获得授权翻译斯坦福CS231n课程笔记系列

获得授权翻译斯坦福CS231n课程笔记系列

CS231n课程笔记翻译：Python Numpy教程

CS231n 课程笔记翻译：Python Numpy教程
numpy for Matlab users

CS231n课程笔记翻译：图像分类笔记（上）

CS231n 课程笔记翻译：图像分类笔记（上）

⋆ 图像分类

DEF: 图像分类

所谓图像分类问题，就是已有固定的分类标签集合，然后对于输入的图像，从分类标签集合中找出一个分类标签，最后把分类标签分配给该输入图像。

困难和挑战： 视角变化(Viewpoint variation)、大小变化(Scale variation)、形变(Deformatio
遮挡(Occlusion)、光照条件(Illumination conditions)、背景干扰(Background clutter)、类内差异(Intra-class variation)

DEF: 数据驱动方法

给计算机很多数据，然后实现学习算法，让计算机学习到每个类的外形。这种方法，就是数据驱动方法。

数据驱动方法的第一步就是收集已经做好分类标注的图片来作为训练集。

图像分类流程： 输入(输入训练集) $\Longrightarrow$学习(训练分类哭或者学习模型)$\Longrightarrow$评价(评价分类器)

⋆ Nearest Neighbor分类器

图像分类数据集： CIFAR-10

DEF: Nearest Neighbor算法

Nearest Neightbor算法将会拿差测试图片和训练集中每一张图片比较，然后将它认为最相似的那个训练集图片的标签赋给这张测试图片。

- **L1距离**

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

- **L2距离**

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

EX: Nearest Neighbor for CIFAR-10

基于CIFAR-10数据的Kaggle算法竞赛排行榜：CIFAR-10 - Object Recognition in Images

★ K-Nearest Neighbor分类器

DEF: K-Nearest Neighbor分类器

其思想：与其只找最相近的那1个图片的标签，不如找最相似的k个图片的标签，然后让他们针对测试图片进行投票，最后把票数最高的标签作为对测试图片的预测。

CS231n课程笔记翻译：图像分类笔记（下）

CS231n 课程笔记翻译：图像分类笔记（下）

★ 用于超参数调优的验证集

DEF: 超参数(hyperparameter)

k-NN分类器需要设定k值，那么选择哪个k值最合适的呢？对于不同的距离函数，比如L1范数和L2范数等，那么选哪个好？还有不少选择需要考虑到(比如：点积)。所有这些选择，被称为**超参数(hyperparameter)**。

Remark: 决不能使用测试集来进行调优。

测试数据集只使用一次，即在训练完成后评价最终的模型时使用。

当你在设计机器学习算法的时候，应该把测试集看做非常珍贵的资源，不到最后一步，绝不使用它。如果你使用测试集来调优，而且算法看起来效果不错，那么真正的危险在于：算法实际部署后，性能可能会远低于预期。这种情况，称之为算法对测试集**过拟合**。

从训练集中取出一部分数据用来调优，我们称之为**验证集（validation set）**。验证集其实就是作为假的测试集来调优。

把训练集分成训练集和验证集。使用验证集来对所有超参数调优。最后只在测试集上跑一次并报告结果。

当训练集数量较小时（因此验证集的数量更小），使用**交叉验证**的方法。

★ Nearest Neighbor分类器的优劣

实现简单，训练时间短，测试时间长。但在实际应用中，更关注测试效率。

CS231n课程笔记翻译：线性分类笔记（上）

CS231n 课程笔记翻译：线性分类笔记（上）

★ 线性分类

评分函数(score function)：原始图像数据到类别分值的映射。

损失函数(loss function)：用来量化预测分类标签的得分与真实标签之间一致性的。

★ 从图像到标签分值的参数化映射

线性分类器

$$f(x_i, W, b) = Wx_i + b$$

★ 理解线性分类器

将图像看做高维度的点

将线性分类器看做模板匹配

偏差和权重的合并技巧

$$f(x_i, W, b) = Wx_i + b \Longrightarrow f(x_i, W) = Wx_i$$

图像数据预处理：零均值的中心化

CS231n课程笔记翻译：线性分类笔记（中）

CS231n 课程笔记翻译：线性分类笔记（中）

★ 损失函数Loss function

DEF: 损失函数(Loss Function)

使用**损失函数(L oss Function)**(有时也叫**代价函数Cost Function**或**目标函数Objective**)来衡量我们对结果的不满意程度。直观地讲，当评分函数输出结果与真实结果之间差异越大，损失函数输出越大，反之越小。

★ 多类支持向量机损失Multiclass Support Vector Machine Loss

DEF: 多类支持向量机（SVM）损失函数

SVM的损失函数想要SVM在正确分类上的得分始终比不正确分类上的得分高出一个边界值Δ。

针对第i个数据的多类SVM的损失函数定义如下：

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

简而言之，SVM的损失函数想要正确分类类别y_i的分数比不正确类别分数高，而且至少要高Δ。如果不满足这点，就开始计算损失值。

对于线性评分函数($f(x_i, W) = Wx_i$)，将损失函数的公式改写如下：

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

其中w_j是权重W的第j行，被变形为列向量。

$\max(0, -)$函数，它常被称为**折叶损失(hinge loss)**。有时候会听到人们使用平方折叶损失SVM(即L2-SVM)，它使用的是$\max(0, -)^2$，将更强烈(平方地而不是线性地)地惩罚过界的边界值。

