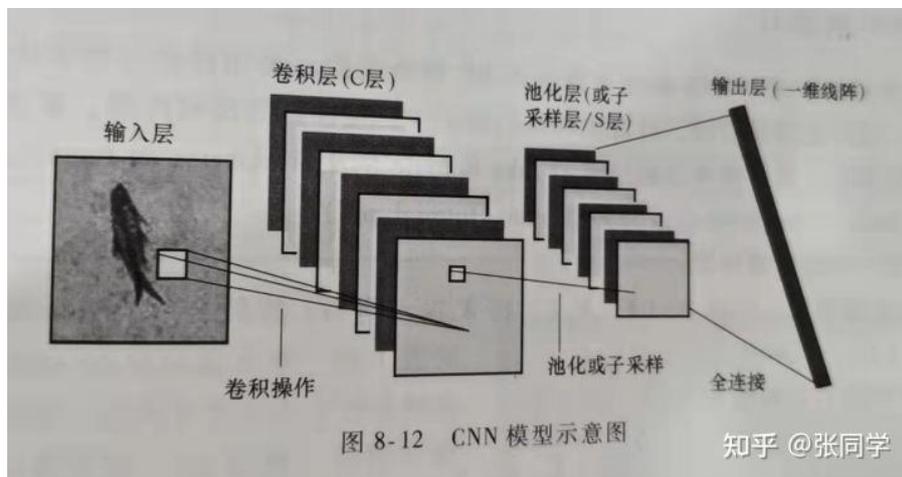


卷积神经网络

卷积神经网络主要由这几类层构成：输入层、卷积层、ReLU层、池化（Pooling）层和全连接层（全连接层和常规神经网络中的一样）。通过将这些层叠加起来，就可以构建一个完整的卷积神经网络。在实际应用中往往将卷积层与ReLU层共同称之为卷积层，所以卷积层经过卷积操作也是要经过激活函数的。

卷积层的作用是提取图像的特征；池化层的作用是对特征进行抽样，可以使用较少训练参数，同时还可以减轻网络模型的过拟合程度。卷积层和池化层一般交替出现在网络中，称一个卷积层加一个池化层为一个特征提取过程，但是并不是每个卷积层后都会跟池化层，大部分网络只有三层池化层。网络的最后一般为1~2层全连接层，全连接层负责把提取的特征图连接起来，最后通过分类器得到最终的分类结果。

卷积神经网络是为识别二维形状（比如照片）而特殊设计的一个多层感知器，由输入层、隐藏层（卷积层、池化层）、输出层组成，隐藏层可以有很多层，每层由一个或多个二维平面组成，而每个平面由多个独立神经元组成。结构图如下：



循环神经网络

循环神经网络是一种人工神经网络，它的节点间的连接形成一个遵循时间序列的有向图

□ 核心思想

– 样本间存在顺序关系，每个样本和它之前的样本存在关联。通过神经网络在时序上的展开，我们能够找到样本之间的序列相关性

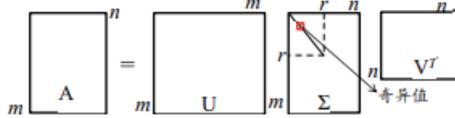
奇异值分解

□ 奇异值分解(Singular value decomposition)

- 对于任意矩阵 $A_{m \times n}$, 存在正交矩阵 $U_{m \times m}$, $V_{n \times n}$, 使得其满足

$$A=U\Sigma V^T \quad U^T U=V^T V=I$$

则称上式为矩阵A的特征分解, 其中 Σ 为 $m \times n$ 的矩阵



- 求解过程

- $A^T A$ 的特征值的 $\{\lambda_i\}$ 和特征向量 $\{v_i\}$
- AA^T 的特征向量 $\{u_i\}$
- $U=[u_1, \dots, u_m], V=[v_1, \dots, v_n], \Sigma=\text{diag}(\sqrt{\lambda_i})$

交叉熵

□ 交叉熵(cross entropy)

- 一般用来求目标与预测值之间的差距, 深度学习中经常用到的一类损失函数度量, 比如在对抗生成网络(GAN)中

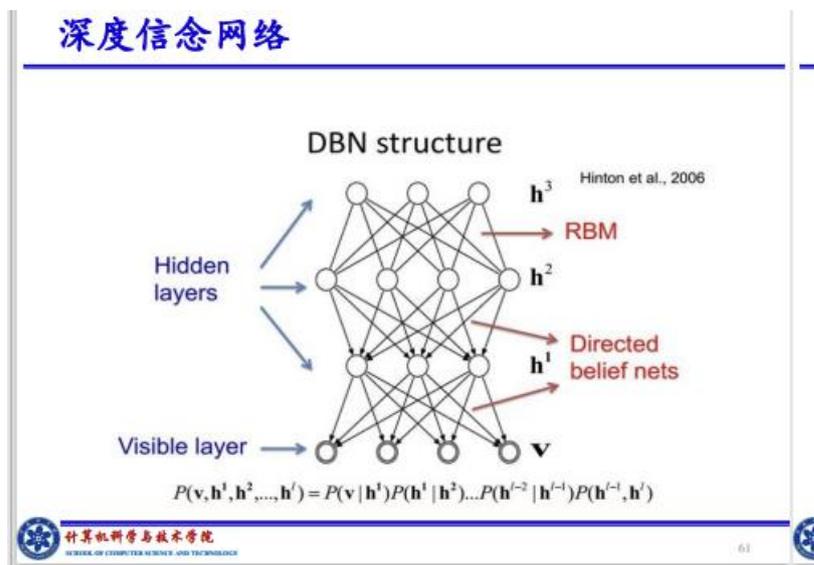
$$\begin{aligned} D(P||Q) &= \sum P(x) \log \frac{P(x)}{Q(x)} = \sum P(x) \log P(x) - \sum P(x) \log Q(x) \\ &= -H(P(x)) - \sum P(x) \log Q(x) \end{aligned}$$

