神经网络与卷积神经网络

Neural Network and Convolutional Neural Network

CONTENT

神经网络

深度神经网络

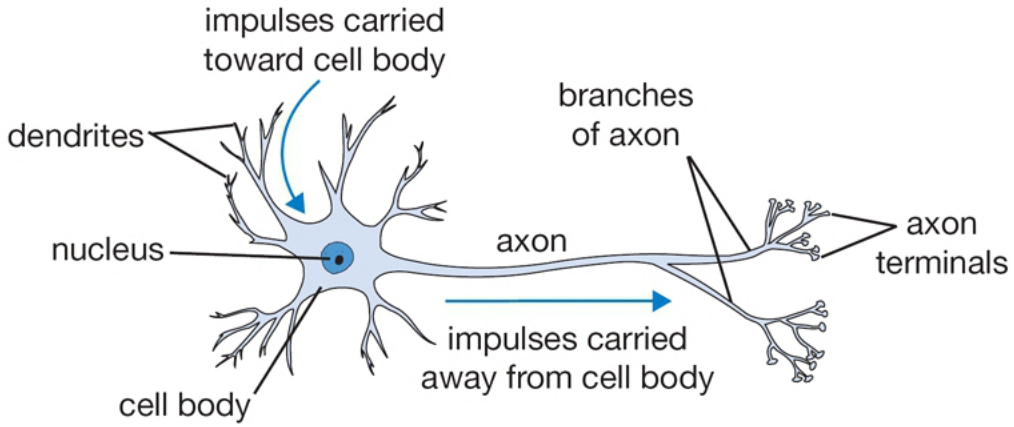
卷积神经网络

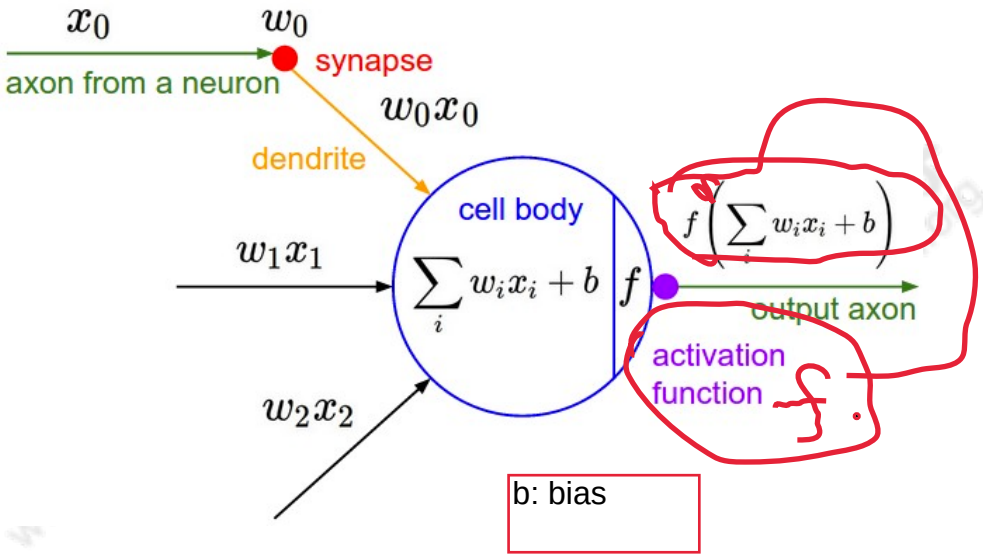
卷积神经网络的应用

神经网络

Neural Network

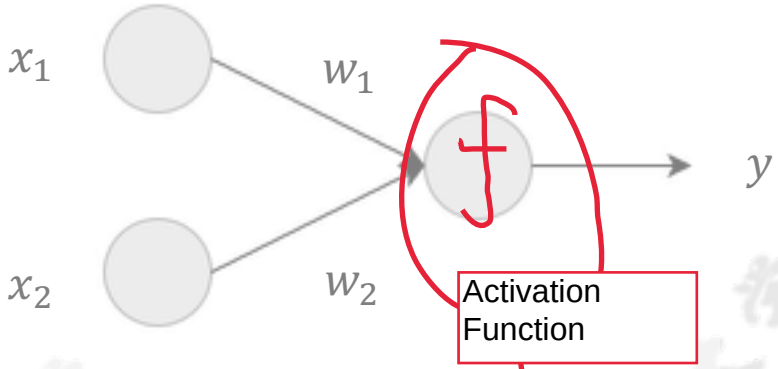
神经网络





Feed Forward Network

前馈神经网络

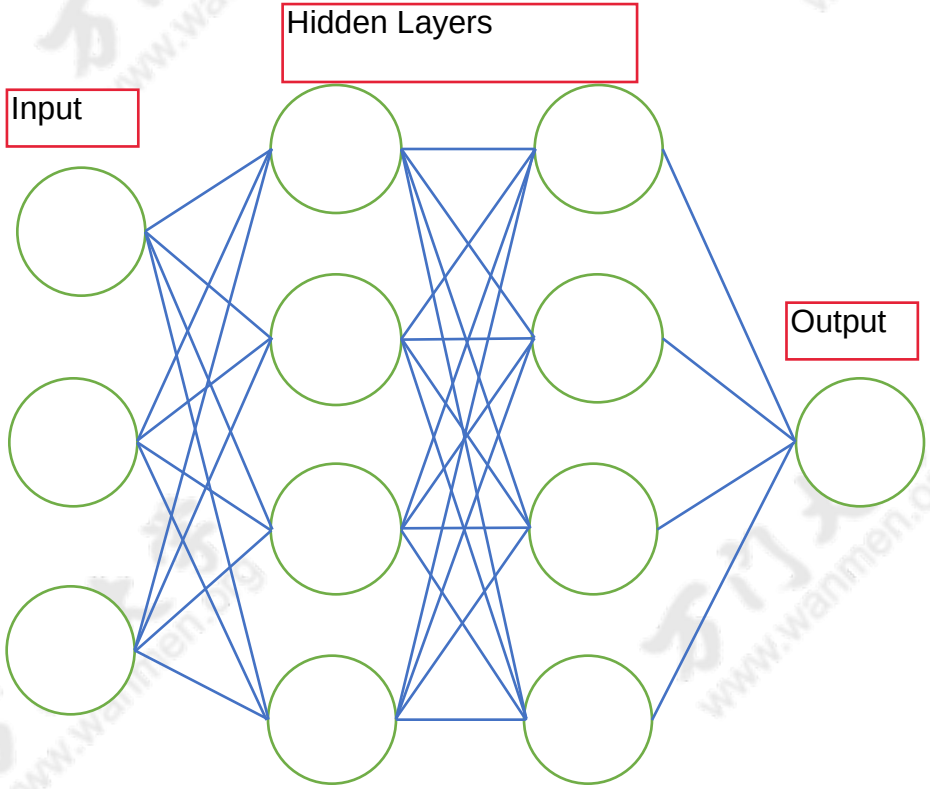


1. $y = w_1x_1 + w_2x_2$
2. $y = \sigma(w_1x_1 + w_2x_2)$

深度神经网络

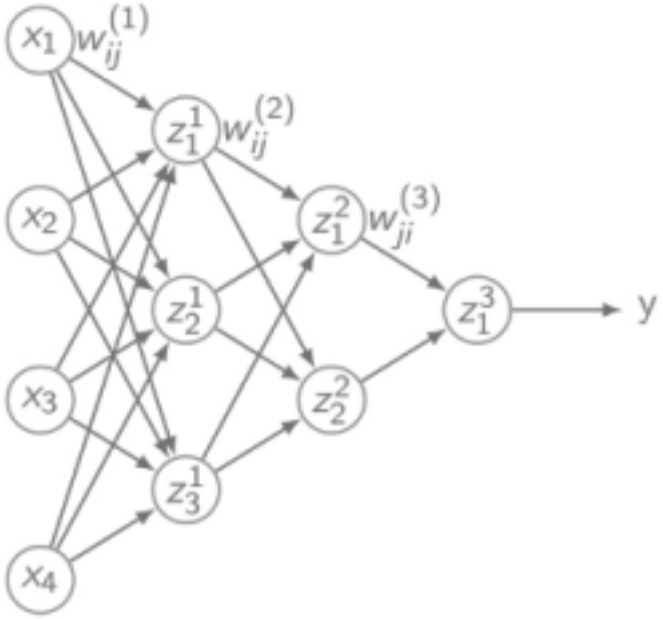
Neural Network

深度神经网络



Back Propagation

反向传播算法**本**质上就是chain rule to calculate y 对于某个参数的partial derivation.



$$z_i^{(1)} = \sum_{j=1}^{4} w_{ji}^{(1)} x_j$$

$$z_i^{(2)} = \sum_{j=1}^{3} w_{ji}^{(2)} \sigma(z_j^{(1)})$$

$$z_i^{(3)} = \sum_{j=1}^{2} w_{ji}^{(3)} \sigma(z_j^{(2)})$$

$$y = \sigma\left(z^{(3)}\right)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

目标函数: $\mathcal{L}(\hat{y}, y)$

- $(\hat{y} - y)^2$
- $y\ log(\hat{y}) + (1 - y) \log(1 - \hat{y})$

梯度下降:

$$w_{i+1} = w_i - \lambda \nabla_w \mathcal{L}(\hat{y}, y)$$

Back Propagation

反向传播算法

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(3)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial w_{ji}^{(3)}} = 2\hat{\delta}\sigma(z_i^{(3)})(1-\sigma(z_i^{(3)}))\sigma(z_j^{(2)})$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{ji}^{(2)}} &= \frac{\partial \mathcal{L}}{\partial \hat{y}} \sum_l \frac{\partial \hat{y}}{\partial z_l^{(3)}} \frac{\partial z_l^{(3)}}{\partial z_i^{(2)}} \frac{\partial z_i^{(2)}}{\partial w_{ji}^{(2)}} \\ &= 2\hat{\delta} \sum_l w_{il}^{(3)} \sigma(z_l^{(3)})(1-\sigma(z_l^{(3)}))\sigma(z_i^{(2)})(1-\sigma(z_i^{(2)}))\sigma(z_j^{(1)})\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{ji}^{(1)}} &= \frac{\partial \mathcal{L}}{\partial \hat{y}} \sum_l \frac{\partial \hat{y}}{\partial z_l^{(3)}} \sum_k \frac{\partial z_l^{(3)}}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial z_i^{(1)}} \frac{z_i^{(1)}}{\partial w_{ji}^{(1)}} \\ &= 2\hat{\delta} \sum_l \sigma(z_l^{(3)})(1-\sigma(z_l^{(3)})) \sum_k w_{kl}^{(3)} w_{ik}^{(2)} \sigma(z_k^{(2)})(1-\sigma(z_k^{(2)})) \\ &\times \sigma(z_i^{(1)})(1-\sigma(z_i^{(1)}) x_j\end{aligned}$$

$$\hat{\delta}_i^{(3)} = 2\hat{\delta}\sigma(z_i^{(3)})(1-\sigma(z_i^{(3)}))$$

$$\hat{\delta}_i^{(2)} = \sum_l w_{il}^{(3)} \hat{\delta}_l^{(3)} \sigma(z_i^{(2)})(1-\sigma(z_i^{(2)}))$$

$$\hat{\delta}_i^{(1)} = \sum_k w_{ik}^{(2)} \hat{\delta}_k^{(2)} \sigma(z_i^{(1)})(1-\sigma(z_i^{(1)}))$$

$$\begin{aligned}\Delta w_{ji}^{(3)} &= \hat{\delta}_i^{(3)} \sigma(z_j^{(2)}) \\ \Delta w_{ji}^{(2)} &= \hat{\delta}_i^{(2)} \sigma(z_j^{(1)}) \\ \Delta w_{ji}^{(1)} &= \hat{\delta}_i^{(1)} x_j\end{aligned}$$

