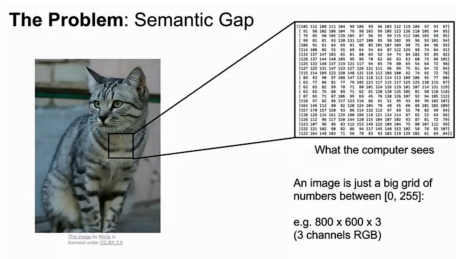
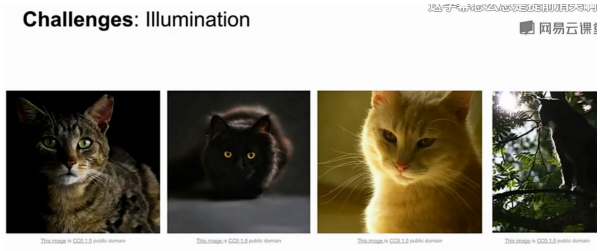
Lecture2 Image Classification

2022年4月2日　17:13

一、图像分类

- 图像分类是计算机视觉领域的基础任务， 目标是给定输入的图片，为其确定分类（标签）来区分图像。图像分类是计算机视觉的核心基本任务，目标检测、语义分割等是在分类任务的基础上进行的
- 图像分类任务的难点：语义鸿沟，机器得到图像的像素矩阵与图片所含标签概念语义存在巨大差别，输入图像的待分类目标在视角、光照、形态、遮挡、背景等因素的变化下会导致整个像素矩阵的变化，分类器需要这种情况下的鲁棒性





二、数据驱动方法

- 曾经基于规则的图像分类方法（如边缘检测 -> 对边缘属性应用精心设计的规则进行分类）由于依赖人工规则，难以保证准确度，且识别特定类别图像的规则不具有可迁移性，难以应用于其他类别图像
- 为了可以识别多种类别的图像，使用数据驱动的方法，不再构造具体的分类规则，而是

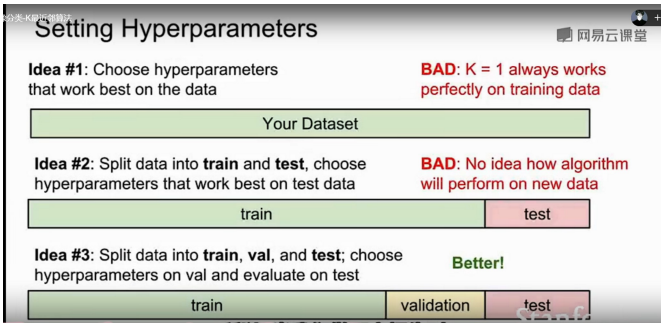
  1）构造图像-标签的数据集 2）使用机器学习方法从数据集训练分类器模型(Train) 3）将模型应用于新数据，评估性能(Evaluate)

三、图像分类流程与调整超参数

- 超参数（Hyperparameters）是分类器不能从数据中学习得到参数，其设置是依赖于具体问题的，像KNN中K和距离度量的选择，需要前提为算法做出选择，尝试哪个超参数最合适。
- 超参数调整的几种策略：

  1）不划分数据集/划分数据集为训练集、测试集，则分类器模型超参数仅针对其在训练集/测试集上的效果进行调整，不能说明分类器的泛化能力

  2）划分数据集为训练集、验证集、测试集，其中训练集用于分类器的训练阶段，验证集用于在分类器的训练中评估分类器效果，作为调整超参数的依据，测试集不参与训练/评估，在完成训练后仅用于分类器测试，可以评测分类器的泛化能力



- 3)k-折交叉验证

  训练数据分为k段，轮流选取其中一段为验证集，其他为训练集，并分别训练一个分类器，根据k个分类器的平均性能调整超参数，但此方法会使训练时间常数倍增加，不适用于算力要求高的场景

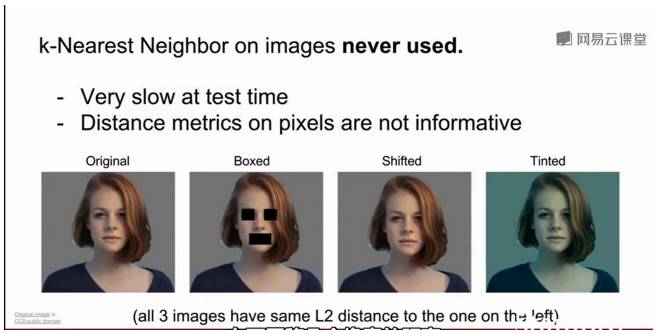
四、分类器举例1：k邻近算法（K-Nearest Neighbor）

- 主要思想是 1）训练阶段简单存储所有数据 2）预测阶段将输入图像与训练集中所有图像作像素矩阵的比较，最相似的K个图像中对应标签占优的作为预测标签
- KNN在实际应用中的问题：

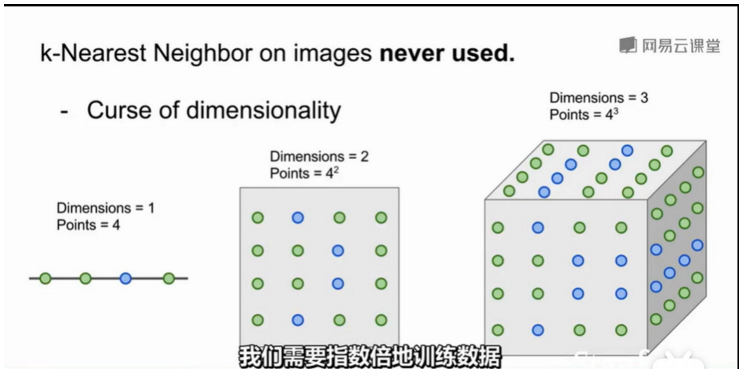
  1）预测的时间复杂度是训练集规模的线性，即训练快预测慢，不符合实际情况（训练往往集中于高算力环境，而实际部署时可能是在移动设备等边缘设备）

  2）KNN使用原始像素矩阵距离进行相似度比较，但像L2距离或者L1距离这种向量化的距离函数不太适合衡量图像间的相似性，

如下各图的差距很明显，但L2距离确是相同的，不能有效的表示出图片之间的差异。



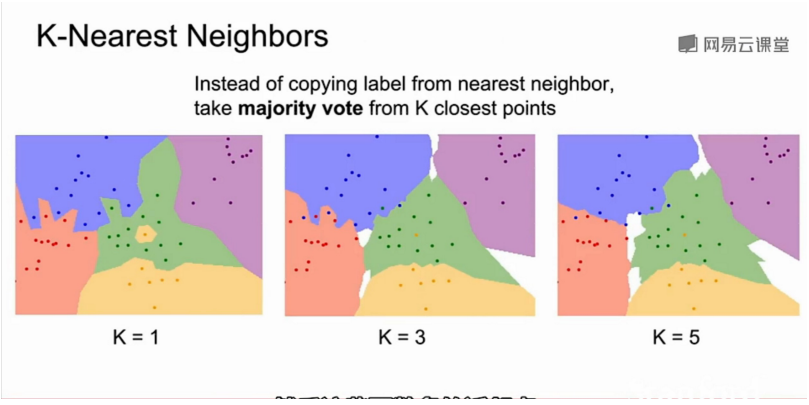
3）当数据稀疏时，最近邻点的实际距离可能很远，因此KNN要求训练数据能密集地分布在空间中，才能达到较好效果，因此数据维度增加时，需要指数倍地训练数据