交叉熵 $H(P, Q) = -\sum P(x) \log Q(x)$

深度信念网络

深度信念网络(Deep Belief Network, DBN)

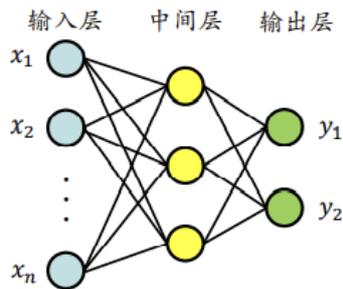
- The top two layers have undirected, symmetric connections between them and form an associative memory. The lower layers receive top-down, directed connections from the layer above. The states of the units in the lowest layer represent a data vector



多层感知器

□ 多层感知器指的是由多层结构的感知器递阶组成的输入值向前传播的网络，也被称为**前馈网络**或**正向传播网络**

- 以三层结构的多层感知器为例，它由**输入层**、**中间层**及**输出层**组成
 - 与M-P模型相同，中间层的感知器通过权重与输入层的各单元相连接，通过阈值函数计算中间层各单元的輸出值
 - 中间层与输出层之间同样是通过权重相连接



简答题：

反向传播算法

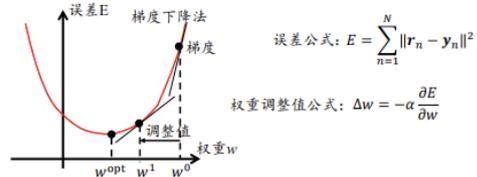
BP算法

- 多层感知器的训练使用**误差反向传播算法** (Error Back Propagation)，即**BP算法**
- BP算法最早由沃博斯于1974年提出，鲁梅尔哈特等人进一步发展了该理论
 - 基本过程：
 - 前向传播计算：由输入层经过隐层向输出层的计算网络输出
 - 误差反向逐层传递：网络的期望输出与实际输出之差的误差信号由输出层经过隐层逐层向输入层传递
 - 由“前向传播计算”与“误差反向逐层传递”的反复进行的网络训练过程



BP算法

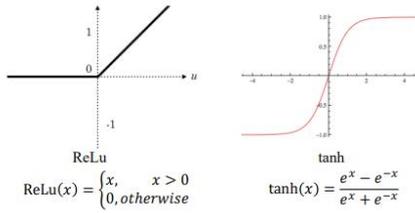
- 多层感知器的训练使用**误差反向传播算法** (Error Back Propagation)，即**BP算法**
- BP算法就是通过比较实际输出和期望输出得到误差信号，把误差信号从输出层逐层向前传播得到各层的误差信号，再通过调整各层的连接权重以减小误差。权重的调整主要使用梯度下降法



BP算法

□ 其他常用的激活函数

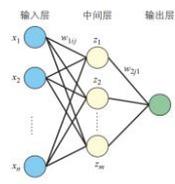
- ReLU (Rectified Linear Unit, 修正线性单元)和tanh等



BP算法

□ BP算法示例

- 以包含一个中间层和一个输出单元y的多层感知器为例: w_{1ij} 表示输入层与中间层之间的连接权重, w_{2j1} 表示中间层与输出层之间的连接权重, i 表示输入层单元, j 表示中间层单元



首先调整中间层与输出层之间的连接权重, 其中 $y=f(u)$, f 是激活函数, $u_{21} = \sum_{j=1}^m w_{2j1} z_j$, 把误差函数E对连接权重 w_{2j1} 的求导展开成复合函数求导:

$$\frac{\partial E}{\partial w_{2j1}} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial u_{21}} \frac{\partial u_{21}}{\partial w_{2j1}} = -(r-y)y(1-y)z_j$$

这里, z_j 表示的是中间层的值

BP算法

□ 第二, 中间层到输出层的连接权重调整值如下所示:

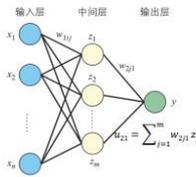
$$\Delta w_{2j1} = \alpha(r-y)y(1-y)z_j$$

BP算法

□ 第三, 调整输入层与中间层之间的连接权重

- 输入层与中间层之间的连接权重调整值是根据输出层的误差函数确定的, 求导公式如下所示:

$$\frac{\partial E}{\partial w_{1ij}} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial u_{21}} \frac{\partial u_{21}}{\partial w_{1ij}} = -(r-y)y(1-y) \frac{\partial u_{21}}{\partial w_{1ij}}$$



其中 u_{21} 由中间层的值 z_j 和连接权重 w_{2j1} 计算得到, 中间层与输出层单元之间的激活值 u_{21} 对中间层的值 z_j 求导, 结果只和连接权重 w_{2j1} 相关

$$\frac{\partial u_{21}}{\partial w_{1ij}} = \frac{\partial u_{21}}{\partial z_j} \frac{\partial z_j}{\partial w_{1ij}} \quad \frac{\partial u_{21}}{\partial z_j} = w_{2j1}$$

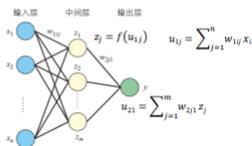
BP算法

□ 中间层的值 z_j 对连接权重 w_{1ij} 求导

$$\frac{\partial z_j}{\partial w_{1ij}} = \frac{\partial z_j}{\partial u_{1j}} \frac{\partial u_{1j}}{\partial w_{1ij}}$$

和 y 一样, z_j 也是sigmoid函数, 对 z_j 求导, 得到下式:

$$\frac{\partial z_j}{\partial u_{1j}} = z_j(1-z_j)$$



BP算法

□ 中间层的值 z_j 对连接权重 w_{1ij} 求导

$$\frac{\partial z_j}{\partial w_{1ij}} = \frac{\partial z_j}{\partial u_{1j}} \frac{\partial u_{1j}}{\partial w_{1ij}}$$