Back Propagation

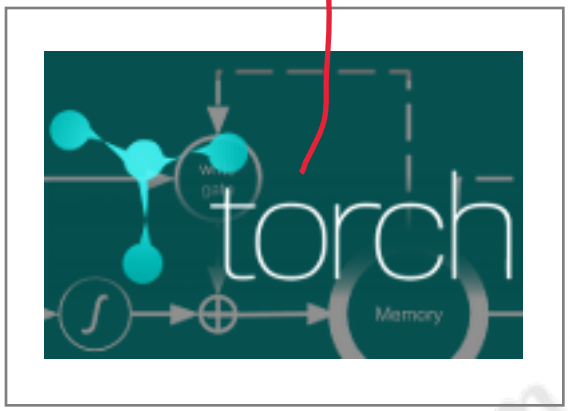
反向传播算法

Tools:
Tensorflow: good for deployment in enterprise level

Keras: good for beginner, R&D researcher

pytorch

keras



Activation

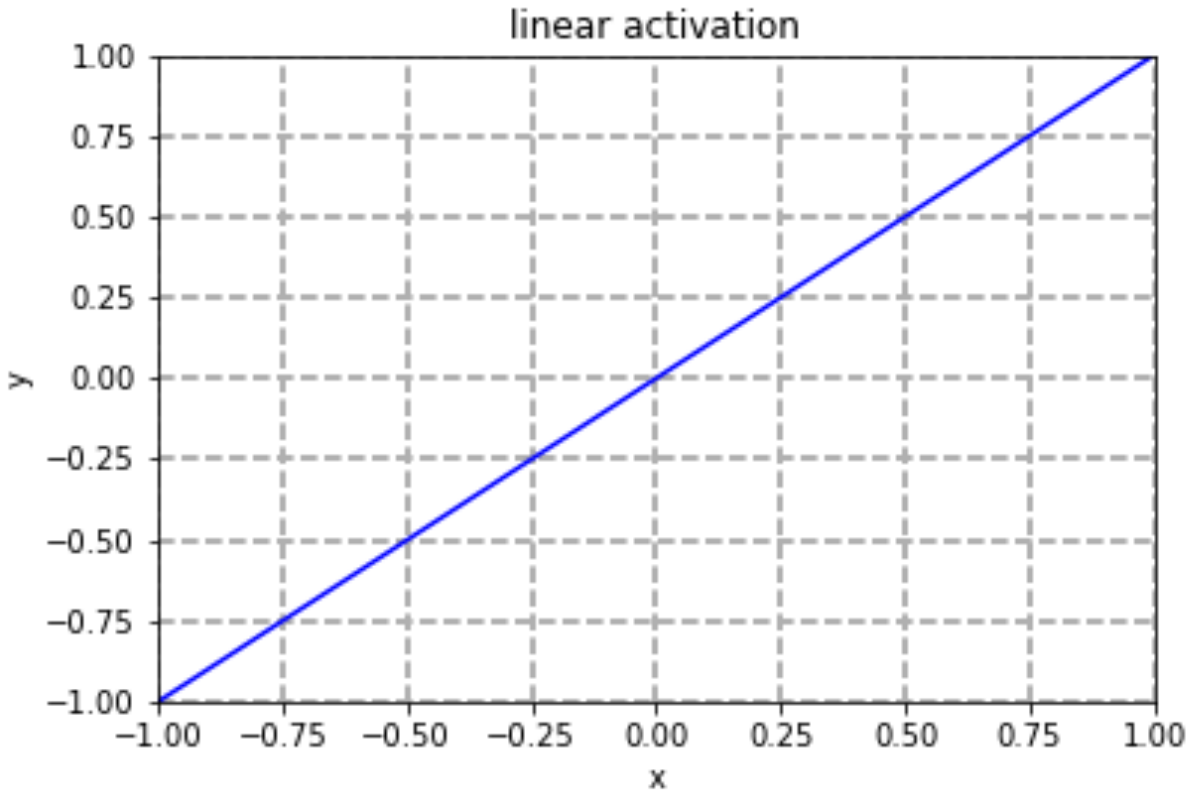
激活函数 Activation Function

- Linear
- Softmax/Sigmoid
- Relu
- LeakyRelu
- PRelu
- Selu
- Tanh

Back Propagation

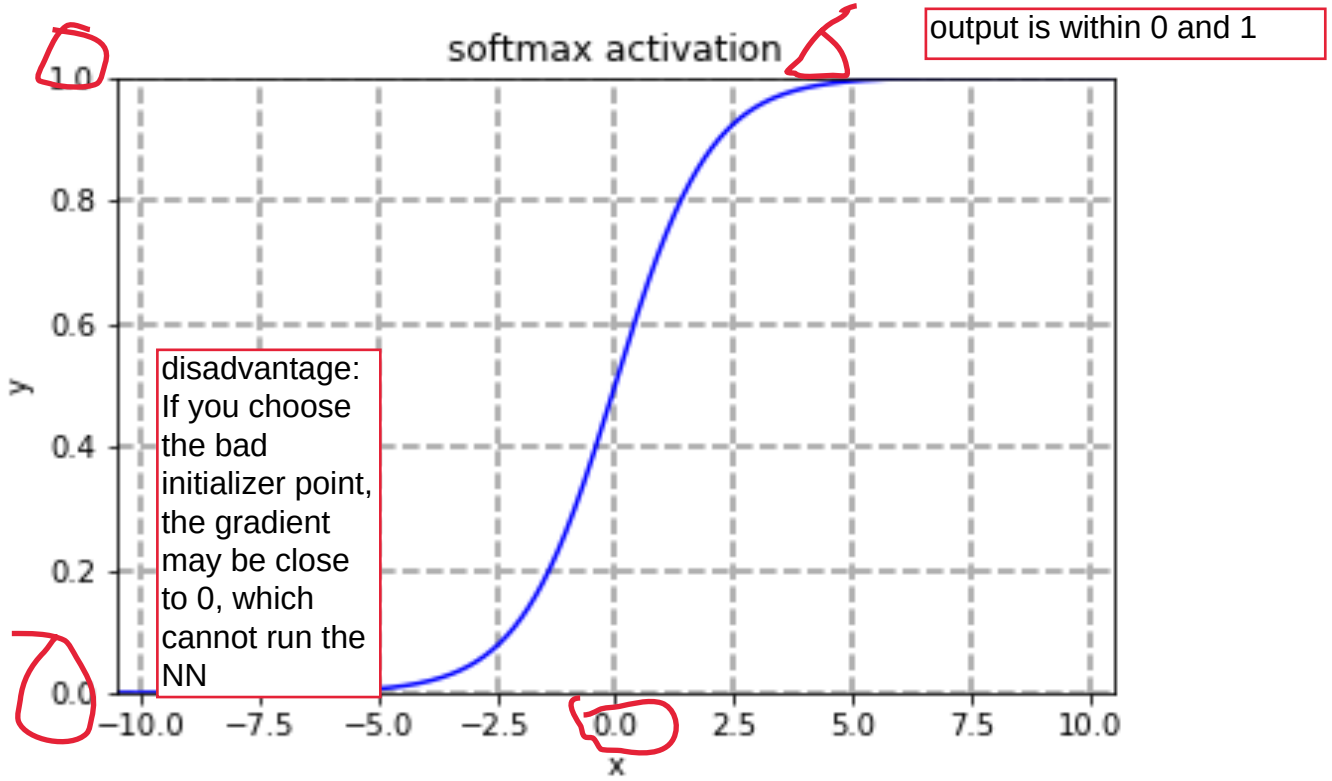
线性激活函数

$$f(x) = x$$



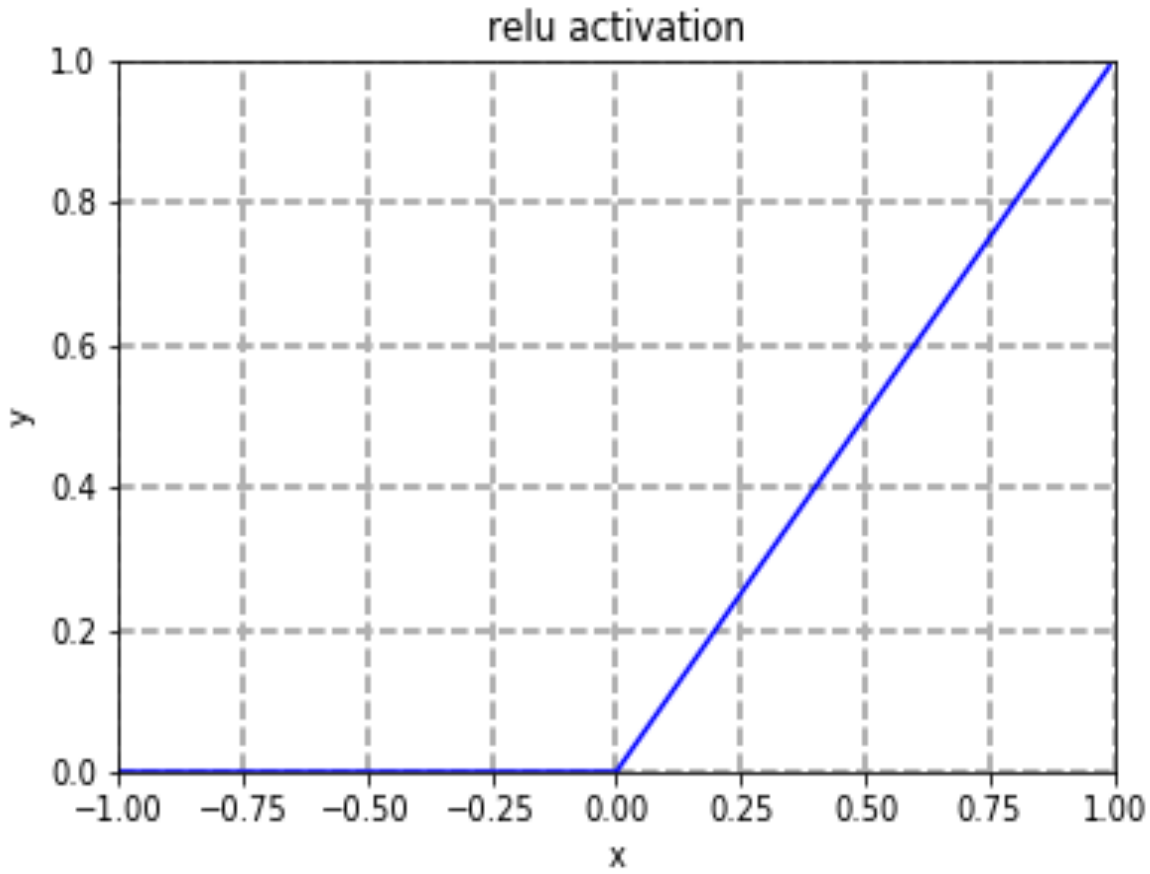
Softmax

$$f(x) = \frac{1}{1 + \exp(-w^T x)}$$



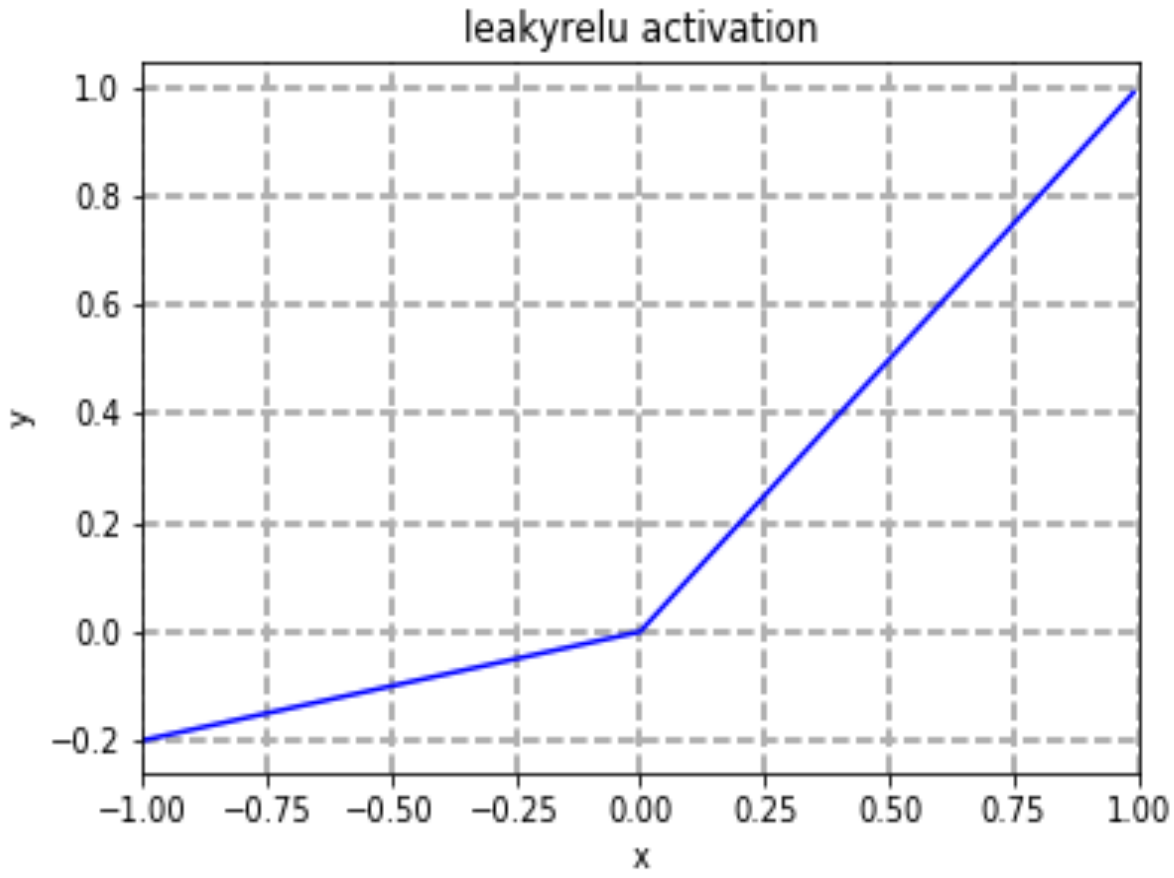
Relu(Rectifier Liner Unit)

