神经网络——
深度学习初步

主讲：刘丽珏

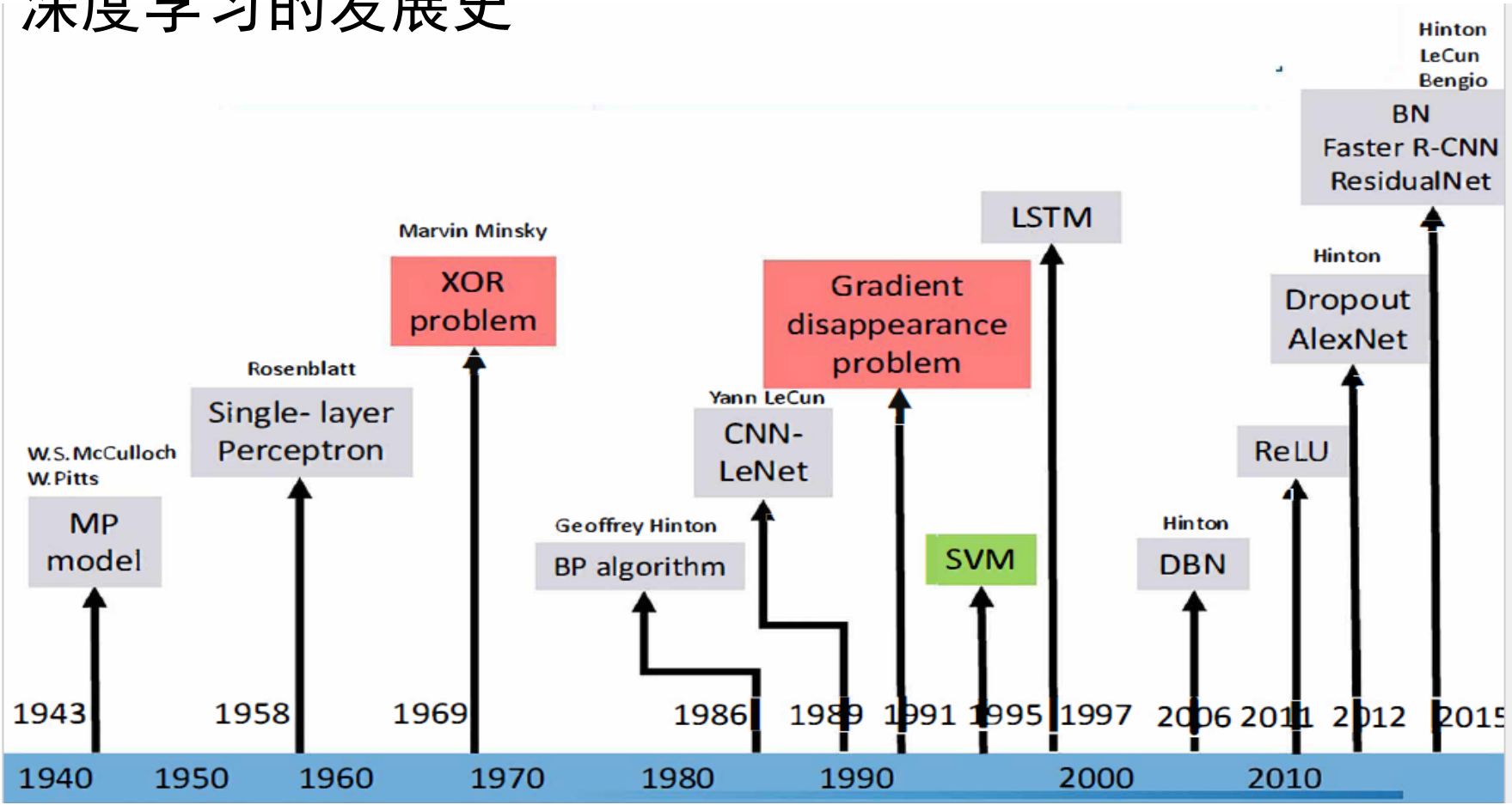
反向传播算法的讨论

- 每个有界的连续函数可以由两层网络以任意小误差逼近
- 任意函数可以被三层网络以任意精度逼近
- 在许多情形中深度2就足够表示任何一个带有给定目标精度的函数，但是
  - 图中所需要的节点数(比如计算和参数数量)可能变的非常大
  - 所需要的节点数随着输入的大小呈指数增长
- 1991年BP算法被指出存在梯度消失问题
  - 误差梯度反向传递的过程中，梯度本来就小，误差梯度传到前层时几乎为0，因此无法对前层进行有效的学习，该问题直接阻碍了多层神经网络的进一步发展
  - **大脑有一个深度架构**
  - 大脑中的表示是在中间紧密分布并且纯局部，是稀疏的，只有1%的神经元同时活动
- 如何解决梯度消失问题，构建深度神经网络（DNN）？

深度学习（Deep Learning）

- 机器学习的分支
- 由一系列算法组成，为数据建立高度抽象的模式
- 优点
  - 自动的特征提取
- 应用领域
  - Computer vision
  - Speech recognition
  - Natural language processing

深度学习的发展史


W.S. McCulloch
W. Pitts
MP model
1943
Rosenblatt
Single- layer Perceptron
1958
Marvin Minsky
XOR problem
1969
Geoffrey Hinton
BP algorithm
1986
Yann LeCun
CNN-LeNet
1989
Gradient disappearance problem
1991
SVM
1995
LSTM
1997
Hinton
DBN
2006
ReLU
2011
Hinton
Dropout
AlexNet
2012
Hinton
LeCun
Bengio
BN
Faster R-CNN
ResidualNet
2015
1940
1950
1960
1970
1980
1990
2000
2010

深度学习的发展史

- 1943: Neural networks
- 1957-62: Perceptron
- 1970-86: Backpropagation, RBM, RNN
- 1979-98: CNN, MNIST, LSTM, Bidirectional RNN
- 2006: “Deep Learning”, DBN
- 2009: ImageNet + AlexNet
- 2014: GANs
- 2016-17: AlphaGo, AlphaZero
- 2017-19: Transformers

深度学习（Deep Learning）的发展史

- **发展期 2006年 - 2012年**
  - 2006年，加拿大多伦多大学教授、机器学习领域泰斗、神经网络之父——Geoffrey Hinton 和他的学生 Ruslan Salakhutdinov 在顶尖学术刊物《科学》上发表了一篇文章，提出了深层网络训练中梯度消失问题的解决方案
    - **无监督预训练对权值进行初始化+有监督训练微调**
    - Hinton, G. E., Osindero, S. and Teh, Y.,A fast learning algorithm for deep belief nets.Neural Computation 18:1527-1554, 2006（DBN）
  - 2011年，ReLU激活函数被提出，该激活函数能够有效的抑制梯度消失问题
    - 2011年以来，微软首次将DL应用在语音识别上，取得了重大突破。微软研究院和Google的语音识别研究人员先后采用DNN技术降低语音识别错误率20%~30%，是语音识别领域十多年来最大的突破性进展

深度学习的发展史

- **爆发期 2012 - 2017**
  - 2012年，Hinton课题组为了证明深度学习的潜力，首次参加ImageNet图像识别比赛，其通过构建的CNN网络*AlexNet*一举夺得冠军，且碾压第二名（SVM方法）的分类性能
    - CNN吸引到了众多研究者的注意

三位图灵奖获得者Yann LeCun, Geoffrey Hinton , Yoshua Bengio

不同的图像识别任务

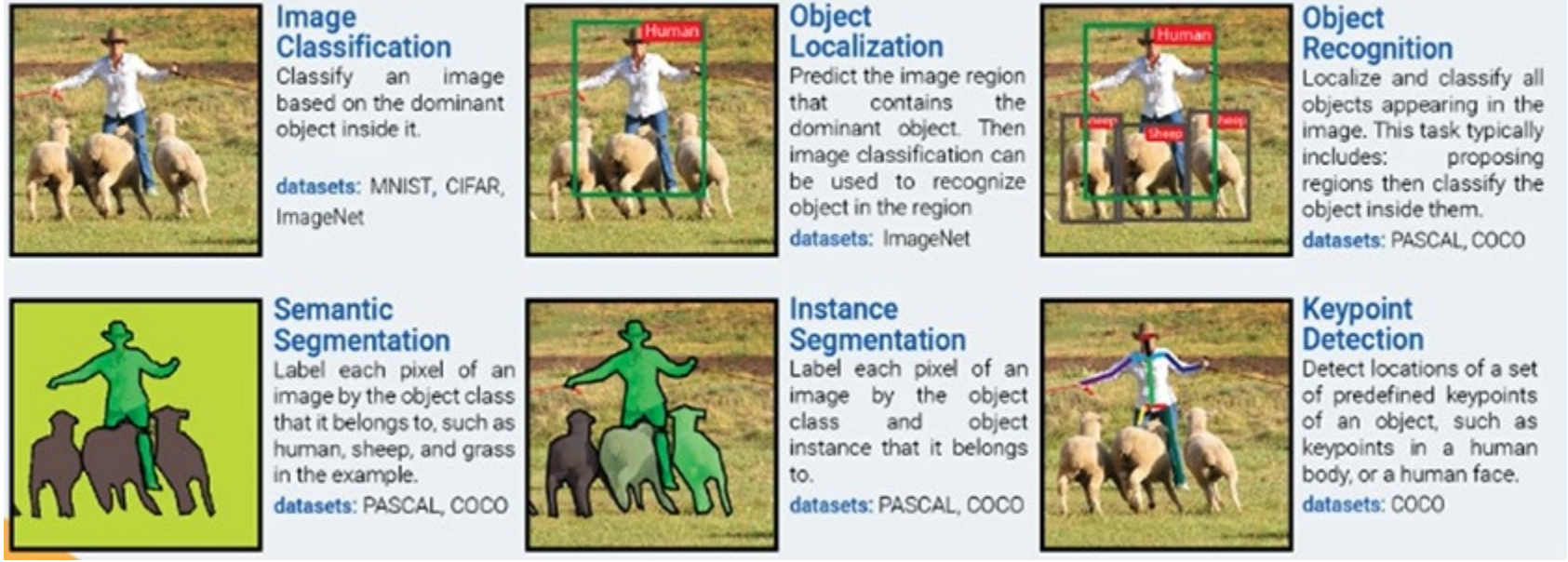
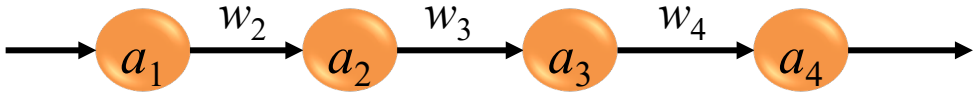


Image Classification
Classify an image based on the dominant object inside it.
datasets: MNIST, CIFAR, ImageNet
Human
Object Localization
Predict the image region that contains the dominant object. Then image classification can be used to recognize object in the region
datasets: ImageNet
Human
Sheep
Sheep
Sheep
Object Recognition
Localize and classify all objects appearing in the image. This task typically includes: proposing regions then classify the object inside them.
datasets: PASCAL, COCO
Semantic Segmentation
Label each pixel of an image by the object class that it belongs to, such as human, sheep, and grass in the example.
datasets: PASCAL, COCO
Instance Segmentation
Label each pixel of an image by the object class and object instance that it belongs to.
datasets: PASCAL, COCO
Keypoint Detection
Detect locations of a set of predefined keypoints of an object, such as keypoints in a human body, or a human face.
datasets: COCO

梯度消失



- 回顾一下BP网络，假设有如上简单的3隐层的网络
- w_2的更新值计算公式如下

$$\Delta w_2 = \frac{\partial Loss}{\partial w_2} = \frac{\partial Loss}{\partial a_4}\frac{\partial a_4}{\partial a_3}\frac{\partial a_3}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

其中$a_{i+1} = S(a_i * w_{i+1})$，$S$是$Sigmoid$函数

$$\frac{\partial a_2}{\partial w_2} = \frac{\partial S(a_1 * w_2)}{\partial w_2} = \frac{\partial S(a_1 * w_2)}{\partial (a_1 * w_2)}\frac{\partial (a_1 * w_2)}{\partial w_2} = S' * a_1 \quad （a_1是输入）$$

$$\frac{\partial a_3}{\partial a_2} = \frac{\partial S(a_2 * w_3)}{\partial a_2} = \frac{\partial S(a_2 * w_3)}{\partial (a_2 * w_3)}\frac{\partial (a_2 * w_3)}{\partial a_2} = S' * w_3, \qquad \frac{\partial a_4}{\partial a_3} = S' * w_4$$

即有 $\Delta w_2 = \frac{\partial Loss}{\partial a_4} * S' * w_4 * S' * w_3 * S' * a_1$，意味着$\Delta w$与$S'$的指数呈正比

梯度消失

- 已经得到

$$\Delta w_2 = \frac{\partial Loss}{\partial a_4} * S' * w_4 * S' * w_3 * S' * a_1$$