和 y 一样, z_j 也是sigmoid函数, 对 z_j 求导, 得到下式:

$$\frac{\partial z_j}{\partial u_{1j}} = z_j(1-z_j)$$

输入层与中间层单元之间的激活值 u_{1j} 对中间层与输出层之间的连接权重 w_{1ij} 求导, 结果只和 x_i 相关:

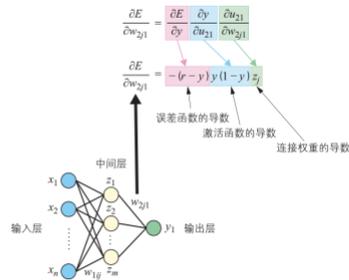
$$\frac{\partial u_{1j}}{\partial w_{1ij}} = x_j$$

综上, 输入层与中间层之间的连接权重调整值如下:

$$\Delta w_{1ij} = \alpha(r-y)y(1-y)w_{2j1}z_j(1-z_j)x_i$$

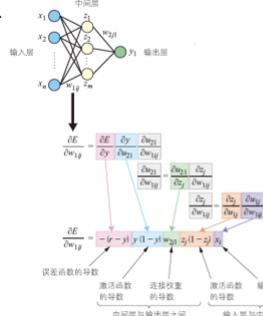
BP算法

□ 中间层到输出层



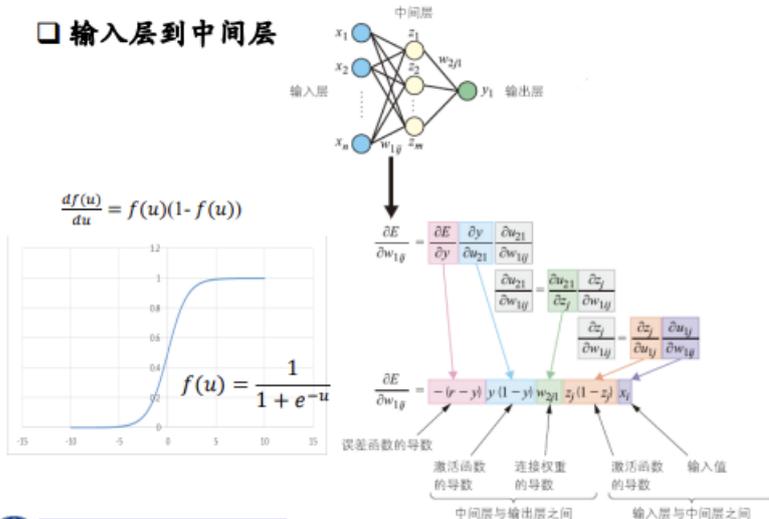
BP算法

□ 输入层到中间层



BP算法

□ 输入层到中间层



过拟合和欠拟合

<h3>误差分析</h3> <h4>□ 过拟合</h4> <ul style="list-style-type: none"> - 是指模型能很好地拟合训练样本，而无法很好地拟合测试样本的现象，从而导致泛化性能下降 - 为防止“过拟合”，可以选择减少参数、降低模型复杂度、正则化等 <p>Overfitting</p> <p>训练集误差小，测试集误差大</p>	<h3>误差分析</h3> <h4>□ 欠拟合</h4> <ul style="list-style-type: none"> - 是指模型还没有很好地训练出数据的一般规律，模型拟合程度不高的现象 - 为防止“欠拟合”，可以选择调整参数、增加迭代深度、换用更加复杂的模型等 <p>Underfitting</p> <p>训练集和测试集误差都比较大</p>
<h3>误差分析</h3> <h4>□ 模型训练的状态</h4> <ul style="list-style-type: none"> - 欠拟合、合适和过拟合的模型 <p>Underfitting</p> <p>Balanced</p> <p>Overfitting</p> <p>训练集和测试集误差都比较大</p> <p>训练集和测试集误差都较小</p> <p>训练集误差小 测试集误差大</p>	<h3>误差分析</h3> <h4>□ 泛化误差分析</h4> <ul style="list-style-type: none"> - 假设数据集上需要预测的样本为Y，特征为X，潜在模型为 $Y=f(X)+\epsilon$，其中 $\epsilon \sim N(0, \sigma_\epsilon)$ 是噪声，估计的模型为 $\hat{f}(X)$ $Err(f) = E[(Y - \hat{f}(X))^2]$ \Downarrow $Err(f) = Bias^2(f) + Var(f) + \sigma_\epsilon^2$ <p>泛化误差可分解为：偏差+方差</p>

误差分析

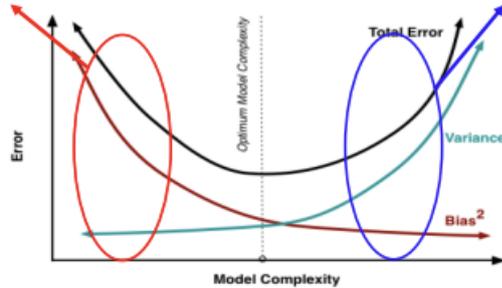
□ 泛化误差分析

欠拟合：高偏差低方差

- 寻找更好的特征，提升对数据的刻画能力
- 增加特征数量
- 重新选择更加复杂的模型

过拟合：低偏差高方差

- 增加训练样本数量
- 减少特征维数，高维空间密度小
- 加入正则化项，使得模型更加平滑



YOLO 算法的主要思想和实现过程

YOLO 系列

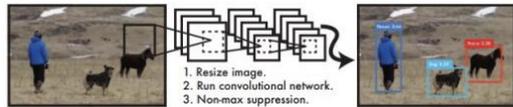
□ YOLO

- 与R-CNN系列最大的区别是用一个卷积神经网络结构就可以从输入图像直接预测bounding box和类别概率，实现了End2End训练
- 速度非常快，实时性好
- 可以学到物体的全局信息，背景误检率比R-CNN降低一半，泛化能力强
- 准确率还不如R-CNN高，小物体检测效果较差

Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. CVPR 2016: 779-788

YOLO 系列

□ 目标检测和识别



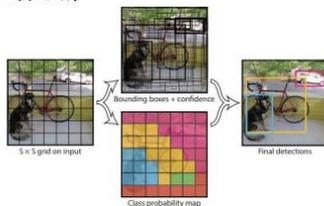
The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448 , (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence

通过模型的置信度对所产生的检测结果进行阈值控制。



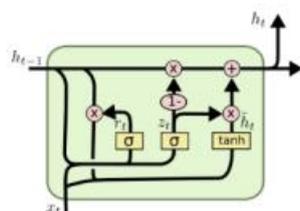
YOLO 系列

□ 目标检测和识别



The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B \cdot 5 + C)$ tensor.