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



LeakyRelu

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



Selu

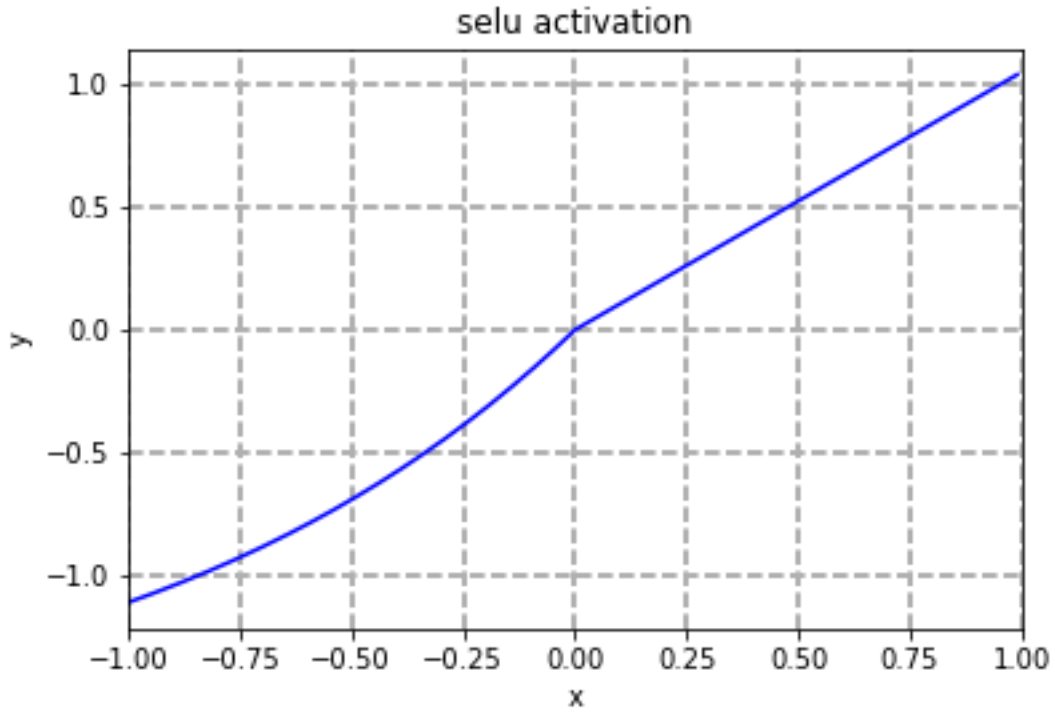
This activation function:
If input is gaussian distribution, output will also be a gaussian distribution.

And gaussian distribution is the common distribution when initializing the parameter.

$$f(x) = \lambda \begin{cases} x & if\ x > 0 \\ \alpha \exp(x) - \alpha & if\ x \leq 0 \end{cases}$$

$$\alpha = 1.6732632$$

$$\lambda = 1.050700987$$



Optimizer

优化算法

- SGD
- Momentum
- Nesterov accelerated gradient
- Adagrad
- Adadelta
- RMSprop
- Adam

优化算法：SGD

Disadvantage: When dimension is higher and higher, there will be more and more saddle point, where model will be stuck at that point forever once encountered.

Saddle Point: e.g. for X axis, it is the local minimum (derivation=0), for Y axis, it is local maximum (derivation=0 as well)

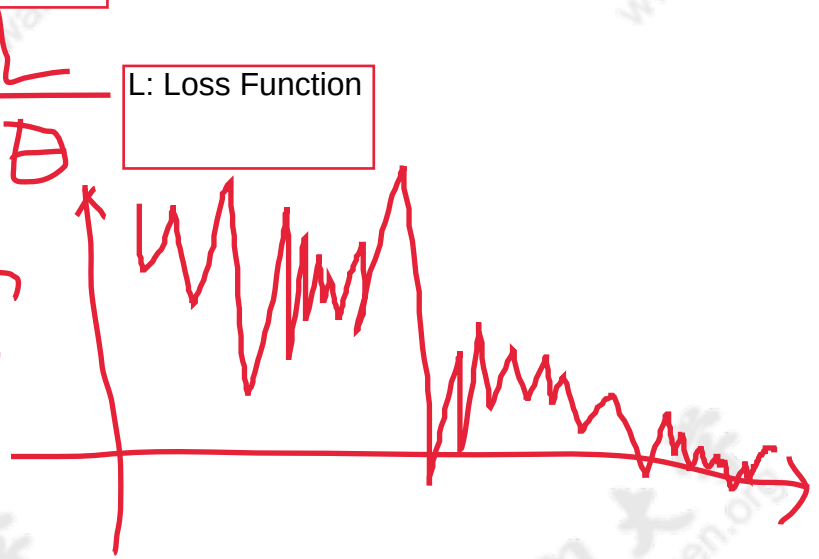
- 随机梯度下降(Stochastic Gradient Descent):

Disadvantage: Stochastic Gradient Descent train sample one by one, since every sample has its noise, the training progress will have high oscillation (震荡)

learning rate

$$\theta_t = \theta_{t-1} - \lambda \nabla_\theta \mathcal{L}\left(\theta, x^{(i)}, y^i\right)$$

L: Loss Function



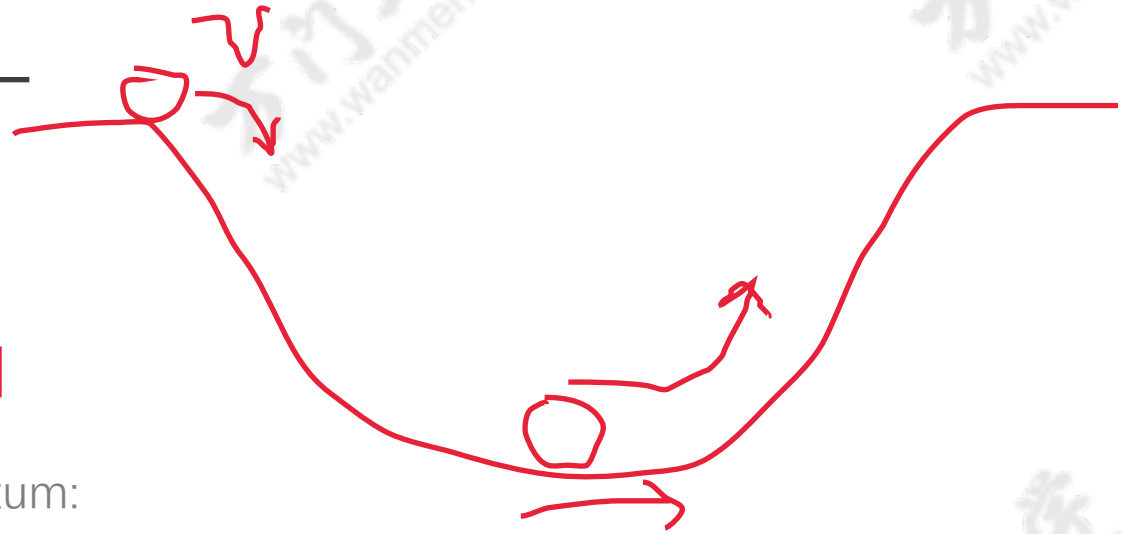
- 小批量梯度下降(Mini-batch Gradient Descent):

Advantage: calculate gradient batch by batch, which will reduce the oscillation.

batch size = n, usually n will choose 32,64,128,256,.....

$$\theta_t = \theta_{t-1} - \lambda \sum_{i=k}^{k+n} \nabla_\theta \mathcal{L}\left(\theta, x^{(i)}, y^i\right)$$

优化算法：Mometum



- 增加动量使得训练能够跳出鞍点 saddle point
- Stochastic Gradient Descent with momentum:

$$v_t = \gamma v_{t-1} + \lambda \nabla_\theta \mathcal{L}(\theta_{t-1})$$

$$\theta_t = \theta_{t-1} - v_t$$

- $\gamma < 1$ resistant parameter, r<1 making the speed will not exceed max speed if there is no friction. (if r=1, means there is no friction, gravitation energy will 100% convert to momentum-> this will likely create overshoot) (if r=0, means no momentum is carried over, which is the same as SGD optimizer) (Usually r set at 0.9)

优化算法：Nesterov Accelerated Gradient

use last step's momentum to update Loss Function, to speed up the training model

- 使用下一步参数处的梯度
- Nesterov Accelerated Gradient:

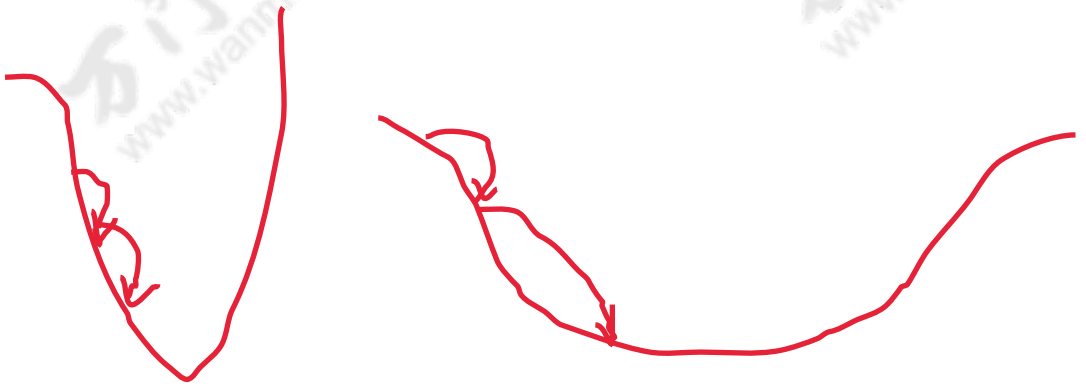
$$v_t = \gamma v_{t-1} + \lambda \nabla_\theta \mathcal{L}(\theta_{t-1} - \gamma v_{t-1})$$

$$\theta_t = \theta_{t-1} - v_t$$

- $\gamma < 1, \gamma \sim 0.9$

优化算法：Adagrad

- 根据参数调整学习率 i.e. if gradient is detected as large, then shorten the learning rate (**步长**), vise versa.
- Adagrad:



$$\theta_{t,i} = \theta_{t-1,i} - \frac{\lambda}{\sqrt{G_{t-1,ii} + \epsilon}} \cdot g_{t-1,i}$$

$$g_{t,i} = \nabla_\theta \mathcal{L}(\theta_{t,i})$$

$$G_{t,ii} = \sum_{j=1}^{t} g_{t,i}^2$$

优化算法：RMSprop

- 根据参数调整学习率, 降低学习率
- Running average of past gradients:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

- Adagrad:

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\lambda}{\sqrt{E[g^2]_{t-1} + \epsilon}} \cdot g_{t-1,i}$$

优化算法：Adadelta

- 根据参数调整学习率, 降低学习率
- Running average of past gradients and parameter changes:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma) \Delta\theta_t^2$$

$$\Delta\theta = \theta_{t+1} - \theta_t$$

- Adagrad:

$$\theta_t = \theta_{t-1,i} - \frac{\sqrt{E[\Delta\theta^2]_{t-2} + \epsilon}}{\sqrt{E[g^2]_{t-1} + \epsilon}} \cdot g_{t-1}$$

this paramater has been replaced/ calculated by a formula (compared to the last page)

