Informal Introduction To AI

Huang Yuefeng (William) @ ZValley

Jan. 2019

https://github.com/old-huang

Overview

- Hello World !
- Fundamentals
- What can CV do?
- What can NLP do?
- ZValley Scenarios
- How are We Doing ?
- 2019 Work Overview
- Conclusion
- Appendix

Hello World

- 手写识别

训练集55000

测试集10000

模型抽象为f(x)=y

x是28*28*1的矩阵

y是1,2,3,…,0

```python
import tensorflow.examples.tutorials.mnist.input_data as input_data
import tensorflow as tf

mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

#define convolution and max_pool operations
def conv2d(x,w):
    return tf.nn.conv2d(x,w,strides=[1,1,1,1],padding='SAME')
def max_pool_2x2(x):
    return tf.nn.max_pool(x,ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')

#define input
x_input=tf.placeholder(tf.float32,[None,784])
x=tf.reshape(x_input,[-1,28,28,1])

#define first layer
w_conv1=tf.Variable(tf.truncated_normal([3,3,1,32],stddev=0.1))
b_conv1=tf.Variable(tf.constant(0.1,shape=[32]))
h_conv1=tf.nn.relu(conv2d(x,w_conv1)+b_conv1)
h_pool1=max_pool_2x2(h_conv1)

#define second layer
w_conv2=tf.Variable(tf.truncated_normal([3,3,32,50],stddev=0.1))
b_conv2=tf.Variable(tf.constant(0.1,shape=[50]))
h_conv2=tf.nn.relu(conv2d(h_pool1,w_conv2)+b_conv2)
h_pool2=max_pool_2x2(h_conv2)

#define third layer
w_fc1=tf.Variable(tf.truncated_normal([7*7*50,1024],stddev=0.1))
b_fc1=tf.Variable(tf.constant(0.1,shape=[1024]))
h_pool2_flat=tf.reshape(h_pool2,[-1,7*7*50])
h_fc1=tf.nn.relu(tf.matmul(h_pool2_flat,w_fc1)+b_fc1)

#define classifier
w_fc2=tf.Variable(tf.truncated_normal([1024,10],stddev=0.1))
b_fc2=tf.Variable(tf.constant(0.1,shape=[10]))
y=tf.nn.softmax(tf.matmul(h_fc1,w_fc2)+b_fc2)

#define loss function
y_gt=tf.placeholder(tf.float32,[None,10])
loss=tf.reduce_mean(-tf.reduce_sum(y_gt*tf.log(y),reduction_indices=[1]))

#define optimizer
train_step=tf.train.AdamOptimizer(1e-4).minimize(loss)

#define evaluation
correct_prediction=tf.equal(tf.argmax(y,1),tf.argmax(y_gt,1))
accuracy=tf.reduce_mean(tf.cast(correct_prediction,tf.float32))

#after definition, run training
with tf.Session() as sess:
    tf.global_variables_initializer().run()
    for i in range(20000):
        batch=mnist.train.next_batch(50)
        if i%100==0:
                train_accuracy=accuracy.eval(feed_dict={x_input:batch[0],y_gt:batch[1]})
                print "step %d,train_accuracy= %g"%(i,train_accuracy)
                train_step.run(feed_dict={x_input:batch[0],y_gt:batch[1]})

    print "test_accuracy= %g"%accuracy.eval(feed_dict={x:mnist.test.images,y_gt:mnist.test.labels})
```

DL Solution

1.定义输入

2.定义网络结构

3.转化为输出向量

4.定义分类

5.定义目标函数

6.定义优化方法

7.将数据灌入上面定义好的模型，开始训练

8.预测未知数据

ML Solution

```
1 import tensorflow.examples.tutorials.mnist.input_data as input_data
2 from sklearn.ensemble import RandomForestClassifier
3 import numpy as np
4
5 mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
6 clf_rf = RandomForestClassifier(n_estimators=150)
7 clf_rf.fit(mnist.train.images,mnist.train.labels)
8 y = clf_rf.predict(mnist.test.images)
9 accuracy = np.mean(np.equal(y,mnist.test.labels))
10 print "test_accuracy = %g"%accuracy
```

1.定义模型

2.训练模型

3.预测未知数据

```python
import tensorflow.examples.tutorials.mnist.input_data as input_data
import tensorflow as tf

mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

#define convolution and max_pool operations
def conv2d(x,w):
    return tf.nn.conv2d(x,w,strides=[1,1,1,1],padding='SAME')
def max_pool_2x2(x):
    return tf.nn.max_pool(x,ksize=[1,2,2,1],strides=[1,2,2,1],padding='SAME')

#define input
x_input=tf.placeholder(tf.float32,[None,784])
x=tf.reshape(x_input,[-1,28,28,1])

#define first layer
w_conv1=tf.Variable(tf.truncated_normal([3,3,1,32],stddev=0.1))
b_conv1=tf.Variable(tf.constant(0.1,shape=[32]))
h_conv1=tf.nn.relu(conv2d(x,w_conv1)+b_conv1)
h_pool1=max_pool_2x2(h_conv1)

#define second layer
w_conv2=tf.Variable(tf.truncated_normal([3,3,32,50],stddev=0.1))
b_conv2=tf.Variable(tf.constant(0.1,shape=[50]))
h_conv2=tf.nn.relu(conv2d(h_pool1,w_conv2)+b_conv2)
h_pool2=max_pool_2x2(h_conv2)

#define third layer
w_fc1=tf.Variable(tf.truncated_normal([7*7*50,1024],stddev=0.1))
b_fc1=tf.Variable(tf.constant(0.1,shape=[1024]))
h_pool2_flat=tf.reshape(h_pool2,[-1,7*7*50])
h_fc1=tf.nn.relu(tf.matmul(h_pool2_flat,w_fc1)+b_fc1)

#define classifier
w_fc2=tf.Variable(tf.truncated_normal([1024,10],stddev=0.1))
b_fc2=tf.Variable(tf.constant(0.1,shape=[10]))
y=tf.nn.softmax(tf.matmul(h_fc1,w_fc2)+b_fc2)

#define loss function
y_gt=tf.placeholder(tf.float32,[None,10])
loss=tf.reduce_mean(-tf.reduce_sum(y_gt*tf.log(y),reduction_indices=[1]))

#define optimizer
train_step=tf.train.AdamOptimizer(1e-4).minimize(loss)

#define evaluation
correct_prediction=tf.equal(tf.argmax(y,1),tf.argmax(y_gt,1))
accuracy=tf.reduce_mean(tf.cast(correct_prediction,tf.float32))

#after definition, run training
with tf.Session() as sess:
    tf.global_variables_initializer().run()
    for i in range(20000):
        batch=mnist.train.next_batch(50)
        if i%100==0:
                train_accuracy=accuracy.eval(feed_dict={x_input:batch[0],y_gt:batch[1]})
                print "step %d,train_accuracy= %g"%(i,train_accuracy)
                train_step.run(feed_dict={x_input:batch[0],y_gt:batch[1]})

    print "test_accuracy= %g"%accuracy.eval(feed_dict={x:mnist.test.images,y_gt:mnist.test.labels})
```

```python
import tensorflow.examples.tutorials.mnist.input_data as input_data
from sklearn.ensemble import RandomForestClassifier
import numpy as np

mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
clf_rf = RandomForestClassifier(n_estimators=150)
clf_rf.fit(mnist.train.images,mnist.train.labels)
y = clf_rf.predict(mnist.test.images)
accuracy = np.mean(np.equal(y,mnist.test.labels))
print "test_accuracy = %g"%accuracy
```

Fundamentals

- 学习问题的基本范式

  f(*X*)=*Y, X* and *Y* is vector.

  *f*被称为模型（也就是一个多维变量的函数）

  根据已知的*X*和*Y*，建立模型，成为训练（或者学习）

  根据训练好的模型对未知的*X*求得对应的*Y*，叫做预测

- ML和DL的区别

  *ML*用于结构性数据：特征向量（需要人工选择）

  *DL*用于非结构性数据：文本，图片 （也可以将其直接作为特征向量，但精度不好）

  （不需要人工选择向量）

- Bechmark

  模型精度（工业上对于*90%*还是*91%*都是一样的，小数点以后的精度只对学术有意义，实际使用要通过数据处理让模型接近学术上的精度）

  模型性能（在边缘计算的实时处理中直接影响用户体验，非常重要）

  模型普适性（在预测集上面精度与训练集接近，衰减不太厉害，*DL*和*ML*是对数据非常敏感的技术，这点最难）

What to Do ?

- 数据处理：数值化，去异常值，补缺失值，正则化，白化，改变统计分布
- 数据标注
- 特征选择 （专门针对ML，DL无需人工选择特征）
- 模型设计（专门针对DL，ML模型结构基本固定）
- 参数调整：改变模型参数，使得精度更高

- 下面针对模型设计和参数调整两方面向下分解

Multi-Classifier

- hypothesis function: $h_\theta(x) = P(y = 1|x, \theta) = \frac{1}{1+e^{-\theta^T x}}$
- $g(z) = \frac{1}{1+e^{-z}}$ is logistic function or sigmoid function

  $g'(z) = ((1 + e^{-z})^{-1})'(e^{-z})' = ((1 + e^{-z})^{-2})(e^{-z})$

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

  $(f'(g(x)) = f'(x)g'(x))$

  $g(z)$ is calculated with low cost

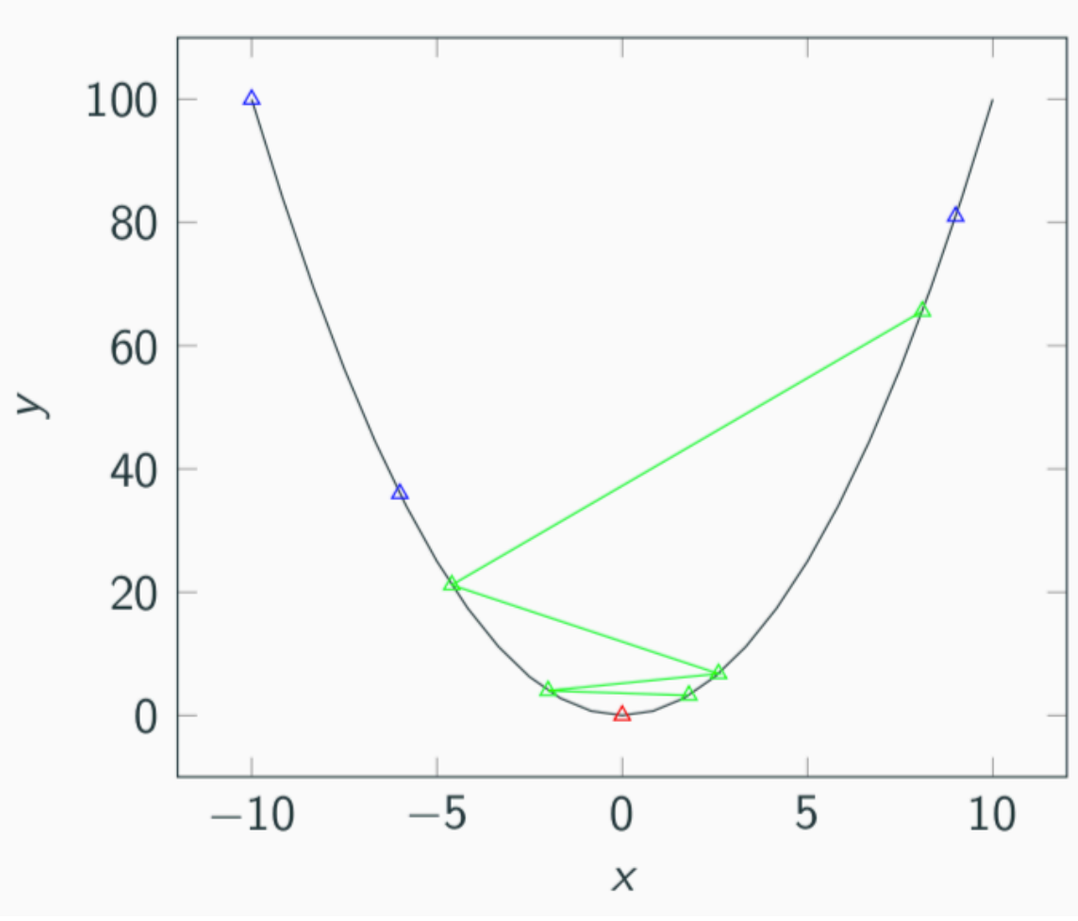
  $h_\theta(x) = g(\theta^T x)$
- $p(y|x, \theta) = P(y = 1|x, \theta)^y P(y = 0|x, \theta)^{1-y} = (h_\theta(x))^y(1 - (h_\theta(x))^{1-y}$
- loss function (actually it is already not loss function, but target function) by Maximum Likelihood:

  $L(\theta) = \prod_{i=1}^m p(y^{(i)}|x^{(i)}, \theta) = \prod_{i=1}^m (h_\theta(x^{(i)}))^{y^{(i)}}(1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$
- log loss function:

  $l(\theta) = \log L(\theta) = \sum_{i=1}^m (y^{(i)} \log h_\theta(x) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})))$

Random Local Search

- question definition: $y = f(x)$, find the minimum value of y.
- random change x to find a better y until y is converged.
- search direction: Δx is random
- search step: t is a small value
- update: $x^{(k+1)} = x^{(k)} + t\Delta x$
- finish: $\Delta y < \mathit{tolerance}$
- issue: forbid to get fake minimum y by small $t\Delta x$



Gradient Descent

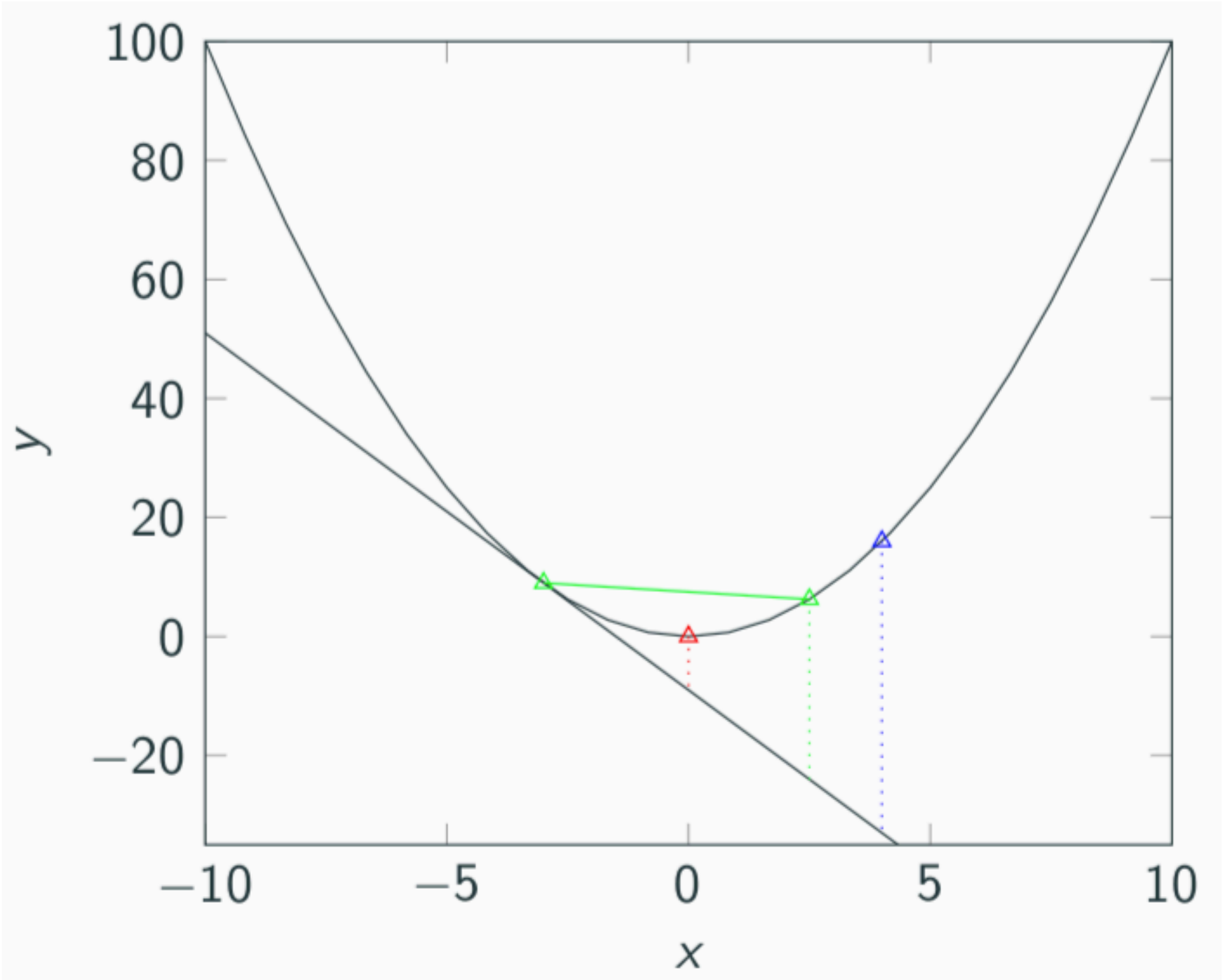
- motivation:change x in the direction of gradient, which is fast to minimum.

  Assume a function can be expressed as a converged summary of a series of power series, in the adjacent domain of x^0, it is:

  $f(x) = f(x^0) + \frac{f^{(1)}(x^0)}{1!}(x - x^0) + \frac{f^{(2)}(x^0)}{2!}(x - x^0)^2 + \ldots + \frac{f^{(n)}(x^0)}{n!}(x - x^0)^n$

  use $f(x) \approx f(x^0) + \frac{f^{(1)}(x^0)}{1!}(x - x^0)$ to get search direction, i.e. first order.

- search direction: $\Delta x = f^{(1)}(x^0) = \nabla f(x^0)$
- search step: t is a small value
- update: $x^{(k+1)} = x^{(k)} + t\nabla f(x^k)$
- finish: $\nabla f(x^k) < tolerance$
- issue: how to choose t, please see SGD tricks.