- 具体而言，KNN的关键在于确定像素矩阵度量方法，以及最近邻数量K这2个超参数

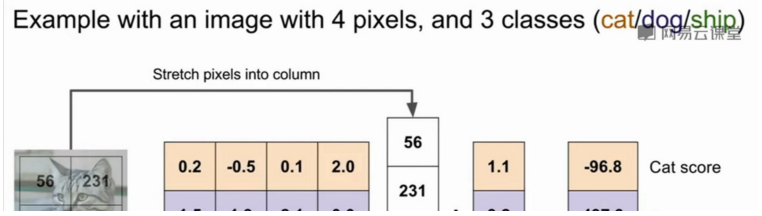
❖ L1\L2距离：站在机器的视角，图像的像素矩阵看做高维张量。L1\L2距离为张量曼哈顿/欧式距离，其中转动坐标轴时将会改变矩阵之间的L1距离，L2不会。当张量各维度为有实际意义的特征数值时，L1可能更好

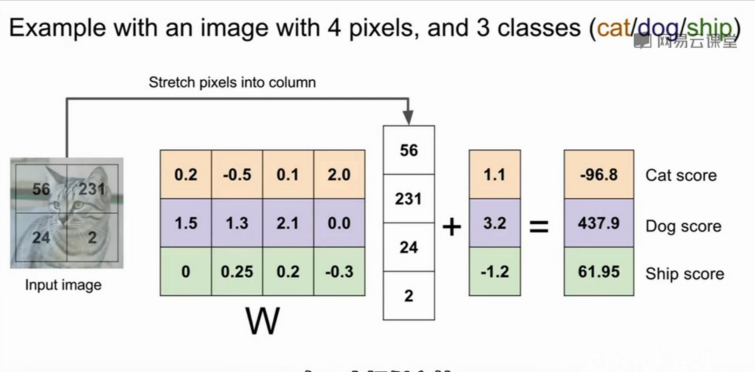
- 实际问题中，需要根据模型性能调整KNN中K的值，当K值过小时，KNN容易受数据集中的噪声影响，如K=1时若存在错误标签，则数据空间中此错误数据附近的数据点均会被错误分类



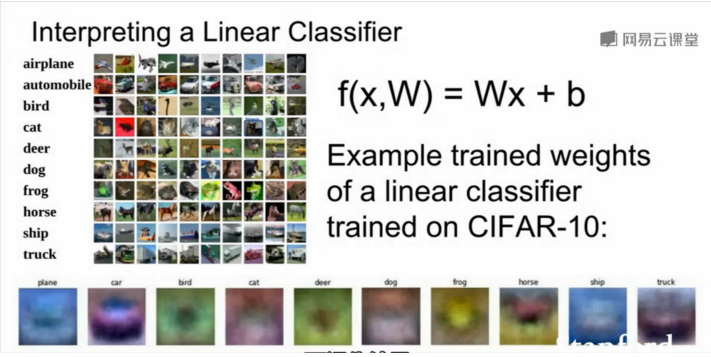
五、分类器举例2：线性分类

- 线性分类器需要学习的模型参数为权重矩阵W与偏差向量b,将输入图像矩阵展成（$W*H*N\_channel,1$）的向量x，后计算$Wx+b$得到（$N\_class,1$）的输出，代表对应各类别的分数





- 因此权重矩阵中每一行向量可视为一种类别的template，其与图像展成列向量的点积为此类别的得分，举例：下图为权重矩阵行向量还原为图像结果
- 另一方面，每张图片类比为高维空间中单个点（例如CIFAR-10中的每个图像是3072维空间的点），线性分类器在这些线性决策边界上尝试画一个线性分类面来划分各个类别，然而实际数据集往往是线性不可分的



Lecture3 Loss Function and optimization

2022年4月8日　　15:14

损失函数：衡量模型参数W的好坏的函数（如Softmax, Cross Entropy）

对于数据集$\{(x_i, y_i)\}_{i=1}^N$，损失记为$loss = \frac{1}{N}\sum_i L_i(f(x_i, W), y_i)$

最优化：寻找能使得损失函数值最小化的参数W的过程（如BGD\SGD\MBGD）

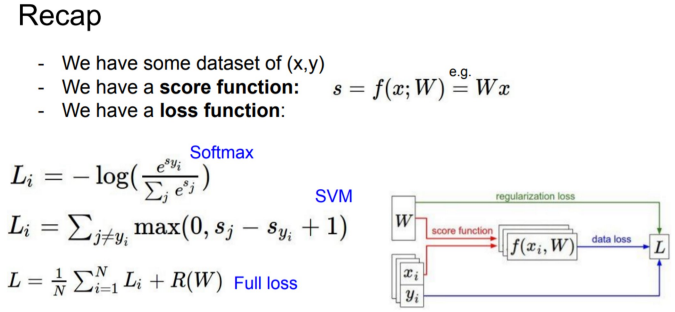
一、损失函数举例：Multiclass SVM Loss(Hinge Loss)、softmax Loss

- 给定图像i,记其通过分类器f得到各类别的得分向量为s，s_{idx}为第idx个分量，y_i为真实标签值，则图像i的Multiclass SVM Loss为

  $$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + \Delta)$$

- 即对于每一非真实标签得分s_j，$s_{y_i} < s_j + \Delta$时，损失函数增加$s_j - s_{y_i} + \Delta$
- 而在softmax Loss中，对s_j分量指数化后进行归一化得到分类概率，损失函数为

  $-logP_{y_i} = -\log(\frac{e^{sy_i}}{\sum_j e^{s_j}})$，即真实标签独热编码后与分类器输出$log_softmax$后的交叉熵（*Cross-Entropy)*

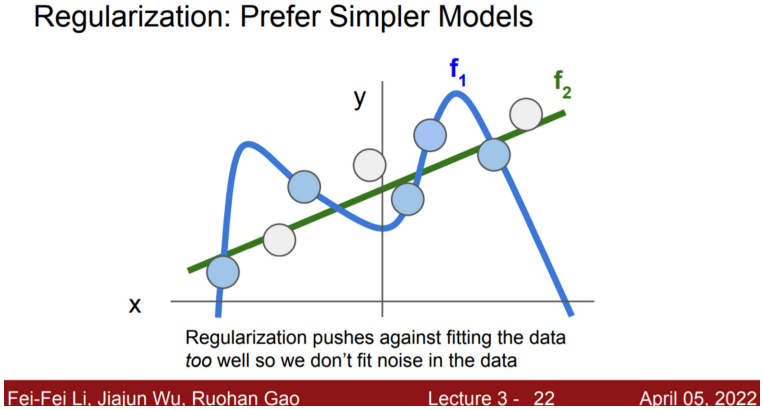


❖ SVM loss与softmax loss对比：

对于得分向量，SVM分类器将它们看做是分类评分，真实分类分值比其他分类的分值高出至少一个边界值，才不会产生损失（且真实分类分值超过阈值后再增长不影响loss）。 Softmax分类器将这些数值看做是每个分类没有归一化的对数概率，所以相比较而言，SVM分类器难以评分值给出直观解释。

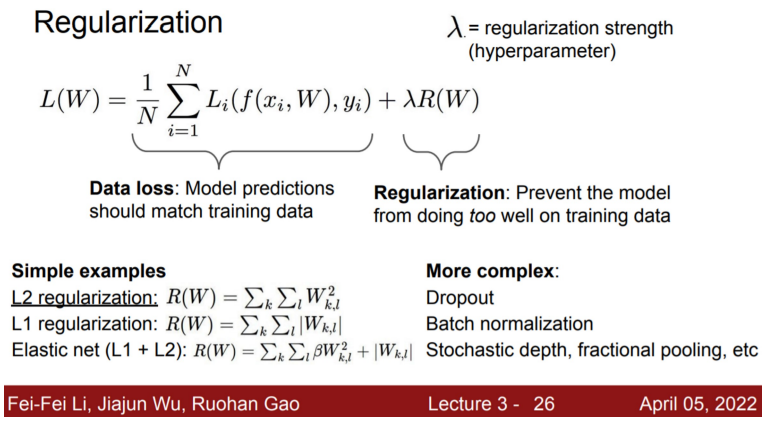
二、正则化

- 在损失函数中加上正则项，称之为正则化（Regularize）。目的：防止模型过拟合，仅考虑分类器在训练数据上的损失时，模型可能会过拟合数据中的噪点，分类决策面因为权重参数过大而陡峭。原理：在损失函数上加上某些规则（限制），缩小解空间，从而减少求出过拟合解的可能性。



