 测试数据集：只可以使用一次数据集，例如
 - 未来的考试
 - 我买的房子售价
 - Kaggle的私人排行榜中使用的数据集

K 折交叉验证

- 在没有足够的数据时非常有用
- 算法：
 - 将训练数据划分为 K 个部分
 - 对于 $i = 1, \dots, K$
 - 使用第 i 部分作为验证集，其余部分用于训练
 - 报告 K 个部分在验证时的平均错误
- 常见 K 值选择：5 - 10

欠拟合 过拟合



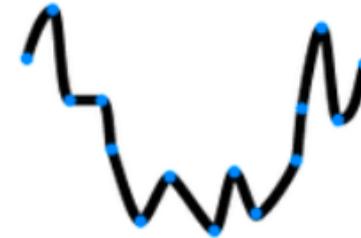
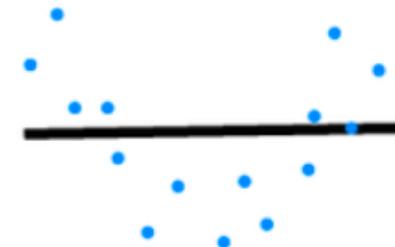
hackernoon.com

欠拟合 和 过拟合

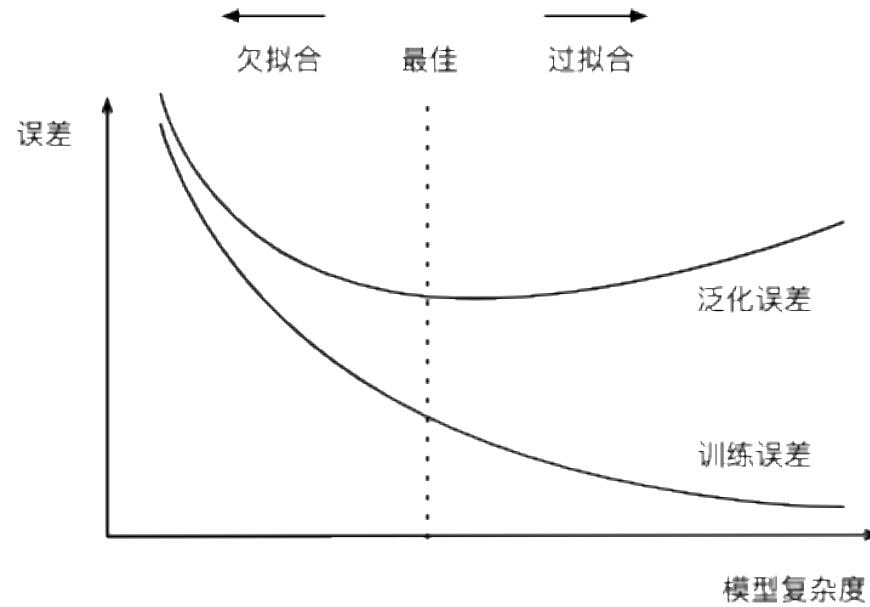
		数据复杂度	
		低	高
模型复杂度	低	正常	欠拟合
	高	过拟合	正常

模型复杂度

- 即“可适应不同数据分布”的能力
- 低容量模型难以适应训练集
 - 欠拟合
- 高容量模型可以记忆训练集
 - 过拟合



模型复杂度的影响



估计模型复杂度

- 很难比较不同算法之间的复杂度
 - 例如 树与神经网络
- 给定算法族，两个主要因素很重要：
 - 参数个数
 - 每个参数采用的值

$$d + 1$$

A diagram of a decision tree. At the bottom are input nodes labeled x_1, x_2, \dots, x_d . Above them is a layer of hidden nodes. The first two hidden nodes have arrows pointing to a single output node labeled o_1 . Ellipses between the hidden nodes indicate more nodes in the layer, and an ellipsis above the output node indicates multiple output nodes.

$$(d + 1)m + (m + 1)k$$

A diagram of a fully connected neural network. At the bottom are input nodes labeled x_1, x_2, x_3, x_4 . Above them is a layer of hidden nodes labeled h_1, h_2, h_3, h_4, h_5 . Every input node has arrows pointing to every hidden node. Above the hidden layer is a layer of output nodes labeled o_1, o_2, o_3 . Every hidden node has arrows pointing to every output node.

VC维度

- 统计学习理论的中心主题
- 对于分类模型，VC维度等于这个数据集的大小，无论我们如何分配标签，都存在一个模型来完美地对它进行分类



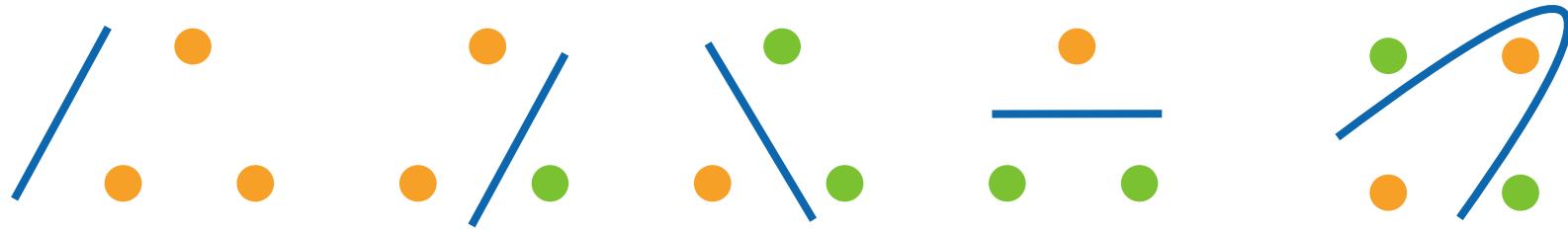
Vladimir Vapnik



Alexey Chervonenkis

线性分类器的VC维度

- 2-D感知机（输入为2-D）： $\text{VCdim} = 3$
- 可以分3类，但不能分4类（XOR）



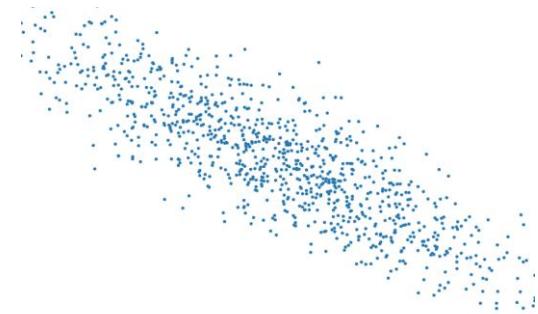
- 具有 N 个参数的感知机： $\text{VCdim} = N$
- 一些多层感知机： $\text{VCdim} = O(N \log_2(N))$

VC维度的效用

- 提供理论解释模型的工作原理
 - 限制了训练误差和泛化误差之间的差距
- 在深度学习的实践中很少使用
 - 边界过于宽松
 - 难以计算深度神经网络的 VC 维数
 - 其他统计学习理论工具也是如此

数据复杂度

- 多种因素很重要
 - 例子数量
 - 每个示例中的元素数量
 - 时间/空间结构
 - 多样性

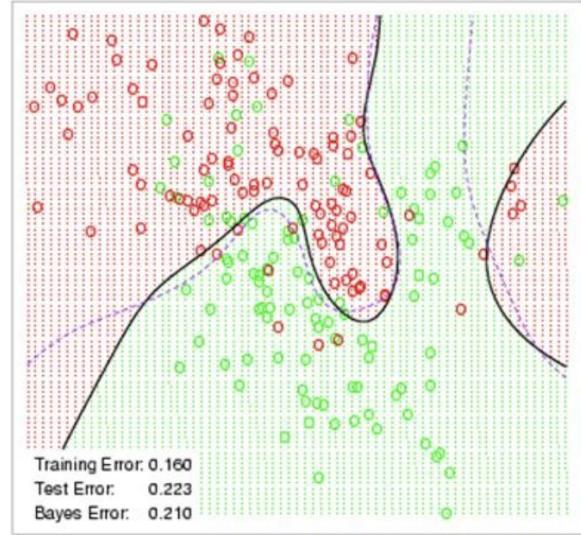


权重衰减

Neural Network - 10 Units, No Weight Decay



Neural Network - 10 Units, Weight Decay=0.02

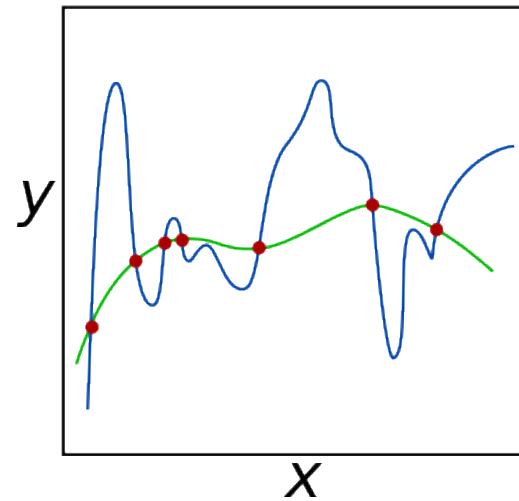


平方正则化作为“硬”约束

- 通过限制参数值范围来降低模型复杂性

$$\min \ell(\mathbf{w}, b) \text{ subject to } \|\mathbf{w}\|^2 \leq \theta$$

- 通常不限制偏差参数 b
- 做或不做在实践中几乎没有区别
- 小 θ 意味着更多的正则化



平方正则化作为“软”约束

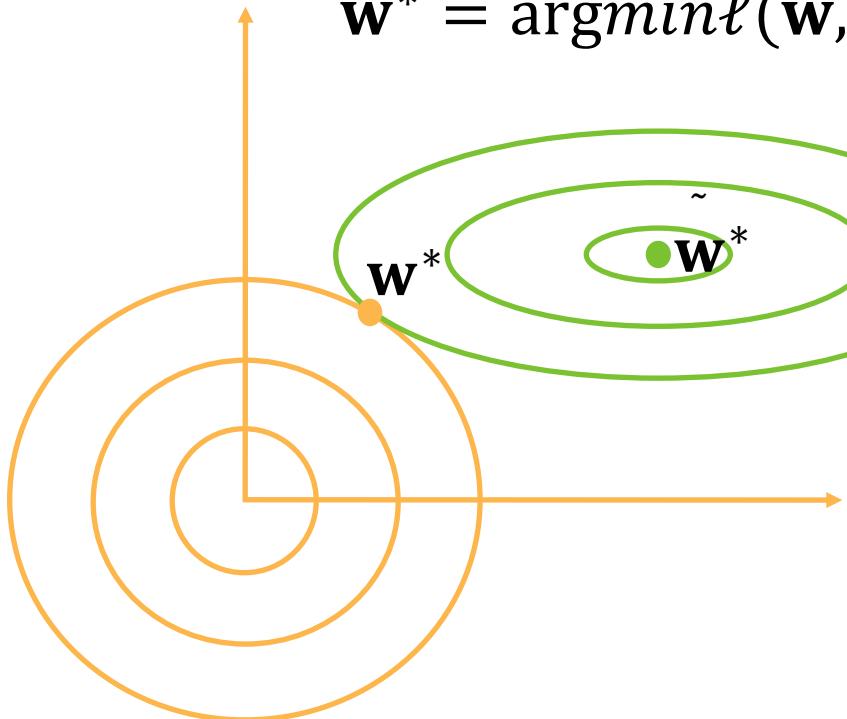
- 对于每一个 θ , 我们都可以找到 λ 将硬约束版本重写为

$$\min \ell(\mathbf{w}, b) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

- 可以用拉格朗日乘数法证明
- 超参数 λ 控制正则化的重要性
- $\lambda = 0$: 没有效果

$$\lambda \rightarrow \infty, \mathbf{w}^* \rightarrow \mathbf{0}$$

图解说明最优解



$$\mathbf{w}^* = \operatorname{argmin} \ell(\mathbf{w}, b) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

$$\tilde{\mathbf{w}}^* = \operatorname{argmin} \ell(\mathbf{w}, b)$$

更新规则

- 计算梯度
- 在时刻 t 更新权重

$$\frac{\partial}{\partial \mathbf{w}} \left(\ell(\mathbf{w}, b) + \frac{\lambda}{2} \| \mathbf{w} \|^2 \right) = \frac{\partial \ell(\mathbf{w}, b)}{\partial \mathbf{w}} + \lambda \mathbf{w}$$

- 通常 $\eta \lambda < 1$, 在深度学习中也称为权重衰减

$$\mathbf{w}_{t+1} = (1 - \eta \lambda) \mathbf{w}_t - \eta \frac{\partial \ell(\mathbf{w}_t, b_t)}{\partial \mathbf{w}_t}$$

丢弃法 (dropout)



动机

- 在输入的适度变化下，一个好的模型应该是稳定的
 - 用输入噪声训练相当于 Tikhonov 正则化
 - 丢弃法：将噪音注入内部隐藏层



添加没有偏差的噪音

- 将噪声添加到 x 中以获得 x' :

$$E[x'] = x$$

- 丢弃法将噪声添加到每个元素:

$$x'_i = \begin{cases} 0 & \text{with probability } p \\ \frac{x_i}{1-p} & \text{otherwise} \end{cases}$$

丢弃法 – 训练

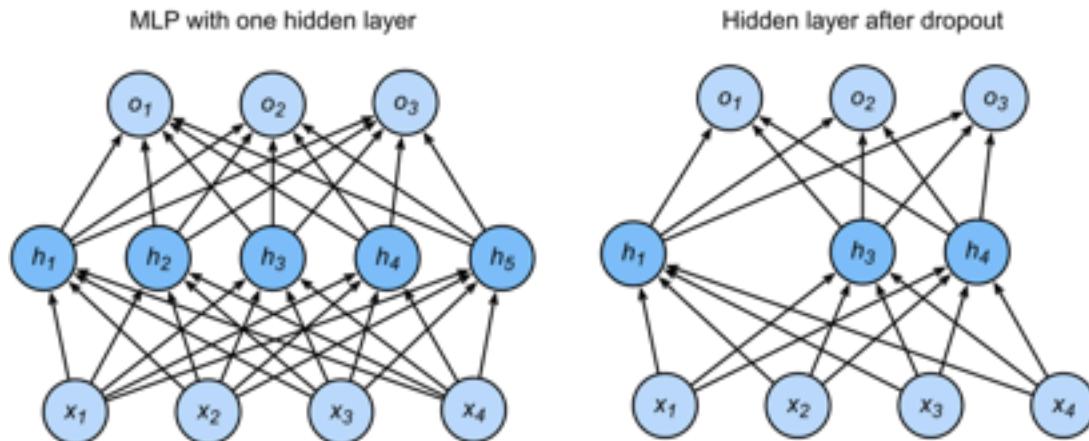
- 通常在全连接层的输出上使用丢弃法

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

$$\mathbf{o} = \mathbf{W}_2 \mathbf{h}' + \mathbf{b}_2$$

$$\mathbf{y} = \text{softmax}(\mathbf{o})$$



丢弃法 – 推断

- 正规化仅用于模型训练
- 丢弃法在推断中用于：

$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

- 保证确定性结果

总结

- 模型选择
 - 训练误差和泛化误差
 - K 折交叉验证
- 欠拟合与过拟合
 - 模型复杂度
 - VC 维度
- 权重衰减
- 丢弃法

动手学深度学习

8. 数值稳定性, 激活函数 和 硬件

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/dropout.html>

概要

- 数值稳定性
 - 梯度爆炸
 - 梯度消失
- 稳定模型训练
 - 权重初始化
- 激活函数
- 硬件

数值稳定性



Wikipedia

神经网络的梯度

- 考虑具有 d 层的神经网络

$$\begin{aligned}\mathbf{h}^t &= f_t(\mathbf{h}^{t-1}) \\ y &= \ell \circ f_d \circ \cdots \circ f_1(\mathbf{x})\end{aligned}$$

- 计算损失 ℓ 的梯度 \mathbf{W}_t

$$\frac{\partial \ell}{\partial \mathbf{W}^t} = \frac{\partial \ell}{\partial \mathbf{h}^d} \frac{\partial \mathbf{h}^d}{\partial \mathbf{h}^{d-1}} \cdots \underbrace{\frac{\partial \mathbf{h}^{t+1}}{\partial \mathbf{h}^t} \frac{\partial \mathbf{h}^t}{\partial \mathbf{W}^t}}_{d-t \text{ 矩阵相乘}}$$

深度神经网络的两个问题

梯度爆炸



梯度消失



$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i}$$

$$1.5^{100} \approx 4 \times 10^{17}$$

$$0.8^{100} \approx 2 \times 10^{-10}$$

例子： MLP

- 假设现有一个MLP (为简单而没有偏差参数)

$$f_t(\mathbf{h}^{t-1}) = \sigma(\mathbf{W}^t \mathbf{h}^{t-1}) \quad \sigma \text{ 是一个激活函数}$$

$$\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = diag(\sigma'(\mathbf{W}^t \mathbf{h}^{t-1})) (\mathbf{W}^t)^T \quad \sigma' \text{ 是 } \sigma \text{ 的导函数}$$

$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} diag(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})) (\mathbf{W}^i)^T$$

梯度爆炸

- 使用 ReLU 作为激活函数

$$\sigma(x) = \max(0, x) \rightarrow \sigma'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

- $\prod_{i=t}^{d-1} (W^i)^T$ 为来自 $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})) (W^i)^T$ 的元素
 - 当 d_t 很大时，可产生很大的值 $1.5^{100} \approx 4 \times 10^{17}$

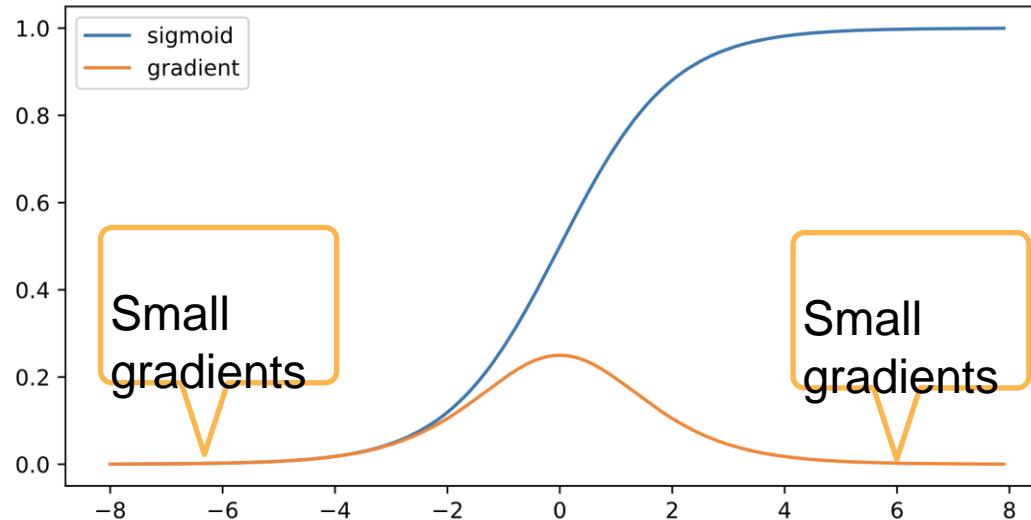
梯度爆炸的问题

- 梯度值超出范围：无穷大值
 - 严重的使用 16 位浮点
 - 范围：[6e-5 , 6e4]
- 对学习率 (LR) 敏感
 - 不够小的 LR -> 更大的权重 -> 更大的梯度
 - 太小的 LR -> 模型训练没有进展
 - 可能需要在训练期间大幅改变 LR

梯度消失

- 使用sigmoid作为激活函数

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$



梯度消失

- 使用sigmoid作为激活函数

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$

- $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})) (\mathbf{W}^i)^T$ 是 t 个较小值的乘积

例如 $0.8^{100} \approx 2 \times 10^{-10}$

梯度消失的问题

- 梯度值趋近为0的渐变
 - 16位浮点 (梯度值小于 $2^{-24} \approx 5.96 \times 10^{-8}$ 即为0)
- 训练没有进展
 - 无论如何选择学习率 (LR)
- 底层训练基本无效
 - 只有顶层训练有效
 - 使网络更深可能并没有更好

稳定模型训练

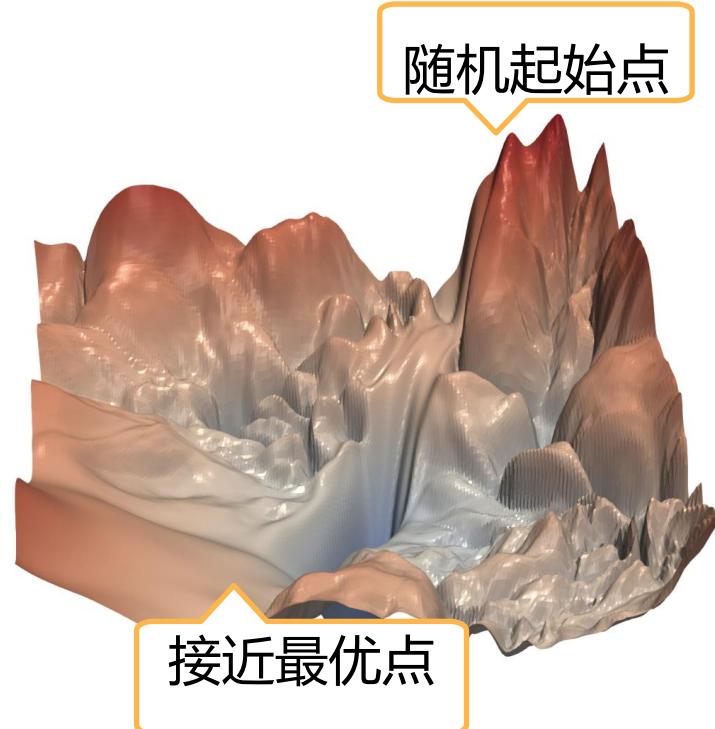


稳定模型训练

- 目标：确保渐变值在适当的范围内
 - 例如 在 $[1e-6, 1e3]$ 之间
- 改变神经网络框架结构（乘法 -> 加）
 - ResNet, LSTM
- 归一化
 - 批量归一化，渐变修剪
- 适当的权重初始化和激活函数

权重初始化

- 使用适当范围内的随机值初始化权重
- 训练的开始容易受到数值不稳定性的
影响
 - 远离最优点的表面可能很复杂
 - 接近最优点的表面可能更平坦
- 根据 $\mathcal{N}(0,0.01)$ 初始化的小网络很有效，
但不保证对深度神经网络也有效



每层神经网络的常数方差

- 将每层的输出和梯度看成随机变量
- 使每层的输出的均值和方差相同，类似于梯度

正向

$$\begin{aligned} \mathbb{E}[h_i^t] &= 0 & \mathbb{E}\left[\frac{\partial \ell}{\partial h_i^t}\right] &= 0 & \text{Var}\left[\frac{\partial \ell}{\partial h_i^t}\right] &= b & \forall i, t \\ \text{Var}[h_i^t] &= a \end{aligned}$$

a 和 b 都是常数

例子： MLP

假设：

- *i.i.d* $w_{i,j}^t$, $\mathbb{E}[w_{i,j}^t] = 0$, $\text{Var}[w_{i,j}^t] = \gamma_t$
- h_i^{t-1} 与 $w_{i,j}^t$ 是独立的
- 激活：用 $\mathbf{h}^t = \mathbf{W}^t \mathbf{h}^{t-1}$, $\mathbf{W}^t \in \mathbb{R}^{n_t \times n_{t-1}}$

$$\mathbb{E}[h_i^t] = \mathbb{E} \left[\sum_j w_{i,j}^t h_j^{t-1} \right] = \sum_j \mathbb{E}[w_{i,j}^t] \mathbb{E}[h_j^{t-1}] = 0$$

正向方差

$$\begin{aligned}\text{Var}[h_i^t] &= \mathbb{E}[(h_i^t)^2] - \mathbb{E}[h_i^t]^2 = \mathbb{E} \left[\left(\sum_j w_{i,j}^t h_j^{t-1} \right)^2 \right] \\ &= \mathbb{E} \left[\sum_j \left(w_{i,j}^t \right)^2 \left(h_j^{t-1} \right)^2 + \sum_{j \neq k} w_{i,j}^t w_{i,k}^t h_j^{t-1} h_k^{t-1} \right] \\ &= \sum_j \mathbb{E} \left[\left(w_{i,j}^t \right)^2 \right] \mathbb{E} \left[\left(h_j^{t-1} \right)^2 \right] \\ &= \sum_j \text{Var}[w_{i,j}^t] \text{Var}[h_j^{t-1}] = n_{t-1} \gamma_t \text{Var}[h_j^{t-1}]\end{aligned}$$


反向均值和方差

- 同正向分析：

$$\frac{\partial \ell}{\partial \mathbf{h}^{t-1}} = \frac{\partial \ell}{\partial \mathbf{h}^t} \mathbf{W}^t \quad \Rightarrow \quad \left(\frac{\partial \ell}{\partial \mathbf{h}^{t-1}} \right)^T = (\mathbf{W}^t)^T \left(\frac{\partial \ell}{\partial \mathbf{h}^t} \right)^T$$

$$\mathbb{E} \left[\frac{\partial \ell}{\partial h_i^{t-1}} \right] = 0$$

$$\text{Var} \left[\frac{\partial \ell}{\partial h_i^{t-1}} \right] = n_t \gamma_t \text{Var} \left[\frac{\partial \ell}{\partial h_j^t} \right] \quad \Rightarrow \quad n_t \gamma_t = 1$$

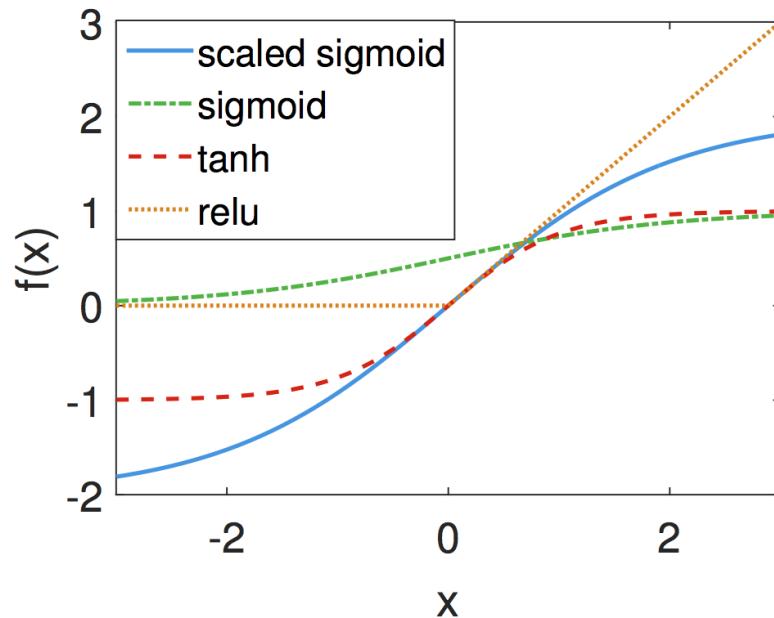
Xavier 初始化

- 满足 $n_{t-1}\gamma_t = 1$ 和 $n_t\gamma_t = 1$ 是冲突的
- Xavier 初始化

$$\gamma_t(n_{t-1} + n_t)/2 = 1 \rightarrow \gamma_t = 2/(n_{t-1} + n_t)$$

- 正态分布 $\mathcal{N}(0, \sqrt{2/(n_{t-1} + n_t)})$
- 统一分配 $u(-\sqrt{6/(n_{t-1} + n_t)}, \sqrt{6/(n_{t-1} + n_t)})$
 - $u[-a, a]$ 方差是 $a^2/3$
- 适应权重形状，特别是在 n_t 变化时

激活函数



简单的线性激活函数

- 假设 $\sigma(x) = \alpha x + \beta$ $\mathbf{h}' = \mathbf{W}^t \mathbf{h}^{t-1}$ and $\mathbf{h}^t = \sigma(\mathbf{h}')$

$$\mathbb{E}[h_i^t] = \mathbb{E} [\alpha h'_i + \beta] = \beta$$



$$\beta = 0$$

$$\text{Var}[h_i^t] = \mathbb{E}[(h_i^t)^2] - \mathbb{E}[h_i^t]^2$$



$$\alpha = 1$$

$$\begin{aligned} &= \mathbb{E}[(\alpha h'_i + \beta)^2] - \beta^2 \\ &= \mathbb{E}[\alpha^2(h'_i)^2 + 2\alpha\beta h'_i + \beta^2] - \beta^2 \end{aligned}$$

$$= \alpha^2 \text{Var}[h'_i]$$

反向

- 假设

$$\sigma(x) = \alpha x + \beta \quad \frac{\partial \ell}{\partial \mathbf{h}'} = \frac{\partial \ell}{\partial \mathbf{h}^t} (W^t)^T \quad and \quad \frac{\partial \ell}{\partial \mathbf{h}^{t-1}} = \alpha \frac{\partial \ell}{\partial \mathbf{h}'}$$

$$\mathbb{E} \left[\frac{\partial \ell}{\partial h_i^{t-1}} \right] = 0$$

 $\beta = 0$

$$\text{Var} \left[\frac{\partial \ell}{\partial h_i^{t-1}} \right] = \alpha^2 \text{Var} \left[\frac{\partial \ell}{\partial h_j'} \right]$$
 $\alpha = 1$

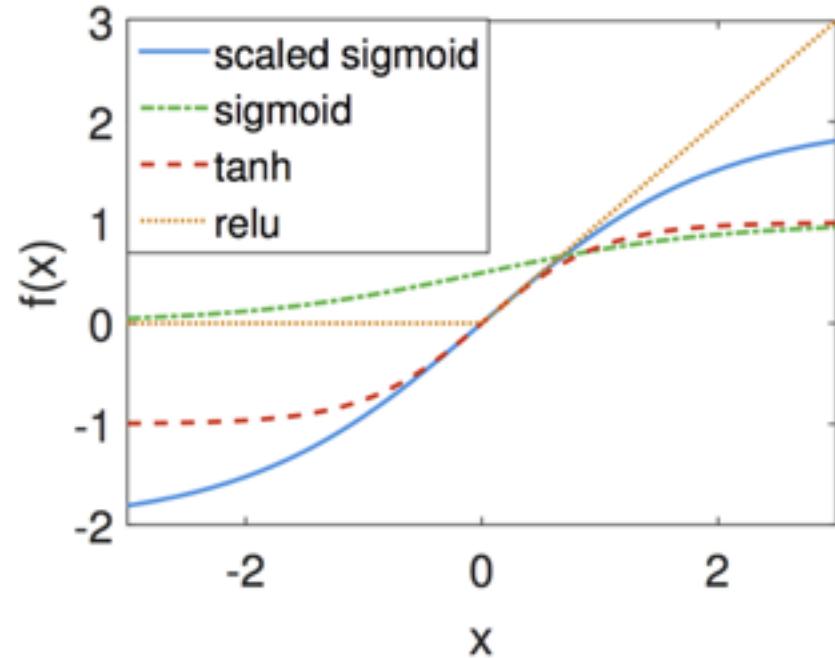
回顾激活函数

- 通过 Taylor 展开式

$$\text{sigmoid}(x) = \frac{1}{2} + \frac{x}{4} - \frac{x^3}{48} + O(x^5)$$

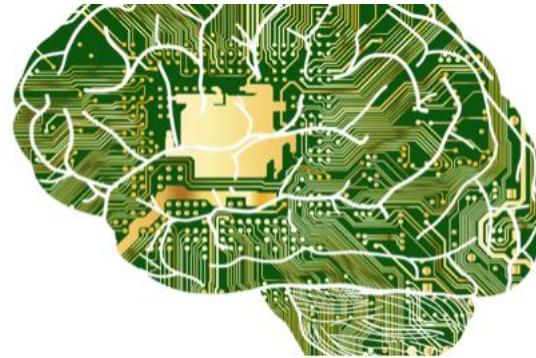
$$\tanh(x) = 0 + x - \frac{x^3}{3} + O(x^5)$$

$$\text{relu}(x) = 0 + x \quad \text{for } x \geq 0$$



- 用以下方法“纠正”sigmoid: $4 \times \text{sigmoid}(x) - 2$

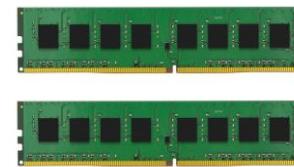
深度学习之硬件



(articleshub360.com)

GPU

Intel i7
0.15 TFLOPS

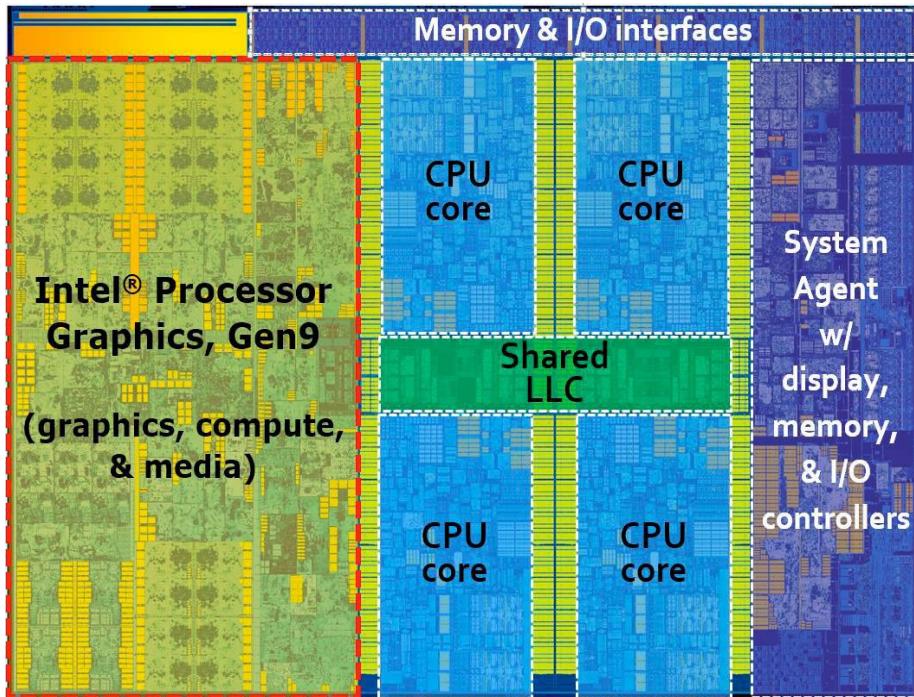


DDR4
32 GB



Nvidia Titan X
12 TFLOPS
16 GB

Intel i7-6700K



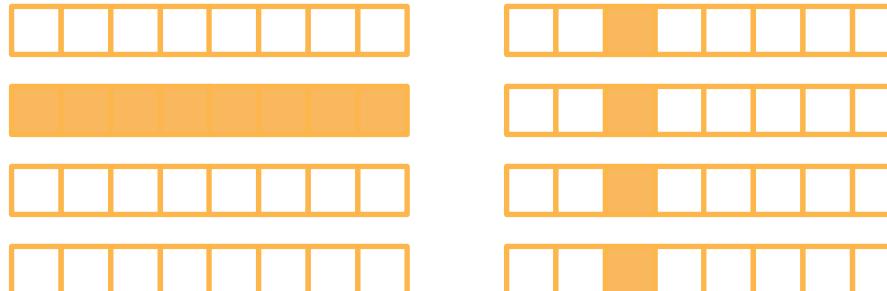
- 4 个物理核
- 每个核
 - 64 KB 一级 (L1) 缓存
 - 256 KB 二级 (L2) 缓存
- 共享 8 MB 三级 (L3) 缓存
- 主内存为 30 GB/s

提高CPU利用率 —

- 在计算 $a + b$ 之前，需要先准备数据
 - 主存 -> L3 -> L2 -> L1 -> 寄存器
 - L1 高速缓存参考时间：0.5 ns
 - L2 缓存参考时间 7 ns ($14 \times L1$)
 - 主存储器参考时间 100ns ($200 \times L1$)
- 改善时间和空间记忆局部性
 - 时间：重用数据，因此我们将它们保留在缓存中
 - 空间：读取数据顺序，以便我们可以预取数据

案例分析

- 对于存储在 raw-major 中的矩阵，访问每列比访问每行要慢
 - CPU每次读取64个字节（高速缓存行）
 - 在需要时，CPU“智能”提前读取下一个缓存行



提高CPU利用率 -- 二

- 服务器CPU可能有几十个核心
 - EC2 P3.16xlarge: 2个Intel Xeon CPU, 共32个物理内核
- 并行化以使用所有核心
 - 由于共享寄存器, 超线程 (Hyper-threading) 可能无济于事

案例分析

Case A: `for i in range(len(a)):`

$c[i] = a[i] + b[i]$

Case B: `c = a + b`

- Python 中 Case A 比右边 Case B 慢
- Case A 发出 n 次调用，每次调用都需要“开销”（例如几微秒）
- Case B 更容易被C ++运算符并行化

`#pragma omp for`

`for (i=0; i<a.size(); i++) {`

$c[i] = a[i] + b[i]$

`}`

Nvidia Titan X (Pascal)

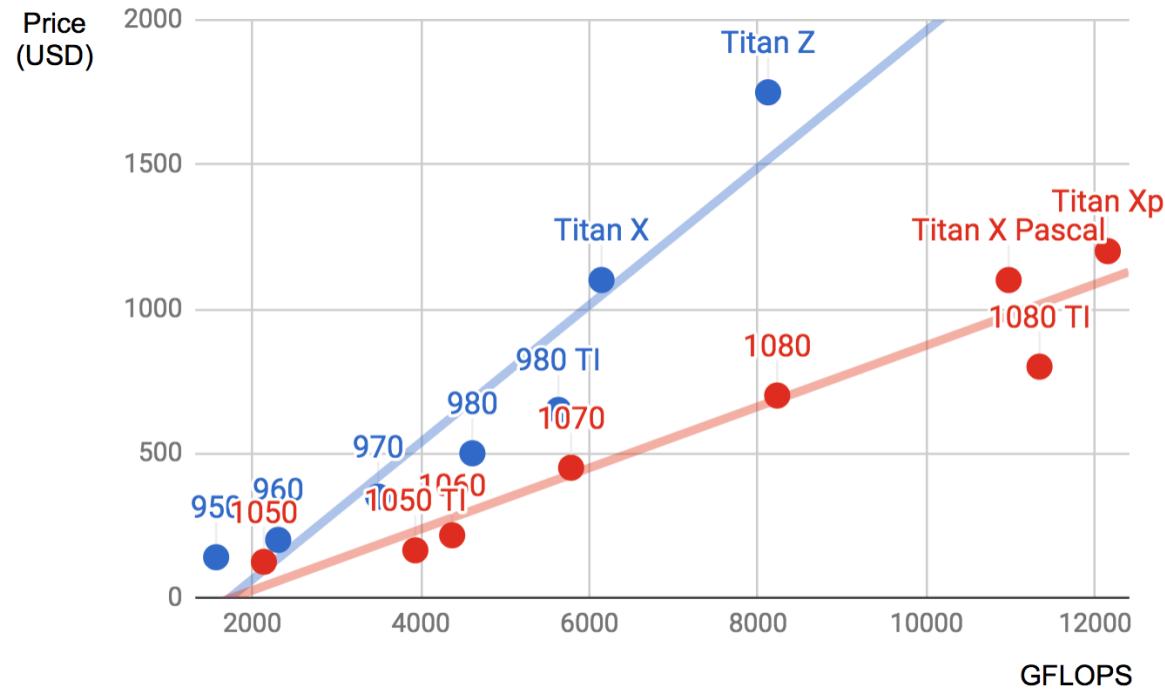


- 2584个核
- 每个核多个算术单位
- 3MB二级缓存
- 480 GB / s内存带宽

提高GPU利用率

- 并行
 - 使用数以千计的线程 (thread)
- 记忆局部性
 - 简单的缓存架构和小缓存容量
- 简单的流量调节器
 - 支持有限
 - 同步性较大

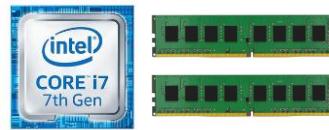
如何购买 GPUs



- 每个系列都改进了前一个系列
 - 购买最新系列
- 每一系列中：价格与功效成线性关系

CPU 与 GPU

Typical / High End

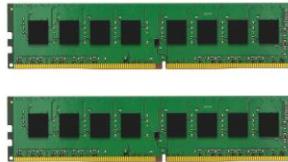


# 核	6 / 64	2K / 4K
每秒浮点运算百亿次数 (TFLOPS)	0.2 / 1	10 / 100
内存大小	32 GB / 1 TB	16 GB / 32 GB
内存带宽	30 GB/s / 100 GB/s	400 GB/s / 1 TB/s
控制流 (Control flow)	Strong	Weak

CPU/GPU 带宽



30 GB/s



PCIe 3.0 16x: 16 GB/s



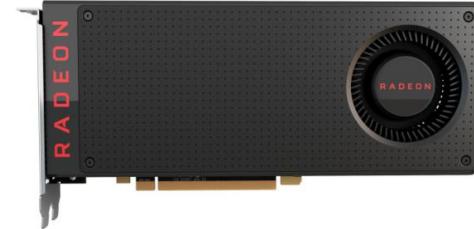
480 GB/s

- 不要经常在GPU和CPU之间复制数据

- 有限的PCIe带宽
- 分布式系统的同步开销

CPUs 与 GPUs

- CPU
 - 桌面 / 服务器: AMD
 - Edge: ARM
- GPU
 - 桌面 / 服务器 GPU: AMD, Intel
 - Edge: ARM, Qualcomm



CPU/GPU 上编程

- CPU: C++ 或任何其他高性能语言
 - 成熟的编译器，性能有保证
- GPU
 - Nvidia的CUDA
 - 丰富的功能，成熟的编译器和驱动程序
 - OpenCL
 - 不同芯片供应商的质量参差不齐

更有前景的硬件



Qualcomm Snapdragon 845

Snapdragon
X20 LTE modem

Wi-Fi

Hexagon 685 DSP

Qualcomm:
Aqstic Audio

System Memory

Adreno 630
Visual Processing
Subsystem

Qualcomm:
Spectra 280 ISP

Kryo 385 CPU

Qualcomm:
Secure Processing Unit

Touch

PMIC

Wi-Fi/BT/NFC

RFFE

Audio Codec

BT

NFC

数字信号处理器

- 专为数字处理算法而设计
 - 点积，卷积，FFT
- 低功耗和高性能
 - 比移动GPU快5倍，耗电量更少
- VLIW：非常长的指令字
 - 数百次乘法累加在一条指令中
- 难以编程和调试
 - 编译器工具链质量因芯片供应商而异

现场可编程门阵列 (FPGA)

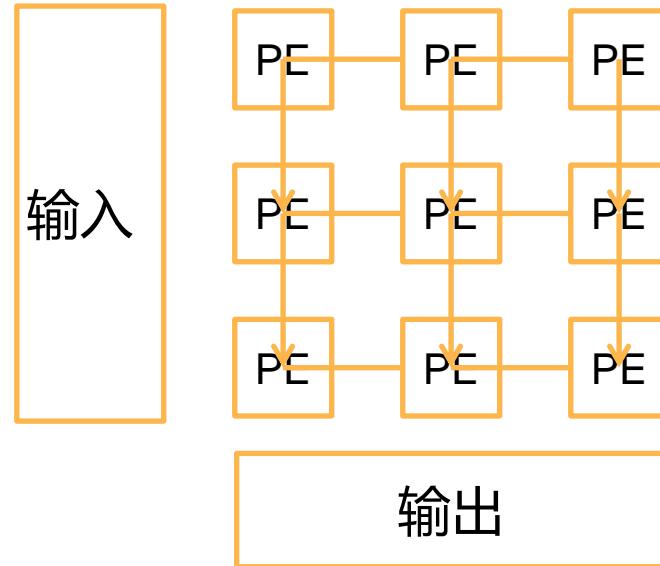
- 包含大量可编程逻辑块和可重配置互连
- 可以配置为执行复杂的功能
 - 常用语言：VHDL和Verilog
- 比通用硬件效率更高
- 工具链的质量各不相同
- 每次“编译”可能需要几个小时

AI ASIC

- 深度学习的热门话题
- 每家大公司都在制造他们的芯片（英特尔，高通，谷歌，亚马逊，Facebook，...）
- Google TPU 是具有里程碑意义的芯片
 - 匹配高端 Nvidia GPU 的性能，但便宜 10x-100x
 - 广泛部署在谷歌
 - 核心是一个收缩阵列

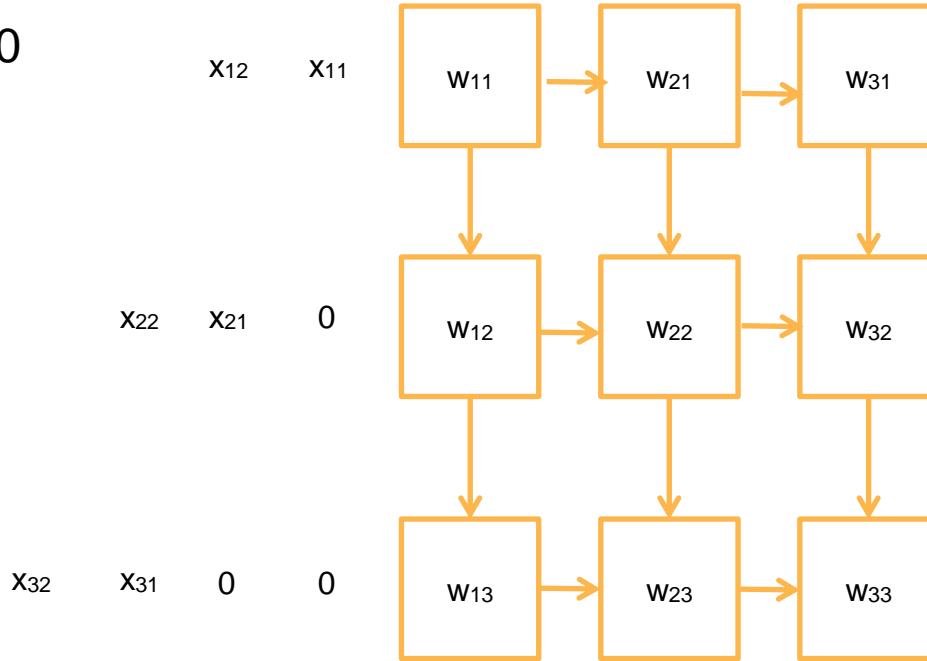
收缩阵列

- 一系列处理元素 (PE)
- 擅长执行“矩阵x矩阵”乘法
- 相对容易/更便宜的建设



具有脉动阵列的矩阵乘法

时间步 0



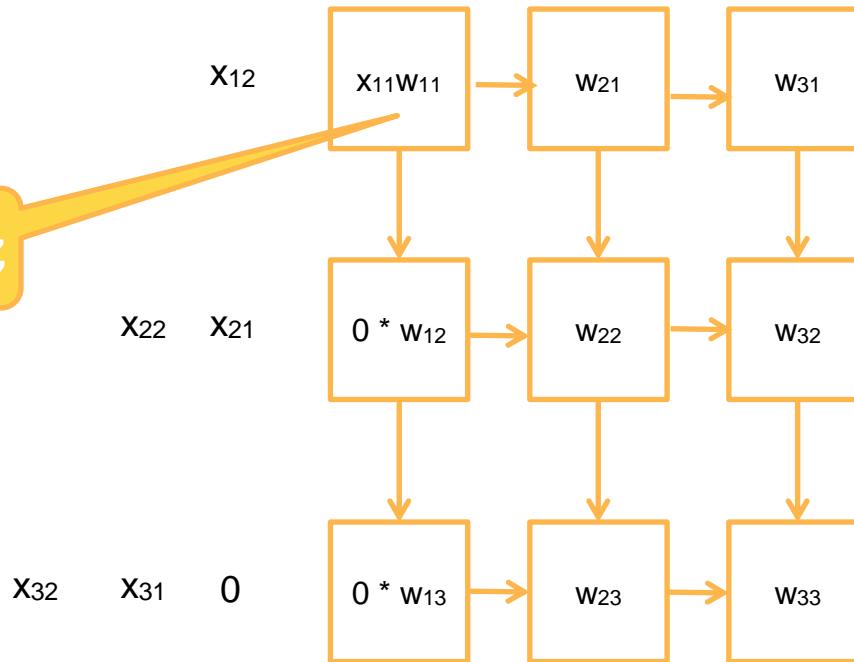
$$Y = WX$$

3x2 3x3 3x2

具有脉动阵列的矩阵乘法

时间步 1

输入左移



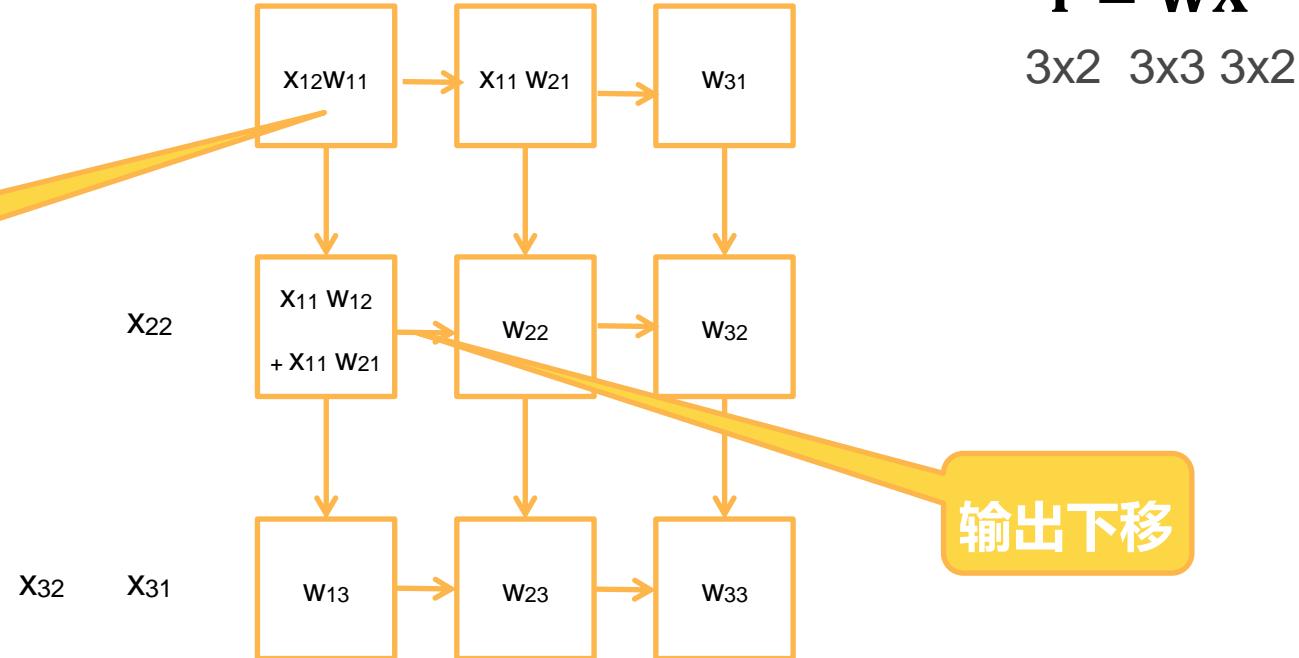
$$Y = WX$$

3x2 3x3 3x2

具有脉动阵列的矩阵乘法

时间步 2

输入左移



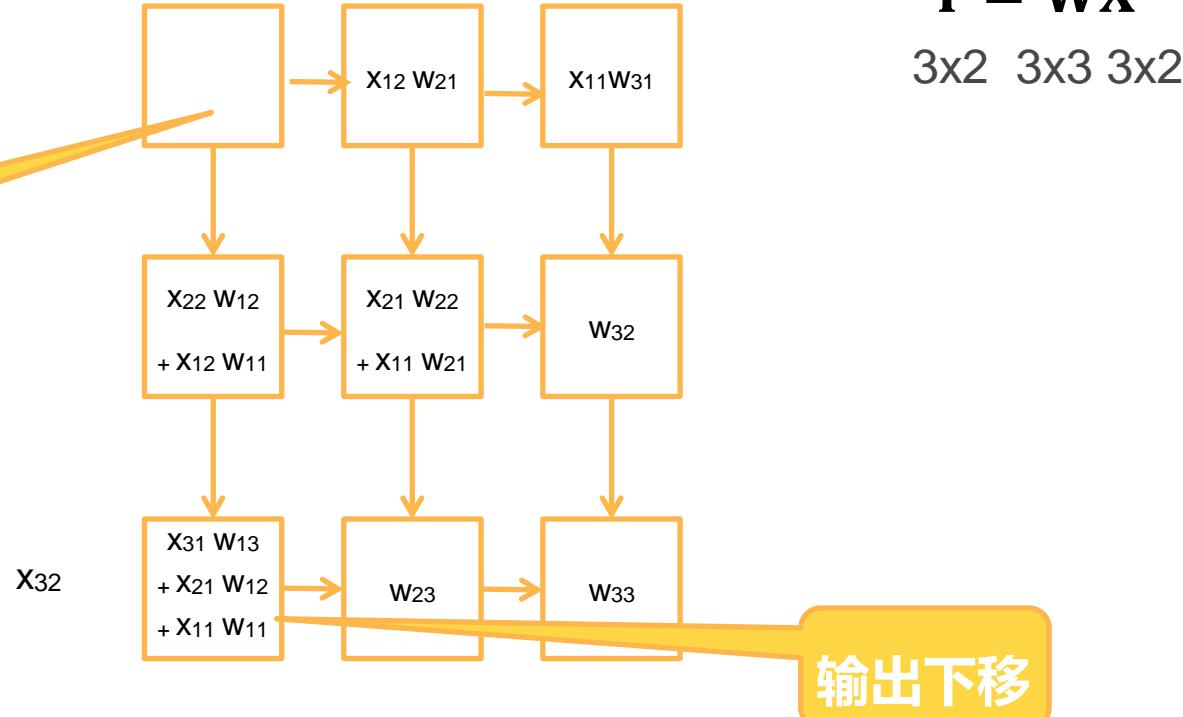
$$Y = WX$$

3x2 3x3 3x2

具有脉动阵列的矩阵乘法

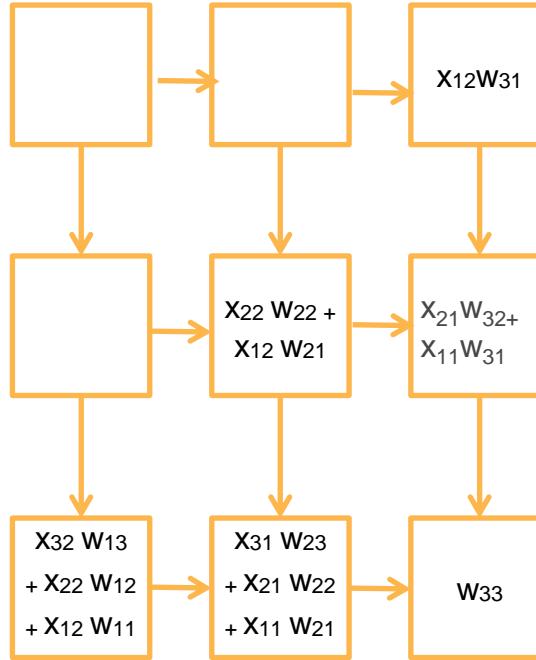
时间步 3

输入左移



具有脉动阵列的矩阵乘法

时间步 4

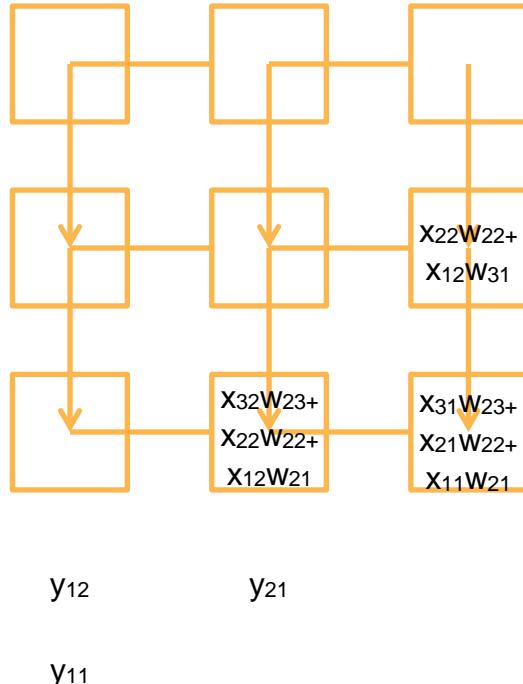


$$Y = WX$$
$$3 \times 2 \quad 3 \times 3 \quad 3 \times 2$$

y_{11}

具有脉动阵列的矩阵乘法

时间步 5

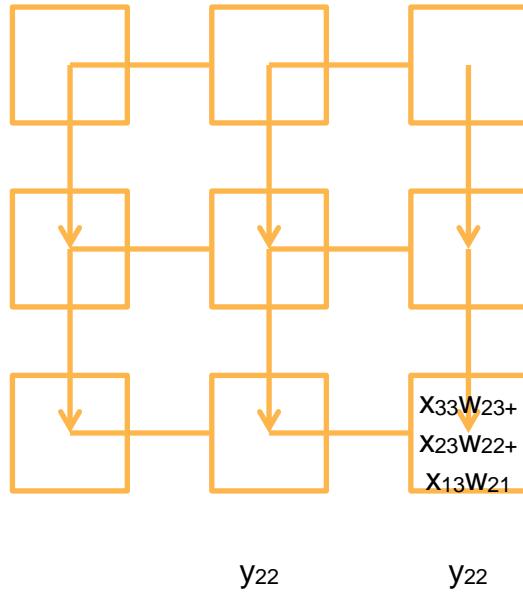


$$\mathbf{Y} = \mathbf{WX}$$

3x2 3x3 3x2

具有脉动阵列的矩阵乘法

时间步 6

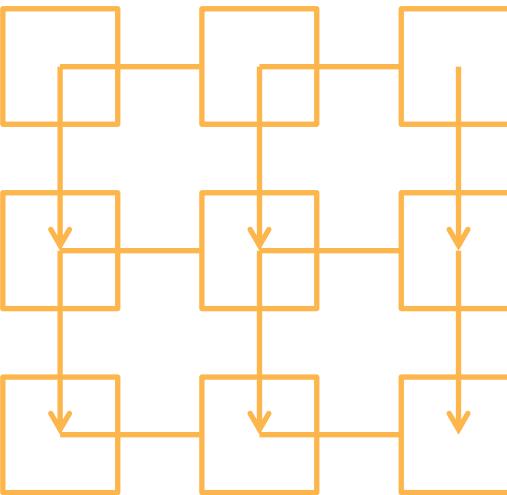


$$\mathbf{Y} = \mathbf{WX}$$

3x2 3x3 3x2

具有脉动阵列的矩阵乘法

时间步 7



y_{11}

y_{21}

y_{12}

y_{32}

y_{31}

y_{22}

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

3x2 3x3 3x2

脉动阵列 (Systolic Array)

- 对于通用尺寸矩阵乘法，切割和填充输入以匹配脉动阵列 (SA) 尺寸
- 批量输入可降低延迟成本
- ASIC 还有其他专用组件用于其他操作，例如sigmoid

灵活性和易用性



Hexagon 685 DSP



性能和功效

D2L.ai

总结

- 数值稳定性
 - 梯度爆炸
 - 梯度消失
- 稳定模型训练
 - 权重初始化
- 激活函数
- 硬件

动手学深度学习

9. 泛化表现，协变量偏移 和 对抗性数据

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/environment.html>



概要

- 泛化表现
- 协变量偏移 (Covariate Shift)
 - 标签偏移
 - 协变量偏移校正
- 对抗性数据
- 非平稳环境



训练 ≠ 测试

- 泛化表现 (经验分布)

$$p_{\text{emp}}(x, y) \neq p(x, y)$$

- 协变量偏移 (协变量分布)

$$p(x) \neq q(x)$$

- 逻辑回归 (修复偏移的工具)

$$\log(1 + \exp(-yf(x)))$$

- 协变量偏移校正

$$\frac{1}{2} (p(x)\delta(1, y) + q(x)\delta(-1, y))$$

- 标签转移 (标签分布所在)

$$p(y) \neq q(y)$$

- 非平稳环境

泛化表现

泛化表现

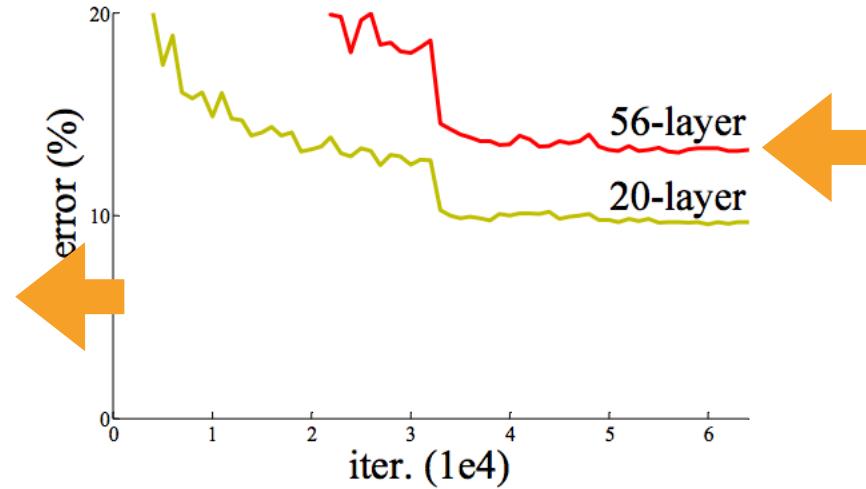
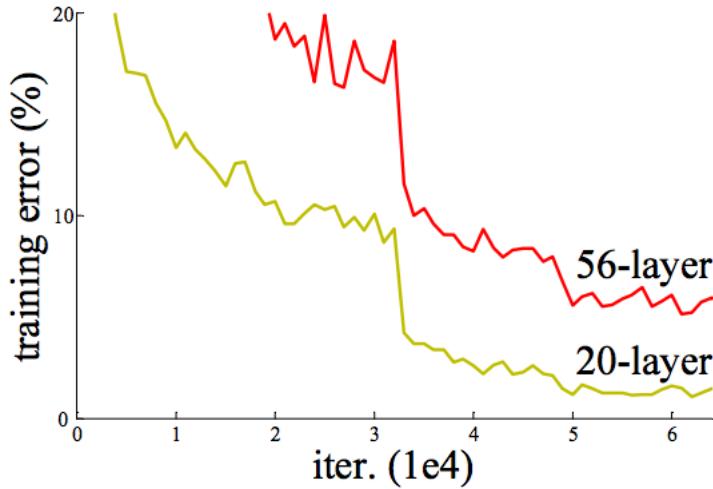


泛化表现



泛化表现

- 图像同理 (例如 He et al., 2015, ResNet paper)



- Alexa ('请关闭咖啡机'与'咖啡机关闭')

为什么呢？

- 数据分布 $p(x, y)$
- 从 $p(x, y)$ 绘制的数据集
- 训练可以最大限度地降低经验风险（加上正规化）

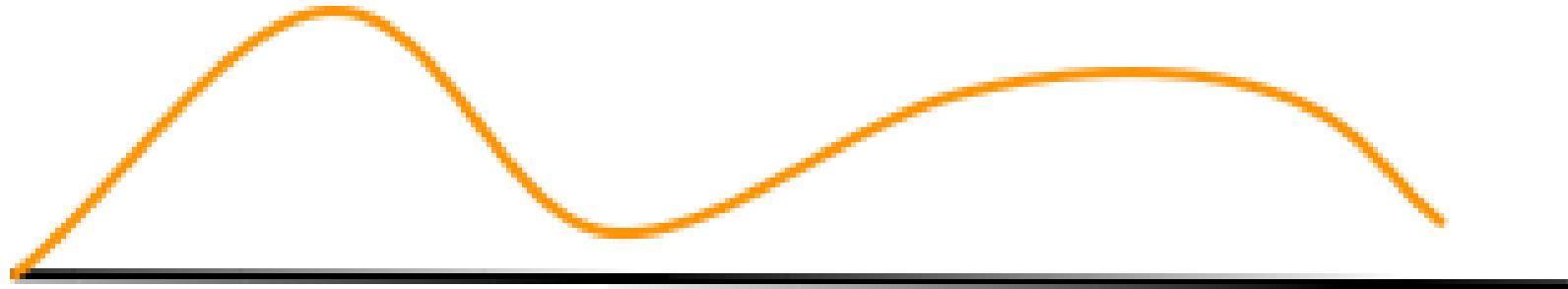
$$\underset{w}{\text{minimize}} \frac{1}{m} \sum_{I=1}^m l(f(x_i, w), y_i)$$

- 在测试时，预期风险很重要（根据所有已观测的其他数据）

$$\mathbf{E}_{(x,y) \sim p} [l(f(x, w), y)]$$

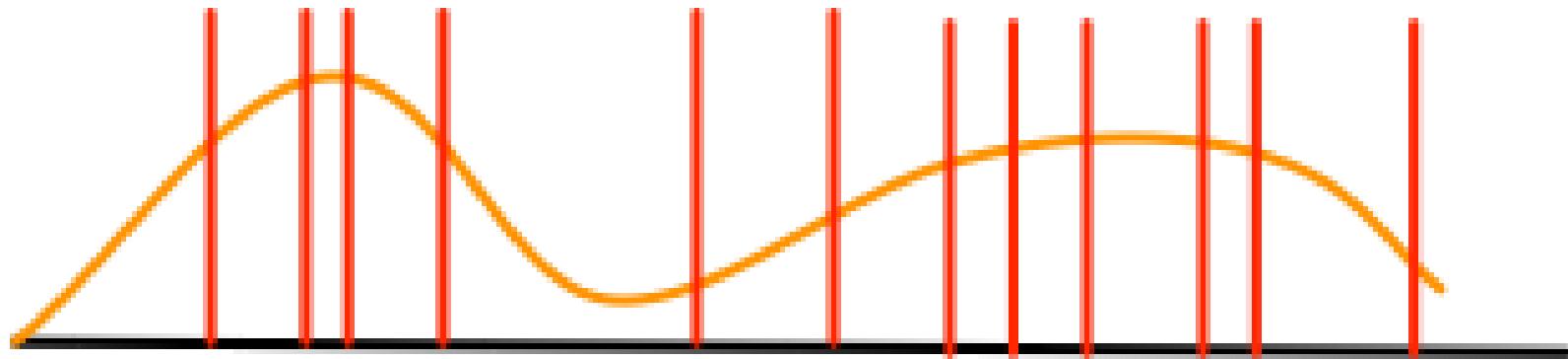
为什么呢？

数据分布



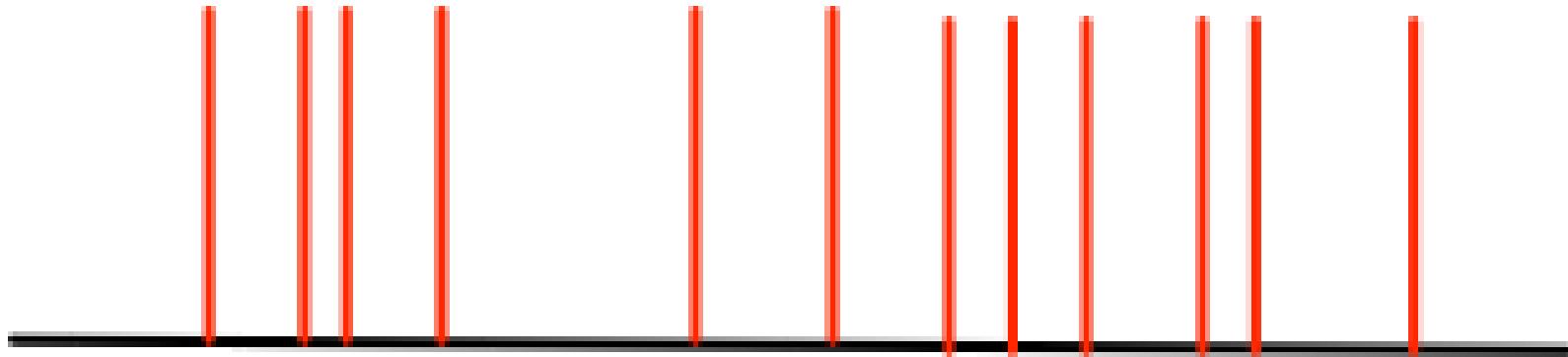
为什么呢？

数据分布与经验样本



为什么呢？

经验样本



怎么修复

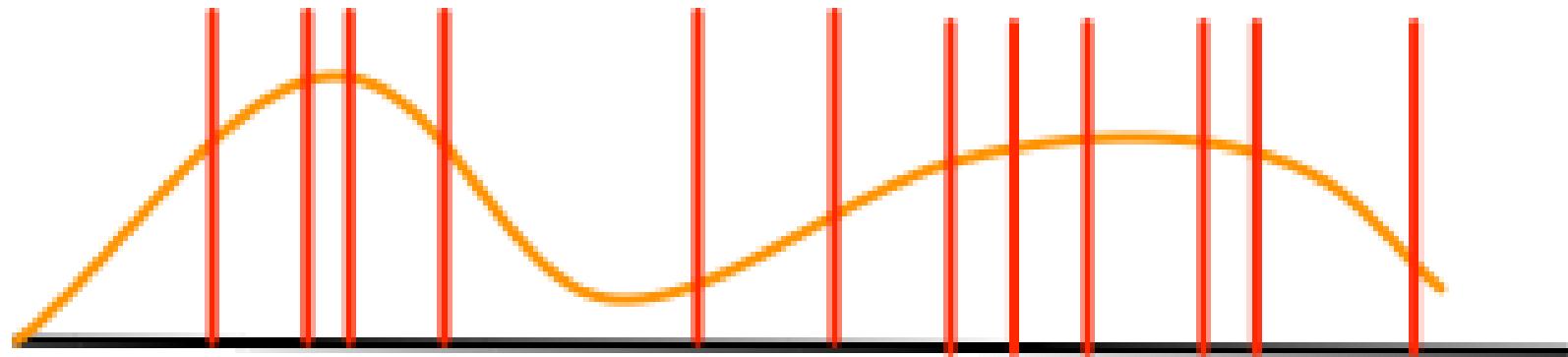
- 验证集（不用于训练的单独数据）
- 切尔诺夫届（Chernoff bound）
- 它为什么有效？

$$\Pr \left\{ \frac{1}{m} \sum_{I=1}^m l(f(x_i), y_i) - \mathbf{E} [l(f(x), y)] > \epsilon \right\} \leq \exp(-2m\epsilon^2)$$

- 验证集未用于训练（经常违反）
- 损失限制在 $[0, 1]$ 内（否则重新缩放）

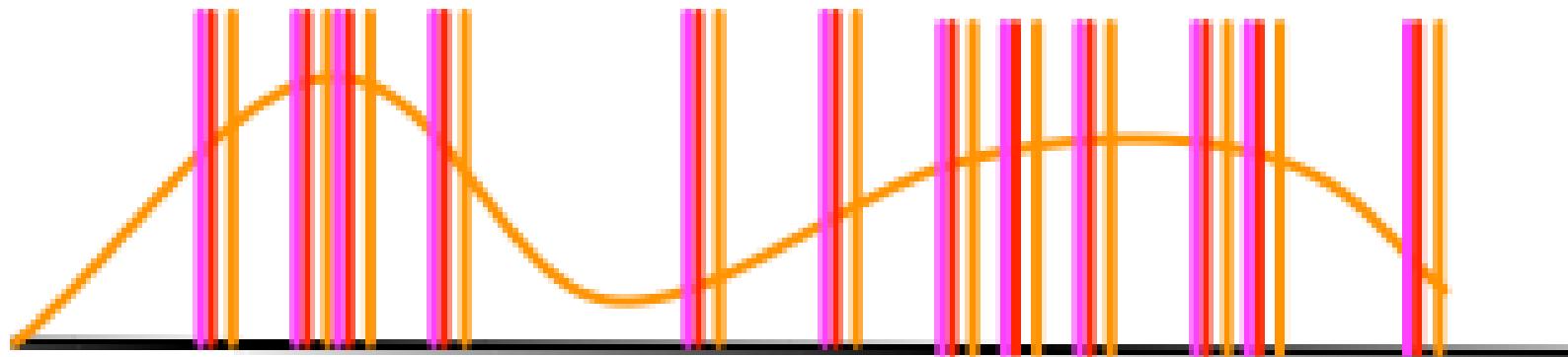
怎么修复

数据分布与经验样本



怎么修复

- 输入噪声（稍后会详细介绍）
- 丢弃法（层内噪声）
- 平滑函数 f （例如 权重衰减）



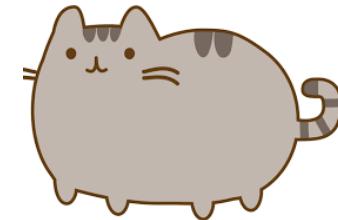
return

covariate shift

训练集



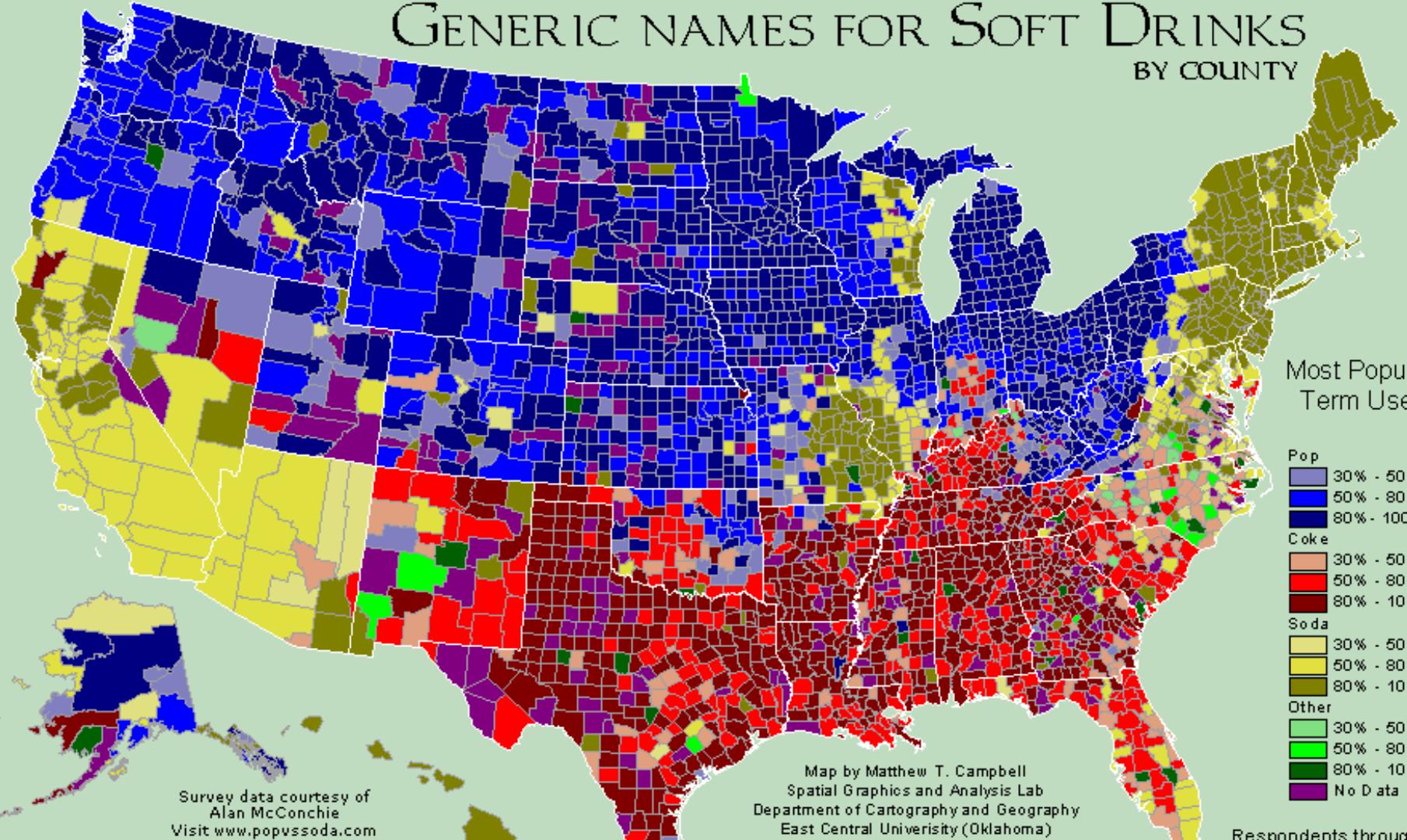
测试时...