GRU



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- x_t : input vector (输入向量)
- h_t : output vector (输出向量)
- z_t : update gate vector (更新门向量)
- r_t : reset gate vector (重置门向量)
- W : parameter matrices and vector (参数矩阵)
- σ : 一般选用Sigmoid函数

胶囊网络

CNN 现存问题

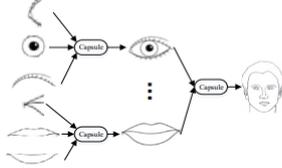
- 池化操作提供了局部不变性，错误解决了需要解决的等变性问题，从而丢失了位置等信息

胶囊网络的改进

- 使用胶囊作为网络基本单元
- 特征向量表示可视实体，对方位等信息进行编码
 - 特征向量表示可视实体
 - 实体的存在概率具有局部不变性——当胶囊覆盖的有限视觉域内的实体变换姿态时，它是不变的
 - 实体的实例化参数具有等变性——由于实例化参数表示实体的姿态坐标，因此随着实体的姿态变化，实例化参数会相应改变
- 动态路由算法代替池化操作

胶囊

- 胶囊输出的实例化参数提供了一种由部分到整体的简单识别方法，由单个有限视觉域逐层向上预测，最终形成对目标图像的解析树，达到识别目的。
- 在连接到下一层时，可以做出一个简单选择——实体是否能够由多个激活的具有正确的空间关系的胶囊表示，并且激活更高级别的胶囊



胶囊

- 设胶囊的输入为 u_i ，使用预测矩阵 W_{ij} 与 u_i 相乘，对高层特征向量进行预测，得到预测的特征向量 $u_{j|i}$

$$u_{j|i} = W_{ij} u_i$$

- 当前层胶囊对所有预测向量求取加权平均，便得到了向量 s_j

$$s_j = \sum_i c_{ij} u_{j|i} \quad c_{ij} \text{ 称作耦合系数}$$

胶囊

- 为了使特征向量的长度能够表示实体存在的概率，需要使用非线性压缩(squashing)函数将向量长度限制在0到1之间

$$v_j = \frac{\|s_j\|^2}{1 + \|s_j\|^2} \frac{s_j}{\|s_j\|}$$

胶囊与神经元对比

类型	输入	操作	输出	图
胶囊	$u_{j i} = W_{ij} u_i$	求和	$s_j = \sum_i c_{ij} u_{j i}$	
神经元	x_i	求和	$a_j = \sum_i W_{ij} x_i + b$	

胶囊间的动态路由

- 胶囊间的动态路由机制可以确保每一层胶囊输出的特征向量被正确地发送到下一层中对应的胶囊
- 耦合系数 c_{ij} 是使用Softmax函数来计算的，并且前一层的胶囊 i 与下一层中的所有胶囊 j 之间的耦合系数总和为1

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})}$$

胶囊间的动态路由

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})}$$

- 其中， b_{ij} 为胶囊 i 应该耦合到胶囊 j 的对数先验概率，它取决于两个胶囊的类型和位置，与当前的输入图像无关，其初始值为0，然后通过预测向量 $u_{j|i}$ 和实际输出向量 v_j 的一致性来修正
- 一个简单的表明一致性关系的函数就是标量积，在计算新的耦合系数之前，要把 $a_{ij} = v_j \cdot u_{j|i}$ 加到初始的 b_{ij} 上

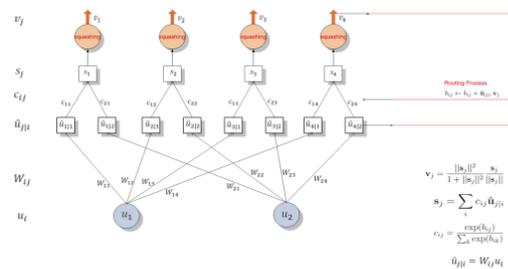
胶囊间的动态路由

- 特别注意的是其整个路由过程不仅仅是存在于胶囊网络的训练过程中，也存在于验证和测试过程，而且需要迭代多次

Procedure 1 Routing algorithm.

- 1: procedure ROUTING($\hat{u}_{j|i}, r, l$)
- 2: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
- 3: for r iterations do
- 4: for all capsule i in layer l : $c_i \leftarrow \text{softmax}(b_i)$
- 5: for all capsule j in layer $(l+1)$: $s_j \leftarrow \sum_i c_{ij} \hat{u}_{j|i}$
- 6: for all capsule j in layer $(l+1)$: $v_j \leftarrow \text{squash}(s_j)$
- 7: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i} \cdot v_j$
- return v_j

胶囊网络单层结构



$$v_j = \frac{\|s_j\|^2}{1 + \|s_j\|^2} \frac{s_j}{\|s_j\|}$$

$$s_j = \sum_i c_{ij} u_{j|i}$$

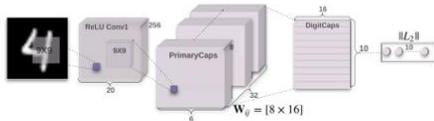
$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})}$$

$$\hat{u}_{j|i} = W_{ij} u_i$$

CapsNet

□ 网络基本结构

- 实现从主胶囊(8D)到数字胶囊(16D)的转换, 即: 低级特征向量向高级特征向量的转换
- 由于使用特征向量长度来表示对应类别存在概率, 所以在最后一层进行分类时, 需要将输出的特征向量取L2范数