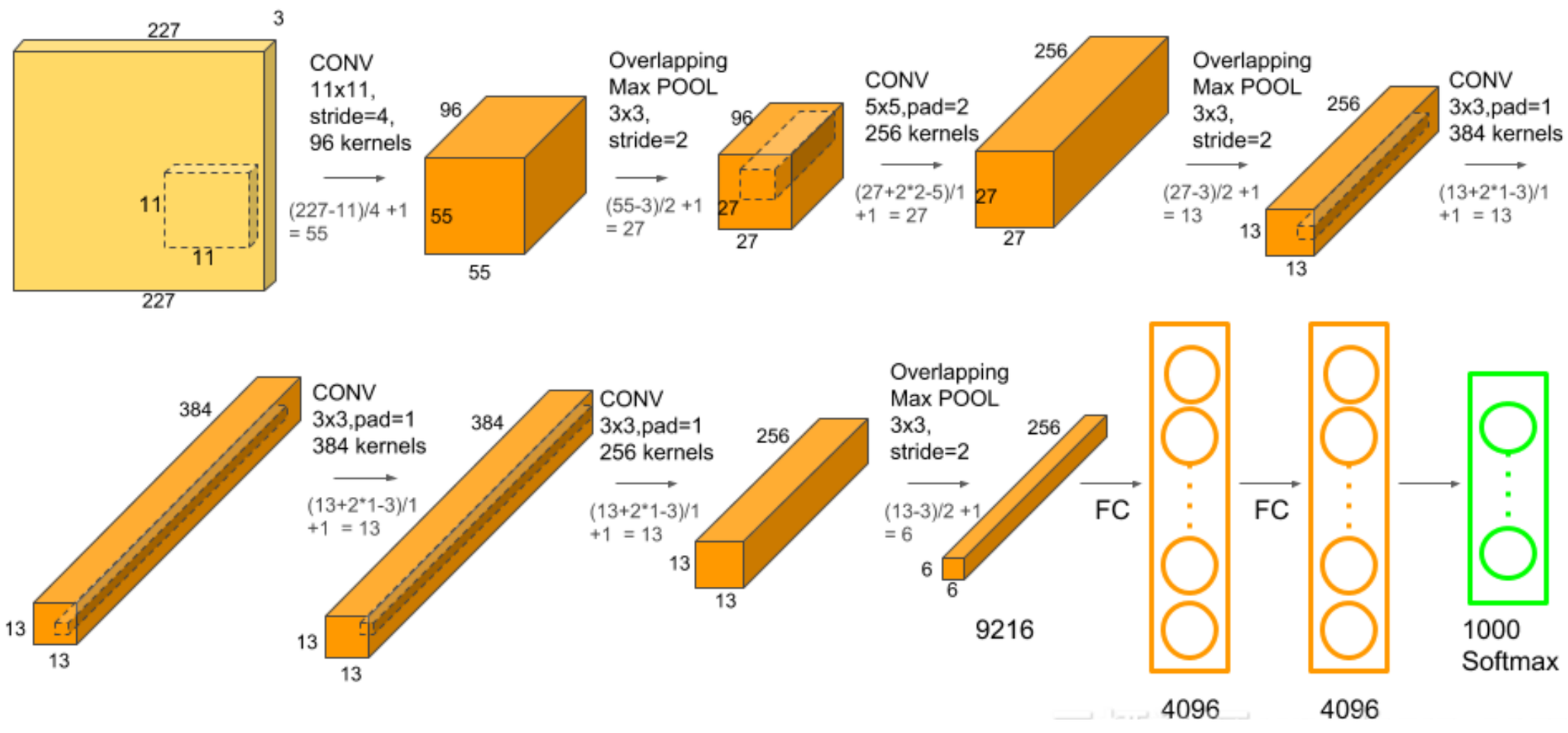


$f(x) = x^2$ and $y - y^0 = f'(x - x^0)$

Neural Network Structure

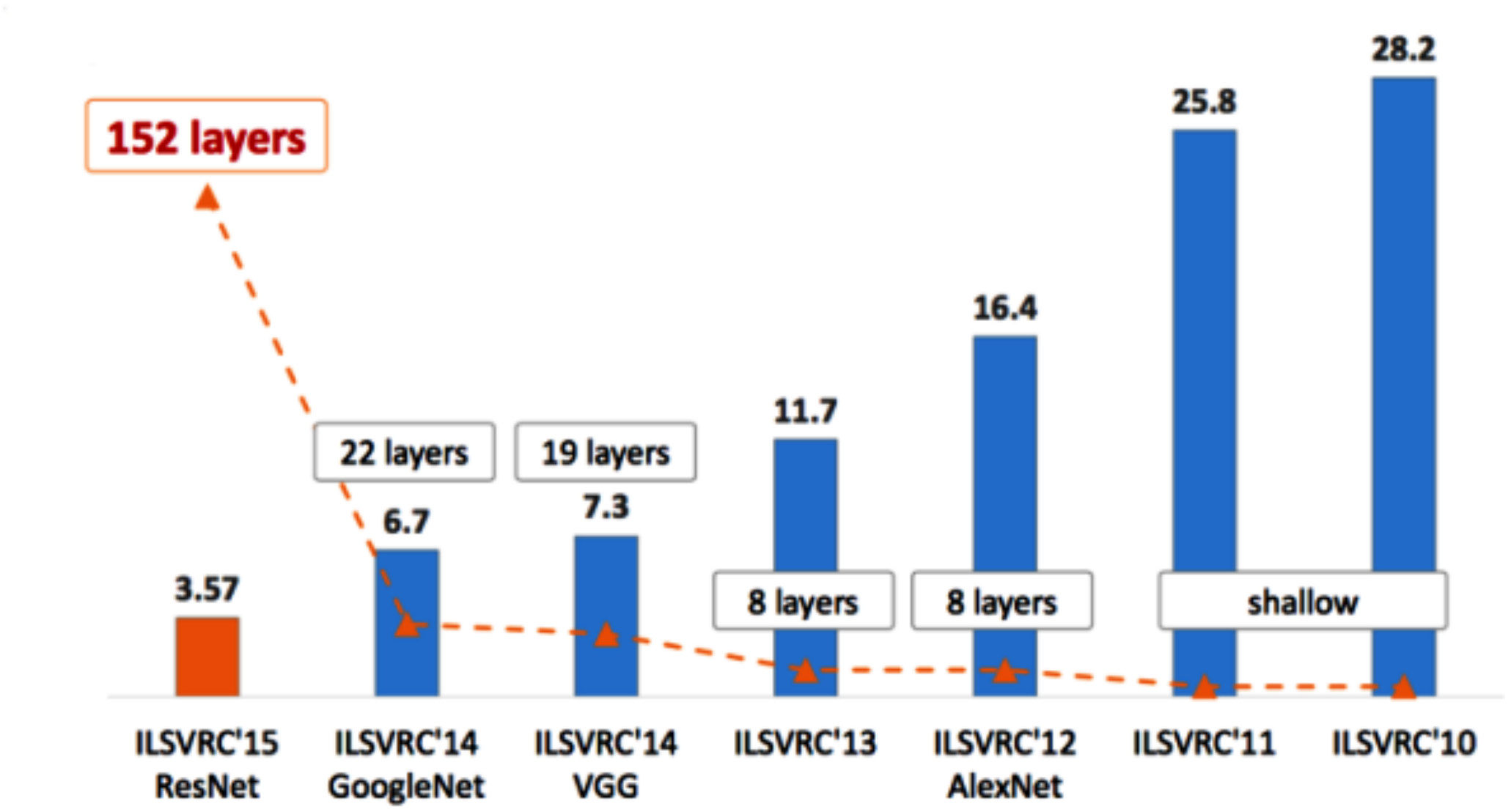
- Layer by layer

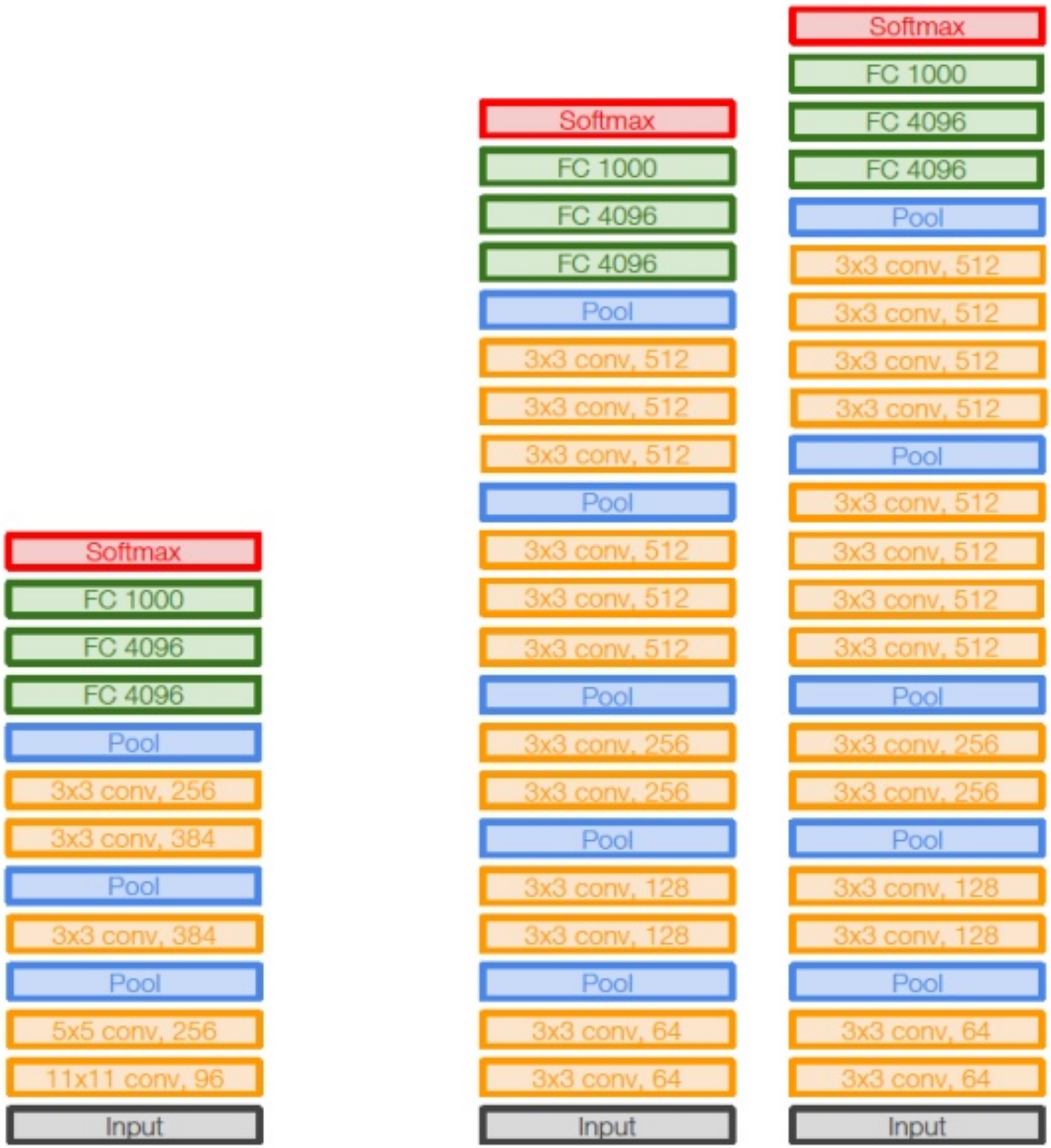
- Input+a*(Conv+Pool)+b*(Conv)+c*(FC)+classifier(regression)



Non-Linear increase model expressive ability of function

- From softmax to AlexNet, you can get this.

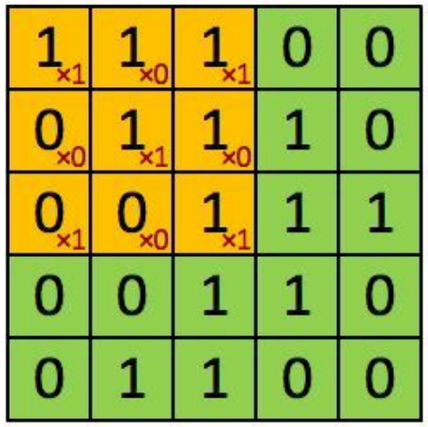



Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 256
3x3 conv, 384
Pool
3x3 conv, 384
Pool
5x5 conv, 256
11x11 conv, 96
Input
AlexNet
Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input
VGG16
Softmax
FC 1000
FC 4096
FC 4096
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 64
3x3 conv, 64
Input
VGG19

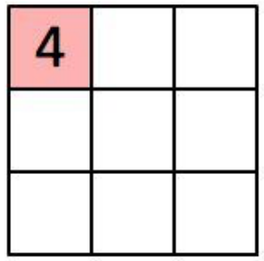
Convolution

- forward $c = Conv(a, b)$

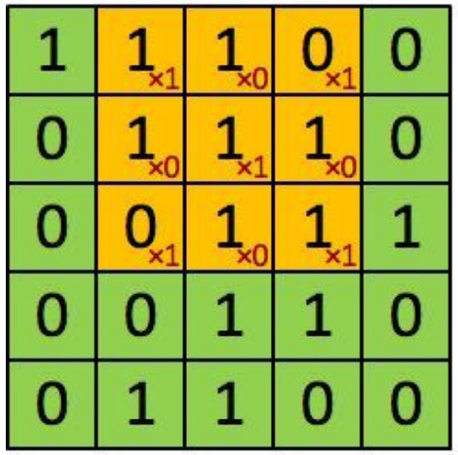
$$a = \begin{Bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{Bmatrix} \; b = \begin{Bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{Bmatrix} \rightarrow c = \begin{Bmatrix} c_1 = a_1 b_1 + a_2 b_2 + a_4 b_3 + a_5 b_4 & c_2 = a_2 b_1 + a_3 b_2 + a_5 b_3 + a_6 b_4 \\ c_3 = a_4 b_1 + a_5 b_2 + a_7 b_3 + a_8 b_4 & c_4 = a_5 b_1 + a_6 b_2 + a_8 b_3 + a_9 b_4 \end{Bmatrix}$$



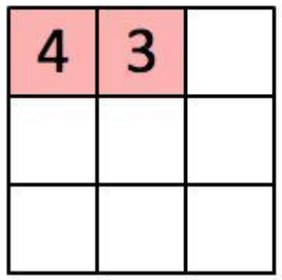
Image



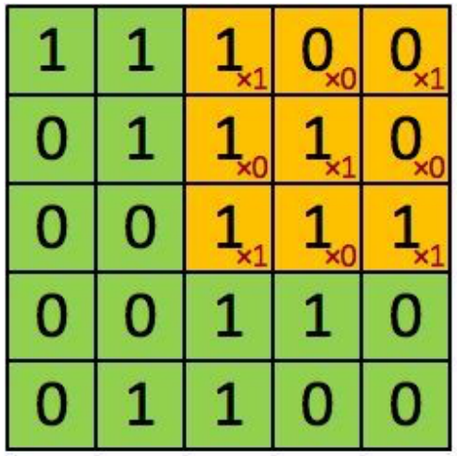
Convolved Feature



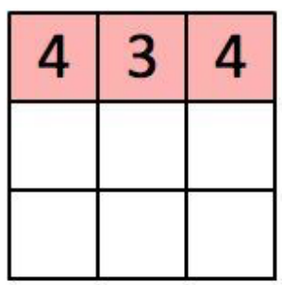
Image



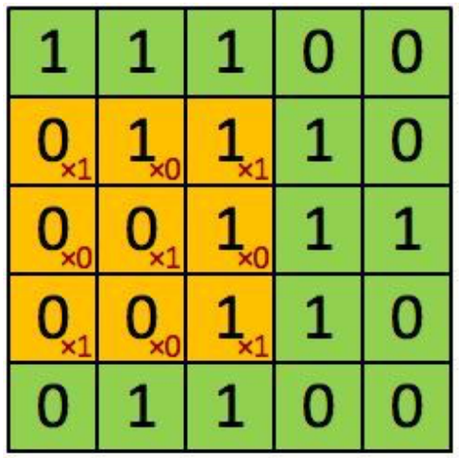
Convolved Feature



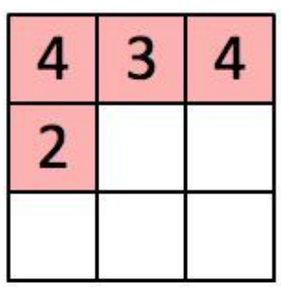
Image



Convolved Feature

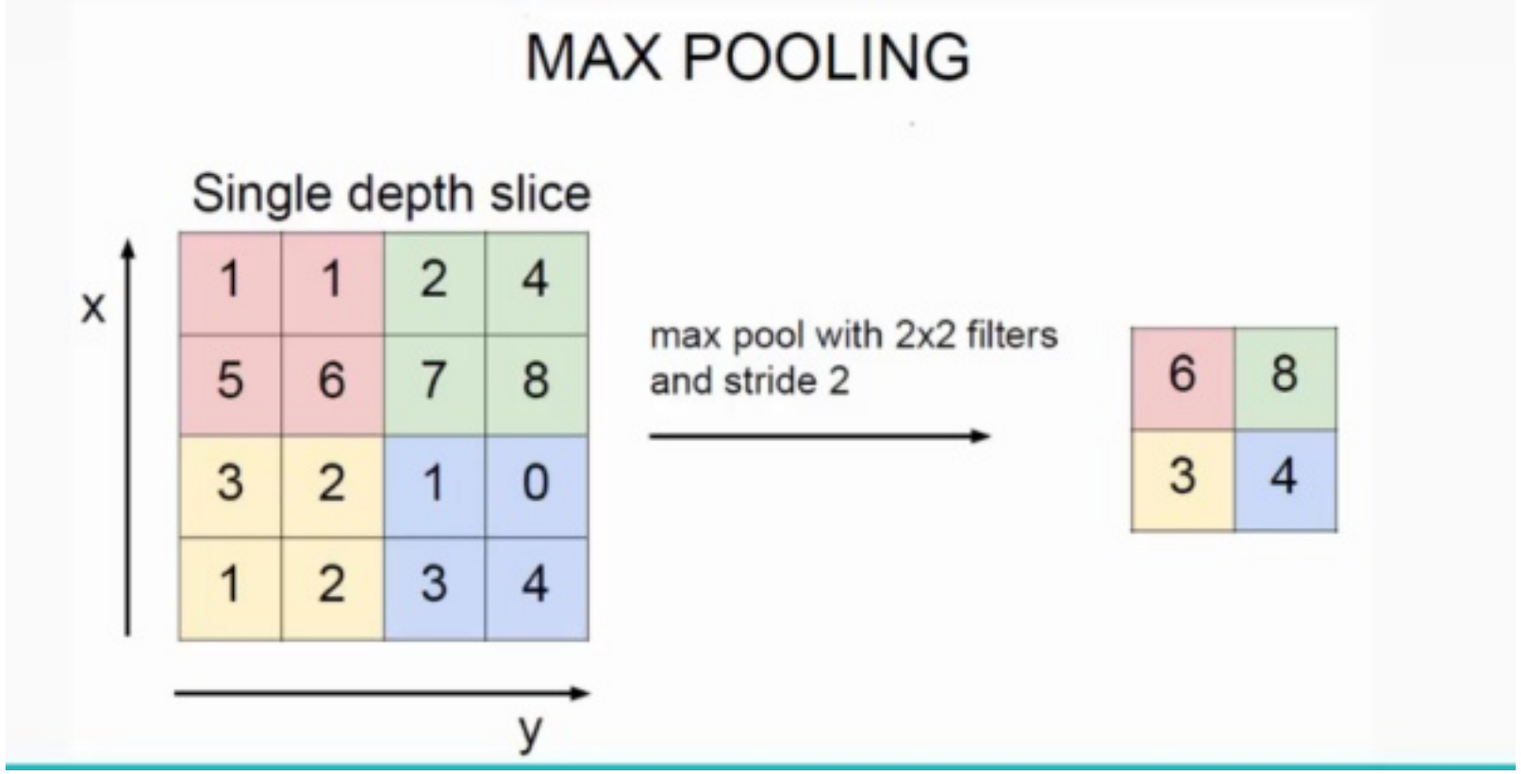


Image



Convolved Feature

Pooling



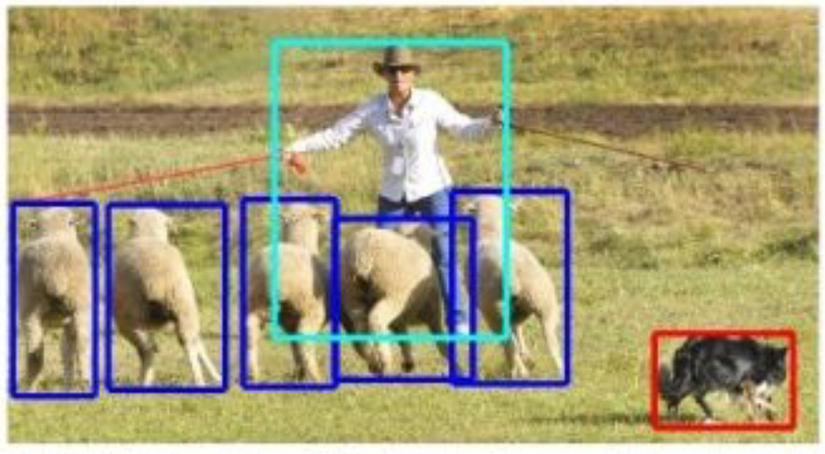
forward $c = Pooling_{max}(a)$

$$a = \begin{Bmatrix} a_1 & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 \\ a_9 & a_{10} & a_{11} & a_{12} \\ a_{13} & a_{14} & a_{15} & a_{16} \end{Bmatrix} \rightarrow c = \begin{Bmatrix} c_1 = a_2 = max\{a_1, a_2, a_5, a_6\} & c_2 = a_7 = max\{a_3, a_4, a_7, a_8\} \\ c_3 = a_9 = max\{a_9, a_{10}, a_{13}, a_{14}\} & c_4 = a_{16} = max\{a_{11}, a_{12}, a_{15}, a_{16}\} \end{Bmatrix}$$

What can CV Do?