协变量转变 (Covariate Shift)

- 网络搜索
 - 训练 - 美国市场的引擎页面相关数据
 - 测试 - 加拿大（或英国，澳大利亚）推荐引擎
- 语音识别
 - 训练 - 西海岸口音
 - 测试 - 南方drawl, 德克萨斯人, 非母语人士
- 语言
 - 训练 - '詹姆斯, 带给我一杯 soda'
 - 测试 - '约翰, 带给我一个'pop' (或可乐等)

GENERIC NAMES FOR SOFT DRINKS BY COUNTY



协变量转变 (Covariate Shift)

- 医疗
 - 训练 - 大学生+老年男性前列腺癌
 - 测试 - 可能生病的老人
- 强化学习
 - 训练 - 使用当前政策收集的数据
 - 测试 - 环境对更新的策略做出反应
- 数据库
 - 训练 - 数据库调整为2017年使用模式
 - 测试 - 2019年部署在AWS上的数据库

为什么呢?

$$q(x, y) = q(x)p(y|x)$$

- 训练损失

$$\underset{w}{\text{minimize}} \int dx p(x) \int dy p(y|x) l(f(x, w), y)$$

or rather $\underset{w}{\text{minimize}} \frac{1}{m} \sum_{I=1}^m l(f(x_i, w), y_i)$

- 测试损失不同

$$\int dx q(x) \int dy p(y|x) l(f(x, w), y)$$

测试集

协变量偏移校正

- 基本代数

$$\int dx q(x) f(x) = \int dx p(x) \underbrace{\frac{q(x)}{p(x)}}_{\alpha(x)} f(x) = \int dx p(x) \alpha(x) f(x)$$


密度比

- 需要找到密度比，但我们没有任何信息
- 直接估算 p 和 q 非常困难，需要专门的工具。我们可以重新利用分类器吗？

训练集



猫

猫



狗



狗



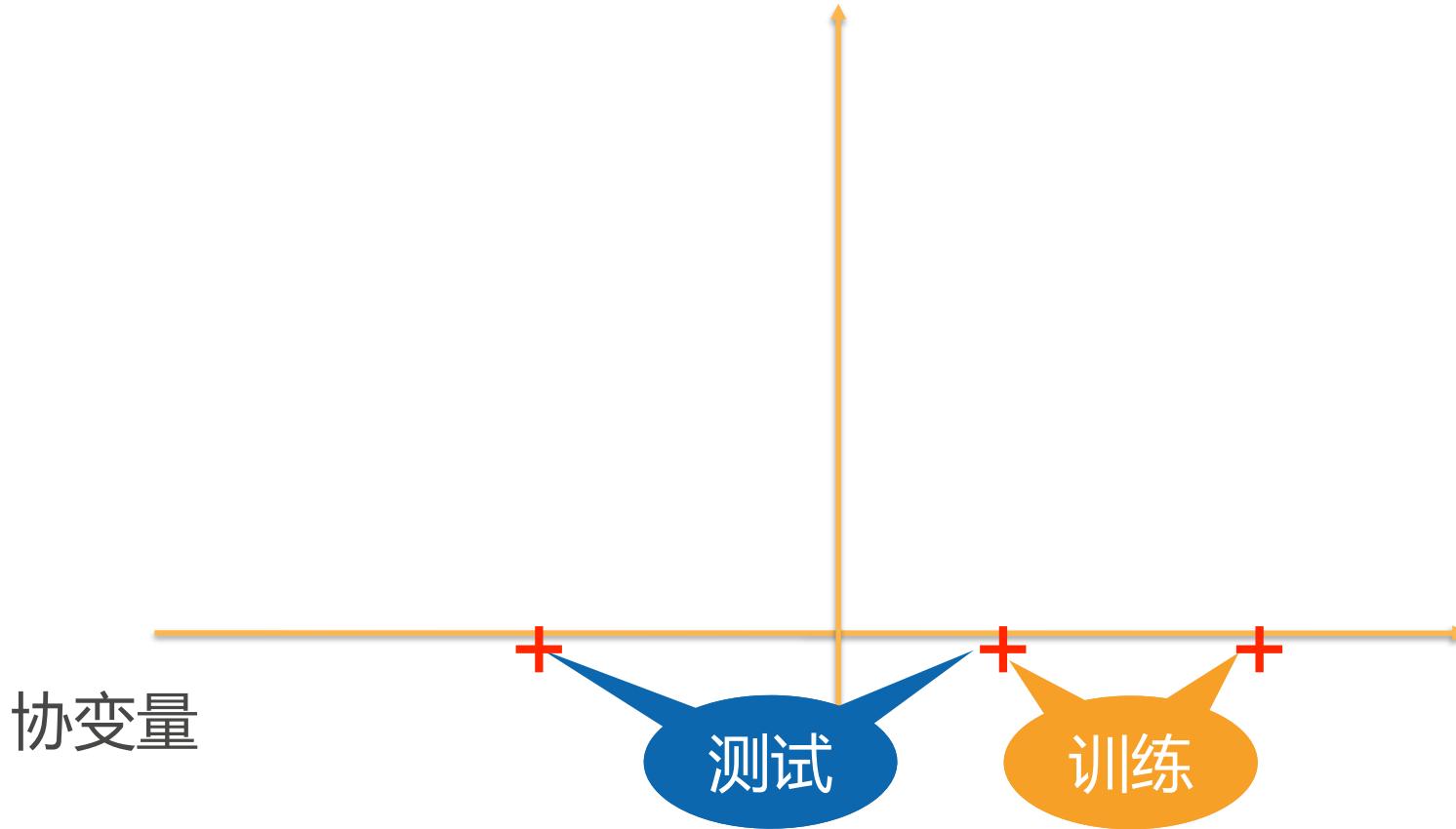
测试集



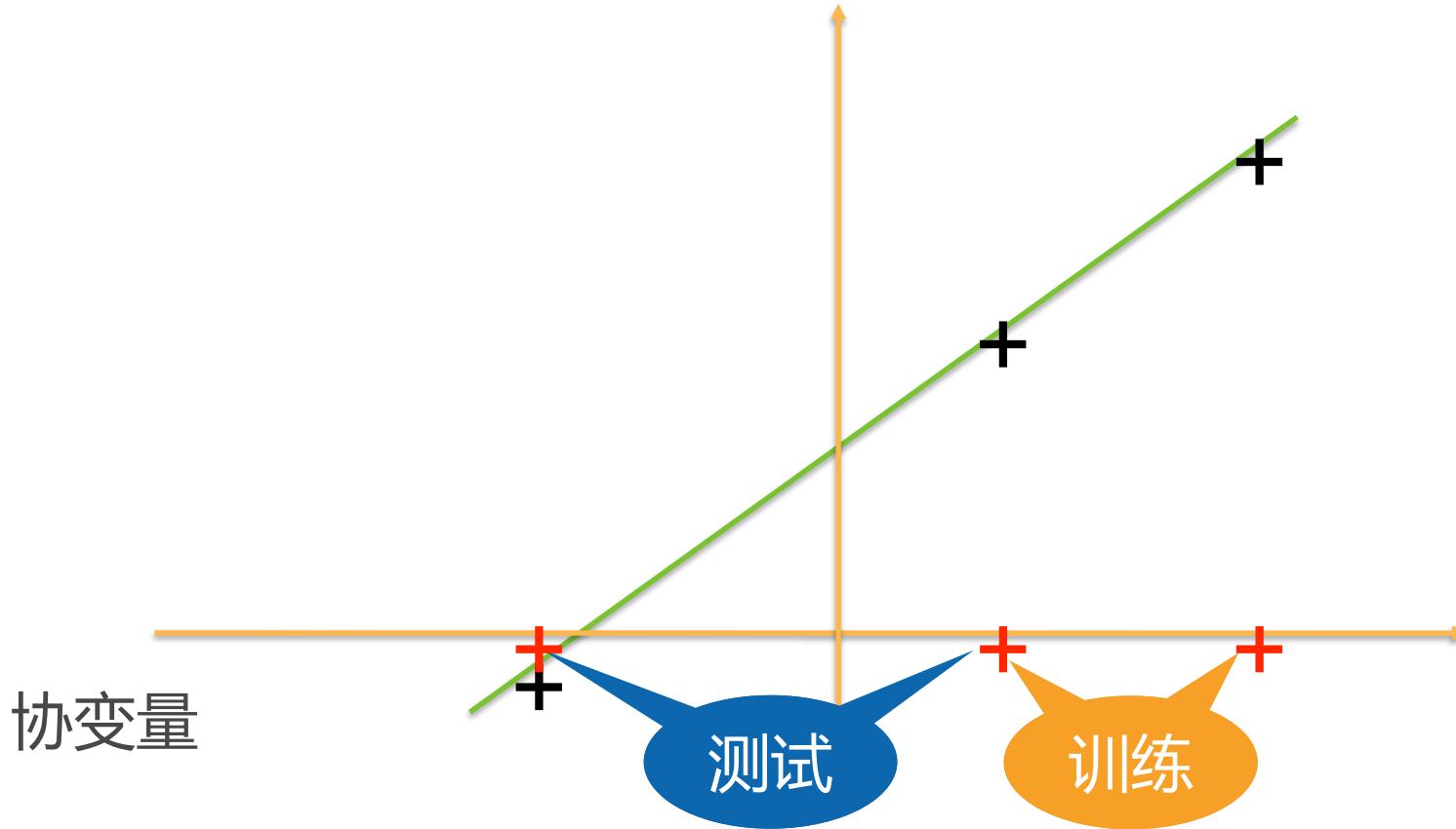
分类器在测试期间可能会执行得更差



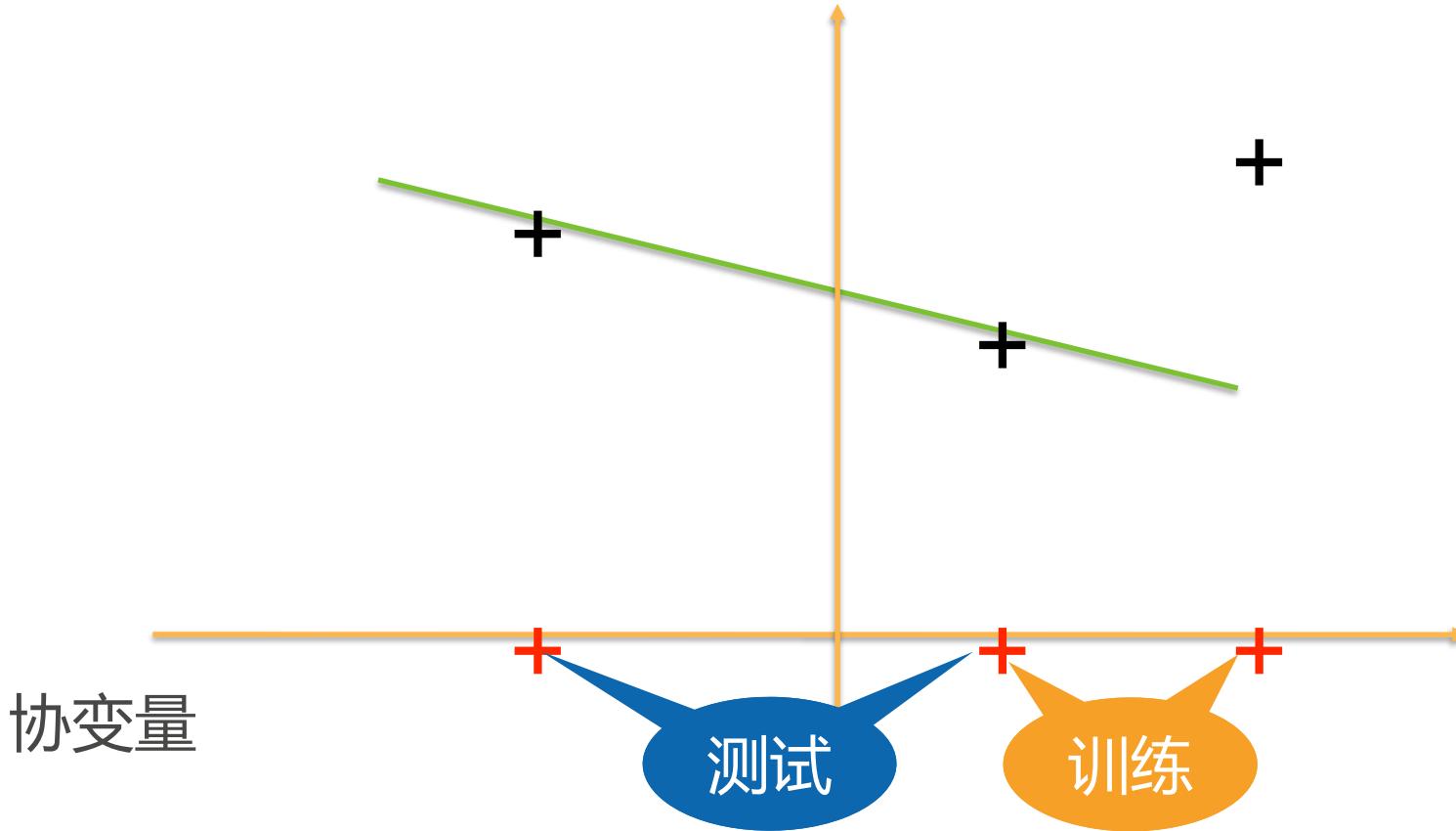
简单的回归问题



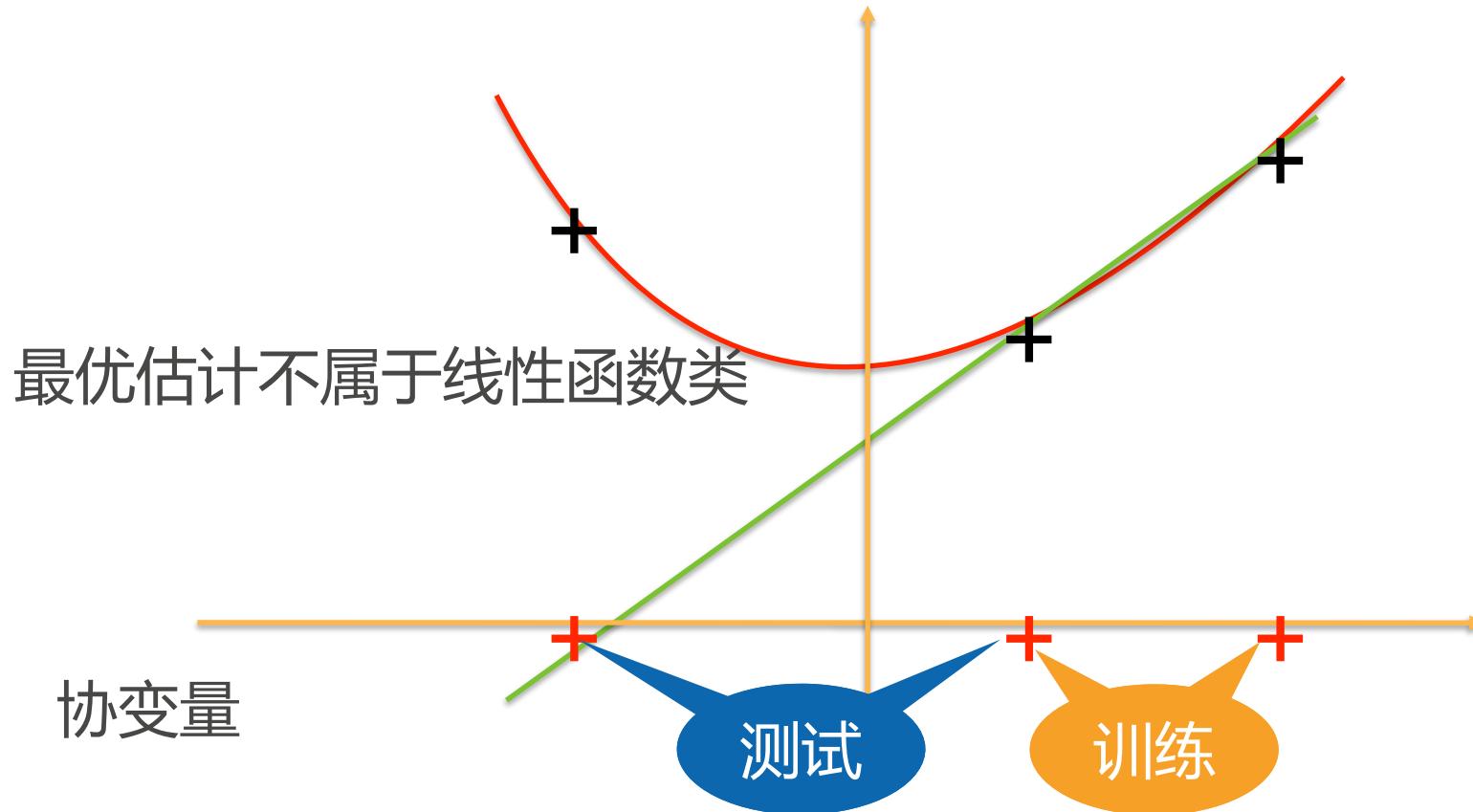
简单的回归问题



简单的回归问题



简单的回归问题



训练误差可能会误导（例如面孔）



>>

在IMDB上训练

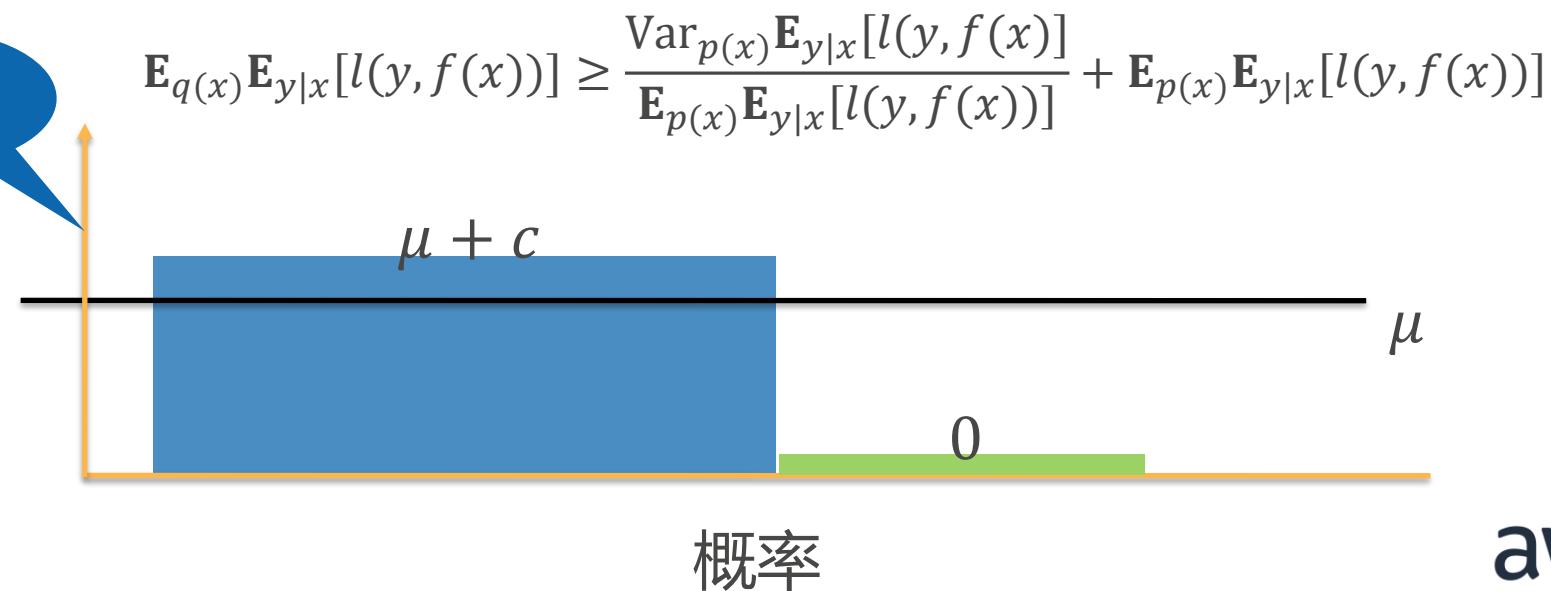


在奇怪的教授图片上测试

没有对偏差的预防

- 估算器对某些数据执行得更好（或更差）
- 我们总能找到更糟糕的分布 q

损失



TL; DR

**在我们没有足够数量的训练数据时
测试，可能会产生奇怪的结果。**



return

协变量偏移校正

covariate shift correction

协变量偏移校正

- 偏移计算

$$\int dx q(x) f(x) = \int dx p(x) \underbrace{\frac{q(x)}{p(x)}}_{\alpha(x)} f(x) = \int dx p(x) \alpha(x) f(x)$$


- 需要找到密度比，但我们没有任何一个。
- 关键思想：在p和q之间训练分类器

$$r(x, y) = \frac{1}{2} [p(x)\delta(y, 1) + q(x)\delta(y, -1)]$$

协变量偏移校正

- 条件类概率

$$r(y = 1|x) = \frac{p(x)}{p(x) + q(x)} \text{ and hence } \alpha = \frac{q(x)}{p(x)} = \frac{r(y = -1|x)}{r(y = 1|x)}$$

- 逻辑回归

$$r(y = 1 | x) = \frac{1}{1 + \exp(-f(x))}$$

$$\Rightarrow \alpha(x) = \frac{r(y = -1 | x)}{r(y = 1 | x)} = \exp(f(x))$$



协变量偏移校正

- 训练和测试数据
- 拆分是二进制分类问题（分别为训练和测试标记-1和1）
- 用逻辑回归训练得到 f
- 使用二进制分类器输出来重新加权数据
- 解决原始问题，但加权

$$\sum_i l(x_i, y_i, g(x_i, w)) \longrightarrow \sum_i \exp(f(x_i)) \cdot l(x_i, y_i, g(x_i, w))$$

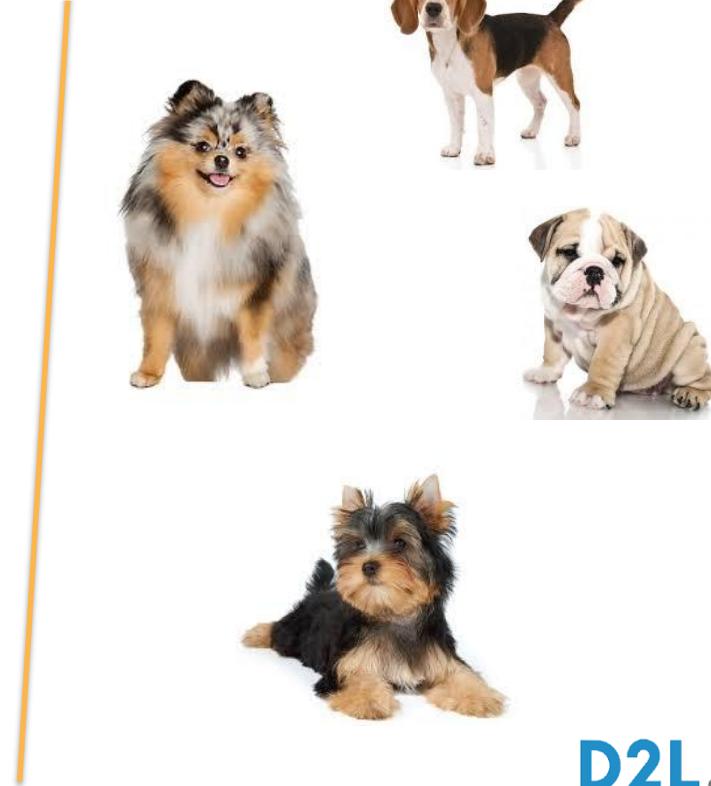


标签偏移

训练集



测试集



标签转移

- 医学诊断
 - 训练时几乎没有病人的数据
 - 流感季节期间的数据测试，其中 $q(\text{流感}) > p(\text{流感})$ ，而流感症状 $p(\text{症状}|\text{流感})$ 仍然相同
- 语音识别
 - 在选举前训练新闻广播数据
 - 选举后的新闻广播测试（新主题，名称，讨论，但仍然是同一种语言）

标签转移

$$q(x, y) = q(y)p(x|y)$$

- 数据生成过程 $p(x | y)$ 不变
- 标签变化 (因基础原因发生变化)

- 需要重新生成数据: $\beta(y) = \frac{q(y)}{p(y)}$

$$\int q(y) dy \int p(x|y) dx l(f(x), y) = \int p(y) \frac{q(y)}{p(y)} dy \int p(x|y) dx l(f(x), y)$$



我们没有 $q(y)$ 的采样 !

标签转移

$$q(x, y) = q(y)p(x|y)$$

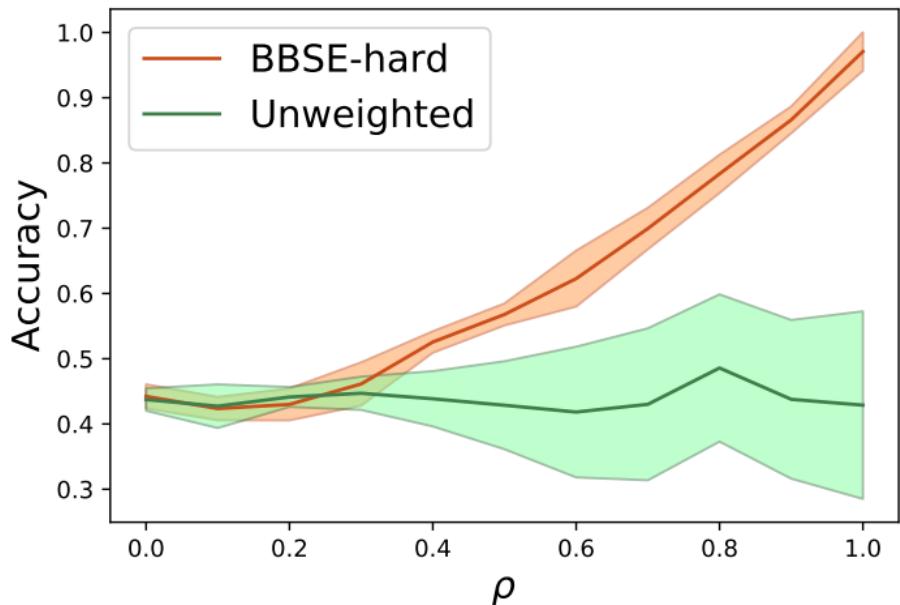
- **关键理念 - 衡量测试集的估计**
 - $p(x|y)$ 在训练和测试时相同
 - 基于 $p(\dots | x, y)$ 预测的分布必须相同
- **简单的“光谱”算法 (Lipton, Wang, Smola, 2018)**
 - 坚持混淆矩阵 $C[y'|y] = \Pr(\hat{y}(x) = y' | y)$
 - 在测试集上的预测标签矢量 $\mu[y'] = \Pr(\hat{y}(x) = y')$
 - 通过矩阵求逆得到 $q(y)$

$$\mu[y'] = \sum_y C[y'|y]q(y)$$

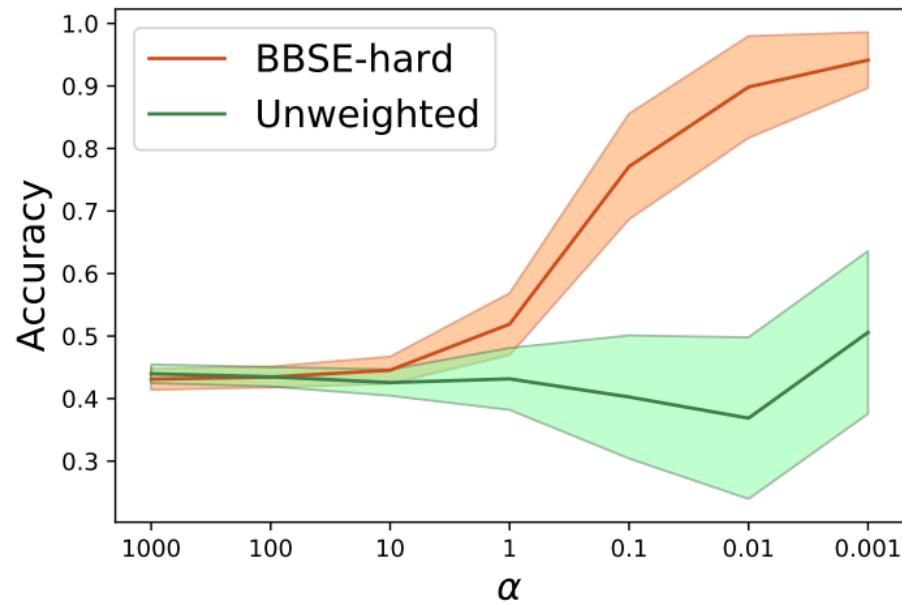
标签转移

- **假设错误稳定**
 - 即使估计 $y(x)$ 错误，也可以校准（保持和测试集上的相同错误）
 - 混淆矩阵和标签向量集中（使用矩阵Bernstein不等式）
- **算法复杂度**
 - 与类别数量的呈立方比
 - 与样本大小呈线性关系

CIFAR10上的黑盒协变量偏移校正



调整一个类概率

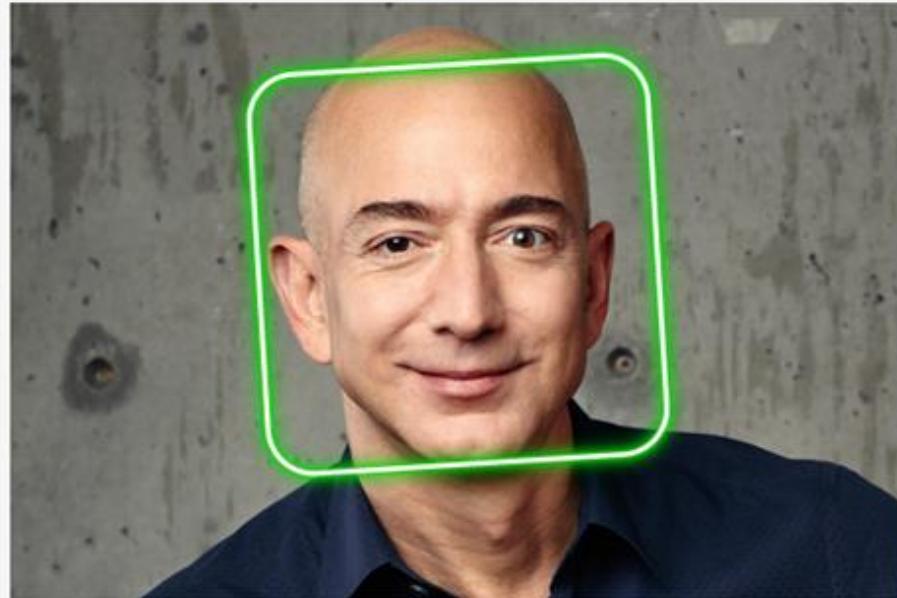


Dirichlet之前偏移

对抗性数据

Celebrity recognition

Rekognition automatically recognizes celebrities in images and provides confidence scores (Your images aren't stored.)



Choose a sample Image



Use your own image

Upload

or drag and drop

Use image URL

Go

Done with the demo?

[Download SDKs](#)

▼ Results



Jeff Bezos
[Learn More](#)

Match confidence

100%

► Request

► Response

对抗性数据 – 图像生成(e.g. Sharif et al. 2017)

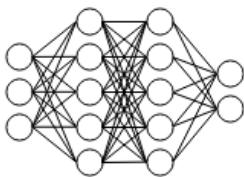
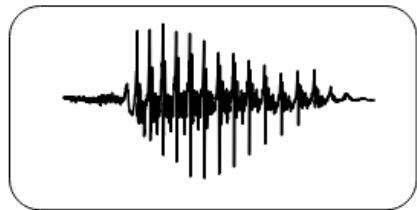


数字操纵来躲避识别

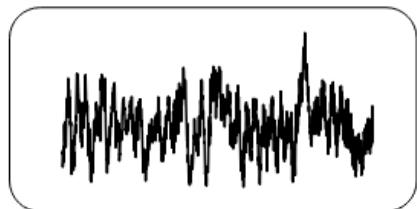


在现实生活中 - 通过3D眼镜

对抗性数据 – 语言生成 (e.g. Carlini & Wagner, 2018)

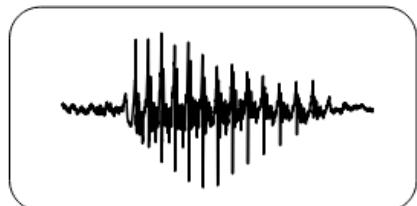


"it was the
best of times,
it was the
worst of times"

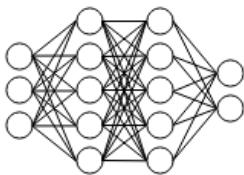


+

$$\times \ 0.001$$



=



"it is a truth
universally
acknowledged
that a single"

- 稍微修改数据，以获得错误的类

$$\underset{\delta}{\text{maximize}} \ l(f(x + \delta), y)$$

$$\text{subject to } \|\delta\| \leq \epsilon$$

不同的规范
不同的数据集
不同的论文.....

'不自然'的数据



- 训练和“自然”测试数据存在于小的子集中
- 对抗性数据略微偏离
- 函数未定义，远离数据发生的位置

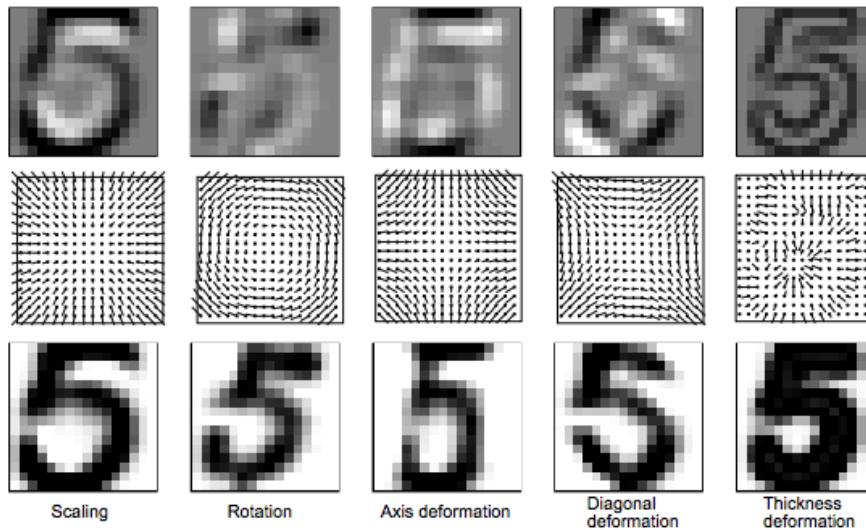
垃圾邮件防御

- While TRUE
 - 邮件主机扩展数据集并训练新的分类器
 - 垃圾邮件发送者的电子邮件被拒绝
 - 垃圾邮件发送者找到了成功的修改
- 例子
 - 将spam可能性较高的单词（或句子）添加到电子邮件中
 - 修改spam可能性较高的句子
 - 改变或伪造标题（‘Dear Alex, ...’）

不变性

- 切线距离 (Simard et al., 1995)

- 不变性变换不会更改标签
- 探索数据及其邻居



不变性

- 虚拟支持向量 (Schoelkopf, 1997)
 - 仅更改边界处的数据 (没有足够的RAM)
- 用于训练的数据增广
 - Imagenet (几乎每篇论文)
 - 裁剪, 缩放, 改变平均值, 每个频道,
 - 语音识别
 - 背景噪声, 场景.....
 - 文档分析
 - 随机子串, 单词删除, 插入

不变性和耐久的损失

- 凸损失 (Teo等, 2005)
 - 变革家族 $\delta \in \Delta$
 - 对极端转变的惩罚 $1 \geq \eta(\delta) \geq 0$
 - 找到每个步骤中“最差”的可能示例

对抗强大的网络

$$L(x, y, f) = \sup_{\delta \in \Delta} \eta(\delta) l(f(x + \delta), y)$$

例如
对抗性示例生成器
(发现最坏的情况)

减少极端扭曲的惩罚

非平稳环境

与环境的互动

- 批量 (下载书籍)
 - 观察训练数据 $(x_1, y_1) \dots (x_l, y_l)$ 然后进行部署
- 在线 (跟随课程)
 - 观察 x , 预测 $f(x)$, 观察 y (股市, 家庭作业)
- 主动学习 (在课堂上提问)
 - 为 x 查询 y , 改进模型, 选择新 x
- 强化学习 (做好功课, 下棋, 开车)
 - 采取行动, 环境响应, 采取新行动...



批量

训练集

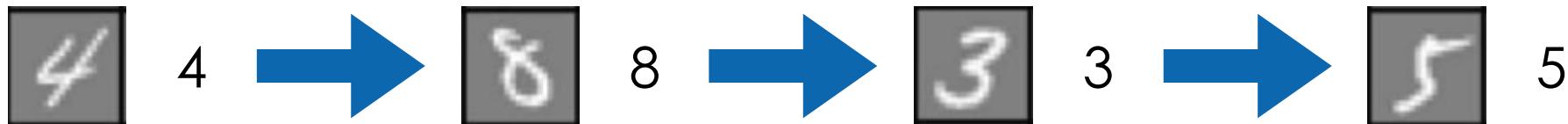
6	5	5	4	1	0
7	4	0	8	4	3
3	4	2	8	1	0
0	0	1	6	5	5
1	1	1	6	7	1
8	6	4	5	3	8
1	7	2	8	4	7
5	2	8	0	4	8
3	3	7	0	5	3
4	8	9	4	0	4

建模

测试集

4	9	1	7
6	4	5	6
7	5	9	7
1	1	5	9
4	1	3	1
7	2	9	1
6	8	9	3
3	7	4	6
1	1	0	3
5	0	5	0

在线学习算法



随着我们看到更多数据

有状态系统



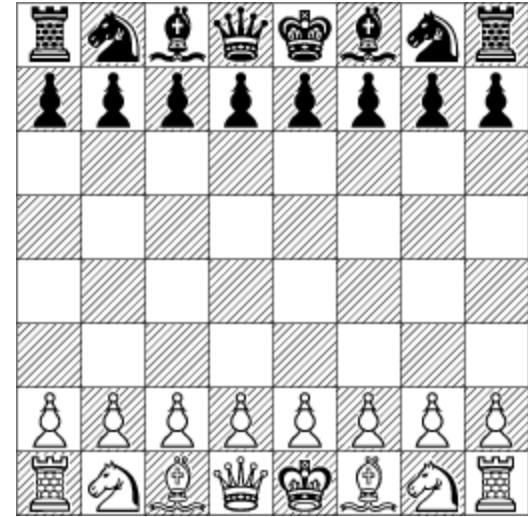
无记忆



有记忆

强化学习

- 采取行动
 - 对环境做出反应
 - 观察一下
 - 更新模型
 - 重复
-
- 环境 (合作, 对手, 不关心)
 - 记忆 (金鱼, 大象)
 - 状态空间 (tic tac toe, 象棋, 汽车)
 - 过去观察数据 (服务器日志, 培训期间生成)



强化学习

- 游戏
 - 国际象棋, 围棋, 步步高 (完全观察)
 - 德州扑克, 星际争霸, ATARI (部分观察, 随机)
- 排比
 - 计算广告, 推荐系统 (多个代理和独立并行游戏)
 - 负载均衡和调度 (多个代理)
- 操作
 - 持续决策 (驾驶, 飞行, 机器人, HVAC)
 - 离散 (电梯, 工作分配)
- 模拟
 - MuJoCo 风格
 - 只有现实 (服务器中心)



训练 ≠ 测试

- 泛化表现 (经验分布)

$$p_{\text{emp}}(x, y) \neq p(x, y)$$

- 协变量偏移 (协变量分布)

$$p(x) \neq q(x)$$

- 逻辑回归 (修复偏移的工具)

$$\log(1 + \exp(-yf(x)))$$

- 协变量偏移校正

$$\frac{1}{2} (p(x)\delta(1, y) + q(x)\delta(-1, y))$$

- 标签转移 (标签分布所在)

$$p(y) \neq q(y)$$

- 非平稳环境



概要

- 泛化表现
- 协变量偏移 (Covariate Shift)
 - 标签偏移
 - 协变量偏移校正
- 对抗性数据
- 非平稳环境



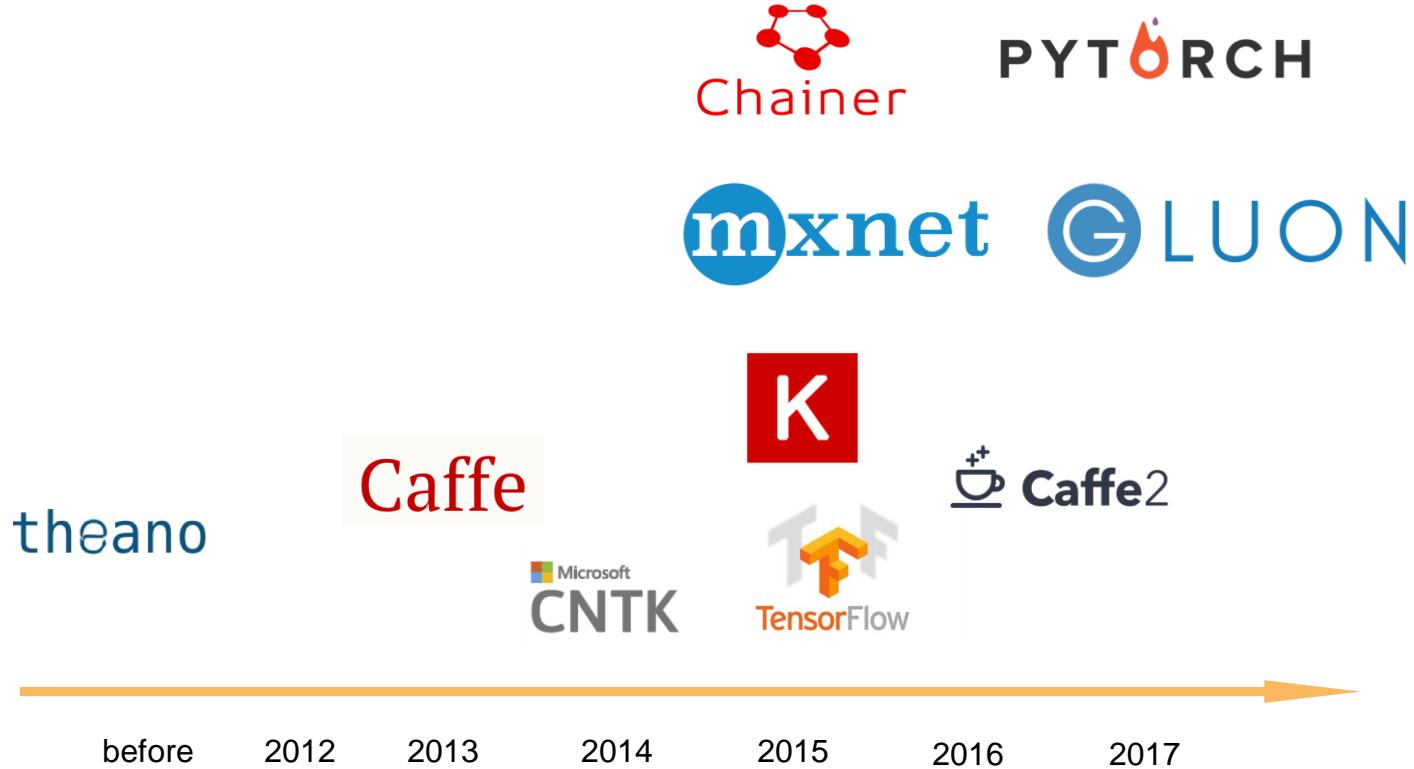
动手学深度学习

10. 深度学习框架

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/layers.html>



Caffe

ResNet-101-deploy.prototext

```
layer {
    bottom: "data"
    top: "conv1"
    name: "conv1"
    type: "Convolution"
    convolution_param {
        num_output: 64
        kernel_size: 7
        pad: 3
        stride: 2
    }
}
```

- Protobuf 作为界面
- 强大的视觉模型覆盖率
- 灵活可移植
- 不易开发

Tensorflow

实现Adam算法

```
# m_t = beta1 * m + (1 - beta1) * g_t
m = self.get_slot(var, "m")
m_scaled_g_values = grad.values * (1 - beta1_t)
m_t = state_ops.assign(m, m * beta1_t,
                      use_locking=self._use_locking)
m_t = state_ops.scatter_add(m_t, grad.indices, m_scaled_g_values,
                            use_locking=self._use_locking)
```

- Python的域特定语言(DSL)
- 丰富的算子
- 丰富的能力
- 代码可读性有限

Keras

```
model = Sequential()  
model.add(Dense(512, activation='relu',  
               input_shape=(784,)))  
model.add(Dropout(0.2))  
model.add(Dense(512, activation='relu'))  
model.add(Dropout(0.2))  
model.add(Dense(10, activation='softmax'))  
  
model.compile(...)  
model.fit(...)
```

- 简单的DSL for Python，可以使用多个后端 (TensorFlow, MXNet, CNTK...)
- 比 TensorFlow 简易
- 相对更慢
- 不易开发和调试

Pytorch

```
class Net(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

- Torch tensors + chainer 神经网络
- 易开发和调试
- 不易配置部署

MXNet

实现Resnet算法

```
bn1 = sym.BatchNorm(data=data, fix_gamma=False)
act1 = sym.Activation(data=bn1, act_type='relu')
conv1 = sym.Convolution(data=act1, num_filter=64, kernel_size=(3, 3), stride=(1, 1), pad=(1, 1))
```

实现Adam算法

```
coef2 = 1. - self.beta2**t
lr *= math.sqrt(coef2)/coef1

weight -= lr*mean/(sqrt(variance) + self.epsilon)
```

- 像 Numpy 一样的 Tensor + 像 Keras 一样的神经网络
- 高性能
- 高实用性

MXNet + Gluon

```
net = gluon.nn.Sequential()  
net.add(gluon.nn.Dense(128, activation='relu'))  
net.add(gluon.nn.Dense(64, activation='relu'))  
net.add(gluon.nn.Dense(10))  
  
loss = gluon.loss.SoftmaxCrossEntropyLoss()  
  
for data, label in get_batch():  
    with autograd.record():  
        l = loss(net(data), label)  
    l.backward()  
    trainer.step(batch_size=data.shape[0])
```

- 像 Numpy 一样的 Tensor + 像 Keras 一样的神经网络
- 易开发和调试
- 高性能

GluonCV

- 计算机视觉工具包
<https://gluon-cv.mxnet.io/>
- 预先训练的模型
- 训练脚本以重现 SOTA 结果

Application

Illustration

Available Models

Dog



Image Classification:

recognize an object in an image.

50+ models, including [ResNet](#), [MobileNet](#), [DenseNet](#), [VGG](#), ...

Dog



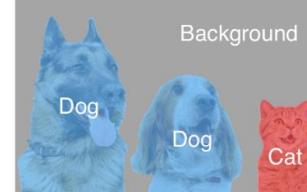
[Faster RCNN](#), [SSD](#), [Yolo-v3](#)

Object Detection:

detect multiple objects with their bounding boxes in an image.

Semantic Segmentation:

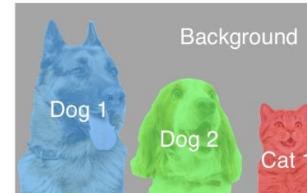
associate each pixel of an image with a categorical label.



[FCN](#), [PSP](#), [DeepLab v3](#)

Instance Segmentation:

associate each pixel of an image with an instance label.



[Mask RCNN](#)



GluonNLP

- NLP工具包
<https://gluon-nlp.mxnet.io/>
- 预先训练的模型
- 训练脚本以重现SOTA结果

Word Embedding

Mapping words to vectors.

Language Modeling

Learning the distribution and representation of sequences of words.

Machine Translation

From "Hello" to "Bonjour".

Text Classification

Categorize texts and documents.

Sentiment Analysis

Classifying polarity of emotions and opinions.

Parsing

Dependency parsing.

Natural Language Inference

Determine if the premise semantically entails the hypothesis.

Text Generation

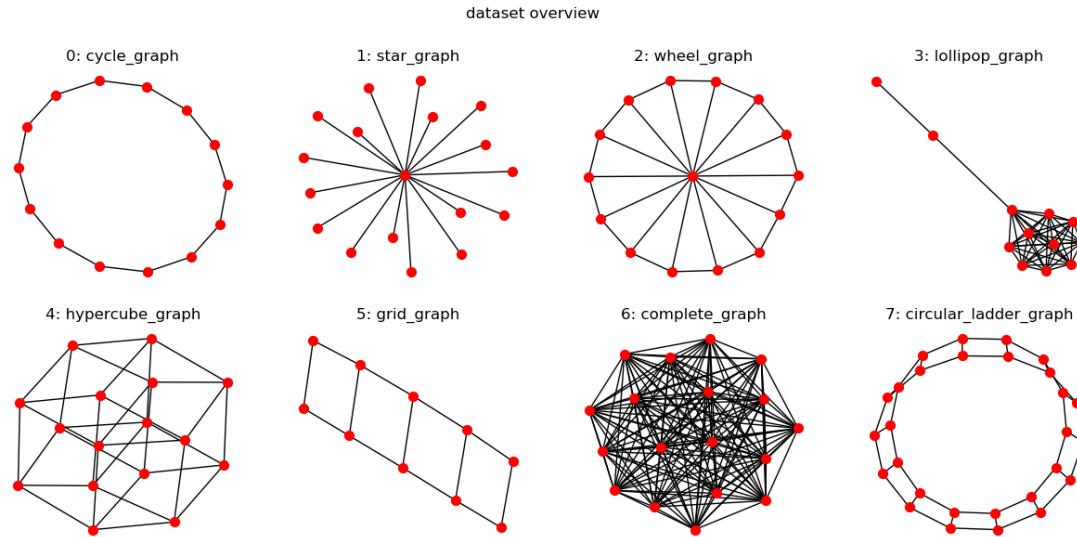
Generating language from models.

BERT

Transfer pre-trained language representations to language understanding tasks.

DGL

- 图神经网络工具包
<https://www.dgl.ai/>
- 相对较新，但具有良好的模型覆盖率
- 有MXNet和PyTorch后端



2019 展望

- 更多工具包
 - 时间序列, AutoML,
- 100%兼容 NumPy
 - mxnet中的一个新的np包
- 集成编译器
 - CPU / GPU性能提升50%
 - 更多硬件覆盖: edge, ASIC,



D2L.ai

Gluon - 命令式神经网络API

- 创建层和神经网络
 - gluon.nn
- 初始化和更新参数
 - gluon.Parameter
- 使用GPU

动手学深度学习

11. 卷积和池化层

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/convnet.html>

概要

- 卷积
 - 平移不变性和局部性
 - 卷积层
 - 填充和步幅
- 多个输入和输出通道
- 池化

老
年
志
願



分类图像中的狗和猫

- 使用好的相机
- RGB图像具有 36M 个元素
- 使用 100 个神经元单隐含层的 MLP 模型：
 - 有 36 亿个参数
 - 超过地球上的狗和猫的数量 (900M 狗+ 600M 猫)



Dual
12MP
wide-angle and
telephoto cameras



2L.ai

单隐含层网络

100个 神经元

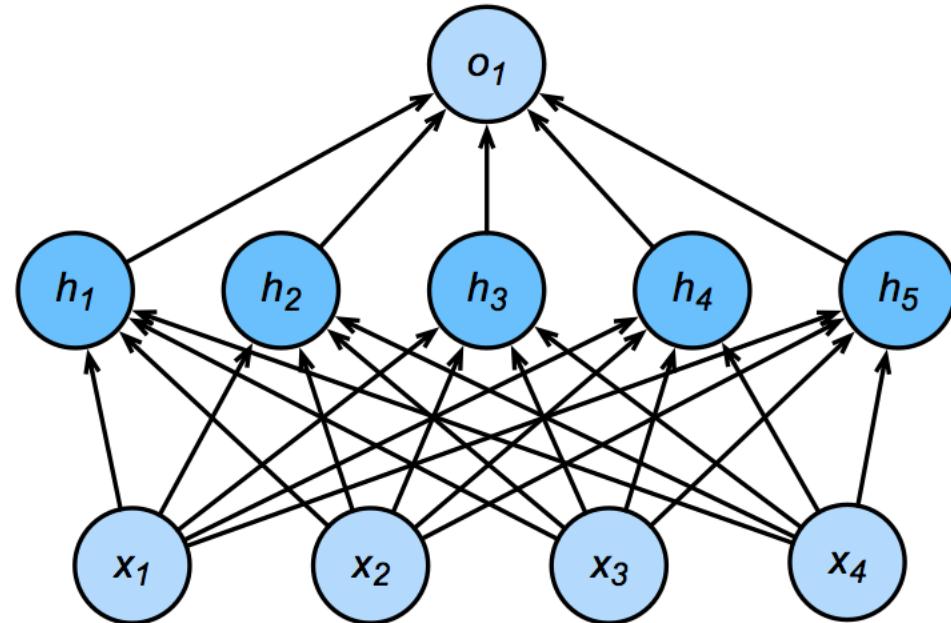
3.6B 参数 = 14GB

36M 特征

Output layer

Hidden layer

Input layer



$$\mathbf{h} = \sigma(\mathbf{Wx} + \mathbf{b})$$

Waldo 在哪?



两个原则

- 平移不变性
- 局部性



重新思考 - 稠密层

- 将输入和输出变形为矩阵（宽度，高度）
- 将权重变形为4-D张量 (h, w) 到 (h', w')

$$h_{i,j} = \sum_{k,l} w_{i,j,k,l} x_{k,l} = \sum_{a,b} v_{i,j,a,b} x_{i+a,j+b}$$

- v 是 w 的重索引

$$v_{i,j,a,b} = w_{i,j,i+a,j+b}$$

原则 #1 - 平移不变性

$$h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a, j+b}$$

- x 的变化也导致 h 的变化
- v 不应该依赖于 (i, j) ，所以 $v_{i,j,a,b} = v_{a,b}$

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a, j+b}$$

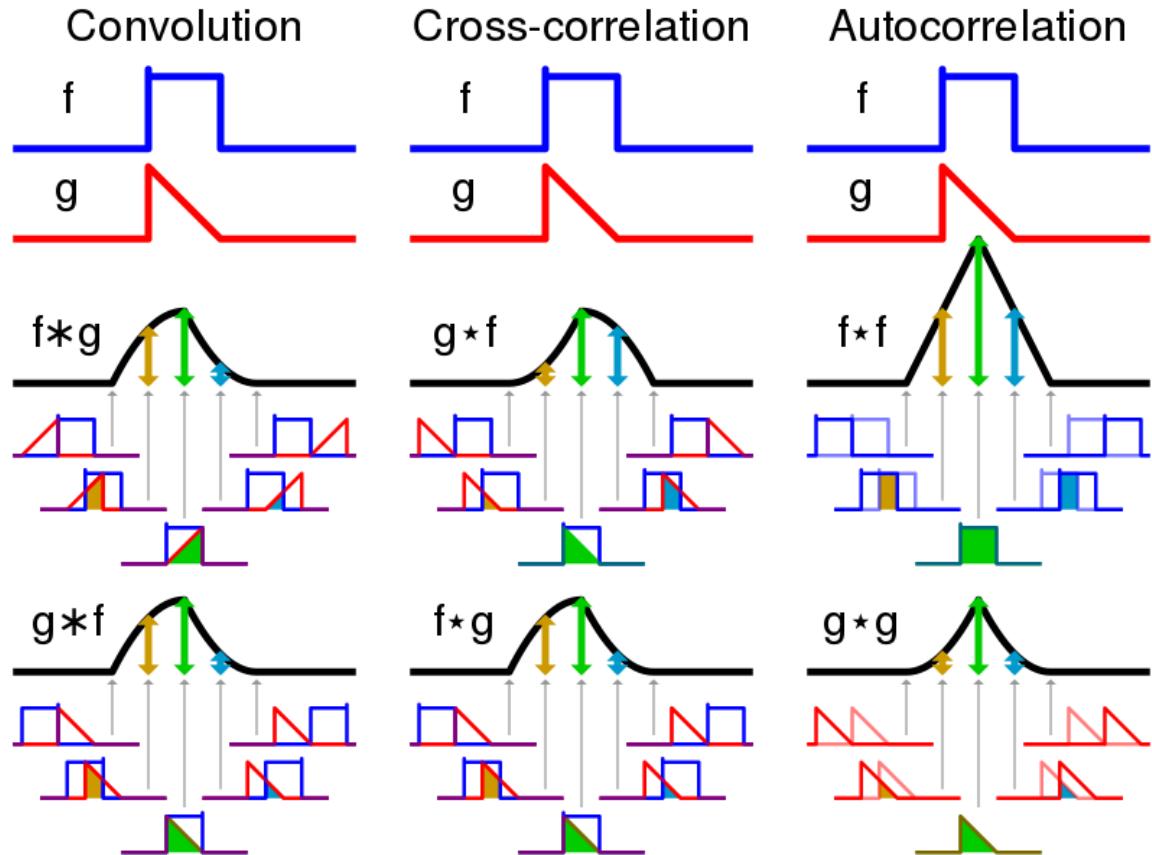
原则 #2 - 局部性

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a, j+b}$$

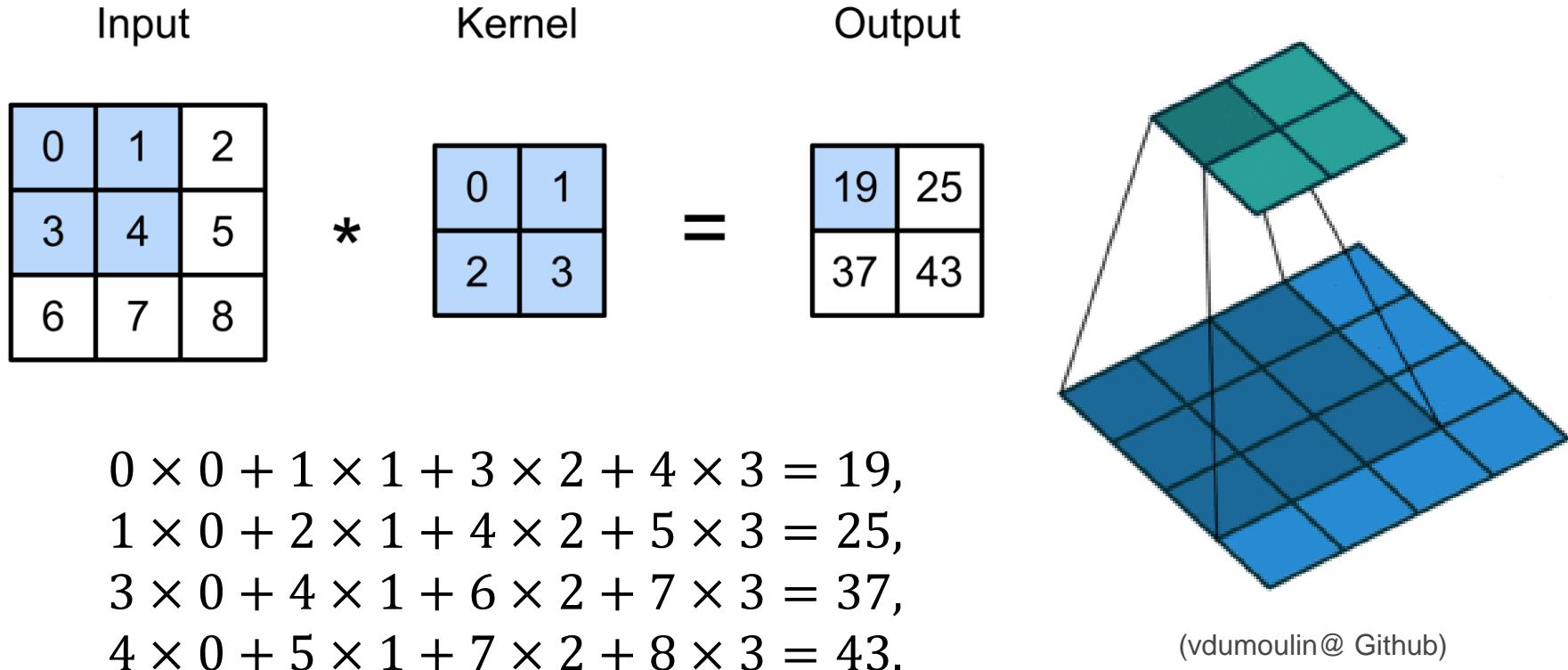
- 当评估 $h(i,j)$ 时，我们不应该用远离 $x(i, j)$ 的参数
- 外部参数 $|a|, |b| > \Delta$ 消失， $v_{a,b} = 0$

$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a, j+b}$$

卷积层



二维交叉相关



(vdumoulin@ Github)

二维卷积层

$$\begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 0 & 1 \\ \hline 2 & 3 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 19 & 25 \\ \hline 37 & 43 \\ \hline \end{array}$$

- $\mathbf{X}: n_h \times n_w$ 输入矩阵
- $\mathbf{W}: k_h \times k_w$ 核矩阵
- \mathbf{b} : 偏差标量
- $\mathbf{Y}: (n_h - k_h + 1) \times (n_w - k_w + 1)$ 输出矩阵

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

- \mathbf{W} 和 b 是可学习的参数

例子

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



(维基百科)

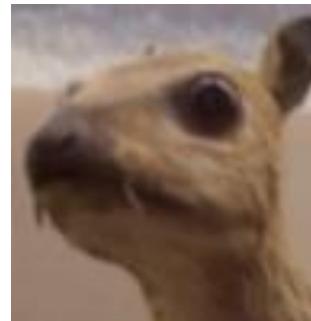
$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



边缘检测



锐化



高斯模糊

例子



(Rob Fergus)



互相关与卷积

- 2-D 互相关

$$y_{i,j} = \sum_{a=1}^h \sum_{b=1}^w w_{a,b} x_{i+a, j+b}$$

- 2-D 卷积

$$y_{i,j} = \sum_{a=1}^h \sum_{b=1}^w w_{-a,-b} x_{i+a, j+b}$$

- 在对称性方面没有差别

1-D 和 3-D 交叉相关

- 1-D

$$y_i = \sum_{a=1}^h w_a x_{i+a}$$

- 文本
- 语音
- 时间序列

- 3-D

$$y_{i,j,k} = \sum_{a=1}^h \sum_{b=1}^w \sum_{c=1}^d w_{a,b,c} x_{i+a, j+b, k+c}$$

- 视频
- 医学图像

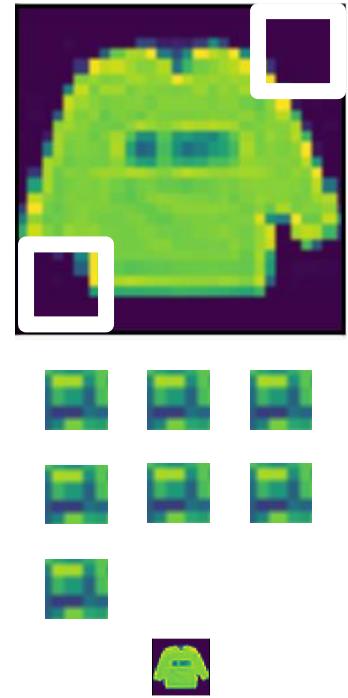
A composite photograph showing a man in a dark jacket and blue jeans crossing a city street four times simultaneously. He is in different stages of his stride in each instance, from starting to cross to having almost reached the other side. The background shows a typical urban street with parked cars, buildings, and other people walking. The overall effect is a visual metaphor for the concepts of 'fill' and 'step' in a workflow.

填充和步幅

填充

- 给定输入图像 (32×32)
- 应用 5×5 大小的 卷积核
- 第1层得到输出大小 28×28
- 第7层得到输出大小 4×4
- 更大的卷积核可以更快地减小输出
- 形状从 $n_h \times n_w$ 减少到

$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$

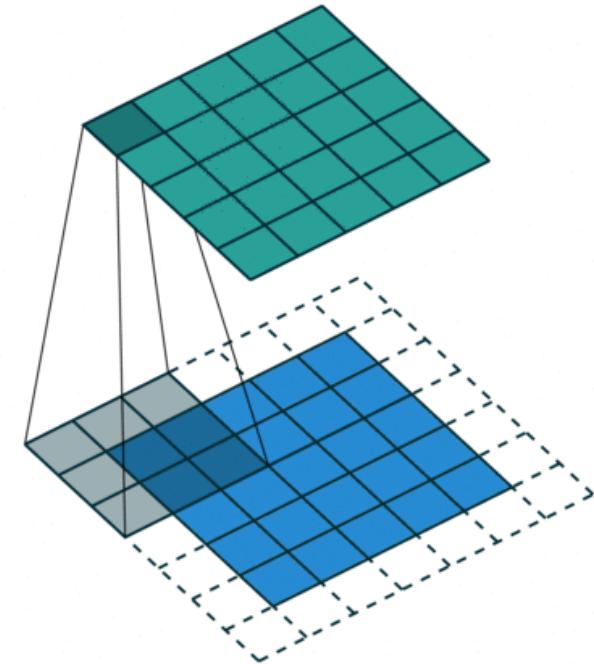


填充

填充：在输入周围添加额外的行 / 列

Input					Kernel		Output					
0	0	0	0	0	*	0	1	=	0	3	8	4
0	0	1	2	0		2	3		9	19	25	10
0	3	4	5	0					21	37	43	16
0	6	7	8	0					6	7	8	0
0	0	0	0	0								

$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$



填充

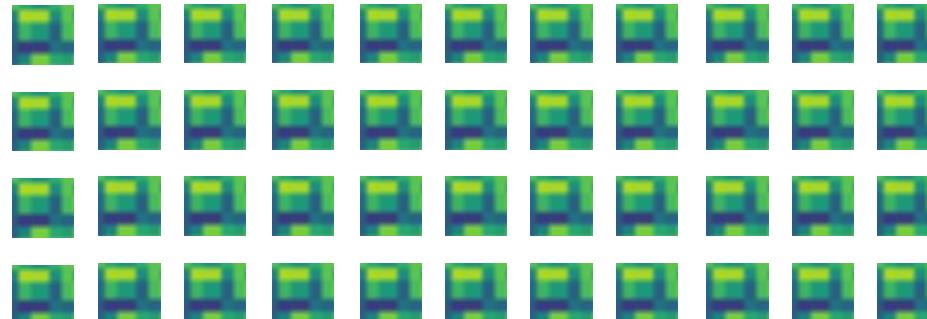
- 填充 p_h 行和 p_w 列，则输出为：

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

- 通常取 $p_h = k_h - 1$, $p_w = k_w - 1$
 - 当 k_h 为奇数：在上下两侧填充 $p_h/2$
 - 当 k_h 为偶数：在上侧填充 $\lfloor p_h/2 \rfloor$, 在下侧填充 $\lceil p_h/2 \rceil$

步幅

- 填充降低的输出大小与层数线性相关
 - 给定输入大小 224×224 ，在使用 5×5 卷积核的情况下，需要44层将输出降低到 4×4
 - 需要大量的计算



步幅

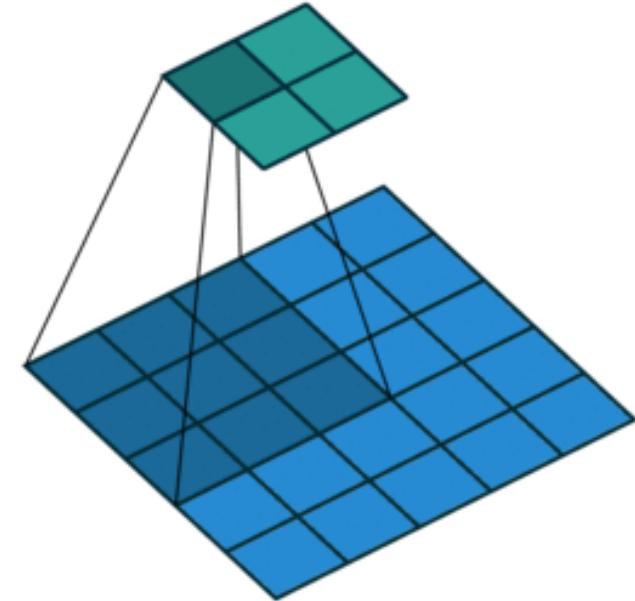
- 步幅是指行/列的滑动步长

例：高度3 宽度2 的步幅

Input					Kernel		Output			
0	0	0	0	0	*	0	1	=	0	8
0	0	1	2	0		2	3		6	8
0	3	4	5	0						
0	6	7	8	0						
0	0	0	0	0						

$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$

$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$



步幅

- 给出高度 s_h 和宽度 s_w 的步幅，输出形状是

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

- 如果 $p_h = k_h - 1, p_w = k_w - 1$

$$\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$$

- 如果输入高度和宽度可以被步幅整除

$$(n_h/s_h) \times (n_w/s_w)$$

An aerial photograph showing a complex network of waterways. The main channel flows from the bottom left towards the top right. Several smaller, parallel channels branch off from the main channel, creating a grid-like pattern. These channels are bordered by dense green vegetation and are filled with clear blue water. The perspective is from above, looking down the length of the channels.

多个输入和输出通道

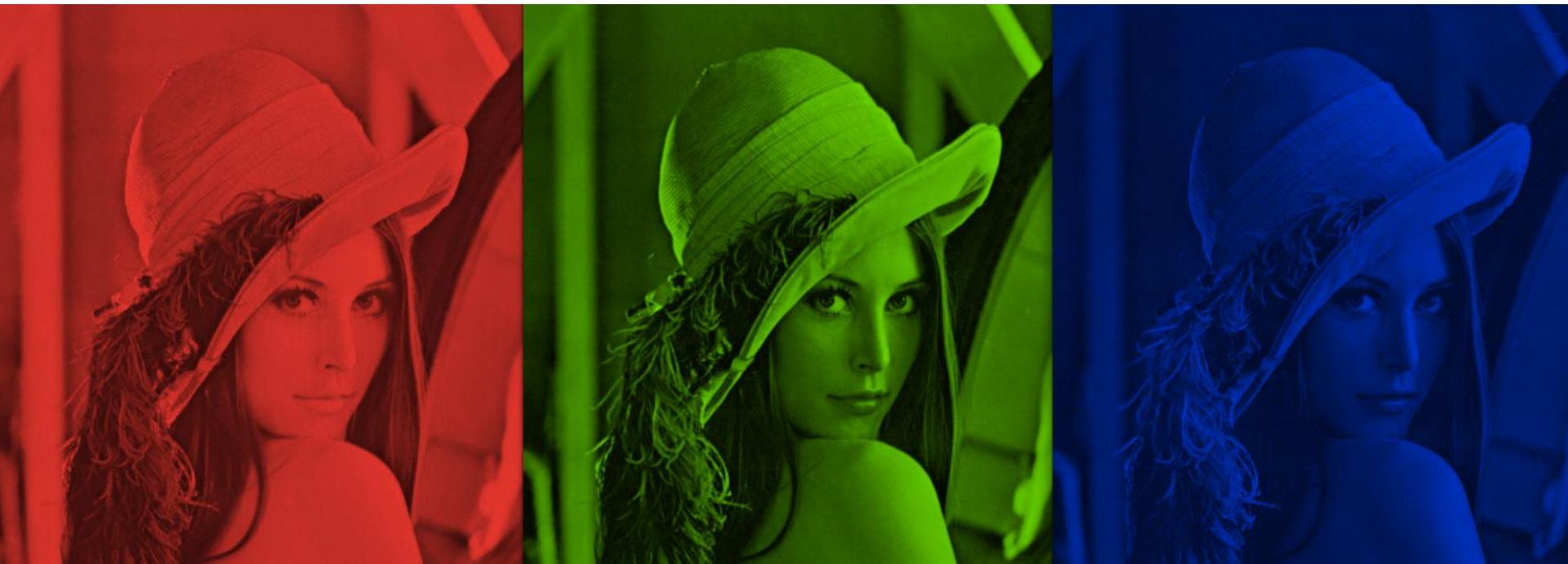
多个输入通道

- 彩色图像可能有 RGB 三个通道
- 转换为灰度会丢失信息



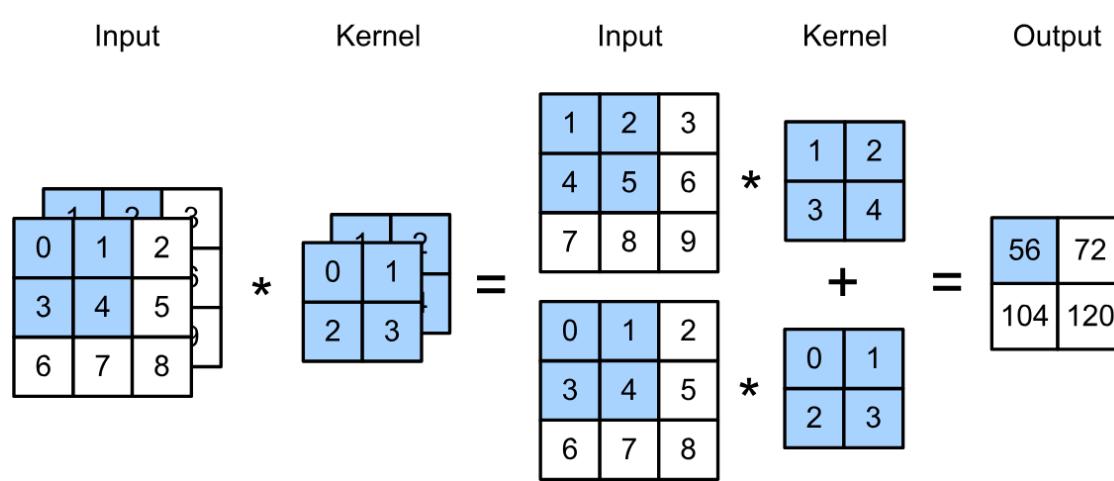
多个输入通道

- 彩色图像可能有 RGB 三个通道
- 转换为灰度会丢失信息



多个输入通道

- 每个通道都有一个卷积核，结果是所有通道卷积结果的和



$$(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 0 + 0 \times 1 + 1 \times 2 + 3 \times 4 + 4 \times 5) = 56$$

多个输入通道

- $\mathbf{X}: c_i \times n_h \times n_w$ 输入
- $\mathbf{W}: c_i \times k_h \times k_w$ 卷积核
- $\mathbf{Y}: m_h \times m_w$ 输出

$$\mathbf{Y} = \sum_{i=0}^{c_i} \mathbf{X}_{i,:,:} \star \mathbf{W}_{i,:,:}$$

多个输出通道

- 无论有多少输入通道，到目前为止我们只用到单输出通道
- 我们可以有多个3-D卷积核，每个核生成一个输出通道

- 输入 $\mathbf{X}: c_i \times n_h \times n_w$
- 内核 $\mathbf{W}: c_o \times c_i \times k_h \times k_w$
- 输出 $\mathbf{Y}: c_o \times m_h \times m_w$

$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:}$$
$$for i = 1, \dots, c_o$$

多个输入和输出通道

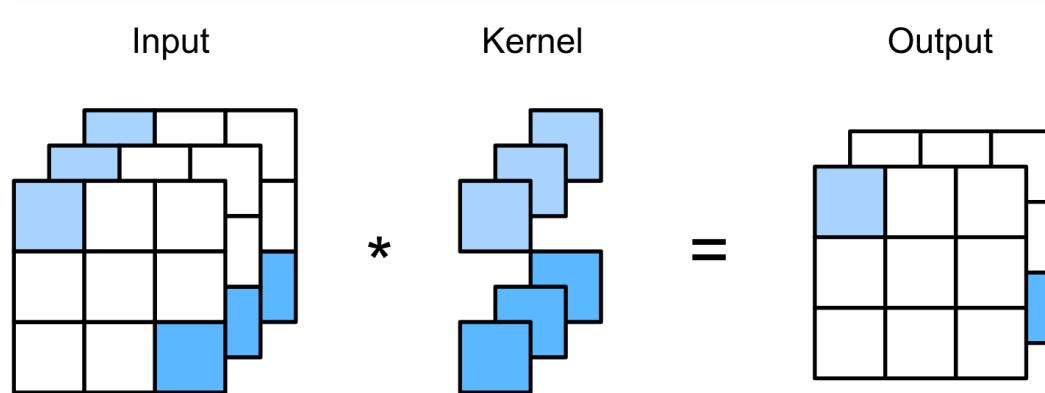
- 每个输出通道可以识别特定模式



- 输入通道内核识别并组合输入中的模式

1x1 卷积层

$k_h = k_w = 1$ 是一个受欢迎的选择。
它不识别空间模式，只是融合通道。



相当于具有输入 $n_h n_w \times c_i$ 和重量 $c_o \times c_i$ 的稠密层。

2-D 卷积层

$$Y = X \star W + B$$

- 输入 $X : c_i \times n_h \times n_w$
- 卷积核 $W : c_i \times c_o \times k_h \times k_w$
- 偏差 $B : c_i \times c_o$
- 输出 $Y : c_o \times m_h \times m_w$

复杂性 (浮点运算FLOP的数量)

$$c_i = c_o = 100$$

$$k_h = k_w = 5$$

$$m_h = m_w = 64$$

$$O(c_i c_o k_h k_w m_h m_w)$$

10层, 1M示例: 10PF
(CPU: 0.15 TF = 18h,
GPU: 12 TF = 14min)



池化层

池化层

- 卷积对位置敏感
 - 检测垂直边缘

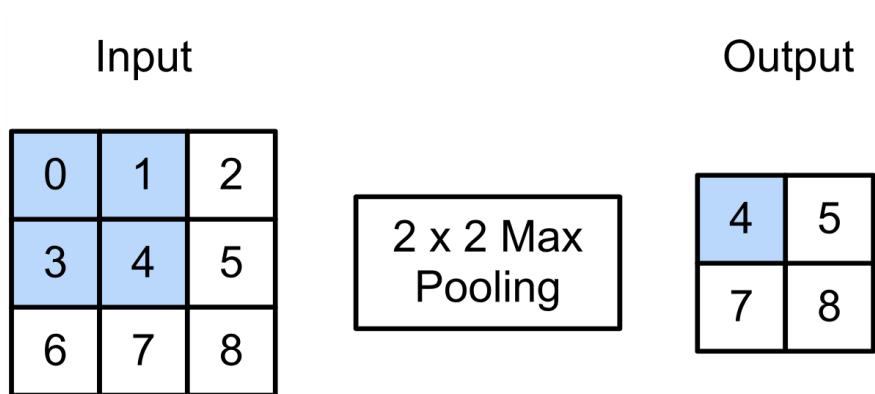
0输出, 1像素移位

$$\begin{array}{c} X \\ \times \end{array} \quad \begin{bmatrix} [1. & 1. & 0. & 0. & 0.] \\ [1. & 1. & 0. & 0. & 0.] \\ [1. & 1. & 0. & 0. & 0.] \\ [1. & 1. & 0. & 0. & 0.] \end{bmatrix} \quad Y \quad \begin{bmatrix} [0. & 1. & 0. & 0.] \\ [0. & 1. & 0. & 0.] \\ [0. & 1. & 0. & 0.] \\ [0. & 1. & 0. & 0.] \end{bmatrix}$$

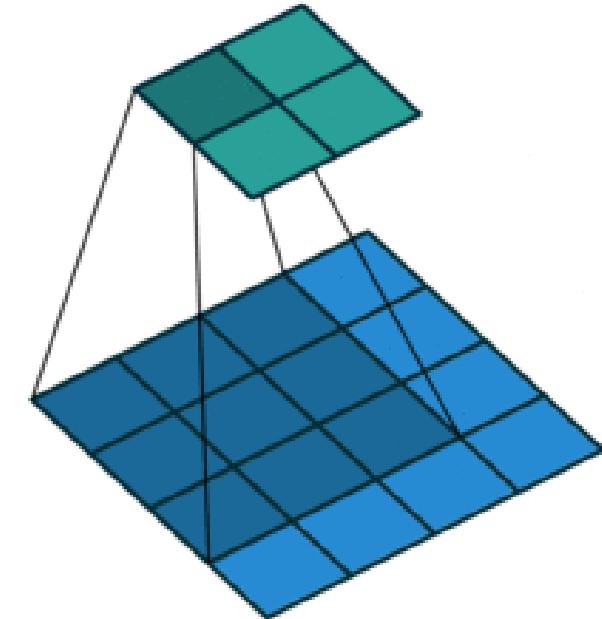
- 需要一定程度的平移不变性
 - 照明, 物体位置, 比例, 外观等等因图像而异

2-D 最大池化

- 返回滑动窗口中的最大值



$$\max(0,1,3,4) = 4$$



2-D 最大池化

- 返回滑动窗口中的最大值

垂直边缘检测