- 256个步幅为1的9×9的卷积核
- 完成图像信息到低级特征的转换
- 实现低级特征到胶囊多维实体的(低级特征向量)的转换
- 具有32个通道的8D胶囊卷积层, 每个主胶囊具有8个步幅为2的9×9的卷积核的卷积单元

CapsNet

□ 边际损失 (margin loss)

- 为了实现同时对多个对象的识别, 每类目标对象对应的胶囊应分别使用边际损失函数得到类损失 L_k , 则总边际损失是所有类损失之和

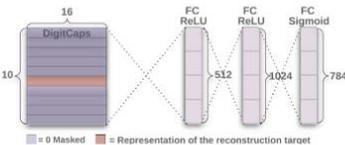
$$L_k = T_k \max(0, m^+ - \|v_k\|)^2 + \lambda(1 - T_k) \max(0, \|v_k\| - m^-)^2$$

- 其中, T_k 是表示 k 类目标对象是否存在, 当 k 类目标对象存在时为1, 不存在时为0
- m^+ 是上界, 一般取0.9
- m^- 是下界, 一般取0.1
- λ 为不存在的对象类别的损失的下调权重, 避免最开始从不存在分类对应的胶囊输出的特征向量中学习, 一般取0.5

CapsNet

□ 重构正则化

- 鲁棒性强的模型一定具有很强的重构能力。为了使胶囊对输入图像进行编码并输出对应实例化参数, 该网络引入了一个重构损失作为正则项。重构损失是逻辑单元的输出和像素强度之间的差的平方和



重构网络

- 重构loss的权重因子0.0005

实验结果

□ 手写数字识别

- 使用3次路由迭代
- (l, p, r) 代表label, 预测的和重构的
- 最右边是失败的例子

(l, p, r)	(2, 2, 2)	(5, 5, 5)	(8, 8, 8)	(9, 9, 9)	(5, 3, 5)	(5, 3, 3)
Input						
Output						

(l, p, r) : label, the prediction and reconstruction target

生成对抗网络

主要原理:

GAN基本原理

□ GAN起源于博弈论中的二人零和博弈 (获胜1, 失败-1)

- 由两个互为敌手的模型组成
 - 生成模型 (假币制造者团队)
 - 判别模型 (警察团队)
- 竞争使得两个团队不断改进他们的方法直到无法区分假币与真币



Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, Yoshua Bengio. Generative Adversarial Nets. NIPS 2014: 2672-2680

GAN 目标函数

GAN 目标函数

$$\min_D \max_G V(G, D) = E_{x \sim p_{data}(x)} [\log D(x)] + E_{z \sim p_z(z)} [\log (1 - D(G(z)))]$$

- 训练GAN的时候, 判别模型希望目标函数最大化, 也就是使判别模型判断真实样本为“真”, 判断生成样本为“假”的概率最大化, 要尽量最大化自己的判别准确率

判别模型也可以写成损失函数的形式

$$L(G, D) = -E_{x \sim p_{data}(x)} [\log D(x)] - E_{z \sim p_z(z)} [\log (1 - D(G(z)))]$$

GAN 损失函数

- 与之相反, 生成模型希望该目标函数最小化, 也就是降低判别模型对数据来源判断正确的概率, 要最小化判别模型的判别准确率

生成模型训练目标

$$V(G, D) = E_{x \sim p_{data}(x)} [\log D(x)] + E_{z \sim p_z(z)} [\log (1 - D(G(z)))]$$

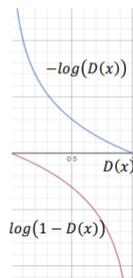
- 实际中效果并不好, 开始时梯度小收敛慢, 因此使用如下目标

$$V = E_{z \sim p_z(z)} [-\log (D(G(z)))]$$

GAN 损失函数

$$V = E_{x \sim p_g(x)} [-\log(D(x))]$$

$$V = E_{x \sim p_g(x)} [\log(1 - D(x))]$$



GAN 模型训练

- GAN在训练的过程中固定一方, 更新另一方的网络权重
- 交替迭代, 在这个过程中, 双方都极力优化自己的网络, 从而形成竞争对抗, 直到双方达到一个动态的平衡 (纳什均衡)
- 此时生成模型的数据分布无限接近训练数据的分布 (造出了和真实数据一模一样的样本), 判别模型再也判别不出来真实数据和生成数据, 准确率为 50%

计算题

Full 卷积

Same 卷积

Valid 卷积

姿势估计模型

[深度学习中的“人体姿势估计”全指南 - 云+社区 - 腾讯云 \(tencent.com\)](#)

<https://cloud.tencent.com/developer/article/1422841>

姿势估计之前好像是回归来做, 每个关节的位置坐标。

后来, 好像是热图去做, 热图表示关节在每个像素发生的概率

图像描述模型

图像描述旨在通过提取图像的特征输入到语言生成模型中最后输出图像对应的描述, 来解决人工智能中自然语言处理与计算机视觉的交叉领域问题——智能图像理解。

[\(27 条消息\) 图像描述 \(image caption\) 历年突破性论文总结_清晨的光明的博客-CSDN 博客_图像描述](#)

<https://blog.csdn.net/kdongyi/article/details/100738678>

一、为什么会产生梯度消失和梯度爆炸？

目前优化神经网络的方法都是基于 BP，即根据损失函数计算的误差通过梯度反向传播的方式，指导深度网络权值的更新优化。其中将误差从未层往前传递的过程需要**链式法则 (Chain Rule)** 的帮助，因此反向传播算法可以说是梯度下降在链式法则中的应用。

而链式法则是一个**连乘的形式**，所以当层数越深的时候，梯度将以指数形式传播。梯度消失问题和梯度爆炸问题一般随着网络层数的增加会变得越来越明显。在根据损失函数计算的误差通过梯度**反向传播**的方式对深度网络权值进行更新时，得到的**梯度值接近 0 或特别大**，也就是**梯度消失或爆炸**。梯度消失或梯度爆炸在本质原理上其实是一样的。