- 举例：L1，L2正则项，即损失函数中加上加上模型权重向量的L1/L2范数，并设置一个正则化常系数λ。正则

化后拟合过程中通常都倾向于让权值尽可能小，参数足够小，测试集数据偏移得多一点也不会对结果造成较大影响，抗扰动能力强



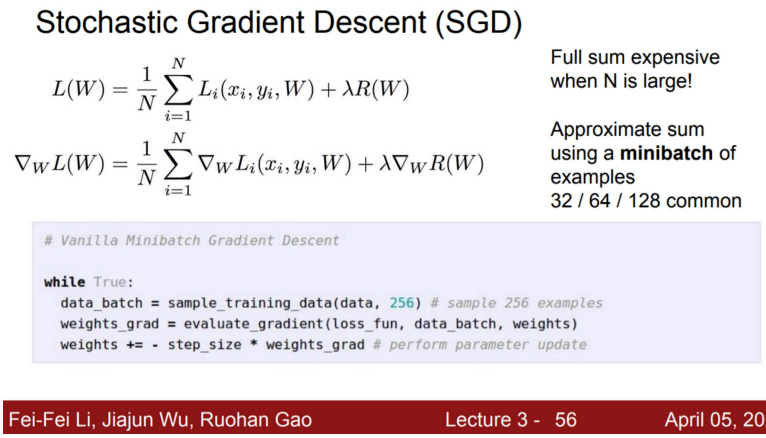
三、参数最优化

- 最优化（Optimization）是寻找能使得损失函数值最小化的参数W的过程，分类器训练中，优化器是一大关键部分，介绍了几种优化器：

1）随即搜索：效果差

2）随机梯度下降 Stochastic Gradient Descent (SGD)

损失函数$L(W)$关于参数向量W的偏导组成的向量，即梯度，是损失函数增长最快的方向，负梯度指向损失函数下降最快的方向。梯度下降法通过沿梯度$\nabla_W L(W)$ 的相反方向更新 W 。学习率η 决定了每一时刻的更新步长

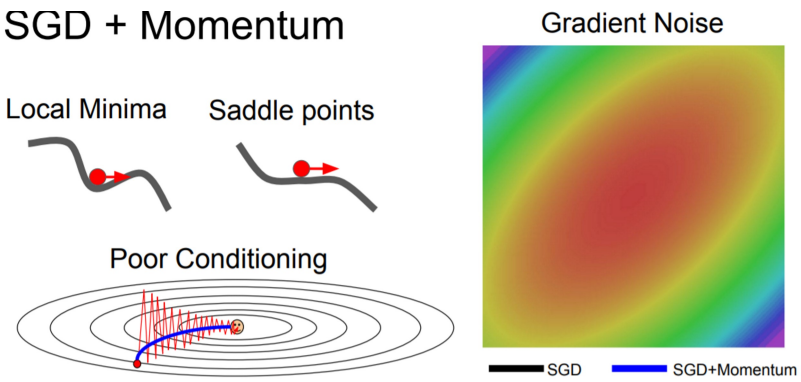


- 数据集规模N很大时，每次参数更新需要在所有数据上计算损失函数值，过于耗时，因此固定批次大小，每次更新参数取一个小批次*(mini-batch)*，取平均梯度代替整体梯度
- 朴素SGD存在的问题

  ①在局部最优点或鞍点时，梯度为0，无法继续更新参数W

  ②在梯度变化较大时参数易震荡 梯度变化很小时参数更新的幅度缺基本不变，难以快速收敛

  ③梯度依据随机小批次数据计算，而小批次数据可能包含噪点，需合理选择batch_size



3）带动量的SGD（SGDM）

- 梯度下降过程加入“惯性”，SGD中按梯度方向下降改为按一阶动量（m）方向下降，而一阶动量是各个时刻梯度方向的指数移动平均值

  即 $m_{t+1} = \rho m_t + \nabla f(w_t)$

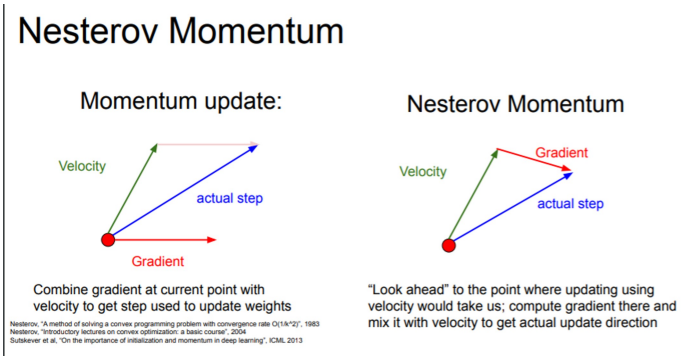
  $w_{t+1} = w_t - \alpha m_{t+1}$
- SGDM可以缓解SGD存在的问题中的①、②

  其中ρ可视为动量的“摩擦系数”的常用经验值为0.9

4）NAG(Nesterov Accelerated Gradient)

- 在SGD、SGD-M的基础上的进一步改进，唯一不同是梯度的计算方式，NAG认为，即使当前的梯度为0，由于动量的存在，仍会继续更新参数w。当前点w的梯度是不太有意义的。假设仅靠动量滚到下一个点的w的梯度方向才是决定当前梯度的重要因素
- $m_{t+1} = \rho m_t - \alpha \nabla f(w_t + \rho m_t)$

  $w_{t+1} = w_t + m_{t+1}$
- 引入动量后不停留在当前参数位置，跟着一阶动量先走一步(Look ahead)

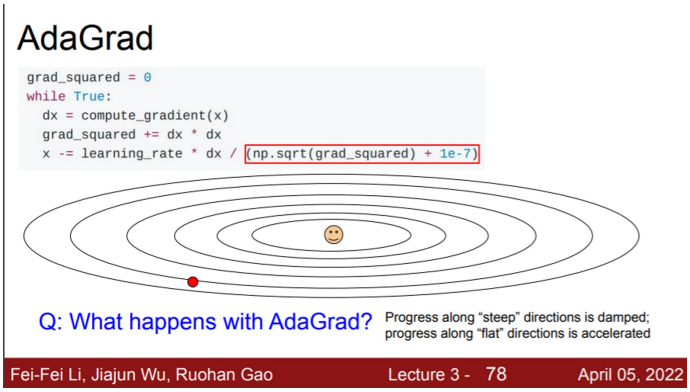


5）AdaGrad

- 在一阶动量(m)的基础上引入二阶动量（V），实现学习率衰减（自适应学习率）
- 参数经常更新，说明已经积累了大量关于它的知识，不希望被单个样本影响太大，希望学习速率慢一些；对于较少更新的参数，希望能从每个偶然出现的样本身上多学一些，即学习率大一些
- 二阶动量就是用于度量参数更新频率的，二阶动量即各参数历史梯度值的平方和

  $V_t = \sum_{j=1}^{t} g_j \odot g_j$，（$\odot$为逐元素相乘）

  $w_{t+1} = w_t - \frac{\alpha}{\sqrt{V_t}} \cdot m_{t+1}$ （$\sqrt{V_t}$为逐元素平方根）
- 由于针对各参数有着不同的实际学习率$\frac{\alpha}{\sqrt{V_t}}$，适用于稀疏数据。同时开启了自适应学习率算法的里程，缺点是学习率单调递减，可能出现训练早停导致模型欠拟合