优化算法：Adam

= Adadelta+Momentum algorithms

- 根据参数调整学习率, 降低学习率
- Running average of past gradients and moments:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_{t-1}$$

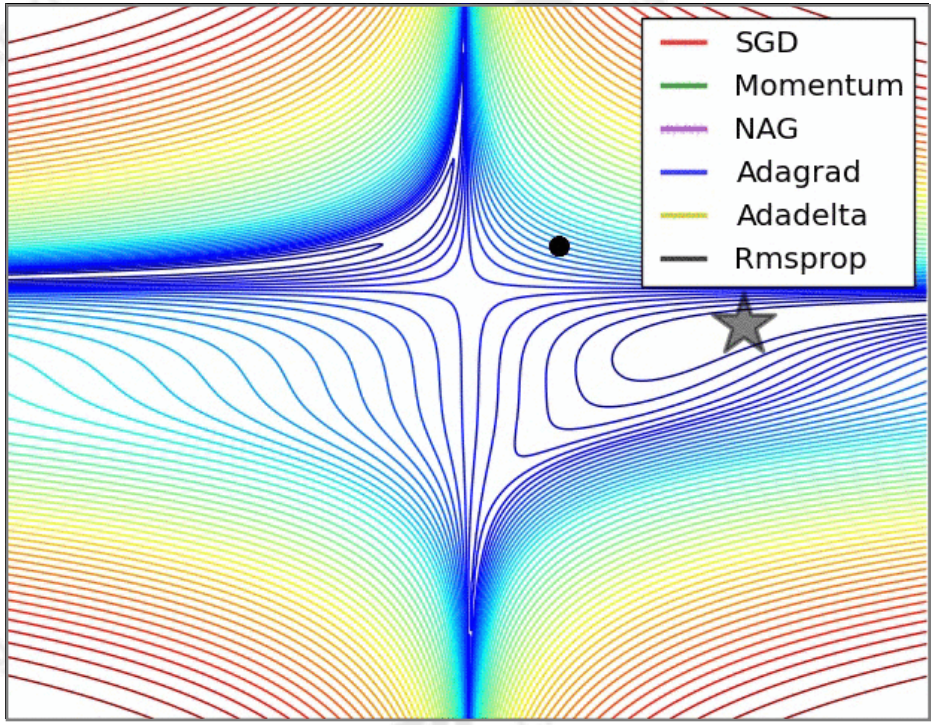
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

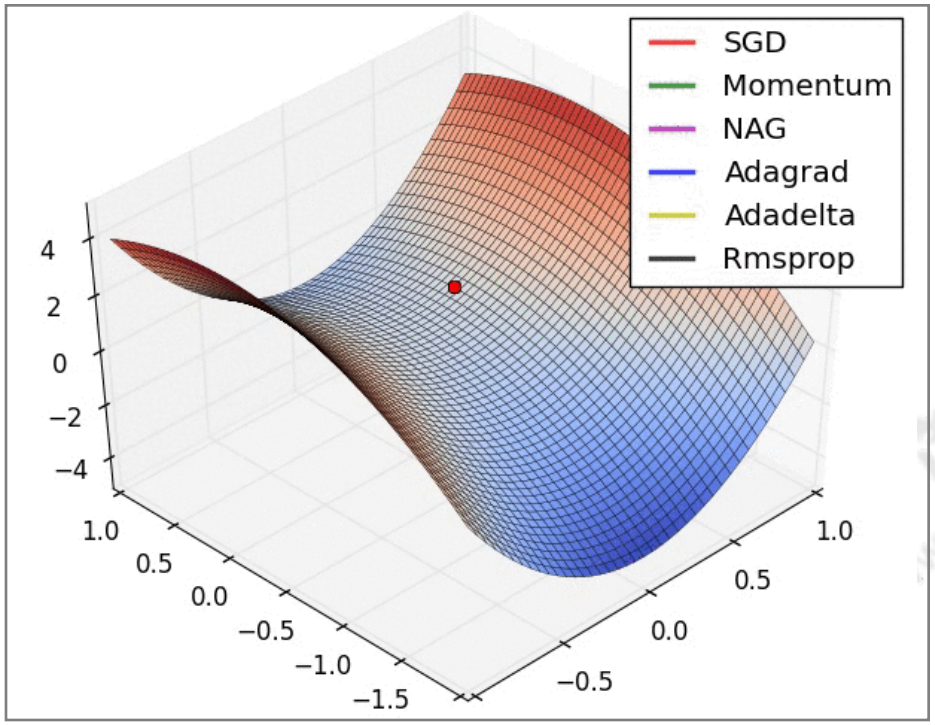
$$\hat{m}_t = {}^{m_t}/_{1-\beta_1}, \hat{v}_t = {}^{v_t}/_{1-\beta_2}$$

- Adagrad:

$$\theta_t = \theta_{t-1} - \frac{\lambda}{\sqrt{v_{t-1} + \epsilon}} \cdot \hat{m}_t$$

优化算法：总结





http://sebastianruder.com/optimizing-gradient-descent

Regularization

Prevent model to overfit

正规化

- L1 and L2
- Early Stopping
- Random Noise (Data Augmentation)
- Dropout
- Weight Normalization
- Batch Normalization

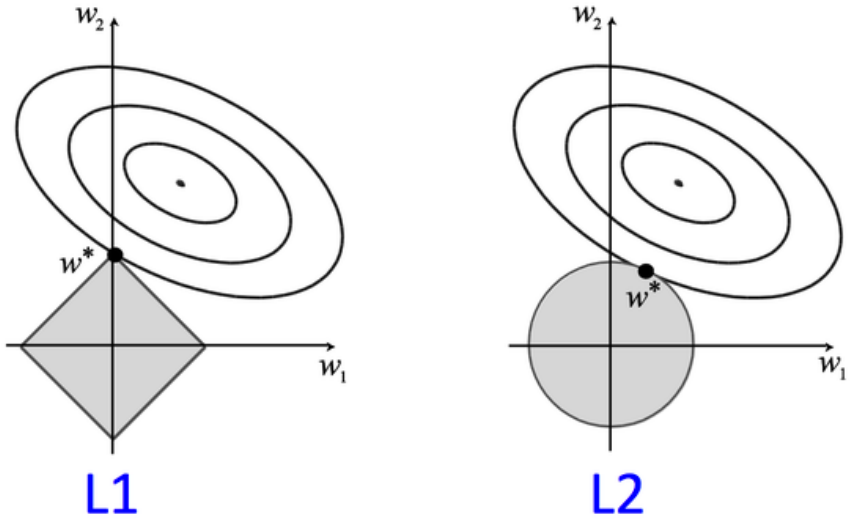
This is not a regularization method

正规化：L1 and L2

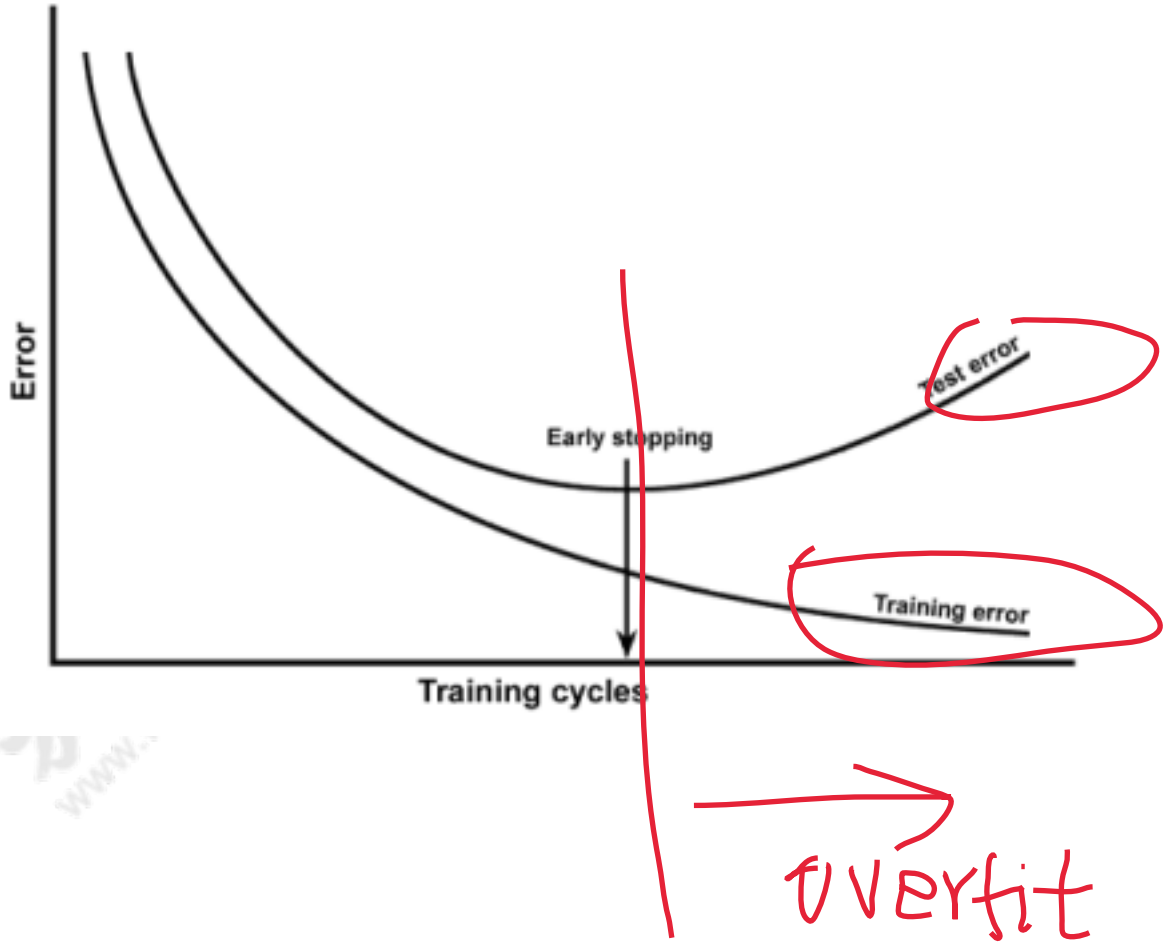
$$L(y, \hat{y}) = \mathcal{L}(y, \hat{y}) + \lambda R(W)$$

L1: $R(w) = \|W\|^2$

L2: $R(w) = \|W\|$

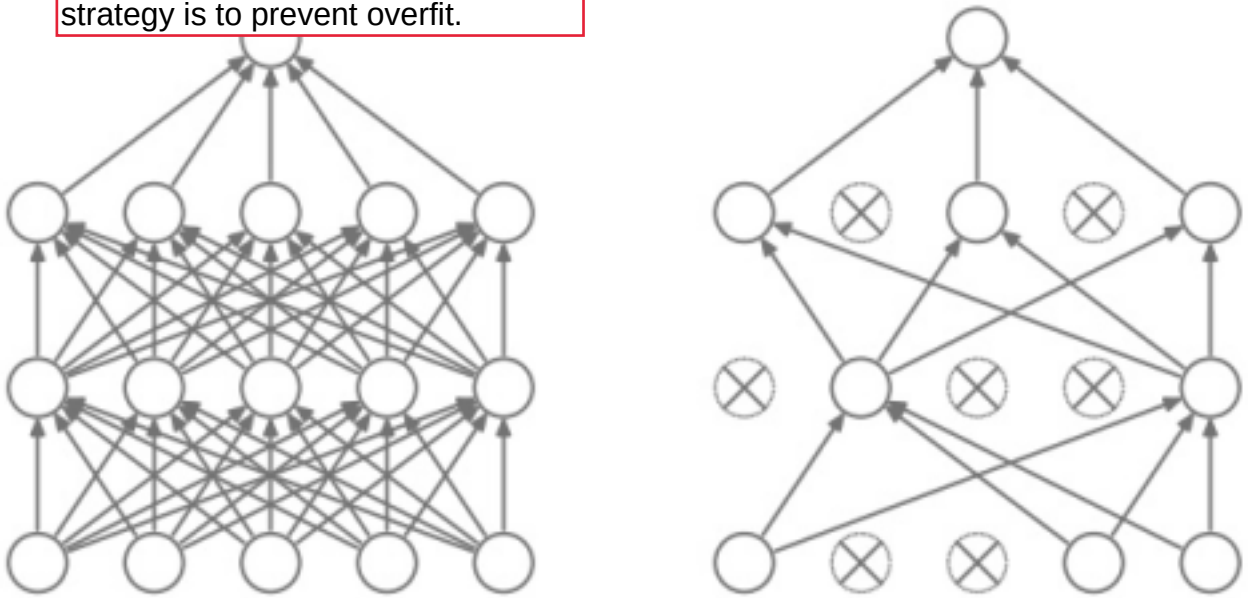


正规化：Early Stopping



正规化：Dropout

randomly freeze some neural, don't update them in current training epoch; in next spoch, de-freeze all neural and random freeze other neural, and so on and so forth. This strategy is to prevent overfit.