- 观察*Sigmoid*函数的导函数

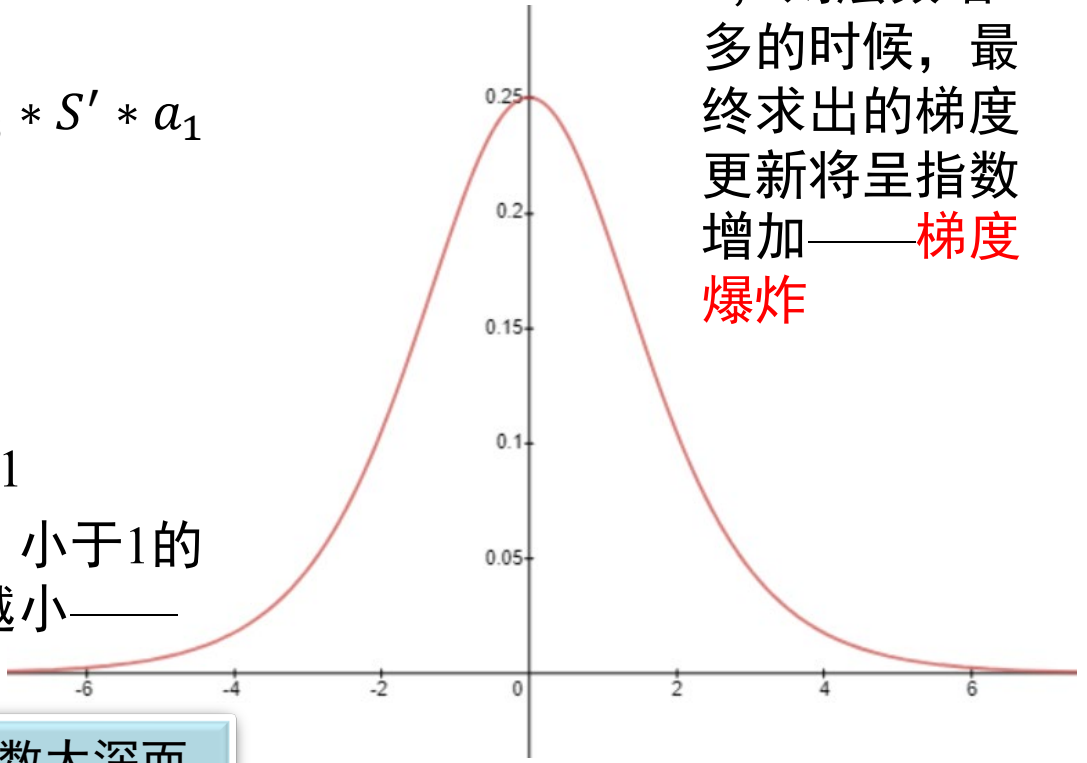
$$S' = S(1-S)$$

显然$rang(S') \in (0,0.25]$

- 初始化权值一般比较小，$|w|<1$
- 显然，越靠近输入层的隐层，小于1的参数连乘得越多，其更新值越小——梯度消失

梯度消失与梯度爆炸都是因为网络层数太深而引发的梯度反向传播中的连乘效应

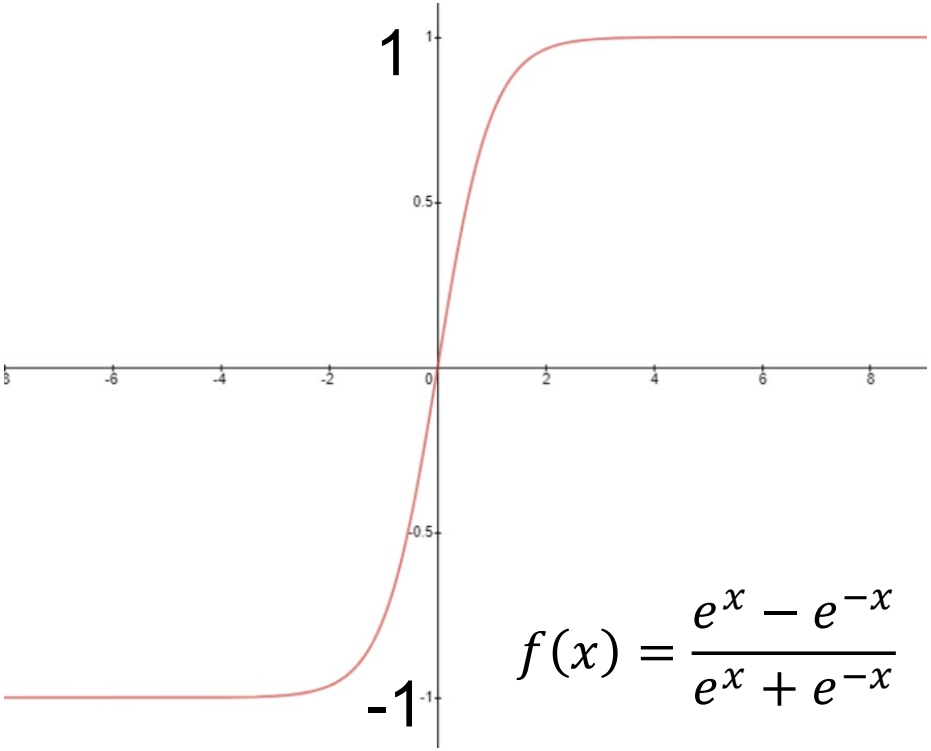
如果权值大于1，则层数增多的时候，最终求出的梯度更新将呈指数增加——梯度爆炸



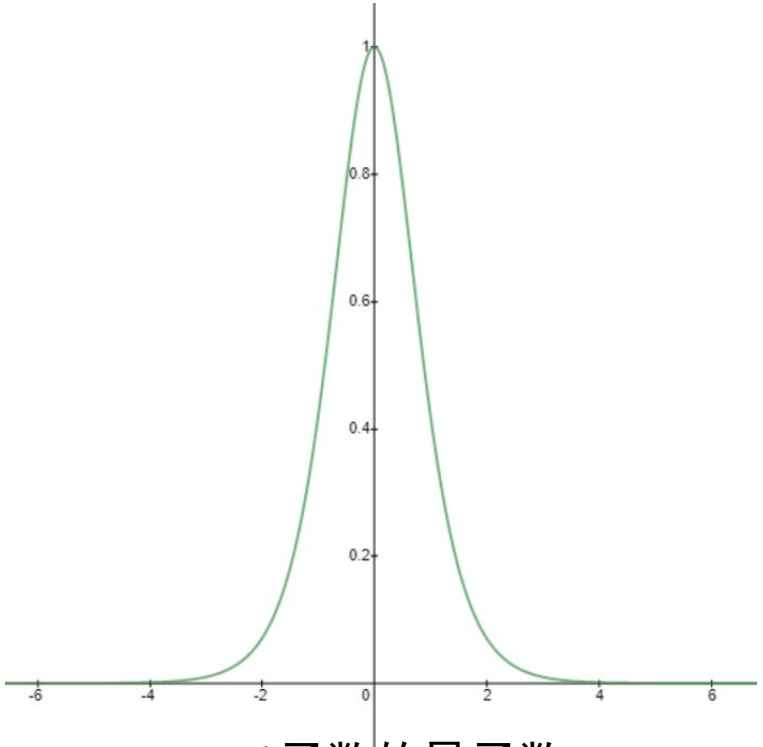
*Sigmoid*函数的导函数

其他激活函数

- *tanh*函数稍好一点，但也有类似的问题



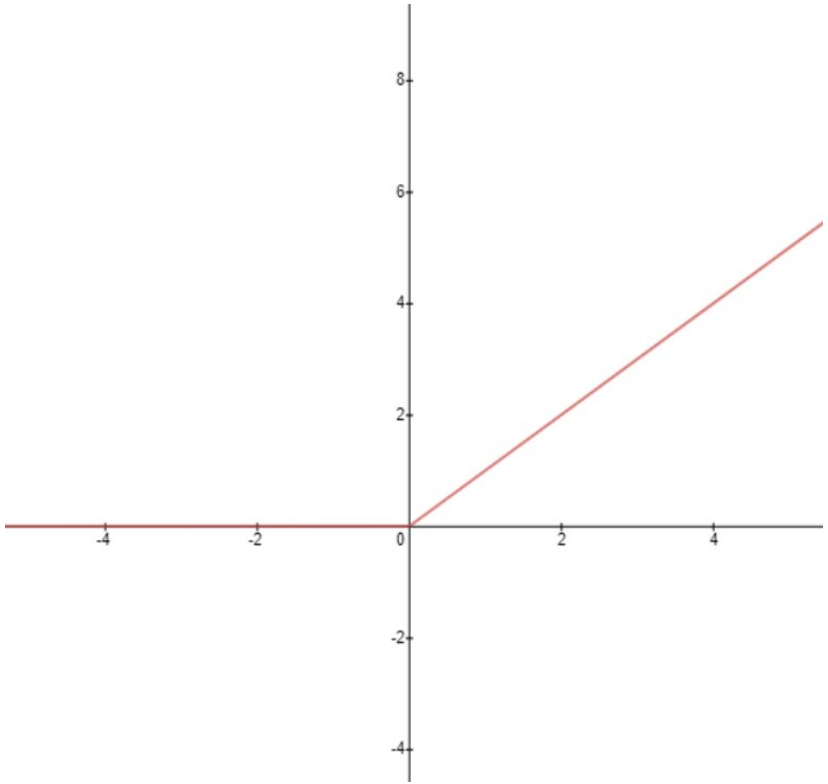
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



*tanh*函数的导函数

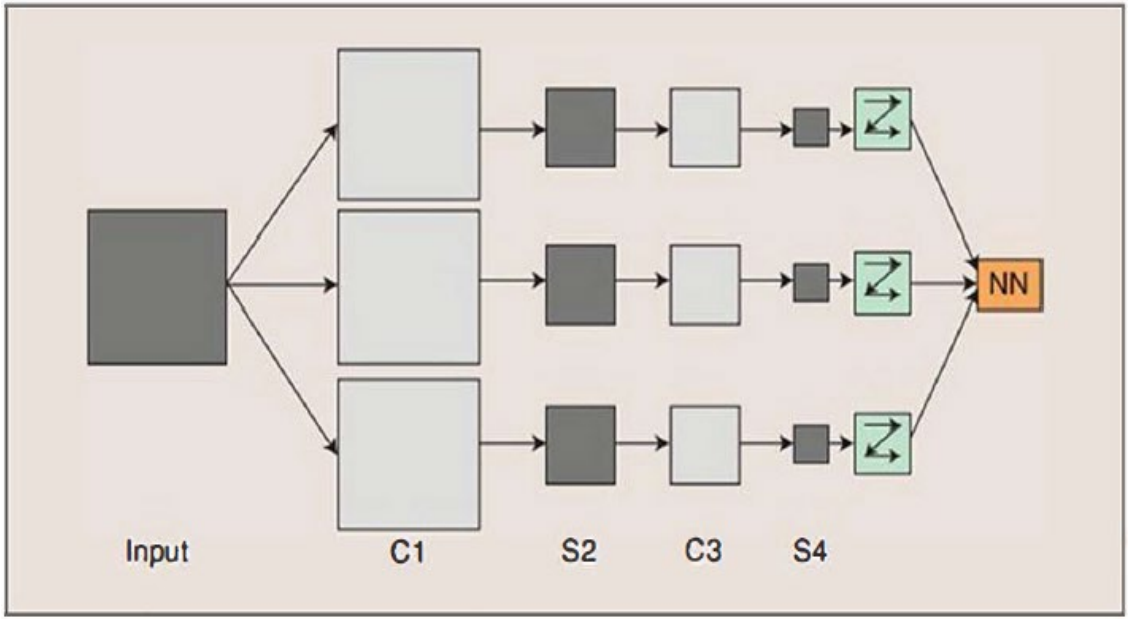
*ReLu*函数

- *ReLu*——自适应线性单元（Rectified linear unit）
- $ReLu$: $f(x) = \max(0, x)$
- 函数的导数在正数部分恒等于1
- *AlexNet*使用*ReLU*代替了传统的激活函数
- 现在*ReLU*已经广泛地使用在了各种CNN结构中