- a图像分类；b物体检测；c语义分割（同类物体）；d实体分割（每个物体）
- 难点：形态不一致，没有明确规则

(a) Image classification



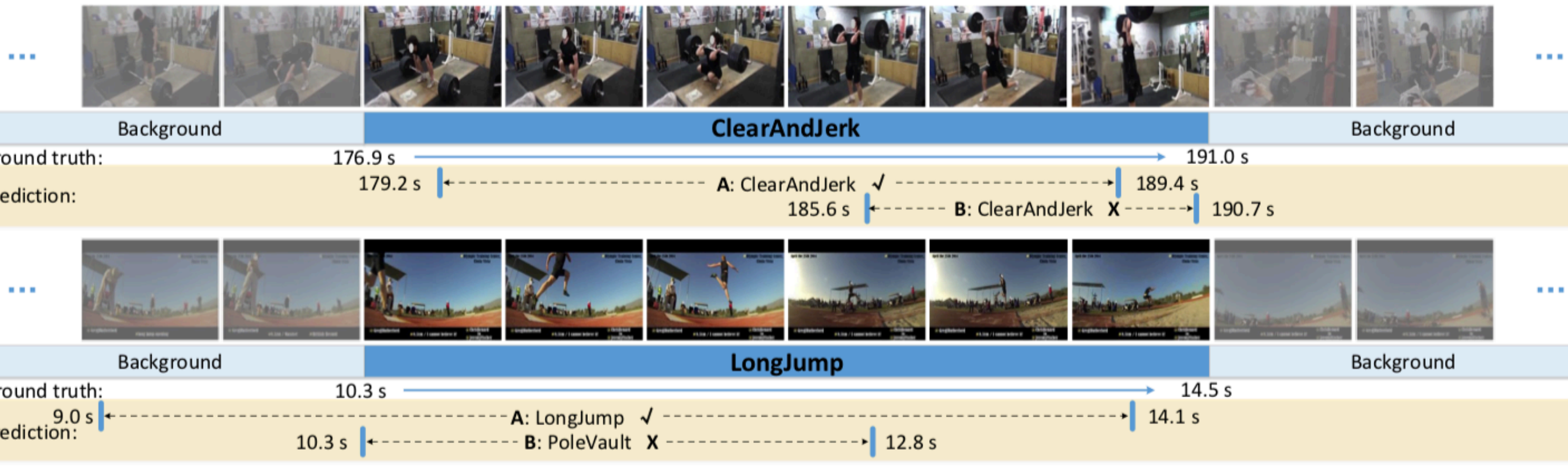
(b) Object localization

(c) Semantic segmentation

(d) This work

时序行为检测

- 从视频中一个动作序列识别出来，知道动作的属性和持续时间



What can NLP Do?

基于KB的QA

问句

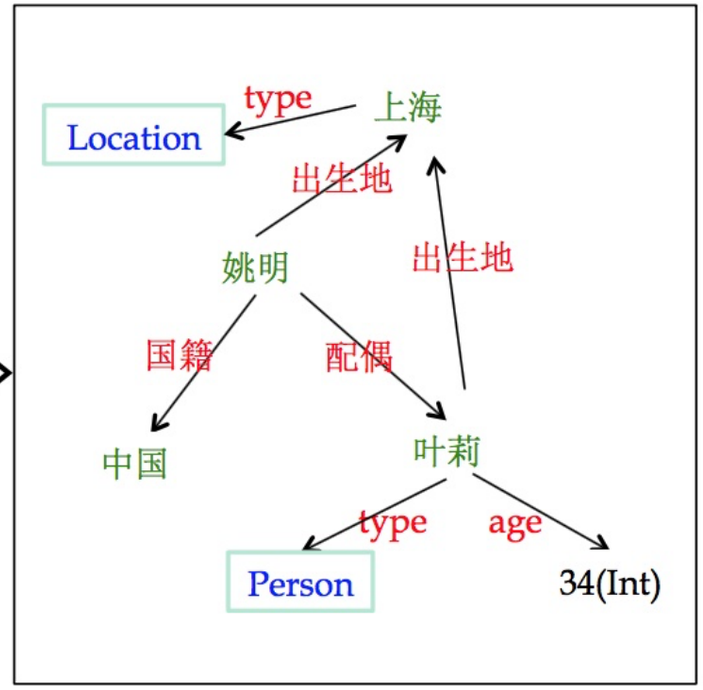
姚明的老婆的国籍是?

语义解析

语义表示

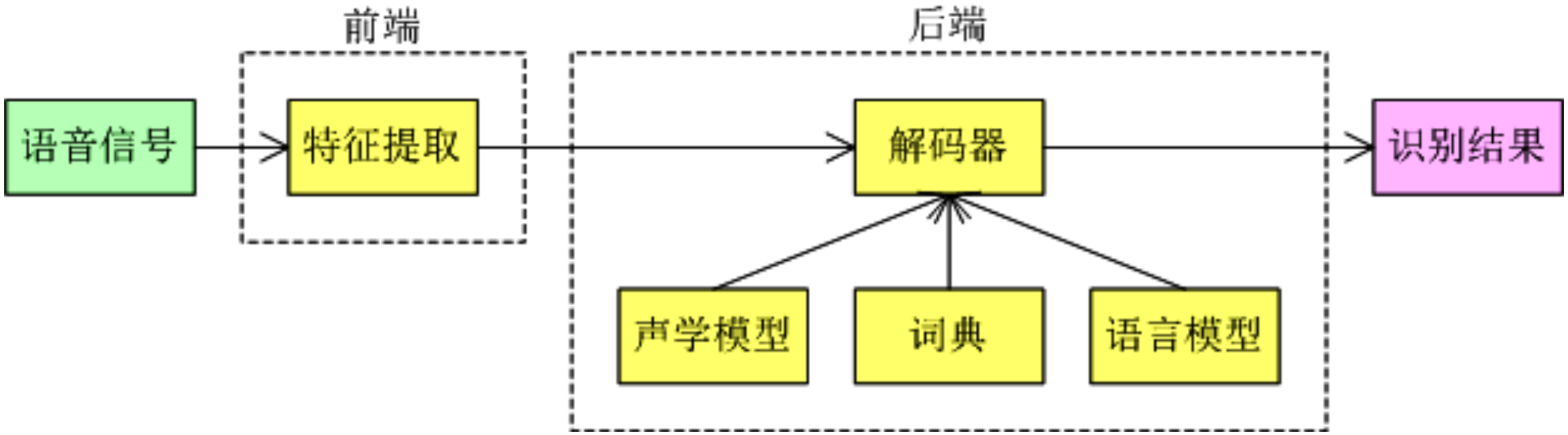
语义匹配、查询、推理

中国

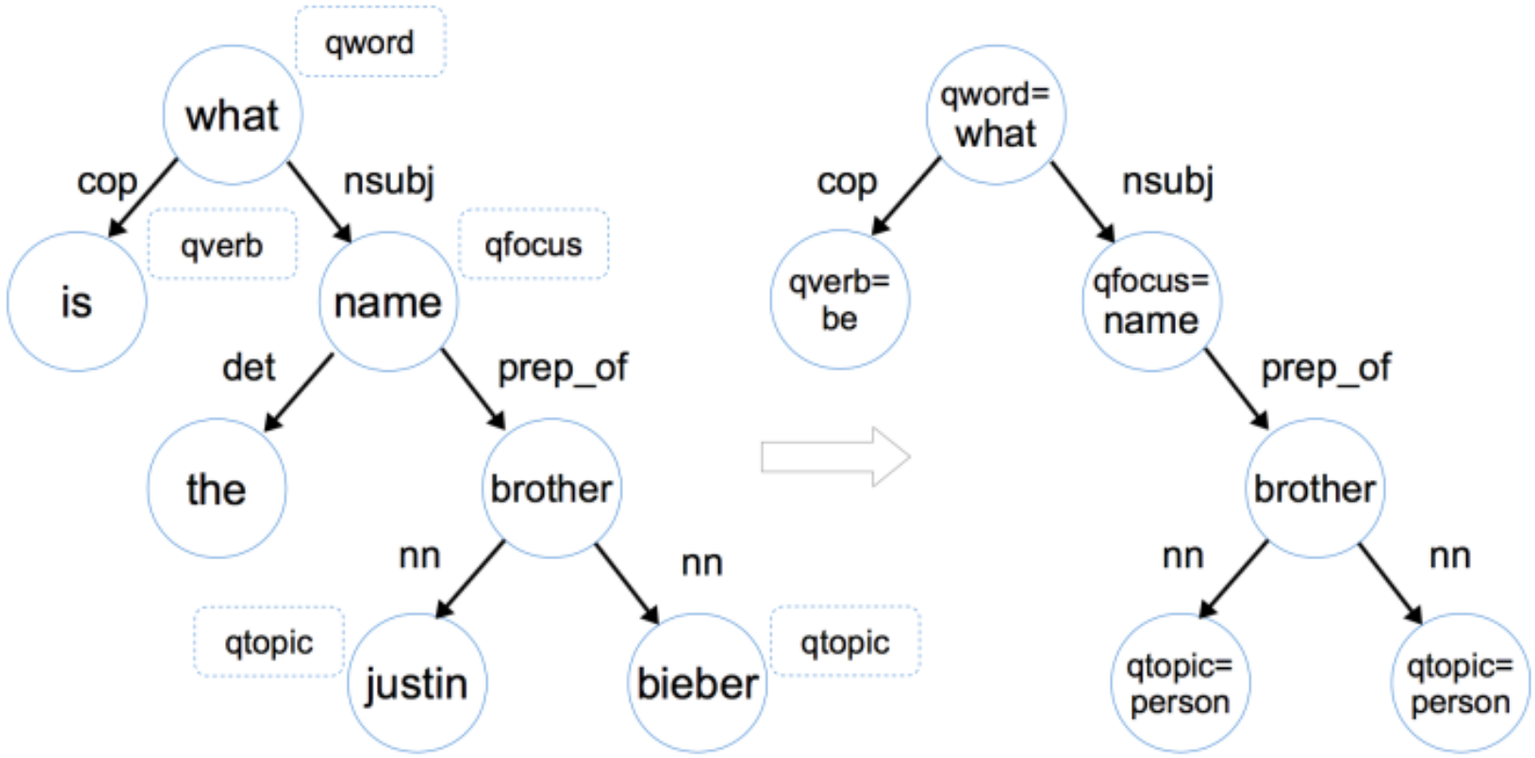


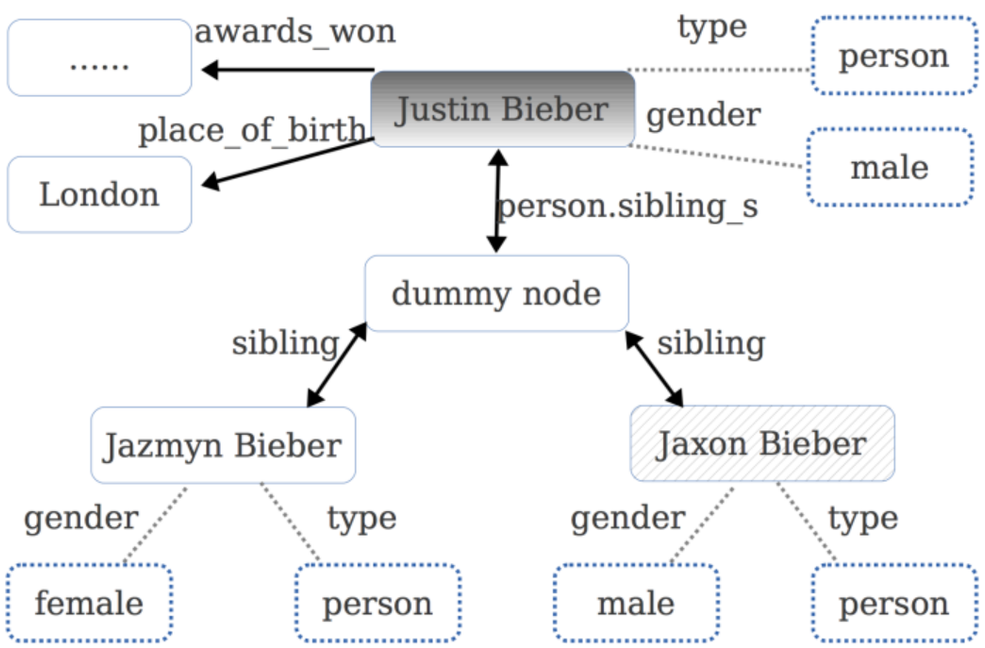
知识库

语音识别



从非结构化数据自动构建kb

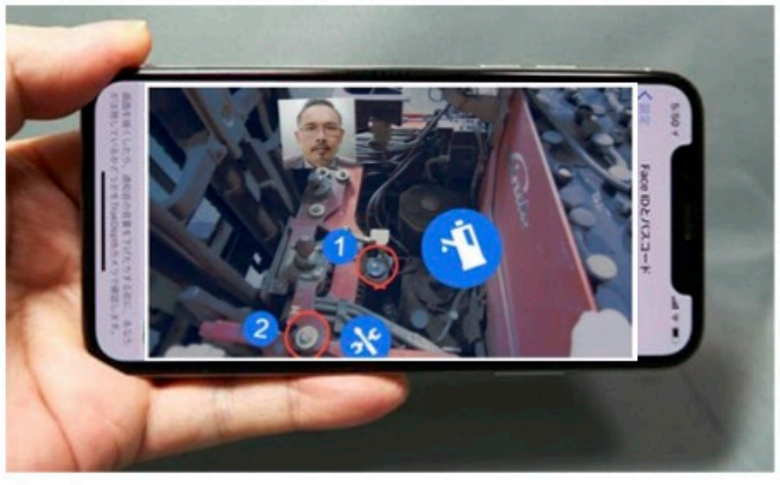




ZValley Scenarios

Scenario：远程技术支持

- 视频连线
- 自然语言交互
- 画面上面自动根据自然语言对图像语义元素标注关键信息



专家：
“第一步，向机油阀门内添加润滑油；
第二步，拧紧节气门”

语义标注

Scenario：智能客服

- 对于不同用户多次重复的问题（FAQ）自动进行分类
- 根据分类提供预先准备的标准答案
- 在用户设备上面自动标注视觉语义标签

如何给发动机换机油?
如何自己进行小保养?
发动机小保养怎么做?
换机油操作方法是什么?

用户FAQ

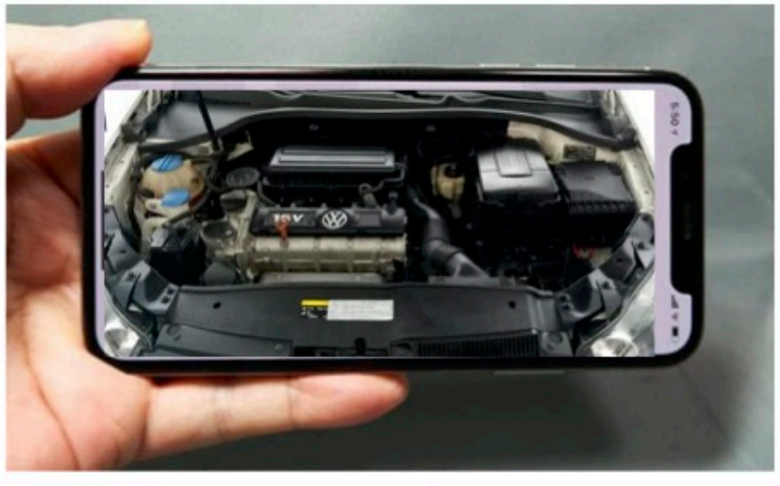
聚类

换机油

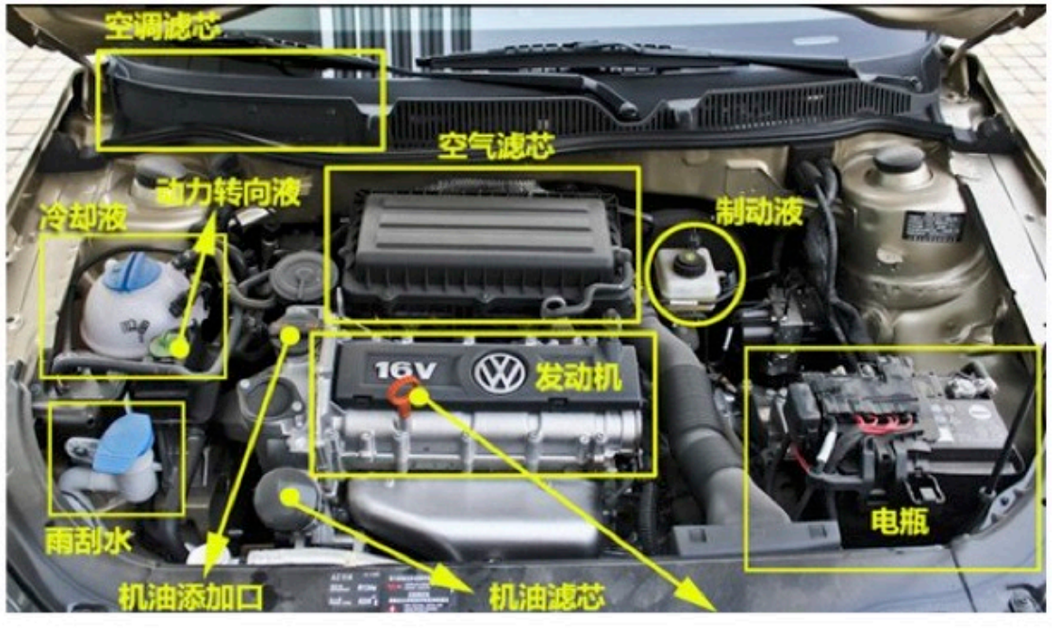
专家预先准备的解决方案

Scenario：智能用户手册

- 将电子用户手册生成知识库
- 用户用自然语言或对设备拍照对手册进行检索
- 检索结果以图像语义标注的形式显示出来



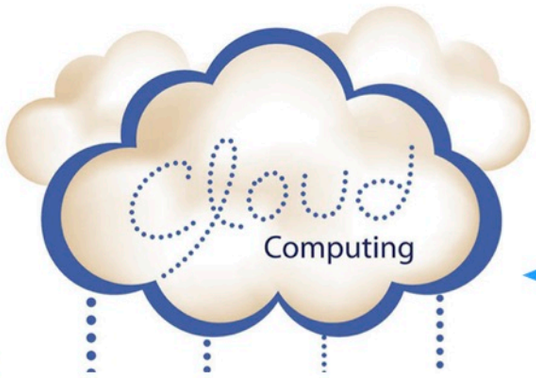
拍照查询



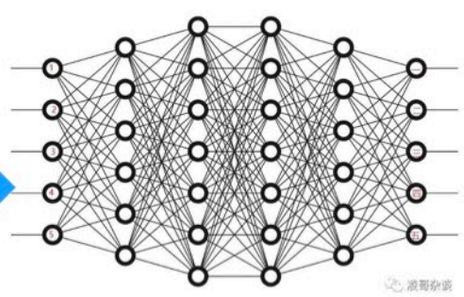
查询结果

远程绩效考核

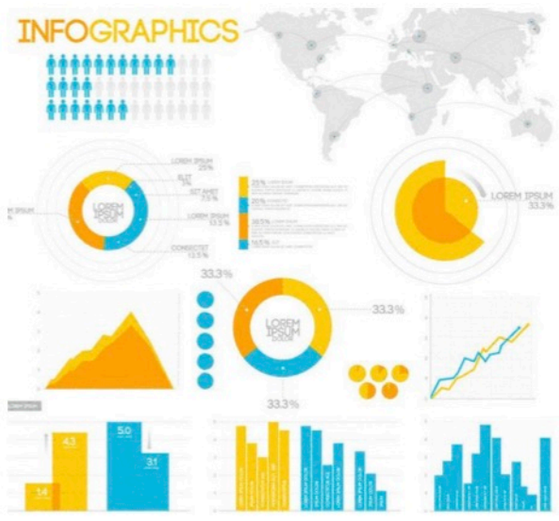
办公室管理人员



视频数据接入



AI数据分析



绩效考核结果

现场工程师

Scenario：自动化行为与流程监管系统

- 监控摄像头对生产过程进行监控，并根据知识库判断违规行为，对违规行为发出警示。

佩戴安全帽

车辆停放违规

进入禁止区域

How Are We Doing ?