(a) Standard Neural Net

(b) After applying dropout.

http://cs231n.github.io/neural-networks-2/

正规化 : Weight Normalization

$$W' = \gamma \frac{W}{||W||}$$

https://arxiv.org/pdf/1602.07868.pdf

正规化：Batch Normalization

this strategy is to prevent covariant shift: each mini-batch distribution might have different mean (noise), normalize those to have mean=0

- mini-batch mean: $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$
- mini-batch variance: $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$
- normalize: $\widehat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$
- scale and shift: $y_i = \gamma\widehat{x_i} + \beta$

https://arxiv.org/pdf/1502.03167.pdf

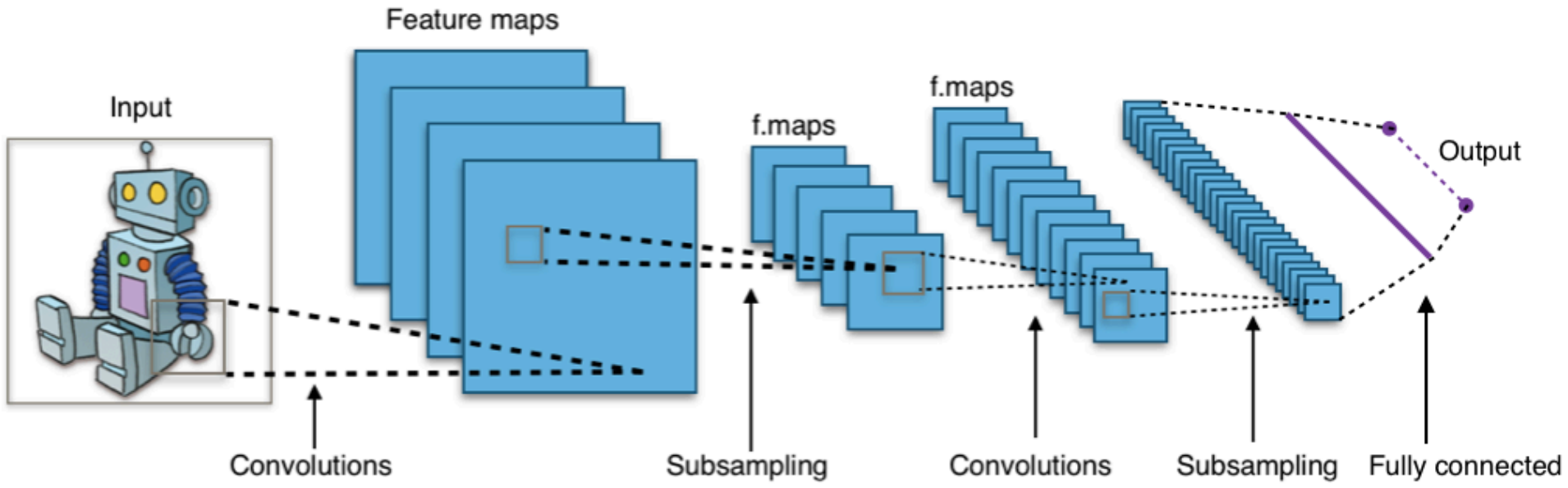
See: Chapter 38 PDF: Convolutional Neural Network for course notes.

卷积神经网络

CNN

Convolutional neural network

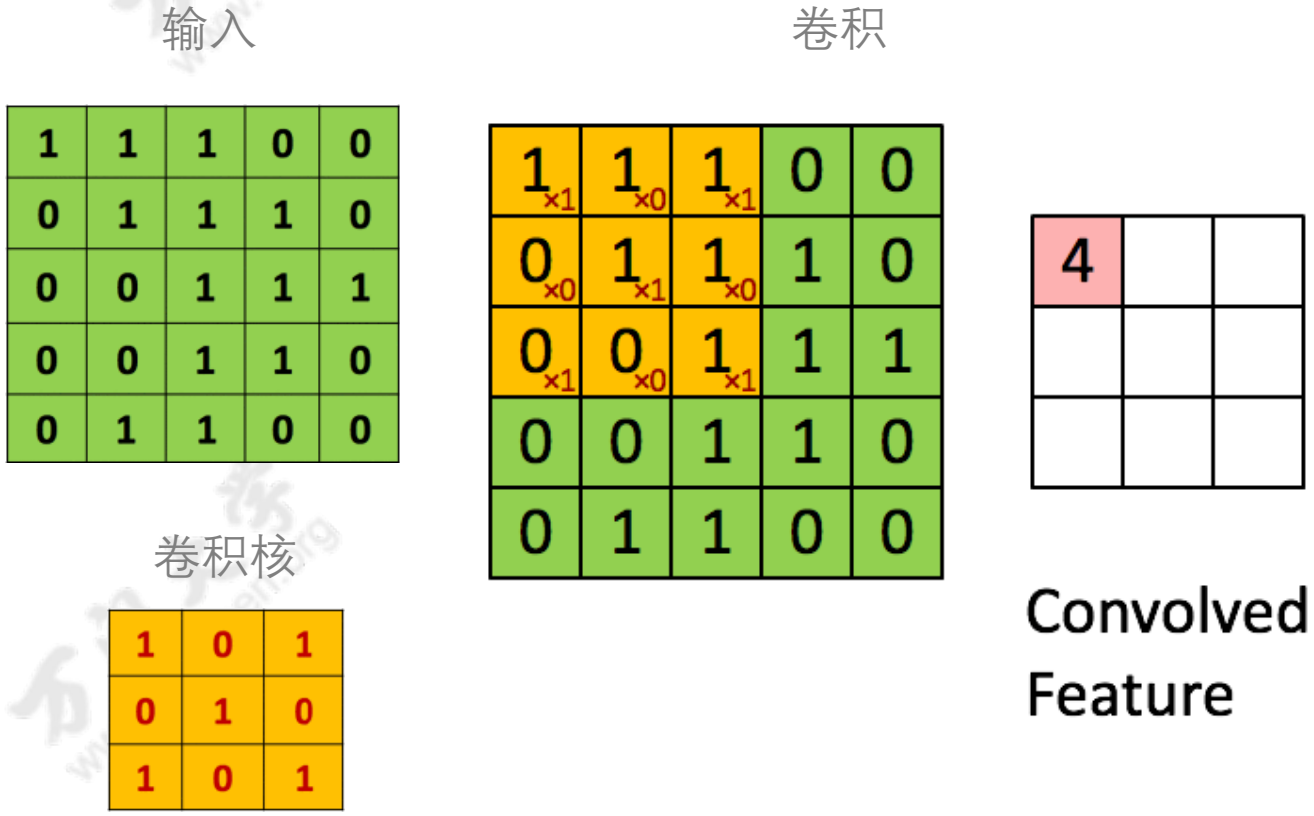
卷积神经网络



https://en.wikipedia.org/wiki/Convolutional_neural_network

Convolution

卷积



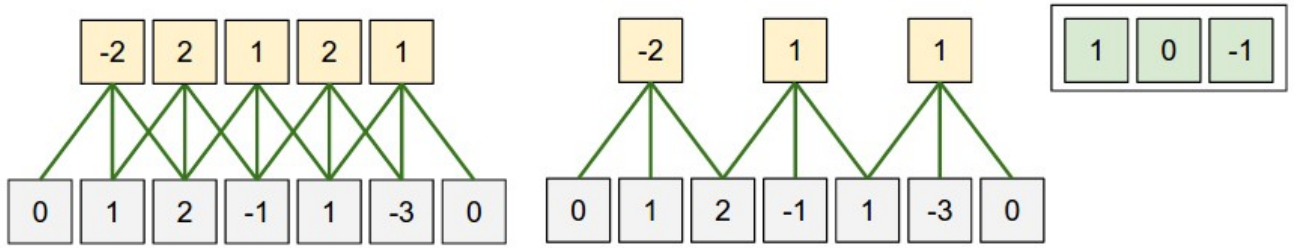
Convolution

卷积

Filter

卷积核

- 尺寸(Size): 卷积核大小(F)
- 深度(Depth): 卷积核个数(K)
- 步长(Stride): 卷积核每次移动的步数(像素点个数)(S)
- 零填充(Zero-padding): 控制输出图片大小(P)



http://cs231n.github.io/convolutional-networks/#architectures

Filter

卷积核

- 输入: $W_1 \times H_1 \times D_1$
- 输出:$W_2 \times H_2 \times D_2$

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

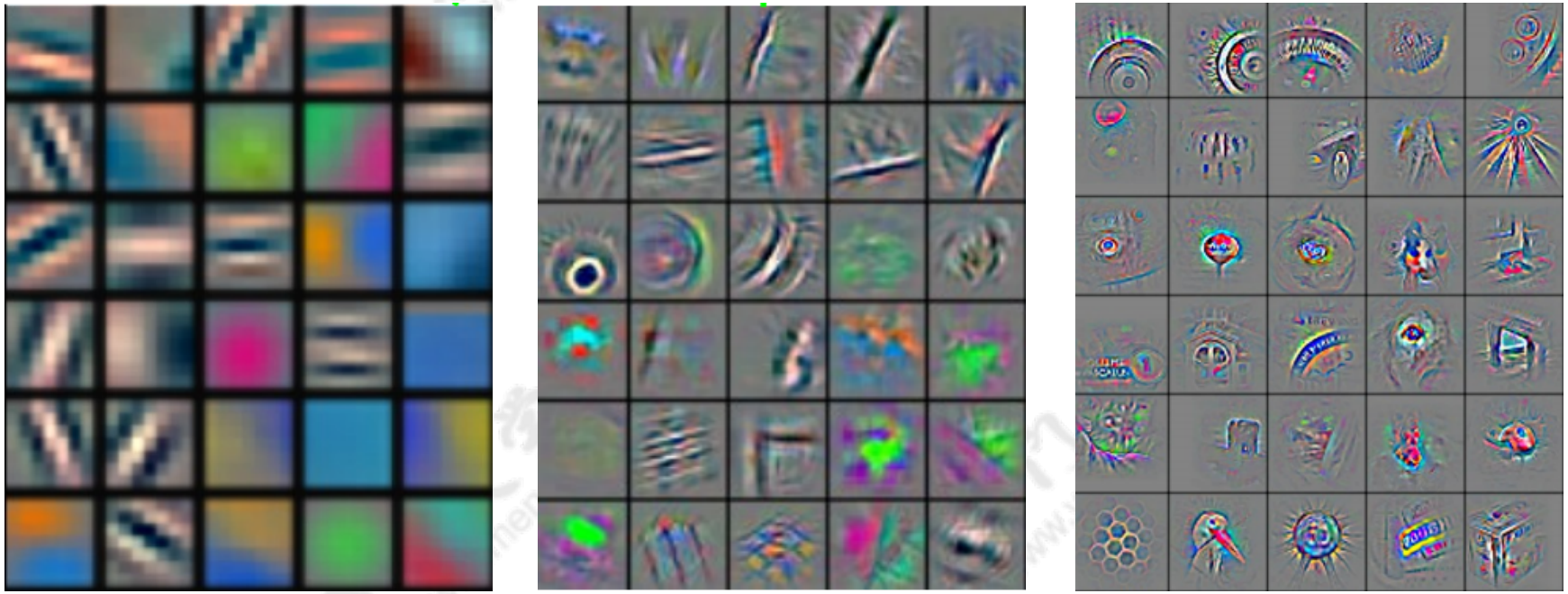
$$H_2 = \frac{W_2 - F + 2P}{S} + 1$$

$$D_2 = K$$

- 参数: $(F \cdot F \cdot D_1) \cdot K$ 权重和K偏差

Filter Visualization

卷积核可视化



https://arxiv.org/pdf/1311.2901.pdf

Padding

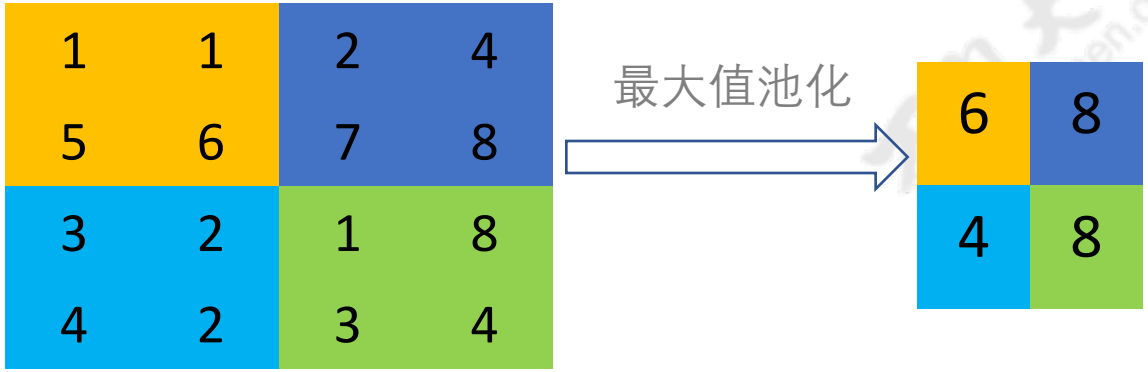
填充

0	0	0	0	0	0
0	32	26	17	34	0
0	2	78	45	23	0
0	4	67	90	12	0
0	68	32	57	25	0
0	0	0	0	0	0