我们对于预测训练集数据分类标签的情况总有一些不满意的，而损失函数就能将这些不满意的程度量化。

正则化惩罚(regularization penalty)

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

包含正则化惩罚后，就能够给出完整的多类SVM损失函数了，它由两个部分组成：**数据损失(data loss)**，即所有样例的的平均损失L_i，以及**正则化损失(regularization loss)**。完整公式如下所示：

$$L = \underbrace{\frac{1}{N}\sum_i L - i +}_{data\ loss} \underbrace{\lambda R(W)}_{regularization\ loss}$$

将其展开完整公式是：

$$L = \frac{1}{N}\sum_i \sum_{j \neq y_i} [\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta)] + \lambda \sum_k \sum_l W_{k,l}^2$$

其中，N是训练集的数据量。现在正则化惩罚添加到了损失函数里面，并用超参数λ来计算其权重。该超参数无法简单确定，需要通过交叉验证来获取。

★ 实际考虑

设置Δ

与二元支持向量机(Binary Support Vector Machine)的关系：二元支持向量机对第i个数据的损失计算公式是：

$$L_i = C \max(0, 1 - y_i w^T x_i) + R(W)$$

其中，C 是一个超参数，并且$y_i \in \{-1, 1\}$。公式中的C 和多类SVM公式中的λ都控制着同样的权衡，而且它们之间的关系是$C \propto \frac{1}{\lambda}$。

备注：在初始形式中进行最优化

备注：其他多类SVM公式

CS231n课程笔记翻译：线性分类笔记（下）

CS231n 课程笔记翻译：线性分类笔记（下）

★ Softmax分类器

交叉熵损失(cross-entropy loss)

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

或等价的

$$L_i = -f_{y_i} + \log\left(\sum_j e^{f_j}\right)$$

函数$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$被称作**softmax函数**：其输入值是一个向量，向量中元素为任意实数的评分值(z中的)，函数对其进行压缩，输出一个向量，其中每个元素值在0到1之间，且所有元素之和为1。

信息理论视角：在“真实”分布p和估计分布q之间的交叉熵定义如下：

$$H(p,q) = -\sum_x p(x) \log q(x)$$

Kullback-Leibler差异（Kullback-Leibler Divergence）也叫做相对熵（Relative Entropy），它衡量的是相同事件空间里的两个概率分布的差异情况。

概率论解释：

$$P(y_i|x_i, W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$

可以解释为是给定图像数据x_i，以W为参数，分配给正确分类标签y_i的归一化概率。

⋆ SVM和Softmax的比较

EX: SVM和Softmax的比较

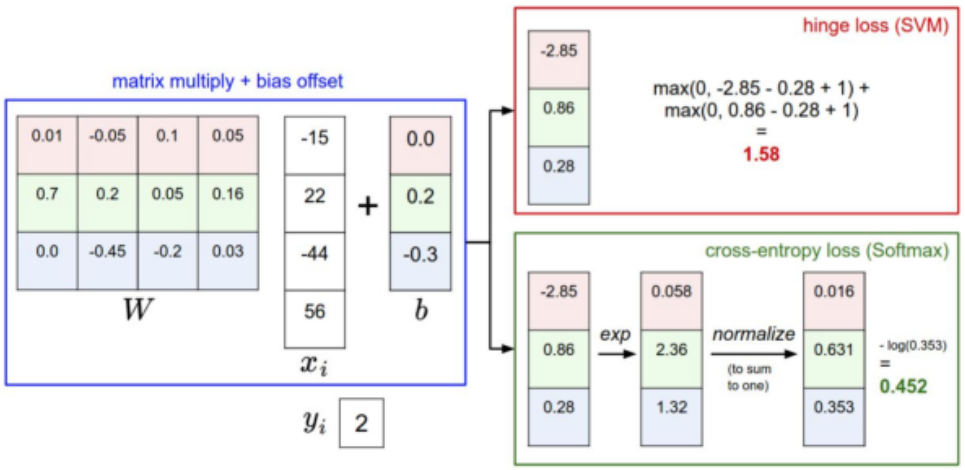


Figure 1: SVM和Softmax的比较

两个分类器都计算了同样的分值向量f(本节中是通过矩阵乘来实现)。不同之处在于对f中分值的解释：SVM分类器将它们看做是分类评分，它的损失函数鼓励正确的分类(本例中是蓝色的类别2)的分值比其他分类的分值高出至少一个边界值。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**，鼓励正确分类的归一化的对数概率变高，其余的变低。

Softmax分类器为每个分类提供了“可能性”