- RMSProp: “Leaky AdaGrad”:为防止AdaGrad中实际学习率$\frac{\alpha}{\sqrt{V_t}}$递减至0停止更新参数，在计算二阶动量时加入历史二阶动量的衰减系数，避免了二阶动量持续累积

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

6）Adam

- Adam=Adaptive + Momentum，结合了动量梯度下降与自适应学习率（如SGDM+RMSProp）、
  - Adam (full form)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

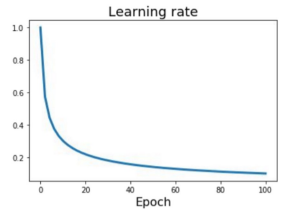
Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

Fei-Fei Li, Jiajun Wu, Ruohan Gao Lecture 3 - 86 April 05, 2022

- Adam衰减率参数β_1, β_2常用经验值为0.9， 0.999，因此初始几次更新时一阶/二阶动量会与梯度大小有较大的偏差，因此乘上偏差校正系数，保证一/二阶动量是各个参数梯度变量的一阶矩/二阶矩的无差估计

四、学习率调节策略

Learning Rate Decay



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: $\alpha_t = \frac{1}{2}\alpha_0 (1 + \cos(t\pi/T))$

Linear: $\alpha_t = \alpha_0(1 - t/T)$

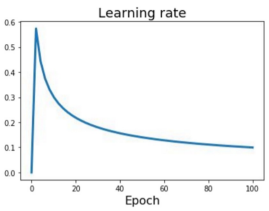
Inverse sqrt: $\alpha_t = \alpha_0/\sqrt{t}$

α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs

Vaswani et al, "Attention is all you need", NIPS 2017

Fei-Fei Li, Jiajun Wu, Ruohan Gao Lecture 3 - 95 April 05, 2022

Learning Rate Decay: Linear Warmup



High initial learning rates can make loss explode; linearly increasing learning rate from 0 over the first ~5,000 iterations can prevent this.

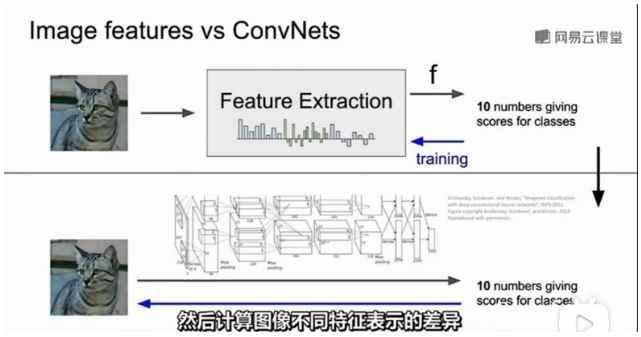
Empirical rule of thumb: If you increase the batch size by N, also scale the initial learning rate by N

Goyal et al, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour", arXiv 2017

Fei-Fei Li, Jiajun Wu, Ruohan Gao Lecture 3 - 96 April 05, 2022

五、图像特征

- 原始像素值矩阵输入分类器，表现并不好。所以在深度网络大规模应用前，首先计算图片的各种代表特征，如颜色直方图、方向梯度直方图，不同的特征向量合到一起，输入分类器。



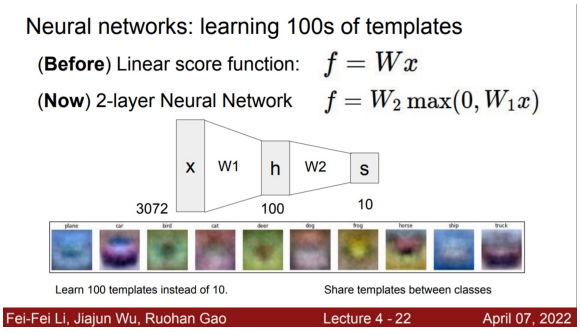
- 卷积神经网络和这些深度神经网络并非人工提取特征，而是直接从数据中学习特征，所以模型进行图像到标签的端到端训练，而不是仅学习特性向量分类器的权重。

Lecture4 Backpropagation and Neural Networks

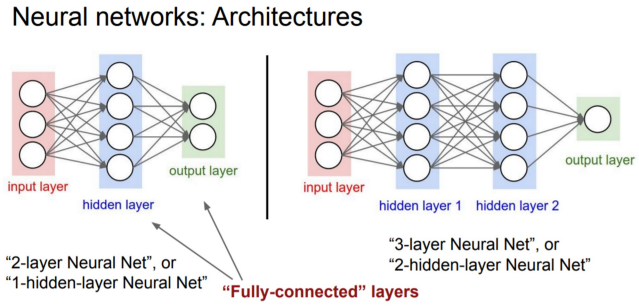
2022年4月8日　　15:14

一、神经网络简介

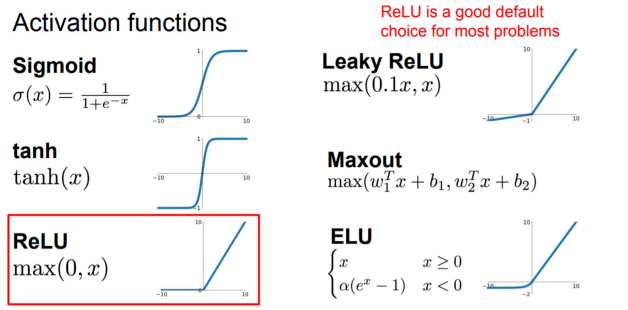
- 神经网络（Neural networks）从函数上看是由简单函数构成的一组函数，其中引入了非线性激活函数形成表达能力更强的非线性函数，以解决数据线性不可分的问题



- 简单的两层神经网络中有两个线性层，线性层实际上进行矩阵乘法，称之为两个全连接层。整体也称之为单隐藏层神经网络（关注有多少隐藏层）

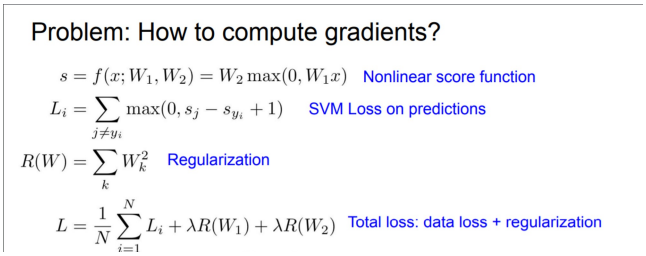


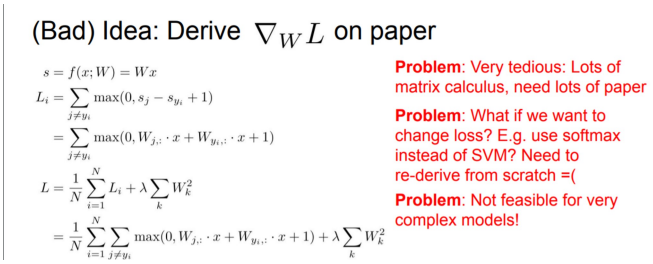
- 常见激活函数



二、反向传播

优化器进行参数更新前，关键的一步是求损失函数关于权重参数的梯度，使用数值上的差分法计算量大，有误差，因此不可行。需要损失函数的参数梯度的精确解析解，手动求梯度表达式困难，且不可迁移。因此需要分解复杂的表达式，使用链式法则自动求导





$R(W) = \sum_k W_k^2$ Regularization

$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$ Total loss: data loss + regularization

If we can compute $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}$ then we can learn W_1 and W_2