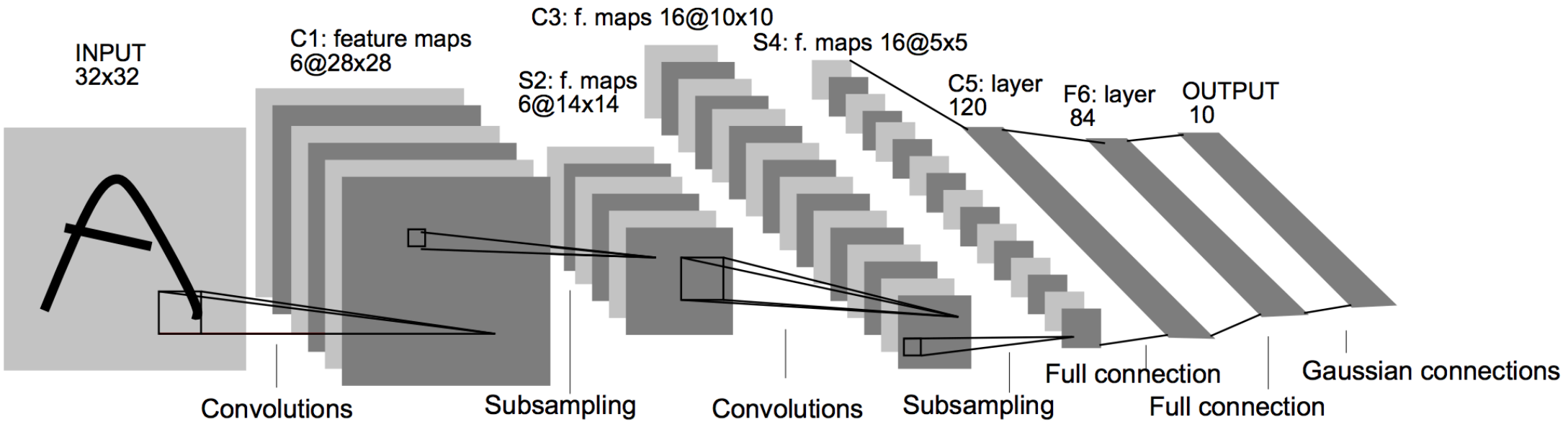
Pooling

池化

- 最大值池化(Max Pooling)
- 平均值池化(Average Pooling)
- L2-范数池化(L2-norm Pooling)

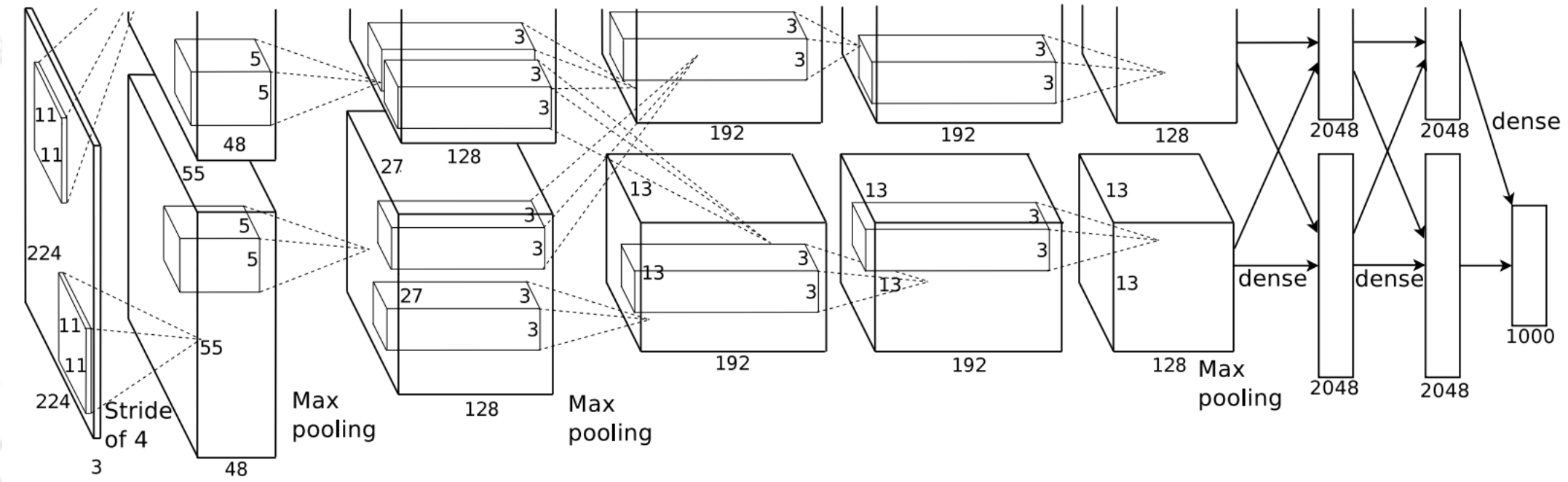


深度卷积神经网络:LeNet-5



http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

深度卷积神经网络:AlexNet

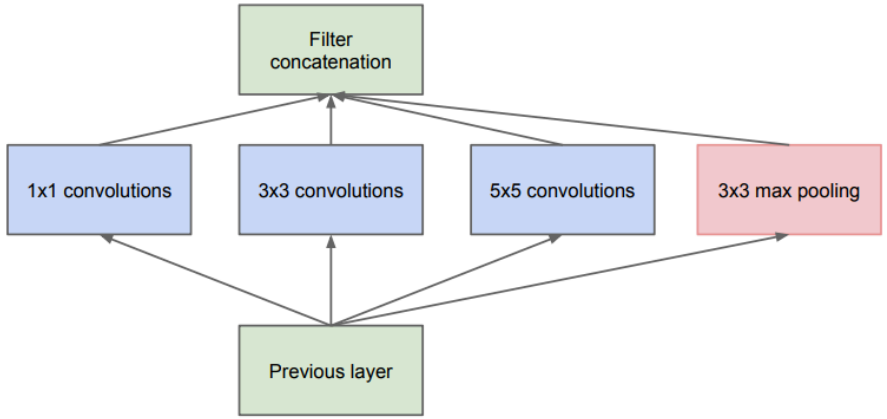


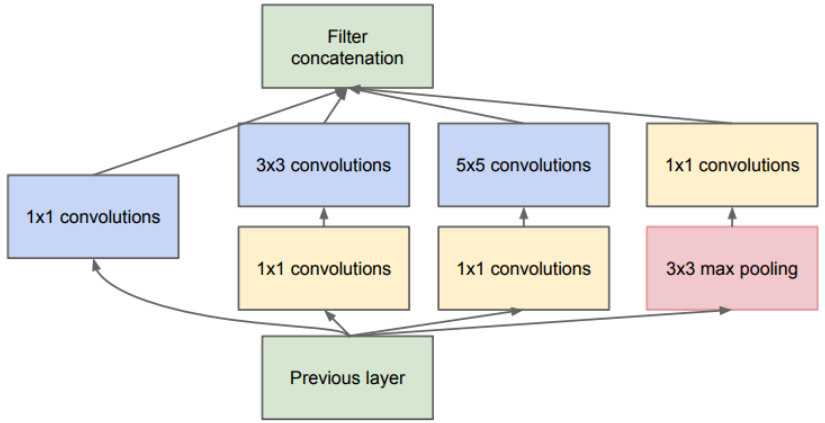
ImageNet Classification with Deep Convolutional Neural Networks