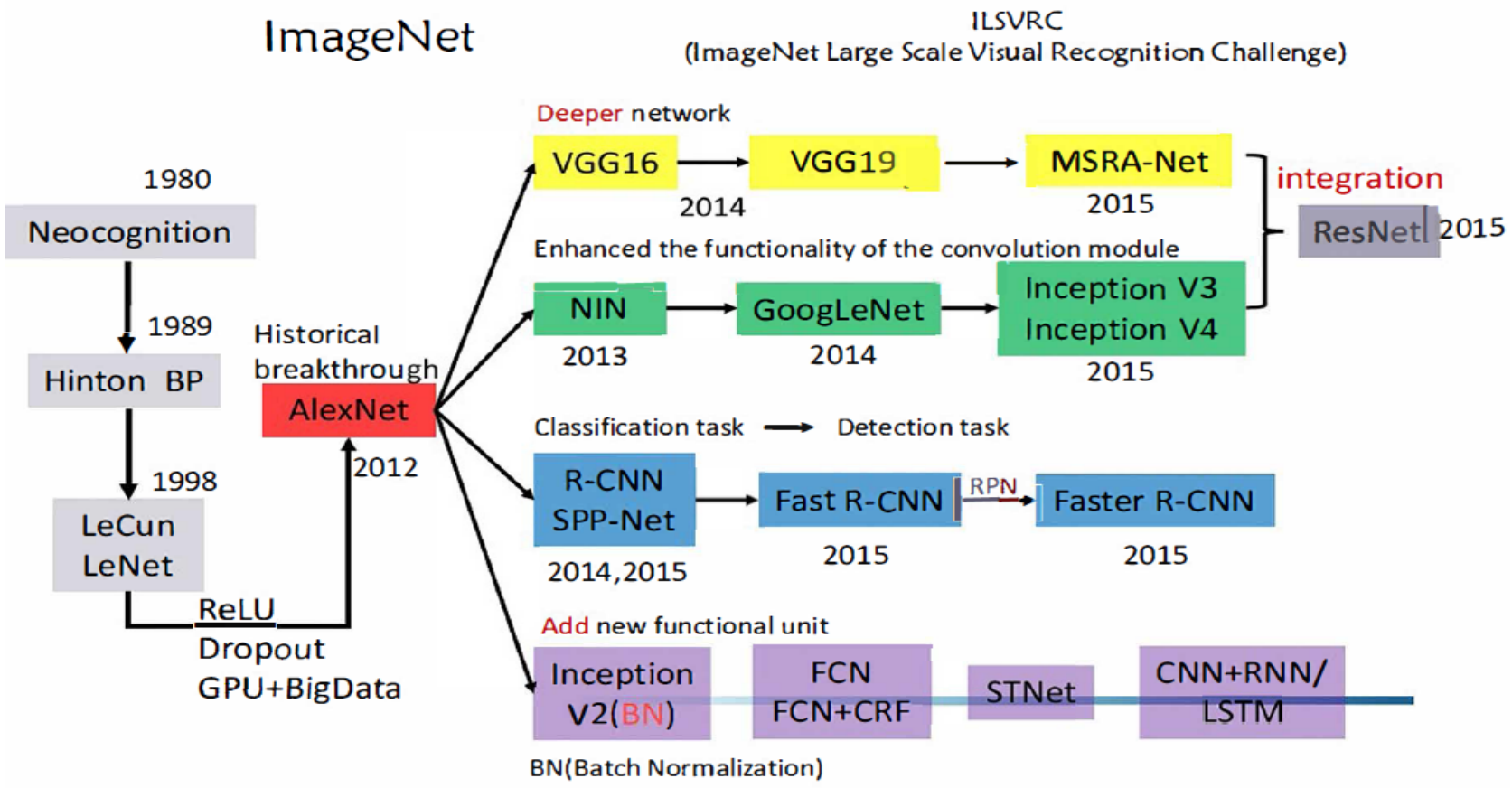


卷积神经网络(CNN)

- 一种前馈型神经网络
- 特点
  - 结构简单
  - 训练参数少
  - 高适应性
- 避免了复杂的图像预处理过程
  - 不需人工提取特征
  - 只需直接输入原始图像
- 基本组成部分
  - 卷积层（Convolution Layers）
  - 池化层（Pooling Layers）
  - 全连接层（Fully connected Layers）

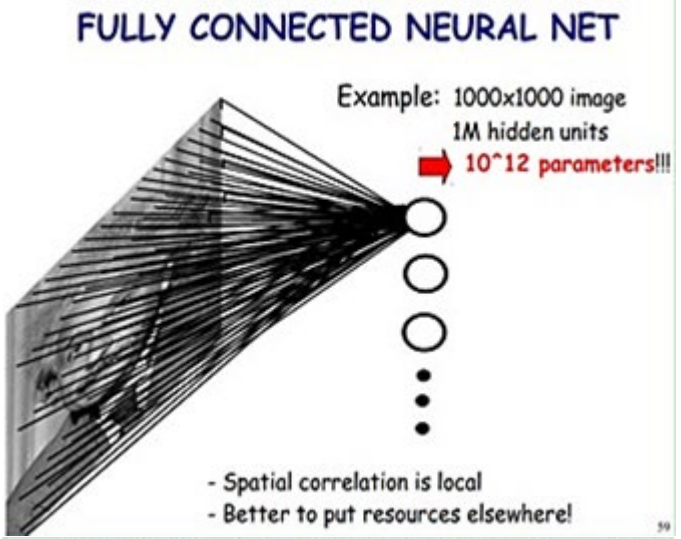


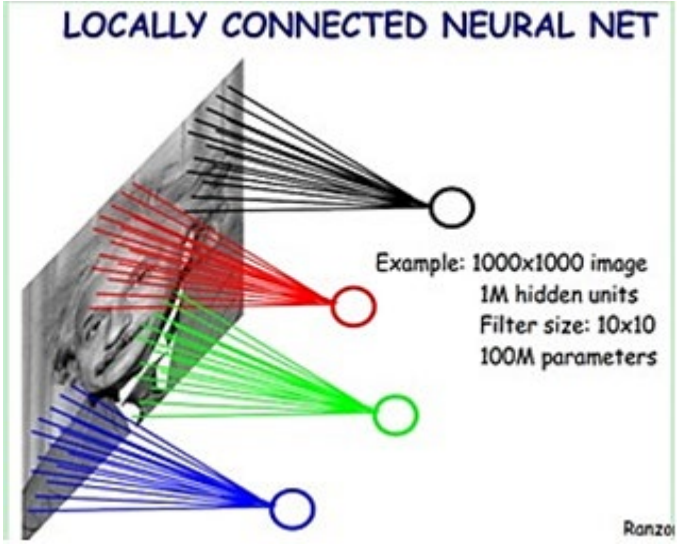
CNN结构发展史


ImageNet
ILSVRC
(ImageNet Large Scale Visual Recognition Challenge)
1980
Neocognition
1989
Hinton BP
1998
LeCun
LeNet
ReLU
Dropout
GPU+BigData
Historical
breakthrough
AlexNet
2012
Deeper network
VGG16
VGG19
MSRA-Net
2014
2015
integration
ResNet
2015
Enhanced the functionality of the convolution module
NIN
GoogLeNet
Inception V3
Inception V4
2013
2014
2015
Classification task
Detection task
R-CNN
SPP-Net
Fast R-CNN
RPN
Faster R-CNN
2014,2015
2015
2015
Add new functional unit
Inception
V2(BN)
FCN
FCN+CRF
STNet
CNN+RNN/
LSTM
BN(Batch Normalization)

卷积神经网络（CNN）

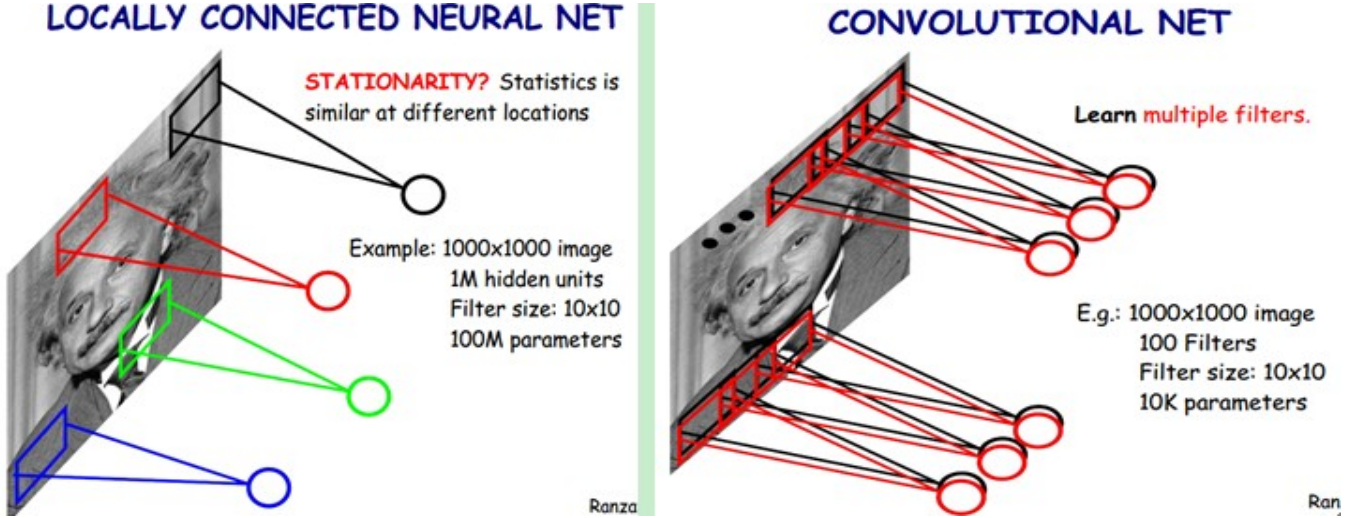
- 通过局部感知减少训练参数
- 图像的空间联系是局部的
  - 每个神经元不需要感受全局图像，只需要感受局部
  - 在更高层将感受不同局部的神经元综合起来得到全局信息
  - 这样可以减少连接的数目，即减少需要训练的权值参数的个数



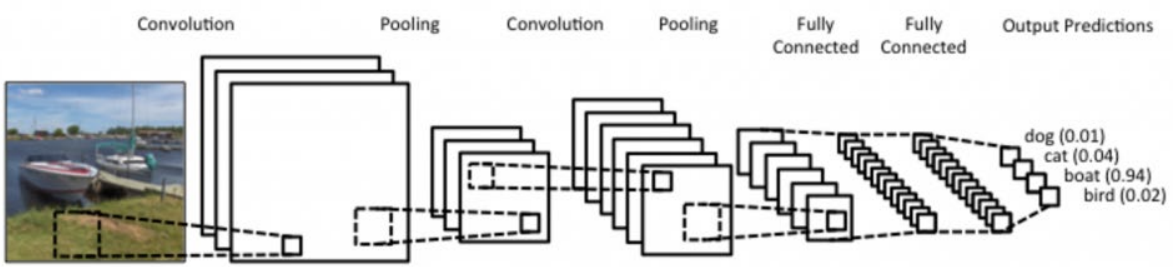


CNN

- 权值共享进一步减少训练的参数
  - 每个神经元连接10x10的图像区域，即每个神经元有100个连接权值参数
  - 如果每个神经元都使用相同的参数配置，即每个神经元用同一个卷积核，就只需要100个参数
  - 在实际问题中，用一个卷积核只能提取一种特征，可以用多个卷积核提取不同特征。



CNN经典结构层



- 卷积层：特征提取
  - 通过卷积运算，使原信号特征增强，降低噪音。
  - 卷积层特征图的大小由卷积核和上一层输入特征图的大小决定，在不考虑步长和扩充的情况下，假设输入特征图大小为$n*n$、卷积核大小为$k*k$，则该层特征图大小为$(n-k+1)*(n-k+1)$。
- 池化层：降采样
  - 根据图像局部相关性的原理，通常每个卷积层后面跟着一个实现局部平均和子抽样的计算层
  - 降低特征映射的分辨率，减少计算量，使特征映射的输出对平移、旋转等变形的敏感度降低。
- 全连接层：识别
  - 通常采用softmax全连接

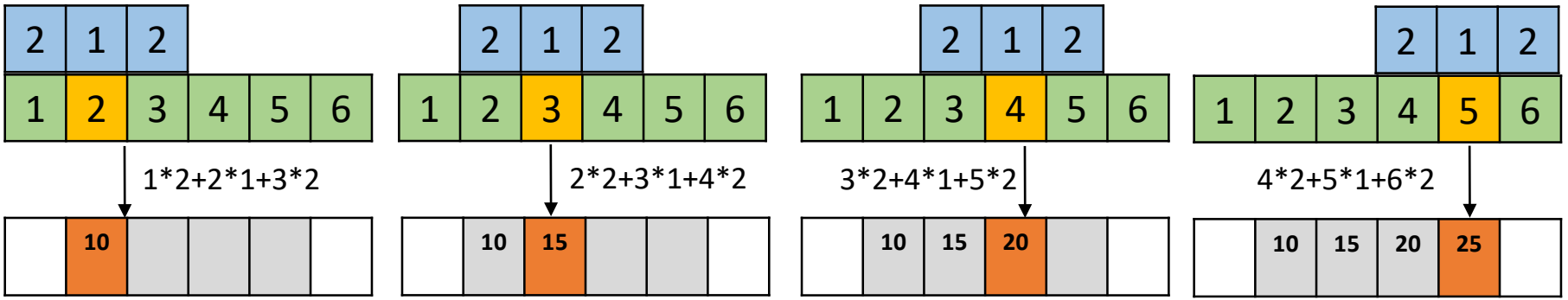
关于卷积

- 一维离散卷积操作：相乘后求和

$$y(n) = \sum_{i=-\infty}^{\infty} x(i)h(n-i)$$

- 例如：$x = [a_0, a_1, a_2, \ldots, a_k]$, $h = [b_0, b_1, b_2, \ldots, b_l]$

  则 $y(n) = a_0b_n + a_1b_{n-1} + a_2b_{n-2} + \ldots + a_nb_0$, ($n<k$且$n<l$)



关于卷积

- 二维离散卷积操作

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$
$$= \sum_m \sum_n I(i-m, j-n)K(m,n)(\text{卷积运算可交换})$$