在实际使用中，SVM和Softmax经常是相似的

CS231n课程笔记翻译：最优化笔记（上）

CS231n 课程笔记翻译：最优化笔记（上）

★ 简介

图像分类任务中的两个关键部分：

1. 基于参数的**评分函数**。该函数将原始图像像素映射为分类评分值(例如：一个线性函数)。
2. **损失函数**。该函数能够根据分类评分和训练集图像数据实际分类的一致性，衡量某个具体参数集的质量好坏。损失函数有多种版本和不同的实现方式(例如：Softmax或SVM)。

线性函数的形式是$f(x_i, W) = Wx_i$，而SVM实现的公式是：

$$L = \frac{1}{N}\sum_i \sum_{j \neq y_i} [\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)] + \alpha R(W)$$

DEF: 最优化Optimization

最优化是寻找能使得损失函数值最小化的参数W的过程。

★ 损失函数可视化

凸函数最优化

次梯度(subgradient)

★ 最优化Optimization

损失函数可以量化某个具体权重集W的质量。而最优化的目标就是找到能够最小化损失函数值的W。

策略#1：一个差劲的初始方案：随机搜索

核心思路：迭代优化 从随机权重开始，然后迭代取优，从而获得更低的损失值。

策略#2：随机本地搜索

从一个随机W开始，然后生成一个随机的扰动δW，只有当$W + \delta W$的损失值变低，我们才会更新。

策略#3：跟随梯度

从数学上计算出最陡峭的方向，这个方向就是损失函数的**梯度(gradient)**。

对一维函数的求导公式如下：

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

CS231n课程笔记翻译：最优化笔记（下）

CS231n 课程笔记翻译：最优化笔记（下）

★ 梯度计算

计算梯度有两种方法：一个是缓慢的近似方法(**数值梯度法**)，但实现相对简单。另一个方法(**分析梯度法**)计算迅速，结果精确，但是实现时容易出错，且需要使用微分。

★ 利用有限差值计算梯度

实践考量：中心差值公式(centered difference formula)$[f(x+h)-f(x-h)]/2h$

在梯度负方向上更新

步长的影响

效率问题

★ 微分分析计算梯度

SVM的损失函数

$$L_i = \sum_{j \neq y_i} [\max(0, w_j^T x_i - w_{j_i}^T x_i + \Delta)]$$

关于w_{y_i}对其进行微分得到：

$$\nabla_{w_{j_i}} L_i = -(\sum_{j \neq y_i} \mathbb{I}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0))x_i$$

注意，这个梯度只是对应正确分类的W的行向量的梯度，那些$j \neq y_i$行的梯度是：

$$\nabla_{w_j} L_i = \mathbb{I}(w_j^T x_i - w_{y_i} T x_i + \Delta > 0)x_i$$

★ 梯度下降

小批量数据梯度下降(Mini-batch gradient descent)

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

斯坦福CS231n课程作业#1简介

斯坦福CS231n课程作业#1简介

CS231n课程笔记翻译：反向传播笔记

CS231n 课程笔记翻译：反向传播笔记

★ 简介

问题陈述：给定函数$f(x)$，其中x是输入数据的向量，需要计算函数f关于x的梯度，也就是$\nabla f(x)$。

★ 简单表达式和理解梯度

函数关于每个变量的导数指明了整个表达式对于该变量的敏感程度。

★ 使用链式法则计算复合表达式

链式法则指出将梯度表达式链接起来的正确方式是相乘，比如$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$。

★ 反向传播的直观理解

链式法则指出，门单元应该将回传的梯度乘以它对其的输入的局部梯度，从而得到整个网络的输出对该门单元的每个输入值的梯度。

★ 模块化：Sigmoid例子

sigmoid函数关于其输入的求导是可以简化的：

$$\sigma(x) = \frac{1}{1+e^{-x}} \rightarrow \frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-2})^2} = (\frac{1+e^{-x}-1}{1+e^{-x}})(\frac{1}{1+e^{-x}}) = (1-\sigma(x))\sigma(x)$$

实现提示：分段反向传播

★ 反向传播实践：分段计算

对前向传播变量进行缓存

在不同分支的梯度要相加

★ 回传流中的模式

加法操作将梯度相等地分发给它的输入。取最大操作将梯度路由给更大的输入。乘法门拿取输入激活数据，对它们进行交换，然后乘以梯度。

★ 用向量化操作计算梯度

矩阵相乘的梯度：可能最有技巧的操作是矩阵相乘（也适用于矩阵和向量，向量和向量相乘）的乘法操作。

CS231n课程笔记翻译：神经网络笔记1（上）

CS231n 课程笔记翻译：神经网络笔记1（上）

★ 快速简介

★ 单个神经元建模

★ 生物动机与连接

生物神经元从它的**树突**获得输入信号，然后沿着它唯一的**轴突(axon)**产生输出信号。

激活函数(activation function) f常选择使用**sigmoid 函数**σ，该函数输入实数值（求和后的信号强度），然后将输入值压缩到0-1之间。