- 从CVPR，ICCV/ECCV，ICLR，AAAI，NIPS等顶会上面拿学术成果下来量产

- 解决学术论文到产品化的gap

- 继续以MNIST数据为例，从时间线看深度学习学术发展

Table 1

CLASSIFIER	PREPROCESSING	TEST ERROR RATE (%)	Reference
Linear Classifiers			
linear classifier (1-layer NN)	none	12.0	LeCun et al. 1998
linear classifier (1-layer NN)	deskewing	8.4	LeCun et al. 1998
pairwise linear classifier	deskewing	7.6	LeCun et al. 1998
K-Nearest Neighbors			
K-nearest-neighbors, Euclidean (L2)	none	5.0	LeCun et al. 1998
K-nearest-neighbors, Euclidean (L2)	none	3.09	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	none	2.83	Kenneth Wilder, U. Chicago
K-nearest-neighbors, Euclidean (L2)	deskewing	2.4	LeCun et al. 1998
K-nearest-neighbors, Euclidean (L2)	deskewing, noise removal, blurring	1.80	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring	1.73	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 1 pixel shift	1.33	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 2 pixel shift	1.22	Kenneth Wilder, U. Chicago
K-NN with non-linear deformation (IDM)	shiftable edges	0.54	Keysers et al. IEEE PAMI 2007
K-NN with non-linear deformation (P2DHMDM)	shiftable edges	0.52	Keysers et al. IEEE PAMI 2007
K-NN, Tangent Distance	subsampling to 16x16 pixels	1.1	LeCun et al. 1998
K-NN, shape context matching	shape context feature extraction	0.63	Belongie et al. IEEE PAMI 2002

http://yann.lecun.com/exdb/mnist/

Table 1

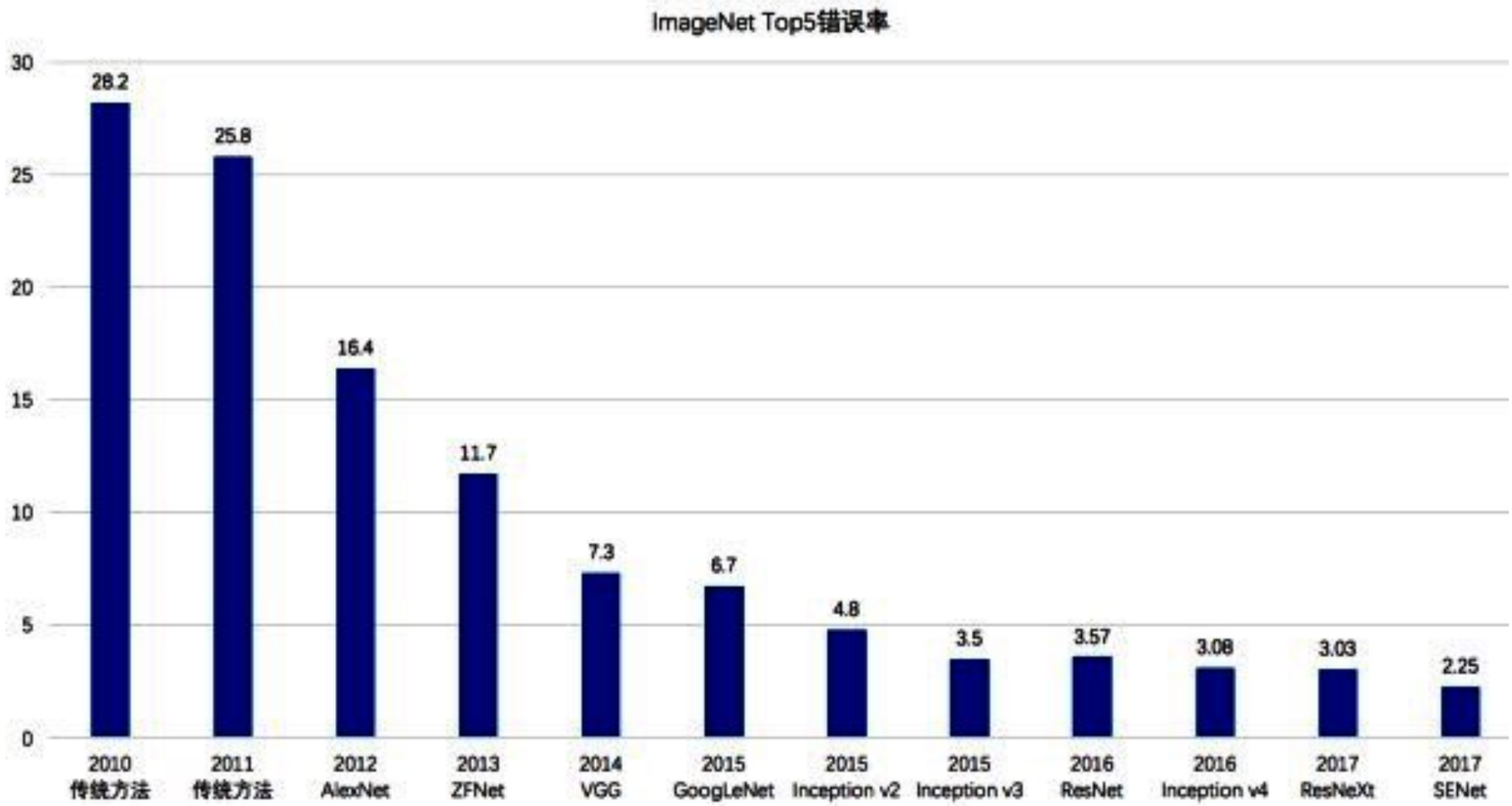
Boosted Stumps			
boosted stumps	none	7.7	Kegl et al., ICML 2009
products of boosted stumps (3 terms)	none	1.26	Kegl et al., ICML 2009
boosted trees (17 leaves)	none	1.53	Kegl et al., ICML 2009
stumps on Haar features	Haar features	1.02	Kegl et al., ICML 2009
product of stumps on Haar f.	Haar features	0.87	Kegl et al., ICML 2009
Non-Linear Classifiers			
40 PCA + quadratic classifier	none	3.3	LeCun et al. 1998
1000 RBF + linear classifier	none	3.6	LeCun et al. 1998
SVMs			
SVM, Gaussian Kernel	none	1.4	
SVM deg 4 polynomial	deskewing	1.1	LeCun et al. 1998
Reduced Set SVM deg 5 polynomial	deskewing	1.0	LeCun et al. 1998
Virtual SVM deg-9 poly [distortions]	none	0.8	LeCun et al. 1998
Virtual SVM, deg-9 poly, 1-pixel jittered	none	0.68	DeCoste and Scholkopf, MLJ 2002
Virtual SVM, deg-9 poly, 1-pixel jittered	deskewing	0.68	DeCoste and Scholkopf, MLJ 2002
Virtual SVM, deg-9 poly, 2-pixel jittered	deskewing	0.56	DeCoste and Scholkopf, MLJ 2002

Table 1

Neural Nets			
2-layer NN, 300 hidden units, mean square error	none	4.7	LeCun et al. 1998
2-layer NN, 300 HU, MSE, [distortions]	none	3.6	LeCun et al. 1998
2-layer NN, 300 HU	deskewing	1.6	LeCun et al. 1998
2-layer NN, 1000 hidden units	none	4.5	LeCun et al. 1998
2-layer NN, 1000 HU, [distortions]	none	3.8	LeCun et al. 1998
3-layer NN, 300+100 hidden units	none	3.05	LeCun et al. 1998
3-layer NN, 300+100 HU [distortions]	none	2.5	LeCun et al. 1998
3-layer NN, 500+150 hidden units	none	2.95	LeCun et al. 1998
3-layer NN, 500+150 HU [distortions]	none	2.45	LeCun et al. 1998
3-layer NN, 500+300 HU, softmax, cross entropy, weight decay	none	1.53	Hinton, unpublished, 2005
2-layer NN, 800 HU, Cross-Entropy Loss	none	1.6	Simard et al., ICDAR 2003
2-layer NN, 800 HU, cross-entropy [affine distortions]	none	1.1	Simard et al., ICDAR 2003
2-layer NN, 800 HU, MSE [elastic distortions]	none	0.9	Simard et al., ICDAR 2003
2-layer NN, 800 HU, cross-entropy [elastic distortions]	none	0.7	Simard et al., ICDAR 2003
NN, 784-500-500-2000-30 + nearest neighbor, RBM + NCA training [no distortions]	none	1.0	Salakhutdinov and Hinton, AI-Stats 2007
6-layer NN 784-2500-2000-1500-1000-500-10 (on GPU) [elastic distortions]	none	0.35	Ciresan et al. Neural Computation 10, 2010 and arXiv 1003.0358, 2010
committee of 25 NN 784-800-10 [elastic distortions]	width normalization, deslanting	0.39	Meier et al. ICDAR 2011
deep convex net, unsup pre-training [no distortions]	none	0.83	Deng et al. Interspeech 2010

Table 1-1

Convolutional nets			
Convolutional net LeNet-1	subsampling to 16x16 pixels	1.7	LeCun et al. 1998
Convolutional net LeNet-4	none	1.1	LeCun et al. 1998
Convolutional net LeNet-4 with K-NN instead of last layer	none	1.1	LeCun et al. 1998
Convolutional net LeNet-4 with local learning instead of last layer	none	1.1	LeCun et al. 1998
Convolutional net LeNet-5, [no distortions]	none	0.95	LeCun et al. 1998
Convolutional net LeNet-5, [huge distortions]	none	0.85	LeCun et al. 1998
Convolutional net LeNet-5, [distortions]	none	0.8	LeCun et al. 1998
Convolutional net Boosted LeNet-4, [distortions]	none	0.7	LeCun et al. 1998
Trainable feature extractor + SVMs [no distortions]	none	0.83	Lauer et al., Pattern Recognition 40-6, 2007
Trainable feature extractor + SVMs [elastic distortions]	none	0.56	Lauer et al., Pattern Recognition 40-6, 2007
Trainable feature extractor + SVMs [affine distortions]	none	0.54	Lauer et al., Pattern Recognition 40-6, 2007
unsupervised sparse features + SVM, [no distortions]	none	0.59	Labusch et al., IEEE TNN 2008
Convolutional net, cross-entropy [affine distortions]	none	0.6	Simard et al., ICDAR 2003
Convolutional net, cross-entropy [elastic distortions]	none	0.4	Simard et al., ICDAR 2003
large conv. net, random features [no distortions]	none	0.89	Ranzato et al., CVPR 2007
large conv. net, unsup features [no distortions]	none	0.62	Ranzato et al., CVPR 2007
large conv. net, unsup pretraining [no distortions]	none	0.60	Ranzato et al., NIPS 2006
large conv. net, unsup pretraining [elastic distortions]	none	0.39	Ranzato et al., NIPS 2006
large conv. net, unsup pretraining [no distortions]	none	0.53	Jarrett et al., ICCV 2009
large/deep conv. net, 1-20-40-60-80-100-120-120-10 [elastic distortions]	none	0.35	Ciresan et al. IJCAI 2011
committee of 7 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	width normalization	0.27 +-0.02	Ciresan et al. ICDAR 2011
committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	width normalization	0.23	Ciresan et al. CVPR 2012

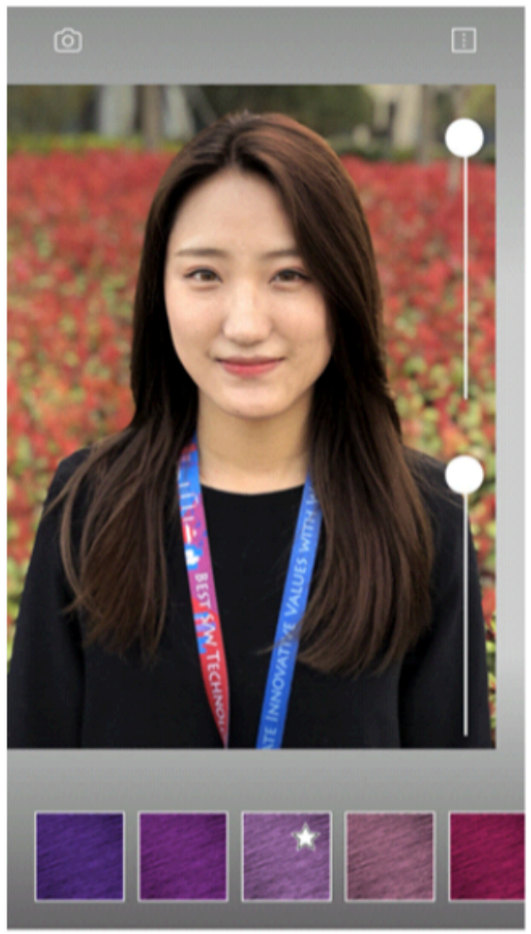


云端网络

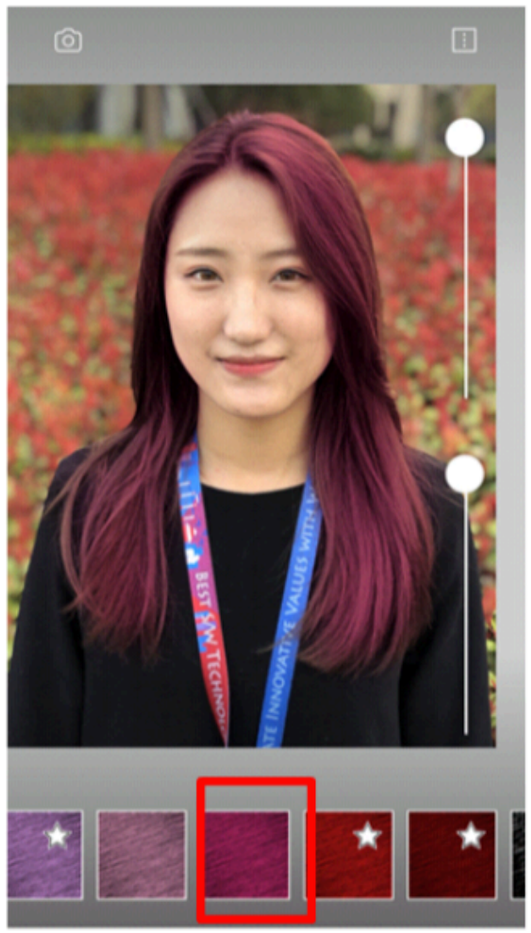
	最早公开日期	发表情况	作者团队	arXiv链接
SqueezeNet	2016.02	ICLR-2017	伯克利&斯坦福	https://arxiv.org/abs/1602.07360
MobileNet	2016.04	CVPR-2017	Google	https://arxiv.org/abs/1704.04861
ShuffleNet	2016.06	CVPR-2017	Face++	https://arxiv.org/abs/1707.01083
Xception	2016.10	N/A	Google	https://arxiv.org/abs/1610.02357

嵌入式网络

Close Gap



▪ Take a picture by phone camera



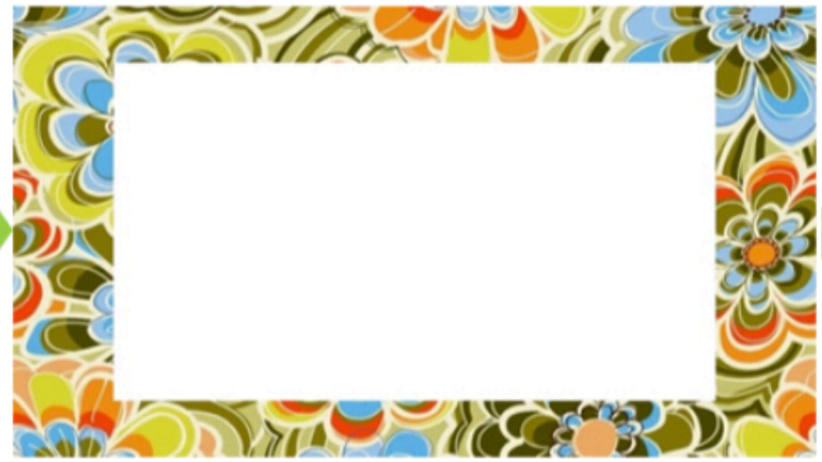
▪Try different color

▪Get recolored image and share in SNS

- 标注cost太高 （学术研究没有针对头发的数据需要重新做）
- 头发细节太多，精度要求高（学术上的模型做不到）
- 要求在三星s9上面实时渲染（学术上做不到）