二、分析产生梯度消失和梯度爆炸的原因

【梯度消失】经常出现，产生的原因有：一是在**深层网络**中，二是采用了**不合适的损失函数**，比如 sigmoid。当梯度消失发生时，接近于输出层的隐藏层由于其梯度相对正常，所以权值更新时也就相对正常，但是当越靠近输入层时，由于梯度消失现象，会导致靠近输入层的隐藏层权值更新缓慢或者更新停滞。这就导致在训练时，只等价于后面几层的浅层网络的学习。

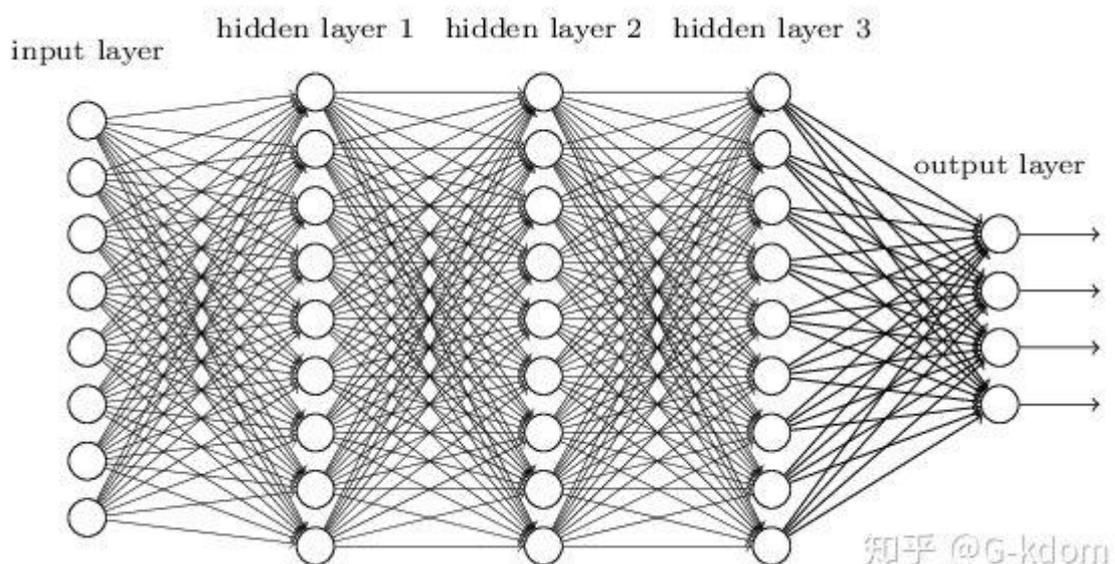
【梯度爆炸】一般出现在**深层网络**和**权值初始化值太大**的情况下。在深层神经网络或循环神经网络中，**误差的梯度可在更新中累积相乘**。如果网络层之间的**梯度值大于 1.0**，那么**重复相乘会导致梯度呈指数级增长**，梯度变的非常大，然后导致网络权重的大幅更新，并因此使网络变得不稳定。

梯度爆炸会伴随一些细微的信号，如：①模型不稳定，导致更新过程中的损失出现显著变化；②训练过程中，在极端情况下，权重的值变得非常大，以至于溢出，导致模型损失变成 NaN 等等。

下面将从这 3 个角度分析一下产生这两种现象的根本原因

(1) 深层网络

一个比较简单的深层网络如下：



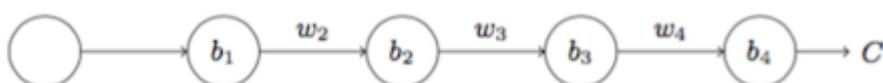
由于深度网络是多层非线性函数的堆砌，整个深度网络可以视为是一个**复合的非线性多元函数**（这些非线性多元函数其实就是每层的激活函数），那么对 loss function 求不同层的权值偏导，相当于应用梯度下降的链式法则，链式法则是一个连乘的形式，所以当层数越深的时候，梯度将以指数传播。

如果接近输出层的激活函数求导后梯度值大于 1，那么层数增多的时候，最终求出的梯度很容易指数级增长，就会产生**梯度爆炸**；相反，如果小于 1，那么经过链式法则的连乘形式，也会很容易衰减至 0，就会产生**梯度消失**。

从深层网络角度来讲，不同的层学习的速度差异很大，表现为网络中靠近输出的层学习的情况很好，靠近输入的层学习的很慢，有时甚至训练了很久，前几层的权值和刚开始随机初始化的值差不多。因此，梯度消失、爆炸，其根本原因在于反向传播训练法则，属于先天不足。

(2) 激活函数

以下图的反向传播为例（假设每一层只有一个神经元且对于每一层，其中 σ 为 sigmoid 函数）

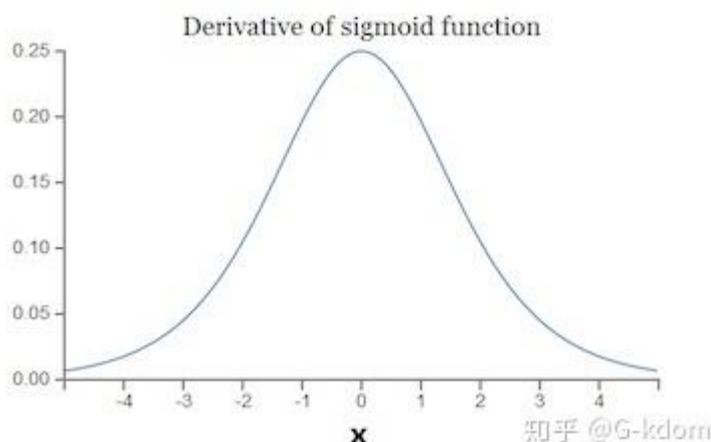


可以推导出：

$$\begin{aligned}\frac{\partial C}{\partial b_1} &= \frac{\partial C}{\partial y_4} \frac{\partial y_4}{\partial z_4} \frac{\partial z_4}{\partial x_4} \frac{\partial x_4}{\partial z_3} \frac{\partial z_3}{\partial x_3} \frac{\partial x_3}{\partial z_2} \frac{\partial z_2}{\partial x_2} \frac{\partial x_2}{\partial z_1} \frac{\partial z_1}{\partial b_1} \\ &= \frac{\partial C}{\partial y_4} \sigma'(z_4) w_4 \sigma'(z_3) w_3 \sigma'(z_2) w_2 \sigma'(z_1)\end{aligned}$$