$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$

$= \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i} \max(0, W_{j,:}\cdot x + W_{y_i,:}\cdot x + 1) + \lambda \sum_k W_k^2$

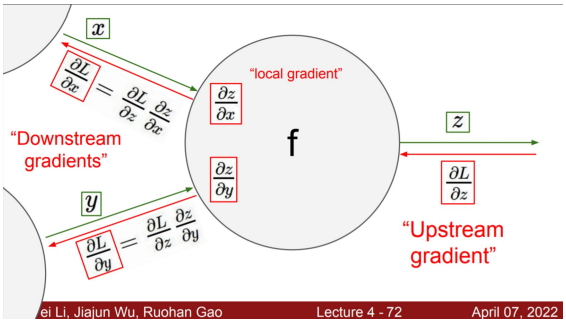
$\nabla_W L = \nabla_W \left(\frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i} \max(0, W_{j,:}\cdot x + W_{y_i,:}\cdot x + 1) + \lambda \sum_k W_k^2 \right)$

re-derive from scratch =(

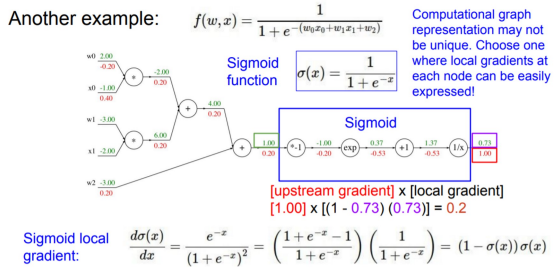
Problem: Not feasible for very complex models!

1. 计算图/计算链

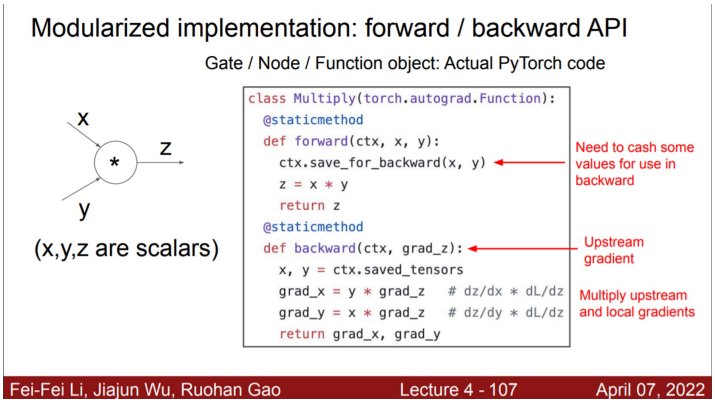
- 将损失函数计算流程分解为若干节点组成有向图，每个节点代表一次计算，入边为输入变量，出边为输出向量，计算梯度时，从最终输出层开始，首先得到当前节点关于输出z的上游梯度，再计算z关于节点输入x, y的本地梯度，二者通过链式法则相乘得到输出L关于$[x, y]$的梯度，此梯度作为后续节点的上游梯度，因此反向传播是按计算图的逆拓扑序进行的



- 计算图中节点有多个输出时，在反向传播中应累加多个节点传来的上游梯度（多元函数的偏导法则）
- 一个计算图实例，将几个简单运算节点合并为Sigmoid函数节点

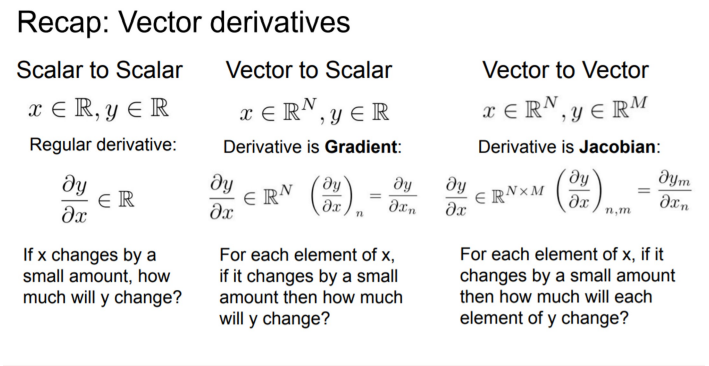


- 计算图将复杂表达式转换为一系列简单运算的序列，因此众多深度学习库将运算节点进行模块化设计，提取公共的前向/反向传播的接口

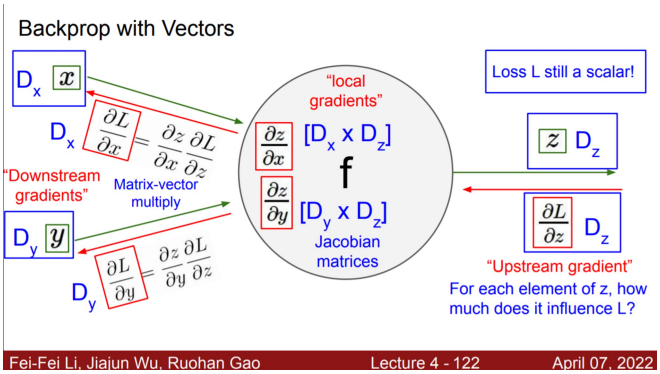


2. 向量求导

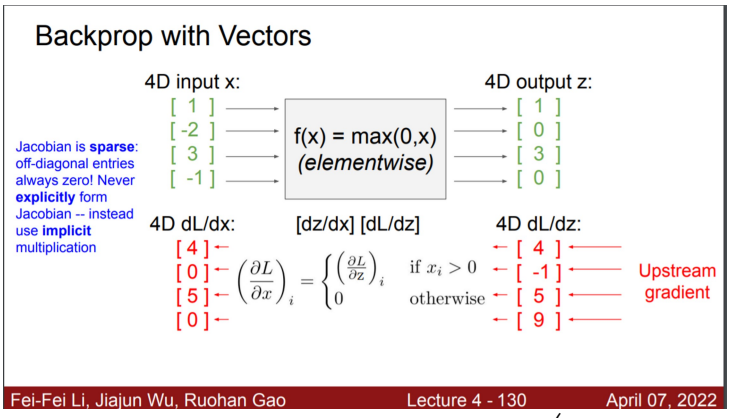
- 基本求导法则



- 损失函数输出为标量，因此所有中间变量/输入的梯度与自身的向量形状相同

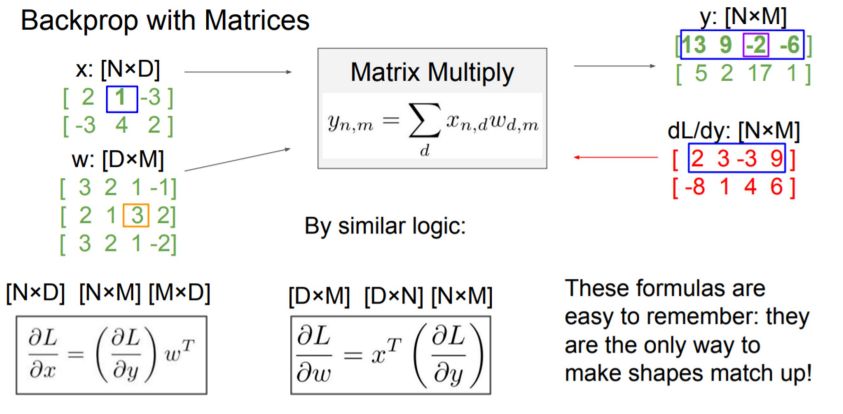


- 本地梯度为雅可比矩阵，而雅可比矩阵常为过大的稀疏阵，因此通常不计算整个雅可比矩阵，如max(vec, 0)中取max操作为逐元素独立计算，因此雅可比矩阵为对角阵，实际上x的梯度等于对上游$\frac{dL}{dz}$进行max操作，而不用真正花费空间创建中间的雅可比矩阵