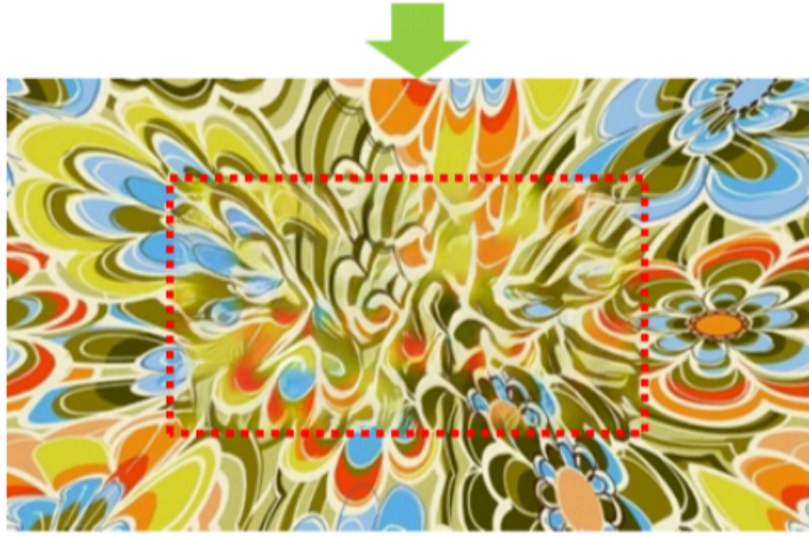
Close Gap

(1)Input 3840*2160



(2)Downscale to 576*324

(3)Warping Forward



(4)Inpainting

(5)Warping Back 576*324

(6)SR to 3840*2160

- 学术上的模型最多做到25%缺失的补全，项目要求50%
- 学术上的模型分辨率最多做到500*500(100块显卡)，项目要求3840*2160
- 项目要求在手机上面2s渲染出来

2019 Work Overview

- 推进云端AI技术落地

  视频内容理解的相关场景

  知识库和对话系统的相关场景

- 推进edge端（手机、眼镜）AI技术落地

  cloud端做模型训练，并做大数据分析，向管理决策服务

  edge端做模型预测，向辅助现场工程师提高工作效率

Conclusion

领导的期许

- “高级人才” == “能做出高级事情的人”
- “高级人才” != “拿到博士学位的人”
- “高级人才” == “hard-core”

- “高级人次” == “项目管理” && “软件工程” && “新技术量产” && “工程师培训” && “架构设计” && “算法工程师” && “写ppt” && “解bug” && “…”
- Conclusion: “高级人才” == “知识担当” && “创新技术leader” && “多面手”

Appendix

Introduction to Machine Learning

Huang Yuefeng

January 11, 2019

Guideline

Introduction

Logistic Regression

Softmax Regression

Unconstrained Optimization

Gradient Decent Tricks

Multiclass SVM

Constrained Optimization: Lagrange Duality

Neural Network

Deep Learning for NLP

Deep Learning for Image Processing

Deep Learning Tricks

Maximum Entropy

Lagrange Dual Problem

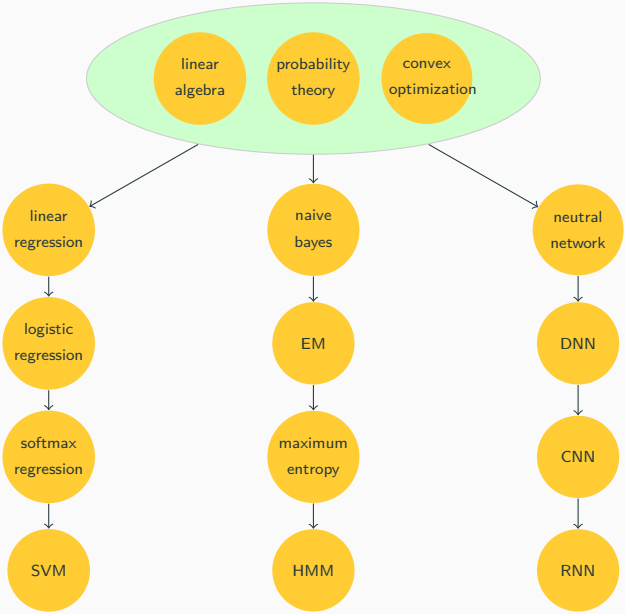
Optimization Algorithm

Introduction

Introduction of Machine Learning

- hypothesis function: $h_\theta : X \to Y$
- loss function: $L : Y, h_\theta(X) \to R$, where R is real number.
- log loss function: $l : log(L)$
- training set: a list of m training example $\{x^{(i)}, y^{(i)}\}$, $y^{(i)}$ called label of $x^{(i)}$
- machine learning = training + predicting
- training: try to get θ at minimum value of L by using $\{x^{(i)}, y^{(i)}\}$ and Convex Optimization method
- predicting: use h with θ to predict y of unknown x

Core Knowledge



Logistic Regression

Logistic Regression

- hypothesis function: $h_\theta(x) = P(y = 1|x, \theta) = \frac{1}{1+e^{-\theta^T x}}$
- $g(z) = \frac{1}{1+e^{-z}}$ is logistic function or sigmoid function
  $g'(z) = ((1 + e^{-z})^{-1})'(e^{-z})' = ((1 + e^{-z})^{-2})(e^{-z})$
  $= \frac{1}{1+e^{-z}}(1 - \frac{1}{1+e^{-z}}) = g(z)(1 - g(z))$
  $(f'(g(x)) = f'(x)g'(x))$
  $g(z)$ is calculated with low cost
  $h_\theta(x) = g(\theta^T x)$
- $p(y|x, \theta) = P(y = 1|x, \theta)^y P(y = 0|x, \theta)^{1-y} = (h_\theta(x))^y(1 - (h_\theta(x))^{1-y}$
- loss function (actually it is already not loss function, but target function) by Maximum Likelihood:
  $L(\theta) = \prod_{i=1}^m p(y^{(i)}|x^{(i)}, \theta) = \prod_{i=1}^m (h_\theta(x^{(i)}))^{y^{(i)}}(1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$
- log loss function:
  $l(\theta) = \log L(\theta) = \sum_{i=1}^m (y^{(i)} \log h_\theta(x) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})))$

Derivatives

- $\frac{\partial}{\theta_j} l(\theta)$

$$
\begin{aligned}
&= \tfrac{\partial}{\theta_j} \textstyle\sum_{i=1}^{m} (y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \\
&= \tfrac{\partial}{\theta_j} \textstyle\sum_{i=1}^{m} (y^{(i)} \log g(\theta^T x^{(i)}) + (1 - y^{(i)}) \log(1 - g(\theta^T x^{(i)}))) \\
&= \textstyle\sum_{i=1}^{m} (y^{(i)} \frac{1}{g(\theta^T x^{(i)})} + (1 - y^{(i)}) \frac{1}{(1 - g(\theta^T x^{(i)}))}) \tfrac{\partial}{\theta_j} g(\theta^T x^{(i)}) \\
&= \textstyle\sum_{i=1}^{m} (y^{(i)} \frac{1}{g(\theta^T x^{(i)})} + (1 - y^{(i)}) \frac{1}{(1 - g(\theta^T x^{(i)}))}) g(\theta^T x^{(i)})(1 - g(\theta^T x^{(i)})) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \tfrac{\partial}{\partial \theta_j} (\theta^T x^{(i)}) \\
&= \textstyle\sum_{i=1}^{m} (y^{(i)} (1 - g(\theta^T x^{(i)}) - (1 - y^{(i)}) g(\theta^T x^{(i)})) \tfrac{\partial}{\theta_j} g(\theta^T x^{(i)}) \\
&= \textstyle\sum_{i=1}^{m} (y^{(i)} (1 - g(\theta^T x^{(i)}) - (1 - y^{(i)}) g(\theta^T x^{(i)})) x^{(j)} \\
&= \textstyle\sum_{i=1}^{m} (y^{(i)} - g(\theta^T x^{(i)})) x^{(j)} \\
&= \textstyle\sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)})) x^{(j)} \\
&= x^{(j)} \textstyle\sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)}))
\end{aligned}
$$

Softmax Regression

Softmax Regression

- hypothesis function: $h_\theta(x^{(i)}) = [p(y^{(i)} = j)|x^{(i)}, \theta] = [\frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}]$
- predict function: $p(y^{(i)} = j|x^{(i)}, \theta) = \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}$
- loss function

  $L(\theta) = -\frac{1}{m} \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}, \theta) = \prod_{i=1}^{m} \prod_{j=1}^{k} (\frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}})^{1\{y_j=1\}}$
- log loss function:

  $l(\theta) = -\frac{1}{m} \log L(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{k} \log(\frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}})^{1\{y_j=1\}}$

  $= -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{k} 1\{y_j = 1\} \log(p(y^{(i)} = j|x^{(i)}, \theta))$

  $\nabla_{\theta_j} l(\theta) = -\frac{1}{m} \sum_{i=1}^{m} 1\{y_j = 1\} \nabla_{\theta_j} (\log \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}})$

Derivatives

- $\nabla_{\theta_j}(\log \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}})$
  $= \nabla_{\theta_j}(\log e^{\theta_j^T x^{(i)}} - \log \sum_{l=1}^{k} e^{\theta_l^T x^{(i)}})$
  $= \frac{1}{e^{\theta_j^T x^{(i)}}} * e^{\theta_j^T x^{(i)}} * x^{(i)} - \frac{1}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}} * e^{\theta_j^T x^{(i)}} * x^{(i)} = (1 - p((y^{(i)} = j)|x^{(i)}))x^{(i)}$
- $\nabla_{\theta_j} l(\theta) = -\frac{1}{m} \sum_{i=1}^{m} 1\{y_j = 1\}(-1 + p((y^{(i)} = j)|x^{(i)}))x^{(i)}$

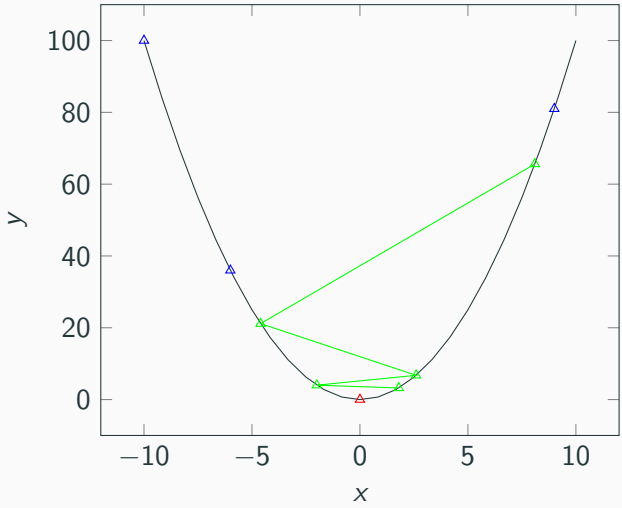
Unconstrained Optimization

Unconstrained Optimization

- General Decent Method
  - Random Local Search
- First Order Method
  - Batch Gradient Decent
  - Mini Batch Gradient Decent
  - Stochastic Gradient Decent (SGD)(On-line Gradient Decent)
- Second Order Method
  - Newton's Method
  - Quasi-Newton's Method
    - L-BFGS
- Online Gradient Decent

General Decent Method: Random Local Search

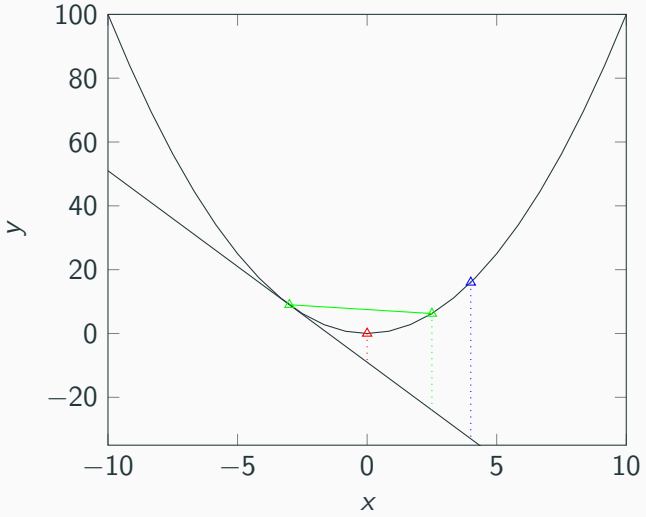
- question definition: $y = f(x)$, find the minimum value of y.
- random change x to find a better y until y is converged.
- search direction: Δx is random
- search step: t is a small value
- update: $x^{(k+1)} = x^{(k)} + t\Delta x$
- finish: $\Delta y < tolerance$
- issue: forbid to get fake minimum y by small $t\Delta x$


100
80
60
40
20
0
y
−10
−5
0
5
10
x

First Order Method: Gradient Decent

- motivation:change x in the direction of gradient, which is fast to minimum.

  Assume a function can be expressed as a converged summary of a series of power series, in the adjacent domain of x^0, it is:
  $f(x) = f(x^0) + \frac{f^{(1)}(x^0)}{1!}(x - x^0) + \frac{f^{(2)}(x^0)}{2!}(x - x^0)^2 + ... + \frac{f^{(n)}(x^0)}{n!}(x - x^0)^n$
  use $f(x) \approx f(x^0) + \frac{f^{(1)}(x^0)}{1!}(x - x^0)$ to get search direction, i.e. first order.
- search direction: $\Delta x = f^{(1)}(x^0) = \nabla f(x^0)$
- search step: t is a small value
- update: $x^{(k+1)} = x^{(k)} + t\nabla f(x^k)$
- finish: $\nabla f(x^k) < tolerance$
- issue: how to choose t, please see SGD tricks.



$f(x) = x^2$ and $y - y^0 = f'(x - x^0)$

First Order Method: Batch Gradient Decent

- Problem Definition: Given $J_\theta = f(y, y^{(i)}), y = g(\theta^T x)$ and a training set of $(x^{(i)}, y^{(i)})$, try to find θ^* by minimizing J_θ
- The problem is simplified to

  Minimizeing $J_\theta = F(\theta, x^{(i)}, y^{(i)})$ for any known $(x^{(i)}, y^{(i)})$
- If find θ by $\nabla J_\theta = 0$ we can resolve the problem. But it is difficult to find analytical θ, so we use iterative algorithm to find $\nabla J_\theta \approx 0$.
- The solution is to calculate $\nabla J_\theta = k(\theta, x^{(i)}, y^{(i)})$ until it is converged.
- From previous discussion on Gradient Decent,

  update: $\theta^{(k+1)} = \theta^{(k)} + t\nabla J_\theta(\theta^k)$

  finish: $\nabla J_\theta(\theta^k) < tolerance$
- Because there are multiple $(x^{(i)}, y^{(i)})$, we should use the average ∇ for better precision. This is Batch Gradient Decent.

First Order Method: Stochastic Gradient Decent

- the batch stochastic gradient is summarized as:

  update: $\theta^{(k+1)} = \theta^{(k)} + t\frac{1}{n}\sum_i^n k(\theta, x^{(i)}, y^{(i)})$

  finish: $\frac{1}{n}\sum_i^n k(\theta, x^{(i)}, y^{(i)}) < tolerance$

- t and *tolerance* is human chosen small value, so the above can be simplified as:

  update: $\theta^{(k+1)} = \theta^{(k)} + t\sum_i^n k(\theta, x^{(i)}, y^{(i)})$

  finish: $\sum_i^n k(\theta, x^{(i)}, y^{(i)}) < tolerance$

- If we choose any $(x^{(i)}, y^{(i)})$ as the representation of the average, we get Stochastic Gradient Decent:

  update: $\theta^{(k+1)} = \theta^{(k)} + tk(\theta, x^{(i)}, y^{(i)})$

  finish: $|k(\theta, x^{(i)}, y^{(i)})| < tolerance$ or $|y^{(k+1)} - y^{(k)}| < tolerance$

- issue: there may be a random $(x^{(i)}, y^{(i)})$, which makes the algorithm finish too early.

First Order Method: Mini Batch Gradient Decent

- instead of using stochastic one $(x^{(i)}, y^{(i)})$, but use a batch of the, we get Mini Batch Gradient Decent

  update: $\theta^{(k+1)} = \theta^{(k)} + t\frac{1}{m}\sum_i^m k(\theta, x^{(i)}, y^{(i)})$

  finish: $\frac{1}{m}\sum_i^m k(\theta, x^{(i)}, y^{(i)}) < tolerance$

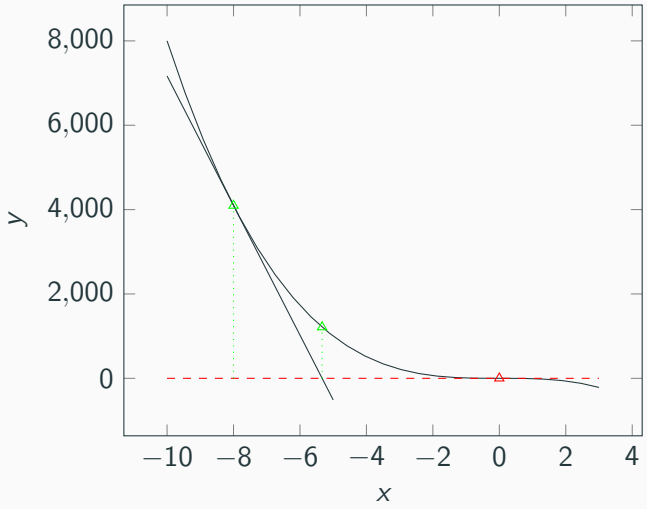
  $m << n$

- Stochastic Gradient Decent and Mini Batch Gradient Decent are the approximation of Batch Gradient Decent. They can be converged as Batch Gradient Decent. The proof of converge is referred in paper.

Second Order Method: Newton's Method

- motivation:change x in the direction of minimum of gradient.

  Assume a function can be expressed as a converged summary of a series of power series, in the adjacent domain of x^0, it is:
  $f(x) = f(x^0) + \frac{f^{(1)}(x^0)}{1!}(x - x^0) + \frac{f^{(2)}(x^0)}{2!}(x - x^0)^2 + ... + \frac{f^{(n)}(x^0)}{n!}(x - x^0)^n$
  use $f(x) \approx f(x^0) + \frac{f^{(1)}(x^0)}{1!}(x - x^0) + \frac{f^{(2)}(x^0)}{2!}(x - x^0)^2$ to get search direction, i.e. second order.
- search direction: $\Delta x = \frac{f^{(1)}(x^0)}{f^{(2)}(x^0)}$ (minimum quadratic function with $-\frac{b}{2a}$)
- search step: t is a small value
- update: $x^{(k+1)} = x^{(k)} + t\frac{f^{(1)}(x^0)}{f^{(2)}(x^0)}$
- finish: $|y^{(k+1)} - y^{(k)}| < tolerance$
- issue: how to choose t, please see SGD tricks.



$f'(x)$ and $f''(x)$

$$x^{(k+1)} = x^{(k)} + \frac{f'(x^{(k)})}{f''(x^{(k)})}$$

Gradient Decent Tricks

Gradient Decent Tricks

- Update Tricks
  - Vanilla Update
  - Momentum update
  - Nesterov's Accelerated Momentum (NAG)
  - Nesterov Momentum
- Learning Rate Update Tricks
  - Annealing the learning rate
  - Adam
  - Adagrad
  - RMSprop
  - Adadelta
- Learning Rate Choose Tricks
  - Line Search
  - Bisection Line Search
  - Backtracking on Armijo
  - Backtracking on Wolfe

Multiclass SVM

Linear Separable Binary SVM

- hypothesis function: $h_\theta(x^{(i)}) = \theta^T x^{(i)}$
- predict function:

$$y^{(i)} = \begin{cases} 1, if(h_\theta(x^{(i)}) > d_1) \\ -1, if(h_\theta(x^{(i)}) < -d_2) \end{cases}$$

- loss function: $L(\theta) = ||\theta||^2$ subject to $y^{(i)}\theta^T x^{(i)} \geq \min(d_1, d_2)$
  (if two points $x^{(a)}$ and $x^{(b)}$ nearest to the line, loss function is geometric margin: $\frac{|\theta^T x^{(a)}|}{||\theta||} + \frac{|\theta^T x^{(b)}|}{||\theta||} = \frac{d_1+d_2}{||\theta||}$, $d_1 + d_2$ is constant, so loss function is simplified to $\max(\frac{1}{||\theta||})$, the same to $||\theta||^2$.)
- we can make $d_1 = d_2 = d$, and replace θ with $\frac{\theta}{d}$, the loss function is not changed, and predict function is simplified to:

$$y^{(i)} = \begin{cases} 1, if(h_\theta(x^{(i)}) > 1) \\ -1, if(h_\theta(x^{(i)}) < -1) \end{cases}$$

- optimization: This is not unconstrained optimization, can't be solved by GD, but solved by Lagrange duality and SMO.