其中I为输入，K为卷积核

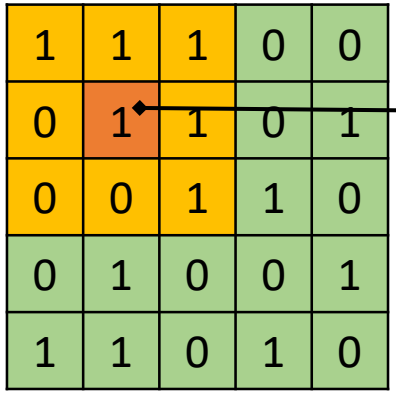
- 卷积运算可交换性的出现是因为上式中将核相对输入进行了翻转（flip），从m 增大的角度来看，输入的索引在增大，核的索引在减小
- 可交换性在证明时很有用，但在神经网络的应用中却不是一个重要的性质
- 许多神经网络库会实现一个相关的函数，称为互相关函数（cross-correlation），和卷积运算几乎一样但是并没有对核进行翻转

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

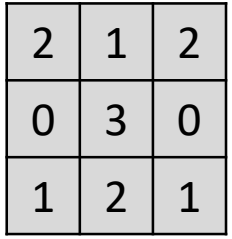
- 许多机器学习的库实现的是互相关函数但是称之为卷积

关于卷积

- 无翻转的二维离散卷积操作

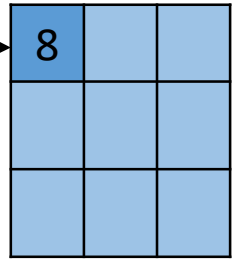


5x5



3x3 卷积核

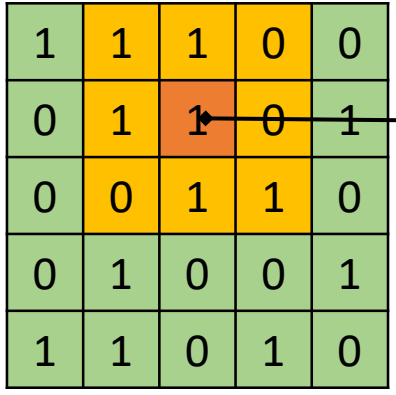
1*2+1*1+1*2+0*0+1*3+1*0+0*1+0*2+1*1



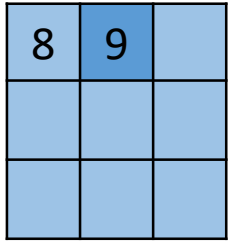
3x3 feature map

padding = 0

striding = 1



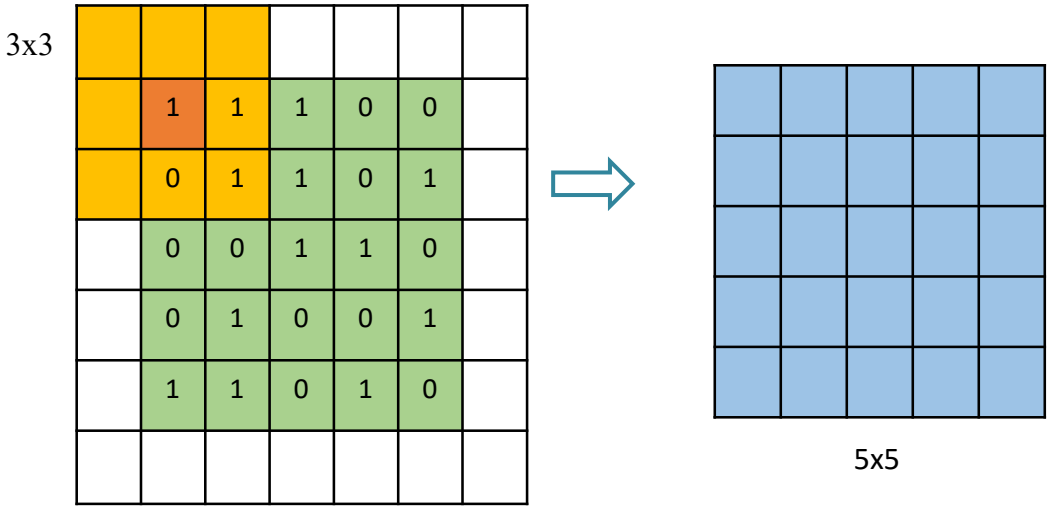
1*2+1*1+0*2+1*0+1*3+0*0+0*1+1*2+1*1



关于卷积

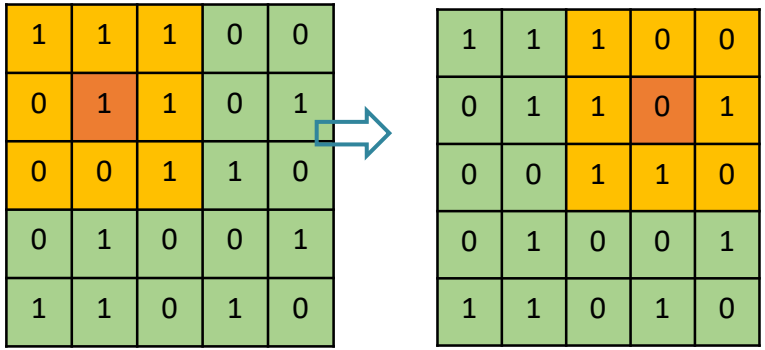
- padding（扩充）：

padding = 1

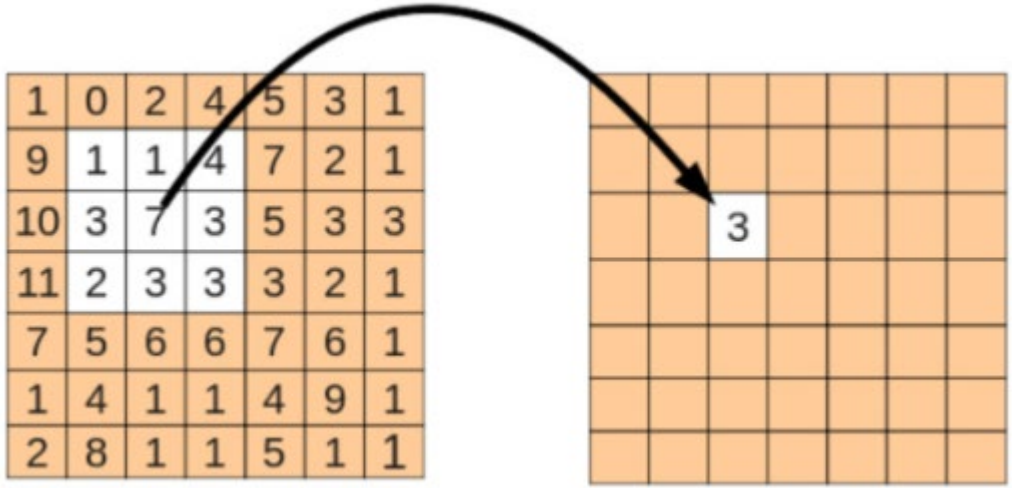


- striding（步长）：

striding = 2



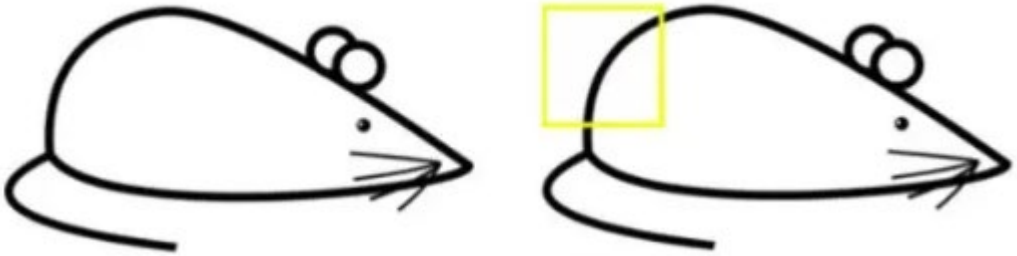
卷积的作用

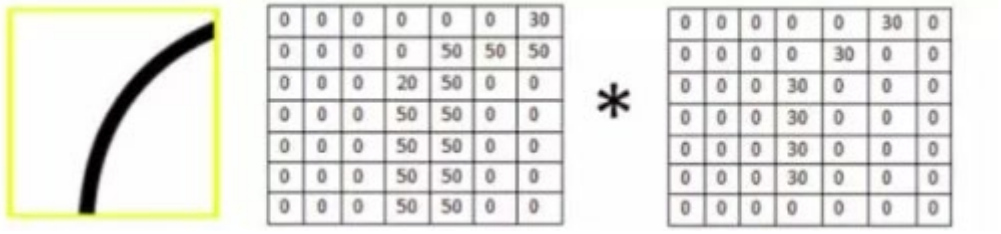


$$W = \frac{1}{9} * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- 相当于滤波

卷积的作用





0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

卷积核

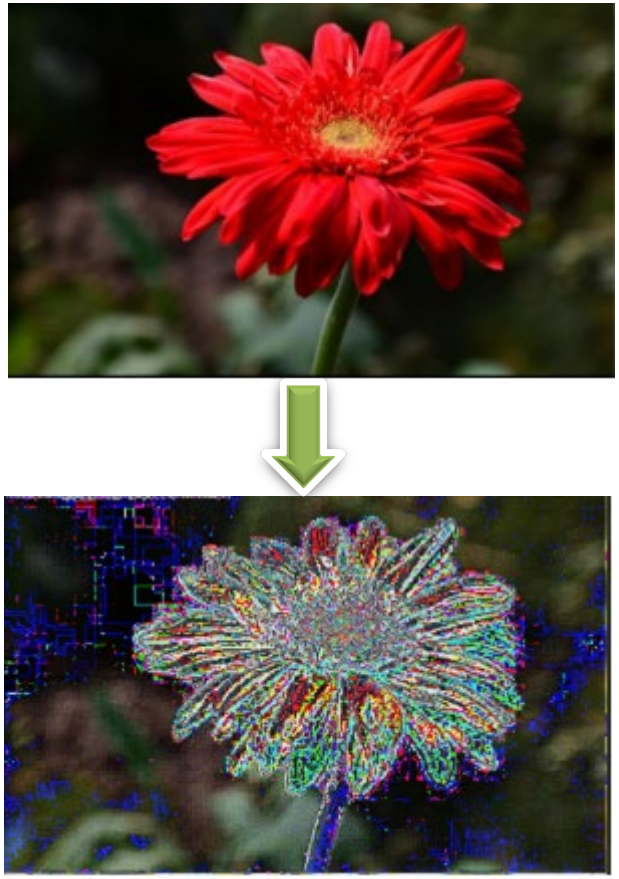
- 黄框区域的图像像素值与滤波器相乘，得到一个很大的值（6600）
- 滤波器移动到其他区域时，得到一个相对很小的值
- 对整个原图进行一次卷积，得到的结果中，在特定曲线和周边区域，值很高，在其他区域，值相对低

卷积的作用

- 卷积核

-1	-1	-1
-1	9	-1
-1	-1	-1

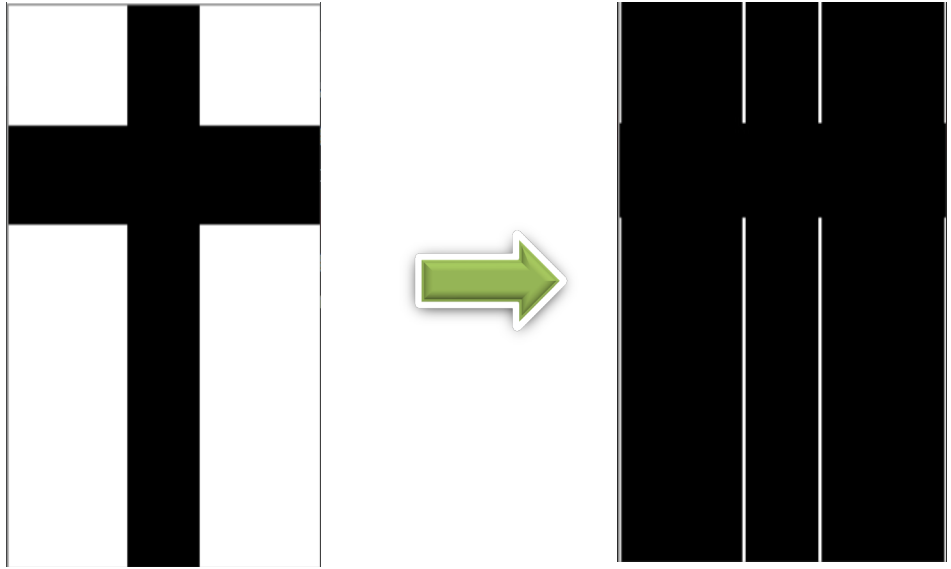
- 提取了边缘特征



卷积的作用

- 卷积核

-1	0	1
-1	0	1
-1	0	1



- 当沿X轴（水平）方向相邻的像素值一样时会相互抵消
- 增强了垂直边缘
- 水平梯度滤波器

思考题：如果我们想增强沿Y轴方向变化大的边缘——垂直梯度滤波器，卷积核应该是什么样的呢？

卷积的作用

- 卷积核的作用——特征提取
  - 相当于滤波
  - 滤除不需要的信息，增强有用的特征
- 卷积操作后的图称为——卷积特征图
- 卷积特征图的尺度
  - 对于一个N* N的原图像，通过填充P(padding)个像素，经过一个步长是S(stride)的F* F(filter)的卷积核，最终特征图的尺寸M为