```
[[1.  1.  0.  0.  0.  
 [1.  1.  0.  0.  0.  
 [1.  1.  0.  0.  0.  
 [1.  1.  0.  0.  0.
```

卷积输出

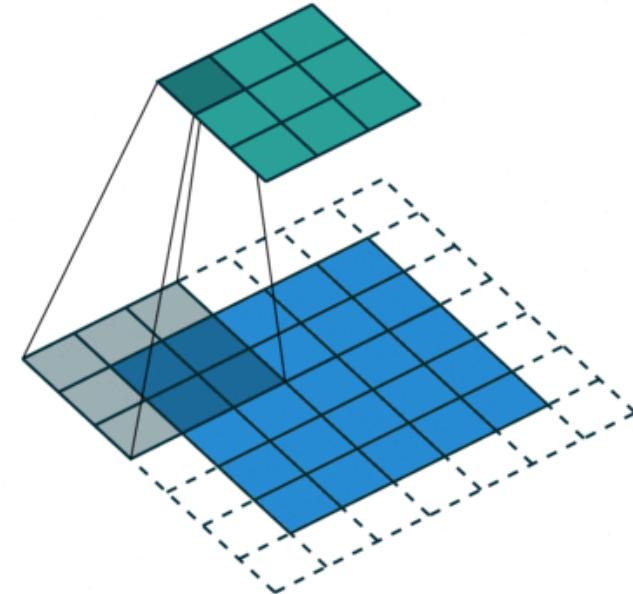
```
[[ 0.   1.   0.   0.  [  
 [ 0.   1.   0.   0.  [  
 [ 0.   1.   0.   0.  [  
 [ 0.   1.   0.   0.  [  
 [ 1.   1.   1.   0.  
 [ 1.   1.   1.   0.  
 [ 1.   1.   1.   0.  
 [ 1.   1.   1.   0.
```

2-D 最大池化

可容1像素移位

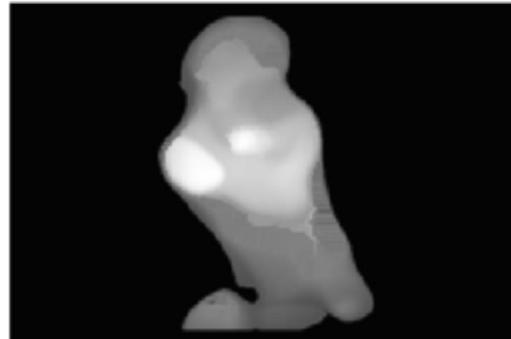
池化层 - 填充，步幅和多个通道

- 池化层与卷积层类似，都具有填充和步幅
- 没有可学习的参数
- 在每个输入通道应用池化层以获得相应的输出通道
- $\# \text{输出通道} = \# \text{输入通道}$



平均池化层

- 最大池化层：每个窗口中最强的模式信号
- 平均池化层：
 - 将最大池化层中的“最大”操作替换为“平均”
 - 窗口中的平均信号强度



最大池化层



平均池化层

总结

- 卷积层
 - 与稠密层相比，模型容量降低
 - 有效地检测空间模式
 - 计算复杂度高
 - 通过填充，步幅和通道控制输出形状
- 最大 / 平均池化层
 - 提供一定程度的平移不变性

动手学深度学习

12. LeNet, AlexNet, VGG 和 NiN

中文教材: zh.d2l.ai

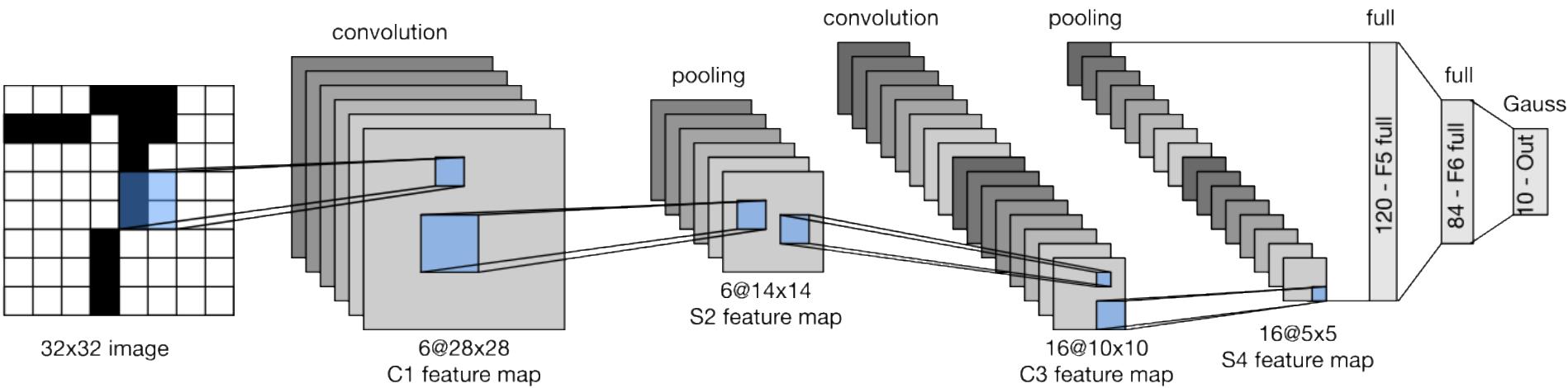
英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/lenet.html>

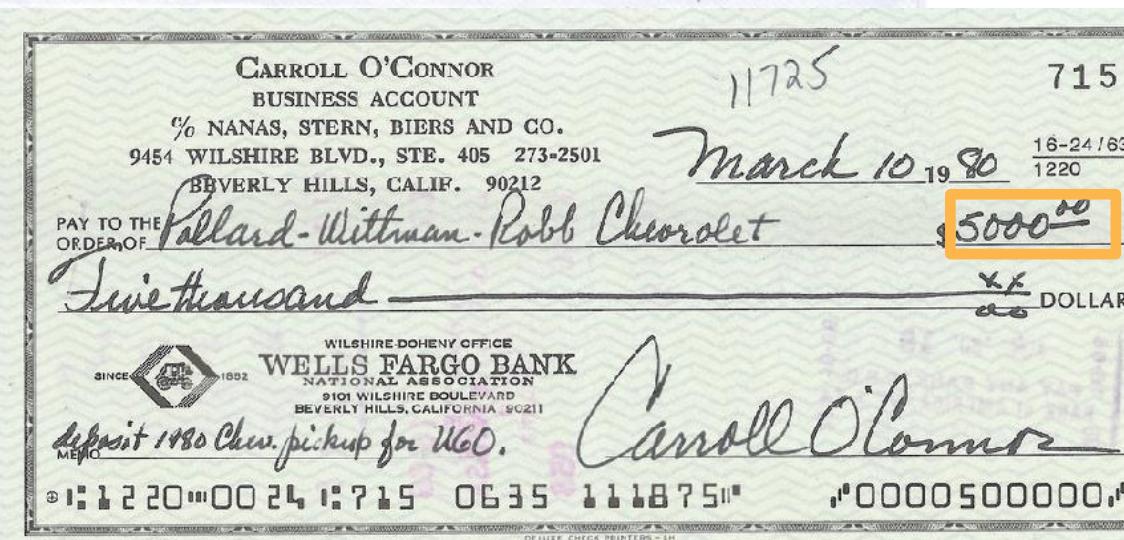
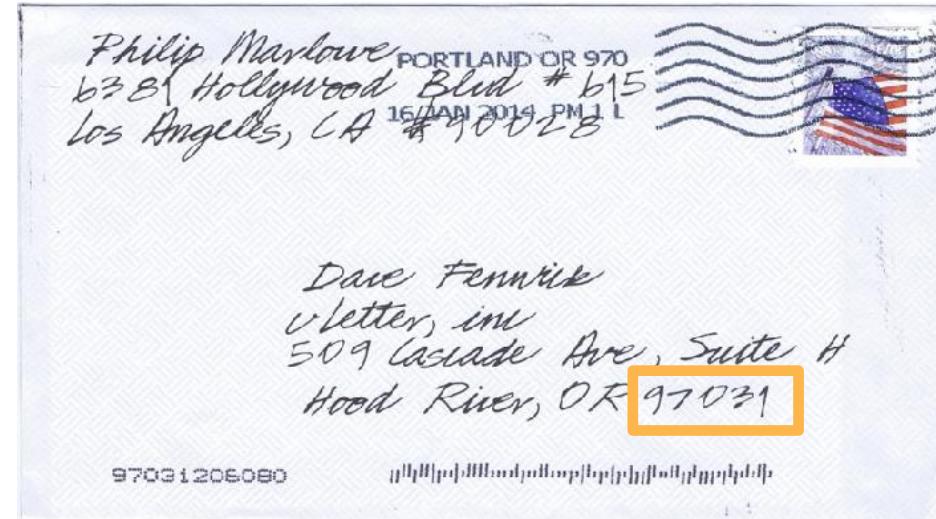
概要

- LeNet (第一个卷积神经网络)
- AlexNet
 - 升级版的 LeNet
 - ReLu 激活, 丢弃法, 平移不变性
- VGG
 - 升华版的 AlexNet
 - 重复的 VGG 块
- NiN
 - 1x1 卷积 + 全局池化

LeNet 架构



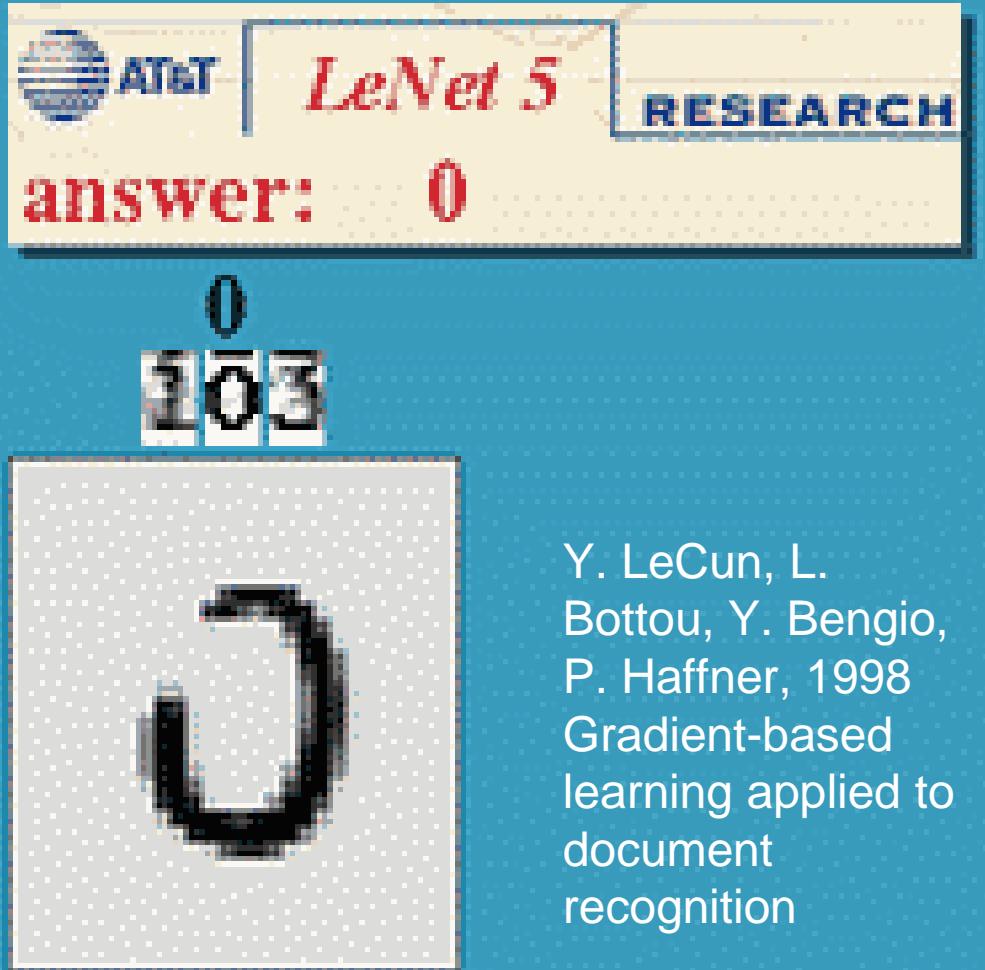
手写的数字识别



MNIST

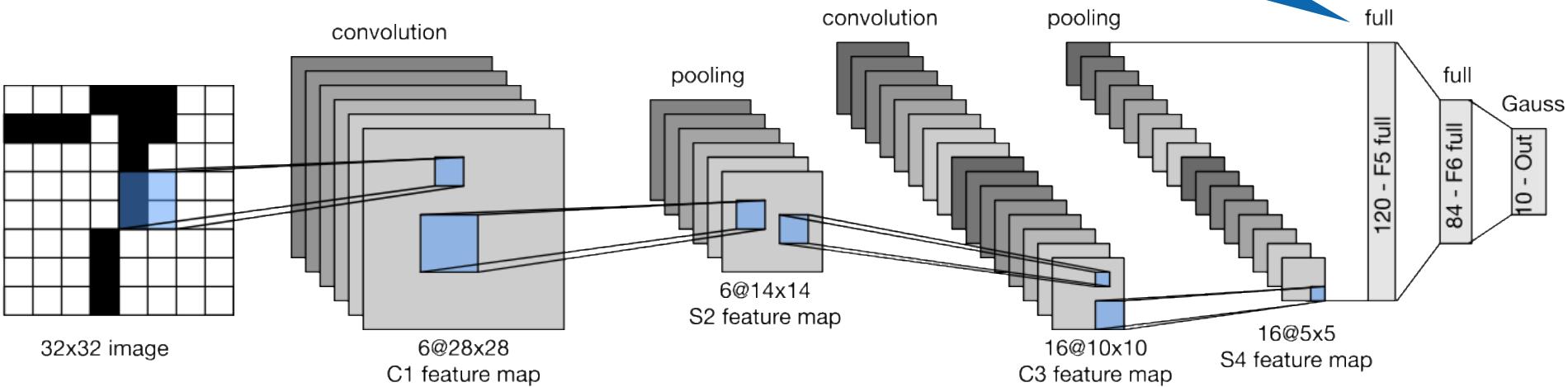
- 居中和缩放
- 50,000 个训练数据
- 10,000 个测试数据
- 图像大小 28×28
- 10 类





Y. LeCun, L.
Bottou, Y. Bengio,
P. Haffner, 1998
Gradient-based
learning applied to
document
recognition

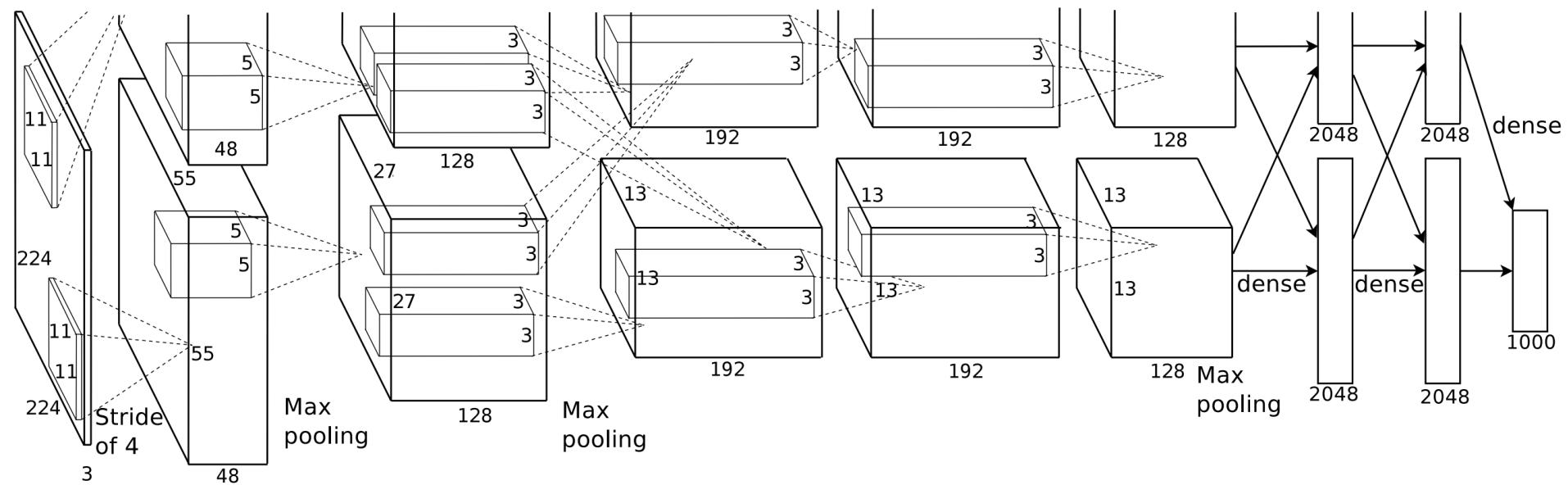
如果我们有很多输出，那开销就很大了



MXNet 中的 LeNet

- net = gluon.nn.Sequential()
- with net.name_scope():
 - net.add(gluon.nn.Conv2D(channels=20, kernel_size=5, activation='tanh'))
 - net.add(gluon.nn.AvgPool2D(pool_size=2))
 - net.add(gluon.nn.Conv2D(channels=50, kernel_size=5, activation='tanh'))
 - net.add(gluon.nn.AvgPool2D(pool_size=2))
 - net.add(gluon.nn.Flatten())
 - net.add(gluon.nn.Dense(500, activation='tanh'))
 - net.add(gluon.nn.Dense(10))
- loss = gluon.loss.SoftmaxCrossEntropyLoss()
- (size and shape inference is automatic)

AlexNet

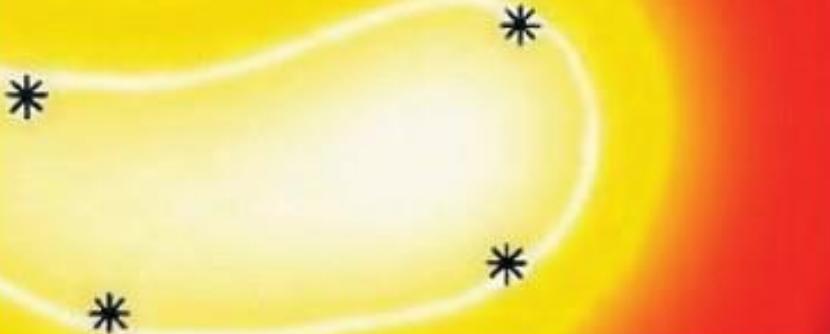


2001

Learning with Kernels

Support Vector Machines, Regularization,
Optimization, and Beyond

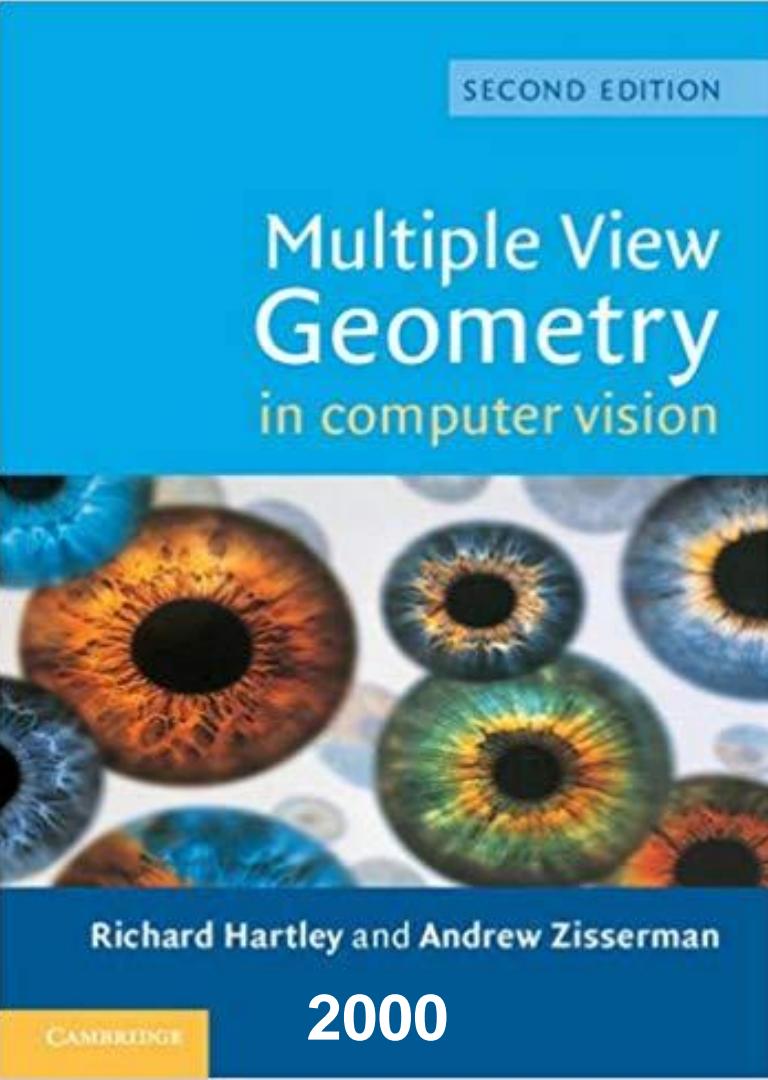
Bernhard Schölkopf and Alexander J. Smola



机器学习

In the 1990s, a new type of learning algorithm was developed, based on results from statistical learning theory: the Support Vector Machine (SVM). This gave rise to a new class of theoretically elegant learning machines that use a central concept of SVMs – kernels – for a number of

- 提取特征
- 选择内核以获得相似性
- 凸优化问题
- 许多完美的定理



几何学

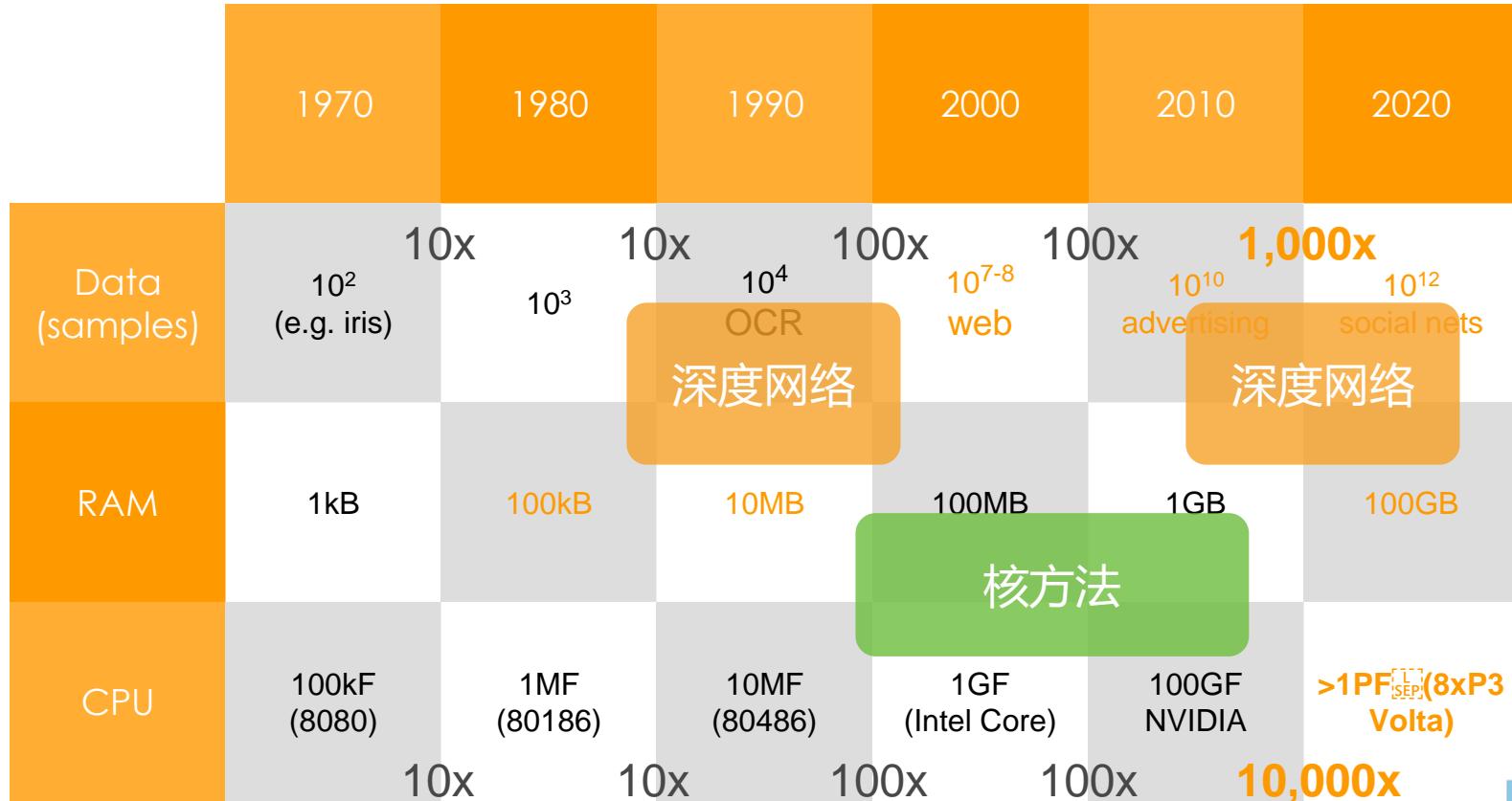
- 提取特征
- 不同角度描述几何（例如多个相机）
- （非）凸优化问题
- 许多完美的定理.....
- 当假设得到满足时，在理论上非常有效

特征工程



- 特征工程很重要
- 特征描述，例如 SIFT（尺度不变特征变换），SURF（更高效的完成特征的提取和描述）
- 视觉词袋（聚类）
- 应用SVM

硬件



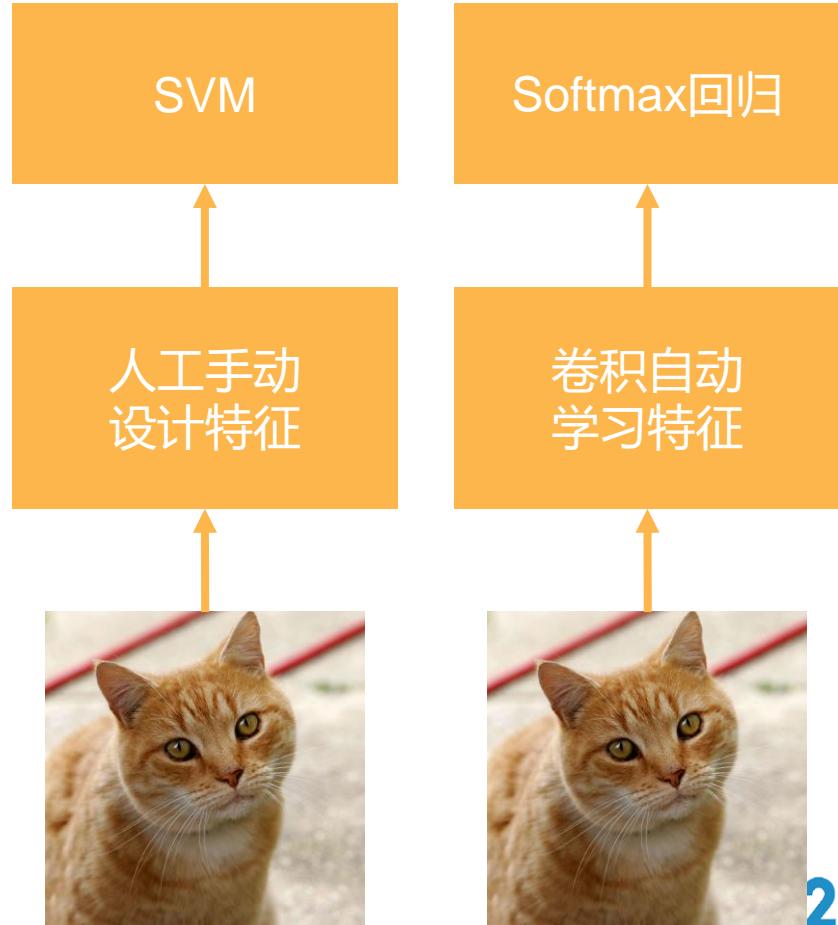
ImageNet 数据集 (2010)



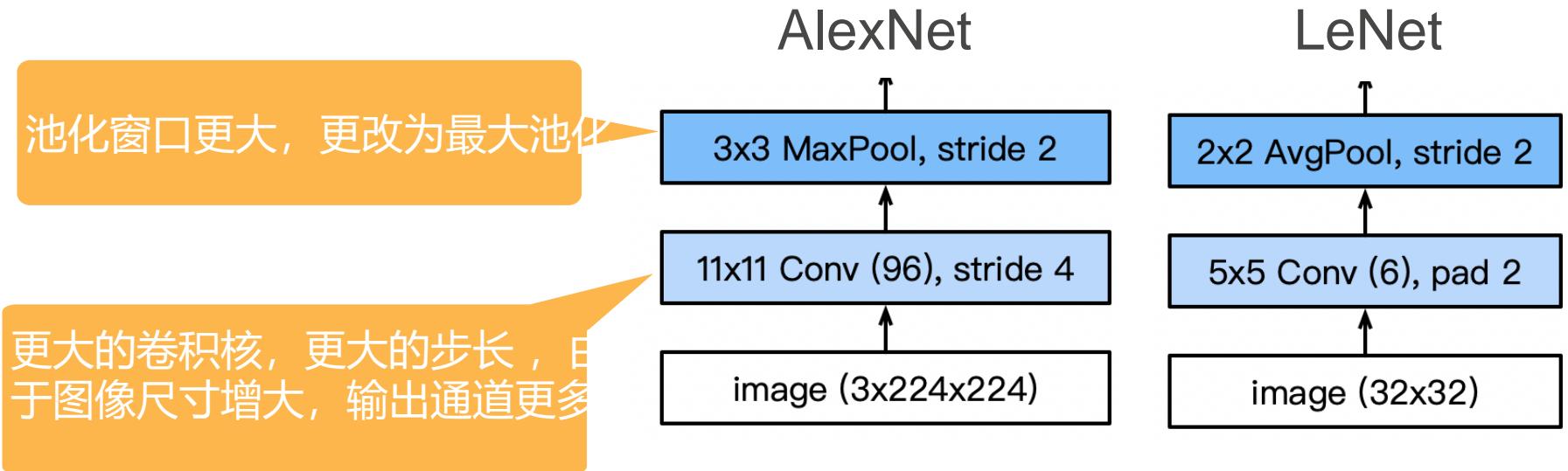
图像	自然物体的彩色图像	手写数字的灰色图像
尺寸	469 × 387	28 × 28
# 样本数	1200万	6万
# 类别数	1,000	10

AlexNet

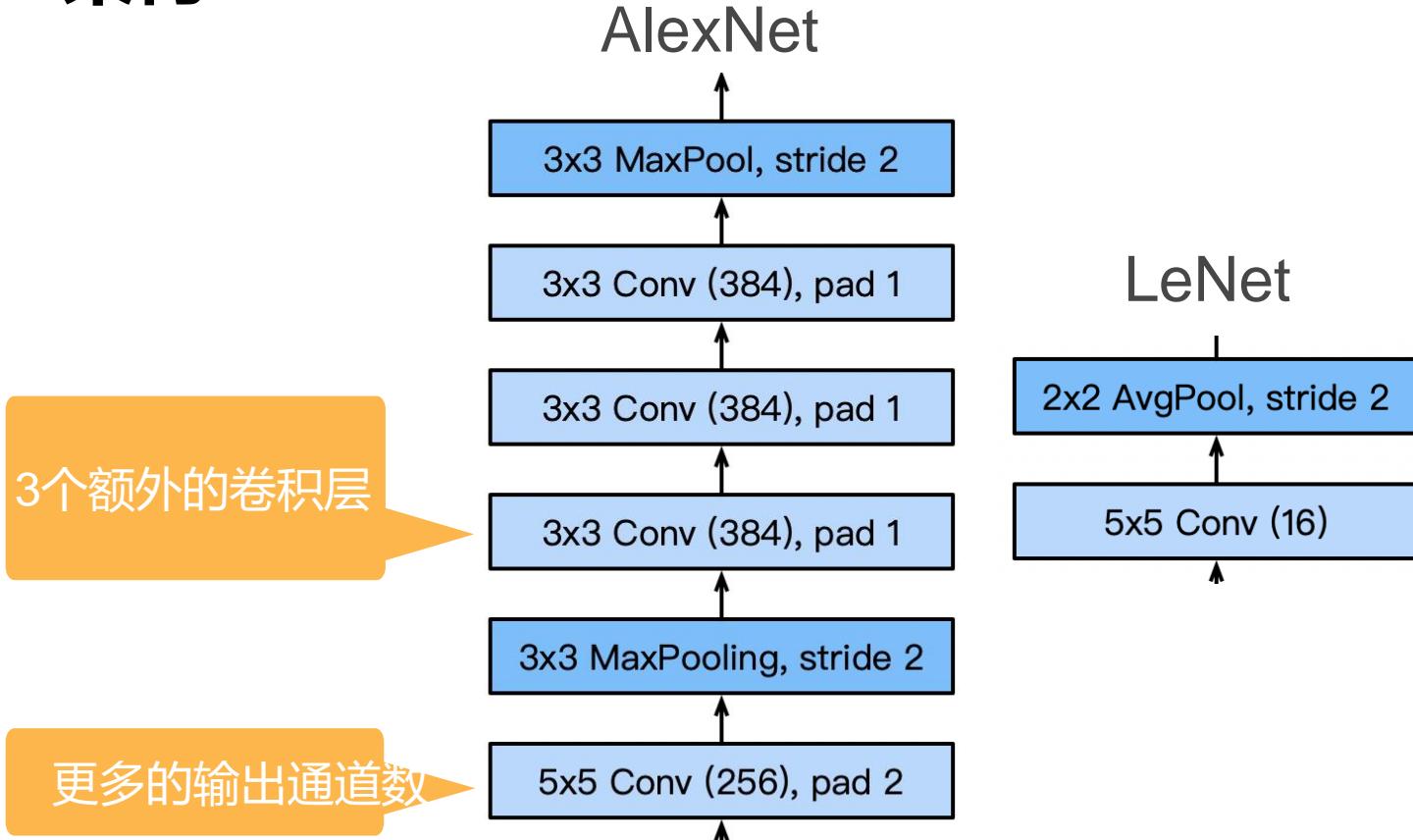
- AlexNet 在 2012 年赢得了 ImageNet 竞赛
- 更深更大的 LeNet
- 主要修改
 - 丢弃法（正则化）
 - ReLu 激活函数（训练）
 - 最大池化法
- 计算机视觉的范式转变



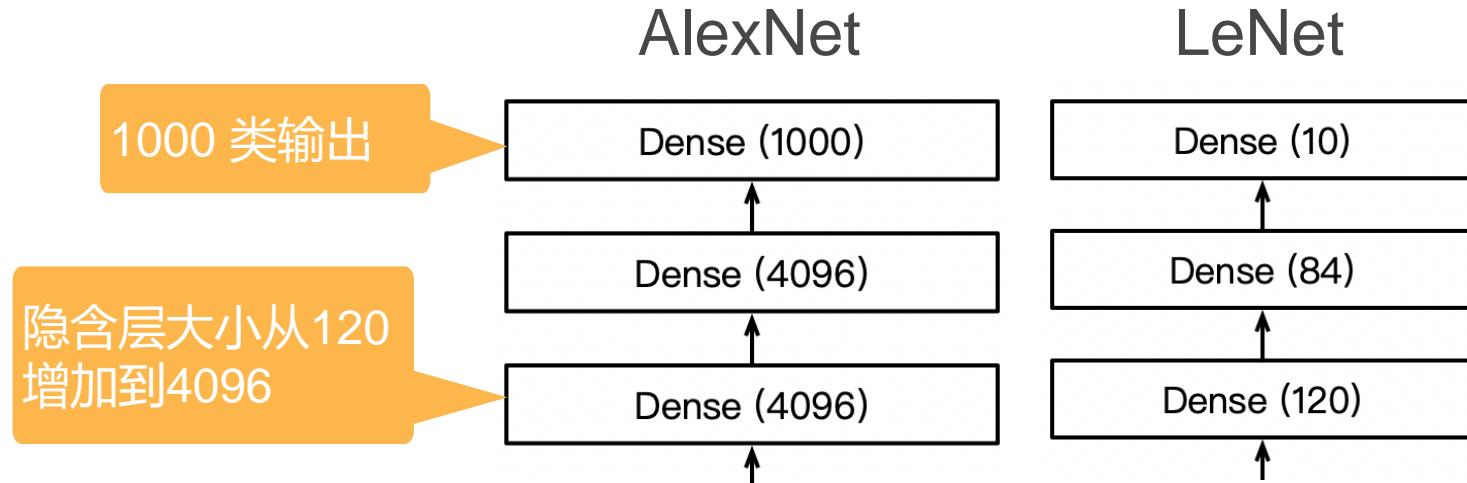
AlexNet 架构



AlexNet 架构

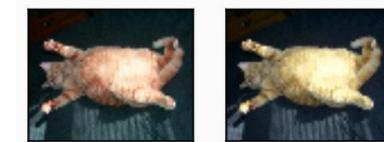


AlexNet 架构



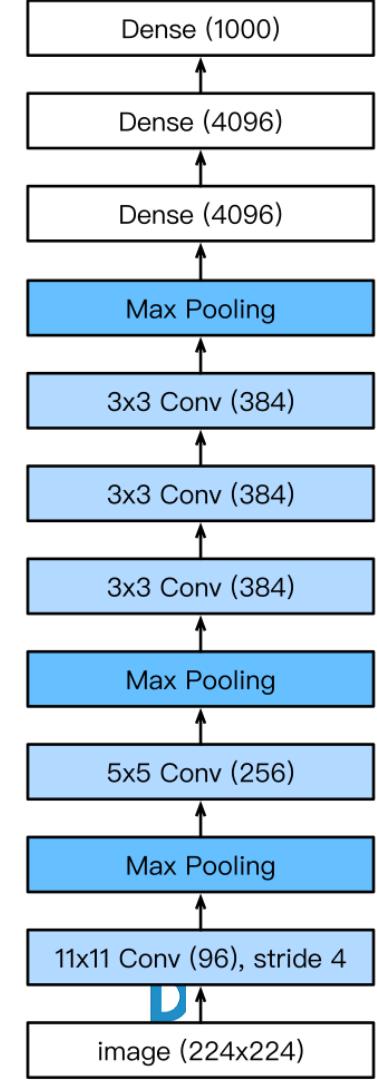
更多技巧

- 将激活函数从 sigmoid 更改为 ReLu (不再梯度消失)
- 在两个隐含层之后应用丢弃法 (更好的稳定性 / 正则化)
- 数据增强

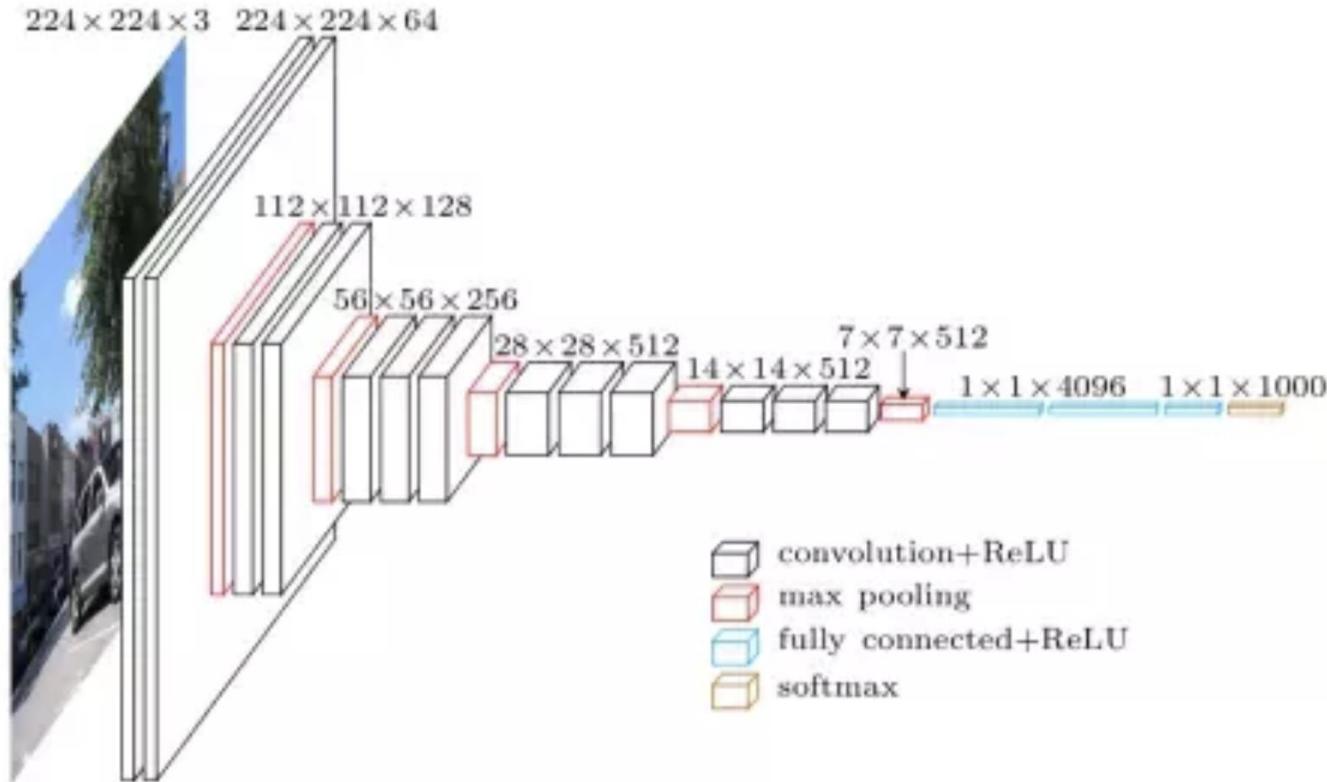


复杂度

	# 参数量	浮点计算量 (FLOP)	AlexNet	LeNet
	AlexNet	LeNet	AlexNet	LeNet
卷积层1	35K	150	101M	1.2M
卷积层2	614K	2.4K	415M	2.4M
卷积层3-5	3M		445M	
稠密层1	26M	0.48M	26M	0.48M
稠密层2	16M	0.1M	16M	0.1M
总共	46M	0.6M	1G	4M
倍数增加	11x	1x	250x	1x

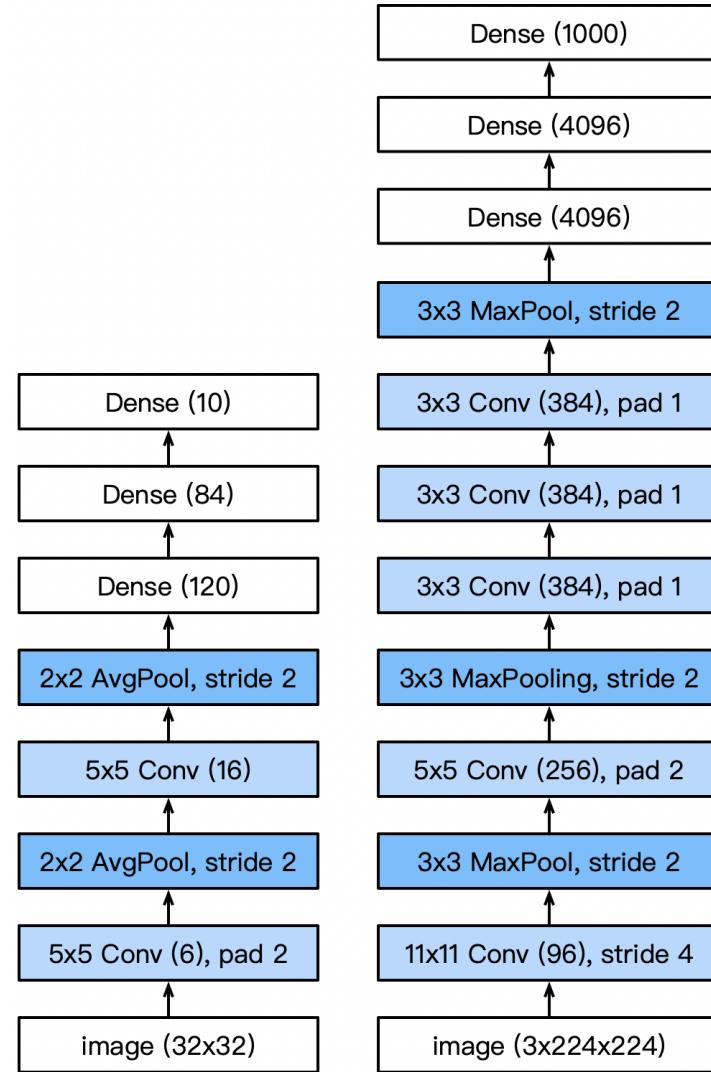


VGG



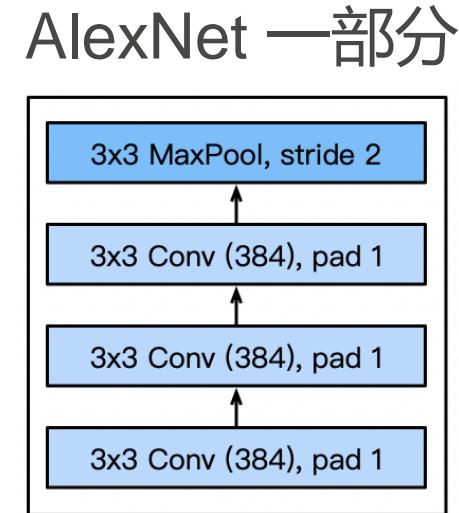
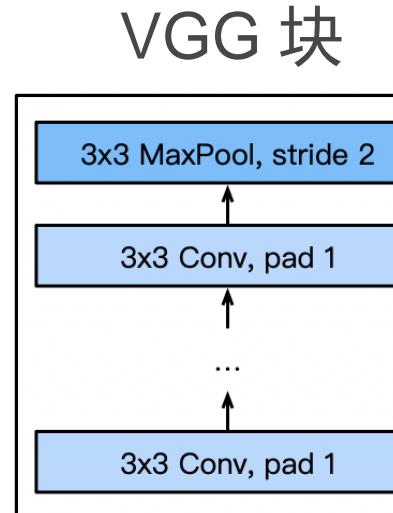
VGG

- AlexNet 比 LeNet 更深入更大，以获得更强性能
- 怎么更大更深？
 - 选项
 - 更多稠密层(开销太大)
 - 更多的卷积层
 - 分组成块



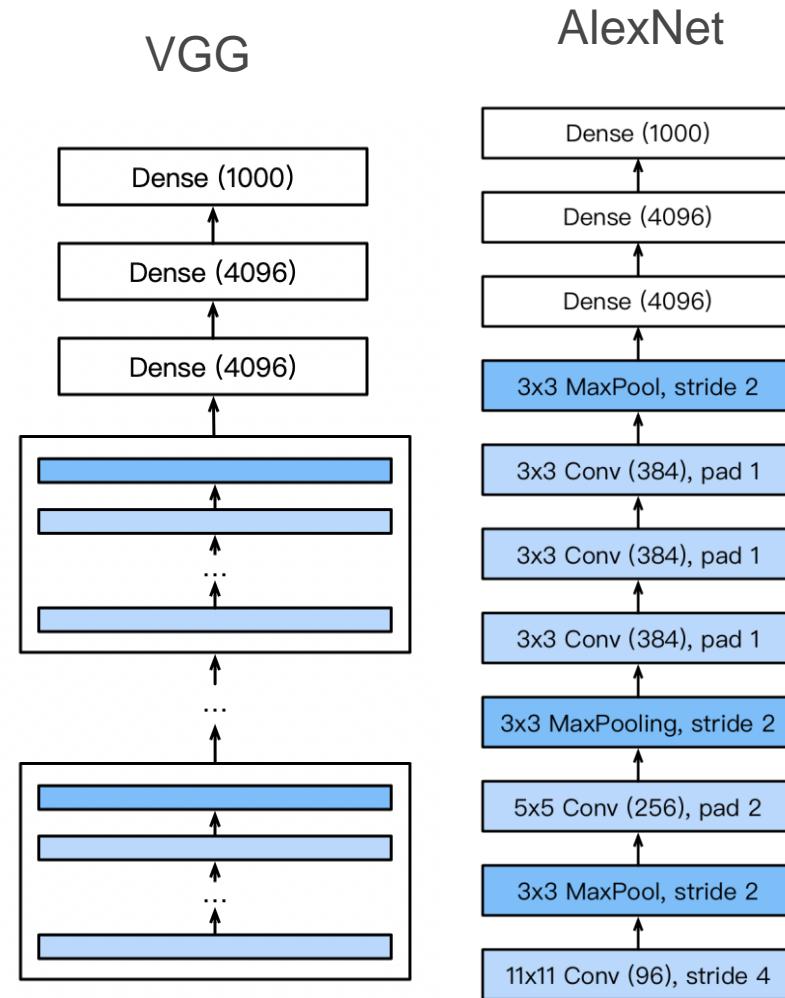
VGG 块

- 更深还是更宽?
 - 5x5 卷积
 - 3x3 卷积 (更多)
 - 更深和更窄更好
- VGG 块
 - 3x3 卷积 (填充=1)
(n层, m个通道)
 - 2x2 最大池化层
(步幅=2)



VGG 结构

- 多个VGG块后加稠密层
- 不同数目的重复VGG块，可获得不同的架构，例如VGG-16, VGG-19,

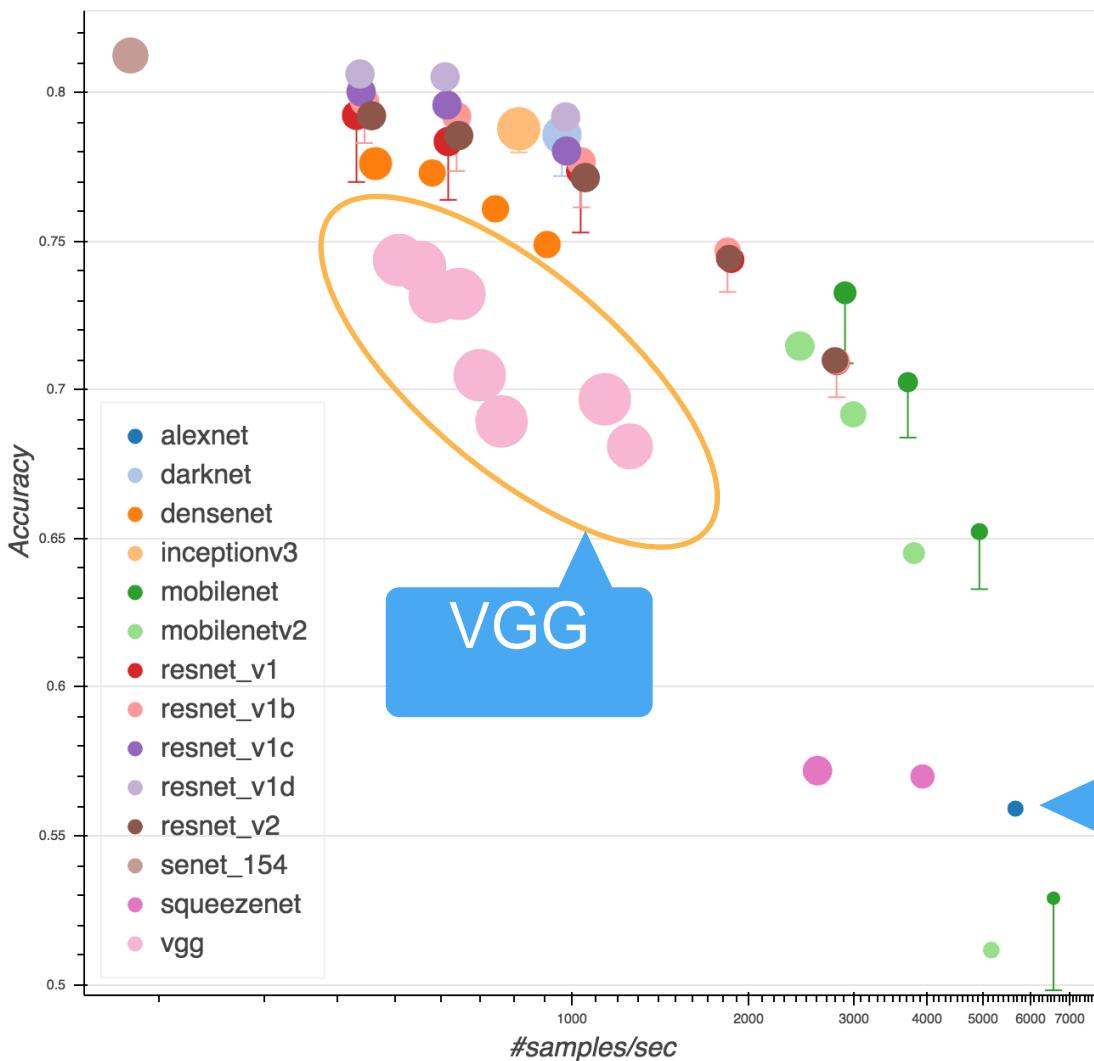


进程

- LeNet (1995)
 - 2 卷积层 + 池化层
 - 2 隐含层
- AlexNet
 - 更大更深的 LeNet
 - ReLu 激活, 丢弃法, 预处理
- VGG
 - 更大更深的 AlexNet (重复的 VGG 块)

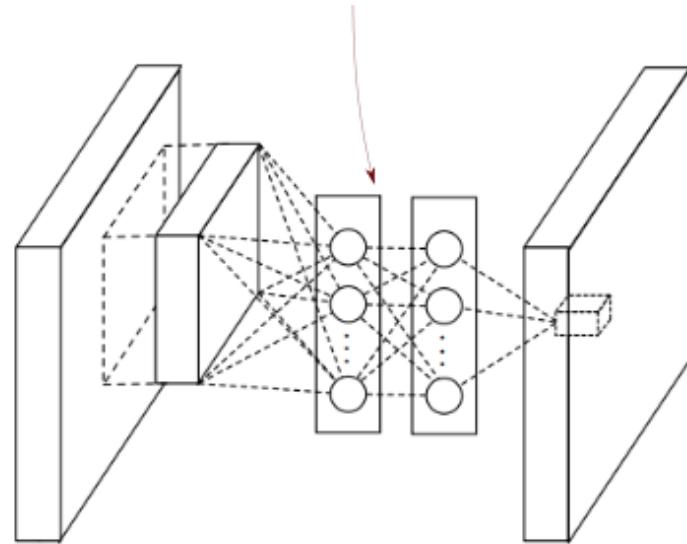
GluonCV Model Zoo

gluon-cv.mxnet.io/model_zoo/classification.html

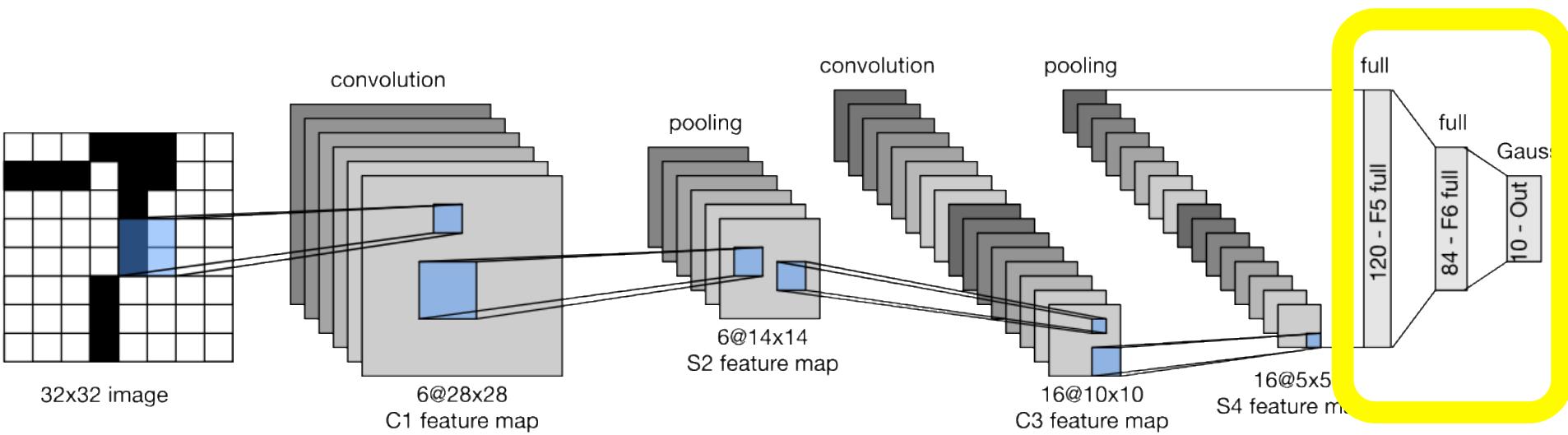


网络中的网络

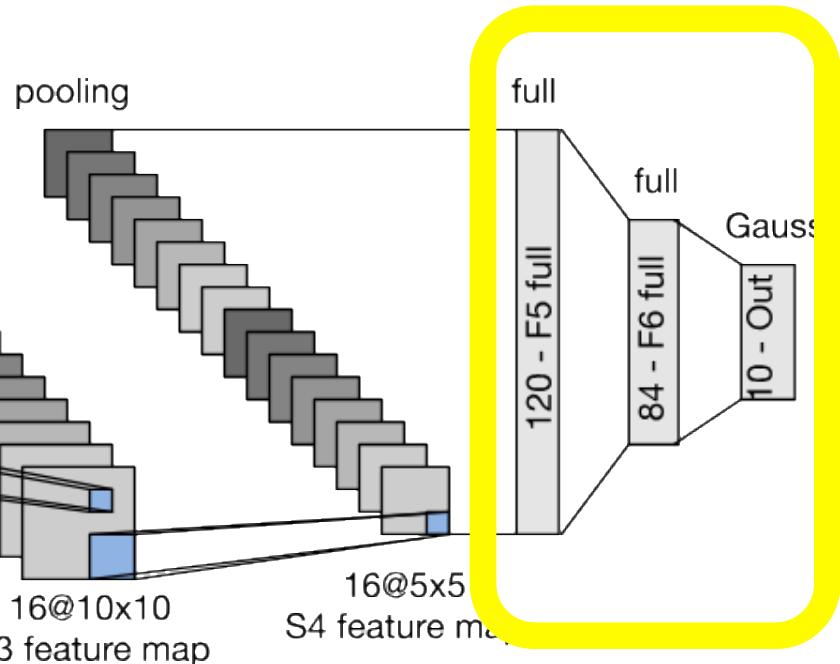
Non linear mapping introduced by mlpconv layer consisting of multiple fully connected layers with non linear activation function.



最后一层的诅咒



最后一层的诅咒



- 卷积层需要相对较少的参数
- $c_i \times c_o \times k^2$
- 最后一层(稠密层)对于n个类的
需 要许多参数

$$c \times m_w \times m_h \times n$$

- LeNet $16 \times 5 \times 5 \times 120 = 48k$
- AlexNet $256 \times 5 \times 5 \times 4096 = 26M$
- VGG $512 \times 7 \times 7 \times 4096 = 102M$

VGG 参数

sequential1 output shape: (1, 64, 112, 112)

sequential2 output shape: (1, 128, 56, 56)

sequential3 output shape: (1, 256, 28, 28)

sequential4 output shape: (1, 512, 14, 14)

sequential5 output shape: (1, 512, 7, 7)

dense0 output shape: (1, 4096)

dropout0 output shape: (1, 4096)

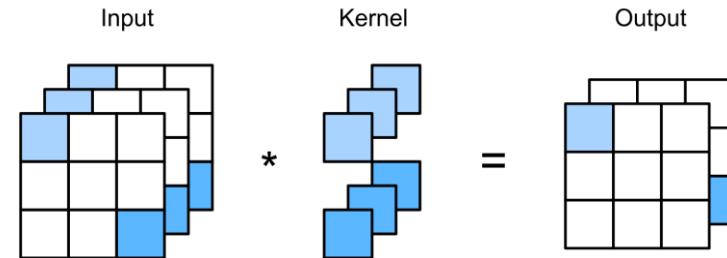
dense1 output shape: (1, 4096)

dropout1 output shape: (1, 4096)

dense2 output shape: (1, 10)

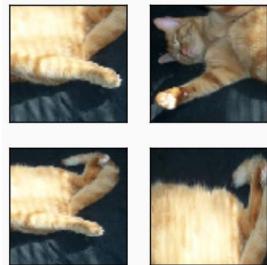
打破最后一层诅咒

- 关键理念
 - 丢弃稠密层的最后一层或者几层
 - 卷积和池化会降低分辨率（例如，步幅为2会降低4倍的分辨率）
- 实施细节
 - 逐步降低分辨率
 - 增加通道数量
 - 使用 1×1 卷积
- 最后应用全局平均池化

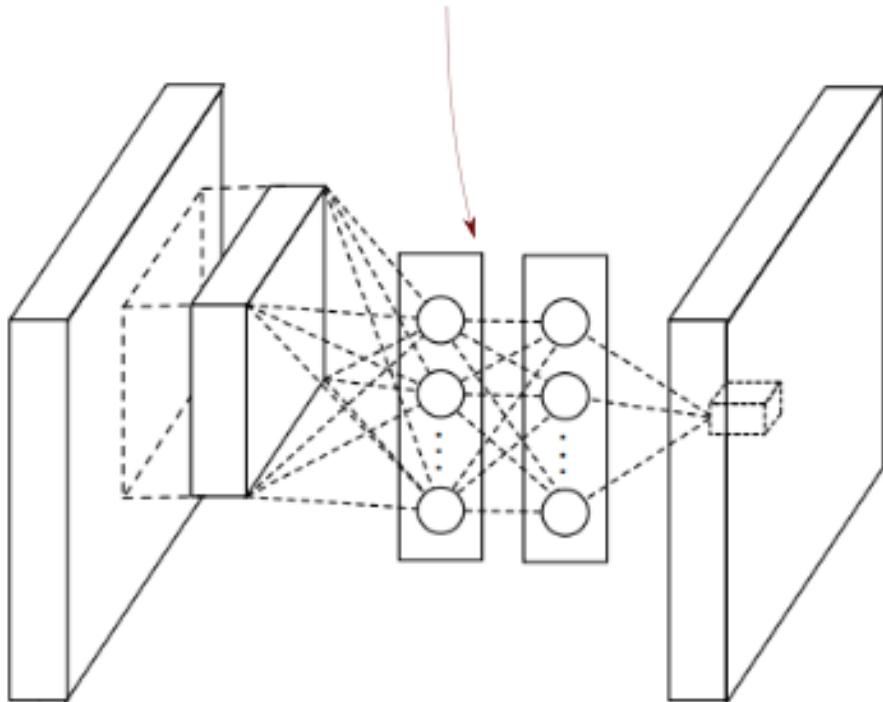


为什么用 1×1 卷积？

- 具有n个通道的极端情况
 1×1 图像
- 相当于MLP
- 推理时：池化增强平移稳定性（例如 5×5 ）

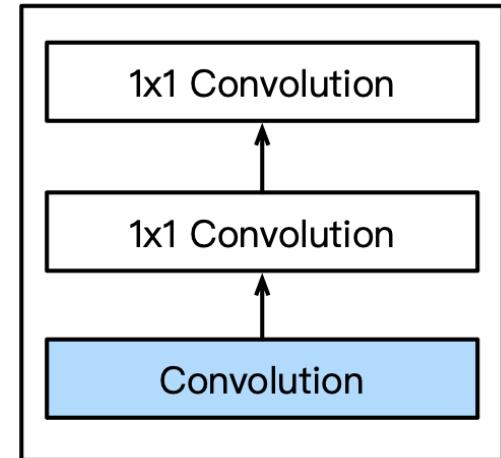


Non linear mapping introduced by mlpconv layer consisting of multiple fully connected layers with non linear activation function.

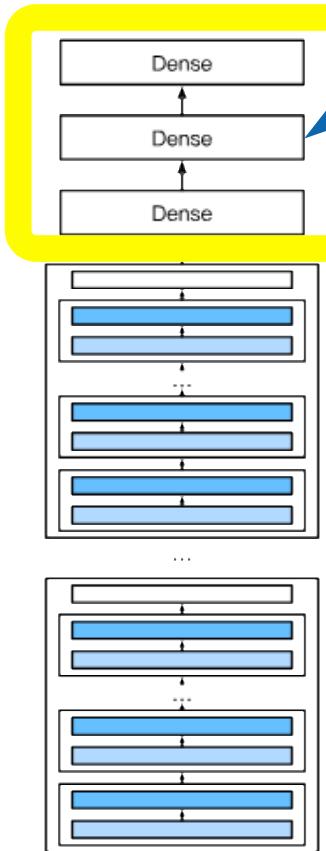
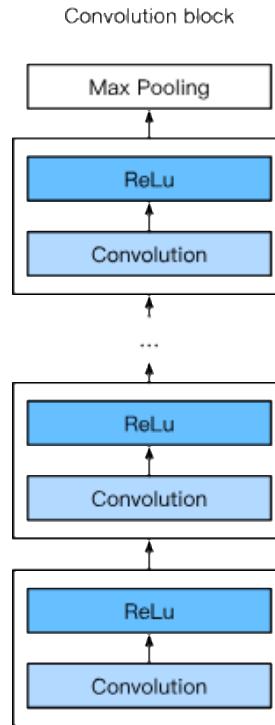


NiN 块

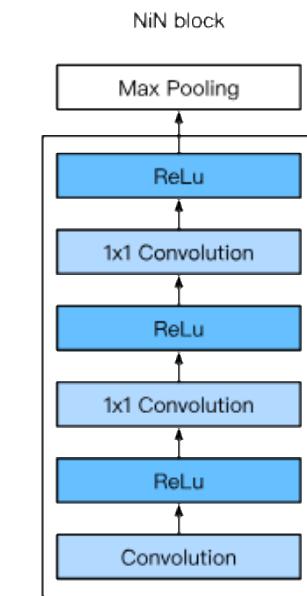
- 卷积层
 - 超参数：卷积核大小，步幅和填充
- 接下来是两个 1×1 卷积层
 - 步幅1
 - 无填充
 - 与第一层输出通道相同
 - 充当稠密层



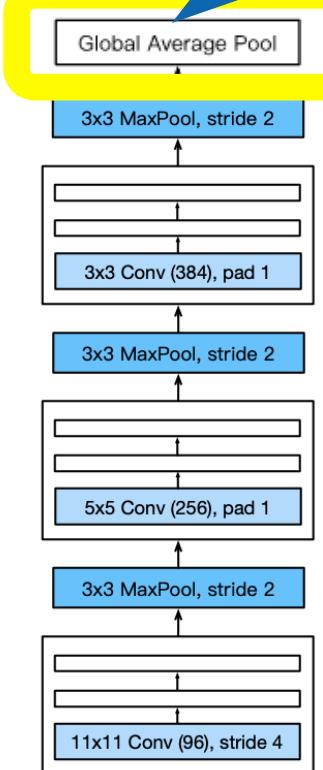
NiN 网络



VGG 网络

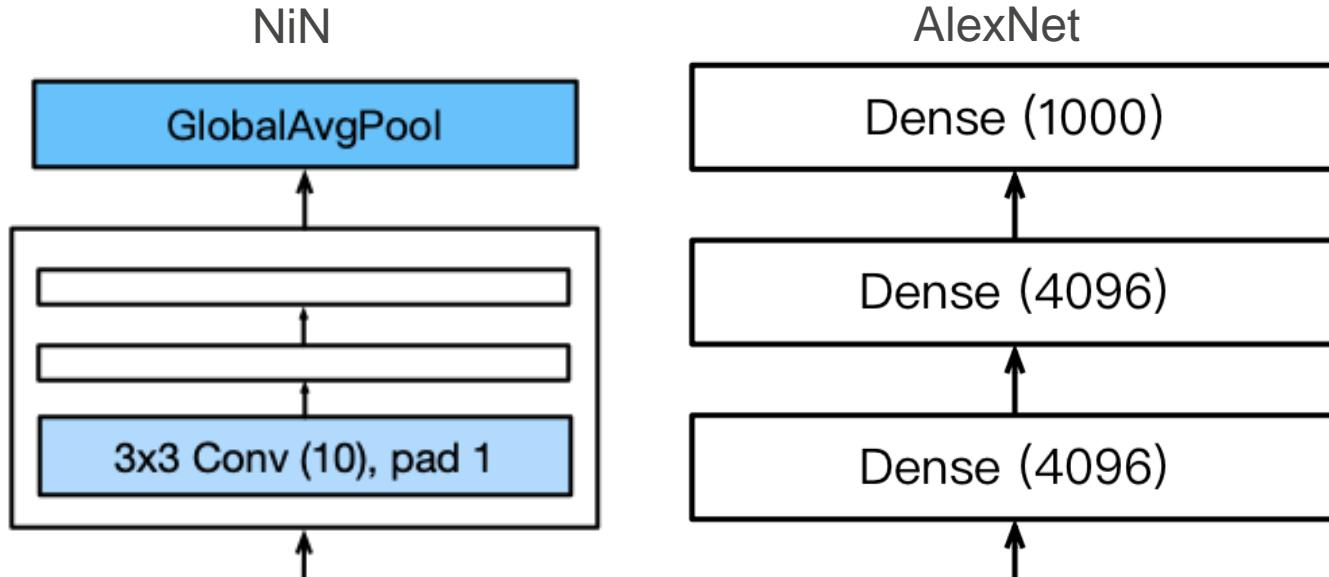


NiN 网络



NiN 最后一层

- 用 NiN 块替换了 AlexNet 的稠密层
- 输出：全局平均池化层



NIN 总结

- 逐步降低图像分辨率
- 增加通道数量
- 应用全局平均退化

sequential1 output shape: (96, 54, 54)

pool0 output shape: (96, 26, 26)

sequential2 output shape: (256, 26, 26)

pool1 output shape: (256, 12, 12)

sequential3 output shape: (384, 12, 12)

pool2 output shape: (384, 5, 5)

dropout0 output shape: (384, 5, 5)

sequential4 output shape: (10, 5, 5)

pool3 output shape: (10, 1, 1)

flatten0 output shape: (10)



总结

- **LeNet** (第一个卷积神经网络)
- **AlexNet**
 - 升级版的 LeNet
 - ReLu 激活, 丢弃法, 平移不变性
- **VGG**
 - 升华版的 AlexNet
 - 重复的 VGG 块
- **NiN**
 - 1x1 卷积 + 使用全局池化层替代稠密层

动手学深度学习

13.Inception, 批量归一化 和 残差网络 (ResNet)

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

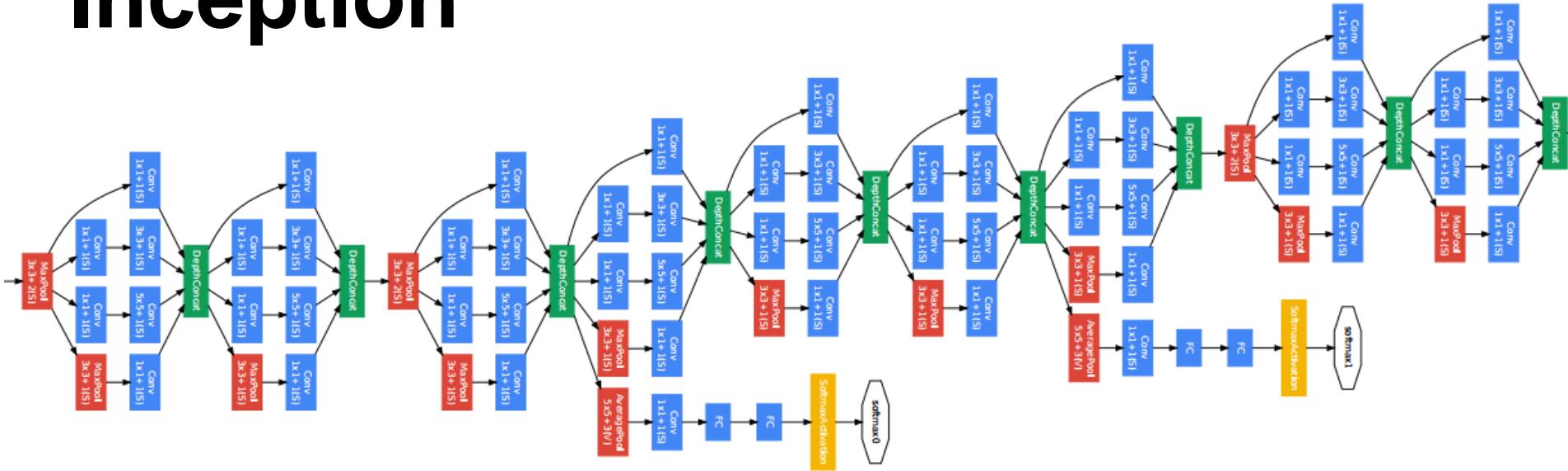
教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/resnet.html>

概要

- **Inception**
 - 卷积的不均匀混合（不同深度）
 - 批量归一正则化
- **ResNet**
 - 泰勒展开式
 - 残差网络（**ResNext**） 分解卷积
- **Zoo**

稠密连接网络（DenseNet），ShuffleNet，可分解卷积层，…

Inception



选最合适的卷积 ...

1x1

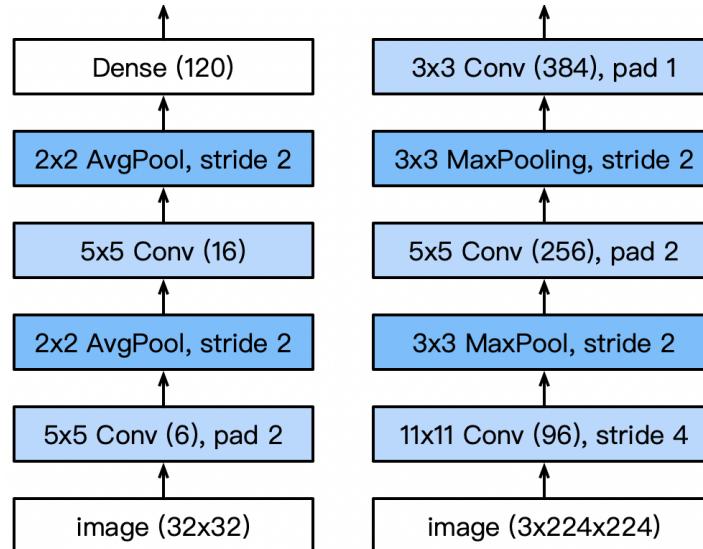
3x3

5x5

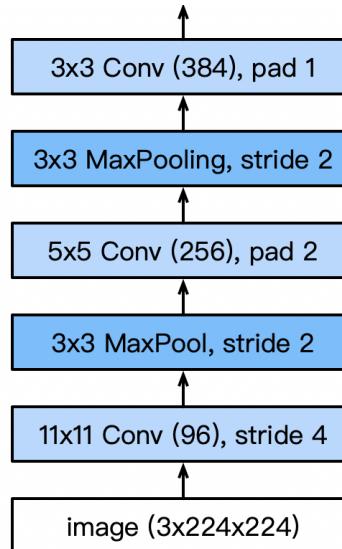
最大退化

许多 1x1

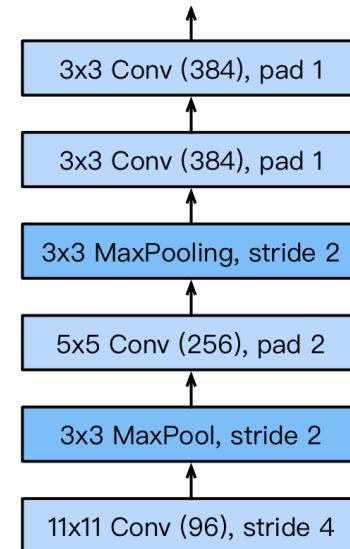
LeNet



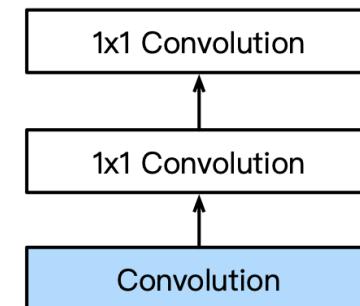
AlexNet



VGG



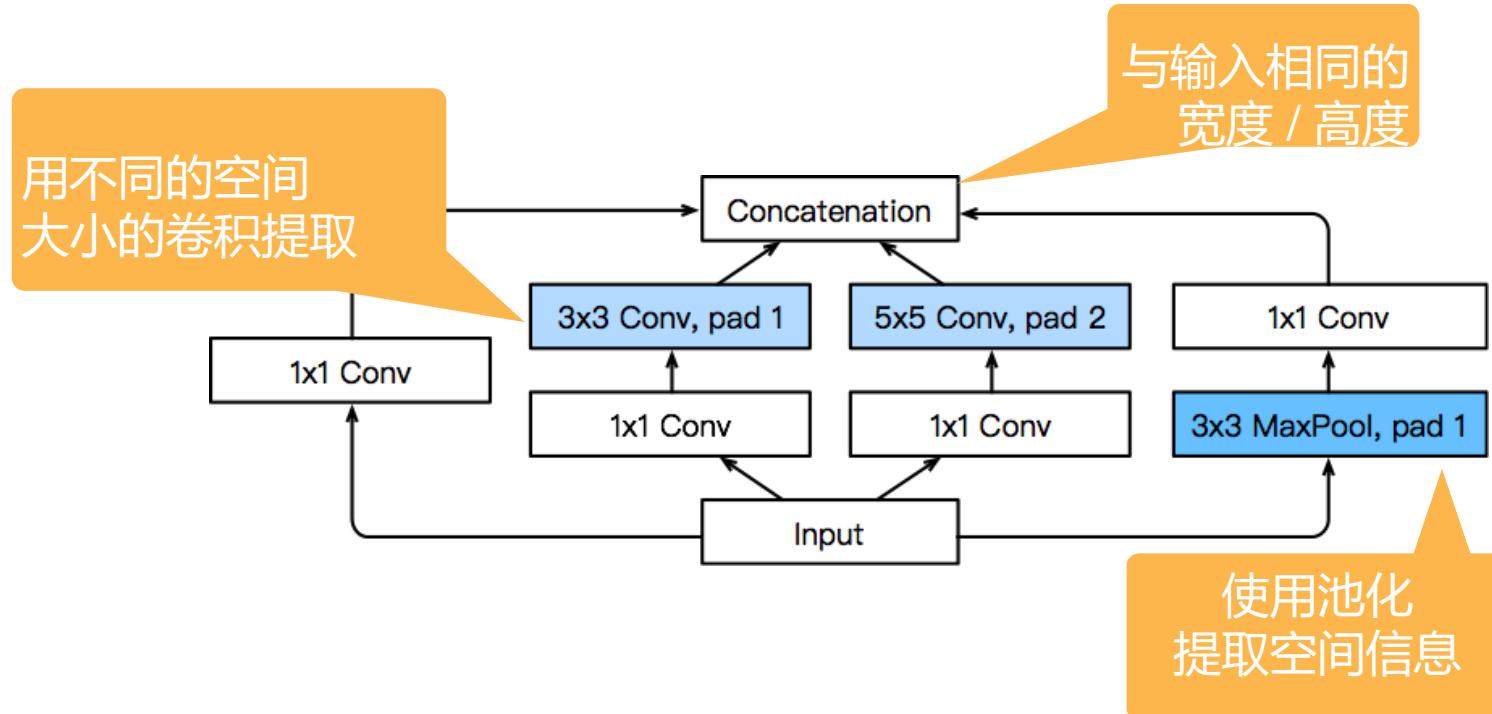
NiN



干嘛选呢？都用就是了。

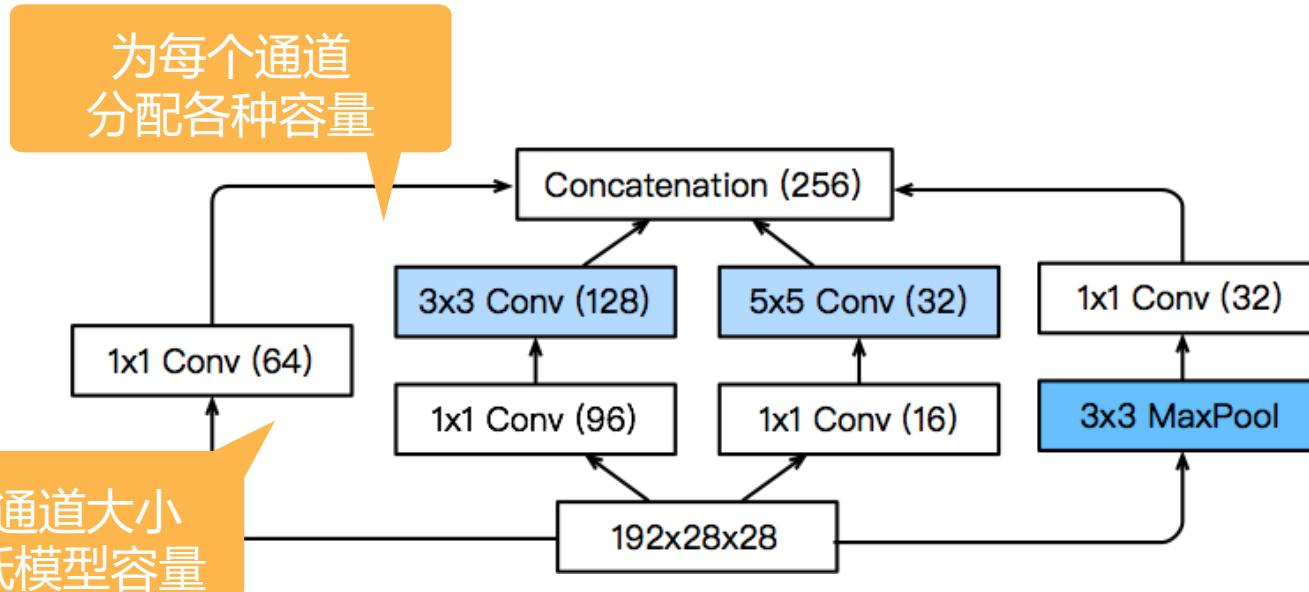
Inception 块

4个路径从不同方面提取信息，然后拼接作为输出通道



Inception 块

(第一个初始块) 指定的通道大小

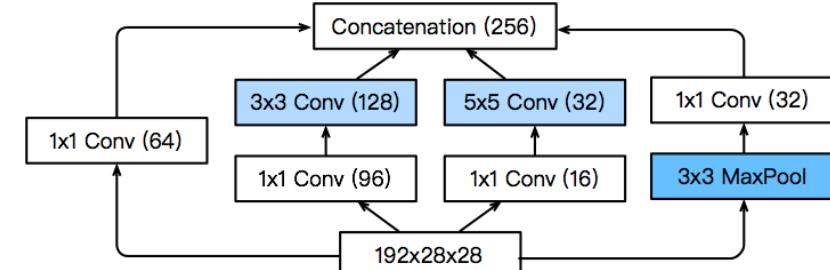


Inception 块

与单个3x3或5x5卷积层相比，初始块具有更少的参数和更低的计算复杂度

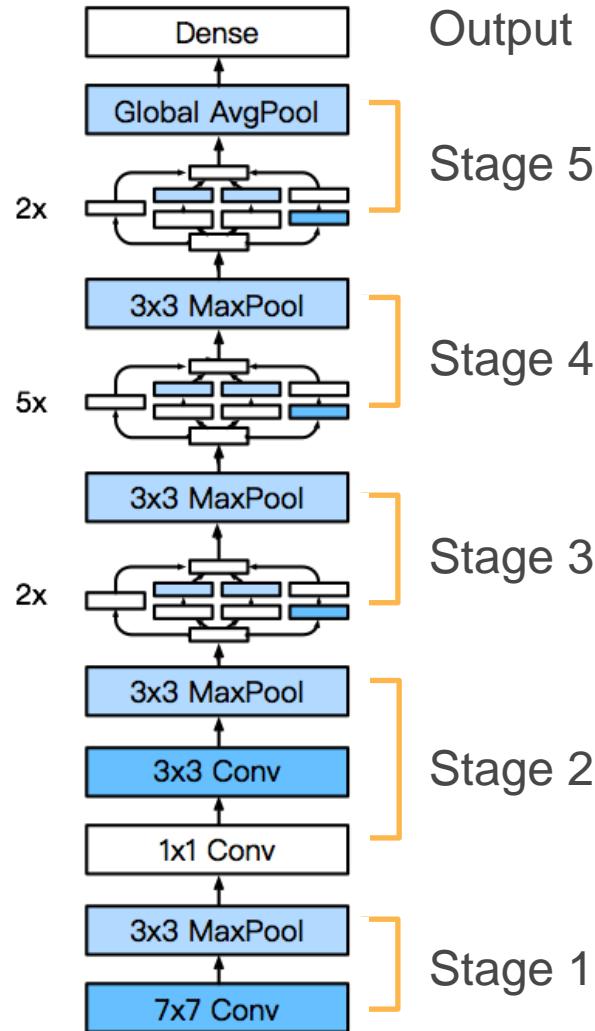
- 不同功能混合（多样的功能类）
- 卷积核计算高效（良好的泛化）

	# 参数	浮点运算 FLOPS
Inception	0.16 M	128 M
3x3 卷积	0.44 M	346 M
5x5 卷积	1.22 M	963 M



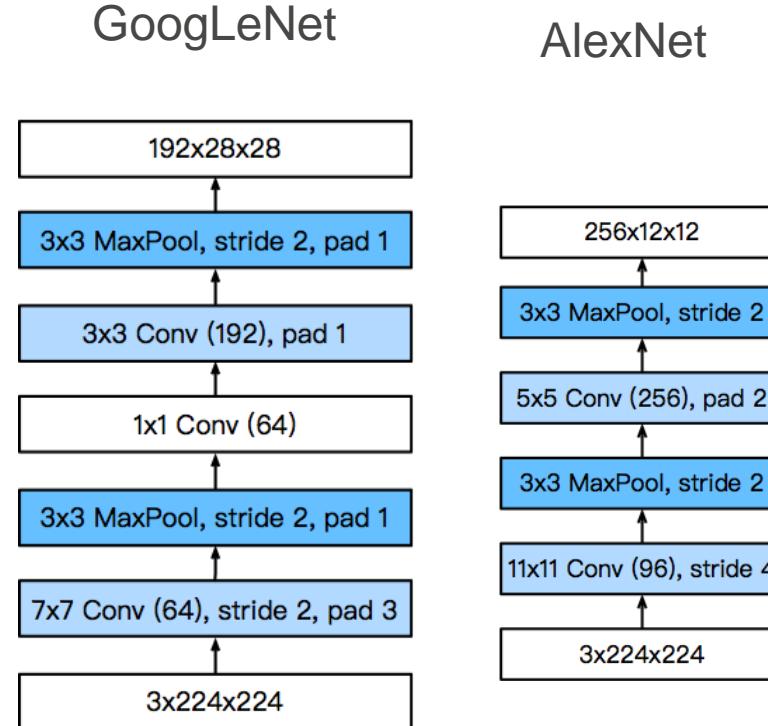
GoogLeNet

- 5 个阶段
- 9 个 Inception 块



阶段 1 & 2

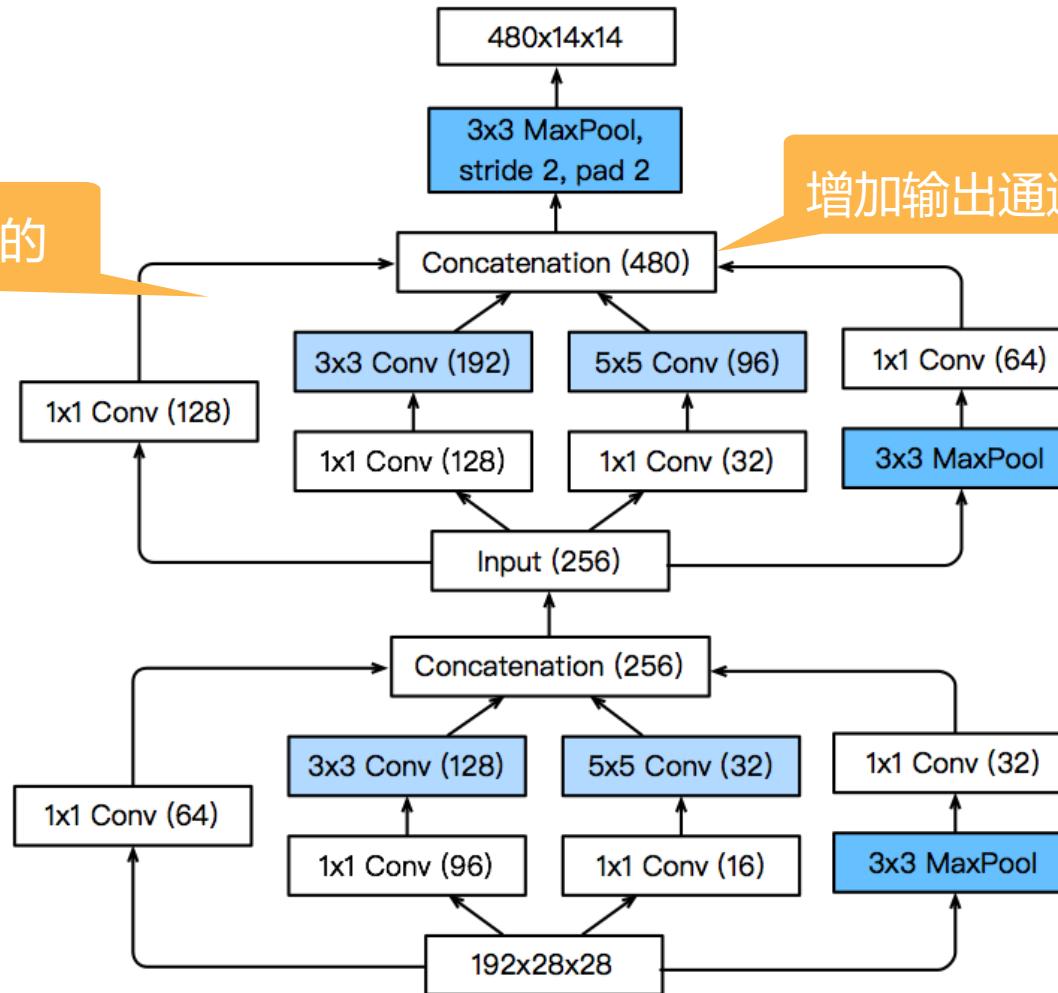
- 由于更多层：
 - 更小的内核
 - 更小的输出通道



阶段 3

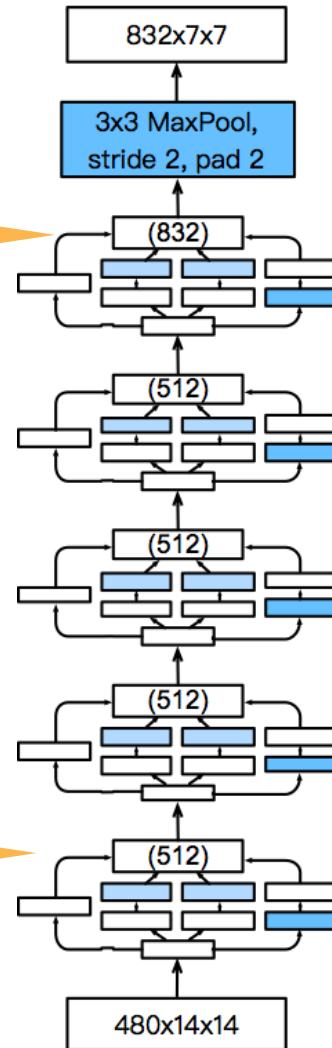
通道分配是不同的

增加输出通道

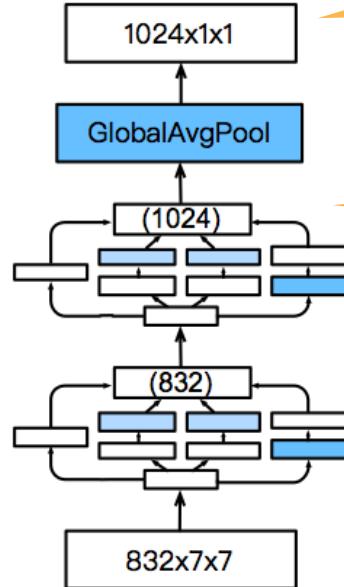


阶段 4 & 5

增加输出通道



1024个输出通道



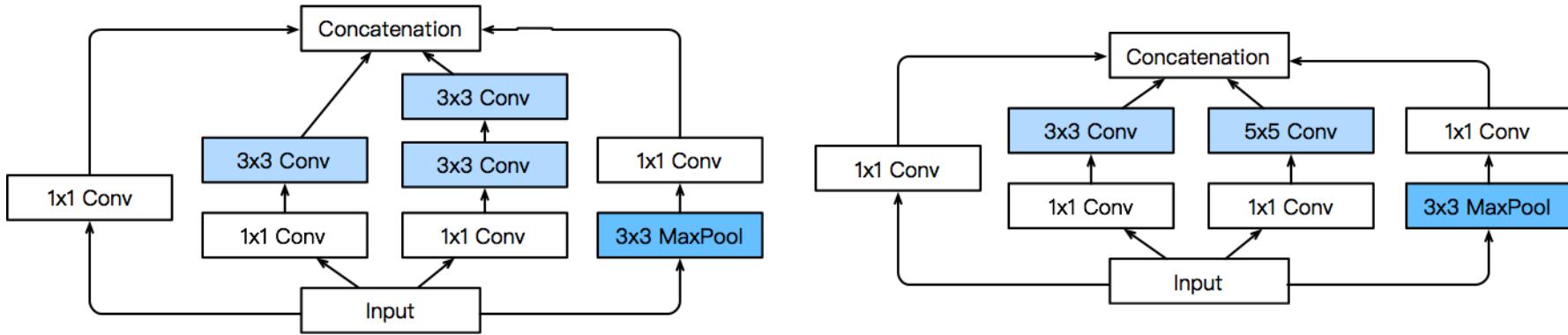
增加输出通道

增加输出通道

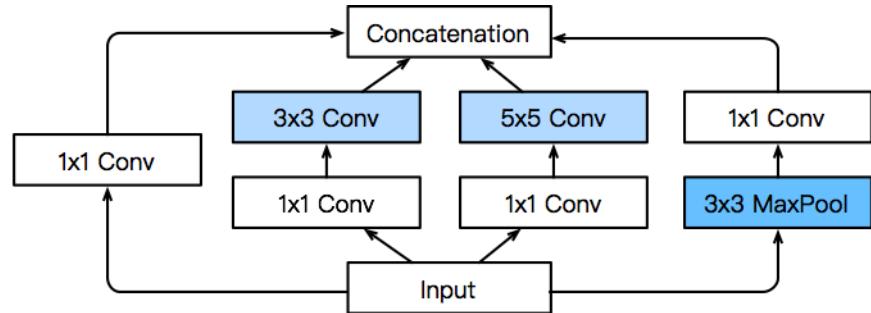
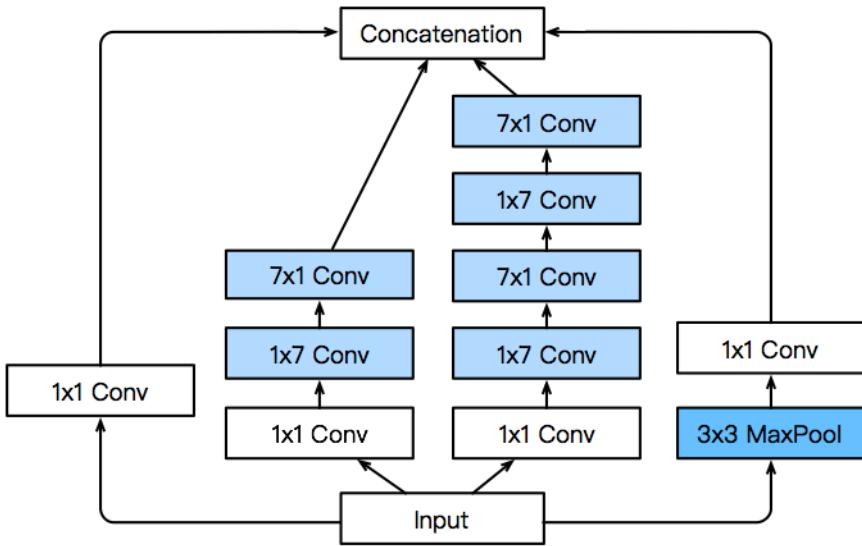
许多种类的 Inception 网络

- Inception-BN (v2) - 添加批量归一化
- Inception-V3 - 修改了初始块
 - 用多个 3×3 卷积替换 5×5
 - 用 1×7 和 7×1 卷积替换 5×5
 - 用 1×3 和 3×1 卷积替换 3×3
 - 通常用更深的堆
- Inception-V4 - 添加残差块连接

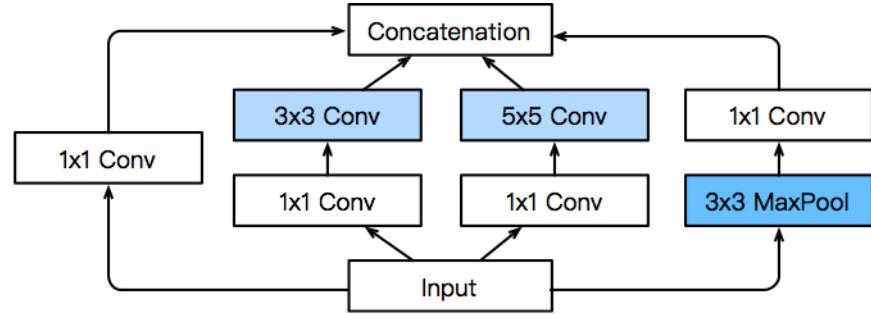
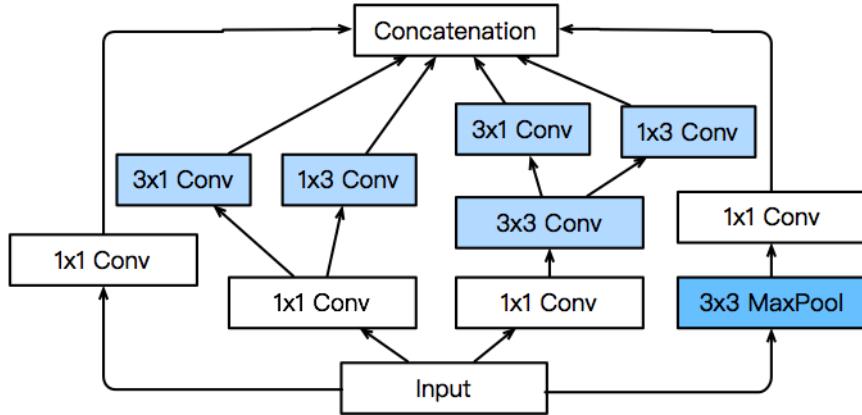
Inception V3 块 - 阶段 3

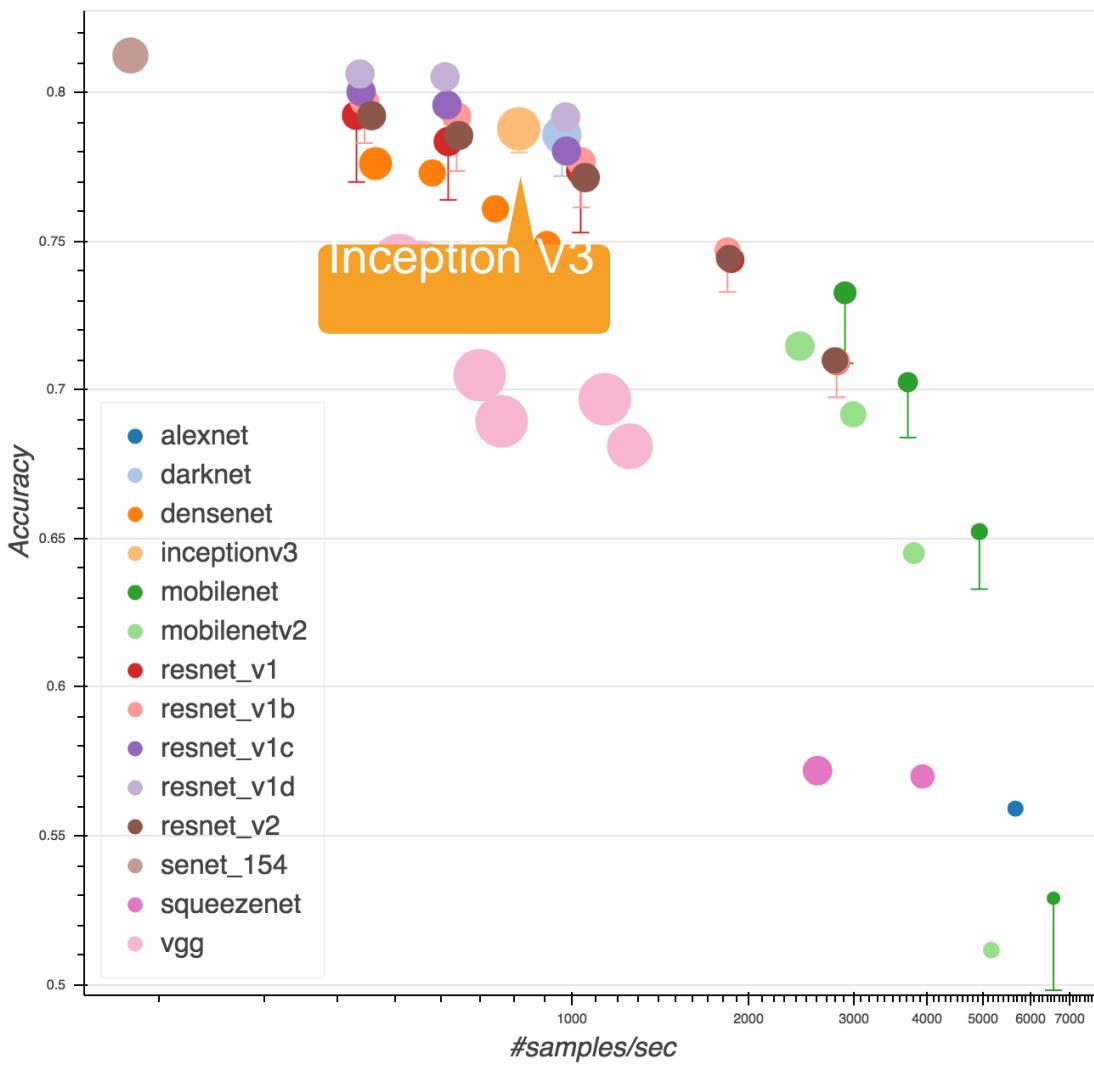


Inception V3 块 - 阶段 4



Inception V3 块 - 阶段 5





GluonCV 模型“动物园”
https://gluon-cv.mxnet.io/model_zoo/classification.html



批量归一化

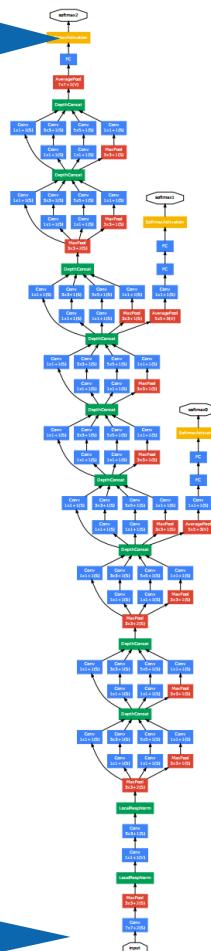
- 损失发生在最后一层
 - 最后一层可快速学习
- 数据插入在第一层（底层）
 - 底层变化 - 一切都变化
 - 最后一层需要多次重新学习
 - 收敛缓慢
- 这就像协变量偏移

我们可以避免在学习第一层时
改变最后一层吗？

损失

数据

D2L.ai



批量归一化

损失

- 学习第一层时，我们可以避免更改最后一层吗？
 - 修正均值和方差

$$\mu_B = \frac{1}{|B|} \sum_{i \in B} x_i \text{ and } \sigma_B^2 = \frac{1}{|B|} \sum_{i \in B} (x_i - \mu_B)^2 + \epsilon$$

- 单独调整：

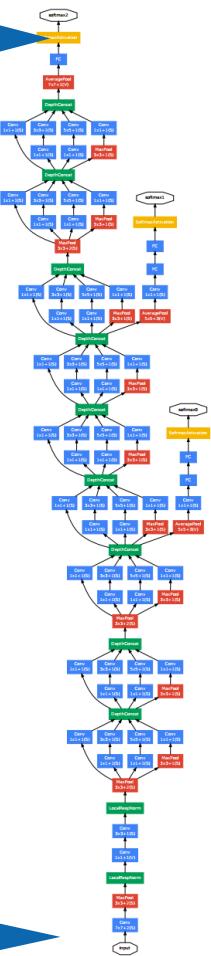
均值

$$x_{i+1} = \gamma \frac{x_i - \mu_B}{\sigma_B} + \beta$$

方差

数据

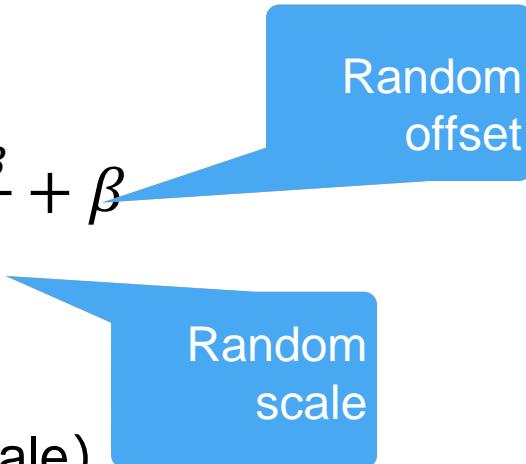
D2L.ai



这是最初的动机.....

批量归一化

- 并没有真正减少协变量的变化 (Lipton et al., 2018)
- 通过注入噪声进行正规化

$$x_{i+1} = \gamma \frac{\hat{x}_i - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$


- 每个小批量随机偏移 (shift)
- 每个小批量的随机转换比例 (scale)
- 无需与丢弃法共用 (两者都是正则化控制)
- 理想的小批量大小为 64-256

细节

gluon.nn.BatchNorm(...)

- **稠密层**

对所有神经元进行批量归一化

- **卷积层**

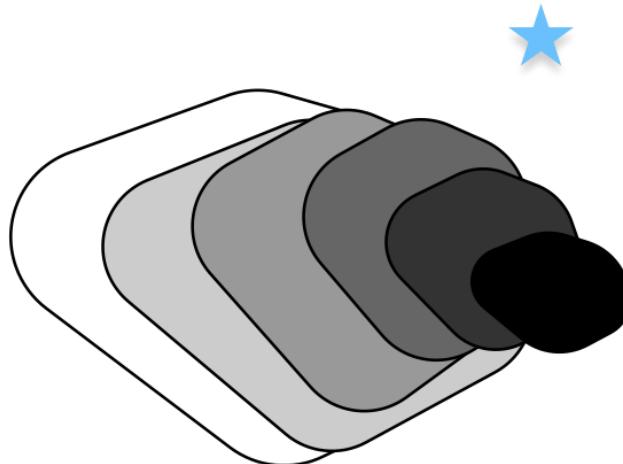
每个通道一次批量归一化

- **计算每个小批量的新均值和方差**

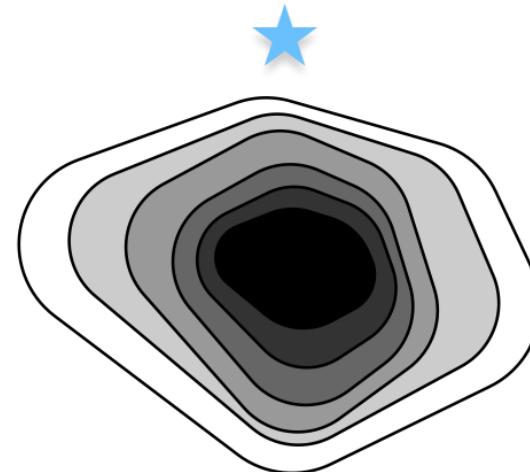
- 有效地充当正则化

- 最佳的批量大小约为128 (与许多机器并行训练)

残差网络 (ResNet)

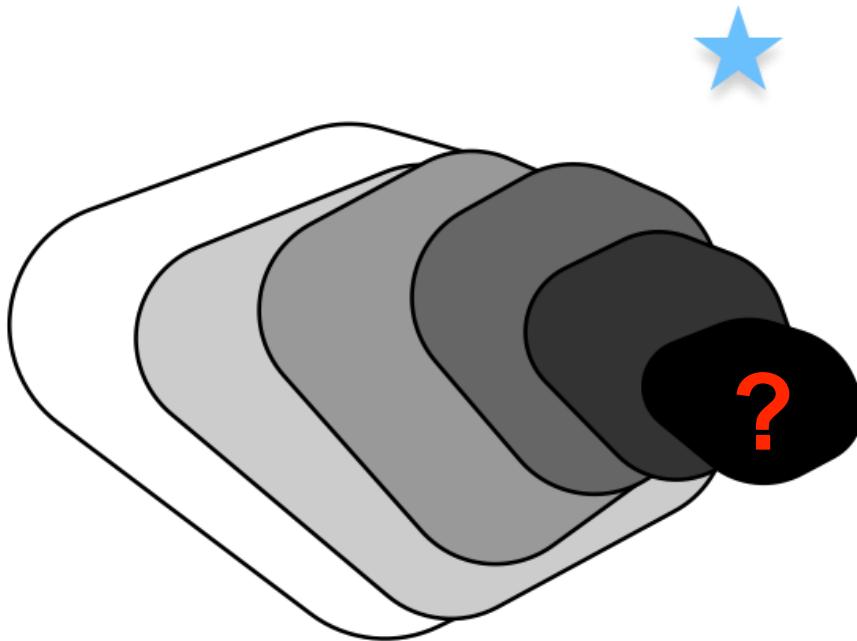


generic function classes

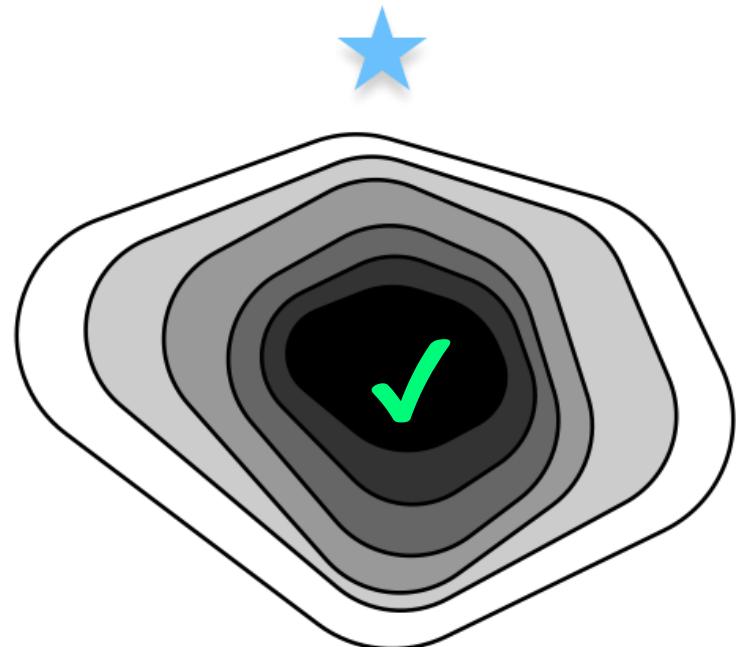


nested function classes

添加层会提高准确性吗?



generic function classes

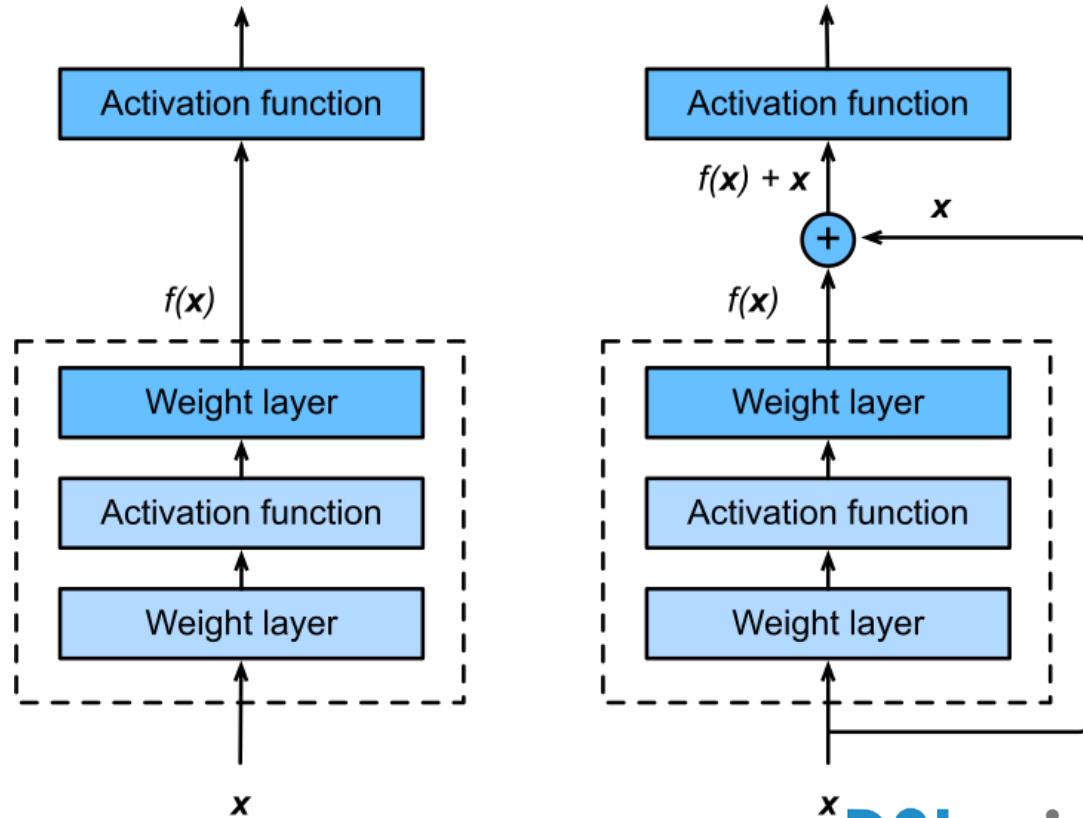


nested function classes

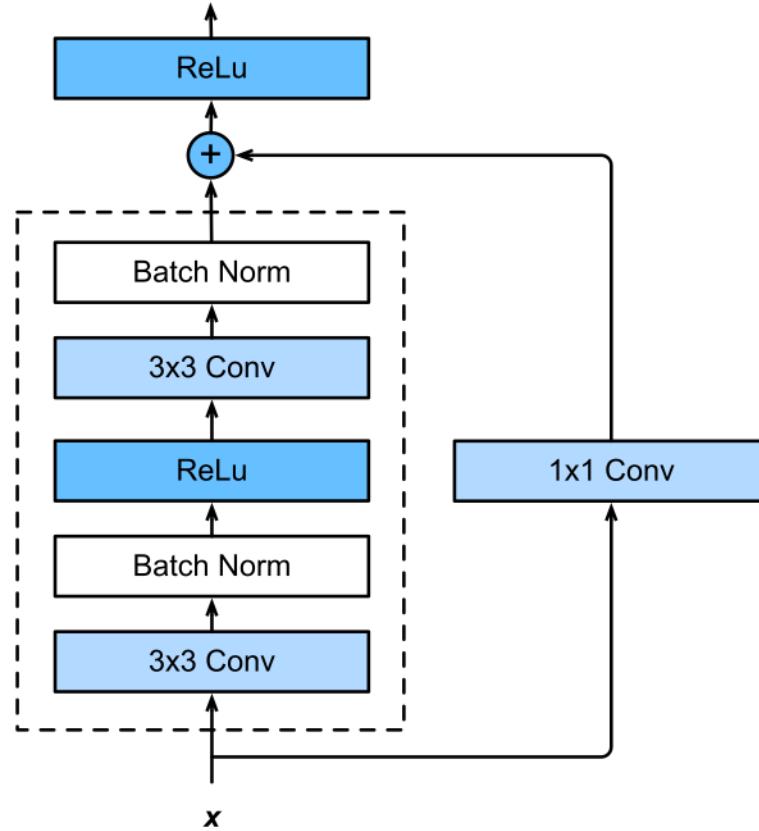
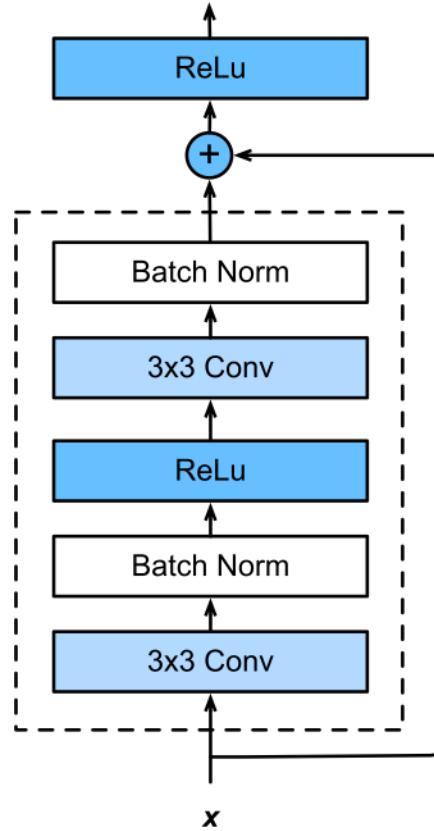
残差网络 (ResNet)

- 添加层会更改原特征类
- 我们想要“加”到原函数中
- “泰勒展开”的参数

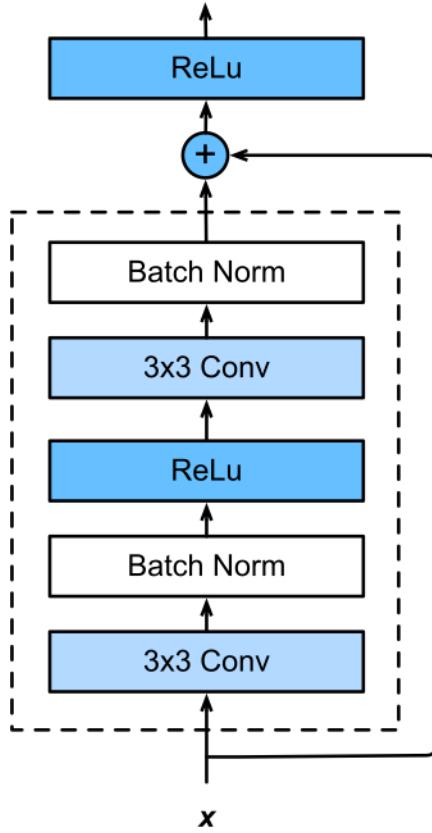
$$f(x) = x + g(x)$$



残差块



残差块