- 二例：矩阵相乘求导，对于矩阵$A_{n\times m}, \frac{dL}{dA} = \begin{pmatrix} \frac{dL}{dA_{1,1}} & \cdots & \frac{dL}{dA_{1,m}} \\ \vdots & \ddots & \vdots \\ \frac{dL}{dA_{n,1}} & \cdots & \frac{dL}{dA_{n,m}} \end{pmatrix}$

通过逐元素分析可知$y = A \times B$ 时$\frac{dL}{dA} = \frac{dL}{dy} \times B^T$，$\frac{dL}{\mathrm{dB}} = A^T \times \frac{dL}{dy}$

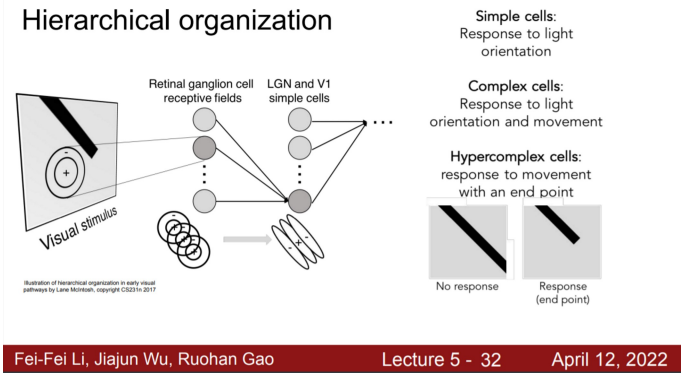


Lecture5 Convlution Neural Network

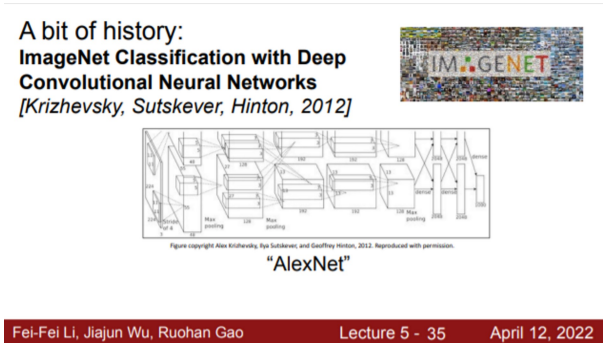
2022年4月8日　　15:14

一、卷积神经网络简介

- 1959年，神经生理学家David Hubel和Torsten Wiesel通过猫的视觉实验，首次发现了视觉初级皮层神经元对于移动边缘刺激敏感，发现了视功能柱结构，为视觉神经研究奠定了基础。他们发现简单细胞对光的朝向和移动产生反应，后续连接的复杂细胞层能响应端点的移动，最终在皮层认出了边角、形状。



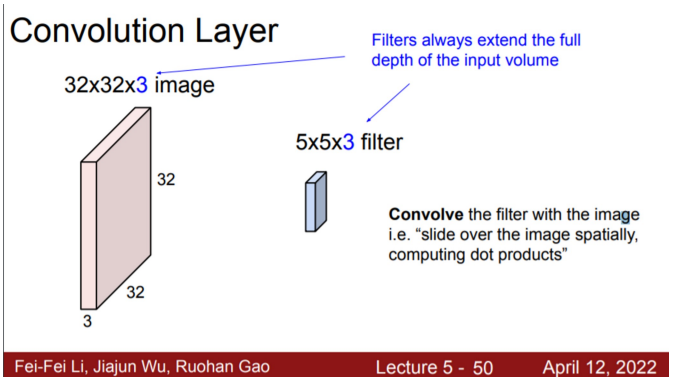
- 1998年，Yann LeCun和Yoshua Bengio首次展示了第一个实例，LeNet，应用反向传播和梯度下降来训练卷积神经网络，这对数字文档识别非常有效，但并不能扩展到复杂数据。
- 2012年，Alex Krizhevsky和Geoffrey Hinton提出了一种现代化的卷积神经网络，称为AlexNet，它和Yann LeCun提出的CNN看上去没有太大差异，只是扩展的更大更深。最重要的一点是它们可以利用ImageNet数据集中海量图像数据。同时充分发挥了GPU并行计算能力



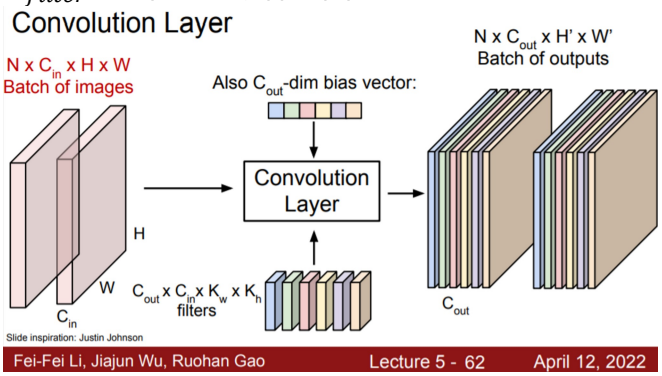
- 卷积层不同于全连接层，输入的图像/向量可以是变长的，卷积核长/宽/通道数不变，可得到不同大小的特征输出
- 如今，CNN被广泛应用与各种计算机视觉任务并取得较好效果，主要包括图像分类、图像检索、目标检测、语义分割、序列图像处理、姿态/行为识别、风格迁移等

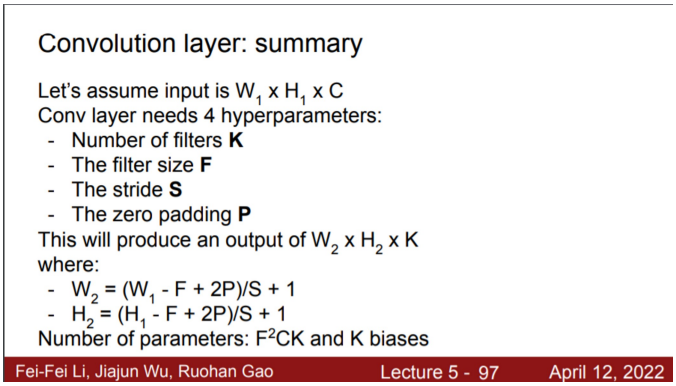
二、CNN基本层：卷积层、池化层

- 卷积层的参数是一系列卷积核（filter）（及其偏置项），卷积核在输入图像的矩阵上作”卷积“操作，即在图像上滑动，卷积核与图像对应位置子向量作点积，卷积核默认与图像的通道数（深度）相同，因此通过N个卷积核得到的激活图通道数为N

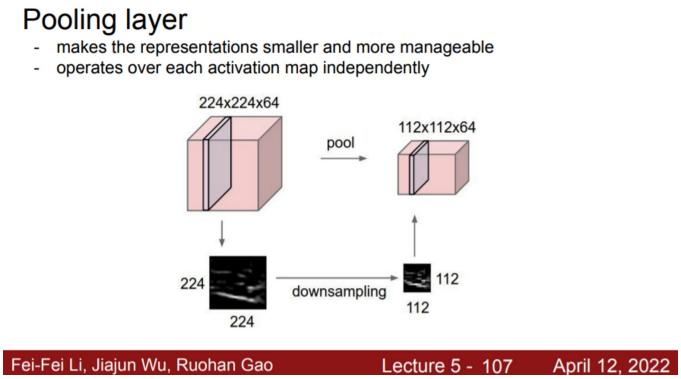


- 因此卷积层输入输出通道为C_{in}, C_{out}时，输入大小为$[C_{in}, H, W]$，输出大小为$[C_{out}, H`, W`]$，卷积核参数量为$C_{in} \times C_{out} \times H_{filter} \times W_{filter}$，卷积层可以保留图像的空间结构