每个神经元都对它的输入和权重进行点积，然后加上偏差，最后使用非线性函数（或称为激活函数）。

⋆ 作为线性分类器的单个神经元

二分类Softmax分类器

二分类SVM分类器

理解正则化

一个单独的神经元可以用来实现一个二分类分类器，比如二分类的Softmax或者SVM分类器。

⋆ 常用激活函数

Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$

Tanh

$$tanh(x) = 2\sigma(2x) - 1$$

ReLU

$$f(x) = \max(0, x)$$

Leaky ReLU

$$f(x) = \mathbb{I}(x < 0)(\alpha x) + ((x >= 0)(x)$$

Maxout

$$\max(w_1^T x + b_1, w_2^T + b_2)$$

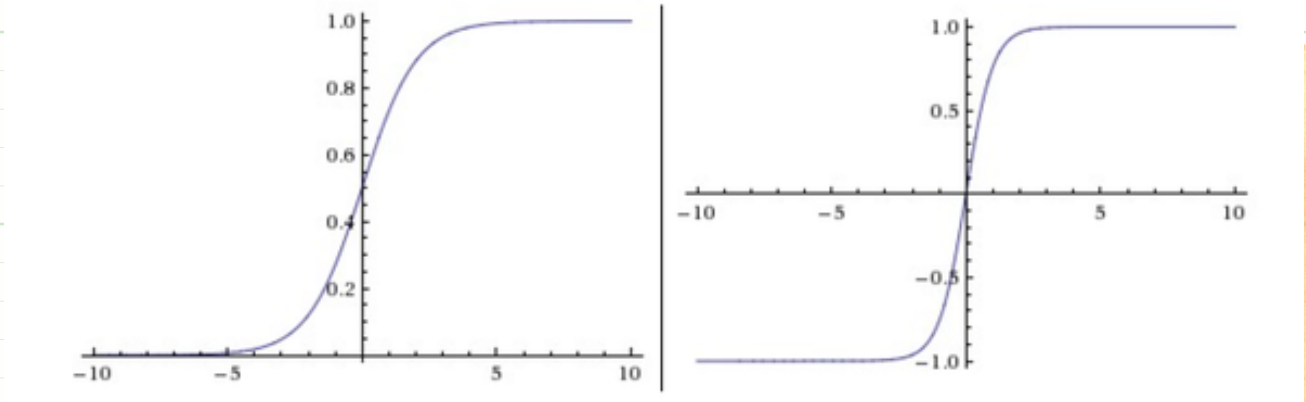


Figure 2: 左边是Sigmoid非线性函数，将实数压缩到$[0, 1]$之间。右边是tanh函数，将实数压缩到$[-1, 1]$。

Sigmoid非线性函数有两个主要缺点：

- Sigmoid函数饱和使梯度消失。

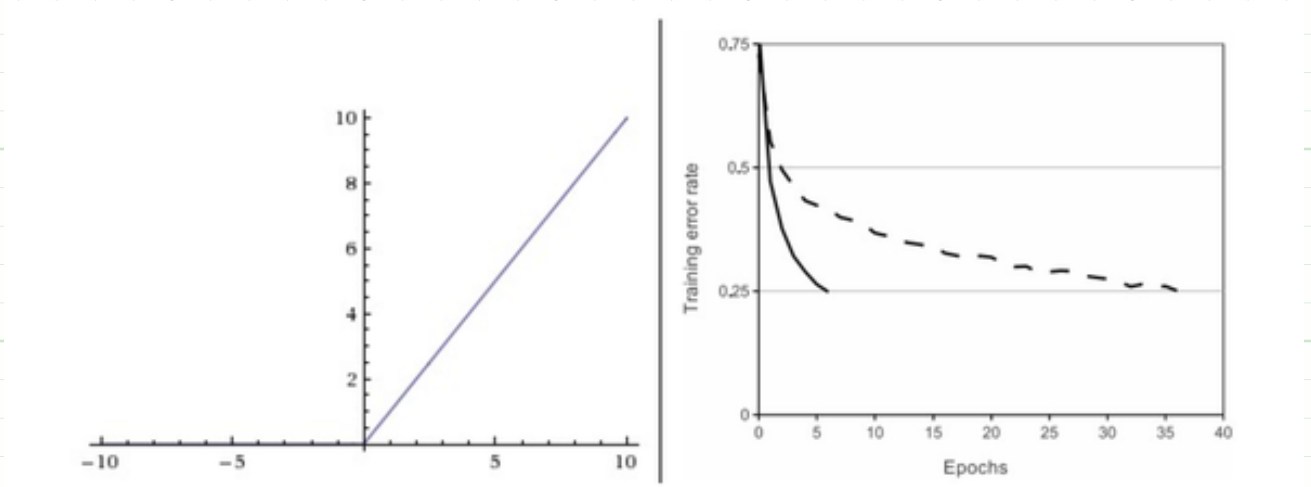


Figure 3: 左边是ReLU(校正线性单元：Rectified Linear Unit)激活函数，当$x = 0$时函数值为0。当$x > 0$函数的斜率为1。右边是从Krizhevsky 等的论文中截取的图表，指明使用ReLU比使用tanh的收敛快6倍。