```
def forward(self, X):
```

```
    Y = self.bn1(self.conv1(X))
```

```
    Y = nd.relu(Y)
```

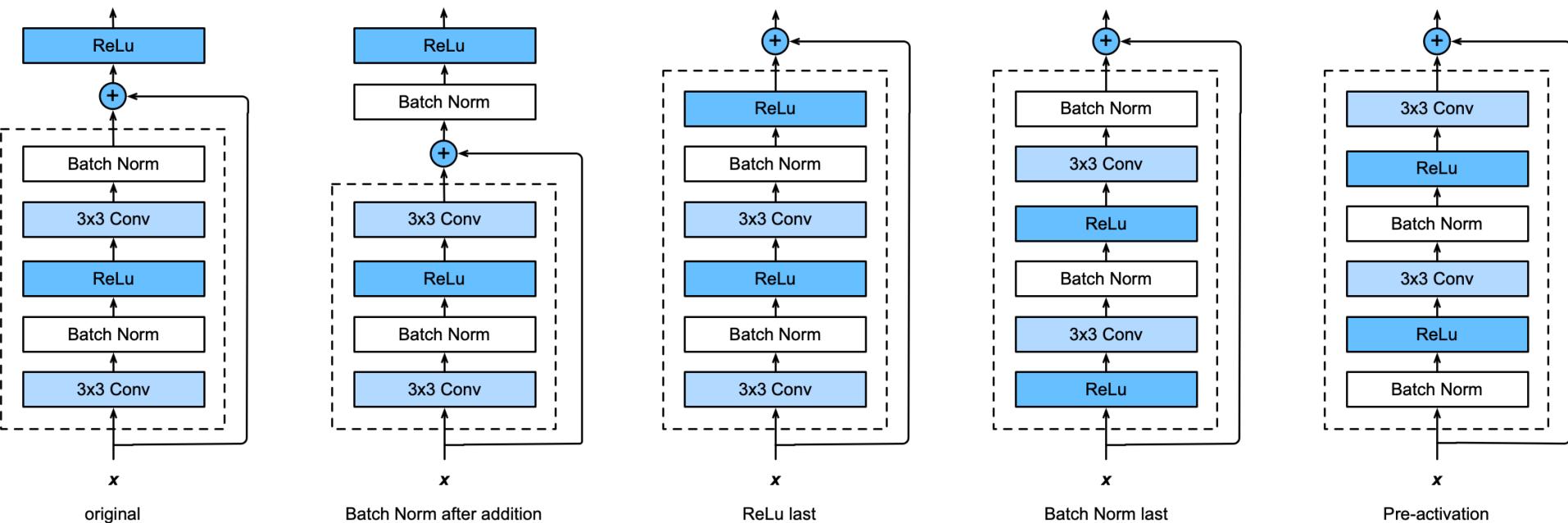
```
    Y = self.bn2(self.conv2(Y))
```

```
    if self.conv3:
```

```
        X = self.conv3(X)
```

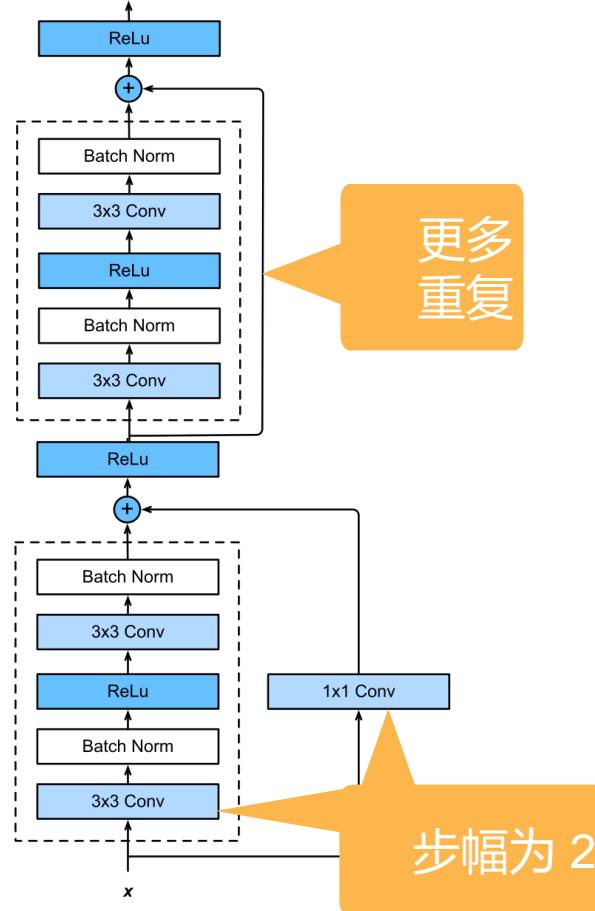
```
    return nd.relu(Y + X)
```

不同的残差块



尝试每一个排列

残差块

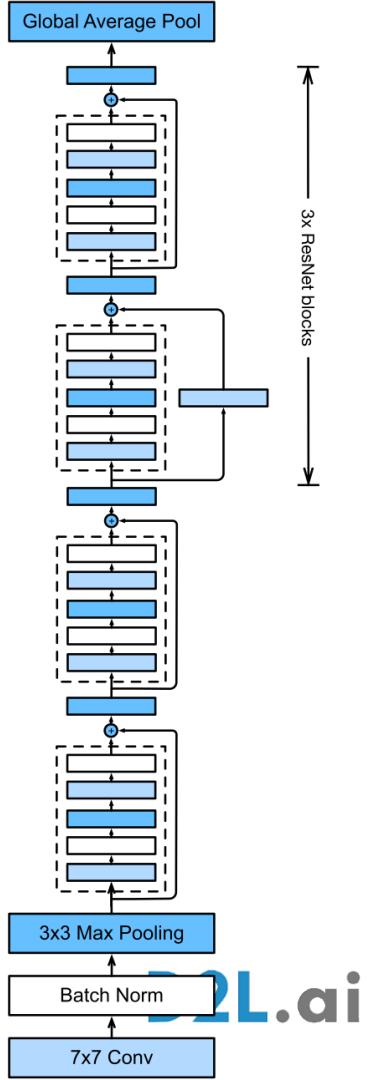


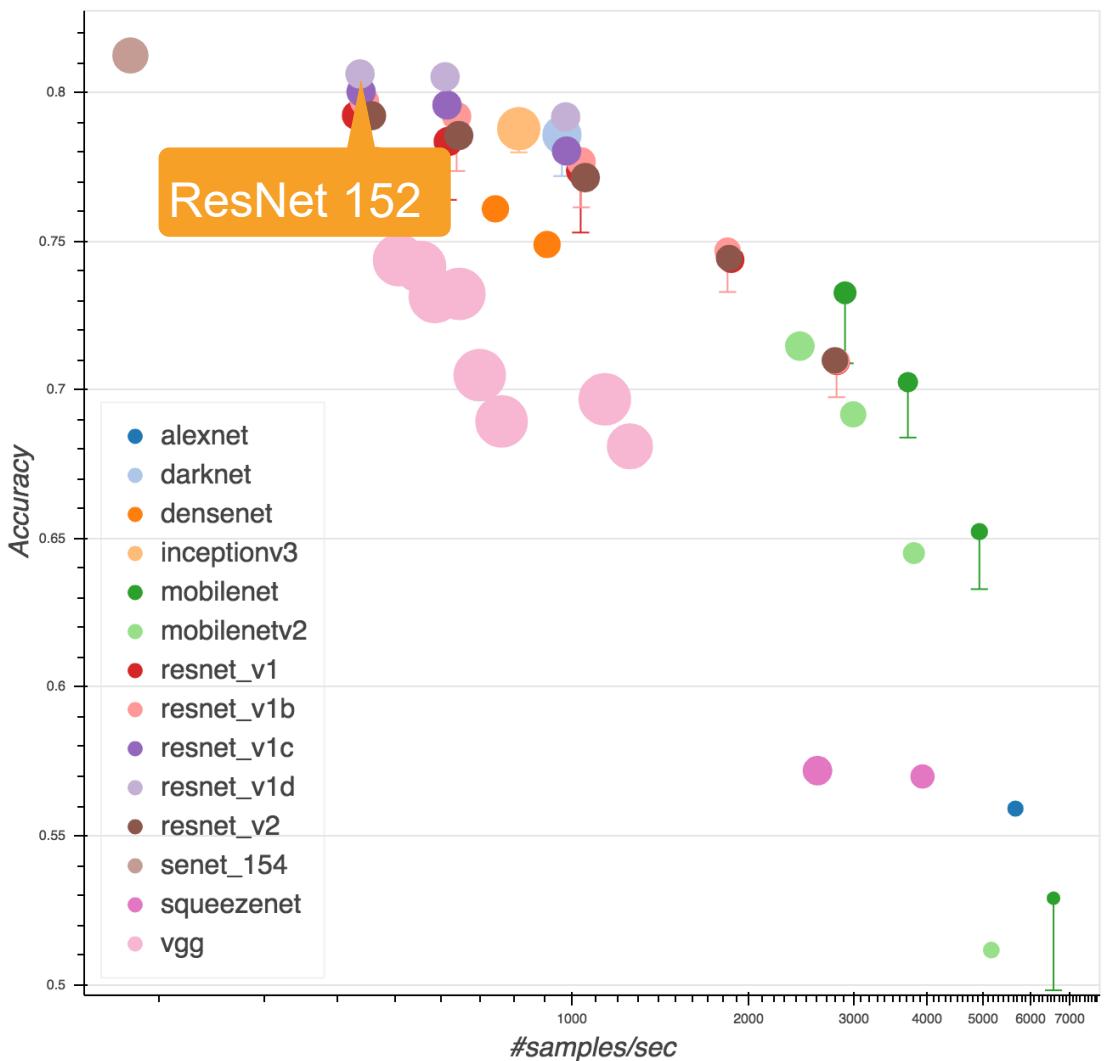
- 每个模块进行降采样 (步幅 = 2)
- 强制实现每个模块中一些非常见，非线性的函数 (通过1x1卷积)
- 堆积在块中

```
blk = nn.Sequential()  
for i in range(num_residuals):  
    if i == 0 and not first_block:  
        blk.add(Residual(num_channels,  
                         use_1x1conv=True, strides=2))  
    else:  
        blk.add(Residual(num_channels))
```

残差网络 (ResNet)

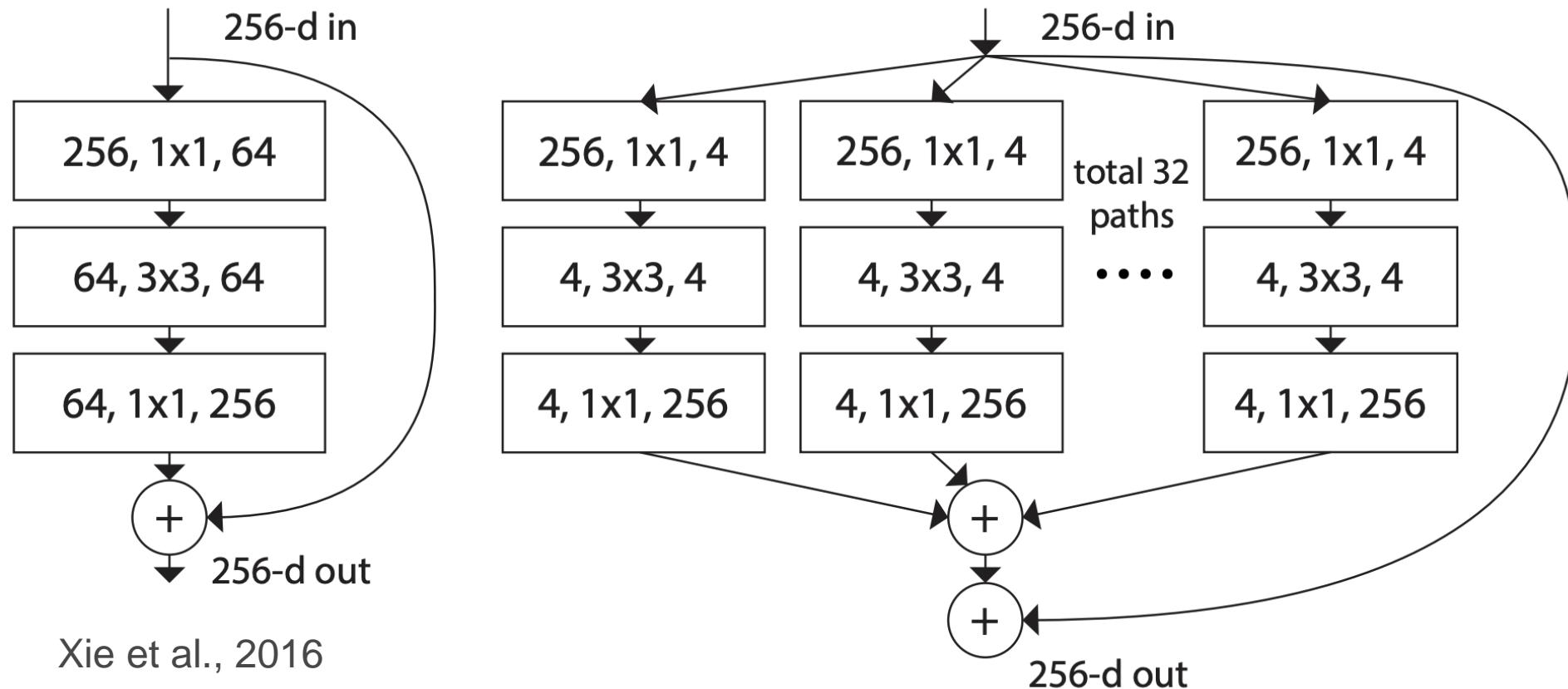
- 相同块结构，例如与VGG 或 GoogleNet使用块结构
- 残差块连接可增加表现力
- 池化 / 步幅 - 减少维度
- 批量归一化 - 容量控制
-大规模训练.....





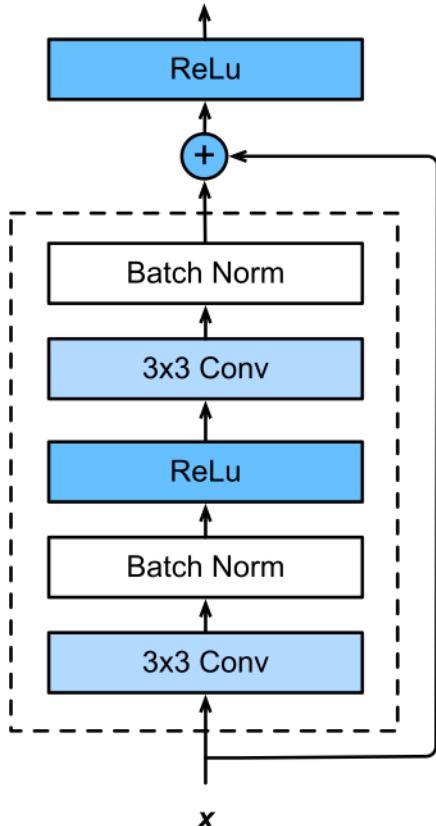
GluonCV 模型“动物园”
https://gluon-cv.mxnet.io/model_zoo/classification.html

残差网络 (ResNet)



Xie et al., 2016

降低卷积的成本



- 参数

$$k_h \cdot k_w \cdot c_i \cdot c_o$$

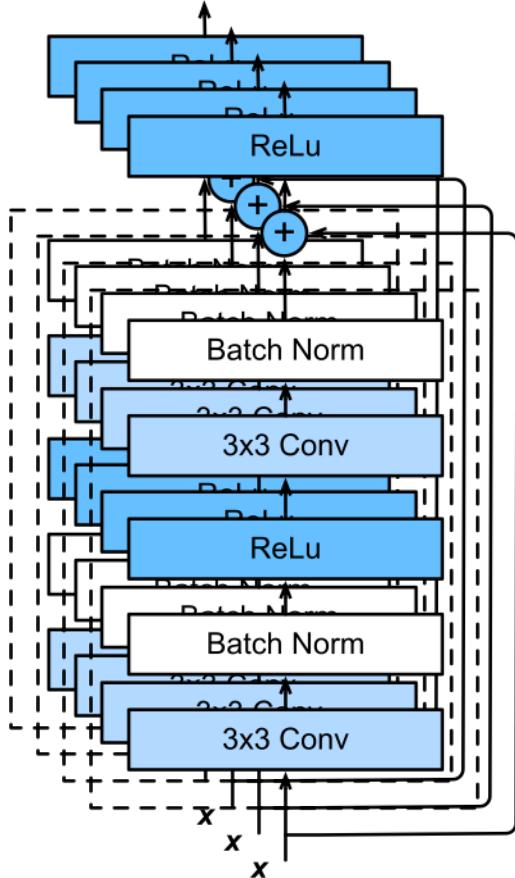
- 计算

$$m_h \cdot m_w \cdot k_h \cdot k_w \cdot c_i \cdot c_o$$

- 切割卷积 (Inception v4) , 例如 3x3 vs. 1x5 和 5x1
- 分解通道 (仅在内部)

$$m_h \cdot m_w \cdot k_h \cdot k_w \cdot \frac{c_i}{b} \cdot \frac{c_o}{b} \cdot b$$

降低卷积的成本



- 参数

$$k_h \cdot k_w \cdot c_i \cdot c_o$$

- 计算

$$m_h \cdot m_w \cdot k_h \cdot k_w \cdot c_i \cdot c_o$$

- 切割卷积 (Inception v4) , 例如 3x3 vs. 1x5 和 5x1
- 分解通道 (仅在内部)

$$m_h \cdot m_w \cdot k_h \cdot k_w \cdot \frac{c_i}{b} \cdot \frac{c_o}{b} \cdot b$$

stage	output	ResNet-50	ResNeXt-50 (32×4d)
conv1	112×112	$7 \times 7, 64, \text{stride } 2$	$7 \times 7, 64, \text{stride } 2$
conv2	56×56	$3 \times 3 \text{ max pool, stride } 2$	$3 \times 3 \text{ max pool, stride } 2$
		$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128, C=32 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3	28×28	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256, C=32 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
conv4	14×14	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512, C=32 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
conv5	7×7	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 1024 \\ 3 \times 3, 1024, C=32 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		25.5×10^6	25.0×10^6
FLOPs		4.1×10^9	4.2×10^9

RexNeXt 成本

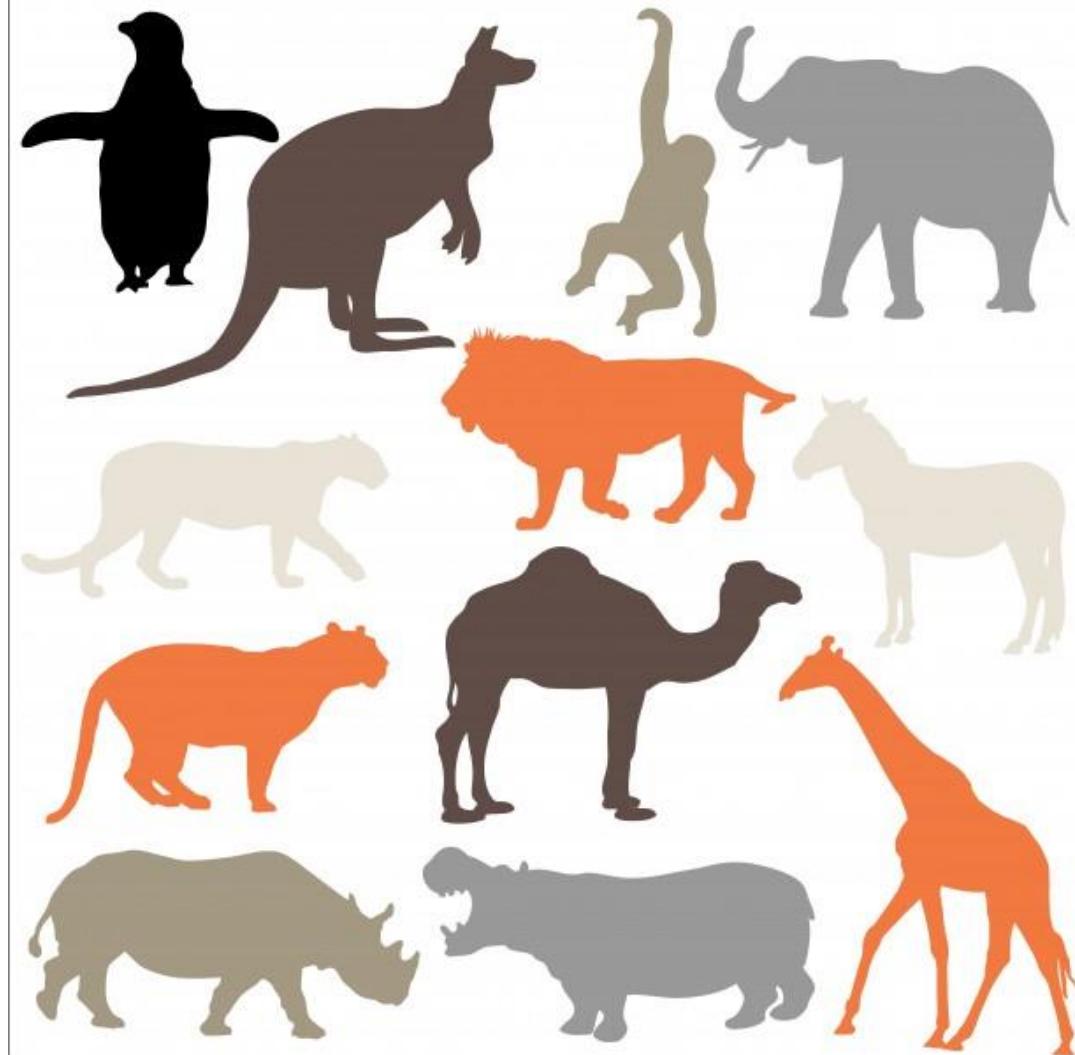
- 切割成 32 个子块
- 可以使用更多尺寸
- 精度更高

`nn.Conv2D(group_width=width,
kernel_size=3,
groups=cardinality)`

Xie et al., 2016

D2L.ai

更多模型



稠密连接网络 (DenseNet)

- DenseNet (Huang et al., 2016)
- ResNet结合 x 和 $f(x)$
- DenseNet 使用更高阶'泰勒系列'扩展

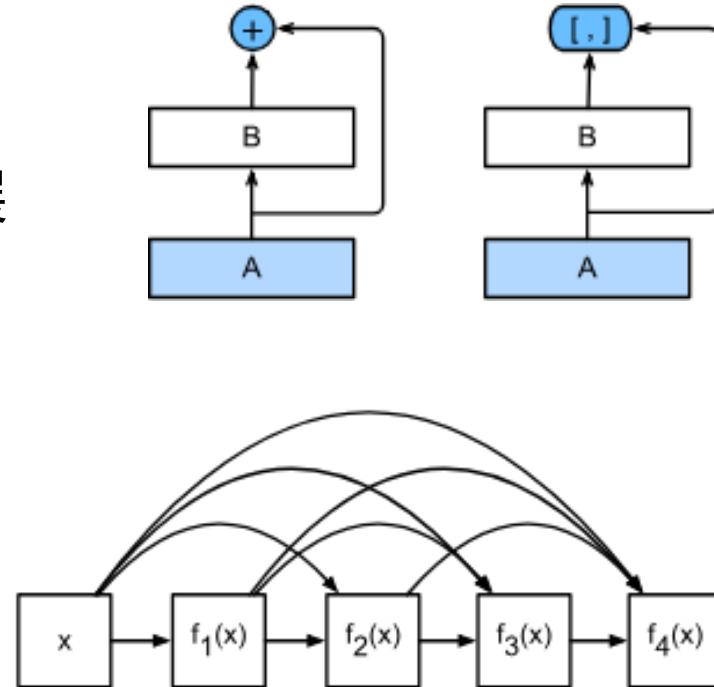
$$x_{i+1} = [x_i, f_i(x_i)]$$

$$x_1 = x$$

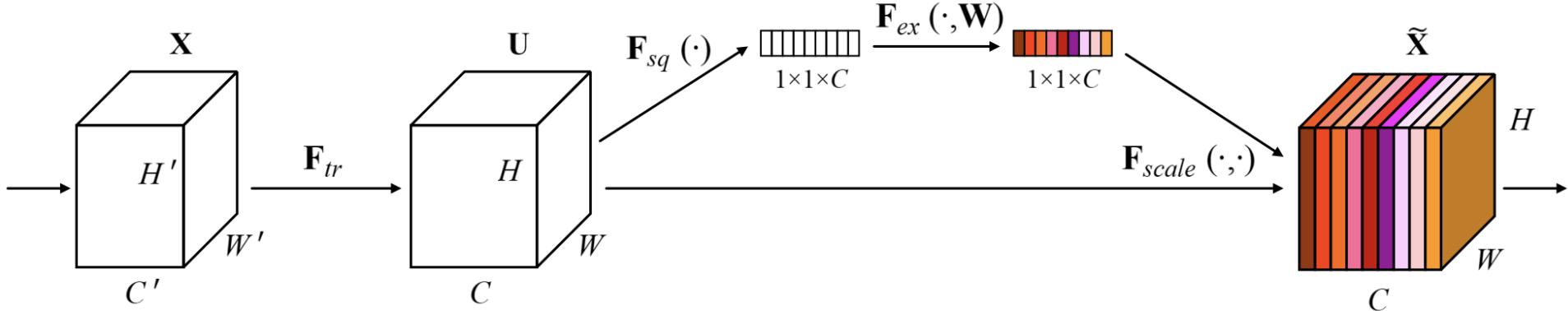
$$x_2 = [x, f_1(x)]$$

$$x_2 = [x, f_1(x), f_2([x, f_1(x)])]$$

- 偶尔需要降低分辨率 (过渡层)

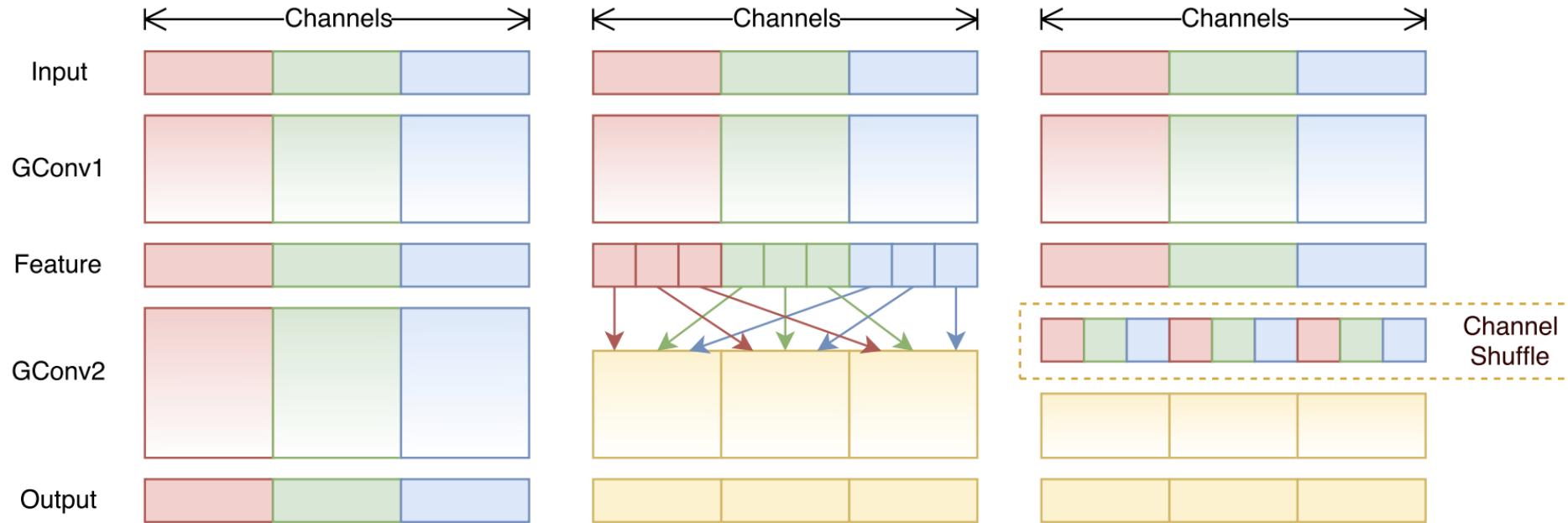


Squeeze-Excite Net



- Squeeze-Excite Net (Hu et al., 2017)
- 学习每个通道的全局加权函数
- 允许在图像的不同位置的像素之间快速传输信息

ShuffleNet (Zhang et al., 2018)



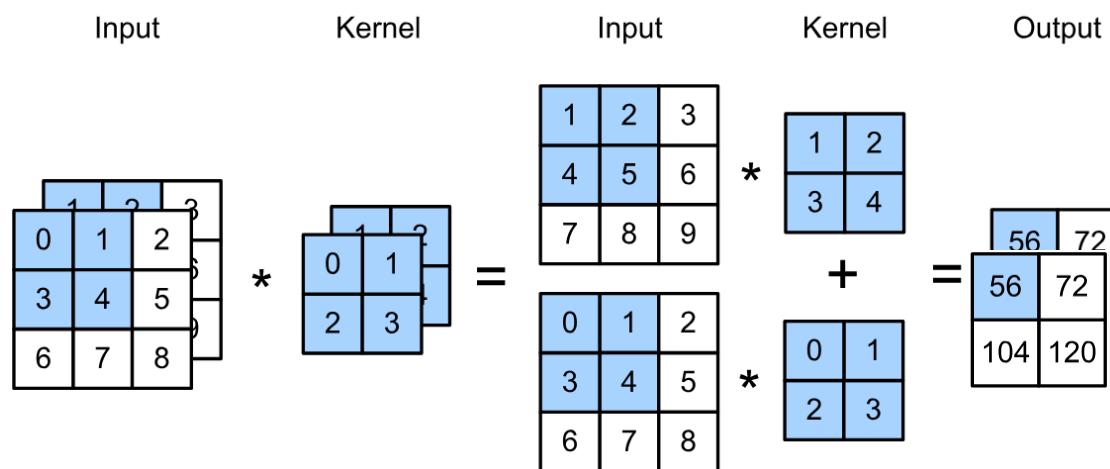
- ResNeXt 将卷积层分成不同通道
- ShuffleNet 通过分组混合不同通道 (对移动设备非常有效)

可分离的卷积层 - 所有通道分开

- 参数
- 计算
- 将通道分解
 - 没有混合多个通道

$$k_h \cdot k_w \cdot c_i \cdot c_o$$
$$m_h \cdot m_w \cdot k_h \cdot k_w \cdot c_i \cdot c_o$$

$$m_h \cdot m_w \cdot k_h \cdot k_w \cdot c$$



总结

- **Inception**
 - 卷积的不均匀混合（不同深度）
 - 批量归一正则化
- **ResNet**
 - 泰勒扩展式
 - 残差网络（**ResNext**） 分解卷积
- **Zoo**

稠密连接网络（DenseNet），ShuffleNet，可分解卷积层，…

动手学深度学习

14. 混合编程, 异步计算, 多GPU训练

中文教材: zh.d2l.ai

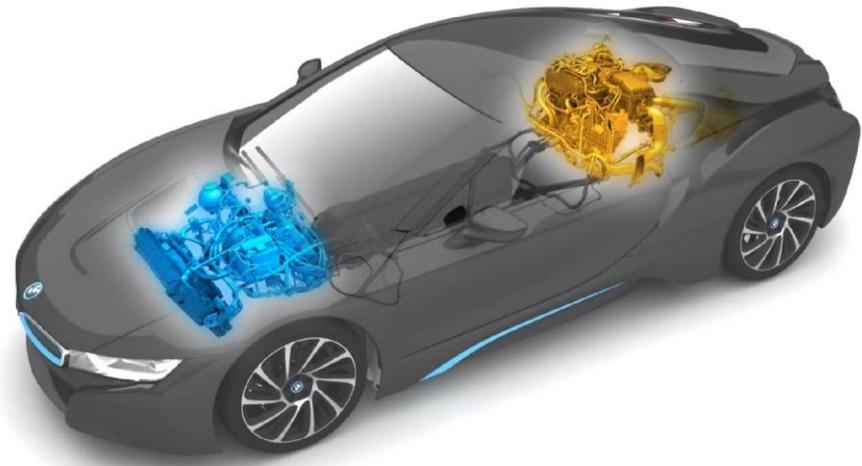
英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/parallel.html>

概要

- 混合编程
 - 命令式编程
 - 符号式编程
- 异步计算
- 自动并行
- 多GPU训练
- 分布训练

命令式与符号式 编程的混合



命令式编程

- 在 Python, Java, C / C ++ 中编程的常用方法……
 - 优点：
 - 简单易用，易于调试
 - 缺点：
 - 总是需要 Python 编译，难以部署模型：
 - 例如：智能手机
 - 性能问题

解释器编译成字节码，然后在虚拟机上执行

a = 1

b = 2

c = a + b

3 次执行

符号式编程

- 首先定义程序，输入数据以便稍后执行
- 数学，SQL,
- 优点：
 - 易于优化
 - 前端开销少
 - 通用型高
- 缺点：
 - 很难用

可以在没有Python
编译器的情况下使用

了解整个程序，
易于优化

```
expr = "c = a + b"
```

```
exec = compile(expr)
```

```
exec(a=1, b=2)
```

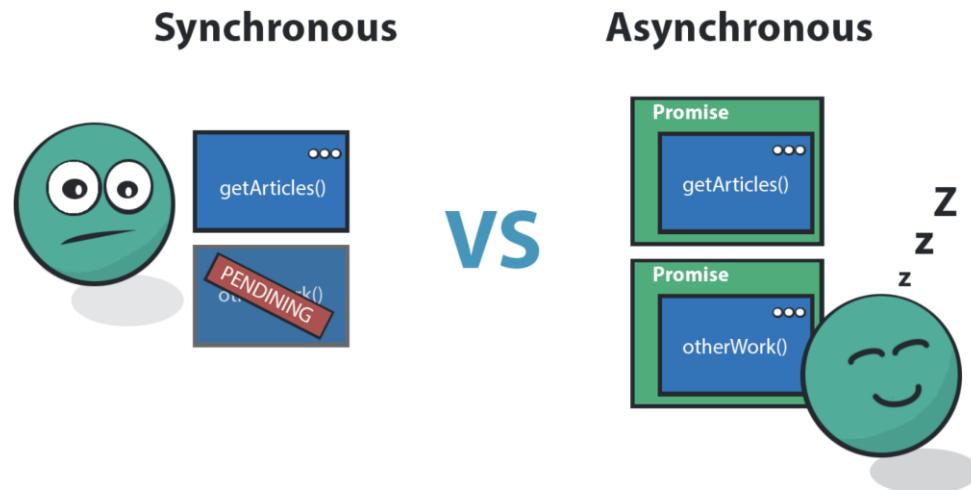
单次执行

Gluon 中的混合编程

- 通过 nn.HybridSequential 或 nn.HybridBlock 定义模型
- 调用 .hybridize() 从命令式执行切换到符号式执行

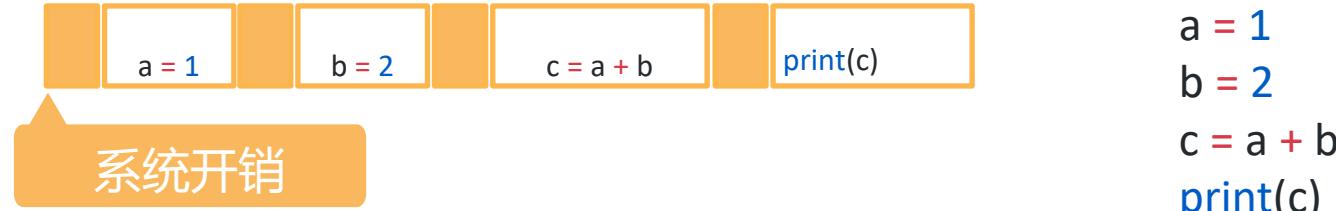
```
net = nn.HybridSequential()  
  
net.add(nn.Dense(256, activation='relu'),  
  
        nn.Dense(10))  
  
net.hybridize()
```

异步计算



异步计算

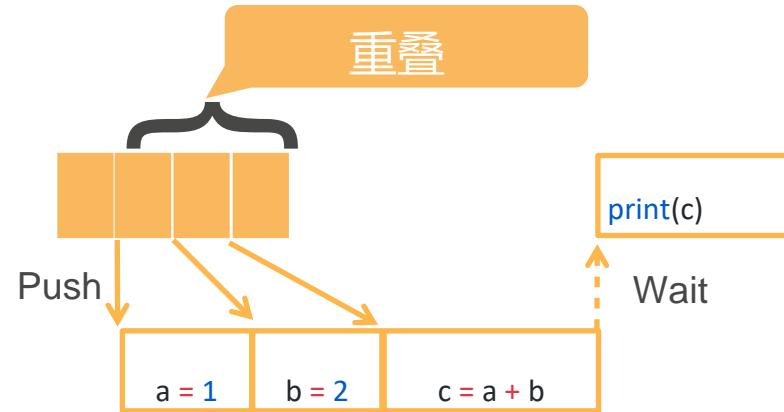
- 一个接一个执行



- 使用后端线程

前端线程

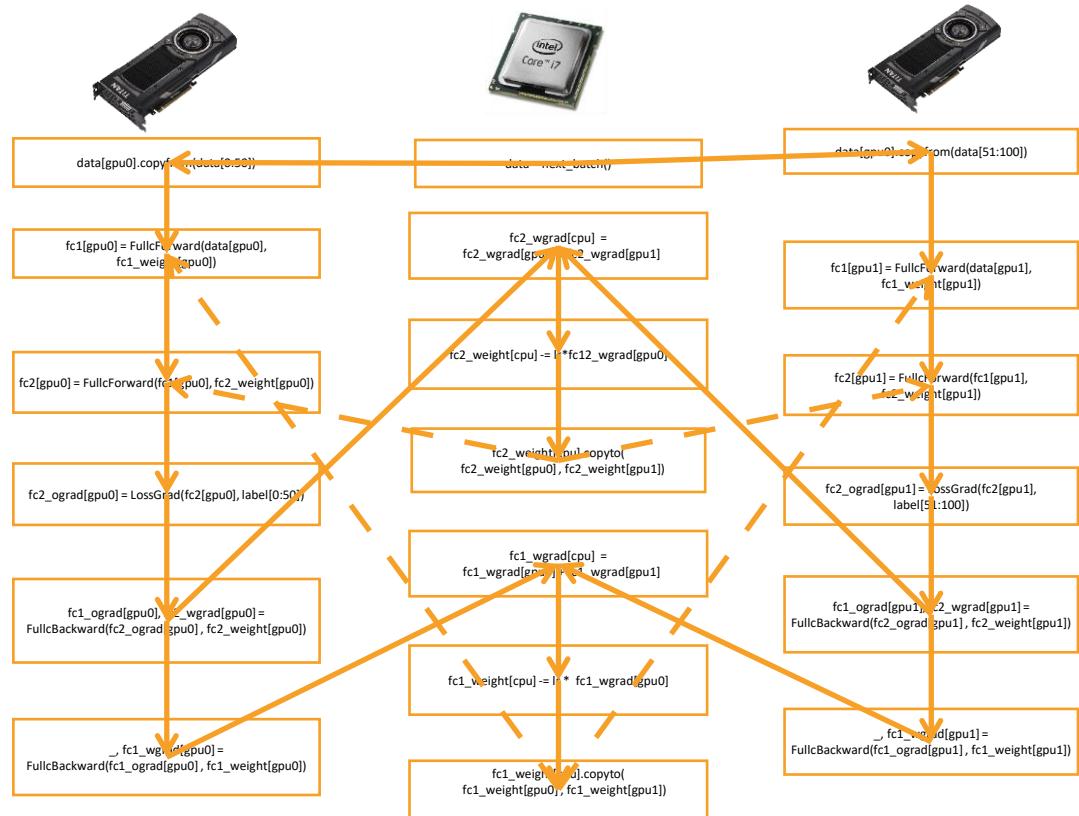
后端线程



自动并行



编写并行程序很痛苦...



- 具有 2 个 GPU 的单个隐藏层 MLP
- 可扩展到数百层和数十个 GPU

自动并行

编写串行程序

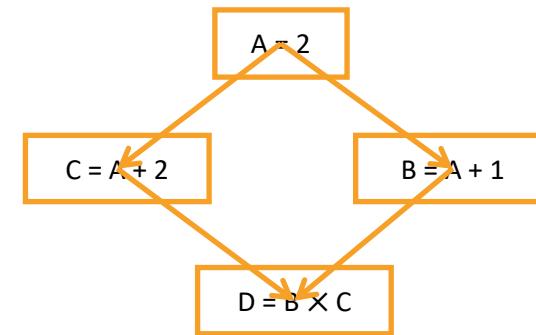
```
A = nd.ones((2,2)) * 2
```