$$M = \frac{N - F + 2P}{S} + 1$$

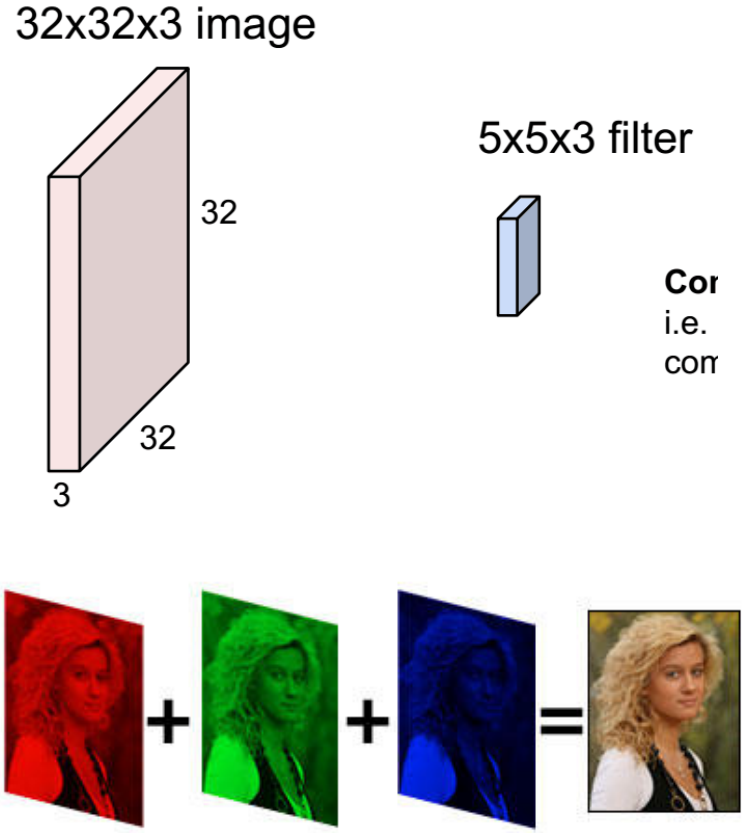
关于卷积

- 卷积特征用在图像上效果显著
- 几个问题
  - 卷积核到底应该设多大？
  - 设多少个？

  超参数（需要凭经验或实验不断调整尝试）
  - 每个卷积核具体应该是由哪些数组成？ 参数
- 深度神经网络通过机器学习算法来自动获取卷积核的具体参数

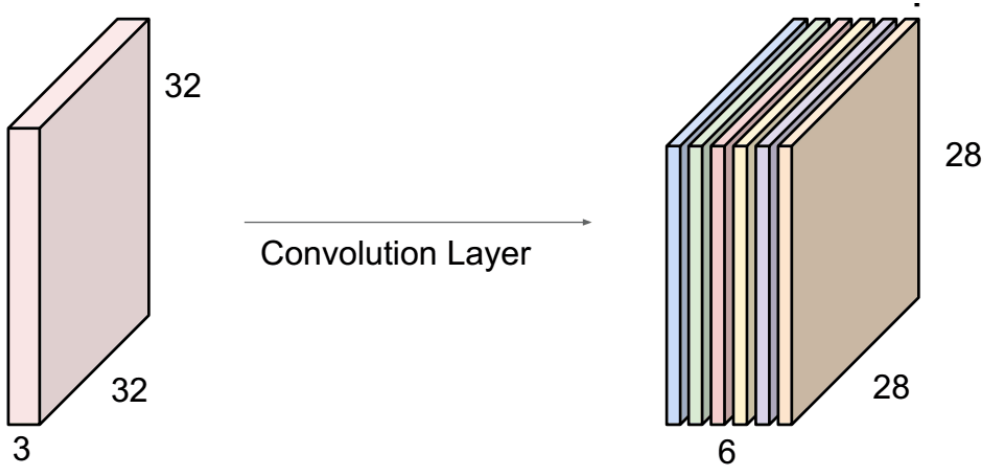
关于卷积

- 多通道卷积
  - 彩色图像通常有RGB三个通道
    - 即输入大小为(height, weight, 3)
  - **每个卷积核**分别对3个通道进行卷积运算，再将结果相加，得到一个单通道特征图
    - 卷积核尺寸(k_h, k_w, 3)
    - **不同通道上的卷积核的参数不同**
  - 总共有几个卷积核，最后输出特征图就有几个通道



关于卷积

- 增加卷积核数量
  - 输入：$W(idth)_{in} * H(ight)_{in} * C(hannel)_{in}$
  - Filter个数：k
  - 卷积核维度：$w * h$
  - Stride，padding：s，p
  - 输出：$W_{out} * H_{out} * C_{out}$
  - $$\begin{cases} W_{out} = \frac{W_{in}+2p-w}{s} + 1 \\ H_{out} = \frac{H_{in}+2p-h}{s} + 1 \\ C_{out} = k \end{cases}$$
  - 参数量：$(w * h * C_{in} + 1) * k$
  - 如，采用6个5*5*3卷积核，就得到6个特征图，则输出有6个通道



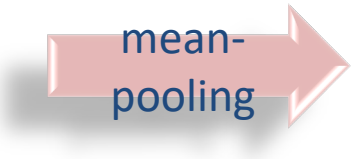
池化

- pooling层(池化层)的输入一般来源于上一个卷积层
- 池化的作用
  - 保留主要特征，减少参数和计算量（降采样）
  - 保持某种不变性，包括translation(平移)，rotation(旋转)，scale(尺度)
- 常用的池化方法
  - 最大池化——更多的保留纹理信息
  - 平均池化——更多的保留图像的背景信息

池化

- 平均池化示例
  - pooling窗的大小是2x2

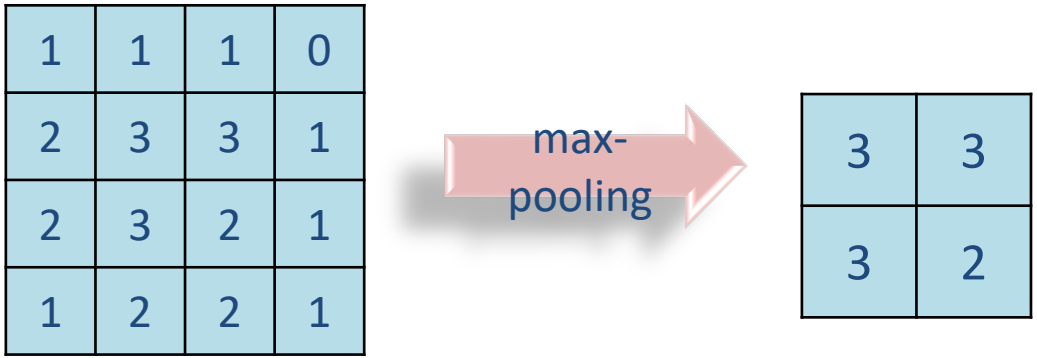
1	1	1	0
2	3	3	1
2	3	2	1
1	2	2	1



7/4	5/4
2	6/4

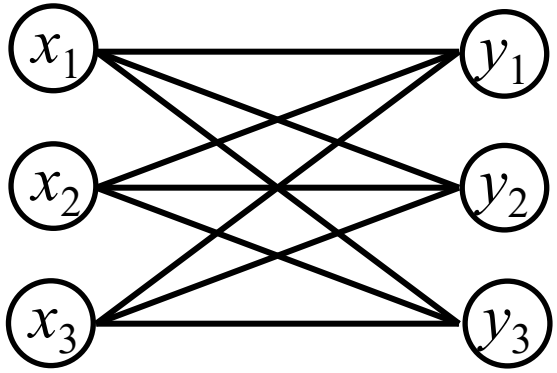
池化

- 最大池化示例
  - pooling窗的大小是2x2



全连接层

- Each node of the fully connected layer is connected to all the nodes of the last layer, which is to combine the features extracted from the front layers



Fully connected layer.

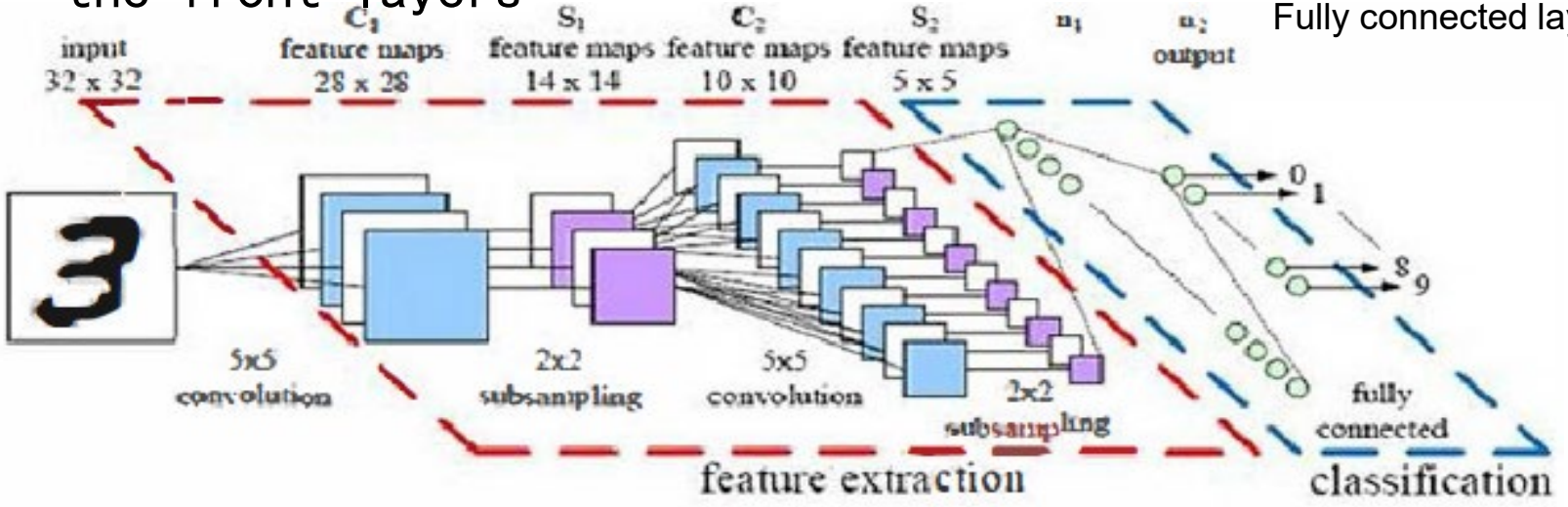


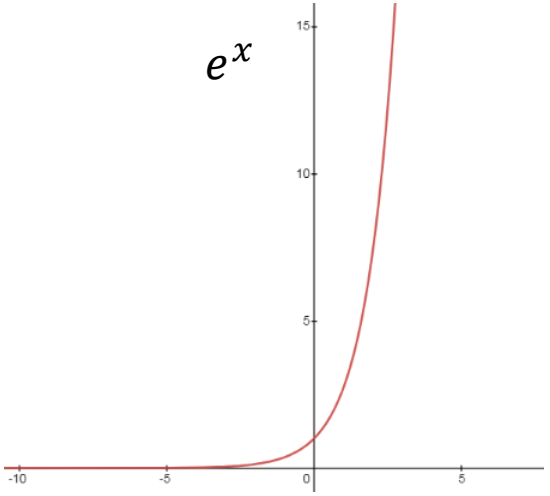
Fig2. Complete CNN structure.

*Softmax*层

- 分类器最后的输出单元需要*Softmax*函数进行处理
- *Softmax*函数

$$f(z_j) = \frac{e^{z_j}}{\sum_{i=1}^{n} e^{z_i}}$$

其中，z_i 是分类器前级输出单元的输出。i 表示类别索引，总的类别个数为 n



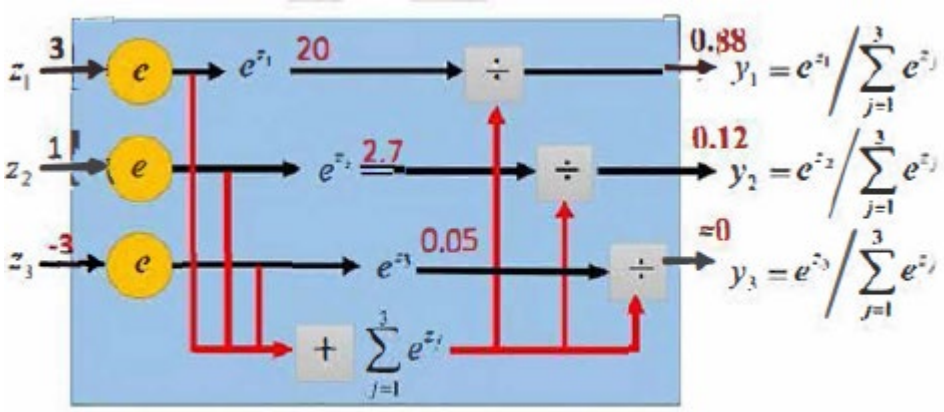


Fig3. Softmax layer.