深度卷积神经网络：VGG-16

Very Deep Convolutional Networks for Large-scale Image Recognition

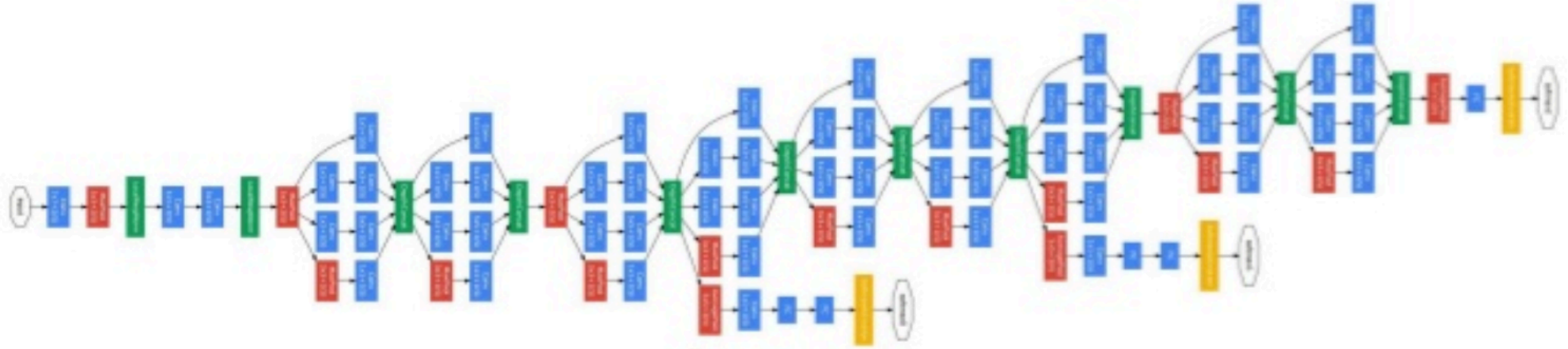
深度卷积神经网络 : Inception Network





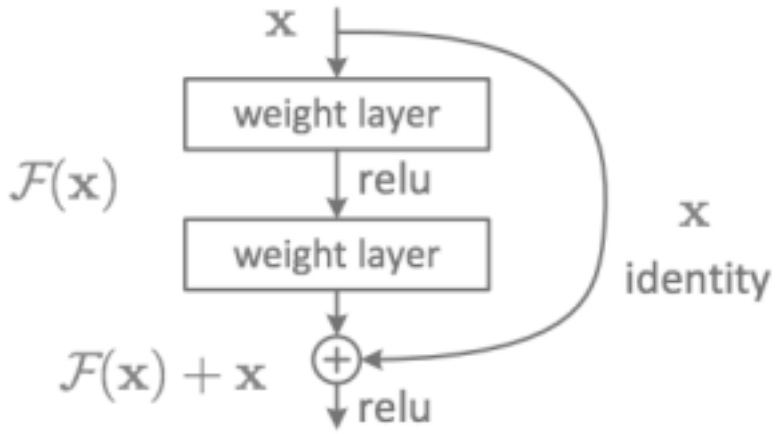
Going deeper with convolutions

深度卷积神经网络 : Inception Network



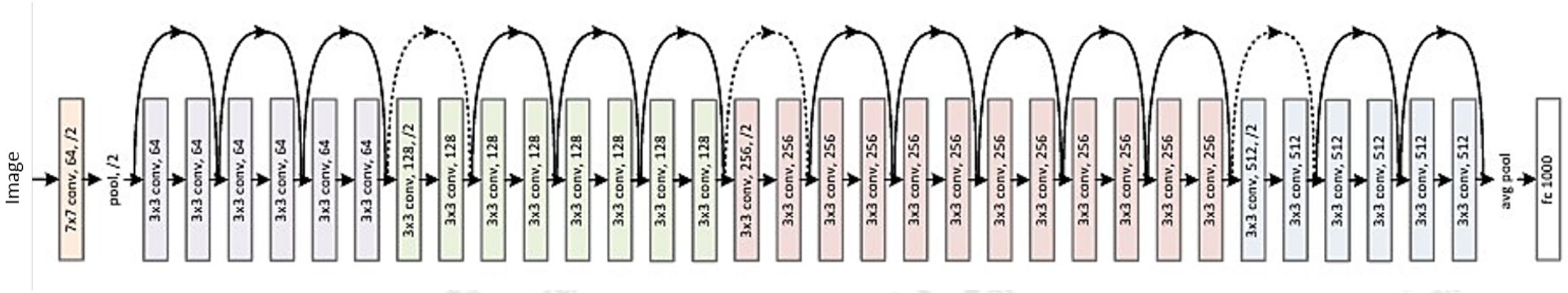
Going deeper with convolutions

深度卷积神经网络 : Residual Network



Deep Residual Learning for Image Recognition

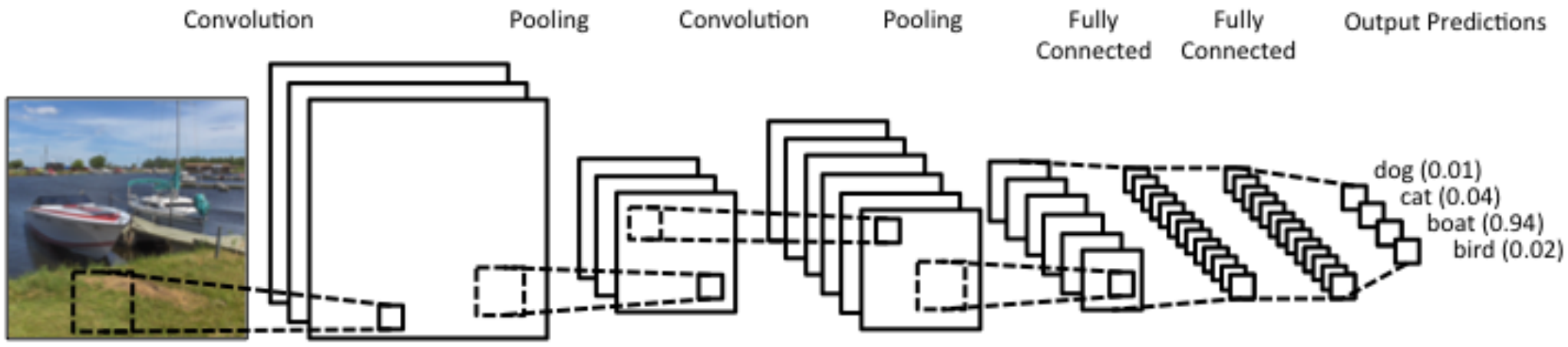
深度卷积神经网络 : Residual Network



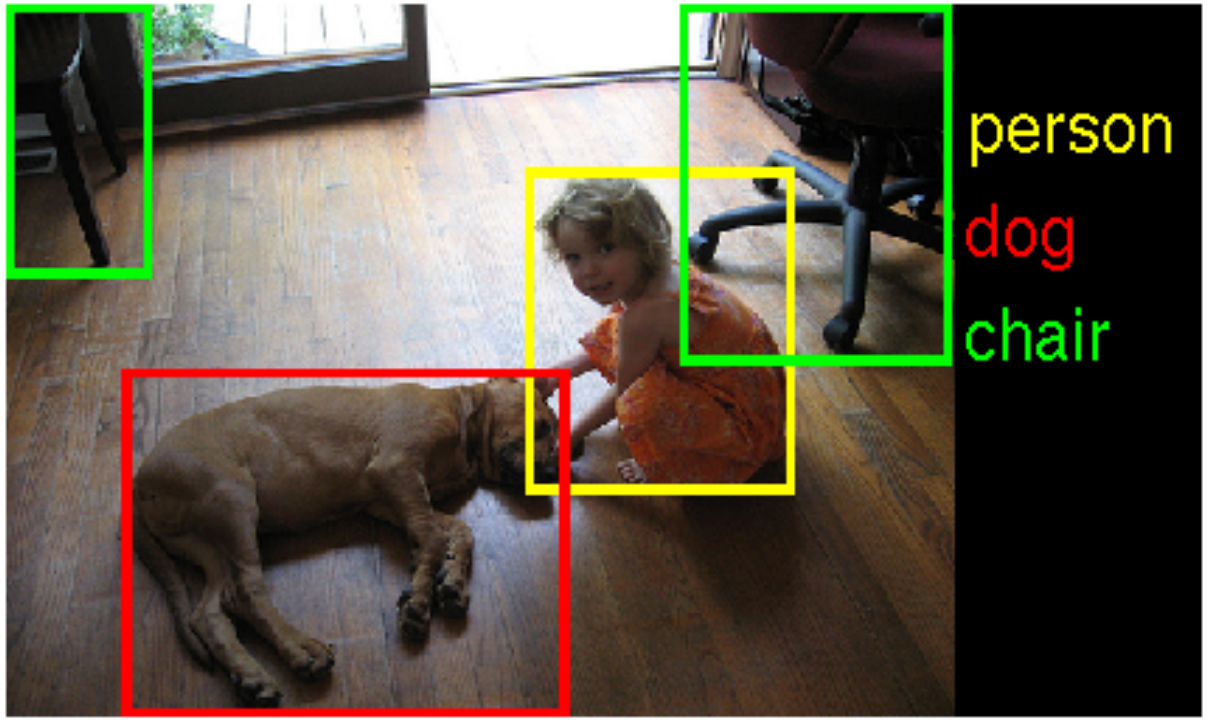
Deep Residual Learning for Image Recognition

卷积神经网络的应用

卷积网络的应用 : 物体分类



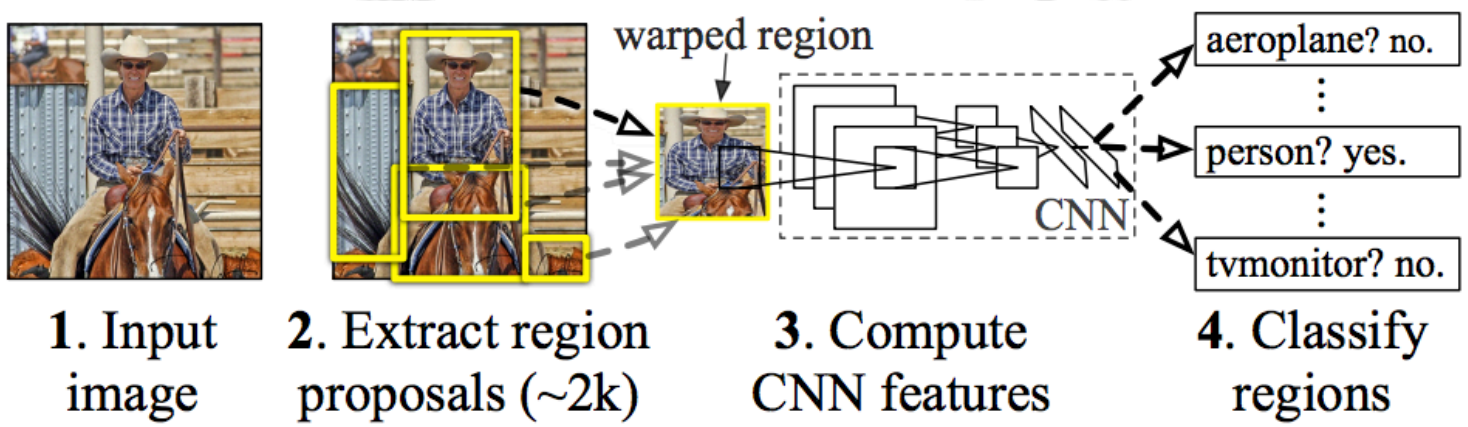
卷积网络的应用 : Object Detection



物体检测：传统方法

物体检测：RCNN

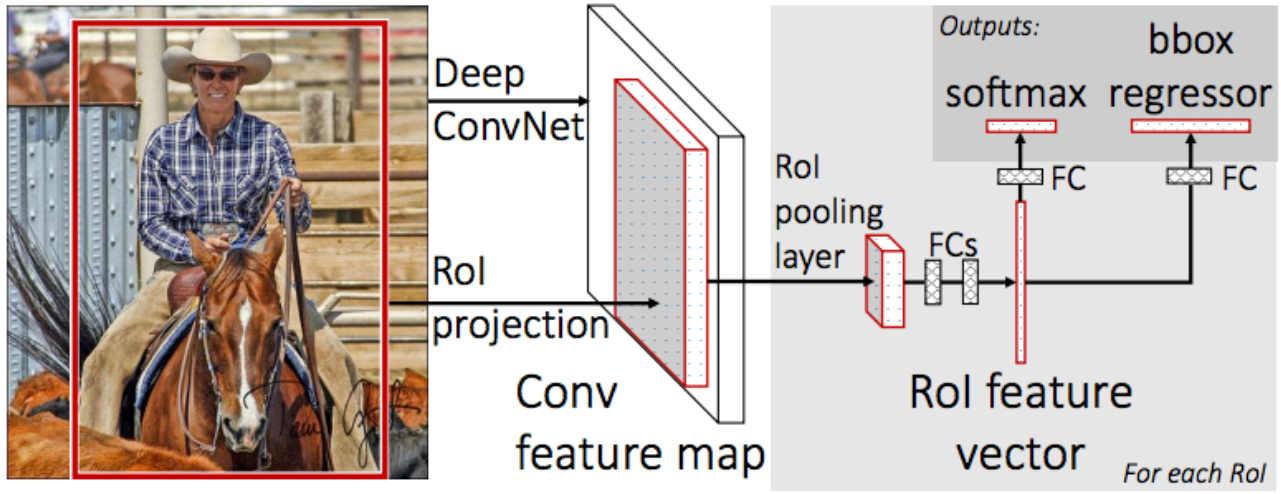
- Selective search算法在图像中提取2000个左右的region proposal
- 每个region proposal缩放成227x227的大小并输入到CNN
- AlexNet for CNN
- CNN的fc7的输出作为特征
- CNN特征输入到SVM进行分类



Rich feature hierarchies for accurate object detection and semantic segmentation

物体检测：Fast R-CNN

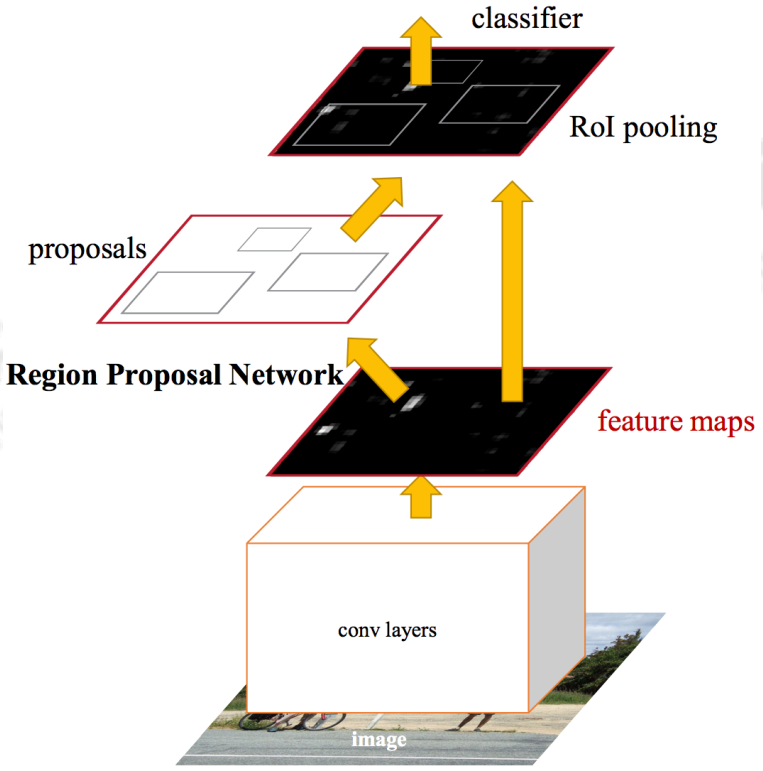
- 最后一个卷积层加入一个ROI pooling layer
- 损失函数使用多任务损失函数(multi-task loss)
- 接近端到端训练
- Region proposal 2~3s, Feature extraction 0.32s