```
C = A + 2
```

```
B = A + 1
```

```
D = B * C
```

平行运行



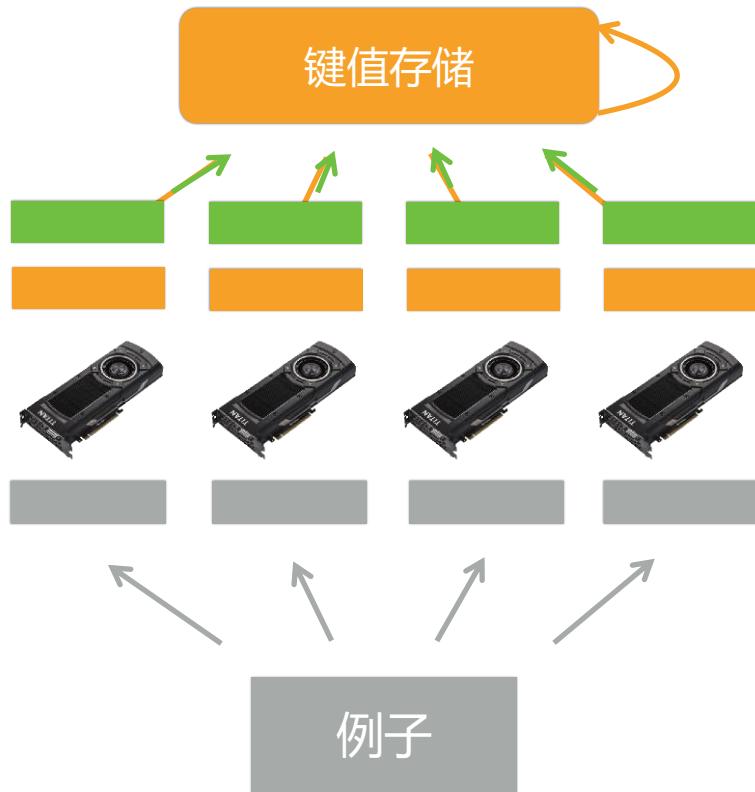
多GPU训练



(Lunar new year, 2014)

D2L.ai

数据并行



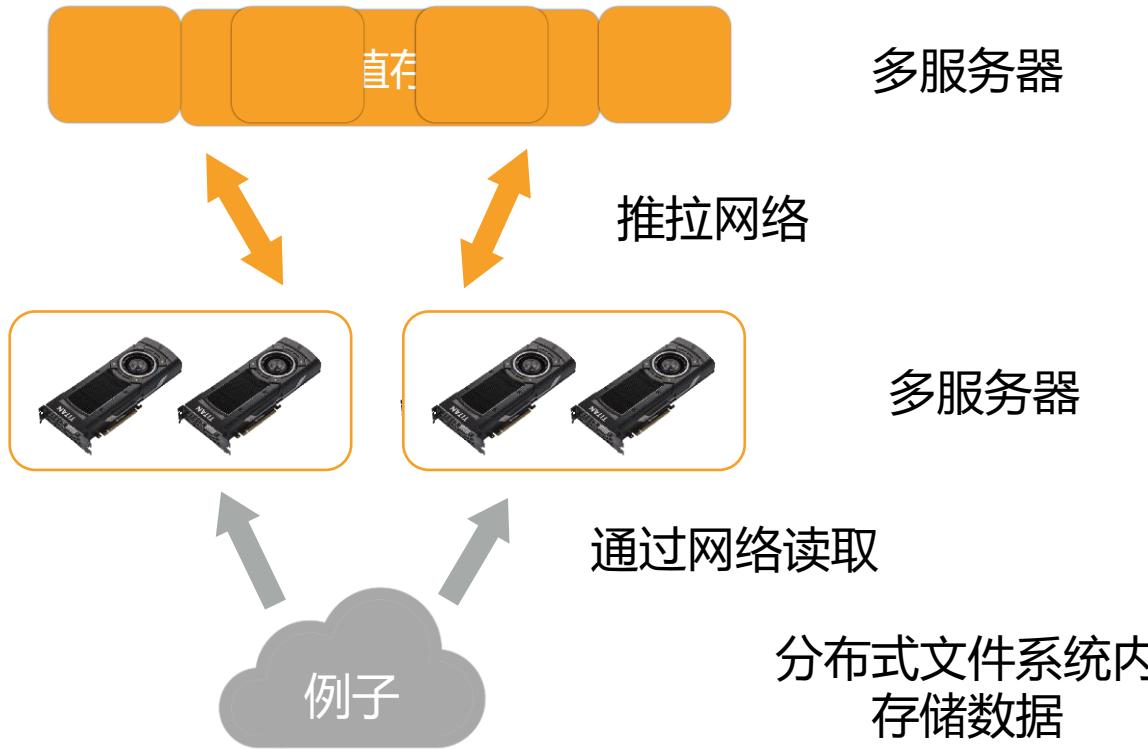
1. 读取数据分区
2. 收集参数
3. 计算梯度
4. 发送渐变
5. 更新参数

分布训练

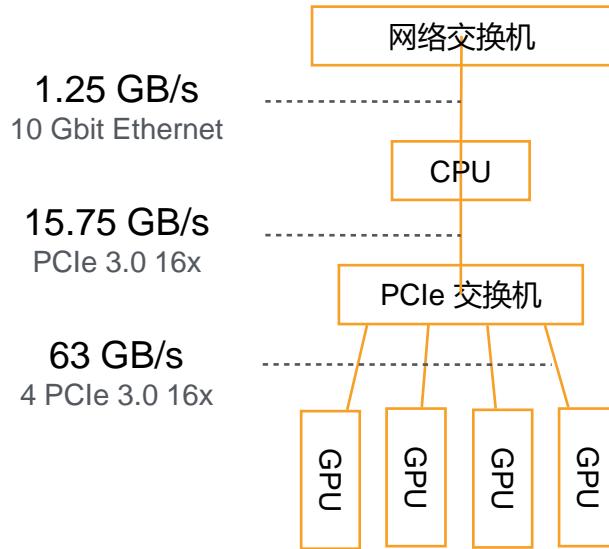


(Alex's frugal GPU cluster at CMU, 2015)

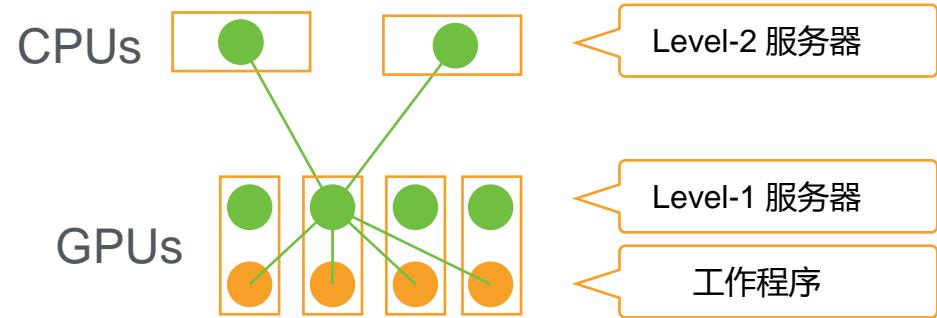
分布计算



GPU机器层次结构

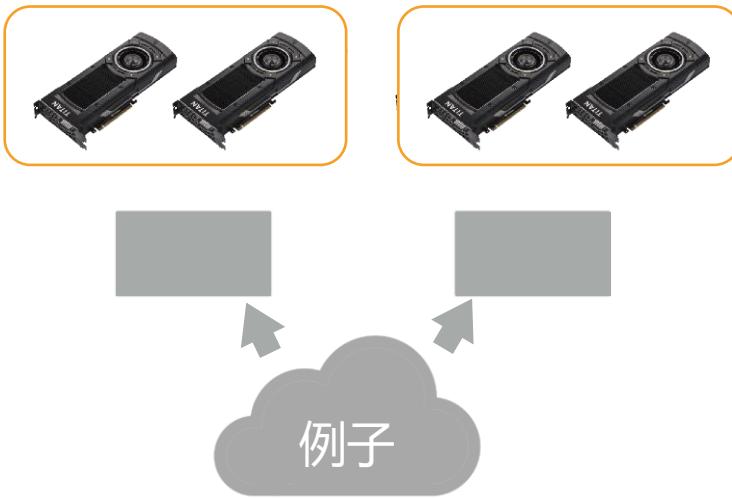


分层参数服务器



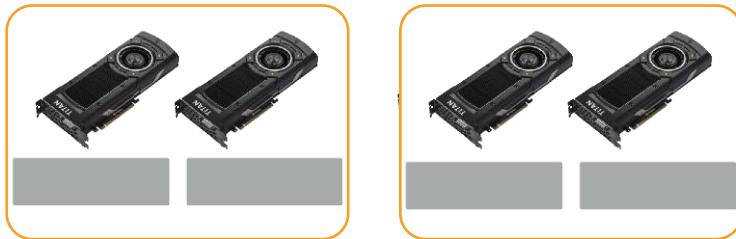
一次迭代

- 每个工作机器读取数
据批次的一部分

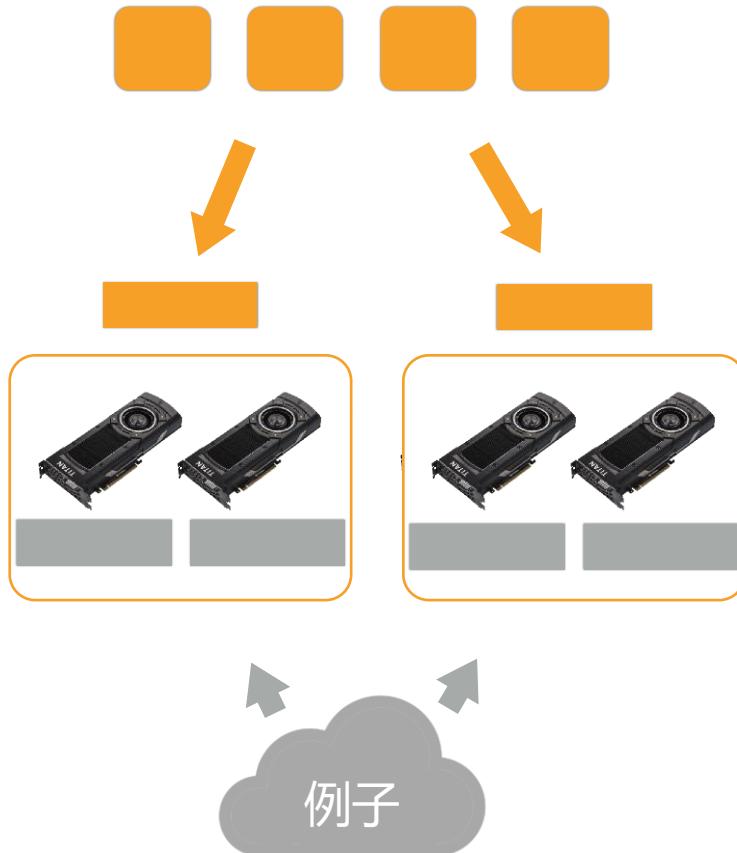


一次迭代

- 进一步拆分并推送到每个GPU



一次迭代

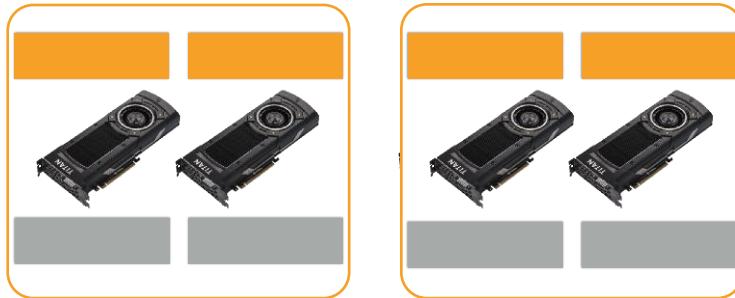


- 每个服务器都维护一部分参数
- 每个工作程序从服务器中提取其全部参数

一次迭代



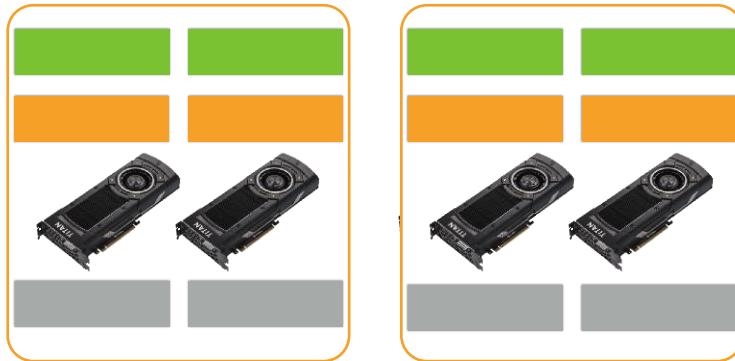
- 将参数复制到每个 GPU 中



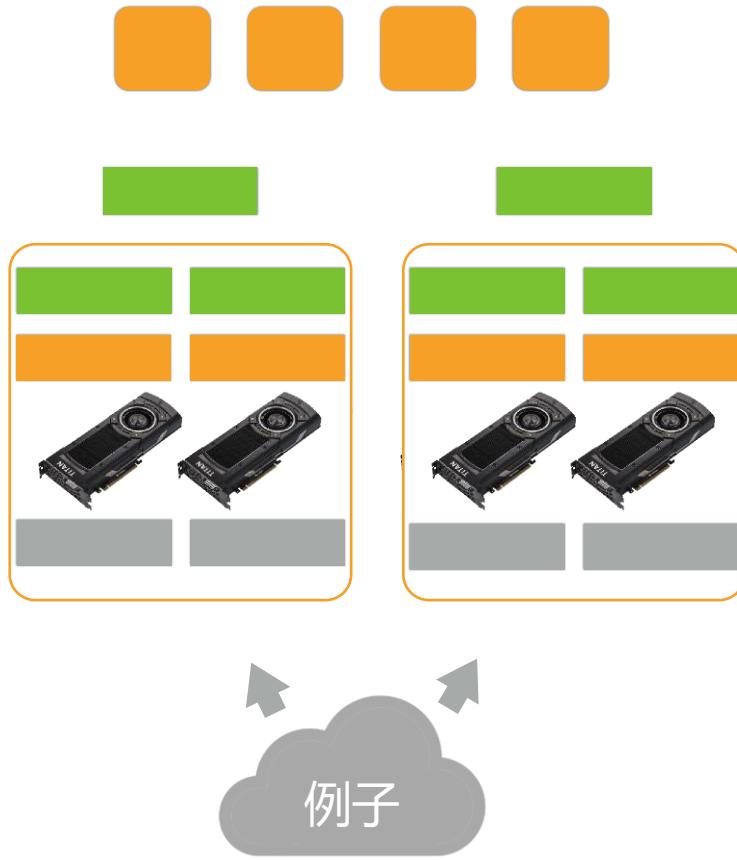
一次迭代



- 每个GPU都计算梯度

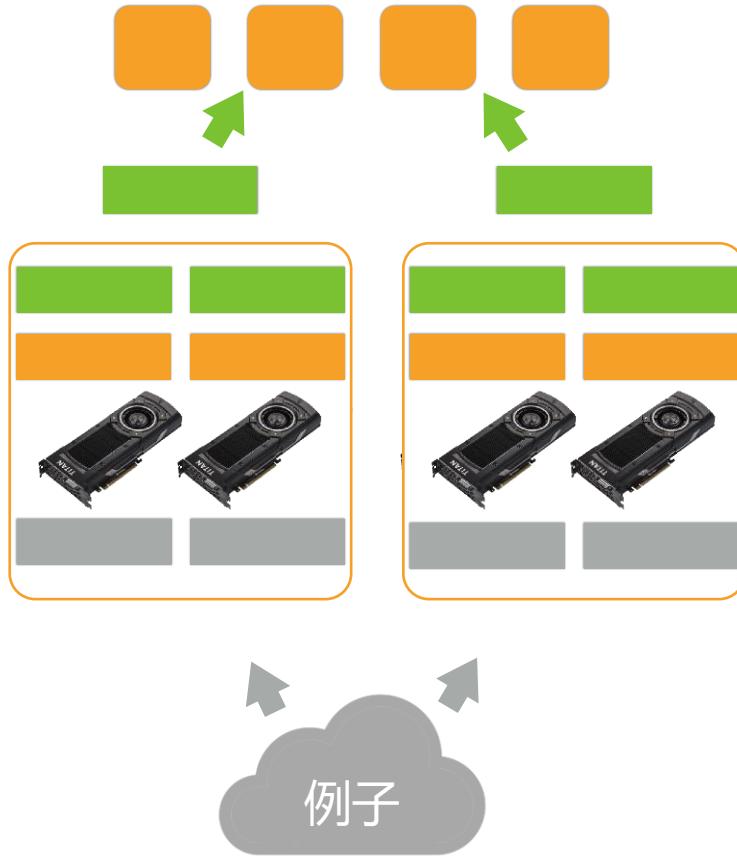


一次迭代



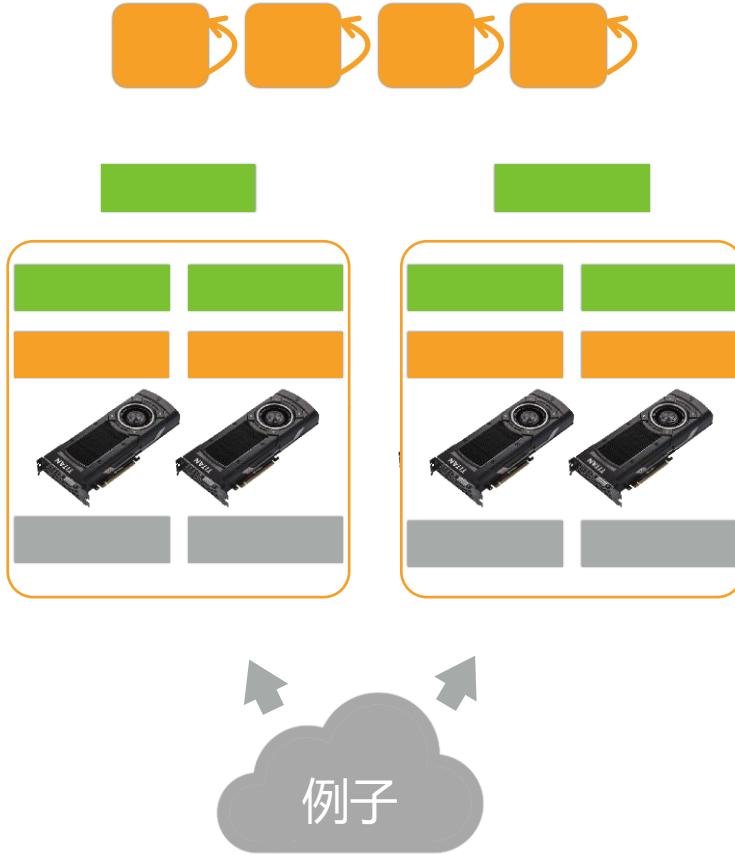
- 对所有GPU的梯度求和

一次迭代



- 将梯度递送给服务器

一次迭代



- 每个服务器统计来自所有工作程序的梯度，然后更新其参数

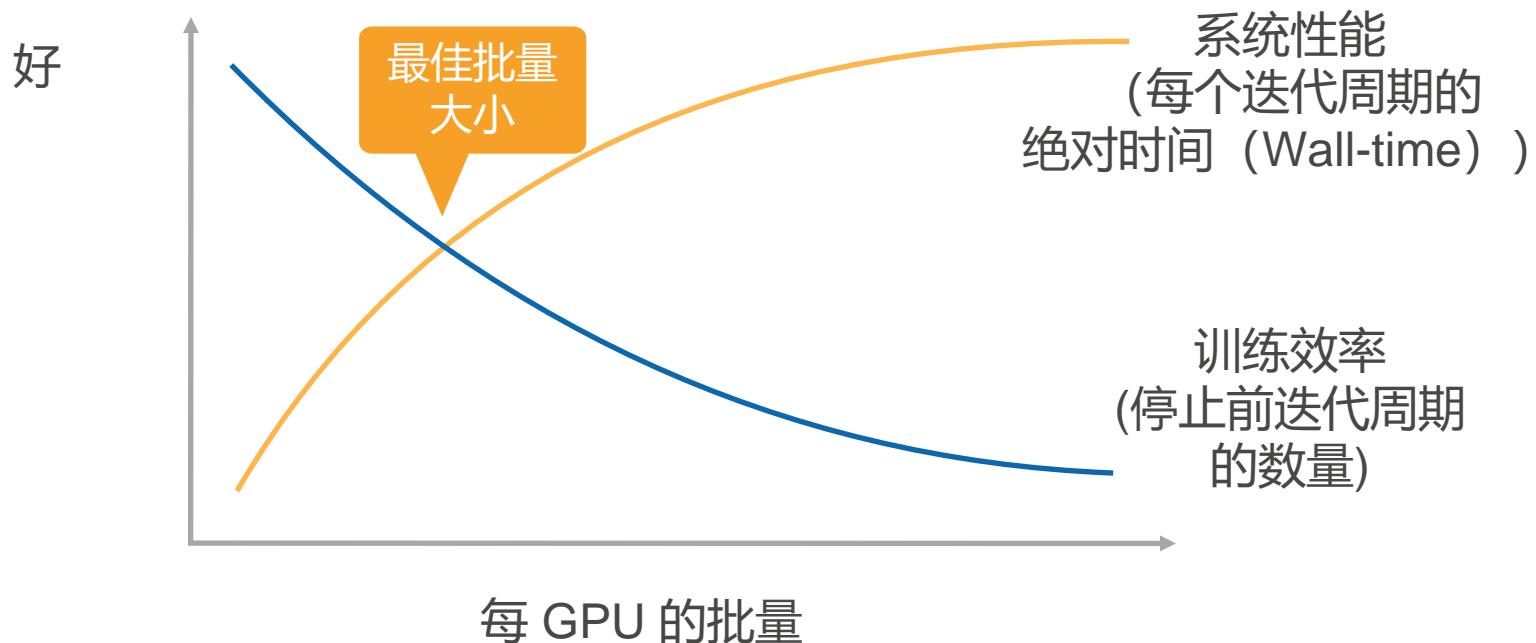
同步 SGD

- 每个工作程序同步运行，称为同步 SGD
- 假设每次 n 个 GPU 和每个 GPU 处理 b 个样本：
 - 同步 SGD 等于单个 GPU 上的小批量 SGD
 - 批量大小为 nb
- 在理想情况下，与单个GPU训练相比，使用 n 个GPU进行训练有 n 倍的加速

性能

- T_1 = 计算 GPU 中 b 个示例梯度的时间
- 假设有 m 个参数，工作程序每次发送和接收 m 个参数/梯度
 - T_2 = 发送和接收的时间
- 每批次的绝对时间 (Wall-time) = $\max(T_1, T_2)$
 - 需要选择足够大的 b
- 增加 b 和/或 n 导致更大的批量大小，这可能需要更多的迭代周期以达到期望的模型质量

性能权衡



实际建议

- 用大型数据集
- 良好的 GPU-GPU 和“机器-机器”带宽
- 高效的数据加载和预处理
- 具有良好浮点计算 (FLOP) 与通信 (模型大小) 比率的模型
 - Inception > ResNet > AlexNet
- 批量大，足以获得良好的系统性能
 - 大批量效率优化的技巧

总结

- 混合编程
 - 命令式编程
 - 符号式编程
- 异步计算
- 自动并行
- 多GPU训练
- 分布训练

动手学深度学习

15. 图像增广，微调 和 样式迁移

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/finetuning.html>

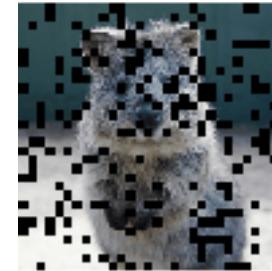
概要

- 图像增广
- 微调
 - 低层特征可通用
 - 高层特征重新初始化
- 样式迁移
 - 内容损失
 - 样式损失
 - 总损失函数

图像增广



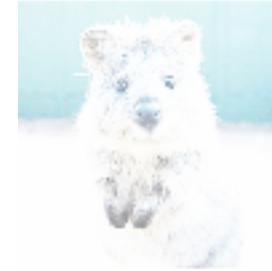
$p=1.0$



$\text{size_percent}=0.30$



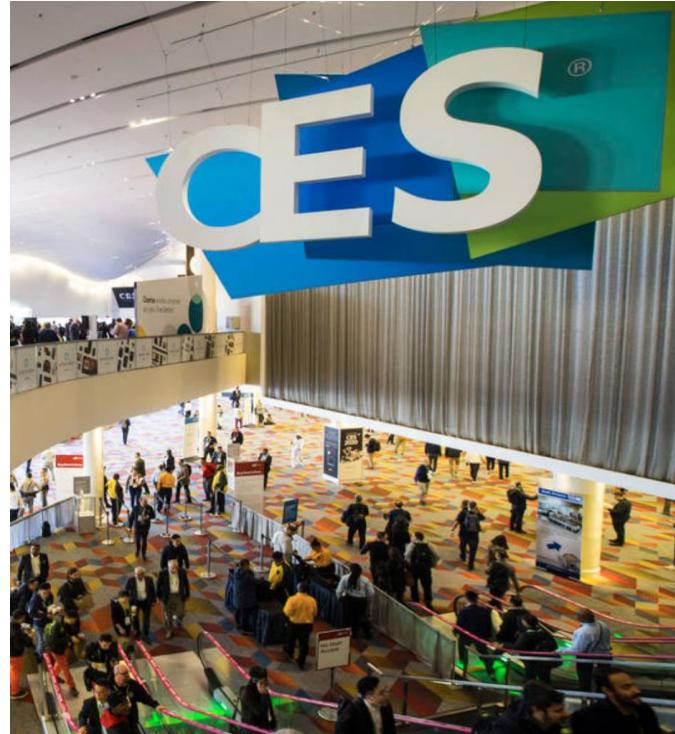
$p=0.50$



$\text{cutoff}=0.00$

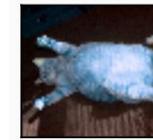
CES'19 真实故事

- 一家初创公司的演示：一款智能自动售货机，通过相机镜头识别客户挑选的物品。
- 但演示失败，因为展厅有不同的：
 - 光温
 - 桌子上的光反射
- 他们整夜工作：
 - 重新收集数据并训练新模型
 - 订购了桌布



数据增广

- 现有数据集基础上，增广为具有更多多样性的数据集：
 - 在训练数据集中添加各种背景噪音
 - 转换为其他图像：改变颜色，改变形状



图像增广



反转

- 左右反转



- 上下反转

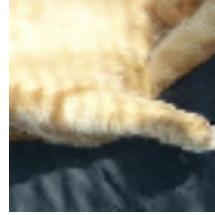


- 不一定都有效



裁剪

- 从图像中裁剪一块区域，然后调整其大小
 - 随机宽高比（例如 [3/4, 4/3]）
 - 随机区域大小（例如 [8%, 100%]）
 - 一个随机的位置



变色

- 调整色调，饱和度和亮度（例如 [0.5, 1.5]）



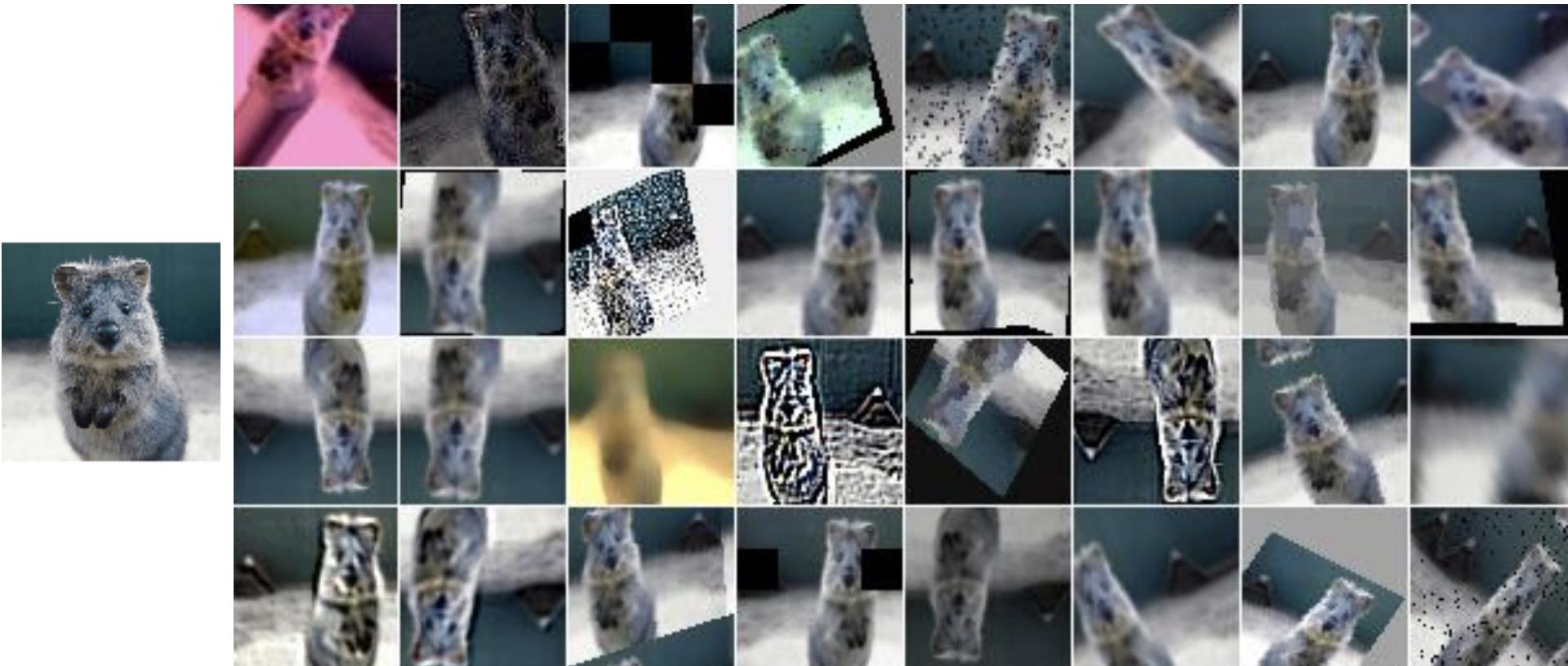
亮度



色调



更多方法



微调



标记数据集非常昂贵

现有数据集



2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6

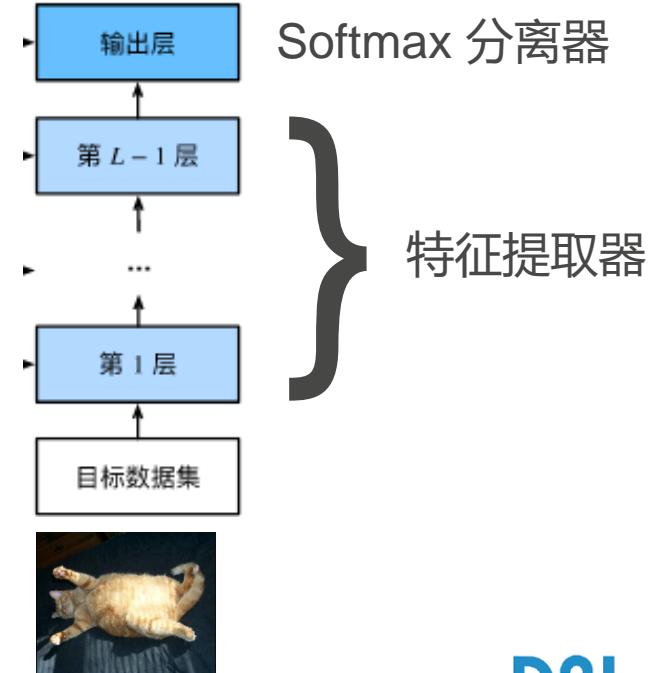
# 样本	1.2 M	50K	60 K
# 类别数	1,000	100	10

网络架构

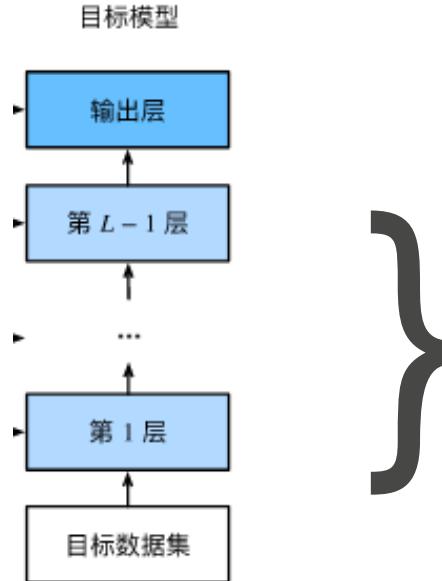
- 神经网络可以大致分为两部分
 - 特征提取器将原始像素映射为线性可分离的特征
 - 用线性分类做决定



目标模型



微调



不直接使用原分类器的最后一层的参数，因为
标签已更改



这些层可能对我的数据
集是很好的特征提取器

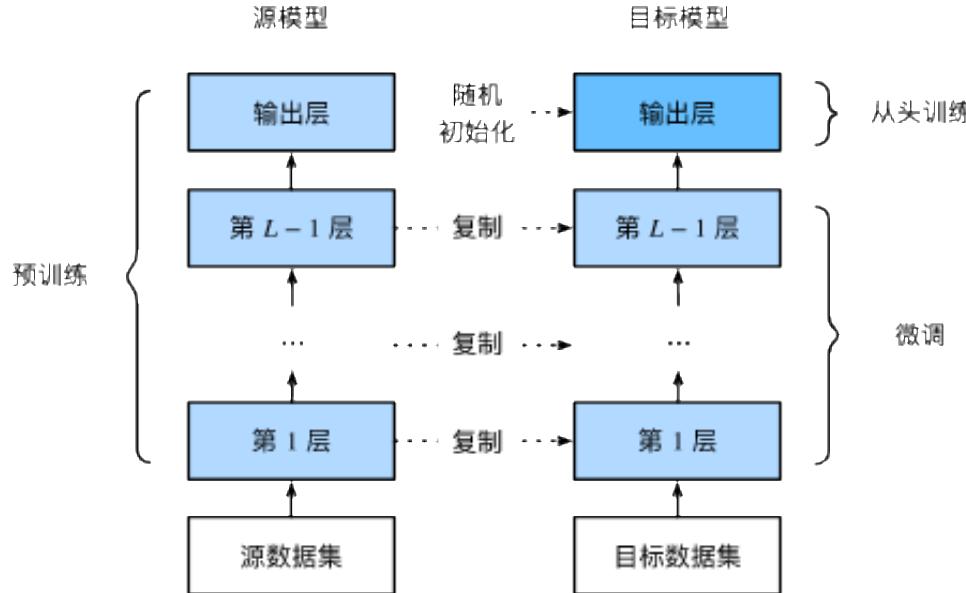
源数据集



目标数据集



微调中的权重初始化



源数据集



目标数据集

微调训练

- 在已有神经网络上微调目标数据集，但具有强大的正则化
 - 使用较小的学习率
 - 使用较少的迭代周期
- 如果源数据集比目标数据集更复杂，则微调通常会得到更高质量的模型

重用已有分类器的参数

- 源数据集可能包含目标数据集中的某些类别
- 在初始化期间使用来自预训练模型的相应权重向量



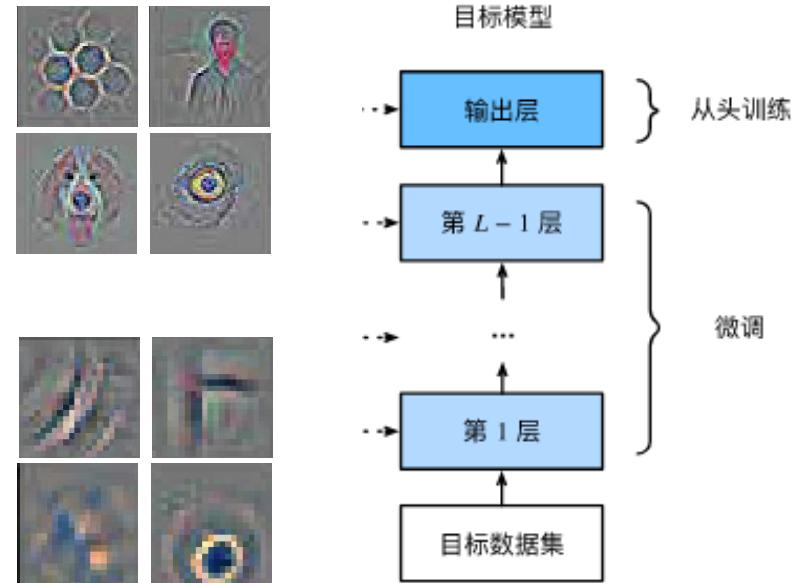
Racer, race car, racing car

A fast car that competes in races



修复一些层

- 神经网络学习分层特征表示
 - 低层特征是通用的
 - 高层特征与数据集中的对象更相关
- 在微调期间修复底层的参数
 - 另一个强有力的原则



样式迁移



+

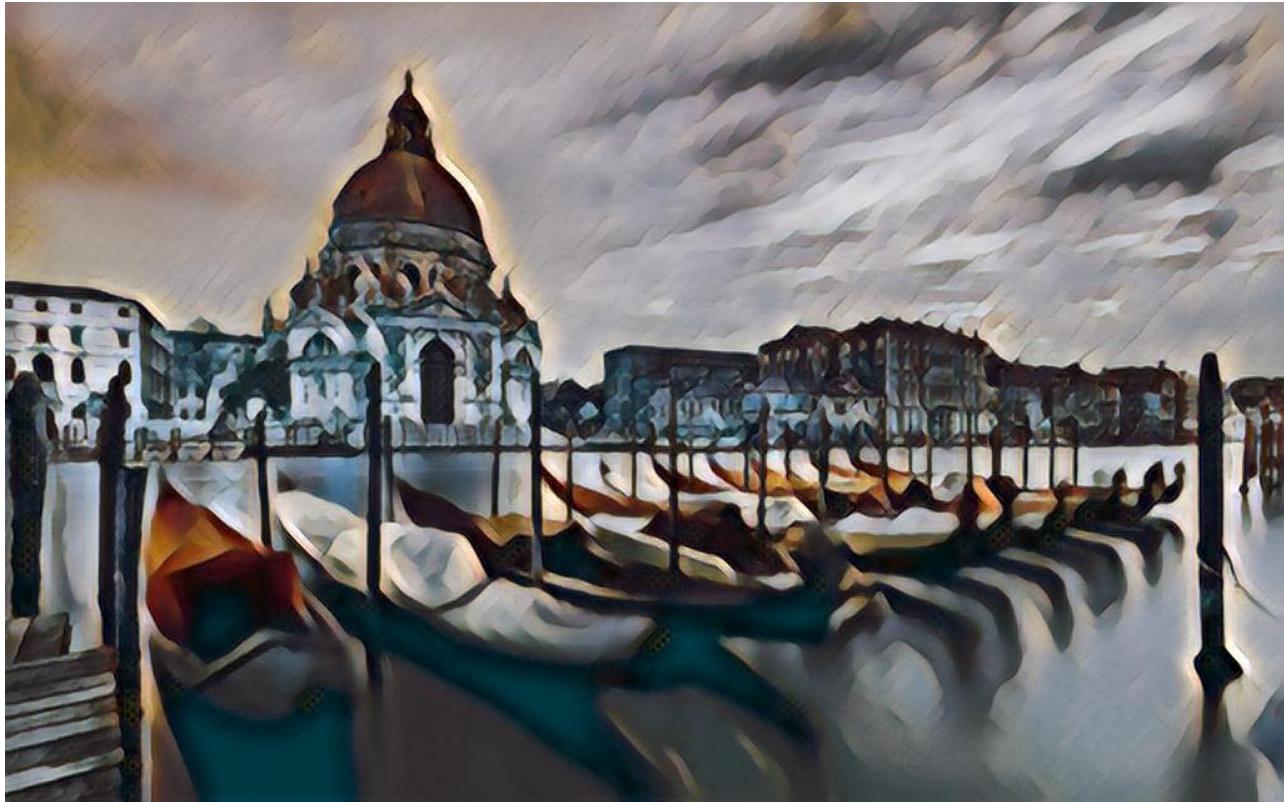


<https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/>

D2L.ai



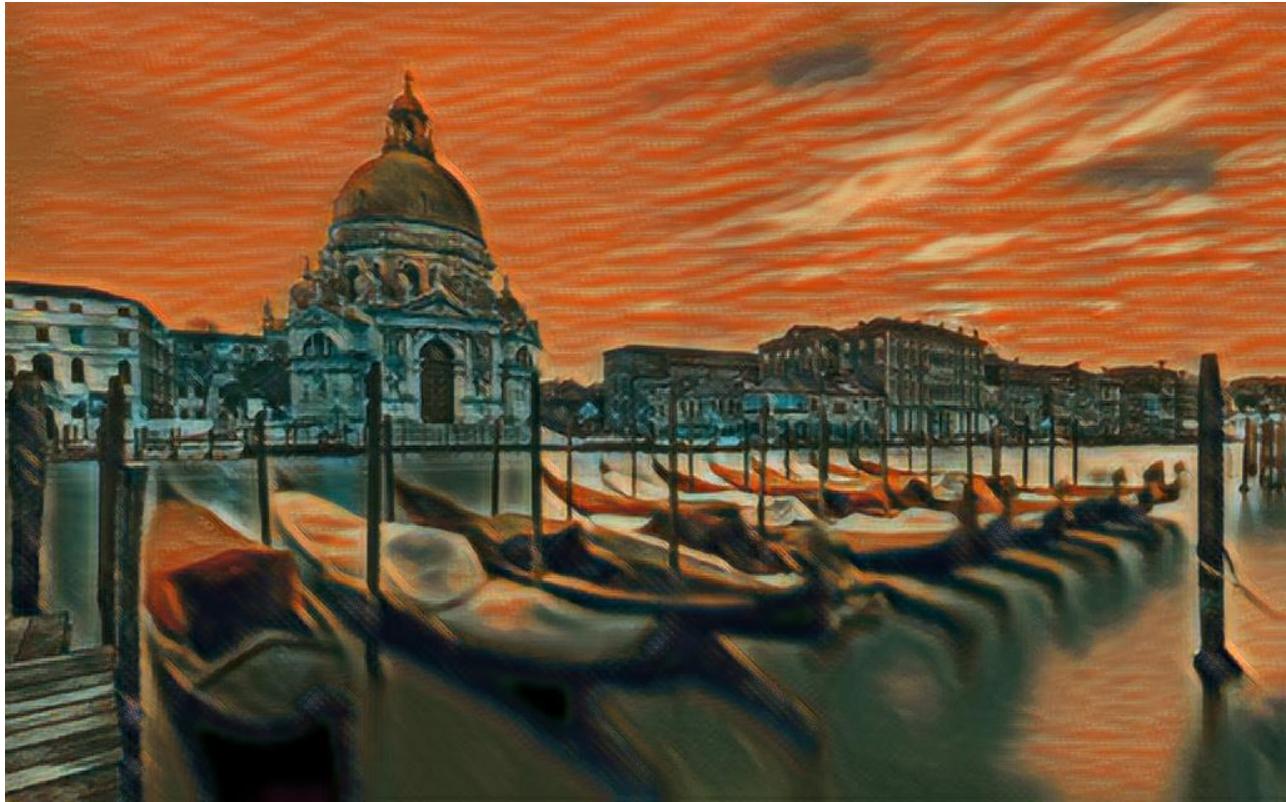
+



D2L.ai



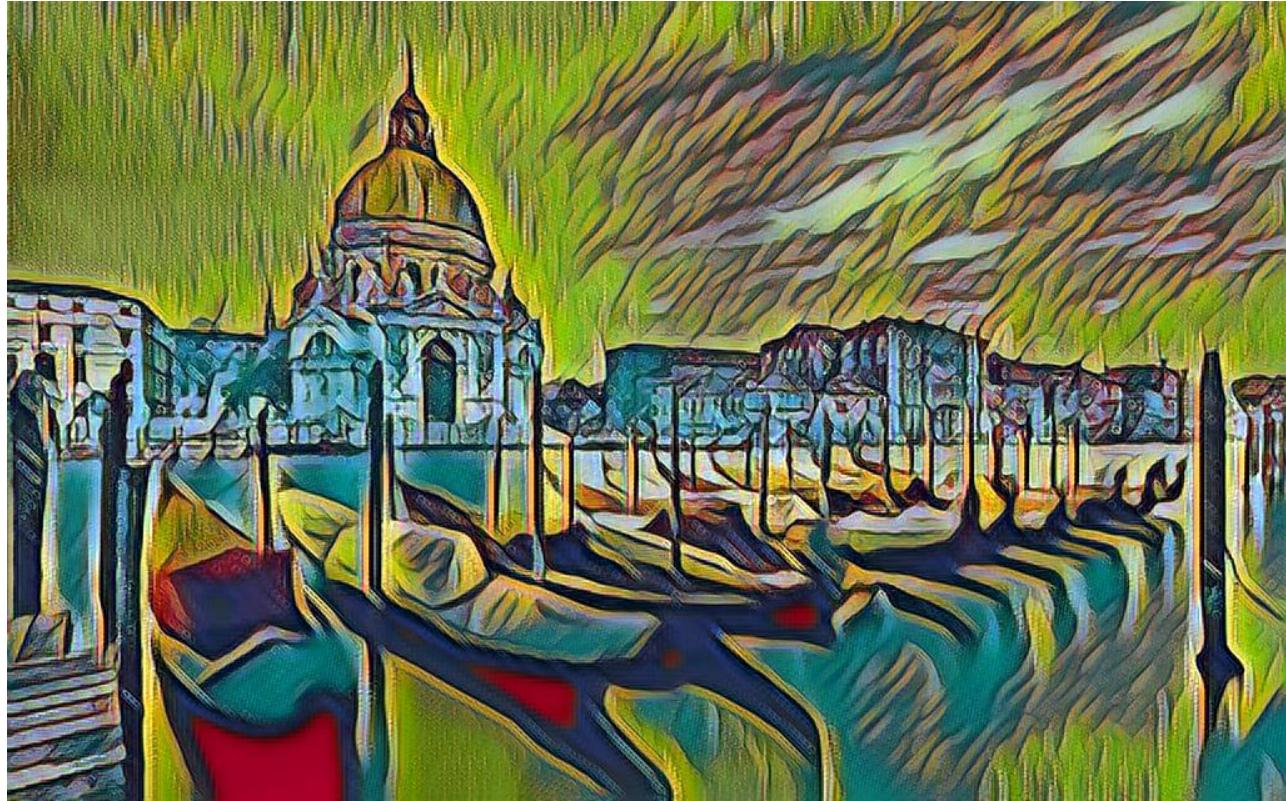
+



D2L.ai



+



D2L.ai



+





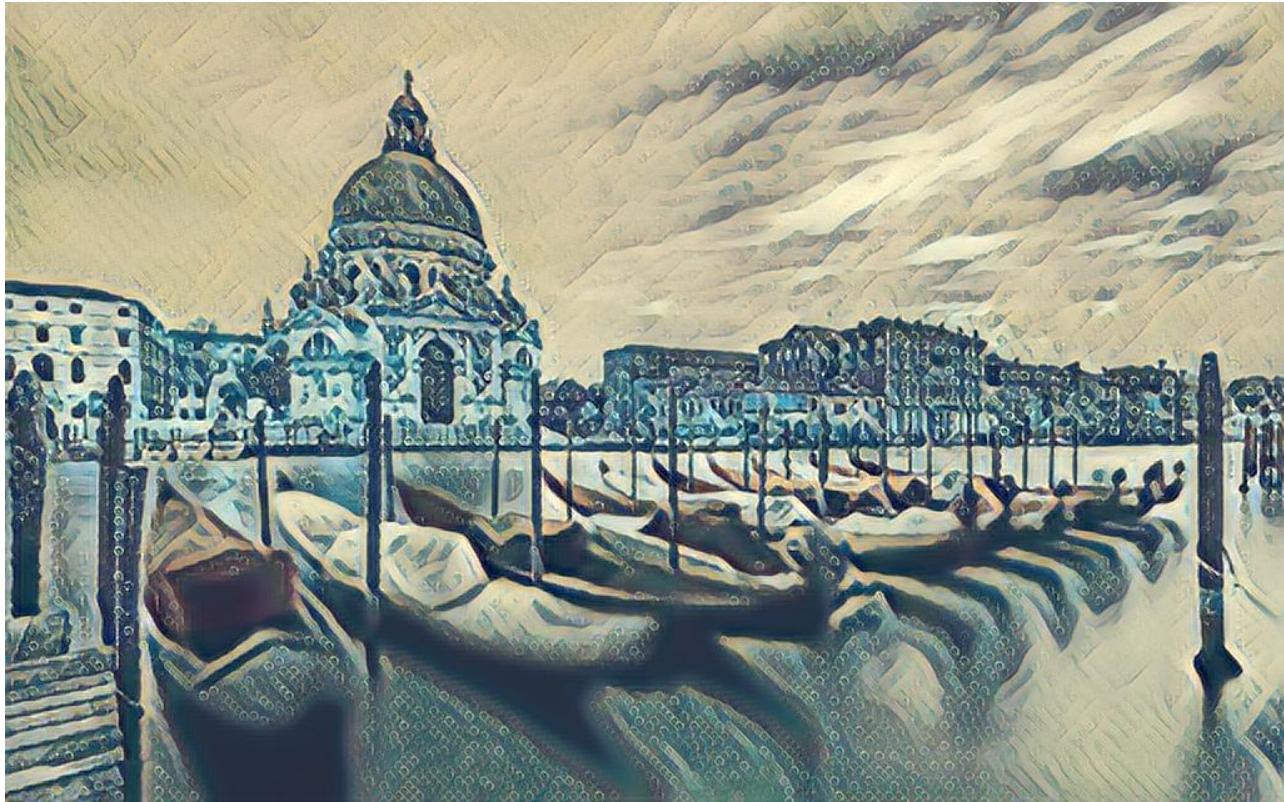
+



D2L.ai



+



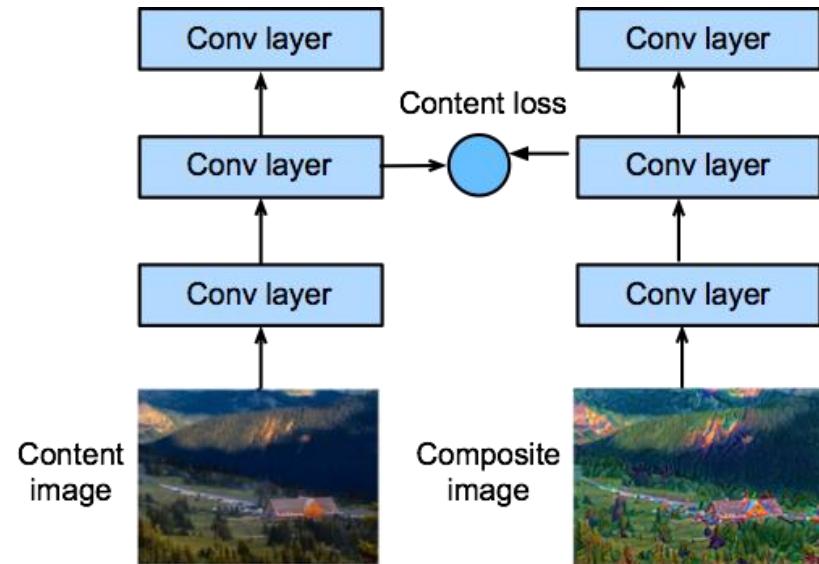
神经元风格

- 学习合成图像：
 - 内容图像中的内容
 - 样式图像中的样式

$$\arg \min_I w_1 \ell_{\text{content}}(I_{\text{content}}, I) + w_2 \ell_{\text{style}}(I_{\text{style}}, I) + w_3 \ell_{\text{noise}}(I)$$

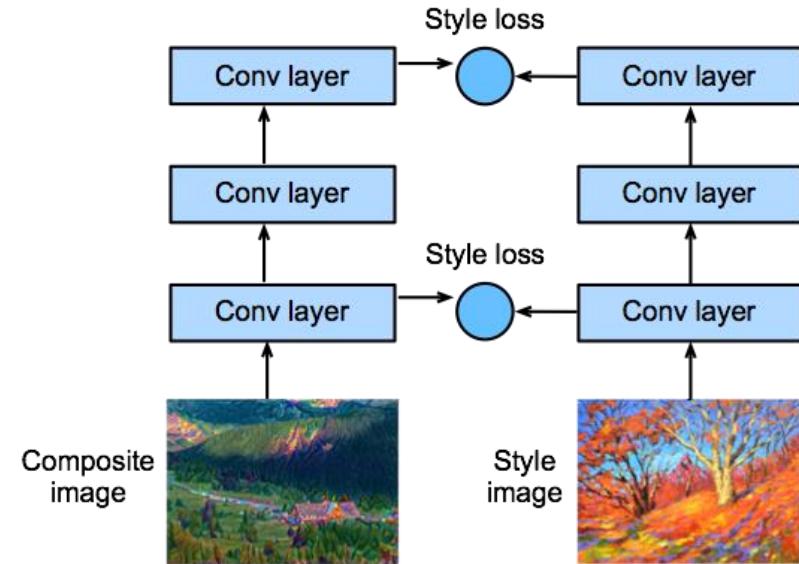
内容损失

- 将内容图像和合成图像都放入 CNN
- 比较双方的平方损失
 - 底层匹配细节
 - 顶层匹配全局内容



样式损失

- 格拉姆矩阵 (Gram matrix) G :
 - $G_{i,j}$ 是通道 i 和 j 之间的内积
 - 通过平方损失比较内部层输出的 G
 - 底层匹配本地样式
 - 顶层匹配全局样式



噪声损失

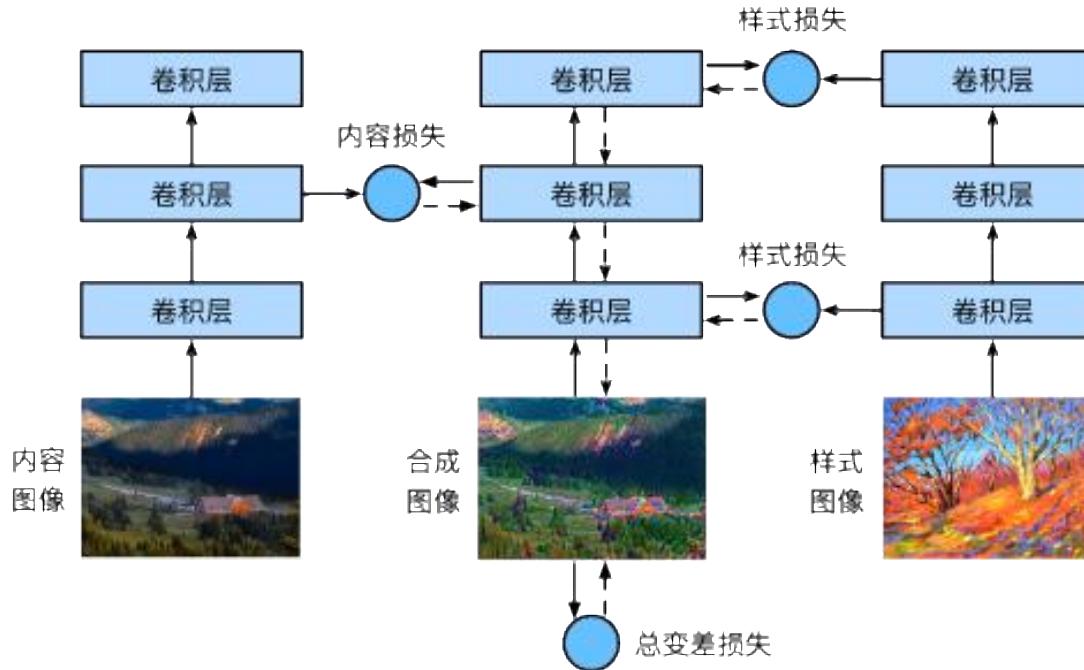
- 学习的合成图像可能具有许多高频噪声
- 使用总变差损失来降噪

$$\sum_{i,j} |x_{i,j} - x_{i+1,j}| + |x_{i,j} - x_{i,j+1}|$$

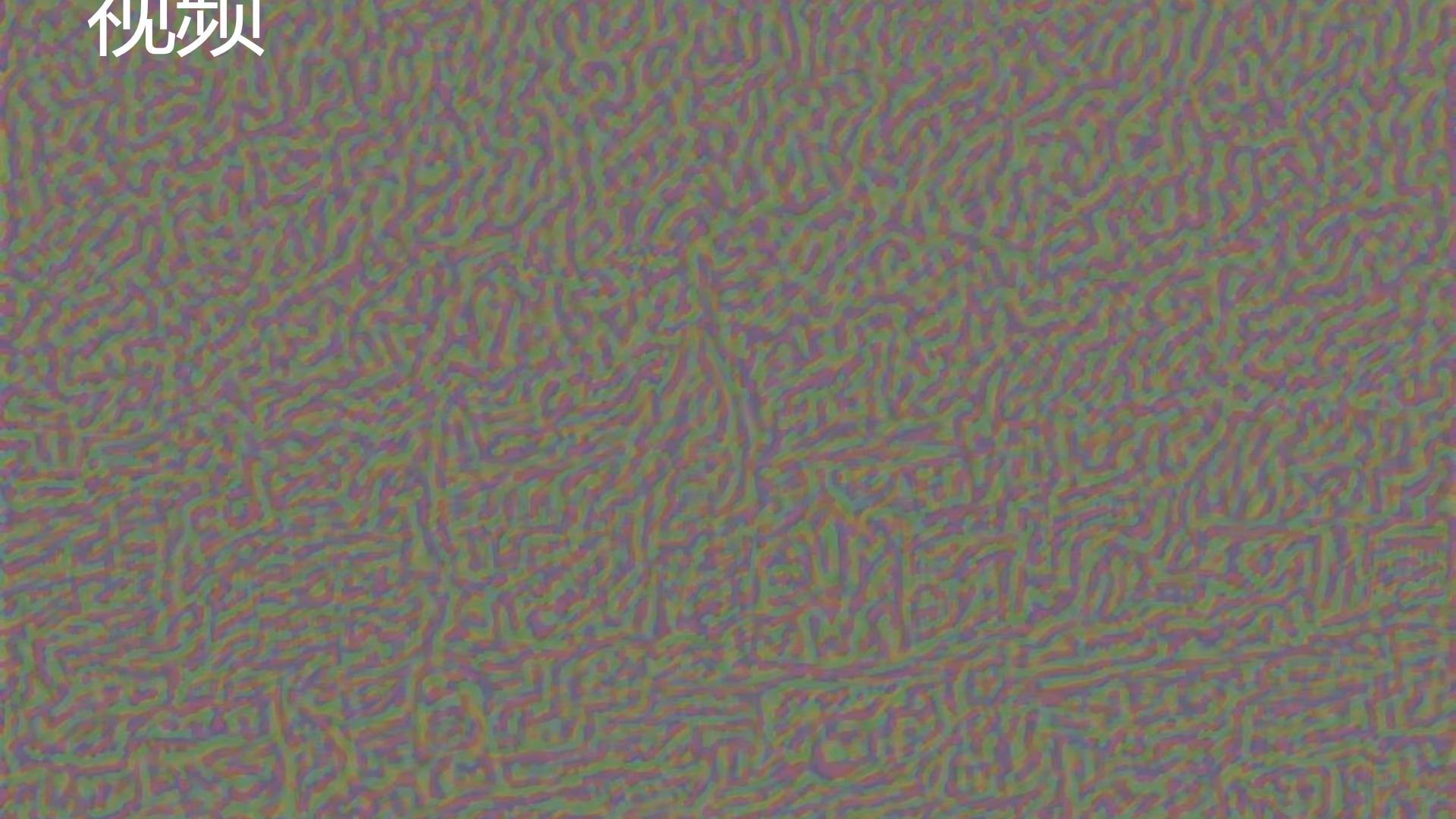


总损失函数

$$\operatorname{argmin}_I w_1 \ell_{\text{content}}(I_{\text{content}}, I) + w_2 \ell_{\text{style}}(I_{\text{style}}, I) + w_3 \ell_{\text{noise}}(I)$$



视频



总结

- 图像增广
- 微调
 - 低层特征可通用
 - 高层特征重新初始化
- 样式迁移
 - 内容损失
 - 样式损失
 - 总损失函数

动手学深度学习

16. 目标检测，计算机视觉训练技巧

中文教材：zh.d2l.ai

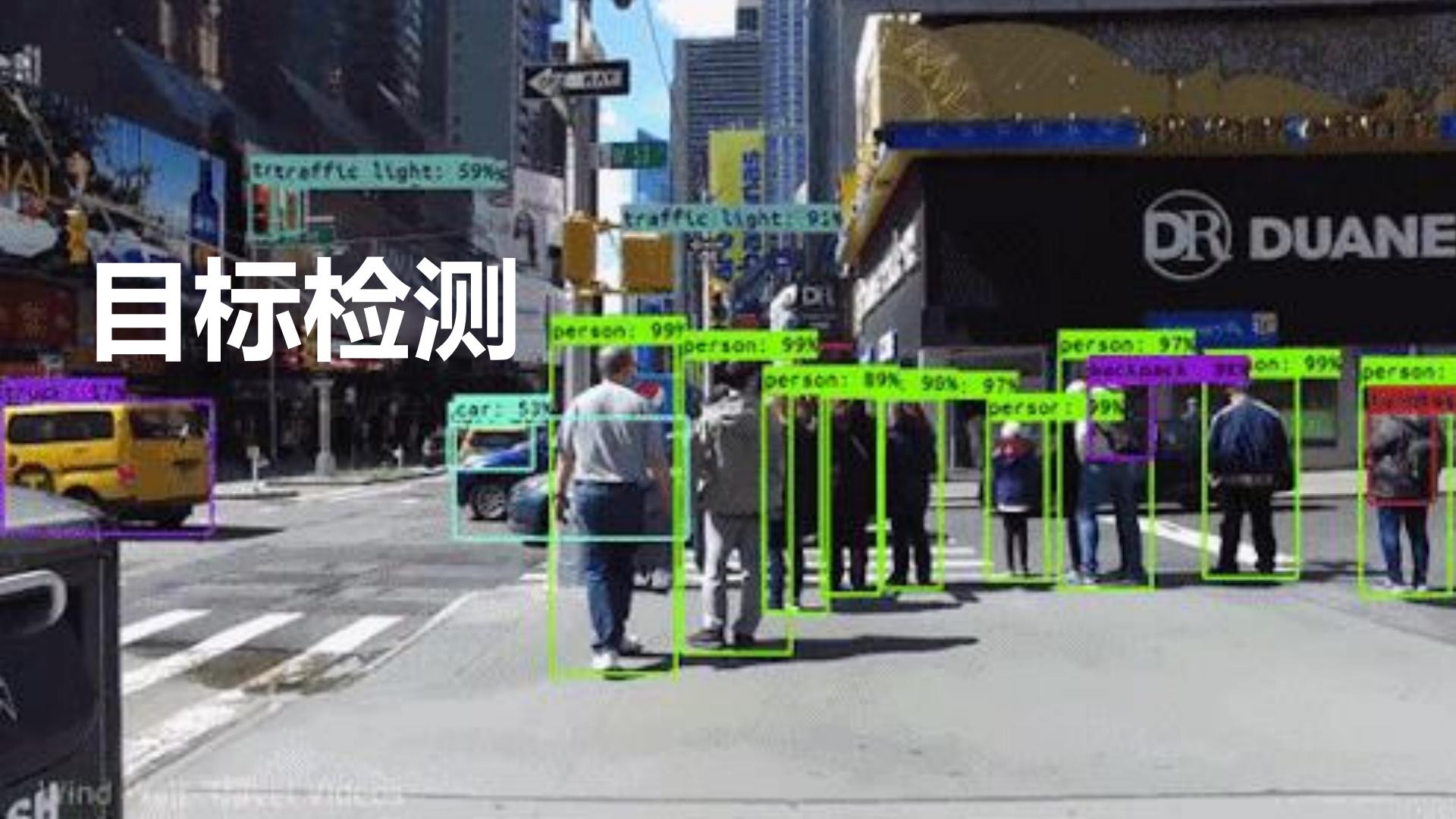
英文教材：www.d2l.ai

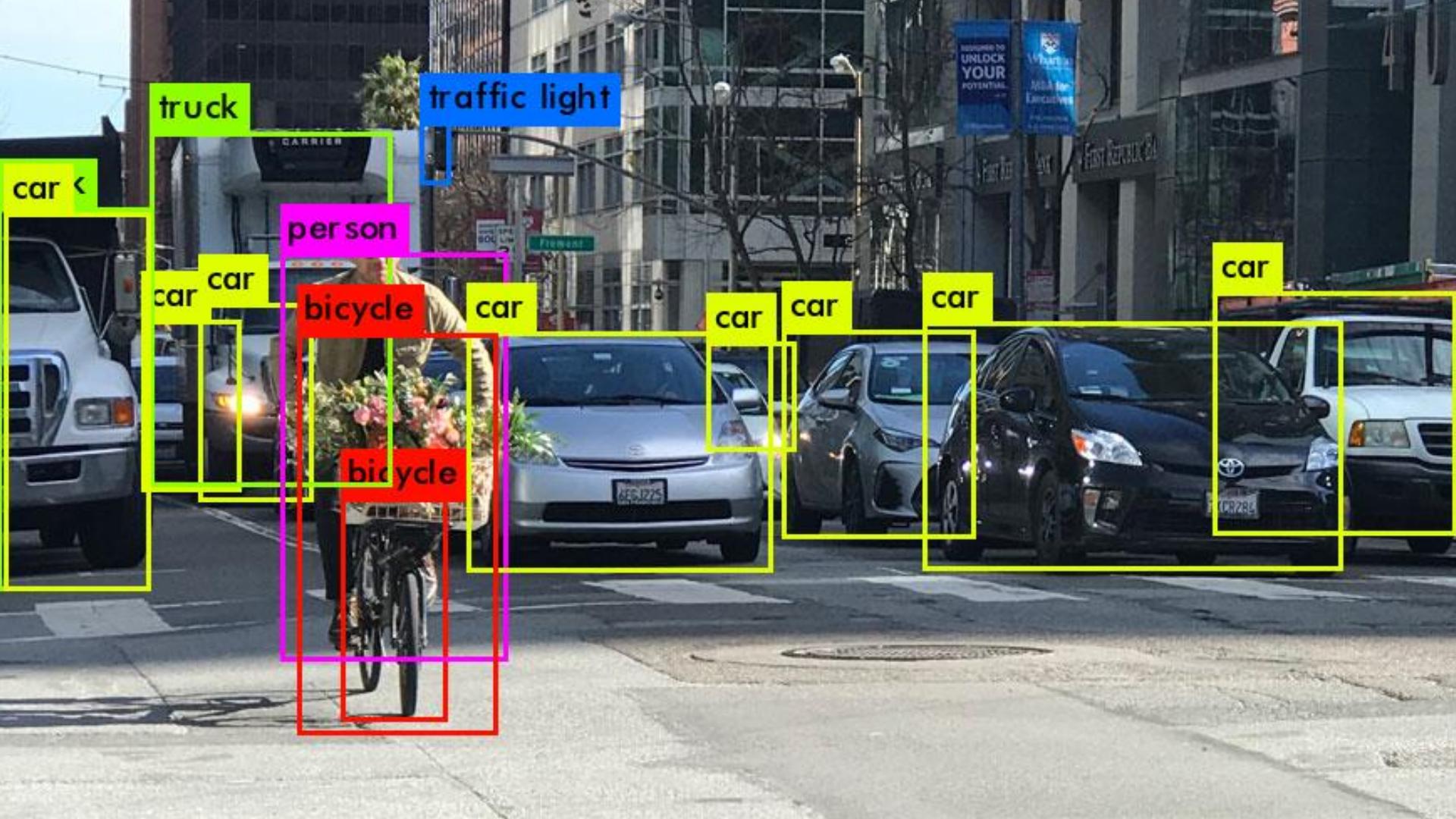
教学视频：<https://courses.d2l.ai/berkeley-stat-157/units/detection.html>

概要

- 目标检测
 - 边界框和锚框
 - 交并比
 - 区域卷积神经网络 (R-CNN)
 - 单发多框检测 (SSD)
- 计算机视觉训练技巧

目标检测



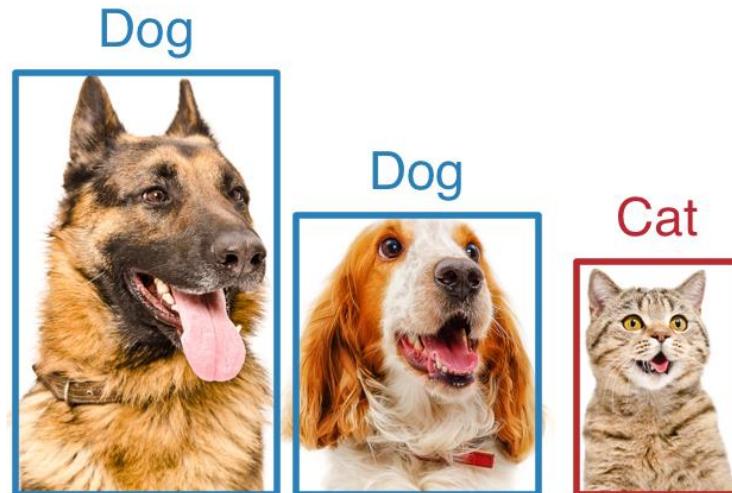


图像分类



Dog

目标检测

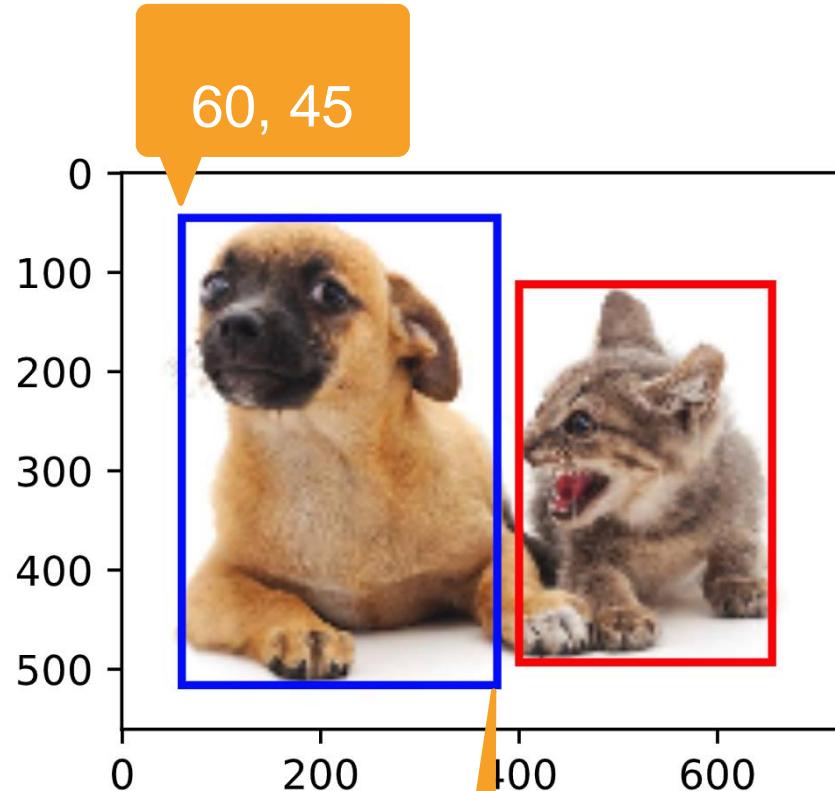


边界框和锚框



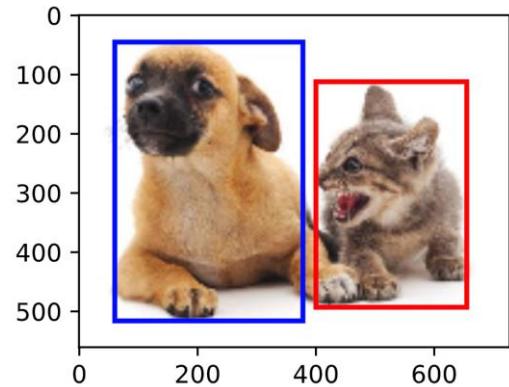
边界框

- 一个边界框可以由 4 个数字定义：
 - (左上角 x, 左上角 y, 右下角 x, 右下角 y)
 - (左上角 x, 右上角 y, 宽度, 高度)



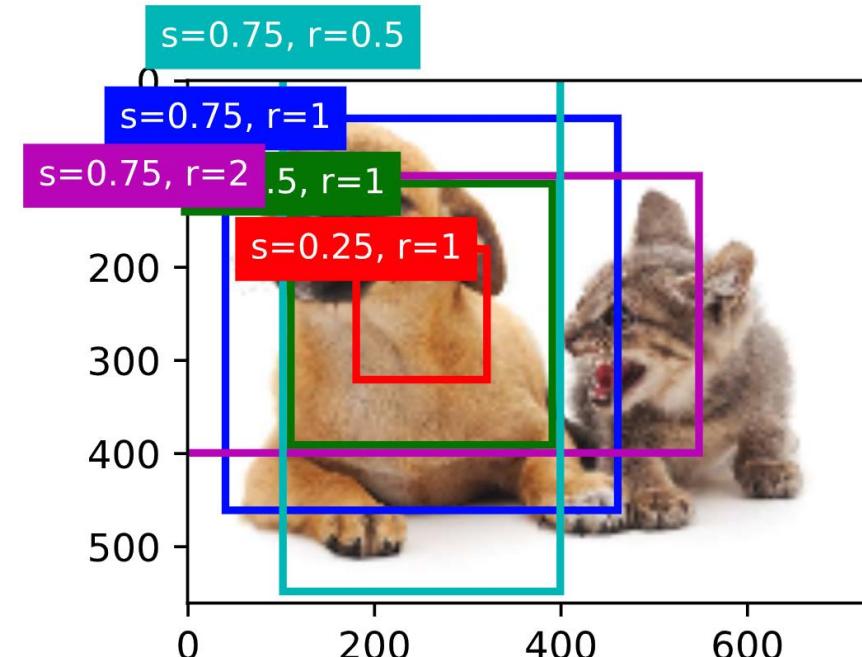
目标检测数据集

- 每行都有一个目标物体
 - 图像名称，物体类别，边界框
- COCO 数据集 (cocodataset.org)
 - 80 个物体
 - 330K 个图像
 - 1.5M 目标物体



锚框

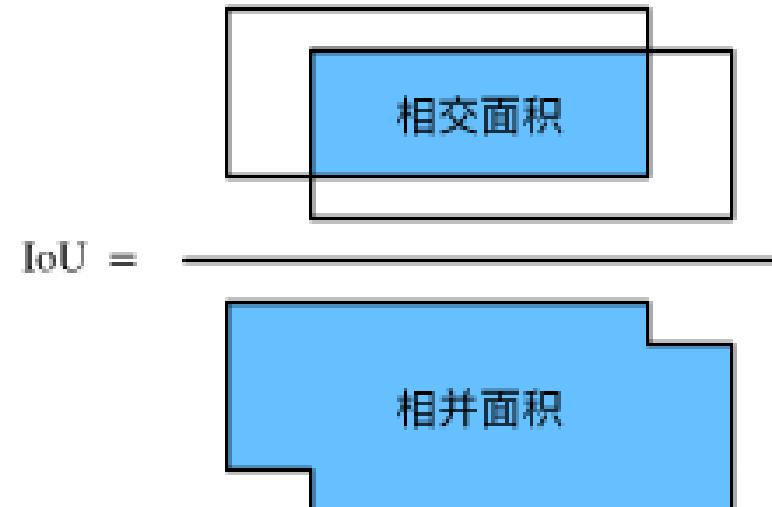
- 目标物体检测算法：
 - 选择多个区域，称为锚框
 - 预测每个锚框是否包含目标物体
 - 如果是，则预测从锚框到实际边界框的偏移量



交并比

- 交并比 (IoU) 测量两个框之间的相似性：
 - 0 表示不重叠
 - 1 表示完全相同
- Jaccard 系数，给定集A和B：

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$



将标签分配给锚框

- 每个锚框都是一个训练样例
- 标记每个锚框：
 - 类别
 - 真实边界框相对锚框的偏移量
- 我们可能会生成大量的锚框
 - 导致大部分负面例子

将标签分配给锚框

真实边界索引 A_i

	1	2	3	4
1				
2			x_{23}	
3				
4				
5				
6				
7	x_{71}			
8				
9				

假设矩阵 X 中最大值为 x_{23} ，
我们将分配真实边界框 B_3 给锚框 A_2 。

然后，丢弃矩阵中第2行和第3列的所有元素，
找出剩余阴影部分的最大元素 x_{71} ，
为锚框 A_7 分配真实边界框 B_1 。

将标签分配给锚框

真实边界索引 A_i

	1	2	3	4
1				
2			x_{23}	
3				
4				
5				
6				
7	x_{71}			
8				
9				

	1	2	3	4
1				
2			x_{23}	
3				
4				
5				
6				
7	x_{71}			
8				
9				

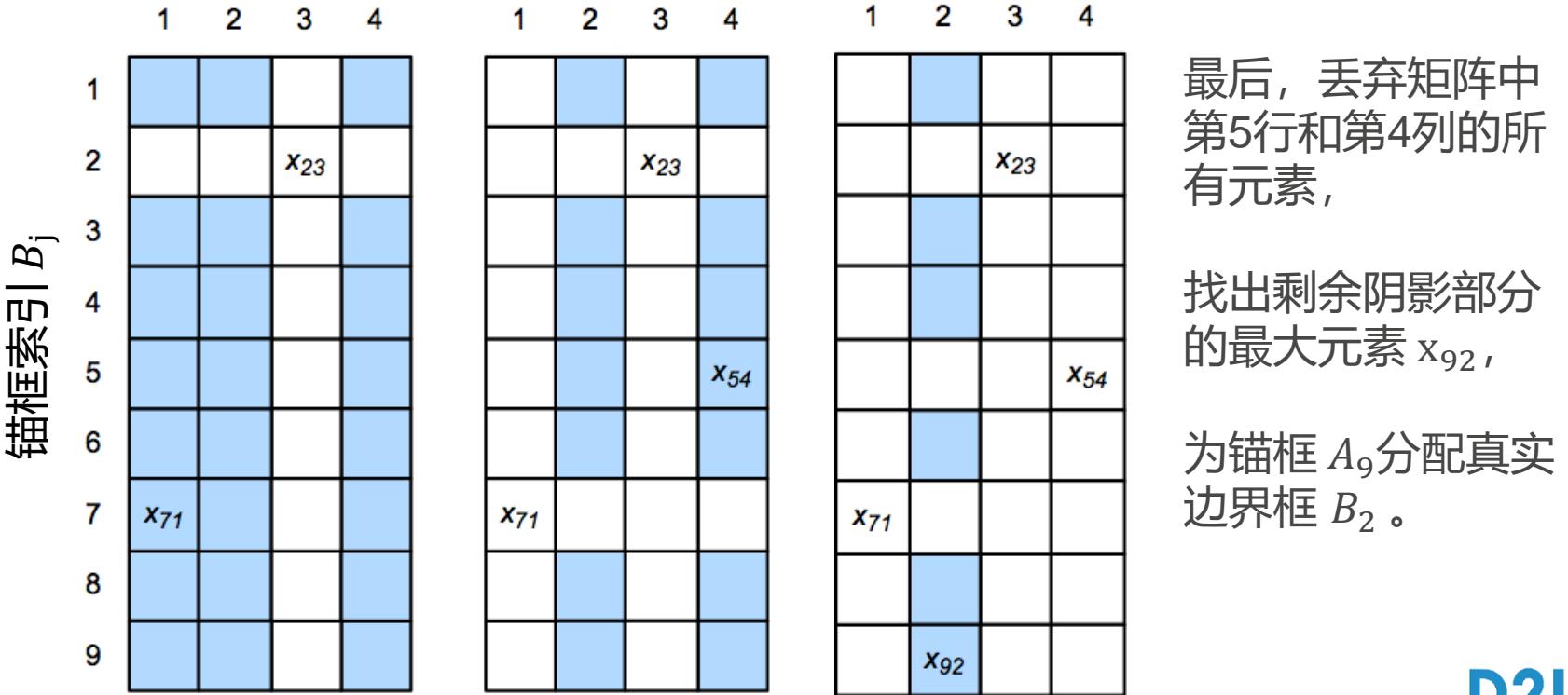
接着，丢弃矩阵中第7行和第1列的所有元素，

找出剩余阴影部分的最大元素 x_{54} ，

为锚框 A_5 分配真实边界框 B_4 。

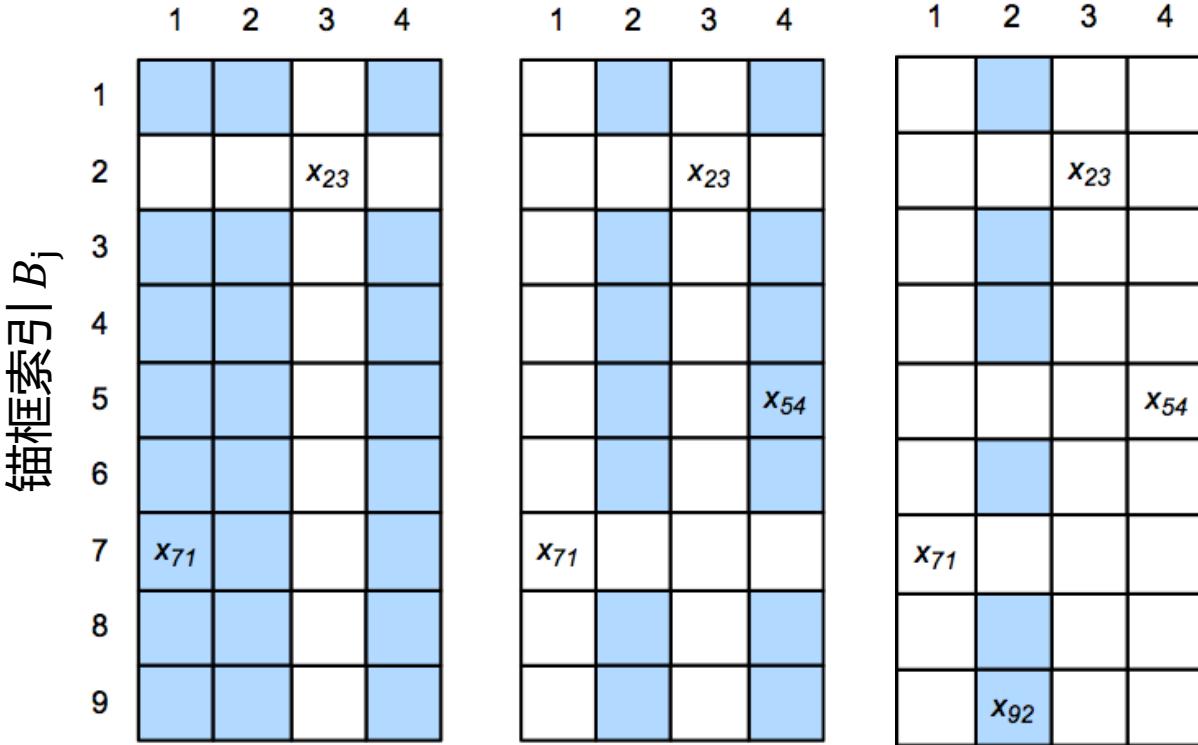
将标签分配给锚框

真实边界索引 A_i



将标签分配给锚框

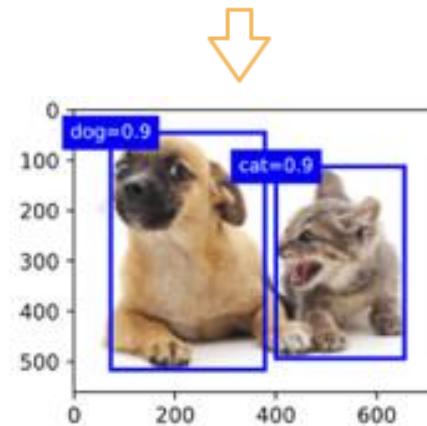
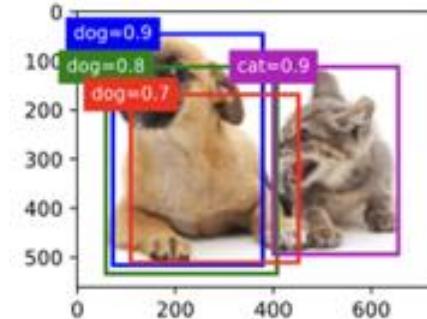
真实边界索引 A_i



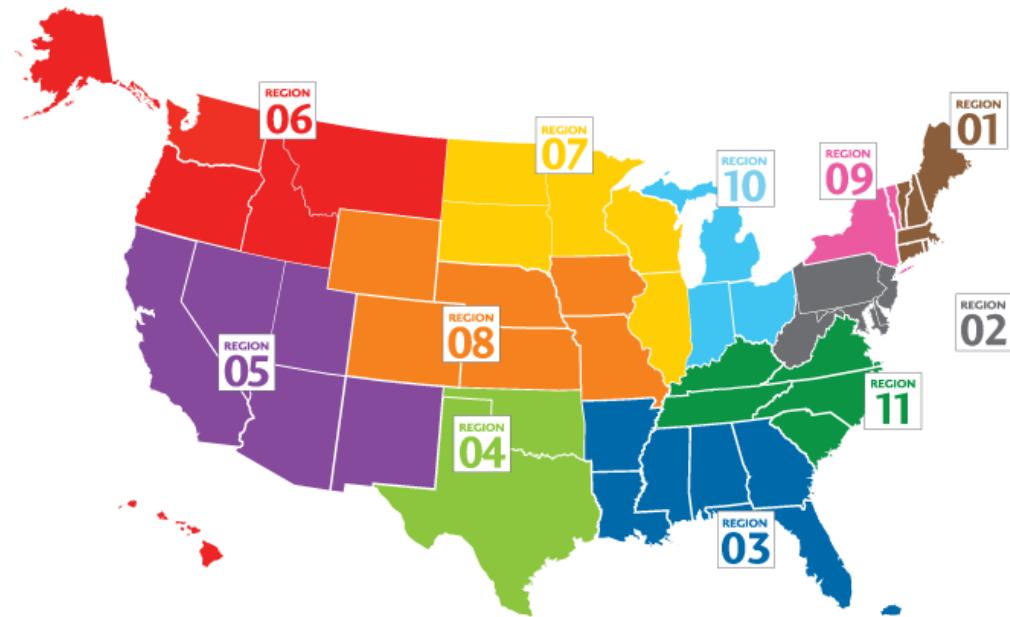
之后，我们只需遍历除去 A_2 , A_5 , A_7 , A_9 的剩余锚框，并根据阈值判断是否为剩余锚框分配真实边界框。

输出预测边界框

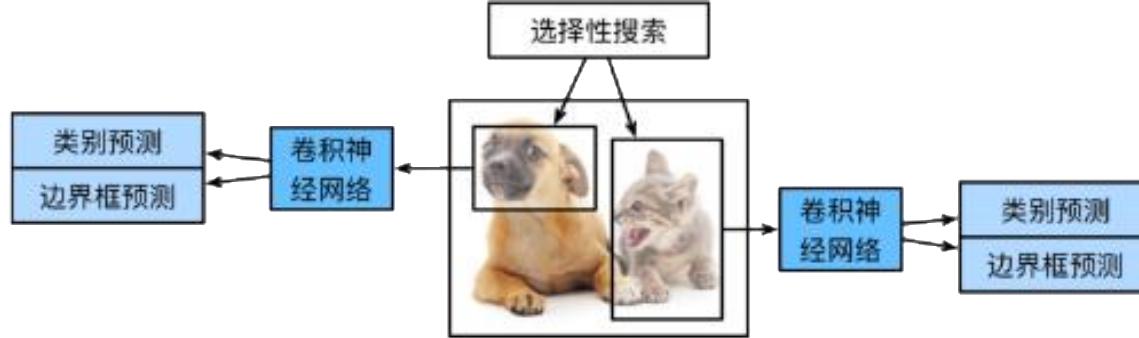
- 非极大值抑制 (non-maximum suppression, NMS)
 - 每个锚框生成一个边界框预测
 - 选择得分最高的那个预测
 - 所有其他预测与所选预测相比，如果 $\text{IoU} > \theta$ ，则删除
 - 重复，直到选中或删除所有内容



区域卷积 神经网络



区域卷积神经网络 (R-CNN)



- 区域卷积神经网络 (region-based CNN, R-CNN)

- 对输入图像使用选择性搜索，来选取多个高质量的提议
- 选取一个预训练的卷积神经网络，并通过前向计算输出抽取的提议区域特征
- 将每个提议区域的特征连同其标注的类别作为一个样本，训练多个支持向量机对目标分类
- 训练线性回归模型来预测真实边界框

RoI 池化层

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

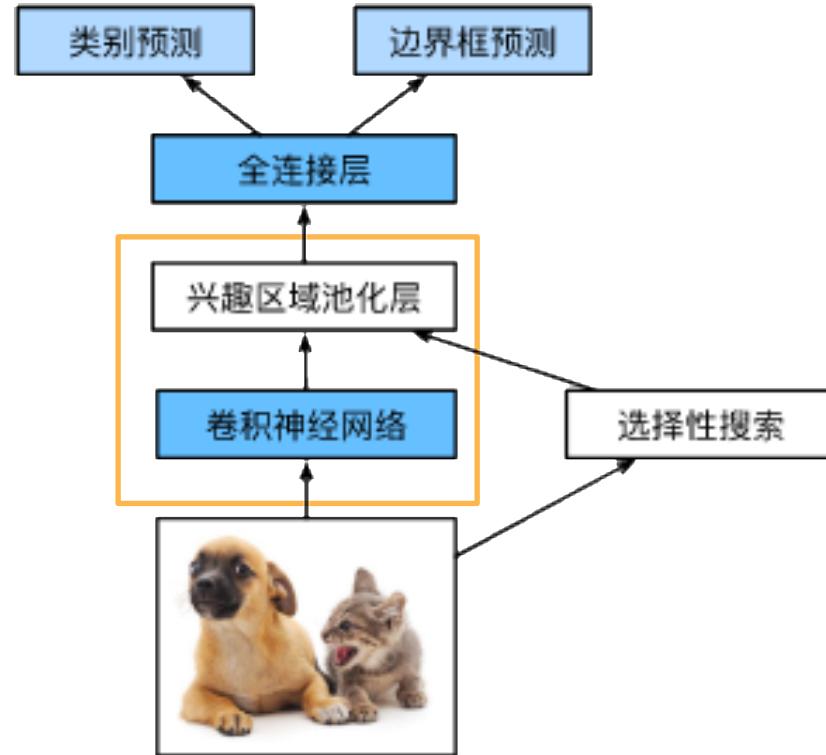
2 × 2 兴趣区
域池化层

5	6
9	10

- 兴趣区域池化层 (region of interest pooling, *RoI* 池化层)
 - 给定一个锚框，将其均匀地切割成 $n \times m$ 个块，输出每个块中的最大值
 - 返回每个锚框的值

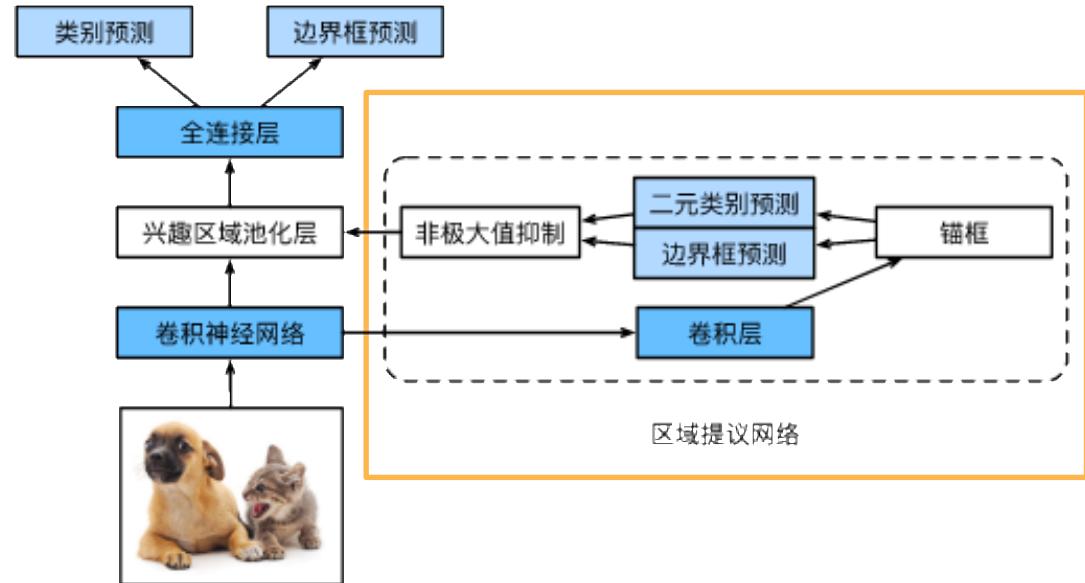
Fast RCNN

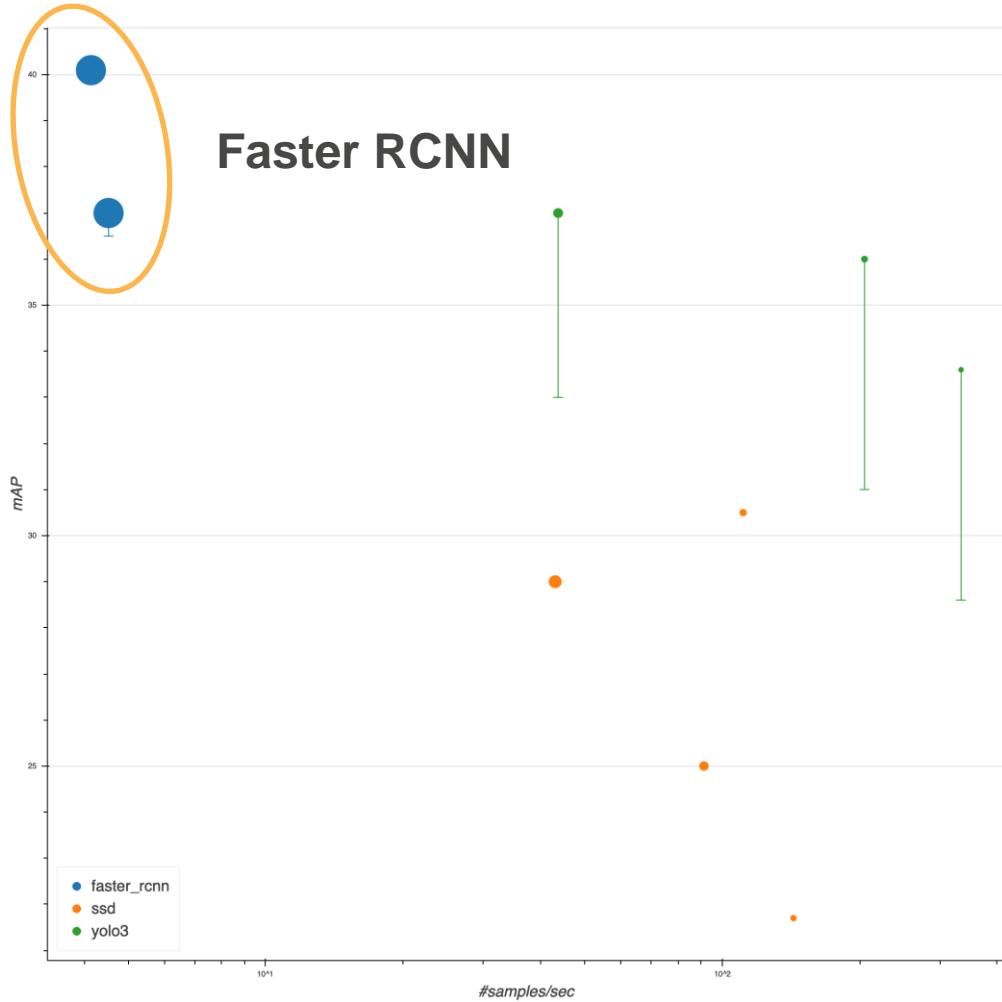
- CNN 快速提取特征：
 - 输入是整个图像，而不是各个提议区域
 - Roi池化为每个锚框返回固定长度的特征



Faster R-CNN

- 与 Fast R-CNN 相比，只有生成提议区域的方法从选择性搜索变成了区域提议网络，而其他部分均保持不变。



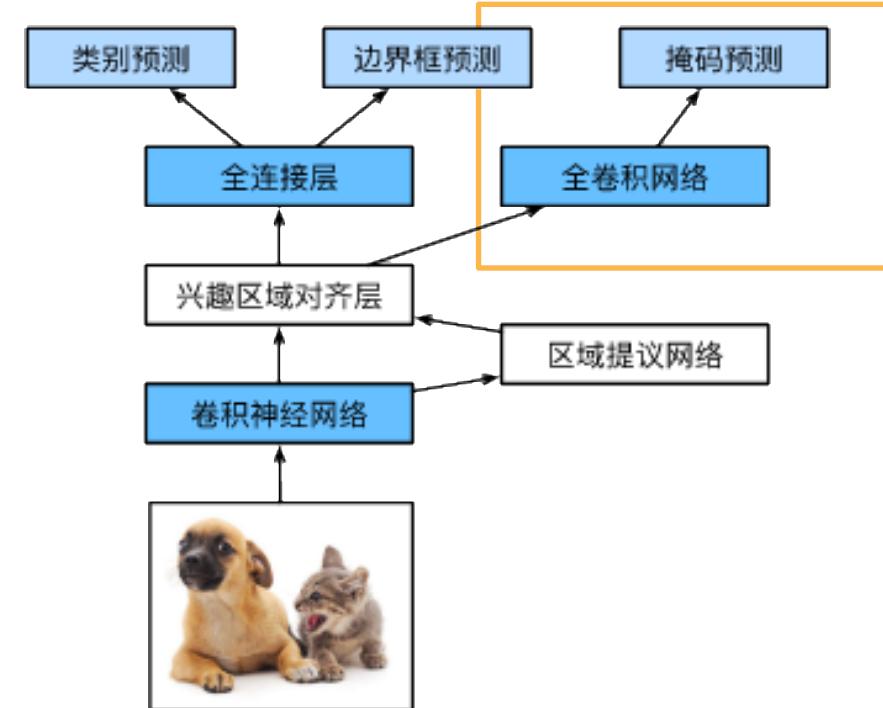
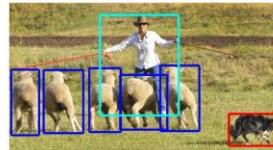


https://gluon-cv.mxnet.io/model_zoo/detection.html

Mask R-CNN

- 如果训练数据还标注了每个目标在图像上的像素级位置，那么 Mask R-CNN 能有效利用这些详尽的标注信息进一步提升目标检测的精度

COCO

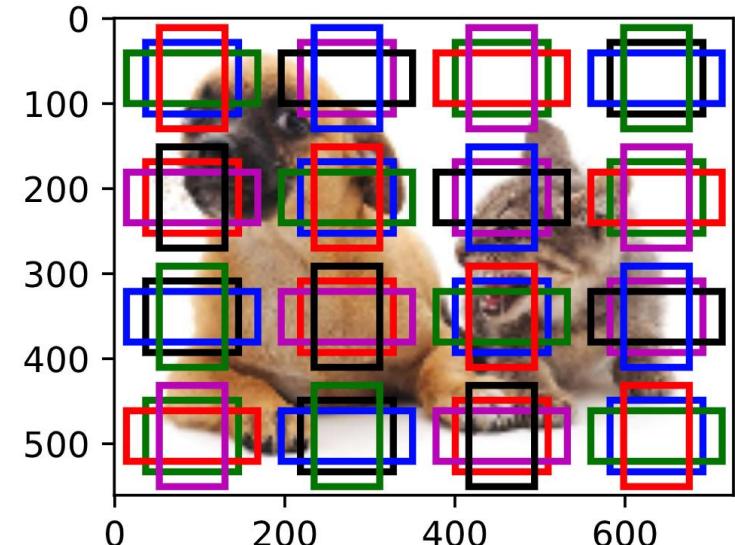


单发多框检测 (SSD)



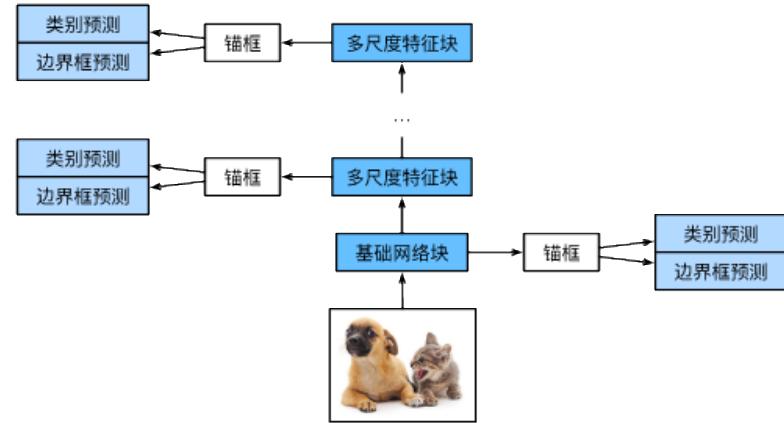
生成锚框

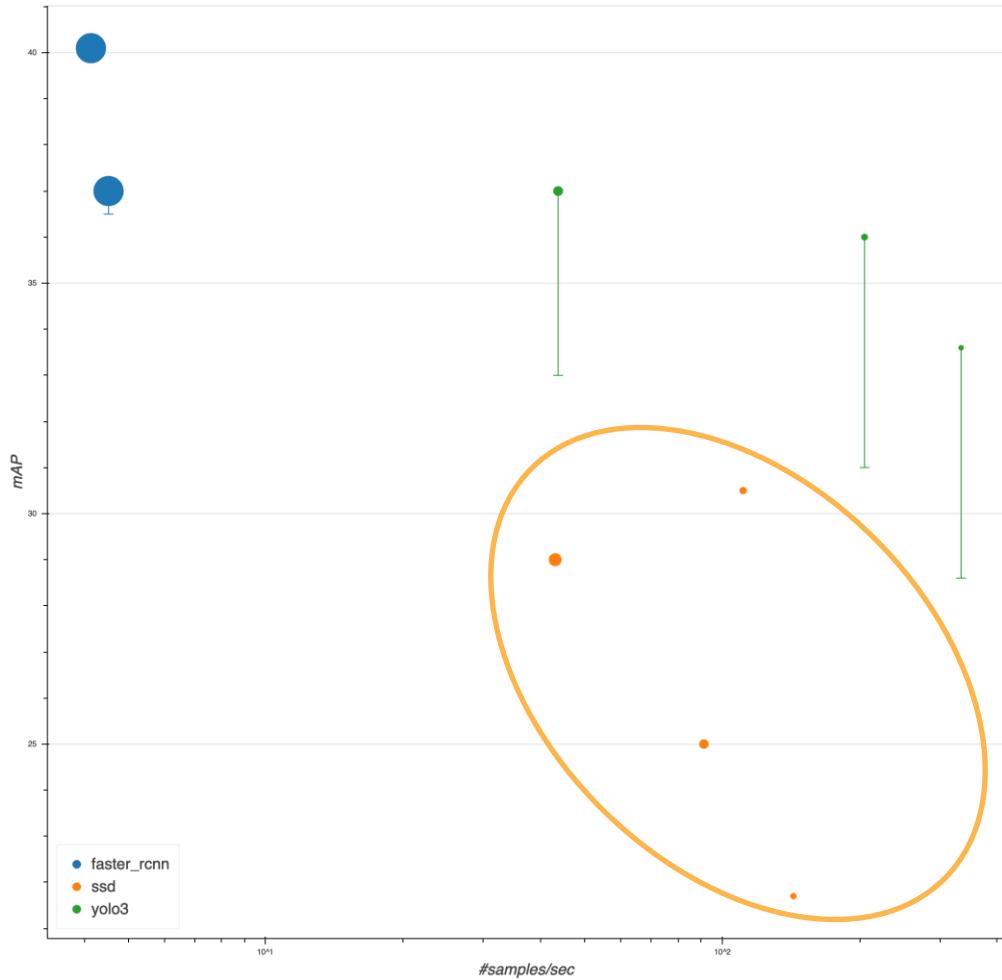
- 对于每个像素，生成以该像素为中心的多个锚框
- 给定 n 个 s_1, \dots, s_n 和 m 个比率 r_1, \dots, r_m ，生成 $n+m-1$ 个锚框 $(s_1, r_1), (s_2, r_1), \dots, (s_n, r_1), (s_1, r_2), \dots, (s_1, r_m)$



单发多框检测 (SSD) 模型

- 单发多框检测是一个多尺度的目标检测模型
 - 先是基础网络从原始图像中抽取特征，使它输出的高和宽较大，可以用来检测尺寸**较小**的目标
 - 接下来的每个多尺度特征块将上一层提供的特征图的高和宽缩小（如减半），并使特征图中每个单元在输入图像上的感受野变得更广阔
 - 越靠近顶部的多尺度特征块输出的特征图越小，故而基于特征图生成的锚框也越少，加之特征图中每个单元感受野越大，因此更适合检测尺寸**较大**的目标预测每个锚框的类和边界框





https://gluon-cv.mxnet.io/model_zoo/detection.html

SSD

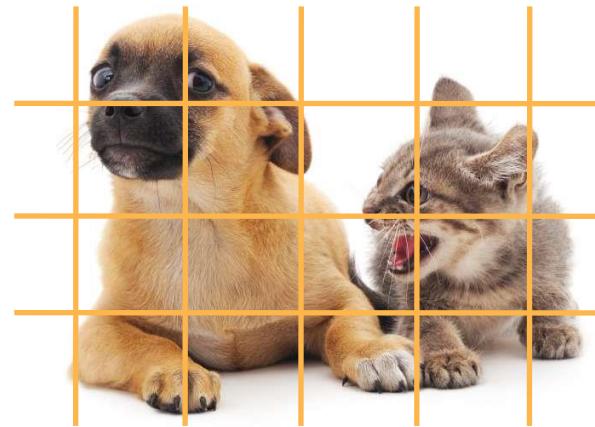
D2L.ai

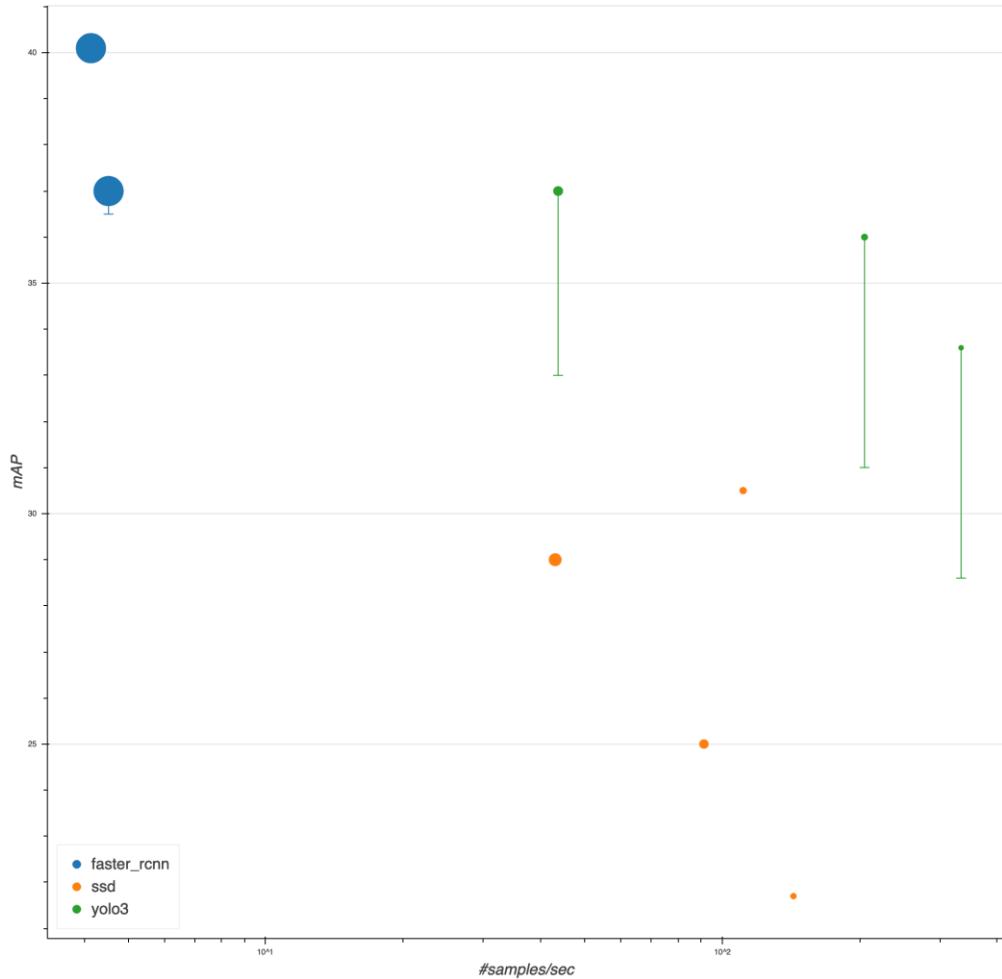
你只需看一次
(YOLO)



YOLO

- Joseph Redmon et al., 2015
- 锚框在 SSD 中高度重叠
- YOLO 将输入图像均匀地切割成 $S \times S$ 锚框
- 每个锚框预测B个边界框
- V2 和 V3 增加了更多改进





https://gluon-cv.mxnet.io/model_zoo/detection.html

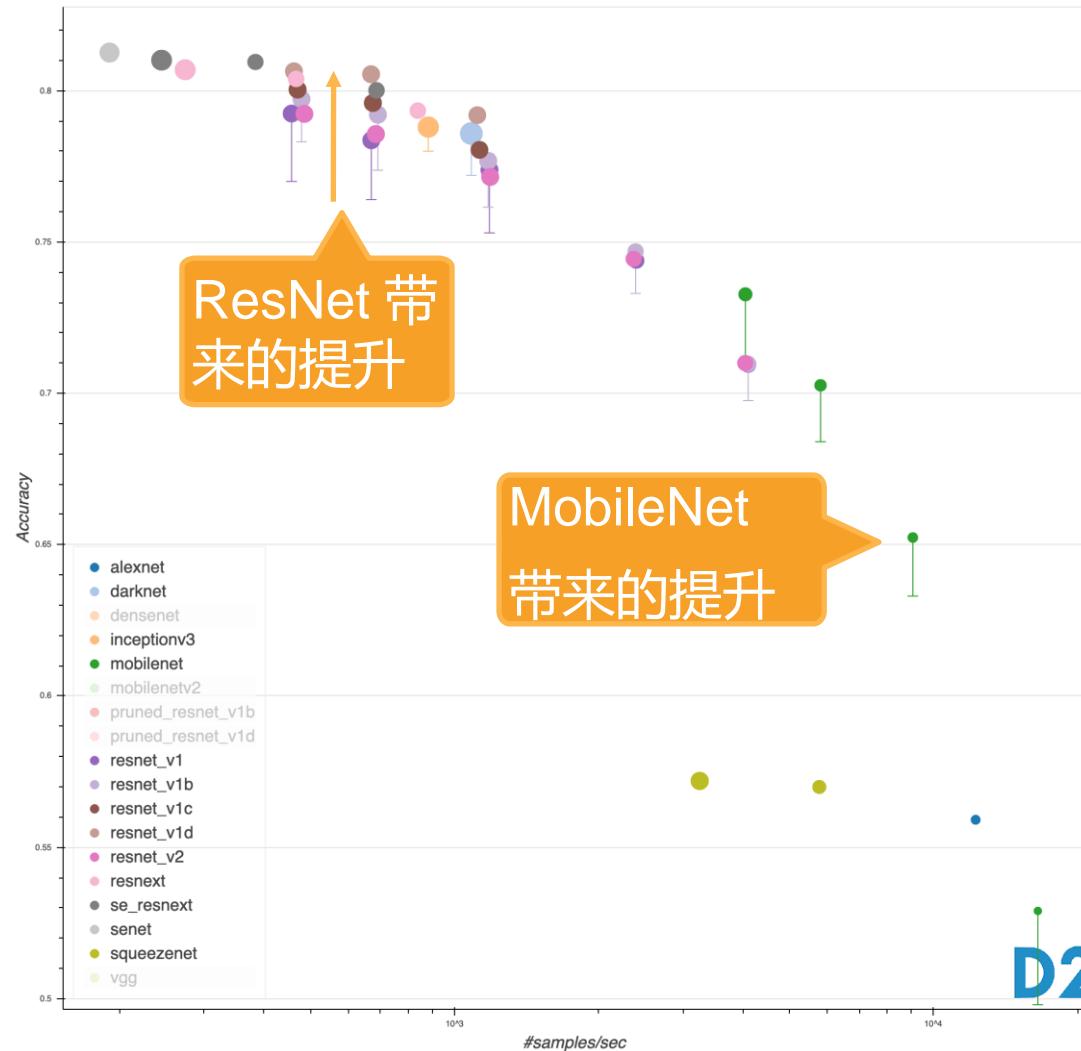
Yolo V3

计算机视觉 训练技巧



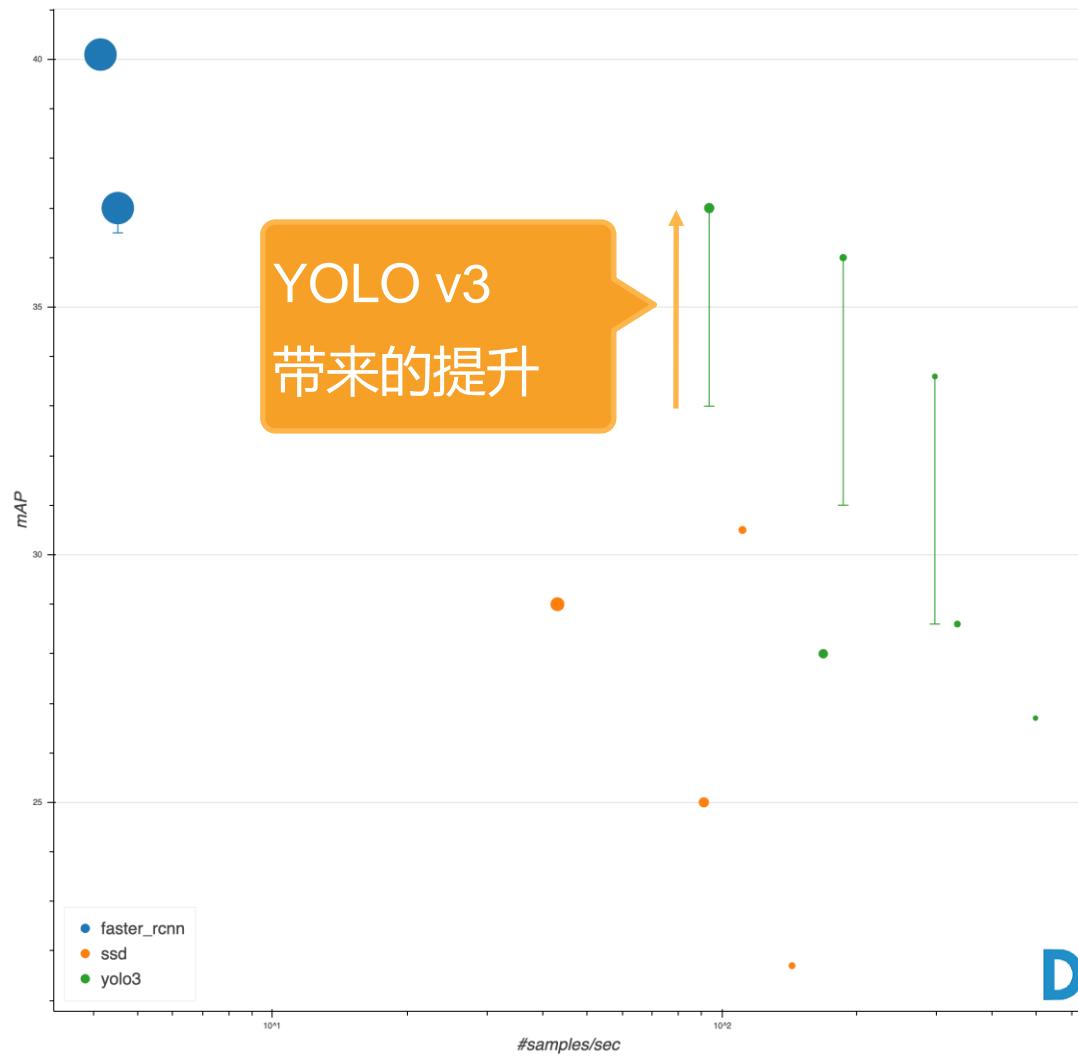
各种训练技巧大大提高了图像分类模型的准确性

GluonCV “模型动物园”
https://gluon-cv.mxnet.io/model_zoo/classification.html



训练技巧也适用于提升目标物体检测模型

GluonCV “模型动物园”
https://gluon-cv.mxnet.io/model_zoo/detection.html



混合训练数据

- 随机选择两个例子 i 和 j , 采样随机数 $\lambda \in [0,1]$
- 计算: $x = \lambda x_i + (1 - \lambda)x_j, \quad y = \lambda y_i + (1 - \lambda)y_j$
- 例如:



bittern	0
...	0
otter	0
...	0
analog_clock	1

* 0.9 +



bittern	1
...	0
otter	0
...	0
analog_clock	0

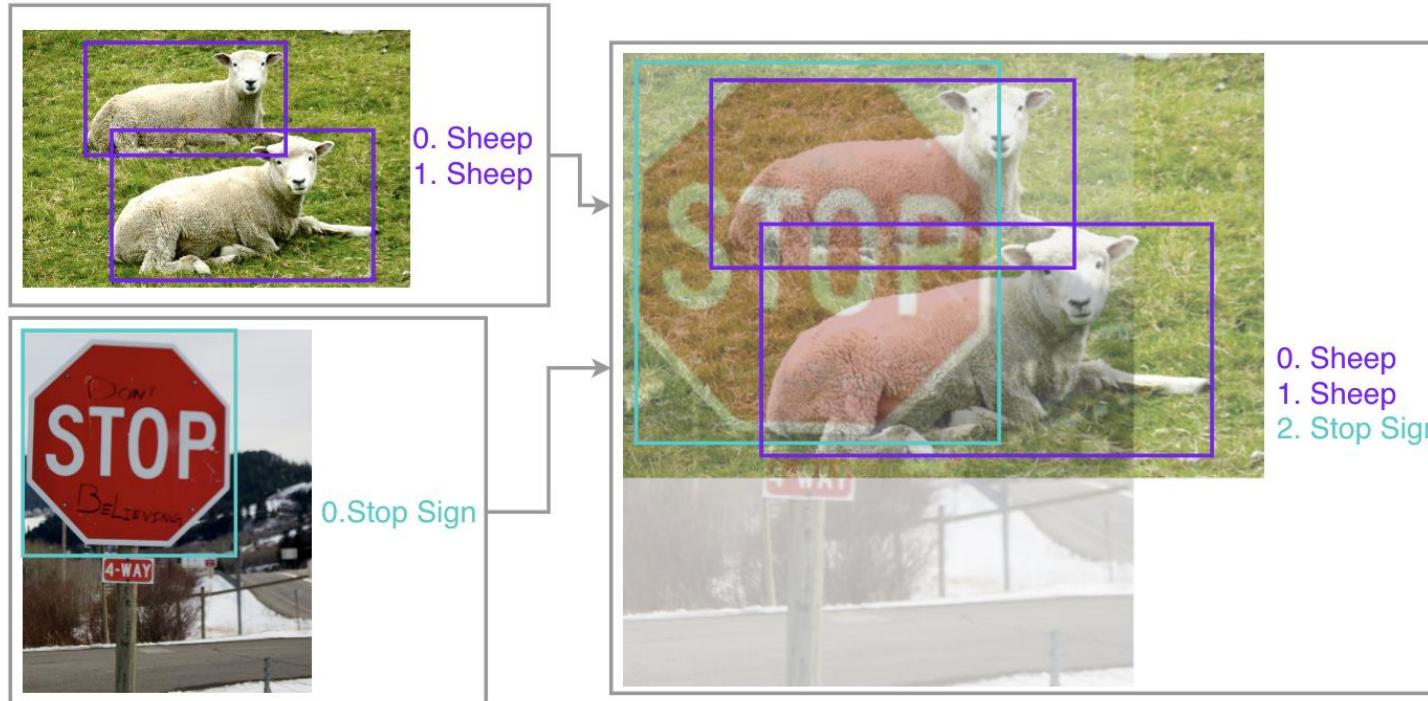
* 0.1 =



bittern	0.1
...	0
otter	0
...	0
analog_clock	0.9

混合训练数据

- 也适用于目标物体检测模型



标签平滑化

- 假设标签 $y \in \mathbb{R}^n$ 是one-hot编码

$$y_i = \begin{cases} 1 & \text{if belongs to class } i \\ 0 & \text{otherwise} \end{cases}$$

- 用 softmax 逼近 0/1 值很难
- 标签平滑化

$$y_i = \begin{cases} 1 - \epsilon & \text{if belongs to class } i \\ \epsilon/(n - 1) & \text{otherwise} \end{cases}$$

- 常用 $\epsilon = 0.1$

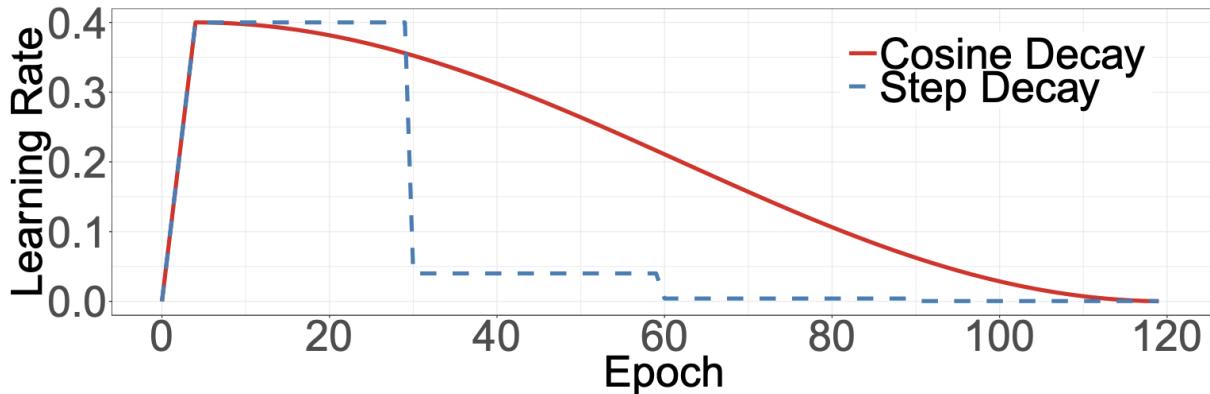
学习率预热

- 在随机初始化参数时， 使用大的学习率可能导致数值稳定性问题
- 预热技巧：在开始时使用较小的学习率，然后将其增加到原有值。例如：
 - 如果我们选择初始学习率为0.1并使用5个时期进行预热
 - 学习速率从0开始，在前5个时期将其线性增加到0.1

余弦衰减

- 为了趋近，我们需要降低 SGD 学习率
- 例如，在 30, 60 和 90 迭代周期各减少 10 倍
- 假设在总 T 次迭代（批次）中，余弦衰减计算迭代时的学习率 t

$$\eta_t = \frac{1}{2}(1 + \cos(t\pi/T))\eta$$



同步批量归一化

- BatchNorm 需要大批量才能获得可靠的统计信息
- 由于 GPU 存储器限制，目标对象检测任务用小批量。
 - 例如，每 GPU 1个图像
- 在多 GPU 培训中，每个 GPU 分别计算均值/方差
- 同步批量归一化计算所有 GPU 的统计信息

随机批量调整形状

- 图像被批量调整为相同的形状，例如， 224 宽度和 224 高度
- 我们可以改变这种形状：
 - 对于每个批次，从224 (7x32) , 256 (8x32) , 228 (9x32) , ... 中选择随机宽度/高度
 - 将所有图像调整为此形状

图像分类

Refinements	ResNet-50-D		Inception-V3		MobileNet	
	Top-1	Δ	Top-1	Δ	Top-1	Δ
Efficient	77.16		77.50		71.90	
+ cosine decay	77.91	+0.75	78.19	+0.69	72.83	+0.93
+ label smoothing	78.31	+0.4	78.40	+0.21	72.93	+0.1
+ mixup	79.15	+0.84	78.77	+0.37	73.28	+0.35

Hang et.al *Bag of Tricks for Image Classification with Convolutional Neural Networks*

YOLO v3 結果

Incremental Tricks	mAP	Δ	Cumu Δ
- data augmentation	64.26	-15.99	-15.99
baseline	80.25	0	0
+ synchronize BN	80.81	+0.56	+0.56
+ random training shapes	81.23	+0.42	+0.98
+ cosine lr schedule	81.69	+0.46	+1.44
+ class label smoothing	82.14	+0.45	+1.89
+ mixup	83.68	+1.54	+3.43

Zhi et al, *Bag of Freebies for Training Object Detection Neural Networks*

总结

- 目标检测
 - 边界框和锚框
 - 交并比
 - 区域卷积神经网络 (R-CNN)
 - 单发多框检测 (SSD)
- 计算机视觉训练技巧

动手学深度学习

18. 序列模型

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/sequence.html>

概要

- 序列模型
 - 马尔可夫 (Markov)
 - 隐马尔可夫模型 (HMM)
- 语言模型
 - 文本预处理

相关的随机变量

数据

- 至今 ...
- 收集观察数据对 $(x_i, y_i) \sim p(x, y)$ 进行训练
- 对于看不见的 $x' \sim p(x)$, 估计 $y|x \sim p(y|x)$
- 例子:
 - 图像和目标对象
 - 回归问题
 - 房子和房价
- **数据的顺序无关紧要**

训练 ≠ 测试

- 泛化表现 (经验分布)

$$p_{\text{emp}}(x, y) \neq p(x, y)$$

- 协变量偏移 (协变量分布)

$$p(x) \neq q(x)$$

- 逻辑回归 (修复偏移的工具)

$$\log(1 + \exp(-yf(x)))$$

- 协变量偏移校正

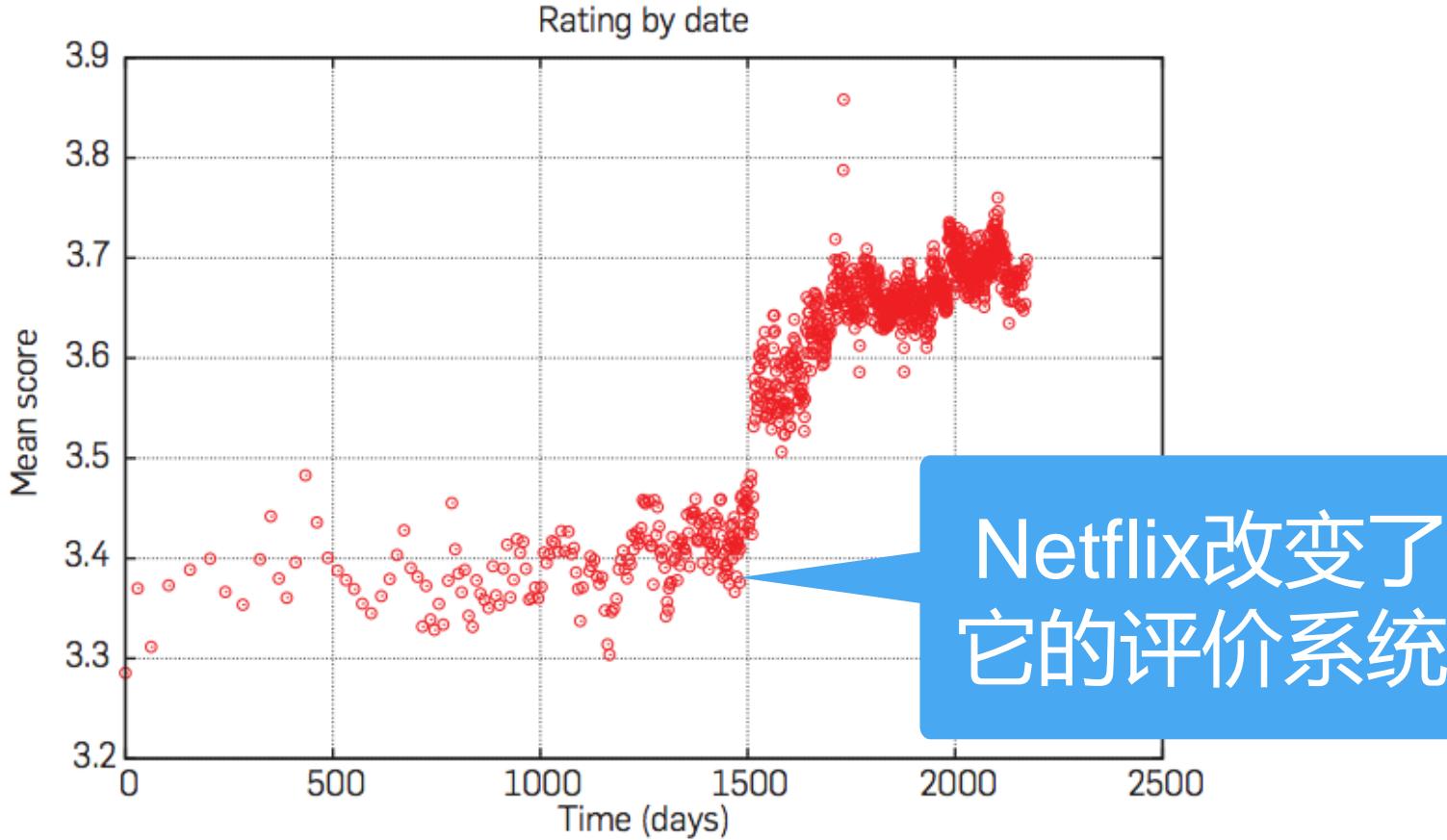
$$\frac{1}{2} (p(x)\delta(1, y) + q(x)\delta(-1, y))$$

- 标签转移 (标签分布所在)

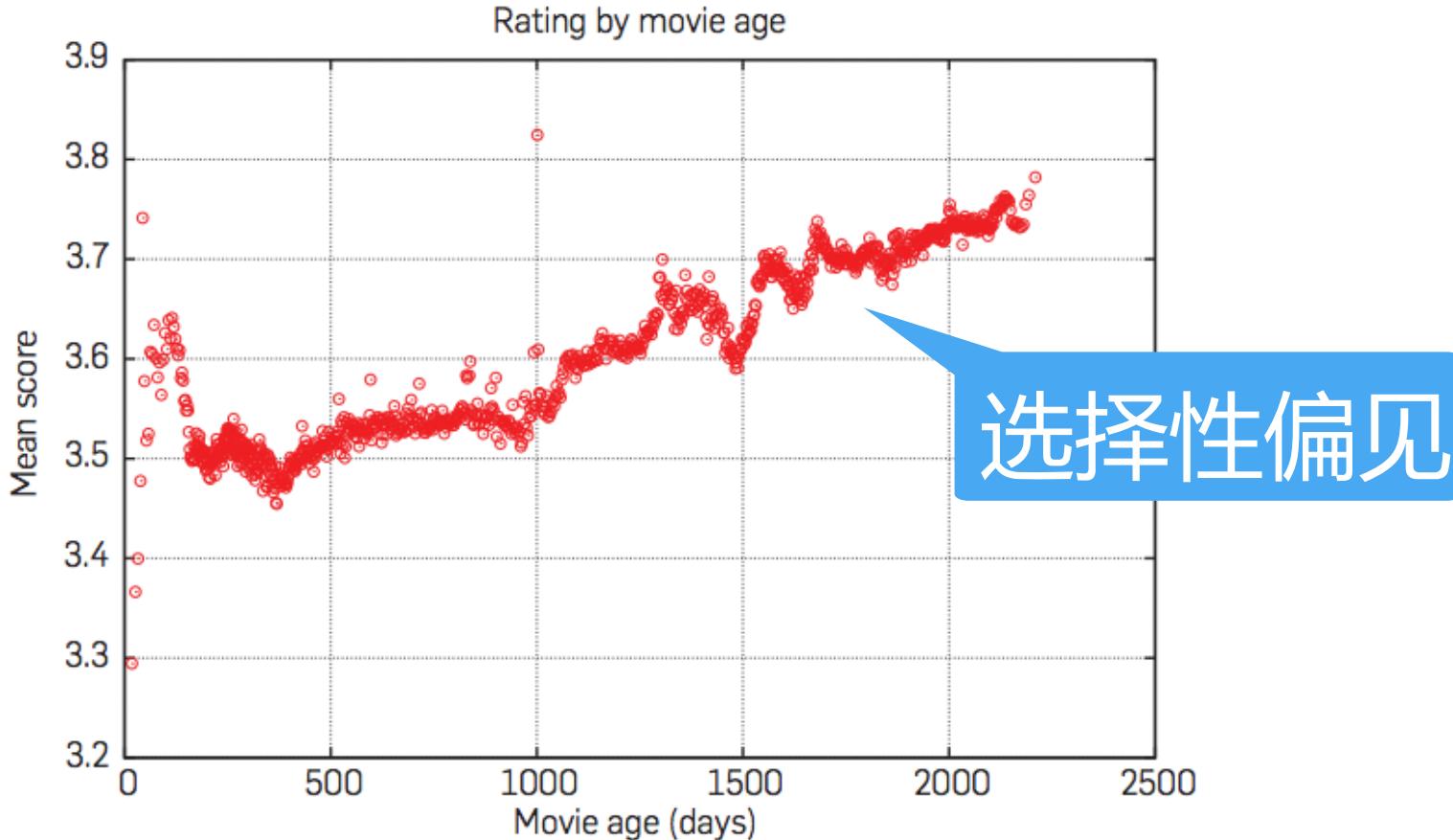
$$p(y) \neq q(y)$$

- 非平稳环境

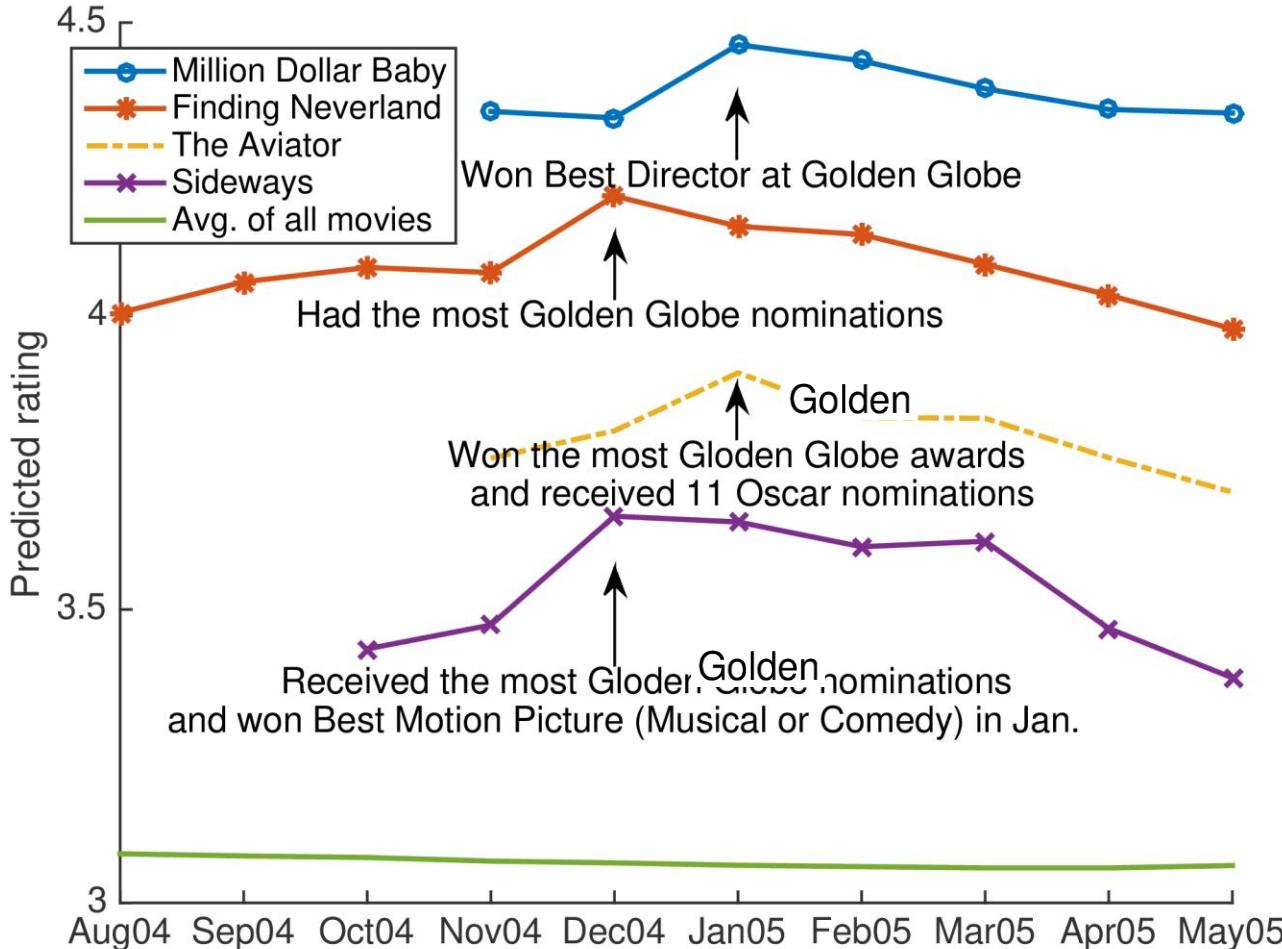
时间重要性 (Koren, 2009)



时间重要性 (Koren, 2009)

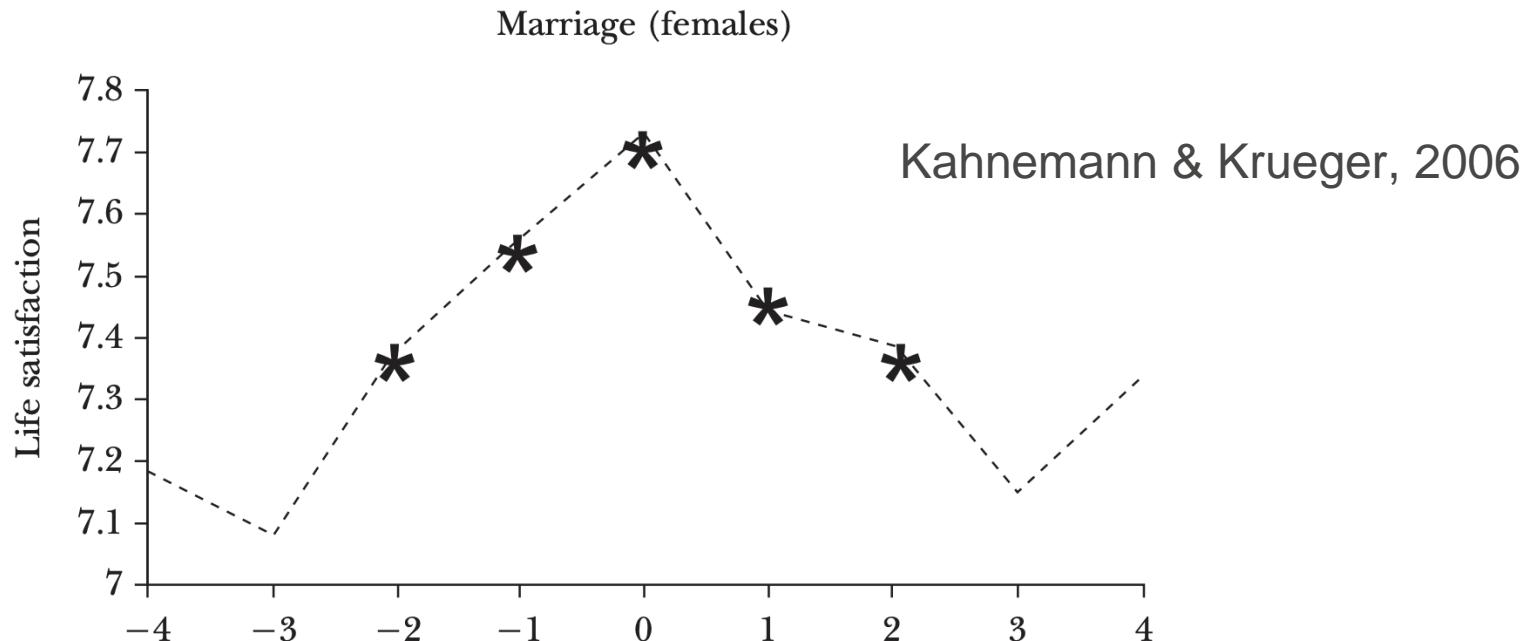


电影评分与奖项



Beutel et al., 2015

Average Life Satisfaction for a Sample of German Women (by year of marriage $t = 0$)



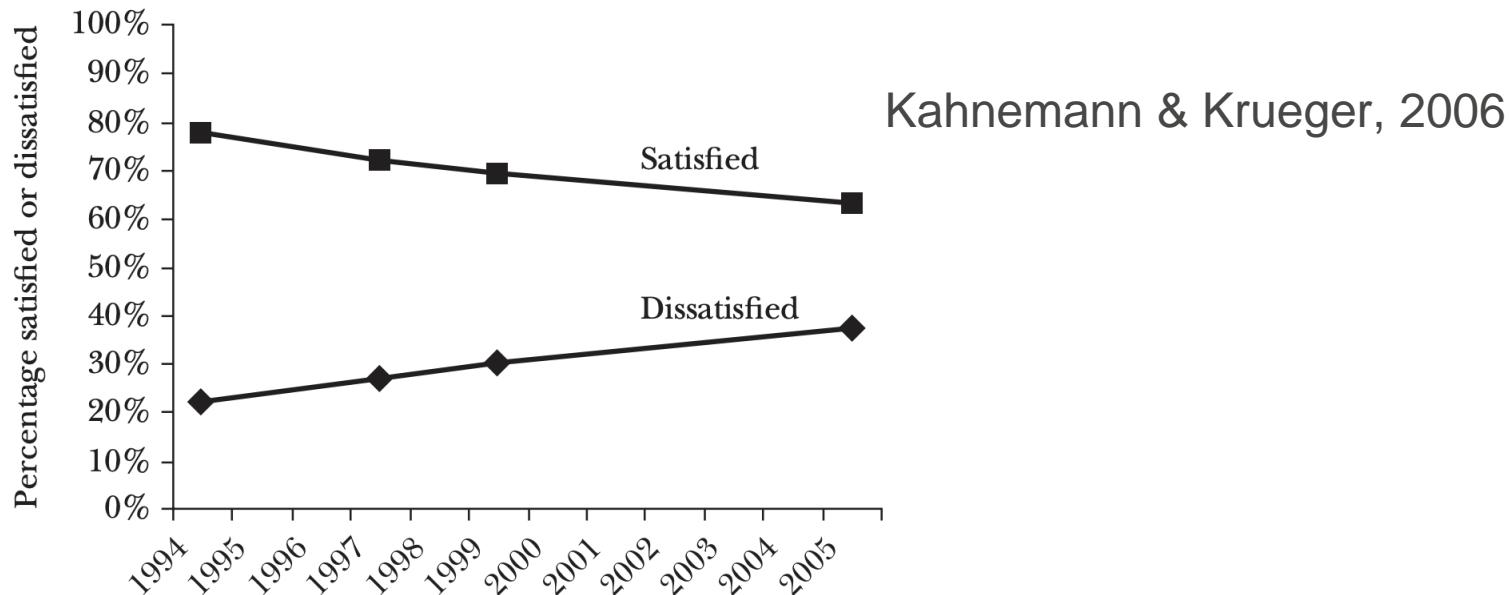
Source: Clark, Diener, Georgellis and Lucas (2003), using data from the German Socioeconomic Panel.

Note: An asterisk indicates that life satisfaction is significantly different from the baseline level.

Life Satisfaction in China as Average Real Income Rises by 250 Percent

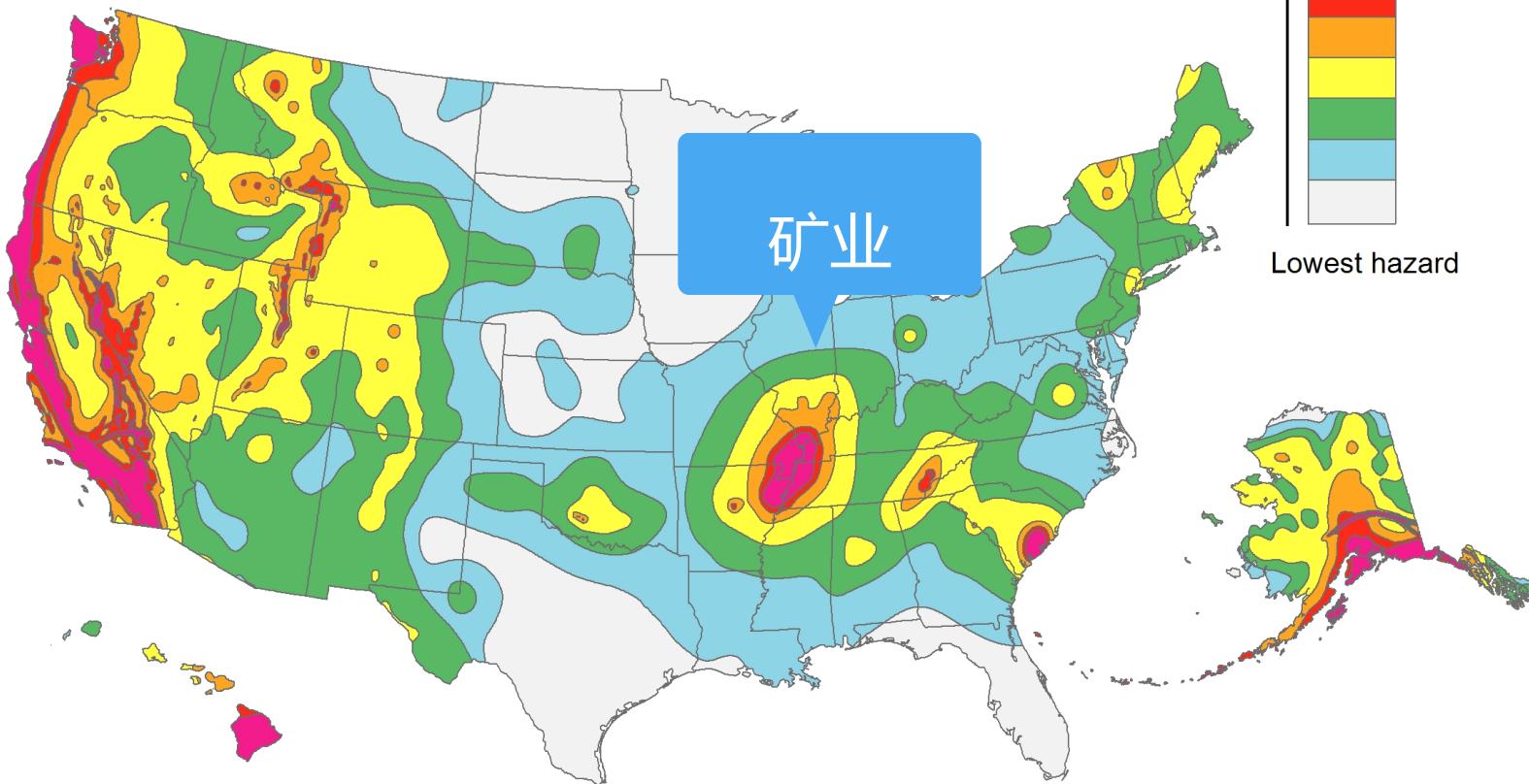
Overall, how satisfied or dissatisfied are you with the way things are going in your life today?

Would you say you are very satisfied, somewhat satisfied, somewhat dissatisfied, or very dissatisfied?



Source: Derived from Richard Burkholder, "Chinese Far Wealthier Than a Decade Ago—but Are They Happier?" The Gallup Organization, <<http://www.gallup.com/poll/content/login.aspx?ci=14548>>.

空间重要性 – 地震预测



FTSE 100



TL;DR –
数据通常不是完全相同且相互独立

序列模型

序列模型

- 相关的随机变量

$$(x_1, \dots x_T) \sim p(x)$$

- 条件概率展开

$$p(x) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_1, x_2) \cdot \dots p(x_T|x_1, \dots x_{T-1})$$

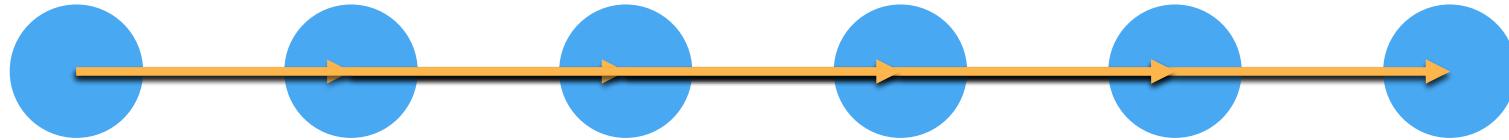
- 总能找到这种展开式
- 也可以找到反向...

$$p(x) = p(x_T) \cdot p(x_{T-1}|x_T) \cdot p(x_{T-2}|x_{T-1}, x_T) \cdot \dots p(x_1|x_2, \dots x_T)$$

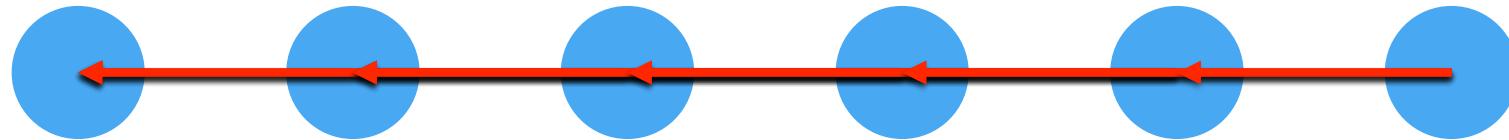
- 所以为什么要这么麻烦?

序列模型

$$p(x) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_1, x_2) \cdot \dots p(x_T|x_1, \dots x_{T-1})$$



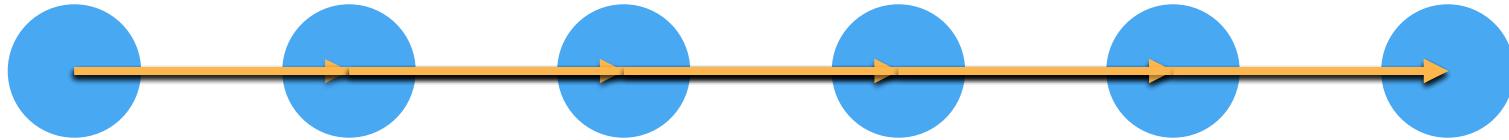
$$p(x) = p(x_T) \cdot p(x_{T-1}|x_T) \cdot p(x_{T-2}|x_{T-1}, x_T) \cdot \dots p(x_1|x_2, \dots x_T)$$



- 因果关系阻止反方向
- 模型的“错误”方向通常要复杂得多

序列模型

$$p(x) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_1, x_2) \cdot \dots \cdot p(x_T|x_1, \dots, x_{T-1})$$



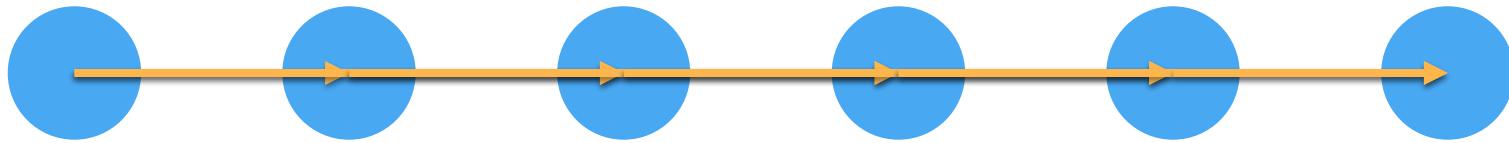
- 自回归模型

$$p(x_t|x_1, \dots, x_{t-1}) = p(x_t|f(x_1, \dots, x_{t-1}))$$

关于以前看到的数据的函数

计划 A - 马尔可夫 (Markov) 假设

$$p(x) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_1, x_2) \cdot \dots \cdot p(x_T|x_{T-\tau}, \dots, x_{T-1})$$



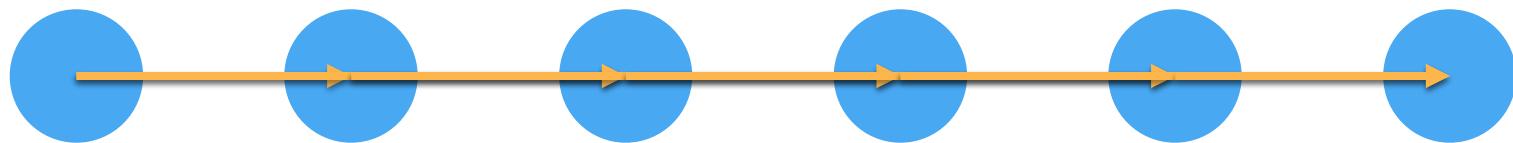
- 假设过去只有固定几个步骤
- 自回归模型

$$p(x_t|x_1, \dots, x_{t-1}) = p(x_t|f(x_{t-\tau}, \dots, x_{t-1}))$$

关于以前看到的数据的函数

计划 A - 马尔可夫 (Markov) 假设

$$p(x) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_1, x_2) \cdot \dots \cdot p(x_T|x_{T-\tau}, \dots, x_{T-1})$$



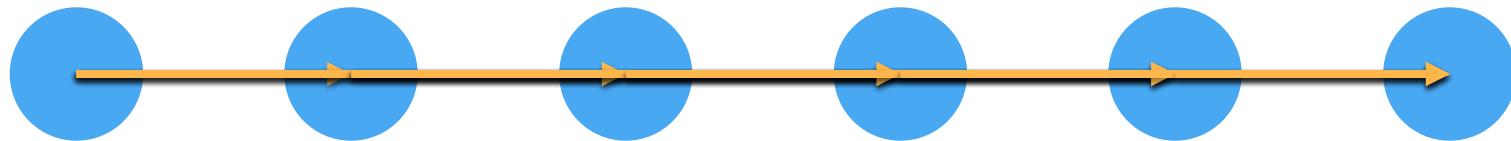
- 在实践中解决回归问题

$$\hat{x}_t = f(x_{t-\tau}, \dots, x_{t-1})$$

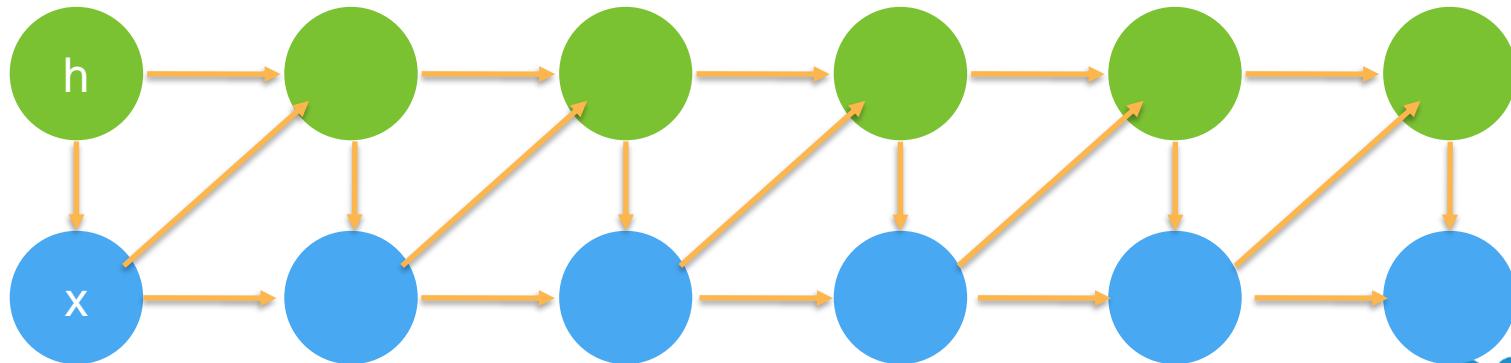
例如 在先前看到的数据上训练MLP

计划 B – 隐马尔可夫模型

$$p(x) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_1, x_2) \cdot \dots \cdot p(x_T|x_1, \dots, x_{T-1})$$



$$p(h_t|h_{t-1}, x_{t-1}), \quad p(x_t|h_t, x_{t-1})$$

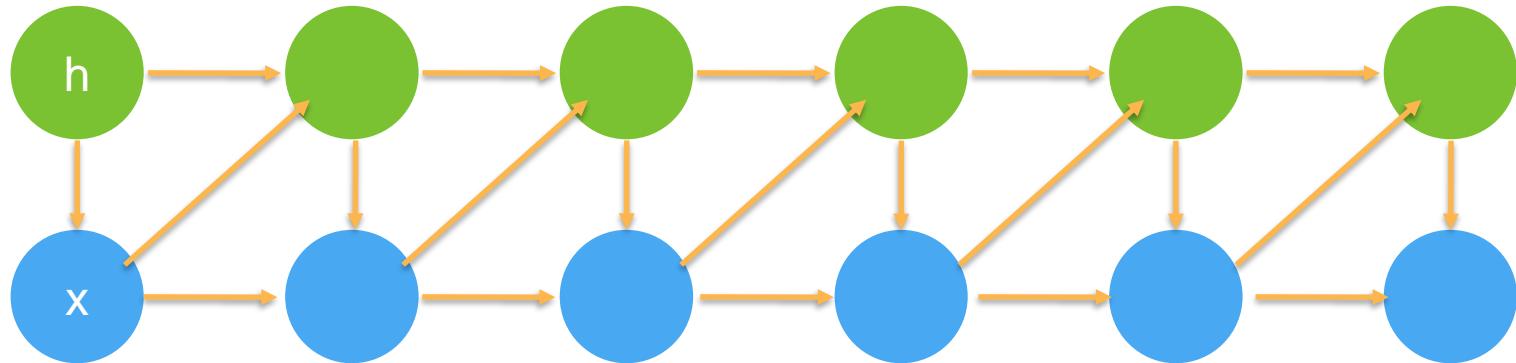


计划 B – 隐马尔可夫模型

- 隐含状态总结了有关过去的所有相关信息：

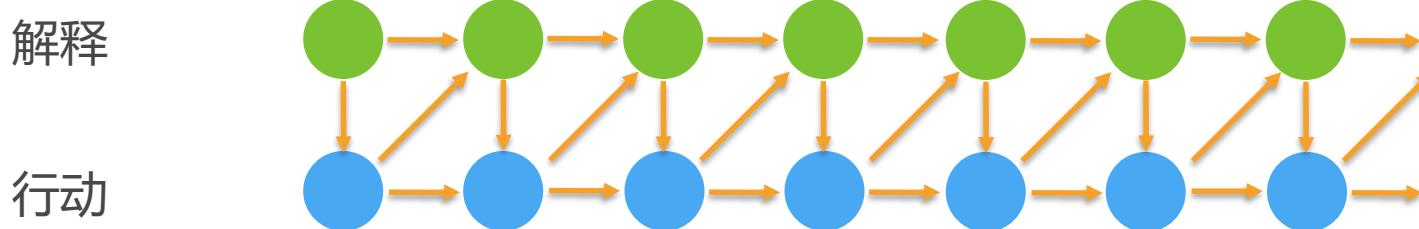
$$h_t = f(x_1, \dots x_{t-1}) = f(h_{t-1}, x_{t-1})$$

$$p(h_t|h_{t-1}, x_{t-1}), \quad p(x_t|h_t, x_{t-1})$$



隐变量模型 (经典)

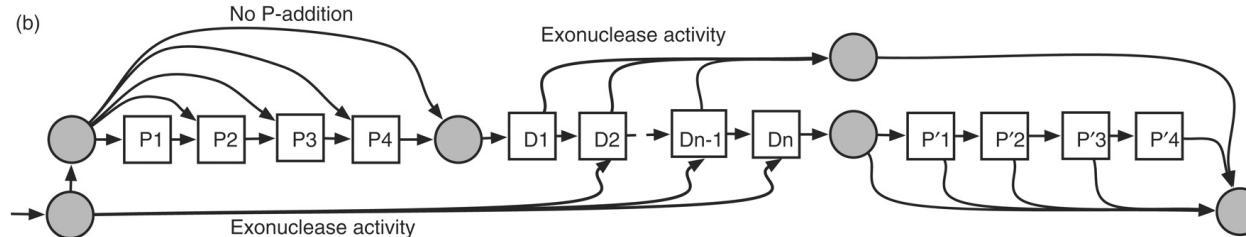
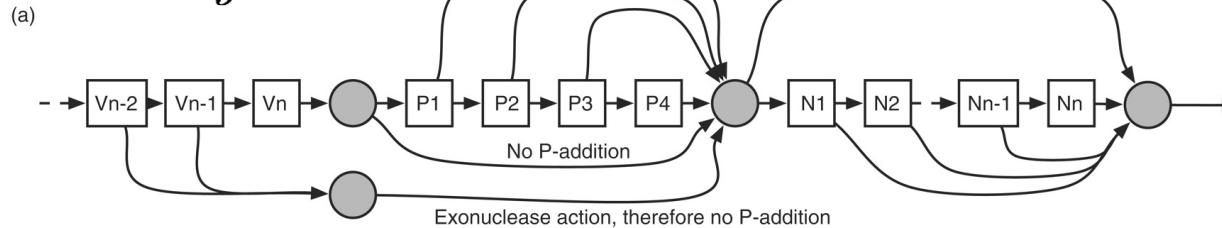
- 时间序列：
 - 购买, 喜欢, 应用程序使用, 电子邮件, 广告点击, 查询, 评级
- 隐含状态解释行为：
 - 集群 (搜索中的导航, 信息查询)
 - 主题 (用户偏好不同)
 - 卡尔曼滤波器 (Kalman Filter) (轨迹和位置建模)



时间聚类 - 隐马尔可夫模型

- 具有顺序依赖性的集群

$$p(x) = \sum_y \prod_{i=1}^n p(y_i|y_{i-1})p(x_i|y_i)$$

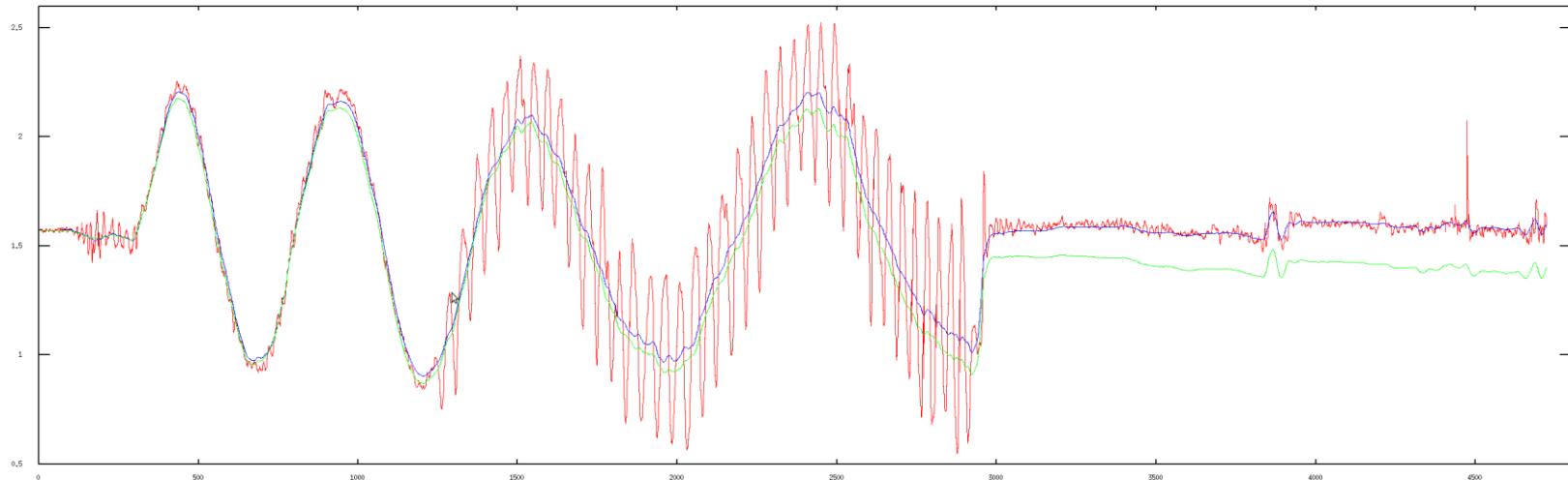


时间PCA - 卡尔曼滤波器

- 潜在因子变量
- 简单的顺序因子结构

$$x_t \sim \mathcal{N}(Ay_t + \mu, K)$$

$$y_t \sim \mathcal{N}(By_{t-1} + \nu, L)$$

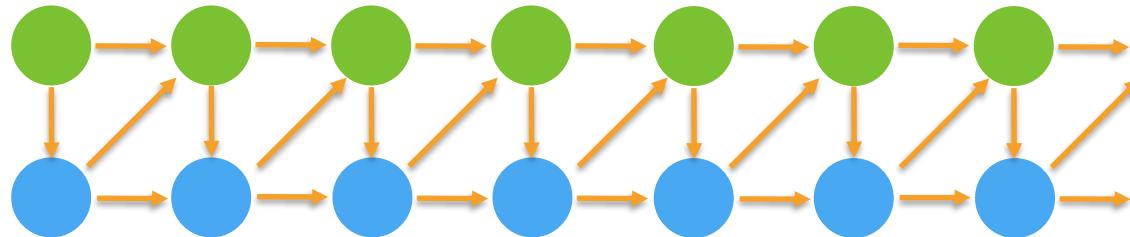


序列模型

- 时间序列：
 - 购买, 喜欢, 应用程序使用, 电子邮件, 广告点击, 查询, 评级
- 隐含状态解释行为：

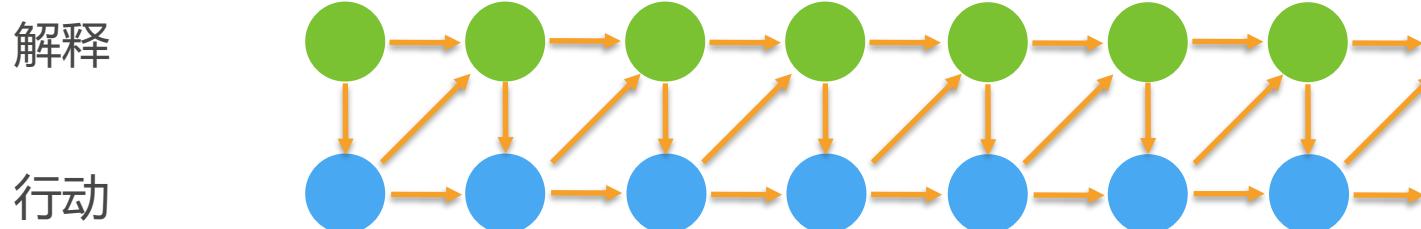
参数模型真的如此吗?

解释
行动



序列模型

- 时间序列：
 - 购买, 喜欢, 应用程序使用, 电子邮件, 广告点击, 查询, 评级
- 状态域：
 - 变量深度, 变量表示, 变量类型
- 时间分辨率：
 - 数据不以量化间隔到达



计划 A - 马尔可夫 (Markov) 假设

马尔可夫 (Markov) 假设

- 下一次观察仅取决于过去的几个观察

$$\hat{x}_t = f(x_{t-\tau}, \dots x_{t-1})$$

- 训练回归模型
- 用它来预测下一步并进行迭代

语言模型

语言模型 101

- 标记不一定是真正的值 (域是有限的)

$$p(w_1, w_2, \dots, w_T) = \prod_{t=1}^T p(w_t | w_1, \dots, w_{t-1})$$

$$\begin{aligned} & p(\text{Statistics}, \text{is}, \text{fun}, \cdot) \\ = & p(\text{Statistics})p(\text{is}|\text{Statistics})p(\text{fun}|\text{Statistics}, \text{is})p(\cdot | \text{Statistics}, \text{is}, \text{fun}) \end{aligned}$$

- 估计

$$\hat{p}(\text{is}|\text{Statistics}) = \frac{n(\text{Statistics}\text{is})}{n(\text{Statistics})}$$

N-grams (更长的标记序列)

- 需要更平滑 (长的 N-grams 序列很少)

$$\hat{p}(w) = \frac{n(w) + \epsilon_1/m}{n + \epsilon_1}$$

$$\hat{p}(w' | w) = \frac{n(w, w') + \epsilon_2 \hat{p}(w')}{n(w) + \epsilon_2}$$

$$\hat{p}(w'' | w', w) = \frac{n(w, w', w'') + \epsilon_3 \hat{p}(w', w'')}{n(w, w') + \epsilon_3}$$

我们来看看实际的语言统计数据

文本预处理

符号化

- 基本理念 - 将文本映射到 ID 序列
- 字符编码 (每个字符都有一个 ID)
 - 小词汇量
 - 效果不好 (DNN 需要学习拼写)
- 单词编码 (每个单词有一个 ID)
 - 准确的拼写
 - 效果不好 (巨大的词汇量 = 昂贵的多项式)
- 字节对编码 (黄金区)
 - 频繁的子序列 (如音节)

小批量生成

The Time Machine by H. G. Wells

小批量生成

- 随机分区
 - 选择随机偏移
 - 通过小批量随机分配序列
 - 样本独立性
 - 需要重置隐含状态

The Time Machine by H. G. Wells

小批量生成

- 顺序分区
 - 选择随机偏移
 - 通过小批量按顺序分配序列
 - 相关样本
 - 保持小批量的隐含状态 (更好)

The Time Machine by H. G. Wells

代码 ...

总结

- 序列模型
 - 马尔可夫 (Markov)
 - 隐变量模型 (HMM)
- 语言模型
 - 文本预处理

动手学深度学习

19. 循环神经网络

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/rnn.html>

概要

- 循环神经网络
 - 实现 RNN 语言模型
 - 截断通过时间反向传播
- 门控循环单元 (GRU)
- 长短期记忆网络 (LSTM)



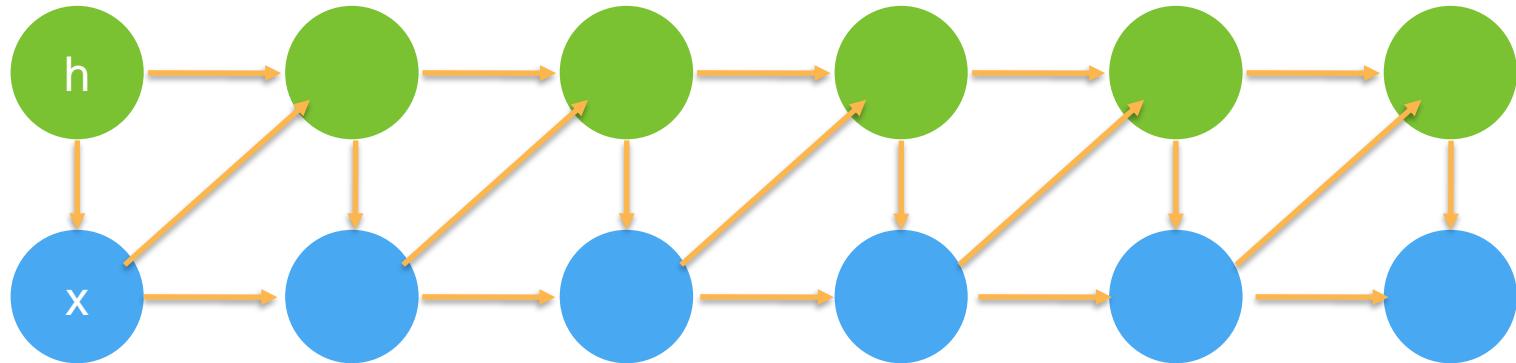
循环神经网络

隐变量模型

- 隐含状态总结了有关过去所有的相关信息

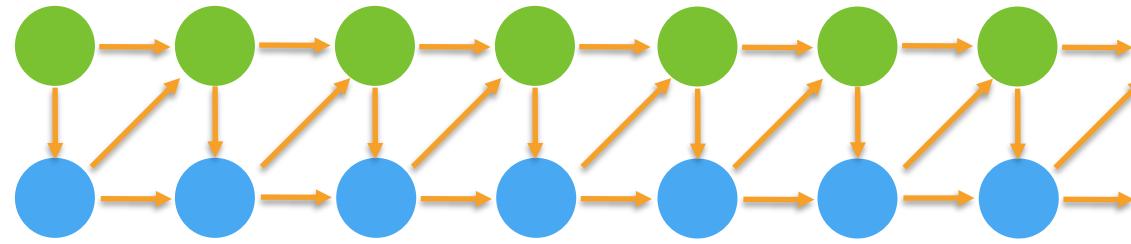
$$h_t = f(x_1, \dots x_{t-1}) = f(h_{t-1}, x_{t-1})$$

$$p(h_t|h_{t-1}, x_{t-1}), \quad p(x_t|h_t, x_{t-1})$$

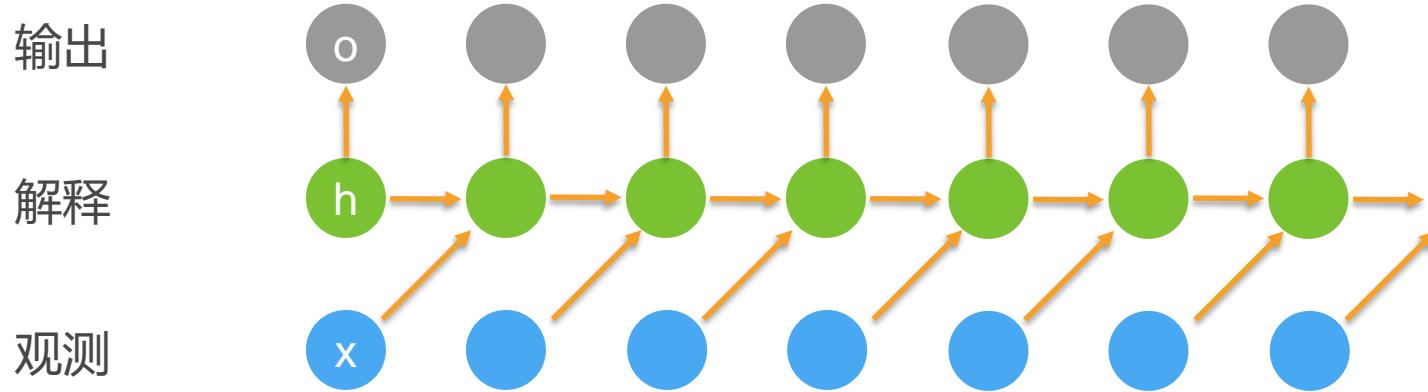


循环神经网络

解释
行动



循环神经网络



- 隐含状态更新

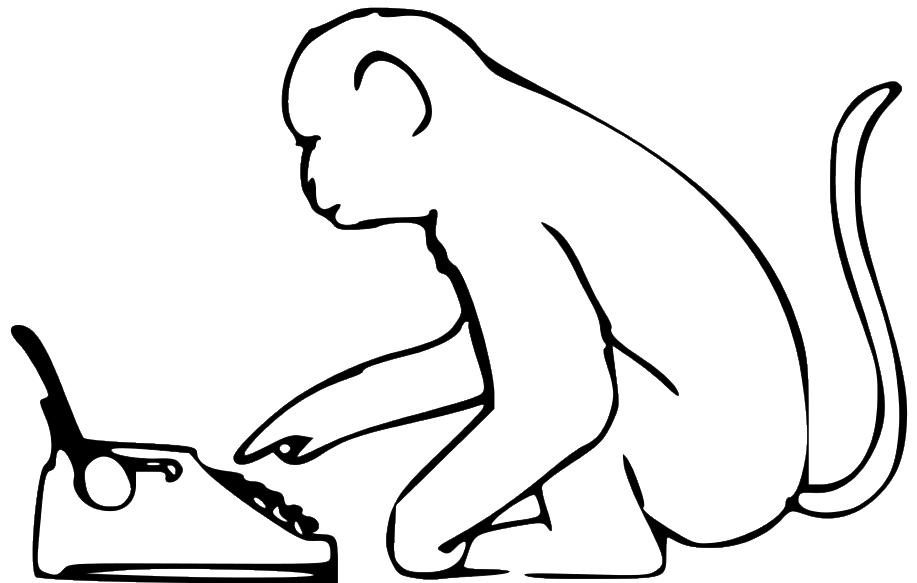
$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_{t-1} + \mathbf{b}_h)$$

- 观察更新

$$\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

代码 ...

实现RNN语言模型



输入编码

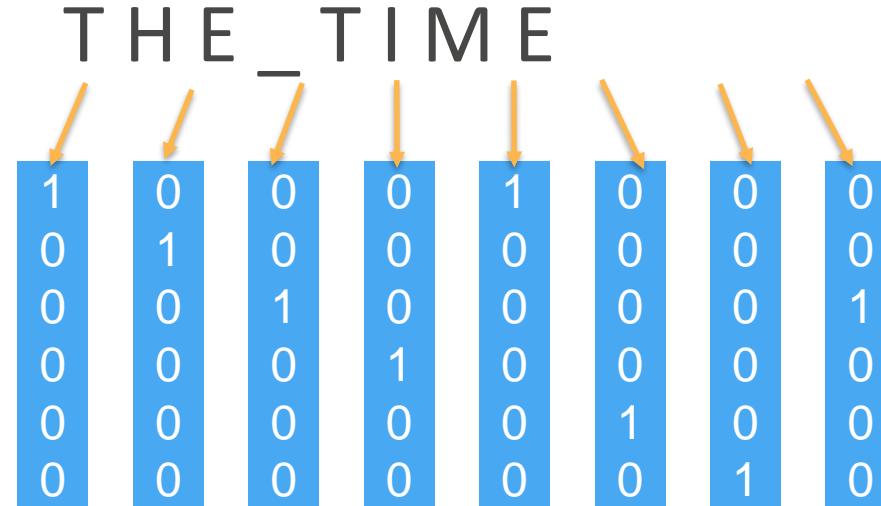
- 需要将输入标记映射到向量
 - 选择颗粒度（单词，字符，音节）
 - 映射到单热门编码的向量