输出概率:

- $1 > y_i > 0$
- $\sum y_i = 1$

上采样

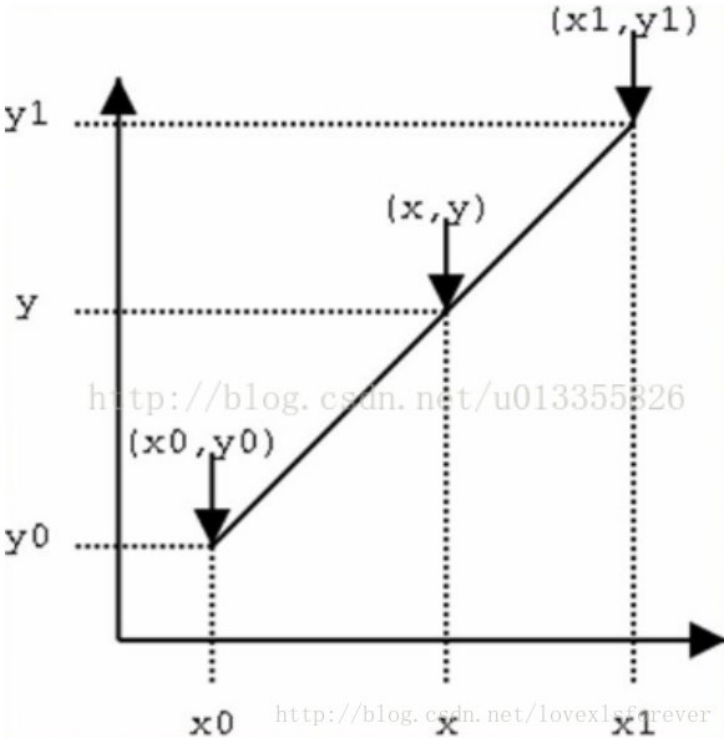
- 上采样(Upsample)
  - 输入图像通过卷积神经网络(CNN)提取特征后，输出的尺寸往往会变小
  - 需要将图像恢复到原来的尺寸以便进行进一步的计算
    - 例：图像的语义分割
  - 采用扩大图像尺寸，实现图像由小分辨率到大分辨率的映射的操作，叫做上采样
  - 上采样有3种常见的方法
    - 双线性插值(BiLinear)
    - 反卷积(Transposed Convolution)
    - 反池化(Unpooling)

双线性插值(BiLinear)

- 单线性插值法

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

$$y = \frac{x_1 - x}{x_1 - x_0} y_0 + \frac{x - x_0}{x_1 - x_0} y_1$$



双线性插值(BiLinear)

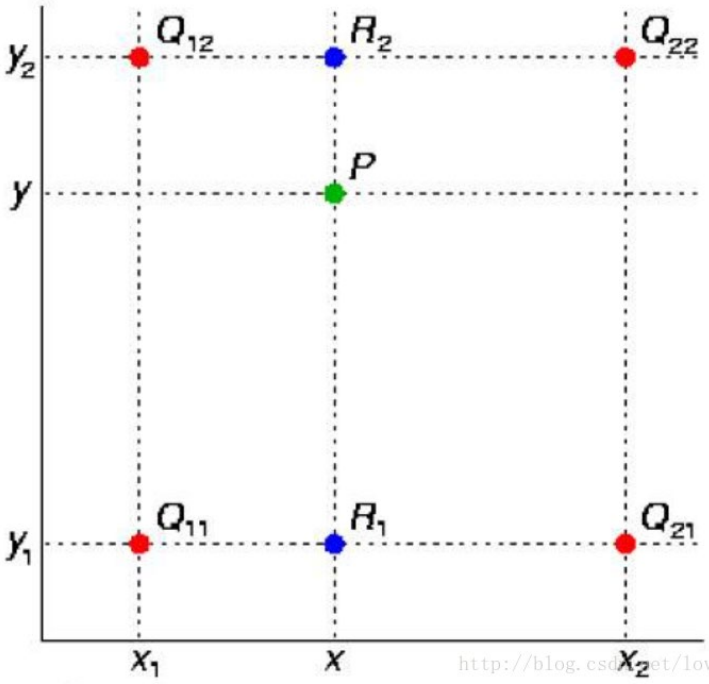
$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \quad \text{where} \quad R_1 = (x, y_1)$$

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \quad \text{where} \quad R_2 = (x, y_2)$$

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2).$$

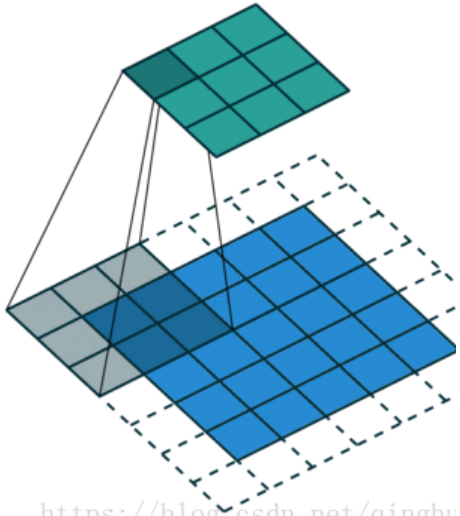
$$f(x,y) \approx \frac{f(Q_{11})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y_2 - y) + \frac{f(Q_{21})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y_2 - y)$$

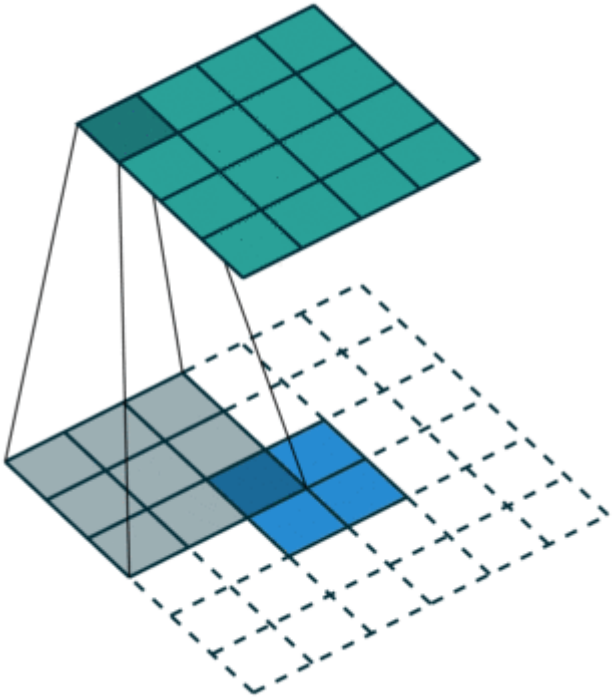
$$+ \frac{f(Q_{12})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y - y_1) + \frac{f(Q_{22})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y - y_1).$$



反卷积

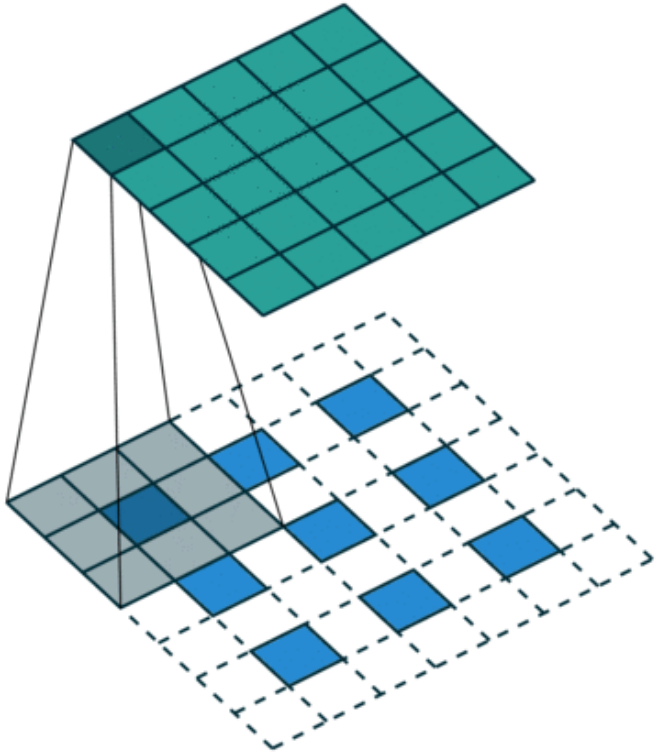
- 卷积过程回顾
- 反卷积
  - 特殊的正向卷积
  - 也叫转置卷积
    - 并不是正向卷积的逆过程
  - 通过补0来扩大输入图像的尺寸
  - 再进行卷积





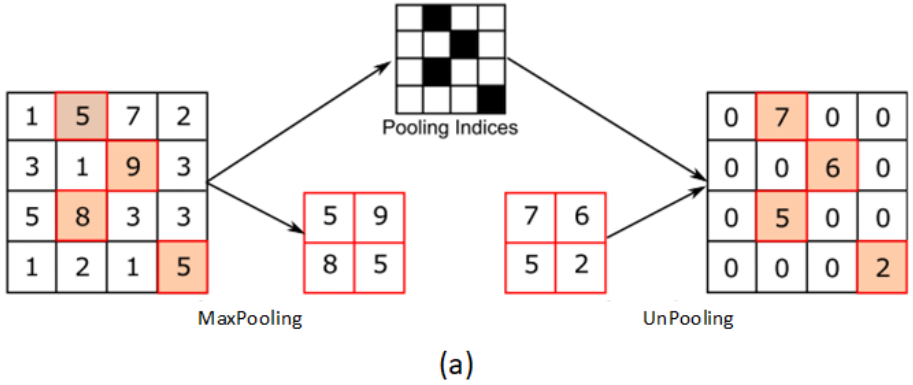
反卷积

- 反卷积的变体



反池化

- 池化的逆操作

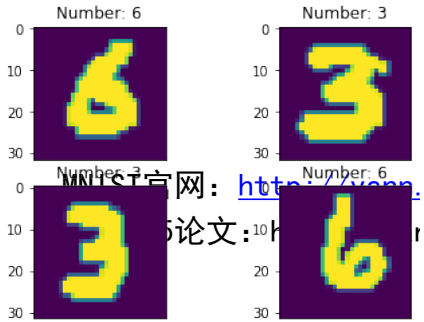

Pooling Indices
MaxPooling
UnPooling
(a)
MaxPooling
UnSampling
(b)

CNN中的训练过程

1. 初始化神经网络
2. 取出一个数据(x,y)（y为标签）
3. 逐层计算l层的激活前输出z^l和激活后输出a^l
4. 计算损失函数值
5. 计算输出层误差δ^L
6. 计算隐层误差
   I. 全连接层：$\delta^l = \left(W^{l+1}\right)^T \delta^{l+1} \odot \sigma'(z^l)$ （$\odot$ 为点积运算，σ'为激活函数的导数）
   II. 卷积层：$\delta^l = \delta^{l+1} * ROT180(W^{l+1}) \odot \sigma'(z^l)$ （*为卷积运算）
   III. 池化层：$\delta^l = upsample(\delta^{l+1}) \odot \sigma'(z^l)$
7. 利用δ^l求损失函数对l层参数的导数
8. 更新权值

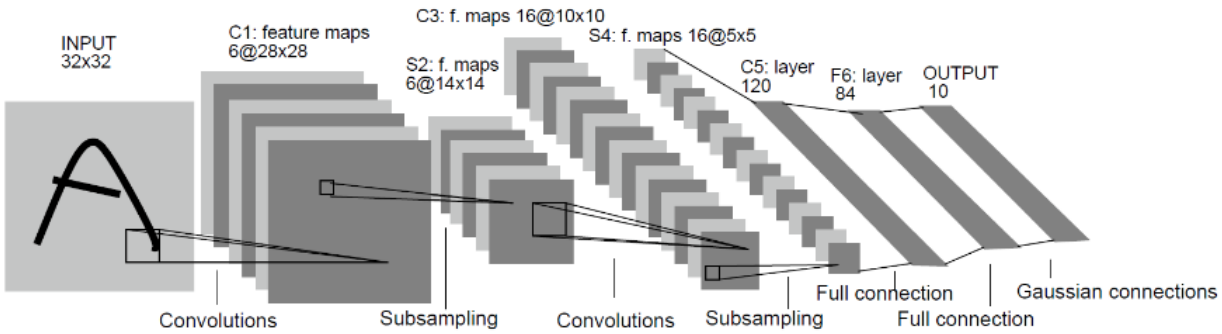
基于MindSport的手写数字识别开发实践

- MNIST是一个手写数字数据集，训练集包含60000张手写数字，测试集包含10000张手写数字，共10类。
- LeNet是最早出现的卷积神经网络之一。
- 在LeNet的基础上，1998年Yann LeCun等构建了卷积神经网络LeNet-5并在手写数字的识别问题中取得成功。LeNet-5及其后产生的变体定义了现代卷积神经网络的基本结构，可谓入门级神经网络模型。