- Sigmoid函数的输出不是零中心的。

Tanh非线性函数也存在饱和问题，但是它的输出是零中心的。
ReLU有以下优缺点：

- 优点：相较于sigmoid和tanh函数，ReLU对于随机梯度下降的收敛有巨大的加速作用。据称这是由它的线性，非饱和的公式导致的。
- 优点：sigmoid和tanh神经元含有指数运算等耗费计算资源的操作，而ReLU可以简单地通过对一个矩阵进行阈值计算得到。
- 缺点：在训练的时候，ReLU单元比较脆弱并且可能“死掉”。

Leaky ReLU是为解决“ReLU死亡”问题的尝试。有些研究者的论文指出Leaky ReLU激活函数表现很不错，但是其效果并不是很稳定。

Maxout是对ReLU和leaky ReLU的一般化归纳，ReLU和Leaky ReLU都是这个公式的特殊情况(比如ReLU就是当$w_1, b_1 = 0$ 的时候)。这样Maxout神经元就拥有ReLU单元的所有优点(线性操作和不饱和)，而没有它的缺点(死亡的ReLU单元)。然而和ReLU对比，它每个神经元的参数数量增加了一倍，这就导致整体参数的数量激增。

一句话：“那么该用那种呢？”用ReLU非线性函数。

CS231n课程笔记翻译：神经网络笔记1（下）

CS231n 课程笔记翻译：神经网络笔记1（下）

★ 神经网络结构

★ 灵活地组织层

全连接层中的神经元与其前后两层的神经元是完全成对连接的，但是在同一个全连接层内的神经元之间没有连接。

输出层的神经元一般是不会有激活函数的。

用来度量神经网络的尺寸的标准主要有两个：一个是神经元的个数，另一个是参数的个数。

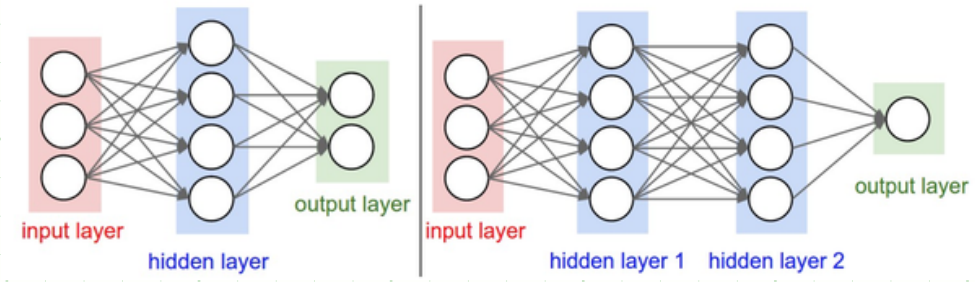


Figure 4: 左边是一个2层神经网络，隐层由4个神经元(也可称为单元(unit))组成，输出层由2个神经元组成，输入层是3 个神经元。右边是一个3层神经网络，两个含4个神经元的隐层。注意：层与层之间的神经元是全连接的，但是层内的神经元不连接。

★ 前向传播计算举例

全连接层的前向传播一般就是先进行一个矩阵乘法，然后加上偏置并运用激活函数。

★ 表达能力

拥有至少一个隐层的神经网络是一个通用的近似器。在研究中已经证明，给出任意连续函数$f(x)$和任意$\epsilon > 0$，均存在一个至少含1个隐层的神经网络$g(x)$(并且网络中有合理选择的非线性激活函数，比如sigmoid)，对于$\forall x$，使得$|f(x) - g(x)| < \epsilon$。换句话说，神经网络可以近似任何连续函数。

既然一个隐层就能近似任何函数，那为什么还要构建更多层来将网络做得更深？答案是：虽然一个2层网络在数学理论上能完美地近似所有连续函数，但在实际操作中效果相对较差。

在实践中3层的神经网络会比2层的表现好，然而继续加深(做到4，5，6层)很少有太大帮助。卷积神经网络的情况却不同，在卷积神经网络中，对于一个良好的识别系统来说，深度是一个极端重要的因素(比如数十(以10为量级)个可学习的层)。

★ 设置层的数量和尺寸

当增加层的数量和尺寸时，网络的容量上升了。即神经元们可以合作表达许多复杂函数，所以表达函数的空间增加。

ConvnetJS demo

DEF: 过拟合(Overfitting)

过拟合(Overfitting)是网络对数据中的噪声有很强的拟合能力，而没有重视数据间(假设)的潜在基本关系。

不要减少网络神经元数目的主要原因在于小网络更难使用梯度下降等局部方法来进行训练：虽然小型网络的损失函数的局部极小值更少，也比较容易收敛到这些局部极小值，但是这些最小值一般都很差，损失值很高。相反，大网络拥有更多的局部极小值，但就实际损失值来看，这些局部极小值表现更好，损失更小。