```
nd.one_hot(nd.array([0, 2]), vocab_size)
```

- 再乘以嵌入矩阵 W

输入编码

简洁向量 v



嵌入矩阵 W



嵌入向量 v'

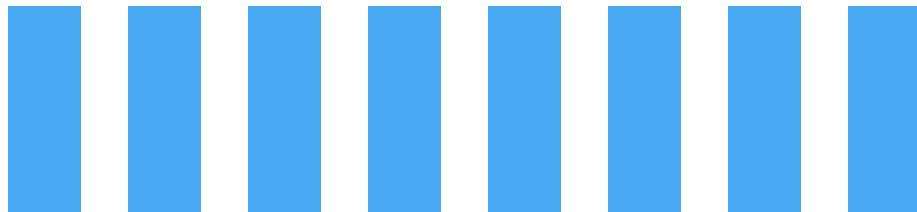


具有隐含状态机制的RNN

- 输入向量序列 $\mathbf{x}_1, \dots, \mathbf{x}_T$
- 隐含状态向量序列
 - $\mathbf{h}_1, \dots, \mathbf{h}_T$
 - $\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$
- 输出向量
 - 序列 $\mathbf{o}_1, \dots, \mathbf{o}_T$; $\mathbf{o}_t = g(\mathbf{h}_t)$
 - 读取序列以生成隐含状态，然后开始生成输出
 - 输出向量通常用作下一个隐含状态的输入

输出编码

输出向量 \mathbf{o}



解码矩阵 W'



$$p(y|\mathbf{o}) \propto \exp(\mathbf{v}_y^\top \mathbf{o}) = \exp(\mathbf{o}[y])$$

单热解码



梯度

- 反向传播的长链依赖关系
 - 需要在内存中保留很多中间值
 - 蝴蝶效应
 - 梯度消失或发散 (稍后会详细介绍)
- 裁剪梯度以防止发散

$$\mathbf{g} \leftarrow \min\left(1, \frac{\theta}{\|\mathbf{g}\|}\right) \mathbf{g}$$

- 重新缩放到最大尺寸为 θ 的梯度

困惑度

- 通常使用对数似然来测量准确度
- 这使得不同长度的输出无法比较
 - (例如，一个坏模型的较短输出的效果可能比一个优秀模型的较长输出的性能具有更好的对数似然)
- 将对数似然标准化为序列长度

$$-\sum_{t=1}^T \log p(y_t | \text{model}) \quad vs. \quad \pi := -\frac{1}{T} \sum_{t=1}^T \log p(y_t | \text{model})$$

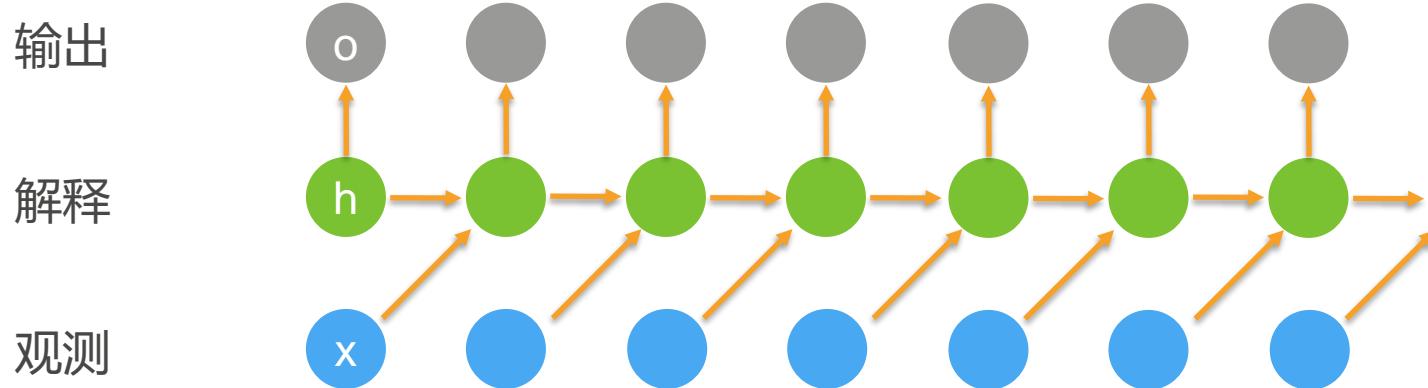
- 困惑度是指数版本 $\exp(\pi)$
(平均有效选择的数量)

代码 ...

A photograph of a large, multi-trunked tree, possibly a willow, against a clear blue sky. The tree has several thick trunks and branches, all covered in dense green foliage. The perspective is from below, looking up at the canopy.

截断通过时间反向传播

循环神经网络



- 隐含状态更新

$$h_t = f(h_{t-1}, x_{t-1}, w)$$

- 观察更新

$$o_t = g(h_t, w)$$

目标函数

- RNN 生成的输出需要与目标标签进行比较

$$L(x, y, w) = \sum_{t=1}^T l(y_t, o_t)$$

- 梯度

$$\partial_w L = \sum_{t=1}^T \partial_w l(y_t, o_t)$$

$$= \sum_{t=1}^T \partial_{o_t} l(y_t, o_t) \left[\partial_w g(h_t, w) + \partial_{h_t} g(h_t, w) \partial_w h_t \right]$$

隐含状态梯度 ${}_{\mathcal{W}} h_t$

- 目标函数

$$\partial_w L = \sum_{t=1}^T \partial_w l(y_t, o_t) = \sum_{t=1}^T \partial_{o_t} l(y_t, o_t) \left[\partial_w g(h_t, w) + \partial_{h_t} g(h_t, w) \partial_w h_t \right]$$

- 梯度递归

$$\partial_w h_t = \partial_w f(x_t, h_{t-1}, w) + \partial_h f(x_t, h_{t-1}, w) \partial_w h_{t-1}$$

$$= \sum_{i=t}^1 \left[\prod_{j=t}^i \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$

隐含状态梯度 $\partial_w h_t$

- 梯度递归

$$\partial_w h_t = \sum_{i=t}^1 \left[\prod_{j=t}^i \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$

太多项

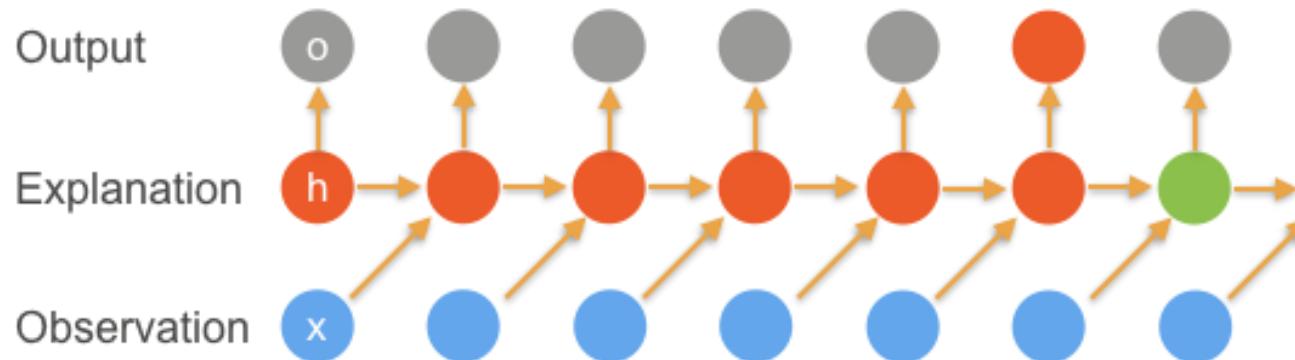
不稳定
(易发散)

开销大

隐含状态梯度 $\partial_w h_t$

- 梯度递归

$$\partial_w h_t = \sum_{i=t}^1 \left[\prod_{j=t}^i \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$

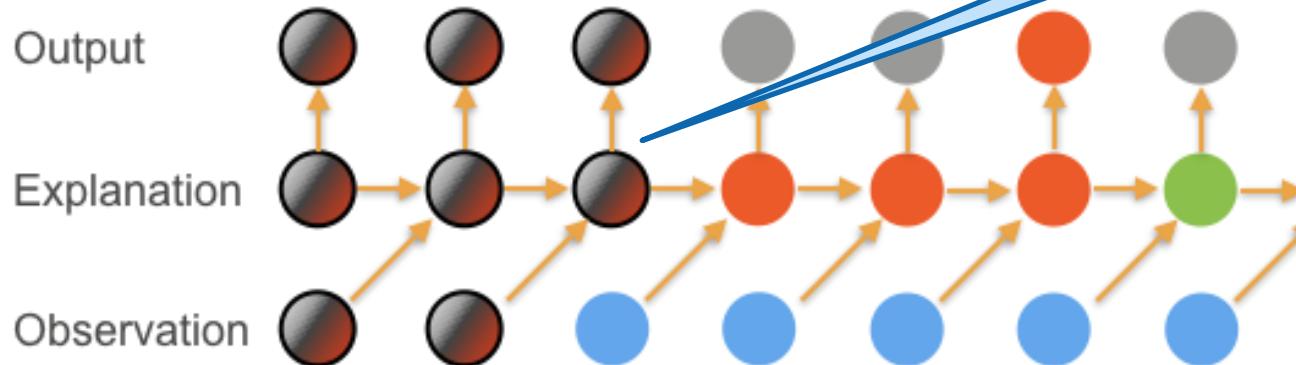


隐含状态梯度 $\partial_w h_t$

- 梯度递归

$$\partial_w h_t = \sum_{i=t}^1 \left[\prod_{j=t}^i \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$

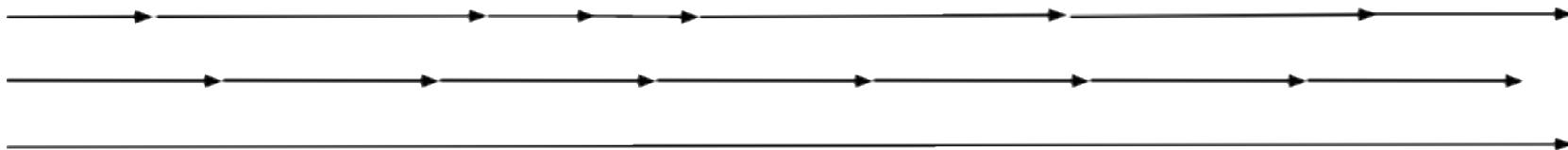
丢弃梯度



截断通过时间反向传播

- 不截断? (天真的策略, 昂贵, 发散)
- 以固定间隔截断 (标准方法, 近似但效果很好)
- 可变长度 (Tallec and Olivier, 2015)
 - 重新加权后可提高近似度, 在实践中效果不佳

The Time Machine by H. G. Wells



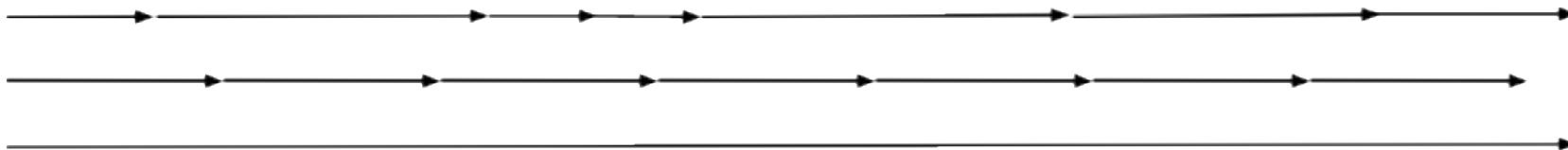
截断通过时间反向传播

- 以固定间隔截断 (标准方法, 近似但效果很好)

$$z_t = \partial_w f(x_t, h_{t-1}, w) + \xi_t \partial_h f(x_t, h_{t-1}, w) \partial_w h_{t-1}$$

- 可变长度 (Tallec and Olivier, 2015)
 - 重新加权后可提高近似度, 在实践中效果不佳

The Time Machine by H. G. Wells



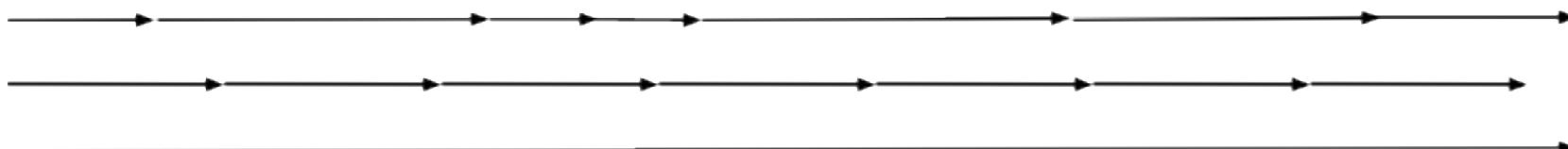
截断通过时间反向传播

- 随机变量而不是简单的截断

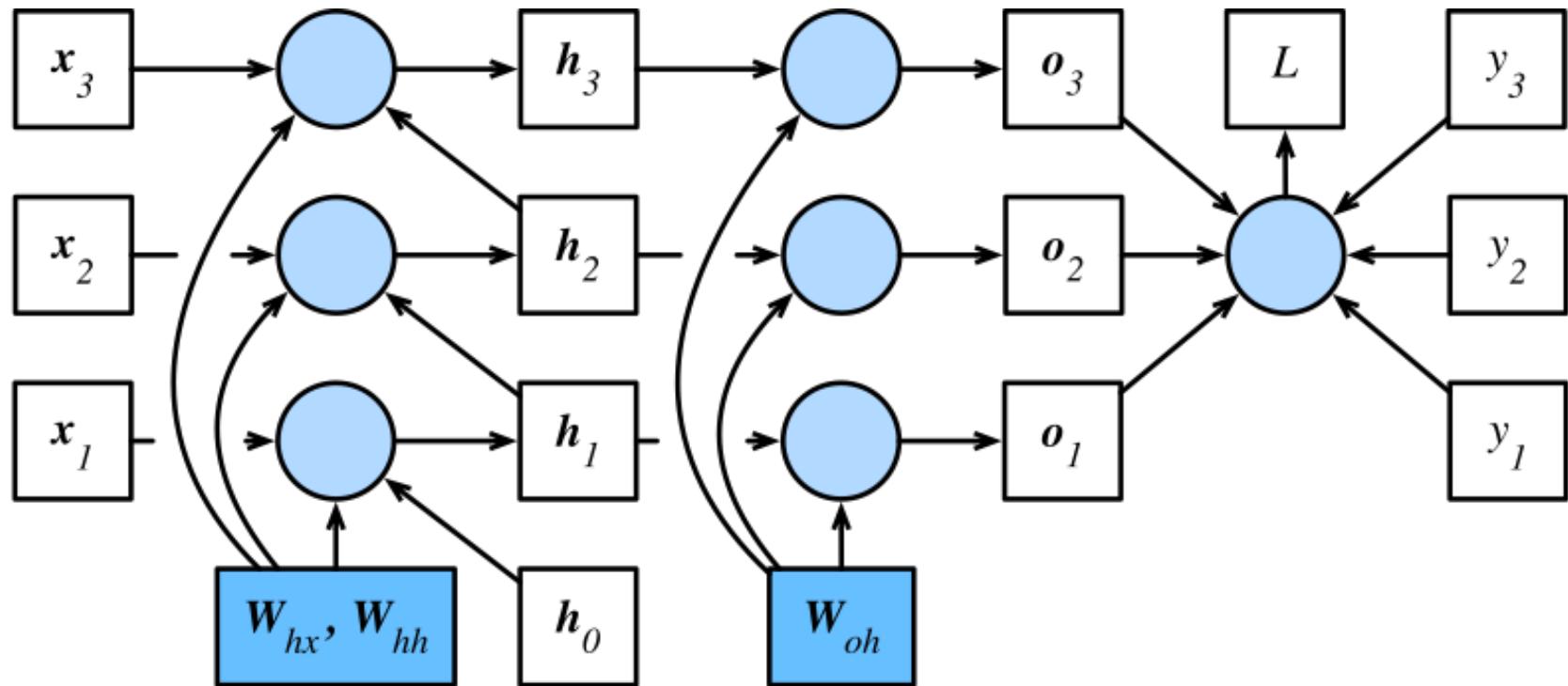
$$z_t = \partial_w f(x_t, h_{t-1}, w) + \xi_t \partial_h f(x_t, h_{t-1}, w) \partial_w h_{t-1}$$

- 变量长度(Tallec and Olivier, 2015)
(和重新加权重之后一样, 实际中并没有改善)

The Time Machine by H. G. Wells



计算图



详细示例

详细示例

- 线性 RNN

$$\mathbf{h}_t = \mathbf{W}_{hx} \mathbf{x}_t + \mathbf{W}_{hh} \mathbf{h}_{t-1} \text{ and } \mathbf{o}_t = \mathbf{W}_{oh} \mathbf{h}_t$$

- 输出梯度

$$\partial_{\mathbf{W}_{oh}} L = \sum_{t=1}^T \text{prod} \left(\partial_{\mathbf{o}_t} l(\mathbf{o}_t, y_t), \mathbf{h}_t \right)$$

- 更新梯度

$$\partial_{\mathbf{W}_{hh}} L = \sum_{t=1}^T \text{prod} \left(\partial_{\mathbf{o}_t} l(\mathbf{o}_t, y_t), \mathbf{W}_{oh}, \partial_{\mathbf{W}_{hh}} \mathbf{h}_t \right)$$

$$\partial_{\mathbf{W}_{hx}} L = \sum_{t=1}^T \text{prod} \left(\partial_{\mathbf{o}_t} l(\mathbf{o}_t, y_t), \mathbf{W}_{oh}, \partial_{\mathbf{W}_{hx}} \mathbf{h}_t \right)$$

详细示例

- 线性 RNN

$$\mathbf{h}_t = \mathbf{W}_{hx} \mathbf{x}_t + \mathbf{W}_{hh} \mathbf{h}_{t-1} \text{ and } \mathbf{o}_t = \mathbf{W}_{oh} \mathbf{h}_t$$

- 更新梯度

$$\partial_{\mathbf{h}_t} \mathbf{h}_{t+1} = \mathbf{W}_{hh}^\top \text{ and thus } \partial_{\mathbf{h}_t} \mathbf{h}_T = (\mathbf{W}_{hh}^\top)^{T-t}$$

- 全部递归

$$\partial_{\mathbf{W}_{hh}} \mathbf{h}_t = \sum_{j=1}^t (\mathbf{W}_{hh}^\top)^{t-j} \mathbf{h}_j$$

$$\partial_{\mathbf{W}_{hx}} \mathbf{h}_t = \sum_{j=1}^t (\mathbf{W}_{hh}^\top)^{t-j} \mathbf{x}_j.$$

丢弃梯度

Truncation in practice

- 计算正向传播
- 计算反向传播只到截断边界 (通常也是小批量边界)
- 在代码中