Dual Problem Optimization

- add Lagrange factor to construct function:
  $L(\theta, \alpha) = \frac{1}{2}||\theta||^2 - \sum_{i=1}^{m} \alpha_i(y^{(i)}\theta^T x^{(i)} - 1)$, subject to $\alpha_i \geq 0$
- Primal problem is: $\min_\theta \max_\alpha L(\theta, \alpha)$
- Dual problem is:$\max_\alpha \min_\theta L(\theta, \alpha)$
- to resolve dual problem, first resolve $\min_\theta L(\theta, \alpha)$ by the following way:
  $\nabla_\theta L(\theta, \alpha) = \theta - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0$ so, we get $\theta = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$
- The dual problem is to be:
  $\max_\alpha W(\alpha) = \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} {x^{(i)}}^T x^{(j)} - \sum_{i,j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} {x^{(i)}}^T x^{(j)} + \sum_{i=1}^{m} \alpha_i$
  $= \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} {x^{(i)}}^T x^{(j)}$, subject to $\alpha_i \geq 0$.
- This can be resolved by GD.

Nonlinear Separable Binary SVM: Kernel Trick

- The dual problem can be written into inner product manner:
  $\max_\alpha W(\alpha) = \sum_{i=1}^m \alpha - \frac{1}{2}\sum_{i,j=1}^m \alpha_i\alpha_j y^{(i)}y^{(j)}{x^{(i)}}^T x^{(j)}$, subject to $\alpha_i \geq 0$.
  $= \sum_{i=1}^m \alpha - \frac{1}{2}\sum_{i,j=1}^m \alpha_i\alpha_j y^{(i)}y^{(j)} < x^{(i)}, x^{(j)} >$, subject to $\alpha_i \geq 0$.
- hypothesis function:$h_\theta(x^{(i)}) = \theta^T x^{(i)}$ can't separate training samples, i.e. nonlinear separable, we can consider more complex relationship between y and x. Such as:
  $h_\theta(x^{(i)}) = \sum_{a,b=0}^n \theta_{a,b} x_a^{(i)} x_b^{(i)} = \theta^T x_{trans}^{(i)}$, where n is dimension of X.
- in loss function: $< x_{trans}^{(i)}, x_{trans}^{(j)} >= \sum_{a,b=0}^n (x_a^{(i)})^2 (x_b^{(i)})^2 = ((x^{(i)})^T (x^{(i)}))^2$
- $< x_{trans}^{(i)}, x_{trans}^{(j)} >$ is calculated in time cost of $O(n)$ instead of $O(n^2)$.
- This is defined as polynomial kernel: $< x_{trans}^{(i)}, x_{trans}^{(j)} >= ((x^{(i)})^T (x^{(j)}) + c)^d$, which mapping original linear y on x to polynomial y on x.

Nonlinear Separable Binary SVM: Kernel Trick

- other kernel function:
- Gaussian Kernel (radial basis function)
  $< x^{(i)}_{trans}, x^{(j)}_{trans} >= \exp - \frac{||x^{(i)} - x^{(j)}||^2}{2\sigma^2}$
  which map to unlimited polynomial y on x by the idea of following function:
  Tylor expansion $e^x = 1 + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + ..., -\infty < x < \infty$
- sigmoid: $< x^{(i)}_{trans}, x^{(j)}_{trans} >= \tanh((x^{(i)})^T x^{(j)} + c)$

Nonlinear Separable Binary SVM: Soft Margin

- hypothesis function: $h_\theta(x^{(i)}) = \theta^T x^{(i)}$
- predict function:

$$y^{(i)} = \begin{cases} 1, if(h_\theta(x^{(i)}) > 1) \\ -1, if(h_\theta(x^{(i)}) < -1) \end{cases}$$

- loss function: $l(\theta) = \lambda||\theta||^2 + \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y^{(i)}\theta^t x^{(i)})$
- derivatives:

$$\nabla_{\theta_j} L = \begin{cases} 1(1 - y^{(i)}\theta^t x^{(i)} > 0)x^{(i)} \\ 1(1 - y^{(i)}\theta^t x^{(i)} < 0)0 \end{cases}$$

- sometimes, to enlarge the margin we have the following loss function:
  $l(\theta) = \lambda||\theta||^2 + \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y^{(i)}\theta^t x^{(i)} + \Delta)$, where $\Delta > 0$

Multiclass SVM

- binary classification: $L_i = max(0, 1 - y^{(i)}\theta^T x^{(i)}) = max(0, 1 - m_i)$
  loss function: $L = L_{data} + \lambda L_{norm} = \frac{1}{m}\sum_{i=1}^{m} max(0, 1 - y^{(i)}\theta^T x^{(i)}) + \lambda L_{norm}$
- multiple classification: $L_i = \sum_{j \neq y^{(i)}}^{k} max(0, \theta_j^T x^{(i)} - \theta_{y^{(i)}}^T x^{(i)} + \Delta)$
  or $L_i = max(0, max_{j \neq y^{(i)}}^{k}(\theta_j^T x^{(i)}) - \theta_{y^{(i)}}^T x^{(i)} + 1)$
- derivative

$$\nabla_{\theta_j} L = \begin{cases} 1(\theta_j^T x^{(i)} - \theta_{y^{(i)}}^T x^{(i)} + \Delta > 0) x^{(i)}, if(j \neq y^{(i)}) \\ -(\sum_{j \neq y^{(i)}}^{k} 1(\theta_j^T x^{(i)} - \theta_{y^{(i)}}^T x^{(i)} + \Delta > 0)) x^{(i)}, if(j = y^{(i)}) \end{cases} \tag{1}$$

$$\nabla_{\theta_j} L = \begin{cases} 1(\theta_j^T x^{(i)} - \theta_{y^{(i)}}^T x^{(i)} + 1 > 0) x^{(i)}, if(j \neq y^{(i)} and \theta_j^T x^{(i)} ismax) \\ 0, if(j \neq y^{(i)} and \theta_j^T x^{(i)} isnotmax) \\ -(\sum_{j \neq y^{(i)}}^{k} 1(\theta_j^T x^{(i)} - \theta_{y^{(i)}}^T x^{(i)} + 1 > 0)) x^{(i)}, if(j = y^{(i)}) \end{cases} \tag{2}$$

Constrained Optimization: Lagrange Duality

Constrained Optimization: Lagrange Duality

- constrained optimization:

$$\begin{cases} \min\limits_{\omega} f(\omega) \\ s.t. g_i(\omega) \leq 0, i = 1, 2, ..., k \\ \quad h_i(\omega) = 0, i = 1, 2, ..., l \end{cases}$$

- to resolve original problem, we construct the following function:

$$\begin{cases} L(\omega, \alpha, \beta) = f(\omega) + \sum\limits_{i=1}^{k} \alpha_i g_i(\omega) + \sum\limits_{j=1}^{l} \beta_j h_j(\omega) \\ \qquad\qquad\qquad\qquad\qquad\qquad s.t. \alpha_i \geq 0. \end{cases}$$

- to resolve original problem, we construct primal problem on function L:
  $\min_{\omega} \max_{\alpha,\beta} L(\omega, \alpha, \beta)$
- and the dual problem on function L:
  $\max_{\alpha,\beta} \min_{\omega} L(\omega, \alpha, \beta)$
- optimization: the dual problem is unconstrained optimization to be resolved by GD, or at least it is easier to be resolved than original problem.

- We will convert constrained optimization $\min_\omega f(\omega)$ to unconstrained optimization $\max_{\alpha,\beta} \min_\omega L(\omega, \alpha, \beta)$,
  which is easier resolved by GD, we will prove:

$$\max_{\alpha,\beta} \min_{\omega} L(\omega, \alpha, \beta) \leq \min_{\omega} \max_{\alpha,\beta} L(\omega, \alpha, \beta) \leq \min_{\omega} f(\omega)$$

  when the following condition (KKT condition) is held, they are equal to each other:

$$\begin{cases} \alpha_i g_i(\omega) = 0, i = 1, 2, ..., k \\ g_i(\omega) \leq 0, i = 1, 2, ..., k \\ \alpha_i \geq 0, i = 1, 2, ..., k \\ \frac{\partial}{\partial \omega_i} L(\omega, \alpha, \beta) = 0 \\ \frac{\partial}{\partial \beta_i} L(\omega, \alpha, \beta) = 0, i = 1, 2, ..., k \end{cases}$$

- second part:$\min_\omega \max_{\alpha,\beta} L(\omega, \alpha, \beta) \leq \min_\omega f(\omega)$

  prove:

  $h_i(\omega) = 0$, and $\alpha_i g_i(\omega) \leq 0$

  $\Rightarrow$ for any α and β, $L(\omega, \alpha, \beta) \leq f(\omega)$

  $\Rightarrow$ for any α and β, $\min_\omega L(\omega, \alpha, \beta) \leq \min_\omega f(\omega)$

  $\Rightarrow \min_\omega \max_{\alpha,\beta} L(\omega, \alpha, \beta) \leq \min_\omega f(\omega)$

- first part:$\max_{\alpha,\beta} \min_\omega L(\omega, \alpha, \beta) \leq \min_\omega \max_{\alpha,\beta} L(\omega, \alpha, \beta)$ prove:

  for any α,β and ω, $\min_\omega L(\omega, \alpha, \beta) \leq L(\omega, \alpha, \beta)$

  $\Rightarrow$ for any ω, $\max_{\alpha,\beta} min_\omega L(\omega, \alpha, \beta) \leq \max_{\alpha,\beta} L(\omega, \alpha, \beta)$

  $\Rightarrow \max_{\alpha,\beta} \min_\omega L(\omega, \alpha, \beta) \leq \min_\omega \max_{\alpha,\beta} L(\omega, \alpha, \beta)$

- when KKT is held, they will be equal to each other obviously.

Neural Network

Neural Network

- Activation Function
- Forward Propagation
- Backward Propagation
- Loss Function

Activation Function

- neuron is a function, mapping from linear combination of multiple vector x to a vector y by some type of activation function f. $y = f(\theta^T x)$
- there are multiple types of activation functions:
  sigmoid function: $f(y) = \frac{1}{1+e^{-y}}$, $f'(y) = f(y)(1 - f(y))$
  tangent function: $f(y) = \frac{e^y - e^{-y}}{e^y + e^{-y}}$, $f'(y) = 1 - (f(y))^2$
  rectified linear unit (ReLU): $f(y) = \max(0, y)$, $f'(y) = 1(x > 0)$
  maxout: $f(y) = \max_{i=0}^{k}(y_i)$, $f'(y) = 1(y_i > 0)$
- multiple neurons connected from layer to layer, is neural network.
  Inside layer, neurons are not connected to each other.
  Between adjacent layers, there are connections between neurons.
  Between nonadjacent layers, there are not connections between neurons.
- input layer doesn't have activation function, every hidden layer has an activation function, and output layer has only loss function.

Forward Propagation

- define

  previous layer $y_{prev} = f_{prev}(\theta_{prev}^T x_{prev})$,

  now layer $y_{now} = f_{now}(\theta_{now}^T x_{now})$,

  next layer $y_{next} = f_{next}(\theta_{next}^T x_{next})$

- forward propagation is calculate from previous layer to now layer, and to next layer. In this way, the result of final layer y_{final} is calculated.

  $x_{now} = y_{prev}$

  $x_{next} = y_{now}$

Backward Propagation

- chain rule of derivative:

$$\delta_{now} = \frac{dy_{final}}{d(\theta_{now}x_{now})}$$
$$= \frac{dy_{final}}{d(\theta_{next}x_{next})}\frac{d(\theta_{next}x_{next})}{d(\theta_{now}x_{now})} = \frac{dy_{final}}{d(\theta_{next}x_{next})}\frac{d(\theta_{next}y_{now})}{d(\theta_{now}x_{now})} = \frac{dy_{final}}{d(\theta_{next}x_{next})}\frac{\theta_{next}dy_{now}}{d(\theta_{now}x_{now})}$$
$$= \frac{dy_{final}}{d(\theta_{next}x_{next})}\frac{\theta_{next}df_{now}(\theta_{now}x_{now})}{d(\theta_{now}x_{now})} = \frac{dy_{final}}{d(\theta_{next}x_{next})}\frac{\theta_{next}f'_{now}d(\theta_{now}x_{now})}{d(\theta_{now}x_{now})}$$
$$= \frac{dy_{final}}{d(\theta_{next}x_{next})}\theta_{next}f'_{now}$$
$$= \delta_{next}\theta_{next}f'_{now}$$

- derivatives:

$$\frac{dy_{final}}{d(\theta_{now})} = \frac{dy_{final}}{d(\theta_{now}x_{now})}\frac{d(\theta_{now}x_{now})}{d\theta_{now}} = \frac{dy_{final}}{d(\theta_{now}x_{now})}x_{now}$$

- by chain rule, we can calculate $\frac{dy_{final}}{d(\theta_{now}x_{now})}$ in backward way. And calculate $\frac{dy_{final}}{d(\theta_{now})}$ in every layer. And use loss function, GD to get optimal θ for every layer.

Loss Function

- Data Loss
  - 0-1 loss
  - log loss
  - hinge loss
  - squared loss
  - exponential loss
- Regulation Loss
  - L_1 norm
  - L_2 norm

Problem Definition

- The loss function is used to evaluate the learning target, by deriving derivative and control iterative processes.

$$\begin{aligned} L &= L_{data} + L_{norm} \\ &= \frac{1}{N}\sum_{i=0}^{N} L(y^{(i)}, h_\theta(x^{(i)})) + \lambda L_R \end{aligned}$$

- L_{data} is data loss, to evaluate the classification.
- $L_{regualtion}$ is regulation loss, to evaluate the complexity of model to avoiding over-fit.
- actually L has uniform form. Let $m_i = y^{(i)}\theta^T x^{(i)}$, $L_i = L(m_i)$.

$$L_{data} = \frac{1}{N}\sum_{i=0}^{N} L(m_i)$$

- $h_{\theta^T}(x^{(i)})$ is called score function, comparing to hypothesis function of ML.

  L is called loss function, comparing to target function of ML.

  Derivative is the same to ML.

Data Loss: 0-1 Loss

- 0-1 loss: if $y^{(i)}$ and $\theta^T x^{(i)}$ have the same sign, $L_i = 0$.

  if they have different sign, $L_i = 1$, totally in the following equation:

$$L_{01} = \begin{cases} 0, if(m \geq 0) \\ 1, if(m < 0) \end{cases} \tag{3}$$

- Derivative is not analytic, so it is seldom used.

Data Loss: Log Loss

- log loss $L_i = y^{(i)} \log h_\theta(x) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$
  $= y^{(i)} \log \frac{1}{1+e^{-\theta^T x}} + (1 - y^{(i)}) \log(1 - \frac{1}{1+e^{-\theta^T x}})$
  $= y^{(i)} \log \frac{1}{1+e^{-\theta^T x}} + (1 - y^{(i)}) \log(\frac{1}{1+e^{\theta^T x}})$
  $= \log \frac{1}{1+e^{-\widetilde{y^{(i)}}\theta^T x}}$ (when $y^{(i)} = 0, \widetilde{y^{(i)}} = -1$; $y^{(i)} = 1, \widetilde{y^{(i)}} = 1$)
  $= \log \frac{1}{1+e^{-m_i}}$
  softmax and logistic have the same expression.
- cross entropy to data loss: $H(p, q) = -\sum_x p(x) \log q(x)$.
  softmax log loss function: $L_{data} = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{k} 1\{y_j^{(i)} = 1\} \log(\frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}})$, with $p(x) = 1\{y_j = 1\}$
- derivative
  $= -\frac{1}{m} \sum_{i=1}^{m} (1\{y_j^{(i)} = 1\} - \log(p(y^{(i)} = j | x^{(i)}, \theta)))x^{(i)}$

Data Loss: Hinge Loss

- binary classification: $L_i = max(0, 1 - y^{(i)}\theta^T x^{(i)}) = max(0, 1 - m_i)$
  loss function: $L = L_{data} + \lambda L_{norm} = \frac{1}{m}\sum_{i=1}^{m} max(0, 1 - y^{(i)}\theta^T x^{(i)}) + \lambda L_{norm}$
- multiple classification: $L_i = \sum_{j \neq y^{(i)}}^{k} max(0, \theta_j^T x^{(i)} - \theta_{y^{(i)}}^T x^{(i)} + \Delta)$
  or $L_i = max(0, max_{j \neq y^{(i)}}^{k}(\theta_j^T x^{(i)}) - \theta_{y^{(i)}}^T x^{(i)} + 1)$
- derivative

$$\nabla_{\theta_j} L = \begin{cases} 1(\theta_j^T x^{(i)} - \theta_{y^{(i)}}^T x^{(i)} + \Delta > 0) x^{(i)}, if(j \neq y^{(i)}) \\ -(\sum_{j \neq y^{(i)}}^{k} 1(\theta_j^T x^{(i)} - \theta_{y^{(i)}}^T x^{(i)} + \Delta > 0)) x^{(i)}, if(j = y^{(i)}) \end{cases} \tag{4}$$

$$\nabla_{\theta_j} L = \begin{cases} 1(\theta_j^T x^{(i)} - \theta_{y^{(i)}}^T x^{(i)} + 1 > 0) x^{(i)}, if(j \neq y^{(i)} and \theta_j^T x^{(i)} ismax) \\ 0, if(j \neq y^{(i)} and \theta_j^T x^{(i)} isnotmax) \\ -(\sum_{j \neq y^{(i)}}^{k} 1(\theta_j^T x^{(i)} - \theta_{y^{(i)}}^T x^{(i)} + 1 > 0)) x^{(i)}, if(j = y^{(i)}) \end{cases} \tag{5}$$

Data Loss: Squared Loss and Exponential Loss

- squared loss is for linear regression, $L_i = (y^{(i)} - \theta^T x^{(i)})^2$ derivative: $\nabla_{\theta_j} L = x^{(i)}$
- exponential loss for boosting algorithm, $L_i = exp(-y^{(i)}\theta^T x^{(i)}) = exp(-m_i)$ derivative:
- all loss function is decreased by increasing x, except squared loss and 0-1 loss.

Regulation Loss

- $L_{norm} = L_p = (\sum_i |\theta_i|^p)^{\frac{1}{p}}$
- $L_1 = \sum_i |\theta_i|$
- $L_2 = (\sum_i |\theta_i|^2)^{\frac{1}{2}}$, which is the most widely used.
- L_{norm} should be added into the calculation of derivatives.