正则化强度是控制神经网络过拟合的好方法。

不应该因为害怕出现过拟合而使用小网络。相反，应该进尽可能使用大网络，然后使用正则化技巧来控制过拟合。

CS231n课程笔记翻译：神经网络笔记2

CS231n 课程笔记翻译：神经网络笔记2

★ 设置数据和模型

神经元在计算内积后进行非线性激活函数计算，神经网络将神经元组织成各个层。这些做法共同定义了评分函数(score function)的新形式，该形式是从前面线性分类章节中的简单线性映射发展而来的。具体来说，神经网络就是进行了一系列的线性映射与非线性激活函数交织的运算。

★ 数据预处理

数据预处理有3个常用的符号，数据矩阵X，假设其尺寸是$[N \times D]$(N是数据样本的数量，D是数据的维度)。

DEF: 均值减法(Mean subtraction)

对数据中每个独立特征减去平均值，从几何上可以理解为在每个维度上都将数据云的中心都迁移到原点。

DEF: 归一化(Normalization)

将数据的所有维度都归一化，使其数值范围都近似相等。

有两种常用方法可以实现归一化。第一种是先对数据做零中心化(zero-centered)处理，然后每个维度都除以其标准差。第二种方法是对每个维度都做归一化，使得每个维度的最大和最小值是1和-1。

DEF: PCA和白化(Whitening)

先对数据进行零中心化处理，然后计算协方差矩阵，它展示了数据中的相关性结构。

白化操作的输入是特征基准上的数据，然后对每个维度除以其特征值来对数值范围进行归一化。该变换的几何解释是：如果数据服从多变量的高斯分布，那么经过白化后，数据的分布将会是一个均值为零，且协方差相等的矩阵。

进行预处理很重要的一点是：任何预处理策略(比如数据均值)都只能在训练集数据上进行计算，算法训练完毕后再应用到验证集或者测试集上。

★ 权重初始化

错误：全零初始化。如果权重被初始化为同样的值，神经元之间就失去了不对称性的源头。

小随机数初始化。权重初始值要非常接近0又不能等于0。如果神经元刚开始的时候是随机且不相等的，那么它们将计算出不同的更新，并将自身变成整个网络的不同部分。

使用$1/\sqrt{(n)}$校准方差。除以输入数据量的平方根来调整其数值范围，这样神经元输出的方差就归一化到1了。

稀疏初始化(Sparse initialization)。

偏置(biases)的初始化。通常将偏置初始化为0，这是因为随机小数值权重矩阵已经打破了对称性。

批量归一化(Batch Normalization)。批量归一化可以理解为在网络的每一层之前都做预处理，只是这种操作以另一种方式与网络集成在了一起。

★ 正则化Regularization

L2正则化对于网络中的每个权重w，向目标函数中增加一个$\frac{1}{2}\lambda w^2$。L2正则化可以直观理解为它对于大数值的权重向量进行严厉惩罚，倾向于更加分散的权重向量。使网络更倾向于使用所有输入特征，而不是严重依赖输入特征中某些小部分特征。

L1正则化对于每个w，都向目标函数增加一个$\lambda|w|$。L1和L2正则化也可以进行组合：$\lambda_1|w|+\lambda_2 w^2$，这也被称作Elastic net regularizaton。L1正则化有一个有趣的性质，它会让权重向量在最优化的过程中变得稀疏(即非常接近0)。

最大范式约束(Max norm constraints)给每个神经元中权重向量的量级设定上限，并使用投影梯度下降来确保这一约束。

随机失活(Dropout)

前向传播中的噪音

偏置正则化

每层正则化

★ 损失函数

分类问题

问题：类别数目巨大

属性(Attribute)分类

回归问题

结构化预测(structured prediction)

CS231n课程笔记翻译：神经网络笔记3（上）

CS231n 课程笔记翻译：神经网络笔记3（上）

★ 学习过程

神经网络的静态部分：如何创建网络的连接、数据和损失函数。

神经网络的动态部分，即神经网络学习参数和搜索最优超参数的过程。

★ 梯度检查

使用中心化公式。在使用有限差值近似来计算数值梯度的时候，常见的公式是：

$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x)}{h} \ (bad, donotuse)$$

其中h是一个很小的数字，在实践中近似为$1e-5$。在实践中证明，使用中心化公式效果更好：

$$\frac{df(x)}{d(x)} = \frac{f(x+h) - f(x-h)}{2h} \ (useinstead)$$

一个公式的误差近似$O(h)$，第二个公式的误差近似$O(h^2)$(是个二阶近似)。

使用相对误差来比较。

$$\frac{|f_a^{'} - f_n^{'}|}{\max(|f_a^{'}|, |f_n^{'}|)}$$

在实践中：

- 相对误差> $le-2$：通常就意味着梯度可能出错。
- $1e-2$ >相对误差> $1e-4$：要对这个值感到不舒服才行。
- $1e-4$ >相对误差：这个值的相对误差对于有不可导点的目标函数是OK的。但如果目标函数中没有kink(使用tanh和softmax)，那么相对误差值还是太高。
- $1e-7$或者更小：好结果，可以高兴一把了。