```
for s in state:
```

```
    s.detach()
```

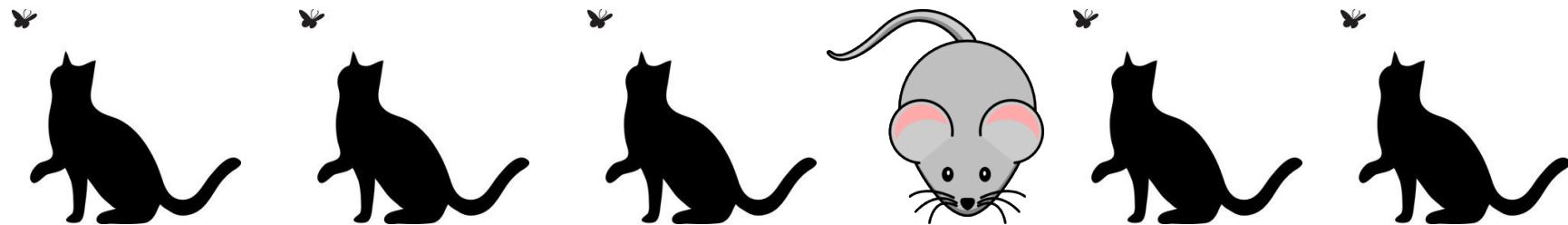
- 顺序采样比随机采样更准确的原因

门控循环单元 (GRU)



在一个序列的注意力

- 并非所有元素都具有同等意义

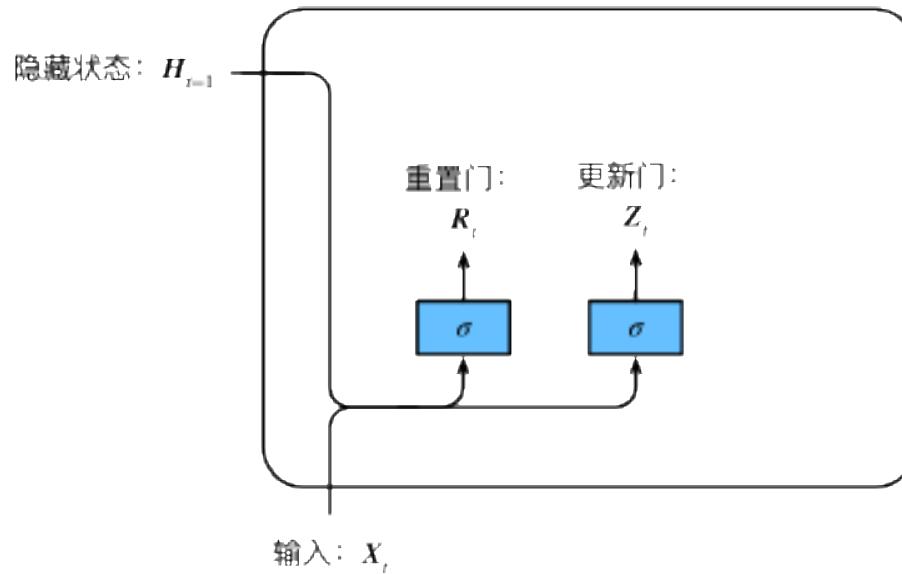


- 只记住相关的元素
 - 需要注意的机制（更新门）
 - 需要忘记的机制（重置门）

门控循环单元

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$

$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$



全连接层和激活函数



按元素运算符



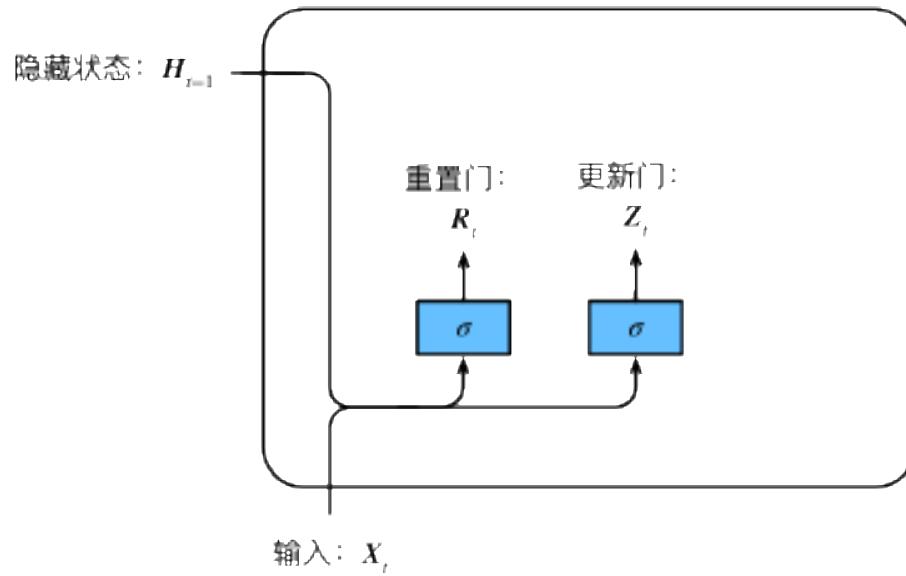
复制



连结

候选隐含状态

$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h)$$



全连接层和激活函数



按元素运算符



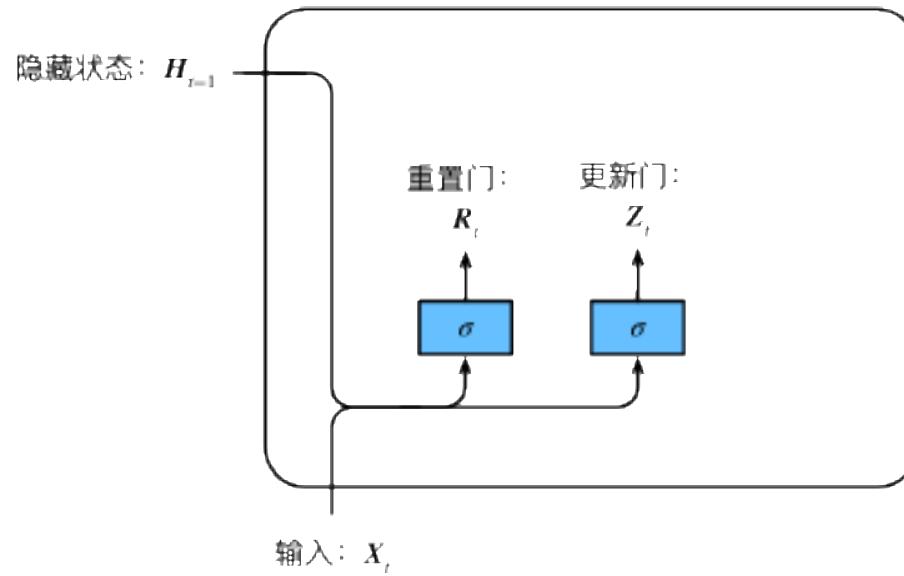
复制



连结

隐含状态

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$



全连接层和激活函数



按元素运算符

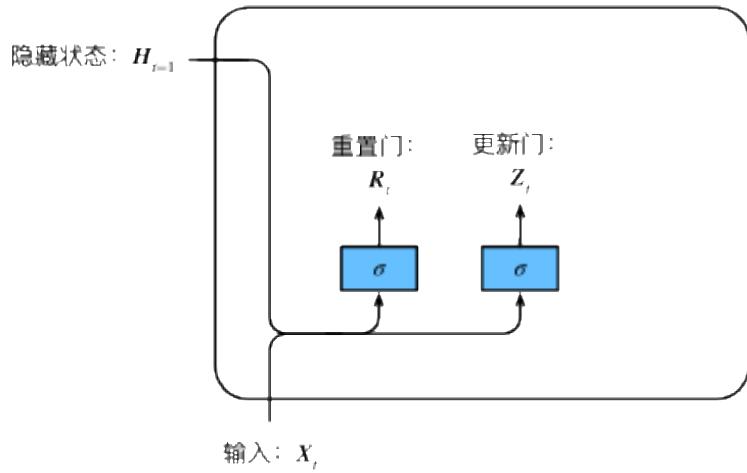


复制



连结

门控循环单元 (GRU) 总结



$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$
$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$
$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h)$$
$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$



全连接层和激活函数



按元素运算符



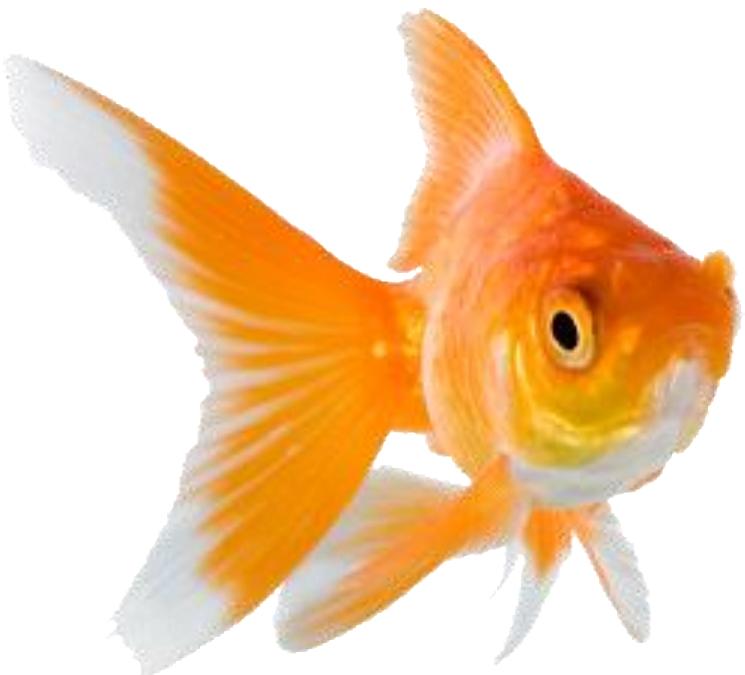
复制



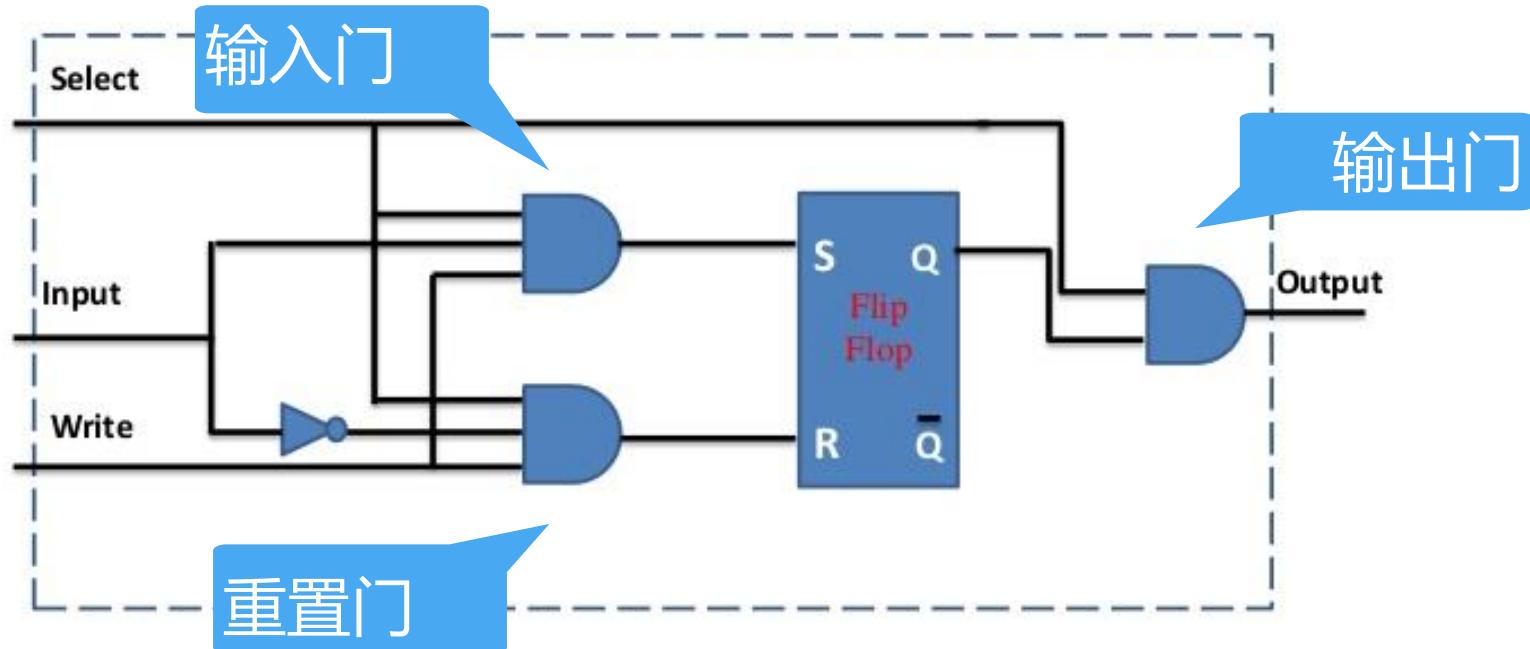
连结

代码 ...

长短期记忆 (LSTM)



电门联想



长短期记忆 (LSTM)

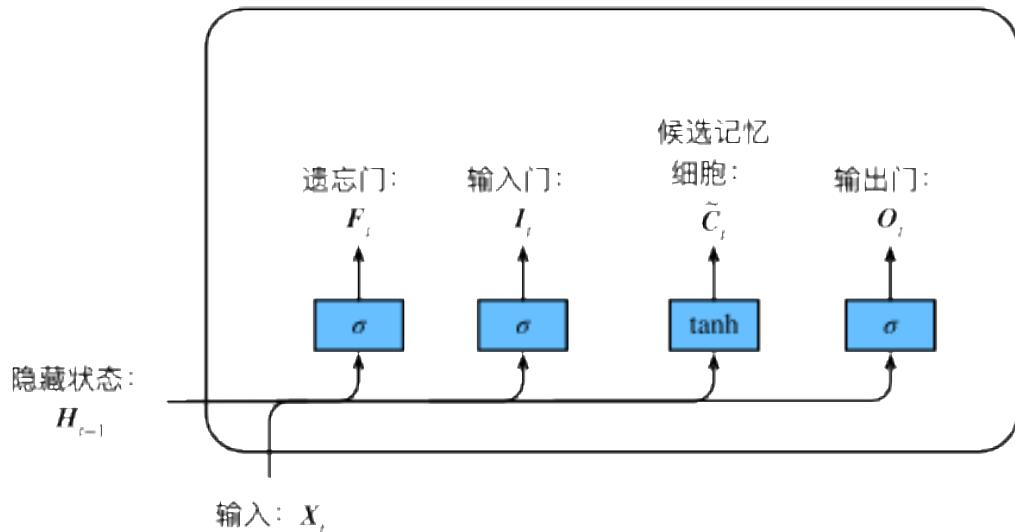
- 遗忘门
 - 将每个值尽可能收缩为零
- 输入门
 - 决定是否应忽略输入数据
- 输出门
 - 决定隐含状态是否用于 LSTM 生成的输出
- 隐含状态
- 隐含状态和记忆细胞

输入门、遗忘门和输出门

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$$



σ

全连接层和激活函数



按元素运算符



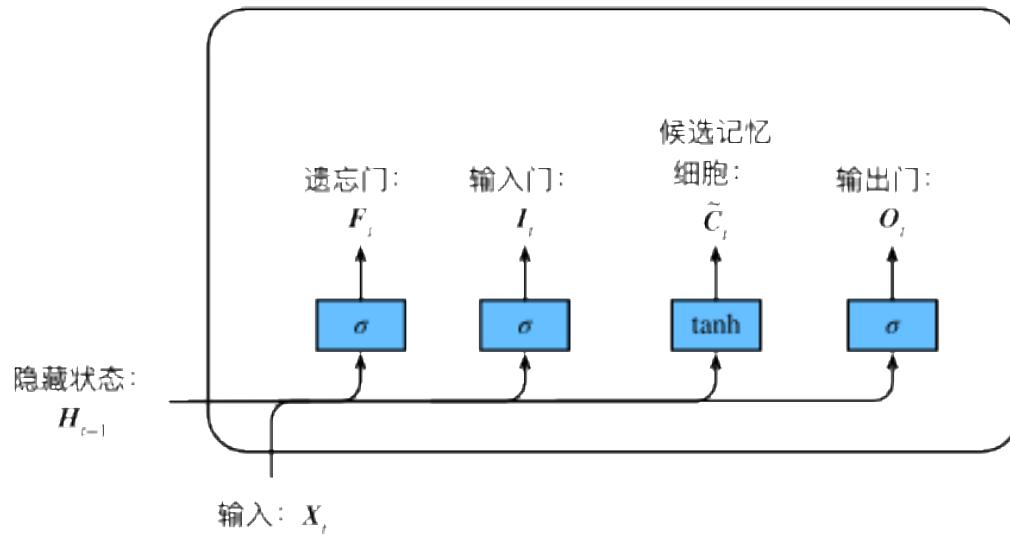
复制



连结

候选记忆细胞

$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$



σ

全连接层和激活函数



按元素运算符



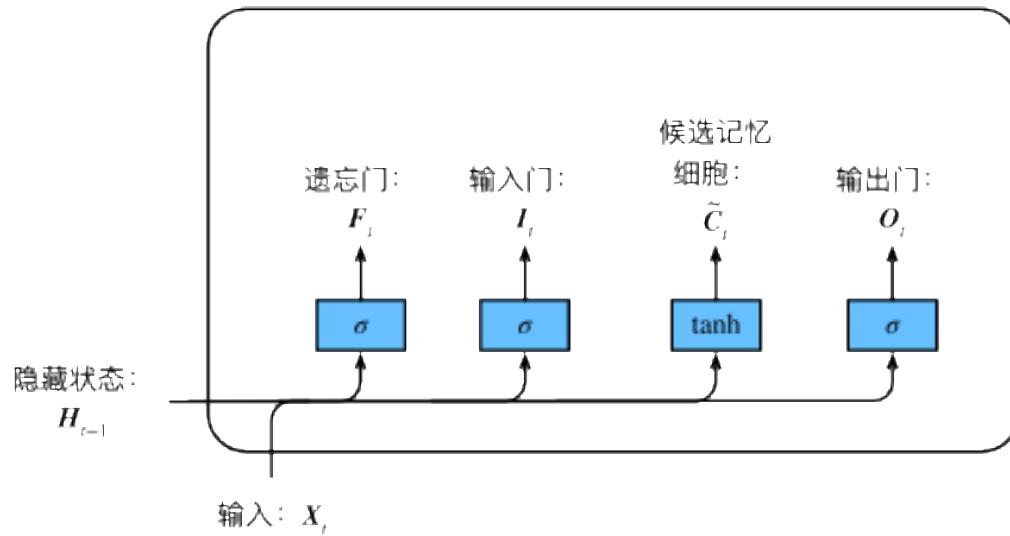
复制



连结

记忆细胞

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$

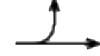


σ

全连接层和激活函数



按元素运算符



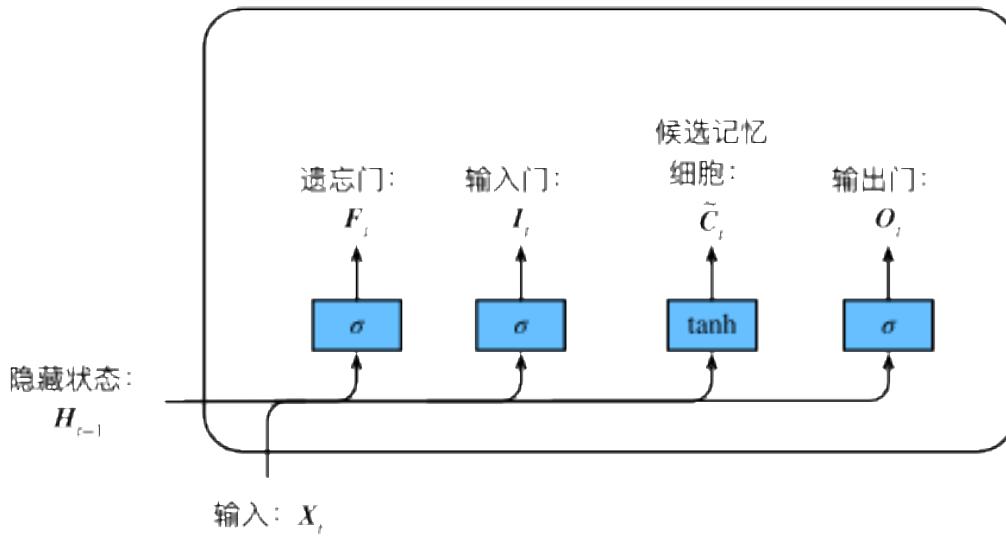
复制



连结

隐含状态

$$H_t = O_t \odot \tanh(C_t)$$



σ

全连接层和激活函数



按元素运算符

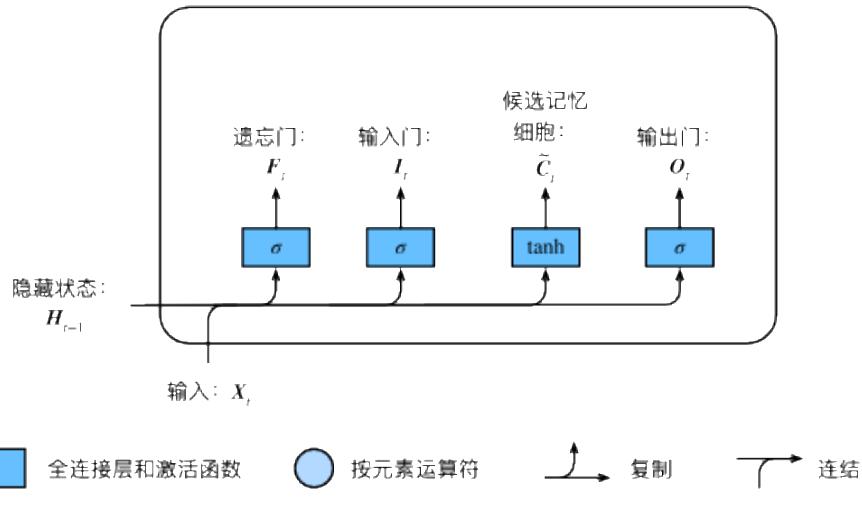


复制



连结

长短期记忆 (LSTM) 总结



$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$$

$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$

$$H_t = O_t \odot \tanh(C_t)$$

代码 ...

总结

- 循环神经网络
 - 实现 RNN 语言模型
 - 截断通过时间反向传播
- 门控循环单元 (GRU)
- 长短期记忆 (LSTM)

动手学深度学习

20. 高级循环神经网络

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/blstm.html>

概要

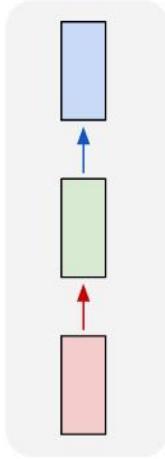
- 深度循环神经网络
- 双向循环神经网络
- 循环神经网络结合
 - 残差网络 (ResNet)
 - 稠密连接网络 (DenseNet)
- 循环神经网络的正则化



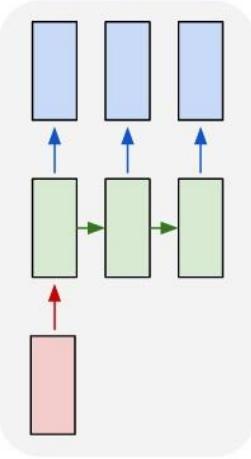
深度循环 神经网络

使用循环神经网络

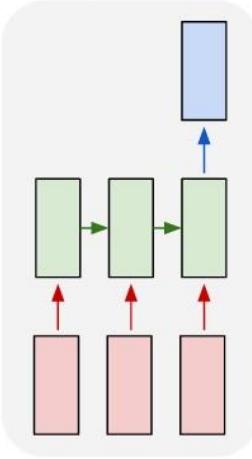
one to one



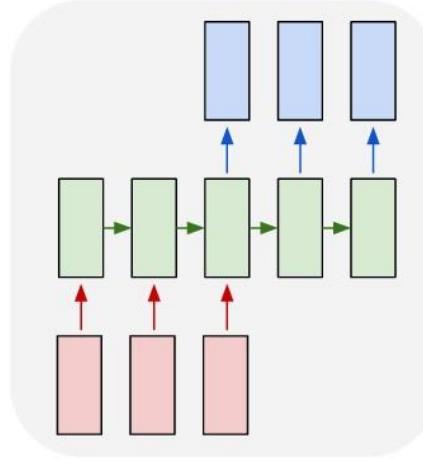
one to many



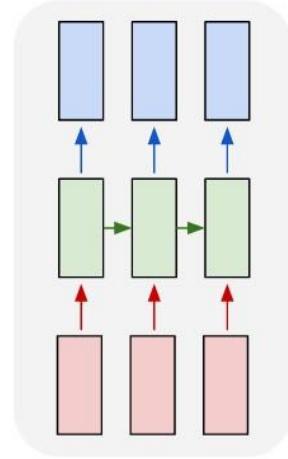
many to one



many to many



many to many

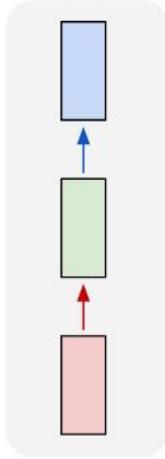


- 编码序列
- 解码序列
- 同时做到这两点

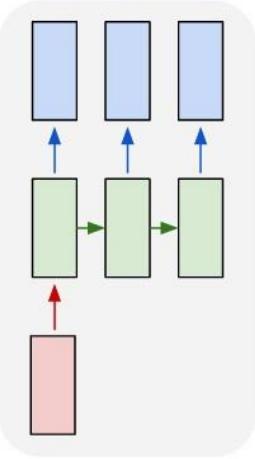
(image courtesy of karpathy.github.io)

使用循环神经网络

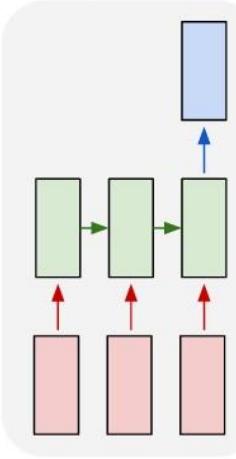
one to one



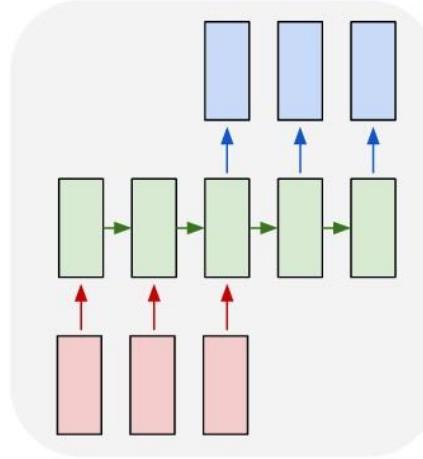
one to many



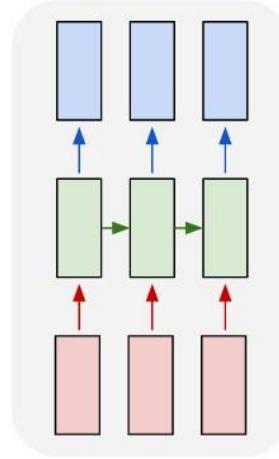
many to one



many to many



many to many



编写诗歌

情感分析

自动问答

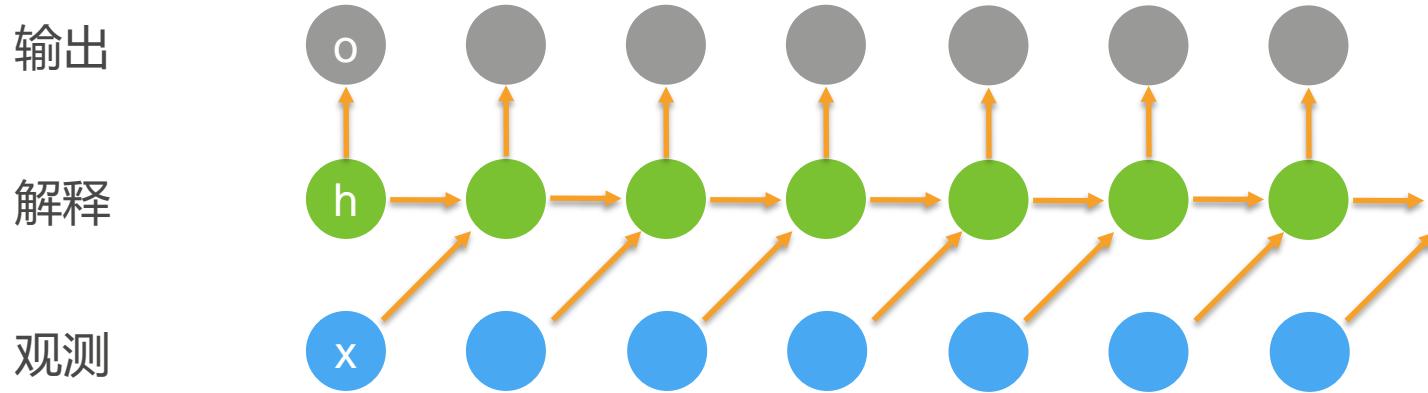
命名实体标记

文本分类

机器翻译

(图片来源: karpathy.github.io)

回顾 - 循环神经网络



- 隐含状态更新

$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_{t-1} + \mathbf{b}_h)$$

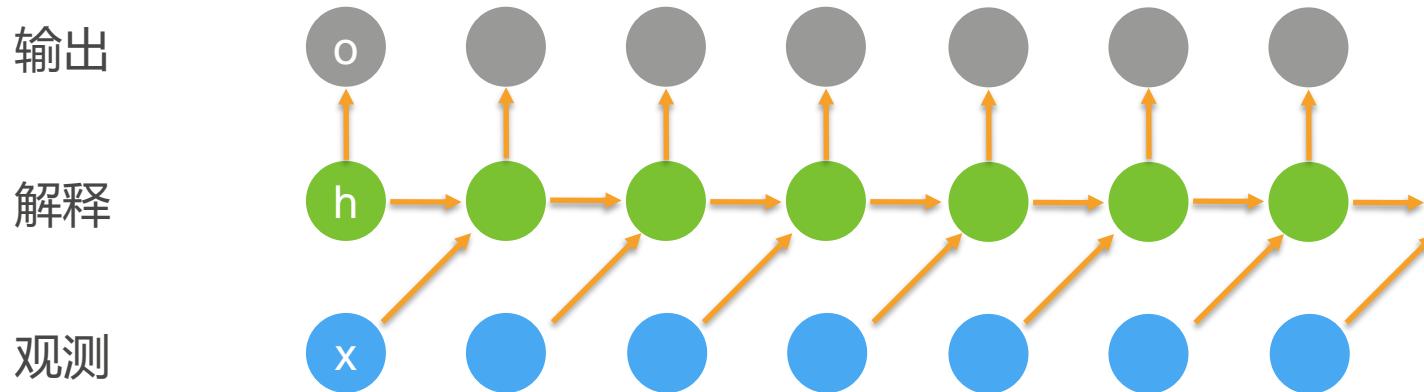
- 观测更新

$$\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

怎么添加
更多非线性的层?

ai

计划 A - 单元的非线性



- 隐含状态更新

$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_{t-1} + \mathbf{b}_h)$$

- 观察更新

$$\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

替代为MLP?

计划 A - 单元的非线性

- 保持潜在空间的结构
- 更复杂的梯度 (非常昂贵)

例如： Zoph et al, 2018

缓慢而昂贵，没有人在实践中使用

- 隐含状态更新

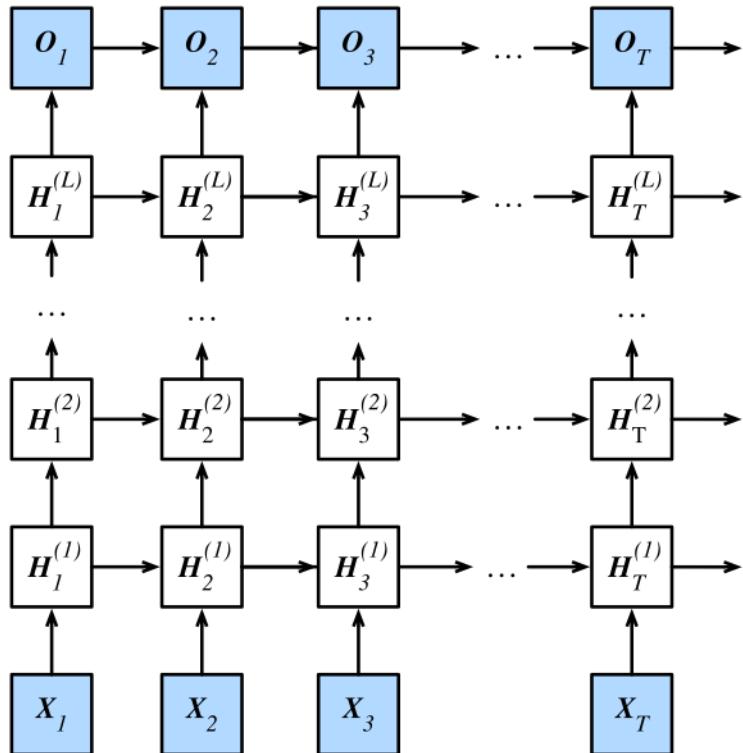
$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_{t-1} + \mathbf{b}_h)$$

- 观察更新

$$\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

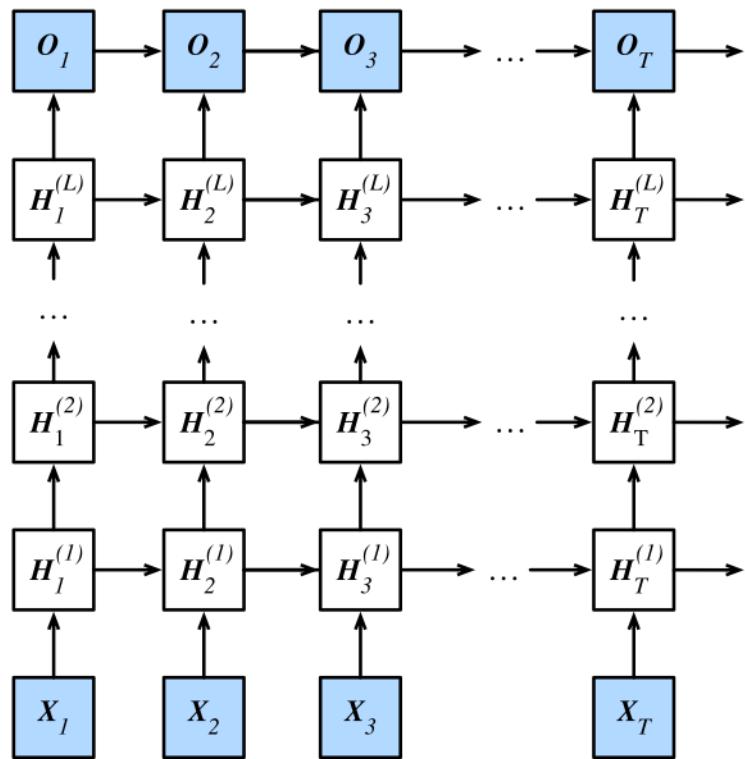
替代为MLP?

计划 B - 深度循环神经网络



- 浅度循环神经网络
 - 输入
 - 隐含层
 - 输出
- 深度循环神经网络
 - 输入
 - 隐含层
 - 隐含层
 - ...
 - 输出

计划 B - 深度循环神经网络



$$\mathbf{H}_t = f(\mathbf{H}_{t-1}, \mathbf{X}_t)$$

$$\mathbf{O}_t = g(\mathbf{H}_t)$$

$$\mathbf{H}_t^1 = f_1(\mathbf{H}_{t-1}^1, \mathbf{X}_t)$$

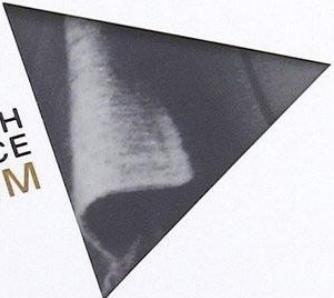
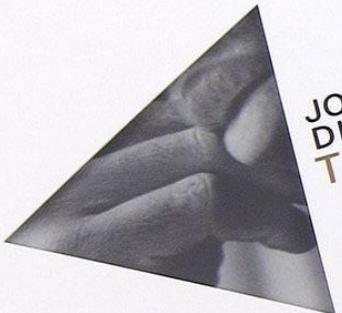
$$\mathbf{H}_t^j = f_j(\mathbf{H}_{t-1}^j, \mathbf{H}_t^{j-1})$$

$$\mathbf{O}_t = g(\mathbf{H}_t^L)$$

代码 ...

双向循环神经网络

JOHN COLTRANE BOTH
DIRECTIONS AT ONCE
THE LOST ALBUM



“未来”的重要性

我_____。

我_____非常饿。

我_____非常饿，我可以吃下一只猪。

“未来”的重要性

我 **饿**。

我 **不是** 非常饿。

我 **非常** 非常饿，我可以吃下一只猪。

“未来”的重要性

我 **饿**。

我 **不是** 非常饿。

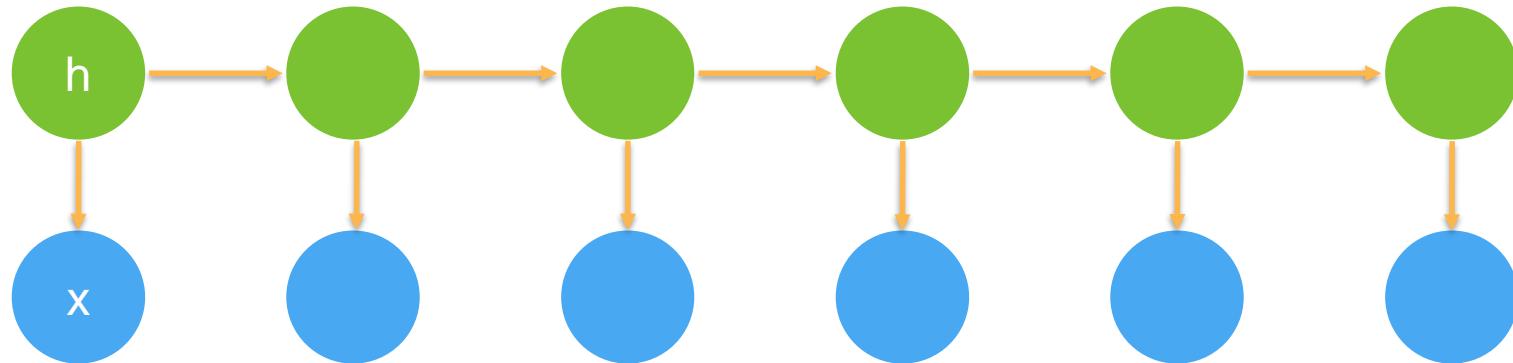
我 **非常** 非常饿，我可以吃下一只猪。

- 填写完全不同的单词，取决于单词的上下文
- 到目前为止，RNN 只关注上文
- 也可以使用**下文**来填写

回顾 - 图模型

- 隐马尔可夫模型 (Hidden Markov Model)

$$p(h_t|h_{t-1}, x_{t-1}) \text{ 和 } p(x_t|h_t, x_{t-1})$$



- 可以联合建模并通过动态编程求解

动态编程

- 联合概率

$$p(x, h) = p(h_1)p(x_1|h_1)\prod_{t=2}^T p(h_t|h_{t-1})p(x_t|h_t)$$

动态编程

$$\begin{aligned} p(x) &= \sum_h p(h_1)p(x_1 | h_1) \prod_{i=2}^T p(h_i | h_{i-1})p(x_i | h_i) \\ &= \sum_{h_2, \dots, h_T} \underbrace{\left[\sum_{h_1} p(h_1)p(x_1 | h_1)p(h_2 | h_1) \right]}_{=: \pi_2(h_2)} p(x_2 | h_2) \prod_{i=2}^T p(h_i | h_{i-1})p(x_i | h_i) \\ &= \sum_{h_3, \dots, h_T} \underbrace{\left[\sum_{h_2} \pi_2(h_2)p(x_2 | h_2)p(h_3 | h_2) \right]}_{=: \pi_3(h_3)} p(x_3 | h_3) \prod_{i=3}^T p(h_i | h_{i-1})p(x_i | h_i) \end{aligned}$$

动态编程

- 联合概率

$$p(x, h) = p(h_1)p(x_1|h_1)\prod_{t=2}^T p(h_t|h_{t-1})p(x_t|h_t)$$

- 正向相乘

$$\pi_{t+1}(h_{t+1}) = \sum_{h_t} \pi_t(h_t)p(x_t|h_t)p(h_{t+1}|h_t)$$

动态编程

$$\begin{aligned} p(x) &= \sum_h \prod_{i=1}^{T-1} p(h_t | h_{t-1}) p(x_t | h_t) \cdot p(h_T | h_{T-1}) p(x_T | h_T) \\ &= \sum_{h_1, \dots, h_{T-1}} \prod_{i=1}^{T-1} p(h_t | h_{t-1}) p(x_t | h_t) \cdot \underbrace{\left[\sum_{h_T} p(h_T | h_{T-1}) p(x_T | h_T) \right]}_{=: \rho_{T-1}(h_{T-1})} \\ &= \sum_{h_1, \dots, h_{T-2}} \prod_{i=1}^{T-2} p(h_t | h_{t-1}) p(x_t | h_t) \cdot \underbrace{\left[\sum_{h_{T-1}} p(h_{T-1} | h_{T-2}) p(x_{T-1} | h_{T-1}) \right]}_{=: \rho_{T-2}(h_{T-2})} \end{aligned}$$

动态编程

- 联合概率

$$p(x, h) = p(h_1)p(x_1|h_1)\prod_{t=2}^T p(h_t|h_{t-1})p(x_t|h_t)$$

- 正向相乘

$$\pi_{t+1}(h_{t+1}) = \sum_{h_t} \pi_t(h_t)p(x_t|h_t)p(h_{t+1}|h_t)$$

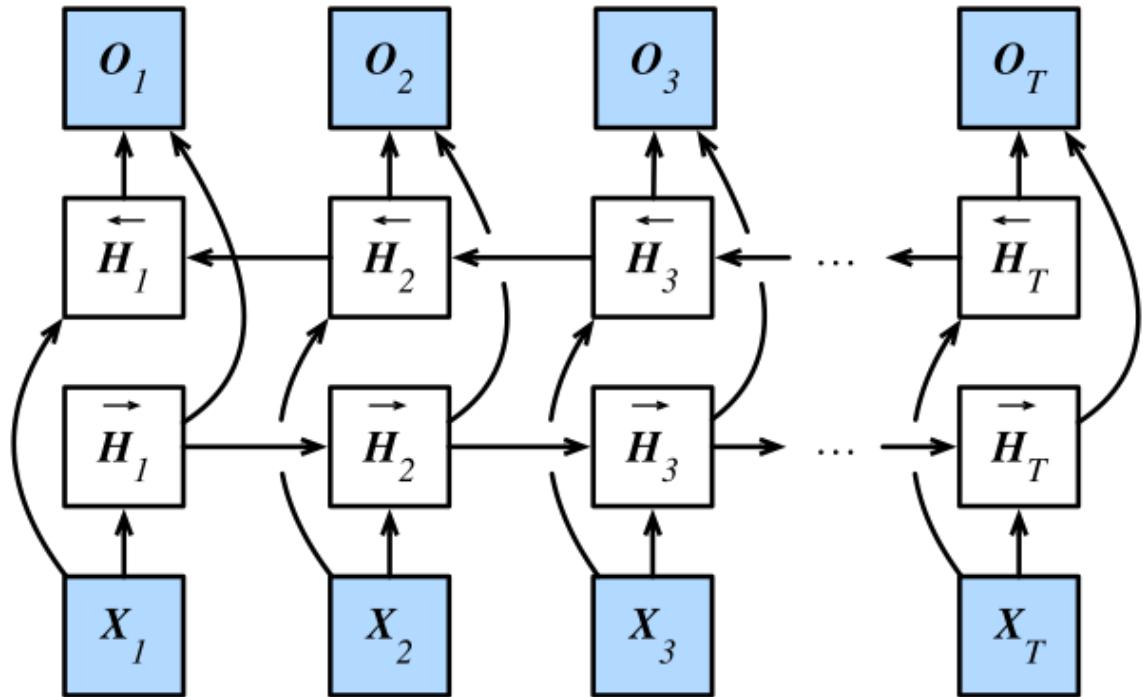
- 反向相乘

$$p(x_j|x_{-j}) \propto \sum_{h_j} \pi_j(h_j)\rho_j(h_j)p(x_j|h_j)$$

I WANT
应用到 RNN 上？

IT ALL

双向循环神经网络



- 一个 RNN 正向传递
- 另一个 RNN 反向传递
- 将两种隐含状态组合在一起以生成输出

```
epoch 600, perplexity 1.016867, time 0.15 sec
- travellerer cumplhp peougunininininin suppepepepepepepepe
- time travellererer fuf this shanatatatatatatatatatatatatatata
epoch 800, perplexity 1.007069, time 0.15 sec
- traveller hime of copspepepep smefsffff'::::::::::::::::::
- time travellerer prefififididididididididididididididididi
epoch 1000, perplexity 1.001932, time 0.15 sec
- travellererererererererererererererererererererererererer
- time travellererererererererererererererererererererererer
```

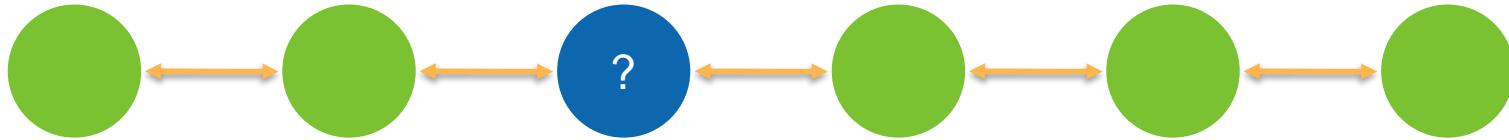
这不适用于序列生成...

```
epoch 600, perplexity 1.016867, time 0.15 sec
- travellerer cumplhp peougunininininin suppepepepepepepepe
- time travellererer fuf this shanatatatatatatatatatatatatatata
epoch 800, perplexity 1.007069, time 0.15 sec
- traveller hime of copspepepep smefsffff'::::::::::::::::::
- time travellerer prefififididididididididididididididididi
epoch 1000, perplexity 1.001932, time 0.15 sec
- travellererererererererererererererererererererererererer
- time travellererererererererererererererererererererererer
```

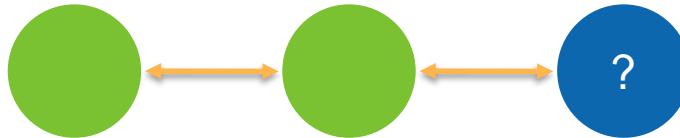
为什么呢？

原因

- 训练阶段



- 测试阶段

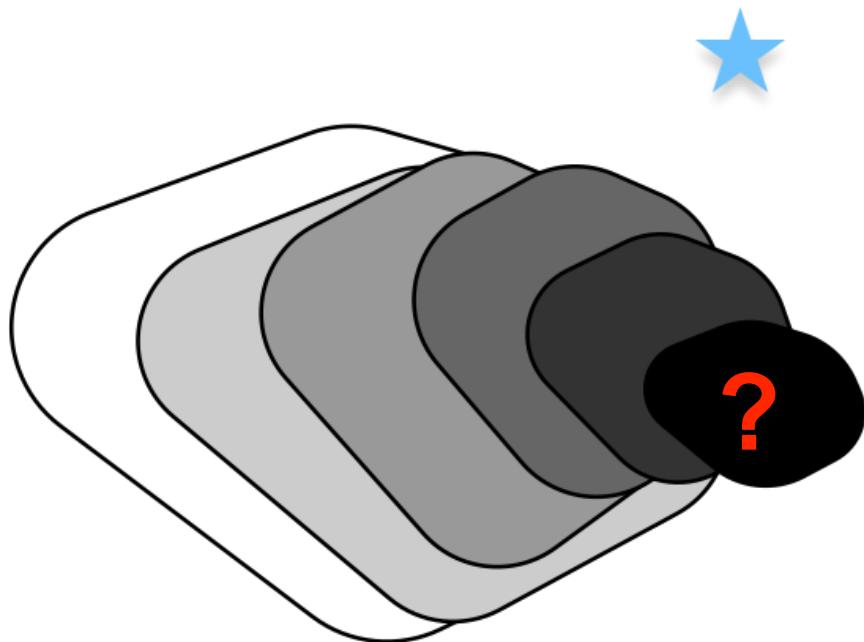


- 仍然可以使用它来编码序列

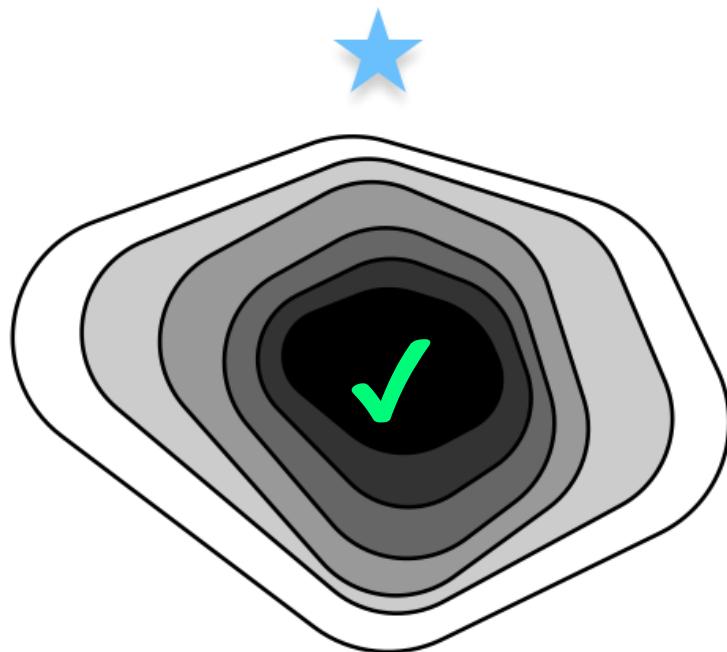


循环神经网络 的残差网络

回顾 - 添加层是否可以提高准确性?



generic function classes

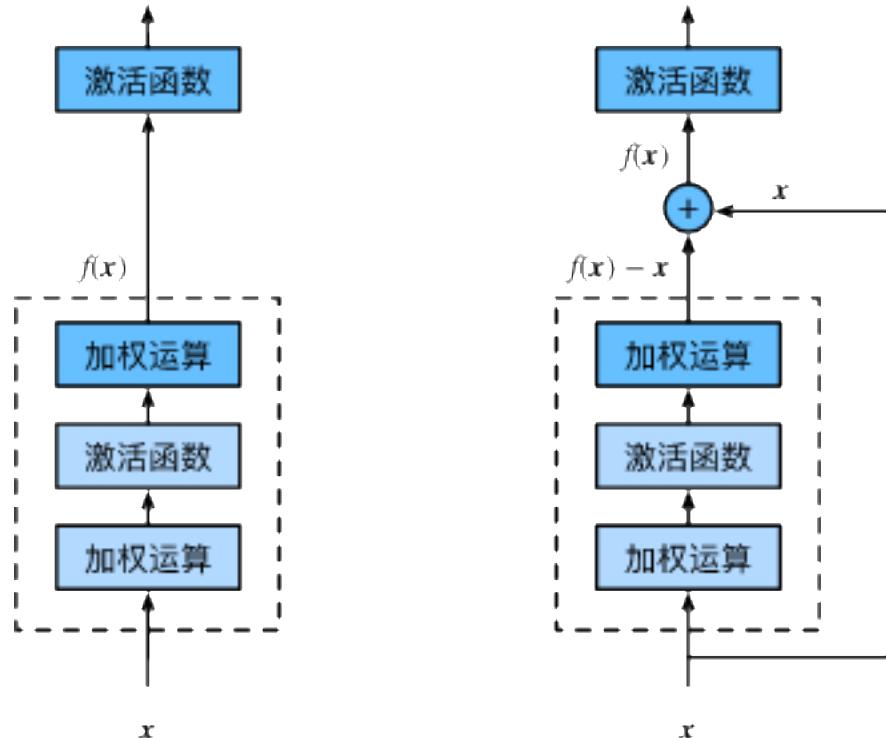


nested function classes

回顾 - 残差网络 (ResNet)

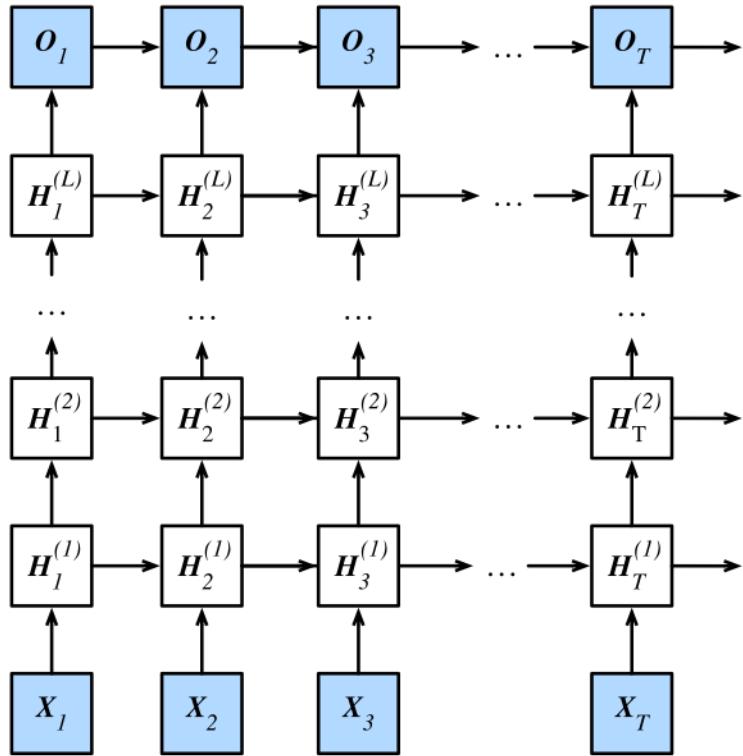
- 添加图层会更改功能类
- 我们想要添加到函数类中
- '泰勒扩展'风格参数化

$$f(x) = x + g(x)$$



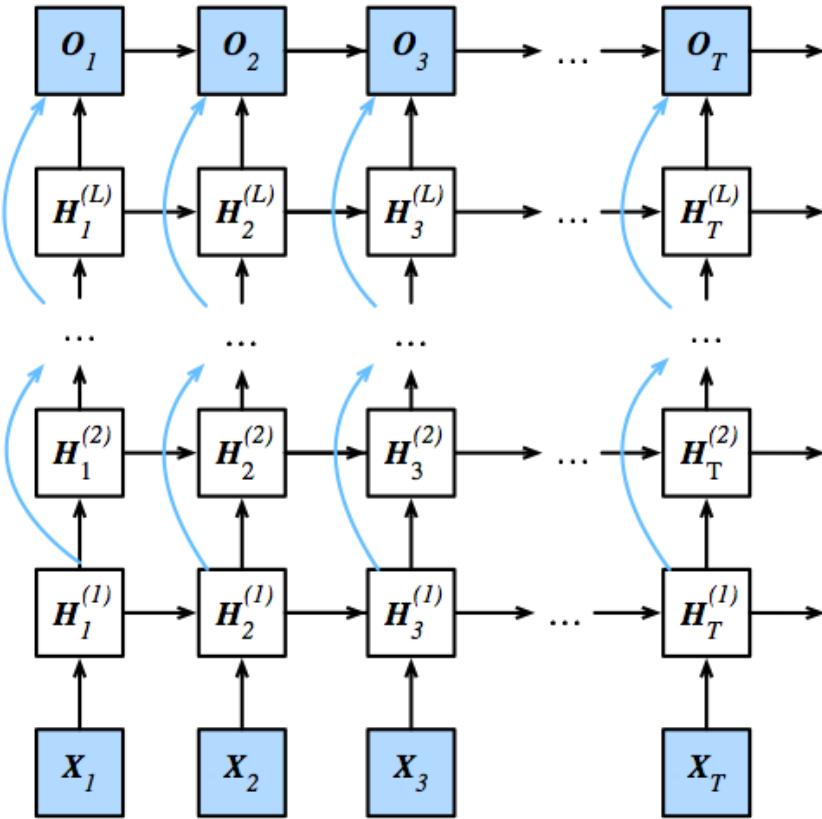
He et al., 2015

回顾 – 深度循环神经网络



- 深度循环神经网络
 - 输入
 - 隐含层
 - 隐含层
 - ...
 - 输出

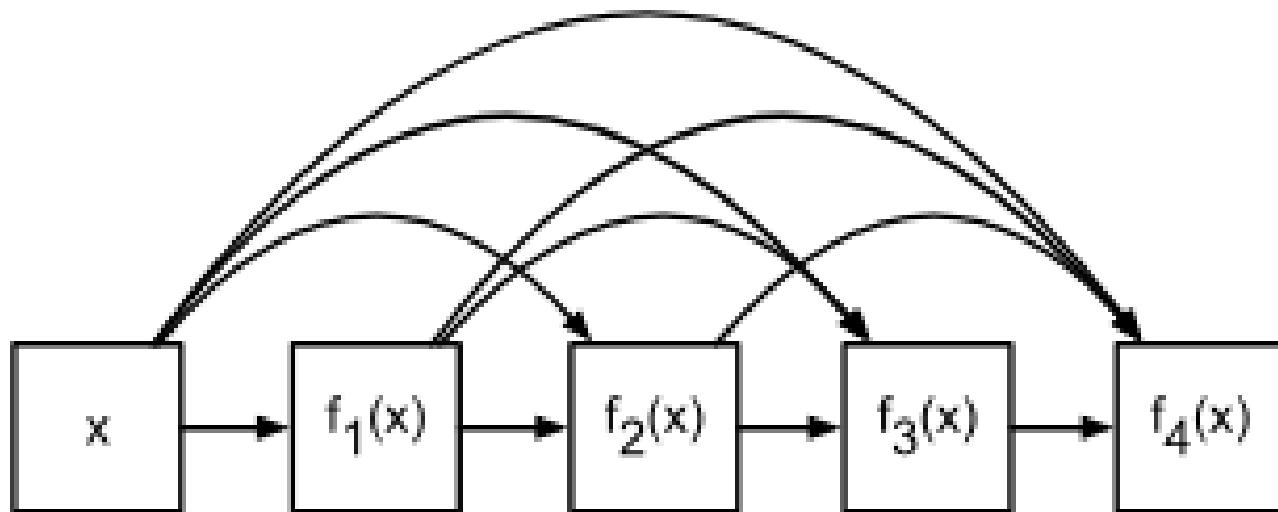
残差网络 (ResNet)



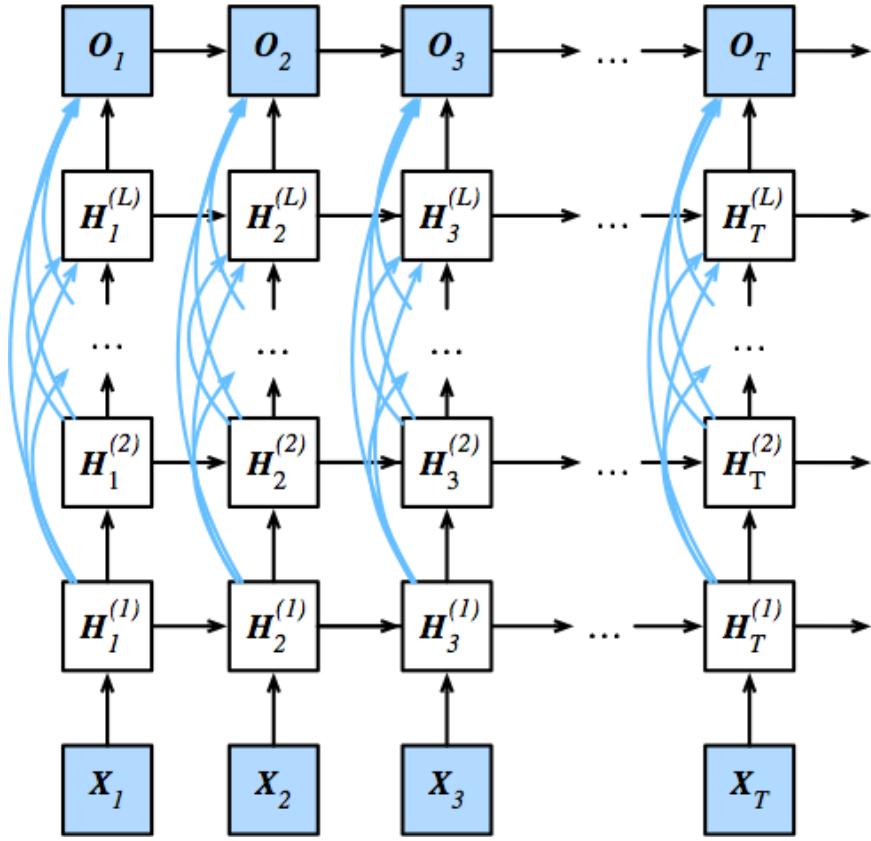
$$\bar{H}_t^{(2i)} = H_t^{(2i)} + H_t^{(2i)-1}$$

- 每个第二层的输入也被添加到其输出 (残余连接)
- 变种
 - 简单的添加
 - 添加前的非线性
 - 也可以连接

循环神经网络 的稠密连接网络



循环神经网络的稠密连接网络 (DenseNet)



$$\bar{\mathbf{H}}_t^{(t)} = [\mathbf{H}_t^{(t)}, \bar{\mathbf{H}}_t^{t-1}]$$

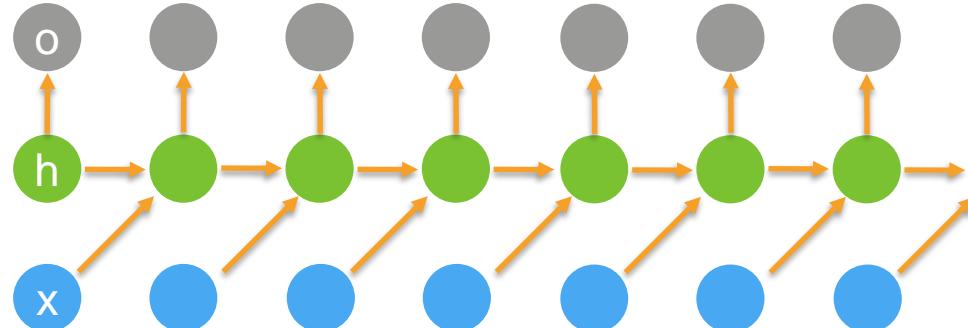
- 将前一层的输出连接为下一层的输入
- 偶尔添加过渡层以减少维度

循环神经网络的正则化



过拟合

- RNN 像任何其他模型一样也会过拟合
- 依赖序列更难以控制
 - 可以控制序列深度，例如：丢弃法
 - 对于顺序部分，巧妙处理变量输入：
 - 例如，可能会跳过某些输入
- 如果我们使用丢弃法，我们可能会错过坐标中的相关信息



回顾 – 丢弃法

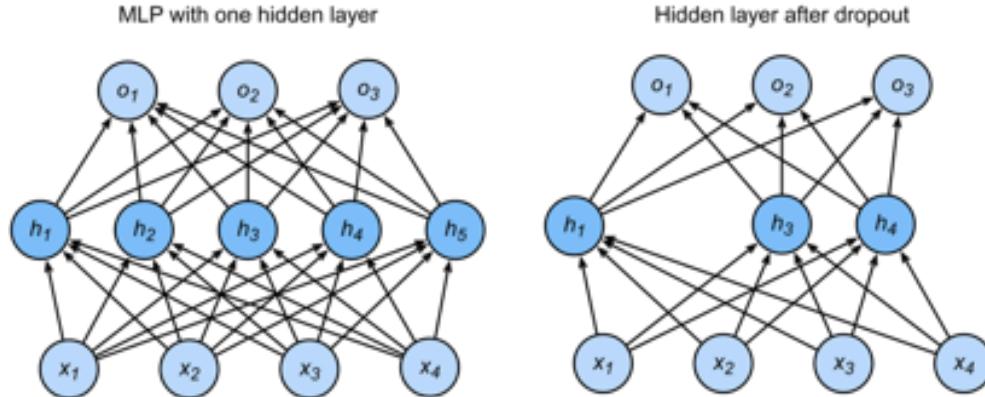
- 经常在隐含层的输出上应用丢弃法

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

$$\mathbf{o} = \mathbf{W}_2 \mathbf{h}' + \mathbf{b}_2$$

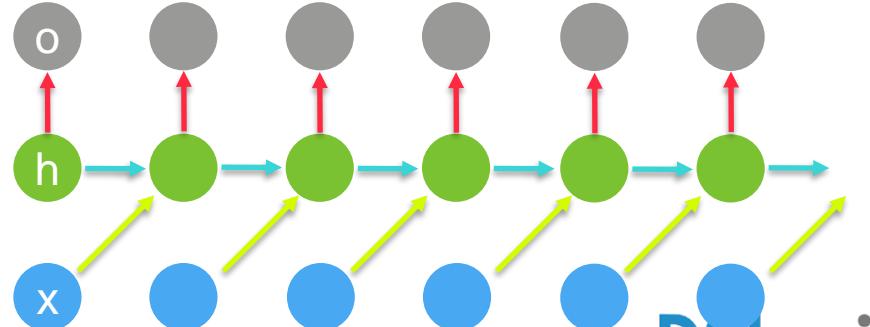
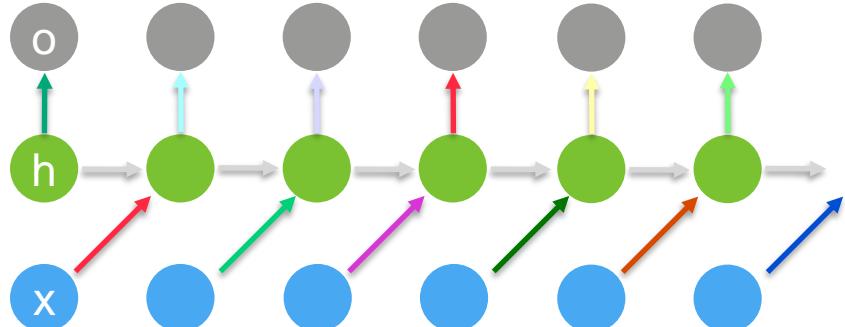
$$\mathbf{y} = \text{softmax}(\mathbf{o})$$



- 在推理时，不使用丢失法，即 $\mathbf{h}' = \text{dropout}(\mathbf{h})$

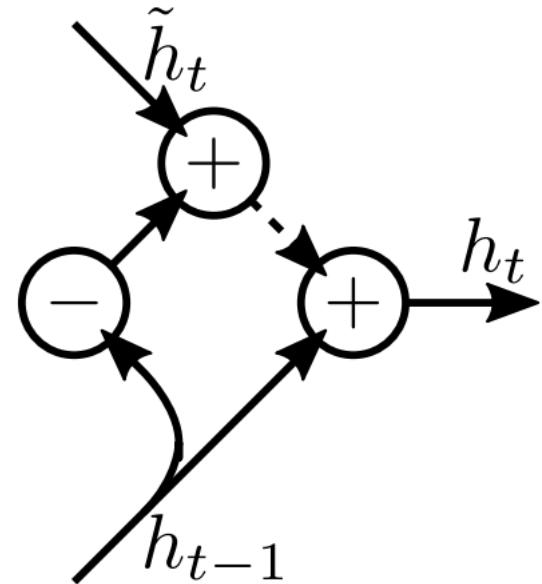
变分丢弃法 (Gal & Ghahramani, 2015)

- 普通丢弃法
 - 每时间段应用
 - 每时间段不同的掩码
- 变分丢弃法
 - 证明在任何神经网络应用丢弃法等同于近似贝叶斯推断的形式
 - 每时间段相同的掩码



Zoneout (Krueger et al., 2016)

- 稳定性抵制在一序列中跳过观察
- 状态稳定性与隐含状态更新有关
- 跳过隐含状态更新 - 训练期间保持前状态
 - $h_t = h_{t-1}$



更多技巧

- 参数均一化 (Merity et al., 2017)
在训练 RNN 每次迭代时应用均一 SGD
- 随机权重均一化 (Wilson et al., 2018)
与上个类似，但同时变化学习率
- “兄弟”丢弃法 (Zolna et al., 2017)
应用丢弃法的同时减少变分，可提高参数稳定性

...

动手学深度学习

22. 嵌入向量, 词嵌入, 子词嵌入, 全局向量的词嵌入

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/word2vec.html>

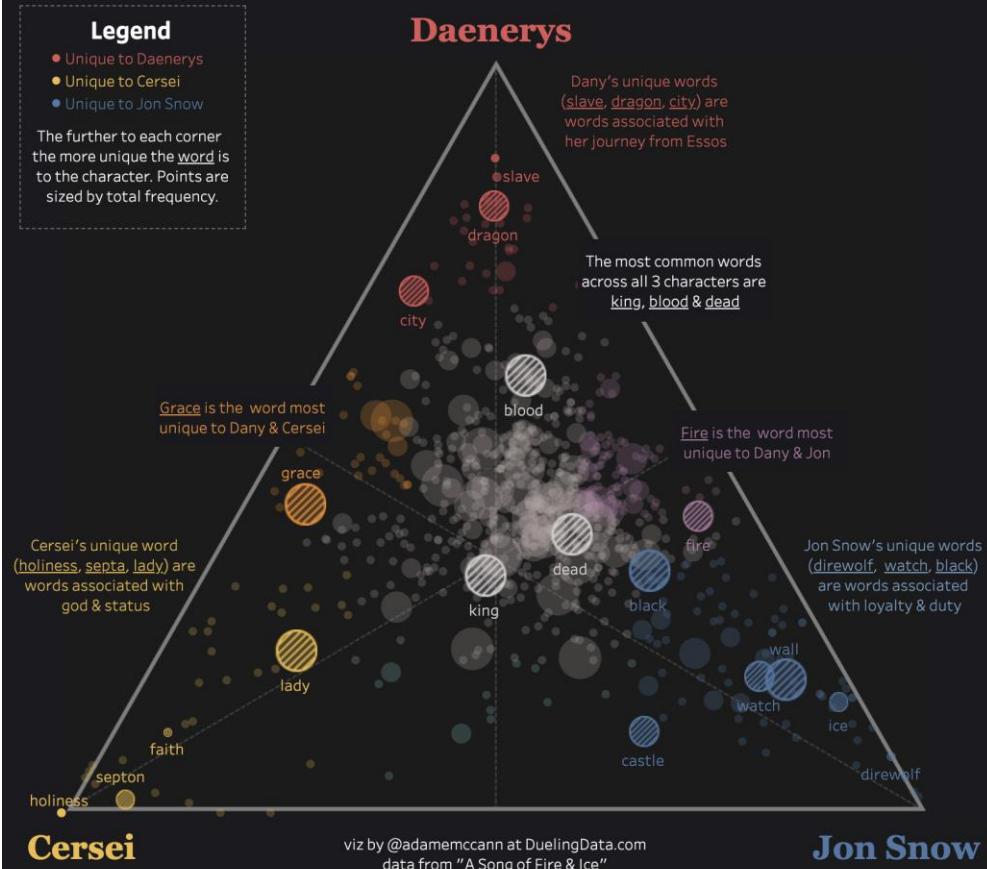
概要

- 嵌入向量 (Embeddings)
- 词嵌入 (Word2vec)
 - Skip-Gram
 - CBOW
- 子词嵌入 (fastText)
- 全局向量的词嵌入 (GloVe)

词嵌入 (Word2vec)

GAME OF THRONES™ IN WORDS

This viz shows the most unique words by character for each chapter in the 5 Game of Thrones books



动机

- 单热向量法将目标对象/单词映射到固定长度向量
- 这些向量仅包含身份信息，而不包含语义含义，例如：

$$\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{z}, \mathbf{y} \rangle = 0$$

	x	y	z
	1	0	0
	0	1	0
:	:	:	:
	0	0	1

词嵌入 (Word2vec)

- 学习每个单词的嵌入向量
- 用于 $\langle x, y \rangle$ 衡量相似性

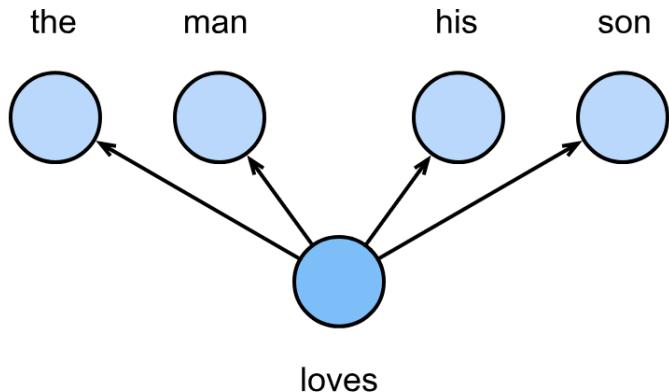
$$\langle x, y \rangle > \langle z, y \rangle$$

- 建立概率模型
- 最大化似然函数优化

	x	y	z
	1	0	0
	0	1	0
:	:	:	:
	0	0	1

Skip-Gram 模型

- 一个单词可用于生成它周围的单词
- 给定中心词，每个上下文词是独立生成的



$$\begin{aligned} & \mathbb{P}(\text{"the"}, \text{"man"}, \text{"his"}, \text{"son"} \mid \text{"loves"}) \\ &= \mathbb{P}(\text{"the"} \mid \text{"loves"}) \cdot \mathbb{P}(\text{"man"} \mid \text{"loves"}) \\ &\quad \cdot \mathbb{P}(\text{"his"} \mid \text{"loves"}) \cdot \mathbb{P}(\text{"son"} \mid \text{"loves"}) \end{aligned}$$

似然函数

全部概率求和非常昂贵

词	嵌入
中心词	w_c $\mathbf{v}_c \in \mathbb{R}^d$
上下文	w_o $\mathbf{u}_o \in \mathbb{R}^d$

$$\mathbb{P}(w_o \mid w_c) = \frac{\exp(\mathbf{u}_o^\top \mathbf{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)}$$

\mathcal{V} : 所有上下文

- 给定长度为 T 的序列，上下文窗口长度为 m ，似然函数：

$$\prod_{t=1}^T \prod_{-m \leq j \leq m, j \neq 0} \mathbb{P}(w^{(t+j)} \mid w^{(t)})$$

负采样

- 将中心词和上下文词同时出现在相同窗口中作为一个“事件”

$$\mathbb{P}(D = 1 | w_c, w_o) = \sigma(\mathbf{u}_c^T \mathbf{v}_o)$$

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

- 将似然函数从 $\prod_{t=1-m}^T \prod_{j \leq m, j \neq 0} \mathbb{P}(w^{(t+j)} | w^{(t)})$ 更改为

$$\prod_{t=1-m}^T \prod_{j \leq m, j \neq 0} \mathbb{P}(D = 1 | w^{(t)}, w^{(t+j)})$$

最简单的解决方案：无限

负抽样

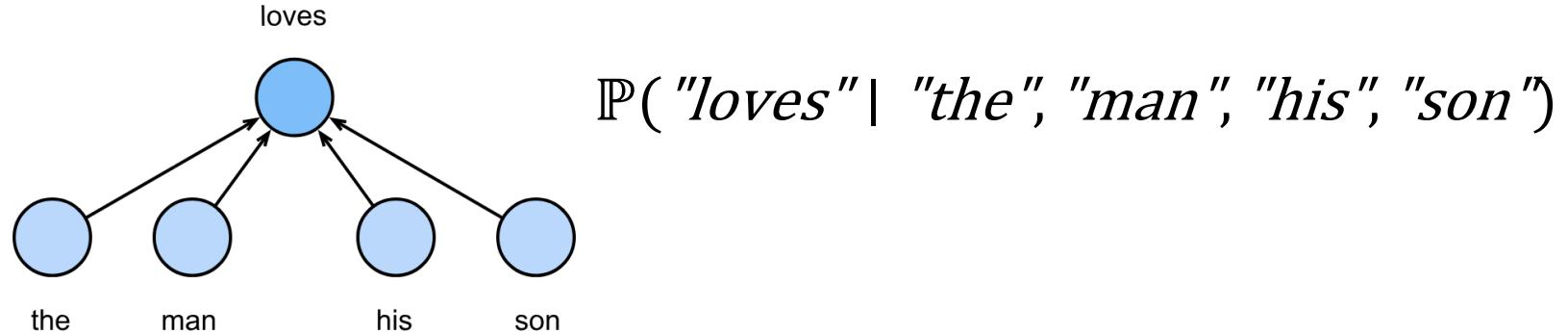
- 样本噪音词 w_n 在窗口中未显示的概率

$$\mathbb{P}(D = 0 | w_c, w_n) = 1 - \sigma(\mathbf{u}_n^T \mathbf{v}_c)$$

- 加入似然函数
- 最大化似然函数相当于用二元逻辑回归损失求解二元分类问题

CBOW 模型

- CBOW - Continuous Bag Of Words
- 基于上下文词生成中心词



似然函数

- 计算概率

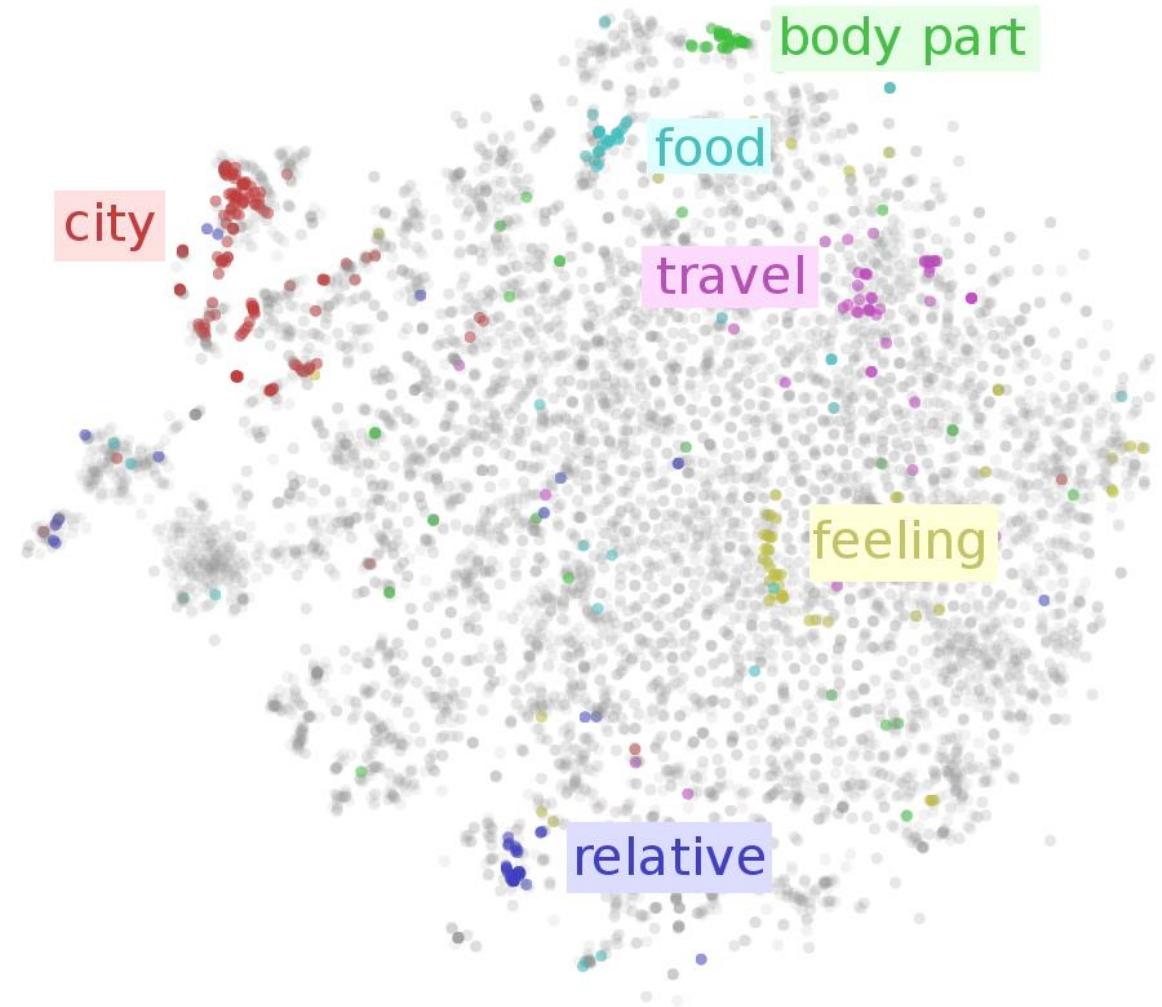
$$\mathbb{P}(w_c \mid w_{o_1}, \dots, w_{o_{2m}}) = \frac{\exp\left(\frac{1}{2m} \mathbf{u}_c^\top (\mathbf{v}_{o_1} + \dots + \mathbf{v}_{o_{2m}})\right)}{\sum_{i \in \mathcal{V}} \exp\left(\frac{1}{2m} \mathbf{u}_i^\top (\mathbf{v}_{o_1} + \dots + \mathbf{v}_{o_{2m}})\right)}$$

- 可能性

$$\prod_{t=1}^T \mathbb{P}(w^{(t)} \mid w^{(t-m)}, \dots, w^{(t-1)}, w^{(t+1)}, \dots, w^{(t+m)})$$

代码 ...

更多嵌入模型



子词嵌入 (fastText)

- 英语单词通常有内部结构和形成方法
 - dog, dogs, dogcatcher
- 每个中心词可以表示为一组子词
 - where -> <where>
 - n-gram ($n = 3$): "<wh", "whe", "her", "ere", "re>"
- 适用于冗长但不常见的单词
 - 例如: pneumonoultramicroscopicsilicovolcanoconiosis
(火山肺矽病)



子词嵌入 (fastText)

- 对于单词 w , \mathcal{G}_w 是长度为 3-6 的词的并集
- 中心向量

$$\mathbf{u}_w = \sum_{g \in \mathcal{G}_w} \mathbf{u}_g$$

- 其余模型与 skip-gram 相同

全局向量的词嵌入 (GloVe)

- 表示: $q_{ij} = \frac{\exp(\mathbf{u}_j^\top \mathbf{v}_i)}{\sum_{k \in \mathcal{V}} \exp(\mathbf{u}_k^\top \mathbf{v}_i)}$
- 重写 skip-gram 的负对数似然函数

$$-\log \left[\prod_{t=1}^T \prod_{-m \leq j \leq m, j \neq 0} \mathbb{P}(w^{(t+j)} | w^{(t)}) \right]$$

- 相当于 $-\sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} x_{ij} \log q_{ij}$ (足量样本 x_{ij})

全局向量的词嵌入 (GloVe)

- 其它表示：

$$-\sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} x_{ij} \log q_{ij} = -\sum_{i \in \mathcal{V}} x_i \sum_{j \in \mathcal{V}} p_{ij} \log q_{ij}$$

$$x_i = \sum_j x_{ij}, p_{ij} = x_{ij} / x_i$$

交叉熵

全局向量的词嵌入 (GloVe)

- 用 (易于计算的) 对数平方损失 (log square loss) 替换
交叉熵损失 (cross entropy loss)

$$\sum_{j \in \mathcal{V}} p_{ij} \log q_{ij} \rightarrow \sum_{j \in \mathcal{V}} (\log p_{ij} - \log q'_{ij})^2$$

- 为中心词和上下文词添加偏见词 $q'_{ij} = \exp(\mathbf{u}_j^\top \mathbf{v}_i)$
- 用 [0,1] 中的单调递增函数替换权重 x_i

$$\sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} h(x_{ij}) (\mathbf{u}_j^\top \mathbf{v}_i + b_i + c_j - \log x_{ij})^2$$

代码...

总结

- 嵌入向量 (Embeddings)
- 词嵌入 (Word2vec)
 - Skip-Gram
 - CBOW
- 子词嵌入 (fastText)
- 全局向量的词嵌入 (GloVe)

动手学深度学习

23. 编码器解码器，Seq2seq模型，束搜索

中文教材：zh.d2l.ai

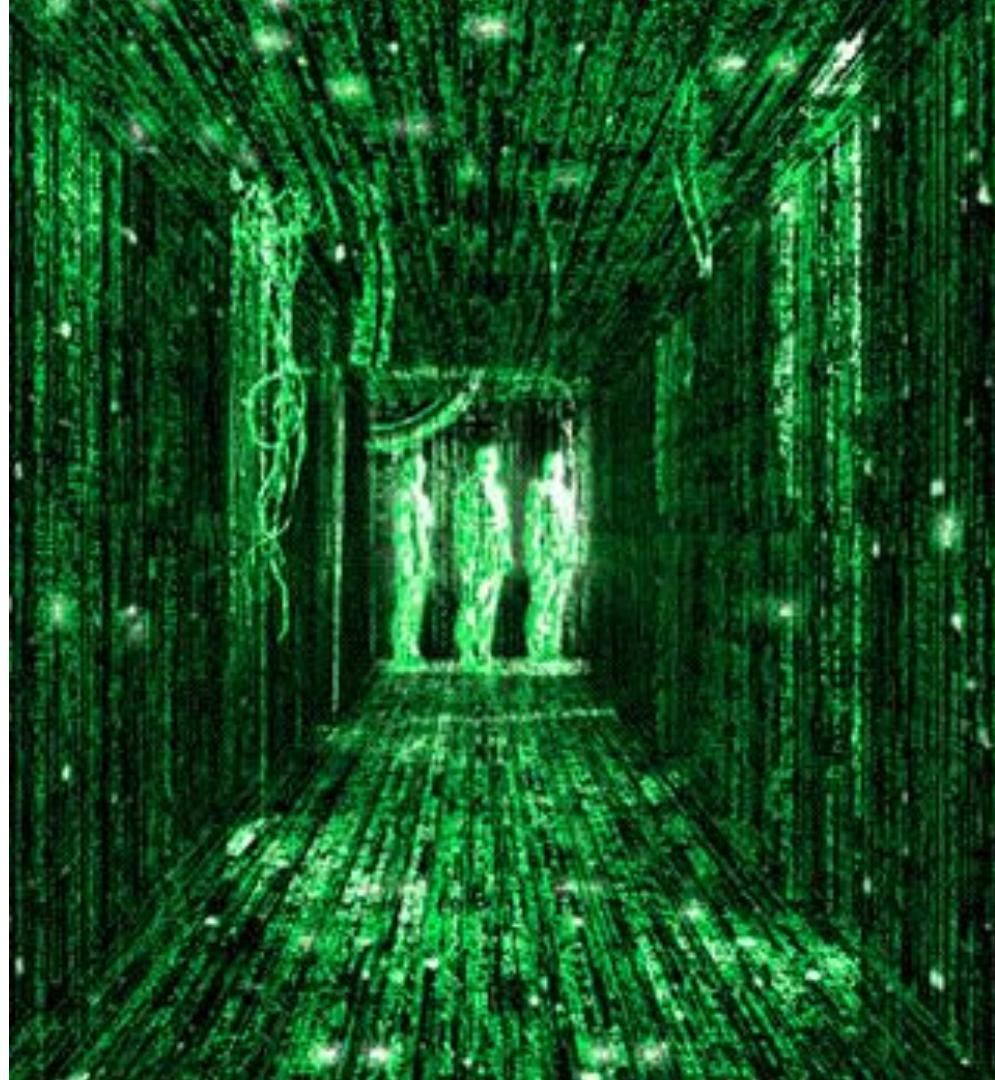
英文教材：www.d2l.ai

教学视频：<https://courses.d2l.ai/berkeley-stat-157/units/seq2seq.html>

概要

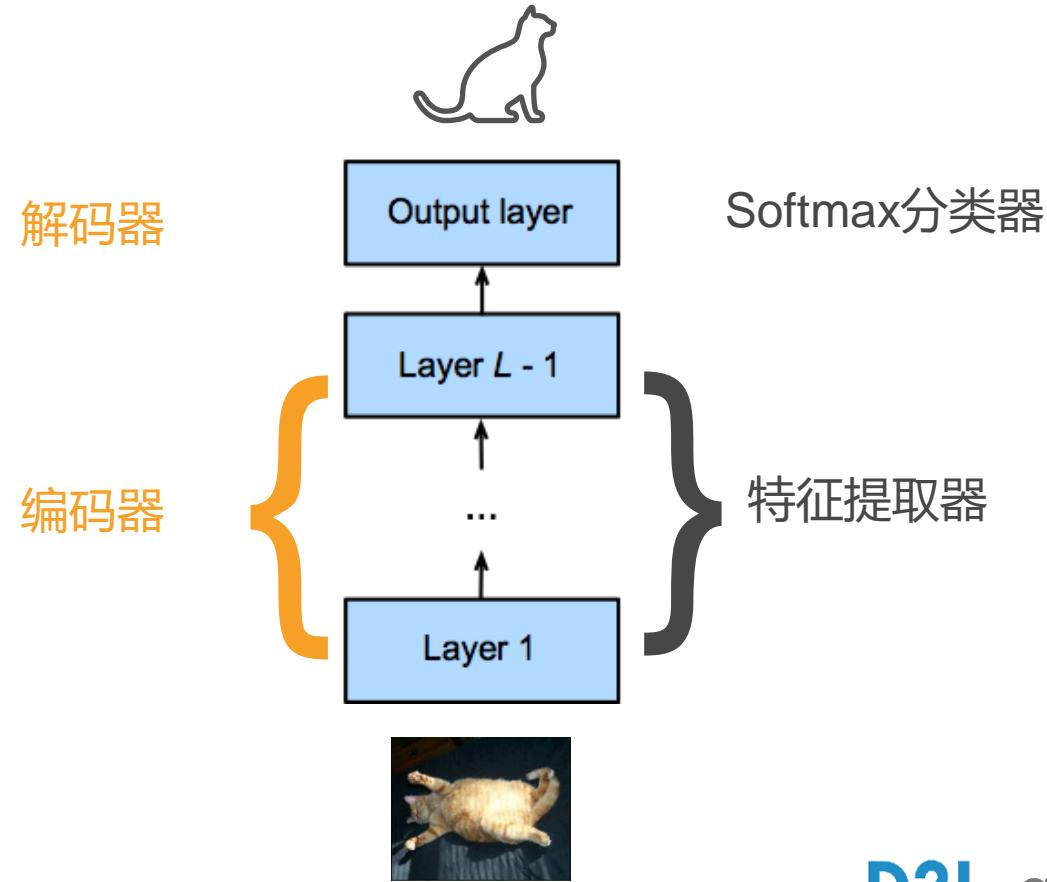
- 编码器 - 解码器 (Encoder - Decoder)
- Seq2seq 模型
- 束搜索 (Beam Search)
 - 贪婪搜索 (Greedy Search)
 - 穷举搜索 (Exhaustive Search)
 - 束搜索

编码器 - 解码器 (Encoder - Decoder)



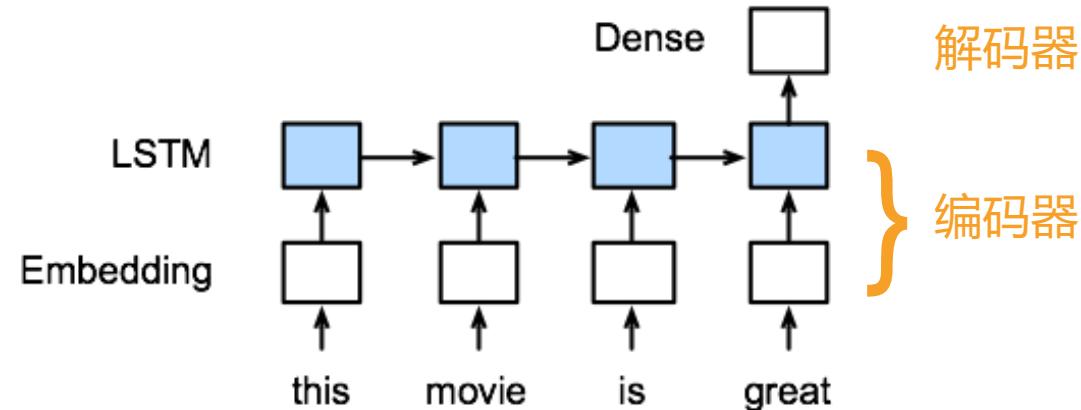
重新思考 CNN

- 编码器(Encoder)：将输入编码为中间表示（特征）
- 解码器(Decoder)：将特征解码为输出



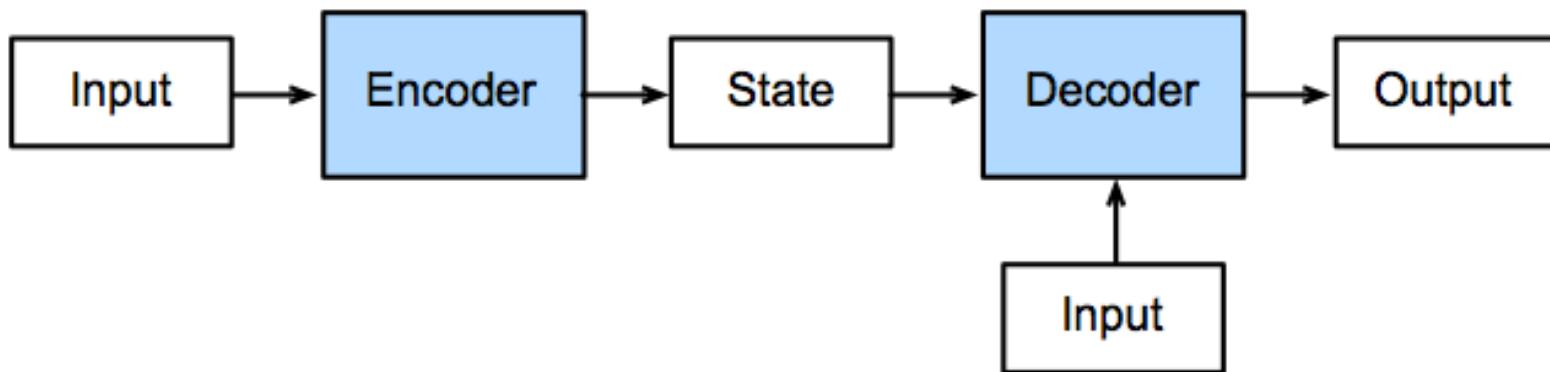
重新思考 RNN

- 编码器 (Encoder) :
将输入编码为中间
表示 (特征)
- 解码器(Decoder) :
将特征解码为输出



编码器 - 解码器架构

- 模型分为两部分
 - 编码器 (Encoder) 加工输入
 - 解码器 (Decoder) 生成输出



编码器 Class