Deep Learning for NLP

Deep Learning for NLP

- Word Vector
  - One-Hot Representation
  - Distributed Representation
    - Skip-Gram
    - Continuous Bag of Words Model (CBOW)
    - Negative Sampling
- Language Model
  - Statistical Language Model
  - Neural Network Language Model
    - Bag-of-Words Model: Neural Network
    - Sequential Models: Recurrent Neural Network (RNN)
    - Long Short Term Memory (LSTM)
    - Tree Structured-Models: Recursive Neural Network (RNN)

Skip-Gram

- One-Hot Representation $v = [0,0,0,...,1,...0,0,0]$, $v \in R^{|V|}$
- Hypothesis Function

  In the context of window size $2m$ of any sentence, for the center word v_{in} and any context word v_{out}, the hypothesis function is defined as:

  $$p(v_{out}|v_{in}) = softmax(v_{out}^T v_{in}) = \frac{\exp(v_{out}^T v_{in})}{\sum_{i=1}^{|V|} \exp(v_i^T v_{in})}$$

- Loss Function: Maximum likelihood (cross entropy) in the window:

  $L(v_{in}, v_{out}) = -\log \prod_{i=in-m, i\neq in}^{in+m} p(v_i|v_{in}) = -\log \prod_{i=in-m, i\neq in}^{in+m} \frac{\exp(v_{out}^T v_{in})}{\sum_{i=1}^{|V|} \exp(v_i^T v_{in})}$

- Gradient:for simplicity, ignore $\prod$

  $L(v_{in}, v_{out}) = -\log \frac{\exp(v_{out}^T v_{in})}{\sum_{i=1}^{|V|} \exp(v_i^T v_{in})} = -\log \exp(v_{out}^T v_{in}) + \log \sum_{i=1}^{|V|} \exp(v_i^T v_{in})$

  $\frac{\partial L}{\partial v_{in}} = -v_{out} + \sum_{i=1}^{|V|} (p(v_i|v_{in}) v_i^T)$

  $\frac{\partial L}{\partial v_{out}} = v_{in}^T(-1 + p(v_{out}|v_{in})), \qquad \frac{\partial L}{\partial v_i} = v_{in}^T(p(v_i|v_{in}))$

- $\sum_{i=1}^{|V|} \exp(v_i^T v_{in})$ in order of $O(|V|)$, optimized by Hierarchical Softmax.

CBOW

- Hypothesis Function

  In the context of window size $2m$ of any sentence, for the center word v_{out} and any context word v_k, the hypothesis function is defined as:

  $$v_{in} = \frac{1}{2m} \sum_{k=-m, k\neq 0}^{k=m} v_k$$
  $$p(v_{out}|v_{in}) = softmax(v_{out}^T v_{in}) = \frac{\exp(v_{out}^T v_{in})}{\sum_{i=1}^{|V|} \exp(v_i^T v_{in})}$$

- Loss Function: Maximum likelihood (cross entropy):

  $$L(v_{in}, v_{out}) = -\log p(v_{out}|v_{in}) = -\log \frac{\exp(v_{out}^T v_{in})}{\sum_{i=1}^{|V|} \exp(v_i^T v_{in})}$$

- Gradient:

  $$L(v_{in}, v_{out}) = -\log \frac{\exp(v_{out}^T v_{in})}{\sum_{i=1}^{|V|} \exp(v_i^T v_{in})} = -\log \exp(v_{out}^T v_{in}) + \log \sum_{i=1}^{|V|} \exp(v_i^T v_{in})$$
  $$\frac{\partial L}{\partial v_{in}} = (-1 + \sum_{i=1}^{|V|} p(v_{out}|v_{in}))v_{out}^T \qquad \frac{\partial L}{\partial v_{out}} = v_{in}^T(-1 + p(v_{out}|v_{in})),$$
  $$\frac{\partial L}{\partial v_i} = v_{in}^T(p(v_{out}|v_{in})) \qquad \frac{\partial L}{\partial v_k} = \frac{\partial L}{\partial v_i}$$

- $\sum_{i=1}^{|V|} \exp(v_i^T v_{in})$ in order of $O(|V|)$, optimized by Hierarchical Softmax.

Negative Sampling

- Hypothesis Function

  In the context of window size $2m$ of any sentence, for the center word v_{in} and any context word v_{out}, and randomly select n negative v_{neg}.

  v_{in} and v_{out} should in the context, while v_{in} and v_{neg} should not in the context. the hypothesis function is defined as:

  $p(true|v_{out}^T v_{in}) = sigmoid(v_{out}^T v_{in}) = \frac{1}{1+\exp(-v_{out}^T v_{in})}$

  $p(false|v_{neg}^T v_{in}) = 1 - sigmoid(v_{neg}^T v_{in}) = 1 - \frac{1}{1+\exp(-v_{neg}^T v_{in})} = sigmoid(-v_{neg}^T v_{in})$

- Loss Function: Maximum likelihood (cross entropy):

  $L(v_{in}, v_{out}) = -\log \prod p(true|v_{out}^T v_{in}) \prod p(false|v_{neg}^T v_{in}) = -\sum \log sigmoid(v_{out}^T v_{in}) - \sum \log sigmoid(-v_{neg}^T v_{in}) =$

  $-\sum \log \sigma(v_{out}^T v_{in}) - \sum \log \sigma(-v_{neg}^T v_{in})$

- Gradient:

  $\frac{\partial L}{\partial v_{in}} = -\sum \sigma'(v_{out}^T v_{in}) v_{out}^T - \sum \sigma'(v_{out}^T v_{in}) v_{neg}^T = \sum(\sigma(v_{out}^T v_{in}) - 1) v_{out}^T - \sum(\sigma(-v_{out}^T v_{in}) - 1) v_{neg}^T$

  $\frac{\partial L}{\partial v_{out}} = v_{in}^T(\sigma(v_{out}^T v_{in}) - 1)$

  $\frac{\partial L}{\partial v_{neg}} = -v_{in}^T(\sigma(-v_{neg}^T v_{in}) - 1)$

Language Model

- Statistical Model

  $p(y_m = x_1, x_2, ..., x_m) = \prod_{i=1}^{i=m} p(x_i | x_1, ..., x_{i-1})$

  if context size is approximately limited into n

  $p(y_m = x_1, x_2, ..., x_m) \approx \prod_{i=1}^{i=m} p(x_i | x_{i-n+1}, ..., x_{i-1})$

  x_m is word frequency, $\widetilde{y}_m$ is sub sentence frequency.

- Neural Network Language Model: Bag-of-Words Model: Neural Network

  $a_h = \sum_{i=m-n}^{m+n} \omega_{hi} x_i$

  $b_h = f(a_h)$

  $p(y_m | context_n) = h(\omega_{oh} b_h)$

  x_i is a word vector, $\widetilde{y}_m$ is a kind of semantic label such as NER.

  h is chosen by the aim of problem, such as *softmax* or *sigmoid*.

  forward pass and backward pass are general process, ignored here.

- Recurrent Neural Network Model

  $p(y^{t_m} | x^1, x^2, ..., x^{t_m}) = f(x^1, x^2, ..., x^{t_m})$

  x^{t_m} is word vector, y^{t_m} is $x^{t_{m+1}}$.

Recurrent Neural Network (RNN)

- Structure: one input layer (x_i,ω_{ih}) of I dimension, one hidden layer (ω_{hh},f) of H units, and one output layer (ω_{kh},h) of K dimension. Key feature is hidden layer connected to itself.
- forward pass:

  in hidden layer, $a_h^t = \sum_{i=1}^{I} \omega_{ih} x_i + \sum_{h'=1}^{H} \omega_{h'h} b_{h'}^{t-1}$

  $$b_h^t = f(a_h^t)$$

  in output layer, $a_k^t = \sum_{h=1}^{H} \omega_{hk} b_h^t$

  $$b_k^t = o(a_k^t)$$

- backward pass:

  from output to hidden layer,

  $$\delta_h^t = \frac{\partial L}{\partial a_k^t} \frac{\partial a_k^t}{\partial b_h^t} \frac{\partial b_h^t}{\partial a_h^t} = \delta_k^t \sum_{h=1}^{H} \omega_{hk} f'(a_h^t)$$

  from hidden to hidden layer,

  $$\delta_h^{t-1} = \frac{\partial L}{\partial a_h^t} \frac{\partial a_h^t}{\partial b_h^{t-1}} \frac{\partial b_h^{t-1}}{\partial a_h^{t-1}} = \delta_h^t \sum_{h'=1}^{H} \omega_{h'h} f'(a_h^t)$$

  from hidden to input layer,

  $$\delta_i^t = \frac{\partial L}{\partial a_h^t} \frac{\partial a_h^t}{\partial x_i^t} = \delta_h^t \sum_{i=1}^{I} \omega_{ih}$$

Gradient Vanishing and Exploding

- gradient scale between qt time stamp:
  $\frac{\partial \delta_h^{t-1}}{\partial \delta_h^t} = diag(f'(a_h^t)) \sum_{h'=1}^{H} \omega_{h'h} = diag(f'(a_h^t)) W_{h'h}$
  $\frac{\partial \delta_h^{t-q}}{\partial \delta_h^t} = (\frac{\partial \delta_h^{t-1}}{\partial \delta_h^t})^q = (diag(f'(a_h^t)) \sum_{h'=1}^{H} \omega_{h'h})^q = (diag(f'(a_h^t))(W_{h'h}))^q$
  $||W_{h'h}|| \leq$ special radius, is unbounded.
  $||diag(f'(a_h^t))|| \leq$ special radius $= |\max f'(a_h^t)|$ is bounded for sigmoid and tanh.

  If their product is less or bigger than 1, the gradient can't backward pass through long time stamp.

- gradient exploding solution
  if $||f'(a_h^t)||\,||W_{h'h}|| \geq 1$, $\frac{\partial \delta_h^{t-q}}{\partial \delta_h^t}$ will increase rapidly by the power of q.
  solution:gradient clipping

  if $|gradient| \geq threshold$, $gradient = \frac{threshold}{|gradient|} gradient$

Long Short Term Memory (LSTM)

- in hidden layer, to avoid gradient vanishing add memory cell c. to avoid confilicating of keep and discard input and output information add input gate ι, output gate ω. to avoid unlimited increasing of c, add forget gate ϕ. to increase model accuracy, add peepholes to new added gates.
- hidden layer input $a_h^t = \sum_{i=1}^{I} \omega_{ih} x_i + \sum_{h'=1}^{H} \omega_{h'h} b_h^{t-1}$ (the same to RNN)
- input gate
  $a_\iota^t = \sum_{i=1}^{I} \omega_{i\iota} x_i^t + \sum_{h=1}^{H} \omega_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} \omega_{c\iota} s_c^{t-1}$
  $b_\iota^t = f(a_\iota^t)$
- forget gate
  $a_\phi^t = \sum_{i=1}^{I} \omega_{i\phi} x_i^t + \sum_{h=1}^{H} \omega_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} \omega_{c\iota} s_c^{t-1}$
  $b_\phi^t = f(a_\phi^t)$
- memory cell $s_c^t = b_\phi^t s^{t-1} + b_\iota^t g(a_h^t)$
- output gate (almost the same to input gate except peephole)
  $a_\omega^t = \sum_{i=1}^{I} \omega_{i\omega} x_i^t + \sum_{h=1}^{H} \omega_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} \omega_{c\iota} s_c^t$
  $b_\omega^t = f(a_\omega^t)$
- hidden layer output $b_h^t = b_\omega^t h(s_c^t)$
- in output layer, $a_k^t = \sum_{h=1}^{H} \omega_{hk} b_h^t$, $b_k^t = o(a_k^t)$ (the same to RNN)

Gradient in LSTM

- output gate: $\delta_\omega^t = \frac{\partial L}{\partial a_\omega^t} = \frac{\partial L}{\partial a_k^t}\frac{\partial a_k^t}{\partial b_h^t}\frac{\partial b_h^t}{\partial b_\omega^t}\frac{\partial b_\omega^t}{\partial a_\omega^t} = \delta_k^t \sum_{h=1}^H \omega_{hk} h(s_c^t) f'(a_\omega^t)$
- memory cell: $\epsilon_c^t = \frac{\partial L}{\partial s_c^t} = \frac{\partial L}{\partial a_k^t}\frac{\partial a_k^t}{\partial b_h^t}\frac{\partial b_h^t}{\partial s_c^t} = \delta_k^t \sum_{h=1}^H \omega_{hk}(b_\omega^t h'(s_c^t) + h(s_c^t) f'(a_\omega^t) \sum_{c=1}^C \omega_{c\iota})$
- input gate: $\delta_\iota^t = \frac{\partial L}{\partial s_c^t}\frac{\partial s_c^t}{\partial b_\iota^t}\frac{\partial b_\iota^t}{\partial a_\iota^t} = \epsilon_c^t g(a_h^t) f'(a_\iota^t)$
- forget gate: $\delta_\phi^t = \frac{\partial L}{\partial s_c^t}\frac{\partial s_c^t}{\partial b_\phi^t}\frac{\partial b_\phi^t}{\partial a_\phi^t} = \epsilon_c^t s_c^{t-1} f'(a_\phi^t)$
- hidden layer input: $\delta_h^t = \frac{\partial L}{\partial s_c^t}\frac{\partial s_c^t}{\partial a_h^t} = \epsilon_c^t b_\iota^t g'(a_h^t)$
- from memory cell to memory cell:
  $\frac{\partial s_c^t}{\partial s_c^{t-1}} = b_\phi^t + s_c^{t-1}\frac{\partial b_\phi^t}{\partial s_c^{t-1}} + g(a_h^t)\frac{\partial b_\iota^t}{\partial s_c^{t-1}} + b_\iota^t g'(a_h^t)\frac{\partial a_h^t}{\partial s_c^{t-1}}$
- truncated backward pass to remove cycle between memory cells:
  $\frac{\partial a_h^t}{\partial b_h^{t-1}} \approx 0,\ \frac{\partial a_\iota^t}{\partial b_h^{t-1}} \approx 0,\ \frac{\partial a_\phi^t}{\partial b_h^{t-1}} \approx 0,\ \frac{\partial a_\omega^t}{\partial b_h^{t-1}} \approx 0$
  i.e. $\frac{\partial a_h^t}{\partial s_c^{t-1}} = \frac{\partial a_h^t}{\partial b_h^{t-1}}\frac{\partial b_h^{t-1}}{\partial s_c^{t-1}} \approx 0,\ \frac{\partial a_\iota^t}{\partial s_c^{t-1}} \approx 0,\ \frac{\partial a_\phi^t}{\partial s_c^{t-1}} \approx 0,\ \frac{\partial a_\omega^t}{\partial s_c^{t-1}} \approx 0$
  i.e. $\frac{\partial s_c^t}{\partial s_c^{t-1}} \approx b_\phi^t$
- There are only information flow to previous t in cells, and decided by b_ϕ^t. b_ϕ^t is decided by a factor of x_i and t. Although it is bounded ≤ 1, information will not vanish as fast as RNN.

Recursive Neural Network (RNN)

- TBD

Deep Learning for Image Processing

Deep Learning for Image Processing: CNN

- Convolutional Layer
- Pooling Layer
  - mean-pooling
  - max-pooling
  - stochastic-pooling
- (Activation Layer)
- Full-connected Layer
- Network Gallery
  - LeNet
  - AlexNet
  - GoogleLeNet
  - VGG
  - ResNet
- Feature Extractor & Fine-tuning

Convolutional Layer

- structure

  Input is M features f_m^{l-1} in previous layer, and out put is N features f_n^l in current layer. Convolutional operator between each pair of f_m^{l-1} and f_n^l is a square matrix $\Theta = [\theta_{i,j}]$ filter and a bias b. One f_m^{l-1} may be corresponding to multiple f_n^l and one f_n^l may be corresponding to multiple f_m^{l-1}. Θ and b is trainable by $\frac{\partial}{\partial b^l} = \delta_{f^l}$, $\frac{\partial}{\partial \Theta^l} = \delta_{f^l} f^{l-1}$.

- forward

  Every sub matrix s of feature k in f_k^{l-1} of size $|\Theta|$ is mapped to one element t in f_l feature:

  $t = \sum_k Conv(f_k^{l-1}, \Theta) = \sum_k \sum_{i,j} (S_{i,j}^k \theta_{i,j}^k)$, where matrix coloumn and row are reduced by $|\Theta|^{0.5} - 1$, and k is mapping k layer $l-1$ to layer l

- backward

  1st: padding matrix δ_t with 0 by $2(|\Theta|^{0.5} - 1)$

  2nd: rotate Θ by 180 degree to get Θ^{rotate}

  3rd: calculate $Conv$ as forward step.

  In summary, $\delta_s = \sum_p DeConv(\delta_t, \Theta) = \sum_p Conv(\delta_t^{padding}, \Theta^{rotate})$

  where p is mapping layers from $l-1$ to p layers in l.

Convolutional Operation Sample

- forward $c = Conv(a, b)$

$$a = \begin{Bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{Bmatrix} \; b = \begin{Bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{Bmatrix} \rightarrow c = \begin{Bmatrix} c_1 = a_1 b_1 + a_2 b_2 + a_4 b_3 + a_5 b_4 & c_2 = a_2 b_1 + a_3 b_2 + a_5 b_3 + a_6 b_4 \\ c_3 = a_4 b_1 + a_5 b_2 + a_7 b_3 + a_8 b_4 & c_4 = a_5 b_1 + a_6 b_2 + a_8 b_3 + a_9 b_4 \end{Bmatrix}$$

- backward $\delta_a = DeConv(\delta_c, b)$ deduced by $\delta_a = \frac{dy}{da} = \frac{dy}{dc}\frac{dc}{da} = \delta_c \frac{dc}{da}$

$$c = \begin{Bmatrix} 0 & 0 & 0 & 0 \\ 0 & \delta_{c_1} & \delta_{c_2} & 0 \\ 0 & \delta_{c_3} & \delta_{c_4} & 0 \\ 0 & 0 & 0 & 0 \end{Bmatrix} \; b = \begin{Bmatrix} b_4 & b_3 \\ b_2 & b_1 \end{Bmatrix} \rightarrow a = \begin{Bmatrix} \delta_{a_1} \delta_{a_2} \delta_{a_3} \\ \delta_{a_4} \delta_{a_5} \delta_{a_6} \\ \delta_{a_7} \delta_{a_8} \delta_{a_9} \end{Bmatrix}$$

$$\begin{array}{lll} \delta_{a_1} = \delta_{c_1} b_1 & \delta_{a_2} = \delta_{c_1} b_2 + \delta_{c_2} b_1 & \delta_{a_3} = \delta_{c_2} b_2 \\ \delta_{a_4} = \delta_{c_1} b_3 + \delta_{c_3} b_1 & \delta_{a_5} = \delta_{c_1} b_4 + \delta_{c_2} b_3 + \delta_{c_4} b_1 & \delta_{a_6} = \delta_{c_2} b_4 + \delta_{c_4} b_2 \\ \delta_{a_7} = \delta_{c_3} b_3 & \delta_{a_8} = \delta_{c_3} b_4 + \delta_{c_4} b_3 & \delta_{a_9} = \delta_{c_4} b_4 \end{array}$$

Pooling Layer

- structure

  Generally, pooling operator is a square matrix of size $2 * 2$, but is not absolute. If there is one feature mapping between layer $l-1$ and layer l, there is a pooling operator. It is not trainable generally, but sometimes there is trained weighted inside pooling operator.

- forward: Every sub matrix s of a feature f_{l-1} is mapped to one element t in f_l feature:

  Max Pooling:$t = Pooling_{max}(s) = max_i(s_i)$

  Average Pooling:$t = Pooling_{average}(s) = \frac{\sum_i(s_i)}{4}$

  Stochastic Pooling:$t = Pooling_{stochastic}(s) = multinomial\frac{s_i}{\sum_i(s_i)}$

- backward: Padding one element in δ_l to $2 * 2$ element in δ_{l-1}.