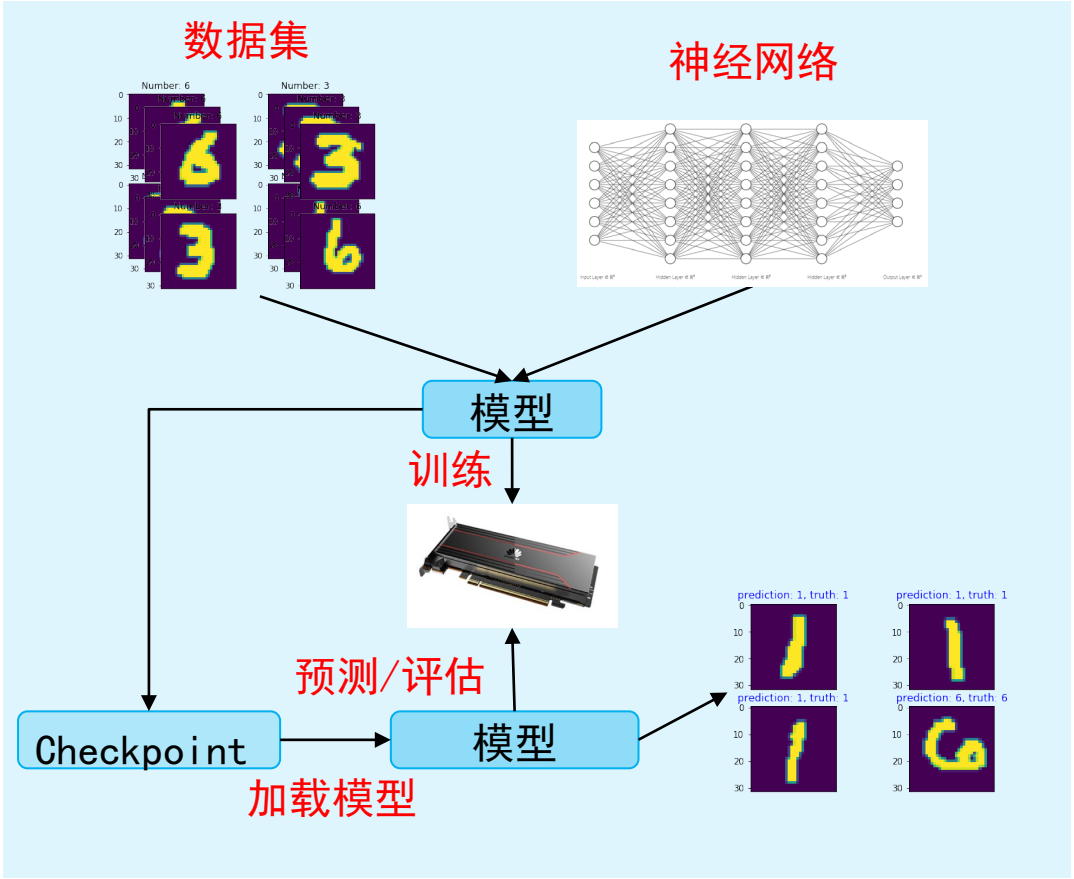
MNIST官网：http://yann.

论文：h



开发流程

- 数据处理
  - 数据加载
  - 数据增强
- 模型定义
  - 定义网络
  - 损失函数
  - 优化器
- 模型训练
  - Loss监控
  - 验证
  - 保存Checkpoint
- 模型推理
  - 推理
  - 部署



代码示例（1）

```
import os
import mindspore as ms
import mindspore.context as context
import mindspore.dataset.transforms.c_transforms as C
import mindspore.dataset.vision.c_transforms as CV

from mindspore import nn
from mindspore.train import Model
from mindspore.train.callback import LossMonitor

context.set_context(mode=context.GRAPH_MODE, device_target='Ascend')

def create_dataset(data_dir, training=True, batch_size=32, resize=(32, 32),
rescale=1/(255*0.3081), shift=-0.1307/0.3081, buffer_size=64):
    data_train = os.path.join(data_dir, 'train')
    data_test = os.path.join(data_dir, 'test')
    ds = ms.dataset.MnistDataset(data_train if training else data_test)

    ds = ds.map(input_columns=["image"], operations=[CV.Resize(resize), CV.Rescale(rescale,
shift), CV.HWC2CHW()])
    ds = ds.map(input_columns=["label"], operations=C.TypeCast(ms.int32))
    ds = ds.shuffle(buffer_size=buffer_size).batch(batch_size, drop_remainder=True)

    return ds
```

① 导入标准库、第三方库和MindSpore模块。

② 设置上下文：GRAPH模式、Ascend设备。

③ 数据处理：数据集加载、缩放、归一化、格式转换、洗牌、批标准化。

代码示例（2）

```
class LeNet5(nn.Cell):
    def __init__(self):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5, stride=1, pad_mode='valid')
        self.conv2 = nn.Conv2d(6, 16, 5, stride=1, pad_mode='valid')
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Dense(400, 120)
        self.fc2 = nn.Dense(120, 84)
        self.fc3 = nn.Dense(84, 10)

    def construct(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = self.relu(self.conv2(x))
        x = self.pool(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)

        return x
```

④模型定义：算子初始化（参数设置），网络构建。

代码示例（3）

```
def train(data_dir, lr=0.01, momentum=0.9, num_epochs=3):
    ds_train = create_dataset(data_dir)
    ds_eval = create_dataset(data_dir, training=False)

    net = LeNet5()
    loss = nn.loss.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
    opt = nn.Momentum(net.trainable_params(), lr, momentum)
    loss_cb = LossMonitor(per_print_times=ds_train.get_dataset_size())

    model = Model(net, loss, opt, metrics={'acc', 'loss'})
    model.train(num_epochs, ds_train, callbacks=[loss_cb], dataset_sink_mode=False)
    metrics = model.eval(ds_eval, dataset_sink_mode=False)
    print('Metrics:', metrics)
```

⑤ 模型训练和验证：实例化net，loss，optimizer，loss_monitor，调用Model接口进行训练和验证。

数据处理Pipeline

- 数据是深度学习的基础，高质量的数据输入会在整个深度神经网络中起到积极作用。
- 在训练开始之前，由于数据量有限，或者为了得到更好的结果，通常需要进行数据处理与数据增强，以获得能使网络受益的数据输入。

- 数据经过处理和增强，像流经管道的水一样源源不断的流向训练系统。

加载数据集

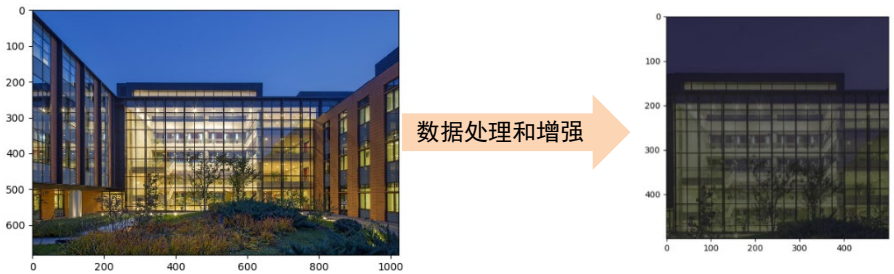
Dataset structure	使用简介
ImageFolderDatasetV2	ds = ds.ImageFolderDatasetV2(DATA_DIR, class_indexing={"cat":0,"dog":1})
MnistDataset	ds = ms.dataset.MnistDataset(DATA_DIR)
Cifar10Dataset	ds = ms.dataset.Cifar10Dataset(DATA_DIR)
CocoDataset	ds = ds.CocoDataset(DATA_DIR, annotation_file=annotation_file, task='Detection')
MindRecord	ds = ds.MindDataset(FILE_NAME)
GeneratorDataset	ds = ds.GeneratorDataset(GENERATOR, [“image", “label"])

TextFileDataset	dataset_files = ["/path/to/1", "/path/to/2"] # contains 1 or multiple text files
	ds = ds.TextFileDataset(dataset_files=dataset_files)

GraphData	ds = ds.GraphData(DATA_DIR, 2) nodes = ds.get_all_edges(0)

数据处理

- 数据处理
  - 以图片为例：将原始图片进行裁剪、大小缩放、归一化、格式转换等，以便深度学习模型能够使用。
- 数据增强
  - 一种创造 “新”数据的方法，从有限数据中生成“更多数据”，防止过拟合现象



数据增强：有限数据和复杂模型→过拟合

格式转换：MindSpore支持通道在前


```
operations = [
    CV.Resize(resize),
    CV.Rescale(rescale, shift),
    CV.HWC2CHW()
]
ds = ds.map(input_columns=["image"],
operations=operations)
ds = ds.shuffle(buffer_size=buffer_size).batch(batch_size,
drop_remainder=True)
```

常用操作

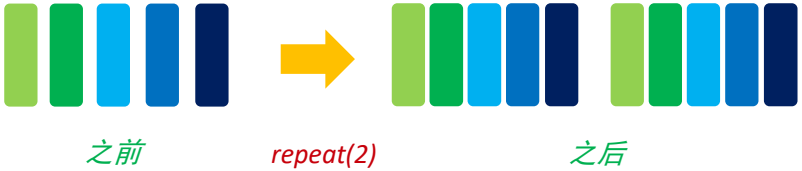
（1）打乱（shuffle）

打乱（shuffle）操作用来打乱数据集中的数据排序。

越大的buffer_size意味着越高的混洗度，但也意味着会花费更多的时间和计算资源。

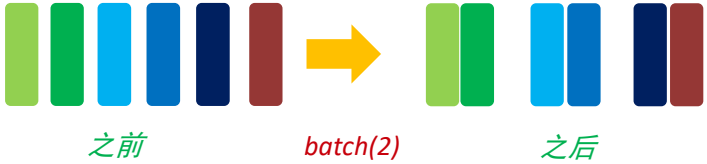
（3）重复（repeat）

可以通过重复（repeat）的方式增加训练数据量，通常放在分批（batch）操作之后。



（2）分批（batch）

在训练时，数据可能需要分批（batch）处理。

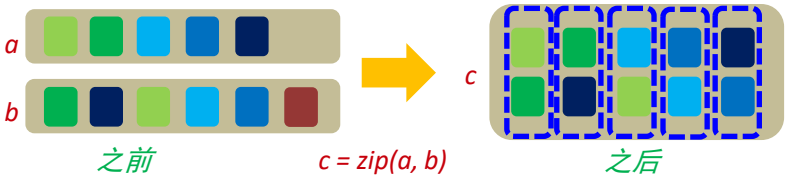


（4）合并（zip）

将多个数据集合并成一个。

若两个数据集的列的名字相同，则不能合并。

若两个数据集的行数不同，合并后的行数以较小的为准。



代码实例

```
def create_dataset(data_dir, training=True, batch_size=32, resize=(32, 32),
                   rescale=1/(255*0.3081), shift=-0.1307/0.3081, buffer_size=64):
    # 训练集和测试集路径
    data_train = os.path.join(data_dir, 'train')
    data_test = os.path.join(data_dir, 'test')
    # 加载MNIST数据集
    ds = ms.dataset.MnistDataset(data_train if training else data_test)

    # 通过map方法对每张图片应用数据处理操作：
    # Resize：缩放图片大小
    # Rescale：将每个像素的灰度值由[0, 255]标准化到[-1, 1]
    # HWC2CHW：将张量格式从NHWC转换成NCHW
    ds = ds.map(input_columns=[ "image" ], operations=[CV.Resize(resize), CV.Rescale(rescale, shift), CV.HWC2CHW()])
    # 将每个标签的数据类型转换为int32
    ds = ds.map(input_columns=[ "label" ], operations=C.TypeCast(ms.int32))
    # 当在Ascend环境上使用数据下沉模型时，设置`dataset_sink_mode=True`
    ds = ds.shuffle(buffer_size=buffer_size).batch(batch_size, drop_remainder=True)

    return ds
```

模型定义——卷积（Convolution）

```
conv1 = nn.Conv2d(in_channels, out_channels,  ①
    ② kernel_size=3, stride=1, has_bias=False, weight_init='normal', bias_init='zeros',
    ③ pad_mode='same', padding=0)
```

① Input and Output Channels　② Kernel Size and Stride　③ Padding

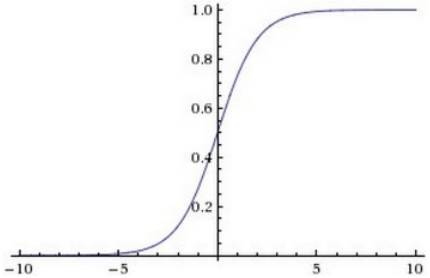
```
y = conv1(x)
```

接受一个4维的张量作为输入，返回一个4维的张量作为输出，4维即batch_size x channels x height x width

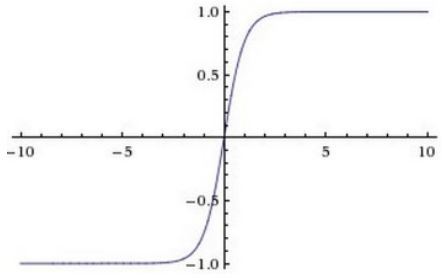
设定的参数决定了输出的特征图的大小：h_out = (h_in - kernel_size + 2*padding)/strides + 1

模型定义——激活函数

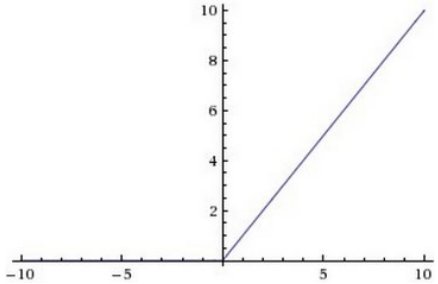
```
relu = nn.Sigmoid()
relu = nn.Tanh()
relu = nn.ReLU()
```



$$Sigmoid(x) = \frac{1}{1 + e^{-x}}$$



$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

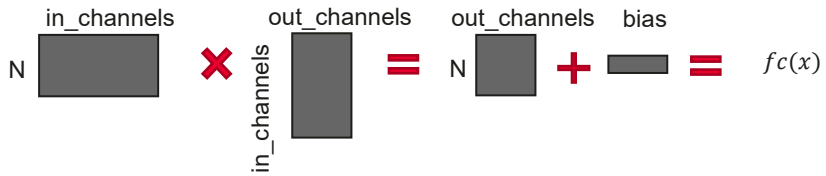


$$ReLU(x) = \max(0, x)$$

- 在卷积层之后引入激活函数进行非线性变换，可以提高模型的拟合能力。

模型定义——全连接（Dense）

```
fc1 = nn.Dense(in_channels, out_channels, weight_init='normal', bias_init='zeros', has_bias=True, activation=None)
```



y = fc1(x)：

- 输入：Tensor shape(batch_size, in_channels).
- 输出：Tensor shape(batch_size, out_channels).