Fast R-CNN

物体检测：Faster R-CNN

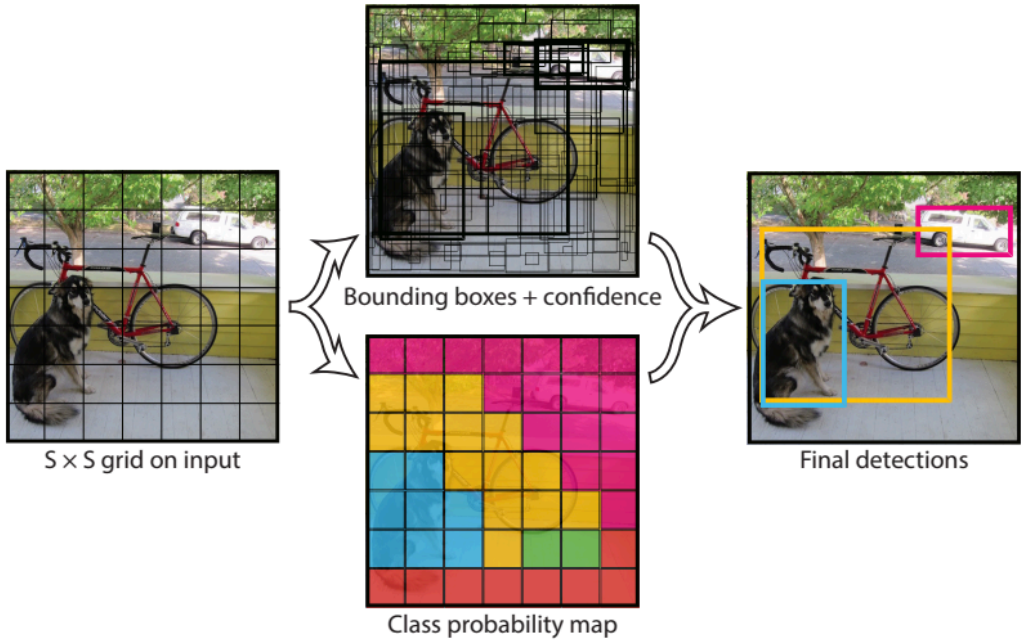
- Region Proposal Network
- 损失函数使用多任务损失函数(multi-task loss)
- 端到端训练
- 目标检测0.2s



Faster R-CNN

物体检测：YOLO

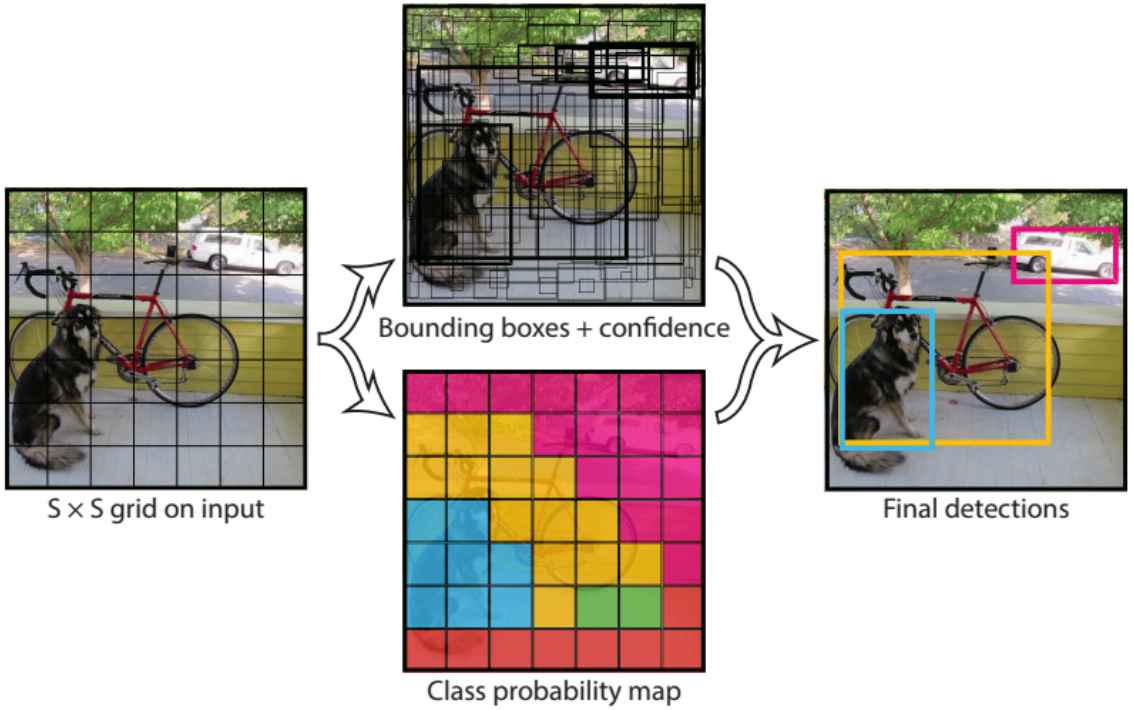
- 将图像划分为7x7的网格
- 对每个网格，预测两个边框(每个边框是目标的概率以及每个边框区域在多个类别上的概率
- 根据阈值去除可能性比较低的目标窗口
- 目标检测0.02s



You Only Look Once: Unified, Real-Time Object Detection

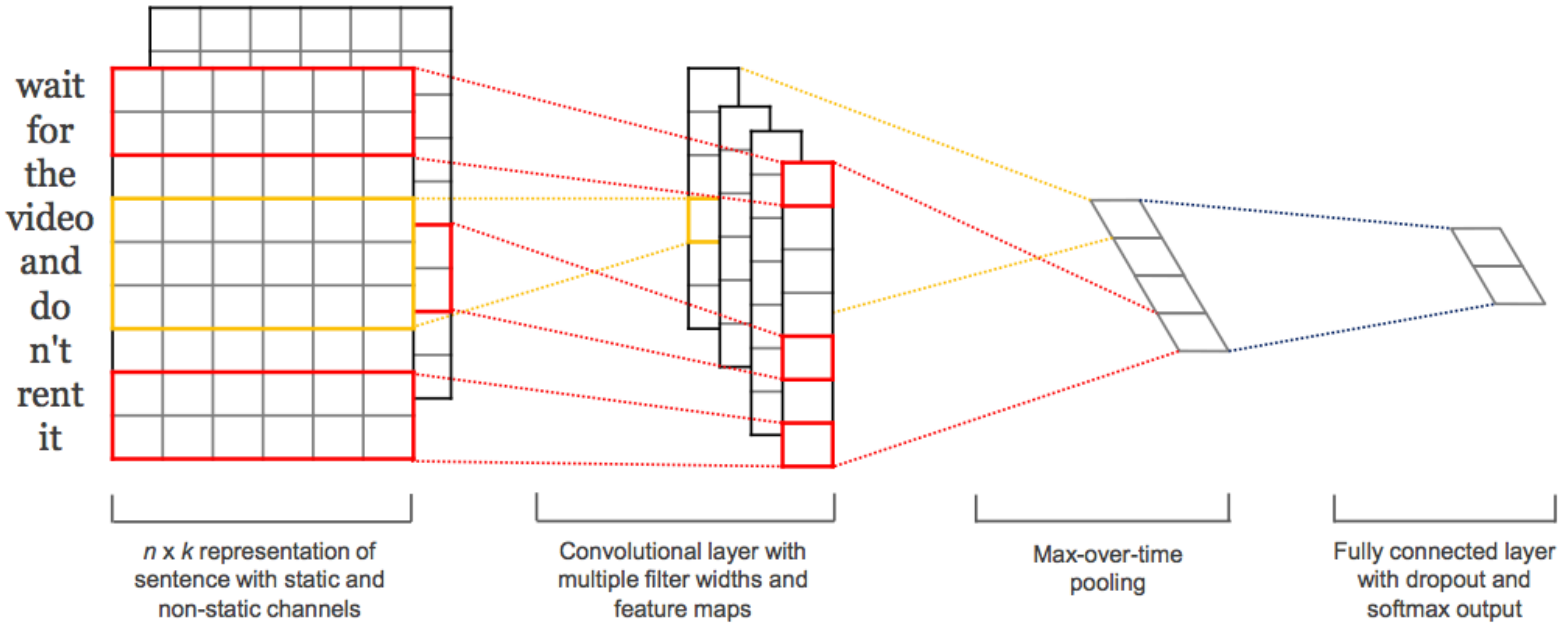
物体检测：SSD

- 使用目标位置周围特征来预测目标位置
- 使用Faster R-CNN的anchor机制
- 目标检测0.02s



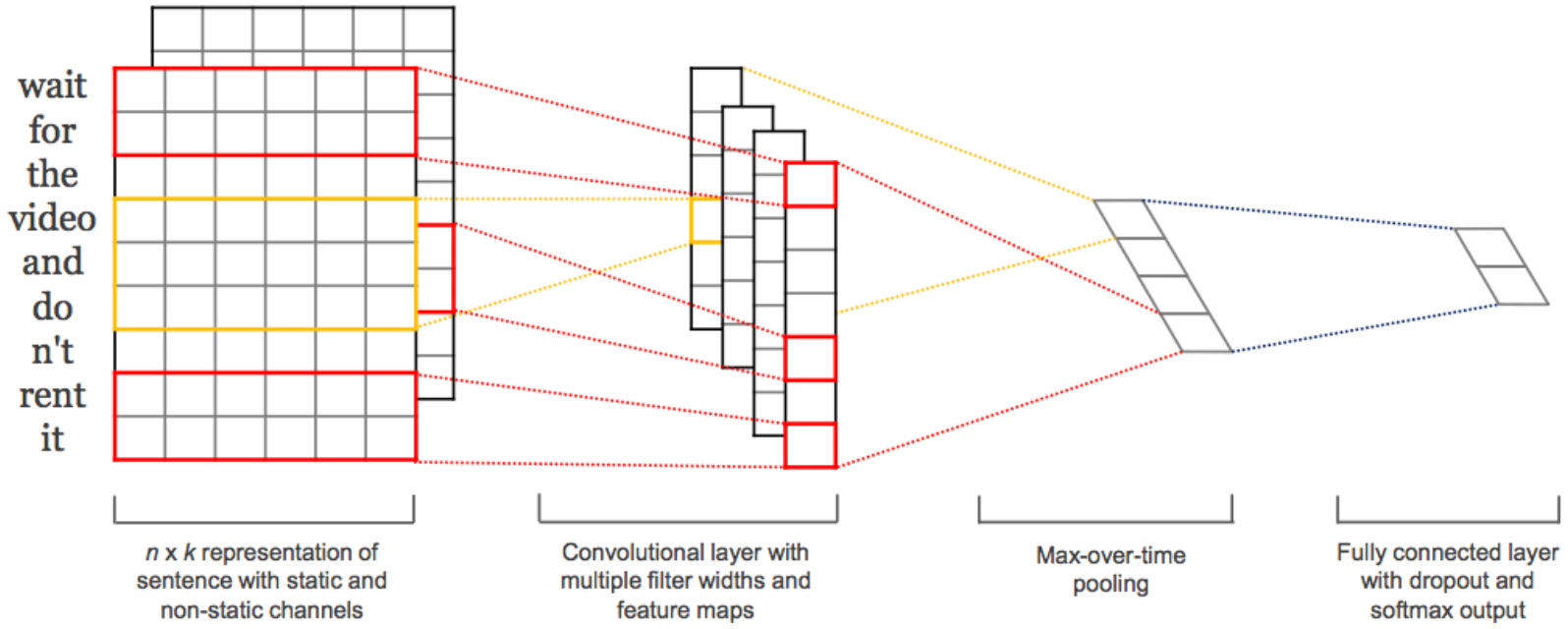
SSD: Single Shot MultiBox Detector

卷积网络的应用：文档的归类



Convolutional Neural Networks for Sentence Classification

卷积网络的应用：文档的归类



Convolutional Neural Networks for Sentence Classification

THANKS.