知乎 @G-kdom

原因看下图，sigmoid 导数的图像。



如果使用 sigmoid 作为损失函数，其梯度是不可能超过 0.25 的，而我们初始化的网络权值通常都小于 1，因此，因此对于上面的链式求导，层数越多，求导结果越小，因而很容易发生梯度消失。

(3) 初始化权重的值过大

$$\begin{aligned} \frac{\partial C}{\partial b_1} &= \frac{\partial C}{\partial y_4} \frac{\partial y_4}{\partial z_4} \frac{\partial z_4}{\partial x_4} \frac{\partial x_4}{\partial z_3} \frac{\partial z_3}{\partial x_3} \frac{\partial x_3}{\partial z_2} \frac{\partial z_2}{\partial x_2} \frac{\partial x_2}{\partial z_1} \frac{\partial z_1}{\partial b_1} \\ &= \frac{\partial C}{\partial y_4} \sigma'(z_4) w_4 \sigma'(z_3) w_3 \sigma'(z_2) w_2 \sigma'(z_1) \end{aligned}$$

知乎 @G-kdom

如上图所示，当 w_4, w_3, w_2 比较大的情况。根据链式相乘(反向传播)可得，则前面的网络层比后面的网络层梯度变化更快，很容易发生梯度爆炸的问题。

三、解决方法

梯度消失和梯度爆炸问题都是因为网络太深，网络权值更新不稳定造成的，本质上是因为梯度反向传播中的连乘效应。解决梯度消失、爆炸主要有以下几种方法：

(1) pre-training+fine-tuning

此方法来自 Hinton 在 2006 年发表的一篇论文，Hinton 为了解决梯度的问题，提出采取无监督逐层训练方法，其基本思想是每次训练一层隐节点，训练时将上一层隐节点的输出作为输入，而本层隐节点的输出作为下一层隐节点的输入，此过程就是逐层“预训练” (pre-training)；在预训练完成后，再对整个网络进行“微调” (fine-

tuning)。此思想相当于是先寻找局部最优，然后整合起来寻找全局最优，此方法有一定的好处，但是目前应用的不是很多了。

(2) 梯度剪切：对梯度设定阈值

梯度剪切这个方案主要是针对梯度爆炸提出的，其思想是设置一个梯度剪切阈值，然后更新梯度的时候，如果梯度超过这个阈值，那么就将其强制限制在这个范围之内。这可以防止梯度爆炸。

(3) 权重正则化

另外一种解决梯度爆炸的手段是采用权重正则化 (weights regularization)，正则化主要是通过对网络权重做正则来限制过拟合。如果发生梯度爆炸，那么权值就会变的非常大，反过来，通过正则化项来限制权重的大小，也可以在一定程度上防止梯度爆炸的发生。比较常见的是 L1 正则和 L2 正则，在各个深度框架中都有相应的 API 可以使用正则化。

关于 L1 和 L2 正则化的详细内容可以参考我之前的文章——[欠拟合、过拟合及如何防止过拟合](#)

(4) 选择 relu 等梯度大部分落在常数上的激活函数

relu 函数的导数在正数部分是恒等于 1 的，因此在深层网络中使用 relu 激活函数就不会导致梯度消失和爆炸的问题。

关于 relu 等激活函数的详细内容可以参考我之前的文章——[温故知新——激活函数及其各自的优缺点](#)