接受一个2维的张量作为输入，返回一个2维的张量作为输出，2维即batch_size x channels

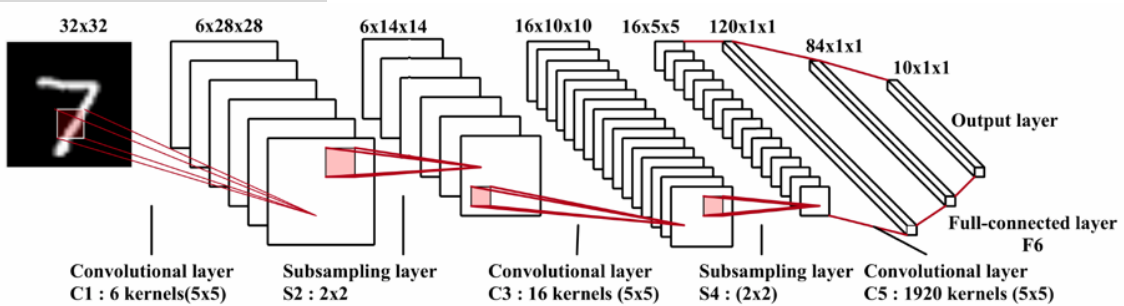
模型定义——网络定义

```
class LeNet5(nn.Cell):                                  ①
    def __init__(self):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5, stride=1, pad_mode='valid')
        self.conv2 = nn.Conv2d(6, 16, 5, stride=1, pad_mode='valid')
        self.relu = nn.ReLU()                           ②
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Dense(400, 120)
        self.fc2 = nn.Dense(120, 84)
        self.fc3 = nn.Dense(84, 10)

    def construct(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = self.relu(self.conv2(x))                    ③
        x = self.pool(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)

        return x
```

① 网络定义必须继承基类nn.Cell；

② __init__()中的语句由Python解析执行；

③ construct()中的语句由MindSpore接管，有语法限制；



(a) LeNet-5 network

模型定义——损失函数（Loss）

```
loss = nn.loss.SoftmaxCrossEntropyWithLogits(is_grad=False, sparse=True, reduction='mean')
```

$$Z_i = W \cdot X_i + b$$

$$P_i = \frac{e^{Z_i}}{\sum_j e^{Z_i}}$$

$$l(Z_i, Y_i) = -\log\left(P_{iY_i}\right)$$

- 对于每个样本N_i，X_i是3D Tensor（CHW），Z_i是3D Tensor，Y_i是真实类别（10类中的一个），P_i是3D Tensor（含10个元素，分别表示属于相应的概率，值域为[0, 1]），$l(Z_i, Y_i)$是标量。
- 损失函数用于描述模型预测值与真实值的误差，用于指导模型的更新方向。MindSpore支持的损失函数有：L1Loss、MSELoss、SoftmaxCrossEntropyWithLogits、CosineEmbeddingLoss等。
- 参数解释：
  - is_grad，是否仅计算梯度，默认为True；
  - sparse，默认为False，为True时对Label数据做one_hot处理；
  - reduction，支持mean、sum。

模型定义——优化器（Optimizer）

```
opt = nn.Momentum(net.trainable_params(), lr, momentum)
```

- 有了优化目标（Loss）之后，还需要定义优化的策略，即优化器。
- MindSpore支持的优化器有：SGD, Momentum, RMSProp, Adam, AdamWeightDecay, LARS, FTRL, ProximalAdagrad等。

SGD with Momentum:

$$v_{t+1} \leftarrow \rho v_t + \nabla_\theta \mathcal{L}(\theta)$$

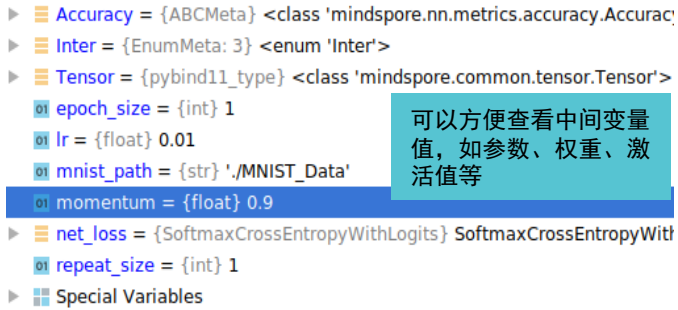
$$\theta_j \leftarrow \theta_j - \epsilon v_{t+1}$$

代码调试

- MindSpore支持Graph和PyNative 2种运行模式，PYNATIVE模式易调试，支持Python原生调试方式。

```
net_loss = SoftmaxCrossEntropyWithLogits(is_grad=False, sparse=True, reduction='
repeat_size = epoch_size  repeat_size: 1
# create the network
network = LeNet5()
# define the optimizer
net_opt = nn.Momentum(network.trainable_params(), lr, momentum)
config_ck = CheckpointConfig(save_checkpoint_steps=1875, keep_checkpoint_max=10)
# save the network model and parameters for subsequence fine-tuning
ckpoint_cb = ModelCheckpoint(prefix="checkpoint_lenet", config=config_ck)
# group layers into an object with training and evaluation features
model = Model(network, net_loss, net_opt, metrics={"Accuracy": Accuracy()})
```

支持Python原生的调试方式，如设置断点



可以方便查看中间变量值，如参数、权重、激活值等

```
def construct(self, x):
    x = self.relu(self.conv1(x))
    x = self.pool(x)
    print(x.shape)
    x = self.relu(self.conv2(x))
    x = self.pool(x)
    x = self.flatten(x)
    x = self.fc1(x)
    x = self.fc2(x)
    x = self.fc3(x)

    return x
```

打印某层算子输出的shape，方便设置下一层算子的参数

GRAPH模式性能高，可调用Print算子打印Tensor或字符串

```
self.print = P.Print()
self.print(x)
Tensor shape:[[const vector][2, 1]]Int32
val:[[1] [1]]
```

模型调优

- mindspore.train.callback可用于训练/验证过程中执行特定的任务。常用的Callback如下：
  - LossMonitor：监控loss值，当loss值为Nan或Inf时停止训练
  - SummaryStep：把训练过程中的信息存储到文件中，用于后续查看或可视化展示
  - ModelCheckpoint：保存模型文件和参数，用于再训练或推理

```
loss_cb = LossMonitor(per_print_times=ds_train.get_dataset_size())
# 打印
epoch: 2 step 1875, loss is 0.4737345278263092
```

- mindspore.nn提供了多种Metric评估指标，如accuracy、loss、precision、recall、F1。
  - 定义一个metrics字典/元组，里面包含多种指标，传递给Model
  - 然后调用model.eval接口来计算这些指标
  - model.eval会返回一个字典，包含各个指标及其对应的值

```
metrics={'acc', 'loss'}
# 输出
{'loss': 0.10531254443608654, 'acc': 0.9701522435897436}
```

模型训练和验证

- mindspore.train.Model提供了高阶模型训练和验证接口。
  - 传入net、loss和opt，并完成Model的初始化
  - 然后调用train接口，指定迭代次数（num_epochs）、数据集（dataset）、回调（callback）
- 训练包含两层循环：外层为数据集的多代（epoch）循环，内层为epoch内多步（batch/step）的迭代

```
def train(data_dir, lr=0.01, momentum=0.9, num_epochs=3):
    ds_train = create_dataset(data_dir)
    ds_eval = create_dataset(data_dir, training=False)

    net = LeNet5()
    loss = nn.loss.SoftmaxCrossEntropyWithLogits(is_grad=False, sparse=True, reduction='mean')
    opt = nn.Momentum(net.trainable_params(), lr, momentum)
    loss_cb = LossMonitor(per_print_times=ds_train.get_dataset_size())

    model = Model(net, loss, opt, metrics={'acc', 'loss'})
    # dataset_sink_mode can be True when using Ascend
    model.train(num_epochs, ds_train, callbacks=[loss_cb], dataset_sink_mode=False)
    metrics = model.eval(ds_eval, dataset_sink_mode=False)
    print('Metrics:', metrics)
```

- LeNet5完整代码示例：https://gitee.com/mindspore/course/tree/master/checkpoint

模型保存和加载

- ModelCheckpoint会生成模型（.meta）和Chekpoint（.ckpt）文件，如每个epoch结束时，都保存一次checkpoint。

```
ckpt_cfg = CheckpointConfig(save_checkpoint_steps=steps_per_epoch, keep_checkpoint_max=5)
ckpt_cb = ModelCheckpoint(prefix=ckpt_name, directory='ckpt', config=ckpt_cfg)
model.train(num_epochs, ds_train, callbacks=[ckpt_cb, loss_cb], dataset_sink_mode=dataset_sink)
# 输出：
lenet-1_1875.ckpt
lenet-2_1875.ckpt
lenet-graph.meta
```

- 如果训练中断后想继续做训练，或者加载模型做微调（Fine-tuning），先使用mindspore.train.serialization提供的load_checkpoint、load_param_into_net功能，再行训练。

```
CKPT_1 = 'ckpt/lenet-2_1875.ckpt‘
# 加载模型，返回参数字典
param_dict = load_checkpoint(CKPT_1)
# 分别将参数加载到网络和优化器上
load_param_into_net(net, param_dict)
load_param_into_net(opt, param_dict)
```

- Checkpoint完整代码示例：https://gitee.com/mindspore/course/tree/master/checkpoint

模型推理

- 使用MindSpore做推理
  - 加载Checkpoint到网络中
  - 调用model.predict()接口进行推理

```
CKPT_2 = 'ckpt/lenet_1-2_1875.ckpt'
def infer(data_dir):
    ds = create_dataset(data_dir, training=False).create_dict_iterator()
    data = ds.get_next()
    images = data['image']
    labels = data['label']
    net = LeNet5()
    load_checkpoint(CKPT_2, net=net)
    model = Model(net)
    output = model.predict(Tensor(data['image']))
    preds = np.argmax(output.asnumpy(), axis=1)
```

- 使用其他平台做推理
  - 加载Checkpoint到网络中
  - 调用mindspore.train.serialization提供的export()接口，导出ONNX等格式的模型文件
  - 在任何支持ONNX模型文件的平台上进行推理

```
input = np.random.uniform(0.0, 1.0, size = [1, 3, 224, 224]).astype(np.float32)
export(resnet, Tensor(input), file_name = 'resnet50-2_32.pb', file_format = 'ONNX')
```

部署推理服务

- MindSpore Serving是一个轻量级、高性能的服务模块，旨在帮助MindSpore开发者在生产环境中高效部署在线推理服务。
  - 使用MindSpore完成模型训练
  - 导出MindSpore模型（BINARY格式）
  - 使用MindSpore Serving创建推理服务

```
ms_serving [--help] [--model_path <MODEL_PATH>] [--model_name <MODEL_NAME>]
        [--port <PORT>] [--device_id <DEVICE_ID>]
```

参数名	属性	功能描述	参数类型	默认值	取值范围
--help	可选	显示启动命令的帮助信息	-	-	-
--model_path <MODEL_PATH>	必选	指定待加载模型的存放路径	str	空	-
--model_name <MODEL_NAME>	必选	指定待加载模型的文件名	str	空	-
--port <PORT>	可选	指定Serving对外的端口号	int	5500	1~65535
--device_id <DEVICE_ID>	可选	指定使用的设备号	int	0	0~7

指导文档链接： https://www.mindspore.cn/serving/docs/zh-CN/master/index.html