```
class Encoder(nn.Block):
    def __init__(self, **kwargs):
        super(Encoder, self).__init__(**kwargs)

    def forward(self, X):
        raise NotImplementedError
```

解码器 Class

- 使用编码器输出和任何其他信息创建状态

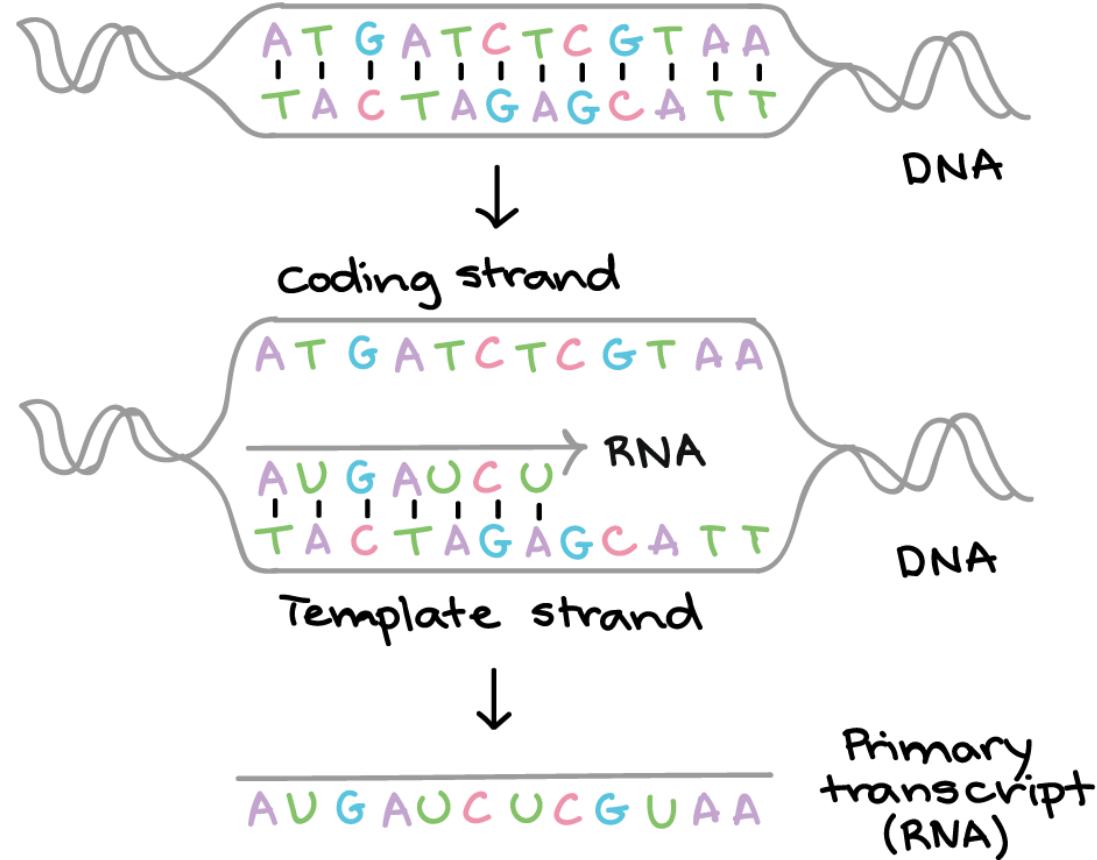
```
class Decoder(nn.Block):  
    def __init__(self, **kwargs):  
        super(Decoder, self).__init__(**kwargs)  
  
    def init_state(self, enc_outputs, *args):  
        raise NotImplementedError  
  
    def forward(self, X, state):  
        raise NotImplementedError
```

编码器-解码器 Class

```
class EncoderDecoder(nn.Block):
    def __init__(self, encoder, decoder, **kwargs):
        super(EncoderDecoder, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder

    def forward(self, enc_X, dec_X, *args):
        enc_outputs = self.encoder(enc_X)
        dec_state = self.decoder.init_state(enc_outputs, *args)
        return self.decoder(dec_X, dec_state)
```

Seq2seq 模型



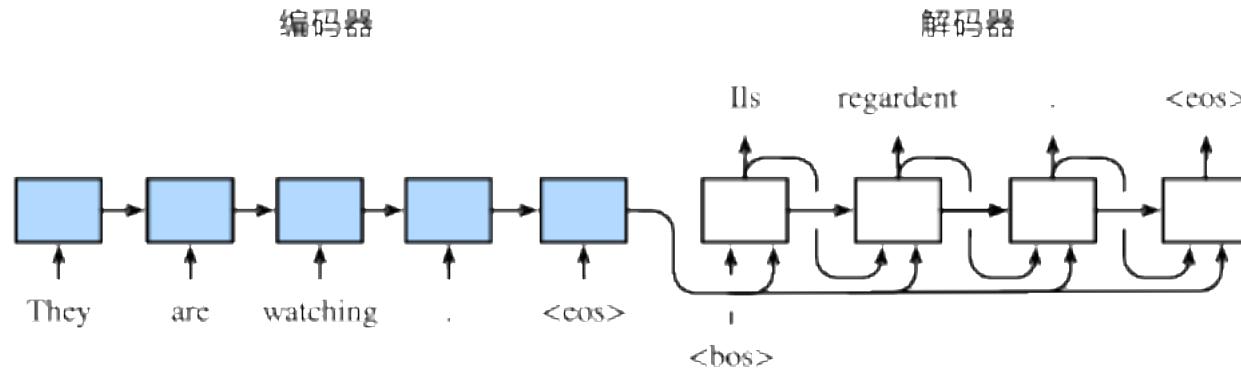
机器翻译

- 给定源语言中的句子，翻译成目标语言
- 这两个序列可以具有不同的长度

The screenshot shows a machine translation interface. At the top, there are tabs for "Text" (selected) and "Documents". Below that is a row of language detection and selection buttons: "ENGLISH - DETECTED", "ENGLISH", "SPANISH", "FRENCH", a dropdown arrow, a double-headed arrow icon, and "CHINESE (SIMPLIFIED)" (selected). Another row of buttons follows: "ENGLISH", "SPANISH", a dropdown arrow, and a star icon. The main area contains two text boxes separated by a double-headed arrow. The left text box contains the English sentence: "STAT-157 is a great deep learning introduction course". The right text box contains the Chinese translation: "STAT-157是一个很棒的深度学习入门课程". Below the Chinese text is its pinyin transcription: "STAT-157 shì yīgè hěn bàng de shēndù xuéxí rùmén kèchéng". At the bottom of each text box are small icons for microphone, speaker, edit, and share.

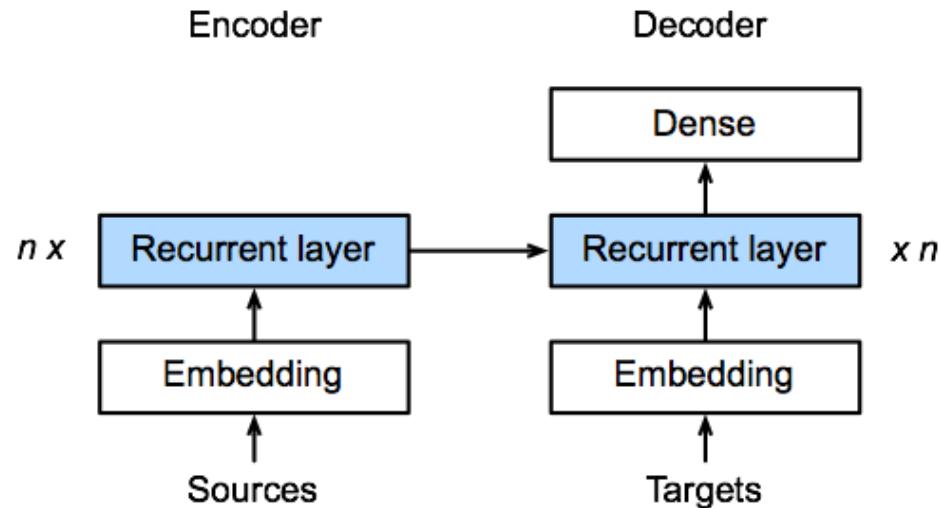
Seq2seq 模型

- 编码器是读取输入序列的 RNN
- 解码器使用另一个 RNN 来生成输出



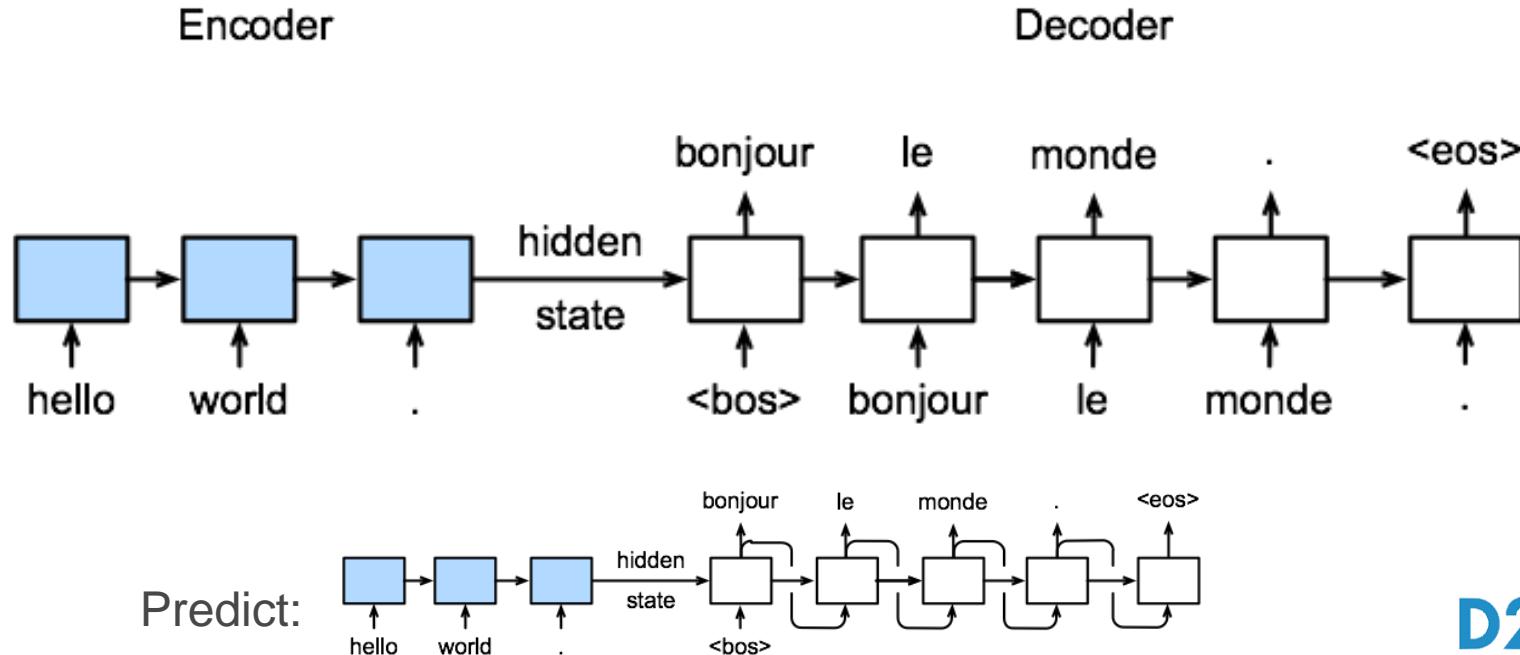
Seq2seq 模型

- 编码器是没有输出层的标准 RNN 模型
- 编码器在上一时间步骤中的隐含状态用作解码器的初始隐藏状态



机器翻译训练

- 在训练期间，解码器（Decoder）用目标语言句子作为输入



代码...

束搜索

Beam Search



贪婪搜索

- 我们在预测期间在 seq2seq 模型中使用了贪婪搜索
- 它可能不是最理想的

贪婪搜索:

$$0.5 \times 0.4 \times 0.4 \times 0.6 = 0.048$$

时间步	1	2	3	4
A	0.5	0.1	0.2	0.0
B	0.2	0.4	0.2	0.2
C	0.2	0.3	0.4	0.2
<eos>	0.1	0.2	0.2	0.6

更好的选择:

$$0.5 \times 0.3 \times 0.6 \times 0.6 = 0.054$$

时间步	1	2	3	4
A	0.5	0.1	0.1	0.1
B	0.2	0.4	0.6	0.2
C	0.2	0.3	0.2	0.1
<eos>	0.1	0.2	0.1	0.6

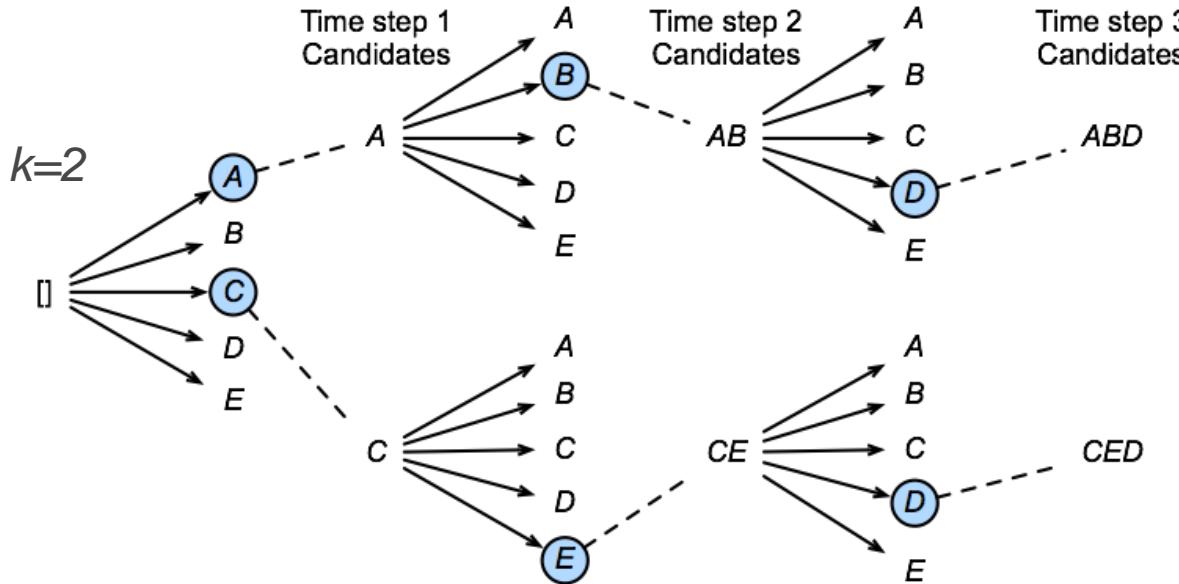
穷举搜索 (exhaustive search)

- 对于每个可能的序列，计算其概率并选择最佳序列
- 如果输出词汇量大小为 n ，并且最大序列长度为 T ，那么我们需要检查 n^T 序列
 - 这在计算上是不可行的

$$n = 10000, T = 10: n^T = 10^{40}$$

束搜索

- 每次都保留最好的 k (束搜索) 候选
- 通过向候选束添加新项目，来搜索 kn 序列，然后保留前 k 个



束搜索

- 时间复杂度为 $O(knT)$

$$k = 5, n = 10000, T = 10:$$

$$knT = 5 \times 10^5$$

- 每个候选的最终得分是

$$\frac{1}{L^\alpha} \log \mathbb{P}(y_1, \dots, y_L) = \frac{1}{L^\alpha} \sum_{t'=1}^L \log \mathbb{P}(y_{t'} \mid y_1, \dots, y_{t'-1}, \mathbf{c})$$

- 常用值 $\alpha = 0.75$

总结

- 编码器 - 解码器 (Encoder - Decoder)
- Seq2seq 模型
- 束搜索 (Beam Search)
 - 贪婪搜索 (greedy search)
 - 穷举搜索 (exhaustive search)
 - 束搜索

动手学深度学习

24. 注意力机制

中文教材: zh.d2l.ai

英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/attention.html>

概要

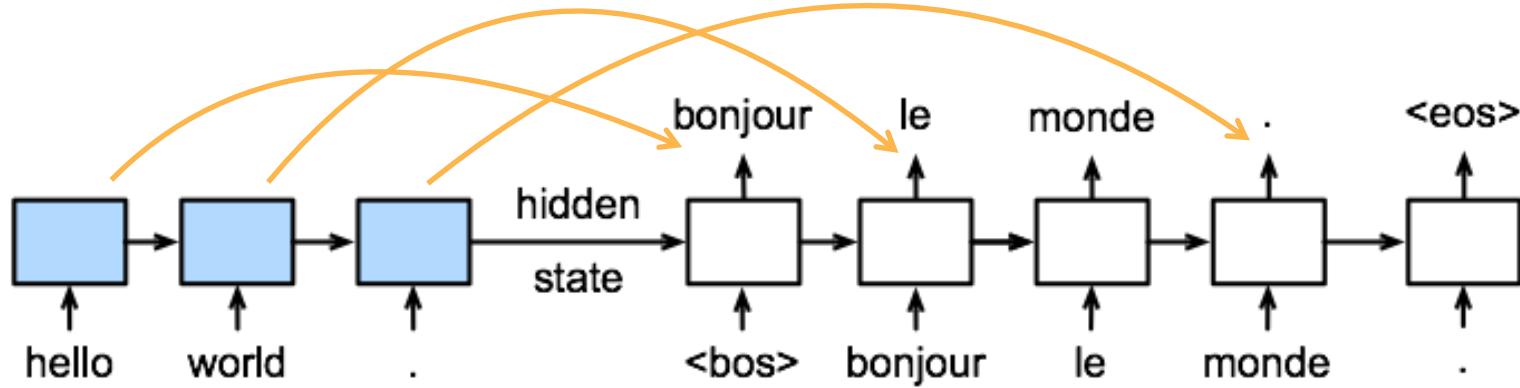
- 注意力机制

注意力机制



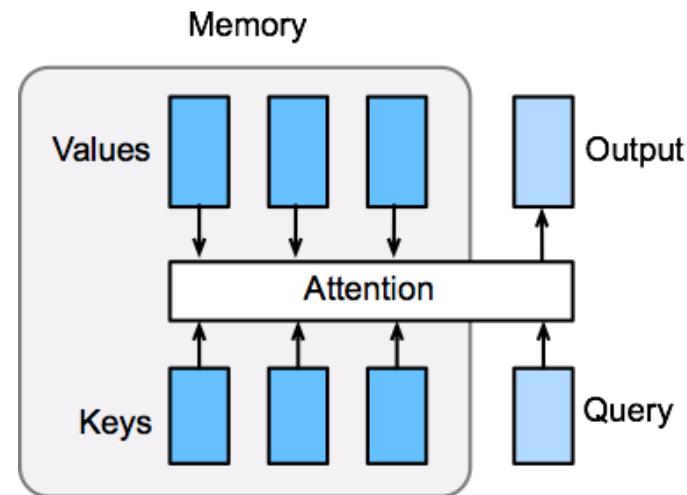
动机

- 每个生成的token可能与不同的源token相关



注意力层

- 注意力层明确选择相关信息
 - 它的存储器 (memory) 由“键值对”组成
 - 键和查询越相似，则输出的值越相近



注意力层

- 假设“一条询问”为 $\mathbf{q} \in \mathbb{R}^{d_q}$, 存储器为 $(\mathbf{k}_1, \mathbf{v}_1), \dots, (\mathbf{k}_n, \mathbf{v}_n)$;
 $\mathbf{k}_i \in \mathbb{R}^{d_k}, \mathbf{v}_i \in \mathbb{R}^{d_v}$
- 计算 n 分数 a_1, \dots, a_n ; $a_i = \alpha(\mathbf{q}, \mathbf{k}_i)$
- 使用 softmax 获得注意力
 $b_1, \dots, b_n = \text{softmax}(a_1, \dots, a_n)$
- 输出是值的加权和
$$\mathbf{o} = \sum_{i=1}^n b_i \mathbf{v}_i$$

改变 α 可以
获得不同
的注意力
层

点乘注意力

- 假设询问的长度与值相同 $\mathbf{q}, \mathbf{k}_i \in \mathbb{R}^d$

$$\alpha(\mathbf{q}, \mathbf{k}) = \langle \mathbf{q}, \mathbf{k} \rangle / \sqrt{d}$$

- 向量化版本

- m 个询问 $\mathbf{Q} \in \mathbb{R}^{m \times d}$ 和 n 个键 $\mathbf{K} \in \mathbb{R}^{n \times d}$

$$\alpha(\mathbf{Q}, \mathbf{K}) = \mathbf{Q}\mathbf{K}^T / \sqrt{d}$$

多层感知注意力

- 可学习的参数 $\mathbf{W}_k \in \mathbb{R}^{h \times d_k}, \mathbf{W}_q \in \mathbb{R}^{h \times d_q}, \mathbf{v} \in \mathbb{R}^h$

$$\alpha(\mathbf{k}, \mathbf{q}) = \mathbf{v}^T \tanh(\mathbf{W}_k \mathbf{k} + \mathbf{W}_q \mathbf{q})$$

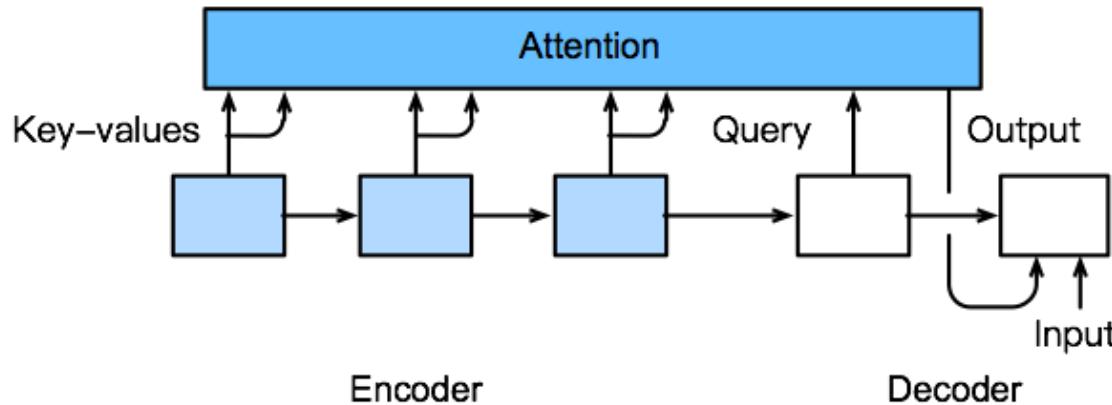
- 相当于连接“键” (key) 和 “询问” (query), 然后输入隐含大小为 h 和输出大小 1 的单个隐含层感知

代码...

Seq2seq 与注意力机制

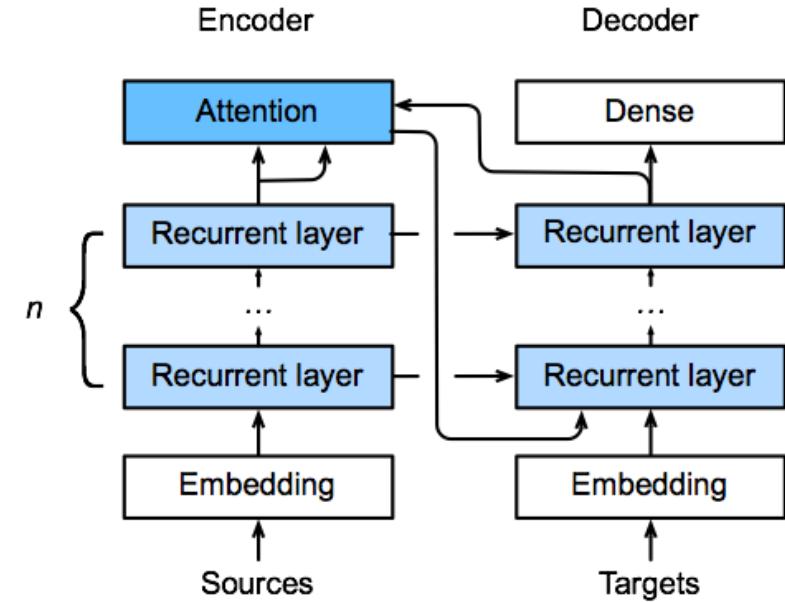
模型架构

- 添加额外的注意层以编码器的输出作为存储器
- 注意力的输出用作解码器的输入



编码器—解码器上的注意力机制

- 使用编码器中最后一个循环神经网络层的输出
- 然后，注意力输出与嵌入输出拼接，以输入解码器中的第一个循环神经网络层



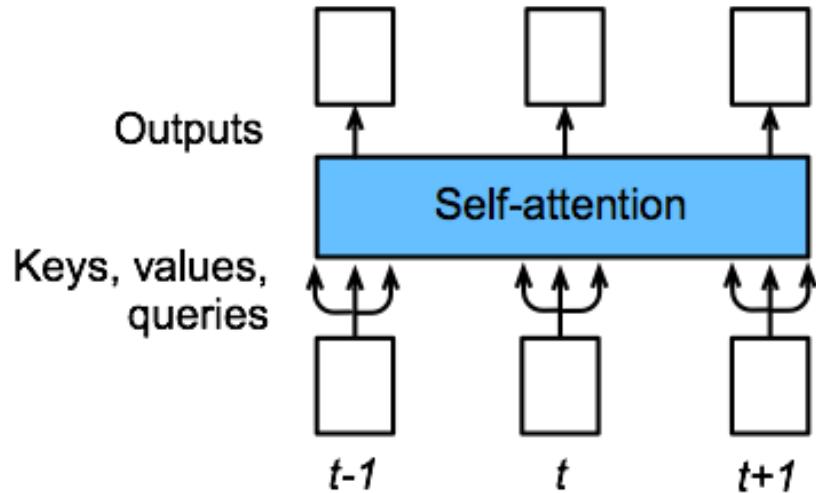
代码...

变换器模型 (Transformer)



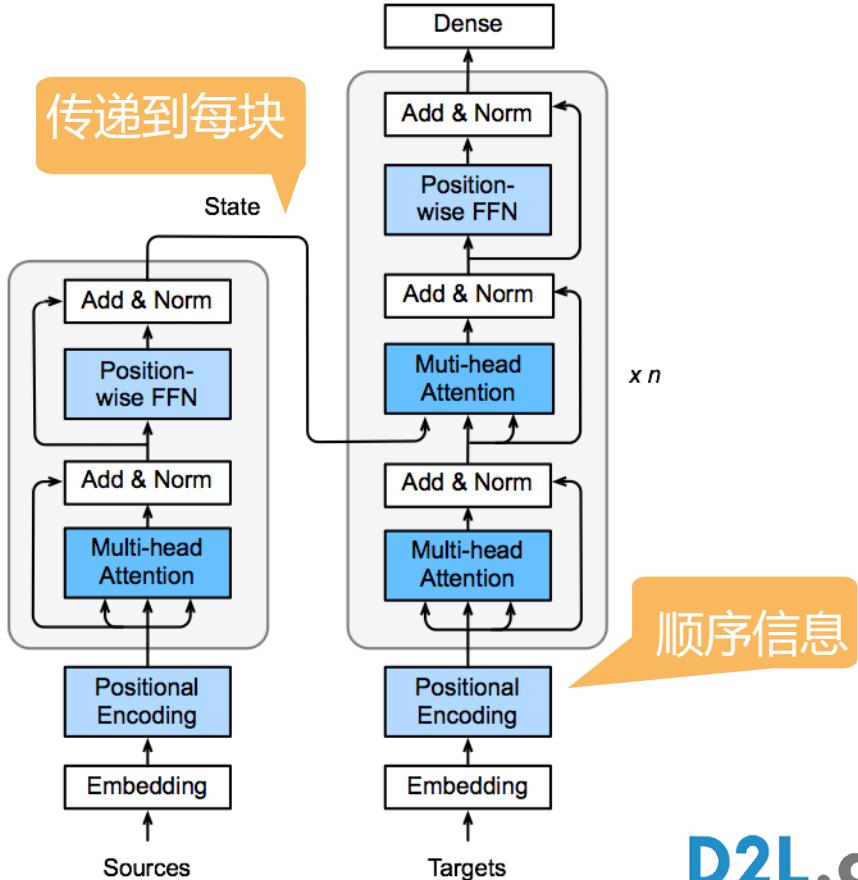
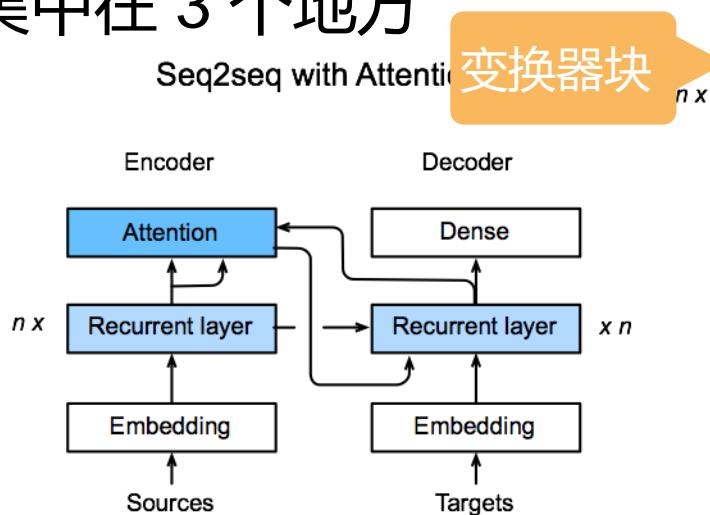
自注意力机制

- 要使用 n 个输入生成 n 个输出，我们可以将每个输入复制为键 (key)、值 (value) 和查询 (query) 中
- 不保留顺序信息
- 并行计算

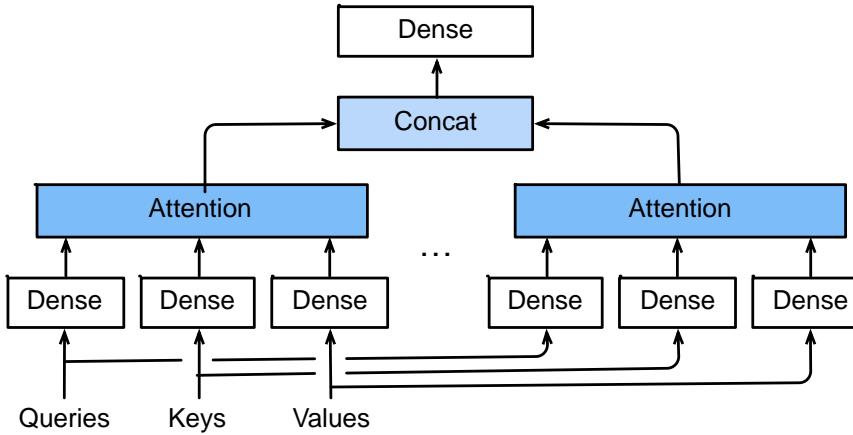


Transformer模型架构

- 它是一个编码器 - 解码器架构
- 与 seq2seq 不同，注意力集中在 3 个地方



多头注意力机制 (Multi-head Attention)

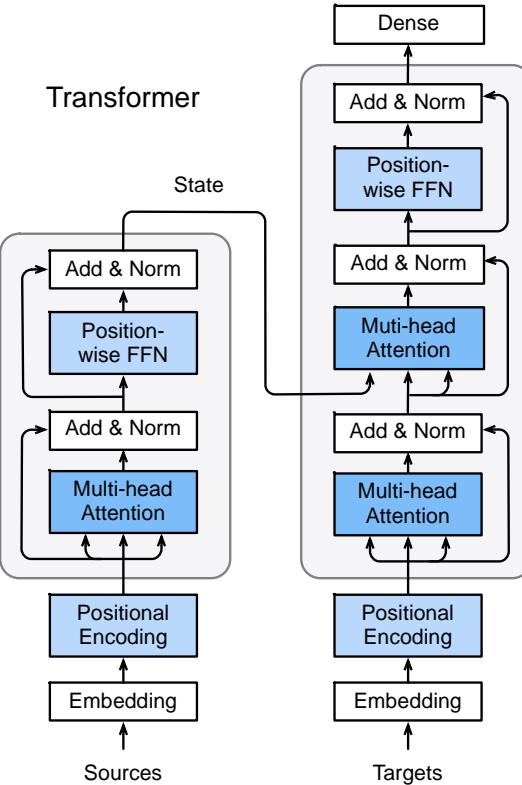


$$\mathbf{W}_q^{(i)} \in \mathbb{R}^{p_q \times d_q}, \mathbf{W}_k^{(i)} \in \mathbb{R}^{p_k \times d_k}, \text{ and } \mathbf{W}_v^{(i)} \in \mathbb{R}^{p_v \times d_v}$$

$$\mathbf{o}^{(i)} = \text{attention}(\mathbf{W}_q^{(i)} \mathbf{q}, \mathbf{W}_k^{(i)} \mathbf{k}, \mathbf{W}_v^{(i)} \mathbf{v})$$

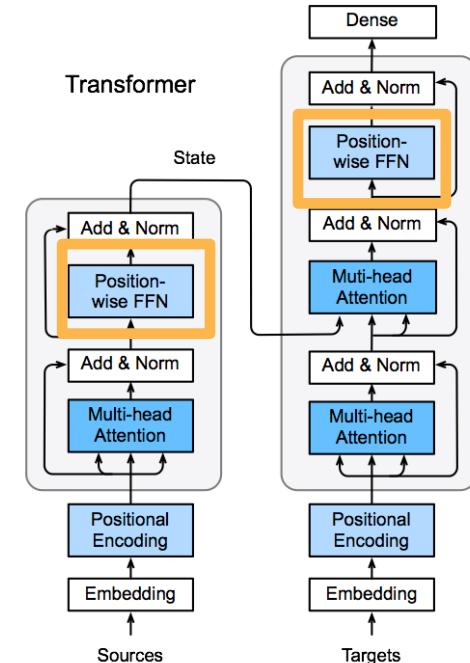
for $i = 1, \dots, h$

$$\mathbf{o} = \mathbf{W}_o \begin{bmatrix} \mathbf{o}^{(1)} \\ \vdots \\ \mathbf{o}^{(h)} \end{bmatrix}$$



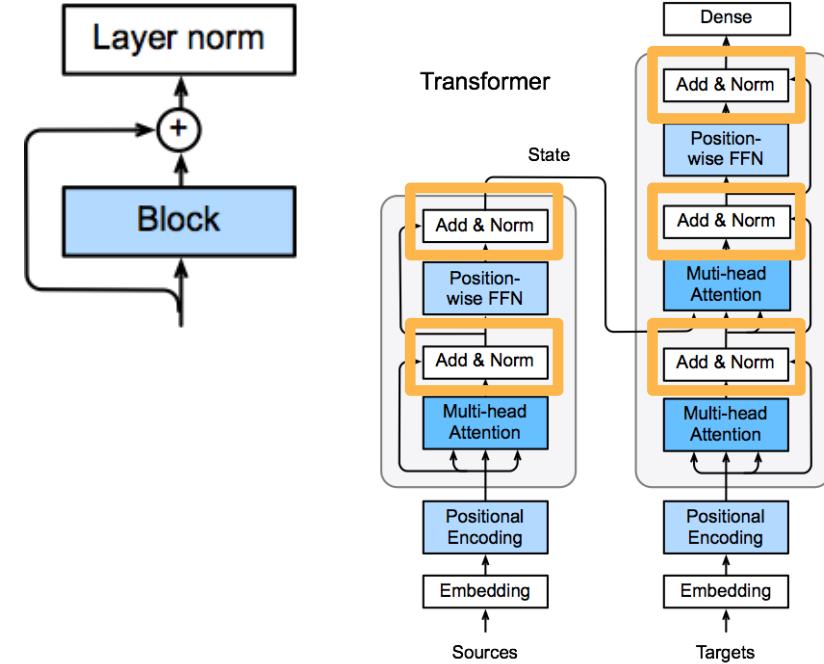
位置前馈网络

- 将输入 (批量大小, 序列长度, 特征集大小) 重新整形为 (批量*序列长度, 特征集大小)
- 用两层 MLP
- 转换为 3-D 形态
- 等于应用两个 (1,1) 个卷积层



添加与归一化

- 层规范 (Layer Norm) 类似于批量规范 (Batch Norm)
- 但是平均值和方差是沿最后一个维度计算的
 - $X.\text{mean} (\text{axis} = -1)$ 而不是批量归一化
 - $X.\text{mean}$ 中的第一个批次维度 ($\text{axis} = 0$)



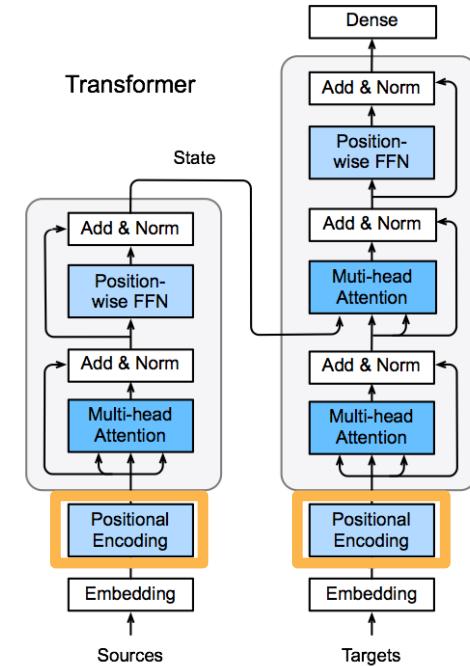
位置编码

- 假设嵌入 $X \in \mathbb{R}^{l \times d}$ 输出的形状 (序列长度, 嵌入维度)
- 创建 $P \in \mathbb{R}^{l \times d}$

$$P_{i,2j} = \sin(i/10000^{2j/d})$$

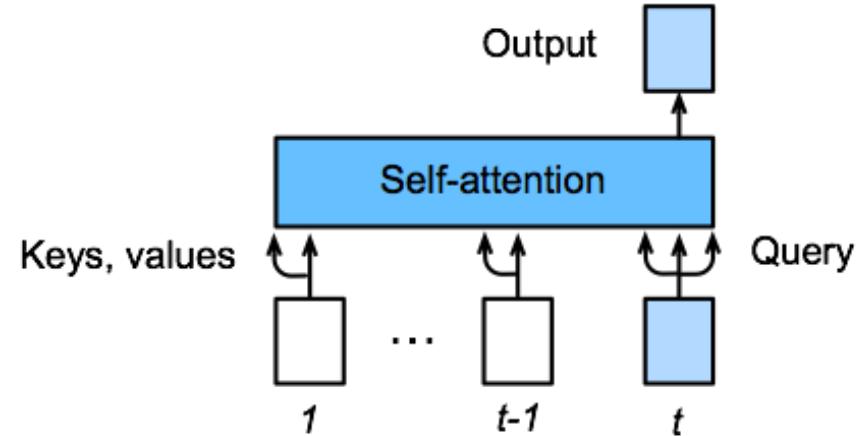
$$P_{i,2j+1} = \cos(i/10000^{2jd})$$

- 输出 $X + P$



预测

- 在时刻 t 预测：
 - 用之前输入的键和值
 - 时刻 t 输入作为查询，以及键和值，以预测输出



代码 ...

BERT



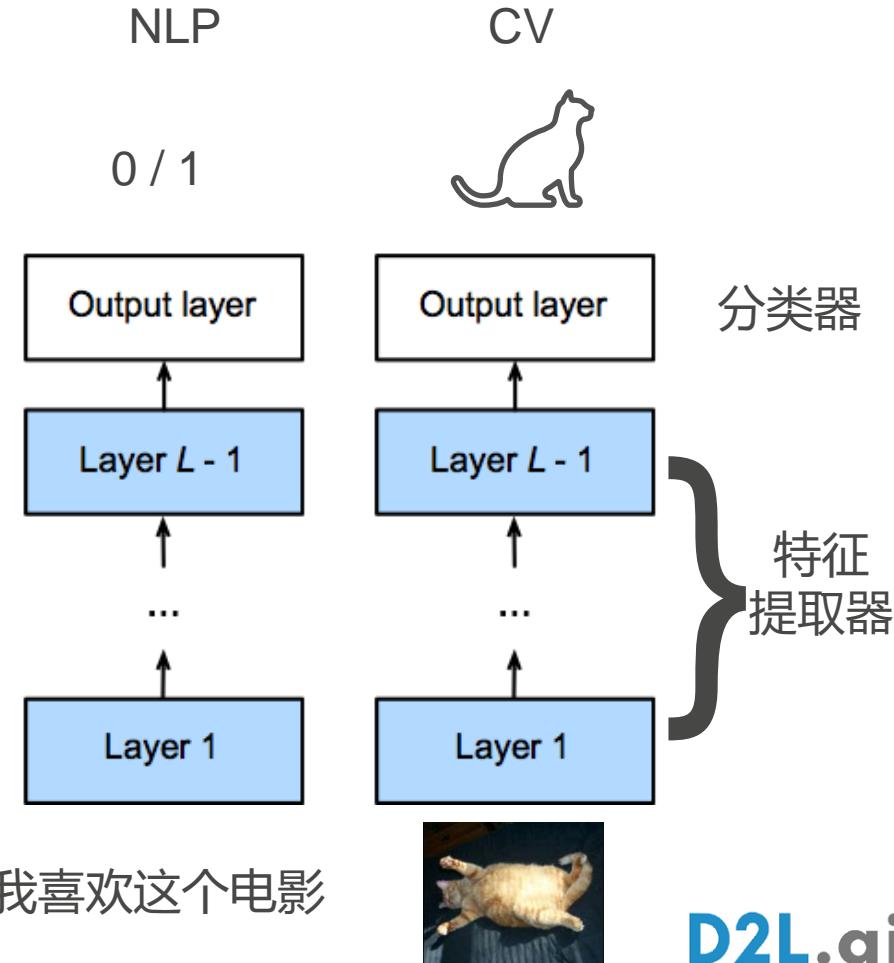
2L.ai

NLP中的迁移学习 (Transfer Learning)

- 使用预先训练的模型为新任务提取单词/句子功能
 - 例如：word2vec 或语言模型
- 通常不更新预先训练的模型的权重
- 需要构建一个新模型来捕获新任务所需的信息
 - Word2vec 忽略顺序信息，这个语言模型只查看单一向

BERT 的动机

- 基于微调的 NLP 方法
- 预训练模型捕获足够多的数据信息
- 只需要为新任务添加简单的输出层

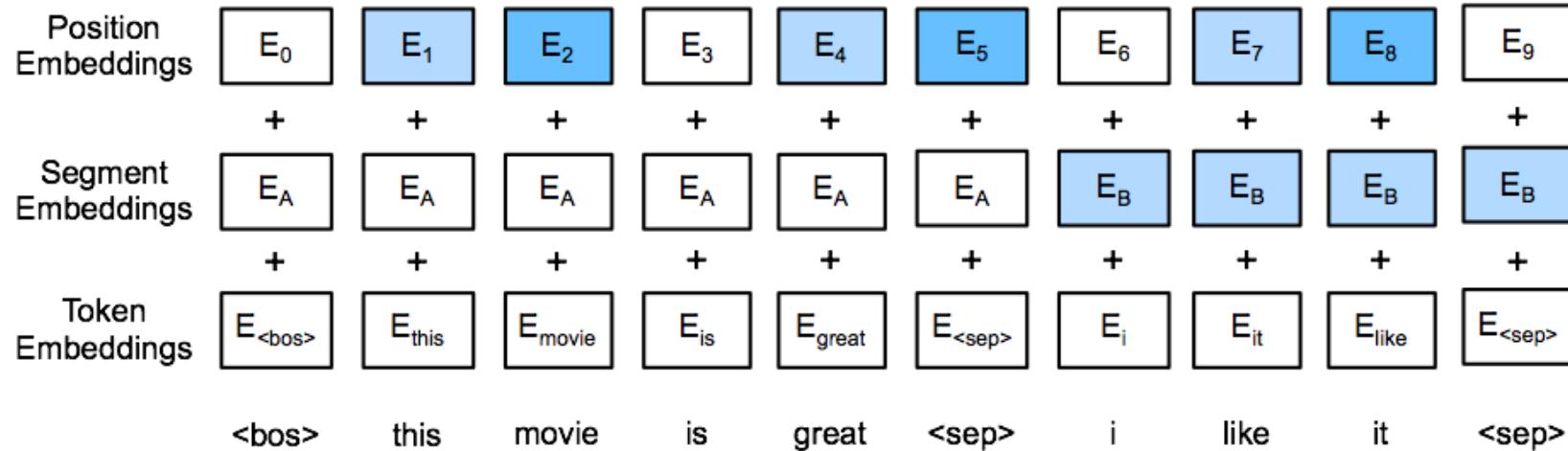


BERT 架构

- 一个（巨大的）变换器模型编码器（没有解码器）
- 两种模型大小：
 - 基础版：# blocks = 12, 隐含大小= 768, # heads = 12, # 参数 = 110M
 - 增强版：# blocks = 24, 隐含大小= 1024, # heads = 16, # 参数= 340M
- 使用超过30亿单词的大型语料库（书籍和维基百科）训练

输入

- 每个样本都是一对句子
- 添加其他细分嵌入



预训练任务1：掩码语言模型

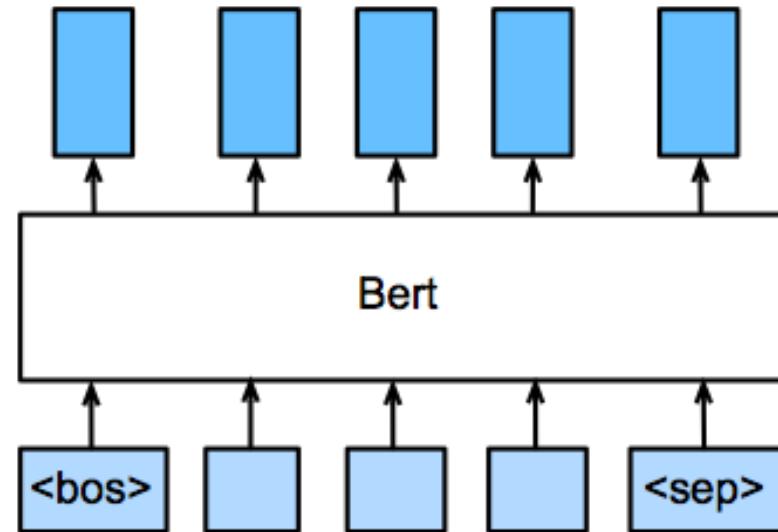
- 在每个句子中随机掩盖（例如 15%）标记，预测这些掩码标记（`<mask>`）
 - 变换器模型是双向的，它打破了标准语言模型的单向限制
- 微调任务中没有掩码标记（`<mask>`）
 - 80% 的时间，用 `<mask>` 替换选定的标记
 - 10% 的时间，用随机挑选的picked tokens替换
 - 10% 的时间，保留原始标记

预训练任务2：下一句话预测

- 50% 的时间，选择一个连续的句子对
 - <bos> 这部电影很棒 <sep> 我喜欢它 <sep>
- 50% 的时间，选择一个随机的句子对
 - <bos> 这部电影很棒 <sep> hello world <sep>
- 将变换器模型的输出 <bos> 输入到稠密层以预测它是否是顺序对 (sequential pair)

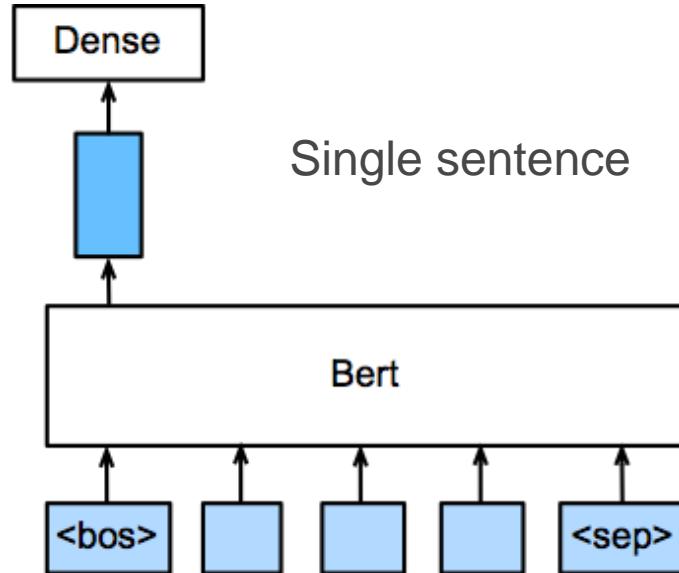
用 Bert 微调

- Bert 为捕获上下文信息的每个标记返回一个特征向量
- 不同的微调任务使用不同的向量集

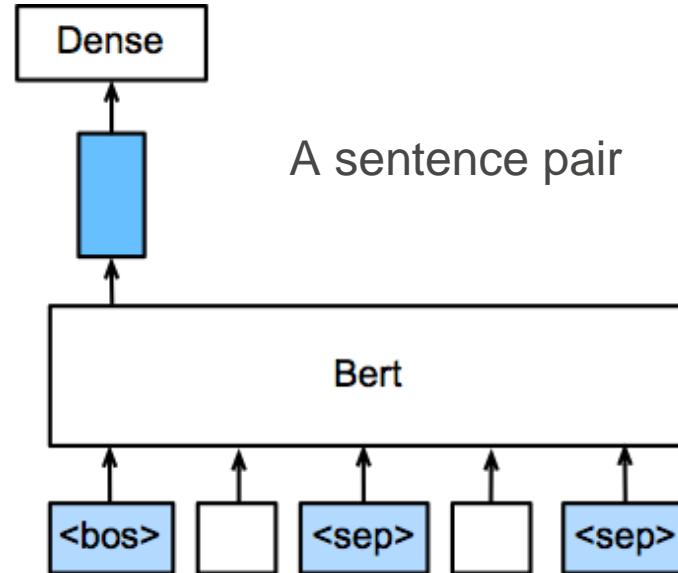


语句分类

- 将 `<bos>` 标记向量输入稠密输出层



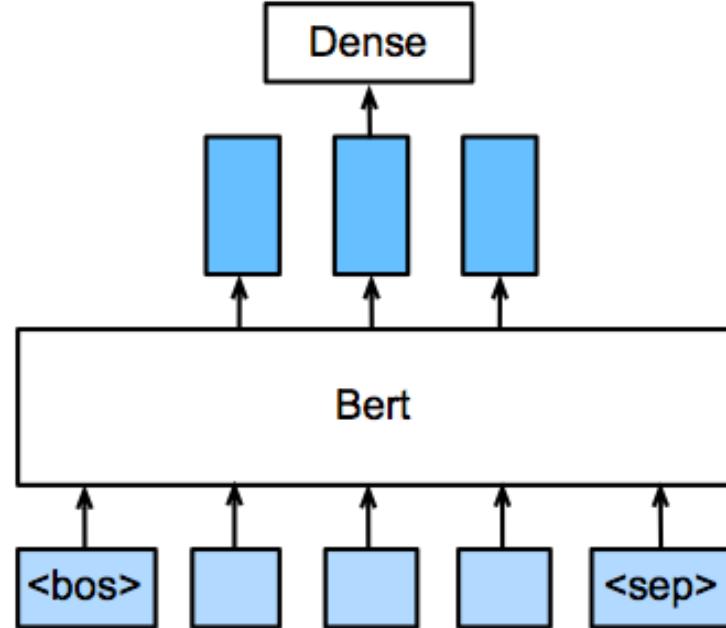
Single sentence



A sentence pair

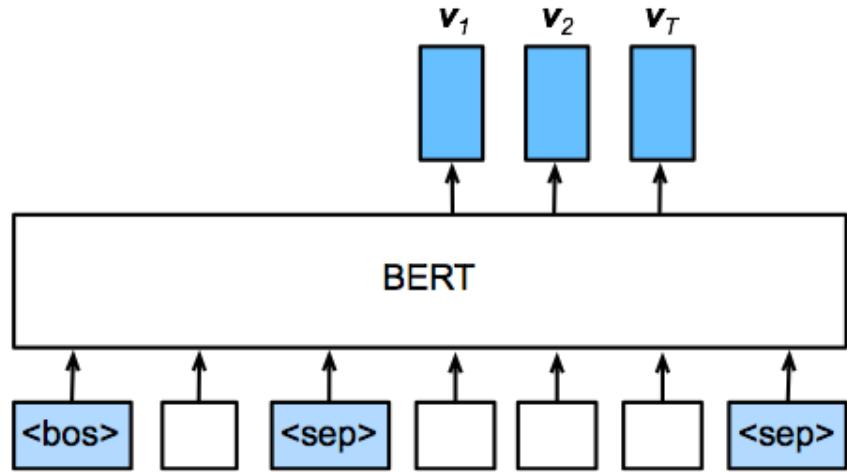
命名实体识别

- 确定标记是否是命名实体，例如人员，组织和位置等等
- 将每个非特殊标记向量馈送到稠密输出层



自动问答

- 给定问题和描述文本，找到答案，这是描述中的文本段
- 给定 p_i ，描述中的第 i 个标记，学习 s 中的 p_i ，第 i 个标记是这段开始的概率：
$$p_1, \dots, p_T = \text{softmax}(\langle s, v_1 \rangle, \dots, \langle s, v_T \rangle)$$
- 同样可以学习第 i 个标记是这段结局的概率



GluonNLP 有更多资源/代码/模型...

<https://gluon-nlp.mxnet.io/>

动手学深度学习

25.优化问题

中文教材: zh.d2l.ai

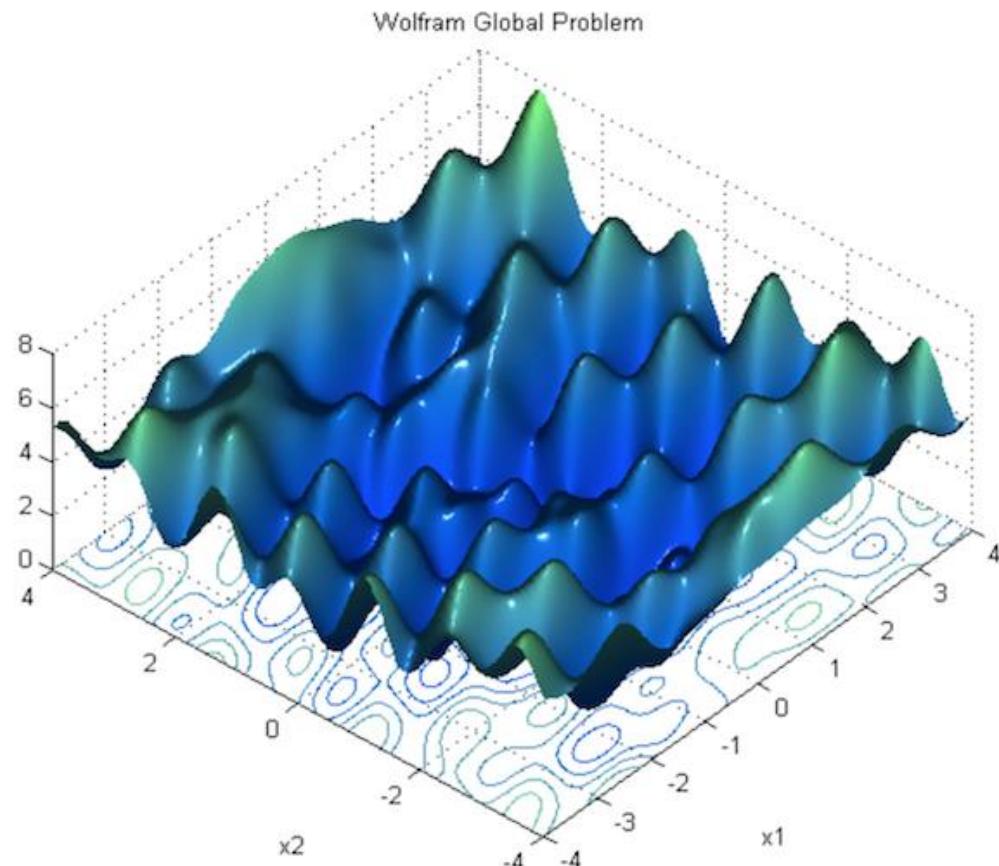
英文教材: www.d2l.ai

教学视频: <https://courses.d2l.ai/berkeley-stat-157/units/optimization.html>

概要

- 优化问题
 - 局部最小值和全局最小值
 - 凸集和凸函数
 - 凸优化证明
- 梯度下降
 - 学习率
 - 收敛率证明
 - 随机梯度下降
 - 小批量随机梯度下降

优化问题



优化问题

- 一般形式：

$\text{minimize } f(\mathbf{x}), \text{ subject to } \mathbf{x} \in C$

- 成本函数 $f: \mathbb{R}^n \rightarrow \mathbb{R}$
- 约束集的例子

$$C = \{\mathbf{x} \mid h_1(\mathbf{x}) = 0, \dots, h_m(\mathbf{x}) = 0, g_1(\mathbf{x}) \leq 0, \dots, g_r(\mathbf{x}) \leq 0\}$$

- 如果 $C = \mathbb{R}^n$ ，则不受约束

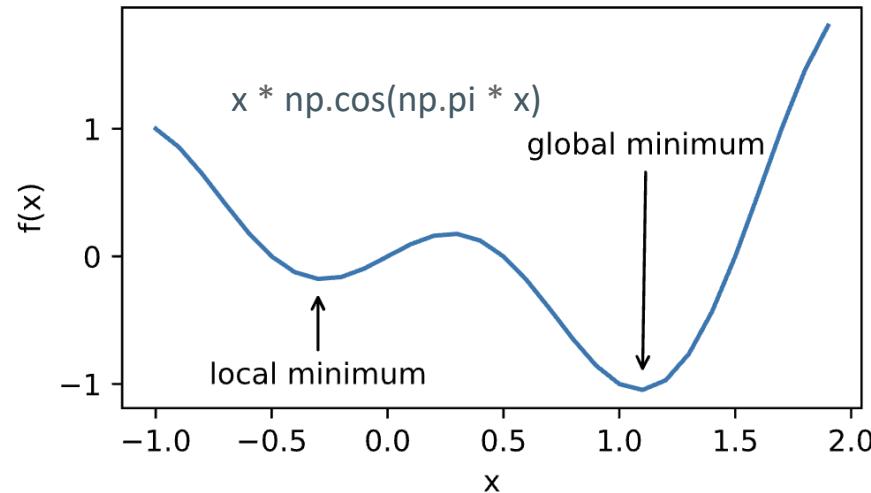
局部最小值和全局最小值

- 大多数优化问题都没有闭式 (closed-form) 解决方案
- 我们的目标是通过迭代方法找到最小值
- 全局最小值 \mathbf{x}^* :

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x} \in C$$

- 局部最小值 \mathbf{x}^* , 存在 ε :

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \forall \mathbf{x}: \|\mathbf{x} - \mathbf{x}^*\| \leq \varepsilon$$

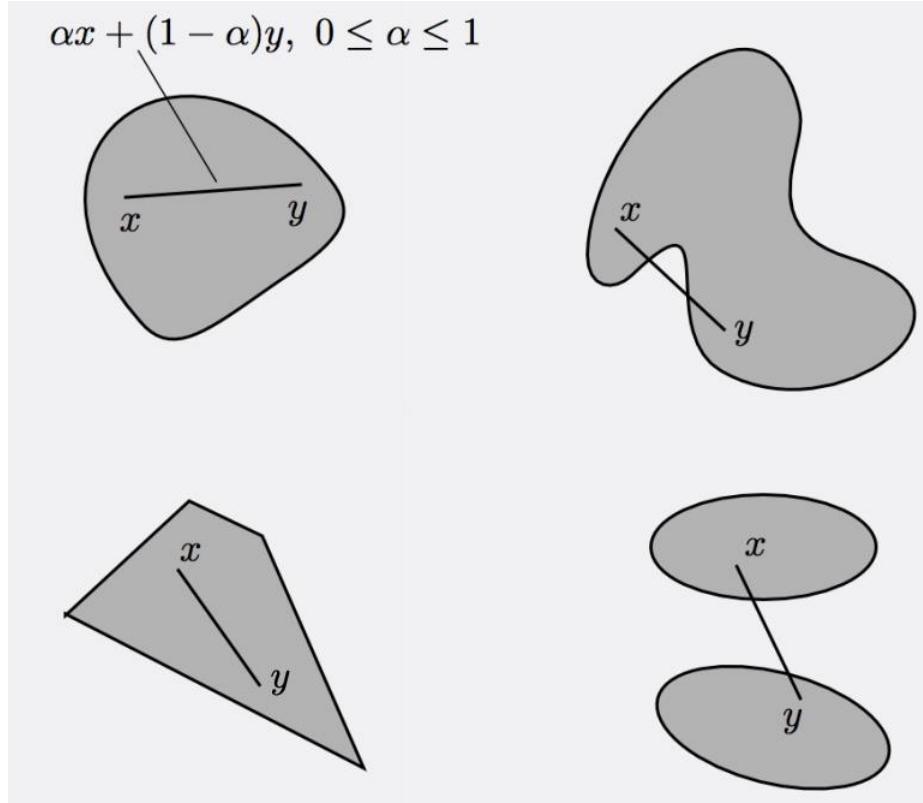


凸集

- \mathbb{R}^n 的子集 C 叫做凸集 (convex set) , 如果满足:

$$\alpha \mathbf{x} + (1 - \alpha) \mathbf{y} \in C$$

$$\forall \alpha \in [0,1] \quad \forall \mathbf{x}, \mathbf{y} \in C$$

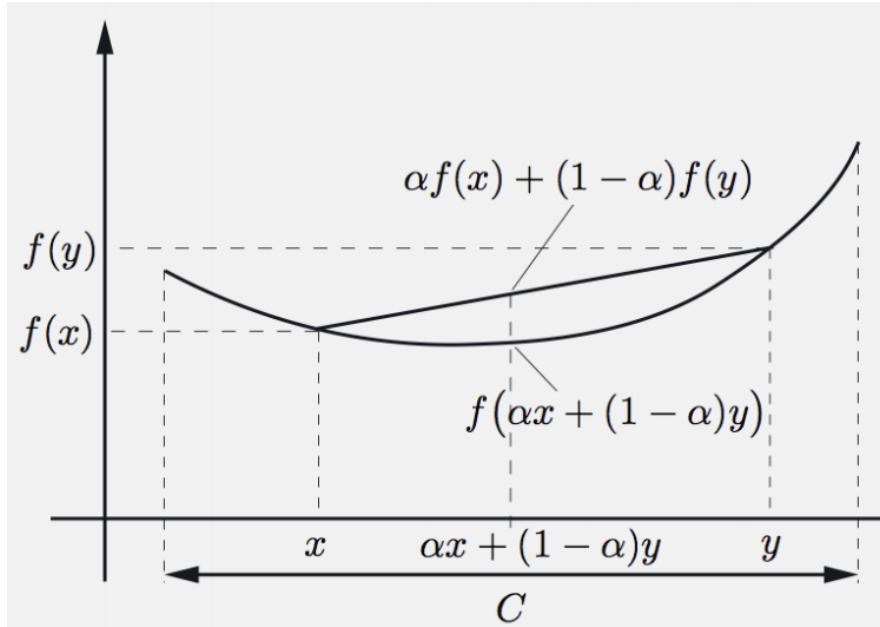


凸函数

- 函数 $f: C \rightarrow \mathbb{R}$ 为凸函数 (convex) , 如果满足:

$$\begin{aligned} & f(\alpha x + (1 - \alpha)y) \\ & \leq \alpha f(x) + (1 - \alpha)f(y) \\ & \quad \forall \alpha \in [0,1] \\ & \quad \forall x, y \in C \end{aligned}$$

- 如果不等式在 $\alpha \in (0,1)$ 和 $x \neq y$ 条件下是严格的, 那么 f 被称为严格凸

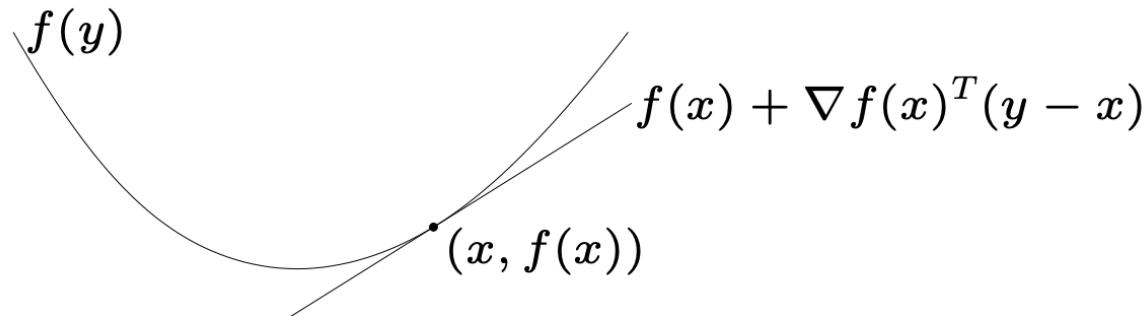


一阶特征条件

- f 是凸函数，当且仅当

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{y} - \mathbf{x}) \quad \forall \mathbf{x}, \mathbf{y} \in C$$

- 如果等号不成立，那么 f 是严格凸的



二阶特征条件

- f 是凸函数，当且仅当：

$$\nabla^2 f(\mathbf{x}) \succeq 0 \quad \forall \mathbf{x} \in C$$

- f 是严格凸的，当且仅当：

$$\nabla^2 f(\mathbf{x}) > 0 \quad \forall \mathbf{x} \in C$$

常见凸集和非凸集

- 凸集

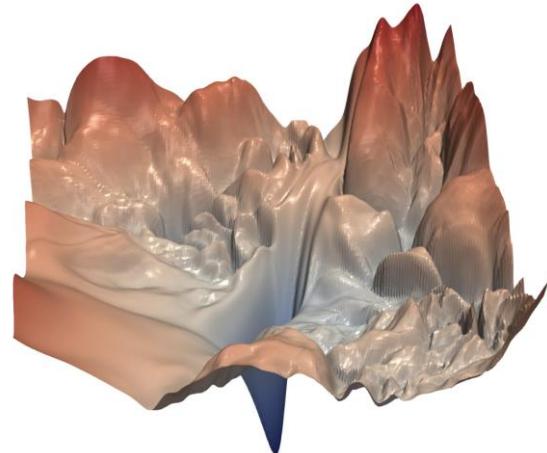
- 线性回归 $f(\mathbf{x}) = \|\mathbf{Wx} - \mathbf{b}\|_2^2$

$$\nabla f(\mathbf{x}) = 2\mathbf{W}^T(\mathbf{Wx} - \mathbf{b}), \nabla^2 f(\mathbf{x}) = 2\mathbf{W}^T$$

- Softmax 回归

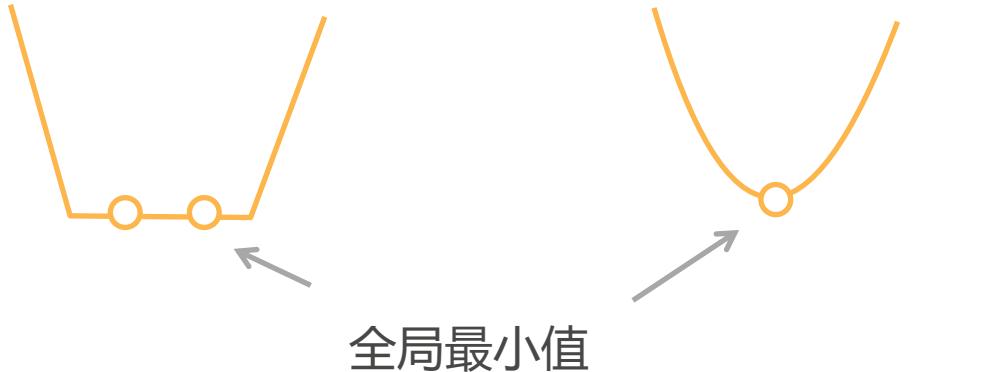
- 非凸集

- 多层感知器
 - 卷积神经网络
 - 循环神经网络



凸优化

- 如果 f 是凸函数，而且 C 是凸集，则该问题称为凸问题：
 - 任何局部最小值都是全局最小值
 - 如果严格凸成立，则为唯一的全局最小值



凸优化证明

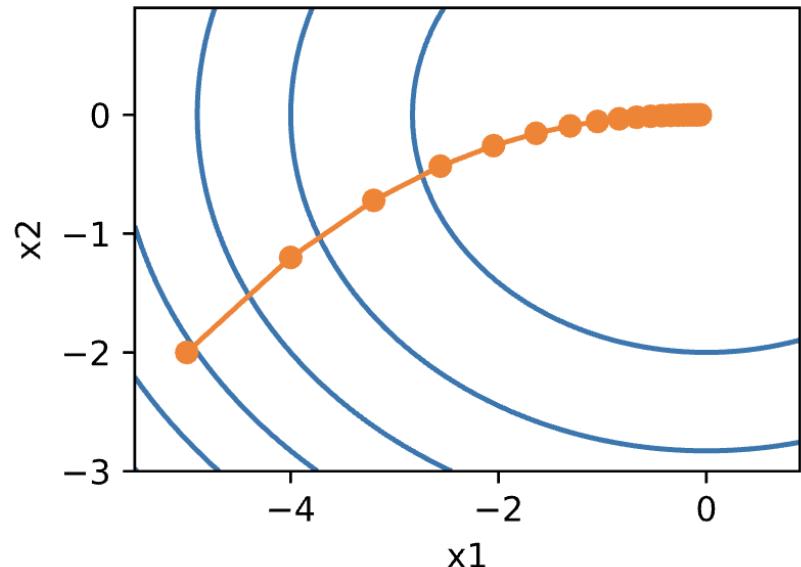
- 局部最小值为 \mathbf{x} , 假设存在全局最小值 \mathbf{y} :
 - 选择一个 $\alpha \leq 1 - \varepsilon/|\mathbf{x} + \mathbf{y}|$ 和一个 $\mathbf{z} = \alpha\mathbf{x} + (1 - \alpha)\mathbf{y}$
 - 然后 $\|\mathbf{x} - \mathbf{z}\| = (1 - \alpha) \|\mathbf{x} + \mathbf{y}\| \leq \varepsilon$
 - 由于 \mathbf{y} 是全局最小值, 所以 $f(\mathbf{y}) < f(\mathbf{x})$
$$f(\mathbf{z}) \leq \alpha f(\mathbf{x}) + (1 - \alpha)f(\mathbf{z}) < \alpha f(\mathbf{x}) + (1 - \alpha)f(\mathbf{x}) = f(\mathbf{x})$$
 - 它与局部最小值 \mathbf{x} 相矛盾

梯度下降



算法

- 选择初始的 \mathbf{x}_0
- 在时间 $t = 1, \dots, T$
$$\mathbf{x}_t = \mathbf{x}_{t-1} - \eta \nabla f(\mathbf{x}_{t-1})$$
- η 被称为学习率



学习率的选择

- 在 $\|\Delta\| < \varepsilon$ 的条件下，对于任何 f ，由泰勒扩展：

$$f(\mathbf{x} + \Delta) \approx f(\mathbf{x}) + \Delta^T \nabla f(\mathbf{x})$$

- 选择足够小的学习率 $\eta \leq \varepsilon / \|\nabla f(\mathbf{x})\|$

$$\| -\eta \nabla f(\mathbf{x}) \| \leq \varepsilon$$

$$f(\mathbf{x} - \eta \nabla f(\mathbf{x})) \approx f(\mathbf{x}) - \eta \|\nabla f(\mathbf{x})\|^2 \leq f(\mathbf{x})$$

收敛率

- 假设 f 是凸的，并且其梯度是 Lipschitz 连续的常数 L .

$$\| \nabla f(\mathbf{x}) - \nabla f(\mathbf{y}) \| \leq L \| \mathbf{x} - \mathbf{y} \|$$

渐变不会发生
显着变化

- 如果使用学习率 $\eta \leq 1/L$, 经过 T 步

$$f(\mathbf{x}_T) - f(\mathbf{x}^*) \leq \frac{\| \mathbf{x}_0 - \mathbf{x}^* \|^2}{2\eta T}$$

- 收敛率 $O(1/T)$
- 要获得 $f(\mathbf{x}_T) - f(\mathbf{x}^*) \leq \epsilon$, 需要 $O(1/\epsilon)$ 次迭代

收敛率证明

- 演变 L -Lipschitz 意味着

$$f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{y} - \mathbf{x}) + \frac{L}{2} \|\mathbf{y} - \mathbf{x}\|^2$$

- 插入 $\mathbf{y} = \mathbf{x} - \eta \nabla f(\mathbf{x})$

$$f(\mathbf{y}) \leq f(\mathbf{x}) - \left(1 - \frac{L\eta}{2}\right) \eta \|\nabla f(\mathbf{x})\|^2$$

- 采用 $0 < \eta \leq 1/L$

$$f(\mathbf{y}) \leq f(\mathbf{x}) - \frac{\eta}{2} \|\nabla f(\mathbf{x})\|^2$$

f 每次都减小

收敛率证明 II

- 根据凸性

$$f(\mathbf{x}) \leq f(\mathbf{x}^*) + \nabla f(\mathbf{x})^T (\mathbf{x} - \mathbf{x}^*)$$

- 插入

$$f(\mathbf{y}) \leq f(\mathbf{x}) - \frac{\eta}{2} \|\nabla f(\mathbf{x})\|^2$$

$$f(\mathbf{y}) \leq f(\mathbf{x}^*) + \nabla f(\mathbf{x})^T (\mathbf{x} - \mathbf{x}^*) - \frac{\eta}{2} \|\nabla f(\mathbf{x})\|^2$$

$$f(\mathbf{y}) - f(\mathbf{x}^*) \leq (2\eta \nabla f(\mathbf{x})^T (\mathbf{x} - \mathbf{x}^*) - \eta^2 \|\nabla f(\mathbf{x})\|^2) / 2\eta$$

$$= (\|\mathbf{x} - \mathbf{x}^*\|^2 + 2\eta \nabla f(\mathbf{x})^T (\mathbf{x} - \mathbf{x}^*) - \eta^2 \|\nabla f(\mathbf{x})\|^2 - \|\mathbf{x} - \mathbf{x}^*\|^2) / 2\eta$$

$$= (\|\mathbf{x} - \mathbf{x}^*\|^2 - \|\mathbf{x} - \eta \nabla f(\mathbf{x}) - \mathbf{x}^*\|^2) / 2\eta$$

$$= (\|\mathbf{x} - \mathbf{x}^*\|^2 - \|\mathbf{y} - \mathbf{x}^*\|^2) / 2\eta$$

收敛率证明 III

- 所有 T 步骤总和：

$$\begin{aligned}\sum_{t=1}^T f(\mathbf{x}_t) - f(\mathbf{x}^*) &\leq \sum_{t=1}^T (\|\mathbf{x}_{t-1} - \mathbf{x}^*\|^2 - \|\mathbf{x}_t - \mathbf{x}^*\|^2)/2\eta \\ &= (\|\mathbf{x}_0 - \mathbf{x}^*\|^2 - \|\mathbf{x}_T - \mathbf{x}^*\|^2)/2\eta \leq \|\mathbf{x}_0 - \mathbf{x}^*\|^2/2\eta\end{aligned}$$

- f 每次都在减少：

$$f(\mathbf{x}_T) - f(\mathbf{x}^*) \leq \frac{1}{T} \sum_{t=1}^T f(\mathbf{x}_t) - f(\mathbf{x}^*) \leq \frac{\|\mathbf{x}_0 - \mathbf{x}^*\|^2}{2\eta T}$$

深度学习应用

- f 是所有训练数据的损失之和的函数， x 是可学习的参数

$$f(\mathbf{x}) = \frac{1}{n} \sum_{i=0}^n \ell_i(\mathbf{x})$$

$\ell_i(\mathbf{x})$ 是第 i 个样本的损失

- f 往往不是凸函数，所以不能应用之前的收敛性分析

随机梯度下降 (SGD)



1000 新加坡元 (SGD)
~740 USD

算法

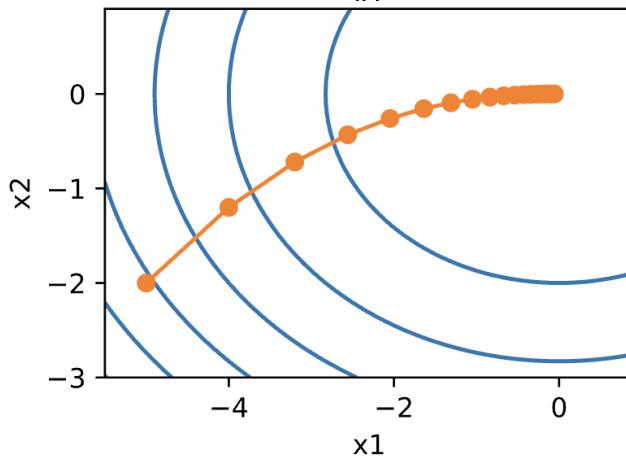
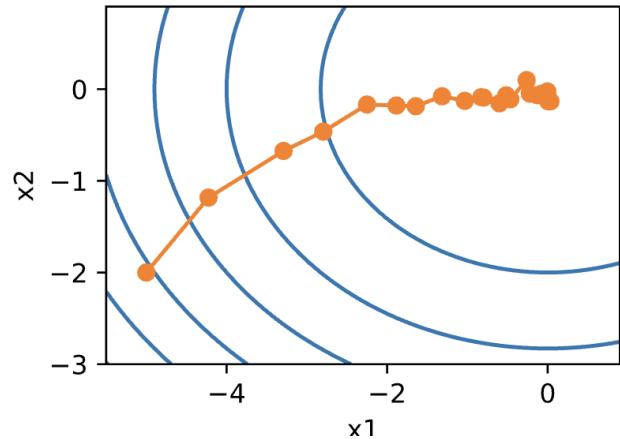
- 在时间 t , 样本示例 t_i

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \eta_t \nabla \ell_{t_i}(\mathbf{x}_{t-1})$$

- 与梯度下降 (GD) 相比

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \eta \nabla f(\mathbf{x}_{t-1})$$

$$f(\mathbf{x}) = \frac{1}{n} \sum_{i=0}^n \ell_i(\mathbf{x})$$



例子

- 在时间 t 举例的两个规则
 - 随机 (random) 规则：随机均匀选择 $i_t \in \{1, \dots, n\}$
 - 循环 (cyclic) 规则：选择 $i_t = 1, 2, \dots, n, 1, 2, \dots, n$
 - 通常称为增量梯度下降
- 随机规则在实践中更为常见

$$\mathbb{E}[\nabla \ell_{t_i}(\mathbf{x})] = \mathbb{E}[\nabla f(\mathbf{x})]$$

- 对梯度的无偏见估计

收敛率

- 假设 f 是凸的，并且 η_t 逐步减小，例如 $\eta_t = O(1/t)$,

$$\mathbb{E}[f(\mathbf{x}_T)] - f(\mathbf{x}^*) = O(1/\sqrt{T})$$

- 在相同的假设下，对于梯度下降

$$f(\mathbf{x}_T) - f(\mathbf{x}^*) = O(1/\sqrt{T})$$

- 假设梯度 L -Lipschitz 并固定 η

$$f(\mathbf{x}_T) - f(\mathbf{x}^*) = O(1/T)$$

- 对 SGD 没有改善

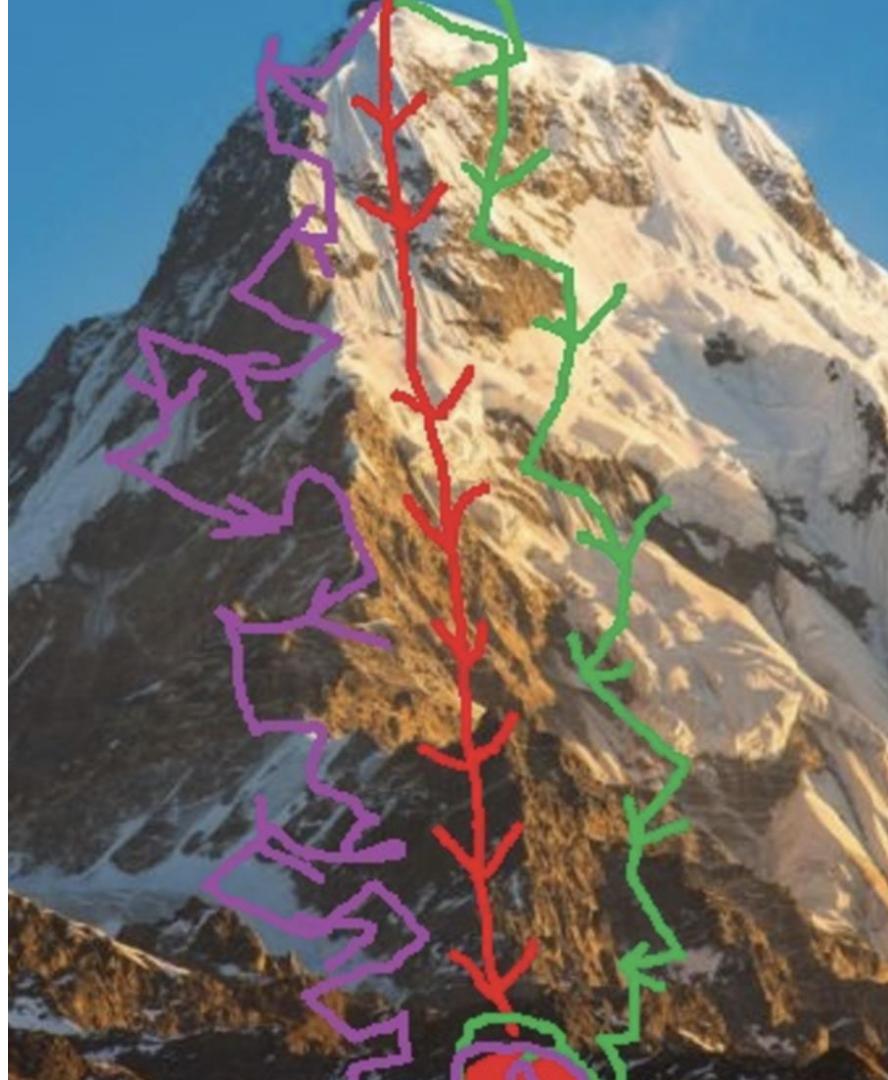
实际应用

- 我们不会如此戏剧性地降低学习率
 - 我们不关心优化到高精度
- 尽管收敛速度较慢，但在每次迭代中，SGD 计算梯度的速度要快于 GD
 - 特别适用于复杂模型和大型数据集的深度学习

代码...

小批量随机梯度下降 (mini-batch SGD)

- Batch Gradient Descent
- Mini-batch Gradient Descent
- Stochastic Gradient Descent



算法

- 在时间 t , 随机采样子集 $I_t \subset \{1, \dots, n\}$ 满足 $|I_t| = b$

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \frac{\eta_t}{b} \sum_{i \in I_t} \nabla \ell_i(\mathbf{x}_{t-1})$$

- 这是一个无偏见的估计

$$\mathbb{E}\left[\frac{1}{b} \sum_{i \in I_t} \nabla \ell_i(\mathbf{x})\right] = \nabla f(\mathbf{x})$$

- 与 SGD 相比, 差异减少 $1/b$

代码...

总结

- 优化问题
 - 局部最小值和全局最小值
 - 凸集和凸函数
 - 凸优化证明
- 梯度下降
 - 学习率
 - 收敛率证明
 - 随机梯度下降
 - 小批量随机梯度下降