(5) batch normalization

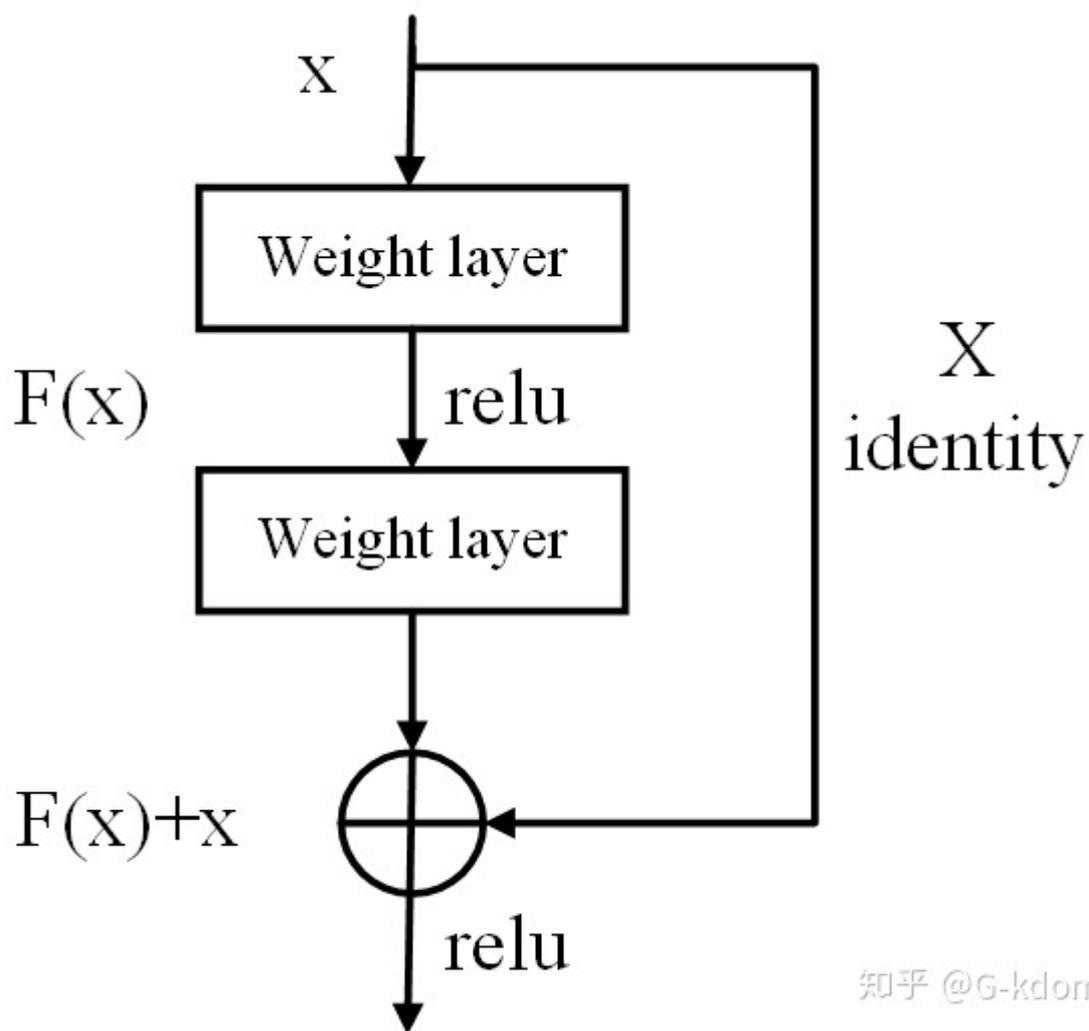
BN 就是通过对每一层的输出规范为均值和方差一致的方法，消除了权重参数放大缩小带来的影响，进而解决梯度消失和爆炸的问题，或者可以理解为 BN 将输出从饱和区拉倒了非饱和区。

关于 Batch Normalization (BN) 的详细内容可以参考我之前的文章——[常用的 Normalization 方法: BN、LN、IN、GN](#)

(6) 残差网络的捷径 (shortcut)

说起残差结构的话，不得不提这篇论文了：Deep Residual Learning for Image Recognition。论文链接：

openaccess.thecvf.com/_____

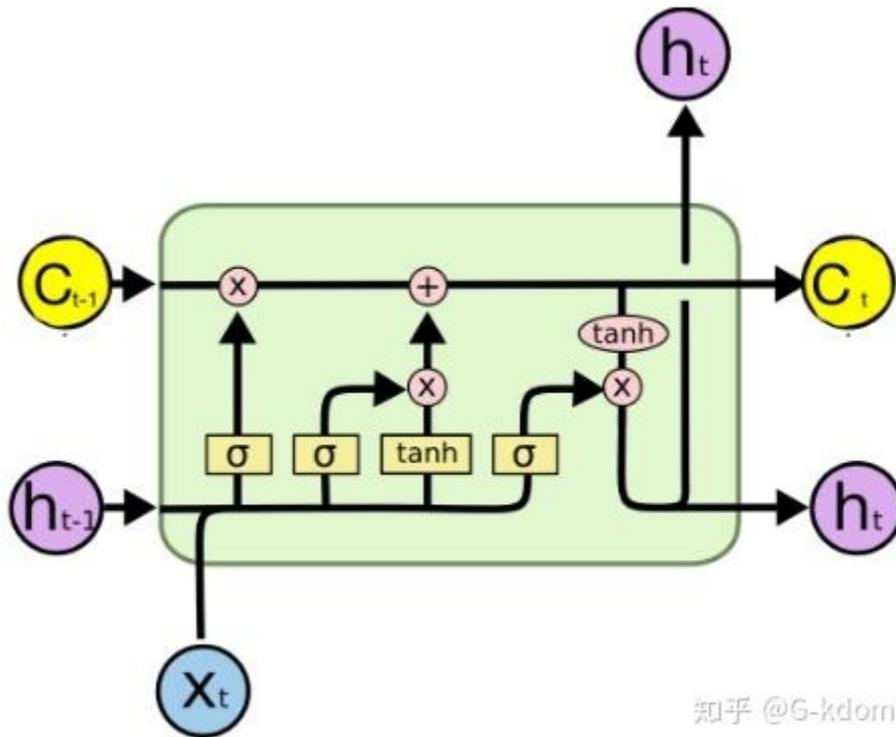


知乎 @G-kdom

相比较于以前直来直去的网络结构，残差中有很多这样（如上图所示）的跨层连接结构，这样的结构在反向传播中具有很大的好处，可以避免梯度消失。

(7) LSTM 的“门 (gate)”结构

LSTM 全称是长短期记忆网络 (long-short term memory networks)，LSTM 的结构设计可以改善 RNN 中的梯度消失的问题。主要原因在于 LSTM 内部复杂的“门” (gates)，如下图所示。



知乎 @G-kdorn

LSTM 通过它内部的“门”可以在接下来更新的时候“记住”前几次训练的“残留记忆”。

需要打印的网址：[简单易懂的 softmax 交叉熵损失函数求导 - 简书 \(jianshu.com\)](https://www.jianshu.com/p/c02a1fbffad6)

<https://www.jianshu.com/p/c02a1fbffad6>

文本生成图像：

[\(28 条消息\) 文本生成图像简要回顾 text to image synthesis mohole zhang 的博客-CSDN 博客_文本生成图片](https://blog.csdn.net/mohole_zhang/article/details/89374420)

https://blog.csdn.net/mohole_zhang/article/details/89374420

https://blog.csdn.net/Hekena/article/details/124959676?csdn_share_tail=%7B%22type%22%3A%22blog%22%2C%22rType%22%3A%22article%22%2C%22rid%22%3A%22124959676%22%2C%22source%22%3A%22Hekena%22%7D&ctrid=3xxWq