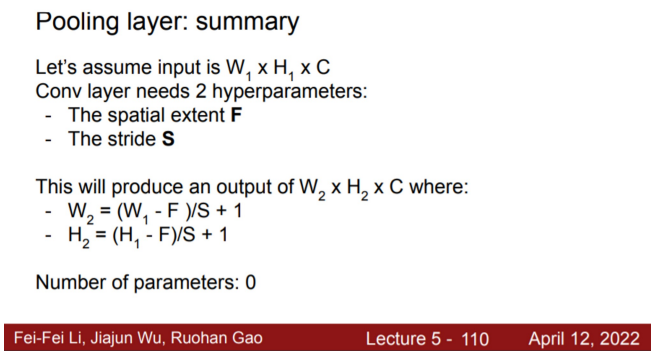
- 不同于全连接层始，卷积层认为输入具有局部连接性，所以不关注输入图片的全部，而是关注局部区域。卷积层某位置的输出对应输出神经元在各个区域所被激发的程度，卷积层某位置的不同通道的输出对应不同的卷积核（从低级特征提取不同角度的更高级特征）



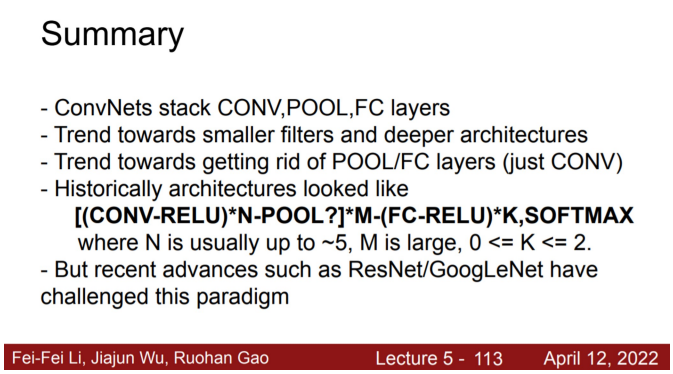
- 池化层（Pooling layers）就是要做的是降采样，池化层会不断地减小数据的空