  Max Pooling: filling the element which is max in forward with δ_l, and three others with 0.

  Average Pooling: filling all 4 elements of δ_{l-1} with $\delta_l/4$.

  Stochastic Pooling: filling the chosen element in foward of δ_{l-1} with δ_l, and three others with 0.

Pool Operation Sample

- forward $c = Pooling_{max}(a)$

$$a = \begin{Bmatrix} a_1 & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 \\ a_9 & a_{10} & a_{11} & a_{12} \\ a_{13} & a_{14} & a_{15} & a_{16} \end{Bmatrix} \rightarrow c = \begin{Bmatrix} c_1 = a_2 = max\{a_1, a_2, a_5, a_6\} & c_2 = a_7 = max\{a_3, a_4, a_7, a_8\} \\ c_3 = a_9 = max\{a_9, a_{10}, a_{13}, a_{14}\} & c_4 = a_{16} = max\{a_{11}, a_{12}, a_{15}, a_{16}\} \end{Bmatrix}$$

- backward $\delta_a = DePooling(\delta_c)$ deduced by $\delta_a = \frac{dy}{da} = \frac{dy}{dc}\frac{dc}{da} = \delta_c \frac{dc}{da}$

$$c = \begin{Bmatrix} \delta_{c_1} & \delta_{c_2} \\ \delta_{c_3} & \delta_{c_4} \end{Bmatrix} \rightarrow a = \begin{Bmatrix} 0 & \delta_{c_1} & 0 & 0 \\ 0 & 0 & \delta_{c_2} & 0 \\ \delta_{c_3} & 0 & 0 & 0 \\ 0 & 0 & 0 & \delta_{c_4} \end{Bmatrix}$$

Activation Layer

- structure

  Generally, activation layer l is of the same size as previous layer $l-1$. It maps from pixel s_i^{l-1} to pixel t_i^l by nonlinear function f. In addition, there is trainable parameter Θ and b for input pixel. Its previous layer may be convolutional or pooling layer.

- forward: pixel s_i^{l-1} to pixel t_i^l mapping is formulated as:

  $$t_i^l = f(\Theta s_i^{l-1} + b)$$

- backward: Trivally as:

  $$\delta_i^{l-1} = \delta_i^l f' \Theta$$

LeNet-5

- Architecture

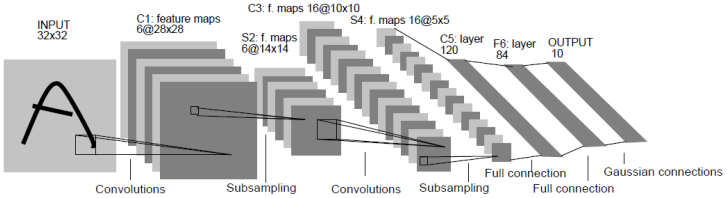


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet-5

- domain: hand-writing number recognition
- structure:

  input: 32*32

  C1: by 6 Conv 5*5, 6*28*28

  S2: by Weighted Mean Pool 2*2, 6*14*14

  C3: by 6 Conv 5*5, 16*10*10

  (0,1,2)->0 (1,2,3)->1 (2,3,4)->2 (3,4,5)->3 (4,5,0)->4 (5,0,1)->5 (0,1,2,3)->6 (1,2,3,4)->7
  (2,3,4,5)->8 (3,4,5,0)->9 (4,5,0,1)->10 (5,0,1,2)->11 (0,1,3,4)->12 (1,2,4,5)->13
  (0,2,3,5)->14 (0,1,2,3,4,5)->15

  S4: by Weighted Mean Pool 2*2, 16*5*5

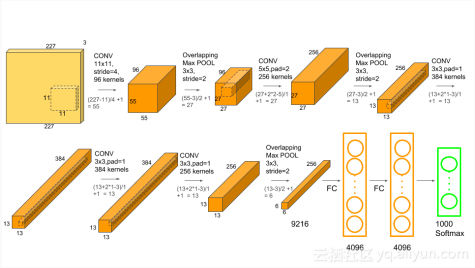
  C5: by 16 Conv 5*5, 120*1*1

  F6: by full connected, 84*1*1

  Output: by full connected, 10*1*1

AlexNet

- Architecture

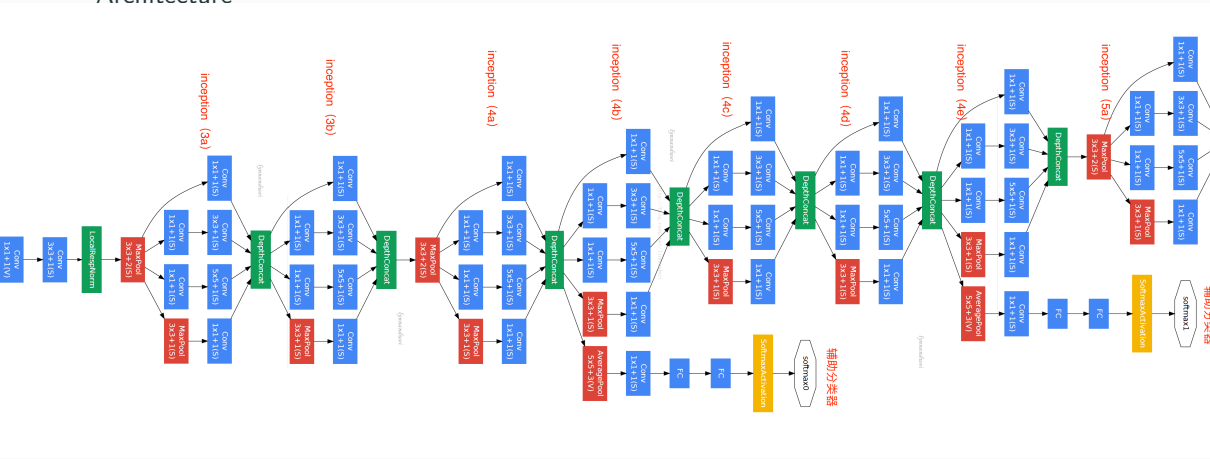


VGG

- Architecture

GoogLeNet

- Architecture



ResNet

- Architecture

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Deep Learning Tricks

Deep Learning Tricks

- Derivative Checking
- Gradient Vanishing and Explosion
- Data Preparation
  - Mean-Subtraction and Normalization
  - PCA and Whitening
  - Batch Normalization
- Weight Initialization
  - Pitfall: All Zero Initialization
  - Small Random Numbers
  - Calibrating the variances with 1/sqrt(n)
  - Sparse initialization
  - Xavier Initialization
- Dropout

Derivative Checking

- Check analytical derivative by numerical version.
  $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$
- why not $\frac{f(x+h)-f(x)}{h}$ but $\frac{f(x+h)-f(x-h)}{2h}$?
  by Tylor expansion at x

$$\begin{aligned}
\frac{f(x+h)-f(x)}{h} &= \frac{1}{h}(f^{(0)}(x) + f^{(1)}(x)h + \frac{1}{2!}f^{(2)}(x)h^2 + \sum_{n=3}^{\infty}\frac{1}{n!}f^{(n)}h^n - f(x)) \\
&= \frac{1}{h}(f^{(1)}(x)h + \sum_{n=2}^{\infty}\frac{1}{n!}f^{(n)}h^n) \\
&= f^{(1)}(x) + \frac{1}{h}\sum_{n=2}^{\infty}\frac{1}{n!}f^{(n)}h^n) \\
\frac{f(x+h)-f(x-h)}{2h} &= \frac{1}{2h}(f^{(0)}(x) + f^{(1)}(x)h + \sum_{n=2}^{\infty}\frac{1}{n!}f^{(n)}h^n \\
&\quad -(f^{(0)}(x) + f^{(1)}(x)(-h) + \sum_{n=2}^{\infty}\frac{1}{n!}f^{(n)}(-h)^n)) \\
&= \frac{1}{2h}(f^{(1)}(x)2h + \sum_{n=2}^{\infty}\frac{2}{(2n-1)!}f^{(2n-1)}2h^{2n-1}))) \\
&= f^{(1)}(x) + \frac{1}{2h}\sum_{n=2}^{\infty}\frac{2}{(2n-1)!}f^{(2n-1)}2h^{2n-1}))) \\
(\frac{f(x+h)-f(x-h)}{2h} - f^{(1)}(x)) &\sim O(h^{2n-1}) \ll (\frac{f(x+h)-f(x)}{h} - f^{(1)}(x)) \sim O(h^n)
\end{aligned}$$

Xavier Initialization

- In order to avoid neurons becoming too correlated and ending up in poor local minimina, it is often helpful to randomly initialize parameters.One of the most frequent initializations used is called Xavier initialization.
- Given a matrix A of dimension $m * n$, select values A_{ij} uniformly from $[\epsilon, \epsilon]$, where

$$\epsilon = \frac{\sqrt{6}}{\sqrt{m+n}}$$

Maximum Entropy

- Maximum Likelihood solution

Table 1: features and labels(x_2 is set to be trivial)

y	x_1	x_2
no	sunny	hot
no	sunny	hot
yes	overcast	hot
yes	rainy	hot
yes	rainy	hot
no	rainy	hot
yes	overcast	hot
no	sunny	hot
yes	sunny	hot
yes	rainy	hot
yes	sunny	hot
yes	overcast	hot
yes	overcast	hot
no	rainy	hot

Table 2: probability by maximum likelihood:y/x_1

	sunny	rainy	overcast
yes	40%	60%	100%
no	60%	40%	0%

- if x_2 is not set to trivial, Maximum Entropy is introduced as follows.

Entropy and Conditional Entropy

- In information theory, Entropy is defined to measure unpredictability of random variable X.

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{6}$$

- Conditional Entropy is defined as random variable of Y determined by another random variable of X.

$$\begin{aligned} H(Y|X) &= \sum_{i=1}^{n} p(x_i) H(Y|X = x_i) \\ &= -\sum_{i=1}^{n} p(x_i) \sum_{j=1}^{m} p(y_j|x_i) \log p(y_j|x_i) \\ &= -\sum_{i=1}^{n} \sum_{j=1}^{m} p(x_i) p(y_j|x_i) \log p(y_j|x_i) \\ &= -\sum_{x,y} p(x) p(y|x) \log p(y|x) \end{aligned} \tag{7}$$

Optimization Objective

- input:label vector $Y = (y_1, y_2)$ and feature vector $X = (x_1, x_2)$
- output:find $p(y_1|x_1, x_2)$ and $p(y_2|x_1, x_2)$ on the objective of

$$max(H(p(Y|X))) = max(-\sum_{x,y} \widetilde{p}(x)p(y|x)\log p(y|x)) \tag{8}$$

$$p(y|x)\widetilde{p}(x)f_i(x, y) = \widetilde{p}(x, y)f_i(x, y), \text{ for } i = 1, ..., k \tag{9}$$

$$\sum_y p(y|x) = 1 \tag{10}$$

- in (6),$\widetilde{p}(x)$ is observed probability of x,$\widetilde{p}(x)p(y|x)$ is $p(y|x)$ model's probability of (x, y). (6) means let entropy of model to be maximum.
- in (7),$\widetilde{p}(x, y)$ is observed probability of (x, y) in labeled corpus. (7) means let model's probability of (x, y) to be the same to observed.
- in (7),$f_i(x, y)$ is feature function, i.e. how much does y is affected by x. If x and y appears at the same time $f_i(x, y) = 1$, else $f_i(x, y) = 0$.
- in (8),$\sum_y p(y|x)$ means if the same x will effected different y, the summary of probability of y must be 1.

Lagrange Dual Problem

Lagrange Dual Problem Optimization

- Lagrange function associated with Maximum Entropy objective function is defined as

$$L(p(y|x), \lambda_i, \lambda_0) = H(p) + \sum_{i=1}^{k}(\lambda_i f_i(x,y)(p(y|x)\widetilde{p}(x) - \widetilde{p}(x,y))) + \lambda_0(\sum_y p(y|x) - 1) \tag{11}$$

- The stationary point of Lagrange function is the same to original Maximum Entropy objective function.

$$\frac{\partial L}{\partial \lambda_i} = 0 \Rightarrow p(y|x)\widetilde{p}(x)f_i(x,y) = \widetilde{p}(x,y)f_i(x,y), \text{ for } i = 1, ..., k \tag{12}$$

$$\frac{\partial L}{\partial \lambda_0} = 0 \Rightarrow \sum_y p(y|x) = 1 \tag{13}$$

$$\frac{\partial L}{\partial p} = 0 \Rightarrow \frac{\partial H(p)}{\partial p} = 0 \tag{14}$$

- from 7 ($H(Y|X) = -\sum_{x,y}\widetilde{p}(x)p(y|x)\log p(y|x)$), 11, 14, we can have

$$\frac{\partial L}{\partial p} = -\widetilde{p}(x)(\log p(y|x) + 1) + \sum_{i=1}^{k}(\lambda_i f_i(x,y)\widetilde{p}(x) + \lambda_0 = 0 \tag{15}$$

$$p^*(y|x) = e^{\sum_{i=1}^{k}\lambda_i f_i(x,y) - \frac{\lambda_0}{\widetilde{p}(x)} - 1} \tag{16}$$

- by 10, we can remove λ_0, in the following way

$$e^{\frac{\lambda_0}{\widetilde{p}(x)}+1} = \sum_y e^{\sum_{i=1}^k \lambda_i f_i(x,y)} \tag{17}$$

$$\begin{cases} p^*(y|x) = \dfrac{e^{\sum_{i=1}^k \lambda_i f_i(x,y)}}{Z(x)} \\ Z(x) = \displaystyle\sum_y e^{\sum_{i=1}^k \lambda_i f_i(x,y)} \end{cases} \tag{18}$$

- by 18, we have simplified $L(p(y|x), \lambda_i, \lambda_0)$ to

$$L(\lambda) = \sum_{x,y} \widetilde{(}p)(x)p^*(y|x) \log Z(x) - \sum_{i=1}^k \widetilde{p}(x,y) \sum_{x,y} \lambda_i f_i(x,y) \tag{19}$$

- 19 is a complex function of λ, by finding the stationary point of this function we can find the possible maximum point of 8. We can define primal problem is:

$$\begin{cases} H(p(Y|X)) = -\displaystyle\sum_{x,y} \widetilde{p}(x)p(y|x) \log p(y|x) \\ \dfrac{\partial H(p)}{\partial p} = 0 \end{cases} \tag{20}$$

Lagrange Dual Problem

- General Lagrange dual problem of 20 is as follows

$$\begin{cases} \dfrac{\partial L}{\partial \lambda} = 0 \\ L(\lambda) = \sum\limits_{x,y} \widetilde{p}(x) p^*(y|x) \log Z(x) - \sum\limits_{i=1}^{k} \widetilde{p}(x,y) \sum\limits_{x,y} \lambda_i f_i(x,y) \\ p^*(y|x) = \dfrac{1}{Z_\lambda(x)} e^{\sum_i \lambda_i f_i(x,y)} \\ Z_\lambda(x) = \sum\limits_{y} e^{\sum_i \lambda_i f_i(x,y)} \end{cases} \tag{21}$$

- In our case, $\widetilde{p}(x) = 1$, and $\widetilde{p}(x, y_1) = 0.4$, and $\widetilde{p}(x, y_2) = 0.6$, and λ_1 is for $f(y_1|x)$, and λ_2 is for $f(y_2|x)$

$$\begin{cases} Z_\lambda(x) = e^{\lambda_1} + e^{\lambda_2} \\ p^*(y_{1or2}|x) = \dfrac{e^{\lambda_{1or2}}}{e^{\lambda_1} + e^{\lambda_2}} \\ L(\lambda) = -\log(e^{\lambda_1} + e^{\lambda_2}) + (0.4\lambda_1 + 0.6\lambda_2) \\ H(p(Y|X)) = -(\dfrac{\lambda_1 e^{\lambda_1} + \lambda_2 e^{\lambda_2}}{e^{\lambda_1} + e^{\lambda_2}} - \log(e^{\lambda_1} + e^{\lambda_2})) \end{cases} \tag{22}$$

Conjugate Function

- From the graphical understanding of previous page, we define conjugate function as

$$f^*(y) = \sup_{x \in \mathrm{dom} f(x)}(yx - f(x)) \tag{23}$$

- from the definition of $f^*(y)$, we can know when $y = f'(x)$, supermum is got

$$f^*(y) = yf'(x) - f(f'(x)) \tag{24}$$

- some conjugate of convex functions are as follows

$$\begin{aligned} &f(x) = ax + b \to f^*(y) = ya - ay + b = b \\ &f(x) = -\log x \to f^*(y) = y(-\frac{1}{y}) + \log(-\frac{1}{y}) = -\log(-y) - 1 \\ &f(x) = x\log x \to f^*(y) = ye^{y-1} - e^{y-1}\log e^y - 1 = e^{y-1} \end{aligned} \tag{25}$$

- use 25 in 11, we can get Lagrange dual at once

$$L(\lambda) = b\sum_{i=1}^{k}\lambda_i - \log(\sum_{i=1}^{k} e^{\lambda_i}) \tag{26}$$

- it matches the result in 22

Optimization Algorithm

Optimization Algorithm

- There are multiple unconstrained optimization algorithm to resolve Lagrange Dual problem $\frac{\partial L(\lambda)}{\partial \lambda} = 0$
- Gradient Decent Method

$$\begin{cases} \lambda_i^{n+1} = \lambda_i^n + C\Delta\lambda_i \\ \Delta\lambda_i = \sum\limits_{x,y} \widetilde{p}(x,y) f_i(x,y) - \sum\limits_{x} \widetilde{p}(x) \sum\limits_{y} p^*(y|x) f_i(x,y) \end{cases} \tag{27}$$

- GIS (Generative Iterative Scaling)

$$\lambda_i^{n+1} = \lambda_i^n + \frac{1}{C} \log \frac{\sum_{x,y} \widetilde{p}(x,y) f_i(x,y))}{\sum_x \widetilde{p}(x) \sum_y p^*(y|x) f_i(x,y)} \tag{28}$$

- IIS (Improved Iterative Scalling) if $\sum_i f_i(x,y)$ is constant,λ_i is got by 29, else by 30

$$\lambda_i = \frac{1}{\sum_i f_i(x,y)} \log \frac{\sum_{x,y} \widetilde{p}(x,y) f_i(x,y)}{\sum_x \widetilde{p}(x) \sum_y p^*(y|x) f_i(x,y)} \tag{29}$$

$$\begin{cases} \lambda_i^{n+1} = \lambda_i^n + \Delta\lambda_i \\ \sum\limits_{x,y} \widetilde{p}(x,y) f_i(x,y) - \sum\limits_{x} \widetilde{p}(x) \sum\limits_{y} p^*(y|x) f_i(x,y) e^{\Delta\lambda_i \sum_i f_i(x,y)} = 0 \end{cases} \tag{30}$$

$$\alpha = arg \min_{\alpha \geq 0} h(\alpha) = arg \min_{\alpha \geq 0} f(x_k + \alpha d_k) \tag{31}$$

$h(\alpha)$ $h'(\alpha) = \nabla f(x_k + \alpha d_k)^T d_k = 0$ d_k $f(x_k)$ x_k $h'(0) \leq 0$ $\hat{\alpha}$ $h'(\hat{\alpha}) > 0$ $\alpha^\star \in [0, \hat{\alpha})$ $h'(\alpha^\star) = 0$ $\alpha^\star$

Questions?