使用双精度。

保持在浮点数的有效范围。

目标函数的不可导点(kinks)。在计算损失的过程中是可以知道不可导点有没有被越过的。在具有$\max(x, y)$形式的函数中持续跟踪所有“赢家”的身份，就可以实现这一点。其实就是看在前向传播时，到底x和y谁更大。如果在计算$f(x+h)$和$f(x-h)$的时候，至少有一个“赢家”的身份变了，那就说明不可导点被越过了，数值梯度会不准确。

使用少量数据点。解决不可导点问题的一个办法是使用更少的数据点。

谨慎设置步长h。在实践中h并不是越小越好，因为当h特别小的时候，就可能就会遇到数值精度问题。

在操作的特性模式中梯度检查。梯度检查是在参数空间中的一个特定(往往还是随机的)的单独点进行的。即使是在该点上梯度检查成功了，也不能马上确保全局上梯度的实现都是正确的。

不要让正则化吞没数据。先关掉正则化对数据损失做单独检查，然后对正则化做单独检查。

记得关闭随机失活(dropout)和数据扩张(augmentation)。一个更好的解决方案就是在计算$f(x+h)$和$f(x-h)$前强制增加一个特定的随机种子，在计算解析梯度时也同样如此。

检查少量的维度。

★ 学习之前：合理性检查的提示与技巧

- **寻找特定情况的正确损失值。**
- 提高正则化强度时导致损失值变大。
- **对小数据子集过拟合。**

★ 检查整个学习过程

★ 损失函数

★ 训练集和验证集准确率

★ 权重更新比例

★ 每层的激活数据及梯度分布

一个不正确的初始化可能让学习过程变慢，甚至彻底停止。还好，这个问题可以比较简单地诊断出来。其中一个方法是输出网络中所有层的激活数据和梯度分布的柱状图。直观地说，就是如果看到任何奇怪的分布情况，那都不是好兆头。

★ 第一层可视化

CS231n课程笔记翻译：神经网络笔记3（下）

CS231n 课程笔记翻译：神经网络笔记3（下）

★ 参数更新

★ 随机梯度下降及各种更新方法

- 普通更新
- 动量(Momentum)更新

普通更新沿着负梯度方向改变参数(因为梯度指向的是上升方向，但是我们通常希望最小化损失函数)。

★ 学习率退火

- **随步数衰减。**每进行几个周期就根据一些因素降低学习率。
- **指数衰减。**数学公式是$\alpha = \alpha_0 e^{-kt}$，其中$\alpha_0, k$是超参数，$t$是迭代次数(也可以使用周期作为单位)。
- **1/t衰减。**数学公式是$\alpha = \alpha_0/(1+kt)$，其中α_0, k是超参数，t是迭代次数。

★ 二阶方法

在深度网络背景下，第二类常用的最优化方法是基于牛顿法的，其迭代如下：

$$x \leftarrow x - [Hf(x)]^{-1}\nabla f(x)$$

这里$Hf(x)$是Hessian矩阵，它是函数的二阶偏导数的平方矩阵。$\nabla f(x)$是梯度向量。直观理解上，Hessian矩阵描述了损失函数的局部曲率，从而使得可以进行更高效的参数更新。具体来说，就是乘以Hessian转置矩阵可以让最优化过程在曲率小的时候大步前进，在曲率大的时候小步前进。