间大小，因此参数的数量和计算量也会下降，这在一定程度上也控制了过拟合。常用池化方法：Max-Pooling

- 增大卷积层stride（步长）与使用池化层的区别：池化层无权重参数



- 经典的卷积神经网络结构（图像分类）：特征图提取：若干卷积层-激活函数-池化层堆叠，后续接入全连接层构成完整分类器



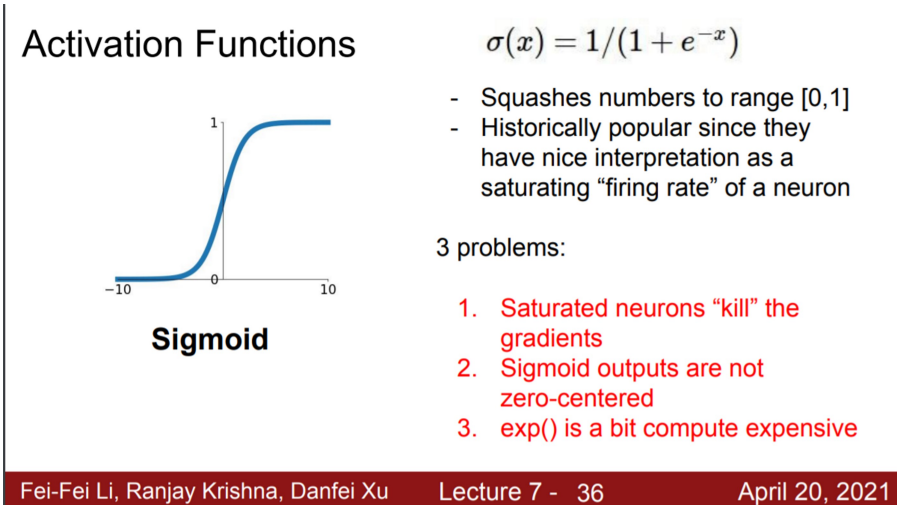
Lecture6/7 Training Neural Networks

2022年4月8日　　15:14

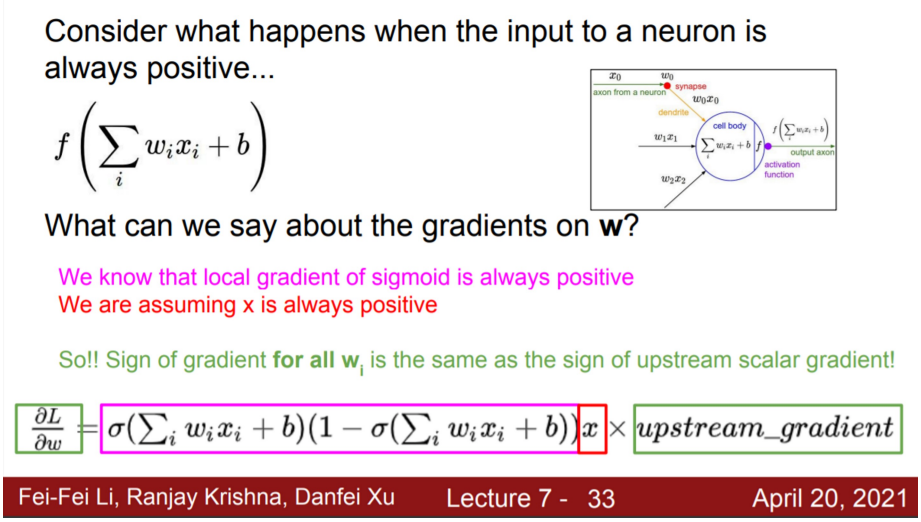
一、非线性激活函数

- Sigmoid缺点：

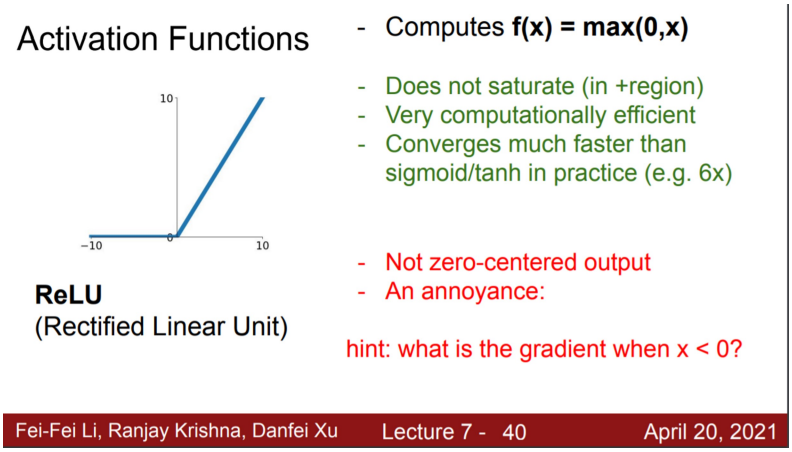
  导数最大值为1/4，易造成梯度消失



- 网络中一层参数的本地梯度始终与上游梯度同号，限制整体参数的更新方向，可能会导致训练收敛慢

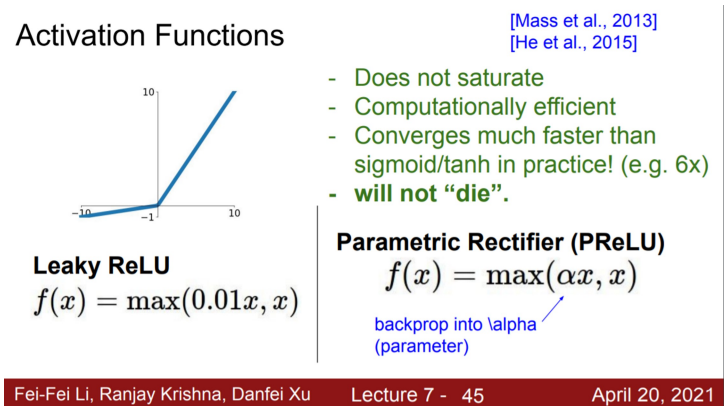


- ReLU对恒等函数负区间截断为0，实践中可以快速收敛



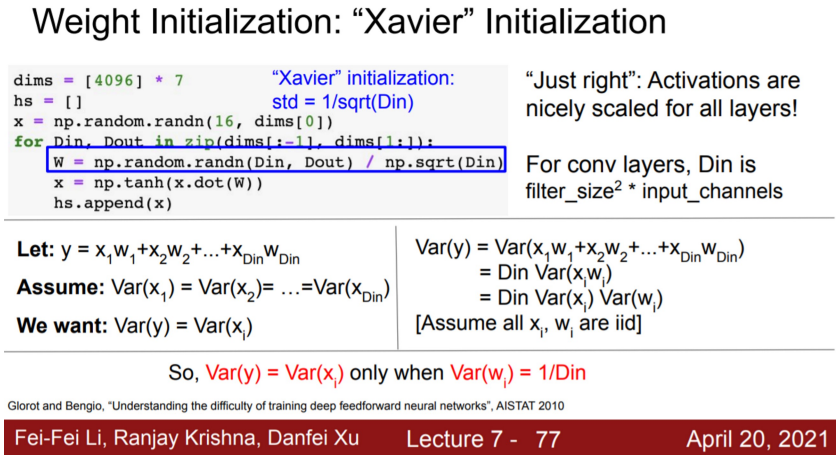
Dead ReLU：当所有数据通过一层ReLU输出为0时，此层ReLU反向传播中无法传递梯度

解决方法：LeakyReLU\ELU\SELU\Maxout

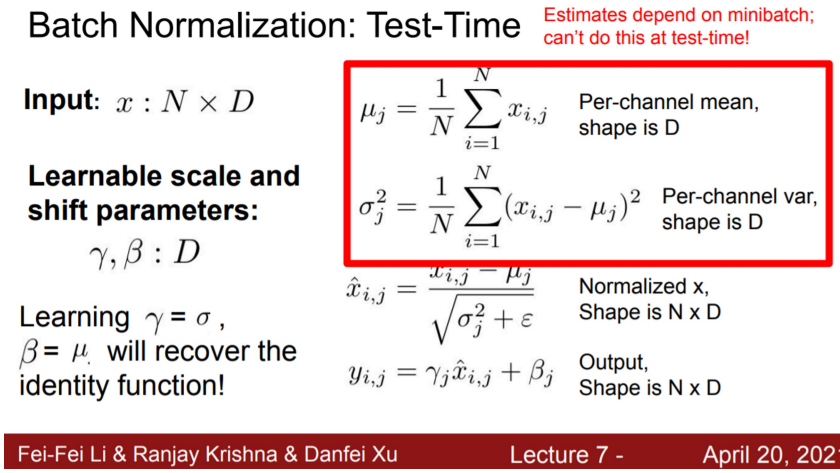


二、Weigjt initation/BN

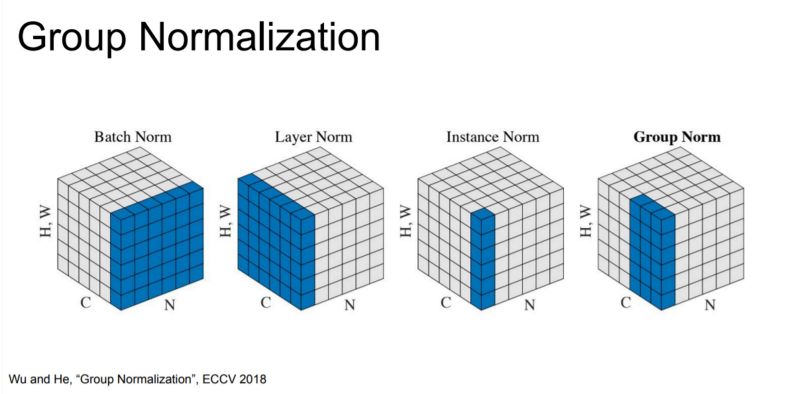
- 权重初始大小过小/大易导致梯度消失/爆炸，Xavier初始化方法的思路：保证输入/输出数据方差相同



- BN层，训练时将数据各维度缩放到均值=0，方差=1（测试时使用训练时统计的均值，方差作标准化），意在消除模型不同层差异过大数据分布，同时防止BN层缩放后对本层激活函数的影响，加入可学习的线性放大系数（实质上是学习数据缩放的程度）

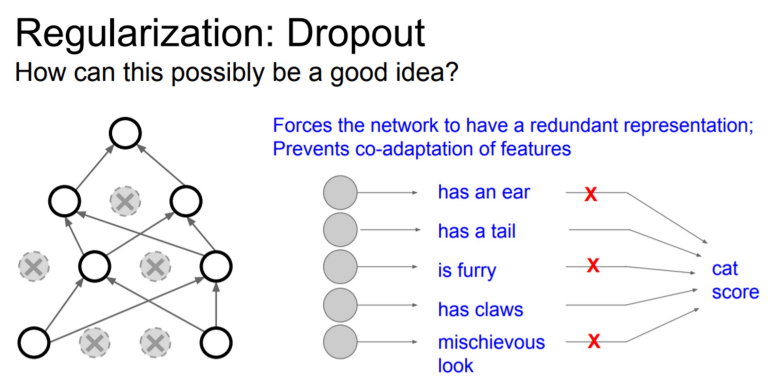


- 不同的标准化层如BN、LN关注的是单个数据内部如何标准归一化（BN受batch-size影响）

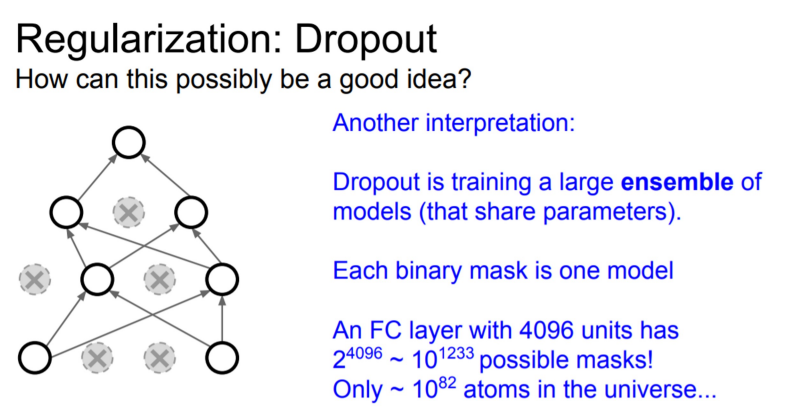
- 

三、正则化策略

- Dropout层一定程度上可以减缓分类器的过拟合程度



- 可视为每次迭代中不同子模型的集成



- 训练集数据增强如旋转、随机遮挡、色彩抖动生成不同视角的数据，减轻模型泛化能力差（如视角变化场景）的问题

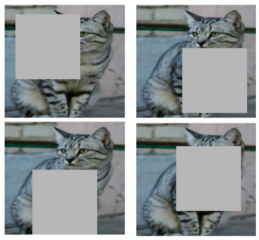
Regularization: Cutout

Training: Set random image regions to zero

Testing: Use full image

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth
Cutout / Random Crop



Works very well for small datasets like CIFAR, less common for large datasets like ImageNet

DeVries and Taylor, "Improved Regularization of Convolutional Neural Networks with Cutout", arXiv 2017