★ 逐参数适应学习率方法

Adagrad

RMSprop

Adam

★ 超参数调优

神经网络最常用的设置有：

- 初始学习率。
- 学习率衰减方式(例如一个衰减常量)。
- 正则化强度(L2惩罚，随机失活强度)。

实现。

比起交叉验证最好使用一个验证集。

超参数范围。

随机搜索优于网格搜索。

对于边界上的最优值要小心。

从粗到细地分阶段搜索。

贝叶斯超参数最优化

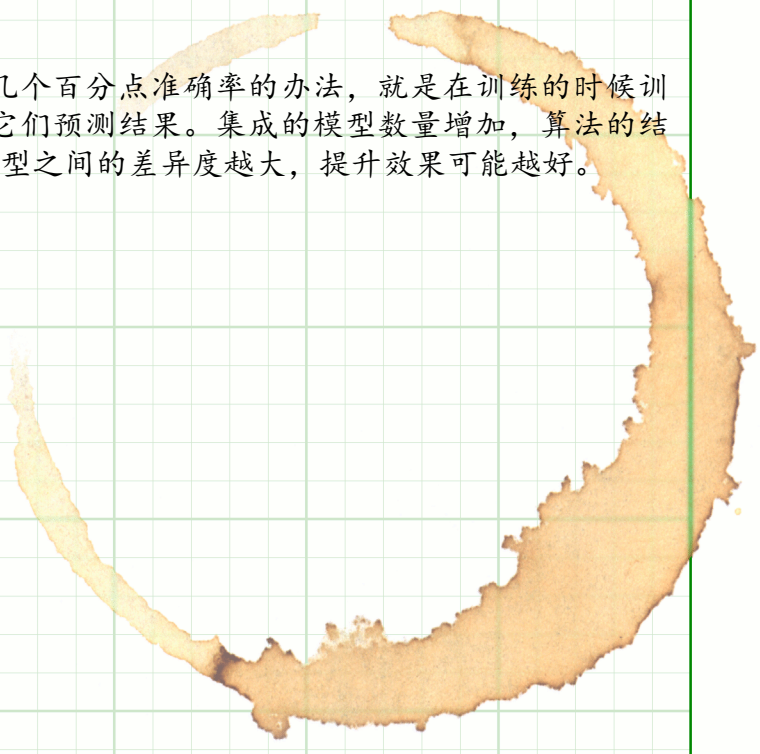
★ 评价

★ 模型集成

在实践的时候，有一个总是能提升神经网络几个百分点准确率的办法，就是在训练的时候训练几个独立的模型，然后在测试的时候平均它们预测结果。集成的模型数量增加，算法的结果也单调提升(但提升效果越来越少)。还有模型之间的差异度越大，提升效果可能越好。

进行集成有以下几种方法：

- **同一个模型，不同的初始化。**
- **在交叉验证中发现最好的模型。**
- **一个模型设置多个记录点。**
- **在训练的时候跑参数的平均值。**



★ 总结

训练一个神经网络需要：

- 利用小批量数据对实现进行梯度检查，还要注意各种错误。
- 进行合理性检查，确认初始损失值是合理的，在小数据集上能得到100%的准确率。
- 在训练时，跟踪损失函数值，训练集和验证集准确率，如果愿意，还可以跟踪更新的参数量相对于总参数量的比例(一般在1e − 3左右)，然后如果是对于卷积神经网络，可以将第一层的权重可视化。
- 推荐的两个更新方法是SGD+Nesterov动量方法，或者Adam方法。
- 随着训练进行学习率衰减。比如，在固定多少个周期后让学习率减半，或者当验证集准确率下降的时候。
- 使用随机搜索(不要用网格搜索)来搜索最优的超参数。分阶段从粗(比较宽的超参数范围训练1-5 个周期)到细(窄范围训练很多个周期)地来搜索。
- 进行模型集成来获得额外的性能提高。

斯坦福CS231n课程作业# 2简介

斯坦福CS231n课程作业# 2简介

斯坦福CS231n课程作业# 3简介

斯坦福CS231n课程作业# 3简介

CS231n课程笔记翻译：卷积神经网络笔记

CS231n 课程笔记翻译：卷积神经网络笔记

★ 卷积神经网络(CNNs / ConvNets)

卷积神经网络的结构基于一个假设，即输入数据是图像。

★ 结构概述

卷积神经网络针对输入全部是图像的情况，将结构调整得更加合理，获得了不小的优势。与常规神经网络不同，卷积神经网络的各层中的神经元是3维排列的：**宽度**、**高度**和深度(这里的**深度**指的是激活数据体的第三个维度，而不是整个网络的深度，整个网络的深度指的是网络的层数)。

层中的神经元将只与前一层中的一小块区域连接，而不是采取全连接方式。

卷积神经网络是由层组成的。每一层都有一个简单的API：用一些含或者不含参数的可导的函数，将输入的3D数据变换为3D的输出数据。

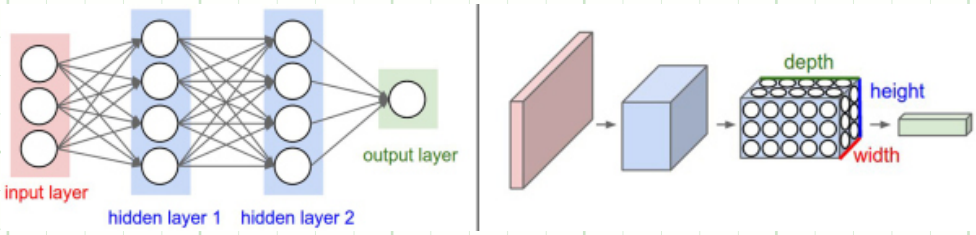


Figure 5: 左边是一个3层的神经网络。右边是一个卷积神经网络，图例中网络将它的神经元都排列成3个维度(宽、高和深度)。卷积神经网络的每一层都将3D的输入数据变化为神经元3D的激活数据并输出。在这个例子中，红色的输入层装的是图像，所以它的宽度和高度就是图像的宽度和高度，它的深度是3(代表了红、绿、蓝3种颜色通道)。

★ **用来构建卷积网络的各种层**

卷积神经网络主要由三种类型的层构成：**卷积层**，**汇聚(Pooling)层**和**全连接层**(全连接层和常规神经网络中的一样)。

★ **卷积层**

★ **汇聚层**

★ **归一化层**

★ **全连接层**

★ **把全连接层转化成卷积层**

★ **卷积神经网络的结构**

★ **层的排列规律**

★ **层的尺寸设置规律**

★ **案例学习**

★ **计算上的考量**

贺完结！CS231n官方笔记授权翻译总集篇发布

贺完结！CS231n官方笔记授权翻译总集篇发布